

In-Database Analytics for NoSQL Key-Value Stores

Dylan Hutchison, 5 December 2016

Advisors: Bill Howe, Dan Suciu

Tons of NoSQL Key-Value Stores!



levelDB



Microsoft Azure
DocumentDB

ORACLE®

BERKELEY DB

APACHE
HBASE





Who uses a K-V DB? How?

Basho Riak Use Cases

➔ Find a "natural key" for your application

| DATA TYPE | KEY | VALUE |
|-------------|--------------------|--|
| Session | User/Session ID | Session Data |
| Content | ID | Documents, Images, Posts, Videos, Texts, JSON/HTML, etc. |
| Advertising | Campaign ID | Ad Content |
| Logs | Date | Log File |
| Sensor | Date/Time | Sensor Updates |
| User Data | Login, Email, UUID | User Attributes |



Who uses a K-V DB? How?

Timely – Time Series Event Monitoring

| Row | ColumnFamily | ColumnQualifier | Value |
|----------------------------|--------------|-------------------------|-------|
| sys.cpu.user\1447879348291 | host=r001n01 | instance=0,rack=r001 | 2.0 |
| sys.cpu.user\1447879348291 | instance=0 | host=r001n01,rack=r001 | 2.0 |
| sys.cpu.user\1447879348291 | rack=r001 | host=r001n01,instance=0 | 2.0 |

- Compound key
- Intentional order
- Multiple values per key
- Repeated values

Many tricks to encode information into K-Vs

Who uses a K-V DB? How?

GeoMesa – Spatiotemporal Store

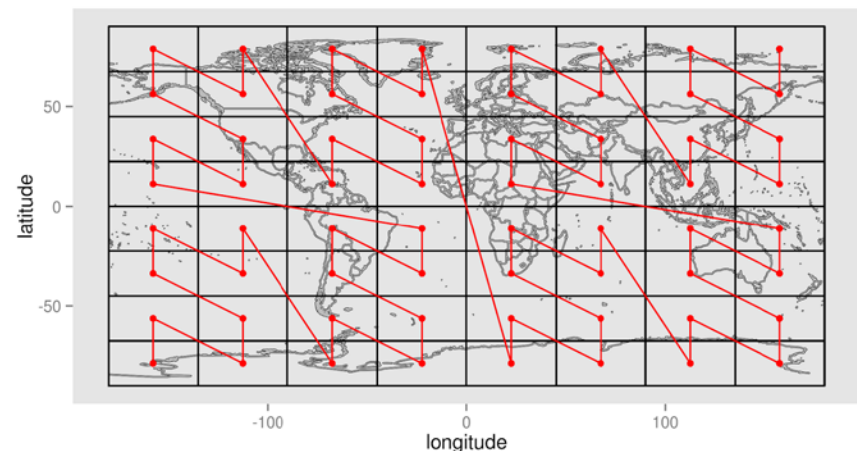
K-V encoding tricks
can be arbitrarily complex

| KEY | | | | | | | VALUE |
|------------------------------|------------------------------------|--------------------------|---------------|------------------|-----------|---------------|--------------------------------------|
| ROW | | | COLUMN | | TIMESTAMP | VIZ | Byte-encoded SimpleFeature |
| | | | COLUMN FAMILY | COLUMN QUALIFIER | | | |
| Epoch Week <i>2 bytes</i> | Z3(x,y,t) <i>8 bytes</i> | Unique ID (such as UUID) | "F" | - | - | Security tags | |

Hierarchical
Storage

Hash
Functions

+ Secondary or Full Indexes on Values



Why use K-V stores?

- > Scale out – large commodity clusters
- > Transparent partitioning, layout, performance
 - Friendly to developers & applications that want control
- > Some have "special features"
 - entry-level access control
- > Simple solution, *if* all your application requires is to read & write entries by key



Why use K-V stores?

- > Scale out – large commodity clusters
- > Transparent partitioning, layout, performance
 - Friendly to developers & applications that want control
- > Some have "special features"
 - entry-level access control
- > Simple solution, *if* all your application requires is to read & write entries by key

What if you want to do more?

Computing beyond read-writes

**Computing @application client
can have drawbacks...**



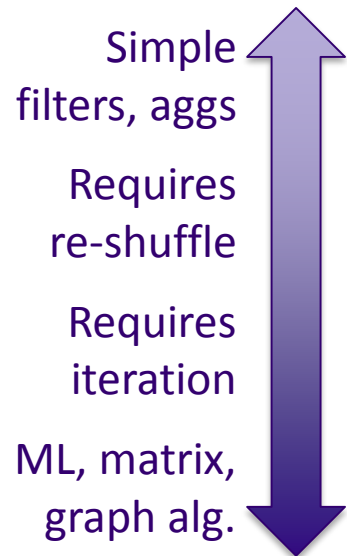


Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

| DATA TYPE | KEY | VALUE |
|-----------|-----------------|--|
| Session | User/Session ID | Session Data (pages visited, shopping cart, ...) |

- > **Fast read-writes perfect for low-latency web server**
- > **What if the website managers want to run analytics?**





Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

| DATA TYPE | KEY | VALUE |
|-----------|-----------------|--|
| Session | User/Session ID | Session Data (pages visited, shopping cart, ...) |

- > **Fast read-writes perfect for low-latency web server**
- > **What if the website managers want to run analytics?**

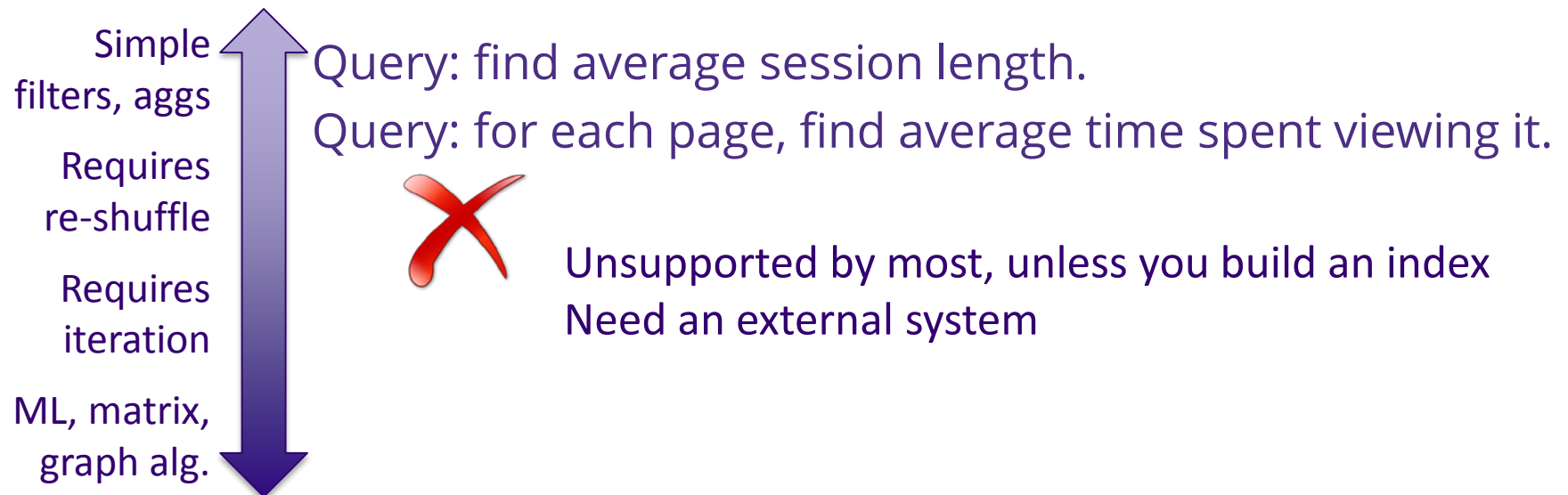


Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

| DATA TYPE | KEY | VALUE |
|-----------|-----------------|--|
| Session | User/Session ID | Session Data (pages visited, shopping cart, ...) |

- > **Fast read-writes perfect for low-latency web server**
- > **What if the website managers want to run analytics?**



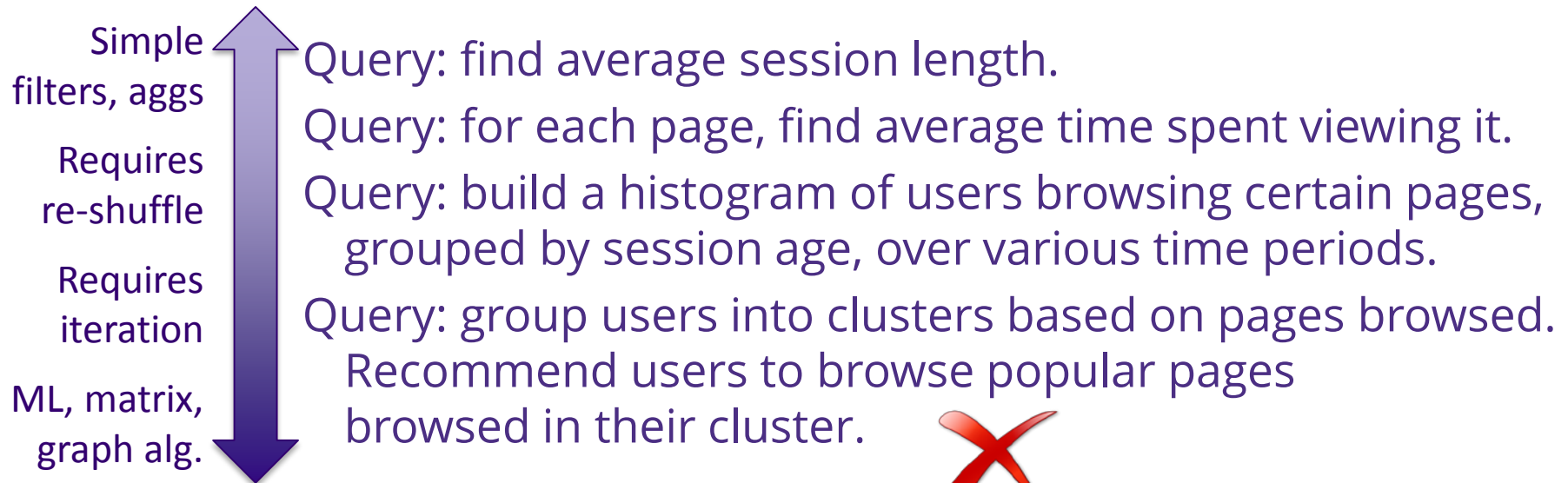


Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

| DATA TYPE | KEY | VALUE |
|-----------|-----------------|--|
| Session | User/Session ID | Session Data (pages visited, shopping cart, ...) |

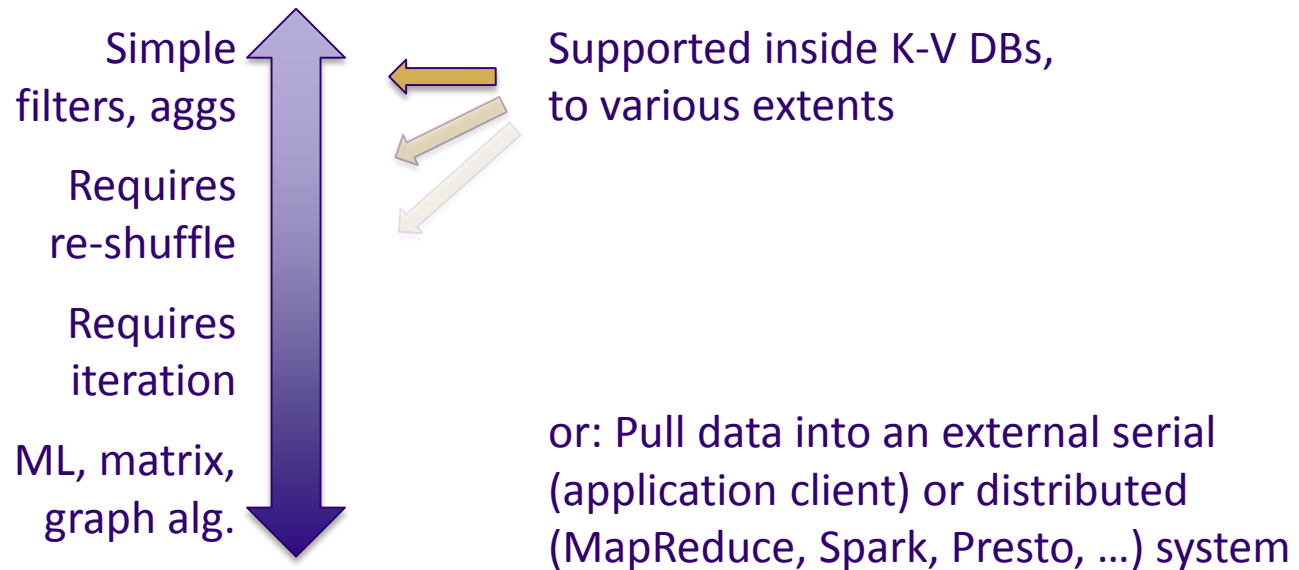
- > **Fast read-writes perfect for low-latency web server**
- > **What if the website managers want to run analytics?**





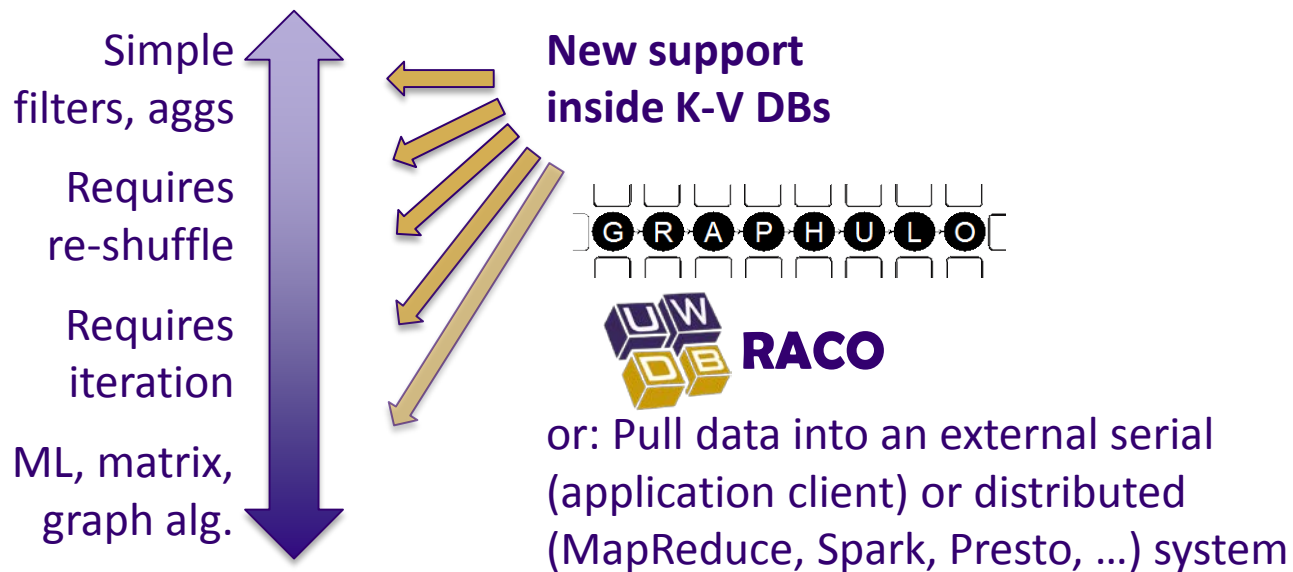
Computing on Key-Value DB

State of the Art



Computing on Key-Value DB

This Talk



- > How to implement analytics in K-V DBs?
- > When is it good to do so?
 - Consider: # of re-shuffles, access path, selectivity, ...



