Managing and Mining Billion Node Graphs

Haixun Wang
Microsoft Research Asia

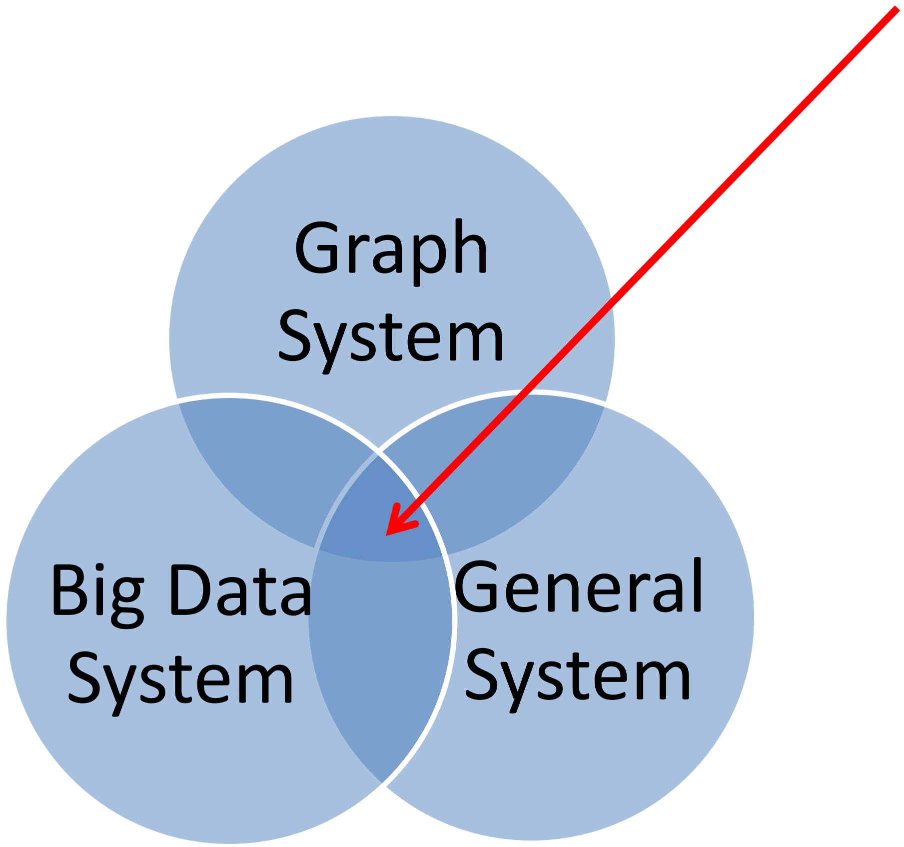
Outline

- Overview
- Storage
- Online query processing
- Offline graph analytics
- Advanced applications

Is it hard to manage graphs?

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

This is hard.



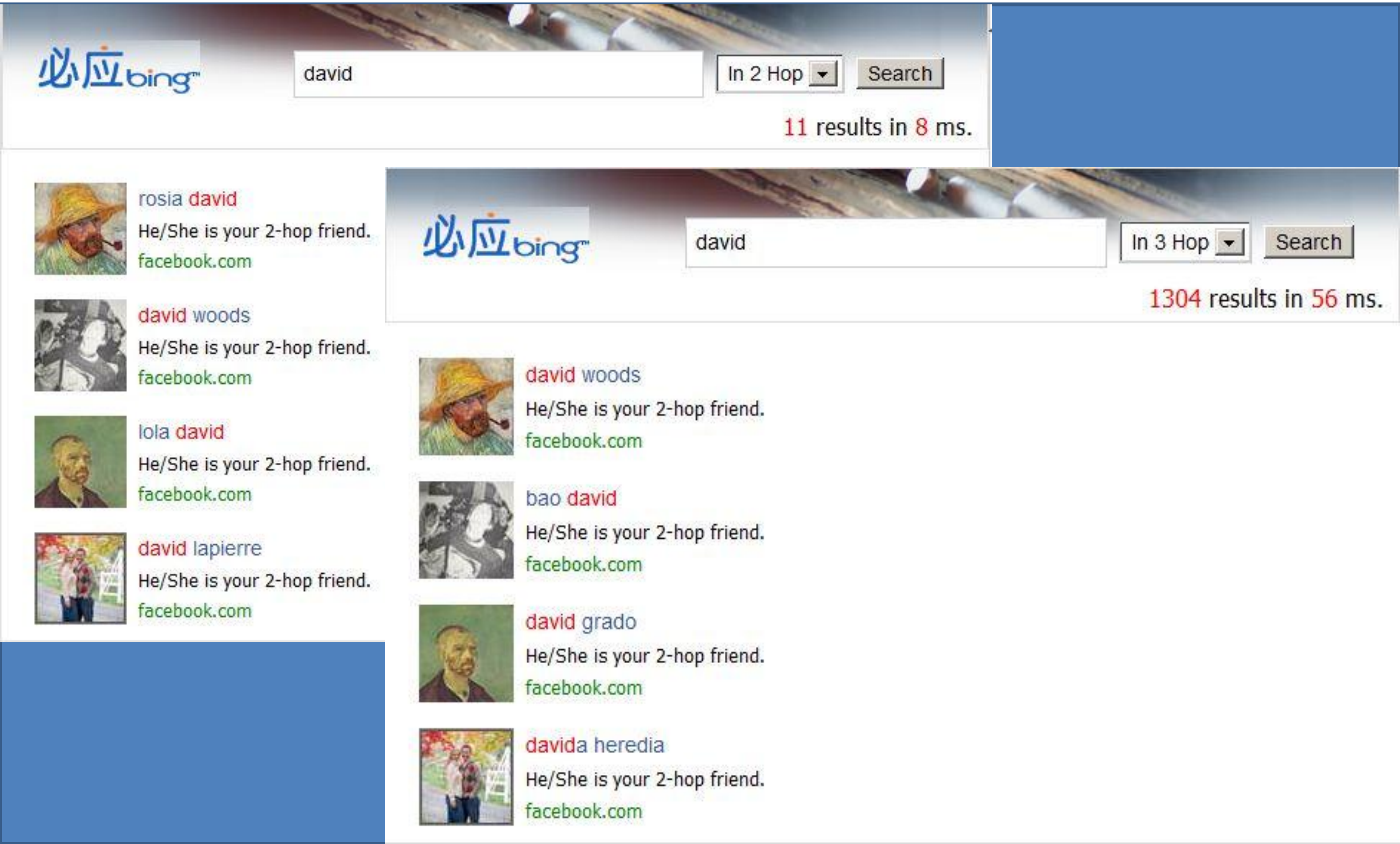
Existing Systems

	Native Graph*	Online Query	Index	Transaction Support	Parallel Graph Processing	Topology in memory	Distributed
Neo4j	YES	YES	Lucene	YES	NO	NO	NO
HyperGraphDB	NO	YES	Built-in	YES	NO	NO	NO
InfiniteGraph	NO	YES	Built-in + Lucene	NO	NO	NO	YES
Pregel	NO	NO	NO	NO	YES	NO	YES

....

An Example:
Online Query Processing in Graphs

People Search Demo



Very efficient graph exploration

- Visiting 2.2 million nodes on 8 machines in 100ms

- Graph models
  - Relational
  - Matrix
  - Native (Trinity)

- Relation/Matrix requires data join operations for graph exploration
  - Costly and producing large intermediary results

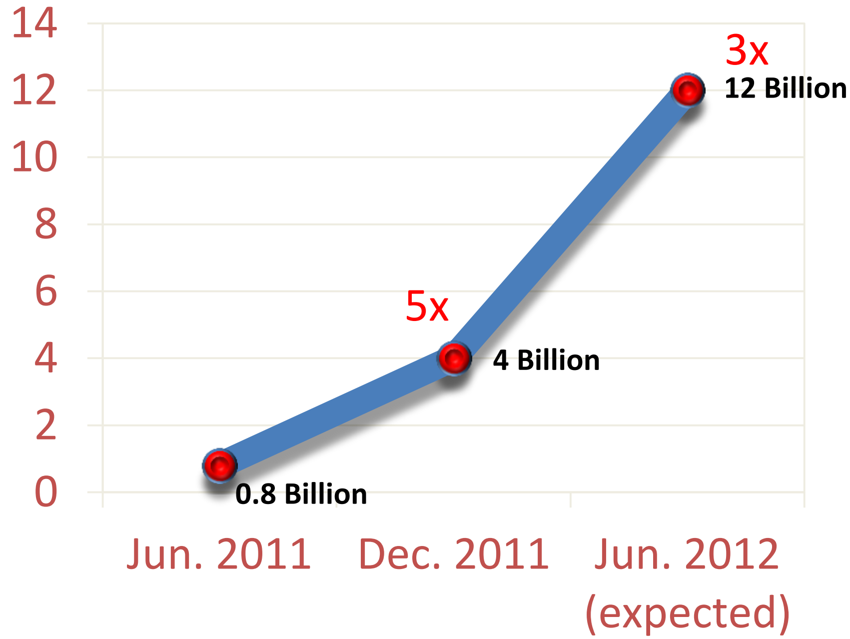
Joins vs. Graph Exploration for subgraph matching

- Subgraph matching
  - Index sub-structures
  - Decompose a query into a set of sub-structures
  - Query each sub-structure
  - Join intermediary results
- Index size and construction time
  - Super-linear for any non-trivial sub-structure
  - Infeasible for billion node graphs
- Graph exploration
  - No structural index is needed
  - Very few join operations
  - Small intermediary results

SPARQL Query On Satori

Microsoft's Knowledge System

One of the world's largest repository of knowledge.



Compare with RDF-3x

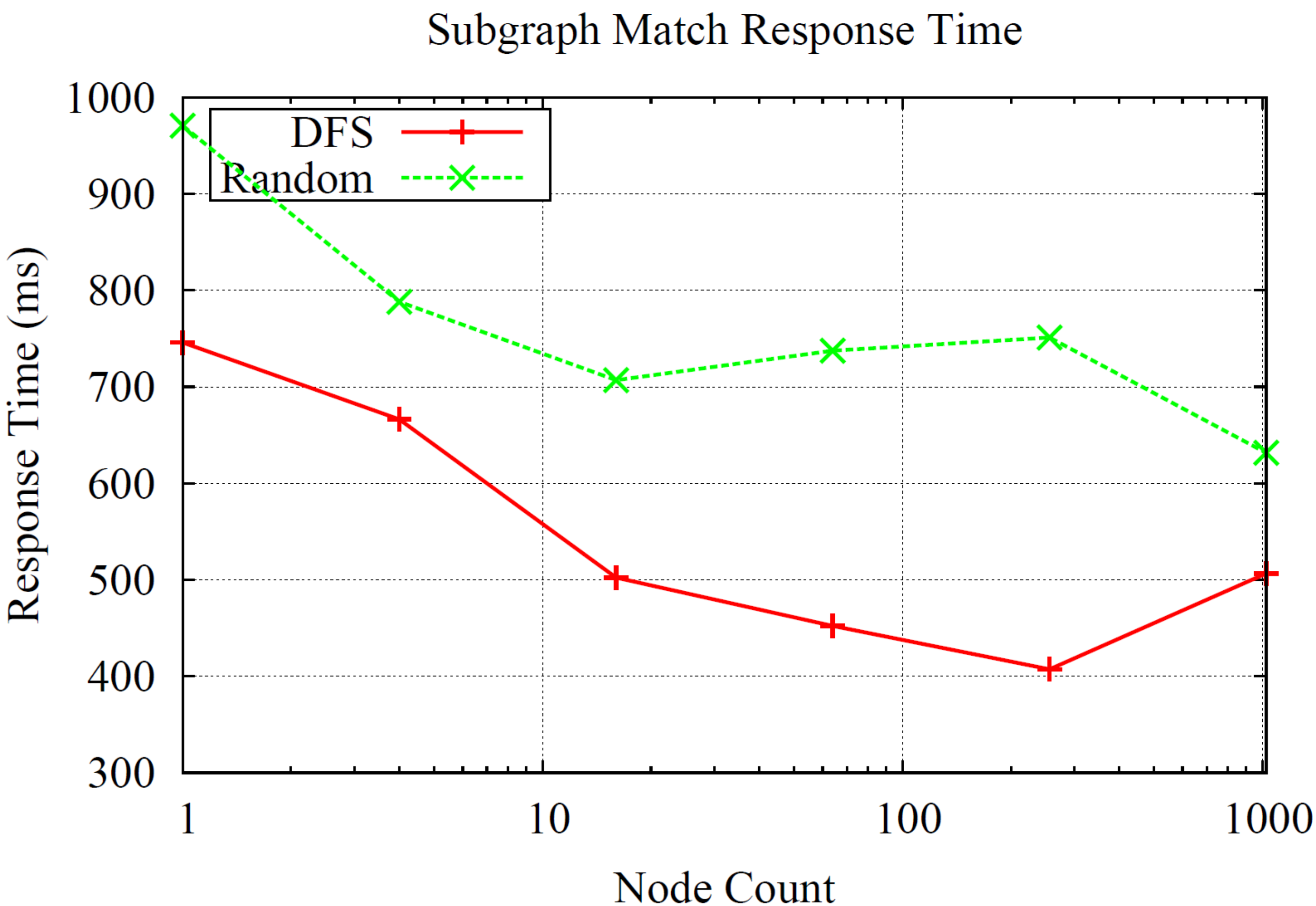
- RDF3x is a state-of-the-art single node RDF engine
- Dataset Statistics
  - LUBM-10240, a public benchmark for RDF
  - Triple number: 1.36 billion
  - Subject/Object number: 0.3 billion
- Queries are chosen from the benchmark queries published with LUBM

