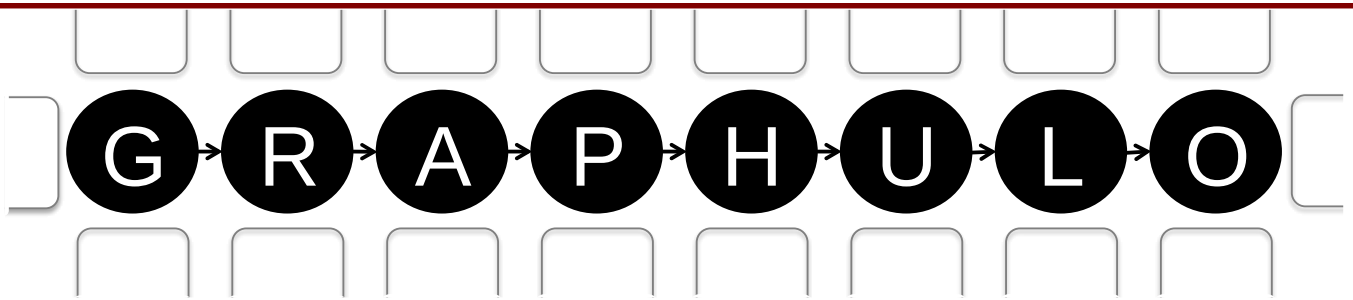


Graphulo Use and Design

Dylan Hutchison* **Vijay Gadepally*** **Jeremy Kepner***

2015 August

*This material is based upon work supported by the National Science Foundation under Grant No. DMS-1312831. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Graphulo Overview

- **Primary Goal**
  - **Open source Apache Accumulo Java library that enables many graph algorithms in Accumulo**
- **Core primitives: GraphBLAS**
- **3 Graph Schemas**
  - **Adjacency, Incidence, Single-Table**
- **4 Demonstration Graph Algorithms**
  - **Degree-filtered Breadth First Search, Jaccard coefficients, k-Truss subgraph, Non-negative Matrix Factorization**
- **Focus on Interactive Computing**
  - **"Queued" / Localized analytics within a neighborhood, as opposed to whole table analytics**
  - **Low latency more important than high throughput**
  - **Progress monitoring for user sanity**
    - ***Is the library working or stuck?***

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Outline

- **Download, install, test, see examples, use as a library. Maven build cycle.**
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Download

- **Clone Git repo at https://github.com/Accla/graphulo**

- **To build a folder containing all Javadoc:**
  - `mvn install -DskipTests`
  - **Javadoc available in `docs/apidocs`**

Test on MiniAccumulo

- **MiniAccumulo – Portable, lightweight Accumulo instance started before and stopped after each test class**
- **Enables testing without a standalone running Accumulo instance**
- **`mvn test`**
- **Test results / client logs saved in: `shippable/testresults/`**
- **If a test fails, recommended to run that test individually**
  - **`mvn clean test –Dtest=TestClassName#testMethodThatFailed`**
  - **To easily view MiniAccumulo server-side logs for the most recent singleton test, run `./lessMiniServerLogs.sh`**

    Opens the Tablet Server log in the directory indicated by the client server log. Look for the entry:

    - INFO – MiniAccumuloTester.before(66) – Temp directory: /tmp/tempMini6963629899066952349

Graphulo Maven Lifecycle

- clean – Delete `target/` and `shippable/` directories
- compile – Using Java 1.7
- test – Run all tests in TEST_CONFIG.java, output to `shippable/`
- package – Create Graphulo artifacts in `target/`
  - `graphulo-VERSION.jar` – Graphulo binaries only
    - Include on client application's classpath to call Graphulo client functions
  - `graphulo-VERSION-alldeps.jar` – Graphulo + all referenced code binaries
    - For Accumulo server installation. Place in Accumulo server's `lib/ext/`
  - `graphulo-VERSION-libext.zip` – Zip of original JARs of all dependencies.
    - For use in D4M Matlab/Octave.
- install – Create Javadoc and Graphulo distribution zip in `target/`
  - Javadoc created in `target/site/. docs/apidocs/` is a symlink.
  - `graphulo-VERSION-dist.zip` – Zip of Graphulo source and assembly files
  - Installs Graphulo into local Maven repo

Quick Accumulo Install

```
mvn package -DskipTests
./deploy.sh
```

D4M not required for Graphulo. Used for testing.

Enables local projects to depend on Graphulo before it is in Maven Central

Test on Full Accumulo

- **Edit TEST_CONFIG.java and put in the connection information for your Accumulo instance (local or remote)**
  - **Example config under the label "local"**

```
case "local":
  AccumuloTester = new RealAccumuloTester("instance",
    "localhost:2181", 5000, "root", new PasswordToken("secret"));
```

  - **Specify: Instance name, Zookeeper address and port, Zookeeper timeout, Accumulo User Name, User Password Token**
- **Pass the label (e.g. "local") to mvn test:**
  - **`mvn test -DTEST_CONFIG=local`**
  - **Server-side logs available in `$ACCUMULO_HOME/logs/tserver*.debug.log`**
  - **If a test fails for any reason, it may leave a test table on your Accumulo instance which will mess up future tests. Delete the tables manually if this happens ➔ `deletetable -p .*Test_test.* -f`**

Examples!

- See all classes in `edu.mit.ll.graphulo.examples` package
- Each fully runnable in MiniAccumulo or standalone Accumulo
- Folder `src/test/resources/data/` contains pre-created graphs
  - Kronecker power law graphs
  - SCALE 10, 12, 14, 16
- Change SCALE parameter in examples to use larger graphs
- See suggestions in comments at bottom of files for variations

Use Graphulo in derivative Maven project

- **Install Graphulo library into local Maven repository:** mvn install
- **Add to your Maven project *pom.xml*:**

```
<dependency>
  <groupId>edu.mit.ll</groupId>
  <artifactId>graphulo</artifactId>
  <version>${version}</version>
</dependency>
```

Create Graphulo Client Executor

```
ClientConfiguration cc = ClientConfiguration.loadDefault()
    .withInstance("instance").withZkHosts("localhost:2181")
    .withZkTimeout(5000);
Instance instance = new ZooKeeperInstance(cc);
AuthenticationToken token = new PasswordToken("secret");
Connector c = instance.getConnector("root", token);

Graphulo graphulo = new Graphulo(conn, token);
```

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- **Motivating algorithm: AdjBFS w/ degree filtering**
  - **Specifying Column Visibilities & Authorizations**
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Teaser Demo AdjBFS

```
dhutchis@dmasterBW:gits$ git clone git@github.com:Accla/graphulo.git
Cloning into 'graphulo'...
remote: Counting objects: 6953, done.
remote: Compressing objects: 100% (119/119), done.
remote: Total 6953 (delta 264), reused 212 (delta 212), pack-reused 6592
Receiving objects: 100% (6953/6953), 48.74 MiB | 11.06 MiB/s, done.
Resolving deltas: 100% (3249/3249), done.
Checking connectivity... done.
dhutchis@dmasterBW:gits$ cd graphulo
dhutchis@dmasterBW:graphulo$

dhutchis@dmasterBW:graphulo$ mvn test -Dtest=AdjBFSExample -DTEST_CONFIG=local
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building graphulo 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
dhutchis@dmasterBW:graphulo$ cat shippable/testresults/edu.mit.ll.graphulo.examples.AdjBFSExample-output.txt
10 Aug 2015 20:04:44,589 WARN - ClientConfiguration.loadFromSearchPath(227) -  Found no client.conf in default paths. Using default cl
10 Aug 2015 20:04:44,780 DEBUG - RealAccumuloTester.before(52) -  setUp ok - ClientConfiguration=org.apache.accumulo.core.client.Clien
10 Aug 2015 20:04:49,766 INFO - ExampleUtil.ingestAdjacencySCALE(35) -  Wrote 16384 edges to D4M Adjacency tables with base name ex10A
10 Aug 2015 20:04:51,811 DEBUG - Graphulo.OneTable(827) -  27 :%00; [] 9223372036854775807 false -> 1087 entries processed
10 Aug 2015 20:04:52,181 DEBUG - Graphulo.OneTable(827) -  99 :%00; [] 9223372036854775807 false -> 8798 entries processed
10 Aug 2015 20:04:52,403 DEBUG - Graphulo.OneTable(827) -  99 :%00; [] 9223372036854775807 false -> 8856 entries processed
10 Aug 2015 20:04:52,404 INFO - AdjBFSExample.exampleAdjBFS(91) -  First few nodes reachable in exactly 3 steps: 338,339,941,945,332,
10 Aug 2015 20:04:52,554 INFO - AdjBFSExample.exampleAdjBFS(103) -  # of entries in output table 'ex10Astep3: 8856
dhutchis@dmasterBW:graphulo$
```

Teaser: AdjBFS with Graphulo

```
int numSteps = 3;
String Atable = "ex10A";          // Input  table
String Rtable = "ex10Astep3";     // Output table
String RTtable = null;
String ADegtable = "ex10ADeg";    //
String degColumn = "out";         // Degree table column qual.
boolean degInColQ = false;        // Degrees in Value
int minDegree = 20;               // High-pass filter
int maxDegree = Integer.MAX_VALUE;
int AScanIteratorPriority = -1; // Default scan iter. priority
String v0 = "1,25,:,27,";         // Starting nodes/range
Map<Key,Value> clientResultMap = null; // Not outputting to client
Authorizations Aauth = EMPTY, ADegauth = EMPTY;
boolean outputUnion = false;  // Return nodes EXACTLY k steps away
MutableLong numEntriesWritten = new MutableLong(); // Output var.

String vReached = graphulo.AdjBFS(Atable, v0, numSteps, Rtable,
    RTtable, clientResultMap, AScanIteratorPriority,
    ADegtable, degColumn, degInColQ, minDegree, maxDegree,
    plusOp, Aauth, ADegauth, numEntriesWritten);
```

Allows different degree table schemas, such as putting degree in Column Qualifier instead of Value

Degree filtering on the fly with SmallLargeRowFilter if no degree table given

Be careful with priority when stacking iterators!

Graphulo Design: Methods take many parameters. Pass null or -1 to use "defaults"

AdjBFS expressed in core Graphulo ops

```
for (int thisk = 1; thisk <= k; thisk++) {
  vk = filterDegreeTable(ADegtable, degColumnText, degInColQ,
                         minDegree, maxDegree, vk);

  GatherColQReducer reducer = new GatherColQReducer();
  reducer.init(Collections.<String, String>emptyMap(), null);

  long numWrites = OneTable(Atable, Rtable, RTtable, clientResultMap,
      AScanIteratorPriority, reducer,
      Collections.<String, String>emptyMap(), plusOp,
      rowFilter, null, // no column filter
      iteratorSettingList, bs, auths);

  vk.clear();
  vk.addAll(reducer.getSerializableForClient());
}
```

Scan degree table, if given, to filter nodes

Gather reached nodes in a Reducer, returned to client

Reached nodes vk set as rowFilter

OneTable for main scan

Reached nodes used in next BFS step

AdjBFS Degree Filter helper calls Accumulo

BatchScanner connected to Degree table.
Passed as an argument to allow thread pool re-use.

```
private Collection<String> filterDegreeTable(BatchScanner bs,Text degColumnText,
    boolean degInColQ, int minDegree, int maxDegree, Collection<Range> ranges) {
  Collection<String> texts = new HashSet<>();
  bs.setRanges(ranges);
  if (!degInColQ)
    bs.fetchColumn(EMPTY_TEXT,degColumnText==null ? EMPTY_TEXT : degColumnText);
  else if (degColumnText != null) {
    IteratorSetting itset = new IteratorSetting(1, ColumnSliceFilter.class);
    ColumnSliceFilter.setSlice(itset, degColumnText.toString(),
        true, Range.followingPrefix(degColumnText).toString(), false);
    bs.addScanIterator(itset);
  }
  bs.addScanIterator(MinMaxFilter.iteratorSetting(50,ScalarType.LONG, minDegree,
    maxDegree,degInColQ,degColumnText==null ? null : degColumnText.toString()));

  Text tmp = new Text();
  for (Map.Entry<Key, Value> entry : bs)
    texts.add(entry.getKey().getRow(tmp).toString());
  return texts;
}
```

ranges given are those of v_k: starting nodes for this step

Moral: Can mix'n'match Graphulo and Accumulo client functions

Authorizations & Column Visibilities

- **Background**
  - **Accumulo users authorized to use a selected set of Visibility labels**
    - **Change via** `connector.securityOperations().changeUserAuthorizations(user, newAuths);`
  - **Authorizations to use decided at (Batch)Scanner creation time**
  - **Only table entries whose Column Visibility matches (Boolean algebra) scanner Authorizations are seen in any scan, including Graphulo's scans**
- **Most Graphulo functions take an Authorizations argument**
  - **Throws RuntimeException if user not authorized to use given labels**
- **Newly created keys inherit Visibility when possible**
  - **Not possible in general with MultiplyOp (two parent Keys)**
  - **`newVisibility` argument overrides, applying to all newly created Keys**
- **For fine-grained Visibility control, implement a custom server-side op**

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- **Three Graph Schemas: Adjacency, Incidence, Single-Table**
  - **Degree Tables and utility functions to help ingest**
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Graph Schemas: Adjacency

- Row = start node label
- Column Qualifier = end node label
- Value = edge weight

Adjacency Table

```
1 :1 []  -> 141
1 :10 []  -> 12
1 :101 [] ->  9
1 :105 [] ->  3
1 :11 []  ->  9
1 :110 [] ->  3
1 :111 [] ->  3
1 :113 [] -> 12
10 :1 []  -> 18
10 :109 [] -> 2
10 :136 [] -> 2
10 :137 [] -> 2
```

Degree Table

- Row = node label
- Column Qualifier = fixed degree label
  - Track both in- and out-degree if desired
- Value = degree

Support for some variants, such as placing degree in the Column Qualifier

Degree Table

```
1 :in [] ->  1084
1 :out [] -> 1027
10 :in [] ->  118
10 :out [] ->  94
100 :in [] ->   8
100 :out [] -> 10
```

Graph Schemas: Incidence (Edge)

- Row = edge label
- Column Qualifier =
  edge direction prefix
  + separator + node label
- Value = edge weight

Incidence Table

```
00001 :in|907 []  -> 2
00001 :out|23 []  -> 2
00010 :in|769 []  -> 2
00010 :out|643 [] -> 2
00011 :in|419 []  -> 2
00011 :out|545 [] -> 2
00020 :in|67 []   -> 3
00020 :out|262 [] -> 3
00030 :in|17 []   -> 2
00030 :out|514 [] -> 2
00031 :in|424 []  -> 2
00031 :out|519 [] -> 2
00040 :in|259 []  -> 2
00040 :out|9 []   -> 2
```

Degree Table

- Row = node label
- Column Qualifier = fixed degree label
  - Track both in- and out-degree if desired
- Value = degree

Degree Table

```
1 :in [] ->  1084
1 :out [] -> 1027
10 :in [] ->  118
10 :out [] ->  94
```

Graph Schemas: Single-table

Always Undirected

- "v1|v2" implies "v2|v1"

Two kinds of rows:

1. **Degree row**
   - Row = node label
   - Column qualifier = "deg"
   - Value is out-degree of node
2. **Edge row**
   - Row = out-node label + separator + in-node label
   - Column qualifier = "edge"
   - Value is edge weight

Single Table

```
163       :deg  [] -> 4
163|1013  :edge [] -> 4
...
270       :deg  [] -> 7
270|1012  :edge [] -> 7
...
933       :deg  [] -> 1
933|1010  :edge [] -> 1
...
1010      :deg  [] -> 5
1010|1013:edge [] -> 2
1010|933  :edge [] -> 3
1012      :deg  [] -> 3
1012|270  :edge [] -> 3
1013      :deg  [] -> 6
1013|1010:edge [] -> 3
1013|163  :edge [] -> 3
```

Utility: Ingesting Graph Data

TripleFileWriter: Easiest way to ingest graph data

- Create delimited text files storing edges
  - Row file contains edge start label
  - Col file contains edge end label
  - Optional val file contains edge weight
- Pass files to TripleFileWriter constructor
- Call Adjacency, Incidence or SingleTable ingest method

See ExampleUtil:

```
TripleFileWriter tripleFileWriter = new TripleFileWriter(conn);
File rowFile = getDataFile(String.valueOf(SCALE)+version+"r.txt");
File colFile = getDataFile(String.valueOf(SCALE)+version+"c.txt");
long cnt = tripleFileWriter.writeTripleFile_Adjacency(
                rowFile, colFile, null, ",", baseName, true, true);
```

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- **Mapping to GraphBLAS**
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

GraphBLAS Functions

Function	Parameters	Returns	Math Notation
SpGEMM	- sparse matrices **A** and **B** - unary functors (op)	sparse matrix	**C** $\oplus$= **A** $\oplus.\otimes$ **B**
SpM{Sp}V **(Sp: sparse)**	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** $\oplus$= **A** $\oplus.\otimes$ **x**
SpEWiseX & **SpEWiseSum**	- sparse matrices or vectors - binary functor and predicate	in place or sparse matrix/vector	**C** $\oplus$= **A** $\otimes$ **B** **C** $\oplus$= **A** $\oplus$ **B**
Reduce	- sparse matrix **A** and functors	dense vector	**y** $\oplus$= $\oplus_i$ **A**(i,:)
SpRef	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** $\oplus$= **A**(**p**,**q**)
SpAsgn	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**,**q**) $\oplus$= **B**
Apply	- any matrix or vector **X** - unary functor (op)	none	**C** $\oplus$= f(**X**)

GraphBLAS Functions

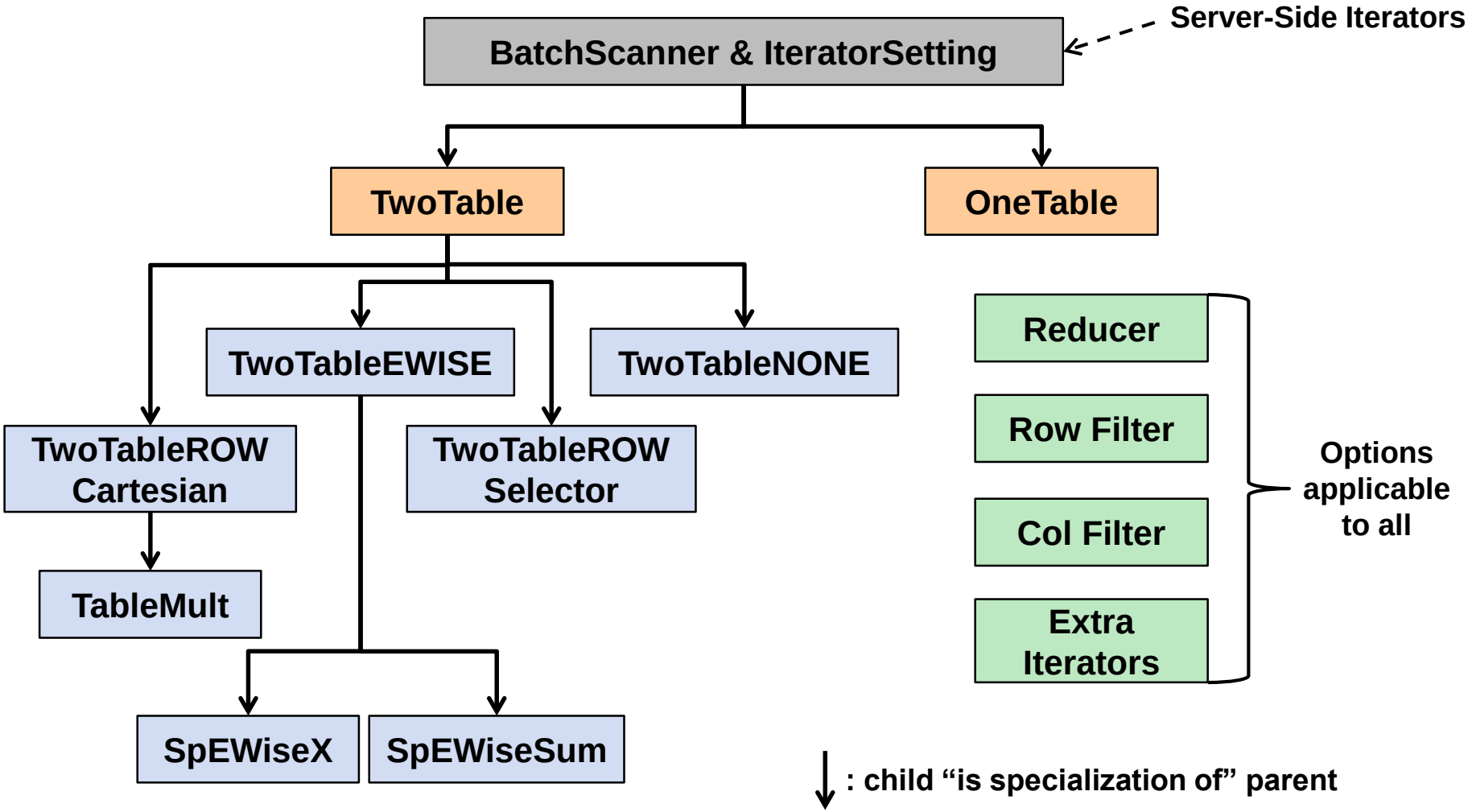
Function	Graphulo Function	Math Notation
SpGEMM	- TableMult	**C** $\oplus$= **A** $\oplus.\otimes$ **B**
SpM{Sp}V	- TableMult (no distinction b/w matrix and vector)	**y** $\oplus$= **A** $\oplus.\otimes$ **x**
SpEWiseX & **SpEWiseSum**	- SpEWiseX - SpEWiseSum, or use OneTable+Combiner	**C** $\oplus$= **A** $\otimes$ **B** **C** $\oplus$= **A** $\oplus$ **B**
Reduce	- OneTable w/ Reducer; gathered at client	**y** $\oplus$= $\oplus_i$ **A**(i,:)
SpRef	- OneTable w/ row and col subsets	**B** $\oplus$= **A**(**p**,**q**)
SpAsgn	- OneTable **B**→**A** w/ custom iter. changing keys - Not as well defined (form of index vectors?)	**A**(**p**,**q**) $\oplus$= **B**
Apply	- OneTable w/ ApplyOp iter.	**C** $\oplus$= f(**X**)

Loop fusion: Graphulo enables performing operations at the same time, up to the point a sort is required (handled by BatchWriter)

