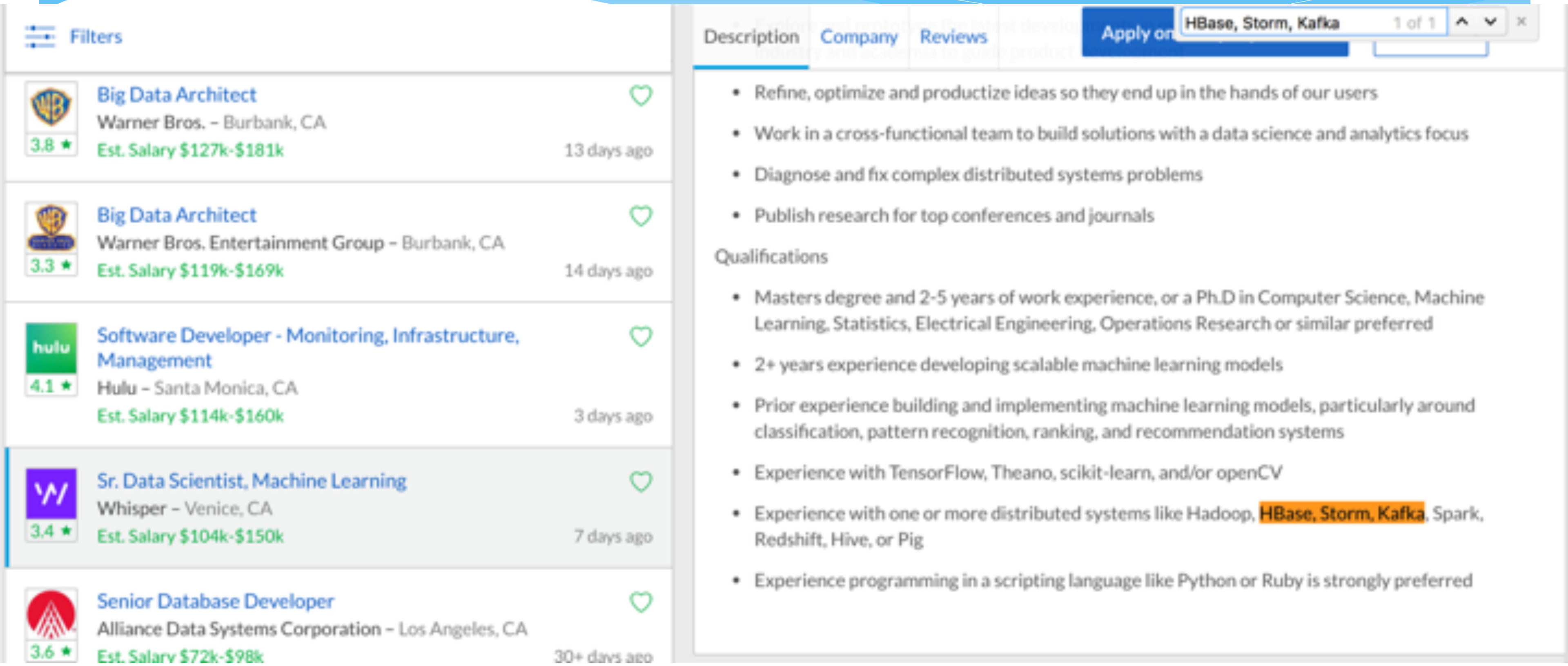
Stat 405 - Integration With Distributed Resources

Christian Gao

Kafka

- Originally developed by LinkedIn
- Opensourced in early 2011
- Written in Scala
- 9 core committers, + 20 contributors

Incentives



Big Data Architect
Warner Bros. – Burbank, CA
3.8 ★
Est. Salary $127k-$181k
13 days ago

Big Data Architect
Warner Bros. Entertainment Group – Burbank, CA
3.3 ★
Est. Salary $119k-$169k
14 days ago

Software Developer - Monitoring, Infrastructure, Management
Hulu – Santa Monica, CA
4.1 ★
Est. Salary $114k-$160k
3 days ago

Sr. Data Scientist, Machine Learning
Whisper – Venice, CA
3.4 ★
Est. Salary $104k-$150k
7 days ago

Senior Database Developer
Alliance Data Systems Corporation – Los Angeles, CA
3.6 ★
Est. Salary $72k-$98k
30+ days ago

- Refine, optimize and productize ideas so they end up in the hands of our users
- Work in a cross-functional team to build solutions with a data science and analytics focus
- Diagnose and fix complex distributed systems problems
- Publish research for top conferences and journals

Qualifications

- Masters degree and 2-5 years of work experience, or a Ph.D in Computer Science, Machine Learning, Statistics, Electrical Engineering, Operations Research or similar preferred
- 2+ years experience developing scalable machine learning models
- Prior experience building and implementing machine learning models, particularly around classification, pattern recognition, ranking, and recommendation systems
- Experience with TensorFlow, Theano, scikit-learn, and/or openCV
- Experience with one or more distributed systems like Hadoop, HBase, Storm, Kafka, Spark, Redshift, Hive, or Pig
- Experience programming in a scripting language like Python or Ruby is strongly preferred

Data Application Lab

Companies that use Kafka

- Linked In
- Yahoo
- Twitter
- Netflix
- Square
- Spotify
- Pinterest
- Uber
- Goldman Sachs
- AirBnb

Uses of Kafka

- Messaging
- Website Activity Tracking
- Metrics – operational monitoring data
- Log Aggregation- Debugging
- Stream Processing
- Event Sourcing
- Commit Log

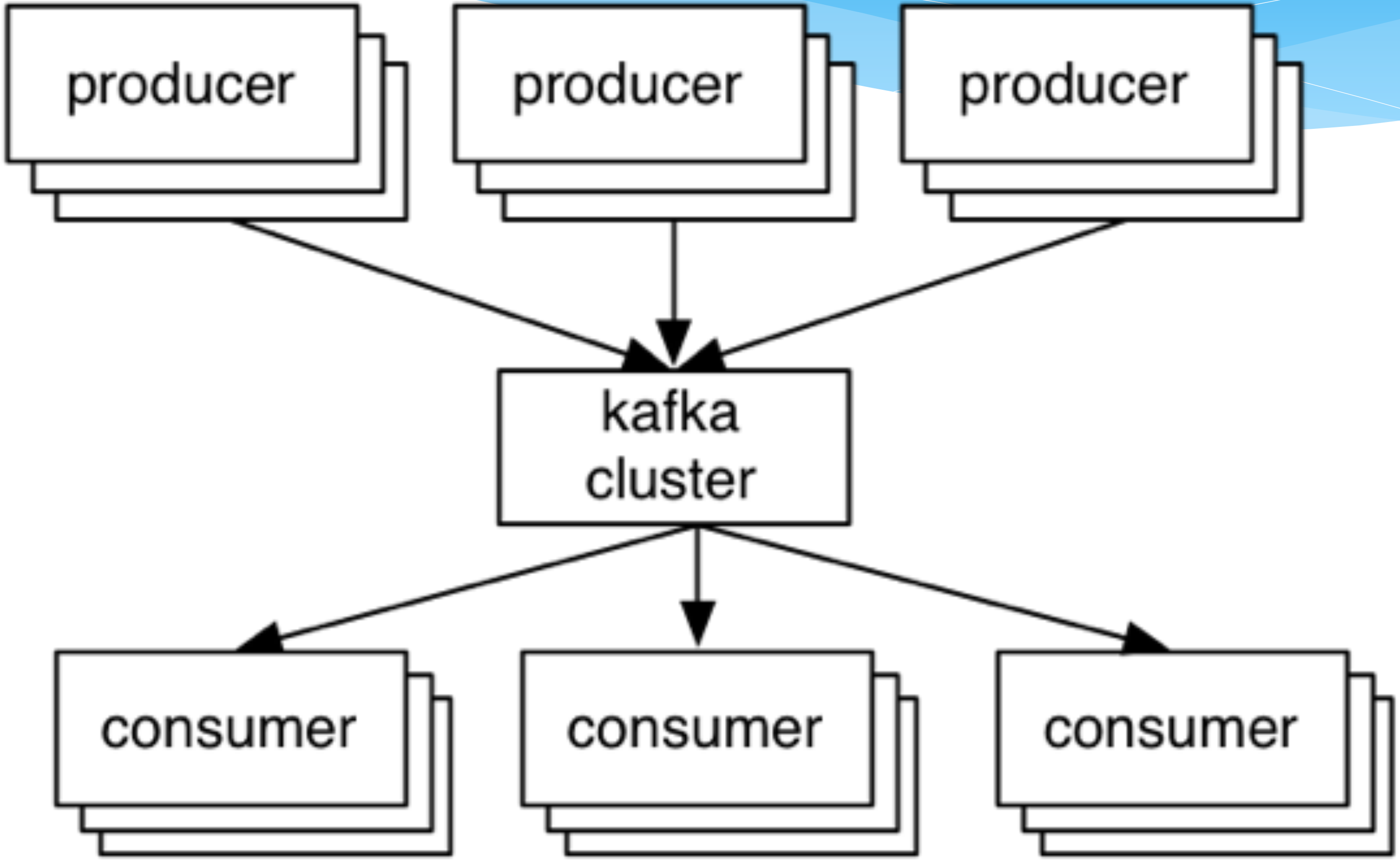
- Kafka with Elastic Search and Kibana
- Demo Monitoring System:
- http://demo.elastic.co/

