Managing and Mining Billion-Node Graphs

Haixun Wang
Microsoft Research Asia

Our Focus

System & Graph Processing

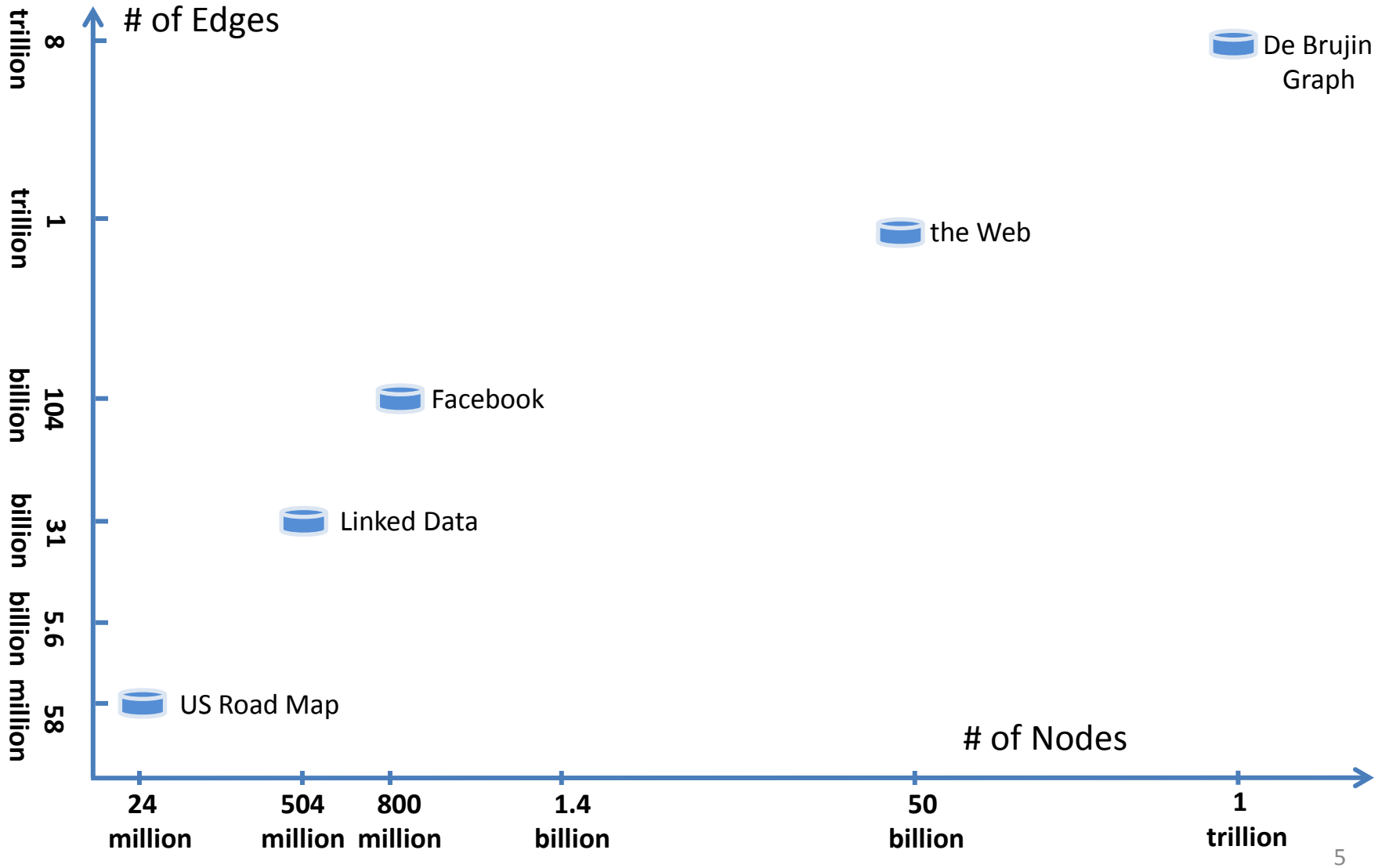
Outline

- Large Graph Challenges
- Systems for Large Graphs
  - RDBMS, Map Reduce, Pregel, Pegasus, Trinity
- Key Graph Algorithms
  - Graph Partitioning, Traversal, Query, Analytics

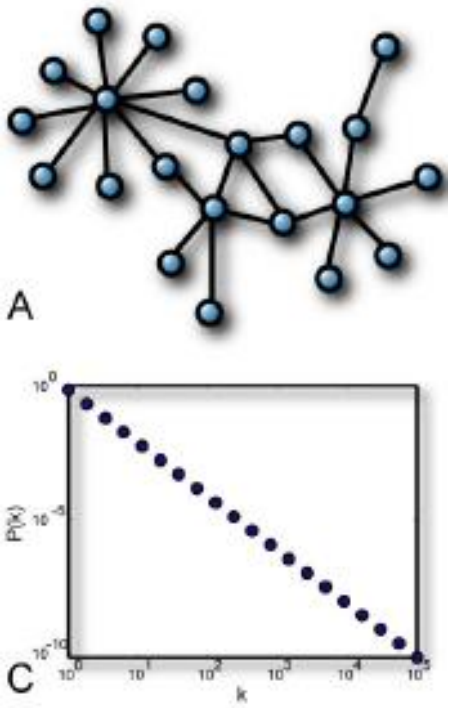
Outline

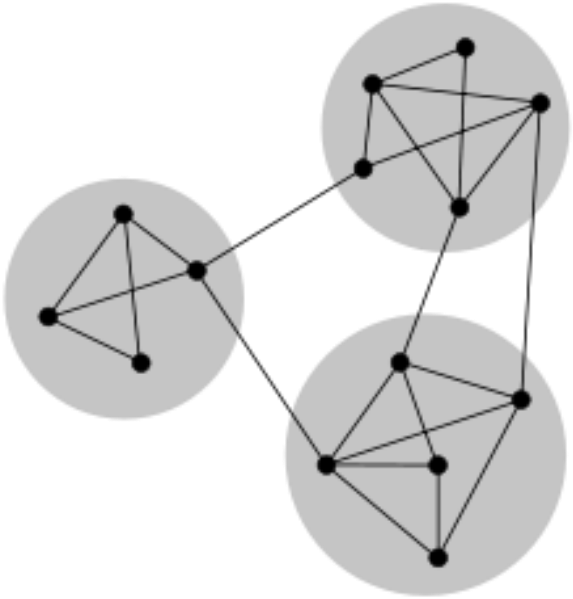
- Large Graph Challenges
- Systems for Large Graphs
  - RDBMS, Map Reduce, Pregel, Pegasus, Trinity
- Key Algorithms
  - Graph Partitioning, Traversal, Query, Analytics

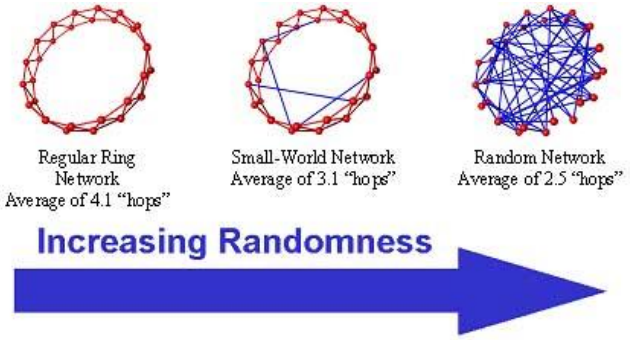
Graphs encode rich relationships



Diversity of Graphs







$P(k) \sim k^{-a}$

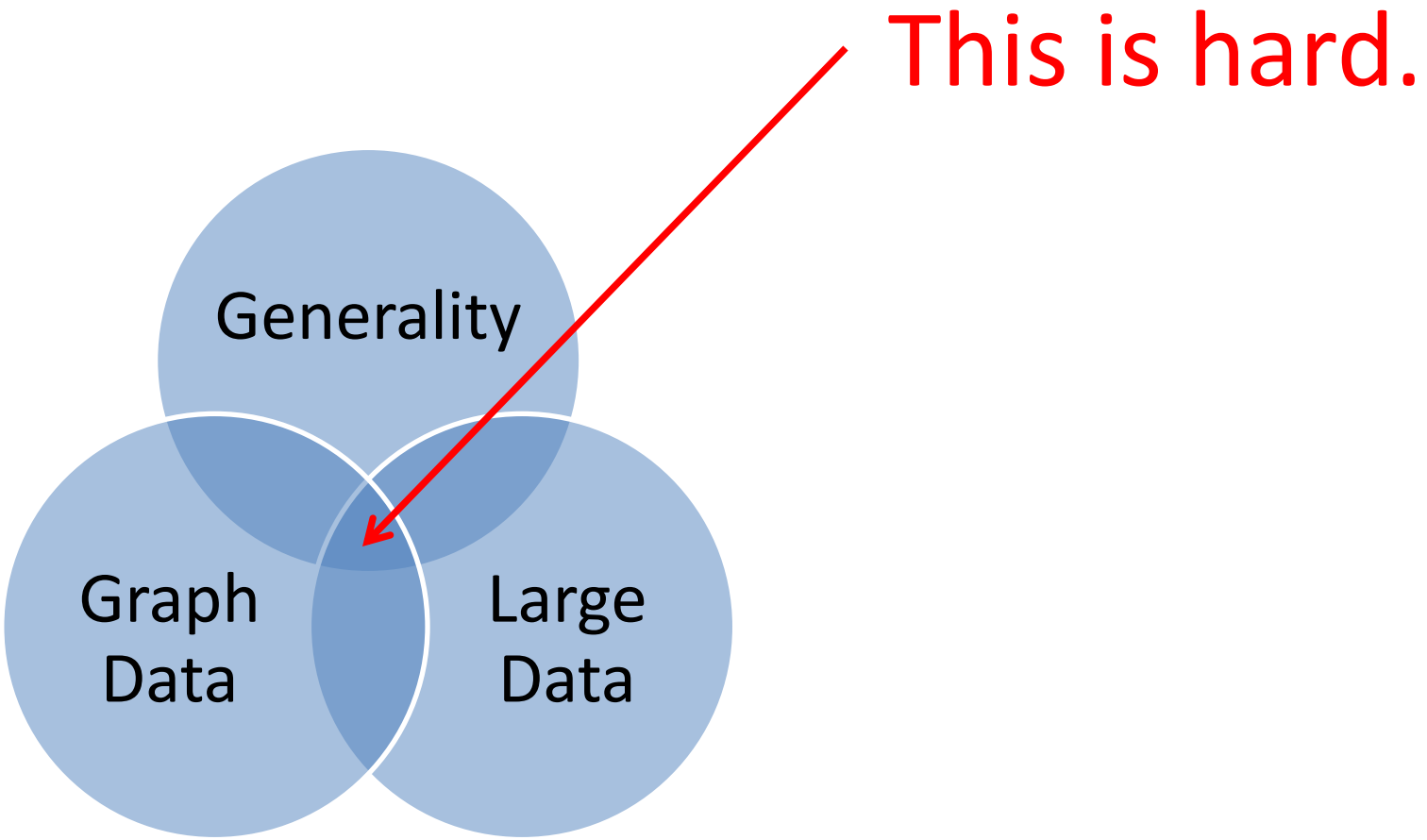
Scale Free Graphs **Community Structure** **Small World**

A Large Variety of Graph Operations

- Online query processing
  - Shortest path query
  - Subgraph matching query
  - SPARQL query
  - …
- Offline graph analytics
  - PageRank
  - Community detection
  - …
- Other graph operations
  - Graph generation, visualization, interactive exploration, etc.

Current Status

- Good systems for processing *graphs*:
  - PBGL, Neo4j
- Good systems for processing *large data*:
  - Map/Reduce, Hadoop
- Good systems for processing specialized *large graph data*:
  - Specialized systems for pagerank, etc.

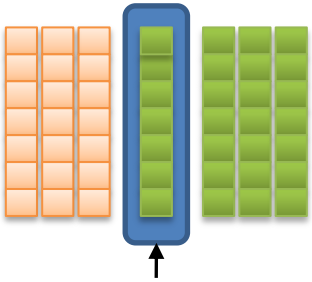


No good system for processing general large graphs

Graph processing without a system is hard!

Fundamental issues

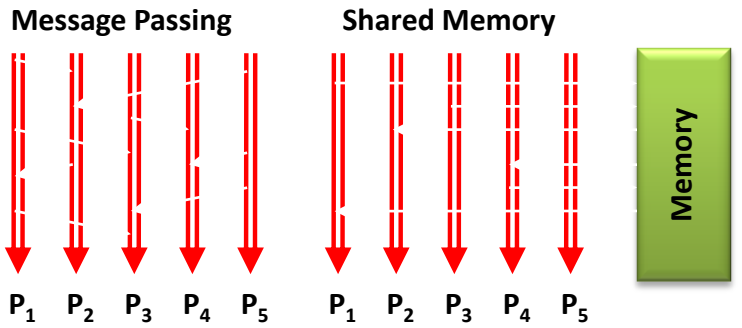
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …



Architectural issues

Flynn's taxonomy (SIMD, MIMD, etc.),
network typology, bisection bandwidth
UMA vs. NUMA, cache coherence

Different programming models

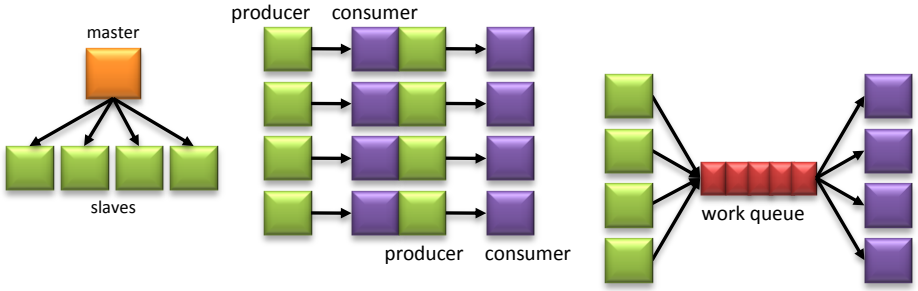


Common problems

livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

Different programming constructs

mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …



Programmer shoulders the burden of managing all these subtle issues…

Adapted from: Jimmy Lin, SIKS/BigGrid Big Data Tutorial (2011)

Benefits of a general purpose system

- Enable applications to focus on algorithm rather than system implementation
- Support a variety of algorithms on the same graph data in the same platform
- Support various types of graphs

Challenges

Graph Data is “Special”

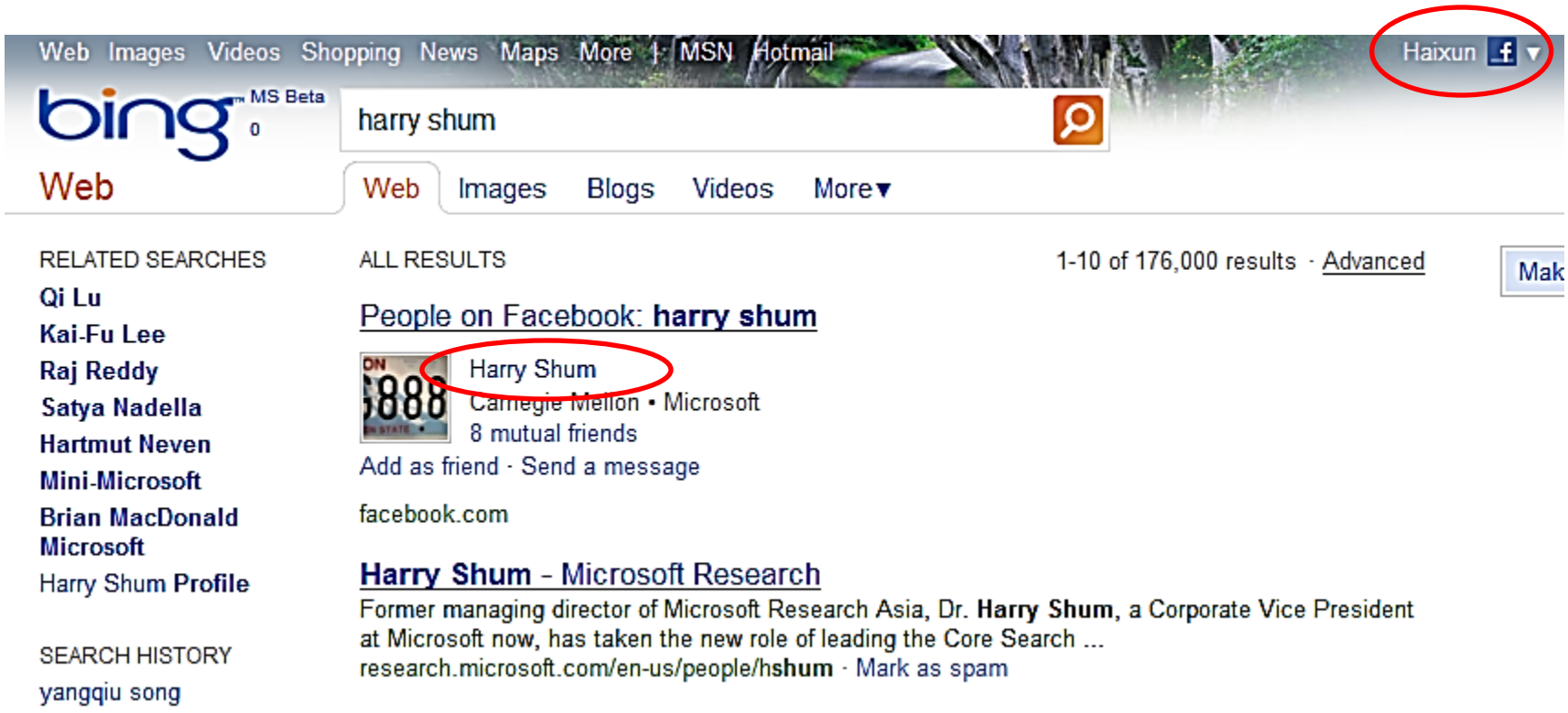
- Poor locality (random access is required)
  - Accessing a node’s neighbor requires “jumping” around no matter how the graph is represented.
- Data or graph structure driven
  - Computation is dictated by graph structure

Challenges of Online Query

Two examples:

- Online graph exploration
- Sub-graph matching

Online graph exploration



- Challenge
  - Limited time budget: ¼ second
  - Large number of nodes to visit:

130
130 x 130
130 x 130 x 130
2,214,030

Subgraph matching

Procedure:

1. Break a graph into basic units (edges, paths, frequent sub-graphs, …)

2. Build index for every possible basic unit

3. Decompose a query into multiple basic unit queries, and join the results.

Subgraph matching in large graphs

- Graph indices
  - Time: 2-hop reachability index requires $O(n^4)$ construction time
  - Space: Depending on the structure of the basic unit. In many cases, super linear.

