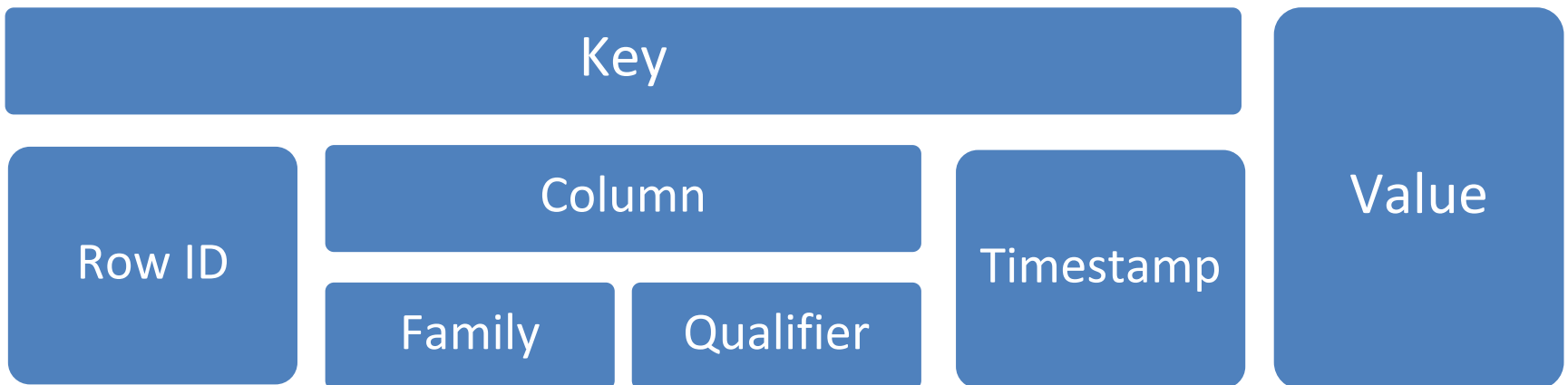


Introduction

- Accumulo is a sparse, distributed, sorted, multi-dimensional map
- Modeled after Google's BigTable design, but with security features built in
- Designed to scale linearly to trillions of records and 10s of petabytes
- Features automatic load balancing, high-availability, fault-tolerance, and dynamic data model

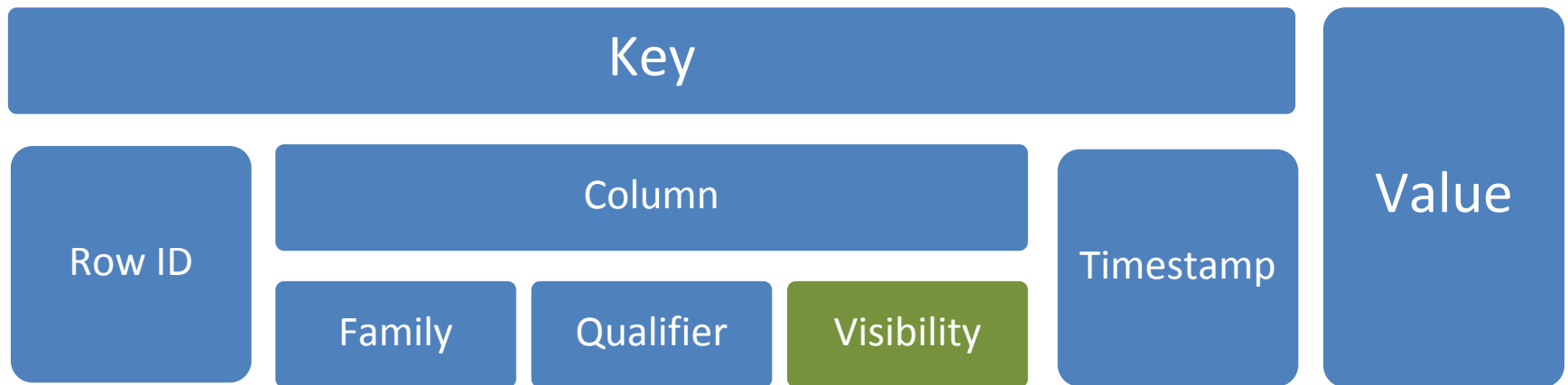
Big Table Design

- Stores records (key/value pairs) in tables which are sorted in lexicographically ascending order
 - All elements of the key/value are byte arrays
 - Except timestamp which is a long (sorted in descending order)
- Records are stored in a table
 - Analogous to a relational database table
 - Table is stored in a distributed fashion across the cluster



Accumulo Design

- Data model features cell level security
- Age off mechanisms
- Powerful abilities to iterate over data and perform complex operations

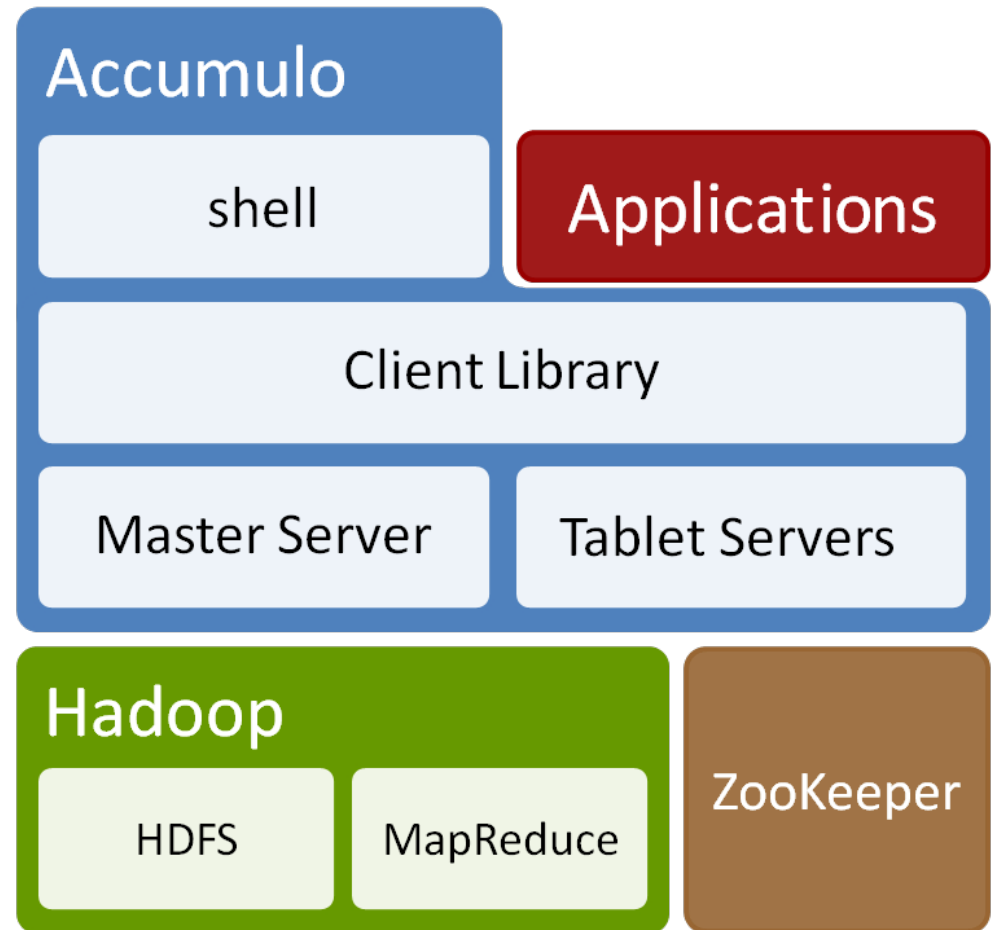


Basic Architecture

Accumulo Architecture

Accumulo

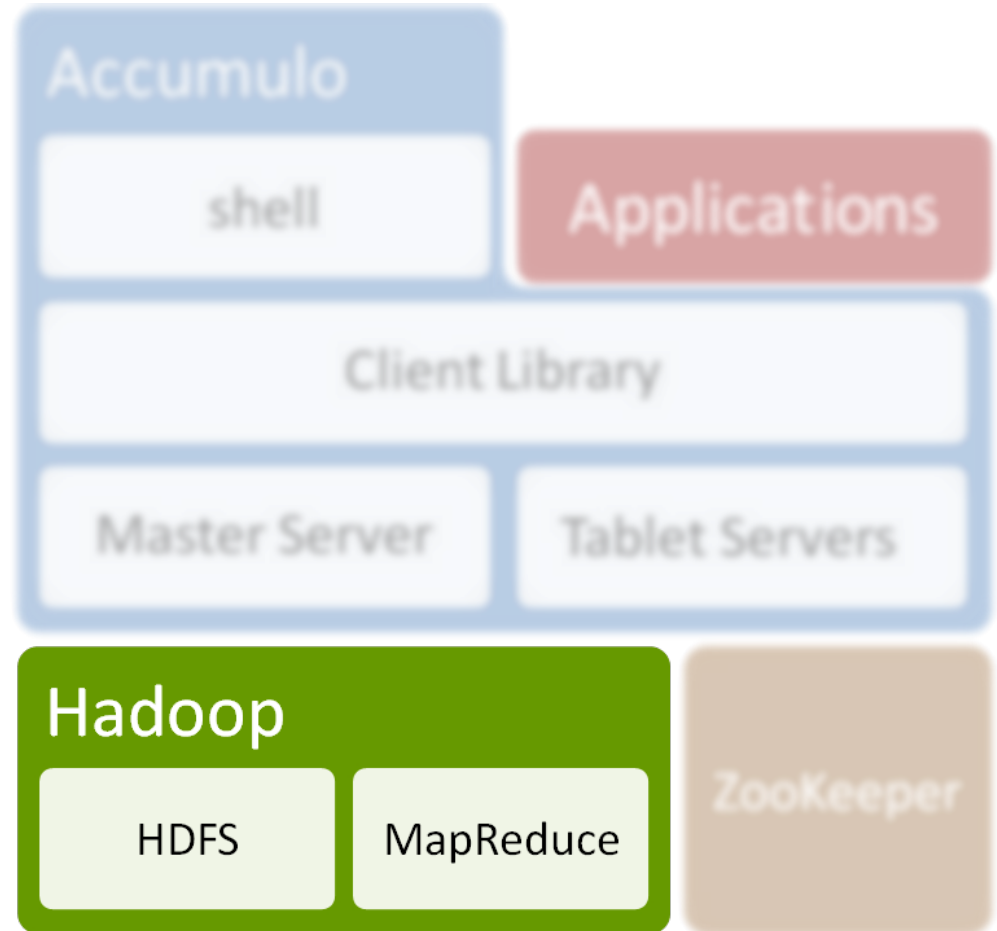
- Written in Java and uses the Hadoop Distributed File System (HDFS) and Apache ZooKeeper to store table data and configuration data
- Supports fault tolerance through replication of tables
- Supports high availability through automatic failover



Accumulo Architecture

Apache Hadoop

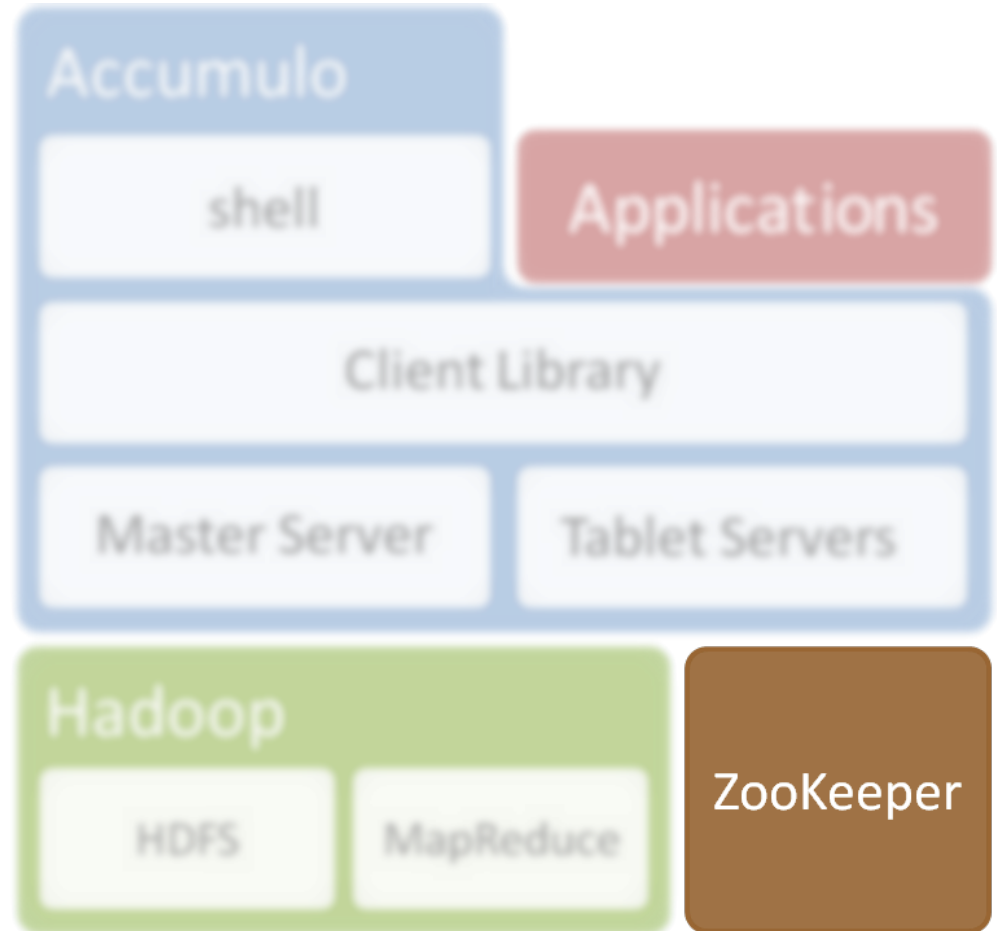
- HDFS is utilized as the underlying filesystem
 - Allows Accumulo to scale
 - Supports redundancy
- MapReduce Integration
 - Allows data stored in Accumulo to be the source and destination of MapReduce jobs