Computing on Key-Value DB

Benefits of Server-Side Computation

1. Data Locality

- Save communication

2. Reuse infrastructure

- One less system to adopt and maintain

3. Database features for free

- Indexed access
- Distributed execution




Graphulo: Linear Algebra for Apache Accumulo

or, conceptually, any
Google BigTable-based K-V DB





GraphBLAS: Working Spec for LA

| | GraphBLAS Kernel |
|---|---------------------------|
| | BuildMatrix (\oplus) |
| | ExtractTuples |
|  | MxM (\oplus, \otimes) |
| | EwiseMult (\otimes) |
| | EwiseAdd (\oplus) |
| | Extract |
| | Apply (f) |
| | Assign |
| | Reduce (\oplus) |
| | Transpose |

for *Sparse* Matrices

Matrix as K-Vs

| Key | | | | Value | |
|-----|--------|-----------|------------|-------|-----------|
| Row | Column | | | | Timestamp |
| | Family | Qualifier | Visibility | | |

- > Adjacency List / Adjacency Matrix
- > (Row, Column Qualifier, Value)
= (v₁, v₂, weight)
[Transpose: (v₂, v₁, weight)]

$$\begin{array}{c}
 1 \quad 10 \quad \dots \\
 1 \begin{bmatrix} 141 & 12 & \dots \\ 18 & \vdots & \\ \vdots & \vdots & \ddots \end{bmatrix} \\
 10 \\
 \vdots
 \end{array}$$

| Example K-Vs | | | |
|--------------|---------|-------|-----|
| row | :colq | ->val | |
| 1 | :1 [] | -> | 141 |
| 1 | :10 [] | -> | 12 |
| 1 | :101 [] | -> | 9 |
| 1 | :105 [] | -> | 3 |
| 1 | :11 [] | -> | 9 |
| 1 | :110 [] | -> | 3 |
| 10 | :1 [] | -> | 18 |
| 10 | :109 [] | -> | 2 |
| 15 | :1 [] | -> | 18 |
| 15 | :109 [] | -> | 2 |

Other schemas supported:

- Edge List / Incidence Matrix
- Single Table with transpose & degrees



Matrix as K-Vs

| Key | | | | Value | |
|-----|--------|-----------|------------|-------|-----------|
| Row | Column | | | | Timestamp |
| | Family | Qualifier | Visibility | | |

- > Adjacency List view of Matrix
- > (Row, Column Qualifier, Value)
= (v₁, v₂, weight)
[Transpose: (v₂, v₁, weight)]

$$\begin{array}{c}
 1 \quad 10 \quad \dots \\
 1 \begin{bmatrix} 141 & 12 & \dots \end{bmatrix} \\
 10 \begin{bmatrix} 18 & & \vdots \end{bmatrix} \\
 \vdots \begin{bmatrix} \vdots & \dots & \ddots \end{bmatrix}
 \end{array}$$

Other schemas supported:

- Edge List / Incidence Matrix
- Single Table with transpose & degrees

| | | Example K-Vs | | |
|---------------------|----------|-----------------|---------|--------|
| | | row :colq ->val | | |
| Partition by Row | Tablet 1 | 1 | :1 [] | -> 141 |
| | | 1 | :10 [] | -> 12 |
| | | 1 | :101 [] | -> 9 |
| | | 1 | :105 [] | -> 3 |
| | | 1 | :11 [] | -> 9 |
| | | 1 | :110 [] | -> 3 |
| | | 10 | :1 [] | -> 18 |
| | | 10 | :109 [] | -> 2 |
| | | | | |
| Tablet 2 | 15 | :1 [] | -> 18 | |
| | 15 | :109 [] | -> 2 | |



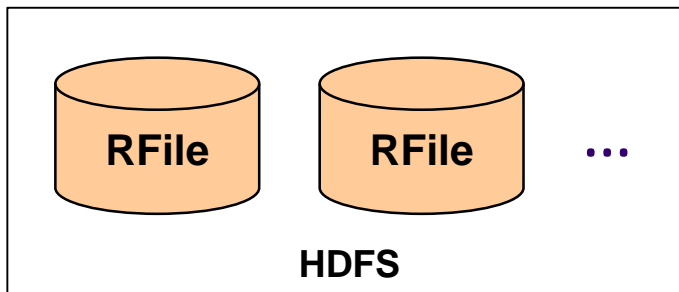


Accumulo Scan Iterator Pipeline

Goal: Understand Accumulo's support for in-database computation in order to re-purpose it for analytics

