

# THE LANDSCAPE OF LARGE SCALE GRAPH PROCESSING: A VIEW FROM HOLLAND

Ana Lucia Varbanescu, University of Amsterdam, The Netherlands

[a.l.varbanescu@uva.nl](mailto:a.l.varbanescu@uva.nl)

With data, work, and (some) slides from a whole team:  
Merijn Verstraaten, Ate Penders, Yong Guo, Alexandru Iosup, and others.

What to do when your graphs get out of control ?

# In this talk ...

- Graph Analytics = any form of graph processing
- Platform = hardware and/or software we can tune and change as a whole
- (Graph) Processing system = computing system that includes one or more platforms (for graph processing)

# Today's headlines

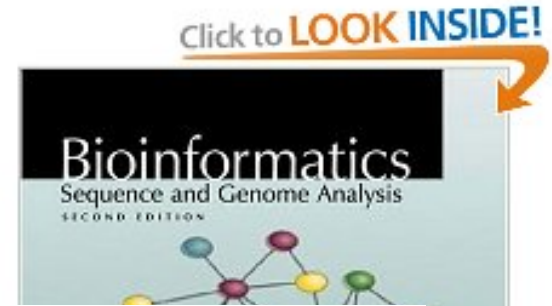
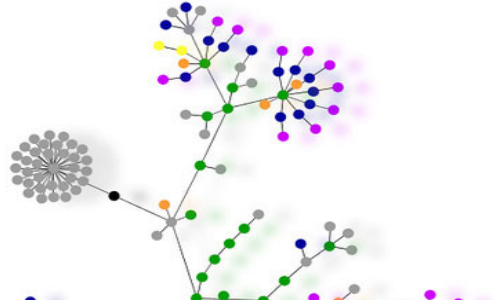
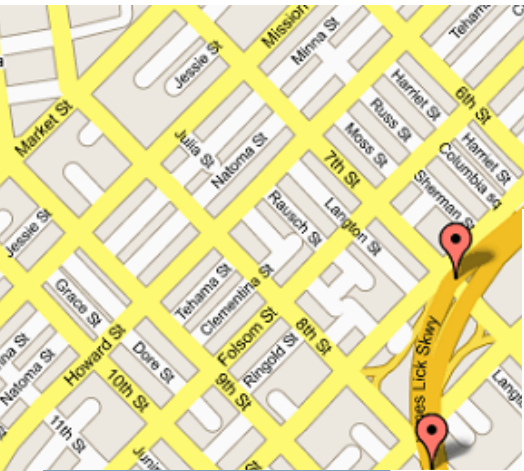


1. Graphs and graph processing
2. Benchmarking I: Algorithms
3. Benchmarking II: Platforms
4. Future research directions
5. Take home message

A horizontal decorative bar at the top of the slide, consisting of a red rectangular section on the left and a blue rectangular section on the right.

# Graphs and graph processing

# Graph analytics at work



Waa

Van Delft

Naar Van

Datum 30 -

Vertrek

+ Meer reiso

# Numbers ...

## Active Social Network Users – January 2014



Does your business still just focus only on LinkedIn?

@andyheadworth | [www.sironasays.com](http://www.sironasays.com)

# In April 2014 ...



The graphic features the LinkedIn logo (a blue square with 'in' in white) at the top center. Below it, the number '300' is written in large, bold, blue, 3D-style font. Underneath the number, the words 'MILLION MEMBERS' are written in a smaller, black, sans-serif font. The background is light gray with a pattern of white stars of various sizes, some of which are slightly faded. At the bottom, there is a paragraph of text in a black, sans-serif font.

**in**

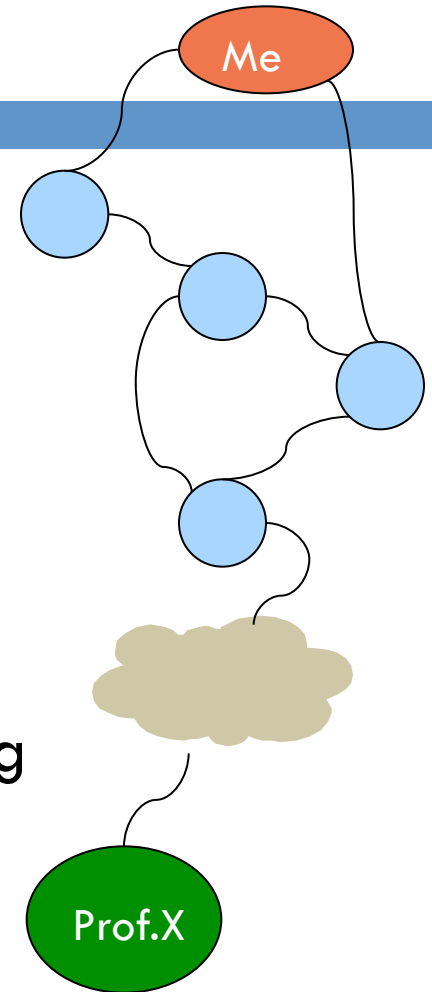
# 300

MILLION MEMBERS

We now have 300 million LinkedIn members, more than half of whom live outside of the U.S. That's enough to make LinkedIn the fourth largest country in the world. In celebration, we took a look back to see how much our membership has grown and diversified over the past five years. It's a helpful reminder of not only where we've been, but also where we're headed as we work to create economic opportunity for every professional in the world.

# Classical analytics

- Statistics
  - ▣ “How many connections do I have?”
- Traversing
  - ▣ “How can I reach Prof. X?”
- Querying
  - ▣ “Find all professionals in Graph Processing around Dresden.”
- Mining
  - ▣ “Find the most influential CS researcher in Amsterdam.”





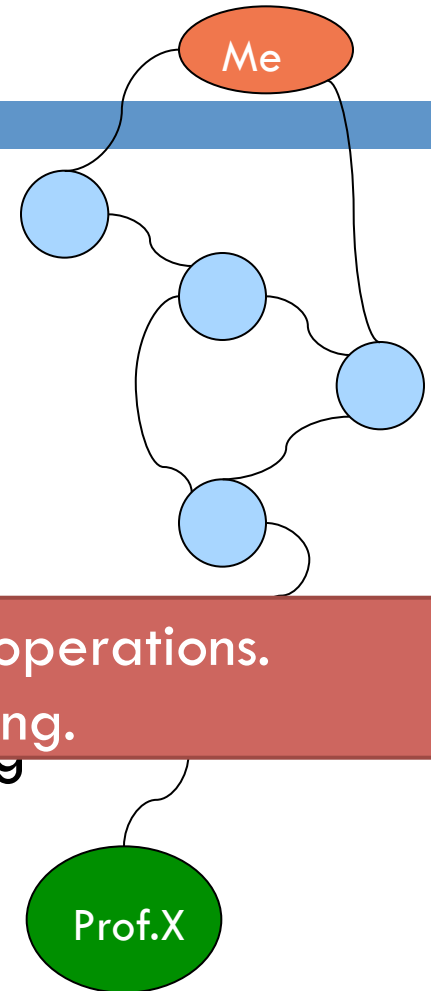
# Classical analytics

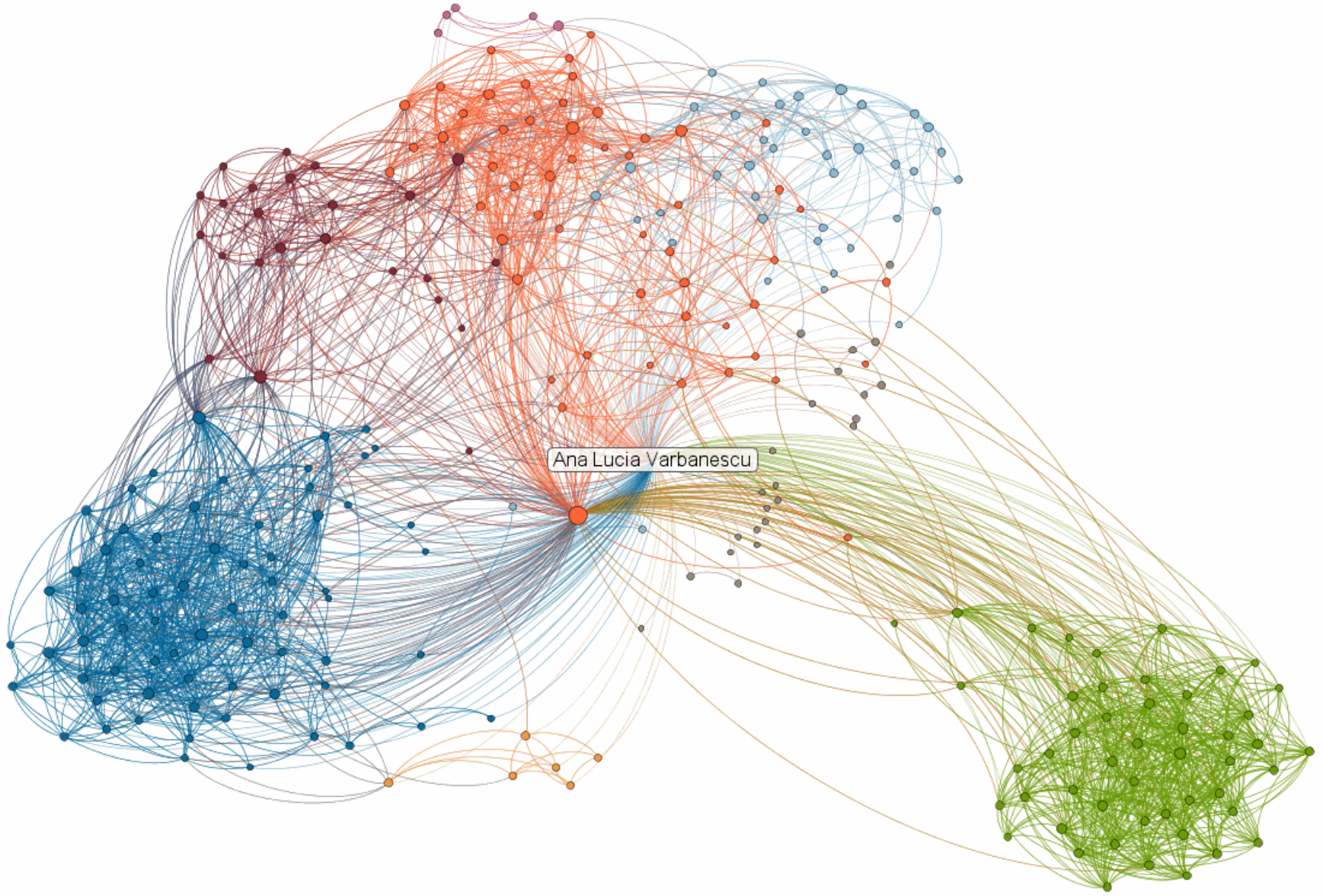
- Statistics
  - ▣ “How many connections do I have?”
- Traversing
  - ▣ “How can I reach Prof. X?”

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

- ▣ Find all professionals in Graph Processing around Dresden.“

- Mining
  - ▣ “Find the most influential CS researcher in Amsterdam.”



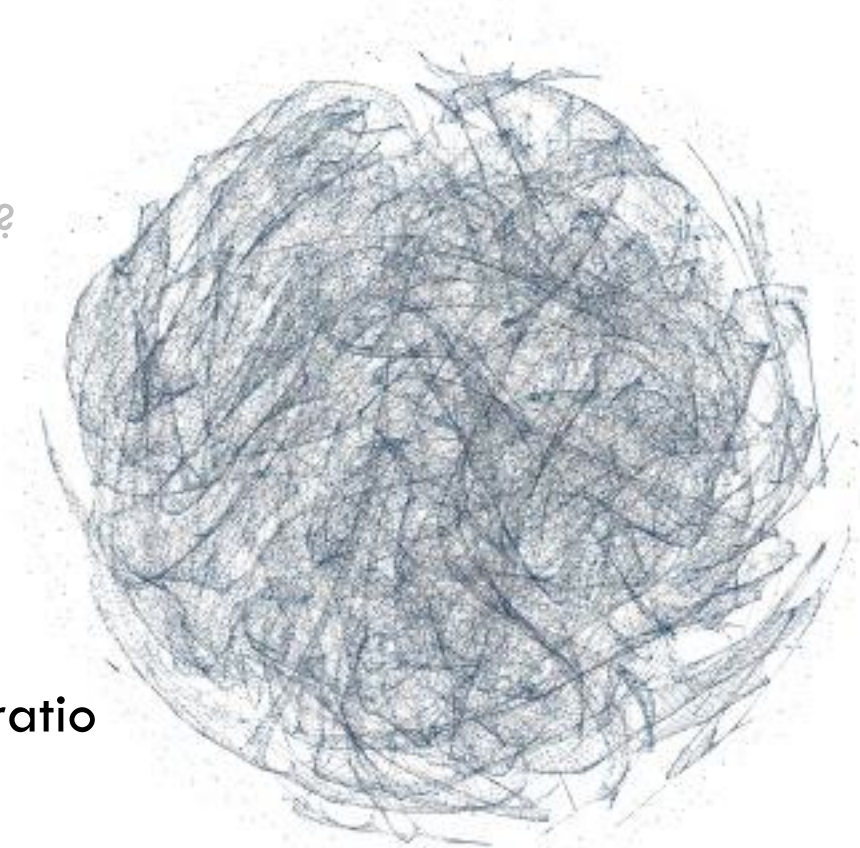


# Your network is so large...

Sorry, but your network is too large to be computed, we are working to increase the limit, stay tuned!

# 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

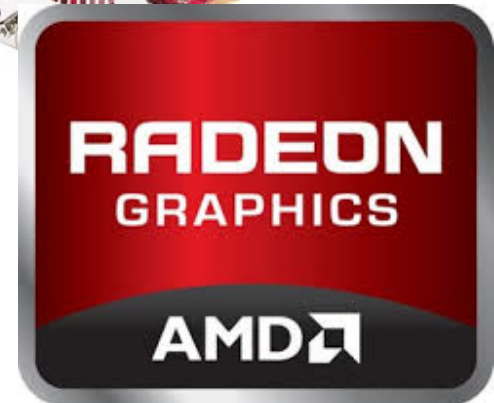


# Large Scale Graph Processing

- 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*.

# More performance? Many-cores!



# Top500 in November 2014

□ Traditional HPC is about computing ... not graphs!

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National University of Defense Technology China	<b>Tianhe-2 (MilkyWay-2)</b> - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel <b>Xeon Phi 31S1P</b> NUDT	3120000	33862.7	54902.4	17808
		<b>195 cores/node!</b>			<b>Accelerated!</b>	
2	DOE/SC/Oak Ridge National Laboratory United States	<b>Titan</b> - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, <b>NVIDIA K20x</b> Cray Inc.	560640	17590.0	27112.5	8209
					<b>Accelerated!</b>	
3	DOE/NNSA/LLNL United States	<b>Sequoia</b> - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	<b>K computer</b> , SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	<b>Mira</b> - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945

# Graph500 $\neq$ Top500 !

Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
1	K computer (Fujitsu - Custom supercomputer)	RIKEN, Japan	<b>65536</b>	524288	40	17977
2	DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	LLNL, USA	<b>65536</b>	1048576	40	16599
3	DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	ANL, USA	<b>49152</b>	786432	40	14328
4	JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	FZJ, Germany	<b>16384</b>	262144	38	5848
5	Fermi (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	CINECA	<b>8192</b>	131072	37	2567
6	Tianhe-2 (MilkyWay-2) (National University of Defense Technology - MPP)	Changsha, China	<b>8192</b>	196608	36	2061
7	Turing (IBM - BlueGene/Q, Power BQC 16C 1.60GHz)	Laboratory, UK	<b>4096</b>	65536	36	1427
8	Blue Joule (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	University of Edinburgh, UK	<b>4096</b>	65536	36	1427
9	DIRAC (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	EDF R&D	<b>4096</b>	65536	36	1427
10	Zumbrota (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)		<b>4096</b>	65536	36	1427

Number 1 in Top500



# The challenges

## □ **Feasibility:**

Can we use multi-core and many-core processors – to address the performance requirements for modern graph algorithms?

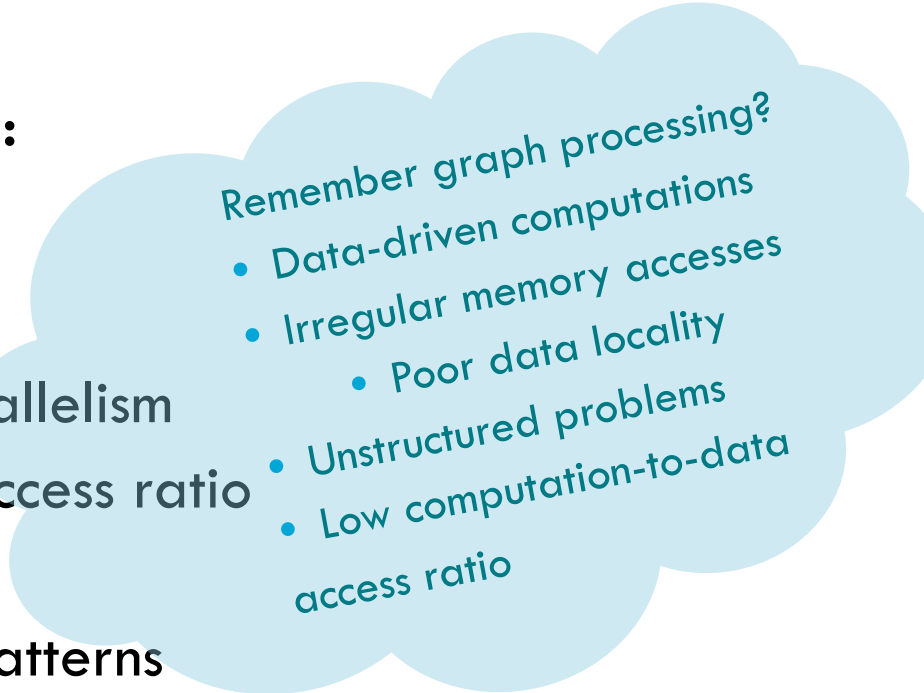
## □ **Usability:**

Is there a systematic solution to enable this match?