Data Application Lab

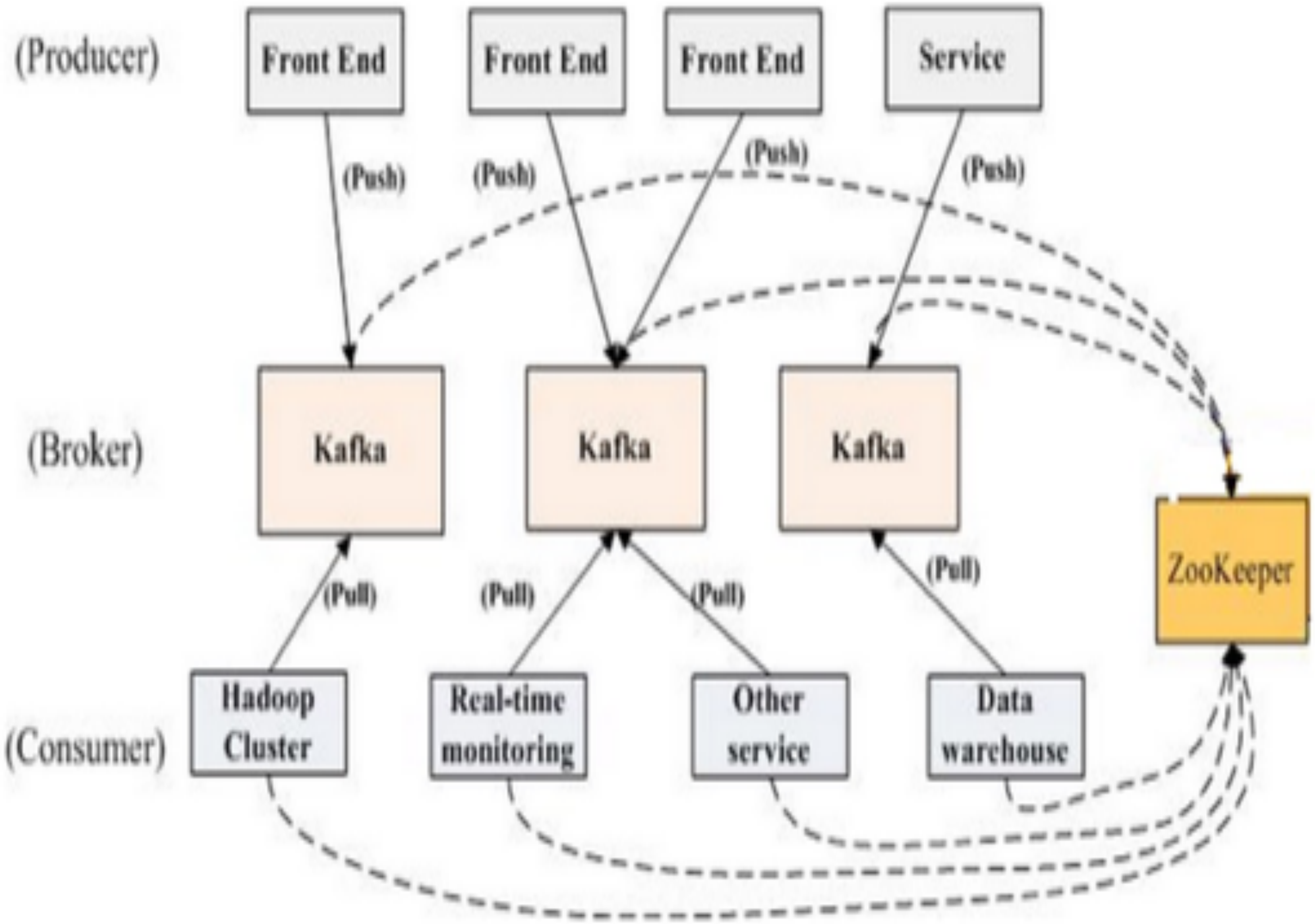
Kafka Features

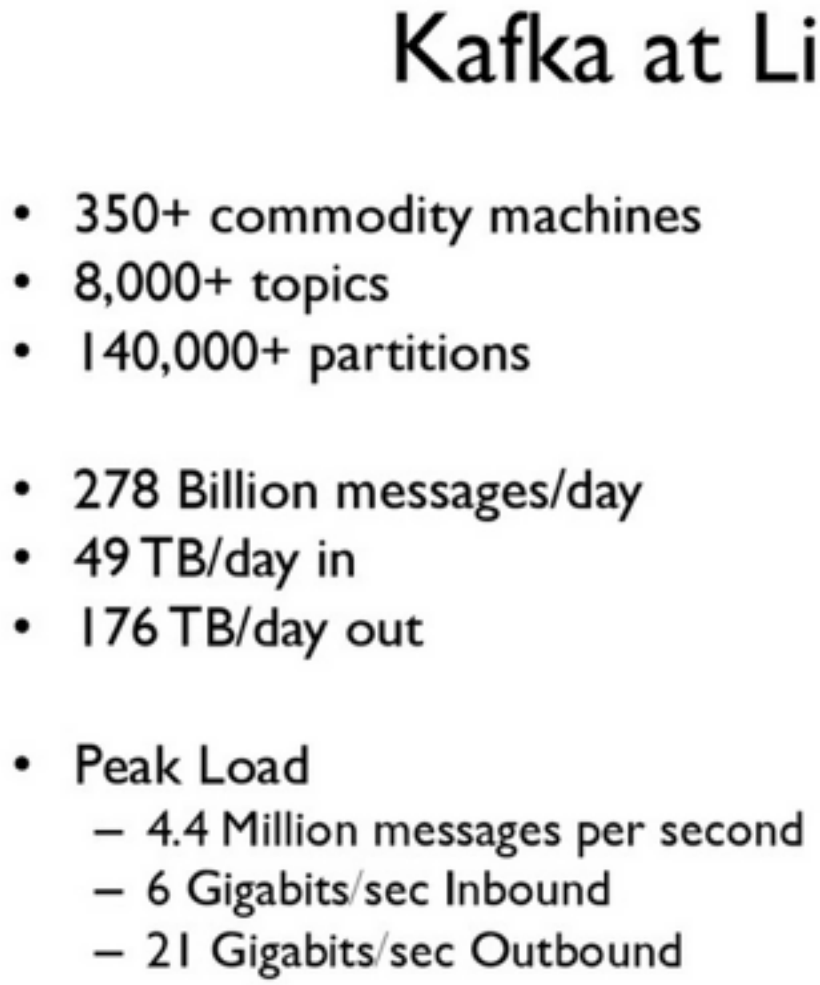
- Distributed messaging system
- Provide a unified, high-throughput, low-latency platform for handling real-time data feeds
- Up to 2M writes/reads per second on 3 commodity machine cluster

Abstract



Data Application Lab


(Producer)
Front End
Front End
Front End
Service
(Push)
(Push)
(Push)
(Push)
(Broker)
Kafka
Kafka
Kafka
ZooKeeper
(Pull)
(Pull)
(Pull)
(Pull)
(Consumer)
Hadoop Cluster
Real-time monitoring
Other service
Data warehouse



Kafka at LinkedIn

- 350+ commodity machines
- 8,000+ topics
- 140,000+ partitions

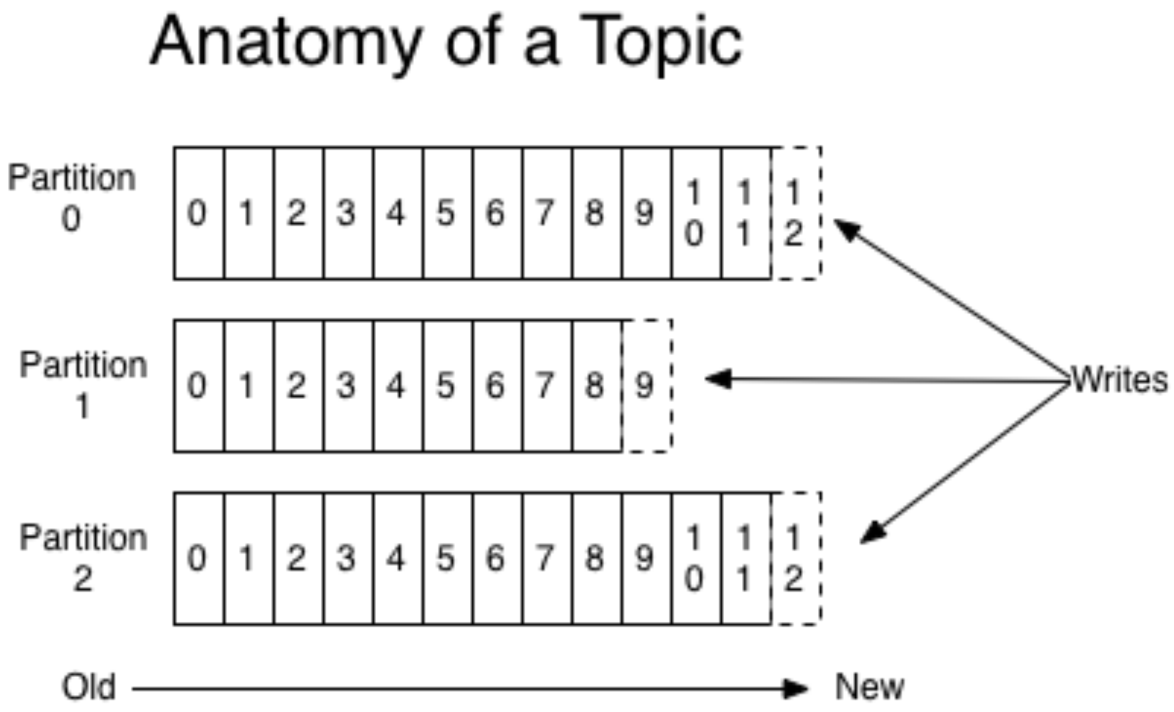
- 278 Billion messages/day
- 49 TB/day in
- 176 TB/day out

- Peak Load
  - 4.4 Million messages per second
  - 6 Gigabits/sec Inbound
  - 21 Gigabits/sec Outbound

Kafka Terms

- Maintains feeds of messages in queues called **topics**
- Call processes that publish messages to a Kafka topic are **producers**
- Call processes that subscribe to topics and process the feed of published messages are **consumers**
- Run as a cluster comprised of one or more servers each of which is called a **broker**

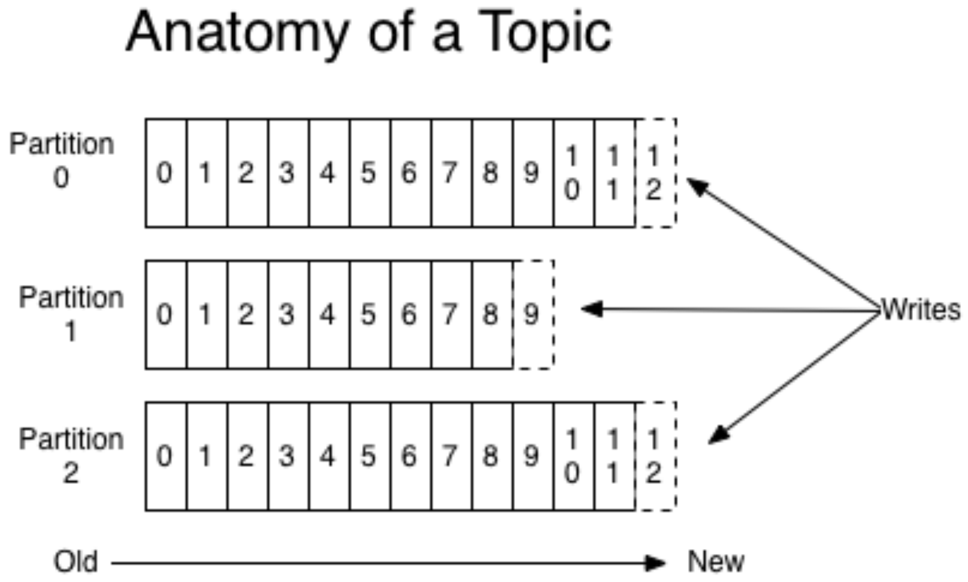
Topic



Data Application Lab

Topic

- Has a partitioned log structure.
- Takes in a key and a record as an input.
- Keys are hashed then sent to a topic - same key always to same partition.

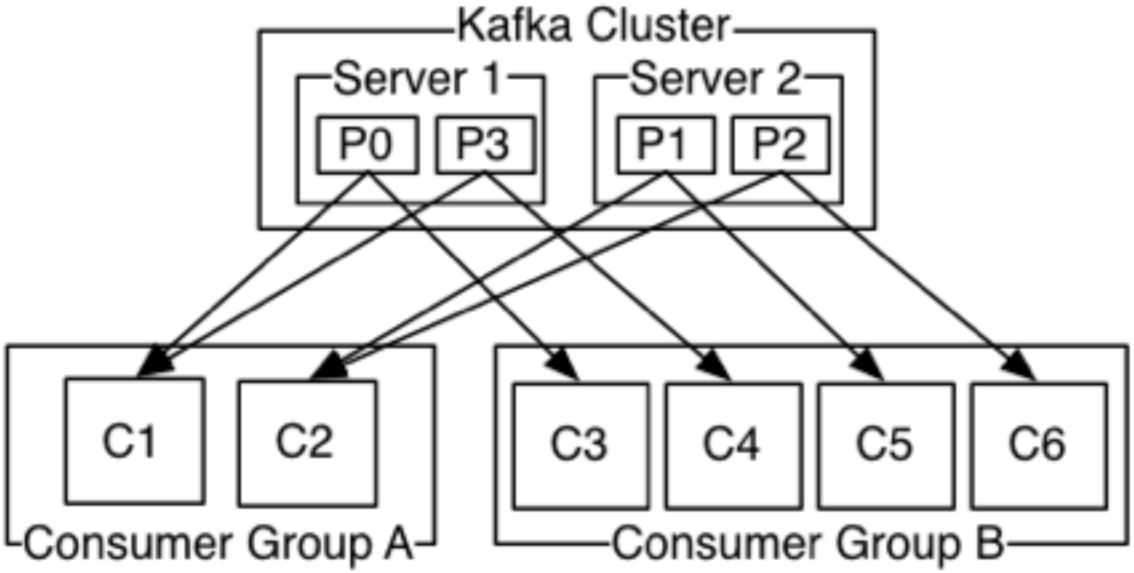


Kafka Partition & Brokers

- Each partition is replicated according to settings.
- Each partition has one broker which acts as the "leader"
- The leader handles all read and write requests
- One or more servers which act as "followers“
- If the leader fails, one of the followers will automatically become the new leader

Consumer

- Consumers assign to consumer group name
- If only one group, work like message queue
- If not, work like publish-subscribe



Data Application Lab

Interview Question!

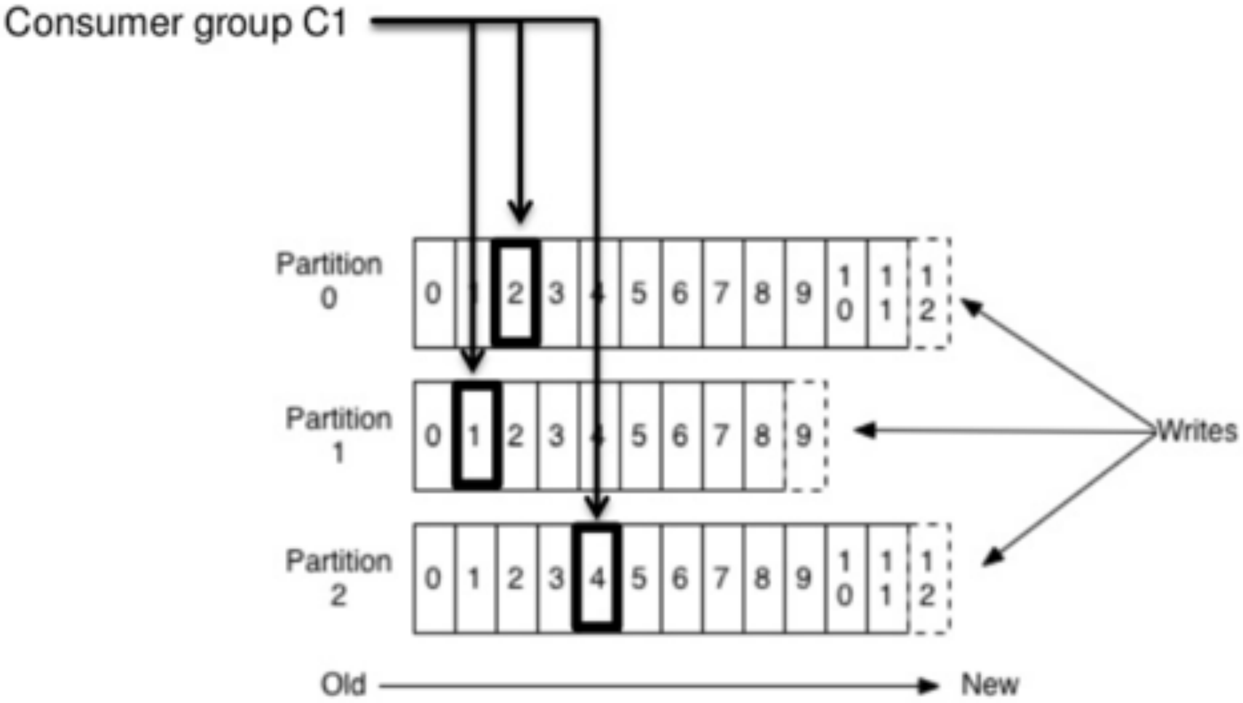
- Suppose you have you have transactional data for users. How would you set up Kafka so that the consumption of transactional data is always in the correct order for each user?