Query ID	1	2	3	4	5	6	7
Our System	**15640**	**9849**	**11184**	**5**	**4**	**9**	**37666**
RDF-3x	39m2s	14194	30585	14	11	65	69560

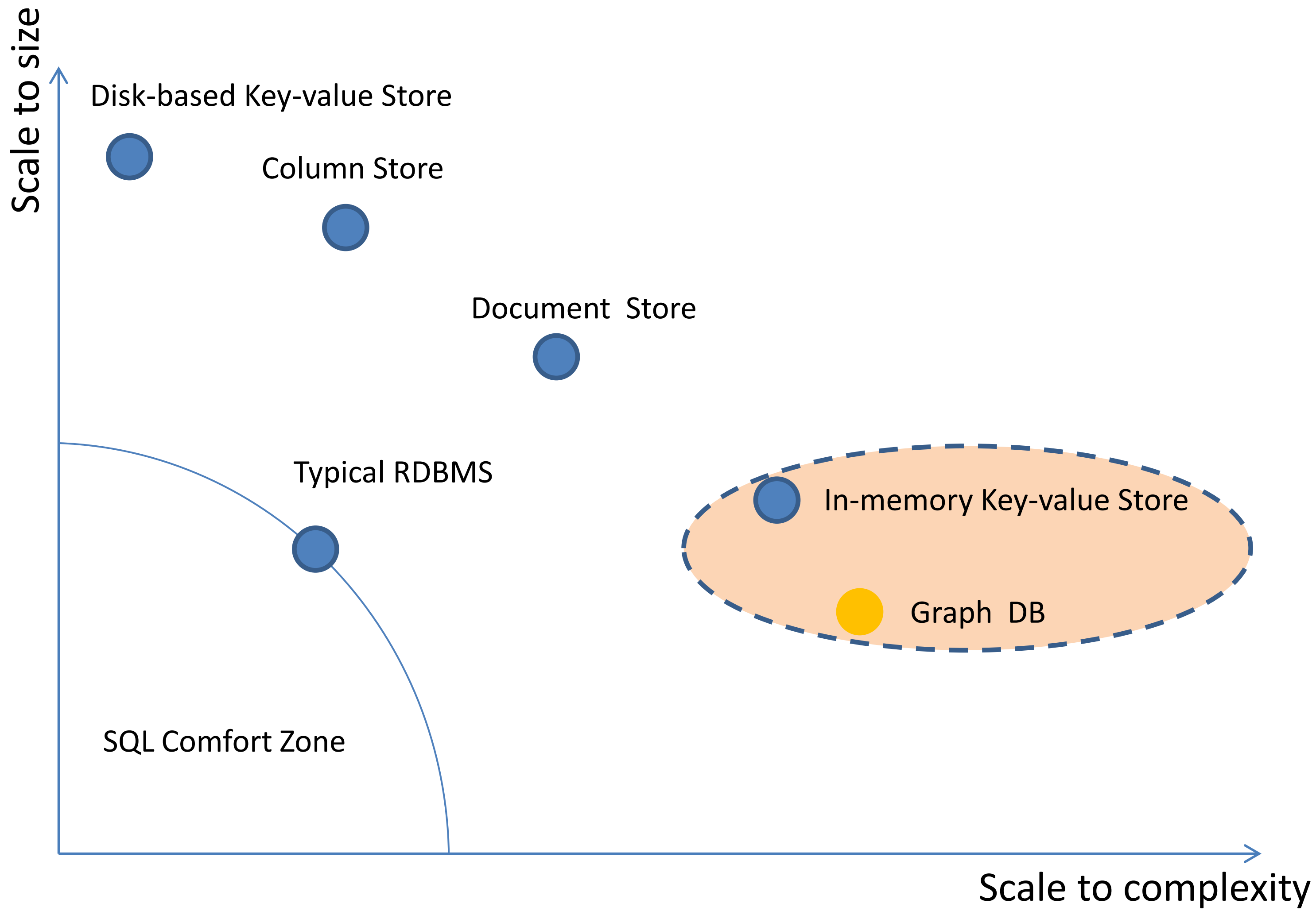
Subgraph matching on a billion node graph

- No feasible solution for billion node graphs
- Super-linear index is not feasible
- Desiderata: requiring no index

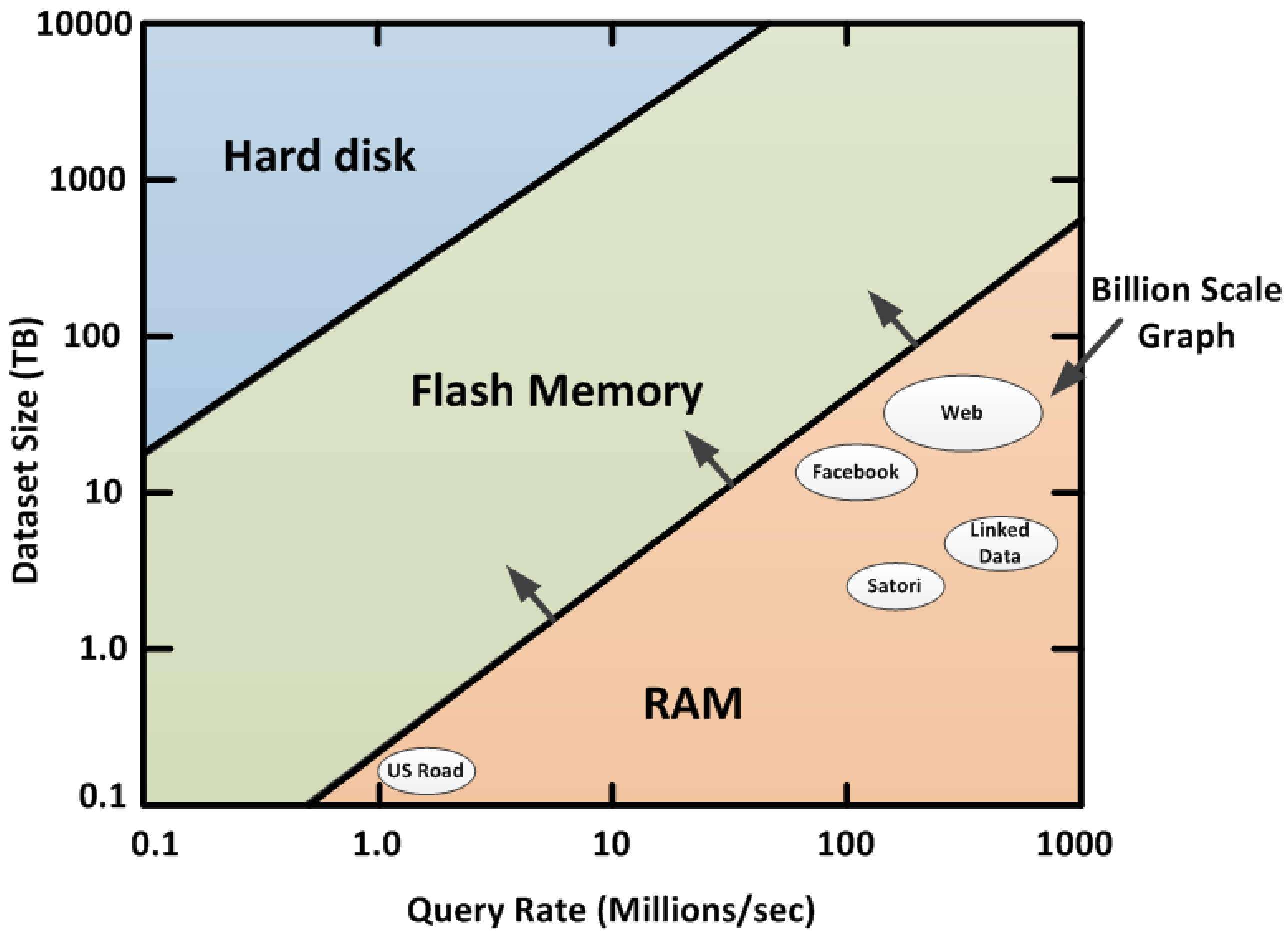
Subgraph Match Query



One size fits all?