# Why challenging?

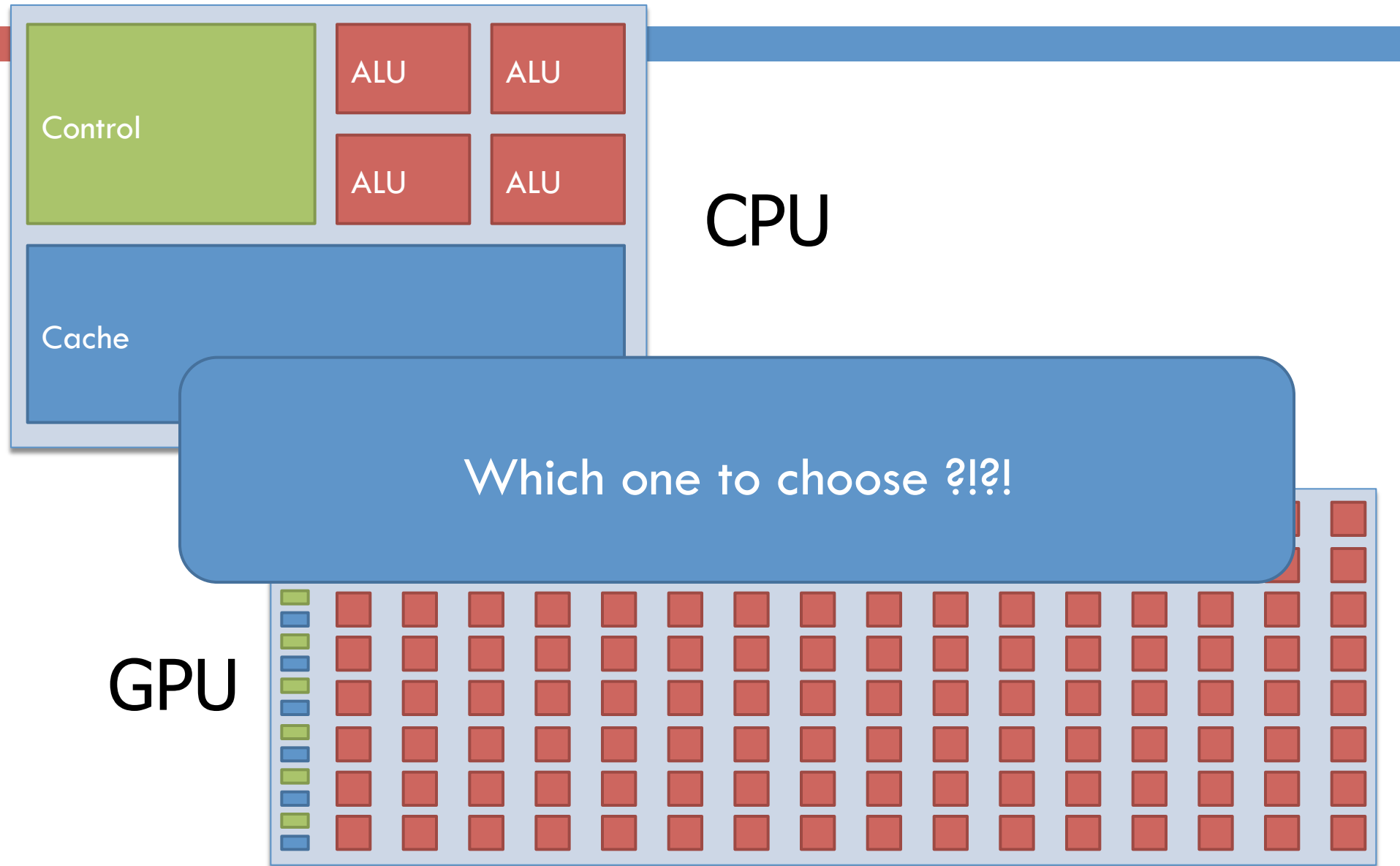
- Many-cores have emerged to improve performance by using massive parallelism.
  
- Performance gain in theory:  
N cores  $\Rightarrow$  N times faster
  
- For this, we need:
  - ▣ massive (multi-layered) parallelism
  - ▣ high computation-to-data access ratio
  - ▣ high data locality
  - ▣ structured, regular access patterns



Remember graph processing?

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

# Additional challenge

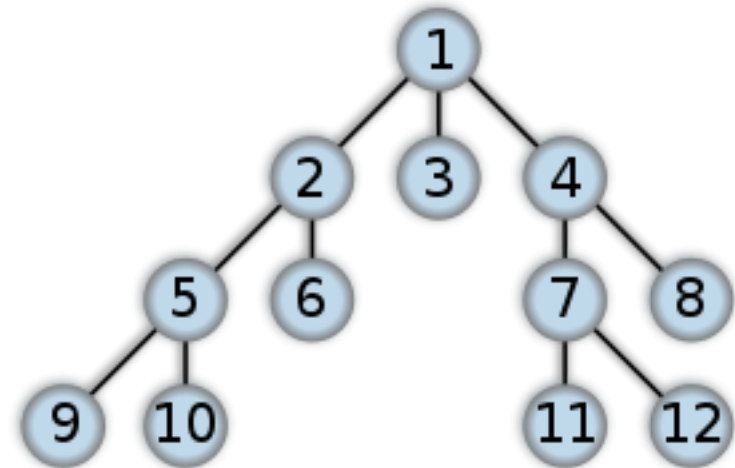


# Benchmarking I: Algorithms

Can we run graph analytics on HPC architectures, efficiently?

# 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



# 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

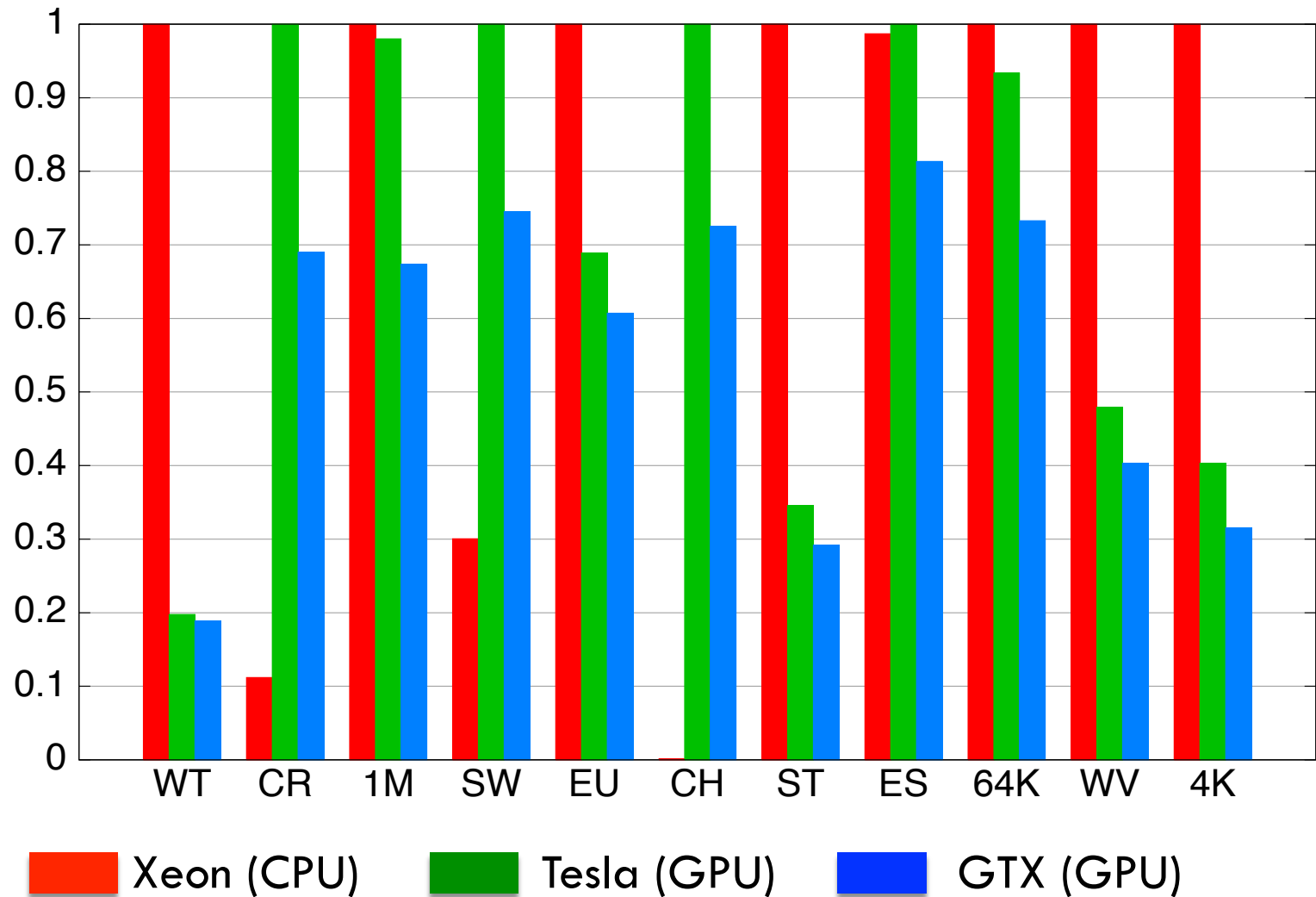
Intel(R) Xeon(R) CPU E5620 @ 2.40GHz

GeForce GTX 480

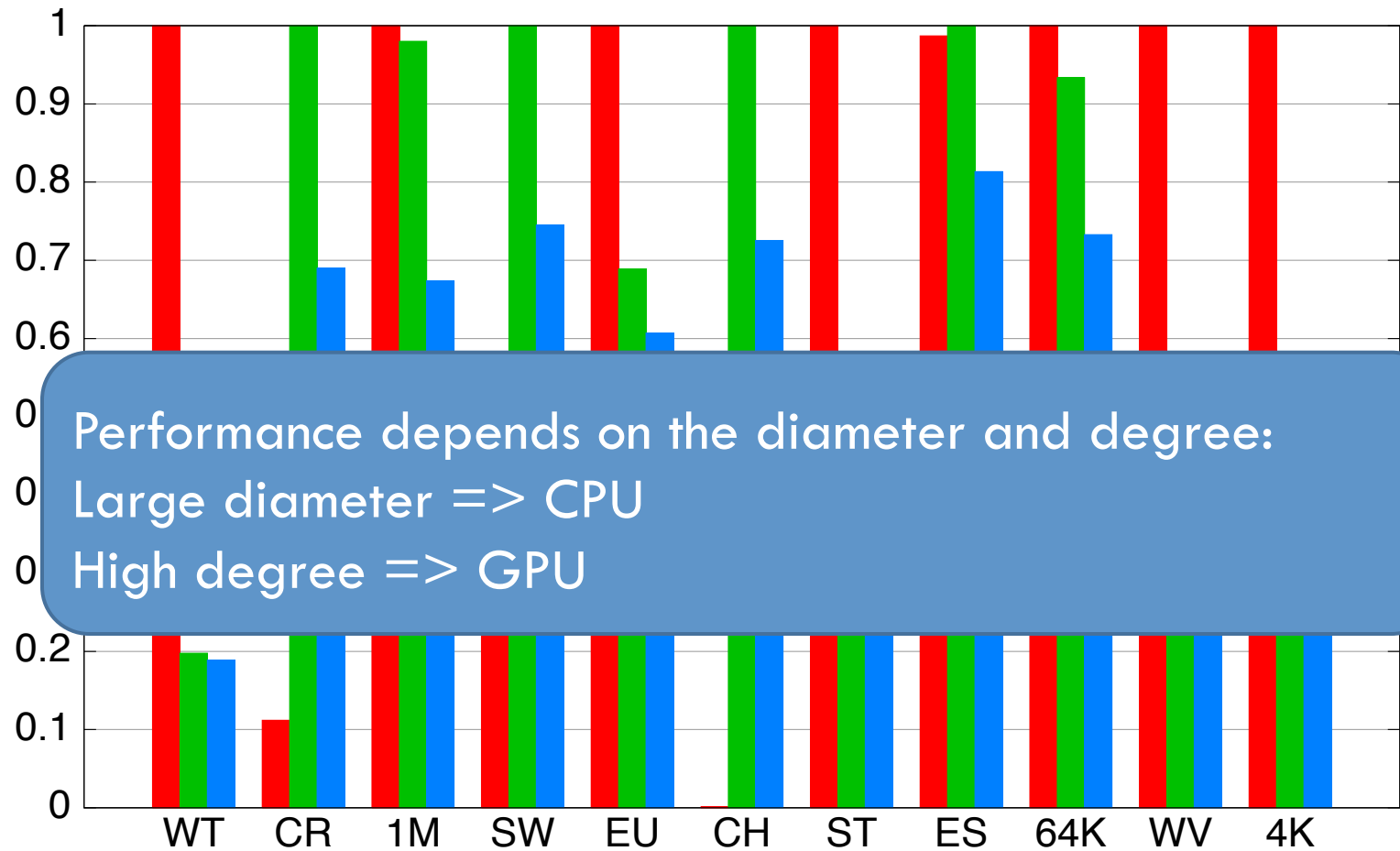
Tesla C2050 / C2070



# BFS – normalized



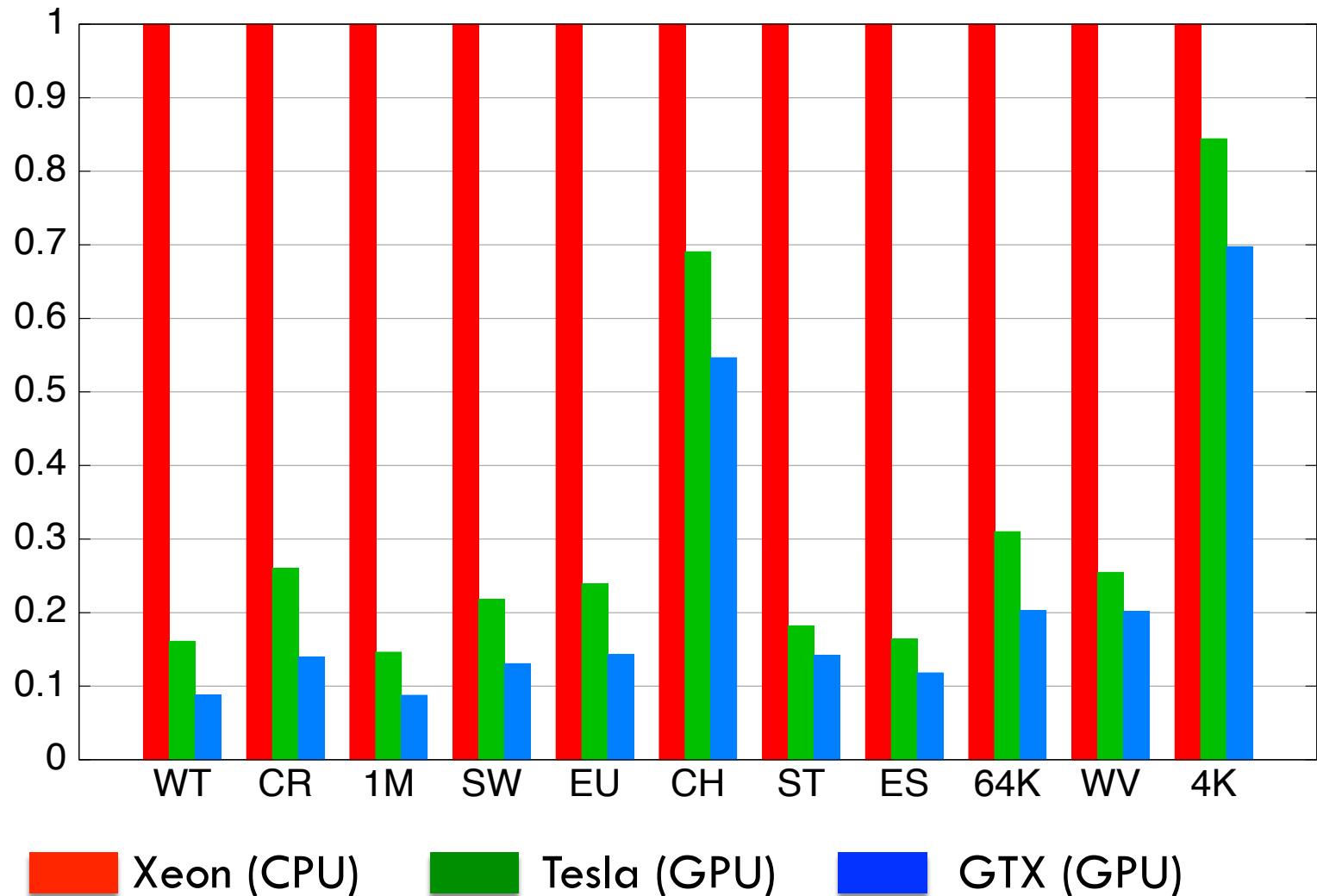
# BFS – normalized



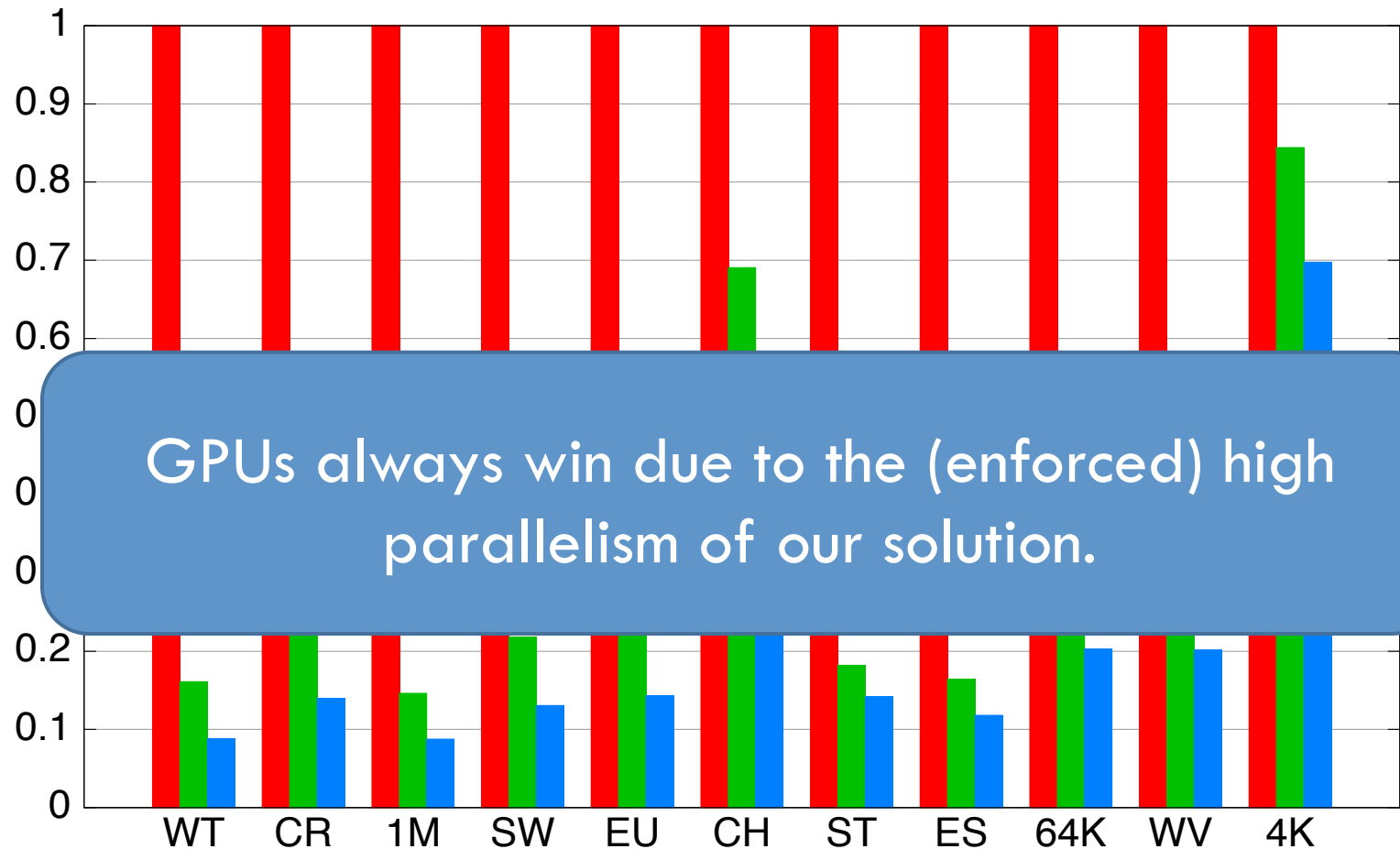
Xeon (CPU)    Tesla (GPU)    GTX (GPU)



