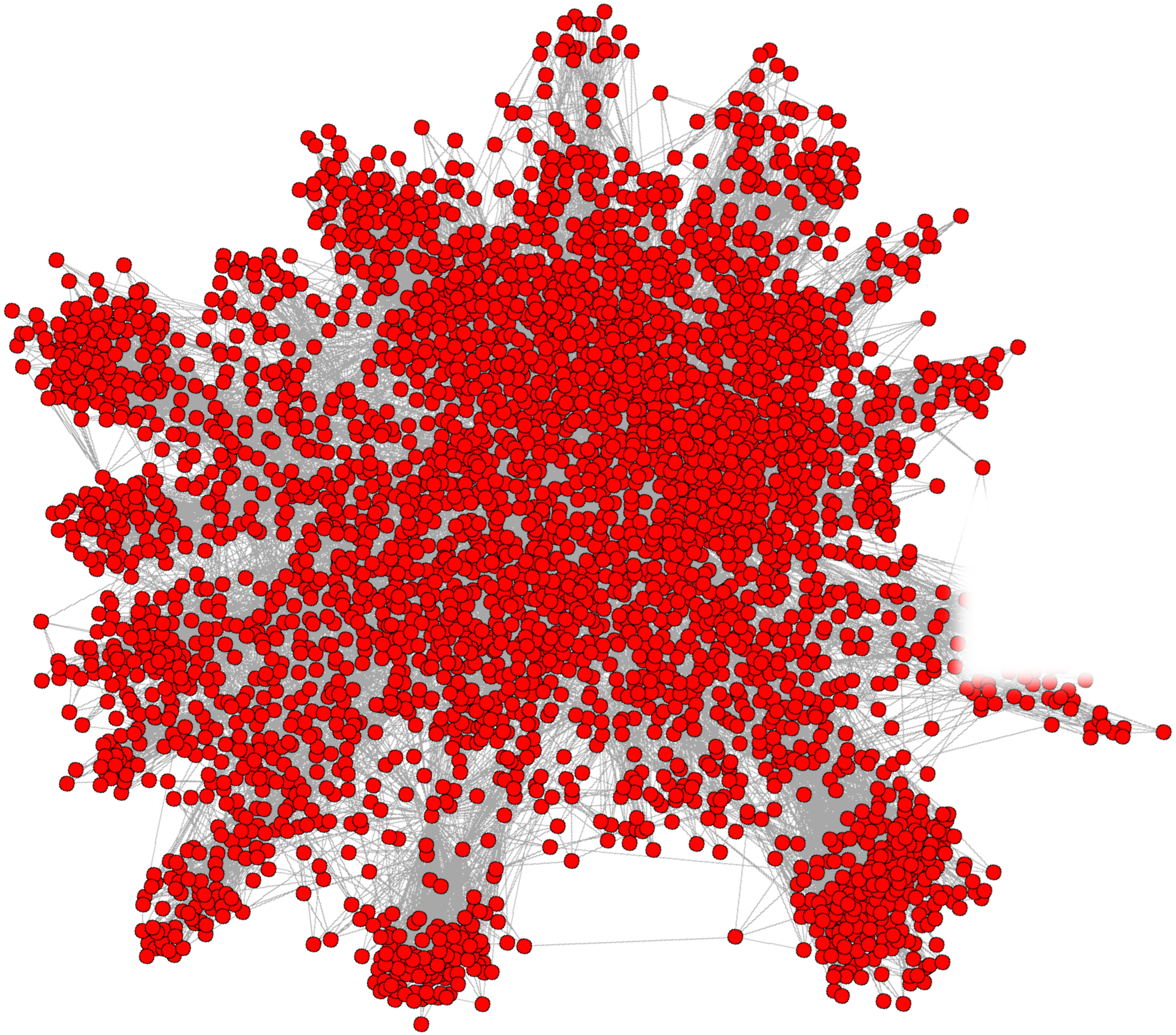
Maciej Besta, Marc Fischer, Tal Ben-Nun, Johannes de Fine Licht, Torsten Hoefler

Substream-Centric Maximum Matchings on FPGA

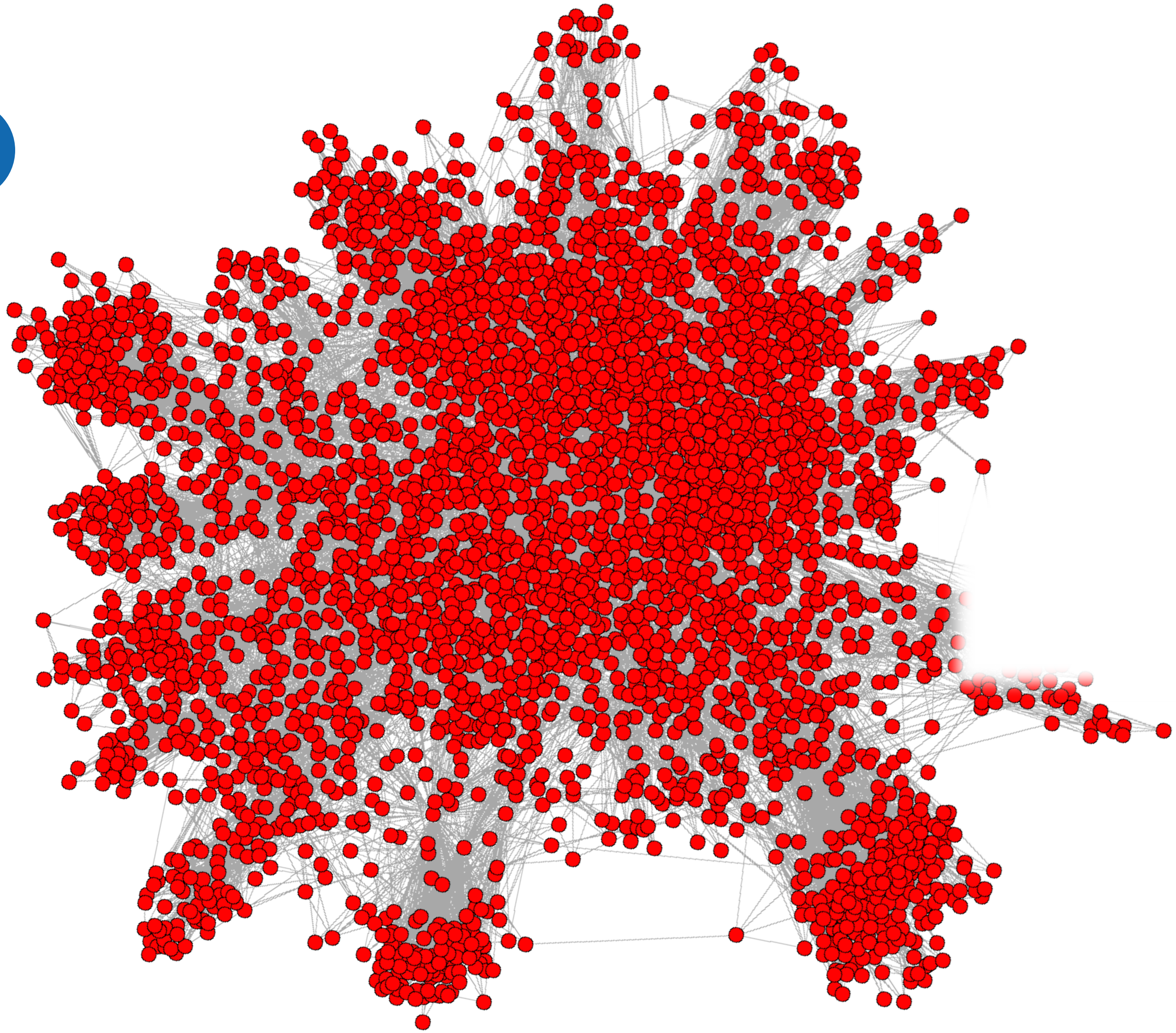
Large graphs…

Large graphs...



Large graphs…

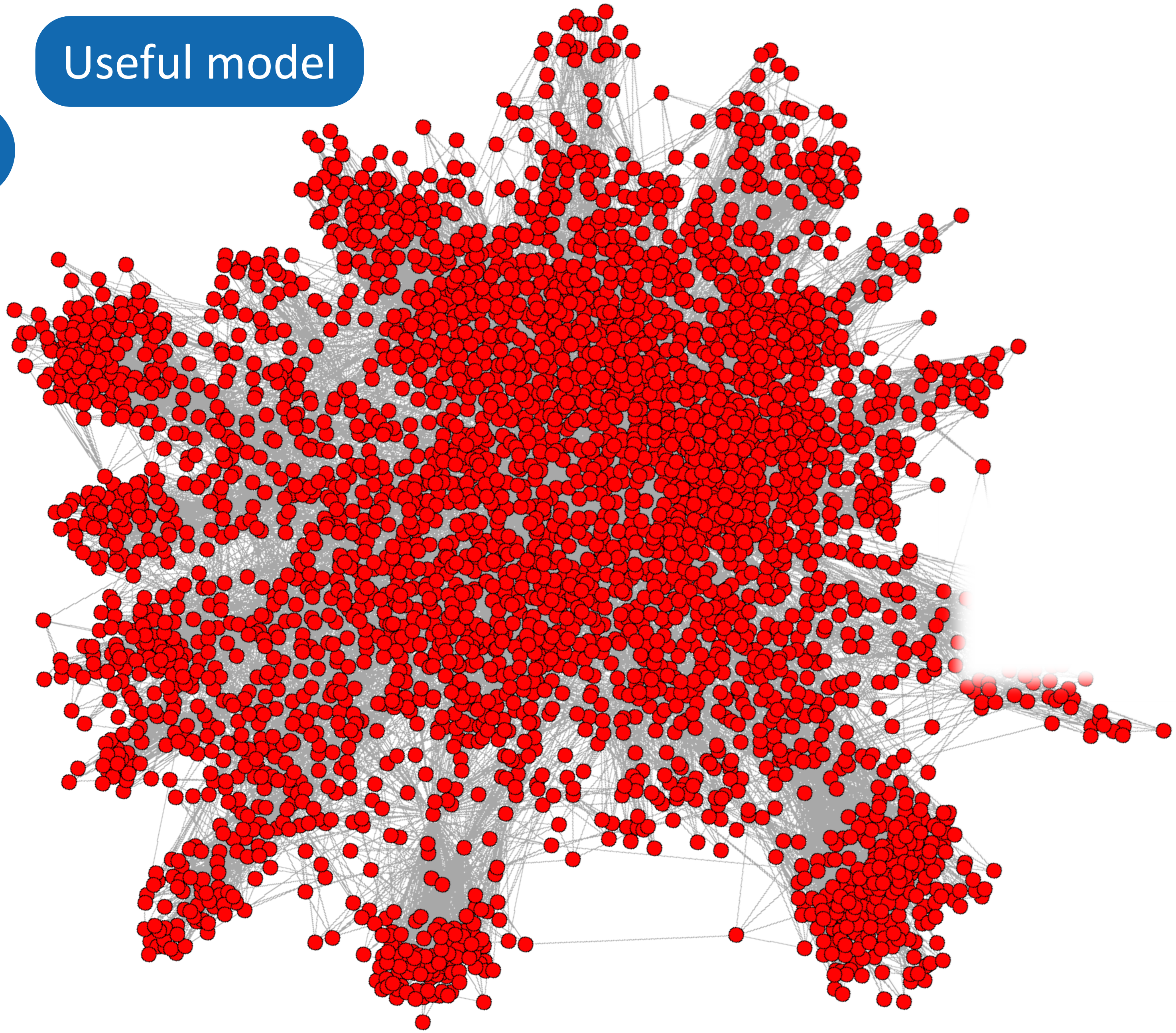
Why do we care?



Large graphs…

Why do we care?

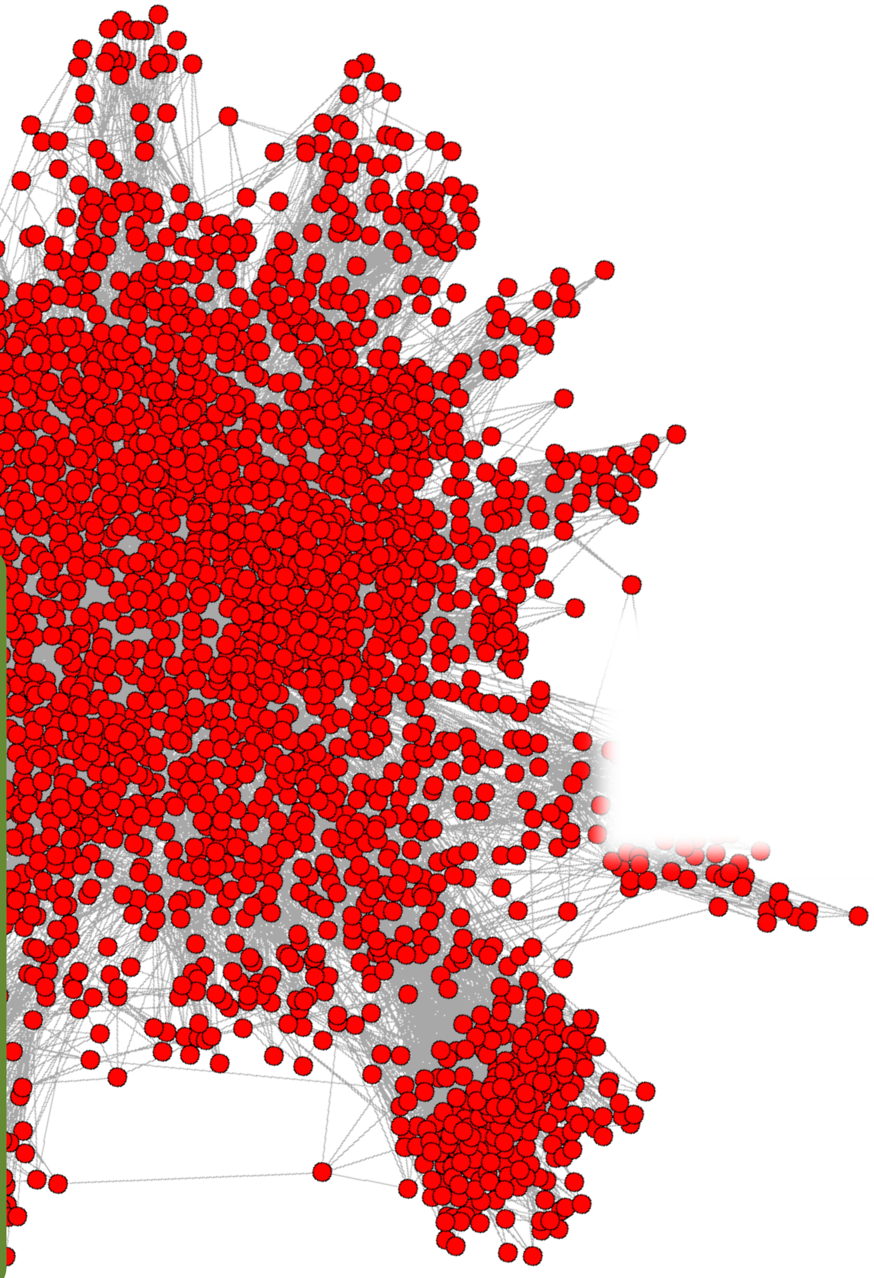
Useful model



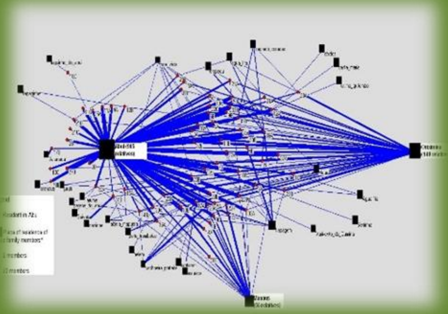
Large graphs...

Useful model

Why do we care?



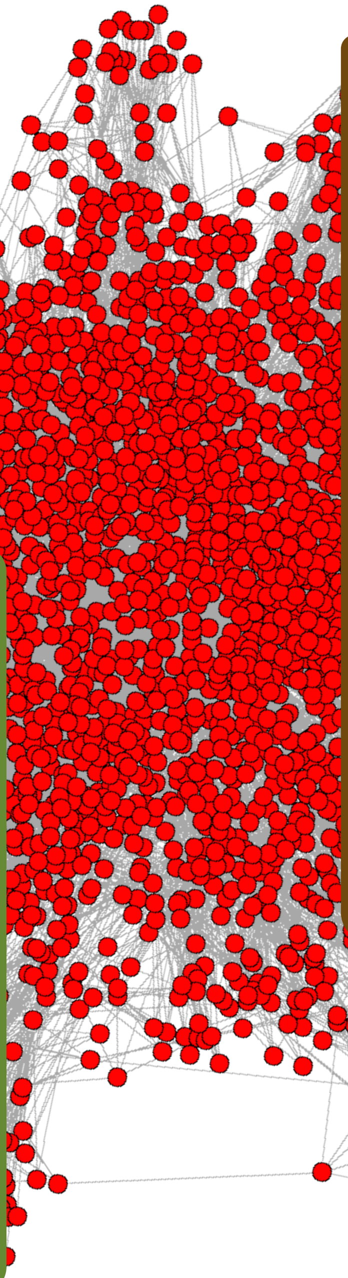
Social networks



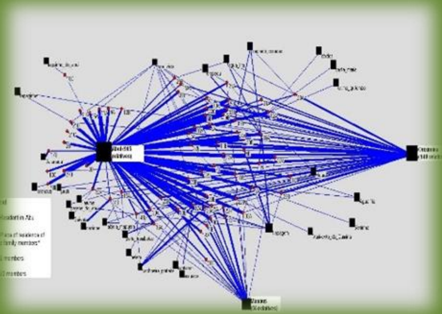
Large graphs…

Useful model

Why do we care?



Social networks



Engineering networks

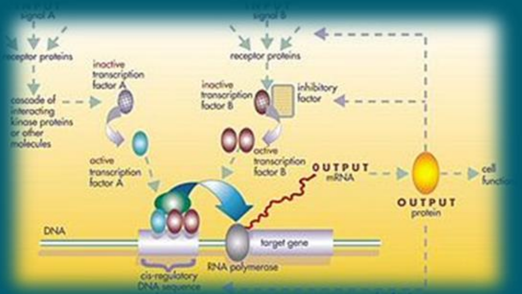
Large graphs…

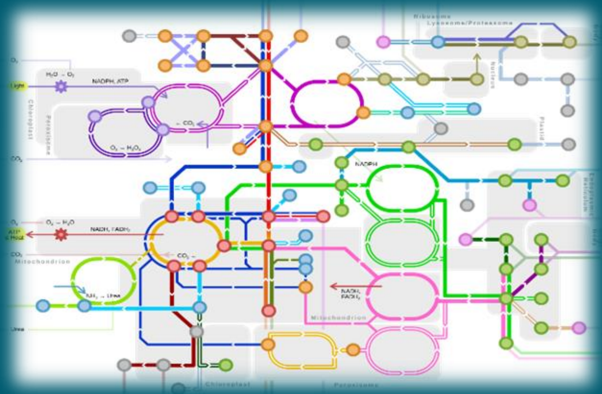
Why do we care?

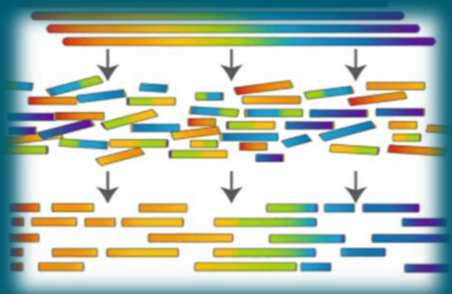
Useful model

Social networks

Biological networks







Engineering networks

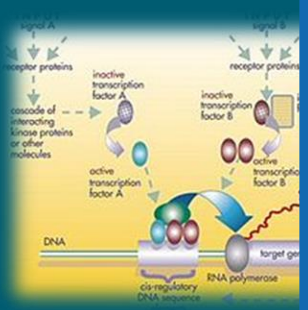
Large graphs…

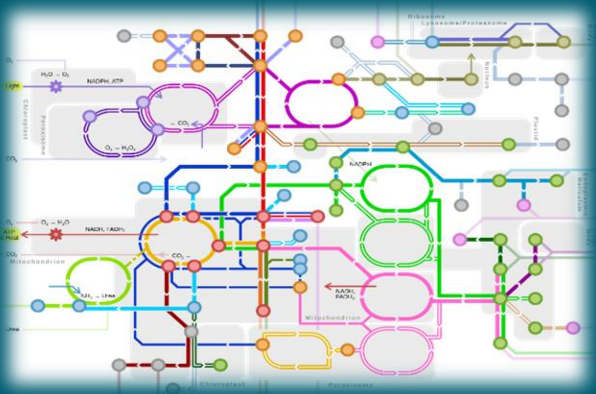
Useful model

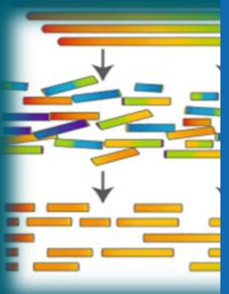
Why do we care?

Social networks

Biological networks

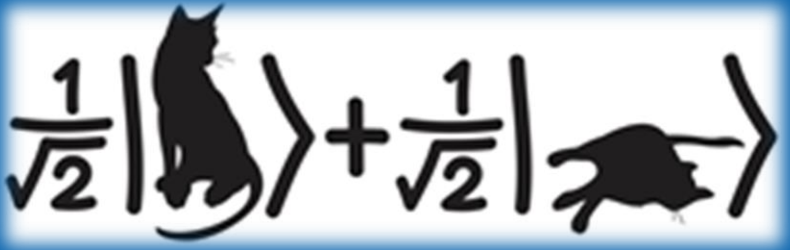


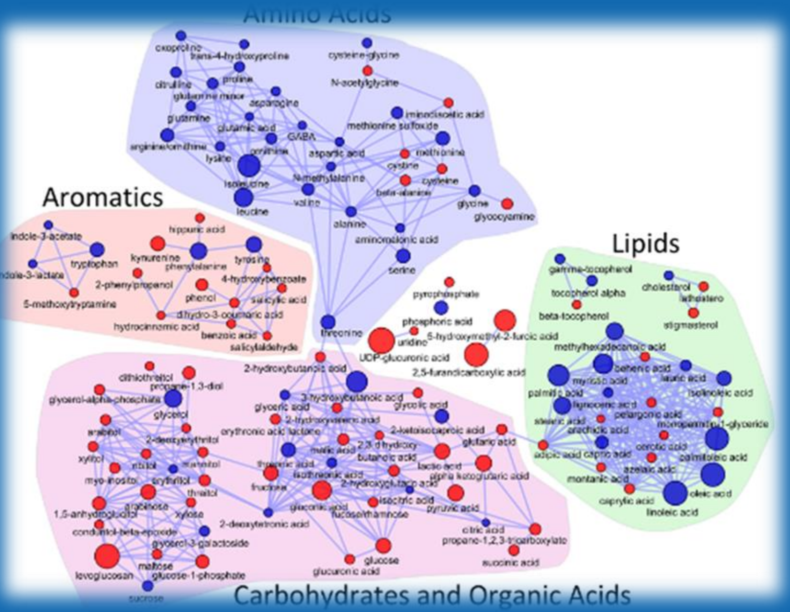




Engineering networks

Physics, chemistry

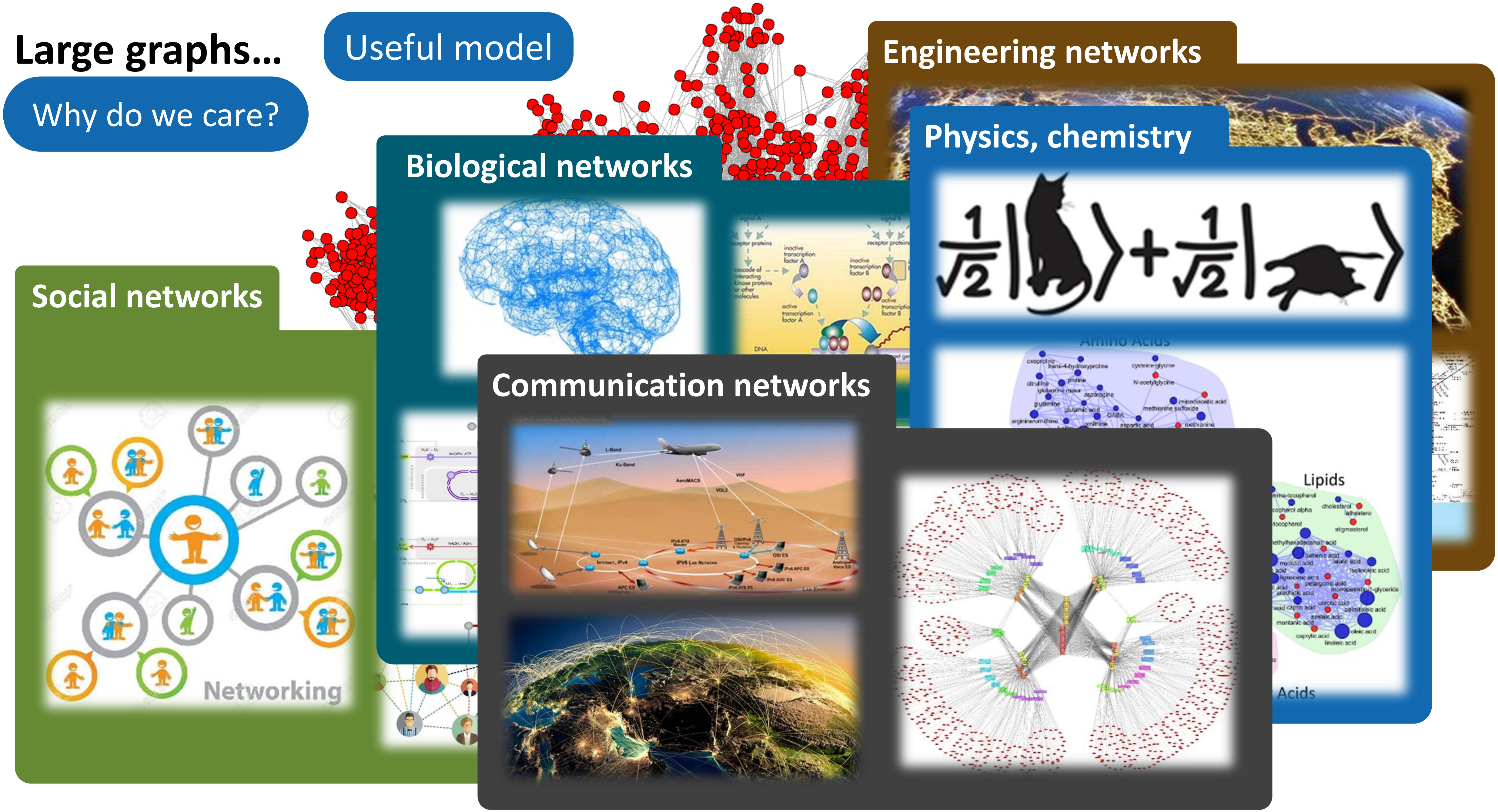




Large graphs…

Why do we care?

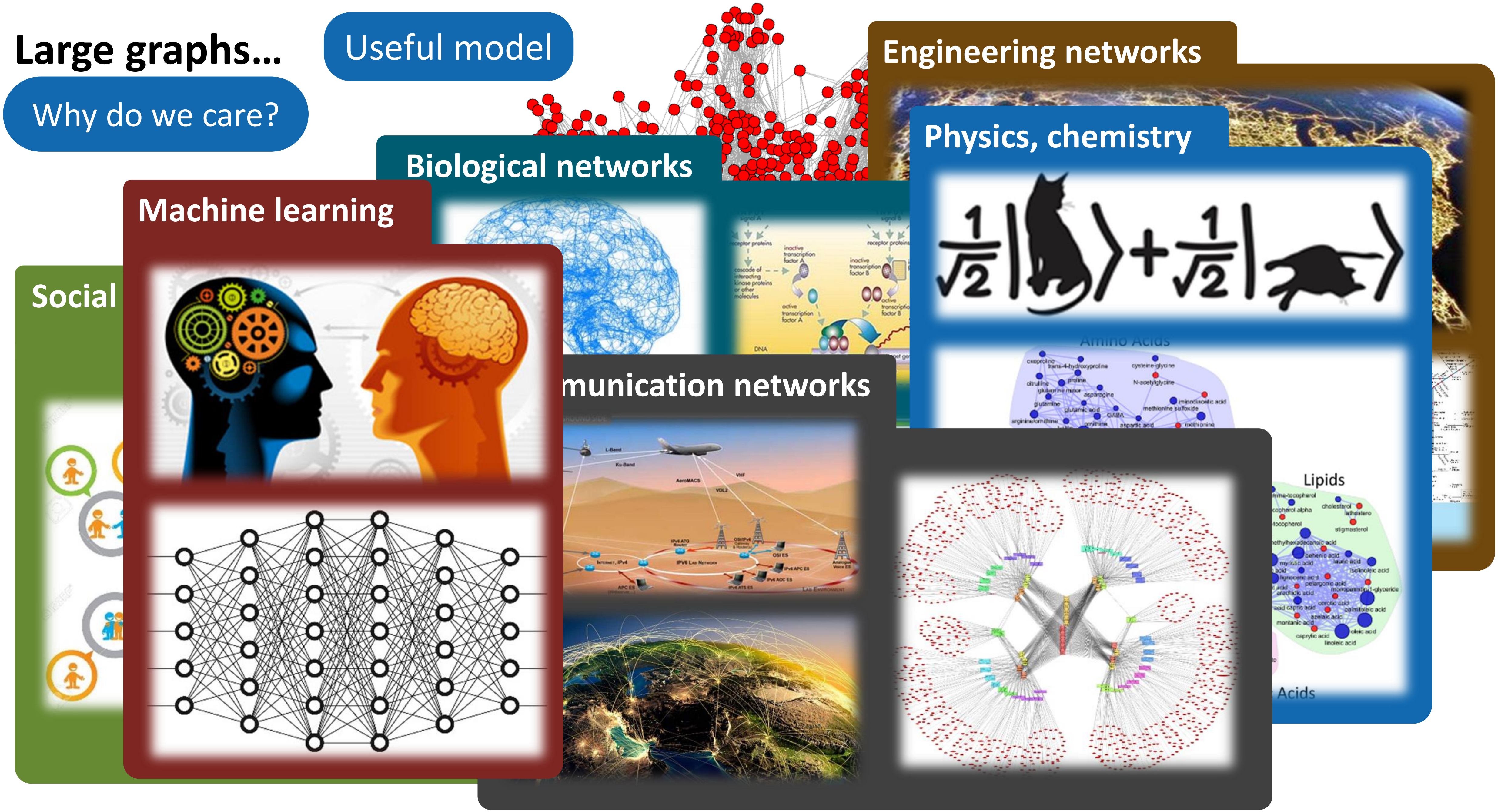
Useful model



Large graphs...

Useful model

Why do we care?



Large graphs...

Why do we care?

Useful model

Engineering networks

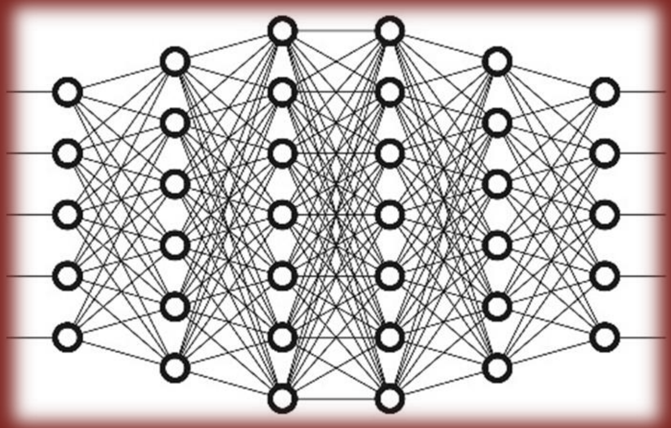
hysics, chemistry

Biological ne

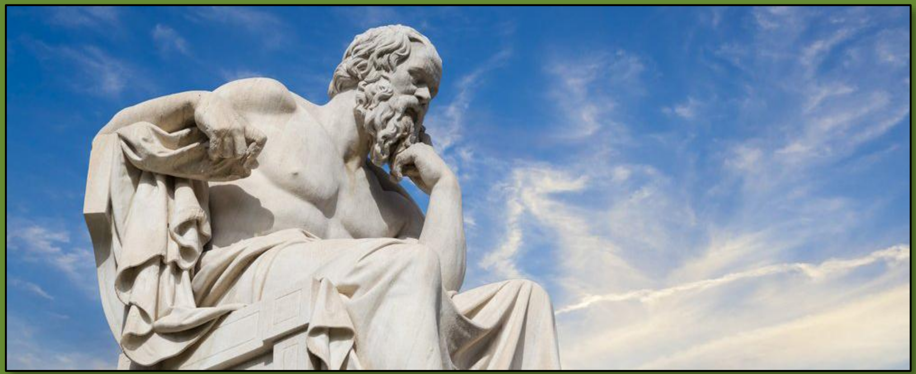
m

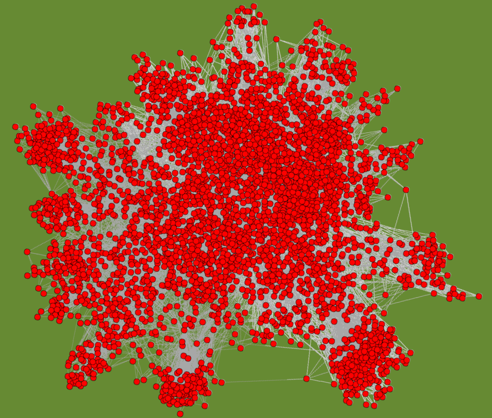
Social

Machine learning



...even philosophy ☺





FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

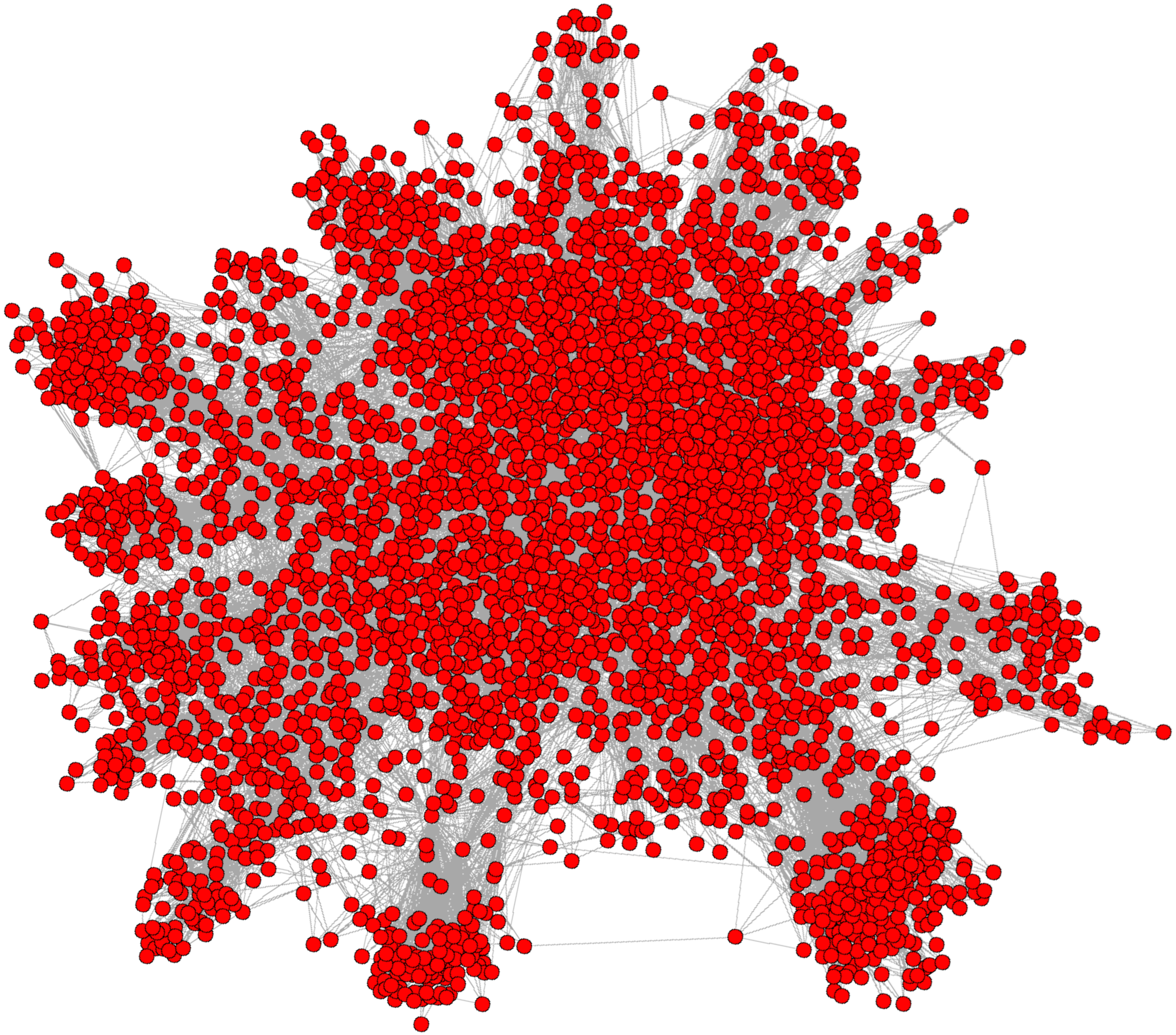
Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

- **Track**: Graph Processing devroom
- **Room**: AW1.126
- **Day**: Saturday
- **Start**: 12:45
- **End**: 13:35

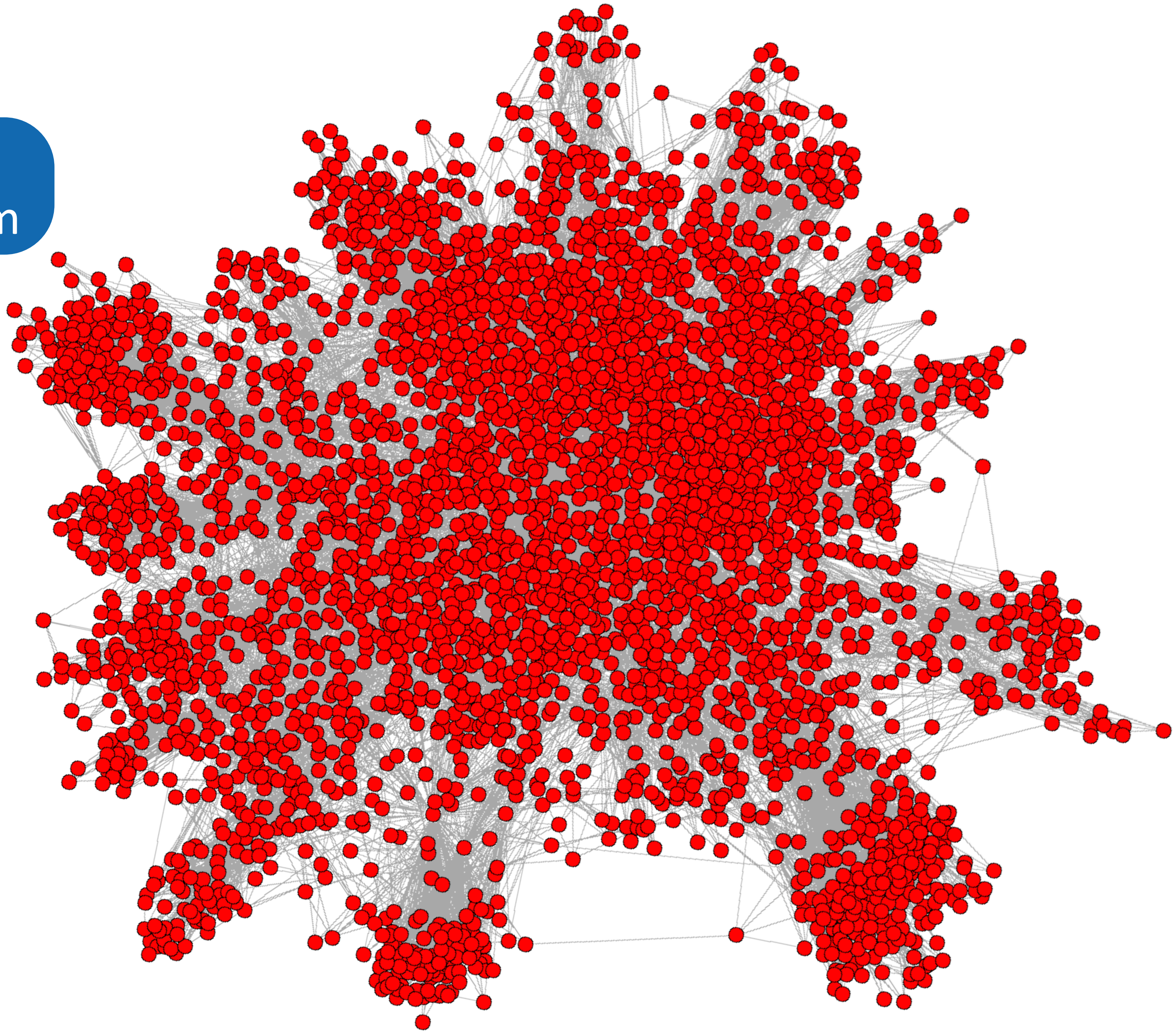
Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

Large graphs…



Large graphs…

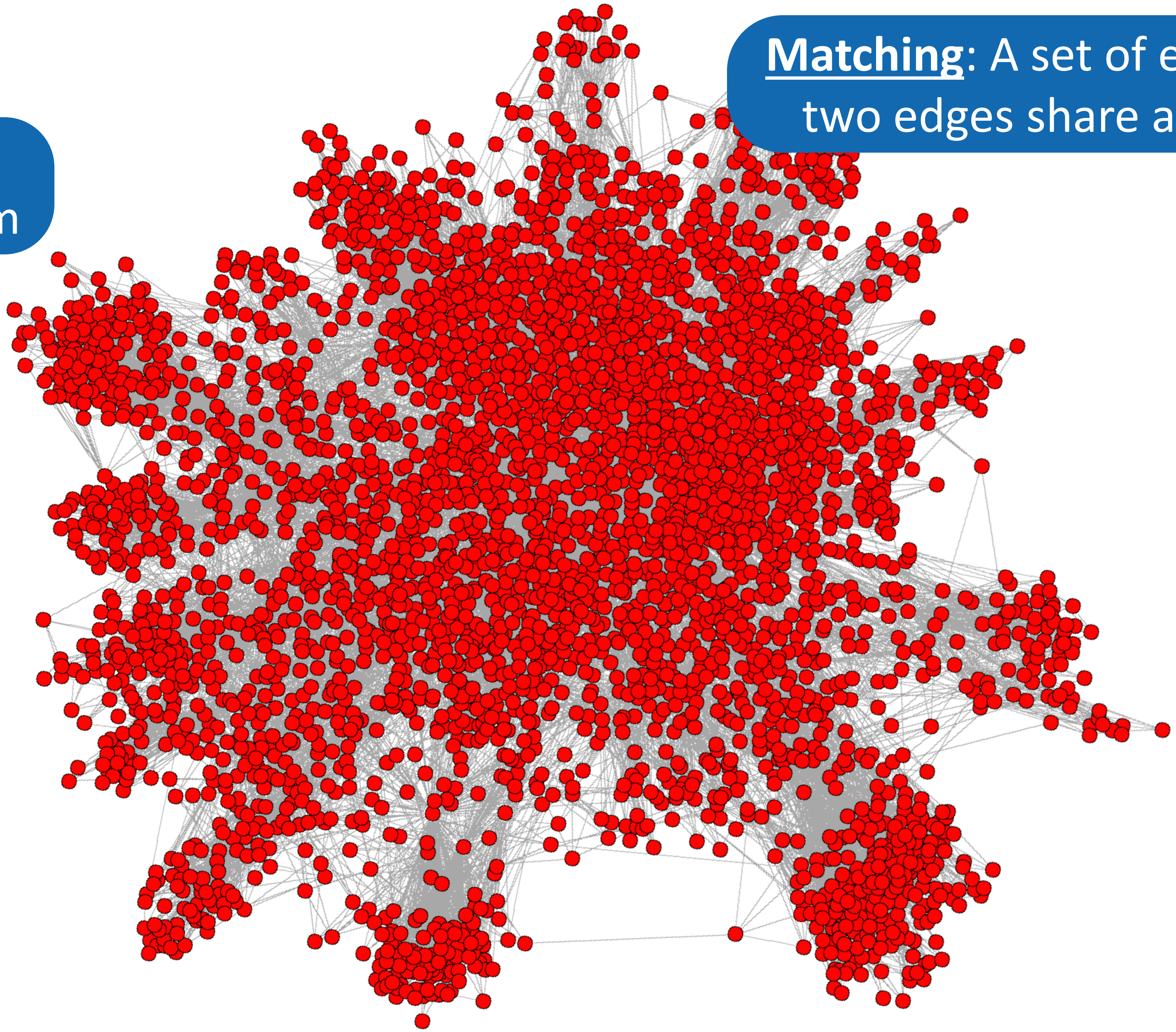
One particularly important problem



Large graphs…

One particularly important problem

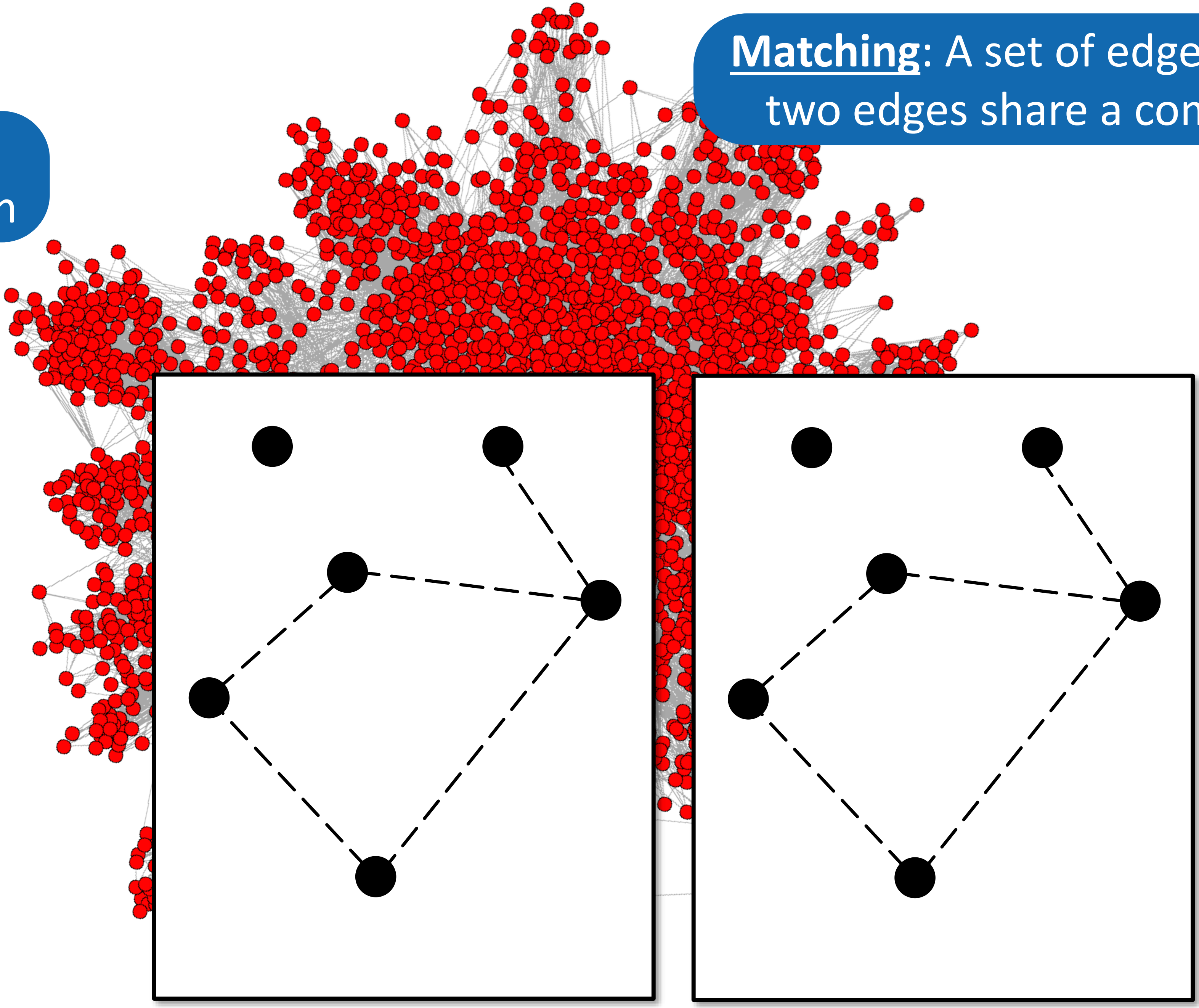
Matching: A set of edges such that no two edges share a common vertex



Large graphs…

One particularly important problem

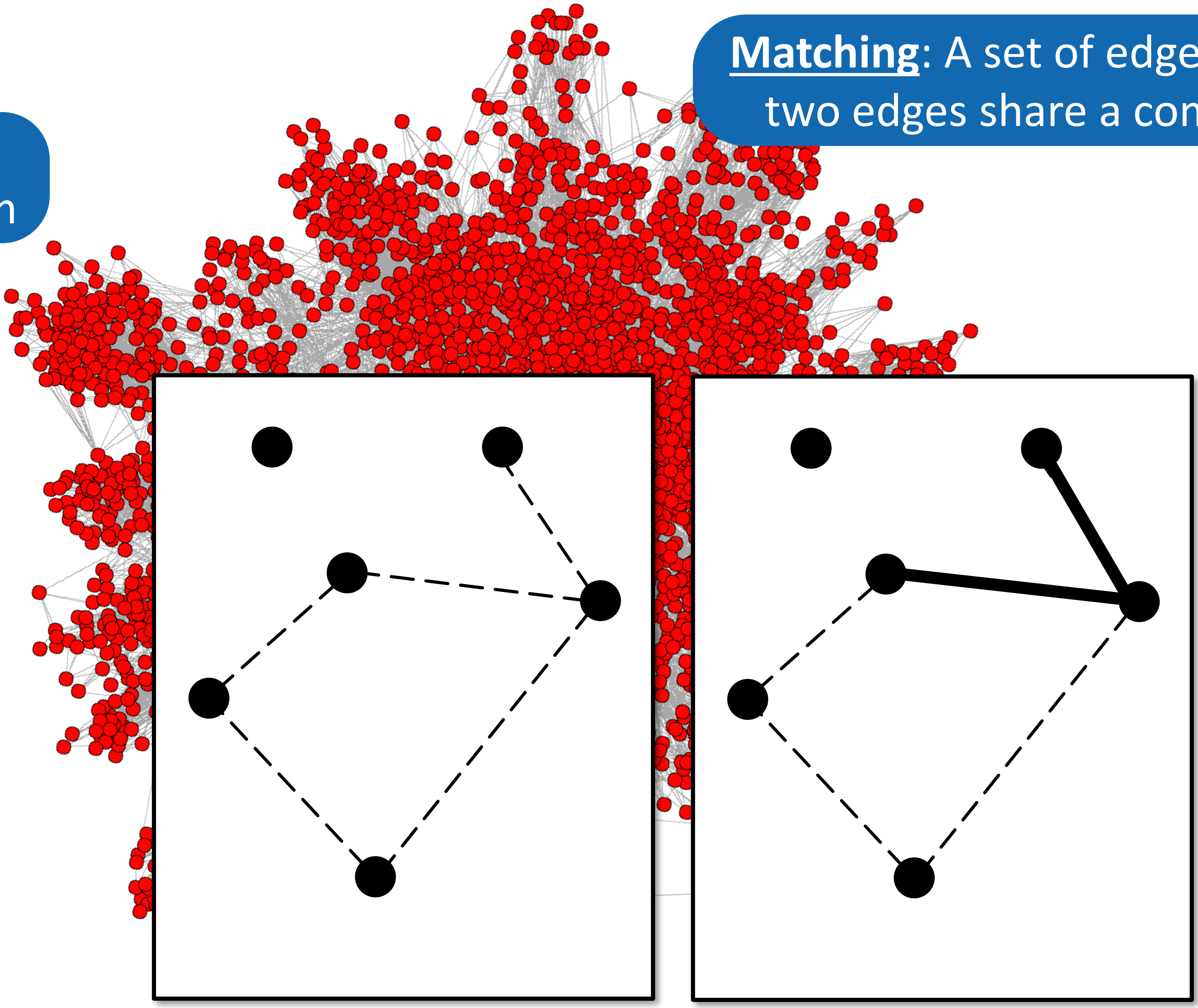
Matching: A set of edges such that no two edges share a common vertex



Large graphs...

One particularly important problem

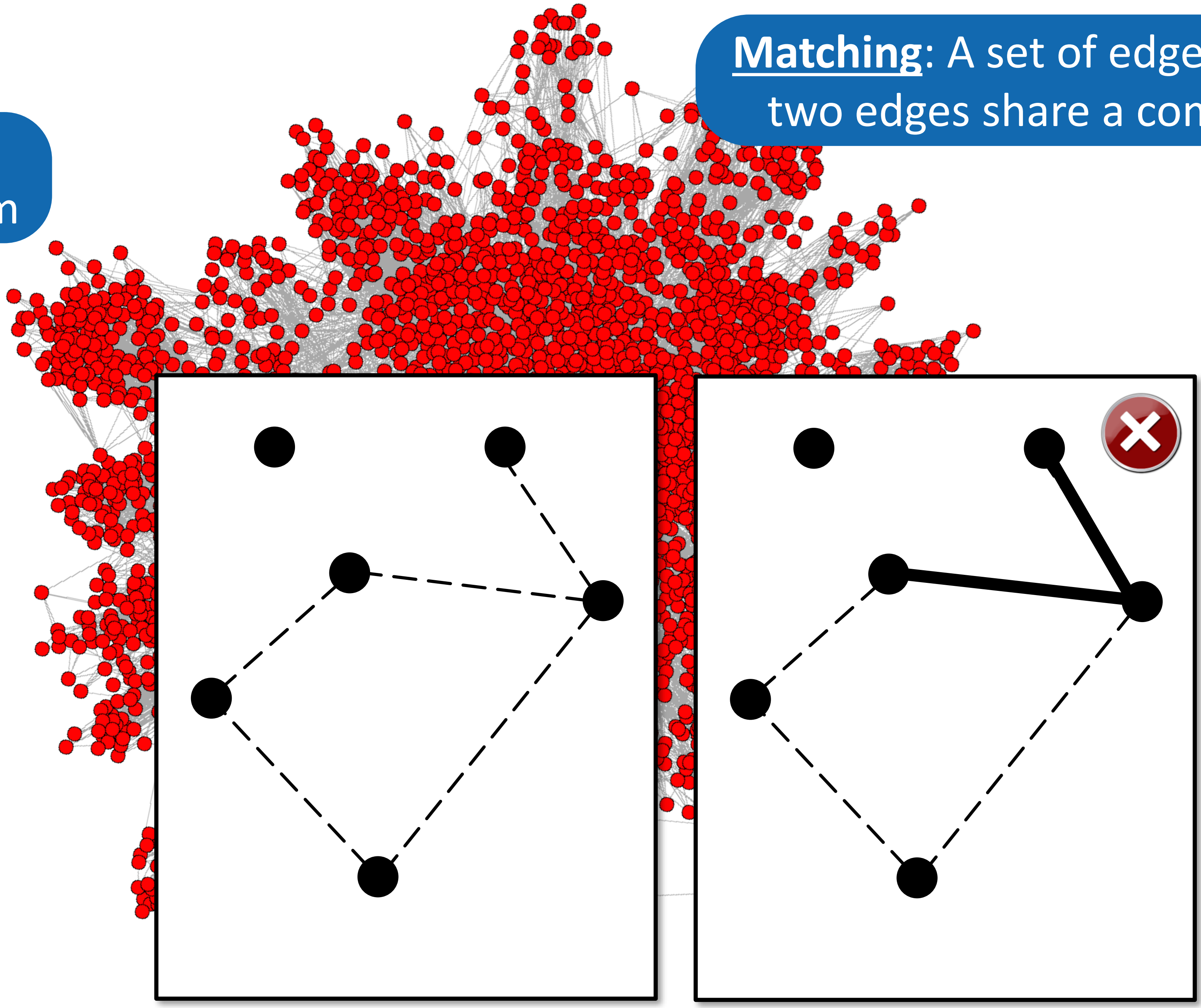
Matching: A set of edges such that no two edges share a common vertex



Large graphs...

One particularly important problem

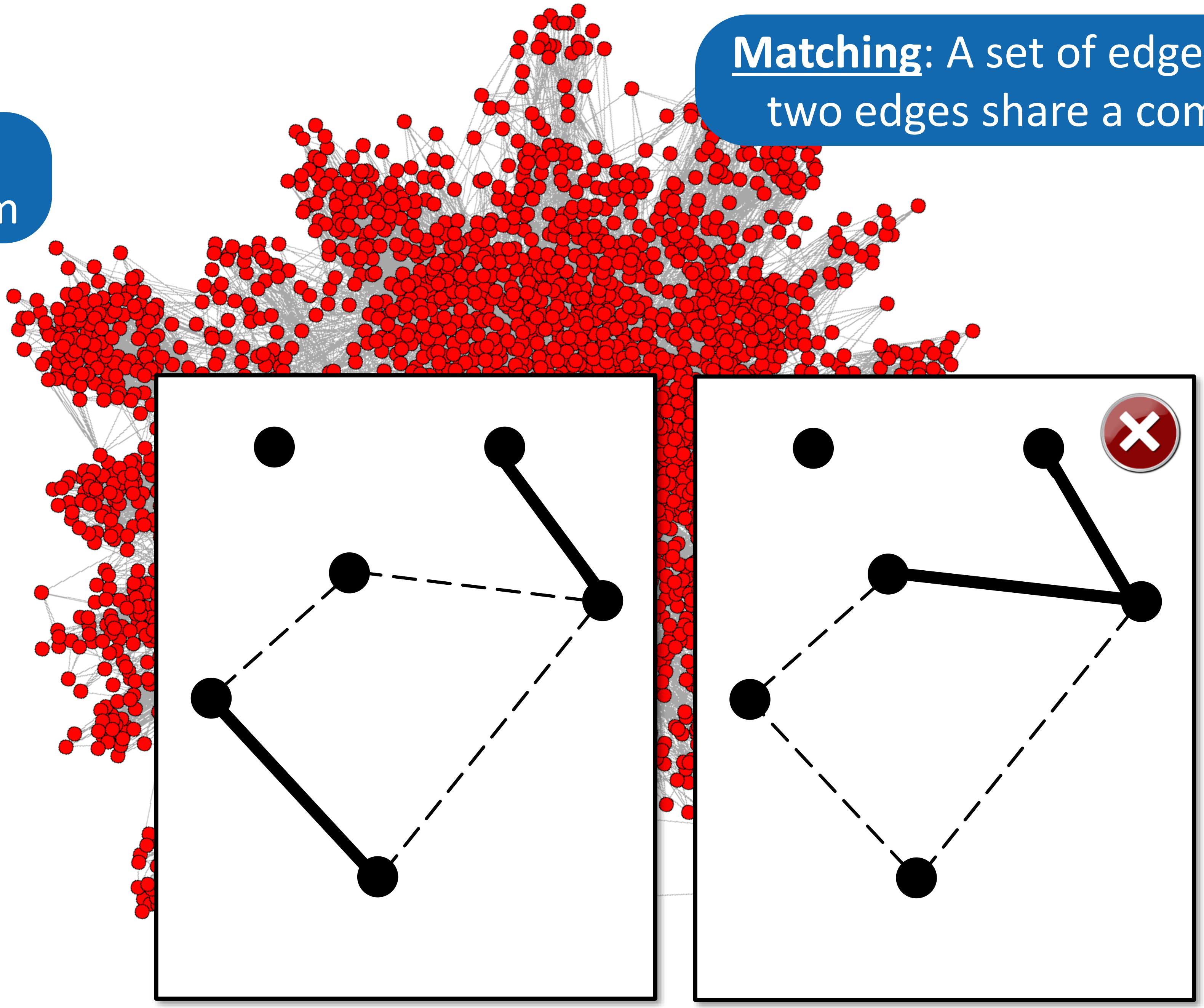
Matching: A set of edges such that no two edges share a common vertex



Large graphs…

One particularly important problem

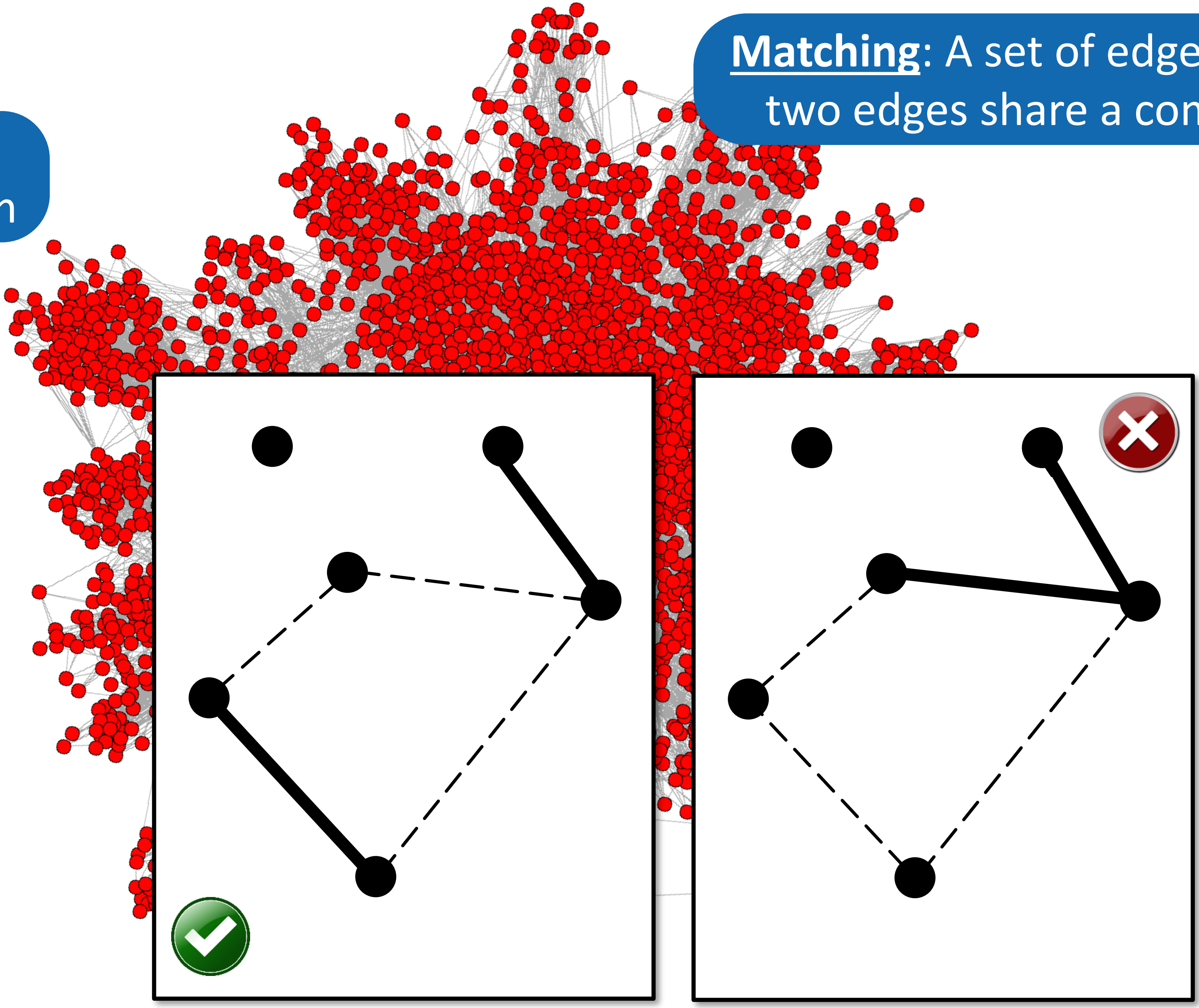
Matching: A set of edges such that no two edges share a common vertex



Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

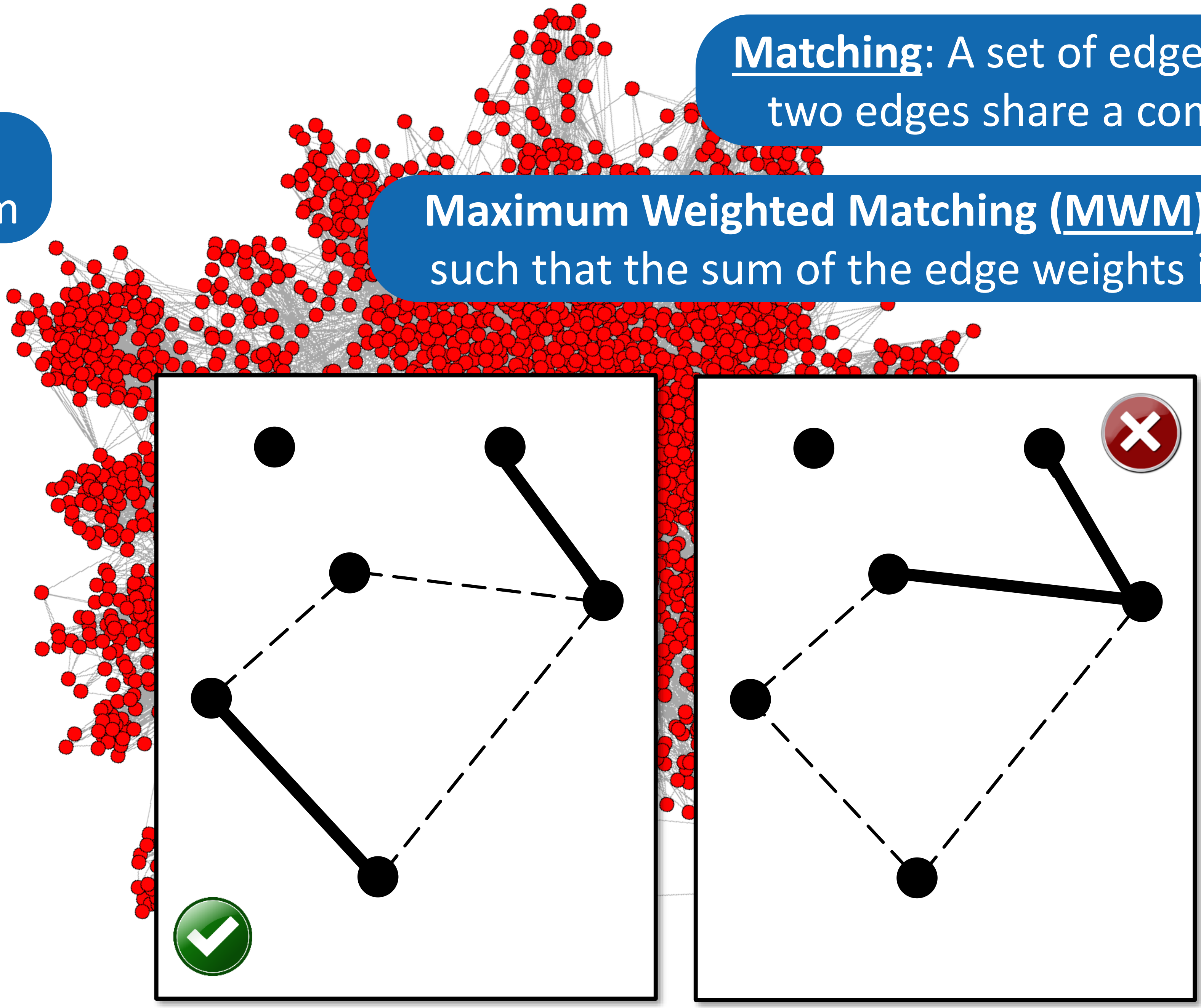


Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

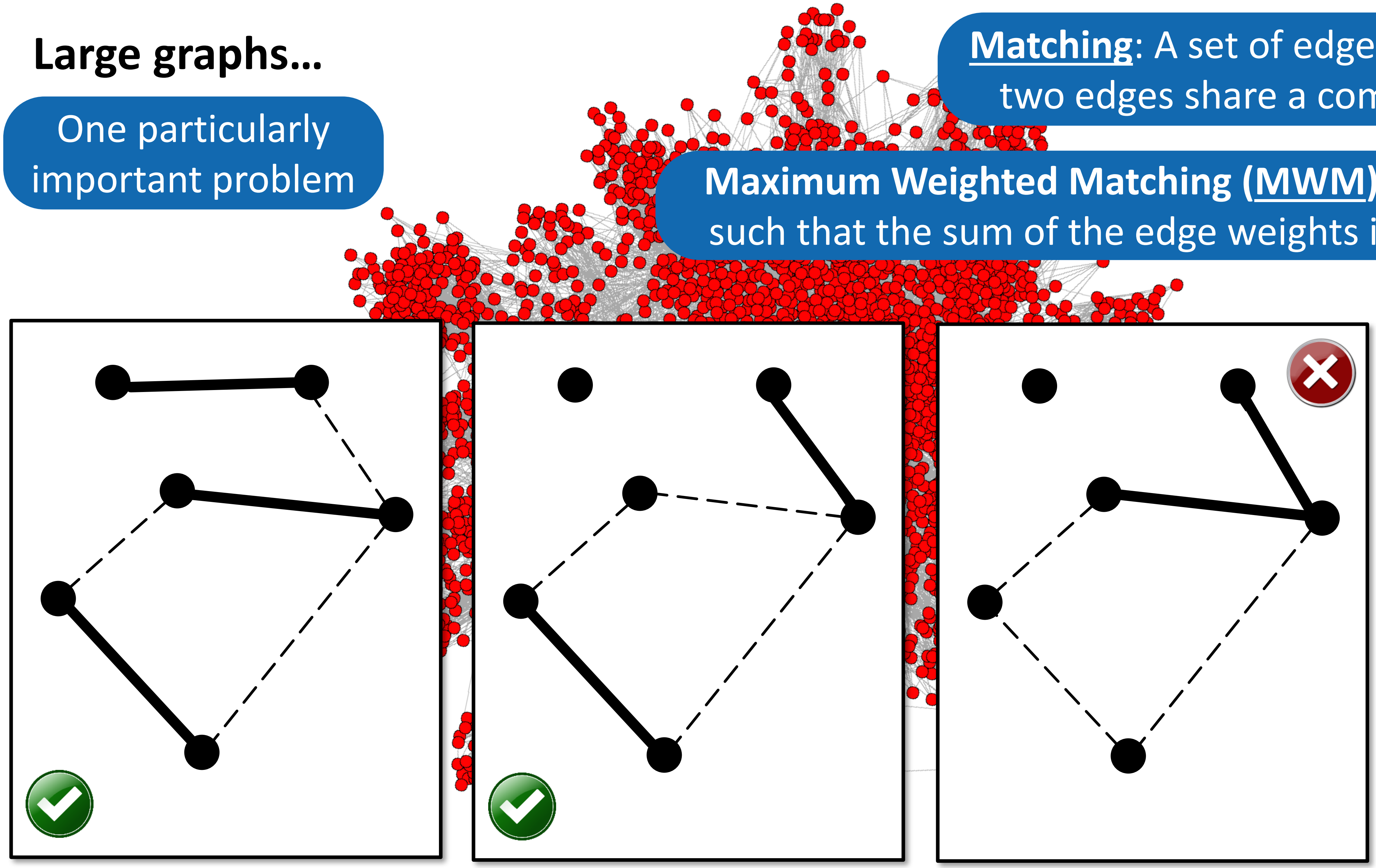


Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

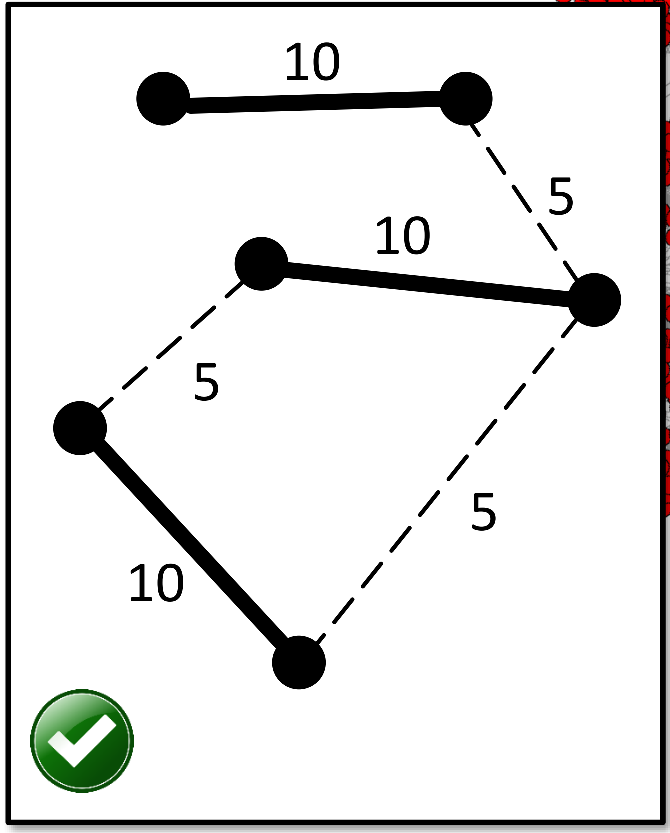


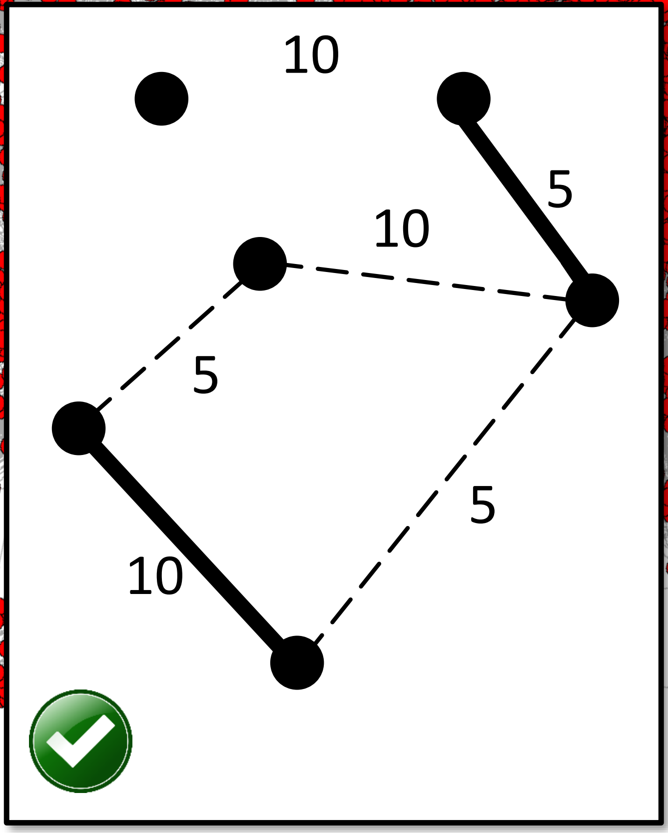
Large graphs…

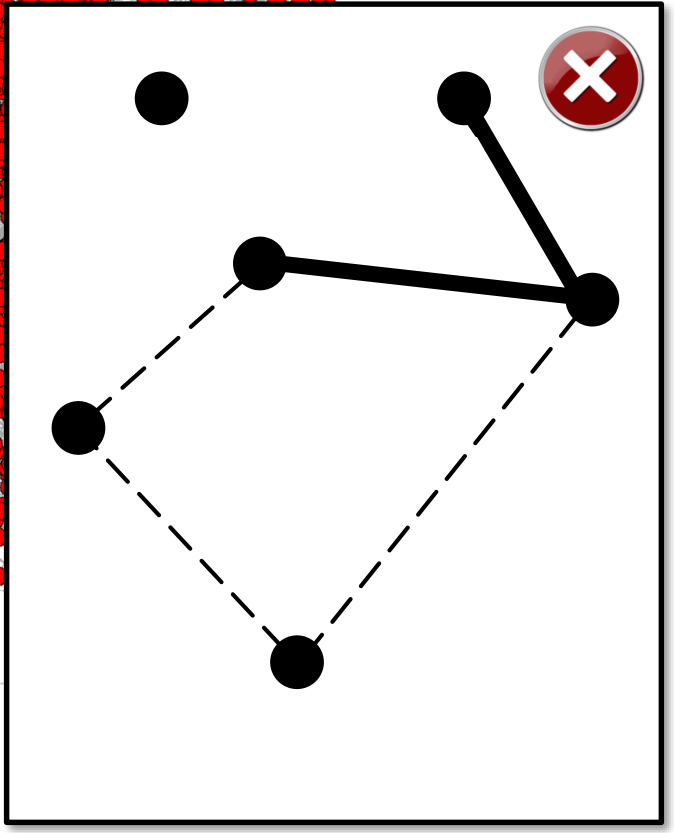
One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized





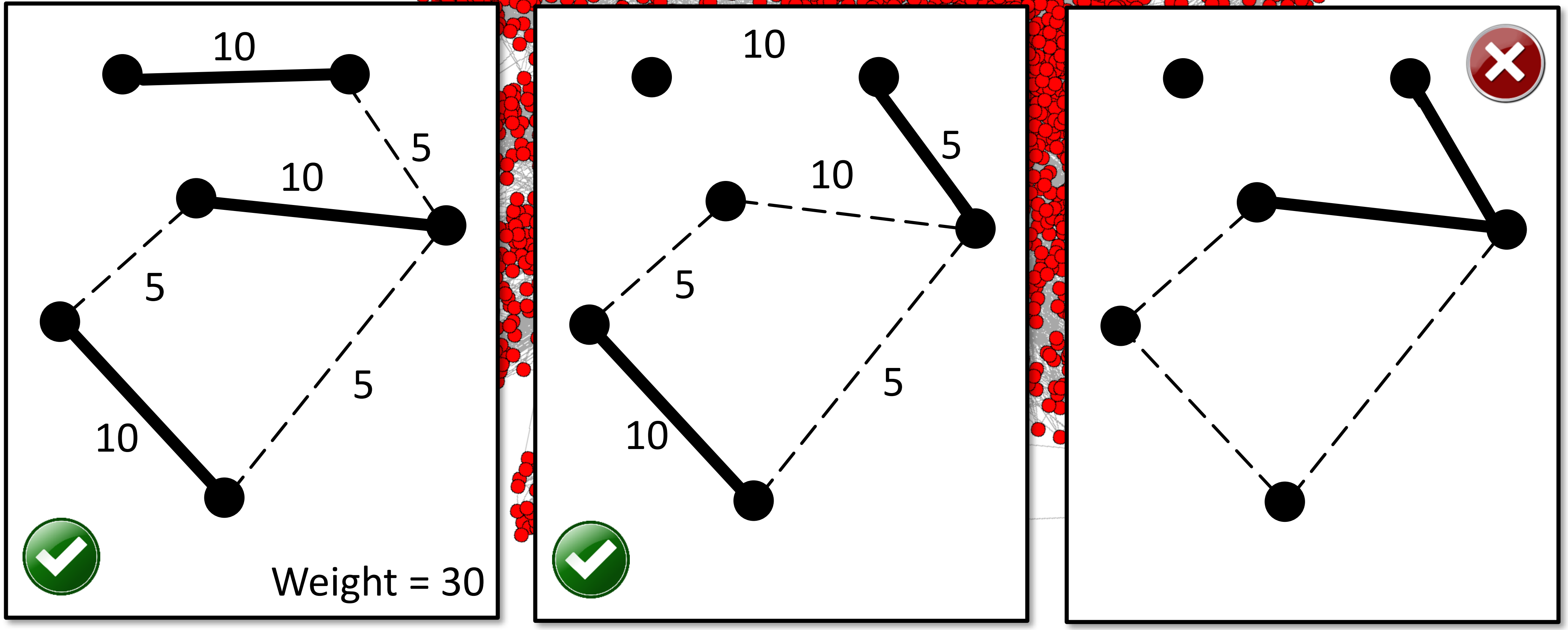


Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

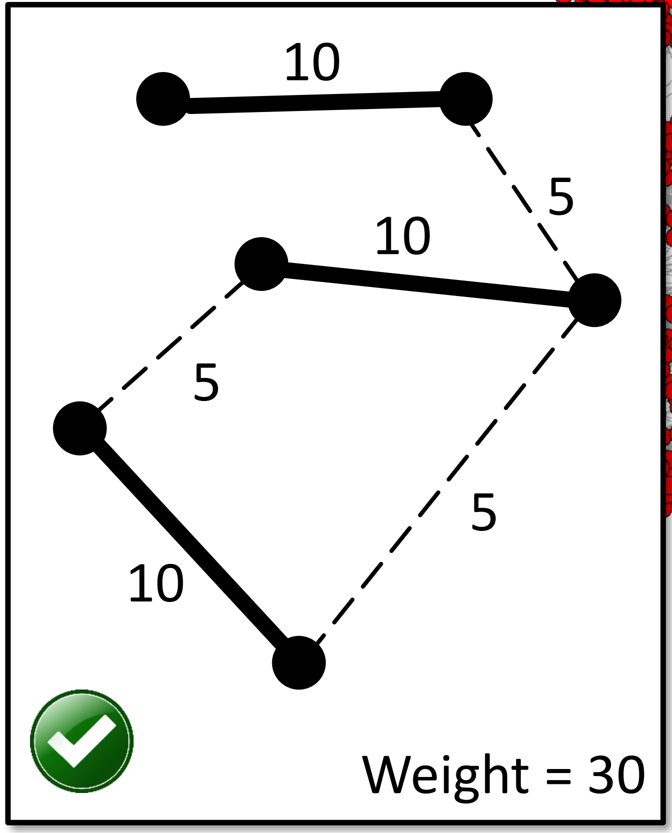


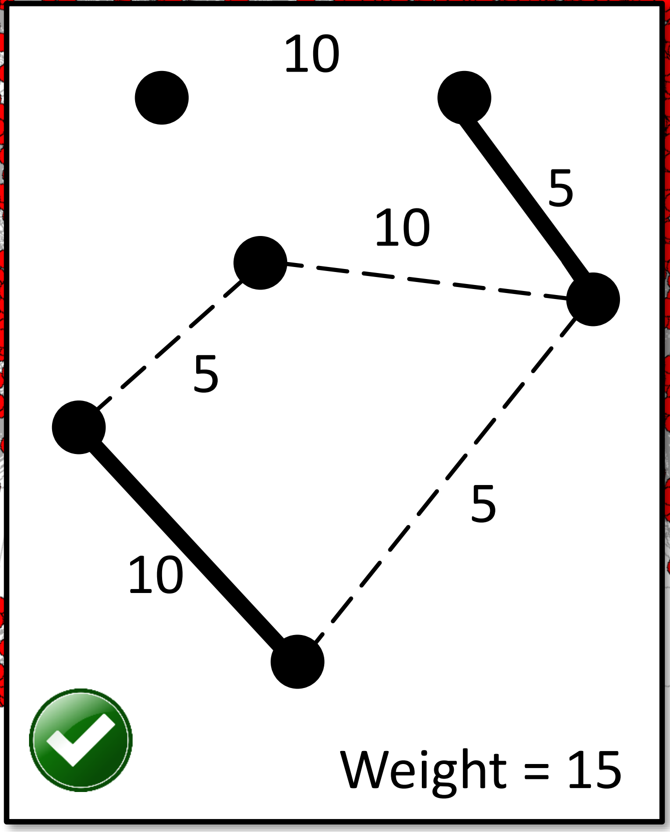
Large graphs…

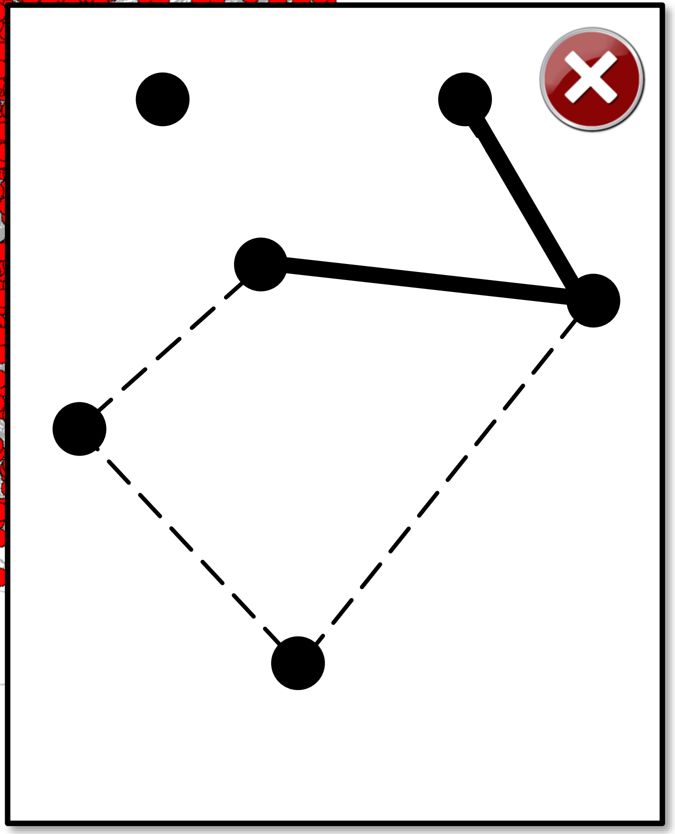
One particularly important problem

<u>Matching</u>: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (<u>MWM</u>): A matching such that the sum of the edge weights is maximized





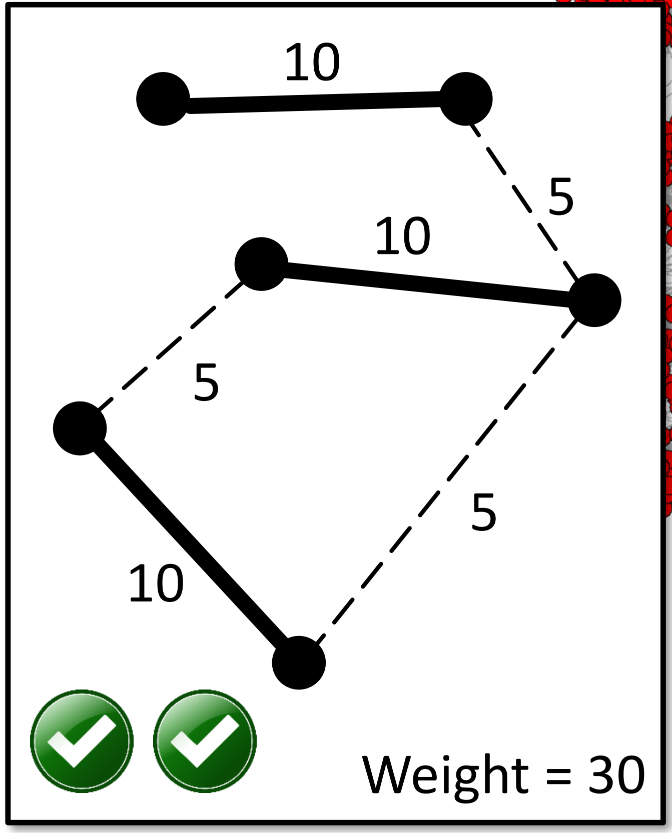


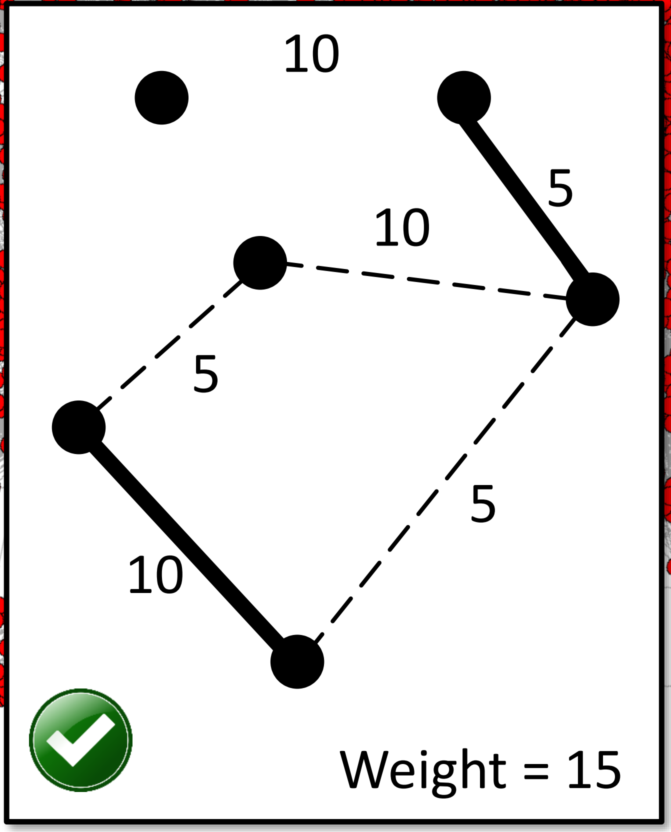
Large graphs…

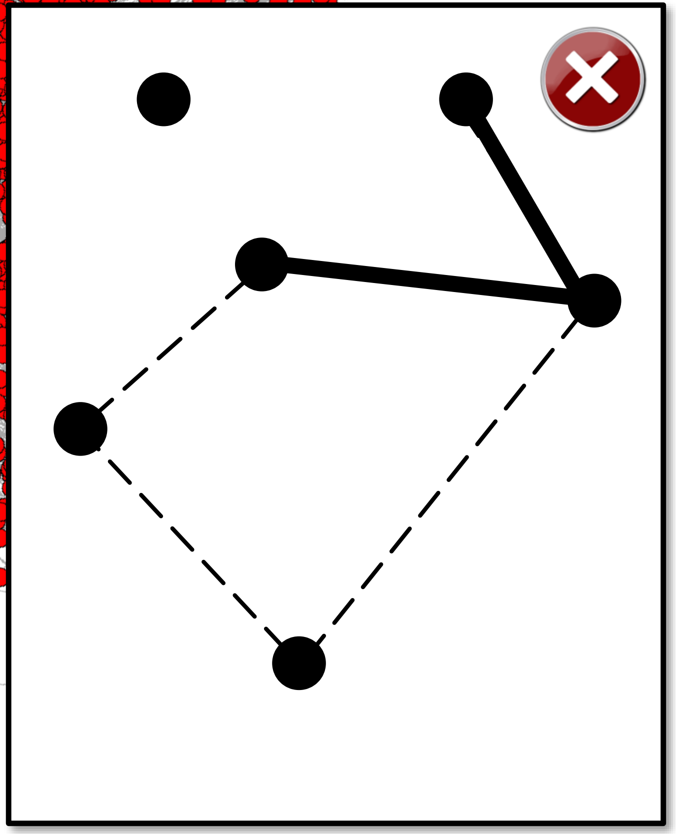
One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized





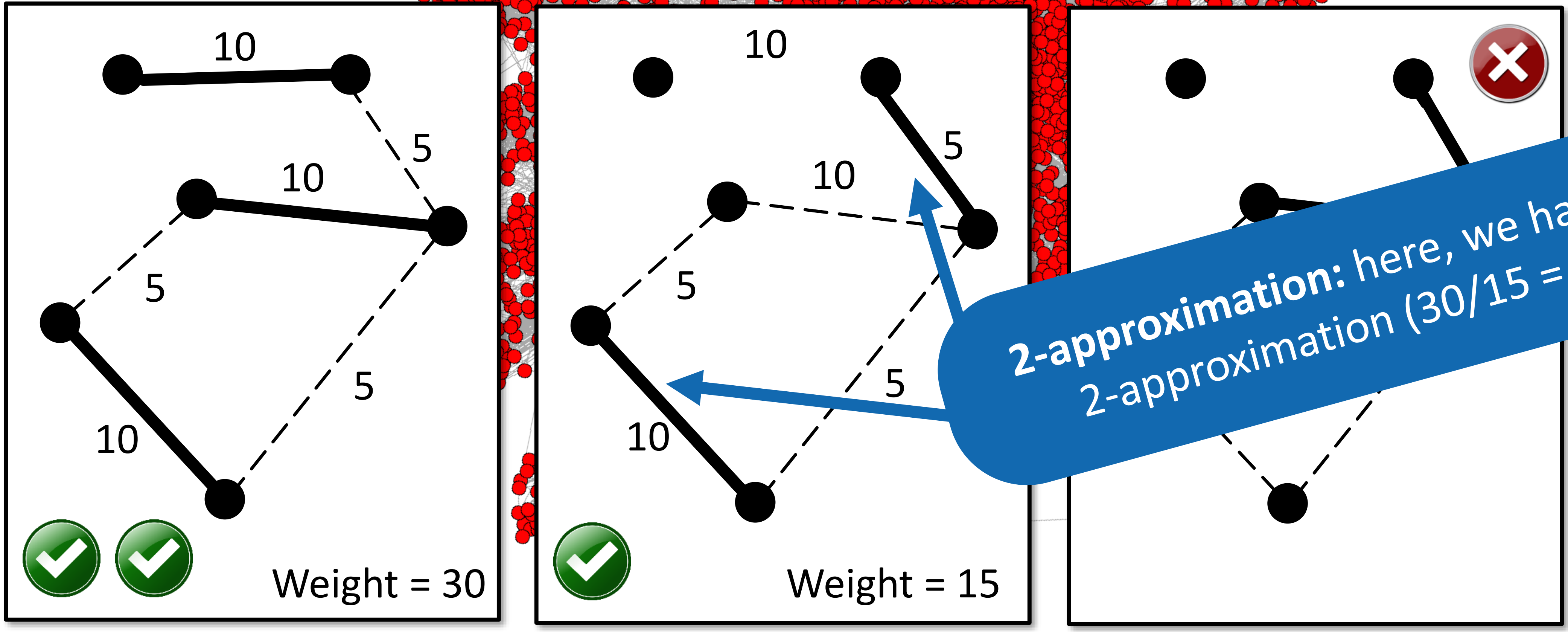


Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized



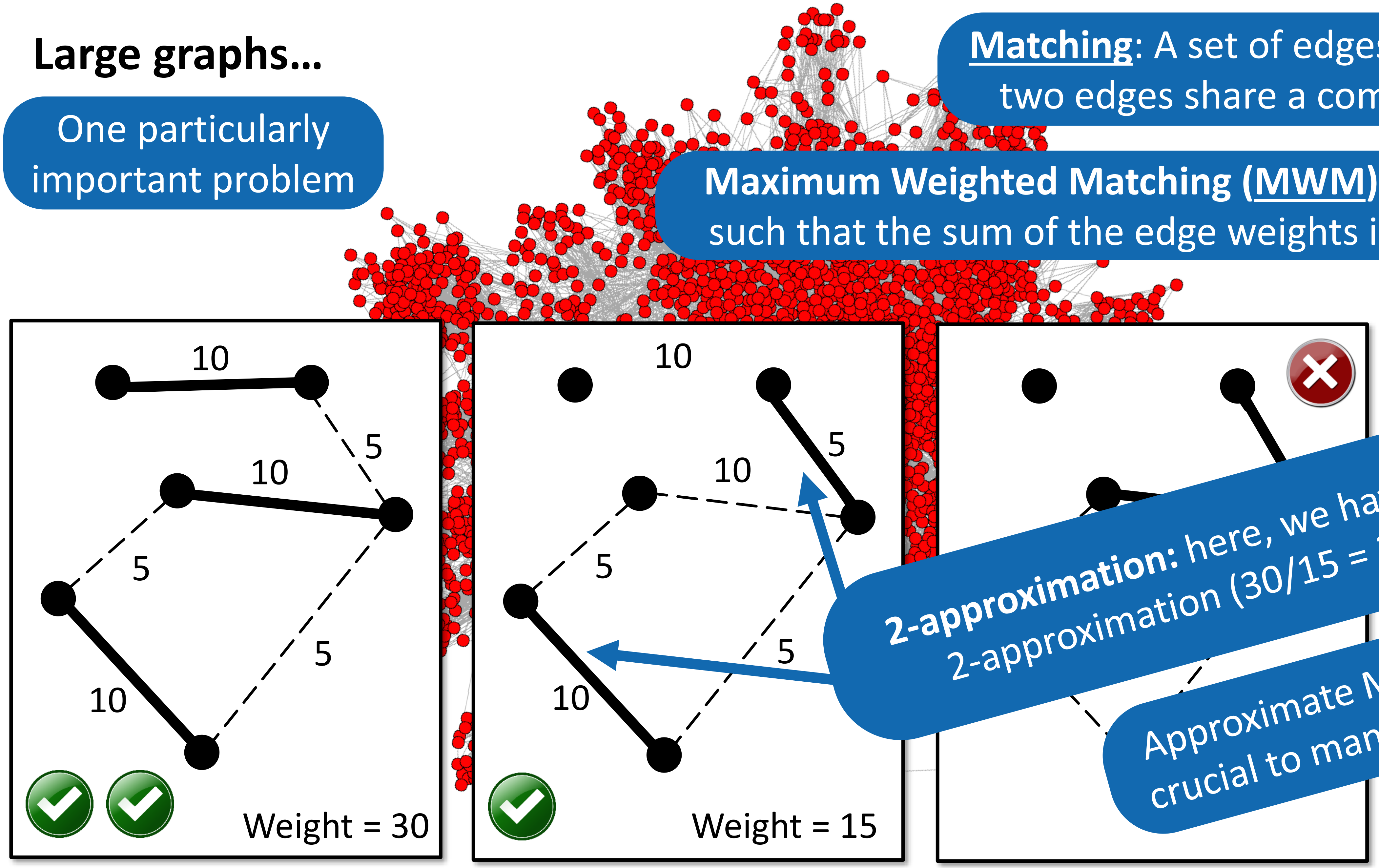
2-approximation: here, we have a 2-approximation (30/15 = 2)

Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized



2-approximation: here, we have a 2-approximation (30/15 = 2)

Approximate MWMs are crucial to many problems

Large graphs…

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

2-approximation: here, we have a 2-approximation (30/15 = 2)

Approximate MWMs are crucial to many problems

Large graphs...

One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Why do we care?

2-approximation: here, we have a 2-approximation (30/15 = 2)

Approximate MWMs are crucial to many problems

Scheduling

Why do we care?

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Weight = 30

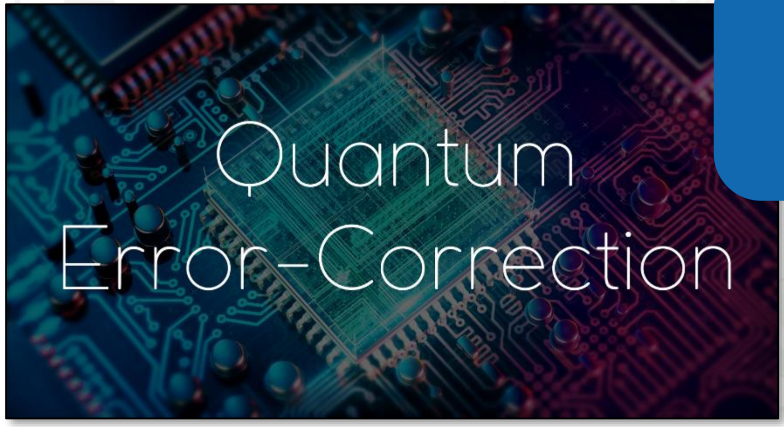
Weight = 15

2-approximation: here, we have a 2-approximation (30/15 = 2)

Approximate MWMs are crucial to many problems

Scheduling

Why do we care?



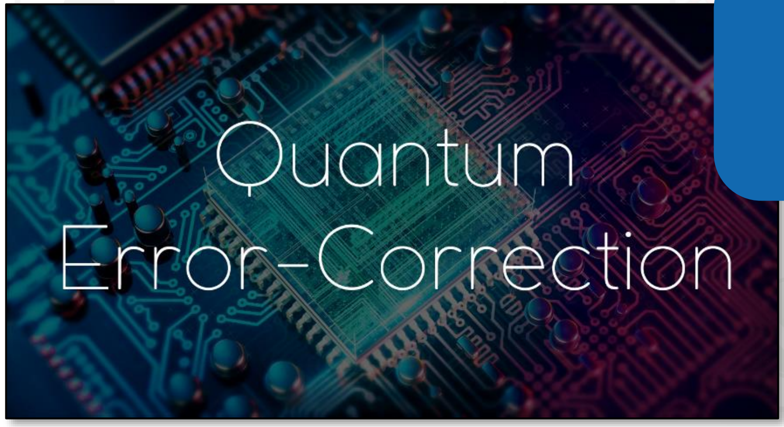
[Quantum] error correcting codes

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Why do we care?

Scheduling



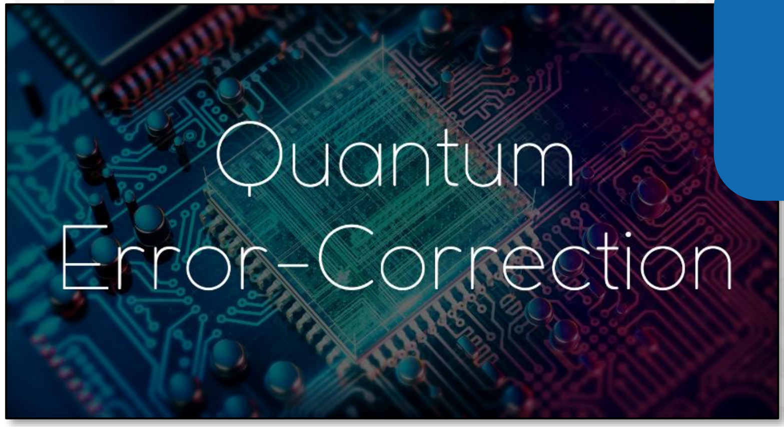
[Quantum] error correcting codes

Transplant matching

Why do we care?

Scheduling

Traveling Salesman Problem



[Quantum] error correcting codes

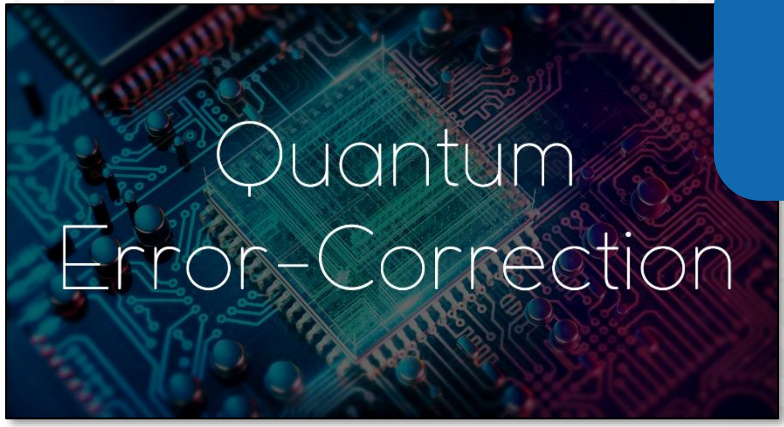
Transplant matching

Scheduling

Traveling Salesman Problem

Many, many others...

Why do we care?



[Quantum] error correcting codes

Transplant matching

In all cases, approximations („reasonably" accurate) are useful

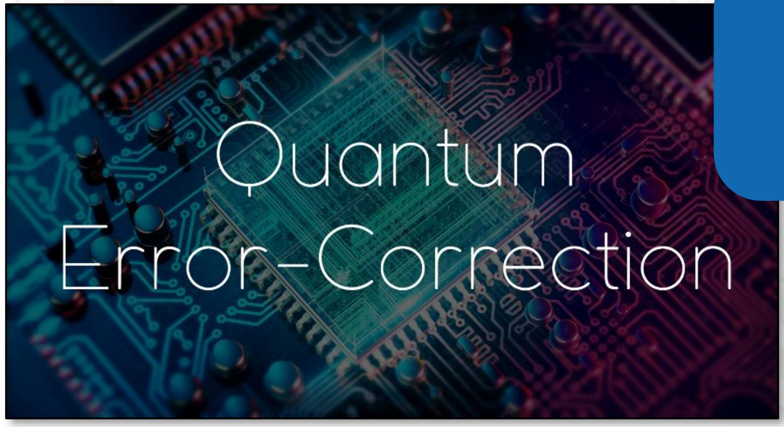
"We live in a system of approximations" — Ralph Waldo Emerson

Many, many others...

Traveling Salesman Problem

Scheduling

Why do we care?



[Quantum] error correcting codes

Transplant matching

Research Questions

Research Questions

Which programming paradigm to use for (approximate) MWM (and other graph problems)?

Research Questions

Which programming paradigm to use for (approximate) MWM (and other graph problems)?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

Research Questions

Which programming paradigm to use for (approximate) MWM (and other graph problems)?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

Research Questions

Which programming paradigm to use for (approximate) MWM (and other graph problems)?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

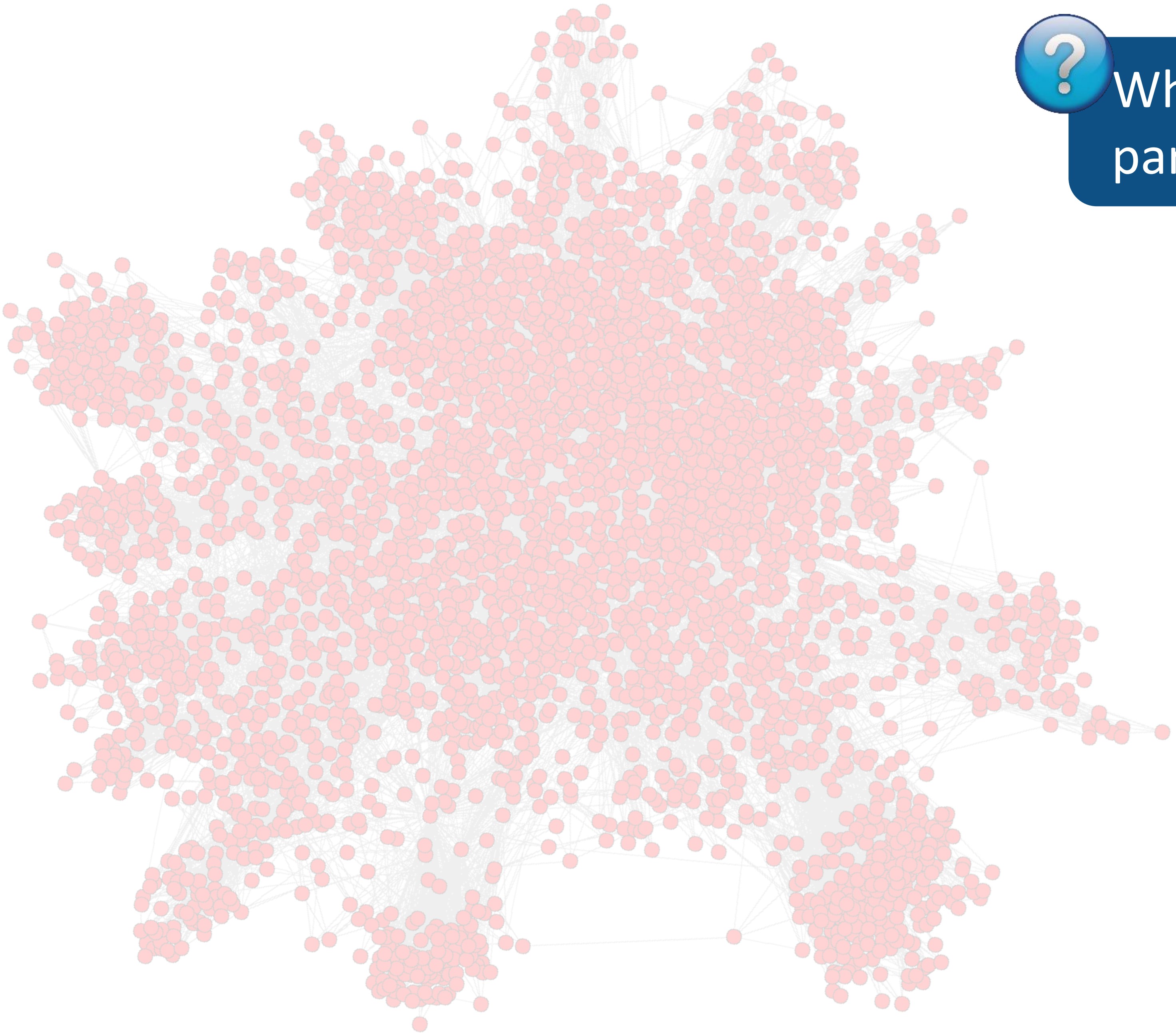
Research Questions

Which programming paradigm to use for (approximate) MWM (and other graph problems)?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

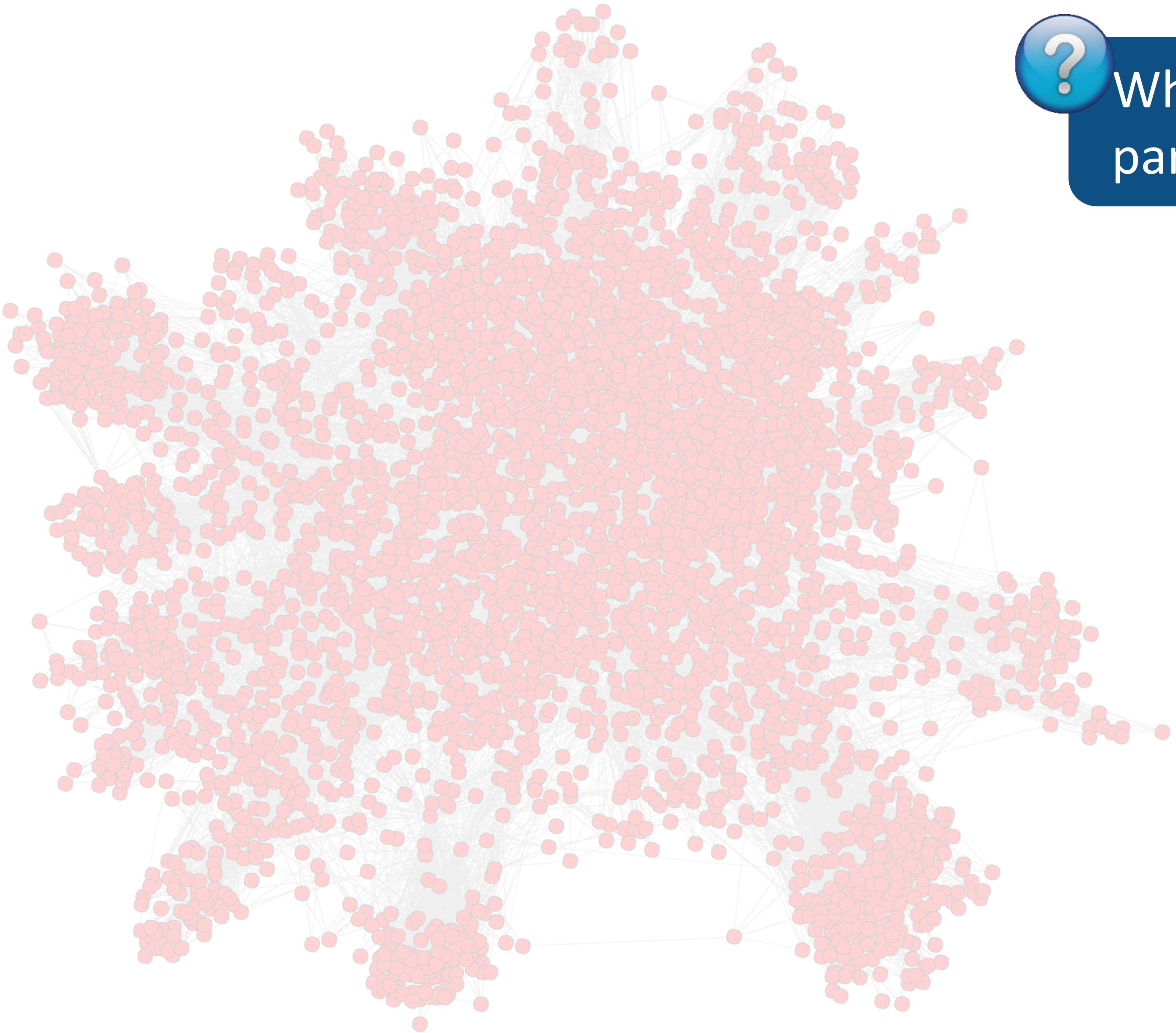
What is the ultimate performance, power consumption, and the related tradeoffs?



What programming paradigm and why?

What programming paradigm and why?

<u>Part 1</u>: Seeking "the best paradigm", we conducted a detailed analysis of graph processing on FPGAs



What programming paradigm and why?

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^{\dagger}$	$m^{\dagger}$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'15	👍	SpMV	Vertex-Centric, BSP	C++ (HLS)	👍	BRAM	17k	126k
Dai [40] **(FPGP)**	FPGA'16	👍	BFS	None	unsp.	👍	DRAM	41.6M	1.4B
Oguntebi [93] **(GraphOps)**	FPGA'16	👍	BFS, SpMV, PR, Vertex Cover	None	MaxJ (HLS)	👎	BRAM	16M	128M
Zhou [134]	FCCM'16	👍	SSSP, WCC, MST	Edge-Centric	unsp.	👎	DRAM	4.7M	65.8M
Engelhardt [49] **(GraVF)**	FPL'16	👍	BFS, PR, SSSP, CC	Vertex-Centric	Migen (HLS)	👎	BRAM	128k	512k
Dai [41] **(ForeGraph)**	FPGA'17	👍	PR, BFS, WCC	None	unsp.	👍	DRAM	41.6M	1.4B
Zhou [136]	SBAC-PAD'17	👍	BFS, SSSP	Hybrid (Vertex- and Edge-Centric)	unsp.	👎	DRAM	10M	160M
Ma [85]	FPGA'17	👍	BFS, SSSP, CC, TC, BC	Transactional Memory [16, 59]	System-Verilog	👍	DRAM	24M	58M
Lee [79] **(ExtraV)**	FPGA'17	👍	BFS, PR, CC, AT* [60]	Graph Virtualization	C++ (HLS)	👎	DRAM	124M	1.8B
Zhou [135]	CF'18	👍	SpMV, PR	Edge-Centric, GAS	unsp.	👎	DRAM	41.6M	1.4B
Yang [125]	report (2018)	👍	BFS, PR, WCC	None	OpenCL	👎		4.85M	69M
Yao [127]	report (2018)	👍	BFS, PR, WCC	None	unsp.	👎	BRAM	4.85M	69M

What programming paradigm and why?

~15 FPGA graph processing frameworks

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^\dagger$	$m^\dagger$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'1								
Dai [40] **(FPGP)**	FPGA'1								
Oguntebi [93] **(GraphOps)**	FPGA'1								
Zhou [134]	FCCM'								
Engelhardt [49] **(GraVF)**	FPL'16								
Dai [41] **(ForeGraph)**	FPGA'1								
Zhou [136]	SBAC-P								
Ma [85]	FPGA'1								
Lee [79] **(ExtraV)**	FPGA'1								
Zhou [135]	CF'18								
Yang [125]	report (								
Yao [127]	report (								

Babb [4]	report (1996)	👎	SSSP	None	Verilog	👍	Hardwired	512	2051
Dandalis [43]	report (1999)	👎	SSSP	None	unsp.	👍	Hardwired	2048	32k
Tommiska [116]	report (2001)	👎	SSSP	None	VHDL	👎	BRAM	64	4096
Mencer [87]	FPL'02	👎	Reachability, SSSP	None	PAM- -Bloks II	👎	Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] **(CyGraph)**	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

What programming paradigm and why?

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^\dagger$	$m^\dagger$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'1								
Dai [40] **(FPGP)**	FPGA'1								
Oguntebi [93] **(GraphOps)**	FPGA'1								
Zhou [134]	FCCM'								
Engelhardt [49] **(GraVF)**	FPL'16								
Dai [41] **(ForeGraph)**	FPGA'1								
Zhou [136]	SBAC-P								
Ma [85]	FPGA'1								
Lee [79] **(ExtraV)**	FPGA'1								
Zhou [135]	CF'18								
Yang [125]	report (								
Yao [127]	report (								

Babb [4]	report (1996)	👎	SSSP	None	Verilog	👍	Hardwired	512	2051
Dandalis [43]	report (1999)	👎	SSSP	None	unsp.	👍	Hardwired	2048	32k
Tommiska [116]	report (2001)	👎	SSSP	None	VHDL	👎	BRAM	64	4096
Mencer [87]	FPL'02	👎	Reachability, SSSP	None	PAM--Bloks II	👎	Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] **(CyGraph)**	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

What programming paradigm and why?

Key techniques, paradigms, challenges, features, …

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^\dagger$	$m^\dagger$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*,	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'1								
Dai [40] **(FPGP)**	FPGA'1								
Oguntebi [93] **(GraphOps)**	FPGA'1								
Zhou [134]	FCCM'								
Engelhardt [49] **(GraVF)**	FPL'16								
Dai [41] **(ForeGraph)**	FPGA'1								
Zhou [136]	SBAC-P								
Ma [85]	FPGA'1								
Lee [79] **(ExtraV)**	FPGA'1								
Zhou [135]	CF'18								
Yang [125]	report								
Yao [127]	report								

Reference	Venue	Generic Design	Considered Problems	Programming Model or Technique	Used Language	Multi FPGAs	Input Location	n	m
Bab							Hardwired	512	2051
Da							Hardwired	2048	32k
To							BRAM	64	4096
Me							Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] **(CyGraph)**	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

Selected parts are in the FPGA paper, the rest is in…

What programming paradigm and why?

Key techniques, paradigms, challenges, features, …

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^{\dagger}$	$m^{\dagger}$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*,	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'1								
Dai [40] **(FPGP)**	FPGA'1								
Oguntebi [93] **(GraphOps)**	FPGA'1								
Zhou [134]	FCCM'								
Engelhardt [49] **(GraVF)**	FPL'16								
Dai [41] **(ForeGraph)**									
Zhou [136]									
Ma [85]									
Lee [79] **(ExtraV)**									
Zhou [135]									
Yang [125]									
Yao [127]									

Bab							Hardwired	512	2051
Dar							Hardwired	2048	32k
Ton							BRAM	64	4096
Me							Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
								300k	3M
								128	466
								38k	72M
								16.8M	1.1B
								8.4M	536M
								16M	512M
								1M	unsp.
								2.4M	5M
								2.1M	65M
								23.9M	58.2M
								33.6M	536.9M
								4.8M	117M

Selected parts are in the FPGA paper, the rest is in…

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for

What programming paradigm and why?

Key techniques, paradigms, challenges, features, …

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^{\dagger}$	$m^{\dagger}$
Kapre [71] (GraphStep)	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] (GraphGen)	FCCM'14	👍	TRW-S*,	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] (GraphSoC)	ASAP'1								
Dai [40] (FPGP)	FPGA'1								
Oguntebi [93] (GraphOps)	FPGA'1								
Zhou [134]	FCCM'								
Engelhardt [49] (GraVF)	FPL'16								
Dai [41] (ForeGraph)									
Zhou [136]									
Ma [85]									
Lee [79] (ExtraV)									
Zhou [135]									
Yang [125]									
Yao [127]									

Bab			
Dar			
Tor			
Me			
Bondhugula [27]	IPDPS'06	👎	APSP
Sridharan[110]			

Input Location	n	m
Hardwired	512	2051
Hardwired	2048	32k
BRAM	64	4096
Hardwired (3-state buffers)	88	7744
DRAM	unsp.	
BRAM	64	88
DRAM	65.5k	1M
	300k	3M
	128	466
	38k	72M
	16.8M	1.1B
	8.4M	536M
	16M	512M
	1M	unsp.
	2.4M	5M
	2.1M	65M
	23.9M	58.2M
	33.6M	536.9M
	4.8M	117M

Selected parts are in the FPGA paper, the rest is in...

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

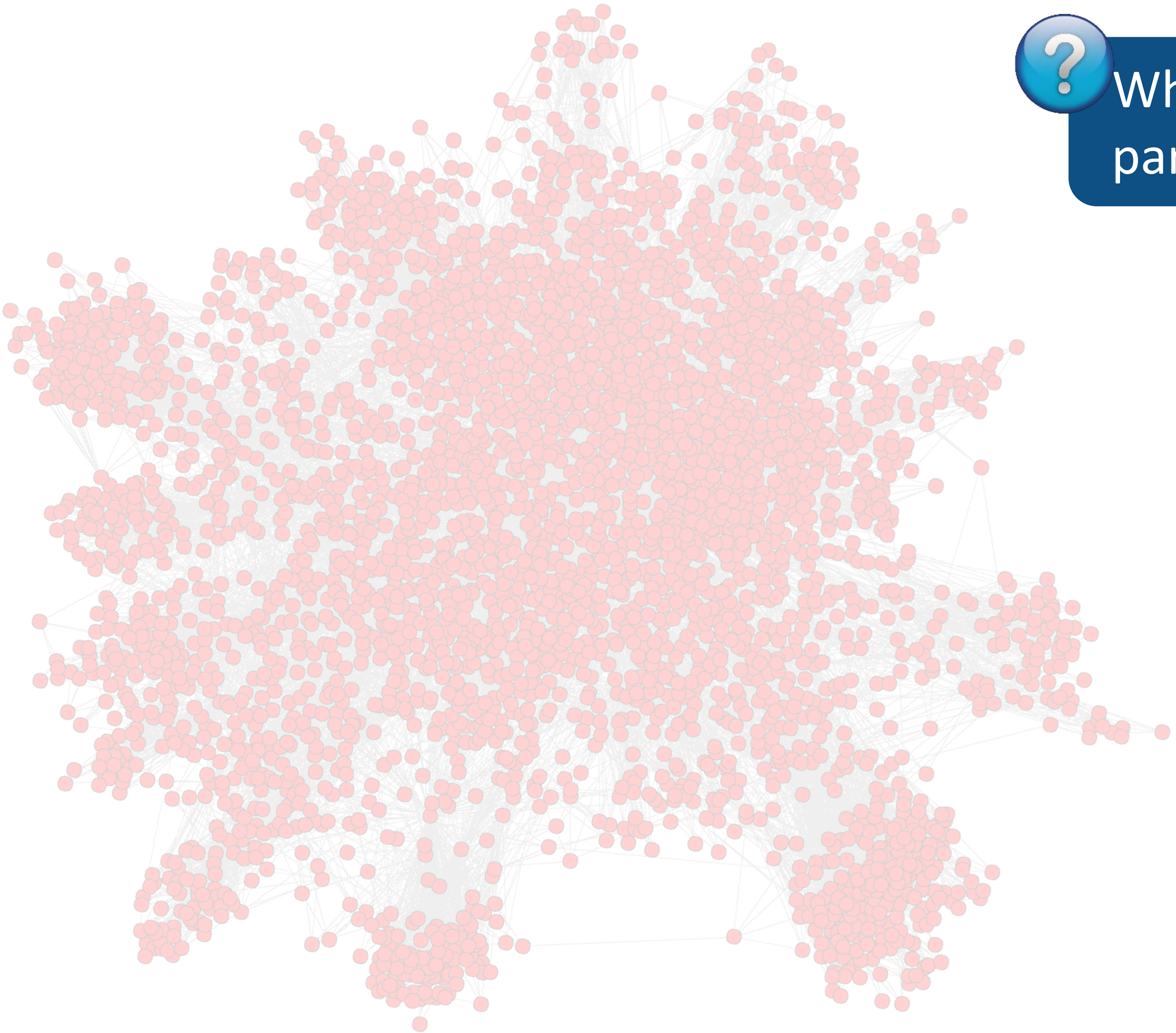
Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for

What programming paradigm and why?

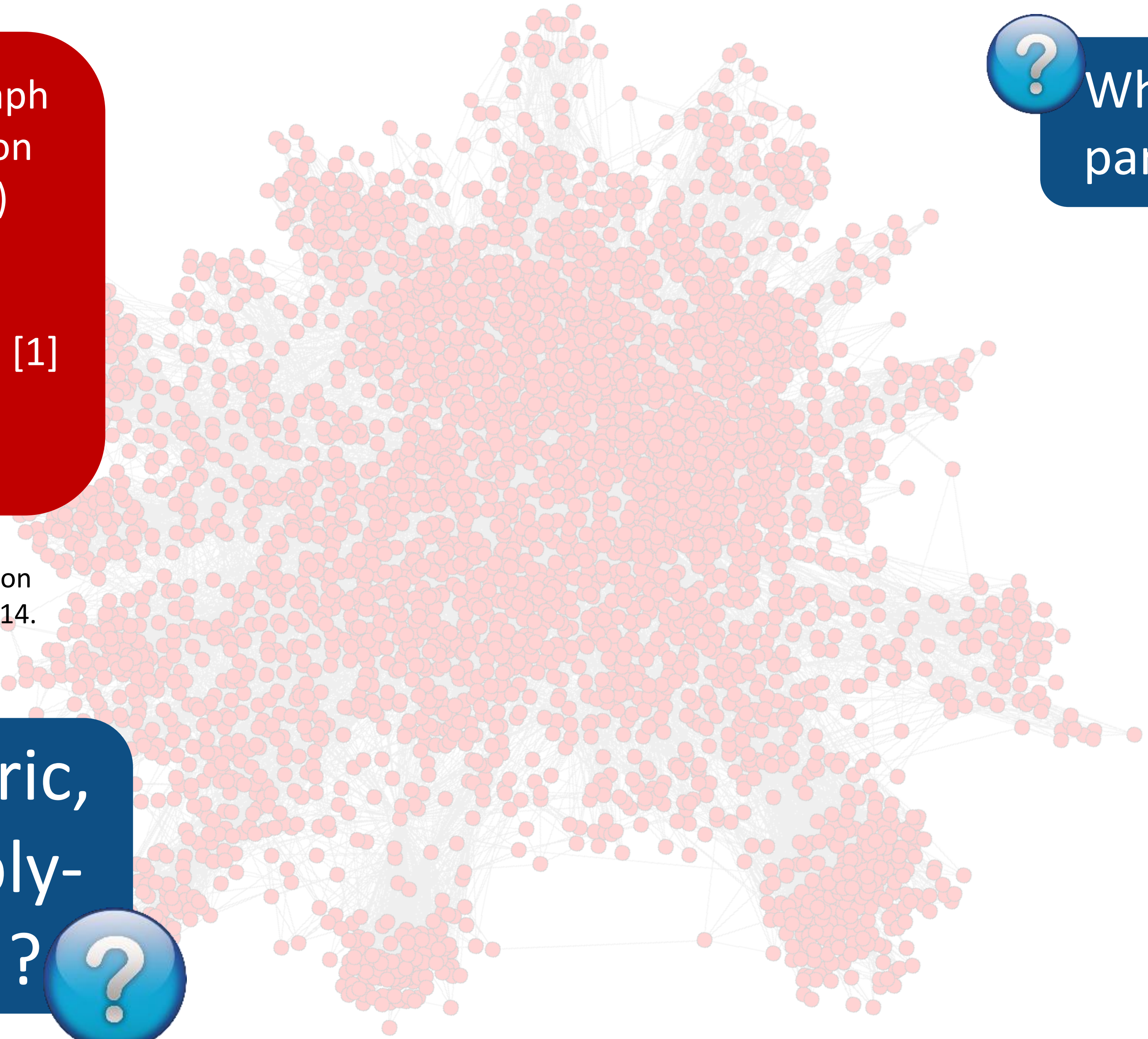
Key techniques, paradigms, challenges, features, ...

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms



What programming paradigm and why?



"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

+ other issues

[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.

What programming paradigm and why?

Vertex-centric, Gather-Apply-Scatter, ... ?

“(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations.” [1]

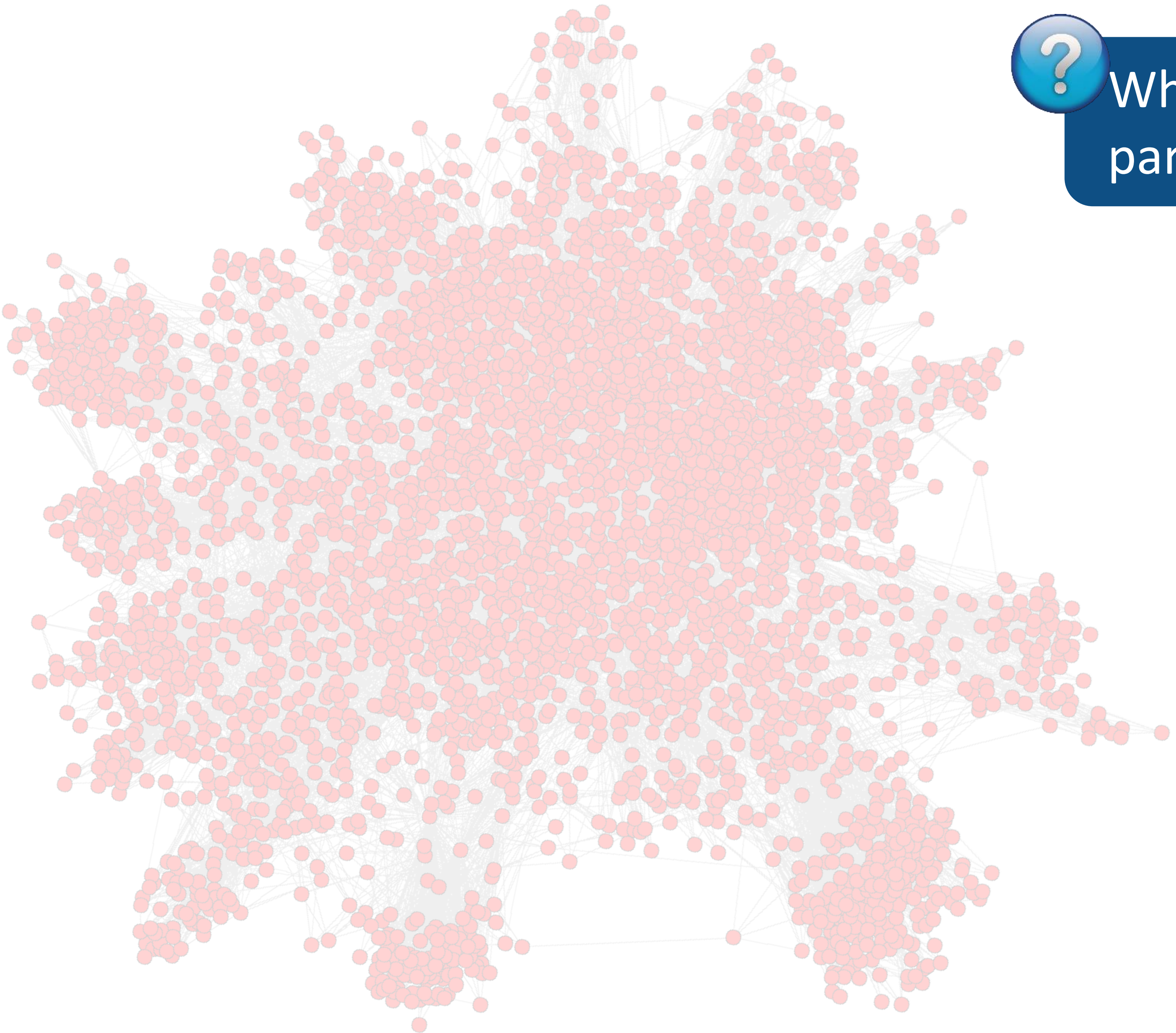
+ other issues

[1] S. Salihoglu and J. Widom, “Optimizing graph algorithms on Pregel-like systems”. VLDB. 2014.

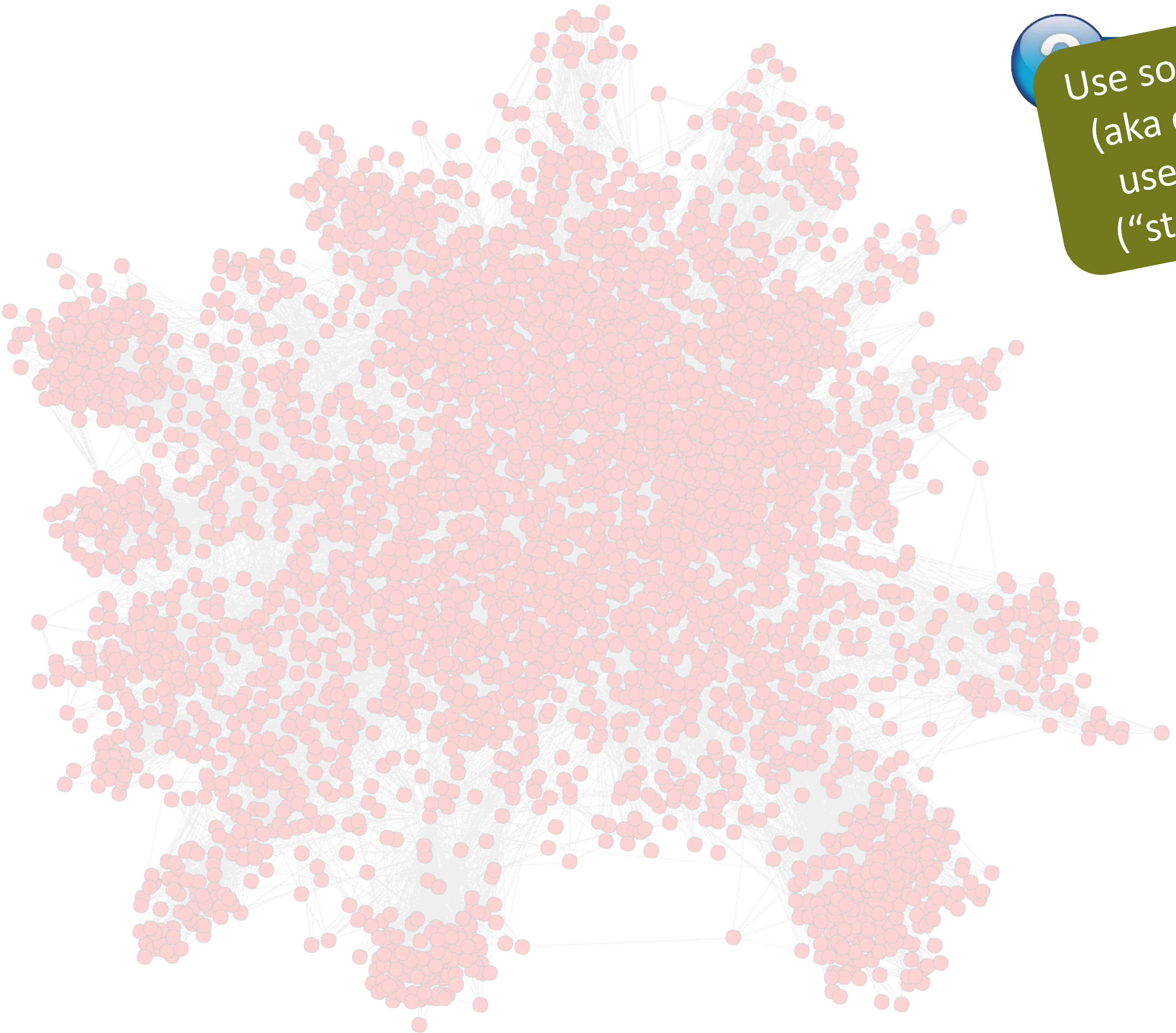
What programming paradigm and why?

To be able to **utilize pipelining well**, we really want to use **streaming** (aka edge-centric)

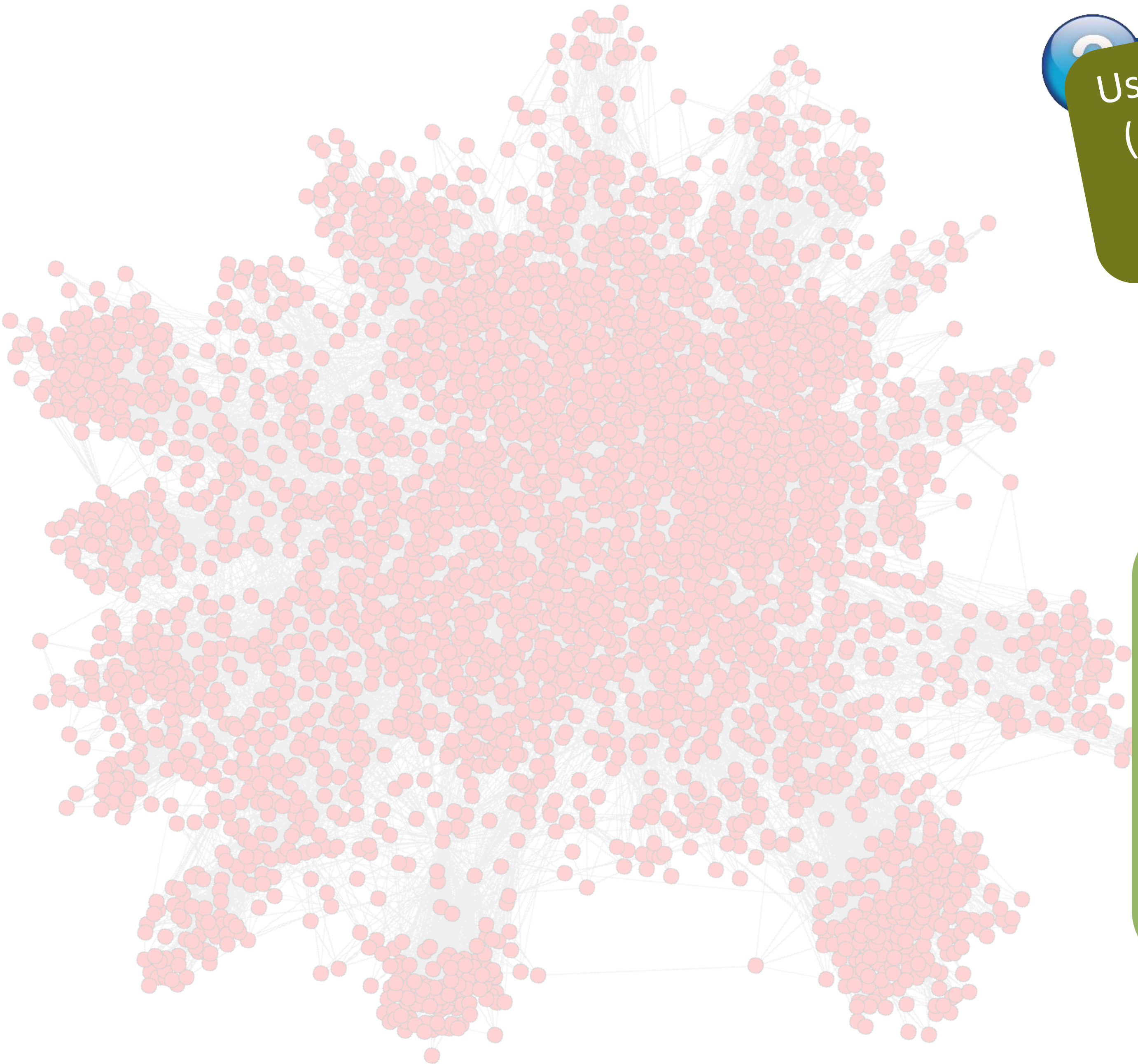
Vertex-centric, Gather-Apply-Scatter, ... ?



What programming paradigm and why?

