SYSTAP, LLC

RDF

Graphs

Graph Databases

Graph Analytics on GPUs

Overview

- Introduction.
  - Who we are and what we do.
- What are graph query and graph mining?
  - Selective indices versus full visitation.
  - Memory bandwidth, $/GTEPS.
  - Implications for system design.
- GPU-accelerated Graph Mining
  - Challenges of ultra high throughput for graph mining on GPUs.
- RDF Graph Mining with SPARQL and vertex programs
- Link Attributes, Key-Value Stores, Property Graphs, RDF, RDR
  - Key value stores as materialized property sets for vertex.
  - Why link attributes are important
  - Fast and efficient representation of link attributes.
  - Stacking of fast world key value stores over graph databases for fast query.

SYSTAP, LLC

Small Business, Founded 2006

100% Employee Owned

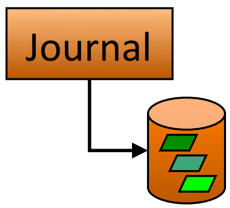
Graph Database

- High performance, Scalable
  - 50B edges/node
  - High level query language
  - Efficient Graph Traversal
  - High 9s solution
- Open Source (Subscriptions)
  - Autodesk, EMC, market data, genomics and personalized medicine, etc.

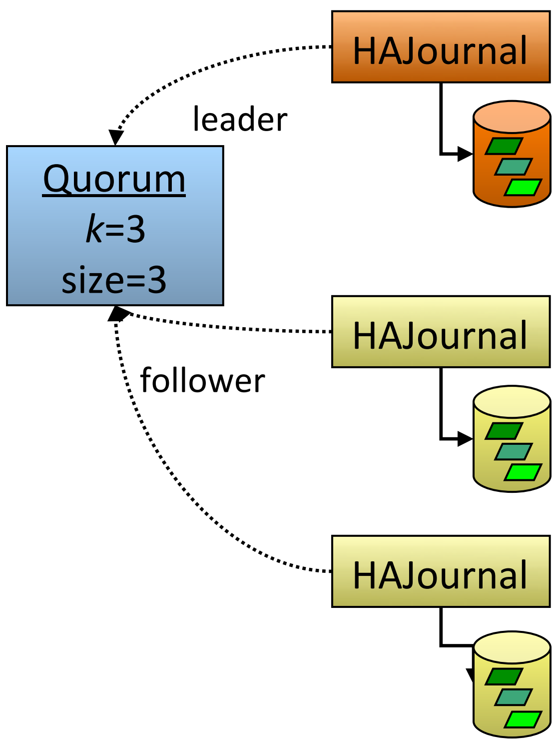
GPU Analytics

- Extreme Performance
  - 5-100x faster than graphlab
  - 10,000x faster than graphdbs
- DARPA funding
- Disruptive technology
  - Early adopters
  - Huge ROIs
- Open Source

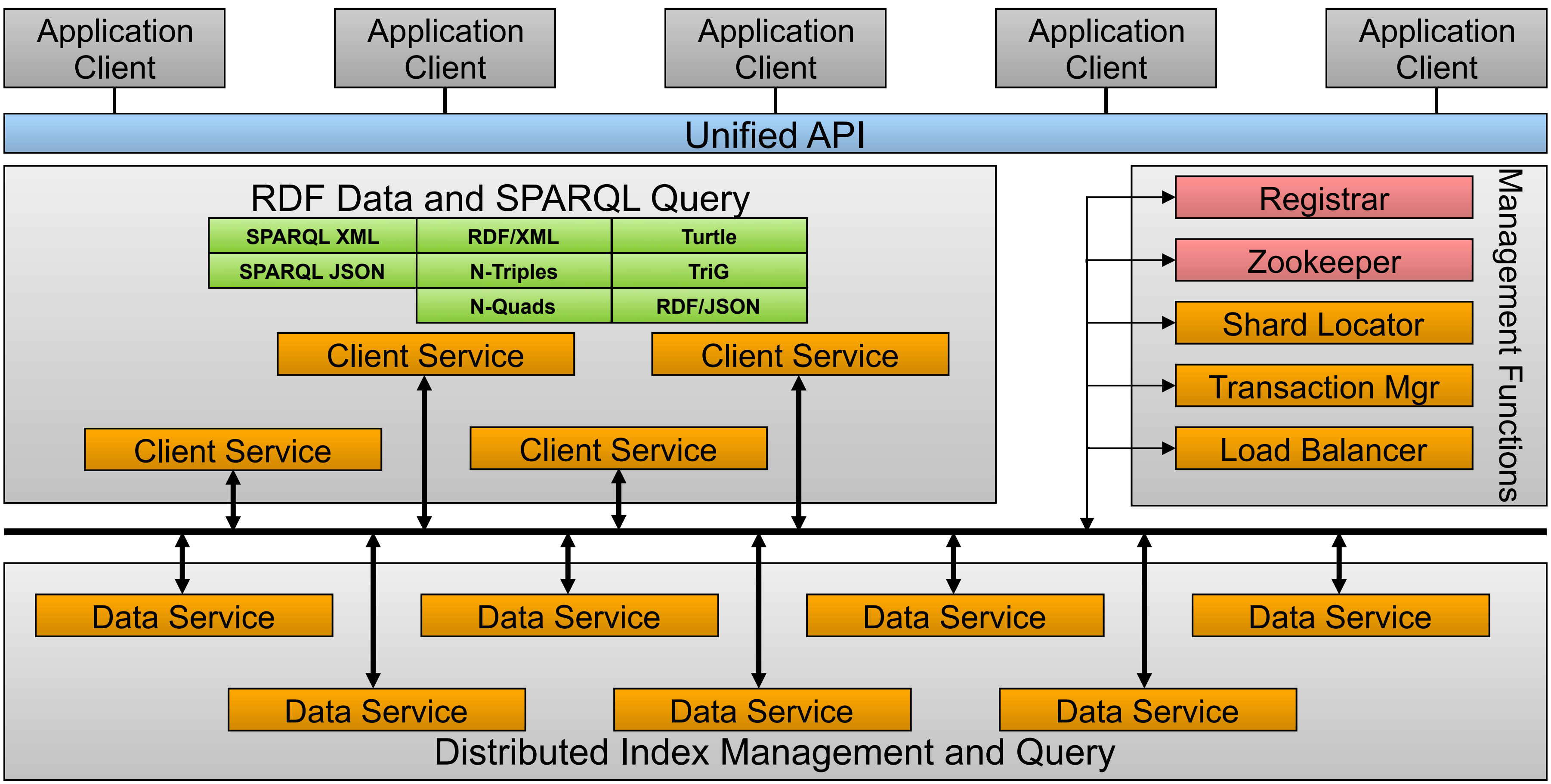
Embedded, Single Server, HA, Scale-out



Embedded, Single Server, HA, Scale-out

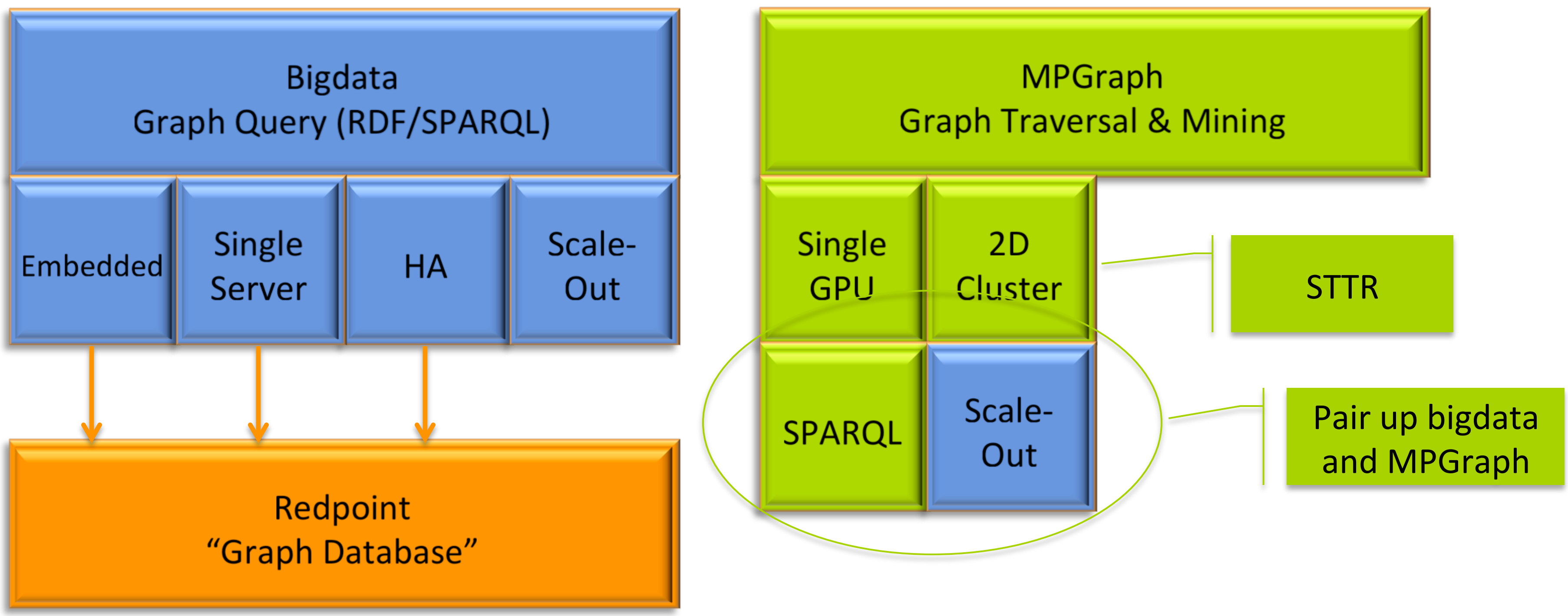


Embedded, Single Server, HA, Scale-out



And now on GPUs

Related "Graph" Technologies



Redpoint repositions existing technology.

MPGraph compares favorably with high end hardware solutions from YARC, Oracle, and SAP, but is open source and uses commodity hardware.

Similar models, different problems

- Graph query and graph analytics (traversal/mining)
  - Related data models
  - *Very* different computational requirements
- Many technologies are a bad match or limited solution
  - Key-value stores (bigtable, Accumulo, Cassandra, HBase)
  - Map-reduce
- Anti-pattern
  - Dump all data into “big bucket”

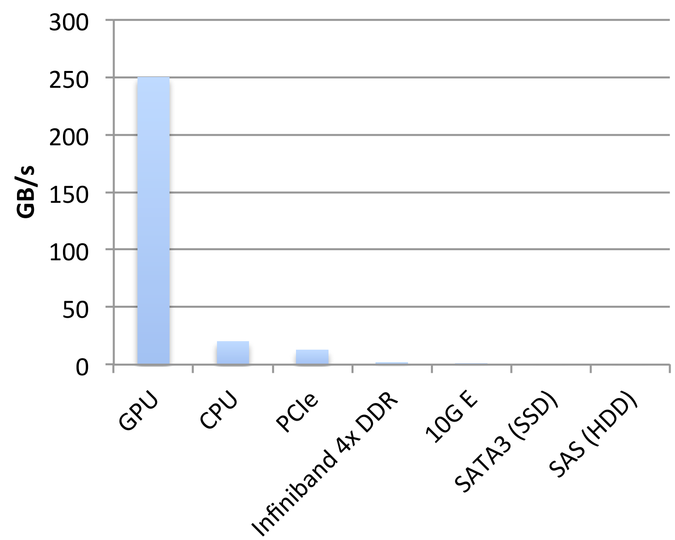
Similar models, different problems

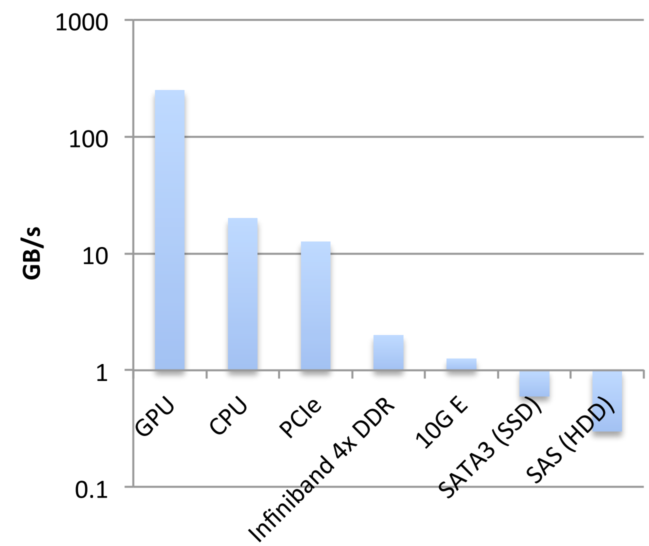
- Graph query and graph analytics (traversal/mining)
  - Related data models
  - *Very* different computational requirements
- Many technologies are a bad match or limited solution
  - Key-value stores (bigtable, Accumulo, Cassandra, HBase)
  - Map-reduce
- Anti-pattern
  - Dump all data into "big bucket"

Storage and computation patterns must be correctly matched for high performance.