# APSP - normalized



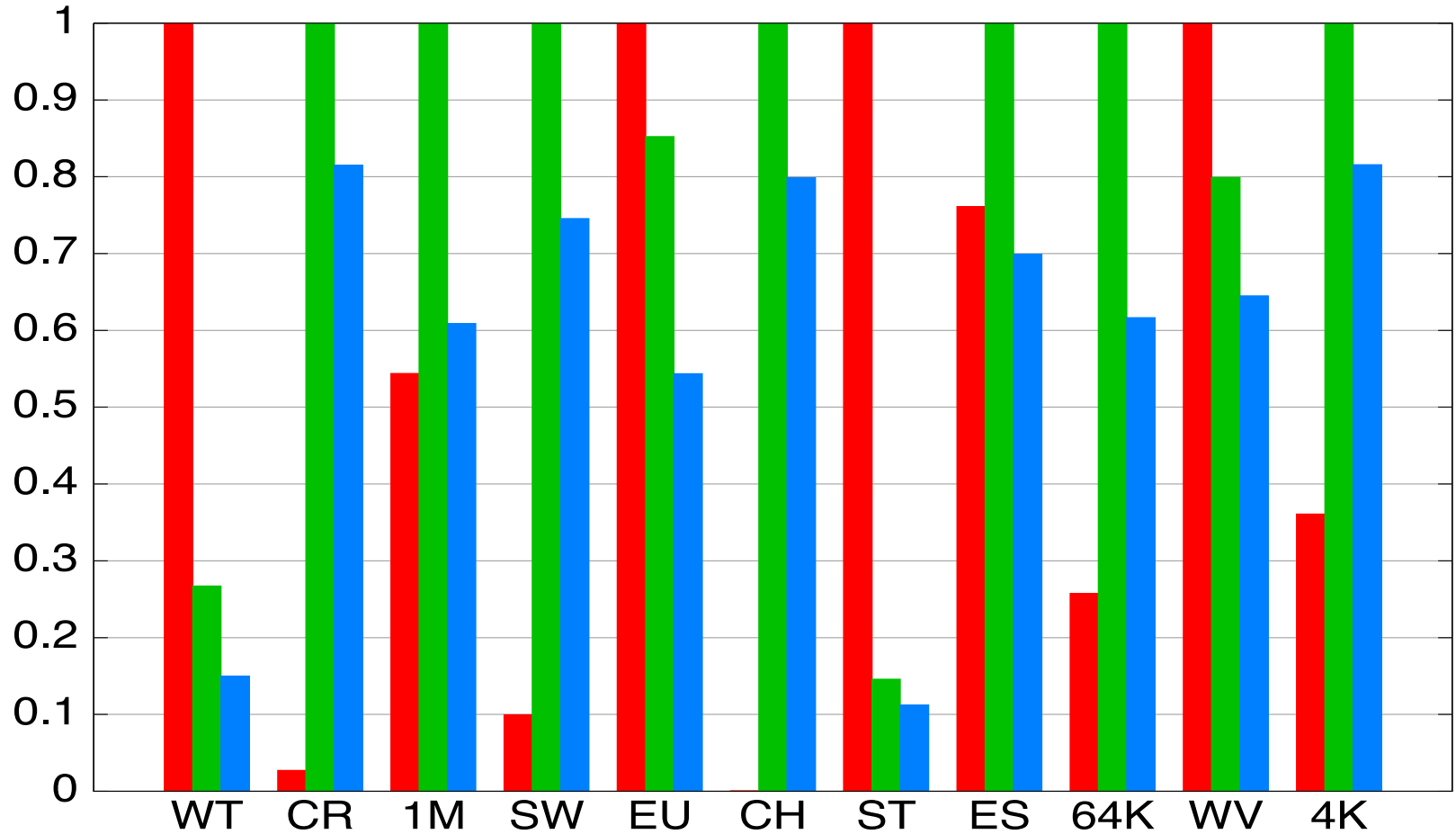
# APSP - normalized



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

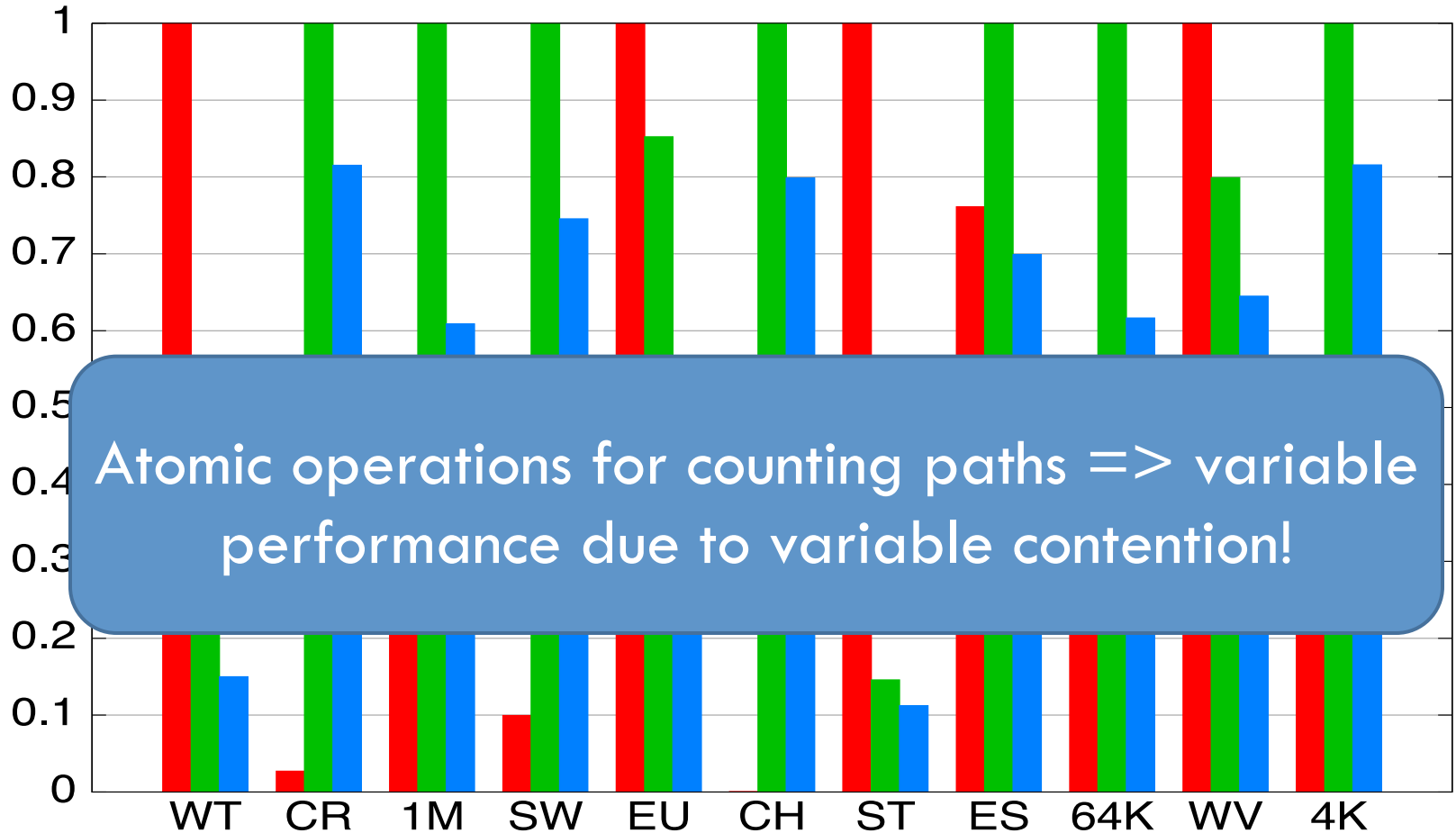
Xeon (CPU) Tesla (GPU) GTX (GPU)

# BC - normalized



Xeon (CPU) Tesla (GPU) GTX (GPU)

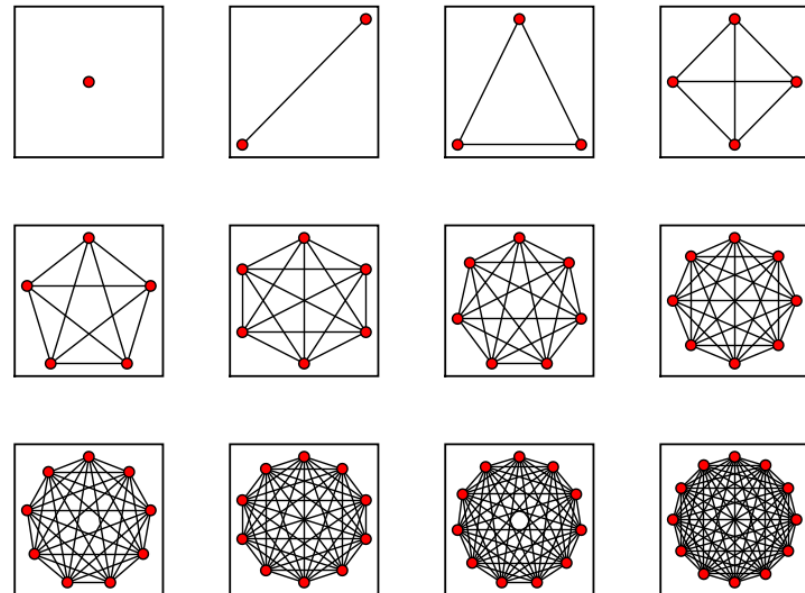
# BC - normalized



Xeon (CPU)    Tesla (GPU)    GTX (GPU)

# Lessons learned [1]

- Graphs seem to be CPU or GPU friendly
  - ▣ Data-dependent performance variations using the same implementation
    - CPU = lower parallelism, more caching
    - GPU = massive parallelism, less caching
  - ▣ Memory size is an issue!
    - ⇒ true large scale ?



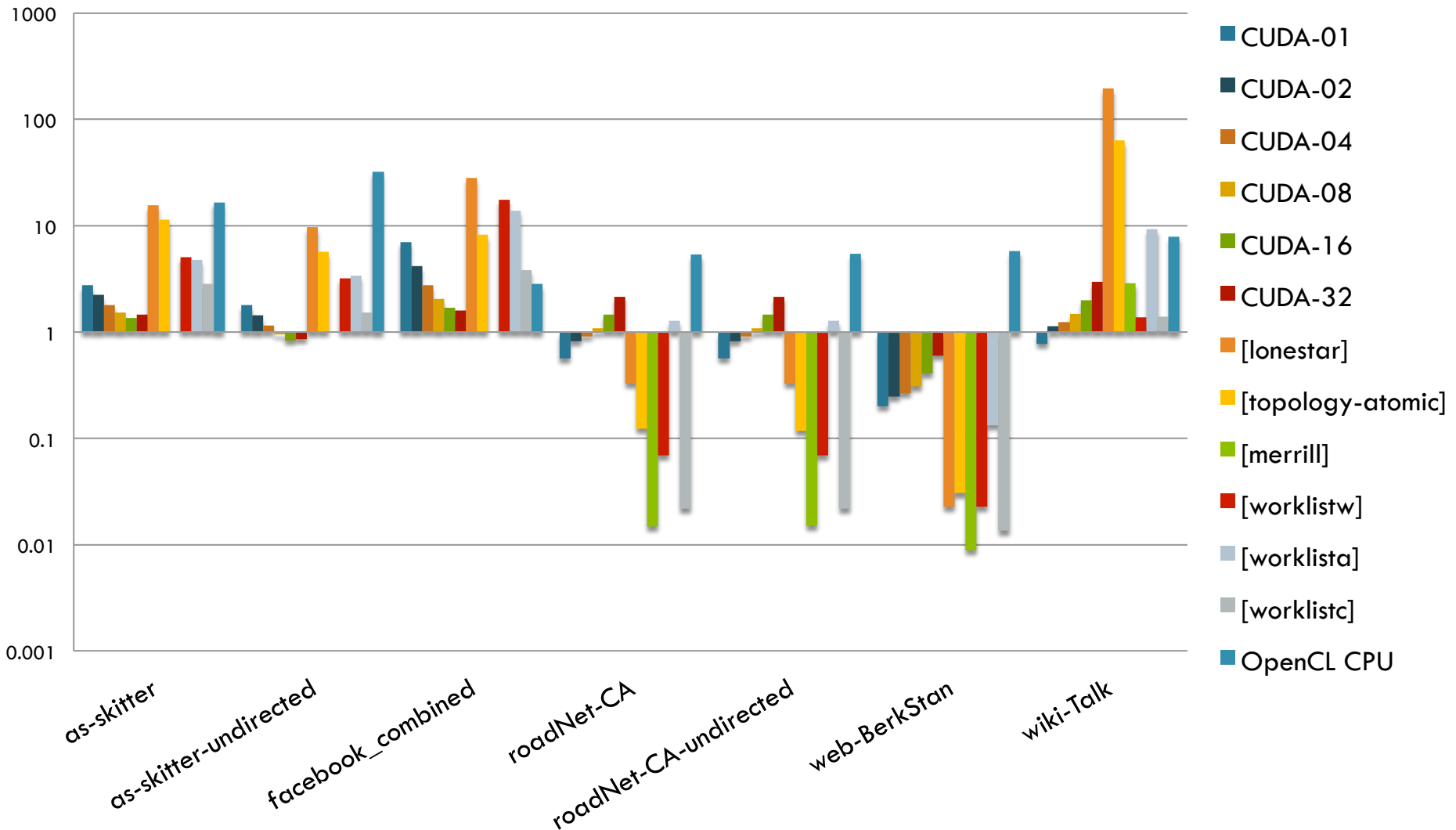
# Lessons learned [2]

- Increased algorithm complexity can increase parallelism
  - ▣ E.g.: ASPS =  $|V|$  x BFS
- Dataset representation and properties increase parallelism
- Synchronization is an important bottleneck
  - ▣ E.g.: BC mixes compute with synchronization
- We have no clear understanding of graph “sizes”
  - ▣ # vertices or # edges?
  - ▣ Diameter?
  - ▣ Other properties?