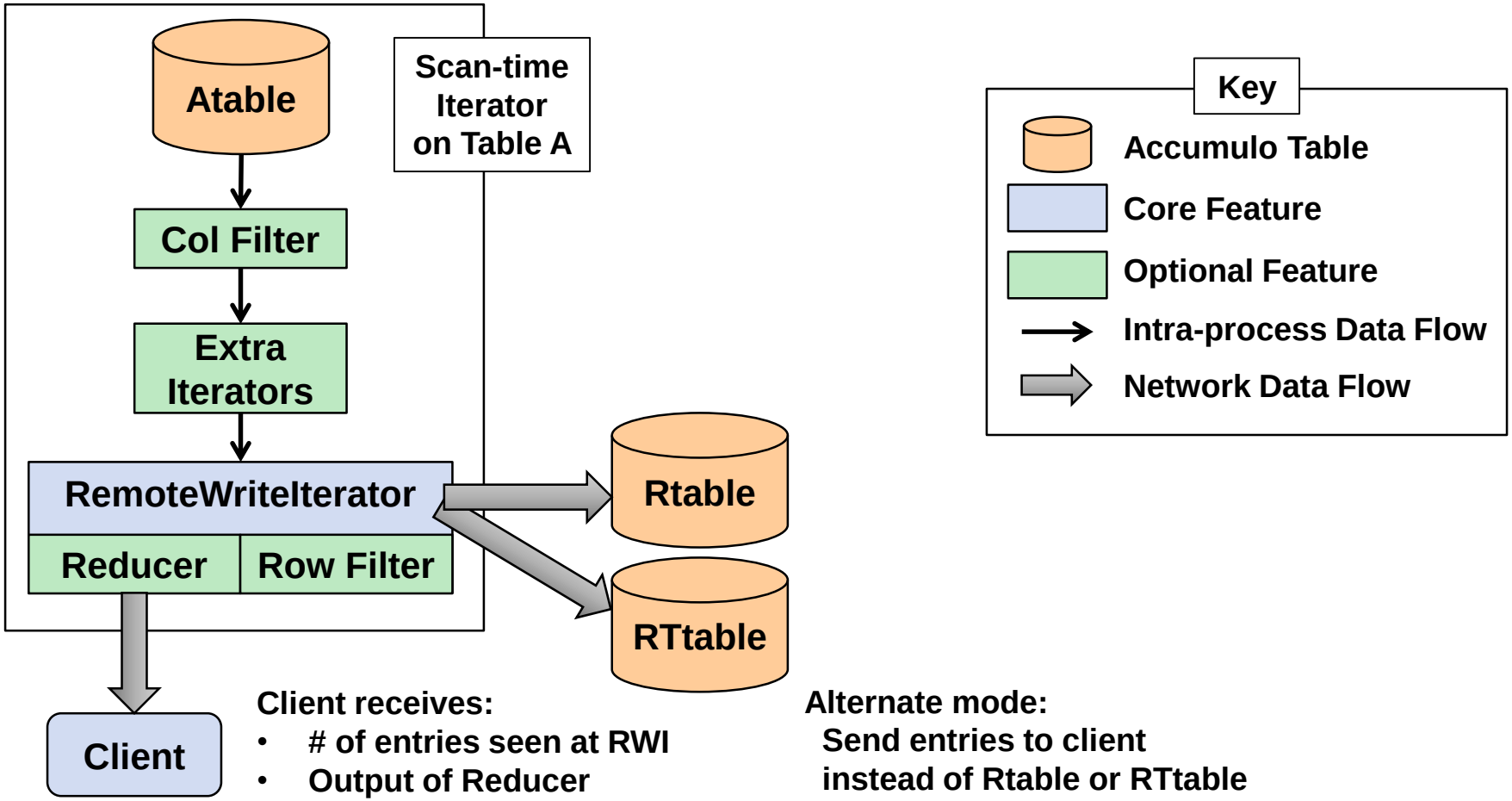
Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- **Graphulo Core Client functions and their Server-side Iterators**
  - **OneTable, Reducer, D4M String format, ApplyOp**
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Graphulo Client Functions



OneTable: Overview

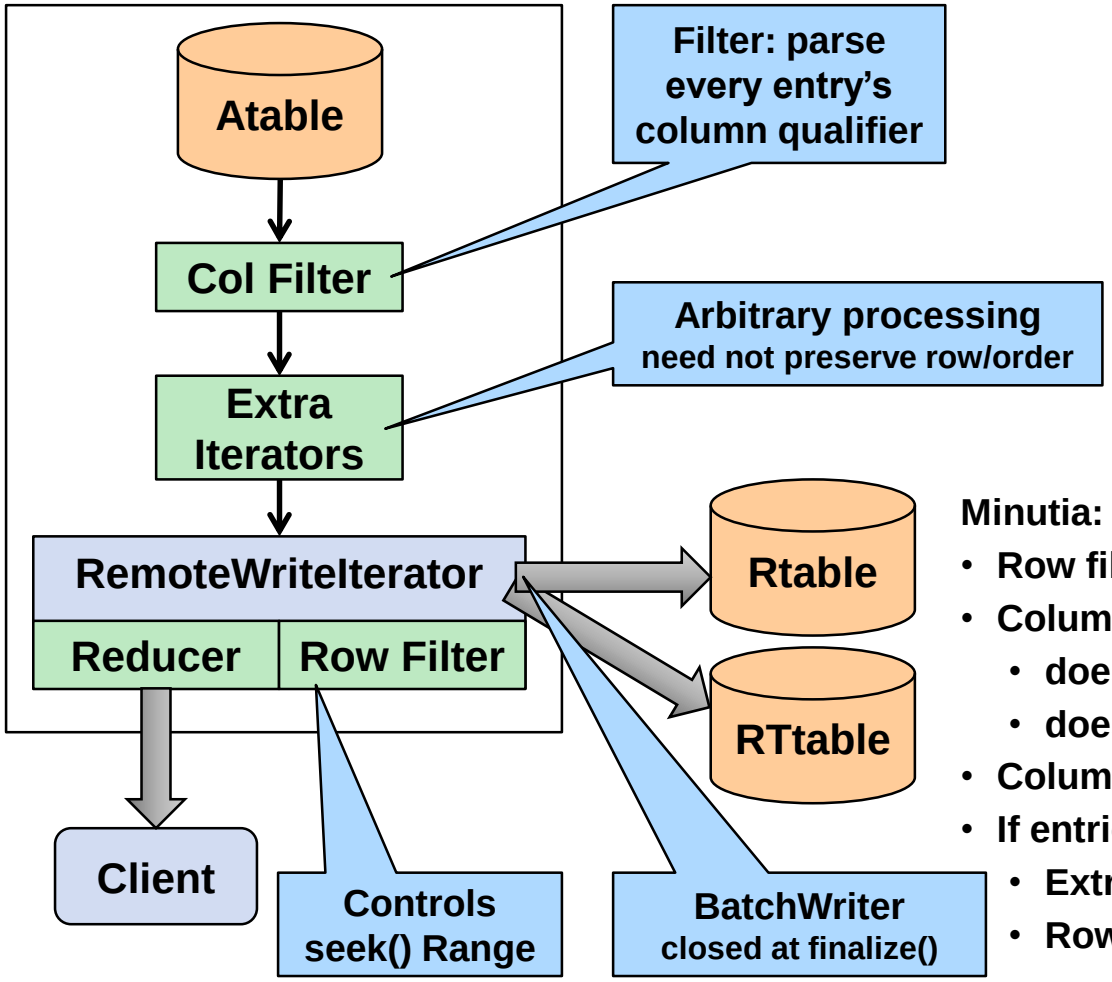


Client receives:

- # of entries seen at RWI
- Output of Reducer

Alternate mode:
Send entries to client instead of Rtable or RTtable

OneTable: Components



Minutia:

- Row filter reduces #entries read from disk
- Column filter
  - does *not* reduce #entries read from disk
  - does reduce #entries seen downstream
- Column & Row filter transmitted as D4M Strings
- If entries sent to client instead of tables,
  - Extra iterators must preserve order
  - Row filter occurs upstream

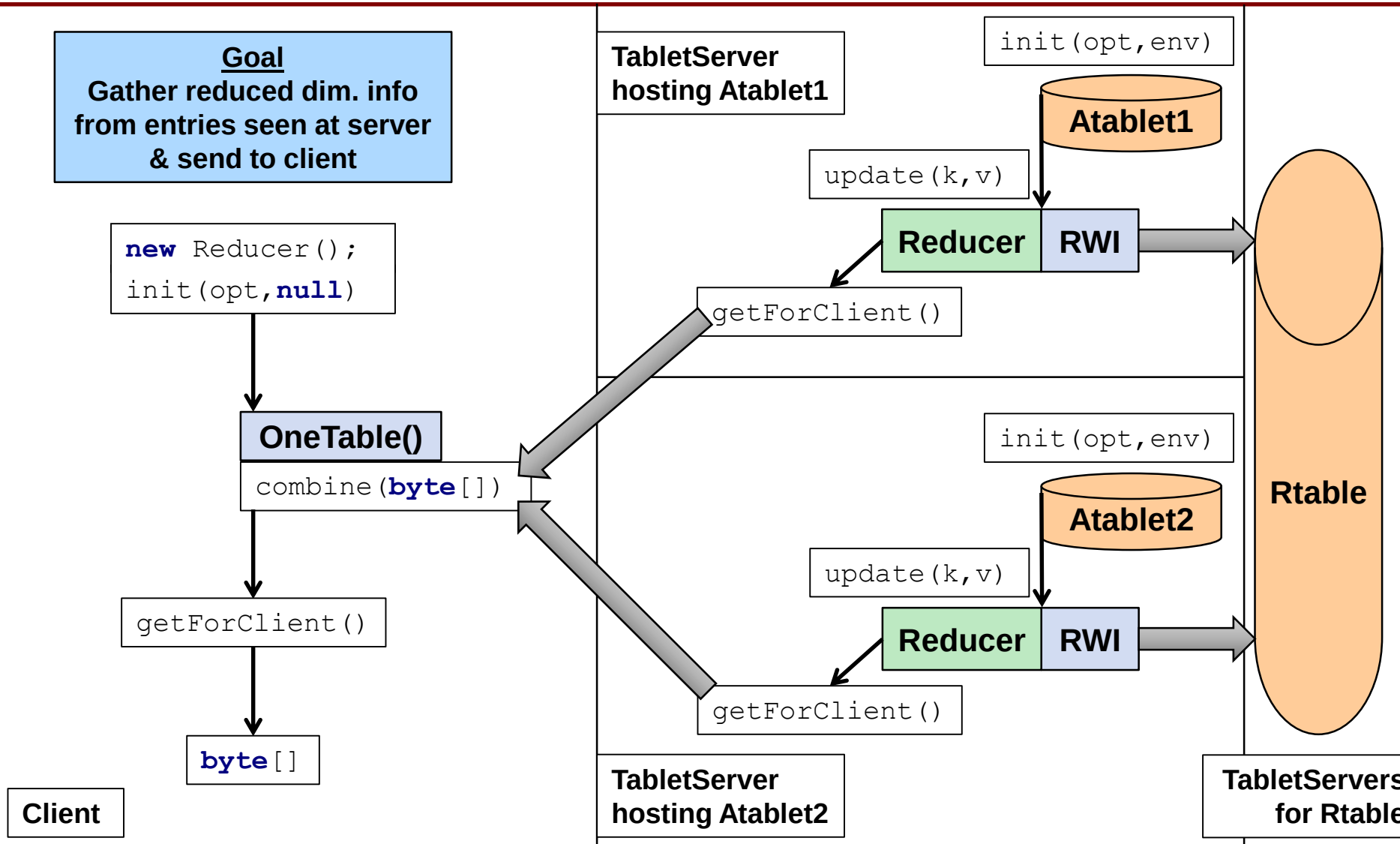
Row/Col Filter Format: D4M String

- **Graphulo controls row and column filtering in server-side iterators. Benefits:**
  - **Control seeks to disjoint Ranges without needing an SKVI to return / cede control**
    - **Enables safe, long-lived BatchWriting**
  - **Specify column range filter to same precision as Row Ranges (still reads all columns)**
- **Row/Col ranges sent to iterators via IteratorOptions – requires String encoding**
  - **Plain `Range` & `Text` objects don't serialize**
  - **Motivates D4M String format**
- **D4M Strings are 1-1 with Ranges**
- **Utilities convert Ranges ←→ D4M String:**
  `GraphuloUtil.d4mRowToRanges( )`
  `GraphuloUtil.rangesToD4MString( )`
  - **Similar utilities convert Range → String**
  - **Helper methods for prefix ranges**
- **Empty String and null always have same semantic meaning to prevent confusion**

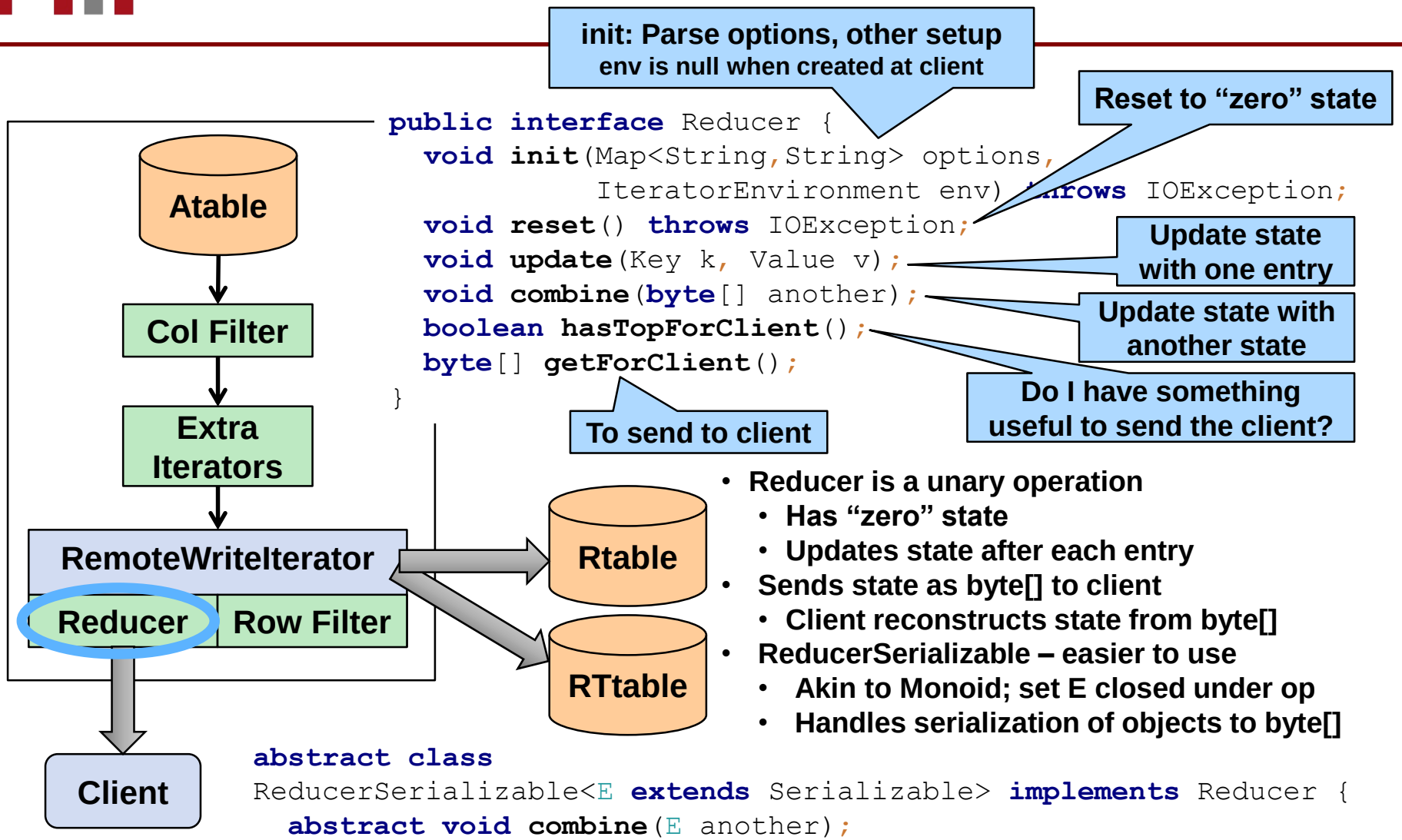
D4M String	Range
`:,`	(-inf,+inf)
`:,c,`	(-inf,c]
`f,:,`	[f,+inf)
`b,:,g`	[b,g]
`b,:,g,x`	[b,g] U [c,c)
`x,`	[x,x)
`x,z,`	[x,x) U [z,z)
`x,z,:`	[x,x) U [z,+inf)
`,`	["",""] (the empty string row)
`,a,`	["",""] U [a,a)
`,:,b,f,:,`	["",b] U [f,+inf)

Arbitrary separator character
Pick one that never occurs elsewhere in String

Reducer: Big Picture



Reducer: Interface



Reducer example: GatherColQ

```
class GatherColQReducer extends ReducerSerializable<HashSet<String>> {

  private HashSet<String> setColQ = new HashSet<>();
  private Text tmpTextColQ = new Text();

  public void reset() throws IOException {
    setColQ.clear();
  }

  public void update(Key k, Value v) {
    setColQ.add(k.getColumnQualifier(tmpTextColQ).toString());
  }

  public void combine(HashSet<String> another) {
    setColQ.addAll(another);
  }

  public boolean hasTopForClient() {
    return !setColQ.isEmpty();
  }

  public HashSet<String> getSerializableForClient() {
    return setColQ;
  }
}
```

Gathers set of unique column qualifiers of all seen entries for transmission to client

Used in AdjBFS to gather nodes reached in current BFS step & send to client

In Graphulo, GatherColQReducer is generalized to GatherReducer, for any part of a Key

OneTable: Method Call

```
// Return #entries processed at RemoteWriteIterator or client
long OneTable(
  String Atable, String Rtable, String RTtable,    // Input, output table names
  Map<Key, Value> clientResultMap,                 // control whether to use RWI
  int AScanIteratorPriority,                       // Scan-time iterator priority
  Reducer reducer, Map<String,String> reducerOpts,// Applies at RWI and/or client
  IteratorSetting plusOp,                    // For output tables; priority matters
  String rowFilter,                          // D4M String format "c1,:,c3,c8,"
  String colFilter,                          // D4M String format
  List<IteratorSetting> midIterator,         // Extra iterators
  BatchScanner bs,                           // Optimization: re-use BatchScanner
  Authorizations authorizations)
```

- **Blocks until operation finishes**
- **Null or -1 default for most parameters**
  - **Reducer mutated if given; must be init()'ed prior to call**
  - **BatchScanner options cleared if given**
  - **clientResultMap filled with entries if given**

ApplyOp: Easy unary function SKVIs

```
interface ApplyOp {
  void init(Map<String,String> opt, IteratorEnvironment env) throws IOException;

  Iterator<? extends Map.Entry<Key, Value>> apply(Key k, Value v);

  // Optional: passes seek range to the applyOp, called before any apply().
  void seekApplyOp(Range range, Collection<ByteSequence> columnFamilies,
                   boolean inclusive) throws IOException;
}
```

- **Easy way to add a custom unary function in the middle of a OneTable or TwoTable**
- **ApplyOp class name passed to ApplyIterator as an option:**

```
IteratorSetting itset = new IteratorSetting(priority, ApplyIterator.class);
itset.addOption(ApplyIterator.APPLYOP, RowToDiagonalApply.class.getName());
```

ApplyOp example: RowToDiagonalApply

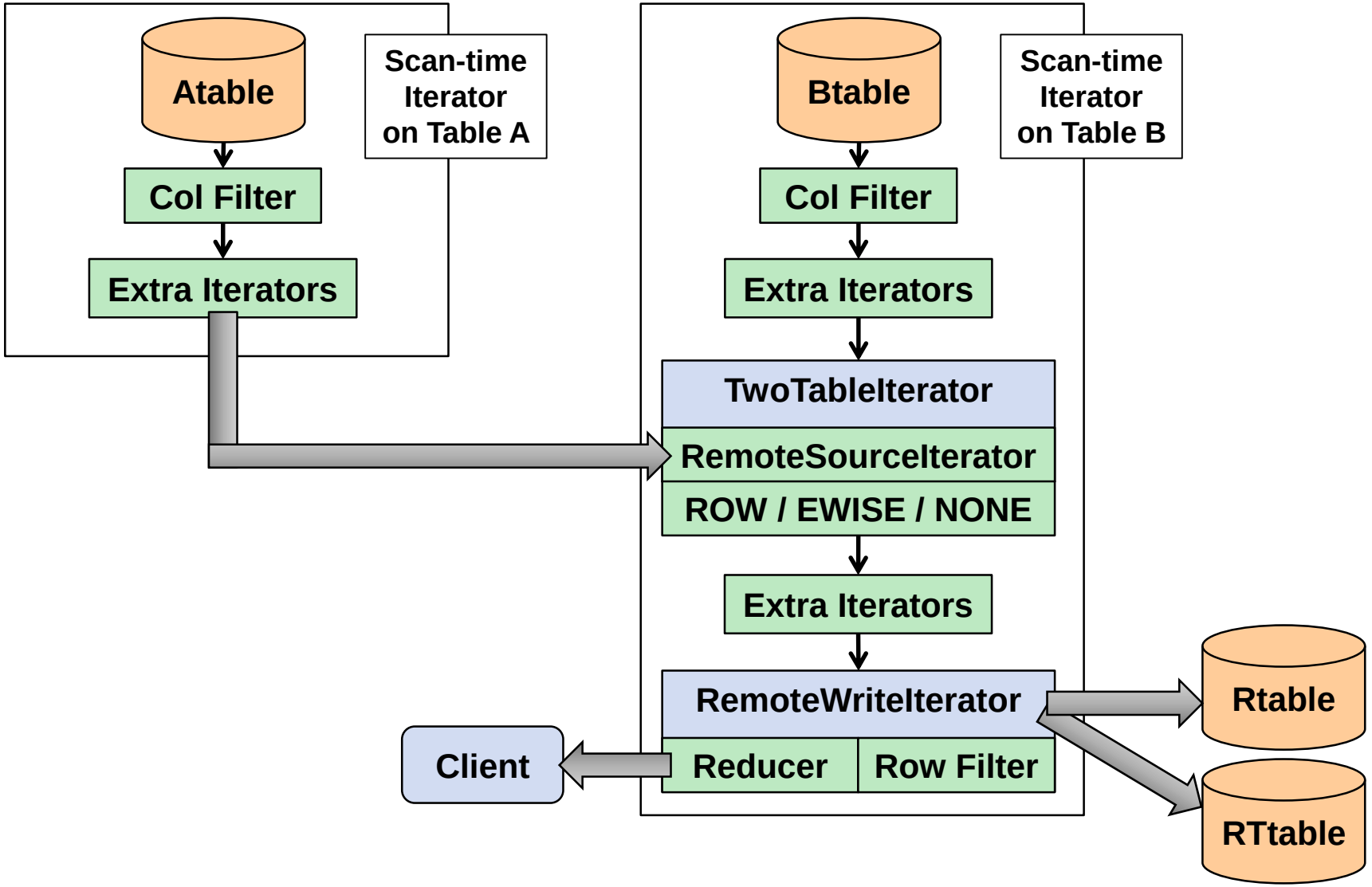
```
Iterator<? extends Map.Entry<Key, Value>> apply(Key k, Value v) {
  Text row = k.getRow();
  Key knew = new Key(row, EMPTY_TEXT, row, System.currentTimeMillis());
  return Iterators.singletonIterator(
      new AbstractMap.SimpleImmutableEntry<>(knew, v));
}
```

- **Modify key: set column qualifier equal to row**

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- **Graphulo Core Client functions and their Server-side Iterators**
  - OneTable, Reducer, D4M String format, ApplyOp
  - **TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp**
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

TwoTable: Overview



TwoTableIterator

Two SKVI sources – parent local iterator or RemoteSourceIterator

1. **Seek both sources to same Range**
2. **Advance sources in lockstep until they match Keys**
   - **ROW mode: match on Row only**
   - **EWISE mode: match on Row, Column Family, Colum Qualifier**
3. **Call operation on matching entries, emitting results**
4. **Advance sources after result entries returned**

Can also call an operation on non-matching entries

RemoteSourceIterator

- Opens a Connector and Scanner to a remote table
- Pass in all credentials
- Options:
  - instanceName
  - tableName
  - zookeeperHost
  - timeout – integer zookeeper timeout
  - username
  - password
  - authorizations
  - rowRanges – D4M String
  - colFilter – D4M String
  - doClientSideIterators – boolean
  - diter.* – DynamicIteratorSetting

