Apache Accumulo 1.4 & 1.5 Features

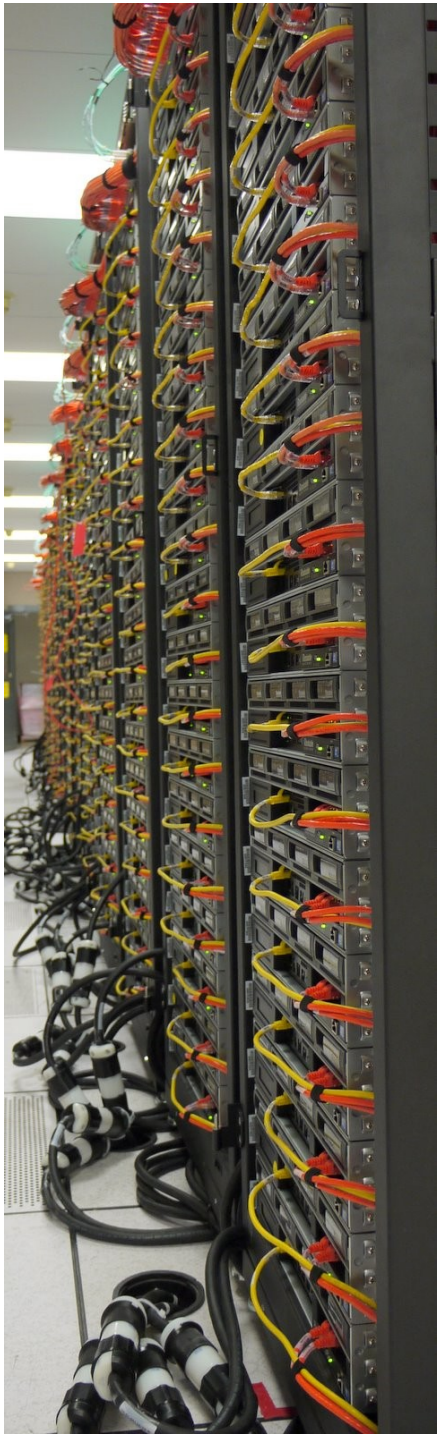
Inaugural Accumulo DC Meetup

Keith Turner

What is Accumulo?

A re-implementation of Big Table

- Distributed Database
- Does not support SQL
- Data is arranged by
  - Row
  - Column
  - Time
- No need to declare columns, types