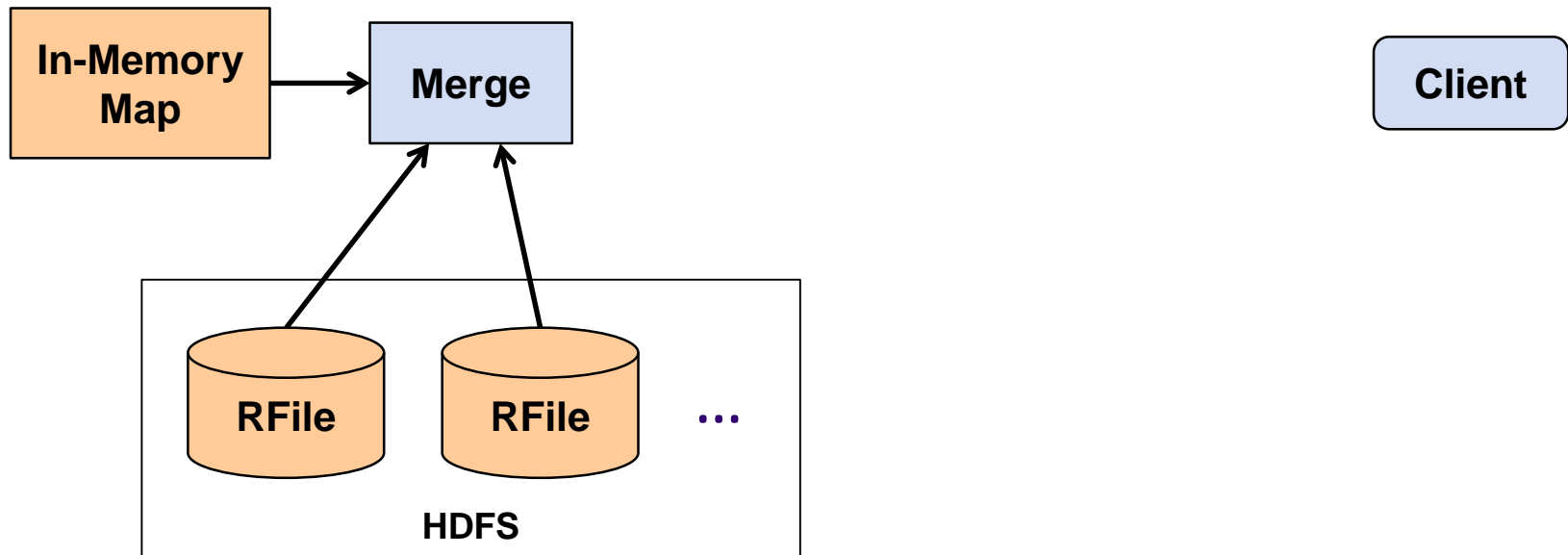
In-Memory Map

Client

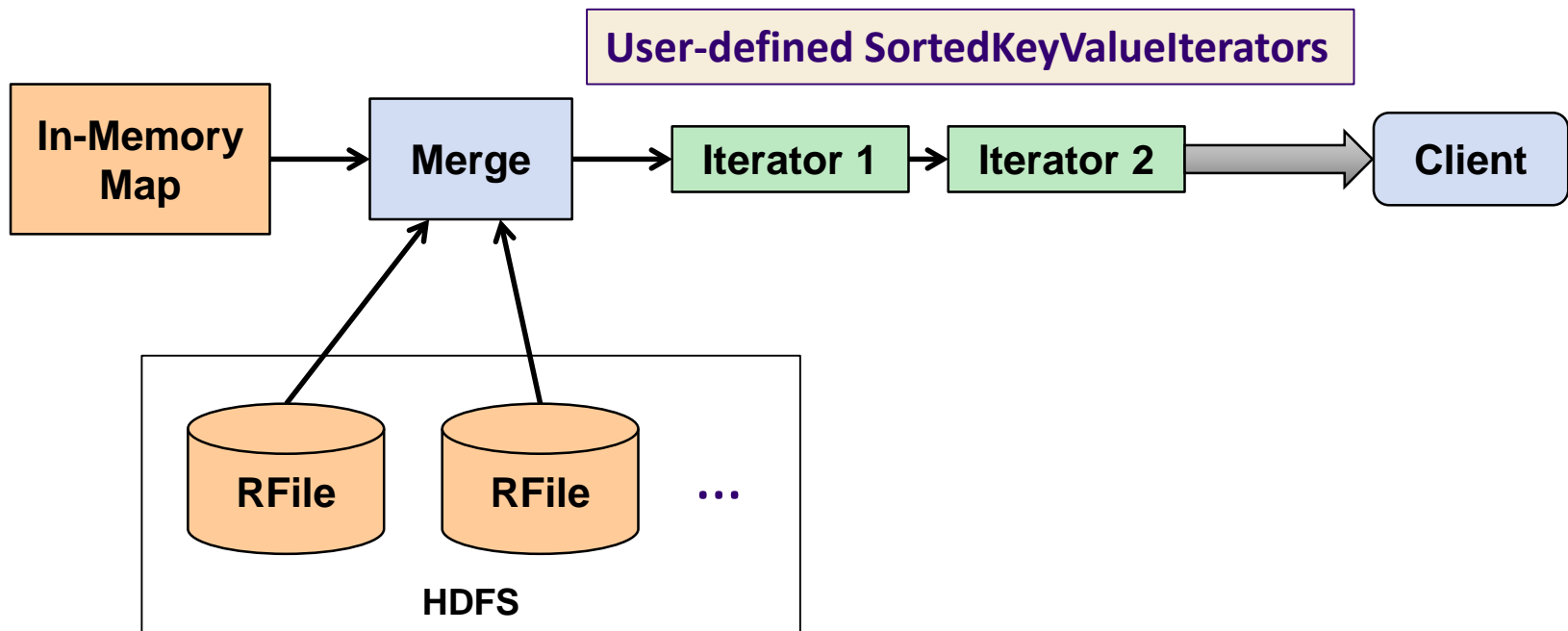




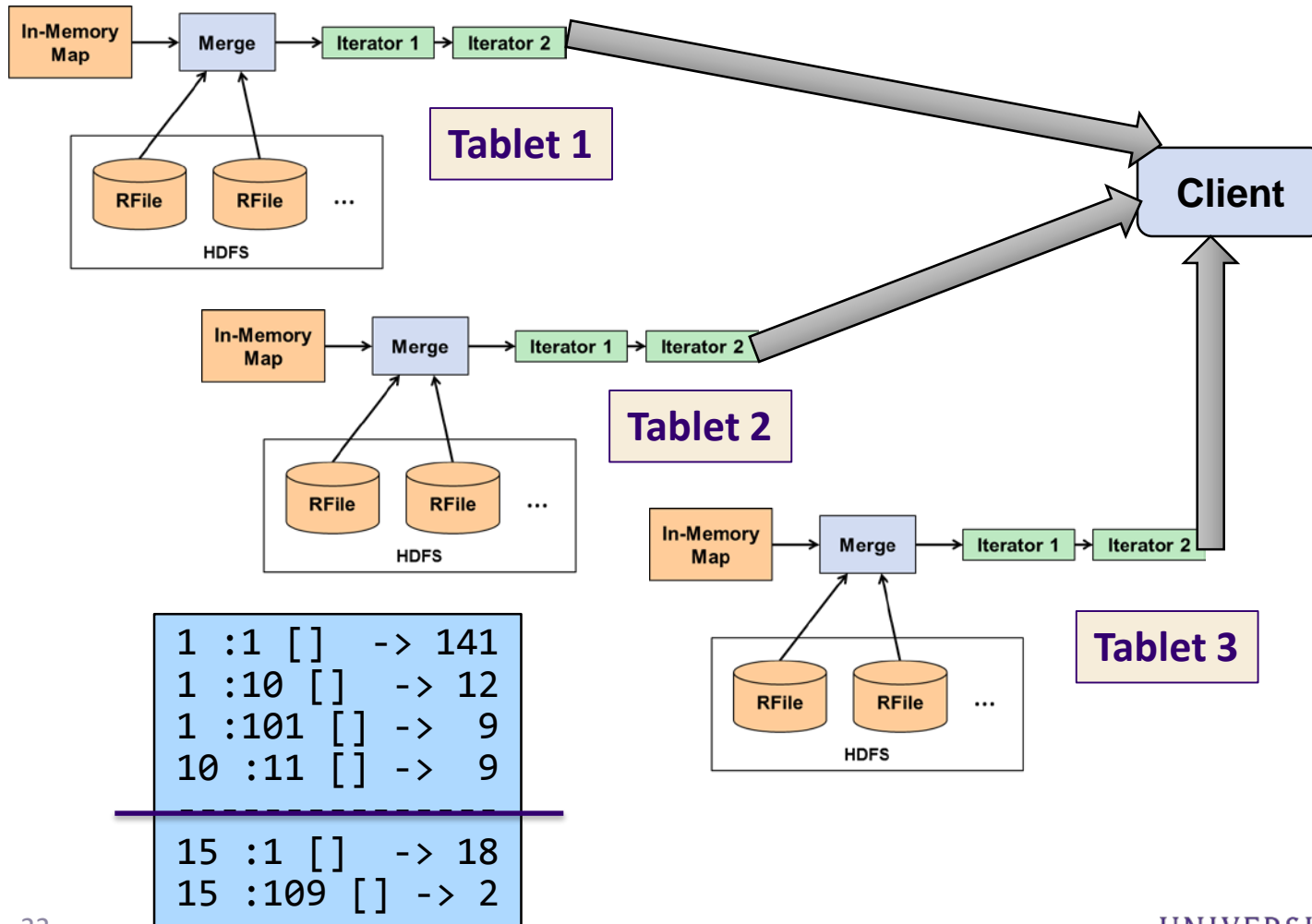
Accumulo Scan Iterator Pipeline



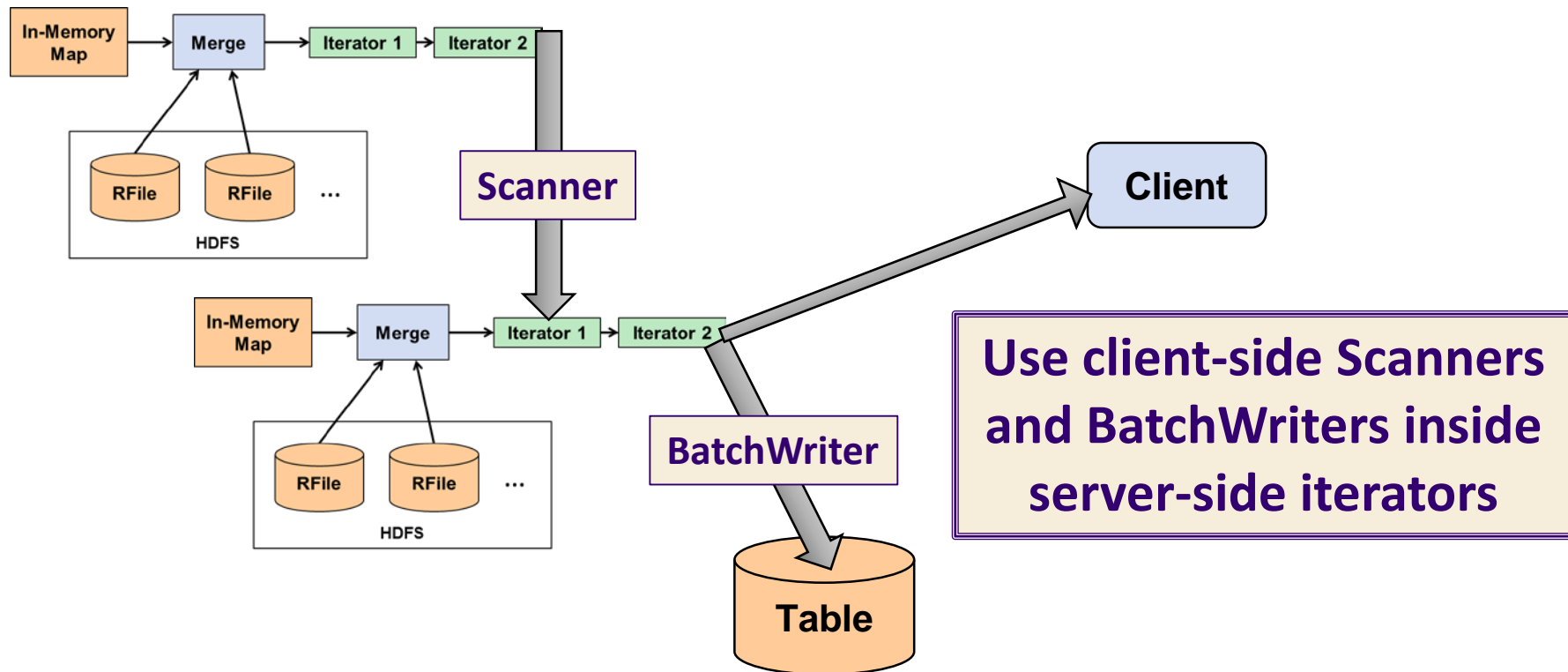
Accumulo Scan Iterator Pipeline



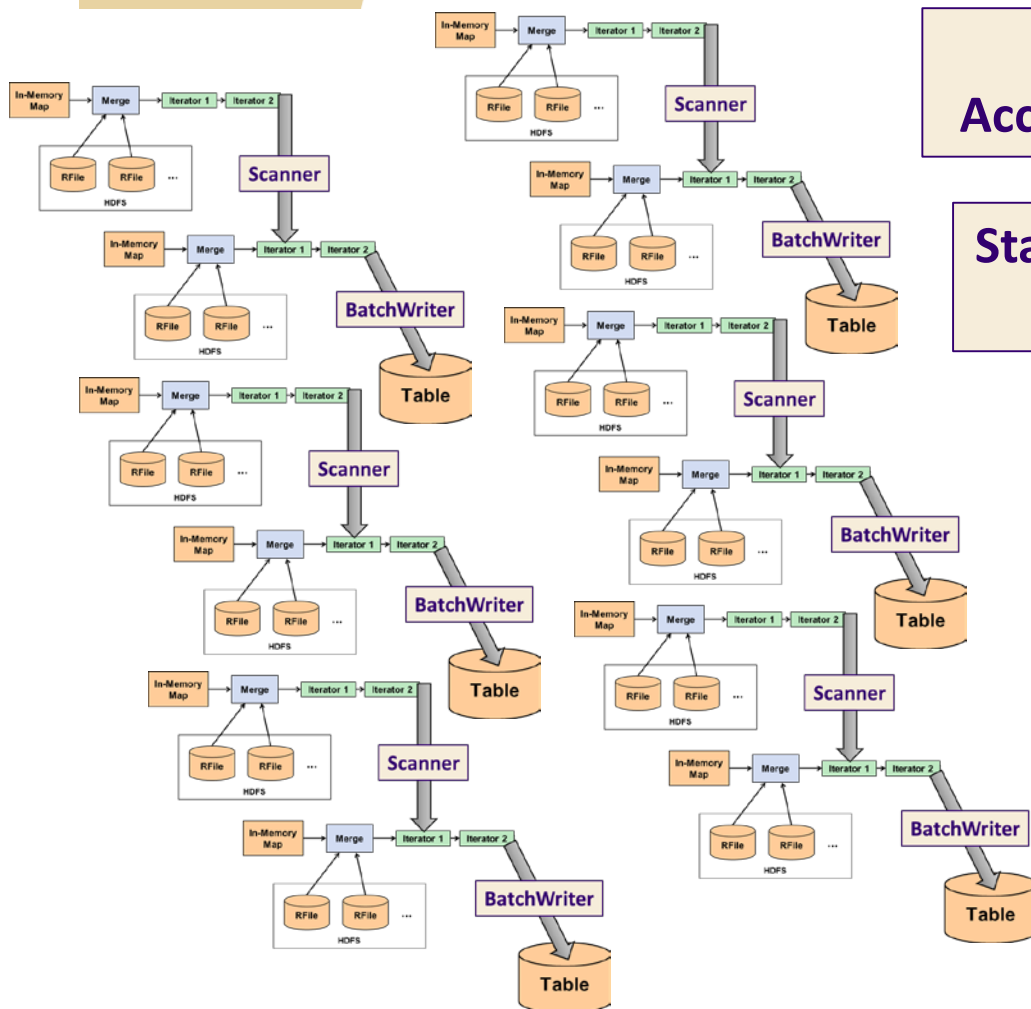
Accumulo BatchScan Iterator Pipeline



Graphulo addition to Iterator Pipeline



Graphulo addition to Iterator Pipeline

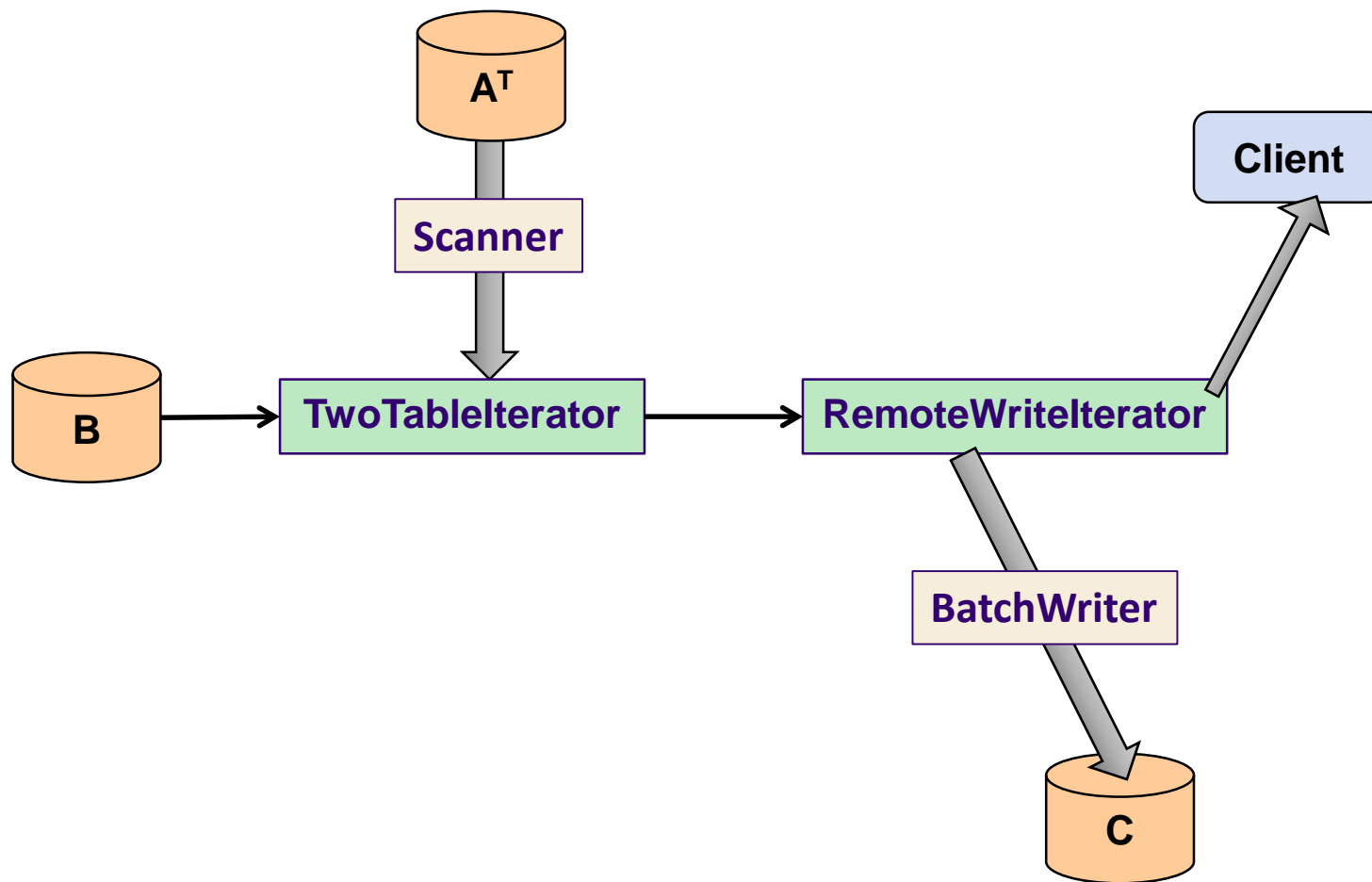


Distributes with
Accumulo's Tablet Servers

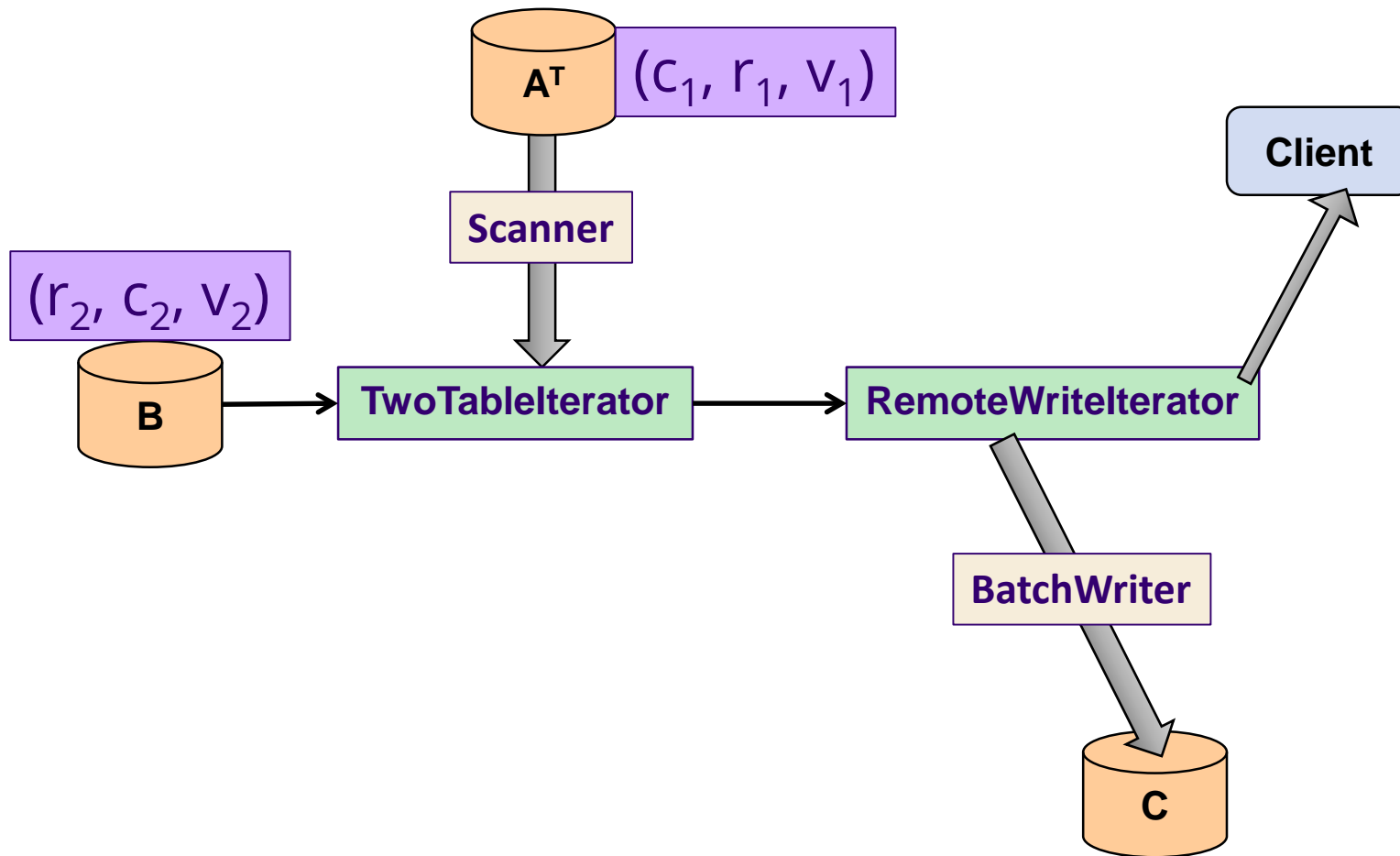
Standing tablet server thread pools
service queries interactively