Setup key-by-key, or call
`RemoteSourceIterator.iteratorSetting()`

DynamicIterator

Load several iterators from a single iterator setting

Example from utility function generateDegreeTable():

```
DynamicIteratorSetting dis = new DynamicIteratorSetting(22, "genDegs");
if (countColumns)                  // Abs0
  dis.append(ConstantTwoScalar.iteratorSetting(1,new Value("1".getBytes())));
dis.append(KeyRetainOnlyApply.iteratorSetting(1,PartialKey.ROW))
   .append(DEFAULT_PLUS_ITERATOR)
   .append(new IteratorSetting(1, RemoteWriteIterator.class,
                               basicRemoteOpts("", Degtable)));
dis.addToScanner(bs);
```

To use with RemoteSourceIterator, build an options map:

```
Map<String, String> opt = new HashMap<>();
// …add other RemoteSourceIterator options to opt…
DynamicIteratorSetting dis = new DynamicIteratorSetting(1, null)
                              .append(DEFAULT_PLUS_ITERATOR);
opt.putAll(dis.buildSettingMap("diter."));
bs.addScanIterator(new IteratorSetting(1, RemoteSourceIterator.class, opt));
```

EWiseOp

```
interface EWiseOp {
  void init(Map<String,String> opt, IteratorEnvironment env) throws IOException;

  Iterator<? extends Map.Entry<Key, Value>> multiply(
      ByteSequence Mrow, ByteSequence McolF, ByteSequence McolQ,
      ByteSequence McolVis, Value Aval, Value Bval);
}
```

- **Class name passed as option to TwoTableIterator**
- **Mrow, McolF, McolQ, McolVis are byte[]s from the matching Keys**
  - **Do not modify them**
- **Aval and Bval are Values from Atable and Btable, in order**
- **Emit any number of entries through the returned Iterator**

- **Aval or Bval can be null when TwoTableIterator configured to emit non-matching entries; otherwise never null**

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- **Graphulo Core Client functions and their Server-side Iterators**
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - **TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE**
    - **TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply**
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

TwoTableROW: RowMultiplyOp

```
interface RowMultiplyOp {
  void init(Map<String,String> opt, IteratorEnvironment env) throws IOException;

  Iterator<Map.Entry<Key,Value>> multiplyRow(
      SortedKeyValueIterator<Key,Value> skviA,
      SortedKeyValueIterator<Key,Value> skviB) throws IOException;
}
```

ROW mode of TwoTableIterator

- **Takes a class that operates on matching rows**
- **Expected to advance the iterators for the two sources to the next row before returning**
- **Can return any number of entries via the returned Iterator**
- **skviA or skviB can be null when TwoTableIterator configured to emit non-matching entries; otherwise never null**
- **Advanced extension interface: RowStartMultiplyOp**
  - **Called at beginning of new row**

```
interface RowStartMultiplyOp extends MultiplyOp {
  boolean startRow(SortedMap<Key,Value> mapRowA,
      SortedMap<Key,Value> mapRowB, boolean isCollision);
}
```

TwoTableROW: CartesianRowMultiply

- CartesianRowMultiply implements standard matrix multiply
- ROWMODE options
  - ONEROWA holds a row of A in memory while iterating through row of B
  - ONEROWB holds a row of B in memory while iterating through row of A
  - TWOROW holds both rows in memory
    - Some operations need to see the entire two matching rows at once
- Actual operation specified by MultiplyOp

```
interface MultiplyOp {
  void init(Map<String,String> opt, IteratorEnvironment env) throws IOException;

  Iterator<? extends Map.Entry<Key, Value>> multiply(
      ByteSequence Mrow, ByteSequence ATcolF, ByteSequence ATcolQ,
      ByteSequence ATcolVis, ByteSequence BcolF, ByteSequence BcolQ,
      ByteSequence BcolVis, Value ATval, Value Bval);
}
```

- Standard multiply: `new Key(ATcolQ, ATcolF, BcolQ)` and `ATval*Bval`
- Flags `alsoDoAA`, `alsoDoBB` to perform A*A or B*B at same time

TwoTableROW: SelectorRowMultiply

Second example of a RowMultiplyOp (simplified from actual code)

```
Iterator<Map.Entry<Key,Value>> multiplyRow(
    SortedKeyValueIterator<Key, Value> skviA,
    SortedKeyValueIterator<Key, Value> skviB) throws IOException {

  // advance selector skvi to next row
  Iterator<Map.Entry<Key,Value>> sri
      = new SKVIRowIterator(skviA);
  while (sri.hasNext())
    sri.next();

  return new SKVIRowIterator(skviB);
}
```

- **Emits rows of B for which there exists a matching row of A**
  - **No multiplication or parsing of columns**
- **SKVIRowIterator creates a Java iterator over an SKVI's current row**
  - **Satisfies post-condition: both iterators advanced to next row**

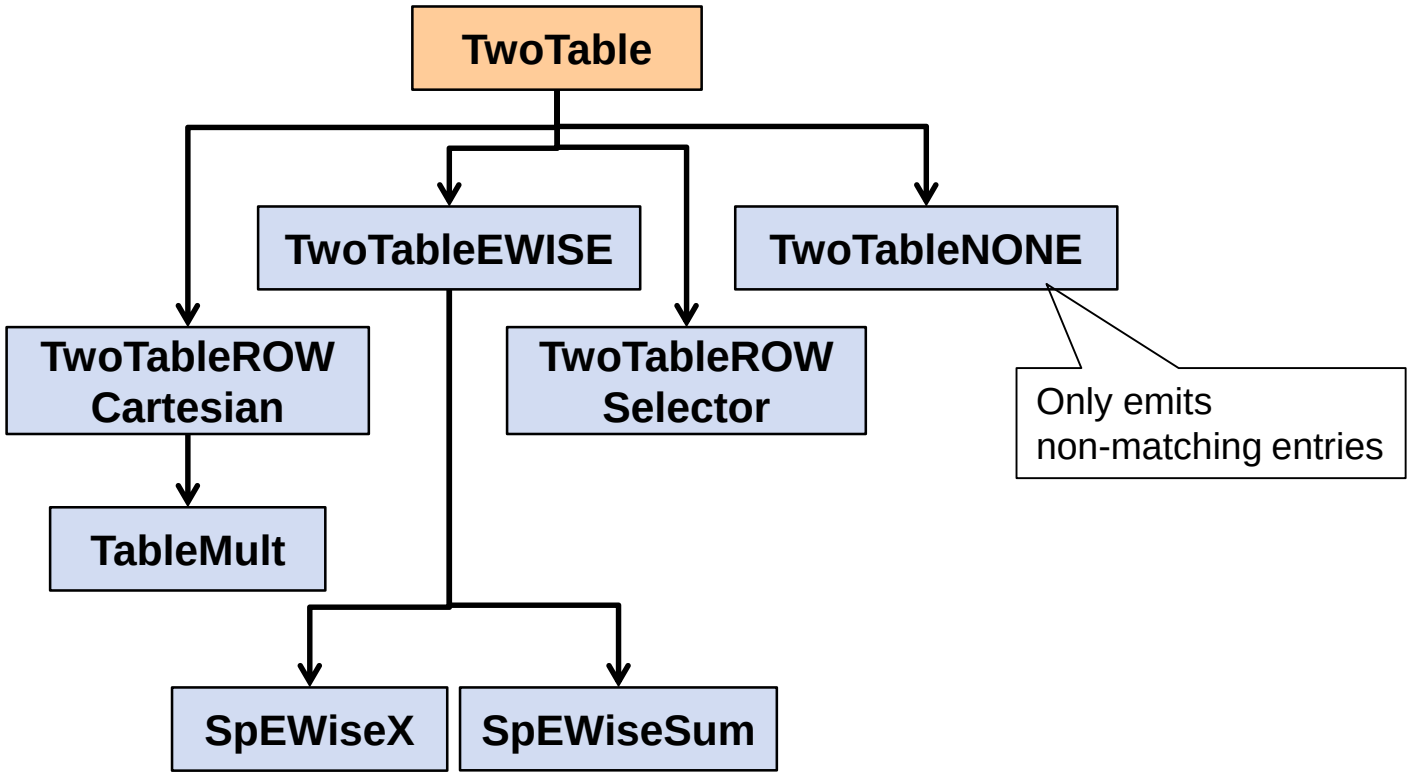
TwoTable: Function Call

```
long TwoTable(String ATtable, String Btable, String Ctable, String CTtable,
              int BScanIteratorPriority,
              TwoTableIterator.DOTMODE dotmode, // ROW, EWISE, NONE
              Map<String, String> optsTT,  // options to setup TwoTableIter.
              IteratorSetting plusOp,      // applied to Ctable, CTtable
              String rowFilter,
              String colFilterAT, String colFilterB,     // D4M Strings
              boolean emitNoMatchA, boolean emitNoMatchB,
              List<IteratorSetting> iteratorsBeforeA,
              List<IteratorSetting> iteratorsBeforeB,
              List<IteratorSetting> iteratorsAfterTwoTable,
              Reducer reducer, Map<String,String> reducerOpts,
              int numEntriesCheckpoint,
              Authorizations ATauth, Authorizations Bauth)
```

- **Options and behavior similar to OneTable()**
- **numEntriesCheckpoint controls how often to send back to client**
  - **Changes B's table.scan.max.memory to 1 byte**
  - **Works but not pretty**

TwoTable: Alias Functions

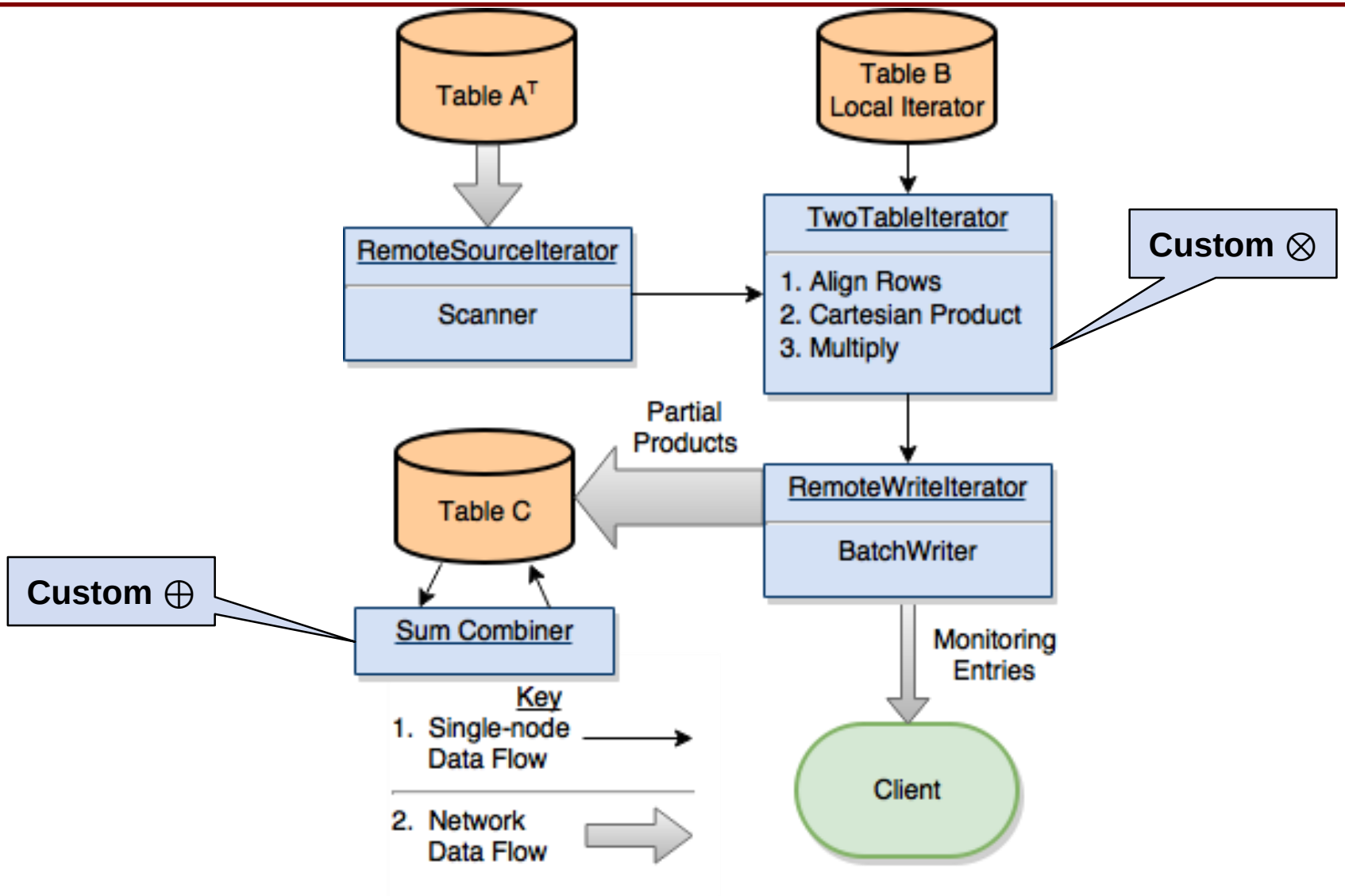
Alias functions call TwoTable, filling in appropriate options



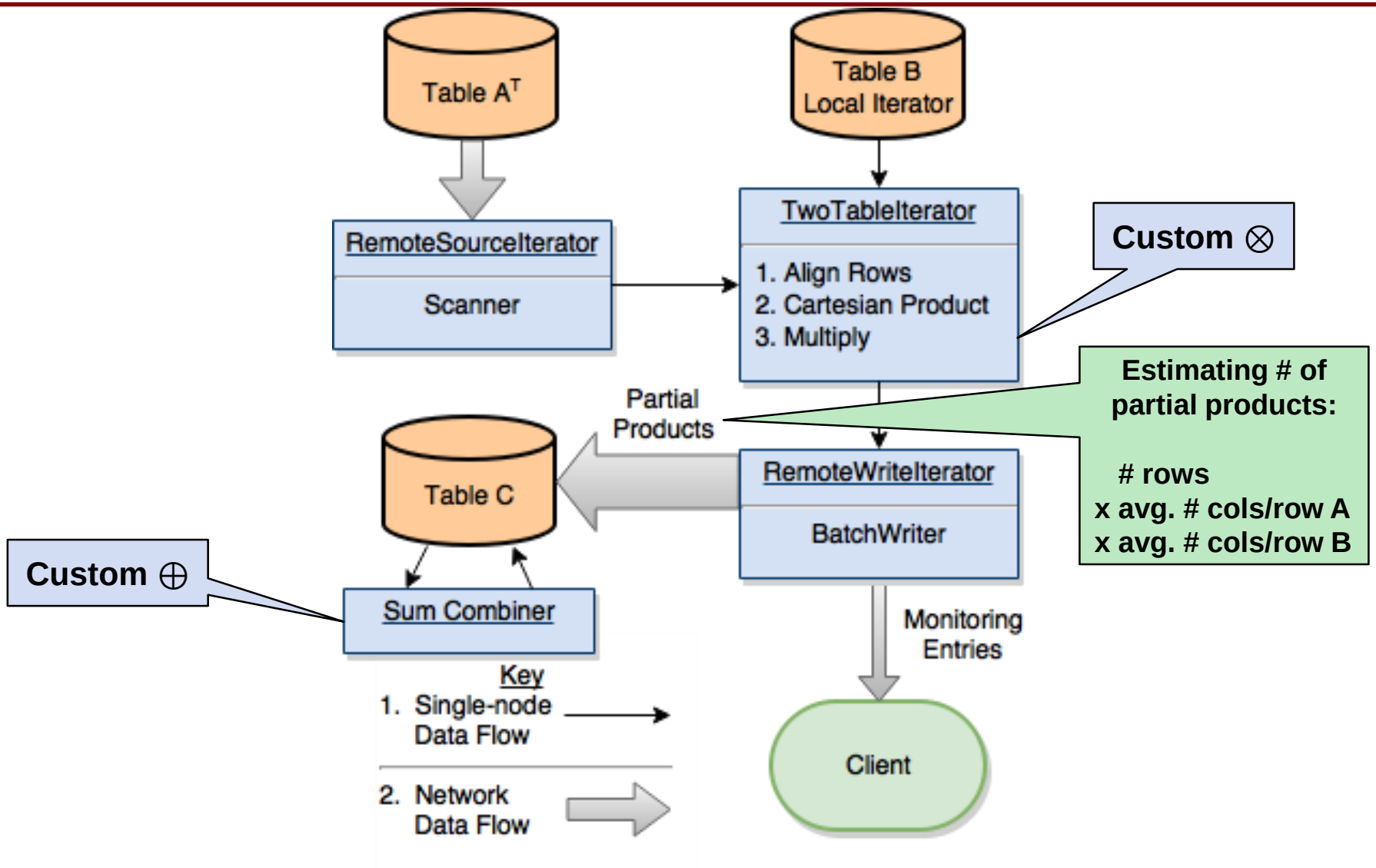
Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- **Graphulo Core Client functions and their Server-side Iterators**
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - **TableMult as TwoTableROW**
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

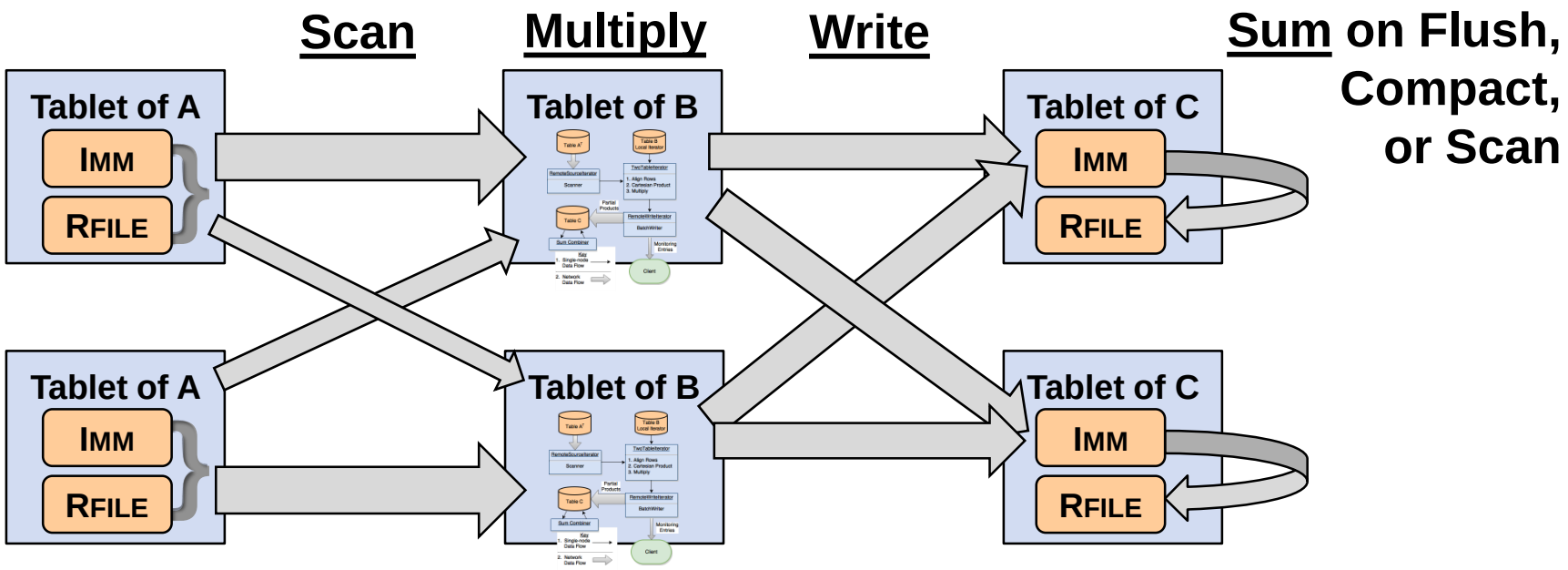
TwoTableROW in TableMult



TwoTableROW in TableMult



TwoTableROW in TableMult: Distributed



- Tablets can be hosted on any tablet server
  - Accumulo load balances tablet allocation
- Matrix multiply iterators run on B's tablets in parallel
  - Scan from A's tablets in parallel
  - BatchWrite to C's tablets in parallel

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- **Graphulo Core Client functions and their Server-side Iterators**
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - **SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar**
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Jack of all Ops: SimpleTwoScalar

- **Simple operations act on Values; no Key manipulation**
- **SimpleTwoScalar interface can stand in for any operation under the Value-only constraint**
  - **Avoids duplicating code for every kind of operation**

```
abstract class SimpleTwoScalar extends Combiner
    implements ApplyOp, MultiplyOp, EWiseOp, Reducer {
  abstract Value multiply(Value Aval, Value Bval);
```

- **MultiplyOp: follows standard matrix multiply result Key**
- **Reducer:**
  - **First update(k,v) stores the given Value**
  - **Subsequent update(k,v) sets** `storedVal = multiply(storedVal, newVal);`
- **ApplyOp: One operand fixed to a constant, given as option**
- **Combiner: Given n Values to combine, runs multiply n-1 times**

Jack of all Math: MathTwoScalar

Common math opts:

```
enum ScalarOp {
  PLUS, TIMES, SET_LEFT, MINUS,
  DIVIDE, POWER, MIN, MAX
}
```

Various String encodings:

```
enum ScalarType {
  LONG, DOUBLE, BIGDECIMAL
}
```

Use static helper methods to create MathTwoScalar IteratorOptions

```
// Reducer Options
MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.DOUBLE, newVisibility, false)
// Combiner:
MathTwoScalar.combinerSetting(6, null, ScalarOp.PLUS, ScalarType.LONG, false)
// Apply constant exponent:
MathTwoScalar.applyOpDouble(1, true, ScalarOp.POWER, 2.0 , false)
```

`false` means don't emit an entry for zero if generated in math, e.g. (-3) + 3 = 0

ConstantTwoScalar class always returns a constant (default "1")

```
ConstantTwoScalar.iteratorSetting(5, new Value("1".getBytes())
```

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- **Algorithms: EdgeBFS**, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

EdgeBFS

- **Incidence table Breadth-First Search**
- **Supports Multi-graph**
  - **Multiple edges between two nodes**
- **Supports Hyper-graph**
  - **Edge between >2 nodes**
- **Implemented as one degree table scan and TableMult per step**
  - **Degree table required for degree filtering**

Reminder of Incidence schema:

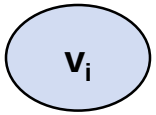
```
00001 :in|907 []  -> 2
00001 :out|23 []  -> 2
00010 :in|769 []  -> 2
00010 :out|643 [] -> 2
00011 :in|419 []  -> 2
00011 :out|545 [] -> 2
00020 :in|67 []   -> 3
00020 :out|262 [] -> 3
```

Degree table:

```
1 :in [] ->  1084
1 :out [] -> 1027
10 :in [] ->  118
10 :out [] ->  94
```

```
String EdgeBFS(String Etable, String v0, int k, String Rtable, String RTtable,
               String startPrefixes, String endPrefixes, String ETDegtable,
               String degColumn, boolean degInColQ, int minDegree, int maxDegree,
               IteratorSetting plusOp, int EScanIteratorPriority,
               Authorizations Eauth, Authorizations EDegauth, String newVisibility,
               boolean outputUnion, MutableLong numEntriesWritten)
```