Accumulo Architecture

Apache ZooKeeper

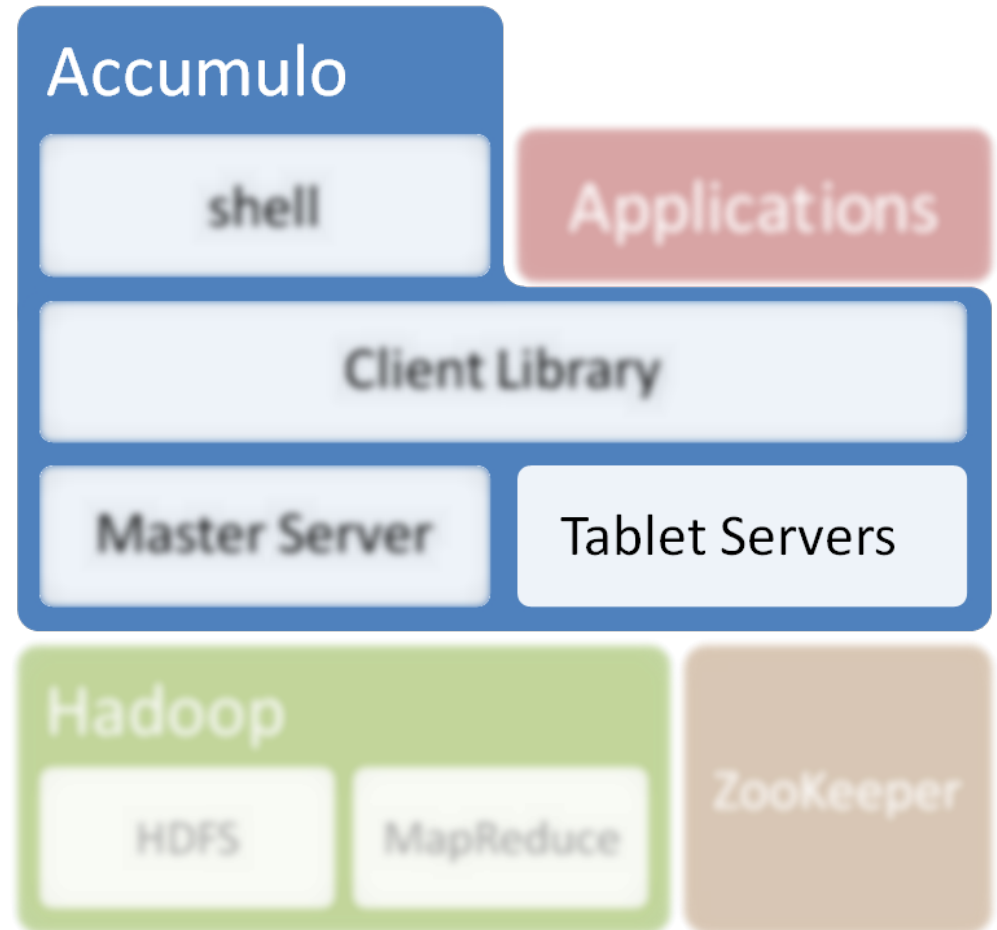
- Highly available synchronization service store that is used for configuration management and naming services
- Uses a quorum to reach consensus across the ZooKeeper nodes



Accumulo Architecture

Tablet Servers

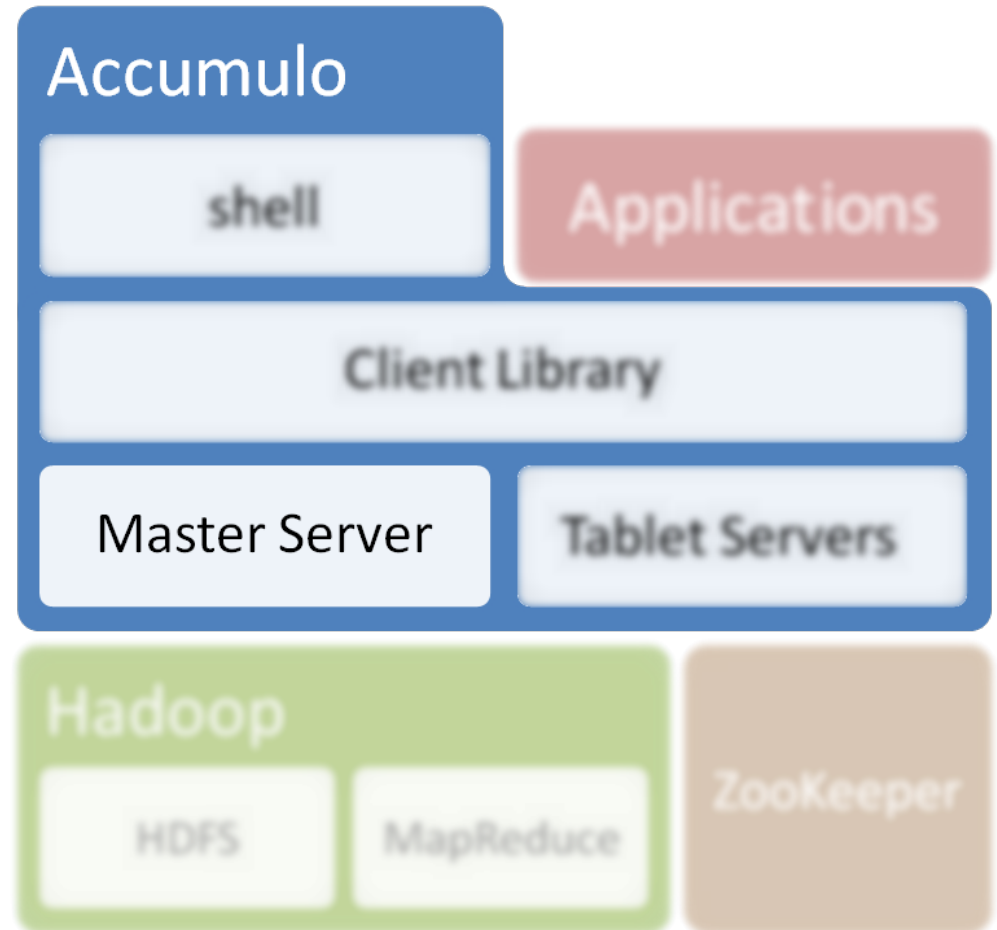
- Each Tablet Server manages some subset of all the tablets
- Services write and read directly from clients
- Uses a write-ahead log to ensure fault tolerance
 - Saved on the nodes local partition (outside of HDFS)
 - Replicated on multiple machines
- Upon ingest, key/value pairs are sorted, stored in memory, and periodically written to HDFS



Accumulo Architecture

Master Server

- Detects and responds to Tablet Server failures by recreating lost tablets
- Automatically balances load by migrating tablets across Tablet Servers
- Ensures each tablet is assigned to only one Tablet Server
- Services table creation, alteration, and deletion requests from clients
- Multiple masters can run concurrently to ensure high availability



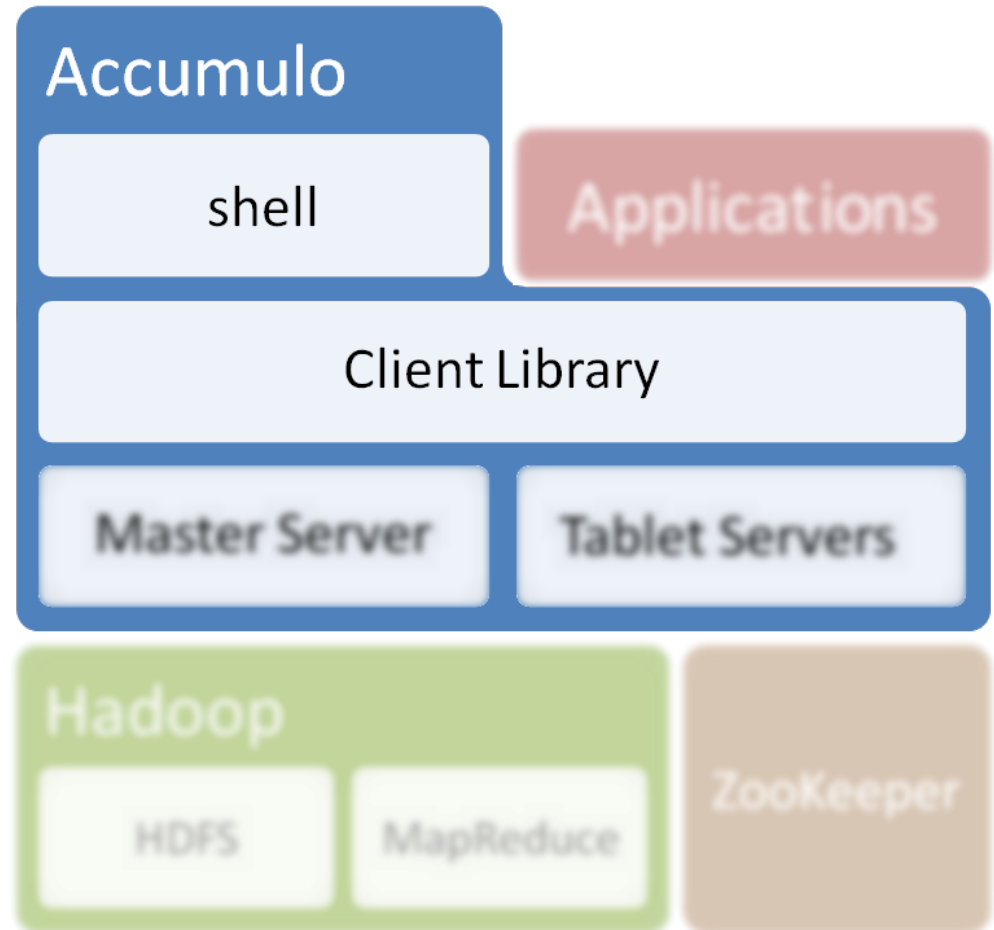
Accumulo Architecture

Client API

- Java API that contains support for finding servers, managing tablets, and communicating with Tablet Servers to read and write key/value pairs

Shell

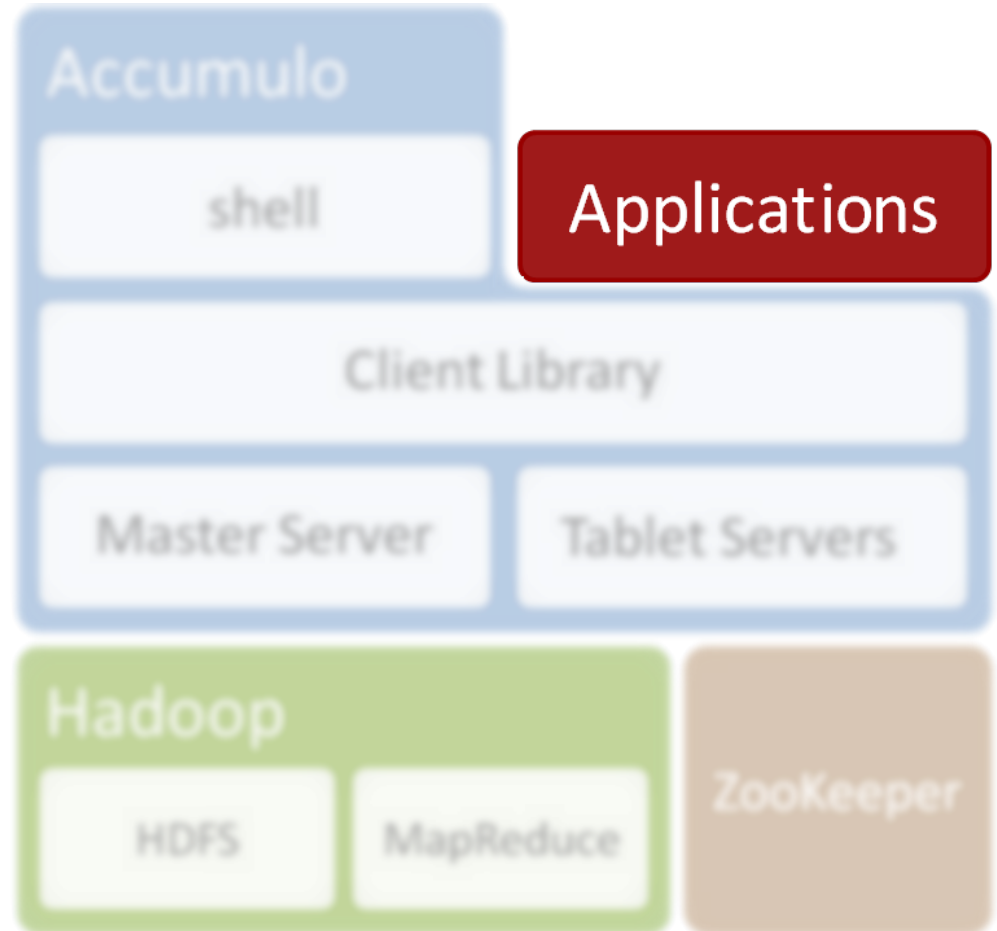
- Used to manage the Accumulo instance
- Allows individual table configuration to be viewed and configured
- Supports insert, update, and deletion of key/value pairs



