Streaming Graph Analytics: Complexity, Scalability, and Architectures

Peter M. Kogge
McCourtney Prof. of CSE
Univ. of Notre Dame
IBM Fellow (retired)

Outline

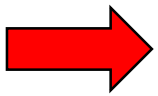
- Motivation and Current Kernels
- Real World Graph Streaming
- Sparsity and Locality
- Using Today's Parallel Architectures
- Emerging Architectures

Motivation: Today's Architectures Don't Match Tomorrow's Apps

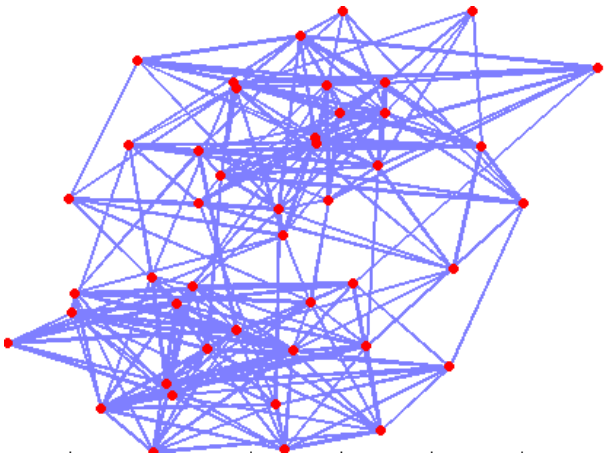
- **Yesterday's Apps**: Dense linear algebra **(HPL)**
  - N^2 operands from memory enough for N^3 flops
  - Big caches & wide memory permit ~100% utilization of many FPUs
- **Today's Apps**: Sparse (but local) Linear Algebra **(HPCG)**
  - Performance now proportional to memory bandwidth
  - Lots of FPUs – largely worthless
- **Emerging's Apps**: Truly Randomly Sparse **(BFS, ML)**
  - Very few flops, mostly "random" accesses
  - Small object size & irregular paths negate caches & wide paths
- These are all "batch"
- **Irregular Memory Accesses is the *Real* Problem!**
- **And this exhibits itself in scalability**

Tomorrow's Apps

- ***Massive persistent*** data sets (petabytes)
  - Very sparse and very random
  - Both "Table" and "Graph," with 1,000s of properties
  - Must be managed independent of specific apps
- ***Multiple apps*** in play at same time
- ***Wide spectrum*** of computational patterns
  - **Batch**: perform specific processing over whole set
  - **Concurrent**: many localized processing against specific subsets
  - **Streaming**: stream of updates flow into system, with need to update prior queries when necessary



Graphs



A Graph

- V = set of Vertices ("Entities")
- E = set of Edges ("Relationships")

Graph Characteristics

- Vertex Property = value assoc. with vertex
- Edge weight = value assoc. with edge
- Degree = # edges from a vertex
- Path = sequence of edges connecting 2 vertices
- Diameter = furthest distance apart

Typical Graph Operations

- Compute vertex properties
- Search vertex properties
- Follow an edge to a neighbor
- Determine a neighborhood
- Find a path
- Look at all paths
- Compute properties of graph

https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18

Kernel Identification Efforts

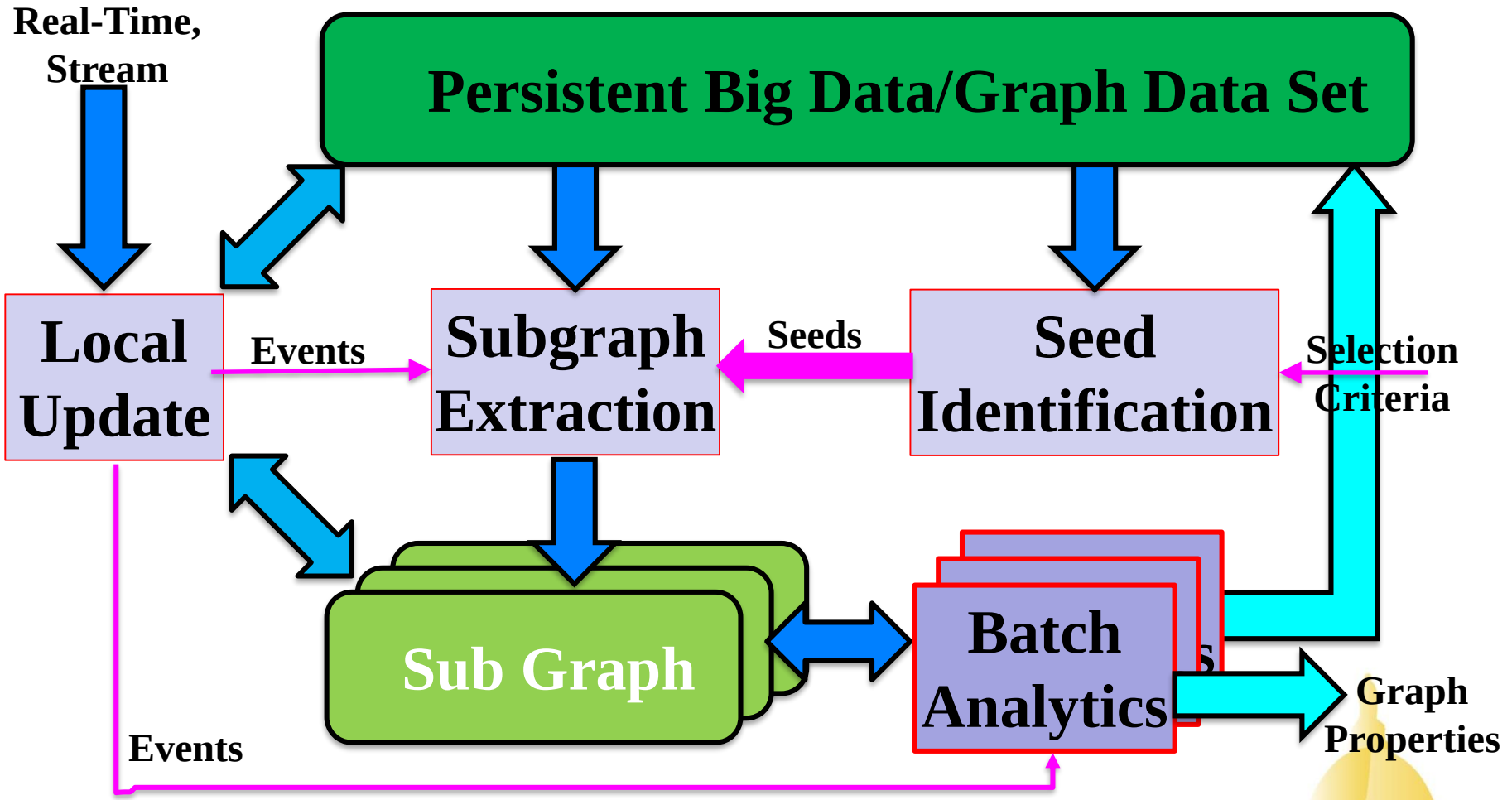
- GraphAnalysis: http://graphanalysis.org/benchmark/index.html
- GraphBLAS: http://www.graphblas.org/home/
- GraphChallenge: http://graphchallenge.org/
- Graph500: http://www.graph500.org/
- Firehose: http://firehose.sandia.gov/
- Mantevo: https://mantevo.org/
- Stinger: https://trac.research.cc.gatech.edu/graphs/wiki/STINGER
- VAST: http://vacommunity.org/HomePage
- Kepner & Gilbert: Graphs Linear Algebra

The Spectrum of Kernels

Kernel	Benchmarking Efforts											Outputs						
	Standalone	Firehose	Graph500	GraphBLAS	Graph Challenge	GAP	HPC Graph Analysis	Kepner & Gilbert	Matenvo	Stinger	Vast 2016	Graph Modification	Compute Vertex Property	Compute Edge Property	Output Global Value	Output O(1) EventsEvents	Output O(|V|) List	Output O($\|V\|^k$) List (k>1)
Anomaly - Fixed Key		S														X		
Anomaly - Unbounded key		S														X		
Anomaly - two-level key		S														X		
BC: Betweeness Centrality						B	B			S			X					
BFS: Breadth First Search			B			B	B	B		B			X				X	
Search for "largest"							B										X	
CC weak: Weakly Connected Component						B		B		S			X					
CC Strong: Strongly Connected Components								B										
Clustering Coefficients										S								
Community Detection										S								
Graph Contraction								B										
GTC: Global Triangle Counting						B									X			
Insert/Delete										S		X						
Jaccard	B/S																	X
MIS: Maximally Independent Set								B										
PageRank						B							X					
SSSP: Single Source Shortest Path						B/S							X				B/S	
All pairs Shortest Path								B										B
TL: Triangle Listing																		B
Geo & Temporal Correlation											S					X		

Real World Graph Processing and Streaming

Canonical Graph Processing Flow



Variations in Batch Analytics

- Search for a/all vertex with a particular property or neighborhood
- Explore region around some # of vertices
- Compute new property for each vertex
- Compute/output a property of a graph as a whole
- Compute/output a list of vertices and/or edges
- Compute/output a list of properties of all subgraphs with certain properties

Real World vs. Benchmarking?

- Many different classes of vertices
- Many different classes of edges
- Vertices may have 1000's of properties
- Edges may have timestamps
- Both batch & streaming are integrated
  - Batch to clean/process existing data sets, add properties
  - Streaming (today) to query graph
  - Streaming (tomorrow) to update graph
- "Neighborhoods" more important than full graph connectivity

Variations in Streaming Problems

"Streaming": Data arrives incrementally
  - With computations to be done incrementally
- Inputs specify vertex & insert/delete edges
  - Less often insert/delete vertex
- Inputs specify vertex & update properties
  - Look for changes in local graph parameters
  - Look for changes in global graph parameters
  - May involve search in neighborhood
- Inputs cause search for a matching vertex
  - And update properties

Stinger

- http://www.stingergraph.com/stinger/doxygen/index.php?id=introduction#whatdoesitdo
- Stinger: data structure for streaming apps from GaTech
- Suite of demo kernels implemented in it
  - Streaming edge insertions and deletions:
  - Streaming clustering coefficients:
  - Streaming connected components:
  - Streaming community detection:
  - Streaming Betweenness Centrality:
  - Parallel agglomerative clustering:
  - K-core Extraction:
  - Classic breadth-first search:

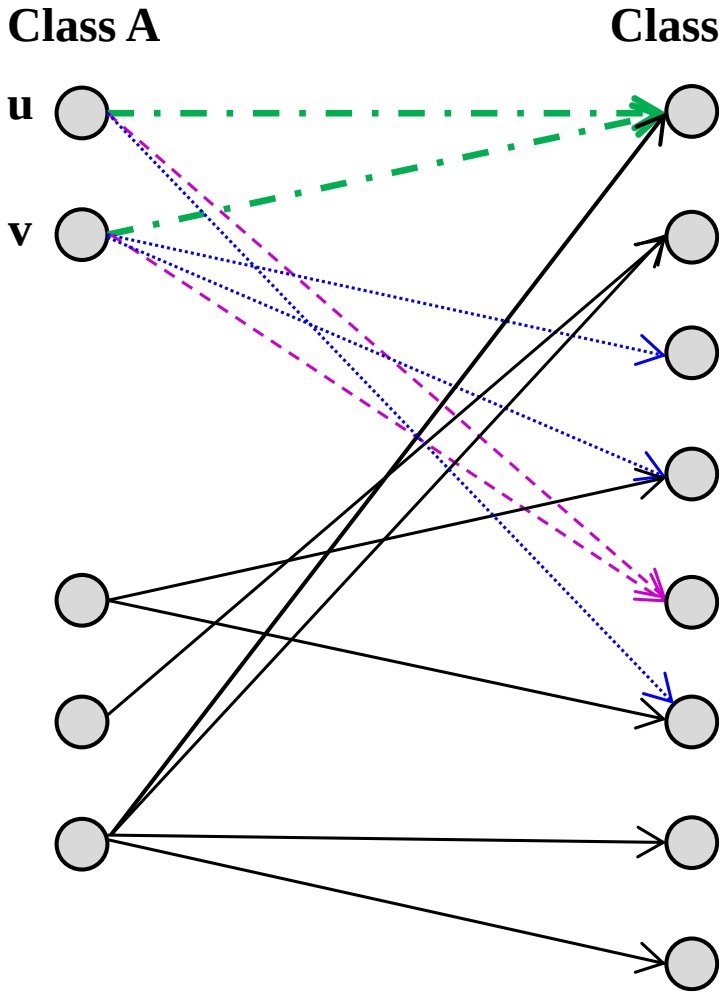
Centrality Metric Computation

- Look for "most important" vertices
  - **Degree**: # of edges each vertex associated with
  - **Closeness**: ave. shortest path to all other vertices
  - **Betweenness**: how often vertex is on shortest path between 2 other vertices
  - **Eigenvector**, **PageRank**, **Katz**: "influence" of a vertex
- Batch: compute metric for each vertex & report "N" largest
  - Output O(|V|) when optionally add/update property
- Stream: If add an edge, how does it change vertex metrics and "Top N"