Data Application Lab

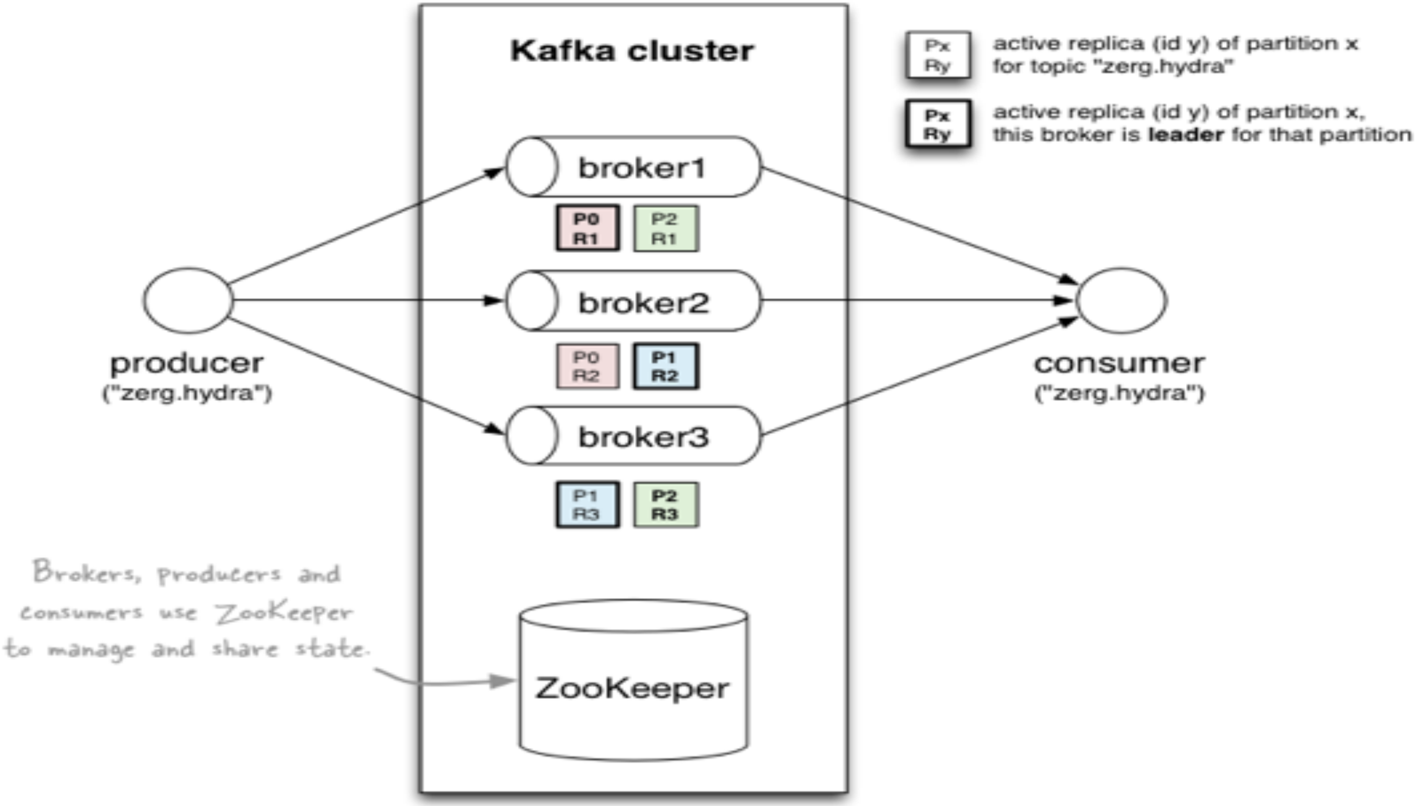
Partition Offsets

- Partition offset keeps track of the consumer’s progress in consuming the data.
- Each new message’s key goes through a modulus algorithm and goes to a partition.
- FIFO- First in first out, consumers consume starting at the earliest message then works backwards.

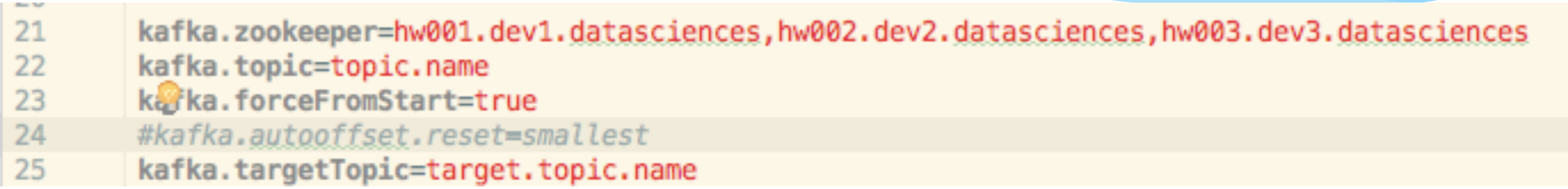
Partition Offsets



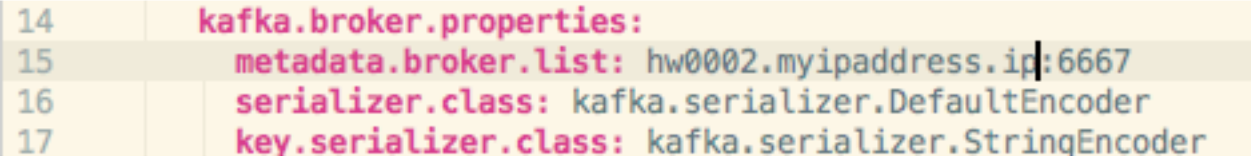
- Interview question- How many servers can go down?


Kafka cluster
Px Ry
active replica (id y) of partition x for topic "zerg.hydra"
Px Ry
active replica (id y) of partition x, this broker is **leader** for that partition
broker1
P0 R1
P2 R1
broker2
P0 R2
P1 R2
broker3
P1 R3
P2 R3
producer ("zerg.hydra")
consumer ("zerg.hydra")
Brokers, producers and consumers use ZooKeeper to manage and share state.
ZooKeeper

Properties and Yaml File



```
kafka.zookeeper=hw001.dev1.datasciences,hw002.dev2.datasciences,hw003.dev3.datasciences
kafka.topic=topic.name
kafka.forceFromStart=true
#kafka.autooffset.reset=smallest
kafka.targetTopic=target.topic.name
```



```
kafka.broker.properties:
  metadata.broker.list: hw0002.myipaddress.ip:6667
  serializer.class: kafka.serializer.DefaultEncoder
  key.serializer.class: kafka.serializer.StringEncoder
```

- State the servers to ask for data, the topic name, and the format of the messages.

Data Application Lab

Kafka Demo

- Kafka Spout:

```
private OpaqueTridentKafkaSpout createKafkaSpout() {
  BrokerHosts zk = new ZkHosts(zkNodeAddress);
  TridentKafkaConfig spoutConf = new TridentKafkaConfig(zk, topicName);
  spoutConf.scheme = new SchemeAsMultiScheme(new StringScheme());
  OpaqueTridentKafkaSpout spout = new OpaqueTridentKafkaSpout(spoutConf);
  return(spout);
}
```

- This method in Kafka Consume Topology takes in credentials and creates a Kafka Spout. The Spout returns a stream.

References

- Kafka Demo Code:
- https://heroku.github.io/kafka-demo/
- Documentation:
- http://kafka.apache.org/documentation.html
- http://www.confluent.io/blog/how-to-choose-the-number-of-topicspartitions-in-a-kafka-cluster/

Data Application Lab

HBase Agenda

- Hbase History
- How HBase works
- Hbase shell
- Distributed NO-SQL architecture
- Design Hbase Table
- Programming with HBase

History

- Google BigTable whitepaper 2006
- Became Hadoop sub-project 2008
- Became Apache top-level project 2010
- Hbase 1.0 released 2015
- Used By Facebook’s Messaging system.

Data Application Lab

Key Features

- Schema-less, no-sql
- Column-oriented
- Good for semi-structure and structured data
- Random, real-time read/write
- Data replica across cluster

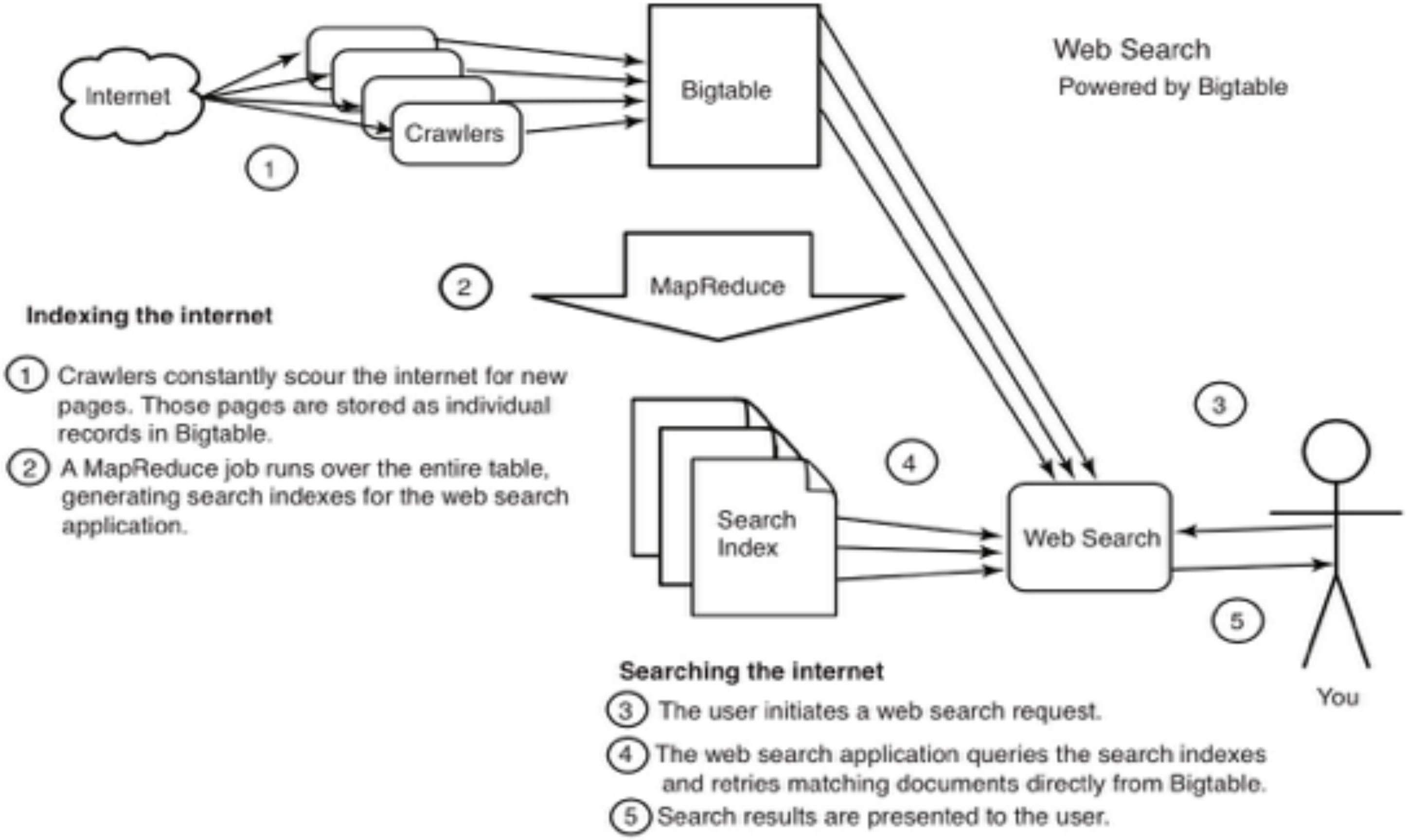
What Hbase is Good For

- Large datasets
- Sparse datasets
- Automatic Backup
- Loosely coupled records- All Rows and be different.
- Lot of concurrent clients- Queries from many users.

