HPC IN THE MANY-CORE ERA: GRAPH PROCESSING CHALLENGES

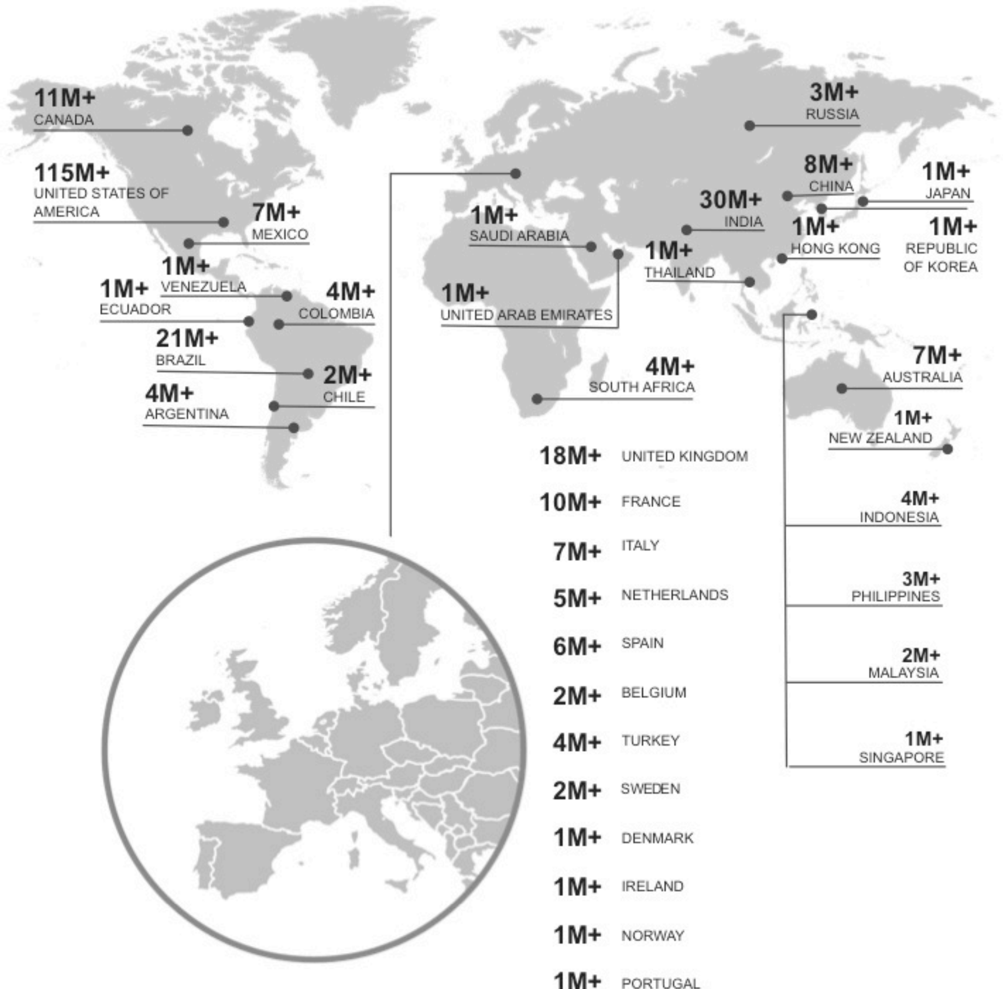
Ana Lucia Varbanescu,
University of Amsterdam, The Netherlands
a.l.varbanescu@uva.nl
Contributions from Merijn Verstraaten, Souley Madougou (UvA),
Yong Guo, Stijn Heldens, the Graphalytics team (TUDelft).

Graph analytics at work

In April 2015 …

364,000,000+

REGISTERED MEMBERS



18M+	UNITED KINGDOM
10M+	FRANCE
7M+	ITALY
5M+	NETHERLANDS
6M+	SPAIN
2M+	BELGIUM
4M+	TURKEY
2M+	SWEDEN
1M+	DENMARK
1M+	IRELAND
1M+	NORWAY
1M+	PORTUGAL

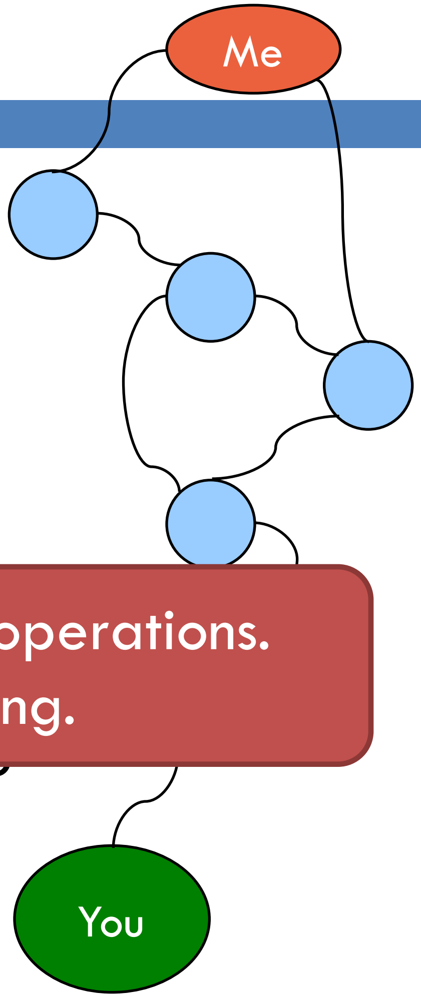
Classical analytics

- □ Statistics
  - “How many connections do I have?”
- □ Traversing
  - “How can I reach Prof. X?”

No textbook algorithms exist for some of these operations.
If they exist, they probably need changing.

around Berkeley.“

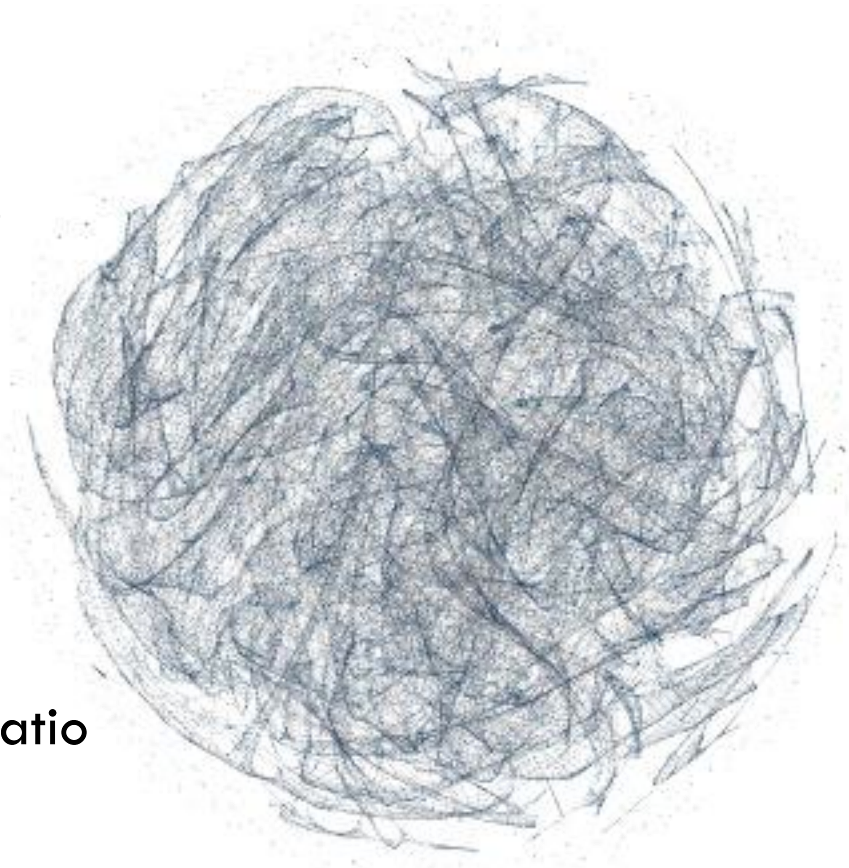
- □ Mining
  - “Find the most influential Graph Processing researcher in Berkeley.”



Large Scale, Graph Processing

- Large-scale
  - Very large data
    - Partitioning and parallel processing are mandatory!
  - Complex analytics
    - Absolute or approximate …
  - Data might evolve in time
    - Fast processing or new algorithms?

- Graph processing
  - Data-driven computations
  - Irregular memory accesses
    - Poor data locality
  - Unstructured problems
  - Low computation-to-data access ratio

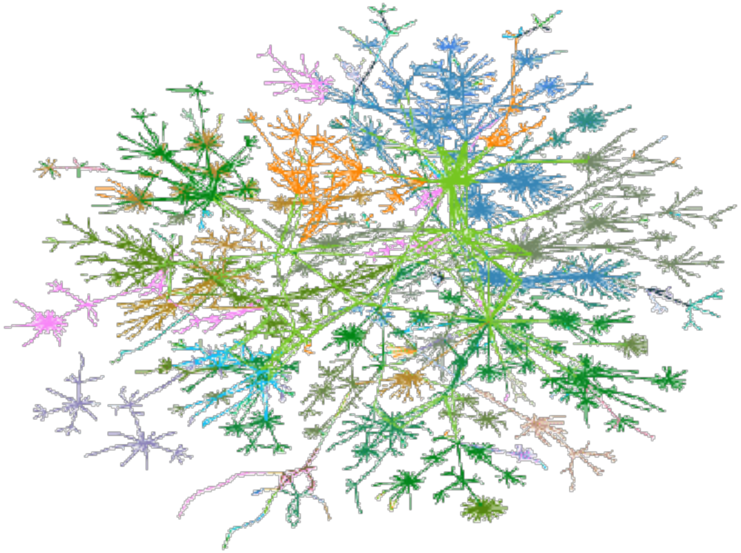


Large Scale Graph Processing

- Graph processing is (very) data-intensive
  - 10x larger graph => 100x or 1000x slower processing
- Graph processing becomes (more) compute-intensive
  - More complex queries => ?x slower processing
- Graph processing is (very) dataset-dependent
  - Unfriendly graphs => ?x slower processing

High performance enables *larger graphs* and support for *more complex analytics.*

Large Scale Graph Processing on GPUs?



+

= ♥ ?

Agenda

- Graphitti:

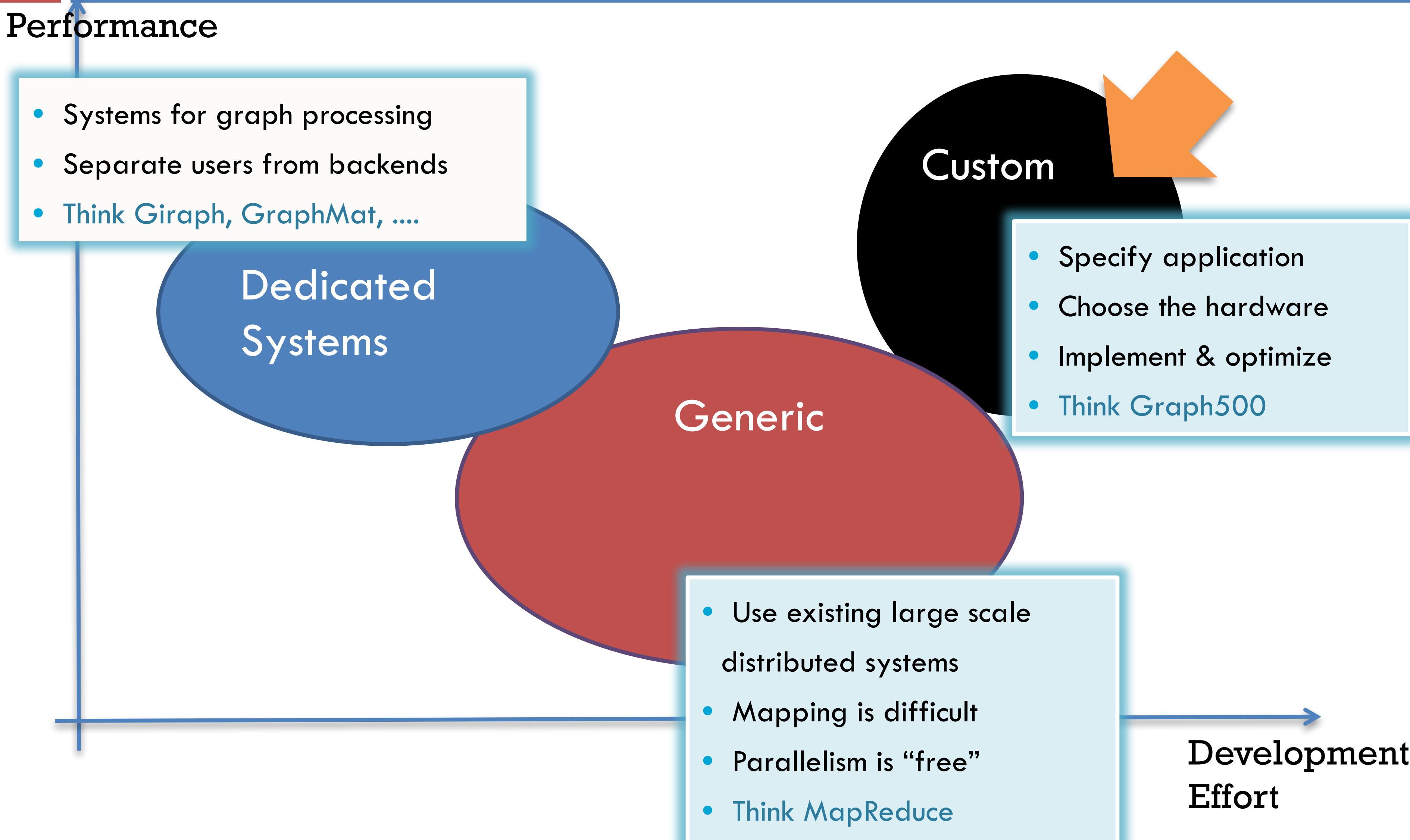
  Investigating the performance factors in graph processing
- HyGraph:

  Yet another GPU-enabled system for graph processing
- Graphpedia:

  Are graphs really everywhere?
- Graphalytics:

  The Landscape of Graph Processing: a Quantitative View

The Landscape of Graph Processing



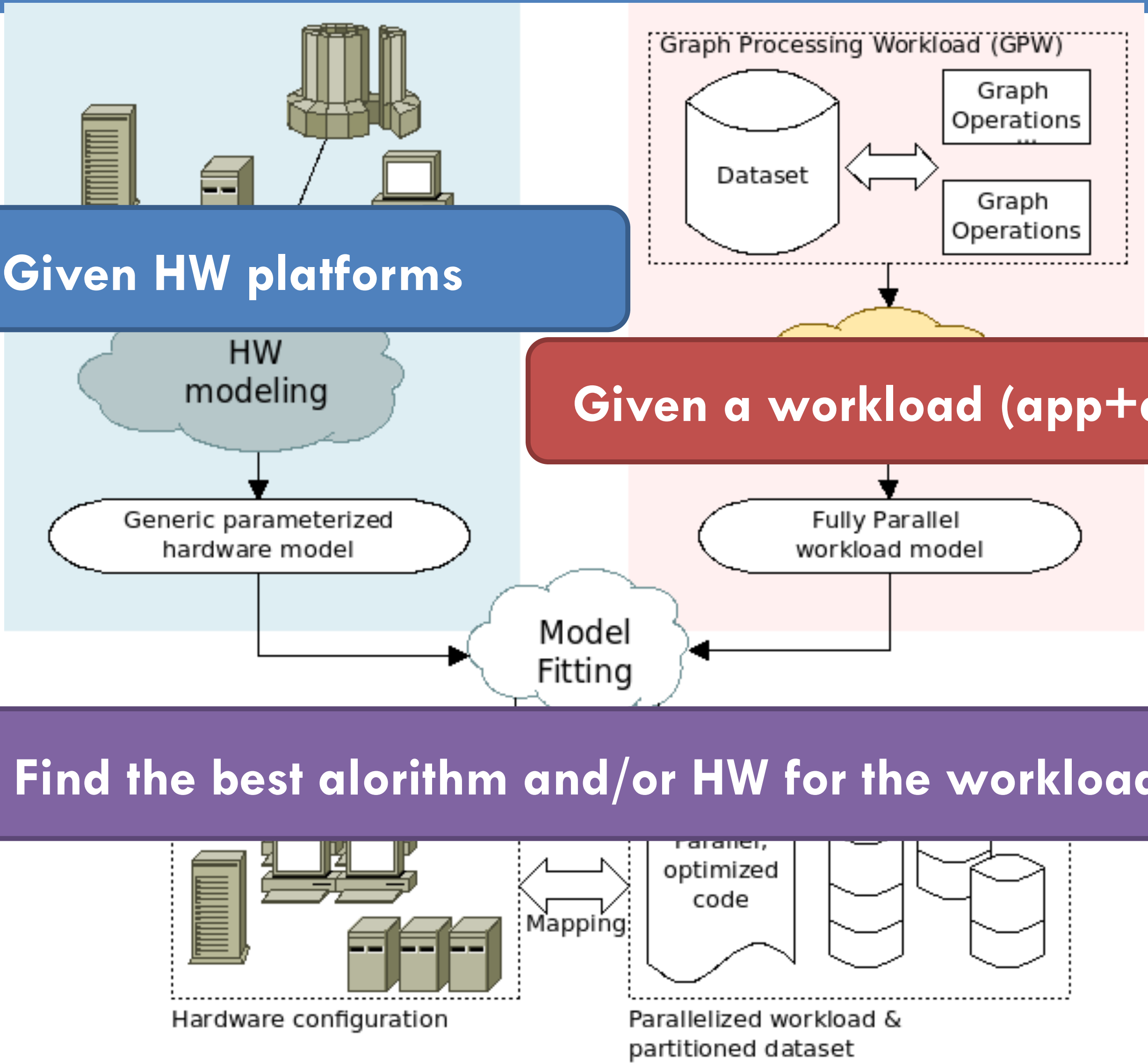
- Systems for graph processing
- Separate users from backends
- Think Giraph, GraphMat, ….