Triangle Counting

- Metric: # of unique triangles in graph
- Batch mode variations
  - Count # in entire graph: O(1) output
  - Count # for each Vertex: O(|V|) outputs (Property)
  - List all triangles: up to $O(|V|^3)$
- Streaming: Add/Delete an edge
  - What is change in triangle count
  - What is change the associated vertices' counts
  - List all new triangles

Jaccard Coefficient Γ(u , v)



d(u) = # of neighbors of i

ɤ(u, v) = # of common neighbors

Γ(u,v) = fraction of all neighbors that are shared

Real world:

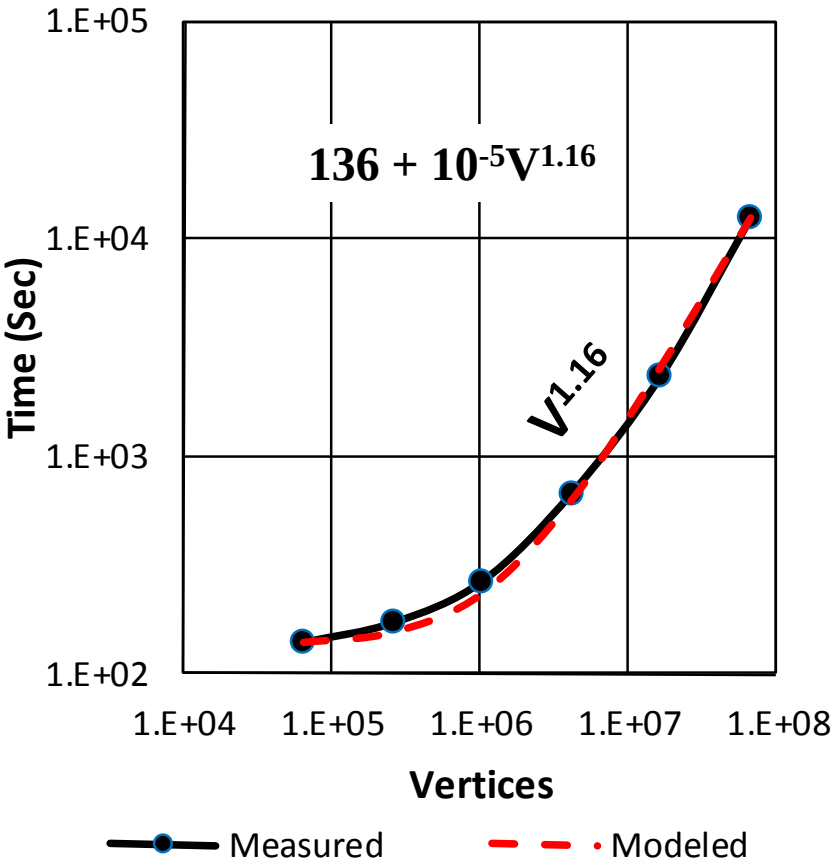
- "Weighting" of paths
- Thresholding of Γ
- Very often $|A|^K$, $k>1$
- "1.5" hops

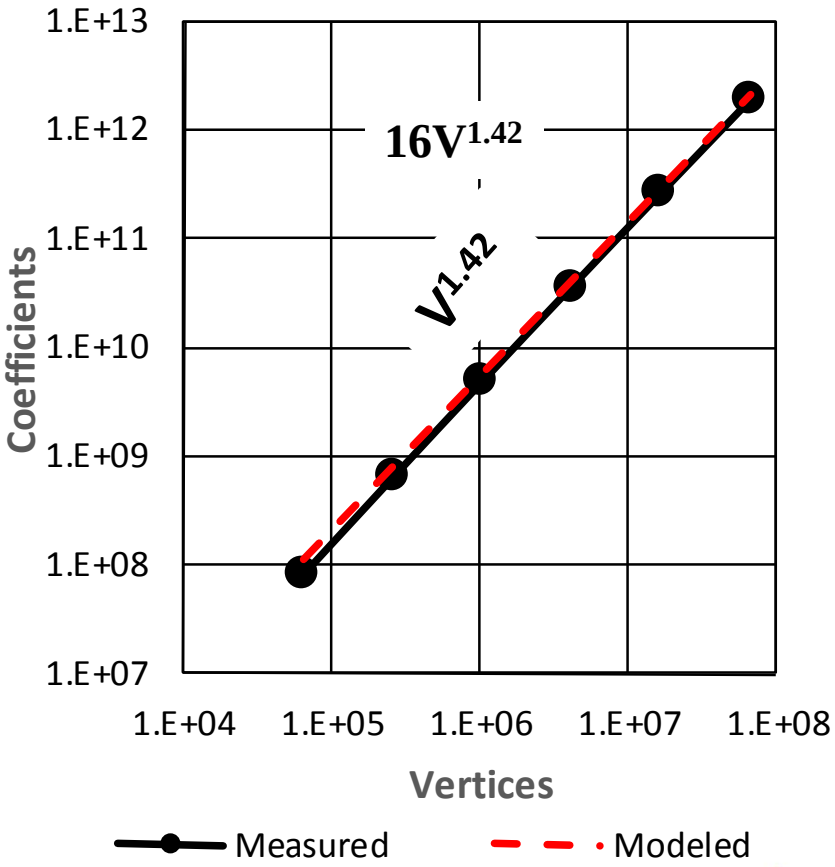
Green and **Purple** lead to common neighbors

Blue lead to non-common neighbors

Jaccard Batch Measurements

MapReduce on 1000 node system, each with 12 cores & 64GB





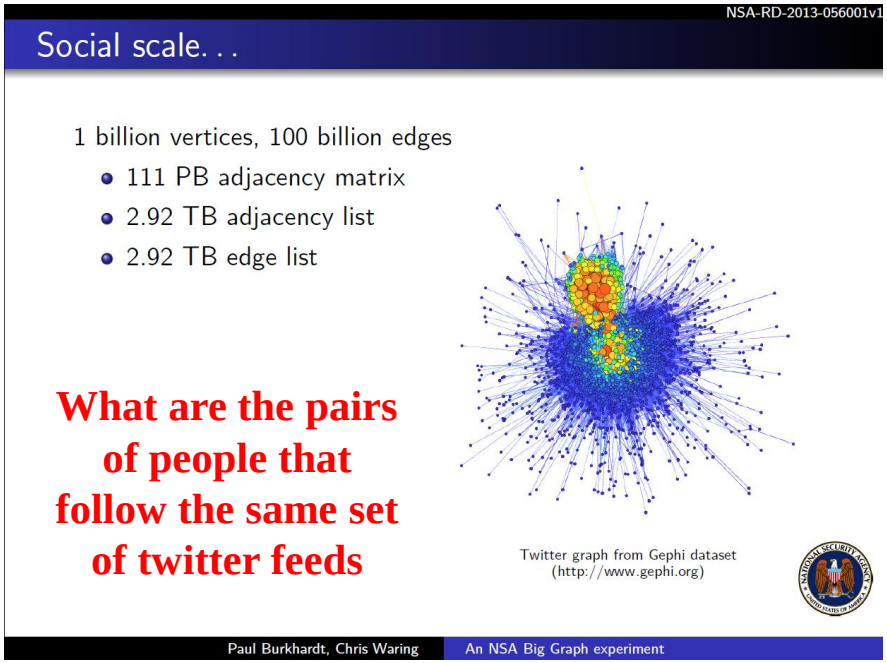
JACS (Jaccard Coefficients / Sec) = $1.6E6*V^{0.26}$

Burkhardt "Asking Hard Graph Questions," Beyond Watson Workshop, Feb. 2014.

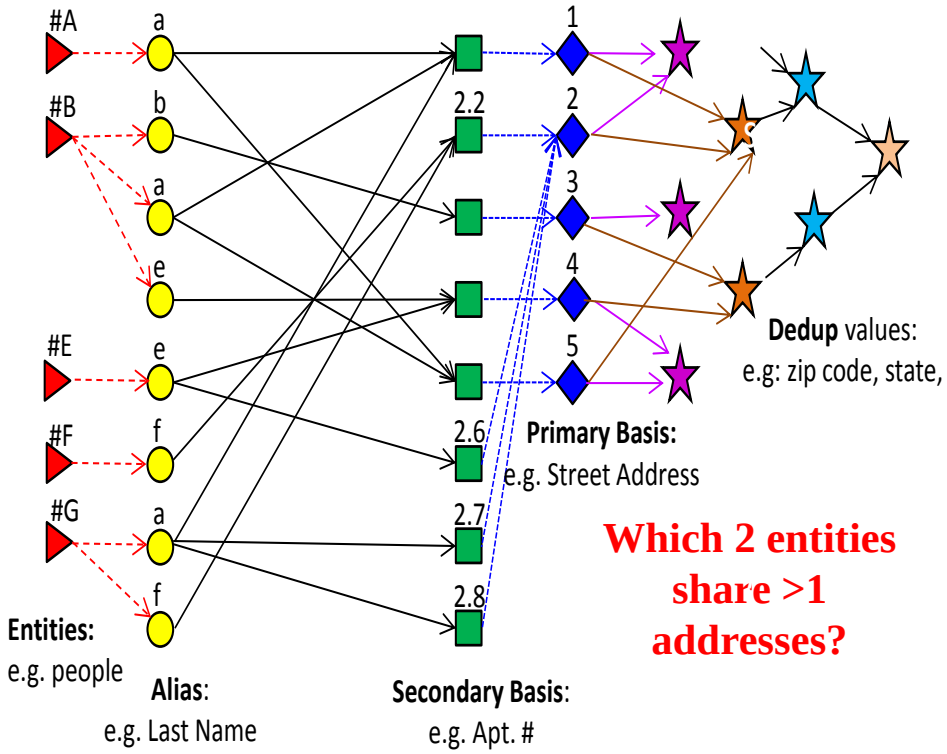
Jaccard Streaming

- Variants:
  - Query: for a particular vertex what's Γ (or γ)
  - Query: for a particular vertex, what are all other vertices, optionally have Γ (or γ) > some threshold
  - Edge insert: does this new edge affect any Jaccard #s
- Constraints: probably cannot afford to store $O(|V|^{1.42})$ coefficients
  - But could store "just statistics" : # non-zeros, largest Γ (and with which vertex), average Γ, etc
- Complexity of (re)computing on a new edge probably on $O(d^2)$ to $O(d^2 \log(d))$

Real World Relationship Problems



from "Burkhardt & Waring, An NSA Big Graph Experiment"
http://www.pdl.cmu.edu/SDI/2013/slides/big_graph_nsa_rd_2013_56002v1.pdf



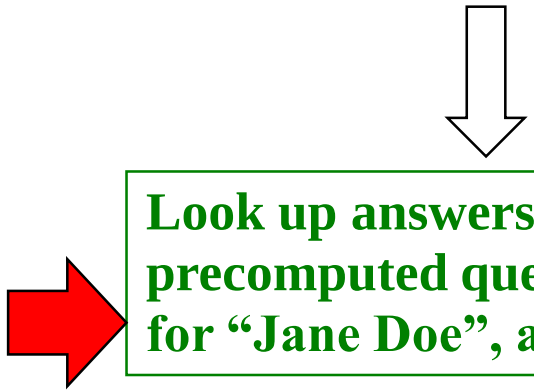
- Tough Problem: Find vertex ***pairs*** that "share some common property"
- Related graph problem: computing "Jaccard coefficients"
- Commercial version: "Non-obvious Relationship Problems" (NORA)

Real World Challenge Problem (From Lexis Nexis)

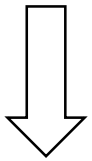
Auto Insurance Co: "Tell me about giving auto policy to Jane Doe" in < 0.1sec

- 40+ TB of Raw Data
- Periodically clean up & combine to 4-7 TB
- Weekly "Boil the Ocean" to precompute answers to all standard queries
  - Does X have financial difficulties?
  - Does X have legal problems?
  - Has X had significant driving problems?
  - Who has shared addresses with X?
  - Who has shared property ownership with X?