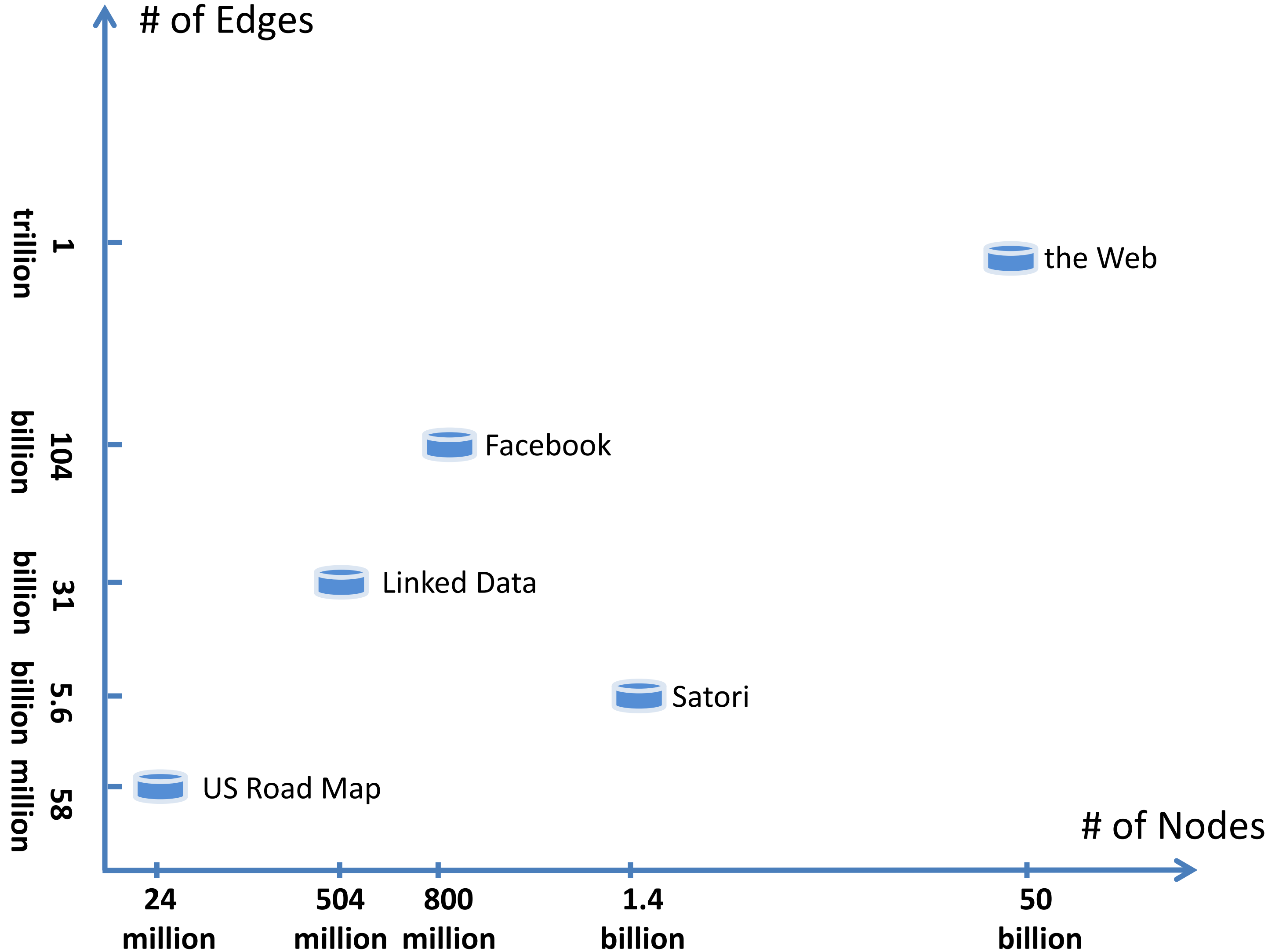
Memory-based Systems



Trinity

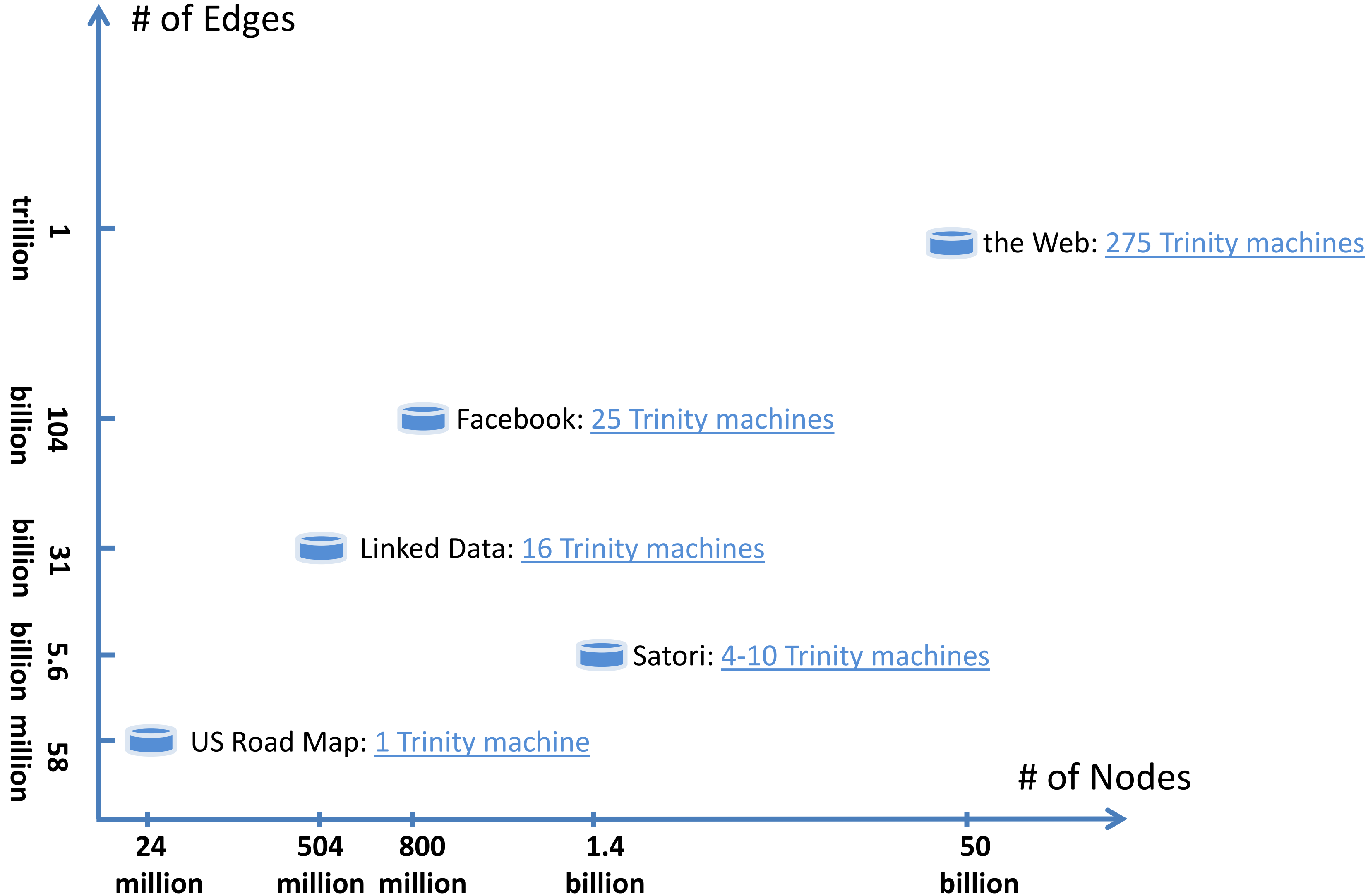
- A distributed, in-memory key/value store
- A graph database for online query processing
- A parallel platform for offline graph analytics

Graphs



# of Edges
1 trillion
104 billion
31 billion
5.6 billion
58 million
the Web
Facebook
Linked Data
Satori
US Road Map
# of Nodes
24 million
504 million
800 million
1.4 billion
50 billion

How many machines?

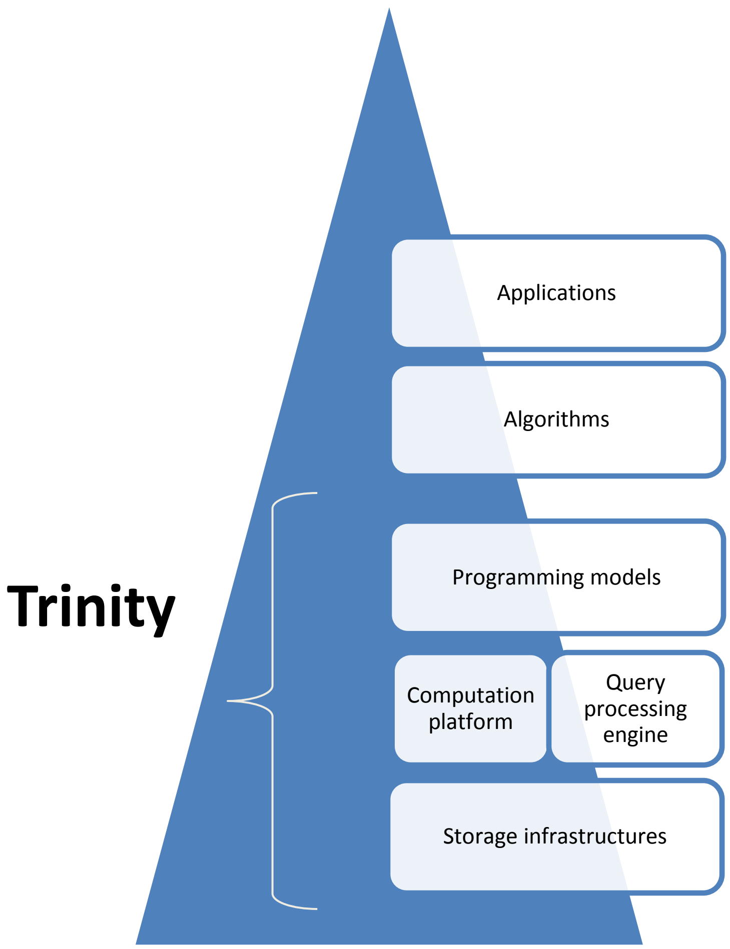


Performance Highlight

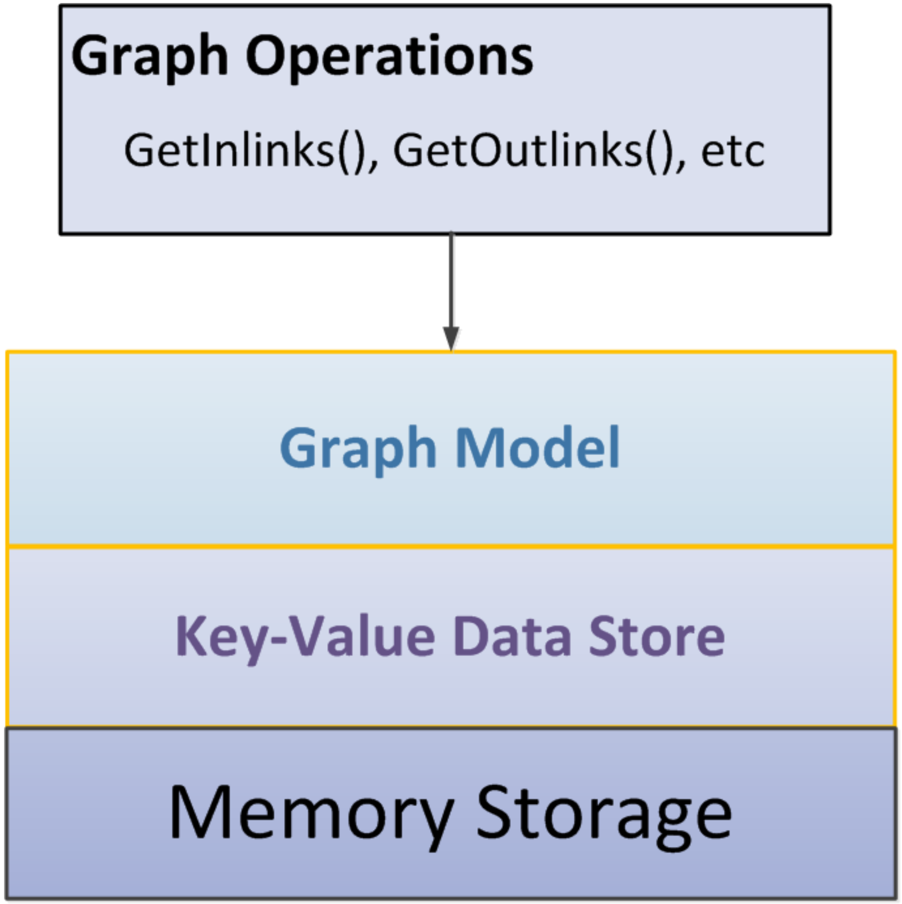
- Graph DB (online query processing):
  - visiting 2.2 million users (3 hop neighborhood) on Facebook: <= 100ms
  - foundation for graph-based service, e.g., entity search
- Computing platform (offline graph analytics):
  - one iteration on a 1 billion node graph: <= 60sec
  - foundation for analytics, e.g., social analytics

Performance Highlight

Typical Job	State-of-the-Art	Trinity
N hop People Search	2 hop search 100 ms; 3 hop is not doable online	2 hop search in 10ms 3 hop search in 100ms
Subgraph Matching	For billion node graphs, indexing takes months to build	No structural index required; response time $\leq$ 500 ms
(Approximate) Shortest Distances/Paths	No efficient solution for large graphs	10 ms for discovering shortest paths within 6-hops
PageRank of Web Graph	Cosmos/MapReduce: 2 days using thousands of machines	1 iteration on 1B graph $\leq$ 60sec using 10 machines
Billion Node Graph Partitioning	<22M nodes	Billions of nodes
Graph Generation	For small graph only	Generates 1 billion node graph of power law distribution in 15 minutes