Bandwidth (2014 - Kepler)

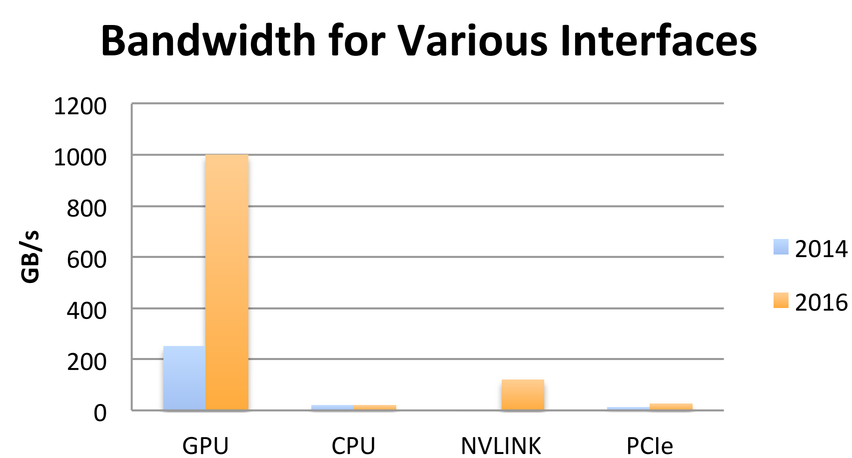
- Data intensive problems are *bandwidth* constrained.
- GPUs have significantly more bandwidth.
- Severe bottlenecks: PCIe, network, disk.
  - Plots are the same data: Linear on the left. Log on the right.





Bandwidth (2016 - Pascal)

- GPU bandwidth is increasing
  - Constant bandwidth/byte.
  - Increasing memory density (6G => 24G).
- NVLINK *exceeds* CPU memory bandwidth.
  - 60 GB/s to 144 GB/s, depending on OEM configuration.



Optimize for the right problem

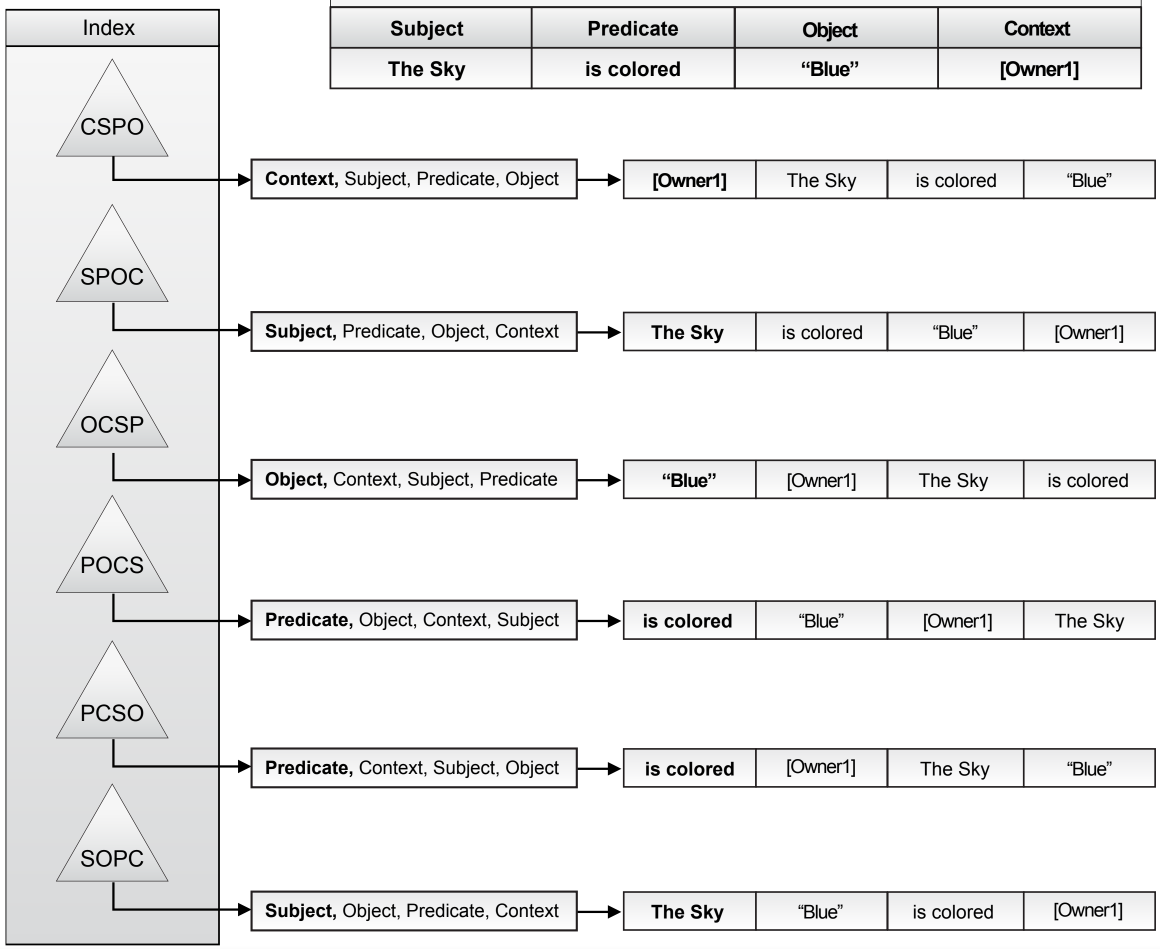
- Historical scale-out architectures are embarrassingly parallel.
  - This is not true for graphs.
- Wide-spread habit to using map/reduce and key-value stores for graphs.
  - This is a bad reflex.
- Map/Reduce
  - Research is warped by dealing with the overhead of the paradigm.
- Key-value stores
  - Do not expose the necessary capabilities for query
  - Do not organize the edges correctly for traversal

Optimize for the right problem

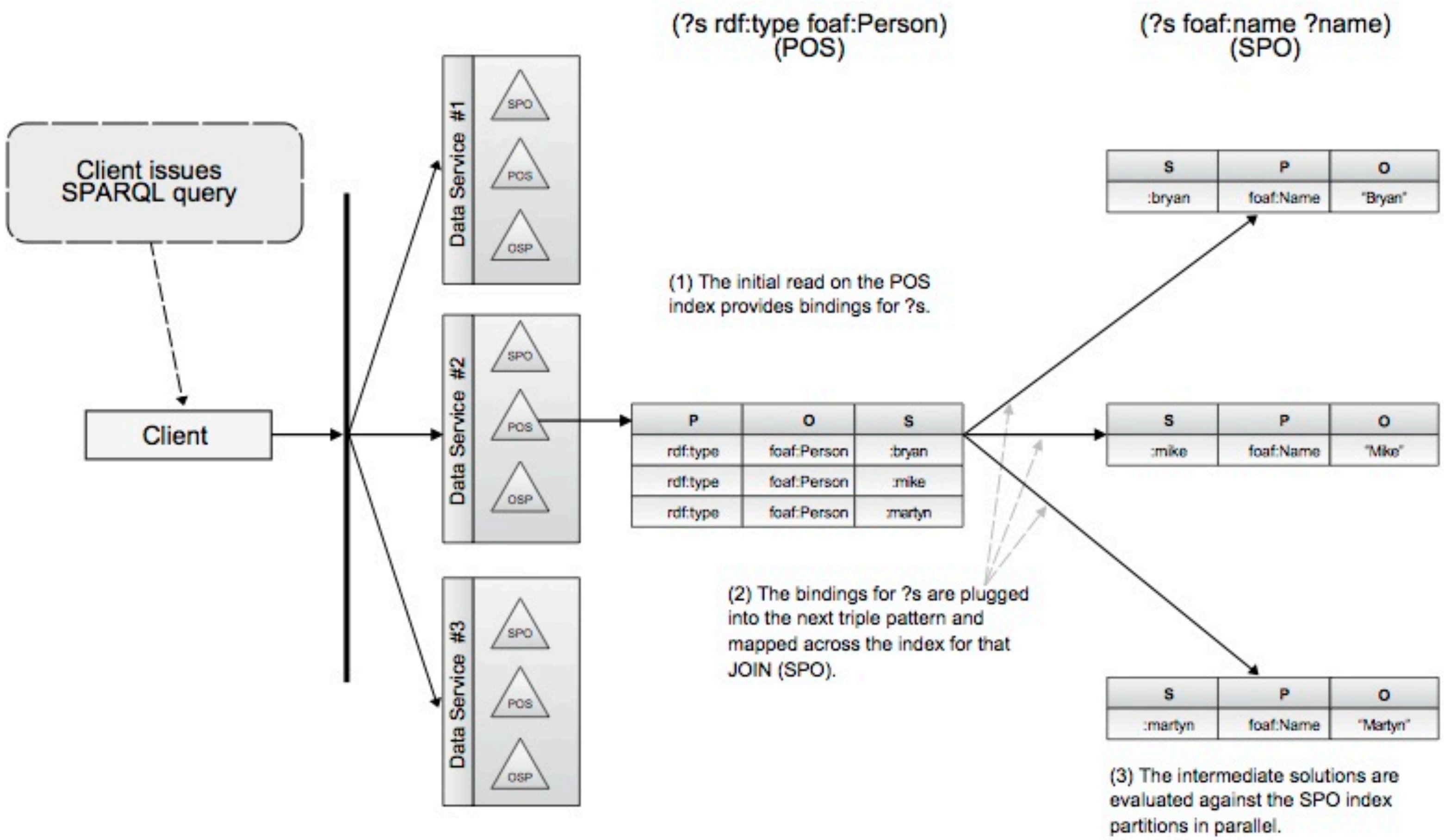
- Graph Query
  - Declarative Query Language (SPARQL)
    - Query optimization is critical for performance.
  - Index Locality (1D partitioning, multiple indices)
    - Get everything about a subject on one page of the index.
  - Scale-out must flow queries over the data
    - Otherwise slams the network and the client (e.g., RYA can not scale).
  - Must order and constrain joins to read as little data as possible
    - As-bound vectored nested index joins (bigdata)
    - Sideways information passing and merge joins (RDF3X)

Covering Indices (Quads)

RDF "Quad" Statement			
Subject	**Predicate**	**Object**	**Context**
The Sky	**is colored**	**"Blue"**	**[Owner1]**