What Hbase is Not Good For

- Not a SQL database
- Not relational
- No joins
- Not a engine for sophisticated query
- Not a replacement for legacy RDBMS

HBase in Google



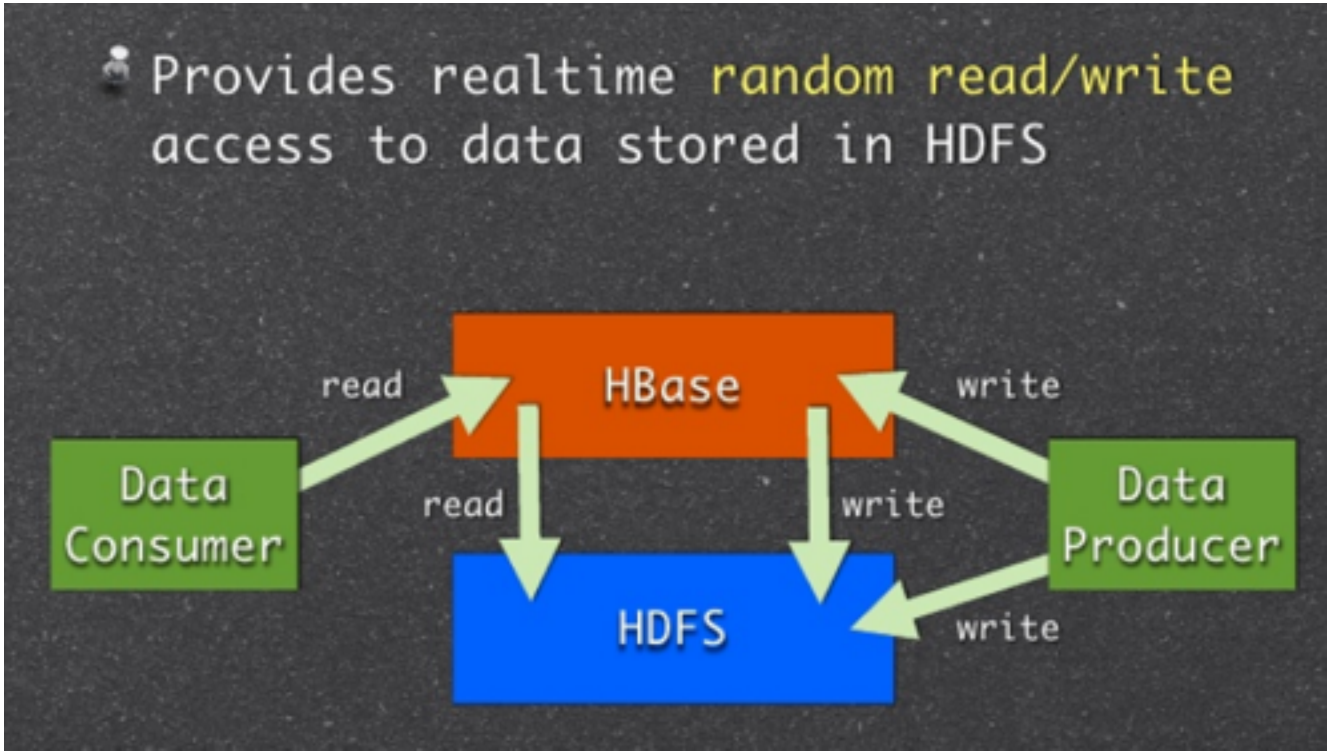
Indexing the internet

1. Crawlers constantly scour the internet for new pages. Those pages are stored as individual records in Bigtable.
2. A MapReduce job runs over the entire table, generating search indexes for the web search application.

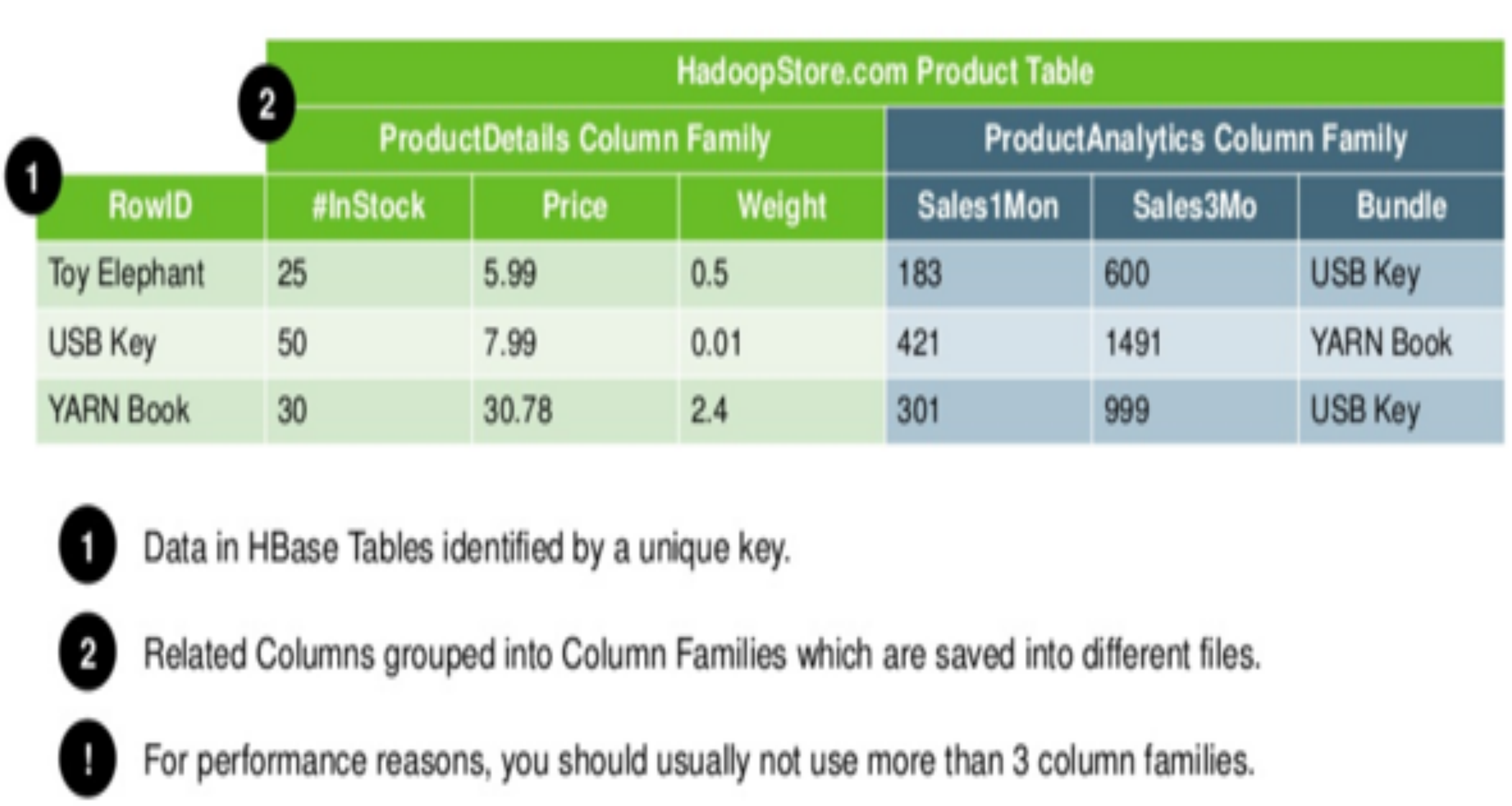
Searching the internet

3. The user initiates a web search request.
4. The web search application queries the search indexes and retries matching documents directly from Bigtable.
5. Search results are presented to the user.

Integrate With Hadoop



- Provides realtime random read/write access to data stored in HDFS



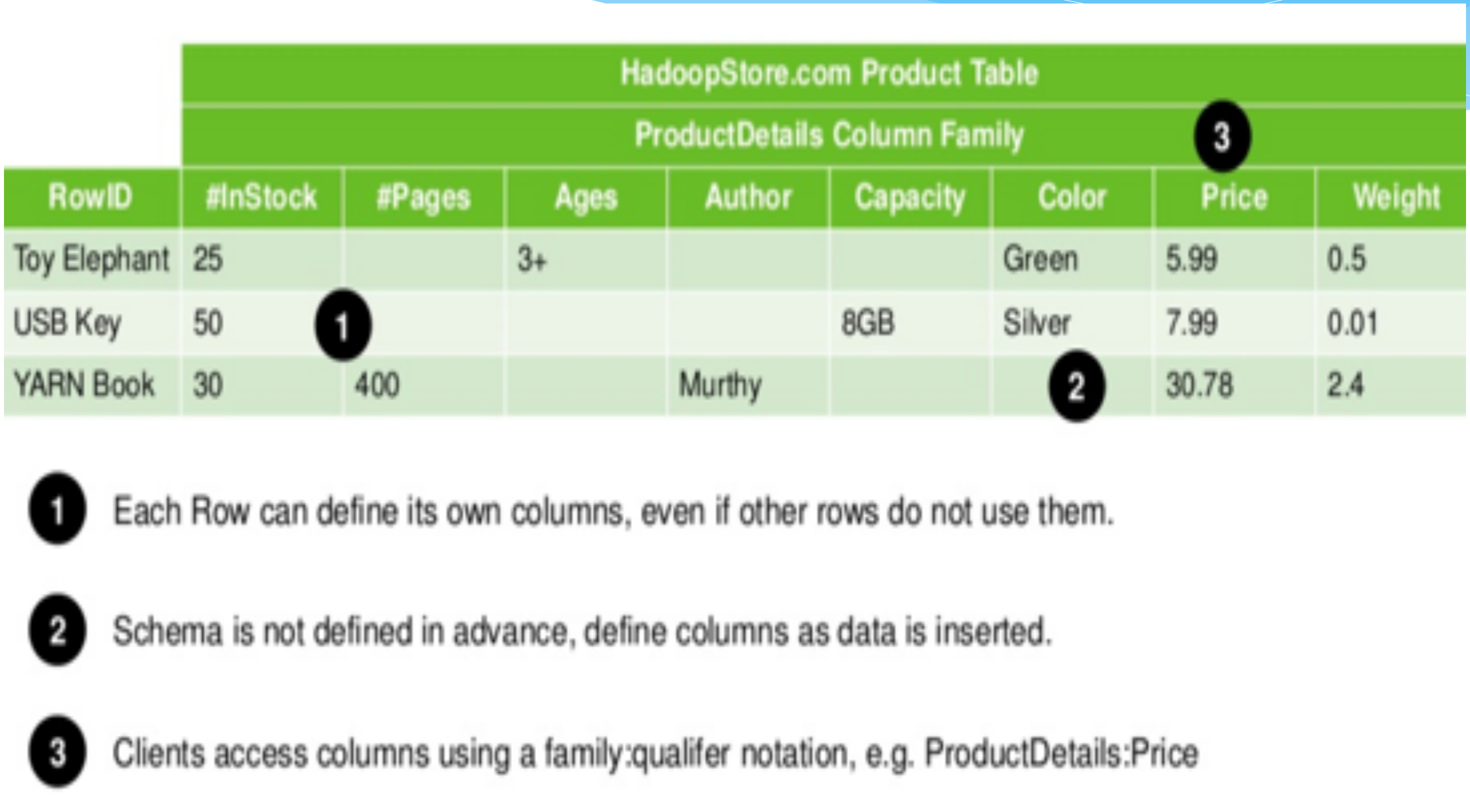
HadoopStore.com Product Table						
	ProductDetails Column Family			ProductAnalytics Column Family		
RowID	#InStock	Price	Weight	Sales1Mon	Sales3Mo	Bundle
Toy Elephant	25	5.99	0.5	183	600	USB Key
USB Key	50	7.99	0.01	421	1491	YARN Book
YARN Book	30	30.78	2.4	301	999	USB Key

1 Data in HBase Tables identified by a unique key.

2 Related Columns grouped into Column Families which are saved into different files.

! For performance reasons, you should usually not use more than 3 column families.