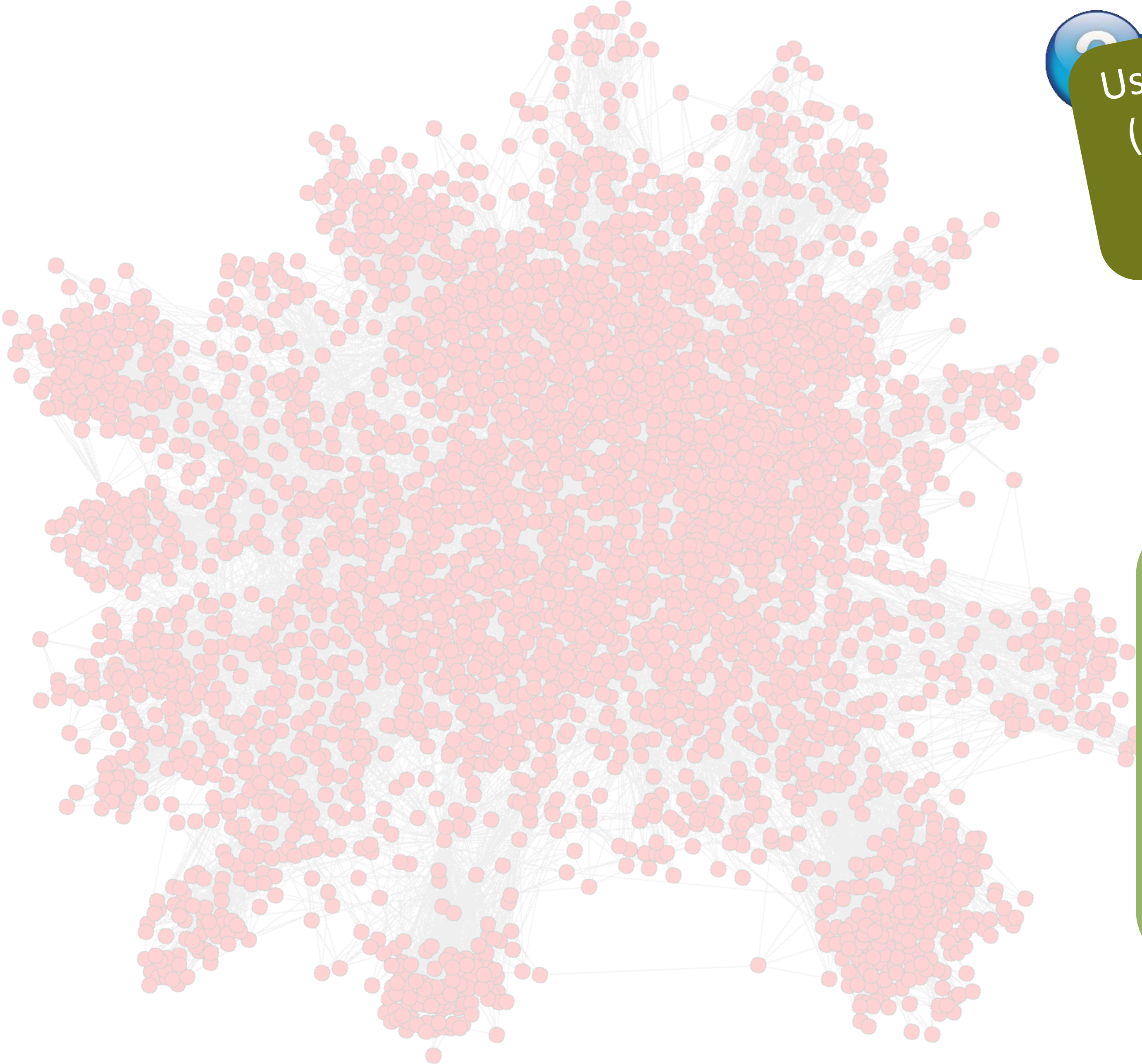


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

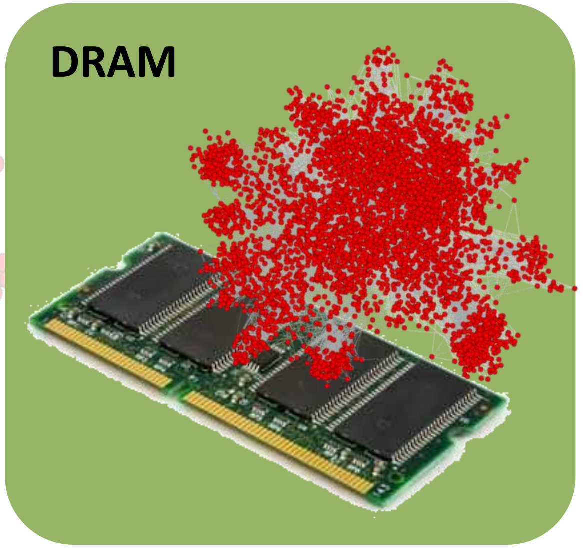


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

DRAM



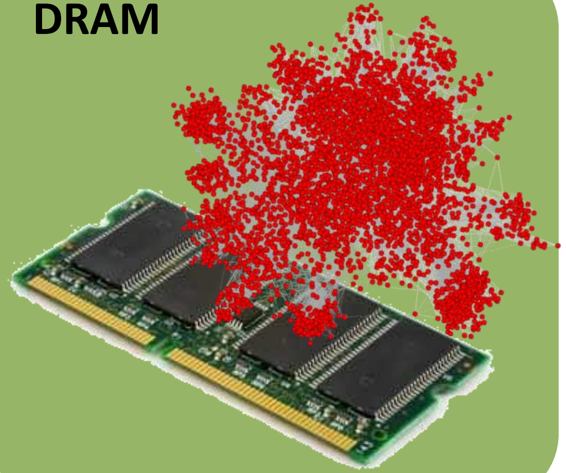
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")



Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

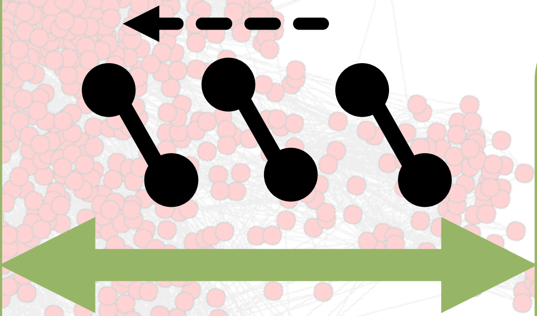
Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

DRAM

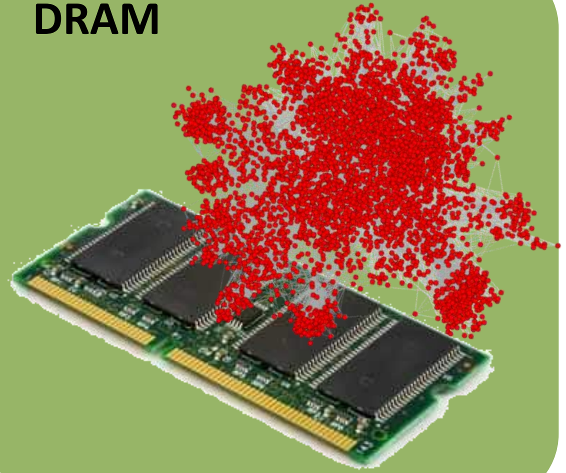


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

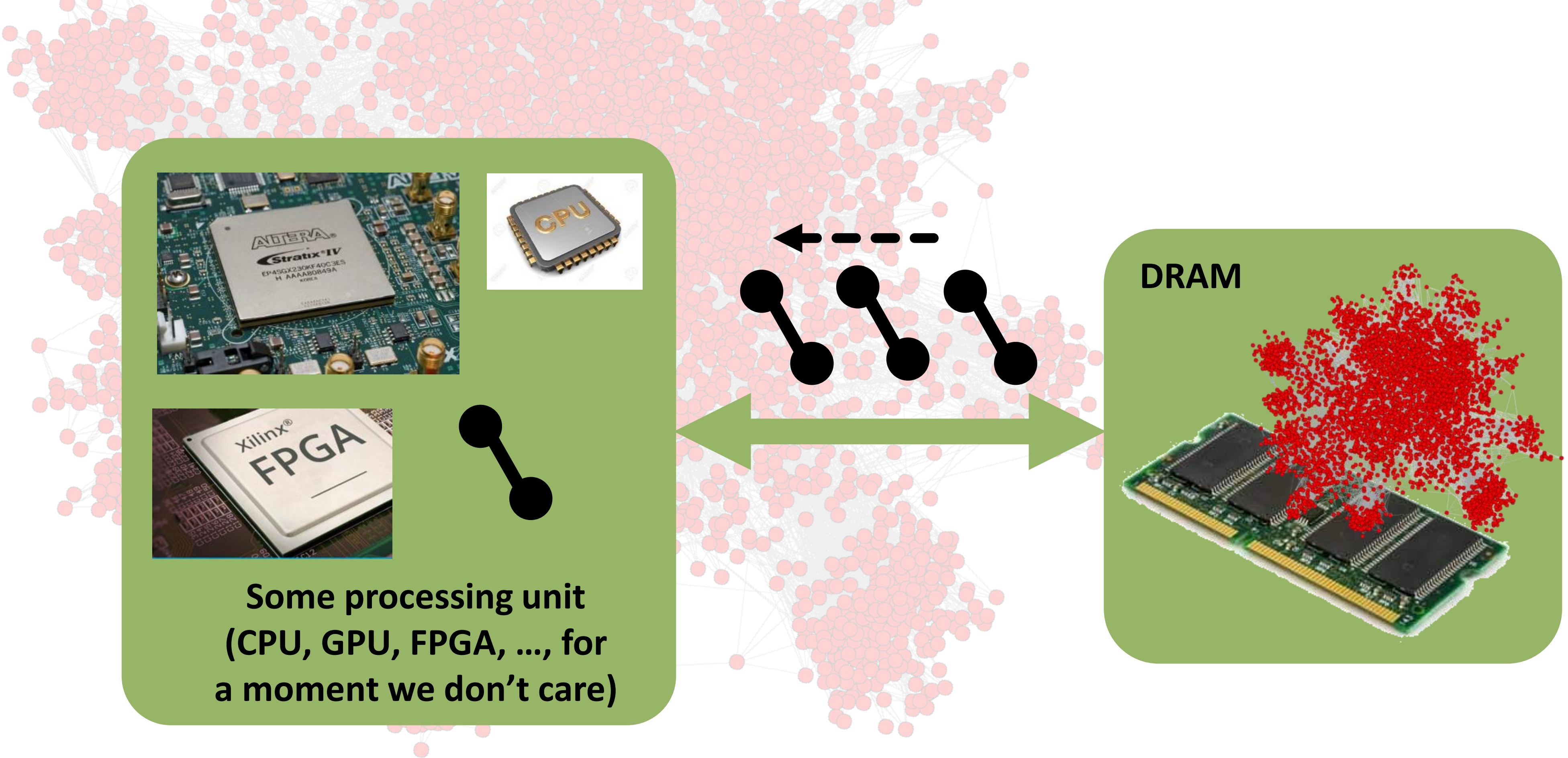
Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)



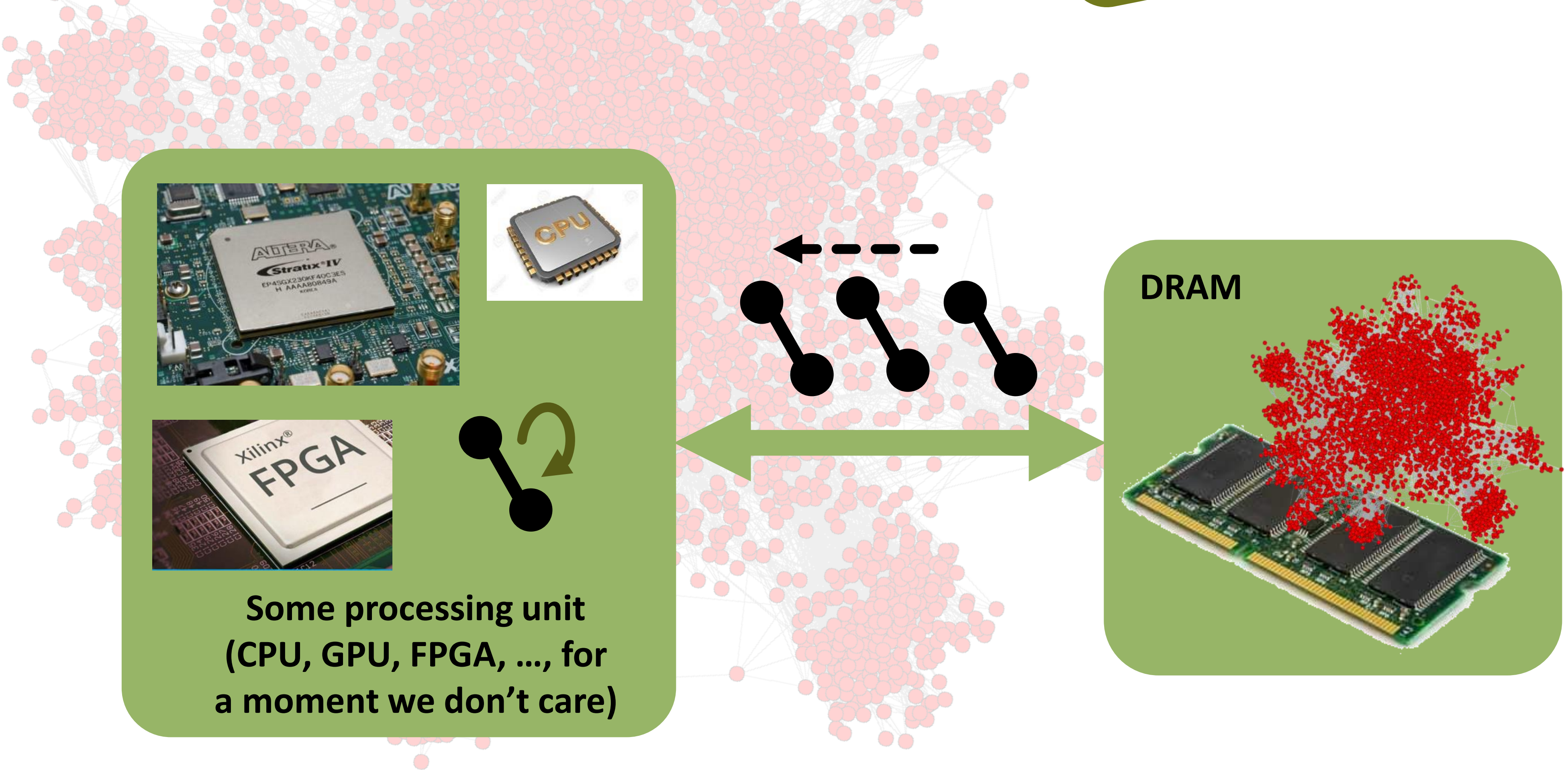
DRAM



Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

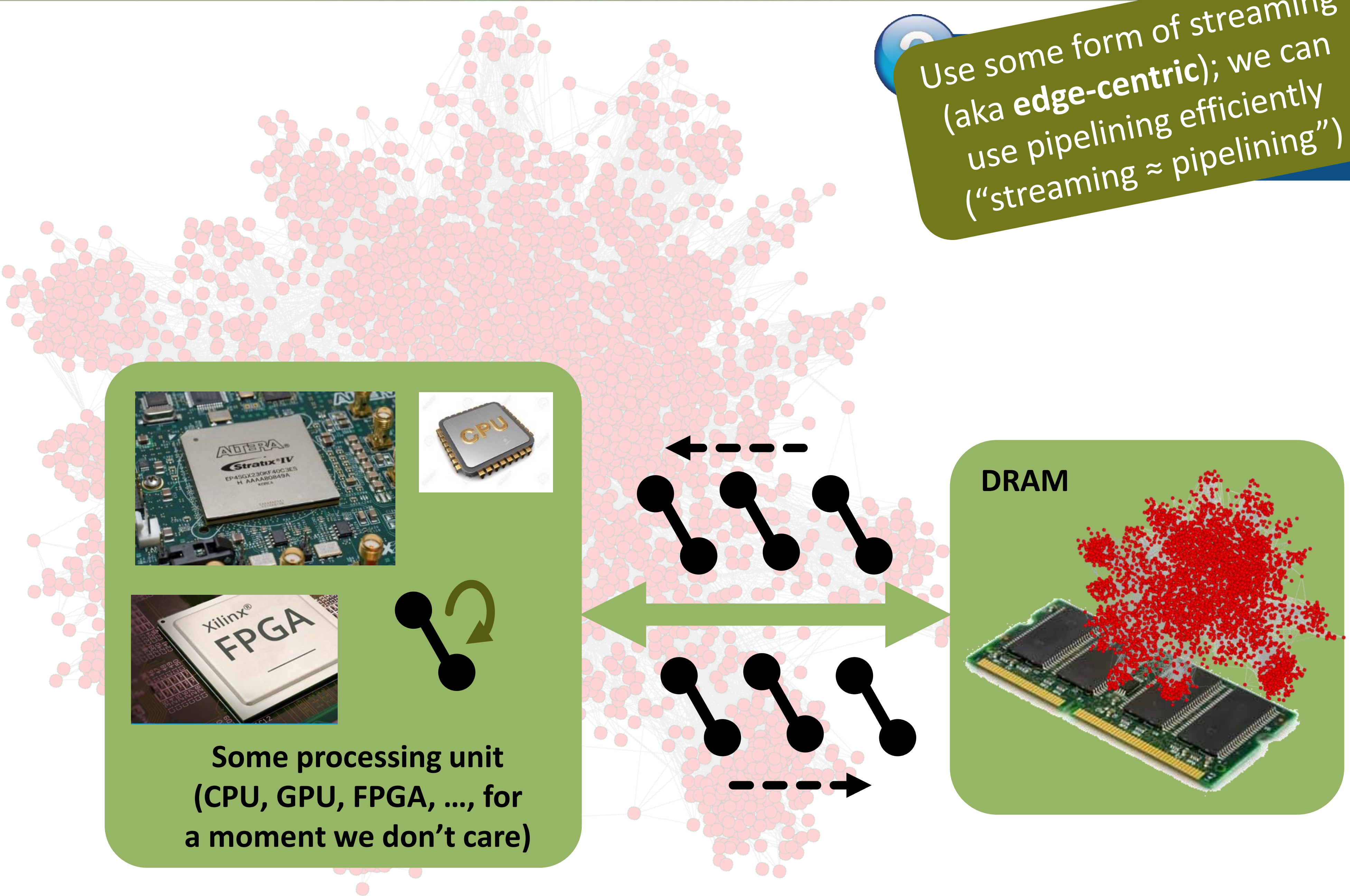


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")



Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

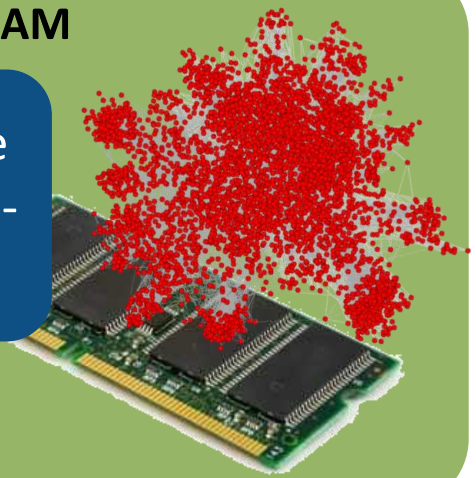


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

DRAM



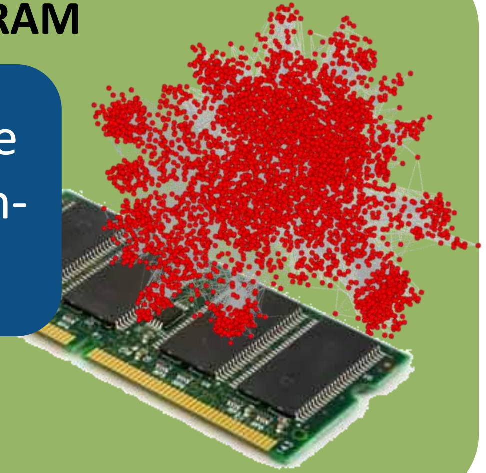
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

Issues…

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

DRAM



…How to minimize the number of "passes" over edges? This can get **really** bad in the "traditional" edge-centric approach (e.g., BFS needs *D* passes; *D* = diameter [1]).

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

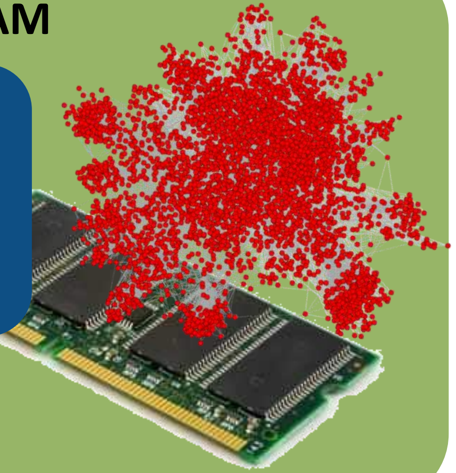
…Processing edges is sequential – **how to incorporate parallelism**?

Issues…

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

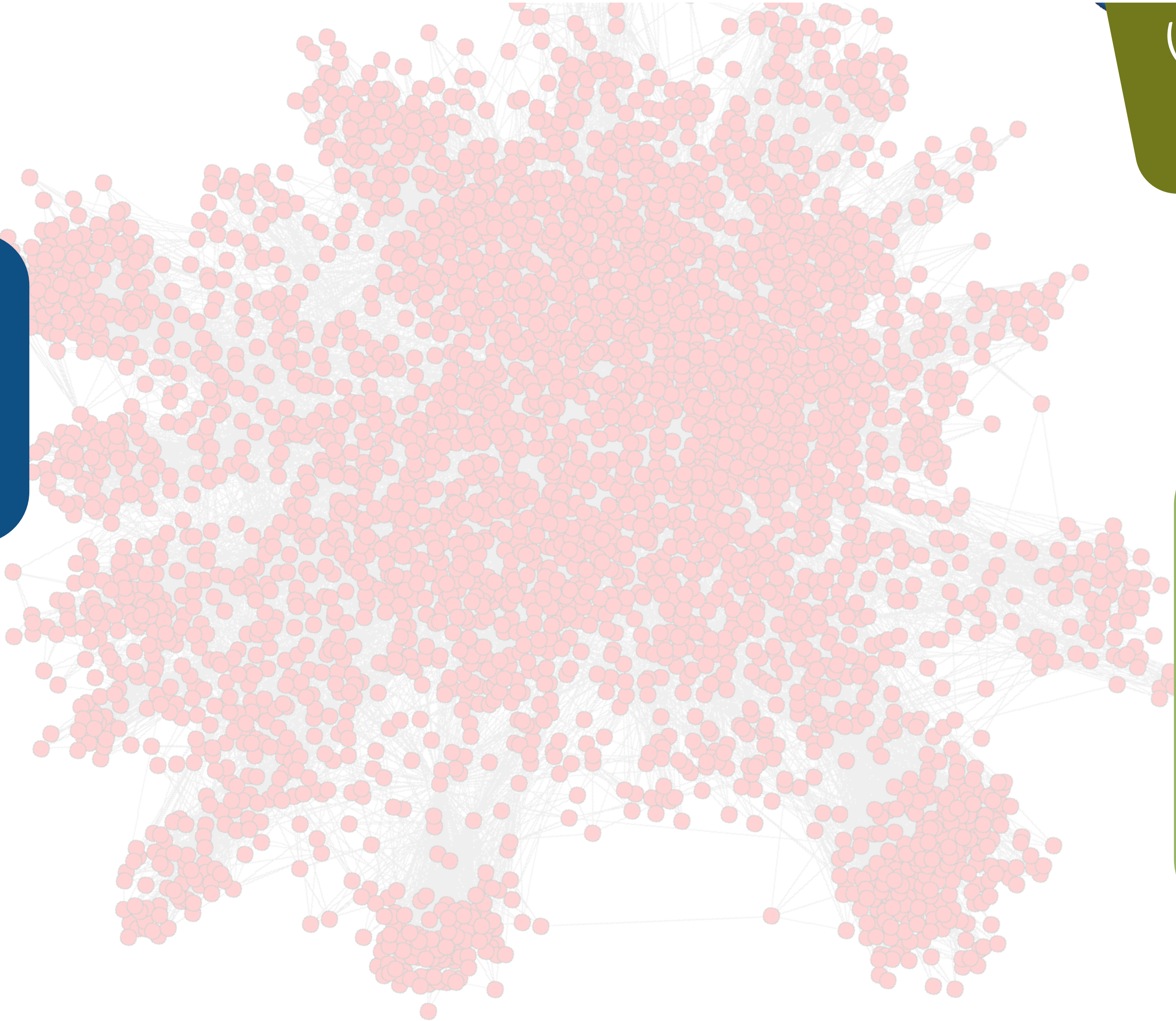
DRAM

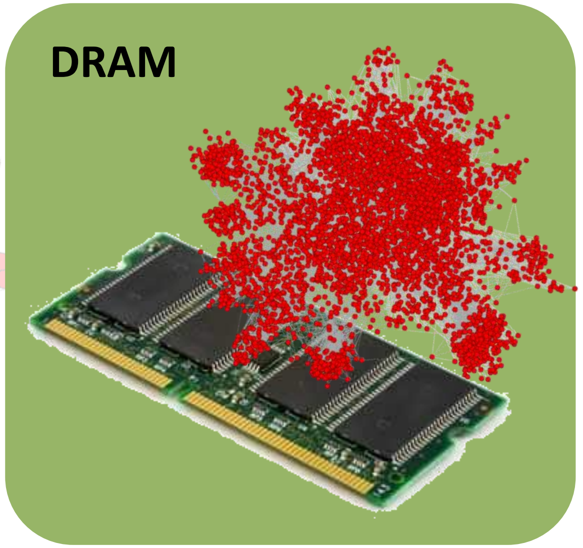


[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. SOSP. 2013.

(aka **edge-ce**
use pipelining efficiently
("streaming ≈ pipelining")

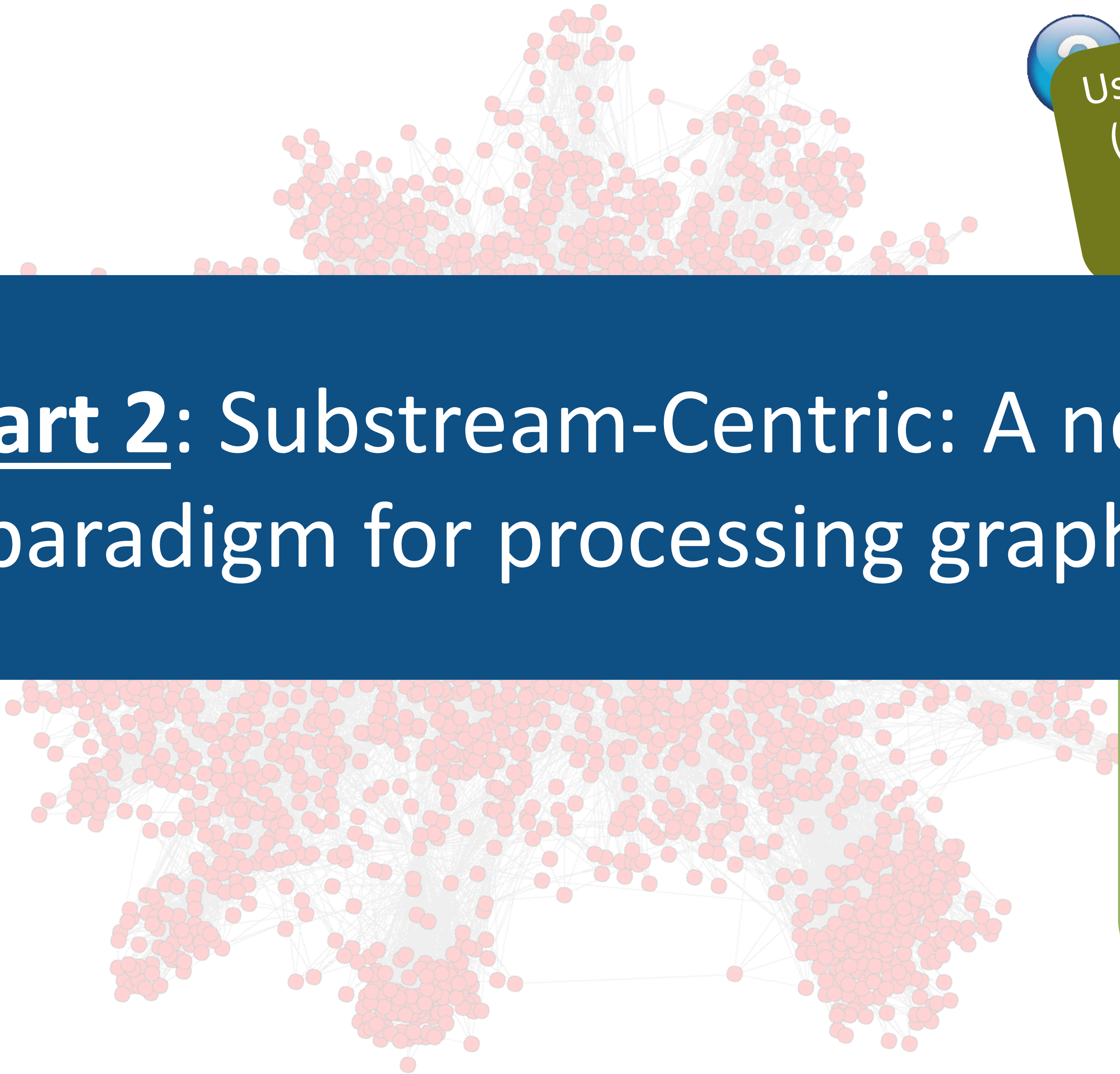
...Processing edges is sequential – **how to incorporate parallelism**?



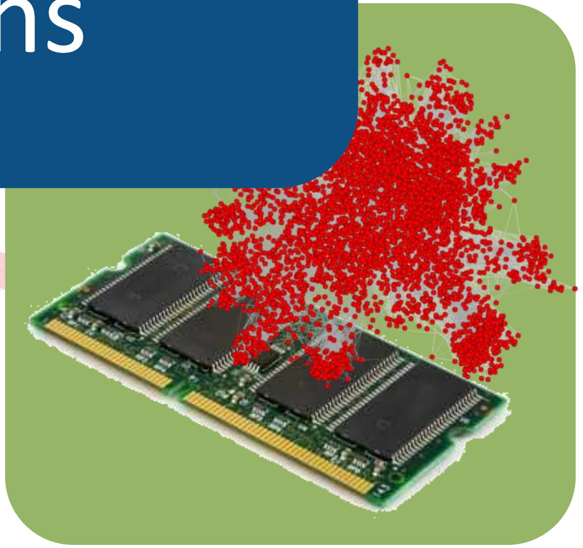


Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Pro
is seq
to i
pa

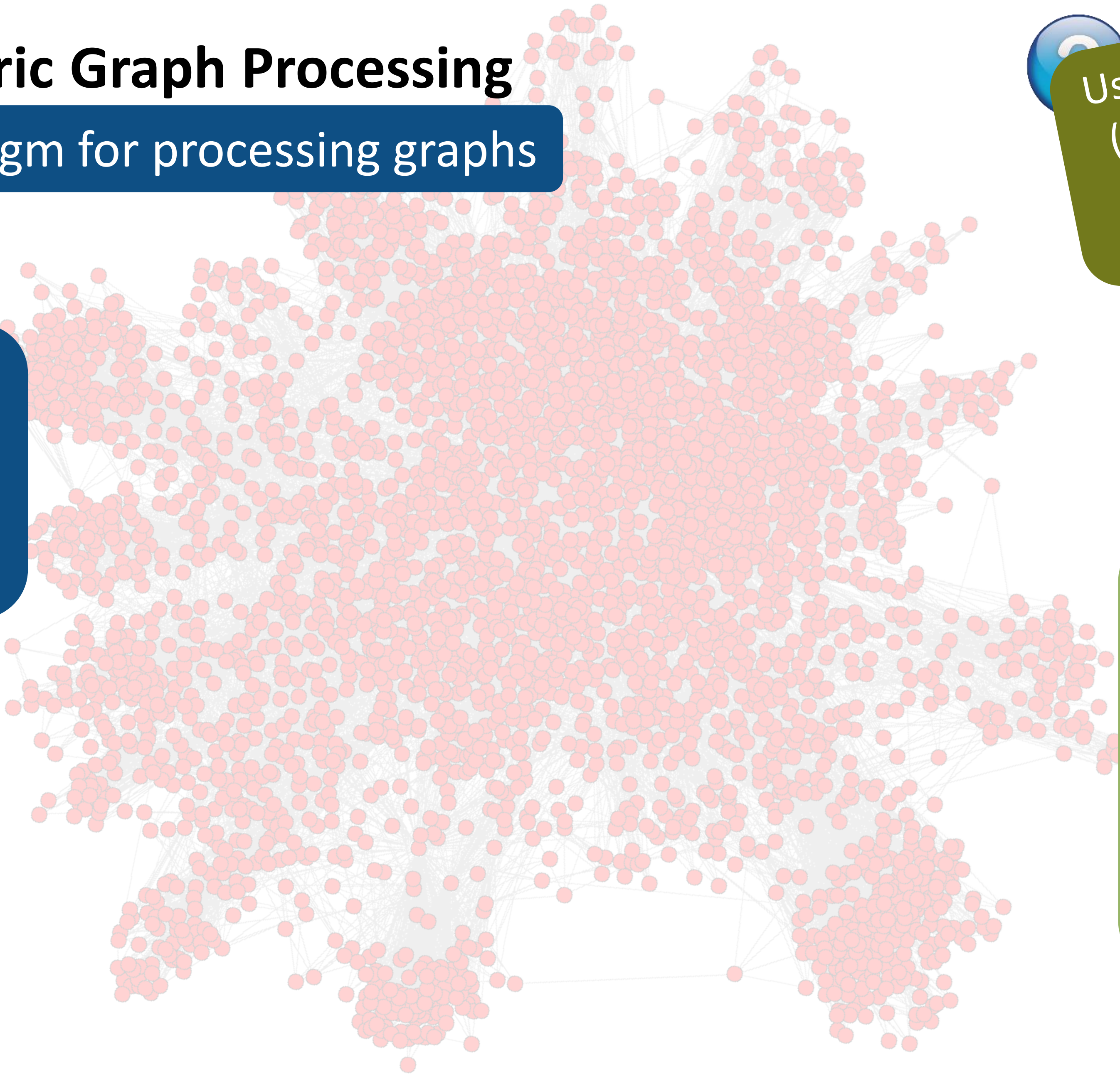


Part 2: Substream-Centric: A new paradigm for processing graphs



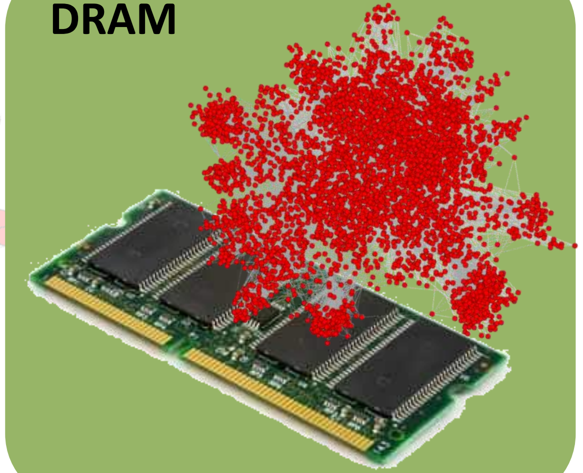
Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs



…Processing edges is sequential – **how to incorporate parallelism**?

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

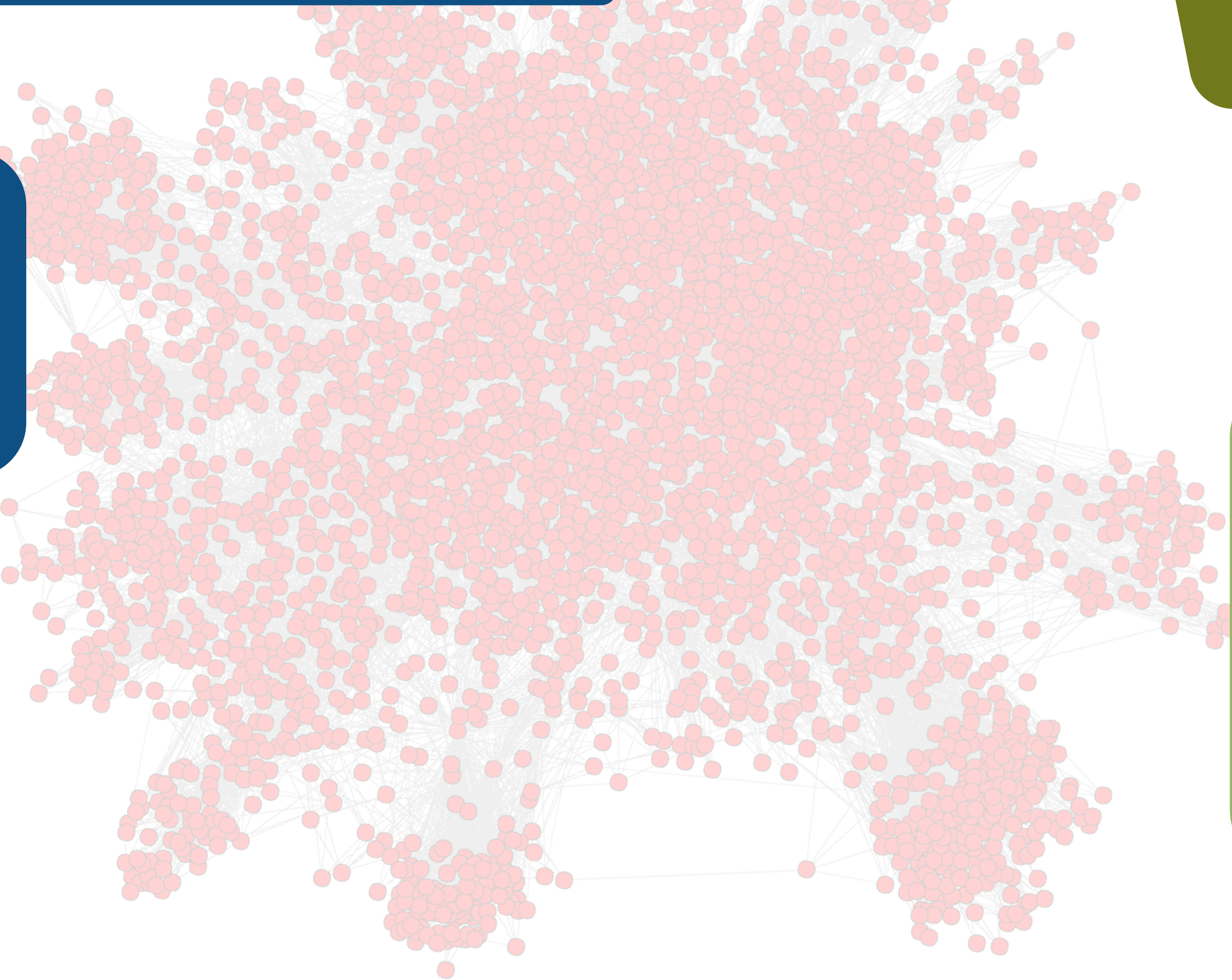


Substream-Centric Graph Processing

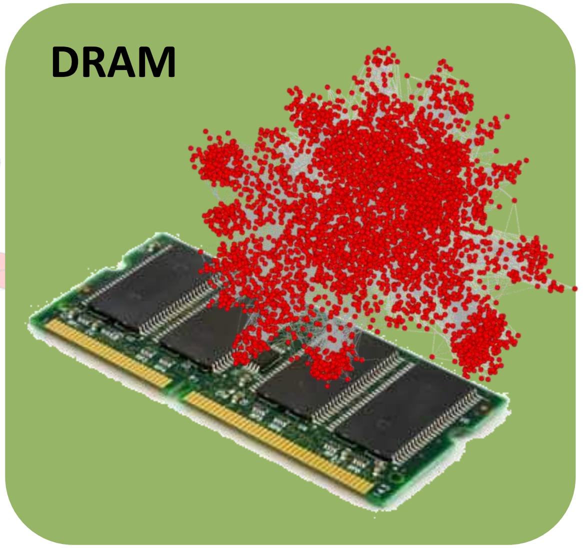
Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)



…Processing edges is sequential – **how to incorporate parallelism**?



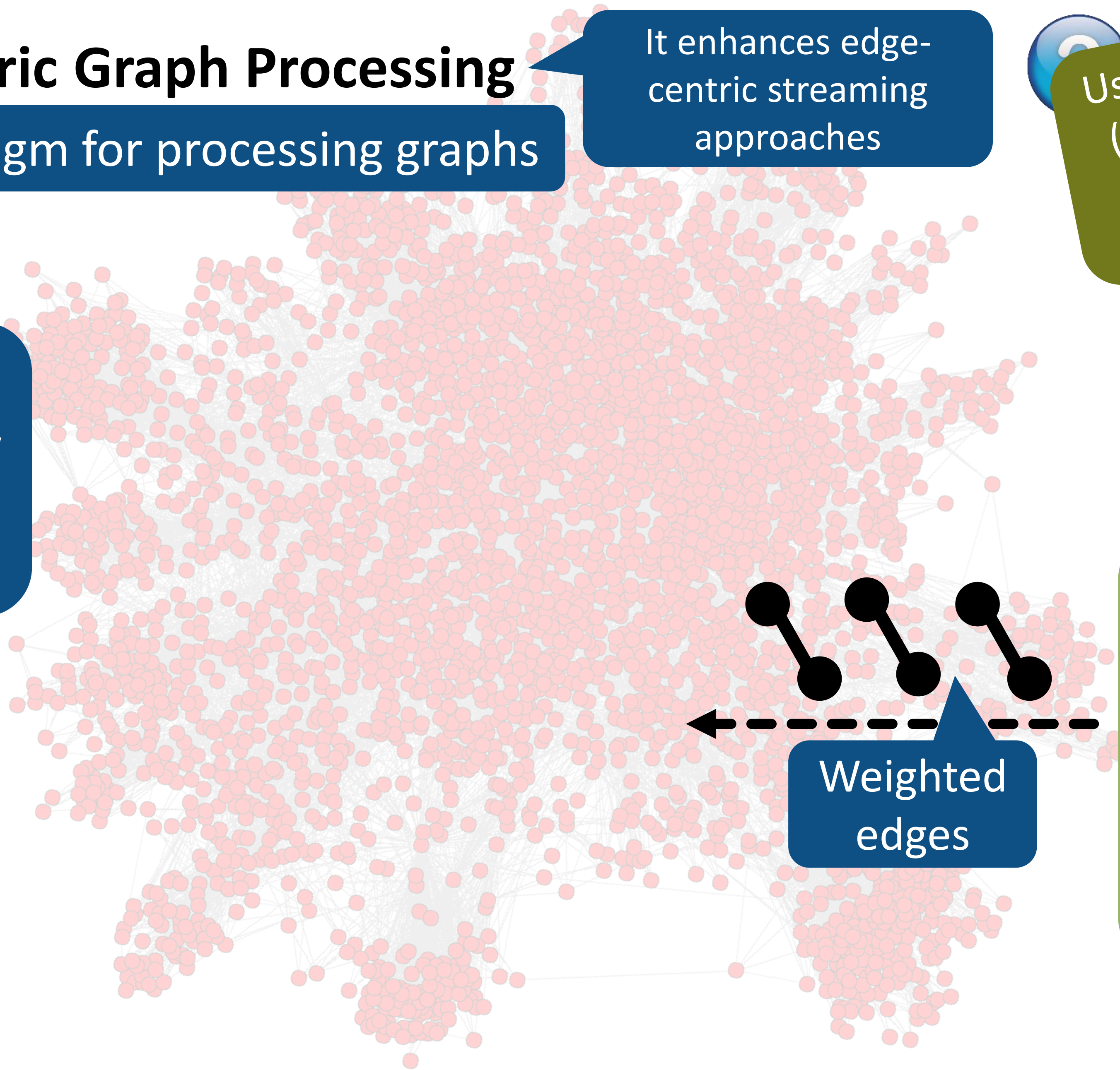
Substream-Centric Graph Processing

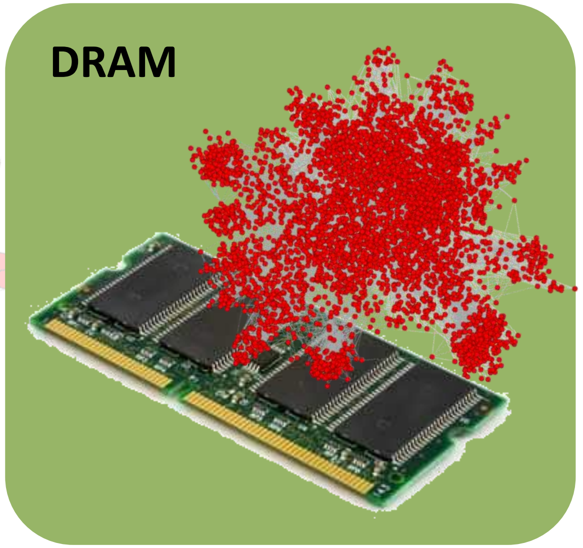
Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?





Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming approaches

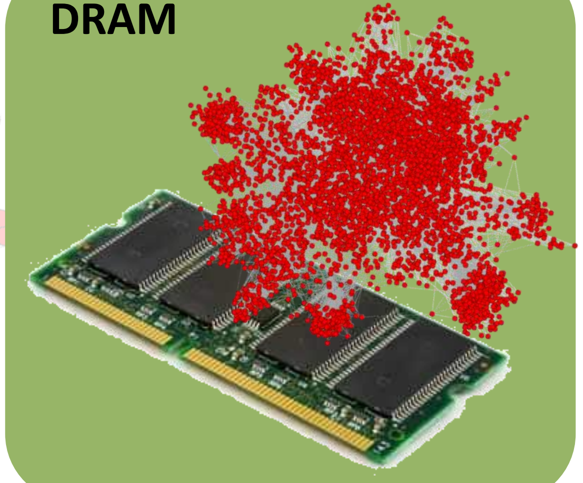
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Divide the input stream of edges according to some (algorithm-specific) pattern

Weighted edges

DRAM



Substream-Centric Graph Processing

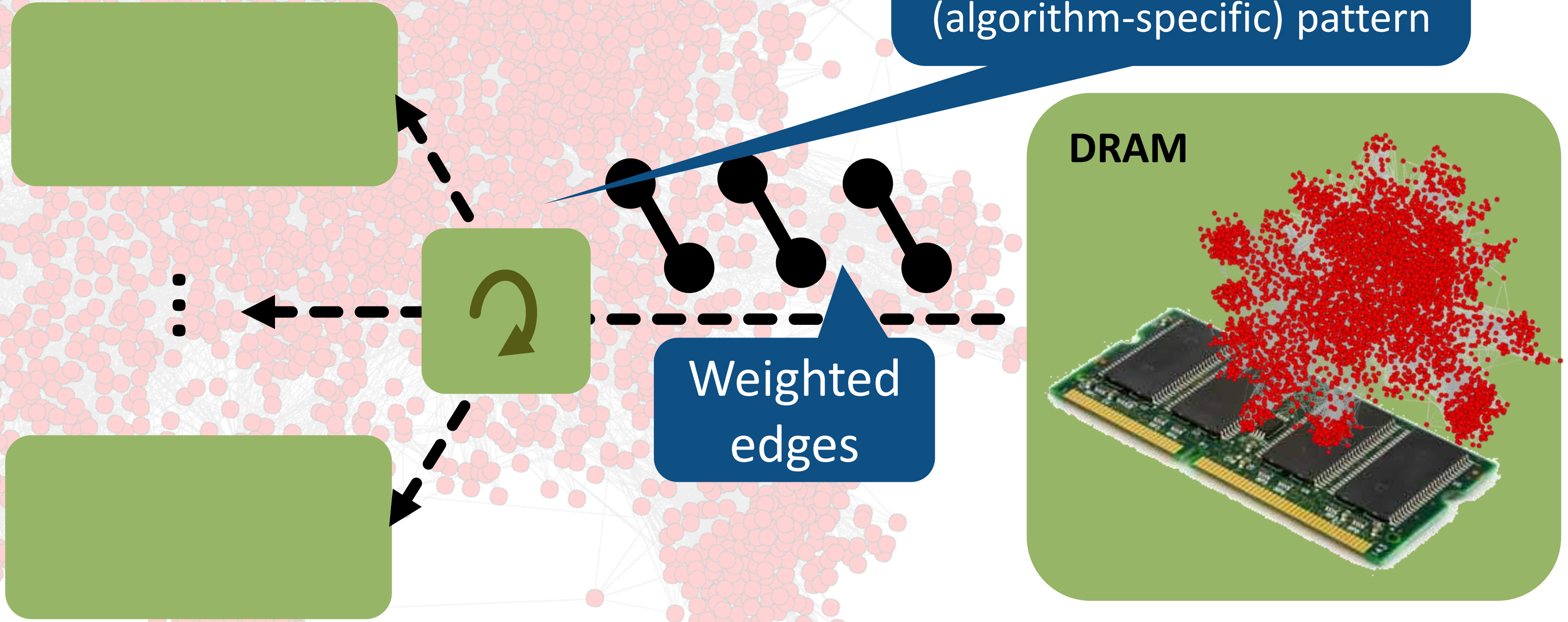
Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

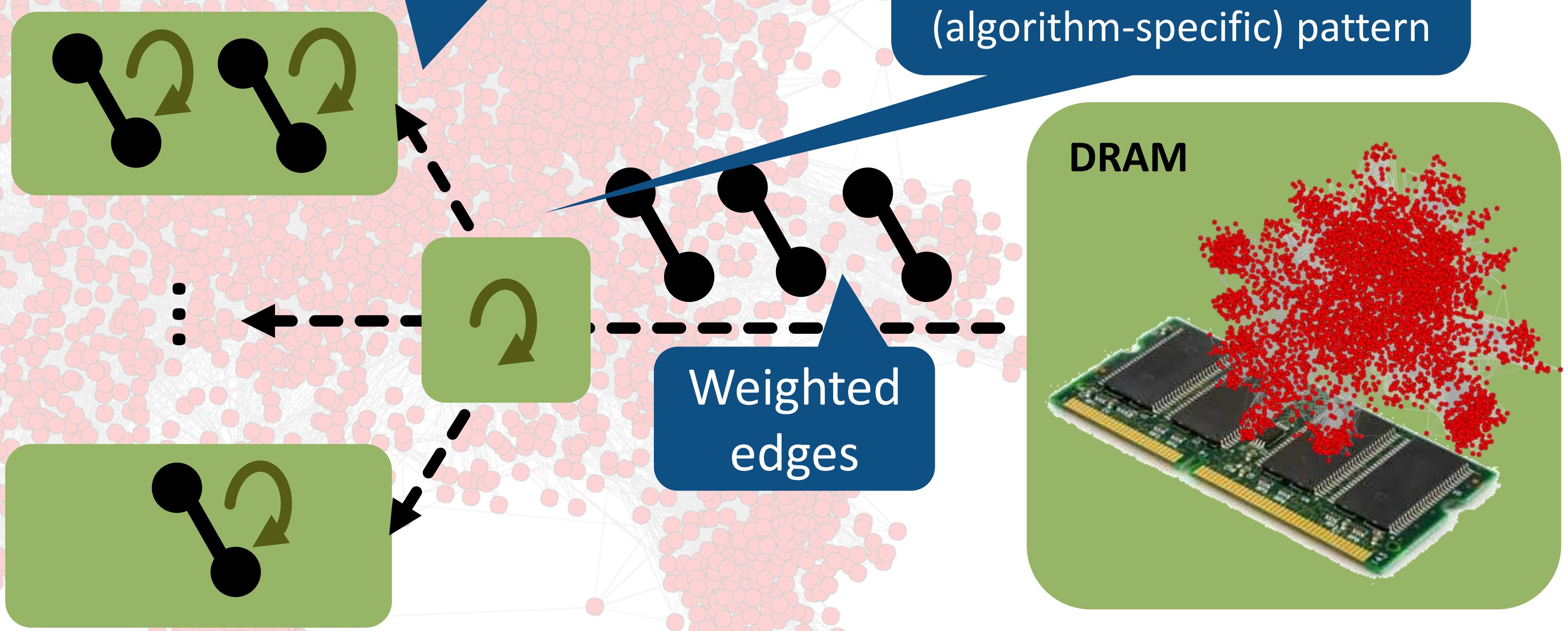
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Process "substreams" independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

It enhances edge-centric streaming approaches

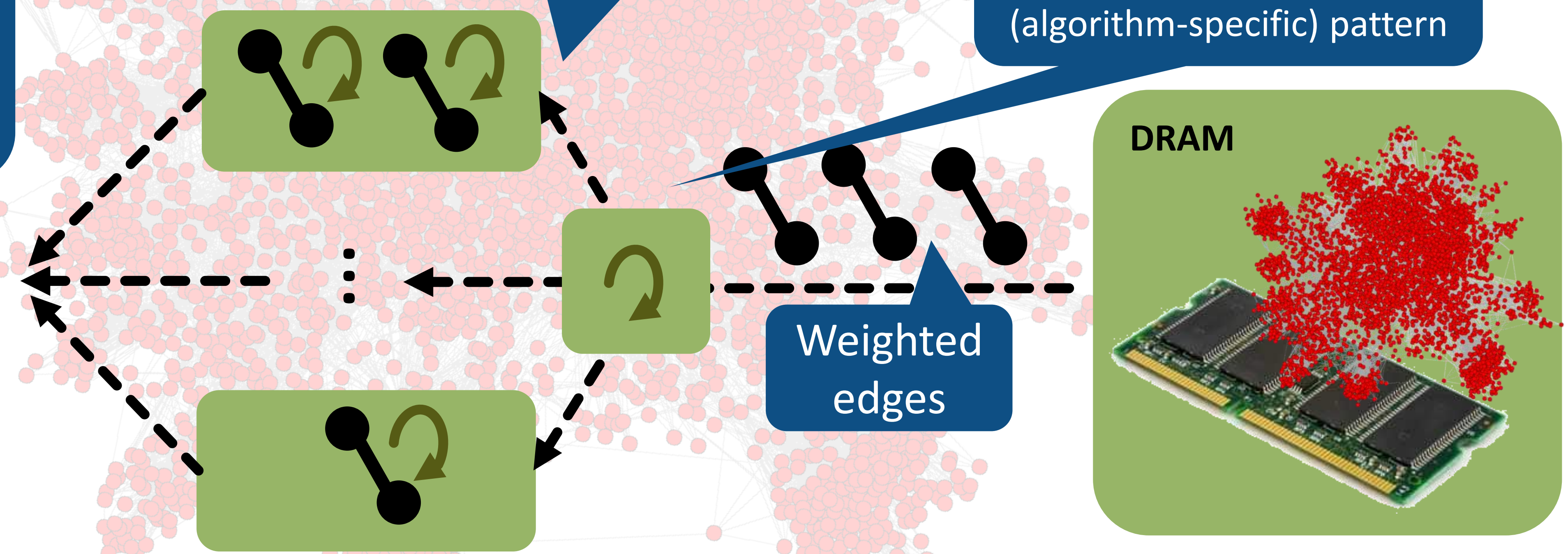
Part 2: A new paradigm for processing graphs

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

...Processing edges is sequential – **how to incorporate parallelism**?

Process "substreams" independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

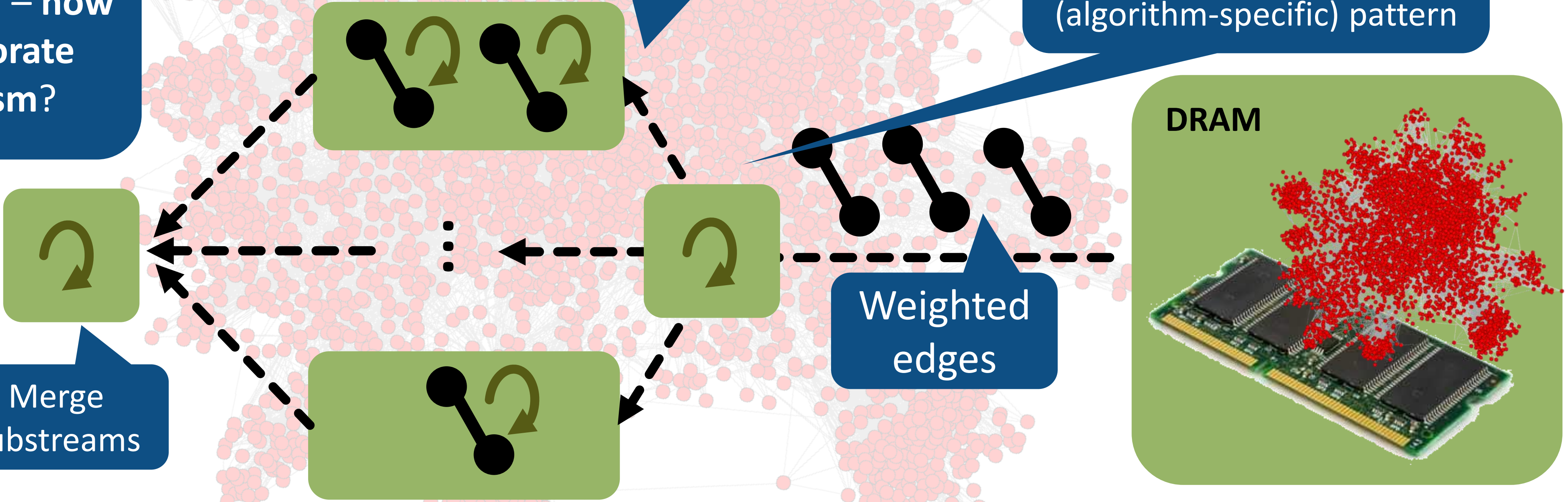
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Process "substreams" independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Weighted edges

Merge substreams

Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

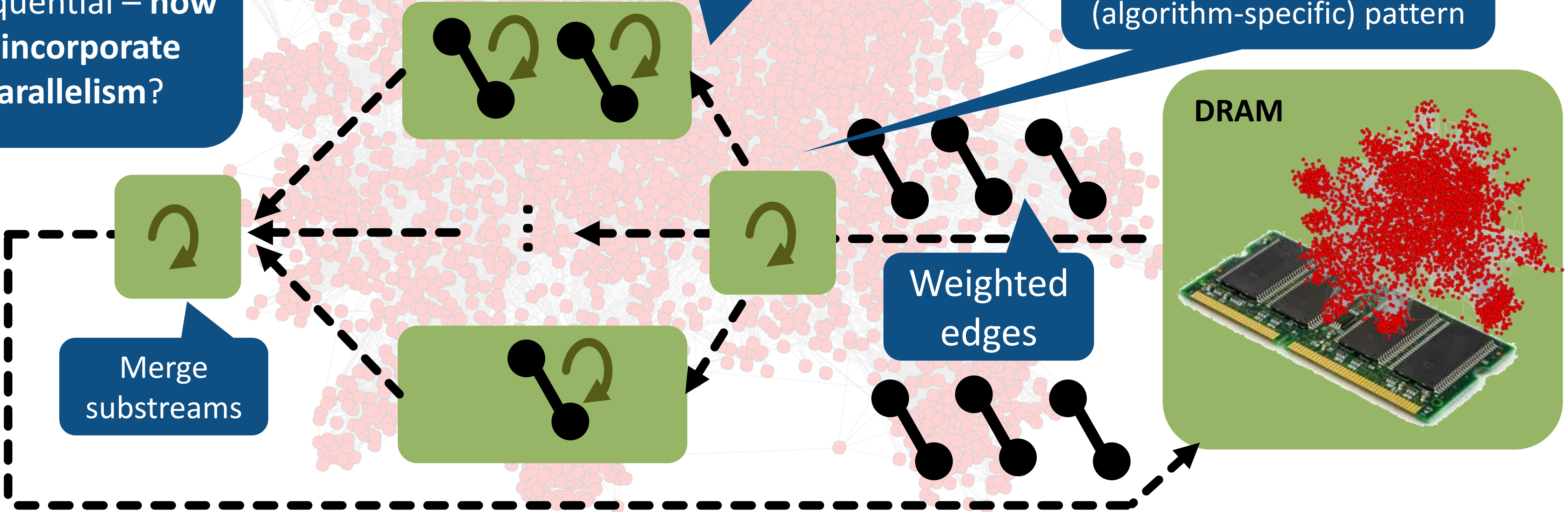
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

…Processing edges is sequential – **how to incorporate parallelism**?

Process “substreams” independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

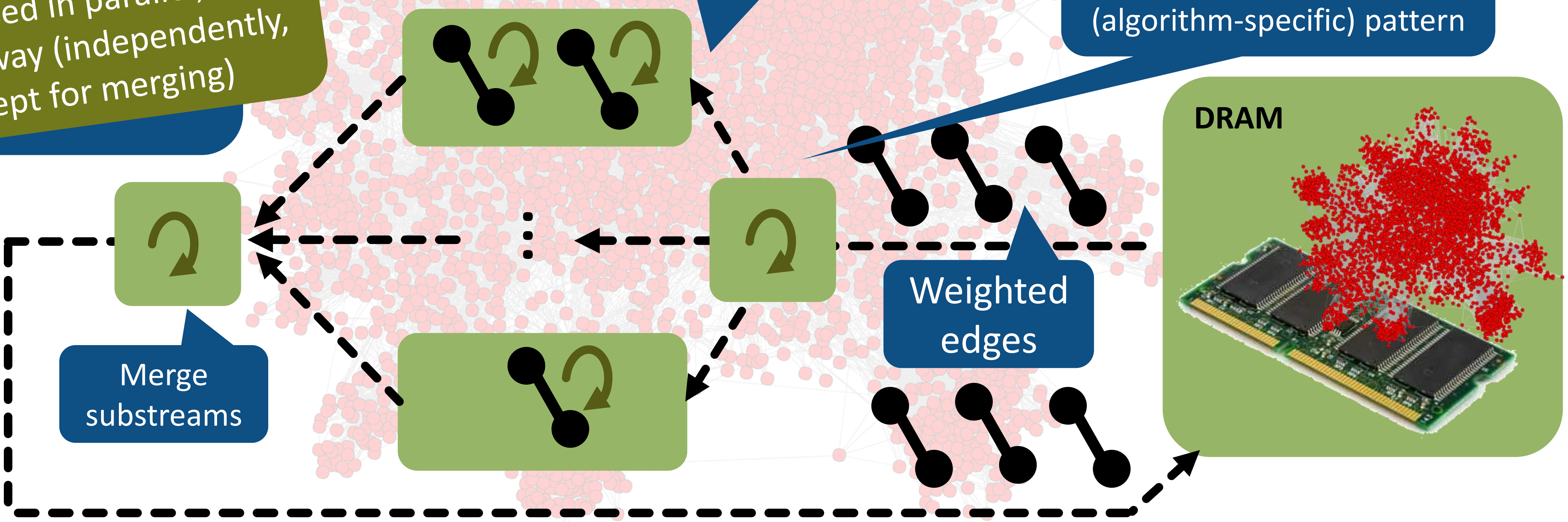
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Process “substreams” independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming
…aches

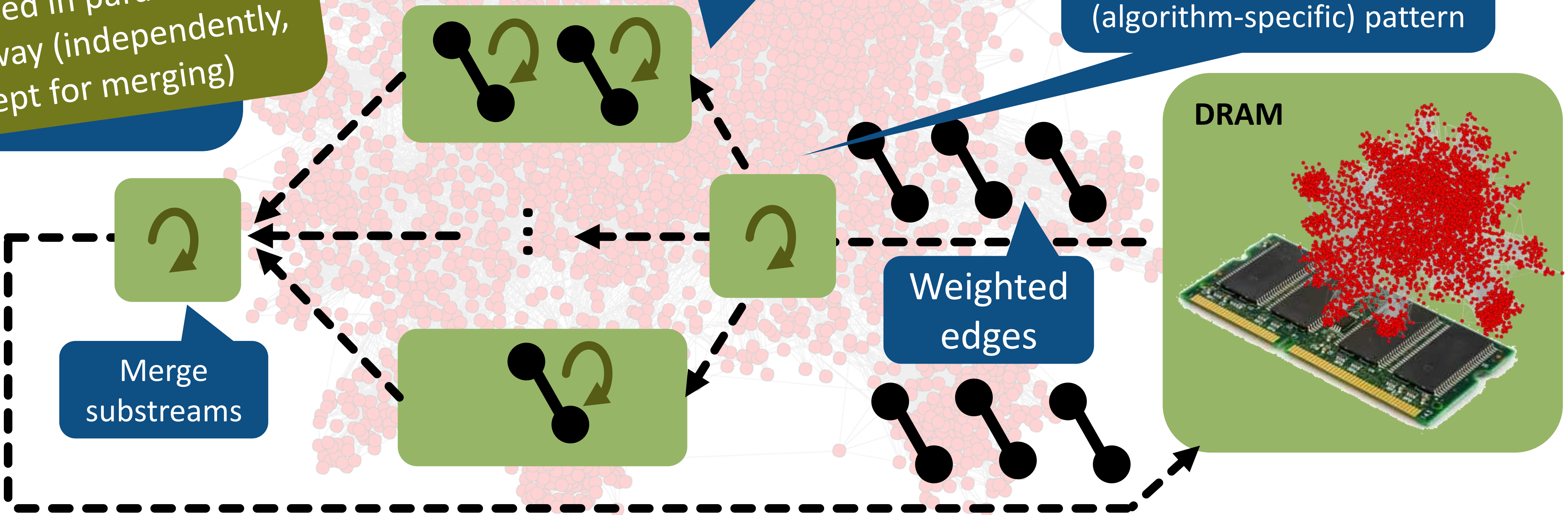
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

Also, it enables (tunable) approximation and a (tunable) number of passes

…treams"
independently

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edge-centric streaming
aches

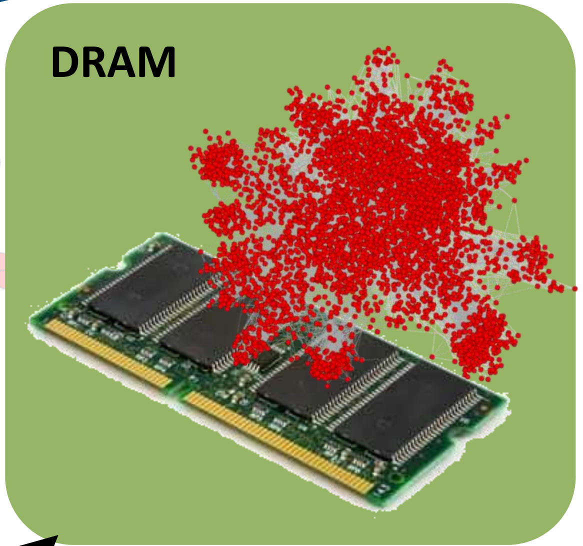
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

Also, it enables (tunable) approximation and a (tunable) number of passes

treams”
independently

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Divide the input stream of edges according to some (algorithm-specific) pattern



ghted
ges

How to express MWM in this paradigm?

Merge substreams

Research Questions

Which programming paradigm to use for (approximate) MWM (and many other problems)?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism, enables easy pipelining, supports approximation)

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

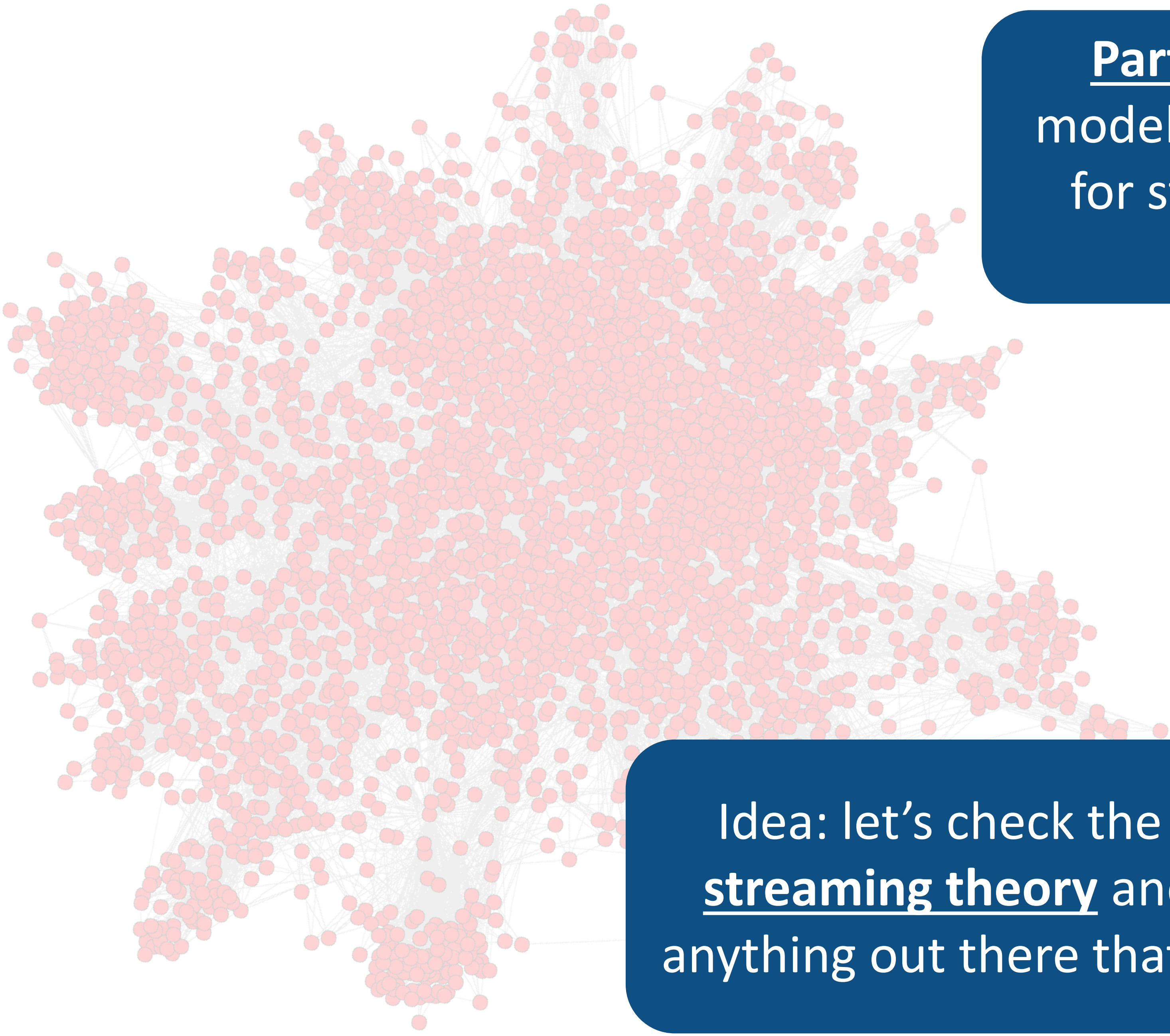
Use **substream-centric processing** (exposes parallelism, enables easy pipelining, supports approximation)

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

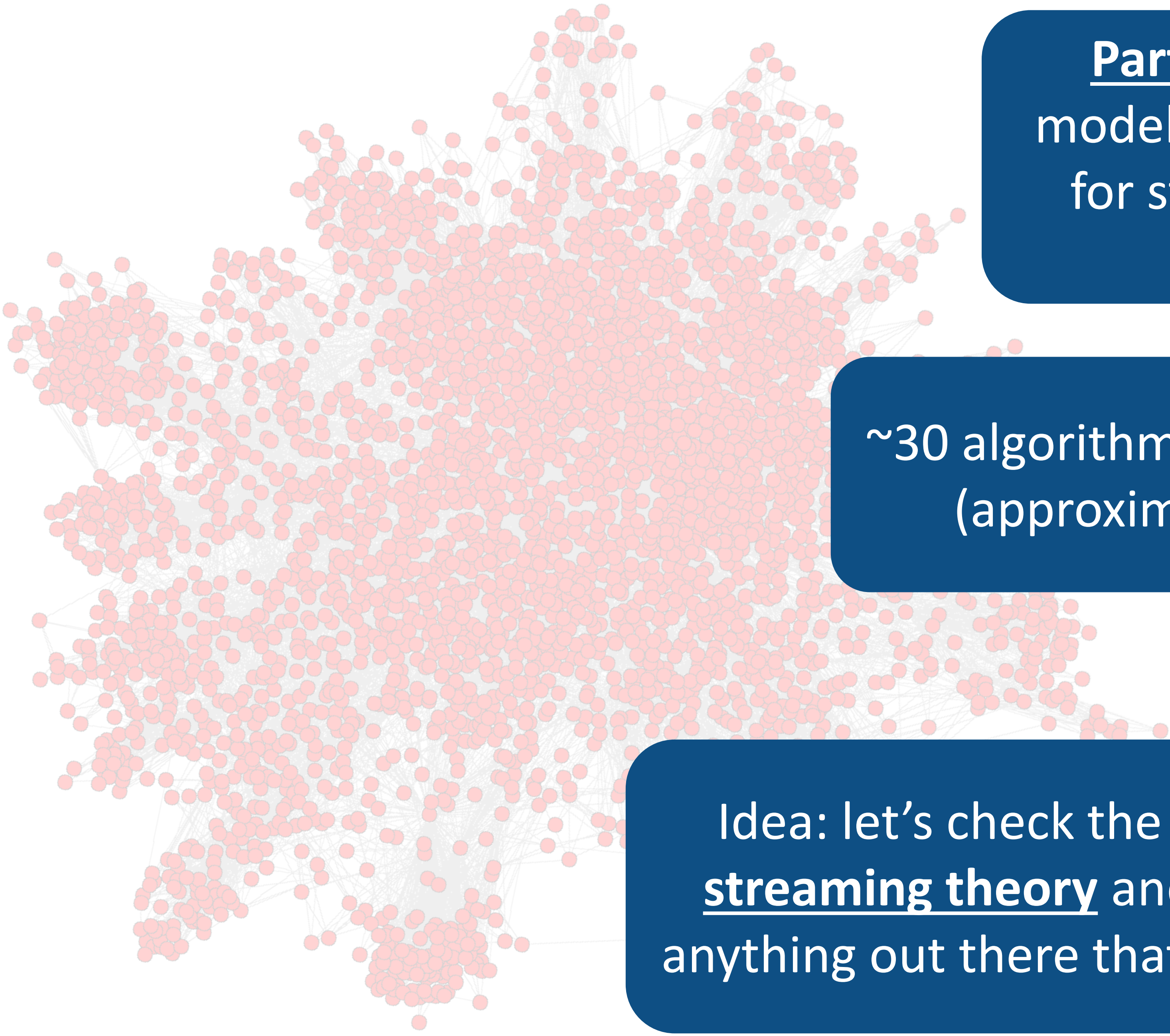
What is the ultimate performance, power consumption, and the related tradeoffs?

<u>Part 3</u>: Analysis of models and algorithms for streaming graph processing



Part 3: Analysis of models and algorithms for streaming graph processing

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

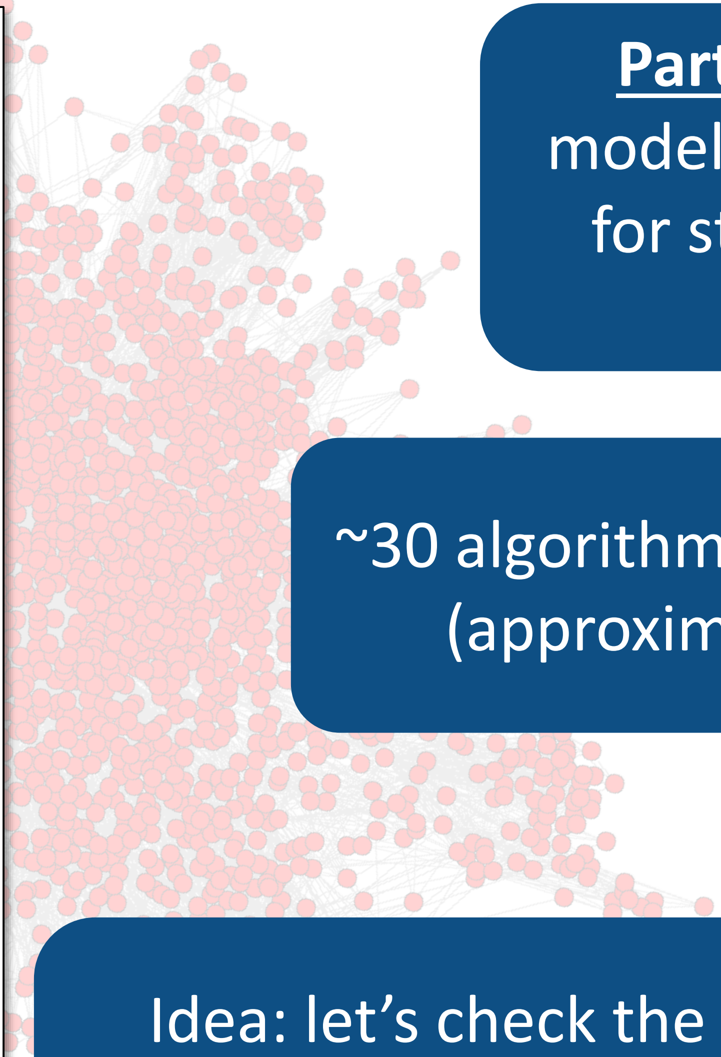


<u>Part 3</u>: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **<u>streaming theory</u>** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[26]	$1/2$	$O(n)$	1	👎	👍	👍
[41, Theorem 6]	$1/2+0.0071$	$O(n\,\mathrm{polylog}(n))$	2	👎	👍	👍
[41, Theorem 2]	$1/2+0.003^*$	$O(n\,\mathrm{polylog}(n))$	1	👎	👍	👍
[36, Theorem 1.1]	$O(\mathrm{polylog}(n))$	$O(\mathrm{polylog}(n))$	1	👎	👍	👍
[26, Theorem 1]	$2/3-\varepsilon$	$O(n\log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	$1-\varepsilon$	$O\left(n\,\mathrm{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[41, Theorem 5]	$1/2+0.019$	$O(n\,\mathrm{polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2+0.005^*$	$O(n\log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2+0.0071^*$	$O(n\,\mathrm{polylog}(n))$	2	👎	👎	👍
[39]	$1-1/e$	$O(n\,\mathrm{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1-\frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n\log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n\,\mathrm{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n\log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n\log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	👍



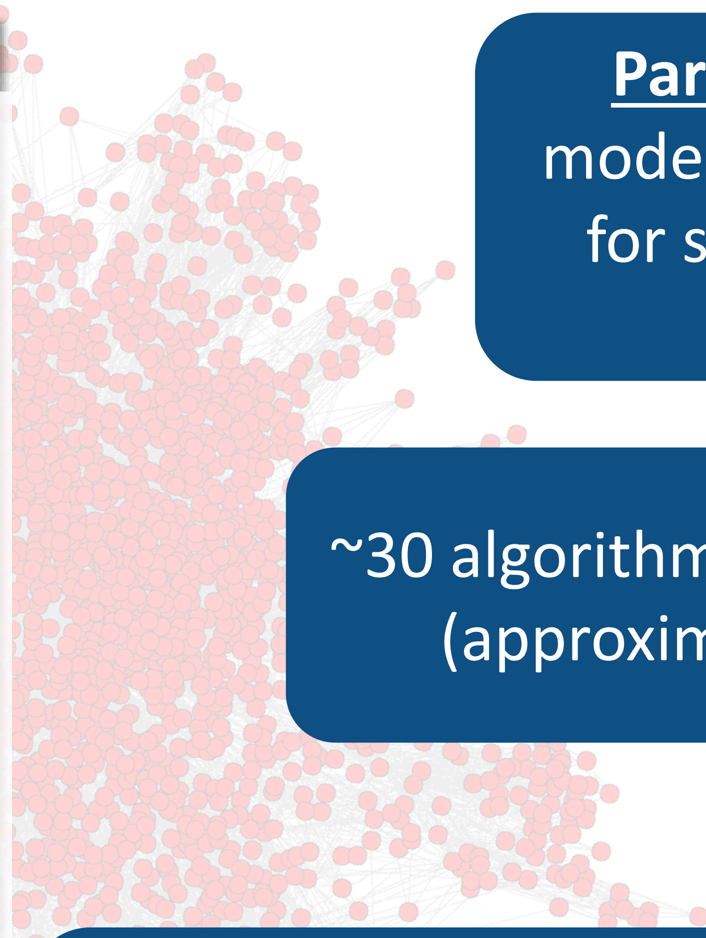
Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[26]	$1/2$	$O(n)$	1	👎	👍	👍
[41, Theorem 6]	$1/2+0.0071$	$O(n \operatorname{polylog}(n))$	2	👎	👍	👍
[41, Theorem 2]	$1/2+0.003^{*}$	$O(n \operatorname{polylog}(n))$	1	👎	👍	👍
[36, Theorem 1.1]	$O(\operatorname{polylog}(n))$	$O(\operatorname{polylog}(n))$	1	👎	👍	👍
[26, Theorem 1]	$2/3-\varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	$1-\varepsilon$	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[41, Theorem 5]	$1/2+0.019$	$O(n \operatorname{polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2+0.005^{*}$	$O(n \log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2+0.0071^{*}$	$O(n \operatorname{polylog}(n))$	2	👎	👎	👍
[39]	$1-1/e$	$O(n \operatorname{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1-\frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n \operatorname{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n \operatorname{polylog}(n))$	1	👍	👍	👍

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

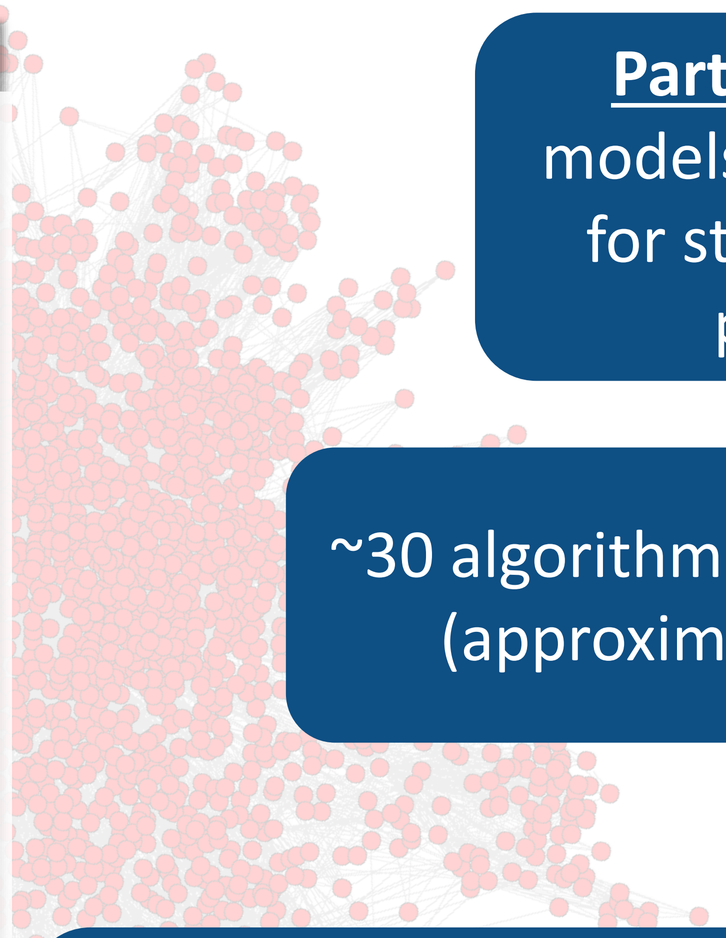
~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	$1/2$	$O(n)$	1	👎	👍	👍
[illegible]	[illegible]$.0071$	$O(n \operatorname{polylog}(n))$	2	👎	👍	👍
[illegible]	[illegible]$.003^*$	$O(n \operatorname{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]$g(n))$	$O(\operatorname{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[illegible]	[illegible]	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[illegible]	[illegible]$.019$	$O(n \operatorname{polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2+0.005^*$	$O(n \log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2+0.0071^*$	$O(n \operatorname{polylog}(n))$	2	👎	👎	👍
[39]	$1-1/e$	$O(n \operatorname{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1-\frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n \operatorname{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log$[illegible]	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log$[illegible]	$O(\log_{1+\varepsilon/3} n)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	[illegible]	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	[illegible]	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n \operatorname{polylog}(n))$	1	👍	👍	👍

Most important goals:

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

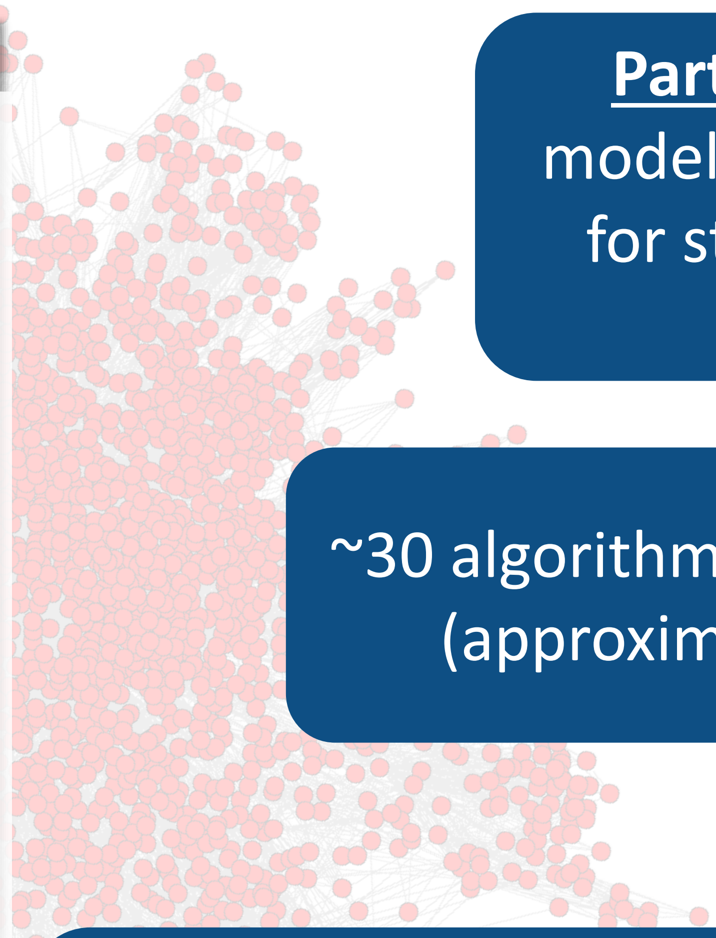
Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	[illegible]	$O(n)$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \text{ polylog}(n))$	2	👎	👍	👍
[illegible]	[illegible]	$O(n \text{ polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(\text{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[illegible]	[illegible]	$O\left(n \text{ polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[illegible]	[illegible]	[illegible]	2	👎	👎	👍
[41, Theorem	[illegible]	[illegible]	1	👎	👎	👍
[41, Theorem	[illegible]	[illegible]	2	👎	👎	👍
[39]	[illegible]	[illegible]	1	👎	👎	👍
[28, Theorem	[illegible]	[illegible]	1	👎	👎	👍
[35, Theorem 2]	[illegible]	[illegible]	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon} + n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	[illegible]	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	[illegible]	[illegible]	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	[illegible]	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	[illegible]	[illegible]	[illegible]	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n \text{ polylog}(n))$	1	👍	👍	👍

Most important goals:

Maximize accuracy

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

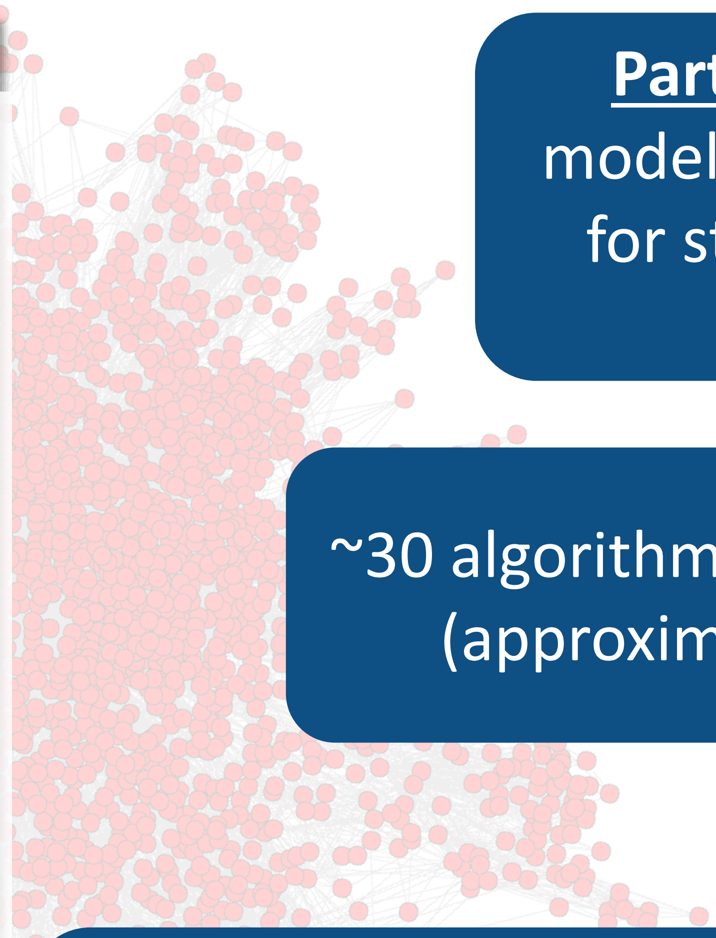
Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n)$	1	👎	👍	👍
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2	👎	👍	👍
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(\mathrm{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \log n)$	$O\left(\log\left(1/\varepsilon\right)/\varepsilon\right)$	👎	👎	👍
[illegible]	[illegible]	$O\left(n\ \mathrm{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	👎	👎	👍
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[41, Theorem [illegible]]	[illegible]	$O(n \log n)$	1	👎	👎	👍
[41, Theorem [illegible]]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[39]	[illegible]	$O(n\ \mathrm{polylog}(n))$	1	👎	👎	👍
[28, Theorem [illegible]]	[illegible]	[illegible]	[illegible]	👎	👎	👍
[35, Theorem 2]	$1 - \frac{[illegible]}{(k-1)!}$	$O(n)$	[illegible]	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	[illegible]	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon} + n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1+\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	👍

Most important goals:

Maximize accuracy

Minimize local space

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n)$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \text{ polylog}(n))$	2	👎	👍	👍
[illegible]	[illegible]	$O(n \text{ polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(\text{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible]	$O(n \log n)$	[illegible]	👎	👎	👍
[illegible]	[illegible]	$O\left(n \text{ polylog}(n)/\varepsilon^2\right)$	[illegible]	👎	👎	👍
[illegible]	[illegible]	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	[illegible]	[illegible]	1	👎	👎	👍
[41, Theorem [illegible]	[illegible]	[illegible]	2	👎	👎	👍
[39]	[illegible]	[illegible]	1	👎	👎	👍
[28, Theorem [illegible]	[illegible]	[illegible]	[illegible]	👎	👎	👍
[35, Theorem 2]	[illegible]	$O(n)$	[illegible]	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	[illegible]	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	❓
[44, Theorem 3]	$2+\varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	❓
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	❓
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	❓
[25]	$4.911+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	❓
[29]	$3.5+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	❓
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	❓
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	❓
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	[illegible]	👍	👍	❓
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]	👍	👍	👍
[6, Theorem 22]	[illegible]	[illegible]	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	[illegible]	[illegible]	[illegible]	👍	👎	👍
Crouch and Stubbs [1]	ε	$O(n \text{ polylog}(n))$	1	👍	👍	👍

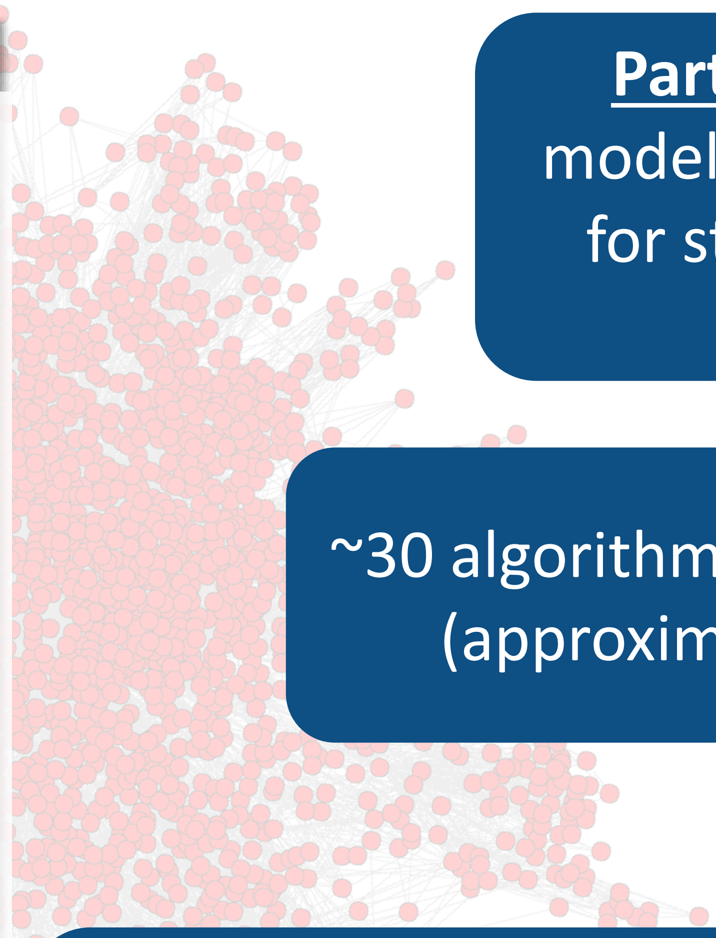
Most important goals:

Maximize accuracy

Minimize local space

Minimize #passes

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n)$	1			
[illegible]	[illegible]	$O(n\ \text{polylog}(n))$	2			
[illegible]	[illegible]	$O(n\ \text{polylog}(n))$	1			
[illegible]	[illegible]	$O(\text{polylog}(n))$	1			
[illegible]	[illegible]	$O(n \log n)$	[illegible]			
[illegible]	[illegible]	$O\left(n\ \text{polylog}(n)/\varepsilon^2\right)$	[illegible]			
[illegible]	[illegible]	$O(n\ \text{polylog}(n))$	2			
[41, Theorem 1]	[illegible]	[illegible]	1			
[41, Theorem [illegible]	[illegible]	[illegible]	2			
[39]	[illegible]	[illegible]	1			
[28, Theorem [illegible]	[illegible]	[illegible]	[illegible]			
[35, Theorem 2]	[illegible]	$O(n)$	[illegible]			
[14]	1	$\tilde{O}\left(k^2\right)$	[illegible]			
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1			
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1			
[26, Theorem 2]	6	$O(n \log n)$	1			
[44, Theorem 3]	$2+\varepsilon$	$O(n\ \text{polylog}(n))$	$O(1)$			
[44, Theorem 3]	5.82	$O(n\ \text{polylog}(n))$	1			
[63]	5.58	$O(n\ \text{polylog}(n))$	1			
[25]	$4.911+\varepsilon$	$O(n\ \text{polylog}(n))$	1			
[29]	$3.5+\varepsilon$	$O(n\ \text{polylog}(n))$	1			
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1			
[27]	$2+\varepsilon$	[illegible]	[illegible]			
[26, Section 3.2]	$2+\varepsilon$	[illegible]	[illegible]			
[6, Theorem 28]	[illegible]	[illegible]	[illegible]			
[6, Theorem 22]	[illegible]	[illegible]	[illegible]			
[6, Theorem 22]	[illegible]	[illegible]	[illegible]			
Crouch and Stubbs [1]	[illegible]	$O(n\ \text{polylog}(n))$	1			

Most important goals:

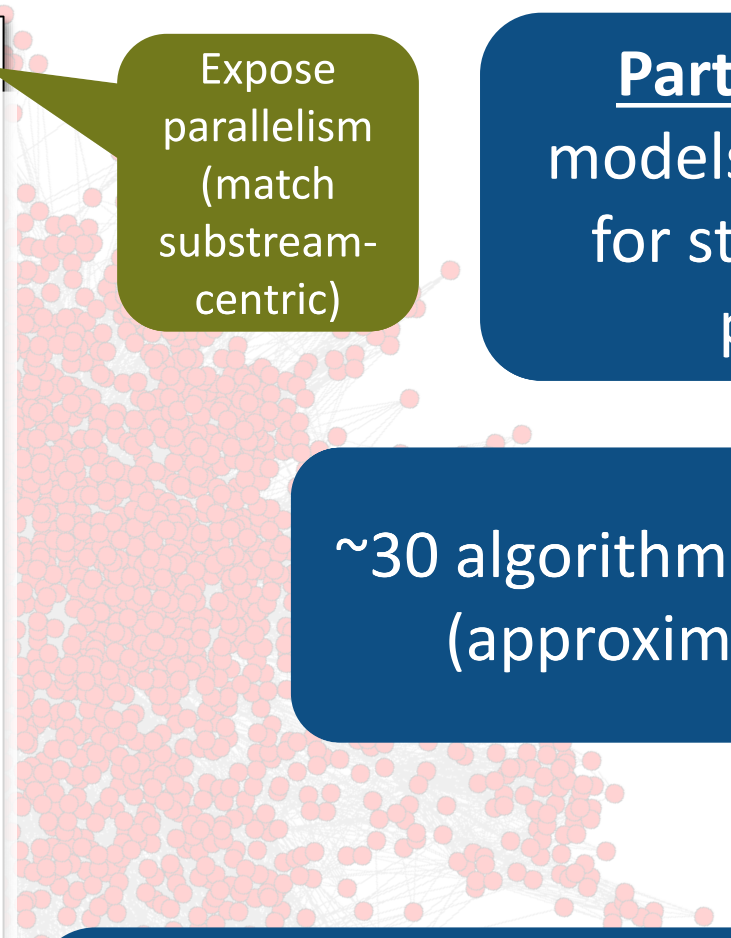
Maximize accuracy

Minimize local space

Minimize #passes

Expose parallelism (match substream-centric)

No worries, no need to analyze it here, all the details are in the paper ☺



Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n)$	1			
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2			
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	1			
[illegible]	[illegible]	$O(\mathrm{polylog}(n))$	1			
[illegible]	[illegible]	$O(n \log n)$	[illegible]			
[illegible]	[illegible]	$O(n\ \mathrm{polylog}/\varepsilon^2)$	[illegible]			
[illegible]	[illegible]	[illegible]	2			
[41, Theorem [illegible]]	[illegible]	[illegible]	1			
[41, Theorem [illegible]]	[illegible]	[illegible]	2			
[39]	[illegible]	[illegible]	1			
[28, Theorem [illegible]]	[illegible]	[illegible]	[illegible]			
[35, Theorem 2]	[illegible]	$O(n)$	[illegible]			
[14]	1	$\tilde{O}(k^2)$				
[14]	$1/\varepsilon$	$\tilde{O}(n^2/\varepsilon^3)$	1			
[7, Theorem 1]	n^ε	$\tilde{O}(n^{2-3\varepsilon}+n^{1-\varepsilon})$	1			
[26, Theorem 2]	6	$O(n \log n)$	1			
[44, Theorem 3]	$2+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	$O(1)$			
[44, Theorem 3]	5.82	$O(n\ \mathrm{polylog}(n))$	1			
[63]	5.58	$O(n\ \mathrm{polylog}(n))$	1			
[25]	$4.911+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1			
[29]	$3.5+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1			
[53]	$2+\varepsilon$	$O(n \log^2 n)$	1			
[27]	$2+\varepsilon$	[illegible]	1			
[26, Section 3.2]	$2+\varepsilon$	[illegible]	[illegible]			
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]			
[6, Theorem 22]	[illegible]	[illegible]	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$			
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	[illegible]	[illegible]			
Crouch and Stubbs [1]	ε	$O(n\ \mathrm{polylog}(n))$	1			

Most important goals:

Maximize accuracy

Minimize local space

Minimize #passes

Expose parallelism (match substream-centric)

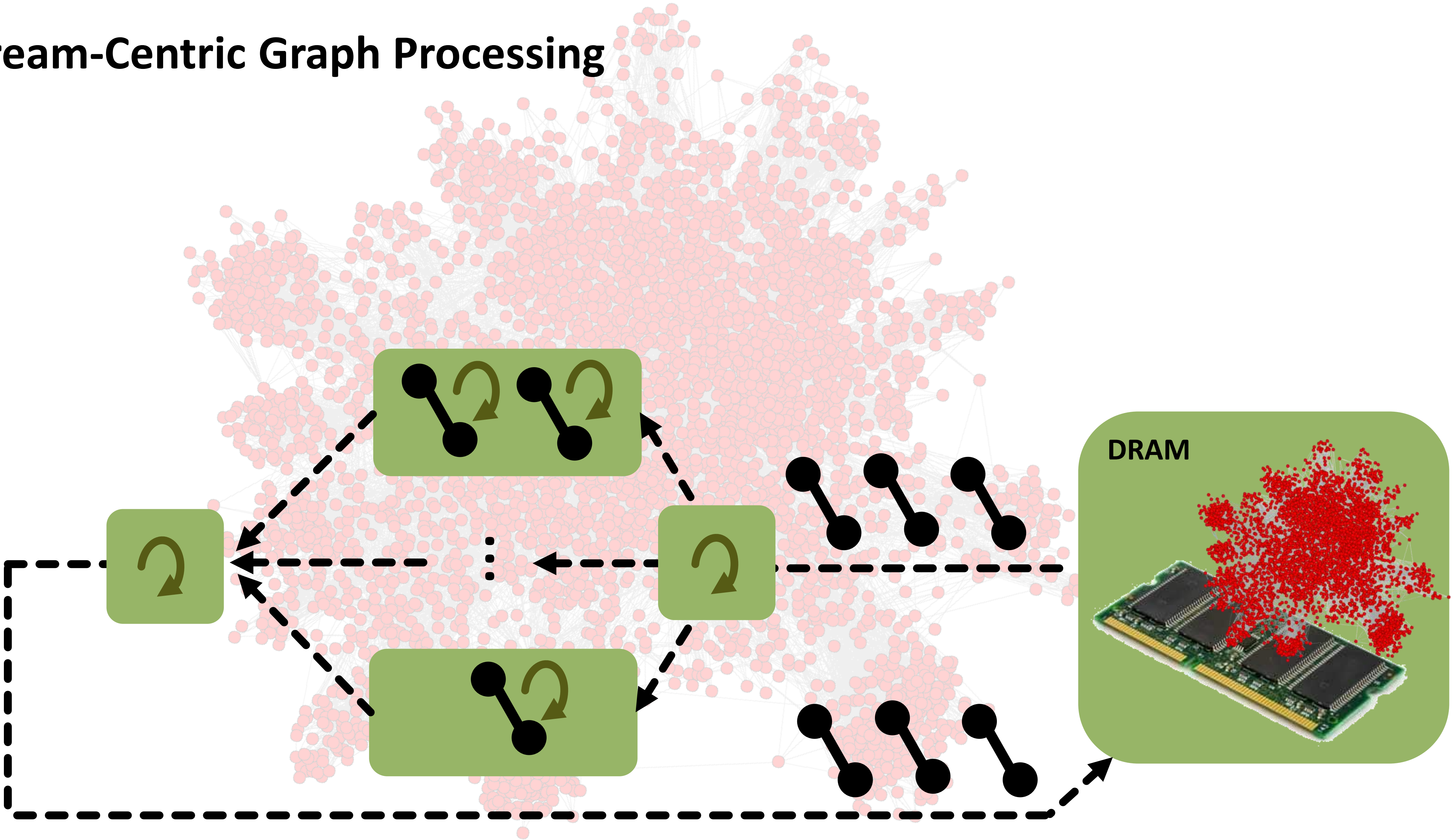
No worries, no need to analyze it here, all the details are in the paper ☺

Part 3: Analysis of models and algorithms for streaming graph processing

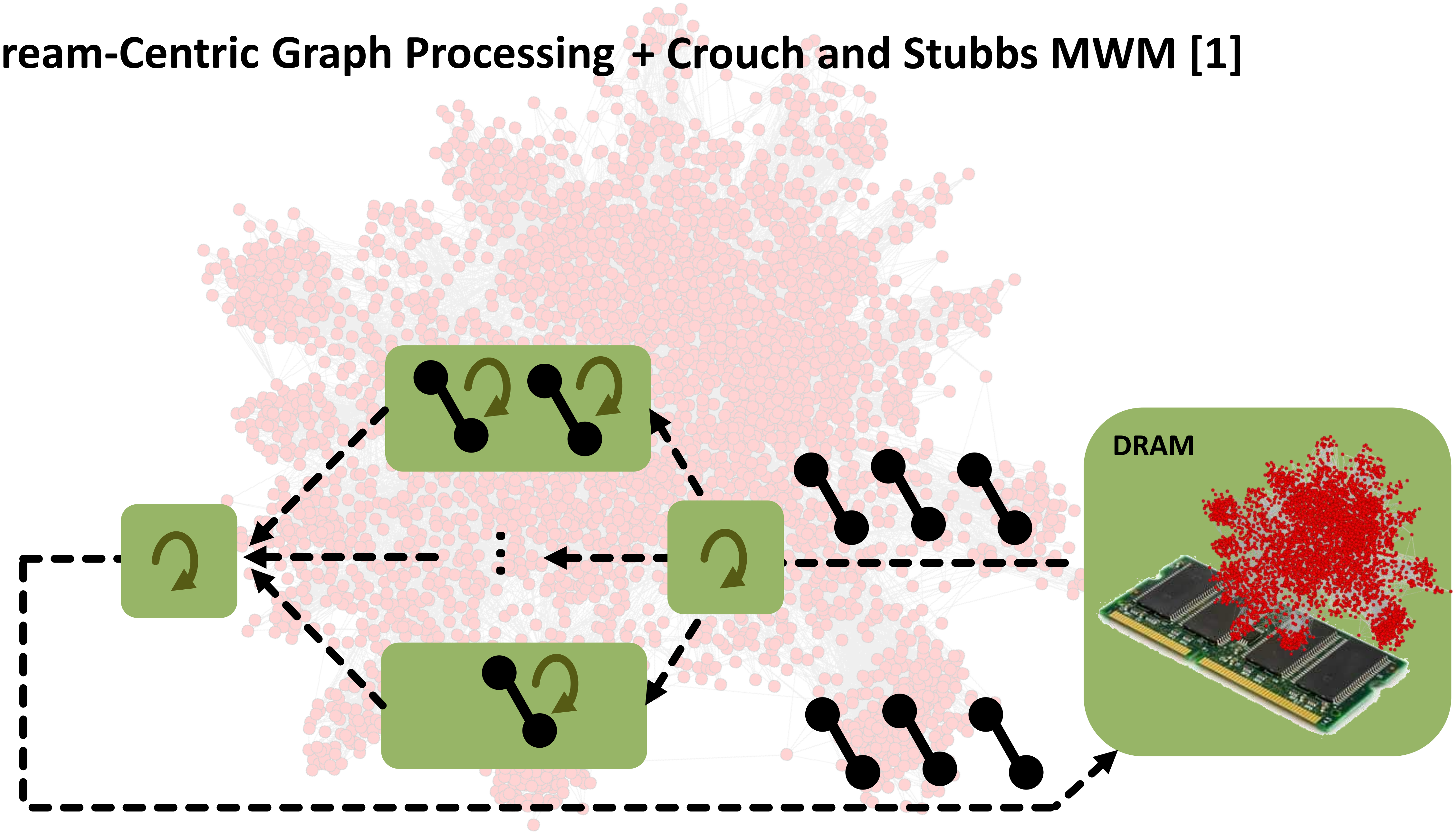
~30 algorithms for streaming (approximate) MWM

Idea: let's check the (rich) world of **streaming theory** and see if there is anything out there that we could use ☺

Substream-Centric Graph Processing

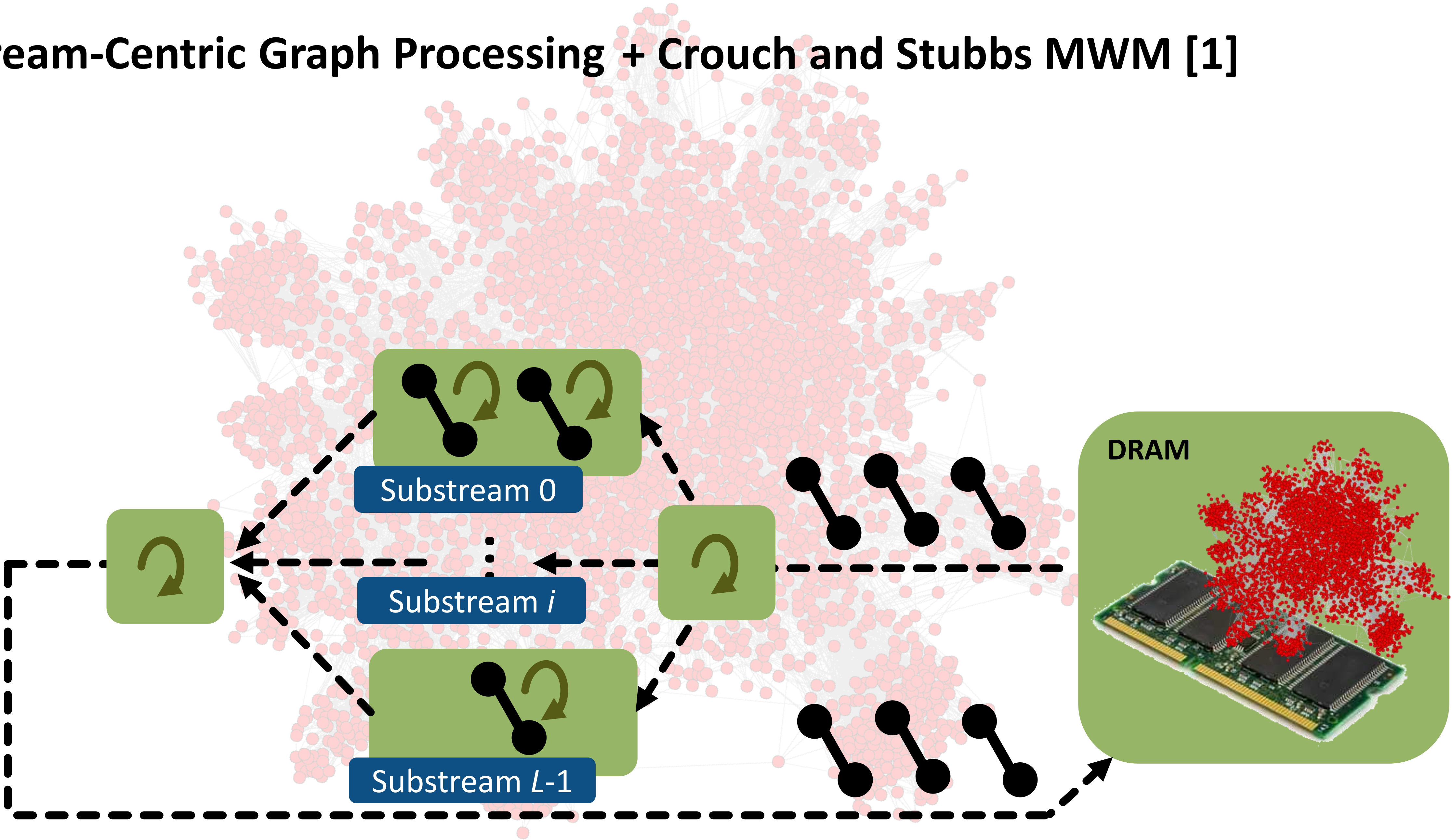


Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



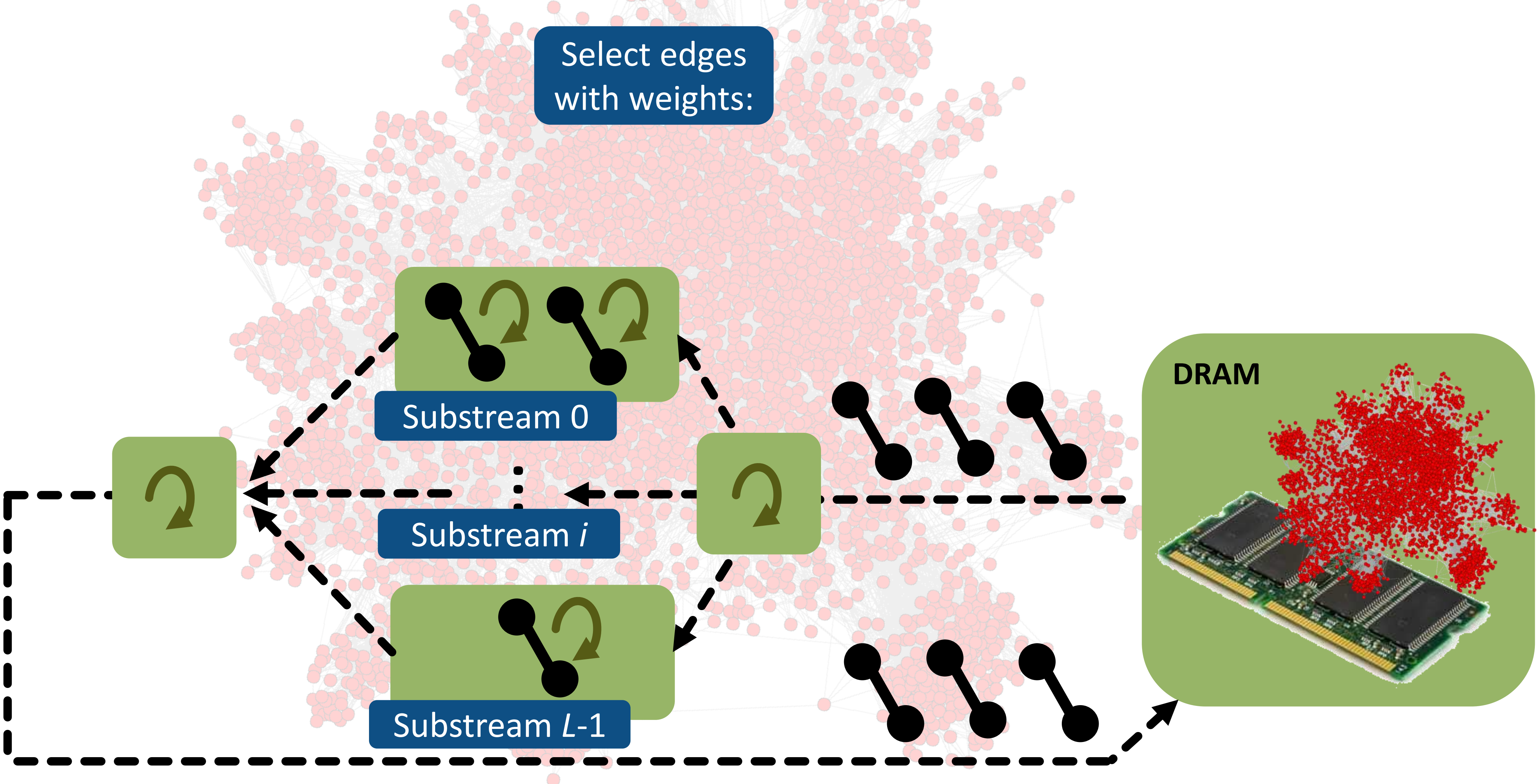
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



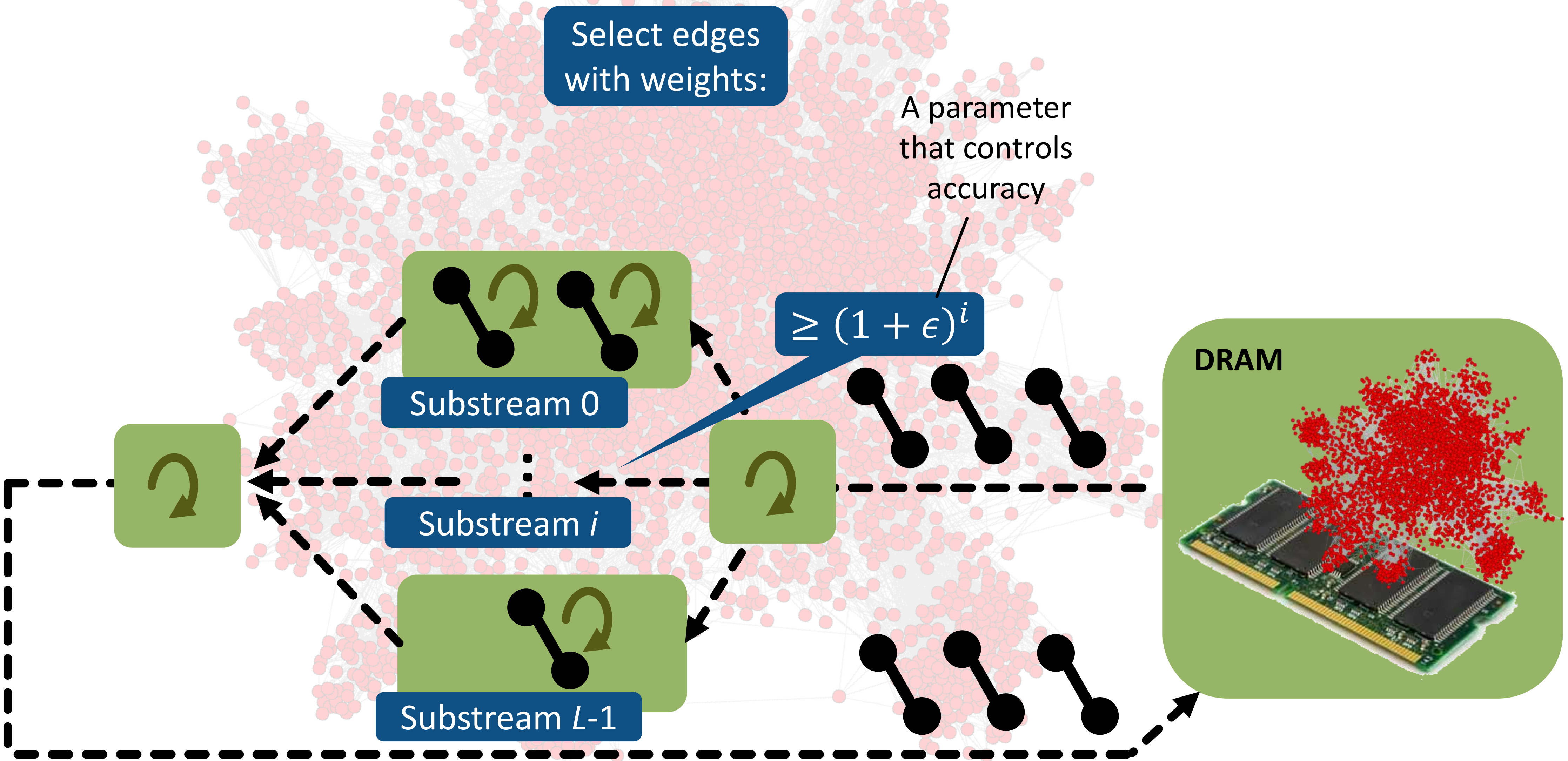
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



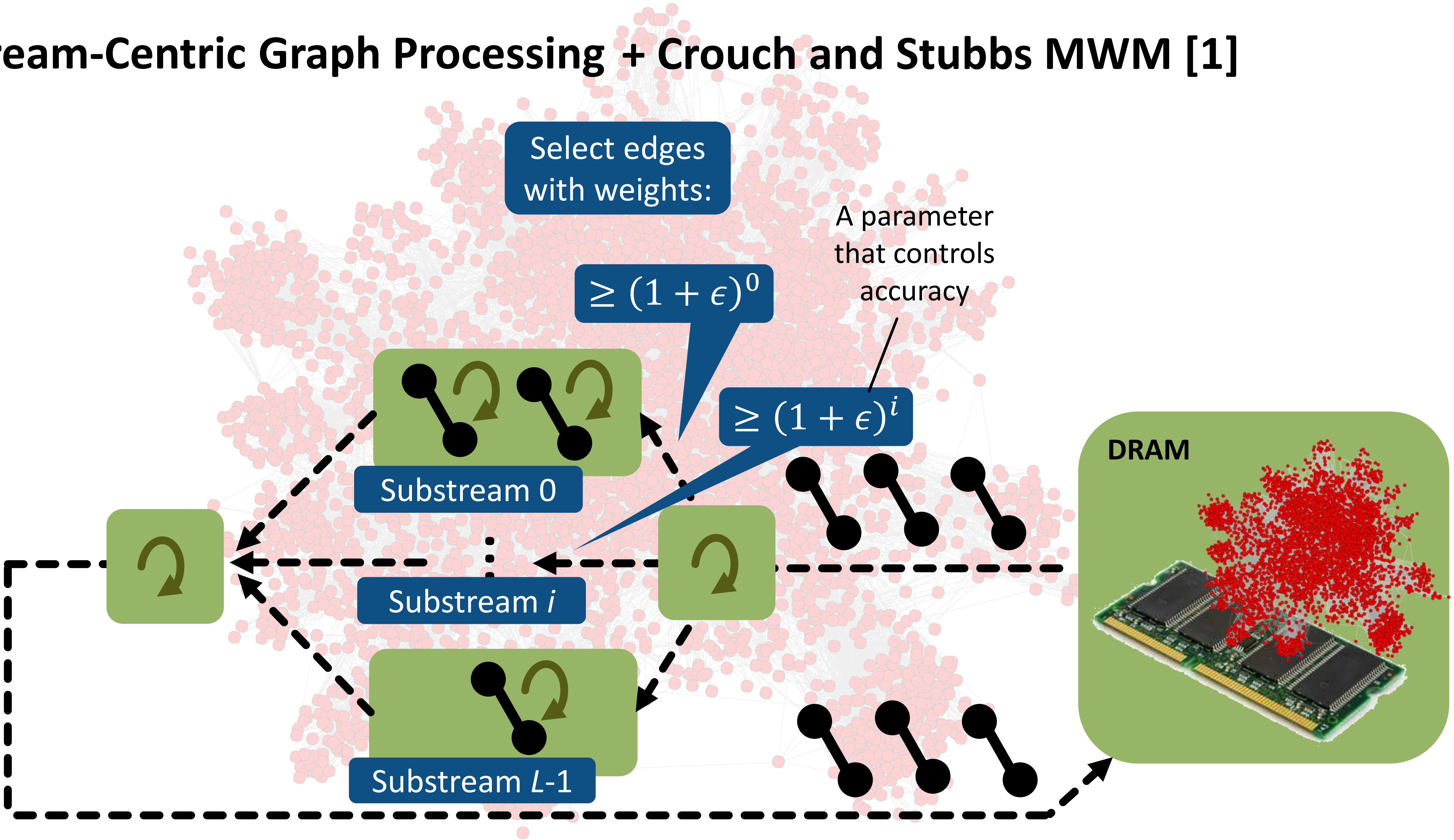
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



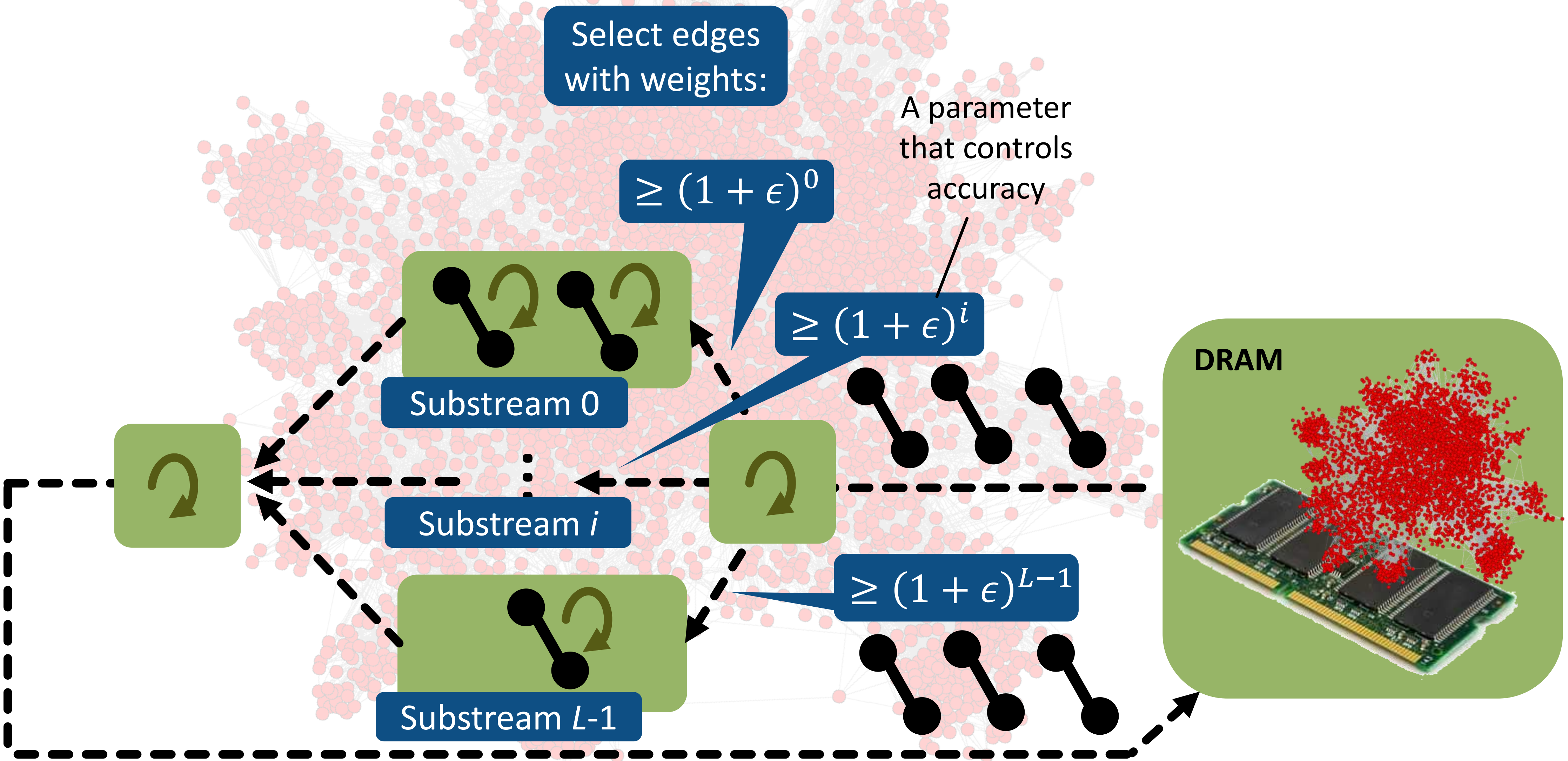
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



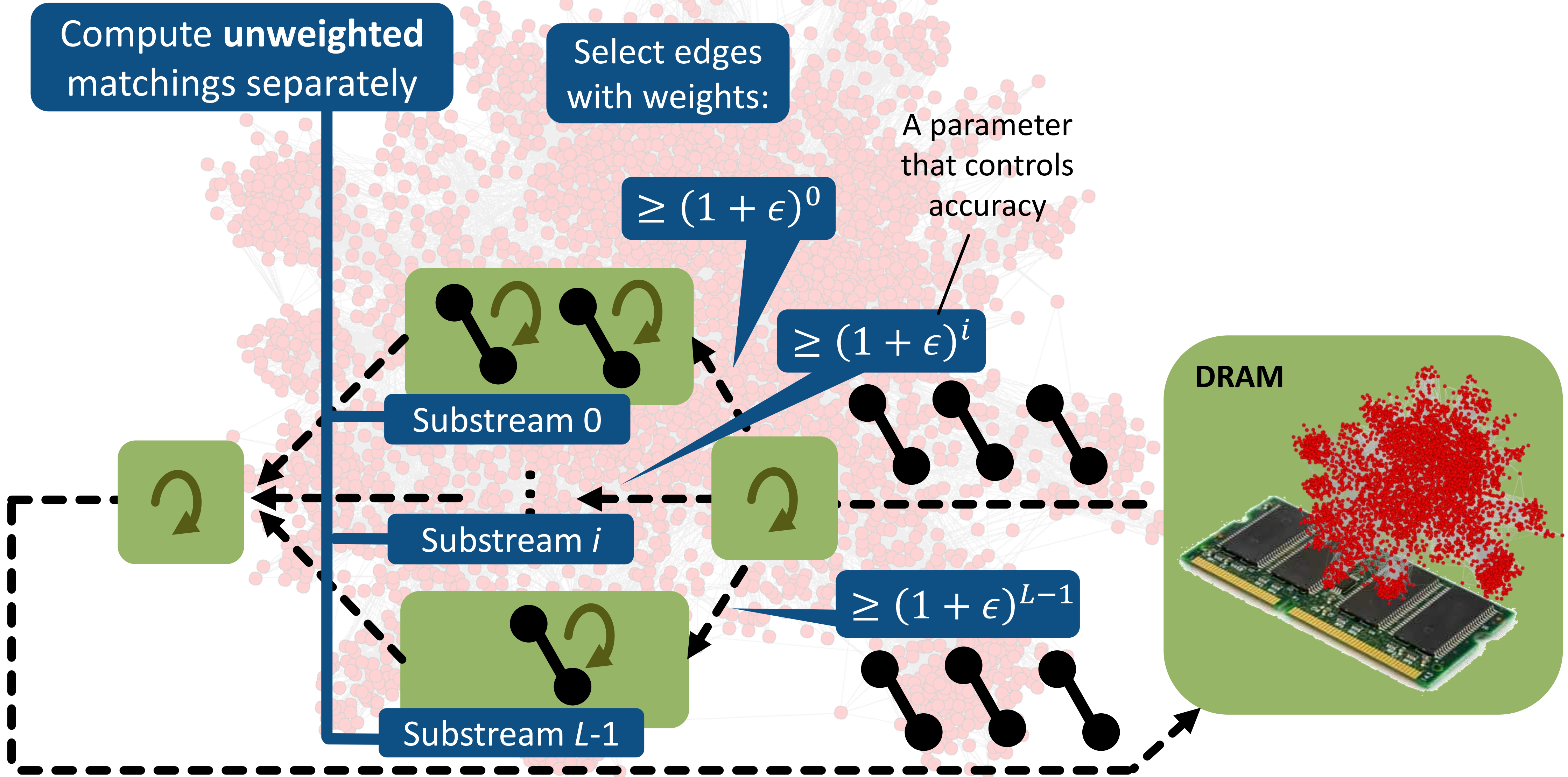
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



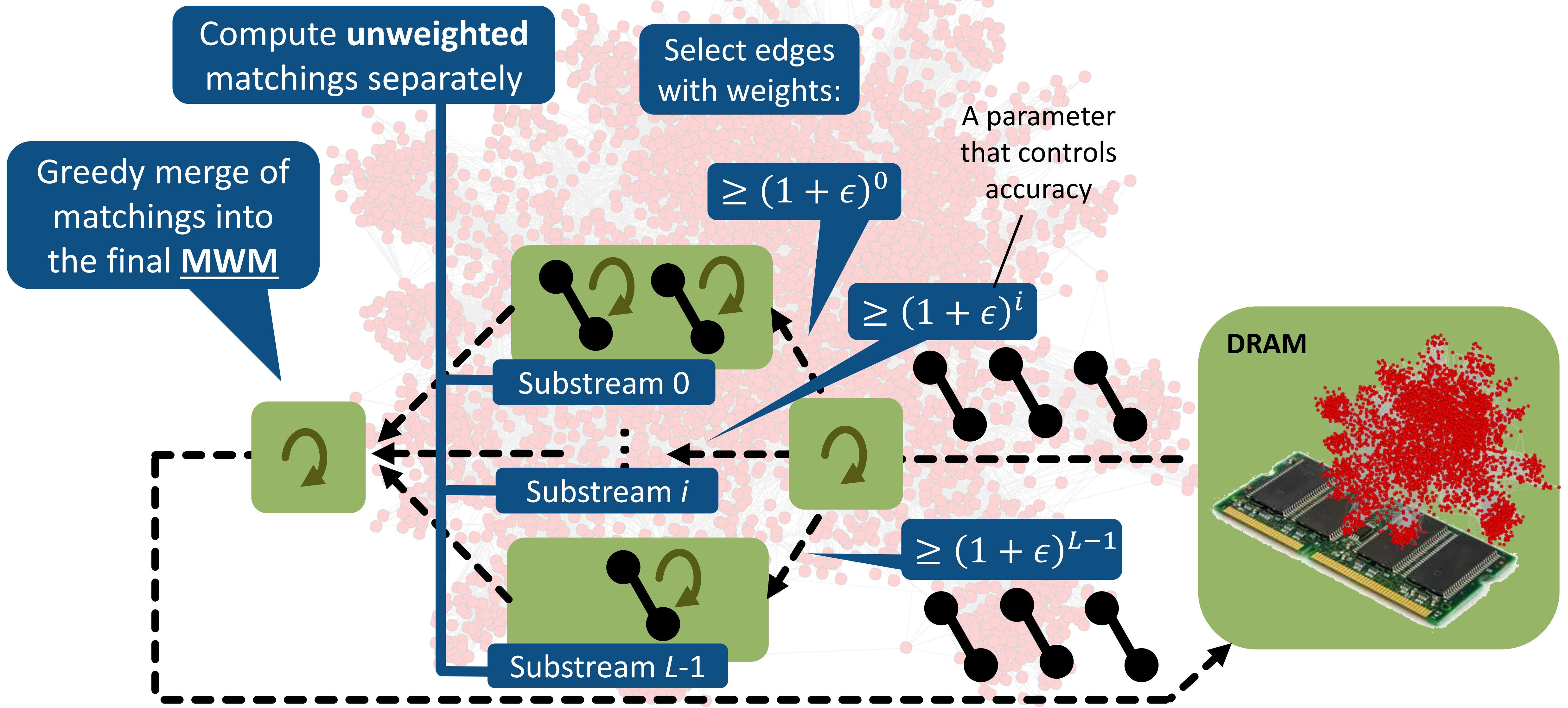
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



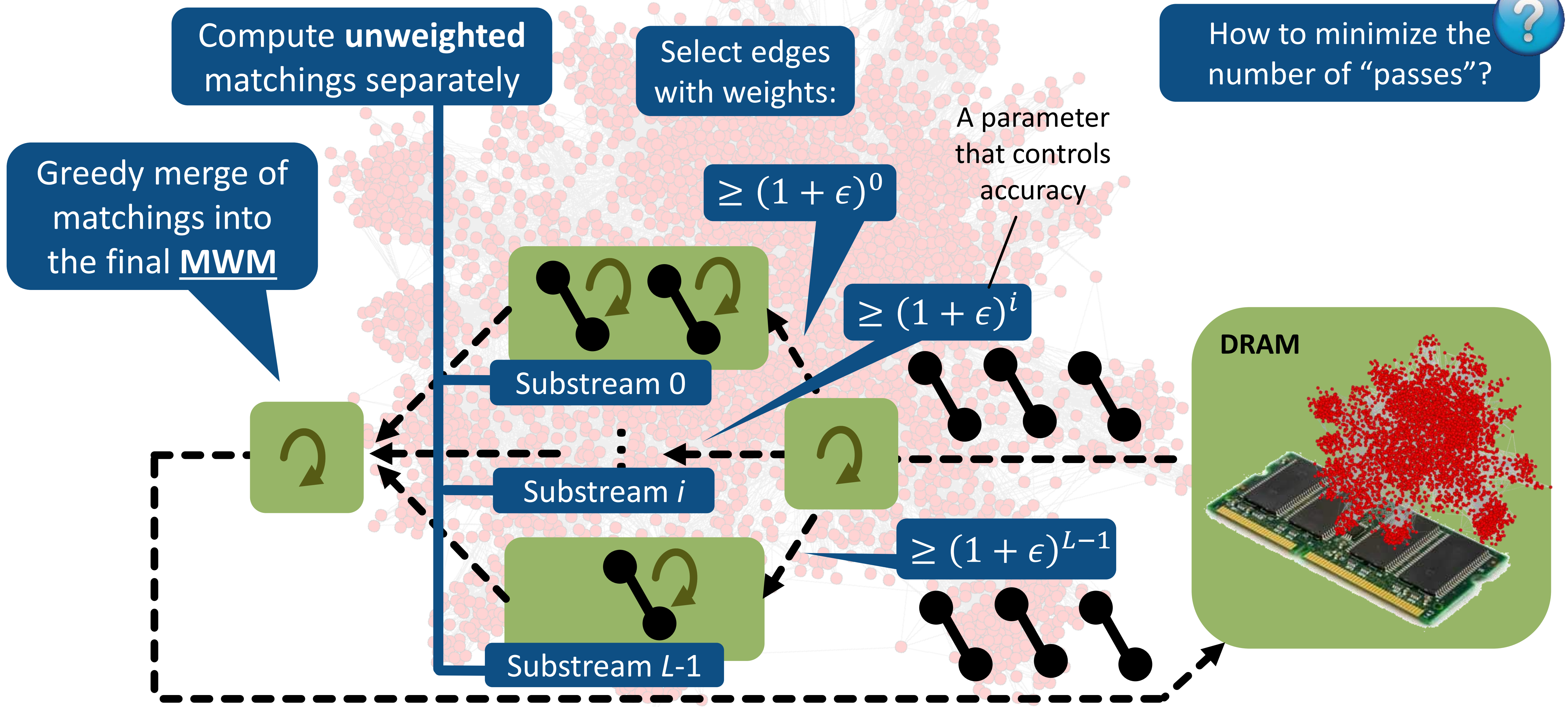
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



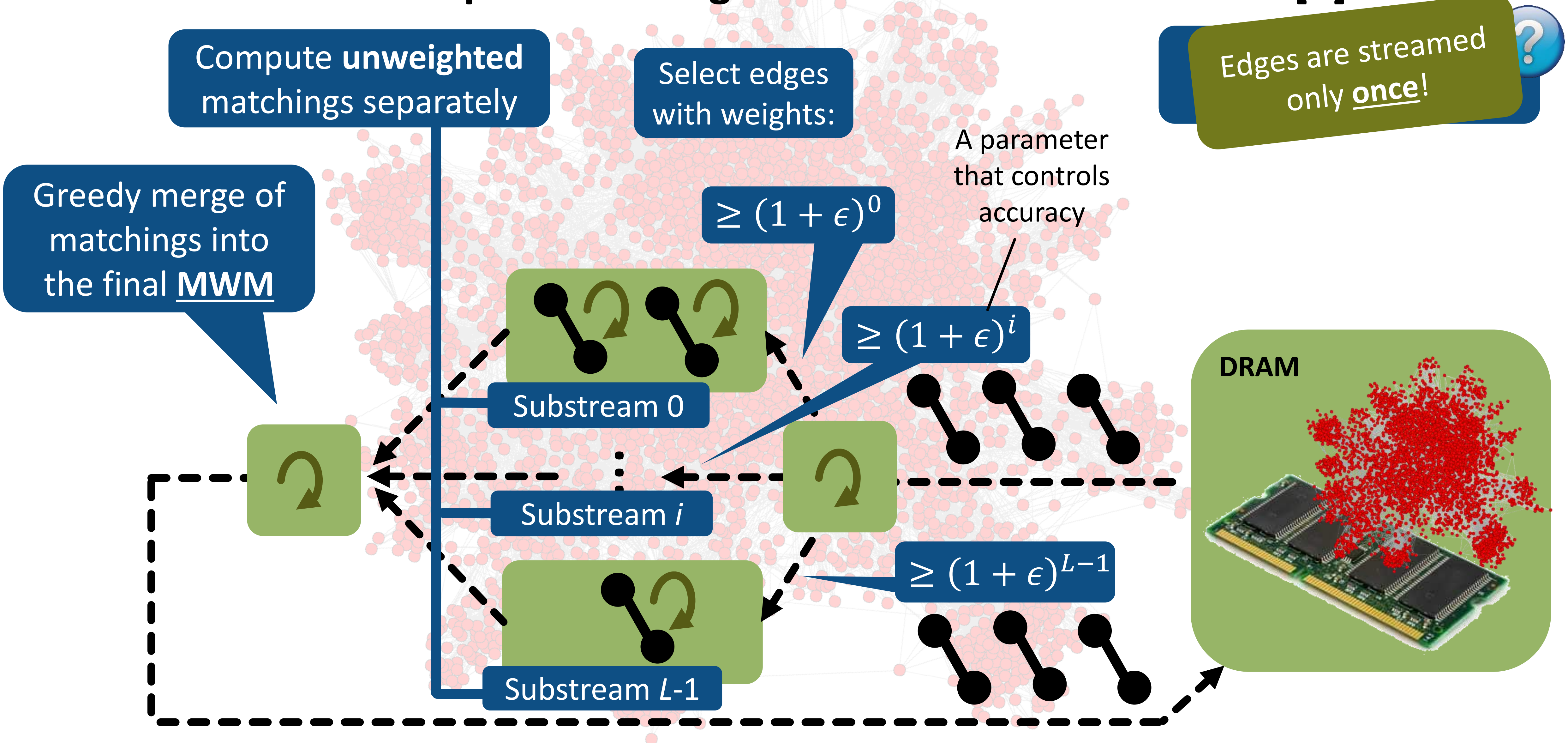
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



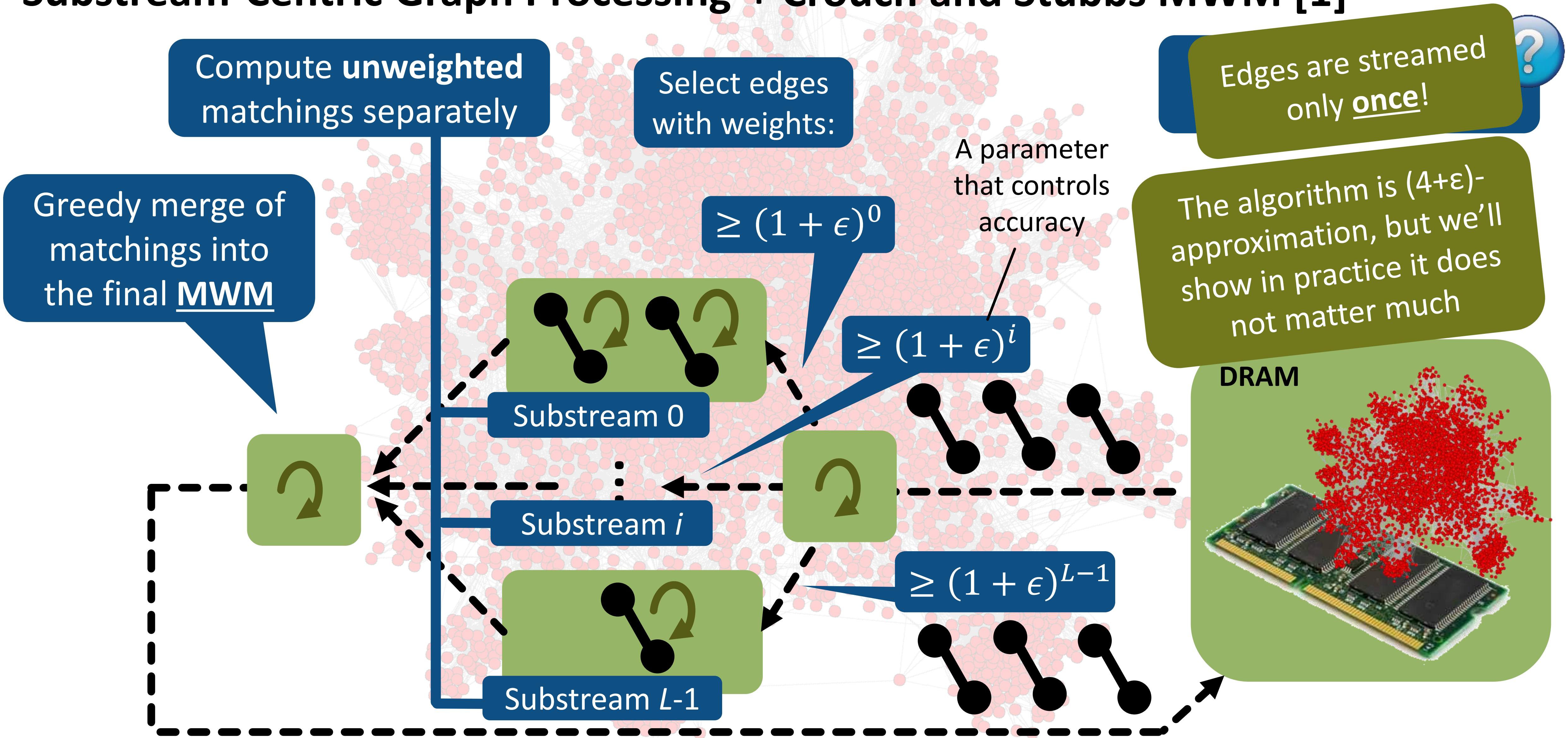
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



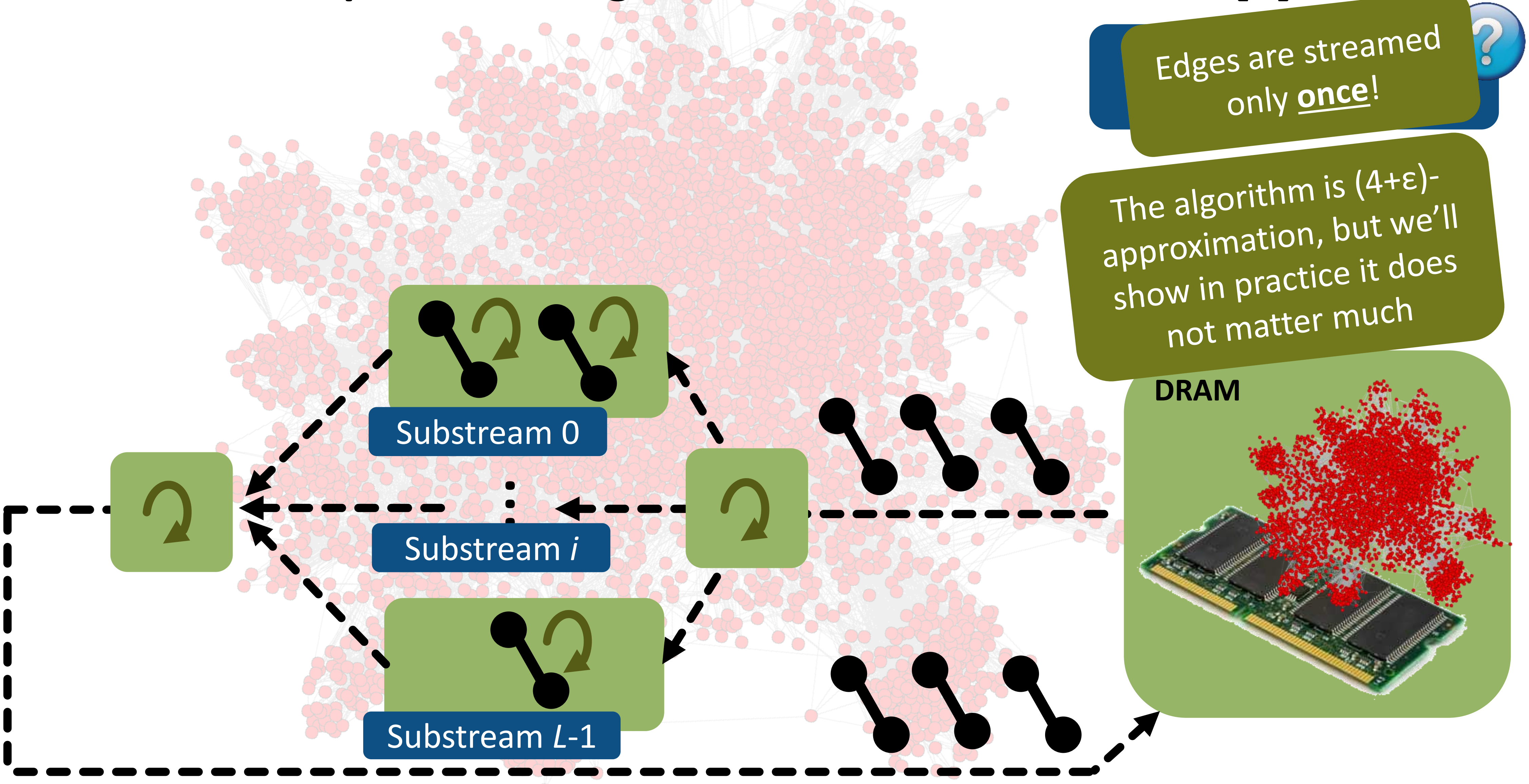
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]

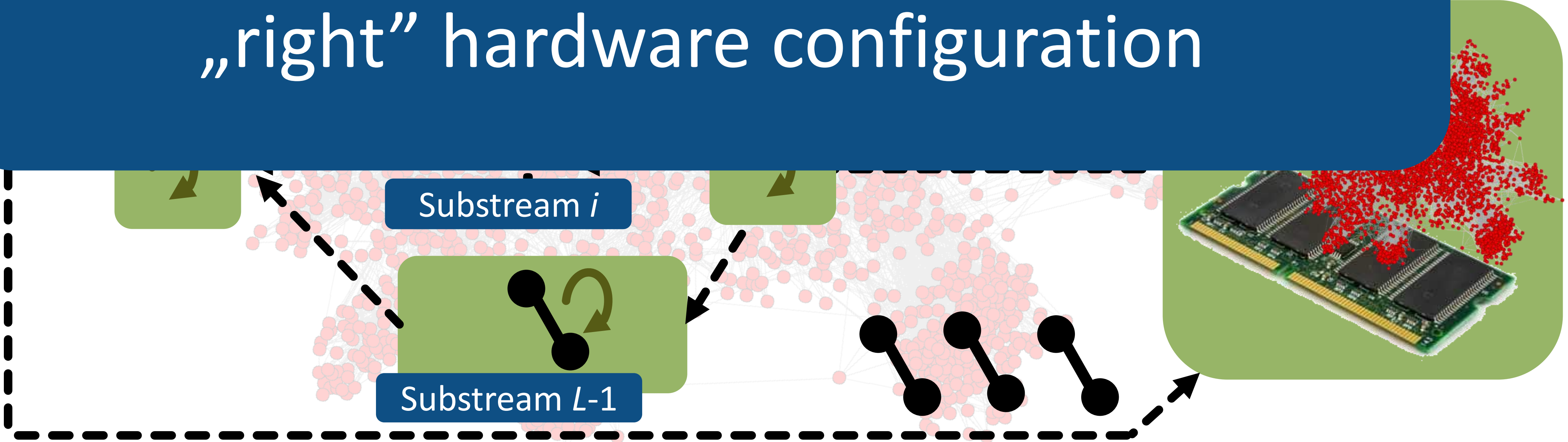


[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]

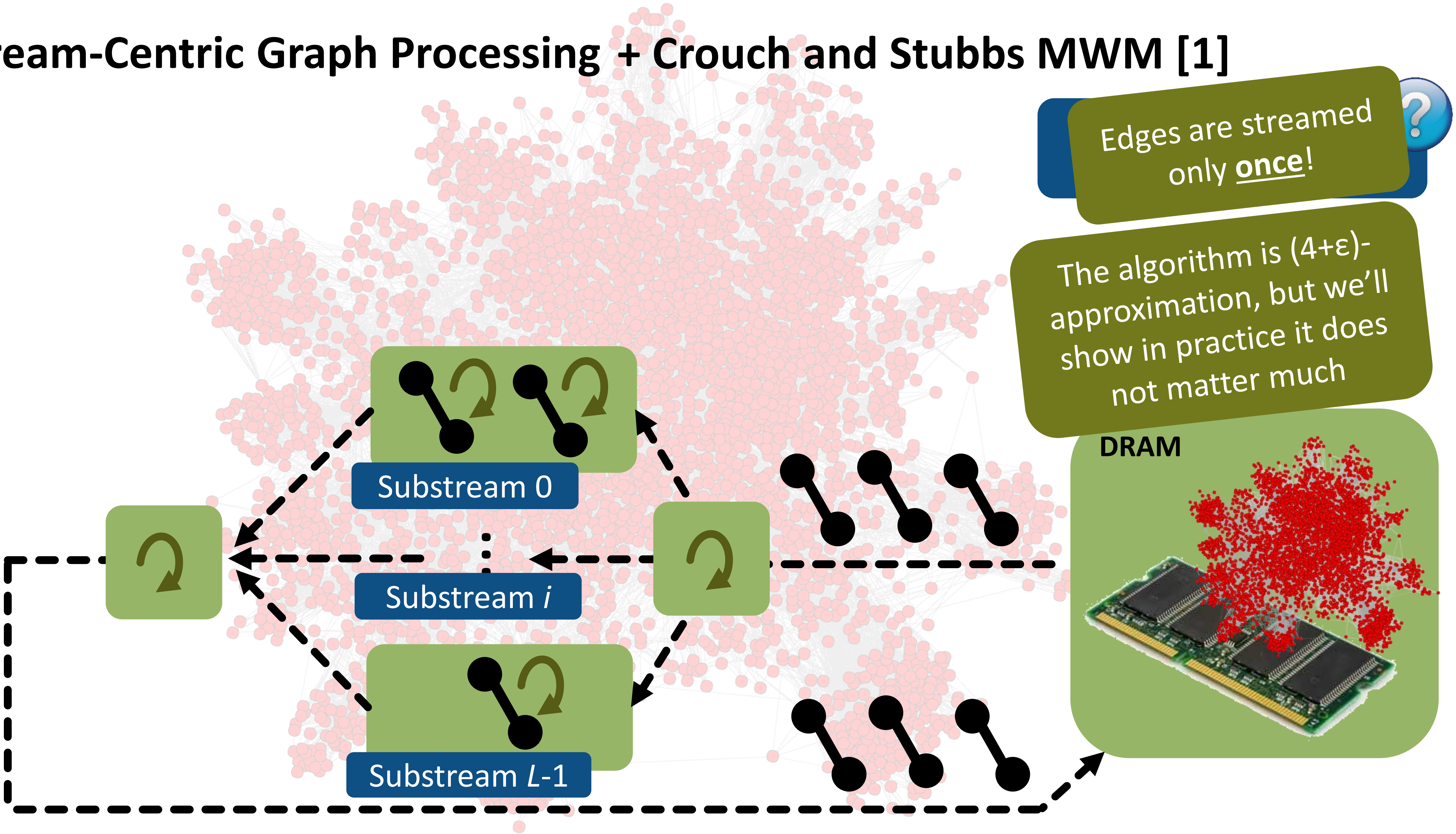
Edges are streamed only **once**!

