Efficient Large-Scale Graph Processing on Single Computer

Eiko Yoneki
eiko.yoneki@cl.cam.ac.uk
http://www.cl.cam.ac.uk/~ey204

Systems Research Group
University of Cambridge Computer Laboratory

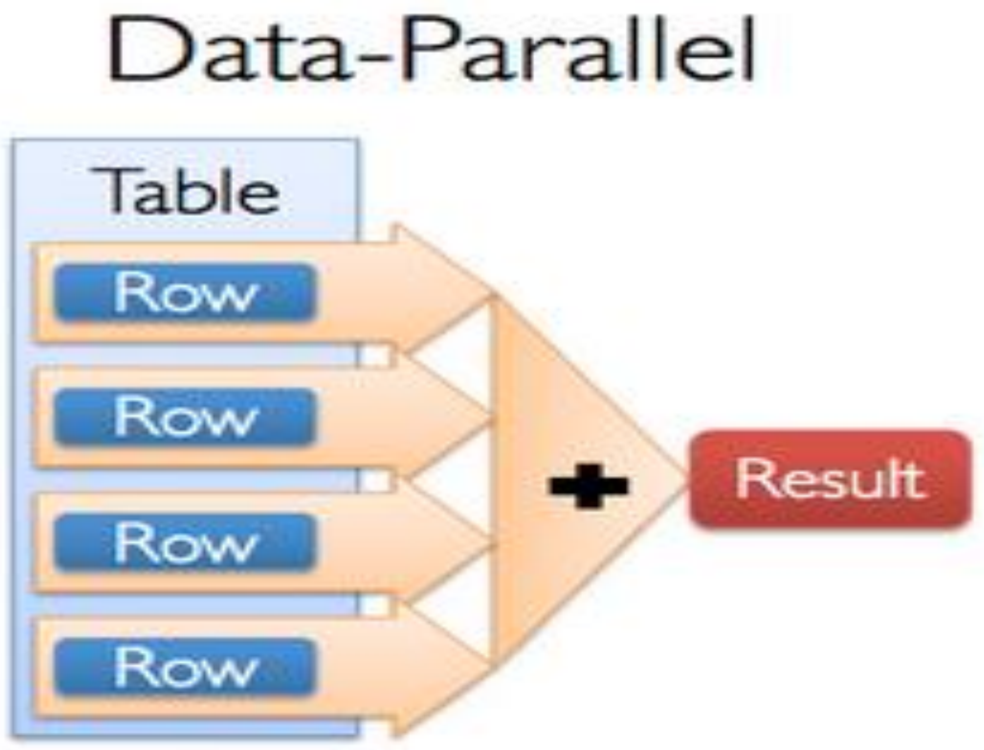
Massive Data: Scale-Up vs Scale-Out

- Popular solution for massive data processing
  → scale and build distribution, combine theoretically unlimited number of machines in single distributed storage

- Scale-up: add resources to single node (many cores) in system (e.g. HPC)
- Scale-out: add more nodes to system (e.g. Amazon EC2)

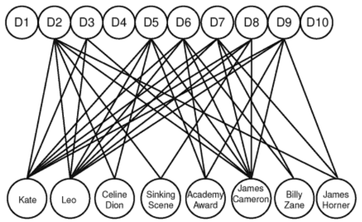
Maximising Parallelism: Data Parallel

- Distributed computing infrastructure with partitioned data (e.g. Word count with MapReduce)

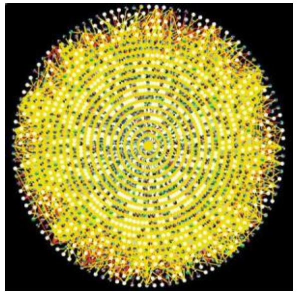


Emerging Massive-Scale Graph Data

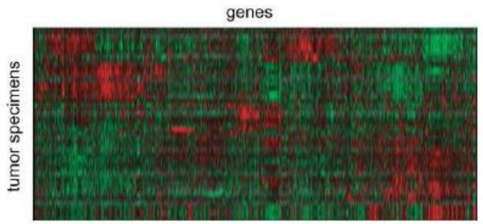
- Massive data forms complex networks: key to solve problems in diverse fields
- Storage is available:1 trillion edges x 16 bytes per edge = 16 TB storage



Bipartite graph of phrases in documents



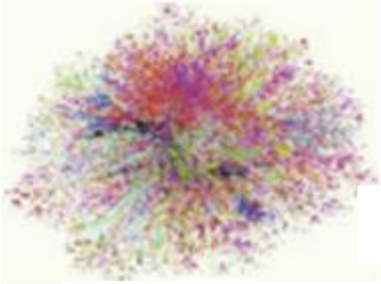
Protein Interactions [genomebiology.com]



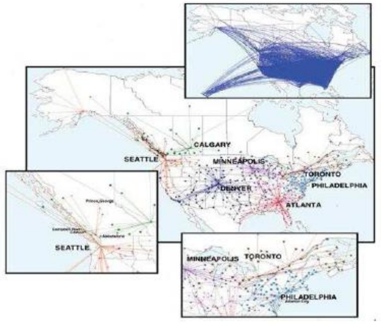
Gene expression data

Brain Networks: 100B neurons(700T links) requires 100s GB memory

Social media data

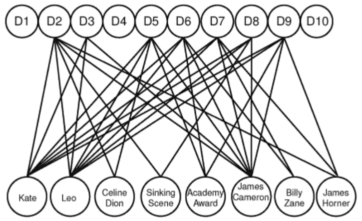


Web 1.4B pages(6.6B links)

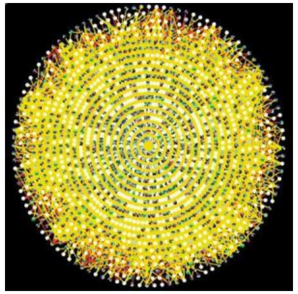


Airline Graphs

Emerging Massive-Scale Graph Data



Bipartite graph of phrases in documents



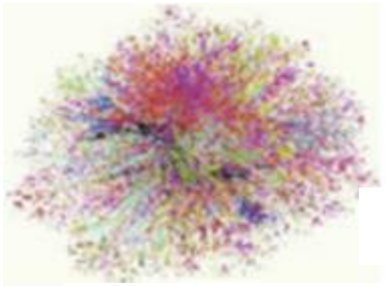
Protein Interactions [genomebiology.com]

Gene expression data

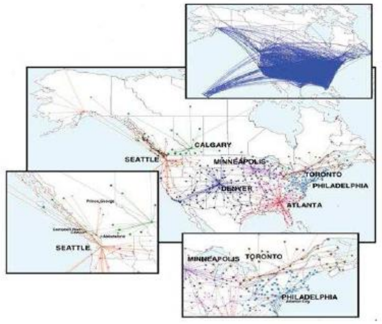
BFS
DFS
CC
SCC
SSSP
ASP
A*
Community
Centrality
Diameter
Page Rank
MIS
SALSA...

Brain Networks: 100B neurons(700T links) requires 100s GB memory

Social media data



Web 1.4B pages(6.6B links)



Airline Graphs

Everything will be connected in Future!

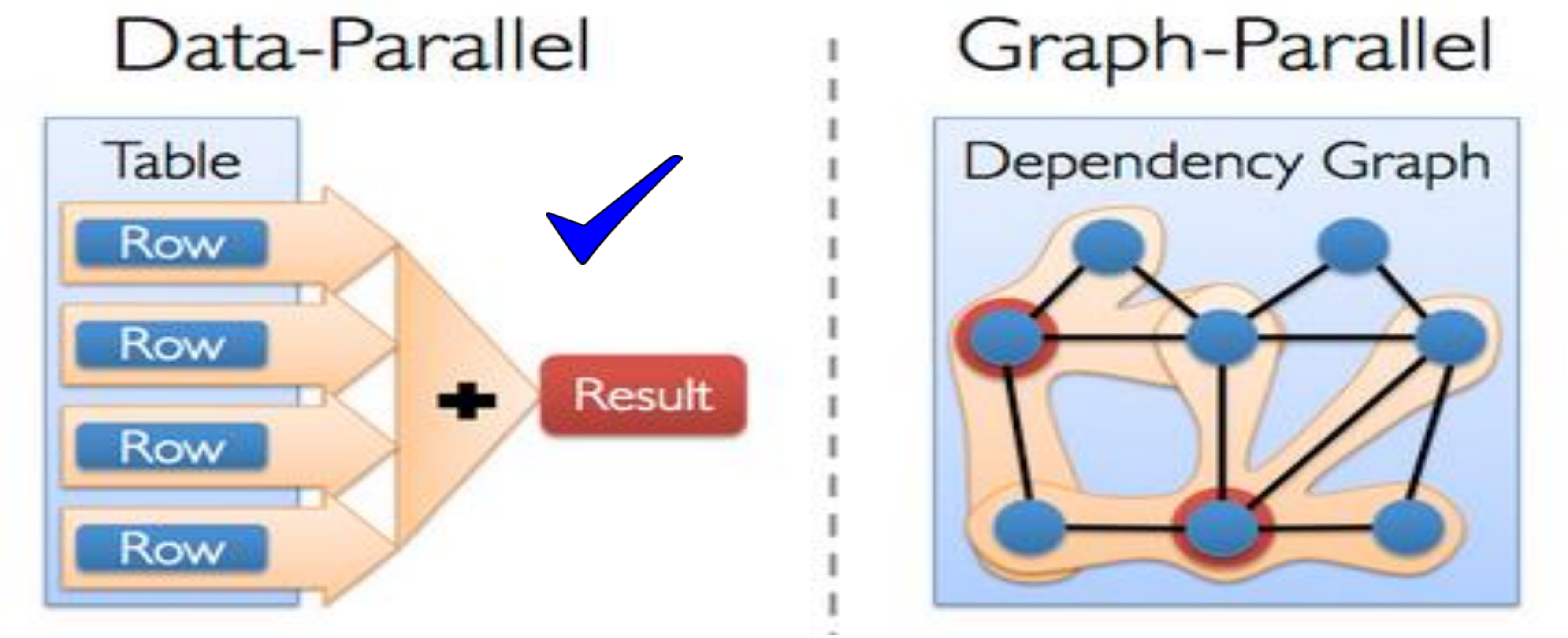
Graph Computation Challenges

1. Graph algorithms (BFS, Shortest path)
2. Query on connectivity (Triangle, pattern)
3. Structure (Community, Centrality)
4. ML & Optimisation (Regression, SGD)

