Bringing HPC Graph Analytics to Modern Graph Databases

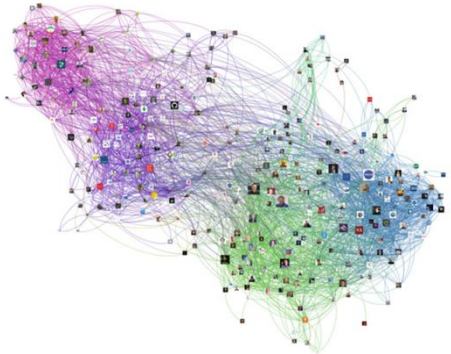
PPAM 2022
14th International Conference on Parallel Processing and Applied Mathematics
Sep 12, 2022

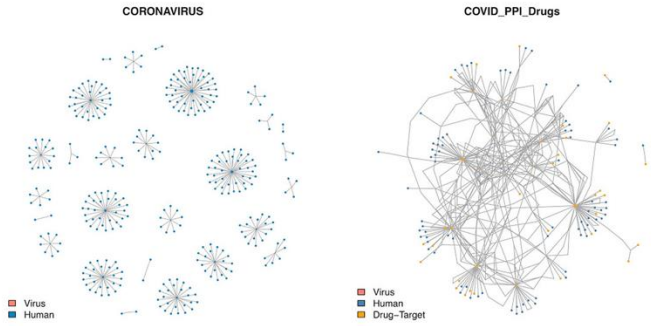
Ümit V. Çatalyürek
Amazon Scholar, Amazon Web Services
Professor, School of Computational Science and Engineering
Georgia Institute of Technology

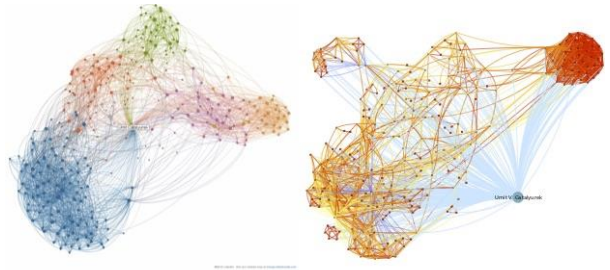
Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

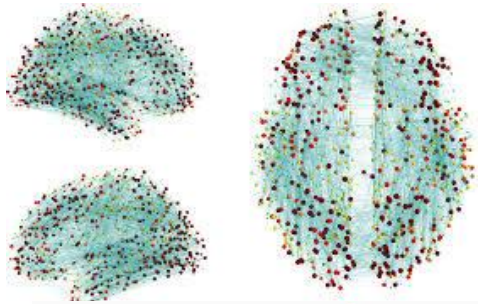
Graphs are Ubiquitous

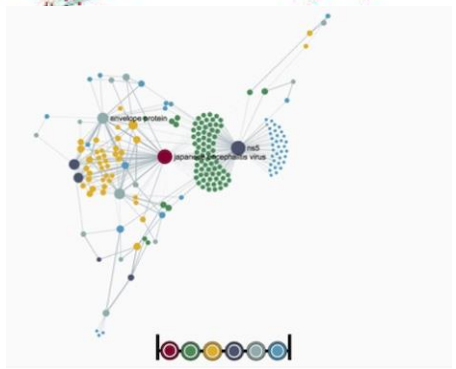






Linkedin Ego-Net DBLP Ego-Net





They are growing. Up to billions of vertices and edges

Fast, efficient analysis is important and pervasive

Many graph processing frameworks, and databases, have been proposed/developed

Image credits:

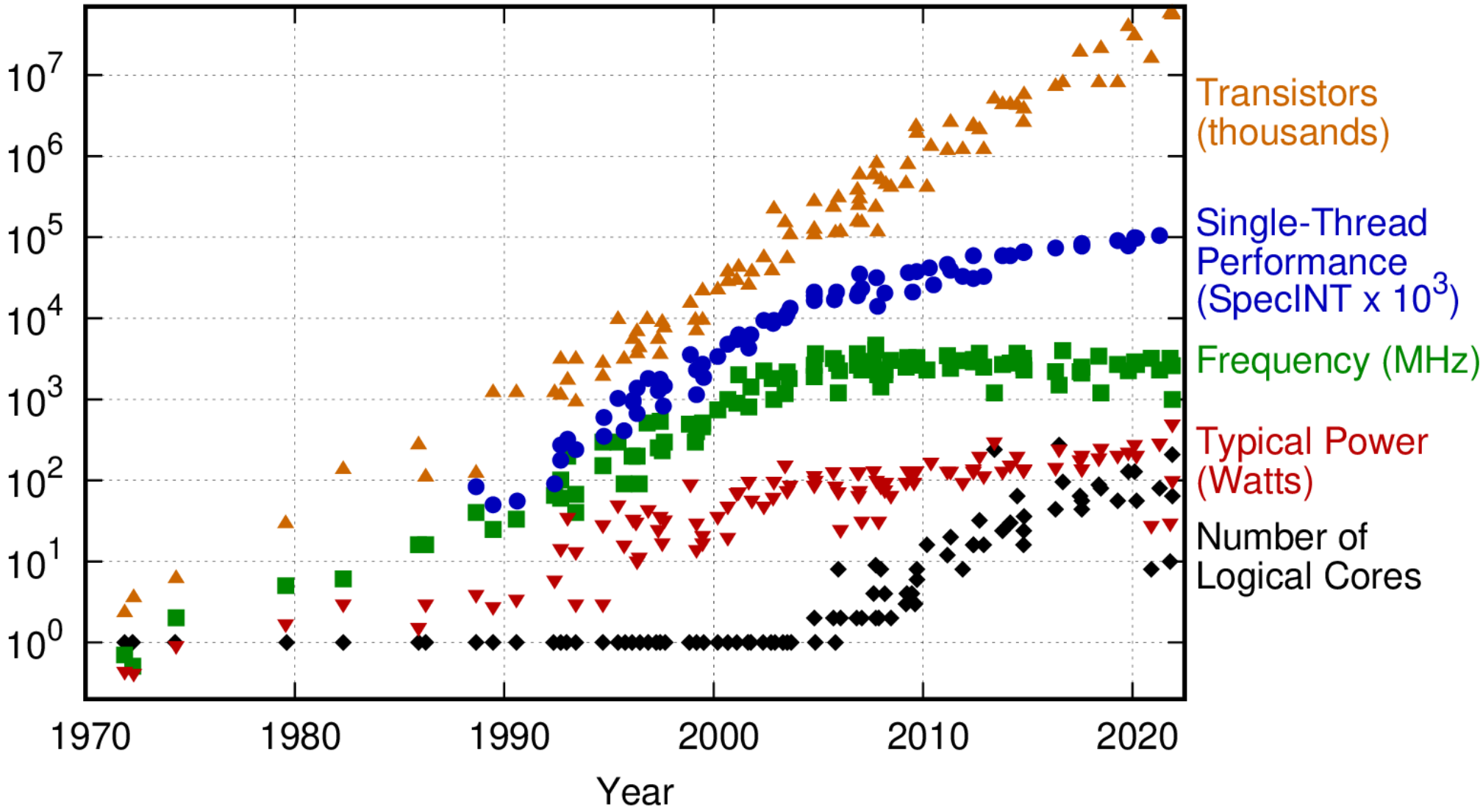
Jenn Caulfield, Social network vector illustration, 2018

Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011

Albert-László Barabási/BarabasiLab 2019

Caleb Jonson, How to Visualize Your Twitter Network, 2014

Why HPC - Hardware Motivation: 50 Years of Microprocessors

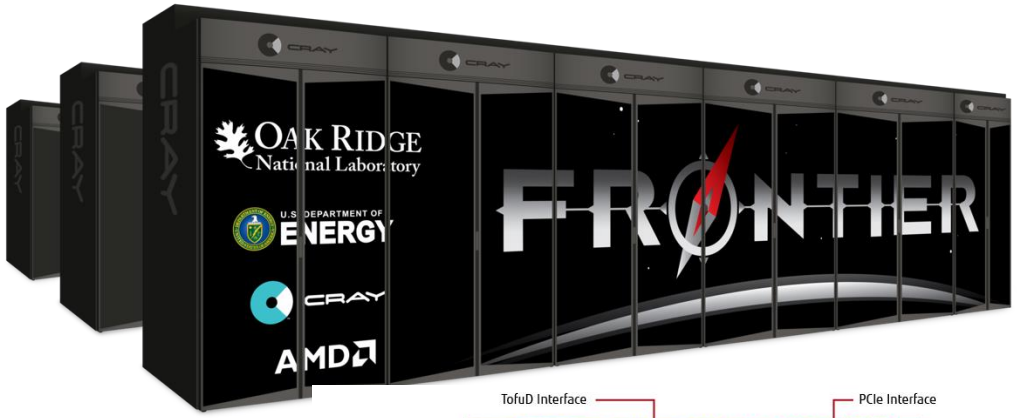


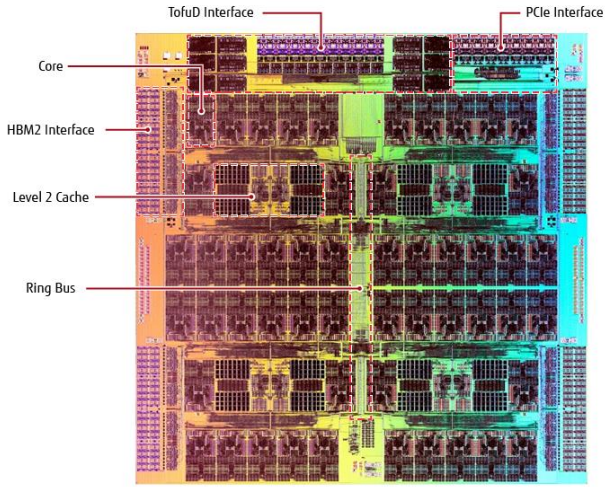
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

https://github.com/karlrupp/microprocessor-trend-data

Why HPC - Hardware Motivation: Current & Future Systems

- More and more machines composed of multi-core and many-core CPUs, and accelerators
- June 2022: Top500's top 10 has 9 with many-core CPU/Accelerators (7 GPUs, Maxtrix-2000, SW26010)
- Some examples:
  - Frontier: 8,730,112 cores (9,408 nodes)
    - 64-core AMD EPYC, 4 AMD MI250X GPUs
  - Fugaku: 7,630,848 cores (>150K nodes):
    - Fujitsu's 48-core A64FX SoC
  - Summit: 2,414,592 cores (4,608 nodes)
    - 2 IBM Power9 22 Cores, 6 NVIDIA Volta GV100
  - Intel Xeon E7-8890 V4
    - 24 cores, 2 threads per core, 2 AVX 512 vector processing units/core
  - SW26010 chip is a Chinese "homegrown" many-core (260 core) processor
    - 4 cluster of 64 CPEs+ 1 MPE
  - NVIDIA A100
    - 6,912 cores, 432 tensor cores, 9.7 / 19.5 TFLOPS (FP64 / FP64 Tensor Core)
- *Cloud* Instances also have "similar" GPUs and and many-core CPUs.