- Multi-way joins
  - Costly for disk resident data

Query Index Examples

Algorithms	Index Size	Index Time	Update Cost
Ullmann [Ullmann76], VF2 [CordellaFSV04]	-	-	-
RDF-3X [NeumannW10]	$O(m)$	$O(m)$	$O(d)$
BitMat [AtreCZH10]	$O(m)$	$O(m)$	$O(m)$
Subdue [HolderCD94]	-	Exponential	$O(m)$
SpiderMine [ZhuQLYHY11]	-	Exponential	$O(m)$
R-Join [ChengYDYW08]	$O(nm^{1/2})$	$O(n^4)$	$O(n)$
Distance-Join [ZouCO09]	$O(nm^{1/2})$	$O(n^4)$	$O(n)$
GraphQL [HeS08]	$O(m + nd^r)$	$O(m + nd^r)$	$O(d^r)$
Zhao [ZhaoH10]	$O(nd^r)$	$O(nd^r)$	$O(d^L)$
GADDI [ZhangLY09]	$O(nd^L)$	$O(nd^L)$	$O(d^L)$

Index-based subgraph matching [Sun2012]

Query Index Examples

Algorithms	Index Size for Facebook	Index Time for Facebook	Query Time on Facebook (s)
Ullmann [Ullmann76], VF2 [CordellaFSV04]	-	-	>1000
RDF-3X [NeumannW10]	1T	>20 days	>48
BitMat [AtreCZH10]	2.4T	>20 days	>269
Subdue [HolderCD94]	-	> 67 years	-
SpiderMine [ZhuQLYHY11]	-	> 3 years	-
R-Join [ChengYDYW08]	>175T	$> 10^{15}$ years	>200
Distance-Join [ZouCO09]	>175T	$> 10^{15}$ years	>4000
GraphQL [HeS08]	>13T(r=2)	> 600 years	>2000
Zhao [ZhaoH10]	>12T(r=2)	> 600 years	>600
GADDI [ZhangLY09]	$> 2 \times 10^5$T (L=4)	$> 4 \times 10^5$ years	>400

Index-based subgraph matching [Sun2012]

Challenges of Offline Analytics

- Offline analytics requires high throughput
- Unstructured nature of graph
  - Difficult to achieve parallelism by partitioning data
  - Hard to get an efficient “Divide and Conquer” solution
- High data-access to computation ratio
  - Huge amount of disk I/O

Design Choices

System Design Choice

- Storage
  - Disk-based storage vs. Memory-based storage
- Scale-“UP” vs. Scale-“OUT”
  - Supercomputer vs. distributed cluster

Technical Trend

- Now: A commodity PC server has dozen gigabytes of memory

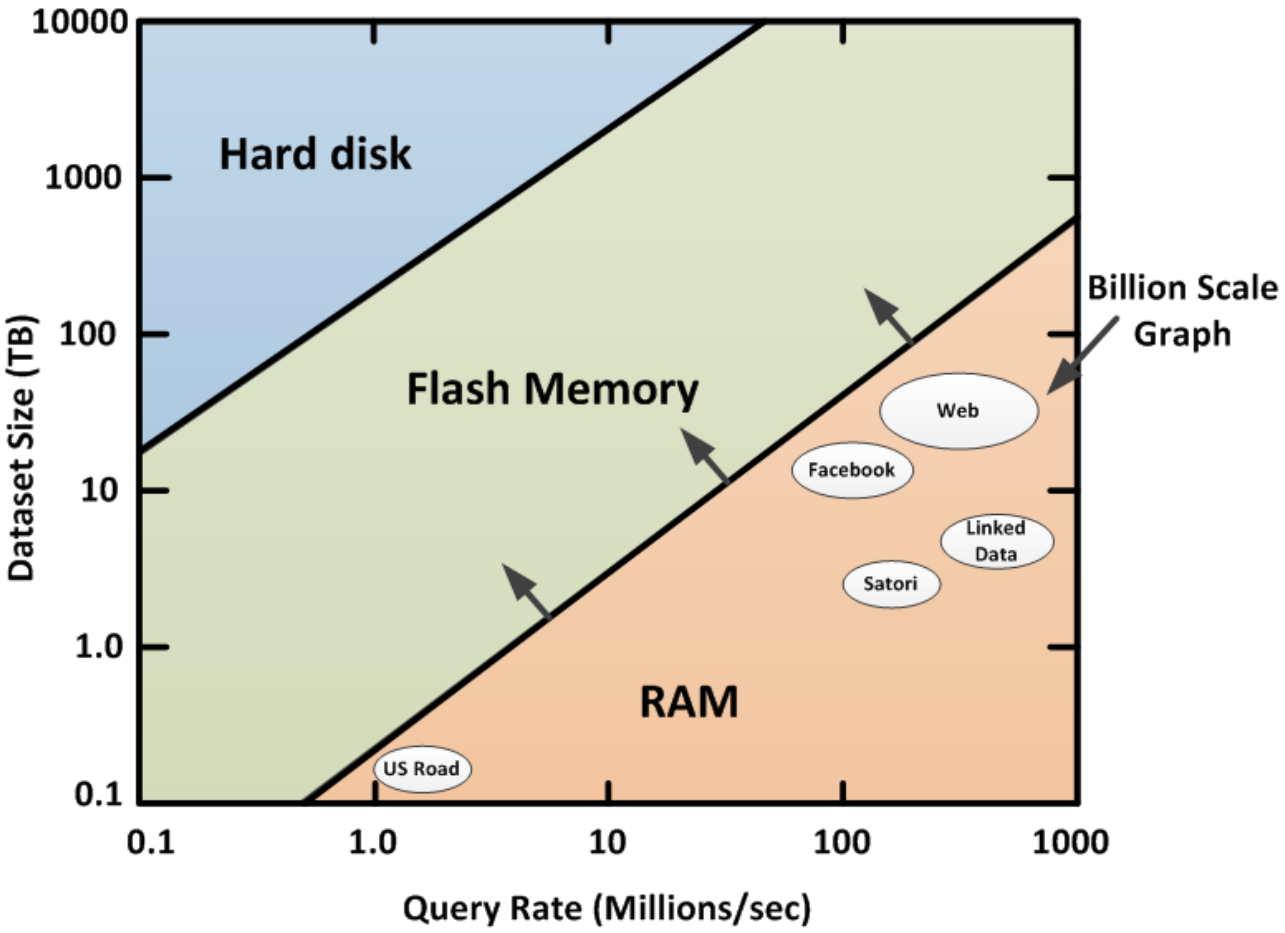
- Future: High-capacity memory and all-in-memory applications

Trend in Cost of Memory Storage

	Today	In 5-10 years
# servers	**1000**	**1000**
GB/server	**64GB**	**1024GB**
Total capacity	**64TB**	**1PB**
Total server cost	**$4M**	**$4M**
$/GB	**$60**	**$4**

Adapted from: John Ousterhout, RAMCloud, 2010

Total Cost of Ownership



Reproduced from Anderson's SOSP 2009 paper

Design Choice: Storage

1. Massive random data access pattern
2. lower total cost of ownership

RAM can be ideal main storage for graphs

Design Choice: Scale-up vs. Scale-out

- Supercomputer model
  - Programming model simple and efficient
    - shared memory address space
  - Expensive and not common
  - Hardware is your ultimate limit

- Distributed cluster model
  - Programming model is complex
    - Message passing and synchronization is more complex
  - Relatively cheaper and can make use of commodity pc
  - More flexible to meet various needs

Outline

- Large Graph Challenges
- Systems for Large Graphs
  - RDBMS, Map Reduce, Pregel, Pegasus, Trinity
- Key Algorithms
  - Graph Partitioning, Traversal, Query, Analytics

Existing Systems

- Systems specialized for certain graph operations:
  - PageRank
- Systems designed for non-graph data
  - RDBMS
  - Map Reduce
- Graph Systems
  - Libraries for graph operations, e.g. PBGL
  - Matrix-based graph processing system, e.g. Pegasus
  - Graph database, e.g. Neo4j, HypergraphDB, Trinity
  - Graph analytics system, e.g. Pregel, Trinity, Giraph,

RDBMS

- Mainstay of business
  - Strong data integrity guarantee via ACID transactions
- Support complex queries
  - Standard query language: SQL
- Limited graph query support
  - Oracle (RDF query support)

Traverse Graph Using Joins

ID	name	….
1	N1	…
2	N2	…
3	N3	…
4	N4	…
5	N5	…
6	N6	…
…	…	…

Node Table: N

src	dst
1	3
2	4
2	1
4	3
1	5
1	6
…	…

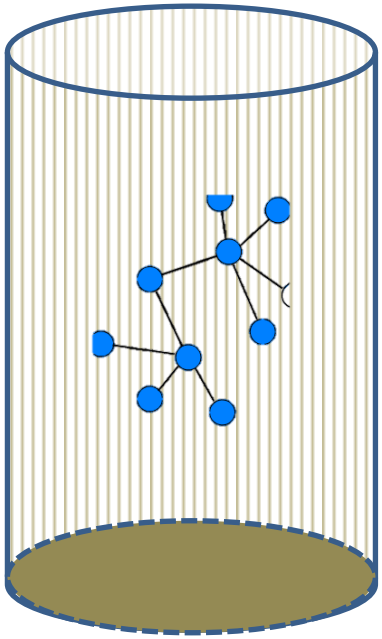
Edge Table: E

Get Neighbors of N1

```
SELECT*
FROM N
LEFT JOIN  E ON N.ID = E.dst
WHERE E.src = 1;
```

Graph in the Jail of RDBMS

- The commonest graph operation "traversal" incurs excessive amount of table joins



Graph in the jail of RDBMS

- **RDBMS: Mature techniques but not for graphs**

Map Reduce

Map Reduce

- High latency, yet high throughput general purpose data processing platform
- Optimized for offline analytics on large data partitioned on hundreds of machines

Map Reduce

- De facto of distributed large data processing

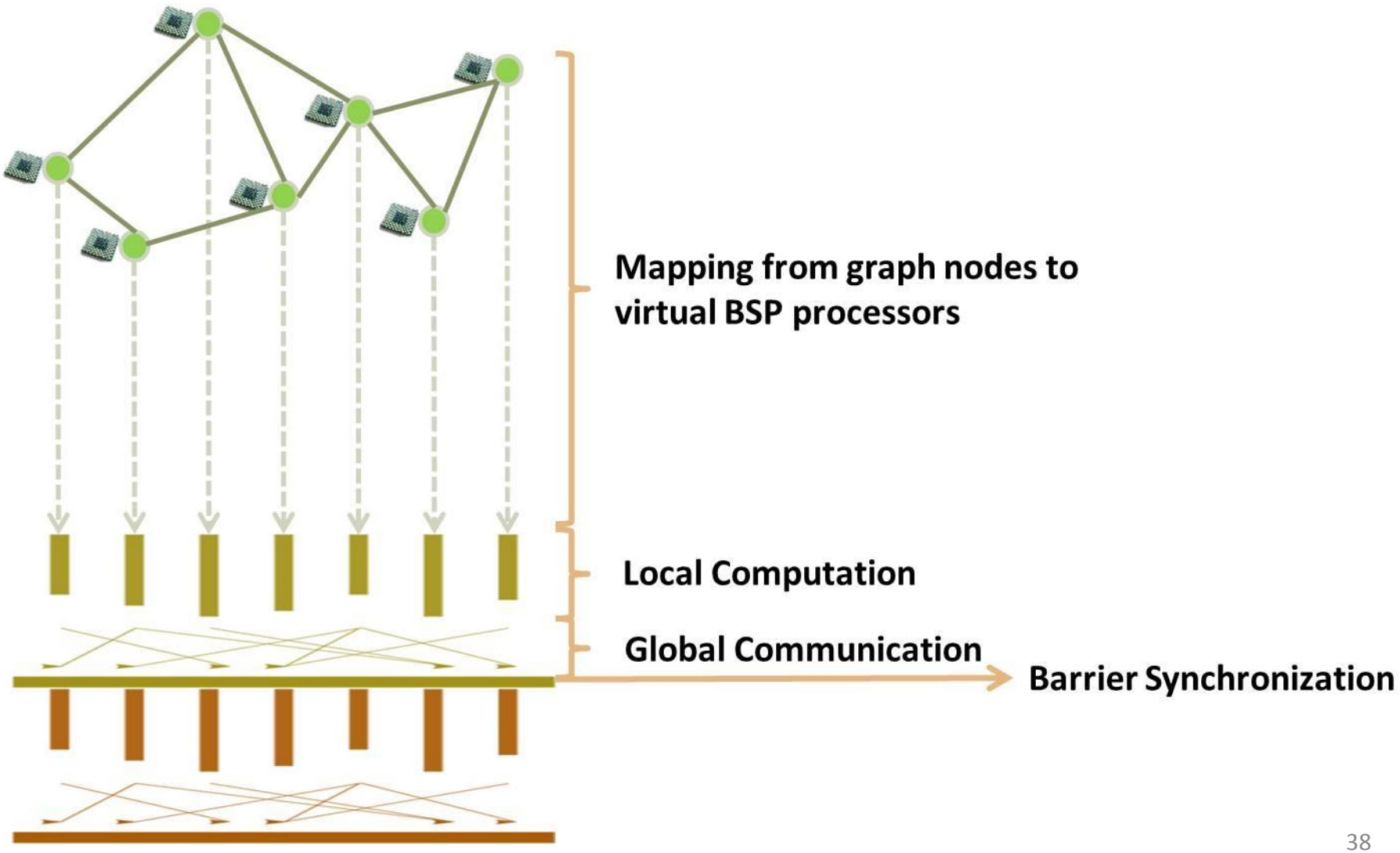
- Great scalability: supports extremely large data

Processing Graph using Map Reduce

- No online query support
- The map reduce data model is not a native graph model
  - Graph algorithms cannot be expressed intuitively
- Graph processing is inefficient on map reduce
  - Intermediate results of each iteration need to be materialized
  - Entire graph structure need to be sent over network iteration after iteration, this incurs huge unnecessary data movement

Pregel

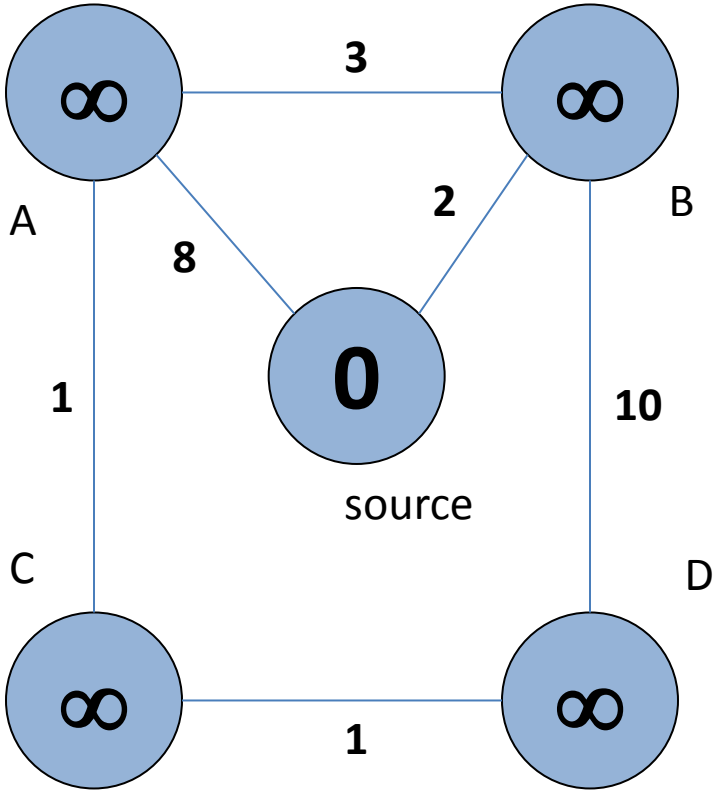
Basic Idea: Think Like a Vertex!



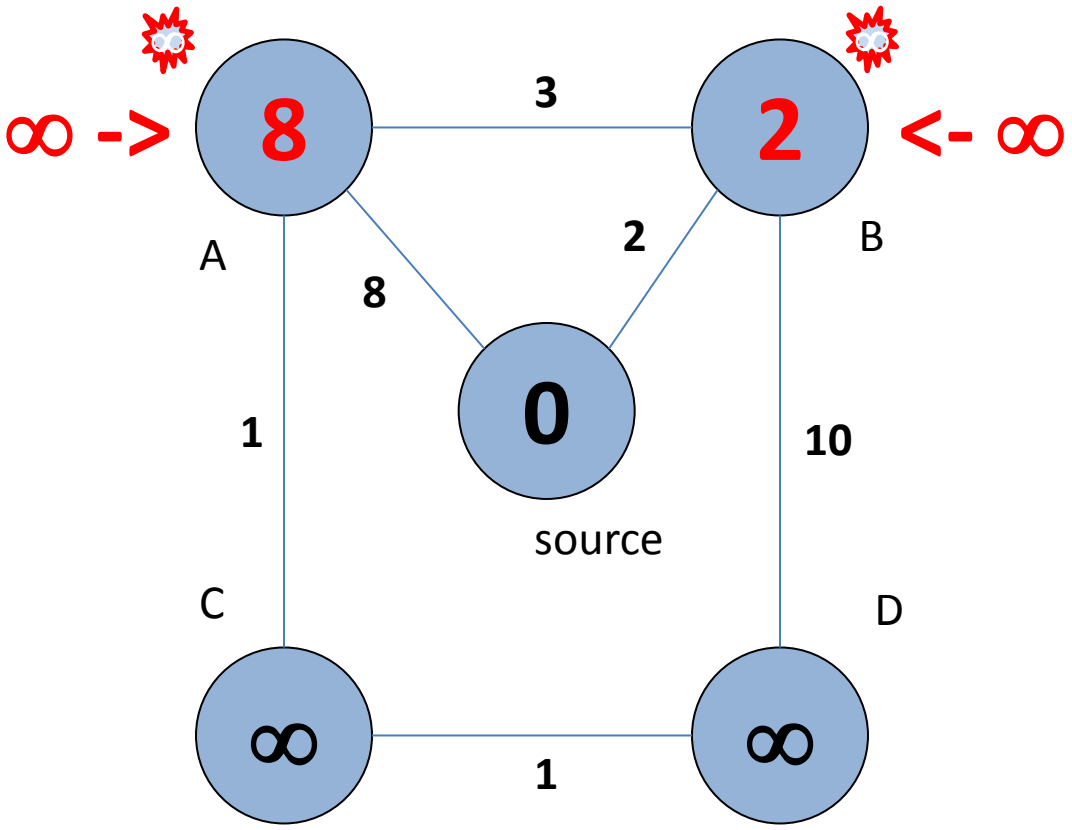
Computation Model

- Graph computation is modeled as many supersteps
- Each vertex reads messages sent in previous superstep
- Each vertex performs computation in parallel
- Each vertex can send messages to other vertices in the end of an iteration