- Data driven computation: dictated by graph's structure and parallelism based on partitioning is difficult
- Poor locality: graph can represent relationships between irregular entries and access patterns tend to have little locality
- High data access to computation ratio: graph algorithms are often based on exploring graph structure leading to a large access rate to computation ratio

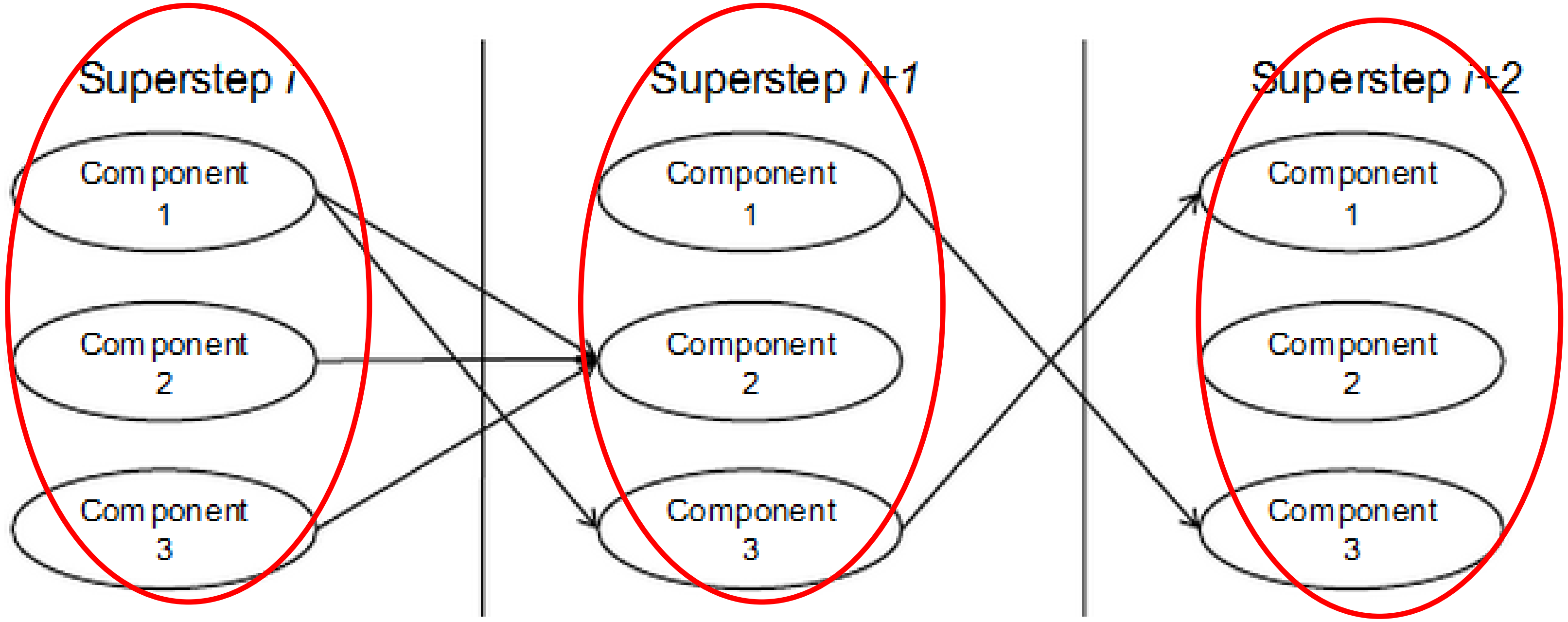
Data-Parallel vs Graph-Parallel

- Graph Parallel (Graph Specific Data Parallel)
  - BSP: Pregel, Giraph, Graphlab
  - Unifying graph- & data-parallel: GraphX/Spark
  - Data-flow programming: NAIAD, DryadLINQ



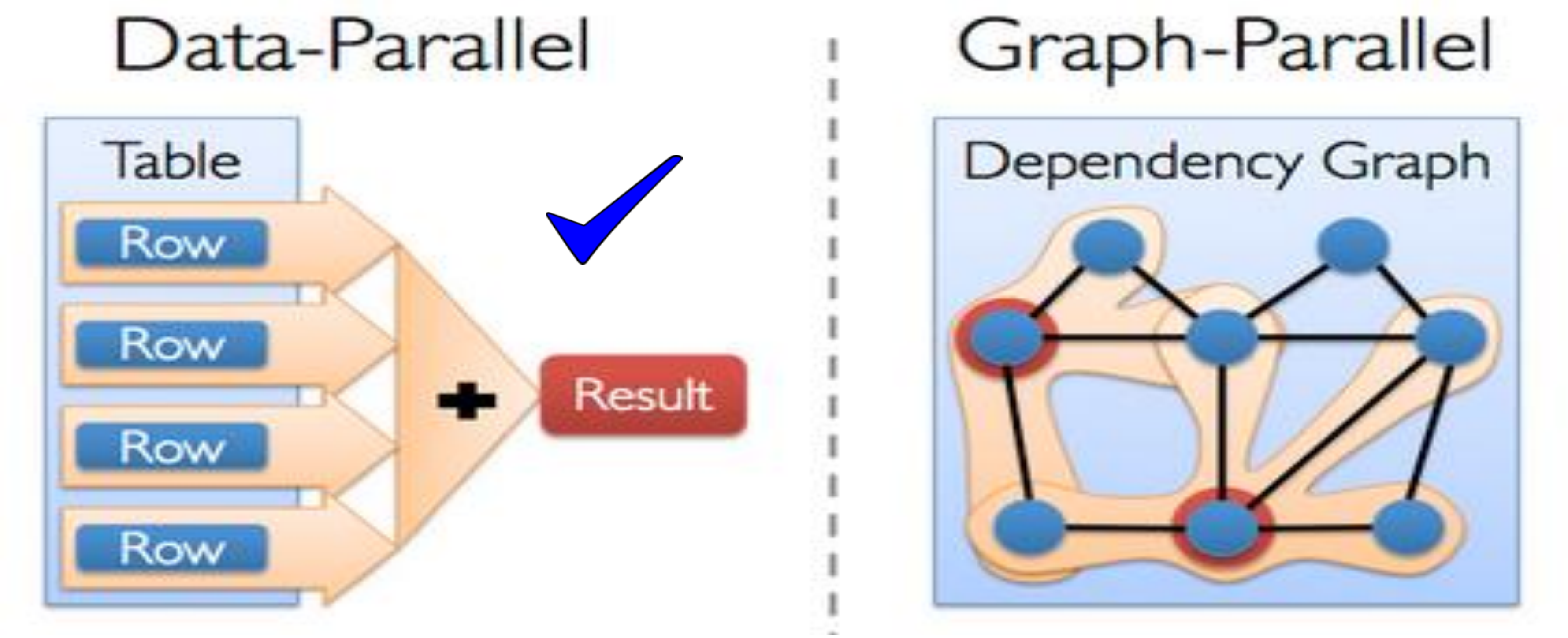
Bulk Synchronous Parallel Model

- Computation is sequence of iterations
- Each iteration is called a super-step
- Computation at each vertex in parallel



Data-Parallel vs Graph-Parallel

- Graph Parallel (Graph Specific Data Parallel)
  - BSP: Pregel, Giraph, Graphlab
  - Unifying graph- & data-parallel: GraphX/Spark
  - Data-flow programming: NAIAD, DryadLINQ

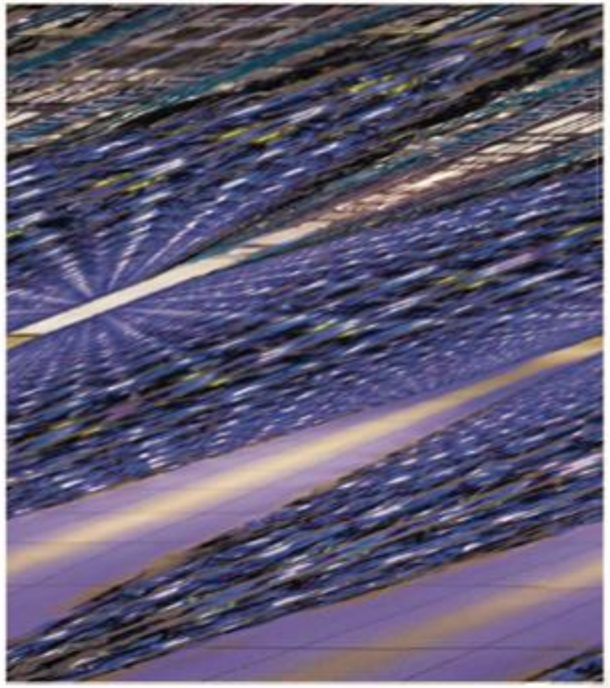


Are Large Clusters and Many-cores Efficient?