Use client-side Scanners
and BatchWriters inside
server-side iterators

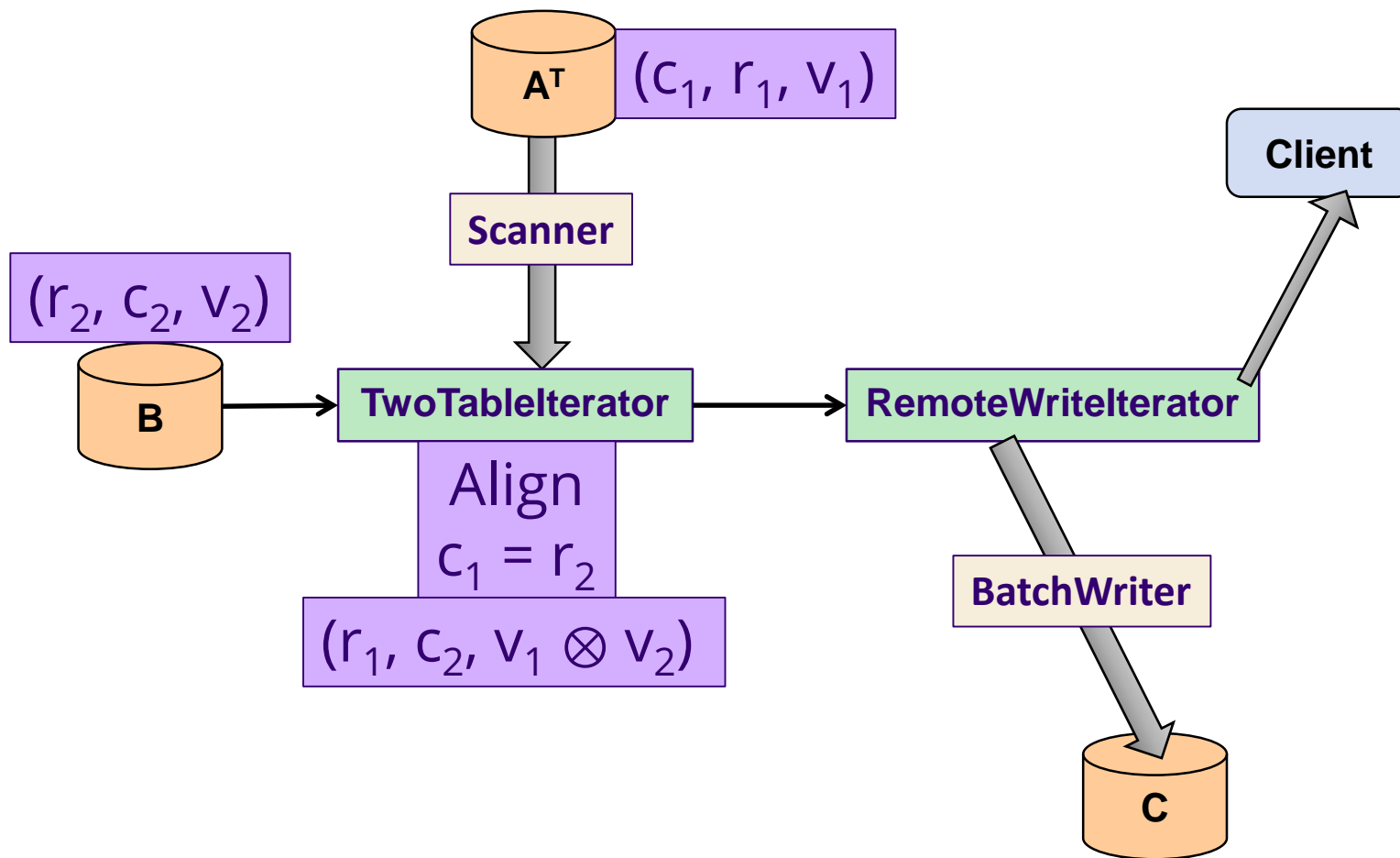
Graphulo MxM: $A^T (\oplus, \otimes) B$



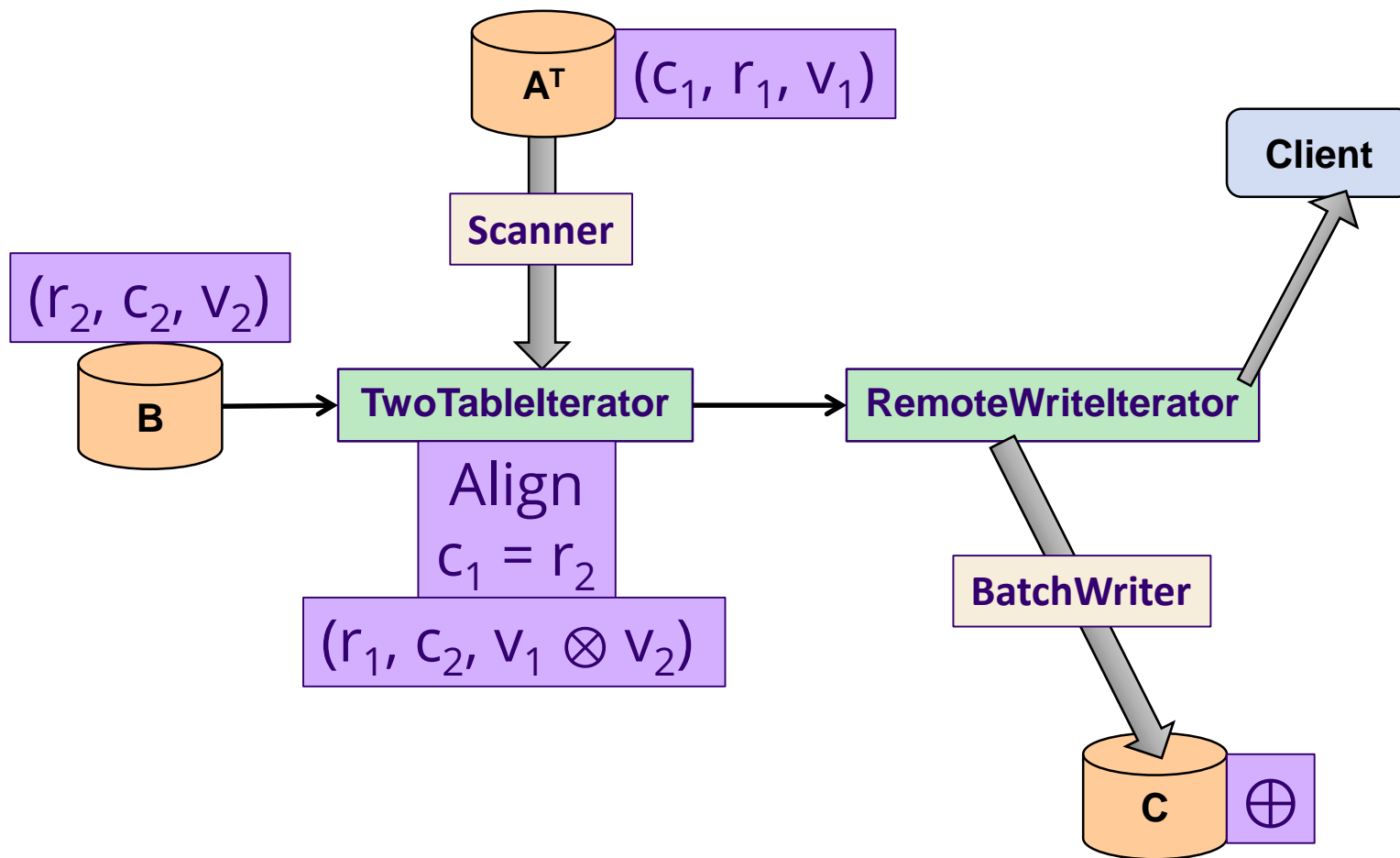
Graphulo MxM: $A^T (\oplus.\otimes) B$



Graphulo MxM: $A^T (\oplus.\otimes) B$



Graphulo MxM: $A^T (\oplus.\otimes) B$





Graphulo Client Functions

```
long TableMult(String Atable, String Btable, String Ctable)
```

```
long SpEwiseX(String Atable, String Btable, String Ctable)
```

```
long SpEwiseSum(String Atable, String Btable, String Ctable)
```

```
...
```

**Simple API abstracts
the iterator pipeline**

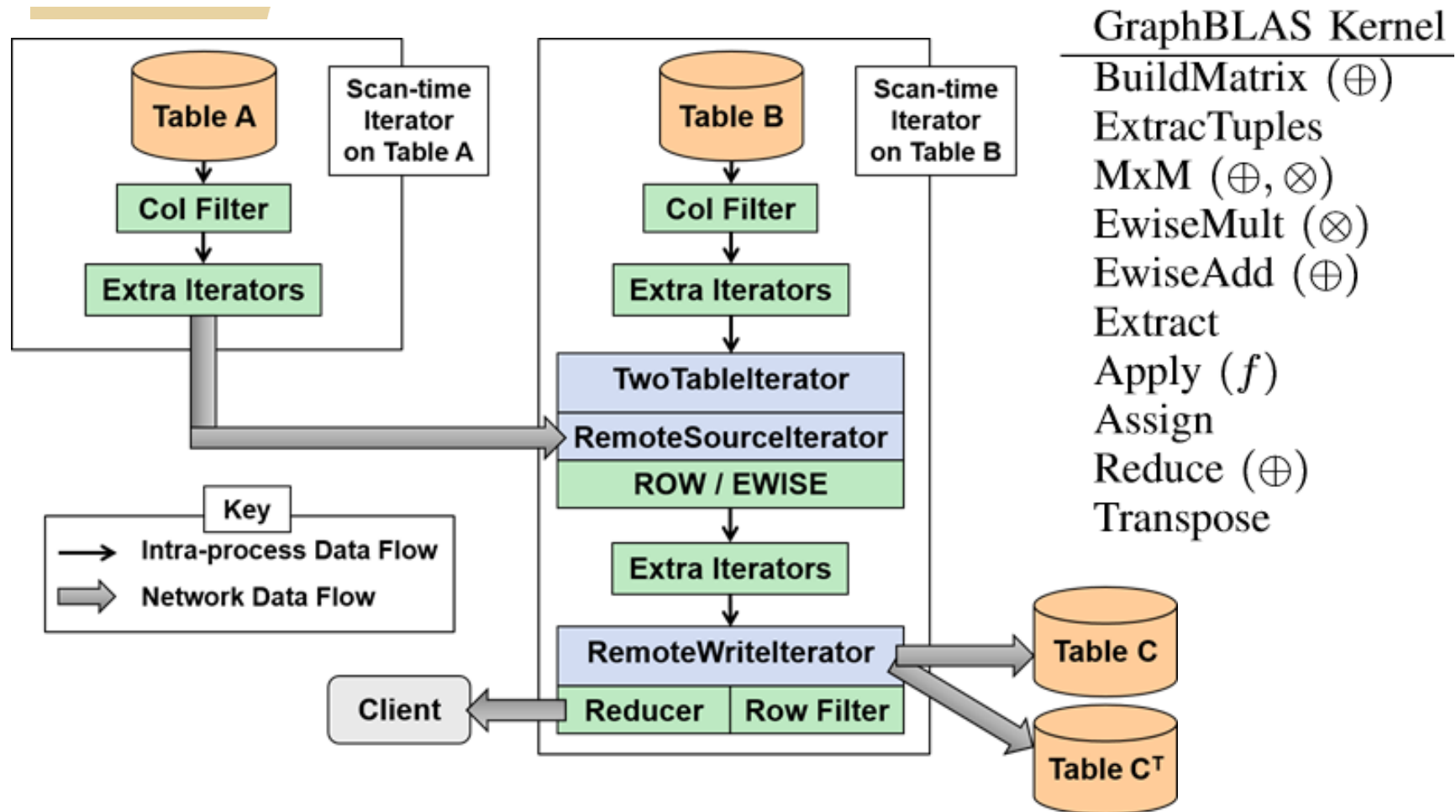


Graphulo Client Functions

```
long TwoTable(  
    String ATtable, String Btable, String Ctable, String CTtable,  
    int BScanIteratorPriority, TwoTableIterator.DOTMODE dotmode,  
    Map<String, String> optsTT, IteratorSetting plusOp,  
    String rowFilter, String colFilterAT, String colFilterB,  
    boolean emitNoMatchA, boolean emitNoMatchB,  
    List<IteratorSetting> iteratorsBeforeA,  
    List<IteratorSetting> iteratorsBeforeB,  
    List<IteratorSetting> iteratorsAfterTwoTable,  
    Reducer reducer, Map<String, String> reducerOpts,  
    int numEntriesCheckpoint, Authorizations ATauthorizations,  
    Authorizations Bauthorizations, int batchWriterThreads  
)
```

**Full control
when you need it**

Graphulo's TwoTable Pipeline



Evaluating Graphulo

When is it better to use Graphulo vs. an external system?