Relationships

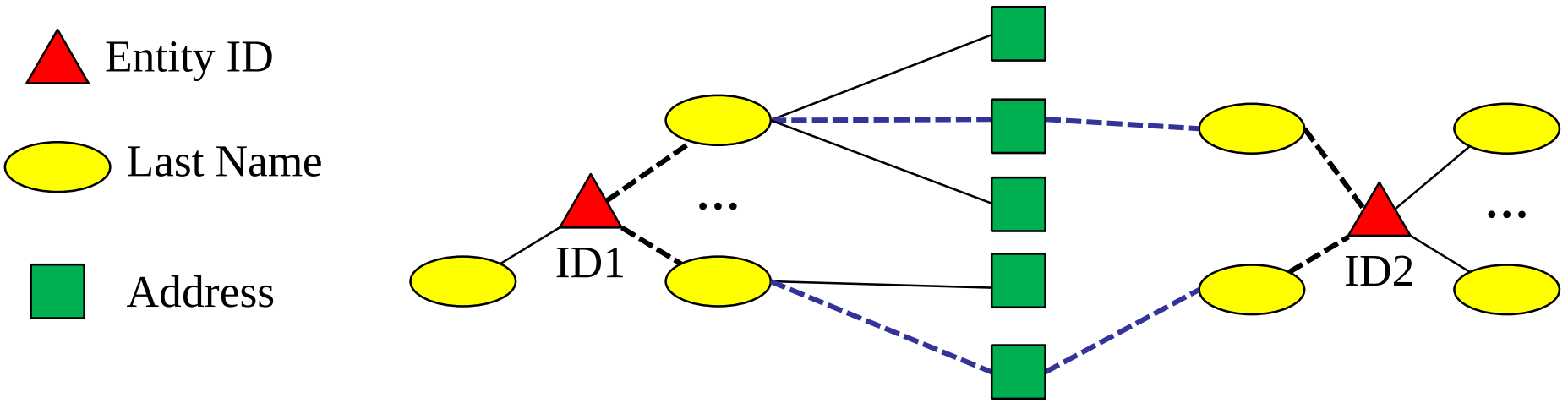


Look up answers to precomputed queries for "Jane Doe", and combine



"Jane Doe has no indicators
But
she has shared multiple addresses with Joe Scofflaw
Who has the following negative indicators …."

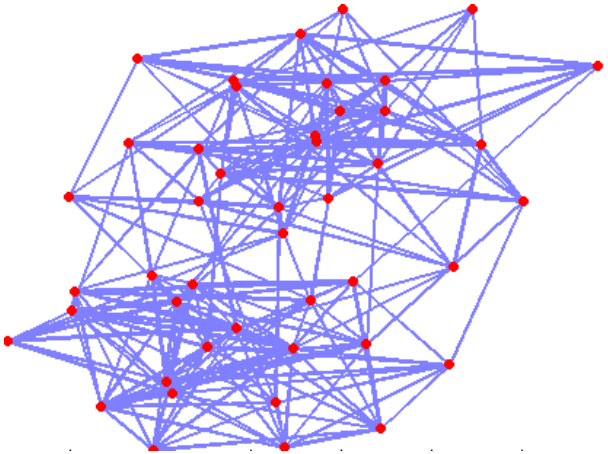
But We Want Real-Time



- Real time streaming of edge/vertex updates
- Streaming queries:
  - –Given specific ID1
  - –Find all ID2s with "relationships" passing some threshold

Sparsity and Locality

Sparsity in Graphs

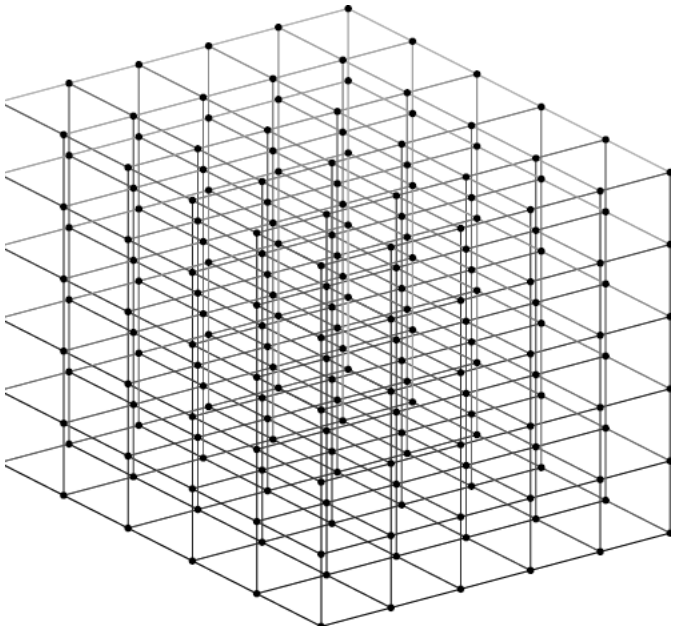


An Irregular Graph[1]

- **Degrees vary greatly**
- **Connectivity uneven**

Example Twitter set[2] (small):

- **11,316,811 vertices**
- **85,331,846 edges**
- **Average degree = 7.5**

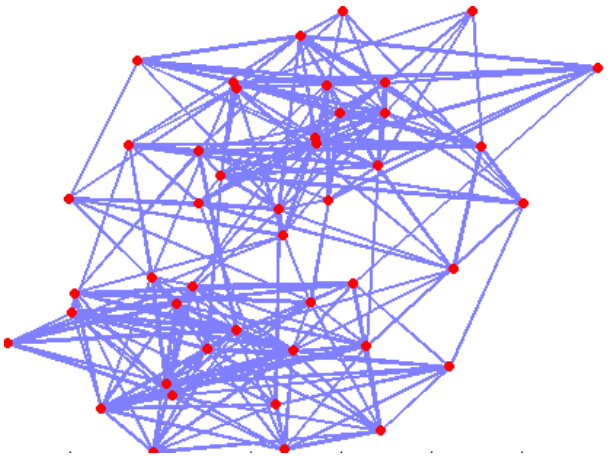


A Regular Graph

- **Each vertex has identically shaped neighborhood**
- **E,g.: each vertex above has 26 close-in neighbors**
- **Degree = 6**

1. https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18
2. http://socialcomputing.asu.edu/datasets/Twitter

Graphs as Tables



A Graph

- **V = set of Vertices (“Entities”)**
- **E = set of Edges (“Relationships”)**

Vertex ID	Property Columns				
v0					
...					
vi					
...					
vn					

(a) Table for each vertex class

Vertex	Vertex	Weight	Weight
...			

(b) Table for each edge class

Vertex	Key	Value
...		

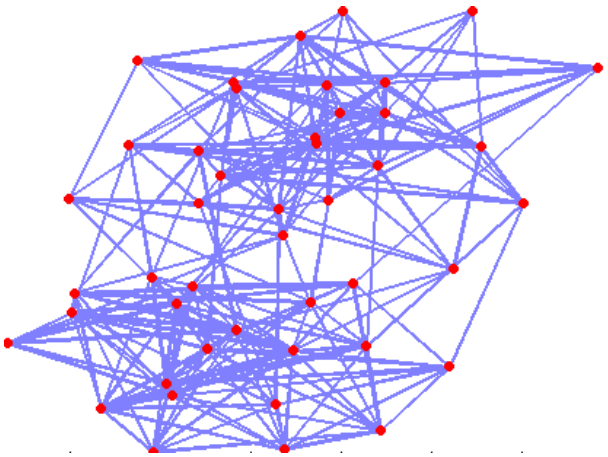
Triple Store

Key can be:

- **Vertex property**
- **Edge Type**

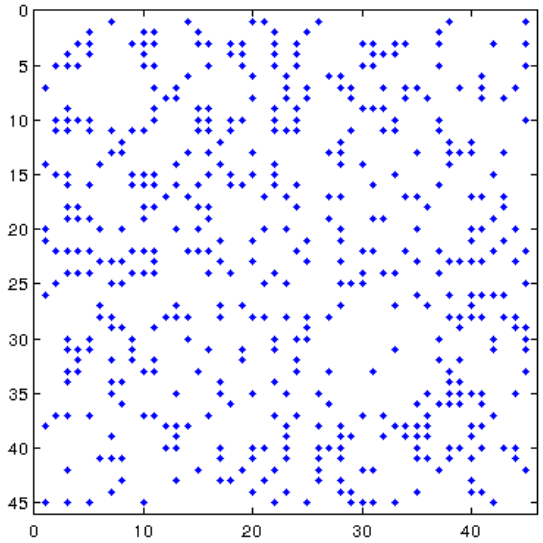
https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18

Graphs as Adjacency Matrices



A Graph

- V = set of Vertices
- E = set of edges
- Degree = # edges from a vertex
- Edge weight = # on edge
- Property = # assoc. with vertex



Adjacency Matrix

- A[i,j]=1 if edge between i & j
  May be edge weight
- NNZ = # non-zeros = # edges
- Degree = # NZ/row

Typical Linear Algebra Ops:

If

- A = bit matrix
- x = bit vector of vertices

Then

- Ax = Reachable in 1 hop
- A^2x = Reachable in 2 hops
- …

Weighted A computes paths

https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18

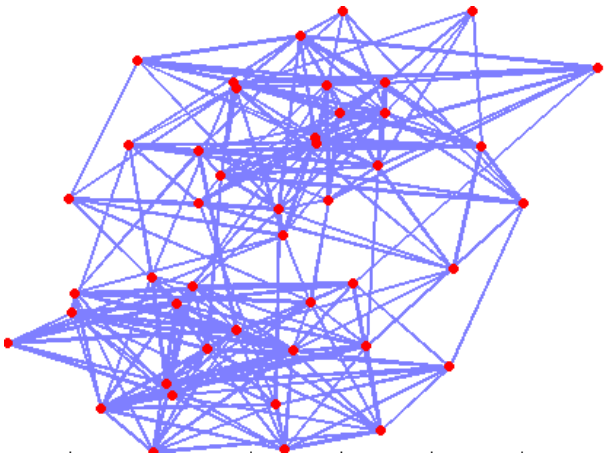
Linear Algebra API for Graphs

Table 4.1: A Mathematical overview of the fundamental GraphBLAS operations supported.

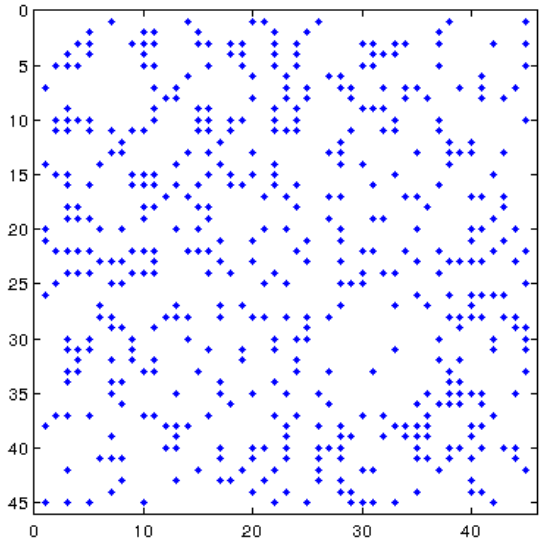
Operation Name	Mathematical Description
mXm	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^T \oplus . \otimes \mathbf{B}^T$
mXv	$\mathbf{c}(\neg\mathbf{m}) \oplus= \mathbf{A}^T \oplus . \otimes \mathbf{b}$
vXm	$\mathbf{c}(\neg\mathbf{m}) \oplus= \mathbf{b} \oplus . \otimes \mathbf{A}^T$
eWiseMult	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^T \otimes \mathbf{B}^T$
eWiseAdd	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^T \oplus \mathbf{B}^T$
reduce (row)	$\mathbf{c}(\neg\mathbf{m}) \oplus= \oplus_j \mathbf{A}^T(:, j)$
apply	$\mathbf{C}(\neg\mathbf{M}) \oplus= f(\mathbf{A}^T)$
transpose	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^T$
extract	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^T(i, j)$
assign	$\mathbf{C}(\neg\mathbf{M})(\mathbf{i}, \mathbf{j}) \oplus= \mathbf{A}^T$
buildMatrix	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbb{S}^{m \times n}(\mathbf{i}, \mathbf{j}, \mathbf{v}, \oplus_{dup})$
extractTuples	$(\mathbf{i}, \mathbf{j}, \mathbf{v}) = \mathbf{A}(\neg\mathbf{M})$

- Sparsity in data set is key differentiator
- http://www.graphblas.org/home/
- http://graphblas.org/index.php/Graph_BLAS_Forum