- Brute force approach efficiently works?
  - Increase of number of cores (including use of GPU)
  - Increase of nodes in clusters

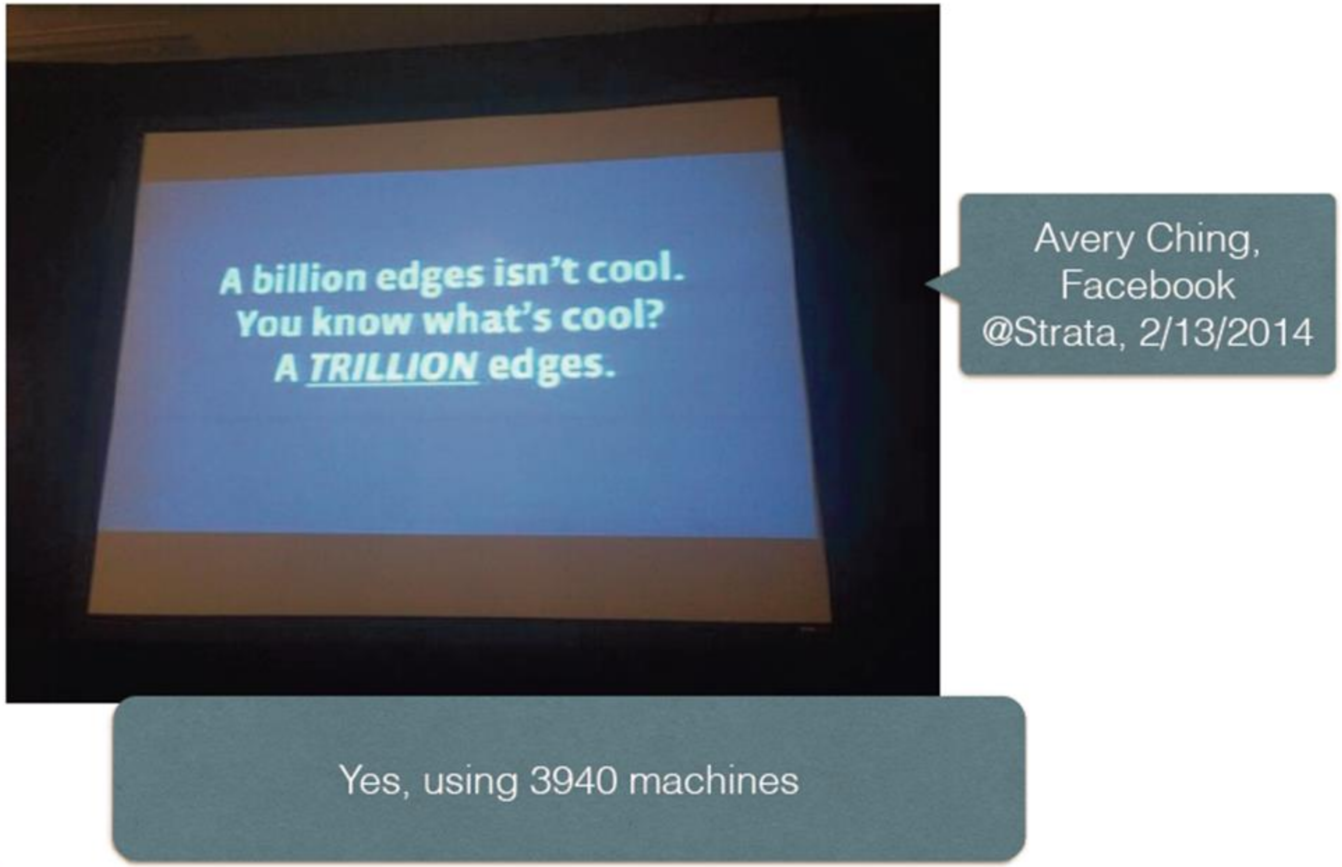
Big Iron

Large Cluster

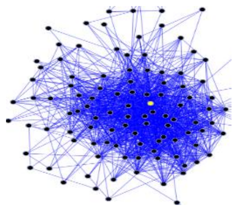


Are Large Clusters and Many-cores Efficient?

- Brute force approach efficiently works?
  - Increase
  - Increase



Do we really need large clusters?



- Laptops are sufficient?

Twenty pagerank iterations

System	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Single thread	1	300s	651s

Label propagation to fixed-point (graph connectivity)

System	cores	twitter_rv	uk_2007_05
Spark	128	1784s	8000s+
Giraph	128	200s	8000s+
GraphLab	128	242s	714s
GraphX	128	251s	800s
Single thread	1	153s	417s

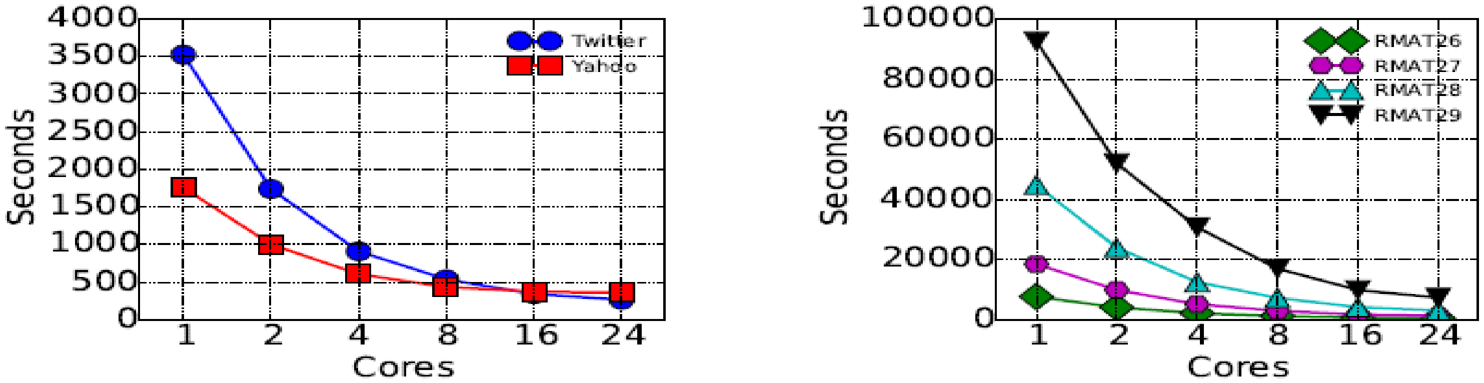
Fixed-point iteration: All vertices active in each iteration (50% computation, 50% communication)

Traversal: Search proceeds in a frontier (90% computation, 10% communication)

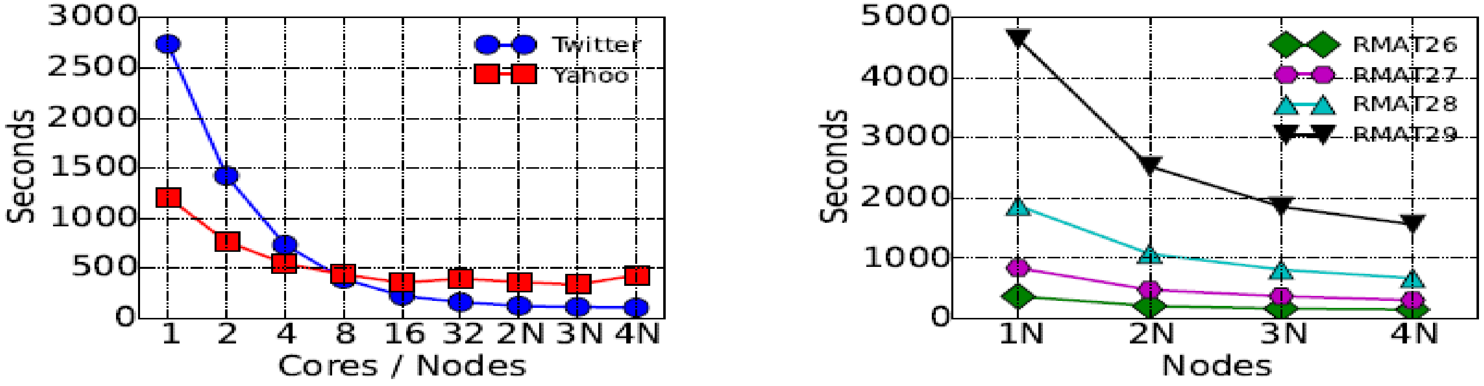
from Frank McSherry HotOS 2015

Do we really need large clusters?