Past Experiments

Compare Graphulo to

- > **Single-node in-memory matrix libraries**
 - D4M Sparse matrix library (MATLAB)
 - MTJ Dense matrix library (Java)
- > **Itself**
 - Benchmark of Graphulo scalability with cluster size

Graphulo Implementation of Server-Side Sparse Matrix Multiply in the Accumulo Database

Dylan Hutchison,^{†§*} Jeremy Kepner,^{†‡◇*} Vijay Gadepally,^{†‡*} Adam Fuchs⁺

[†]MIT Lincoln Laboratory, [§]University of Washington,

[‡]MIT Computer Science & AI Laboratory, [◇]MIT Mathematics Department, ⁺Sqrrl, Inc.

2015
HPEC

Abstract
distributed storage
graph data
and persistent
calculations

The GraphBLAS standard provides a compact and efficient basis for a wide range of graph applications through a small number of sparse matrix operators. Graphulo is a server-side implementation of GraphBLAS that leverages Accumulo. We compare the matrix-vector product implementation achieved in Accumulo to the peak write rate. We use Graphulo library that graph analytics within

Graphulo faster than single-node in-memory external systems on I/O-bound, single-pass computation

This
LAS
atics

In this paper we focus on Sparse Generalized Matrix Multiply (SpGEMM), the core kernel at the heart of GraphBLAS.

used in terms of
addition func-
wide range of
[5] and many

1 in Accumulo
accumulo tables.
sparse matrices.

- > **Task: Matrix Multiply (MxM)**
- > **Graphulo vs. D4M, Single-node**
- > **Result: Graphulo universally faster**
- > **Why?**
 - > **I/O costs dominate (locality is good)**
 - > **MxM can be formulated as a single-pass algorithm (use outer product)**

From NoSQL Accumulo to NewSQL Graphulo: Design and Utility of Graph Algorithms inside a BigTable Database

2016
HPEC

Dylan Hutchison[†] Jeremy Kepner^{‡§◇} Vijay Gadepally^{‡§} Bill Howe[†]

[†]University of Washington [‡]MIT Lincoln Laboratory

[§]MIT Computer Science & AI Laboratory [◇]MIT Mathematics Department

Best Student
Paper

Results hold for more complex graph algorithms

Abstract—
key-value store
Recently the
workloads that demand distributed computation local to data
store
a tree
set of
article
kernel
phases
Accumulo
two
performance
results
execution
an
relationships

- > **Tasks: Find Jaccard coefficients (vertex similarity), k-Truss subgraph (community detection)**
- > **Graphulo vs. D4M vs. MTJ**
- > **Result: Graphulo universally faster at Jaccard; D4M/MTJ universally faster at k-Truss**
- > **Why?**
 - > Jaccard can be expressed as a fused MxM
 - > k-Truss iterations require one pass each
 - > Accumulo writes intermediary tables to disk

store) and
before sending
entries to a client (or a file, in the case of a compaction)
for
ed
ch,
nd
on.
ck
ent
w
ers
es
.

Benchmarking the Graphulo Processing Framework

Timothy Weale[†], Vijay Gadepally^{‡§*}, Dylan Hutchison[°] and Jeremy Kepner^{‡§+}
[†]Department of Defense, [‡]MIT Lincoln Laboratory, [§]MIT Computer Science & AI Laboratory
⁺MIT Mathematics Department, [°]University of Washington

2016
HPEC

Abstract—Graph algorithms have wide applicability to a variety of domains and are often used on massive datasets. Recent standardization efforts such as the GraphBLAS specify a set of key computational kernels that hardware and software developers can adhere to. Graphulo is a processing framework that enables GraphBLAS kernels in the Apache Accumulo database.

Graphulo multiplies the results of scaling the Graphulo engine to larger problems and scalability when a greater number of resources is used. Specifically, we present two experiments that demonstrate Graphulo scaling performance is linear with the number of available resources.

processing results on large graphs in a single experiment on a large graph. These benchmarks show that Graphulo is a framework that anyone who wish to

is provided in [12]. It is natural that a processing framework designed for databases makes use of the GraphBLAS kernels.

Graphulo [13] is a specialized graph processing framework built to work with Apache Accumulo and to conform to the GraphBLAS standard. Apache Accumulo is a NoSQL database

wide adoption in a variety of government and non-government settings.

As shown in Figure 1, Accumulo's data model has a key

Graphulo scales with Accumulo as cluster size increases

- > **Tasks: MxM, Subgraph Extraction**
- > **Graphulo Scalability Benchmark**
- > **Results: Linear weak & strong scaling on MxM. Subgraph extraction scales in interactive range (1-2k edge extraction)**
- > **Why?**
 - > **Graphulo designed to scale with Accumulo**

nn qualifier, Graphulo provides text), column number of visibility and paper.

Past experiments compare Graphulo to itself & to in-memory single-node external systems.

Where's the experiment comparing Graphulo to *distributed* external systems?

(answer: this talk)





MxM: Graphulo vs. MapReduce

- > MapReduce well supported by Accumulo
 - "Use Accumulo for low-latency queries on subgraphs"
 - "Use MapReduce for high-throughput analytics"
 - Are these assumptions true?
- > MxM experiment $C = A^T B$
 - Map phase: BatchScan & Join A^T , B on their common key
 - Reduce phase: Multiply, BatchWrite partial products
 - > # of reducers not a major factor
 - > Disable speculative execution for correctness
 - Sum partial products via iterator on C
 - Very similar to Graphulo code, except more room to optimize (earlier projection, loop fusion, fewer copies)



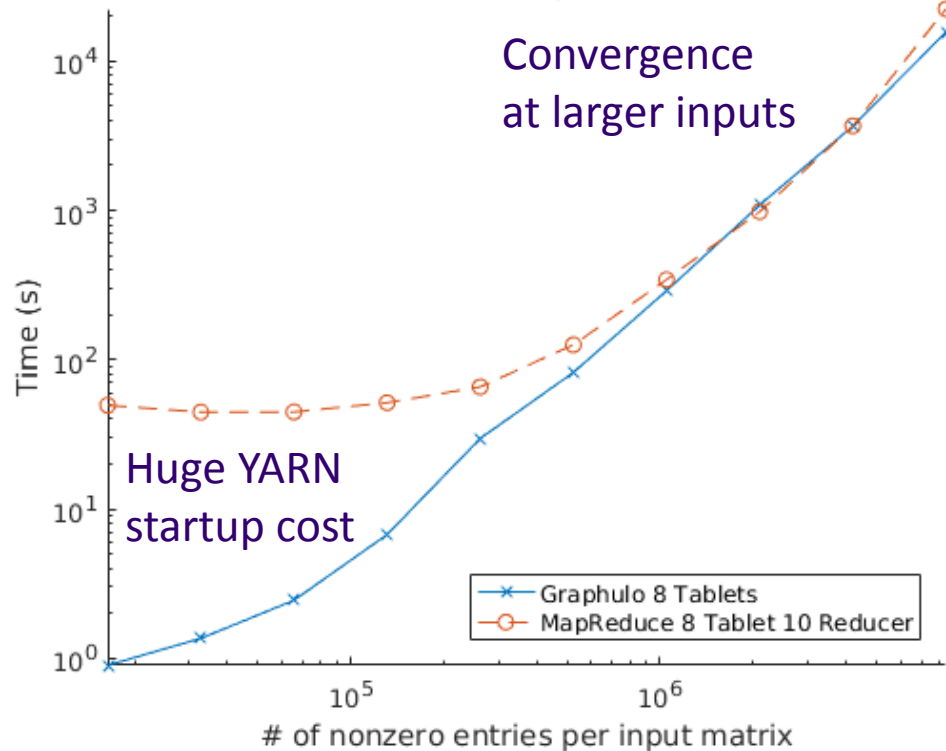
MxM: Graphulo vs. MapReduce

Hardware & Setup

- > **12 x m3.large nodes on Amazon EC2, \$38/day**
 - 7.5 GB memory, 2 vCPUs, 30 GB SSD per node
 - 8 worker tablet servers
 - > 3 GB memory for Accumulo, 3 GB for YARN
 - > One tablet per node (low intra-node parallelism)
 - 3 coordinator nodes (YARN ResourceManager, Accumulo Master, HDFS NameNode, Zookeeper)
 - 1 metric-monitoring node (Grafana with InfluxDB)
- > **Graph500 Power law matrix generator**
 - From 2^{10} to 2^{19} rows, 16 non-zeros/row, in each input
 - Skew!

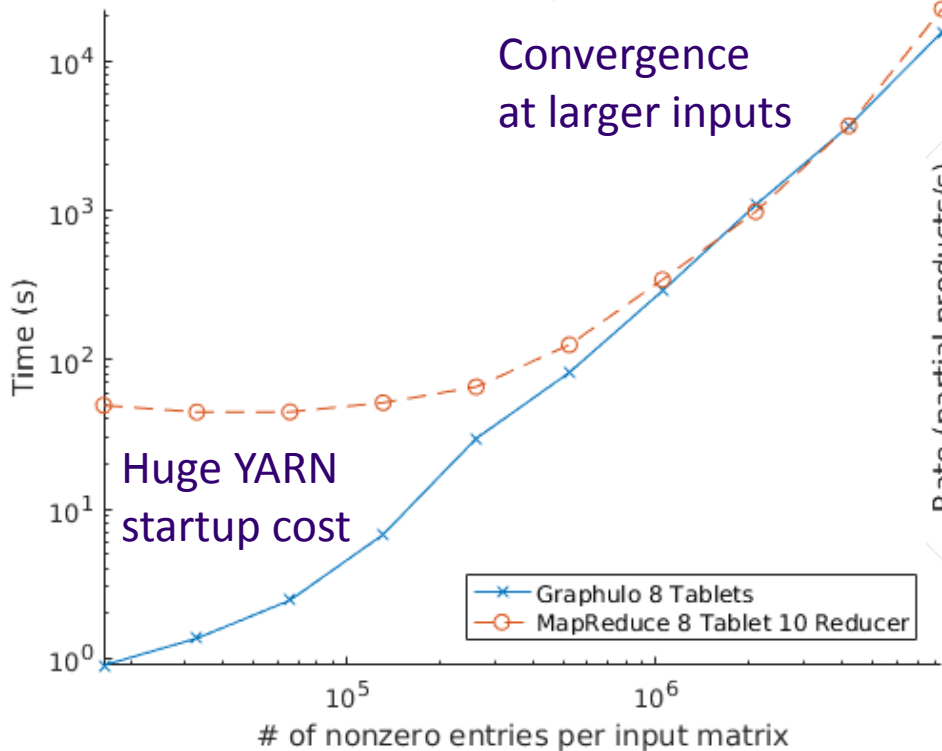
MxM: Graphulo vs. MapReduce

TableMult Runtime, Scale 10 to 19

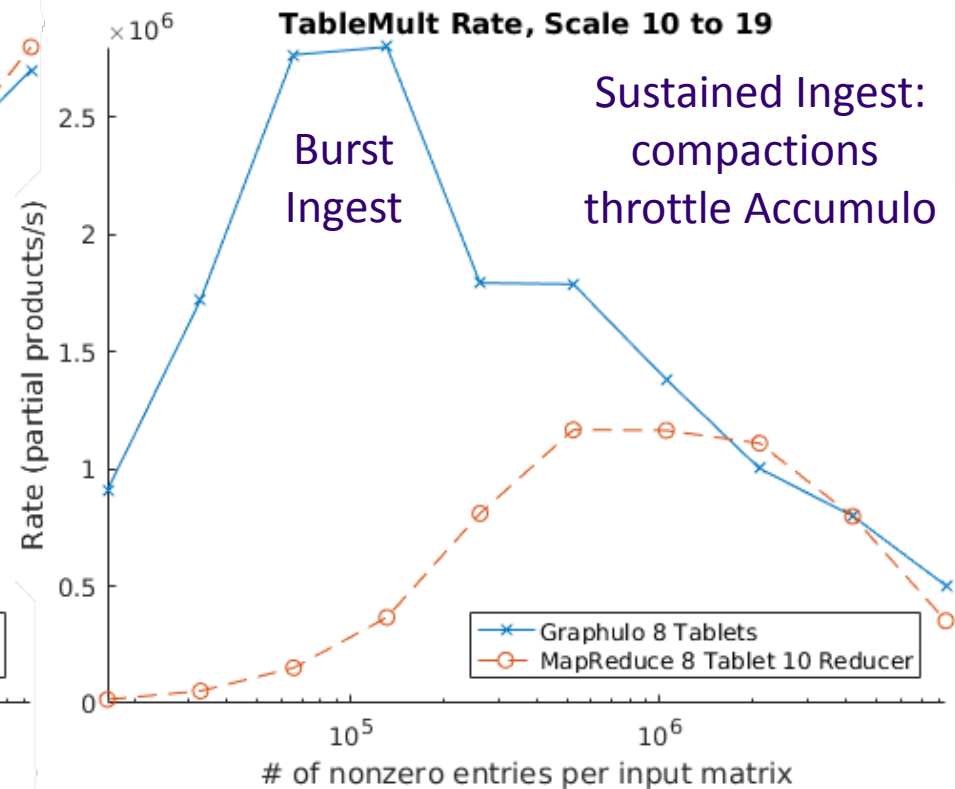


MxM: Graphulo vs. MapReduce

TableMult Runtime, Scale 10 to 19



TableMult Rate, Scale 10 to 19





When to use Graphulo?

- > Use Graphulo for I/O-bound single-pass analytics
 - Big speedup for smaller problem sizes
 - Equivalent at larger problem sizes
 - Holds against external in-memory processing (D4M, MTJ matrix libraries) & distributed (MapReduce)
- > Use an in-memory system for CPU-bound or multi-pass analytics
 - k-Truss task: expressed as a graph algorithm that loops over GraphBLAS kernels
 - Horrible to write each intermediary table to disk
 - > Yet, even HPC systems write *checkpoints*



RACO: Relational Algebra for Apache Accumulo

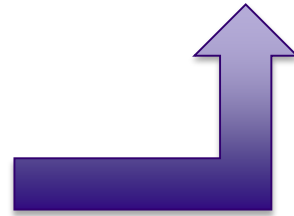
π σ \bowtie γ \cup

How to model a relation in Accumulo?