Part 4: Mapping the algorithm to the „right” hardware configuration



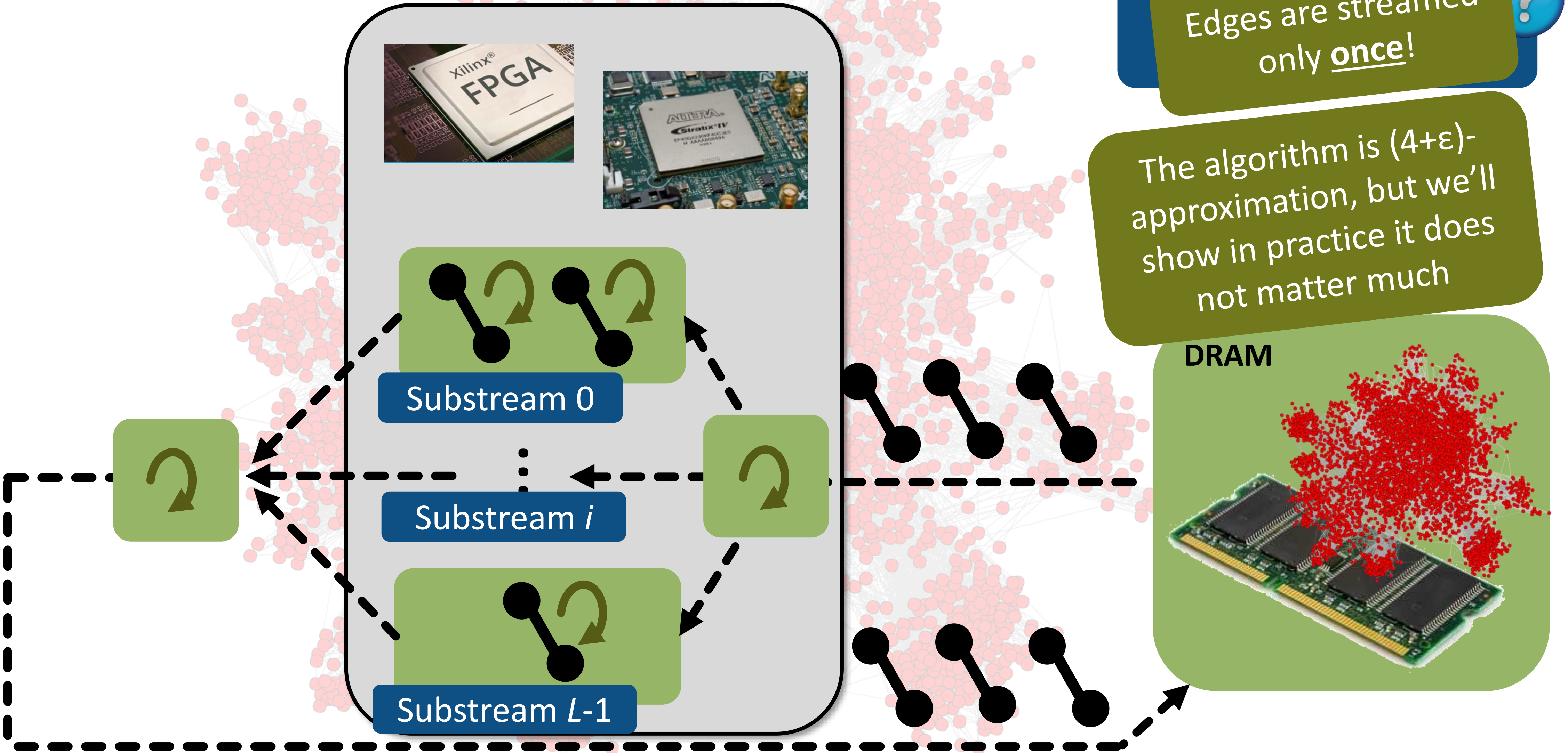
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]

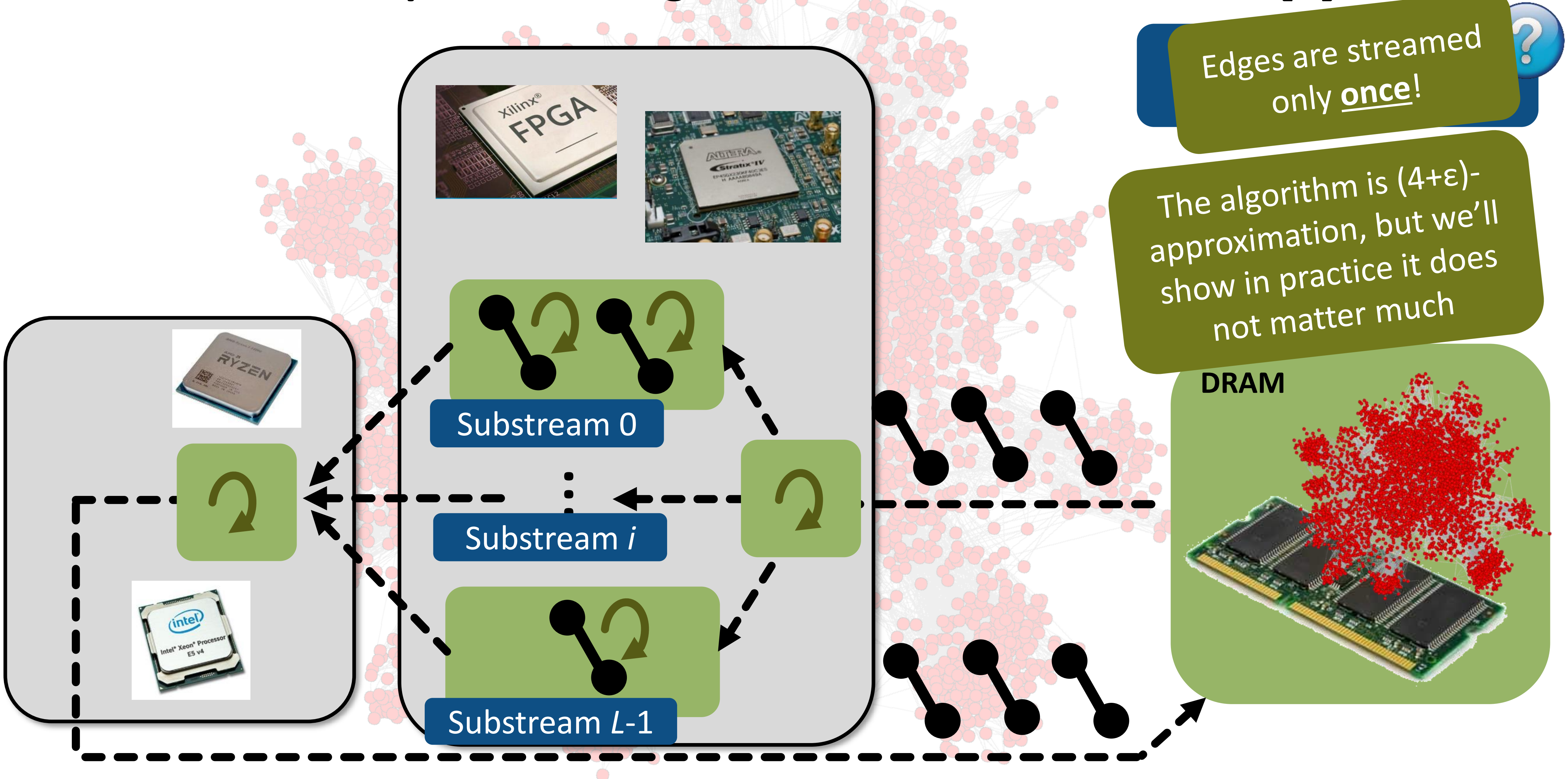


Edges are streamed only **once**!

The algorithm is (4+ε)-approximation, but we'll show in practice it does not matter much

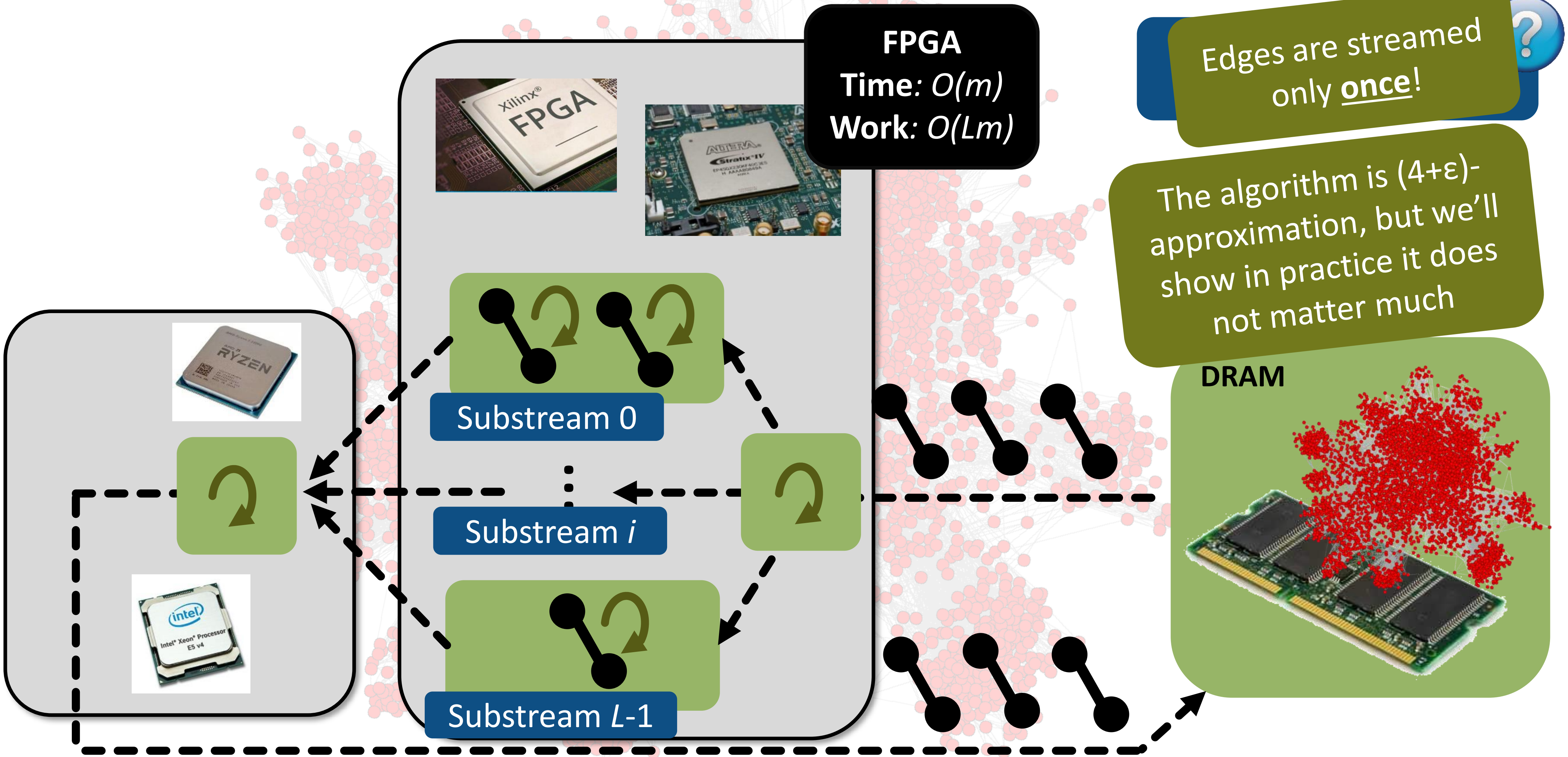
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



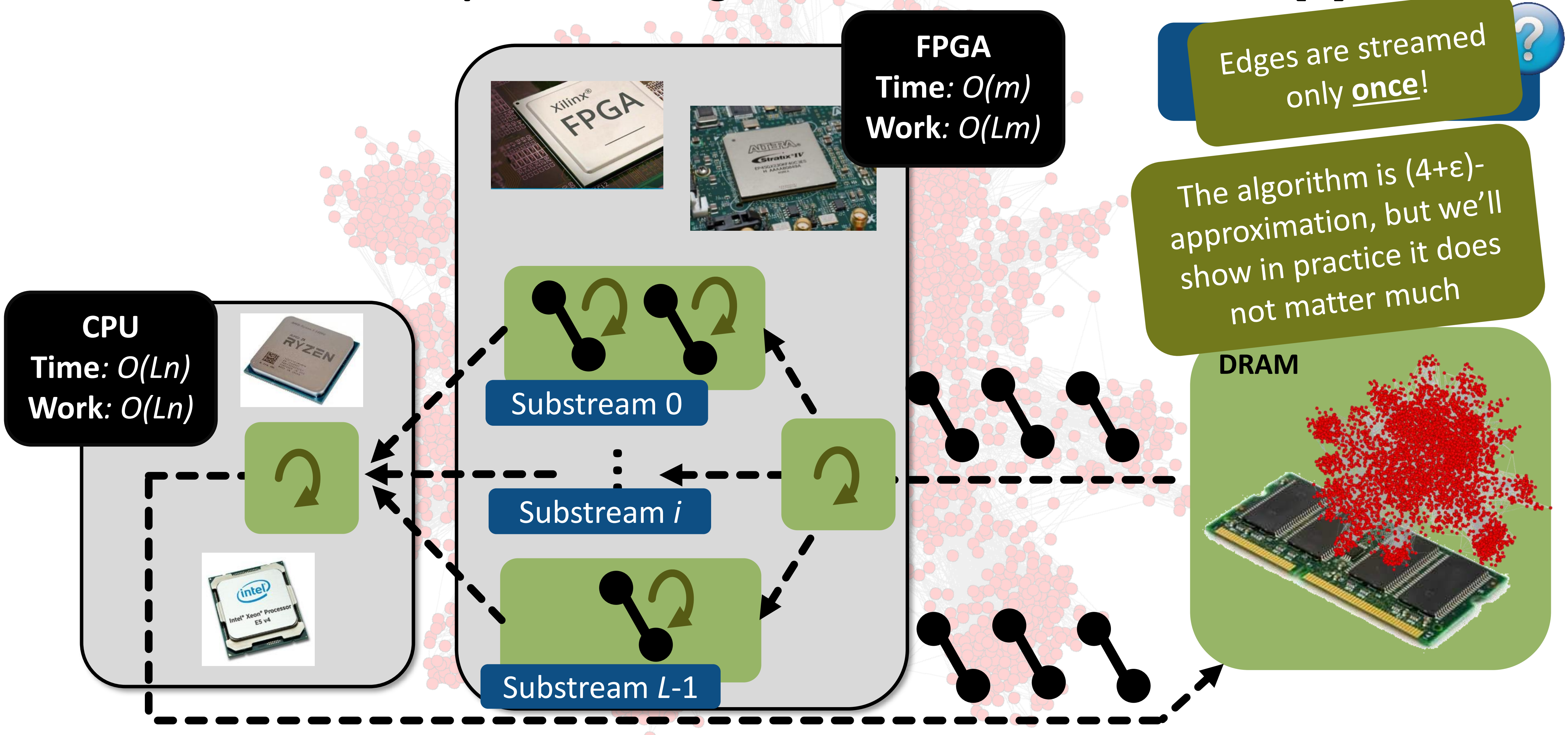
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



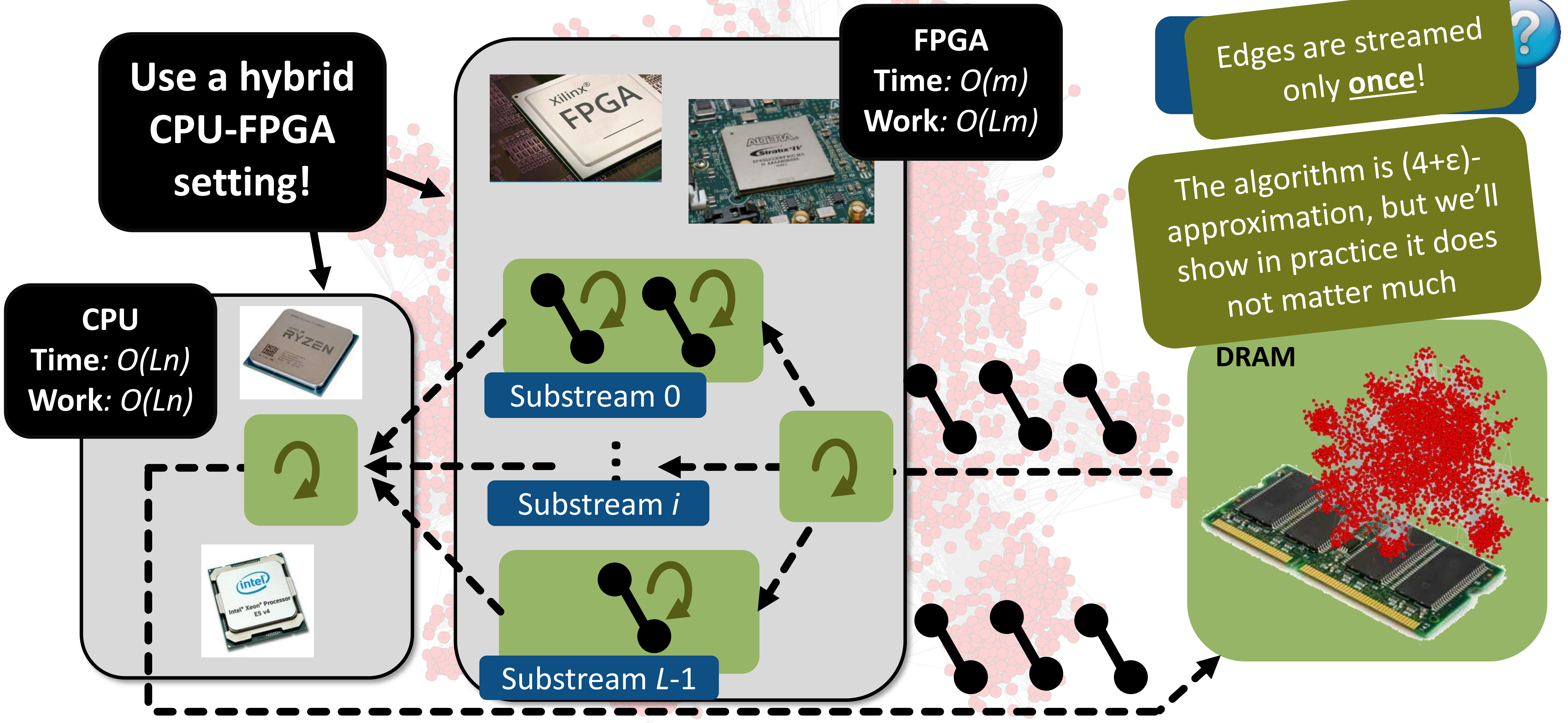
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



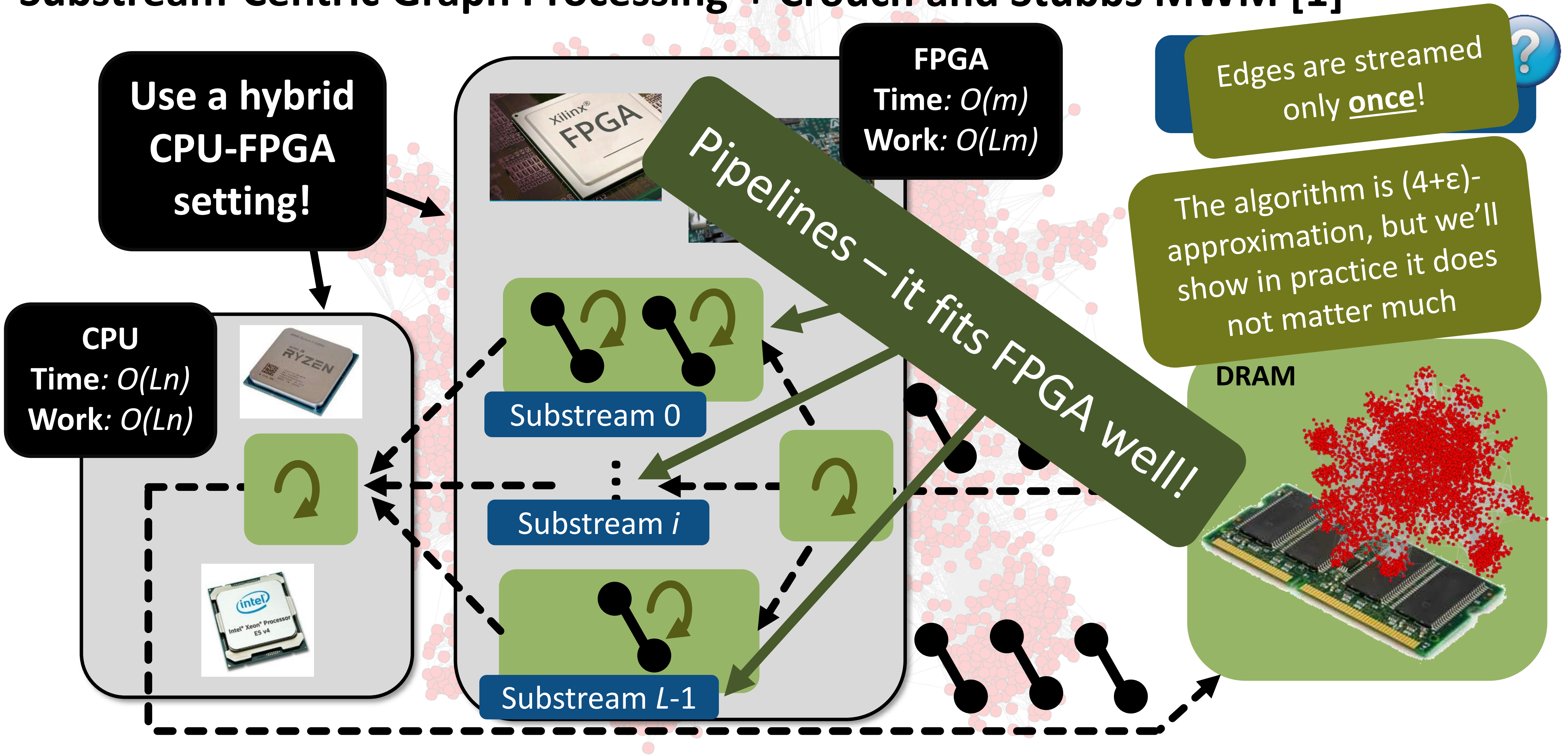
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



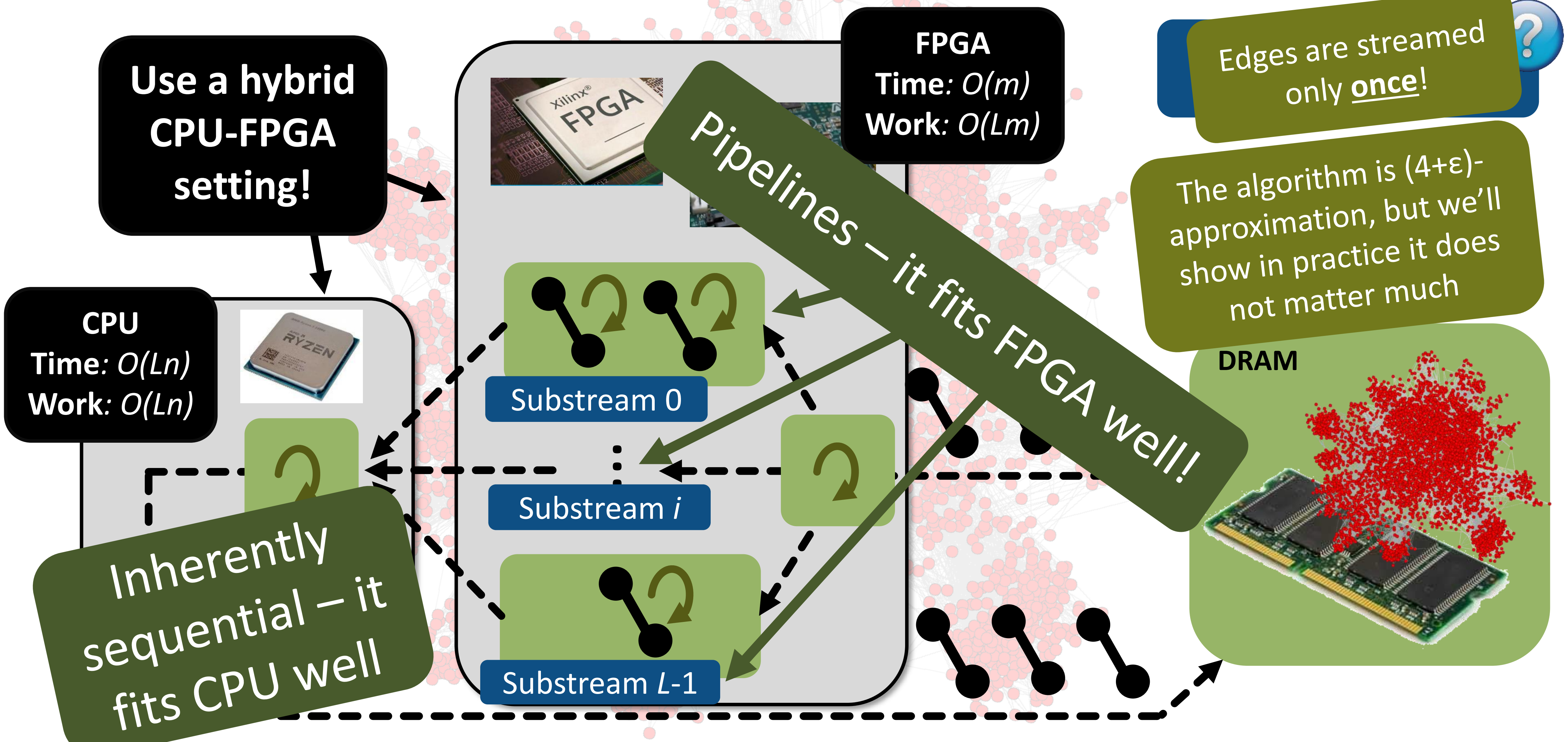
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]



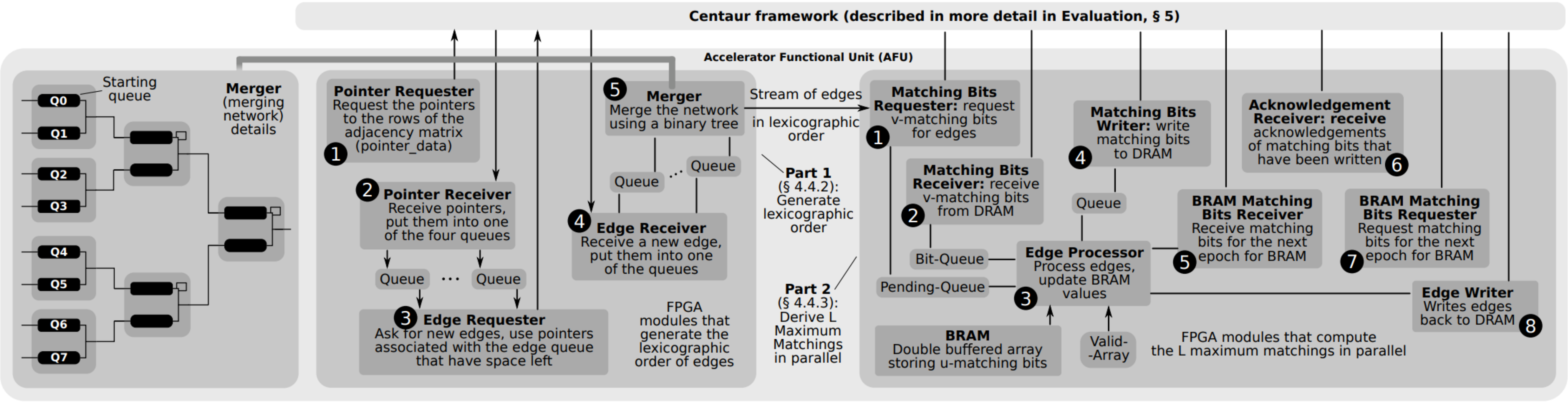
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]

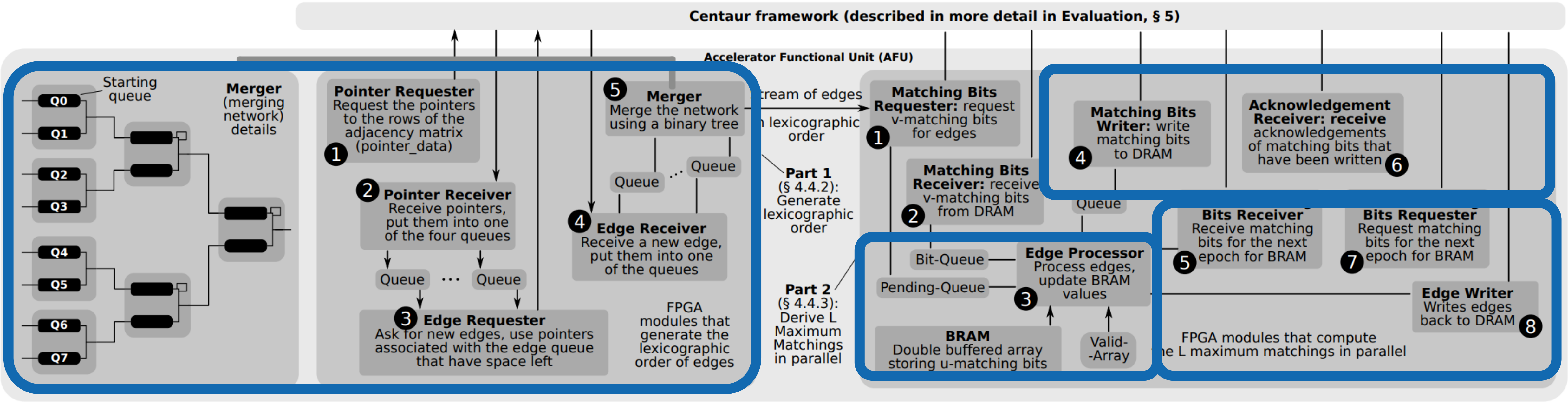


[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

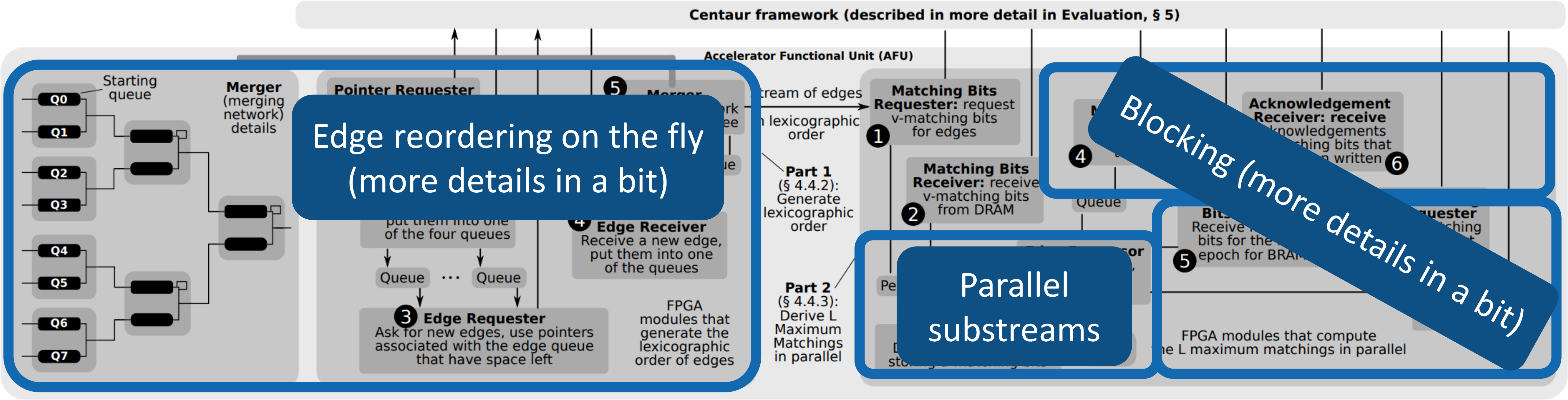
Substream-Centric MWM: FPGA optimizations



Substream-Centric MWM: FPGA optimizations

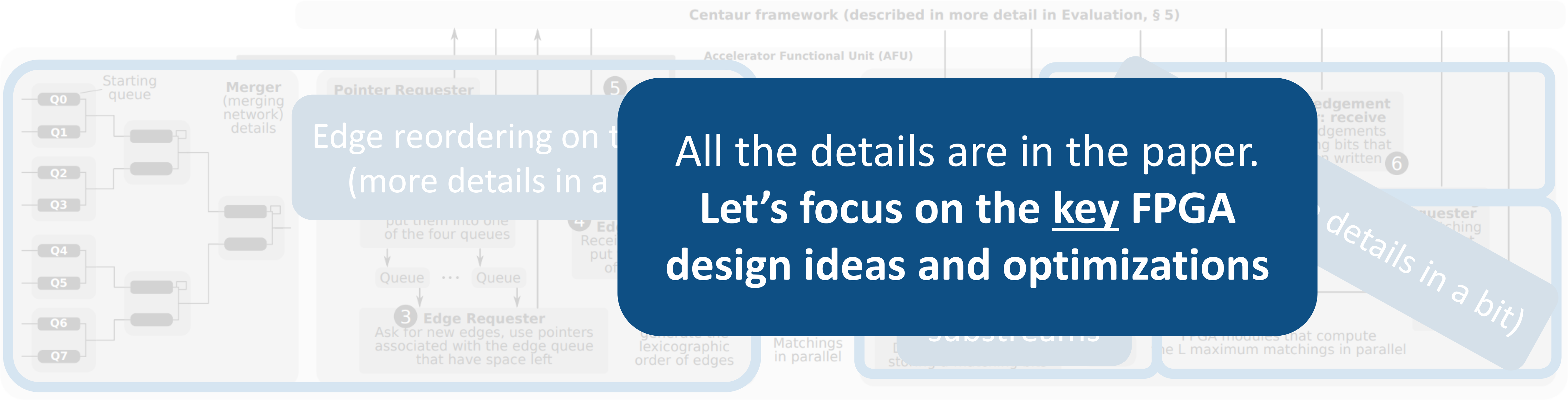


Substream-Centric MWM: FPGA optimizations



Substream-Centric MWM: FPGA optimizations

Substream-Centric MWM: FPGA optimizations

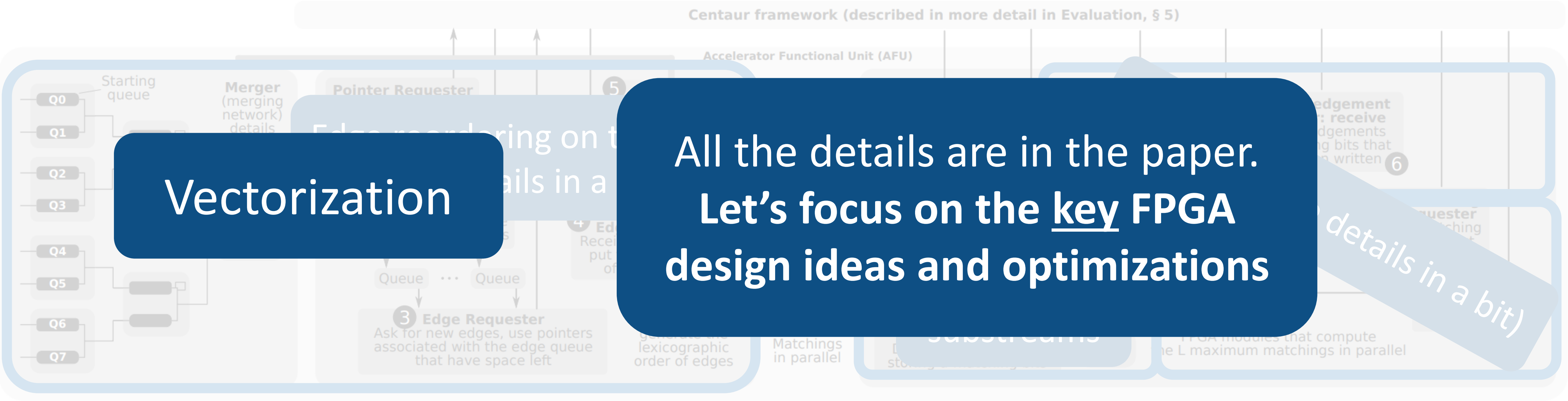


All the details are in the paper.

Let's focus on the <u>key</u> FPGA design ideas and optimizations

Substream-Centric MWM: FPGA optimizations

Blocking



Vectorization

All the details are in the paper.
Let's focus on the key FPGA design ideas and optimizations

Prefetching

Pipelining

Substream-Centric MWM: FPGA optimizations

Blocking

Vectorization

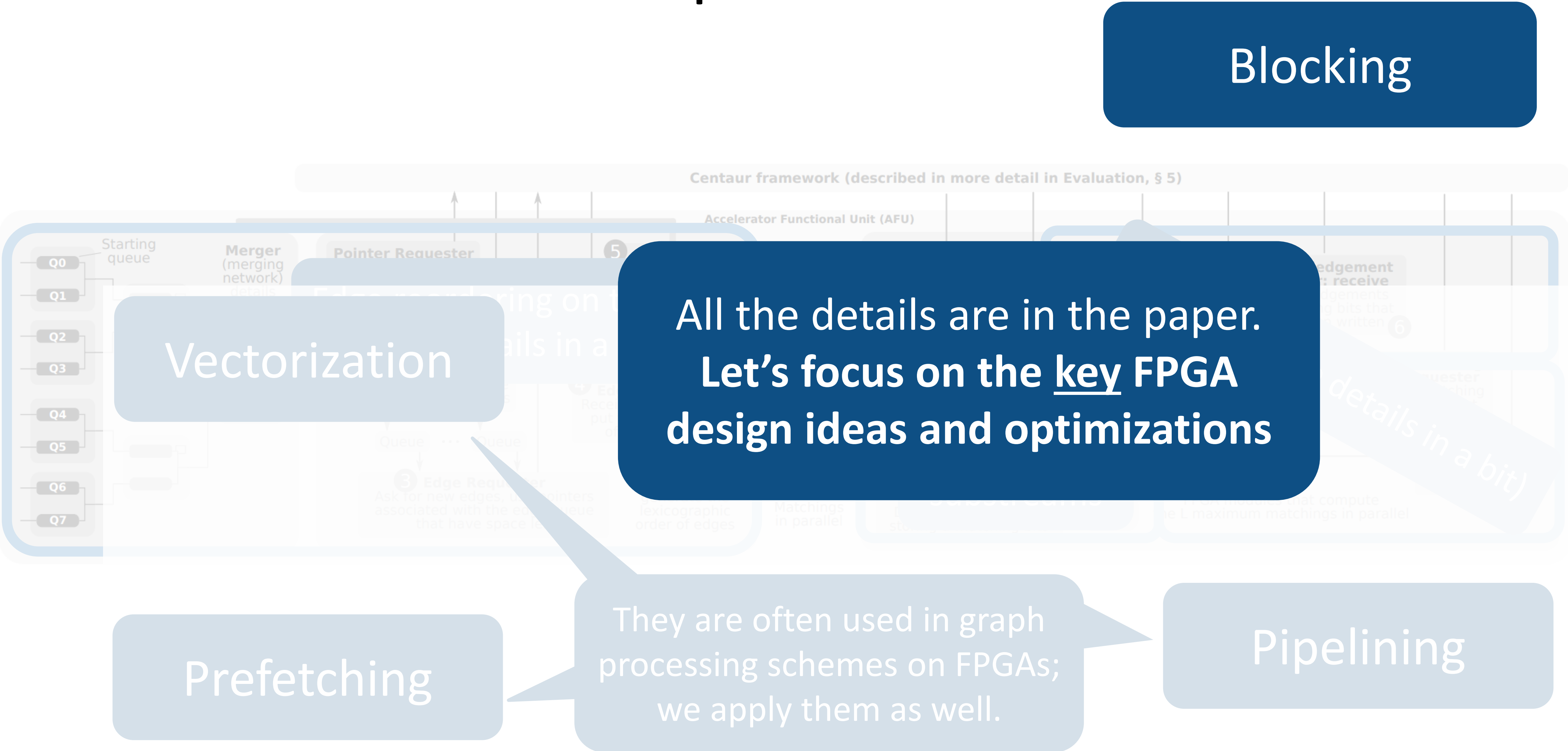
All the details are in the paper.
Let's focus on the <u>key</u> FPGA design ideas and optimizations

Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

Pipelining

Substream-Centric MWM: FPGA optimizations



Substream-Centric MWM: FPGA optimizations

Blocking

Substream-Centric MWM: FPGA optimizations
Blocking

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 0 \\ & & 0 & 1 & 0 & 1 \\ & & & 0 & 1 & 0 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix}$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations
Blocking

$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & & 0 & 1 & 0 & 1 & 0 \\
2 & & & 0 & 1 & 0 & 1 \\
3 & & & & 0 & 1 & 0 \\
4 & & & & & 0 & 1 \\
5 & & & & & & 0
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

Column IDs correspond to vertex IDs

Row IDs correspond to vertex IDs

$$
\begin{array}{cc}
 & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0 \\
 & & 0 & 1 & 0 & 1 \\
 & & & 0 & 1 & 0 \\
 & & & & 0 & 1 \\
 & & & & & 0
\end{bmatrix}
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

Column IDs correspond to vertex IDs

Row IDs correspond to vertex IDs

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 0 \\ & & 0 & 1 & 0 & 1 \\ & & & 0 & 1 & 0 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \end{array}$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

Column IDs correspond to vertex IDs

$$\begin{array}{cc} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 0 \\ & & 0 & 1 & 0 & 1 \\ & & & 0 & 1 & 0 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \end{array}$$

Row IDs correspond to vertex IDs

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

Column IDs correspond to vertex IDs

$$
\begin{array}{c c}
 & \begin{matrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{matrix} \\
\begin{matrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{matrix} &
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0 \\
 & & 0 & 1 & 0 & 1 \\
 & & & 0 & 1 & 0 \\
 & & & & 0 & 1 \\
 & & & & & 0
\end{bmatrix}
\end{array}
$$

Row IDs correspond to vertex IDs

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

An edge between vertices 0 and 2

Column IDs correspond to vertex IDs

Row IDs correspond to vertex IDs

$$
\begin{array}{c}
\begin{array}{cccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array}
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0 \\
 & & 0 & 1 & 0 & 1 \\
 & & & 0 & 1 & 0 \\
 & & & & 0 & 1 \\
 & & & & & 0
\end{bmatrix}
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

An edge between vertices 0 and 2

Column IDs correspond to vertex IDs

$$
\begin{array}{c}
 \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5}
\end{array}
\begin{array}{c}
\begin{array}{cccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0 \\
 & & 0 & 1 & 0 & 1 \\
 & & & 0 & 1 & 0 \\
 & & & & 0 & 1 \\
 & & & & & 0
\end{bmatrix}
\end{array}
$$

Row IDs correspond to vertex IDs

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

An edge between vertices 0 and 2

Column IDs correspond to vertex IDs

Row IDs correspond to vertex IDs

$$
\begin{array}{cc}
 & \begin{array}{cccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 1 \\
 & 0 & 1 & 0 & 1 & 0 \\
 & & 0 & 1 & 0 & 1 \\
 & & & 0 & 1 & 0 \\
 & & & & 0 & 1 \\
 & & & & & 0
\end{bmatrix}
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

An edge between vertices 0 and 2

Column IDs correspond to vertex IDs

Row IDs correspond to vertex IDs

$$
\begin{array}{c|cccccc|}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & & 0 & 1 & 0 & 1 & 0 \\
2 & & & 0 & 1 & 0 & 1 \\
3 & & & & 0 & 1 & 0 \\
4 & & & & & 0 & 1 \\
5 & & & & & & 0 \\
\hline
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

Blocking

An edge between vertices 0 and 1

An edge between vertices 0 and 2

Column IDs correspond to vertex IDs

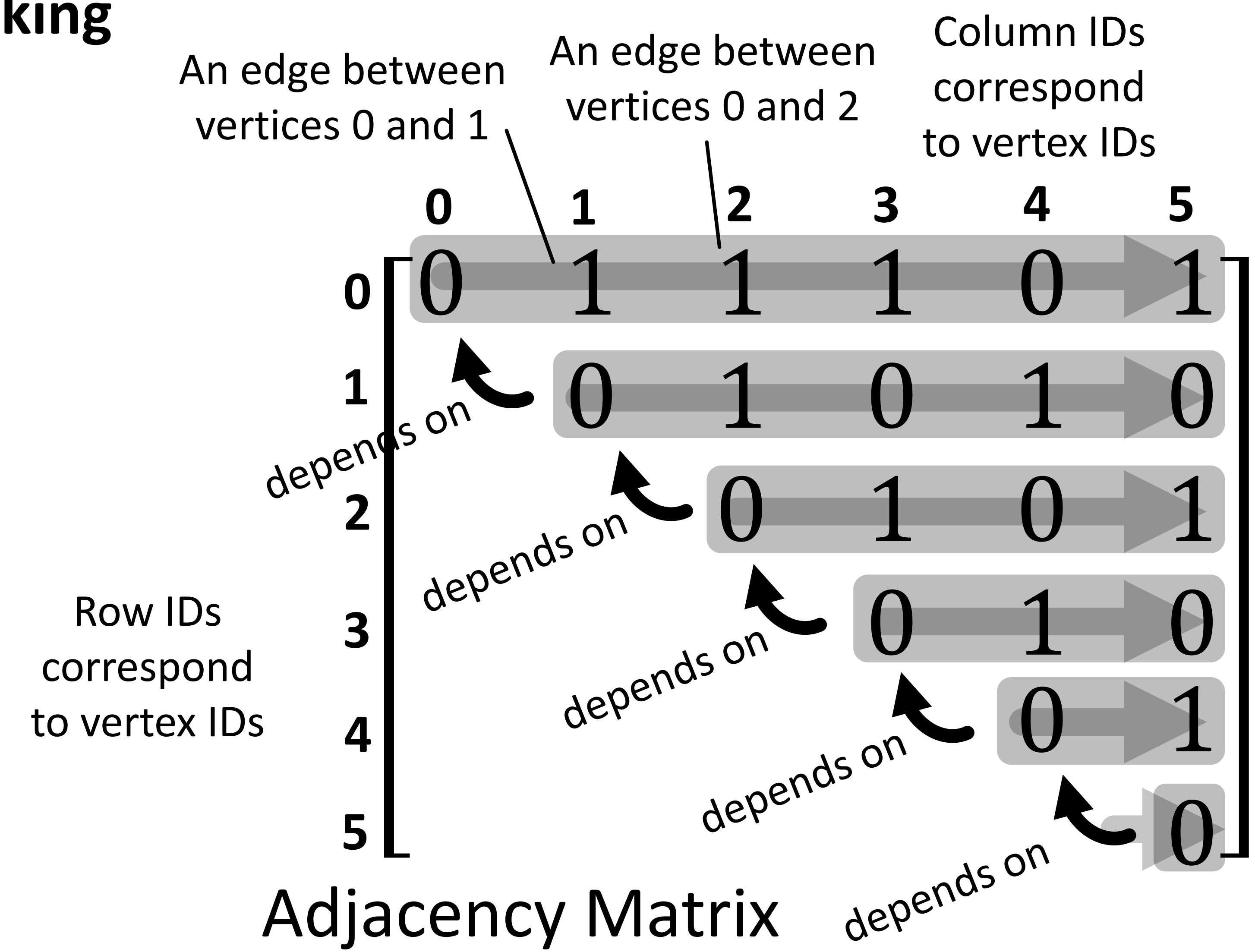
Row IDs correspond to vertex IDs

$$
\begin{array}{c|cccccc}
 & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\
\hline
\mathbf{0} & 0 & 1 & 1 & 1 & 0 & 1 \\
\mathbf{1} & & 0 & 1 & 0 & 1 & 0 \\
\mathbf{2} & & & 0 & 1 & 0 & 1 \\
\mathbf{3} & & & & 0 & 1 & 0 \\
\mathbf{4} & & & & & 0 & 1 \\
\mathbf{5} & & & & & & 0
\end{array}
$$

Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

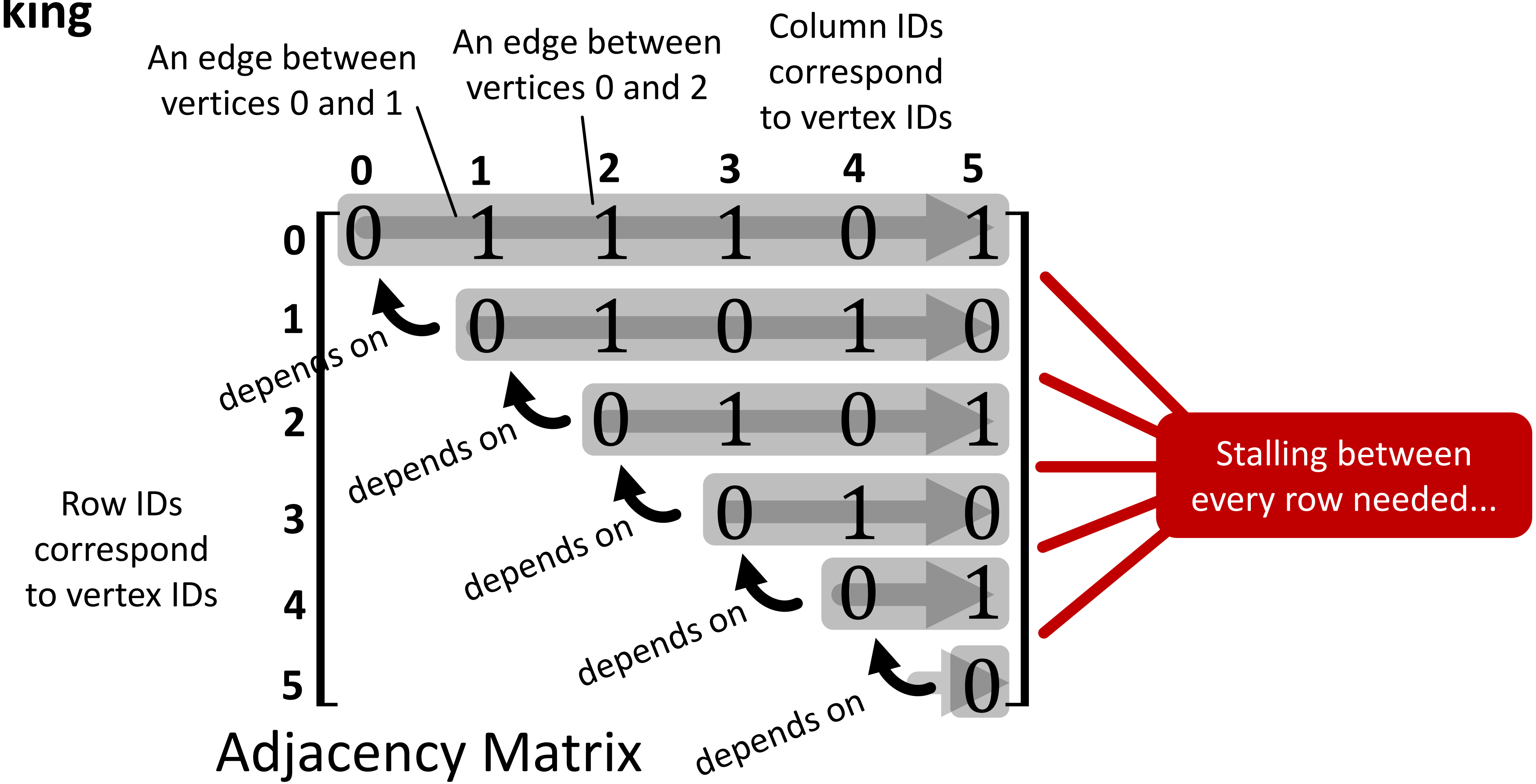
Blocking



Adjacency Matrix

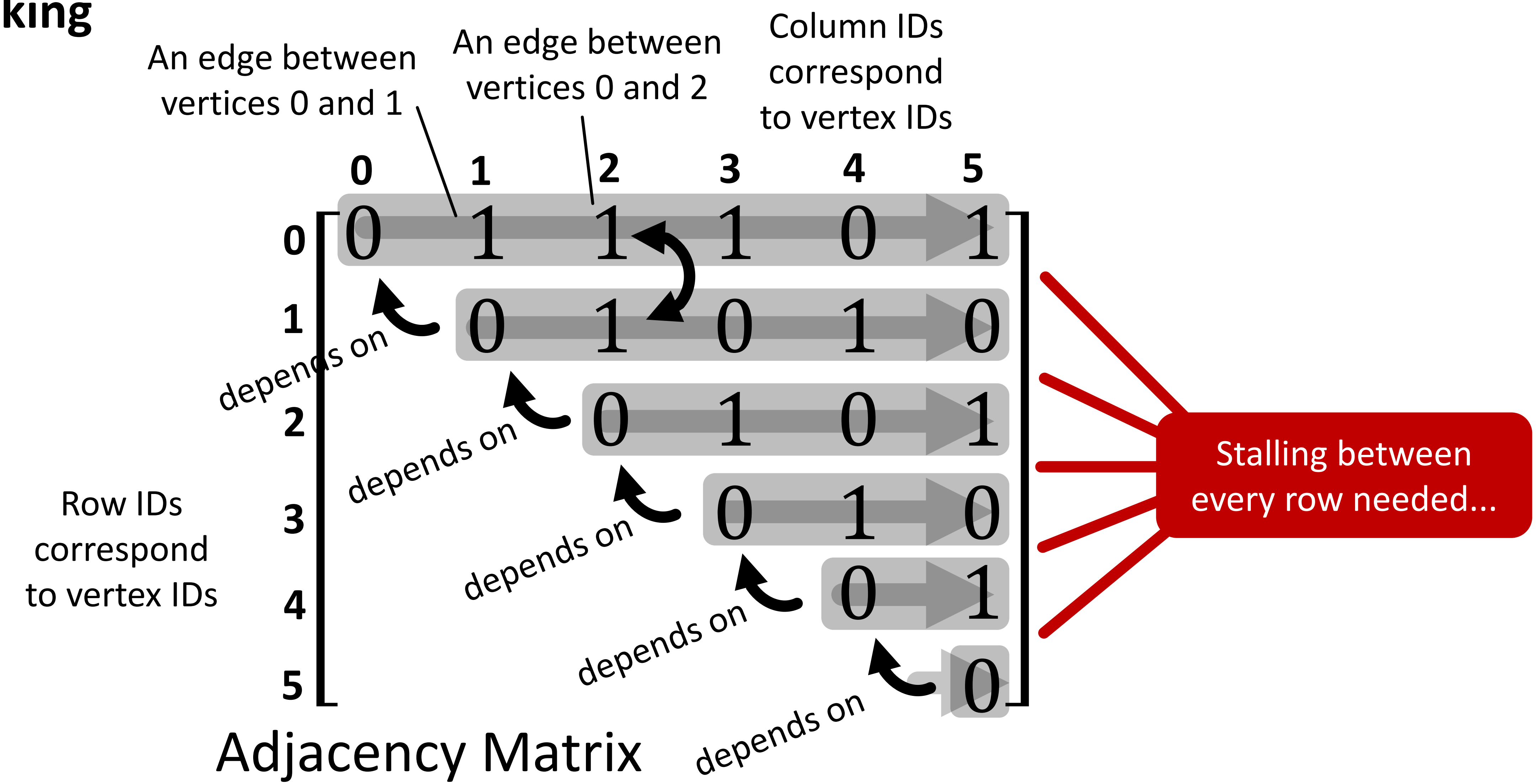
Substream-Centric MWM: FPGA optimizations

Blocking



Substream-Centric MWM: FPGA optimizations

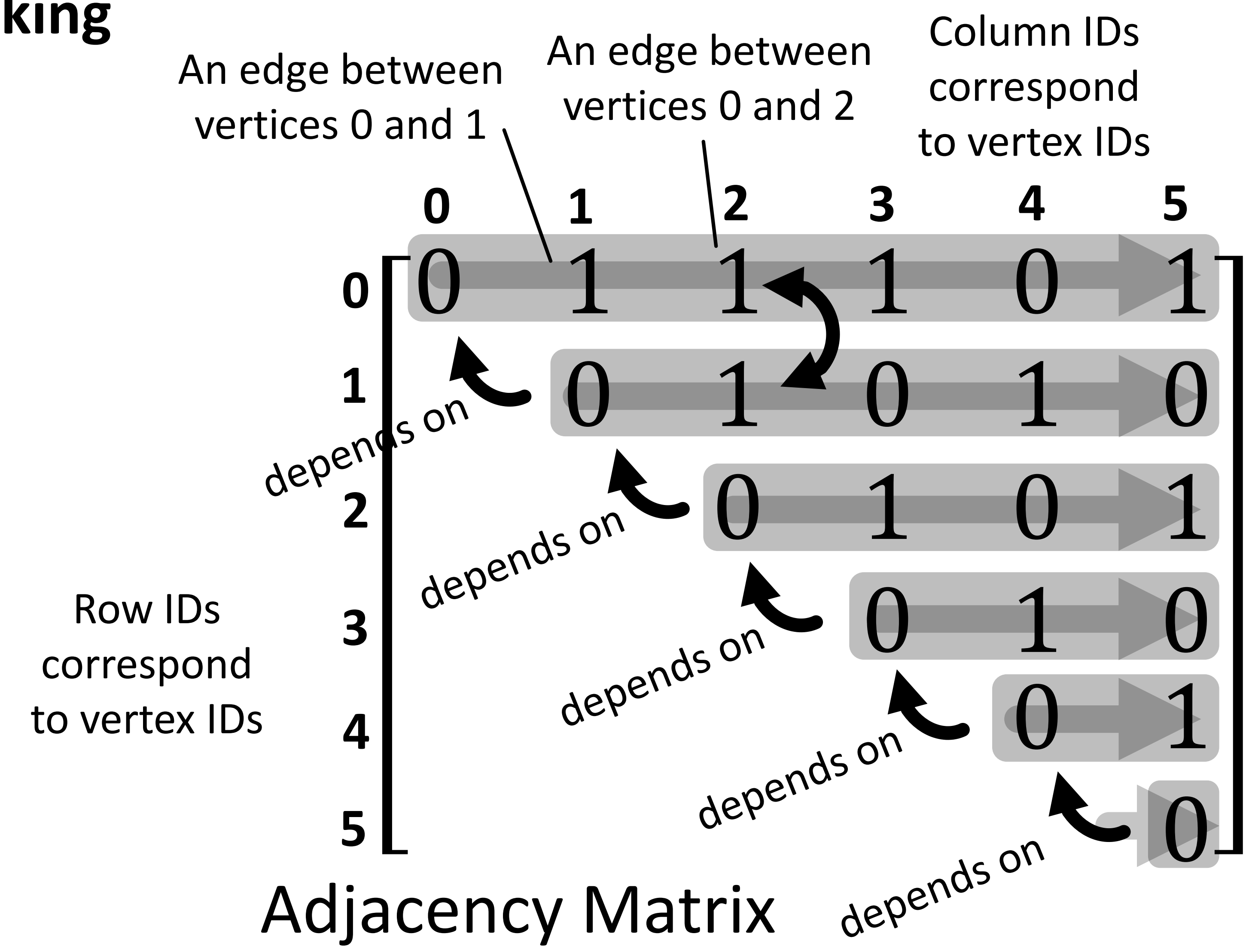
Blocking



Adjacency Matrix

Substream-Centric MWM: FPGA optimizations

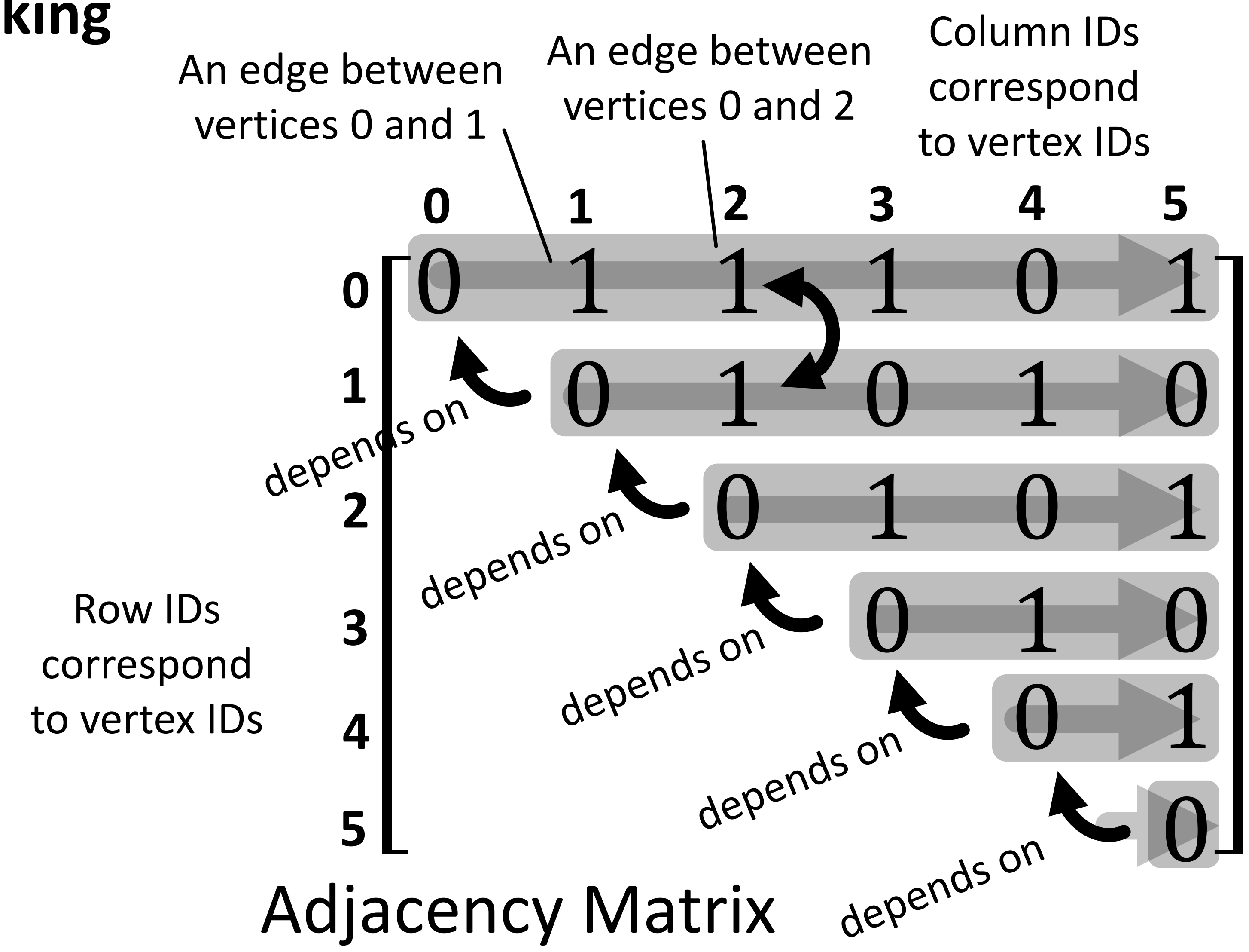
Blocking



Substream-Centric MWM: FPGA optimizations

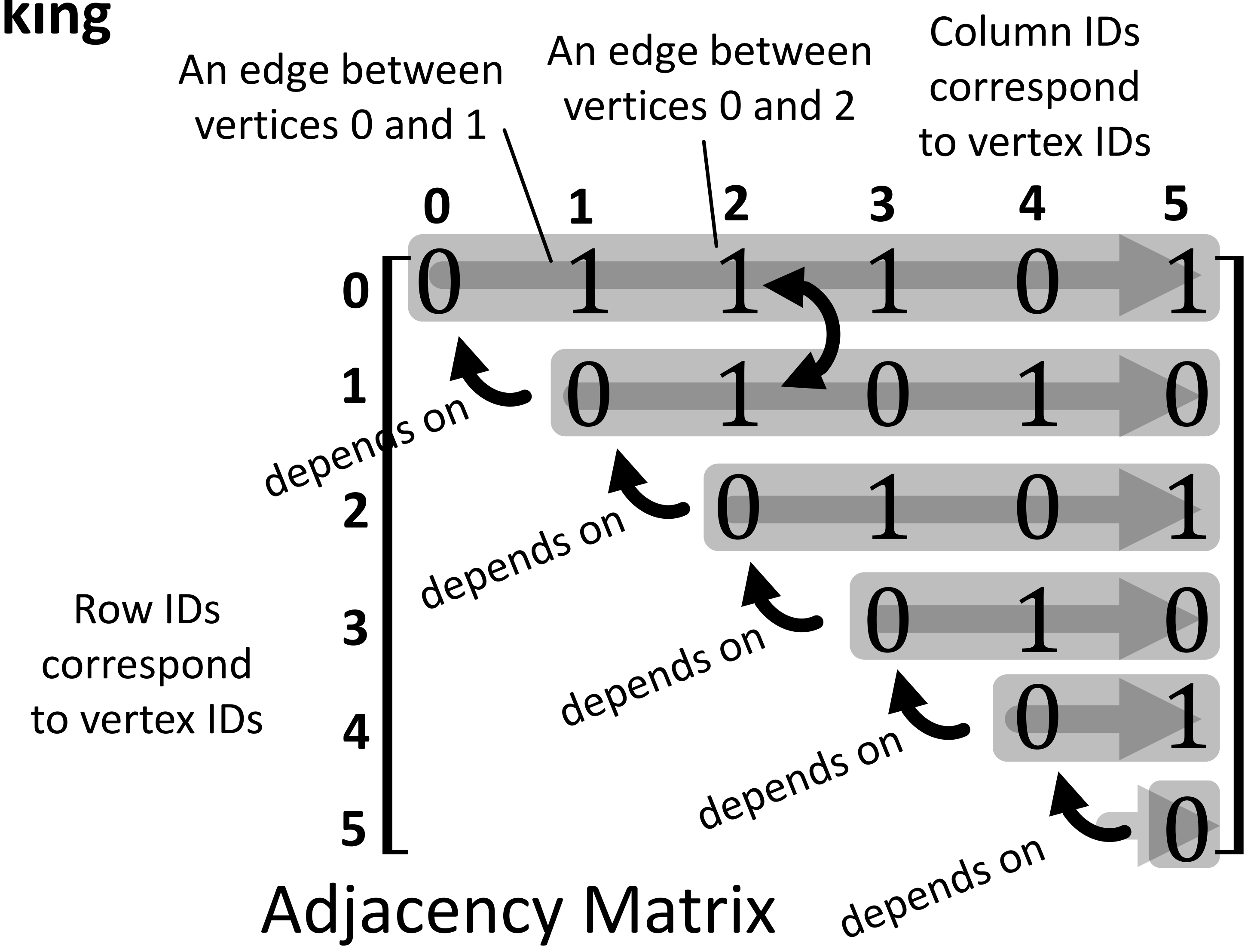
Blocking

Introduce a (tunable) „blocking parameter” K



Substream-Centric MWM: FPGA optimizations

Blocking

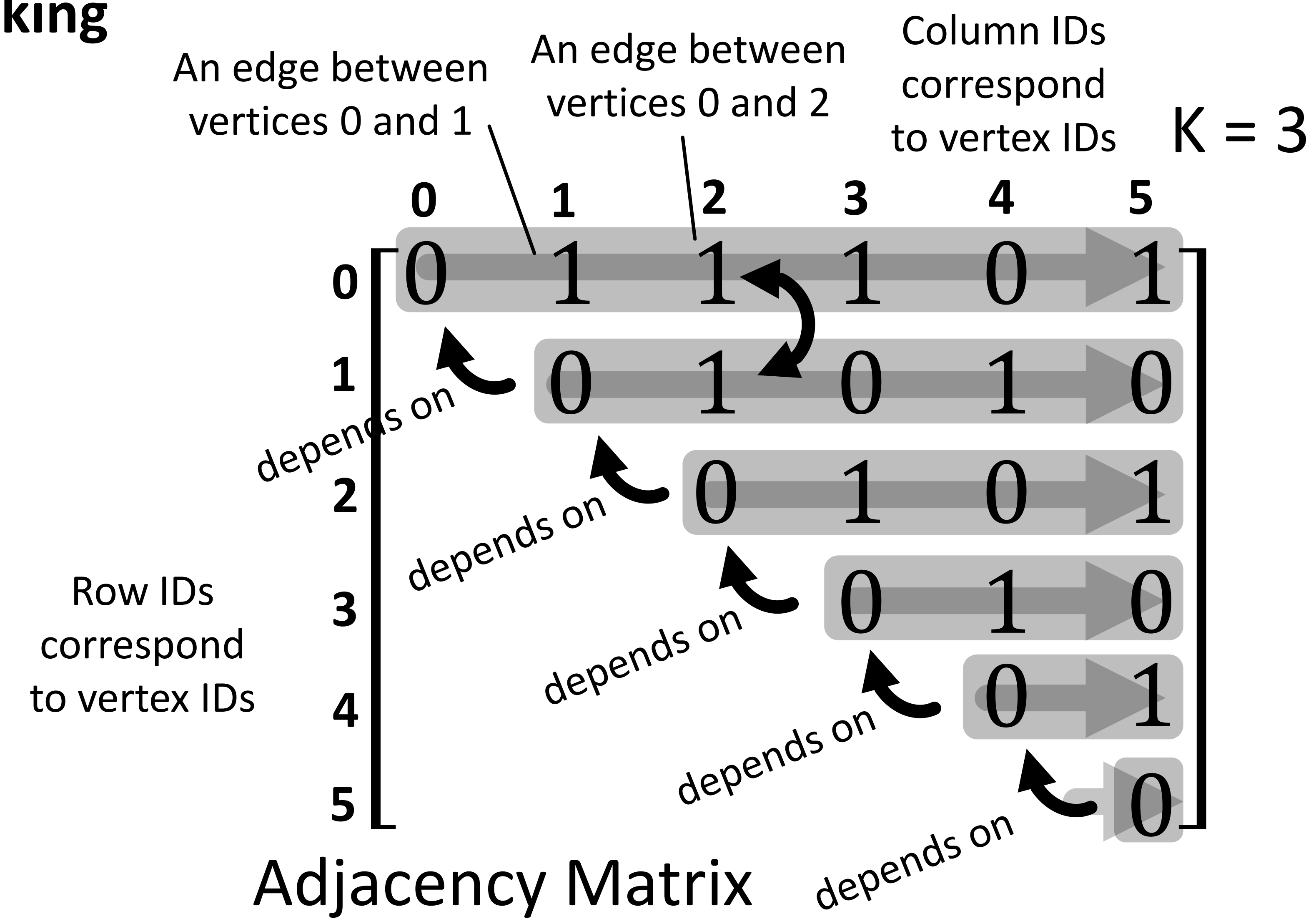


Introduce a (tunable) „blocking parameter” K

K determines how many stalls are allowed

Substream-Centric MWM: FPGA optimizations

Blocking



Introduce a (tunable) „blocking parameter” K

K determines how many stalls are allowed

Substream-Centric MWM: FPGA optimizations
Blocking

K = 3

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 0 \\ & & 0 & 1 & 0 & 1 \\ & & & 0 & 1 & 0 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix}$$

depends on

Adjacency Matrix

Introduce a (tunable) „blocking parameter” K

K determines how many stalls are allowed

Portions of rows are ordered „lexicographically” (i.e., no strict ordering that enforces a stall is required)

Algorithm still (provably) correct

Substream-Centric MWM: FPGA optimizations
Blocking

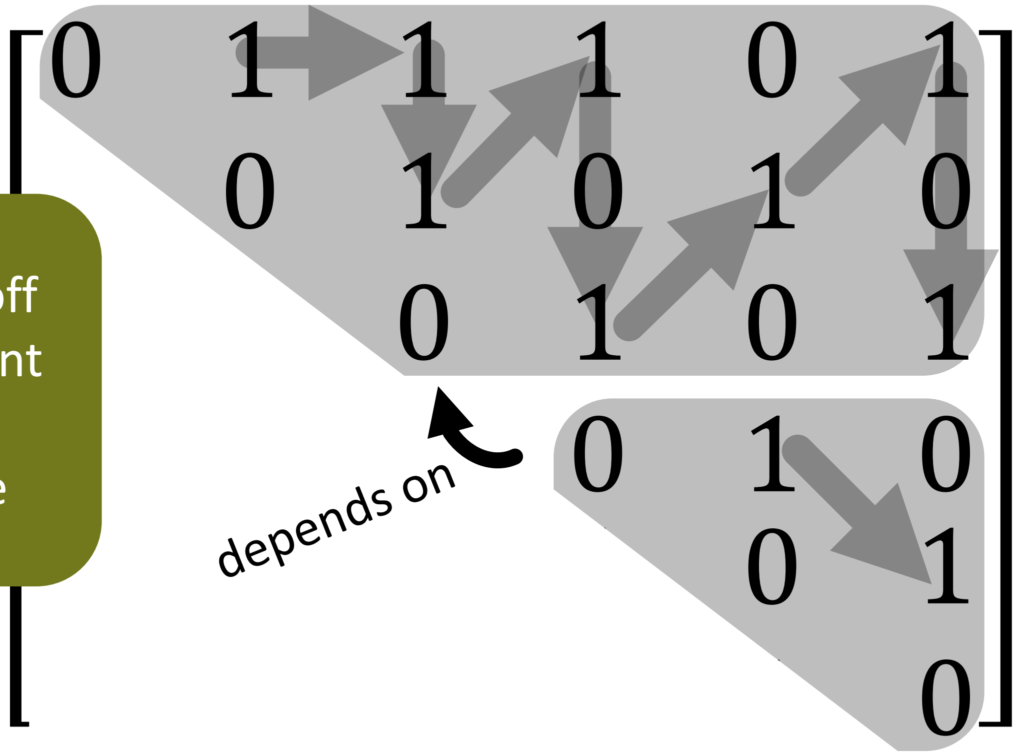
Introduce a (tunable) „blocking parameter” K

K determines how many stalls are allowed

Portions of rows are ordered „lexicographically” (i.e., no strict ordering that enforces a stall is required)

K = 3

K is **tunable**: it controls the tradeoff between the amount of the used FPGA resources and the performance



Adjacency Matrix

Algorithm still (provably) correct

Research Questions

Use **substream-centric processing** (exposes parallelism, enables easy pipelining, supports approximation)

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism, enables easy pipelining, supports approximation)

Combine the MWM algorithm by Crouch and Stubbs **with a hybrid CPU-FPGA setting**

The proper use of blocking, vectorization, pipelining, prefetching

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism, enables easy pipelining, supports approximation)

Combine the MWM algorithm by Crouch and Stubbs **with a hybrid CPU-FPGA setting**

The proper use of blocking, vectorization, pipelining, prefetching

What is the ultimate performance, power consumption, and the related tradeoffs?

Performance Analysis

Types of machines

Part 5: Evaluation

Performance Analysis

Types of machines

Performance Analysis
Types of machines

CPU: Intel Broadwell
Xeon E5-2680 v4 @3.3 GHz
14 Cores (28 Threads)

Altera Arria 10 @200MHz

Performance Analysis

Types of graphs

Performance Analysis

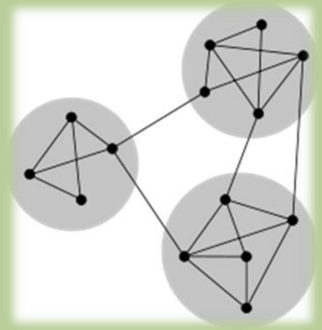
Types of graphs

Synthetic graphs

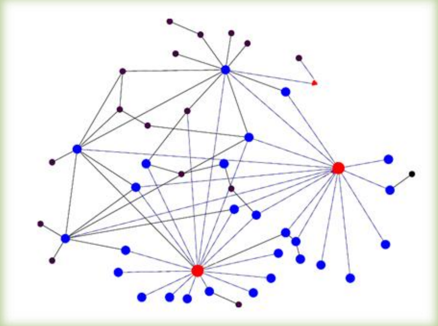
Performance Analysis

Types of graphs

Synthetic graphs



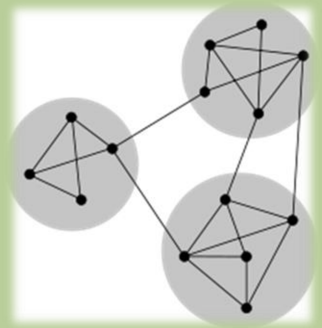
Kronecker [1]



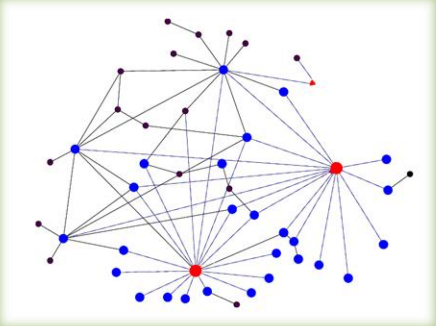
[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.

Performance Analysis
Types of graphs

Synthetic graphs



Kronecker [1]



Real-world graphs (SNAP [2], KONECT [3], DIMACS [4])

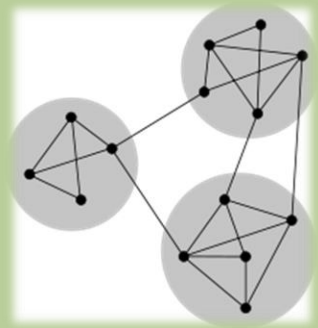
[2] SNAP. https://snap.stanford.edu

[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.

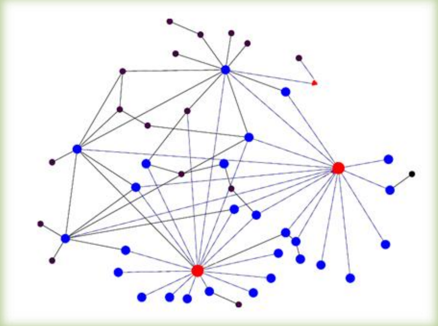
Performance Analysis

Types of graphs

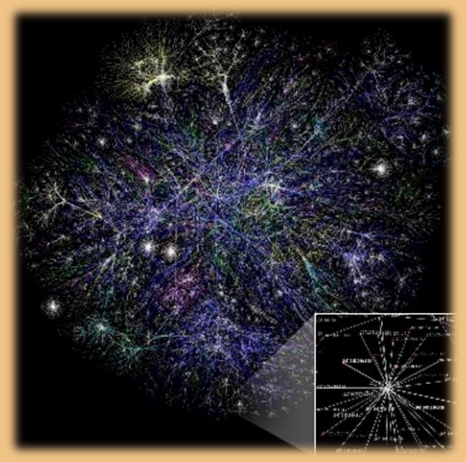
Synthetic graphs



Kronecker [1]



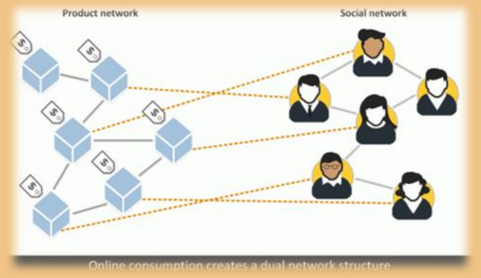
Real-world graphs (SNAP [2], KONECT [3], DIMACS [4])



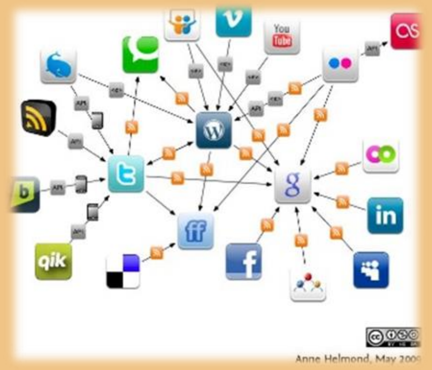
Web graphs

Road networks

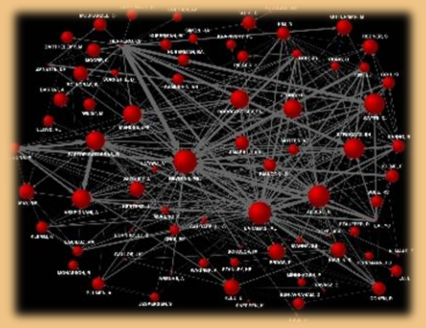
Social networks



Purchase networks



Communication graphs



Citation graphs

[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.
[2] SNAP. https://snap.stanford.edu
[3] KONECT. https://konect.cc
[4] DIMACS Challenge

Performance Analysis

Algorithms

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.
[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.

PERFORMANCE ANALYSIS
ALGORITHMS

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Our FPGA design, (4+ε)-approximation

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.
[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.

Performance Analysis

Algorithms

CPU implementations of the original Crouch scheme, (4+ε)-approximation

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Our FPGA design, (4+ε)-approximation

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.

Performance Analysis

Algorithms

CPU implementations of the original Crouch scheme, (4+ε)-approximation

State-of-the-art MWM algorithm, space-optimal, **time-optimal (O(m))**, (2+ε)-approximation

Algorithm	Pl orm
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Our FPGA design, (4+ε)-approximation

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.
[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.

Performance Analysis

Algorithms

CPU implementations of the original Crouch scheme, (4+ε)-approximation

State-of-the-art MWM algorithm, space-optimal, **time-optimal (O(m))**, (2+ε)-approximation

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Our FPGA design, (4+ε)-approximation

We test both CPU and hybrid (FPGA+CPU) platforms

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.

Performance Analysis

Various Graphs

Parameters:
Blocking size (K) = 32,
#Substreams (L) = 64
#Threads = 4, ε = 0.1

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

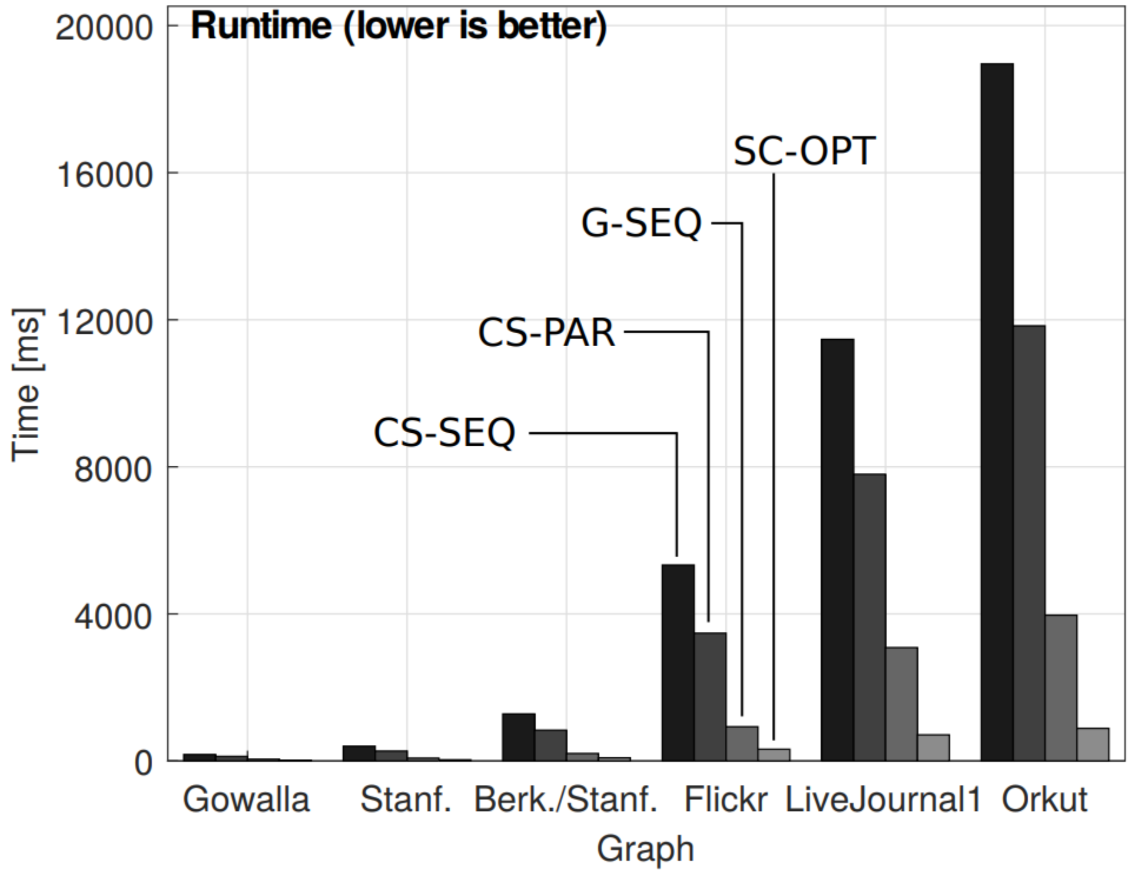
Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770

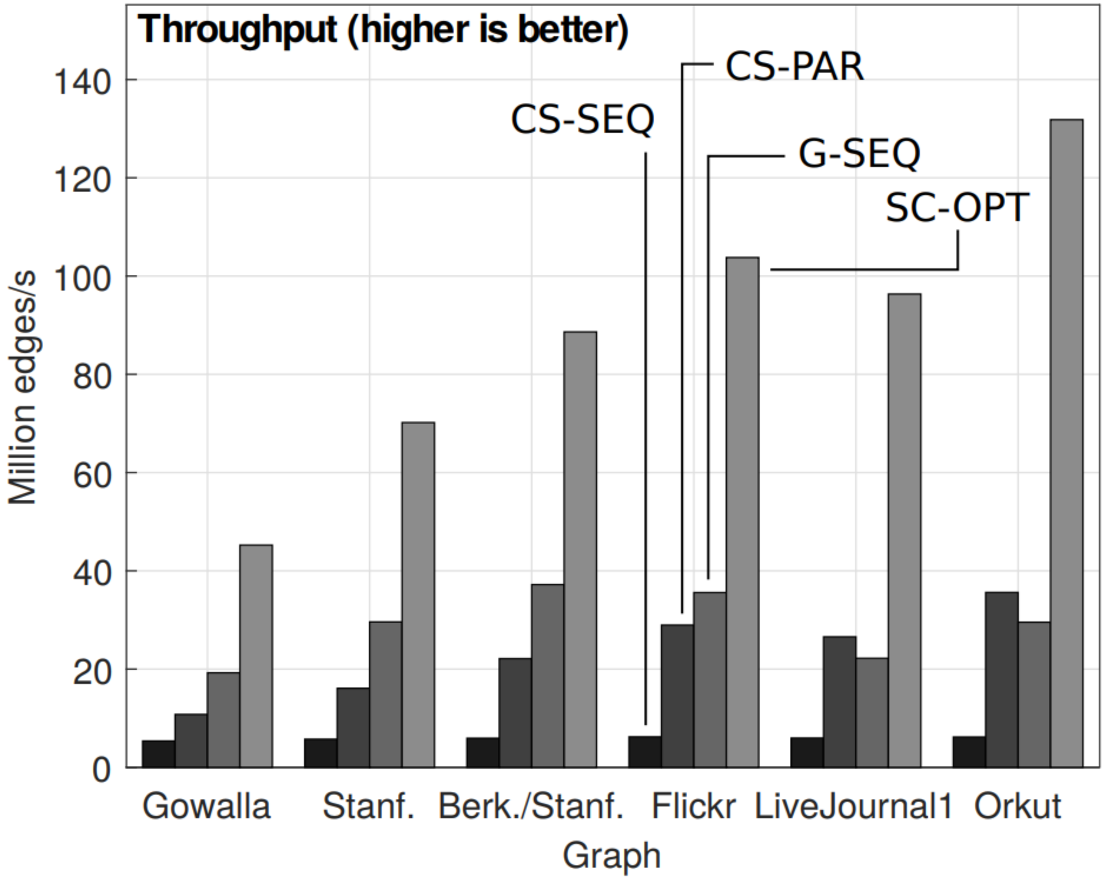
Performance Analysis
Various Graphs

Parameters:
Blocking size (K) = 32,
#Substreams (L) = 64
#Threads = 4, ε = 0.1

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770



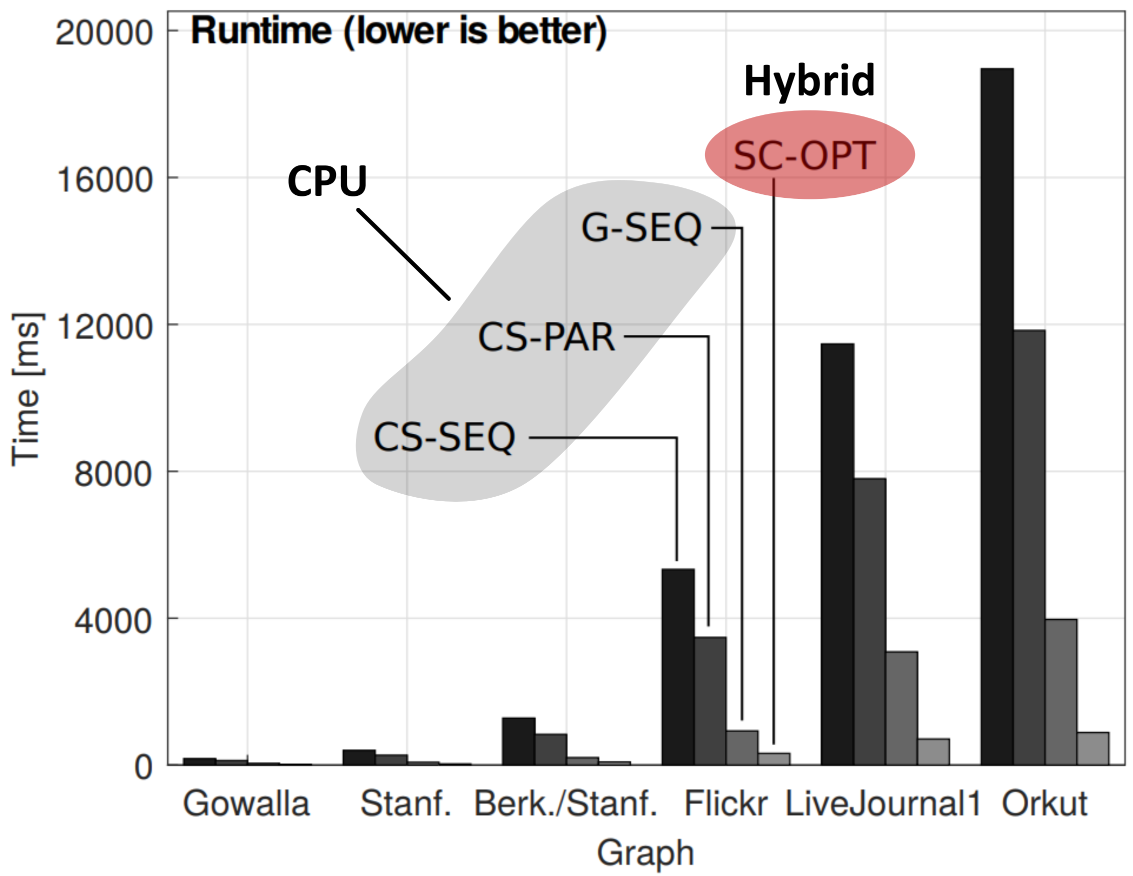


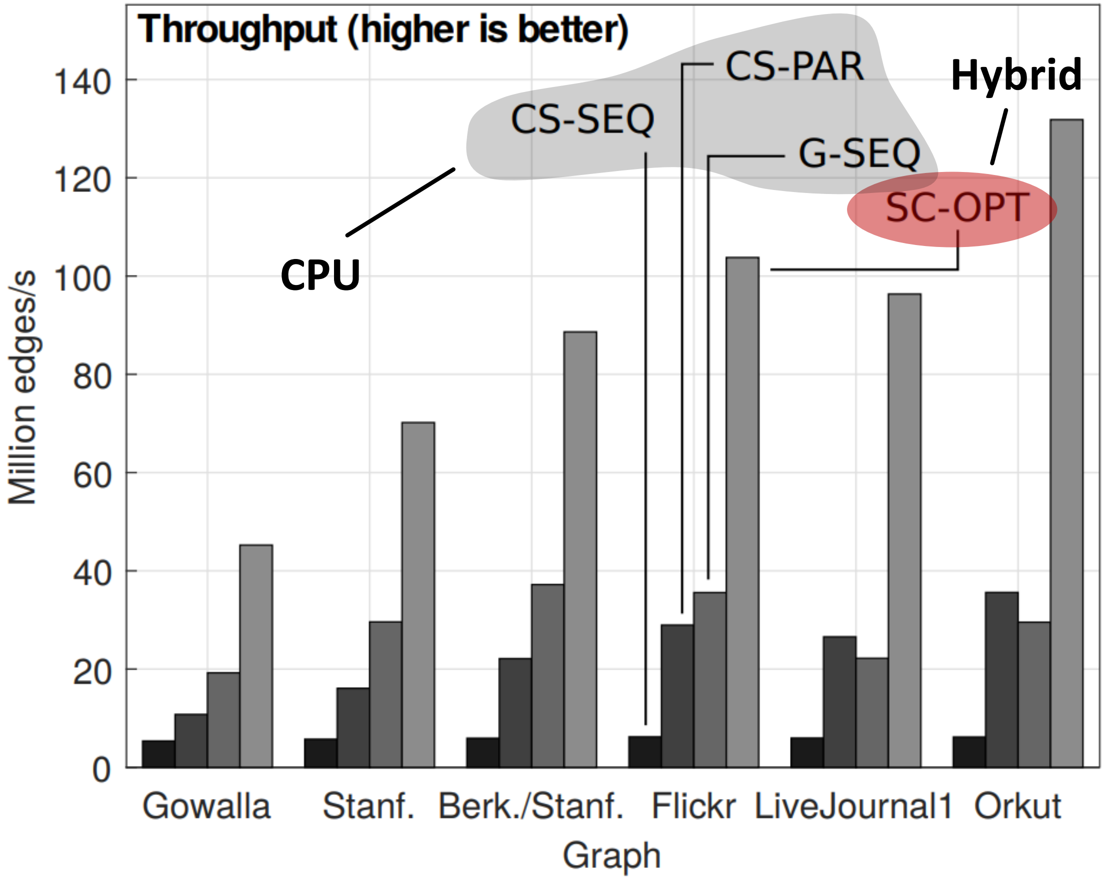
Performance Analysis
Various Graphs

Parameters:
Blocking size (K) = 32,
#Substreams (L) = 64
#Threads = 4, ε = 0.1

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770





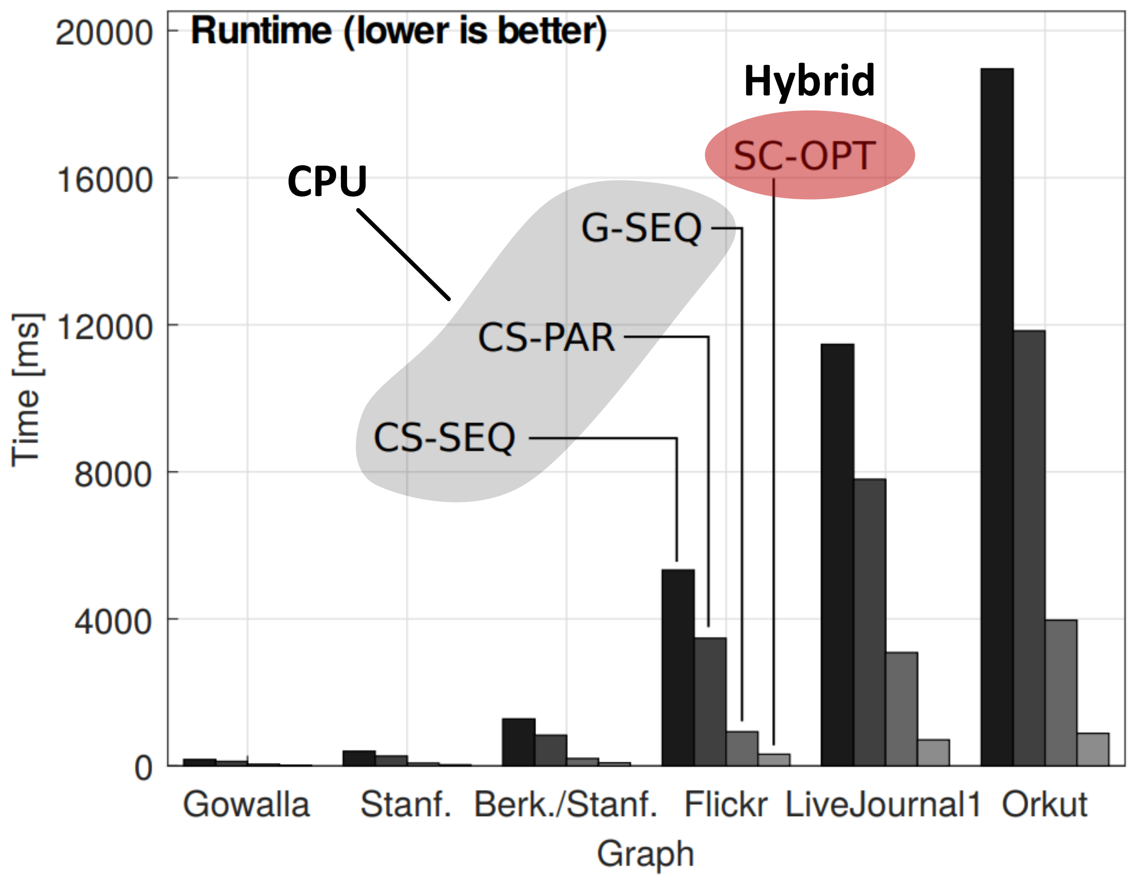
Performance Analysis
Various Graphs

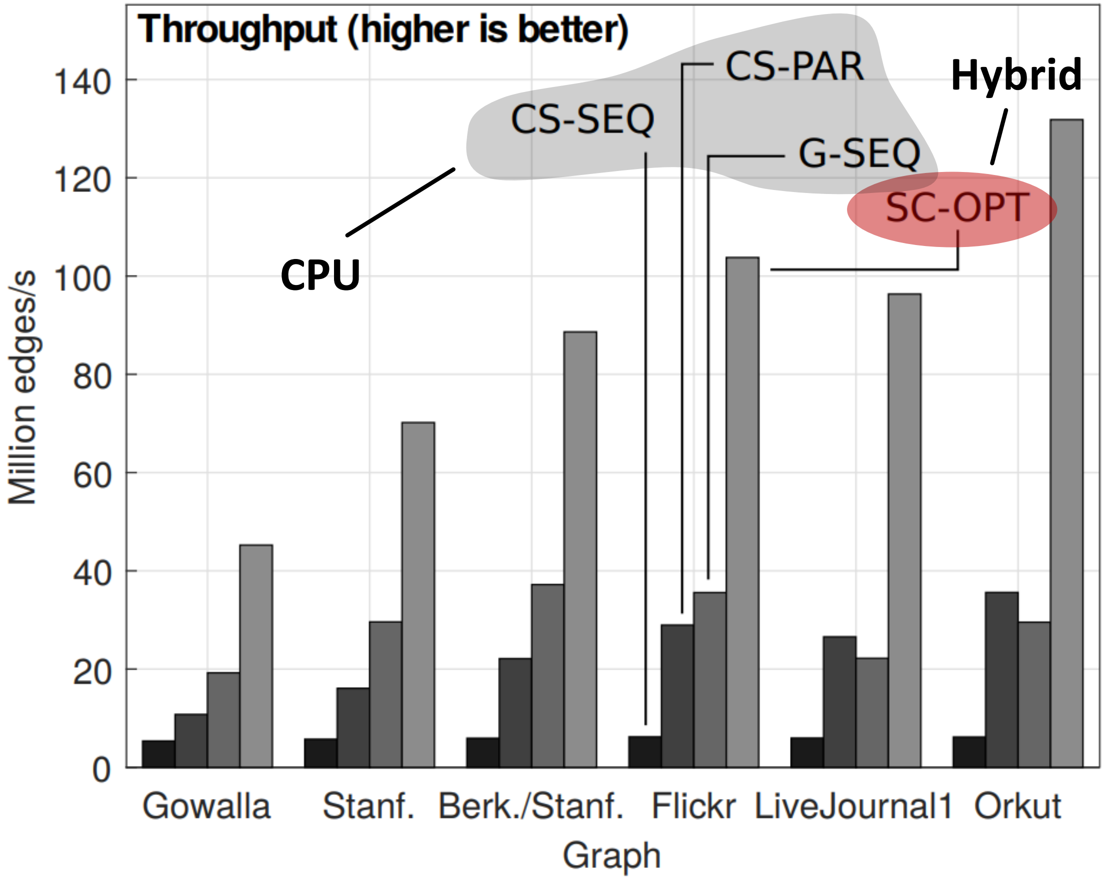
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size (K) = 32,
#Substreams (L) = 64
#Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \dots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770





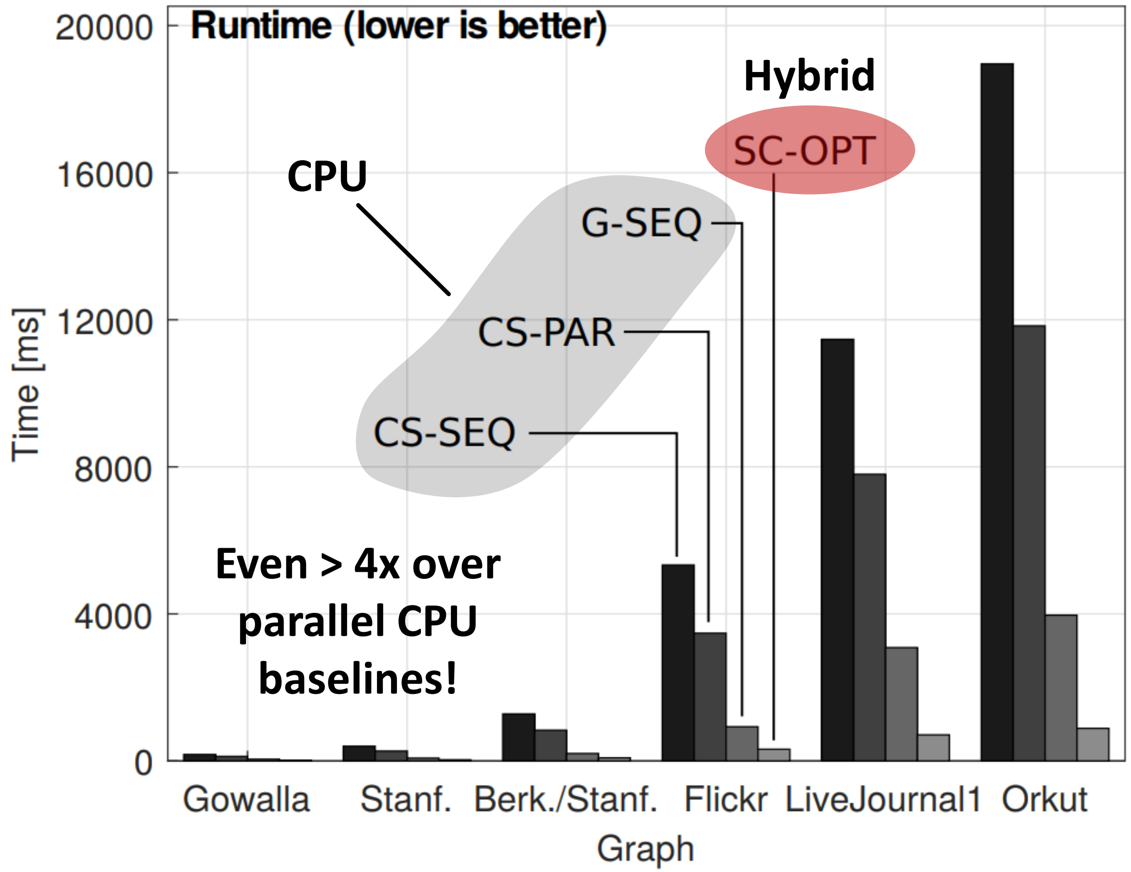
Performance Analysis
Various Graphs

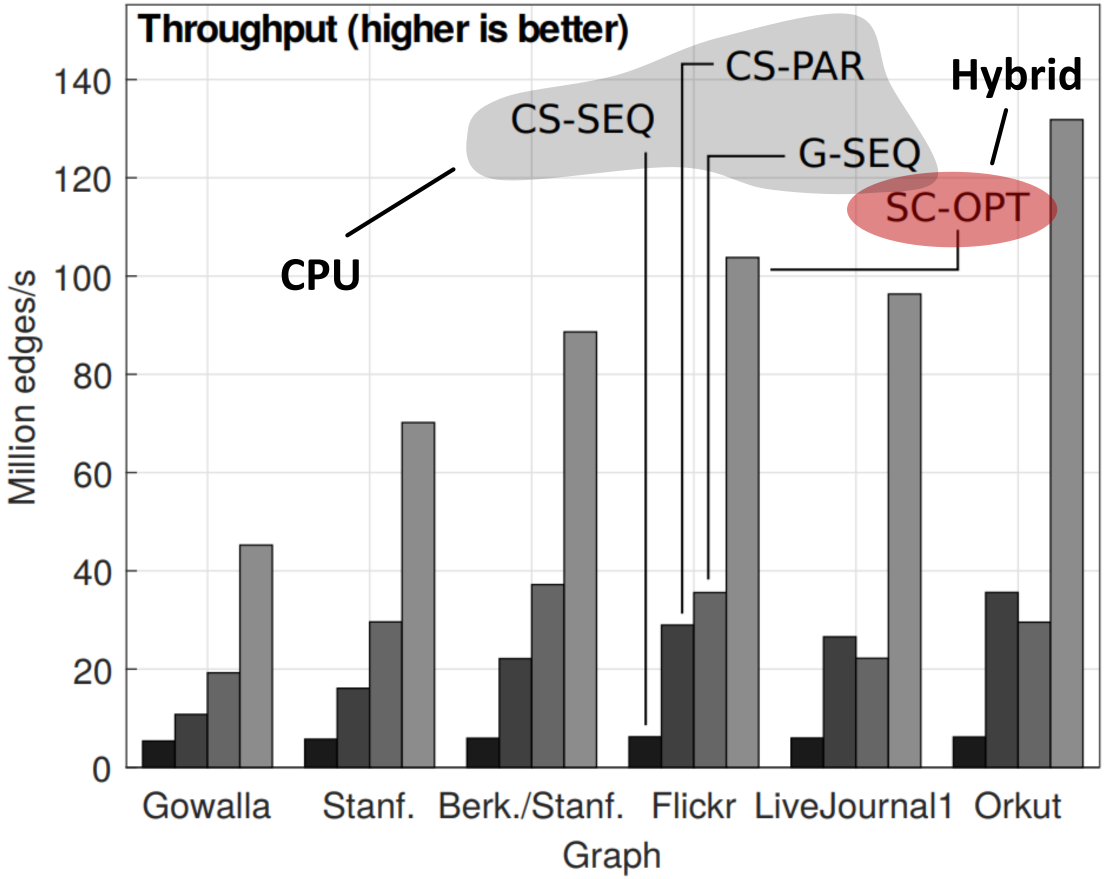
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size (K) = 32,
#Substreams (L) = 64
#Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770





Performance Analysis

Approximation (Accuracy)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

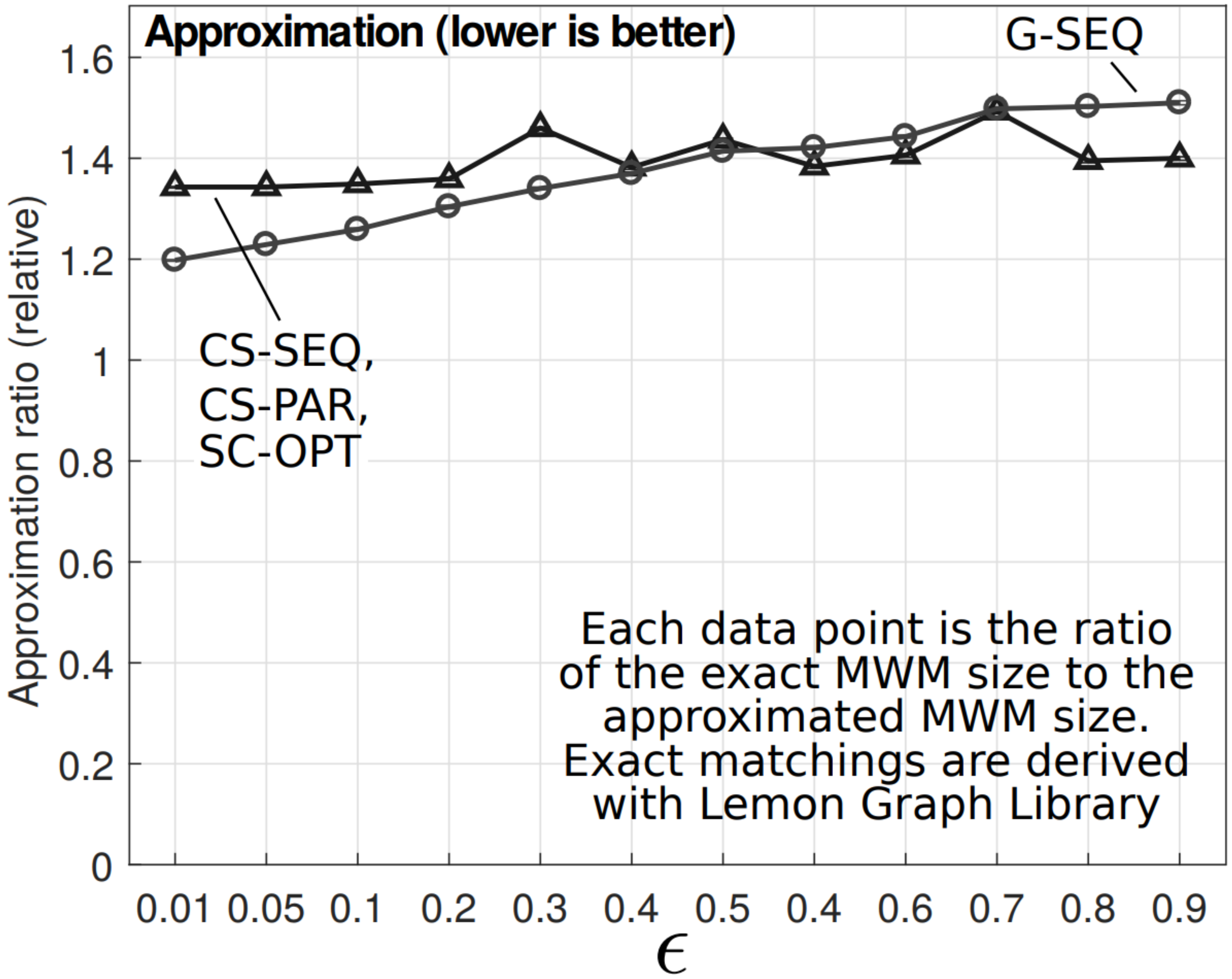
Parameters:
#Substreams (L) = 128,
Blocking size (K) = 32,
#threads = 4, #edges = 8M
(Kronecker)

Performance Analysis

Approximation (Accuracy)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
#Substreams (L) = 128,
Blocking size (K) = 32,
#threads = 4, #edges = 8M
(Kronecker)

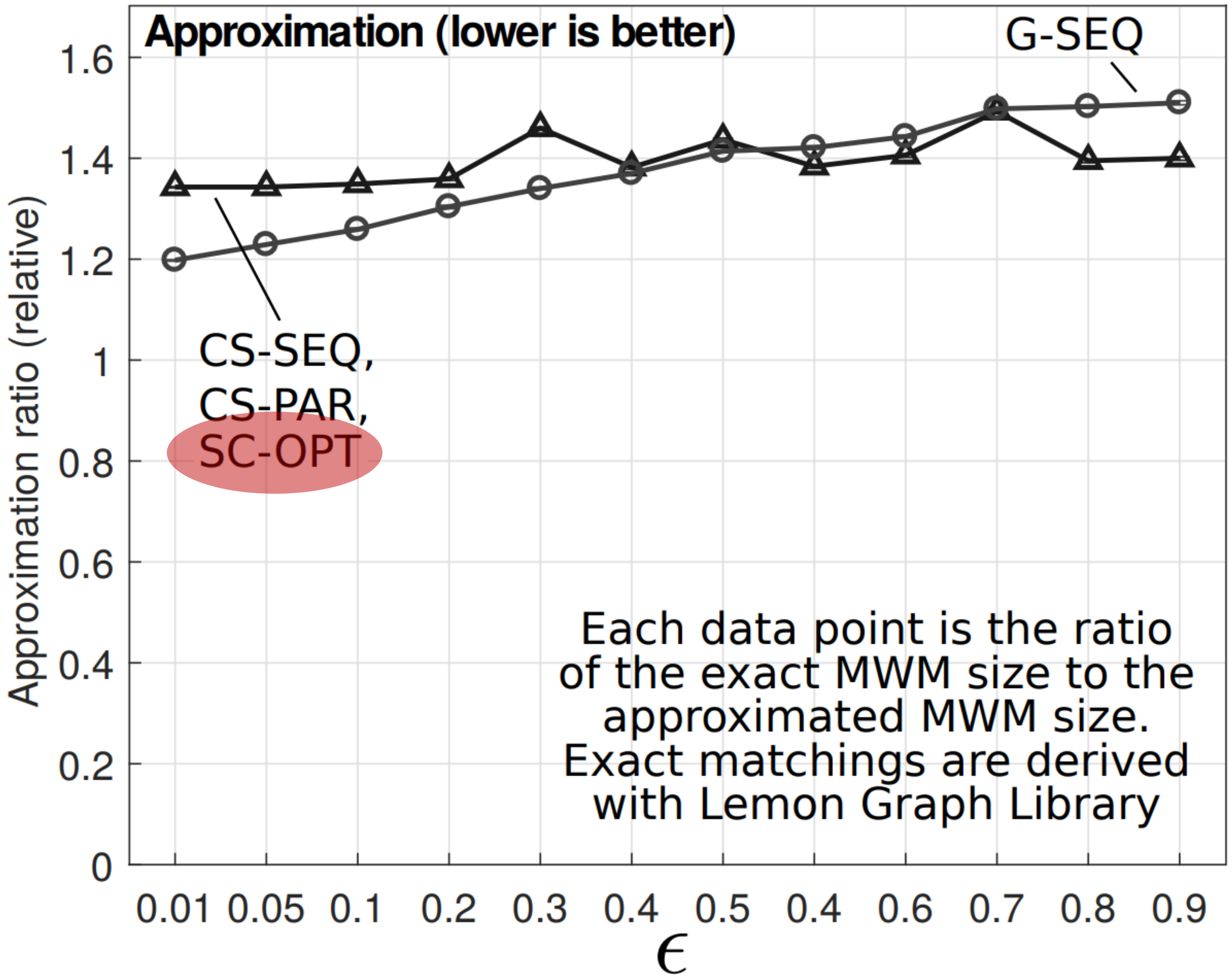


Performance Analysis

Approximation (Accuracy)

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Parameters:
#Substreams (L) = 128,
Blocking size (K) = 32,
#threads = 4, #edges = 8M
(Kronecker)

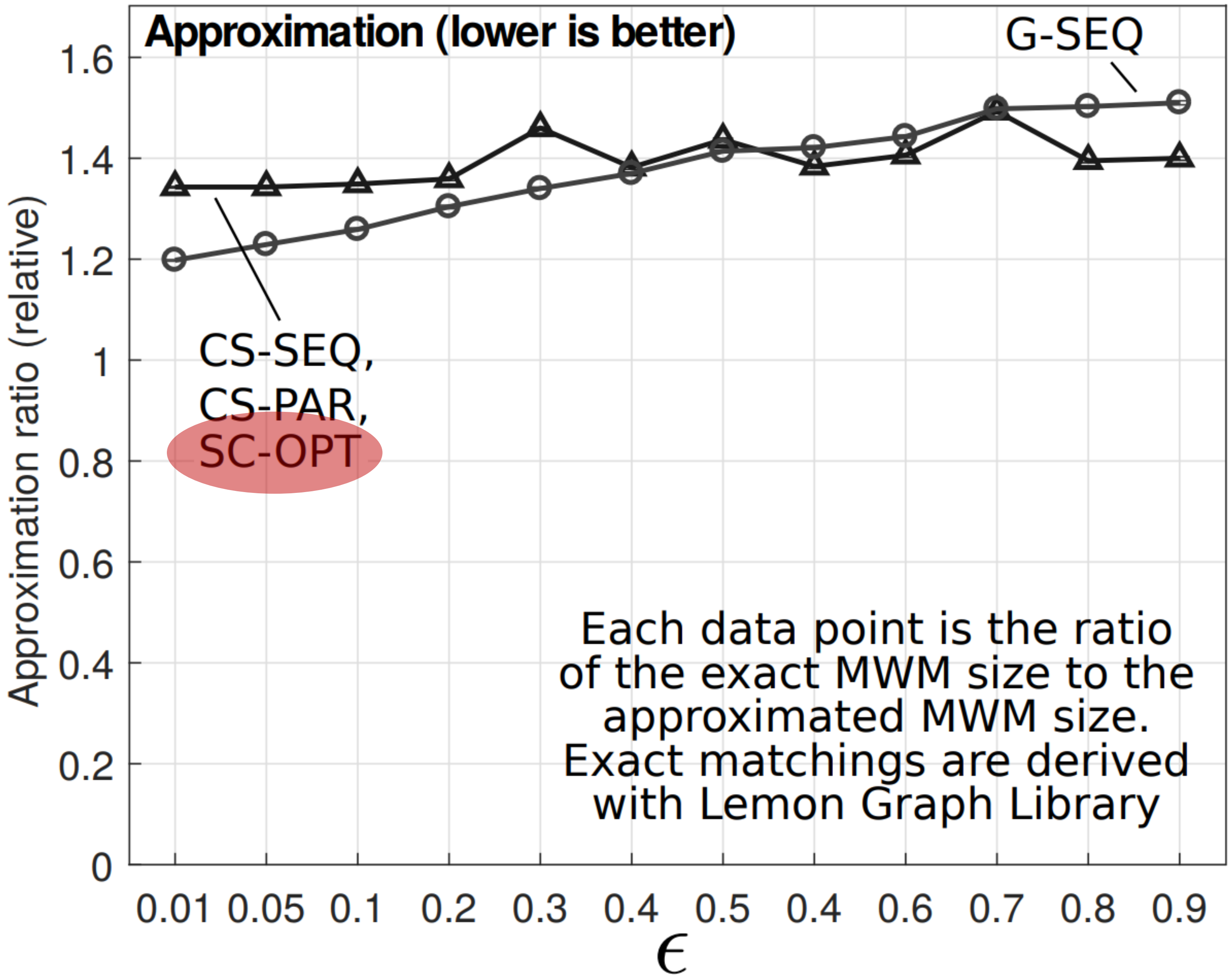


Performance Analysis
Approximation (Accuracy)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
#Substreams (L) = 128,
Blocking size (K) = 32,
#threads = 4, #edges = 8M
(Kronecker)

SC-OPT is comparable to the (2+ε)-approximation by Ghaffari et al.



Performance Analysis

Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		**Platform**
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Performance Analysis

Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

Performance Analysis

Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

Performance Analysis

Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation

Performance Analysis

Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric	(SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation

FPGA Algorithm	Parameters	Used BRAM	Used ALMs
SC-OPT	$K = 32, L = 512$	11.5 MBit (21%)	151,998 (32%)
SC-OPT	$K = 256, L = 128$	24.8 MBit (45%)	350,556 (82%)

Performance Analysis
Energy Consumption, Resource Utilization

Parameters:
L: #Substreams (pipelines),
K: Blocking size,
T: #CPU threads

Algorithm		Platform
Crouch et al. [1]	Sequential (CS-SEQ)	CPU
Crouch et al. [1]	Parallel (CS-PAR)	CPU
Ghaffari [2]	Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)		Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation

FPGA Algorithm	Parameters	Used BRAM	Used ALMs
SC-OPT	$K = 32, L = 512$	11.5 MBit (21%)	151,998 (32%)
SC-OPT	$K = 256, L = 128$	24.8 MBit (45%)	350,556 (82%)

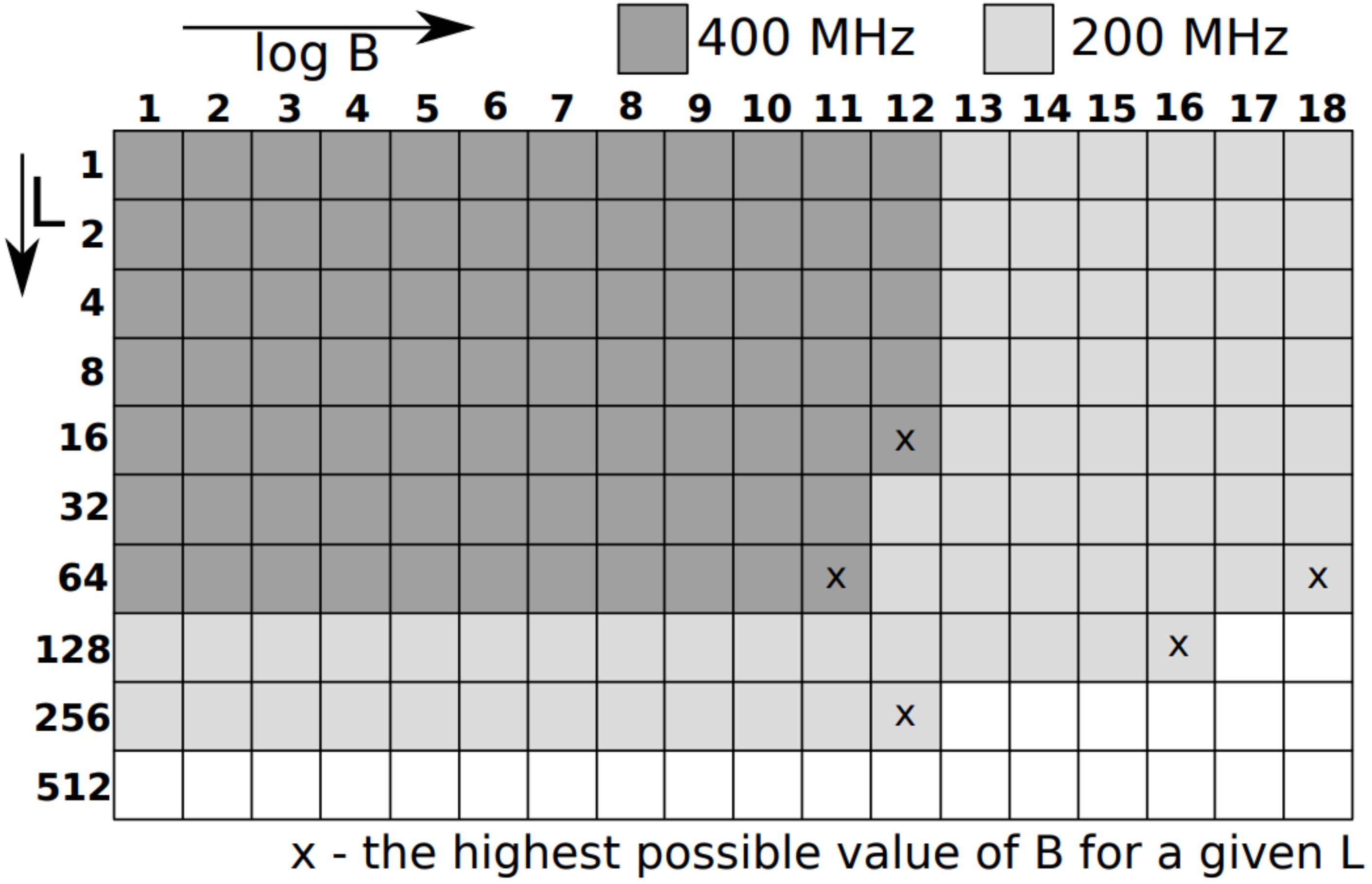
Blocking needs more resources (but is **tunable**!)