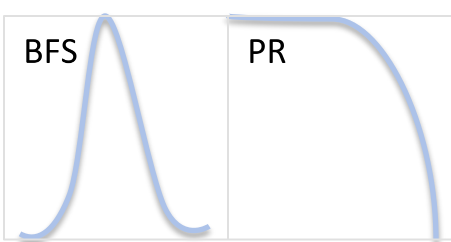


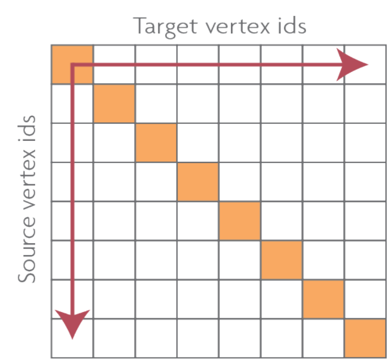
Vectored Query in Scale-Out



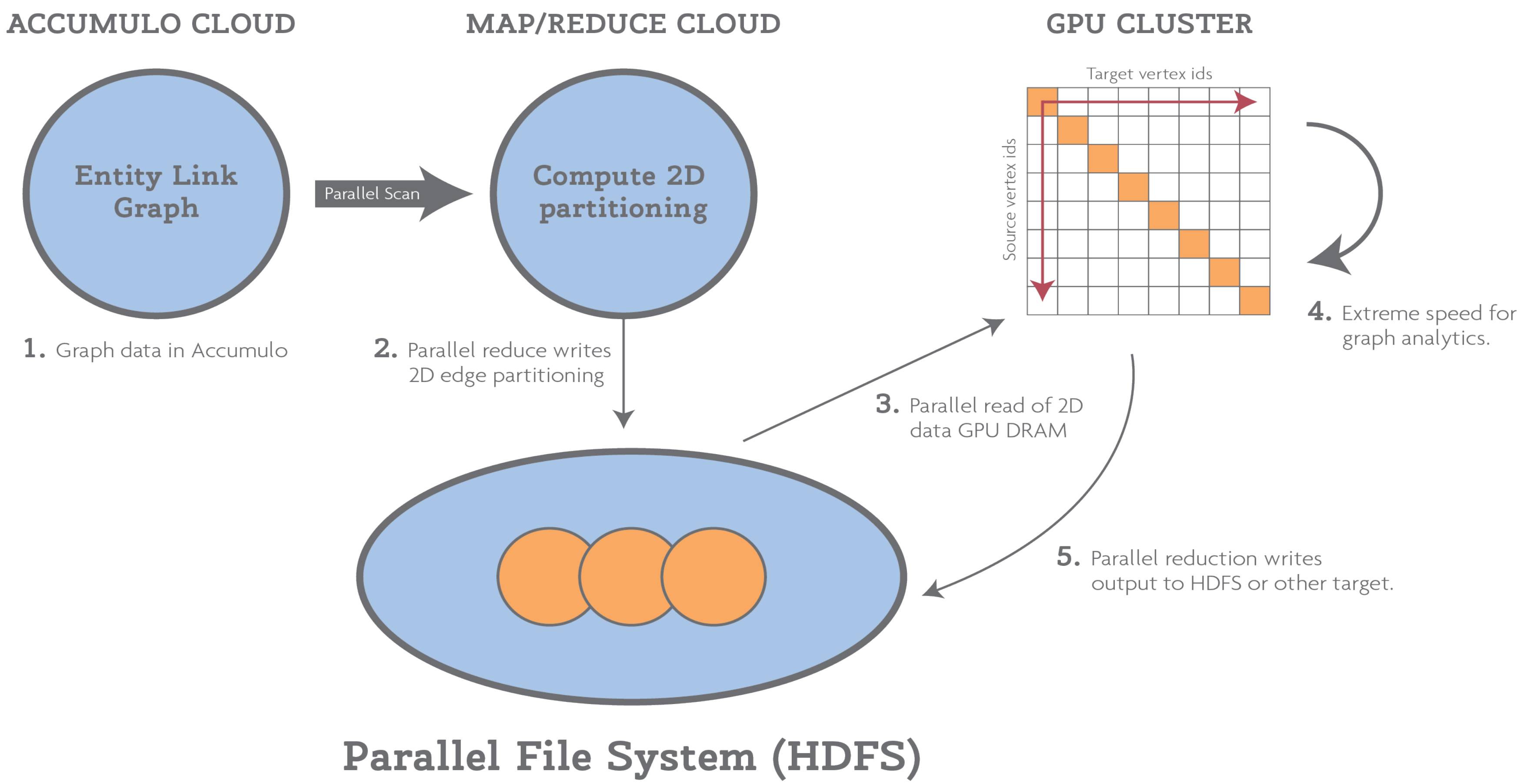
Optimize for the right problem

- Storage and computation patterns must be correctly matched for high performance.
- Graph analytics:
  - Parallelism – work must be distributed and balanced.
  - Memory bandwidth – memory, not disk, is the bottleneck
  - 2D partitioning – O(N) communications pattern (versus O(N*N))
    - 1D design looses locality when updating link weights for reverse indices.





Accelerated Graph Analytics



Fast Estimations of Traversals

- "Fast and Accurate Estimation of Shortest Paths in Large Graphs," Gubichev et al., 2010.
  - 193-4000 ms (TreeSketch) on graphs with up to 48M edges.
    - Run times on the order of 50-100ms with hot cache (unpublished).
    - Running time for Dijkstra is 4-119s on the same graphs (exact method).
    - The Orkut data set has 223M edges and 2700ms runtime. This graph might not fit into the 6G on a K20, but would fit into a 12G K40 (untested).
- "On the Embeddability of Random Walk Distances," X. Zhao et al, 2014.
  - Random walks on graphs with up to 1.6M edges in 80ms.
- A GPU has comparable or better performance with exact results.
  - BFS on 89M edges of a power-law graph is 47ms (MPGraph).
  - Do exact results matter? They could for cell-level security on graphs.