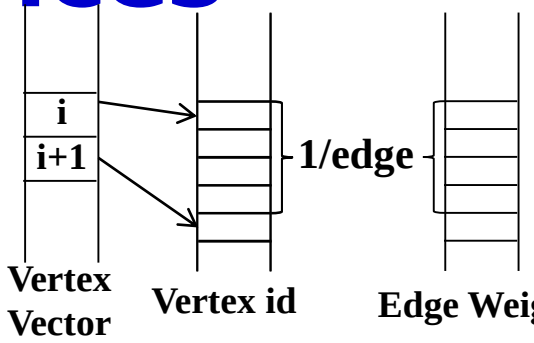
Representing Sparse Adjacency Matrices



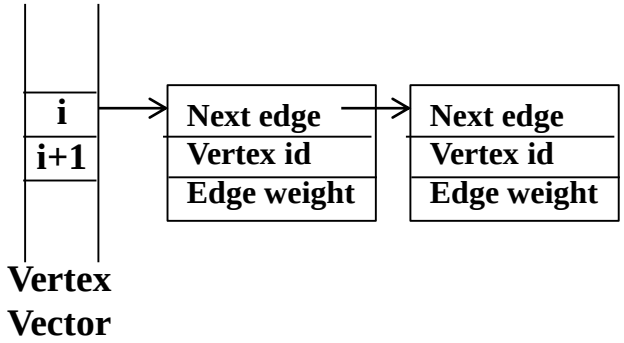
A Graph



Adjacency Matrix



(a) CSR

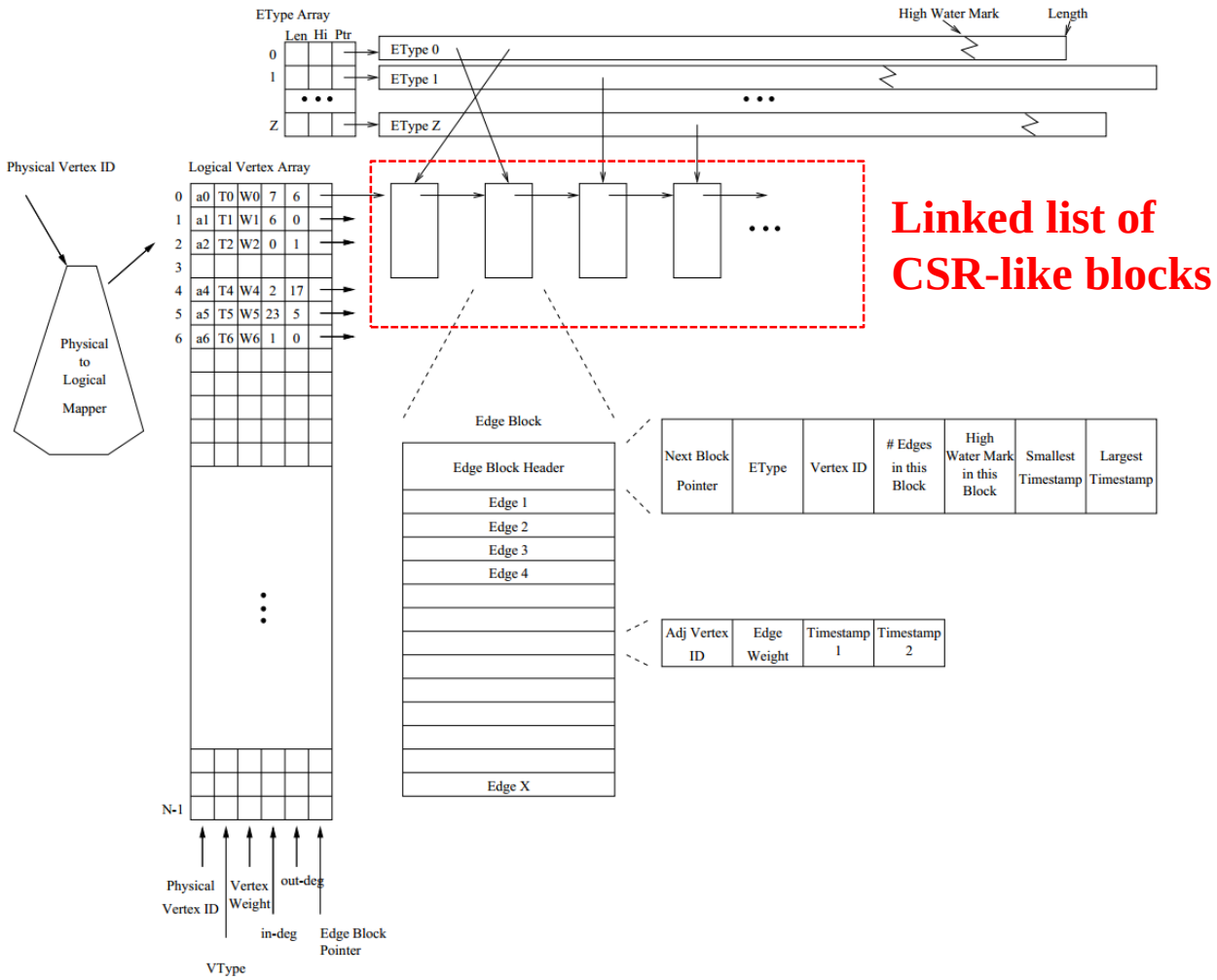


(b) Linked List

Real Adjacency Matrices often have a few non-zeros per x-million element rows
Corresponds to very low degree relative to |V|

https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18

Stinger: A Mixed Representation



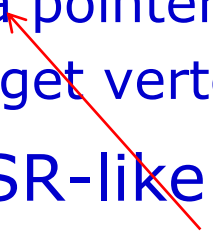
http://www.stingergraph.com/data/uploads/stinger_design.png

Sparsity and Locality

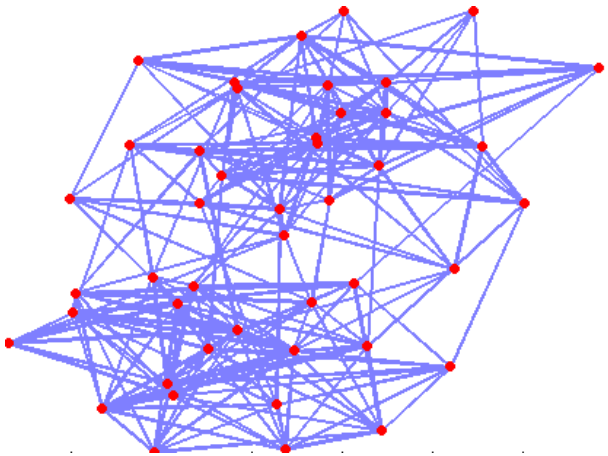
- Assume multi-node parallel system
  - Vertex data "striped" across all nodes
  - Unique subset on each node

Where do edge representations get stored?

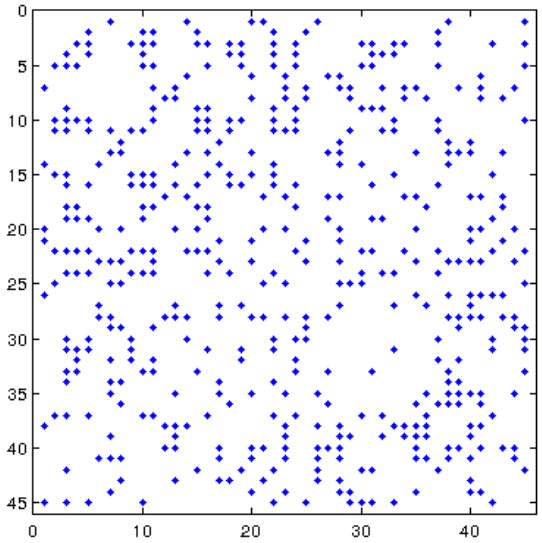
- With source vertex (especially with CSR)
  - All local accesses for edge information
  - Requires remote accesses for target vertex properties
- With target vertex (especially with Linked list)
  - Sequential remote accesses to get next non-zero
  - With extra pointer access to get "next edge"
  - But all target vertex properties are then local
- Stinger: CSR-like block on node with those targets
  - Eliminates intermediate pointer chasing



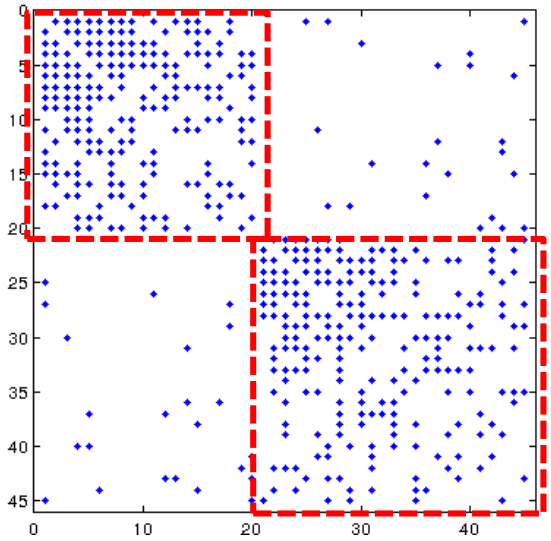
Sometimes We Can Reorder



A Graph



Adjacency Matrix



Let's "reorder" the Vertices

- Locality = how many NZ in same memory

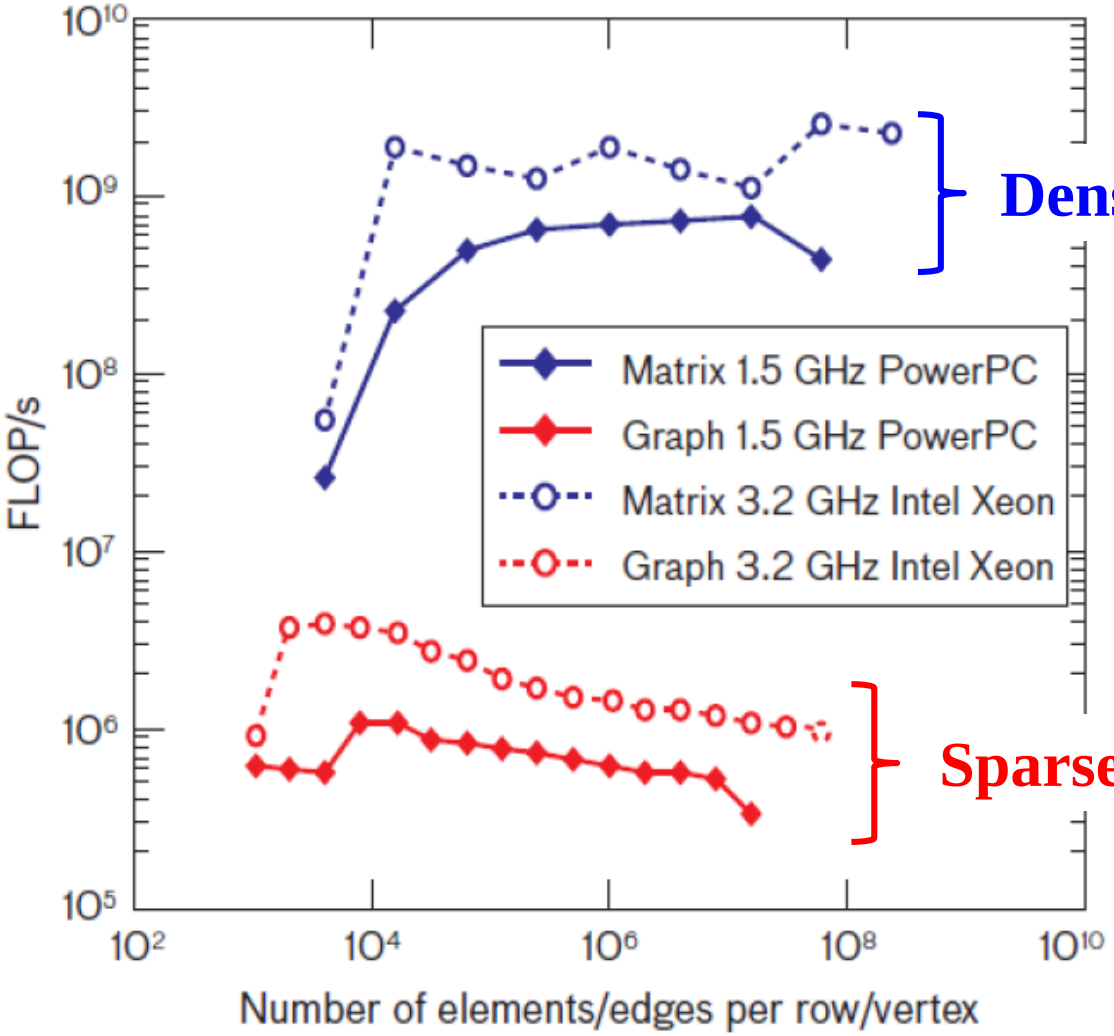
Seldom Possible to Do This Reordering: esp. for Streaming Updates

https://www.math.ksu.edu/~albin/matlab_html/graph_partitioning/gp_demo.html#18

What About High-Degree Vertices?