EdgeBFS: Design

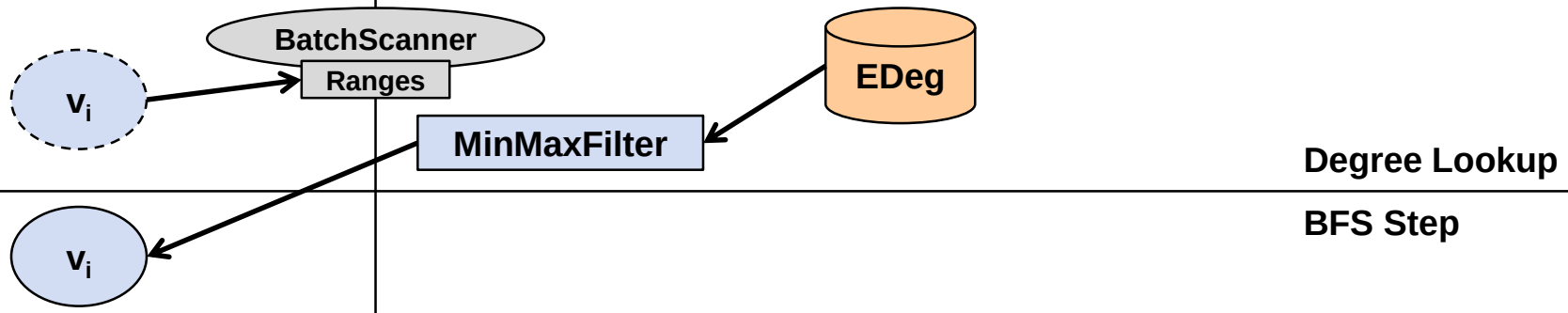


Degree Lookup

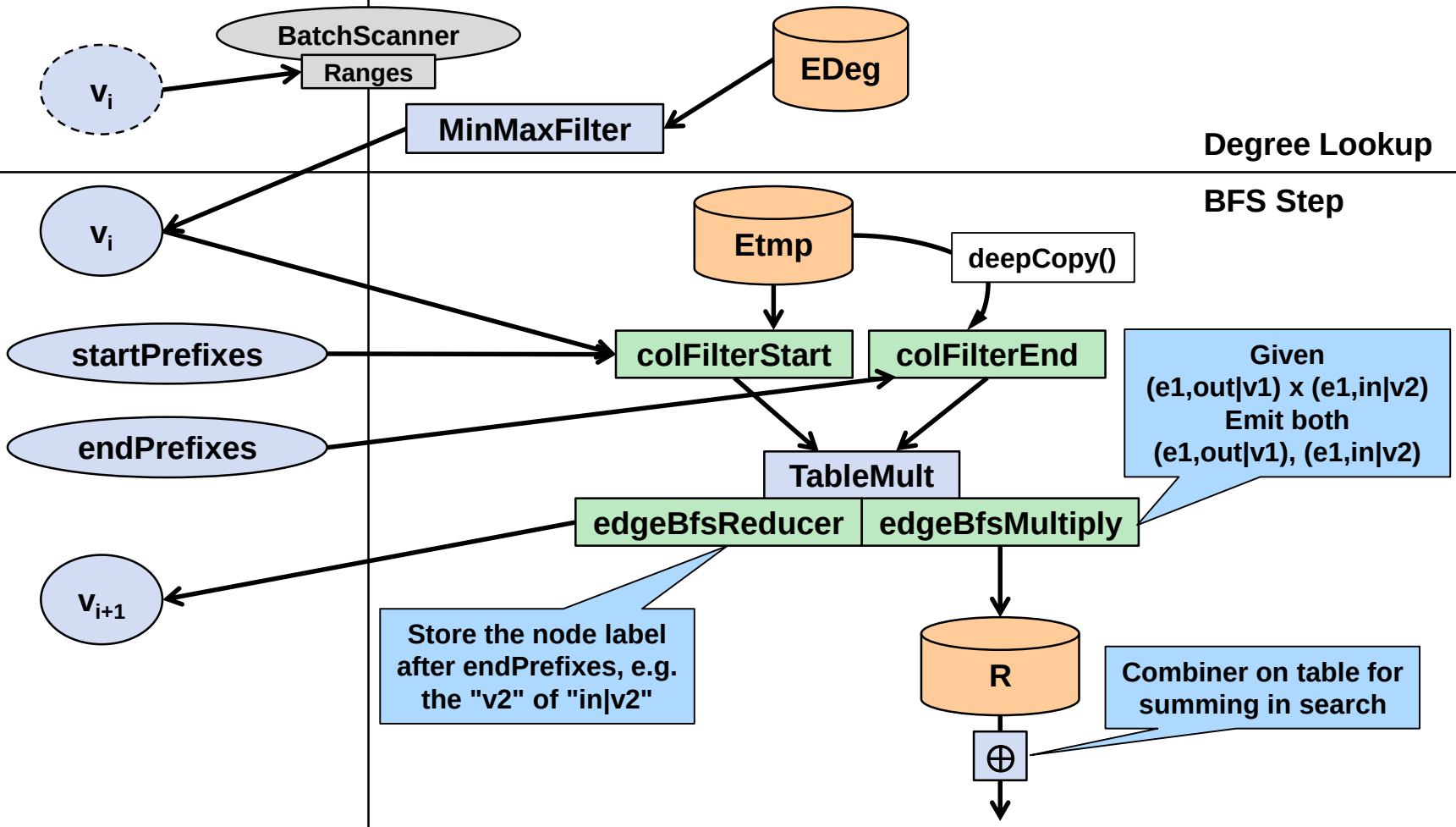
BFS Step

Client | Server

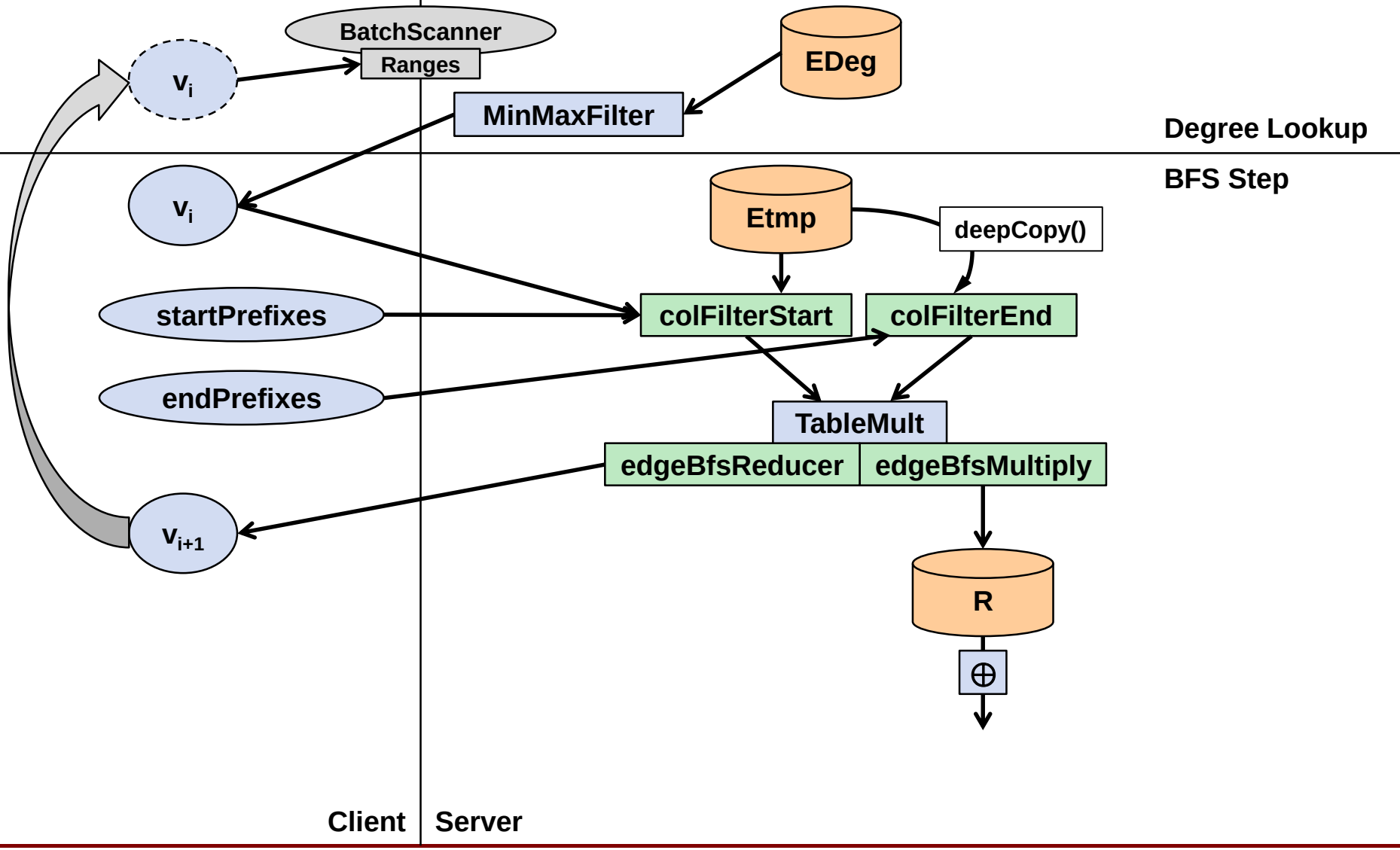
EdgeBFS: Design



EdgeBFS: Design



EdgeBFS: Design



EdgeBFSMultiply and EdgeBFSReducer

EdgeBFSReducer

```
private String findNodeAfter(byte[] cqBytes) {
  // sequential search: try every inColumnPrefix.
  for (byte[] inColumnPrefix : inColumnPrefixes) {
    String nodeAfter = GraphuloUtil.stringAfter(inColumnPrefix, cqBytes);
    if (nodeAfter != null)
      return nodeAfter;
  }
  return null;
}
public void update(Key k, Value v) {
  String nodeAfter = findNodeAfter(k.getColumnQualifierData()
                                   .getBackingArray());
  if (nodeAfter != null)
    setNodesReached.add(nodeAfter);
}
```

Only stores node names after an endPrefix, e.g. "in|"

EdgeBFSMultiply

```
Iterator<? extends Map.Entry<Key, Value>> multiply(ByteSequence Mrow, ByteSequence ATcolF,
    ByteSequence ATcolQ, ByteSequence ATcolVis, ByteSequence BcolF, ByteSequence BcolQ,
    ByteSequence BcolVis, Value ATval, Value Bval) {
  emitKeyFirst = new Key(Mrow.getBackingArray(), ATcolF.getBackingArray(), ATcolQ.getBackingArray(),
      useNewVisibility ? newVisibility : ATcolVis.getBackingArray(), System.currentTimeMillis());
  emitKeySecond = new Key(Mrow.getBackingArray(), BcolF.getBackingArray(), BcolQ.getBackingArray(),
      useNewVisibility ? newVisibility : BcolVis.getBackingArray(), System.currentTimeMillis());
  emitValueFirst = new Value(ATval);
  emitValueSecond = new Value(Bval);
  return this;
}
```

Logic: A x B = {A, B}
Works because column filtering sets A to startPrefixes and B to endPrefixes

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- **Algorithms:** EdgeBFS, **SingleBFS**, Jaccard, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

SingleBFS

- **Single-table schema Breadth-First Search**
- **Similar to AdjBFS**
  - **Degree scan**
  - **Edge scan**
- **Iterator creates transpose entries**
  - **Found edge "vIn|vOut" ➔ emit both "vIn|vOut" and "vOut|vIn"**
  - **Maintains undirected-ness**
- **Reducer gathers reached nodes**
  - **Trick: timestamp parity used to mark reached nodes vs. starting nodes for Reducer**

Degrees assumed to be out-degrees

Schema

```
1010      :deg []  -> 3
1010|933  :edge [] -> 3
1011      :deg []  -> 3
1011|2    :edge [] -> 3
1012      :deg []  -> 3
1012|270  :edge [] -> 3
1013      :deg []  -> 6
1013|163  :edge [] -> 3
1013|74   :edge [] -> 3
1015      :deg []  -> 3
1015|37   :edge [] -> 3
```

Other variations possible like degInColQ, SDegtable– change the code & signature for your use case

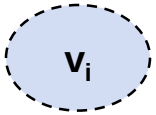
```
String SingleBFS(String Stable, String edgeColumn, char edgeSep,
                 String v0, int k, String Rtable, String SDegtable, String degColumn,
                 boolean copyOutDegrees, boolean computeInDegrees,
                 ScalarType degSumType, ColumnVisibility newVisibility,
                 int minDegree, int maxDegree, IteratorSetting plusOp,
                 boolean outputUnion, Authorizations Sauth,
                                      MutableLong numEntriesWritten)
```

Adding degrees to result table optional

Degrees of reached nodes requires extra scan/write step

Return union of all nodes reached instead of nodes reached in exactly k steps

SingleBFS: Design



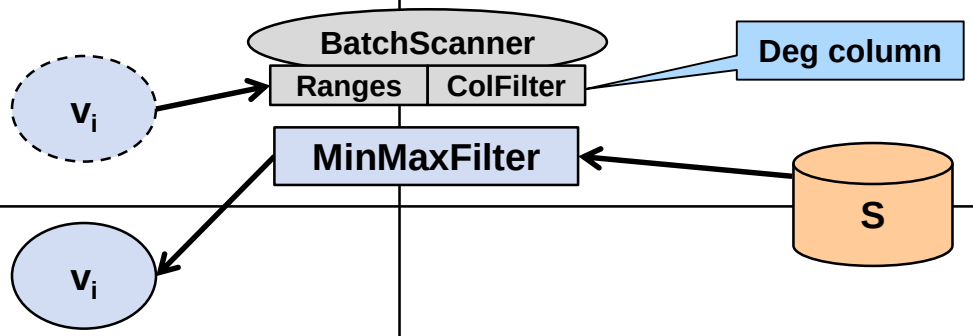
Degree Lookup

BFS Step

computeInDegrees Extension

Client | Server

SingleBFS: Design



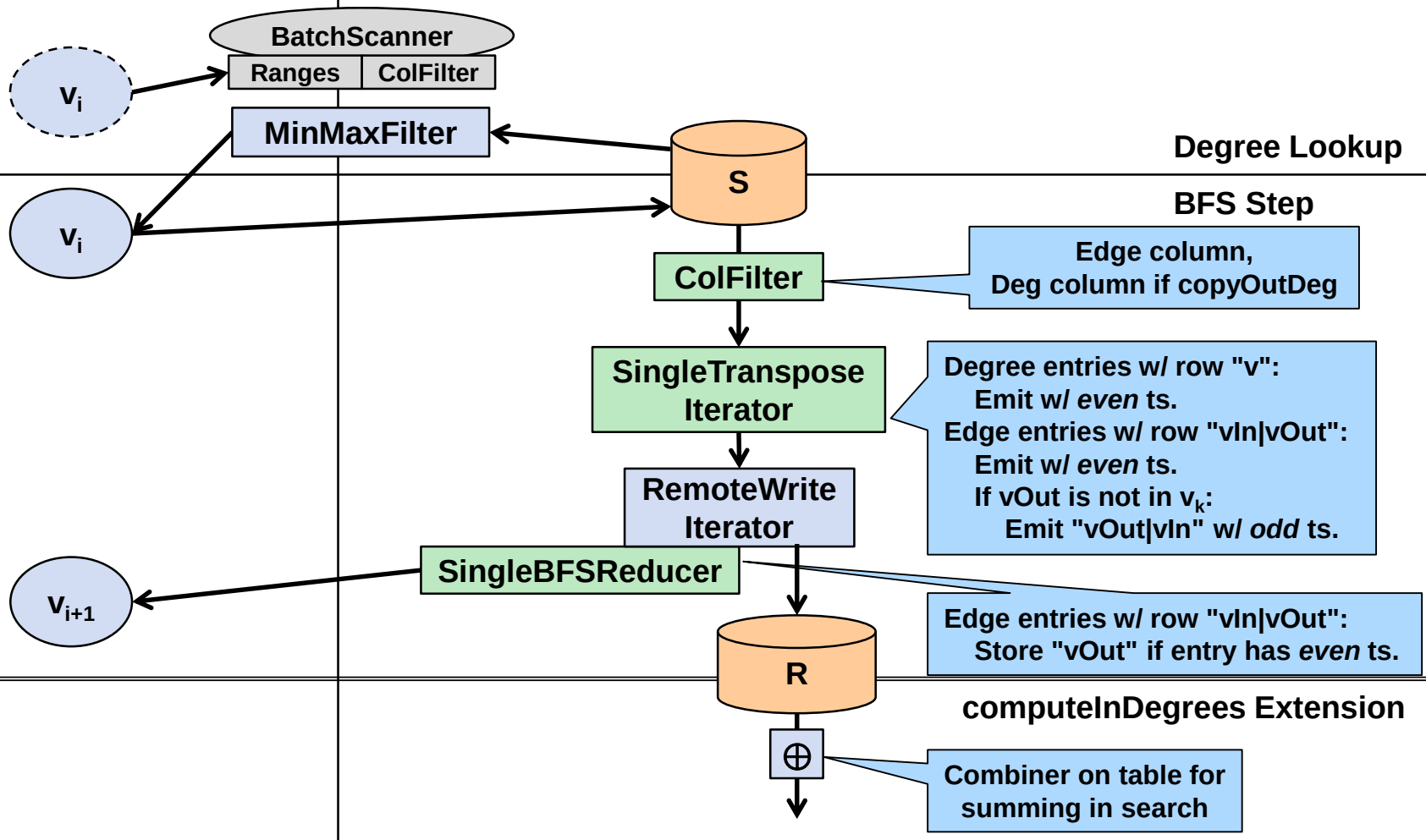
Degree Lookup

BFS Step

computeInDegrees Extension

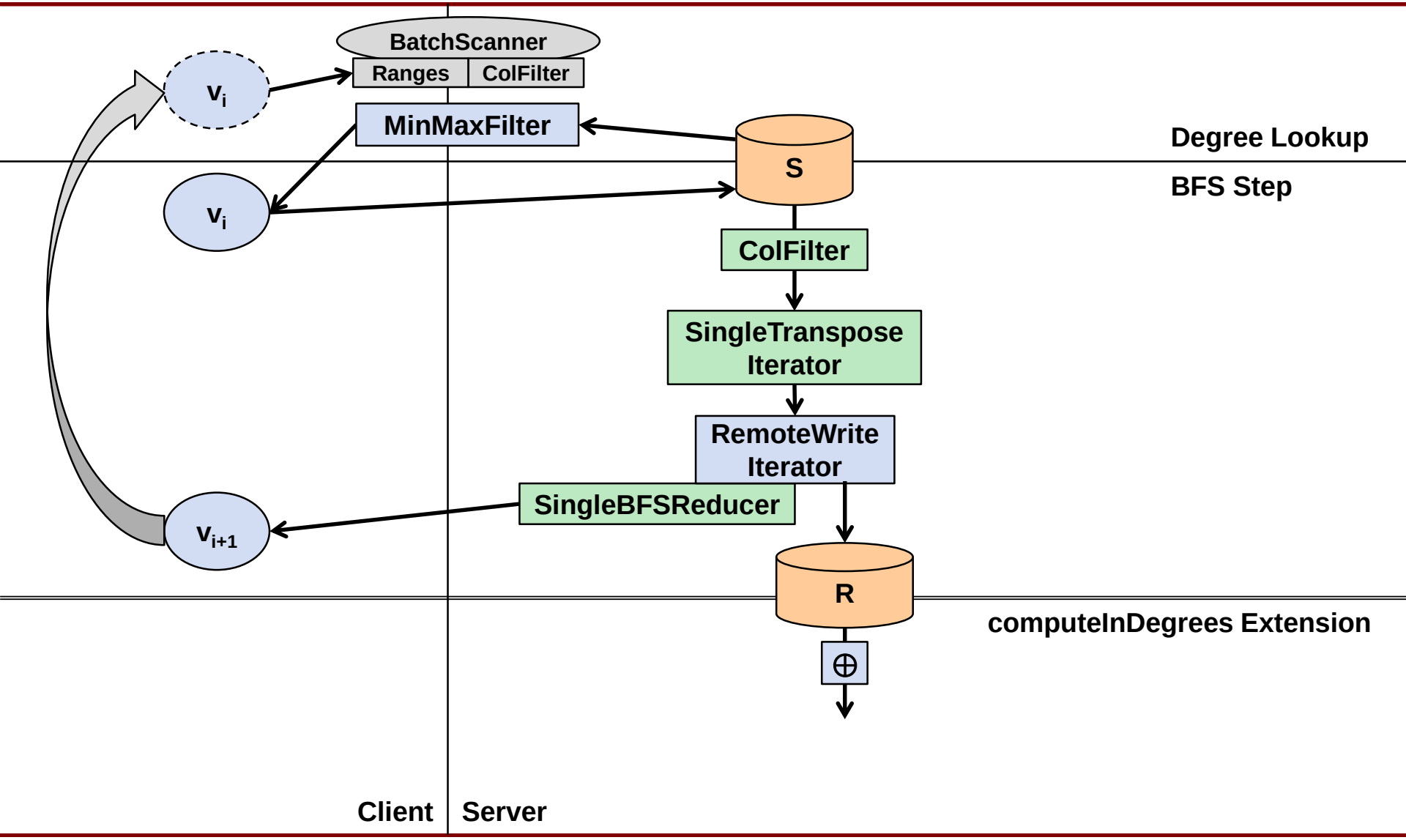
Client | Server

SingleBFS: Design

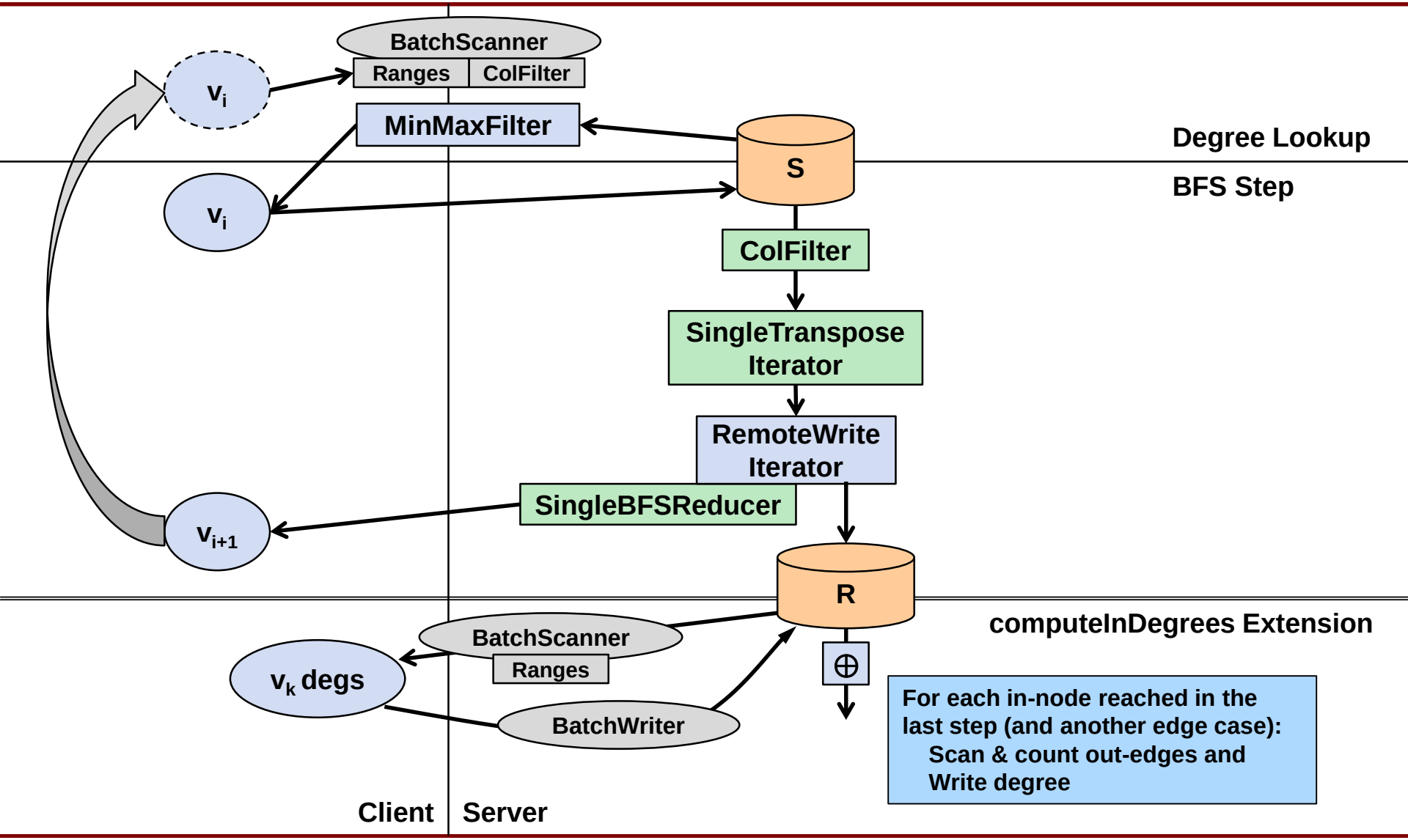


ts: Timestamp portion of Key

SingleBFS: Design



SingleBFS: Design



SingleBFS: SingleTransposeIterator & SingleBFSReducer

```
void prepNext(boolean doNext) throws IOException {
  // …topKey/topValue manipulation code omitted…
  if (!source.hasTop())
    return;
  topKey1 = source.getTopKey();
  topValue1 = source.getTopValue();
  Text rowText = topKey1.getRow();
  String rStr = rowText.toString();
  int pos = rStr.indexOf(edgeSep);
  if (pos == -1) return; // return if degree row

  long ts = topKey1.getTimestamp();
  long tsEven = ts % 2 == 0 ? ts : ts-1;
  topKey1 = new Key(rowText,
      topKey1.getColumnFamily(),
      topKey1.getColumnQualifier(),
      topKey1.getColumnVisibility(), tsEven);

  String toNode = rStr.substring(pos+1);
  if (!isInStartNodes(toNode)) {
    long tsOdd = tsEven+1;
    String fromNode = rStr.substring(0,pos);
    topKey2 = new Key(
        new Text(toNode+edgeSep+fromNode),
        topKey1.getColumnFamily(),
        topKey1.getColumnQualifier(),
        topKey1.getColumnVisibility(), tsOdd);
    topValue2 = topValue1;
} }
```

from SingleTransposeIterator

Degree entries w/ row "v":
Emit w/ *even* ts.
Edge entries w/ row "vIn|vOut":
Emit w/ *even* ts.
If vOut is not in v_k:
Emit "vOut|vIn" w/ *odd* ts.

from SingleBFSReducer

```
void update(Key k, Value v) {
 // Signal from SingleTransposeIterator
 if (k.getTimestamp() % 2 != 0)
   return;
 ByteSequence rd = k.getRowData();
 String rs = new String(rd.getBackingArray(),
                 rd.offset(), rd.length());
 int pos = rs.indexOf(edgeSep);
 if (pos == -1)
   return; // return if degree row
 String toNode = rs.substring(pos+1);
 setNodesReached.add(toNode);
}
```

Edge entries w/ row "vIn|vOut":
Store "vOut" if entry has *even* ts.