Performance Analysis

Design Space Exploration

Performance Analysis
Design Space Exploration

B – BRAM size allocated for matching data structures,
L – number of substreams (pipelines)

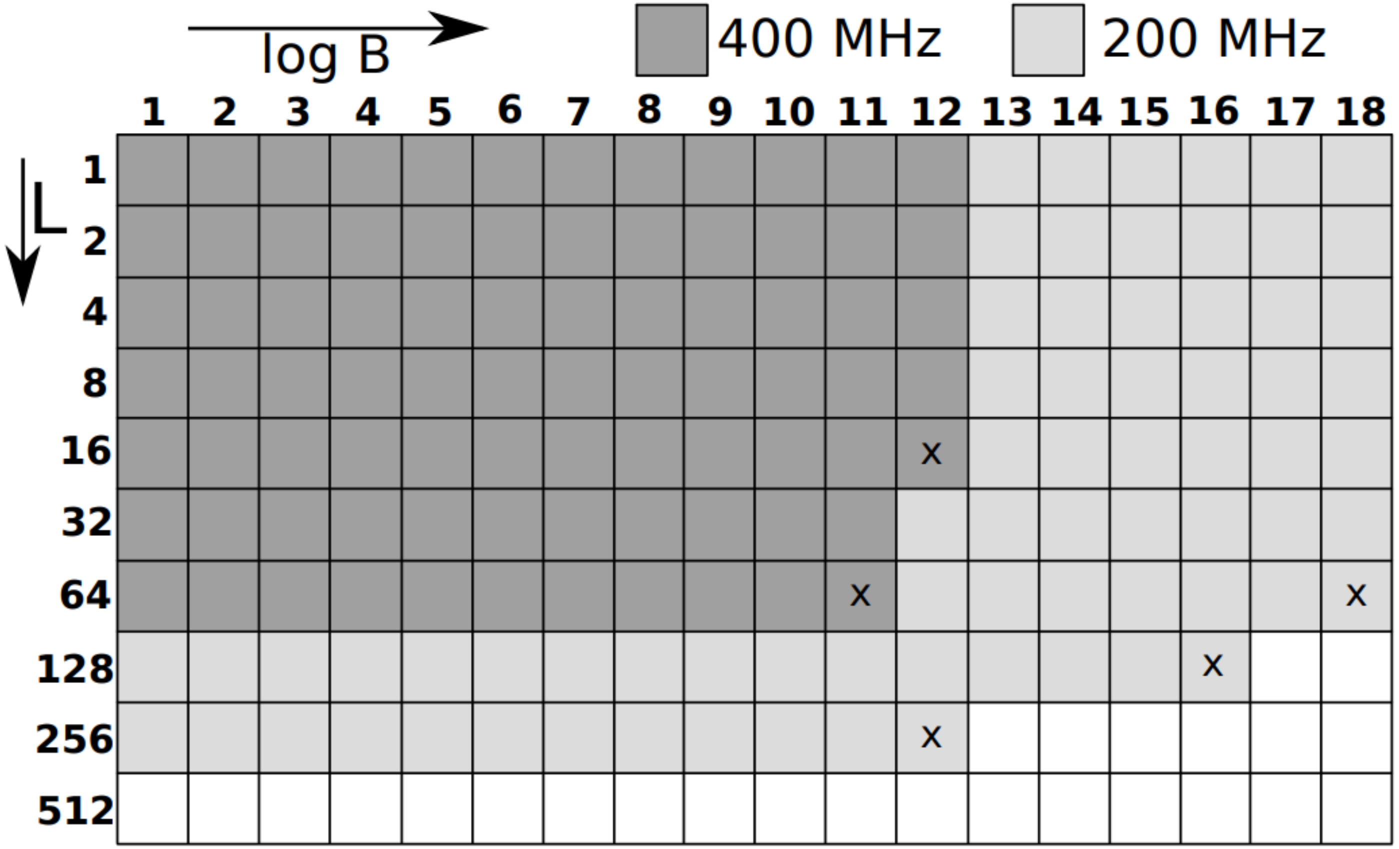


Performance Analysis

Design Space Exploration

B – BRAM size allocated for matching data structures,
L – number of substreams (pipelines)

Addition complexity grows linearly with *L*



x - the highest possible value of B for a given L

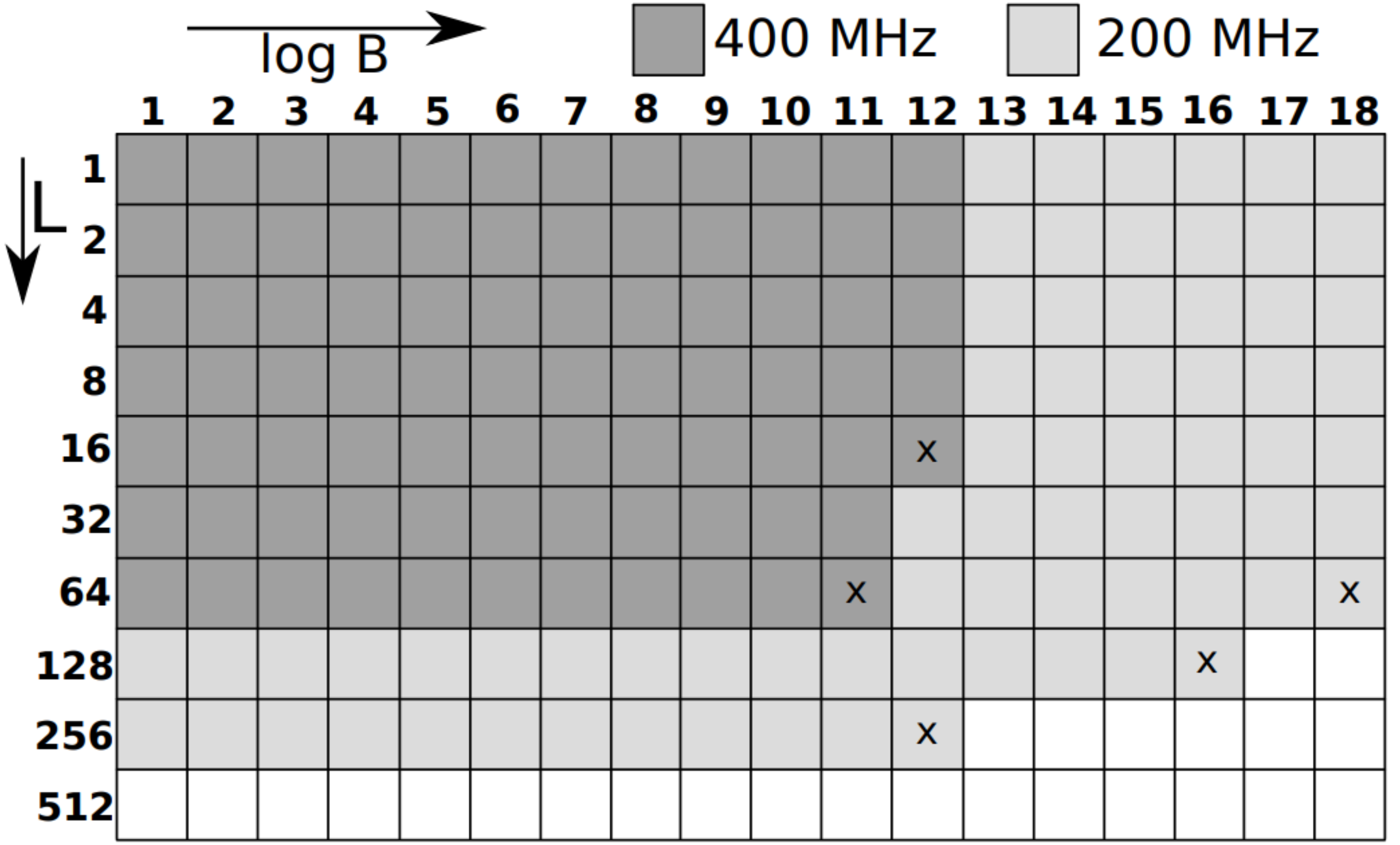
Performance Analysis

Design Space Exploration

B – BRAM size allocated for matching data structures,
L – number of substreams (pipelines)

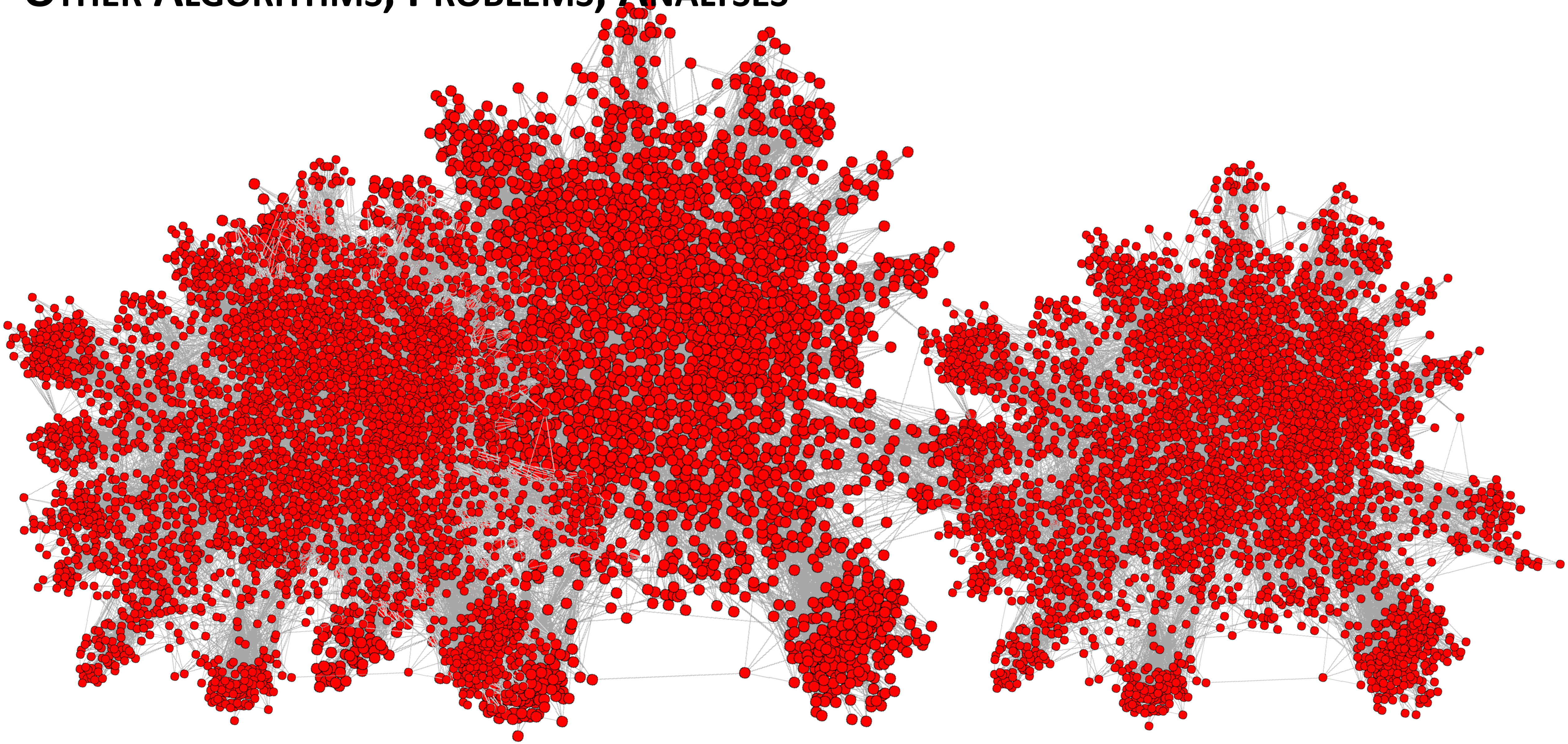
BRAM signal propagation limits the frequency

Addition complexity grows linearly with *L*

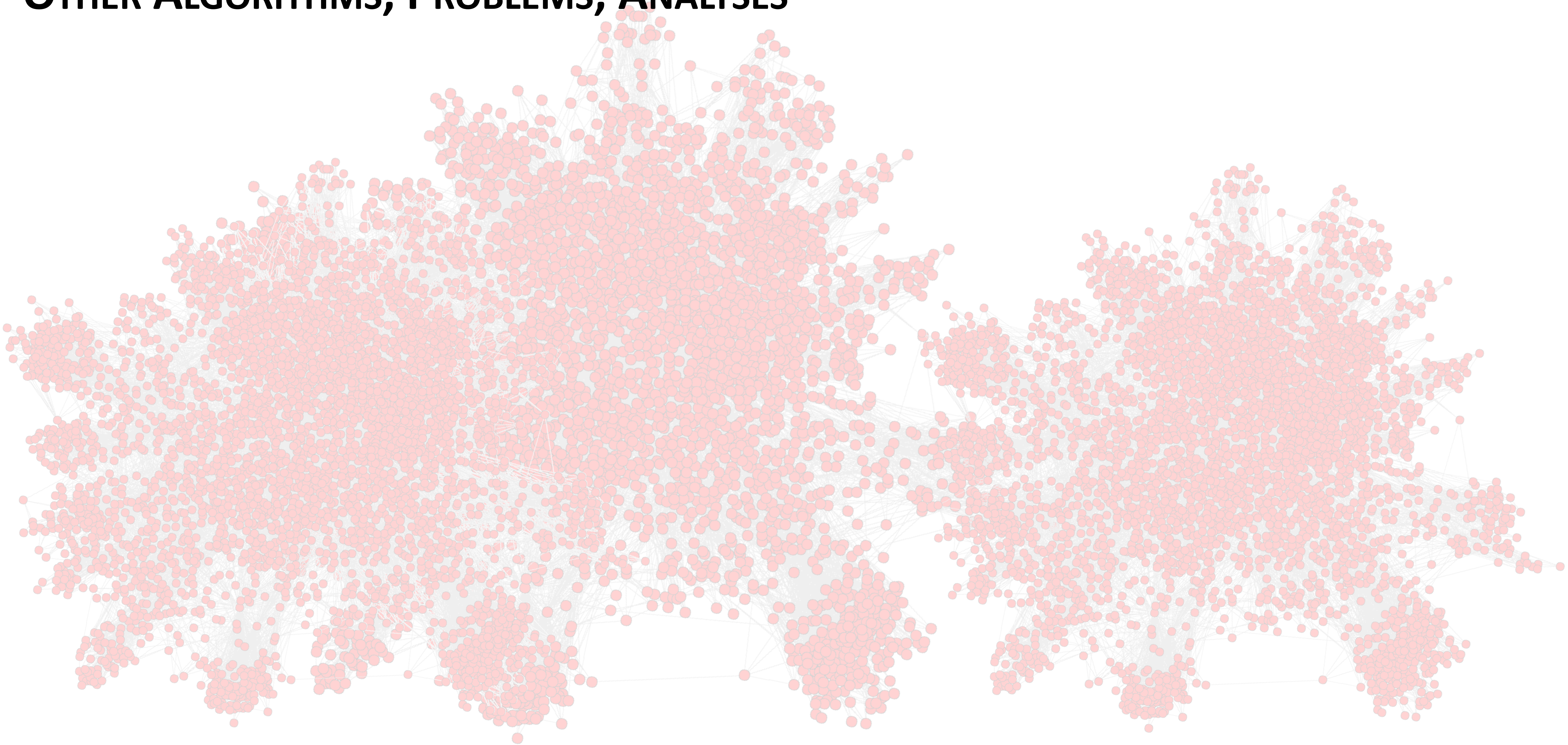


x - the highest possible value of B for a given L

Other Algorithms, Problems, Analyses

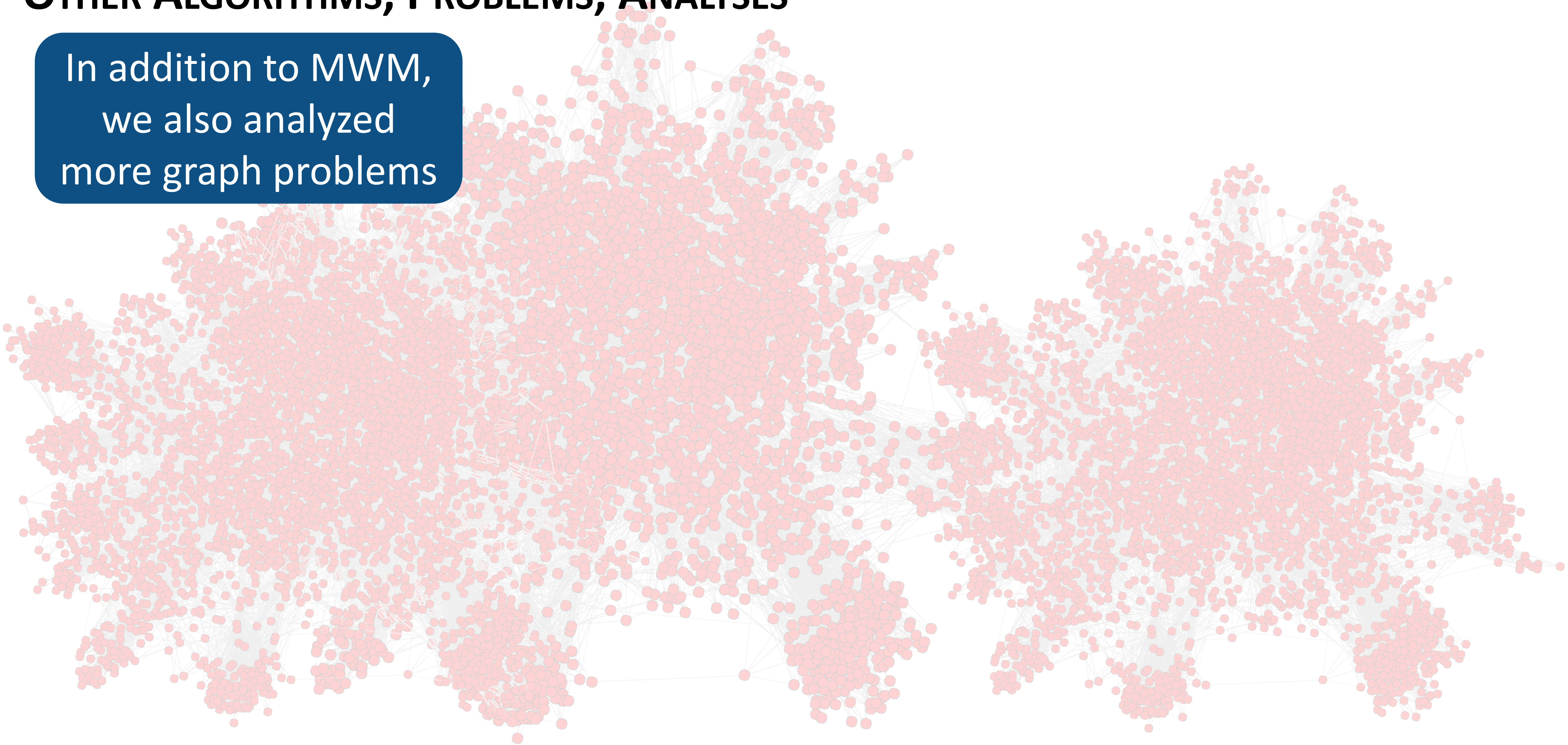


Other Algorithms, Problems, Analyses



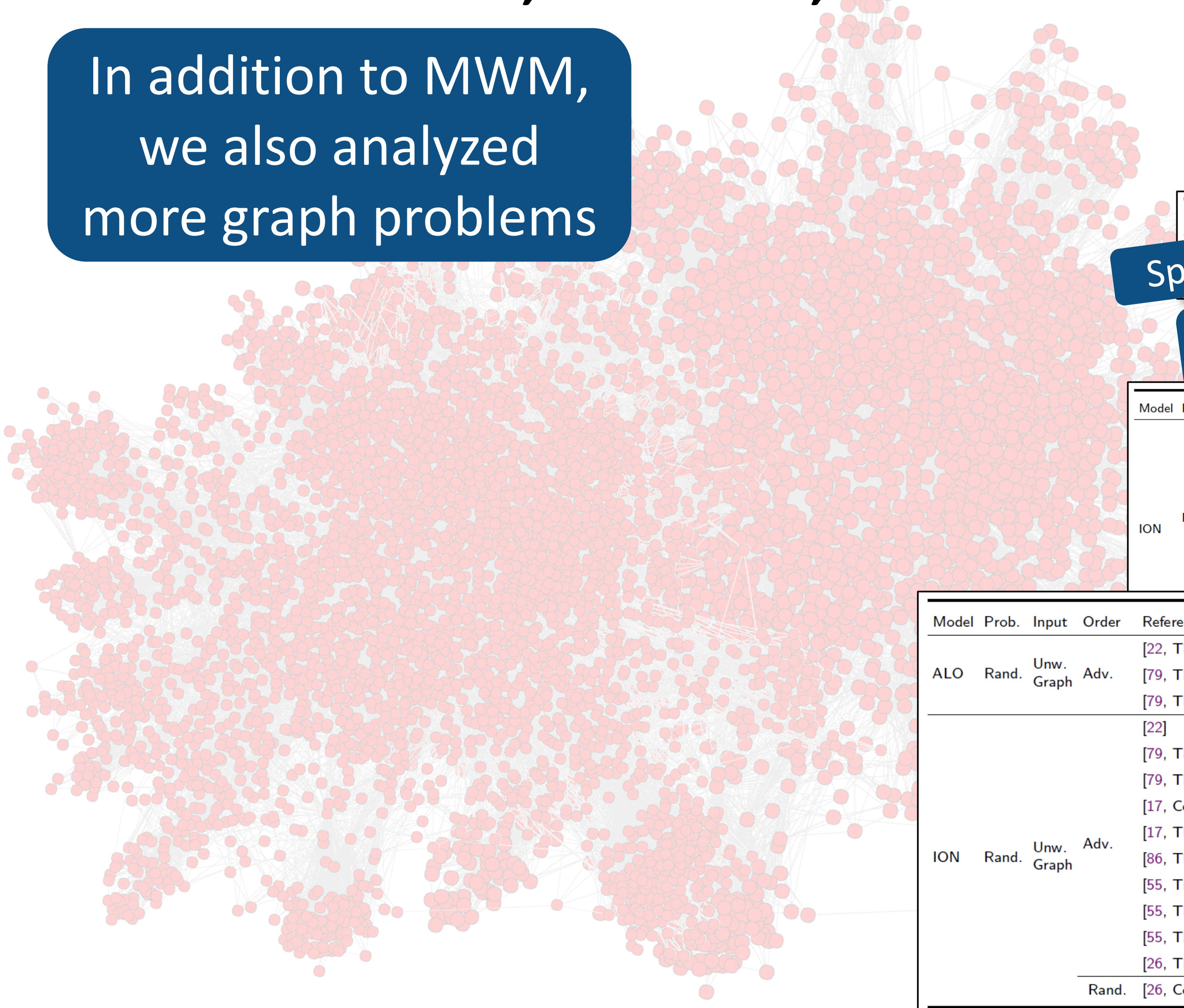
Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems



Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems



Bipartiteness

Motif counting

Triangle counting

Connected components

Colorings

Minimum spanning trees

Random walks

Spanners

Connectivity

Sparsification

Densest subgraphs

K-edge connectivity

Triangle counting

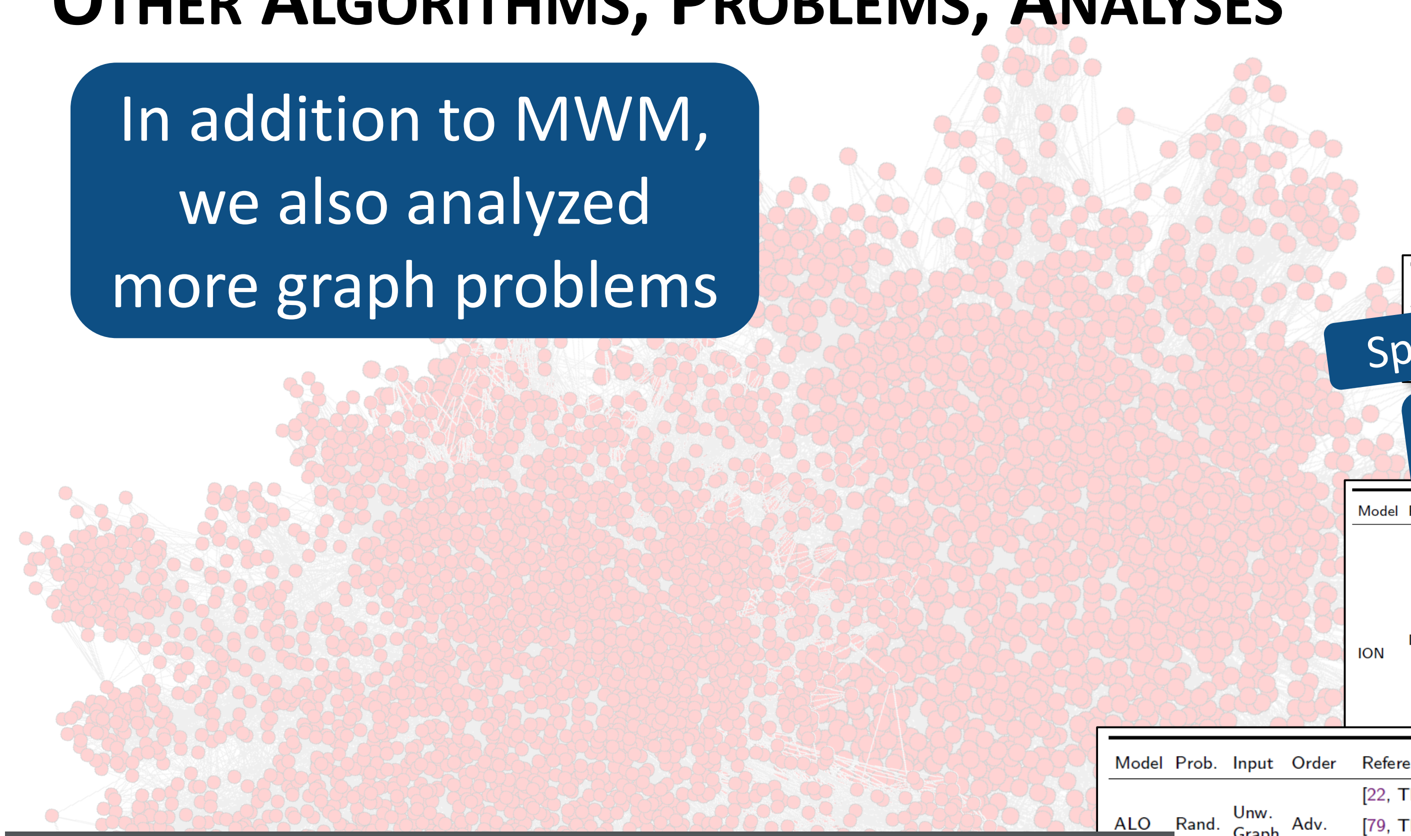
K-vertex connectivity

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
				[37, Theorem 2]	6	$O(n \log(n))$	$O(1)^*$	1	Greedy
				[75, Theorem 3]	$2+\epsilon$ (5.82)	$O(n \text{ polylog}(n))$	$O(1)^*$	$O(1)$ (1)	Greedy
				[102]	5.58	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[34]	$4.911+\epsilon$	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[28]	$4+\epsilon$	$O(n \text{ polylog}(n))$	$\Omega(\log(n))^*$	1	-

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
ALO	Rand.	Unw. Graph	Adv.	[22, Theorem 4]	(ϵ, δ)	$\tilde{O}(n(T_2+T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 3]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2} m/\sqrt{T})$	-	1	Sampling
				[79, Theorem 5]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2} m^{3/2}/T)$	-	2	Sampling
ION	Rand.	Unw. Graph	Adv.	[22]	(ϵ, δ)	$\tilde{O}((T_1+2T_2+3T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 10]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2} m/\sqrt{T})$	-	2	Sampling
				[79, Theorem 14]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2} m^{3/2}/T)$	-	3	Sampling
				[17, Corollary 3.9]	$(\epsilon, 1/3)$	$\tilde{O}(m^{3/2}/T)$	-	4	Reserv. Sampling
				[17, Theorem 3.1]	$(\epsilon, 1/3)$	$\tilde{O}(m^2/T)$	-	2	Reserv. Sampling
				[86, Theorem 3.3]	(ϵ, δ)	$O(6/\epsilon^2 m\triangle/T \log(2/\delta))$	-	1	-
				[55, Theorem 1]	(ϵ, δ)	$\tilde{O}(\log(1/\delta) m\triangle^2/T)$		1	Sampling
				[55, Theorem 2]	(ϵ, δ)	$\tilde{O}(\log(1/\delta)((m^3+mC_4+C_6)/T^2+1))$		1	Sampling
				[55, Theorem 3]	(ϵ, δ)	$O(n+1/\epsilon^2 \log(1/\delta)(T_2/T+1)\log(n))$		3	Sampling
				[26, Theorem 5]	$(\epsilon, 1/2)$	$O(m/(\epsilon^{2.5}\sqrt{T}) \text{ polylog}(n))^*$	-	2	Sampling
			Rand.	[26, Corollary 6]	$(1/3+\epsilon, -)$	$O(m/(\epsilon^{4.5}\sqrt{T}))$	-	1	Sampling

Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems



Bipartiteness
Motif counting
Triangle counting
Connected components
Colorings
Minimum spanning trees
Random walks
Spanners
Sparsification
Connectivity
Triangle counting
K-vertex connectivity
Densest subgraphs
K-edge connectivity

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
				[37, Theorem 2]	6	$O(n \log(n))$	$O(1)^*$	1	Greedy
				[75, Theorem 3]	$2+\epsilon$ (5.82)	$O(n \text{ polylog}(n))$	$O(1)^*$	$O(1)$ (1)	Greedy
				[102]	5.58	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[34]	$4.911+\epsilon$	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[28]	$4+\epsilon$	$O(n \text{ polylog}(n))$	$\Omega(\log(n))^*$	1	-

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
ALO	Rand.	Unw. Graph	Adv.	[22, Theorem 4]	(ϵ, δ)	$\tilde{O}(n(T_2+T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 3]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2} m/\sqrt{T})$	-	1	Sampling
				[79, Theorem 5]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2} m^{3/2}/T)$	-	2	Sampling
	Rand.	Unw. Graph	Adv.	[22]	(ϵ, δ)	$\tilde{O}((T_1+2T_2+3T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 10]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2} m/\sqrt{T})$	-	2	Sampling
				[79, Theorem 14]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2} m^{3/2}/T)$	-	3	Sampling
				[17, Corollary 3.9]	$(\epsilon, 1/3)$	$\tilde{O}(m^{3/2}/T)$	-	4	Reserv. Sampling
				[17, Theorem 3.1]	$(\epsilon, 1/3)$	$\tilde{O}(m^2/T)$	-	2	Reserv. Sampling
				[86, Theorem 3.3]	(ϵ, δ)	$O(6/\epsilon^2 m\triangle/T \log(2/\delta))$	-	1	-
				[55, Theorem 1]	(ϵ, δ)	$\tilde{O}(\log(1/\delta) m\triangle^2/T)$		1	Sampling
				[55, Theorem 2]	(ϵ, δ)	$\tilde{O}(\log(1/\delta)((m^3+mC_4+C_6)/T^2+1))$		1	Sampling
				[55, Theorem 3]	(ϵ, δ)	$O(n+1/\epsilon^2 \log(1/\delta)(T_2/T+1)\log(n))$		3	Sampling
				[26, Theorem 5]	$(\epsilon, 1/2)$	$O(m/(\epsilon^{2.5}\sqrt{T}) \text{ polylog}(n))^*$	-	2	Sampling
			Rand.	[26, Corollary 6]	$(1/3+\epsilon, -)$	$O(m/(\epsilon^{4.5}\sqrt{T}))$	-	1	Sampling

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems

Bipartiteness

Motif counting

Triangle counting

Connected components

Colorings

Minimum spanning trees

Random walks

Spanners

Connectivity

Sparsification

Densest subgraphs

K-vertex connectivity

K-edge connectivity

Triangle counting

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
				[37, Theorem 2]	6	$O(n\log(n))$	$O(1)^*$	1	Greedy
				[75, Theorem 3]	$2+\epsilon$ (5.82)	$O(n \text{ polylog}(n))$	$O(1)^*$	$O(1)$ (1)	Greedy
				[102]	5.58	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[34]	$4.911+\epsilon$	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[28]	$4+\epsilon$	$O(n \text{ polylog}(n))$	$\Omega(\log(n))^*$	1	-

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
ALO	Rand.	Unw. Graph	Adv.	[22, Theorem 4]	(ϵ, δ)	$\tilde{O}(n(T_2+T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 3]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2}m/\sqrt{T})$	-	1	Sampling
				[79, Theorem 5]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	-	2	Sampling
	Rand.	Unw. Graph	Adv.	[22]	(ϵ, δ)	$\tilde{O}((T_1+2T_2+3T_3)/T_3 \log(2/\delta))$	-	1	Sampling
				[79, Theorem 10]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2}m/\sqrt{T})$	-	2	Sampling
				[79, Theorem 14]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	-	3	Sampling
				[17, Corollary 3.9]	$(\epsilon, 1/3)$	$\tilde{O}(m^{3/2}/T)$	-	4	Reserv. Sampling
				[17, Theorem 3.1]	$(\epsilon, 1/3)$	$\tilde{O}(m^2/T)$	-	2	Reserv. Sampling
				[86, Theorem 3.3]	(ϵ, δ)	$O(6/\epsilon^2 m\triangle/T \log(2/\delta))$	-	1	-
				[55, Theorem 1]	(ϵ, δ)	$\tilde{O}(\log(1/\delta)m\triangle^2/T)$		1	Sampling
				[55, Theorem 2]	(ϵ, δ)	$\tilde{O}(\log(1/\delta)((m^3+mC_4+C_6)/T^2+1))$		1	Sampling
				[55, Theorem 3]	(ϵ, δ)	$O(n+1/\epsilon^2\log(1/\delta)(T_2/T+1)\log(n))$		3	Sampling
				[26, Theorem 5]	$(\epsilon, 1/2)$	$O(m/(\epsilon^{2.5}\sqrt{T}))$ polylog$(n))^*$	-	2	Sampling
			Rand.	[26, Corollary 6]	$(1/3+\epsilon, -)$	$O(m/(\epsilon^{4.5}\sqrt{T}))$	-	1	Sampling

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

OTHER ALGORITHMS, PROBLEMS, ANALYSES

In addition to MWM, we also analyzed more graph problems

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

Bipartiteness
Motif counting
Triangle counting
Connected components
Colorings
Minimum spanning trees
Random walks
Spanners
Connectivity
Sparsification
Densest subgraphs
Triangle counting
K-vertex connectivity
K-edge connectivity

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
				[37, Theorem 2]	6	$O(n\log(n))$	$O(1)^*$	1	Greedy
				[75, Theorem 3]	$2+\epsilon$ (5.82)	$O(n \text{ polylog}(n))$	$O(1)^*$	$O(1)$ (1)	Greedy
				[102]	5.58	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[34]	$4.911+\epsilon$	$O(n \text{ polylog}(n))$	$O(1)^*$	1	-
				[28]	$4+\epsilon$	$O(n \text{ polylog}(n))$	$\Omega(\log(n))^*$	1	-

Model	Prob.	Input	Order	Reference	Approx.	Space	Time	Passes	Method
ALO	Rand.	Unw. Graph	Adv.	[22, Theorem 4]	(ϵ,δ)	$\tilde{O}(n(T_2+T_3)/T_3\log(2/\delta))$	-	1	Sampling
				[79, Theorem 3]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2}m/\sqrt{T})$	-	1	Sampling
				[79, Theorem 5]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	-	2	Sampling
	Rand.	Unw. Graph	Adv.	[22]	(ϵ,δ)	$\tilde{O}((T_1+2T_2+3T_3)/T_3\log(2/\delta))$	-	1	Sampling
				[79, Theorem 10]	$(\epsilon, 1/50)$	$\tilde{O}(\epsilon^{-2}m/\sqrt{T})$	-	2	Sampling
				[79, Theorem 14]	$(\epsilon, 1/100)$	$\tilde{O}(\epsilon^{-2}m^{3/2}/T)$	-	3	Sampling
				[17, Corollary 3.9]	$(\epsilon, 1/3)$	$\tilde{O}(m^{3/2}/T)$	-	4	Reserv. Sampling
				[17, Theorem 3.1]	$(\epsilon, 1/3)$	$\tilde{O}(m^2/T)$	-	2	Reserv. Sampling
				[86, Theorem 3.3]	(ϵ,δ)	$O(6/\epsilon^2 m\triangle/T\log(2/\delta))$	-	1	-
				[55, Theorem 1]	(ϵ,δ)	$\tilde{O}(\log(1/\delta)m\triangle^2/T)$		1	Sampling
				[55, Theorem 2]	(ϵ,δ)	$\tilde{O}(\log(1/\delta)((m^3+mC_4+C_6)/T^2+1))$		1	Sampling
				[55, Theorem 3]	(ϵ,δ)	$O(n+1/\epsilon^2\log(1/\delta)(T_2/T+1)\log(n))$		3	Sampling
				[26, Theorem 5]	$(\epsilon, 1/2)$	$O(m/(\epsilon^{2.5}\sqrt{T}))$ polylog$(n))^*$	-	2	Sampling
			Rand.	[26, Corollary 6]	$(1/3+\epsilon, -)$	$O(m/(\epsilon^{4.5}\sqrt{T}))$	-	1	Sampling

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

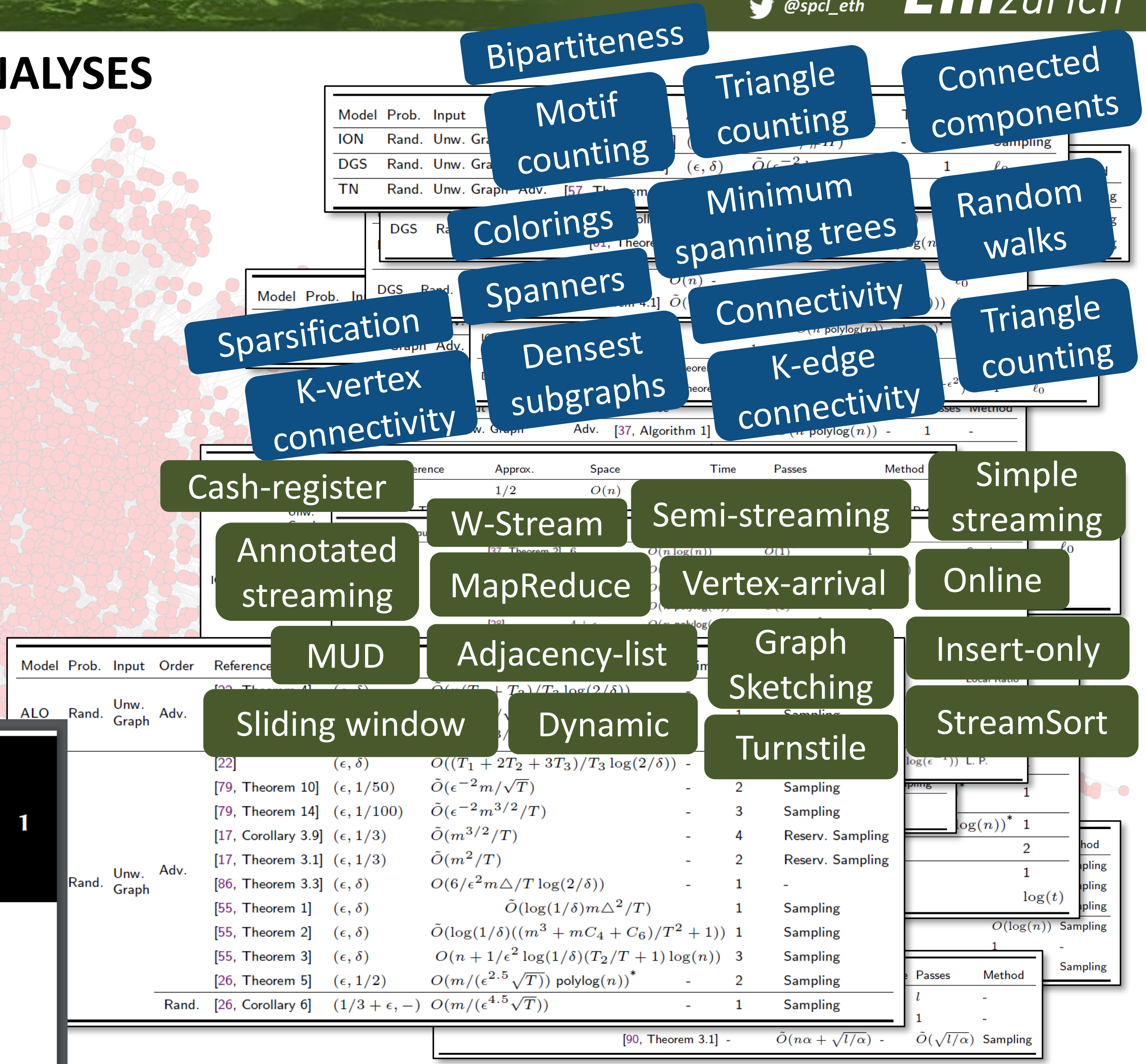
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)



http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf

(submitted to arXiv, will appear tonight)

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

Bipartiteness
Motif counting
Triangle counting
Connected components
Colorings
Minimum spanning trees
Random walks
Spanners
Connectivity
Sparsification
Densest subgraphs
Triangle counting
K-vertex connectivity
K-edge connectivity

Cash-register
Simple streaming
W-Stream
Semi-streaming
Annotated
MapReduce
Vertex-arrival
Online

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Other Algorithms, Problems, Analyses

In addition to MWM, we also analyzed more graph problems

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

Bipartiteness
Motif counting
Triangle counting
Connected components
Colorings
Minimum spanning trees
Random walks
Spanners
Connectivity
Sparsification
Densest subgraphs
Triangle counting
K-vertex connectivity
K-edge connectivity

Cash-register
W-Stream
Semi-streaming
Simple streaming
Annotated
MapReduce
Vertex-arrival
Online

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing
Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Other Algorithms, Problem

Work in progress on the distributed setting ☺

In addition to MWM, we also analyzed more graph problems

Bipartiteness

Motif counting

Triangle counting

Connected components

Colorings

Minimum spanning trees

Random walks

Spanners

Connectivity

Triangle counting

ification

K-vertex connectivity

Densest subgraphs

K-edge connectivity

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

Cash-register

W-Stream

Semi-streaming

Simple streaming

Annotated

MapReduce

Vertex-arrival

Online

http://spcl.inf.ethz.ch/Publications/.pdf/graphs-fpgas-survey.pdf
(submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

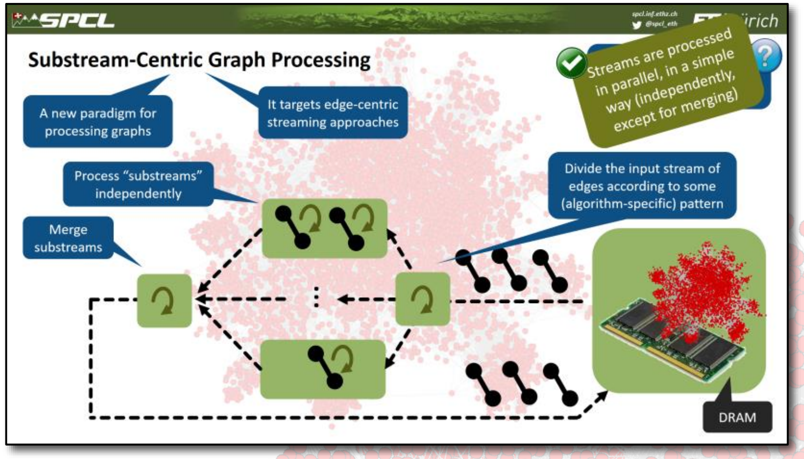
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

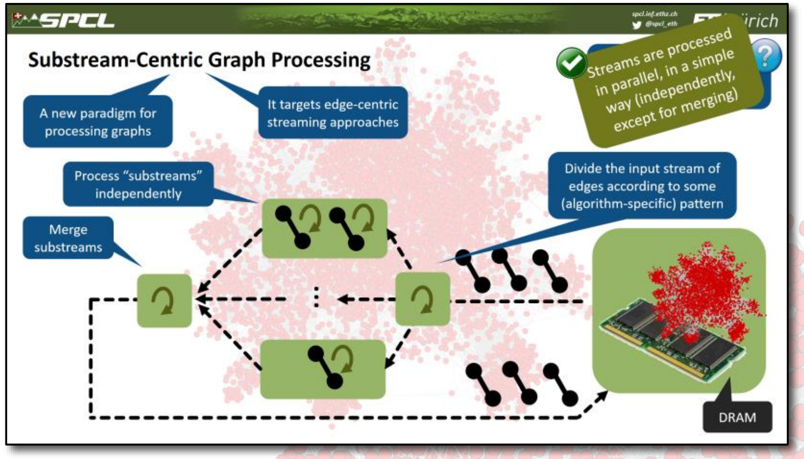
Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

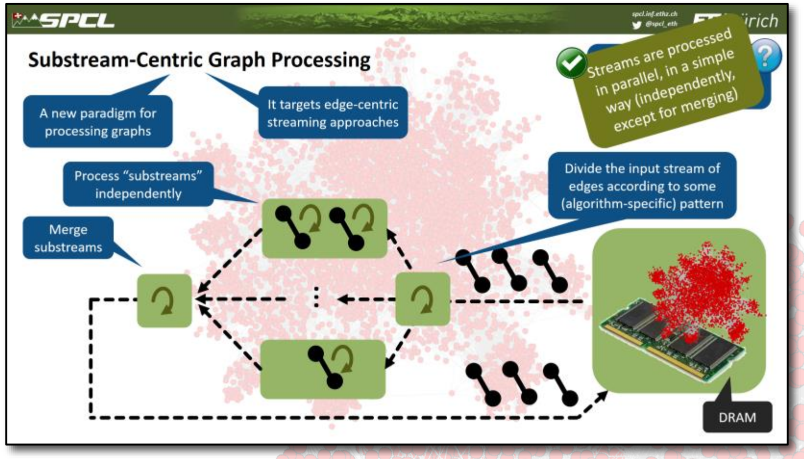


SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



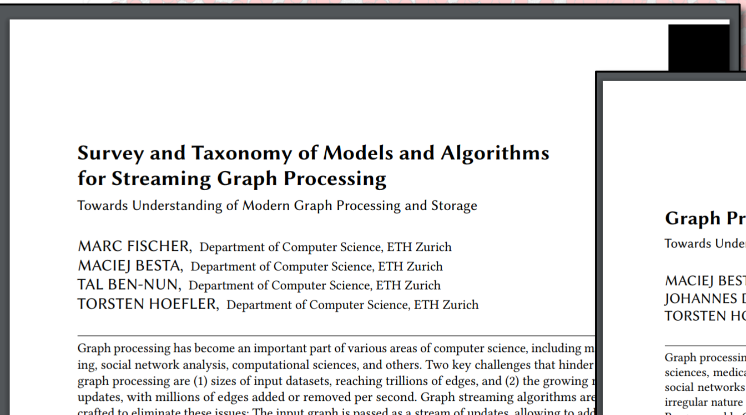
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS



SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

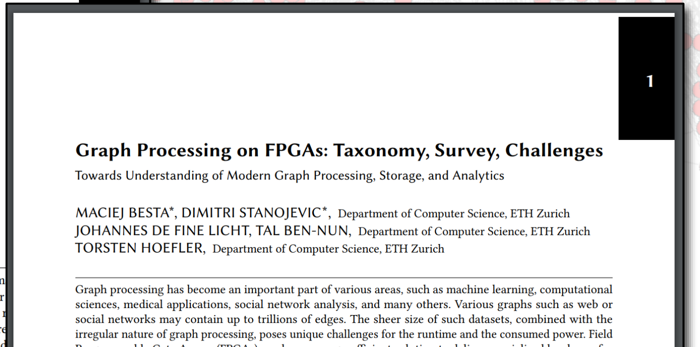


Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including m ing, social network analysis, computational sciences, and others. Two key challenges that hinder graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms are crafted to eliminate these issues: The input graph is passed as a stream of updates, allowing to add



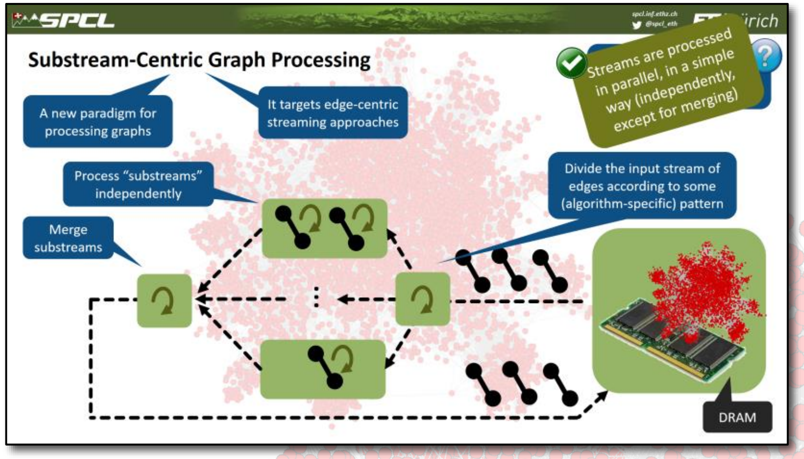
1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field



SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including m... ing, social network analysis, computational sciences, and others. Two key challenges that hinder... graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing... updates, with millions of edges added or removed per second. Graph streaming algorithms are... crafted to eliminate these issues: The input graph is passed as a stream of updates, allowing to add...

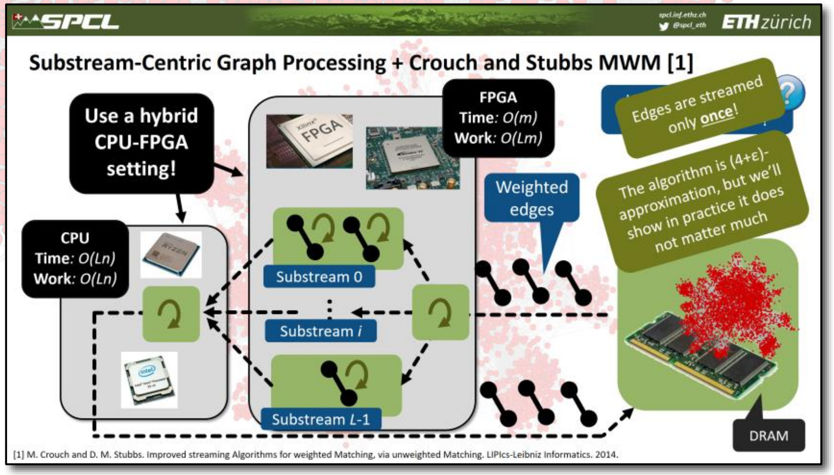
1

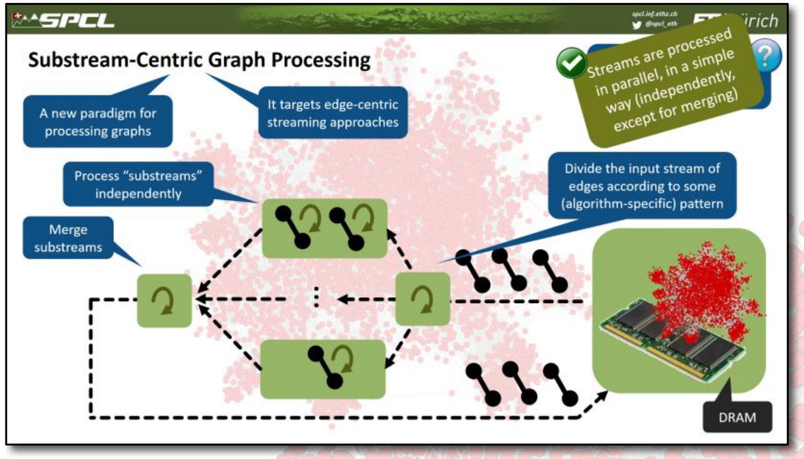
Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

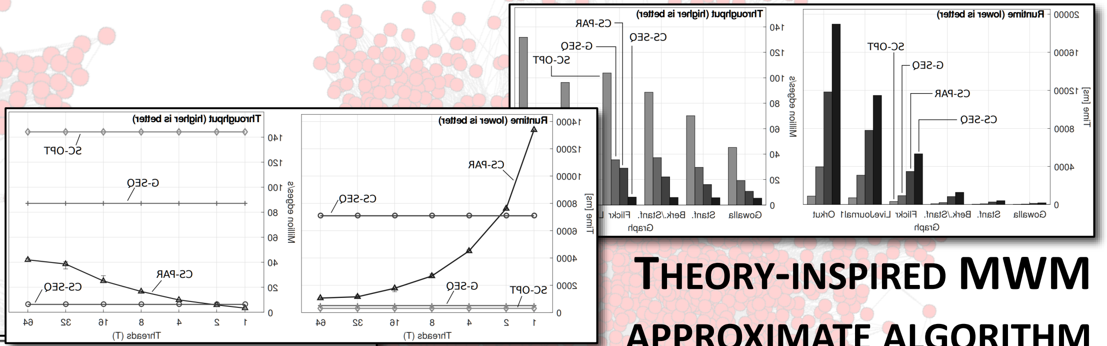
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field ...



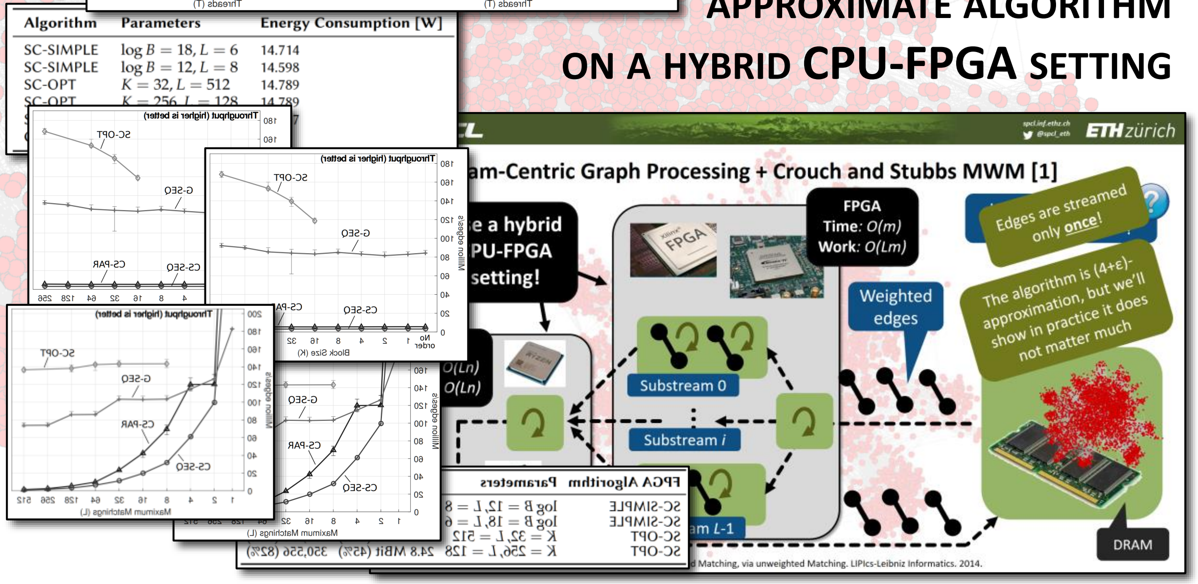


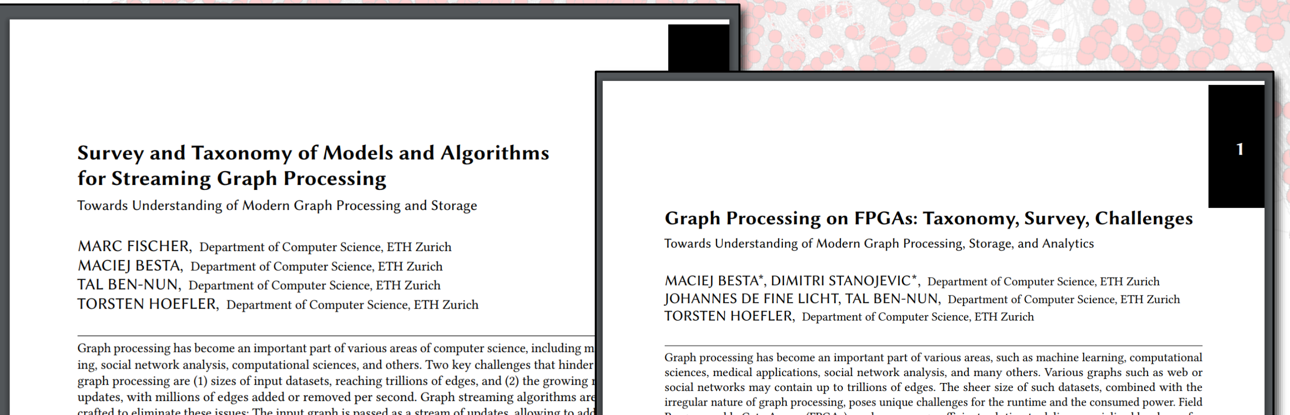
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS





Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including m… ing, social network analysis, computational sciences, and others. Two key challenges that hinder… graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing… updates, with millions of edges added or removed per second. Graph streaming algorithms are…

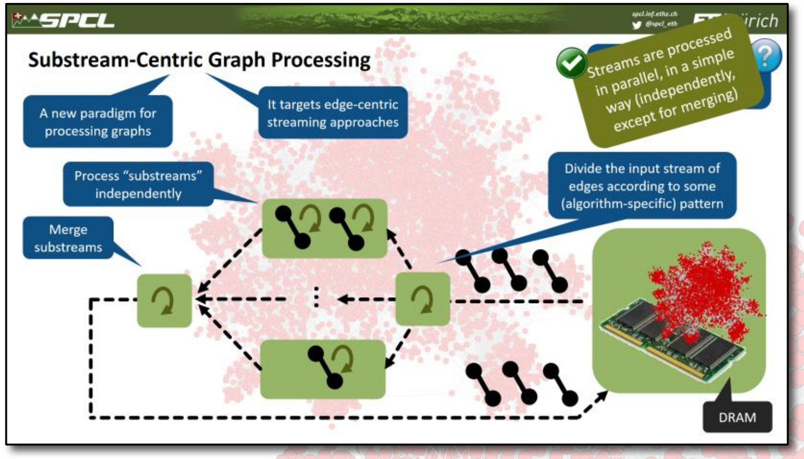
1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

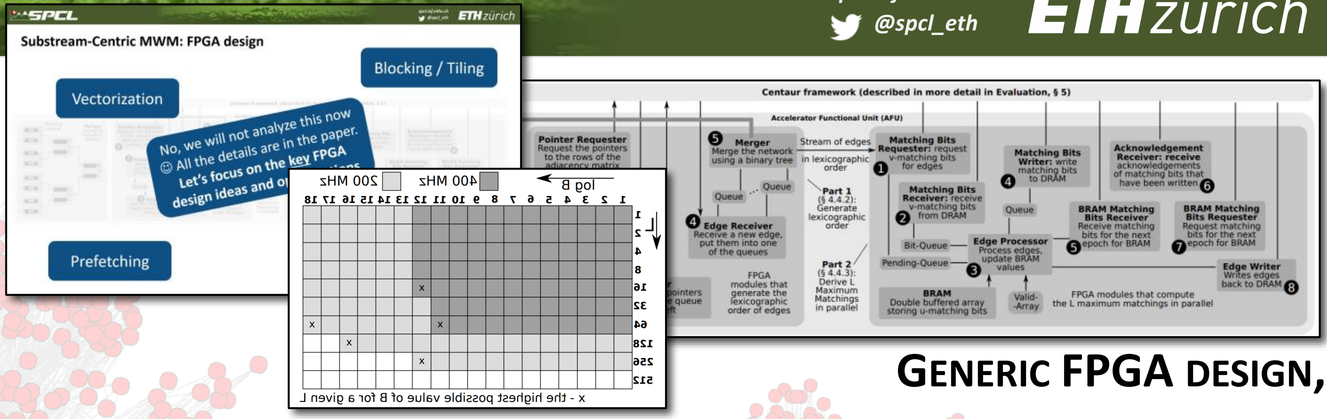
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

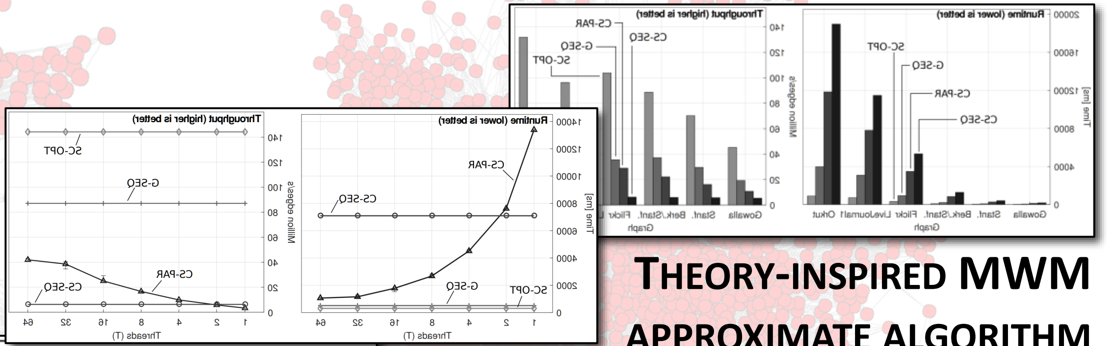
Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field



SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



GENERIC FPGA DESIGN, CODE AVAILABLE



THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including ... ing, social network analysis, computational sciences, and others. Two key challenges that hinder ... graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing ... updates, with millions of edges added or removed per second. Graph streaming algorithms are ...

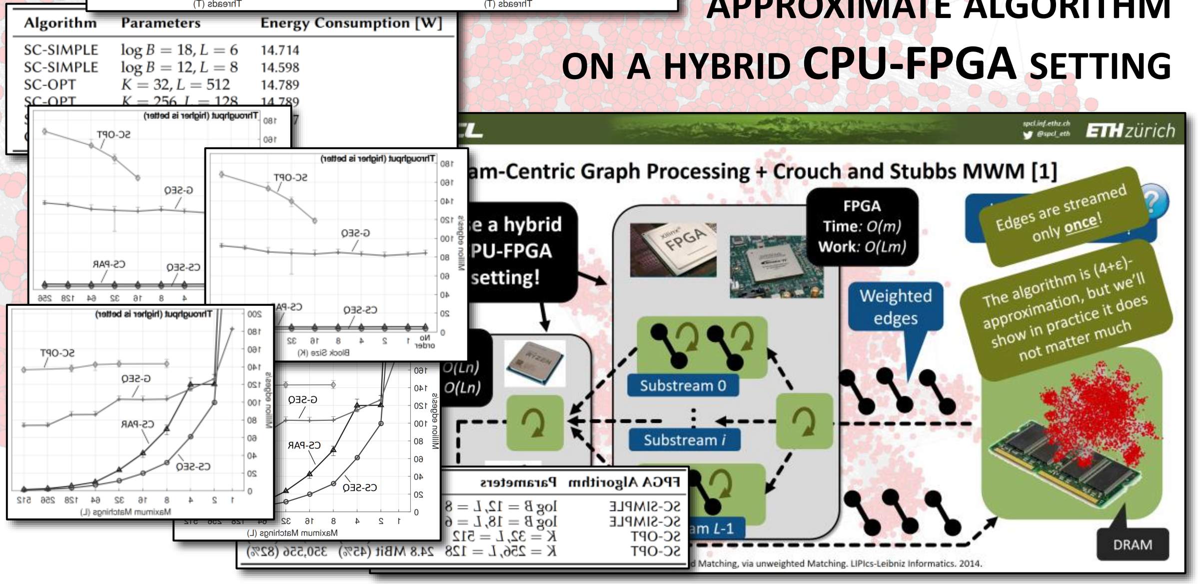
1

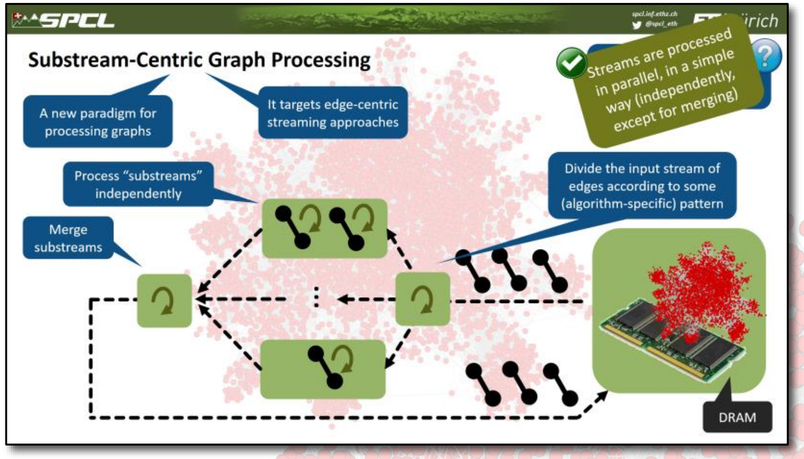
Graph Processing on FPGAs: Taxonomy, Survey, Challenges

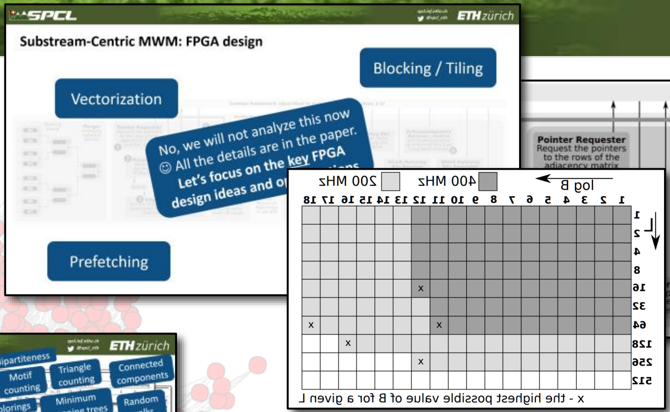
Towards Understanding of Modern Graph Processing, Storage, and Analytics

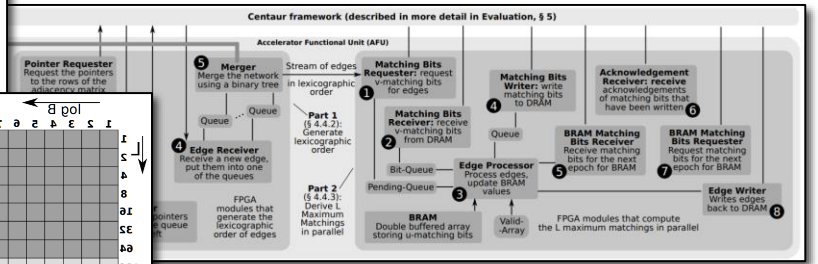
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field ...

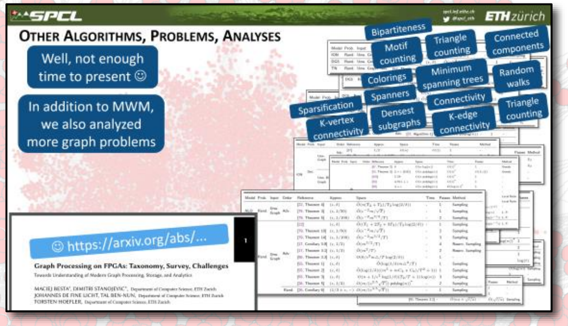






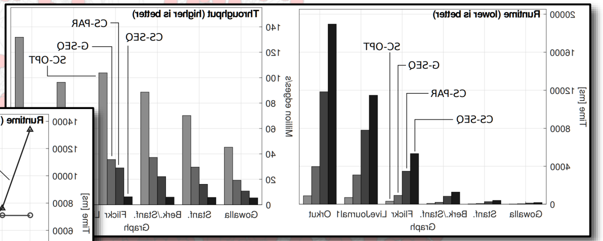


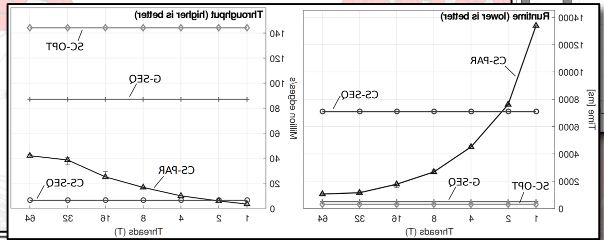
GENERIC FPGA DESIGN, CODE AVAILABLE



SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

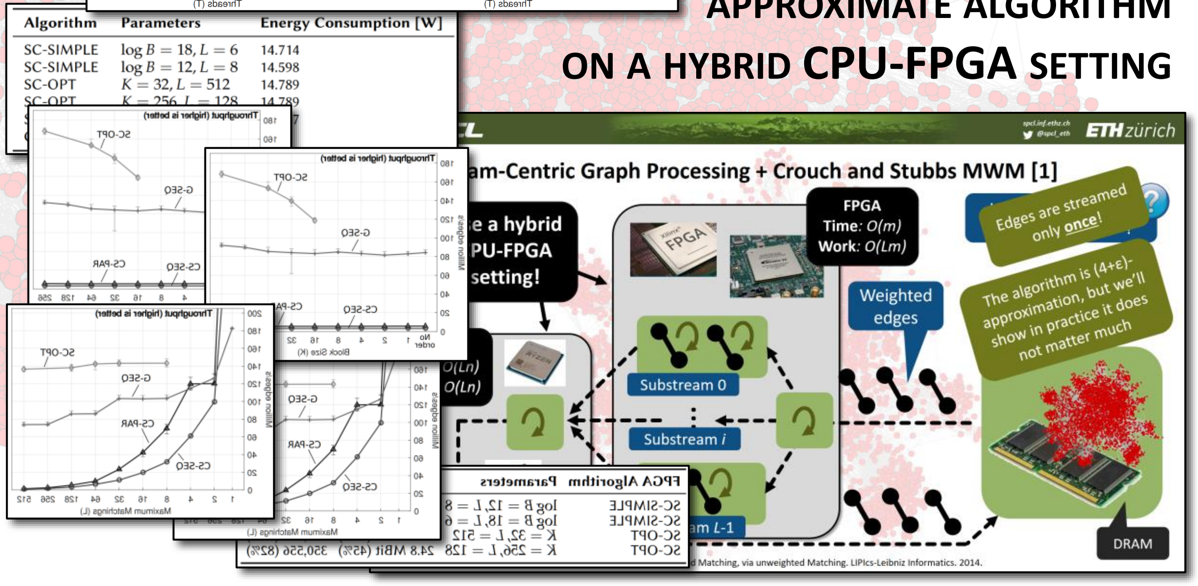
GENERALIZABILITY TO OTHER GRAPH PROBLEMS AND SETTINGS

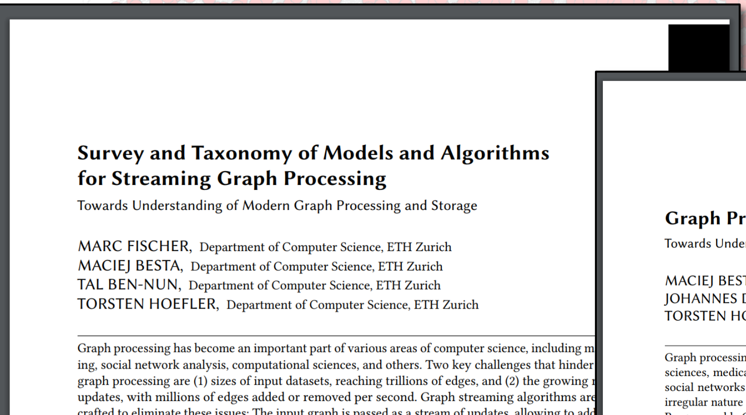




THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

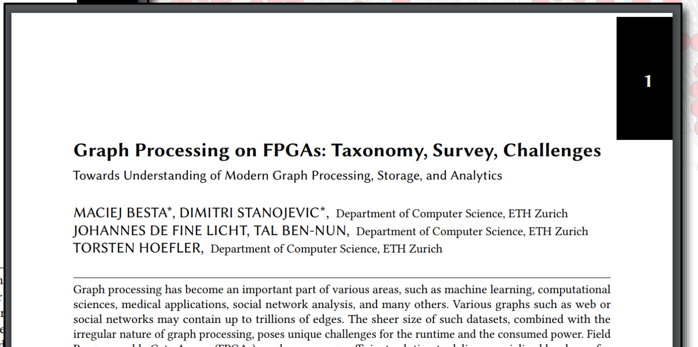
DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS




Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

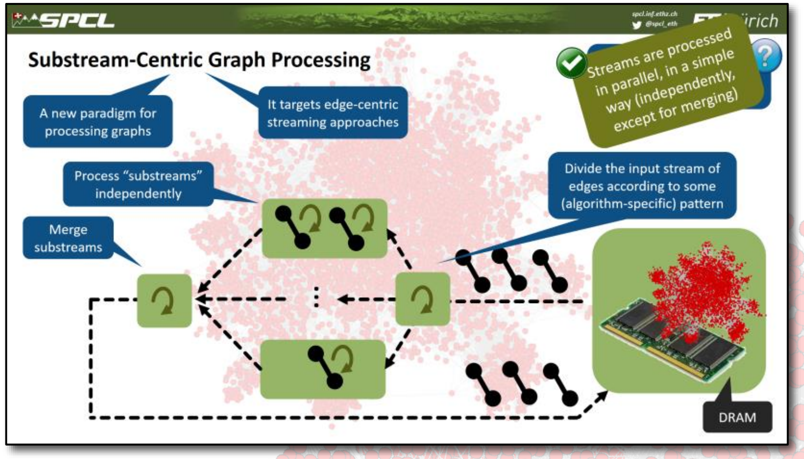

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

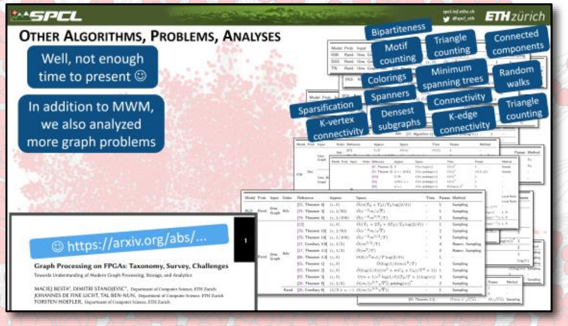
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

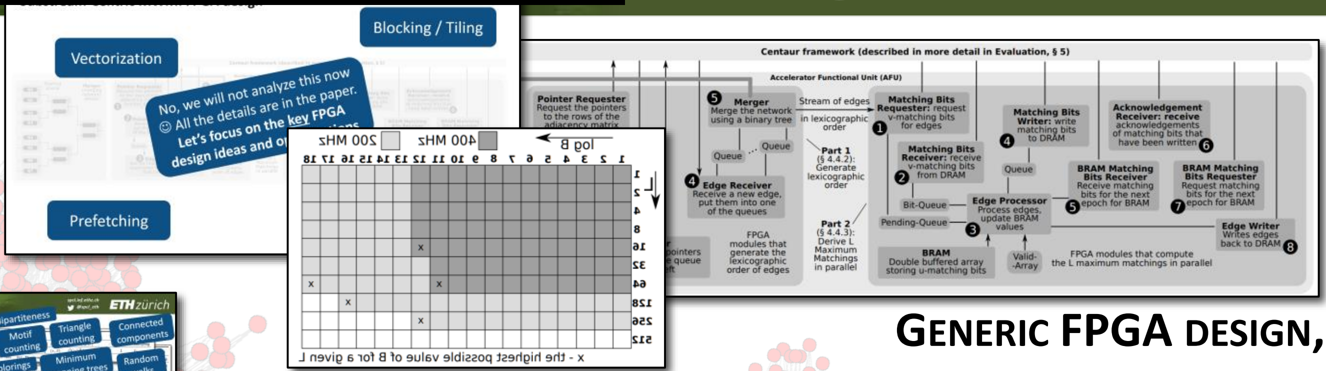
Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field



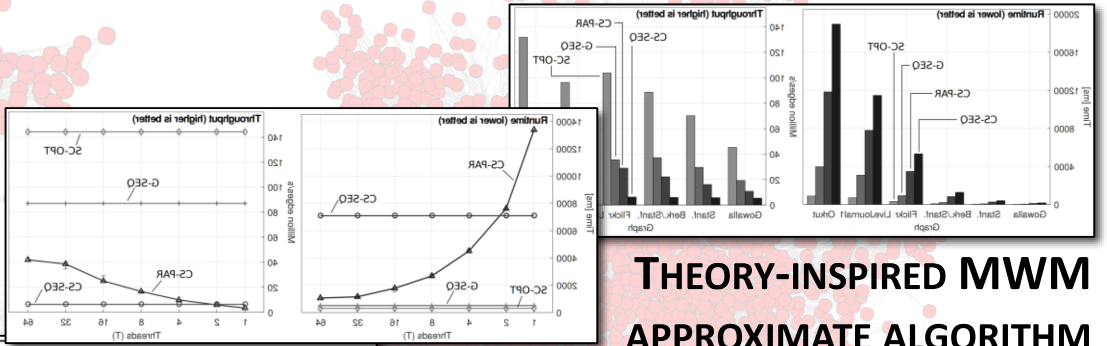
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



GENERALIZABILITY TO OTHER GRAPH PROBLEMS AND SETTINGS

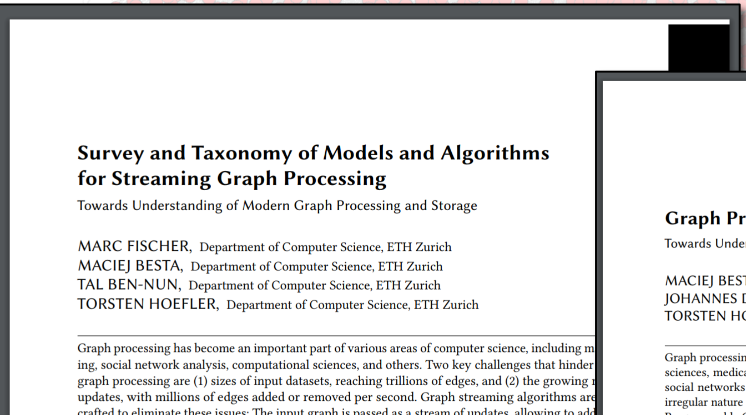


GENERIC FPGA DESIGN, CODE AVAILABLE



THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

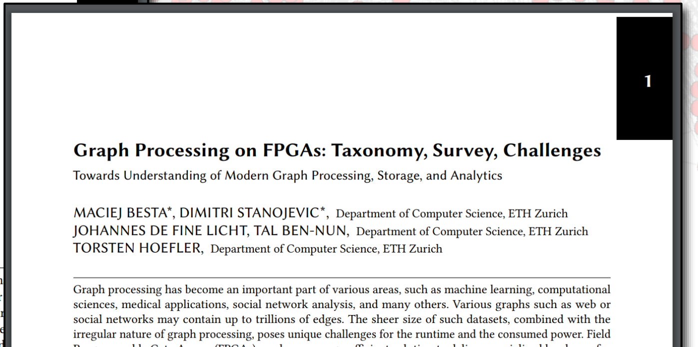


Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including m... ing, social network analysis, computational sciences, and others. Two key challenges that hinder graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms are crafted to eliminate these issues: The input graph is passed as a stream of updates, allowing to add...

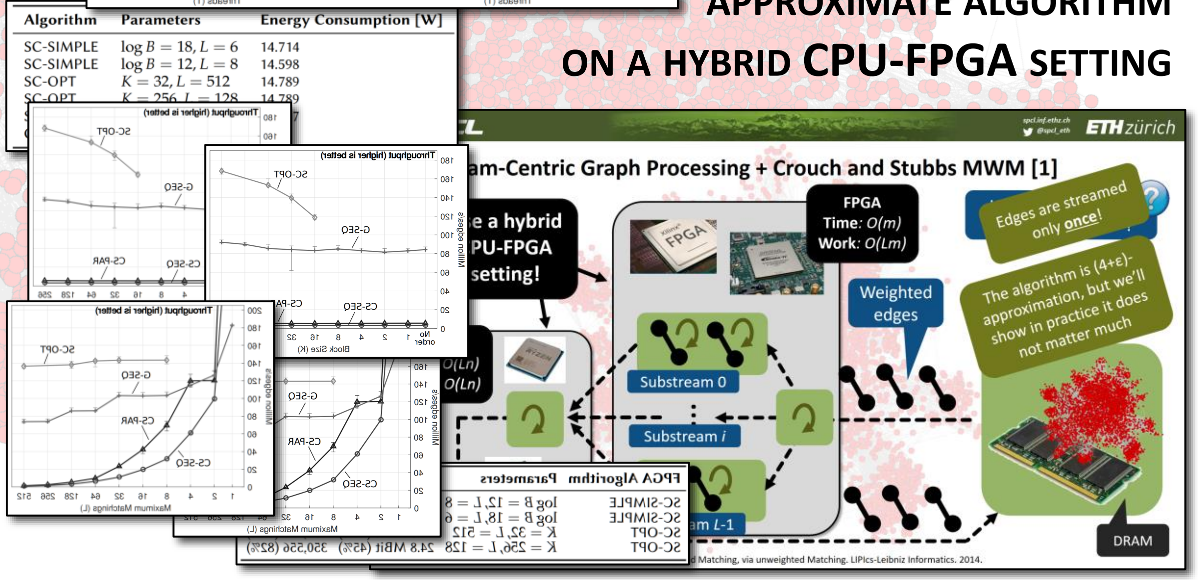


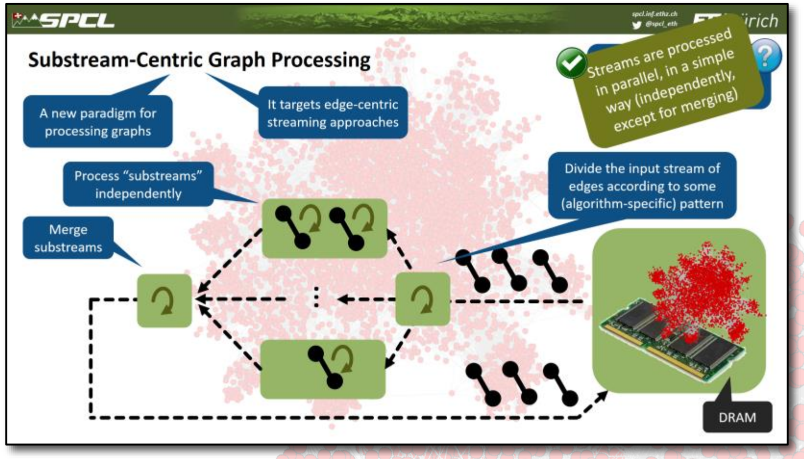
Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

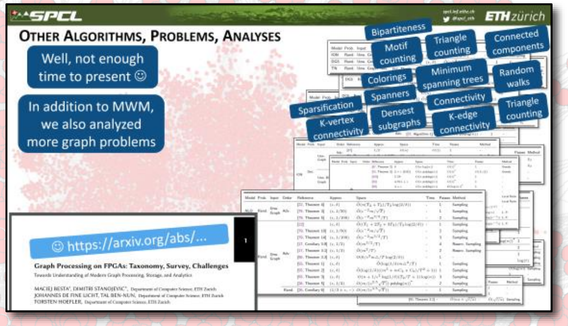
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field ...

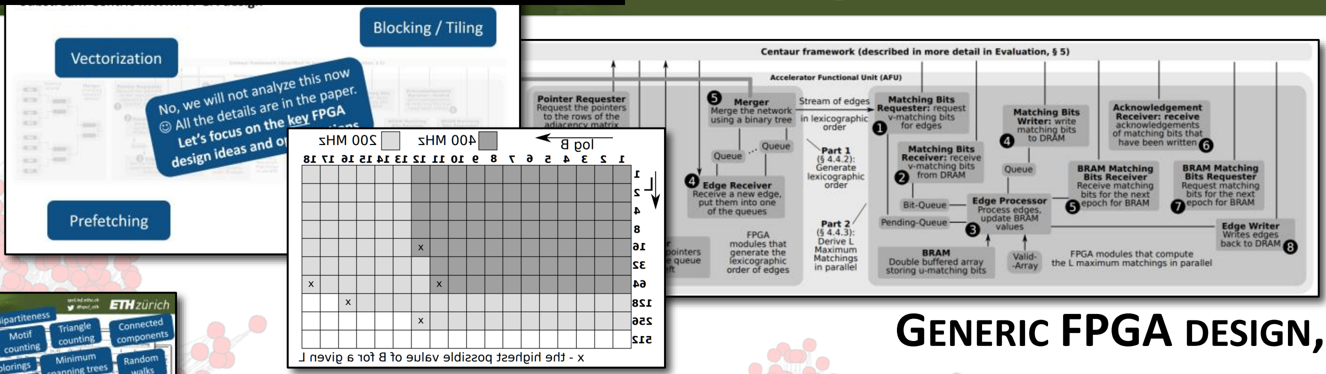




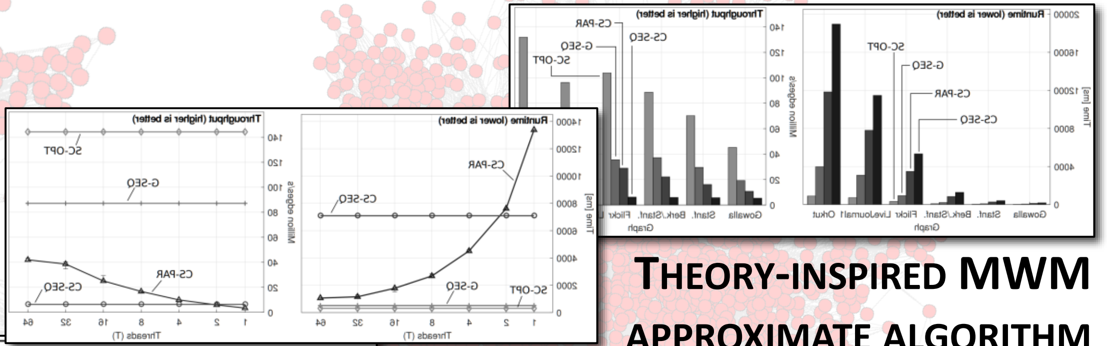
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



GENERALIZABILITY TO OTHER GRAPH PROBLEMS AND SETTINGS



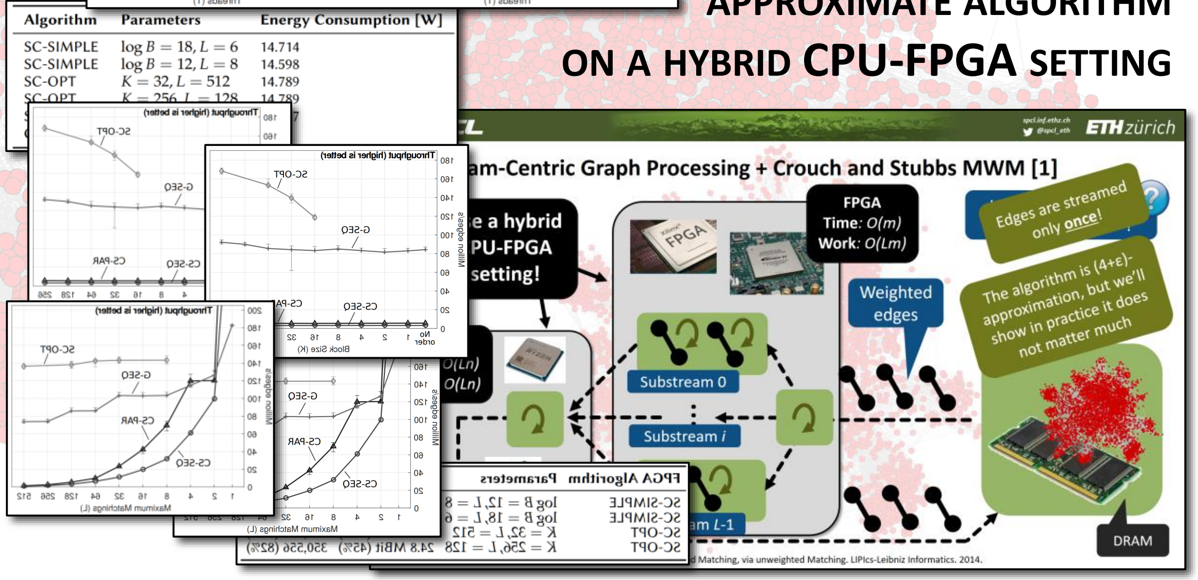
GENERIC FPGA DESIGN, CODE AVAILABLE

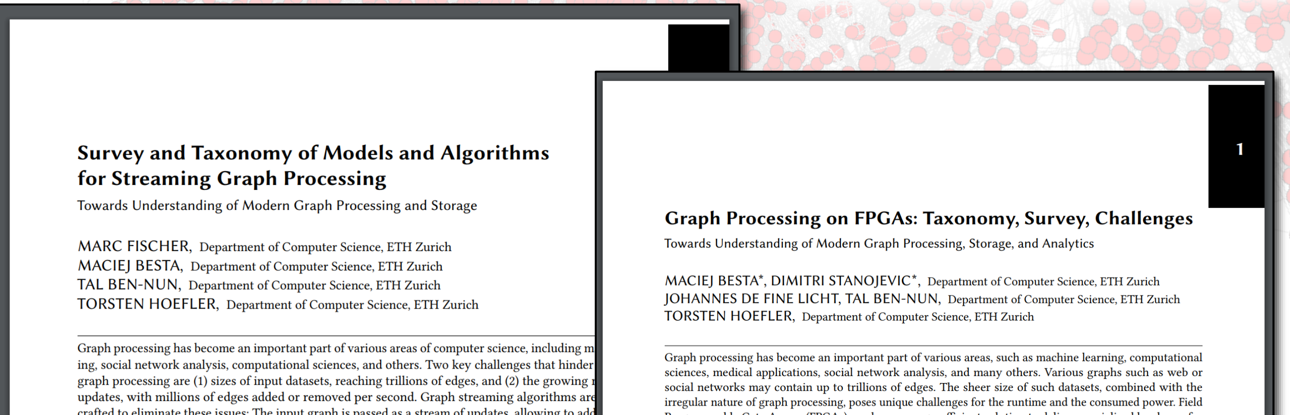


THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

Thank you for your attention





Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including m... ing, social network analysis, computational sciences, and others. Two key challenges that hinder graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms ar...

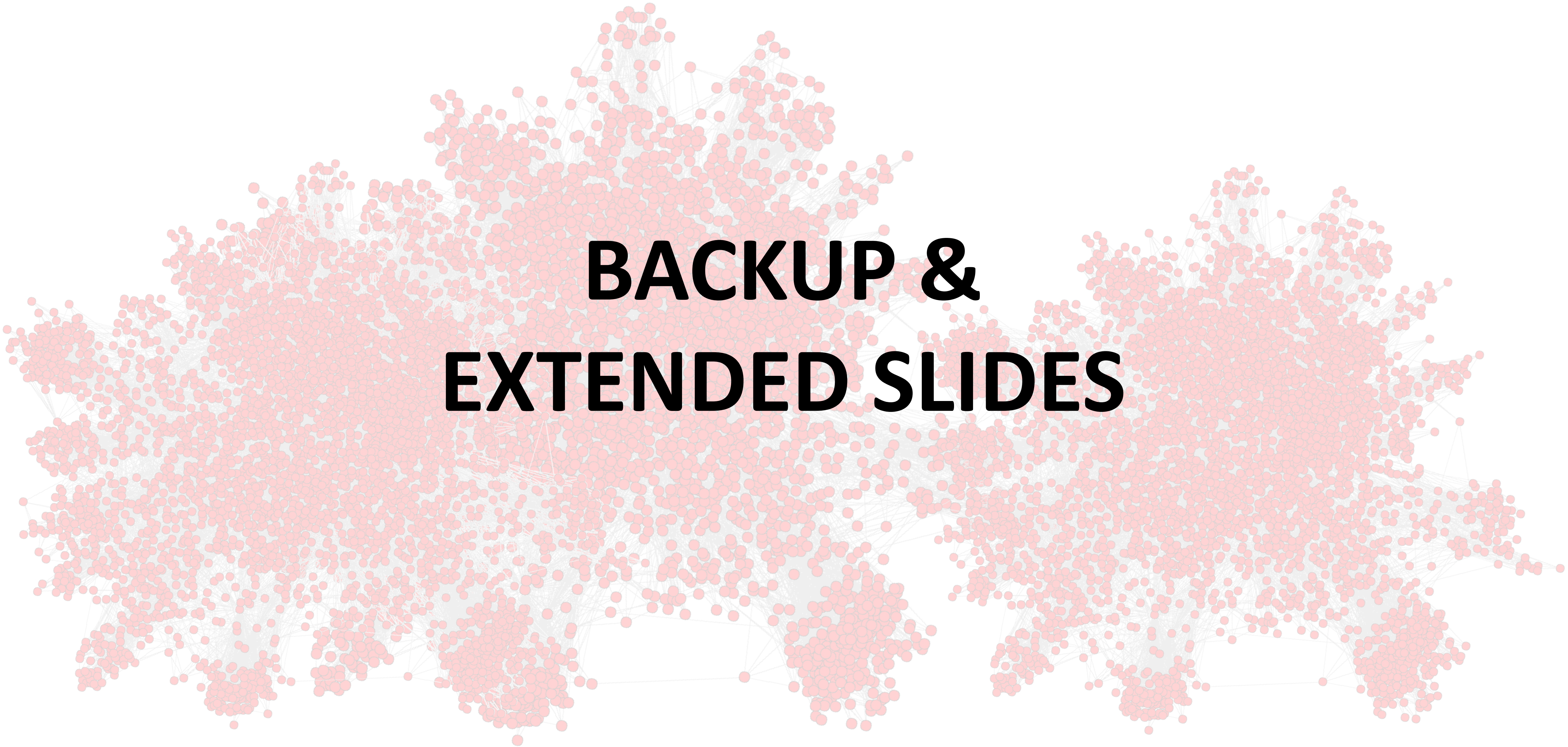
1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

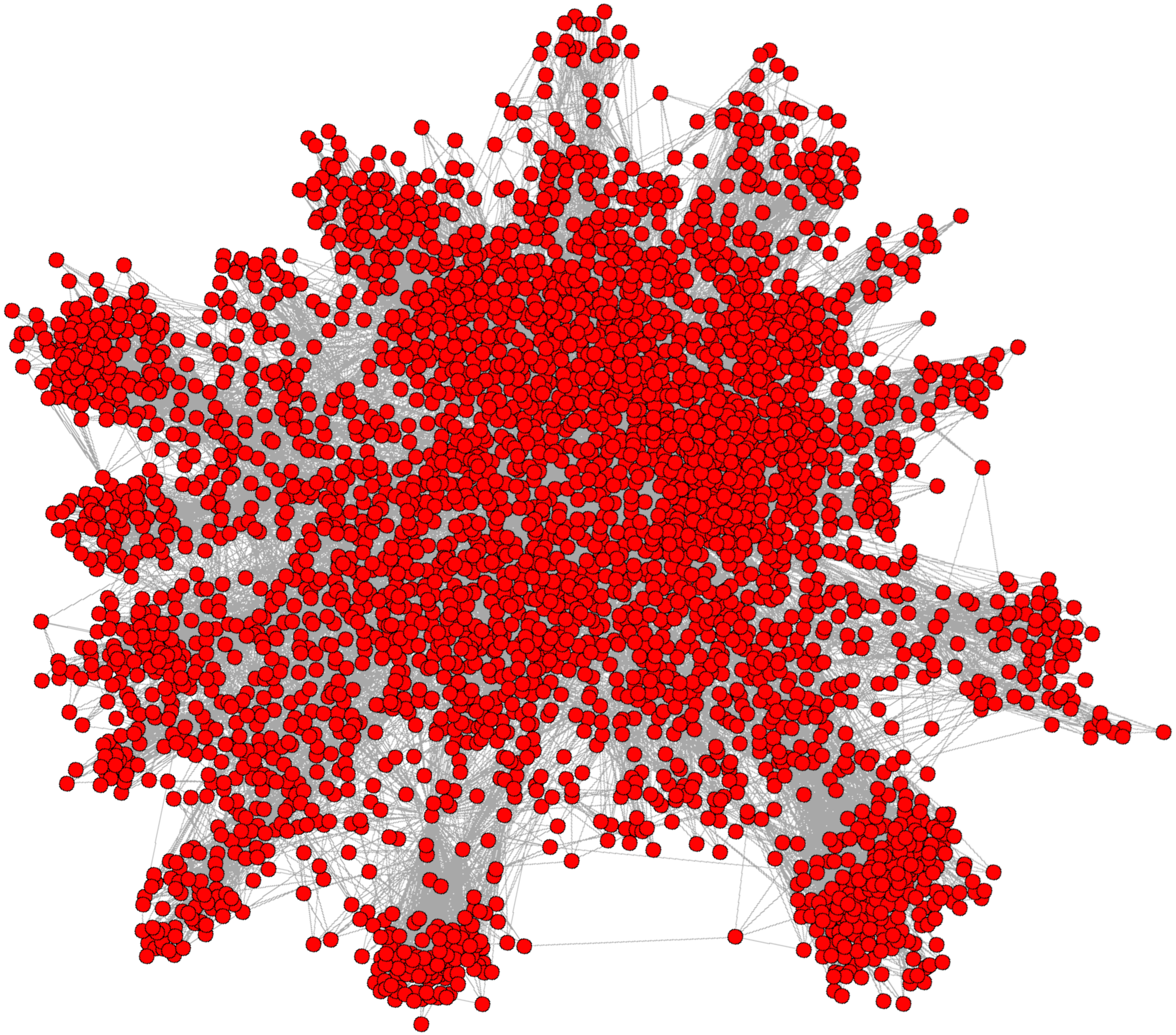
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field



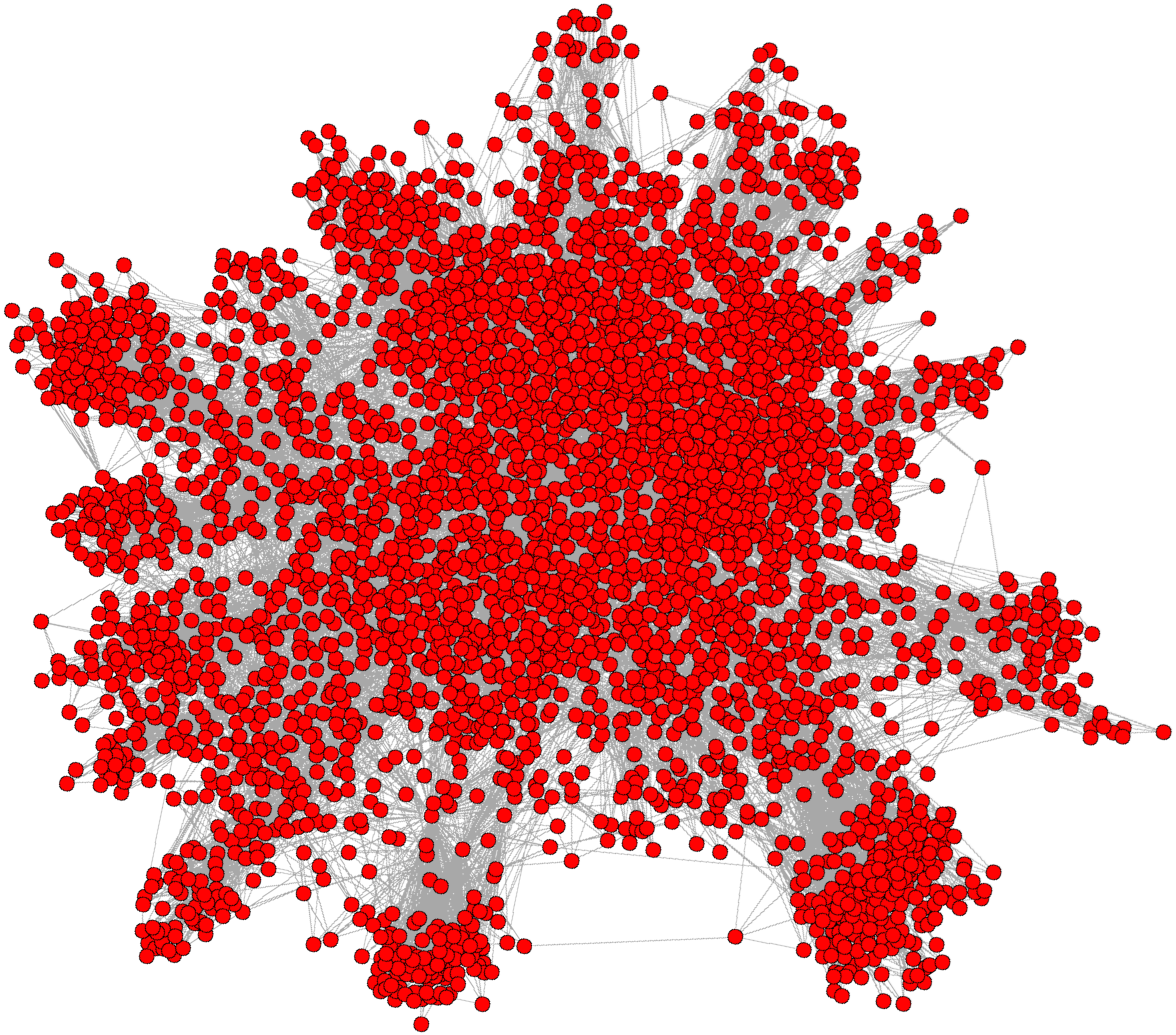
BACKUP & EXTENDED SLIDES

Large graphs…



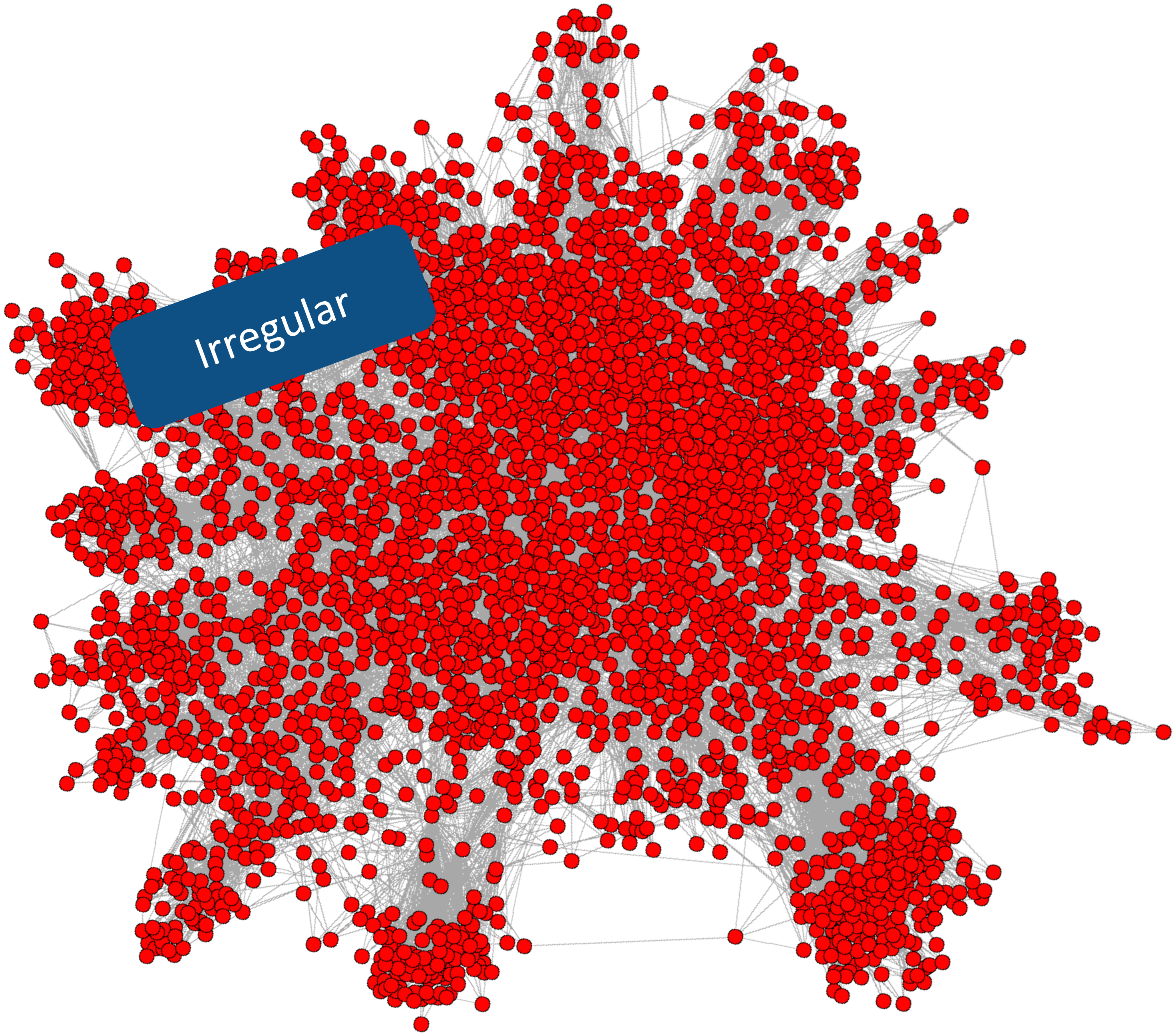
Large graphs…

Problems?



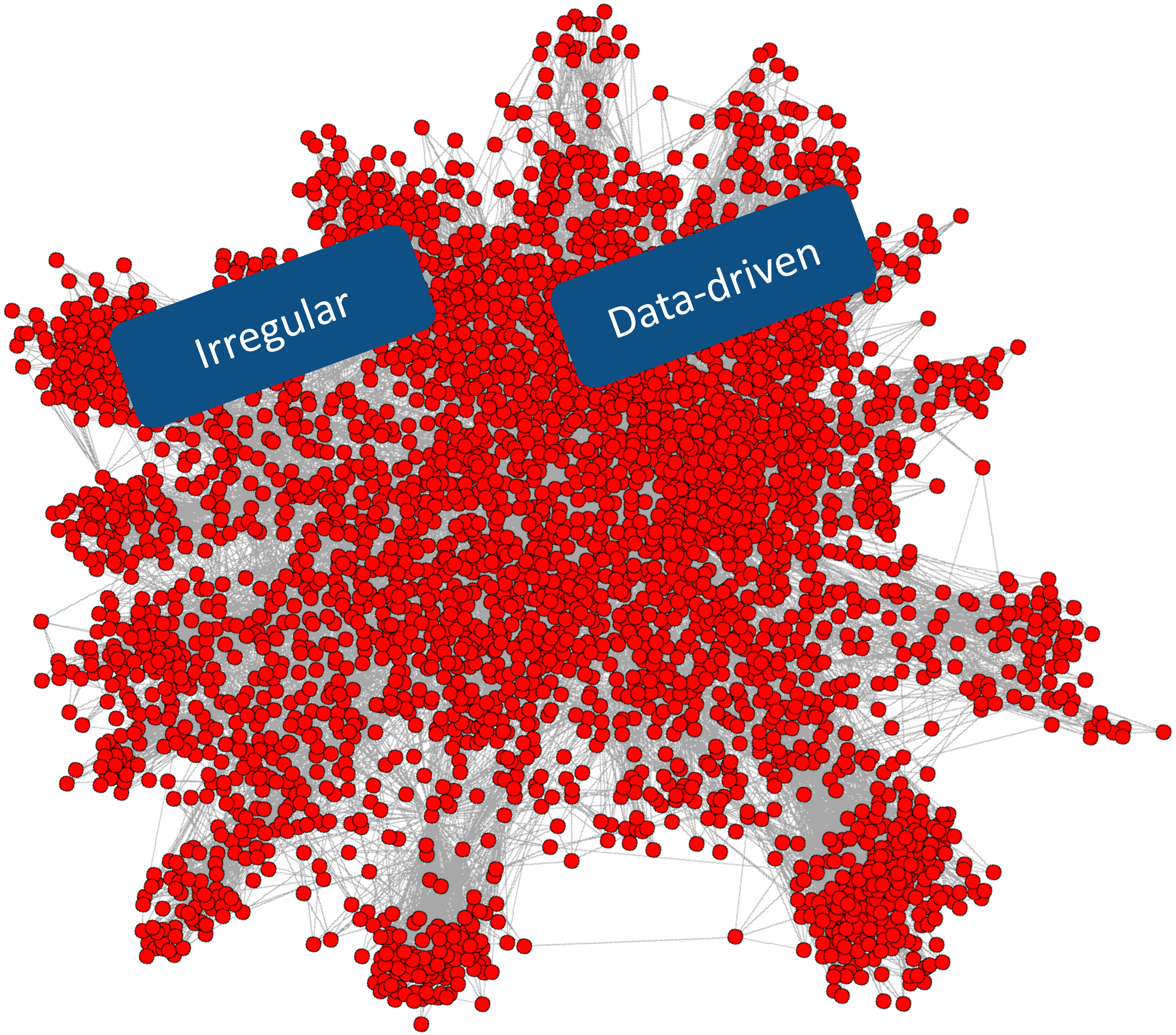
Large graphs…

Problems?



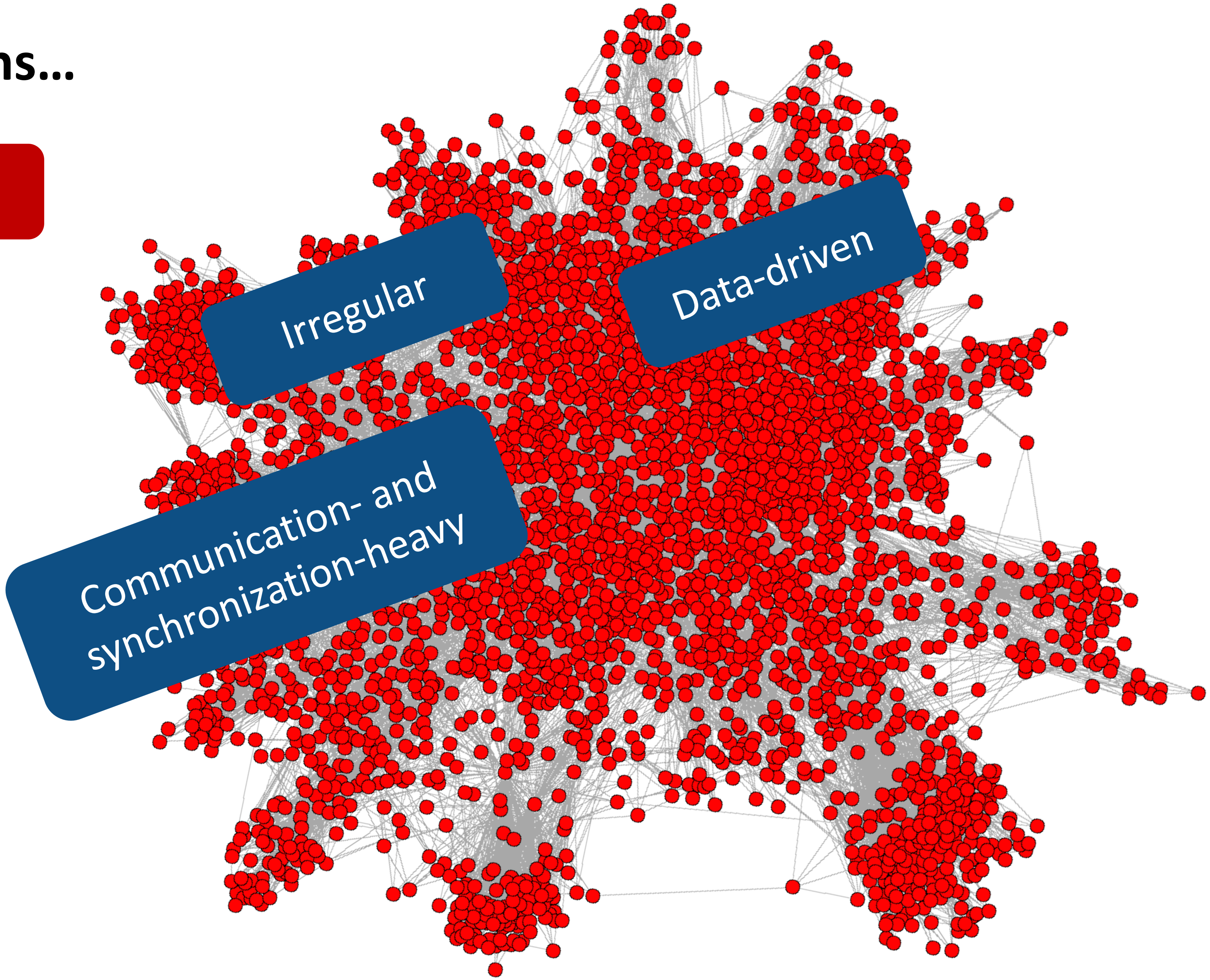
Large graphs…

Problems?



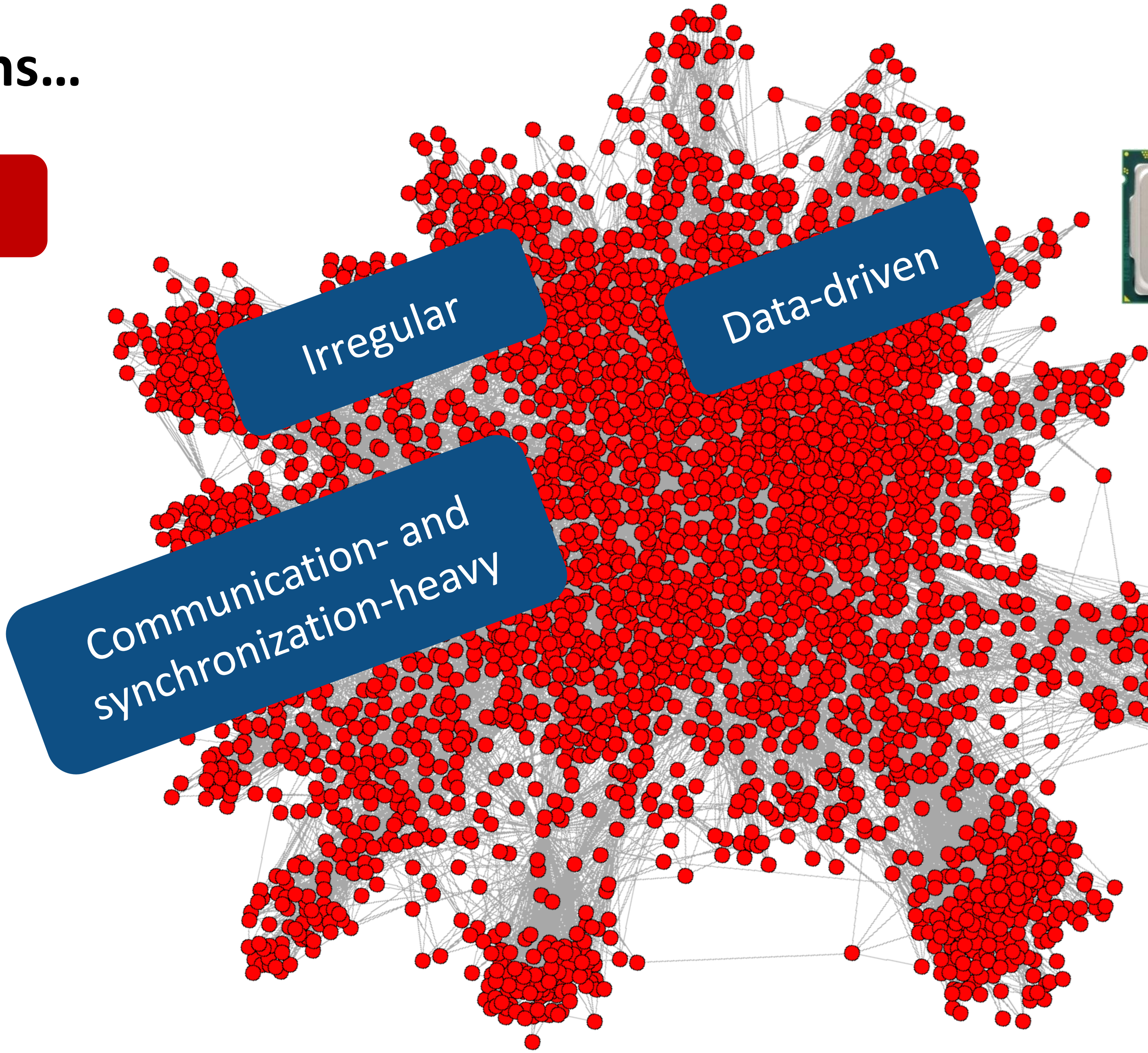
Large graphs…

Problems?



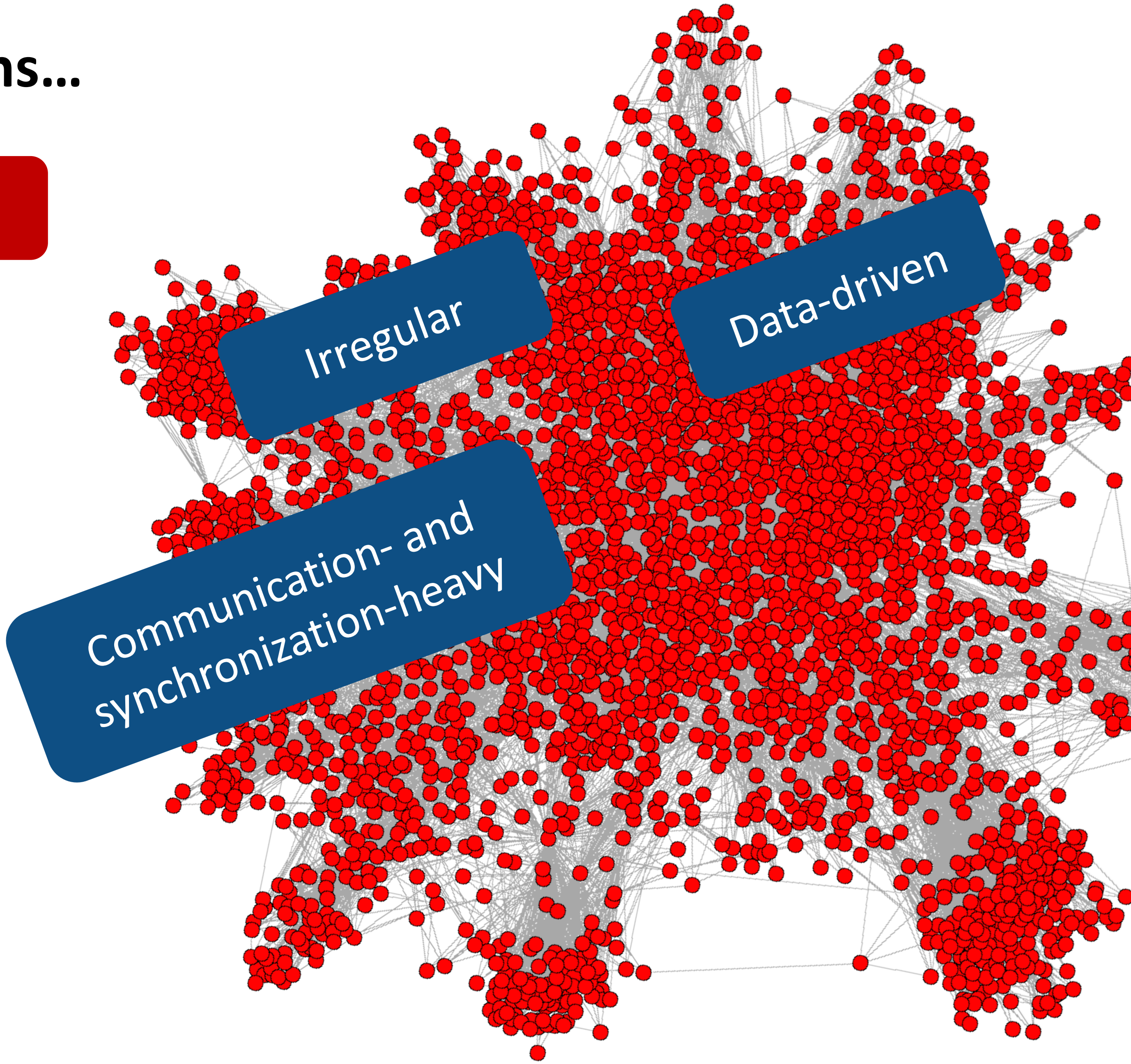
Large graphs…

Problems?



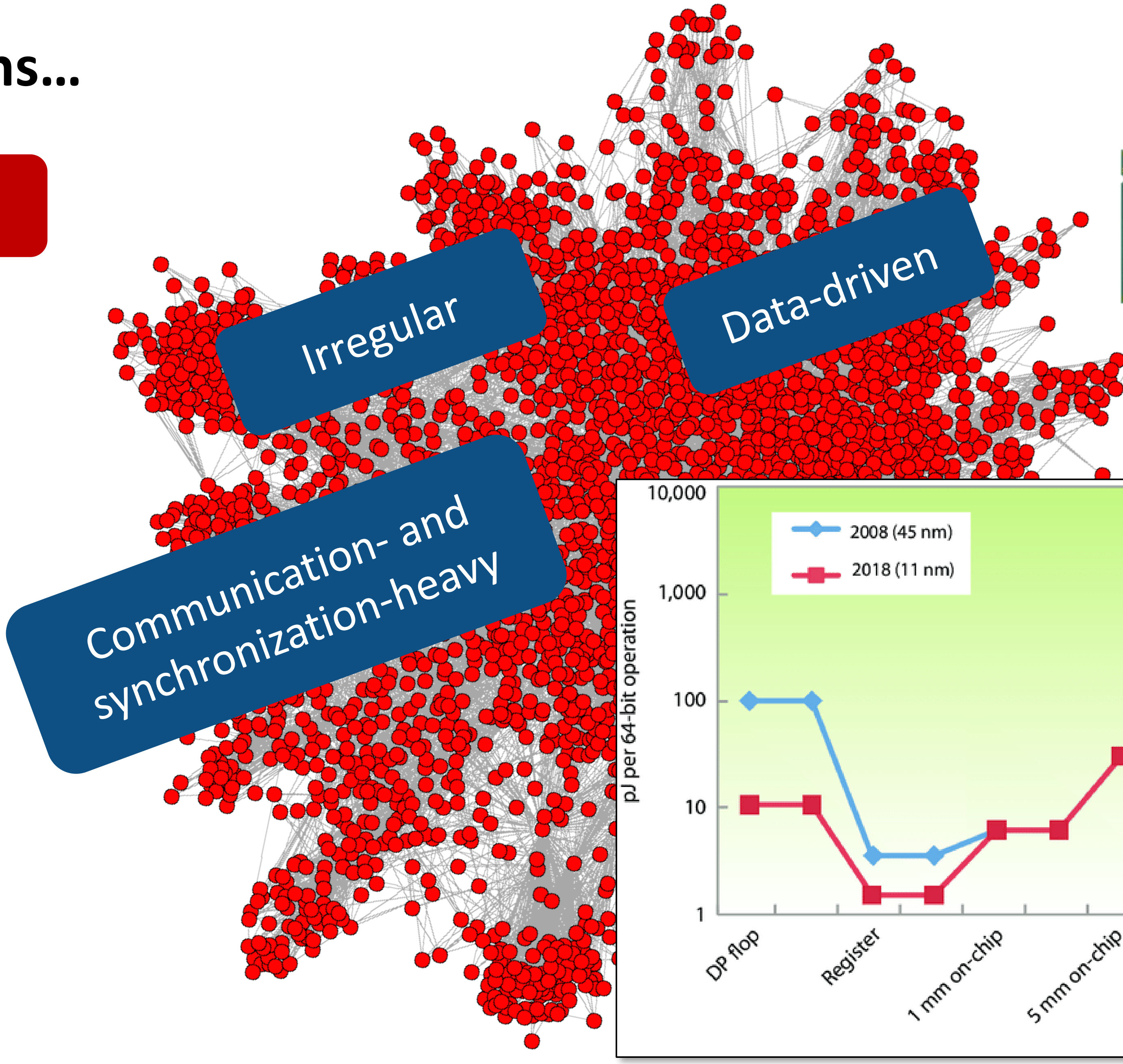
Large graphs…

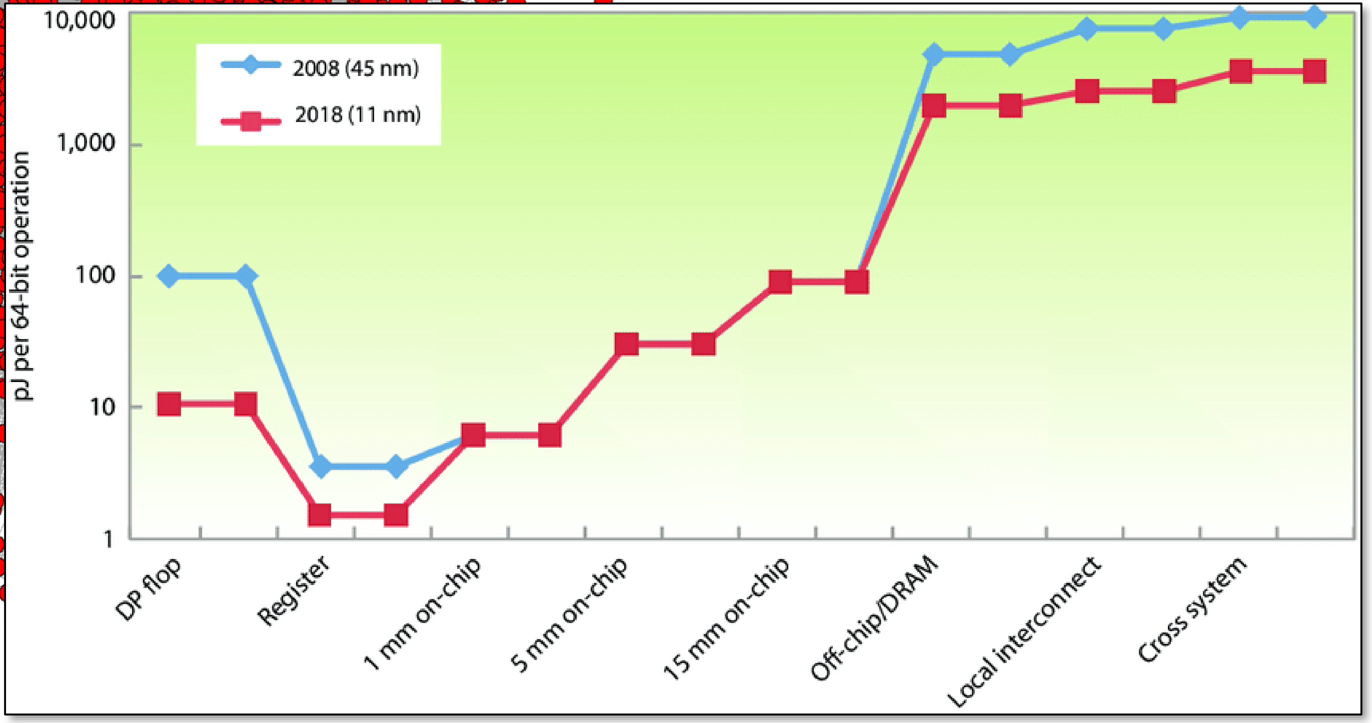
Problems?



Large graphs…

Problems?

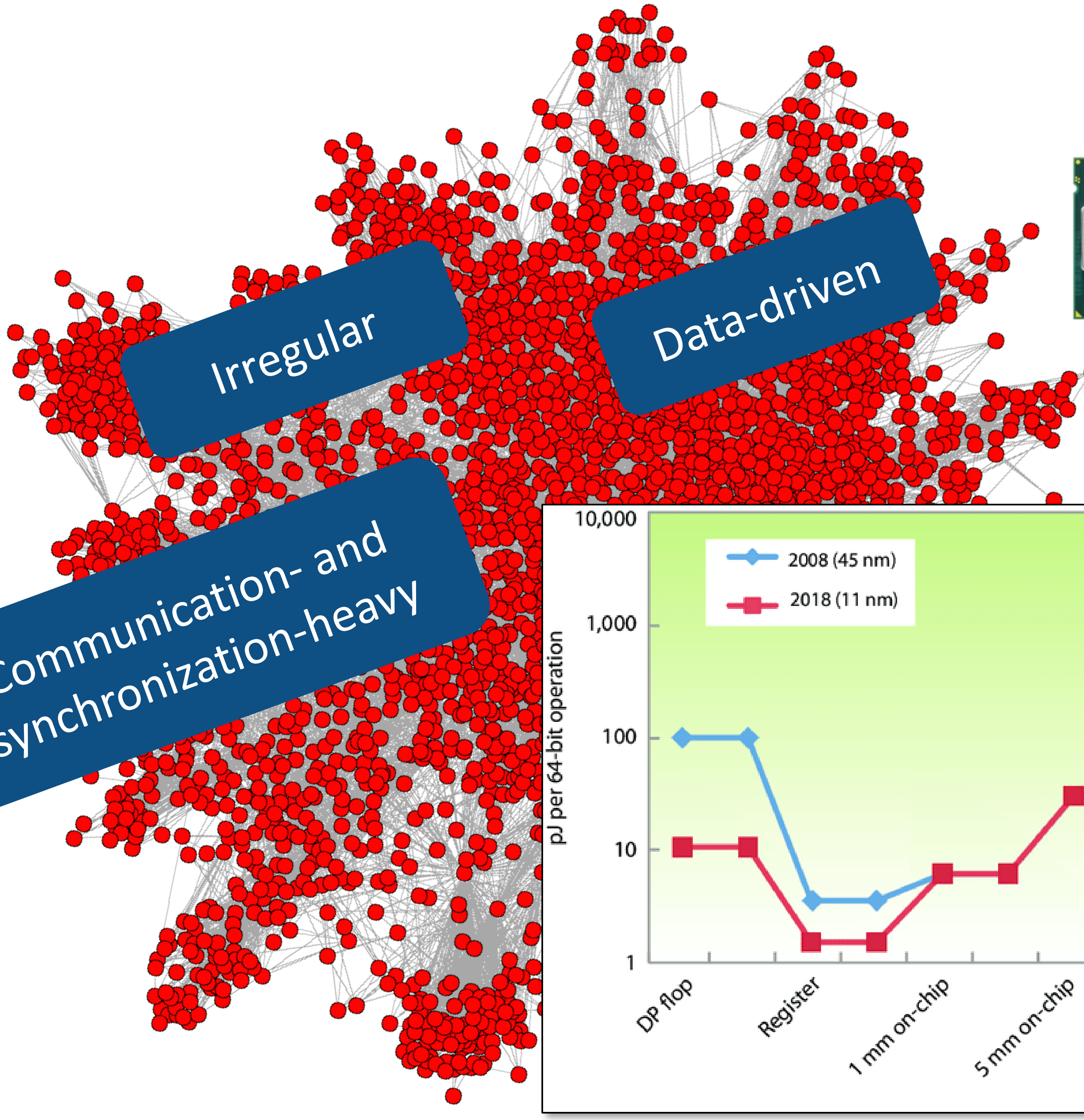




Large graphs…

Problems?

Low power efficiency!



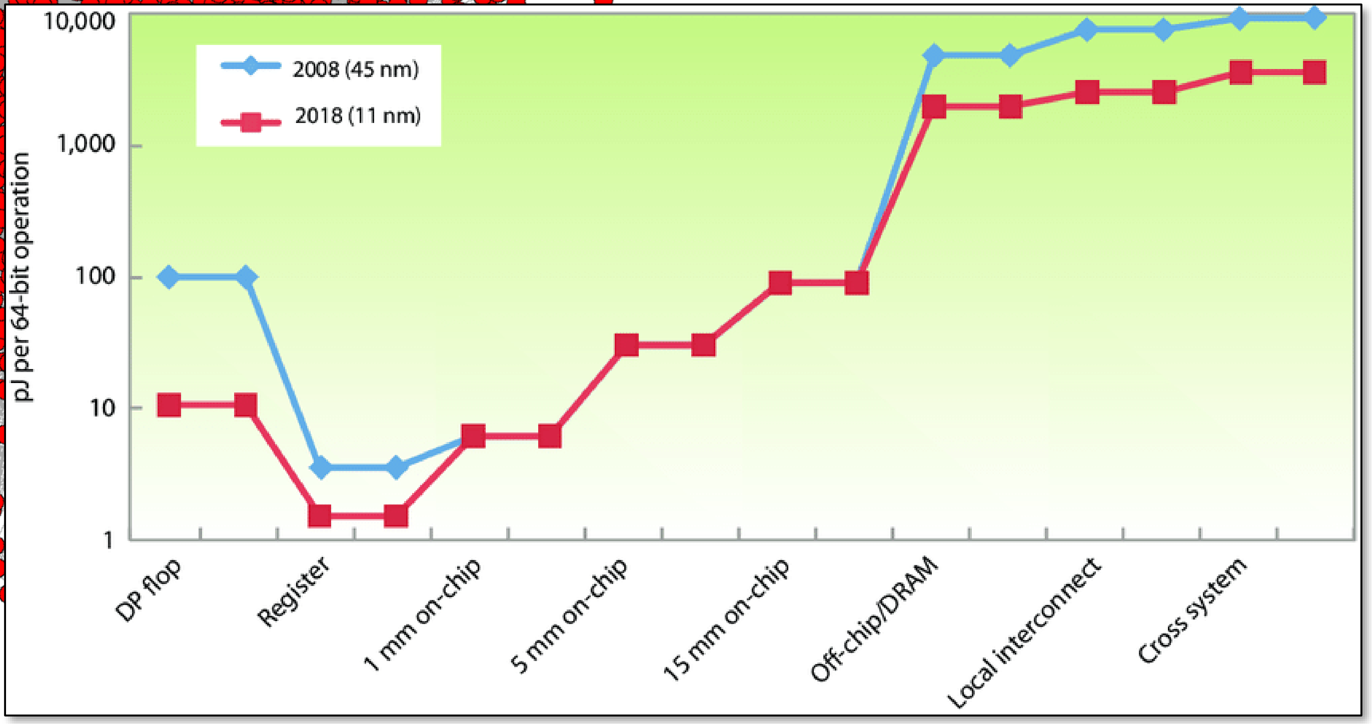
Irregular

Data-driven

Communication- and synchronization-heavy

120 Watts

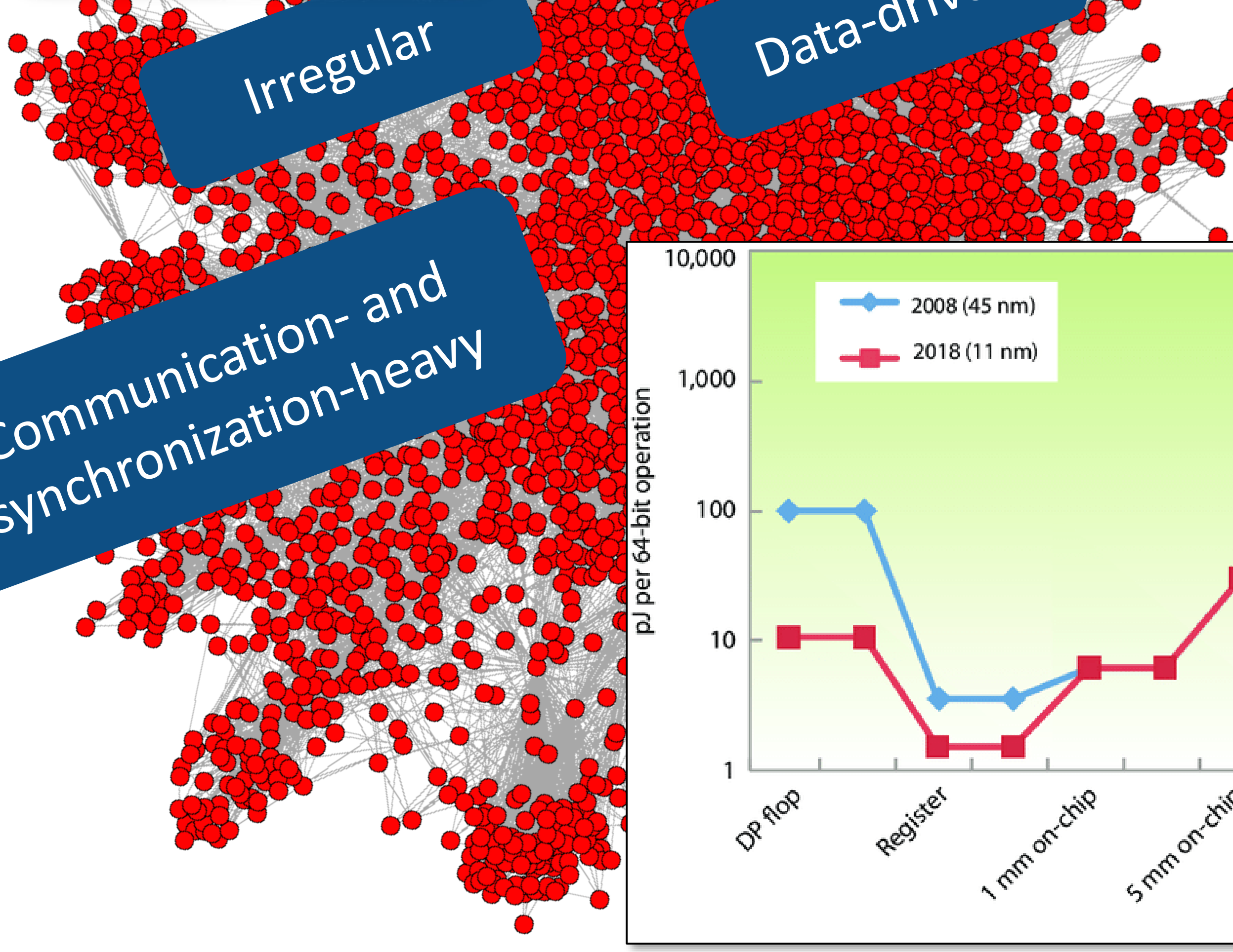
250 Watts



Large graphs…

Problems?

Low power efficiency!



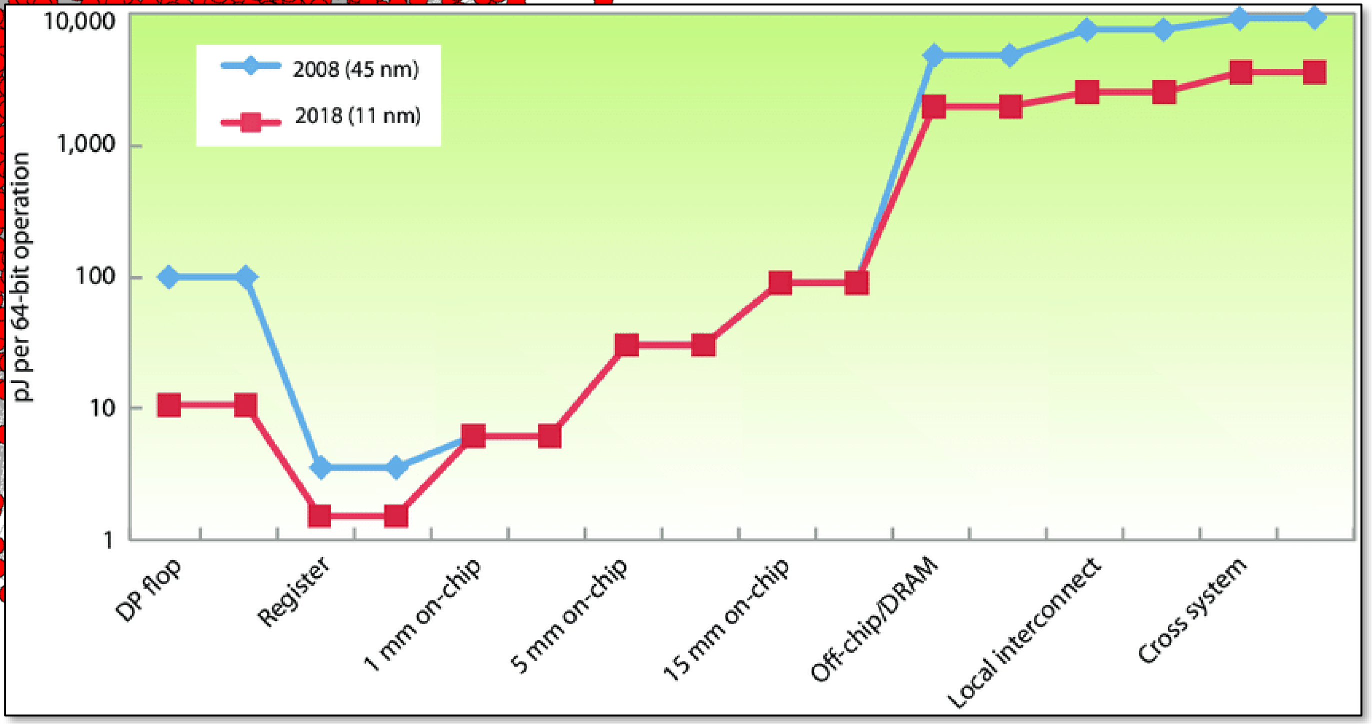
Irregular

Data-driven

Communication- and synchronization-heavy

120 Watts

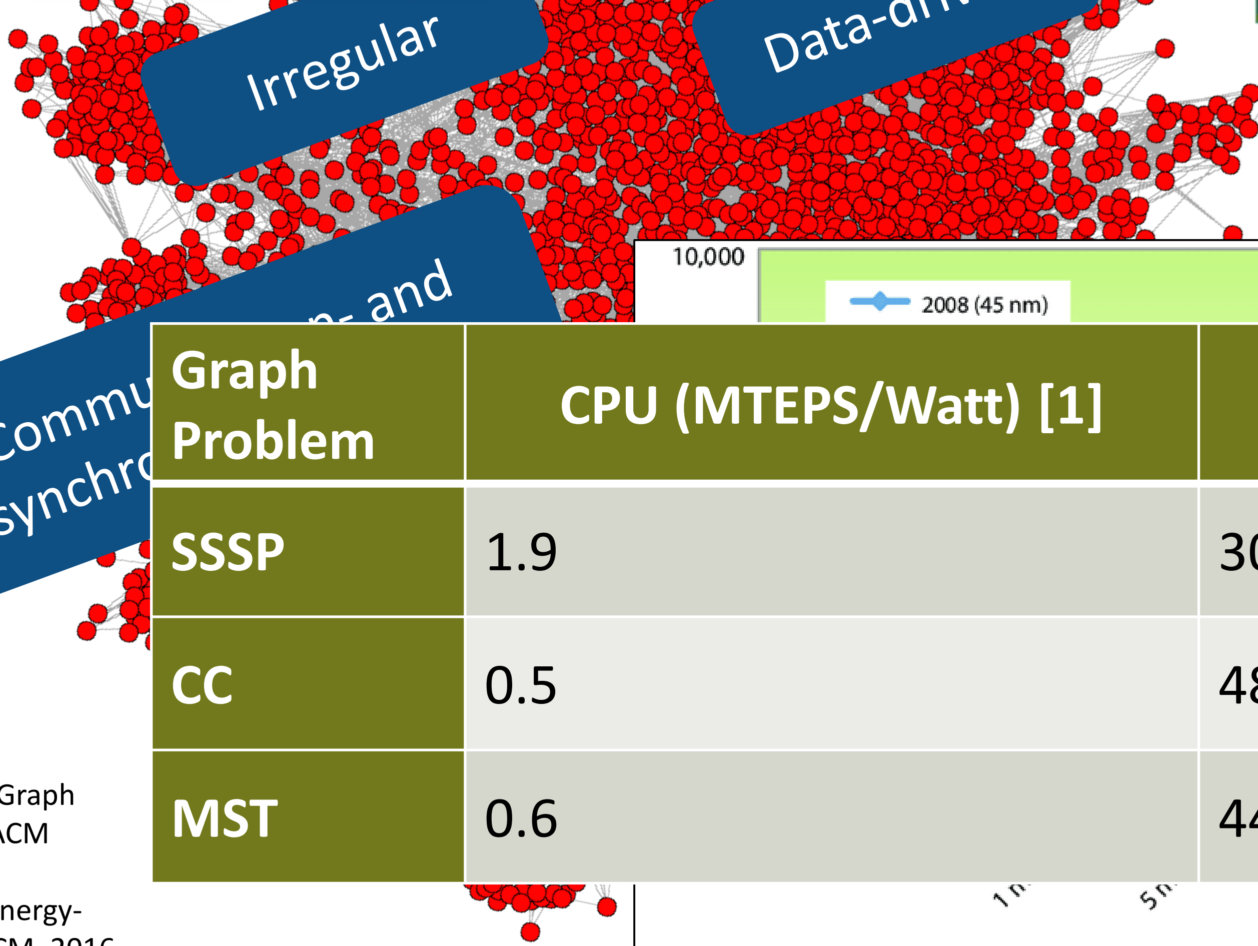
250 Watts



Large graphs...