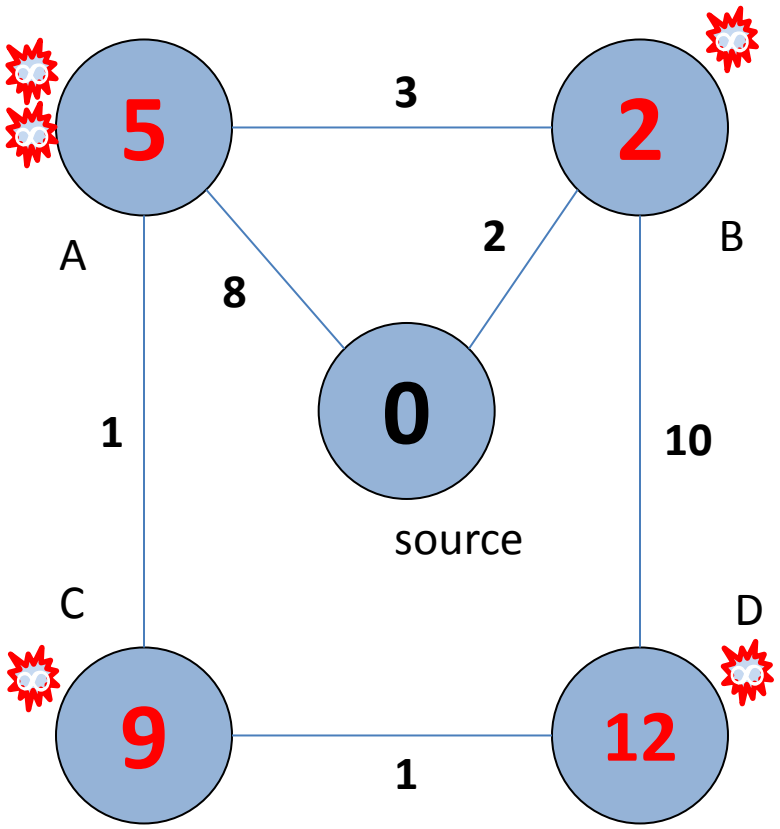
Example: SSSP in Pregel



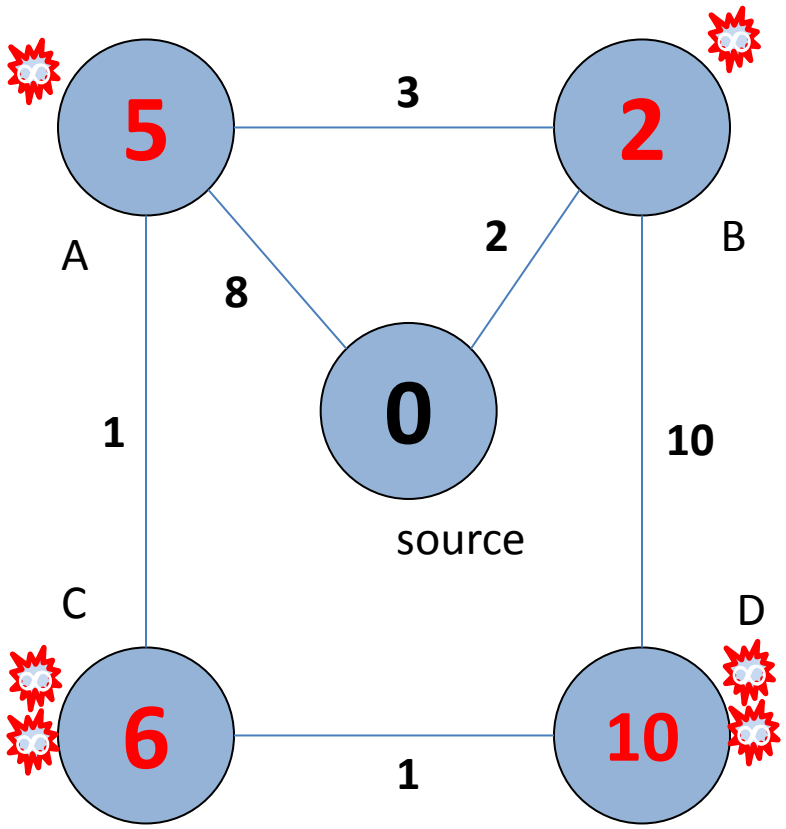
Example: SSSP in Pregel



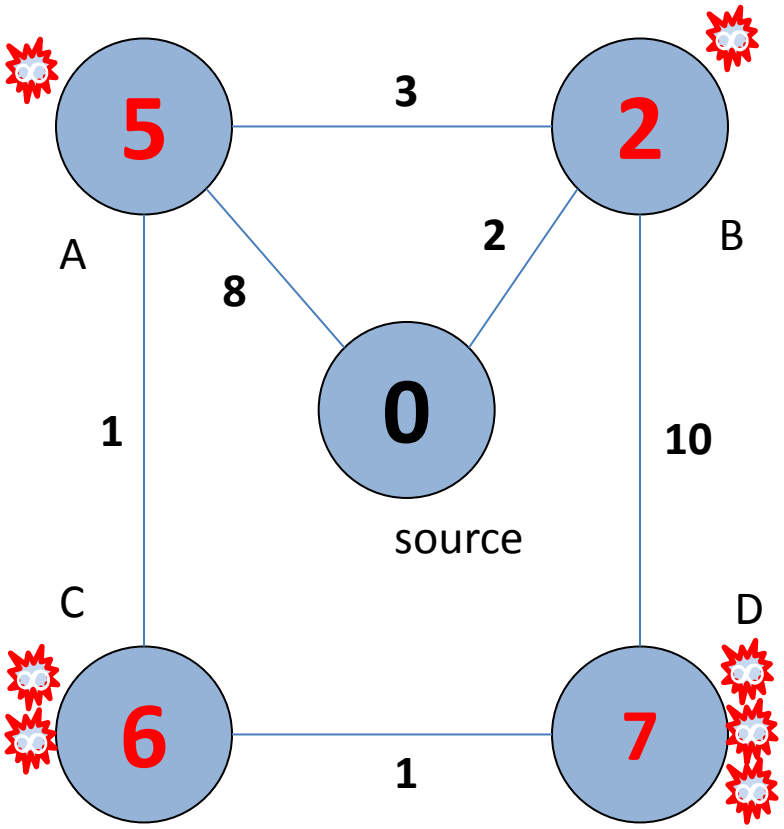
Example: SSSP in Pregel



Example: SSSP in Pregel



Example: SSSP in Pregel



Pregel vs. Map Reduce

- Exploits fine-grained parallelism at node level
- Pregel doesn’t move graph partitions over network, only messages among nodes are passed at the end of each iteration
- Not many graph algorithms can be implemented using vertex-based computation model elegantly

Pegasus: Peta-Scale Graph Mining

Pegasus

- An open source large graph mining system
  - Implemented on Hadoop

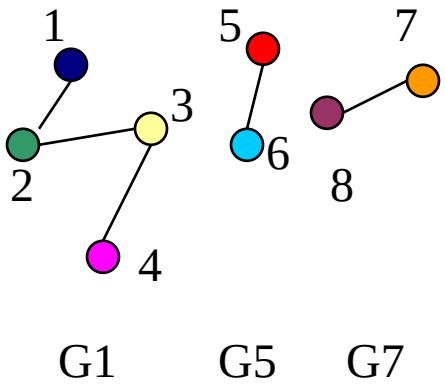
- Key Idea
  - Convert graph mining operations into iterative matrix-vector multiplication

Data Model of Pegasus

- Use vectors and matrices to represent graphs

- Specifically:
  - a graph with n vertices is represented by an $n \times n$ matrix
  - each cell (i, j) in the matrix represents an edge $(source = i, destination = j)$

Example



	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		
6					1			
7								1
8							1	

Source: Pegasus, Kevin Andryc, 2011

Generalized Iterated Matrix-Vector Multiplication (GIM-V)

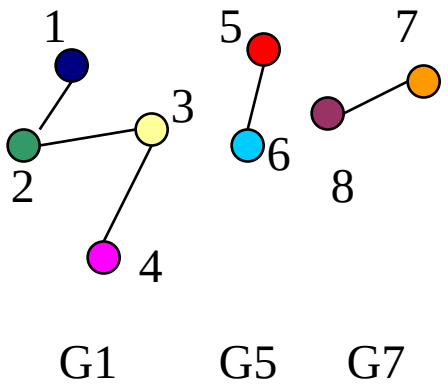
$$M \times v = v' \quad \text{where} \quad v_i' = \sum_{j=1}^{n} m_{i,j} \times v_j$$

- Three primitive graph mining operations
  - *combine2*(): Multiply $m_{i,j}$ and v_j
  - *combinAll*$_i$*()*: Sum n multiplication results
  - *assign*(): Update v_j

Graph Mining in Pegasus

- Graph mining is carried out by customizing the three operations
  - *combine2*()
  - *combineAll*()
  - *assign*()

Example: Connected Components



	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		
6					1			
7								1
8							1	

Source: Pegasus, Kevin Andryc, 2011

Graph Mining

Operations	**Plain M-V**	**PageRank**	**RWR**	**Diameter**	**Conn. Comp.**
combine2()	Multiply	Multiply with c	Multiply with c	Multiply bit-vector	Multiply
combineAll()	Sum	Sum with prob. of random jump	Sum with restart prob.	BIT-OR()	MIN
assign()	Assign	Assign	Assign	BIT-OR()	MIN

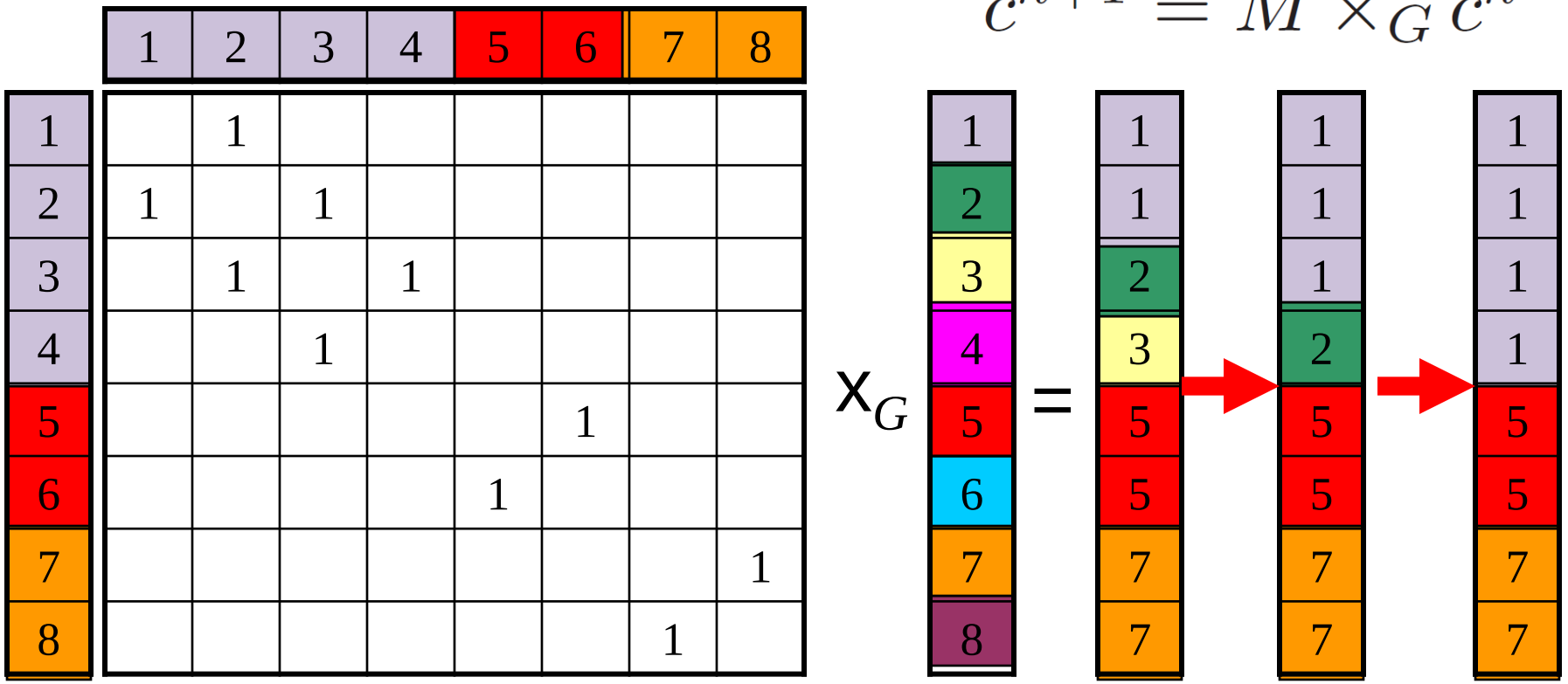
Source: Pegasus, Kevin Andryc, 2011

Pegasus

- Matrix based graph mining platform
- Support large scale graphs
- Many graph operations cannot be modeled by matrix-vector multiplications

Not a very natural programming model

$$c^{h+1} = M \times_G c^h$$



$$c_i^{h+1} = \texttt{assign}(c_i^h, \texttt{combineAll}_i(\{x_j \mid j = 1..n, \text{ and } x_j = \texttt{combine2}(m_{i,j}, c_j^h)\}))$$

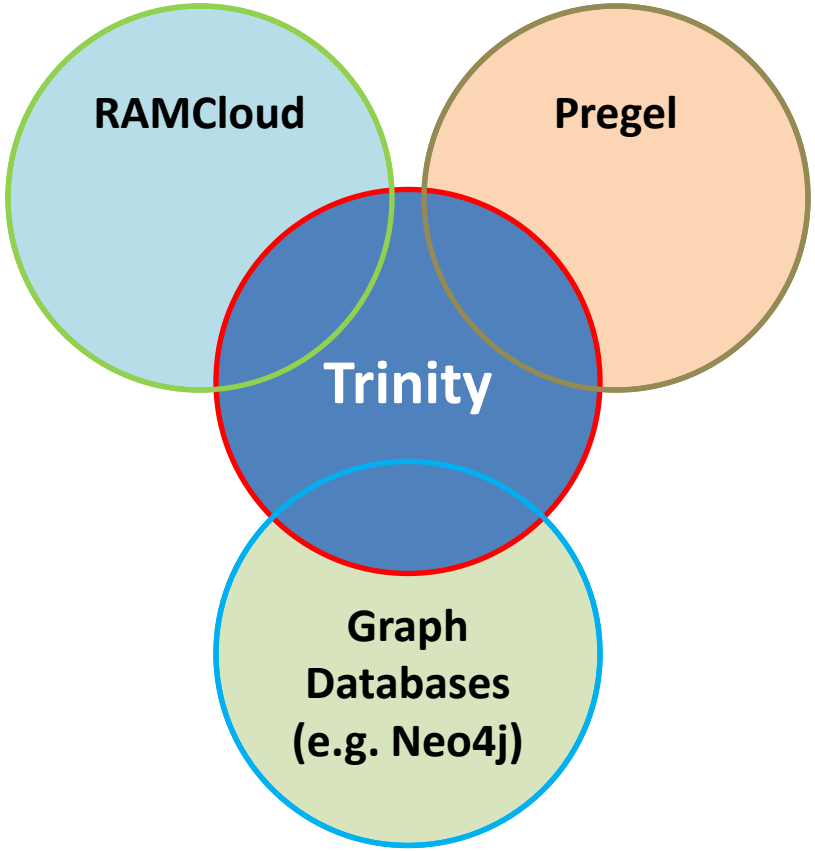
Adapted from: Pegasus, Kevin Andryc, 2011

Trinity: Distributed Graph Engine

Trinity

- A distributed, in-memory key/value store
- A graph database for online query processing
- A parallel platform for offline graph analytics

All-in-memory General Purpose Graph Engine



Memory Cloud

Distributed Memory Cloud

- Provides fast random access over large scale graphs

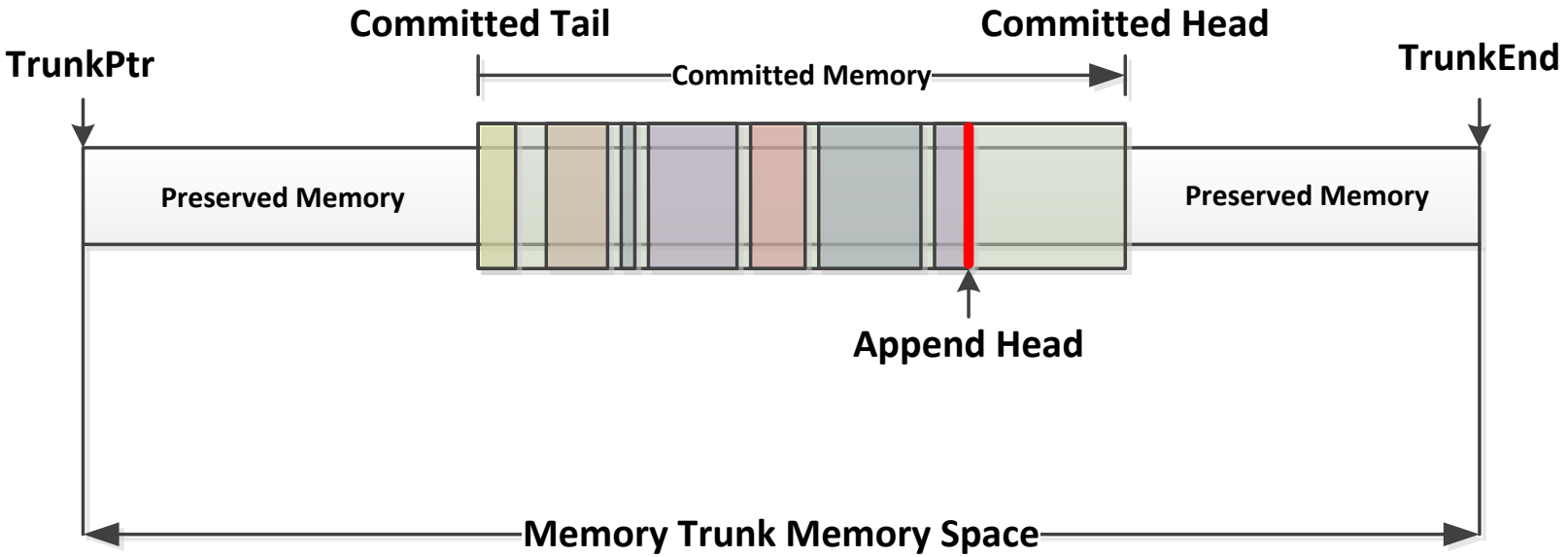
- Fast graph exploration can be implemented based on memory cloud