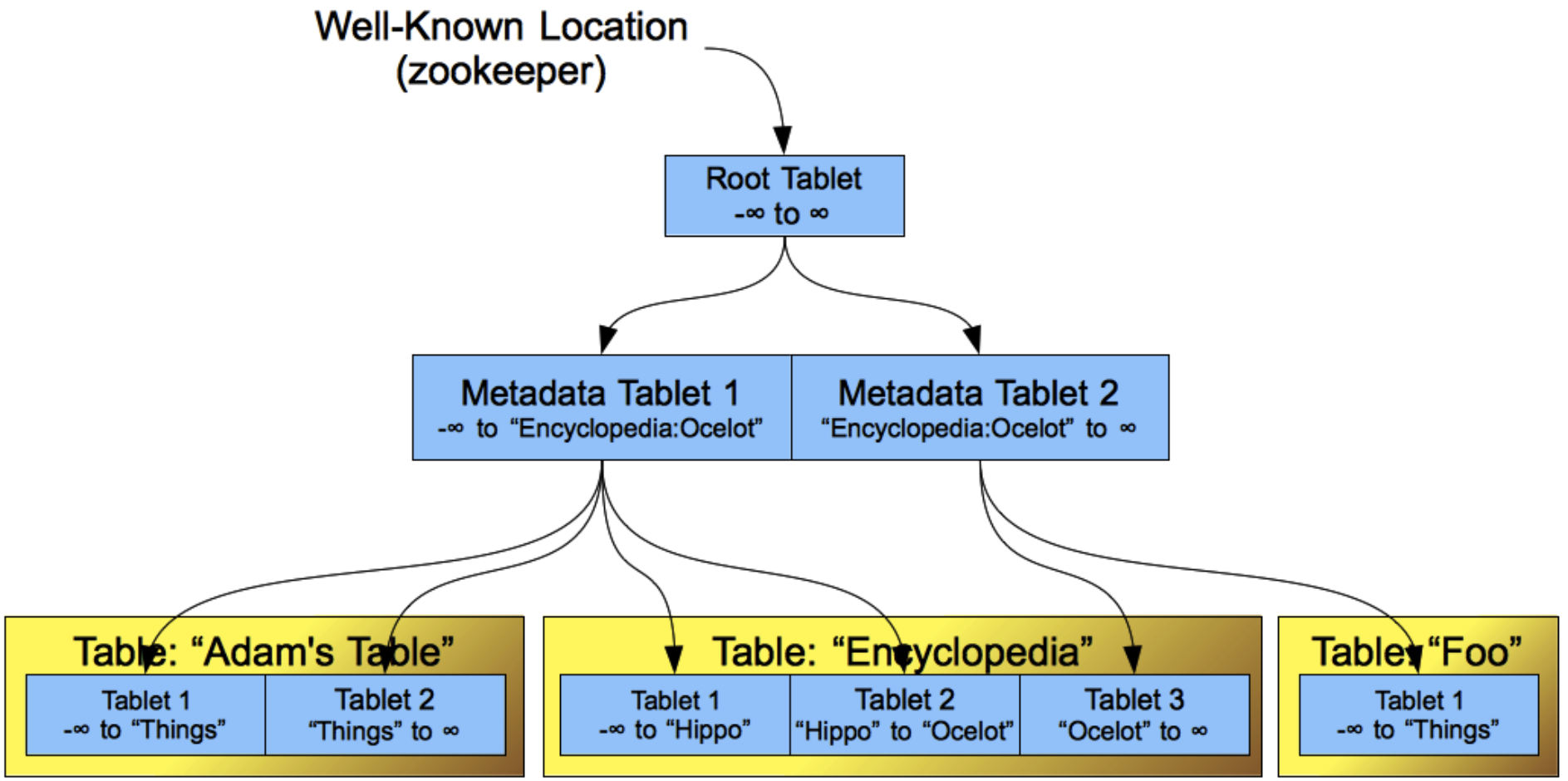


Massively Distributed

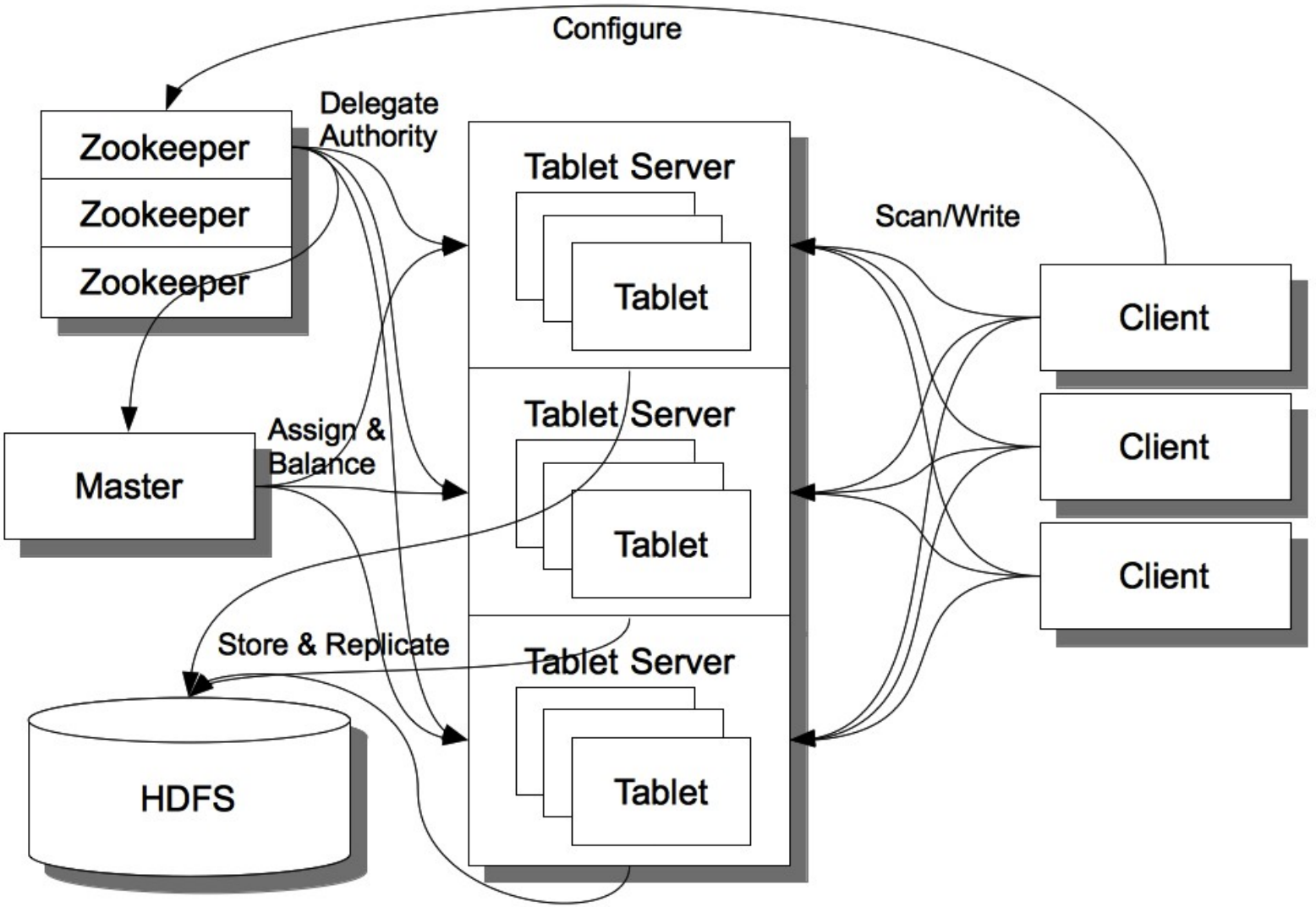
Key Structure

Key					Value
Row	Column Family	Column Qualifier	Column Visibility	Time Stamp	Value
Primary Partitioning Component	Secondary Partitioning Component	Uniqueness Control	Access Control	Versioning Control	Value