Problems?

Low power efficiency!



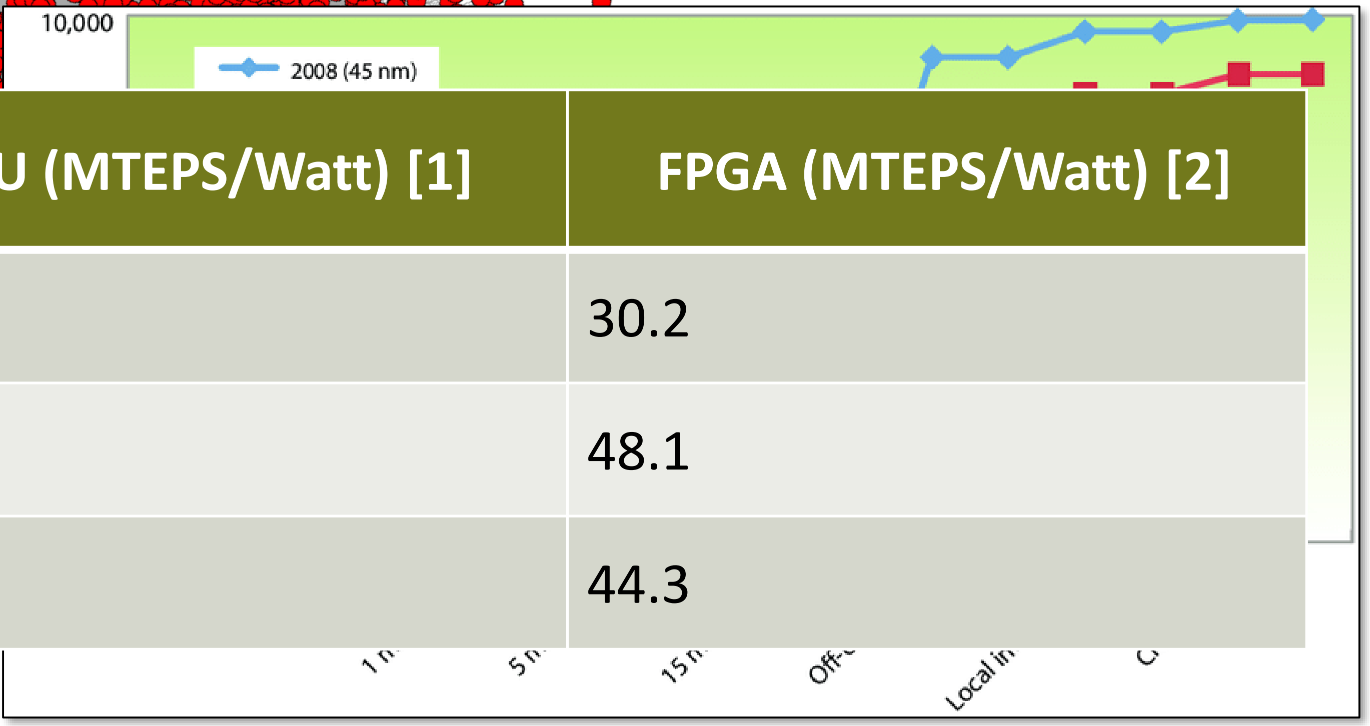
Irregular

Data-driven

Commu...n- and synchr...

120 Watts

250 Watts



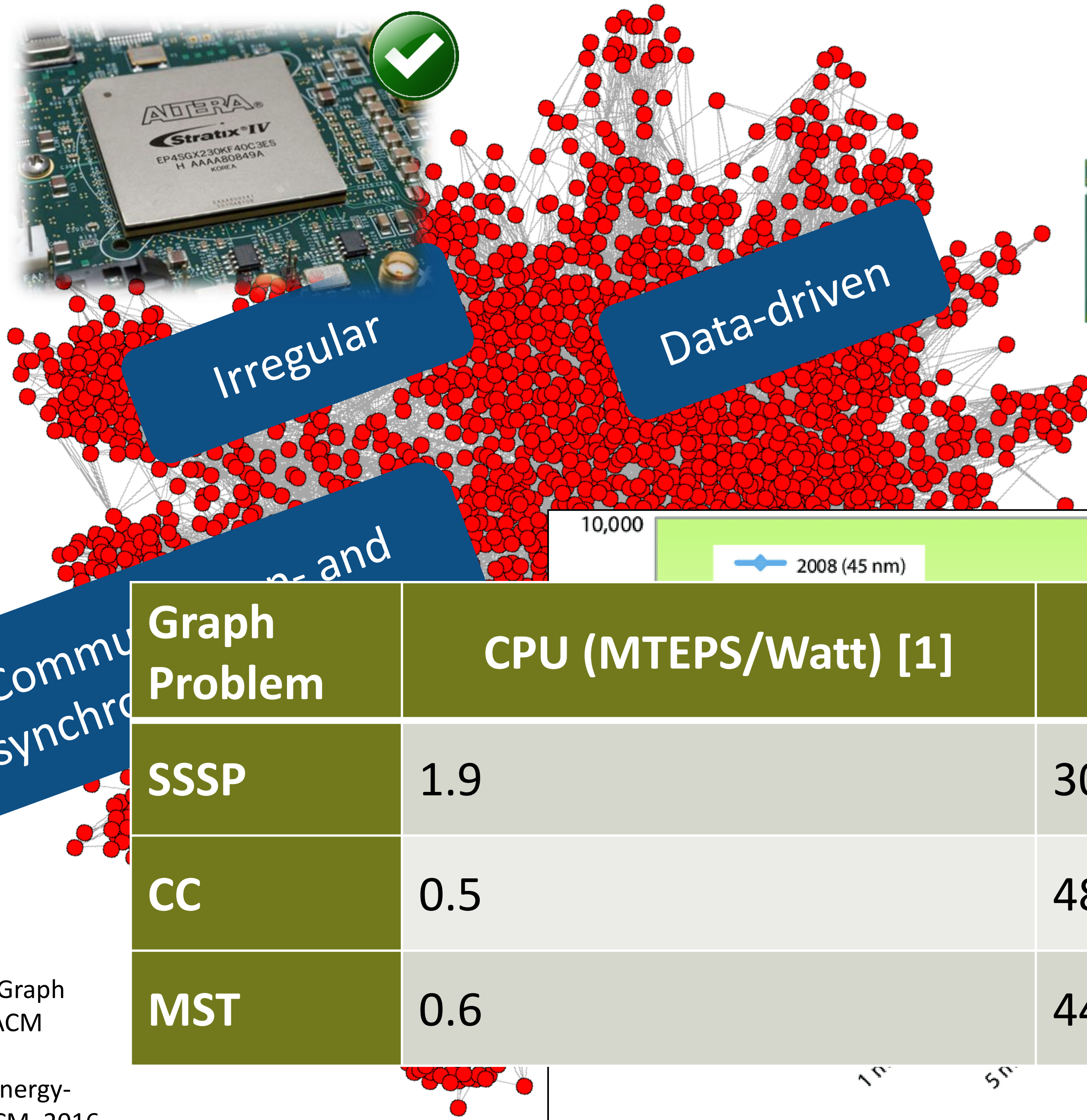
Graph Problem	CPU (MTEPS/Watt) [1]	FPGA (MTEPS/Watt) [2]
SSSP	1.9	30.2
CC	0.5	48.1
MST	0.6	44.3

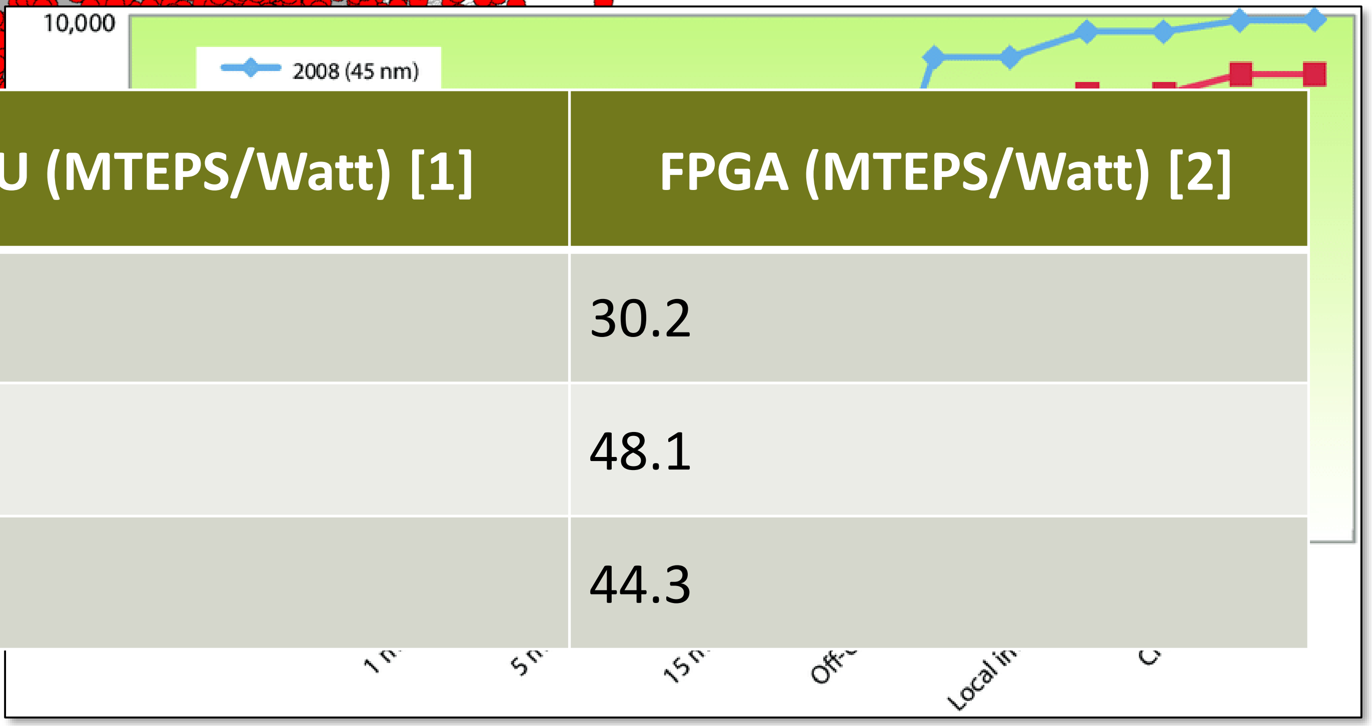
[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.
[2] S. Zhou et al. High-throughput and Energy-efficient Graph Processing on FPGA. FCCM. 2016.

Large graphs…

Problems?

Low power efficiency!





Graph Problem	CPU (MTEPS/Watt) [1]	FPGA (MTEPS/Watt) [2]
SSSP	1.9	30.2
CC	0.5	48.1
MST	0.6	44.3

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.
[2] S. Zhou et al. High-throughput and Energy-efficient Graph Processing on FPGA. FCCM. 2016.

Large graphs...

Problems?

Low power efficiency!

Irregular

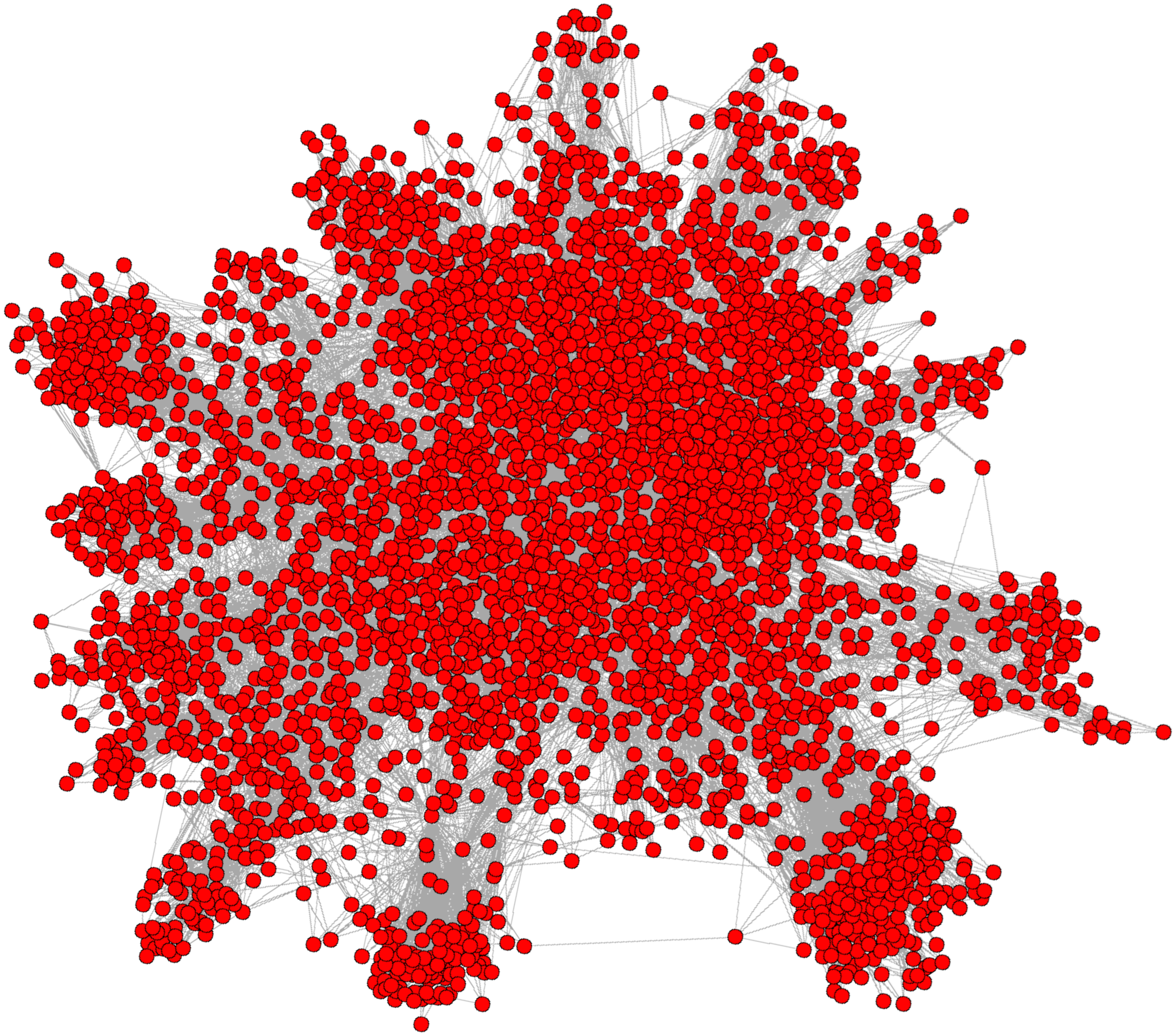
Communication- and synchronization

Let’s use FPGAs for Maximum Matchings...

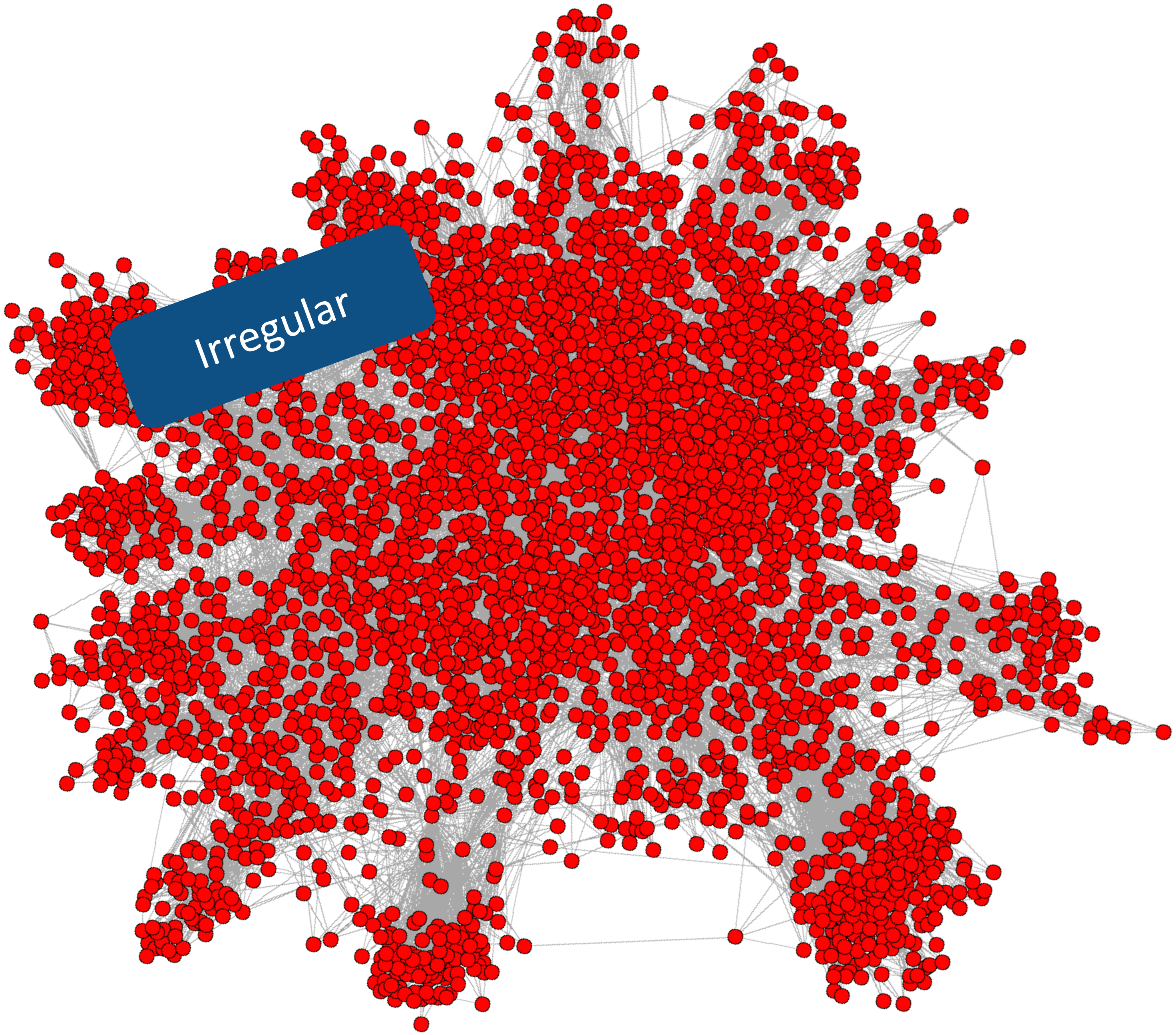
Graph Problem	CPU (MTEPS/Watt) [1]	FPGA (MTEPS/Watt) [2]
SSSP	1.9	30.2
CC	0.5	48.1
MST	0.6	44.3

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.
[2] S. Zhou et al. High-throughput and Energy-efficient Graph Processing on FPGA. FCCM. 2016.

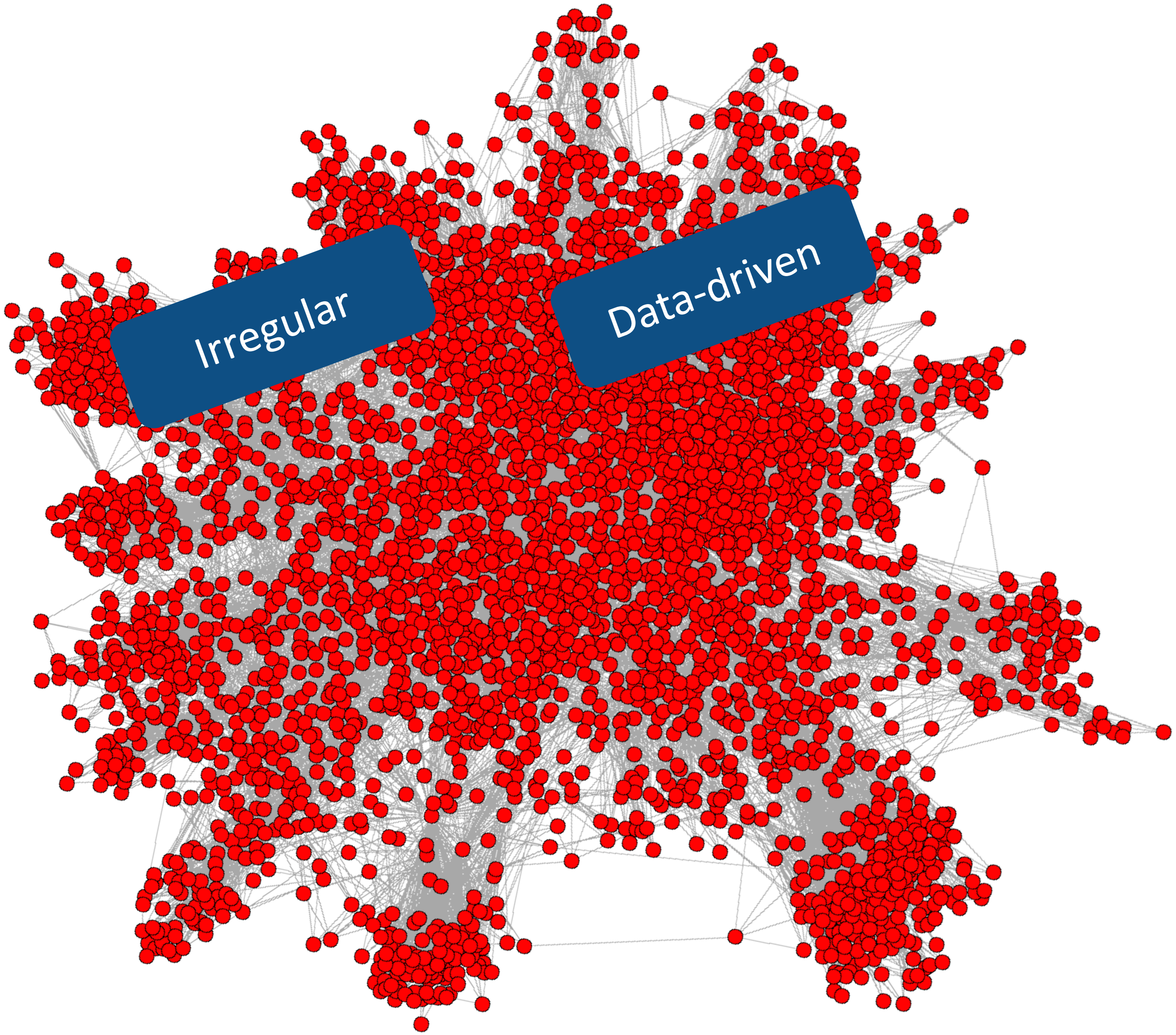
Large graphs…



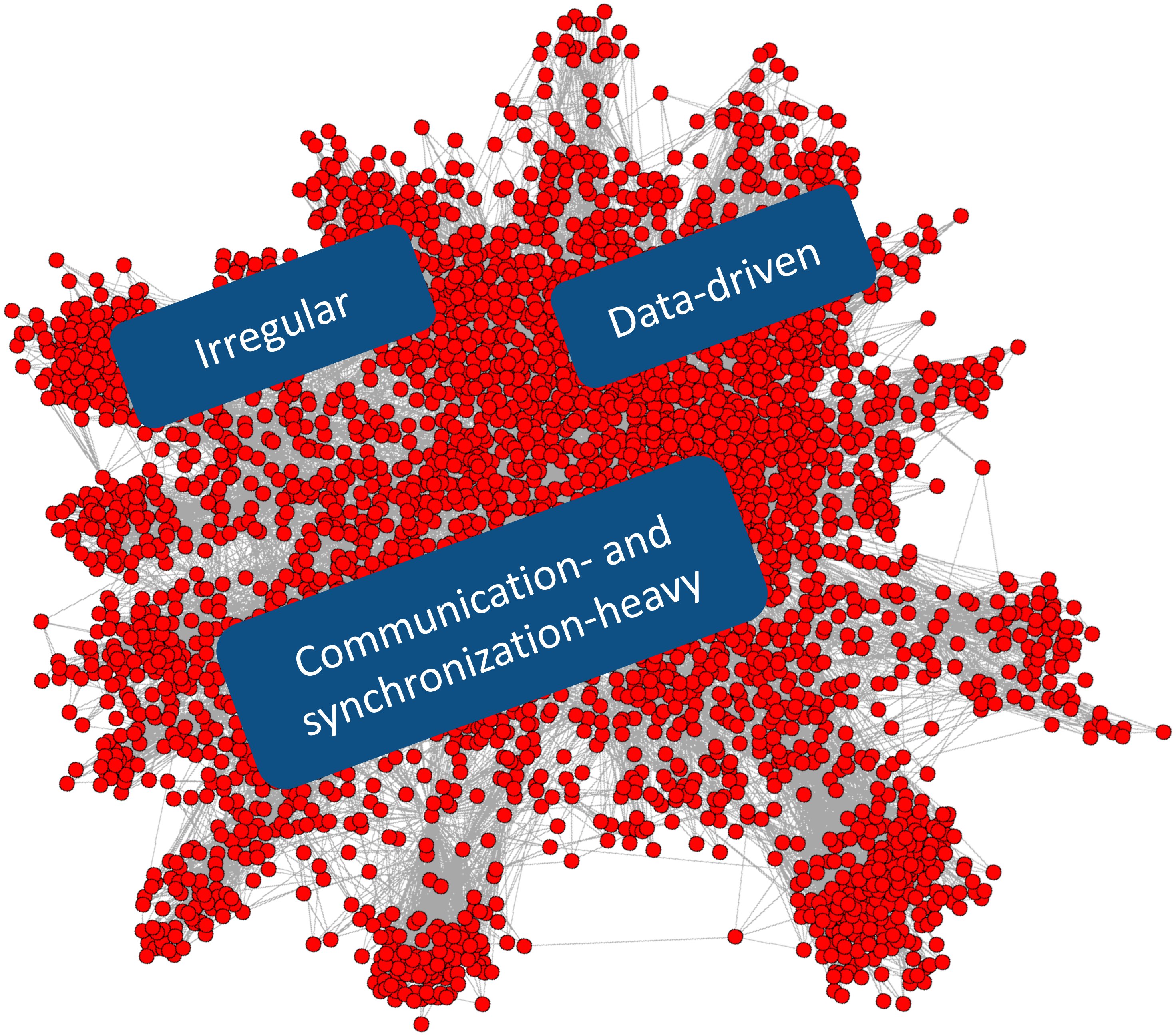
Large graphs…



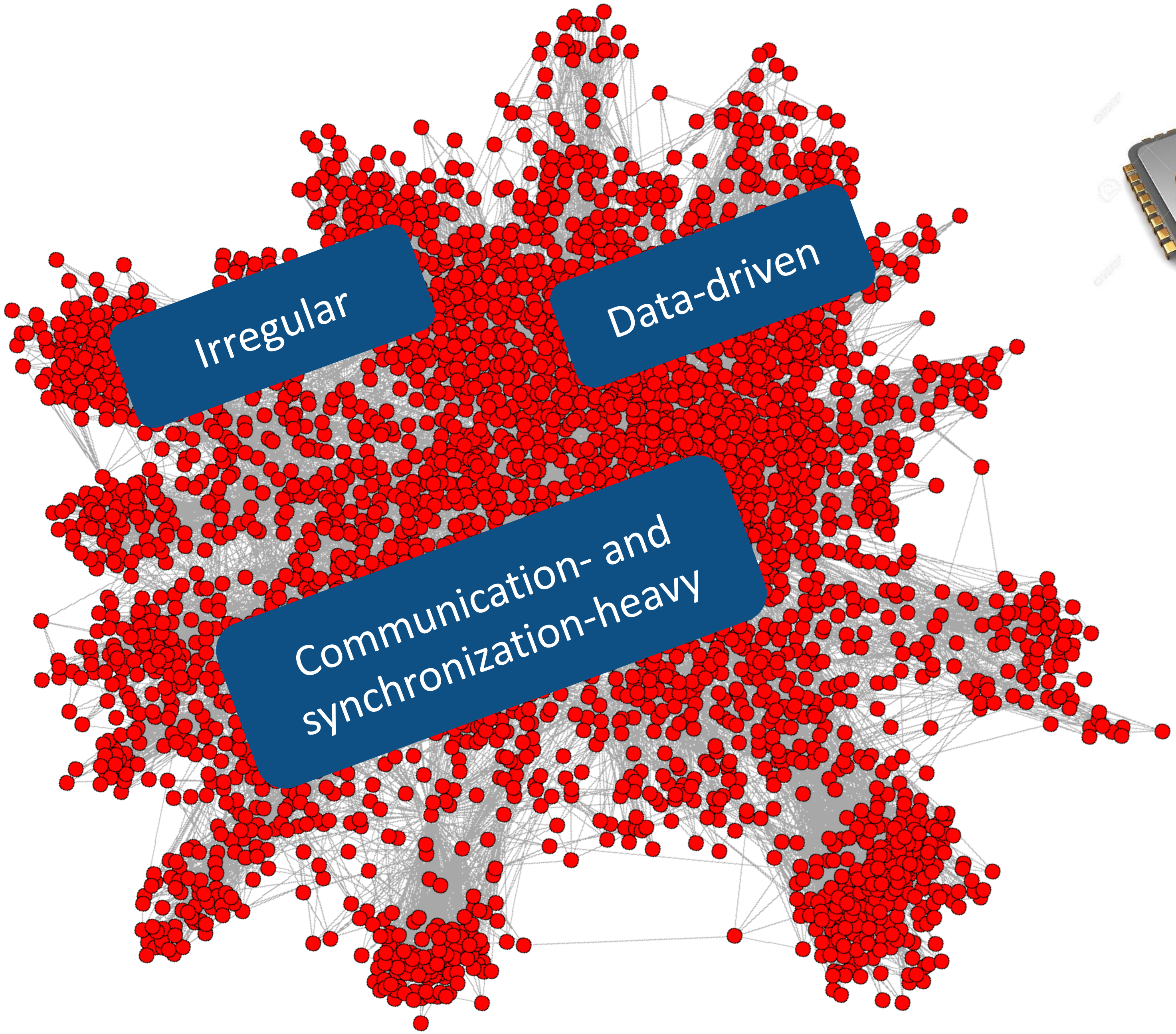
Large graphs...



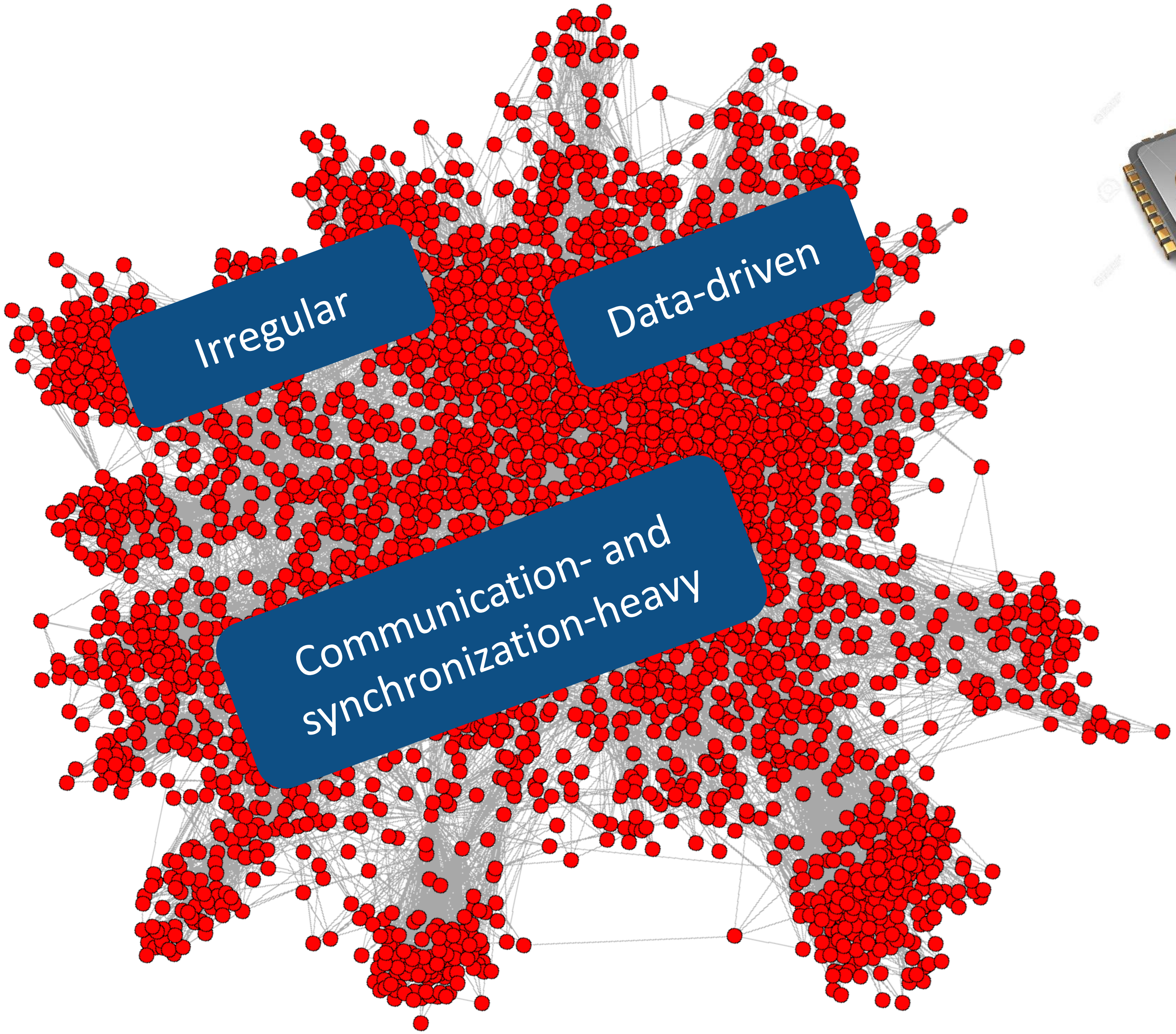
Large graphs…



Large graphs…

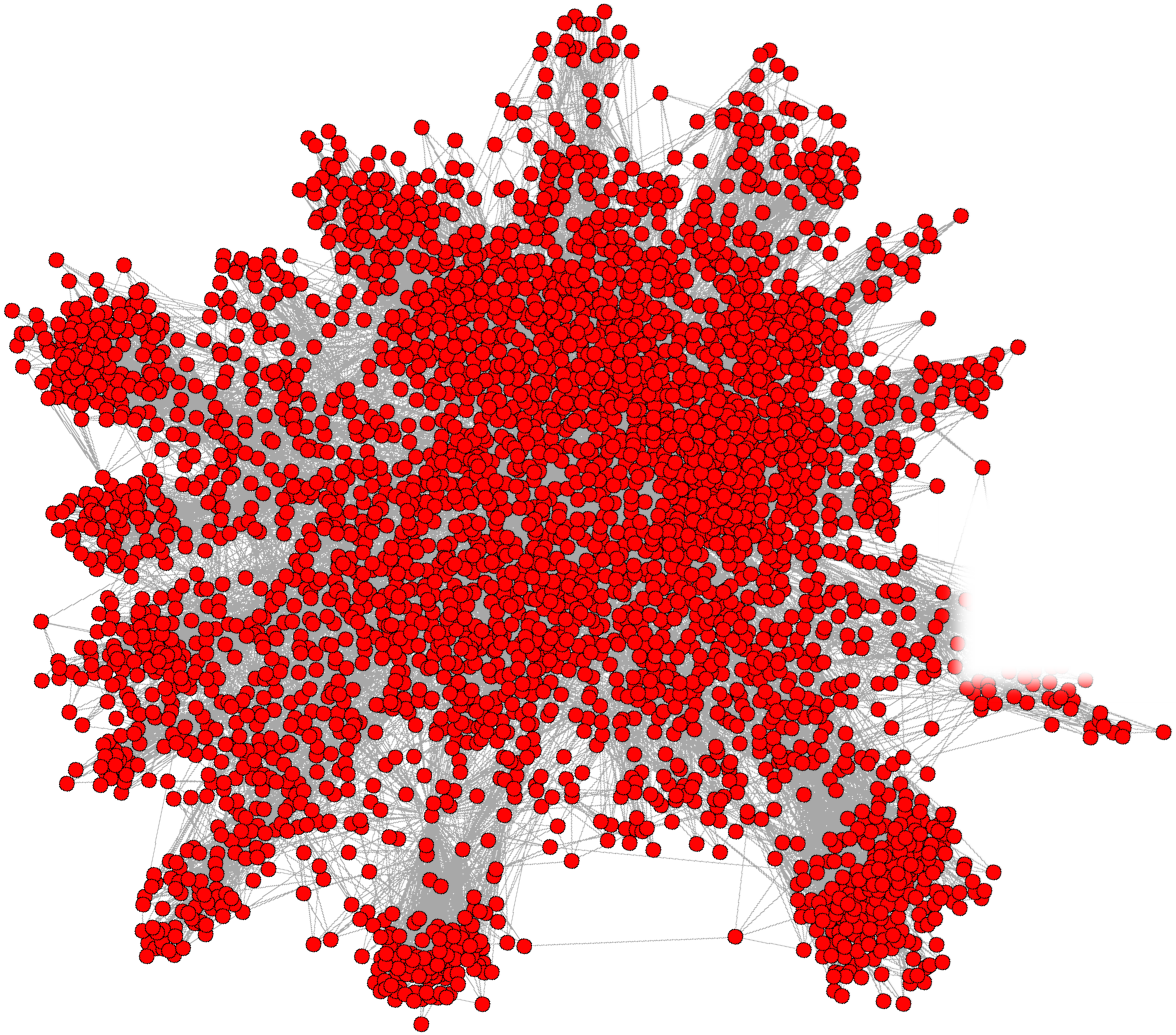


Large graphs…



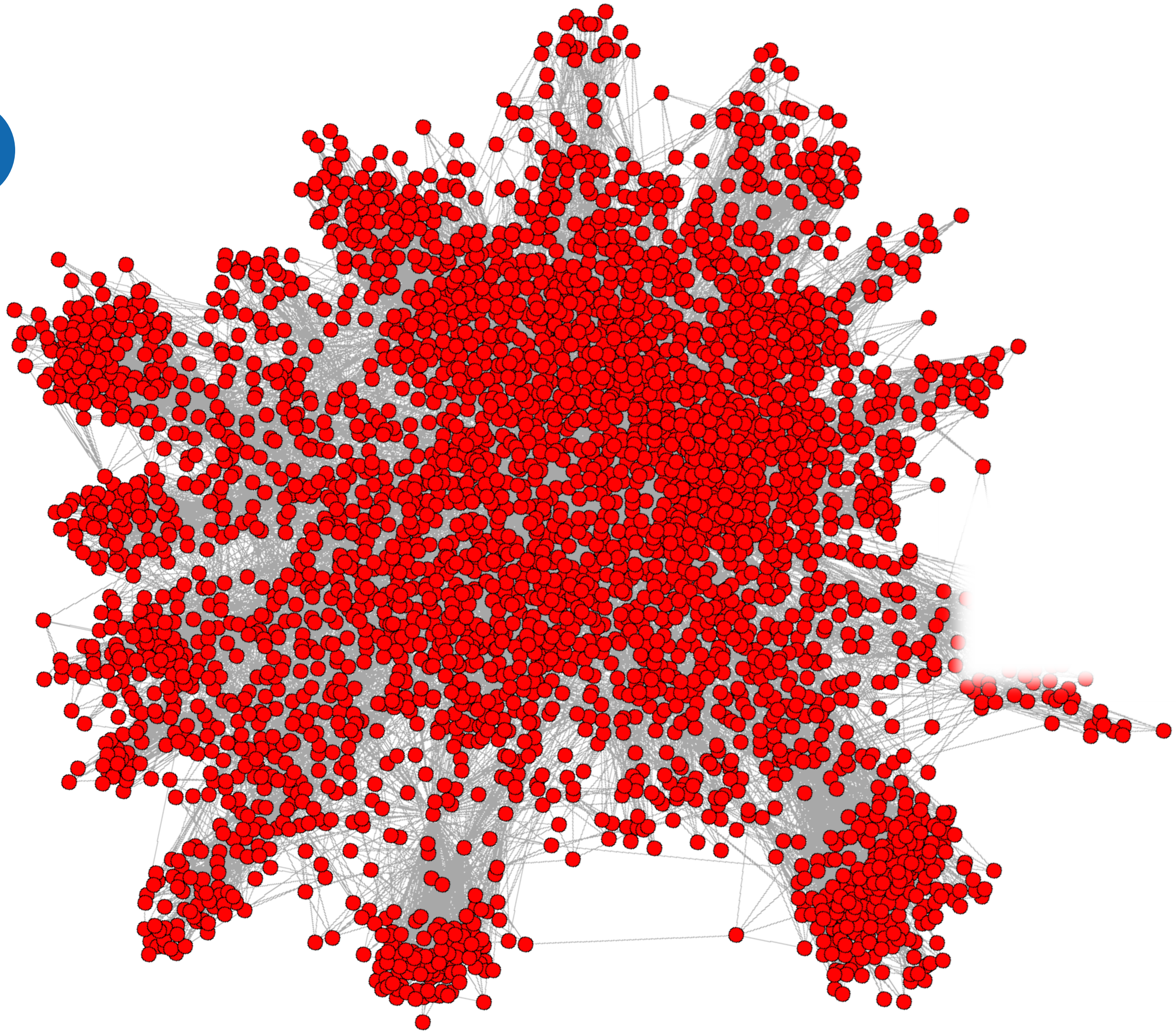
Large graphs…

Large graphs…



Large graphs…

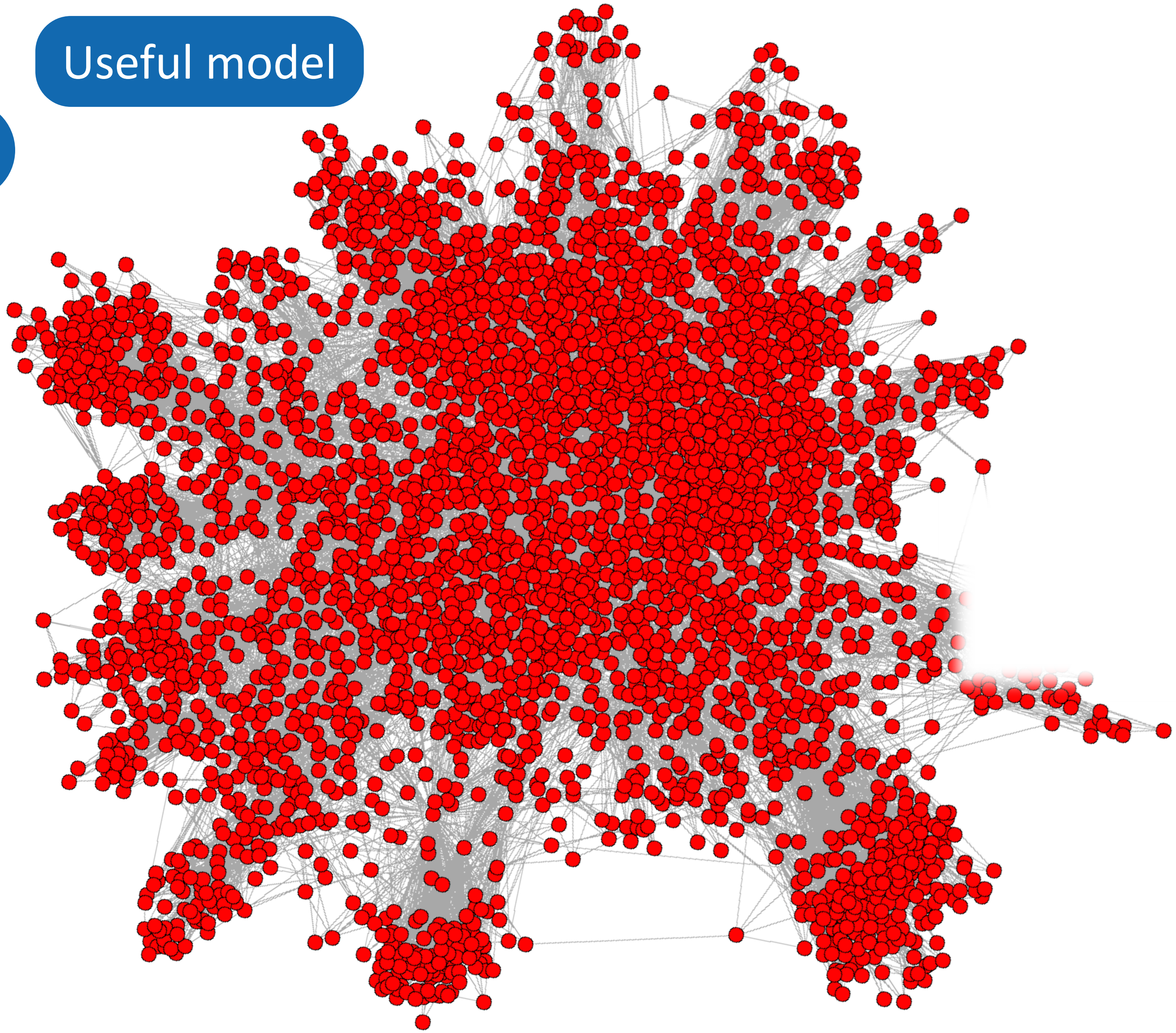
Why do we care?



Large graphs...

Why do we care?

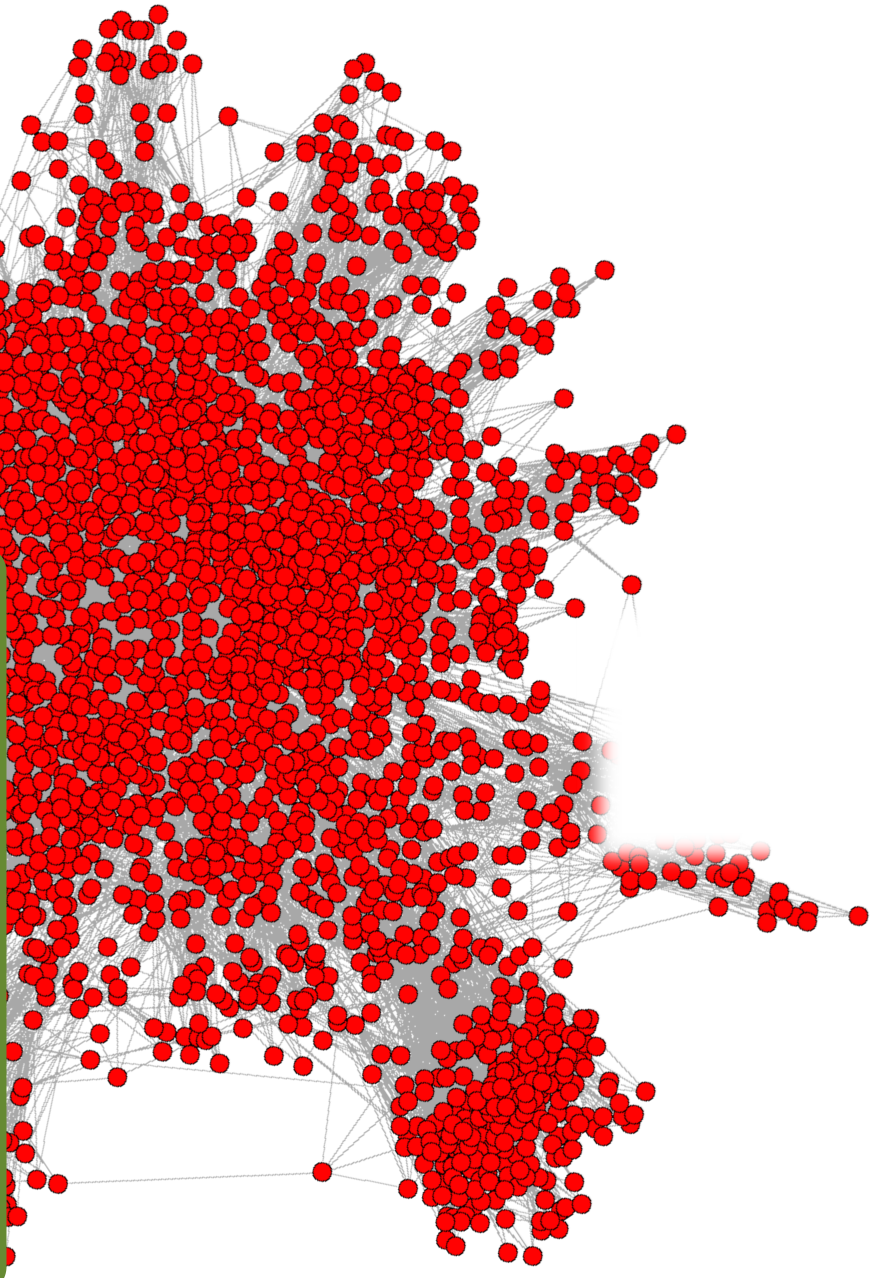
Useful model



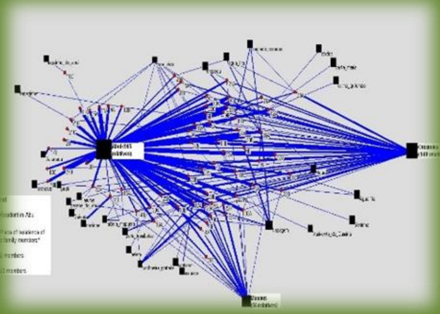
Large graphs…

Useful model

Why do we care?



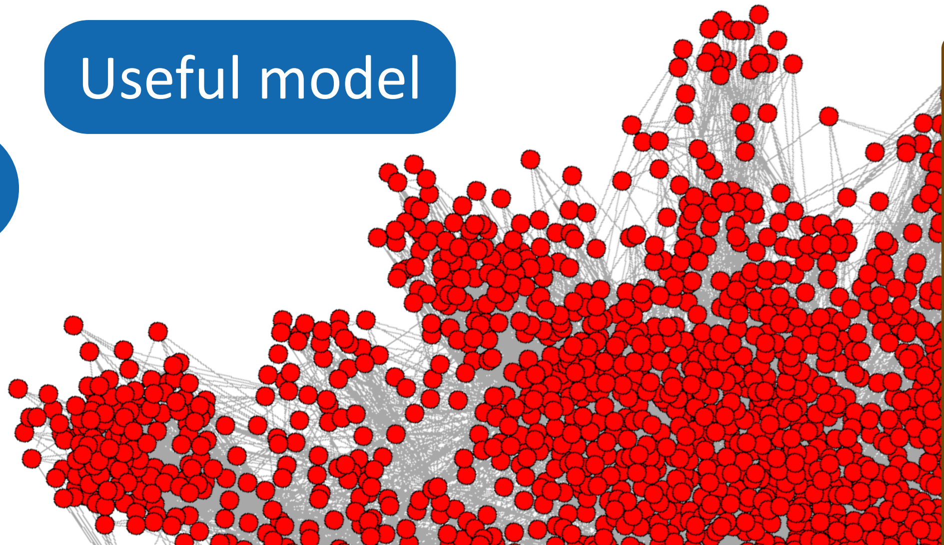
Social networks



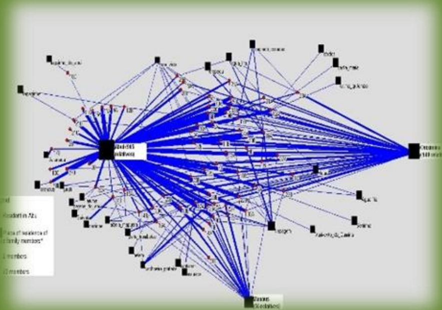
Large graphs...

Useful model

Why do we care?



Social networks



Engineering networks

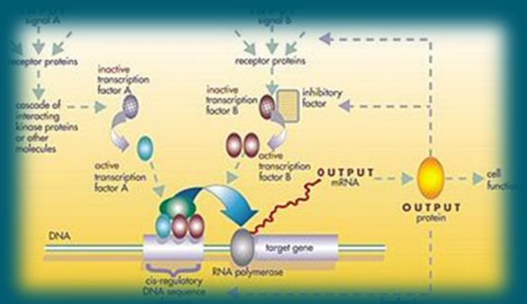
Large graphs…

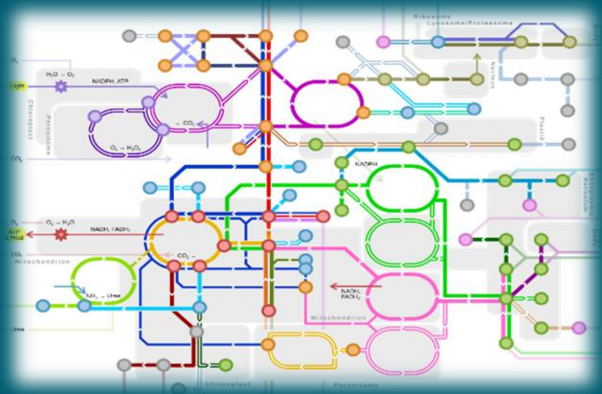
Useful model

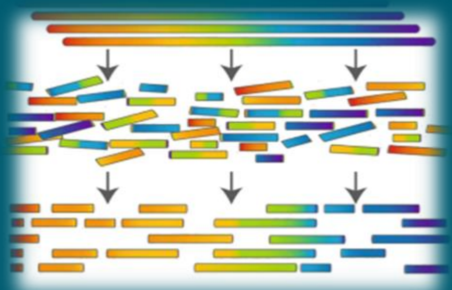
Why do we care?

Social networks

Biological networks





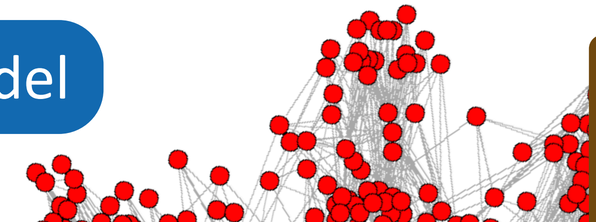


Engineering networks

Large graphs…

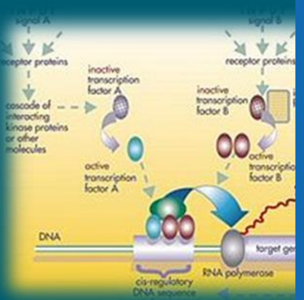
Useful model

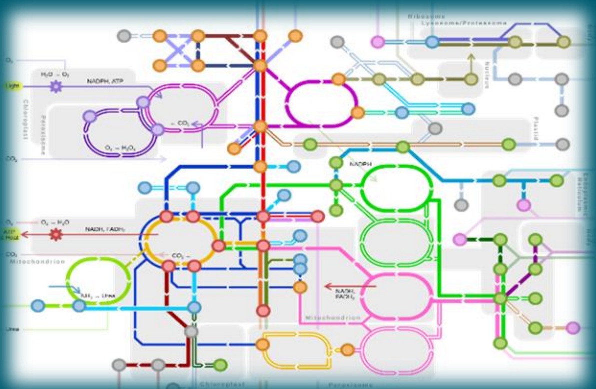
Why do we care?

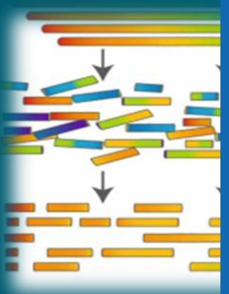


Social networks

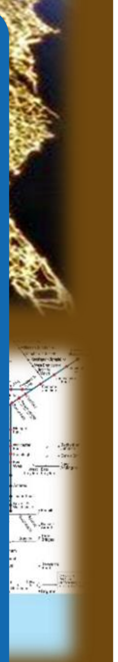
Biological networks



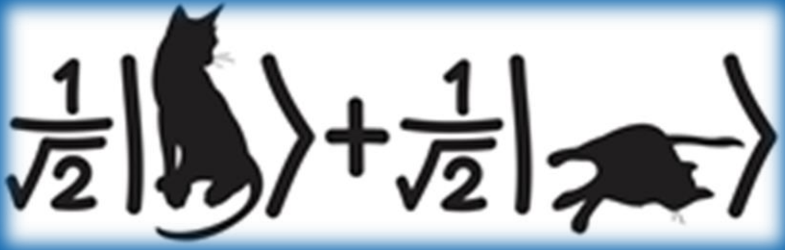


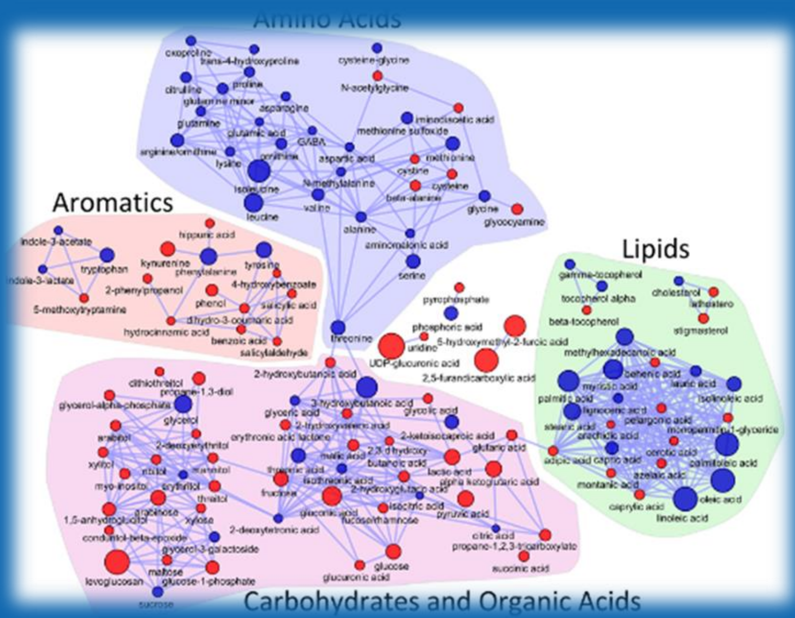


Engineering networks



Physics, chemistry

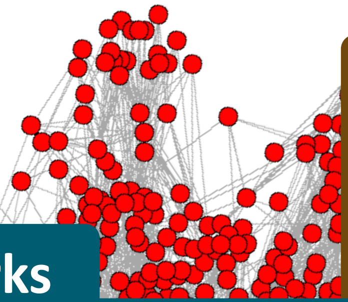




Large graphs…

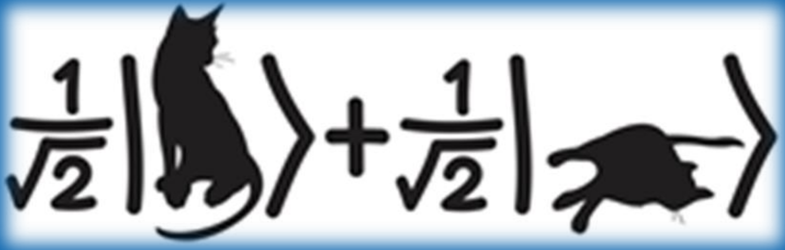
Useful model

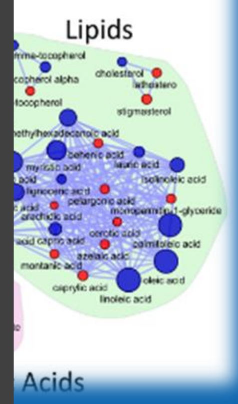
Why do we care?



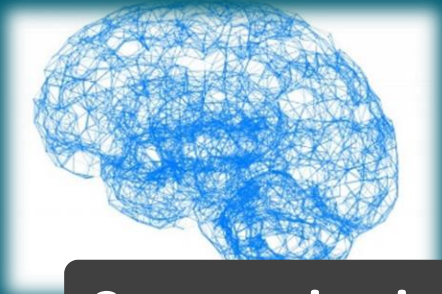
Engineering networks

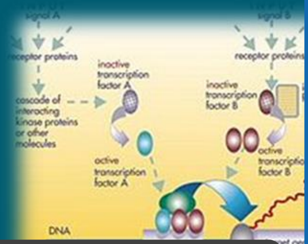
Physics, chemistry

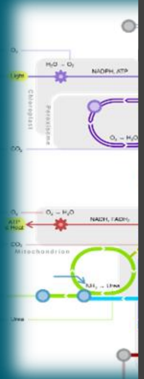




Biological networks

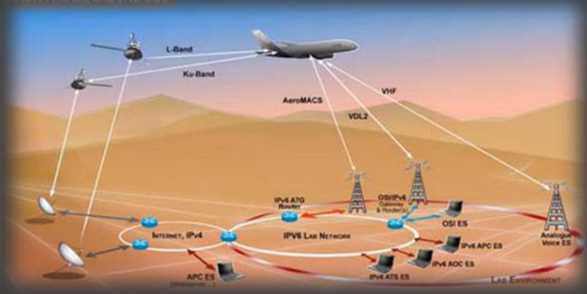


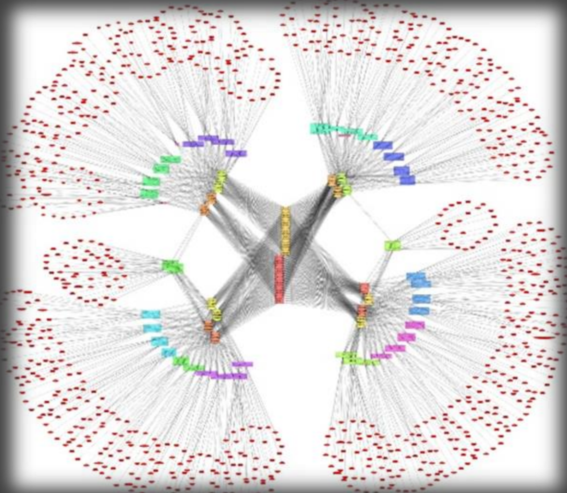




Social networks

Communication networks

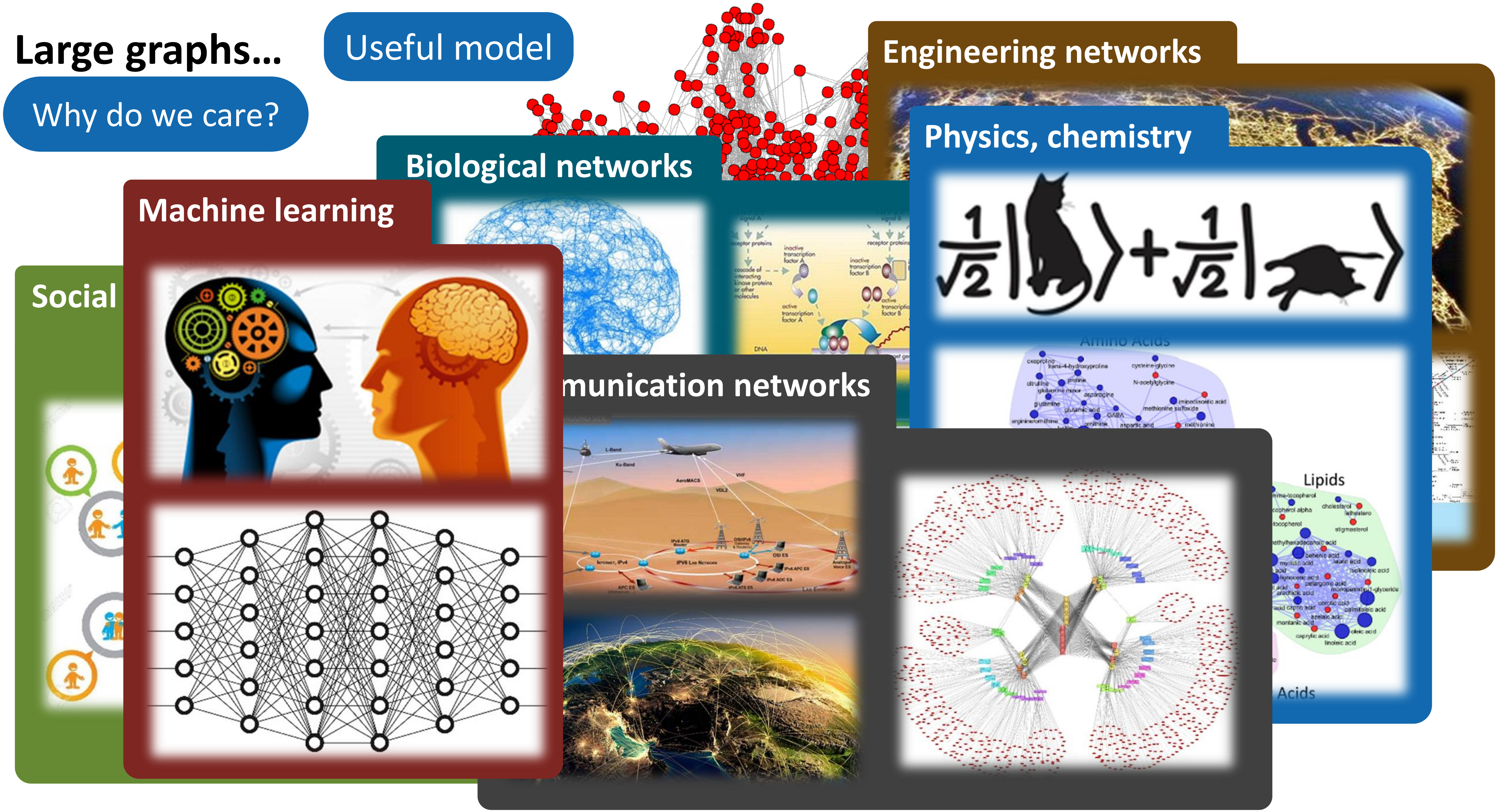




Large graphs…

Why do we care?

Useful model



Large graphs…

Useful model

Why do we care?

Engineering networks

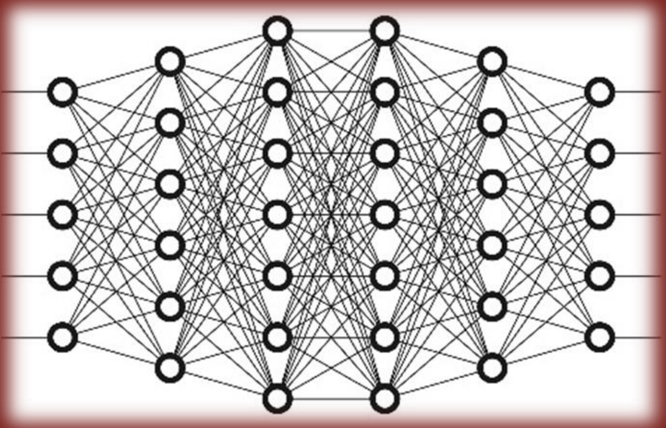
…even philosophy ☺

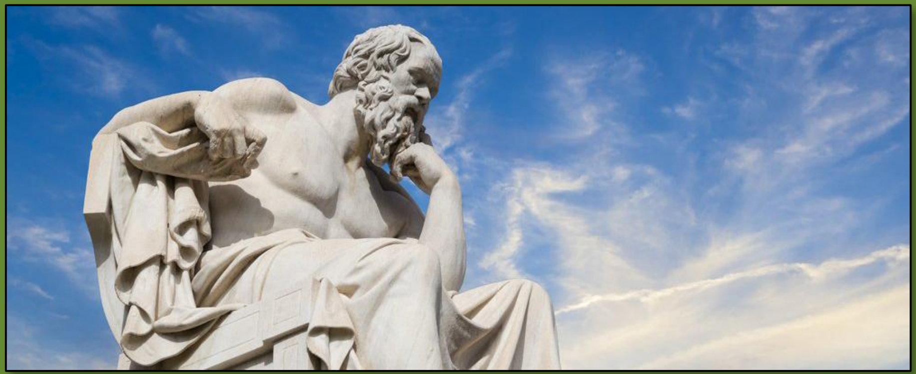
hysics, chemistry

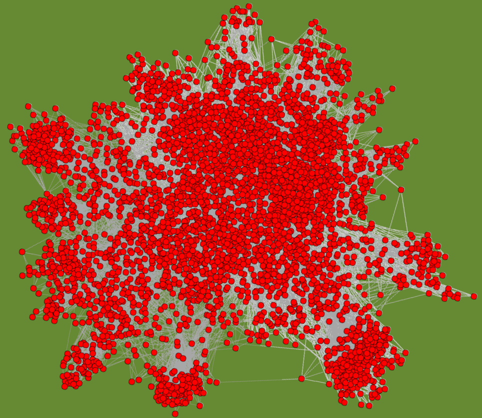
Biological ne

Machine learning

Social







FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

- **Track**: Graph Processing devroom
- **Room**: AW1.126
- **Day**: Saturday
- **Start**: 12:45
- **End**: 13:35

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

Large graphs...

Useful model

Why do we care?

Engineering networks

...even philosophy ☺

hysics, chemistry

Biological ne

Machine learning

Social

m

FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

- Track: Graph Processing devroom
- Room: AW1.126
- Day: Saturday
- Start: 12:45
- End: 13:35

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

Large graphs...

Useful model

Why do we care?

Engineering networks

...even philo

Machine learning

Social

Shortest network path?

Gene alignment?

Most reliable grid network?

Best phone connection?

Disease spread channels?

Least expensive computer network?

Collaborator experience?

Most relevant protein?

Brain structure?

Terrorism prevention?

Most efficient road network?

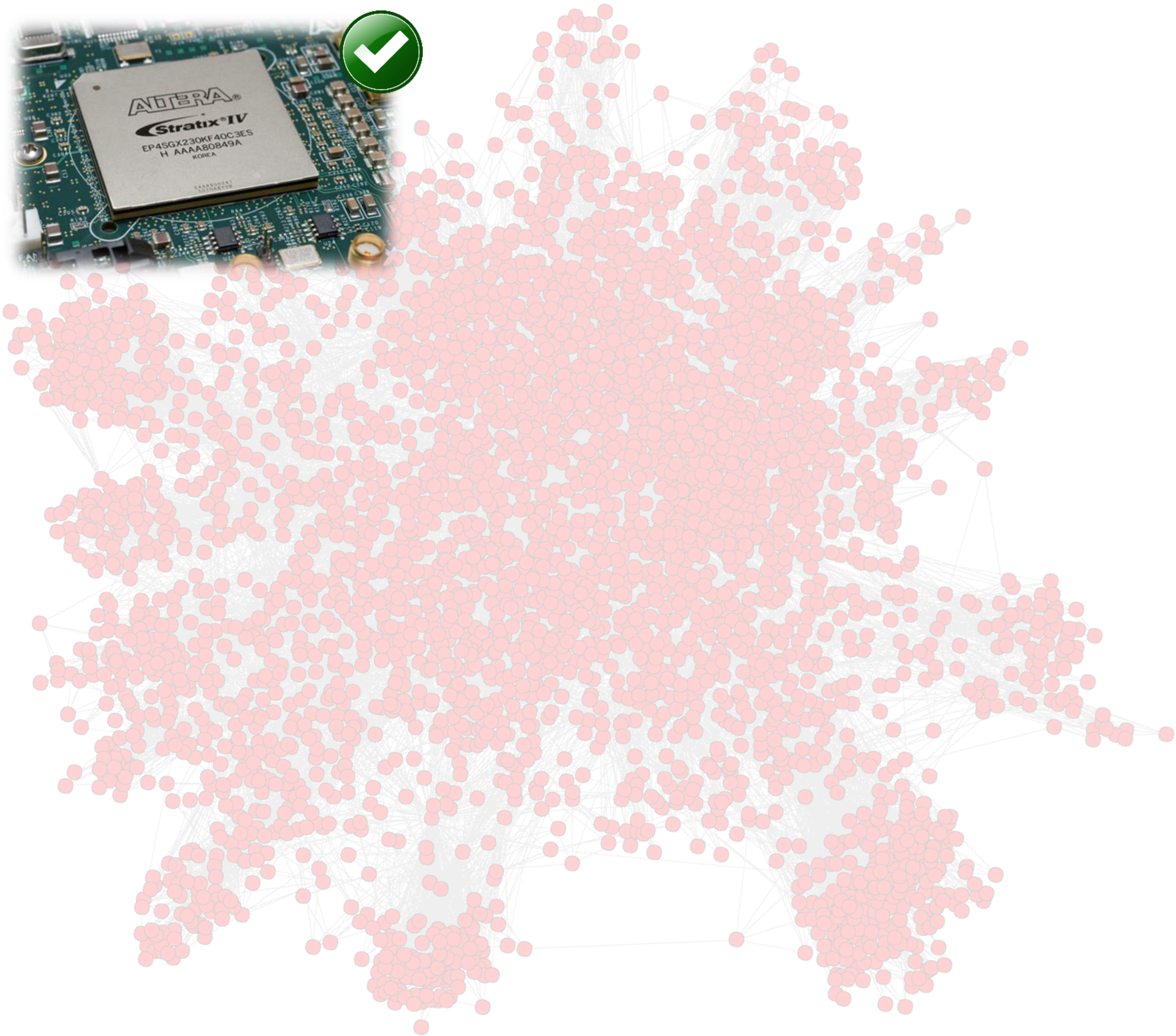
Shortest path?

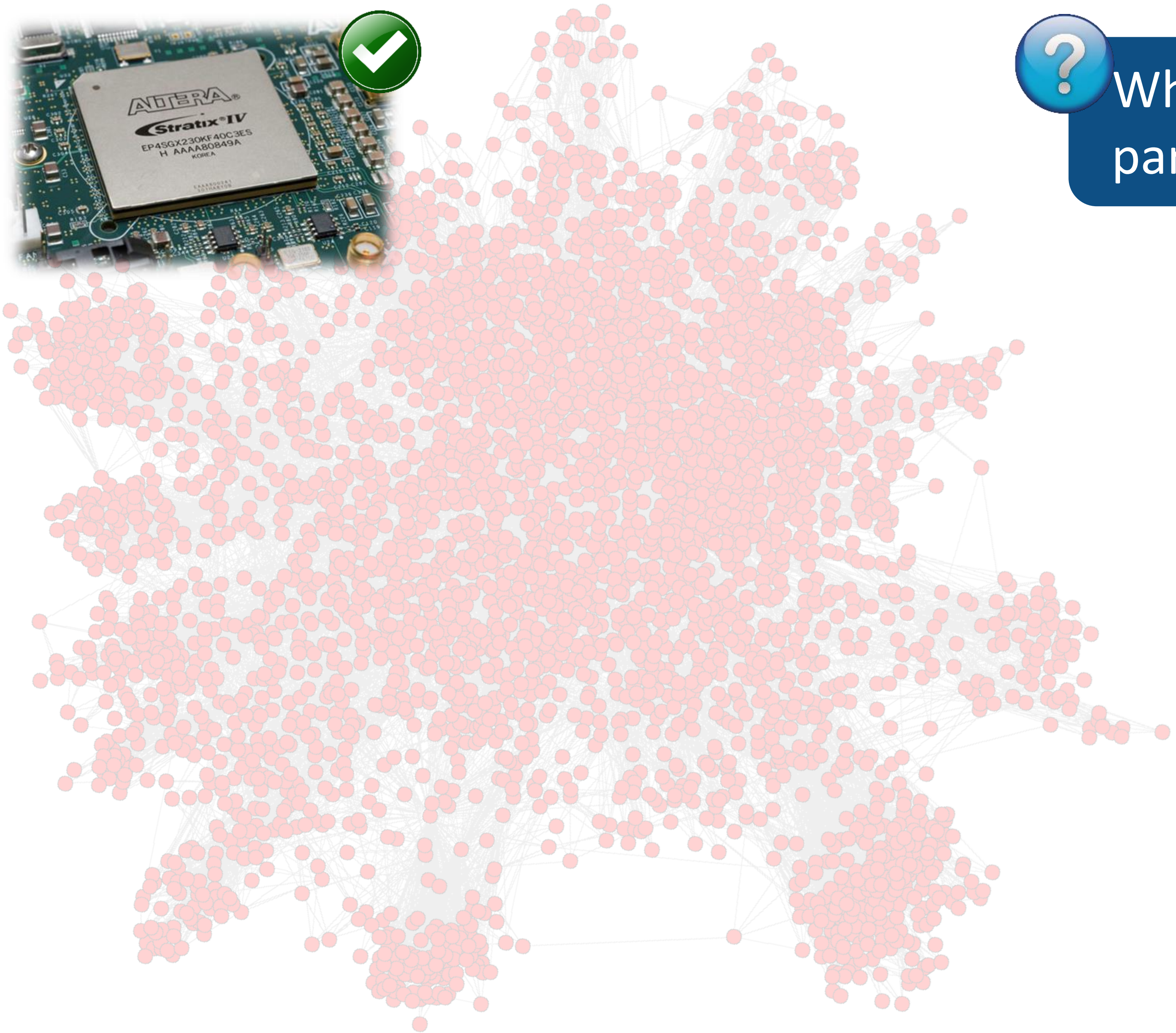
FOSDEM 2016 / Schedule / Events / Developer rooms / ... / Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

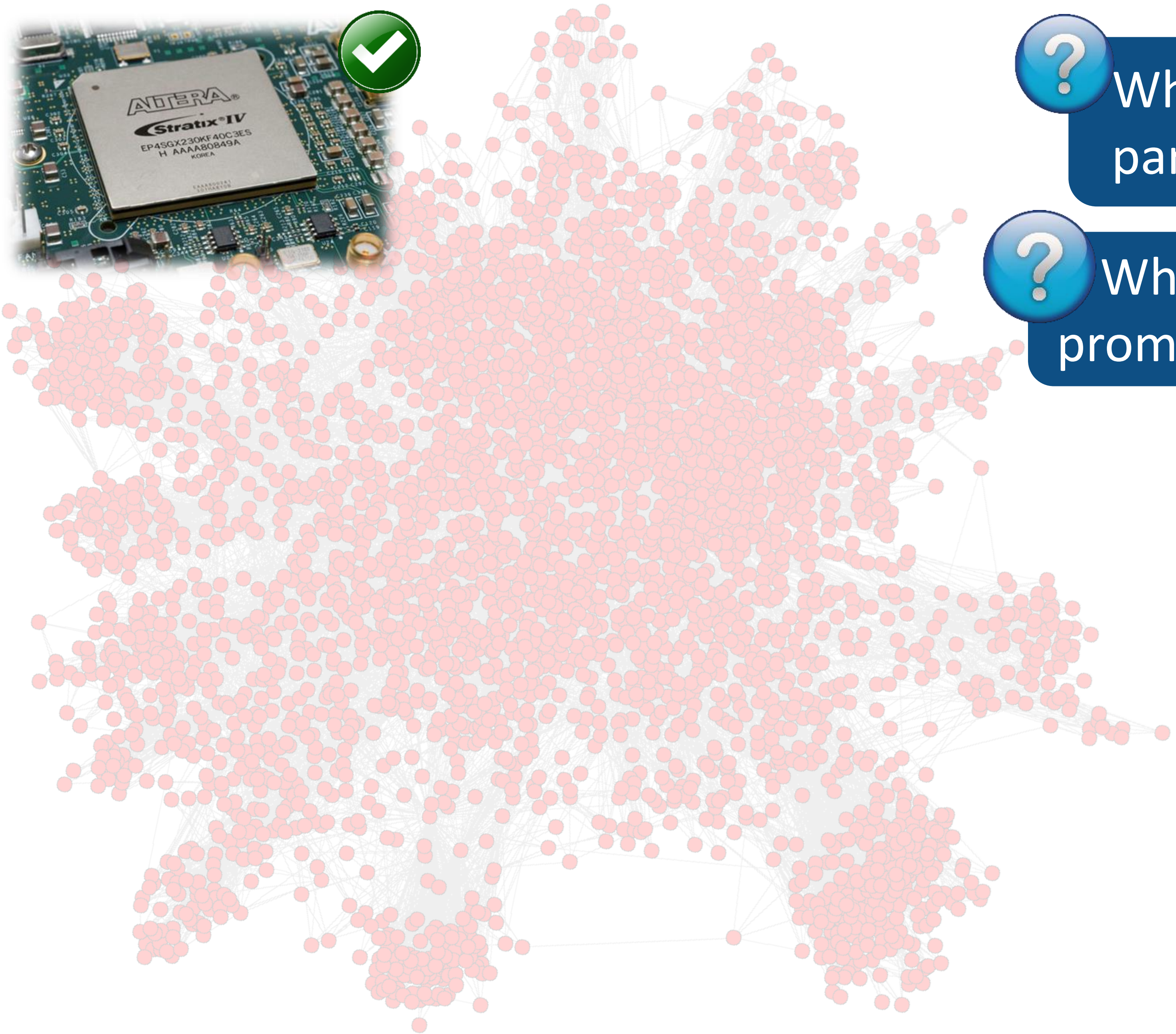
Track: Graph Processing devroom

Room: AW1.126


ALTERA
Stratix IV
EP4SGX230KF40C3ES
H AAAA80849A
KOREA



What programming paradigm and why?



What programming paradigm and why?

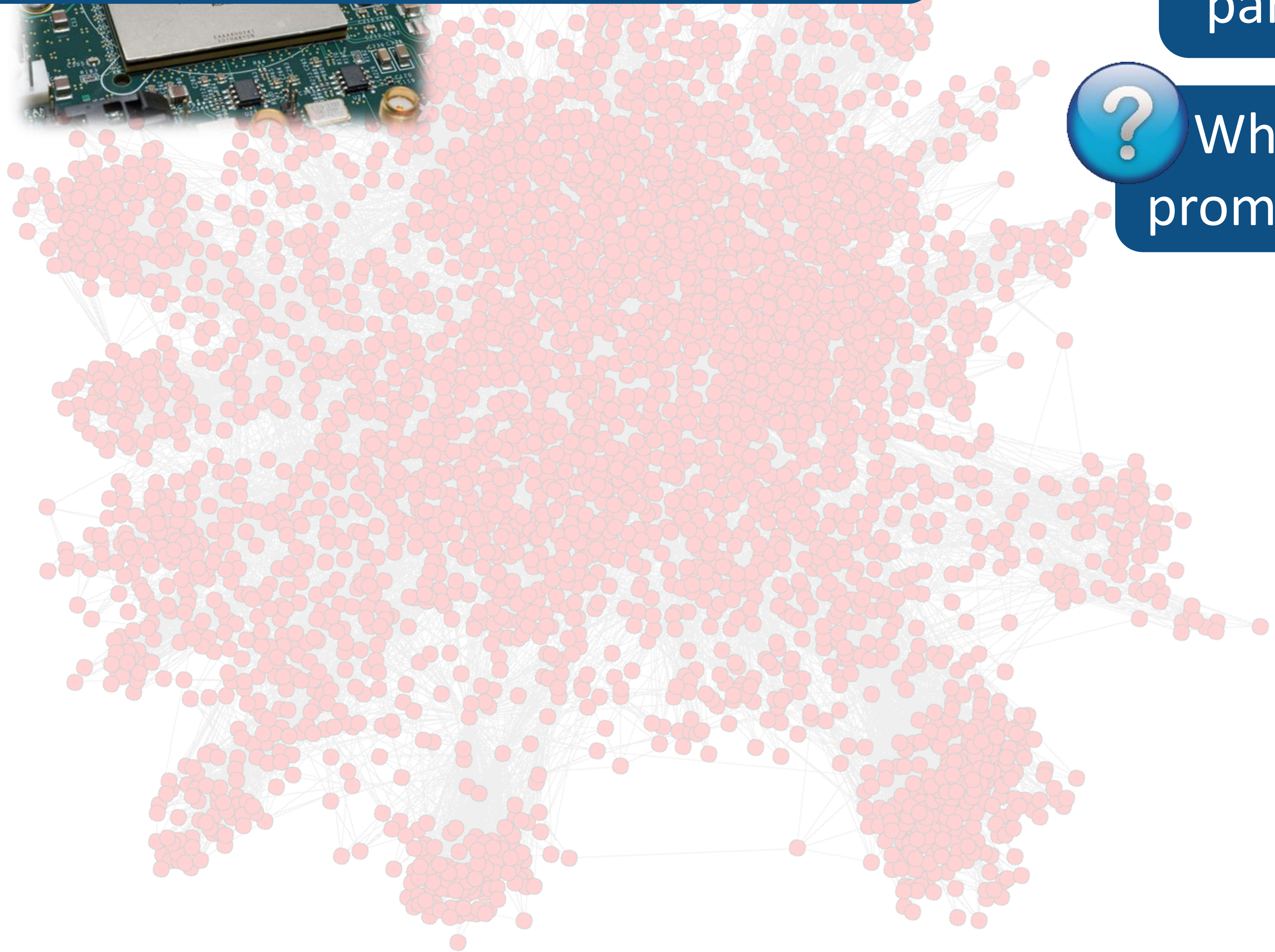
What are the most promising techniques?

What programming paradigm and why?

What are the most promising techniques?

<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs



What programming paradigm and why?

What are the most promising techniques?

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

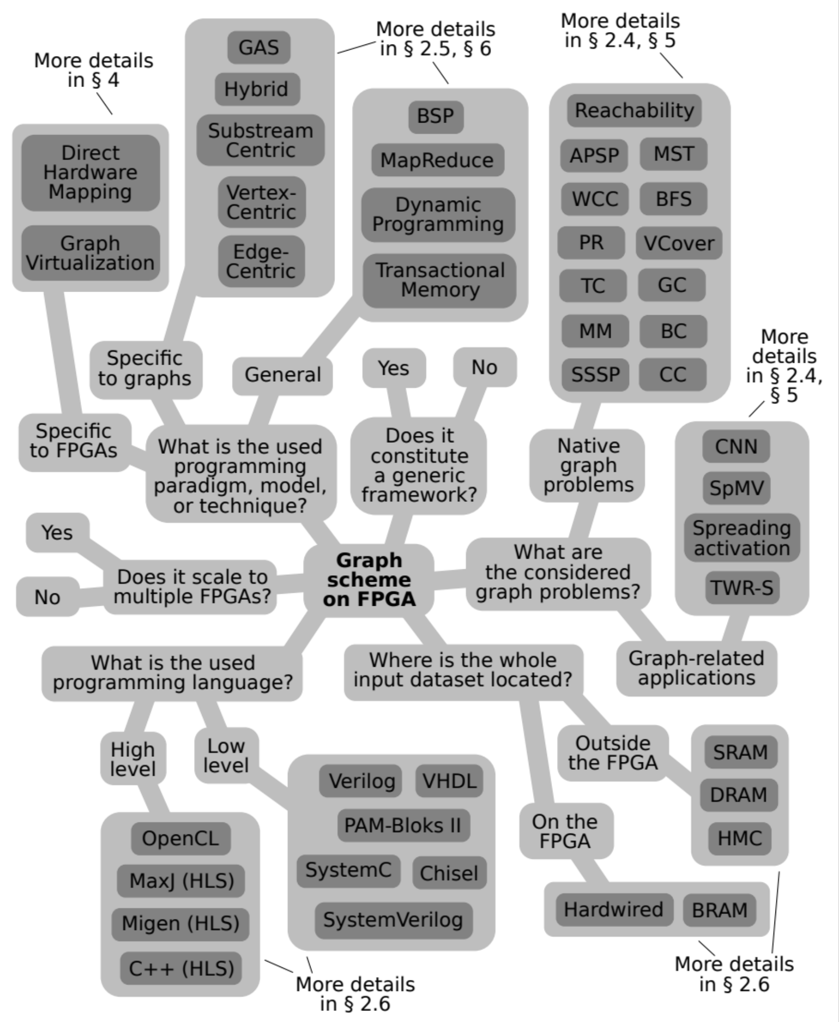


Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

What programming paradigm and why?

What are the most promising techniques?

7 paradigms

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

What programming paradigm and why?

What are the most promising techniques?

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^\dagger$	$m^\dagger$
Kapre [71] **(GraphStep)**	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] **(GraphGen)**	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre [70] **(GraphSoC)**	ASAP'15	👍	SpMV	Vertex-Centric, BSP	C++ (HLS)	👍	BRAM	17k	126k
Dai [40] **(FPGP)**	FPGA'16	👍	BFS	None	unsp.	👍	DRAM	41.6M	1.4B
Oguntebi [93] **(GraphOps)**	FPGA'16	👍	BFS, SpMV, PR, Vertex Cover	None	MaxJ (HLS)	👎	BRAM	16M	128M
Zhou [134]	FCCM'16	👍	SSSP, WCC, MST	Edge-Centric	unsp.	👎	DRAM	4.7M	65.8M
Engelhardt [49] **(GraVF)**	FPL'16	👍	BFS, PR, SSSP, CC	Vertex-Centric	Migen (HLS)	👎	BRAM	128k	512k
Dai [41] **(ForeGraph)**	FPGA'17	👍	PR, BFS, WCC	None	unsp.	👍	DRAM	41.6M	1.4B
Zhou [136]	SBAC-PAD'17	👍	BFS, SSSP	Hybrid (Vertex- and Edge-Centric)	unsp.	👎	DRAM	10M	160M
Ma [85]	FPGA'17	👍	BFS, SSSP, CC, TC, BC	Transactional Memory [16, 59]	System-Verilog	👍	DRAM	24M	58M
Lee [79] **(ExtraV)**	FPGA'17	👍	BFS, PR, CC, AT* [60]	Graph Virtualization	C++ (HLS)	👎	DRAM	124M	1.8B
Zhou [135]	CF'18	👍	SpMV, PR	Edge-Centric, GAS	unsp.	👎	DRAM	41.6M	1.4B
Yang [125]	report (2018)	👍	BFS, PR, WCC	None	OpenCL	👎		4.85M	69M
Yao [127]	report (2018)	👍	BFS, PR, WCC	None	unsp.	👎	BRAM	4.85M	69M

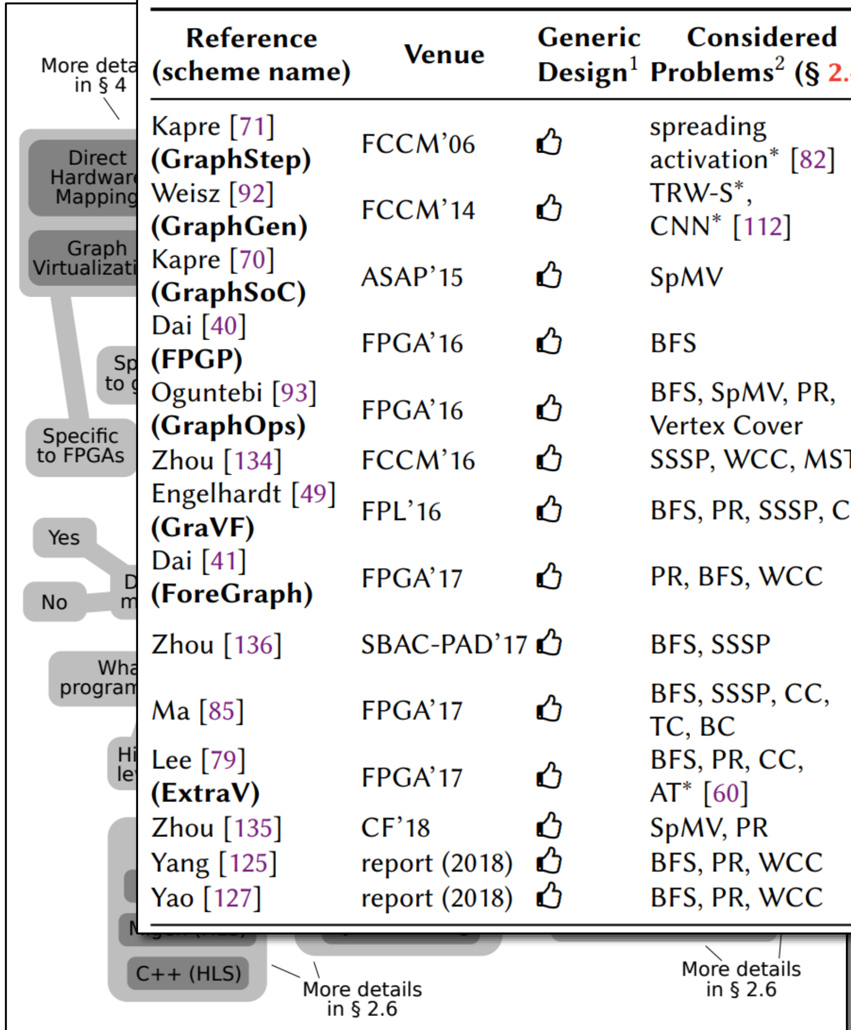


Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

7 paradigms

~15 FPGA graph processing frameworks

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

What programming paradigm and why?

What are the most promising techniques?

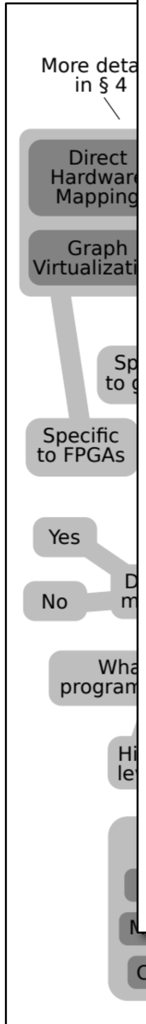


Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^\dagger$	$m^\dagger$
Kapre [71] (GraphStep)	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] (GraphGen)	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k
Kapre (Grap									126k
Dai [4 (FPGI									1.4B
Ogun (Grap									28M
Zhou									.8M
Engel (GraV									512k
Dai [4 (Fore									1.4B
Zhou									60M
Ma [8									58M
Lee [7 (Extra									1.8B
Zhou									1.4B
Yang [									69M
Yao [1									69M

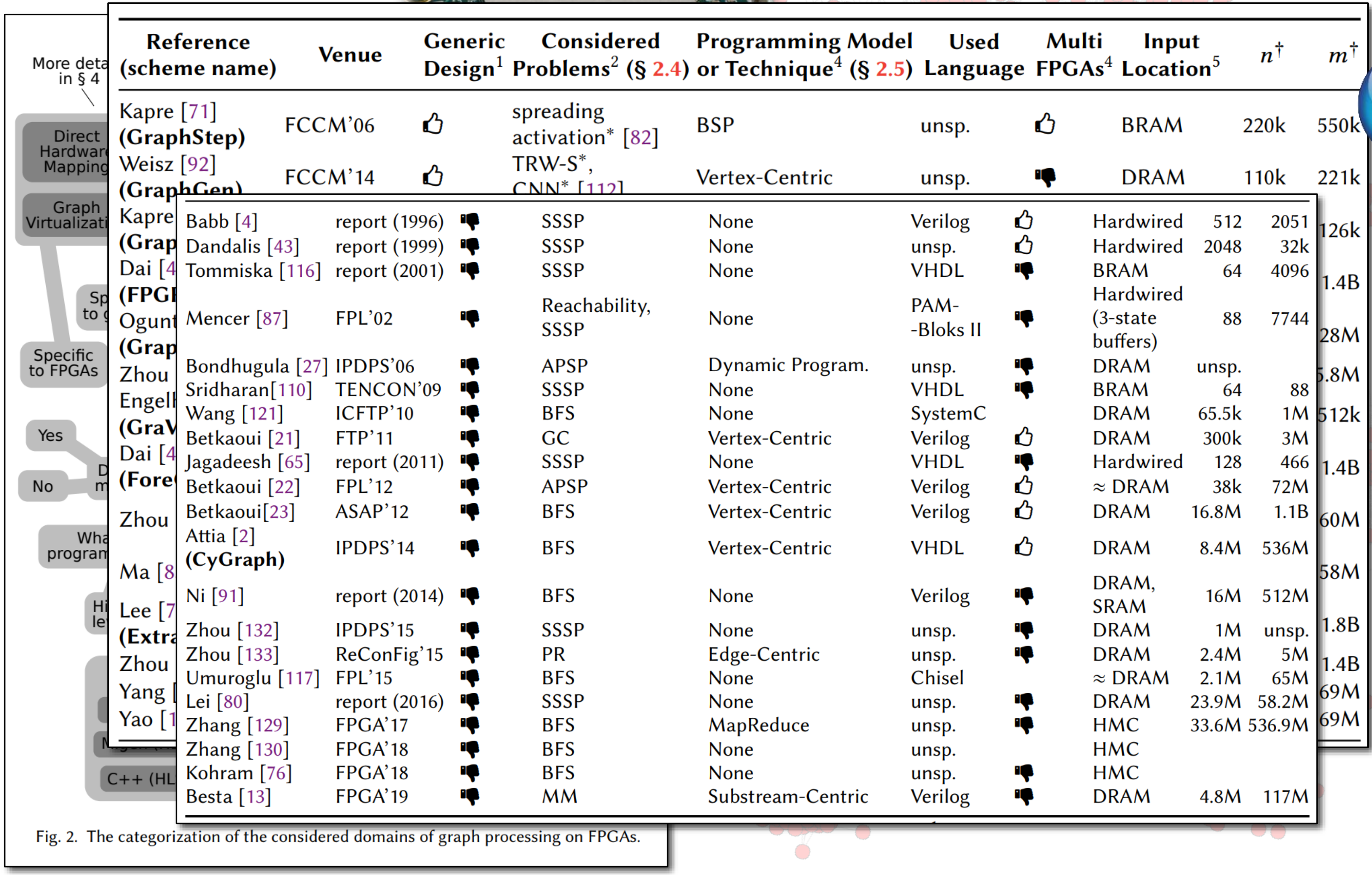
Babb [4]	report (1996)	👎	SSSP	None	Verilog	👍	Hardwired	512	2051
Dandalis [43]	report (1999)	👎	SSSP	None	unsp.	👍	Hardwired	2048	32k
Tommiska [116]	report (2001)	👎	SSSP	None	VHDL	👎	BRAM	64	4096
Mencer [87]	FPL'02	👎	Reachability, SSSP	None	PAM--Bloks II	👎	Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] (CyGraph)	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs



Reference (scheme name)	Venue	Generic Design[1]	Considered Problems[2] (§ 2.4)	Programming Model or Technique[4] (§ 2.5)	Used Language	Multi FPGAs[4]	Input Location[5]	$n^{\dagger}$	$m^{\dagger}$
Kapre [71] (GraphStep)	FCCM'06	👍	spreading activation* [82]	BSP	unsp.	👍	BRAM	220k	550k
Weisz [92] (GraphGen)	FCCM'14	👍	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	👎	DRAM	110k	221k

Babb [4]	report (1996)	👎	SSSP	None	Verilog	👍	Hardwired	512	2051
Dandalis [43]	report (1999)	👎	SSSP	None	unsp.	👍	Hardwired	2048	32k
Tommiska [116]	report (2001)	👎	SSSP	None	VHDL	👎	BRAM	64	4096
Mencer [87]	FPL'02	👎	Reachability, SSSP	None	PAM--Bloks II	👎	Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] (CyGraph)	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

Selected MWM-related parts are in the FPGA paper, the rest is in…

Reference (scheme name)	Venue	Generic Design[1]							$m^\dagger$
Babb [4]	report (1996)	👎	SSSP	None	Verilog	👍	Hardwired	512	2051
Dandalis [43]	report (1999)	👎	SSSP	None	unsp.	👍	Hardwired	2048	32k
Tommiska [116]	report (2001)	👎	SSSP	None	VHDL	👎	BRAM	64	4096
Mencer [87]	FPL'02	👎	Reachability, SSSP	None	PAM--Bloks II	👎	Hardwired (3-state buffers)	88	7744
Bondhugula [27]	IPDPS'06	👎	APSP	Dynamic Program.	unsp.	👎	DRAM	unsp.	
Sridharan[110]	TENCON'09	👎	SSSP	None	VHDL	👎	BRAM	64	88
Wang [121]	ICFTP'10	👎	BFS	None	SystemC		DRAM	65.5k	1M
Betkaoui [21]	FTP'11	👎	GC	Vertex-Centric	Verilog	👍	DRAM	300k	3M
Jagadeesh [65]	report (2011)	👎	SSSP	None	VHDL	👎	Hardwired	128	466
Betkaoui [22]	FPL'12	👎	APSP	Vertex-Centric	Verilog	👍	≈ DRAM	38k	72M
Betkaoui[23]	ASAP'12	👎	BFS	Vertex-Centric	Verilog	👍	DRAM	16.8M	1.1B
Attia [2] (CyGraph)	IPDPS'14	👎	BFS	Vertex-Centric	VHDL	👍	DRAM	8.4M	536M
Ni [91]	report (2014)	👎	BFS	None	Verilog	👎	DRAM, SRAM	16M	512M
Zhou [132]	IPDPS'15	👎	SSSP	None	unsp.	👎	DRAM	1M	unsp.
Zhou [133]	ReConFig'15	👎	PR	Edge-Centric	unsp.	👎	DRAM	2.4M	5M
Umuroglu [117]	FPL'15	👎	BFS	None	Chisel		≈ DRAM	2.1M	65M
Lei [80]	report (2016)	👎	SSSP	None	unsp.	👎	DRAM	23.9M	58.2M
Zhang [129]	FPGA'17	👎	BFS	MapReduce	unsp.	👎	HMC	33.6M	536.9M
Zhang [130]	FPGA'18	👎	BFS	None	unsp.		HMC		
Kohram [76]	FPGA'18	👎	BFS	None	unsp.	👎	HMC		
Besta [13]	FPGA'19	👎	MM	Substream-Centric	Verilog	👎	DRAM	4.8M	117M

Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, …

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

Selected MWM-related parts are in the FPGA paper, the rest is in...

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

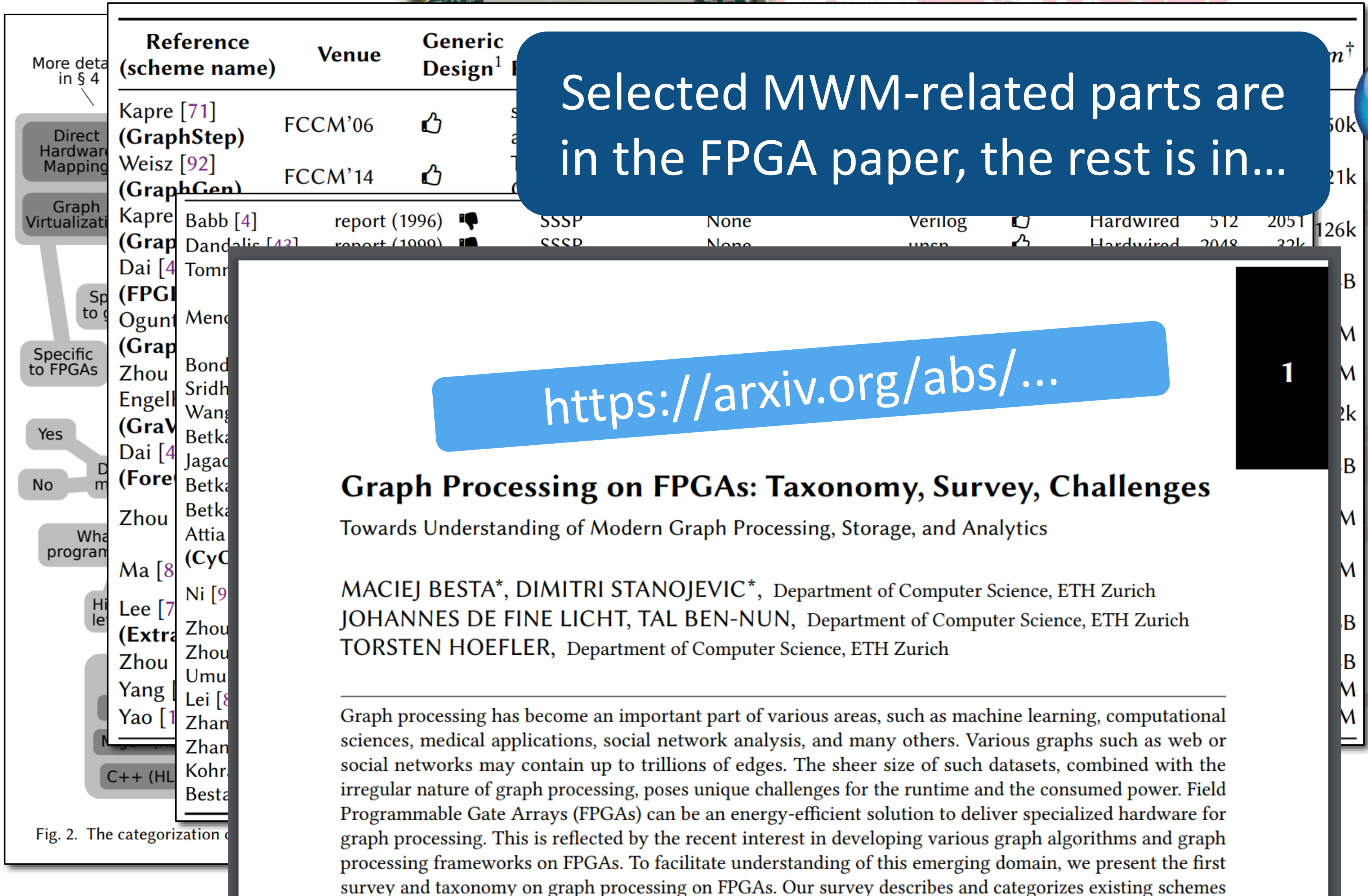
Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for graph processing. This is reflected by the recent interest in developing various graph algorithms and graph processing frameworks on FPGAs. To facilitate understanding of this emerging domain, we present the first survey and taxonomy on graph processing on FPGAs. Our survey describes and categorizes existing schemes

Fig. 2. The categorization

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

What programming paradigm and why?

What are the most promising techniques?



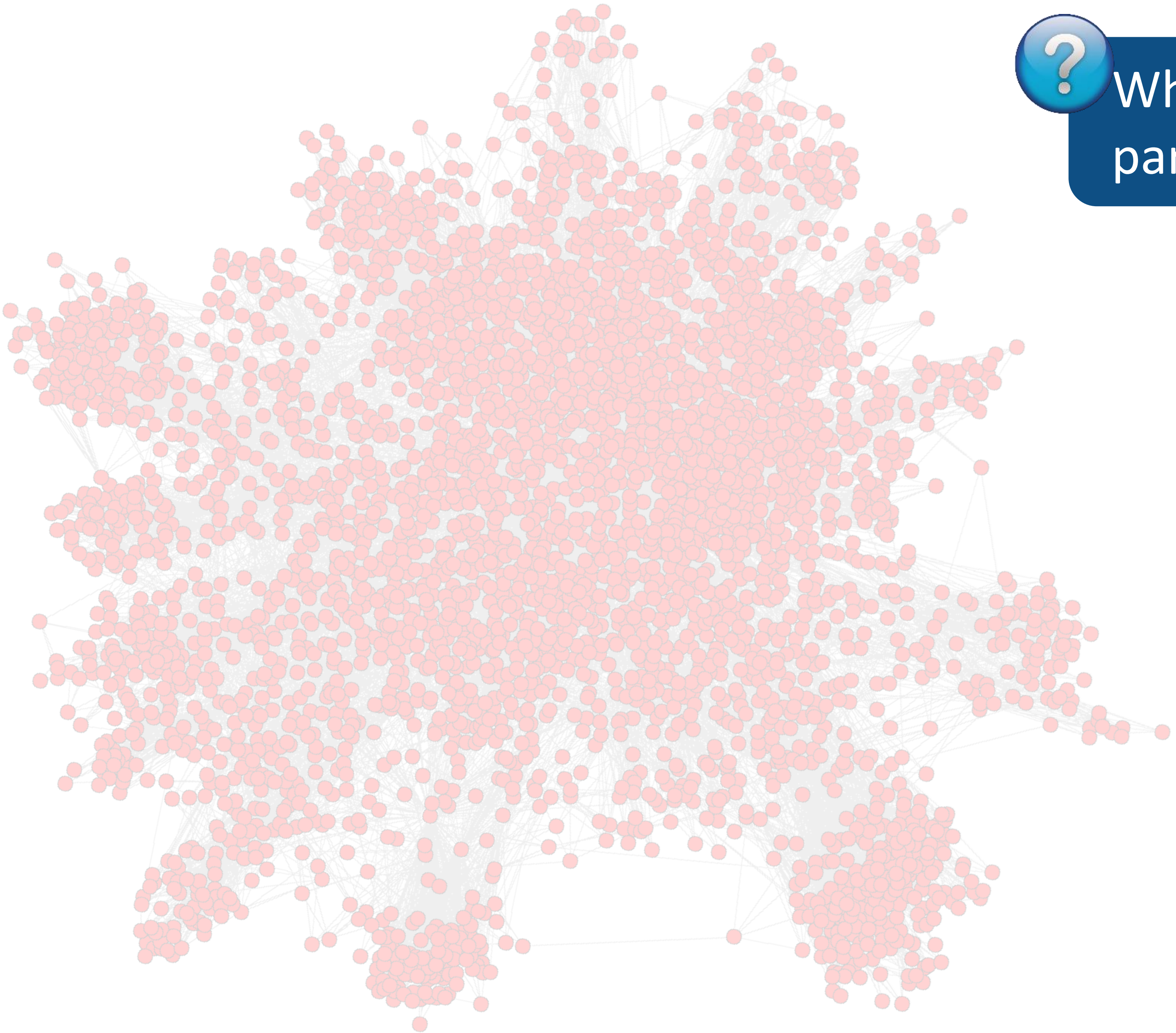
Selected MWM-related parts are in the FPGA paper, the rest is in...

Key techniques, challenges, features, ...

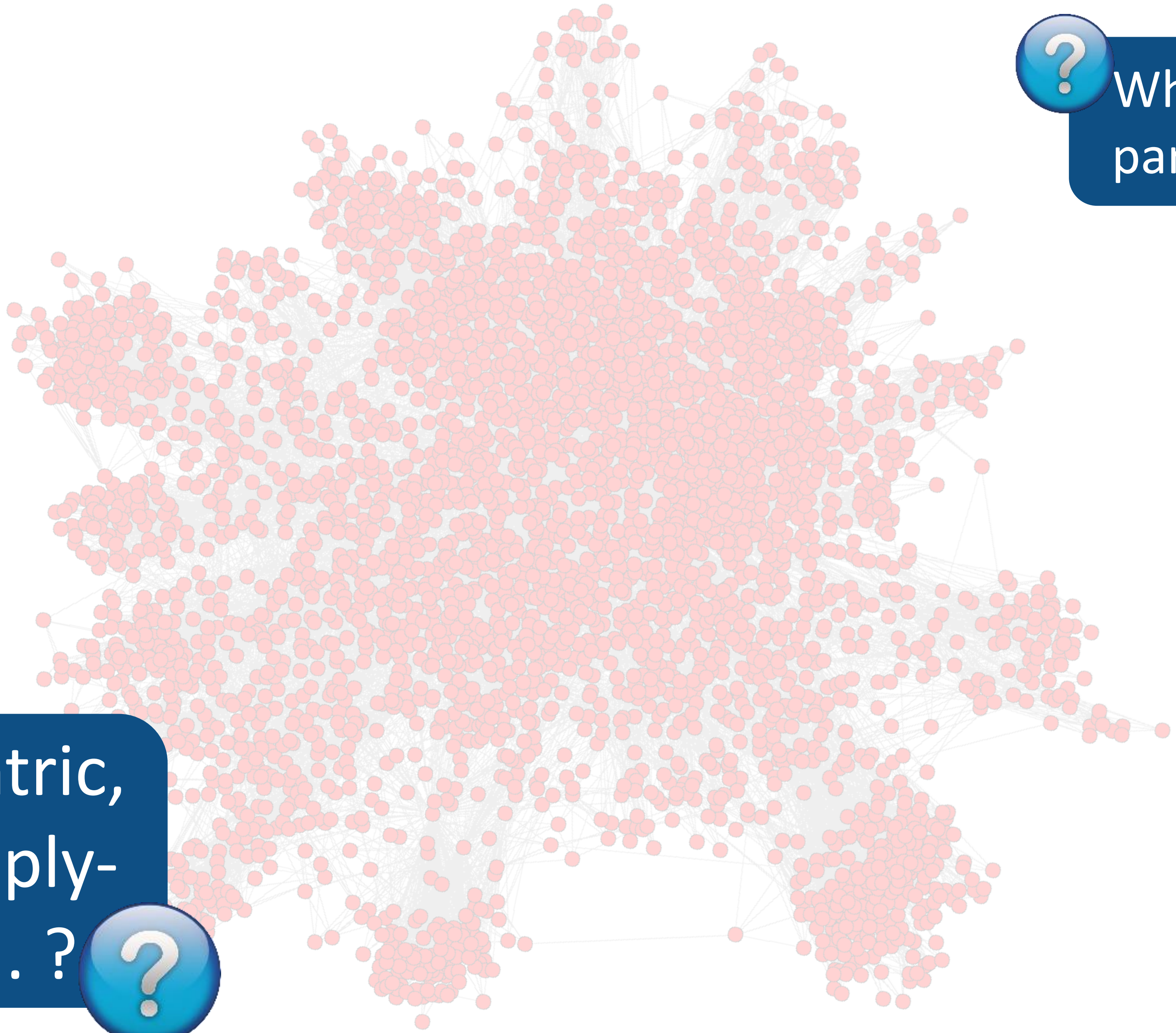
7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

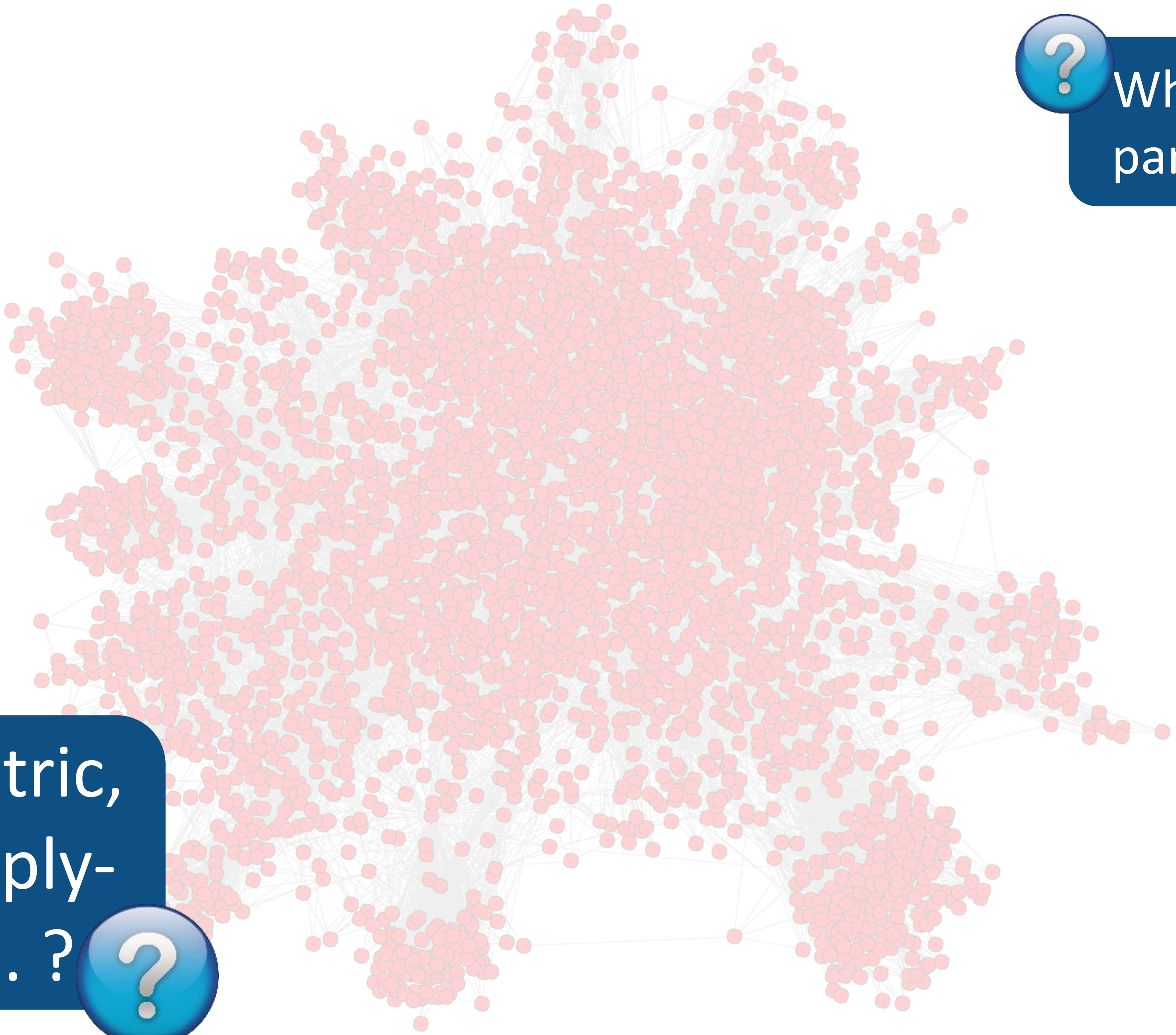


What programming paradigm and why?

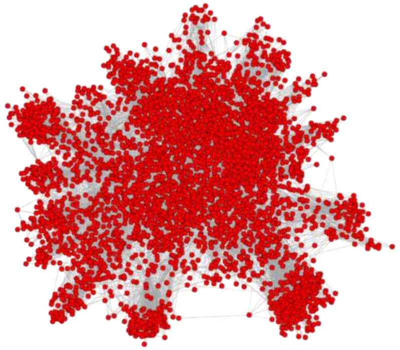


What programming paradigm and why?

Vertex-centric, Gather-Apply-Scatter, ... ?

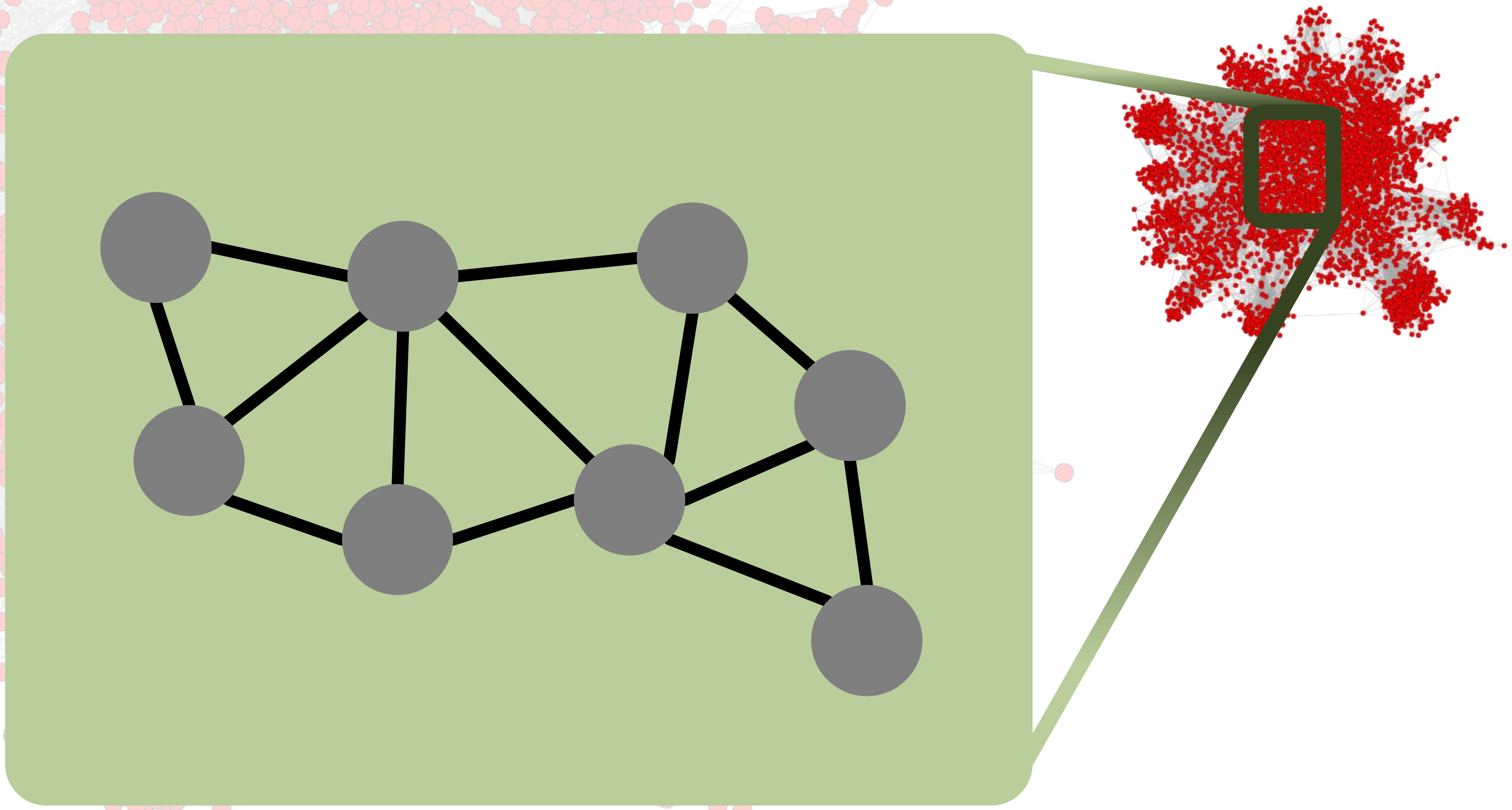


What programming paradigm and why?

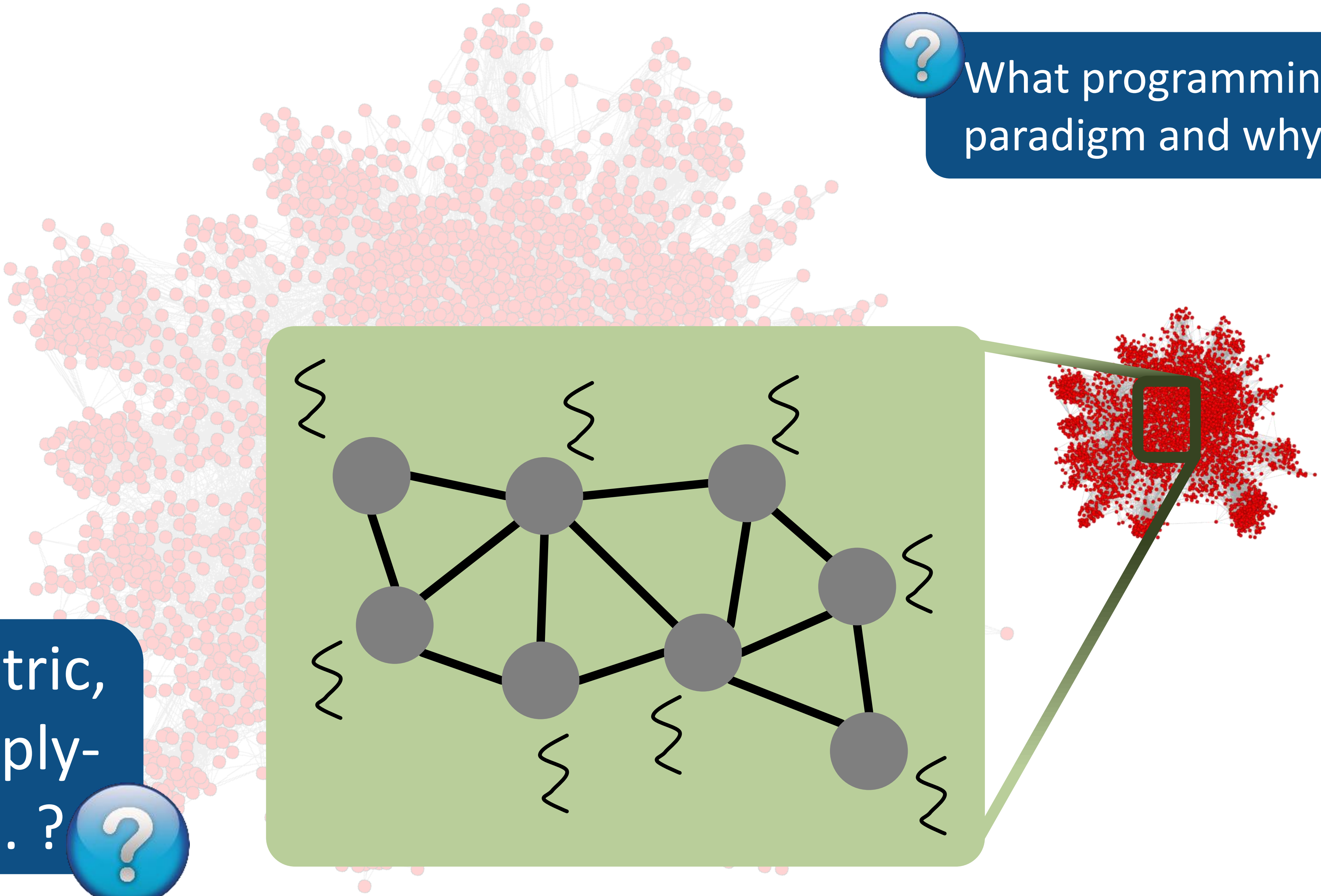


Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



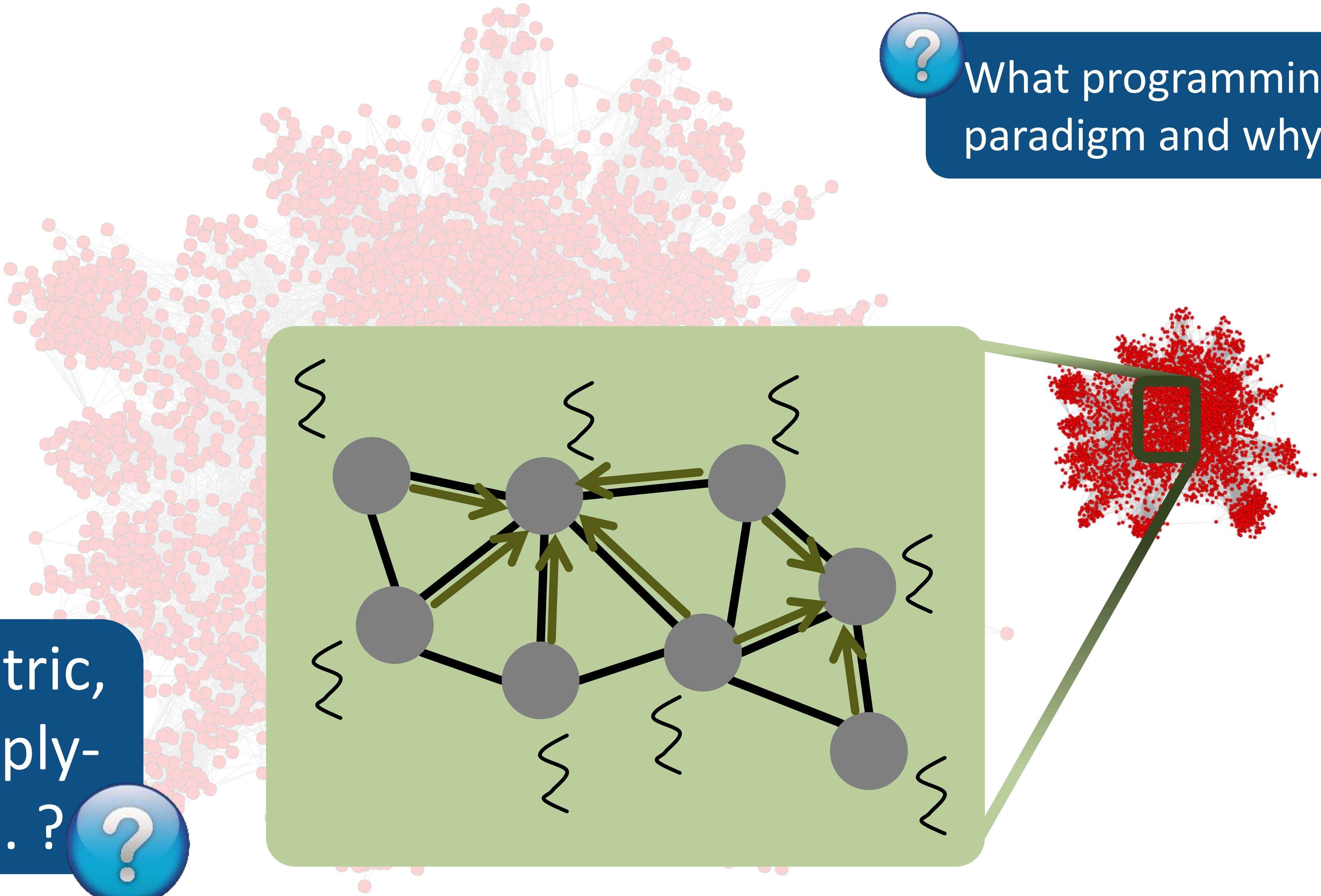
Vertex-centric, Gather-Apply-Scatter, ... ?



What programming paradigm and why?

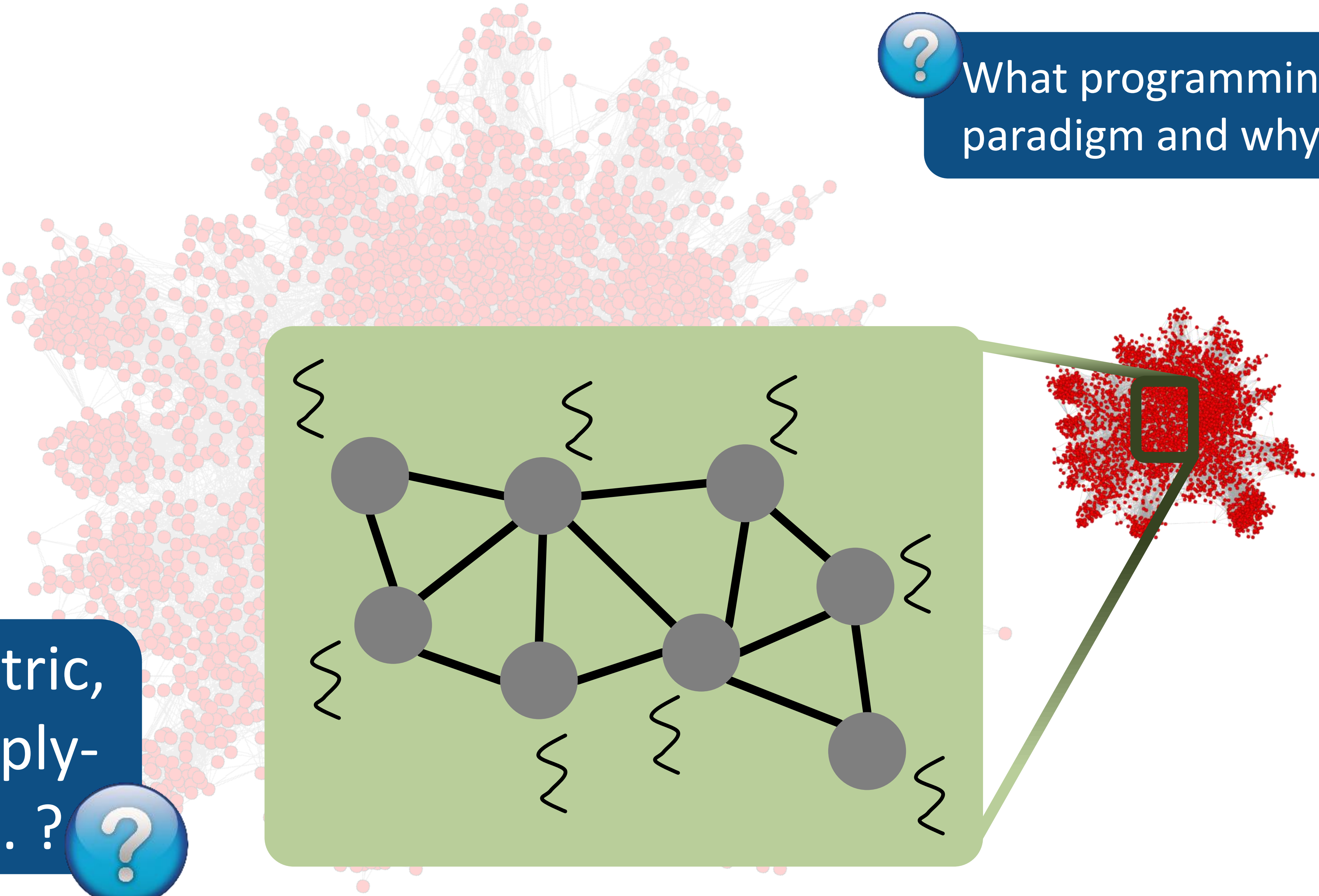
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



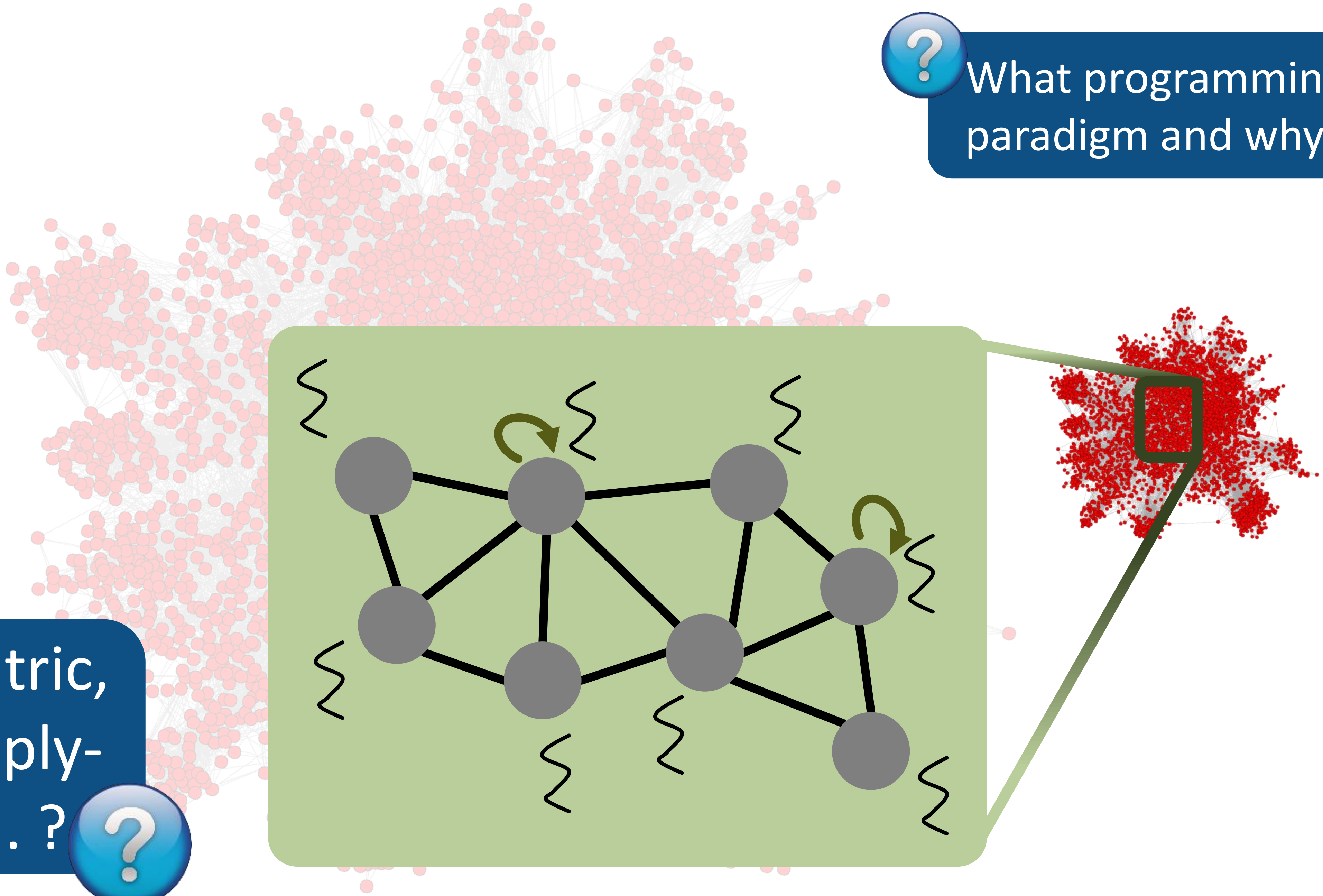
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



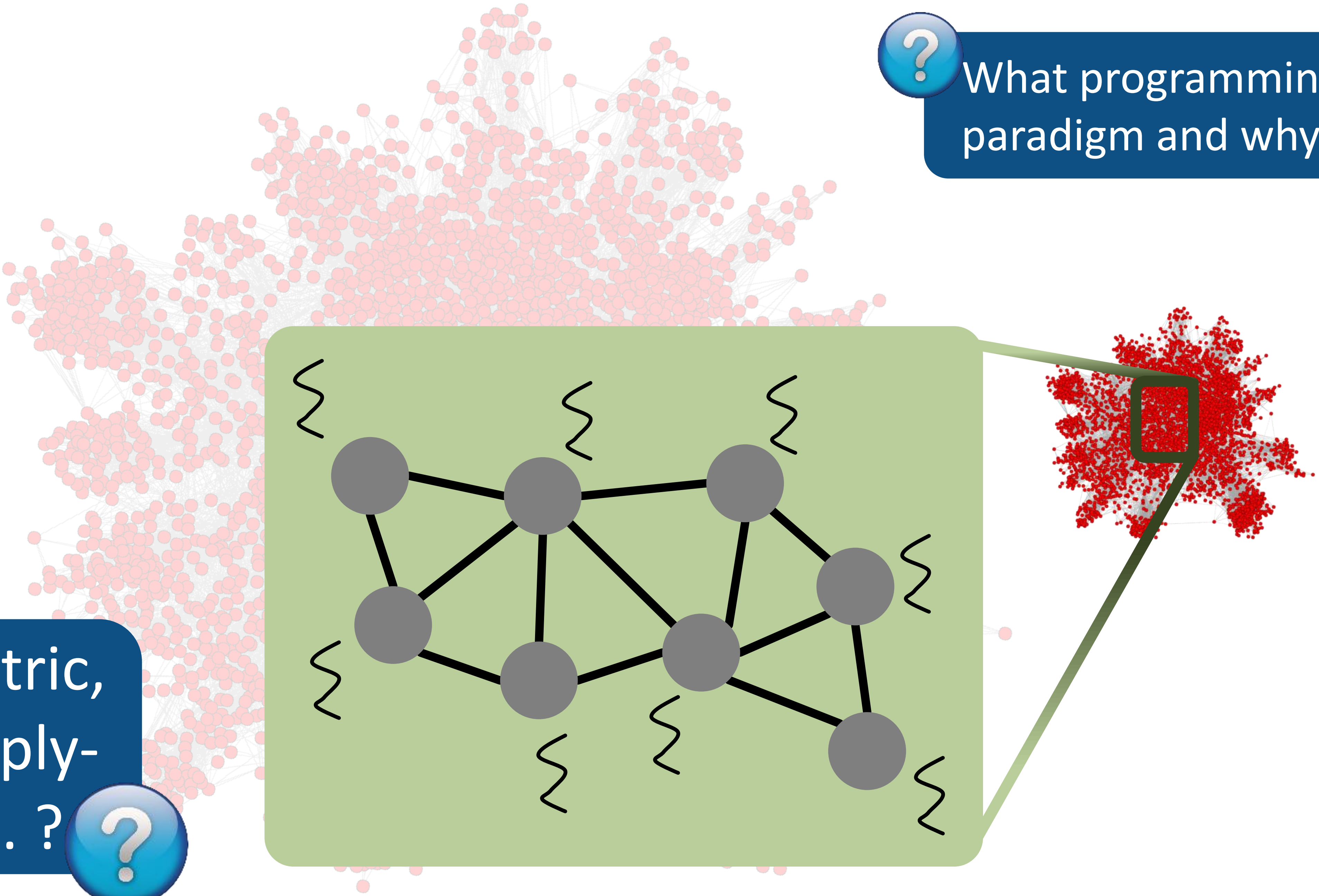
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



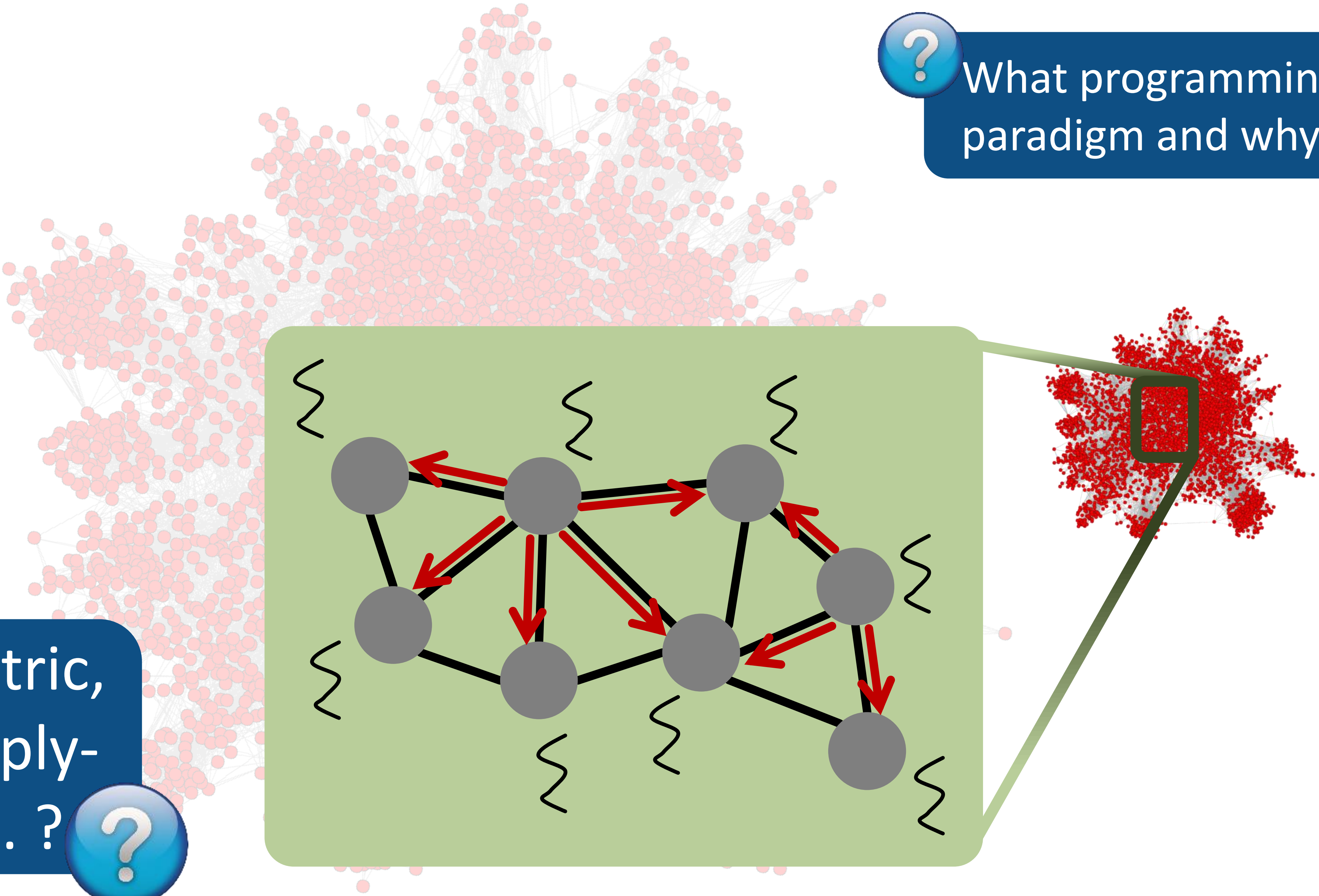
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



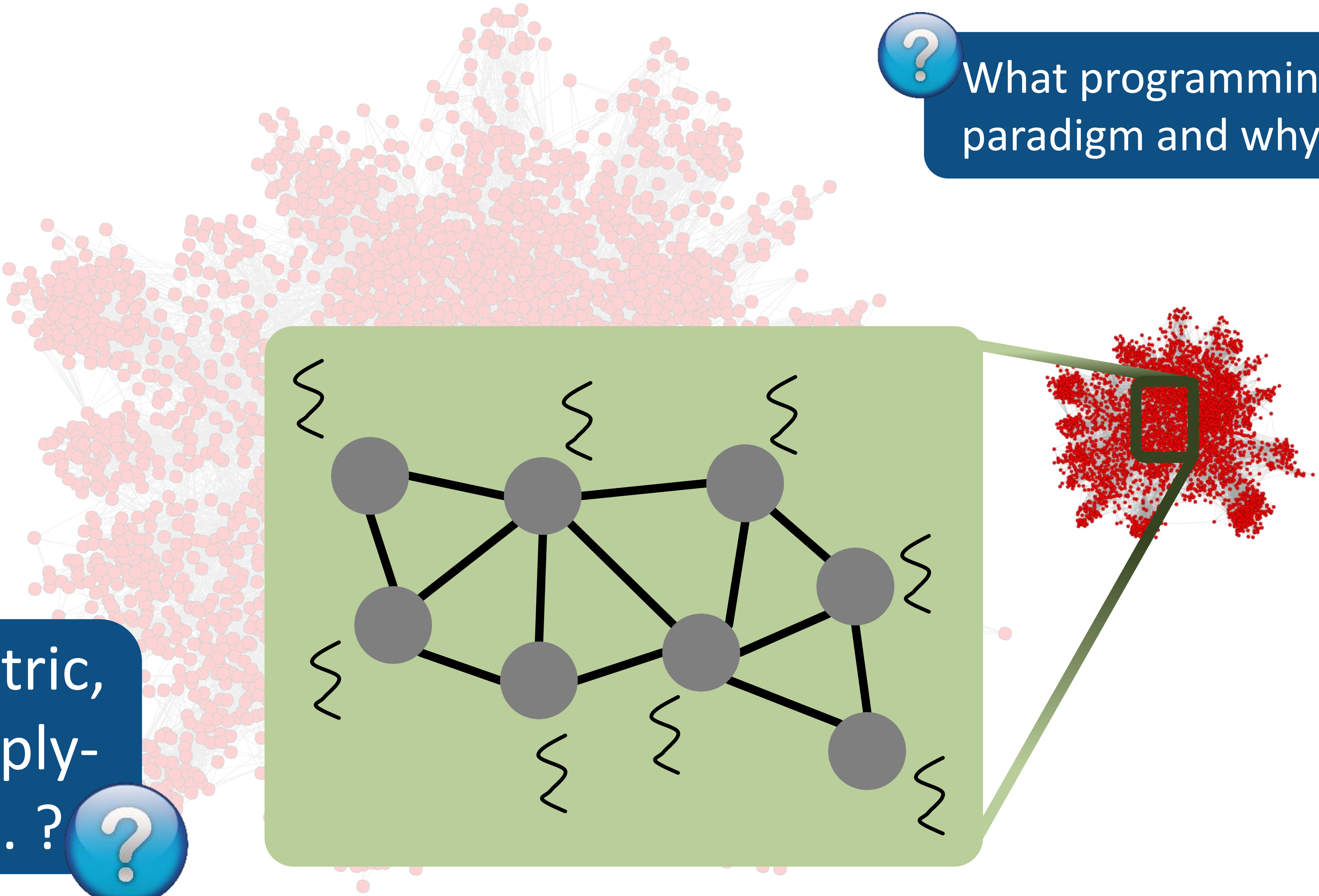
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



Vertex-centric, Gather-Apply-Scatter, ... ?