Accumulo Architecture

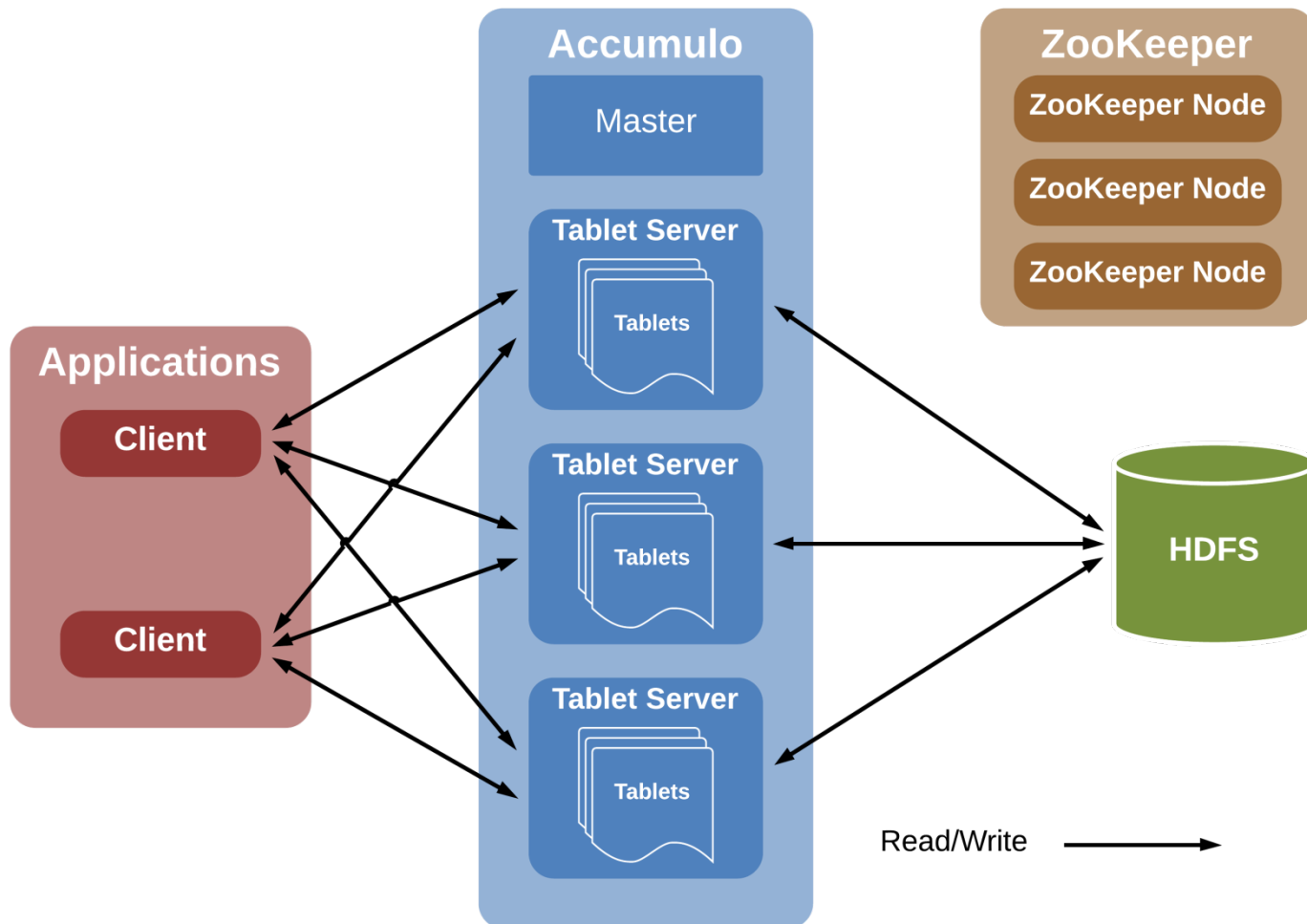
Applications

- Custom applications utilize the Accumulo APIs
- Applications can be in many forms
 - Stand-alone Java program
 - MapReduce job
 - RESTful web service
 - Etc.