- Specify application
- Choose the hardware
- Implement & optimize
- Think Graph500

- Use existing large scale distributed systems
- Mapping is difficult
- Parallelism is “free”
- Think MapReduce

Graphitti

Our goal: Graphitti


Graph Processing Workload (GPW)
Dataset
Graph Operations
Graph Operations
HW modeling
Generic parameterized hardware model
Fully Parallel workload model
Model Fitting
Parallel, optimized code
Mapping
Hardware configuration
Parallelized workload & partitioned dataset

Given HW platforms

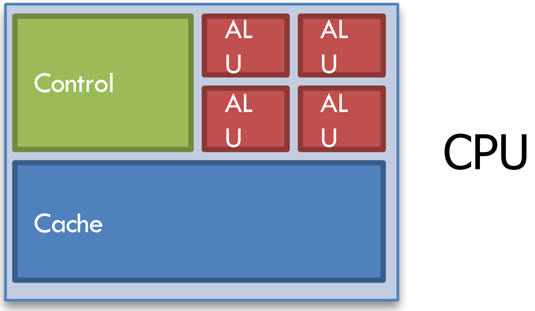
Given a workload (app+data)

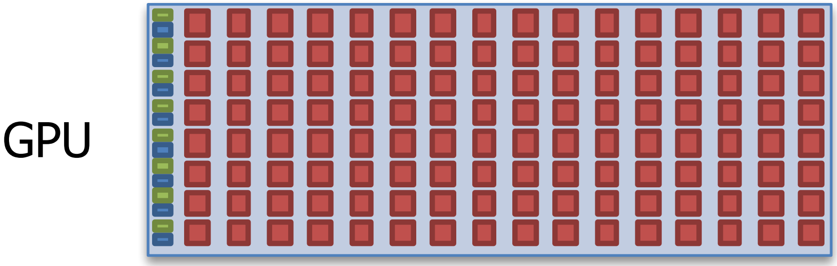
Find the best alorithm and/or HW for the workload

How difficult can it be ?!

Experiment 1: CPU and/or GPU *

- Question:
  - Which multi-/many-core architectures are suitable for graph processing?
- Setup:
  - Three parallelized algorithms
  - Use different graphs
  - Use different hardware

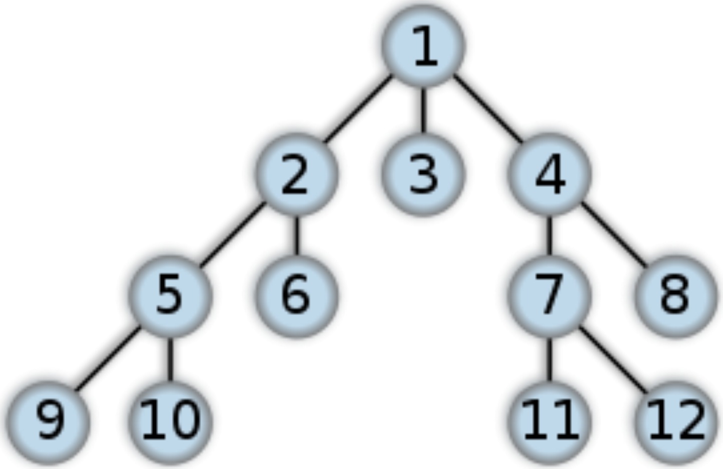




*A.L.Varbanescu et al, "Can Portability Improve Performance? A Graph Processing Case-Study" ICPE'15

Algorithms: BFS➔APSP➔BC

- Graph traversal (Breadth First Search, BFS)
  - Traverses all vertices "in levels"
- All-Pairs Shortest Paths (APSP)
  - Repeat BFS for each vertex
- Betweenness Centrality (BC)
  - APSP once to determine paths
  - Bottom-up BFS to count paths



- Implementation in OpenCL*
  - Same algorithm
  - CPU- and GPU-specific **tuning** applied

*Ate Penders MSc thesis
"Accelerating graph processing using modern accelerators"

Data sets & devices

	Abbreviation	Vertices	Edges	Diameter	Avg. Degree
Wikipedia Talk Network	WT	2,394,385	5,021,410	9	2,10
California Road Network	CR	1,965,206	5,533,214	850	2,81
Rodinia Graph 1M	1M	1,000,000	6,000,000	36	6,00
Stanford Web Graph	SW	281,903	2,312,497	740	8,20
EU Email Communication Network	EU	265,214	420,045	13	1,58
Star	ST	100,000	99,999	1	0,99
Chain	CH	100,000	99,999	99,999	1,00
Epinions Social Network	ES	75,879	508,837	13	6,70
Rodinia Graph 64K	64K	64,000	393,216	28	6,14
Wikipedia Vote Network	VW	7,115	103,689	7	14,57
Rodinia Graph 4K	4K	4000	25,356	19	6,38

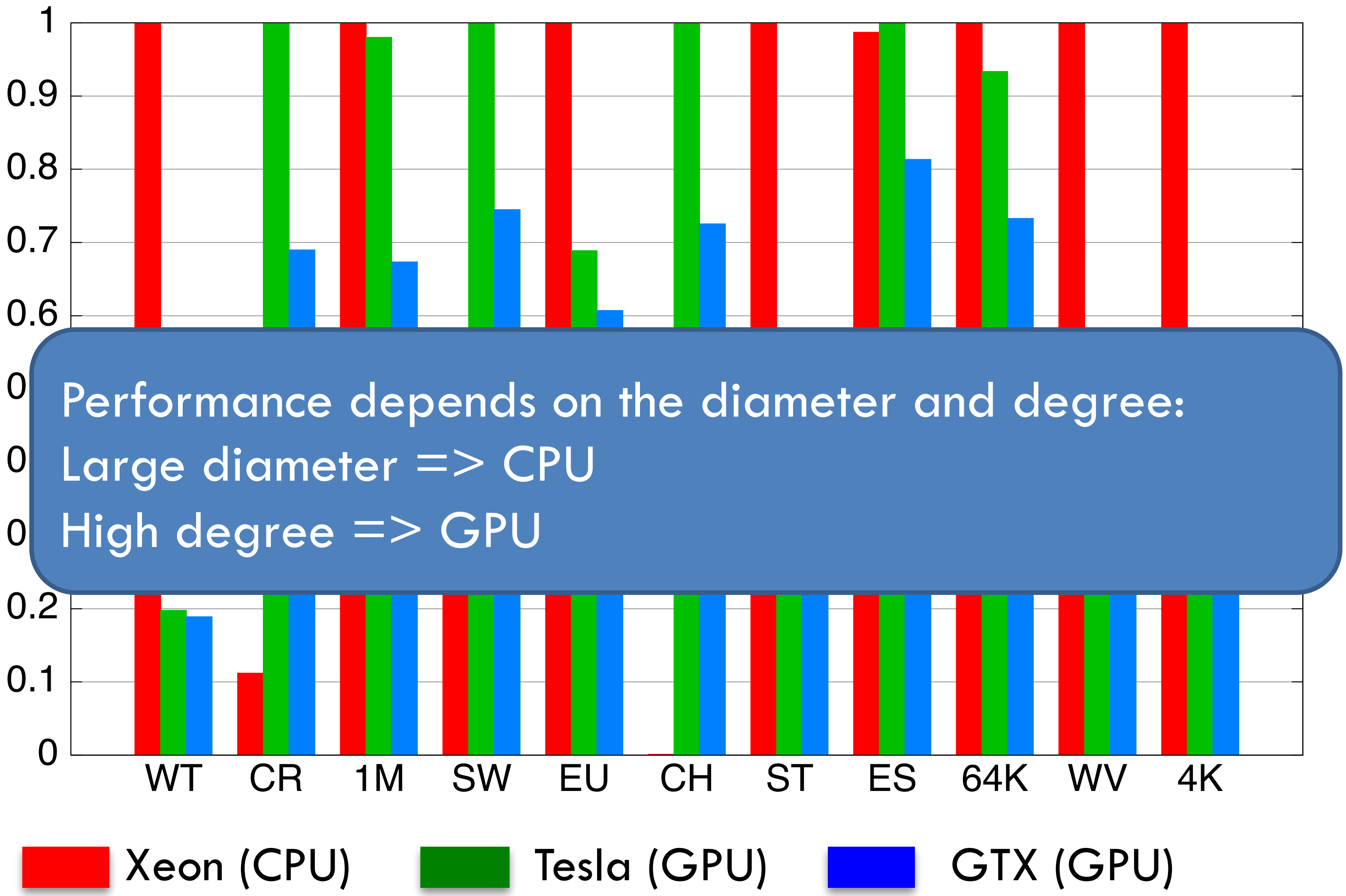
- □ **Devices**

  Intel(R) Xeon(R) CPU E5620 @ 2.40GHz

  GeForce GTX 480

  Tesla C2050 / C2070

BFS – normalized

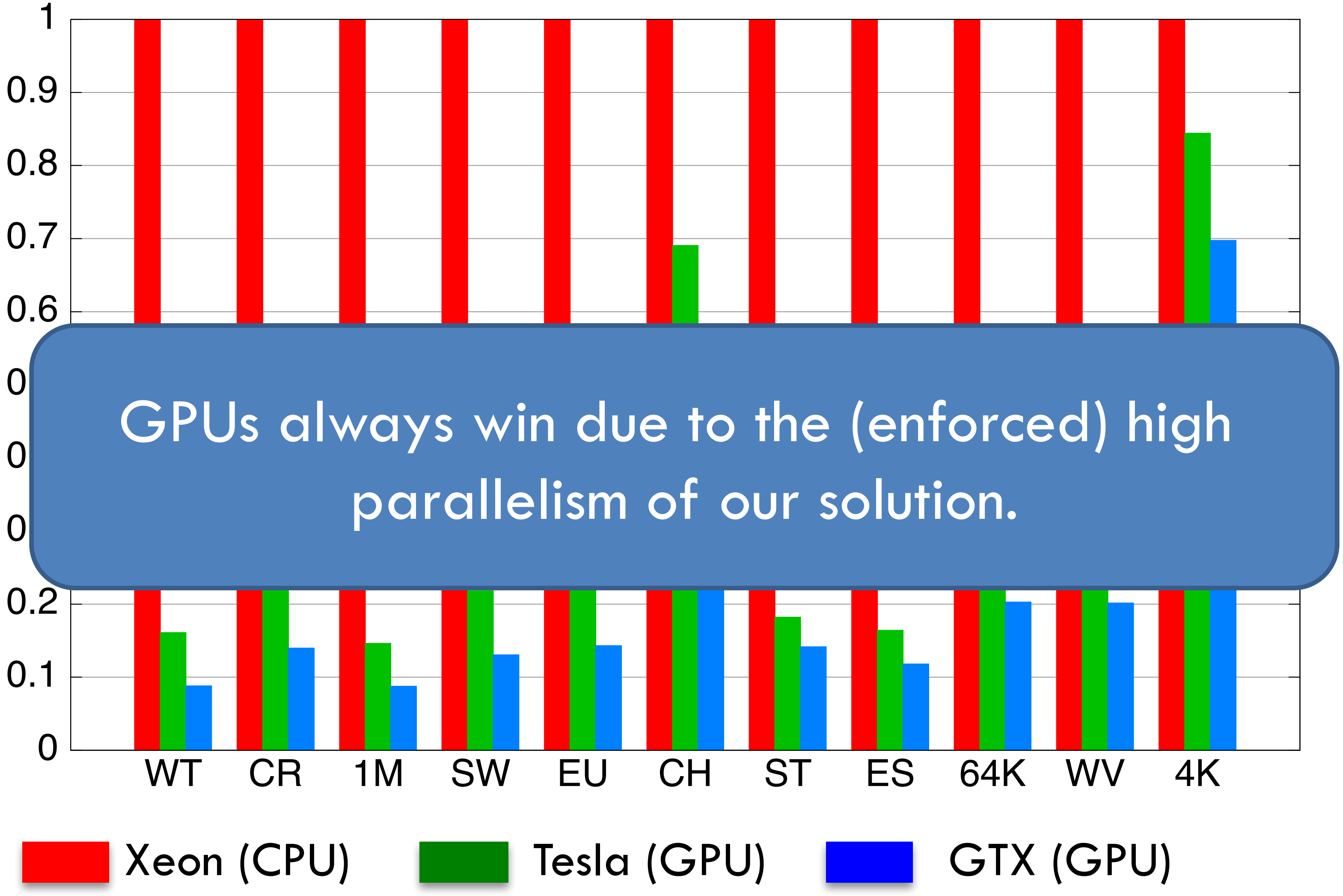


Performance depends on the diameter and degree:

Large diameter => CPU

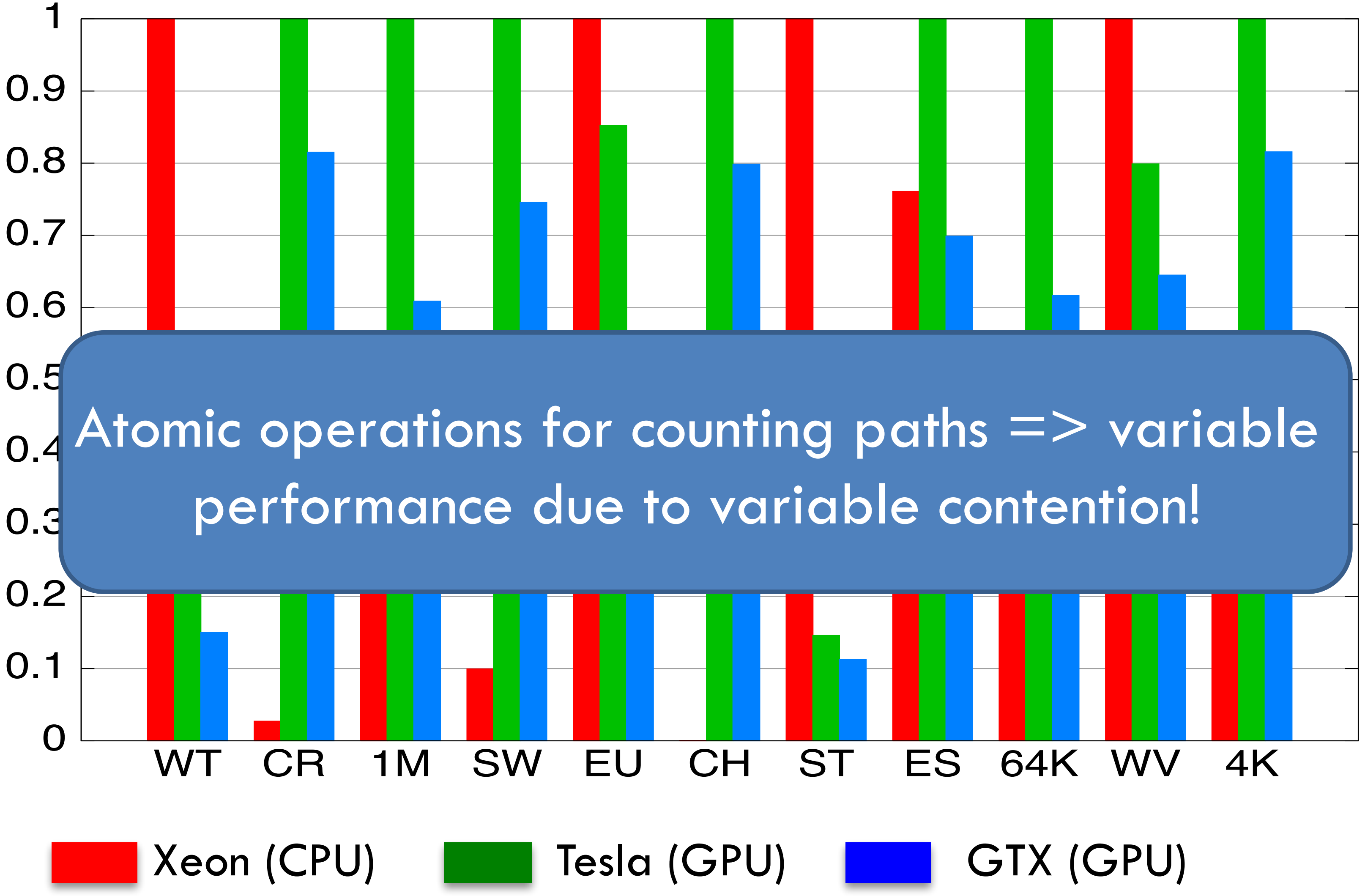
High degree => GPU

APSP - normalized



GPUs always win due to the (enforced) high parallelism of our solution.

BC - normalized



Atomic operations for counting paths => variable performance due to variable contention!