Graph Processing

GPUs

Graphs

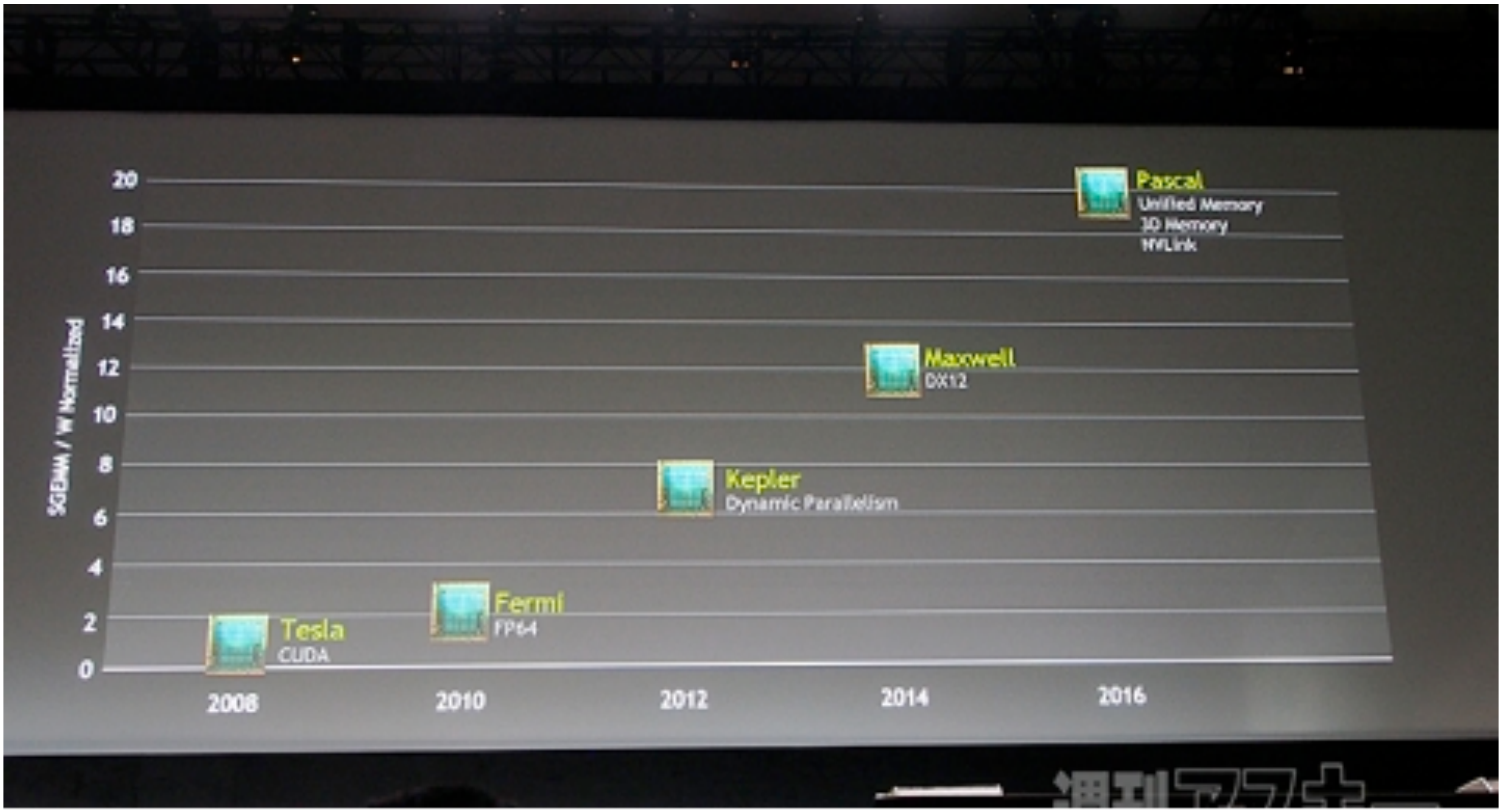
and

Graph Data Mining

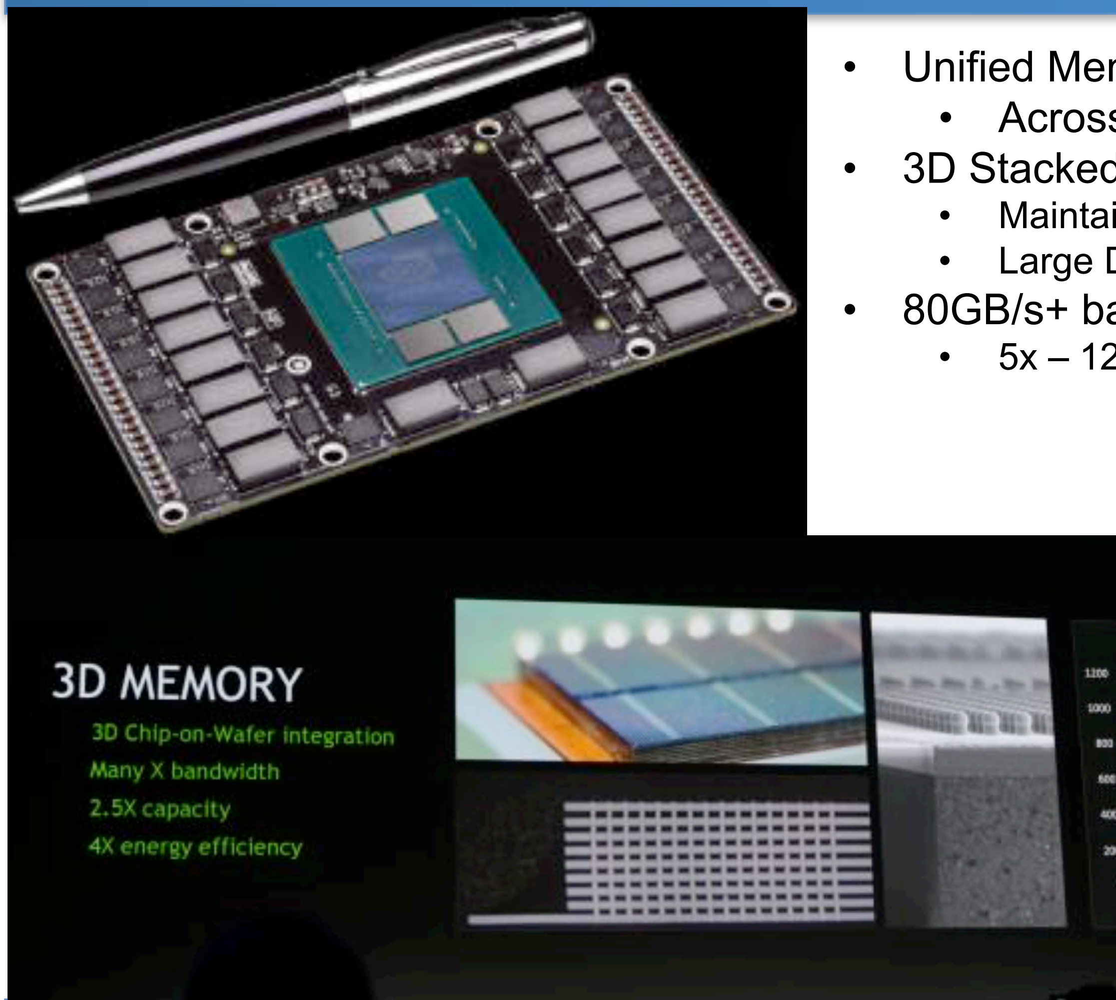
GPU Graph Processing

- Motivation – *speed*
  - 3 out of the top 5 super computers are GPU clusters
  - 3.3 B traversed edges per second (one GPU : Merrill, 2011)
  - 8.3 B traversed edges per second (quad GPU configuration : ibid)
- Goal
  - Blindingly fast SPARQL query and *graph traversal* on GPU clusters
  - 20 minutes on Accumulo => 27 milliseconds on a GPU.
- Open source
  - Deploy in workstations, HPC clusters, EC2, or your own data center

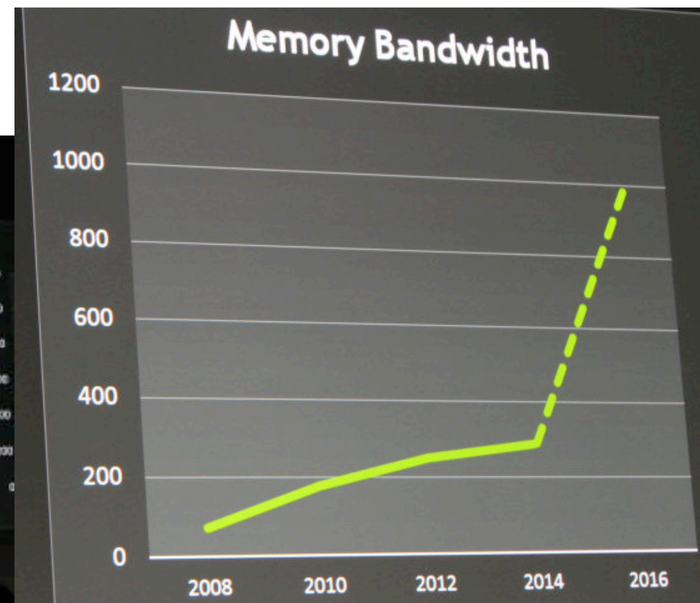
NVIDIA Roadmap



NVIDIA Pascal (2016,Q1)



- Unified Memory
  - Across CPU, GPUs
- 3D Stacked Memory @ 1000 GB/s
  - Maintains bandwidth / byte ratio
  - Large DRAM (24 GB+)
- 80GB/s+ bandwidth between GPUs (NVLINK)
  - 5x – 12x more bandwidth

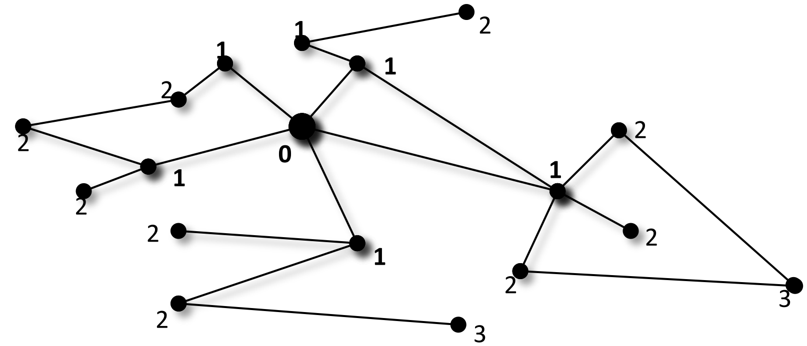


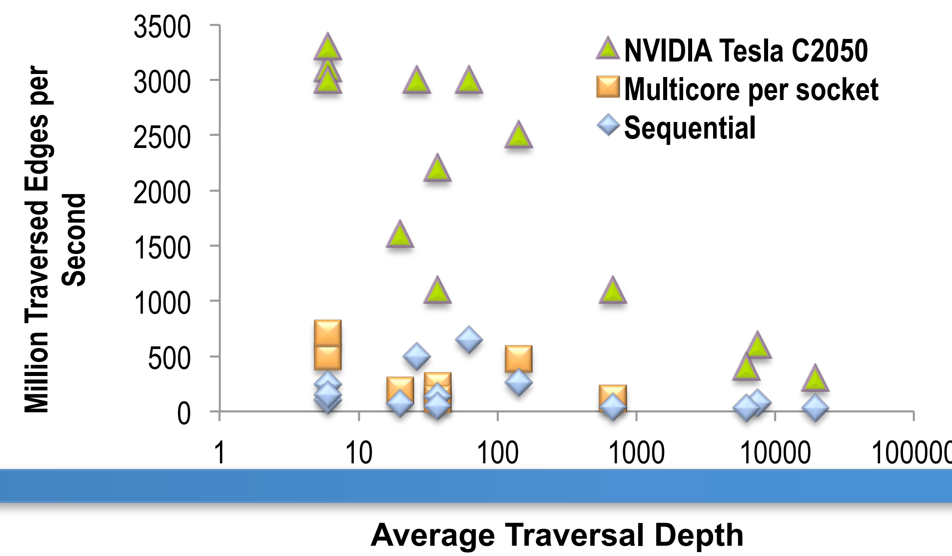
GPUs – A Game Changer for Graph Analytics?

- Graphs are everywhere in data, also a powerful data model for federation
- GPUs may be the technology that finally delivers real-time analytics on large graphs
  - 10x speedup over CPU
  - 10x DRAM bandwidth
- This is a hard problem
  - Data dependent parallelism
  - Non-locality
  - PCIe bus is bottleneck
- Significant speed up over CPU on BFS
  - 3 billion edges per second on one GPU (see chart).
- Roadmap
  - GPU accelerated vertex-centric graph mining platform.
  - GPU accelerated graph query

Breadth-First Search on Graphs

10x Speedup on GPUs





MPGraph

- High-level graph processing framework
  - High programmability
    - GPU architecture
    - Optimization techniques
    - CUDA
  - High performance
    - Comparable to low-level approach

MPGraph

- High-level graph processing framework
  - High programmability
    - ~~GPU architecture~~
    - ~~Optimization techniques~~
    - ~~CUDA~~
  - High performance
    - Comparable to low-level approach

Think Like a Vertex

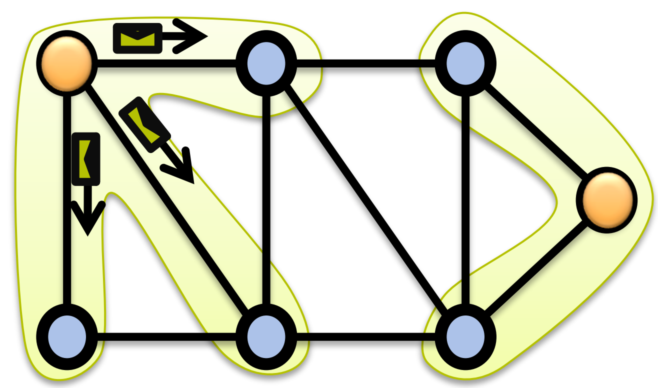
- Simple APIs

```
pageRank(Message m) {
    total = m.value();
    vertex.val = .15 * .85 + total;
    for(nbr : out_neighbors) {
        SendMsg(nbr, vertex.val/num_out_nbrs);
    }
}
```

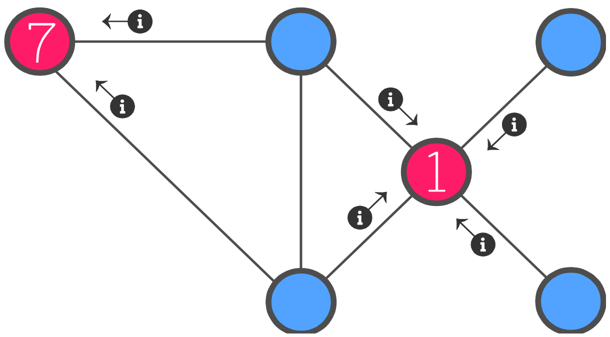
- Lots of algorithms
  - Betweenness-Centrality, Personalized Page Rank, k-means clustering, Loopy Belief Propagation, Graph search (crisp and approximate), etc.
- Lots of platforms
  - Pregel, Sedge, Signal/Collect, graphlab, PowerGraph, Apache Griaph, Apache Hama, GoldenOrb, Knowledge Discovery Kit, etc.

GAS – a Graph-Parallel Abstraction

- Graph-Parallel Vertex-Centric API ala GraphLab
- "Think like a vertex"
- **G**ather: collect information about my neighborhood
- **A**pply: update my value
- **S**catter: signal adjacent vertices
- Can write all sorts of graph algorithms this way
  – BFS, PageRank, Connected Component, Triangle Counting, Max Flow, etc.