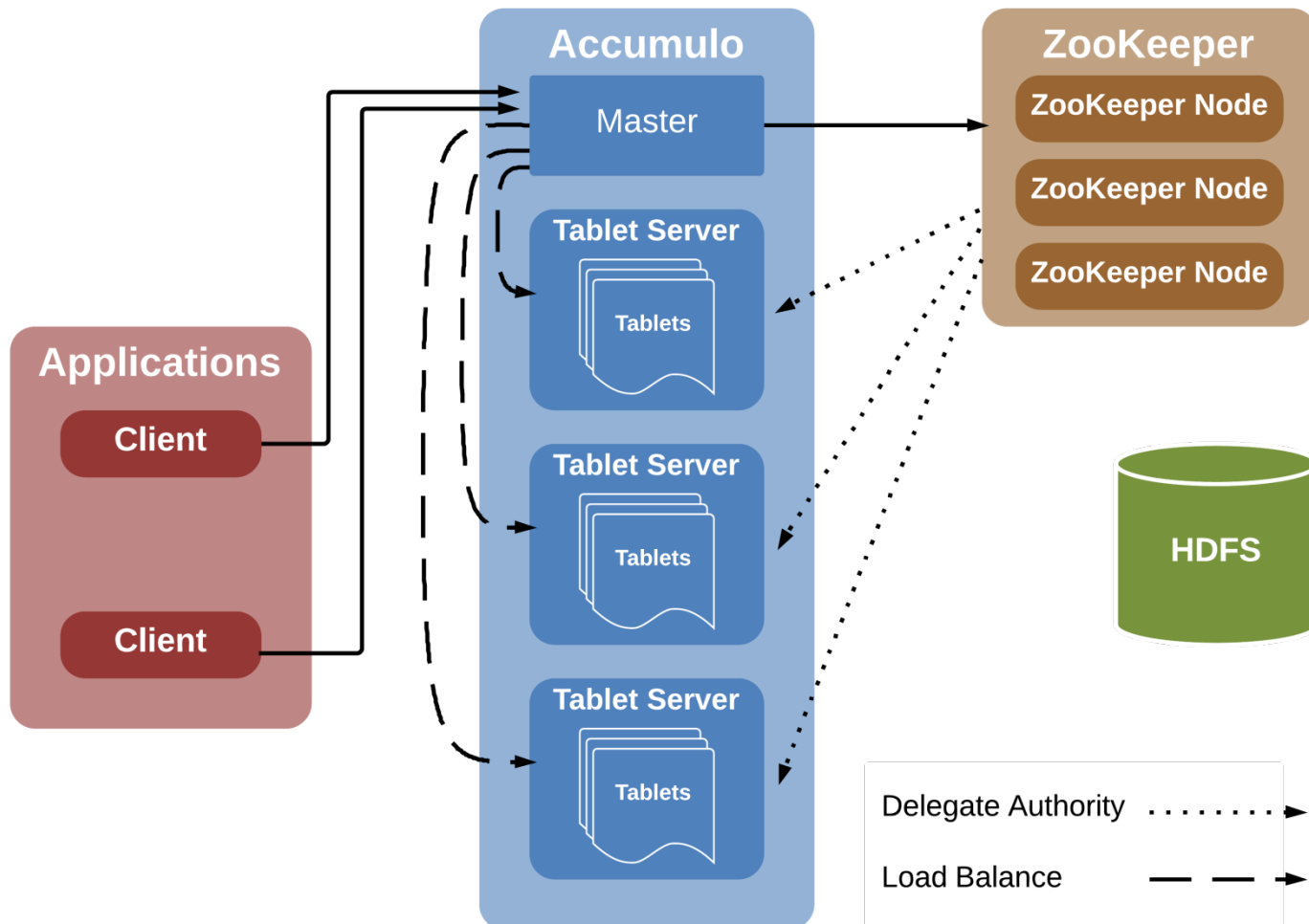


Advanced Architecture

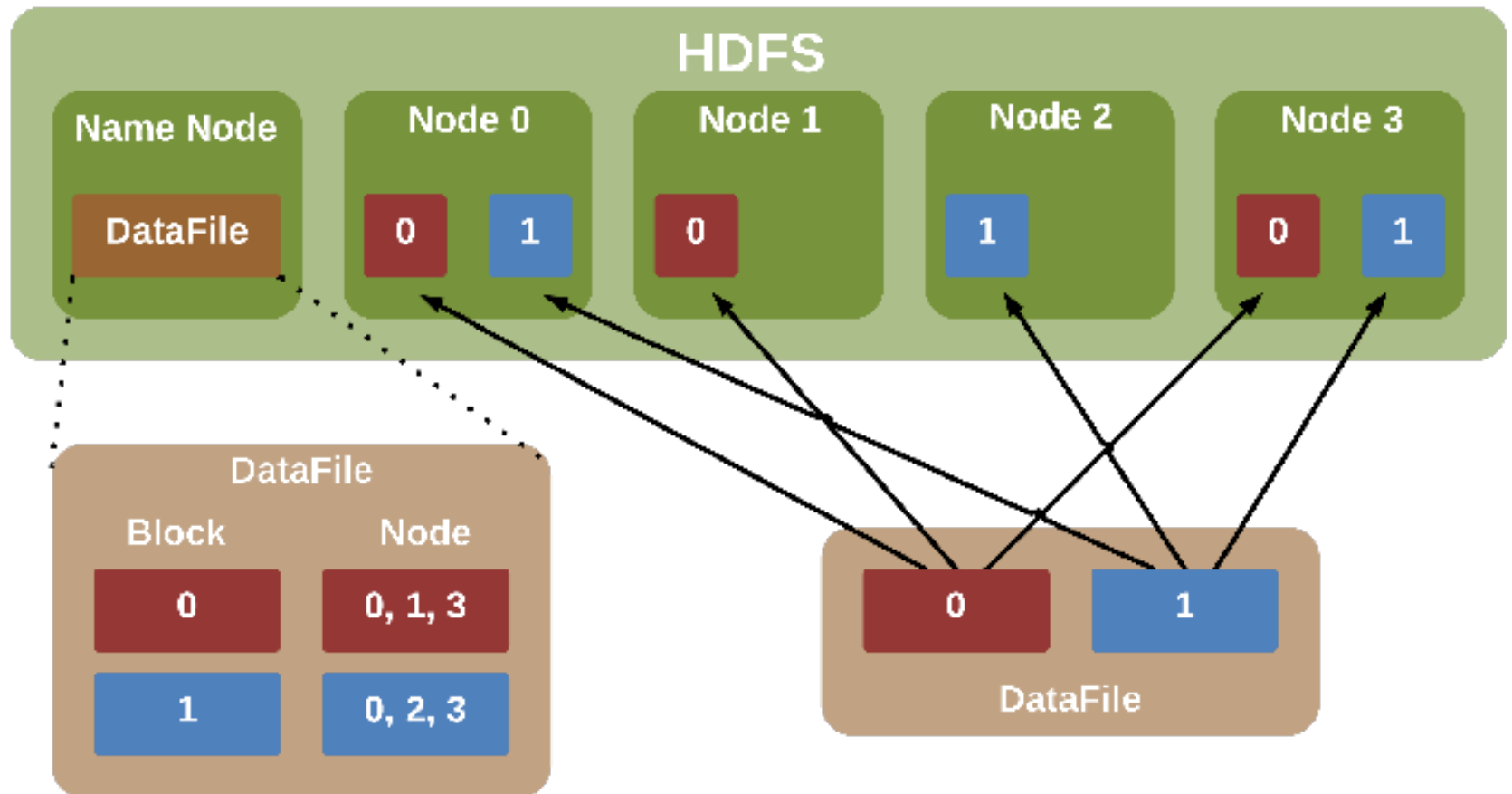
Accumulo Architecture



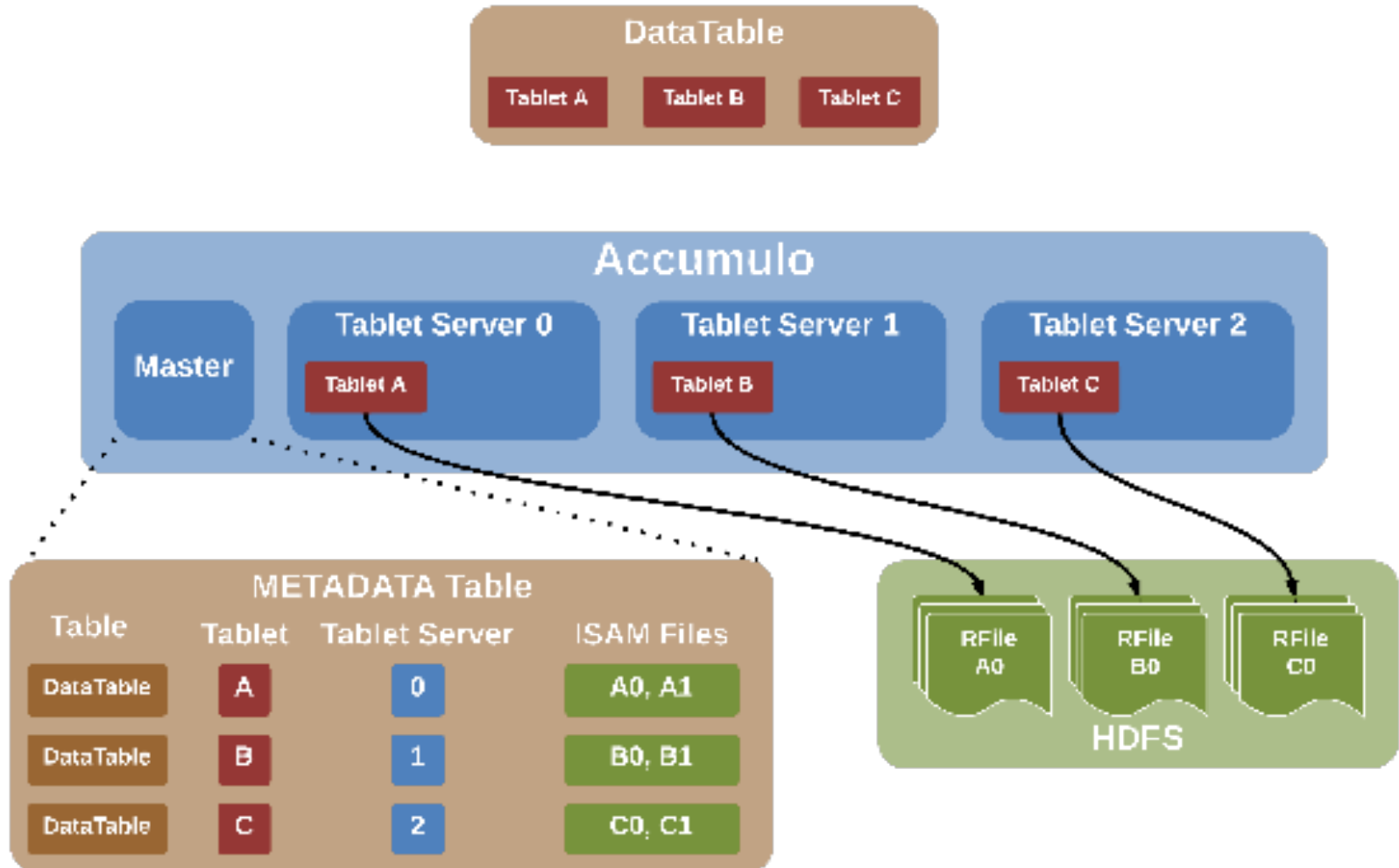
Accumulo Architecture



HDFS Architecture



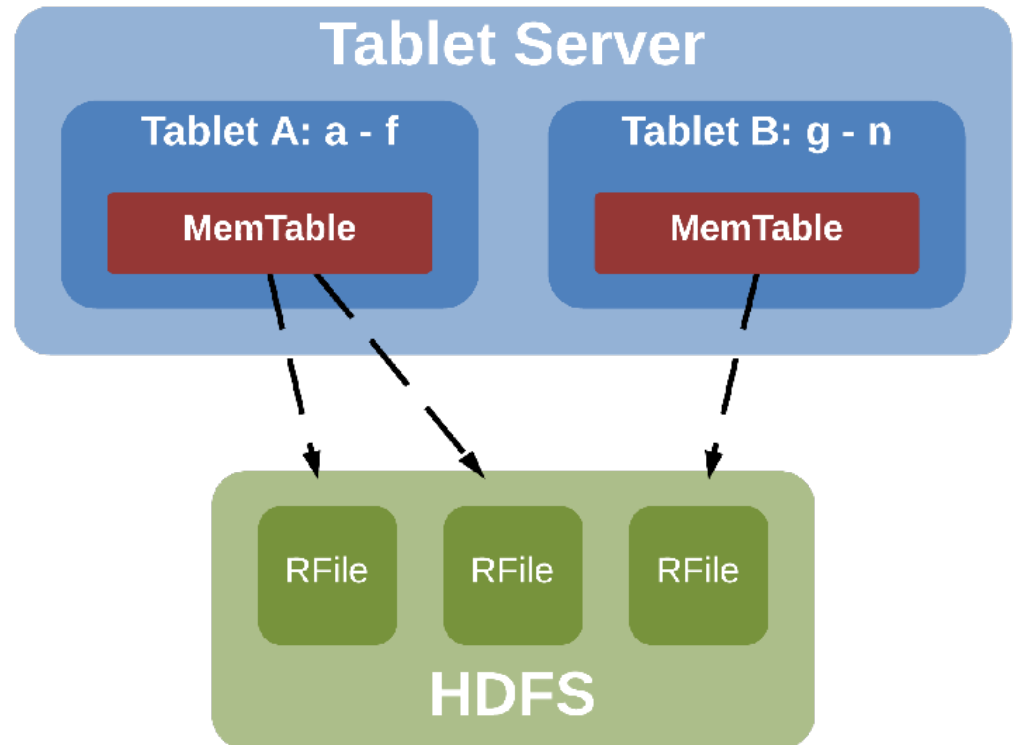
Accumulo Architecture



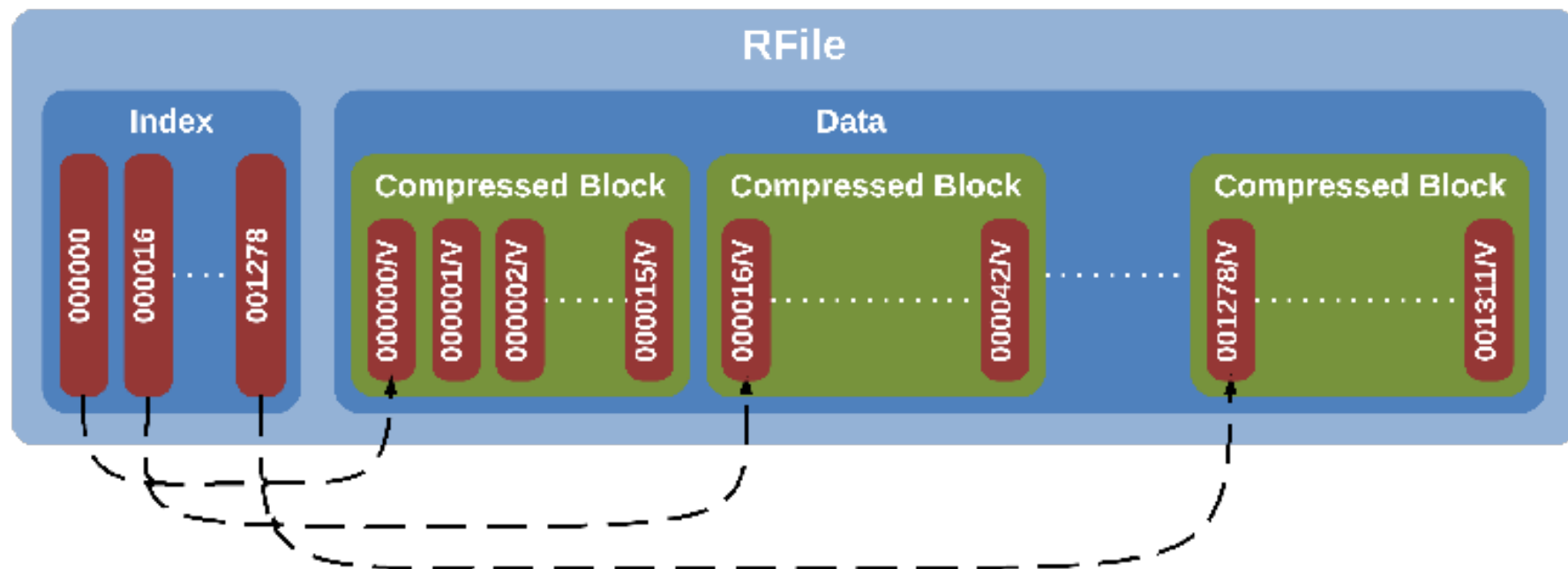
Tablet Server Architecture

Tablet Servers

- Tablet Servers can serve multiple tablets
- Each Tablet serves a specific range
- Each Tablet has a MemTable which is an in-memory map that serves as a buffer for key/value pairs before they get written out to an RFile on HDFS
- key/value pairs can exist in the MemTable and RFile simultaneously
- Queries will return key/value pairs from the MemTable and RFile



RFiles



- The file format used to store Accumulo key/value pairs on HDFS
- RFiles contain an index section that holds references to the first key/value pair in a compressed block
- Compressed Blocks store key/value pairs in lexicographic sorted order

Exercise 1

Accumulo Shell

Exercise 1

- Objective
 - Start the Accumulo server and use Accumulo shell commands to create tables, insert data, and view data
- Tasks
 - Start the Accumulo Server
 - Log in to the Accumulo shell interpreter
 - Become familiar with Accumulo shell commands

Start/stop Accumulo

- Start the Accumulo service

```
$sudo service accumulo start
```

- Stop the Accumulo service

```
$sudo service accumulo stop
```

- Accumulo Instance Overview: localhost:50095

Access the Accumulo Shell

- Start the Accumulo shell by using the following command:

```
$sudo accumulo shell -u root
```

Accumulo Basic Shell Commands

Command	Description
help	Provide information about the available commands
tables	Display a list of all existing tables
table	Switch to the specified table
createtable	Creates a new table with the specified name
deletetable	Deletes the table with the specified name
insert	Inserts a record
scan	Scans the table and displays the resulting records
delete	Deletes a record from the table
getauths	Displays the maximum scan authorizations for a user
setauths	sets the maximum scan authorizations for a user
quit	Exits the shell

Data Layout

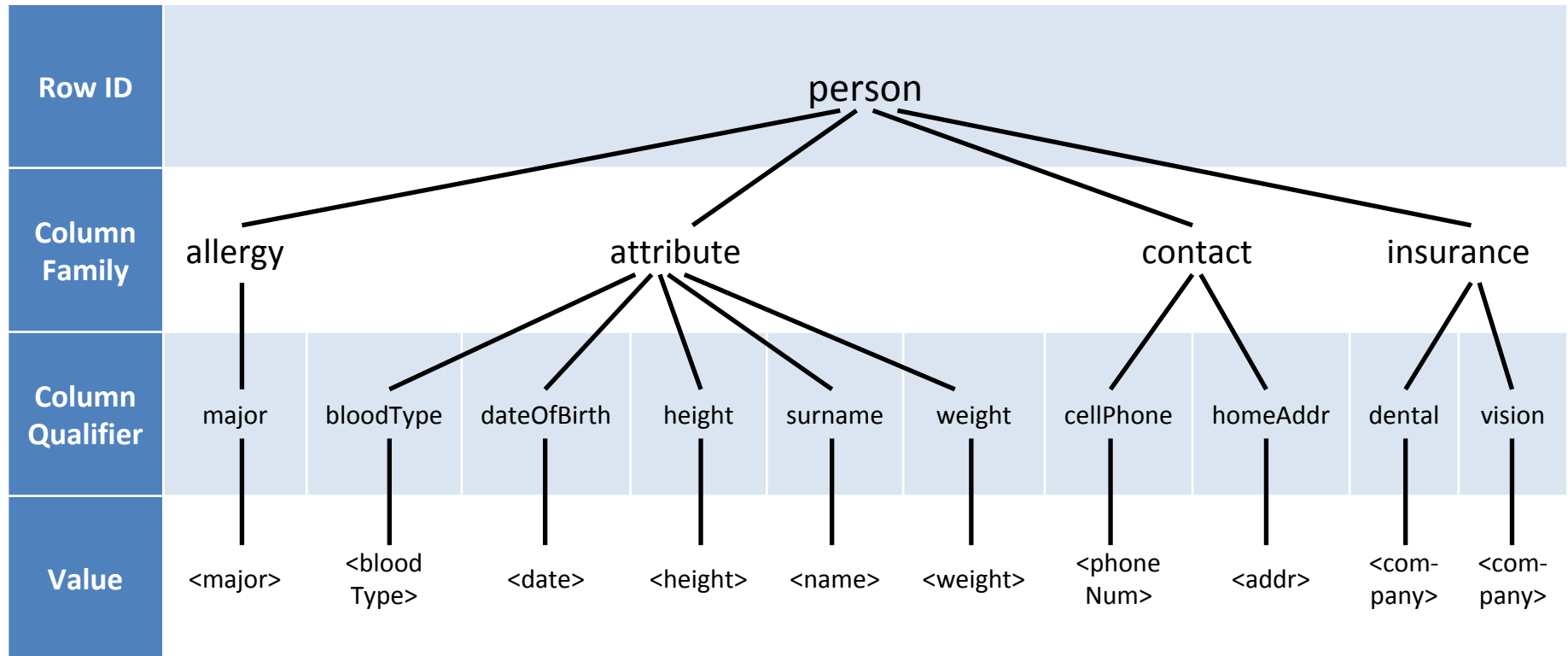
Data Layout

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
bob	attribute	dateOfBirth	public	Jul 2013	19850509
bob	attribute	height	public	Jun 2012	5'11"
bob	insurance	dental	private	Sep 2009	MetLife
jane	attribute	bloodType	public	Jul 2011	ab-
jane	attribute	surname	public	Aug 2013	doe
jane	contact	cellPhone	public	Dec 2010	(808) 345-9876
jane	insurance	vision	private	Jan 2008	VSP
john	allergy	major	private	Feb 1988	amoxicillin
john	attribute	weight	public	Sep 2013	180
john	contact	homeAddr	public	Mar 2003	34 Baker LN

2-D Table Representation

	allergy: major	attrib ute: blood Type	attribute: dateOfBir th	attrib ute: heigh t	attrib ute: surna me	attrib ute: weig ht	contact: cellPhone	contact: homeAddr	insuran ce: dental	insur ance: vision
bob	-	-	19850509	5'11"	-	-	-	-	MetLife	-
jane	-	ab-	-	-	doe	-	(808) 345-9876	-	-	VSP
john	amoxicillin	-	-	-	-	180	-	34 Baker LN	-	-

Hierarchical Decomposition



Hierarchical Decomposition

Row ID	bob			jane			
Column Family	attribute		insurance	attribute		contact	insurance
Column Qualifier	dateOfBirth	height	dental	bloodType	surname	cellPhone	vision
Value	19850509	5'11"	MetLife	ab-	doe	(808) 345-9876	VSP

JSON Representation

Row ID -----	Column Family -----	Column Qualifier -----	Value -----
"bob": {	"attribute": {	"dateOfBirth":	19850509,
		"height":	5'11"
	},		
	"insurance": {		
		"dental":	MetLife
	}		
},			
"jane": {	"attribute": {	"bloodType":	ab-
		"surname":	Doe
	},		
	"contact": {		
		"cellPhone":	(808) 345-9876
	},		
	"insurance": {		
		"vision":	VSP
	}		
}			

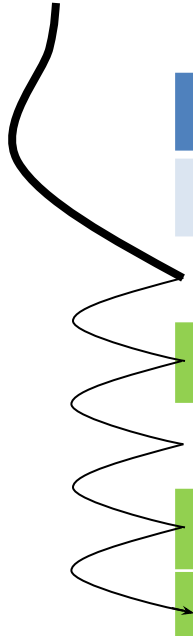
Locality Groups

Locality Group

Row ID	Column Family	Column Qualifier	Value
com.abc	content	-	<!DOCTYPE ...
com.cnbc	content	-	<!DOCTYPE ...
com.cnbc	link	google.com	to cnbc
com.cnn	content	-	<!DOCTYPE ...
com.cnn	link	google.com	cnn.com
com.cnn	link	yahoo.com	cnn.com
com.nbc	content	-	<!DOCTYPE ...
com.nbc	link	yahoo.com	NBC

Locality Group

Query: link data
for CNBC and CNN



Row ID	Column Family	Column Qualifier	Value
com.abc	content	-	<!DOCTYPE ...
com.cnbc	content	-	<!DOCTYPE ...
com.cnbc	link	google.com	to cnbc
com.cnn	content	-	<!DOCTYPE ...
com.cnn	link	google.com	cnn.com
com.cnn	link	yahoo.com	cnn.com
com.nbc	content	-	<!DOCTYPE ...
com.nbc	link	yahoo.com	NBC

Locality Group

Query: link data
for CNBC and CNN

Row ID	Column Family	Column Qualifier	Value
com.abc	content	-	<!DOCTYPE ...
com.cnbc	content	-	<!DOCTYPE ...
com.cnn	content	-	<!DOCTYPE ...
com.nbc	content	-	<!DOCTYPE ...
...			
com.cnbc	link	google.com	to cnbc
com.cnn	link	google.com	cnn.com
com.cnn	link	yahoo.com	cnn.com
com.nbc	link	yahoo.com	NBC

Locality Groups

Table

Locality Group Name
(arbitrary)

```
> setgroups -t webcrawl metagrps=link,time contgrps=content,pictures  
> getgroups -t webcrawl  
metagrps=link,time  
contgrps=content,pictures
```