Memory Management

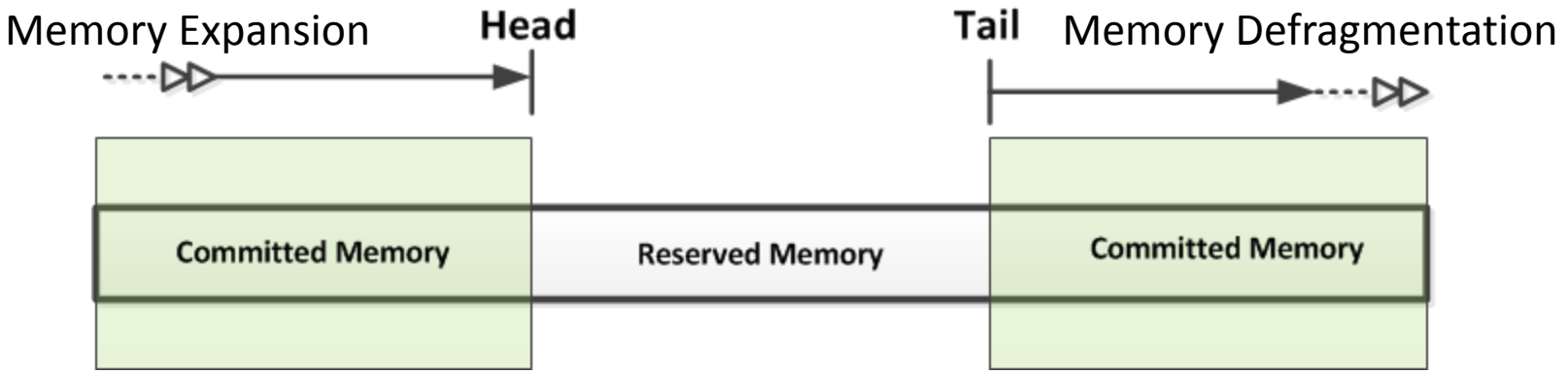
- Efficient memory management is crucial to an in-memory system

- Memory Management in Trinity
  - Ring Memory Management
  - Hash Memory Storage