- PTDL (Triangle Listing): More cores/nodes increases overhead



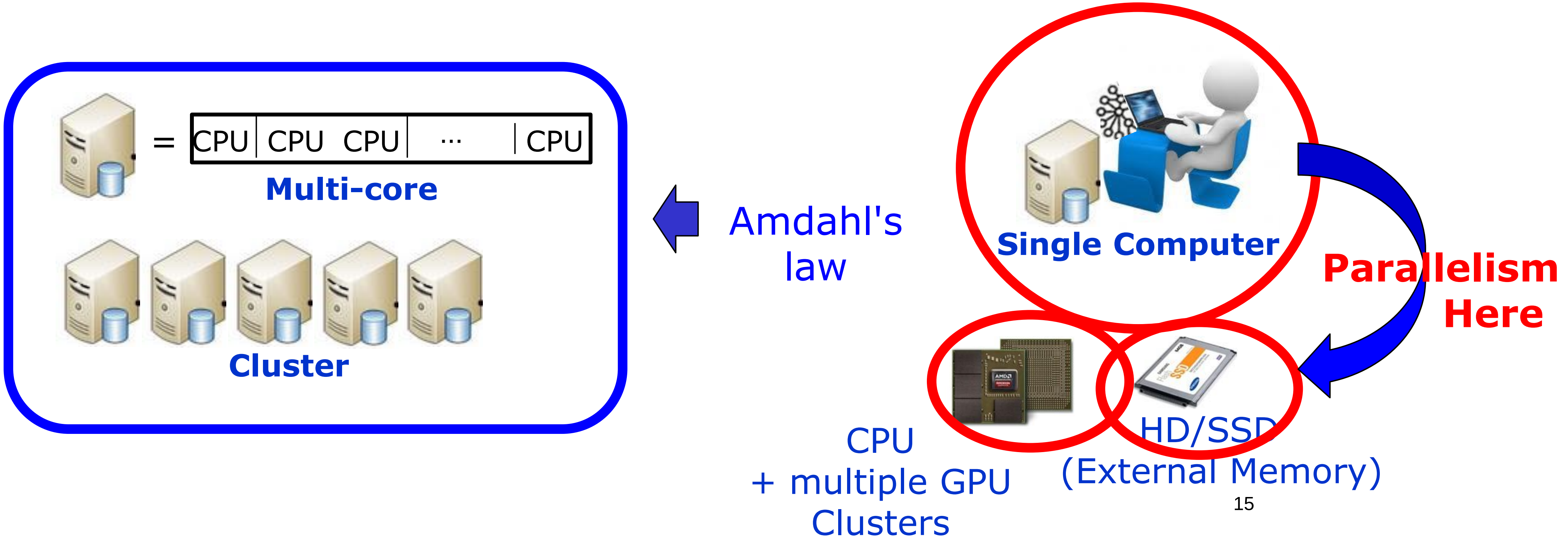
PDTL in Local Multicore: Total Time



PDTL in EC2: Total Time

Bring Massive Data Processing to Single Computers

- Use of powerful HW/SW parallelism
  - SSDs as external memory
  - CPU/GPU integrated heterogeneous many core architecture
- Open up massive graph processing to everyone



Storage Centric View

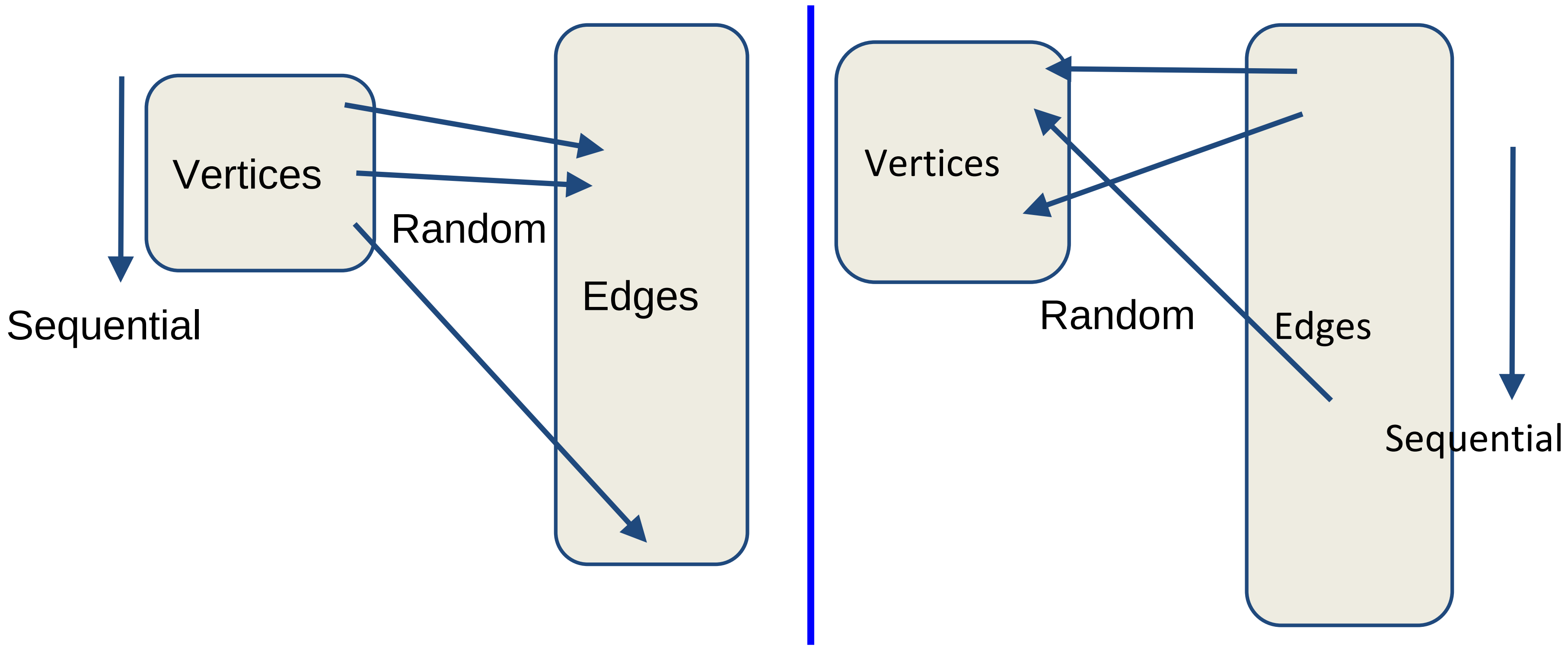
- Lot of work on computation
- Little attention to storage
  - Store LARGE amount of graph structure data (majority of data is edges)
  - Efficiently move it to computation (algorithm)

Potential solutions:

- Cost effective but efficient storage
  - Move to SSDs (or HD) from RAM
- Reduce latency
  - Runtime prefetching
  - Streaming (edge centric approach)
- Reduce storage requirements
  - Compressed Adjacency Lists

Vertex/Edge Centric Access

- Vertex centric access is random
- Edge centric access is more sequential



PrefEdge and X-Stream

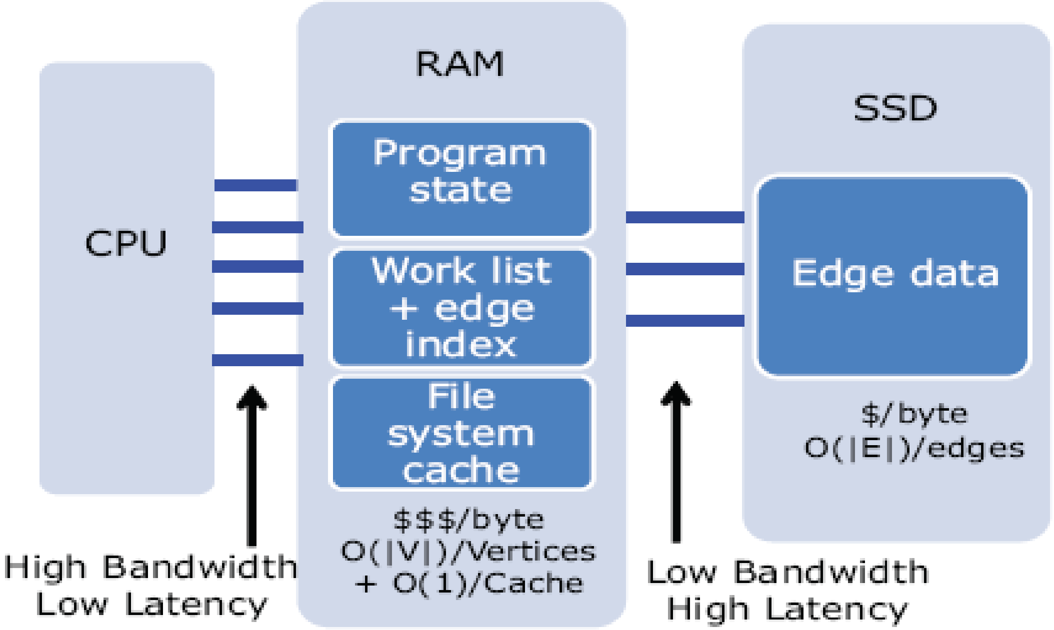
- Storage-Centric: 2 different ways to access graph structured data
  - Batch processing of large graphs on single machine
  - Establish useful limits for single machine processing
  - Directly address storage bottlenecks

PrefEdge: Accelerates random access using a novel prefetcher by Cambridge

X-Stream: Sequentially streaming a large set of (potentially unrelated) edges by EPFL

PrefEdge

- Simplest possible abstraction
  - One machine (low cost)
  - Most of graph on SSD (low cost)
  - Synchronous I/O