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- **Algorithms:** EdgeBFS, SingleBFS, **Jaccard**, kTrussAdj, kTrussEdge
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Jaccard Coefficients

- **Measures neighborhood overlap of two vertices**
- **Input: unweighted, undirected adjacency matrix**
- **Expressed as (for vertices v_i and v_j), where *N* is the neighbor function:**

$$J_{ij} = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

- **See mathematics:**
  - V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner, "Graphulo: Linear algebra graph kernels for NoSQL databases," in International Parallel & Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2015.

```
long Jaccard(String Aorig, String ADeg, String Rfinal,
             String filterRowCol, Authorizations Aauth,
             String RNewVisibility)
```

- **Implemented as a single TableMult**

Demo Jaccard

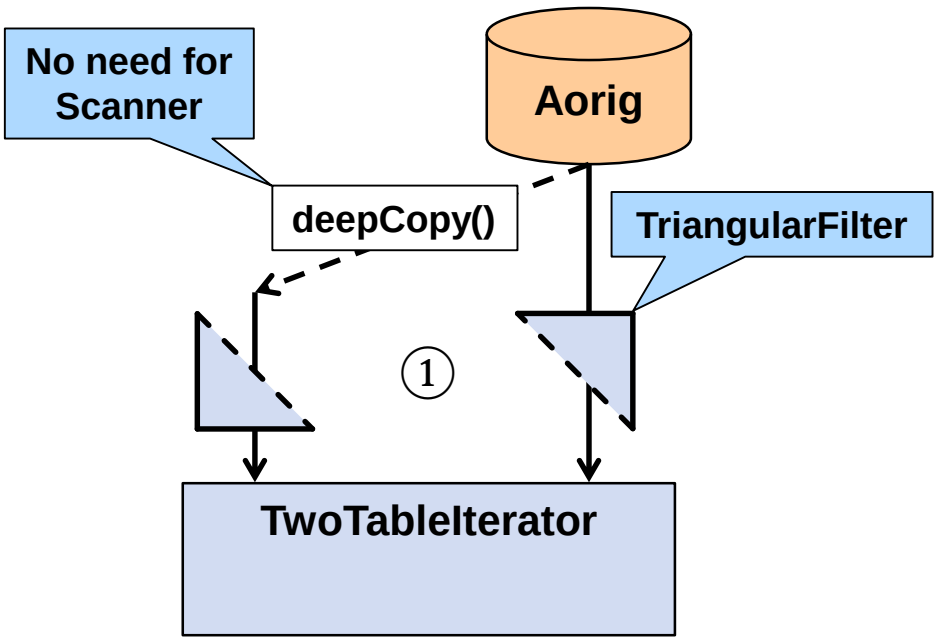
```
dhutchis@dmasterBW:graphulo$ mvn test -Dtest=JaccardExample -DTEST_CONFIG=local
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building graphulo 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
```

```
dhutchis@dmasterBW:graphulo$ cat shippable/testresults/edu.mit.ll.graphulo.examples.JaccardExample-output.txt
10 Aug 2015 21:14:50,240 WARN - ClientConfiguration.loadFromSearchPath(227) -  Found no client.conf in default paths. Using default cl
10 Aug 2015 21:14:50,437 DEBUG - RealAccumuloTester.before(52) -  setUp ok - ClientConfiguration=org.apache.accumulo.core.client.Clien
10 Aug 2015 21:14:53,715 INFO - ExampleUtil.ingestAdjacencySCALE(35) -  Wrote 16384 edges to D4M Adjacency tables with base name ex10A
10 Aug 2015 21:14:54,441 DEBUG - Graphulo.OneTable(827) -  8 :%00; [] 9223372036854775807 false -> 98 entries processed
10 Aug 2015 21:14:54,442 INFO - JaccardExample.exampleJaccard(82) -  Nodes reached from v0: 801,281,195,197,34,37,193,641,531,389,199,
69,139,17,314,18,15,13,513,453,21,175,419,65,721,554,518,131,514,97,657,145,813,93,149,293,325,495,521,523,354,297,258,257,259,425,50,
10 Aug 2015 21:14:54,450 INFO - JaccardExample.exampleJaccard(83) -  Does AtableSub exist? true
10 Aug 2015 21:14:55,421 DEBUG - Graphulo.TwoTable(612) -  :%00; [] 9223372036854775807 false -> 1123 entries processed
10 Aug 2015 21:14:55,425 DEBUG - Graphulo.Jaccard(2083) -  Jaccard #partial products 1123
10 Aug 2015 21:14:55,426 INFO - JaccardExample.exampleJaccard(87) -  Number of partial products sent to result table: 1123
10 Aug 2015 21:14:55,646 INFO - JaccardExample.exampleJaccard(113) -  Jaccard min: 7.342143906020558E-4
10 Aug 2015 21:14:55,647 INFO - JaccardExample.exampleJaccard(114) -  Jaccard max: 0.5
10 Aug 2015 21:14:55,652 INFO - JaccardExample.exampleJaccard(115) -  Jaccard sum: 12.157391112336697
10 Aug 2015 21:14:55,652 INFO - JaccardExample.exampleJaccard(116) -  Jaccard cnt: 1025.0
10 Aug 2015 21:14:55,652 INFO - JaccardExample.exampleJaccard(117) -  Jaccard avg: 0.01186086937788946
```

```
root@instance ex10J> scan
1 :129 []    0.0014641288433382138
1 :13 []    8.944543828264759E-4
1 :131 []    8.771929824561404E-4
1 :139 []    9.523809523809524E-4
1 :145 []    8.857395925597874E-4
1 :149 []    9.416195856873823E-4
1 :15 []    0.001899335232668566
1 :161 []    8.726003490401396E-4
1 :17 []    0.0014598540145985
1 :175 []    9.737098344693282E-4
1 :18 []    8.77963125548727E-4
```

```
175 :199 []    0.0625
175 :21 []    0.011494252873563218
175 :259 []    0.010309278350515464
175 :297 []    0.021739130434782608
175 :3 []    0.0030120481927710845
175 :34 []    0.008849557522123894
175 :354 []    0.1
175 :387 []    0.03333333333333333
175 :419 []    0.125
175 :425 []    0.07142857142857142
175 :5 []    0.0030581039755351682
175 :531 []    0.025
```

Jaccard: Design



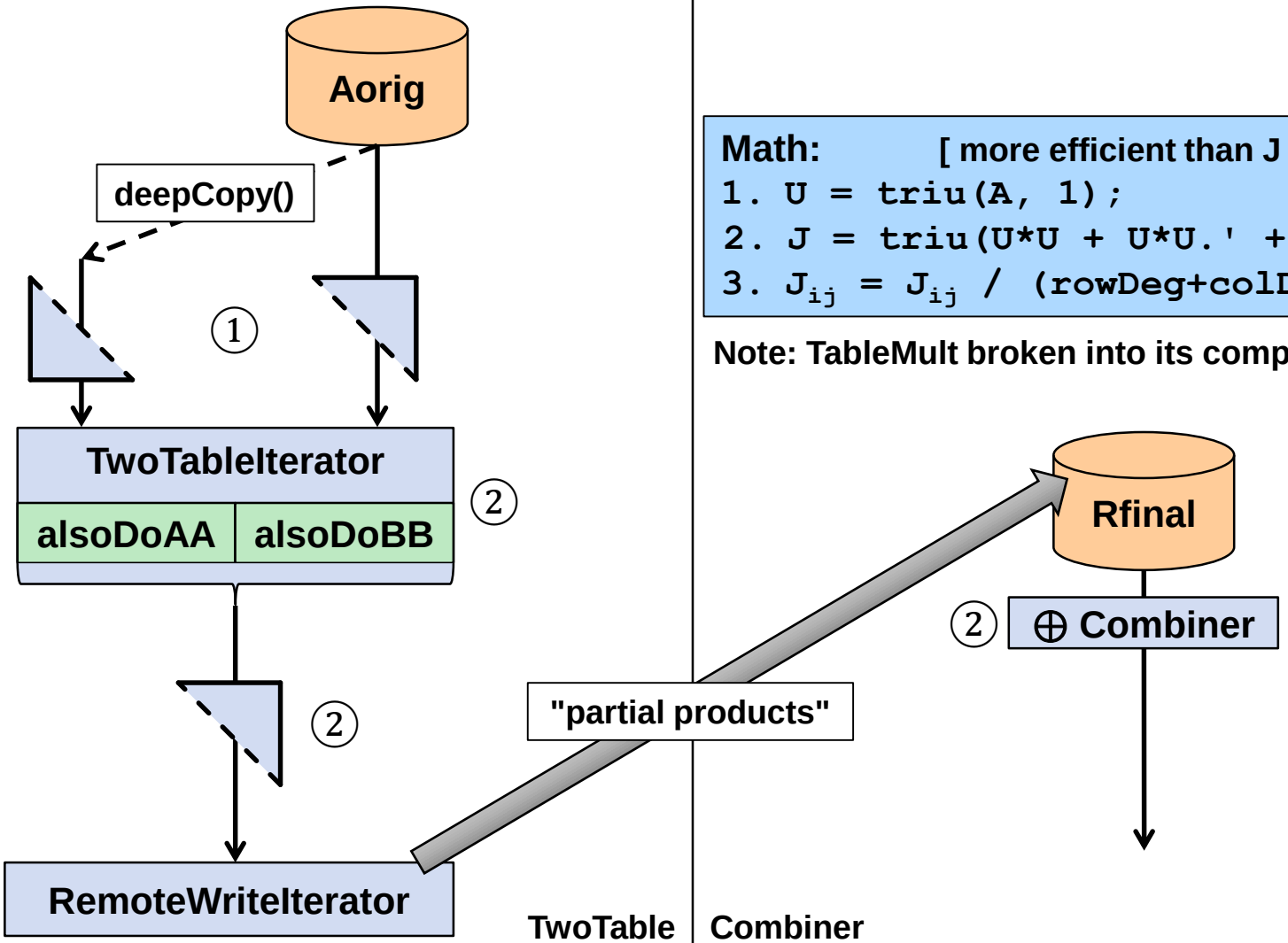
Math: [more efficient than J = triu(A*A, 1)]

1. `U = triu(A, 1);`
2. `J = triu(U*U + U*U.' + U.'*U,1);`
3. $J_{ij} = J_{ij} / (rowDeg+colDeg-J_{ij});$

Note: TableMult broken into its component iterators

TwoTable scan on Aorig

Jaccard: Design

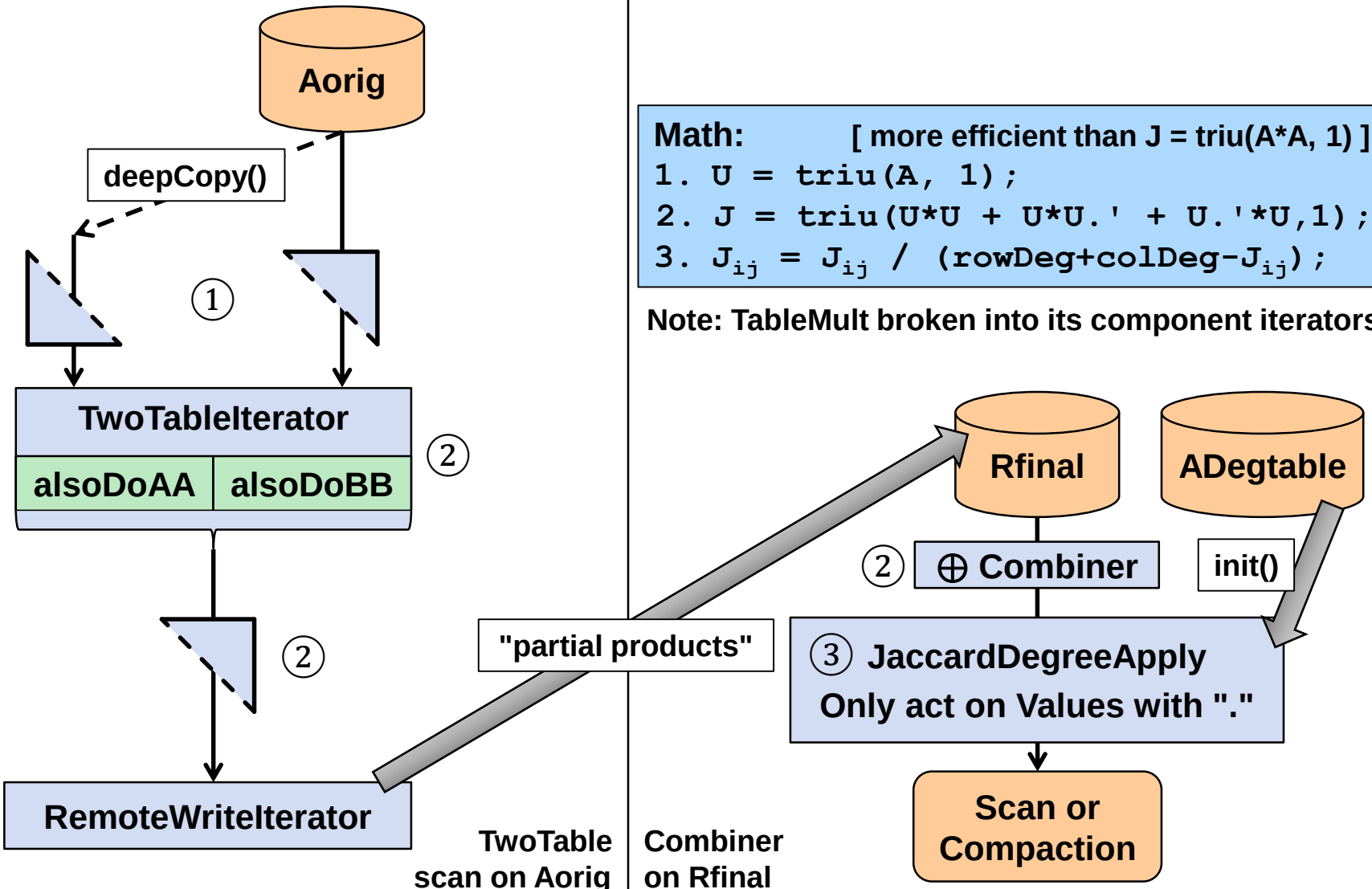


Math: [more efficient than J = triu(A*A, 1)]

```
1. U = triu(A, 1);
2. J = triu(U*U + U*U.' + U.'*U,1);
3. J_ij = J_ij / (rowDeg+colDeg-J_ij);
```

Note: TableMult broken into its component iterators

Jaccard: Design



Math: [more efficient than J = triu(A*A, 1)]

```
1. U = triu(A, 1);
2. J = triu(U*U + U*U.' + U.'*U,1);
3. J_ij = J_ij / (rowDeg+colDeg-J_ij);
```

Note: TableMult broken into its component iterators

Jaccard: Implementation

```
import static TriangularFilter.TriangularType;

long Jaccard(String Aorig, String ADeg, String Rfinal, String filterRowCol,
             Authorizations Aauthorizations, String RNewVisibility){
  // "Plus" iterator to set on Rfinal
  IteratorSetting RPlusIteratorSetting = new DynamicIteratorSetting(6, null)
      .append(MathTwoScalar.combinerSetting(1, null, ScalarOp.PLUS, ScalarType.LONG, false))
      .append(JaccardDegreeApply.iteratorSetting(1,
          basicRemoteOpts(ApplyIterator.APPLYOP + ApplyIterator.OPT_SUFFIX, ADeg)))
      .toIteratorSetting();


  // Use deepCopy of local iterator on A for left part of the TwoTable
  long npp = TableMult(TwoTableIterator.CLONESOURCE_TABLENAME, Aorig, Rfinal, null, -1,
      MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.LONG,
                                                   RNewVisibility, false),
      RPlusIteratorSetting,
      filterRowCol == null ? null : GraphuloUtil.d4mRowToRanges(filterRowCol),
      filterRowCol, filterRowCol,
      true, true,
      Collections.singletonList(TriangularFilter.iteratorSetting(1, TriangularType.Lower)),
      Collections.singletonList(TriangularFilter.iteratorSetting(1, TriangularType.Upper)),
      Collections.singletonList(TriangularFilter.iteratorSetting(1, TriangularType.Upper)),
      null, null, -1, Aauth, Aauth);

  log.debug("Jaccard #partial products " + npp);
  return npp;
}
```

A's Values assumed all "1"s (A is unweighted)

Filter on rows and cols (A is undirected)

⊕ Combiner

Degree table conn. params

Don't write transpose, default scan priority

alsoDoAA, alsoDoBB

iters. BeforeA, BeforeB, AfterTT

Jaccard: TriangularFilter

```
enum TriangularType {Upper, UpperDiagonal,
  Lower,LowerDiagonal,Diagonal,NoDiagonal}

public boolean accept(Key k, Value v) {
  int cmp = k.getRowData().compareTo(
      k.getColumnQualifierData());
  switch (triangularType) {
    case Upper:          return cmp <  0;
    case UpperDiagonal:  return cmp <= 0;
    case Lower:          return cmp >  0;
    case LowerDiagonal:  return cmp >= 0;
    case Diagonal:       return cmp == 0;
    case NoDiagonal:     return cmp != 0;
    default: throw new AssertionError();
  }
}
```

- **Ordinary SKVIs usable in Graphulo**
  - **Easier but not necessary to use ApplyIterator for simple functions**
  - **Think of ApplyIterator as a development shortcut**
    - **Filter is also a shortcut**
- **Options passed to init()**
  - **Enables setting iterator at runtime**
  - **Reduces code duplication**
- **Static method constructs IteratorOptions**
  - **Developers don't have to remember option names**

```
static IteratorSetting iteratorSetting(int priority, TriangularType type) {
  IteratorSetting itset = new IteratorSetting(priority,TriangularFilter.class);
  itset.addOption(TRIANGULAR_TYPE, type.name());
  return itset;
}
static final String TRIANGULAR_TYPE = "triangularType";
void init(SortedKeyValueIterator<Key,Value> source, Map<String,String> opts,
          IteratorEnvironment env) throws IOException {
  super.init(source, opts, env); // initializes NEGATE
  if (opts.containsKey(TRIANGULAR_TYPE))
    triangularType = TriangularType.valueOf(opts.get(TRIANGULAR_TYPE));
}
```

JaccardDegreeApply

```
void init(Map<String,String> opts, IteratorEnvironment env)throws IOException{
  remoteDegTable = new RemoteSourceIterator();
  remoteDegTable.init(null, opts, env);
  degMap = new HashMap<>();
  scanDegreeTable();
}
private void scanDegreeTable() throws IOException {
  remoteDegTable.seek(new Range(),Collections.<ByteSequence>emptySet(),false);
  Text rowHolder = new Text();
  while (remoteDegTable.hasTop()) {
    degMap.put(remoteDegTable.getTopKey().getRow(rowHolder).toString(),
        Double.valueOf(remoteDegTable.getTopValue().toString()));
    remoteDegTable.next();
  }
}
```

Load degree table into memory: Map: Node String -> Degree

More advanced solution would use seekApplyOp() to only load the part of the degree table needed for this tablet of R, i.e., (seekRangeStartKey, +inf)

```
Iterator<? extends Map.Entry<Key, Value>> apply(final Key k, Value v) {
  // Period indicates already processed Double Value; no period unprocessed Long Value
  String vstr = v.toString();
  if (vstr.contains("."))
    return Iterators.singletonIterator(new AbstractMap.SimpleImmutableEntry<>(k, v));
  String row = k.getRow().toString(), col = k.getColumnQualifier().toString();
  double rowDeg = degMap.get(row), colDeg = degMap.get(col);
  double Jij = Long.parseLong(vstr);
  return Iterators.singletonIterator( new AbstractMap.SimpleImmutableEntry<>(k,
      new Value(Double.toString(Jij / (rowDeg+colDeg-Jij)).getBytes())
  ));
}
```

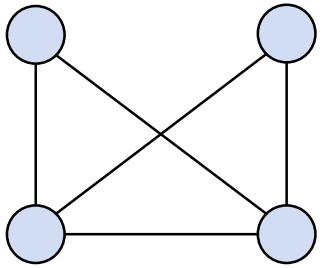
Create idempotence

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- **Algorithms:** EdgeBFS, SingleBFS, Jaccard, **kTrussAdj, kTrussEdge**
- Extensions
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