# Experiment 2: BFS traversals

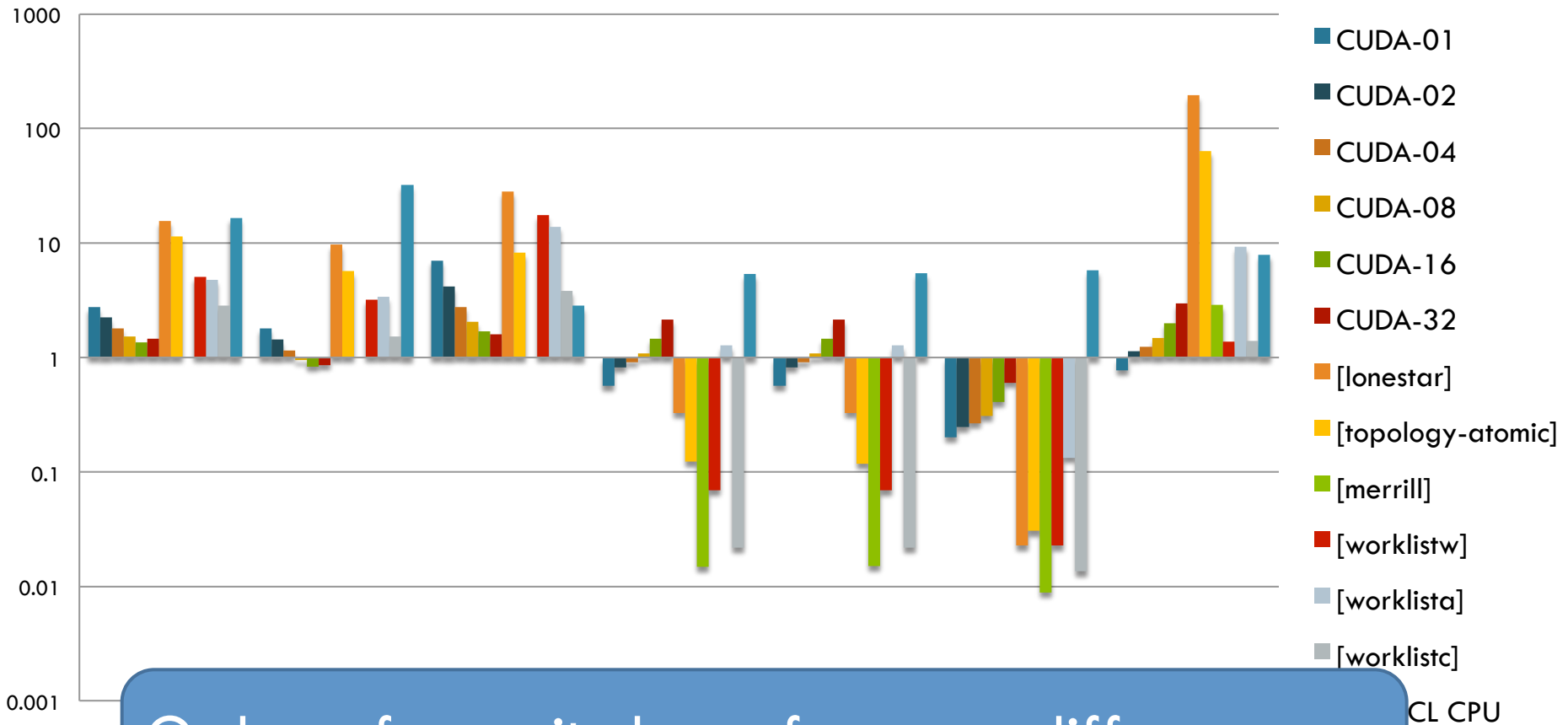
- Question:
  - ▣ Is there a **best** BFS algorithm?
    - On GPUs ?
    - Overall ?
- Setup:
  - ▣ Run multiple BFS implementations
    - Including the ones claimed to be the best @ LonestarGPU
  - ▣ Run on different graphs
    - 6 datasets
  - ▣ Run on different hardware

# Normalized on naïve GPU, kernel



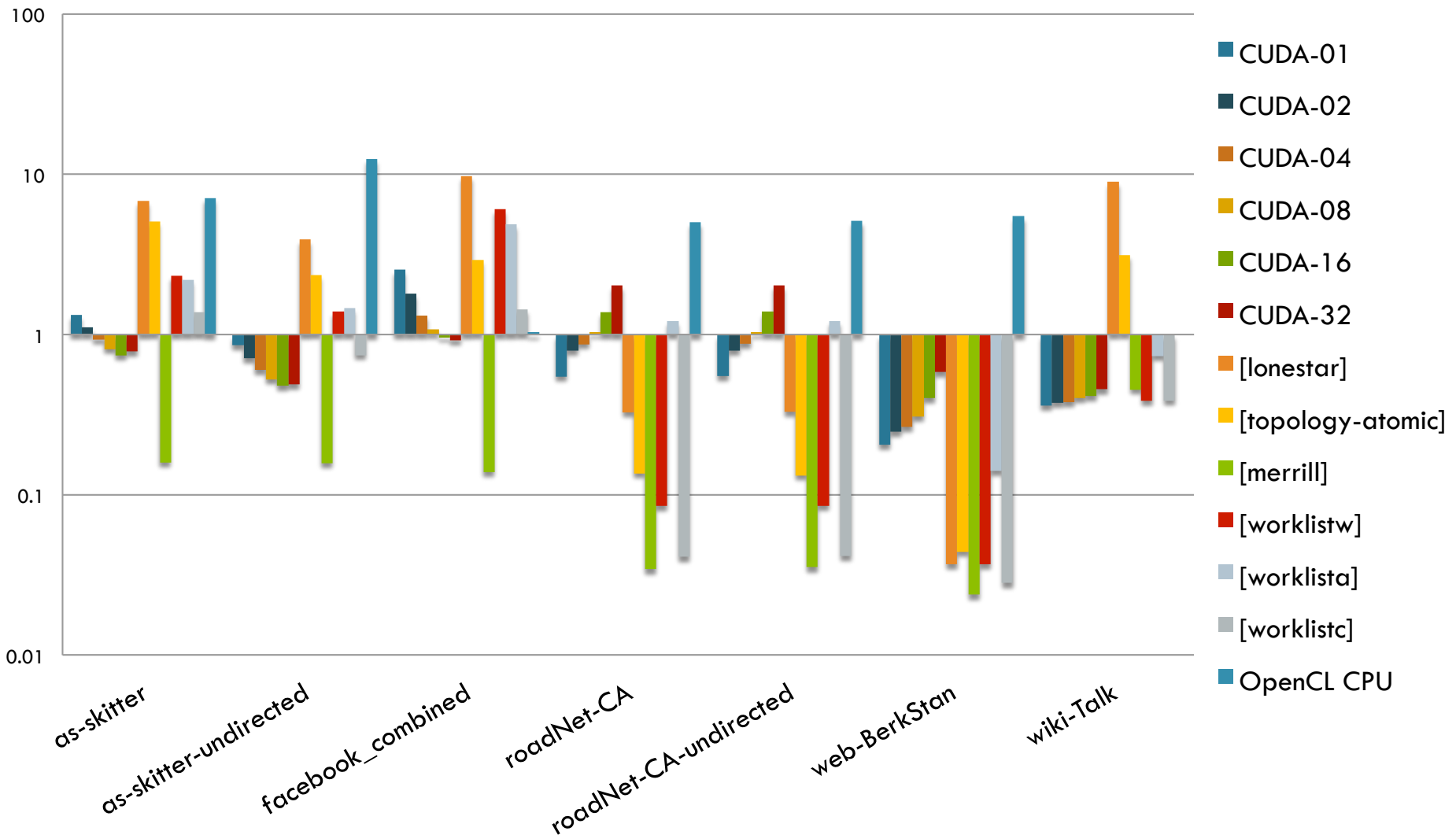


# Normalized on naïve GPU, kernel

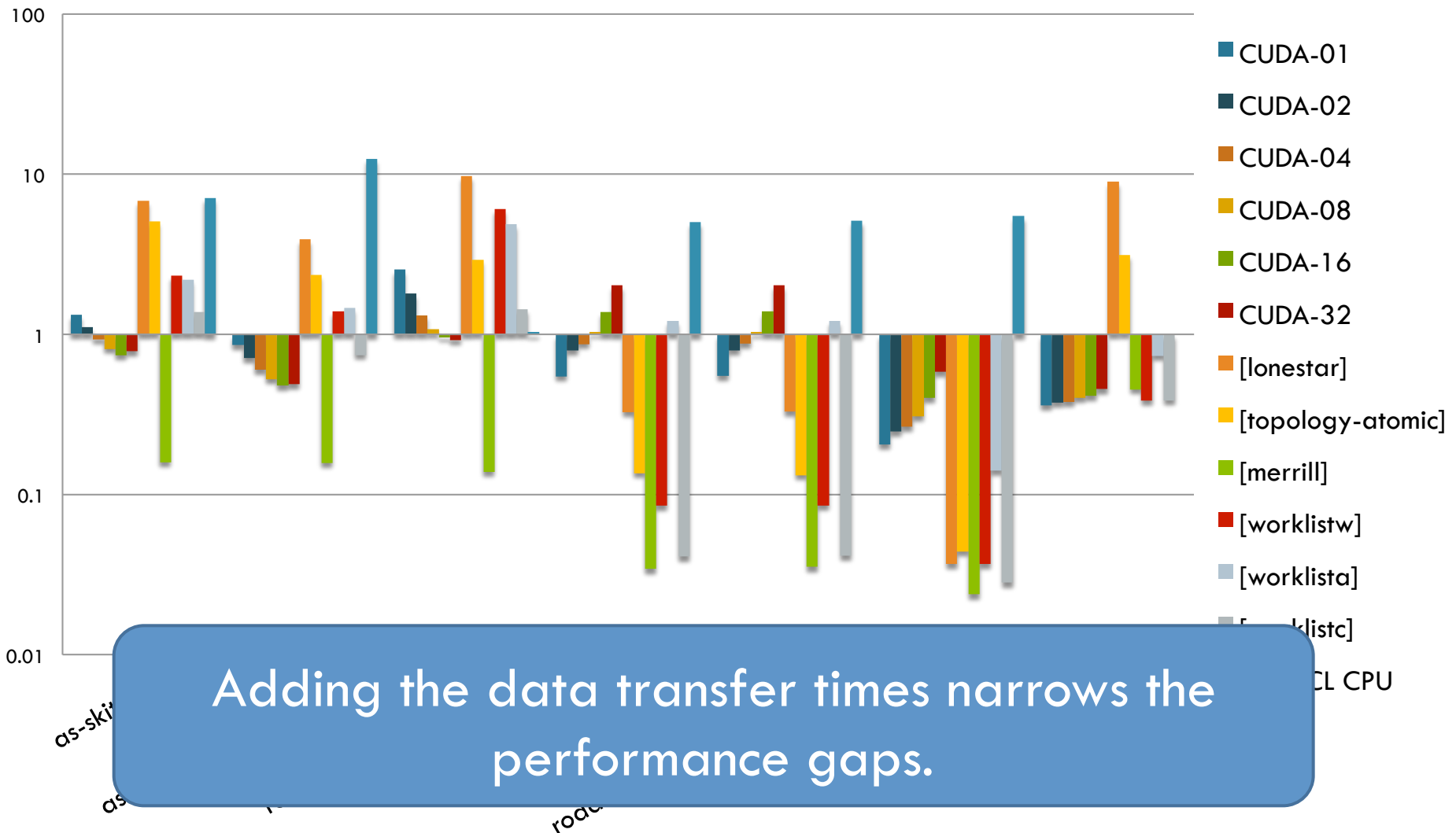


Orders of magnitude performance difference.  
No clear winner.

# Normalized on naïve GPU, full exec.



# Normalized on naïve GPU, full exec.



Adding the data transfer times narrows the performance gaps.

# Lessons learned [1]

- Large variability in performance depending on the graph.
  - ▣ Fastest to slowest ratio varies.
- The relative performance of one BFS implementation varies for different graphs.
  - ▣ Fastest on one graph CAN BE slowest on another graph.
- Data representation and data structures make the difference
  - ▣ List of edges vs adjacency lists
  - ▣ Lock-free frontier

# Lessons learned [2]

- Two disjointed classes of algorithms
  - ▣ GPU-optimized
  - ▣ Portable to CPU (= naïve)
- A naïve CPU implementation can be competitive with some of the GPU implementations.
  - ▣ On small graphs (GPUs are underutilized)
  - ▣ When data transfer is an issue.
- Better CPU solutions do exist ...



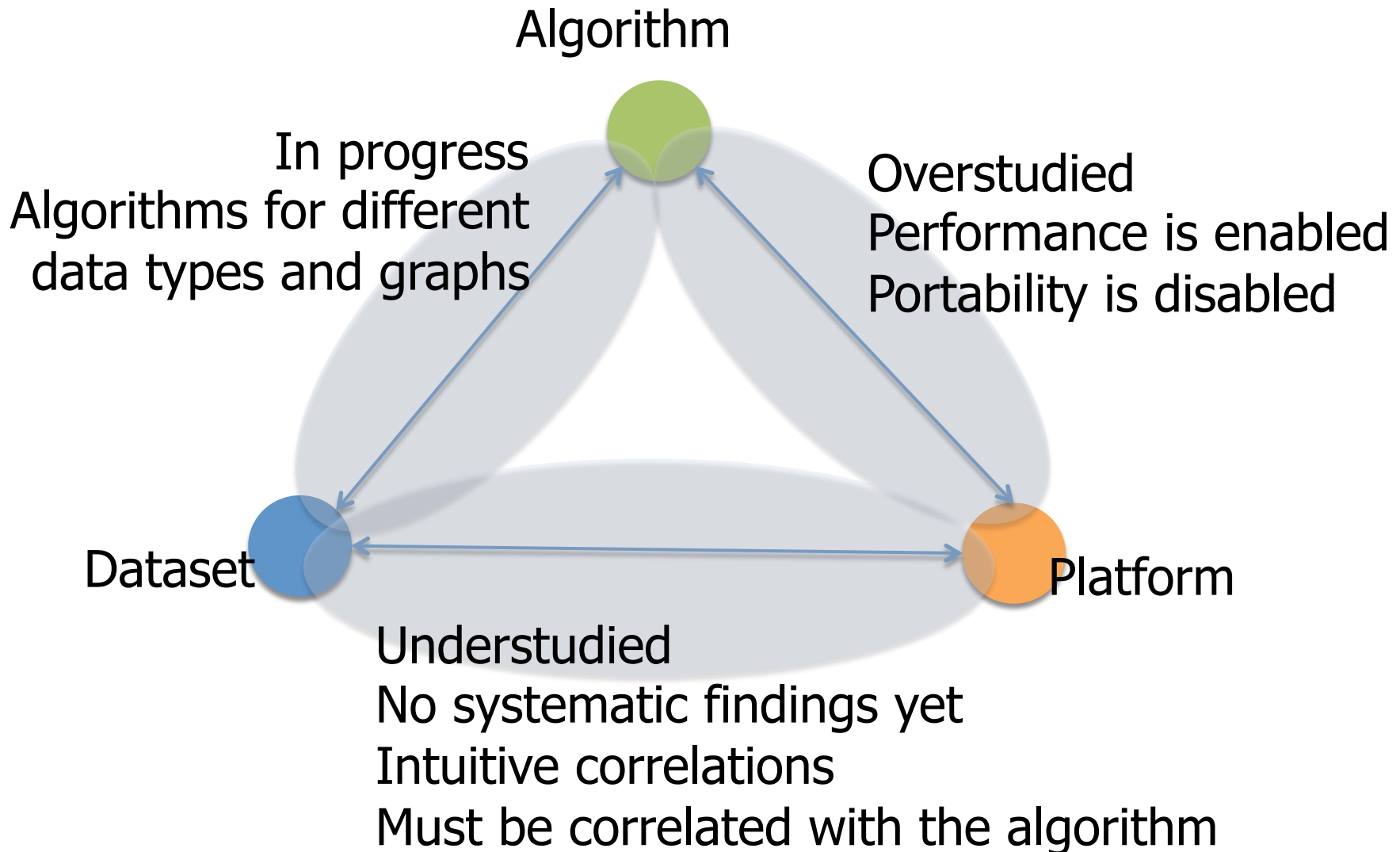
# Summary

# Take home message



- Large scale graph processing IS high performance computing
  - ▣ Due to/for data scale \*and\* analysis complexity
- HPC hardware (many-core processors) are feasible for graph processing
  - ▣ yet performance is (for now) unpredictable
- Performance is dependent on all three “axes”
  - ▣ Performance =  $f$  (dataset, algorithm, hardware)

# P-A-D triangle





A horizontal decorative bar at the top of the slide, consisting of a red rectangular section on the left and a larger blue rectangular section on the right.

# Benchmarking II: Platforms

# Graph processing @ scale

- The characteristics of graph processing
  - ▣ Poor locality
  - ▣ Unstructured computation
  - ▣ Variable parallelism
  - ▣ Low computer-to-memory ratio
- @ Scale
  - ▣ Distributed processing is mandatory
  - ▣ Parallel processing is very useful

Implementing graph applications is already difficult. Dealing with large scale systems on top (below, in fact) them is even harder.

# Graph processing systems

- Provide simplified ways to develop graph processing applications
  - ▣ Typical scenario: analytics on single- or multi-node platforms
  - ▣ Heterogeneity is becoming popular
- Target \*productivity\* and \*performance\*
  - ▣ Productivity => ease-of-implementation, development time
  - ▣ Performance => optimized back-ends / engines / runtimes
  - ▣ Portability comes “for free”
- Both commercial and academic, many open-source

# Graph processing systems

Performance

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

Dedicated  
Systems

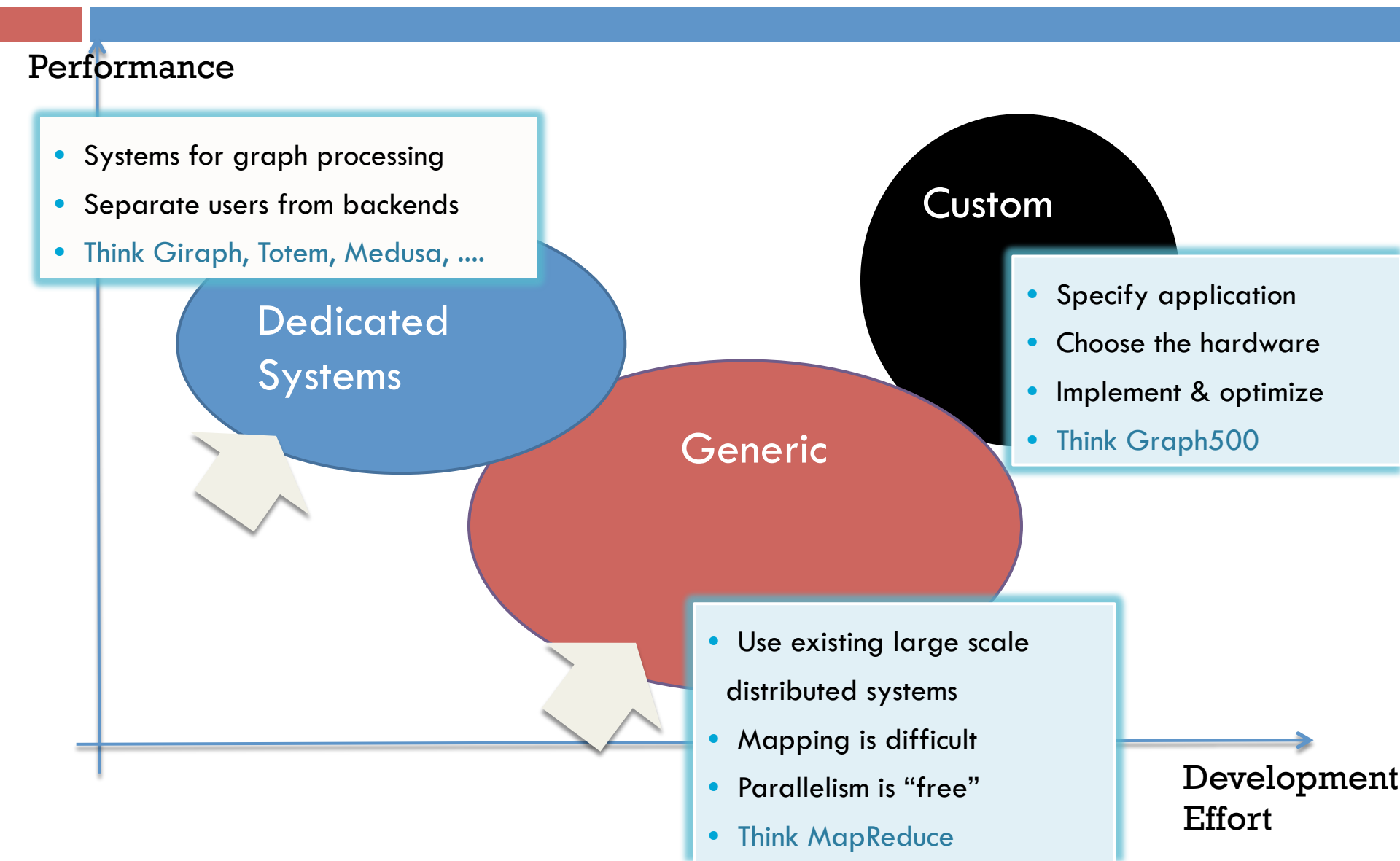
Custom

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

Generic

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

Development  
Effort



A horizontal decorative bar at the top of the slide, consisting of a red rectangular section on the left and a larger blue rectangular section on the right.