More details: [public:adhoc:netflow](#)

| Name | Type |
|-----------|-------------|
| StartTime | STRING_TYPE |
| Dur | DOUBLE_TYPE |
| Proto | STRING_TYPE |
| SrcAddr | STRING_TYPE |
| Sport | STRING_TYPE |
| Dir | STRING_TYPE |
| DstAddr | STRING_TYPE |
| Dport | STRING_TYPE |
| State | STRING_TYPE |
| sTos | LONG_TYPE |
| dTos | LONG_TYPE |
| TotPkts | LONG_TYPE |
| TotBytes | DOUBLE_TYPE |
| SrcBytes | LONG_TYPE |
| Label | STRING_TYPE |

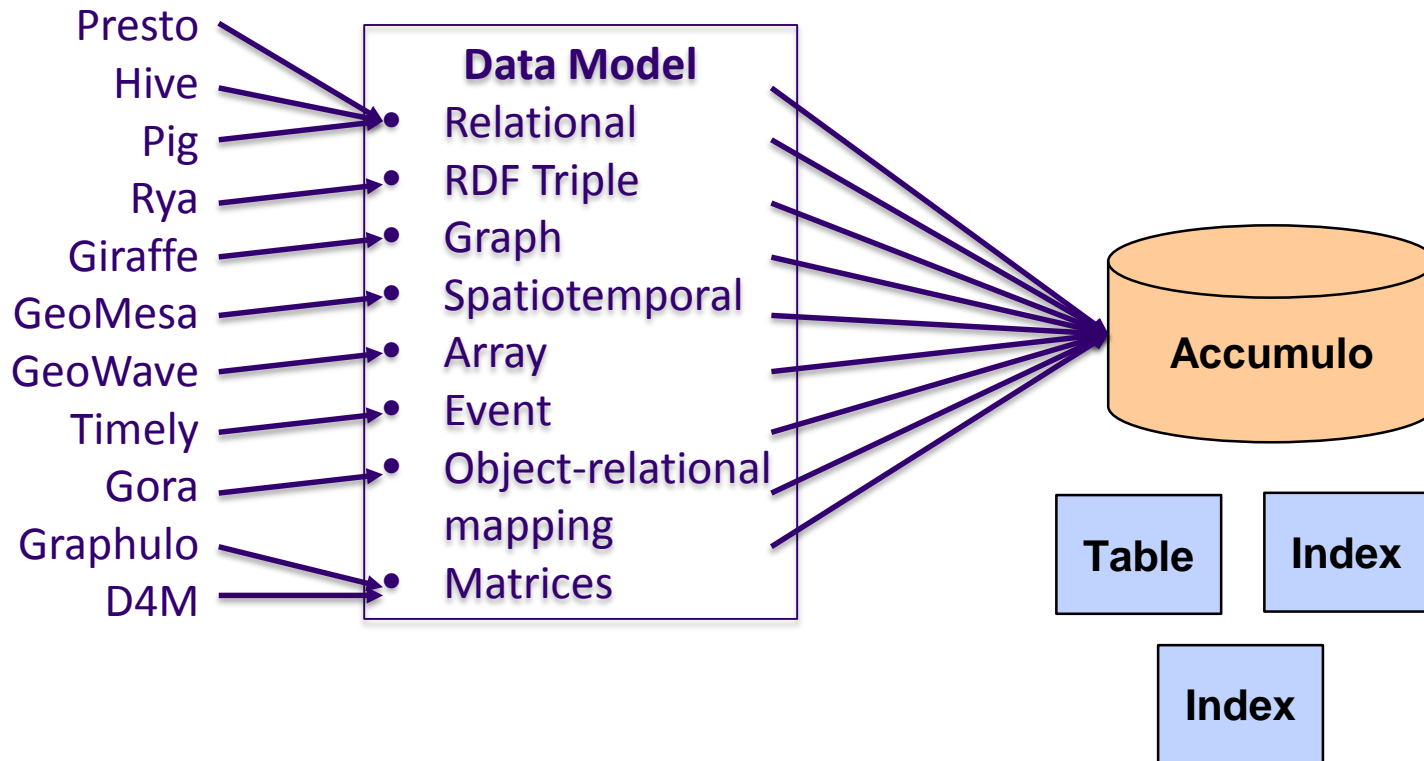
| Key | | | | | Value |
|-----|--------|-----------|------------|-----------|-------|
| Row | Column | | | Timestamp | |
| | Family | Qualifier | Visibility | | |



- > Relations have lots of attributes
- > More choices than with matrices
 - Graphulo: adjacency, incidence, single-table schemas
- > Choice affects partitioning
- > Should work for any Accumulo data



Accumulo Access Paths "in the wild"



Accumulo Access Path "Physical Schema"

For each family, divide attributes into three parts:

DAP | LAP | VAP

DAP: Distributed Access Path

- How data is partitioned across servers; null disallowed

LAP: Local Access Path

- How data is sorted within a server; null disallowed

VAP: Value Access Path

- Other attributes, unsorted; null allowed

| Key | | | | | | | Value |
|-----|--------|-----------|------------|-----|-----------|----|-------|
| Row | Column | | | | Timestamp | | |
| | Family | Qualifier | Visibility | | | | |
| DAP | FAMILY | LAP | VAP | VIS | VAP | TS | VAP |





Access Path Netflow Example

DAP | LAP | VAP

TotBytes, StartTime | | SrcAddr, DstAddr, Dur

| Key | | | | | Value |
|---------------------|--------|-----------|------------|-----------|-----------------------|
| Row | Column | | | Timestamp | |
| | Family | Qualifier | Visibility | | |
| 09:46:59.607825,276 | "" | "" | public | 145635001 | DstAddr 94.44.127.113 |
| 09:46:59.607825,276 | "" | "" | public | 145635001 | Dur 1.026539 |
| 09:46:59.607825,276 | "" | "" | public | 145635001 | SrcAddr 147.32.84.59 |

Each tuple is a set of Key-Values

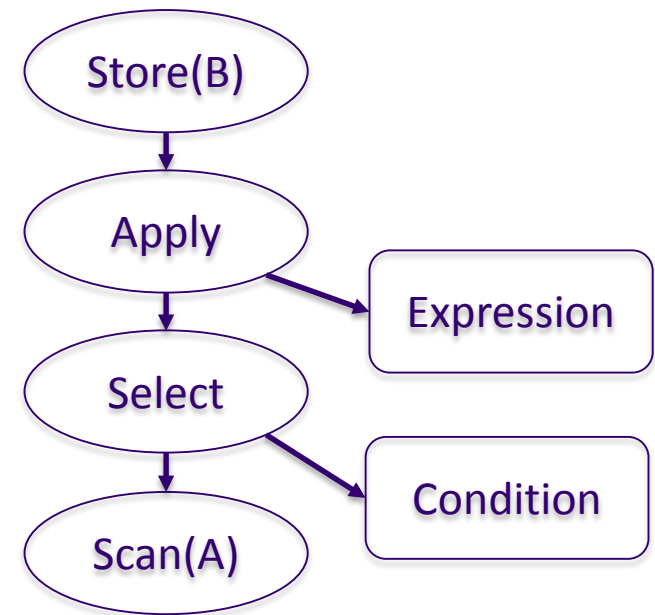
- Facilitates projection, filtering, missing values



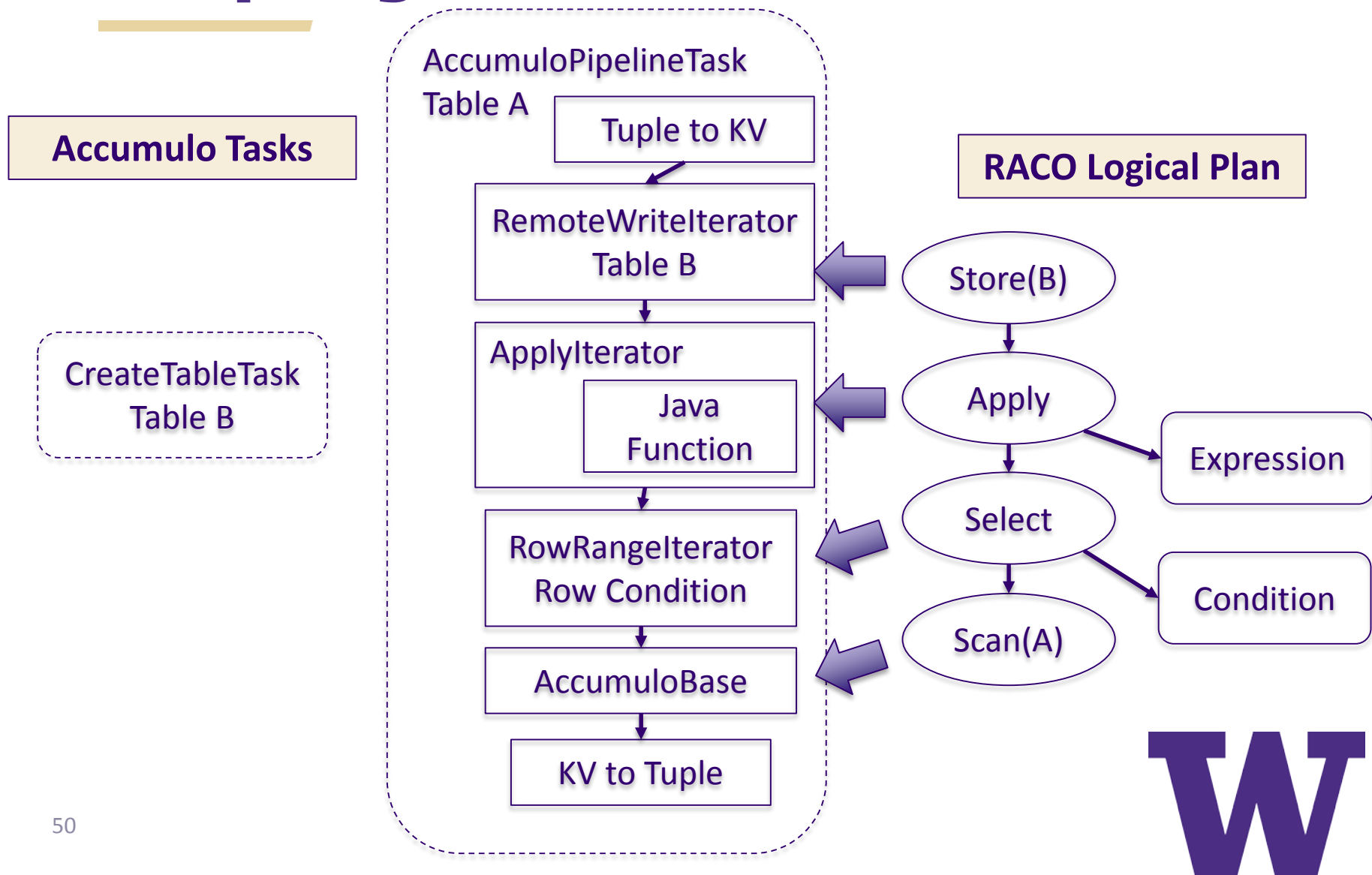
Compiling RA onto Accumulo

Accumulo Tasks

RACO Logical Plan



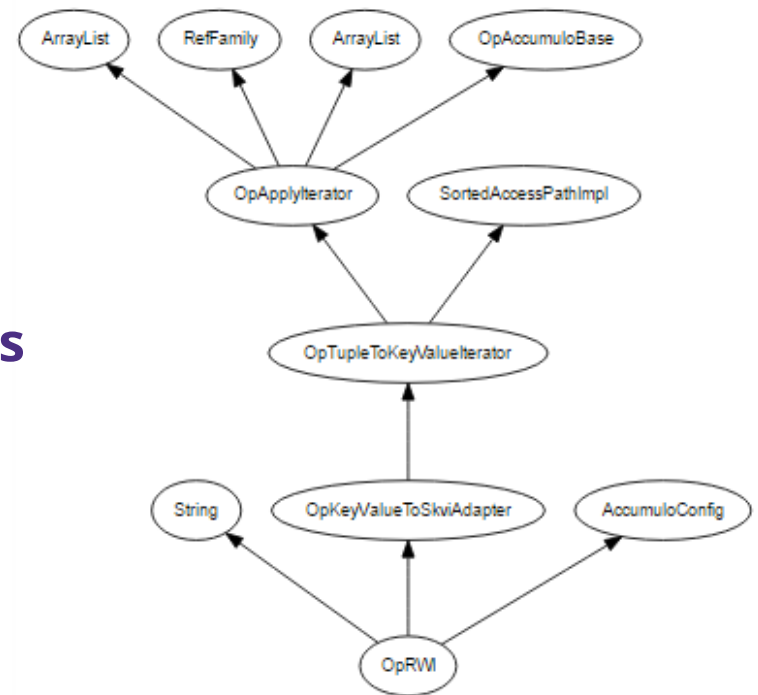
Compiling RA onto Accumulo



Takeaways from RA on Accumulo

Polystore Demo

- > **RA Compilation used in an October sponsor demo**
 - Federated plan executed part Accumulo, part Spark or C
 - Accumulo query stored results in a CSV file for Spark/C
 - "Prototype works!"
- Much more to do...