K-Truss Subgraph

- **A graph is a k-Truss if each edge is part of at least k-2 triangles**
  - **A graph's k-Truss may be the empty graph.**
- **One way to construct graph "core"**



Example 3-truss

- **See mathematics:**
  - V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner, “Graphulo: Linear algebra graph kernels for NoSQL databases,” in International Parallel & Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2015.
- **Two Versions**
  - **Adjacency Table**

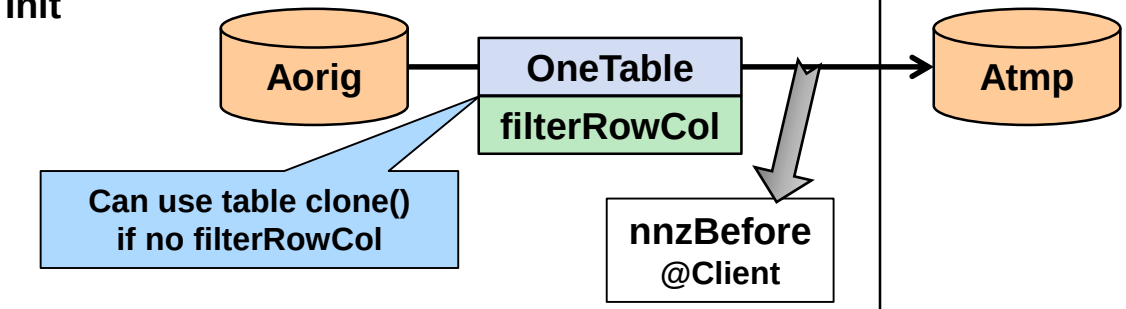
```
long kTrussAdj(String Aorig, String Rfinal, int k, String filterRowCol,
   boolean forceDelete, Authorizations Aauth, String RNewVisibility)
```

  - **Incidence Table**

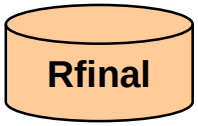
```
long kTrussEdge(String Eorig, String ETorig, String Rfinal, String RTfinal,
   int k, String edgeFilter, boolean forceDelete, Authorizations Eauth)
```

kTrussAdj: Design

Init

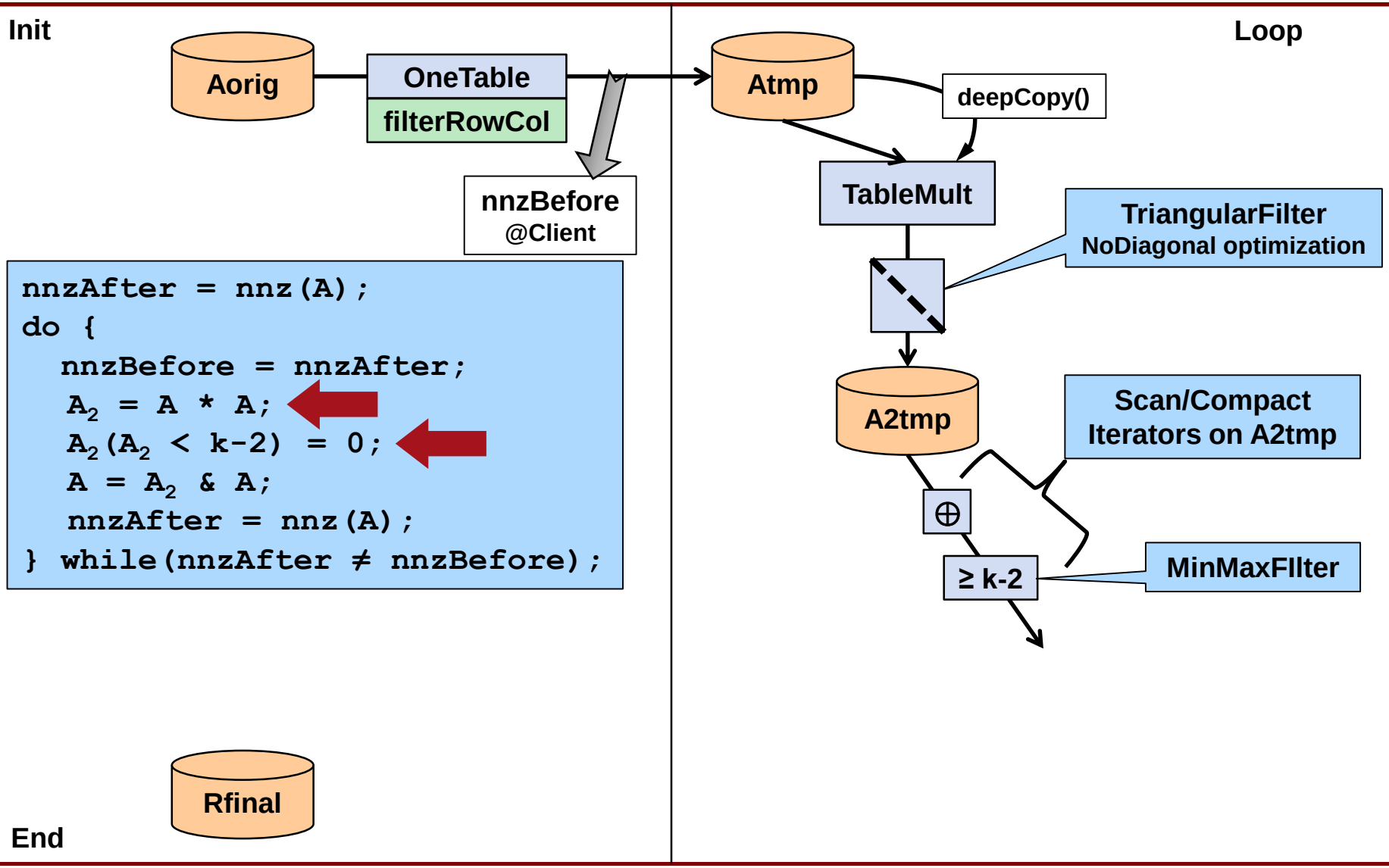



```
nnzAfter = nnz(A);
do {
  nnzBefore = nnzAfter;
  A2 = A * A;
  A2(A2 < k-2) = 0;
  A = A2 & A;
  nnzAfter = nnz(A);
} while(nnzAfter ≠ nnzBefore);
```



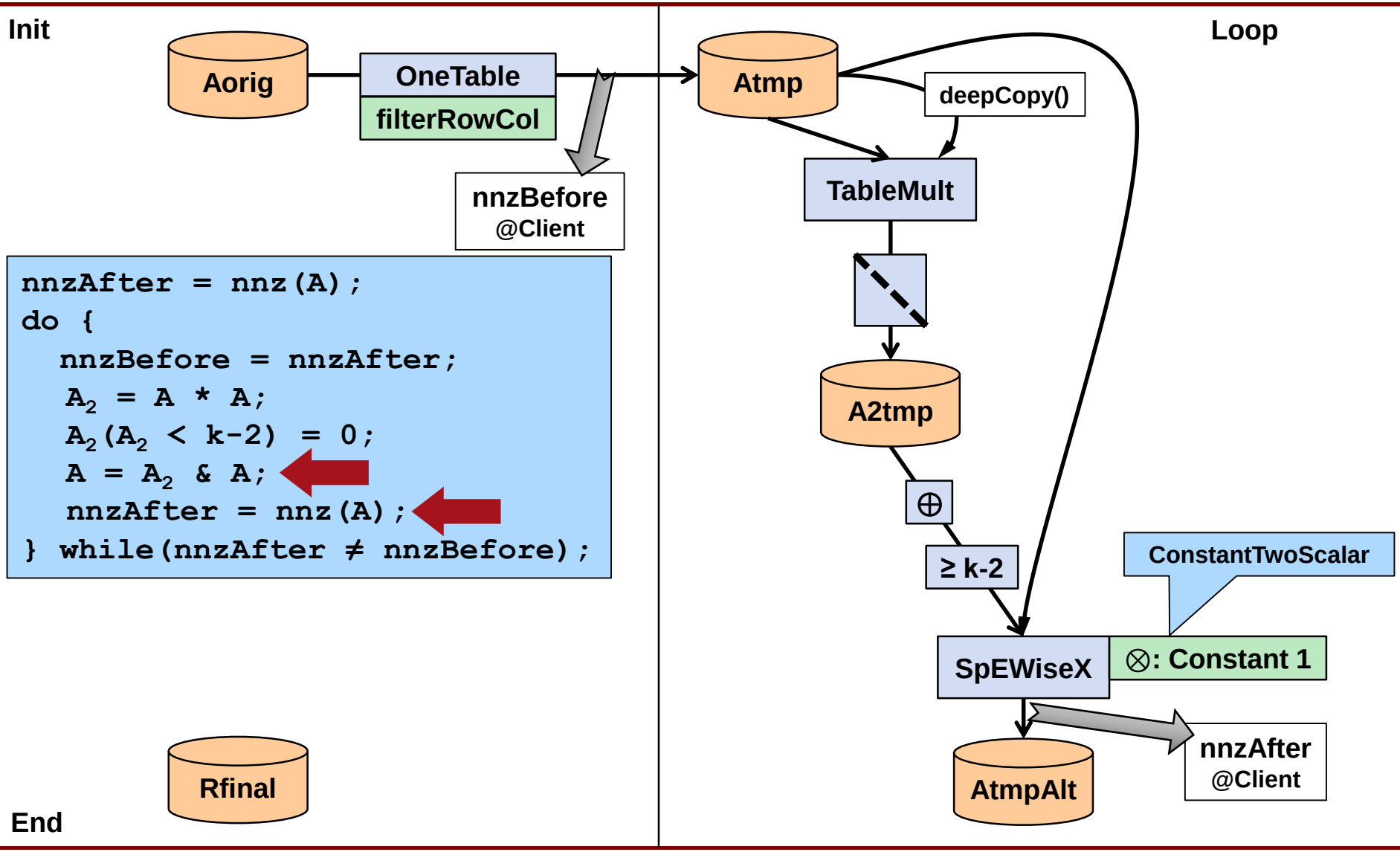
End

kTrussAdj: Design

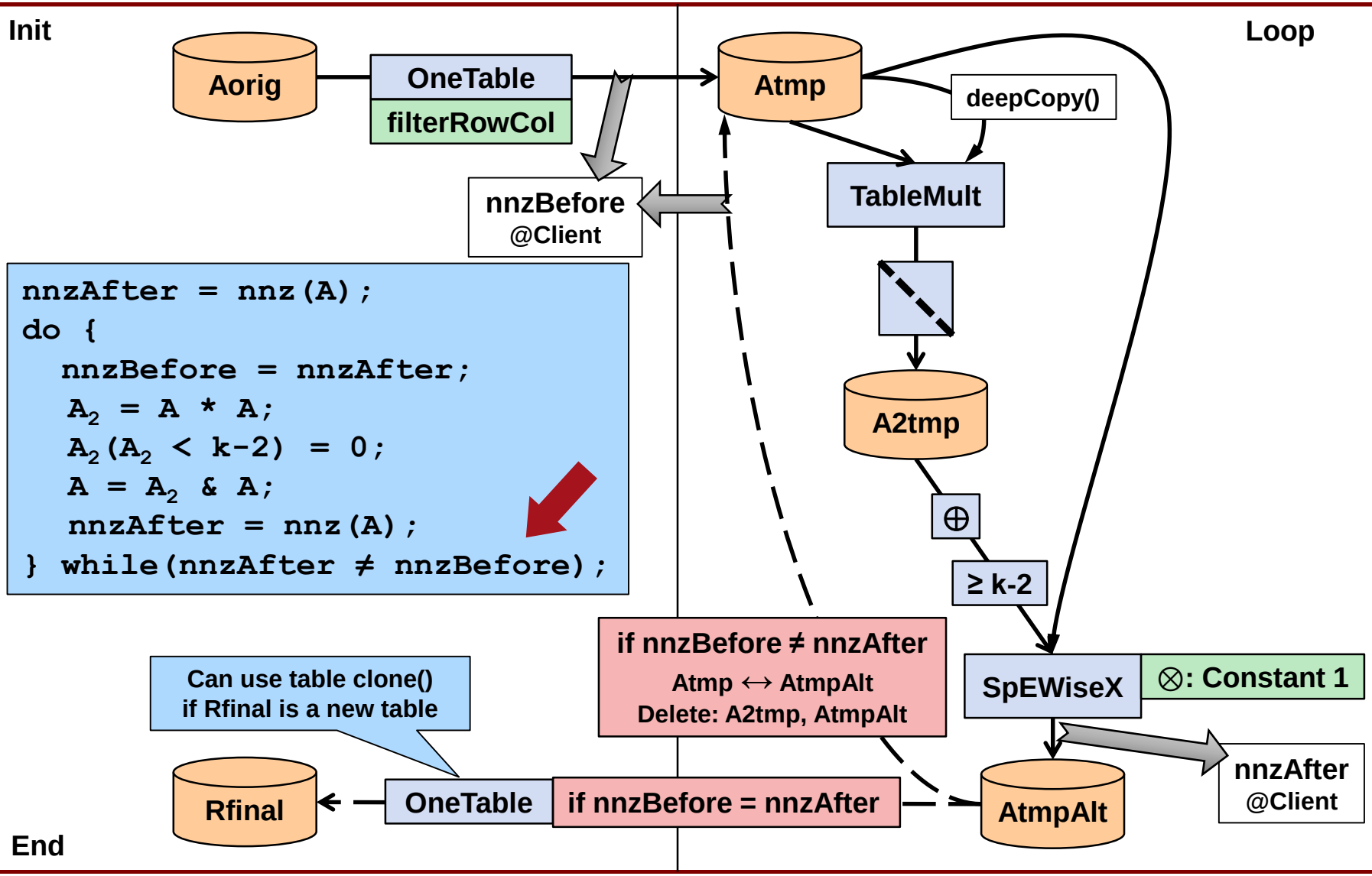



```
nnzAfter = nnz(A);
do {
  nnzBefore = nnzAfter;
  A2 = A * A;
  A2(A2 < k-2) = 0;
  A = A2 & A;
  nnzAfter = nnz(A);
} while(nnzAfter ≠ nnzBefore);
```

kTrussAdj: Design



kTrussAdj: Design



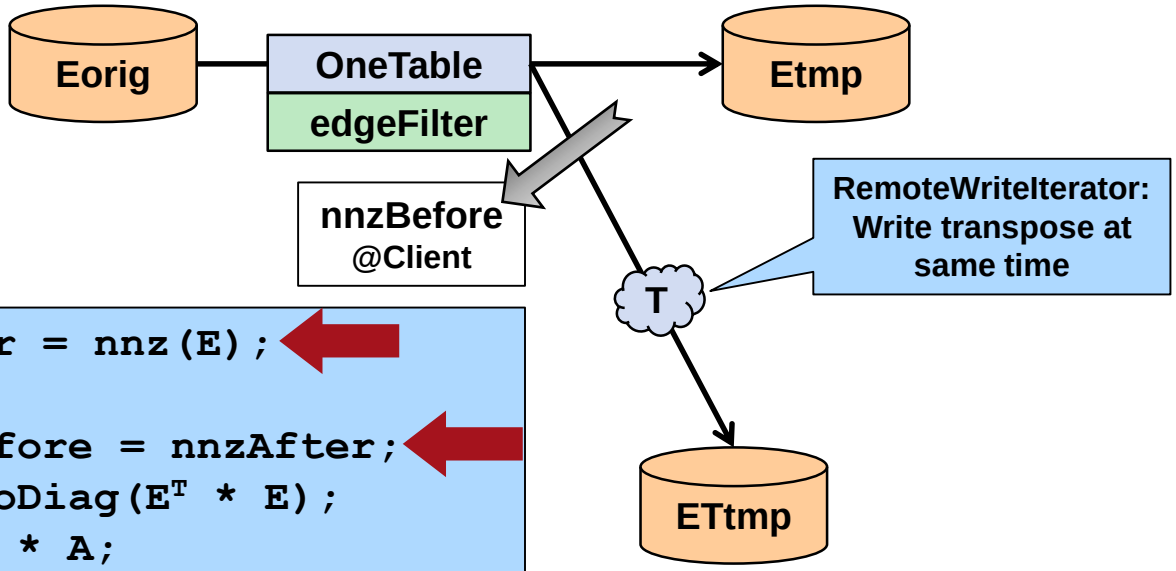
kTrussAdj: Implementation

```
long kTrussAdj(String Aorig, String Rfinal, int k, String filterRowCol,
               boolean forceDelete, Authorizations Aauth, String RNewVisibility) {
  nnzAfter = OneTable(Aorig, Atmp, null, null, -1, null, null, null,
      filterRowCol == null ? null : GraphuloUtil.d4mRowToRanges(filterRowCol),
      filterRowCol, null, null, Aauth);
  IteratorSetting sumAndFilter = new DynamicIteratorSetting(6, null)
      .append(PLUS_ITERATOR_LONG)
      .append(MinMaxFilter.iteratorSetting(10, ScalarType.LONG, k-2, null))
      .toIteratorSetting();
  List<IteratorSetting> noDiagFilter = Collections.singletonList(
      TriangularFilter.iteratorSetting(1, TriangularFilter.TriangularType.NoDiagonal));
  do {
    nnzBefore = nnzAfter;
    TableMult(TwoTableIterator.CLONESOURCE_TABLENAME, Atmp, A2tmp, null, -1,
        ConstantTwoScalar.class, ConstantTwoScalar.optionMap(new Value("1".getBytes()),
                                 RNewVisibility), sumAndFilter, null, null, null,
        false, false, null, null, noDiagFilter, null, null, -1, Aauth, Aauth);

    nnzAfter = SpEWiseX(A2tmp, Atmp, AtmpAlt, null, -1, ConstantTwoScalar.class,
        ConstantTwoScalar.optionMap(new Value("1".getBytes()), RNewVisibility),
        null, null, null, null, null, null, null, null, null, -1, Aauth, Aauth);

    tops.delete(Atmp); tops.delete(A2tmp);
    { String t = Atmp; Atmp = AtmpAlt; AtmpAlt = t; }
  } while (nnzBefore != nnzAfter);

  AdjBFS(Atmp, null, 1, Rfinal, null, null, -1, null, null,
         false, 0, Integer.MAX_VALUE, null, Aauth, Aauth);
  return nnzAfter;
}
```

Either AdjBFS or OneTable work here

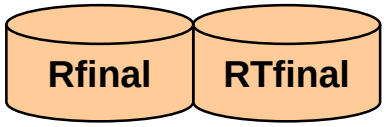
kTrussEdge: Design

Init



Loop

```
nnzAfter = nnz(E);
do {
  nnzBefore = nnzAfter;
  A = NoDiag(E^T * E);
  R = E * A;
  s = sum(R == 2, 2);
  E = E(s ≥ k-2, :);
  nnzAfter = nnz(E);
}while(nnzAfter≠nnzBefore);
```

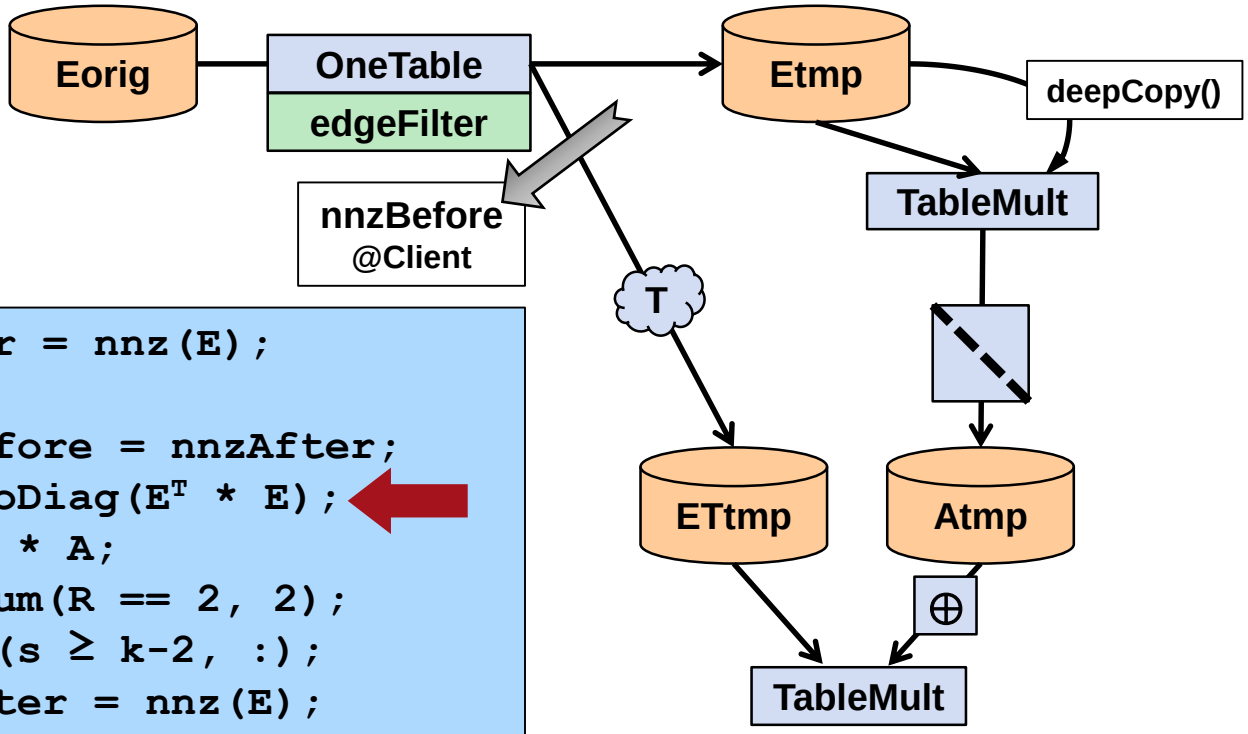


End

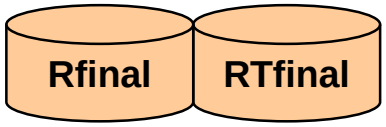
kTrussEdge: Design

Init

Loop

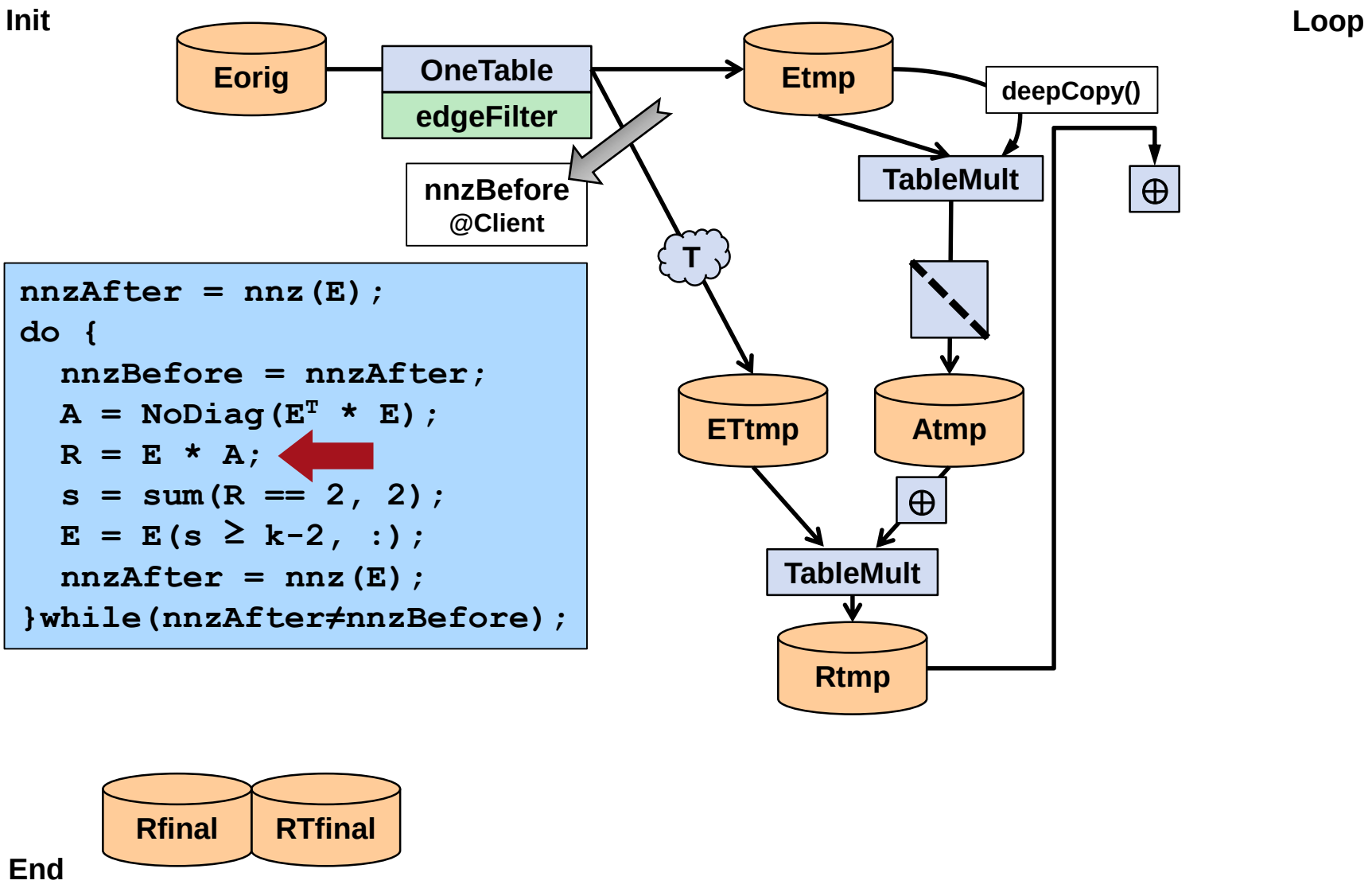



```
nnzAfter = nnz(E);
do {
  nnzBefore = nnzAfter;
  A = NoDiag(E^T * E);
  R = E * A;
  s = sum(R == 2, 2);
  E = E(s ≥ k-2, :);
  nnzAfter = nnz(E);
}while(nnzAfter≠nnzBefore);
```