# GPU-enabled dedicated systems

# MapGraph\*

- GPU-only graph processing
  - ▣ CPU, single- and multi-GPU versions
- Vertex-centric API based on Gather-Apply-Scatter (GAS @ GraphLab)
  - ▣ **Gather**: reads the vertex's neighborhood.
  - ▣ **Apply**: updates the vertex based on the gather result.
  - ▣ **Scatter**: pushes updates to the vertex's neighborhood.
    - Users write functions for the G-A-S phases
- Two data structures (user-defined)
  - ▣ VertexList: data for each vertex.
  - ▣ EdgeList: data for each edge

# Medusa\*

- GPU-only graph processing
  - ▣ Single-node, multiple GPUs
- Programmability-driven, based on BSP
  - ▣ EMV (edge-message-vertex) model
    - Extension of the Vertex-centric Pregel-like model
  - ▣ GPU-specific back-end optimization
- Simple API that hides GPU programming
  - ▣ Define data structures
  - ▣ Define operations for edges, messages, vertices
  - ▣ Compose the algorithm from these operations
  - ▣ Run (iteratively) over the graph

# Totem\*

- Heterogeneous CPU+GPUs graph processing
  - ▣ C+CUDA for specifying applications
  - ▣ (Thin) API for heterogeneity
  - ▣ Based on BSP (and close to Pregel)
- Partitions data (edge-based) between CPUs and GPUs
  - ▣ Based on processing capacity
  - ▣ Minimizing the overhead of communication
    - Buffer schemes, aggregation, smart partitioning
- A user-defined vertex-centric kernel runs simultaneously on each partition (CPUs, GPUs)
  - ▣ Vertices are processed in parallel within each partition
  - ▣ Messages can be combined



# Experiments\*

\*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	$\bar{D}$	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 ( $\times 10^{-5}$ ).  $\bar{D}$  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).

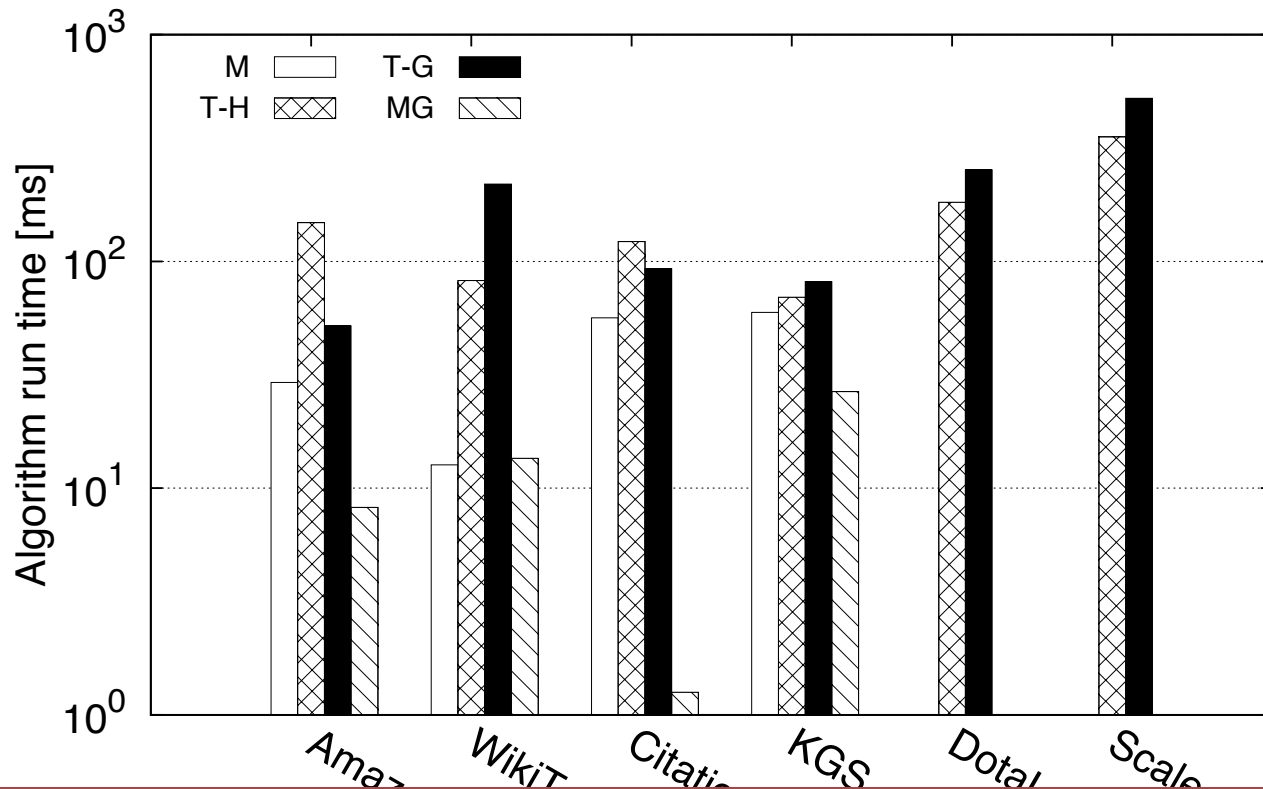
# Setup: datasets

Graphs	V	E	d	$\bar{D}$	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)	4,611,438	258,672,162	1.2	56	257,010
Scale-25 (U)	17,062,472	1,047,207,019	0.4	61	639,144

CSR-represented graphs fit in memory (GTX480)

V and E are the vertex count and edge count of the graphs. d is the link density ( $\times 10^{-5}$ ).  $\bar{D}$  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).

# BFS [algorithm]

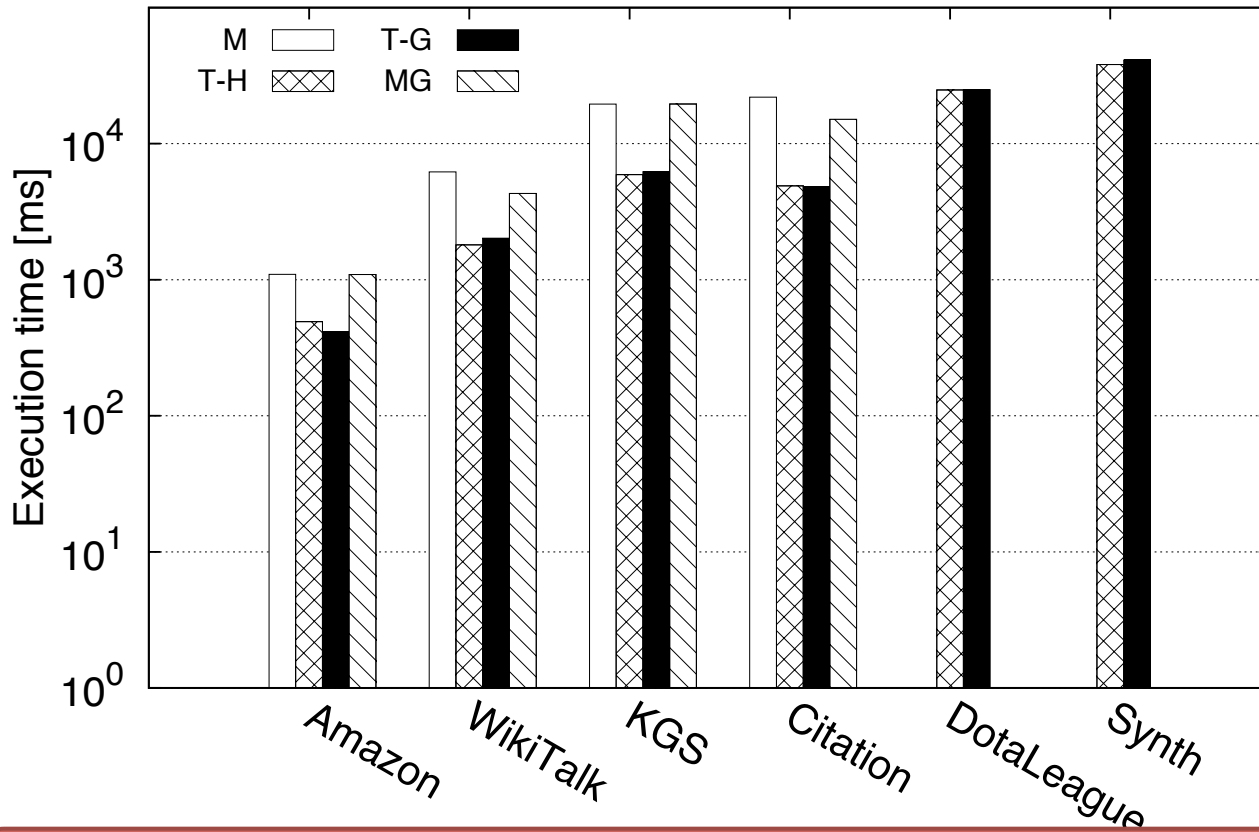


Strong dependency on the graph.

Totem is the worst performer.

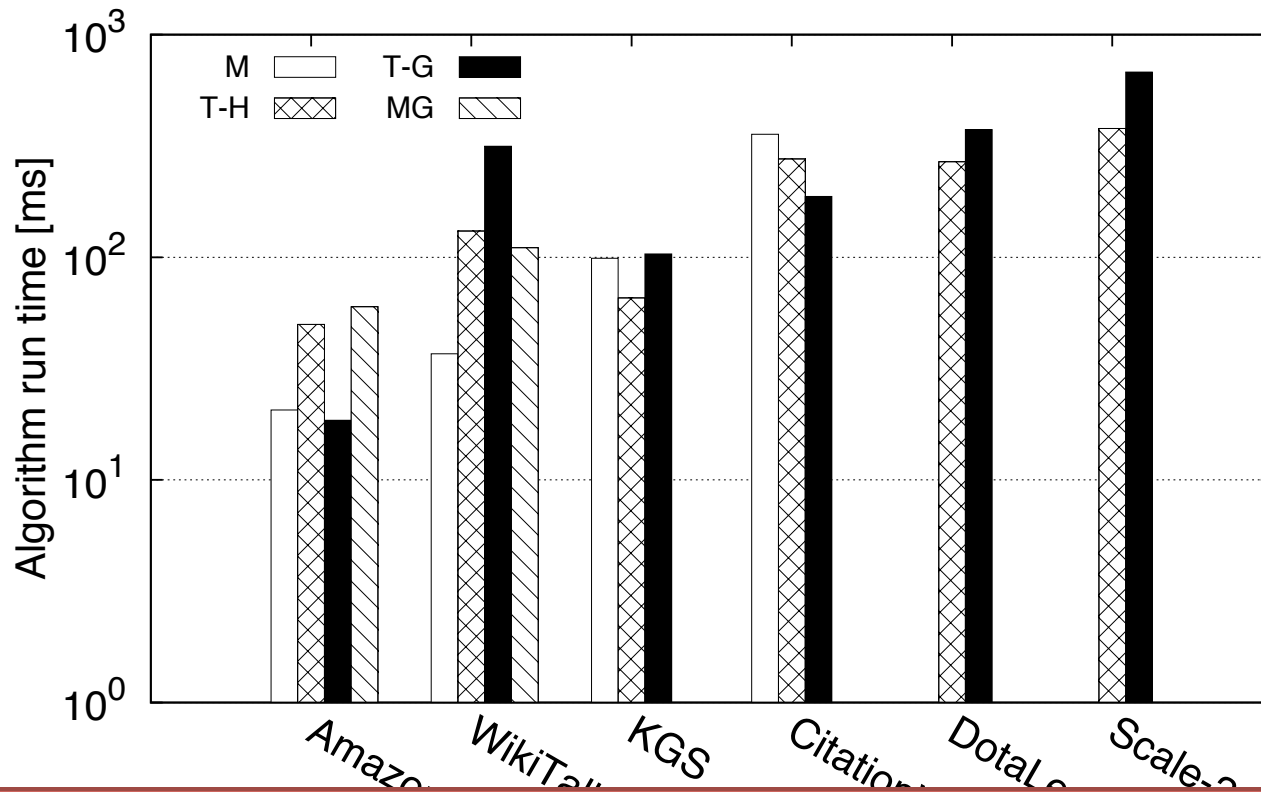
Medusa and MapGraph cannot handle large graphs.

# BFS [full]



Totem becomes the best performer !

# WCC [algorithm]

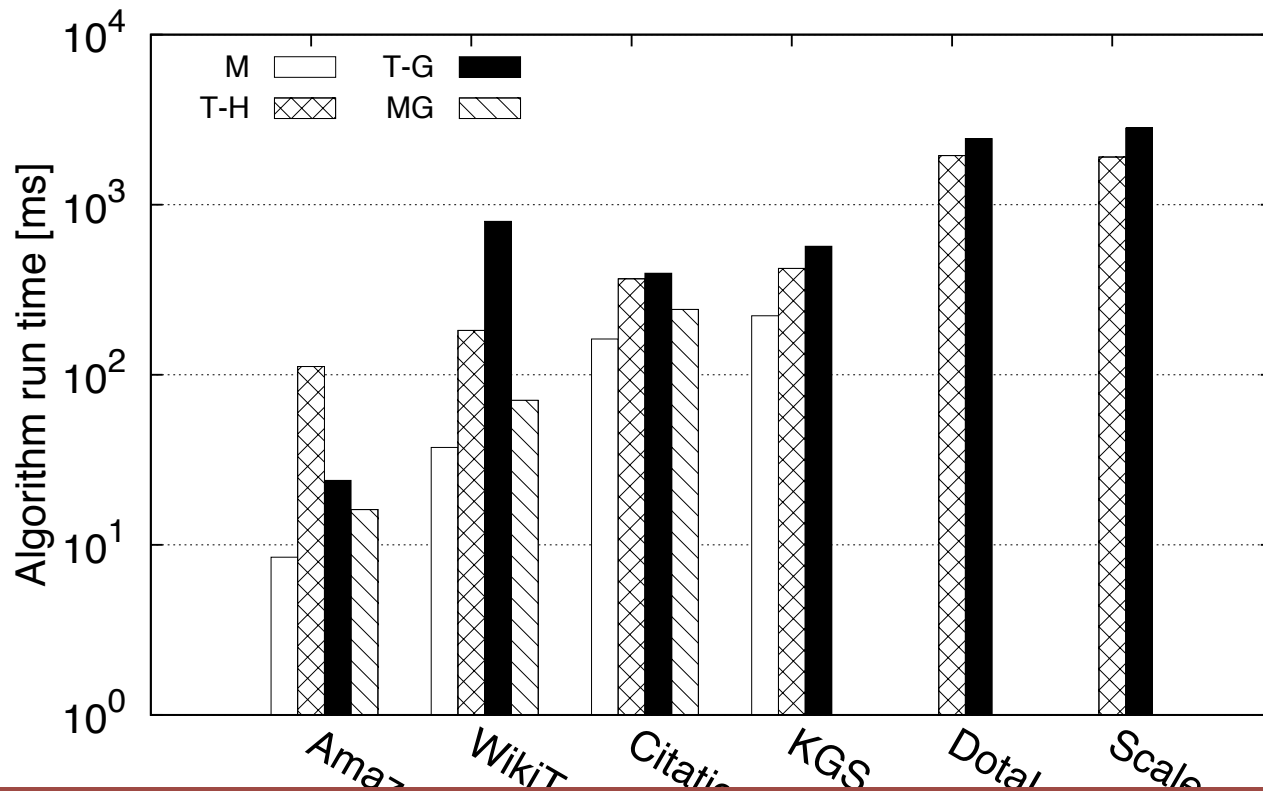


Strong dependency on the graph.

No best/worst performer.

More crashes of MapGraph.