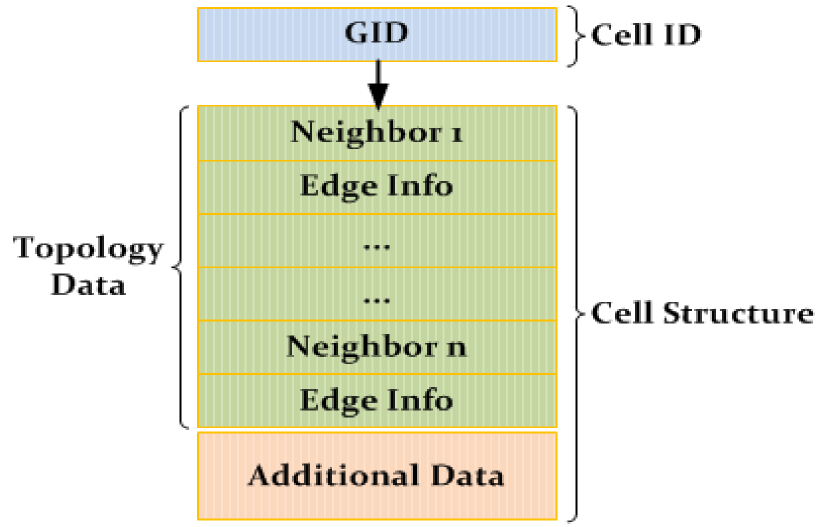

Trinity
Applications
Algorithms
Programming models
Computation platform
Query processing engine
Storage infrastructures

Storage and Communication


Graph Operations
GetInlinks(), GetOutlinks(), etc
Graph Model
Key-Value Data Store
Memory Storage

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

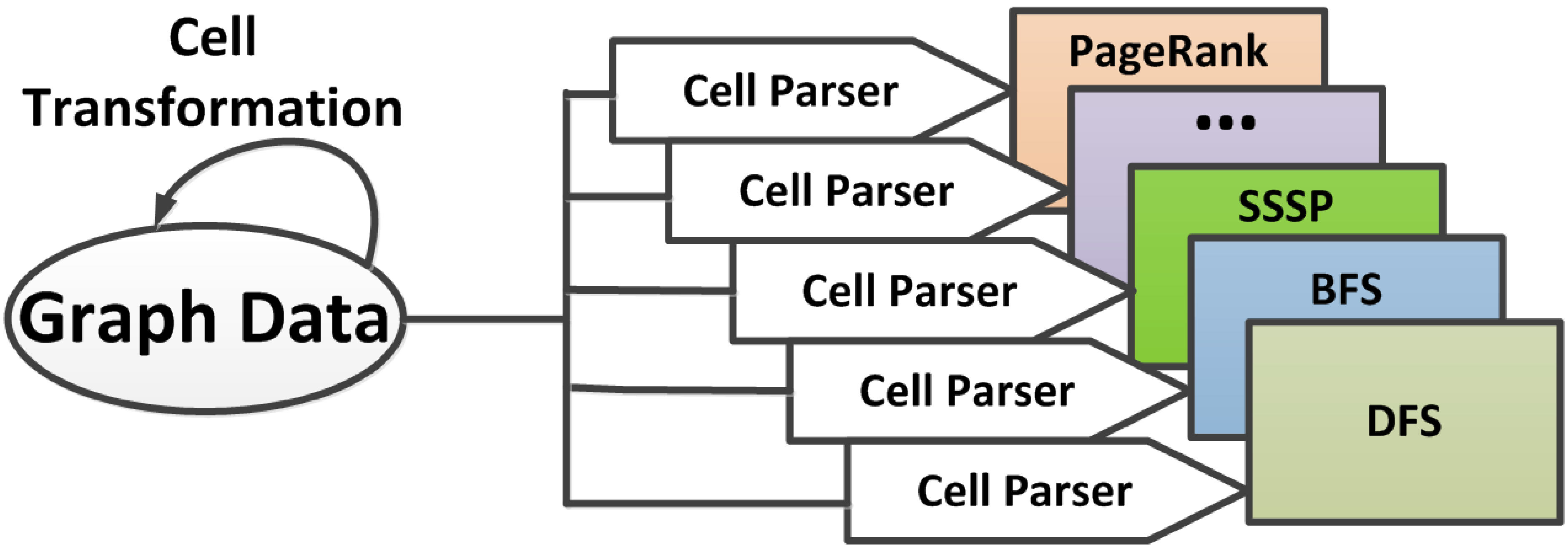


Cell Schema Example

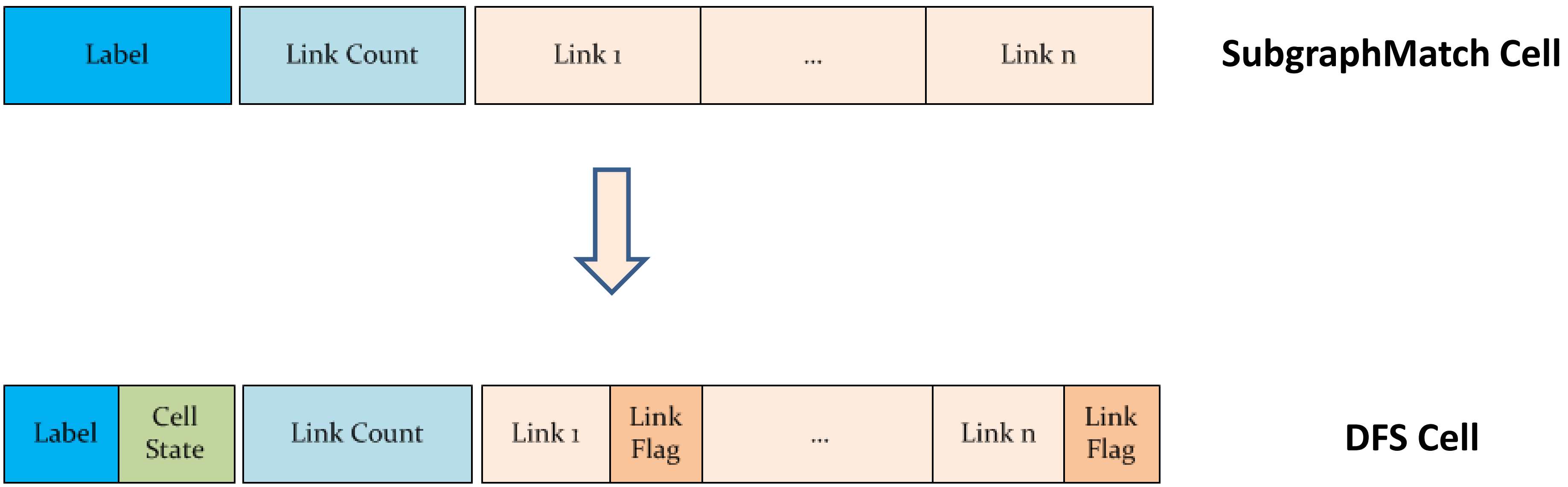
```
public class myCell: ISubgraphMatchCell, IVisualCellParser
{
    public byte[] label;
    public List<long> inlinks;
    public List<long> outlinks;
    internal long cell_id;

    public unsafe byte[] ToBinary();
    public unsafe ISubgraphMatchCell FromBinary(byte[] bytes);

    public void VisualParseCell(long nodeID,
                                byte[] cell_bytes,
                                VisualizerCallbackToolkit toolkit);
}
```

Subgraph Matching Cell

Cell Transformation and Cell Parsing



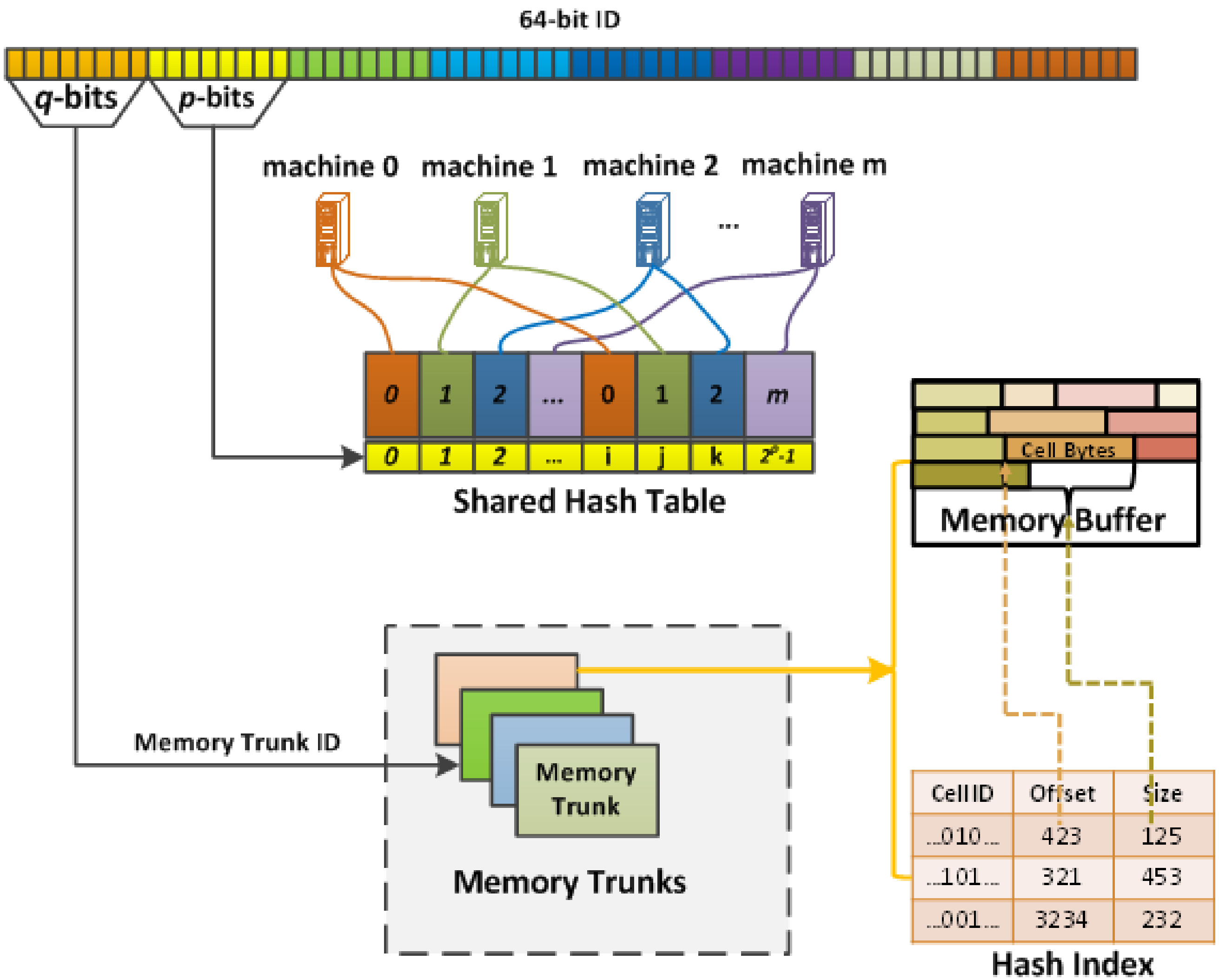
Cell Transformation



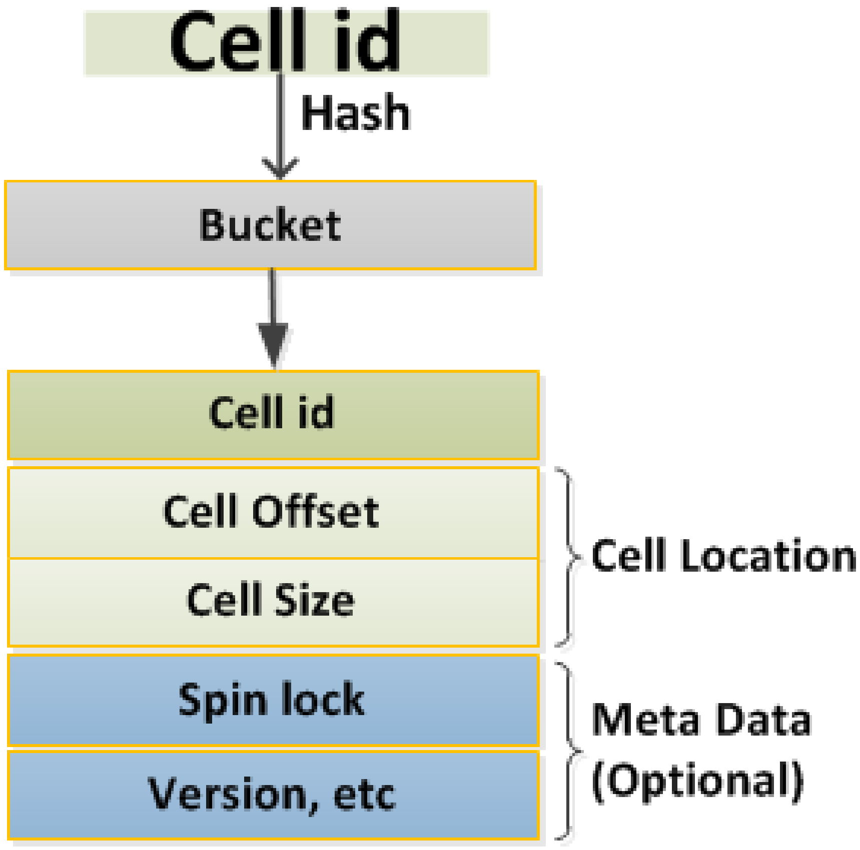
Partitioning

- Default: random partitioning by hashing
- Customizing hashing function
- Re-partitioning a graph (as an analytical job)