- Real graphs have huge variation in degree
  - Consider Google or Amazon in a web graph
- CSR bad fit for huge out-degree
- Linked list bad fit for huge in-degree
- Option: each node has own copy of vertex info on each high-degree vertex
  - Local edge lists only for edges terminating on this node
- Algorithms over high-degree vertices now different

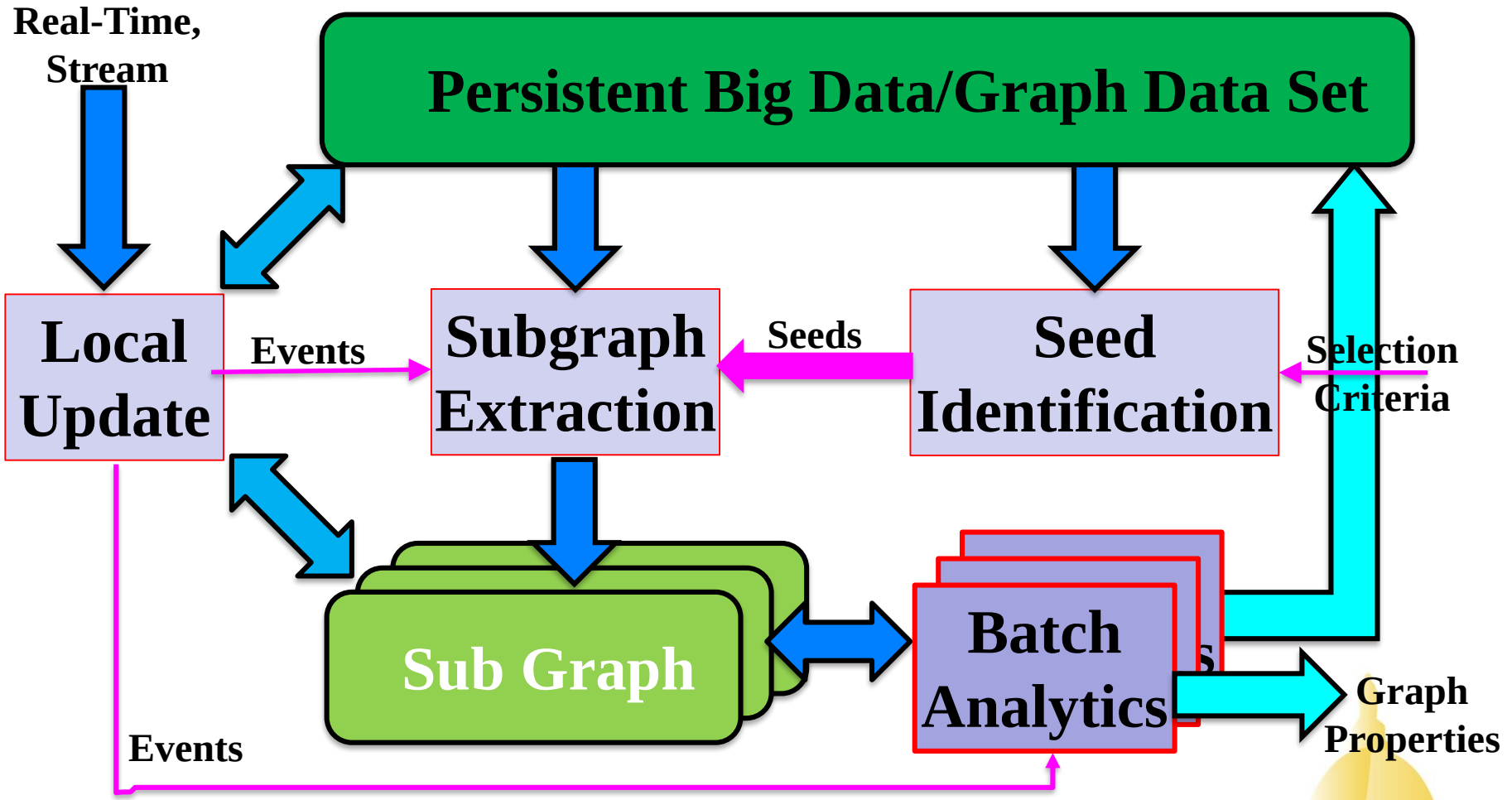
Sparsity on a Single Core



Song, et al. "Novel Graph Processor Architecture, Prototype System, and Results," IEEE HPEC 2016

Using Today's Parallel Architectures

Canonical Graph Processing Flow



Seed Identification

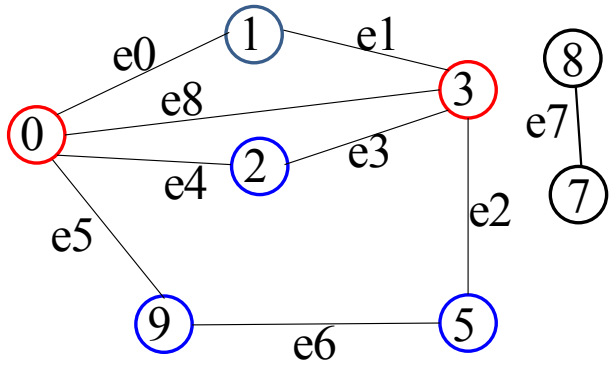
- Typical algorithm: **Scored Search**
  - Step 1: Search thru all vertices
    - Apply select predicate to vertex properties
  - Step 2: Score passing vertices
    - Scale selected columns by weighting factor & sum
  - Step 3: Report "Top N" passing scores
    - Typically 10-100
- Steps 1, 2 embarrassingly parallel if each vertex has all properties on same node
- Finding Top N is major parallelization issue
  - O(N*log(P)): Do local Top N and exchange only at end
  - Or send to distributed structure, with threshold feedback

Some Analytics

- **HPCG**: Solving Ax = b in 3D grid, but
  - Extremely sparse, notionally ~27 non zeros per row
  - But sparseness can be localized
- **SpMV**: Sparse matrix-vector product
  - Core kernel of both HPCG and many graph analytics
  - Many non-HPCG cases both sparser with significant variation in out-degree
- **BFS**: Breadth First Search from GRAPH500
  - Again very sparse, but huge variation in out-degree
  - Very little "programmable" locality

Breadth First Search

- Core of GRAPH 500 rankings
- Start with a root, find all reachable vertices
- Metric: **TEPS**: Traversed Edges/sec
- Performance issues
  - Massive data sets
  - Very irregular access
  - Very challenging load balancing



Starting at 1: 1, 0, 3, 2, 9, 5

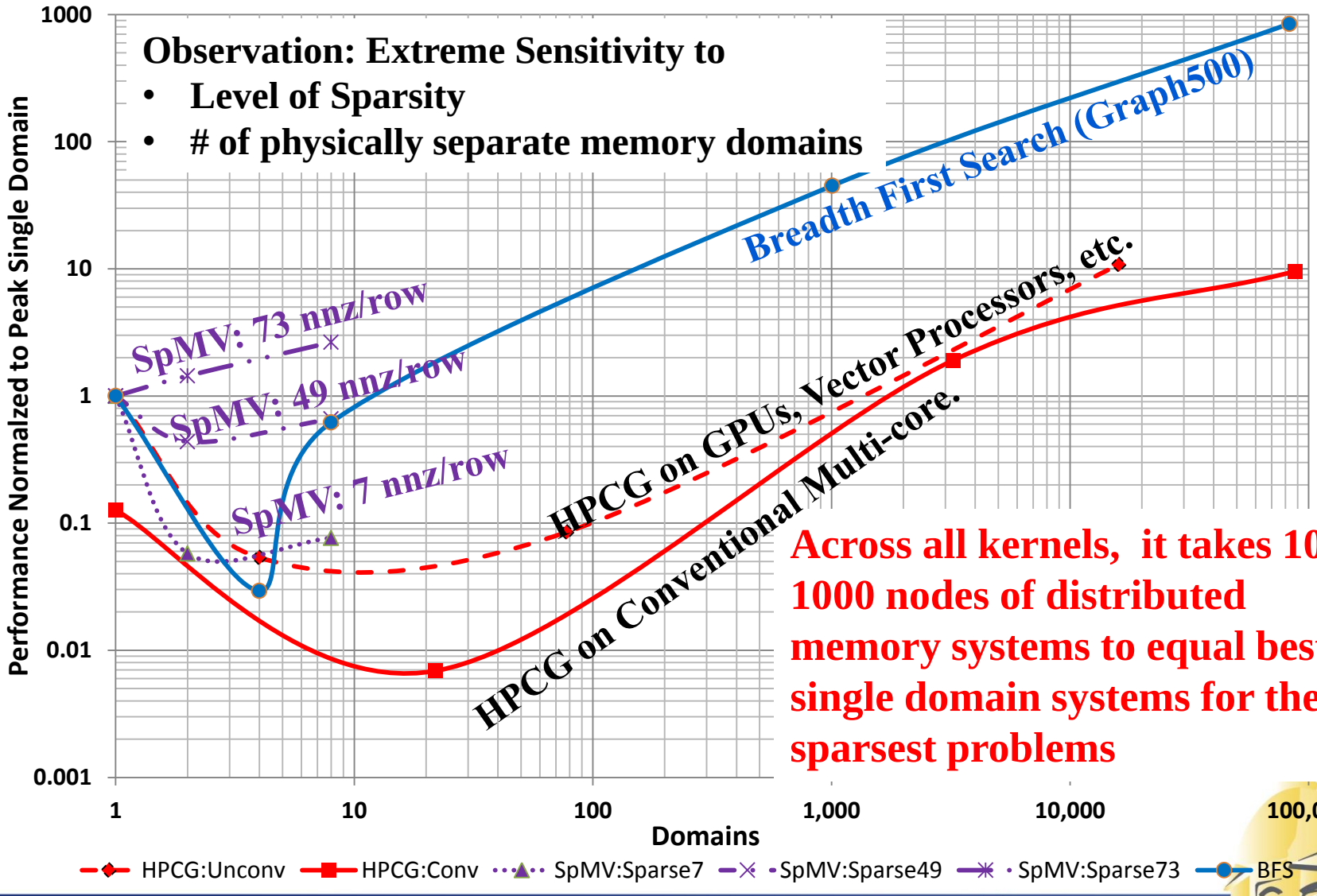
GRAPH500 Graph Sizes

Level	Scale	Size	Vertices (Billion)	TB	Bytes /Vertex
10	26	Toy	0.1	0.02	281.8048
11	29	Mini	0.5	0.14	281.3952
12	32	Small	4.3	1.1	281.472
13	36	Medium	68.7	17.6	281.4752
14	39	Large	549.8	141	281.475
15	42	Huge	4398.0	1,126	281.475
				Average	281.5162

Scale = log2(# vertices)

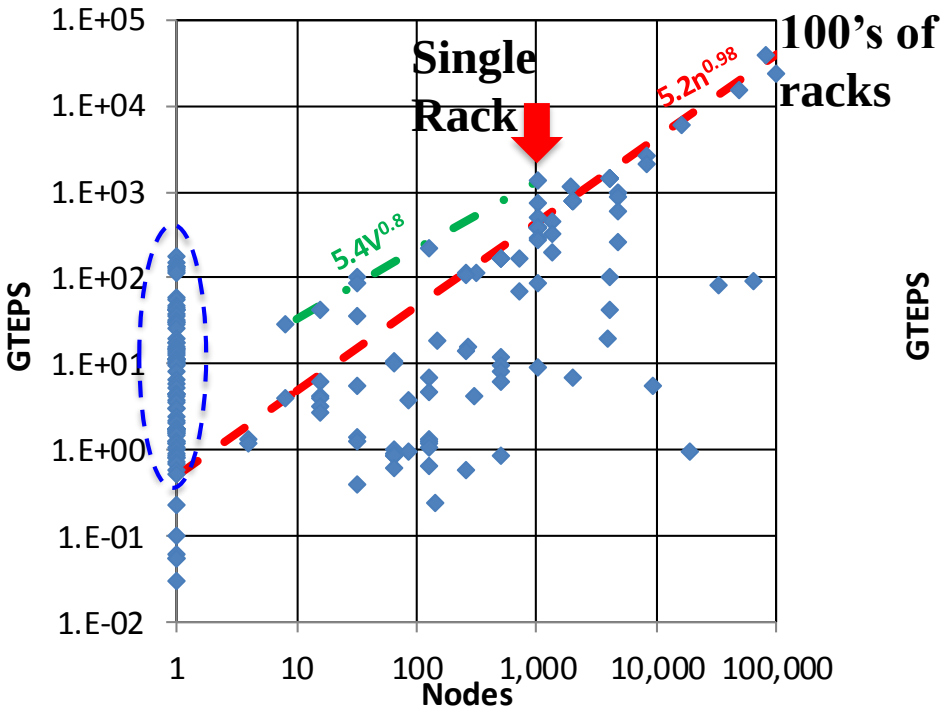
Real-World: Eqvt of BFS thru 1-3 layers only

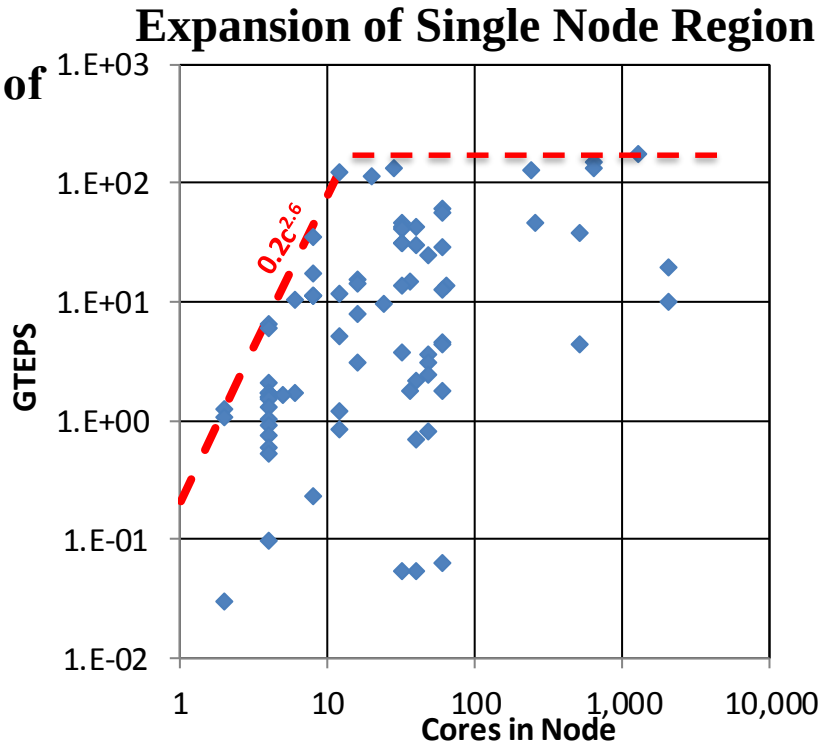
Sparsity & Parallelism



Across all kernels, it takes 10-1000 nodes of distributed memory systems to equal best of single domain systems for the sparsest problems