# PageRank [algorithm]



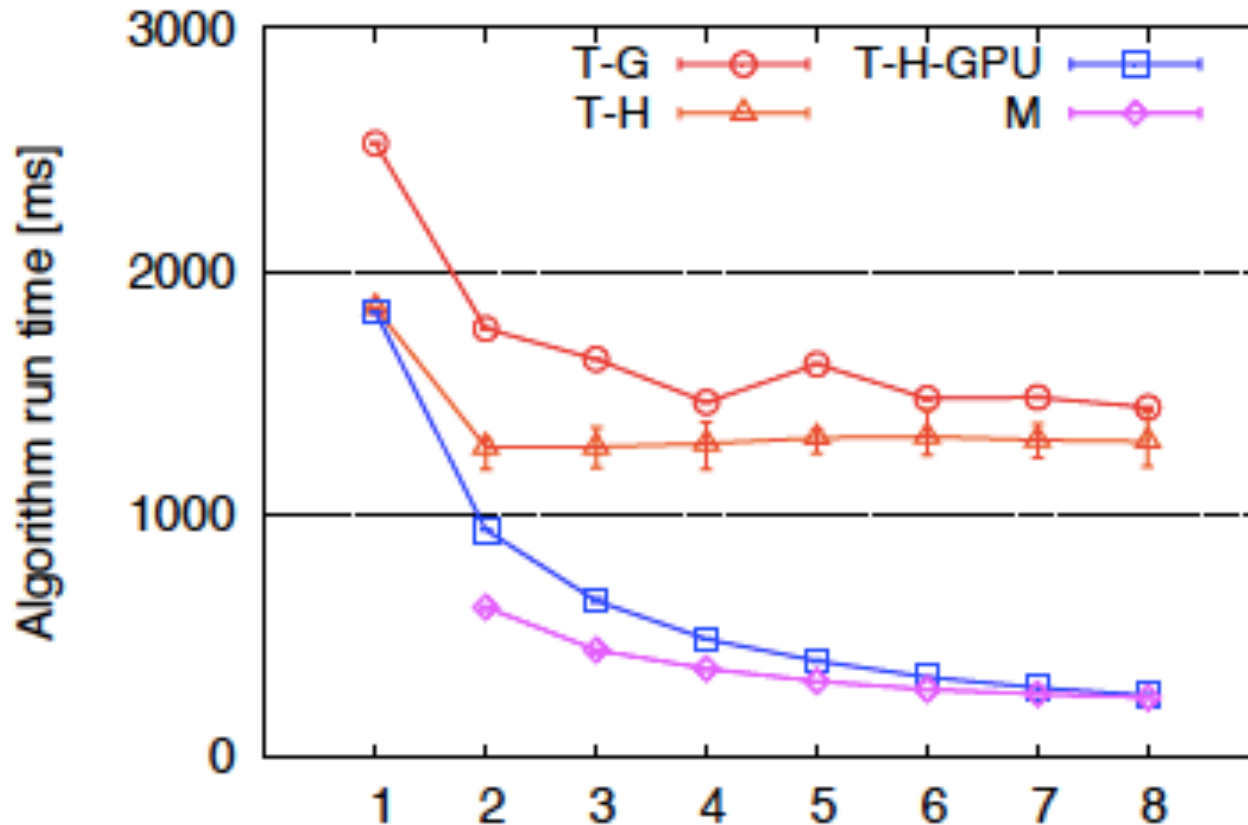
Most compute-intensive.

Totem performs worst.

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

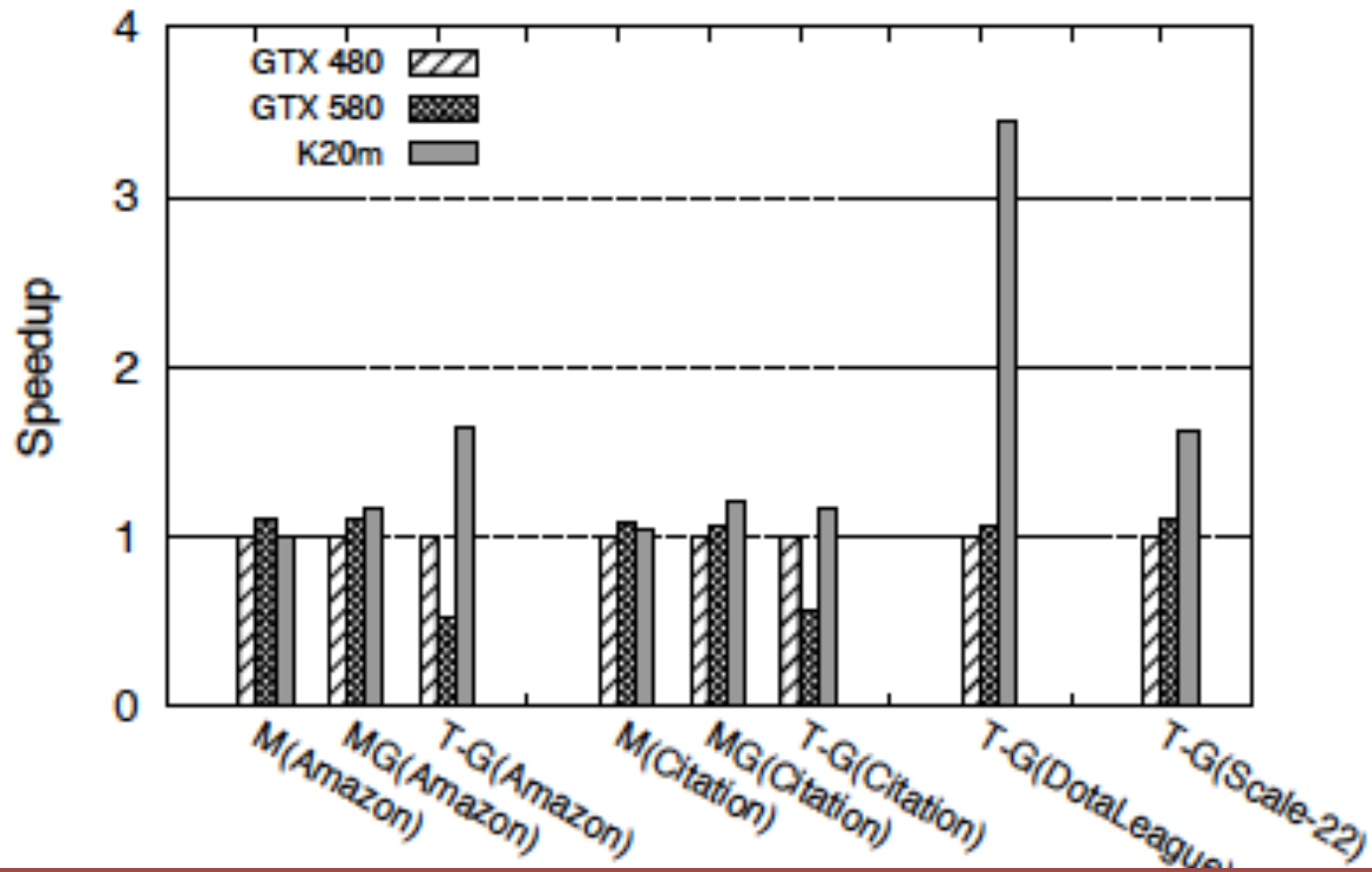


# Multi-GPU scalability



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.

# 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.



# Distributed/Large Scale platforms

# Graph processing systems

Performance

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

Dedicated  
Systems

Custom

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

Generic

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

Development  
Effort

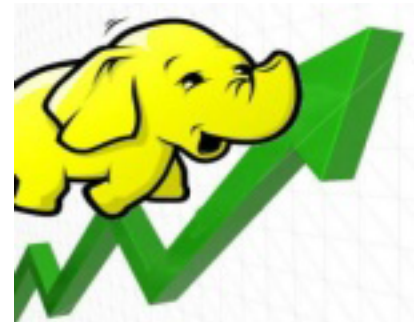
# Hadoop (Generic)

- The most popular MapReduce implementation
  - ▣ Generic system for large-scale computation
- Pros:
  - ▣ Easy to understand model
  - ▣ Multitude of tools and storage systems
- Cons:
  - ▣ Express the graph application in the form of MapReduce
  - ▣ Costly disk and network operations
  - ▣ No specific graph processing optimizations



# Hadoop2 with YARN (Generic)

- Next generation of Hadoop
  - ▣ Supports old MapReduce jobs
  - ▣ Designed to facilitate multiple programming models (frameworks, e.g., Spark)
- Separates resource management (YARN) and job management
  - ▣ MapReduce manages jobs using resources provided by YARN



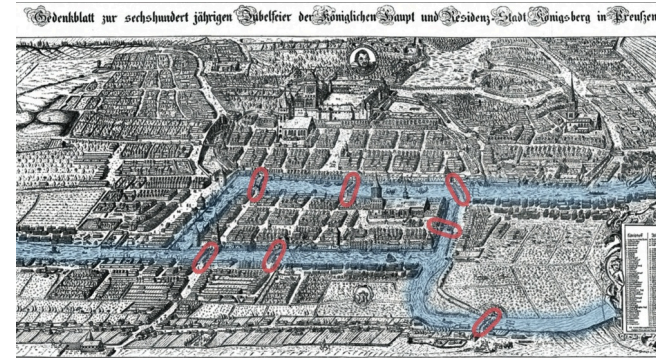
# Stratosphere (Generic)

- Now Apache Flink
- Nephele resource manager
  - ▣ Scalable parallel engine
  - ▣ Jobs are represented as DAGs
  - ▣ Supports data flow in-memory, via network, or on files
- PACT job model
  - ▣ 5 second-order functions (MapReduce has 2):  
Map, Reduce, Match, Cross, and CogGroup
  - ▣ Code annotations for compile-time plans
  - ▣ Compiled as DAGs for Nephele

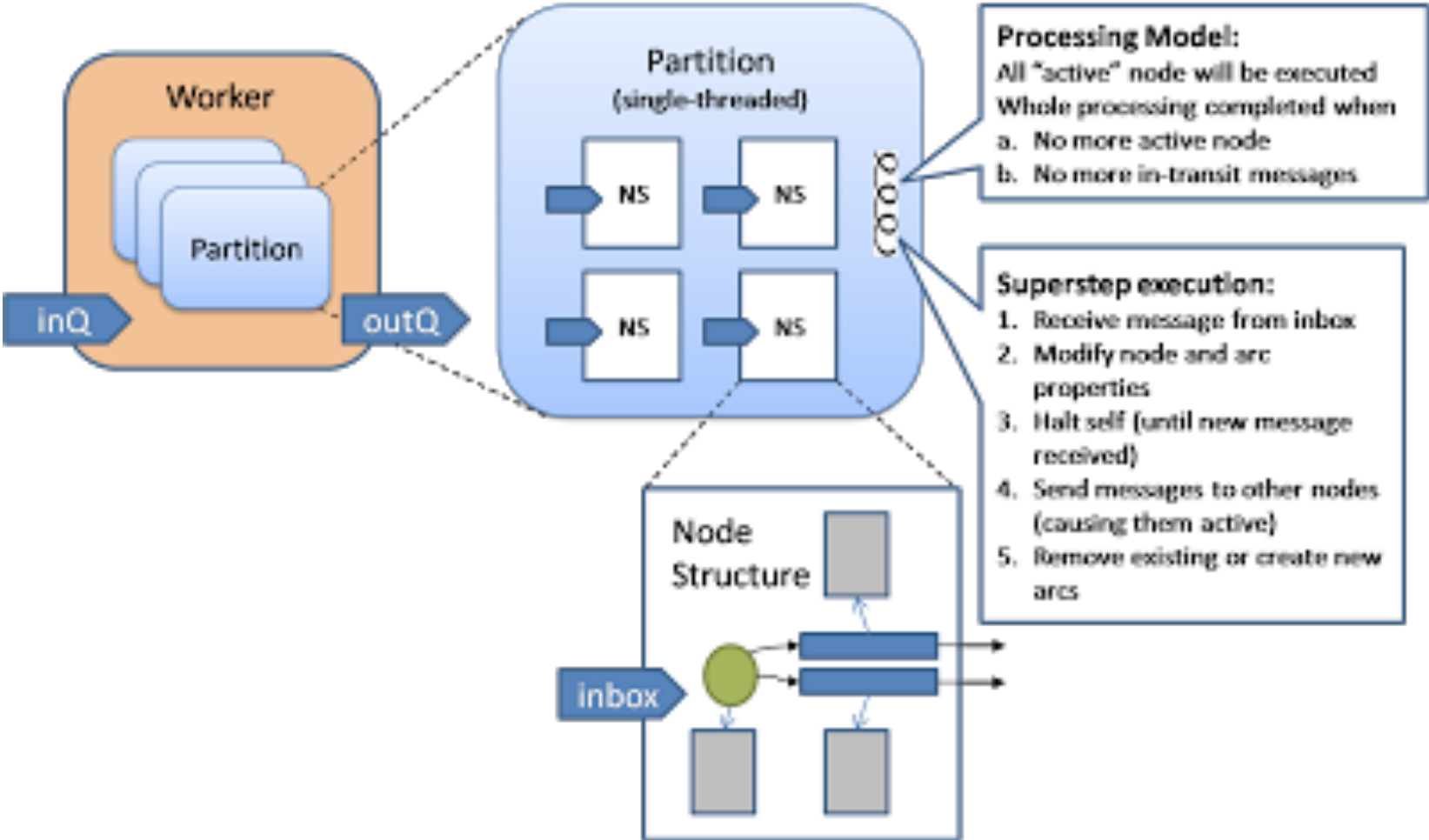


# Pregel graph-processing model

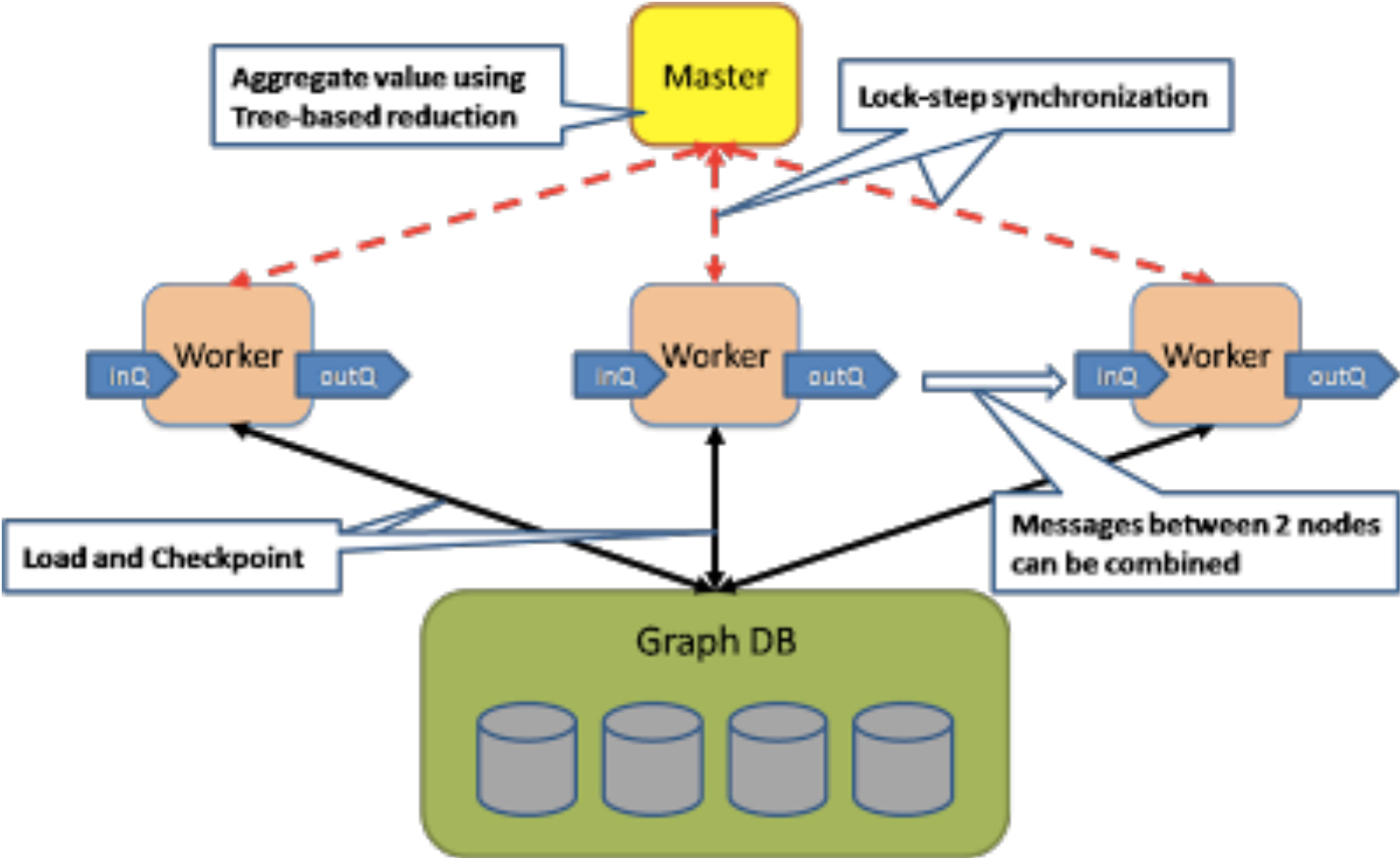
- Proposed a **vertex-centric** approach to graph processing
  - ▣ Graph-to-graph transformations
- Front-end:
  - ▣ Write the computation that runs on all vertices
  - ▣ Each vertex can vote to halt
    - All vertexes halt => terminate
  - ▣ Can add/remove edges and vertices
- Back-end:
  - ▣ Uses the BSP model
  - ▣ Message passing between nodes
    - Combiners, aggregators
  - ▣ Checkpointing for fault-tolerance



# Pregel

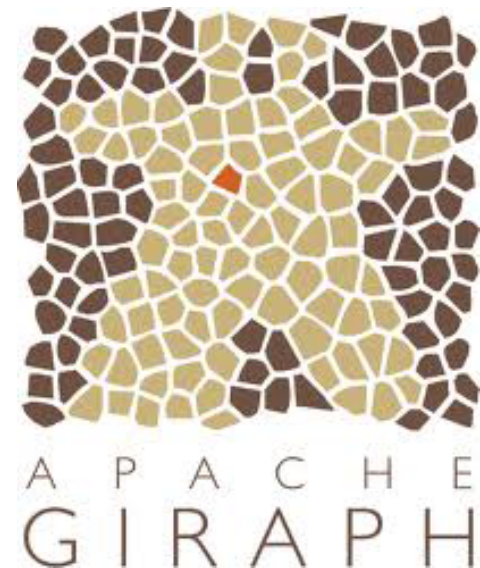


# Pregel



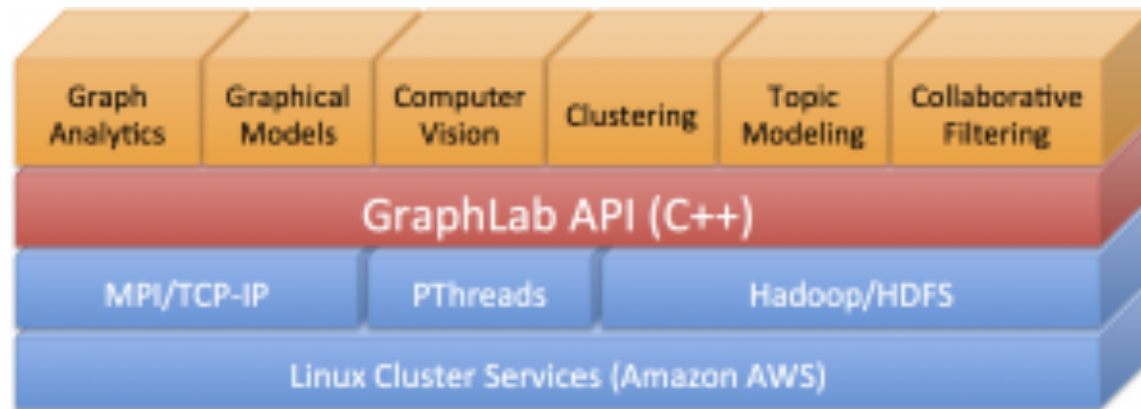
# Apache Giraph (Dedicated)