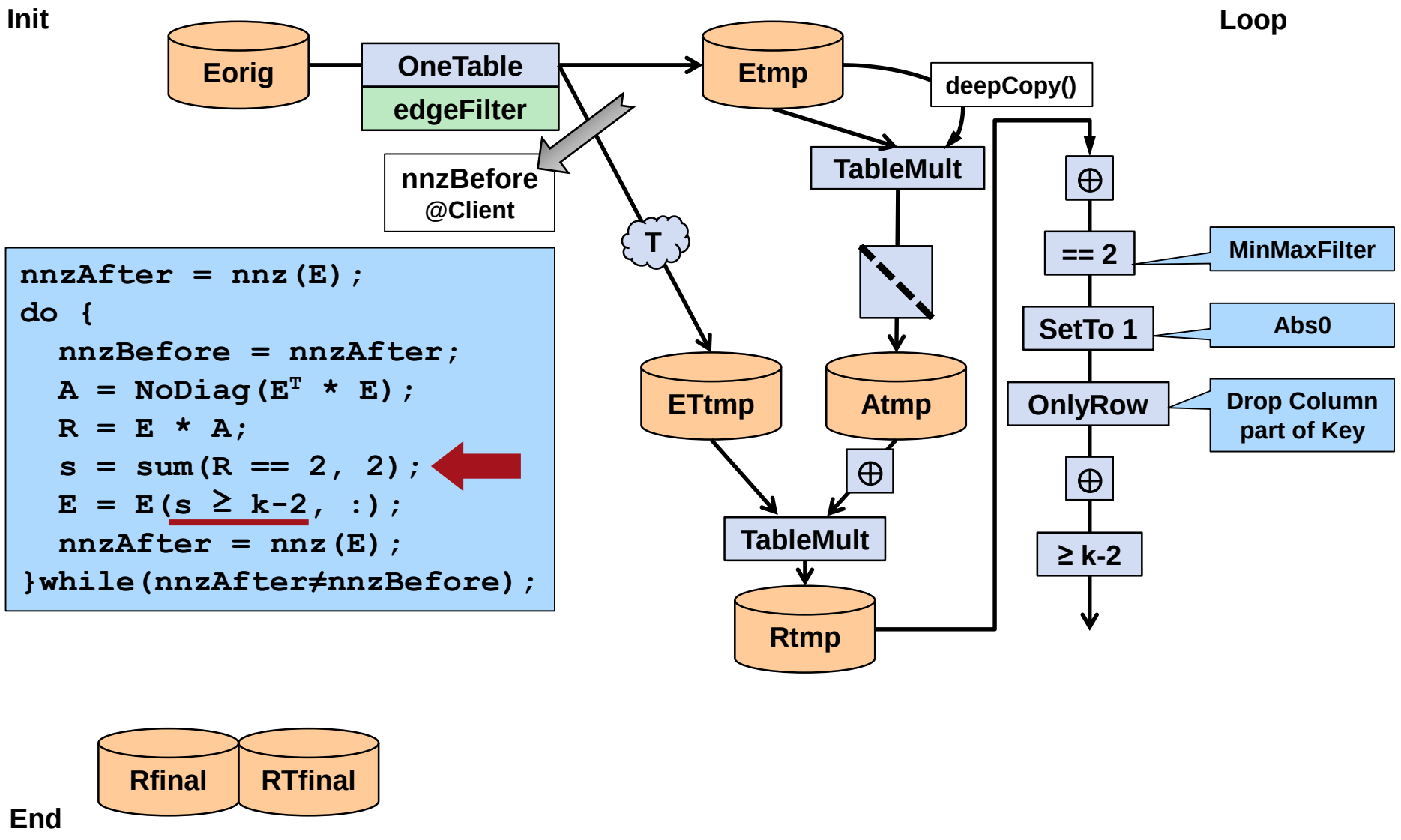


End

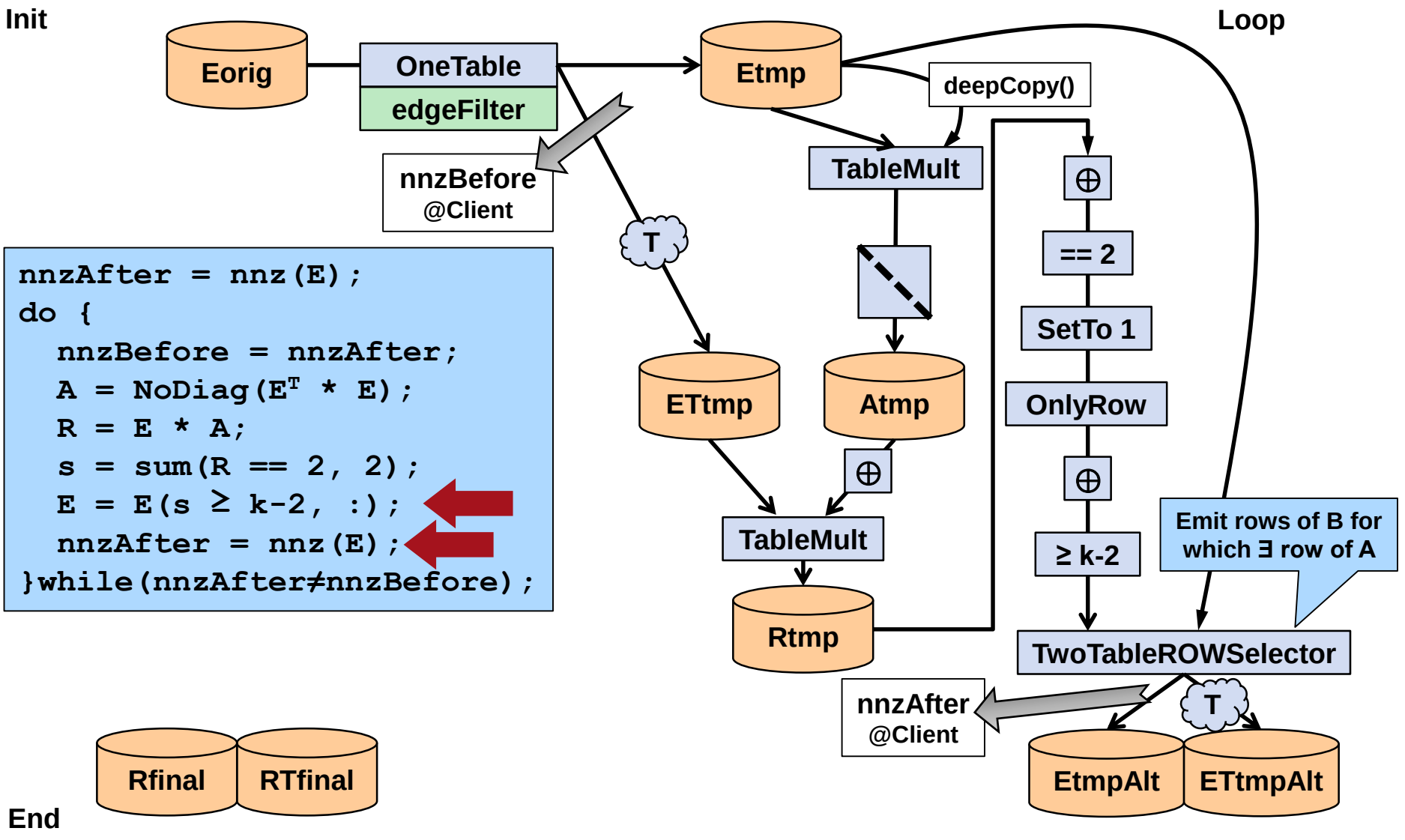
kTrussEdge: Design



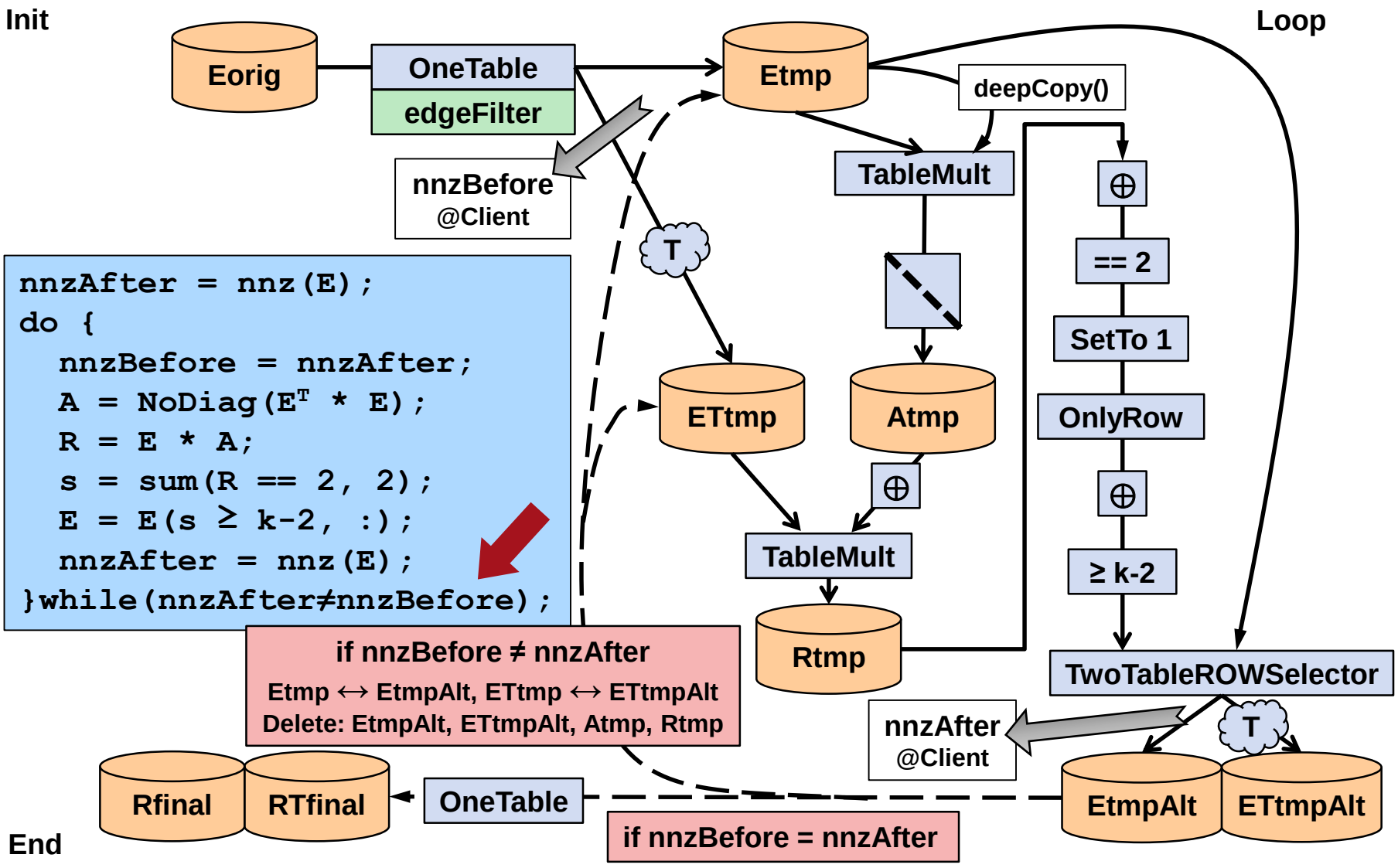
kTrussEdge: Design



kTrussEdge: Design



kTrussEdge: Design



kTrussEdge: Implementation

```
long kTrussEdge(String Eorig, String ETorig, String Rfinal, String RTfinal,
       int k, String edgeFilter, boolean forceDelete, Authorizations Eauth) {
  // …
  IteratorSetting noDiag = TriangularFilter.iteratorSetting(1, TriangularType.NoDiagonal);
  IteratorSetting itsBeforeR = new DynamicIteratorSetting(6, null)
      .append(PLUS_ITERATOR_LONG)
      .append(MinMaxFilter.iteratorSetting(1, ScalarType.LONG, 2, 2))
      .append(ConstantTwoScalar.iteratorSetting(1, new Value("1".getBytes())))
      .append(KeyRetainOnlyApply.iteratorSetting(1, PartialKey.ROW))
      .append(PLUS_ITERATOR_LONG)
      .append(MinMaxFilter.iteratorSetting(10, ScalarType.LONG, k-2, null))
      .toIteratorSetting();
  do {
    nnzBefore = nnzAfter;

    TableMult(TwoTableIterator.CLONESOURCE_TABLENAME, Etmp, Atmp, null, -1,
        ConstantTwoScalar.class, null, PLUS_ITERATOR_LONG, null, null, null, false, false,
        null, null, Collections.singletonList(noDiagFilter), null, null, -1, Eauth, Eauth);

    TableMult(ETtmp, Atmp, Rtmp, null, ConstantTwoScalar.class, itsBeforeR, Eauth, Eauth);
    tops.delete(ETtmp); tops.delete(Atmp);

    nnzAfter = TwoTableROWSelector(Rtmp, Etmp, EtmpAlt, ETtmpAlt, -1, null, null, null,
        true, null, null, null, null, null, -1, Eauth, Eauth);
    tops.delete(Etmp); tops.delete(Rtmp);
    { String t = Etmp; Etmp = EtmpAlt; EtmpAlt = t; }
    { String t = ETtmp; ETtmp = ETtmpAlt; ETtmpAlt = t; }
  } while (nnzBefore != nnzAfter);
  // …                                                    return nnzAfter;
```

Outline

- Download, install, test, see examples, use as a library. Maven build cycle.
- Motivating algorithm: AdjBFS w/ degree filtering
  - Specifying Column Visibilities & Authorizations
- Three Graph Schemas: Adjacency, Incidence, Single-Table
  - Degree Tables and utility functions to help ingest
- Mapping to GraphBLAS
- Graphulo Core Client functions and their Server-side Iterators
  - OneTable, Reducer, D4M String format, ApplyOp
  - TwoTableIterator, RemoteSourceIterator, DynamicIterator, EWiseOp
  - TwoTable variants: TwoTableROW, TwoTableEWISE, TwoTableNONE
    - TwoTableROW variants: RowMultiplyOp, CartesianRowMultiply (& MultiplyOp), SelectorRowMultiply
  - TableMult as TwoTableROW
  - SimpleTwoScalar: MathTwoScalar, ConstantTwoScalar
- Algorithms: EdgeBFS, SingleBFS, Jaccard, kTrussAdj, kTrussEdge
- **Extensions**
- Topics not covered: NMF, Monitoring, Benchmark, Debug, Other algs.: TF-IDF, SCC, …

Extensions

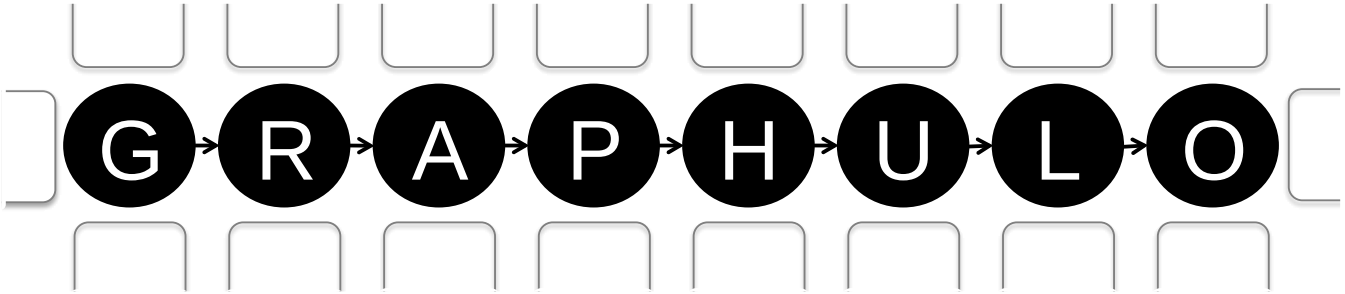
- **Parallelism**
  - Graphulo functions currently block
  - Could run in parallel with different threads
- **LruCacheIterator**
  - Goal is to reduce # of entries written through BatchWriter
    by pre-summing entries a Combiner would sum at remote table anyway
  - Akin to the local Combiner optimization in MapReduce
  - Place in out-of-order entry streams
  - Performance increase ranges from negligible to extreme
    based on input sparsity pattern
- **ThreeTableIterator? NTableIterator?**
  - TwoTableIterator scans two tables along a shared dimension
  - Could extend to 3+ tables
- **Supporting Column Family more rigorously. Encrypting IteratorOptions**
- **More algorithms!**

Conclusion

- **Try it! https://github.com/Accla/graphulo**
- **Please send bug reports to dhutchis@uw.edu**

To an Era of Graph Analytics Server-side on Accumulo

Enabling Insight at Scale: Bigger, Faster, Distributed, Secure



Backup Slides

Please note the NMF algorithm has changed to a more efficient version, not yet fully documented.

Massachusetts Institute of Technology

Random Points

- + can be null, meaning no combiner
  - new entries overwrite colliding old entries, as per VersioningIterator
- Monitoring
  - Idea: send entries from scan iterators to client indicating progress
  - Requires careful iterator design to re-create state if Accumulo tears down the iterator stack after it returns an entry
    - All Graphulo iterators designed to work in the event of a tear-down
  - Problem: Accumulo batches entries before sending them to the client
    - Client would not see monitoring entries until a significant number of them accumulated. The client should see monitoring entries right away.
  - Solution: Reduce `table.scan.max.memory` parameter for the scanned table to 1 byte.
    - Works but not pretty. Affects all concurrent scans on the table & logs warnings.

Non-negative Matrix Factorization

- **Used for topic modeling into k topics**
  - **Group similar row and column labels**
    - **Used in recommender systems**
  - **Other methods (not presently in Graphulo):**
    - **Latent Dirichlet allocation, Latent semantic analysis**
    - **SVD taking eigenvectors with top k eigenvalues**

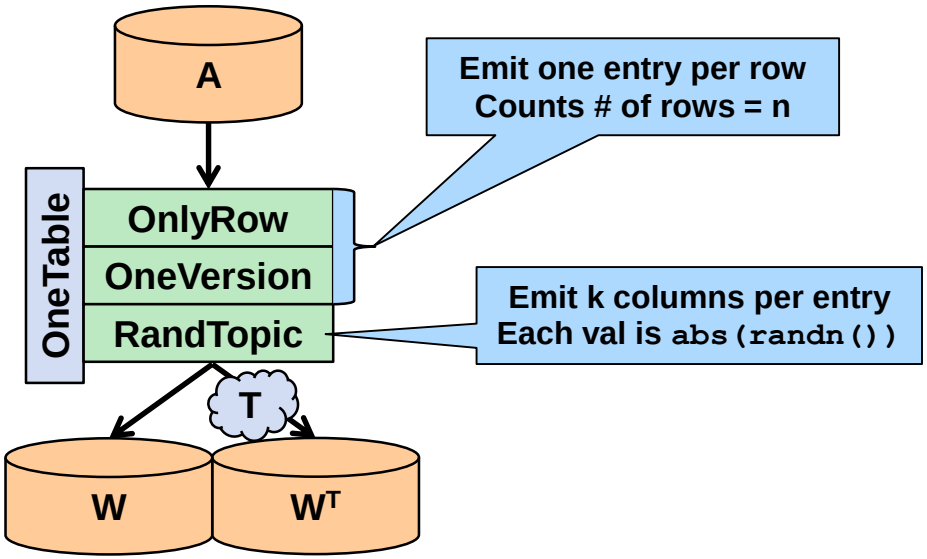
$$A_{mxn} = W_{mxk} * H_{kxn}$$

- **W and H are *dense* numeric weights**
- **See mathematics:**
  - V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner, "Graphulo: Linear algebra graph kernels for NoSQL databases," in International Parallel & Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2015.
- **Often run on incidence matrix**
  - **Adjacency matrix leads to symmetric W and H**

```
double NMF(String Aorig, String ATorig,
           String Wfinal, String WTfinal,
           String Hfinal, String HTfinal,
           final int K, final int maxiter,
           boolean forceDelete)
```

NMF: Create Random W

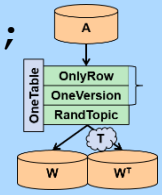
```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```

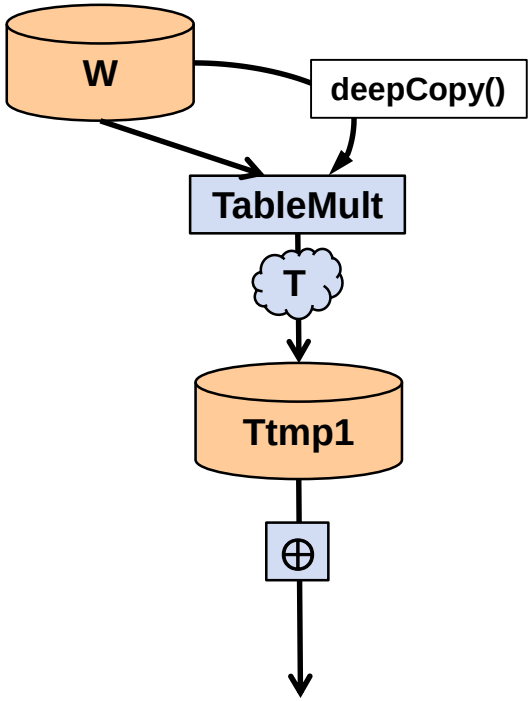


NMF: Iteration H Step 1

Delete H, H^T

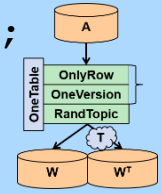
```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```