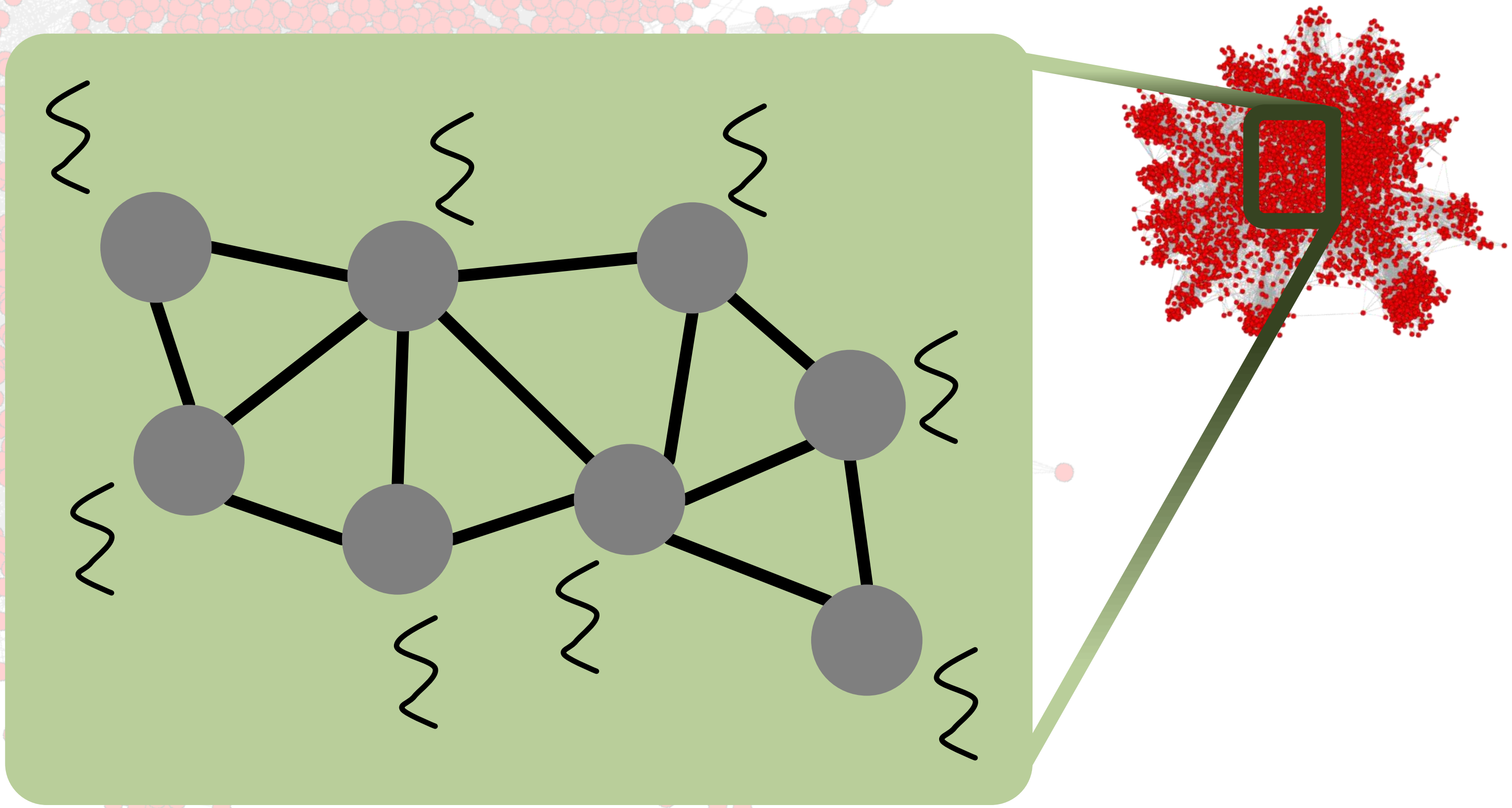
What programming paradigm and why?



Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?

Well...

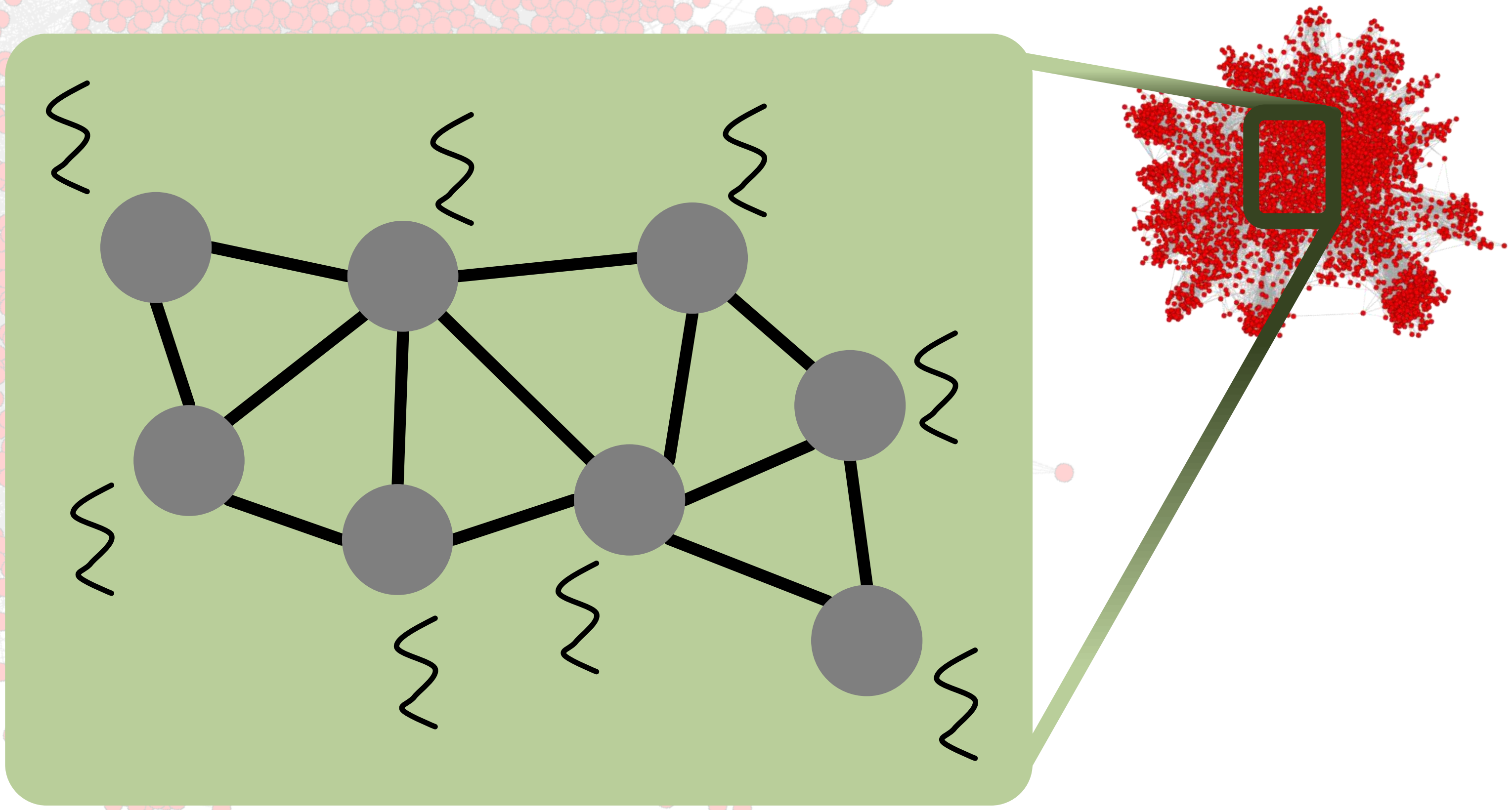


Vertex-centric, Gather-Apply-Scatter, ... ?

It was designed with the "batch" analytics in mind.

Well...

What programming paradigm and why?



Vertex-centric, Gather-Apply-Scatter, ... ?

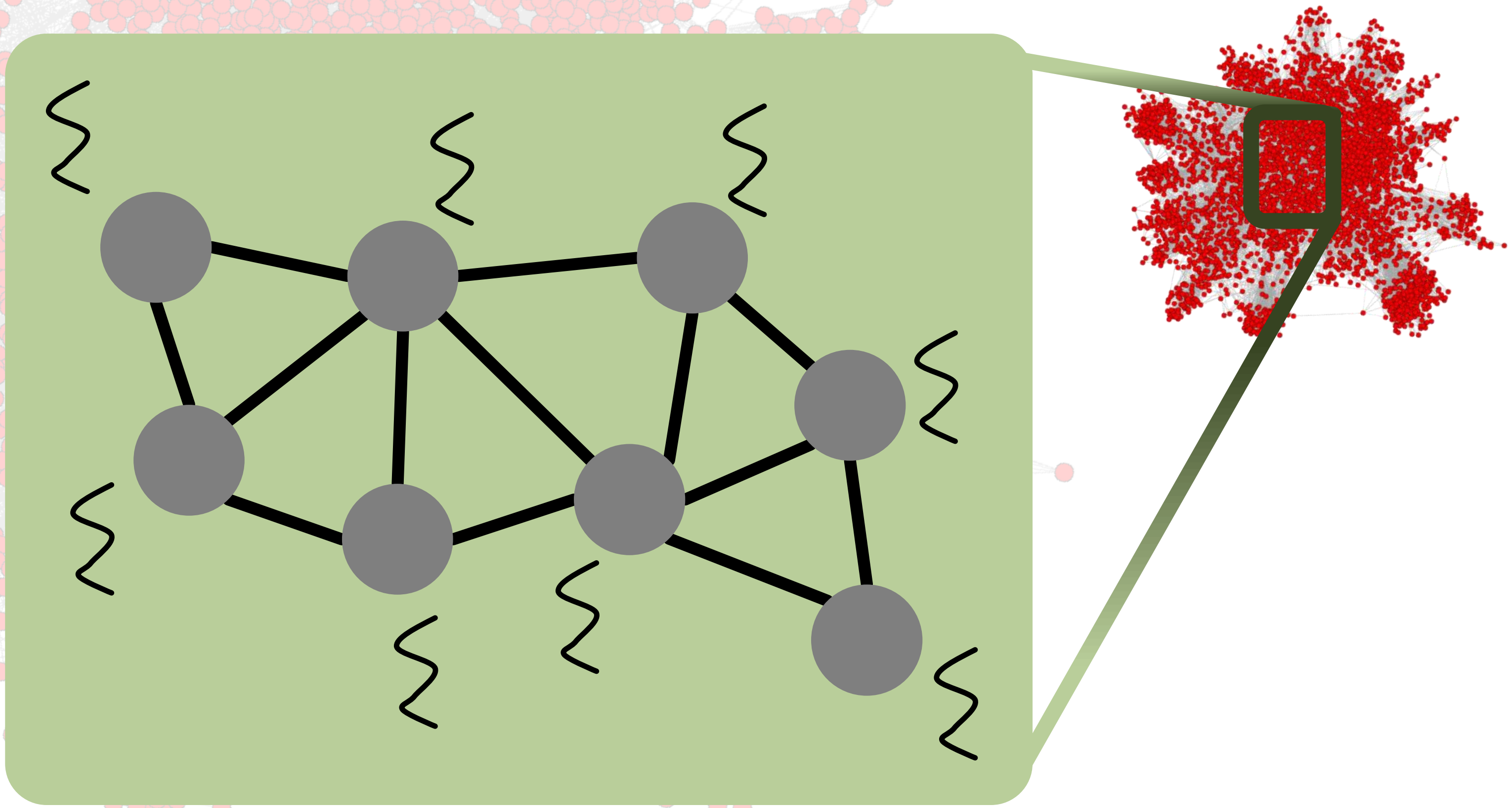
It was designed with the "batch" analytics in mind.

Well...

What programming paradigm and why?

Assumes the whole input graph is accessible...

Vertex-centric, Gather-Apply-Scatter, ... ?



It was designed with the "batch" analytics in mind.

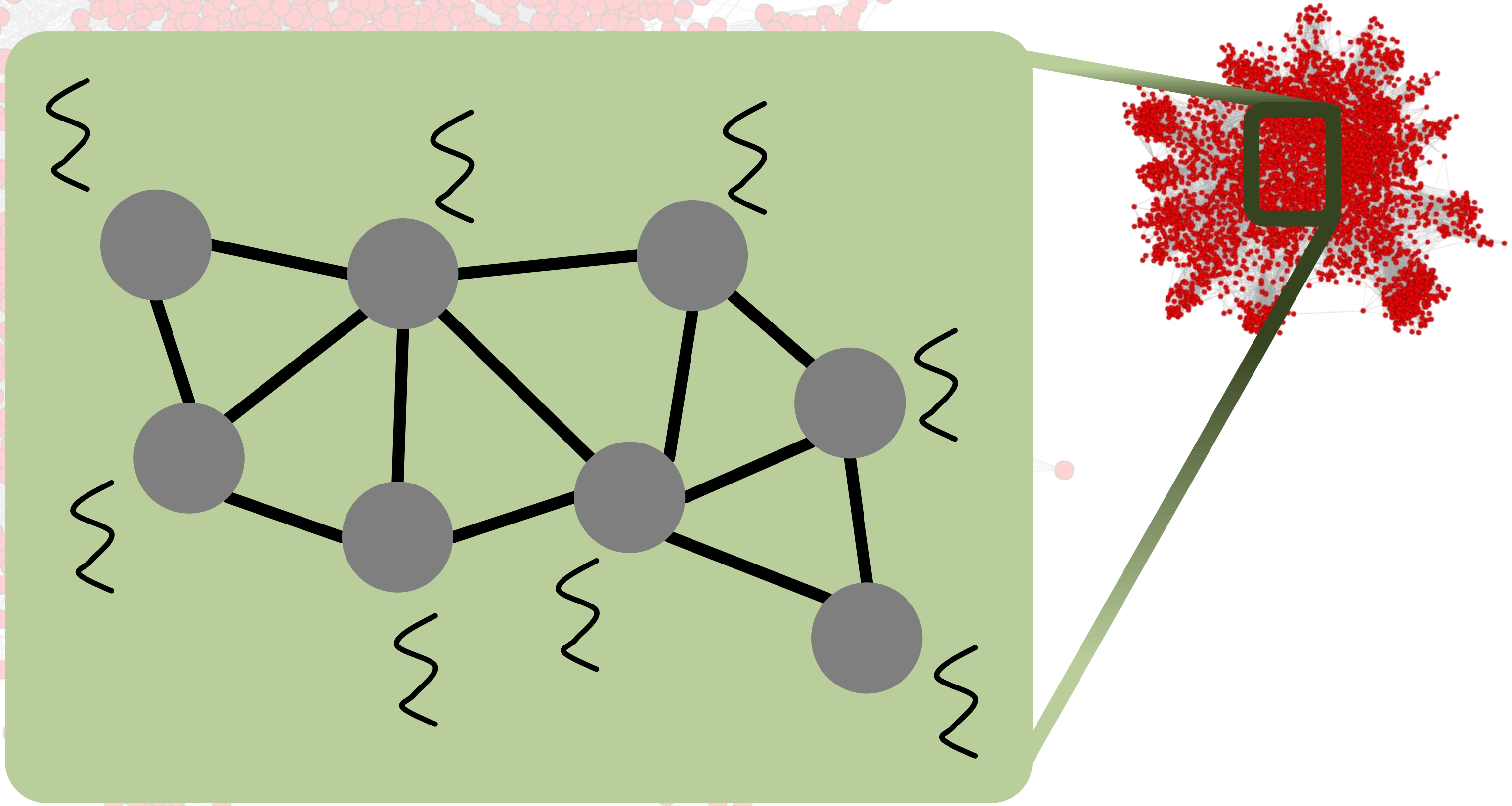
Well…

What programming paradigm and why?

Assumes the whole input graph is accessible…

…when in BRAM, size is severely limited

Vertex-centric, Gather-Apply-Scatter, … ?



It was designed with the "batch" analytics in mind.

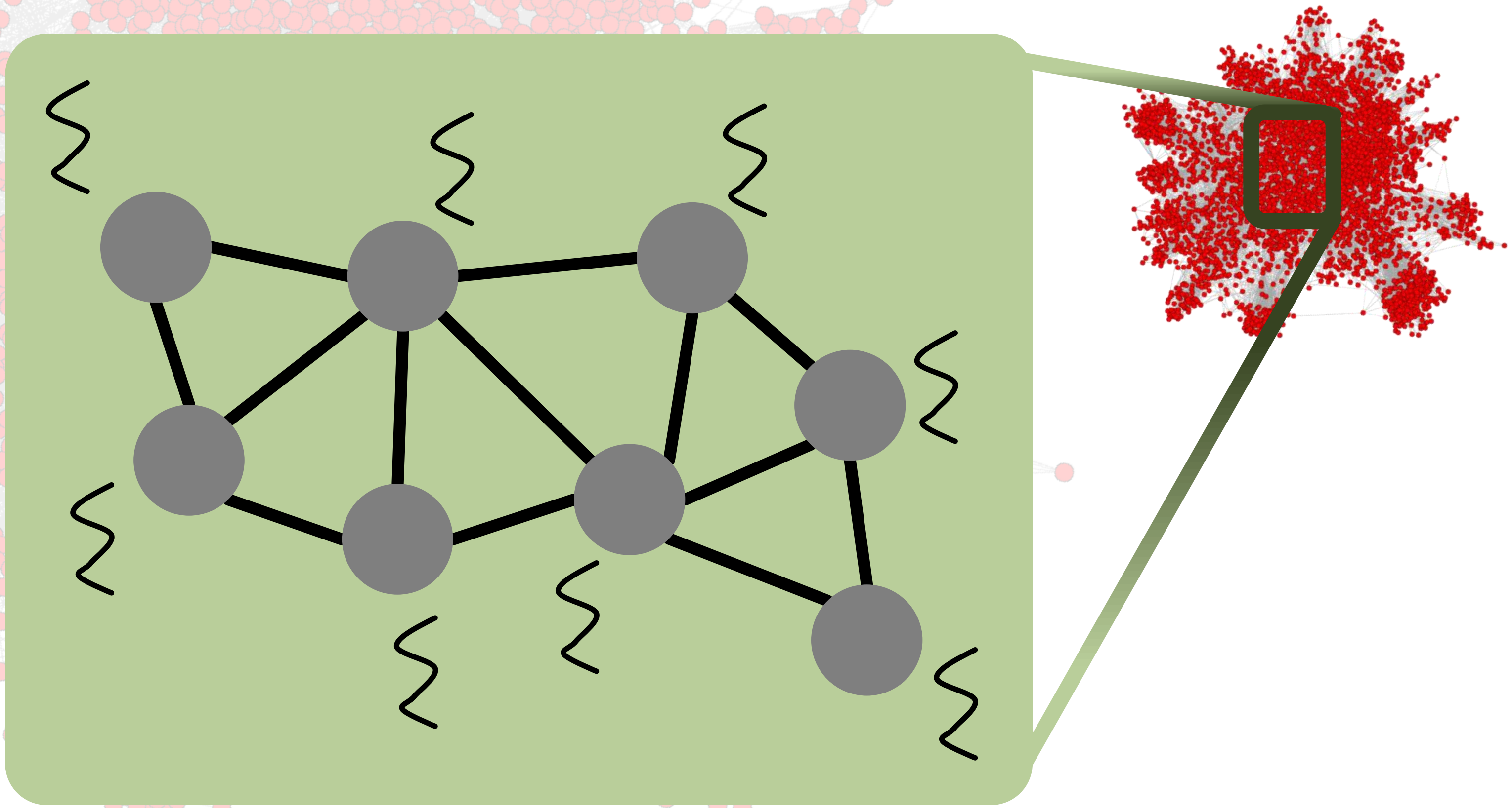
...when in DRAM, accessing & pipelining become complex

What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

Vertex-centric, Gather-Apply-Scatter, ... ?



It was designed with the “batch” analytics in mind.

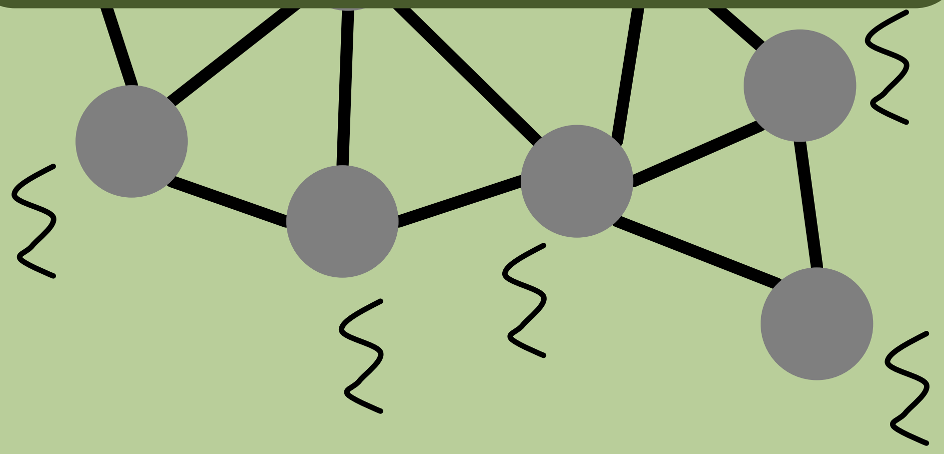
…when in DRAM, accessing & pipelining become complex

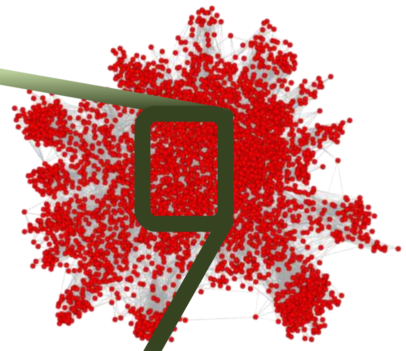
What programming paradigm and why?

Assumes the whole input graph is accessible…

…when in BRAM, size is severely limited

“(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations.” [1]





Vertex-centric, Gather-Apply-Scatter, ... ?

[1] S. Salihoglu and J. Widom, “Optimizing graph algorithms on Pregel-like systems”. VLDB. 2014.

It was designed with the "batch" analytics in mind.

...when in DRAM, accessing & pipelining become complex

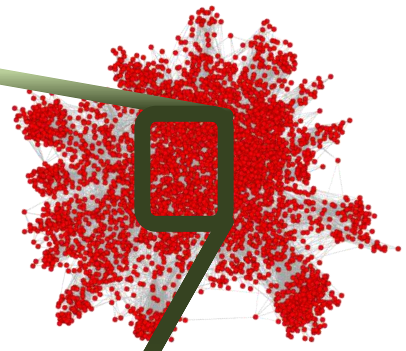
What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

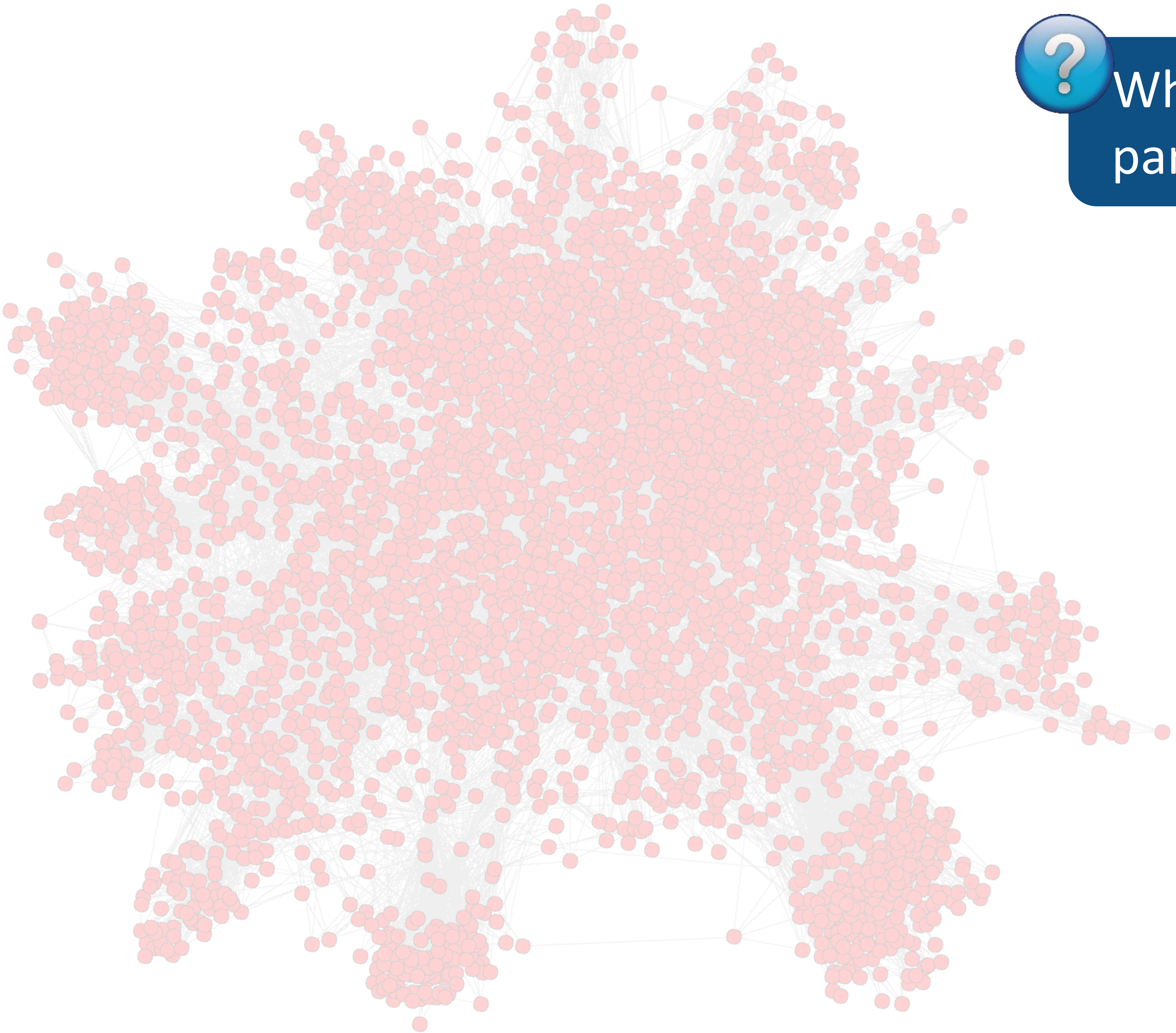
"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

Vertex-Centric (aka Pregel-like) approach is complex for problems such as matchings, spanning trees, etc.

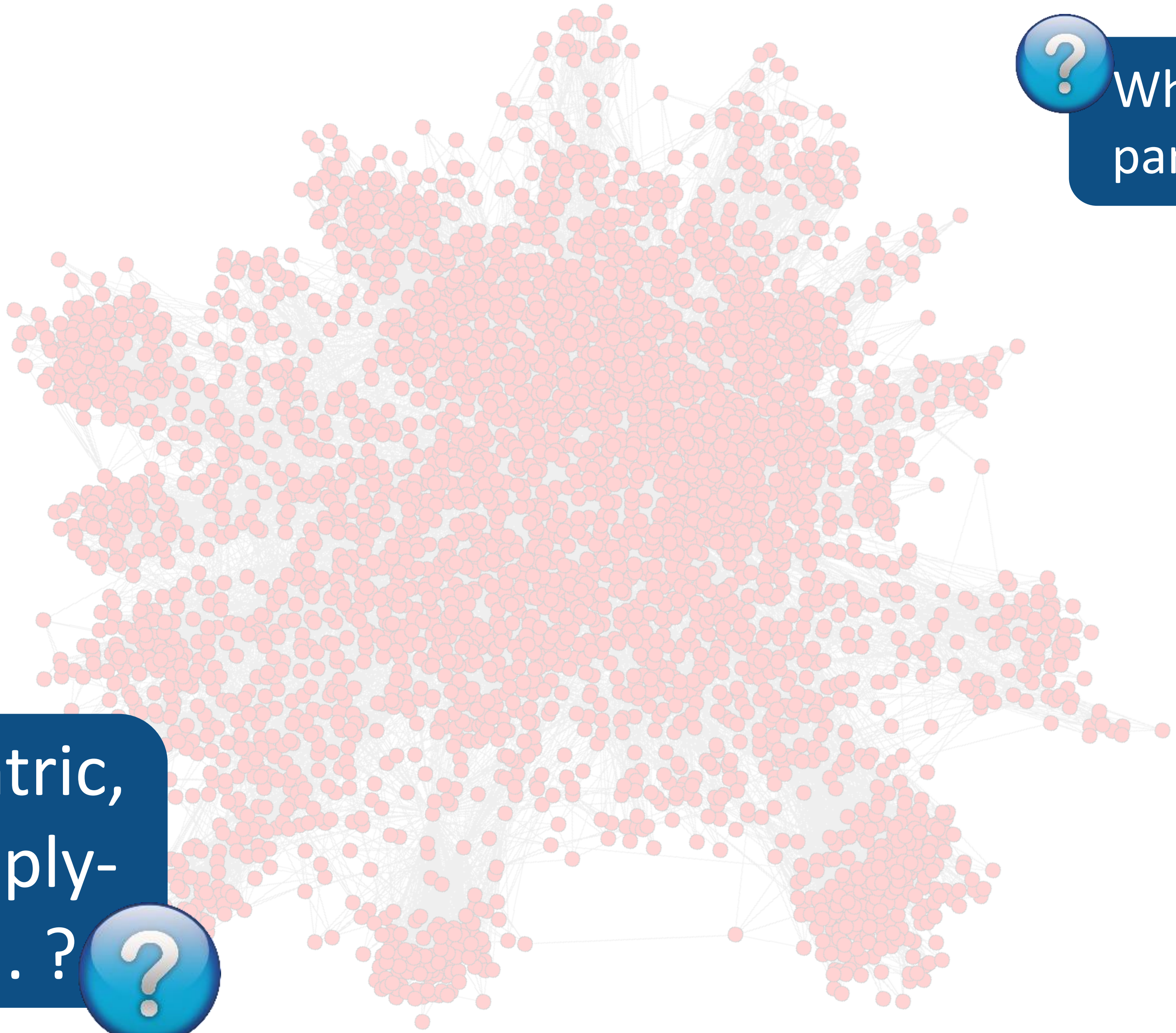


Vertex-centric, Gather-Apply-Scatter, ... ?

[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.



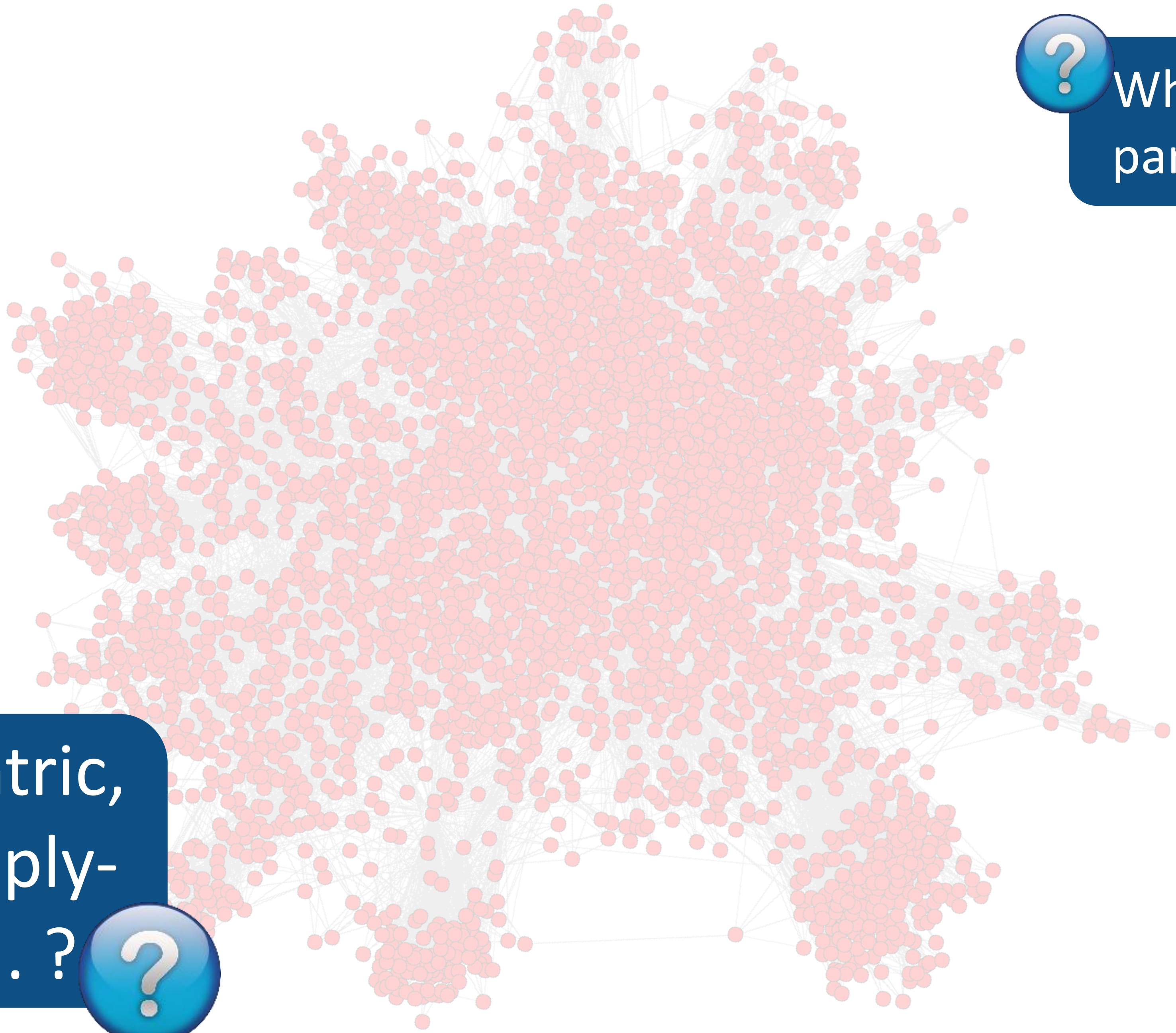
What programming paradigm and why?

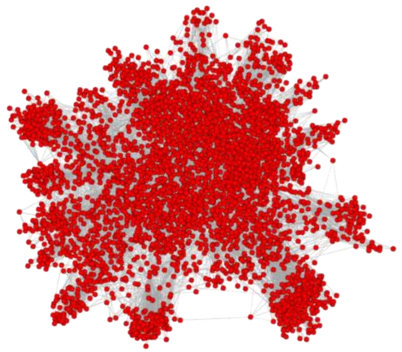


What programming paradigm and why?

Vertex-centric, Gather-Apply-Scatter, ... ?

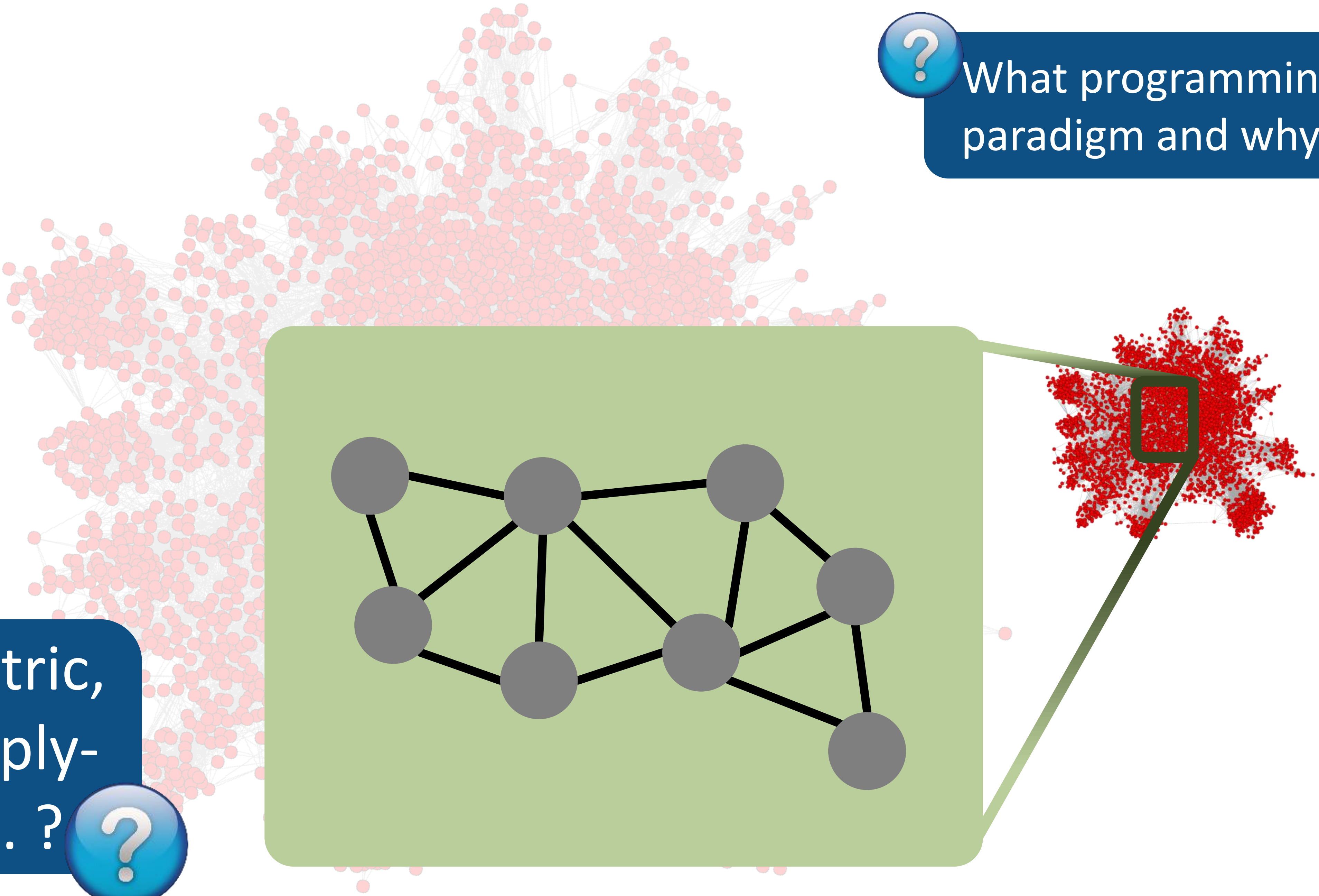
What programming paradigm and why?



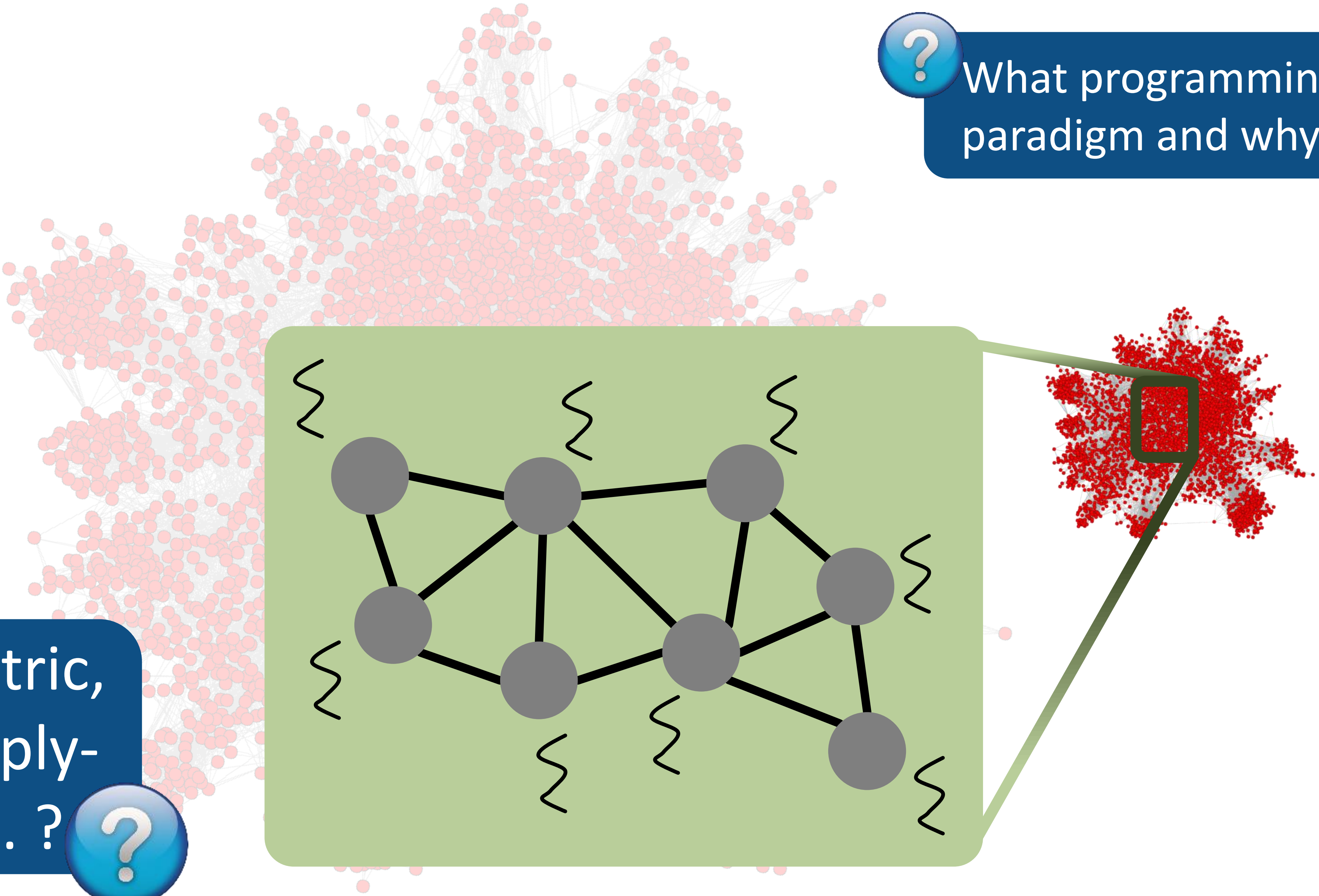


Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



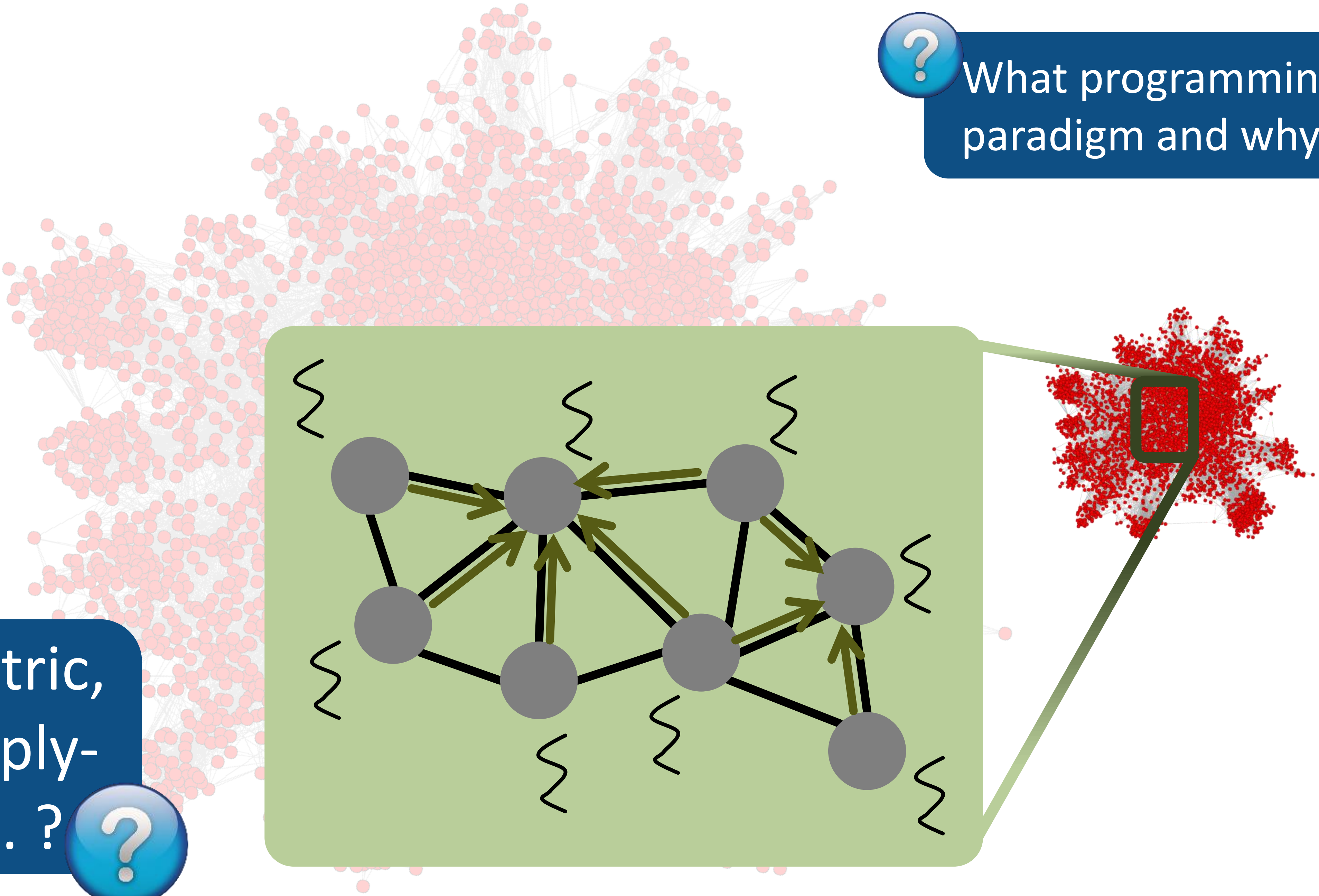
Vertex-centric, Gather-Apply-Scatter, ... ?



What programming paradigm and why?

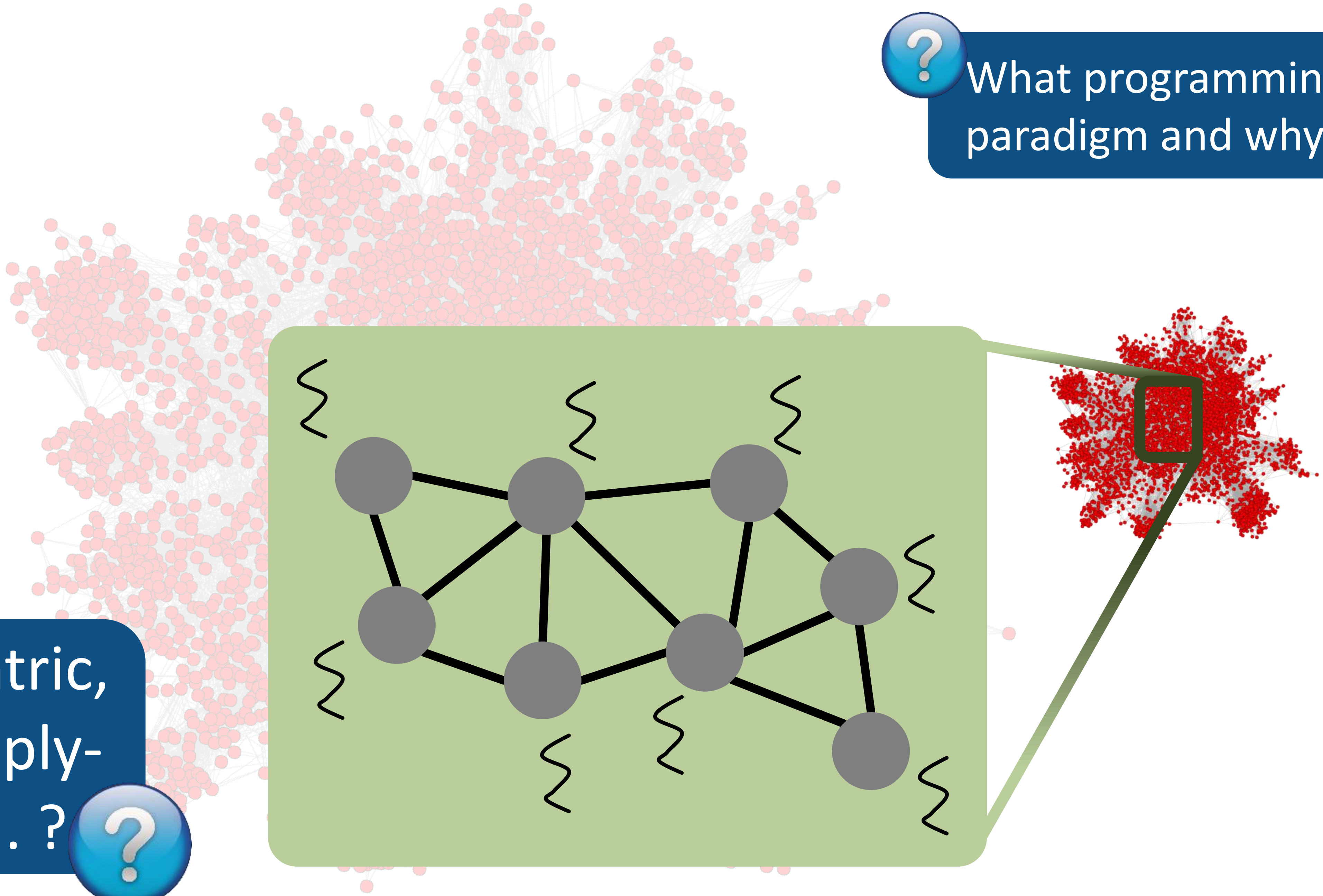
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



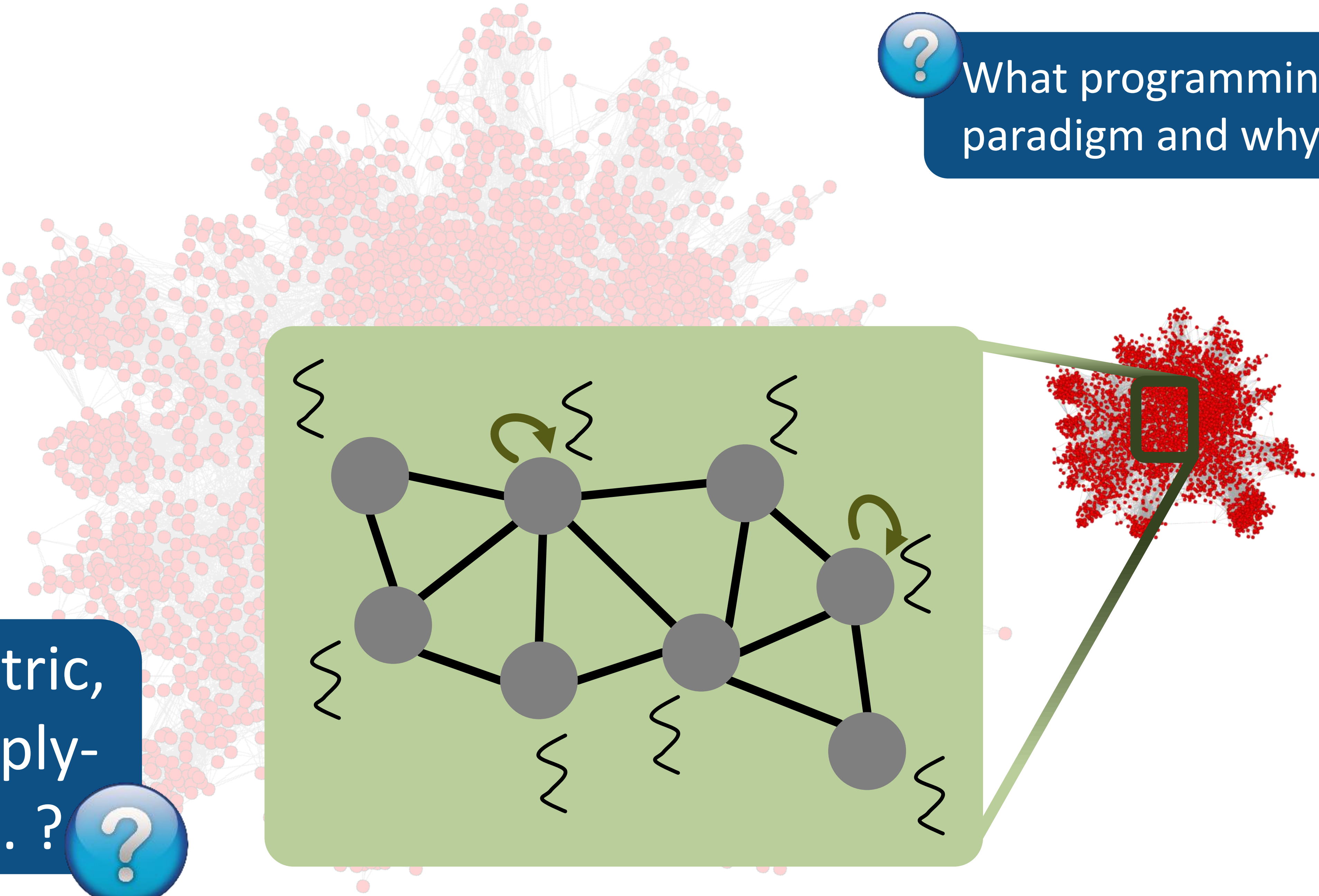
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



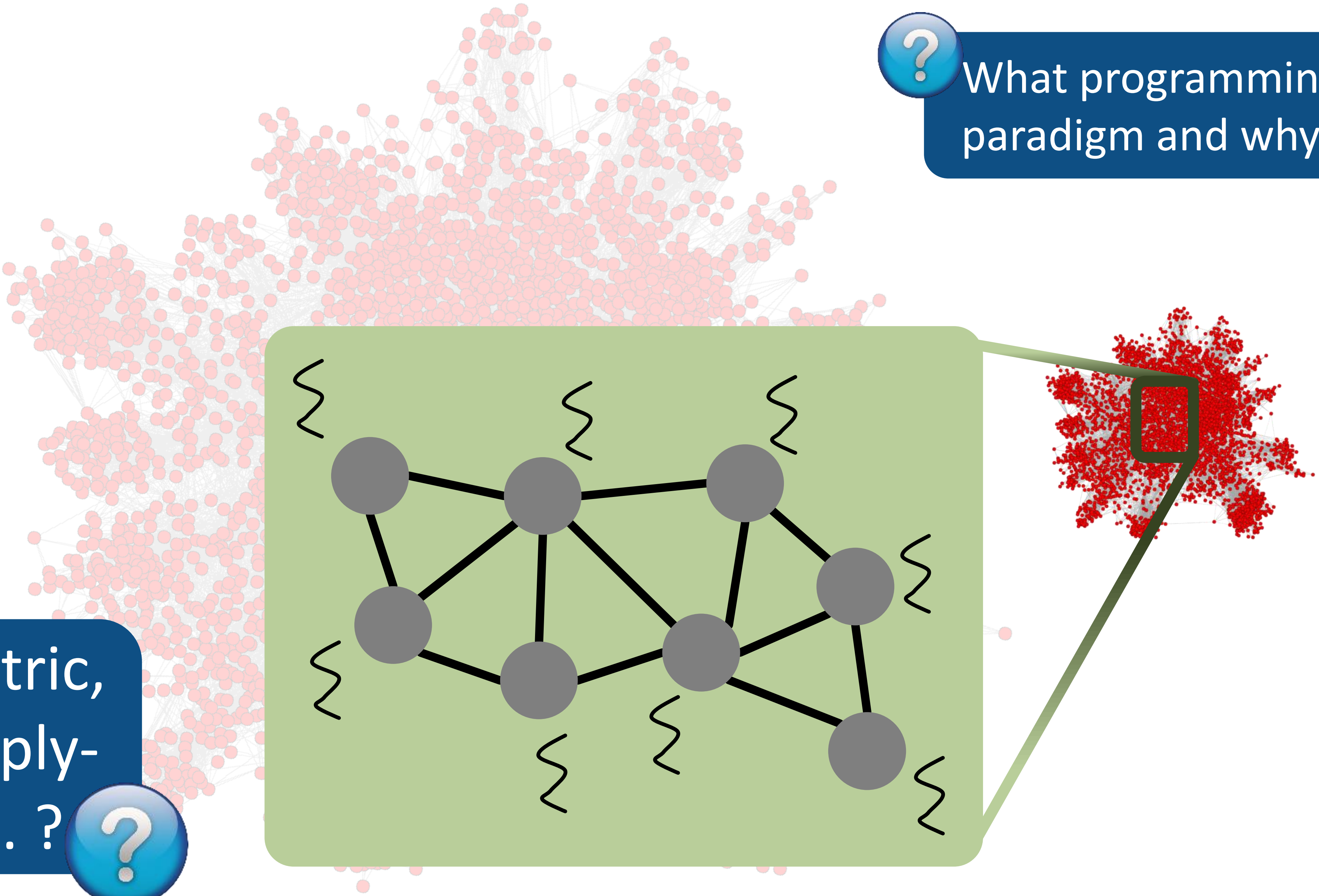
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



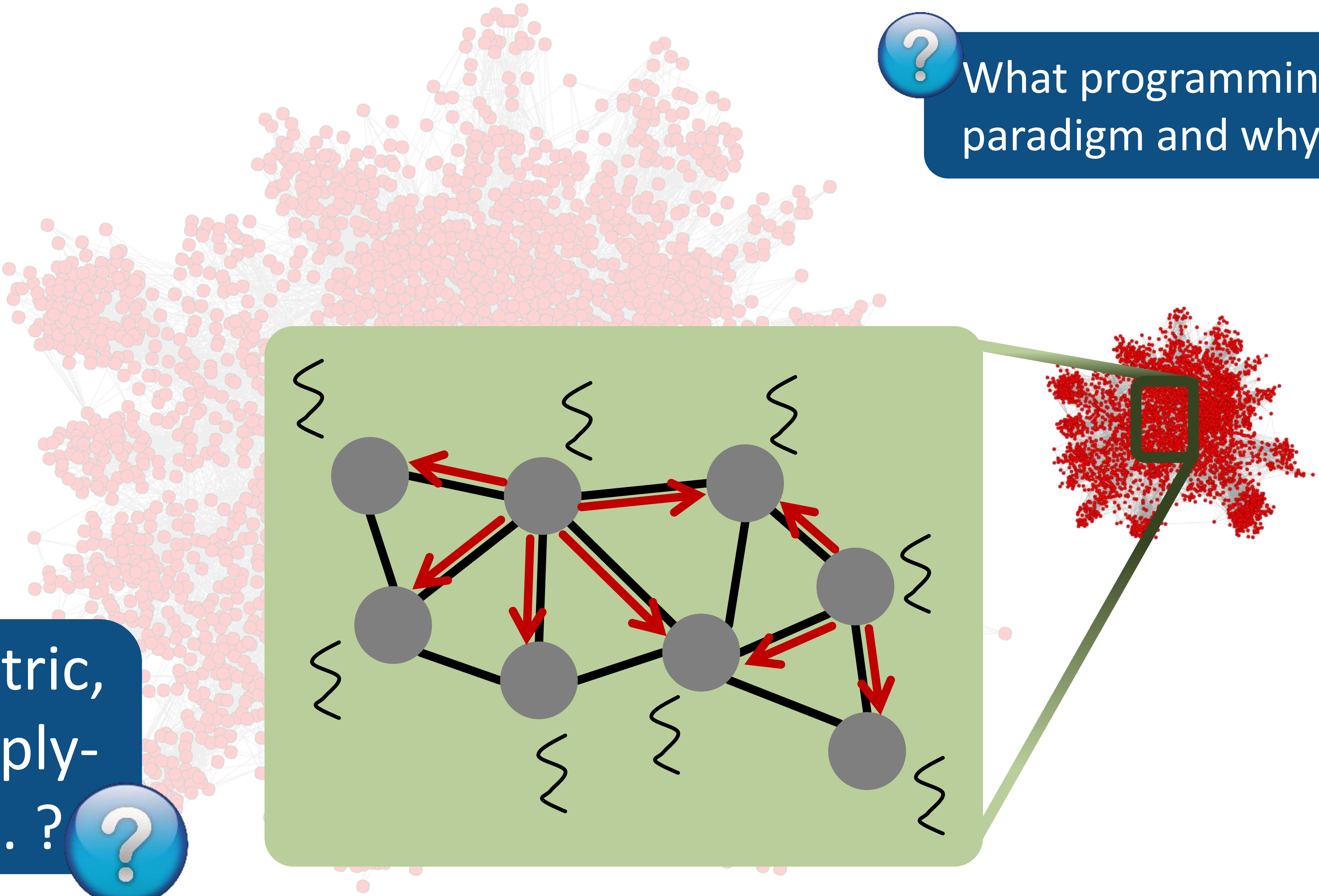
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



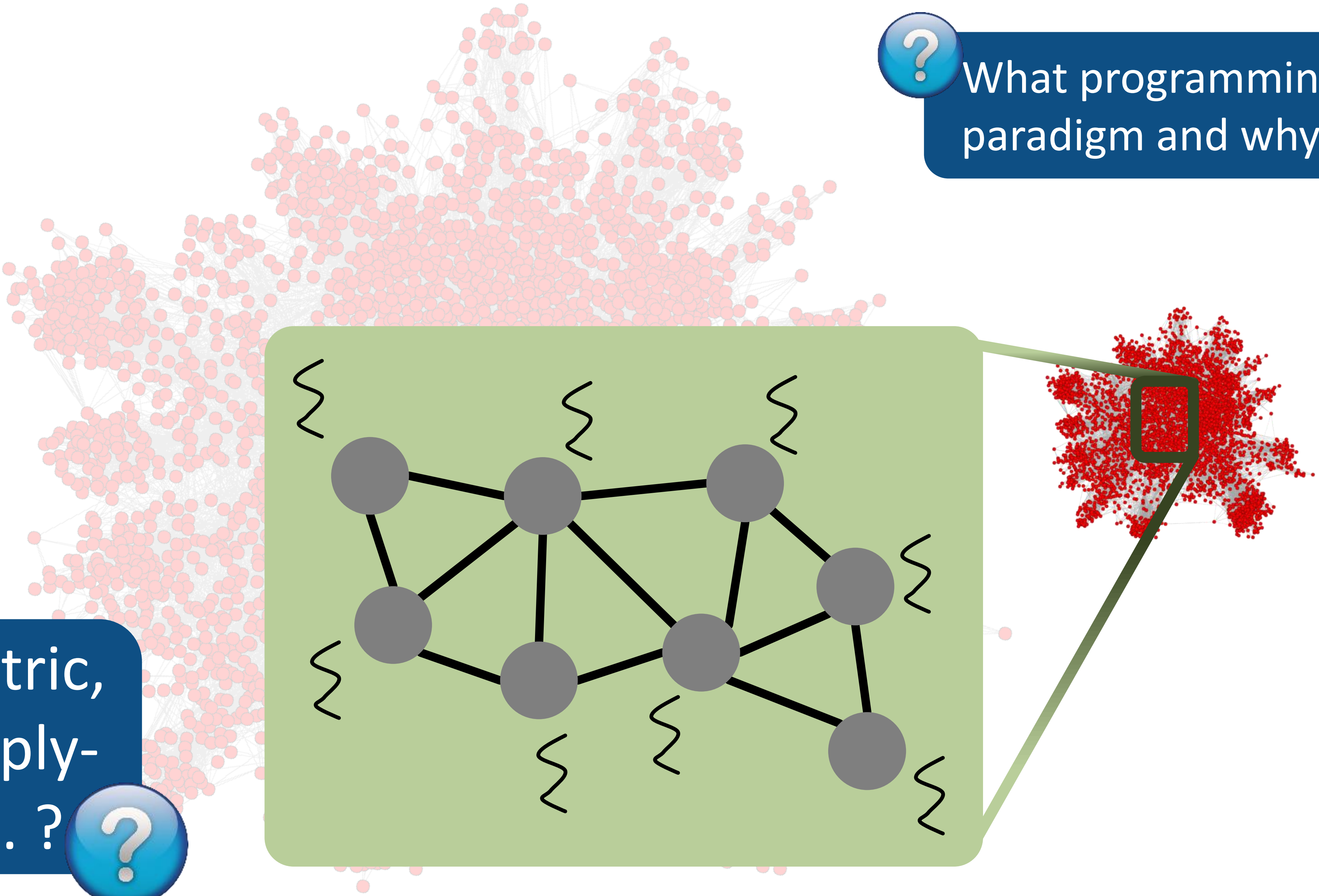
Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?



Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?

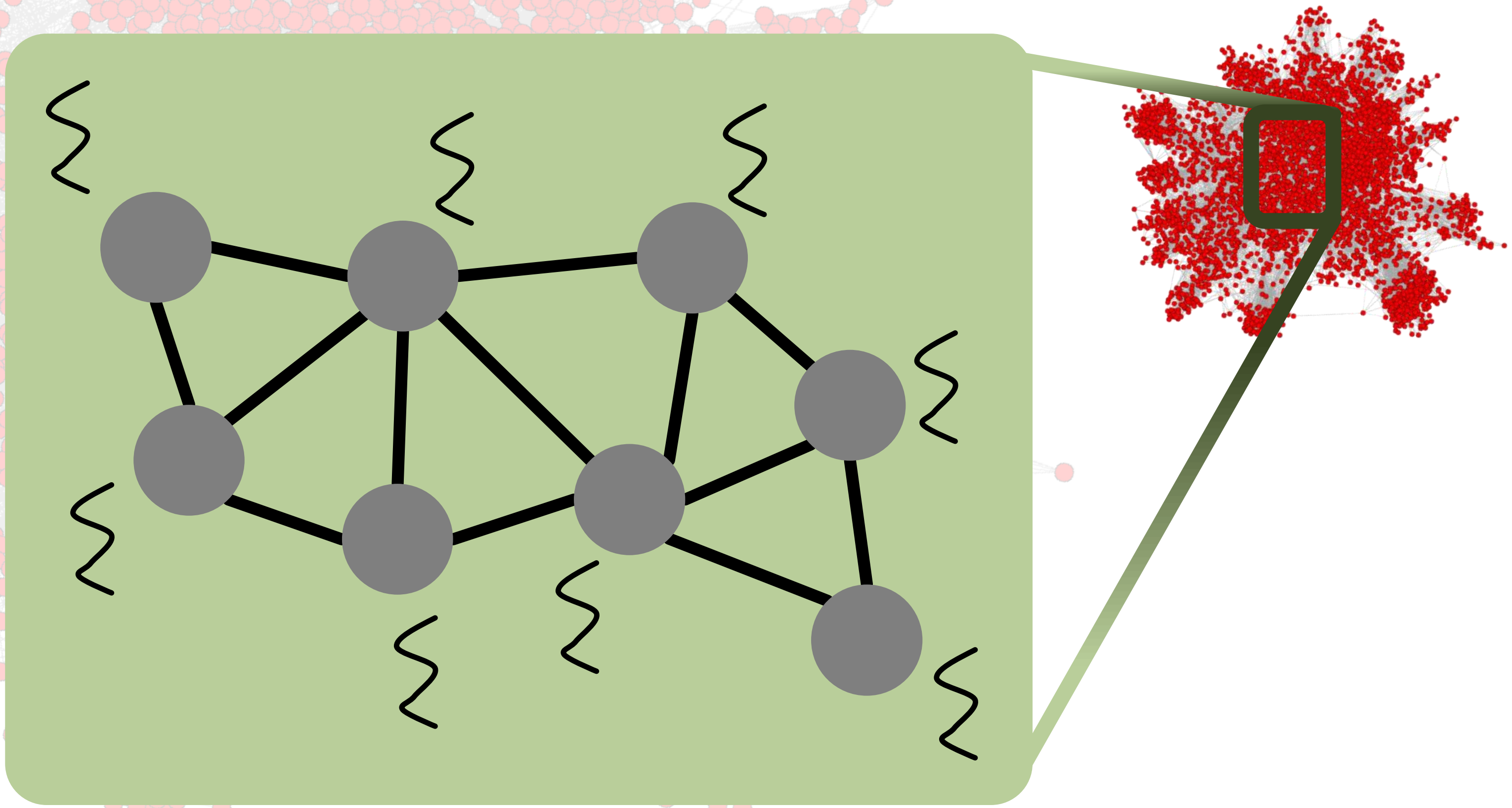


Vertex-centric, Gather-Apply-Scatter, ... ?

- It was designed with the "batch" analytics in mind.
- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?

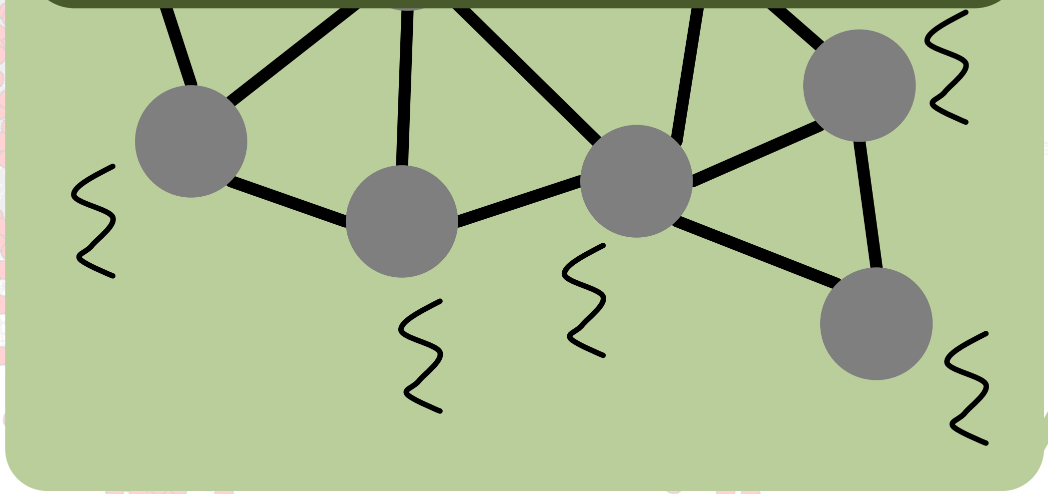


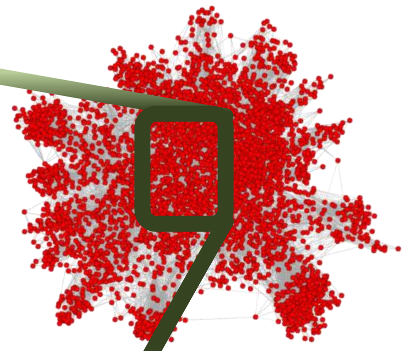
- It was designed with the “batch” analytics in mind.
- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

What programming paradigm and why?

Vertex-centric, Gather-Apply-Scatter, ... ?

“(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations.” [1]





[1] S. Salihoglu and J. Widom, “Optimizing graph algorithms on Pregel-like systems”. VLDB. 2014.

- It was designed with the “batch” analytics in mind.
- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

Vertex-centric, Gather-Apply-Scatter, ... ?

What programming paradigm and why?

“(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations.” [1]

Vertex-Centric (aka Pregel-like) approach is complex for problems such as matchings, spanning trees, etc.

[1] S. Salihoglu and J. Widom, “Optimizing graph algorithms on Pregel-like systems”. VLDB. 2014.

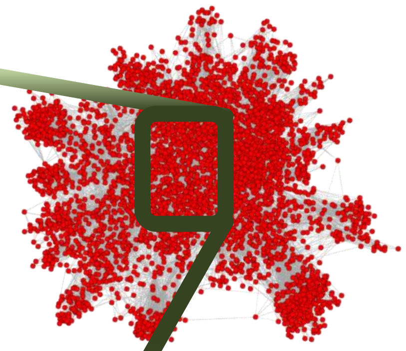
- It was designed with the “batch” analytics in mind.
- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

To be able to **utilize pipelining well**, we really want to use **streaming** (aka the edge-centric paradigm)

What programming paradigm and why?

“(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations.” [1]

Vertex-Centric (aka Pregel-like) approach is complex for problems such as matchings, spanning trees, etc.



Vertex-centric, Gather-Apply-Scatter, ... ?

[1] S. Salihoglu and J. Widom, “Optimizing graph algorithms on Pregel-like systems”. VLDB. 2014.

Can be used but it was designed with the "batch" analytics in mind

...when in DRAM, accessing & parallelization become complex

What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

en	Wikipedia edits (en)	50,757,442	572,591,272
TW	Twitter (WWW)	41,652,230	1,468,365,182
TF	Twitter (MPI)	52,579,682	1,963,263,821
FR	Friendster	68,349,466	2,586,147,869
UL	UK domain (2007)	105,153,952	3,301,876,564

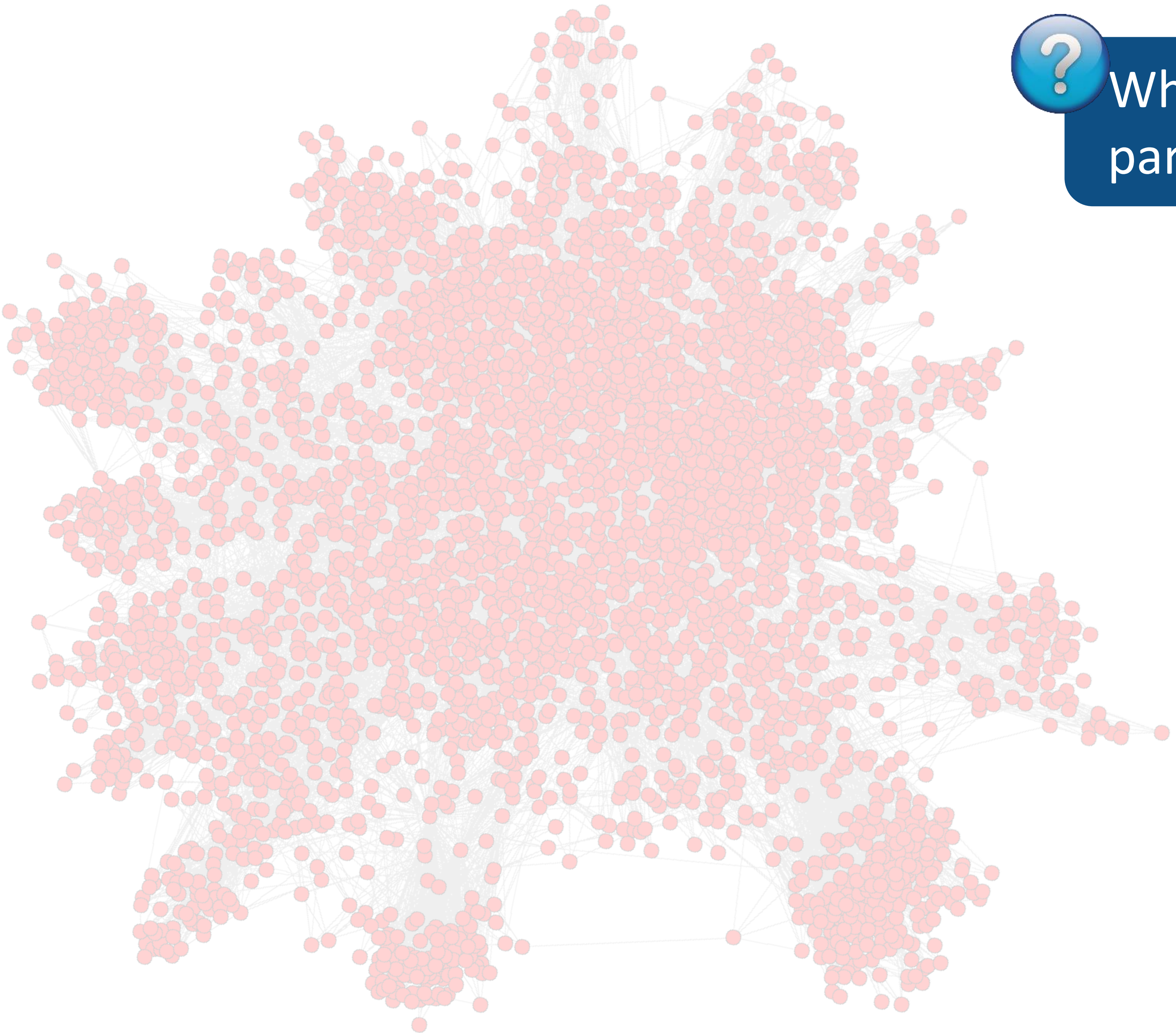
KONECT graph datasets

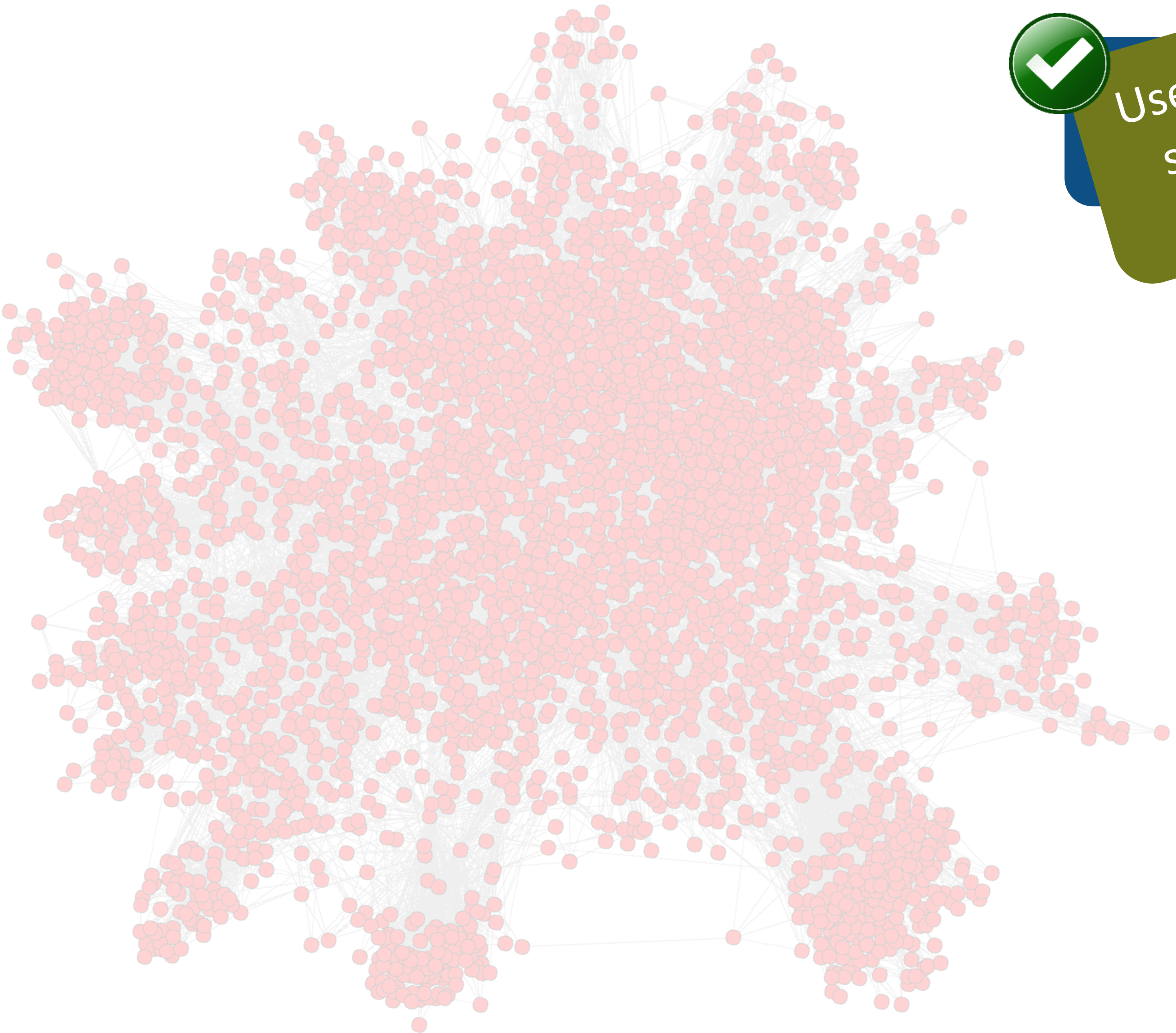
Vertex-centric, Gather-Apply Scatter, ... ?

Webgraph datasets

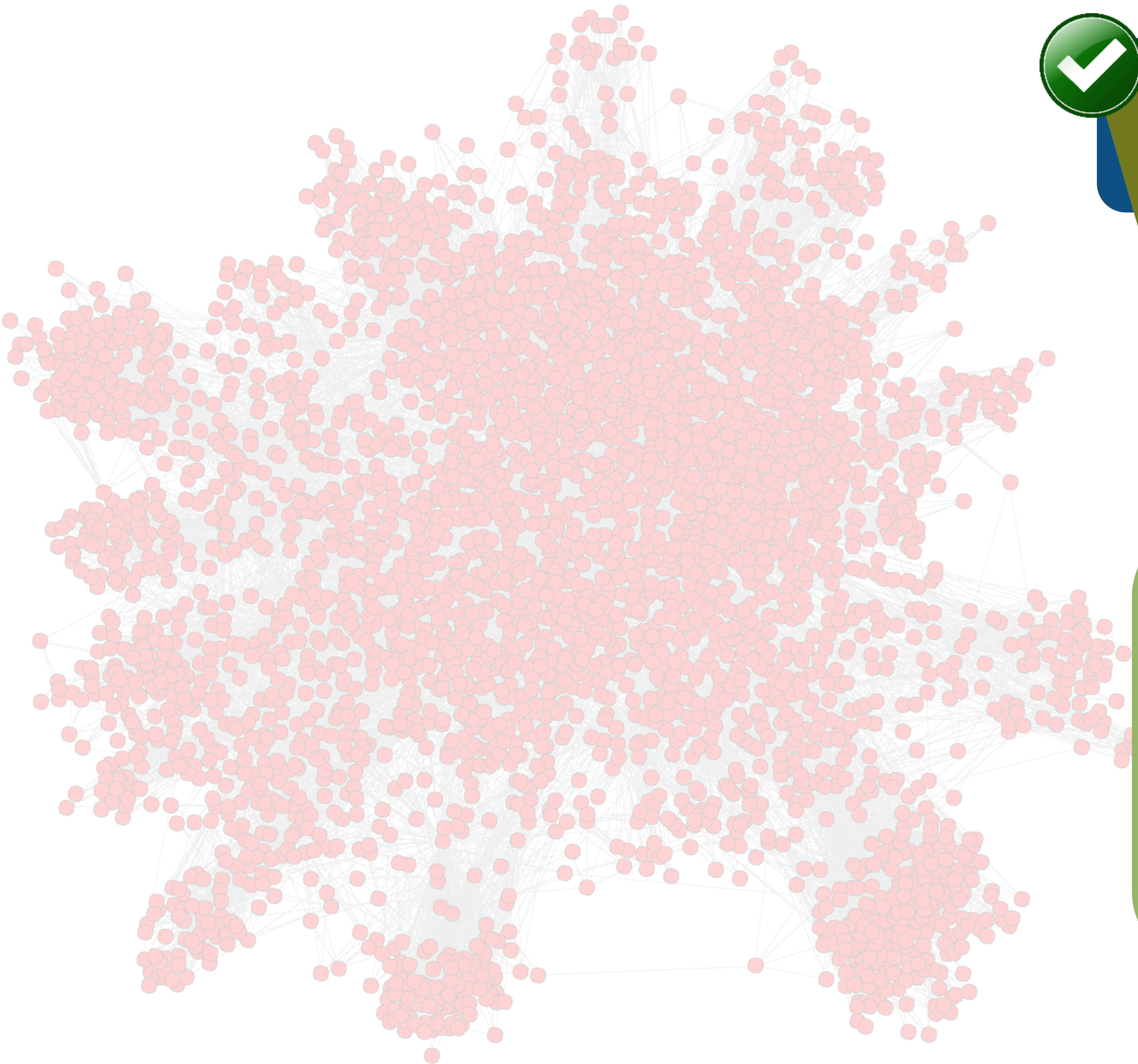
Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152
uk-2014-host	2014	4 769 354	50 829 923
eu-2015-host	2015	11 264 052	386 915 963
gsh-2015-host	2015	68 660 142	1 802 747 600
uk-2014-tpd	2014	1 766 010	18 244 650
eu-2015-tpd	2015	6 650 532	170 145 510
gsh-2015-tpd	2015	30 809 122	602 119 716
clueweb12	2012	978 408 098	42 574 107 469
uk-2002	2002	18 520 486	298 113 762

What programming paradigm and why?





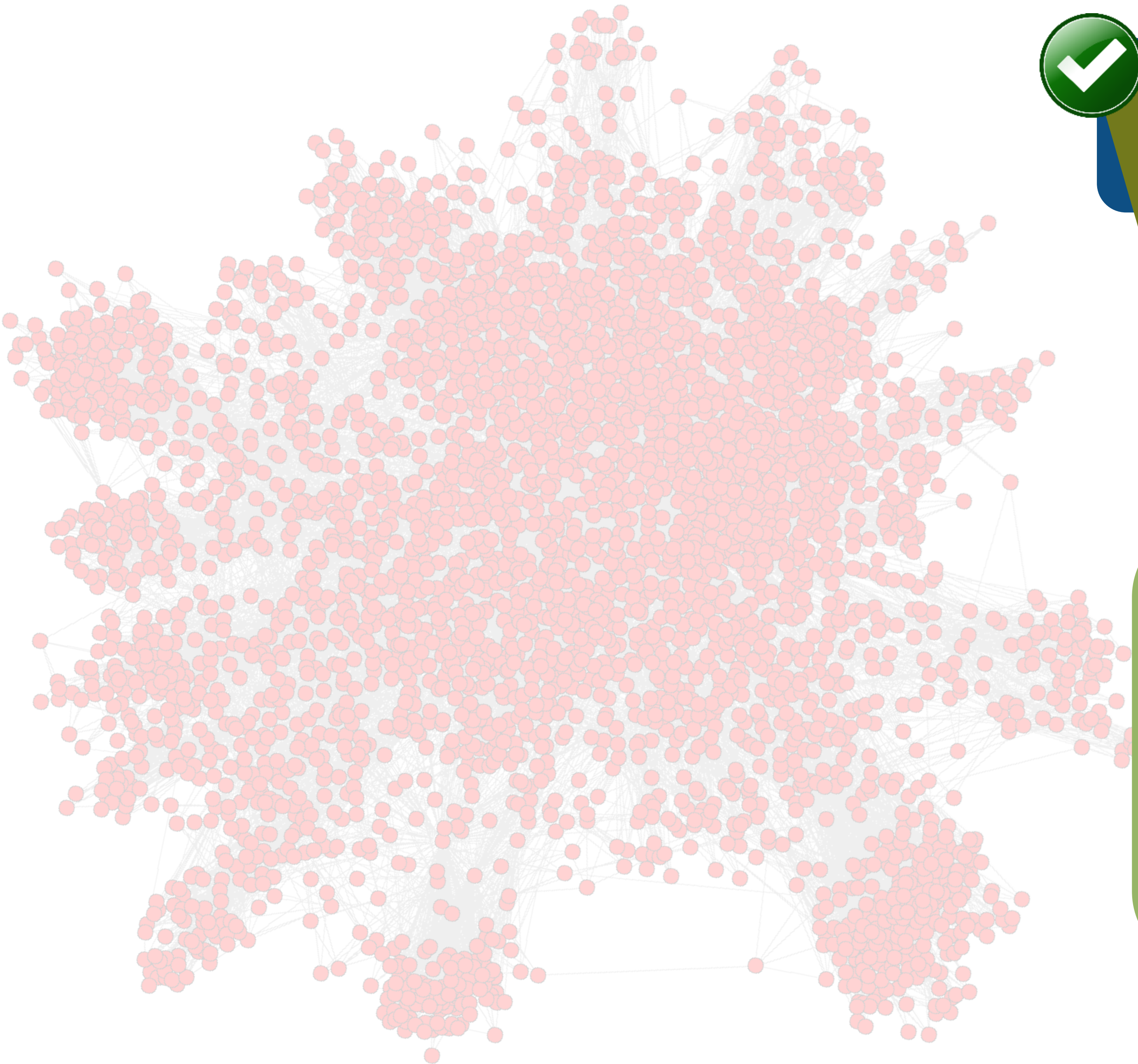
Use some form of streaming (aka **edge-centric**)

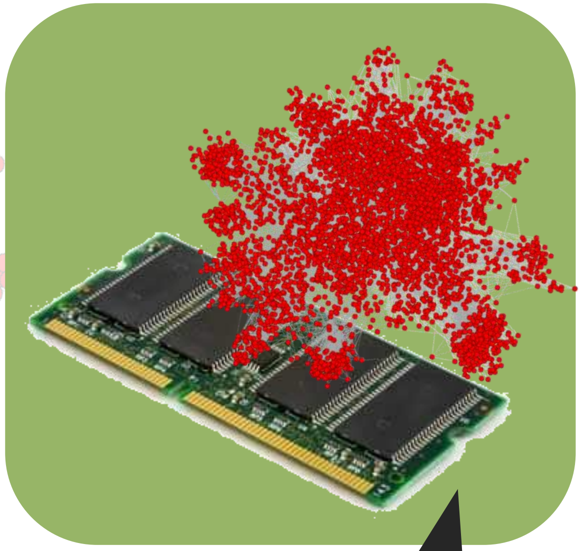


Use some form of streaming (aka **edge-centric**)

DRAM

Use some form of streaming (aka **edge-centric**)

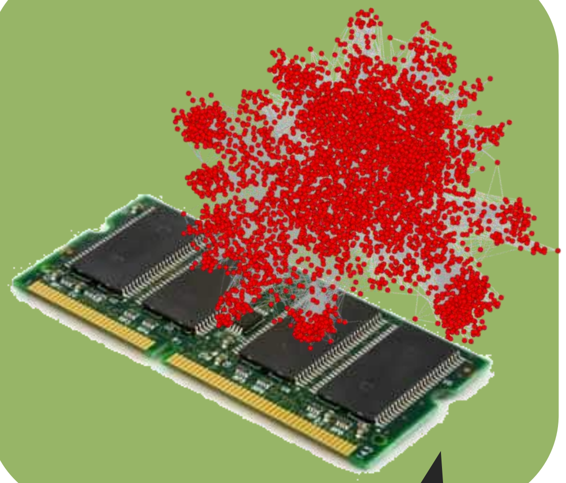




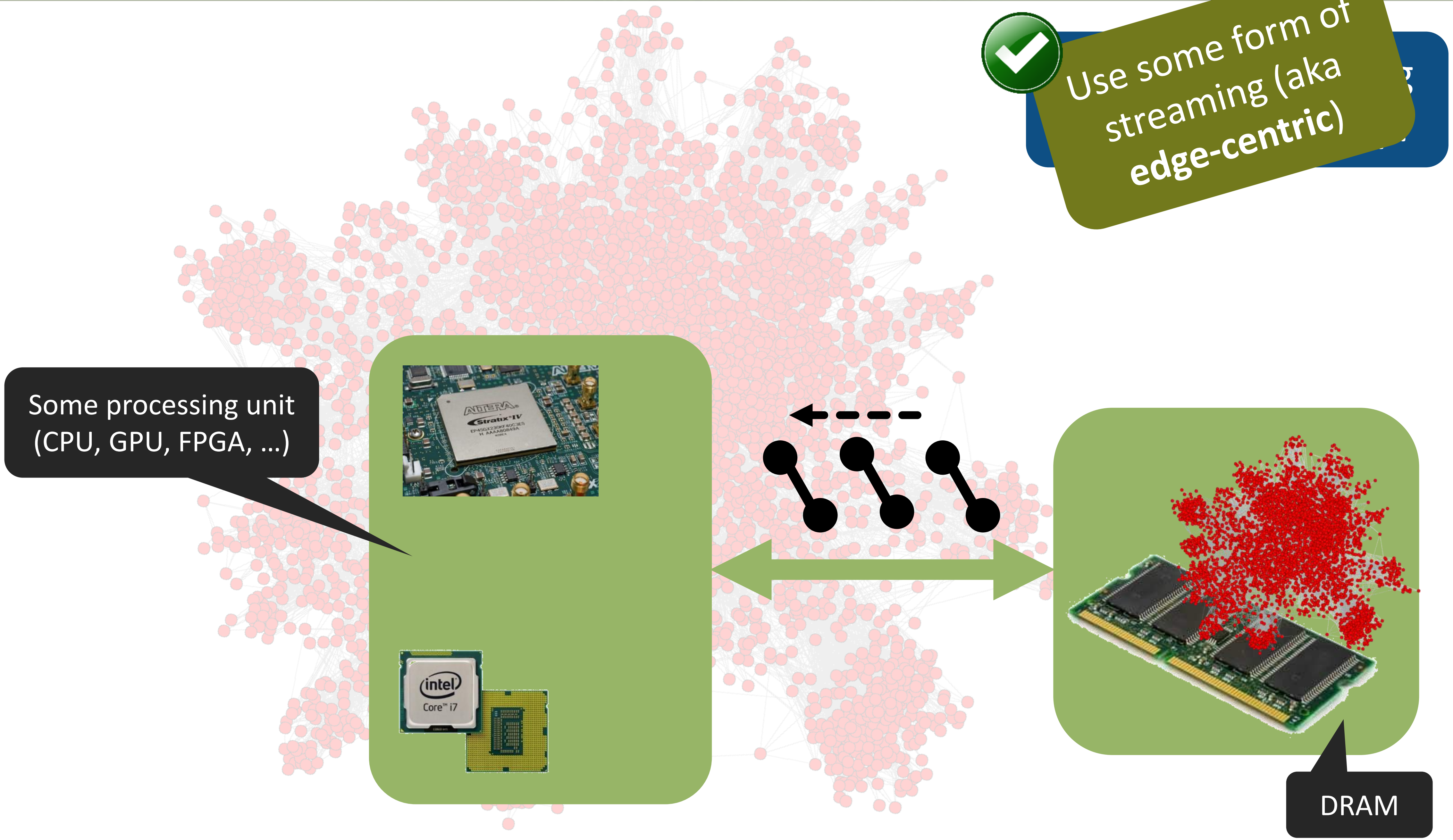
DRAM

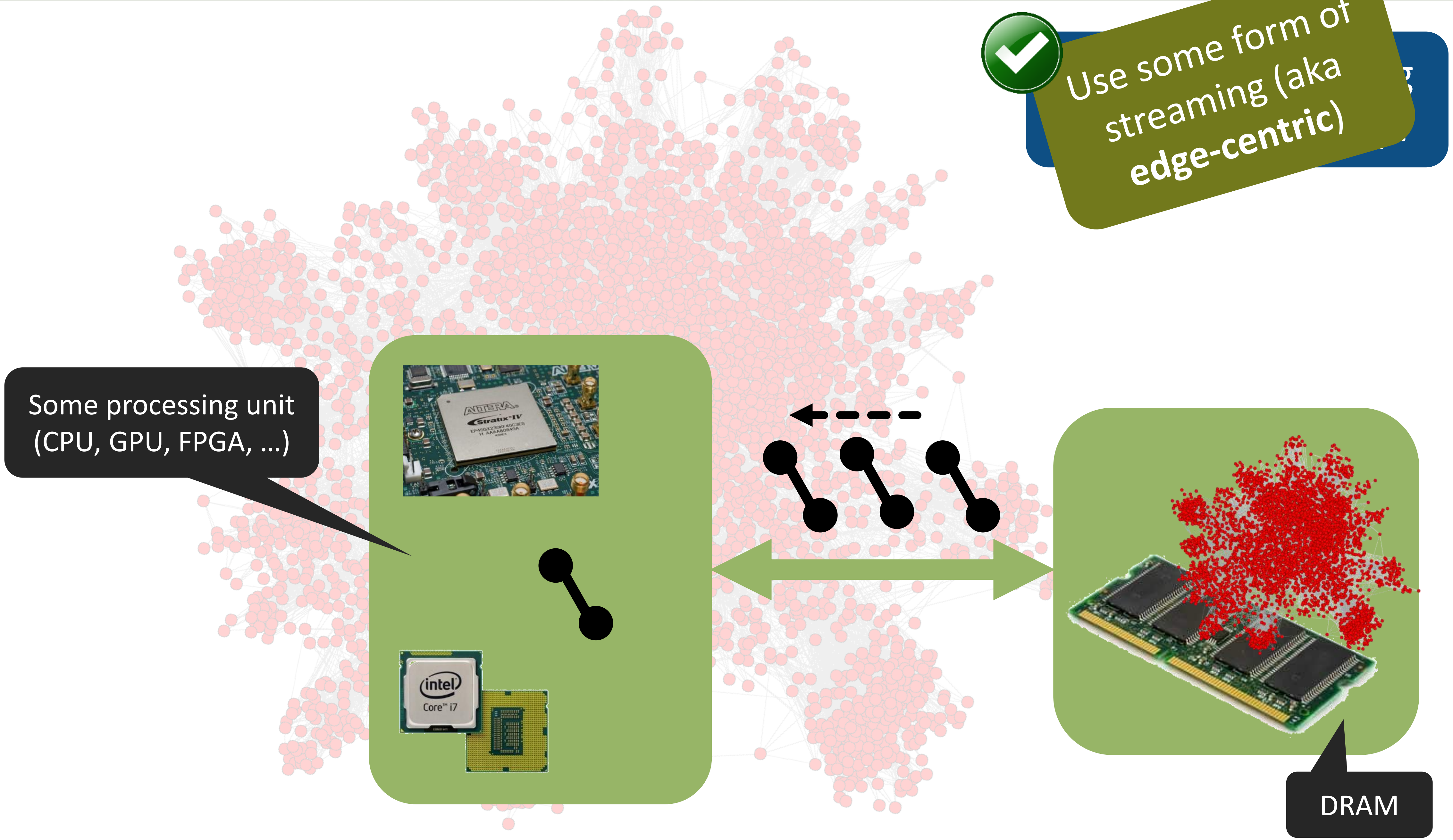
Use some form of streaming (aka **edge-centric**)

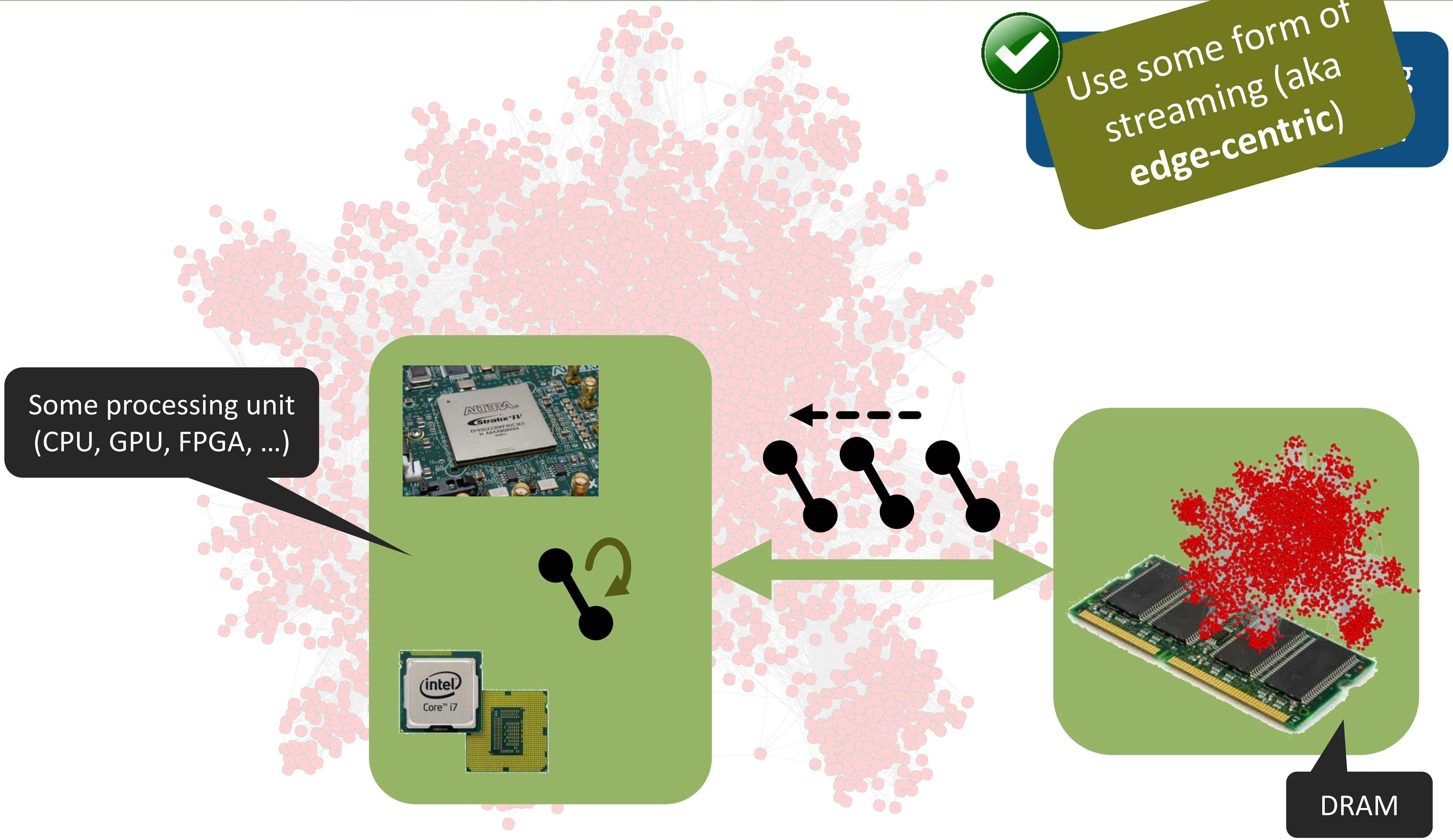
Some processing unit (CPU, GPU, FPGA, …)

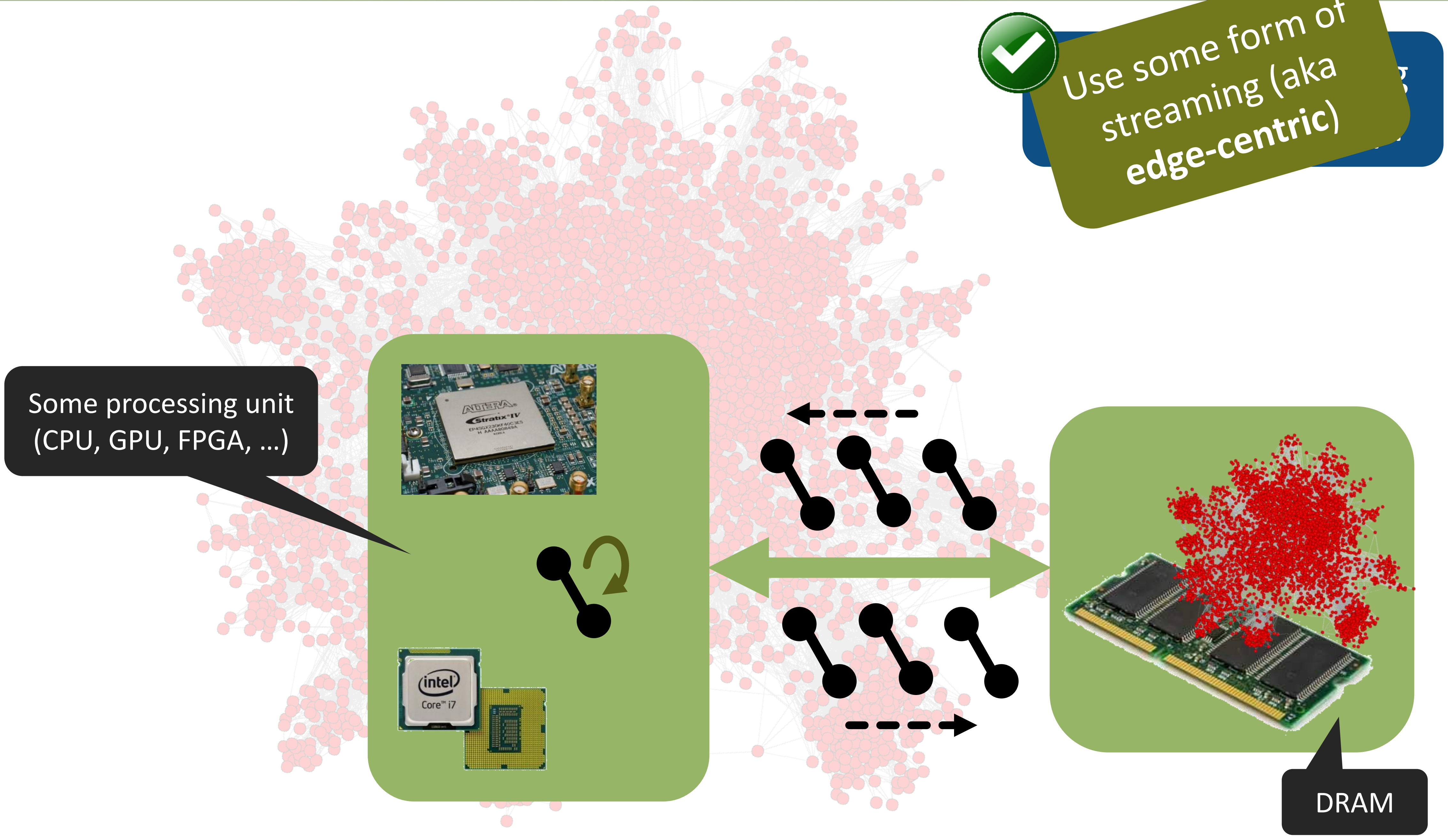


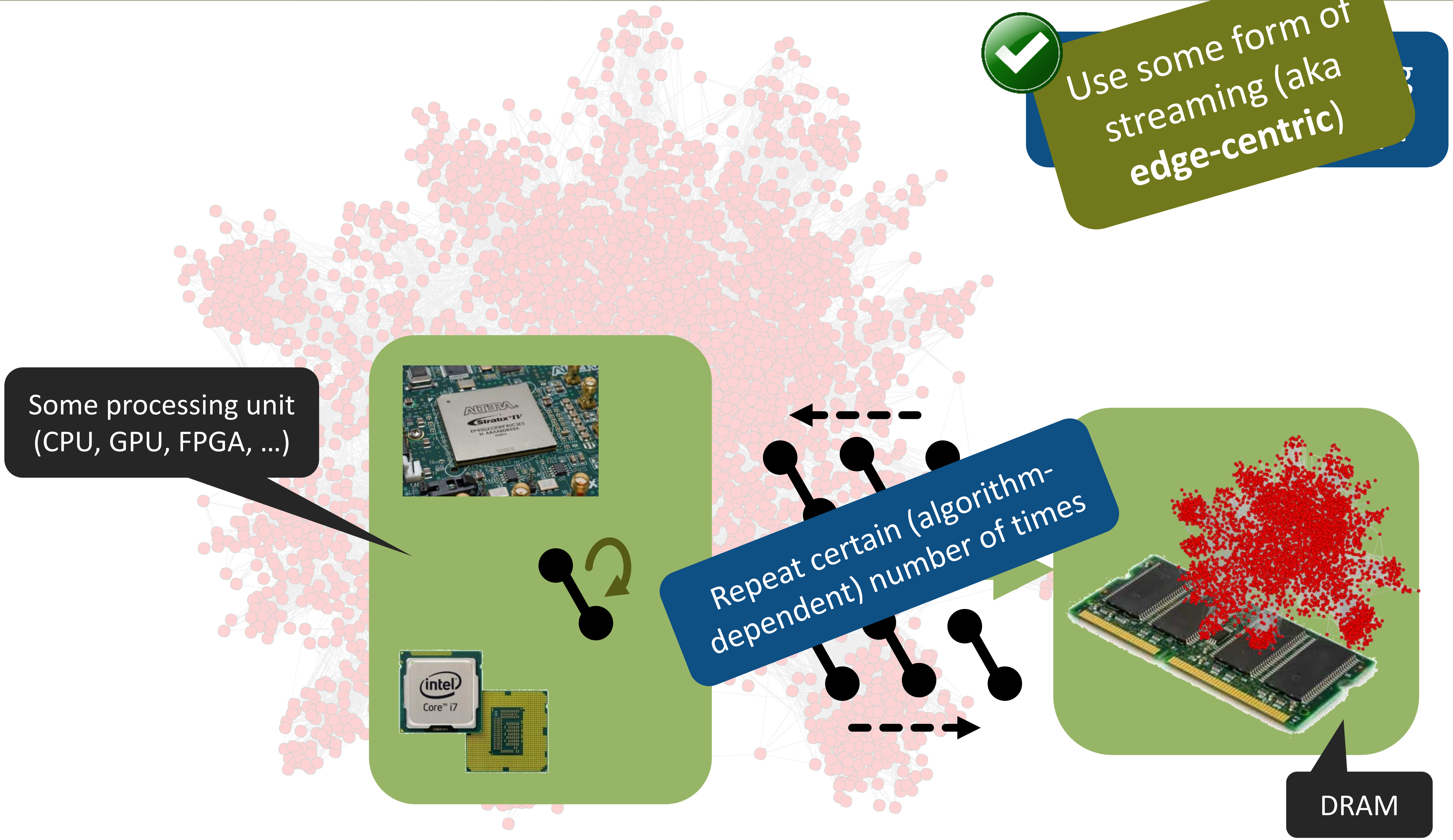
DRAM


Use some form of streaming (aka **edge-centric**)
Some processing unit (CPU, GPU, FPGA, …)
DRAM


Use some form of streaming (aka **edge-centric**)
Some processing unit (CPU, GPU, FPGA, …)
DRAM

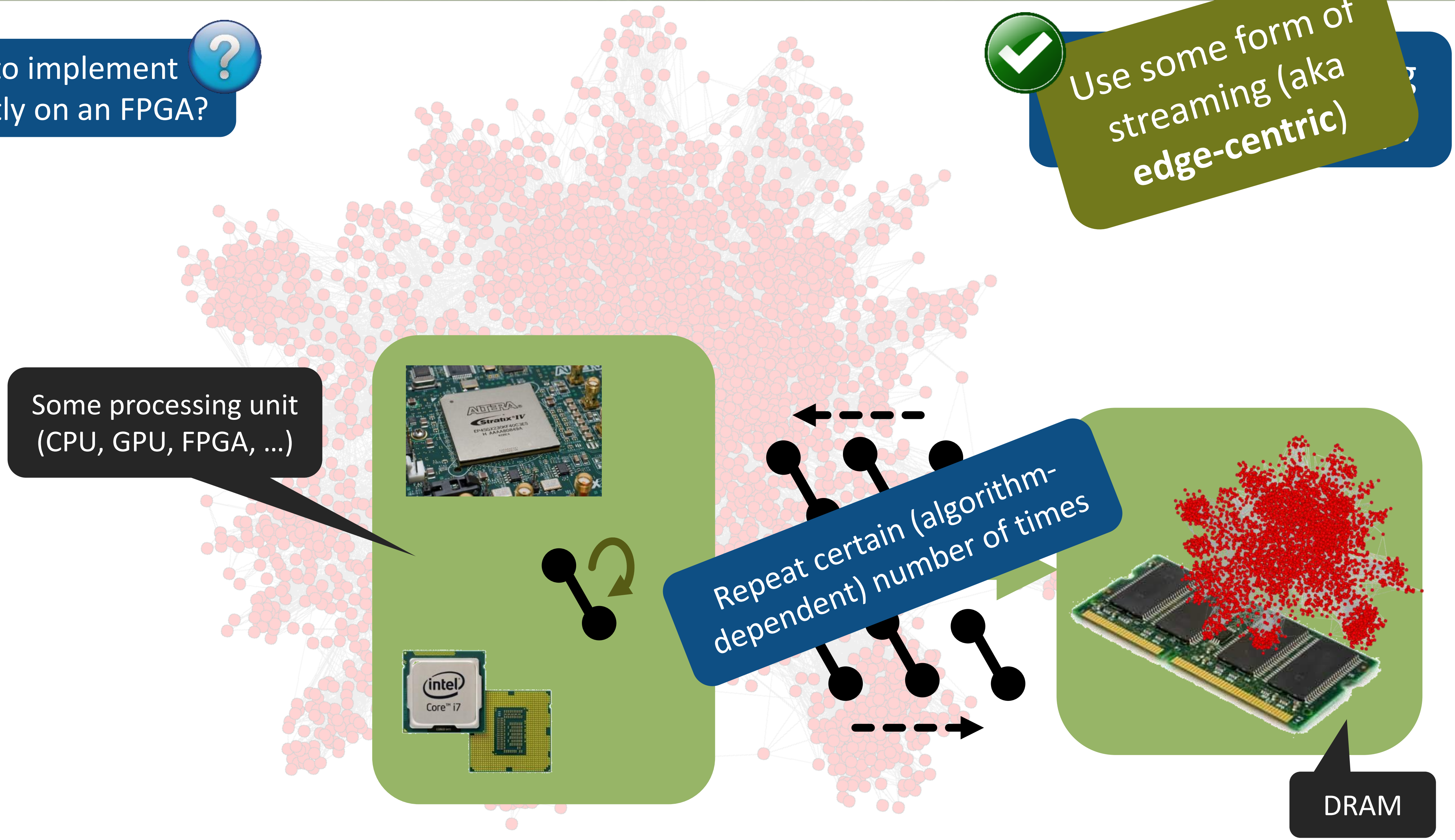

Use some form of streaming (aka **edge-centric**)
Some processing unit (CPU, GPU, FPGA, …)
DRAM


Use some form of streaming (aka **edge-centric**)
Some processing unit (CPU, GPU, FPGA, …)
DRAM


Use some form of streaming (aka **edge-centric**)
Some processing unit (CPU, GPU, FPGA, …)
Repeat certain (algorithm-dependent) number of times
DRAM

How to implement efficiently on an FPGA?

Use some form of streaming (aka **edge-centric**)



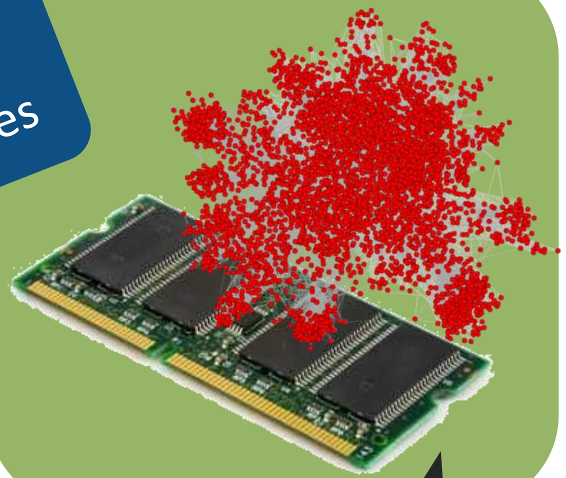
How to implement efficiently on an FPGA?

Processing edges is sequential – how to incorporate parallelism?

Use some form of streaming (aka **edge-centric**)

Some processing unit (CPU, GPU, FPGA, …)

Repeat certain (algorithm-dependent) number of times



DRAM

How to implement efficiently on an FPGA?

Processing edges is sequential – how to incorporate parallelism?

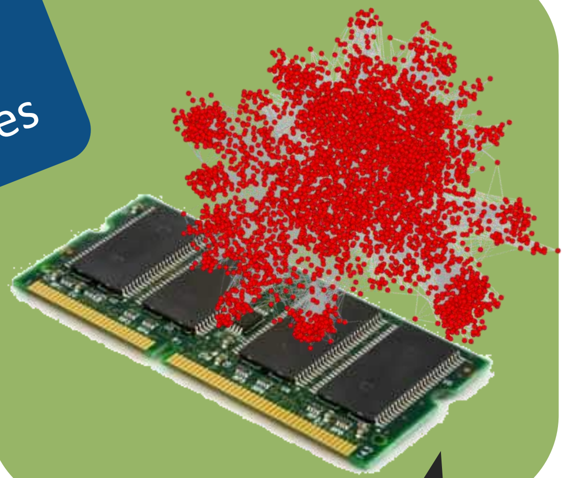
How to minimize the number of “passes” over edges? (This can get **really** bad in the “traditional” edge-centric approach, e.g., BFS normally needs *O(m+n)* work, while in the edge-centric approach it takes *O(D m)* work (*D* passes [1]),

Use some form of streaming (aka **edge-centric**)

m: #edges in a graph
n: #vertices in a graph
D: graph’s diameter (usually ~5-15)

Some processing unit (CPU, GPU, FPGA, …)

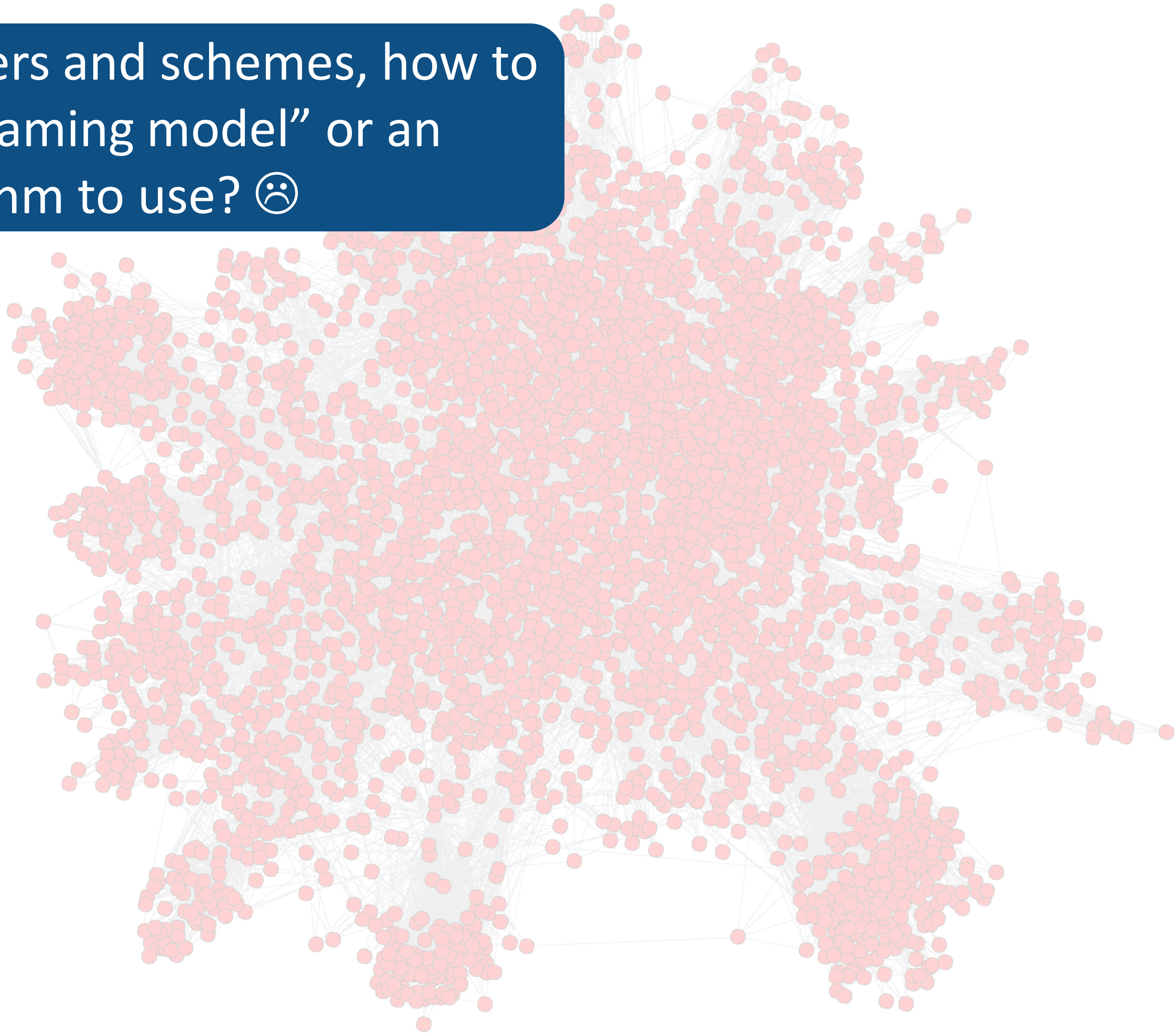
Repeat certain (algorithm-dependent) number of times



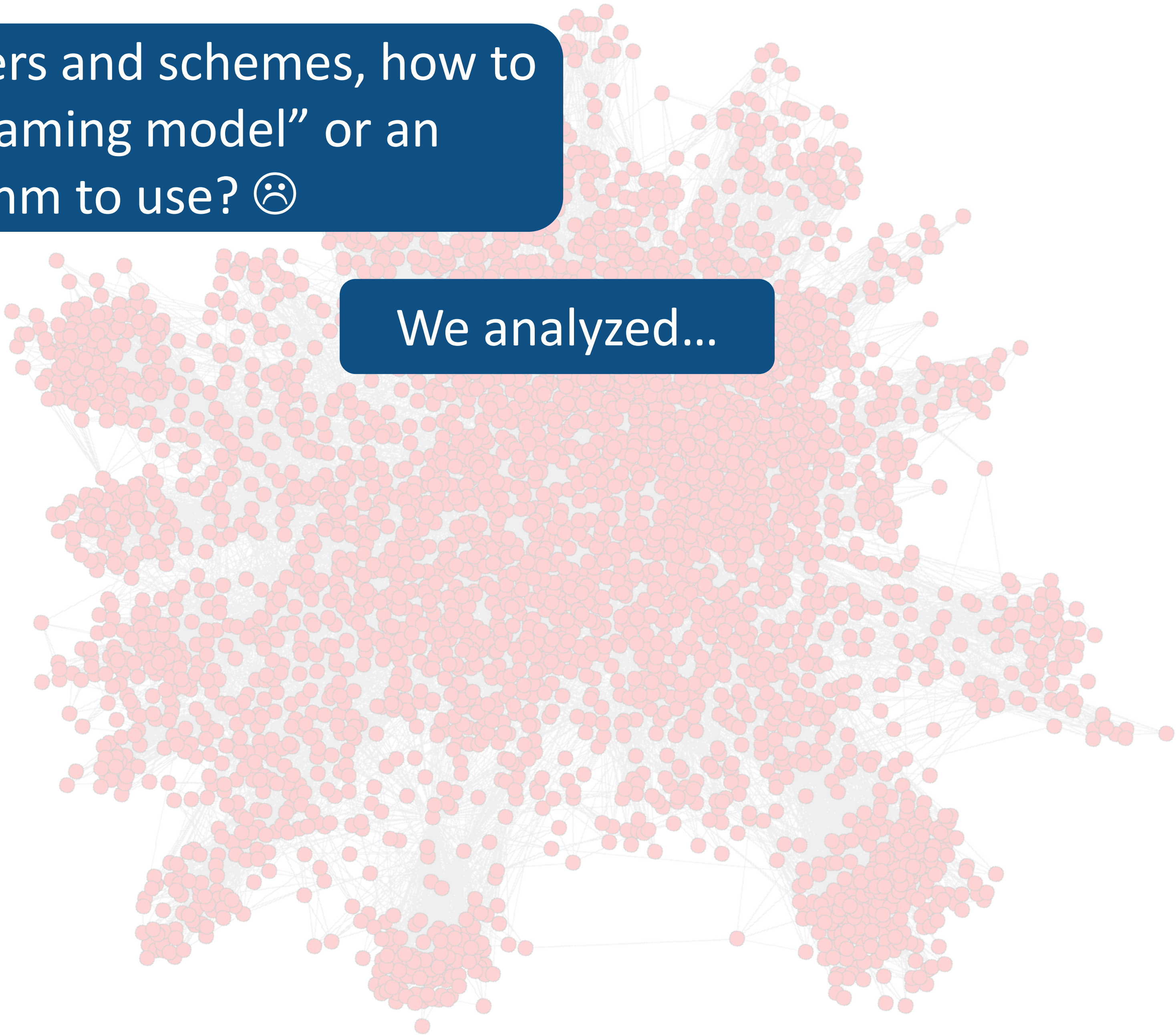
DRAM

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. SOSP. 2013.

Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹



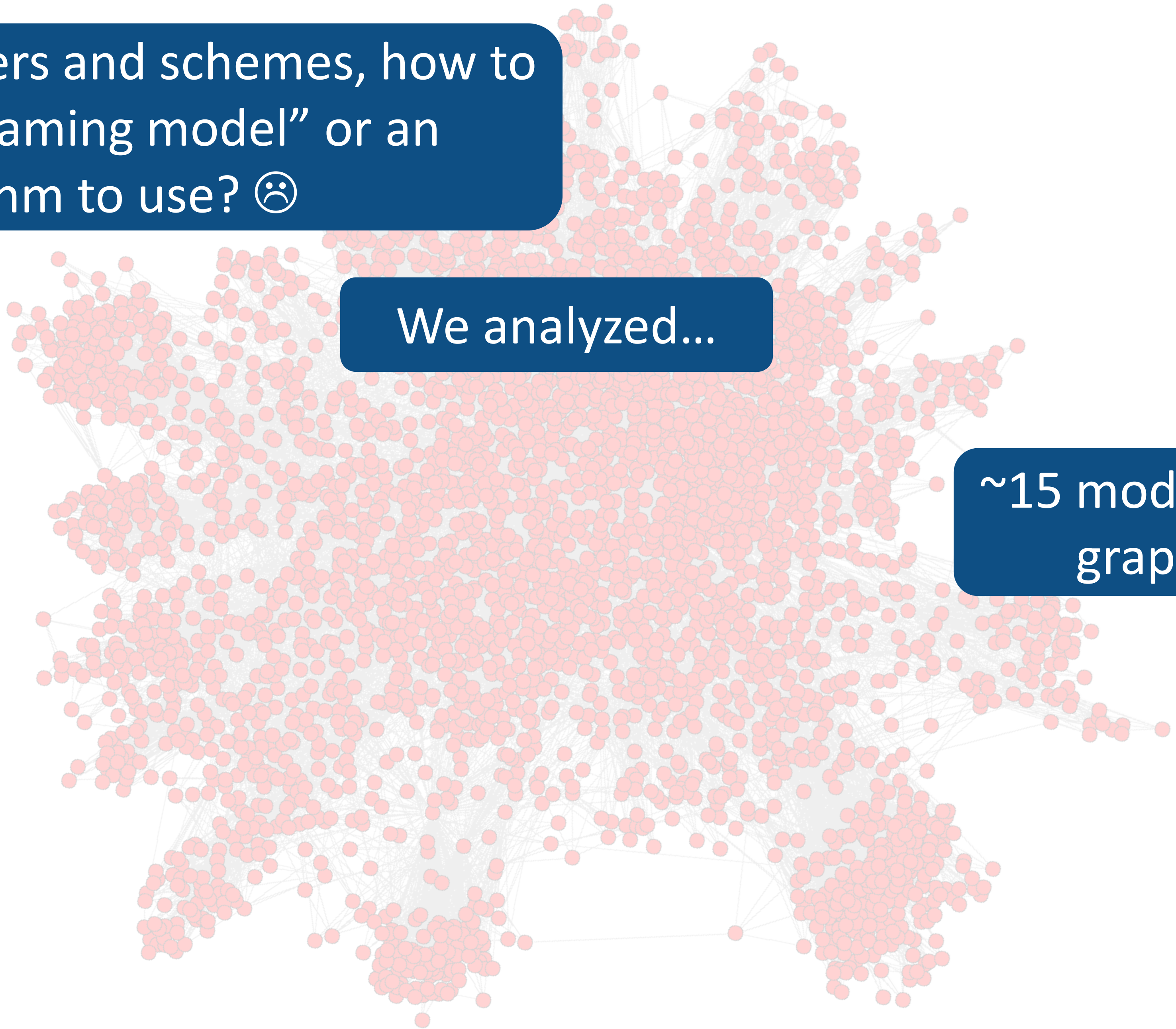
Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹



We analyzed…

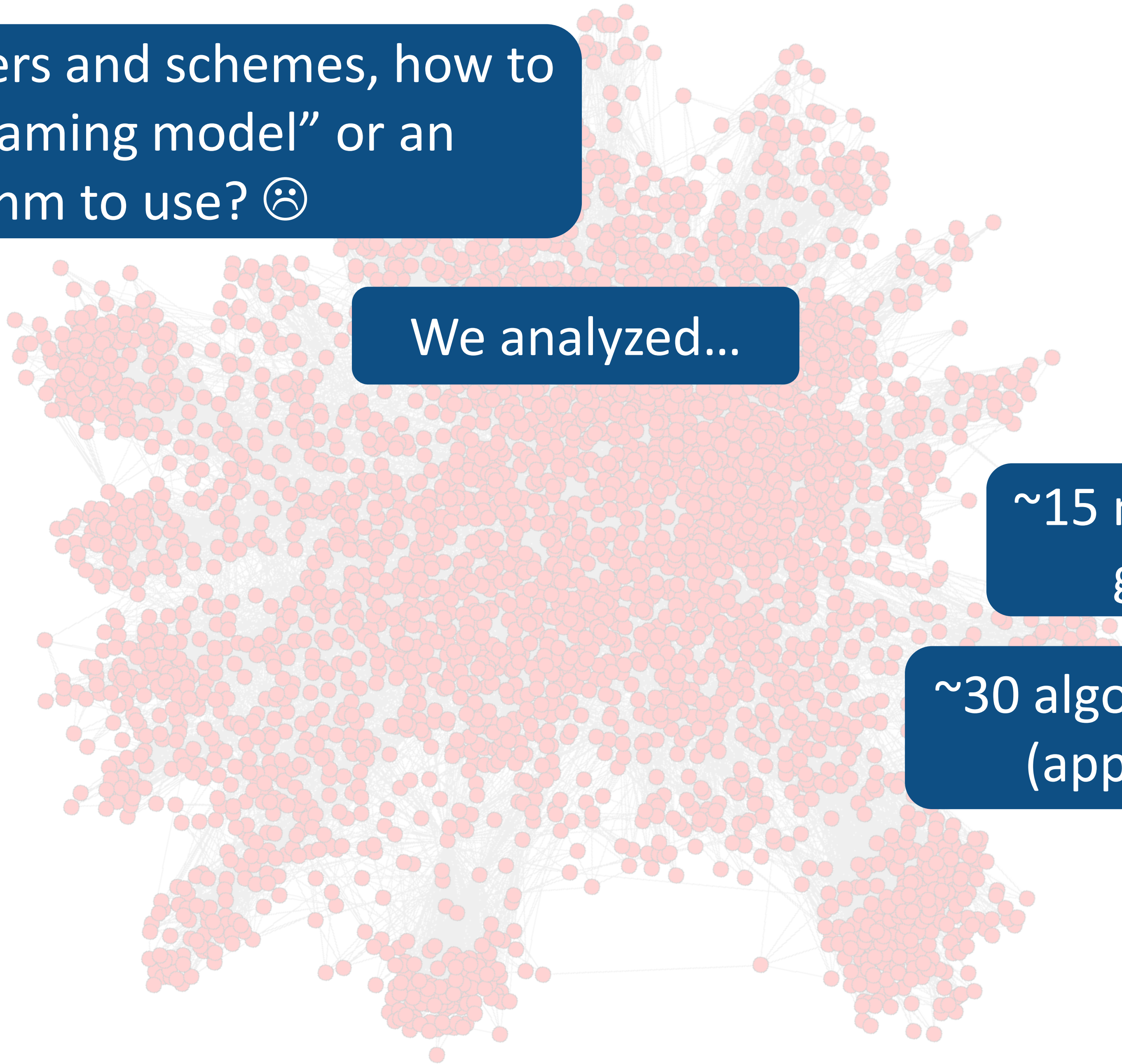
Hundreds of papers and schemes, how to select a “streaming model” or an algorithm to use? ☹

We analyzed…



~15 models for streaming graph processing

Hundreds of papers and schemes, how to select a “streaming model” or an algorithm to use? ☹



We analyzed…

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹

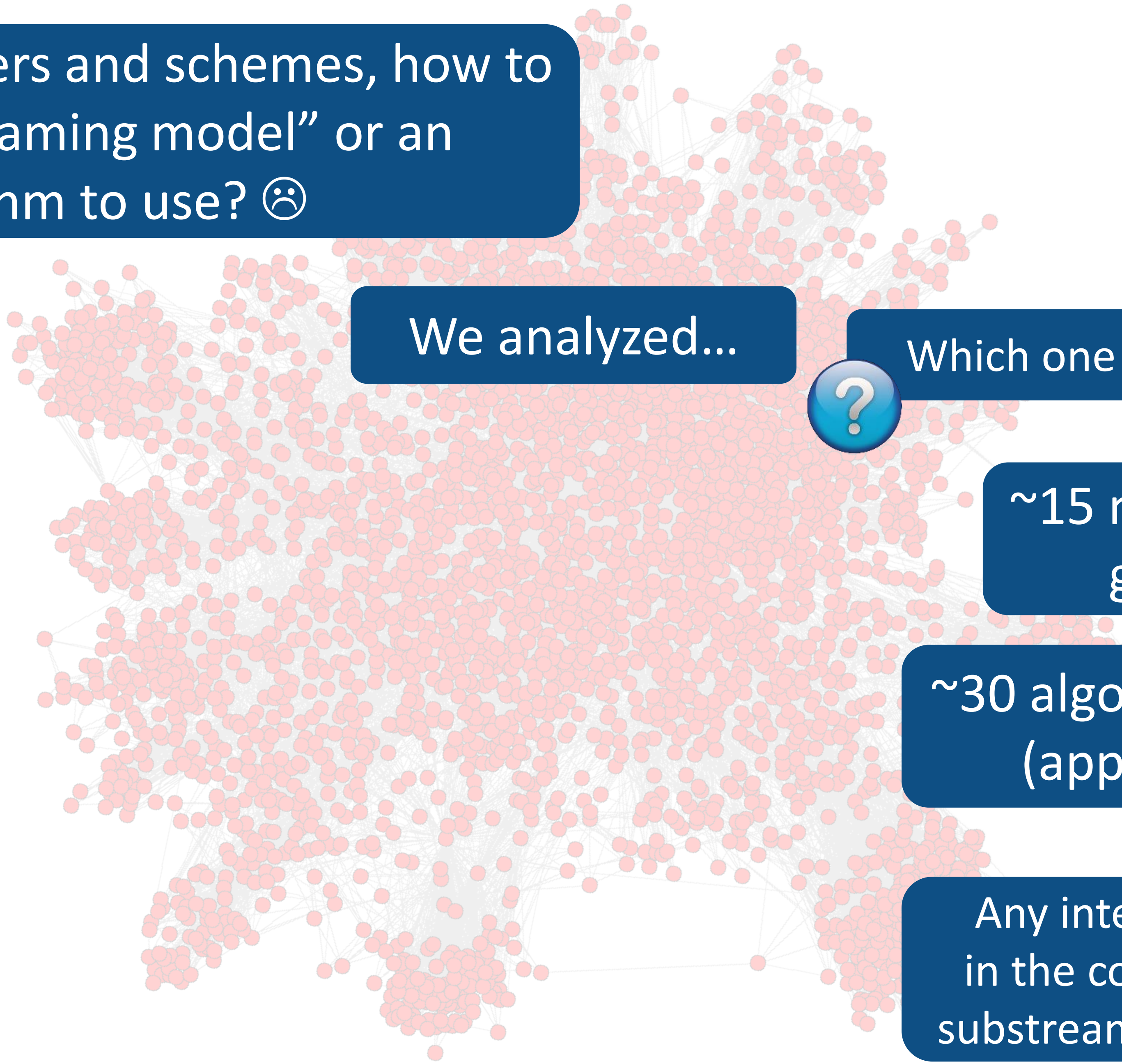
We analyzed...