Column Families

Column Visibility

Column Visibility Syntax

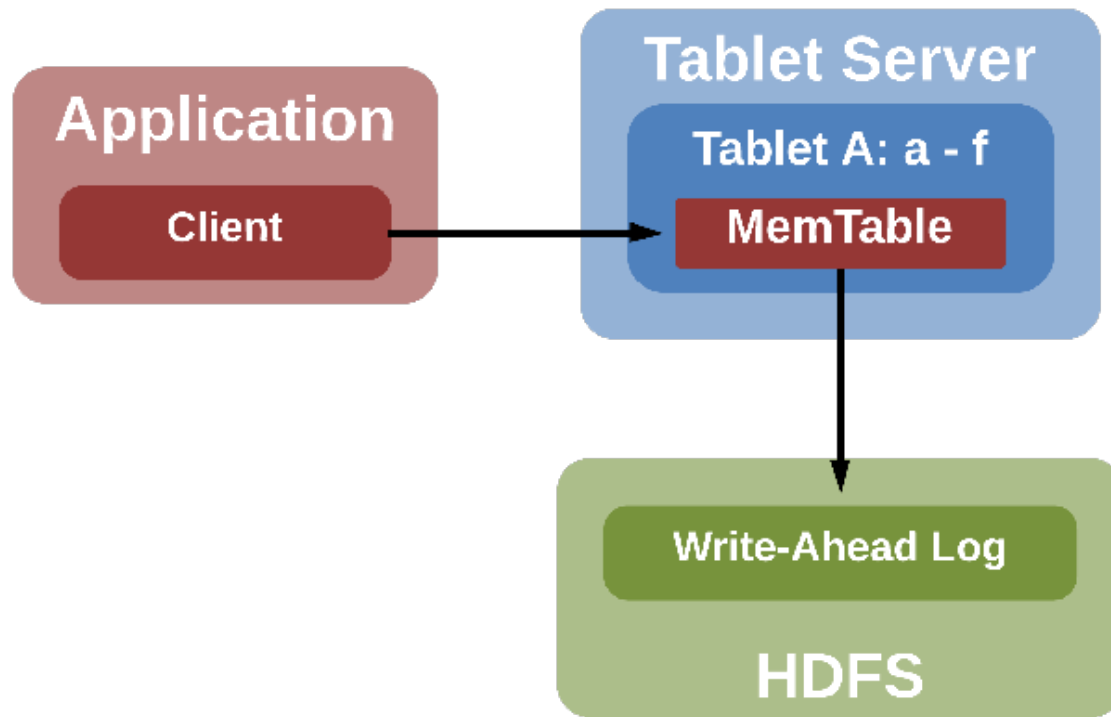
Label	Description
A & B	Both 'A' and 'B' are required
A B	Either 'A' or 'B' is required
A & (C B)	'A' and 'C' or 'A' and 'B' is required
A (B & C)	'A' or 'B' and 'C' is required
(A B) & (C & D)	?
A & (B & (C D))	?

Column Visibility Examples

- Social Network Data
admin | ownerId | (friendId & photoId)
- Medical Data
(doctorId & practiceId) | patientId
- Corporate Data
ceo | cfo | cio | (auditorId & country)

Writing to Accumulo

Writing to Accumulo



Data Flow →

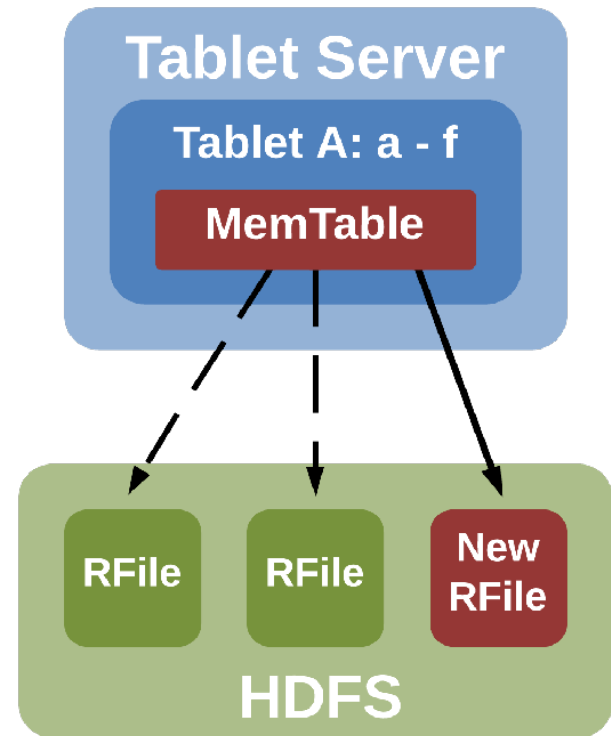
Compactions

- Minor Compaction
 - The process of flushing records held in a MemTable to HDFS
- Major Compaction
 - The process of combining RFiles, referenced by a Tablet, into a single more optimized RFile

Writing to Accumulo

Minor Compaction

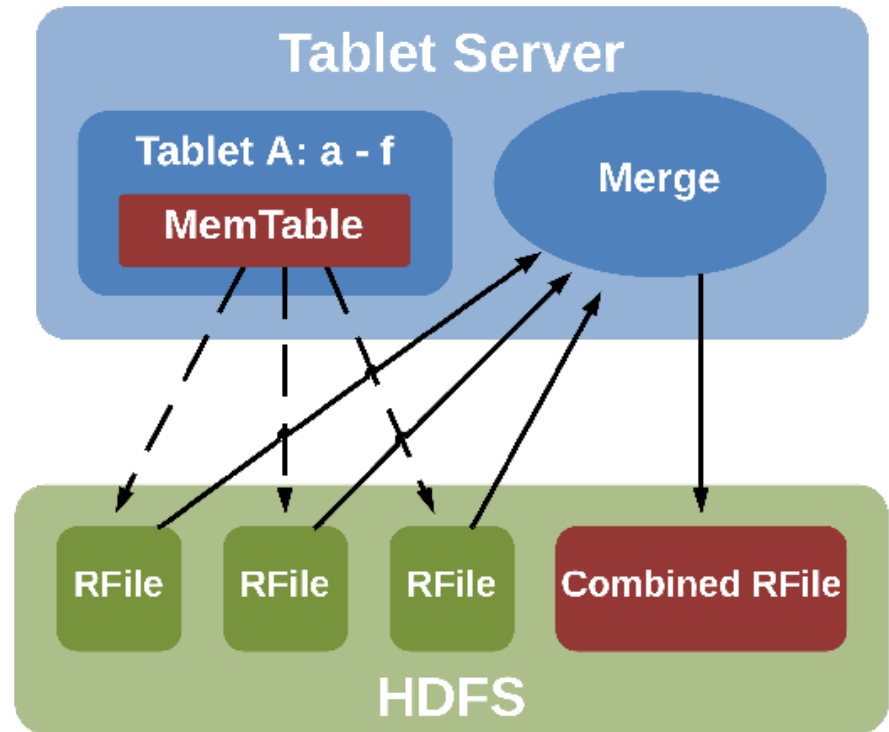
- Once the MemTable reaches a certain size the TabletServer writes out the sorted key/value pairs to a file in HDFS called RFile
- Each Minor Compaction creates one new RFile
- After the RFile is written to HDFS, a new MemTable is then created and the compaction is recorded in the Write-Ahead Log



Writing to Accumulo

Major Compaction

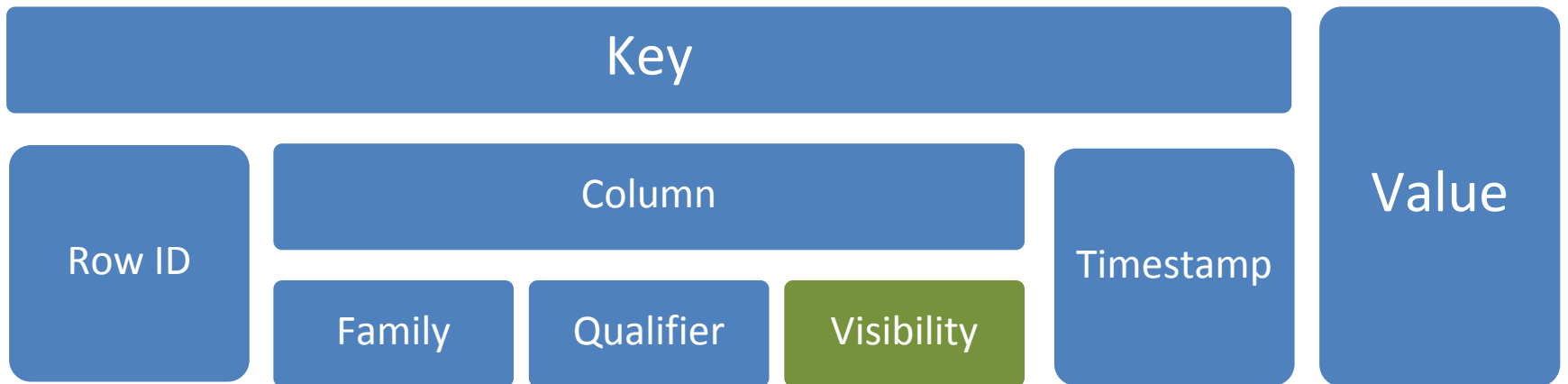
- Periodically, the TabletServer will combine a set of RFiles to minimize the number of RFiles tracked per tablet
- During the merge, any key/value pairs suppressed by the a delete entry are omitted
- After a Major Compaction occurs the previous RFiles are set to be cleaned up by the Garbage Collector



Exercise 2

- Objective
 - Create a Java program that takes twitter data as input and stores the data in an Accumulo table. The program should parse the twitter data, connect to the local Accumulo instance, create a table, and store the twitter data in the newly created Accumulo table.
- Tasks
 - Configure a ZooKeeperInstance object and Accumulo Client Connector object to connect to the local Accumulo instance
 - Create an Accumulo table if it does not already exist
 - Create and configure a BatchWriter object
 - Using a Mutation object store information for each tweet
 - Check the output of the program by using the Accumulo shell

Accumulo Design Review



Twitter Data Table

Row ID	Column Family	Column Qualifier	Visibility	Timestamp	Value
{tweetID}	text	-	-	-	{tweet text}
{tweetID}	uid	{userID}	-	-	{user Screen Name}

Writing Data to Accumulo

- Before writing to Accumulo, clients must identify the Accumulo Instance to which they will be connecting

```
ZooKeeperInstance instance = new ZooKeeperInstance(  
    instanceName,    // The name of the Accumulo instance  
    zooServers);    // A list of ZooKeeper servers  
  
Connector connector = new Connector(  
    instance,        // The previously created instance  
    userName,        // The user name  
    userPass.getBytes()); // The user password
```

Writing Data to Accumulo

- Mutations are used to represent all the changes to the columns of a single row
- Changes are made atomically in the TabletServer

```
Mutation mutation = new Mutation(rowID);
```

```
mutation.put(  
    colFam,  
    colQual,  
    colVis,  
    timestamp,  
    value);
```

Writing Data to Accumulo

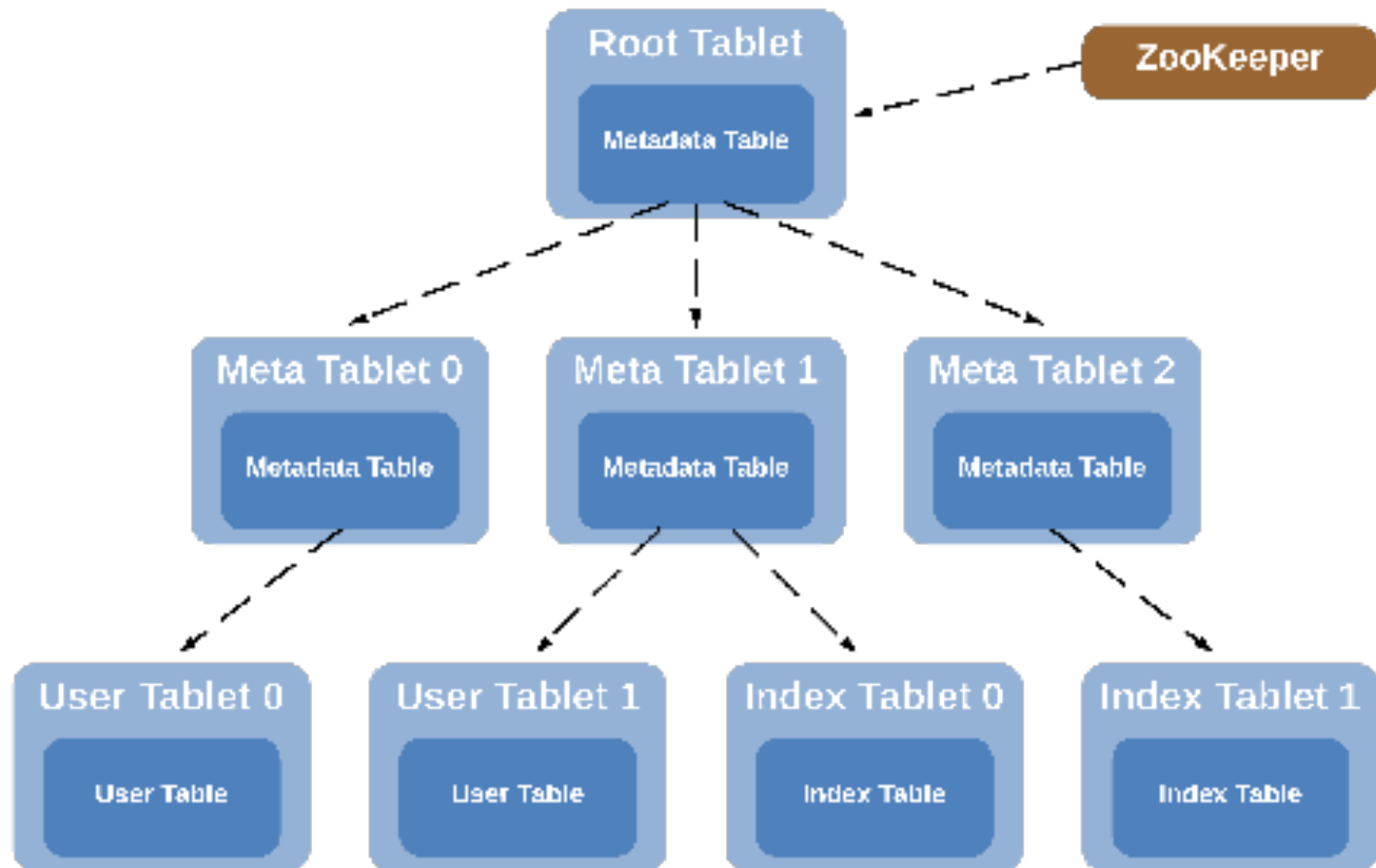
- A BatchWriter can be used to send Mutations to TabletServers

```
long memBuf = 1000000L; // bytes to store before sending a batch
long timeout = 1000L;    // ms to wait before sending
int numThreads = 10;     // Number of concurrent threads
```