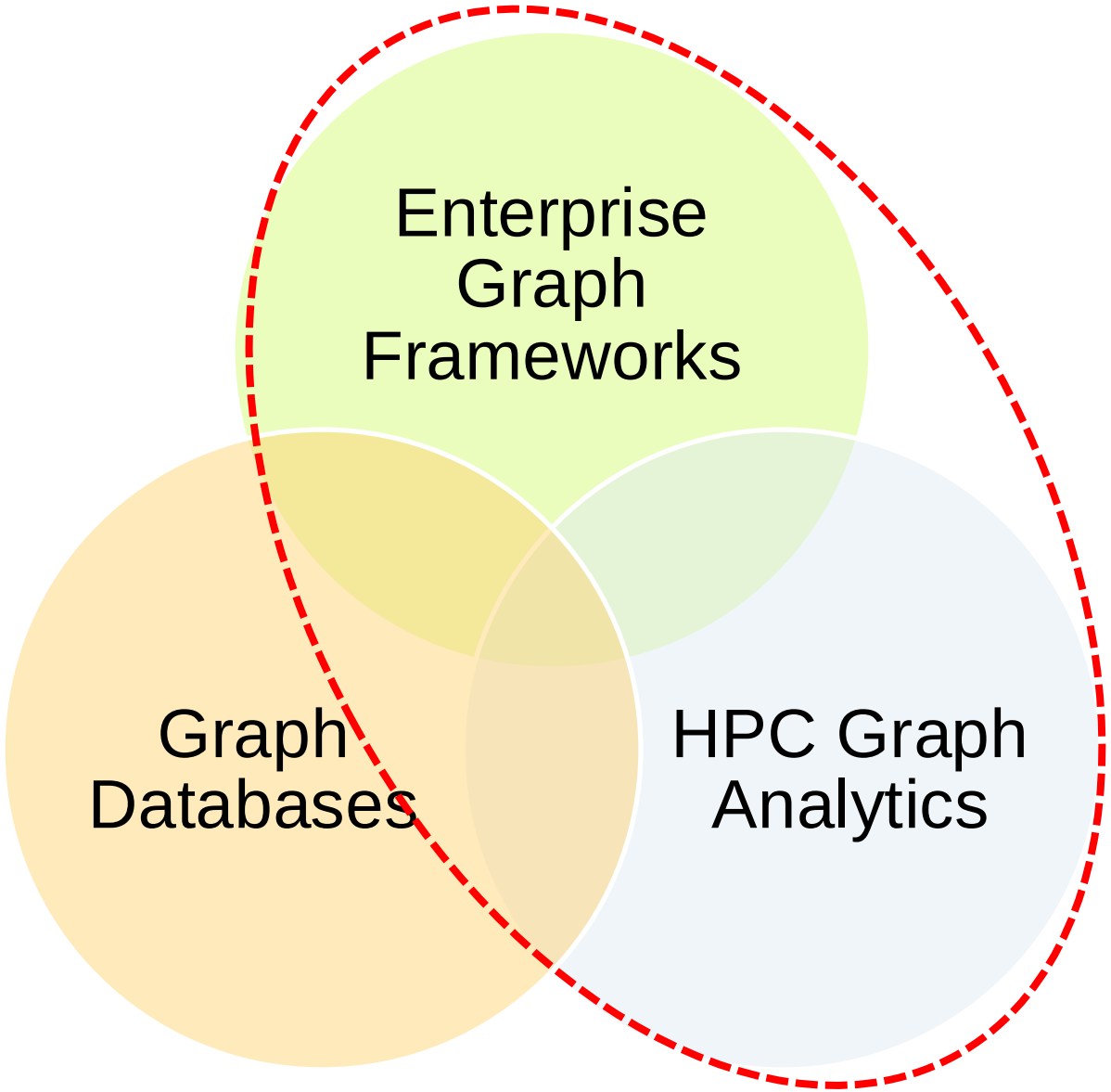


Die photo of A64FX CPU

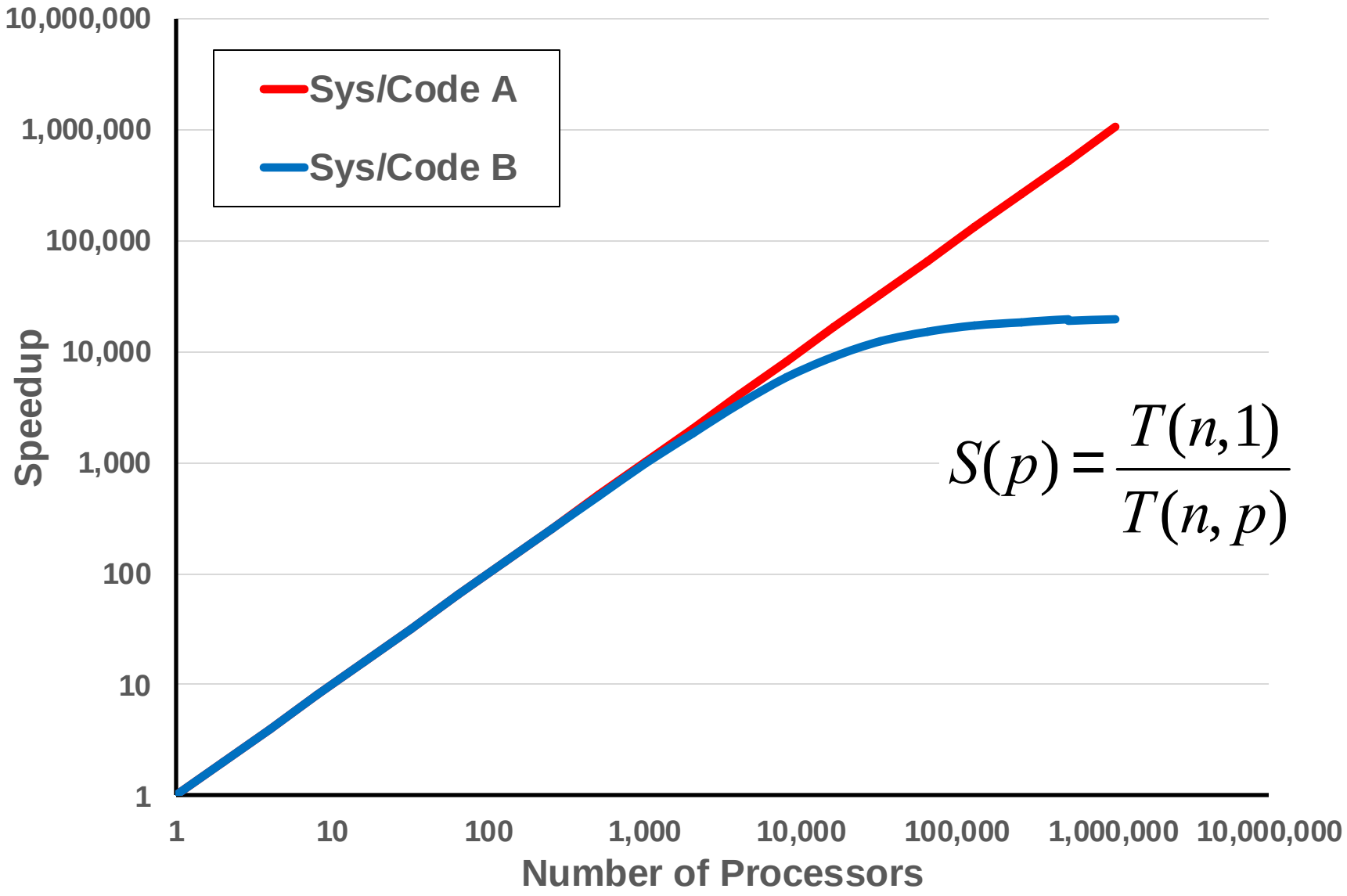
Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

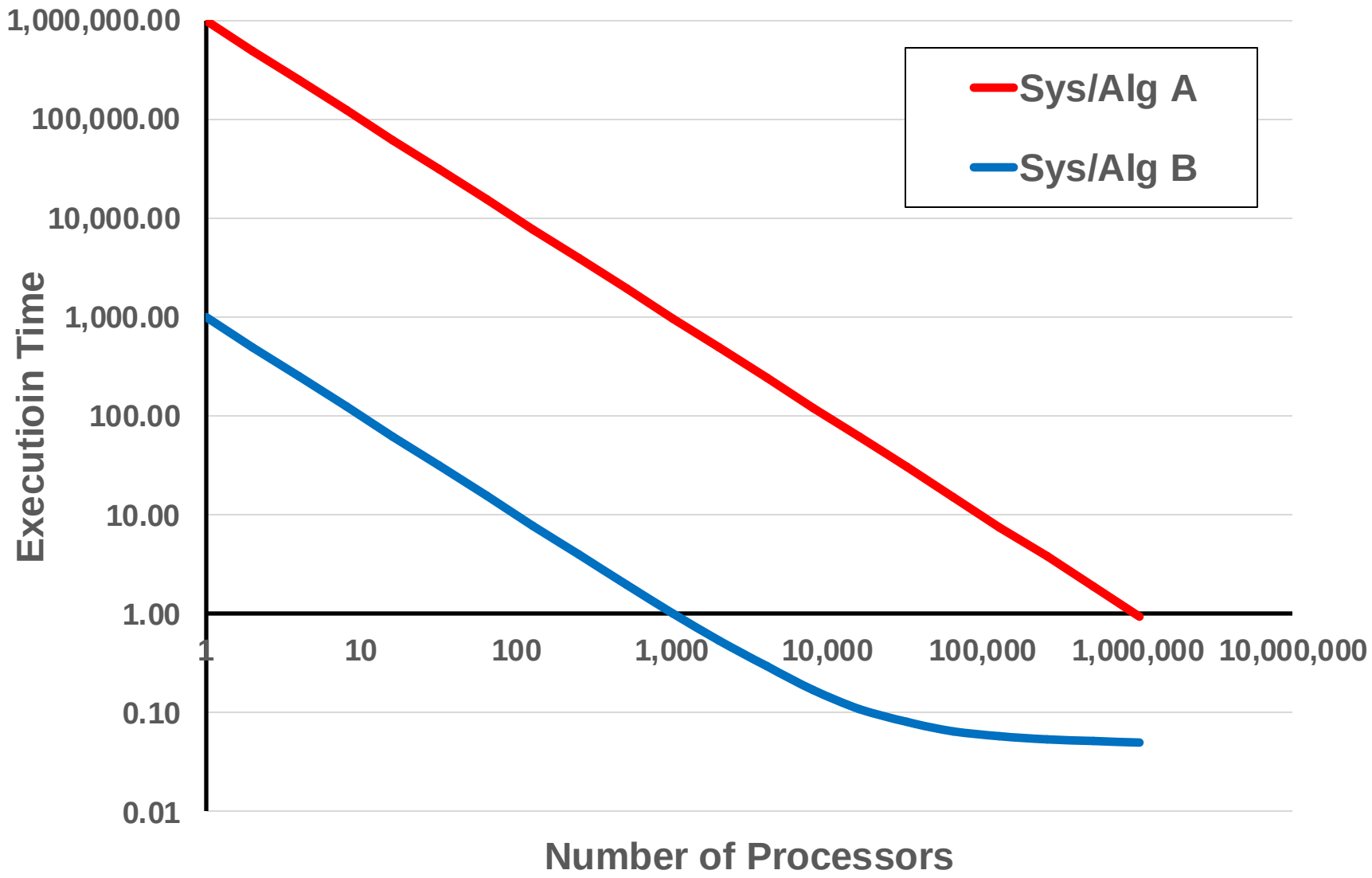
Landscape of current “Graph World”



Scalability



Scalability



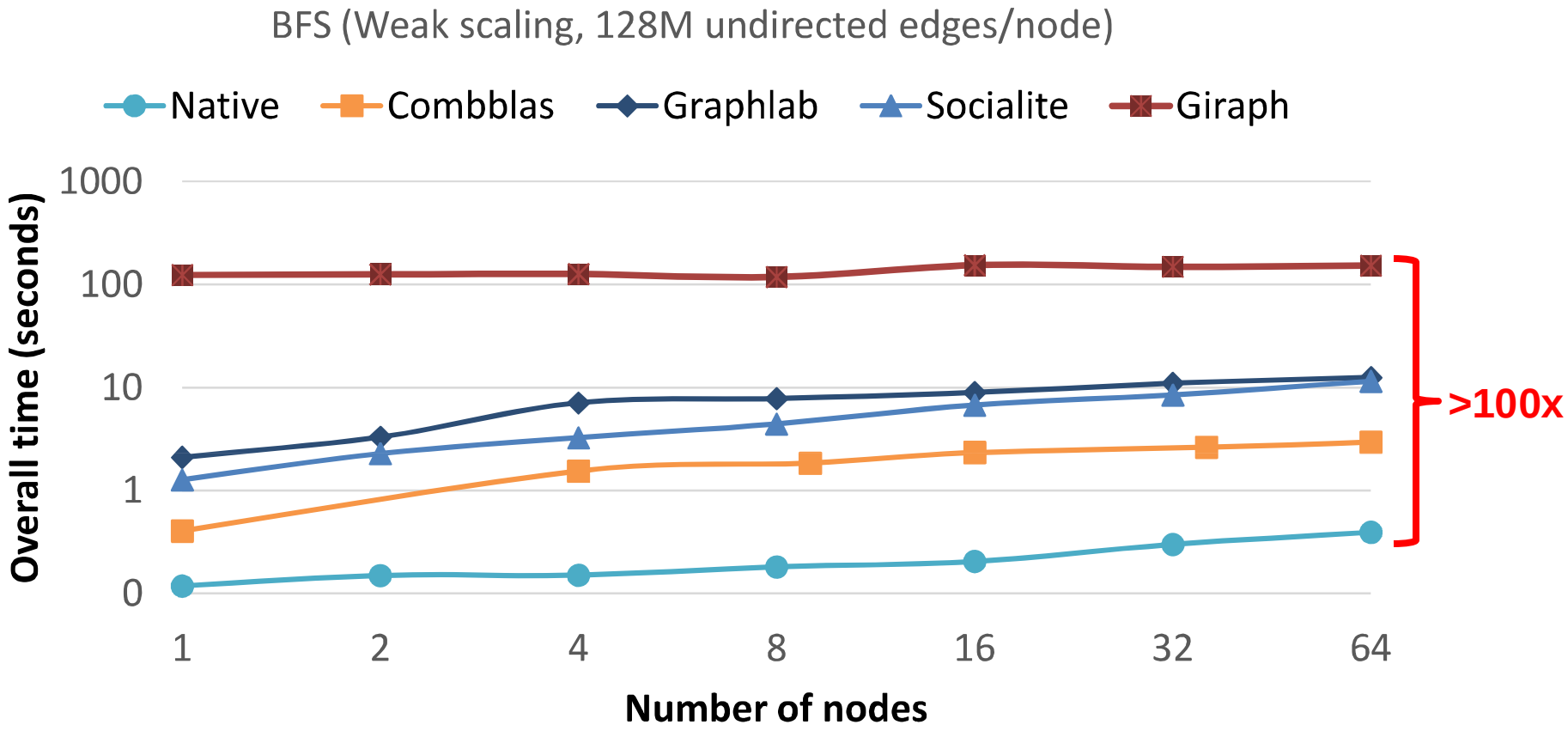
Scalability! But at what COST?

scalable system	cores	twitter	uk-2007-05
Stratosphere [6]	16	950s	-
X-Stream [17]	16	1159s	-
Spark [8]	128	1784s	$\geq$ 8000s
Giraph [8]	128	200s	$\geq$ 8000s
GraphLab [8]	128	242s	714s
GraphX [8]	128	251s	800s
Single thread (SSD)	1	153s	417s

Table 3: Reported elapsed times for label propagation, compared with measured times for single-threaded label propagation from SSD.