Cell Addressing



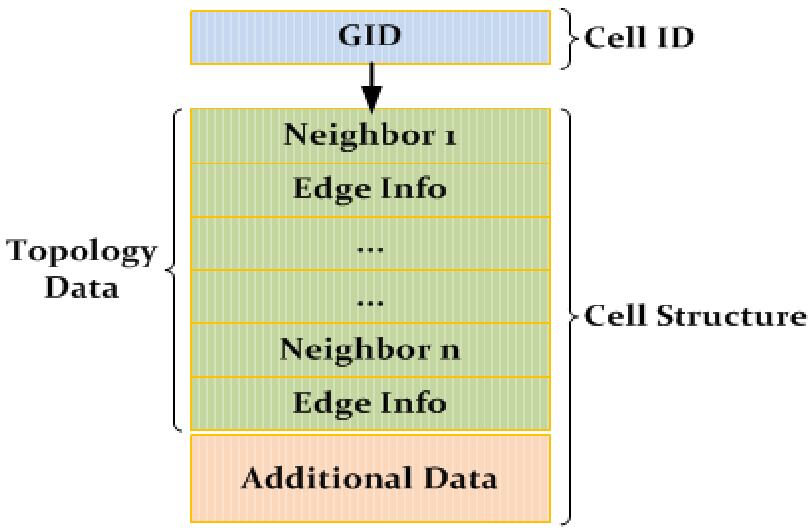
Cell Addressing

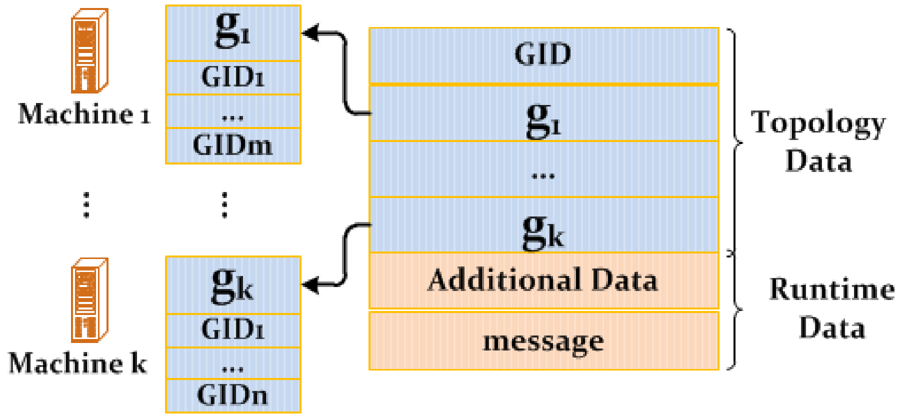


Big cell

A warning to graph system builders: Lady Gaga has 40,000,000 fans.

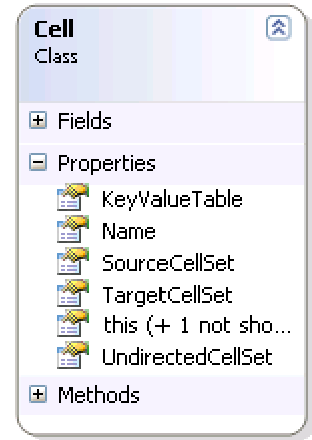
Her cell takes 320 Mb.





Cell as memory objects vs. blobs

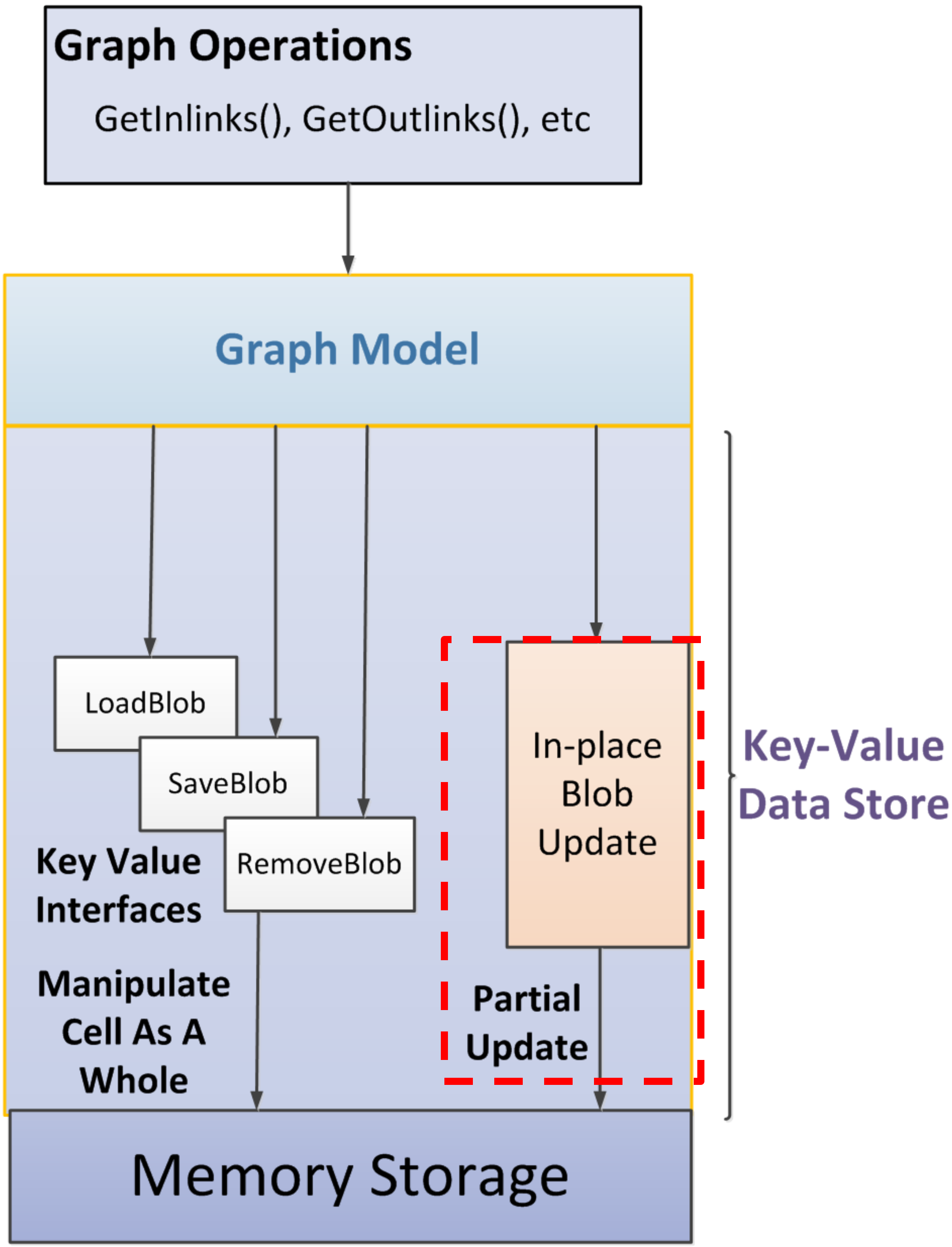
- A memory object
  - Runtime object on heap



- A Blob
  - A binary stream
  - Benefit:
    - 50,000,000 cells, each of size 35 bytes
    - CLR heap objects: 3,934 MB
    - Flat memory streams: 1,668 MB
  - Challenge:
    - Parsing

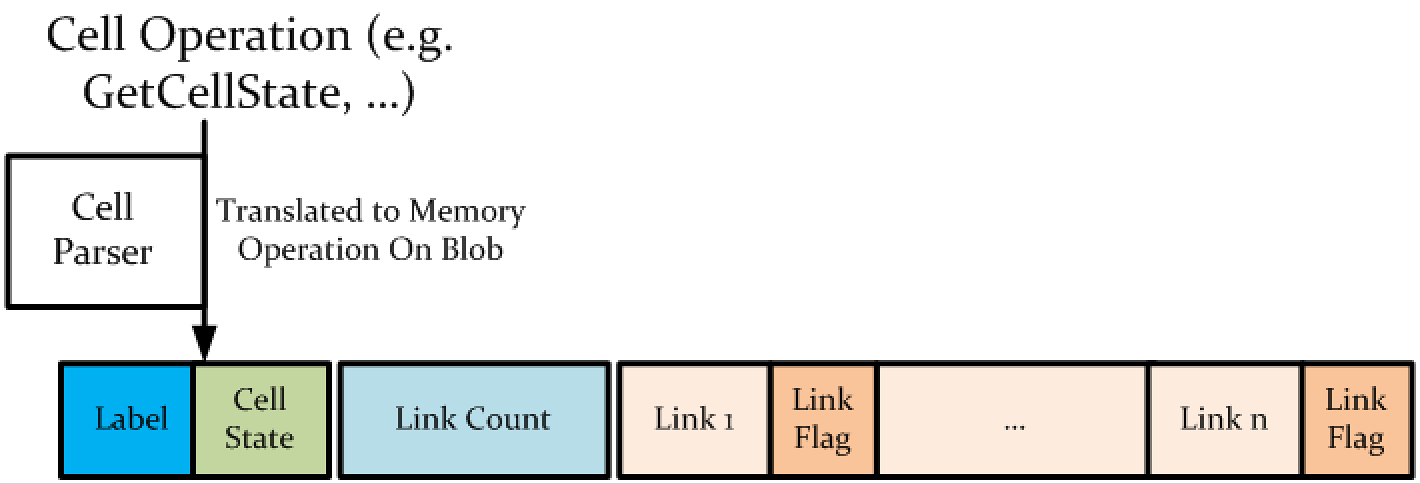
vs.

Data Model


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

‘In-place’ Blob Update

- The operations to a cell can be translated to memory manipulations to blob using a cell parser, e.g.