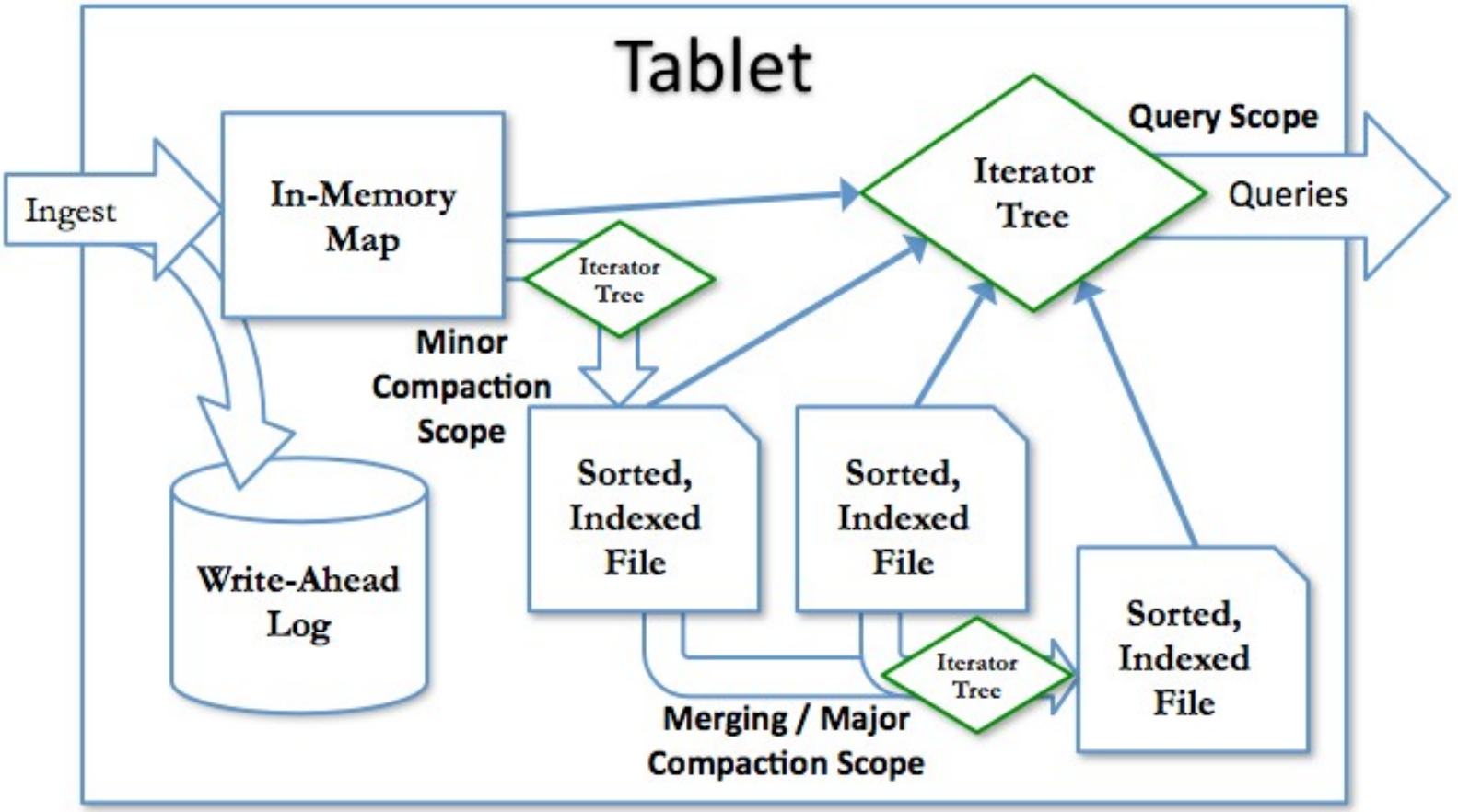
Tablet Organization



System Processes



Tablet Data Flow

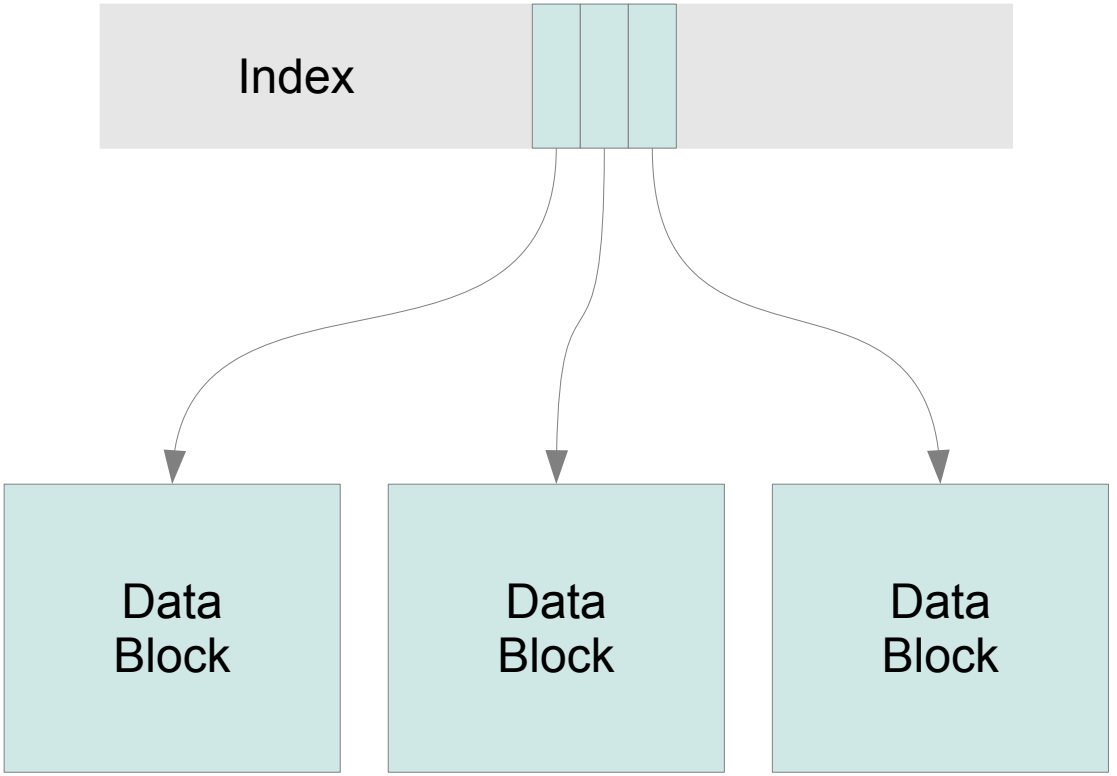


Rfile Multi-level index

1.3 Rfile Index

- Index is an array of keys
- One key per data block
- Complete index loaded into memory when file opened
- Complete index kept in memory when writing file
- Slow load times
- Memory exhaustion