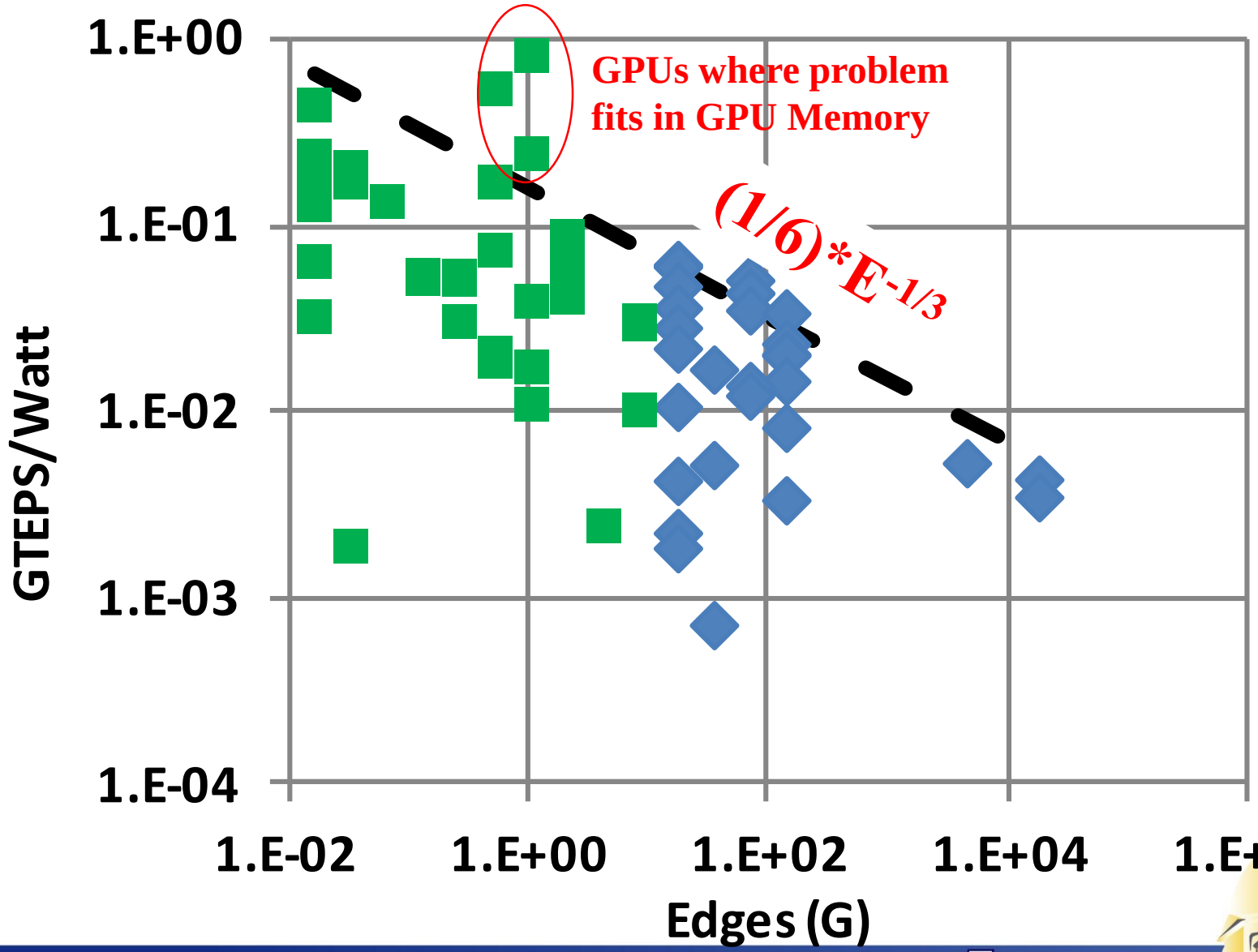
Recent GRAPH500 Data





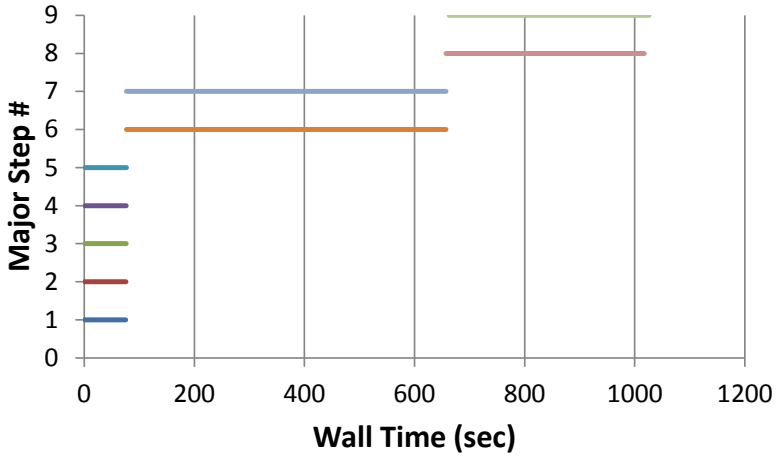
Single Node/Domain systems are more efficient than multi-node BUT do not scale