F. McSherry, M. Isard, and D. G. Murray, “Scalability! But at what COST?,” HotOS, 2015.

Productivity vs Performance



N. Satish, N. Sundaram, M. M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey, "*Navigating the maze of graph analytics frameworks using massive graph datasets*". SIGMOD 2014.

Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

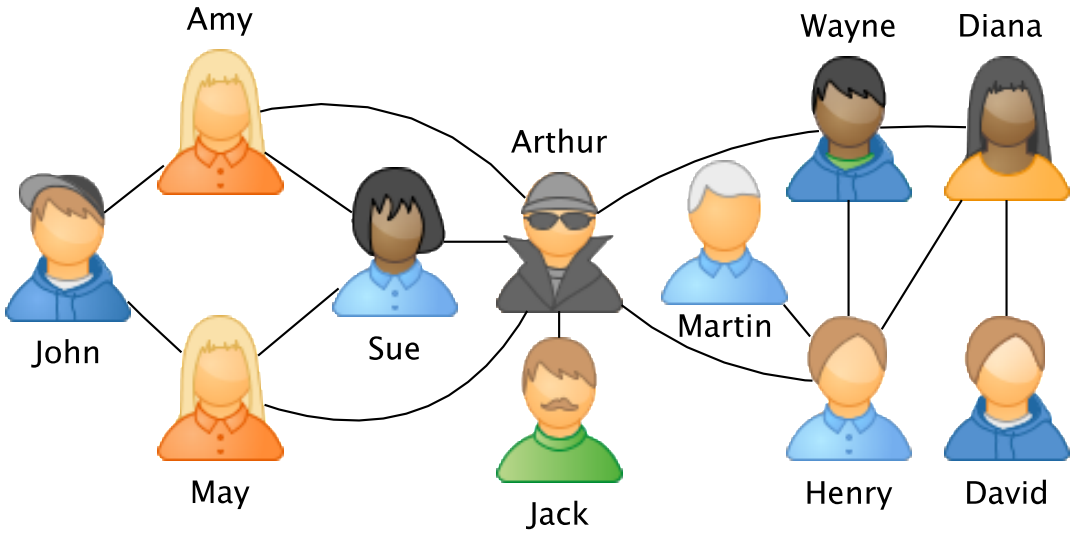
HPC Graph Analytics - Tips/Tricks

- **It is *almost* all about data movement and maximizing data reuse/locality**
  - Smaller problems are easier to solve (and they take less memory space!)
    - Reduce your problem size: Graph manipulations/compression
  - Memory/Data Structures Optimizations
    - Align (/reorder) to memory/compute architecture
    - Make it memory hierarchy (cache) friendly
  - Processor optimizations
    - Take advantage of (instruction) parallelism (e.g., Vectorization, Tensor-Cores etc.)
      - Well, sometimes non-work optimal algorithm (with redundant computation) runs faster!
  - Hybrid (Memory + Processor)
    - Task-based execution: 1D vs 2D tasks
    - Task scheduling: pipelining, multi-buffering for computation and communication overlap
- Eliminate “redundant” computation
  - Can you reduce the solution space?
  - Would approximation be appropriate/good?
  - Can you sparsify?

Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

Graph manipulations for fast centrality computations



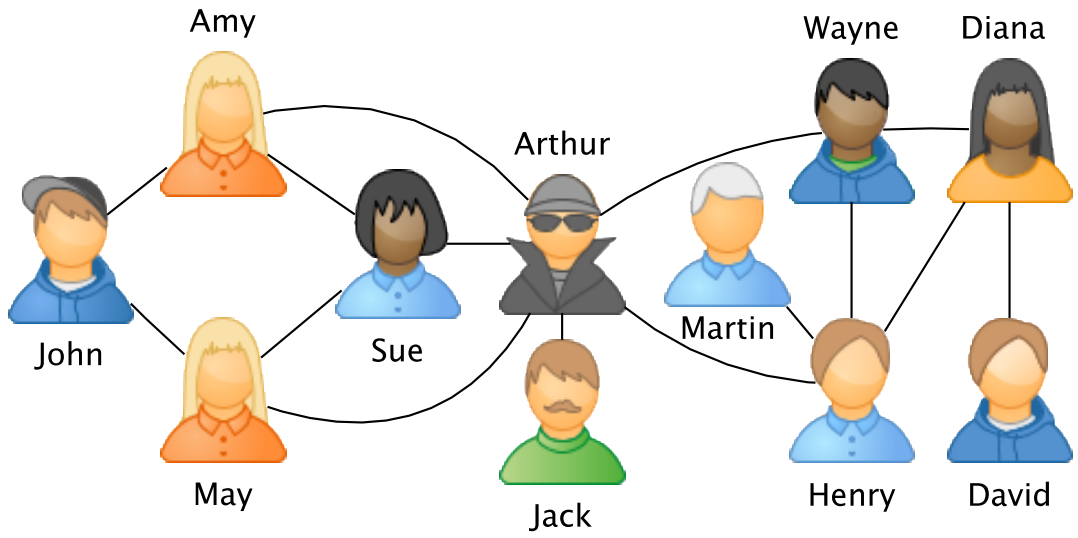
- Let **G** = (**V**, **E**) be an undirected and unweighted graph with a vertex set **V** of **n** vertices and an edge set **E** of **m** edges

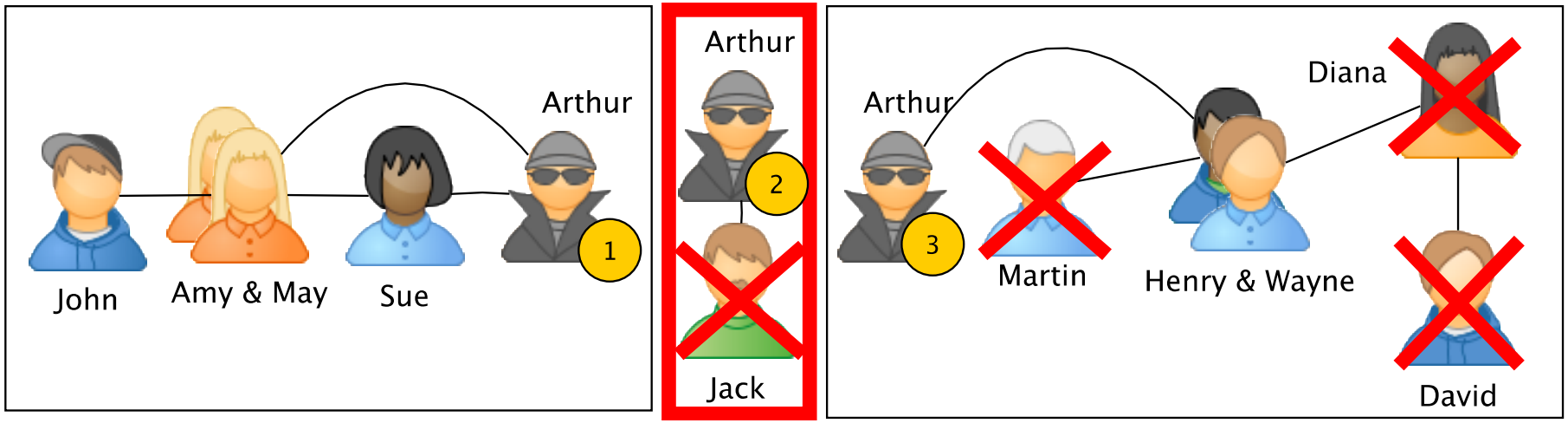
- Closeness: $\texttt{far}[u] = \sum_{\substack{v \in V \\ d_G(u,v) \neq \infty}} d_G(u,v)$ $\qquad \texttt{cc}[u] = \frac{1}{\texttt{far}[u]}$

- Betweenness: $bc(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$

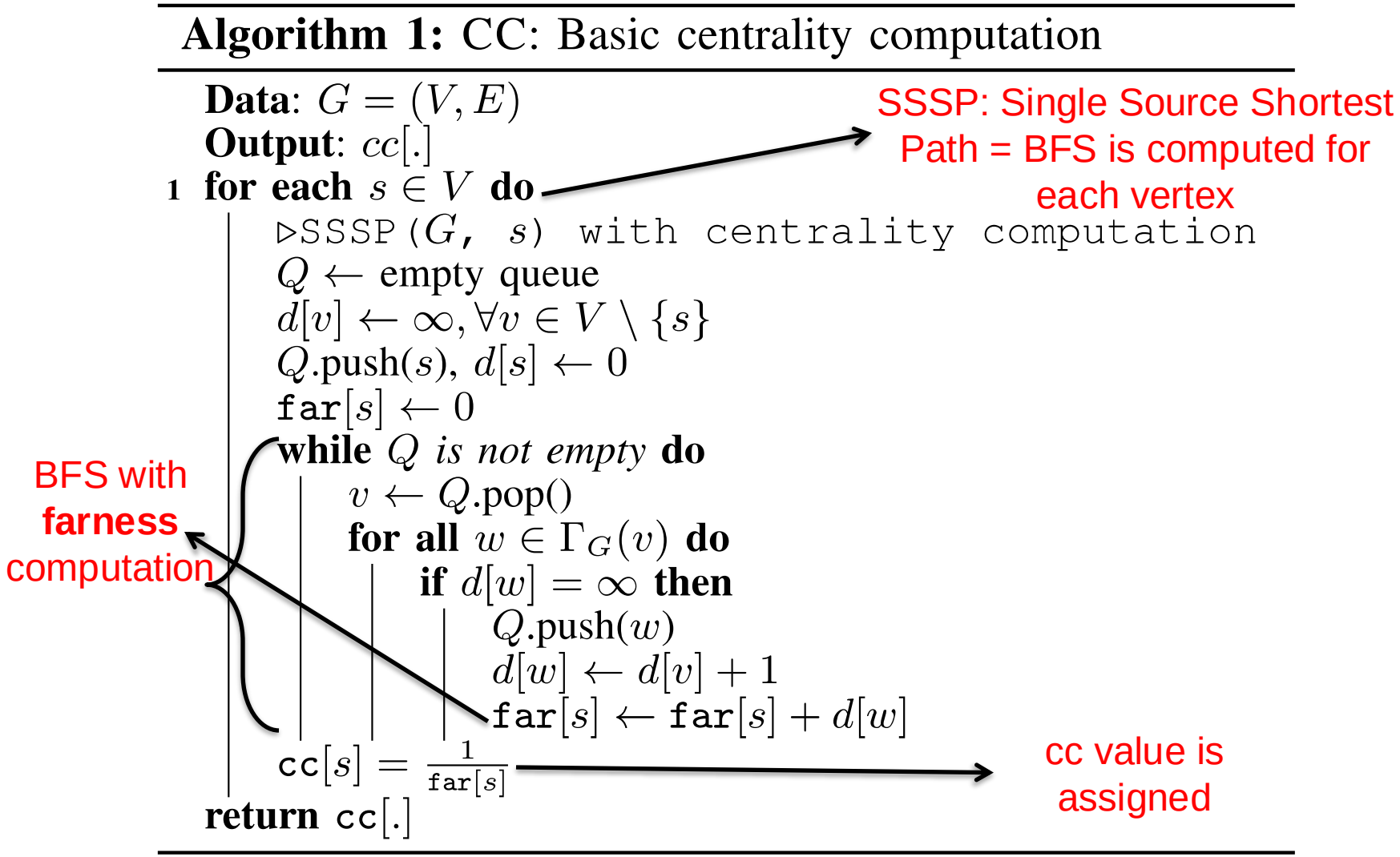
  - O(**mn**) complexity for both metrics