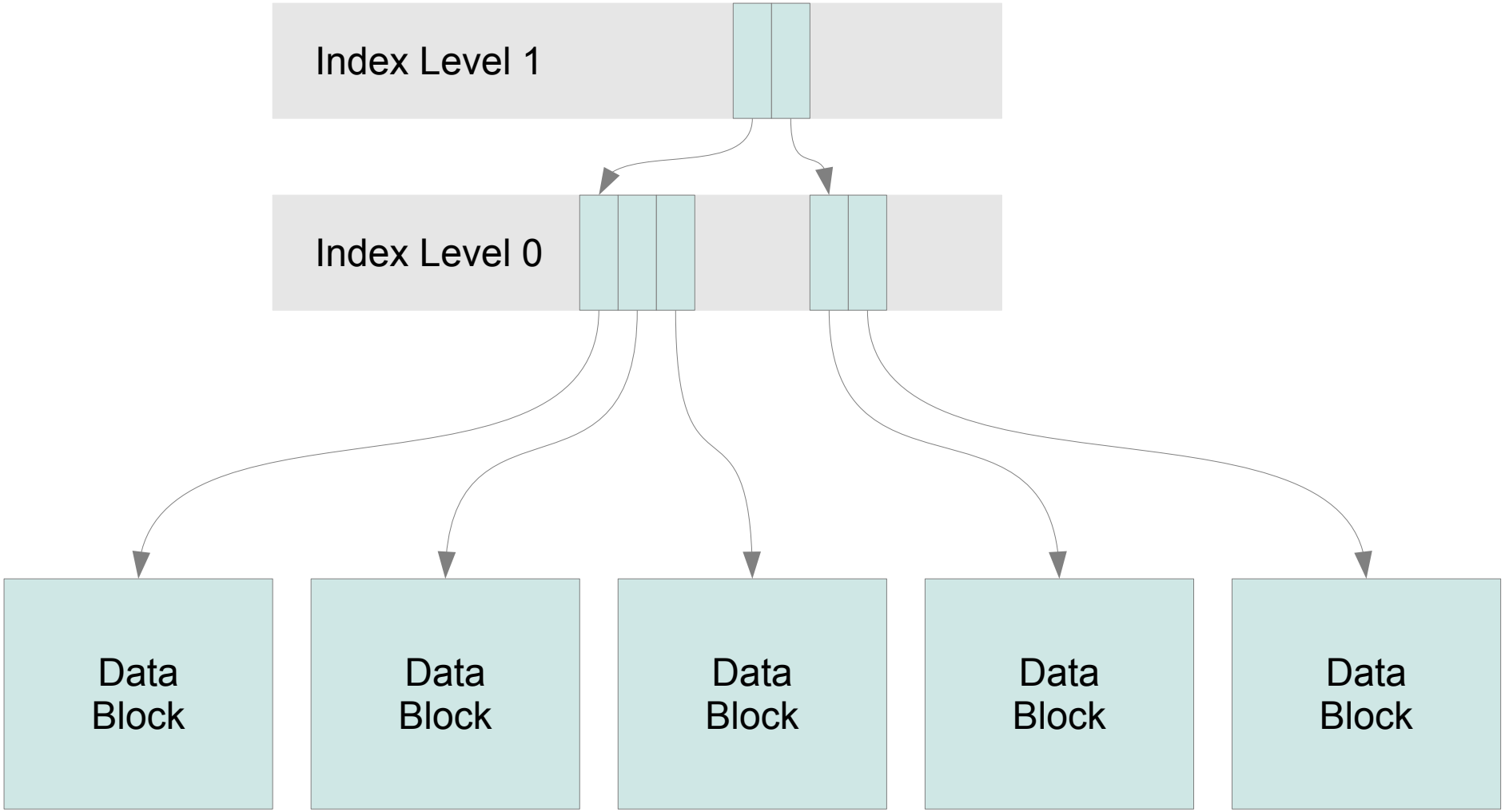
1.3 RFile



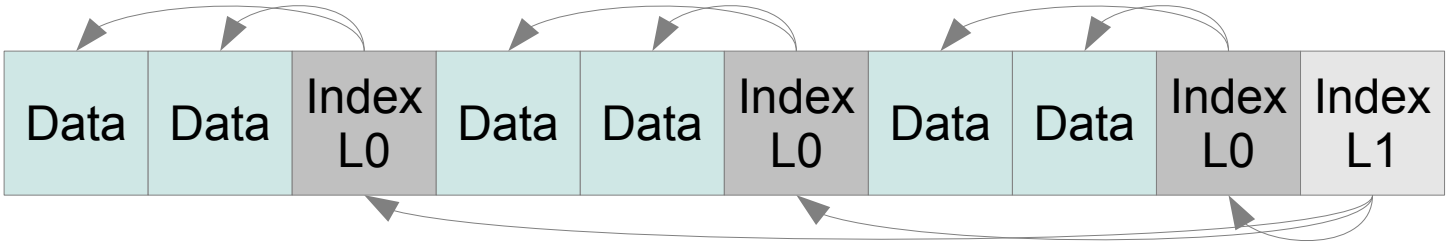
Non multi-level 9G file

```
Locality group          : <DEFAULT>
   Start block           : 0
   Num   blocks          : 182,691
   Index level 0 size    : 8,038,404 bytes
   First key             : 00000008611ae5d6 0e94:74e1 [] 1310491465052 false
   Last key              : 7ffffff7daec0b4e 43ee:1c1e [] 1310491475396 false
   Num entries           : 172,014,468
   Column families       : <UNKNOWN>

Meta block     : BCFile.index
   Raw size              : 2,148,491 bytes
   Compressed size       : 1,485,697 bytes
   Compression type      : lzo

Meta block     : RFile.index
   Raw size              : 8,038,470 bytes
   Compressed size       : 5,664,704 bytes
   Compression type      : lzo
```

1.4 RFile


Index Level 1
Index Level 0
Data Block
Data Block
Data Block
Data Block
Data Block

File layout



- Index blocks written as data is written
- complete index not kept in memory during write
- Index block for each level kept in memory

Multi-level 9G file

```
Locality group          : <DEFAULT>
   Start block          : 0
   Num   blocks         : 187,070
   Index level 1        : 4,573 bytes  1 blocks
   Index level 0        : 10,431,126 bytes  82 blocks
   First key            : 00000008611ae5d6 0e94:74e1 [] 1310491465052 false
   Last key             : 7ffffff7daec0b4e 43ee:1c1e [] 1310491475396 false
   Num entries          : 172,014,468
   Column families      : <UNKNOWN>

Meta block     : BCFile.index
   Raw size             : 5 bytes
   Compressed size      : 17 bytes
   Compression type     : lzo

Meta block     : RFile.index
   Raw size             : 4,652 bytes
   Compressed size      : 3,745 bytes
   Compression type     : lzo
```

Test Setup

- Files in FS cache for test (cat test.rf > /dev/null)
- Too much variability when some of file was in FS cache and some was not
- Easier to force all of file into cache than out

Open, seek 9G file

Multilevel Index	Cache	Avg Time
N	N	139.48 ms
N	Y	37.55 ms
Y	N	8.48 ms
Y	Y	3.90 ms