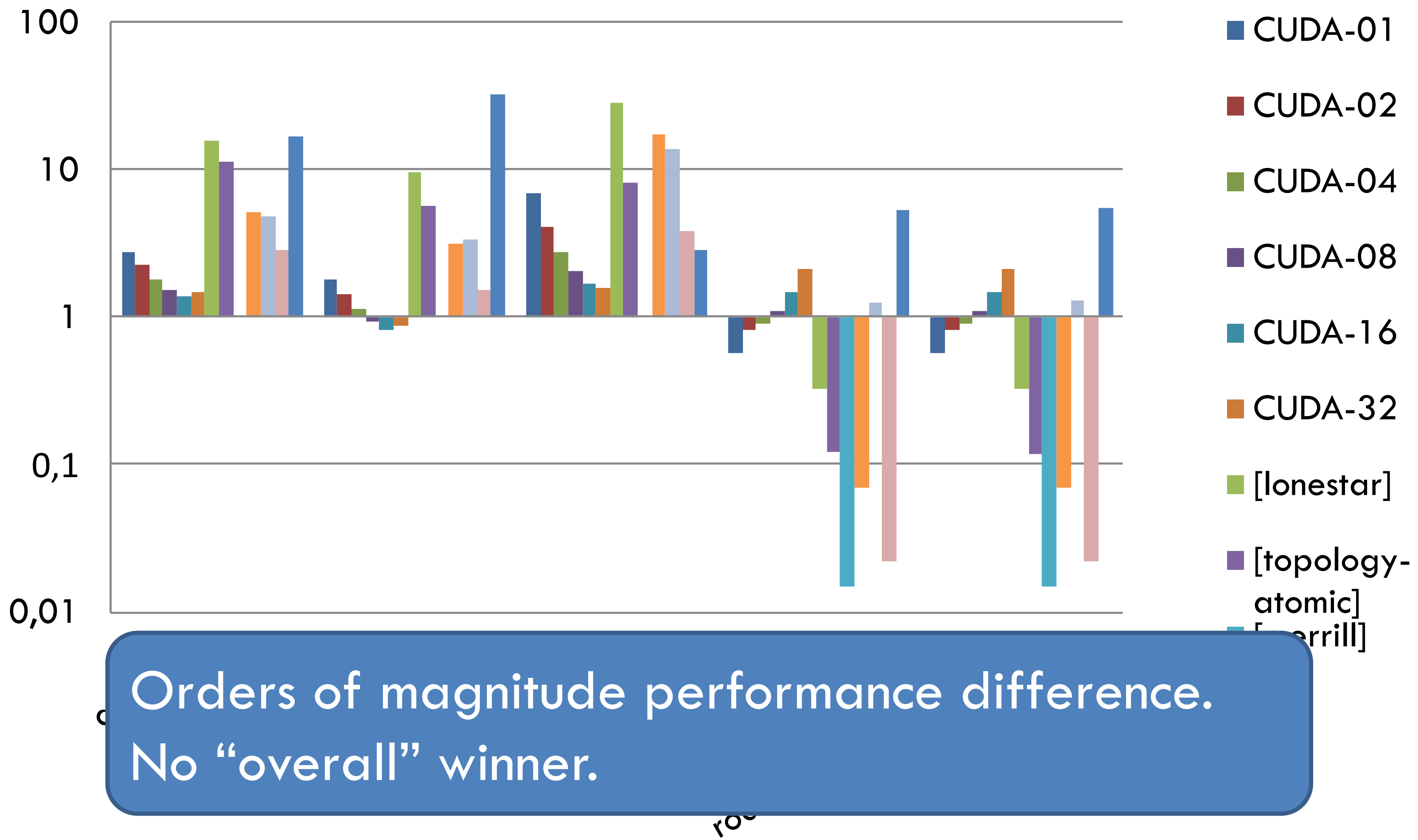
Lessons learned

- Increased algorithm complexity => increase parallelism => improved performance ?
- Dataset properties + data representation => increase parallelism => improved performance ?
- Synchronization can be a hidden bottleneck
- We have no clear understanding of graph "sizes"
  - # vertices or # edges? Diameter? Other properties?
- Graphs seem to be CPU or GPU friendly
  - Heterogeneous processing?

Experiment 2: BFS traversals

- Question:
  - Is there a best BFS algorithm?
    - On GPUs ?
    - Overall ?
- Setup:
  - Run different BFS implementations
    - LonestarGPU
    - Warp-grain
  - Run on different graphs
    - 6 datasets
  - Run on different hardware

Normalized on naïve GPU, kernel



Orders of magnitude performance difference.
No "overall" winner.

Lessons learned

- Depending on the graph …
  - Large variability in performance (fastest to slowest ratio)
  - The relative performance of BFS implementation varies.
    - Fastest on one graph CAN BE slowest on another graph.
- Data representation and data structures make a BIG difference
- A naive CPU implementation can be competitive with some of the GPU implementations.
  - On small graphs (GPUs are underutilized)
  - When data transfer is an issue (think BFS)

Experiment 3: PageRank*

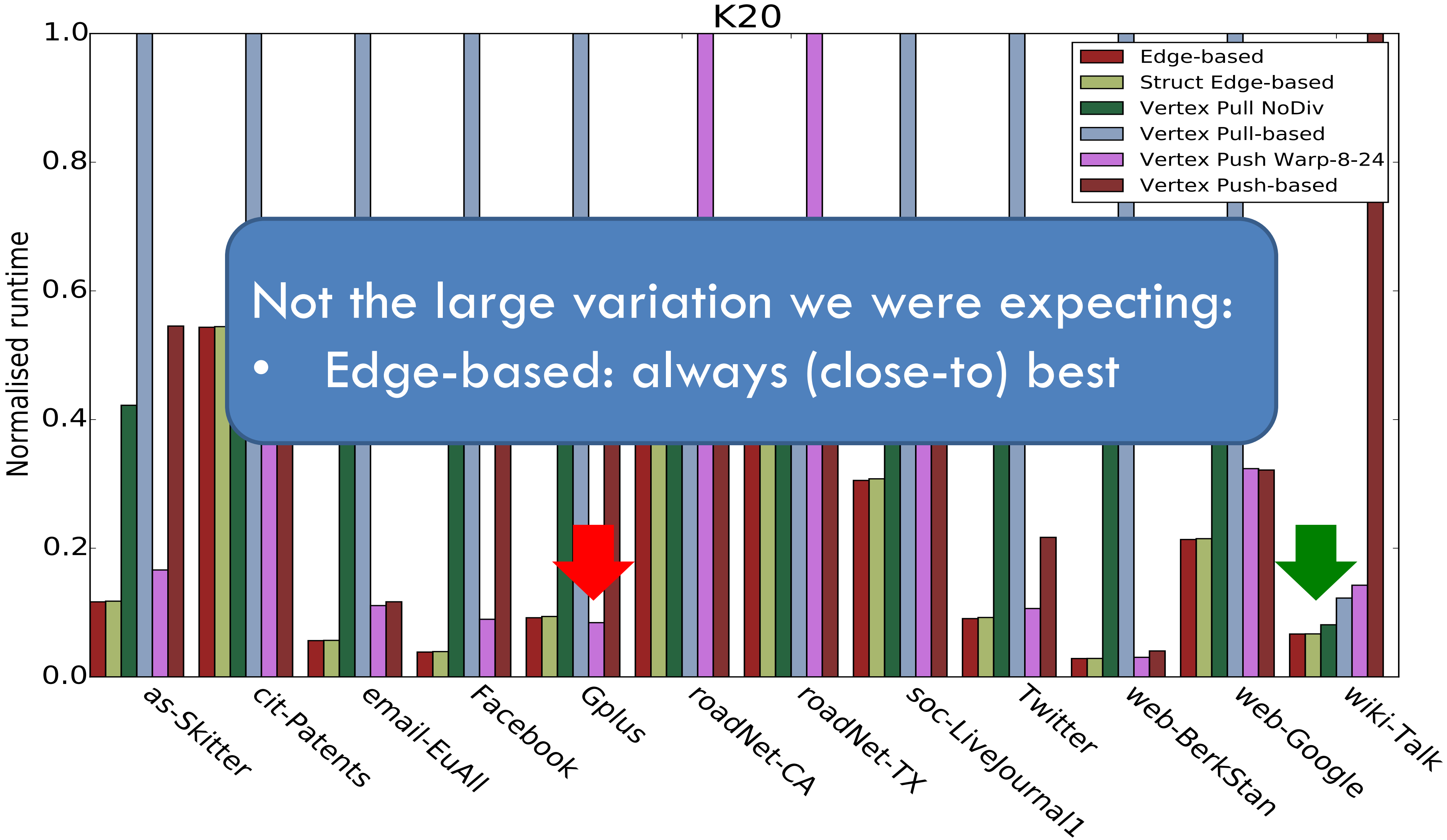
- Question:
  - How does PageRank depend on data?
- Setup:
  - Run different PageRank implementations
    - Edge-based (+variations)
    - Vertex-based, pull (+variations)
    - Vertex-based, push (+variations)
  - Run on different graphs
    - SNAP datasets
    - Synthetic datasets
  - Run on different GPUs
    - Only K20 shown here

*M.E.Verstraaten et al, "Quantifying the Performance Impact of Graph Structure on Neighbours Iteration Strategies" PELGA'15 under review

Graphs

	#V	#E	Triangles	Diameter	90% Diam
as-Skitter	1.696.415	11.095.298	28.769.868	25	6
cit-Patents	3.774.768	16.518.948	7.515.023	22	9,4
email-EuAll	265.214	420.045	267.313	14	4,5
Facebook	4.039	88.234	1.612.010	8	4,7
GPlus	107.614	13.673.453	1.073.677.742	6	3
roadNet-CA	1.965.206	2.766.607	120.676	849	500
roadNet-TX	1.379.917	1.921.660	82.869	1.054	670
soc-Livejournal	**4.847.571**	**68.993.773**	285.730.264	16	6,5
Twitter	81.306	1.768.149	13.082.506	7	4,5
web-BerkStan	685.230	7.600.595	64.690.980	514	9,9
web-Google	875.713	5.105.039	13.391.903	21	8,1
wikiTalk	2.394.385	5.021.410	9.203.519	9	4

PageRank on K20 (lower is better)



Not the large variation we were expecting:

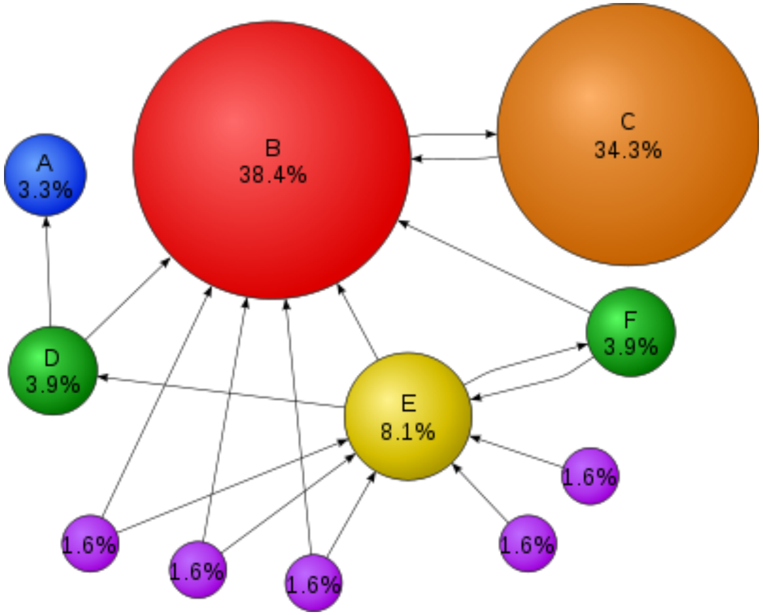
- Edge-based: always (close-to) best

PageRank in detail

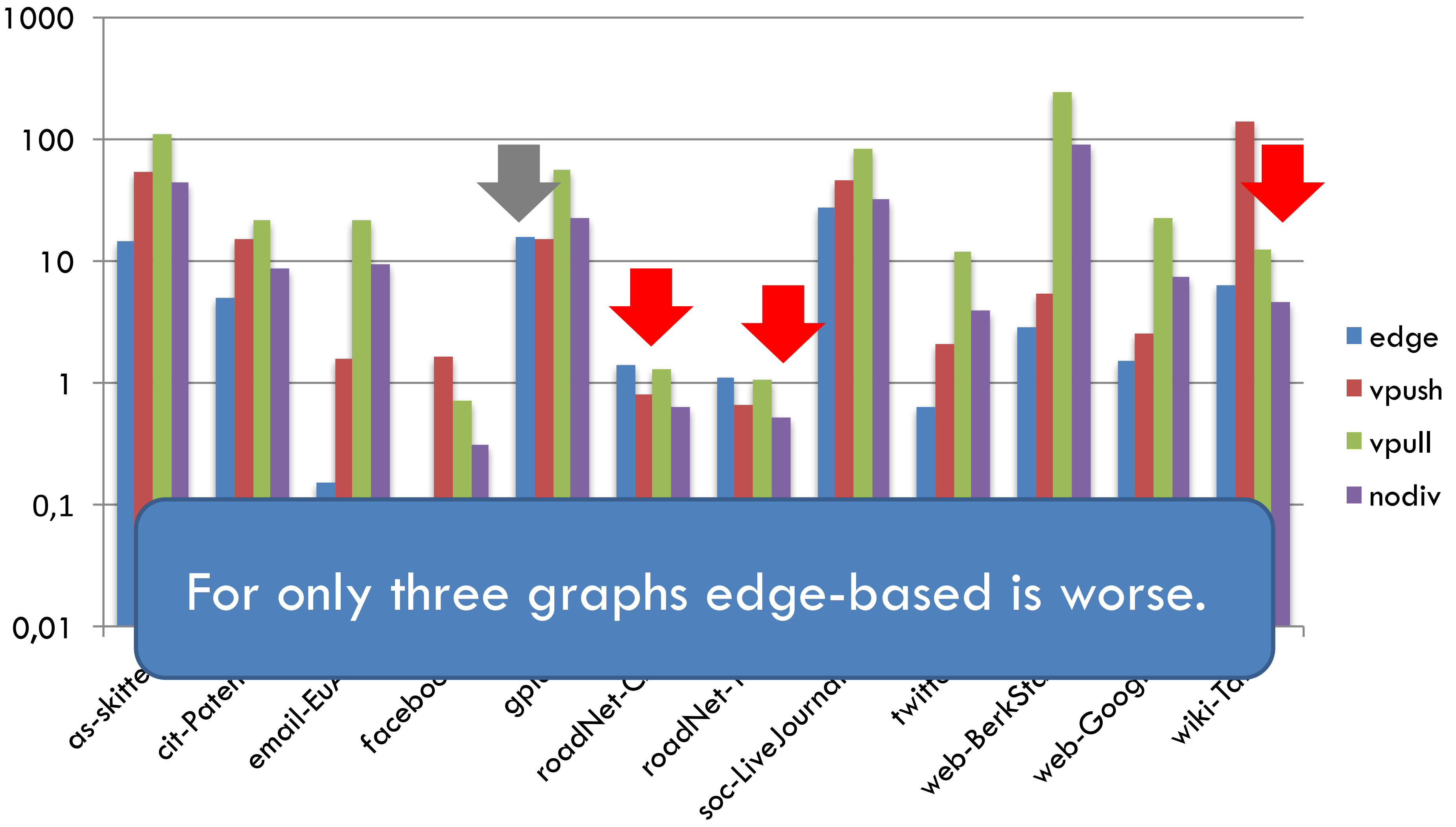
Computing PageRank

- Iterative algorithm
- Every iteration updates all vertices:

$$PR(v) = \frac{1-d}{|V|} + d \sum_{w \in N(v)} \frac{PR(w)}{\rho(w)}$$



Execution time (real graphs)



For only three graphs edge-based is worse.

Attempt to model (1)

Statistical modeling:

- Step 1. Collect data
  - Execution time, performance counters, …
- Step 2. Use a modeling tool => prediction model
-

We have plenty of data !
… or do we?

Result = performance prediction

How to get relevant data?

- Find graphs with similar properties
  - What is similar?
- Use graph generators
  - How fine grained are they? How diverse?
- Use synthetic graphs

Build your own graph generator to fine-tune graph properties.

Harder than we thought…*

*M.E.Verstraaten et al, "Synthetic Graph Generation for Systematic Exploration of Graph Structural Properties", PELGA'16

Attempt to model (2)

Analytical modeling

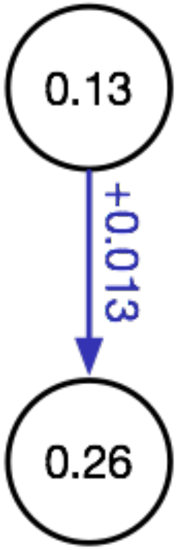
- Step 1. Build a work model
- Step 2. Define a GPU parallel execution model
- Step 3. "Deploy" the work model on the GPU model

Result = performance prediction

Skip

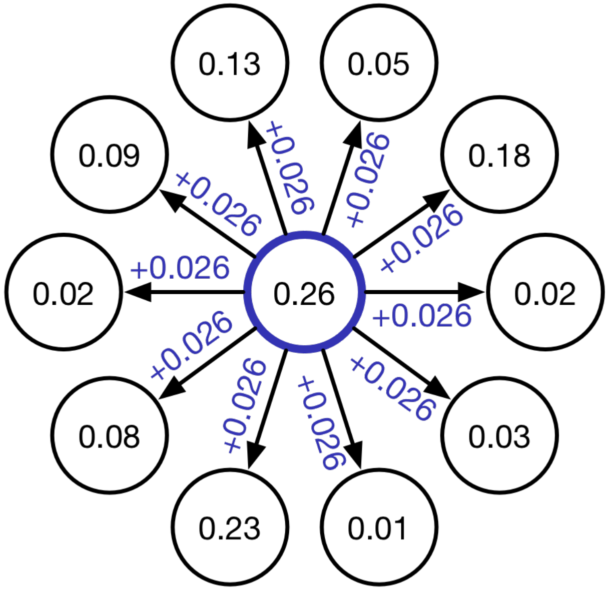
Edge-based

```
function EdgeBased($edge$)
    $origin \leftarrow edge.origin$
    $dest \leftarrow edge.destination$
    $outgoingRank \leftarrow \frac{origin.pagerank}{origin.degree}$
    $dest.newRank.atomicAdd(outgoingRank)$
end function
```



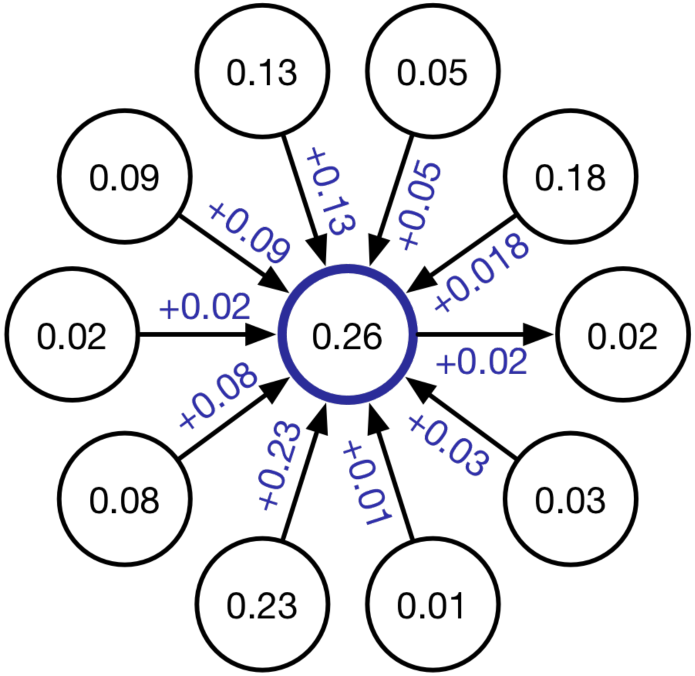
Vertex-centric push

```
function $\textsc{VertexPush}(vertex)$
    if $vertex.degree \neq 0$ then
        $outgoingRank \leftarrow \frac{vertex.pagerank}{vertex.degree}$
    end if
    for $nbr \in vertex.neighbours$ do
        $nbr.newRank.atomicAdd(outgoingRank)$
    end for
end function
```



Vertex-centric pull

```
function $\textsc{VertexPull}(vertex)$
    $newRank = 0$
    for $nbr \in vertex.neighbours$ do
        $newRank \mathrel{+}= \frac{nbr.pagerank}{nbr.degree}$
    end for
    $vertex.newRank \leftarrow newRank$
end function
```



Build a work model

- Graph processing is memory bound => Use *only* read, write, and atomics

$$
\begin{aligned}
T_{push} &= 5 * |V| * T_{read} + 2 * |V| * T_{write} + (|V| + |E|) * T_{atom} \\
T_{pull} &= (3 * |E| + 2 * |V|) * T_{read} + 3 * |V| * T_{write} + |V| * T_{atom} \\
T_{NoDiv} &= (|E| + 4 * |V|) * T_{read} + 3 * |V| * T_{write} + |V| * T_{atom} \\
T_{edge} &= (3 * |E| + 2 * |V|) * T_{read} + 2 * |V| * T_{write} + (|V| + |E|) * T_{atom}|
\end{aligned}
$$

- Validated using hardware counters

Define a GPU execution model

- Work distribution:
  - Workers = #SM * cores_per_SM
  - Simplistic: T = Work / Workers
    - How about the scheduling? Warps and such?