Graph manipulations for fast centrality computations





Closeness Centrality

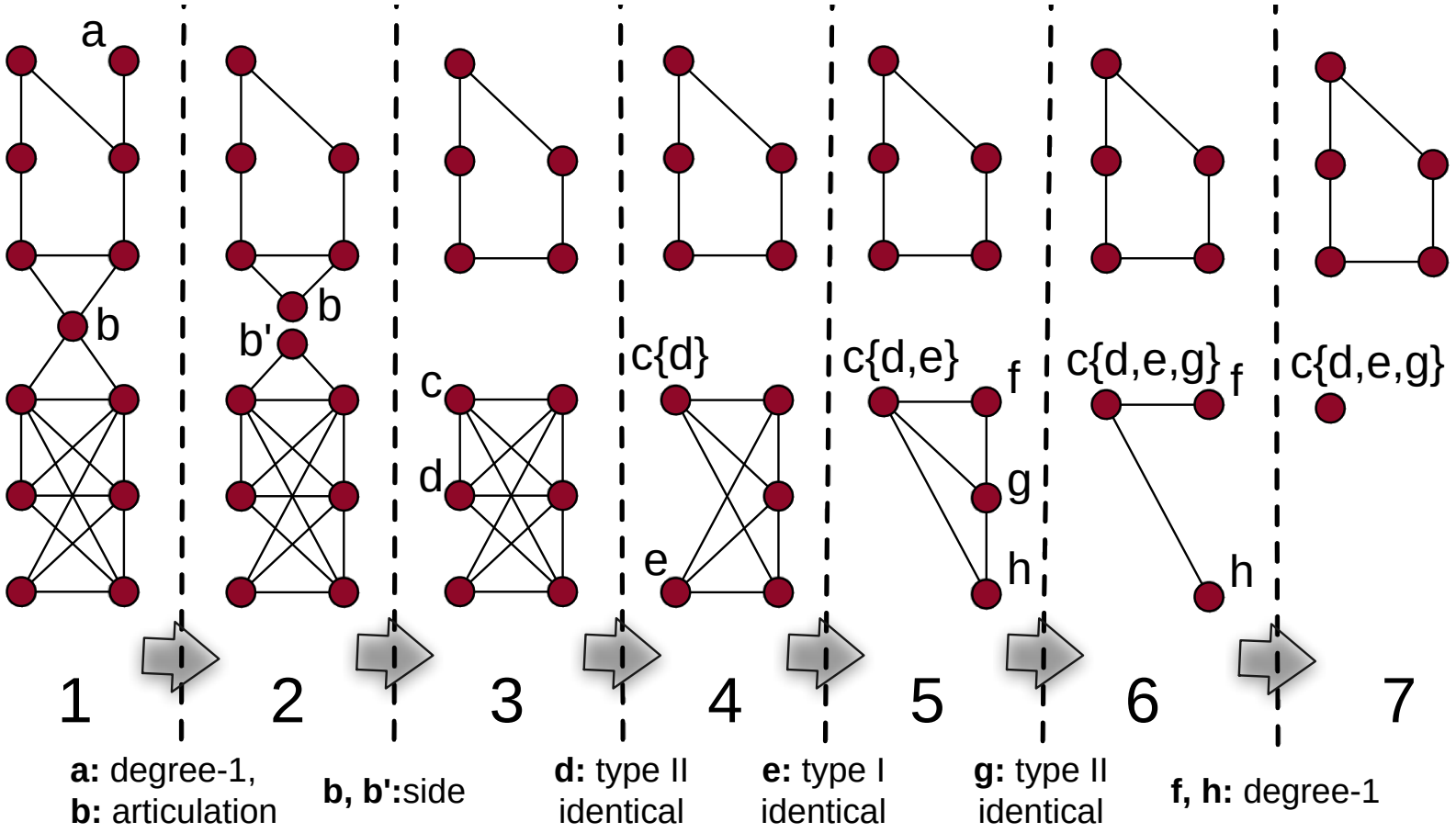


Betweenness Centrality

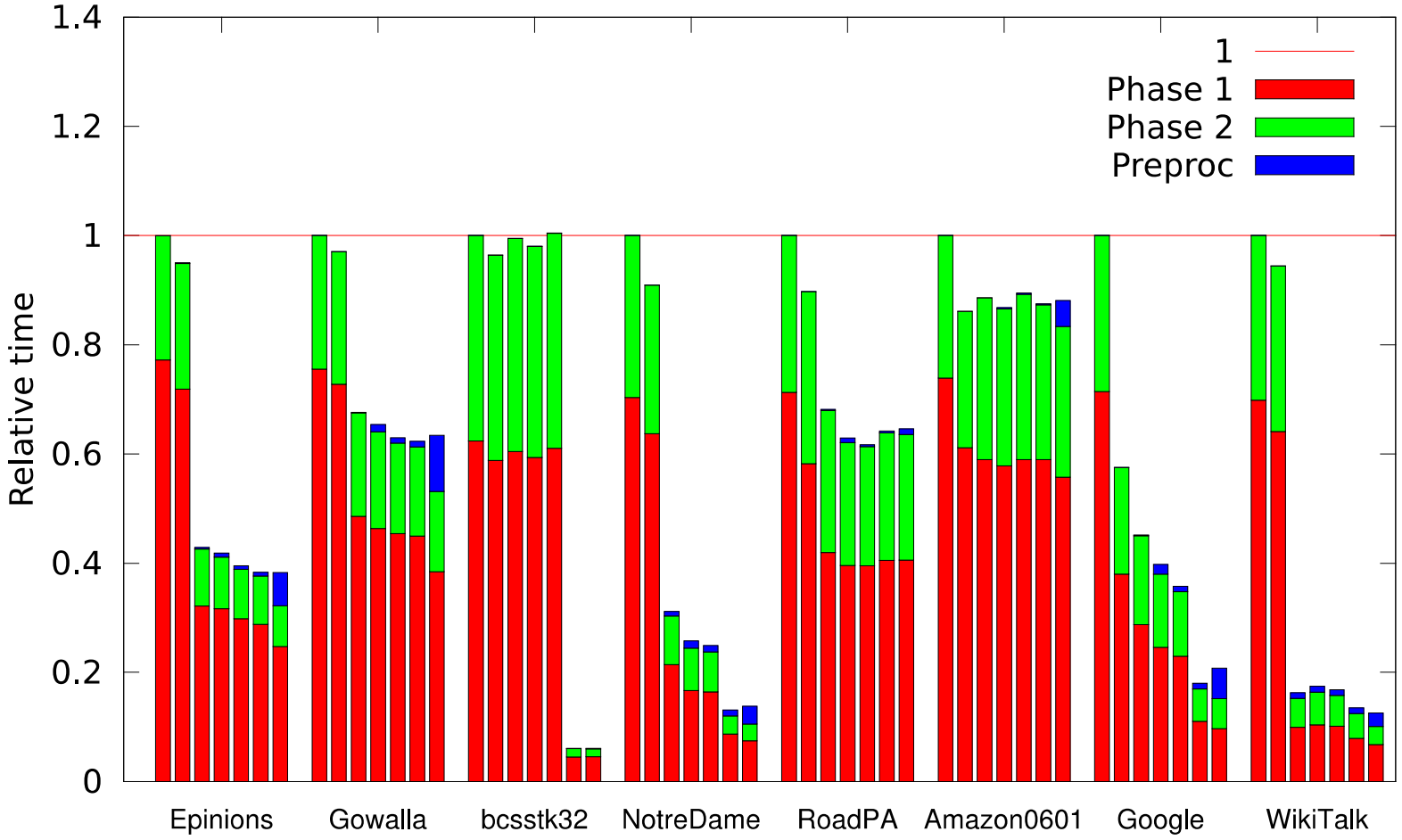
- The current best algorithm (by Brandes)
  - Phase 1: simple BFS with shortest path counting
  - Phase 2: computing partial BC scores with counted paths

```
Data: G = (V, E)
bc[v] ← 0, ∀v ∈ V
for each s ∈ V do
    S ← empty stack, Q ← empty queue
    P[v] ← empty list, σ[v] ← 0, d[v] ← −1, ∀v ∈ V
    Q.push(s), σ[s] ← 1, d[s] ← 0
    // Phase 1: BFS from s
    while Q is not empty do
        v ← Q.pop(), S.push(v)
        for all w ∈ Γ(v) do
            if d[w] < 0 then
                Q.push(w)
                d[w] ← d[v] + 1
            if d[w] = d[v] + 1 then
1               σ[w] ← σ[w] + σ[v]
                P[w].push(v)
    // Phase 2: Back propagation
    δ[v] ← 1/σ[v], ∀v ∈ V
    while S is not empty do
        w ← S.pop()
        for v ∈ P[w] do
2           δ[v] ← δ[v] + δ[w]
        if w ≠ s then
3           bc[w] ← bc[w] + (δ[w] ⋅ σ[w] − 1)
```

Graph Manipulations for Centrality with BADIOS



BC Experiments for BADIOS



Epinions	Gowalla	bcsstk32	NotreDame	RoadPA	Amazon0601	Google	WikiTalk
36m	1h38m	12m	**2h**	1d8h	12h	1d18h	**5d5h**
14m	1h	41s	**17m**	20h	10h	8h	**16h**

BASE, o, do, dao, dbao, dbaio, dbaiso

[SDM'13, TKDD'17]

Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

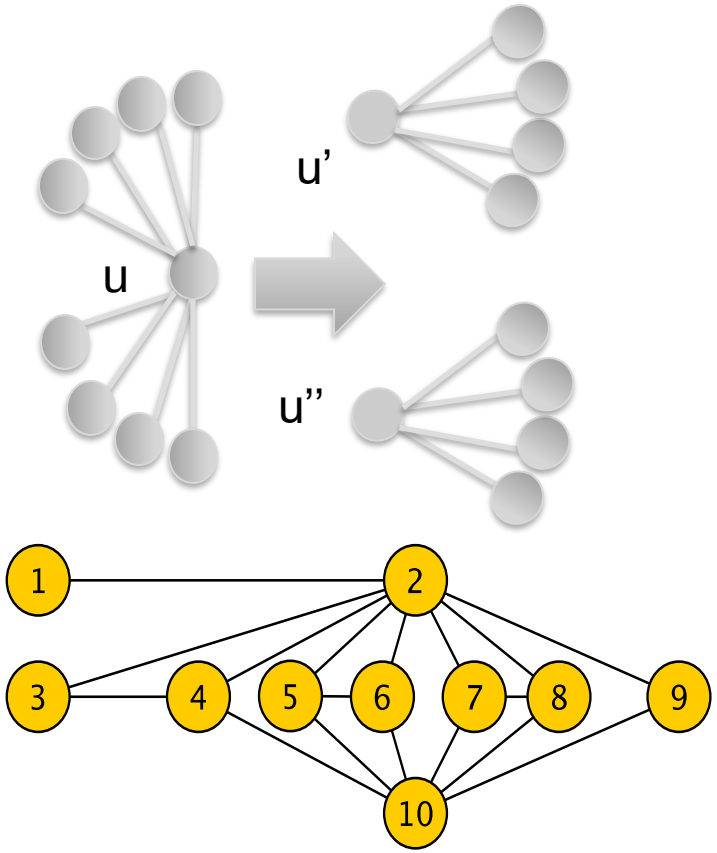
How to implement Centrality Computations in GPU?