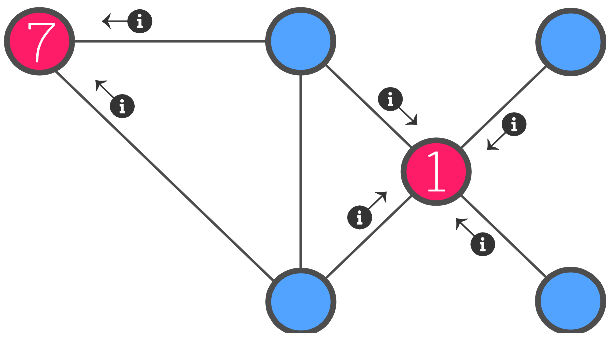


GAS Abstraction

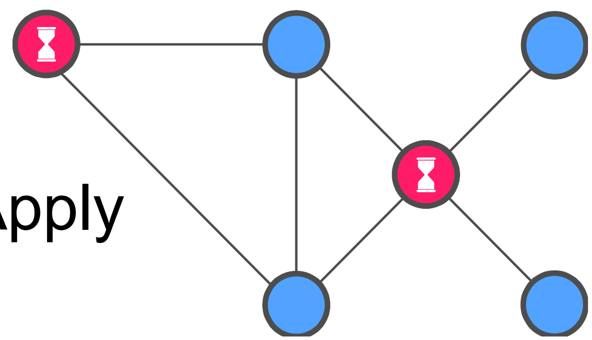


Gather

GAS Abstraction

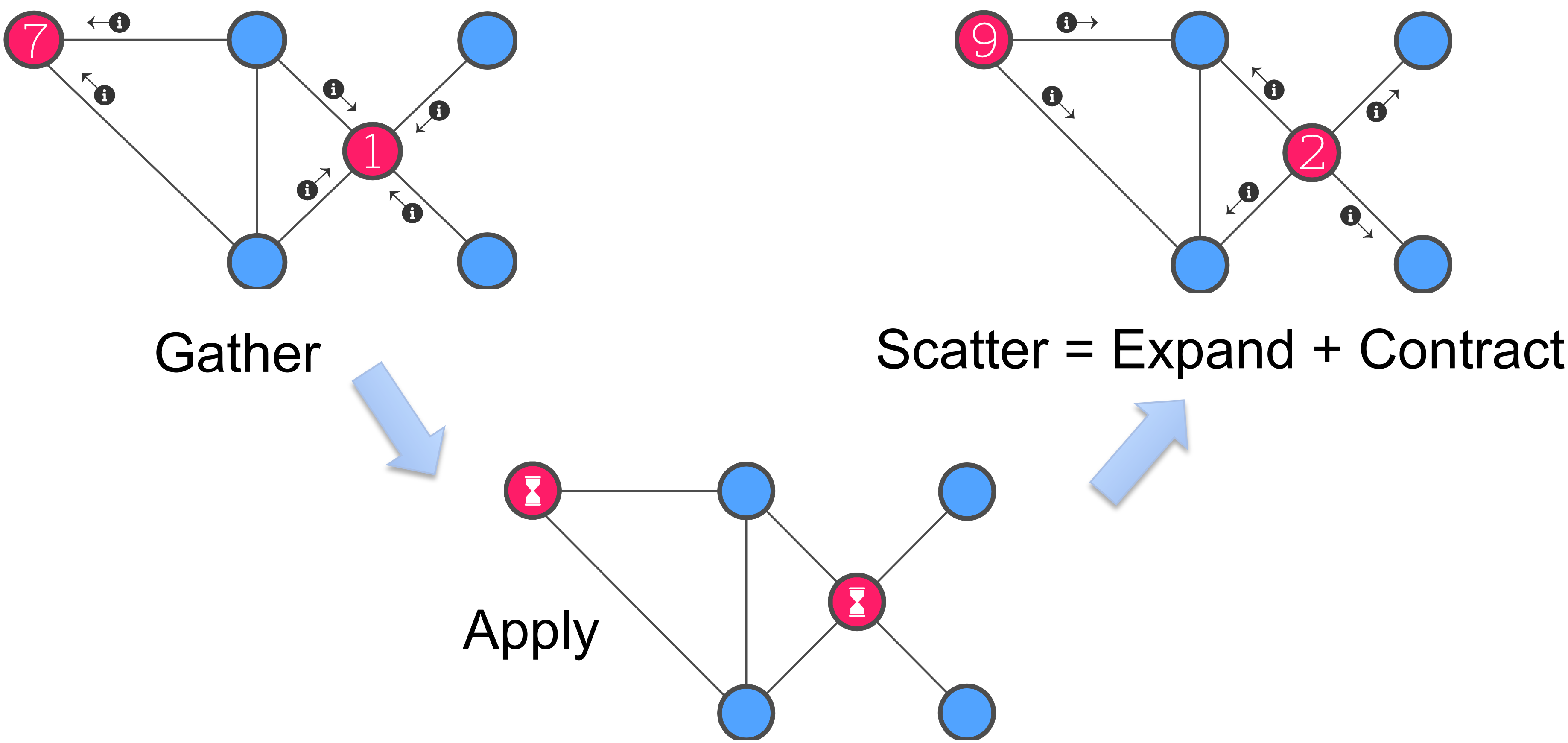


Gather

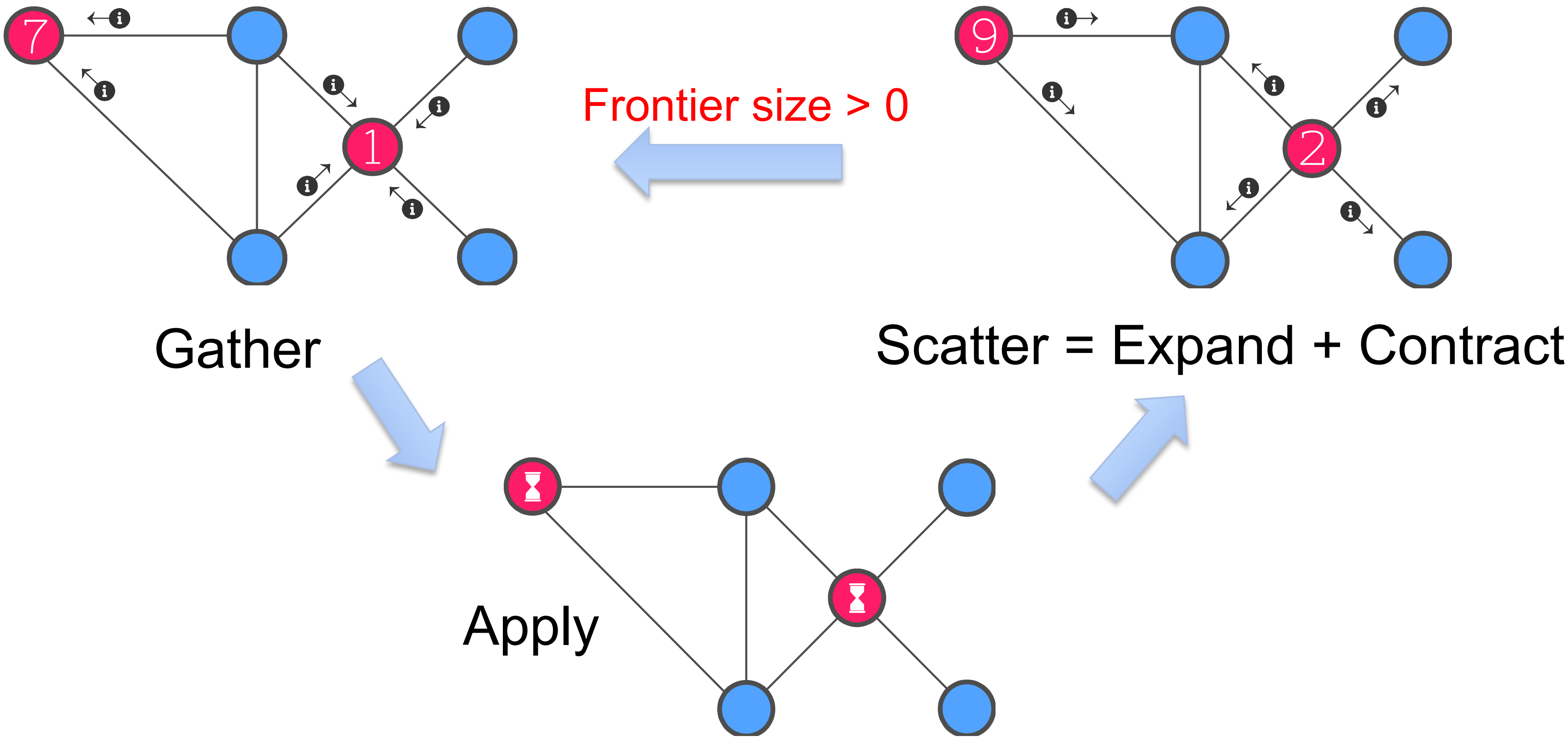


Apply

GAS Abstraction



GAS Abstraction



Example: BFS Implementation

- BFS is just a Scatter operation (also true for SSSP).

Data	VertexType	int[] d_label // vertex depth
Scatter	expandOverEdges()	OUT_EDGES
	expand_vertex()	true // always expand
	expand_edge()	frontier = neighbor_id // visit neighbor

	contract()	int label = d_label[vertex_id];
		if (label != -1) { // already labeled?
		  vertex_id = -1; // de-activate
		} else { // label & schedule
		  d_label[vertex_id] = iteration + 1;
		}

Example: PageRank Implementation

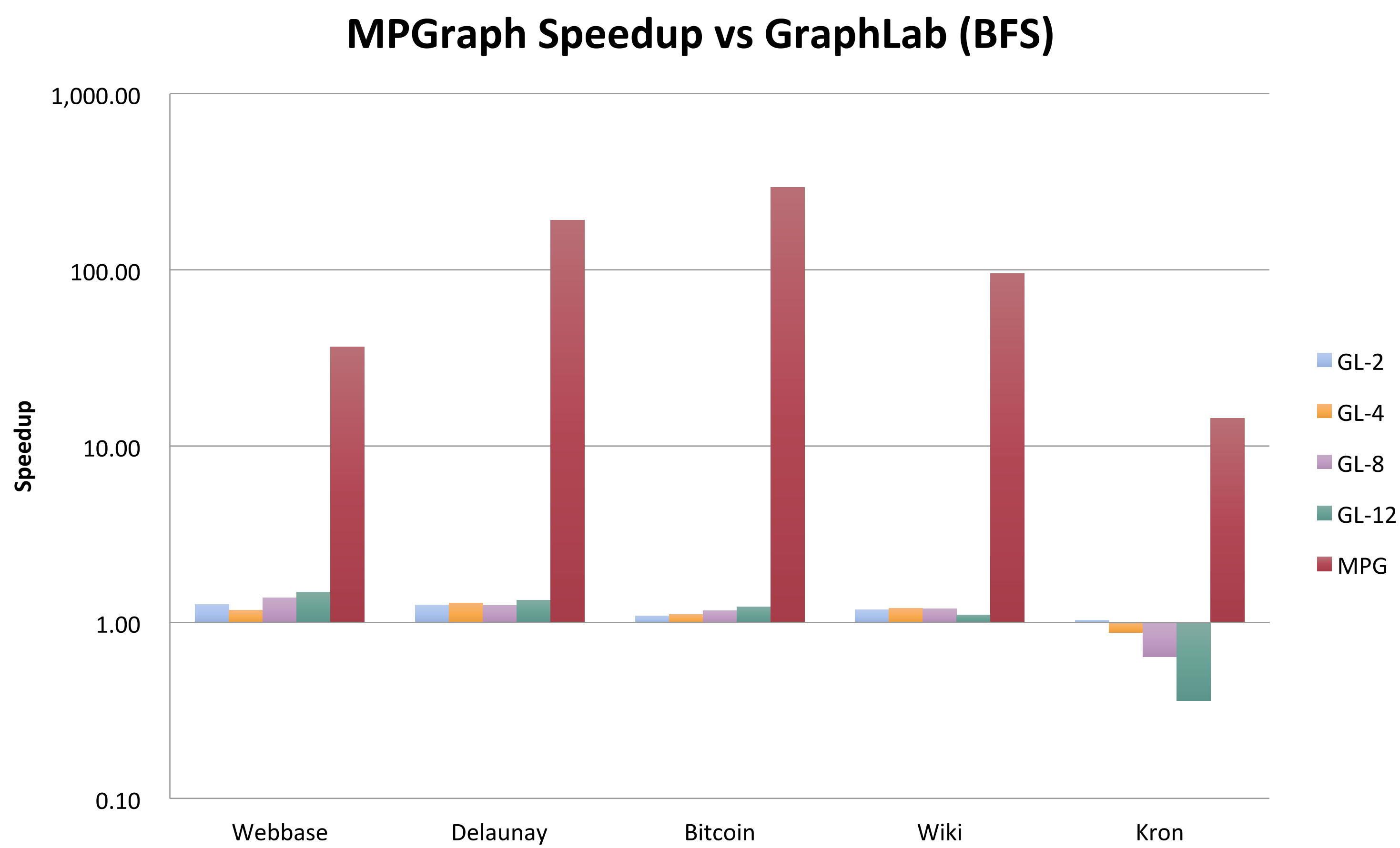
- PageRank uses all three phases.