Flexible Schema

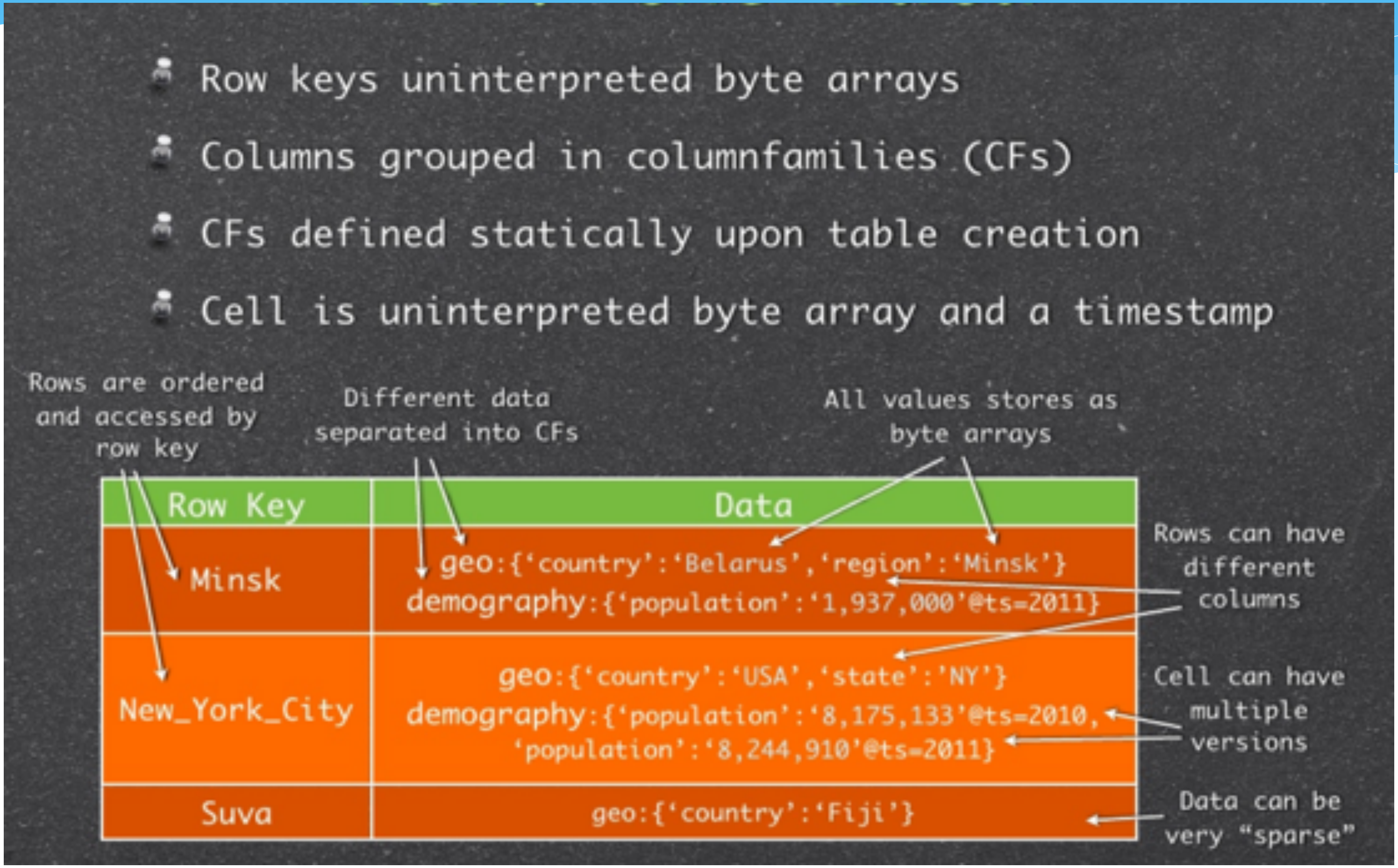


	HadoopStore.com Product Table							
	ProductDetails Column Family						3	
RowID	#InStock	#Pages	Ages	Author	Capacity	Color	Price	Weight
Toy Elephant	25		3+			Green	5.99	0.5
USB Key	50	1			8GB	Silver	7.99	0.01
YARN Book	30	400		Murthy		2	30.78	2.4

1. Each Row can define its own columns, even if other rows do not use them.
2. Schema is not defined in advance, define columns as data is inserted.
3. Clients access columns using a family:qualifer notation, e.g. ProductDetails:Price

Data Application Lab

Sorted Row Key



- Row keys uninterpreted byte arrays
- Columns grouped in columnfamilies (CFs)
- CFs defined statically upon table creation
- Cell is uninterpreted byte array and a timestamp

Row Key	Data

Minsk	geo:{‘country’:‘Belarus’,‘region’:‘Minsk’}
	demography:{‘population’:‘1,937,000’@ts=2011}

New_York_City	geo:{‘country’:‘USA’,‘state’:‘NY’}
	demography:{‘population’:‘8,175,133’@ts=2010,
	‘population’:‘8,244,910’@ts=2011}

Suva	geo:{‘country’:‘Fiji’}

Rows are ordered and accessed by row key

Different data separated into CFs

All values stores as byte arrays

Rows can have different columns

Cell can have multiple versions

Data can be very “sparse”

Backup With Timestamp

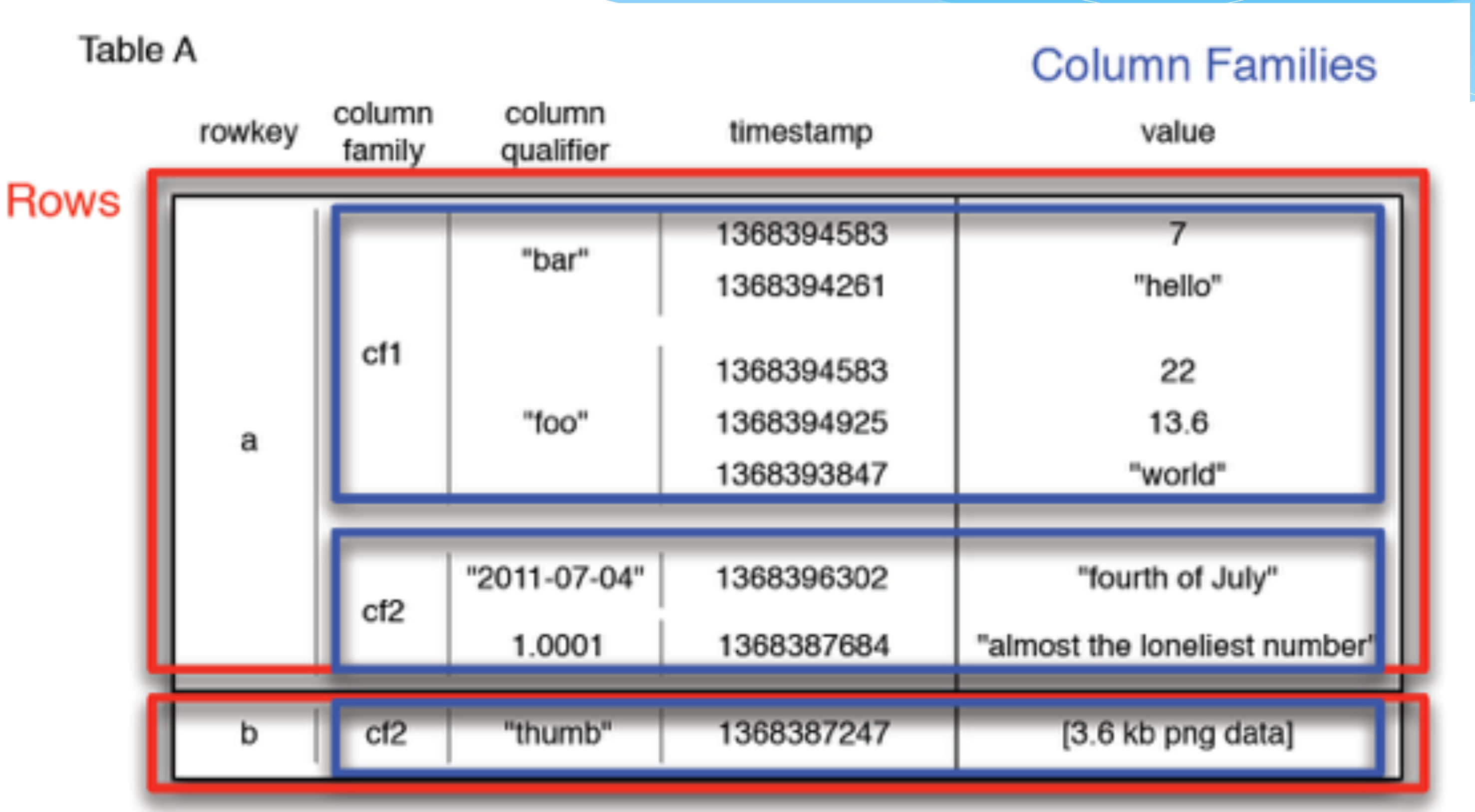


Table A

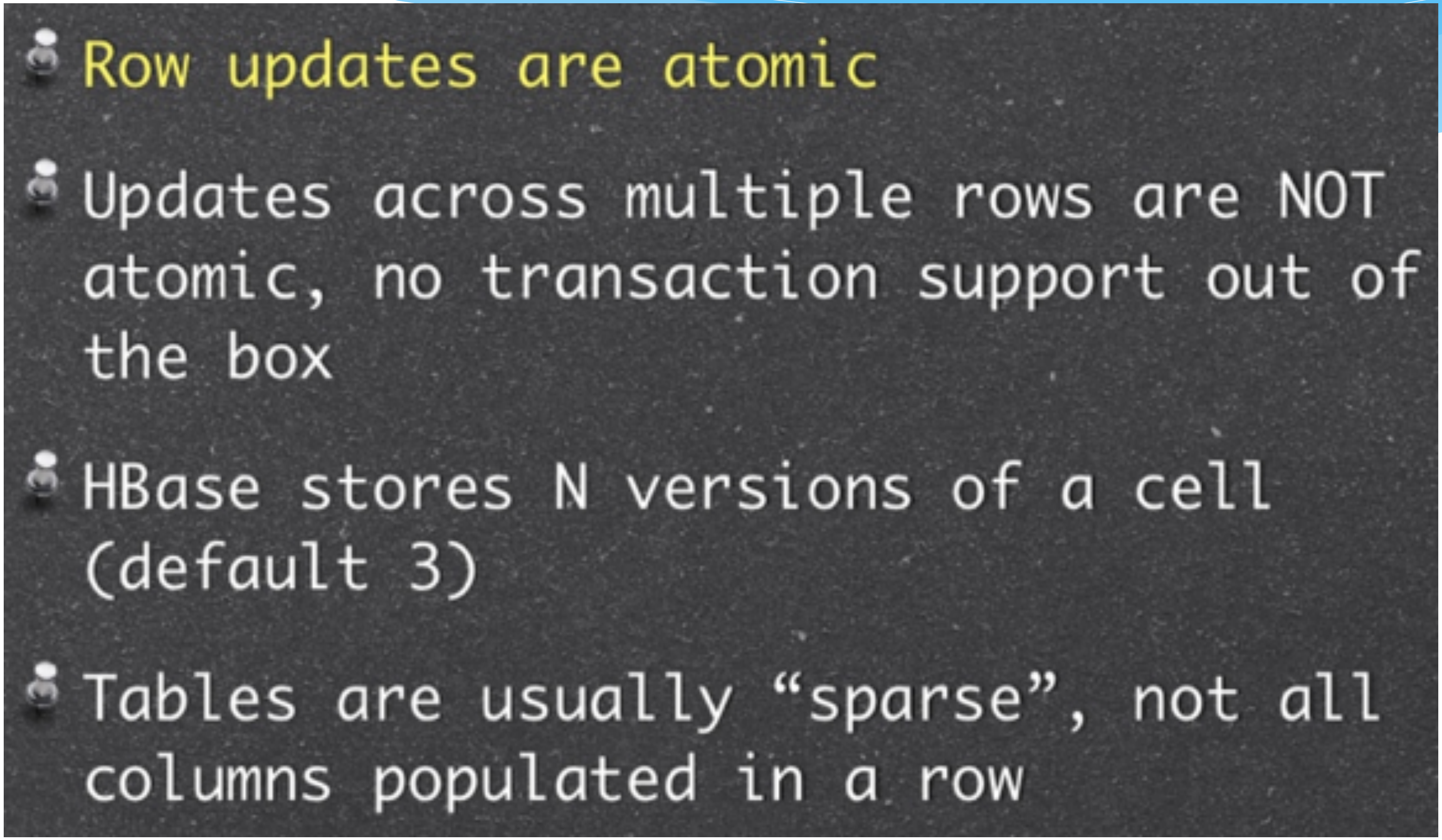
Column Families

Rows

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
			1368393847	"world"
	cf2	"2011-07-04"	1368396302	"fourth of July"
		1.0001	1368387684	"almost the loneliest number"
b	cf2	"thumb"	1368387247	[3.6 kb png data]