- Based on the Pregel model
- Uses YARN as back-end (yet another framework)
- In-memory
  - ▣ Limitations in terms of partition sizes
  - ▣ Spilling to disk is work in progress
- Enables
  - ▣ Iterative data processing
  - ▣ Message passing, aggregators, combiners



# GraphLab (Dedicated)

- Distributed programming model for machine learning
  - ▣ Provides an API for graph processing, C++ based (now Python)

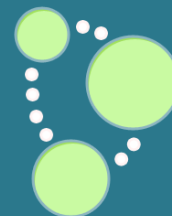


- All in-memory
- Supports asynchronous processing
- GraphChi is its single-node version, Dato as GraphLab company



# Neo4J (Dedicated)

- Very popular graph **database**
  - ▣ Graphs are represented as relationships and annotated vertices
- Single-node system
  - ▣ Uses parallel processing
  - ▣ Additional caching and query optimizations
  - ▣ All in-memory
- The most widely used solutions for medium-scale problems



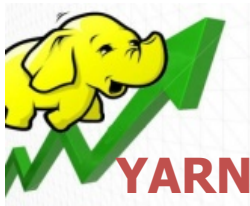
Neo4j  
the graph database

# Experiments\*

\*Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke.  
How Well do Graph-Processing Platforms Perform? An Empirical  
Performance Evaluation and Analysis, IPDPS 2014

# Platforms we have evaluated

- Distributed or non-distributed
- Dedicated or generic



Distributed (Generic)



Distributed  
(Dedicated)



Non-distributed  
(Dedicated)



# Setup








- Benchmarking-like experiment
  - 6 algorithms
  - 7 data-sets
  - 7 platforms
- Implement **all algorithms on all platforms**
- Run and compare ...
  - Performance
- Estimate usability\*

# Hardware

- DAS4: a multi-cluster Dutch grid/cloud
  - ▣ Intel Xeon 2.4 GHz CPU (dual quad-core, 12 MB cache)
  - ▣ Memory 24 GB
  - ▣ 1 Gbit/s Ethernet network
- Size
  - ▣ Most experiments take 20 working machines
  - ▣ Up to 50 working machines
- HDFS used as distributed file system

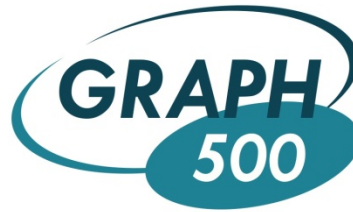


# Datasets

	Graphs	#V	#E	d	$\bar{D}$	Directivity
 G1	Amazon	262,111	1,234,877	1.8	4.7	directed
 G2	WikiTalk	2,388,953	5,018,445	0.1	2.1	directed
 G3	KGS	293,290	16,558,839	38.5	112.9	undirected
 G4	Citation	3,764,117	16,511,742	0.1	4.4	directed
 G5	DotaLeague	61,171	50,870,316	2,719.0	1,663.2	undirected
 G6	Synth	2,394,536	64,152,015	2.2	53.6	undirected
 G7	Friendster	65,608,366	1,806,067,135	0.1	55.1	undirected



<https://snap.stanford.edu/>



<http://www.graph500.org/>

**The Game Trace Archive**

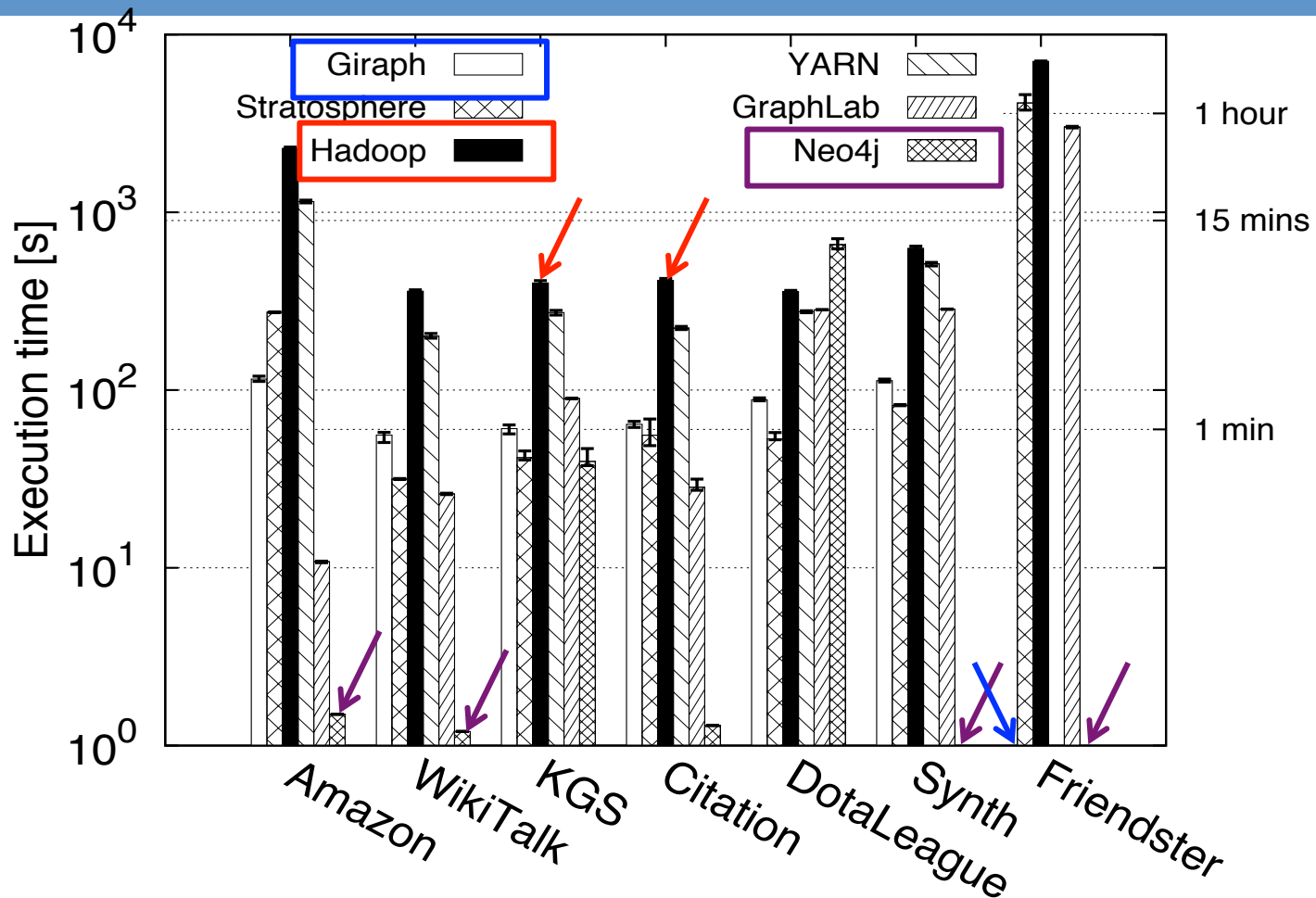
<http://gta.st.ewi.tudelft.nl/>

# Graph-Processing Algorithms

- Literature survey
  - ▣ 10 top research conferences: SIGMOD, VLDB, HPDC ...
  - ▣ 2009–2013, 124 articles

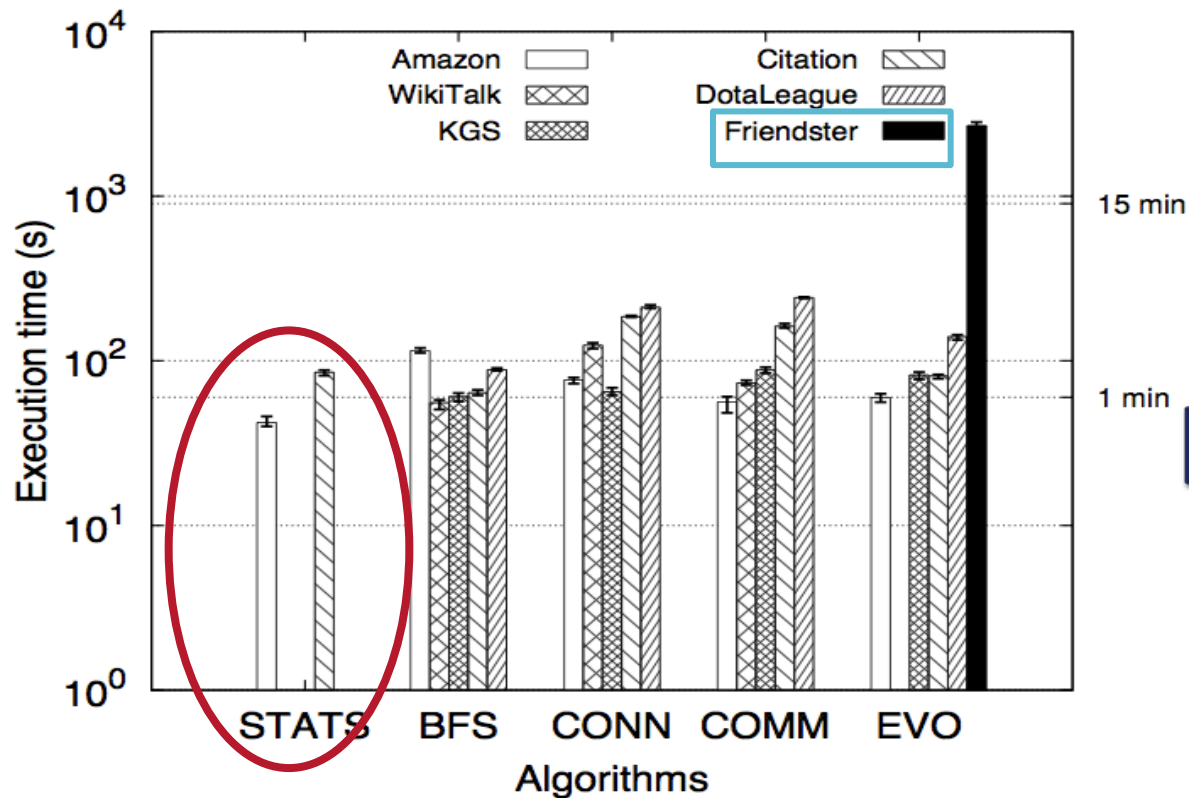
Class	Examples	%
Graph Statistics	Diameter, PageRank	16.1
Graph Traversal	BFS, SSSP, DFS	46.3
Connected Component	Reachability, BiCC	13.4
Community Detection	Clustering, Nearest Neighbor	5.4
Graph Evolution	Forest Fire Model, PAM	4.0
Other	Sampling, Partitioning	14.8

# BFS: results for all-2-all



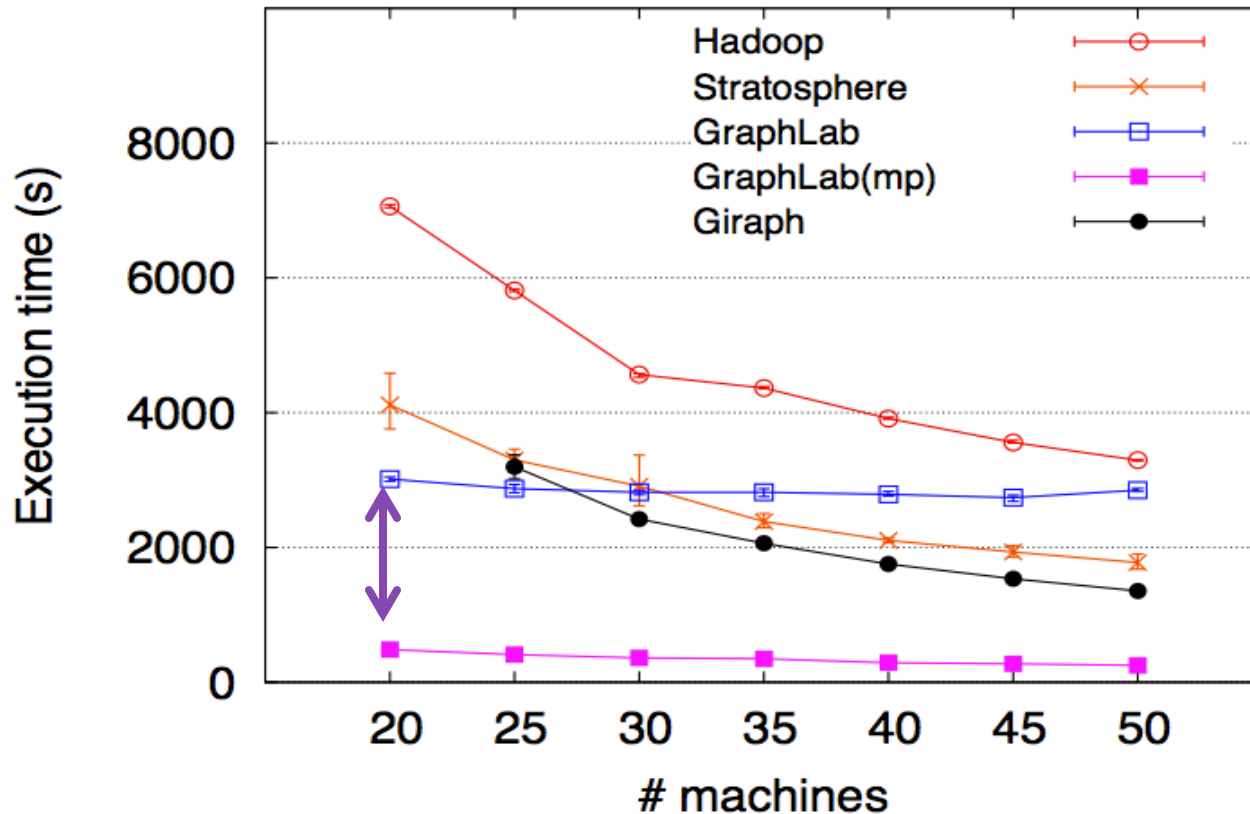
No platform runs fastest for all graphs, but Hadoop is the worst performer.  
Not all platforms can process all graphs, but Hadoop processes everything.

# Giraph: results for all algorithms, all data sets



Storing the whole graph in memory helps Giraph perform well  
Giraph may crash when **graphs** or number of **messages** large

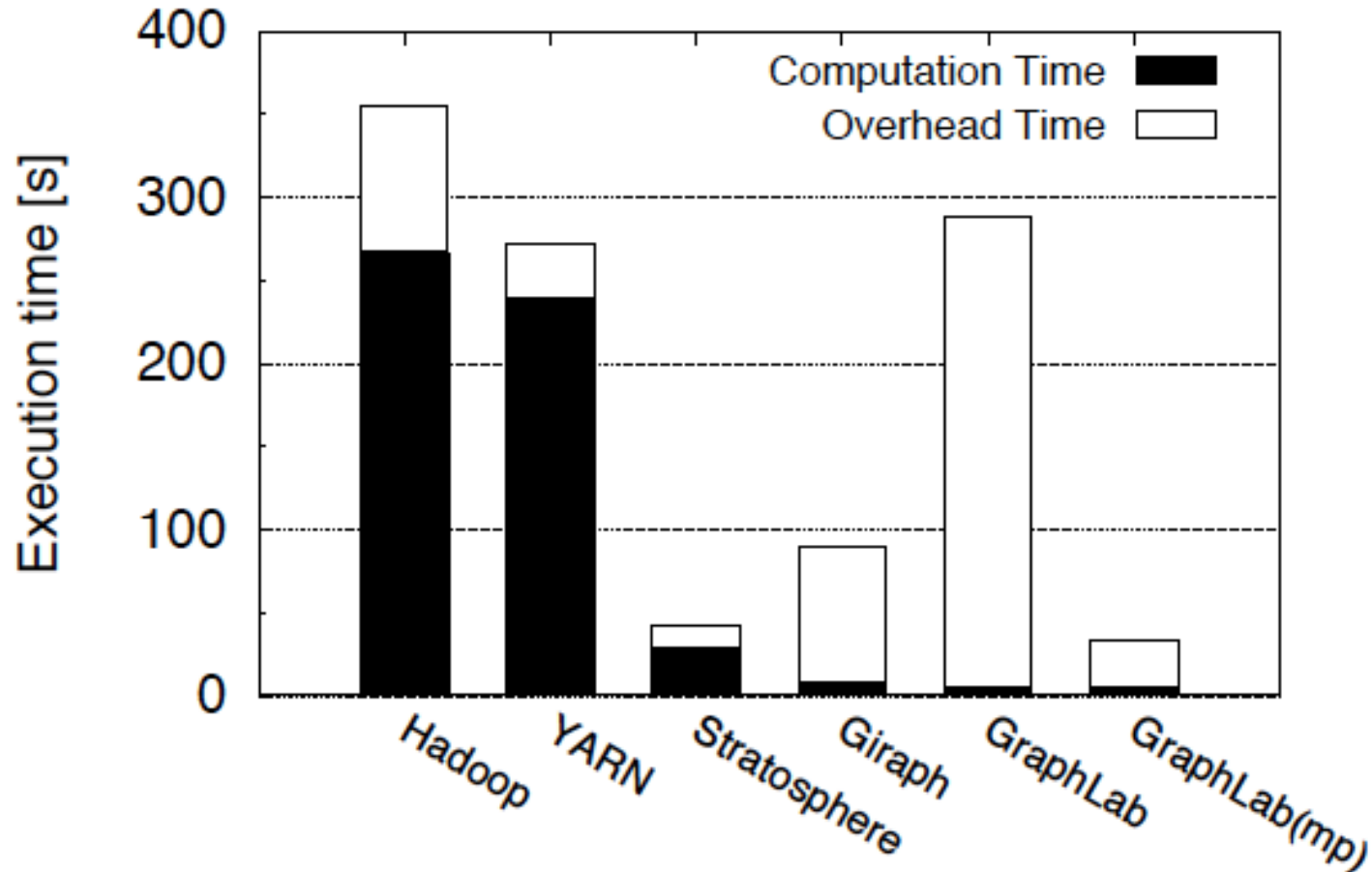
# Horizontal scalability: BFS on Friendster (31 GB)



Using more computing machines can reduce execution time

Tuning needed for horizontal scalability, e.g., for GraphLab, split large input files into number of chunks equal to the number of machines

# Overhead (BFS, DotaLeague)



We need new metrics, to capture meaning of computation time (more later)  
In some systems, overhead is by and large wasted time (e.g., in Hadoop)



# Additional Overheads: Data ingestion

- Data ingestion
  - ▣ Batch system: one ingestion, multiple processing
  - ▣ Transactional system: one ingestion, one processing
  
- Data ingestion matters even for batch systems

	Amazon	DotaLeague	Friendster
HDFS	1 second	7 seconds	5 minutes
Neo4J	4 hours	<b>days</b>	<b>n/a</b>

# Productivity

	Hadoop(Java)	Stratosphere(Java)	Giraph(Java)	GraphLab(C++)	Neo4j(Java)
BFS	1 d, 110 loc	1 d, 150 loc	1 d, 45 loc	1 d, 120 loc	1 h, 38 loc
CONN	1.5 d, 110 loc	1 d, 160 loc	1 d, 80 loc	0.5 d, 130 loc	1 d, 100 loc

- Low throughput in terms of LOC for all models
- Days to hours development time for the simpler applications

We need better productivity metrics!