Cell Placements in Memory

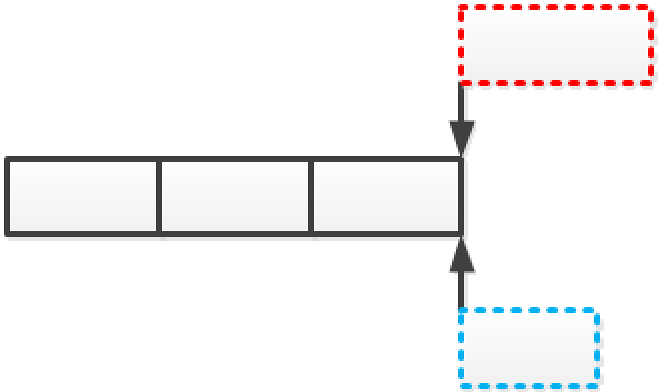
- Cells are of different size
- Two types of applications:
  - Mostly read-only, few cell-expansion operations (e.g., a knowledge taxonomy, a semantic network)
  - Many cell-expansion operations (e.g., synthetic graph generation)

Cell Placements in Memory

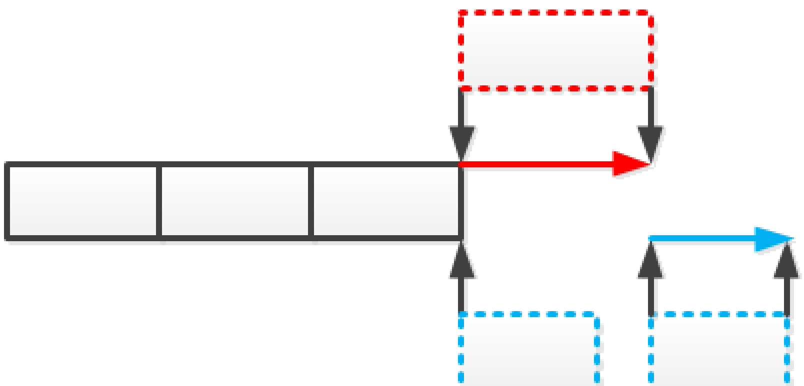
- Sequential memory storage
  - Cells are continuously stored in memory
  - Compact.

- Sparse hash storage
  - A cell may be stored as non-continuous blocks
  - Dynamic. (streaming updates)

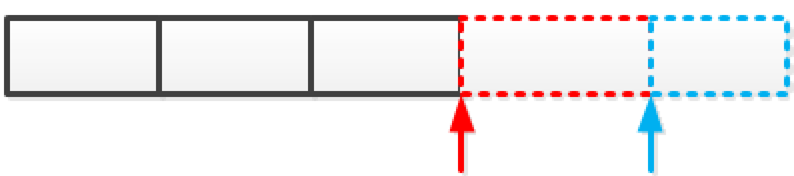
Concurrent Sequential Memory Allocation



Concurrent Memory
Allocation Request



Atomically Incrementing
Memory Pointer
(A CAS-compare and swap-instruction)



Concurrent write from
allocated buffer header

Concurrent operations on a cell (using a Spin Cell Lock)

```
while(CAS(ref lockArray[index], 1, 0) != 0);
```

```
if(CAS(ref spin_lock, 1, 0) != 0)
{
    While(true)
    {
        if(volatile_read(ref spin_lock) == 0 && CAS(ref lockArray[index], 1, 0) == 0)
            return;
    }
}
```

Optimized Version

Hash Storage

- The memory address space is a hash space.
- Use a hash function to allocate memory.
- Support lock-free streaming cell updates.

Hash Storage

		head	id	links	-1				
	head	id	links	…	-1				
	head	id1	links	…	jmp	head	id	links	-1
	links	…	…	…	jmp	head	id	links	…
…	-1					Chaining			
						Links…	…	-1	

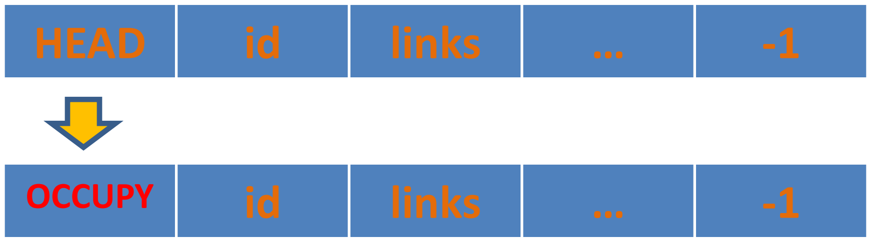
Conflict Resolution in Hash Storage

- Conflicts
  - Two node IDs are hashed to the same slot
  - One node ID is hashed to the middle of the other
- Rehashing

```
long Hash( long key, int iteration )
{
   long hashCore = GetHashCore( key );
   return ( hashCore + iteration * ( 1 + ( ( ( hashCore ) + 1 ) % ( Capacity - 1 ) ) ) ) % Capacity;
}
```

Concurrency Control

- To lock: change HEAD flag to OCCUPY flag
- To unlock: change OCCUPY back to HEAD
- The Flag changing must be atomic (e.g. Interlocked CAS)

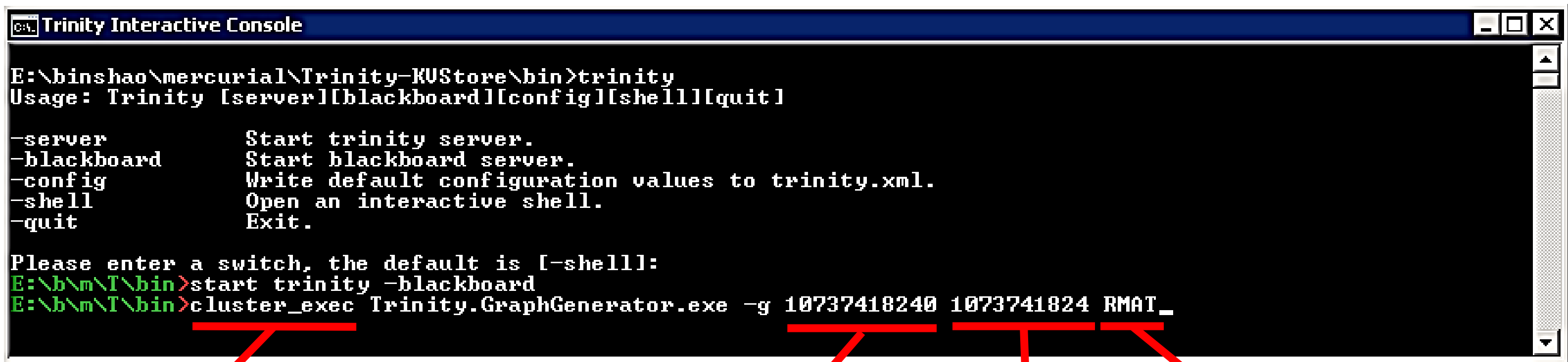


Fast Billion Node Graph Generator

Challenge: i) massive random access; ii) cell size change

Existing graph generators	Trinity graph generator
• Slow • For small graphs only	• generate a 1 billion node graph of power law distribution in 15 minutes

Trinity Graph Generator



parallel execution
in the entire cluster

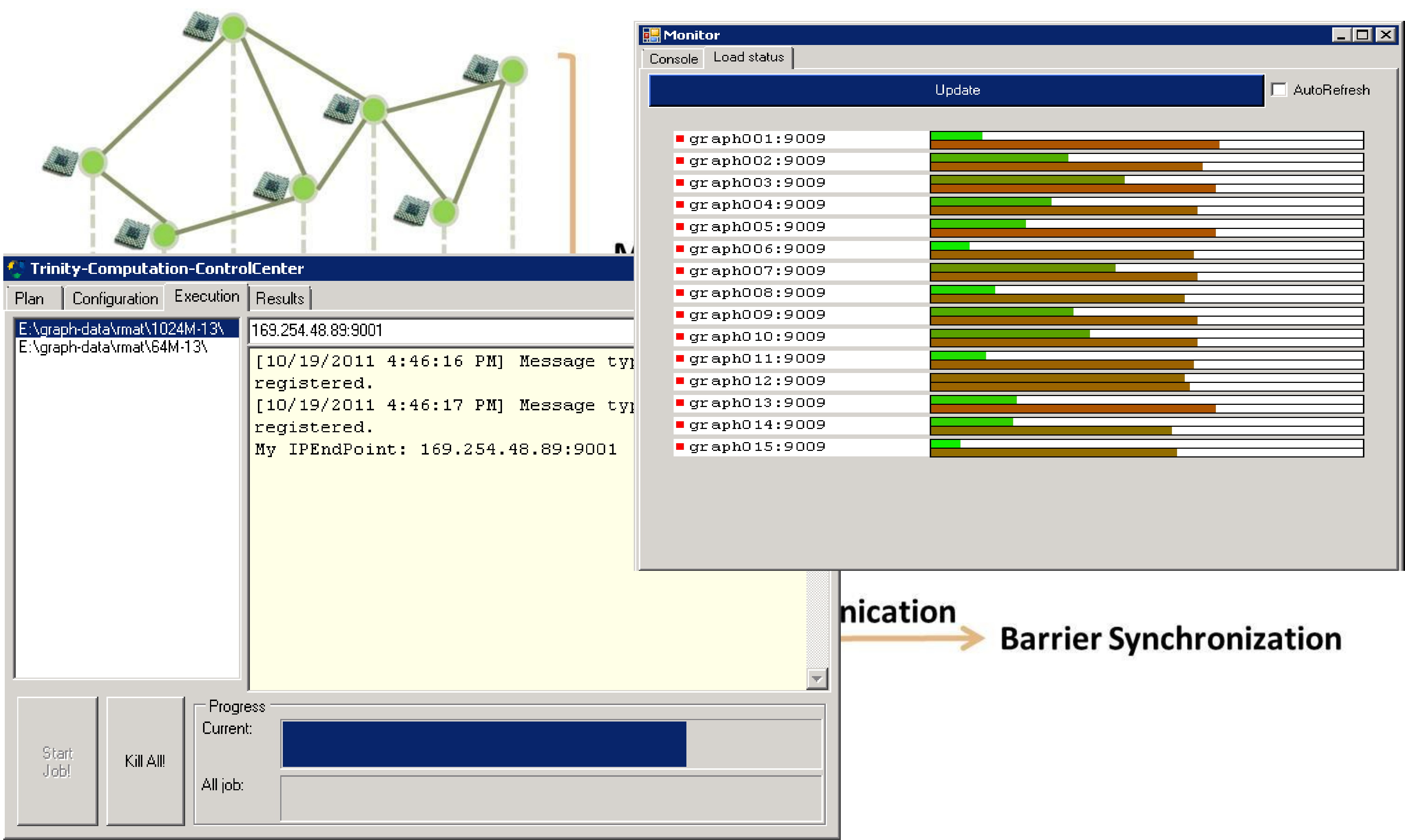
edge #:
10 B

node #:
1 B

graph type

Offline Graph Analytics

Vertex-Based Graph Computation



Page Rank Script

```
public class Cell : BaseCell
{
   public int OutDegree;
   public Message message;

   public override void Initialize()
   {
      OutDegree = outlinks.Length;
      message.PageRankScore = 1.0f / GlobalNodeNumber;
   }
   public override void Computation()
   {
      float PageRankScore = 0;
      if (CurrentIterationRoundNumber > 100)
         VoteToStop();
      foreach (long msg_id in inlinks)
      {
         Message msg =
(Message)CacheManager.GetMessage(msg_id);
         PageRankScore += msg.PageRankScore;
      }
      message.PageRankScore = PageRankScore / OutDegree;
   }
}

public struct Message
{
   public float PageRankScore;
}
```