Takeaways from RA on Accumulo

Access Paths hint at a general physical schema

> Primitives

- **Block[n](...):** Store contiguous range of n entries together
- **Group(...):** Each entry stored separately; **Block[1]**
- **Range(...):** All entries stored together; **Block[N]**

> Examples

- **Accumulo:** Block(row) Group(family)
Range(row, family, qualifier, visibility, timestamp) value
- **Myria hash partitioning:** Group(a_1, \dots, a_k) a_1, \dots, a_k
- **TileDB matrix:** Block(j, i) Range(i, j) v
- **More access paths for indices**
- **Applications pick both a logical & physical schema**

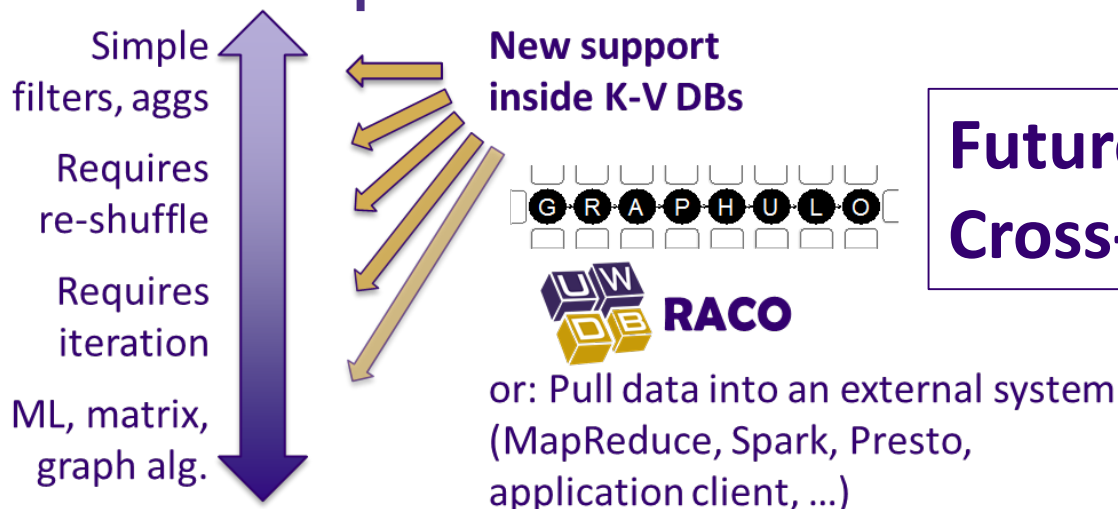
In-Database Analytics for K-V DBs

> Graphulo: LA on K-V DBs

- Graphulo vs. MapReduce experiment
 - > Comparable at scale; in-DB dominates "scale down"
- Use in-DB approach for I/O-bound single-pass analytics

> RACO: RA on K-V DBs

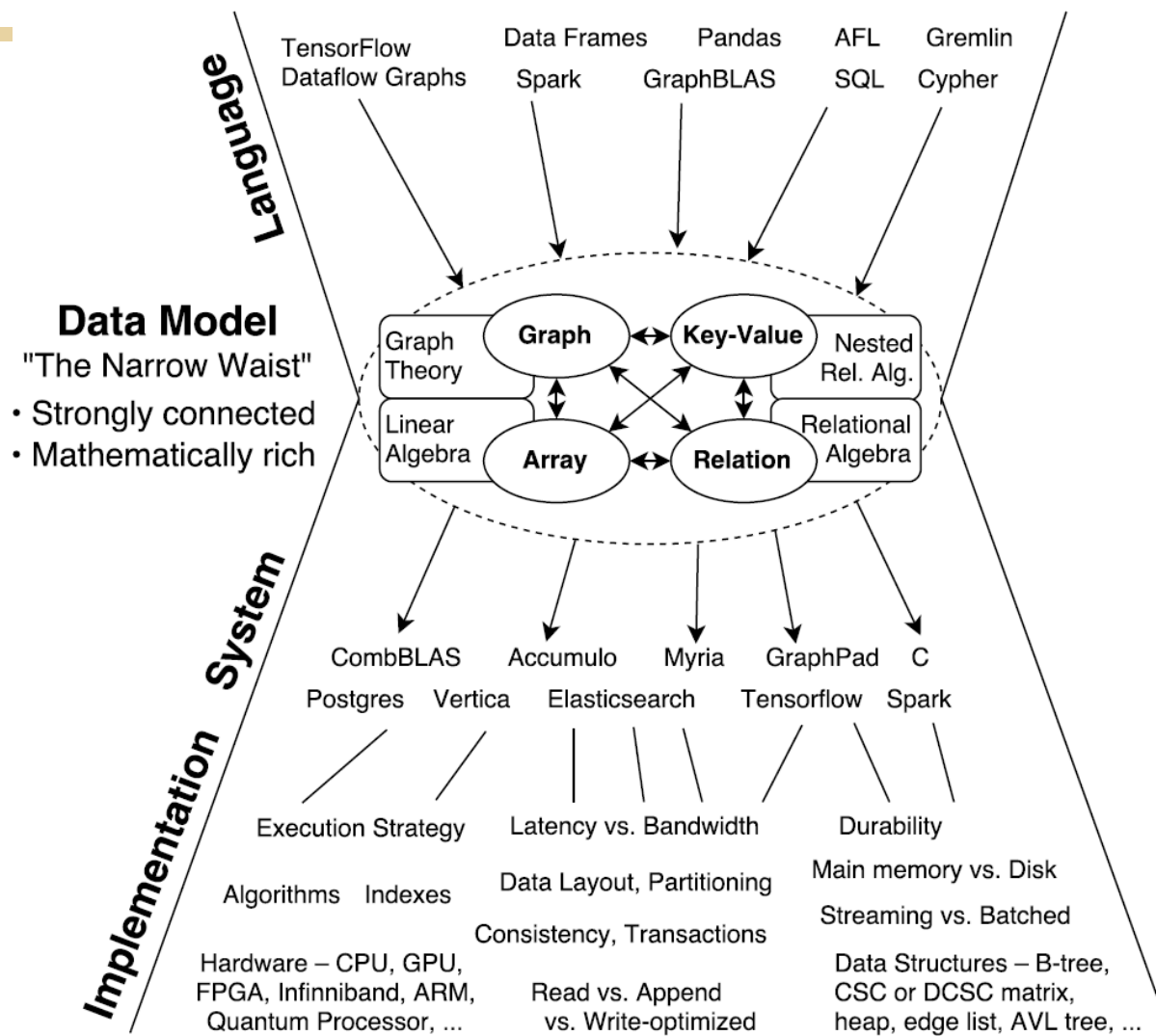
- Access Paths helpful as a "physical schema"
- Compiles into Accumulo iterators



**Future: A Polystore
Cross-Platform Optimizer**



Why does LA, RA work on K-V DBs?





Backup Slides



Background on Accumulo

| Key | | | | Value | |
|-----|--------|-----------|------------|-------|-----------|
| Row | Column | | | | Timestamp |
| | Family | Qualifier | Visibility | | |

Best for:

- > Large, de-normalized tables; no schema necessary
 - Unlimited columns; un-interpreted values; everything is a byte[]
- > TBs to PBs of data; robust horizontal scaling
- > Hadoop HDFS / Java ecosystem
- > Cell-level access control
- > Row store by default
 - Scan over rows for $O(\log n)$ lookup & sorted order
 - Maintain "Transpose Table" for column-major access path
- > Iterator processing framework





Past Experiments

Compare Graphulo to

- > **Single-node in-memory matrix libraries**
 - D4M Sparse matrix library (MATLAB)
 - MTJ Dense matrix library (Java)
- > **Itself**
 - Benchmark of Graphulo scalability with cluster size

Tasks:

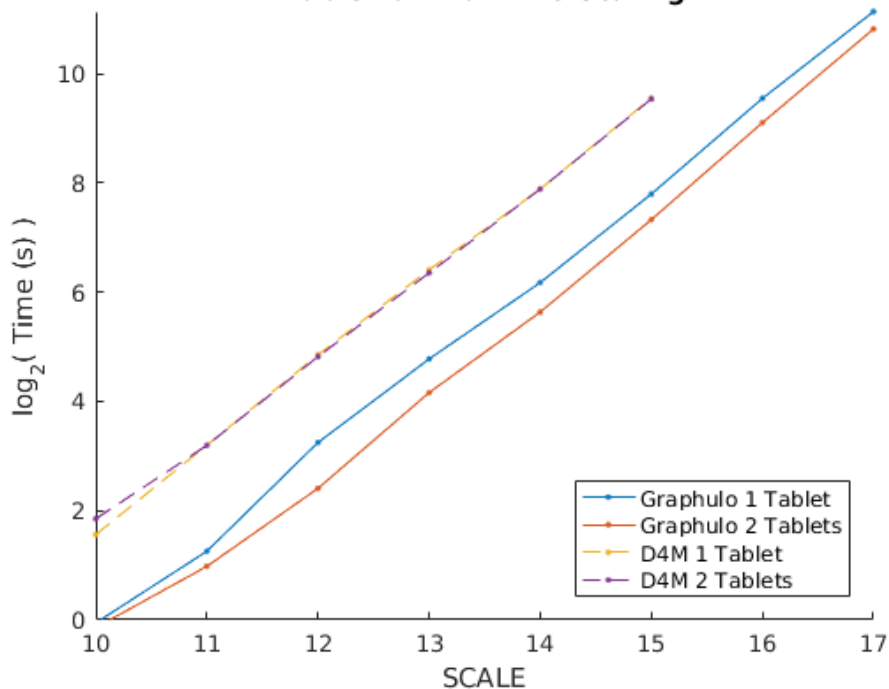
- > **Matrix Multiply (MxM), the backbone of LA**
- > **Subgraph Extraction, via MxM with a cue table**
- > **Jaccard Coefficients, a vertex similarity metric**
- > **k-Truss Subgraph, used to detect communities**

-
- > **2015 HPEC: MxM, single-node**
 - Graphulo vs. D4M (MATLAB sparse matrix library)
 - Result: Graphulo universally better
 - > **2016 HPEC: Jaccard, k-Truss, single-node**
 - Graphulo vs. D4M vs. MTJ (Java dense matrix library)
 - Result: Graphulo universally better at Jaccard;
D4M/MTJ universally better at k-Truss
 - > **2016 HPEC: MxM, subgraph extraction, cluster**
 - Graphulo vs. itself on increasing cluster size
 - Result: Linear weak & strong scaling on MxM
Subgraph extraction scales, in interactive range

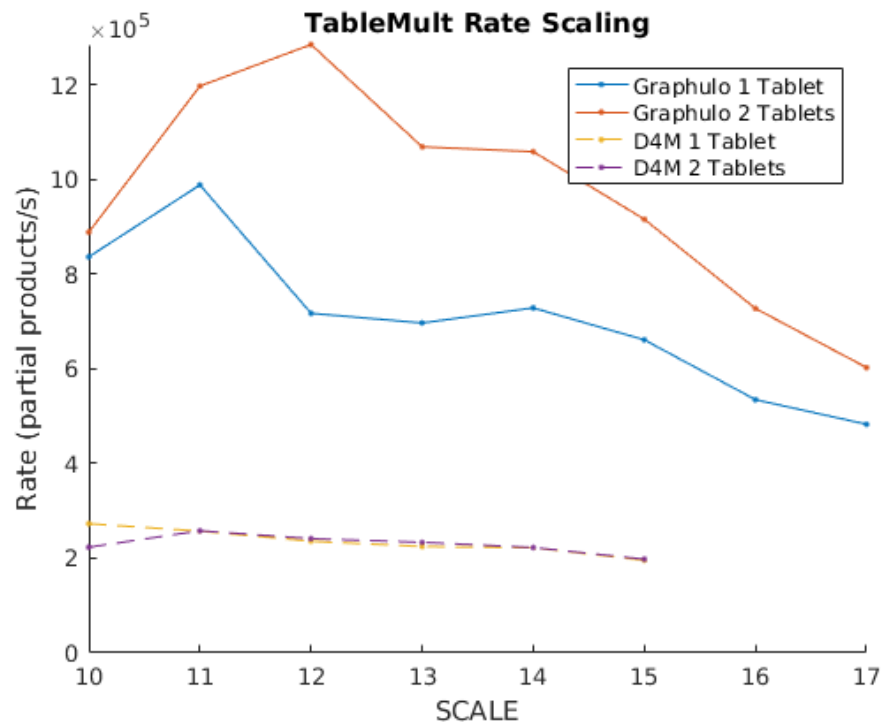


MxM Single-Node Experiment

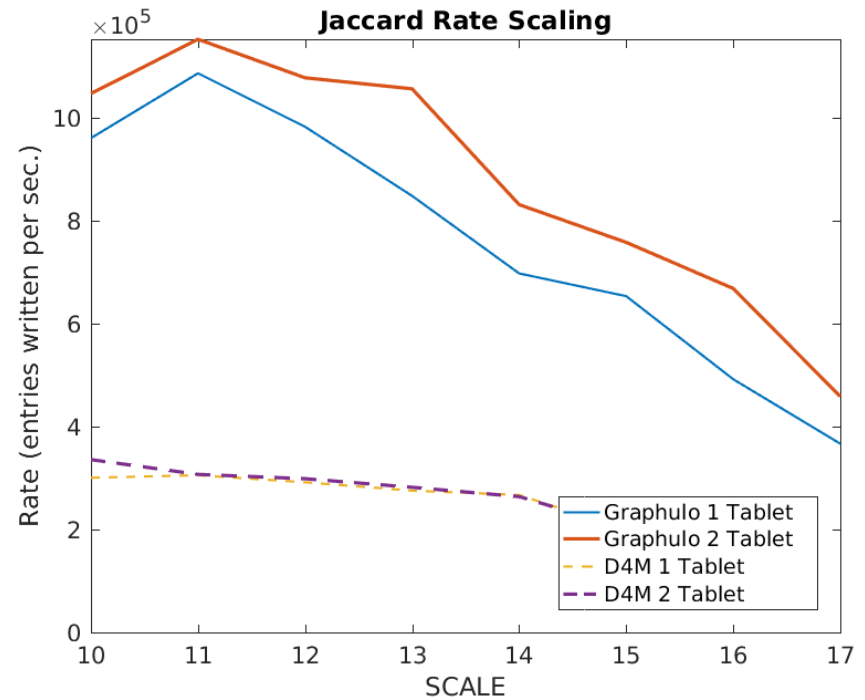
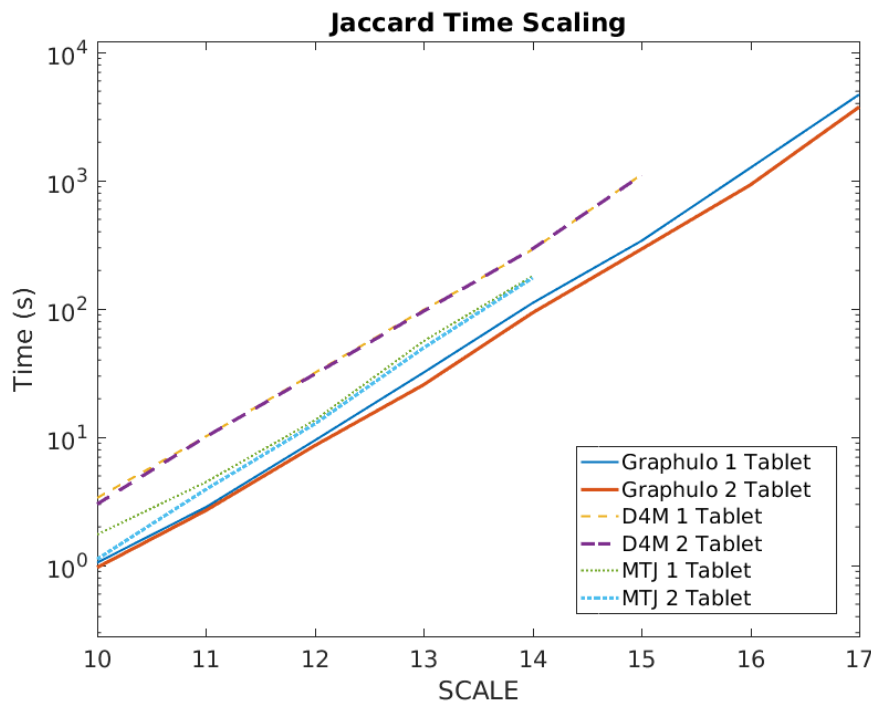
TableMult Runtime Scaling