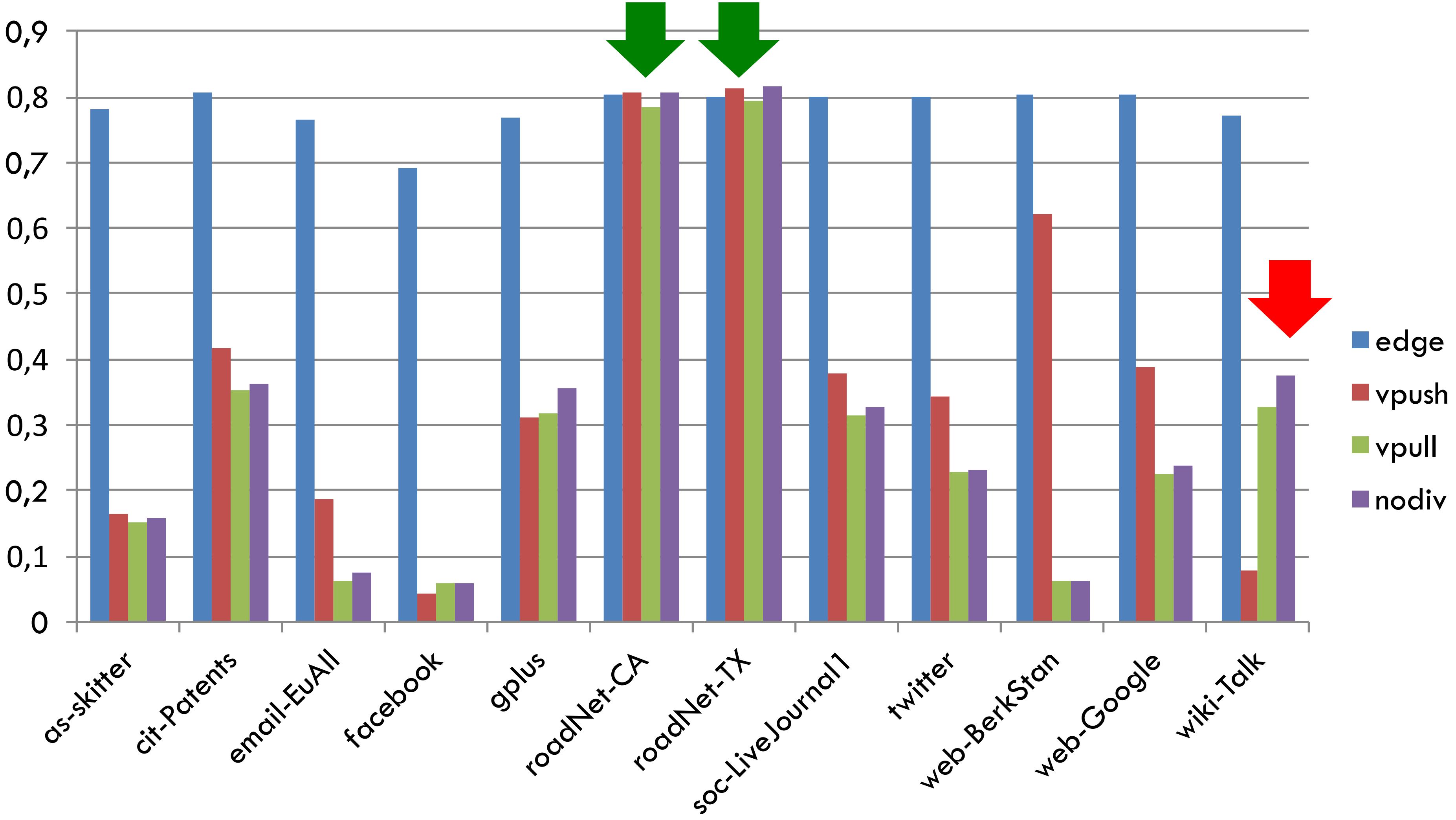
Use performance metrics to define an approximation of the schedule.

“Deploy” work on GPU

- Calibrate the values
  - T_read, T_write, T_atom, …
- Find the right approximation for the schedule

- What are the most important GPU metrics that correlate with performance?
  - Achieved occupancy? Multiprocessor activity? Reads?Writes? …

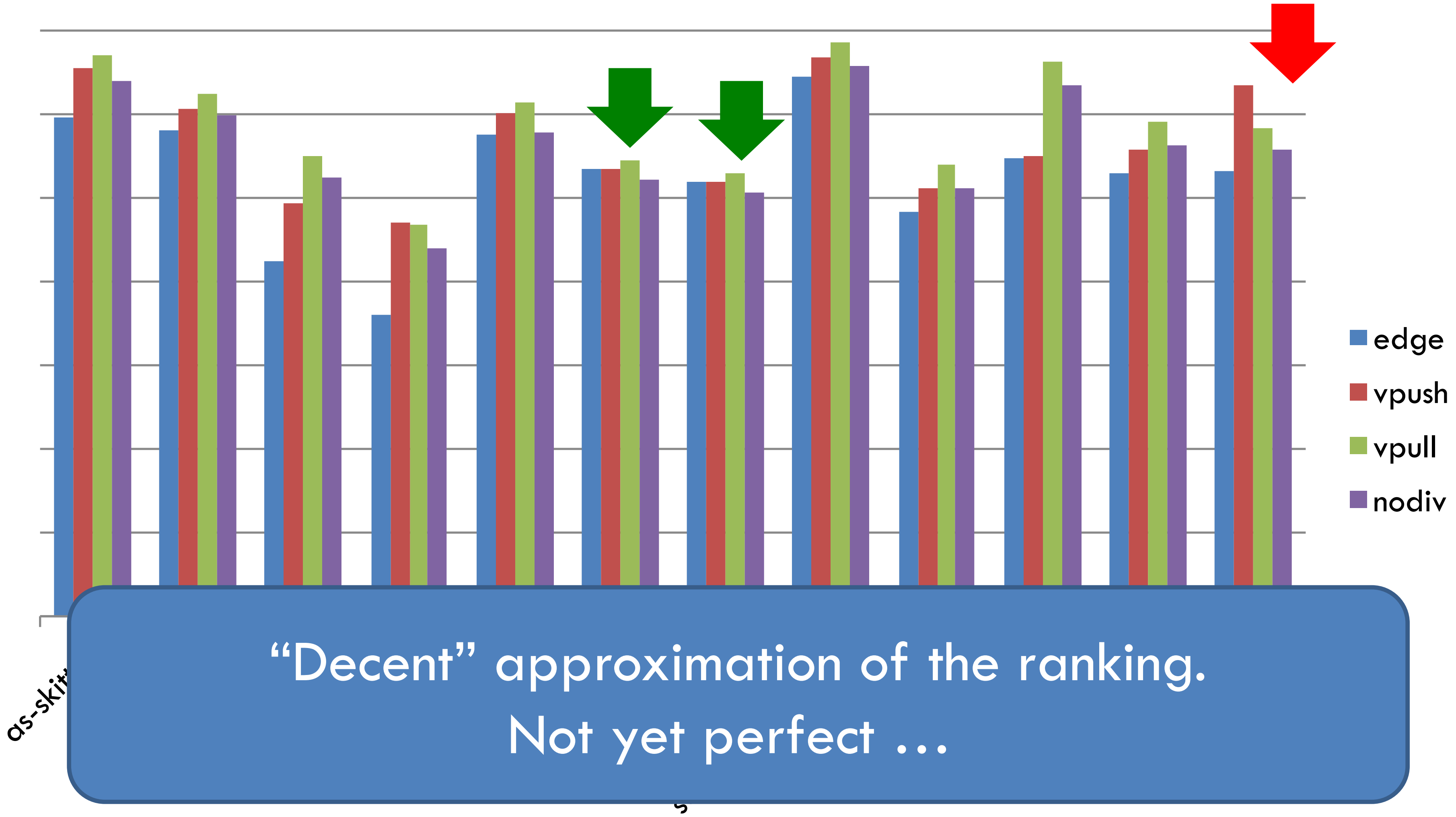
Achieved occupancy



Current “coarse” approximation

- Ignore latencies
  - Bad idea if the differences between them are large
- Use occupancy to “reduce” the number of workers
  - Real_workers = #SMs * cores_SM * occcupancy
- Rank performance
  - No accurate numbers, but we mostly care about ranking

Performance ranking



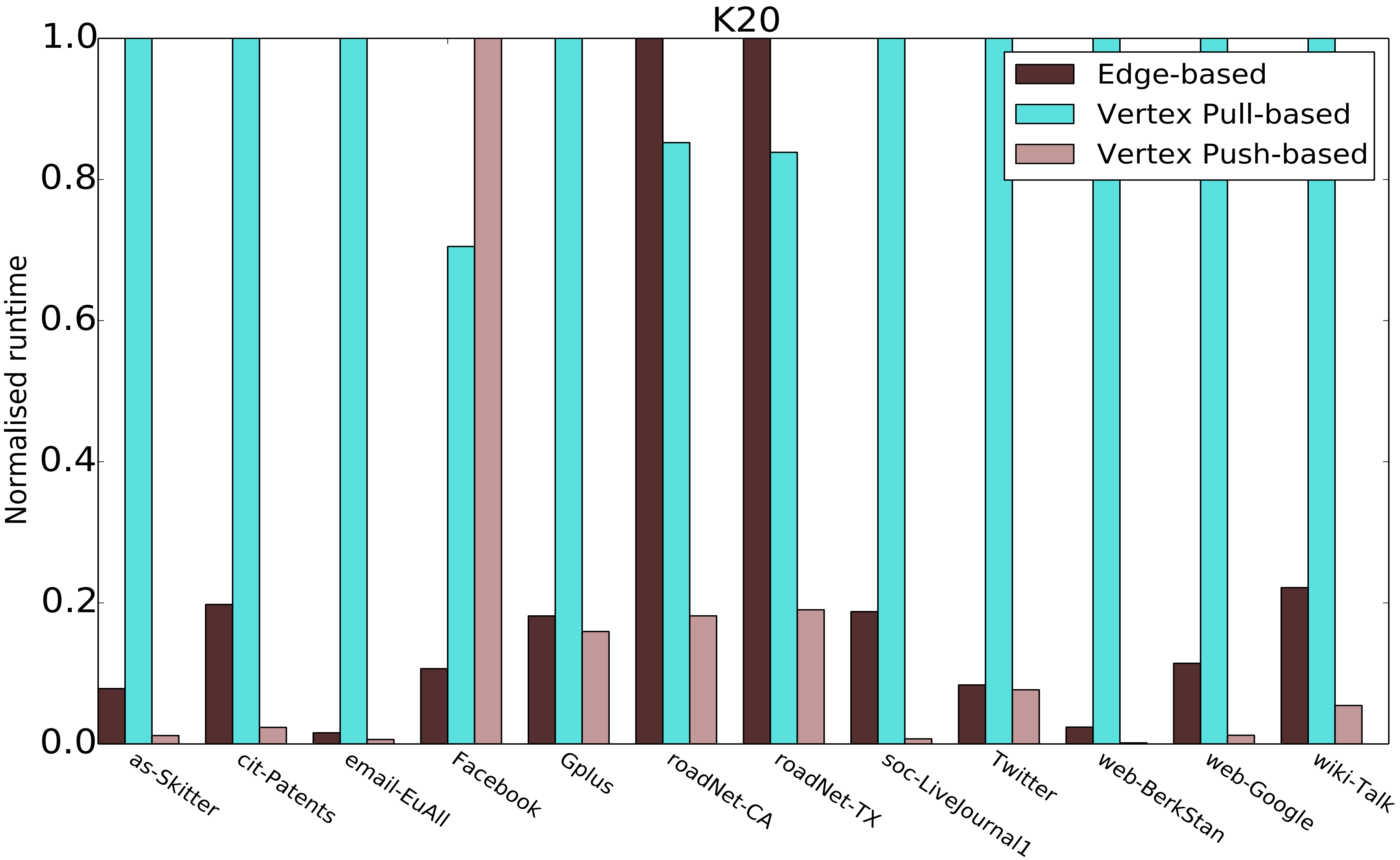
“Decent” approximation of the ranking.
Not yet perfect …

More general problem

- Which graph properties impact performance most?
  - Still trying a statistical approach here
  - PR: degree distribution, ordering, …
- Can we use them to estimate occupancy?
  - PR: yes
  - BFS: not yet

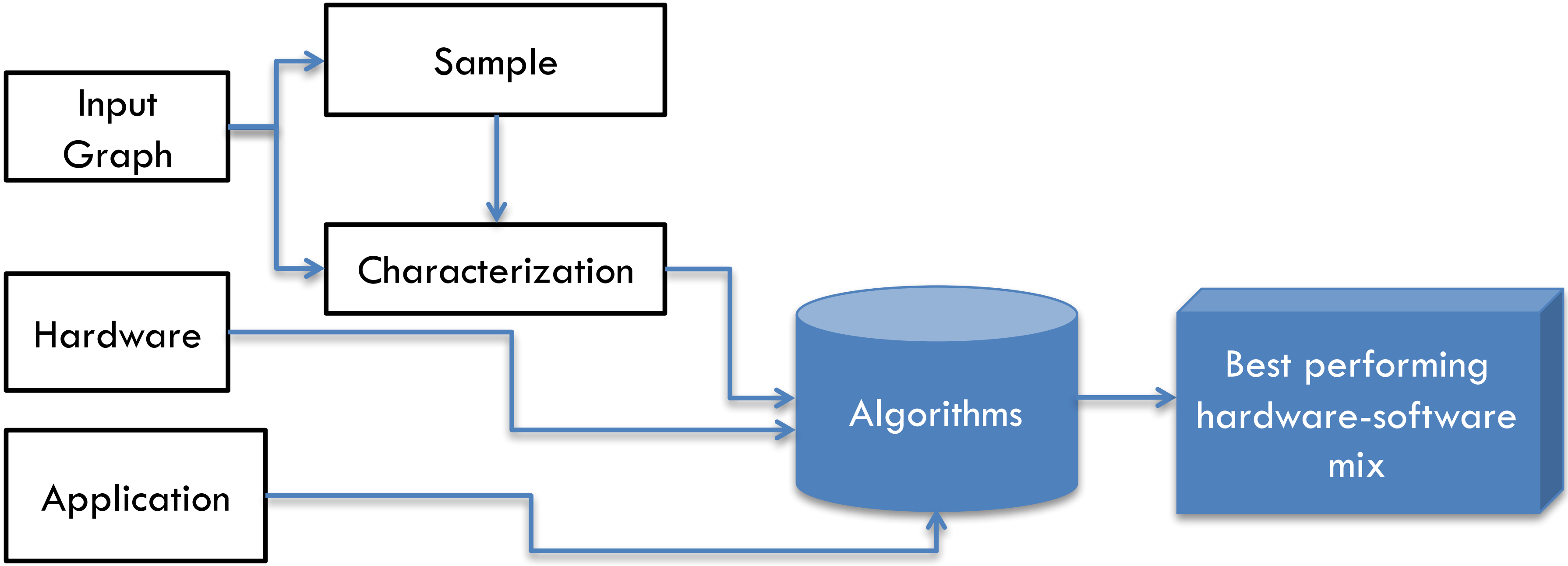
- In general: attempt to understand the impact of properties on occupancy.
  - Statistical modeling again? Simulation? Microbenchmarking?

How about BFS?