BFS and Energy



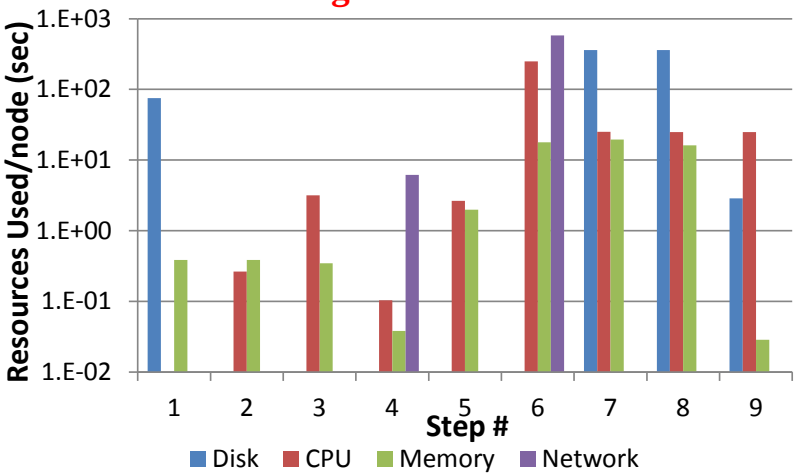
Modeling the LexisNexis Problem

Baseline: 10 Racks of 2012 Commodity servers

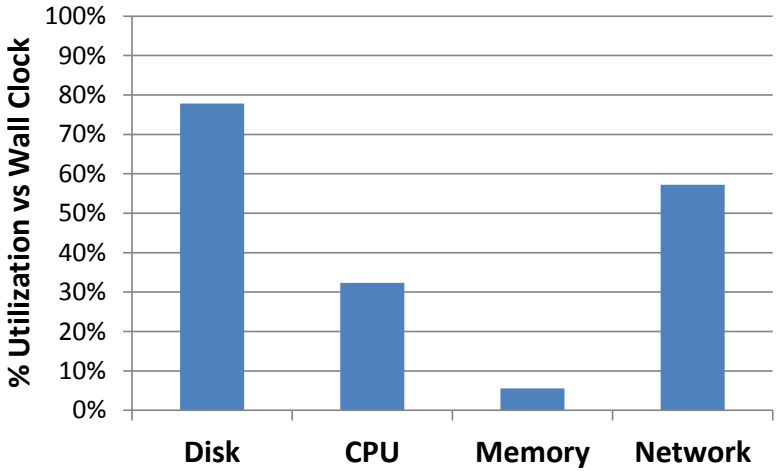


(a) Wall clock time (in sec) by step

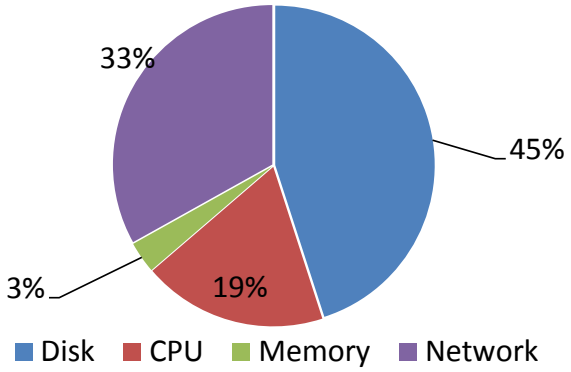
Note Logarithmic Y Axis



(b) Resource utilization by step

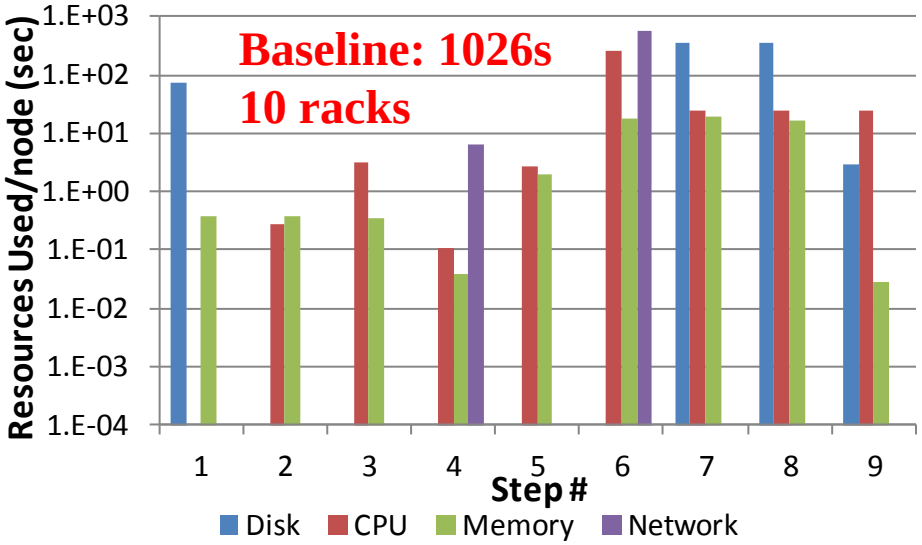


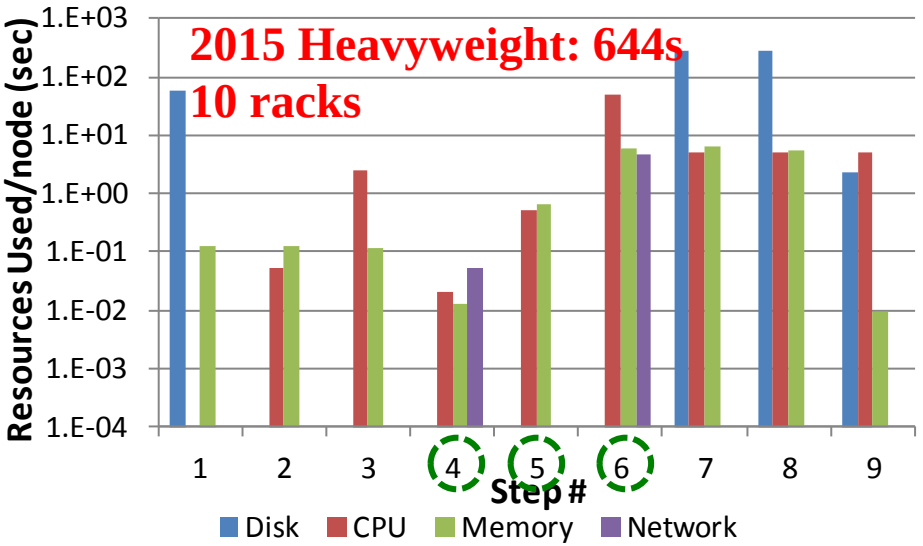
(c) Resource utilization vs. Wall Clock

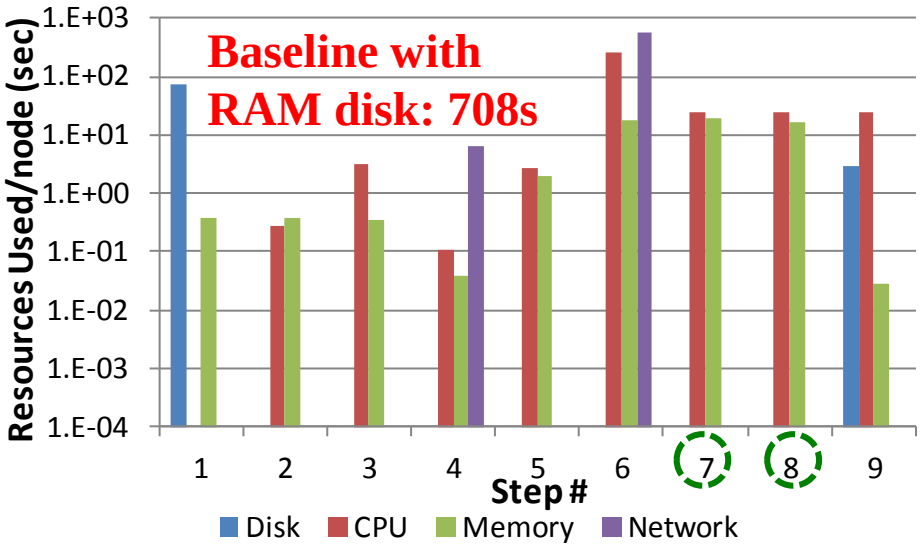


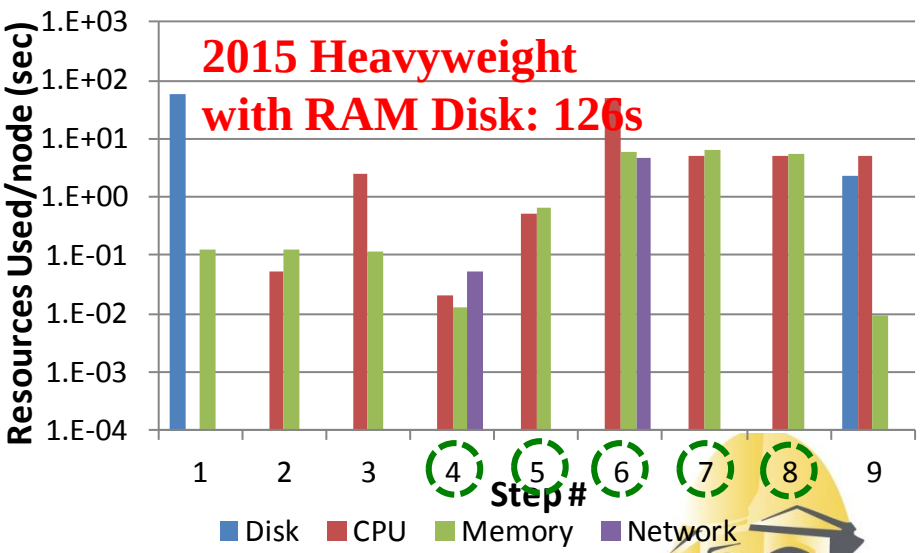
(d) Breakdown of overall resources

Details: Heavyweight Alternatives



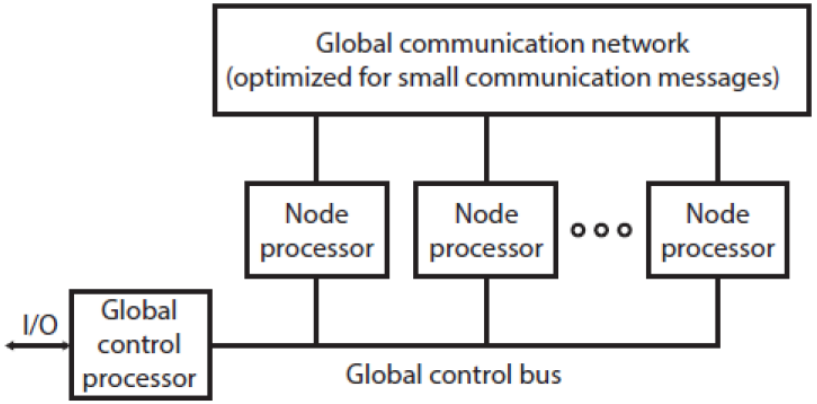


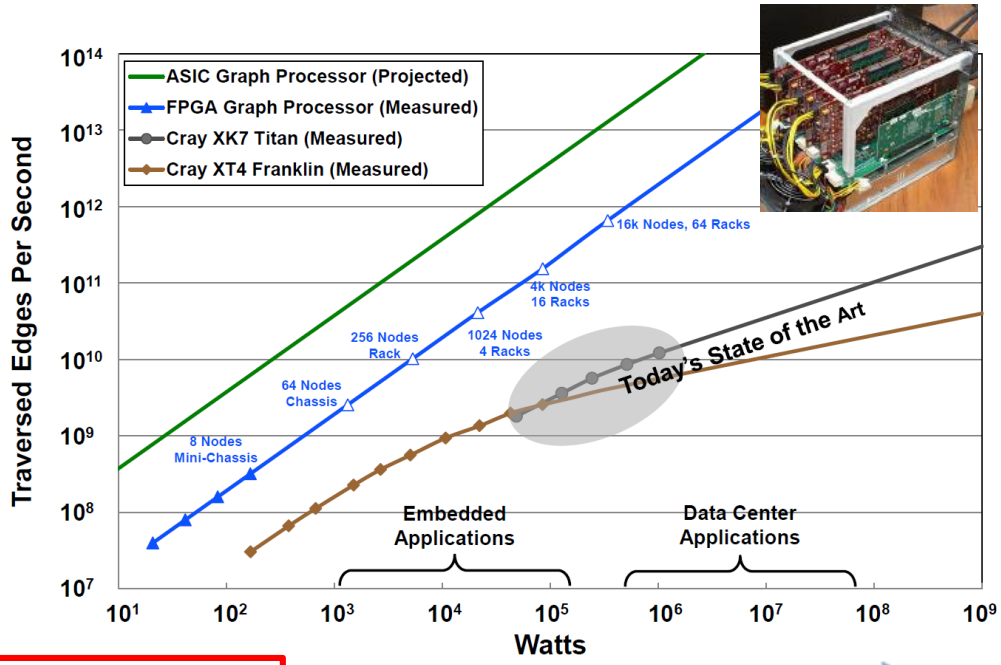


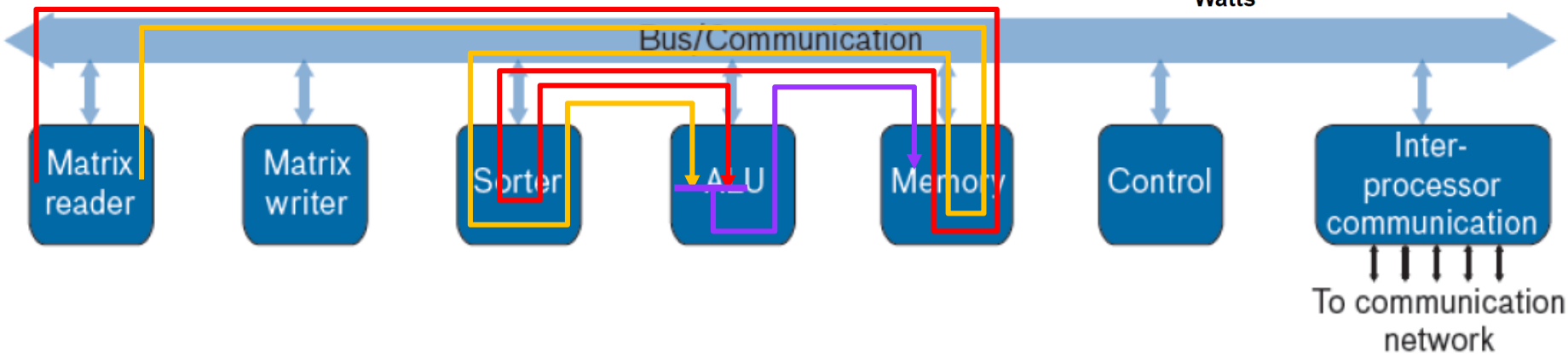


Emerging Architectures

A Novel Sparse Graph Processor





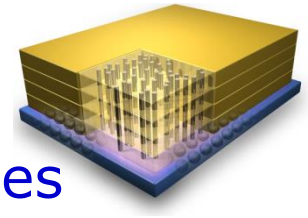


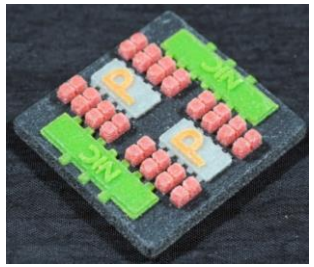
Song, et al. "Novel Graph Processor Architecture, Prototype System, and Results," IEEE HPEC 2016

Return to LexisNexis Problem

2013 study looked at "future" alternatives

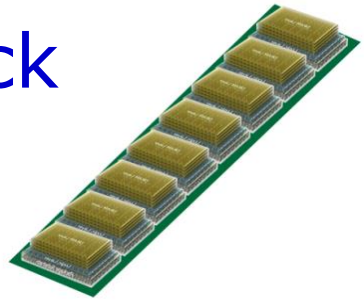
- "Lightweight" systems
  - Lower power, lower performance cores
  - Study assumed Calxeda 4-core ARMs
  - but systems like HP Moonshot similar



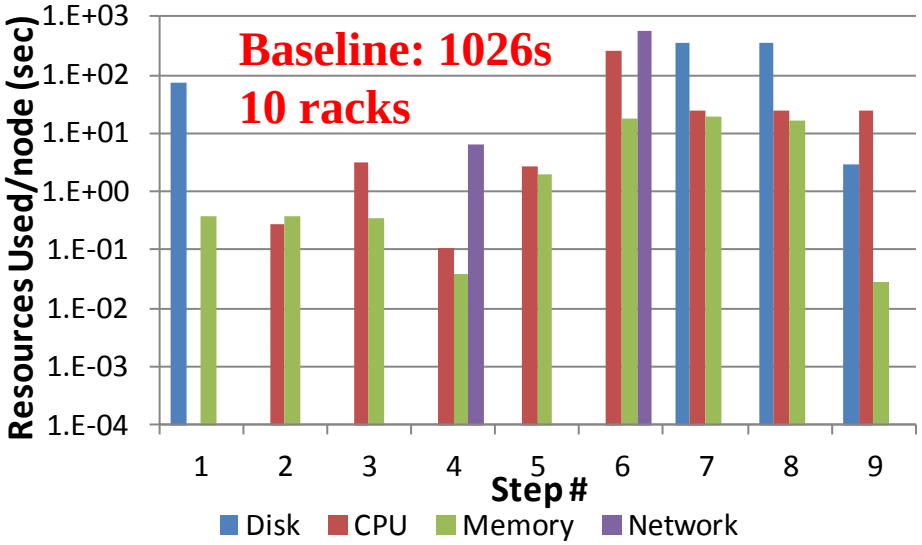


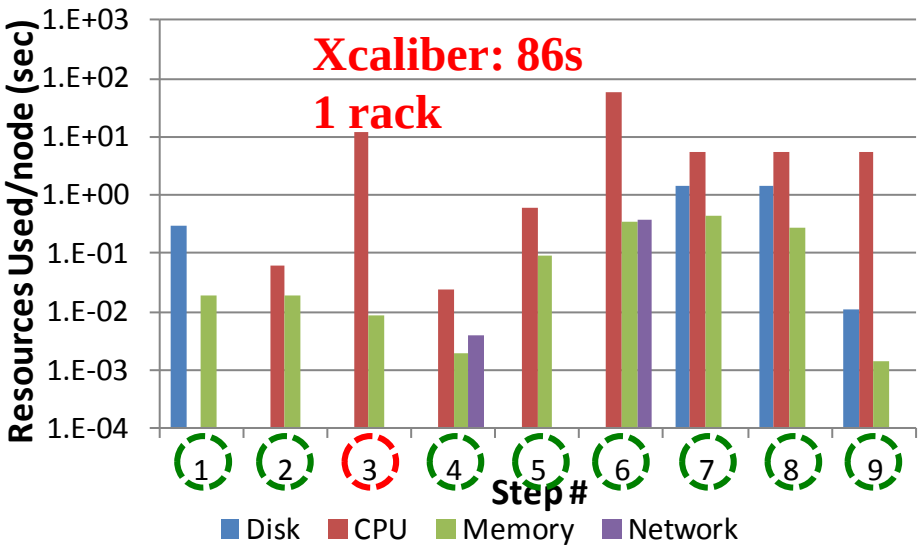
(b) X-caliber Node Mockup

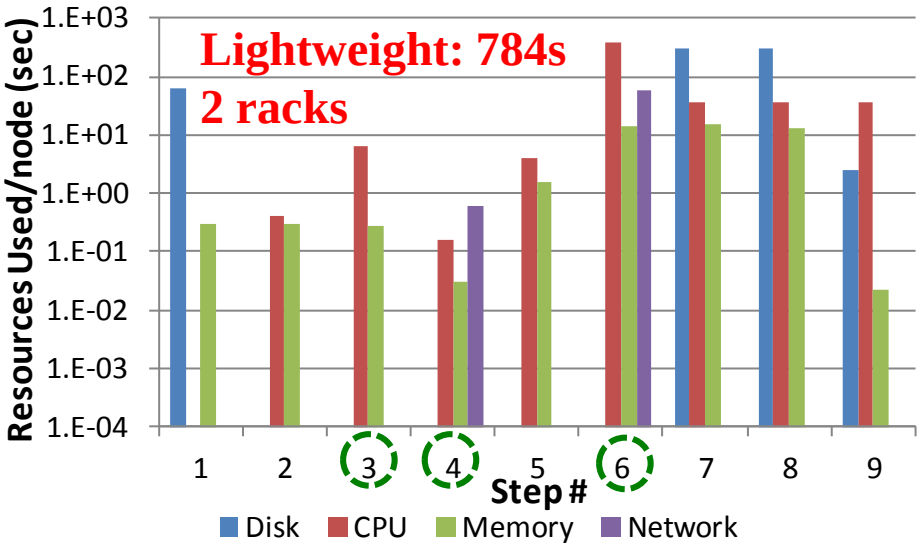
- Sandia's X-Caliber project
  - Heavyweight with HMC-like memories
  - Now like Intel's Knights Landing
- All processing on bottom of 3D stack
  - System = "sea" of stacks