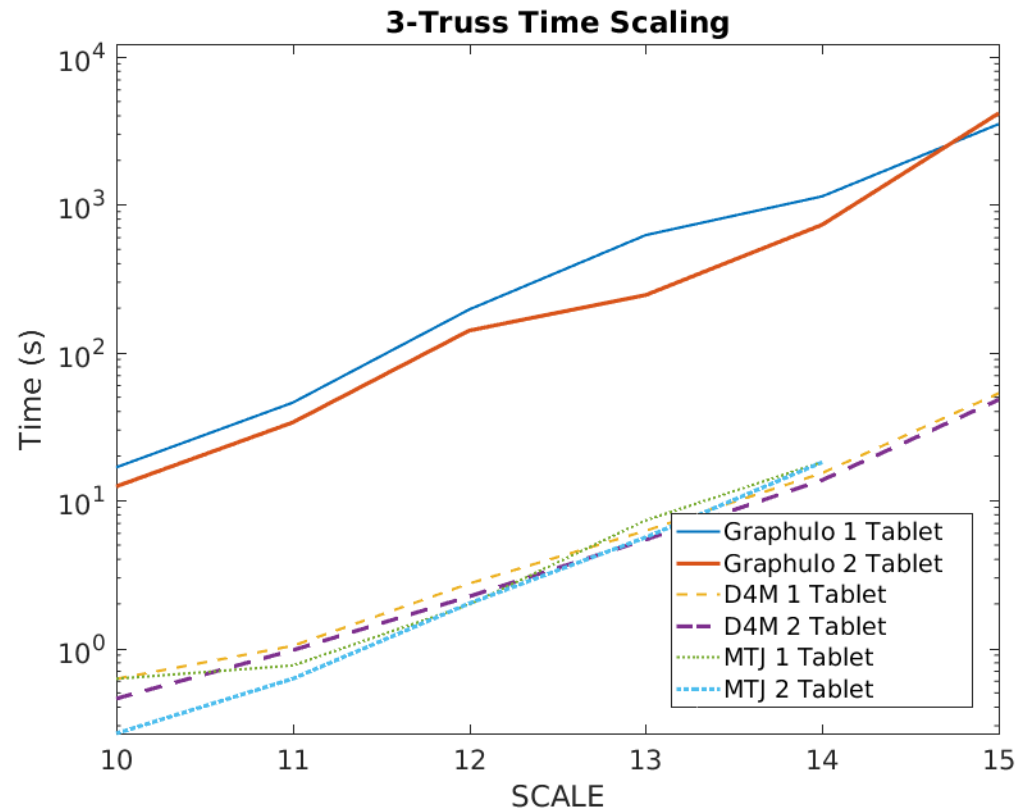
TableMult Rate Scaling



Jaccard Experiment Graphs



3-Truss Experiment Graph

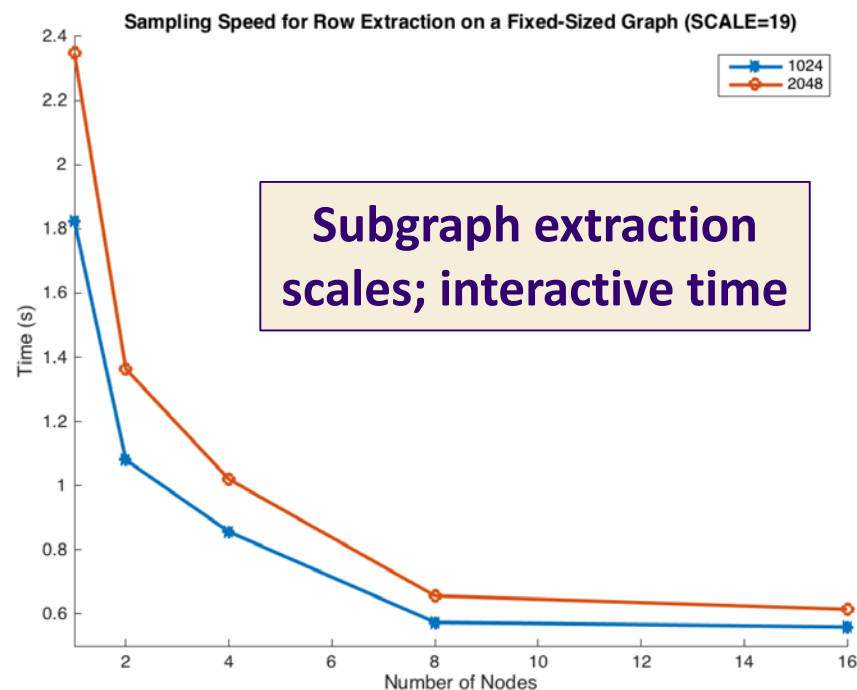
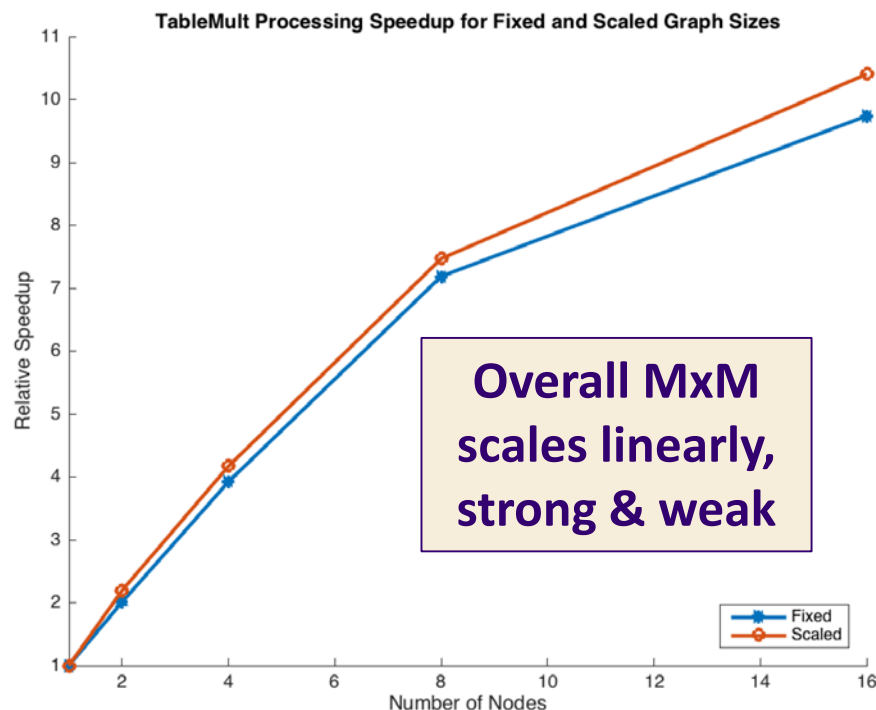


Some points may
need re-evaluating

Past Exp: Benchmarking Graphulo

Benchmarking the Graphulo Processing Framework

Timothy Weale[†], Vijay Gadepally^{‡§*}, Dylan Hutchison[°] and Jeremy Kepner^{‡§+}
[†]Department of Defense, [‡]MIT Lincoln Laboratory, [§]MIT Computer Science & AI Laboratory
⁺MIT Mathematics Department, [°]University of Washington



Alg 1: Jaccard Coefficients

Input: Unweighted, undirected adjacency matrix A

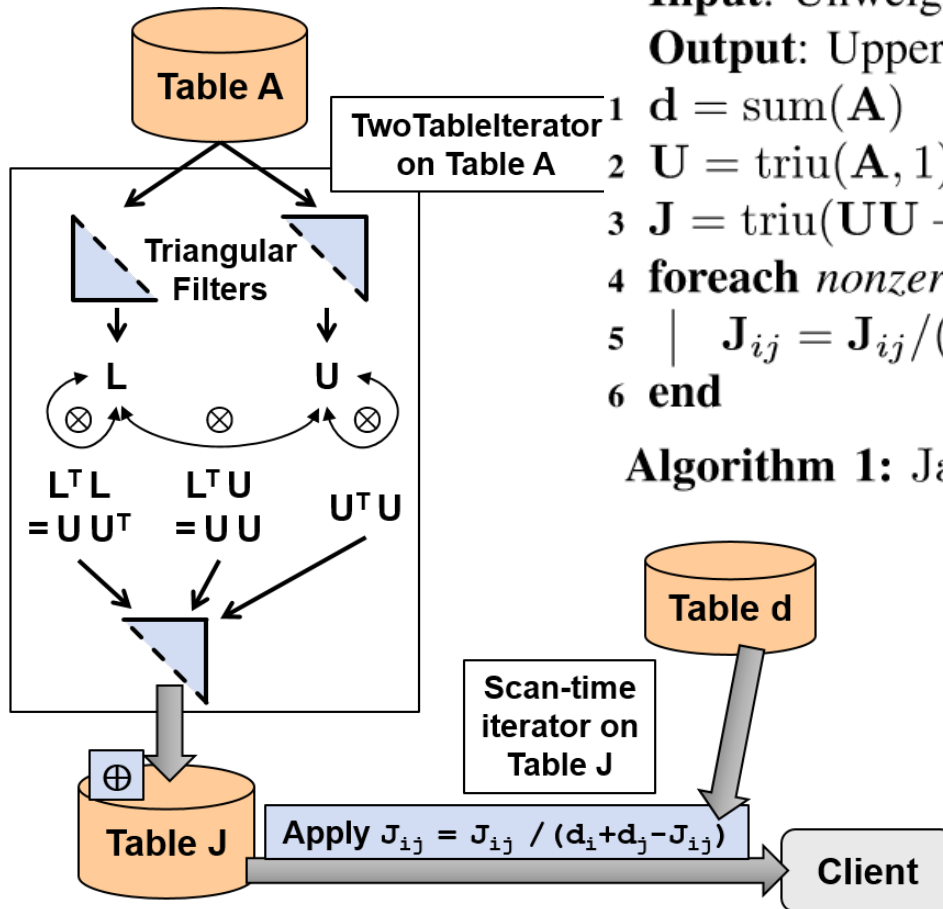
Output: Upper triangle of Jaccard coefficients J

```

1  $d = \text{sum}(A)$  // pre-computed in degree table
2  $U = \text{triu}(A, 1)$  // strict upper triangle filter
3  $J = \text{triu}(UU + UU^T + U^T U, 1)$  // fused MxM
4 foreach nonzero entry  $J_{ij} \in J$  do
5 |    $J_{ij} = J_{ij} / (d_i + d_j - J_{ij})$  // stateful Apply on J
6 end
    
```

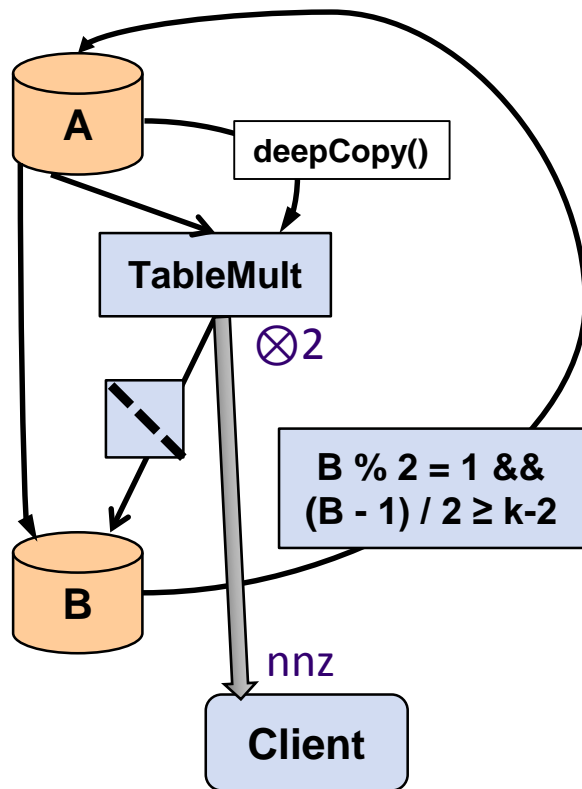
Algorithm 1: Jaccard

Fusion:
A single Graphulo
TwoTable pass!



Note! This has been improved since the HPEC paper. Doable as a special OneTable operation.

Alg 2: k-Truss Subgraph



Input: Unweighted, undirected adjacency matrix A_0 , integer k

Output: Adjacency matrix of k -truss subgraph A

```

1  $z' = \infty, A = A_0$  // table clone
2 repeat
3    $z = z'$ 
4    $B = A$  // table clone
5    $B = B + 2AA$  //  $M \times M$  with  $a \otimes b = 2$  if  $a, b \neq 0$ 
6    $B(B \% 2 == 0) = 0$  // filter on B
7    $B((B - 1)/2 < k - 2) = 0$  // filter on B
8    $A = |B|_0$  // Apply on B; switch  $A \leftrightarrow B$ 
9    $z' = \text{nnz}(A)$  // Reduce, gathering nnz at client
10 until  $z == z'$  // client controls iteration
    
```

Algorithm 2: kTruss

Iterations of Graphulo TwoTables