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹



We analyzed…

Which one to select?

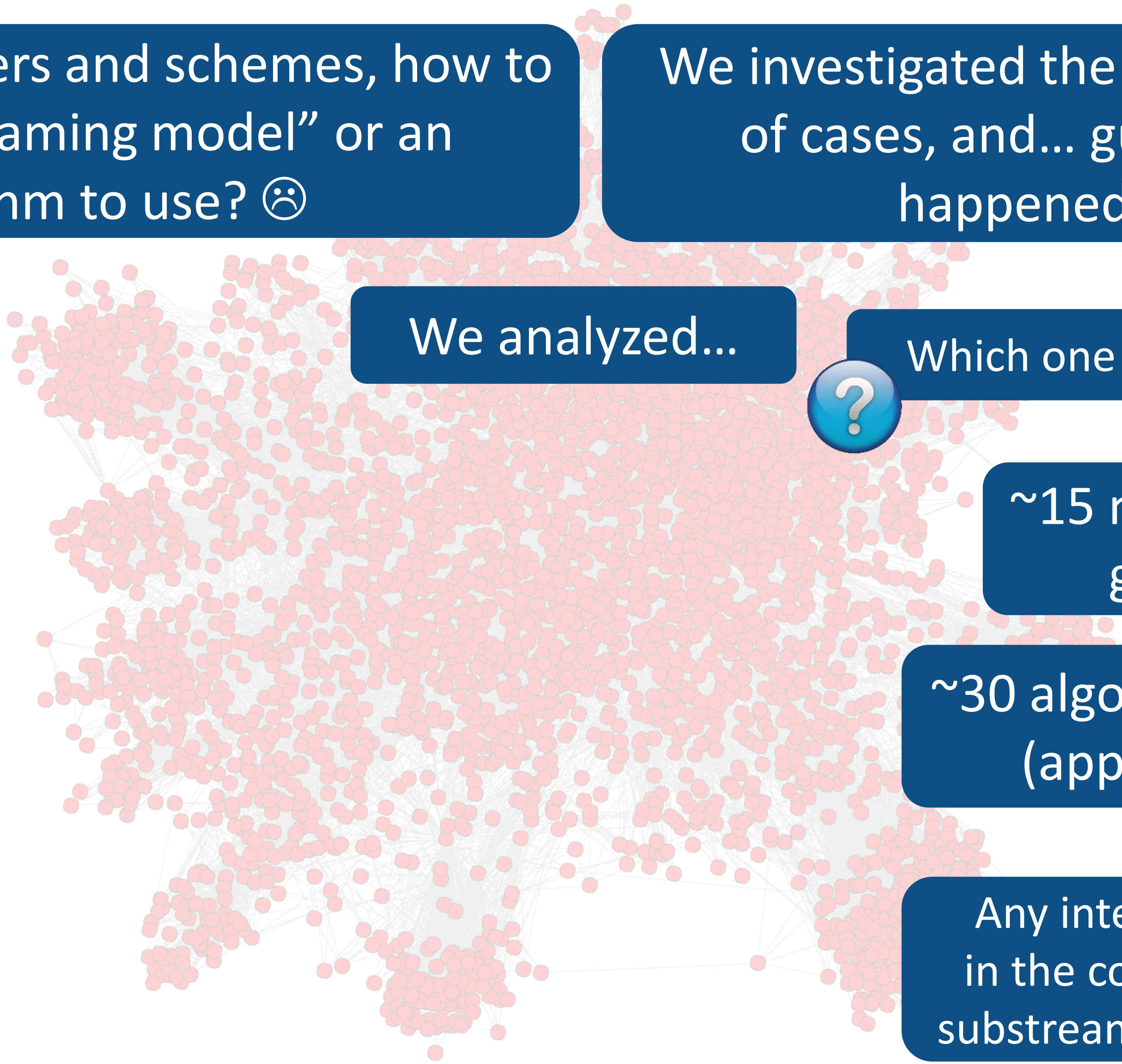
~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?

Hundreds of papers and schemes, how to select a “streaming model” or an algorithm to use? ☹

We investigated the vast majority of cases, and… guess what happened ☺



We analyzed…

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?

Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹

We investigated the vast majority of cases, and… guess what happened ☺

We analyzed…

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including machine learning, social network analysis, computational sciences, and others. Two key challenges that hinder accelerating graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing rate of graph updates, with millions of edges added or removed per second. Graph streaming algorithms are specifically crafted to eliminate these issues: The input graph is passed as a stream of updates, allowing to add and remove edges in a simple way. Recent years have seen the development of many such algorithms. However, they differ in the time needed to add or remove an edge, the required random access memory space, the number of passes

Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? ☹

We investigated the vast majority of cases, and… guess what happened ☺

We analyzed…

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?

It is almost ready (may take ~1 month more) – if you want to check it out earlier, let us know!

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

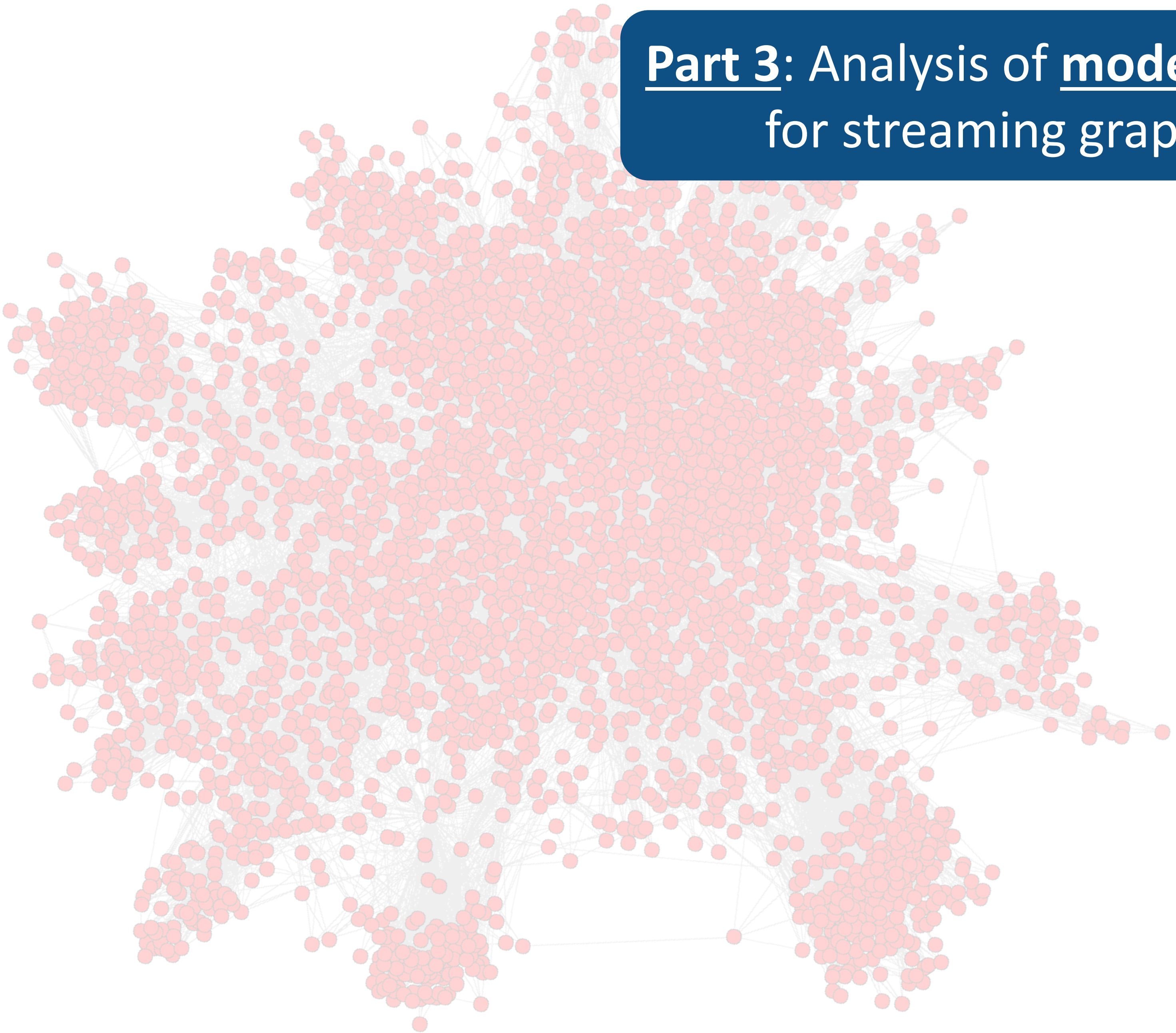
Towards Understanding of Modern Graph Processin

MARC FISCHER, Depart
MACIEJ BESTA,
TAL BEN-NUN,
TORSTEN HOEF

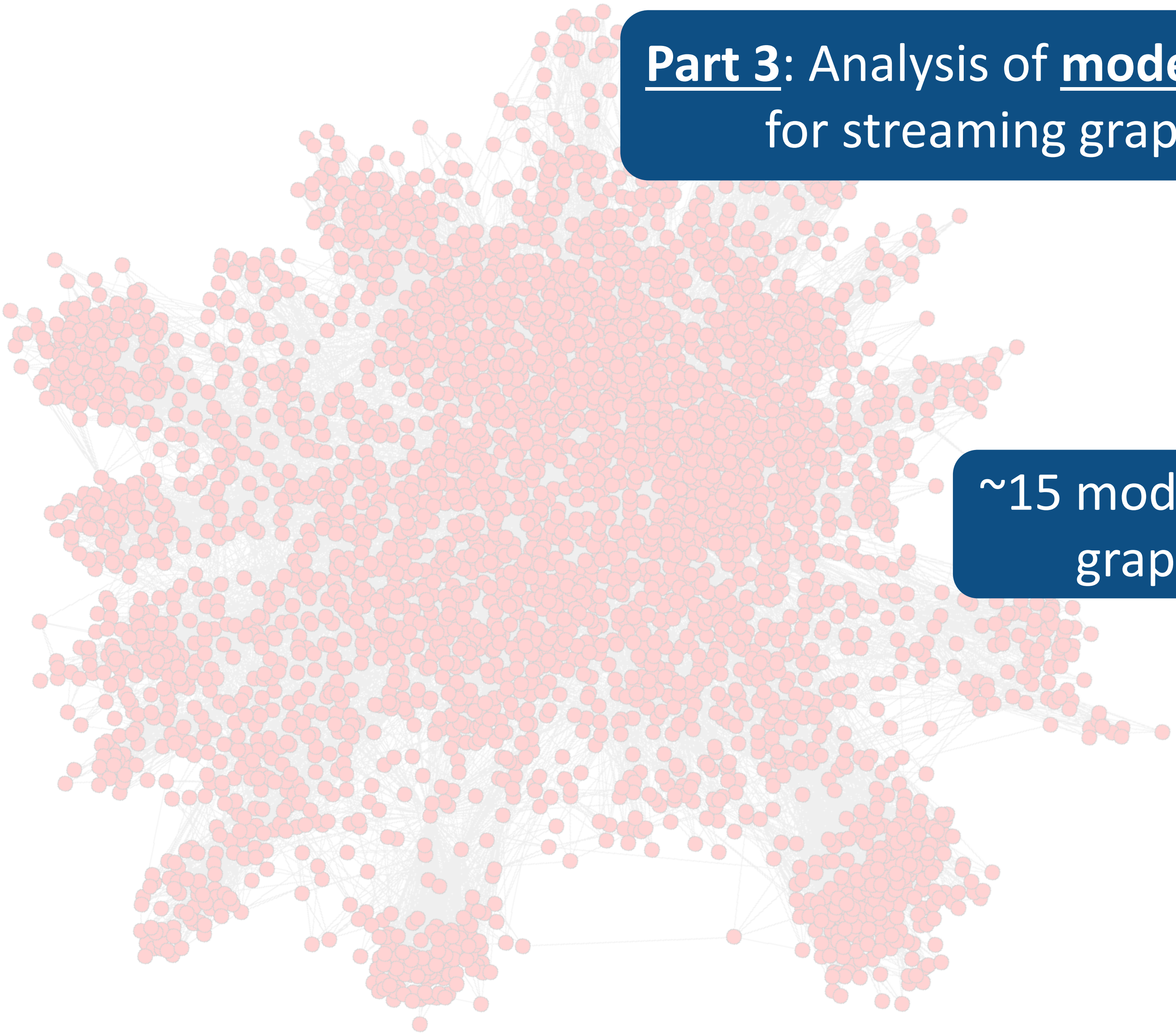
Graph processing has … machine learn-
ing, social network ana… that hinder accelerating
graph processing are (1) … and (2) the growing rate of graph
updates, with millions o… Graph streaming algorithms are specifically
crafted to eliminate these … passed as a stream of updates, allowing to add and remove
edges in a simple way. Rece… seen the development of many such algorithms. However, they differ

Part 3: Analysis of **models** and algorithms for streaming graph processing

<u>Part 3</u>: Analysis of **<u>models</u>** and algorithms for streaming graph processing

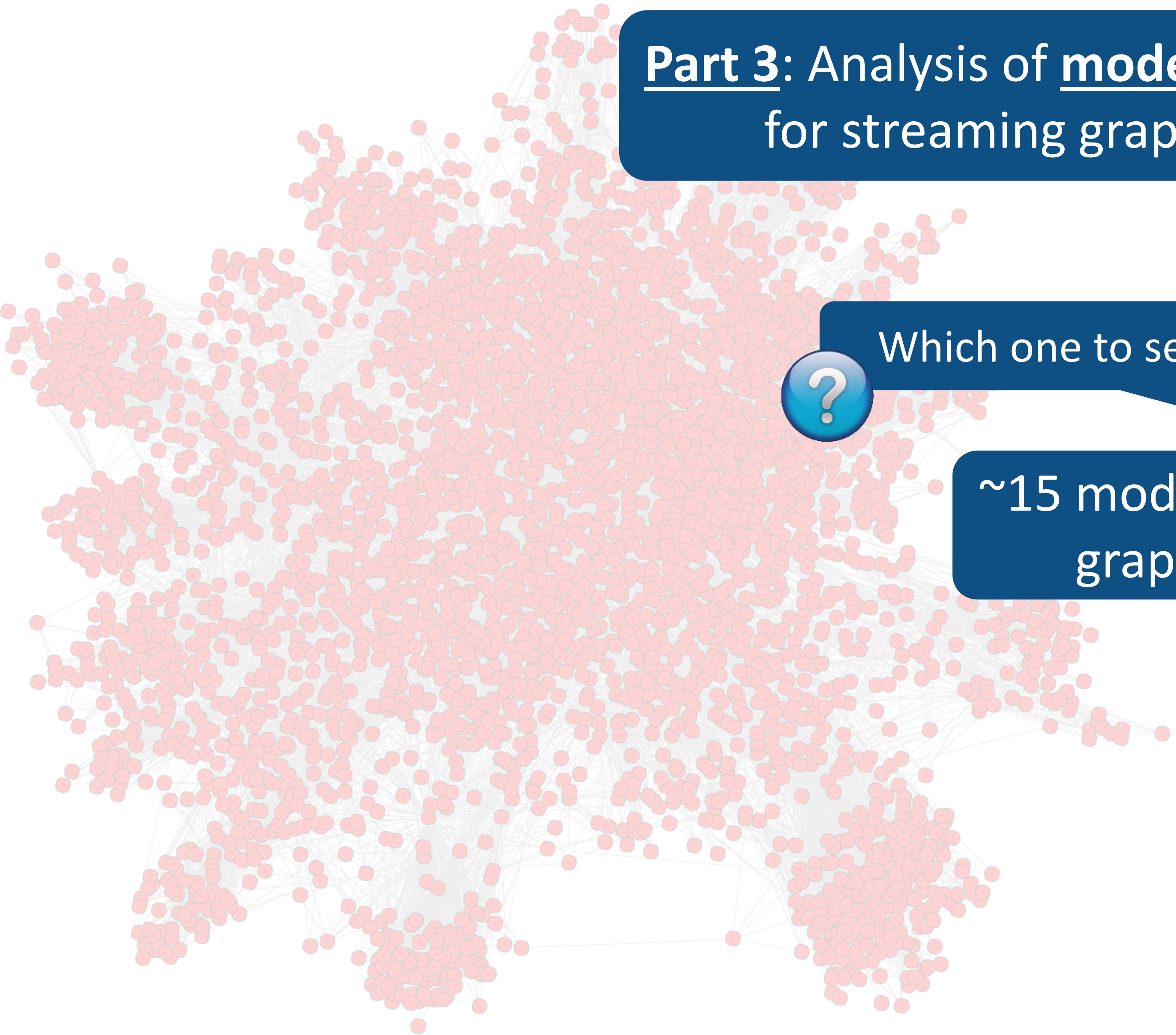


Part 3: Analysis of **models** and algorithms for streaming graph processing



~15 models for streaming graph processing

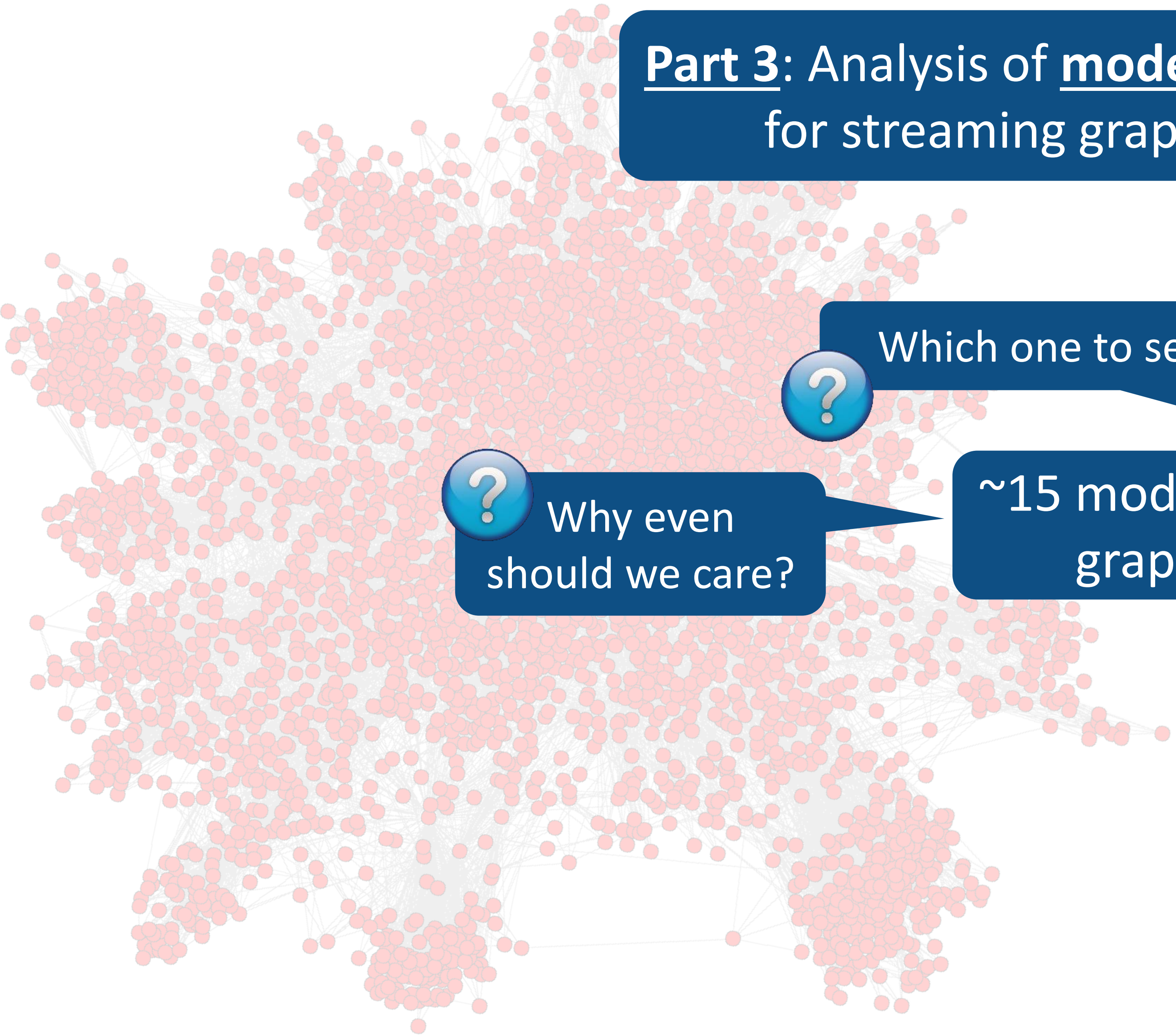
Part 3: Analysis of **models** and algorithms for streaming graph processing



Which one to select?

~15 models for streaming graph processing

<u>Part 3</u>: Analysis of **<u>models</u>** and algorithms for streaming graph processing

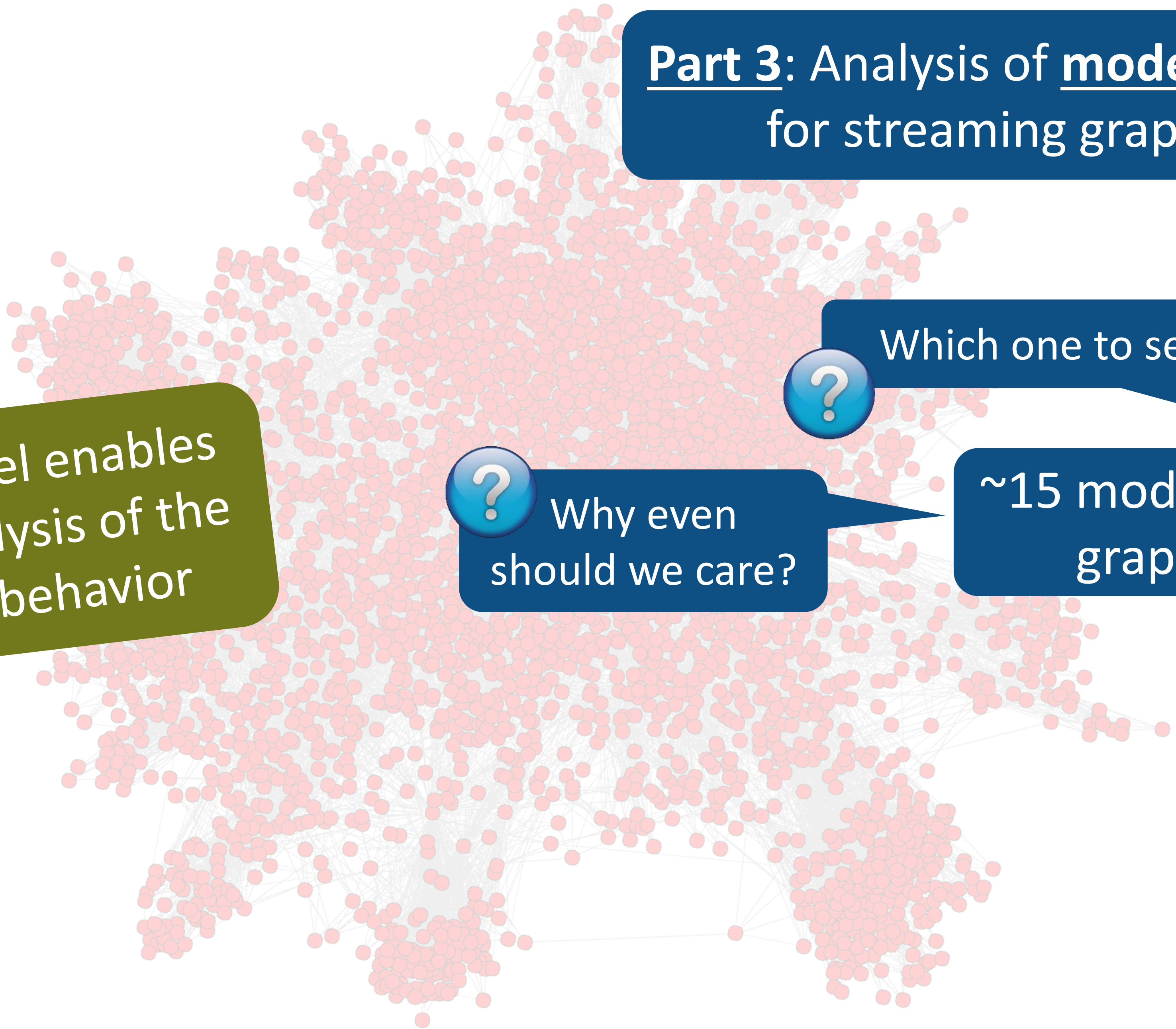


Which one to select?

~15 models for streaming graph processing

Why even should we care?

<u>Part 3</u>: Analysis of **<u>models</u>** and algorithms for streaming graph processing



Which one to select?

Why even should we care?

~15 models for streaming graph processing

Using a model enables rigorous analysis of the algorithm behavior

Part 3: Analysis of **models** and algorithms for streaming graph processing

Which one to select?

~15 models for streaming graph processing

Why even should we care?

Using a model enables rigorous analysis of the algorithm behavior

Selecting a right model enables more realistic predictions about the algorithm behavior on given hardware

Part 3: Analysis of **models** and algorithms for streaming graph processing

Which one to select?

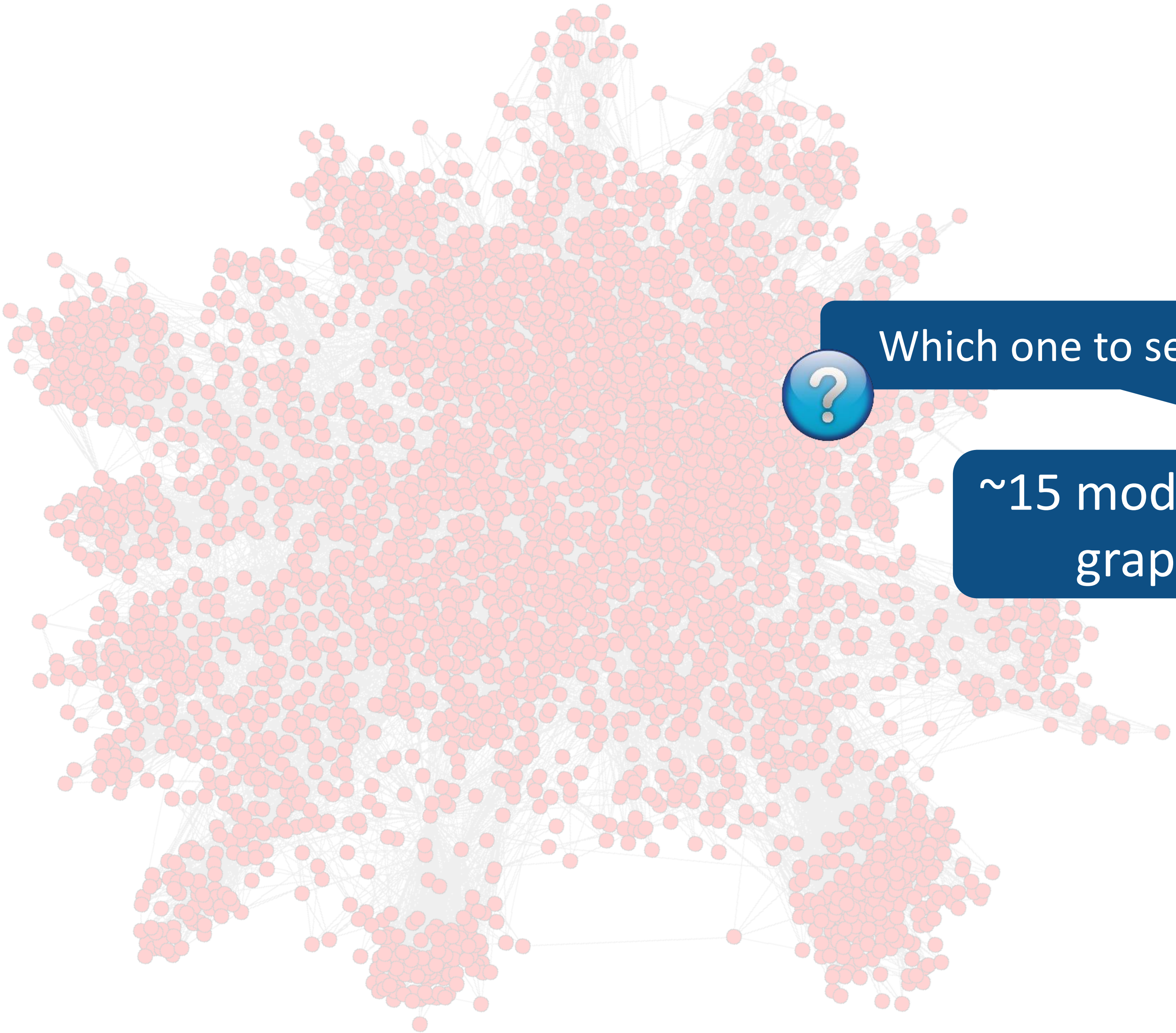
Why even should we care?

~15 models for streaming graph processing

Using a model enables rigorous analysis of the algorithm behavior

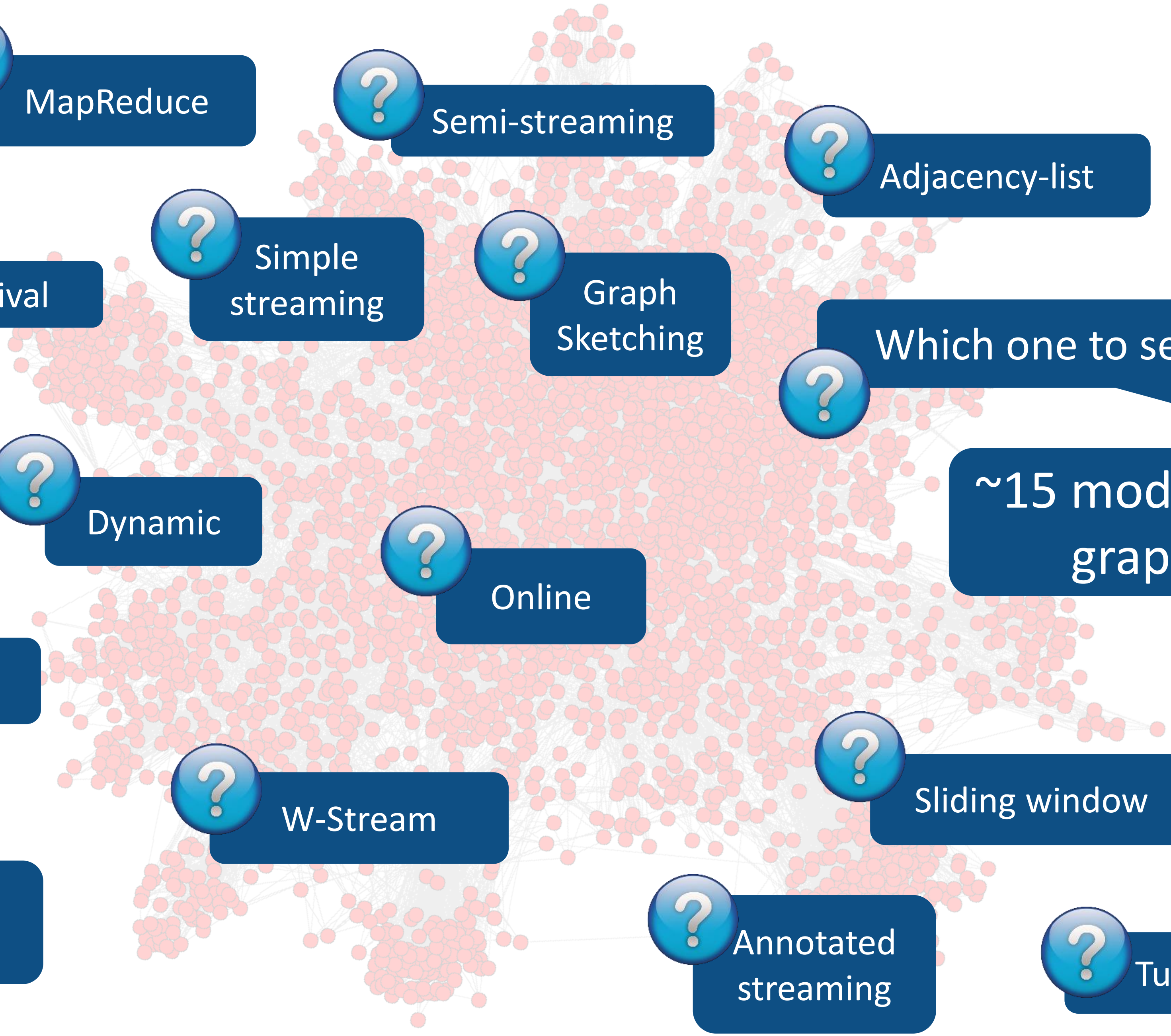
Limiting oneself to a particular model helps to select the best algorithm or technique (within that model)

Selecting a right model enables more realistic predictions about the algorithm behavior on given hardware



Which one to select?

~15 models for streaming graph processing



MUD

MapReduce

Semi-streaming

Insert-only

Adjacency-list

Vertex-arrival

Simple streaming

Graph Sketching

Which one to select?

~15 models for streaming graph processing

Dynamic

Online

StreamSort

Sliding window

W-Stream

Cash-register

Annotated streaming

Turnstile

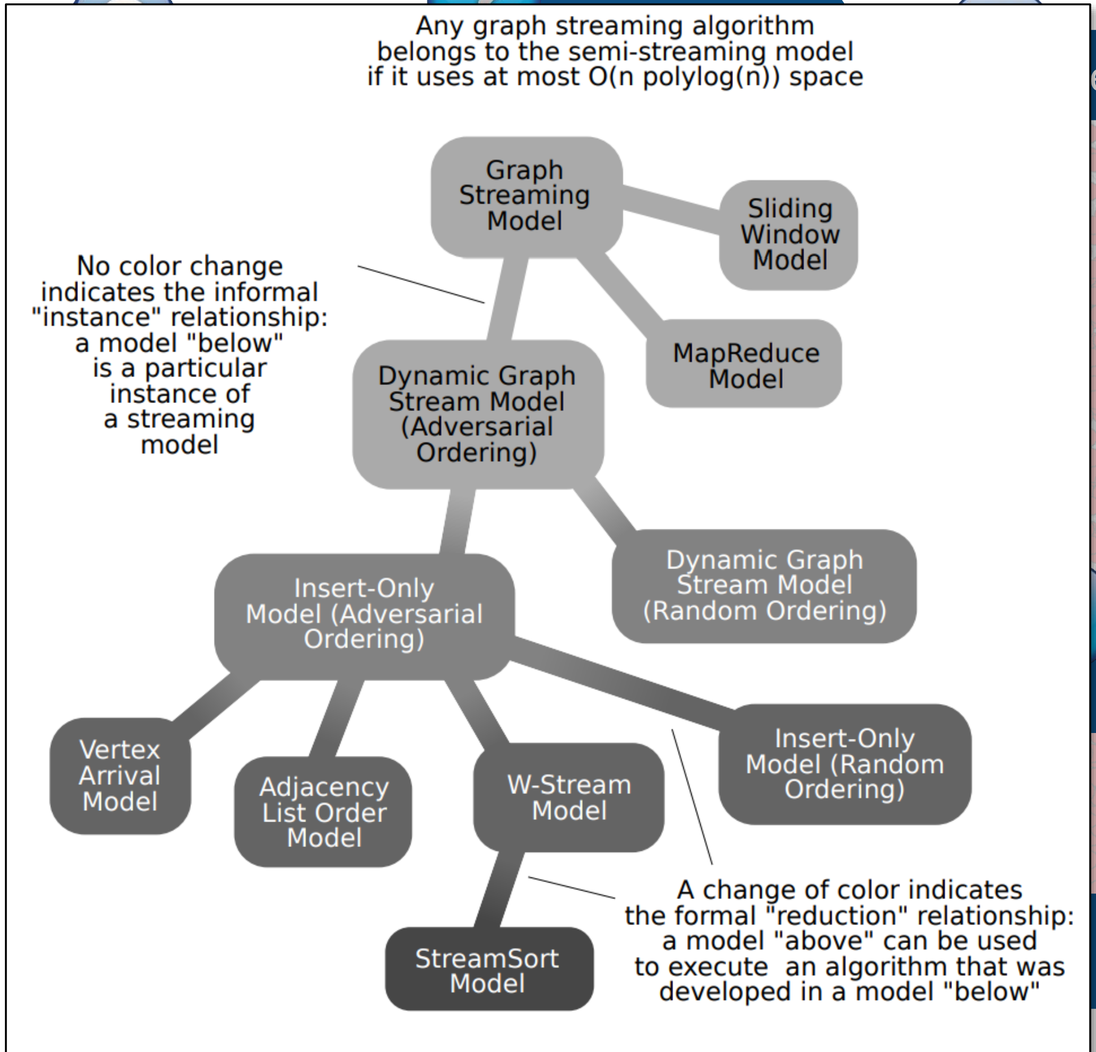


Fig. 1. The hierarchy of the graph streaming models.

emi-streaming

Insert-only

Adjacency-list

Graph Sketching

Which one to select?

~15 models for streaming graph processing

Online

Sliding window

register

Annotated streaming

Turnstile

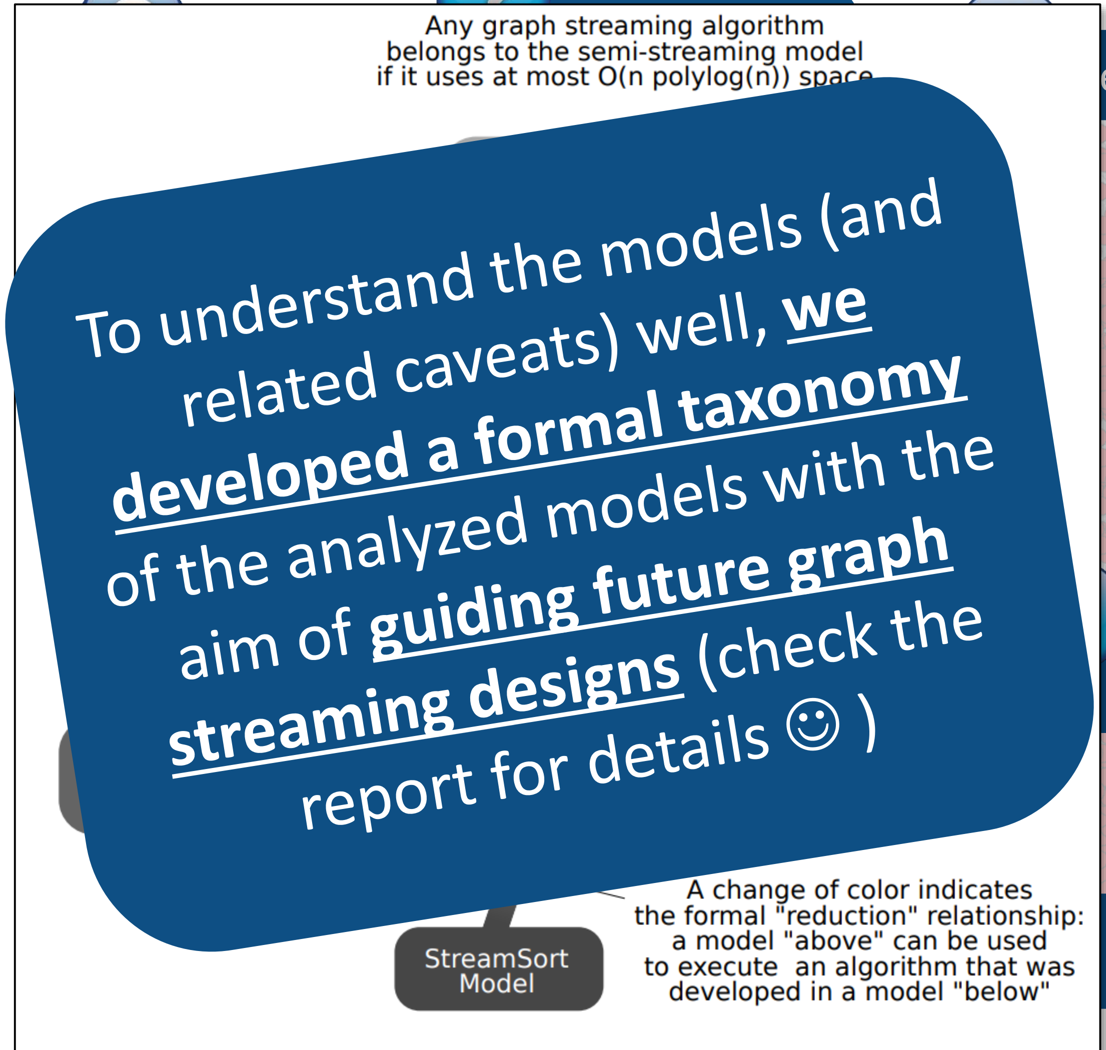
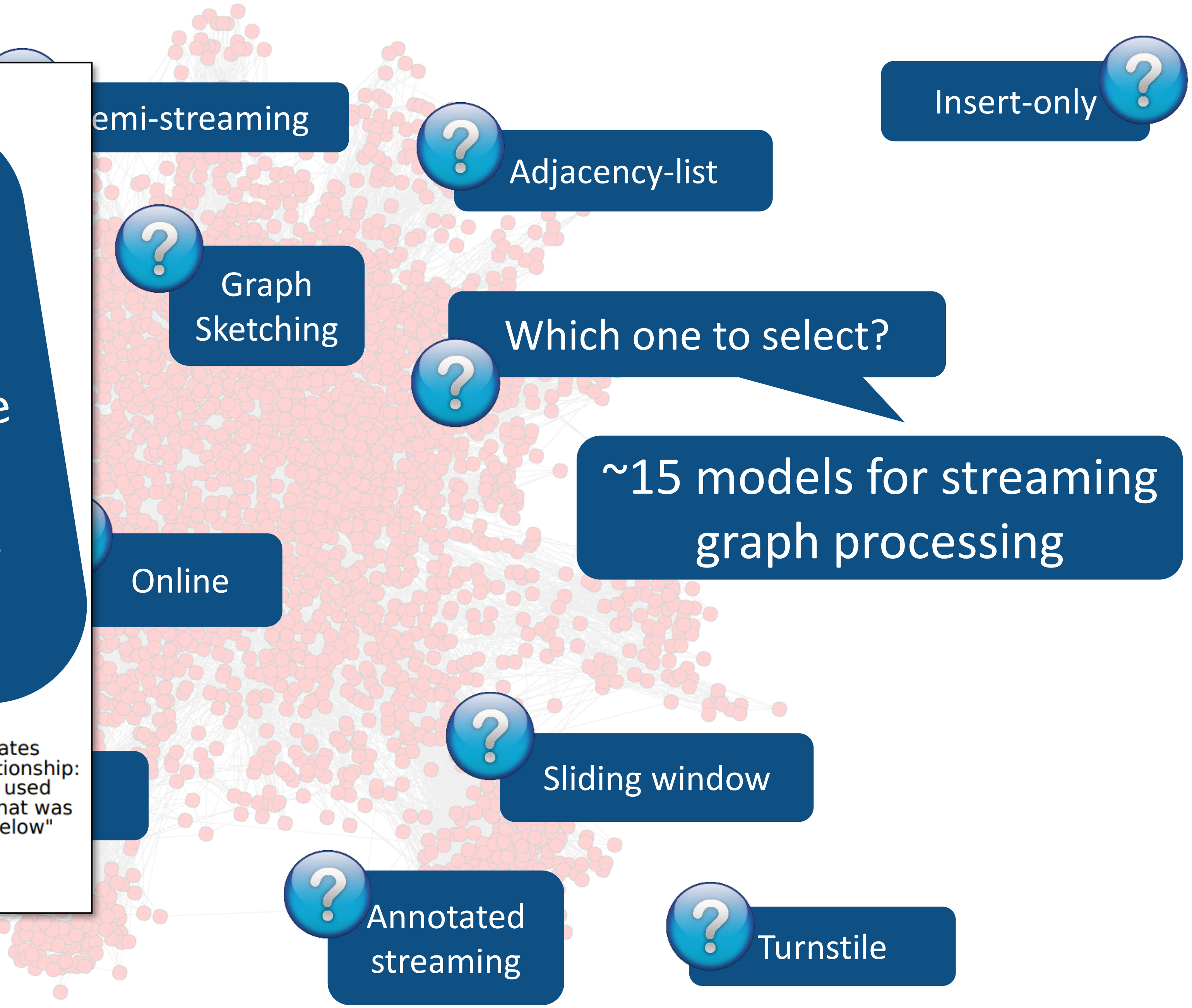
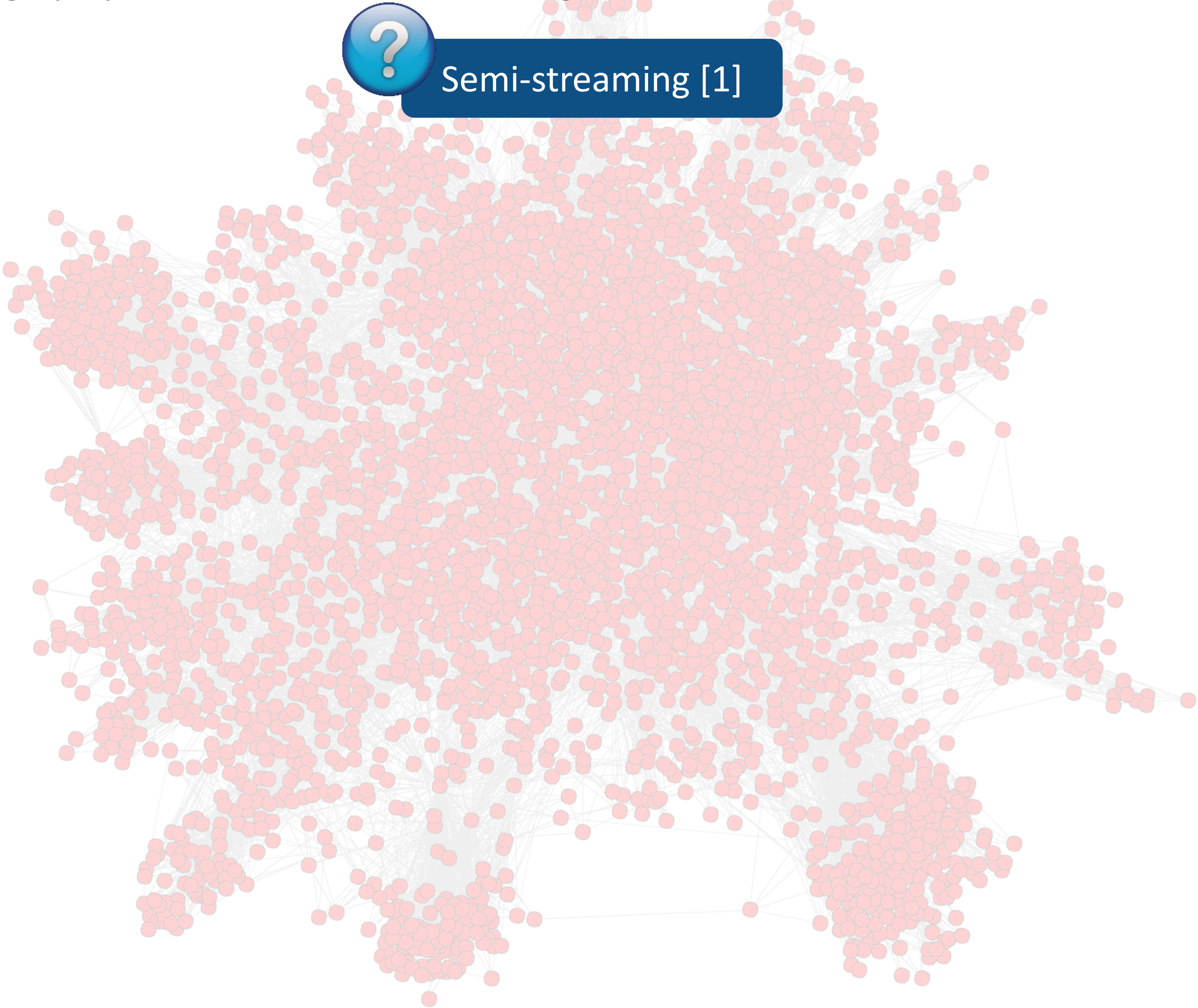


Fig. 1. The hierarchy of the graph streaming models.

To understand the models (and related caveats) well, **we developed a formal taxonomy** of the analyzed models with the aim of **guiding future graph streaming designs** (check the report for details ☺)



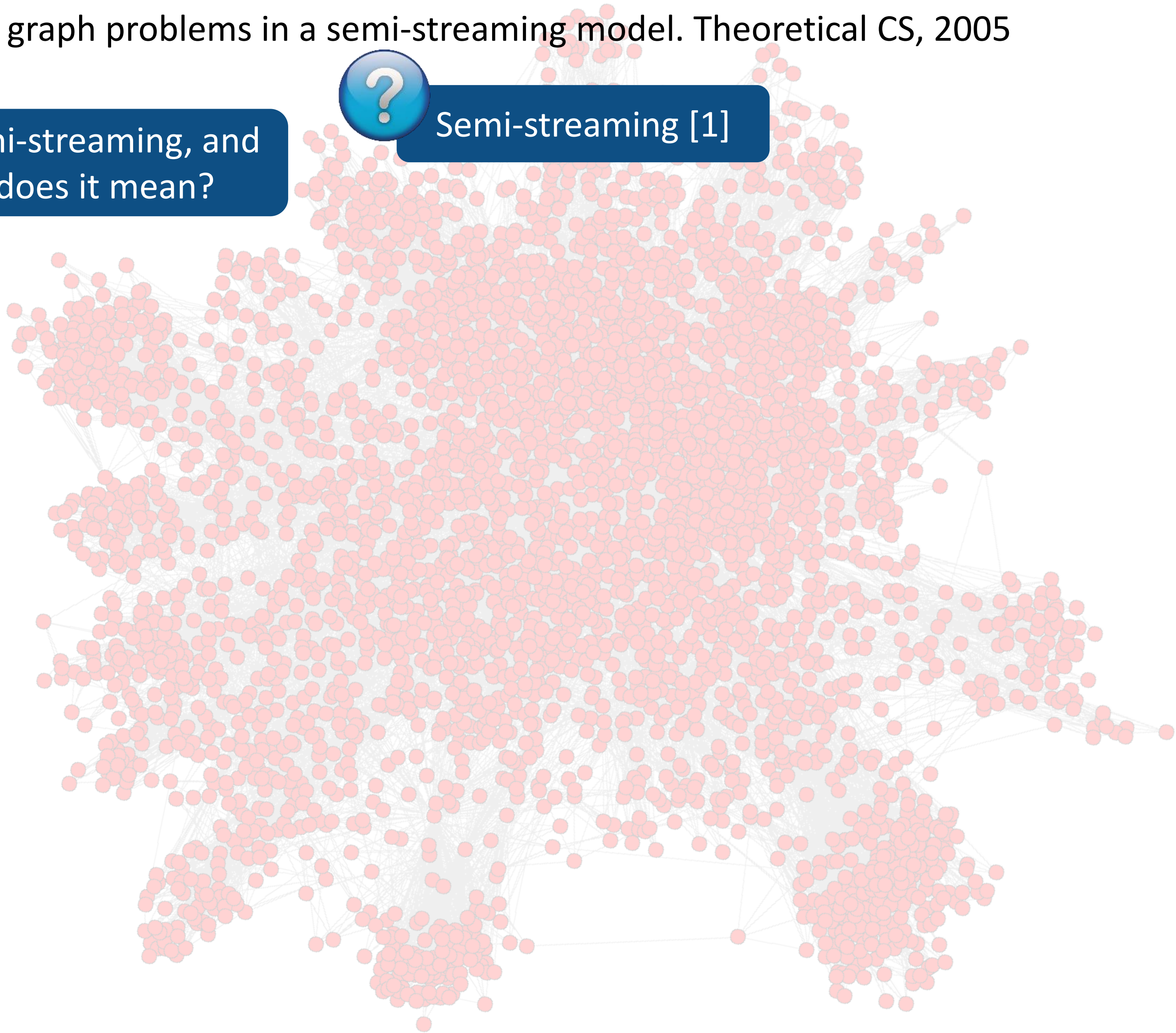
[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005



[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Semi-streaming [1]

Why semi-streaming, and what does it mean?

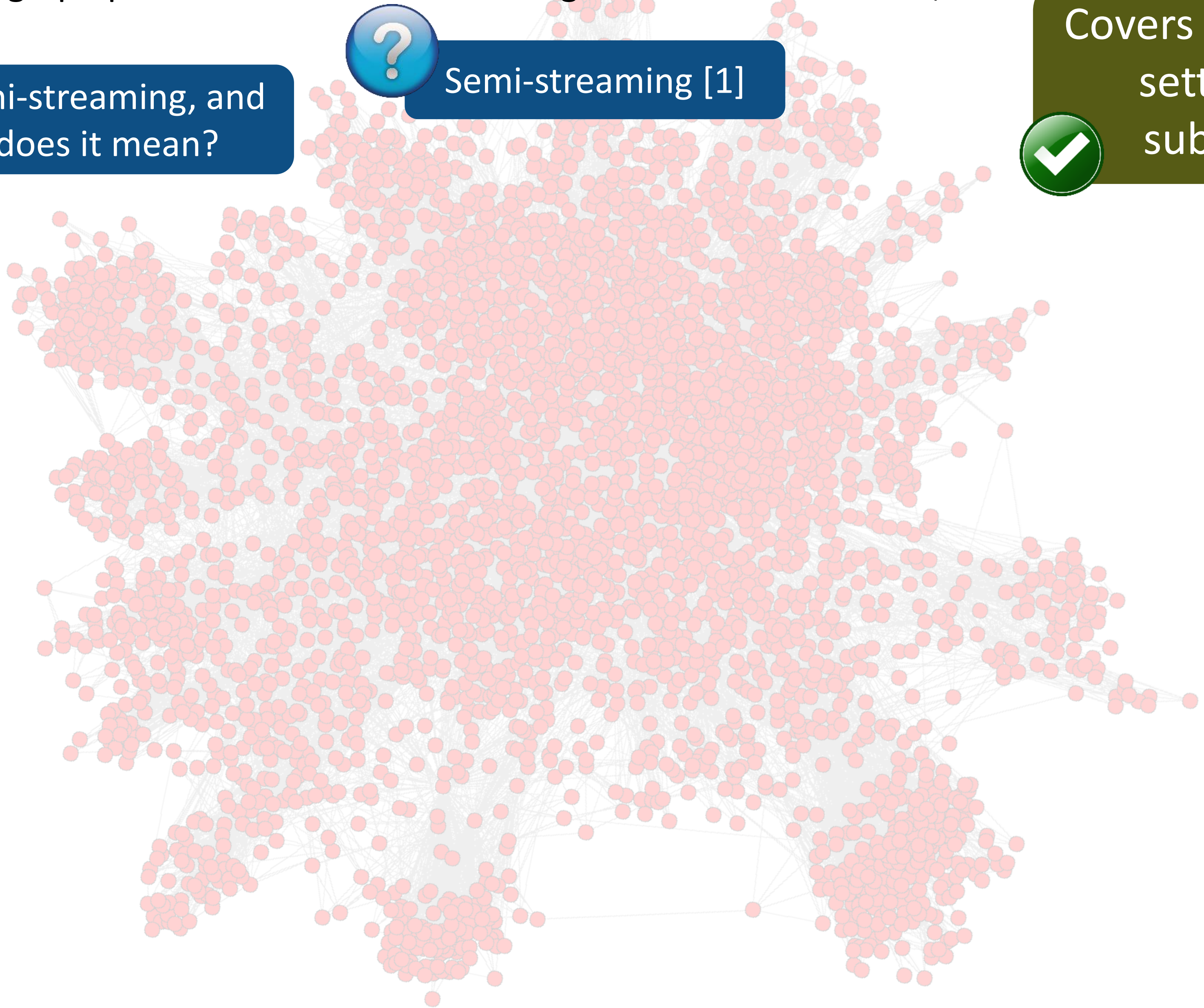


[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Why semi-streaming, and what does it mean?

Semi-streaming [1]

Covers a general streaming setting (= works for substream-centric)

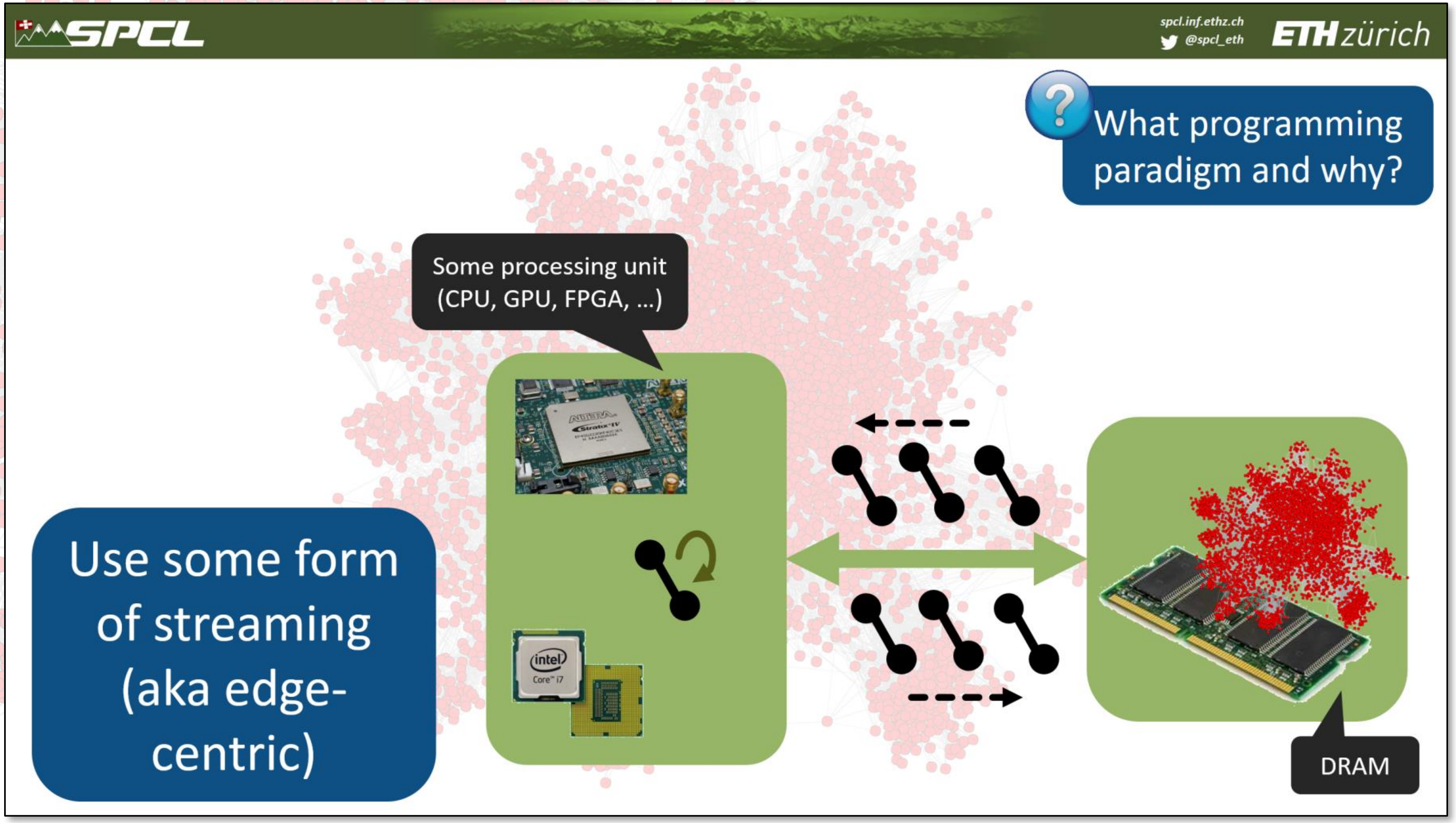


[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Why semi-streaming, and what does it mean?

Semi-streaming [1]

Covers a general streaming setting (= works for substream-centric)



[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Why semi-streaming, and what does it mean?

Semi-streaming [1]

Covers a general streaming setting (= works for substream-centric)

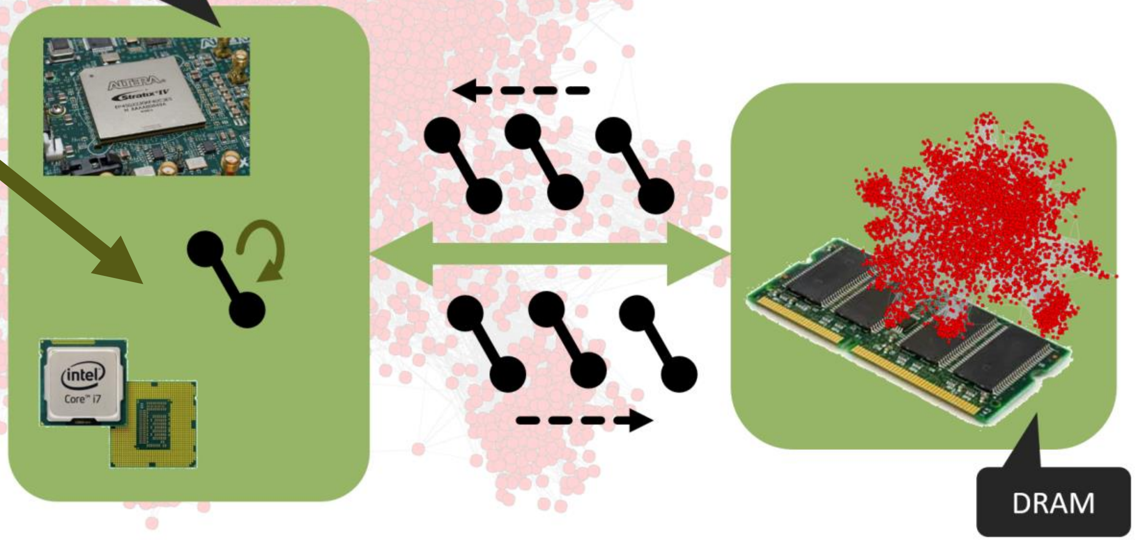
Assumes $O(n \log^c n)$ local space that can be used for processing an edge → fits well FPGA BRAM constraints!

spcl.inf.ethz.ch
@spcl_eth

What programming paradigm and why?

Some processing unit (CPU, GPU, FPGA, ...)

Use some form of streaming (aka edge-centric)



[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

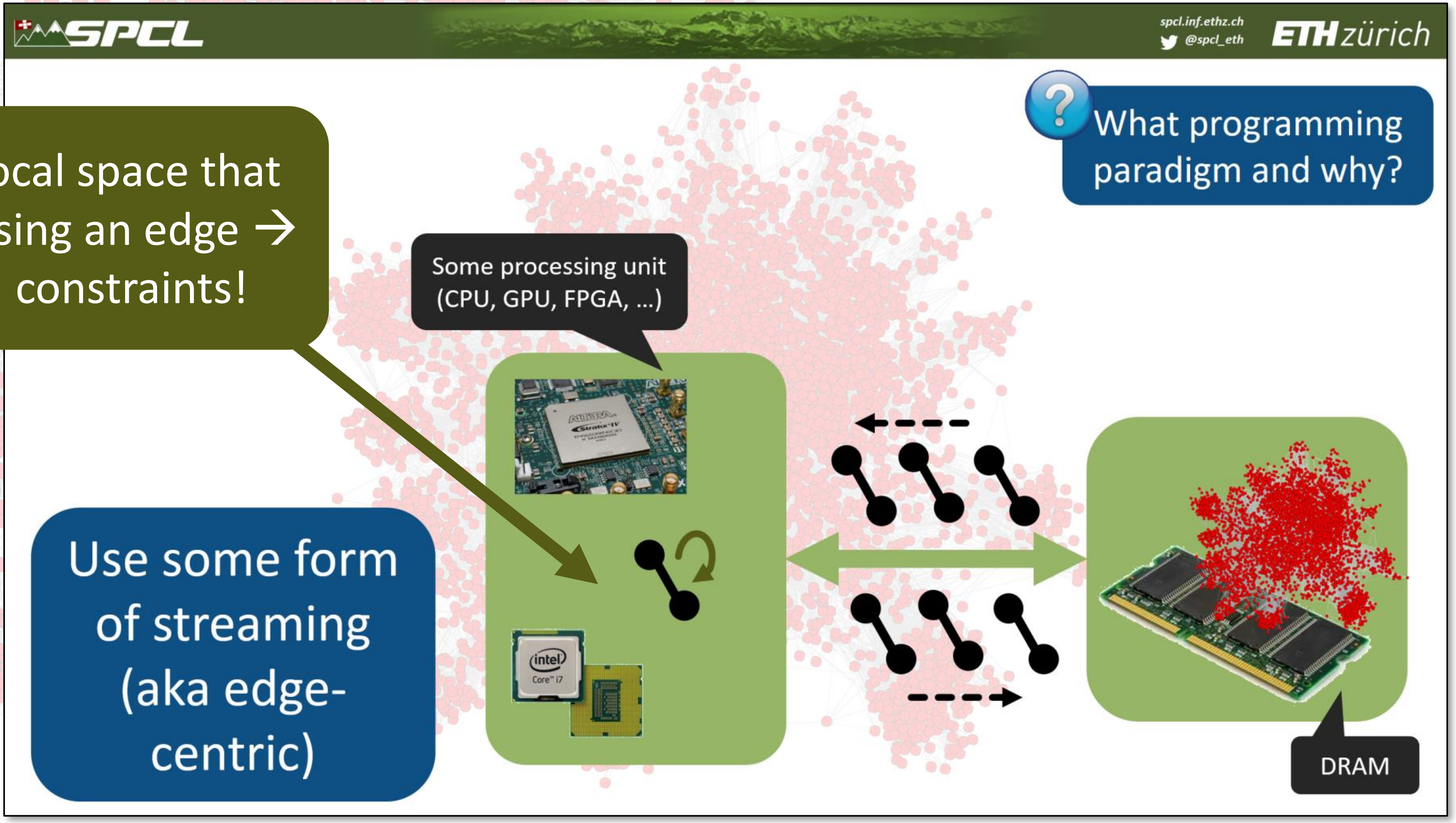
Why semi-streaming, and what does it mean?

Semi-streaming [1]

Covers a general streaming setting (= works for substream-centric)

Assumes $O(n \log^c n)$ local space that can be used for processing an edge → fits well FPGA BRAM constraints!

Offers (potentially powerful) MWM algorithms



Research Questions

Which programming paradigm to use for (approximate) MWM?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

...ming
...digm to use for
(approximate) MWM?

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

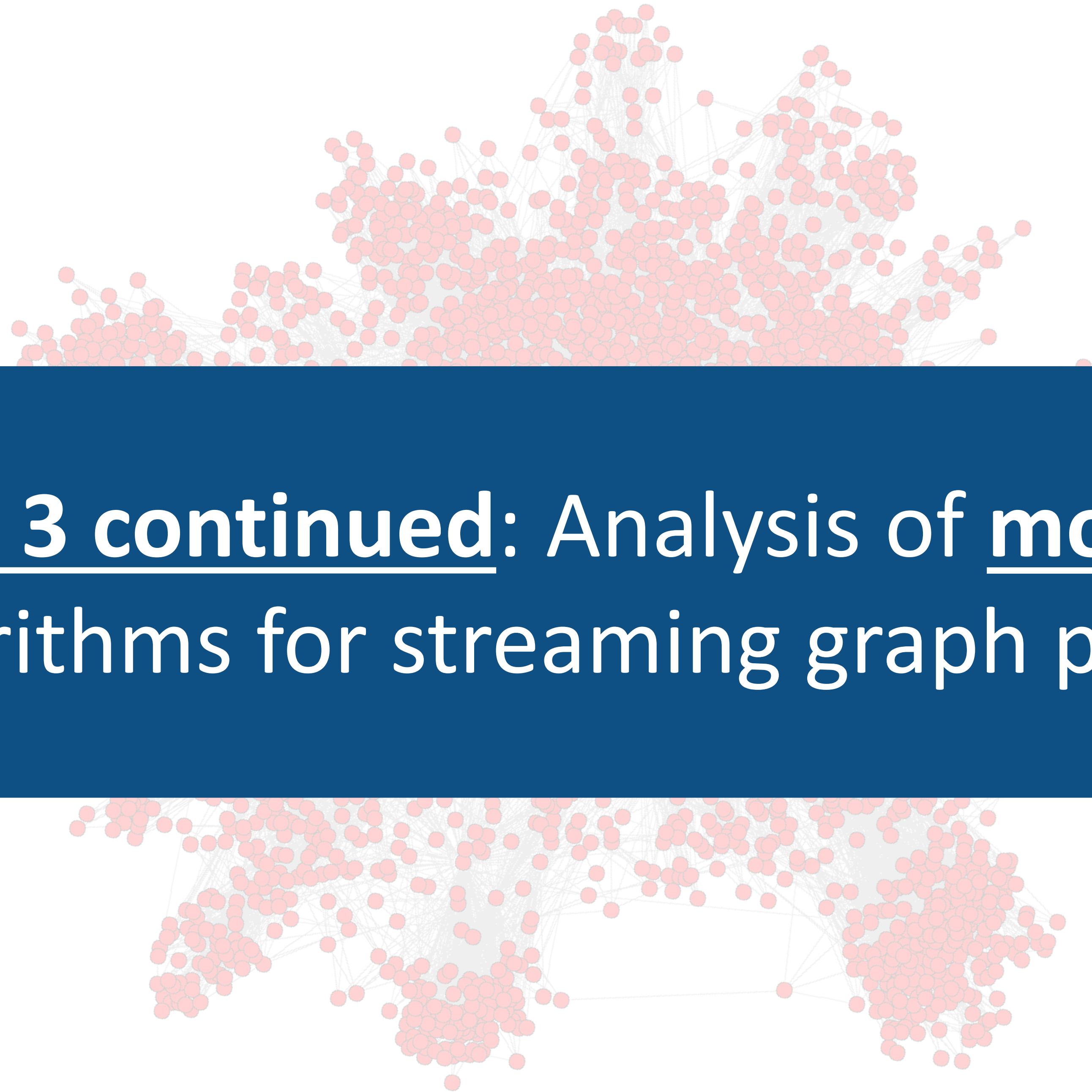
Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

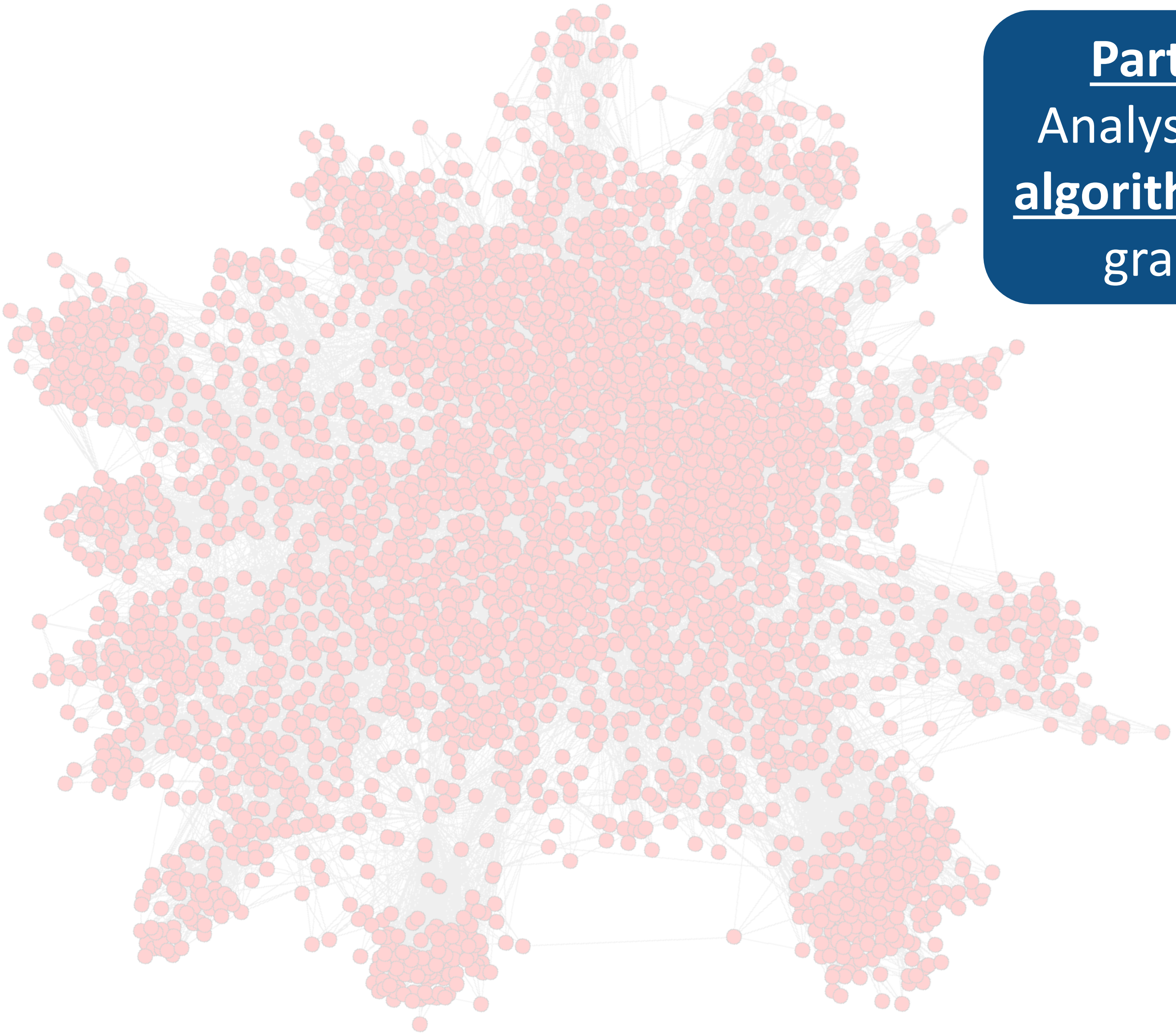
What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

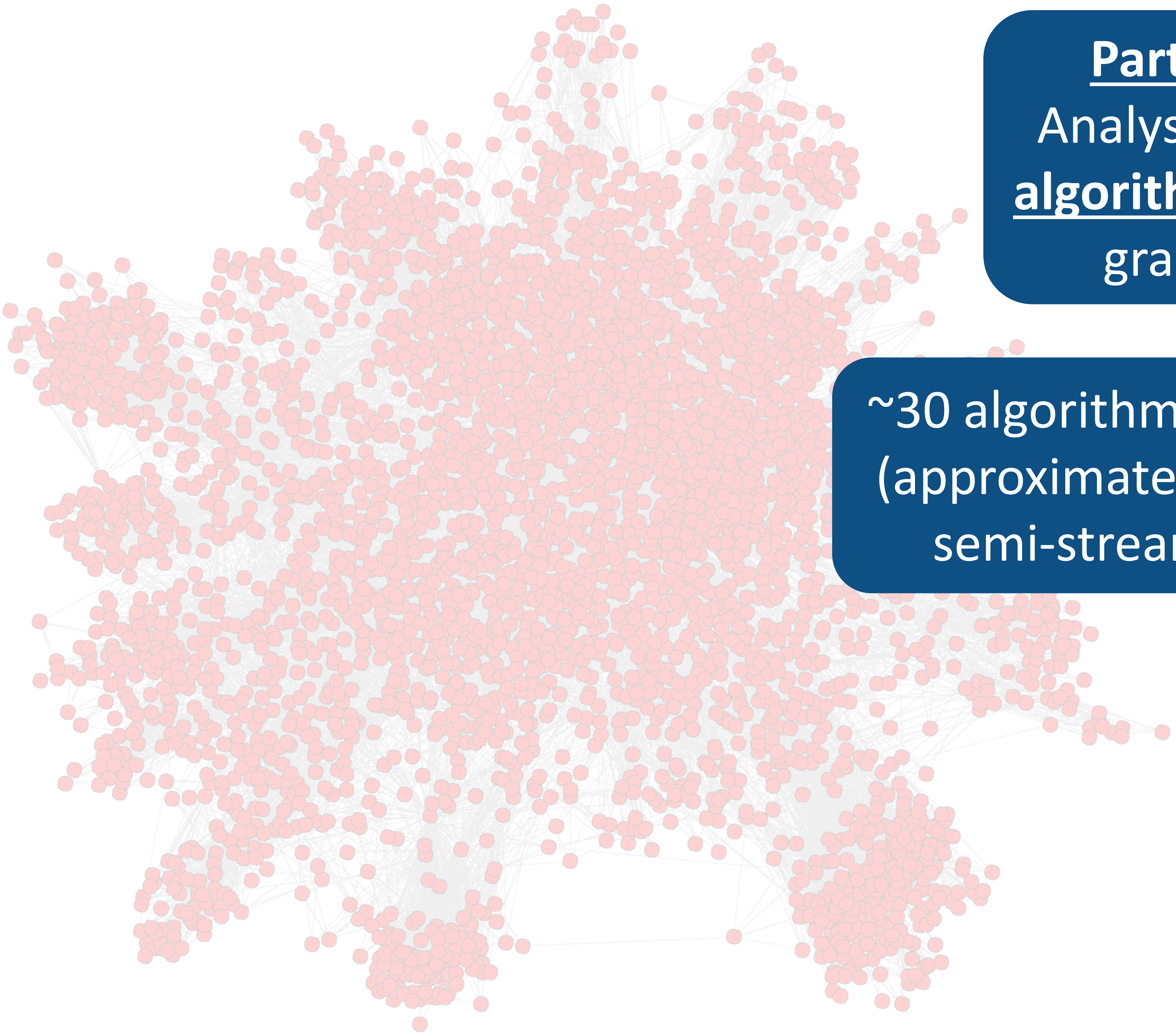


<u>Part 3 continued</u>: Analysis of **<u>models</u>** and algorithms for streaming graph processing

Part 3 continued: Analysis of models and **algorithms** for streaming graph processing

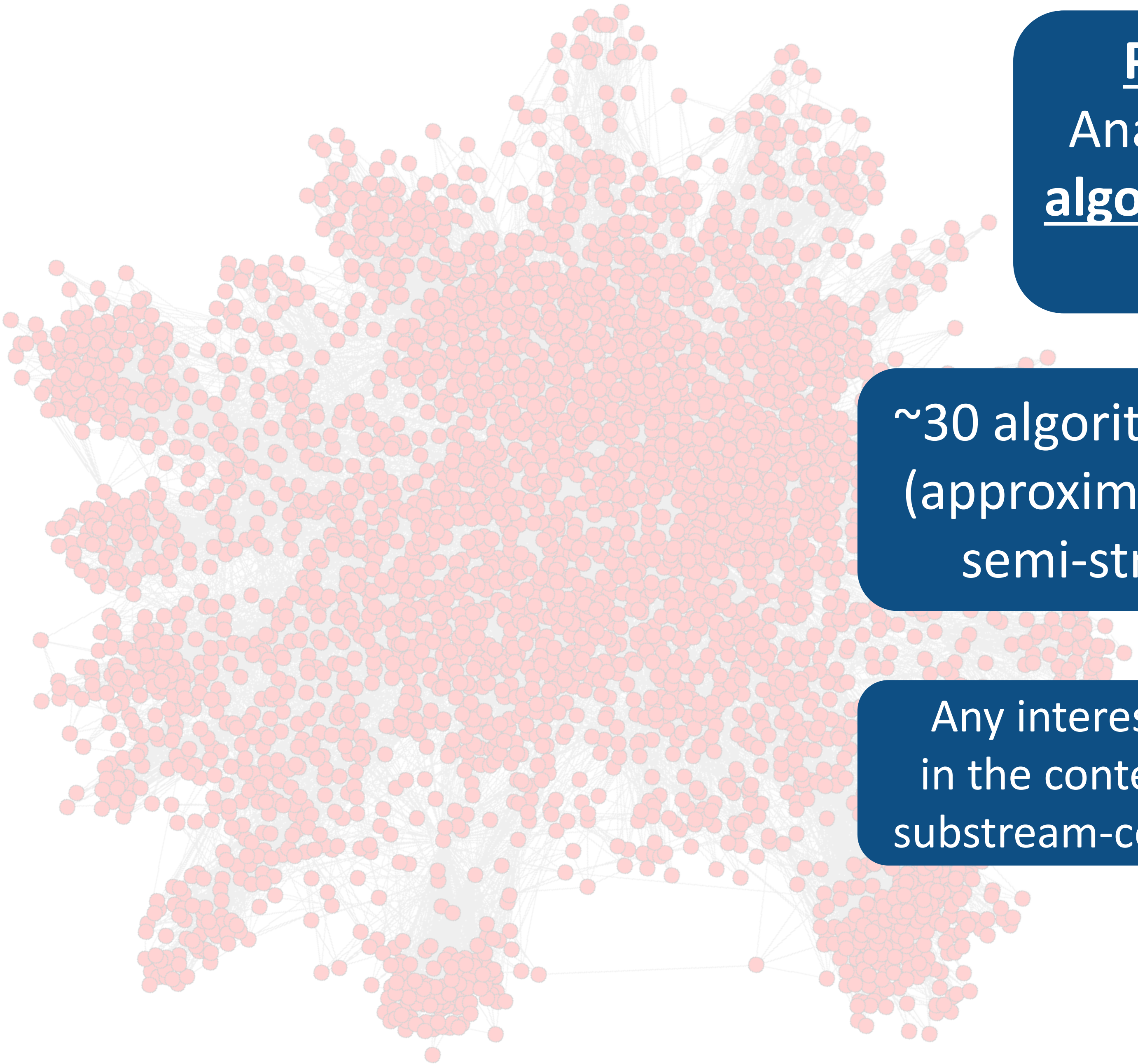


Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing



<u>Part 3 continued</u> :
Analysis of models and **<u>algorithms</u>** for streaming graph processing

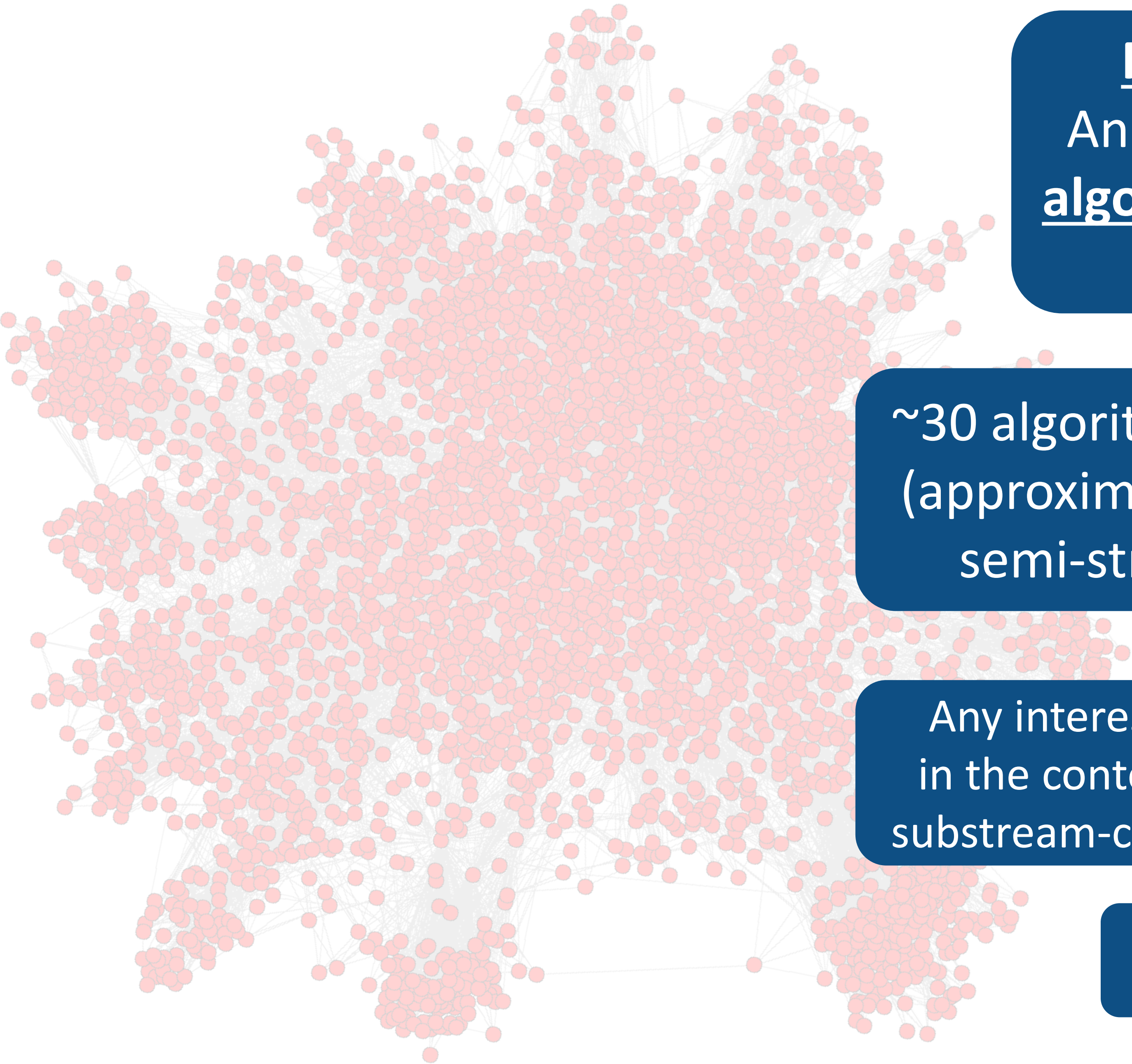
~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[26]	$1/2$	$O(n)$	1	👎	👍	👍
[41, Theorem 6]	$1/2 + 0.0071$	$O(n \text{ polylog}(n))$	2	👎	👍	👍
[41, Theorem 2]	$1/2 + 0.003^{*}$	$O(n \text{ polylog}(n))$	1	👎	👍	👍
[36, Theorem 1.1]	$O(\text{polylog}(n))$	$O(\text{polylog}(n))$	1	👎	👍	👍
[26, Theorem 1]	$2/3 - \varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	$1 - \varepsilon$	$O\left(n \text{ polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[41, Theorem 5]	$1/2 + 0.019$	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2 + 0.005^{*}$	$O(n \log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2 + 0.0071^{*}$	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[39]	$1 - 1/e$	$O(n \text{ polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1 - 1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon} + n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2 + \varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	?
[25]	$4.911 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[29]	$3.5 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[53]	$2 + \varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2 + \varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n \log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4} \log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
[17]	$4 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	👍

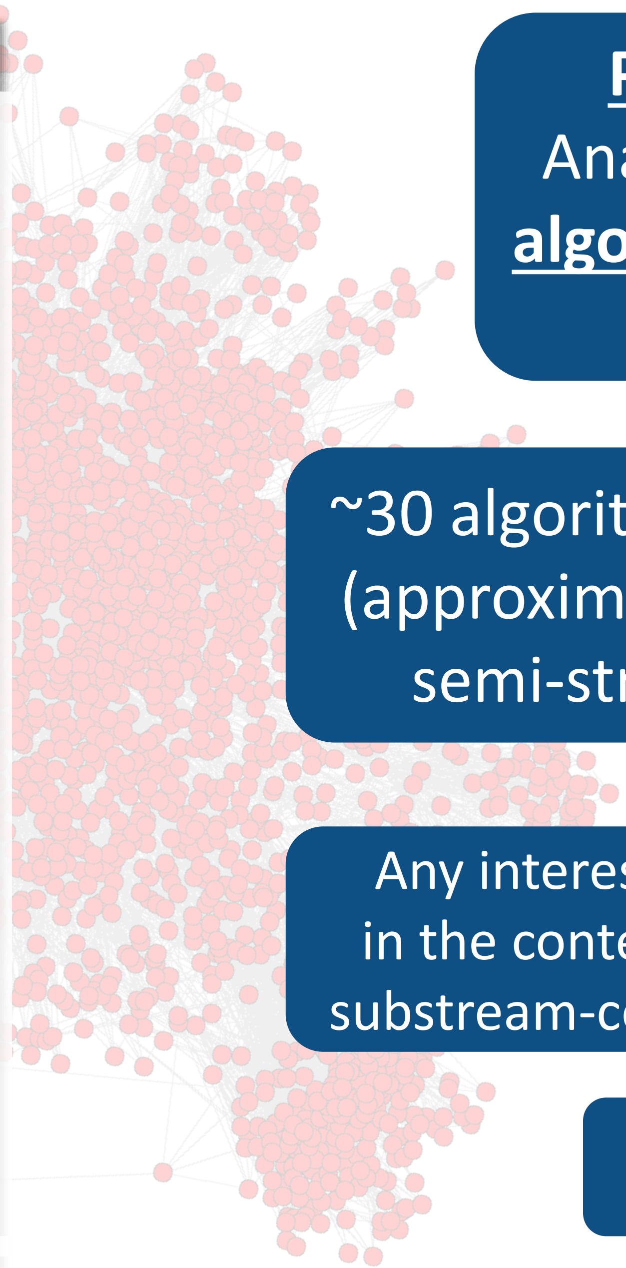
Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[26]	$1/2$	$O(n)$	1	👎	👍	👍
[41, Theorem 6]	$1/2+0.0071$	$O(n \text{ polylog}(n))$	2	👎	👍	👍
[41, Theorem 2]	$1/2+0.003^*$	$O(n \text{ polylog}(n))$	1	👎	👍	👍
[36, Theorem 1.1]	$O(\text{polylog}(n))$	$O(\text{polylog}(n))$	1	👎	👍	👍
[26, Theorem 1]	$2/3-\varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	$1-\varepsilon$	$O\left(n \text{ polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[41, Theorem 5]	$1/2+0.019$	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2+0.005^*$	$O(n \log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2+0.0071^*$	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[39]	$1-1/e$	$O(n \text{ polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1-\frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n \log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4} \log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
[17]	$4+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	👍



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
	$1/2$	$O(n)$	1	👎	👍	👍
	$1/2+0.0071$	$O(n\ \mathrm{polylog}(n))$	2	👎	👍	👍
	$1/2+0.003^*$	$O(n\ \mathrm{polylog}(n))$	1	👎	👍	👍
1]	$O(\mathrm{polylog}(n))$	$O(\mathrm{polylog}(n))$	1	👎	👍	👍
]	$2/3-\varepsilon$	$O(n\log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	$1-\varepsilon$	$O(n\ \mathrm{polylog}(n)/\varepsilon^2)$	$O(\log\log(1/\varepsilon)/\varepsilon^2)$	👎	👎	👍
[41, Theorem 5]	$1/2+0.019$	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[41, Theorem 1]	$1/2+0.005^*$	$O(n\log n)$	1	👎	👎	👍
[41, Theorem 4]	$1/2+0.0071^*$	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[39]	$1-1/e$	$O(n\ \mathrm{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1-\frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}(k^2)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}(n^2/\varepsilon^3)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}(n^{2-3\varepsilon}+n^{1-\varepsilon})$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n\log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O(n\log^2 n)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n\log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n\log n)$	$O(\log_{1+\varepsilon/3} n)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O(n\log(n)/\varepsilon^4)$	$O(\varepsilon^{-4}\log n)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	👍	👎	👍
[17]	$4+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	👍

Our goals:

Part 3 continued : Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

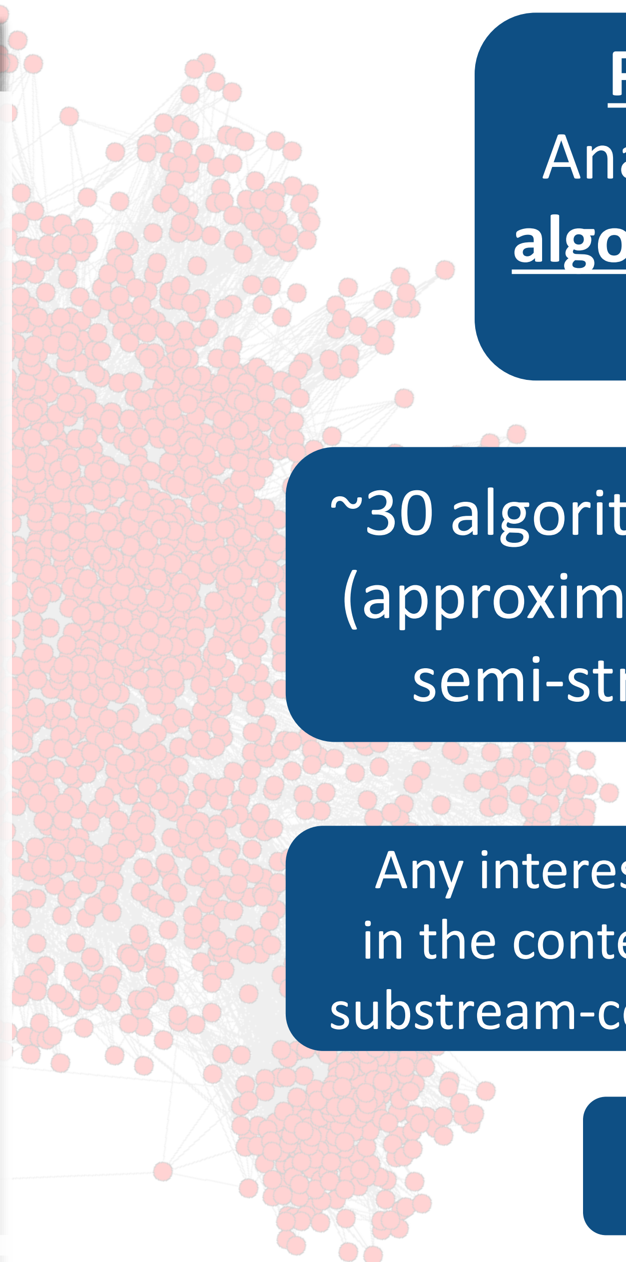
Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	$1/2$	$O(n)$	1	👎	👍	👍
[illegible]	$1/2 + 0.0071$	$O(n \text{ polylog}(n))$	2	👎	👍	👍
[illegible]	$1/2 + 0.003^*$	$O(n \text{ polylog}(n))$	1	👎	👍	👍
[illegible]	$O(\text{polylog}(n))$	$O(\text{polylog}(n))$	1	👎	👍	👍
[illegible]	[illegible] ε	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	[illegible]	$O\left(n \text{ polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log(1/\varepsilon)/\varepsilon^2\right)$	👎	👎	👍
[4[illegible]	[illegible]9	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[4[illegible]	[illegible]5*	$O(n \log n)$	1	👎	👎	👍
[4[illegible]	[illegible]71*	$O(n \text{ polylog}(n))$	2	👎	👎	👍
[39]	[illegible]	$O(n \text{ polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1 - 1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	$O(n)$	k	👎	👎	👍
[14]	1	$\tilde{O}\left(k^2\right)$	1	👎	👍	👍
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon} + n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2 + \varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	?
[25]	$4.911 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[29]	$3.5 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[53]	$2 + \varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2 + \varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n \log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4} \log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
[17]	$4 + \varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	👍

Our goals:

Maximize accuracy



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

SPCL spcl.inf.ethz.ch @spcl_eth ETH zürich

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	$1/2$	$O(n)$	1	👎	👍	👍
[illegible]	$1/2+0.0071$	$O(n\ \mathrm{polylog}(n))$	2	👎	👍	👍
[illegible]	$1/2+0.003^{*}$	$O(n\ \mathrm{polylog}(n))$	1	👎	👍	👍
[illegible]	$O(\mathrm{polylog}(n))$	$O(\mathrm{polylog}(n))$	1	👎	👍	👍
[illegible]	$[illegible]+\varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	[illegible]	$O(n\ \mathrm{polylog}(n)/\varepsilon^2)$	$O(\log\log(1/\varepsilon)/\varepsilon^2)$	👎	👎	👍
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[illegible]	[illegible]	$O(n \log n)$	1	👎	👎	👍
[illegible]	[illegible]	$O(n\ \mathrm{polylog}(n))$	2	👎	👎	👍
[39]	[illegible]	$O(n\ \mathrm{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1-1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	[illegible]	[illegible]	k	👎	👎	👍
[14]	[illegible]	[illegible]	1	👎	👍	👍
[14]	$1/\varepsilon$	$O(n^2/\varepsilon^3)$	1	👎	👍	👍
[7, Theorem 1]	n^{ε}	$\tilde{O}(n^{2-3\varepsilon}+n^{1-\varepsilon})$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O(n \log^2 n)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O(\log_{1+\varepsilon/3} n)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O(n \log(n)/\varepsilon^4)$	$O(\varepsilon^{-4} \log n)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	👍	👎	👍
[17]	$4+\varepsilon$	$O(n\ \mathrm{polylog}(n))$	1	👍	👍	👍

Our goals:

Maximize accuracy

Minimize local space

Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

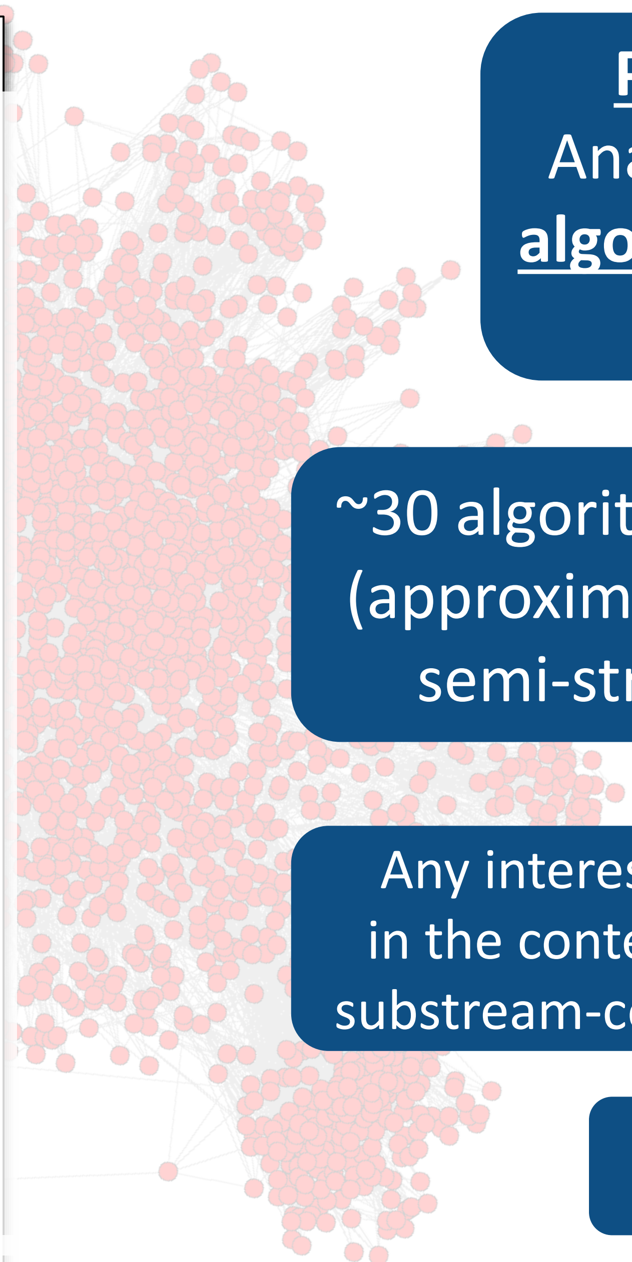
Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	$1/2$	$O(n)$	[illegible]	👎	👍	👍
[illegible]	$1/2 + 0.0071$	$O(n\,\mathrm{pol}$[illegible]	[illegible]	👎	👍	👍
[illegible]	$1/2 + 0.003^*$	$O(n\,\mathrm{pol}$[illegible]	[illegible]	👎	👍	👍
[illegible]1]	$O(\mathrm{polylog}(n))$	$O(\mathrm{polyl}$[illegible]	[illegible]	👎	👍	👍
[illegible]	[illegible] ε	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]	[illegible]	$O(n\,\mathrm{polylog}(n)/\varepsilon^2)$	$O(\log\log(1/\varepsilon)/\varepsilon^2)$	👎	👎	👍
[4[illegible]	[illegible]	$O(n\,\mathrm{polylog}(n))$	2	👎	👎	👍
[4[illegible]	[illegible]	$O(n \log n)$	1	👎	👎	👍
[4[illegible]	[illegible]	$O(n\,\mathrm{polylog}(n))$	2	👎	👎	👍
[39]	[illegible]	$O(n\,\mathrm{polylog}(n))$	1	👎	👎	👍
[28, Theorem 20]	$1 - 1/e$	$O(n)$	1	👎	👎	👍
[35, Theorem 2]	[illegible]	[illegible]	k	👎	👎	👍
[14]	[illegible]	[illegible]	1	👎	👍	👍
[14]	$1/\varepsilon$	$O(n^2/\varepsilon^3)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}(n^{2-3\varepsilon} + n^{1-\varepsilon})$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2 + \varepsilon$	$O(n\,\mathrm{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[25]	$4.911 + \varepsilon$	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[29]	$3.5 + \varepsilon$	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	?
[53]	$2 + \varepsilon$	$O(n \log^2 n)$	1	👍	👍	?
[27]	$2 + \varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O(\log_{1+\varepsilon/3} n)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O(n \log(n)/\varepsilon^4)$	$O(\varepsilon^{-4} \log n)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$	👍	👎	👍
[17]	$4 + \varepsilon$	$O(n\,\mathrm{polylog}(n))$	1	👍	👍	👍

Our goals:

Maximize accuracy

Minimize local space

Minimize #passes



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[26]	1/2	$O(n)$		👎	👍	👍
	$1/2 + 0.0071$	$O(n$ pol		👎	👍	👍
	$1/2 + 0.003^*$	$O(n$ pol		👎	👍	👍
1]	$O(\text{polylog}(n))$	$O(\text{polyl}$		👎	👍	👍
]	$+\varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	👎	👎	👍
[6, Theorem 19]		$O(n \text{ polylog}(n)/\varepsilon^2)$	$O(\log(1/\varepsilon)/\varepsilon^2)$	👎	👎	👍
[4	9	n polylog		👎	👎	👍
[4	5^*	$n \log n)$		👎	👎	👍
[4	71^*	n polylog		👎	👎	👍
[39]	$1-1/e$	n polylog		👎	👎	👍
[28, Theorem 20]	$1-1/e$	$n)$	1	👎	👎	👍
[35, Theorem 2]			k	👎	👎	👍
[14]		)	1	👎	👍	👍
[14]	$1/\varepsilon$	$O(n^2/\varepsilon^3)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}(n^{2-3\varepsilon} + n^{1-\varepsilon})$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n \text{ polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O(n \log^2 n)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O(\log_{1+\varepsilon/3} n)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O(n \log(n)/\varepsilon^4)$	$O(\varepsilon^{-4} \log n)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$	👍	👎	👍
[17]	$4+\varepsilon$	$O(n \text{ polylog}(n))$	1	👍	👍	👍

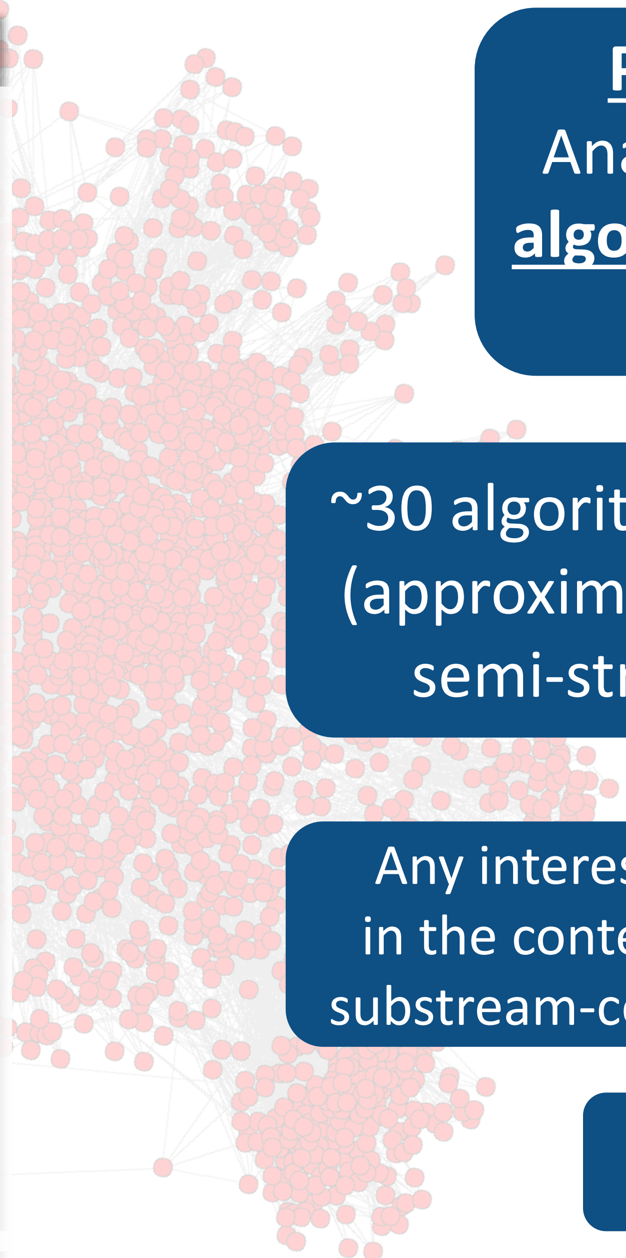
Our goals:

Maximize accuracy

Minimize local space

Minimize #passes

Accept weighted graphs



Part 3 continued : Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically…

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n$ [illegible]				
[illegible]	$1/2+0.0071$	$O(n$ pol[illegible]				
[illegible]	$1/2+0.003^*$	$O(n$ pol[illegible]				
[illegible]1]	[illegible]lylog$(n))$	$O($polyl[illegible]				
[illegible]	[illegible]$+\varepsilon$	$O(n \log n)$	$O(\log$ [illegible] $/\varepsilon)$ /			
[6, Theorem 19]	[illegible]	$O(n$ polylog$(\ldots)/\varepsilon^2)$	$O($[illegible]$\log(1/$[illegible]			
[4[illegible]	[illegible]	n polylog[illegible]	[illegible]			
[4[illegible]	[illegible]	$n \log n)$	[illegible]			
[4[illegible]	[illegible]	n polylog[illegible]	[illegible]			
[39[illegible]	[illegible]	n polylog[illegible]	[illegible]			
[28, Theorem 20]	$1-1/e$	[illegible]$n)$	1			
[35, Theorem 2]	[illegible]	[illegible]	k			
[14]	[illegible]	[illegible])	1			
[14]	$1/\varepsilon$	$O(n^2/\varepsilon^3)$	1			
[7, Theorem 1]	n^ε	$\tilde{O}(n^{2-3\varepsilon}+n^{1-\varepsilon})$	1			
[26, Theorem 2]	6	$O(n \log n)$	1			
[44, Theorem 3]	$2+\varepsilon$	$O(n \text{ polylog}(n))$	$O(1)$			
[44, Theorem 3]	5.82	$O(n \text{ polylog}(n))$	1			
[63]	5.58	$O(n \text{ polylog}(n))$	1			
[25]	$4.911+\varepsilon$	$O(n \text{ polylog}(n))$	1			
[29]	$3.5+\varepsilon$	$O(n \text{ polylog}(n))$	1			
[53]	$2+\varepsilon$	$O(n \log^2 n)$	1			
[27]	$2+\varepsilon$	$O(n \log n)$	1			
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O(\log_{1+\varepsilon/3} n)$			
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O(n \log(n)/\varepsilon^4)$	$O(\varepsilon^{-4} \log n)$			
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$			
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O(\varepsilon^{-2} \log(\varepsilon^{-1}))$			
[17]	$4+\varepsilon$	$O(n \text{ polylog}(n))$	1			

Our goals:

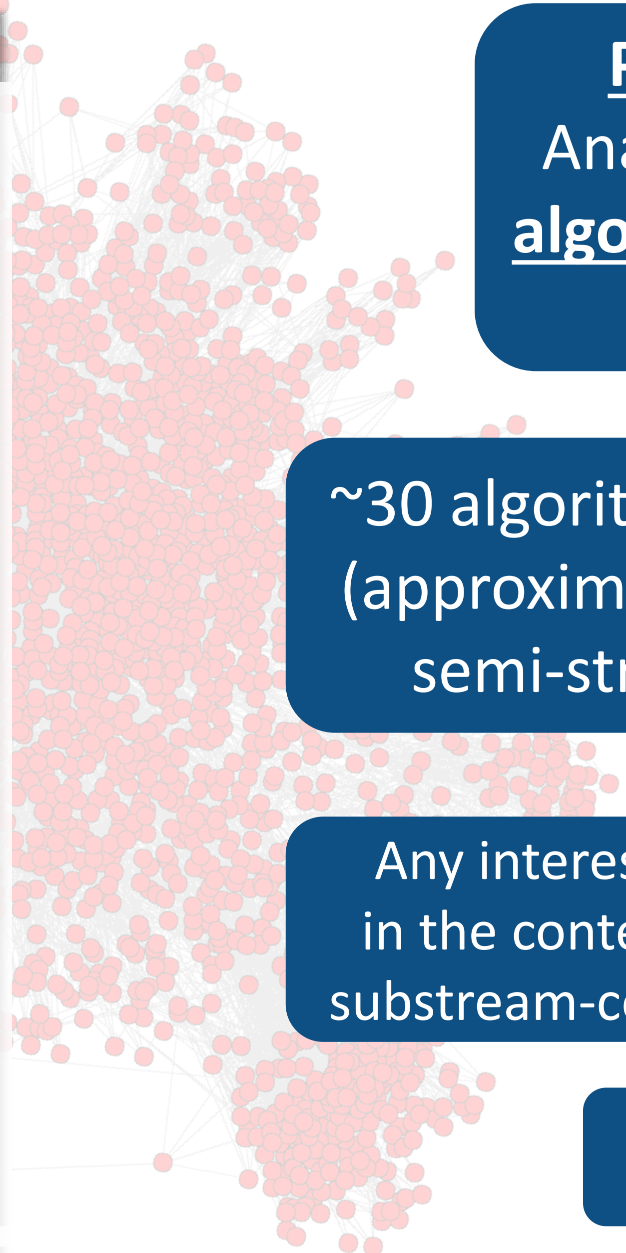
Maximize accuracy

Minimize local space

Minimize #passes

Accept weighted graphs

Accept general (not just bipartite) graphs



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	$1/2$	$O(n)$	[illegible]	👎	[illegible]	👍
[illegible]	$1/2+0.0071$	$O(n$ pol[illegible]	[illegible]	👎	[illegible]	👍
[illegible]	$1/2+0.003^*$	$O(n$ pol[illegible]	[illegible]	👎	[illegible]	👍
[illegible]1]	$O(\text{polylog}(n))$	$O(\text{polyl}$[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]ε	$O(n\log n)$	$O(\log$[illegible]$/\varepsilon)/$[illegible]	[illegible]	[illegible]	[illegible]
[6, Theorem 19]	[illegible]	$O(n\,\text{polylog}$[illegible]	$O($[illegible]$\log(1/$[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	[illegible]$n\,\text{polylog}$[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	[illegible]$n\log n)$	[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	[illegible]$n\,\text{polylog}$[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[39[illegible]	[illegible]	[illegible]$n\,\text{polylog}$[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[28, Theorem 20]	$1-1/e$	[illegible]$n)$	[illegible]	👎	👎	👍
[35, Theorem 2]	[illegible]	[illegible]	k	👎	👎	👍
[14]	[illegible]	[illegible]	1	👎	👍	👍
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n\log n)$	1	👍	👍	?
[44, Theorem 3]	$2+\varepsilon$	$O(n\,\text{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\,\text{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\,\text{polylog}(n))$	1	👍	👍	?
[25]	$4.911+\varepsilon$	$O(n\,\text{polylog}(n))$	1	👍	👍	?
[29]	$3.5+\varepsilon$	$O(n\,\text{polylog}(n))$	1	👍	👍	?
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	👍	👍	?
[27]	$2+\varepsilon$	$O(n\log n)$	1	👍	👍	?
[26, Section 3.2]	$2+\varepsilon$	$O(n\log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n-\log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
[17]	$4+\varepsilon$	$O(n\,\text{polylog}(n))$	1	👍	👍	👍

Our goals:

Maximize accuracy

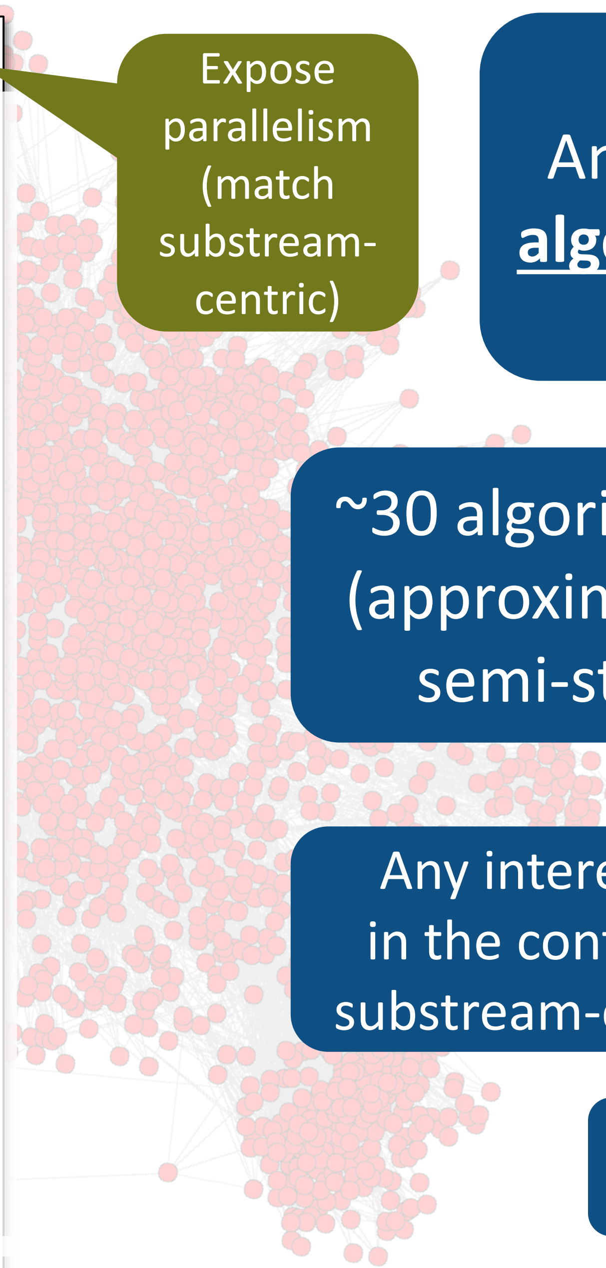
Minimize local space

Minimize #passes

Accept weighted graphs

Accept general (not just bipartite) graphs

Expose parallelism (match substream-centric)



Part 3 continued :
Analysis of models and **algorithms** for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

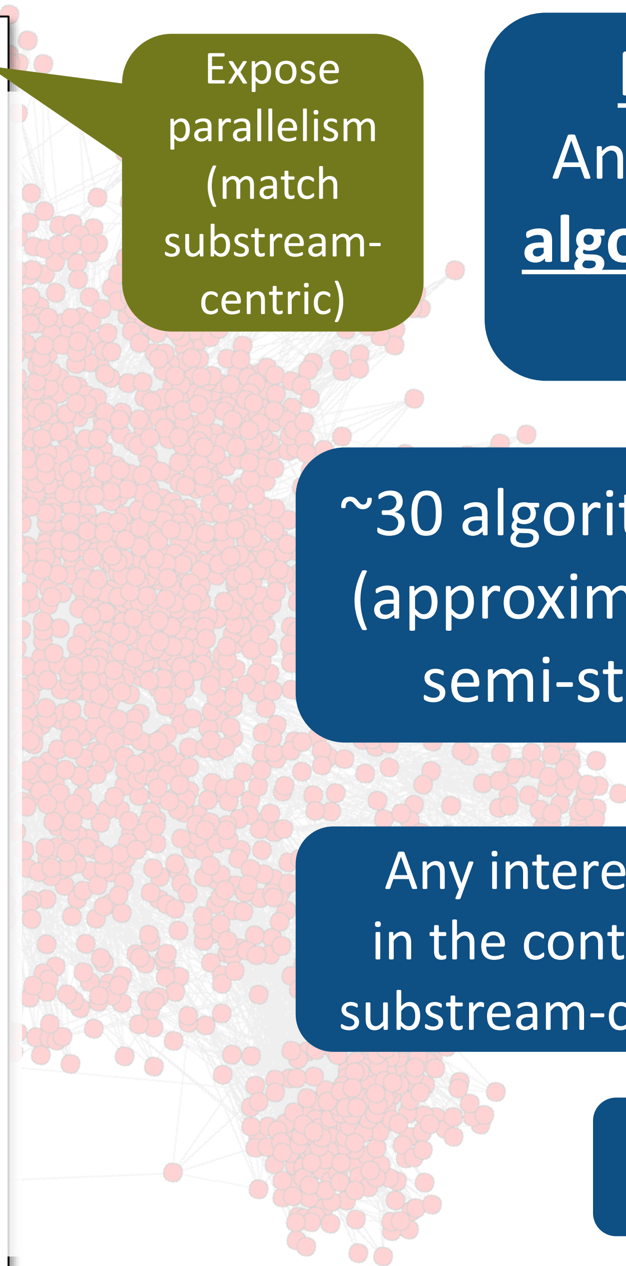
More specifically...

Part 3 continued : Analysis of models and algorithms for streaming graph processing

Our goals:

- Maximize accuracy
- Minimize local space
- Minimize #passes
- Accept weighted graphs
- Accept general (not just bipartite) graphs
- Expose parallelism (match substream-centric)

Reference	Approx.	Space	#Passes	Wgh[1]	Gen[2]	Par[3]
[illegible]	1/2	$O(n)$	[illegible]	👎	[illegible]	👍
[illegible]	1/2 + 0.0071	$O(n$ pol[illegible]	[illegible]	👎	[illegible]	👍
[illegible]	1/2 + 0.003*	$O(n$ pol[illegible]	[illegible]	👎	[illegible]	👍
[illegible]1]	$O($polylog$(n))$	$O($polyl[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible] ε	$O(n \log n)$	$O($log[illegible]$/\varepsilon)$ / [illegible]	[illegible]	[illegible]	[illegible]
[6, Theorem 19]	[illegible]	$O(n$ polylog[illegible]	[illegible] log (1/[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	n polylog[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	$n \log n)$	[illegible]	[illegible]	[illegible]	[illegible]
[4[illegible]	[illegible]	n polylog[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[39]	[illegible]	n polylog[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[28, Theorem 20]	$1 - 1/e$	[illegible]	[illegible]	👎	👎	👍
[35, Theorem 2]	[illegible]	[illegible]	k	👎	👎	👍
[14]	[illegible]	[illegible]	1	👎	👍	👍
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	👎	👍	👍
[7, Theorem 1]	n^ε	$\tilde{O}\left(n^{2-3\varepsilon} + n^{1-\varepsilon}\right)$	1	👎	👎	👍
[26, Theorem 2]	6	$O(n \log n)$	1	👍	👍	?
[44, Theorem 3]	$2 + \varepsilon$	$O(n\ \text{polylog}(n))$	$O(1)$	👍	👍	?
[44, Theorem 3]	5.82	$O(n\ \text{polylog}(n))$	1	👍	👍	?
[63]	5.58	$O(n\ \text{polylog}(n))$	1	👍	👍	?
[25]	$4.911 + \varepsilon$	$O(n\ \text{polylog}(n))$	1	👍	👍	?
[29]	$3.5 + \varepsilon$	$O(n\ \text{polylog}(n))$	1	👍	👍	?
[53]	$2 + \varepsilon$	$O\left(n \log^2 n\right)$	1	👍	👍	?
[27]	$2 + \varepsilon$	$O(n \log n)$	1	👍	👍	?
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	👍	👍	?
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n \log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4} \log n\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👍	👍
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2} \log\left(\varepsilon^{-1}\right)\right)$	👍	👎	👍
[17]	$4 + \varepsilon$	$O(n\ \text{polylog}(n))$	1	👍	👍	👍



~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

More specifically...

Research Questions

Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

How to design a high-performance MWM algorithm (as dictated by the used paradigm)?

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

Use the MWM algorithm by Crouch and Stubbs in the substream-centric paradigm **in a hybrid CPU-FPGA setting**

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

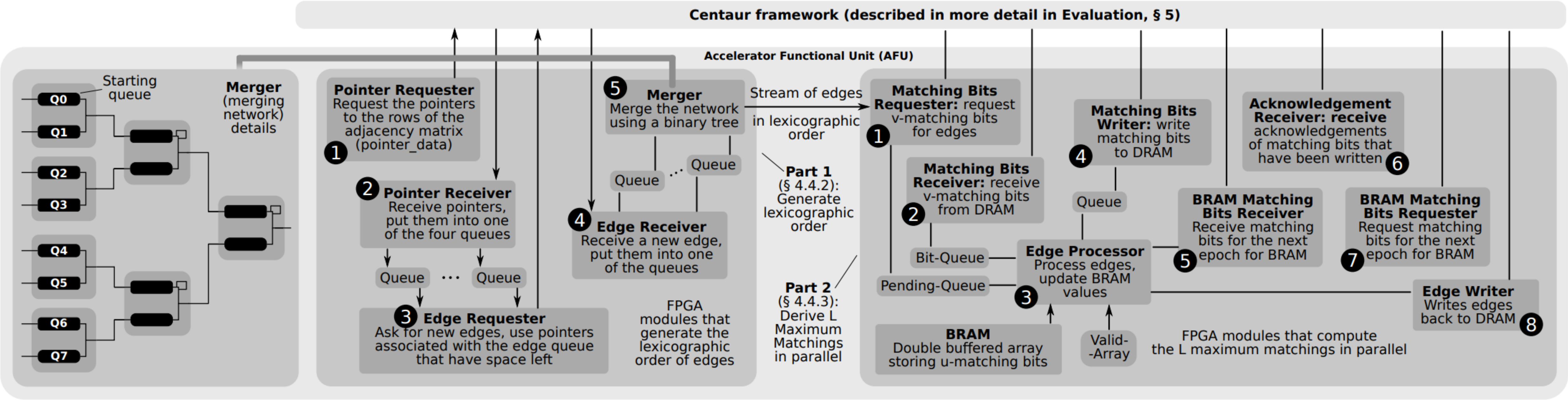
For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

Use the MWM algorithm by Crouch and Stubbs in the substream-centric paradigm **in a hybrid CPU-FPGA setting**

What is the HW FPGA design that ensures high performance?

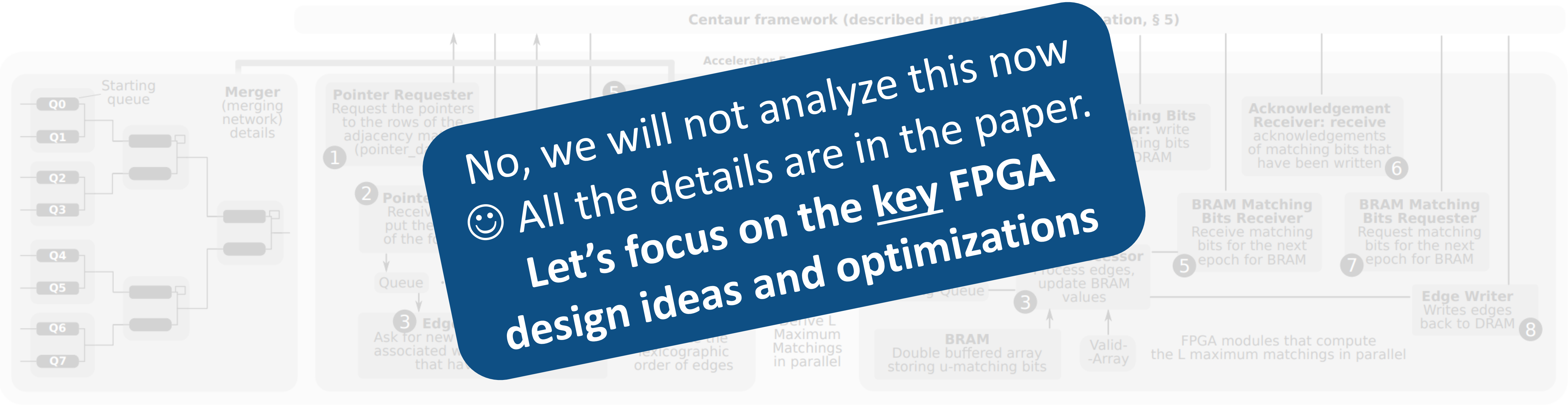
What is the ultimate performance, power consumption, and the related tradeoffs?

Substream-Centric MWM: FPGA optimizations



Substream-Centric MWM: FPGA optimizations

Substream-Centric MWM: FPGA optimizations

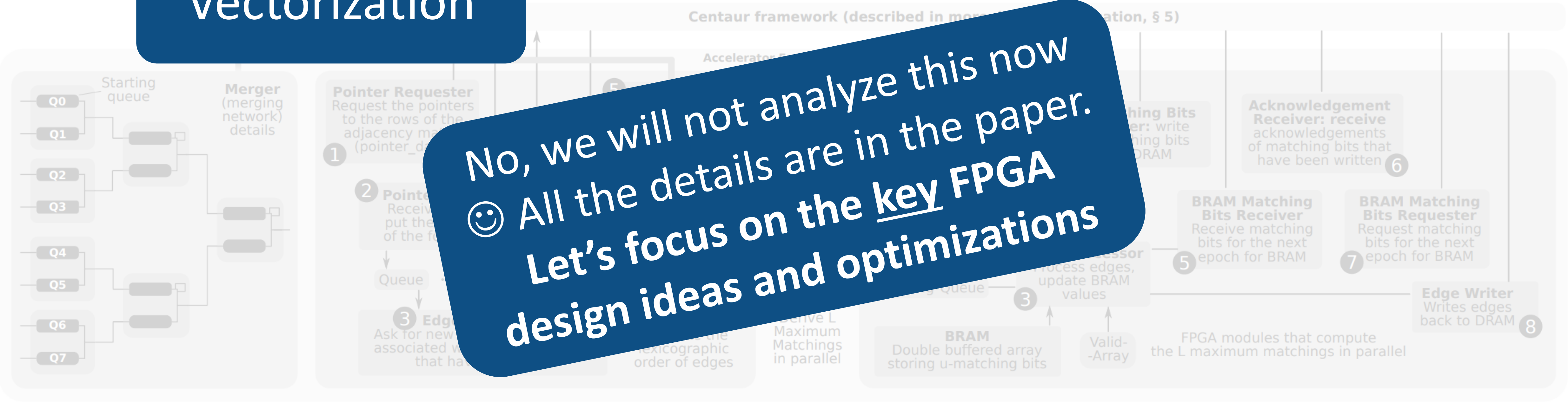


No, we will not analyze this now
☺ All the details are in the paper.
Let's focus on the key FPGA design ideas and optimizations

Substream-Centric MWM: FPGA optimizations

Blocking / Tiling

Vectorization



No, we will not analyze this now
☺ All the details are in the paper.
Let's focus on the key FPGA design ideas and optimizations

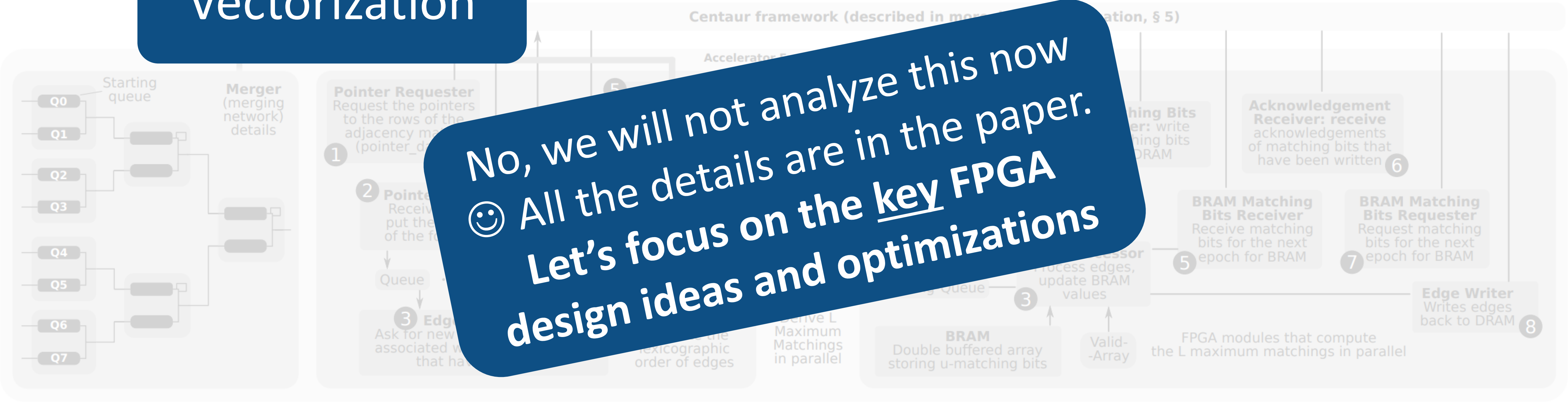
Prefetching

Pipelining

Substream-Centric MWM: FPGA optimizations

Blocking / Tiling

Vectorization



No, we will not analyze this now ☺ All the details are in the paper.
Let's focus on the <u>key</u> FPGA design ideas and optimizations

Prefetching

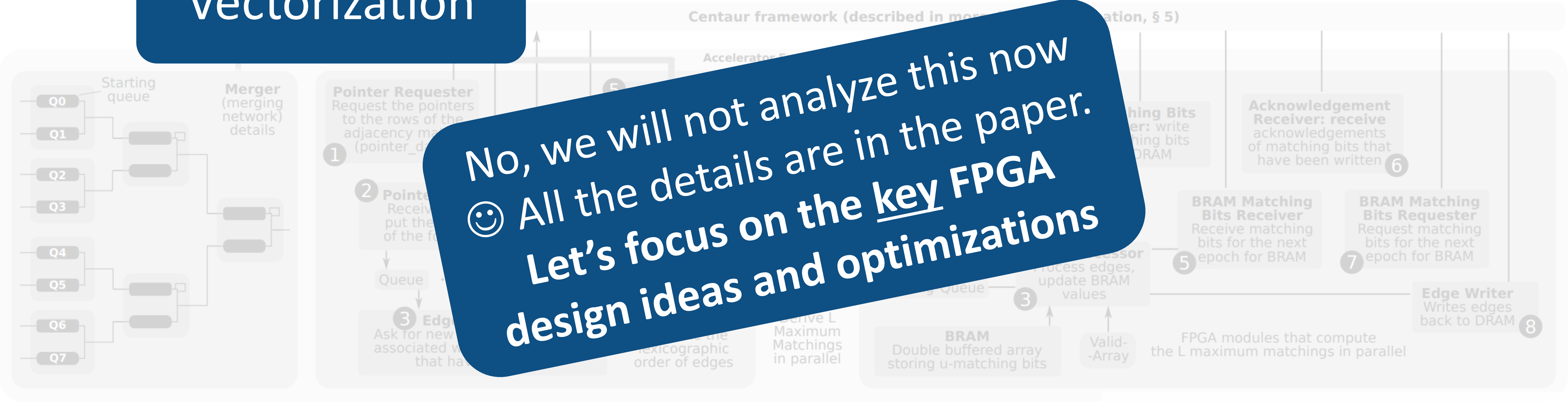
They are often used in graph processing schemes on FPGAs; we apply them as well.

Pipelining

Substream-Centric MWM: FPGA optimizations

Blocking / Tiling

Vectorization



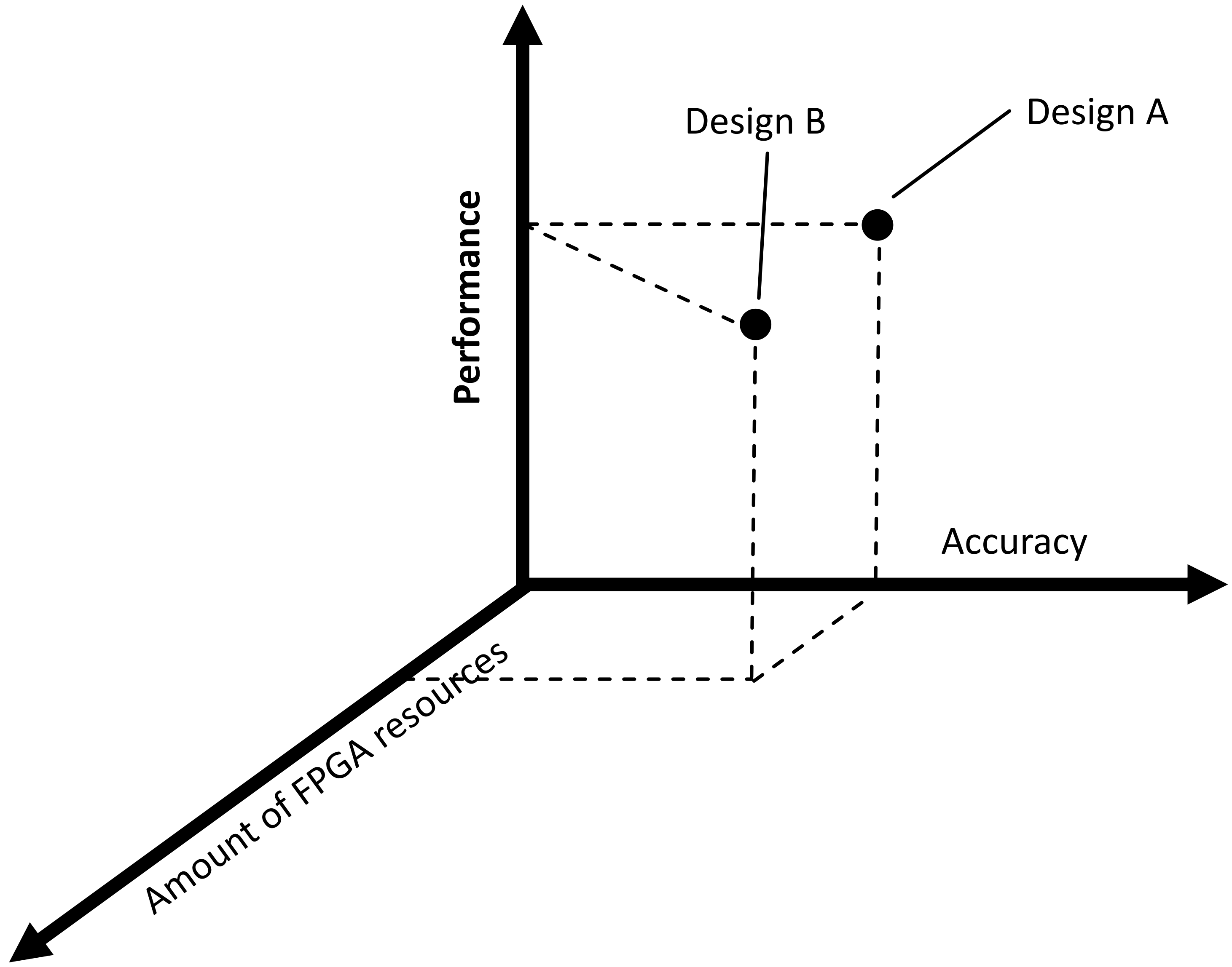
No, we will not analyze this now ☺ All the details are in the paper.
Let's focus on the <u>key</u> FPGA design ideas and optimizations

Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

Pipelining

The Space of Substream-Centric



Research Questions

Use **substream-centric processing** (exposes parallelism)

Use the MWM algorithm by Crouch and Stubbs in the substream-centric paradigm **in a hybrid CPU-FPGA setting**

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

What is the HW FPGA design that ensures high performance?

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

Use the MWM algorithm by Crouch and Stubbs in the substream-centric paradigm **in a hybrid CPU-FPGA setting**

The proper use of **blocking**, **vectorization**, and others (pipelining, prefetching)

What is the ultimate performance, power consumption, and the related tradeoffs?