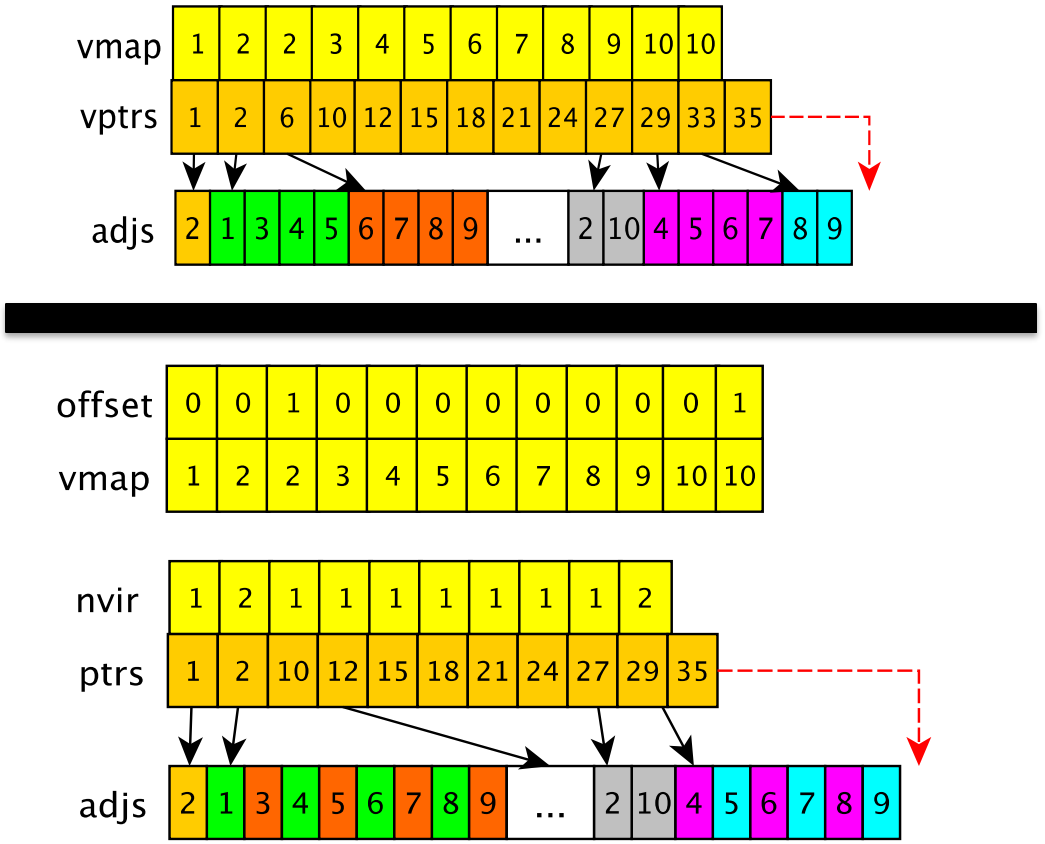
- **Vertex-based parallelism**
  - Assign a thread to each vertex
  - When a thread processes a vertex **u**, if there exists an unvisited vertex **w** in **u**'s neighborhood, set d[**w**] = d[**u**] + 1
- **Edge-based parallelism**
  - Assign a thread to each edge
  - For each edge (**u**,**v**), if **u** is in the current level and **v** is not visited yet set d[**v**] = d[**u**] + 1

- **Both has problems**
  - Vertex-based: load imbalance
  - Edge-based: too many atomic operations

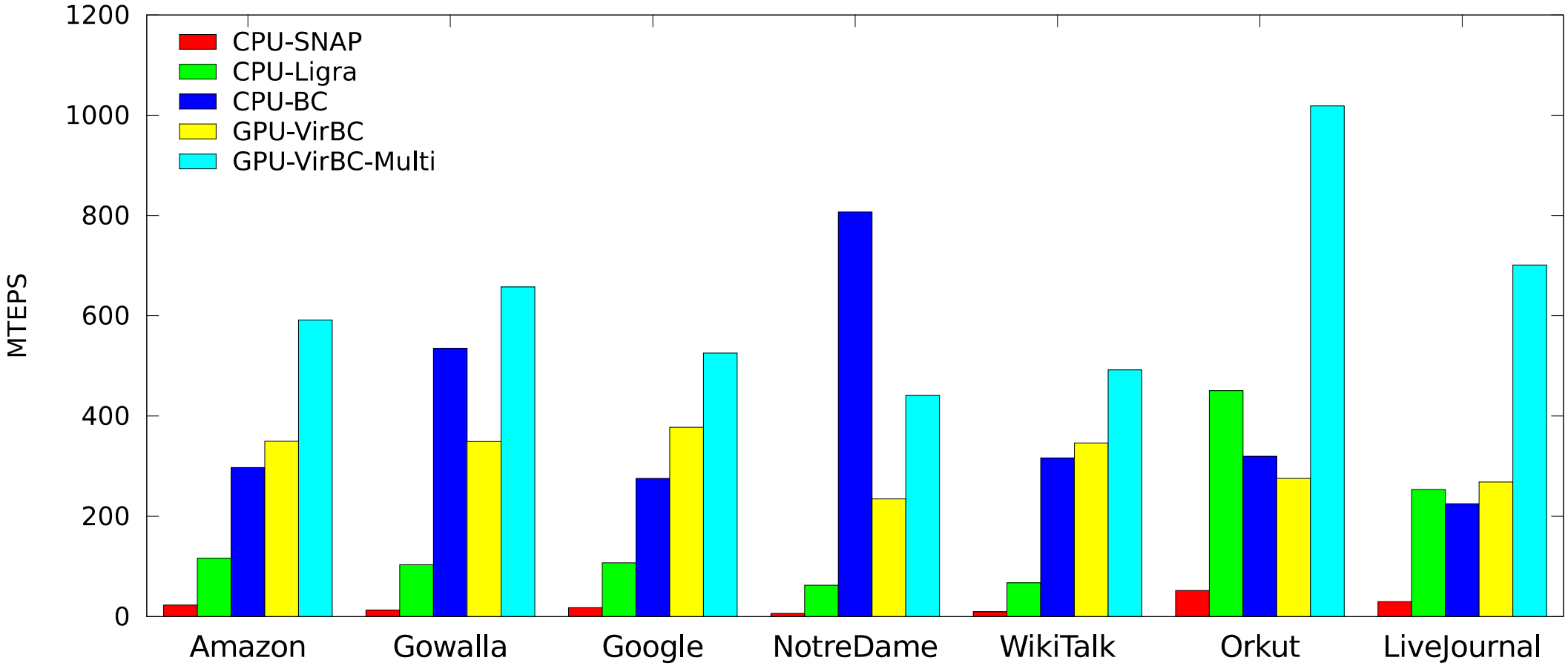
Centrality Computations on GPU: Virtual vertices

- Use multiple virtual vertices for a vertex with high degree
  - Hybrid edge/vertex parallelism.
- Restructure computation to take advantage of coalesced memory access





Results for GPU parallelism



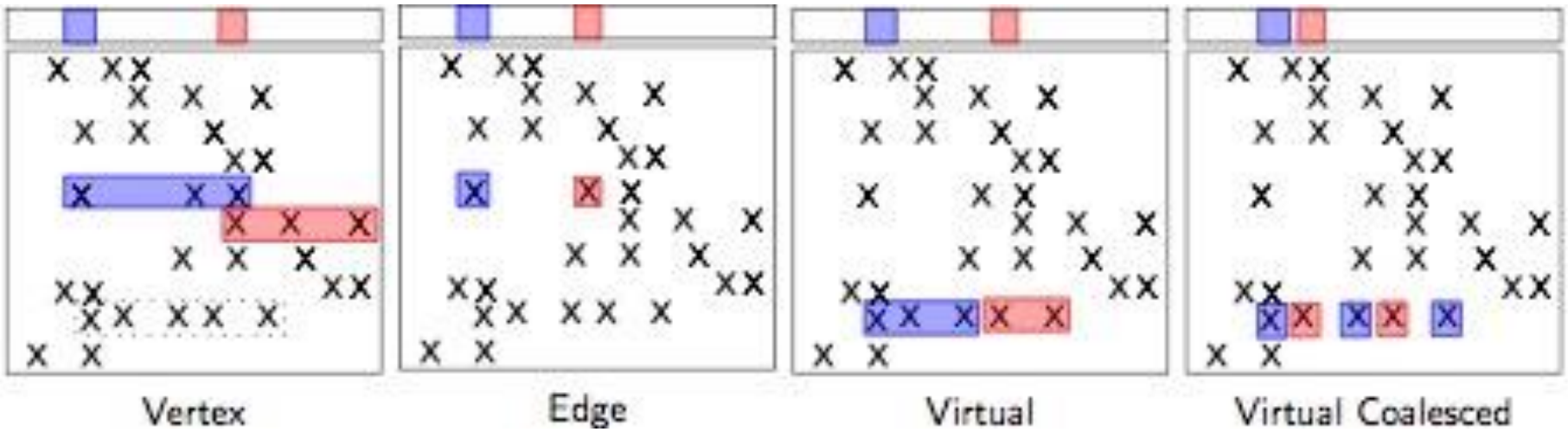
- VirBC-Multi (vectorized BC with Virtual Vertices) is better 6 out of 7
- VirBC-Multi is 4.7x faster than Ligra, 1.6x faster than our CPU code

[GPGPU'13, JPDC'15]

Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

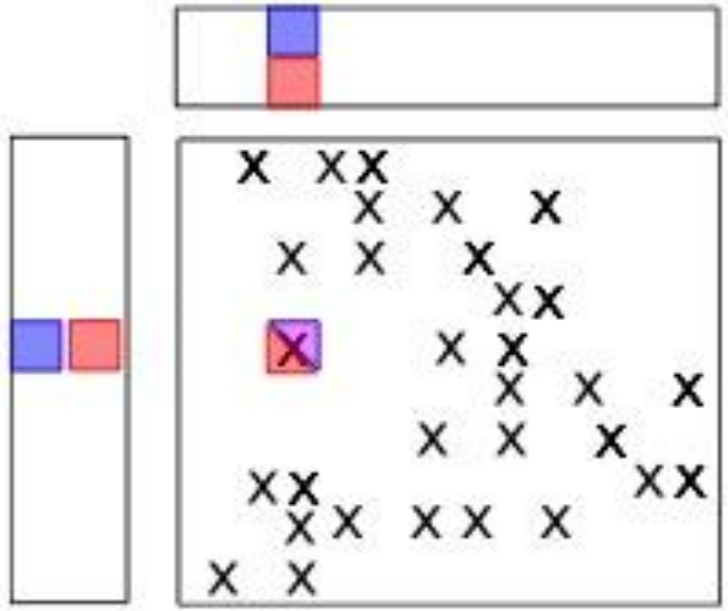
No Vector Coalescing



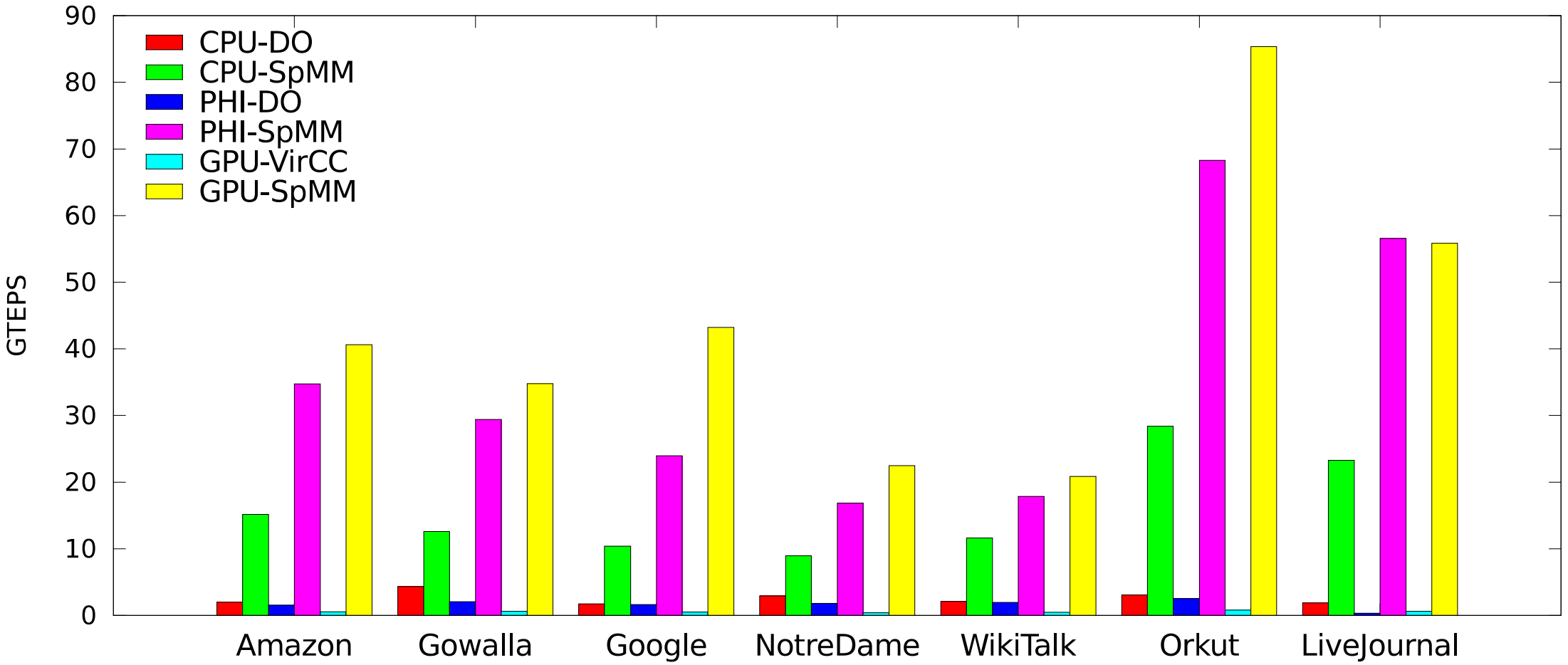
- All the representations give vector coalescing only ``if you are lucky''

Different Formulation for Vectorization

- Vectorization is overlooked in most graph problems
  - Irregular nature of applications
- We make our applications more "**regular**"
  - Process B traversals at once
- Level synchronous execution.
- Model the CC/traversal with **SpMV** operation (like GraphBLAS does)
  - **Sp**arse **M**atrix **V**ector Multiplication: Multiplying a sparse matrix with a vector
- Switch from SpMV to **SpMM**: **Sp**arse **M**atrix **M**atrix Multiplication
  - **Increases the complexity** from O(|E|) to O(d*|E|), but suitable for vectorization!