User Data	VertexType	float[] d_rank // page rank for vertex int[] d_nout // #of out edges for vertex
Gather	gatherEdges()	IN_EDGES
	gather_edge()	d_rank[neighbor_id] / d_nout[neighbor_id]
	gather_sum()	left + right // binary sum

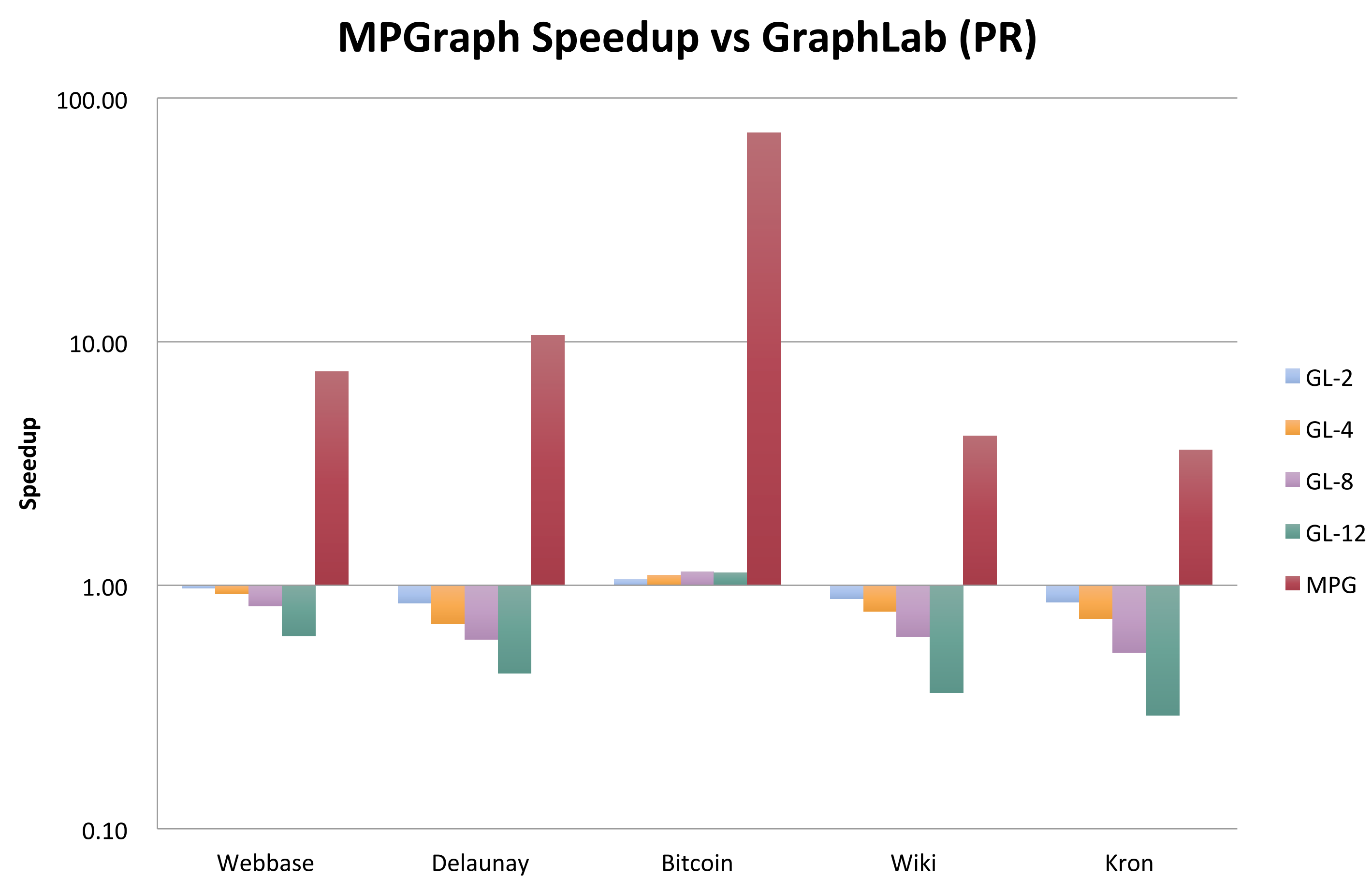
Apply	apply()	float oldval = d_rank[vertex_id]; // get old rank
		float newval = 0.85f * gathervalue + 0.15f;
		changed = fabs(oldval – newval) >= 0.01f;
		d_rank[vertex_id] = newval; // put new rank

Scatter	expandEdges()	OUT_EDGES
	expand_vertex()	Changed // expand iff changed
	expand_edge()	frontier = neighbor_id // visit neighbor

BFS Results : MPGraph vs GraphLab

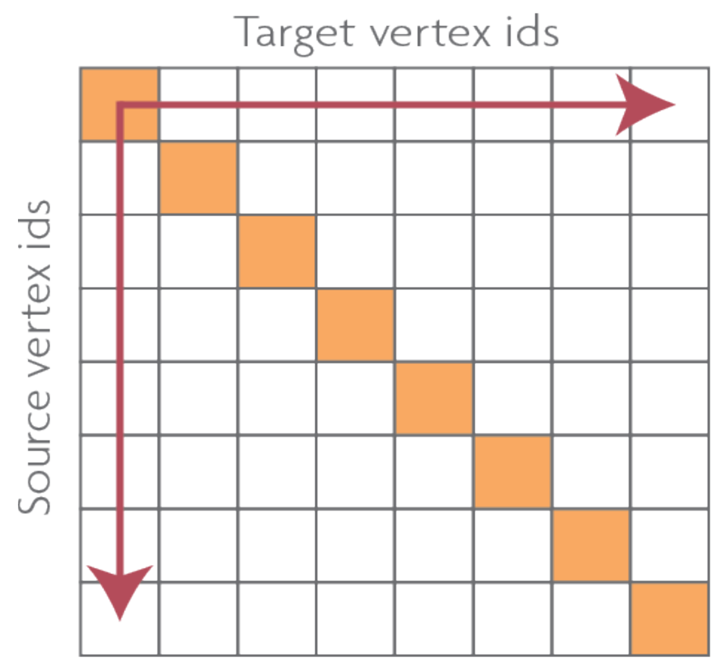


PageRank : MPGraph vs GraphLab



Graph Mining on GPU Clusters

- 2D partitioning (aka vertex cuts)
- Minimizes the communication volume.
- Batch parallel Gather in row, Scatter in column.



Graph Processing

RDF

SPARQL

GPUs

Ideal Approach

- RUN {*vertex-program*}
  - FROM data source(s)
  - SELECT vertex projection
    - WHERE graph-pattern(s)

- Declarative query extracts data of interest
- Data dynamically partitioned onto cluster
- Vertex program runs over that data.

- It should be that easy.

Graph Mining using SPARQL

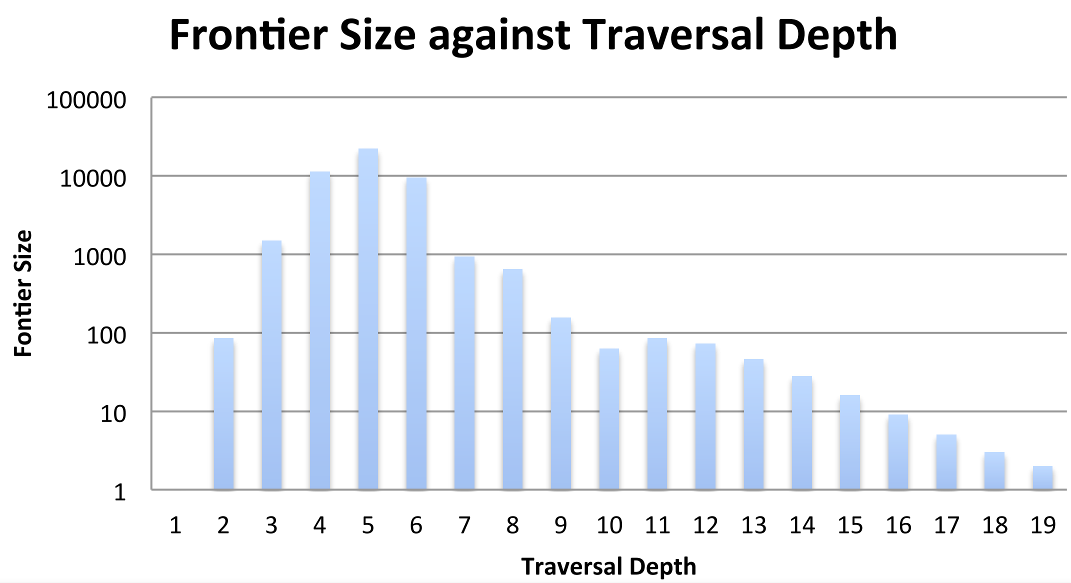
- Gather Apply Scatter (GAS) model for RDF graphs
  - Can specify restrictions on the link types visited.
  - Efficient parallel execution on the server (no round trips)
- GAS Algorithms implemented using simple Java API
  - BFS, SSSP, CC, PageRank, etc.
  - Easy to write more algorithms:
    - http://wiki.bigdata.com/wiki/index.php/RDF_GAS_API
- Graph algorithms are trivially combined with SPARQL
  - "SERVICE" abstraction for GAS algorithm execution.
  - Will support execution against GPU soon.

SPARQL Graph Traversal (BFS)

```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>
SELECT ?depth (count(?out) as ?cnt) {
 SERVICE gas:service {
   gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" .
   gas:program gas:in <http://www.w3.org/People/Berners-Lee/card#i> . # one or more times.
   gas:program gas:out ?out . # exactly once - will be bound to the visited vertices.
   gas:program gas:out1 ?depth . # exactly once - will be bound to the depth of the visited vertices.
  }
}
group by ?depth
order by ?depth
```

\# Query is ~ 325 ms on about 306,805 edges

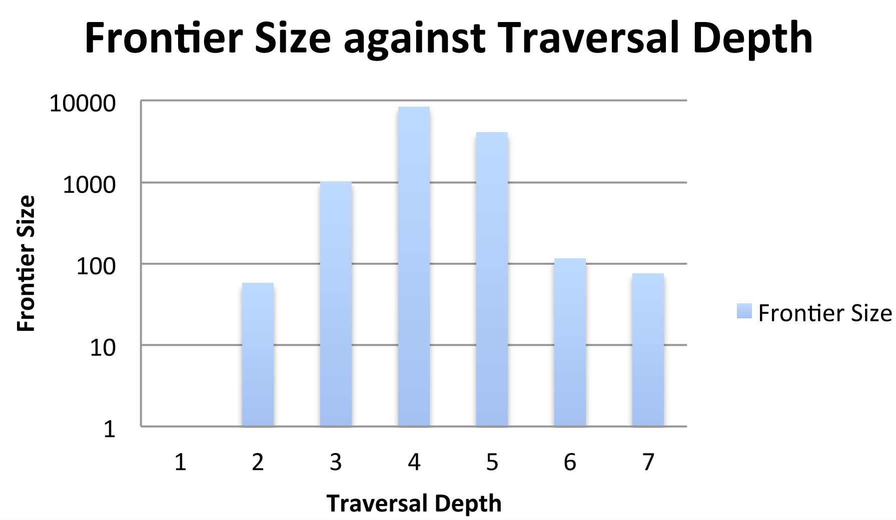
\# Query runs against bigdata (NOT MPGraph).



SPARQL Graph Traversal (BFS)

```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?depth (count(?out) as ?cnt) {
  SERVICE gas:service {
     gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" .
     gas:program gas:in <http://www.w3.org/People/Berners-Lee/card#i> . # one or more times.
     gas:program gas:out ?out . # exactly once - will be bound to the visited vertices.
     gas:program gas:out1 ?depth . # exactly once - will be bound to the depth of the visited vertices.
     gas:program gas:linkType foaf:knows . # optional restriction on the type of traversed links.
    }
}
group by ?depth
order by ?depth


# Query is ~ 120 ms on about 306,805 edges
# Query runs against bigdata (NOT MPGraph).
```



Graph Traversal - Predecessor

```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>
SELECT ?depth ?out ?predecessor {
 SERVICE gas:service {
   gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" .
   gas:program gas:in <http://www.w3.org/People/Berners-Lee/card#i> . # one or more times.
   gas:program gas:target <http://www.w3.org/People/all#eric> . # only paths to this vertex.
   gas:program gas:out ?out . # exactly once - will be bound to the visited vertices.
   gas:program gas:out1 ?depth . # exactly once - will be bound to the depth of the visited vertices.
   gas:program gas:out2 ?predecessor . # exactly once - will be bound to the predecessor.
 }
}
order by ?depth
```

Query is ~ 250 ms on about 306,805 edges.