Graph-centric framework

- Understand **graph features**
  - What makes a graph "special"
- Select the best algorithm for a given graph



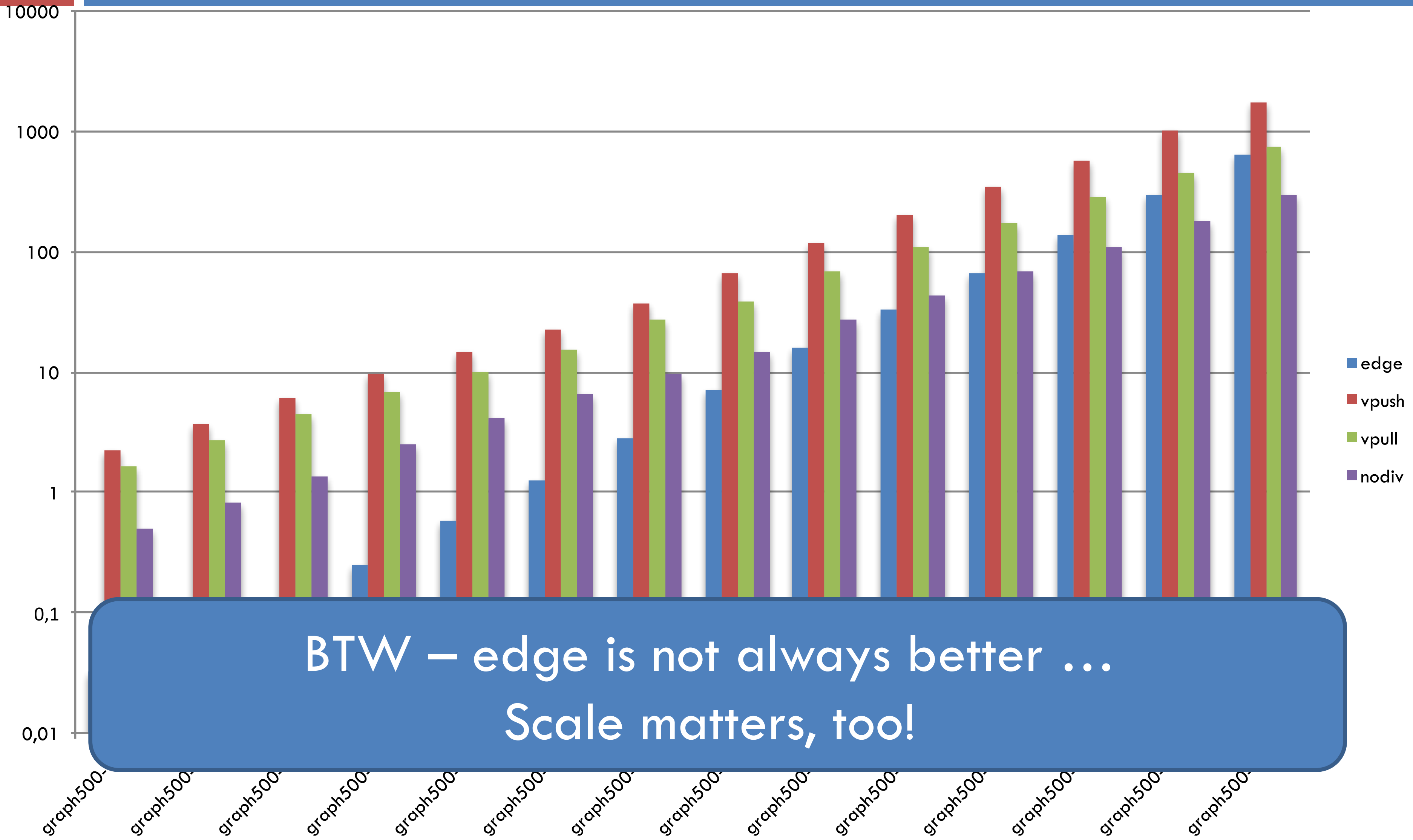
Lessons learned

- Performance gaps are quite large

- Graph properties matter!
- Scale also matters!

- Machine learning is not a solution for everything
  - There are not enough graphs out there to drive statistical approaches.

Execution time (Graph500 graphs)

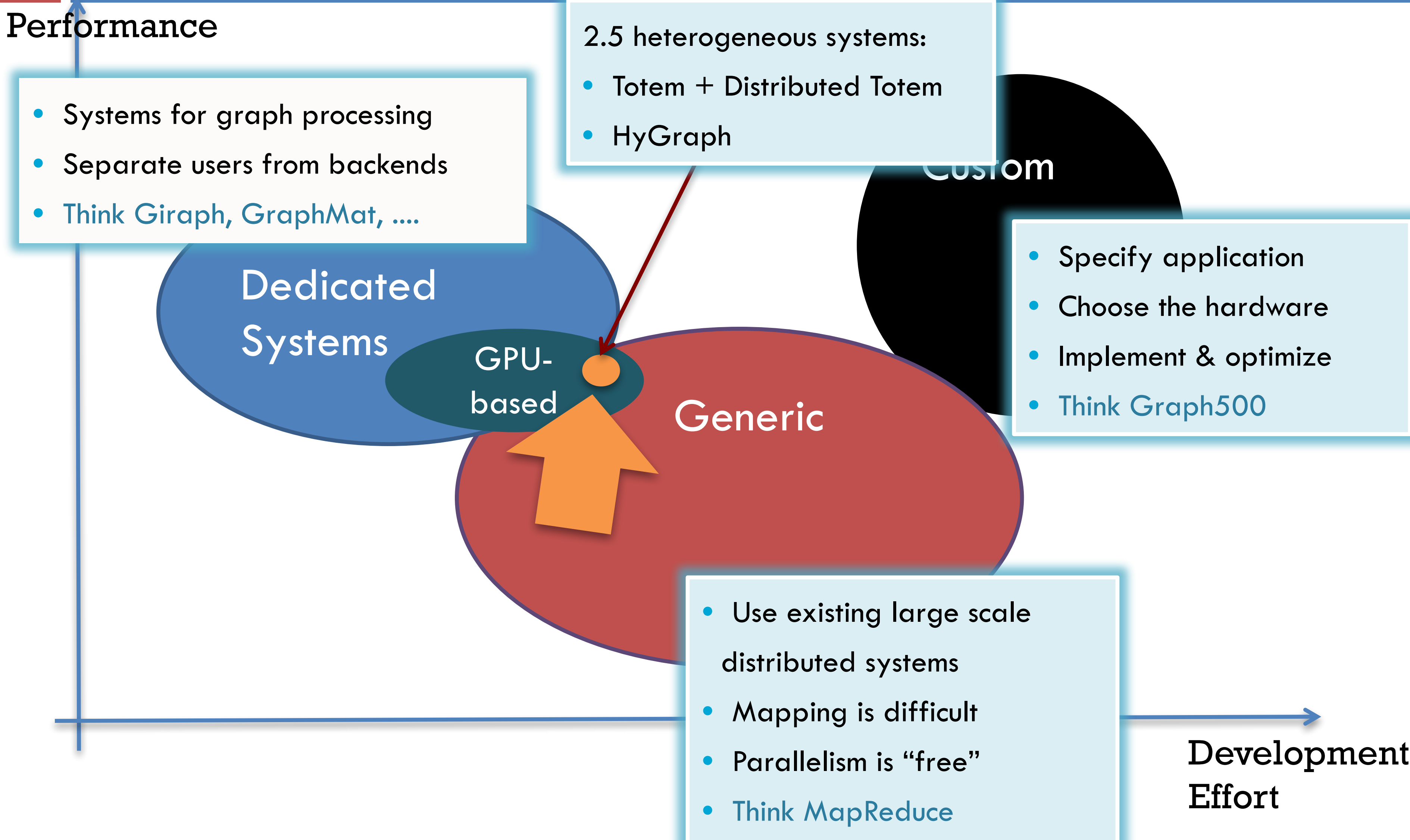


BTW – edge is not always better …

Scale matters, too!

Questions?

The Landscape of Graph Processing



- Systems for graph processing
- Separate users from backends
- Think Giraph, GraphMat, ...

2.5 heterogeneous systems:

- Totem + Distributed Totem
- HyGraph

- Specify application
- Choose the hardware
- Implement & optimize
- Think Graph500

- Use existing large scale distributed systems
- Mapping is difficult
- Parallelism is "free"
- Think MapReduce

GPU-enabled systems

HyGraph

Conclusions

GPU-enabled graph processing*

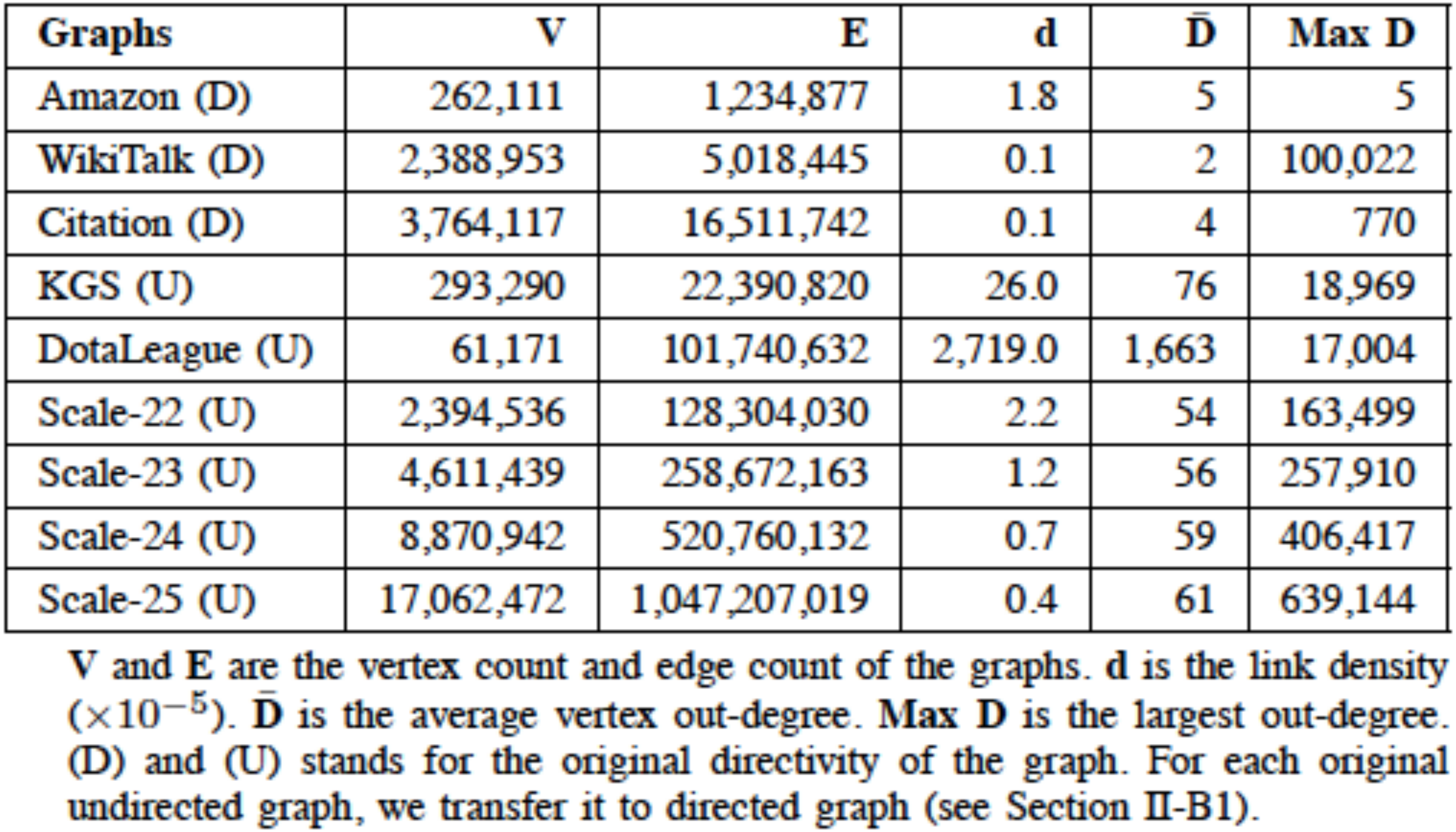
- MapGraph
    - CPU, single- and multi-GPU versions
    - Vertex-centric API
- Medusa
    - Single-node, multiple GPUs
    - Programmability-driven, based on BSP
- Totem
    - Heterogeneous, multi-GPU
    - Based on BSP
    - Also looks at partitioning
- We know of …
    - CuSha
    - Gunrock
    - Ligra
    - …

*Yong Guo et. al, "An Empirical Performance Evaluation of GPU-Enabled Graph-Processing Systems", CCGrid 2015

Setup: Algorithms & Systems

- Algorithms
  - BFS (traversal)
  - PageRank
  - Weakly connected components
- Hardware: GPU-enabled nodes in DAS4
  - GTX480 (most results), GTX580, and K20
- Processing systems:
  - Totem - GPU-only and Hybrid
  - Medusa – single- and multi-GPU
  - MapGraph – single-GPU

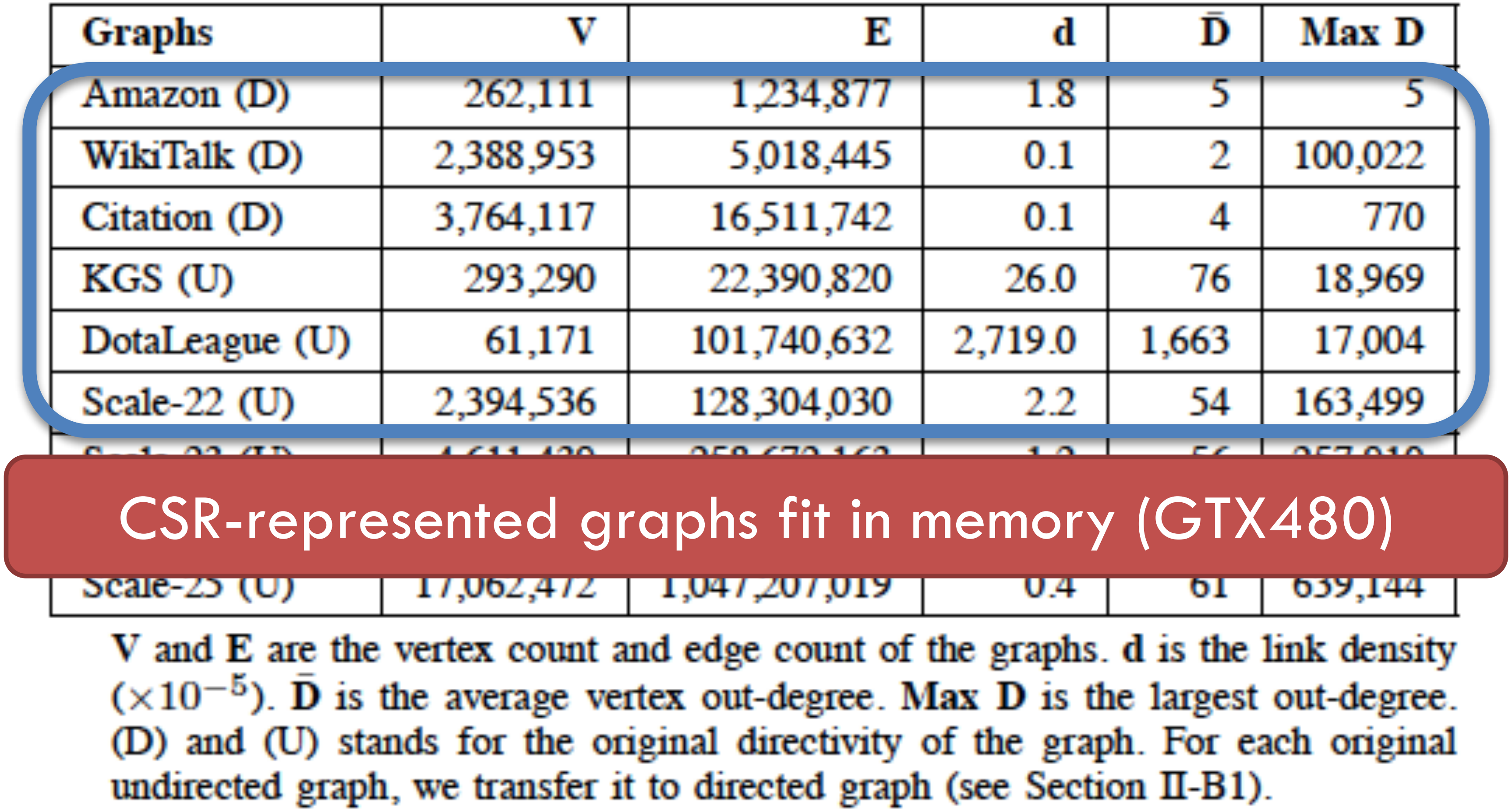
Setup: datasets



Graphs	**V**	**E**	**d**	$\bar{\mathbf{D}}$	**Max D**
Amazon (D)	262,111	1,234,877	1.8	5	5
WikiTalk (D)	2,388,953	5,018,445	0.1	2	100,022
Citation (D)	3,764,117	16,511,742	0.1	4	770
KGS (U)	293,290	22,390,820	26.0	76	18,969
DotaLeague (U)	61,171	101,740,632	2,719.0	1,663	17,004
Scale-22 (U)	2,394,536	128,304,030	2.2	54	163,499
Scale-23 (U)	4,611,439	258,672,163	1.2	56	257,910
Scale-24 (U)	8,870,942	520,760,132	0.7	59	406,417
Scale-25 (U)	17,062,472	1,047,207,019	0.4	61	639,144

V and **E** are the vertex count and edge count of the graphs. **d** is the link density ($\times 10^{-5}$). $\bar{\mathbf{D}}$ is the average vertex out-degree. **Max D** is the largest out-degree. (D) and (U) stands for the original directivity of the graph. For each original undirected graph, we transfer it to directed graph (see Section II-B1).