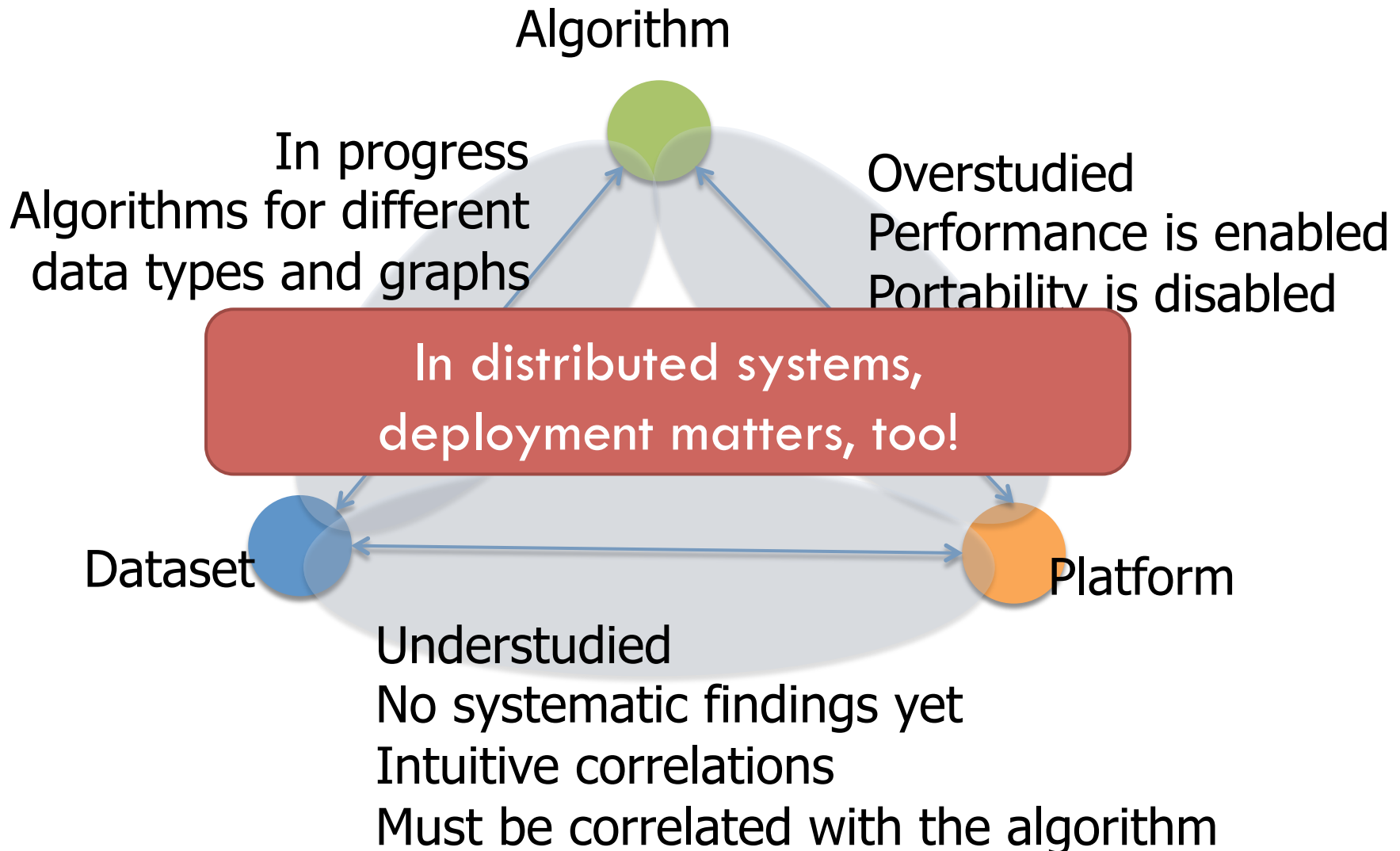
# Lessons learned\*

- Performance is function of (Dataset, Algorithm, Platform, Deployment)
  - ▣ Previous performance studies may lead to tunnel vision
- Platforms have their own drawbacks (crashes, long execution time, tuning, etc.)
  - ▣ Best-performing is not only low response time
  - ▣ Ease-of-use of a platform is very important
- Some platforms can scale up reasonably with cluster size (horizontally) or number of cores (vertically)
  - ▣ Strong vs weak scaling still a challenge
    - workload scaling tricky

\*All results and details:

<http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2013/PDS-2013-004-4.pdf>

# P-A-D triangle revisited





# Future directions

# Graphalytics\*

- Benchmarking graph processing systems
  - ▣ Selection of datasets
    - Synthetic, with real profiles
    - Real-life
  - ▣ Selection of algorithms
    - Problem-based
    - Code-based
  - ▣ Selection of metrics
  - ▣ Selection of systems
    - All of them.
  - ▣ Reporting the results
    - Goal-oriented

Oracle Labs



UNIVERSITY OF AMSTERDAM

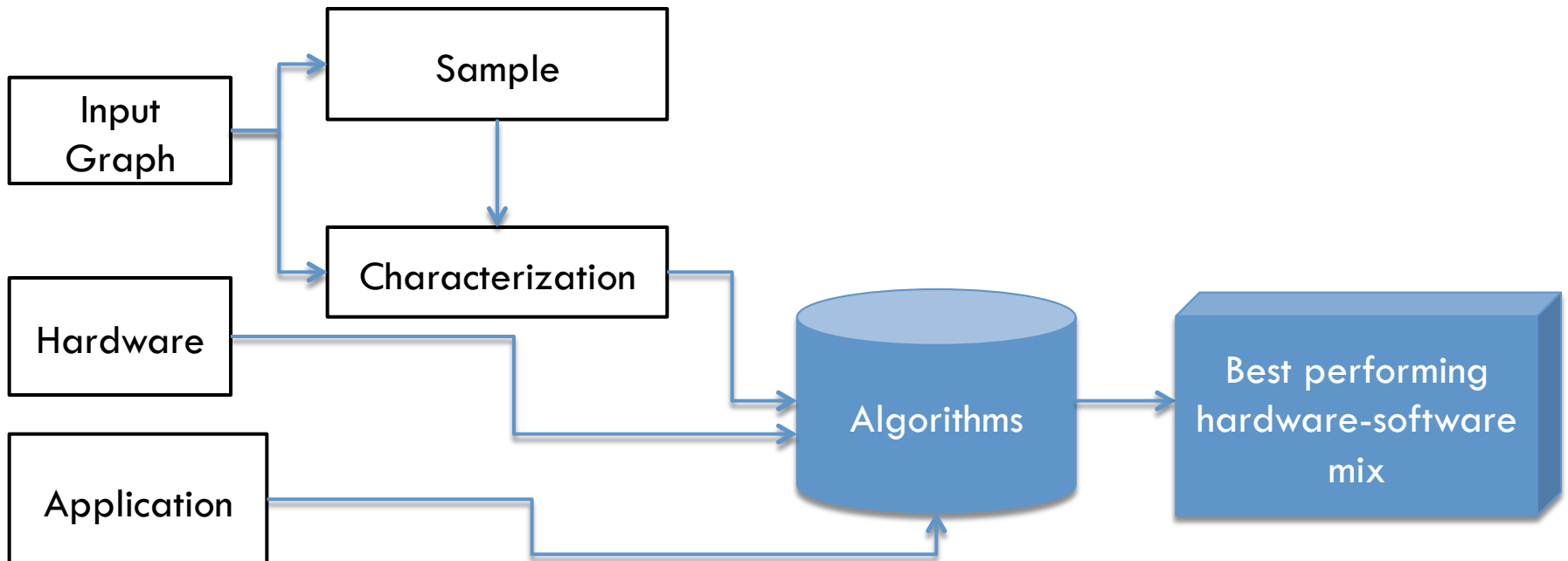


# Heterogeneous computing

- Build *\*the first\** multi-node heterogeneous graph processing system
  - ▣ Model performance
    - Are GPUs always useful in a distributed setup?
    - Which partition goes where?
  - ▣ Implementation
    - Based on existing distributed systems
    - Add graph-specific scheduling and resource allocation

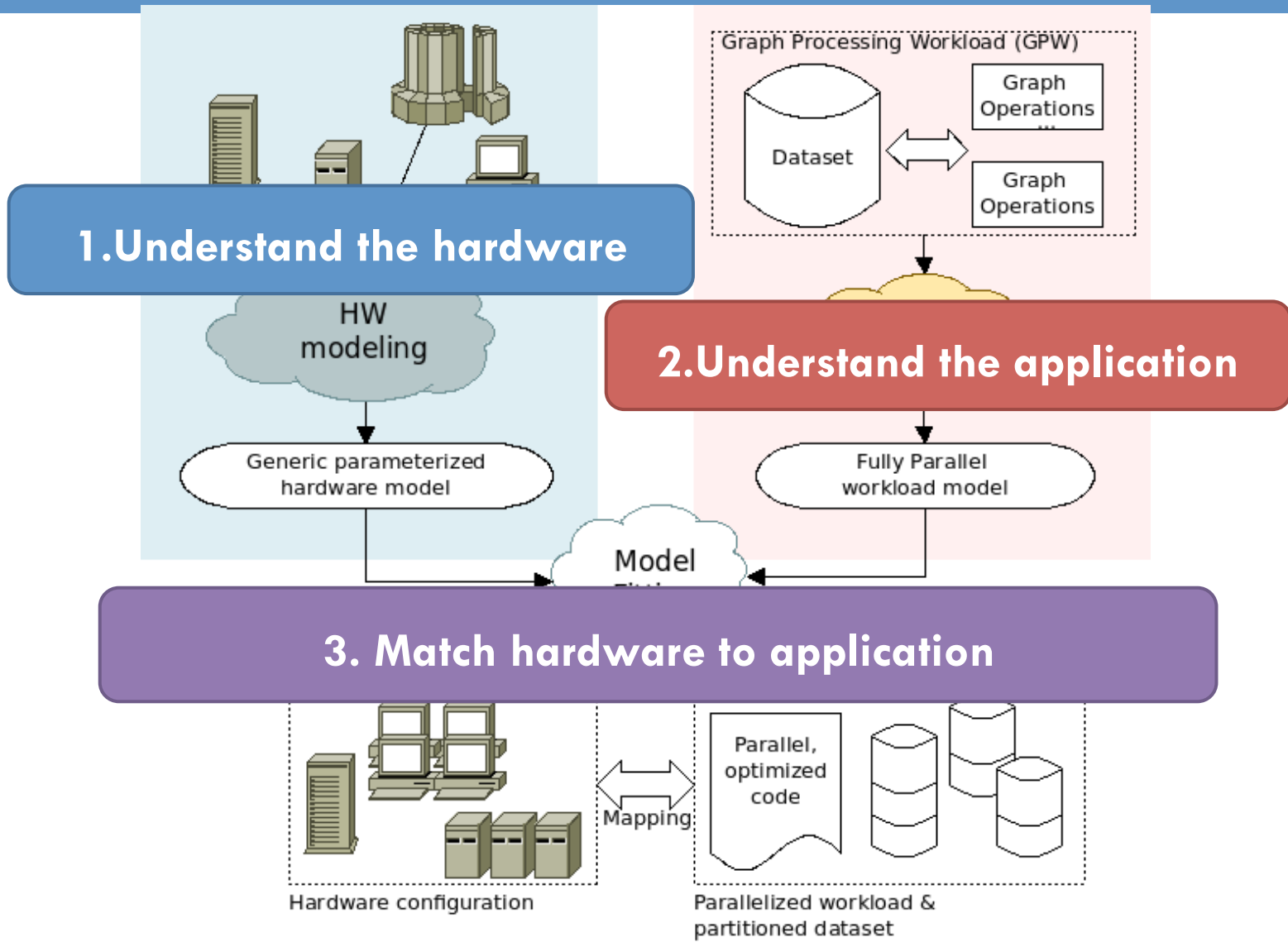
# Graph-centric framework

- Understand **graph features**
  - ▣ What makes a graph “special”
- Select the best algorithm for a given graph

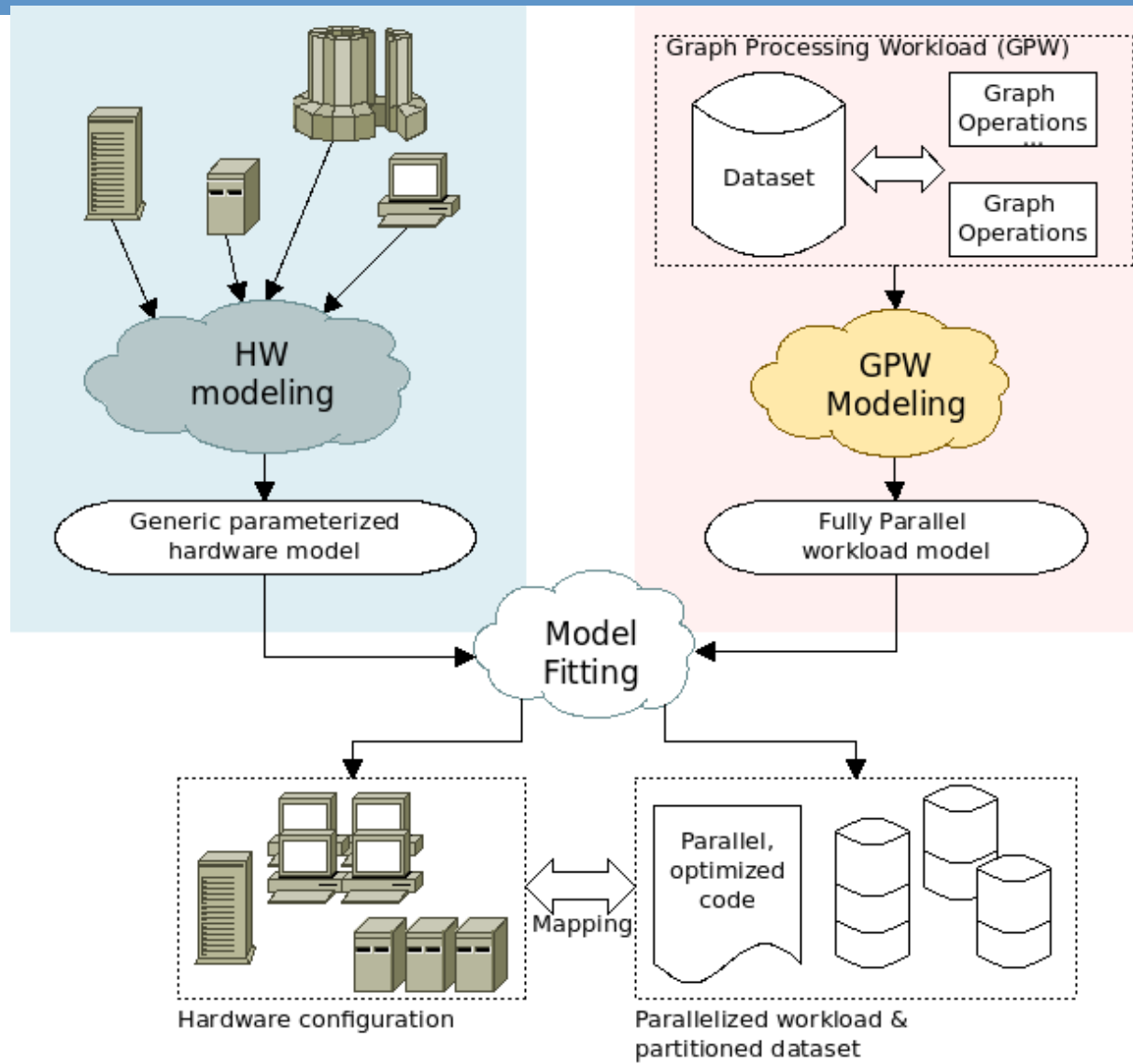




# End goal: Graphitti



# End goal: Graphitti





# Summary

# Take home message



- Graph processing is a hot topic for both software and hardware developers
- Challenges in scale and irregularity
- Existing graph processing systems : 80+
- Choose which one to use
  - Quick-Pick: choose a platform where your graph fits and you can program.
  - Systematic: meta-benchmarking, a.k.a., Graphalytics

# Take home message



- Comprehensive and systematic performance study of graph processing systems is difficult.
- Main challenges
  - ▣ fairness of comparison
  - ▣ development time
- Large-scale systems are promising, but adoption remains low.
- GPU-enabled systems show promising performance, but ... no dedicated distributed GPU-enabled graph processing systems – YET!



# Future research directions

---

- Improved benchmarking
- Heterogeneous computing
- Workload characterization
- Smart resource allocation

# Questions ?

---



A.L.Varbanescu@uva.nl

Online: [graphalytics.ewi.tudelft.nl](http://graphalytics.ewi.tudelft.nl)