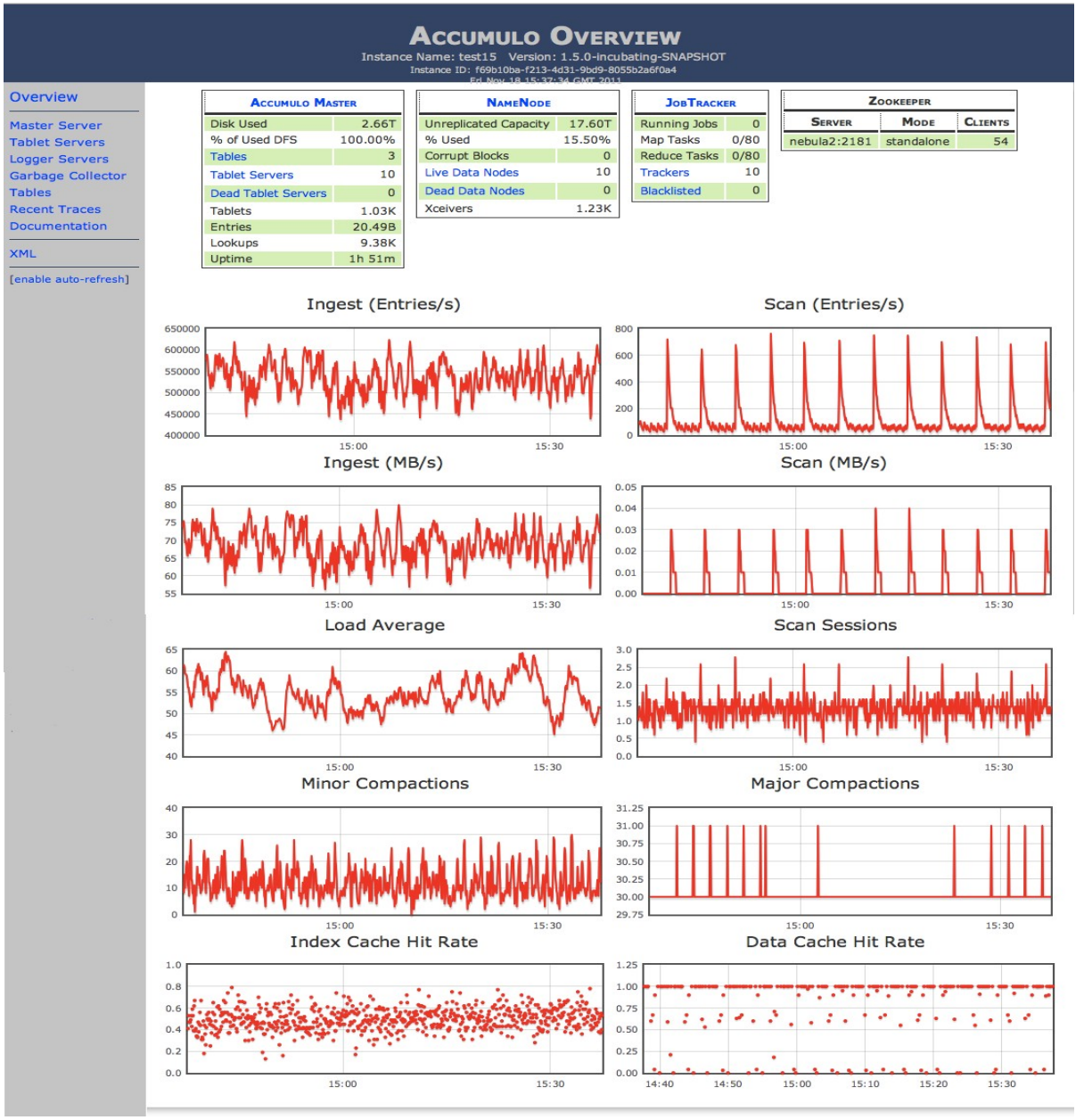
Randomly seeking 9G file

Multilevel Index	Cache	Avg Time
N	N	2.08 ms
N	Y	2.32 ms
Y	N	4.33 ms
Y	Y	2.14 ms

Configuration

- Index block size configurable per table
- Make index block size large and it will behave like old rfile
- Enabled index cache by default in 1.4

Cache hit rate on monitor page



Fault tolerant concurrent table operations

Problematic situations

- Master dies during create table
- Create and delete table run concurrently
- Client dies during bulk ingest
- Table deleted while writing or scanning

Create table fault

- Master dies during
  - Could leave accumulo metadata in bad state
- Master creates table, but then dies before notifying client
  - If client retries, it gets table exist exception

FATE
Fault Tolerant Executor

- If process dies, previously submitted operations continue to execute on restart.
- Serializes operation in zookeeper before execution
- Master uses FATE to execute table operations

Bulk import test

- Create table with summation aggregator
- Bulk import files of +1 or -1, graph ensures final sum is 0
- Concurrently merge, split, compact, and consistency check table
- Exit point verifies 0

Adampotent

- Idempotent f(f(x)) = f(x)
- Adampotent f(f'(x)) = f(x)
  - f'(x) denotes partial execution of f(x)