Setup: datasets

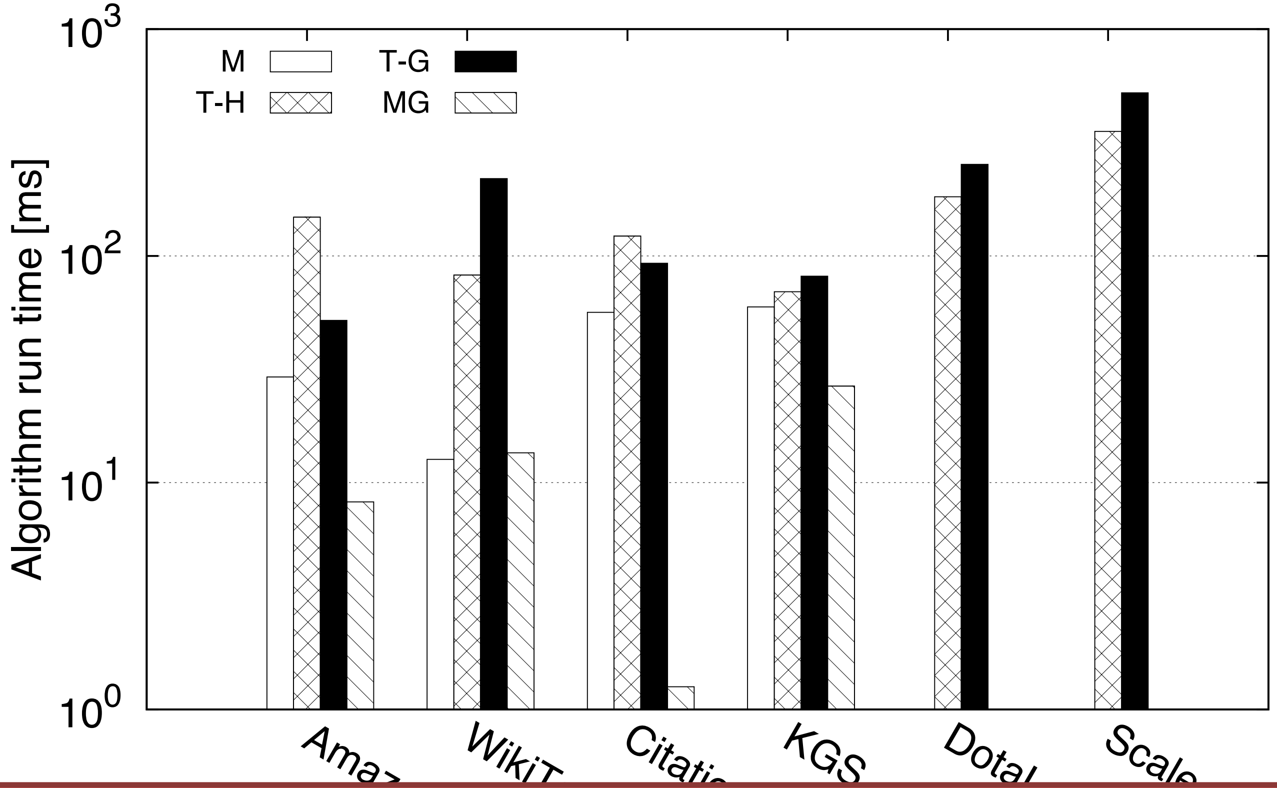


Graphs	V	E	d	D̄	Max D
Amazon (D)	262,111	1,234,877	1.8	5	5
WikiTalk (D)	2,388,953	5,018,445	0.1	2	100,022
Citation (D)	3,764,117	16,511,742	0.1	4	770
KGS (U)	293,290	22,390,820	26.0	76	18,969
DotaLeague (U)	61,171	101,740,632	2,719.0	1,663	17,004
Scale-22 (U)	2,394,536	128,304,030	2.2	54	163,499
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Scale-25 (U)	17,062,472	1,047,207,019	0.4	61	639,144

V and **E** are the vertex count and edge count of the graphs. **d** is the link density ($\times 10^{-5}$). **D̄** is the average vertex out-degree. **Max D** is the largest out-degree. (D) and (U) stands for the original directivity of the graph. For each original undirected graph, we transfer it to directed graph (see Section II-B1).

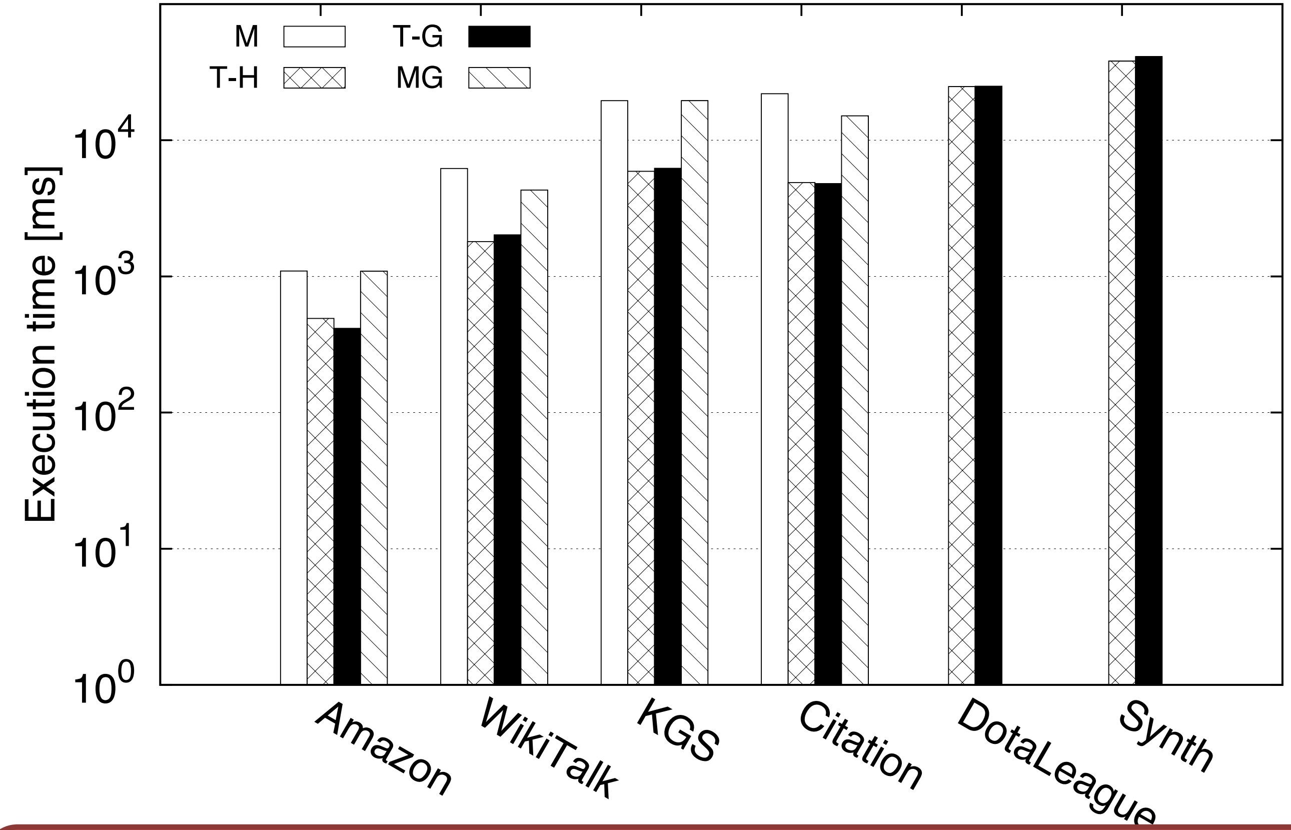
CSR-represented graphs fit in memory (GTX480)

BFS [algorithm]



Strong dependency on the graph.
Totem is the worst perfomer.
Medusa and MapGraph cannot handle large graphs.

BFS [full]



Totem becomes the best performer !

Lessons learned

- Brave attempts to enable the use of GPUs *inside* graph processing *systems*
- Every system has its own quirks
  - Lower level programming allows more optimizations, better performance.
  - Higher level APIs allow more productivity.
- Data pre-processing and data structure are crucial to both performance and capability.
- No clear winner, performance-wise.

HyGraph

Is there a case for heterogeneous computing in graph processing?

Conclusion

So how about Totem?

- The only heterogeneous graph processing system
  - Single node CPU+multi-GPU
  - Communication optimization
- What's "wrong"/missing ?
  - Static partitioning only
  - BSP model
  - It's not distributed
    - We fixed that*

*Yong Guo et. al, "Design and Experimental Evaluation of Distributed Heterogeneous Graph-Processing Systems", CCGrid 2015

Challenges for heterogeneous GP

- Granularity mismatch
  - The CPU requires coarse granularity (i.e., larger jobs),
  - The GPU requires fine granularity (i.e., many tiny jobs).
- Scheduling & load-balancing
  - Jobs need to be assigned to the CPU and/or the GPU.
- CPU-GPU Expensive Communication
  - CPU and GPU need to communicate to synchronize

An alternative: HyGraph*

- Simple vertex-centric API
  - Code is generated for CPU (OpenMP) and GPU (CUDA)
- Data is replicated on all devices
  - Largest graph in our experiments: 0.24GB of memory
- The graph is split into blocks** (groups of vertices)
  - CPU: one block per thread
  - GPU: one block per SM

* S.Heldens et al, "HyGraph: Fast Graph Processing on Hybrid CPU-GPU Platforms by Adaptive Load-Balancing" (in submission)

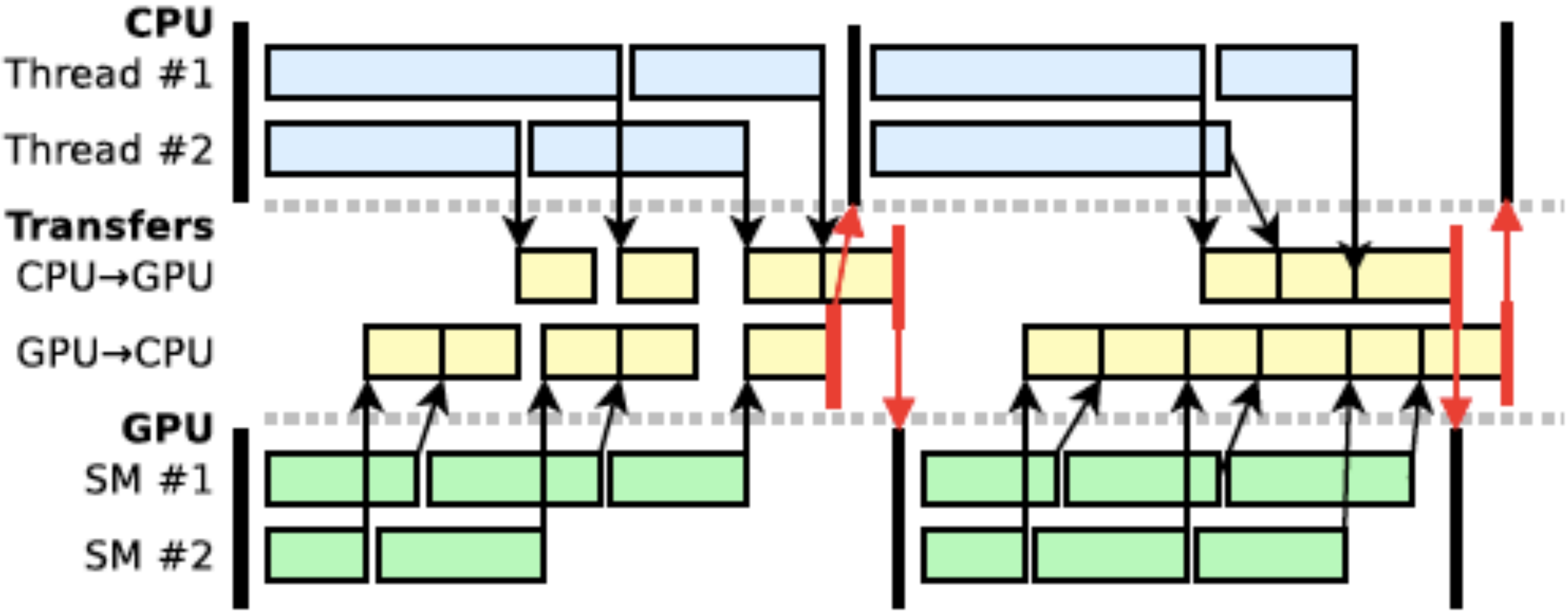
** Similar to shards in G-shards in CuSha and matrix rows GraphMat

HyGraph key points

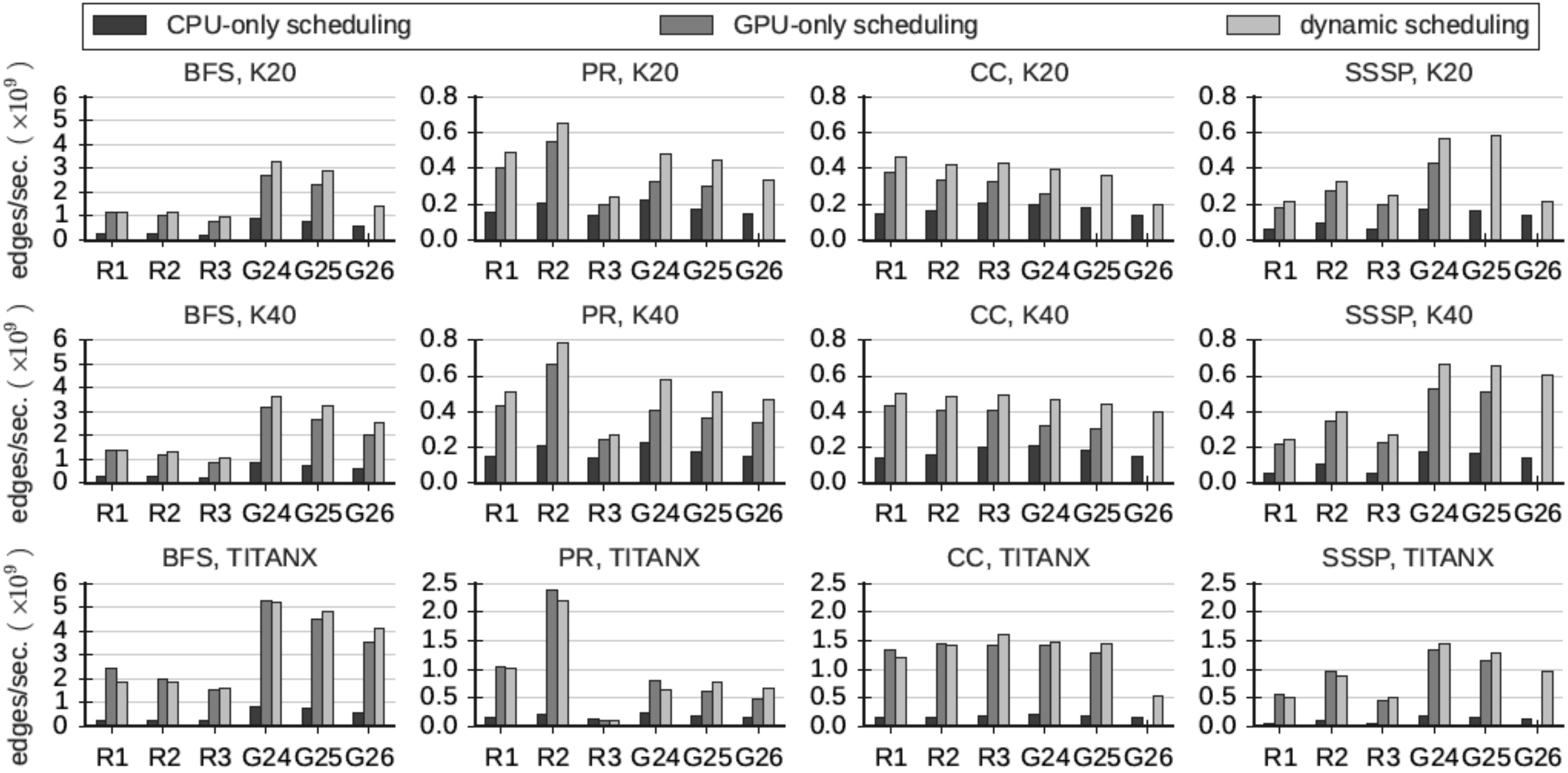
- Pre-processing
  - Reorganizes the graph in a block-based structure
- Granularity
  - Different block sizes for CPU and GPU
- Scheduling
  - Cooperation between CPU and GPU only at block-level
- Communication-computation overlap
  - As soon as a block is finished, results are sent
    - We use CUDA streams and multi-job kernels

HyGraph CPU+GPU processing

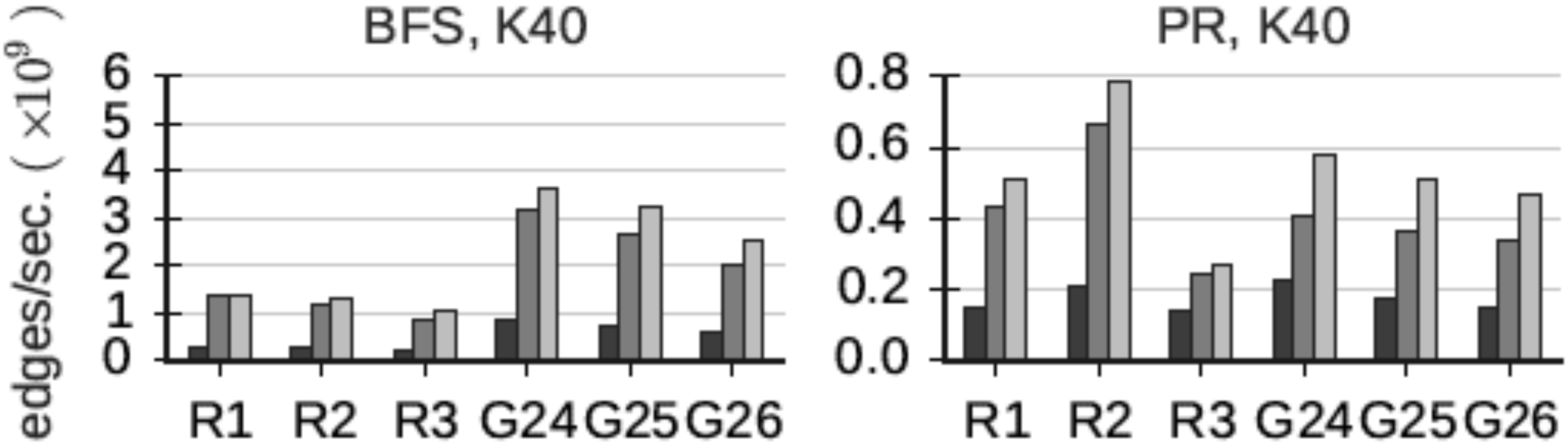
- Jobs dispatched on CPU and GPU



HyGraph results: performance



HyGraph results: performance

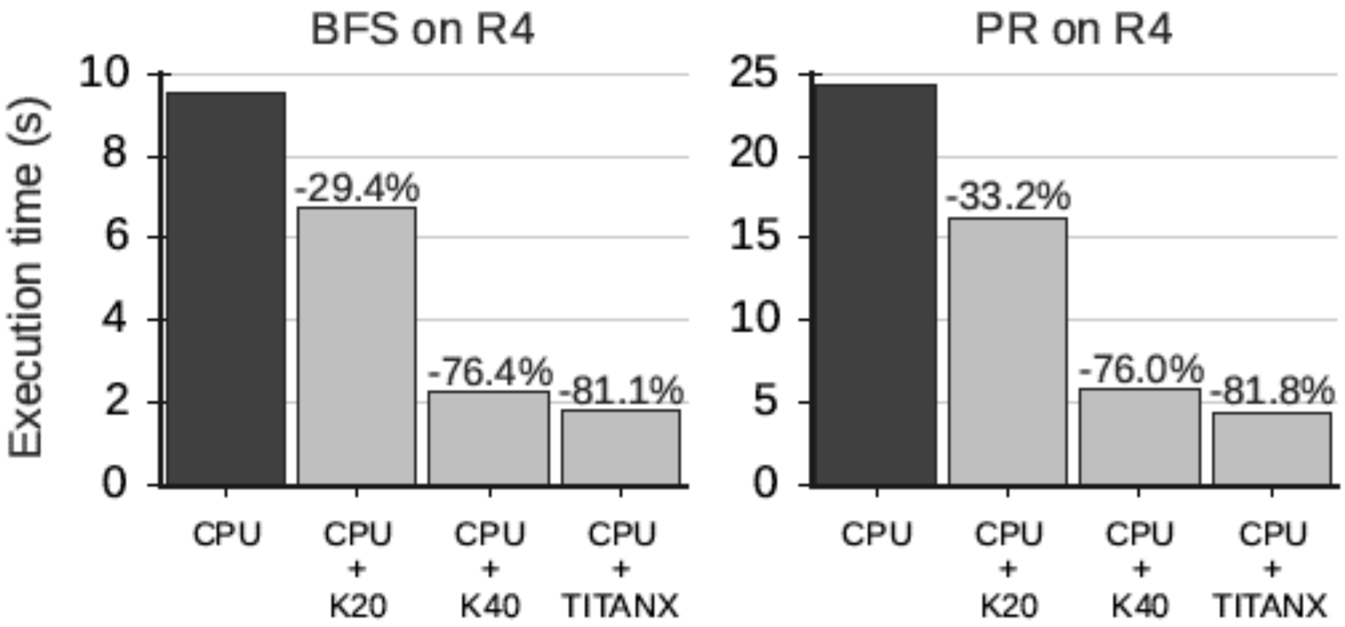


The GPU outperforms the CPU.
The hybrid performance improvement is between 3% and 37.3%

Dynamic scheduling adds little overhead, and outperforms static partitioning.