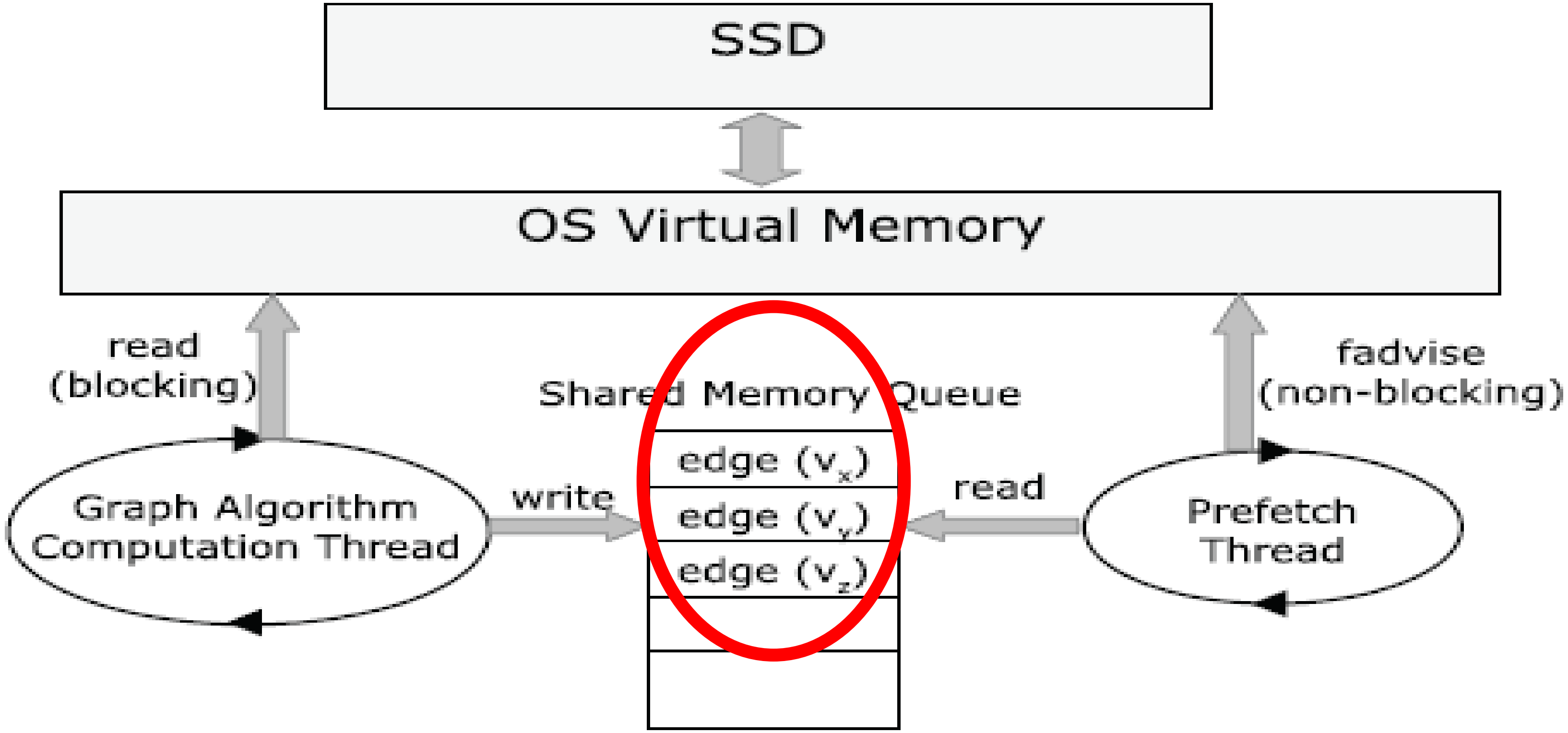


- Traverse graph (BFS, SSSP)

- Conventional wisdom is that this will never work
  - Graphs have no locality
  - Every traversed edge will miss the main memory cache
  - Single threaded synchronous I/O will kill performance

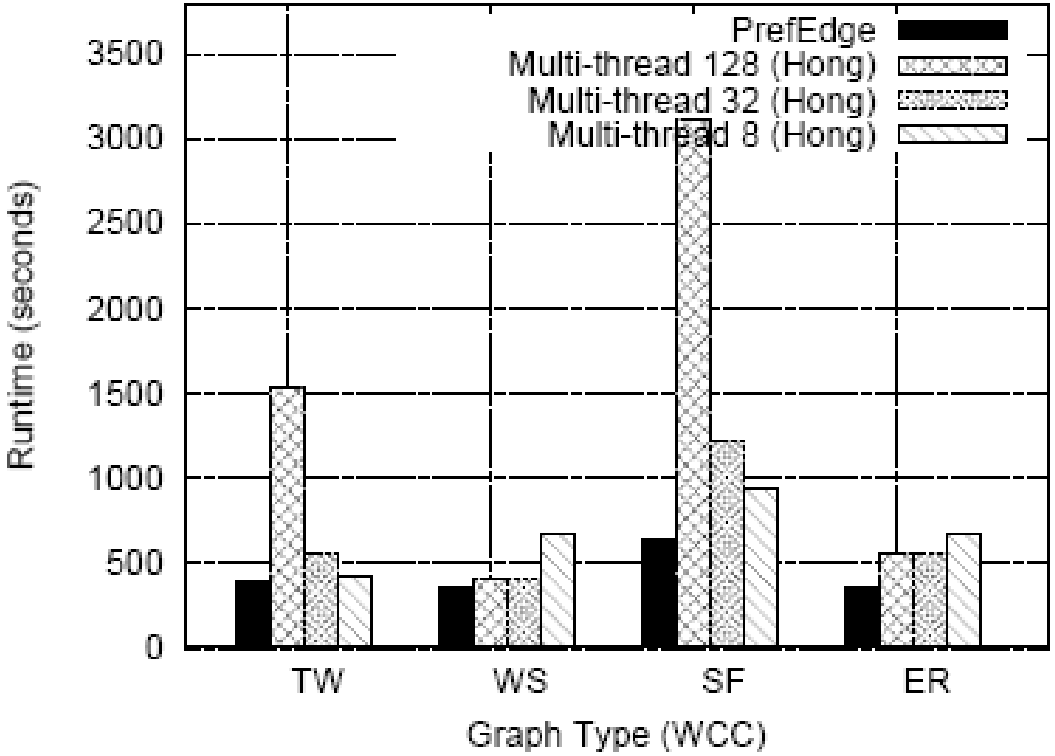
SSD Prefetcher for Large-Scale Graph Traversal

- Perform asynchronous prefetching: Mitigates I/O latency and maximises throughput → allow graph traversal to keep queue sufficiently deep
- Decouple CPU and I/O-level parallelism (advantage of embedded SSD parallelism): can compete with multi-threaded approach



PrefEdge: Comparison with Multi-threading

- Faster than multi-threaded implementation
- With only 2GB RAM, no multi-threading in graph computation, simple programming, use of embedded parallelism in SSD random access



With Twitter Data (~40M vertices)

Algorithm	Baseline / PrefEdge	PrefEdge / In-memory
WCC	5.67x	2.74x
SSSP	10.10x	4.82x
PR	2.29x	1.11x
SCC	6.63x	2.11x
K-CORES	5.47x	1.42x

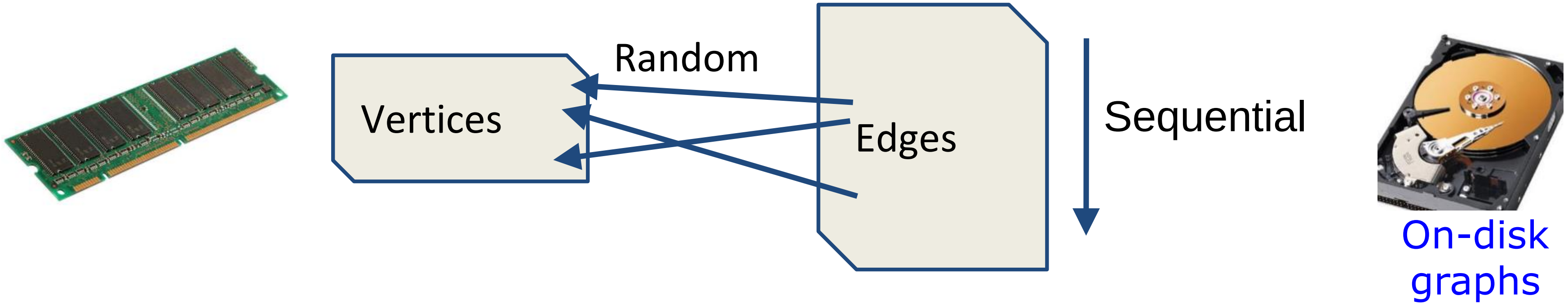
Random Access vs Sequential Access

Random access is inefficient for storage

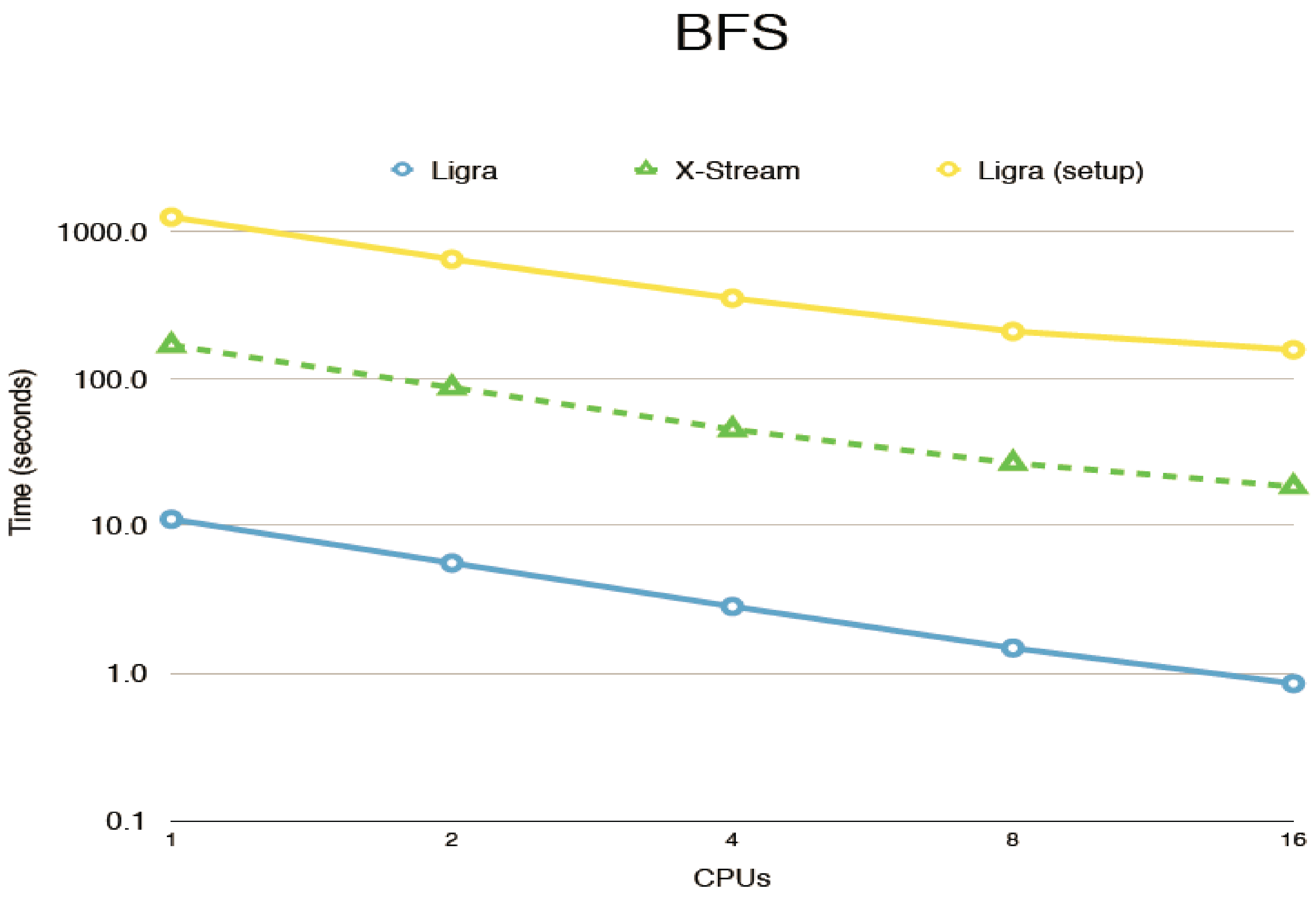
- Disk (500X slower)
- SSD (20X slower)
- RAM (2X slower)