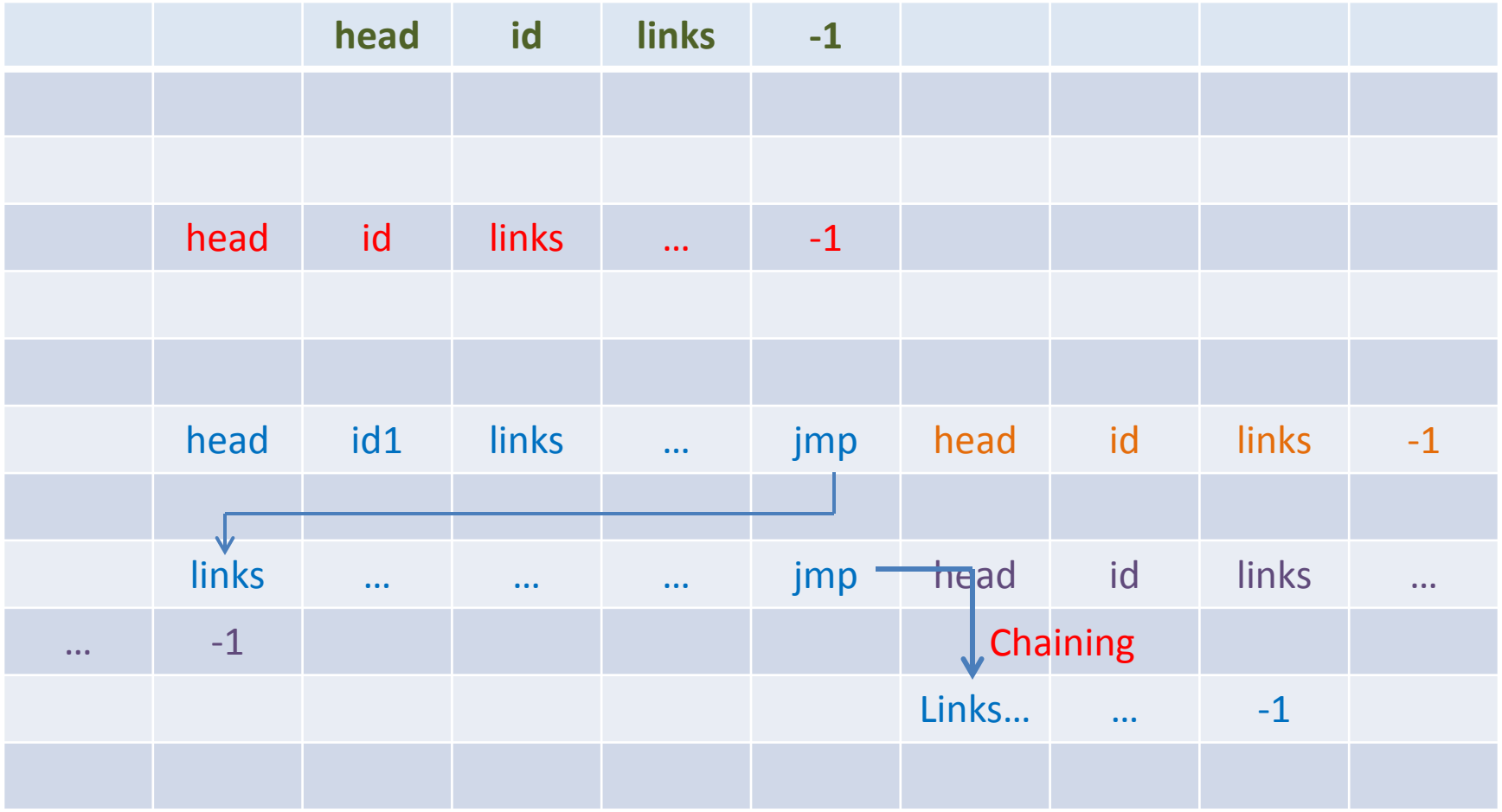
Ring Memory Management



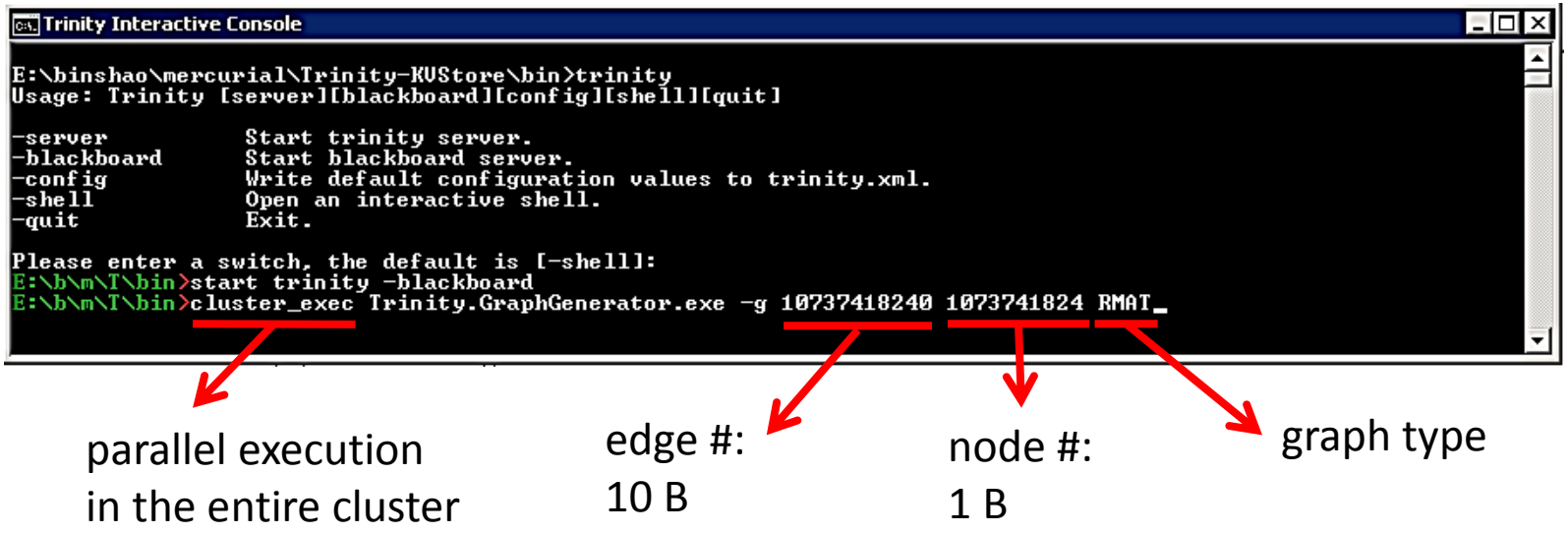
Circular Movement



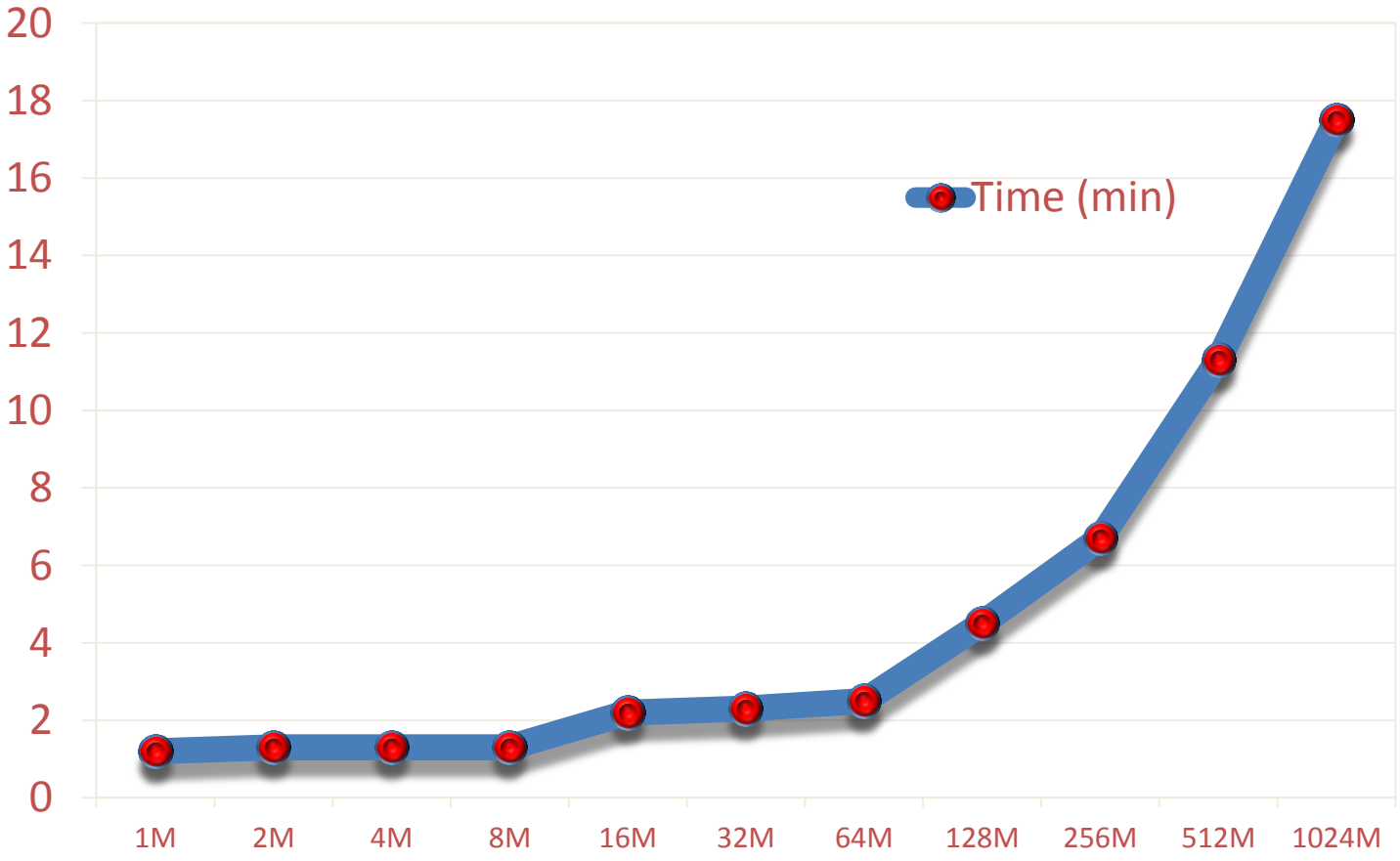
Hash Storage



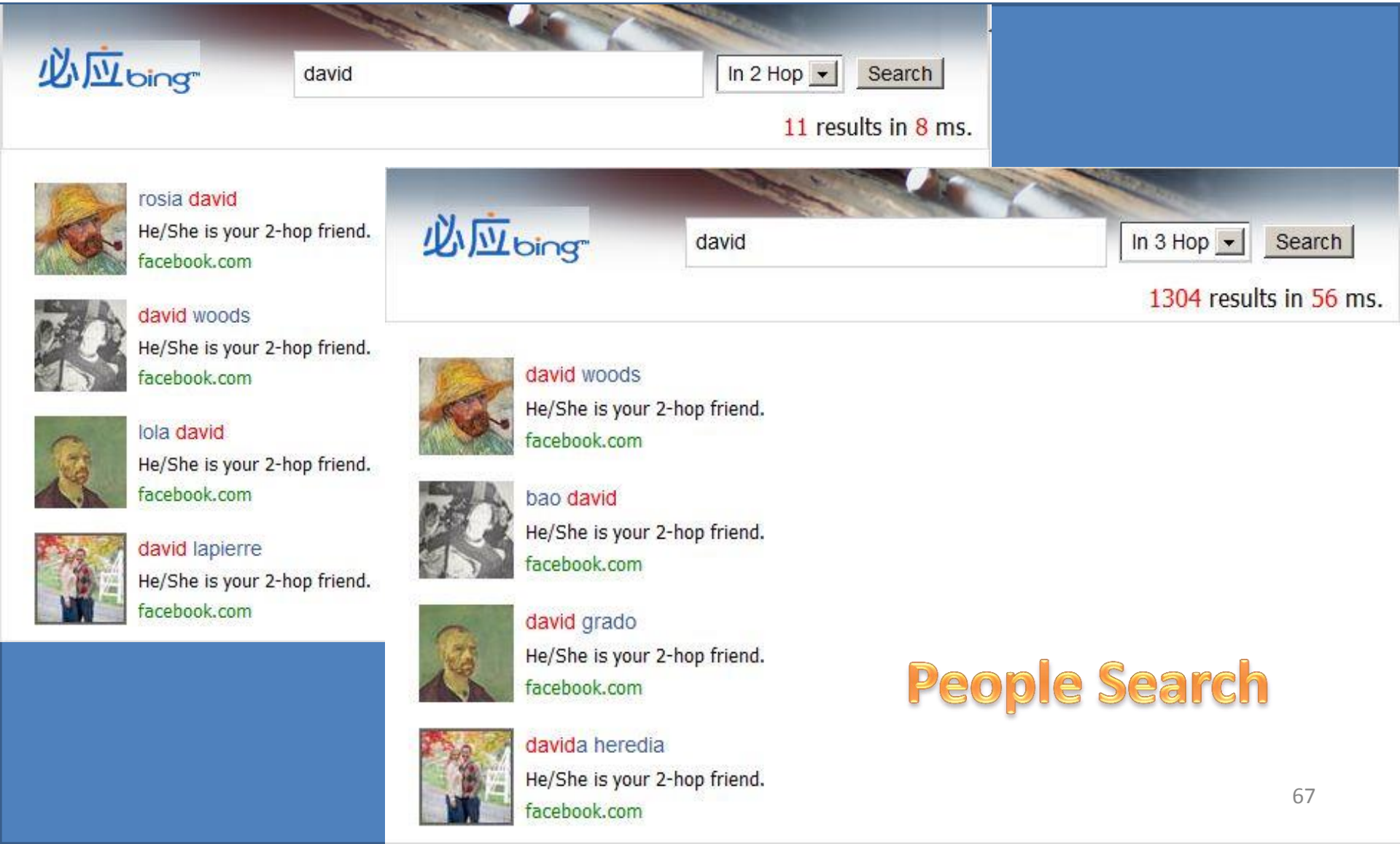
Graph Update: An extreme case



Case Study:
Trinity Graph Generation



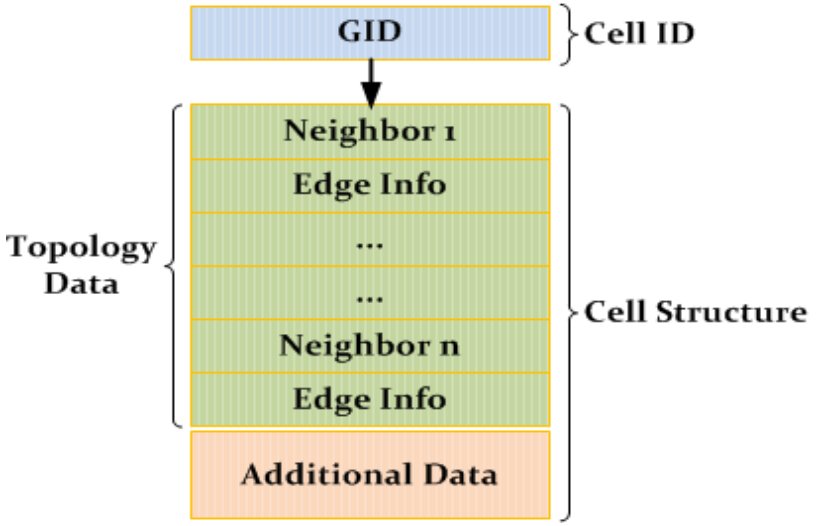
Example: People Search Query



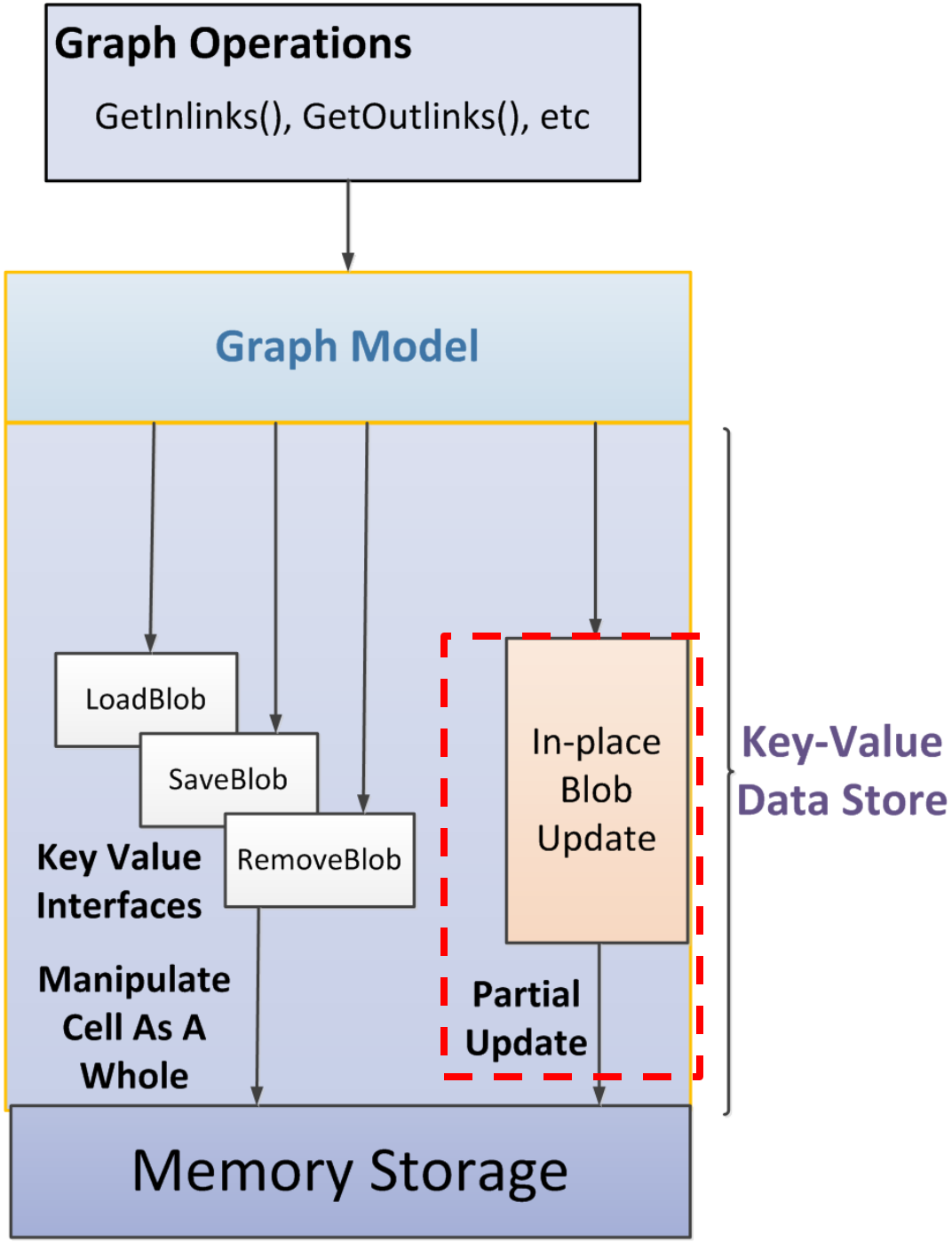
Cell-based Graph Model

Modeling a graph

- Basic data structure: Cell

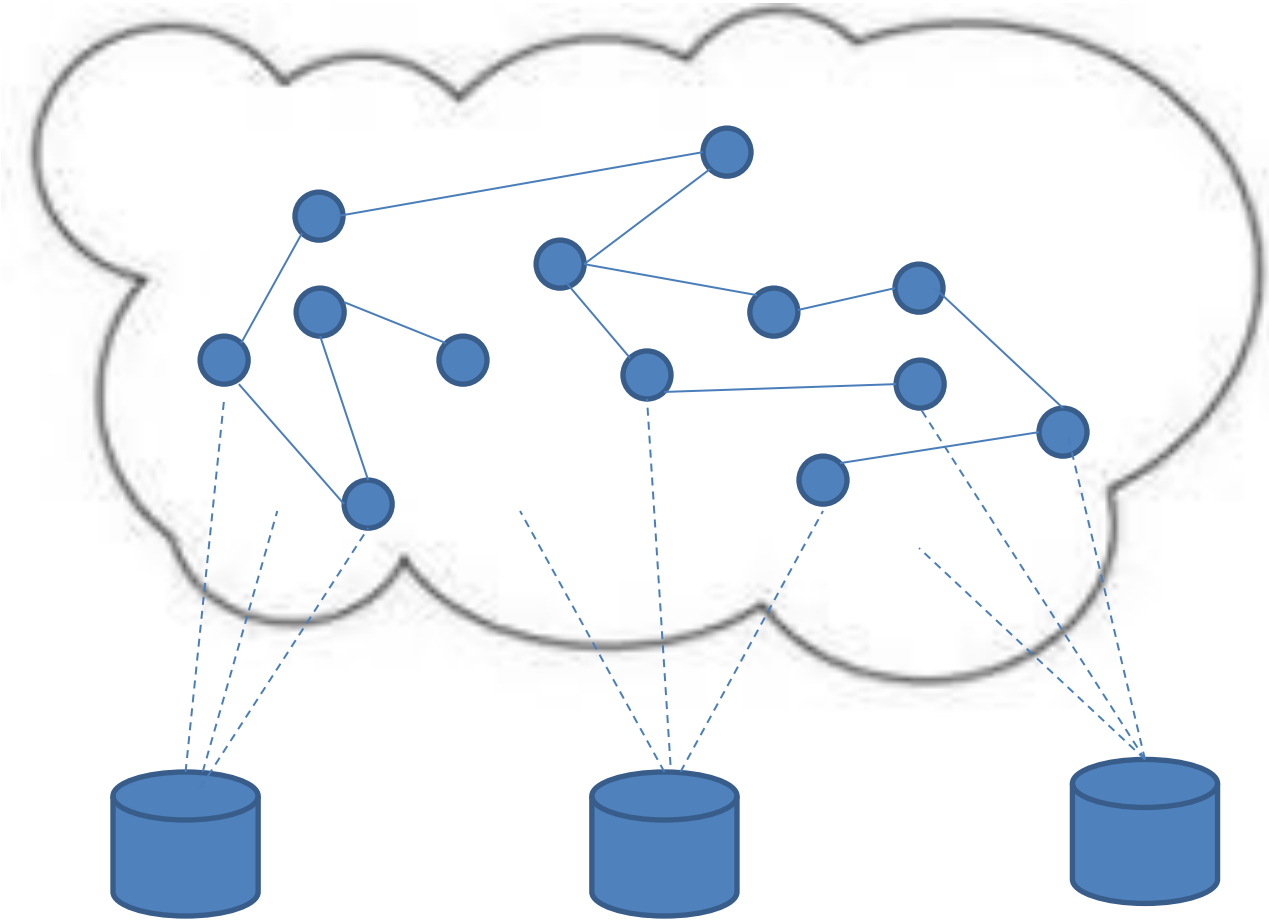


- A graph *node* is a Cell
- A graph *edge* may or may not be a Cell
  - Edge has no info
  - Edge has very simple info (e.g., label, weight)
  - Edge has rich info: an independent cell

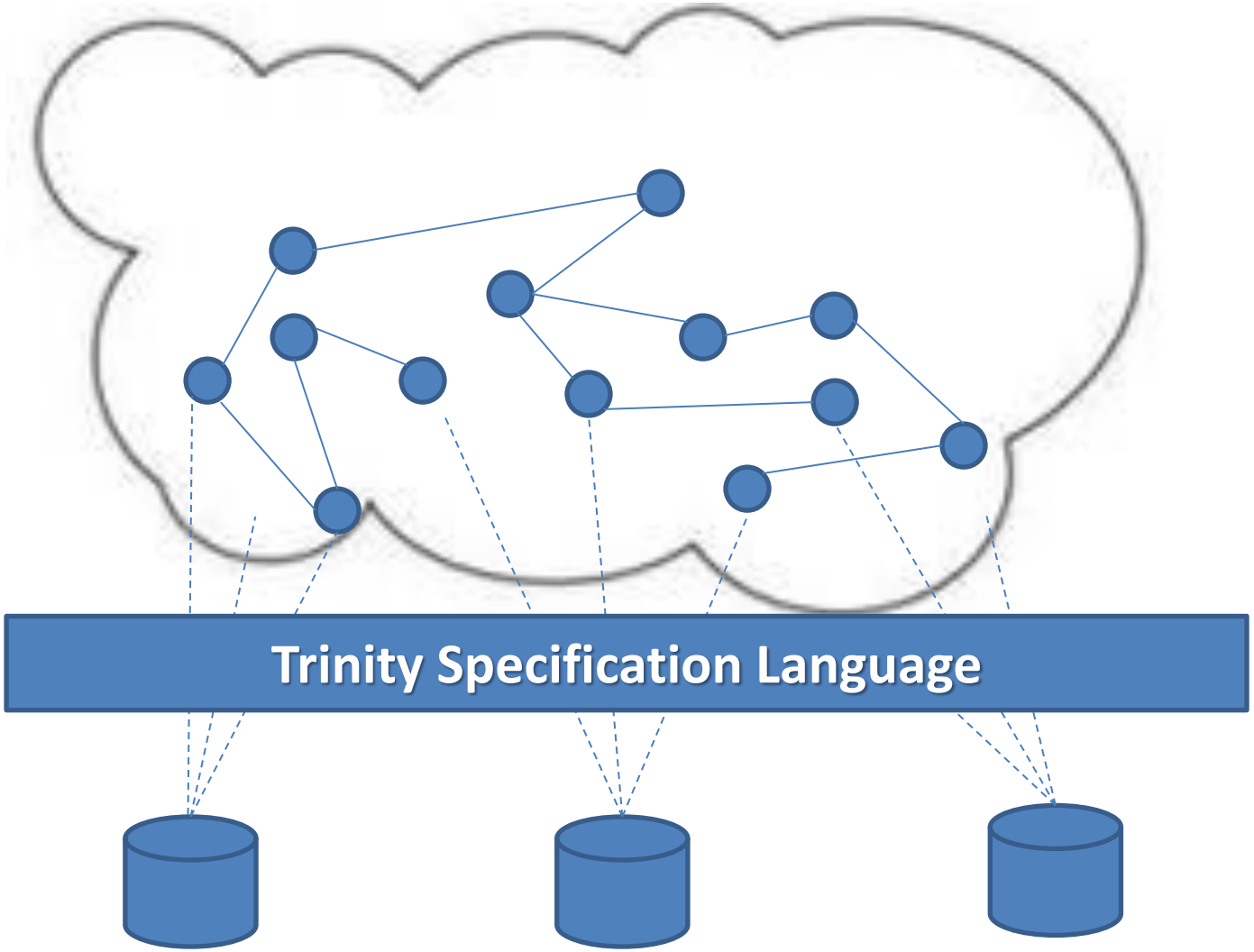

Graph Operations
GetInlinks(), GetOutlinks(), etc
Graph Model
LoadBlob
SaveBlob
RemoveBlob
Key Value Interfaces
Manipulate Cell As A Whole
In-place Blob Update
Partial Update
Key-Value Data Store
Memory Storage

Data Model

Memory cloud provides join-free fast exploration through relationships



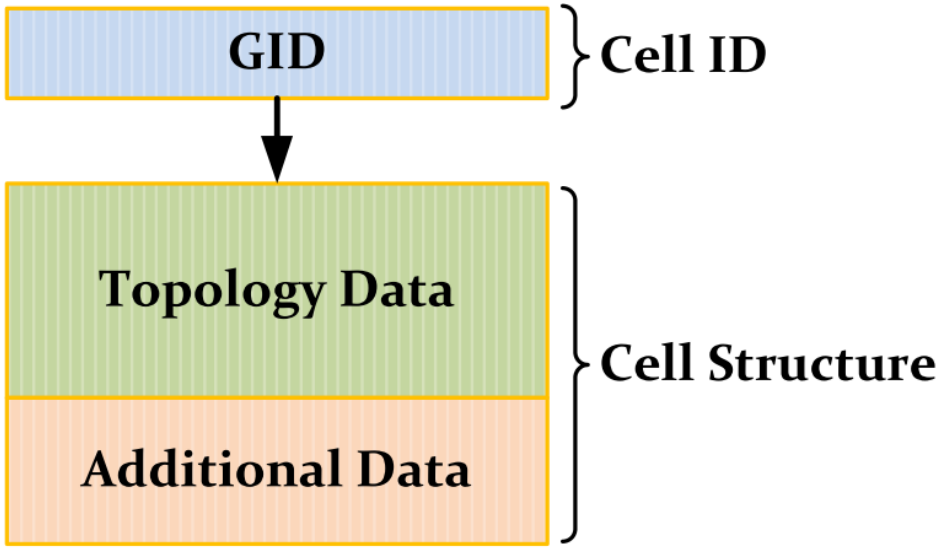
RDBMS provides additional storage and access methods, and persistence


Trinity Specification Language

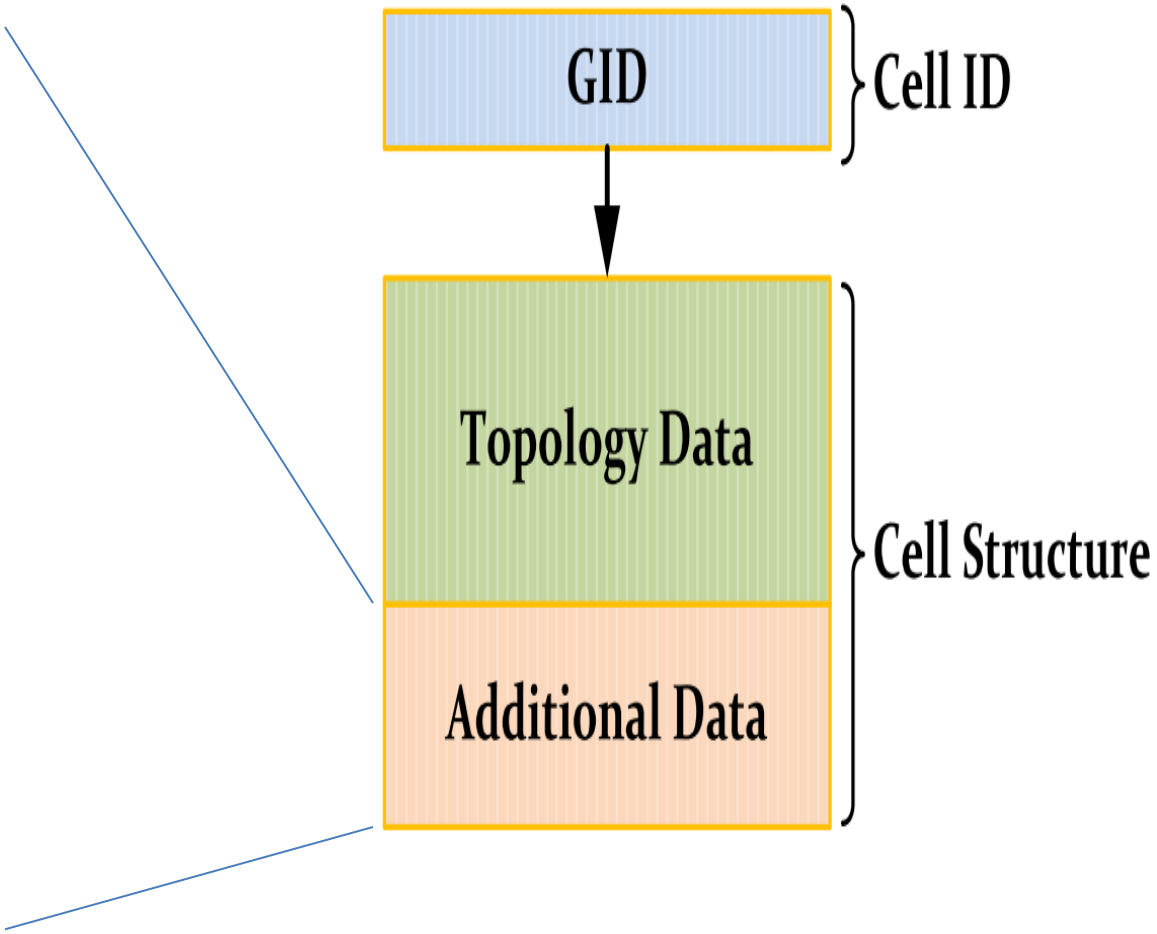
TSL

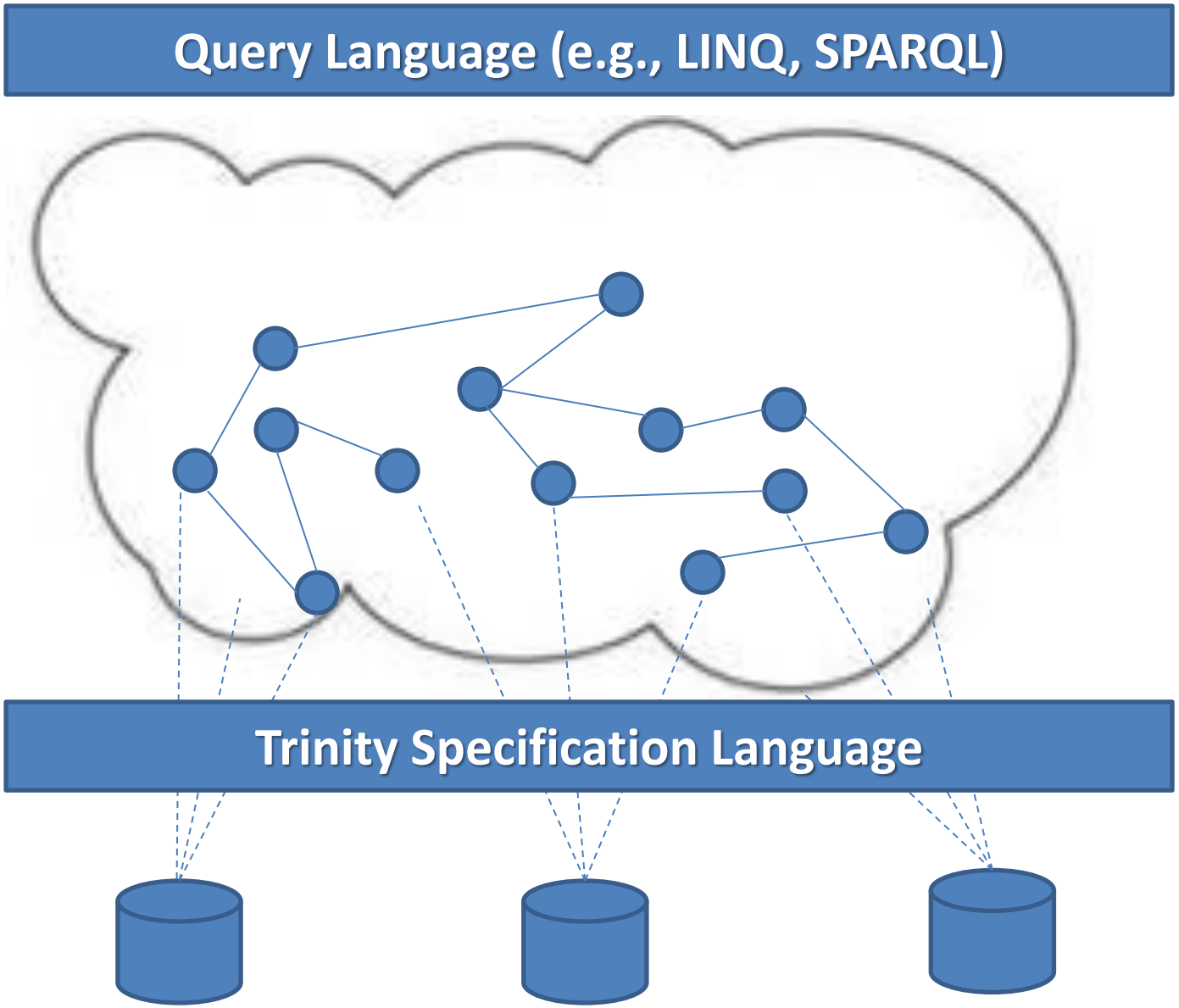
- Declaration of
  - Data schema and schema mapping
  - Indexing
  - Message passing protocols