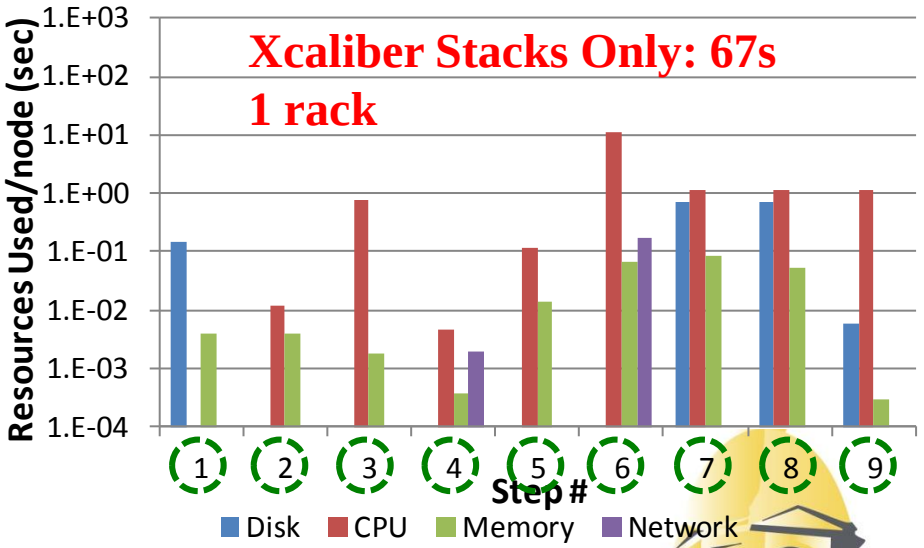


Matching Projections

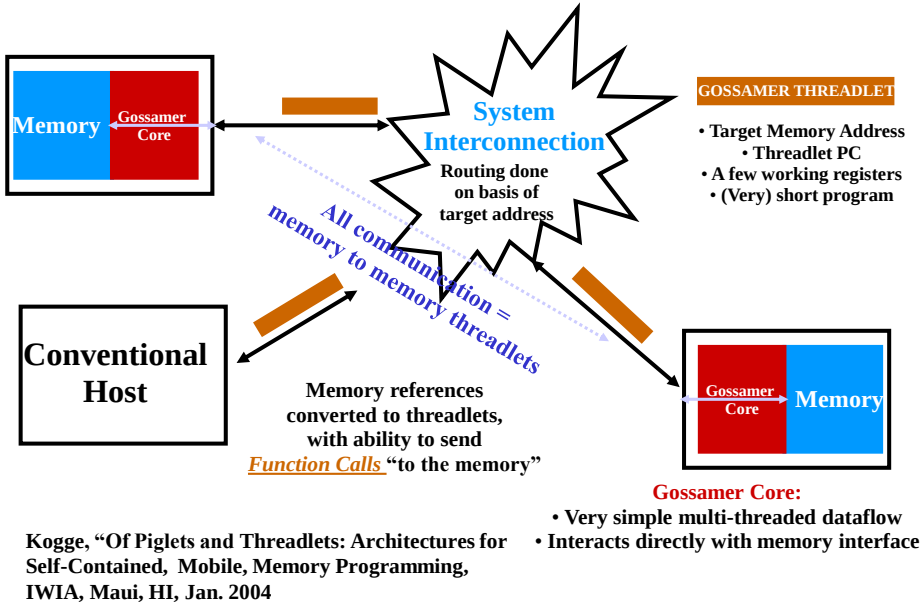






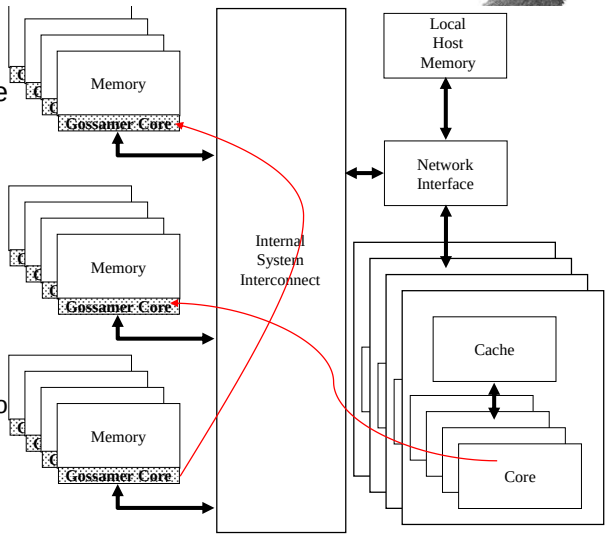


Traveling Threads



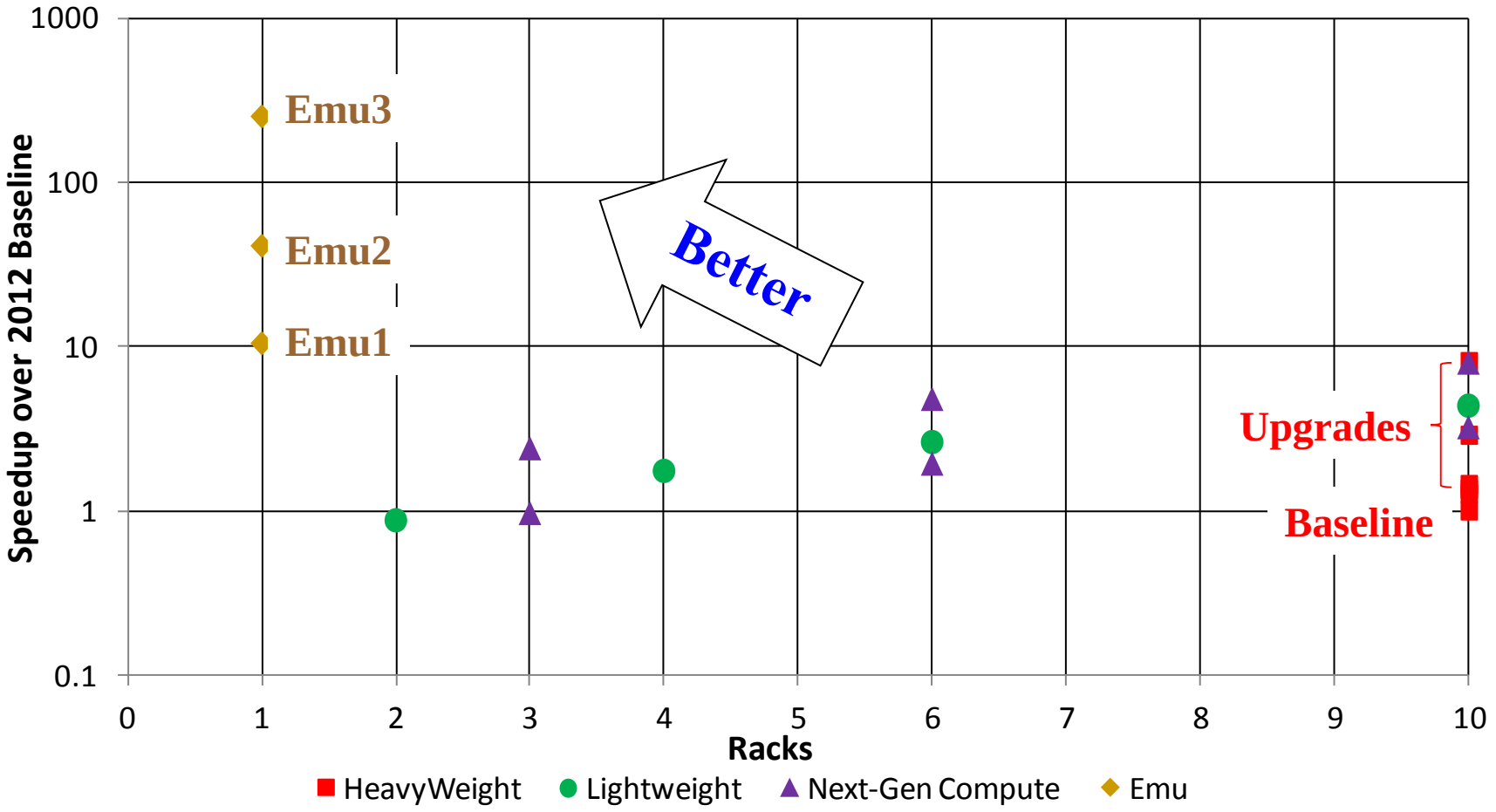
Kogge, "Of Piglets and Threadlets: Architectures for Self-Contained, Mobile, Memory Programming, IWIA, Maui, HI, Jan. 2004

- Single Address Space
- Visible to all Hosts & Gossamer Cores
- Hosts can issue
  - Reads and Writes
  - ***Threadlets***
- Gossamer Cores can
  - Spawn new threadlets
  - Migrate threadlets to other cores

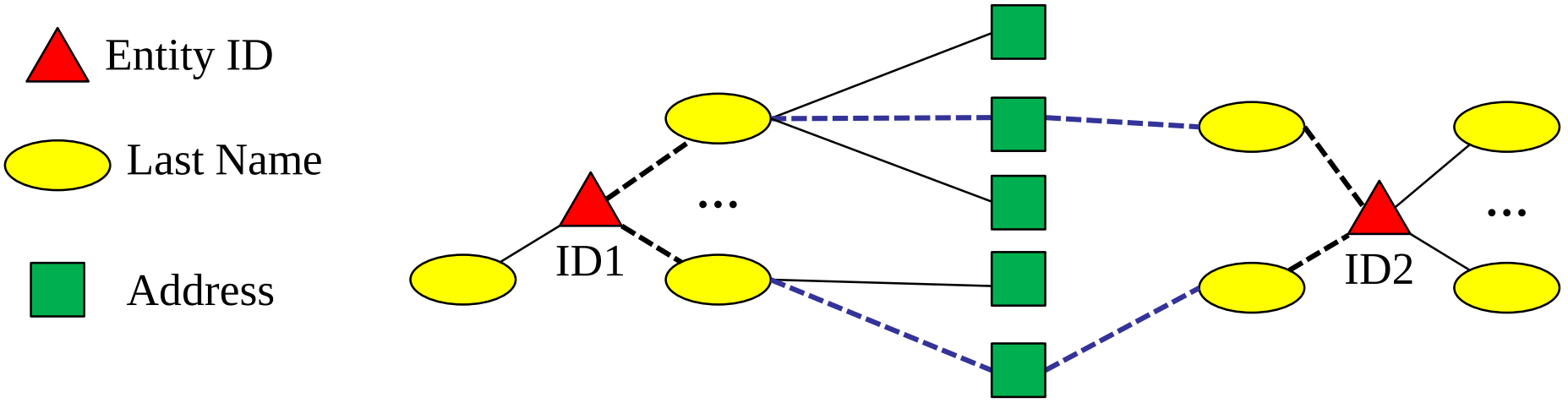


- Single Address Space Visible to all Hosts & Gossamer Cores
- Hosts can launch:
  - Reads and Writes of Memory
  - Threads for execution on Gossamer core
- Threads on a Gossamer Core can
  - Spawn new threads
  - Migrate threads to other cores
- Memory need not be just today's DRAM

Comparison on LexisNexis Problem



But You Want Real-Time?



Query: Given specific ID1 find all ID2s meeting requirements

Estimated Gain	Emu1 vs 2013	Emu3 vs 2013
Response Time	250X	250X
Queries/Sec	30X	100,00X

Summary

- Huge variation in application kernels
- Today's benchmarks != real-world
- Streaming != batch
  - What is computed greatly affects complexity
  - May look more like sparse problems
- Sparsity & locality huge drivers for parallel
  - Phenomena of Single domain systems
- Issues are with memory, not processing
- Wide range of emerging architectures attack the memory, not compute

Acknowledgements

Funded in part by

- The Univ. of Notre Dame
- NSF Grant CCF-1642280