X-Stream: Streaming Partitions

- Sequential access to any medium
- Eliminate random access to edges
- Ensure randomly accessed vertices held in cache
- Stream Partition
  - A subset of the vertices that fits in RAM
  - All edges whose source vertex is in that subset
- Reorganize computation to stream edges



Comparison with Ligra (HPC memory based)



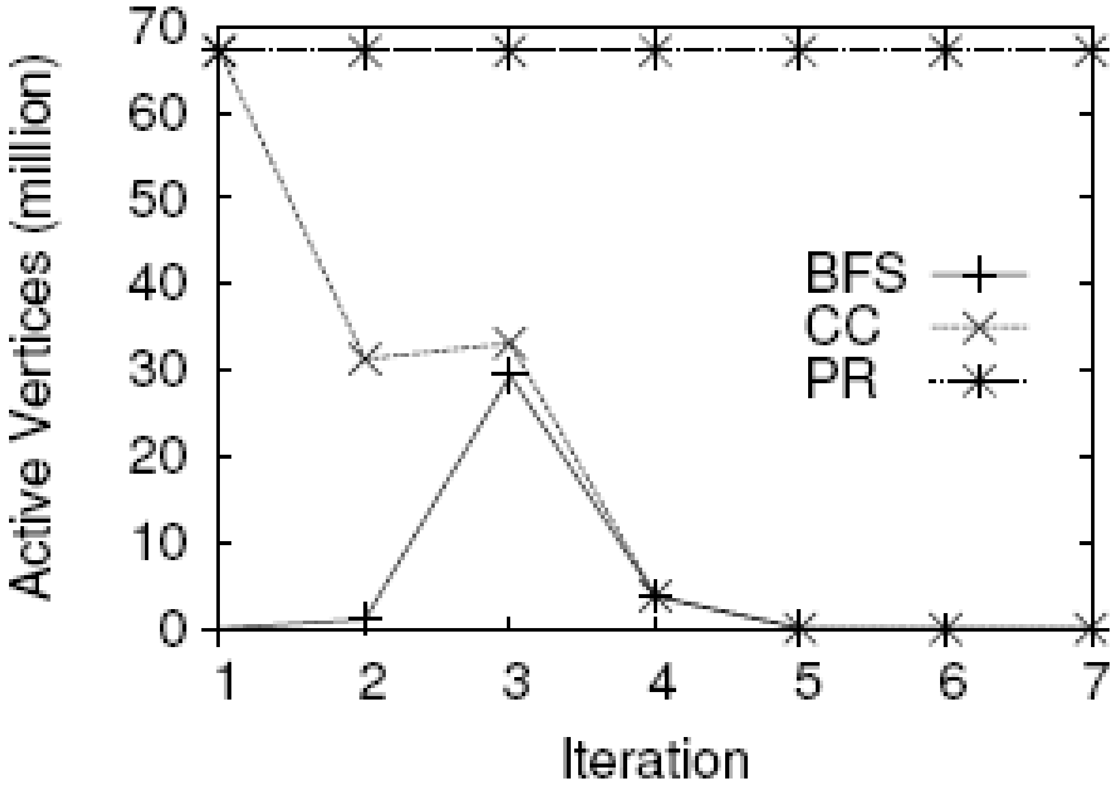
Pros and Cons

- PrefEdge clearly provides impressive speedup
  - Improving inefficiency of random access by prefetching
  - Limitation
    - Focus on traversal based graph computation
- X-Stream takes advantage of sequential access
  - Single building block of streaming partitions
    - Works well with RAM, SSD, and Magnetic Disk
  - Limitation
    - A large number of potentially unrelated edges

Hybrid Approach

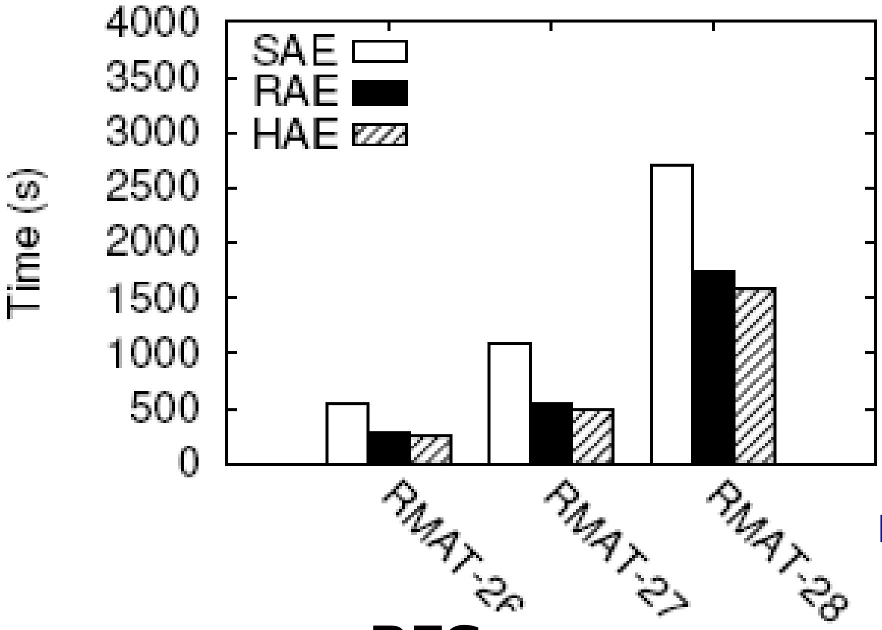
- Allow streaming partitions to sort their associated edges and access them randomly
  - Starting point is X-stream style streaming
  - Low utilisation of edges due to few active vertices triggers index building
  - Switch to PrefEdge style prefetching after index is available
- PrefEdge mitigates limitations of X-Stream
  - Wasted edges due to inactive vertices
  - Particular problem for high diameter graphs

Number of Active Vertices



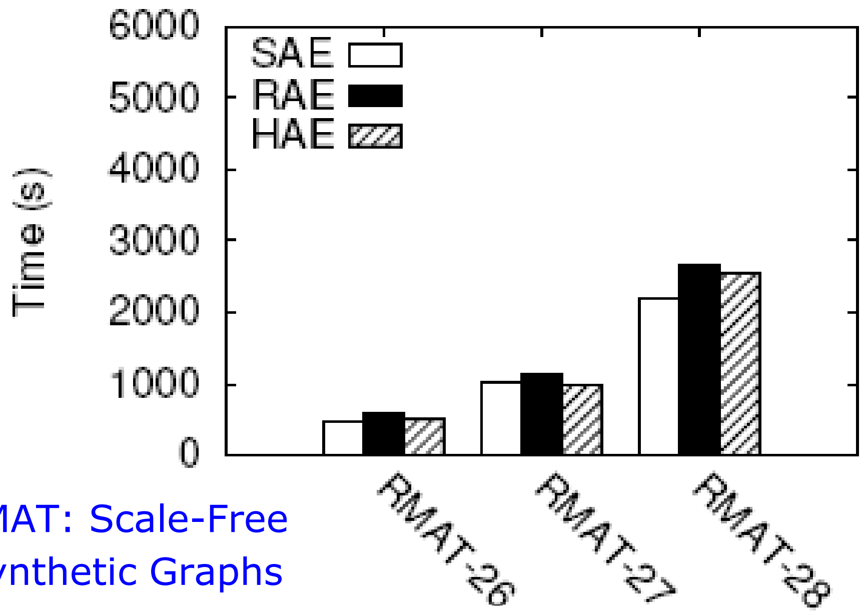
Algorithm Comparison

- Traversal algorithms: good with RAE (Random Access Edges) while PR (fix-point iteration type of operation) with SAE (Sequential Access Edges) more efficient