Cell Structure

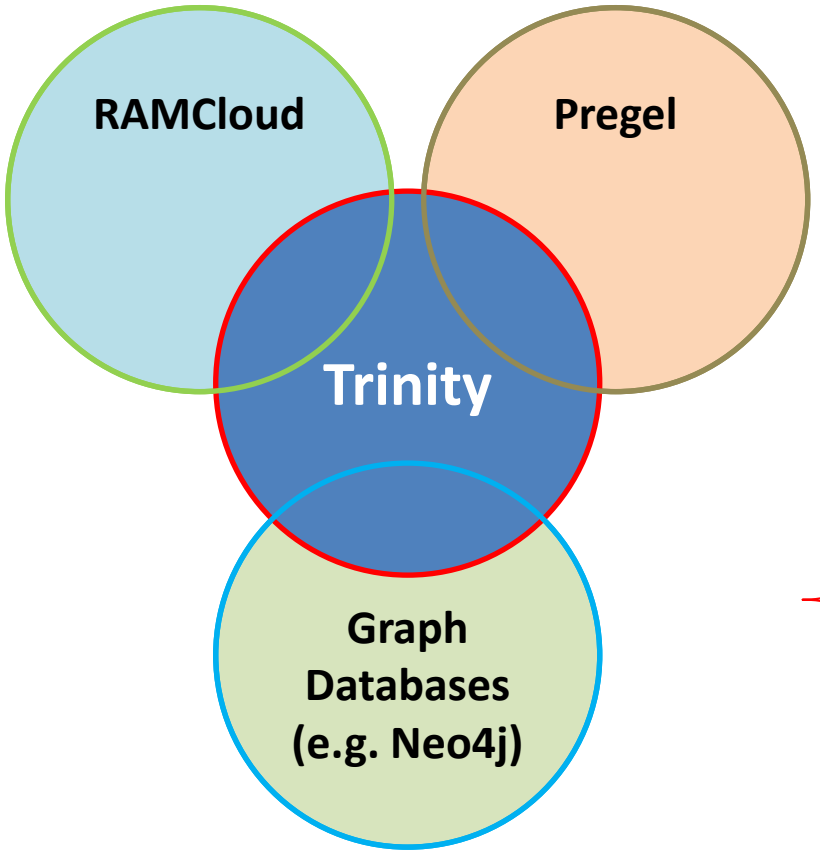


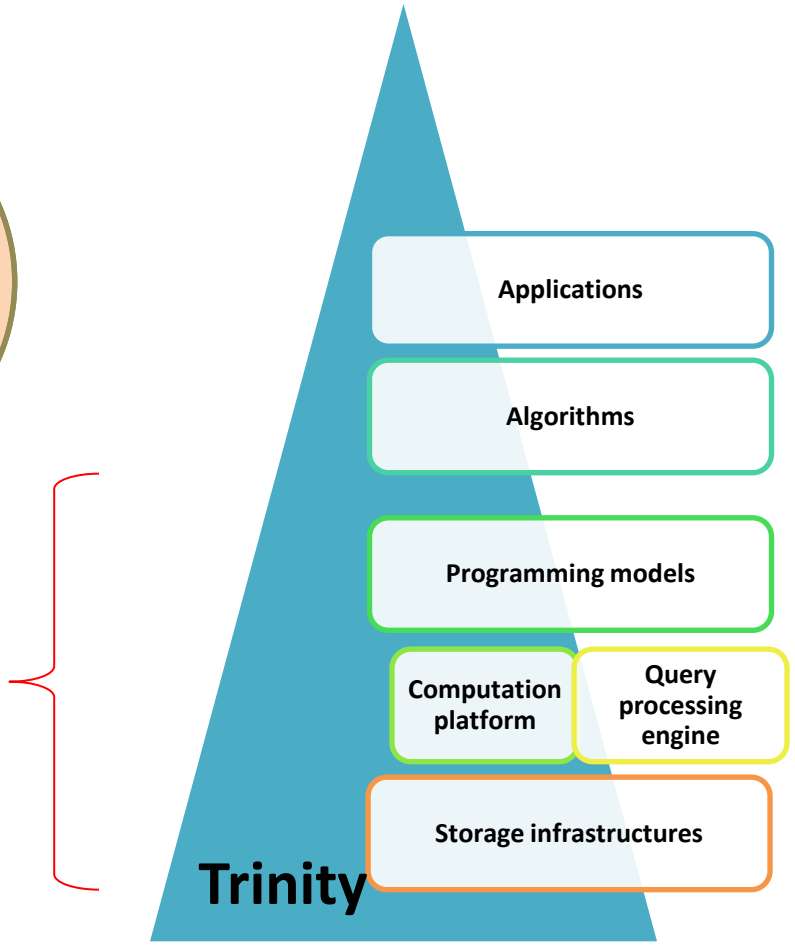
- Mapping
  - a field can reside on Trinity, on RDBMS, or on both
- Index
  - option of index: exact, full-text, spatial




Query Language (e.g., LINQ, SPARQL)
Trinity Specification Language

All-in-memory General Purpose Graph Engine





Summary

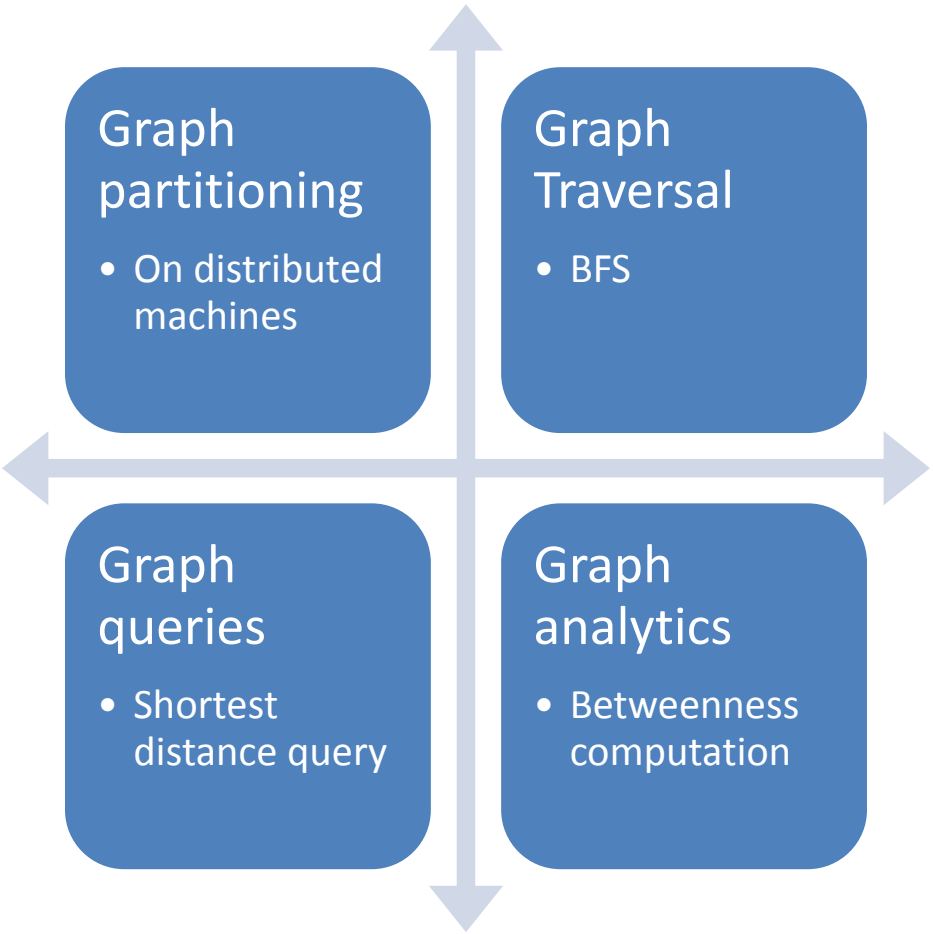
- Building a general purpose large scale graph system is important but very challenging

- Memory-based scale-out approach is very promising

Outline

- Large Graph Challenges
- Systems for Large Graphs
  - RDBMS, Map Reduce, Pregel, Pegasus, Trinity
- Key Graph Algorithms
  - Graph Partitioning, Traversal, Query, Analytics

Outline

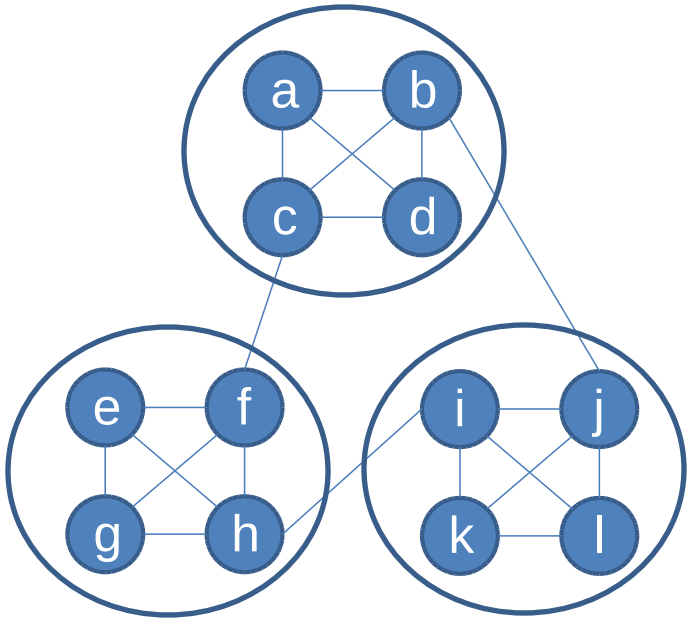


- Graph is big
  - billion node graphs
- Graph is distributed
  - Instead of centrically stored
- Solutions should be general enough to run on a general-purpose graph system

Graph partitioning

Graph Partitioning

- Problem definition
  - Divide a graph into k almost equal-size parts, such that the number of edges among them is minimized.
- Why?
  - Load balance
  - Reduce communication
- NP-complete [Garey74]
- Example: BFS on the graph
  - Best partitioning needs 3 remote accesses
  - Worst partitioning needs 17



Graph partitioning in current systems

- Default: *Random partitioning*
  - PBGL, Pregel, neo4j, InfiniteGraph, HyperGraphDB
  - Quality is significantly worse than a refined method
- No partitioning algorithm is supported at the system level
- PBGL and Pregel support *user-defined partitioning*

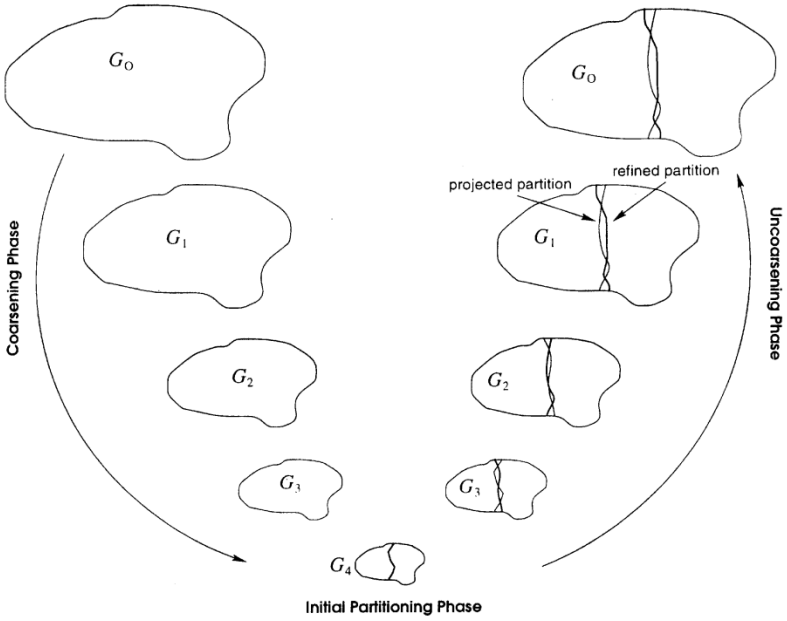
Existing graph partitioning algorithms

- Classical partitioning algorithm
  - Swapping selected node pairs: KL [kernighan 72] and FM [Fiduccia 82]
  - Simulated annealing based solutions [Johnson89]
  - Genetic algorithms [Bui96]
  - For small graphs (but with high quality)
- Multi-level partitioning solutions
  - METIS [Karypis95], Chaco [Hendrickson] and Scotch [Pellegrini96]
  - For million-node graphs
- Parallelized Multi-level partitioning solutions
  - ParMetis [Karypis98] and Pt-Scotch [Chevalier08].
  - For at most tens-of-million-node graphs.

- No good solutions for partitioning *billion-node graphs* on a *general-purpose* distributed system.

State-of-art method: Metis

- A multi-level framework
- Three phrases
  - Coarsening by maximal match until the graph is small enough
  - Partition the coarsest graph by KL algorithm [kernighan 72]
  - Uncoarsening

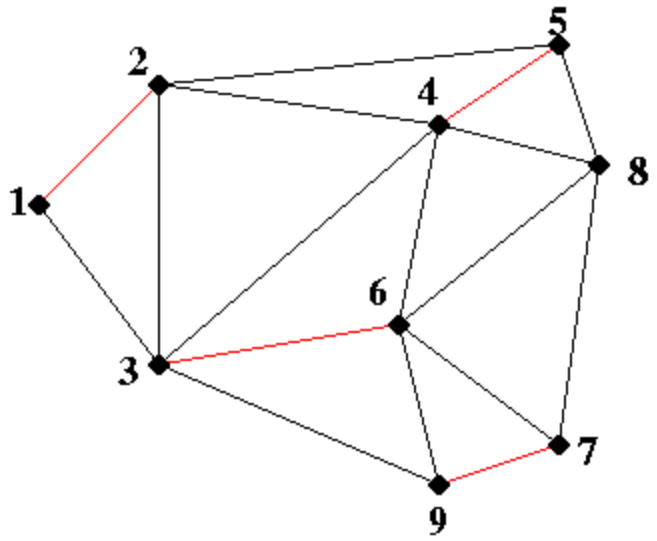


[Metis 1995]

Maximal Matching

- A matching of a graph G(V,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- A maximal matching of a graph G(V,E) is a matching E_m to which no more edges can be added and remain a matching
- Find a maximal matching using simple greedy algorithms