Hardware+Software Vectorization is the way to go



Vectorized code (*-SpMM) is **5.9x** faster in CPU, **21x** in Xeon Phi,
70x on GPU (compared to previous best)

[MTAAP'14, JPDC'15]

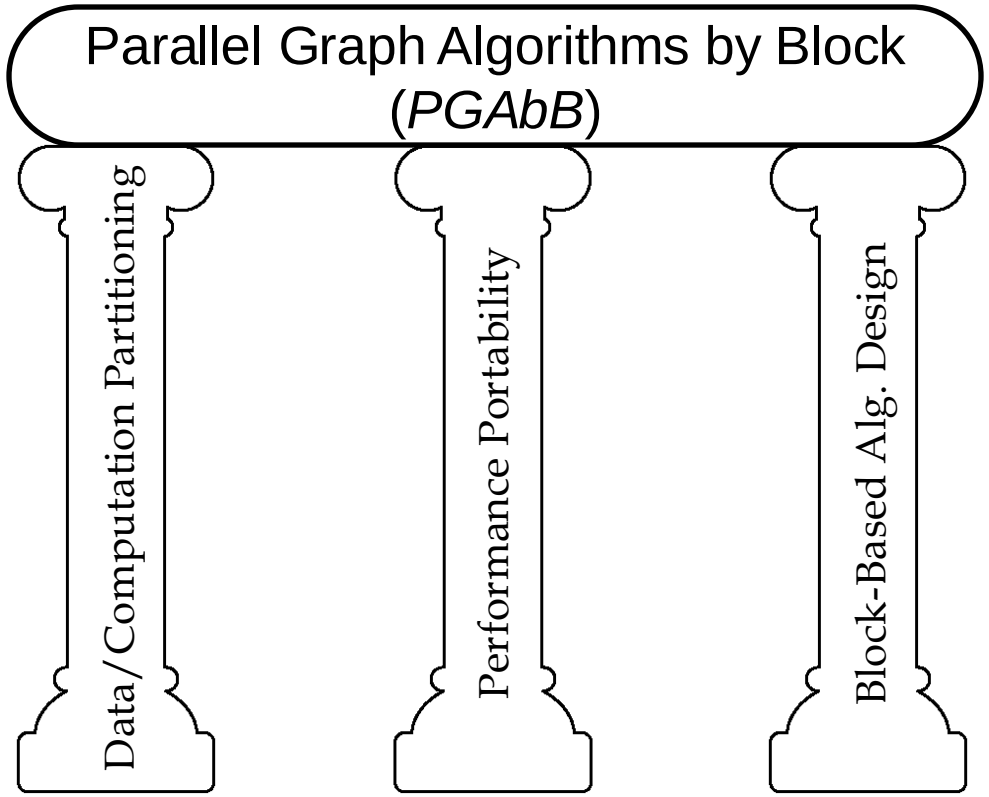
Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

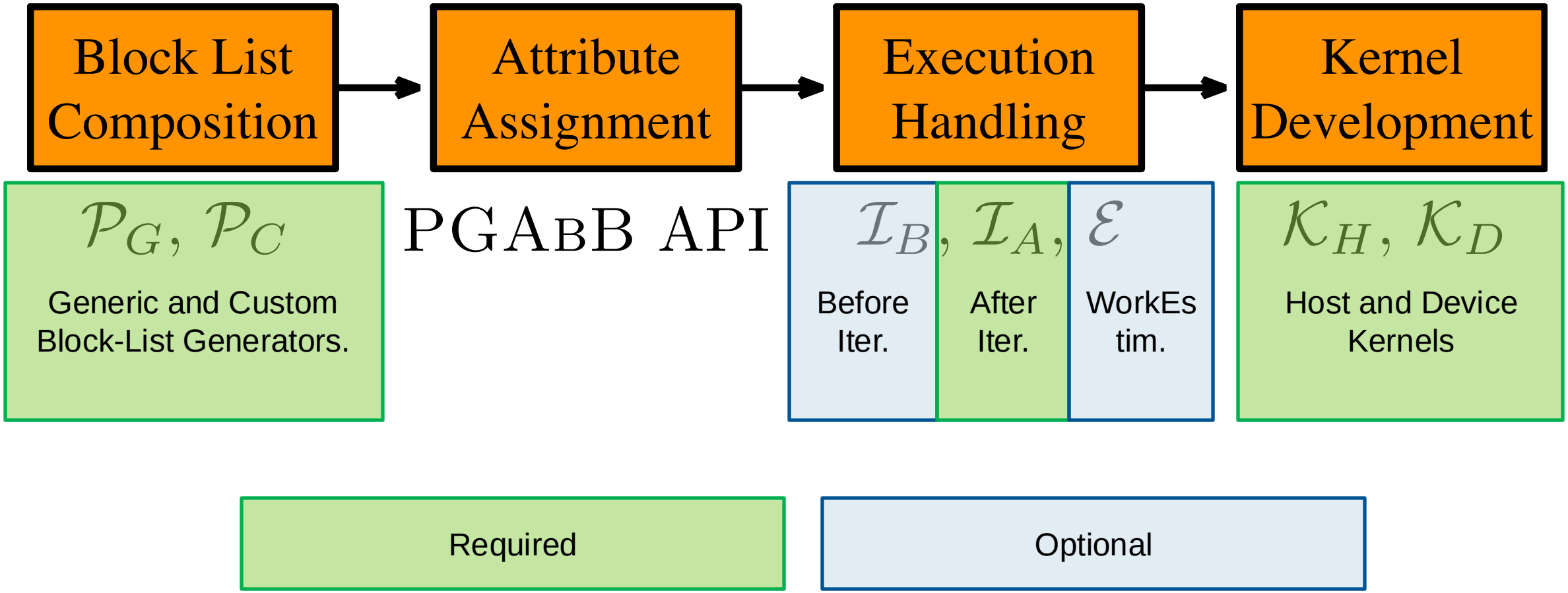
Parallel Graph Algorithms by Blocks (PGAbB)

How can we propose architecture agnostic graph algorithms that run well on **shared-memory and heterogeneous systems** as well as distributed-memory systems?

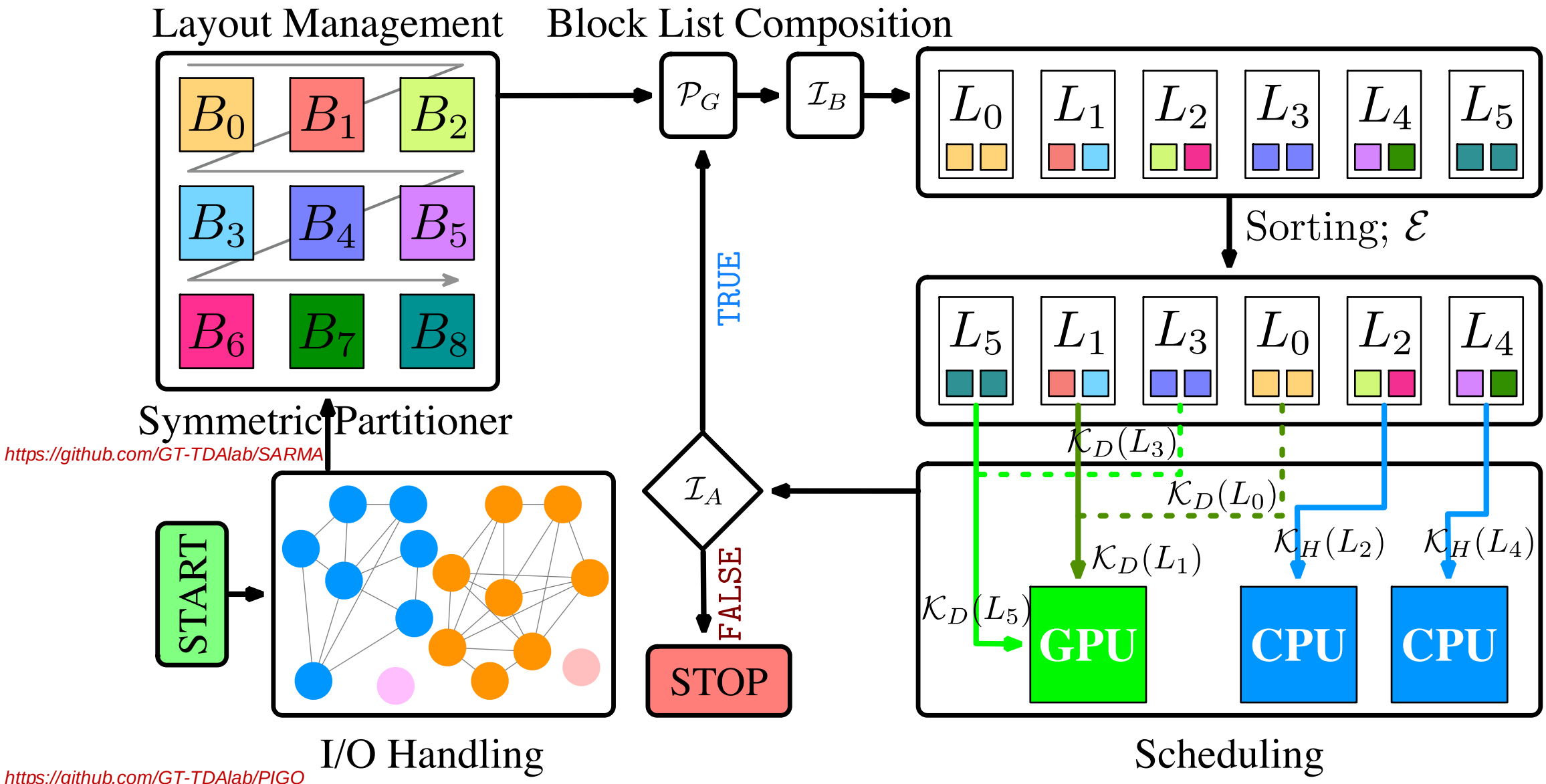
Block-based graph algorithms offer a sweet spot between efficient parallelism and architecture agnostic algorithm design