HyGraph results: size

- 1.8B edges graph
  - K20 : 32.7% , K40 : 79%, TITANX : 84.3%2



Lessons learned

- Hybrid graph processing possible
  - HyGraph provides this “for free”
  - Reasonable impact in performance (5-37%)
  - Significant impact as “extra-buffer” for GPU memory
- Performance gain and simplicity of design due to GPU improvements
- Graph ordering and block-size tuning are essential for performance
- Static partitioning is too general to fit iterative graph processing

Questions?

Agenda

- ~~Graphitti:~~

  ~~Investigating the performance factors in graph processing~~
- ~~HyGraph:~~

  ~~Yet another GPU-enabled system for graph processing~~
- Graphpedia:

  Are graphs really everywhere?
- Graphalytics:

  The Landscape of Graph Processing: a Quantitative View

Conclusions

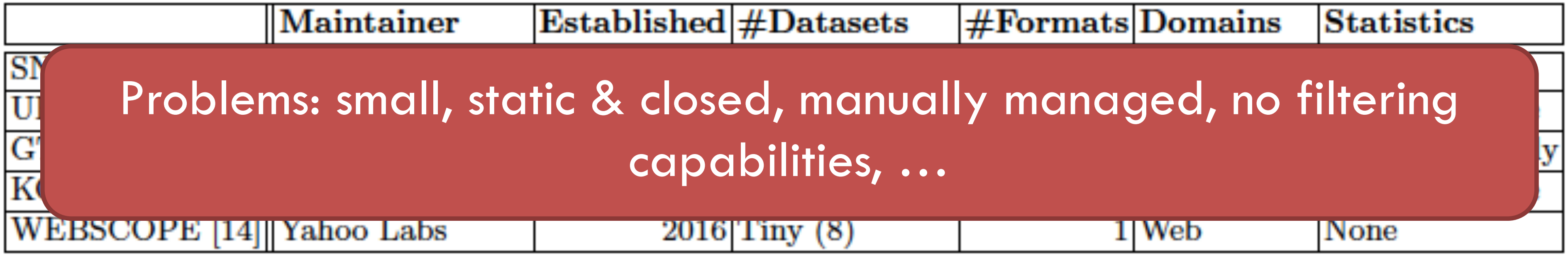
GraphPedia

Are graphs really everywhere?

Conclusion

Graphs, anyone?

- Two sources to obtain graphs
  - Public repositories



	Maintainer	Established	#Datasets	#Formats	Domains	Statistics
SN						
UI						
G						y
K						
WEBSCOPE [14]	Yahoo Labs	2016	Tiny (8)	1	Web	None

Problems: small, static & closed, manually managed, no filtering capabilities, …

- Synthetic generators (20+ in the last 20+ years)
  - Graph500

Problems: slow, no archiving, no fine-grain control of the generated graphs

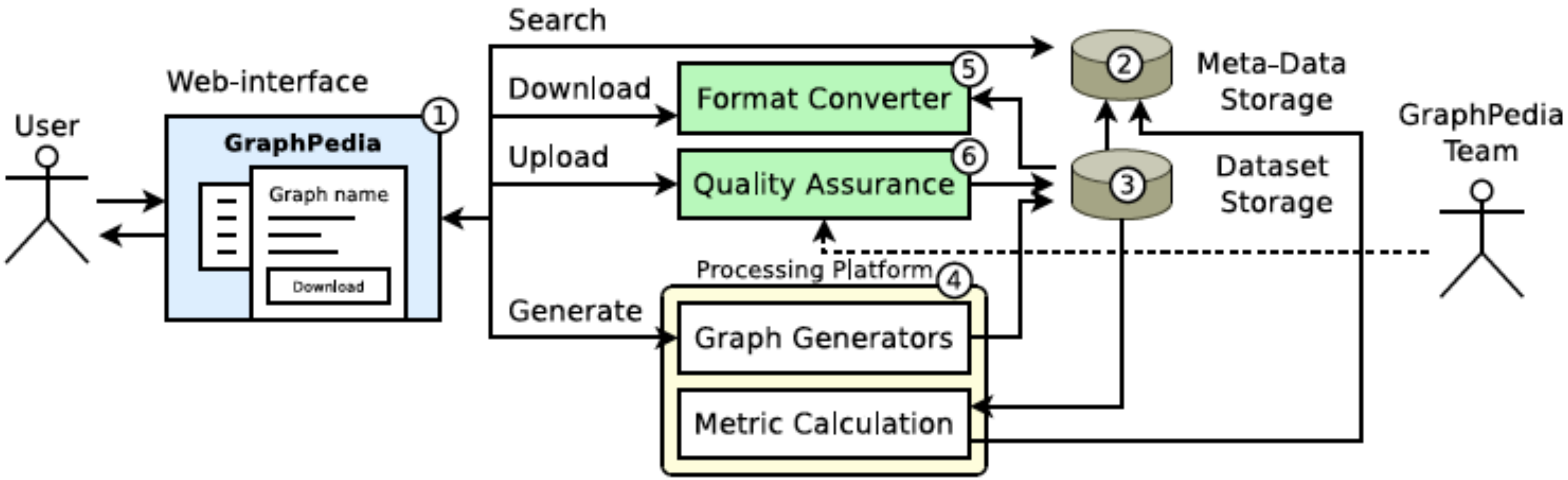
  - Internet Graph Generator

Next-Generation Network Archives*

- Improve Variety
  - Collect different graphs of different kinds from different domains
- Encourage Sharing.
  - Meeting space for researchers to exchange both knowledge and data.
- Enable Different Storage formats.
- Focus on Usability.
  - Allow users to browse and search the large collection of available datasets.
- Include synthetic datasets.
  - Provide access to synthetic datasets
  - Create datasets on-demand
- Provenance & impact.
  - Include datasets provenance
  - Measure & track impact

*S.Heldens et al., "Towards the Next Generation of Large-Scale Network Archives", PELGA'16

GraphPedia



Challenge #1: Efficiency

Challenge #2: Impact analysis

Challenge #3: Improving synthetic generation

Graphalytics

Conclusion

Graphalytics, in a nutshell

- An LDBC benchmark*
- Many classes of algorithms used in practice
- Diverse real and synthetic datasets
- Diverse set of experiments representative for practice
- Granula for manual choke-point analysis
- Modern software engineering practices
- Supports many platforms

http://graphalytics.ewi.tudelft.nl
https://github.com/tudelft-atlarge/graphalytics/

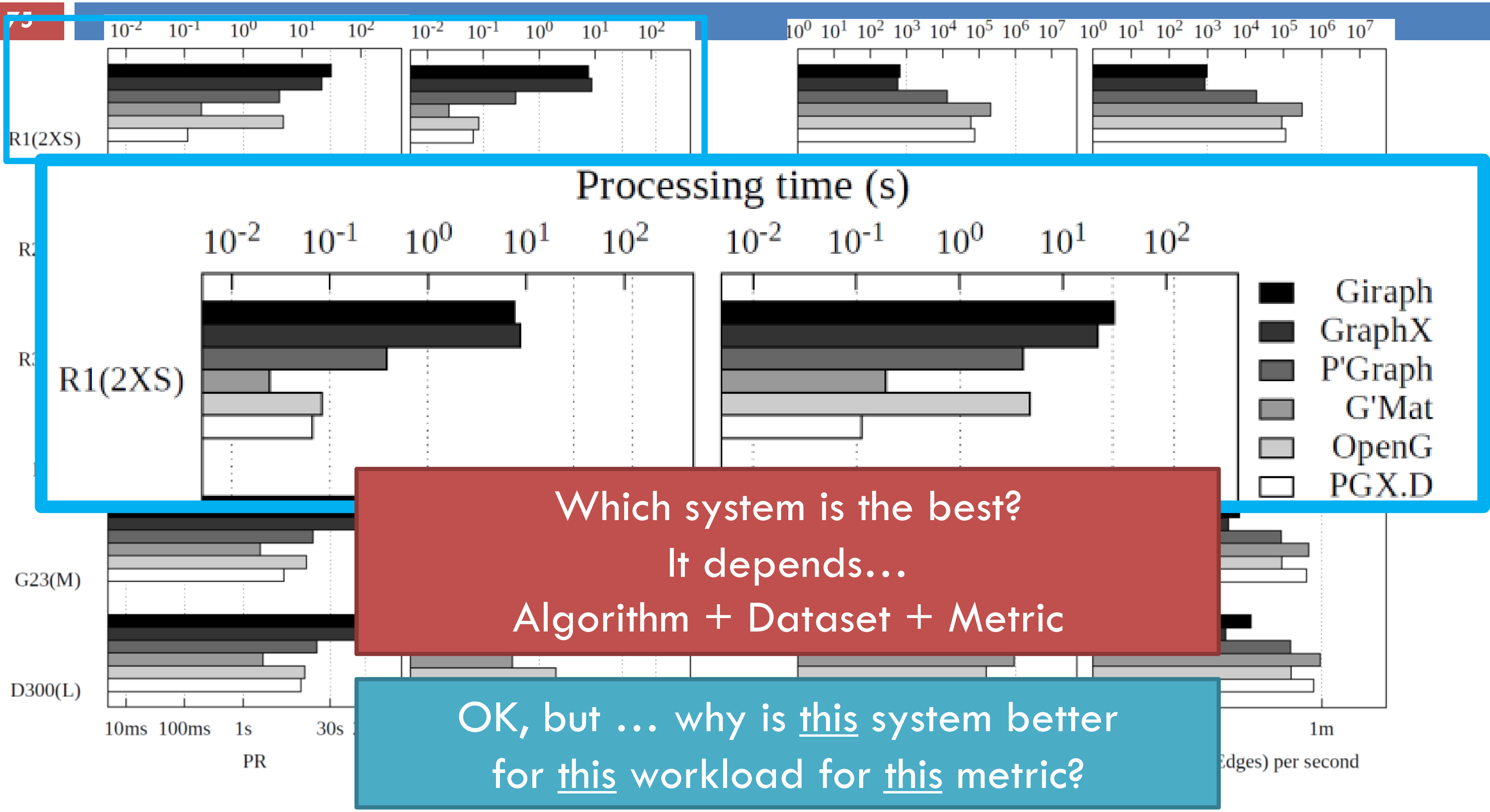
Implementation status

G=validated, on GitHub
V=validation stage

	MR2	Giraph	GraphX	PowerGraph	GraphLab	Neo4j	PGX.D	GraphMat	OpenG	TOTEM	MapGraph	Medusa
LCC	G	G	G	G	G	G	--	G	G	--	--	--
BFS	G	G	G	G	G	G	G	G	G	V	V	V
WCC	G	G	G	G	G	G	G	G	G	V	V	V
CDLP	G	G	G	G	G	G	G	G	G	--	--	--
P'Rank	--	G	G	G	V	--	G	G	G	V	V	V
SSSP	--	G	G	G	--	--	G	G	G	--	--	--

Benchmarking and tuning performed by vendors

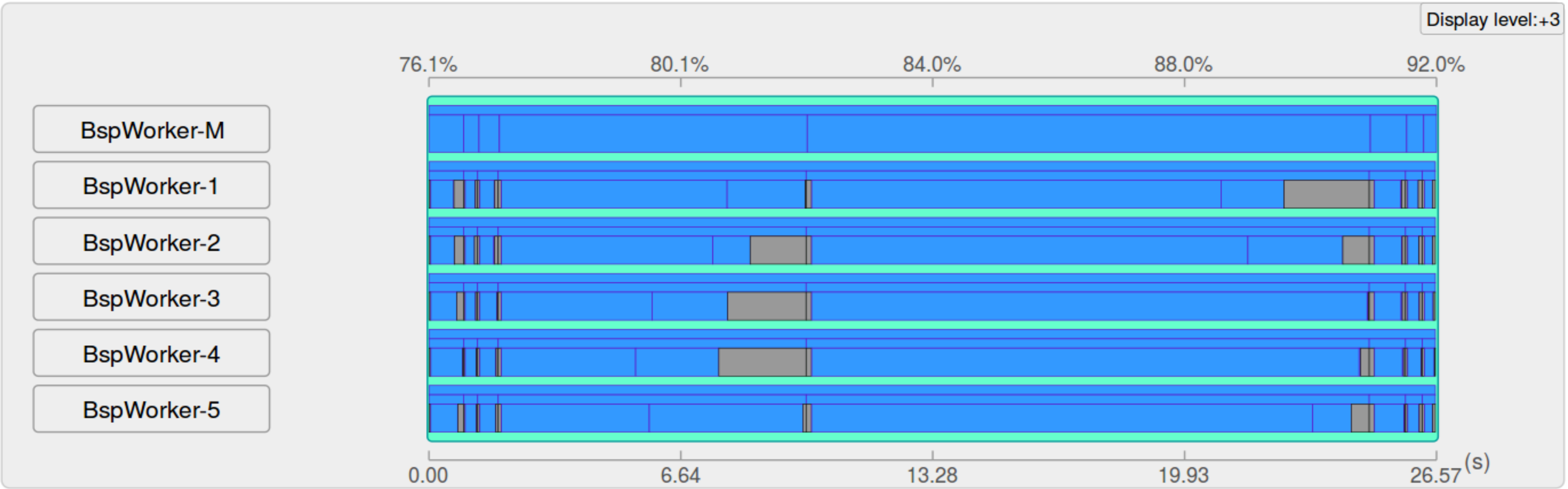
Processing time (s) + Edges[+Vertices]/s

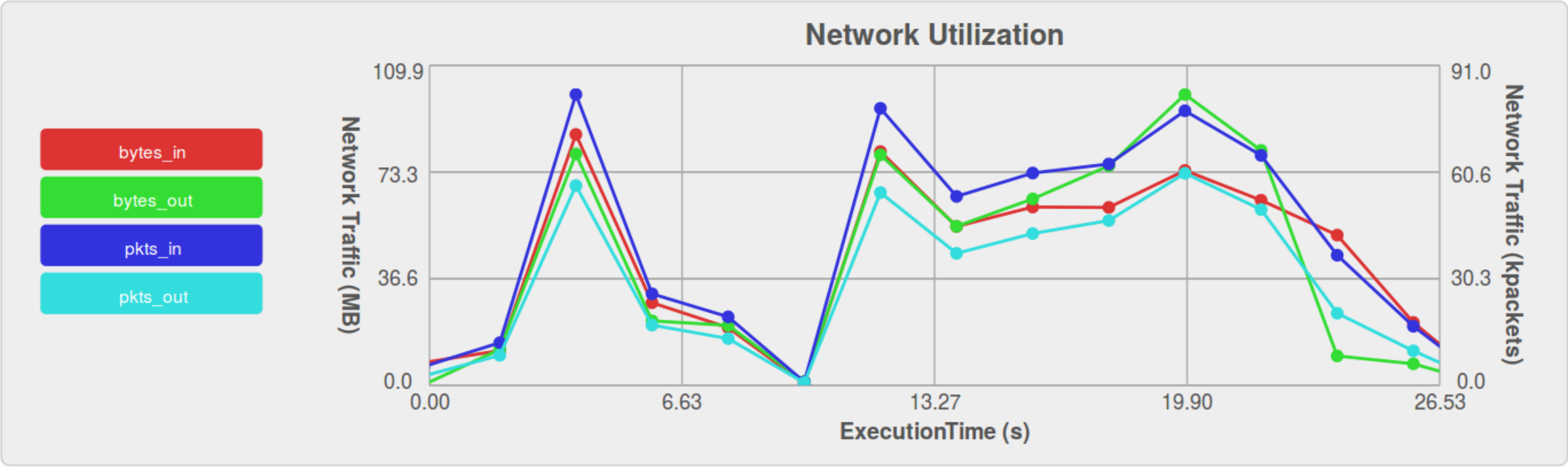


Which system is the best?
It depends…
Algorithm + Dataset + Metric

OK, but … why is this system better
for this workload for this metric?