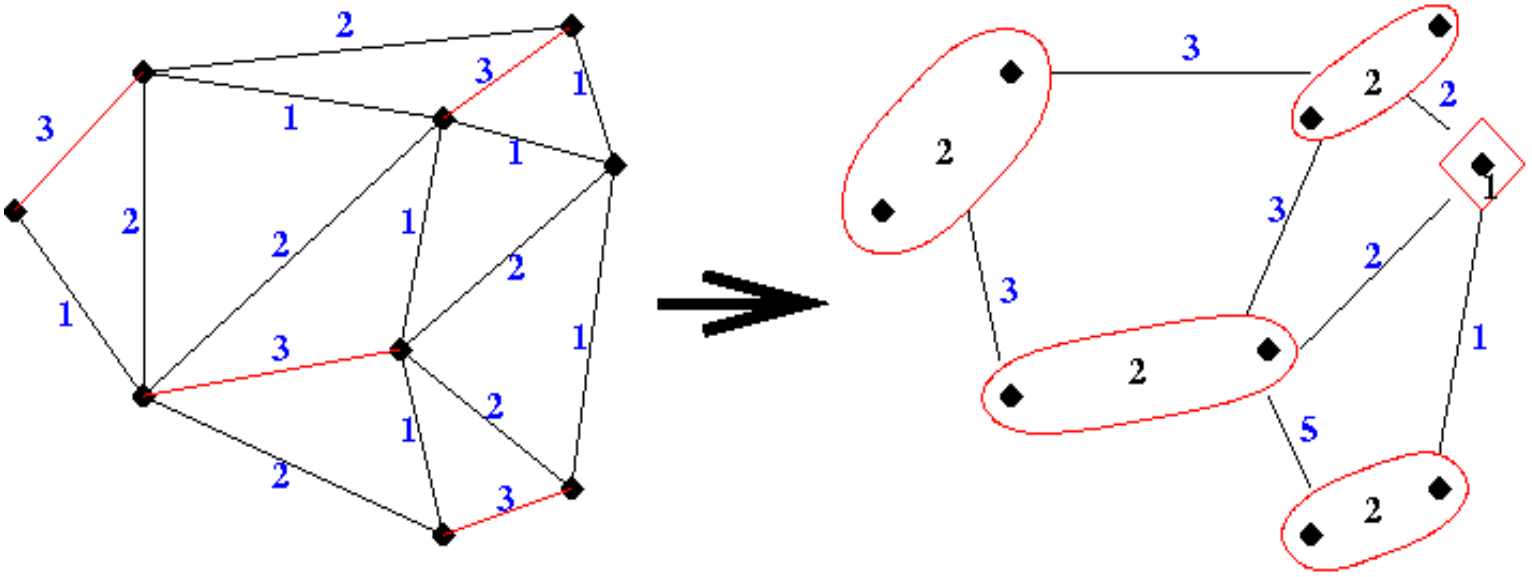
Maximal Matching - Example



Edges in read are a maximal matching

From James Demmel, http://www.cs.berkeley.edu/~demmel/cs267_Spr99

How to coarsen a graph using a maximal matching



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c, E_c)$

N_c is shown in red

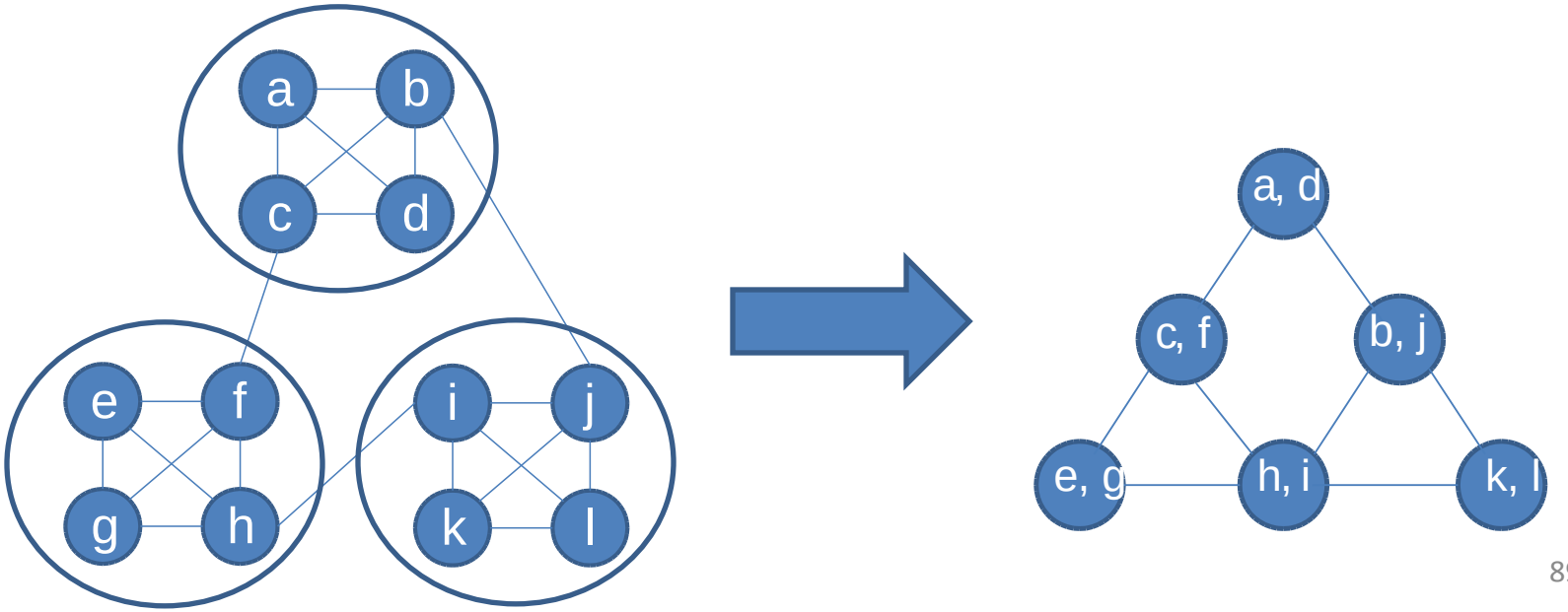
Edge weights shown in blue

Node weights shown in black

From James Demmel, http://www.cs.berkeley.edu/~demmel/cs267_Spr99

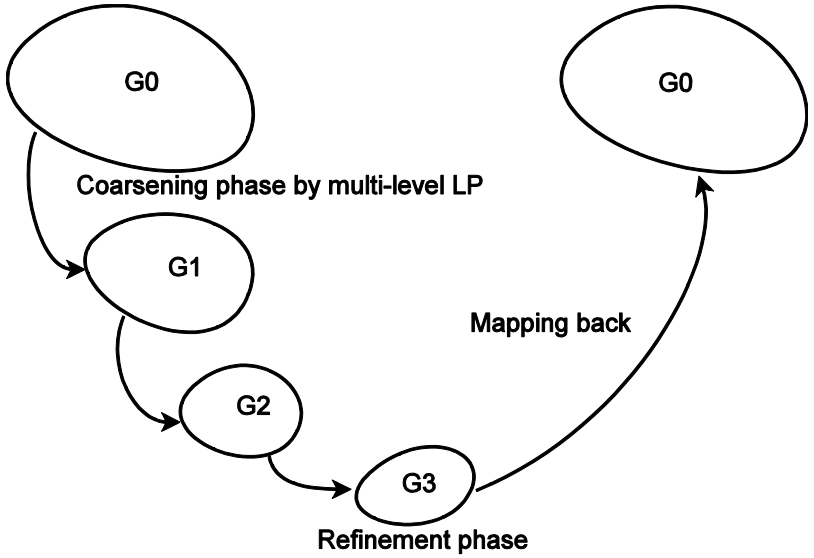
Coarsening in Metis

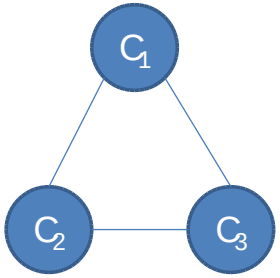
- Assumption of coarsening: An optimal partitioning on a coarser graph is a good partitioning in the finer graph
- It holds *only when node degree is bounded* (2D, 3D meshes).
- But real networks have *skewed degree distribution*



Possible solution : Multi-level Label Propagation

- Coarsening by label propagation
  - In each iteration, a vertex takes ***the label that is prevalent in its neighborhood*** as its own label.
- Lightweight
  - Easily implemented by message passing
  - Easily parallelizable
- Effective (for real networks)
  - Capable of discovering inherent community structures





How to handle imbalance?

- Imbalance caused by label propagation
  - Too many small clusters
  - Some extremely large clusters

- Possible solution:
  - First, limit the size of each cluster
  - Second, merge small clusters by
    - *multiprocessor scheduling* (MS)
    - *weighted graph partitioning* (WGP)

- For details:
  - How to partition a billion node graph, Microsoft Technical Report, 2012

Distance Oracle

Shortest distance query

- A basic graph operator
  - Used in many graph algorithms, including computing centrality, betweenness, etc.

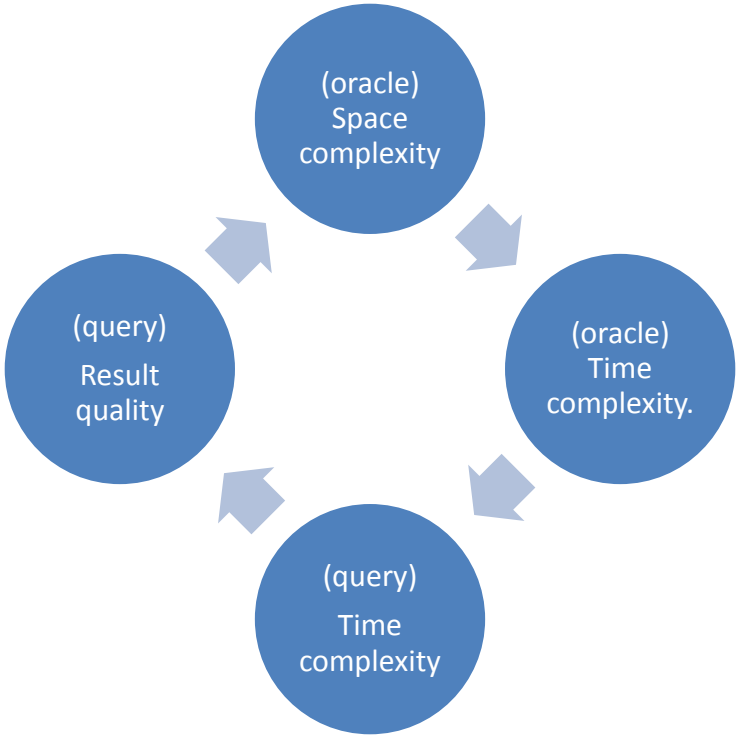
- Exact shortest distance solutions
  - Online computation
    - Dijkstra-like algorithms on weighted graph
    - BFS on unweighted graph
    - At least linear complexity, taking hours on large graphs
  - Pre-computing all pairs of shortest path
    - quadratic space complexity, prohibitive on large graphs

Distance Oracle

- A pre-computed data structure that enables us to find the (approximate) shortest distance between any two vertices in constant time.

- Oracle construction
  - Space complexity
    - linear, or sub-linear
  - Time complexity.
    - Quadratic complexity is unaffordable

- Query answering
  - Time complexity
    - *constant time*.
  - Quality
    - Approaching exact shortest distances



Current Solutions

- With performance bound:
  - Thorup-and-Zwick's distance oracle [Thorup01]
  - Reduce the construction time on weighted [Baswana06] or unweighted graphs [Baswana062],
  - Reduce space cost on power-law graphs [Chen09] or ER random graphs [Enachescu08].

- Heuristic approaches:
  - Sketch based [Potamias09, Gubichev10, Sarma10, Goldberg05, Tretyakov11]
  - Coordinate based [zhao2010,zhao2011]

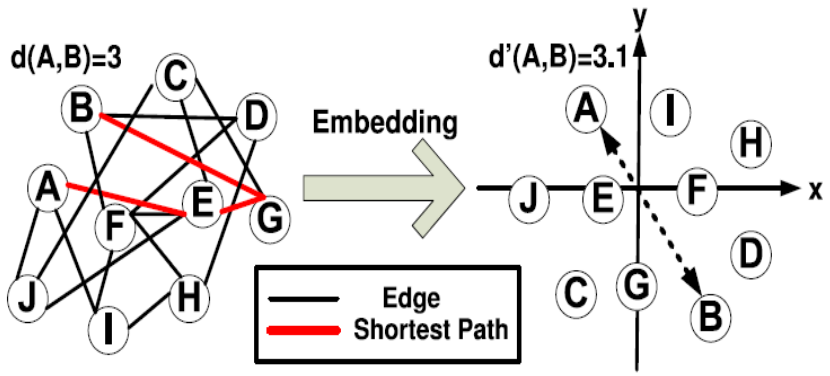
Thorup-and-Zwick's distance oracle

- A parameter k
- $O(kn^{1+1/k})$ space, can be constructed within $O(kmn^{1/k})$ time
- Distance query can be answered in $O(k)$ time with at most $2k - 1$ multiplicative distance estimation.

- When $k = 1$, the distance oracle returns the exact distance but occupies quadratic space
- When $k = 2$, the worst distance is 3-times of the exact distance and the oracle occupies $O(n^{1.5})$ space

Coordinate-based solution

- Mapping all vertices into a hyperspace
- Use the distance of two vertices in the hyperspace to approximate the shortest distance in the graph



[Zhao 2010, Zhao2011]

Coordinate-based solution

- Select landmarks (~100)
  - **Heuristics:** degree , betweenness, …
- Calculate the exact distance from each landmark to all other vertices by ***BFS starting from each landmark***
- Calculate the coordinates of landmarks by ***simplex downhill*** according to the precise distance among landmarks
- Calculate the coordinates of other vertexes by ***simplex downhill*** according to the distance from these vertex to each landmark

Sketch-based solution

- Basic idea
  - Create *a sketch of bounded size for each vertex*
  - Estimate *the distance using the sketch*

- Advantage
  - linear space
  - If the sketch encodes enough useful information, it can produce highly accurate answers in short time (in most cases in constant time).

Sketch-based solution

- Procedure
  - Select landmarks
  - Generate the shortest distances from each landmark to any other vertex
    - Sketch[u] ={(w_0, δ_0), . . . , (w_r, δ_r)}, where w_i is a vertex (called seeds) and δ_i is the shortest distance between u and w_i
  - Estimate the distance between vertices *u* and *v by the* minimal value of $d(u,w) + d(w, v)$ over all $w \in L$

Improvement on sketch-based solution

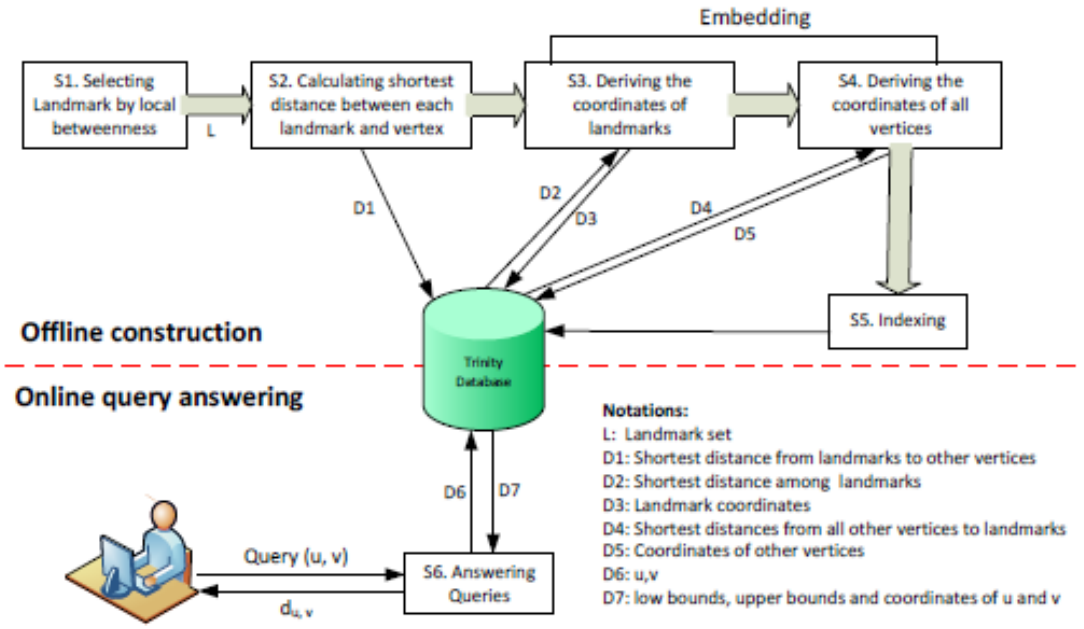
- Improve distance estimation
  - cycle elimination and tree-structured sketch [Gubichev10]
  - uses the distance to the least common ancestor of *u* and *v* in a shortest path tree [Tretyakov11]
- Improve landmark selection
  - optimal landmark selection is NP-hard
  - betweenness is a good landmark [Potamias09]
  - randomized seed selection [Sarma10]

Distance oracle summary

- Accuracy may be compromised
  - Node degrees, instead of their centrality, is usually used as a criterion for landmark selection to reduce the cost.
- Some distance oracles take very long time to create.
  - For example, in [Tretyakov11], it takes 23h to approximate betweenness for a 0.5 billion node graphs [Tretyakov11] even on a 32-core server with 256G memory.
- None of the previous distance oracles is designed for distributed graphs.
- Can hardly scale to billion node graphs on a general purpose graph system

Scale to billion node graphs with high accuracy

- Smart landmark selection
  - in local graphs
- Smart distributed BFS
- Smart answer generation rule



For details:

Towards a Distance Oracle for Billion Node Graphs, Microsoft Technical Report, 2012

Distributed BFS

Asynchronized BFS

- Asynchronized BFS (Bellman-ford)
  - In each iteration, update each vertex's distance as the minimal distance of its neighbors plus one
  - Any time when a vertex distance is updated, it will trigger all its neighbors to update their distance
  - Until no distance update

- O(|V||E|), but flexible, allows fine-grained parallelism

Level-synchronized BFS [Andy05]

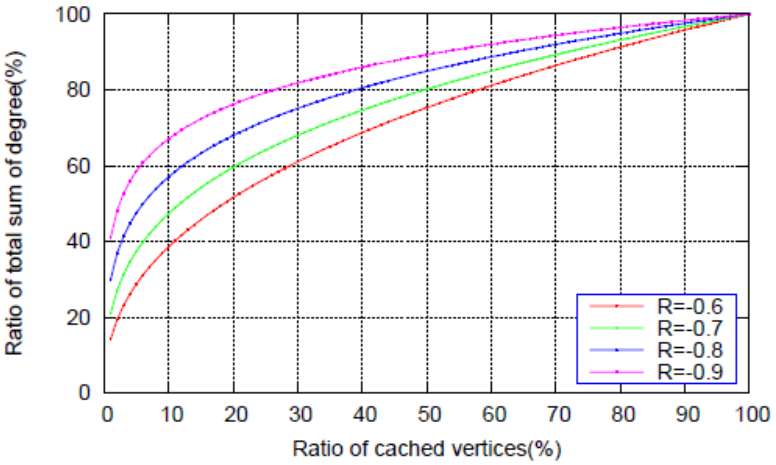
- Explore from a node level by level
- ***Iterate:***
  - Each vertex of level *x* sends distance update messages to their neighbors
  - Each vertex waits for messages for itself. If its distance is still unknown, update its distance as x+1.
  - ***Synchronize at the end of each level***
- O(E) complexity

Possible Optimizations

- Observation:
  - Vertices of large degree are frequently visited even their distances to the source have already been computed.
- Optimization:
  - Cache the distances of large degree nodes on each machine
- "80/20" rule in real graphs
- Results on Trinity
  - 50% savings

- Scale-free graph

$$d(v) = \frac{1}{N^R} r(v)^R$$



For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012

Betweenness computation

Betweenness computation

- Betweenness counts the fraction of shortest paths passing through a vertex
- Applications of betweenness
  - Vertex importance ranking
  - Landmark selection in distance oracle
  - Community detection

$$C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

- Where, σ_{st} denotes the number of shortest paths from s to t, $\sigma_{st}(u)$ denote the number of shortest paths from *s* to *t* that pass through u

Betweenness computation on large graphs

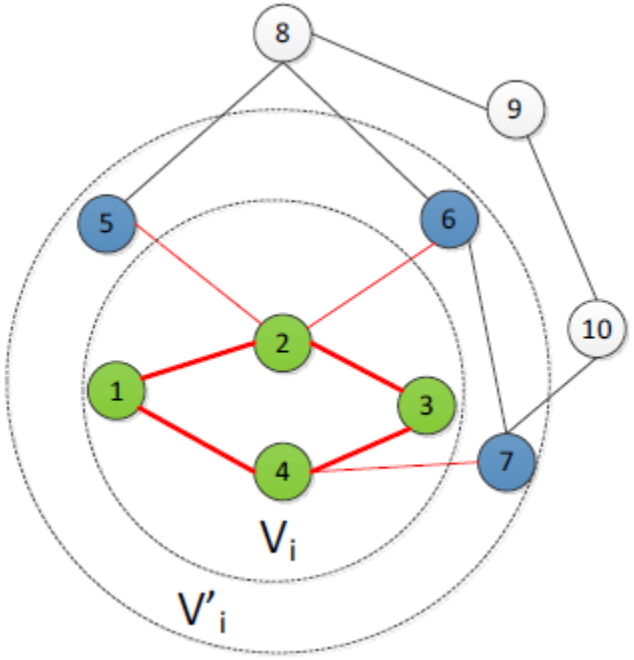
- Exact betweenness costs O(nm) time, unacceptable for large graphs [Brandes01]
- Lightweight approximate betweenness
  - Count shortest paths rooted at sampled vertices [Ercsey10]
  - Count shortest path with limited depth [Bader06][Brandes07]
- Parallelized exact betweenness [Bader06, Madduri09, Edmonds10, Tan09, Tu09]
  - On massive multithreaded computing platform [Bader06, Madduri09],
  - On distributed memory system [Edmonds10]
  - On multi-core systems [Tan09, Tu09].

A distributed view of the graph

Local Graph V_i

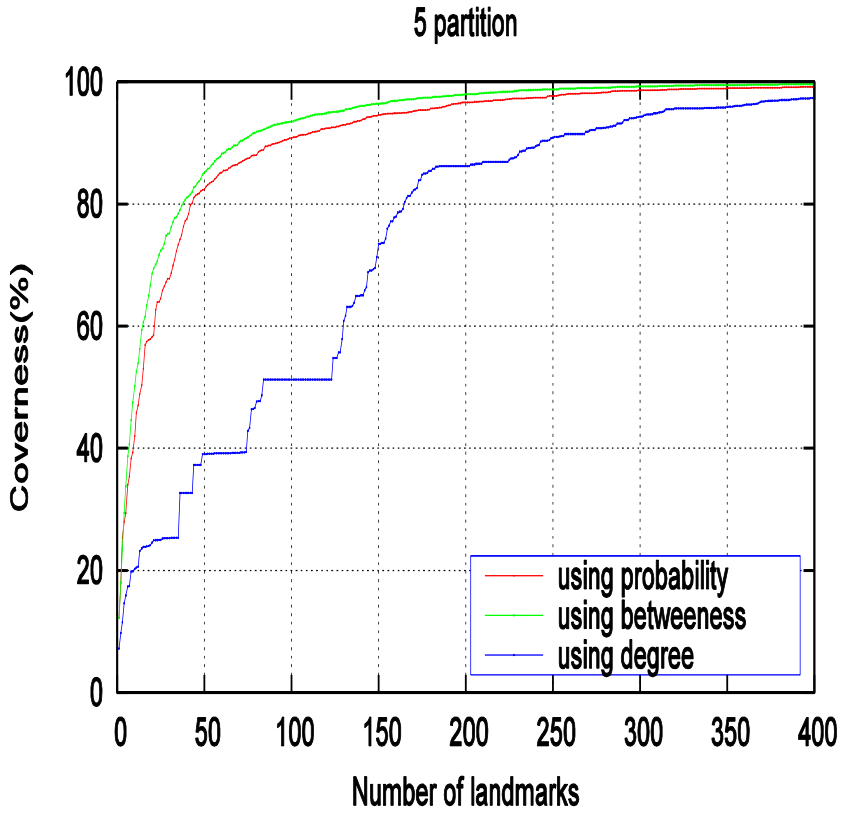
Extended Local Graph V_i'

Graph G



A lightweight Distributed betweenness approximation

- On distributed platform
  - Compute the shortest paths within each machines
  - Local shortest path with two ends both in the local core has high quality
  - Sample the local shortest with the probability proportional to its quality
  - Use the high-quality local shortest path to approximate exact betweenness



For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012

Distributed computation on graphs

- Computation carried out on a single partition
  - Density of the graph, etc.
  - Derive global results from results on a single partition
- Computation carried out on each single partition
  - Betweenness, connected components, etc.
  - Derive global results by merging results from single partitions
- Network communication reduced.