Query runs against bigdata (NOT MPGraph).

Query provides a minimum hop path to target.

Depth	Predecessor
0	[unbound]
1	http://www.w3.org/People/Berners-Lee/card#i
2	http://www.ivan-herman.net/foaf.rdf#me
3	http://semanticweb.org/id/Ivan_Herman
4	http://www.ivan-herman.net/me
5	http://www.ivan-herman.net/foaf
6	http://www.ivan-herman.net/foaf#me
7	http://www.w3.org/2001/sw/#activity
8	t175704
9	t175706

Graph Traversal – Extract Subgraph

```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?depth ?out ?p ?o {
  SERVICE gas:service {
    gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS" .
    gas:program gas:in <http://www.w3.org/People/Berners-Lee/card#i> . # one or more times.
    gas:program gas:out ?out . # exactly once - will be bound to the visited vertices.
    gas:program gas:out1 ?depth . # exactly once - will be bound to the depth of the visited vertices.
    gas:program gas:maxIterations 4 . # optional limit on breadth first expansion.
    gas:program gas:linkType foaf:knows . # optional restriction on the type of traversed links.
  }
  ?out ?p ?o . # extract all links and vertex attributes for the visited vertices.
}

# Query is ~ 940 ms on about 306,805 edges and extracts 77,883 statements.
# Query runs against bigdata (NOT MPGraph).
```

Query extracts all “edges” for “vertices” touched by an N-hop BFS traversal.

Fun with Database Schema

RDF, Reification Done Right (RDR), Key-Value Stores (Accumulo), Security, and Bi-Temporal data

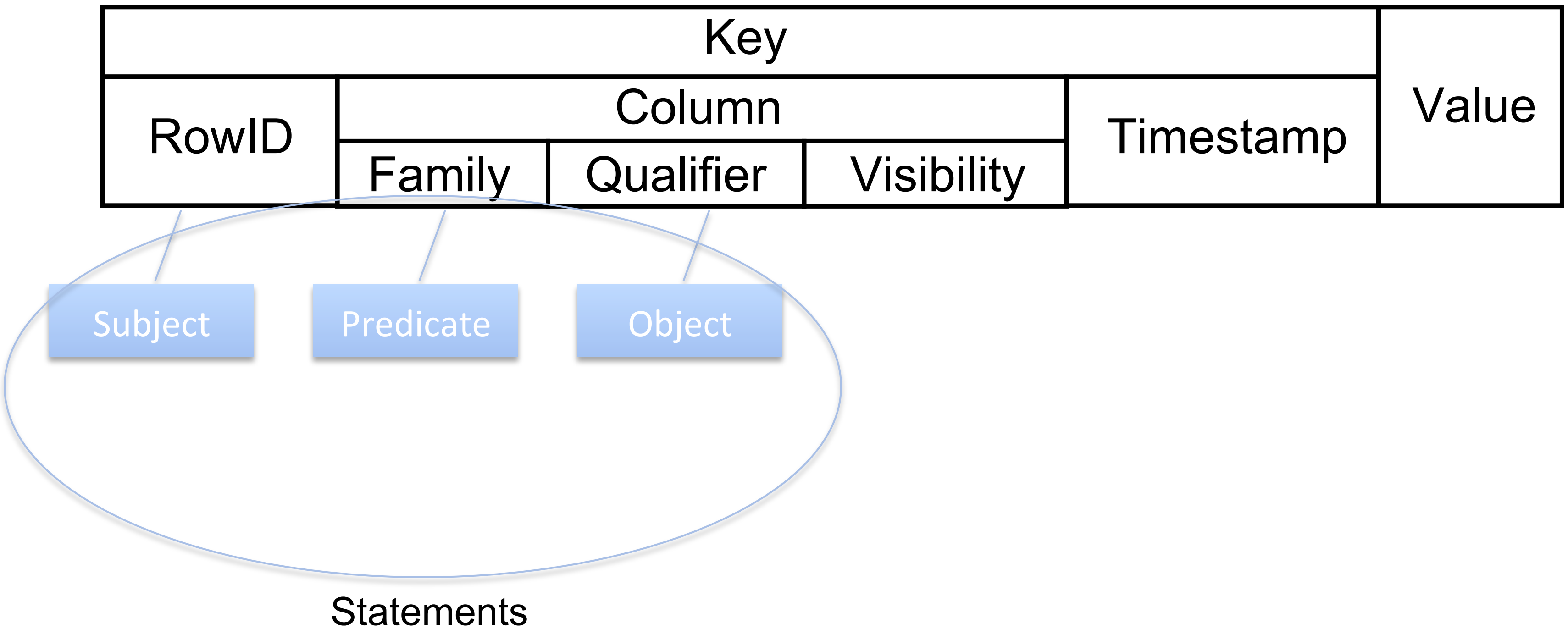
Goal is to unify approaches

- My goal here is to point out that
  - (a) Link attributes are *very* useful; and
  - (b) We can unify a wide variety of approaches.
- Each of these approaches has its reasons and its weaknesses.
- Eventually, we should be able to interoperate efficiently access these models.

Modeling Graphs with Accumulo

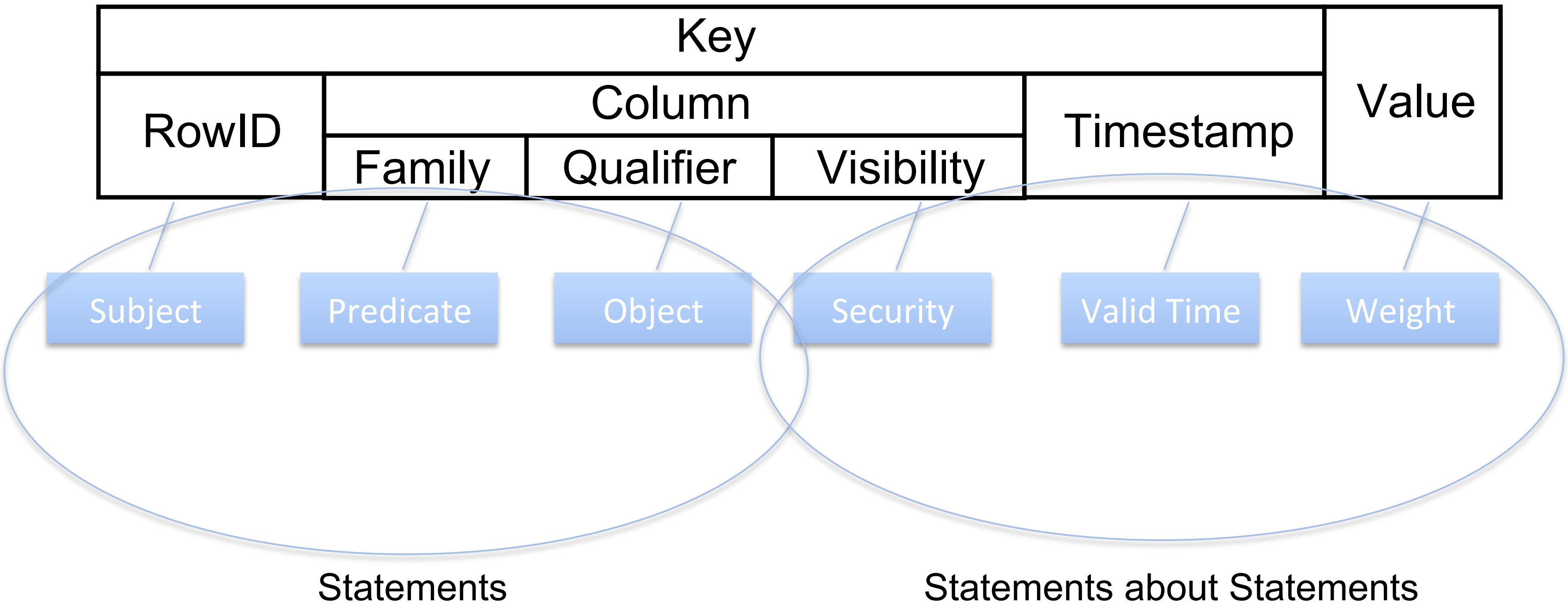
Key					Value
RowID	Column			Timestamp	
	Family	Qualifier	Visibility		