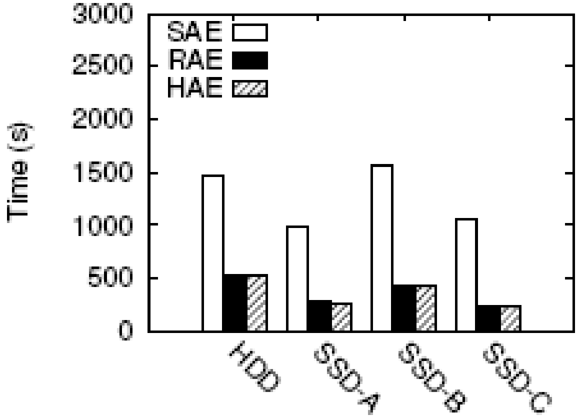
BFS

RMAT: Scale-Free synthetic Graphs

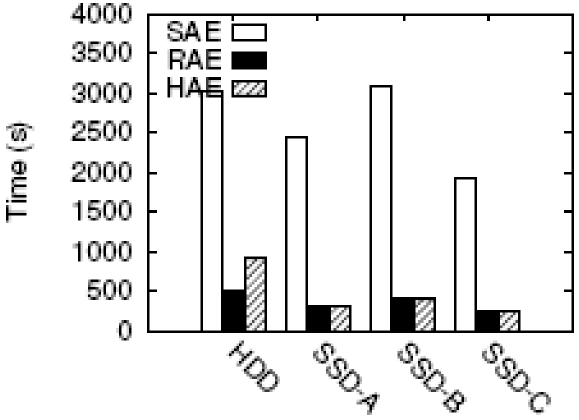


PR

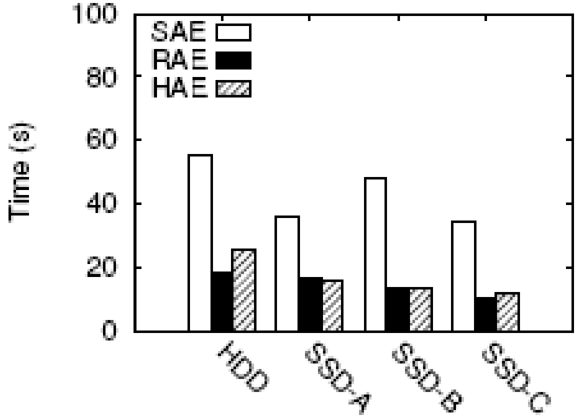
Real World Graph



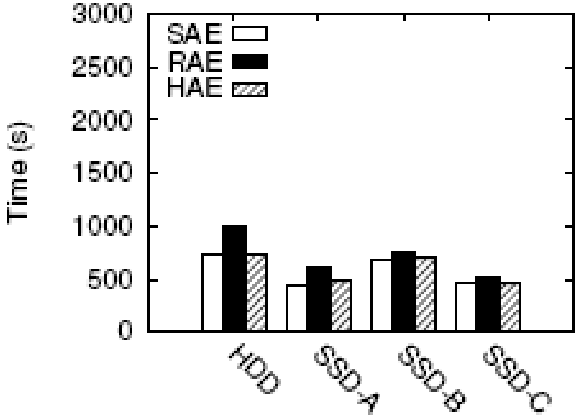
Twitter, BFS



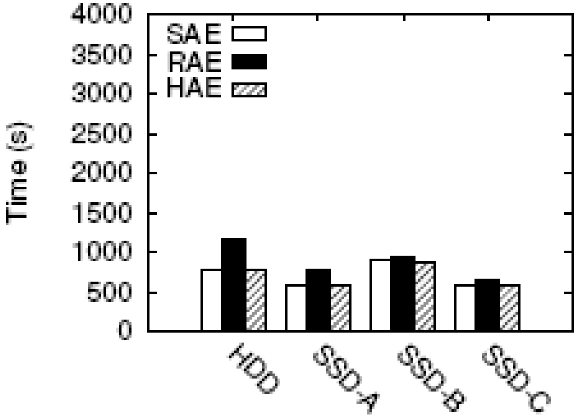
SK-2005, BFS



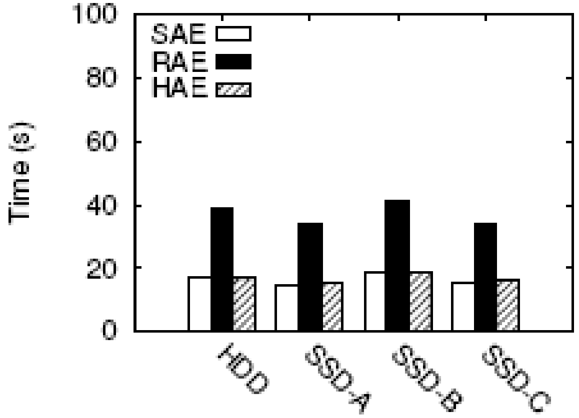
Netflix, BFS



Twitter PR



SK-2005 PR



Netflix PR

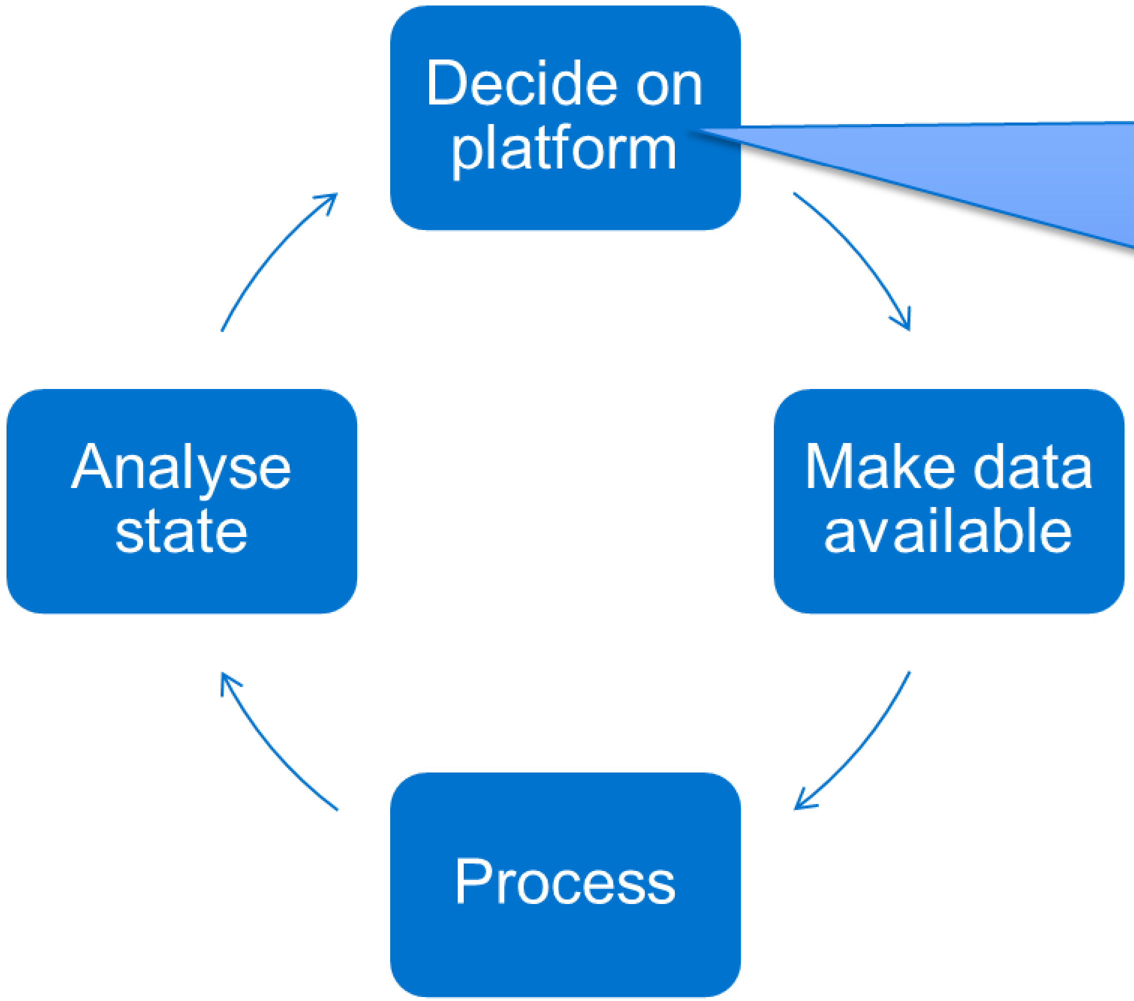
Graph Processing and GPU

Challenge	GPU Constraints
Large-scale data	• Limited capacity local memory • DMA bottleneck
Irregular programs	• SIMD (*Single instruction, multiple data*) thread model
Skewed workload	• Thread divergence = serialisation

- These factors mean that the correct platform to use may be both **program- and data-dependent**.