Local Variables

Vertex-based Computation

Fetch Message

Generate New Message

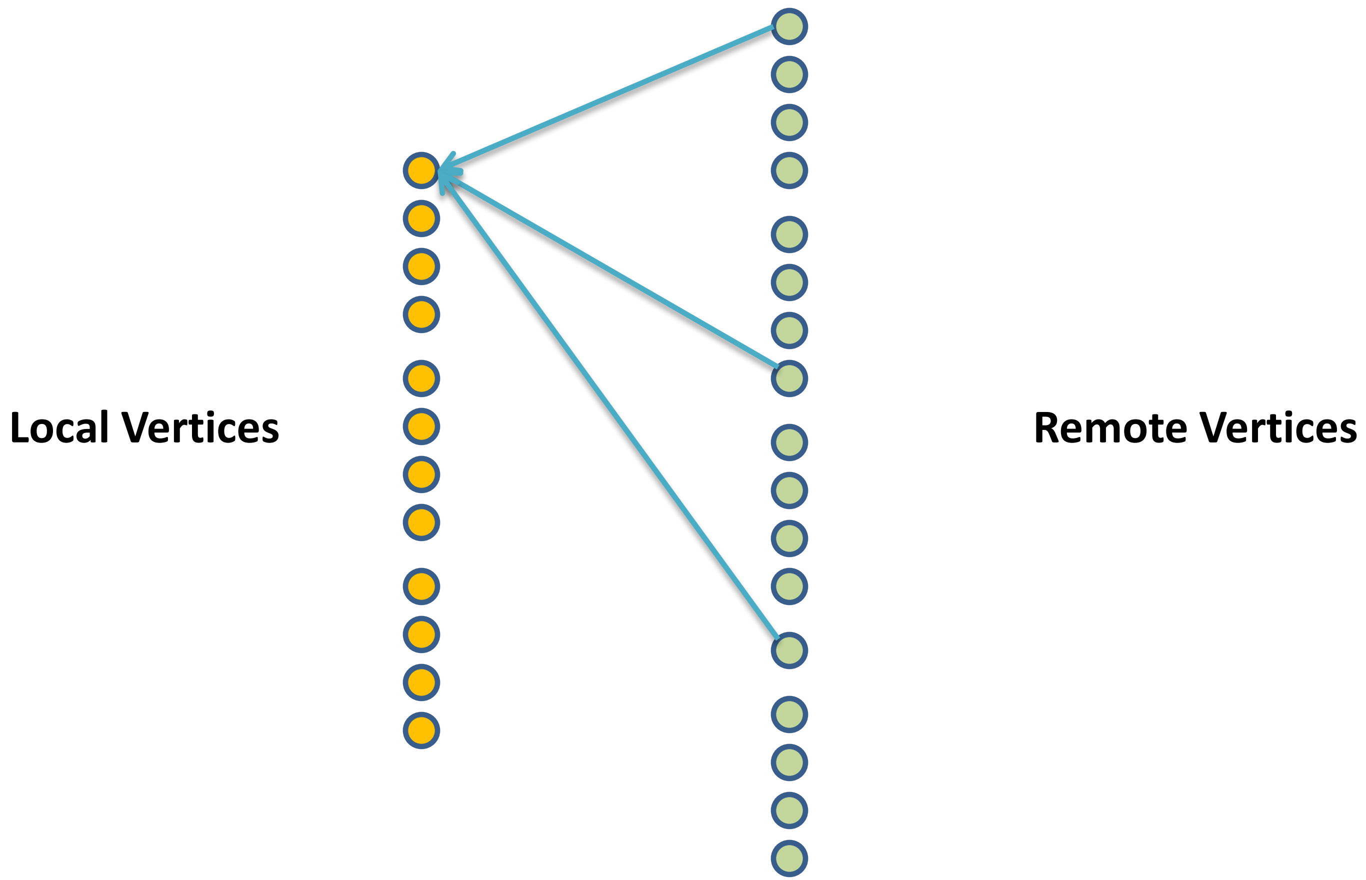
Vertex-based Computation

- Take each graph vertex as a processor
- Synchronous communication model (BSP)
- Used in Pregel and its open source variants, such as Giraph and GoldenOrb

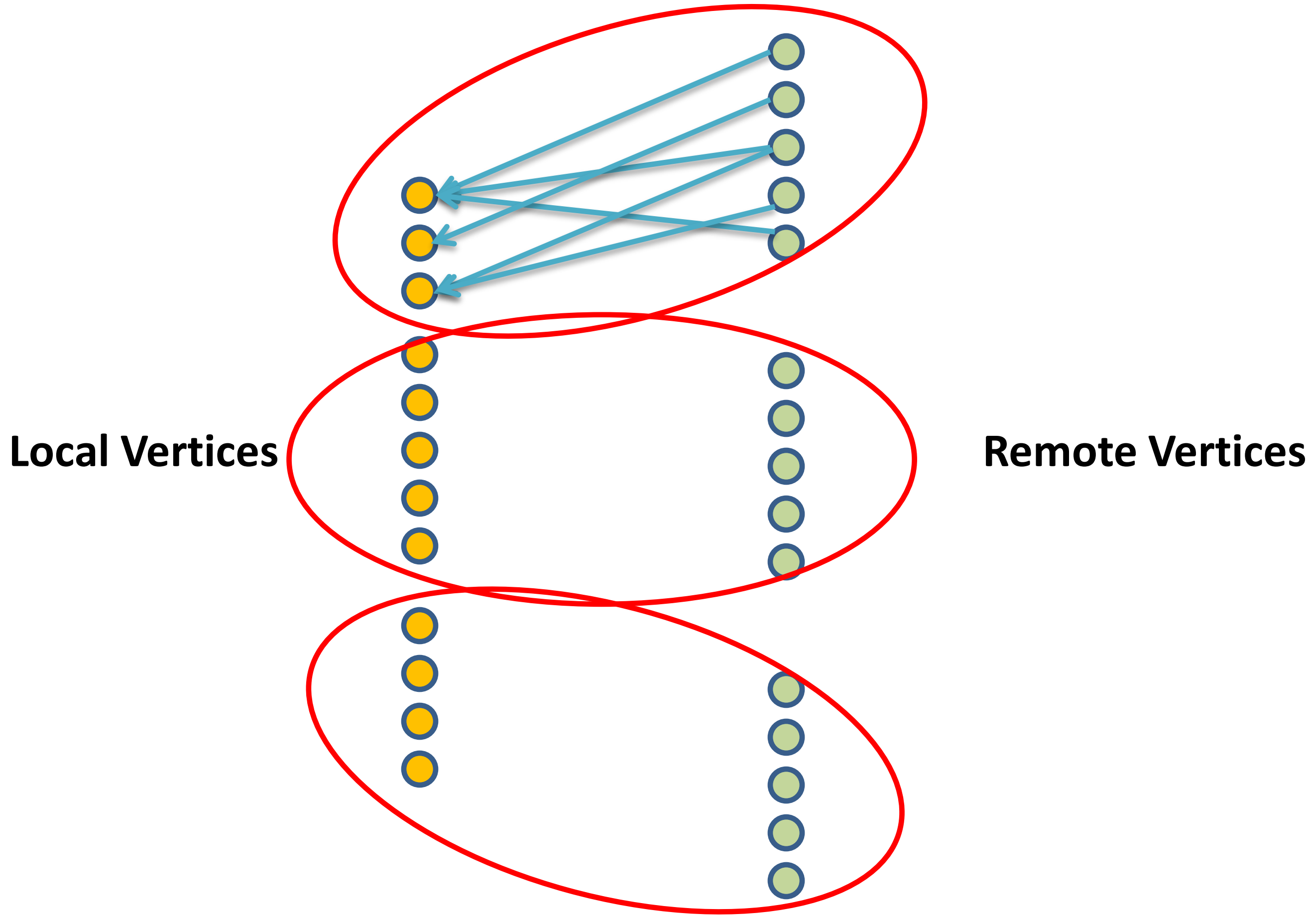
Restricted vertex-based computation

- In each super step, the sender and receiver set are known beforehand
- Messages can be categorized by their hotness
- Messages can be pre-fetched and cached

A bi-partite view



A bi-partite view



How many machines do we need?

- Facebook
  - 800 million users, 130 friends per user
  - 30 Trinity machines

- Web
  - 25 billion pages, 40 links per page
  - 1~~5~~0 Trinity machines → 30 Trinity machines

Distribution-Aware Computing

- Support message sending by two APIs:

- Local message sending
  – sending messages to neighboring nodes on the same machine

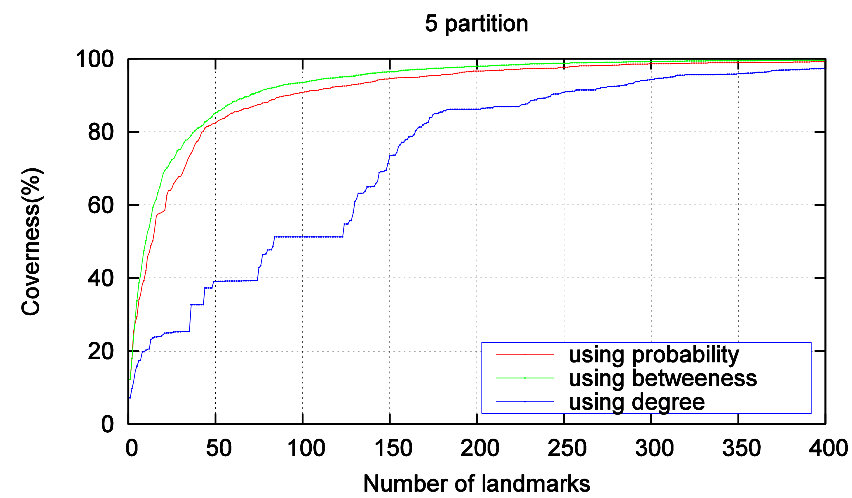
- Message sending
  – sending messages to neighboring nodes

Distribution-Aware Computing

- Nodes are distributed on N machines (randomly)
  - Each machine has 1/N nodes, and d/N edges
- Can we perform computation on 1 machine and estimate the results for the entire graph?
  - Graph density estimation
- Can we perform computation on each machine locally, and aggregate their results as the final step?
  - Finding connected components

Distributed betweenness approximation

- Betweeness is the most effective guideline to select landmarks
- Exact betweenness costs O(nm) time, unacceptable on large graphs
- Approximate fast betweenness is expected
    - Count shortest paths rooted at sampled vertices
    - Count shortest path with limited depth
- On distributed platform
    – We count the shortest paths within each machines



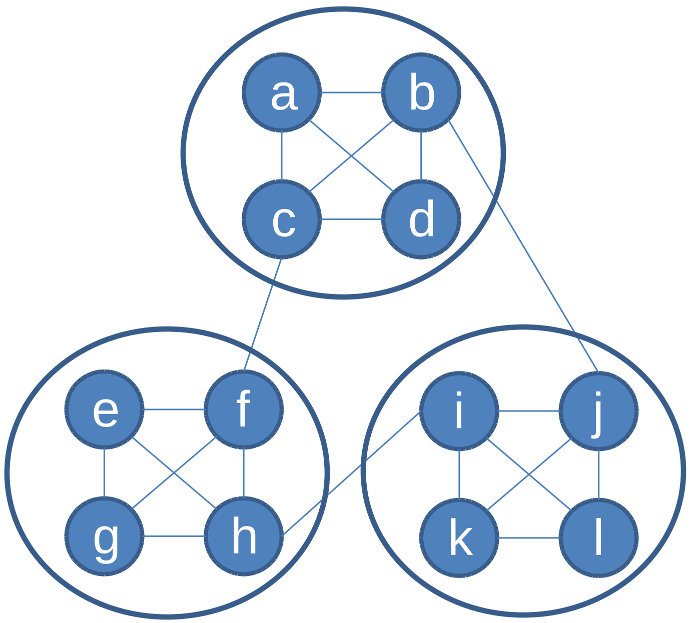
How many machines do we need?

- Facebook
  - 800 million users, 130 friends per user
  - ~~3~~0 Trinity machines → 1 Trinity machine
- Web
  - 25 billion pages, 40 links per page
  - 1~~5~~0 Trinity machines → 1 Trinity machine

Billion-node graph partitioning

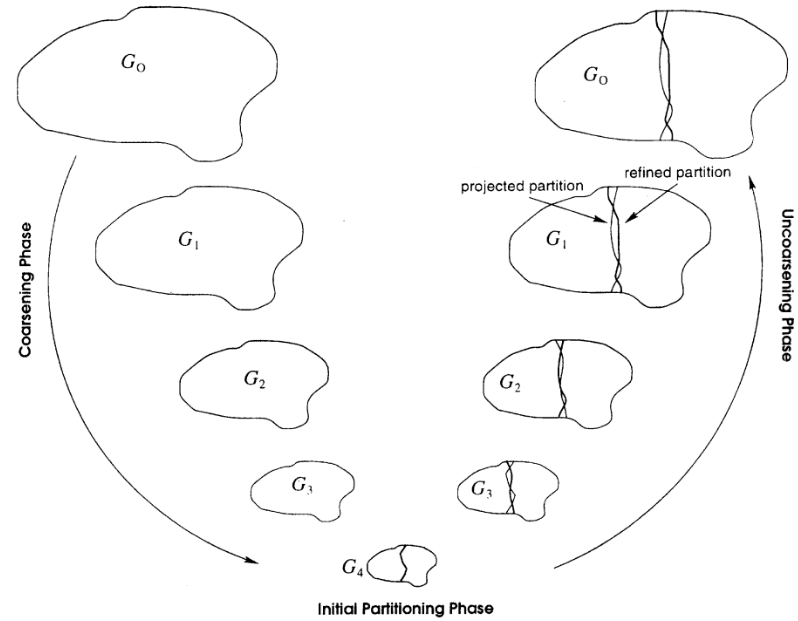
Why partition?