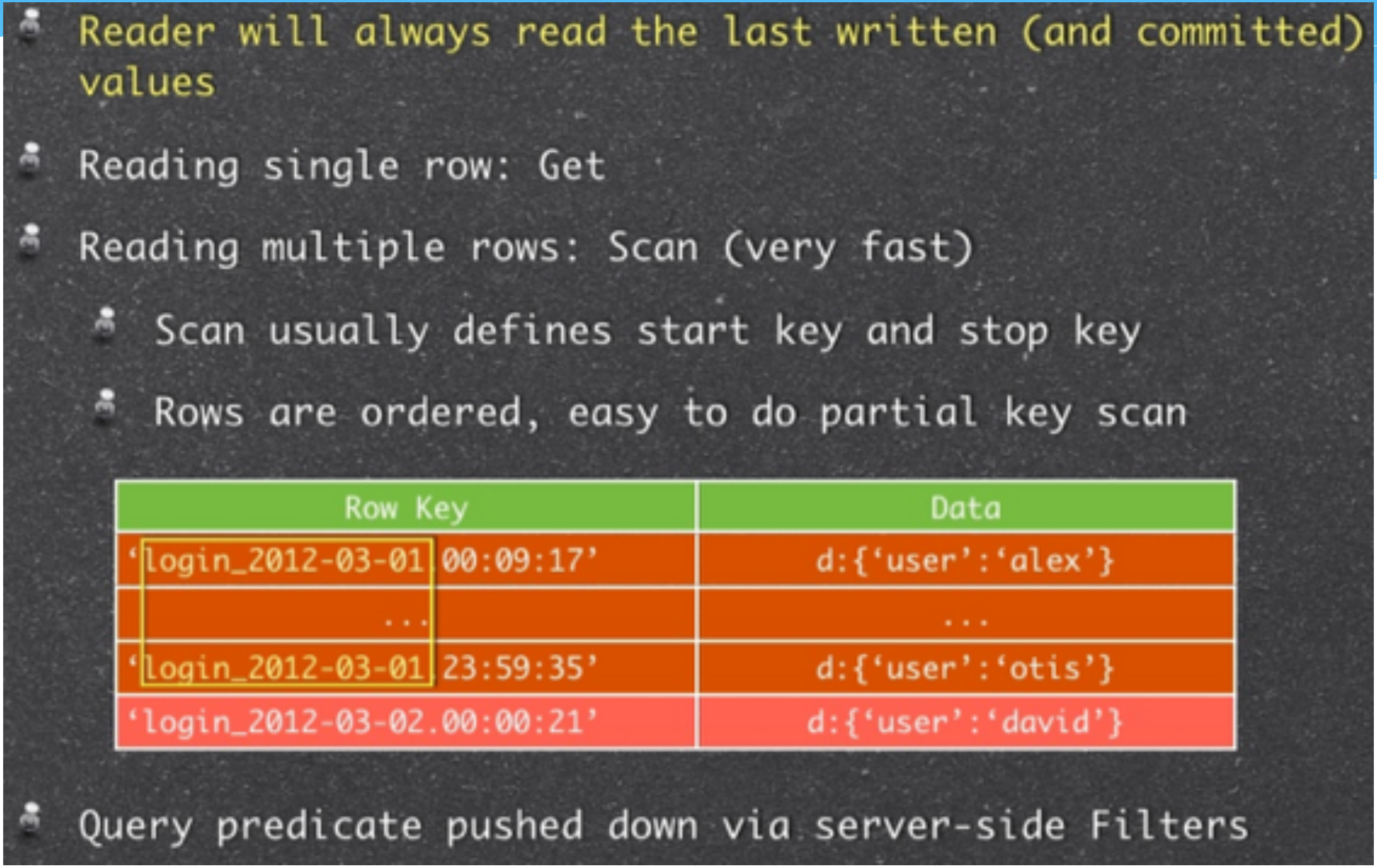
Data Application Lab

Data Writing



- Row updates are atomic
- Updates across multiple rows are NOT atomic, no transaction support out of the box
- HBase stores N versions of a cell (default 3)
- Tables are usually “sparse”, not all columns populated in a row

Reading Data



- Reader will always read the last written (and committed) values
- Reading single row: Get
- Reading multiple rows: Scan (very fast)
  - Scan usually defines start key and stop key
  - Rows are ordered, easy to do partial key scan

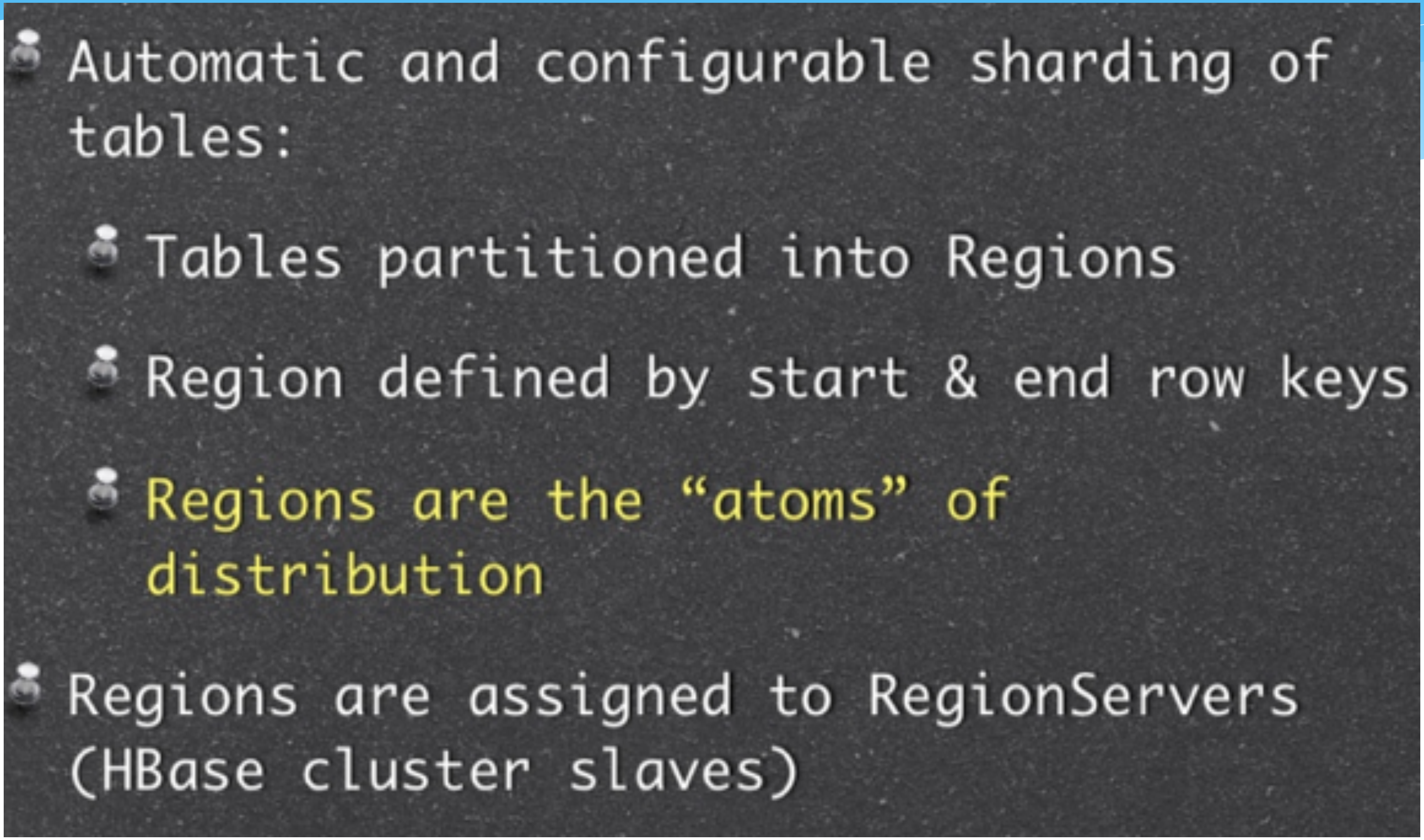
Row Key	Data
‘login_2012-03-01 00:09:17’	d:{‘user’:‘alex’}
...	...
‘login_2012-03-01 23:59:35’	d:{‘user’:‘otis’}
‘login_2012-03-02.00:00:21’	d:{‘user’:‘david’}

- Query predicate pushed down via server-side Filters

HBase Shell Demo

- Create table
- Populate data
- Scan
- Query
- Drop table

Data Sharding and Duplication



- Automatic and configurable sharding of tables:
  - Tables partitioned into Regions
  - Region defined by start & end row keys
  - Regions are the “atoms” of distribution
- Regions are assigned to RegionServers (HBase cluster slaves)