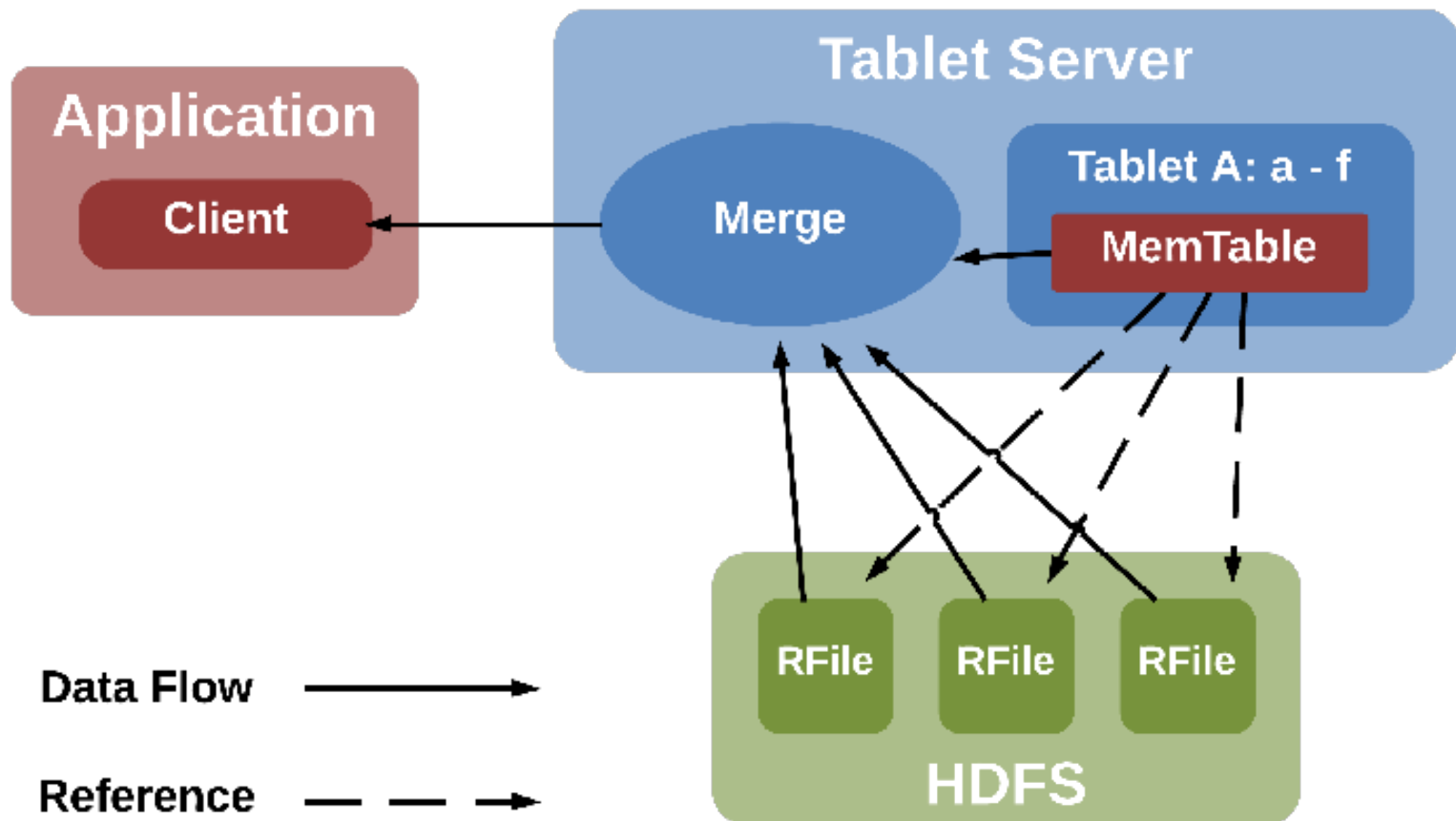
```
BatchWriter writer = connector.createBatchWriter(
    "table",          // The table to write to
    memBuf,
    timeout,
    numThreads);
```


Reading from Accumulo

Reading From Accumulo



Reading from Accumulo



Exercise 3

Reading from Accumulo

Exercise 3

- Objective
 - Create a Java program that reads twitter data stored in Accumulo. The program should connect to the local Accumulo instance, read from the table created by the AccumuloWriter program, and only display entries that contain tweet text.
- Tasks
 - Configure a ZooKeeperInstance object and Accumulo Client Connector object to connect to the local Accumulo instance
 - Check to ensure an Accumulo table containing twitter data already exist
 - Create and configure a Scanner object
 - Iterate through the Scanner and display each tweets ID and text

Reading Data from Accumulo

- A Scanner can be used to read records from a table

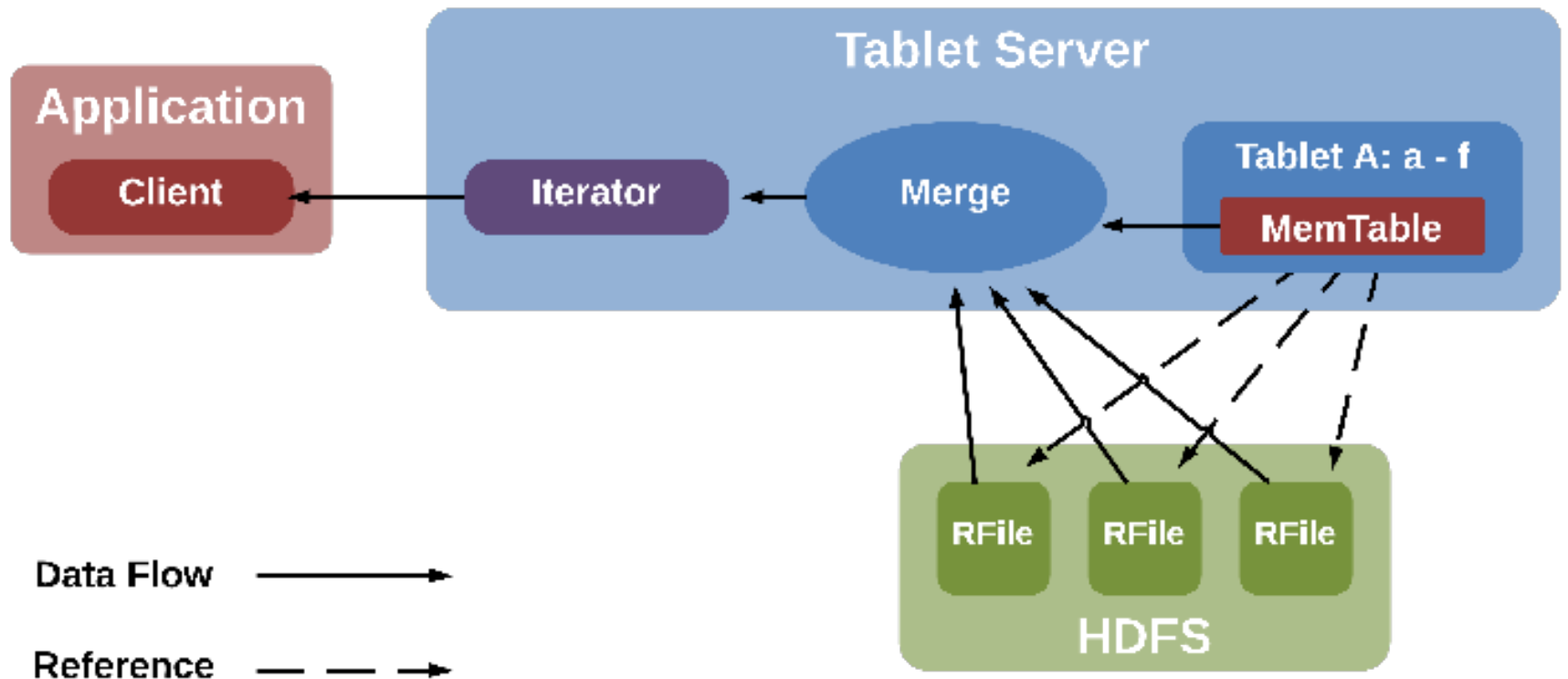
```
Scanner scanner = conn.createScanner(  
    "table",                // Name of the table to write to  
    new Authorizations()); // The users visibility  
  
// Getting the key and value for each row  
for(Entry<Key,Value> entry : scanner) {  
    String row = e.getKey().getRow();  
    Value value = e.getValue();  
}
```

Iterators

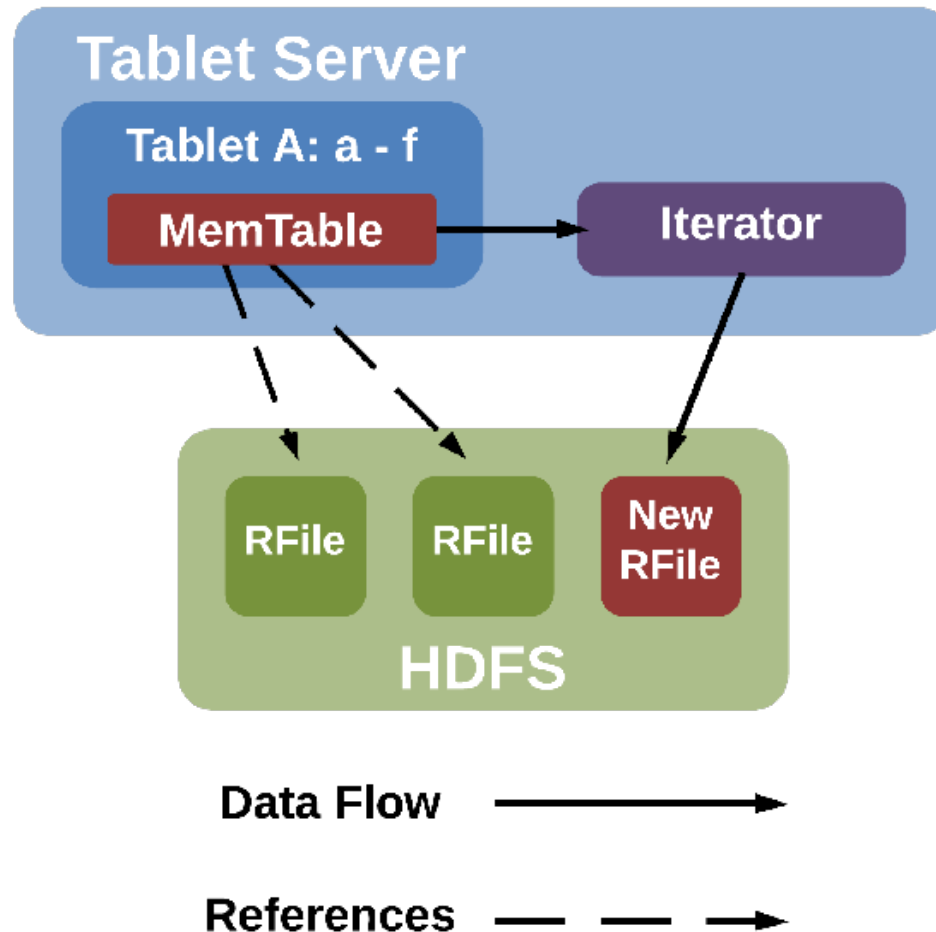
Iterators

- Iterators provide a modular mechanism for adding functionality that can be executed at 3 places in the Accumulo Stack
 1. Scan Time
 2. Minor Compaction
 3. Major Compaction
- Multiple iterators are included with the Accumulo distribution and can be found in the **org.apache.accumulo.core.iterators.user** package
- Custom iterators can be created using the Iterator API

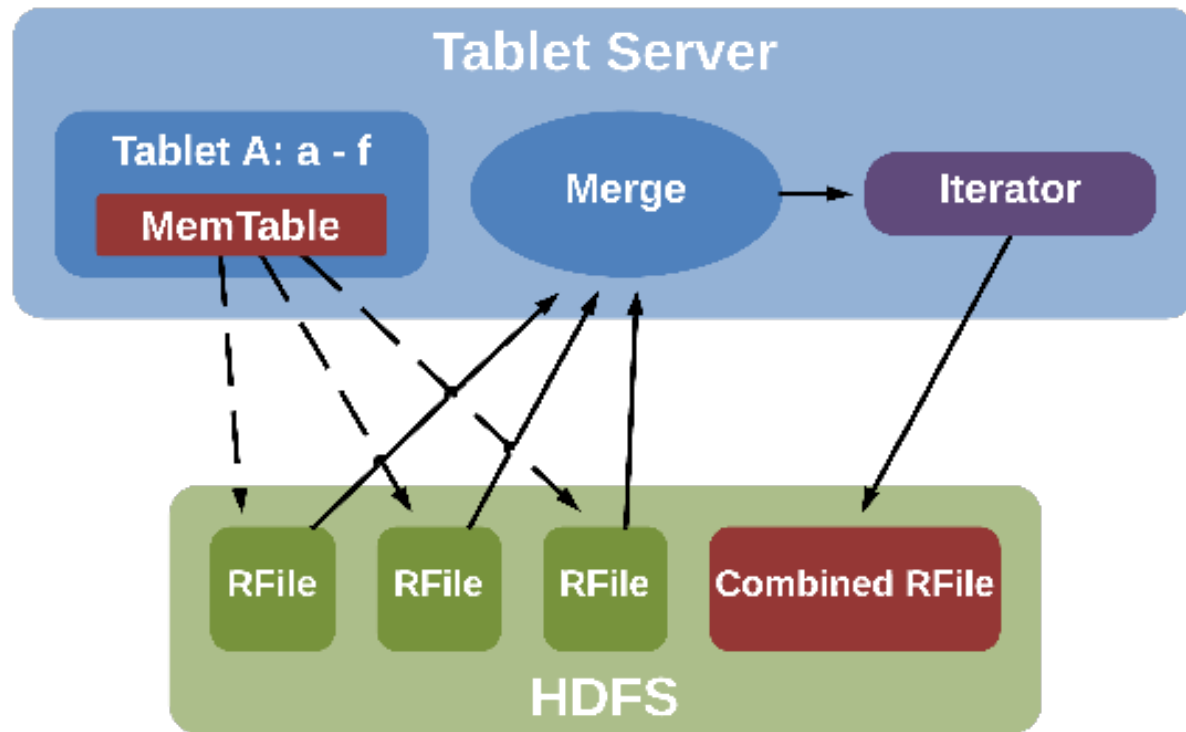
Scan Time Iterator



Minor Compaction Iterator



Major Compaction Iterator



Data Flow —————→

References - - - - ->

Iterator Types

Versioning – Configure the number of identical key/value pairs to store

Filtering – Apply arbitrary filtering to key/value pairs

Combiners – Aggregate values from any key that shares a Row ID, Column Family, and Column Qualifier

Viewing Applied Iterators

Table
Name

Search
Term

```
> config -t newTable -f iterator
```

SCOPE	NAME	VALUE
table	table.iterator.majc.vers	20,org.apache.accumulo.core.iterators.user.VersioningIterator
table	table.iterator.majc.vers.opt.maxVersions	1
table	table.iterator.minc.vers	20,org.apache.accumulo.core.iterators.user.VersioningIterator
table	table.iterator.minc.vers.opt.maxVersions	1
table	table.iterator.scan.vers	20,org.apache.accumulo.core.iterators.user.VersioningIterator
table	table.iterator.scan.vers.opt.maxVersions	1

Versioning Iterator

Given multiple version of the same row, what operations can we perform?