- Divide a graph into k (almost) equal-size partitions, such that the number of edges between partitions is minimized.
- A better partition helps
  – Load balance
  – Reduce communication
- Example: BFS on the graph
  – Best partitioning needs 3 communications
  – Worst partitioning needs 17



State-of-art method: Metis

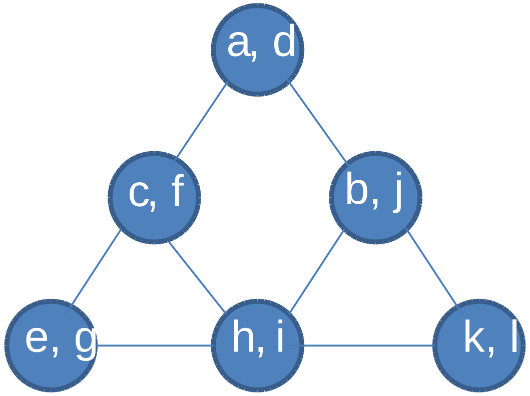
- A multi-level framework
- Three phrases
  - Coarsening by maximal match until the graph is small enough
  - Partition the coarsest graph by KL algorithm [kl1972]
  - Uncoarsening



Ref: [Metis 1995]

Weakness of metis

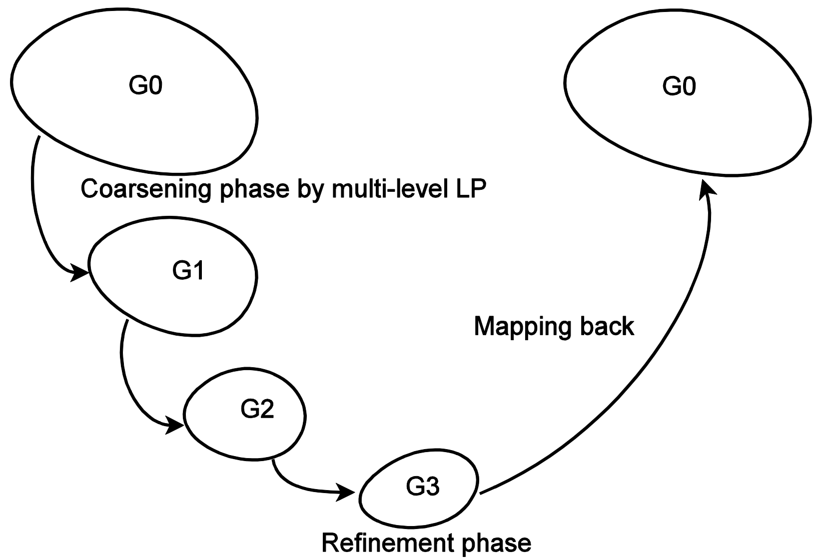
- Not semantic aware: Coarsening is ineffective on real graphs
  - Principle of coarsening
    - An optimal partitioning on a coarser graph implies a good partitioning in the finer graph
  - Coarsening by maximal match can guarantee this *only when node degree is bounded*, for example 2D, 3D meshes
  - Real networks have *skewed degree distribution*

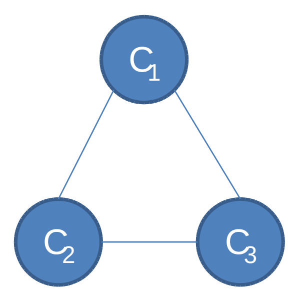


- Inefficiency
  - To support uncoarsening, many mappings and intermediate graphs need to be stored, leading to bad performance
  - For example, a 4M graphs consumes 10G memory
- Not general enough
  - Exclusive for partitioning

Suggested solution : Multi-level Label Propagation

- Coarsening by label propagation
  - Lightweight
    - Can be easily implemented by message passing
  - Semantic aware
    - Can discover the inherent community structure of real graph
- Label propagation
  - In each iteration, a vertex takes ***the label that is prevalent in its neighborhood*** as its own label.





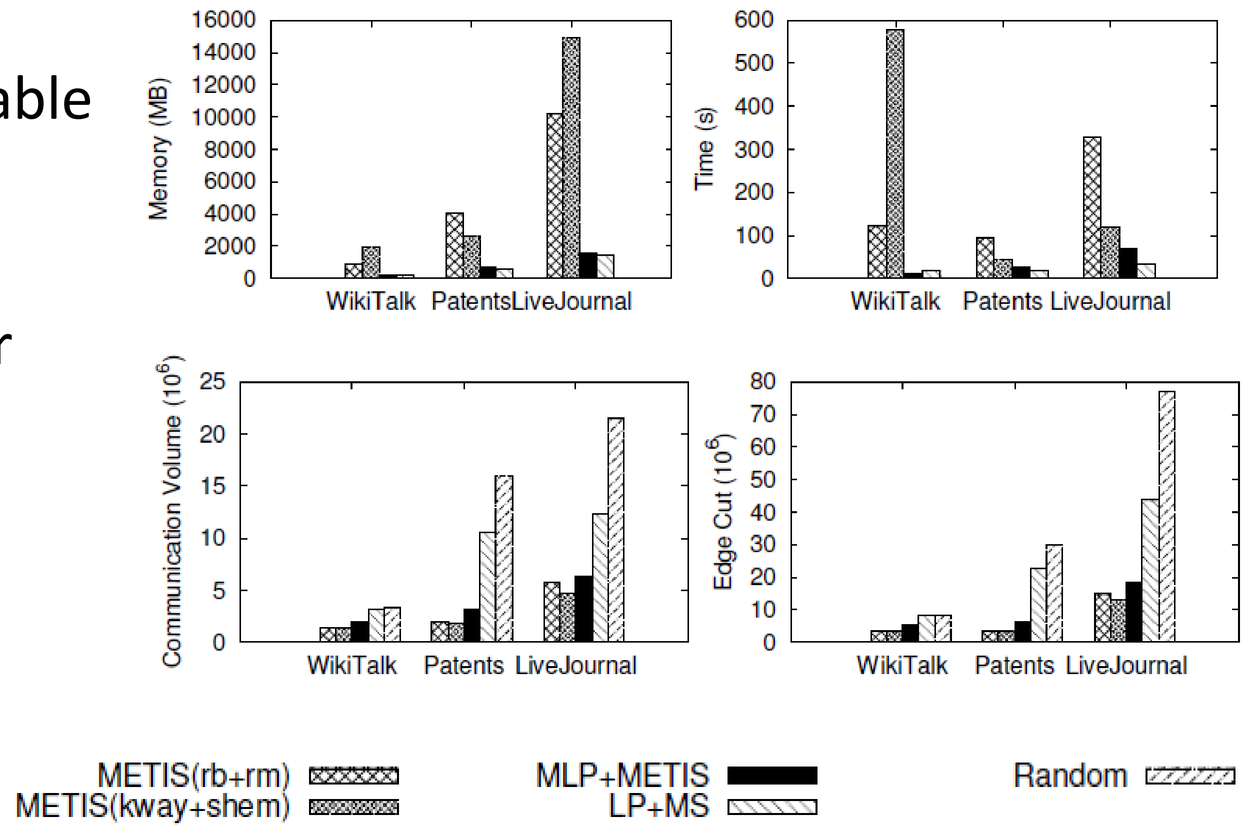
Handling Imbalance

- Imbalance:
  - Too many small clusters
  - Some extremely large clusters (monster clusters)

- Our approach
  - First, we set size constraint on each label, to limit monster clusters
  - Second, we use the following model to merge small cluster
    - *multiprocessor scheduling* (MS)
    - *weighted graph partitioning* (WGP)

Results - quality and performance

- Quality is comparable to Metis
- Performance is significantly better than Metis



Results- Scalability

- We use 8 machines, each of which has 48G memory.
- Our solution takes 4 hours on a graph with 512M node and 6.5G edges

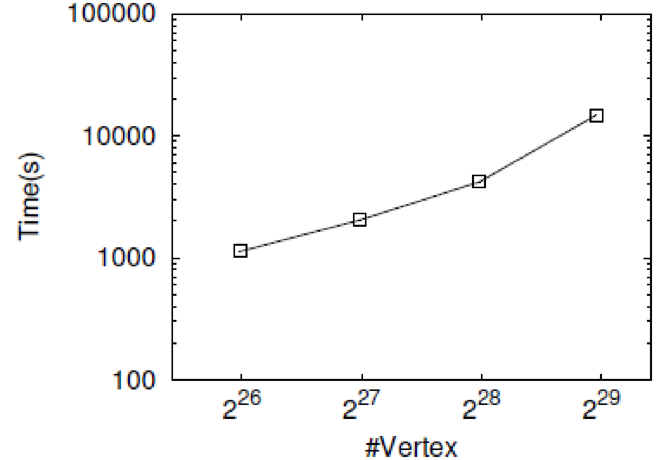


Figure 9: Scalability to billion-node graphs

Community search on large graphs

Distance Oracle

Why approximate distance oracle?

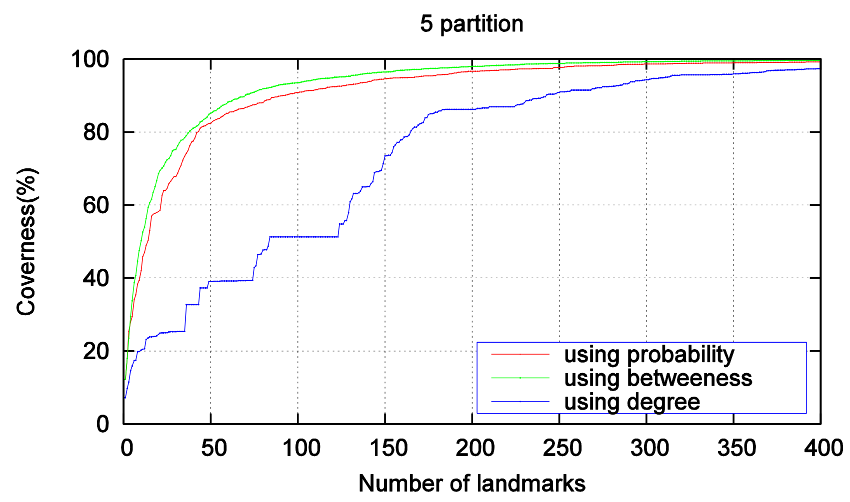
- What is your Erdos number?
  - The shortest distance from you to mathematician *Paul Erdos* in the scientific collaboration network
  - Shortest distance can also be used to compute centrality, betweenness
- Exact solutions
  - Online computation
    - Dijkstra-like algorithms, BFS
    - At least linear complexity, computational prohibitive on large graphs
  - Pre-computing all pairs of shortest path
    - Of quadratic space complexity
- Approximate distance oracle
  - Report *approximate* distance between any two vertices in a graph in constant time by *pre-computation*
  - When graph is huge, approximation is acceptable, and pre-computation is necessary

Current Solutions

- Two possible solutions
  - Coordinate based
  - Sketch based
- Problem solved
  - Approximate distance in (almost) constant time by pre-computation.
  - Currently for undirected, un-weighted graph only
  - With potential consideration for dynamic graphs

Distributed betweenness approximation

- Betweeness is the most effective guideline to select landmarks
- Exact betweenness costs O(nm) time, unacceptable on large graphs
- Approximate fast betweenness is expected
  - Count shortest paths rooted at sampled vertices
  - Count shortest path with limited depth
- On distributed platform
  - We count the shortest paths within each machines



Information about Trinity

- http://research.microsoft.com/trinity

Thanks!

Buffered Asynchronous Message Sender

- Lock free