Research Questions

Use **substream-centric processing** (exposes parallelism)

For enabling theoretical analysis and rigor **in the context of FPGAs** (small local space), use the **semi-streaming model**

Use the MWM algorithm by Crouch and Stubbs in the substream-centric paradigm **in a hybrid CPU-FPGA setting**

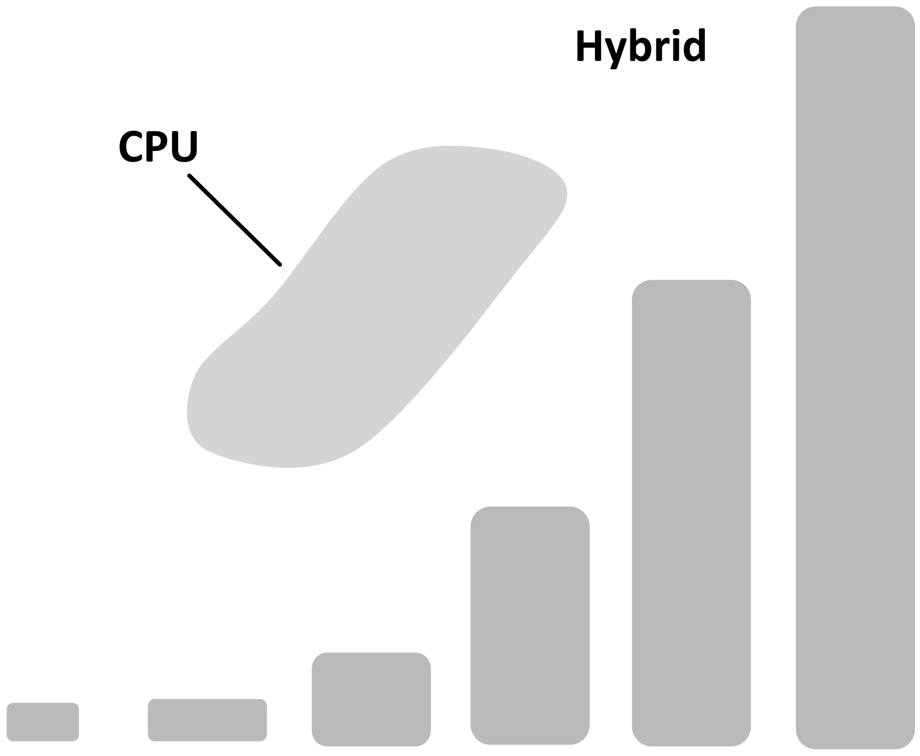
The proper use of **blocking**, **vectorization**, and others (pipelining, prefetching)

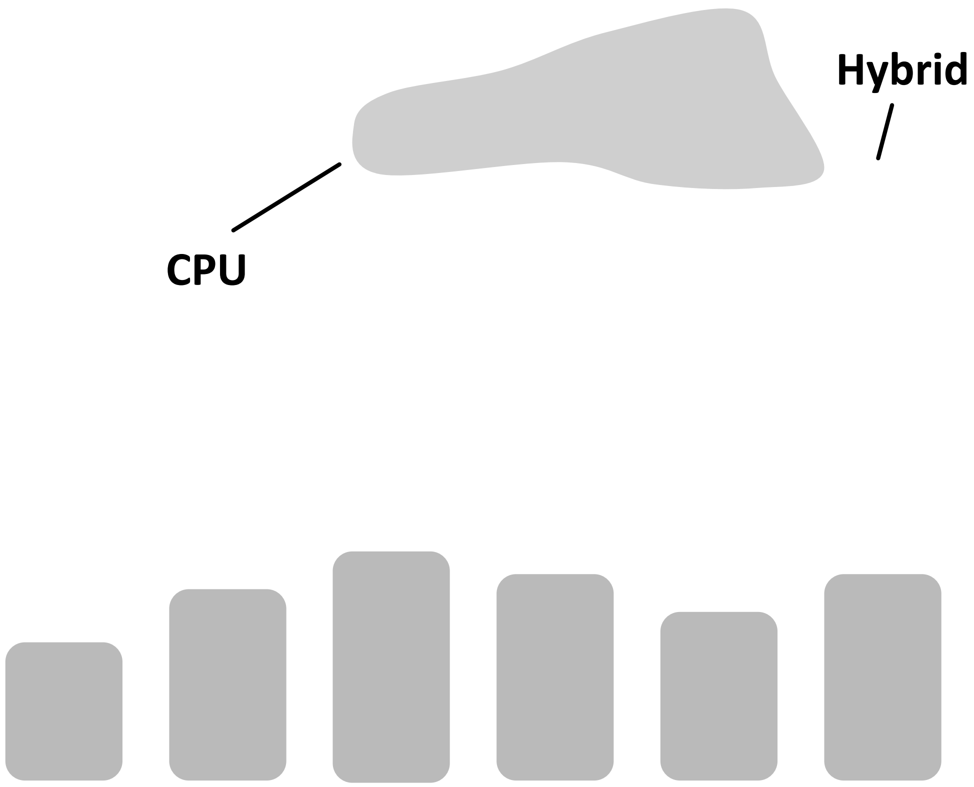
What is the ultimate performance, power consumption, and the related tradeoffs?

Performance Analysis
Various Graphs

Parameters:
Blocking size = 32, #Substreams = 64
#Threads = 4, ε = 0.1

Algorithm	**Platform**
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid



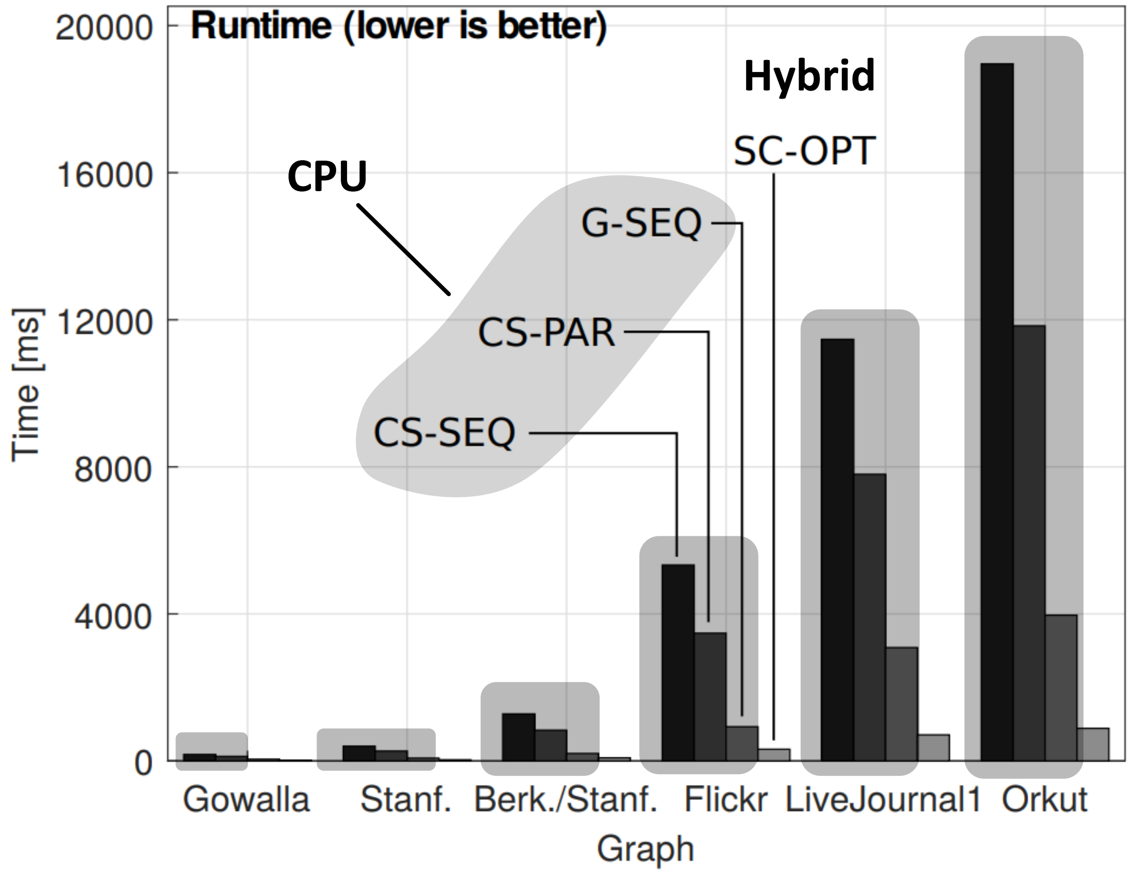


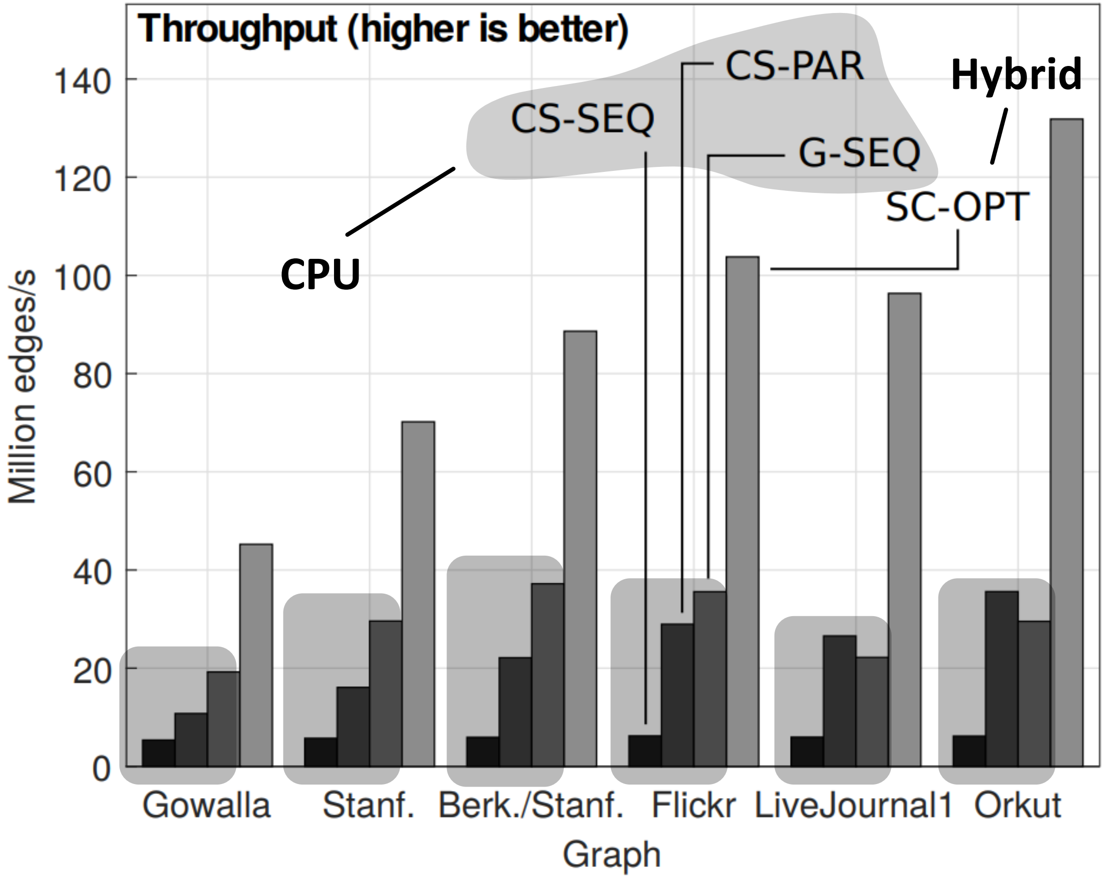
Performance Analysis
Various Graphs

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size = 32, #Substreams = 64
#Threads = 4, ε = 0.1

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770





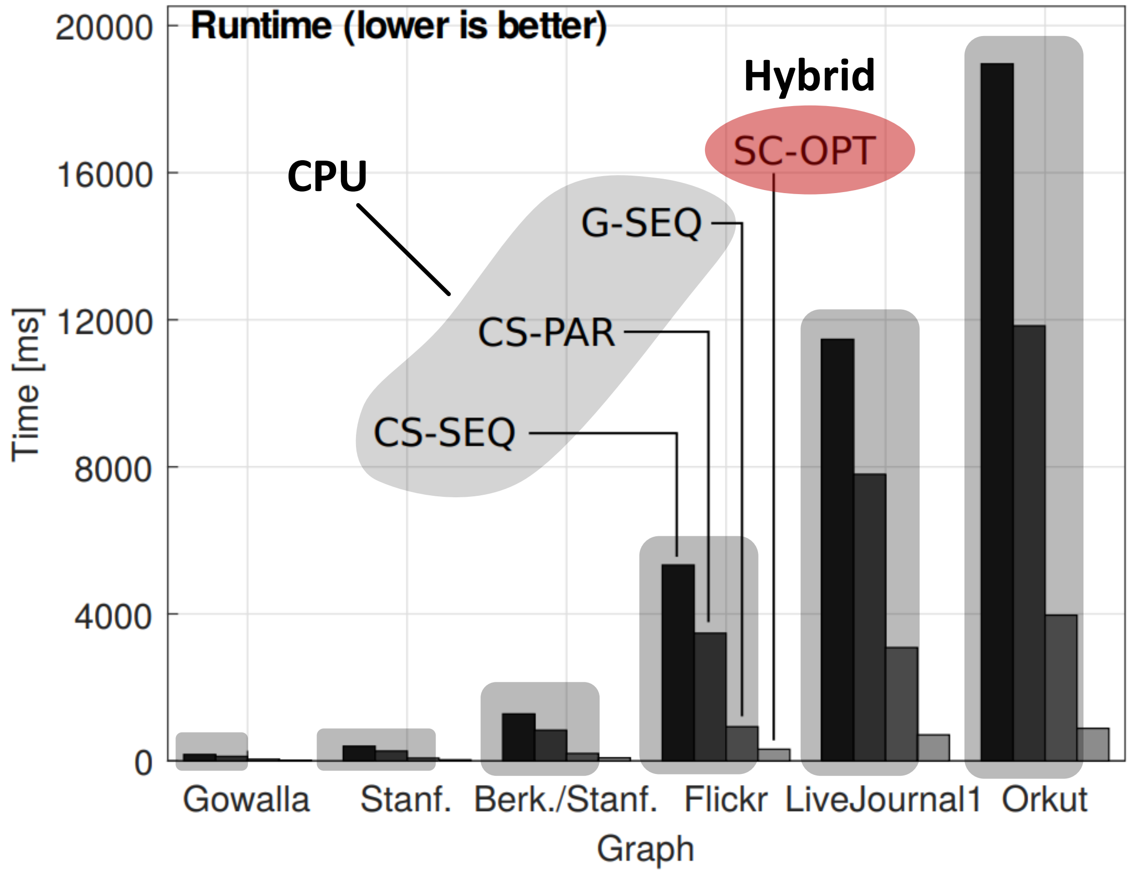
Performance Analysis
Various Graphs

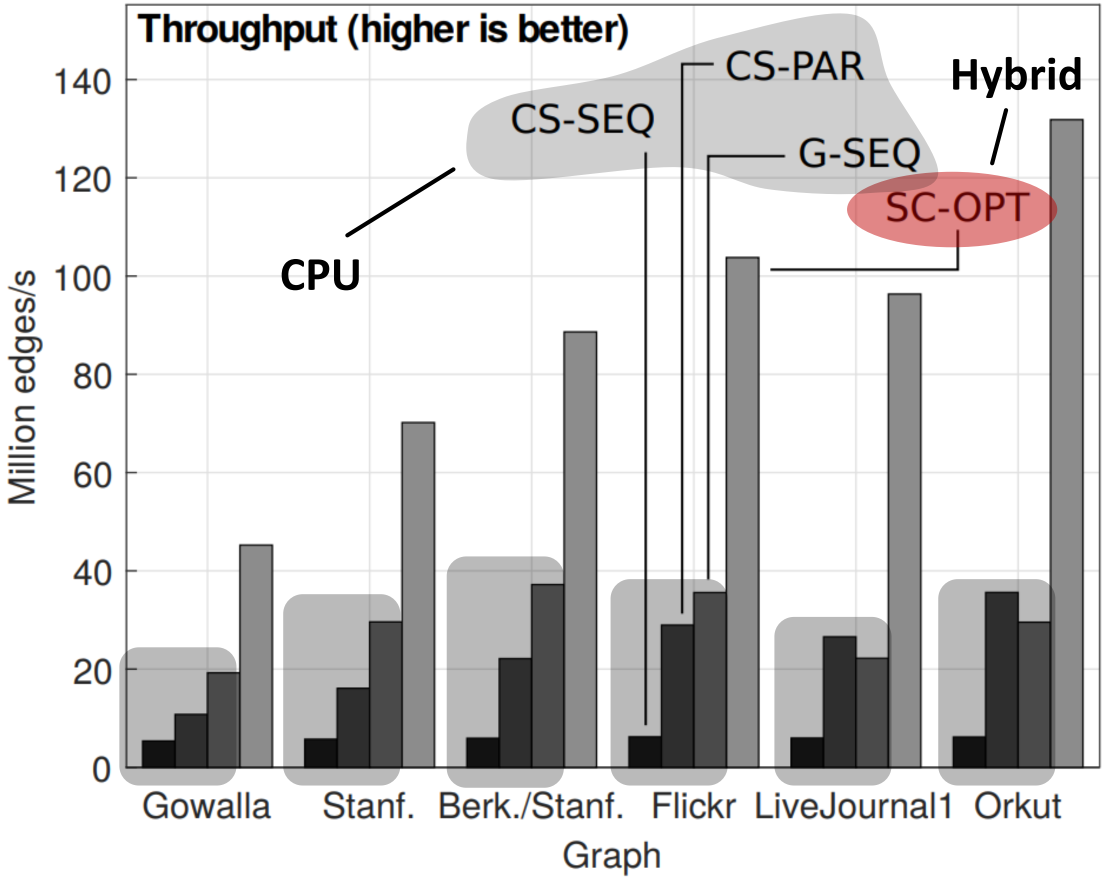
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size = 32, #Substreams = 64
#Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770





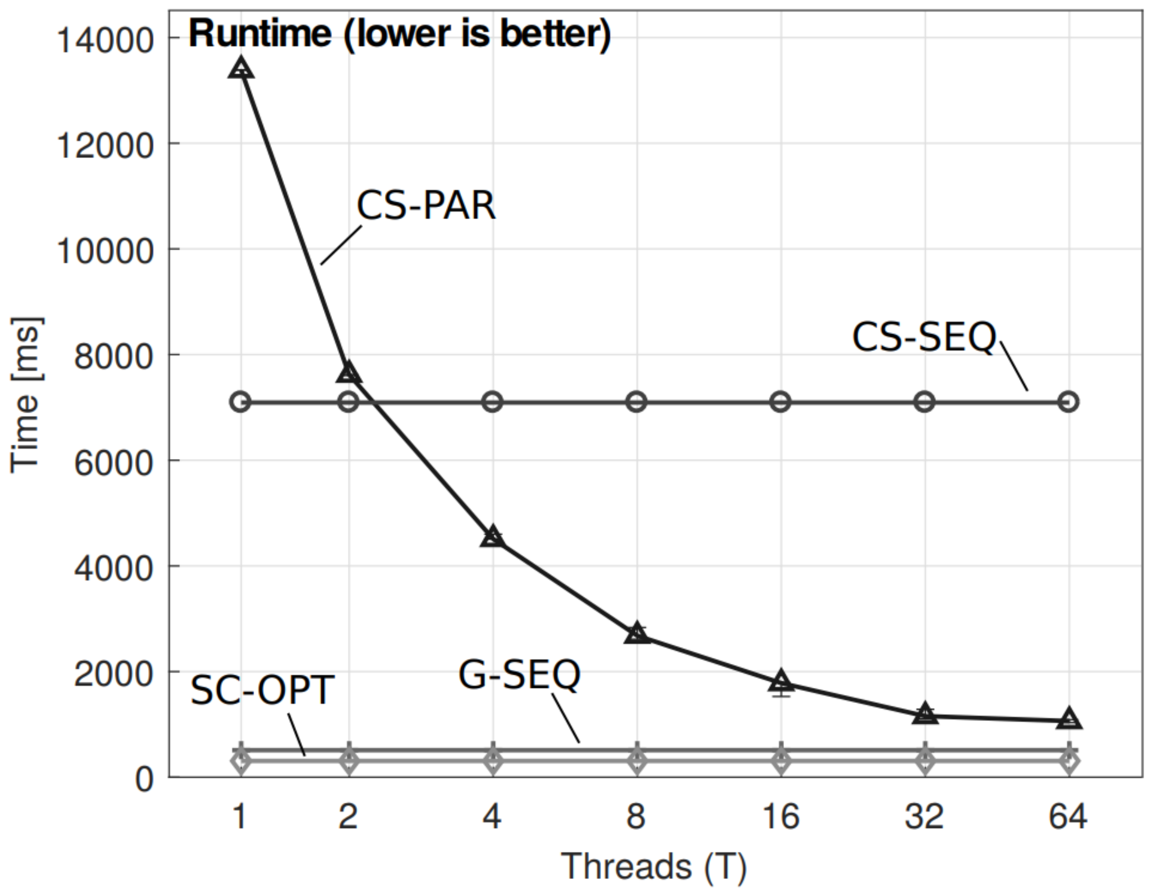
Performance Analysis
Various Thread (CPU) Counts

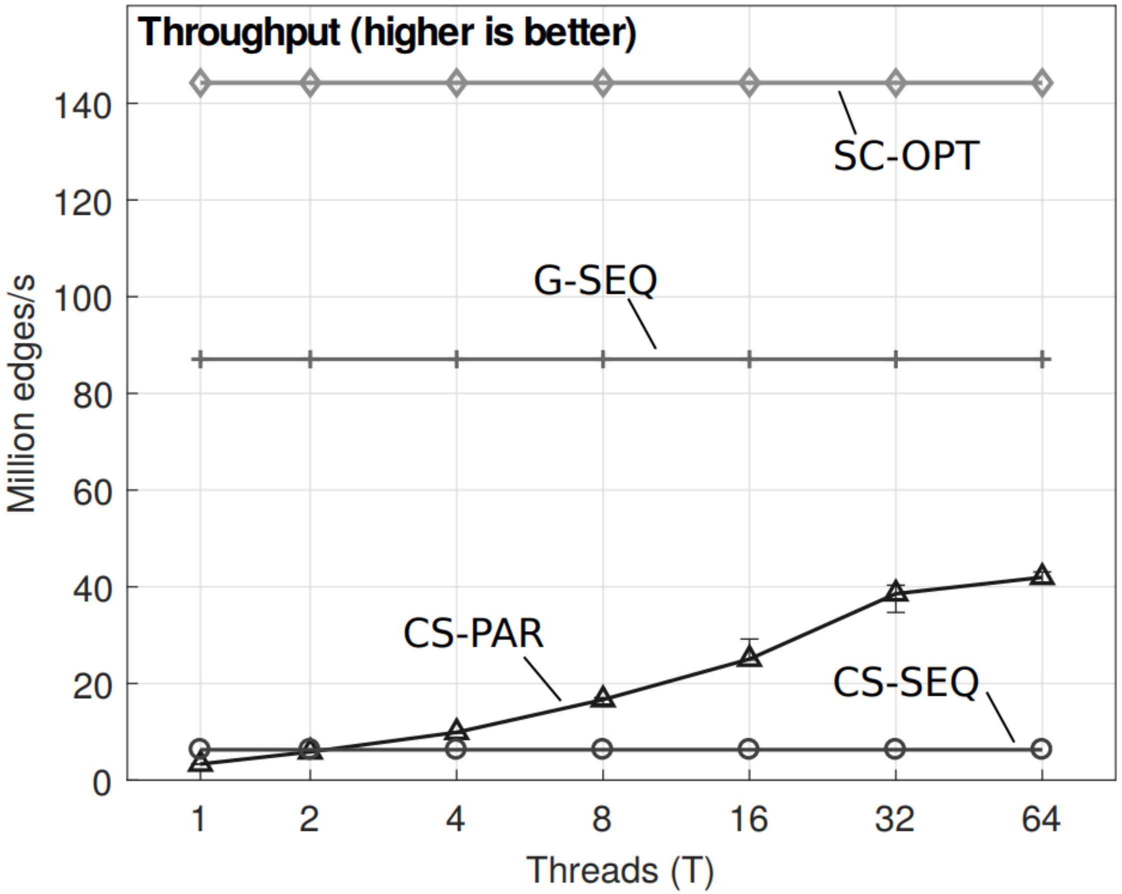
Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

Performance Analysis
Various Thread (CPU) Counts

Blocking size = 32, #Substreams = 64
#edges = 16M (Kronecker), ε = 0.1

Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)



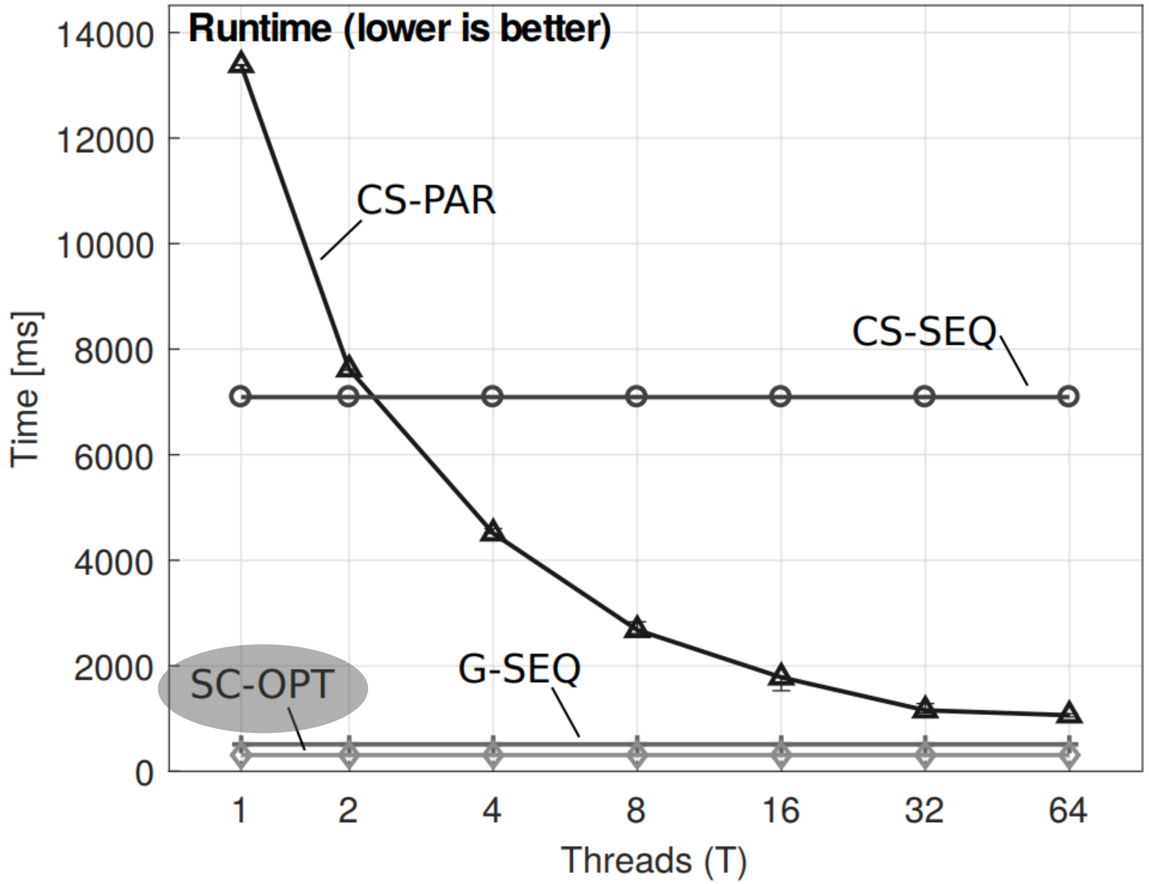


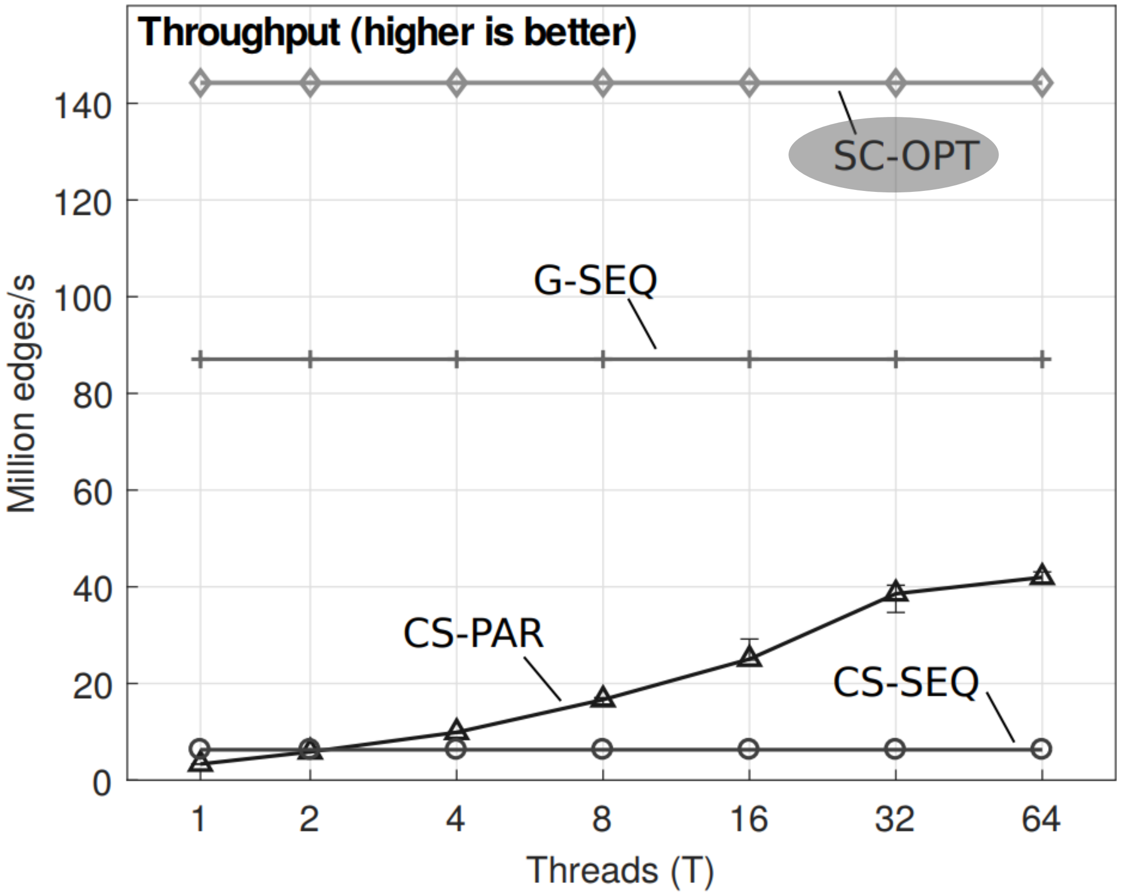
Performance Analysis
Various Thread (CPU) Counts

Blocking size = 32, #Substreams = 64
#edges = 16M (Kronecker), ε = 0.1

Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

SC-OPT secures highest performance for all considered numbers of threads





Performance Analysis

Various blocking size (K) and #substreams (L)

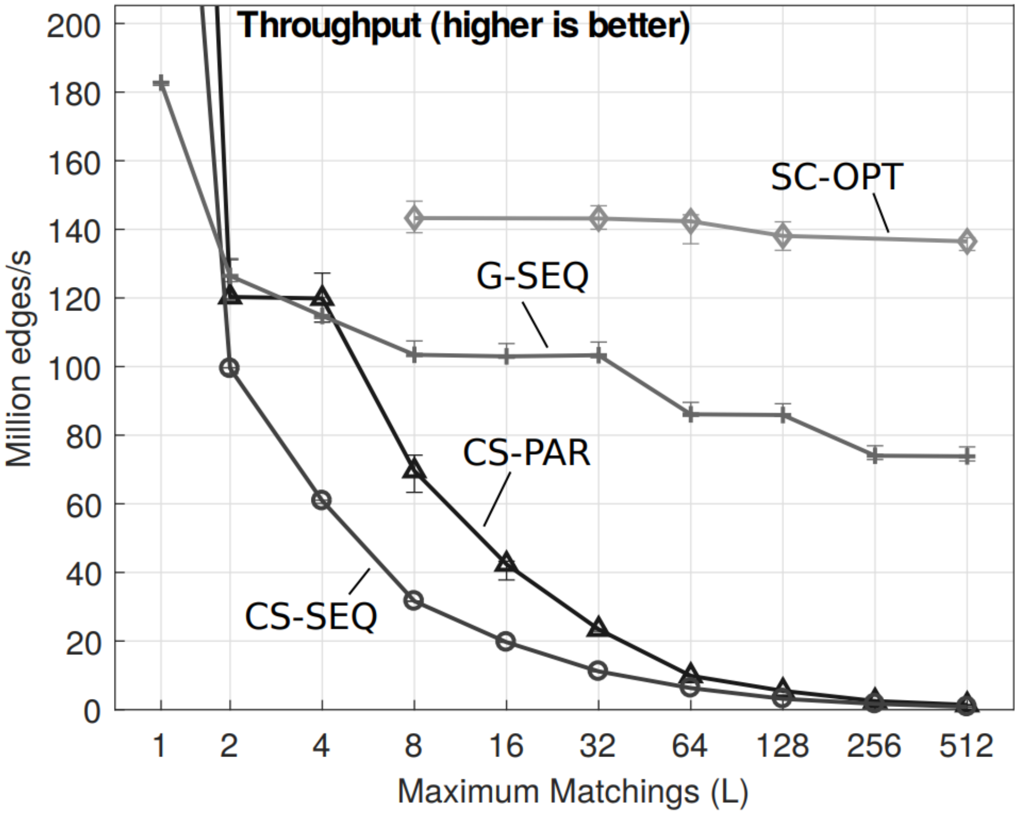
Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

Performance Analysis

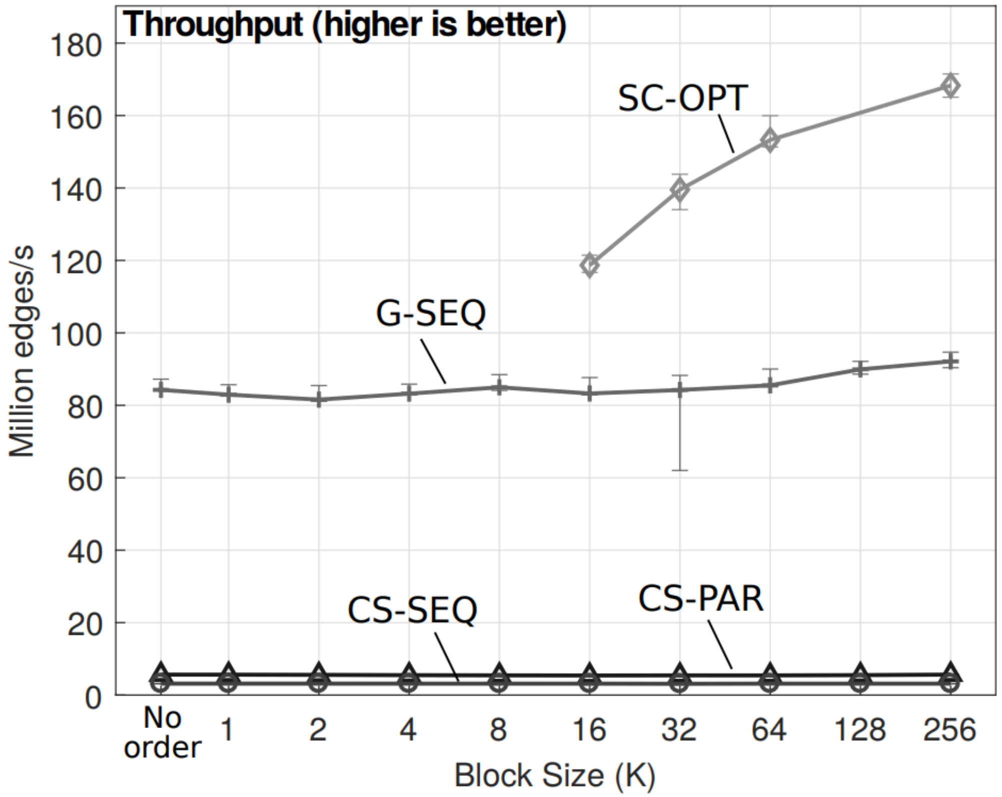
Various blocking size (K) and #substreams (L)

Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1



#Substreams (L) = 128, #threads = 4, #edges = 16M (Kronecker), ε = 0.1

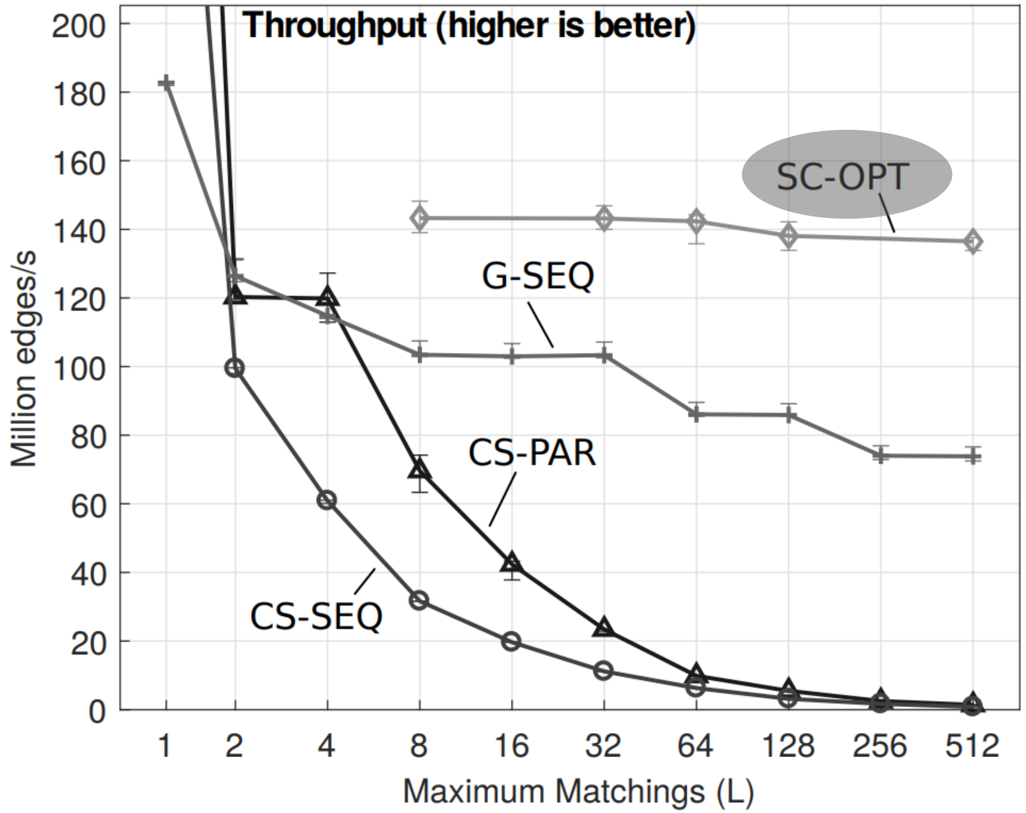


Performance Analysis
Various blocking size (K) and #substreams (L)

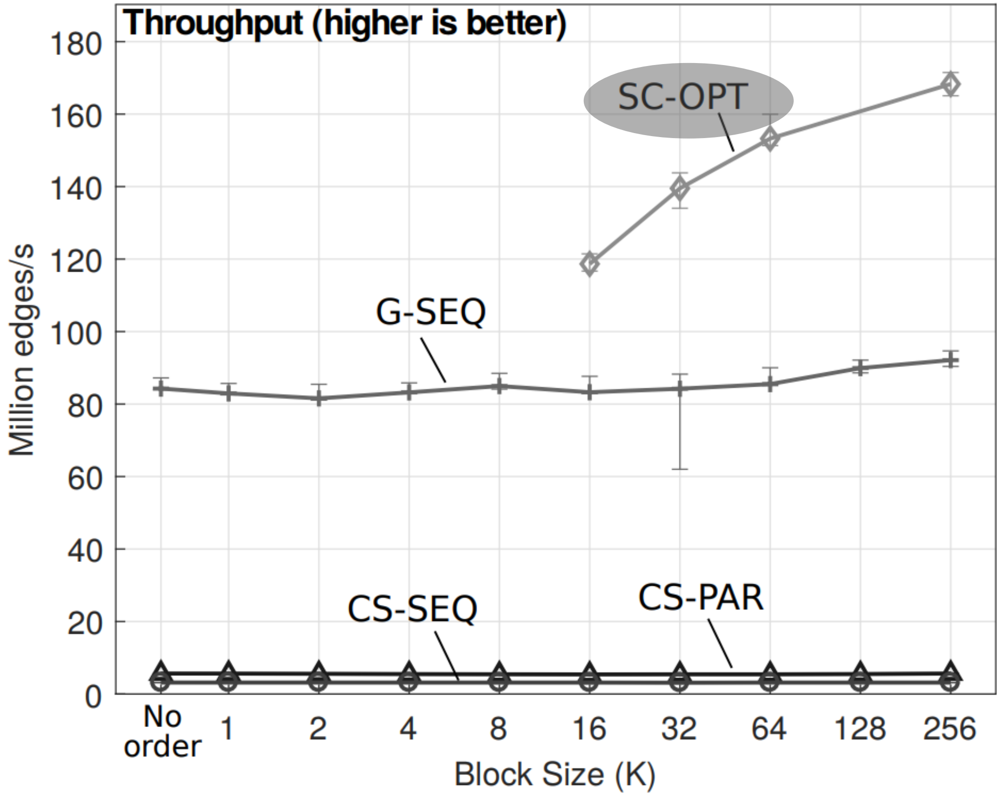
Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

SC-OPT secures highest performance for all considered values of parameters

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1



#Substreams (L) = 128, #threads = 4, #edges = 16M (Kronecker), ε = 0.1



PERFORMANCE ANALYSIS

APPROXIMATION

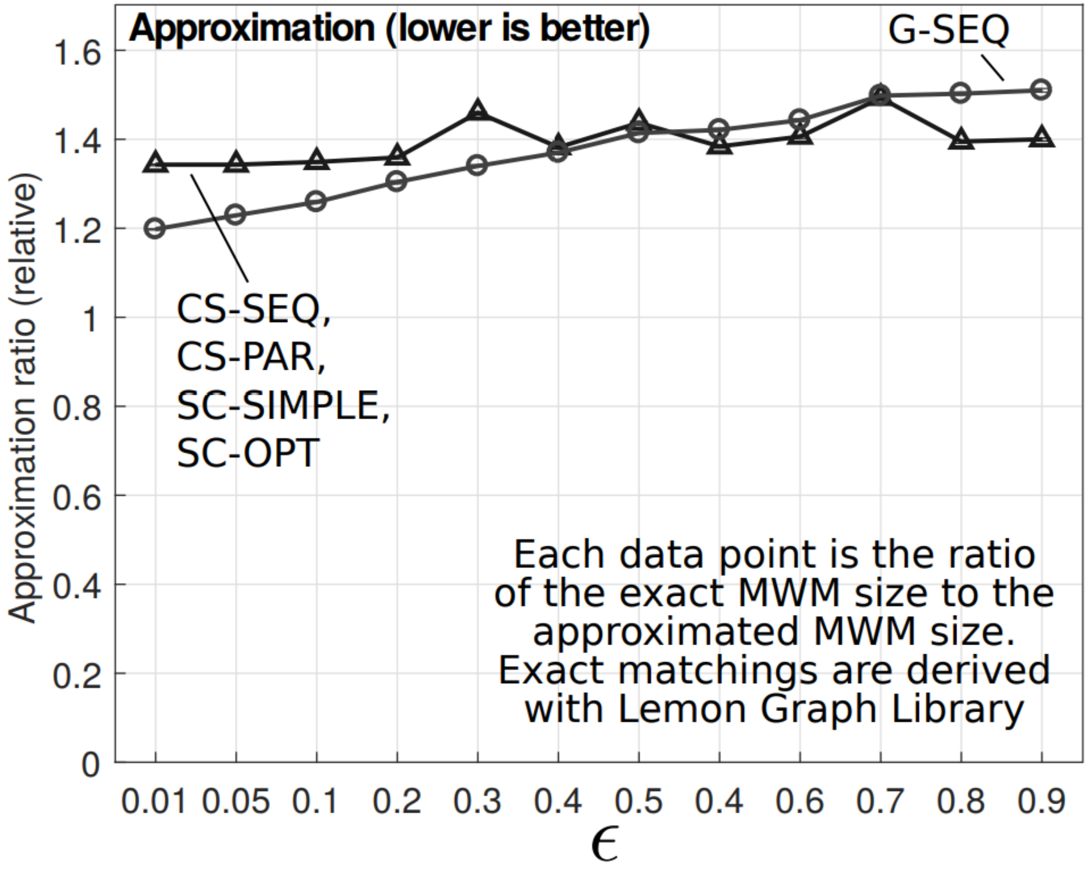
Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

Performance Analysis

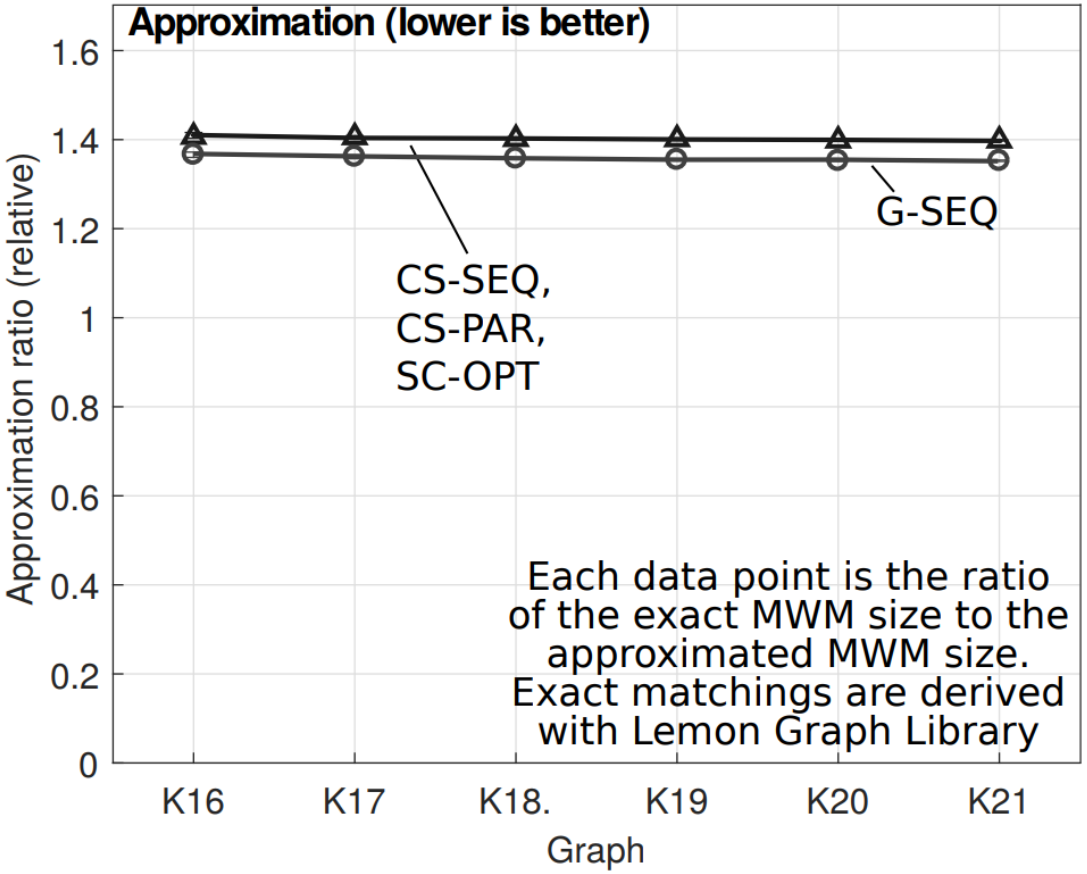
Approximation

Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)



Blocking size (K) = 32, #threads = 4, #Substreams (L) = 128 , ε = 0.1

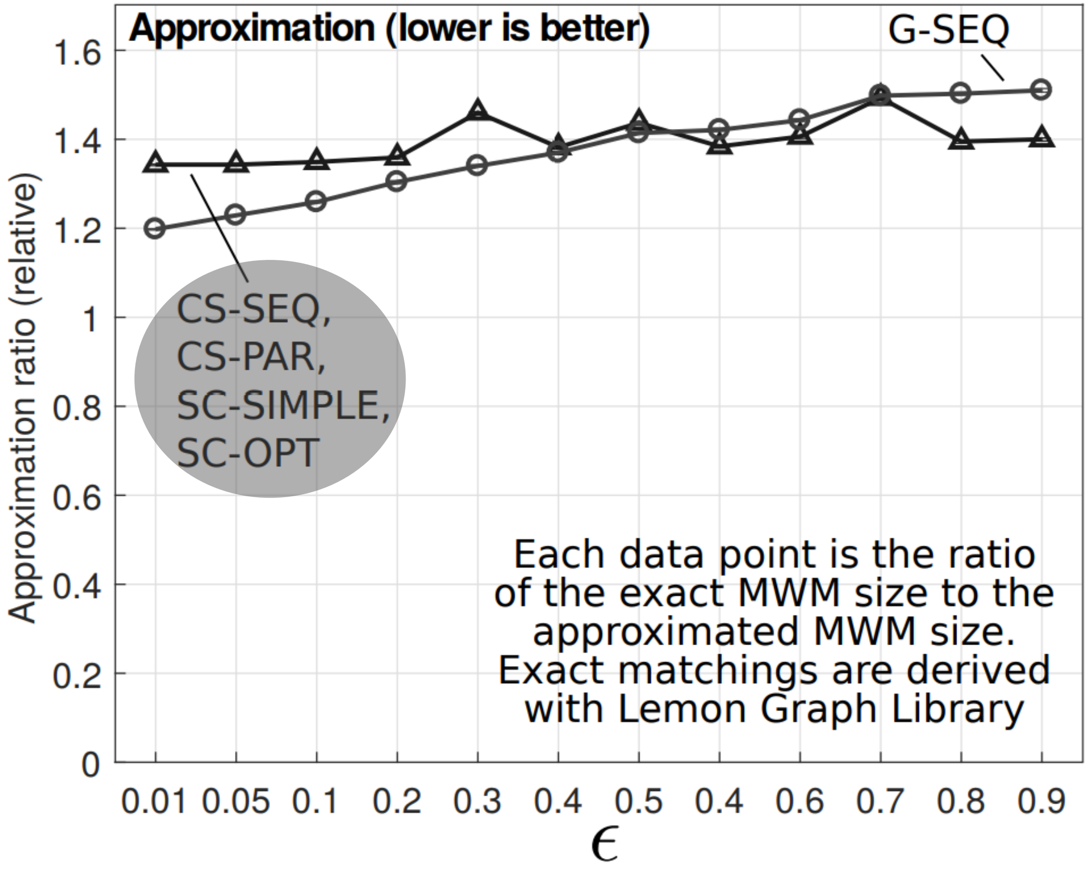


Performance Analysis
Approximation

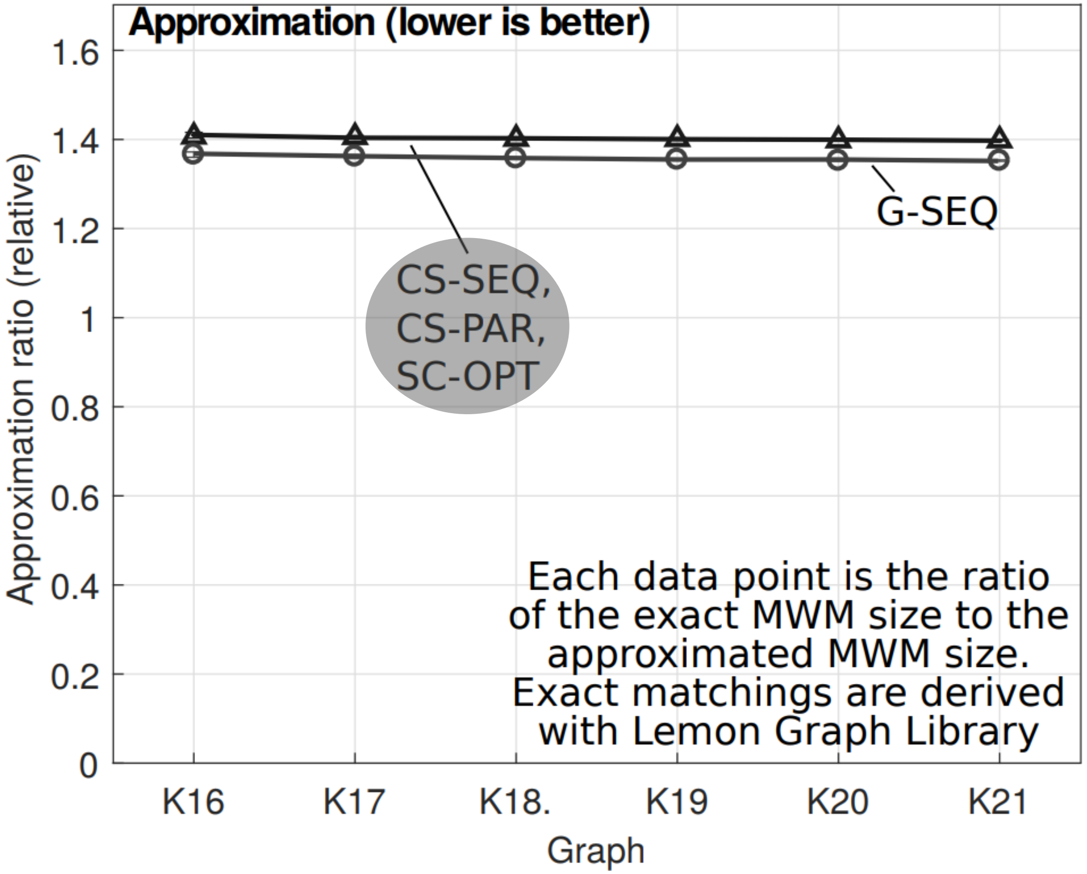
Algorithm
Crouch et al. [1] Sequential (CS-SEQ)
Crouch et al. [1] Parallel (CS-PAR)
Ghaffari [2] Sequential (G-SEQ)
Substream-Centric, no blocking (SC-SIMPLE)
Substream-Centric, with blocking (SC-OPT)

SC-OPT is comparable to the (2+ε)-approximation by Ghaffari et al.

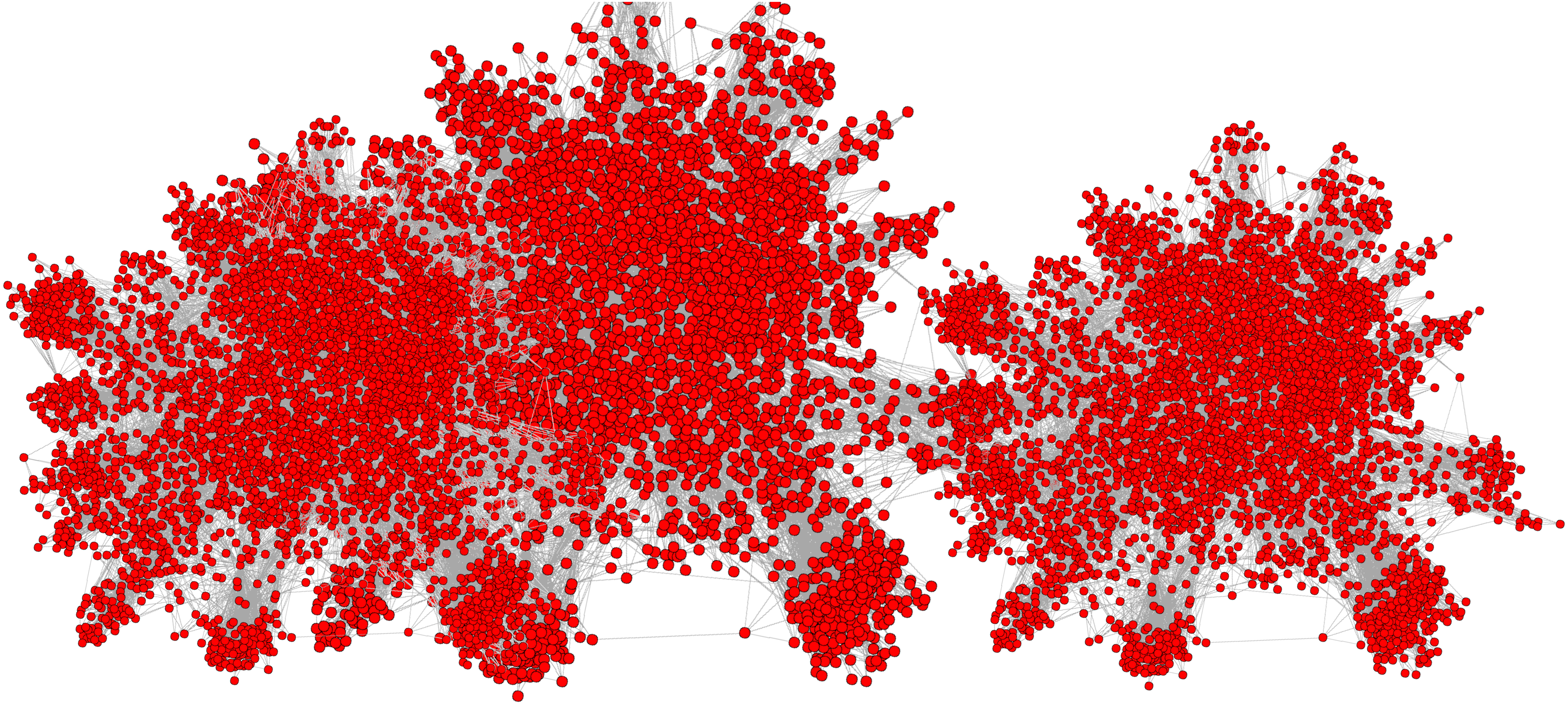
#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)



Blocking size (K) = 32, #threads = 4, #Substreams (L) = 128 , ε = 0.1



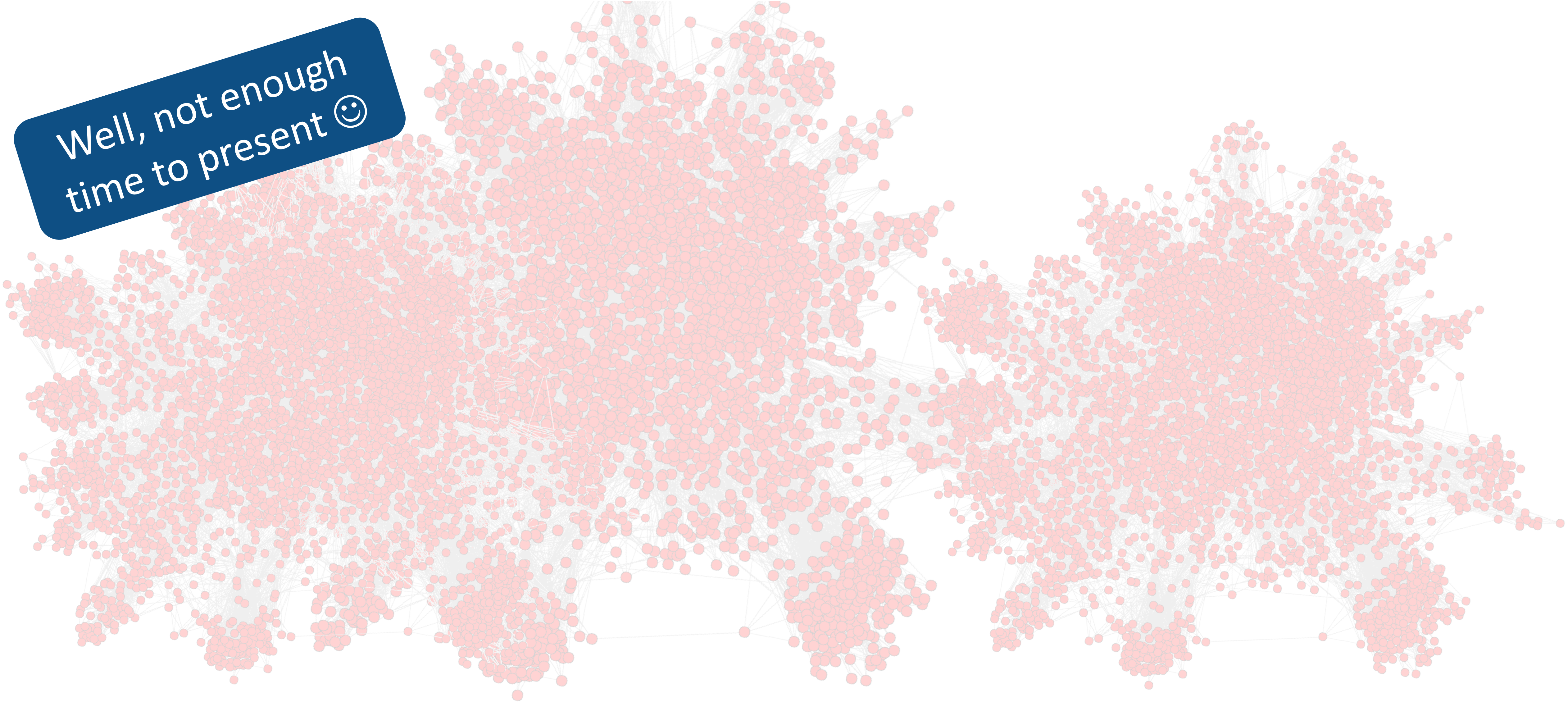
Other Algorithms, Problems, Analyses



Other Algorithms, Problems, Analyses

Other Algorithms, Problems, Analyses

Well, not enough time to present ☺



Other Algorithms, Problems, Analyses

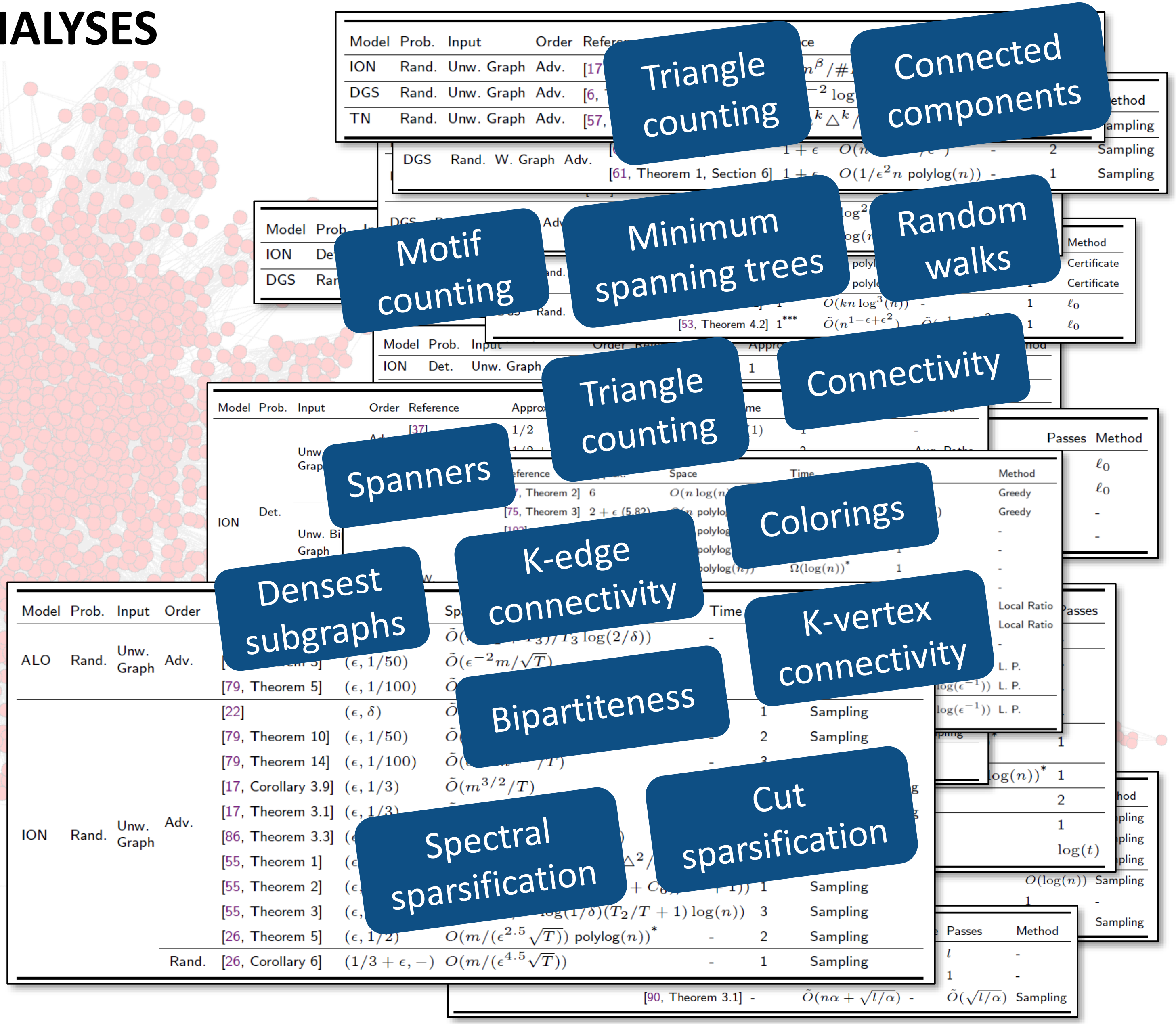
Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

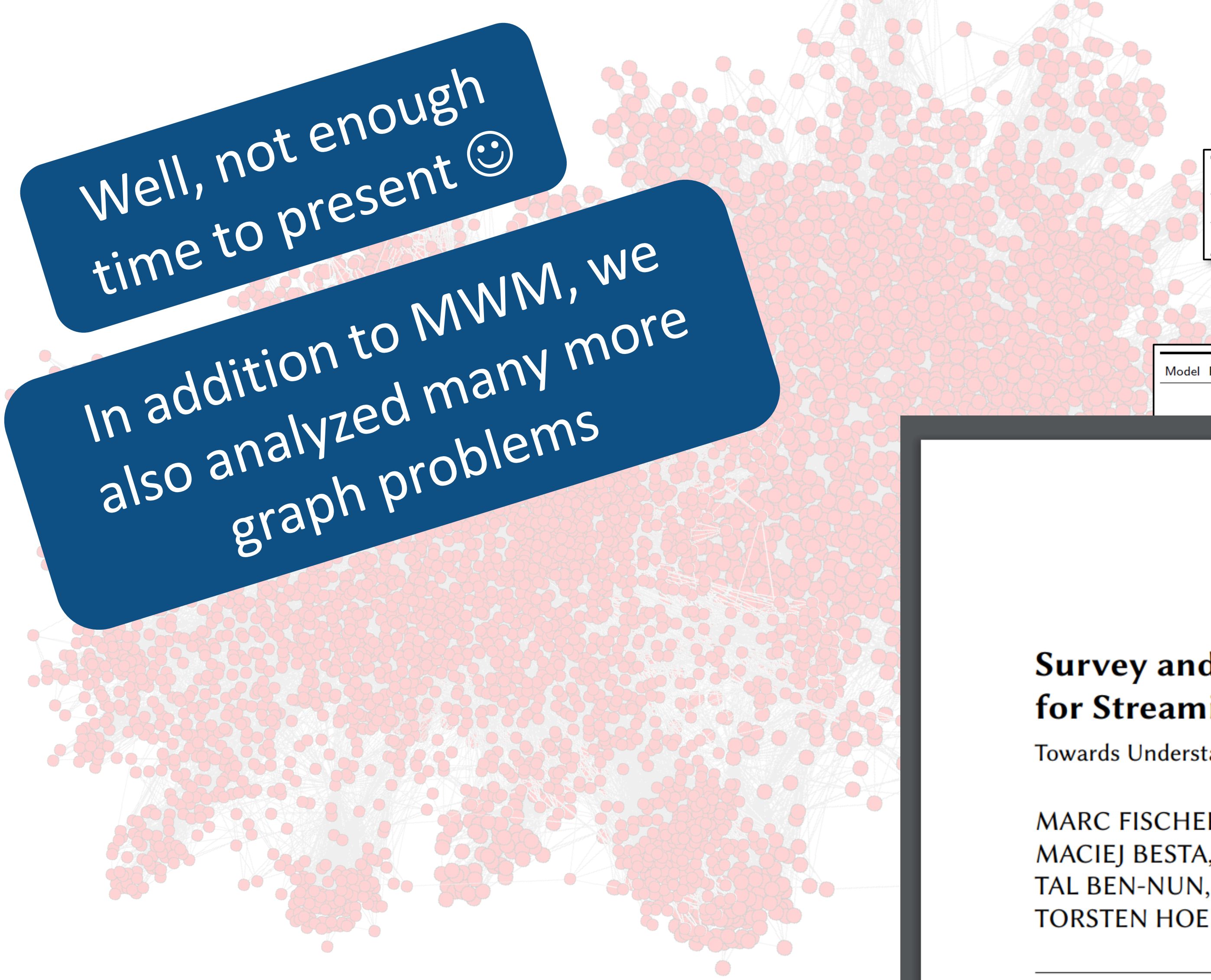
Other Algorithms, Problems, Analyses

Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems



Other Algorithms, Problems, Analyses



Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

Model	Prob.	Input	Order	Reference
ION	Rand.	Unw. Graph	Adv.	[17
DGS	Rand.	Unw. Graph	Adv.	[6,
TN	Rand.	Unw. Graph	Adv.	[57,

									Method
DGS	Rand.	W. Graph	Adv.	[61, Theorem 1, Section 6]	$1+\epsilon$	$O(1/\epsilon^2 n \text{ polylog}(n))$	-	1	Sampling

1

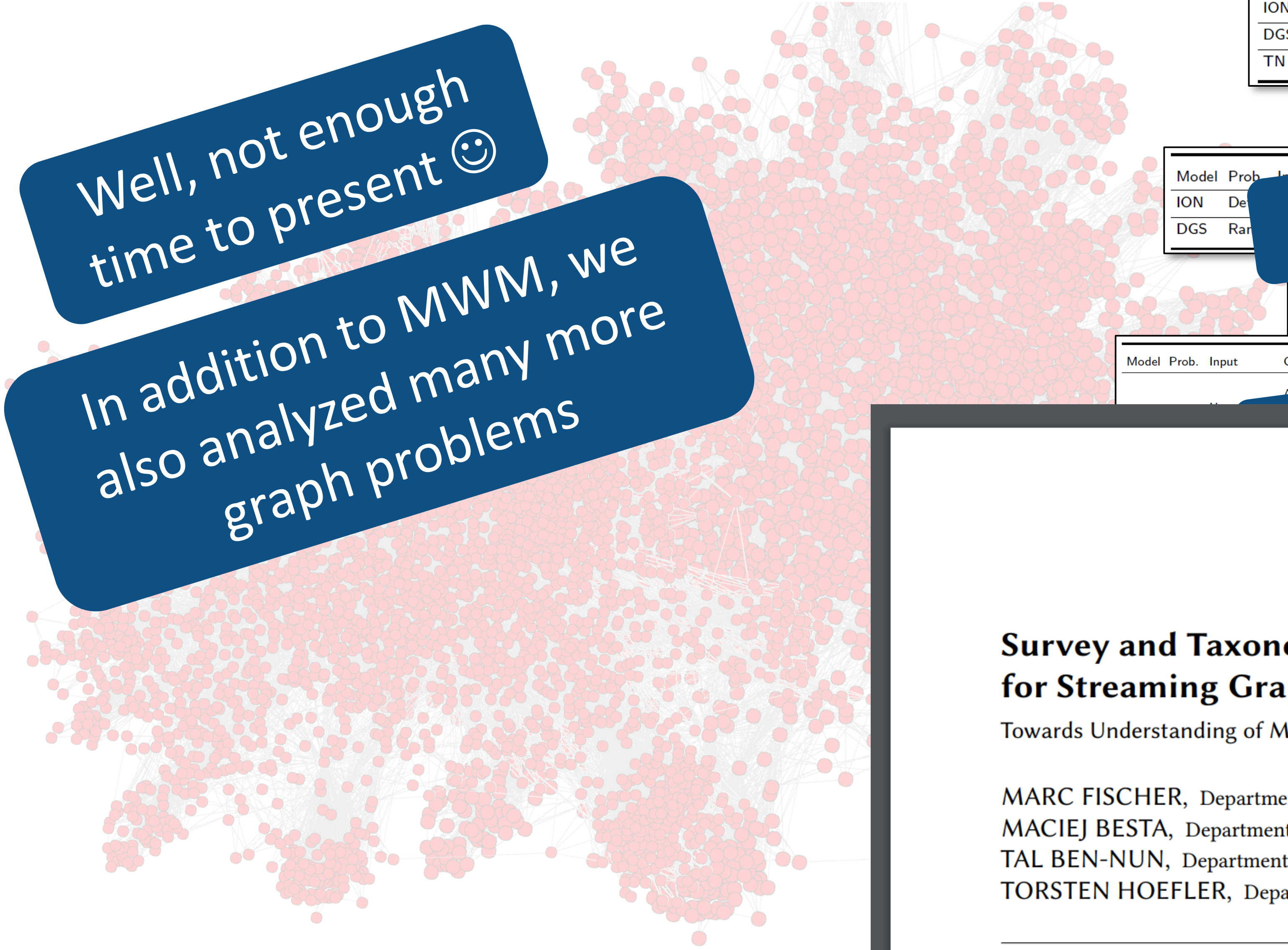
Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich
MACIEJ BESTA, Department of Computer Science, ETH Zurich
TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including machine learning, social network analysis, computational sciences, and others. Two key challenges that hinder accelerating

Other Algorithms, Problems, Analyses



Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

Model	Prob.	Input	Order	Reference
ION	Rand.	Unw. Graph	Adv.	[17
DGS	Rand.	Unw. Graph	Adv.	[6,
TN	Rand.	Unw. Graph	Adv.	[57,

DGS	Rand.	W. Graph	Adv.	[61, Theorem 1, Section 6]	$1+\epsilon$	$O(1/\epsilon^2 n \text{ polylog}(n))$	-	1	Sampling

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Sto

MARC FISCHER, Department of Computer

MACIEJ BESTA, Department of Computer

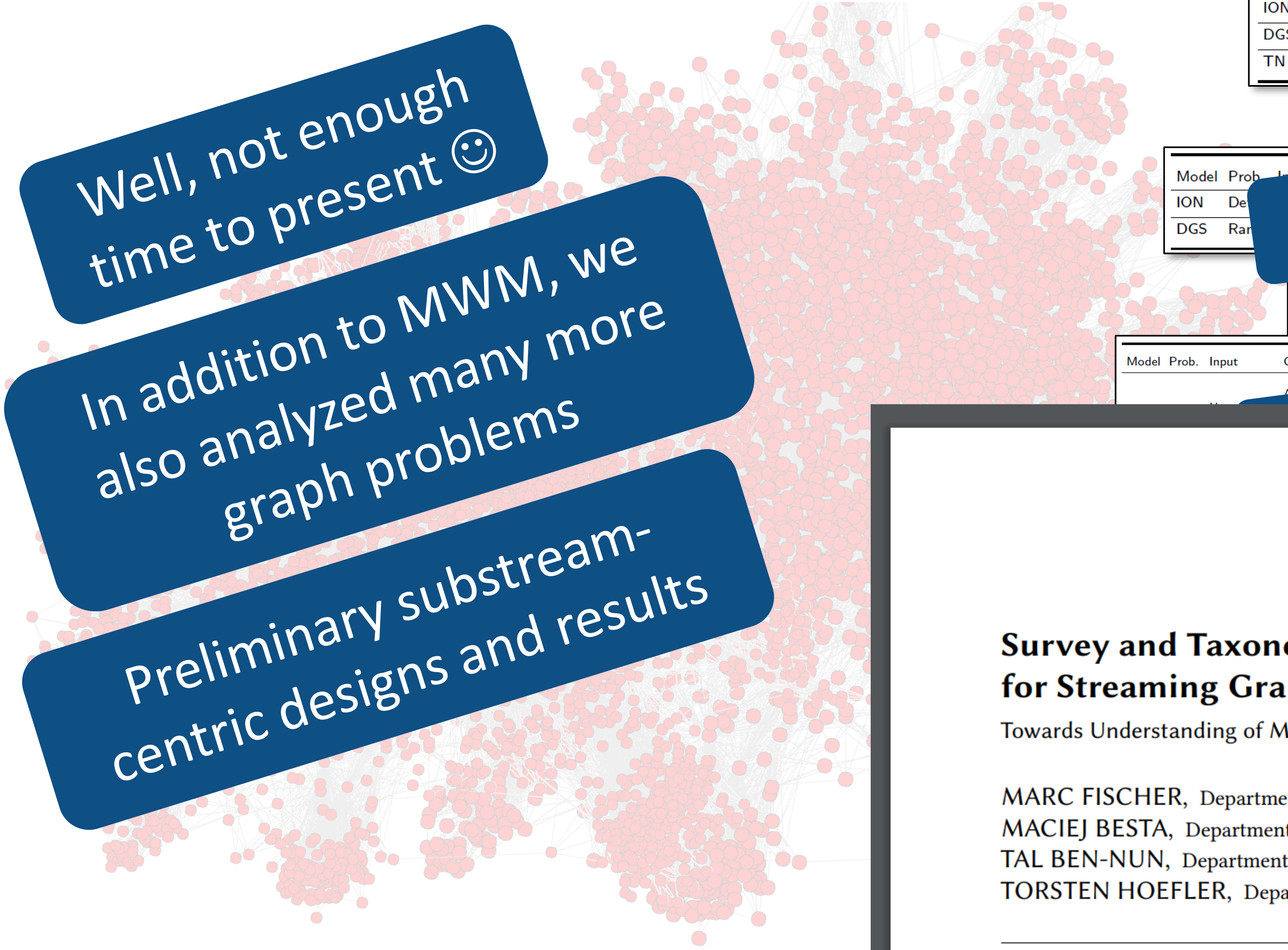
TAL BEN-NUN, Department of Computer S

TORSTEN HOEFLER, Department of Comp

Graph processing has become an important part of va ... including machine learning, social network analysis, computational sciences, ... challenges that hinder accelerating graph processing are (1) sizes of input datasets, reach ... of edges, and (2) the growing rate of graph

Again, the most relevant parts are in the FPGA paper, the rest in this survey (ready in ~1-2 months)

OTHER ALGORITHMS, PROBLEMS, ANALYSES



Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

Preliminary substream-centric designs and results

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Sto…

MARC FISCHER, Department of Computer…
MACIEJ BESTA, Department of Computer…
TAL BEN-NUN, Department of Computer S…
TORSTEN HOEFLER, Department of Comp…

Graph processing has become an important part of va… …including machine learn-ing, social network analysis, computational sciences, … challenges that hinder accelerating

Again, the most relevant parts are in the FPGA paper, the rest in this survey (ready in ~1-2 months)

Other Algorithms, Problems, Analyses

Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

Preliminary substream-centric designs and results

Work in progress on the distributed setting ☺

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Stor...

MARC FISCHER, Department of Computer...
MACIEJ BESTA, Department of Computer...
TAL BEN-NUN, Department of Computer S...
TORSTEN HOEFLER, Department of Comp...

Graph processing has become an important part of va... including machine learning, social network analysis, computational sciences, ... challenges that hinder accelerating graph processing are (1) sizes of input datasets, reach... of edges, and (2) the growing rate of graph...

Again, the most relevant parts are in the FPGA paper, the rest in this survey (ready in ~1-2 months)

Other Algorithms, Problems, Analyses

Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

…y substream-… …sults

Work in progress on the distributed setting ☺

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Sto…

MARC FISCHER, Department of Computer…
MACIEJ BESTA, Department of Computer…
TAL BEN-NUN, Department of Computer S…
TORSTEN HOEFLER, Department of Comp…

Graph processing has become an important part of va… …, including machine learning, social network analysis, computational sciences, … …challenges that hinder accelerating graph processing are (1) sizes of input datasets, reach… …of edges, and (2) the growing rate of graph…

Again, the most relevant parts are in the FPGA paper, the rest in this survey (ready in ~1-2 months)

OTHER ALGORITHMS, PROBLEMS, ANALYSES

Well, not enough time to present ☺

In addition to MWM, we also analyzed many more graph problems

Work in progress on the distributed setting ☺

Triangle counting

Connected components

Motif counting

Minimum spanning trees

Random walks

Triangle counting

Connectivity

☺ https://arxiv.org/abs/...

1

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich
JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich
TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Sto...

MARC FISCHER, Department of Computer...
MACIEJ BESTA, Department of Computer...
TAL BEN-NUN, Department of Computer S...
TORSTEN HOEFLER, Department of Comp...

Graph processing has become an important part of va... ...including machine learn-ing, social network analysis, computational sciences, ... challenges that hinder accelerating graph processing are (1) sizes of input datasets, reach... of edges, and (2) the growing rate of graph...

Again, the most relevant parts are in the FPGA paper, the rest in this survey (ready in ~1-2 months)

MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge...

Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... *e = (u, v, weight)*

MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

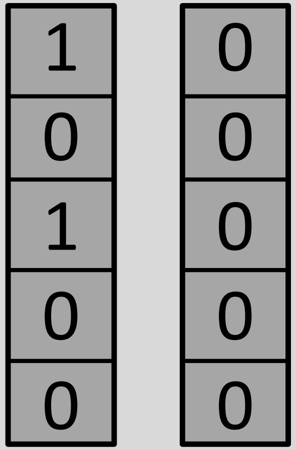
An incoming edge... *e = (u, v, weight)*

Vertices + matchings (correctness)

Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

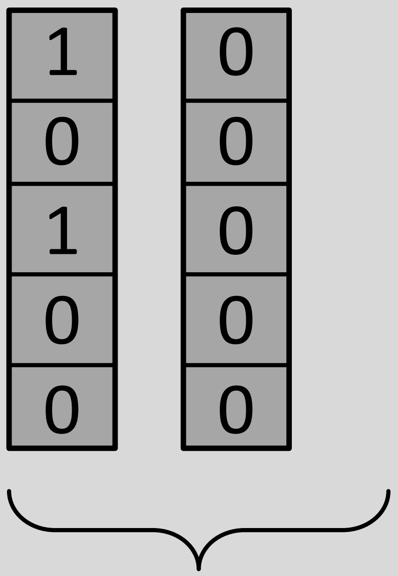
Vertices + matchings (correctness)



Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... *e = (u, v, weight)*

Vertices + matchings (correctness)

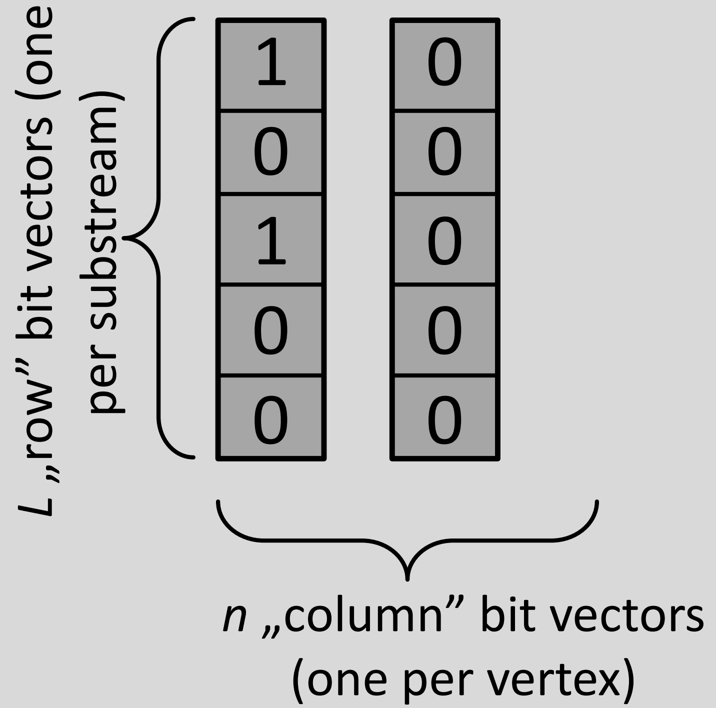


n „column” bit vectors
(one per vertex)

MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

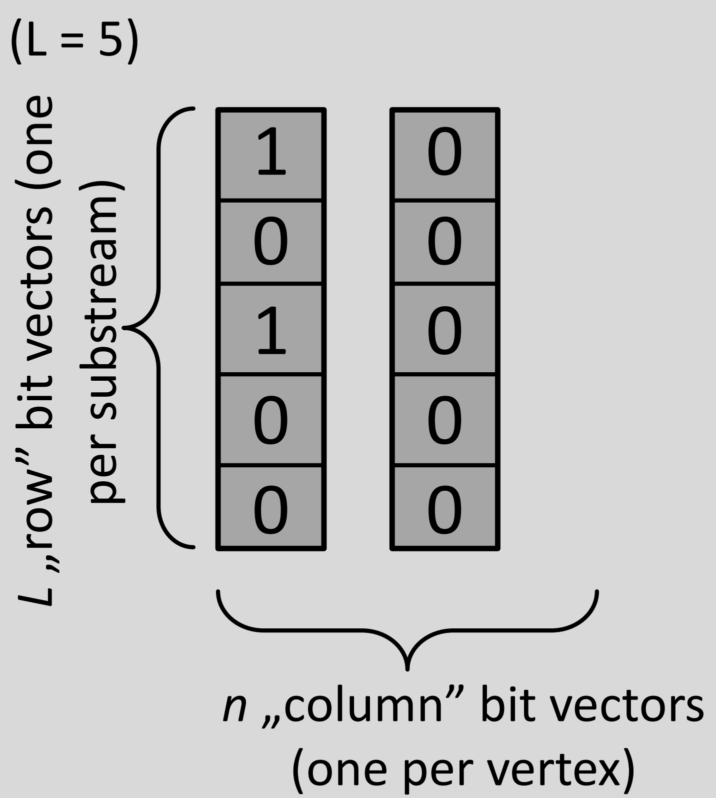
Vertices + matchings (correctness)



MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

Vertices + matchings (correctness)

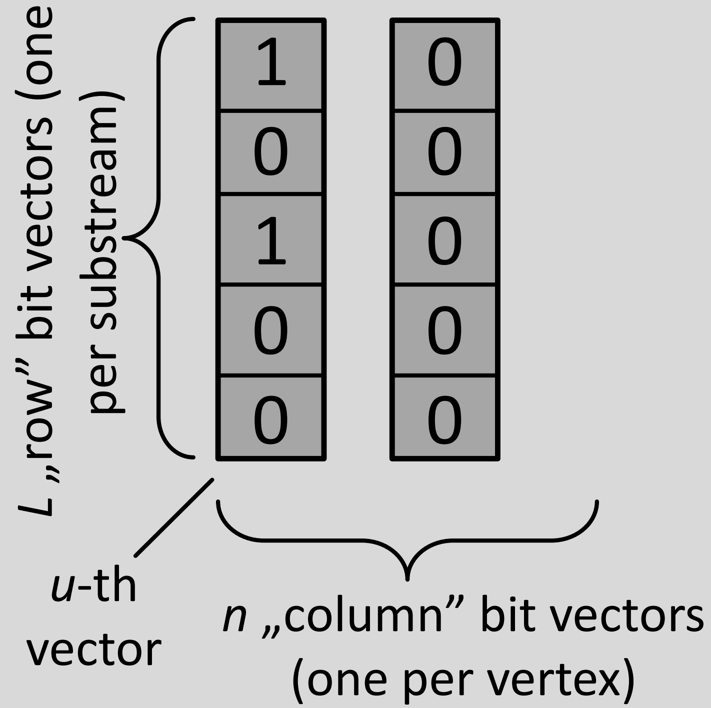


Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

Vertices + matchings (correctness)

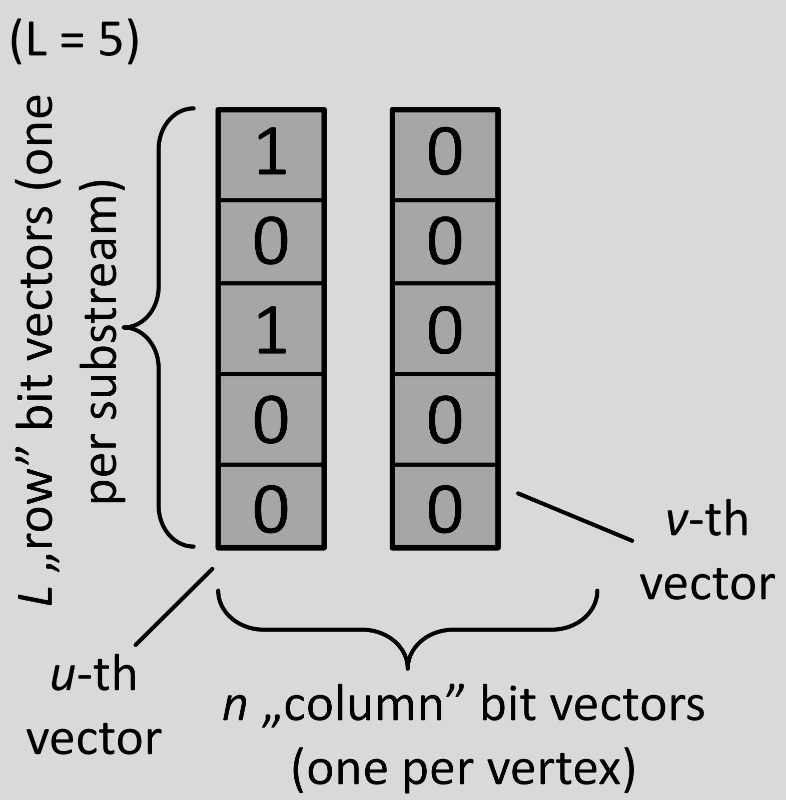
(L = 5)



MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

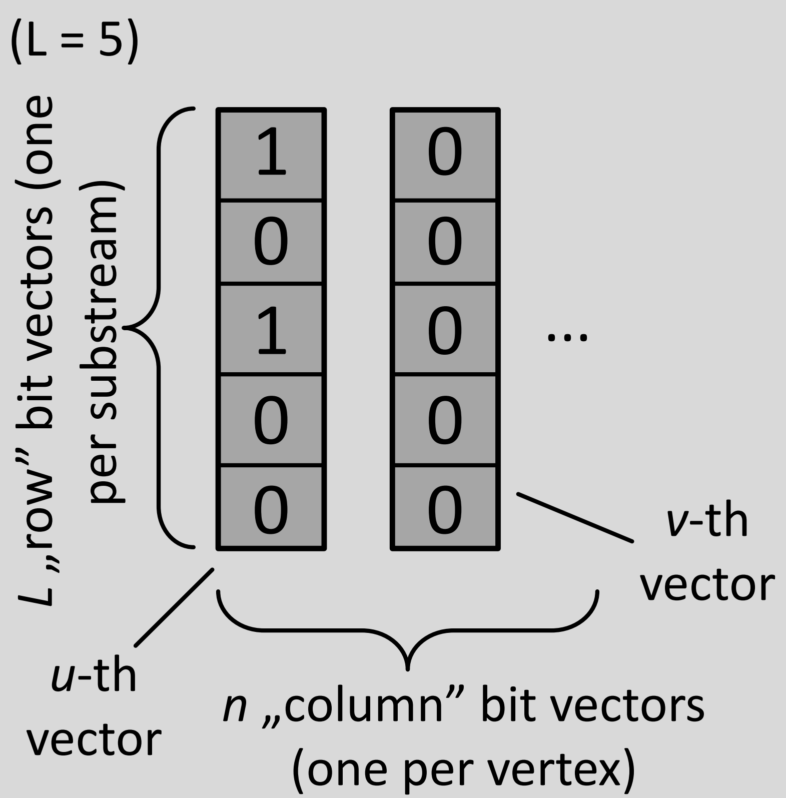
Vertices + matchings (correctness)



MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

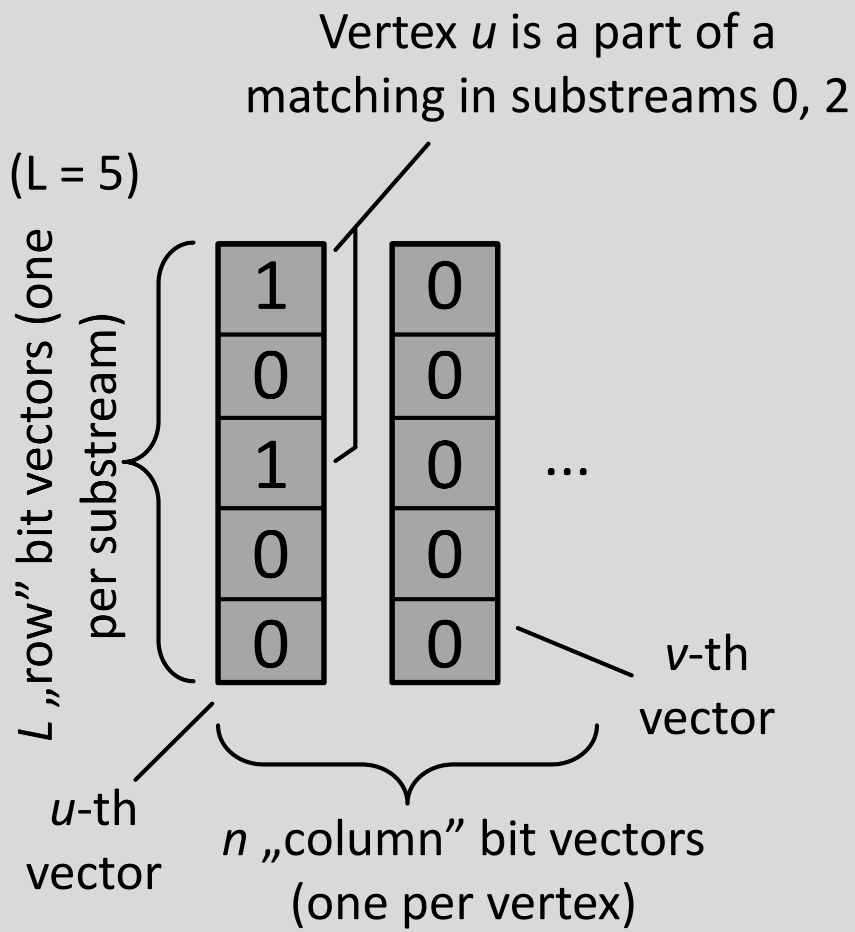
Vertices + matchings (correctness)



MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

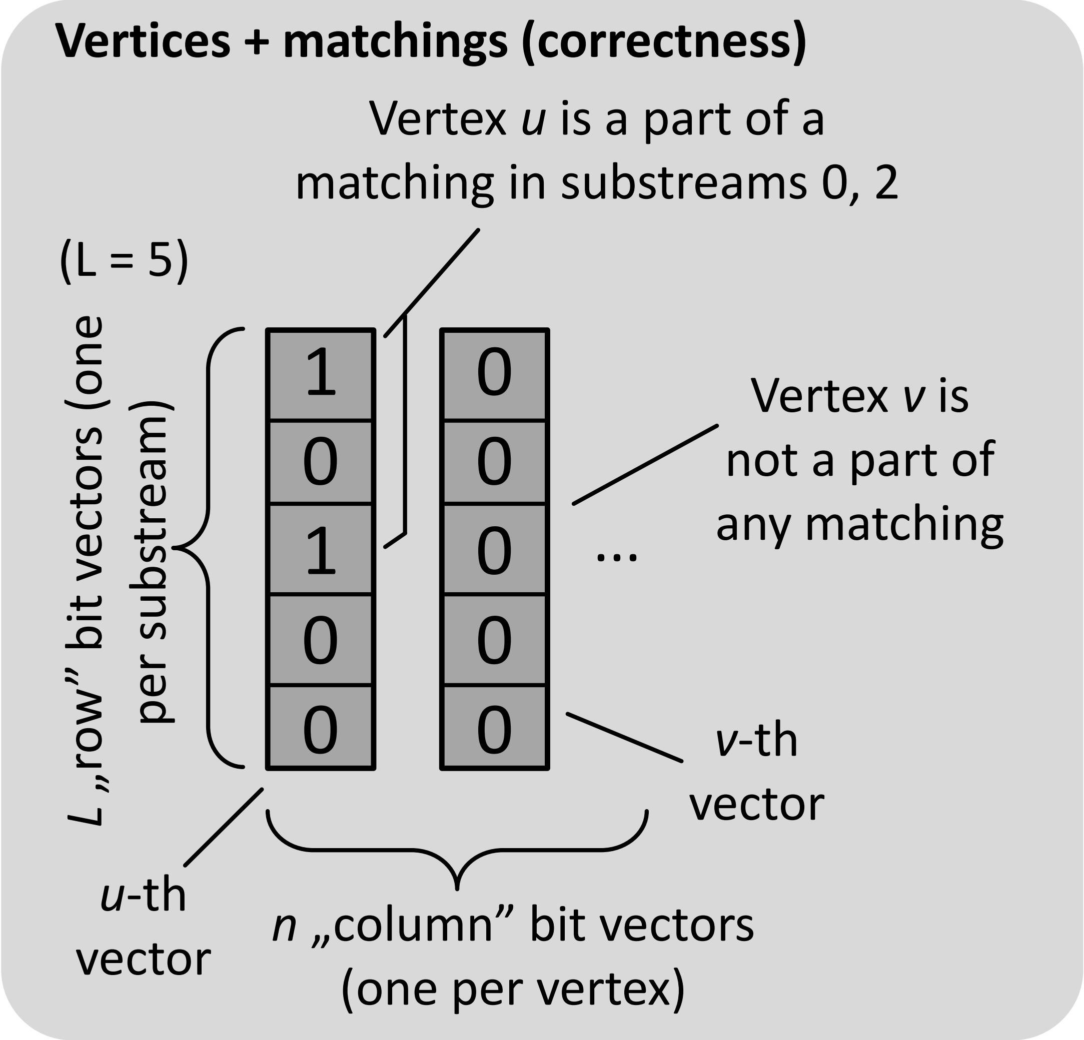
Vertices + matchings (correctness)



Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

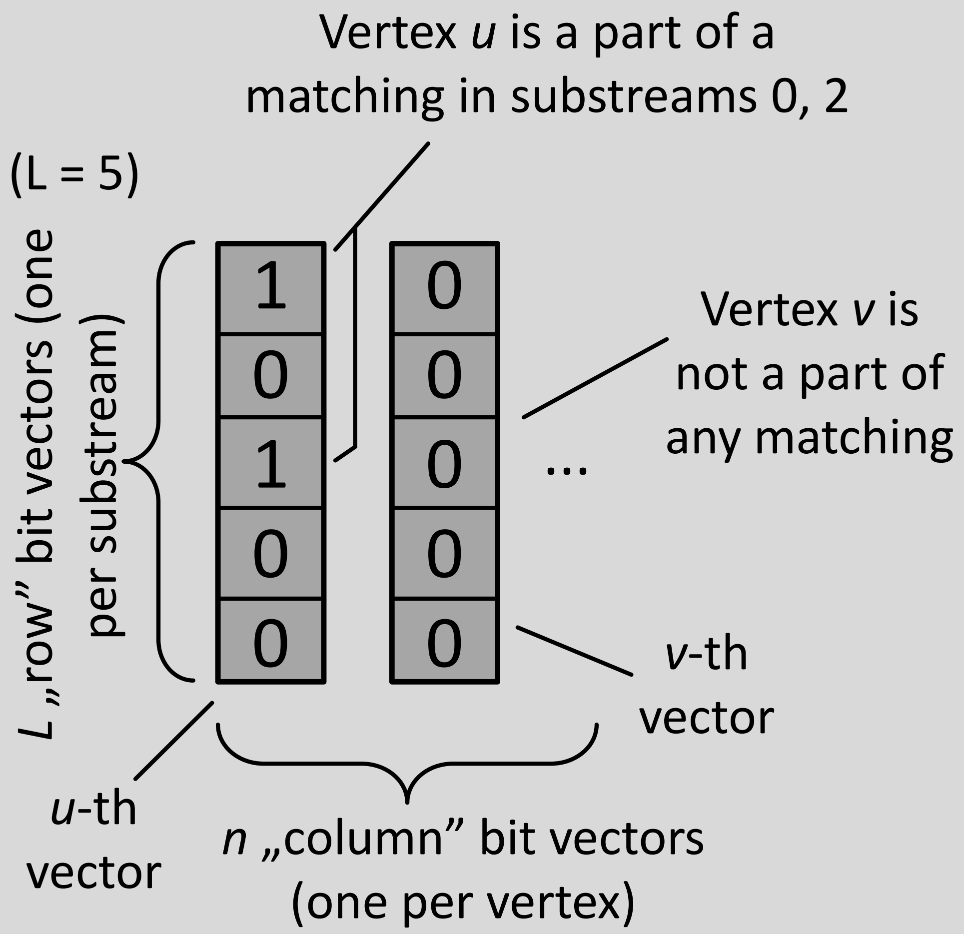
Vertices + matchings (correctness)



Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

Vertices + matchings (correctness)

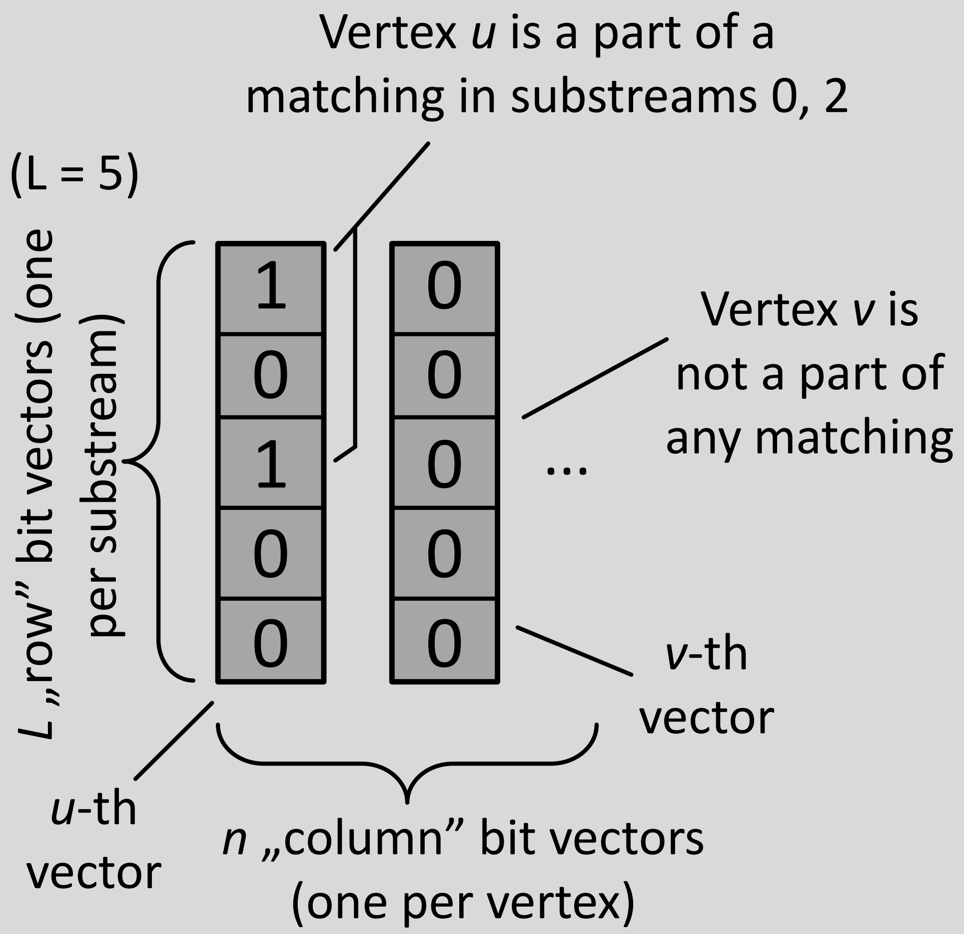


Edges + matchings (more performance)

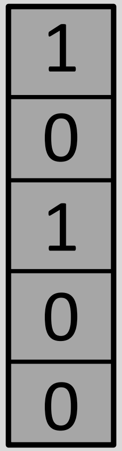
Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... *e = (u, v, weight)*

Vertices + matchings (correctness)



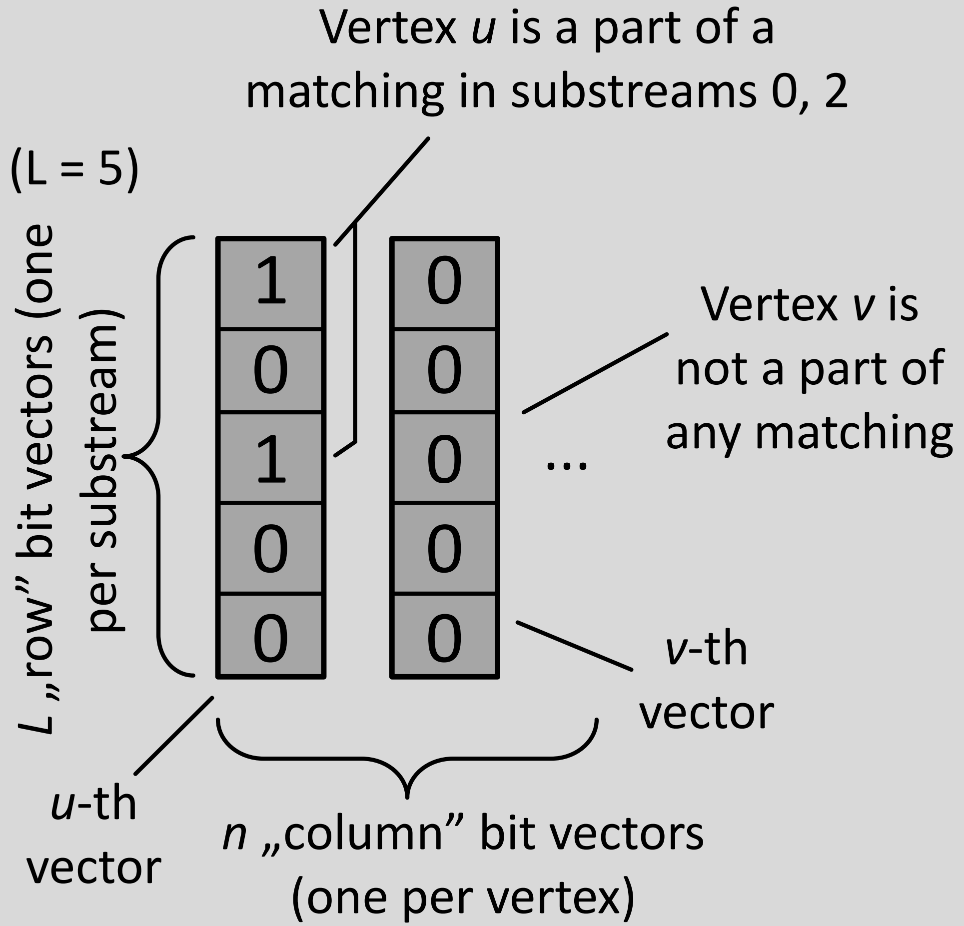
Edges + matchings (more performance)



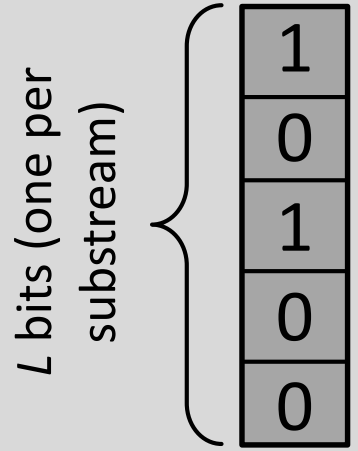
Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

Vertices + matchings (correctness)

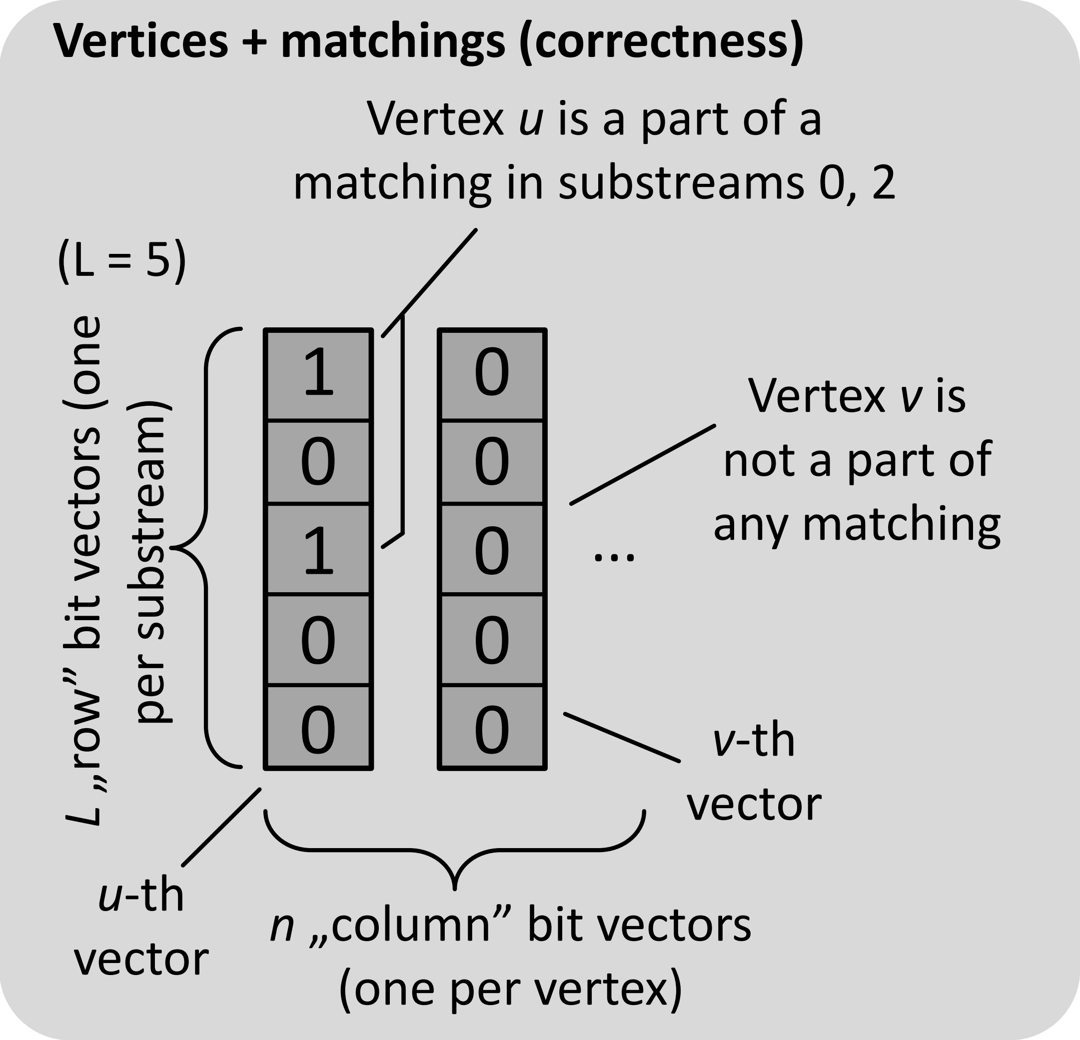


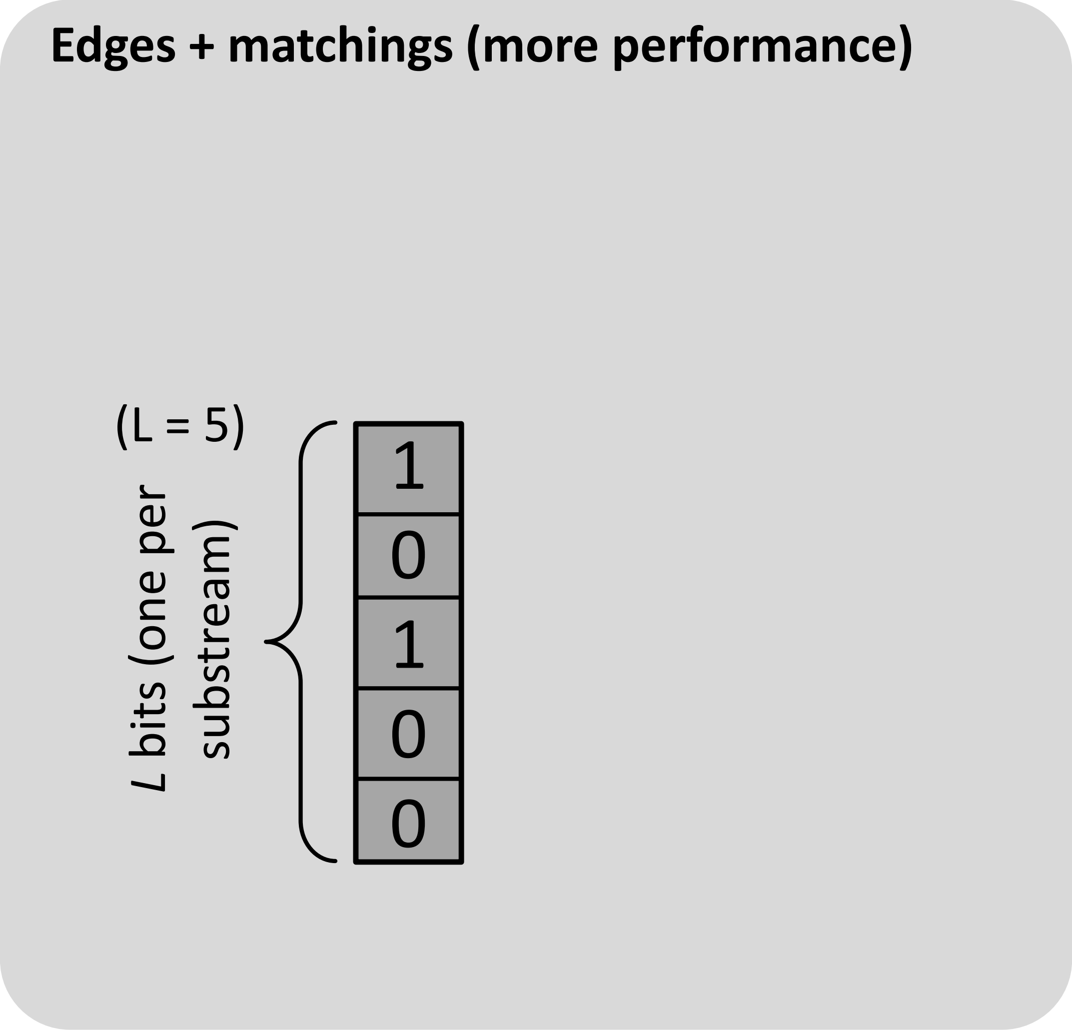
Edges + matchings (more performance)



Matching Bits: Key Data Structures for Maintaining Information on Matchings

An incoming edge... $e = (u, v, weight)$

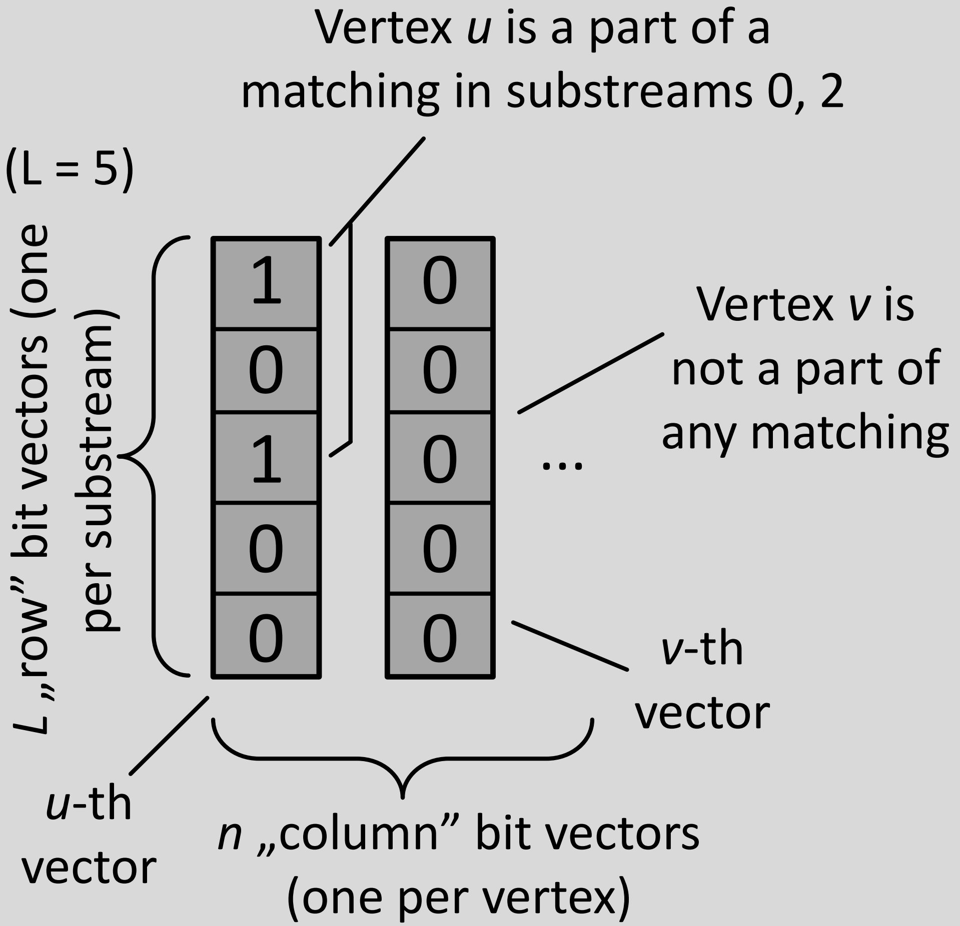




Matching Bits: Key Data Structures for Maintaining Information on Matchings

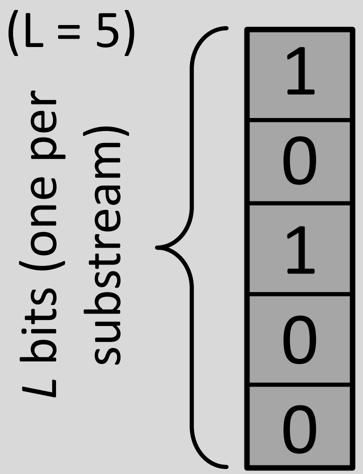
An incoming edge... $e = (u, v, weight)$

Vertices + matchings (correctness)



Edges + matchings (more performance)

For the current edge, is...



MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

An incoming edge... *e = (u, v, weight)*

Vertices + matchings (correctness)

Vertex *u* is a part of a matching in substreams 0, 2

(L = 5)

L „row" bit vectors (one per substream)

u-th vector	*v*-th vector
1	0
0	0
1	0
0	0
0	0

...

Vertex *v* is not a part of any matching

u-th vector

v-th vector

n „column" bit vectors (one per vertex)

Edges + matchings (more performance)

For the current edge, is...

(L = 5)

L bits (one per substream)

Bit	
1	$weight \geq (1+\epsilon)^0$?
0	$weight \geq (1+\epsilon)^1$?
1	$weight \geq (1+\epsilon)^2$?
0	$weight \geq (1+\epsilon)^3$?
0	$weight \geq (1+\epsilon)^4$?

Performance Analysis

Various #substreams (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size (K) = 32,
#threads = 4,
#edges = 16M
(Kronecker), ε = 0.1

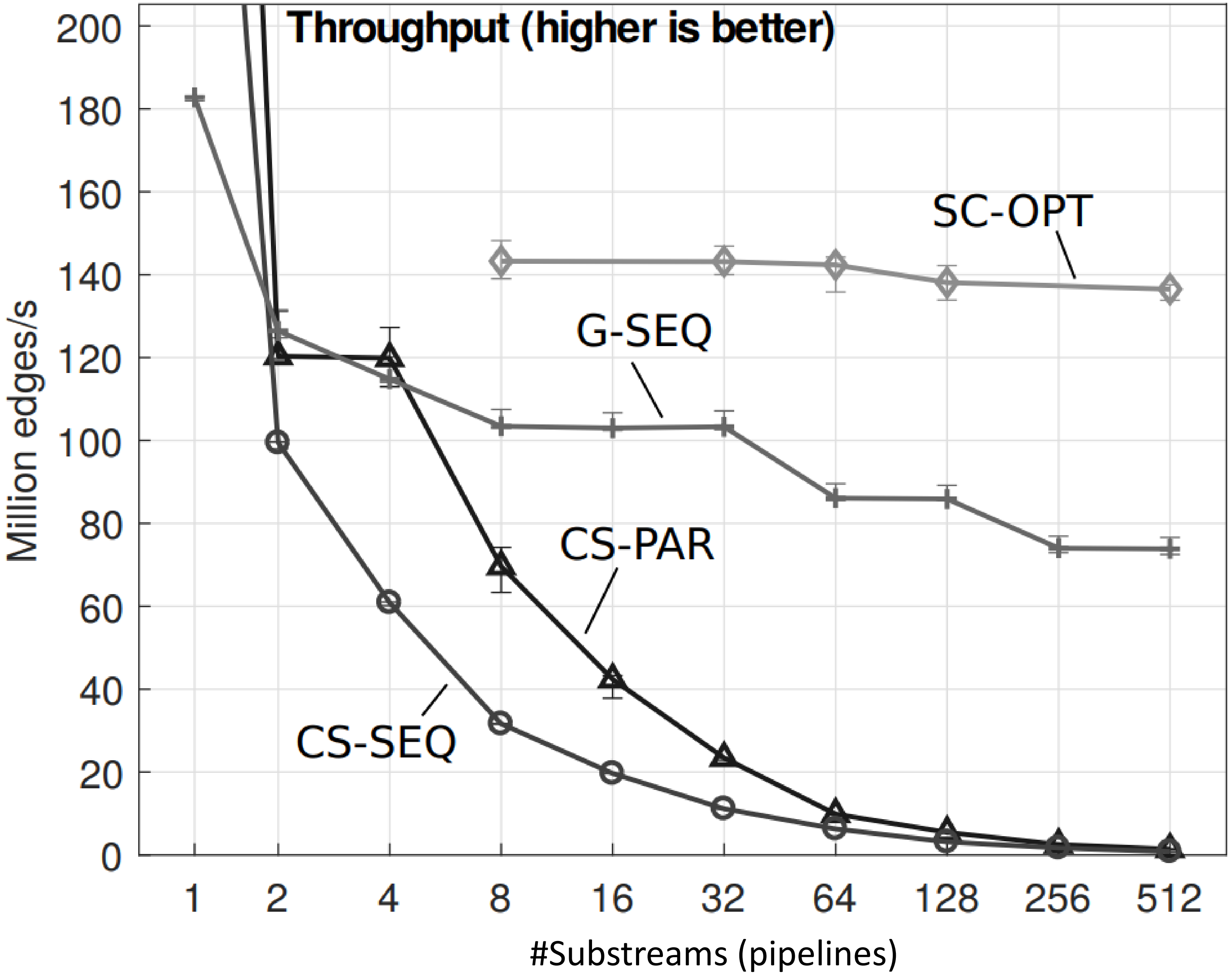
#Substreams (pipelines)

Performance Analysis

Various #substreams (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

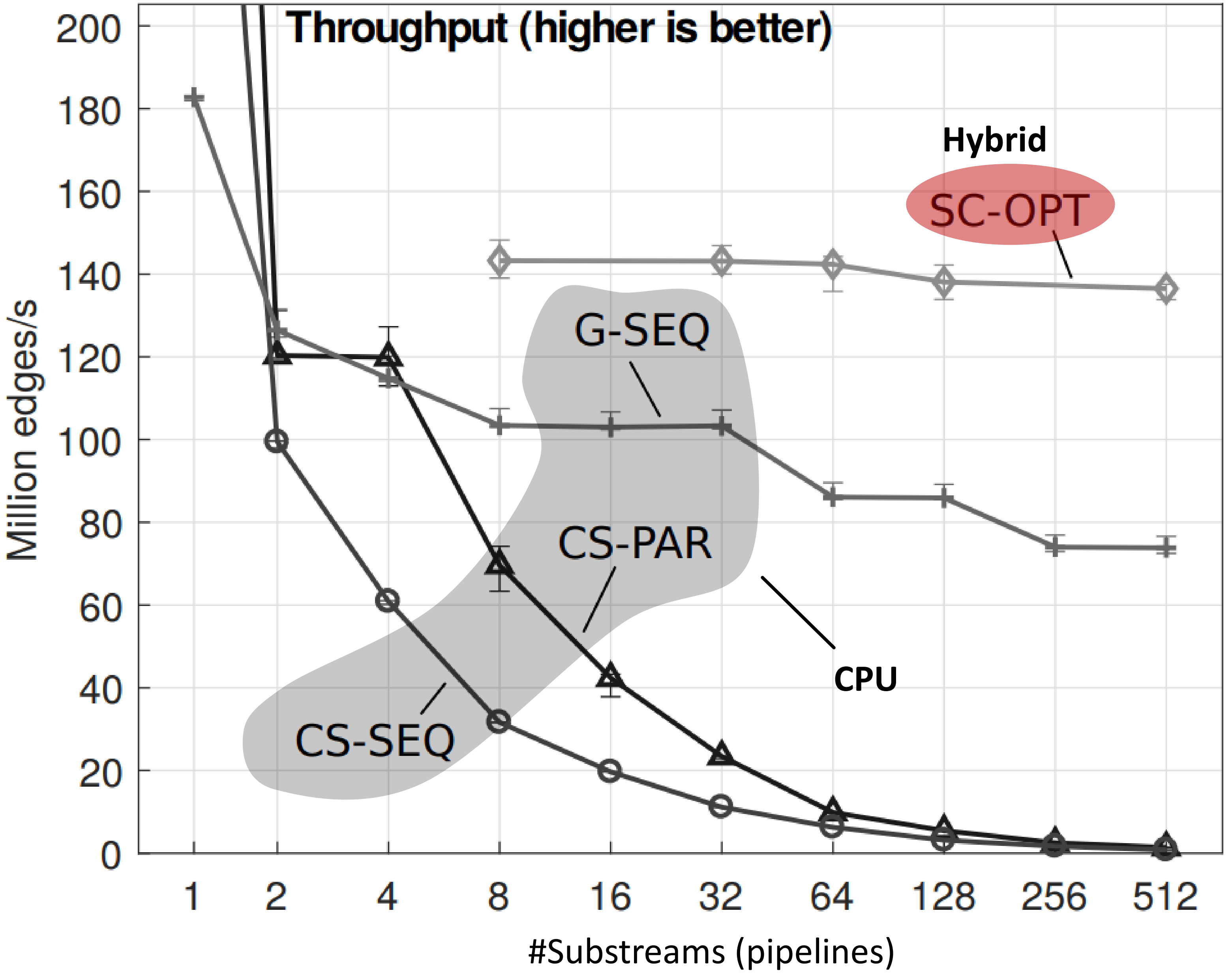
Parameters:
Blocking size (K) = 32,
#threads = 4,
#edges = 16M
(Kronecker), ε = 0.1



Performance Analysis
Various #substreams (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size (K) = 32,
#threads = 4,
#edges = 16M
(Kronecker), ε = 0.1

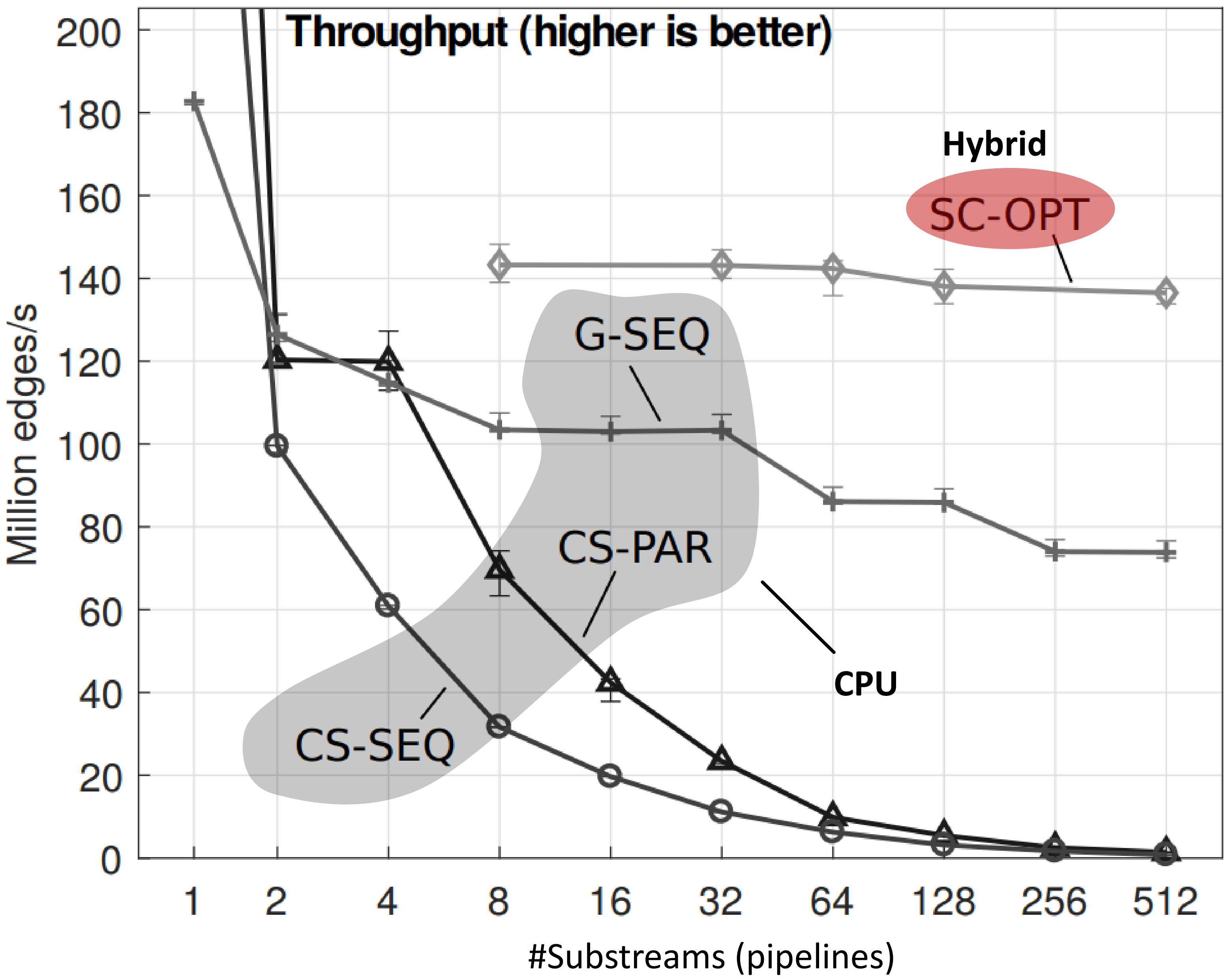


Performance Analysis
Various #substreams (L)

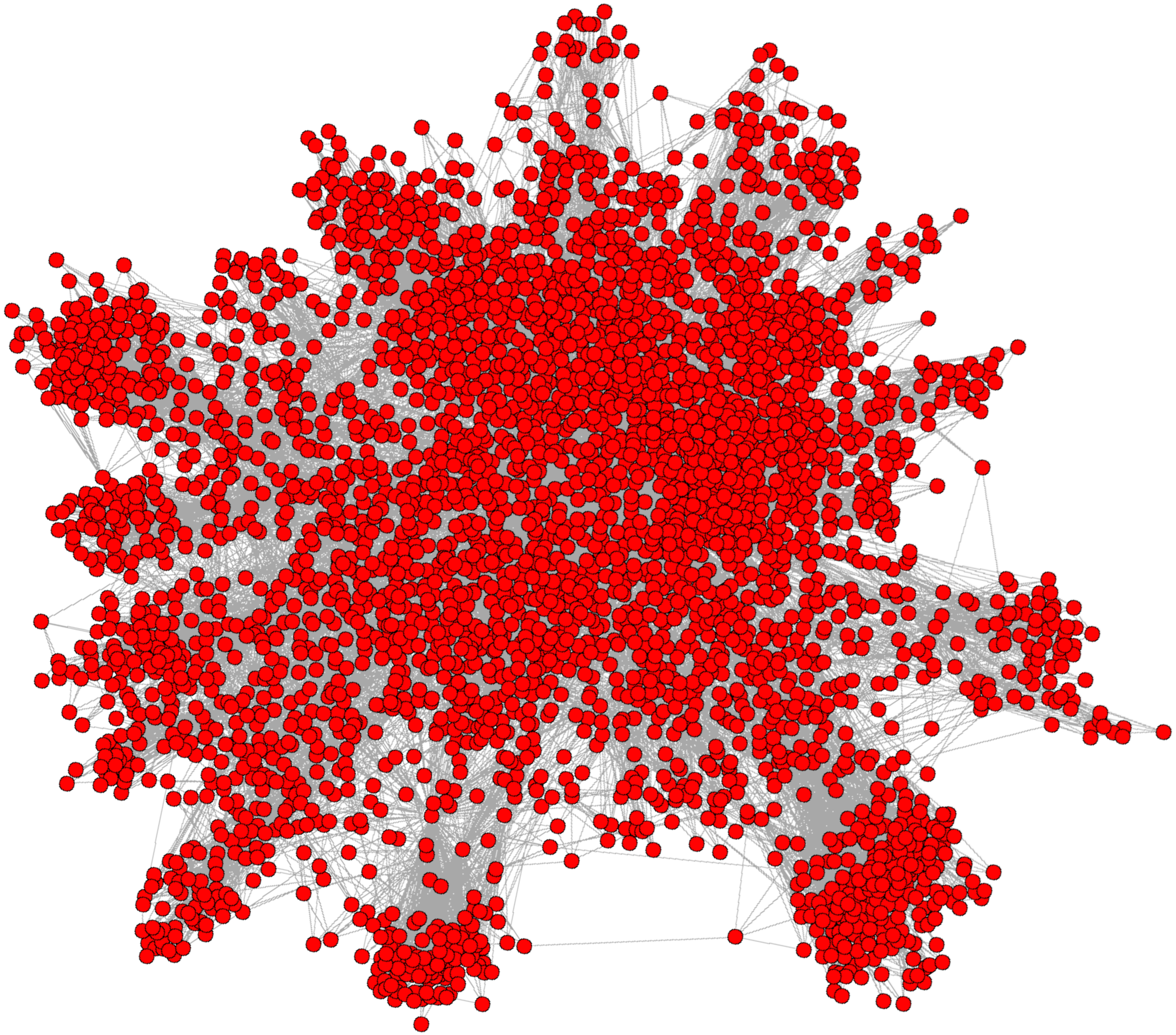
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:
Blocking size (K) = 32,
#threads = 4,
#edges = 16M
(Kronecker), ε = 0.1

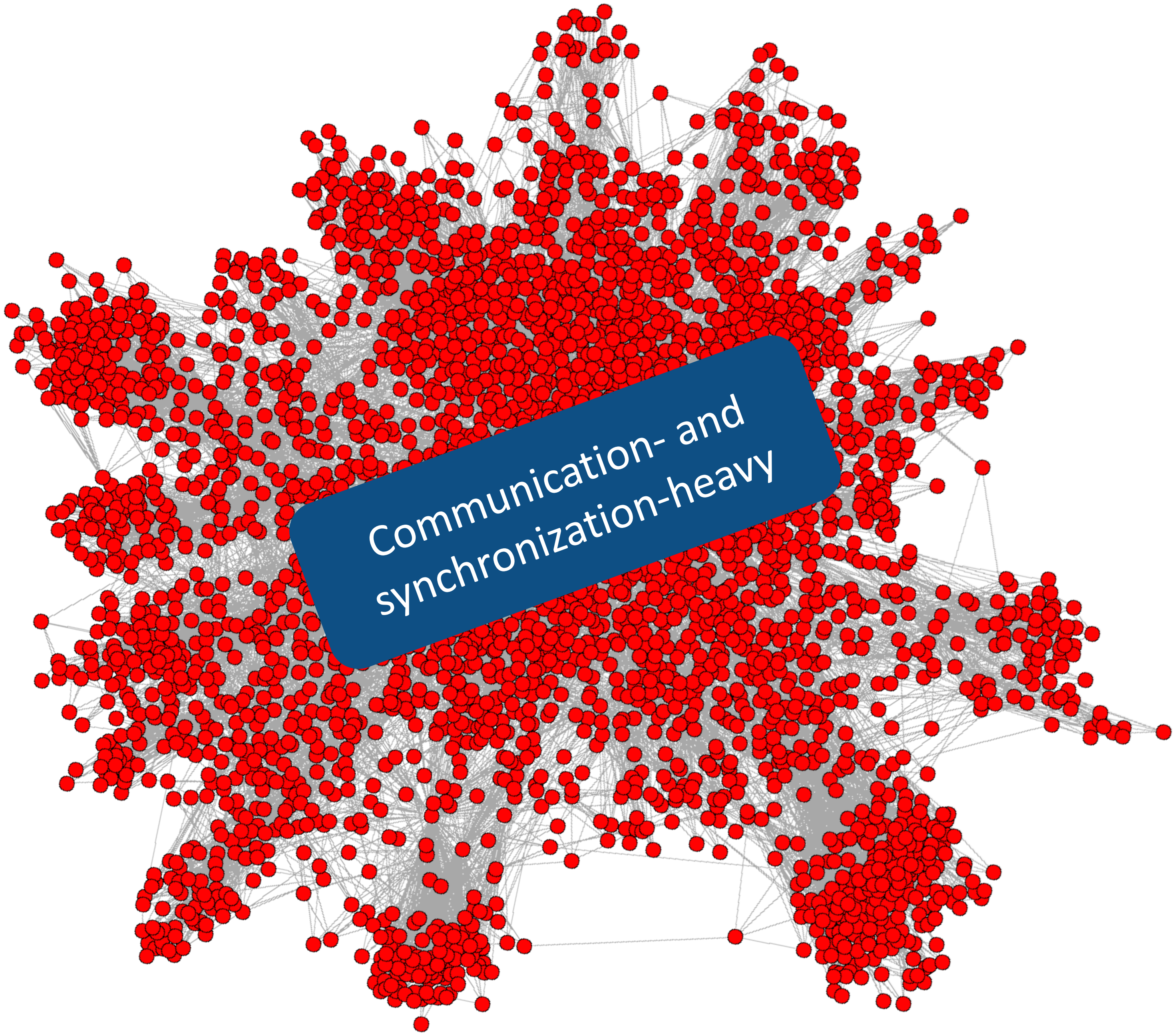
SC-OPT secures highest performance for all considered values of parameters



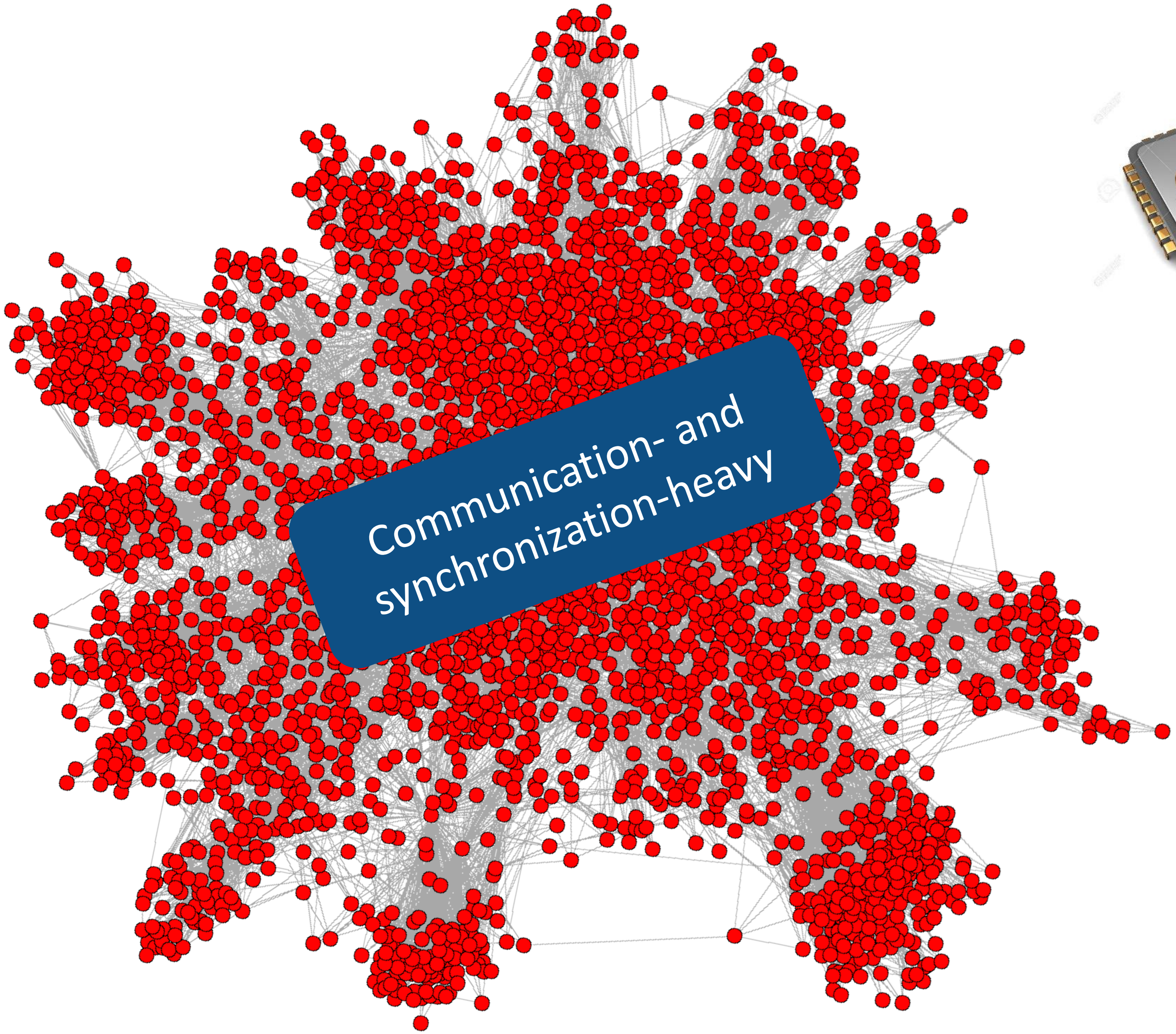
Large graphs…



Large graphs…



Large graphs…



Large graphs…