Algorithm Design Steps



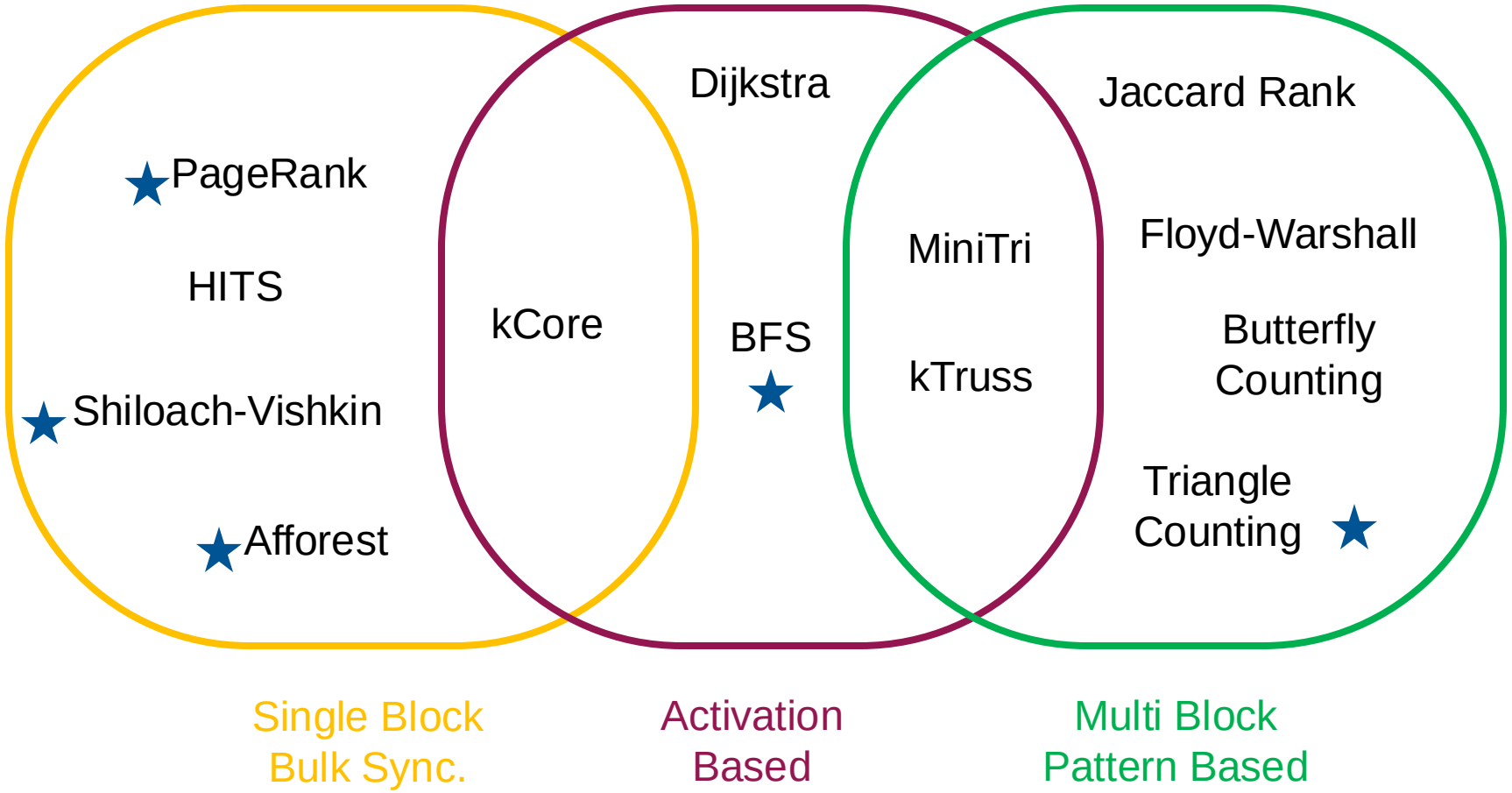
Execution Flow



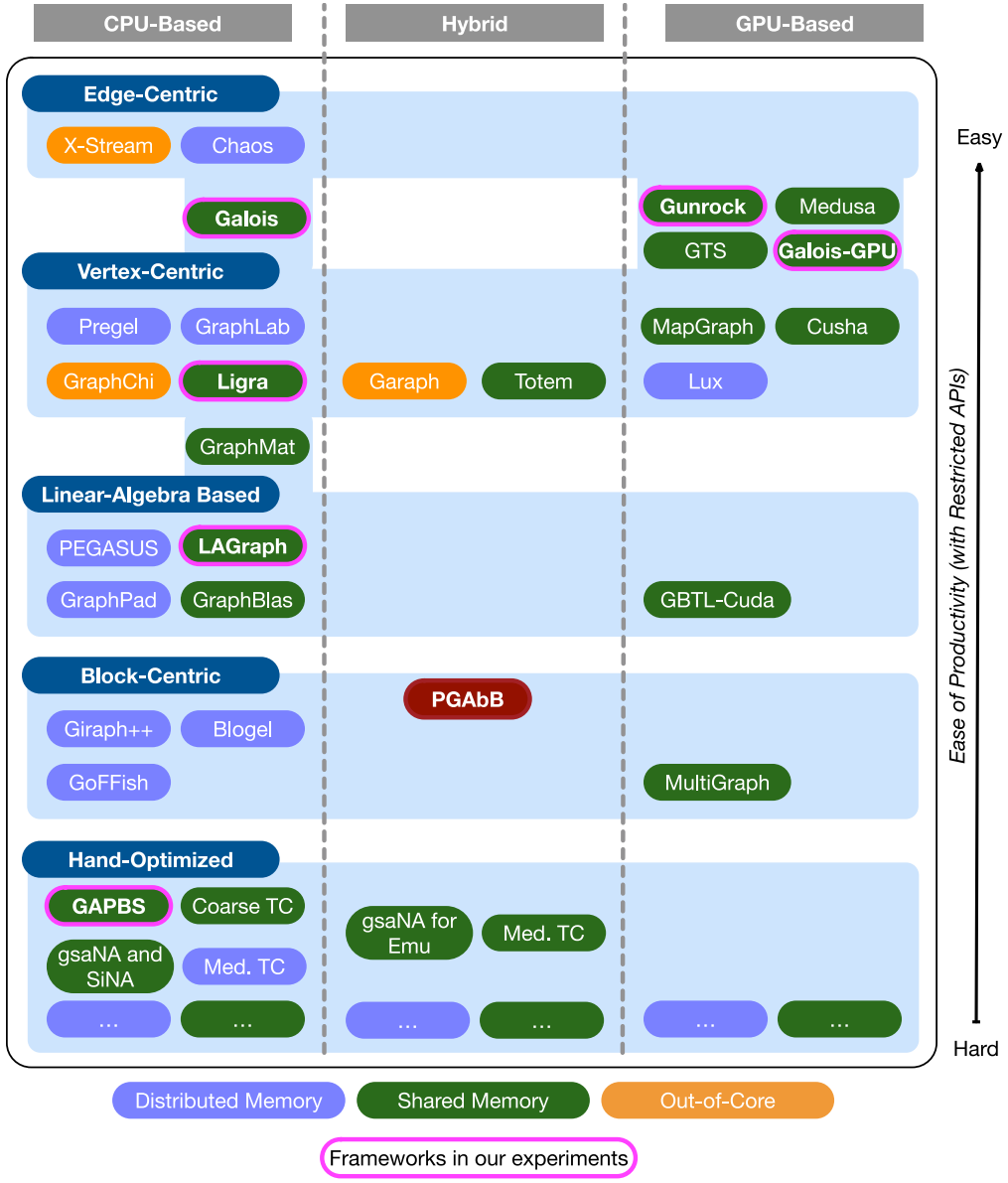
https://github.com/GT-TDAlab/SARMA

https://github.com/GT-TDAlab/PIGO

Categorizing Graph Algorithms



Related Work



Frameworks in Our Experiments

GAPBS: Beamer, et al., 2015. "The GAP benchmark suite.", ArXiV

Galois: Kulkarni, et al. 2007. *"Optimistic parallelism requires abstractions"*, PLDI

Ligra: Shun and Blelloch. 2013. *"Ligra: a lightweight graph processing framework for shared memory"*, PPoPP

LAGraph: Davis. 2019. *"Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra"*, TOMS

Galois-GPU: Martin Burtscher, et al. 2012. *"A quantitative study of irregular programs on GPUs"*, IISWC

Gunrock: Wang, et al. 2016. *"Gunrock: A high-performance graph processing library on the GPU"*, PPoPP

Experiments on Selected Graphs

- Power9 (2 x 16 x 4) CPUs & Volta100 GPU.
  - 320 GB Host Memory. 32 GB Device Memory.
  - CPU-GPU bandwidth: ~60GB/s
- PGAbB: Kokkos at the backend with OpenMP (Host) and Cuda (Device)

Graph	Number of Vertices	Edges	Triangles	CC
Twitter7	41.6 M	1.2 B	34.8 B	0.001
Com-Orkut	3 M	117 M	627 M	0.041
Sk-2005	50.6 M	1.8 B	84.9 B	0.002
Kmer_V1r	214 M	232 M	49	0.000
Europe-OSM	50.9 M	54.1 M	61 K	0.003
Myciel.19	393 K	451 M	0	0
Kron-Scale21	2.1 M	91 M	8.8 B	0.044

		Social		Web	Gene	Road	Synthetic	
		twitter7	Orkut	sk-2005	kmer_V1r	eu_osm	myciel19	kron21
Galois	PR	0.83	1.01	1.01	0.89	1.03	6.96	0.78
	SV/LP	8.40	1.71	1.68	2.29	1.81	1.25	1.12
	CC	0.84	1.56	0.98	0.64	0.64	2.94	0.81
	BFS	0.26	0.59	0.46	0.34	2.14	0.39	0.18
	TC	0.69	1.06	0.63	0.90	1.21	0.44	0.40
Ligra	PR	0.39	0.60	0.99	0.43	0.53	2.59	0.72
	SV/LP	1.24	0.70	1.05	0.18	0.02	0.58	0.66
	CC	0.02	0.04	0.00	0.02	0.01	0.03	0.02
	BFS	0.61	0.67	0.93	0.68	0.16	1.37	0.82
	TC	0.31	0.35	0.12	0.30	0.17	0.43	0.69
LAGraph	PR	0.75	0.98	0.60	0.75	0.65	3.21	0.71
	SV/LP	14.24	1.64	0.89	0.30	0.13	7.70	0.92
	CC	0.17	0.21	0.12	0.14	0.05	0.27	0.09
	BFS	0.79	0.33	0.77	0.27	0.33	0.75	0.30
	TC	0.38	0.87	0.66	0.29	0.16	0.52	0.37
Galois-GPU	PR	0.00	2.72	0.00	1.01	1.49	12.12	1.62
	SV/LP	0.00	3.67	0.00	2.43	2.71	2.65	1.57
	CC	0.00	0.46	0.00	1.16	0.99	0.09	0.15
	BFS	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TC	1.03	0.85	0.90	0.00	0.00	0.38	0.65
Gunrock	PR	0.00	1.28	0.00	1.44	1.34	5.42	0.97
	SV/LP	0.00	1.88	0.00	3.18	1.22	3.90	0.97
	CC	0.00	0.24	0.00	1.51	0.44	0.14	0.09
	BFS	4.61	1.48	0.00	3.59	0.80	3.45	5.73
	TC	0.00	0.74	0.00	0.04	0.02	0.29	0.23
PGAbB-GPU	PR	4.20	4.72	0.74	0.53	0.64	13.60	2.30
	SV/LP	19.19	9.96	3.16	6.45	3.63	9.21	3.85
	CC	1.68	1.08	5.52	3.56	1.37	0.64	0.31
	BFS	0.18	0.85	0.97	0.28	0.32	1.06	0.27
	TC	3.09	3.39	2.34	0.52	0.32	2.87	2.33
PGAbB	PR	4.64	4.67	0.80	0.53	0.64	10.76	1.79
	SV/LP	18.02	5.95	1.90	5.73	2.95	7.70	1.98
	CC	1.25	1.53	2.14	1.91	0.96	2.40	0.87
	BFS	0.16	0.89	0.77	0.90	0.33	1.00	0.29
	TC	3.02	3.01	1.69	1.11	3.91	5.39	3.48

Outline

- Motivation
- Current Landscape of “Graph World”
- Few Examples of HPC Graph Analytics
  - HPC Graph Analytics - Tips/Tricks
  - Faster centrality computations
    - Graph manipulations for fast centrality [SDM’13, TKDD’17]
    - Faster centrality computations on GPU [GPGPU’13]
    - Vectorized centrality computations [MTAAP’14,JPDC’15]
- A Middle Ground: Task-based Execution on Heterogeneous Environments
    - Parallel Graph Algorithms by Blocks (PGAbB) [HPEC’19,TPDS'22,underreview]
- What about Graph Databases
  - Interoperability Challenges
  - Design Challenges
- Conclusions & Future Directions

Graph Databases

Two camps

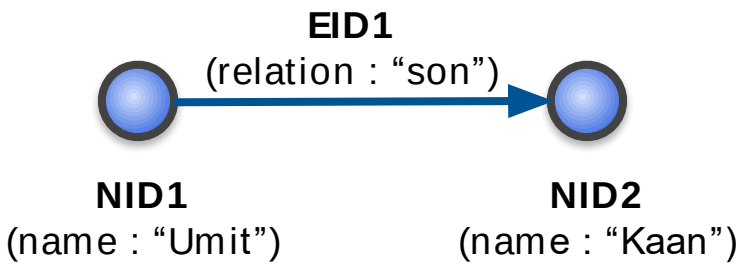
LPG: Labeled Property Graphs

Vertices

Nodes: Label/ID + *Properties* (set of key-value pairs)