Sharding

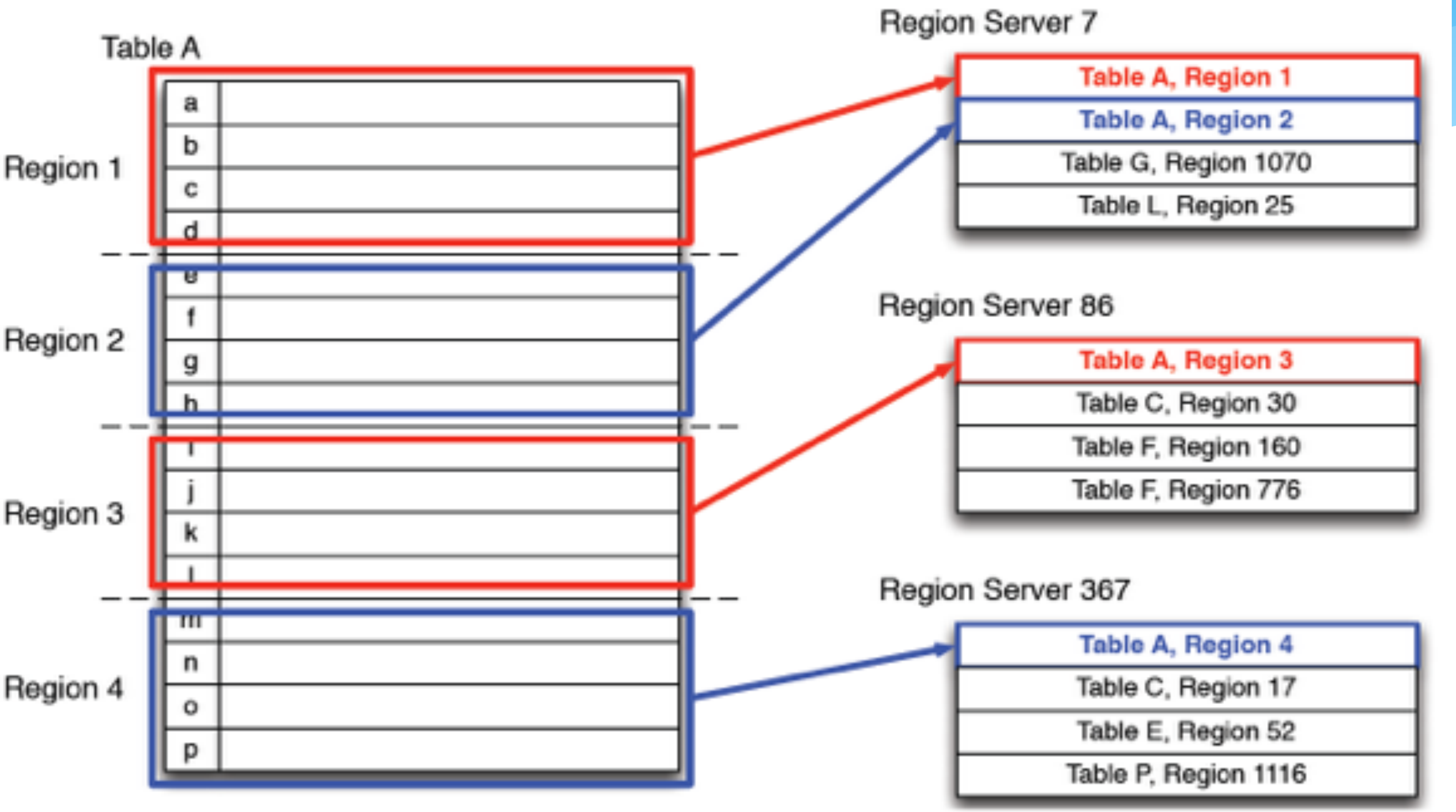


Table A

Region 1

a
b
c
d

Region 2

e
f
g
h

Region 3

i
j
k
l

Region 4

m
n
o
p

Region Server 7

Table A, Region 1
Table A, Region 2
Table G, Region 1070
Table L, Region 25

Region Server 86

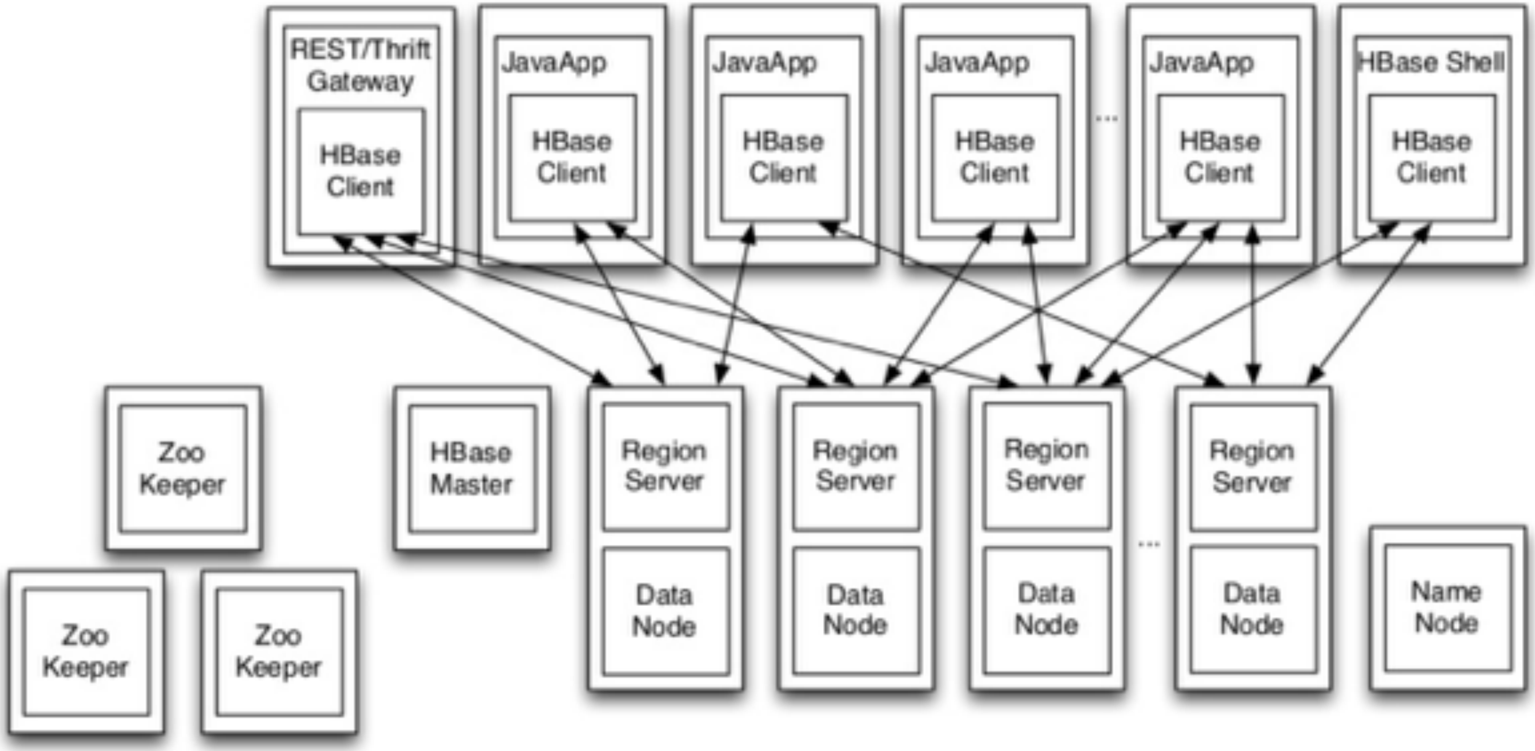
Table A, Region 3
Table C, Region 30
Table F, Region 160
Table F, Region 776

Region Server 367

Table A, Region 4
Table C, Region 17
Table E, Region 52
Table P, Region 1116

Physical Architecture

Distribution and Data Path



Legend:

- An HBase RegionServer is collocated with an HDFS DataNode.
- HBase clients communicate directly with Region Servers for sending and receiving data.
- HMaster manages Region assignment and handles DDL operations.
- Online configuration state is maintained in ZooKeeper.
- HMaster and ZooKeeper are NOT involved in data path.

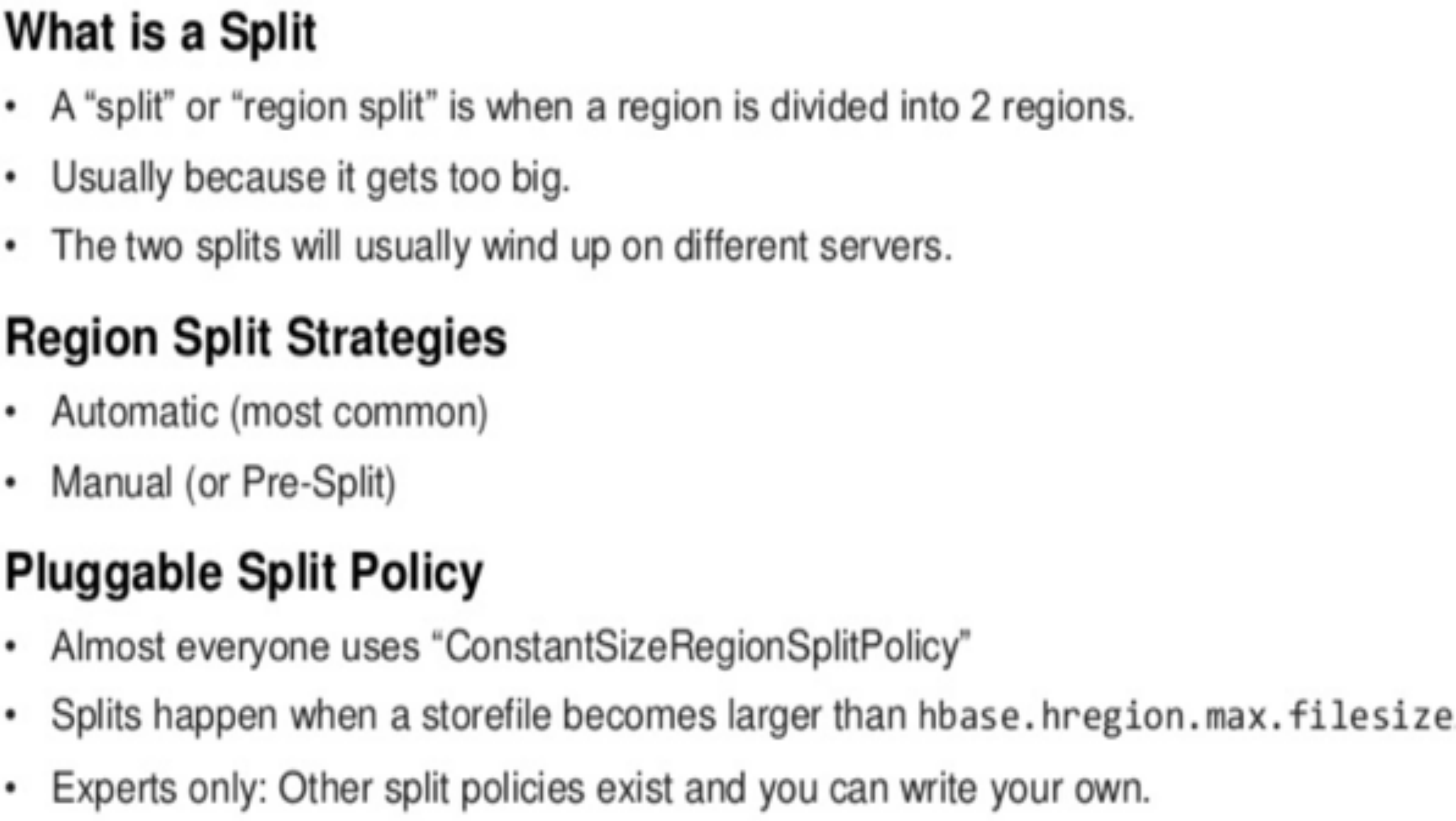
Master Server

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers
- Maintains the state of the cluster by negotiating the load balancing
- Is responsible for schema changes and other metadata operations such as creation of tables and column families

Region Server

- Communicate with the client and handle data-related operations
- Handle read and write requests for all the regions under it
- Decide the size of the region by following the region size thresholds

Region Splits



What is a Split

- A “split” or “region split” is when a region is divided into 2 regions.
- Usually because it gets too big.
- The two splits will usually wind up on different servers.

Region Split Strategies

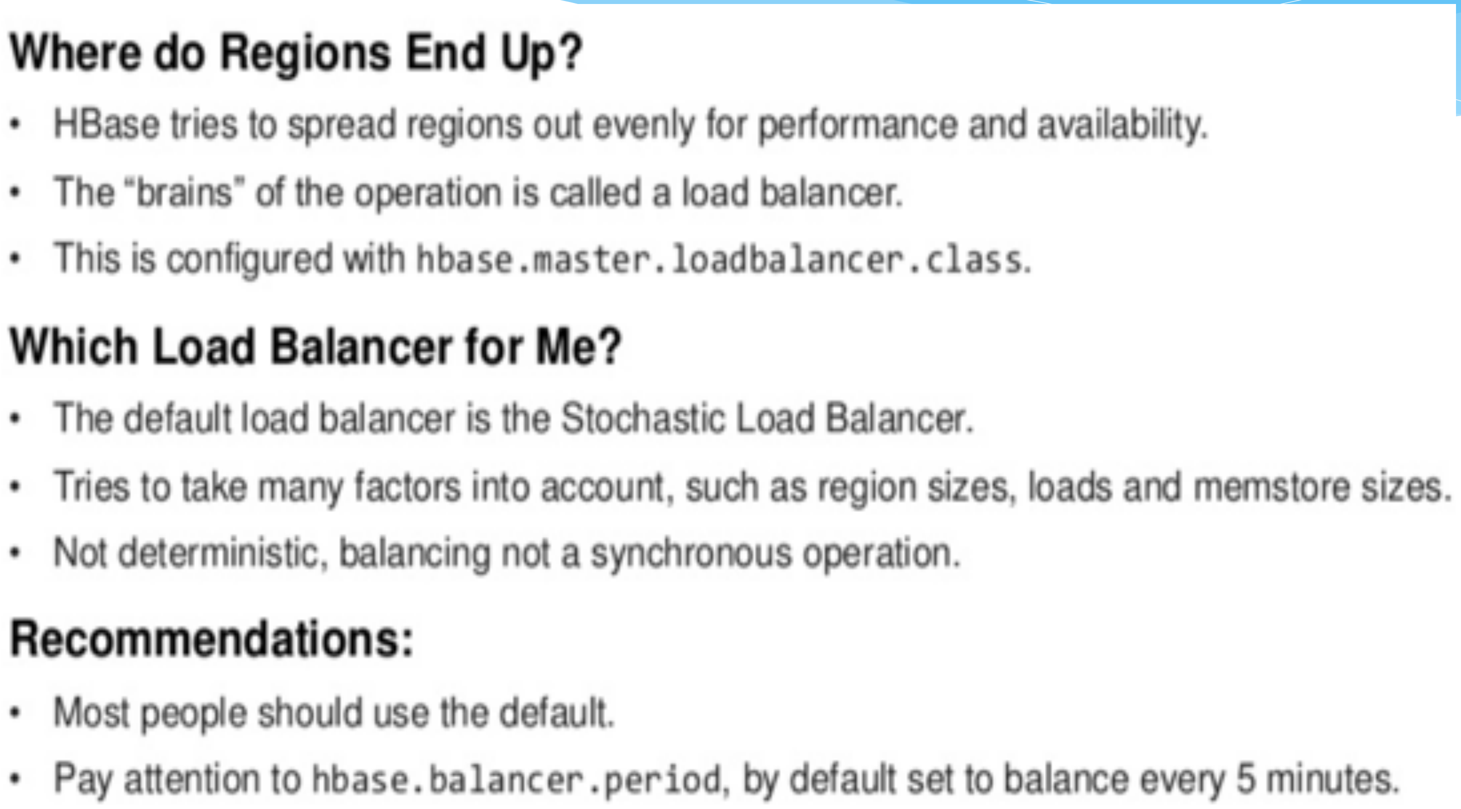
- Automatic (most common)
- Manual (or Pre-Split)

Pluggable Split Policy

- Almost everyone uses “ConstantSizeRegionSplitPolicy”
- Splits happen when a storefile becomes larger than `hbase.hregion.max.filesize`
- Experts only: Other split policies exist and you can write your own.

Data Application Lab

Load Balancer



Where do Regions End Up?

- HBase tries to spread regions out evenly for performance and availability.
- The “brains” of the operation is called a load balancer.
- This is configured with `hbase.master.loadbalancer.class`.

Which Load Balancer for Me?

- The default load balancer is the Stochastic Load Balancer.
- Tries to take many factors into account, such as region sizes, loads and memstore sizes.
- Not deterministic, balancing not a synchronous operation.