REPO
Repeatable Persisted Operation

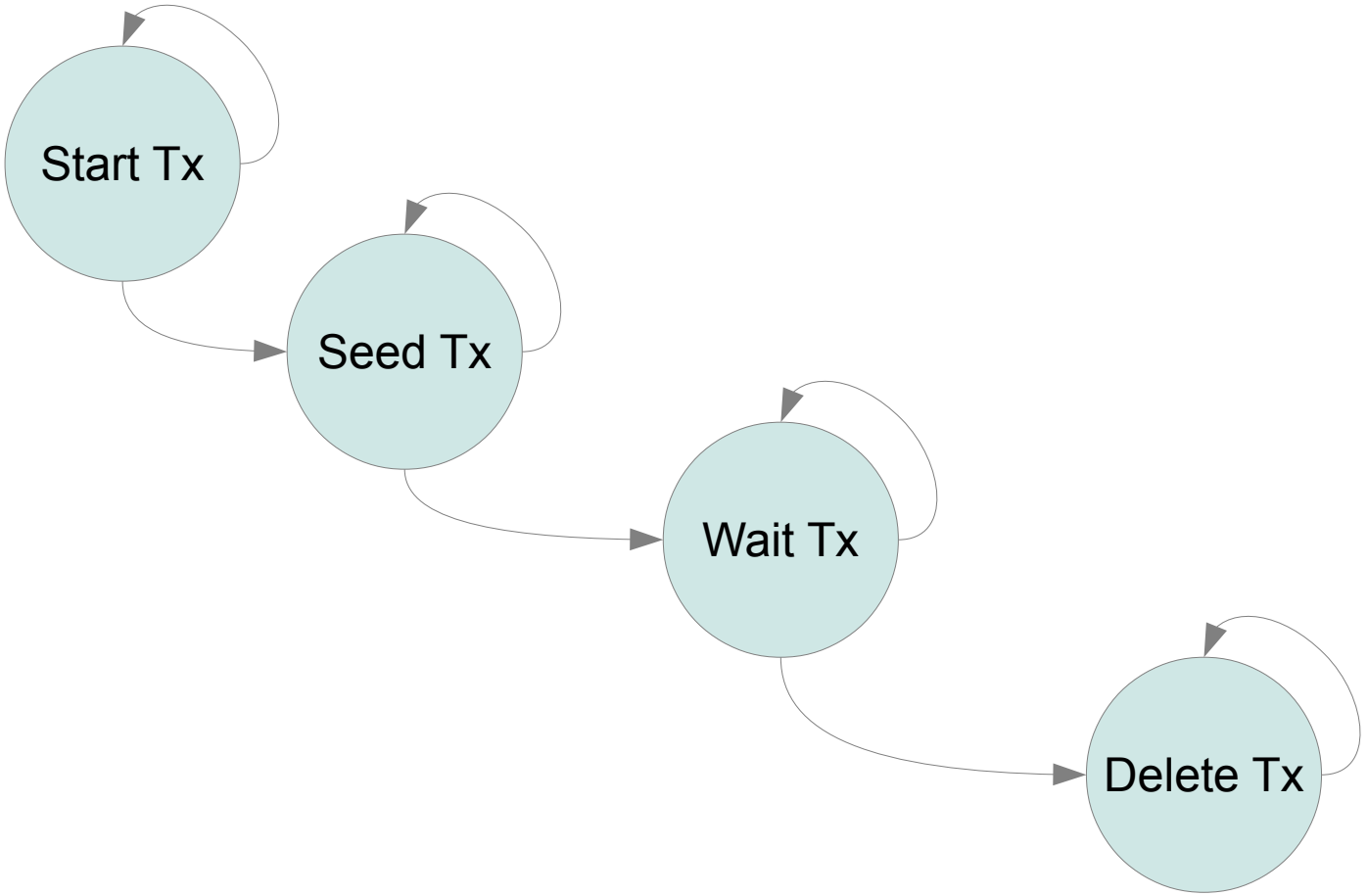
```
public interface Repo<T> extends Serializable {
     long isReady(long tid, T environment) throws Exception;
     Repo<T> call(long tid, T environment) throws Exception;
     void undo(long tid, T environment) throws Exception;
}
```

- call() returns next op, null if done
- call() and undo() must be adampotent
- undo() should clean up partial execution of isReady() or call()

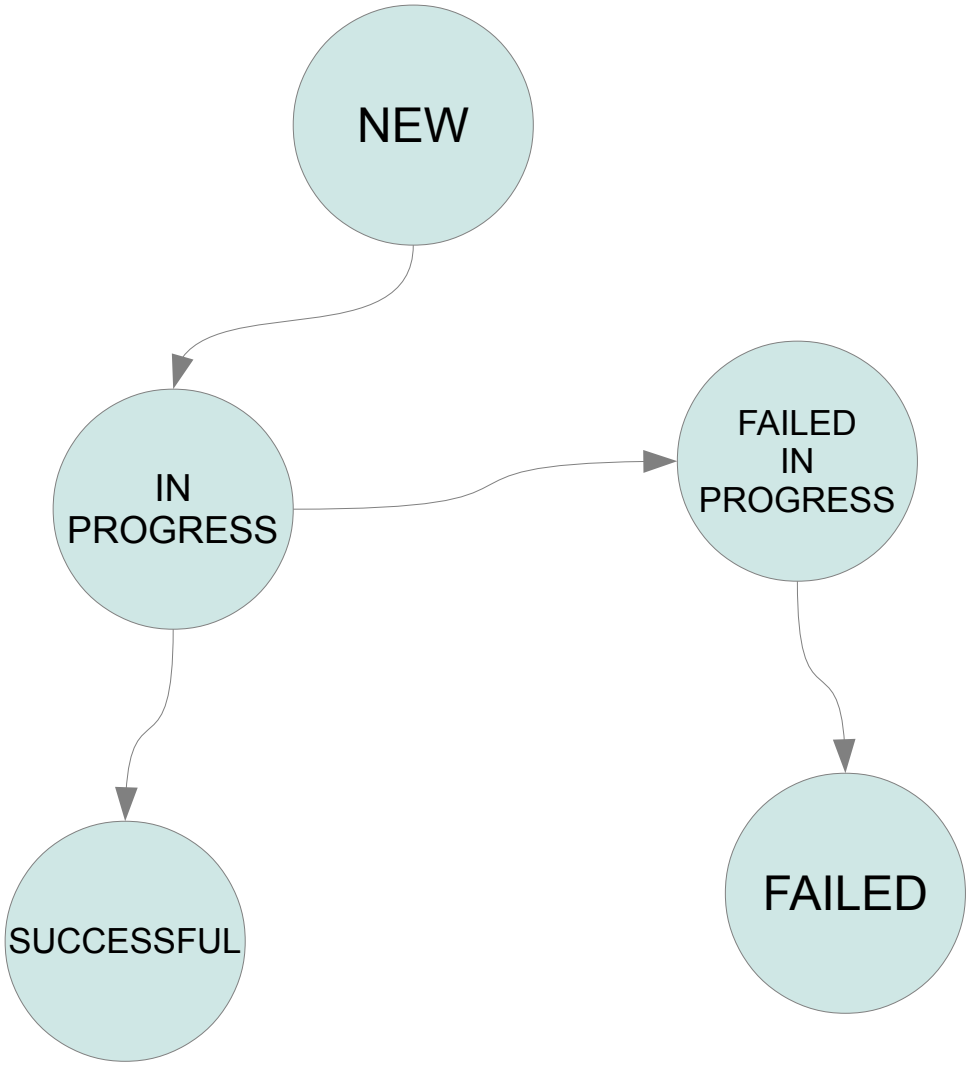
FATE interface

```
long startTransaction();
void seedTransaction(long tid, Repo op);
TStatus waitForCompletion(long tid);
Exception getException(long tid);
void delete(long tid);
```