Edges

Relationships: Label/ID + Type + Properties



RDF: Resources Description Framework

Vertices

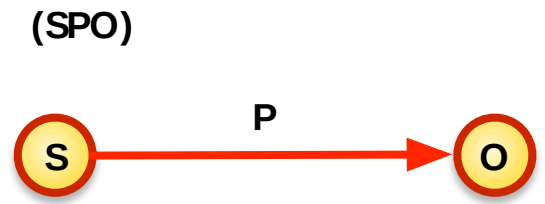
Resources: URIs

Attribute Values: Literals

Edges

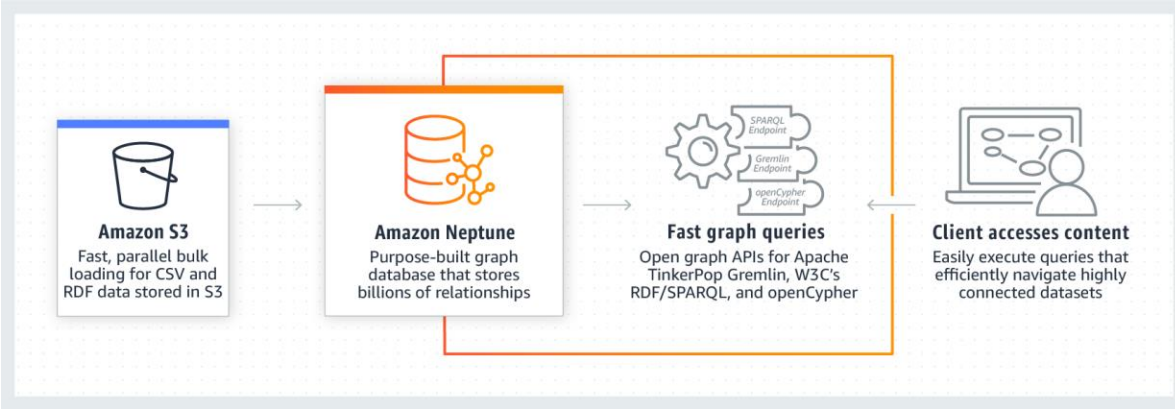
Relationships: URIs

RDF Triple: **S**ubject-**P**redicate-**O**bject



There is no internal structure for nodes and edges

Graph interoperability



- Amazon Neptune
  - managed, cloud-based graph database service
  - supports RDF (SPARQL) and LPG (Gremlin & openCypher)
- User has to choose either RDF or LPG
  - this choice also determines which query languages are available
  - the choice is not always easy, and is hard to reverse later
- RDF vs. LPG
  - RDF offers a formal model, LPG not so much
  - RDF is “sometimes seen as academic”, and developers tend to prefer LPG
  - different strengths and weaknesses

Graph interoperability

- What if we did not have to choose between RDF and LPG?
- What if we could use Gremlin over RDF, or SPARQL over LPG?
- Interoperability: single graph (meta)model, free use of any query language
  - we are not interested in “qualified” interoperability where one metamodel is *implemented* using the other
- RDF-star is a step towards having LPG features in RDF

- 1G model (“one graph to rule them all”)
  - "Graph? Yes! Which one? Help!", O. Lassila, M. Schmidt, B. Bebee, D. Bechberger, W. Broekema, A. Khandelwal, K. Lawrence, R. Sharda, B. Thompson, arXiv:2110.13348v1, 2021.

Interoperability challenges

- Edge properties, multiple edge instances, reification
- Triples vs. graph abstraction
- Datatype alignment
- Graph partitioning
- Graph merging, external identifiers
- Lack of formal foundation
- Update query semantics

The Neptune team seek support from the broader community to look into these issues

Storage Challenges

- Interoperability: serve both RDF and LPG
  - 1G Graph Storage
- Graph is not static (well, obviously!)
  - Many HPC Graph Analytics kernels assumes graph is not changing.
  - Even dynamic ones conveniently *ignores* deletion.
- Scalability: Scaling Up (vertical/single-node) and Scaling Out (horizontal/multi-node)
  - Read scaling is "easy"
  - Write scaling with transaction support is challenging:
    - Distributed in-memory graph storage with logging is still challenging to implement.
  - How to (dynamically) distribute data
    - Node partition vs Edge partition vs Blocked partition
- What does it mean to provide Graph Analytics under transactional system?

Computational Infrastructure Challenges

- Interoperability: serve multiple query languages and graph analytics
- Can we implement once, and run everywhere: from multi-core to multi-host with potentially accelerators?
- Yes!
  - Multi-Level Intermediate Representation (MLIR) for Graphs
  - Event-based runtime system
  - “Coarse-grained” Labeled-Dataflow Execution
- OLAP vs OLTP tradeoffs and Scheduling
  - multiple, mixed (OLAP and OLTP), concurrent queries.

Other Challenges

- Where does the Graph come from?
- Authoring
- Toolkits
- Visualization
- TCO: Total Cost of Ownership
  - Price/performance and price/scale options.

Conclusions & Future Directions

- Graphs ubiquitous and market is growing extremely fast
  - "*By 2025, graph technologies will be used in 80% of the data and analytics innovations, up from 10% in 2021, facilitating rapid decision making across the enterprise*" Gartner "Market Guide: Graph Database Management Solutions", M. Adrian, A. Jaffri, D. Feinberg, 24 May 2021.
- HTAP (i.e., Hybrid OLTP and OLAP) solutions are needed!
  - Enterprise Graph Systems gives the *illusion* of read scaling, while failing in absolute performance, and write/update scaling (they just leave that to file system)
  - HPC Graph Analytics codes/libraries, are one-off, focused on narrow set of kernels and fail to provide end-to-end solutions
  - Existing "Real" Graph Databases, provides OLTP but fails to deliver OLAP
- Interoperability is a big challenge!
  - SPARQL, Gremlin and OpenCypher queries for both OLTP and OLAP workloads
- Graph as a Service
- **It is exciting times for Graphs!**

Acknowledgements – TDAlab Members and Collaborators

Centrality

Erdem Sarıyüce

Erik Saule

Kamer Kaya

PGAbB

Abdurrahman Yaşar

Sivasankaran Rajamanickam

Jonathan Berry

Amazon Neptune Team

(Other) TDAlab Members

Ümit V. Çatalyürek

Kasimir Gabert

Kaan Sancak

Yusuf Özkaya

Xiaojing An

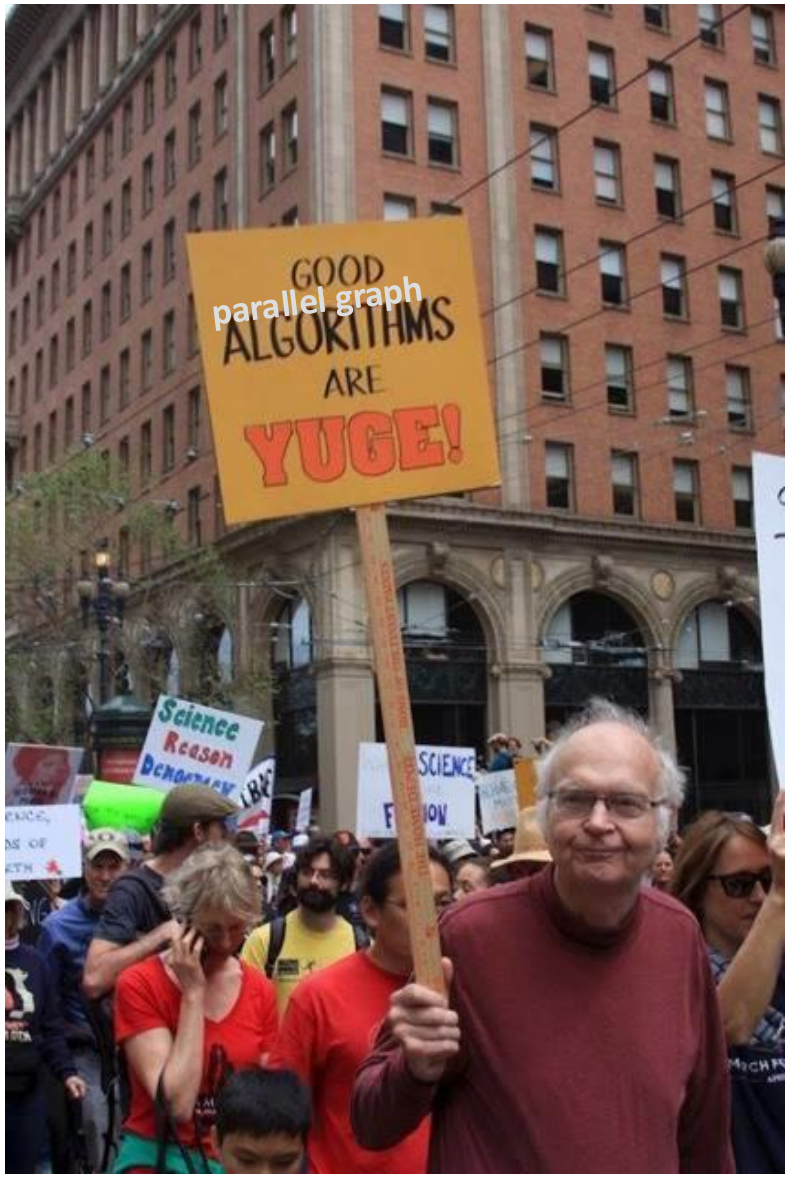
James Fox

Fatih Balin

Benjamin Cobb

Thanks

- For more information
  - email: umit@gatech.edu
  - twitter: @catalyurek
  - web: tda.gatech.edu
- Acknowledgement of (Previous) Support



References

[SDM'13] A.E. Sarıyüce, E. Saule, K. Kaya, and Ü.V. Çatalyürek, "Shattering and Compressing Networks for Betweenness Centrality", 13th SIAM International Conference on Data Mining (SDM), May 2013.

[TKDD'17] A.E. Sarıyüce, K. Kaya, E. Saule, and Ü.V. Çatalyürek, "Graph Manipulations for Fast Centrality Computation", ACM Transactions on Knowledge Discovery from Data, Vol. 11, No. 3, Apr 2017.

[GPGPU'13] A.E. Sarıyüce, K. Kaya, E. Saule, and Ü.V. Çatalyürek, "Betweenness Centrality on GPUs and Heterogeneous Architectures", Workshop on General Purpose Processing Using GPUs (GPGPU), in conjunction with ASPLOS, Mar 2013.

[MTAAP'14] A.E. Sarıyüce, E. Saule, K. Kaya, and Ü.V. Çatalyürek, "Hardware/Software Vectorization for Closeness Centrality on Multi-/Many-Core Architectures", Proceedings of 28th IPDPSW, W. on Multithreaded Architectures and Applications (MTAAP), May 2014.

[JPDC'15] A.E. Sarıyüce, E. Saule, K. Kaya, and Ü.V. Çatalyürek, "Regularizing Graph Centrality Computations", Journal of Parallel and Distributed Computing, Vol. 76, pp. 106-119, Feb 2015.

[HPEC'18] A. Yaşar, S. Rajamanickam, M.M. Wolf, J. W. Berry, and Ü.V. Çatalyürek, "Fast Triangle Counting Using Cilk", High Performance Extreme Computing Conference (HPEC), Sep 2018.

[HPEC'19] A. Yaşar, S. Rajamanickam, J. W. Berry, M.M. Wolf, J. Young, and Ü.V. Çatalyürek, "Linear Algebra-Based Triangle Counting via Fine-Grained Tasking on Heterogeneous Environments", High Performance Extreme Computing Conference (HPEC), Sep 2019.

[TPDS'22] A. Yaşar, S. Rajamanickam, J. W. Berry, and Ü.V. Çatalyürek, "A Block-Based Triangle Counting Algorithm on Heterogeneous Environments". IEEE Transactions on Parallel and Distributed Systems, 2022.

See http://tda.gatech.edu/publications for links/slides/PDFs.

Amazon Neptune – We are hiring!

Amazon Neptune is hiring! If you are excited by the possibility of building the internals of one of the fastest growing graph databases in the world, look no further! We're looking for some fearless SDEs to work on cutting edge technologies and core foundations of HPC, distributed transaction management, graph databases, algorithms and many more. (Locations: US, CAN, GER, IND)

Visit www.amazon.jobs and search “Neptune” for more info on multiple positions/openings.

Contact: uvc@amazon.com