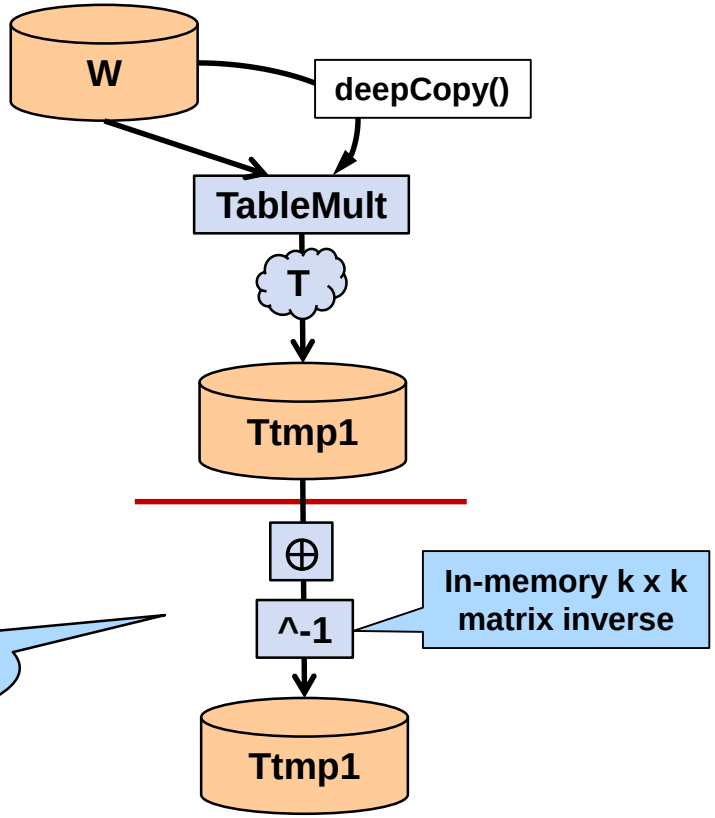




NMF: Iteration H Step 2

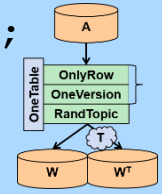
```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```



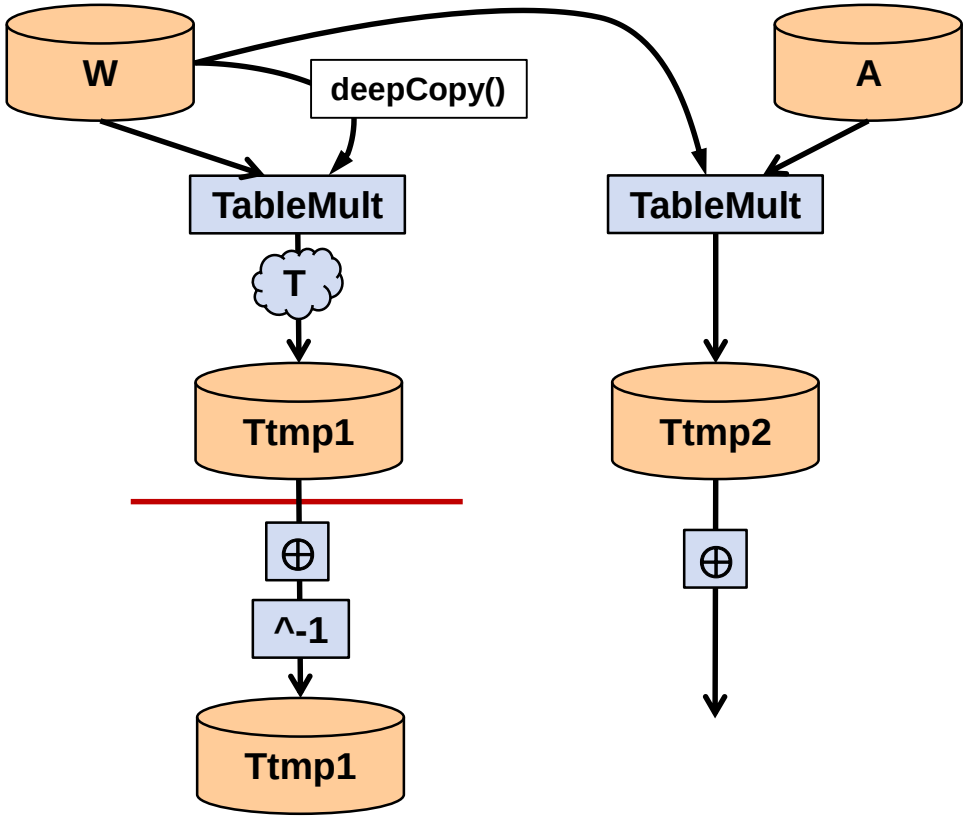


NMF: Iteration H Step 3

```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```

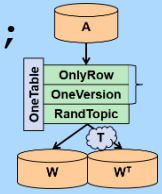


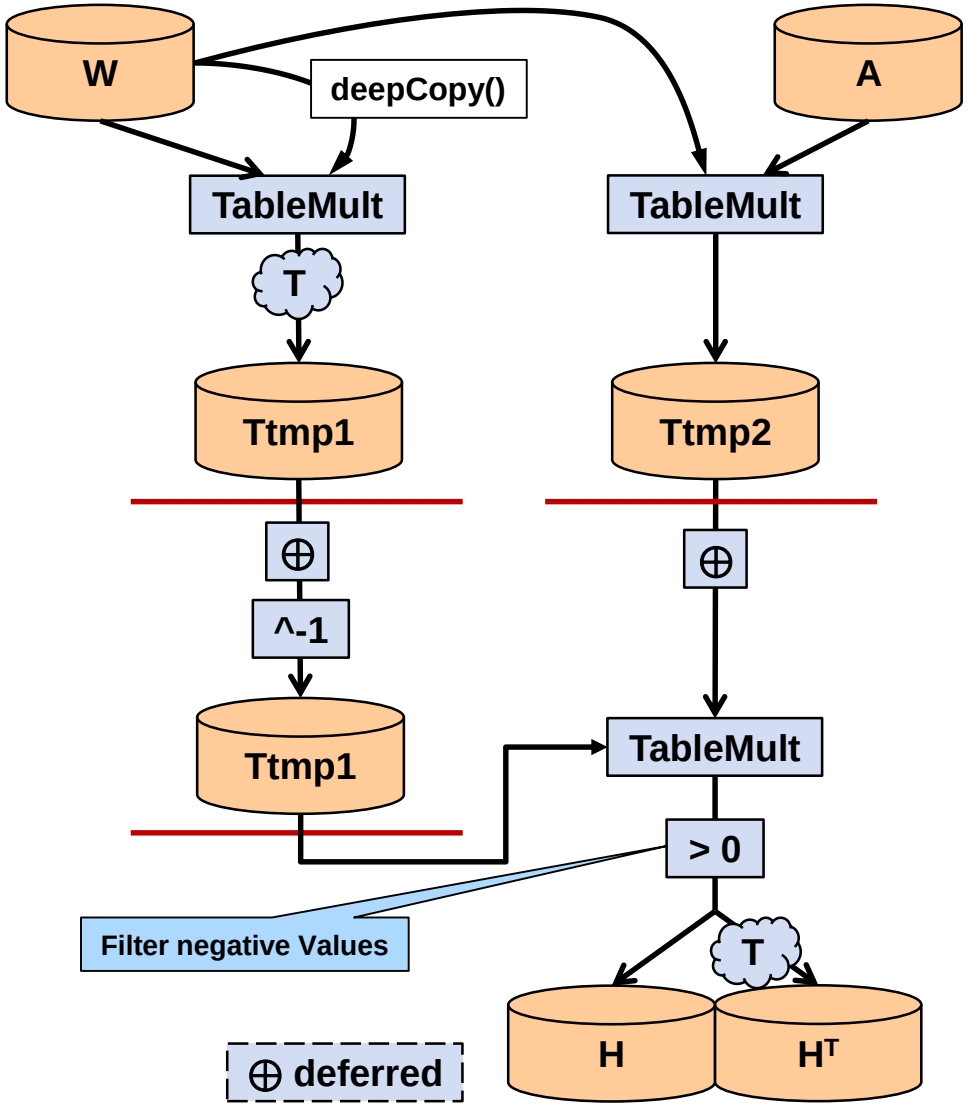
Step 3 can run in parallel with 1 and 2



NMF: Iteration H Step 4

```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```



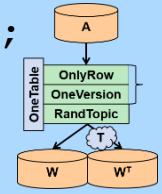


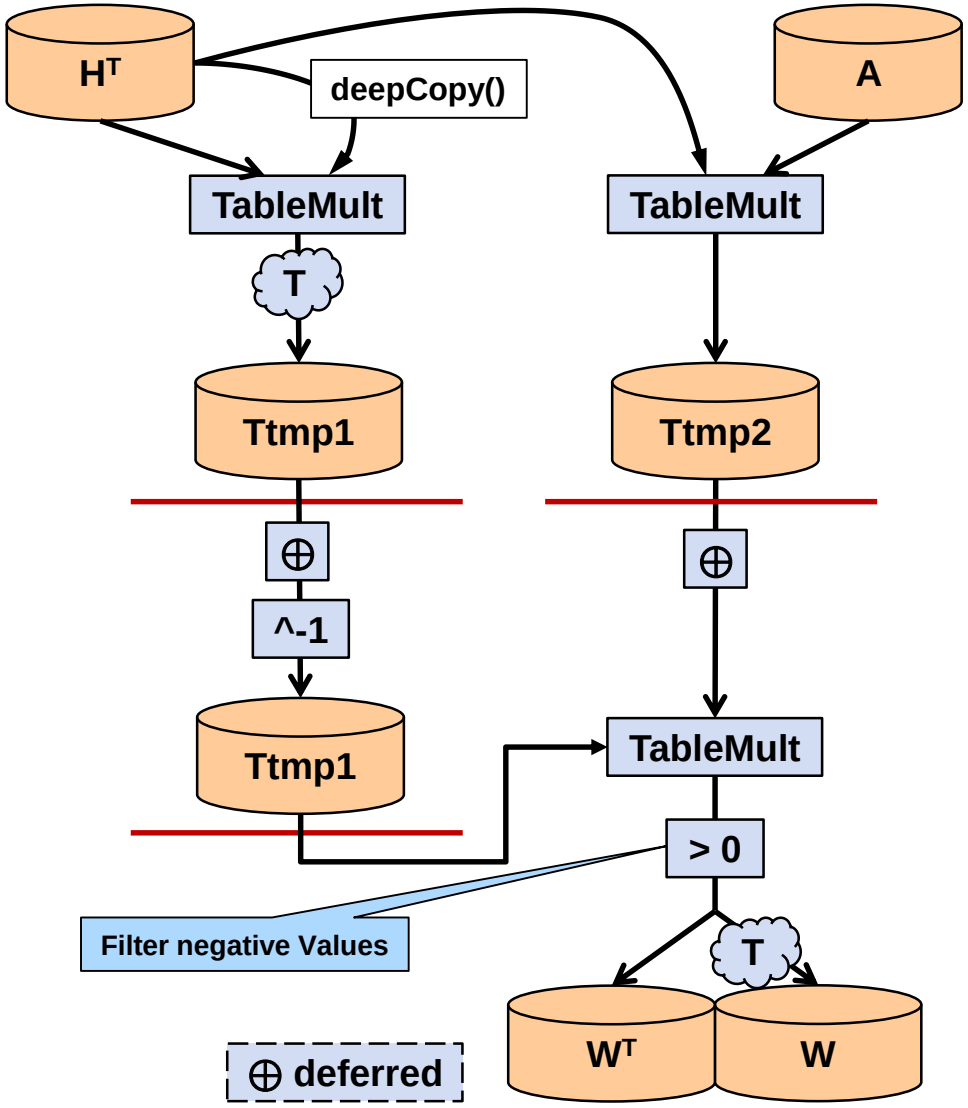
Delete Ttmp1, Ttmp2

⊕ deferred

NMF: Iteration W^T Step 4

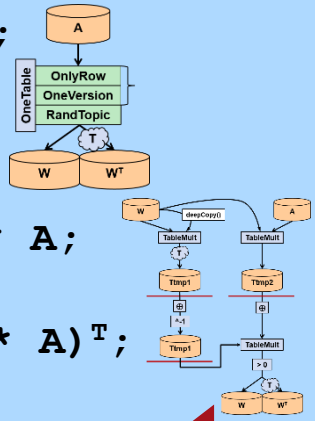
```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```

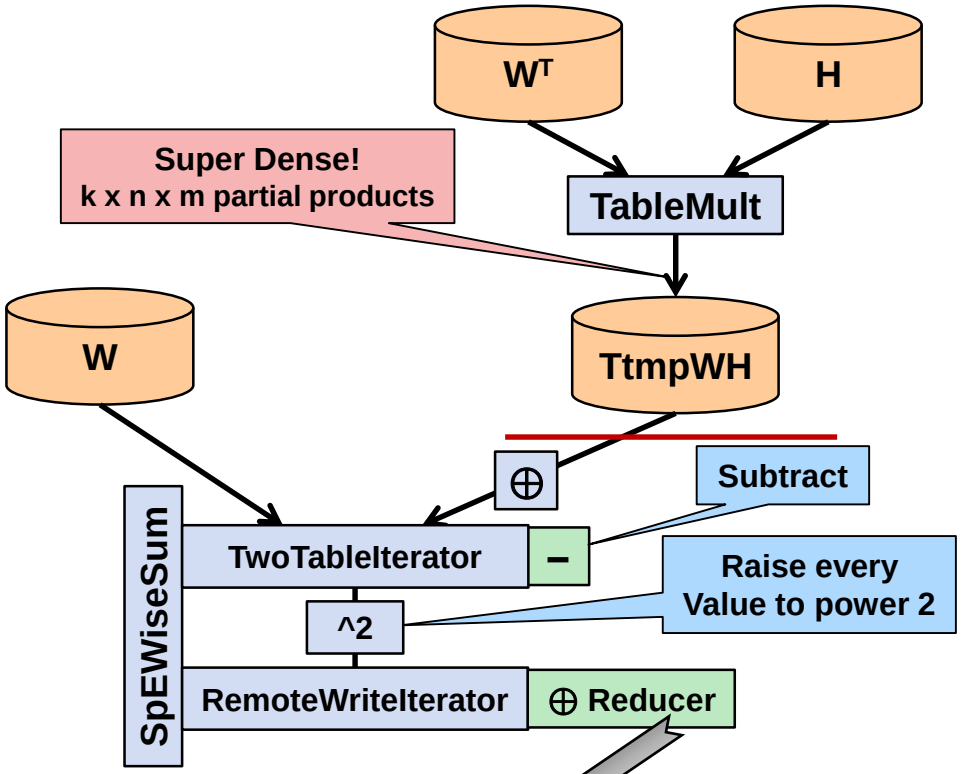




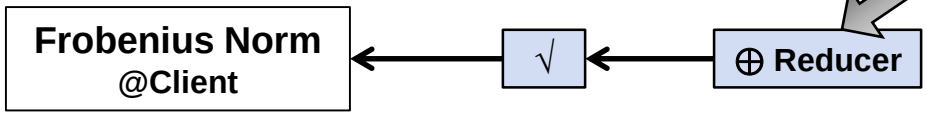
NMF: Frobenius Norm

```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```





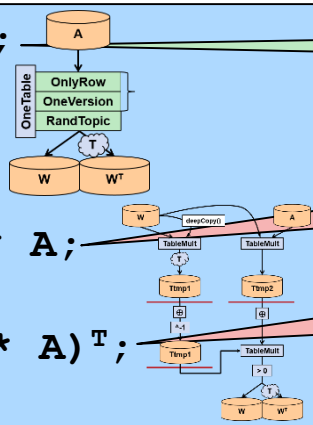
$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$



Delete TtmpWH

NMF: Cost of Iterations

```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```



1x OneTable

3x TableMult
1x Compaction

3x TableMult
1x Compaction

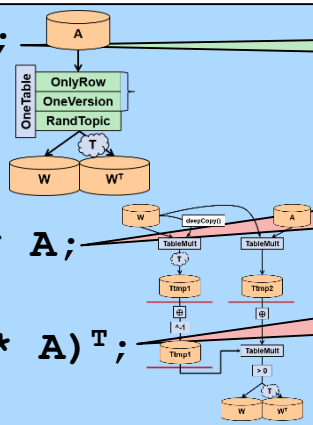
1x TableMult
1x SpEWiseSum

Heavy Iterations…

Repeat while difference in errors is > 0.01
or reached maximum # of iterations

NMF: Cost of Iterations

```
W = abs(randn(n,k));
newer = 0;
do {
  olderr = newerr;
  H = (W^T*W)^-1 * W^T * A;
  H = H .* H>0;
  W = ((H*H^T)^-1 * H * A)^T;
  W = W .* W>0;
  newerr = FroNorm(A-W*H);
}while(abs(newerr-olderr)>.01);
```



1x OneTable

3x TableMult
1x Compaction

3x TableMult
1x Compaction

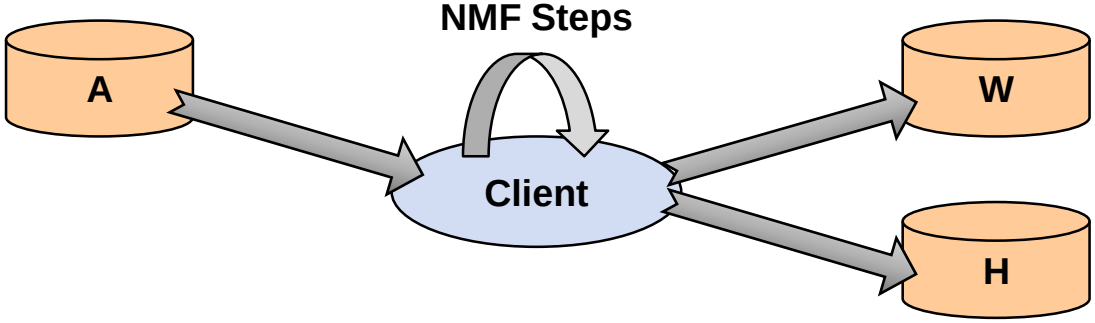
1x TableMult
1x SpEWiseSum

Heavy Iterations…

Repeat while difference in errors is > 0.01
or reached maximum # of iterations

➔ Motivates in-memory version
Requires holding dense
n x m matrix in memory

NMF Steps



NMF: Create Random W

```
List<IteratorSetting> itCreateTopicList = new DynamicIteratorSetting()
    .append(KeyRetainOnlyApply.iteratorSetting(1, PartialKey.ROW))
    .append(new IteratorSetting(1, VersioningIterator.class))
    .append(RandomTopicApply.iteratorSetting(1, K))
    .getIteratorSettingList();

long NK = OneTable(Aorig, Wfinal, WTfinal, null, -1,
        null, null, null, null, null, itCreateTopicList, null);
```

Inside RandomTopicApply.java:

```
Iterator<? extends Map.Entry<Key, Value>> apply(Key k, Value v) {
  Text row = k.getRow();
  SortedMap<Key,Value> map = new TreeMap<>();
  for (int i = 1; i <= knum; i++) {
    Key knew = new Key(row, EMPTY_TEXT, new Text(Integer.toString(i)),
                       System.currentTimeMillis());
    Value vnew = new Value(Double.toString(
        Math.abs(rand.nextGaussian())).getBytes());
    map.put(knew, vnew);
  }
  return map.entrySet().iterator();
}
```

NMF: Top-level Loop

```
double newerr = 0, olderr;
int numiter = 0;

do {
  numiter++;
  olderr = newerr;

  nmfStep(K, Wfinal, Aorig, Hfinal, HTfinal, Ttmp1, Ttmp2);

  nmfStep(K, HTfinal, ATorig, WTfinal, Wfinal, Ttmp1, Ttmp2);

  newerr = nmfDiffFrobeniusNorm(Aorig, WTfinal, Hfinal, Ttmp1);

} while (Math.abs(newerr - olderr) > 0.01d && numiter < maxiter);

return Math.abs(newerr - olderr);
```

NMF: Iteration Step

```
void nmfStep(int K, String in1, String in2, String out1, String out2,
             String tmp1, String tmp2) {
  deleteTables(true, out1, out2);

  TableMult(in1, in1, null, tmp1, -1,
    MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.DOUBLE),
    MathTwoScalar.combinerSetting(6, null, ScalarOp.PLUS, ScalarType.DOUBLE),
    null, null, null, false, false, -1, false);

  connector.tableOperations().compact(tmp1, null, null,
    Collections.singletonList(InverseMatrixIterator.iteratorSetting(7, K)), true, true);

  TableMult(in1, in2, tmp2, null, -1,
    MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.DOUBLE),
    MathTwoScalar.combinerSetting(6, null, ScalarOp.PLUS, ScalarType.DOUBLE),
    null, null, null, false, false, -1, false);

  IteratorSetting sumFilterOp = new DynamicIteratorSetting()
    .append(MathTwoScalar.combinerSetting(1, null, ScalarOp.PLUS, ScalarType.DOUBLE))
    .append(MinMaxFilter.iteratorSetting(1,ScalarType.DOUBLE,
                                         Double.MIN_NORMAL,Double.MAX_VALUE))
    .toIteratorSetting(6);

  TableMult(tmp1, tmp2, out1, out2, -1,
    MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.DOUBLE),
    sumFilterOp, null, null, null, false, false, null, null, null, null, null, -1, false);

  deleteTables(true, tmp1, tmp2);
}
```

⊕ combiner could be applied once at compaction instead, if VersioningIterator disabled

Full major compaction with iterator
Flushes before compact. Blocks until finished.

Safe to remove ⊕ combiner after compaction

Filter +eps to +inf

NMF: Frobenius Norm

```
double nmfDiffFrobeniusNorm(String Aorig, String WTfinal, String Hfinal,
                            String WHtmp) {

  TableMult(WTfinal, Hfinal, WHtmp, null, -1,
      MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.TIMES, ScalarType.DOUBLE),
      MathTwoScalar.combinerSetting(6, null, ScalarOp.PLUS, ScalarType.DOUBLE),
      null, null, null, false, false, -1, false);

  List<IteratorSetting> iterAfterMinus = Collections.singletonList(
                           MathTwoScalar.applyOpDouble(1, true, ScalarOp.POWER, 2.0));

  Map<String,String> sumOpt = MathTwoScalar.optionMap(ScalarOp.PLUS, ScalarType.DOUBLE);
  MathTwoScalar sumReducer = new MathTwoScalar();
  sumReducer.init(sumOpt, null);

  SpEWiseSum(Aorig, WHtmp, null, null, -1,
      MathTwoScalar.class, MathTwoScalar.optionMap(ScalarOp.MINUS, ScalarType.DOUBLE),
      null, null, null, null, null, null,
      iterAfterMinus, sumReducer, sumReducerOpts, -1, false);

  deleteTables(true, WHtmp);
  if (!sumReducer.hasTopForClient())
    return 0.0;
  return Math.sqrt(Double.parseDouble(new String(sumReducer.getForClient())));
}
```

Reducer created outside SpEWiseSum because it runs at client as well as server

Reducer updated as a side effect