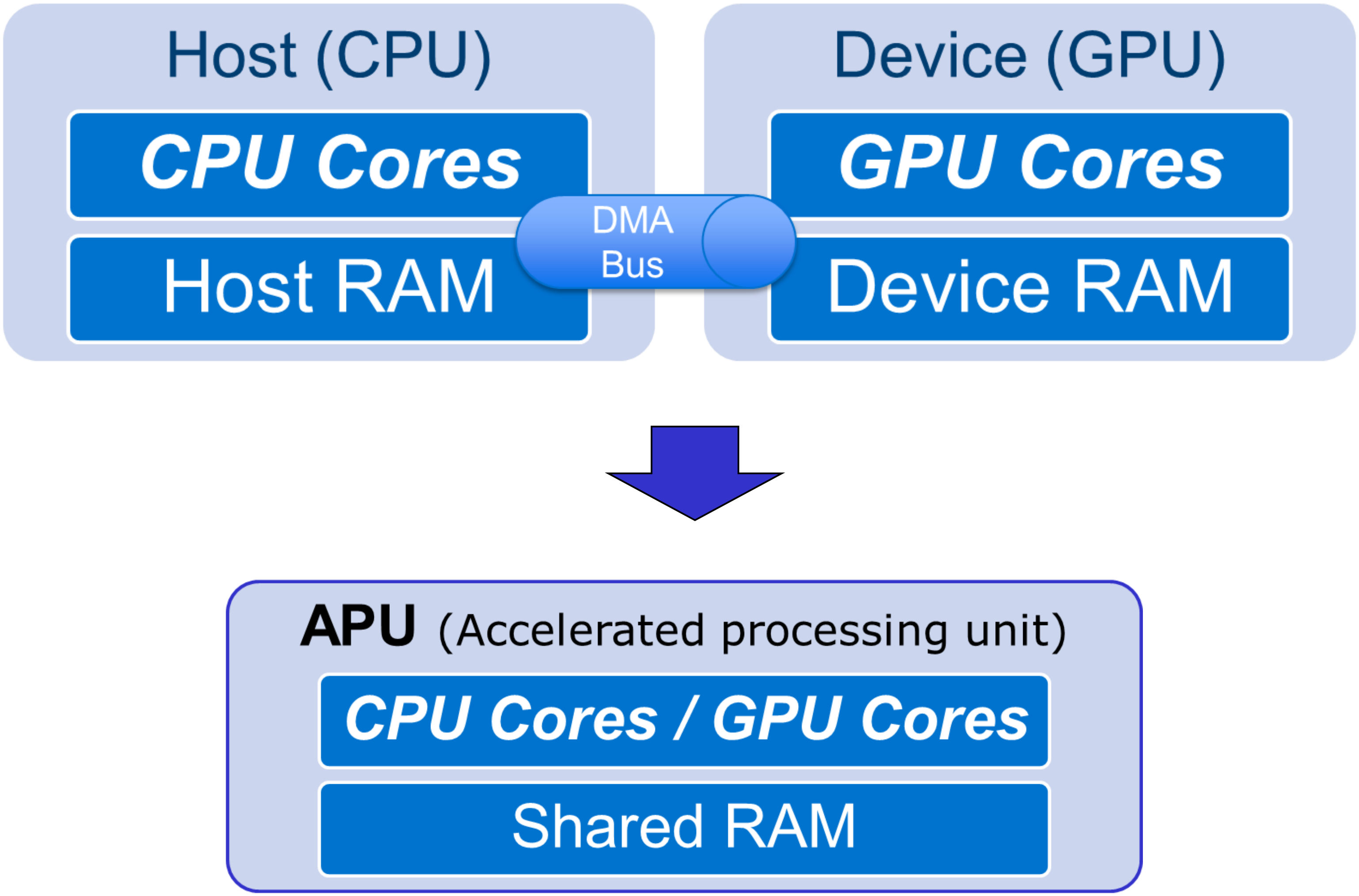
Heterogeneous Operation

- Existing heterogeneous operation over CPU/GPU



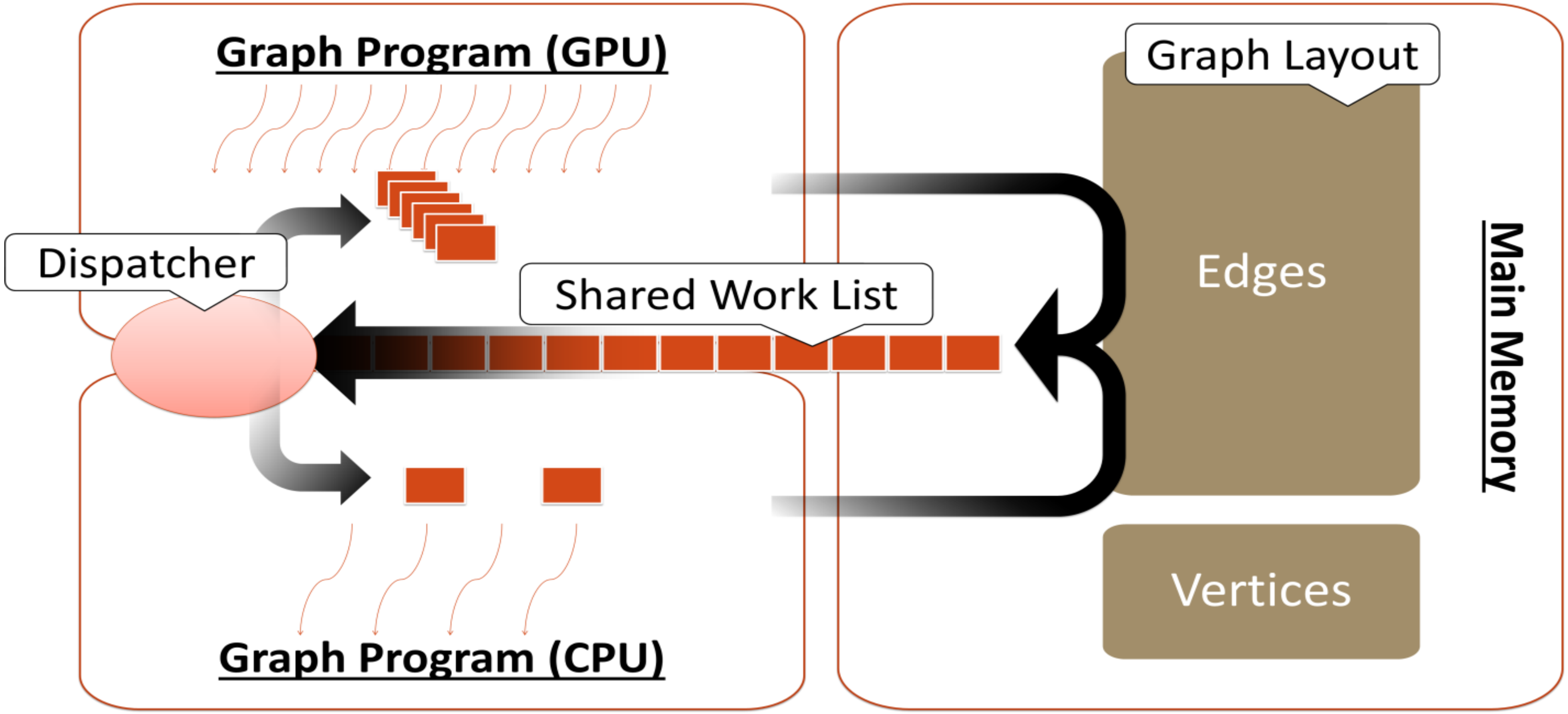
- **Temporally partition** the program, e.g. Hong et al. (2011)
- **Spatially partition** the data, e.g. Medusa and Totem

Integrated GPU



Dynamic Scheduling to CPU/GPU

- Work-list abstraction ensures only active tasks are dispatched to the GPU
- Use graph topology information (e.g. degree) for scheduling

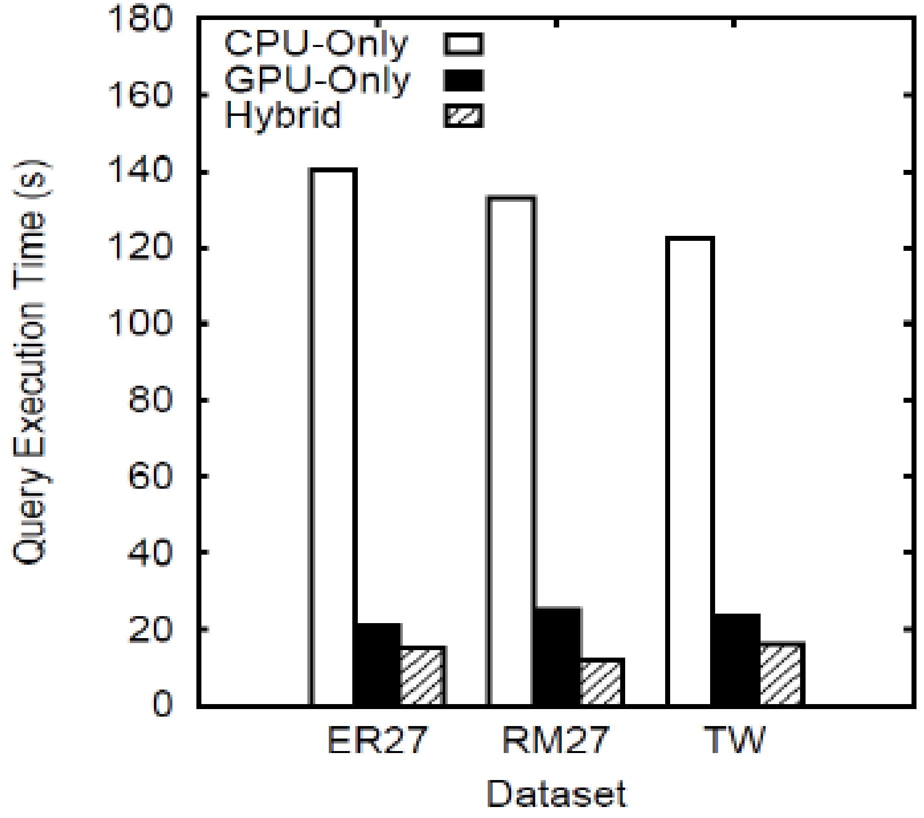


Preliminary Results

- Hybrid vs CPU-only: ~7x faster
- Hybrid vs GPU-only: 1.2 x faster

- Stable across synthetic and real data, with multiple queries running concurrently

- Optimisation to improve memory access
- Auto adjustment of scheduling criteria



Conclusions

- Algorithms, S/W and H/W for mainstream parallel approaches are not effective for more complex structured data from real world
- Data and algorithms dictate complex and irregular graph data processing: Utilise systems' parallelisms and resource coordination - no burden for algorithm implementation itself

- Massive graph processing on single computer
- Exploit different parallelism at different scales
- Current project: General auto-tuning and scheduling optimisation using structural Bayesian Optimisation for computer systems

Conclusions

- Algorithms, S/W and H/W for mainstream parallel approaches are not effective for more complex structured data from real world
- Data and algorithms dictate complex and irregular graph data processing: Utilise systems' parallelisms and resource coordination - no burden for algorithm implementation itself

- Massive graph processing on single computer
- Exploit different parallelism at different scales
- Current project: General auto-tuning and scheduling optimisation using structural Bayesian Optimisation for computer systems

Thank you!