Granula Visualizer (bottleneck analysis)





Summary

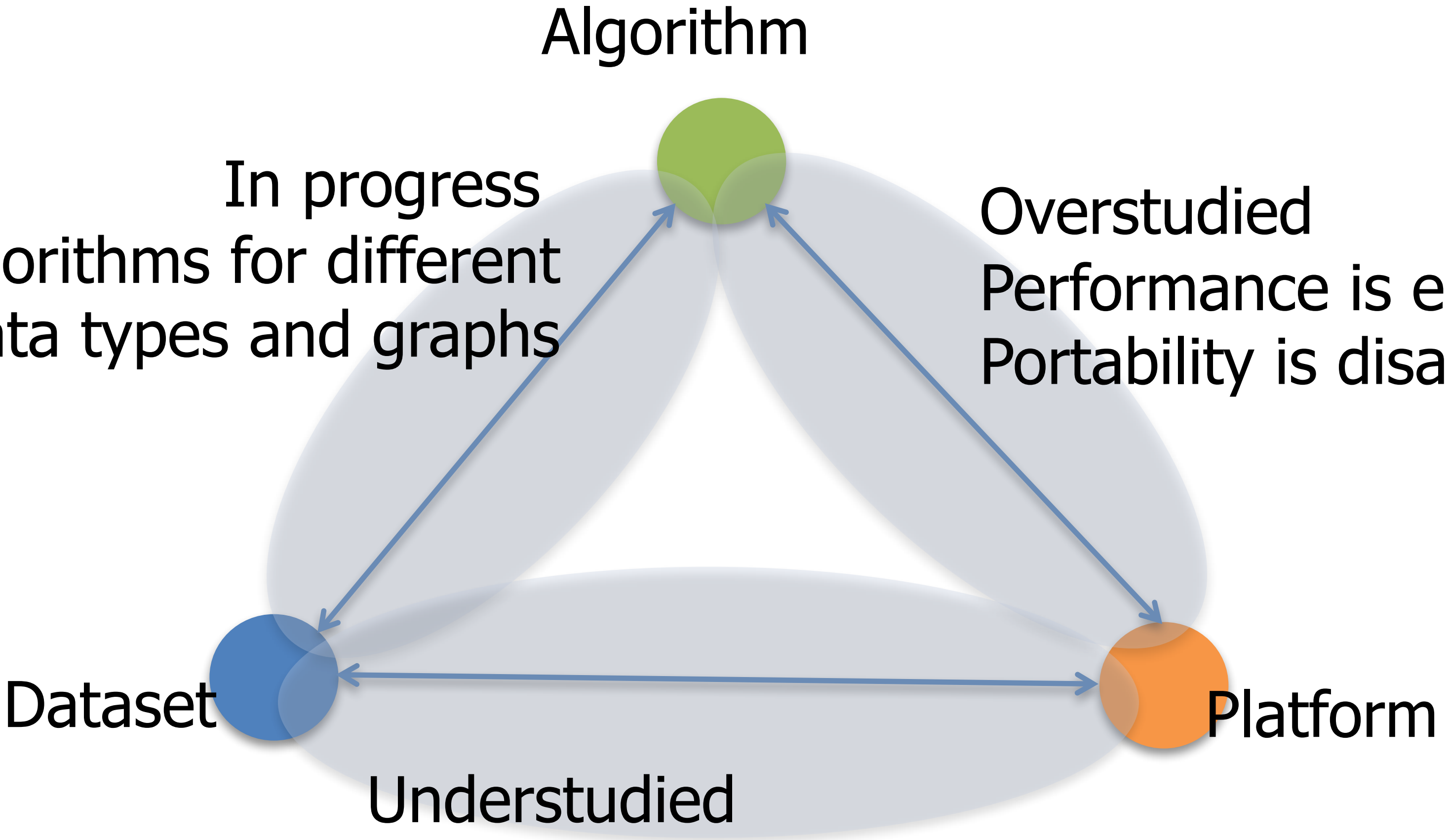
Take home message

- Graph processing *is* high performance computing
  - Due to/for data scale *and* analysis complexity
- HPC hardware is useful for graph processing
  - yet performance is (for now) unpredictable
- Performance is dependent on all three “axes”
  - Performance = *f* (dataset, algorithm, hardware)
    - Dataset = different graphs, different representation
    - Algorithm = variations of parallelization
    - Hardware = CPU/GPU/… ?

Take home message

- (Part of) What we do: graph processing + GPUs
  - Performance measurement
  - Graph processing benchmarking – **Graphalytics**
  - Performance analysis and prediction - **Graphitti**
  - Heterogeneous and Distributed system design – **HyGraph**
  - Next generation graph archives - **GraphPedia**

P-A-D triangle



Algorithm

In progress
Algorithms for different
data types and graphs

Overstudied
Performance is enabled
Portability is disabled

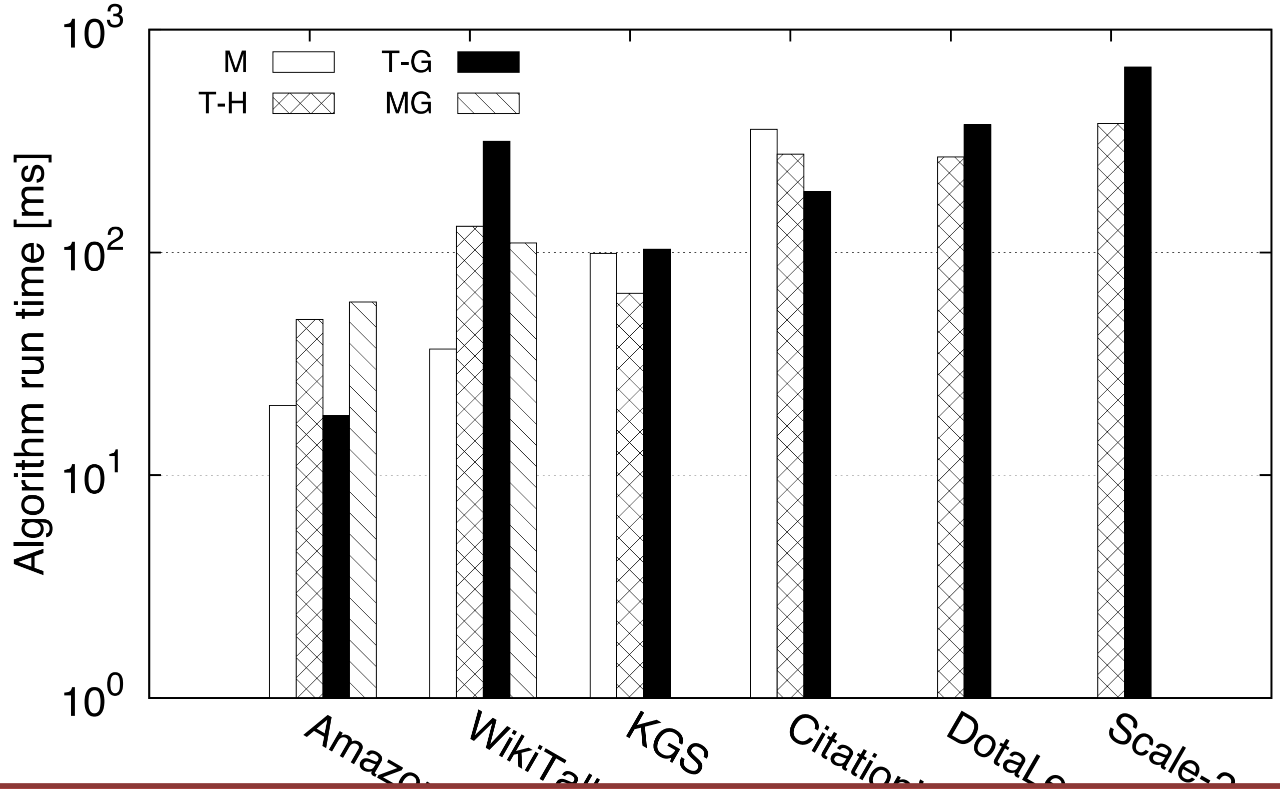
Dataset

Platform

Understudied
No systematic findings yet
Intuitive correlations
Must be correlated with the algorithm

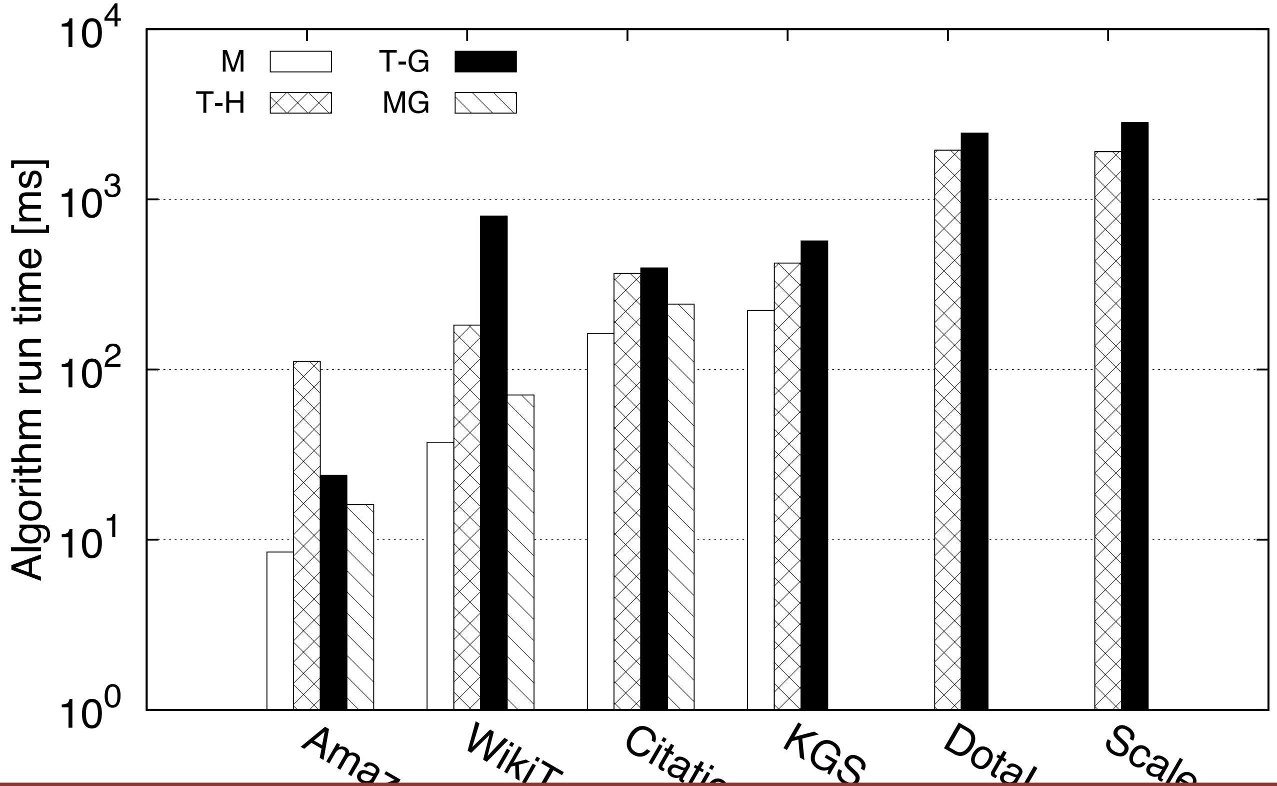
Backup slides

WCC [algorithm]



Strong dependency on the graph.
No best/worst performer.
More crashes of MapGraph.

PageRank [algorithm]

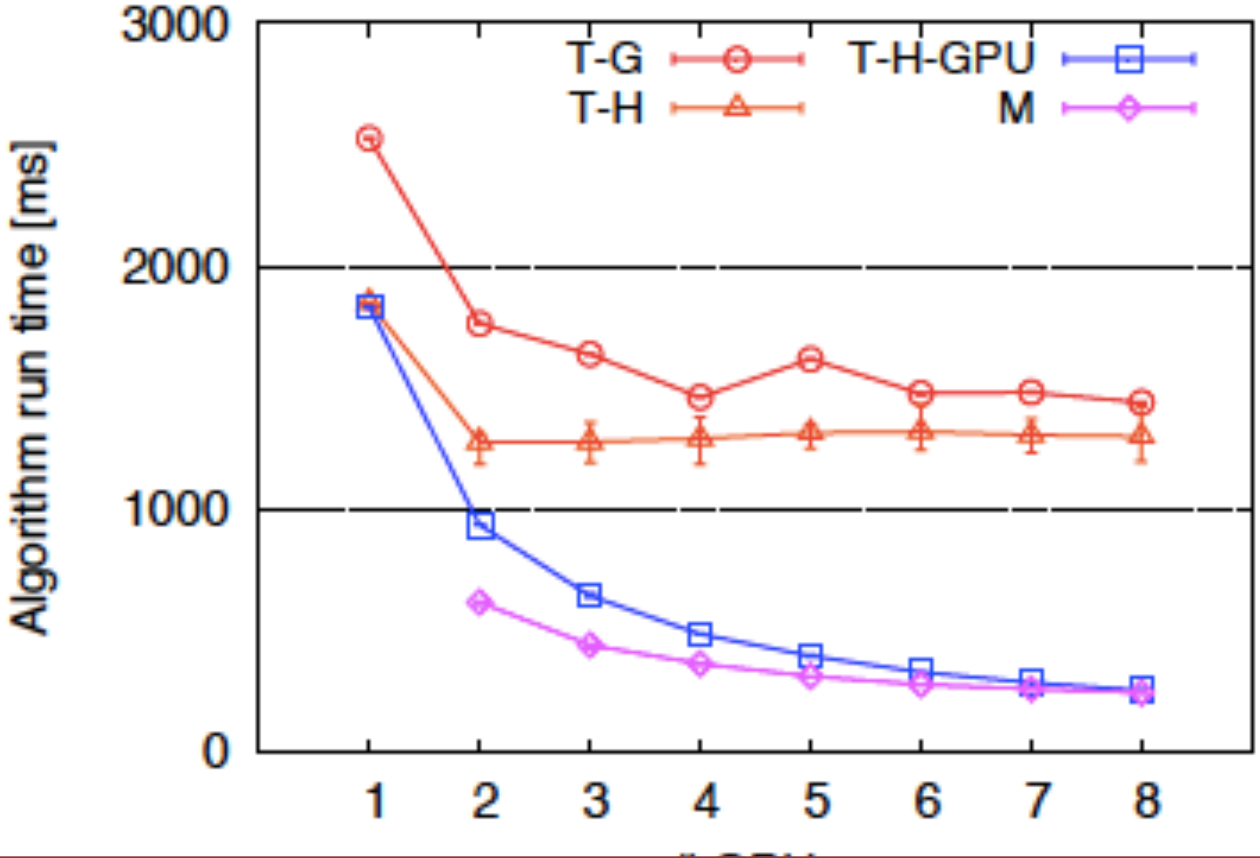


Most compute-intensive.

Totem performs worst.

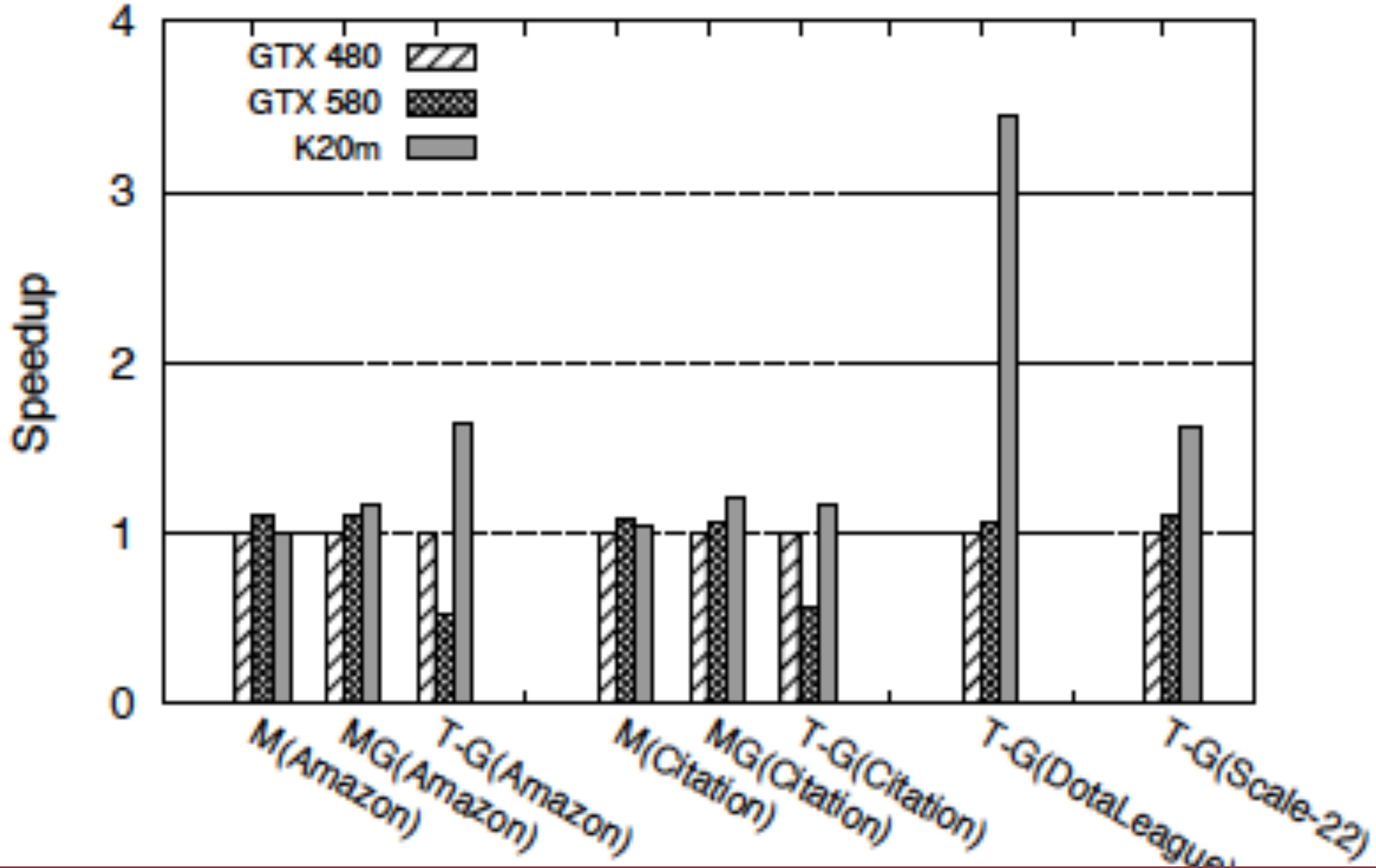
For large graphs, Totem-GPU is worse than hybrid.

Multi-GPU scalability



Platforms can use multiple GPUs efficiently.
Load balancing matters.

GPU versions



No guaranteed gain for newer GPUs

Larger graphs seem to benefit more from K20m.