Modeling Graphs with Accumulo



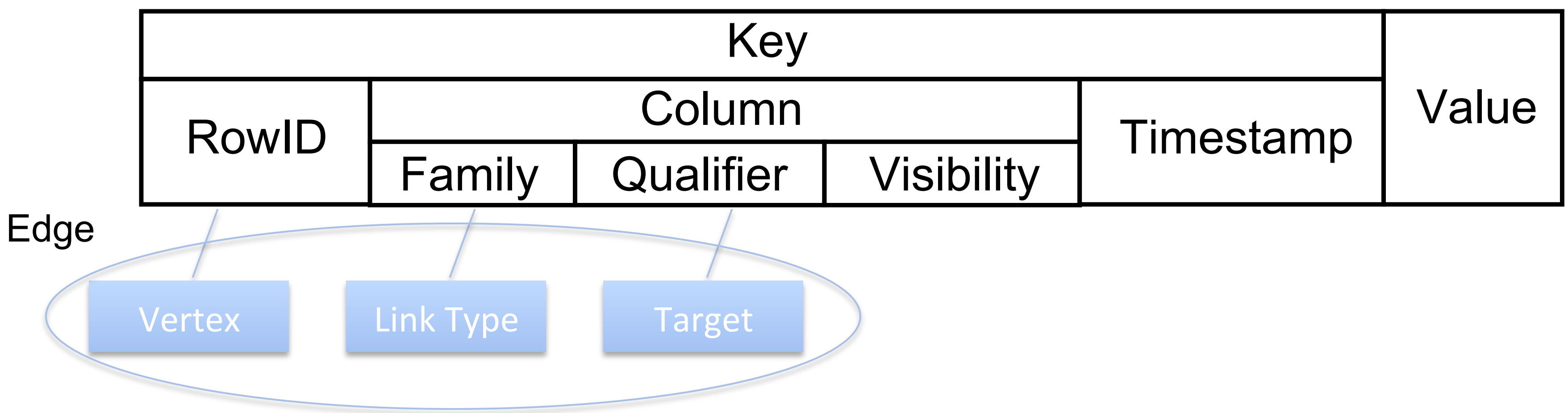
RDF Data Model

Modeling Graphs with Accumulo



RDF Data Model

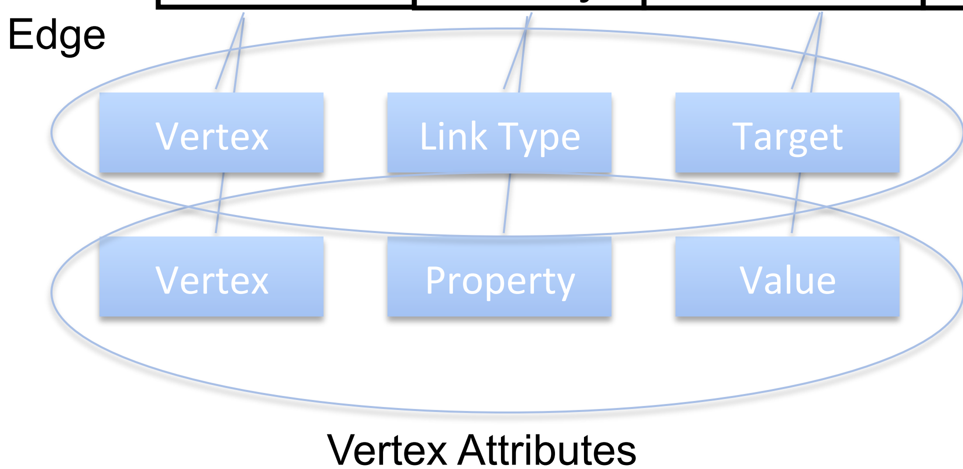
Modeling Graphs with Accumulo



Property Graph Model

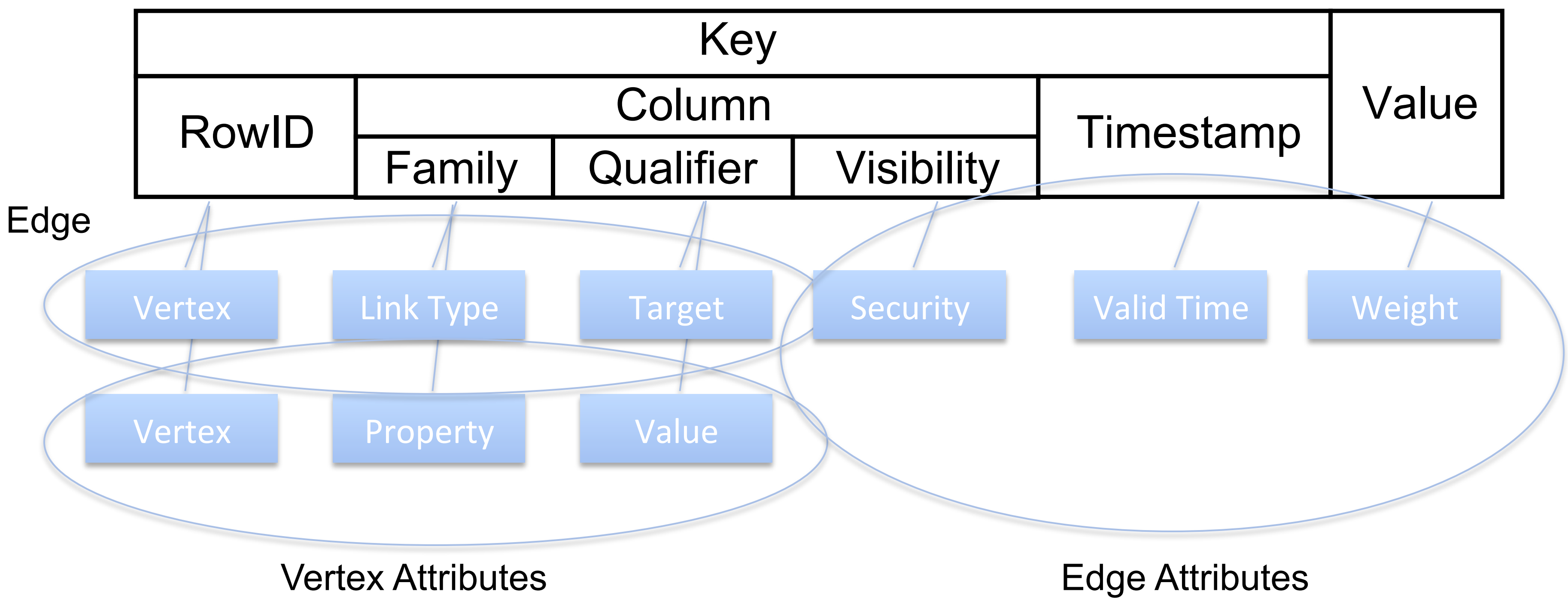
Modeling Graphs with Accumulo

<table>
<tr><th colspan="5">Key</th><th rowspan="3">Value</th></tr>
<tr><td rowspan="2">RowID</td><td colspan="3">Column</td><td rowspan="2">Timestamp</td></tr>
<tr><td>Family</td><td>Qualifier</td><td>Visibility</td></tr>
</table>



Property Graph Model

Modeling Graphs with Accumulo



Property Graph Model

Reification Done Right

RDF Graphs with efficient link attributes

Statement Level Metadata

- Important to know where data came from in a mashup

- <u>:mike :memberOf :SYSTAP .</u>
- source dc:source <http://www.systap.com> .

- But you CAN NOT say that in RDF.

RDF "Reification"

- Creates a "model" of the statement:

```
_:s1 rdf:subject :mike .
_:s1 rdf:predicate :memberOf .
_:s1 rdf:object :SYSTAP .
_:s1 rdf:type rdf:Statement .
```

- Then you can say:

```
_:s1 dc:source <http://www.systap.com> .
```

Reification Done Right

- Outcome from Dagstuhl 2012 Semantic Data Management workshop.
  - Collaborative effort with SYSTAP, Open Link, Humboldt University, Karlsruhe Institute of Technology.
- Harmonized with RDF model theory & SPARQL algebra.
  - Olaf Hartig, "Specification of a Reification Extension for SPARQL" - http://www.bigdata.com/whitepapers/reifSPARQL.pdf
  - Extensions for N3, TURTLE, and SPARQL are proposed.

Works with triples or quads

- *Inline* statements into statements.

```
<< :mike :memberOf :SYSTAP >>
        dc:source <http://www.systap.com> .
```

- Same syntax works for query

```
select ?s ?o ?source where {
   << ?s :memberOf ?o >> dc:source ?source .
}
```

RDR Use Cases

- Uniform approach for:
  - Time series data
  - Datum level security models
  - Link attributes (we've already seen this)
  - Bi-temporal systems (backup slides)
- Subsumes the blueprints model
  - Blueprints attributes are simple objects.
  - Attribute cardinality must be zero or one.
  - Does not allow links about links (aka statements about statements).
- Database free to choose efficient physical schema:
  - Reified statement models are an option, not a necessity.
  - Inline statements into statements (variable length identifiers).
  - Rotate link attributes into a null-able columns.

Basic Triple Table Schema

Subject	Predicate	Object
:widget1	:hasColor	:red
:widget1	:hasColor	:green

Primary Key

Time series data (key-value stores)

Subject	Predicate	Object	timestamp
:widget1	:hasColor	:red	t1
:widget1	:hasColor	:green	t12

Primary Key

<<:widget1, :hasColor, :red>> :timestamp :t1 .
<<:widget2, :hasColor, :green>> :timestamp :t12 .

- The "timestamp" column is Statements about Statements.
  - This is NOT a wide table for the properties of a subject.
- The timestamp part of primary key for key-value stores.
  - How do we express this cardinality constraint?

Cell-Level Security (Accumulo)

Subject	Predicate	Object	timestamp	Visibility
:widget1	:hasColor	:red	t1	Public
:widget2	:hasColor	:green	t12	Private

Primary Key

Cell level Security

<<:widget1, :hasColor, :red>> :visibility :Public .
<<:widget2, :hasColor, :green>> :visibility :Private .

- Security is Statements about Statements.

Bi-Temporal Data

<<:widget1, :hasColor, :red>>

:businessStartTime "2013-01-01T00:00:00"^^<xsd:#dateTime> ;

:businessStopTime ”2013-04-01T00:00:00"^^<xsd:#dateTime> ;

:transactionTime "2012-11-14T15:10:22"^^<xsd:#dateTime> .

<<:widget1, :hasColor, :green>>

:businessStartTime "2013-04-01T00:00:00"^^<xsd:#dateTime> ;

:businessStopTime ”2013-07-01T00:00:00"^^<xsd:#dateTime> ;

:transactionTime "2012-11-14T15:10:22"^^<xsd:#dateTime> .

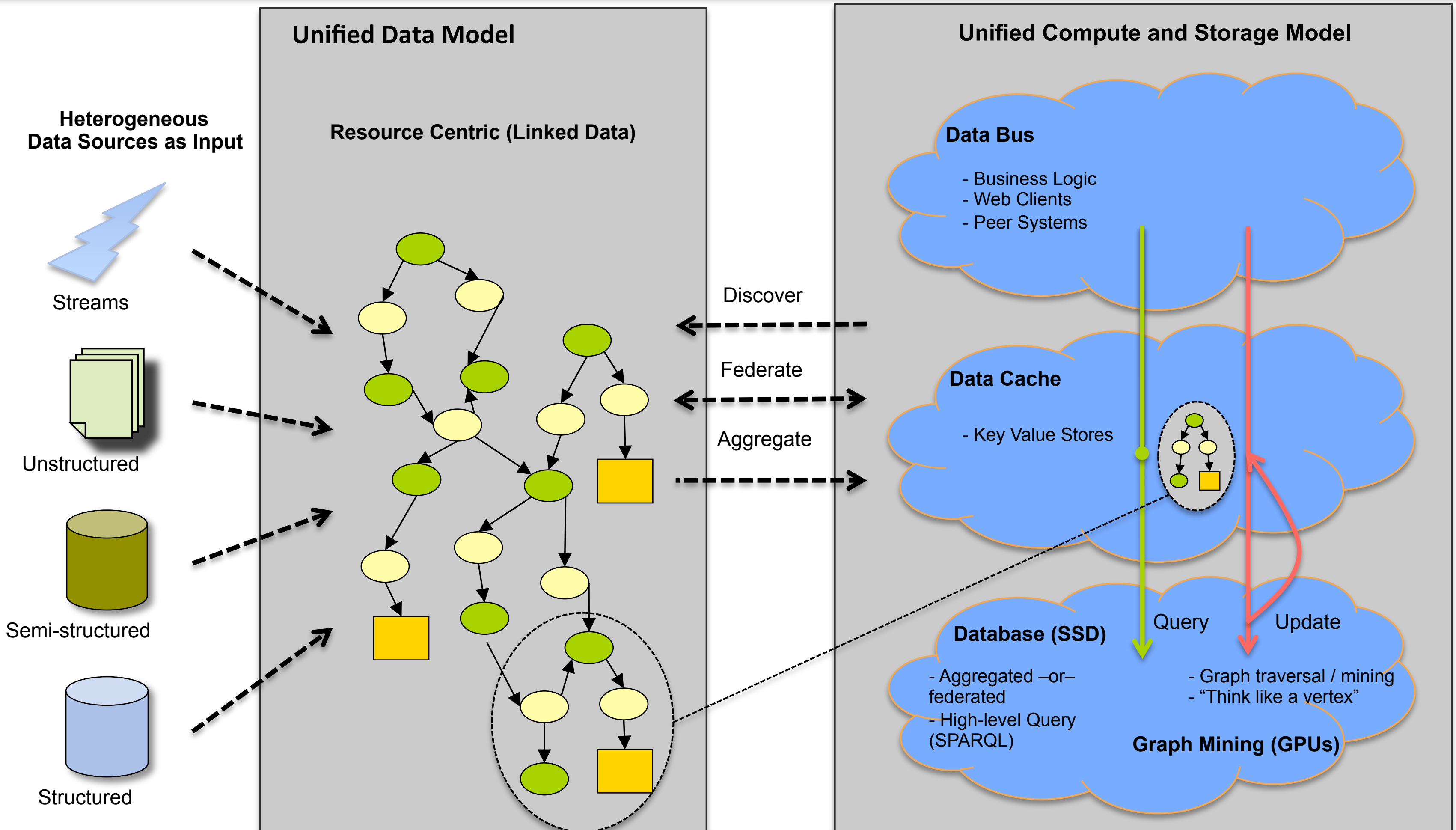
Bi-Temporal Data

Subject	Predicate	Object	transTime	startTime	stopTime
:widget1	:hasColor	:red	2012Q3	2013Q1	2013Q2
:widget1	:hasColor	:green	2012Q3	2013Q2	2013Q3

Primary Key Valid Time

- One physical schema for bi-temporal data.
- Use case is not covered by key-value stores.
  - Can we declaratively configure SPARQL DBs to handle this?
- Link attributes are *very useful*.

Unifying Architecture (example)



bigdata ®

Bryan Thompson
SYSTAP, LLC
bryan@systap.com

http://www.systap.com/bigdata.htm http://www.bigdata.com/blog