Client calling FATE on master



FATE transaction states



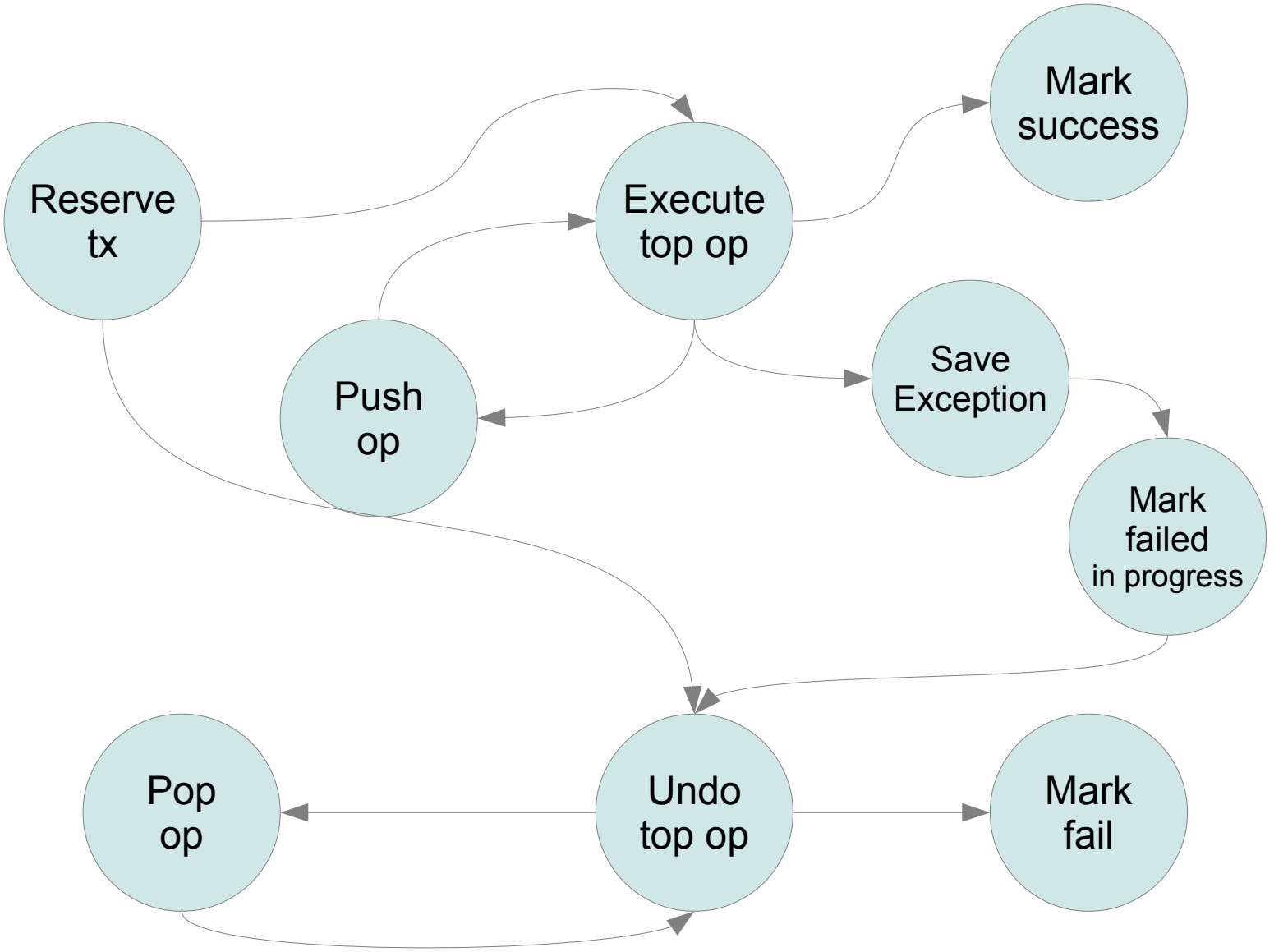
FATE Persistent store

```
public interface TStore {
     long reserve();
     Repo top(long tid);
     void push(long tid, Repo repo);
     void pop(long tid);
     Tstatus getStatus(long tid);
     void setStatus(long tid, Tstatus status);
          .
          .
          .
}
```

Seeding

```
void seedTransaction(long tid, Repo repo){
     if(store.getStatus(tid) == NEW){
          if(store.top(tid) == null)
               store.push(tid, repo);
          store.setStatus(tid, IN_PROGRESS);
     }
}
```

FATE transaction execution



Concurrent table ops

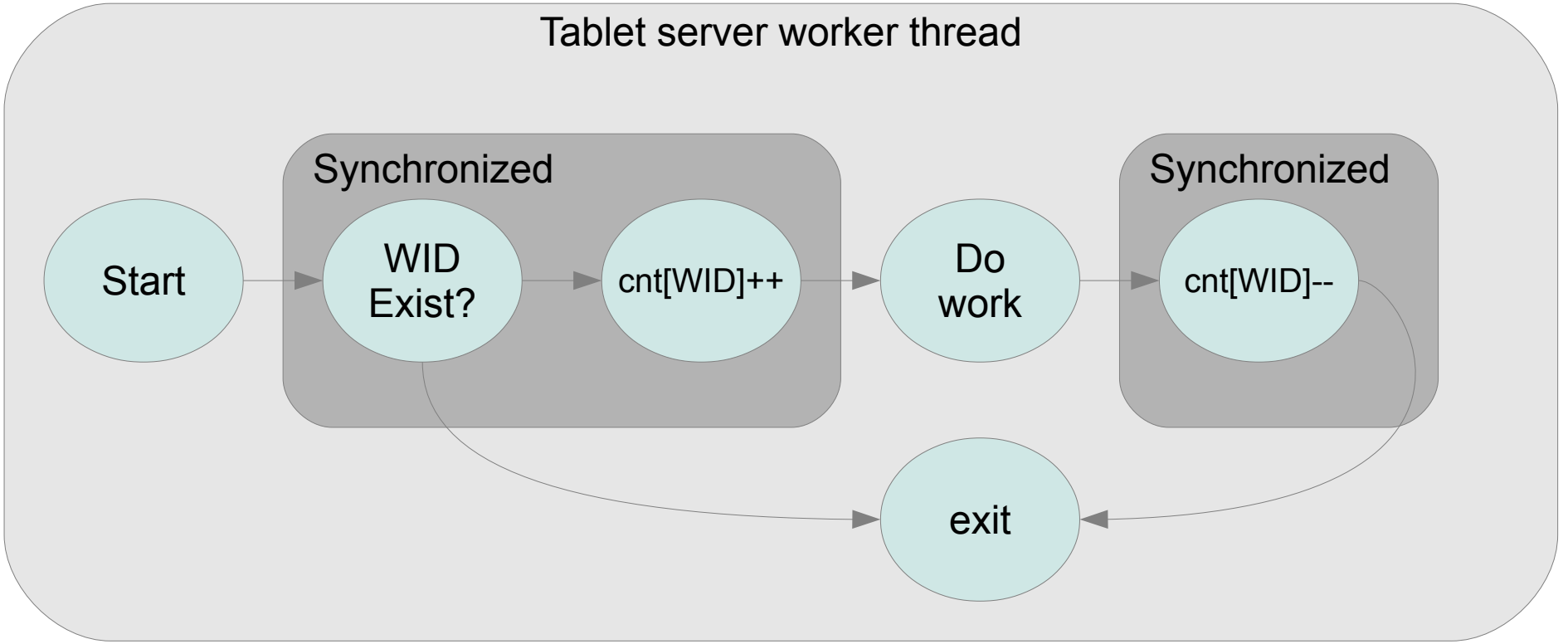
- Per-table read/write lock in zookeeper
- Zookeeper recipe with slight modification :
  - store fate tid as lock data
  - Use persistent sequential instead of ephemeral sequential