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
bob	attribute	height	public	1005	5'11"
bob	attribute	height	public	1004	5'5"
bob	attribute	height	public	1003	5'
bob	attribute	height	public	1002	4'10"
bob	attribute	height	public	1001	4'9"
bob	attribute	height	public	1000	4'3"

Versioning Accumulo Shell

Display the latest n versions of a key/value pair

```
> config -t mytable -s table.iterator.scan.vers.opt.maxVersions=3  
> config -t mytable -s table.iterator.minc.vers.opt.maxVersions=3  
> config -t mytable -s table.iterator.majc.vers.opt.maxVersions=3
```

Versioning Iterator

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
bob	attribute	height	public	1005	5'11"
bob	attribute	height	public	1004	5'5"
bob	attribute	height	public	1003	5'
bob	attribute	height	public	1002	4'10"
bob	attribute	height	public	1001	4'9"
bob	attribute	height	public	1000	4'3"

Logical Time

- If milliseconds are used for the timestamp clock skew between TabletServers could cause keys to be stored out of order
- Logical time is a counter that is increased every time a key/value pair is written to Accumulo
- A table can only be configured to use logical time when the table is created

```
> createtable -t1 <tableName>
```

Deletes

- A special kind of versioning Iterator
- Special keys in Accumulo that are stored along with all of the other key/value pairs
- After a delete is inserted, Accumulo will not return any data with a timestamp less than or equal to the delete key
- During Major Compactions, any key/value pairs with timestamps equal to or older than the delete key are omitted from the Compaction and are eventually cleaned up by the Garbage Collector

Filtering Iterator

- AgeOff
- ColumnAgeOff
- Timestamp
- ReqVis
- RegEx

Age-Off Iterator

Current Time: 1102

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value	
bob	attribute	height	public	1005	5'11"	Entries < 100s old
bob	attribute	height	public	1004	5'5"	
bob	attribute	height	public	1003	5'	
bob	attribute	height	public	1002	4'10"	
bob	attribute	height	public	1001	4'9"	Entries > 100s old
bob	attribute	height	public	1000	4'3"	

Age-Off Iterator

Current Time: 1103

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value	
bob	attribute	height	public	1005	5'11"	Entries < 100s old
bob	attribute	height	public	1004	5'5"	
bob	attribute	height	public	1003	5'	
bob	attribute	height	public	1002	4'10"	Entries > 100s old
bob	attribute	height	public	1001	4'9"	
bob	attribute	height	public	1000	4'3"	

Age-Off Iterator

Current Time: 1104

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value	
bob	attribute	height	public	1005	5'11"	Entries < 100s old
bob	attribute	height	public	1004	5'5"	
bob	attribute	height	public	1003	5'	Entries > 100s old
bob	attribute	height	public	1002	4'10"	
bob	attribute	height	public	1001	4'9"	
bob	attribute	height	public	1000	4'3"	

Age-Off Accumulo Shell


Table
Name

Iterator
Scope

Iterator
Priority

Iterator Name
(arbitrary)

Age-Off
Filter



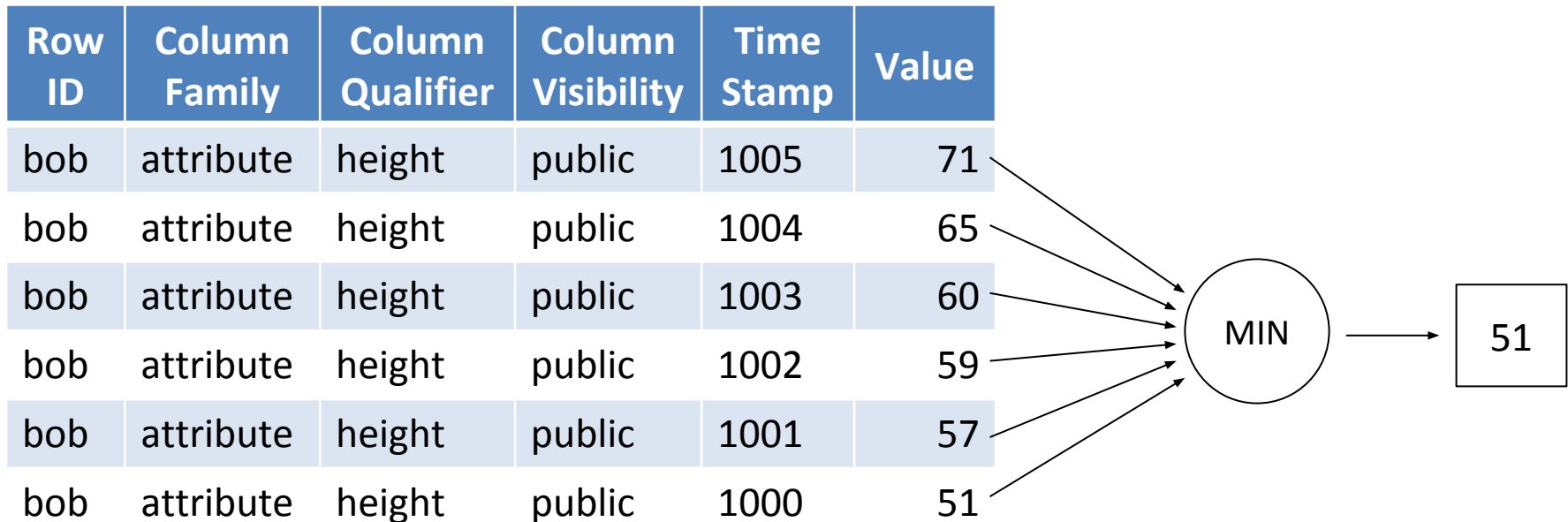
```
> setiter -t mytable -scan -p 10 -n AGE_OFF -ageoff
AgeOffFilter removes entries with timestamps more than <ttl>
milliseconds old
-----> set AgeOffFilter parameter negate, default false keeps
k/v that pass accept method, true rejects k/v that pass accept
method: [ENTER]
-----> set AgeOffFilter parameter ttl, time to live
(milliseconds): 5000
-----> set AgeOffFilter parameter currentTime, if set, use the
given value as the absolute time in milliseconds as the current
time of day: [ENTER]
```

Included Filter Iterators

Name	Description
AgeOffFilter	Age off key/value pairs based on the key's timestamp
ColumnAgeOffFilter	Age off key/value pairs based on the column's timestamp
Greplterator	Perform an exact string match on the key and value
LargeRowFilter	Omit rows that exceed a specified number of columns
RegExFilter	Match entries based on Java regular expressions
ReqVisFilter	Match entries with a non-empty ColumnVisibility
RowFilter	Return rows that meet a given criteria (optimized)
TimestampFilter	Match entries whose timestamp occur within a range
Versioninglterator	Keep a fixed number of versions for each key
VisibilityFilter	Filter entries by a set of authorizations or filter invalid labels from corrupt files
WholeRowlterator	Return a whole row (includes all versions) that meet a given criteria

Combiner Iterators

Apply a function to all available versions of a particular key



Included Combiner Iterators

Name	Description
MaxCombiner	Return the largest value for a row
MinCombiner	Return the smallest value for a row
SummingArrayCombiner	Return an array of element-wise sums from values that contain arrays of Longs
SummingCombiner	Return the sum of the values for a row

Included Transformation Iterators

Name	Description
IndexedDocIterator	Extends the IntersectingIterator by supporting document-partitioned indexing
IntersectingIterator	Group a set of documents and index the documents into a single row
RowDeletingIterator	Delete whole rows
TransformingIterator	Change portions of the key (except Row ID)

Iterator Access

- Scan Time
 - Iterator has access to all of the keys in a row that the user has authorizations to see
- Minor Compaction
 - Iterator has access to all of the keys that are in the MemTable
 - Iterator does not have access to keys stored in RFiles
- Major Compaction
 - Iterator only has access to keys that are in RFiles being compacted

Exercise 4

Iterators from the Shell

Exercise 4

- Objective
 - Create and configure iterators from the Accumulo shell
- Tasks
 - Configure and test the versioning iterator
 - Implement, configure and test the age-off iterator

Table Design

Accumulo Tables

- Accumulo's flexible data model offers table designers complete control over how data is stored
- Specific table designs enable Accumulo to be utilized for many varying applications

Indexing Types

- Out of the box, Accumulo only indexes Row IDs
- On insert, Accumulo sorts Row IDs Lexicographically and maintains an index of the location of Row IDs
- Utilizing Accumulo's flexible structure and creative table design, indexes for existing data can be created and stored in an Accumulo table
- Data that can be indexed:
 - Basic
 - Geo
 - Numerical

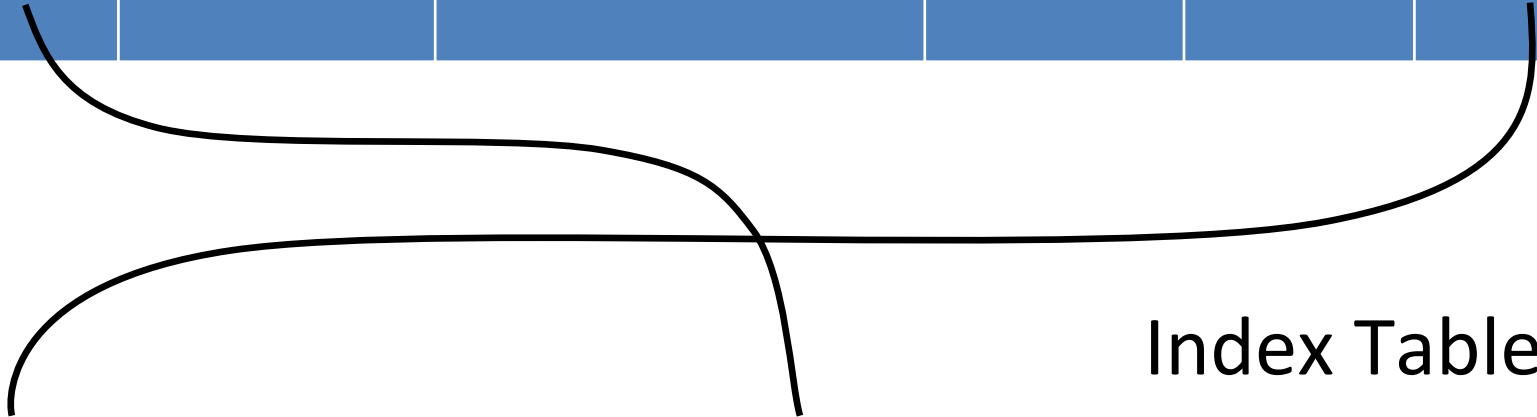
Basic Index for Structured Data

Event Table

Row ID	Column Fam	Column Qual	Visibility	Time	Value
--------	------------	-------------	------------	------	-------

Index Table

Value	Column Fam	Column Qual:Row ID	Visibility	Time	-
-------	------------	--------------------	------------	------	---



Basic Index for Structured Data

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
bob	attribute	dateOfBirth	public	Jul 2013	19850509
bob	attribute	height	public	Jun 2012	5'11"
bob	insurance	dental	private	Sep 2009	MetLife
jane	attribute	bloodType	public	Jul 2011	ab-
jane	attribute	surname	public	Aug 2013	doe
jane	contact	cellPhone	public	Dec 2010	(808) 345-9876
jane	insurance	vision	private	Jan 2008	VSP
john	allergy	major	private	Feb 1988	amoxicillin
john	attribute	weight	public	Sep 2013	180
john	contact	homeAddr	public	Mar 2003	34 Baker LN

Basic Index for Structured Data

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
(808) 345-9876	contact	cellPhone:jane	public	Dec 2010	-
180	attribute	weight:john	public	Sep 2013	-
19850509	attribute	dateOfBirth:bob	public	Jul 2013	-
34 Baker LN	contact	homeAddr:john	public	Mar 2003	-
5'11"	attribute	height:bob	public	Jun 2012	-
MetLife	insurance	dental:bob	private	Sep 2009	-
VSP	insurance	vision:jane	private	Jan 2008	-
ab-	attribute	bloodType:jane	public	Jul 2011	-
amoxicillin	allergy	major:john	private	Feb 1988	-
doe	attribute	surname:jane	public	Aug 2013	-

Basic Index for Unstructured Data

Event Table

Row ID	Column Fam	Column Qual	Visibility	Time	words to index
--------	------------	-------------	------------	------	----------------

Index Table

index	Column Fam	Column Qual:Row ID	Visibility	Time	-
to	Column Fam	Column Qual:Row ID	Visibility	Time	-
words	Column Fam	Column Qual:Row ID	Visibility	Time	-

Range Queries

- Indexing and sorting enable range queries and implicitly provide wildcard query ability
- Data is physically stored sorted by Row ID therefore performing a range query on Row ID is efficient
- A range query on Row ID amounts to sequential reads instead of random I/O, which is what a hash based system would return

Range Queries

Search for “cater” (single term range)

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
cat	text	docId:0104726	-	-	-
catalog	text	docId:1976227	-	-	-
catcher	text	docId:9802347	-	-	-
cater	text	docId:2439023	-	-	-
catfish	text	docId:7622344	-	-	-
cathedral	text	docId:0973852	-	-	-
cats	text	docId:1088231	-	-	-
catsup	text	docId:9824602	-	-	-
cattle	text	docId:2097262	-	-	-
catwalk	text	docId:2903262	-	-	-

Range Queries

Prefix Search for “catc” to “cath” (multi-term range)

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
cat	text	docId:0104726	-	-	-
catalog	text	docId:1976227	-	-	-
catcher	text	docId:9802347	-	-	-
cater	text	docId:2439023	-	-	-
catfish	text	docId:7622344	-	-	-
cathedral	text	docId:0973852	-	-	-
cats	text	docId:1088231	-	-	-
catsup	text	docId:9824602	-	-	-
cattle	text	docId:2097262	-	-	-
catwalk	text	docId:2903262	-	-	-