Local Clustering

Motivation

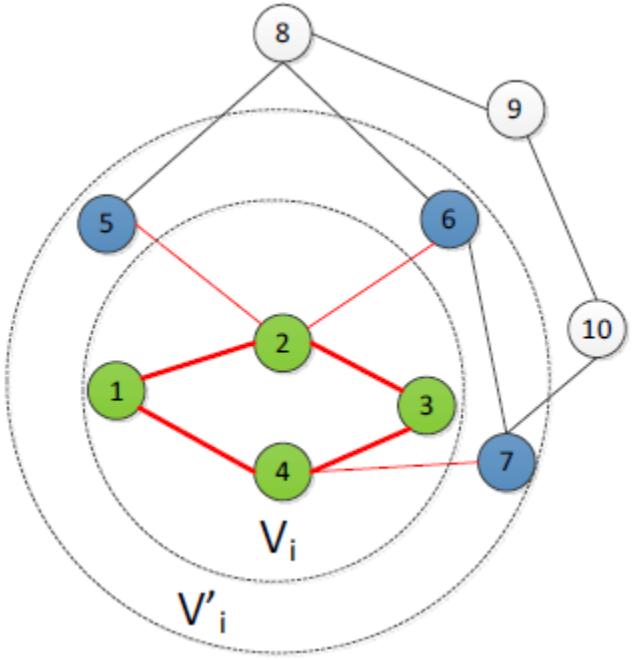
- In distributed graph systems, remote communications are costly
- To save remote access, we may cache information about remote neighbors
- Local vertices sharing similar remote neighbors should be processed together

A distributed view of the graph

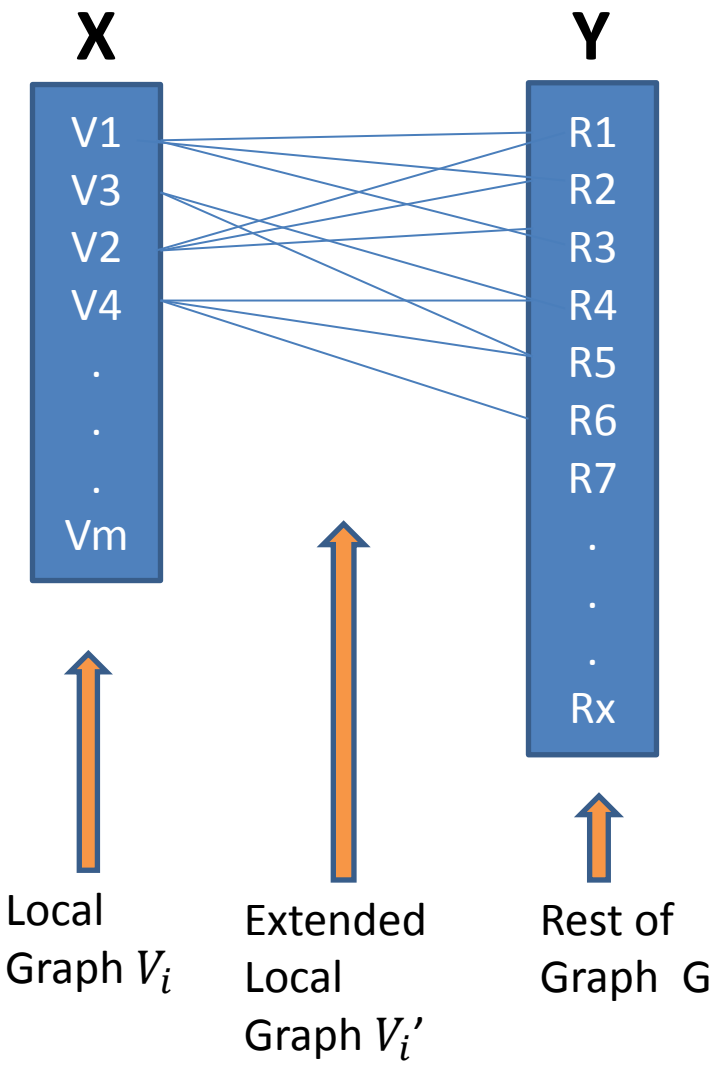
Local Graph V_i

Extended Local Graph V_i'

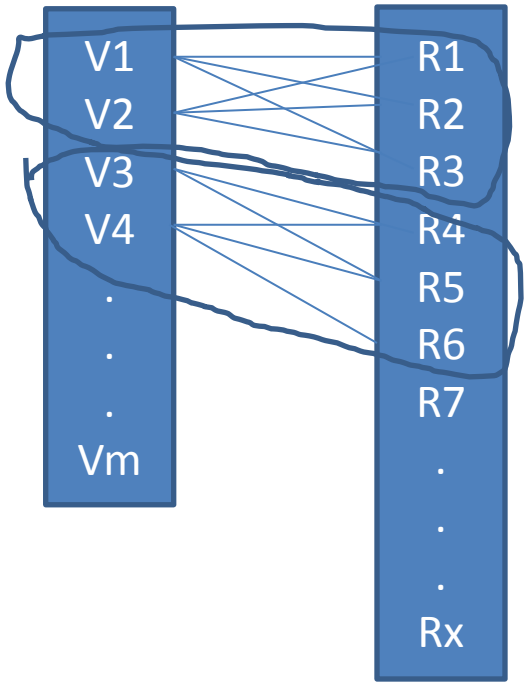
Graph G



A bipartite view



Clustering the bipartite view



Analysis

- G(V, E) is partitioned across k machines.
- Each machine has a block B of the graph
- Before reordering
  - Let d be the average remote degree, c be the average size of a remote message
  - On the average, we need
    - **O(d|B|) number of remote access**
    - **O(d|B|c) communication volume**
- After reordering
  - In the best case (all local vertices sharing the same remote neighbors)
    - we need **O(d)** remote access
    - we need **O(dc)** communication volume
  - A multiplication factor |B| is saved

Related works

- Many bipartite graph algorithms
- Poor scalability
  - Matrix based method, spectral clustering …

Design constraint

- Cache size constraint S
- For a block, it is possible that |R(B)|>S
  - R(B) are the remote neighbors of B
- We cache at most S vertices in R(B)

Problem definition

- We have $G'(V_i \cup V - V_i, E')$. Let X = V_i, and Y = $V - V_i$
- Partition X into A_1 … A_t and Y into C_1 … C_t such that $|C_i| \leq S$, and maximize $\sum_i E(A_i, C_i)$
  - $E(A_i, C_i)$ = #edges between A_i and C_i
- NP-hard
  - since bipartite graph partitioning is NP-hard

Baseline methods

- Using existing partitioning methods
  - such as METIS

- Challenges:
  - How to set the # of partitions?
  - Balance: How to ensure each $|Y_i| \leq S$?

of Partitions

- For a bipartite graph G and a number t, let Q(t) be the minimal number of cross edges among all possible t-partitioning on G

- Focus on $\frac{|Y|}{S} \leq t \leq 2\frac{|Y|}{S}$
  - Q(t) is not necessarily a monotonic
  - Enumerate t from $\frac{|Y|}{S}$ to $2\frac{|Y|}{S}$ with increment Δt
  - Run METIS for each t

of Partitions

- Why not beyond $2\frac{|Y|}{S}$?
- Because t $\geq 2\frac{|Y|}{S} \rightarrow Q(t) \leq Q(t+1)$
  - Suppose $Q(t) > Q(t+1)$
  - There must exist two clusters Y_i and Y_j such that $|Y_i| \leq \frac{S}{2}$ and $|Y_j| \leq \frac{S}{2}$
  - We can merge Y_i and Y_j to construct a valid solution for t with at least the same quality

Balance

- Naive Metis is a balanced graph partitioning solution
- We only seek balance on the Y part
- Solution: vertex-weighted graph partitioning
  - Assign weight 0 to each vertex in X
  - Assign weight 1 to each vertex in y

Disadvantage

- Effectiveness
  - Too restrictive on balance
    - metis tends produce Yi of same size
    - we just need each |Yi| is less than S
  - Approximate balance
    - Solution produced by METIS may violate the restriction
- Efficiency
  - METIS does not support billion node graphs

Framework of our solution

Step 1 : Generate an initial partitioning with each $|Y_i| \leq S$

– random partitioning, for large graph

Step 2: Iteratively improve the partitioning

1. Relabel each v in X by the labels of its neighbors in Y
2. Relabel each v in Y by the labels of its neighbors in X

Relabel X

- Relabel v in X greedily
  - Choose the most popular label in v's neighbors as its new label
  - When there are more than one such most popular labels, we arbitrarily select one
  - Complexity: Linear with #edge

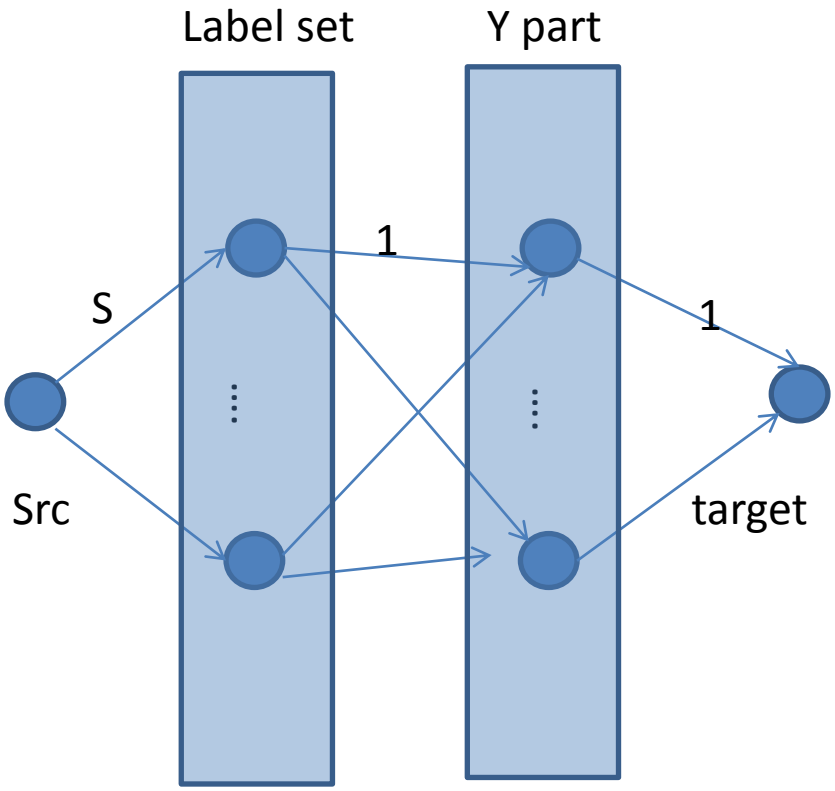
Relable Y

Problem:

- Let L={L1,…Lt} be the labels of X. Assign a label in L to each vertex in Y, such that
  - (quality) the number of edges among different labels are minimized
  - (constraint) each label is used by Y no more than S times

Solution: Maximal Flow Maximum Cost (MFMC)

- Cost
  - $Cost(L_i, v)$ denotes $v \in Y$ being assigned label l_i
  - It equals the number of edges between v and vertices in X with label L_i
- Capacity
  - From source to labels: S
  - From labels to Y: 1
  - From Y to target: 1



How to solve MFMC?

- Cycle Canceling: progressive solution [Ref]
  - Motivation: Directly improve the existing solution
  - Remove one cycle with accumulated w is equivalent to remove w cross edges from the original graph
  - In the worst case, we need O(M) cycle removing
  - Cost to find each cycle: $O(|L|^2|Y|)$
    - Note that our graph is bipartite
    - Using bellman-ford to find cycles with positive weight
      - Its complexity is O(DL|Y|), where D is the maximal shortest path length.
    - We can prove that in the bipartite graph constructed here, D<=2#labels+1
    - Hence, Motivation: Directly improve the existing solution
  - Overall cost: $O(|L|^2|Y|M)$
- Where, M is the number of edges of meta bipartite graph
- Prons
  - Optimal for reliable Y
- Cons
  - costly

Summary of key algorithms

- Most existing graph solutions are designed for centralized platforms
- Power of distributed computing has not been fully exploited for solving challenging graph problems
- Properties of distributed graphs have not been fully explored yet
- Previous existing solutions can hardly scale to real large graphs, especially billion node graphs

References-1

- [Newman] M. E. J. Newman et al., “Finding and evaluating community structure large networks,” Phys.Rev.E, vol. 79, p. 066107.
- [Leung] I. X. Y. Leung et al., “Towards real-time community detection in large networks,” *Phys.Rev.E*, vol. 79, p. 066107
- in networks,” Phys. Rev. E, vol. 69, no. 2, p. 026113.
- [Sozio10]M. Sozio et al., “The community-search problem and how to plan a successful cocktail party,” in *KDD’10*.
- [Clauset05]A. Clauset., “Finding local community structure in networks,” Phys. Rev. E, vol. 72, p. 026132. 2005
- [Bagrow08] Bagrow, James P., “Evaluating local community methods in networks” *J. Stat. Mech.*, vol. 2008, p. 05001. 2008
- [zhao 2010]X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao, “Orion: Shortest path estimation for large social graphs,” in *Proc. of WOSN*, Boston, MA, June 2010.
- [zhao 2011]X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao, Fast and Scalable Analysis of Massive Social Graphs, arXiv:1107.5114v2
- [atish 2011] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In Proceedings of WSDM'2010. pp.401~410
- [nweman2005]M. E. J. Newman, **Power laws, Pareto distributions and Zipf's law,** Contemporary Physics 46, 323-351 (2005)
- [metis1995] G. Karypis et al., “Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0,” Tech. Rep., 1995.
- [kl1972] B. Kernighan et a., “An efficient heuristic procedure for partitioning graphs,” *Bell Systems Journal*, vol. 49, pp. 291–307, 1972.
- [Sun2012] Z. Sun et al., “Efficient Subgraph Matching on Billion Node Graphs”, in PVLDB 2012.
- [RAMCloud] J. Ousterhout, et al., The case for RAMClouds: scalable high-performance storage entirely in dram. SIGOPS Oper. Syst. Rev., 43:92–105, January 2010.

References-2

- [andy05] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L, SC'05
- [Fal99]M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, Aug. 1999.
- [Garey74] M. R. Garey et al., "Some simplified np-complete problems," in *STOC '74*.
- [kernighan72] B. Kernighan et a., "An efficient heuristic procedure for partitioning graphs," *Bell Systems Journal*, vol. 49, pp. 291–307, 1972.
- [Fiduccia82] C. M. Fiduccia et al., "A linear-time heuristic for improving network partitions," in *DAC '82*.
- [Johnson89] D. S. Johnson et al., "Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning," *Oper. Res.*,vol. 37, pp. 865–892, 1989.
- [Bui96] T. N. Bui et al., "Genetic algorithm and graph partitioning," *Computers, IEEE Transactions on*, vol. 45, no. 7, pp. 841 –855, 1996.
- [Hendrickson] B. Hendrickson et al., "The chaco user's guid, version 2.0," *Technical Report SAND94-2692, Sandia National Laboratories*.
- [Karypis 95] G. Karypis et al., "Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0," Tech. Rep., 1995.
- [Pellegrini 96] F. Pellegrini et al., "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *HPCN Europe '96*
- [Karypis98] G. Karypis et al., "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *J. Parallel Distrib. Comput.*, vol. 48, pp. 71–95, 1998.
- [Chevalier08] C. Chevalier et al., "Pt-scotch: A tool for efficient parallel graph ordering," *Parallel Comput.*, vol. 34, pp. 318–331, 2008.
- [Pregel]G. Malewicz et al., "Pregel: a system for large-scale graph processing," in *SIGMOD '10*
- [PBGL] D. Gregor et al., "The parallel bgl: A generic library for distributed graph computations," in *POOSC '05*.
- [Neo4j] http://neo4j.org/.
- [Infgraph] http://www.infinitegraph.com/.
- [Hypgraph] B. Iordanov, "Hypergraphdb: a generalized graph database," in *WAIM'10*.

References-3

- [Brandes01] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, 2001.
- [Ercsey10] M. Ercsey-Ravasz and Z. Toroczkai, "Centrality scaling in large networks," *Phys. Rev. Lett.*, 2010.
- [Bader06] D. A. Bader and K. Madduri, "Parallel algorithms for evaluating centrality indices in real-world networks," in *ICPP '06*
- [Brandes07] U. Brandes and C. Pich, "Centrality estimation in large networks," in *Special issue "Complex Networks' Structure and Dynamics" of the International Journal of Bifurcation and Chaos*, 2007
- [Madduri09] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarria-Miranda, "A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality
- on massive datasets," in *IPDPS '09*.
- [Edmonds10] N. Edmonds, T. Hoefler, and A. Lumsdaine, "A space-efficient parallel algorithm for computing betweenness centrality in distributed memory," in *ICPP'10*.
- [Tan09] G. Tan, D. Tu, and N. Sun, "A parallel algorithm for computing betweenness centrality," in *ICPP '09*.
- [tu09] D. Tu and G. Tan, "Characterizing betweenness centrality algorithm on multi-core architectures," in *ISPA'09*.
[Pegasus] U. Kang, C. E. Tsourakakis, C. Faloutsos. "PEGASUS: A Peta-Scale Graph Mining System – Implementation and Observations." IEEE 9th International Conference on Data Mining, Dec. 2009.
- [Brin1998] S. Brin and L. Page, "The anatomy of a large-scalehypertextual (web) search engine." in. WWW, 1998
- [Pan2004] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu,"Automatic multimedia AC
- cross-modal correlation discovery,"SIGKDD, Aug. 2004.
- [Kang2008] U. Kang, C. Tsourakakis, A. Appel, C. Faloutsos, and J. Leskovec, "Hadi: Fast diameter
- estimation and mining in massive graphs with hadoop," CMU-ML-08-117, 2008.
- [MapReduce] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters,"
- OSDI, 2004.

Thanks!