Concurrent delete table

- Table deleted during read or write 1.3
  - Most likely, scan hangs forever
  - Rarely, permission exception thrown (seen in test)
- In 1.4 check if table exists when :
  - Nothing in !METADATA
  - See a permission exception

Rogue threads

- Bulk import fate op pushes work to tservers
- Threads on tserver execute after fate op done
- Master deletes WID in ZK, waits until counts 0



Create Table Ops

1. CreateTable
   - Allocates table id
2. SetupPermissions
3. PopulateZookeeper
   - Reentrantly lock table in isReady()
   - Relate table name to table id
4. CreateDir
5. PopulateMetadata
6. FinishCreateTable

Random walk concurrent test

- Perform all table operation on 5 well known tables
- Run multiple test clients concurrently
- Ensure accumulo or test client does not crash or hang
- Too chaotic to verify data correctness
- Found many bugs

Future Work

- Allow multiple processes to run fate operations
- Stop polling
- Hulk Smash Tolerant

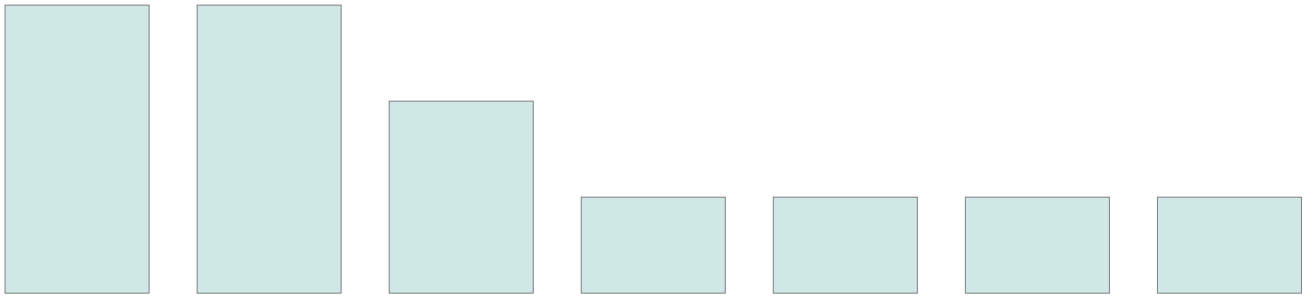
Other 1.4 Features

Tablet Merging

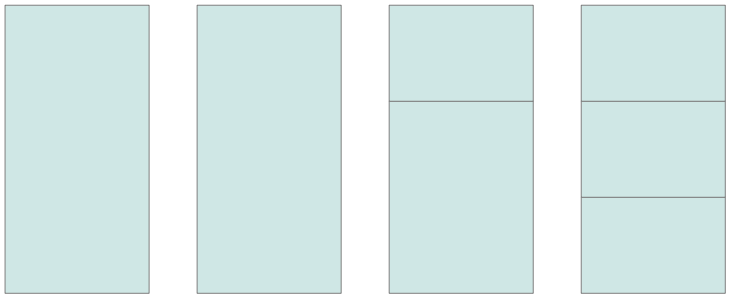
- Splits exists forever; data is often aged off
- Admin requested, not automatic
- Merge by range or size

Merging Minor Compaction

- 1.3: minor compactions always add new files:



- 1.4: limits the total of new files:



Merging Minor Compactions

- Slows down minor compactions
- Memory fills up
- Creates back-pressure on ingest
- Prevents “unable to open enough files to scan”

Table Cloning

- Create a new table using the same read-only files
- Fast
- Testing
  - At scale, with representative data
  - Compaction with a broken iterator: no more data
- Offline and Copy
  - create an consistent snapshot in minutes

Range Operations

- Compact Range
  - Compact all tablets that fall within a row range down to a single file
  - Useful for tail-insert indexes
  - Data purge
- Delete Range
  - Delete all the content within a row range
  - Uses split and will delete whole tablets for efficiency
  - Useful for data purge

Logical Time for Bulk Import

- Bulk files created on different machines will get different actual times (hours)
- Bulk files always contain the timestamp created by the client
- A single bulk import request can set a consistent time across all files
- Implemented using iterators

Roadmap

1.4.1 Improvements

- Snappy, LZ4
- HDFS Kerberos compatability
- Bloom filter improvements
- Map Reduce directly over Accumulo files
- Server side row select iterator
- Bulk ingest support for map only jobs
- BigTop support
- Wikisearch example improvements

Possible 1.5 features

Performance

- Multiple namenode support
- WAL performance improvements
- In-memory map locality groups
- Timestamp visibility filter optimization
- Prefix encoding
- Distributed FATE ops

API

- Stats API
- Automatic deployment of iterators
- Tuplestream support
- Coprocessor integration
- Support other client languages
- Client session migration

Admin/Security

- Kerberos support/pluggable authentication
- Administration monitor page
- Data center replication
- Rollback/snapshot backup/recovery

Reliability

- Decentralize master operations
- Tablet server state model

Testing

- Mini-cluster test harness
- Continuous random walk testing