Range Queries

Search for “cat*”

Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	Value
cat	text	docId:0104726	-	-	-
catalog	text	docId:1976227	-	-	-
catcher	text	docId:9802347	-	-	-
cater	text	docId:2439023	-	-	-
catfish	text	docId:7622344	-	-	-
cathedral	text	docId:0973852	-	-	-
cats	text	docId:1088231	-	-	-
catsup	text	docId:9824602	-	-	-
cattle	text	docId:2097262	-	-	-
catwalk	text	docId:2903262	-	-	-

Wildcards

- Lexicographical order allows trailing wildcard capability: cat*
- Storing terms in reverse order (cat -> tac) enable leading wildcards: *at
- Indexing n-grams (cat, hed, ral) enables both leading and trailing wildcards

Indexing Dates

- Dates/times
 - 04-15-2009 7:45 pm
 - 201104151945000 (Date converted to ms)
 - To index in reverse order (newest dates come first)
the date must be transformed

$$\begin{array}{r} 9999999999999999 \\ -201104151945000 \\ \hline 798895848054999 \end{array}$$

- Before queries are performed, dates need to be transformed

Indexing Integers

- Lexicographical sorting of numbers is not the same as numeric sorting
 - Numeric Sort: 1,2,3,10,50,75,100,200,1000
 - Lexicographical Sort: 1,10,100,1000,2,200,3,50,75
- To support numeric range scans, transforming or encoding of integers is required
- Options:
 - Zero pad
 - Use byte values

Indexing Integers

- Zero padding to the longest length int or long in your index
 - Max signed Integer: 2147483647 (10 digits)
 - Max signed long: 9223372036854775808 (19 digits)
 - Account for the sign by prepending a “0” for negative numbers and “1” for positive numbers
 - Transform negative numbers by adding 99999...9 to value

Integer	Transformed for Indexing
-1024	0999999999999999998975
-5	0999999999999999999994
64	1000000000000000000064
2096	10000000000000000002096

Indexing Integers

- Using the binary value and flipping the sign bit allows negatives to sort correctly

– Example transformation

Number	2's Compliment	Invert 1 st bit	Hex Value
1234	0000 0100 1101 0010	1000 0100 1101 0010	0x84 0xD2
-1234	1111 1011 0010 1110	0111 1011 0010 1110	0x7B 0x2E

– Sample data stored in sorted order

Integer	Transformed for indexing
-1234	7B2E
5	8005
1234	84D2

Indexing Floats and Doubles

- Floats and doubles have the same lexicographic sorting problems as integers, but the solution is more complicated
- Zero padding would need to be done on both sides of the decimal
- This is problematic and wasteful
- The alternate solution is based on transforming the binary representation of floats and doubles

Indexing Floats and Doubles

[sign bit][exponent bits][mantissa bits]

- For negative numbers: Invert all bits
 - XOR'ing each byte with 0xFF will do this
- For positive numbers: invert just the sign bit
 - XOR'ing left most byte with 0x80 will do this

Indexing IP addresses

- ID addresses need to be transformed to support proper IP range queries
- IPv4 addresses are 4 byte unsigned integers and they can be indexed just like integers (no sign bit necessary)
- Another option is zero padding the individual octets for human readable IPs

IP Address	Transformed for Indexing
1.1.1.1	001.001.001.001
192.168.0.2	192.168.000.002

Indexing Domain Names and URLs

- To exploit the locality within domain names and URLs it is recommended to transform the entities when indexing
- This allows sub-domains of the same site sort together

Domain Name / URL	Transformed for Indexing
google.com	com.google
mail.google.com	com.google.mail
yahoo.com/path/file.html	com.yahoo/path/file.html

Twitter Data Table + Tweet Index

Row ID	Column Family	Column Qualifier	Visibility	Timestamp	Value
{tweetID}	text	-	-	-	{tweet text}



Row ID	Column Family	Column Qualifier	Visibility	Timestamp	Value
{token}	{tweetID}	-	-	-	-

Accumulo Input Format

- Accumulo can be set as the input for a MapReduce job by configuring the AccumuloInputFormat

```
AccumuloInputFormat.setInputInfo(  
    job.getConfiguration(), // The job configuration  
    userName,               // The user name  
    userPass.getBytes(),    // The user password  
    table,                  // The table to read from  
    new Authorizations());  // The users visibility
```

```
AccumuloInputFormat.setZooKeeperInstance(  
    job.getConfiguration(), // The job configuration  
    instanceName,           // The name of the Accumulo instance  
    zookeepers);            // A list of ZooKeeper servers
```

Accumulo Output Format

- Accumulo can be set as the output of a MapReduce job by configuring the AccumuloOutputFormat

```
AccumuloOutputFormat.setInputInfo(  
    job.getConfiguration(), // The job configuration  
    userName,               // The user name  
    userPass.getBytes(),    // The user password  
    createTables,           // Create the table if it doesn't exist  
    table);                 // The table to write to
```

```
AccumuloOutputFormat.setZooKeeperInstance(  
    job.getConfiguration(), // The job configuration  
    instanceName,           // The name of the Accumulo instance  
    zookeepers);            // A list of ZooKeeper servers
```

Reading Data from Accumulo

- A Range can be used to specify a range of records to read

```
LinkedList<Range> searchRange = new LinkedList<Range>();  
  
// Create a list of terms to search for  
for(String searchTerm : terms) {  
    searchRange.add(new Range(searchTerm));  
}
```

Reading Data from Accumulo

- A BatchScanner can be used to efficiently retrieve several ranges simultaneously

```
BatchScanner batchScan = conn.createBatchScanner(  
    "table",           // Name of the table to write to  
    new Authorizations(), // The users visibility  
    10);               // Number of concurrent threads  
  
//Set the search ranges for the BatchScanner  
batchScan.setRanges(searchRanges);  
  
// Getting the key and value for each row  
for(Entry<Key,Value> entry : batchScanner) {  
    String row = e.getKey().getRow();  
    Value value = e.getValue();  
}
```

D4M :
Dynamic Distributed Dimension Data
Model

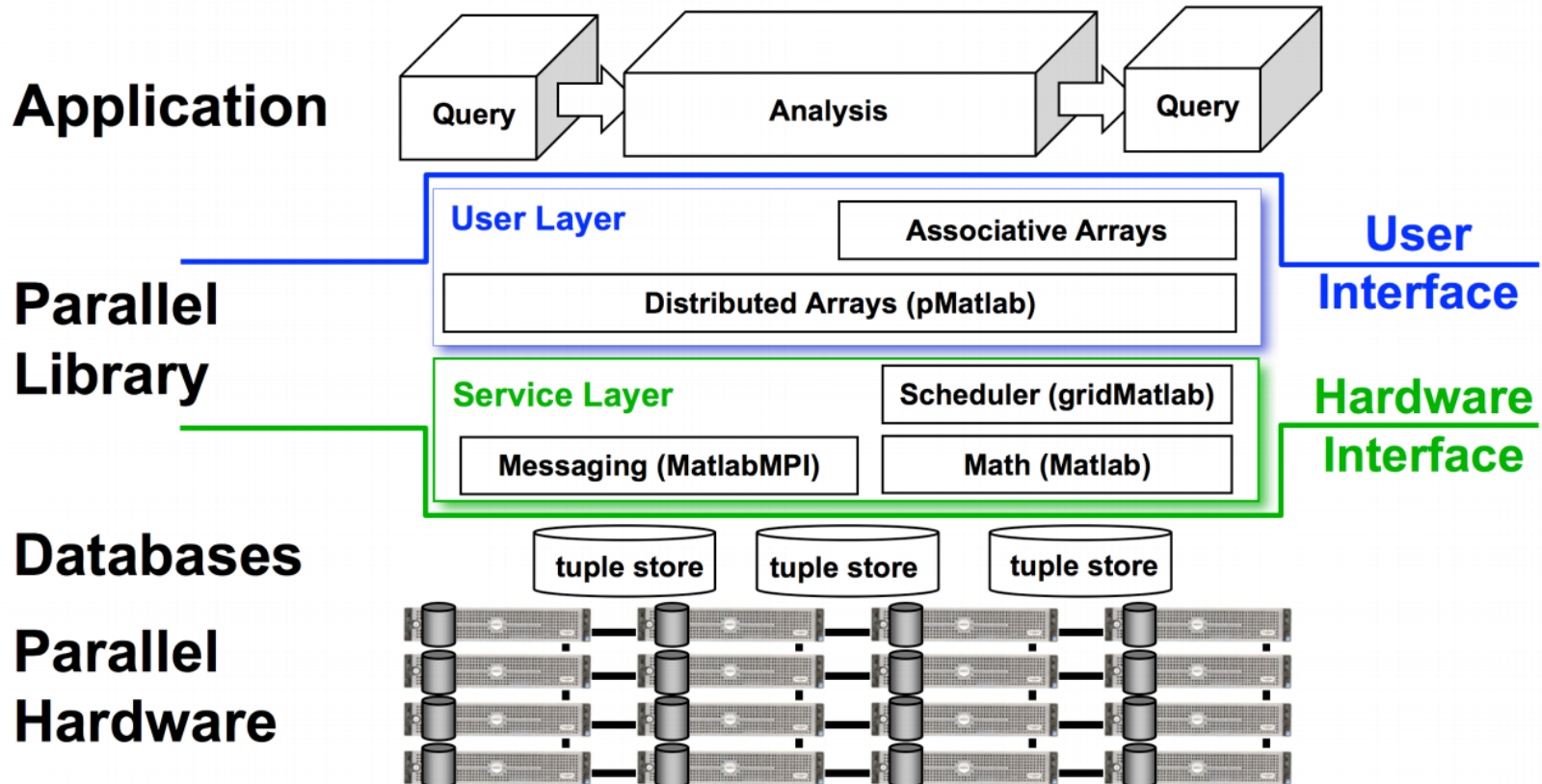
D4M : Intro

D4M was developed to provide a mathematical rich interface to tuple stores and traditional databases to solve the challenges associated with big data and computation complexity.

D4M allows linear algebra to be readily applied to big data.

D4M

D4M: Dynamic Distributed Dimensional Data Model



D4M: Connecting to Accumulo

- Connecting to Accumulo

```
DB = DBserver(hostname,cb_type,instance_name);
```

- Creating or Connecting to a Table

```
T = DB(table1);
```

- Creating or Connecting to a DBtablePair

```
TT = DB(table1,table2);
```

D4M: Ingesting Data into Accumulo

Ingesting data into a DBtable or a DBtablePair

- To insert data into a table from an Assoc Array

```
Table = Put(Table, num2str(MyData));
```

- num2str function necessary because Accumulo stores Strings
- To clear tables for fresh ingest, delete them and recreate them

```
Table = deleteForce(Table);  
Table = DB('tableName');
```

D4M: Querying Data on Accumulo

Querying for data from a DBtable or a DBtablePair will return information in an Assoc Object. The first argument queries Rows, the second argument queries Columns.

- To query for Everything in a table use

```
RESULT = T(:, :)
```

- To query for a range in the rows use

```
RESULT = T('cat:mouse,', :)
```

D4M: Querying Data on Accumulo

- To query for a specific range, use a comma delimited list, ending with a comma. In this example we are scanning the columns.

```
RESULT = T(:, 'cat,dog,mouse,')  
// FASTER with a DBtablePair  
RESULT = TT(:, 'cat,dog,mouse,')
```

- When querying columns, using a DBtablePair will return results faster.

PyAccumulo

- Takes advantage of the Thrift Proxy in Accumulo
- Requires Accumulo $\geq 1.5.0$
- Allows Developers to write python clients and applications for Accumulo

PyAccumulo : Installing

Requires python-pip package

```
$ sudo pip install thrift  
$ git clone git@github.com:accumulo/pyaccumulo.git  
$ cd pyaccumulo  
$ sudo python ./setup.py
```


PyAccumulo : Configure and Start Accumulo Proxy

Configure the proxy.properties file

```
# $ACCUMULO_HOME/proxy/proxy.properties
org.apache.accumulo.proxy.ProxyServer.useMockInstance=false
org.apache.accumulo.proxy.ProxyServer.useMiniAccumulo=false
org.apache.accumulo.proxy.ProxyServer.protocolFactory=
    org.apache.thrift.protocol.TCompactProtocol$Factory
org.apache.accumulo.proxy.ProxyServer.port=50096
org.apache.accumulo.proxy.ProxyServer.instanceName=training
org.apache.accumulo.proxy.ProxyServer.zookeepers=localhost:2181
```

Start the Proxy Server

```
$ $ACCUMULO_HOME/bin/accumulo proxy -p \
    $ACCUMULO_HOME/proxy/proxy.properties
```

PyAccumulo : Connecting to Accumulo via Accumulo Proxy

Configure the proxy.properties file

```
#!/bin/env python

from pyaccumulo import Accumulo, Mutation, Range
conn = Accumulo(host="localhost", port=50096, user="root",
    password="secret")
```

PyAccumulo : Table Operations

Configure the proxy.properties file

```
. . .  
  
table = "pyTable"  
if not conn.table_exists(table):  
    conn.create_table(table)
```

PyAccumulo : Writing

Simple Writes

```
. . .  
for num in range(0, 1000):  
    m = Mutation("row_%d"%num)  
    m.put(cf="cf1", cq="cq1", val="%d"%num)  
    m.put(cf="cf2", cq="cq2", val="%d"%num)  
    conn.write(table,m)
```

Batch Writer

```
. . .  
wr = conn.create_batch_writer(table)  
for num in range(0, 1000):  
    m = Mutation("row_%d"%num)  
    m.put(cf="cf1", cq="cq1", val="%d"%num)  
    m.put(cf="cf2", cq="cq2", val="%d"%num)  
    wr.add_mutation(m)  
wr.close()
```

PyAccumulo : Scanning

Scanner

```
. . .  
for entry in conn.scan(table, scanrange=Range(srow='row_1',  
erow='row_2')):  
    print entry.row, entry.cf, entry.cv, entry.ts, entry.val
```

Batch Scanner

```
. . .  
for entry in conn.batch_scan(table, numthreads=10):  
    print entry.row, entry.cf, entry.cv, entry.ts, entry.val
```

awells@clearedgeit.com

Credits

D4M examples come from the D4M Documentation

PyAccumulo Examples come from the PyAccumulo
Documentation