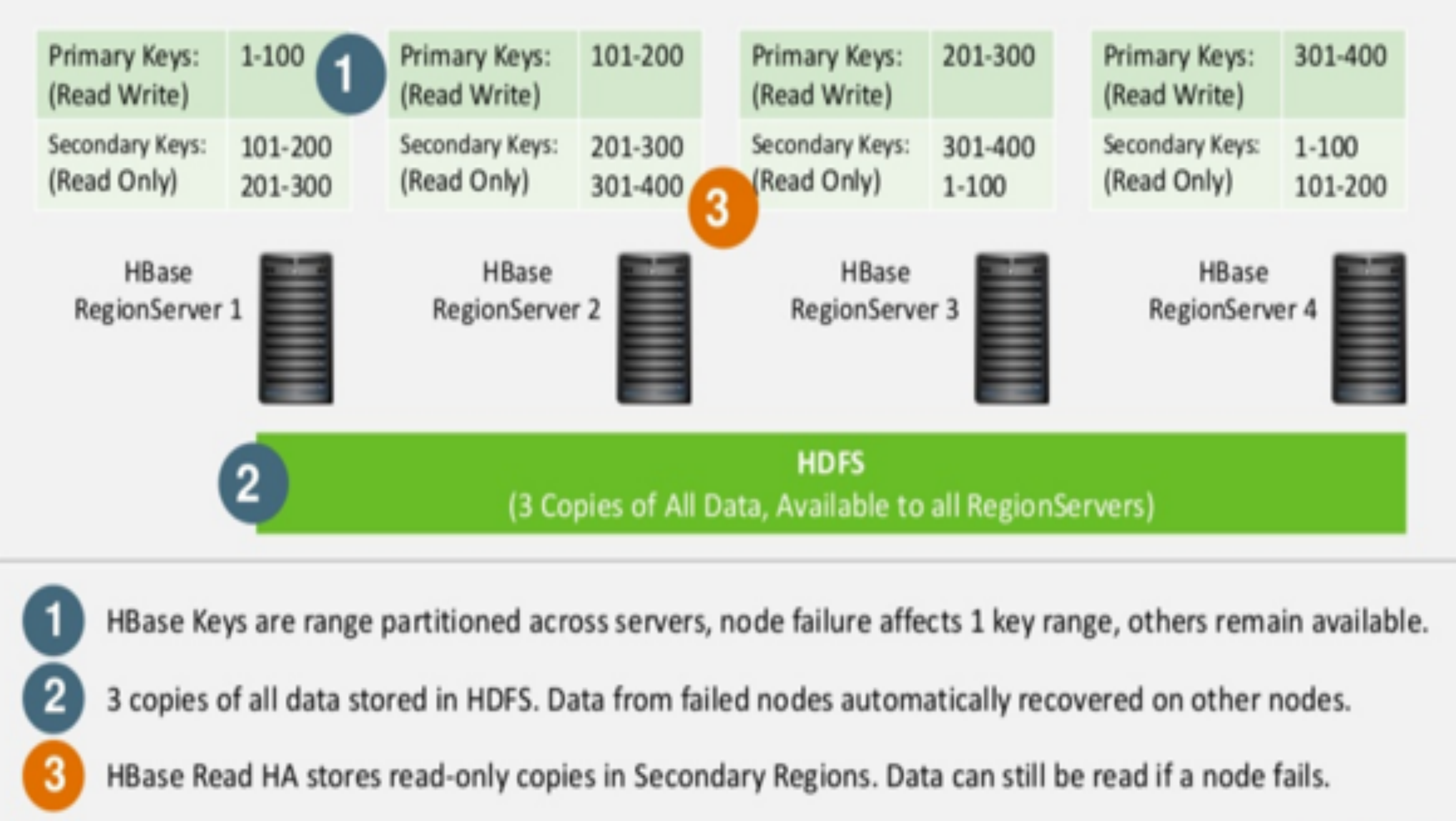
Recommendations:

- Most people should use the default.
- Pay attention to `hbase.balancer.period`, by default set to balance every 5 minutes.

High Availability

- Data is range partitioned across independent RegionServers
- All data is stored in HDFS with 3 copies
- If a region server is lost, data is automatically recover on a remaining region server
- Optionally, data can be hosted on multiple region server, to ensure continuous read availability

Read Write Storage



Primary Keys: (Read Write)	1-100
Secondary Keys: (Read Only)	101-200 201-300

Primary Keys: (Read Write)	101-200
Secondary Keys: (Read Only)	201-300 301-400

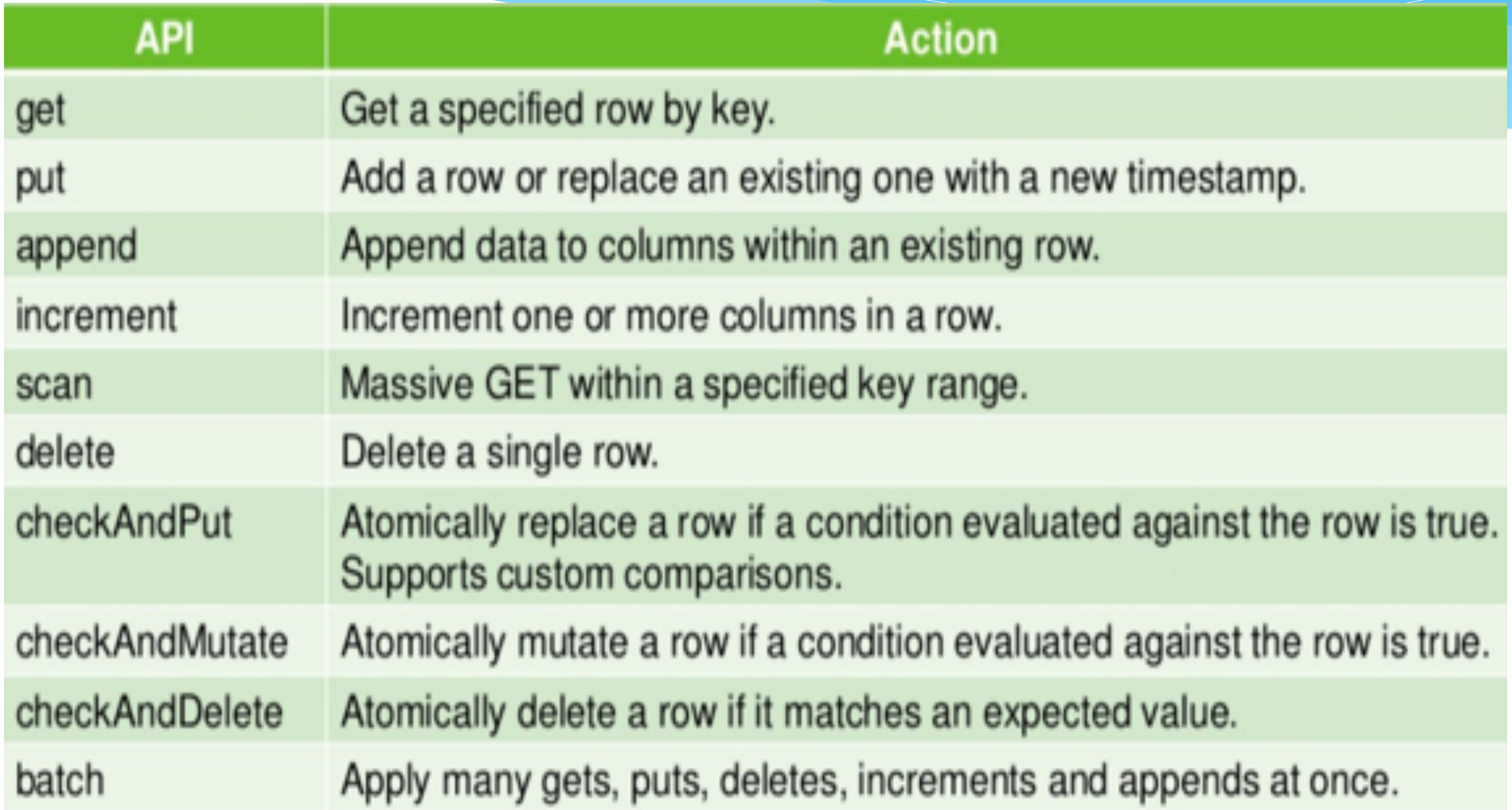
Primary Keys: (Read Write)	201-300
Secondary Keys: (Read Only)	301-400 1-100

Primary Keys: (Read Write)	301-400
Secondary Keys: (Read Only)	1-100 101-200

1. HBase Keys are range partitioned across servers, node failure affects 1 key range, others remain available.
2. 3 copies of all data stored in HDFS. Data from failed nodes automatically recovered on other nodes.
3. HBase Read HA stores read-only copies in Secondary Regions. Data can still be read if a node fails.

Data Application Lab

HBase NoSQL APIs

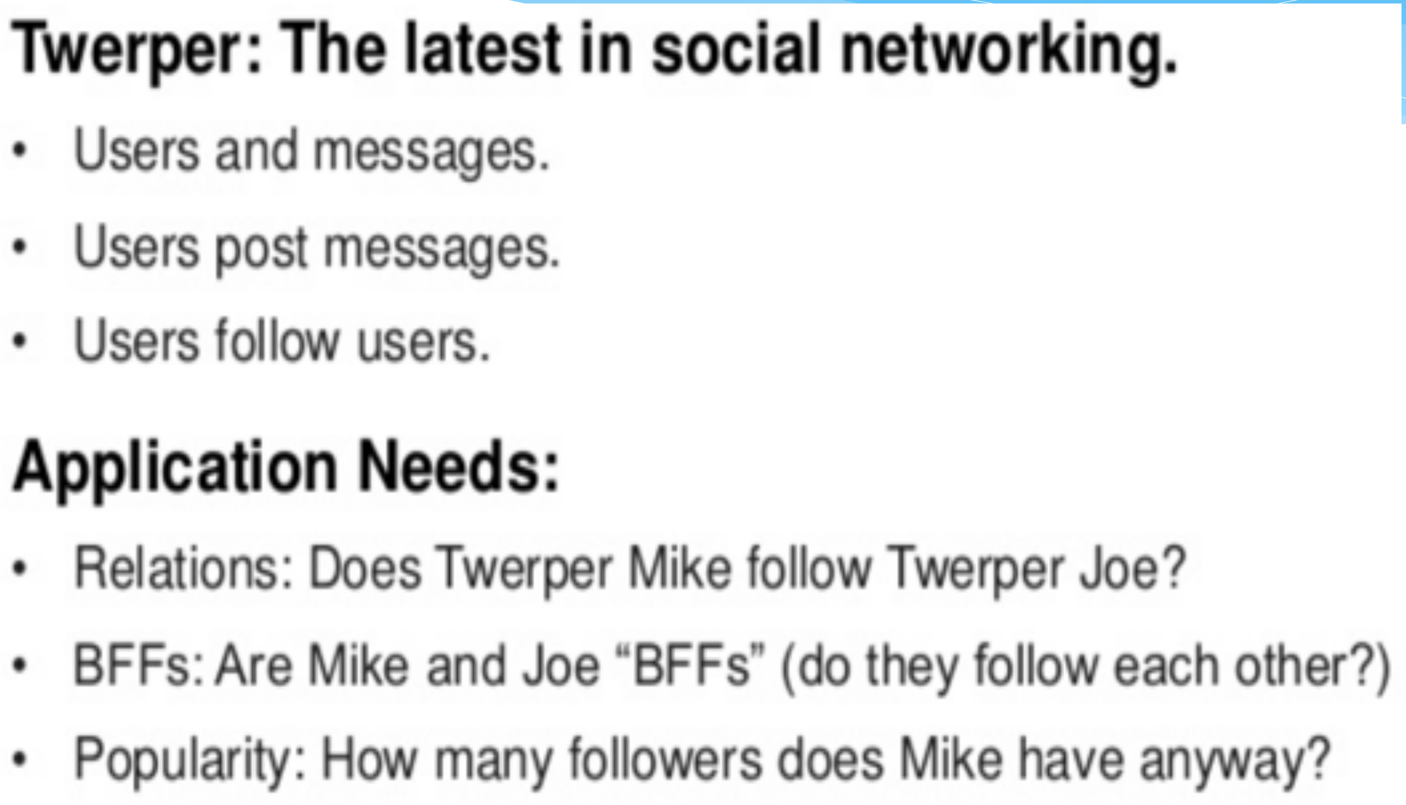


API	Action
get	Get a specified row by key.
put	Add a row or replace an existing one with a new timestamp.
append	Append data to columns within an existing row.
increment	Increment one or more columns in a row.
scan	Massive GET within a specified key range.
delete	Delete a single row.
checkAndPut	Atomically replace a row if a condition evaluated against the row is true. Supports custom comparisons.
checkAndMutate	Atomically mutate a row if a condition evaluated against the row is true.
checkAndDelete	Atomically delete a row if it matches an expected value.
batch	Apply many gets, puts, deletes, increments and appends at once.

Demo- HBase With Storm

Data Application Lab

Why use HBase? Example



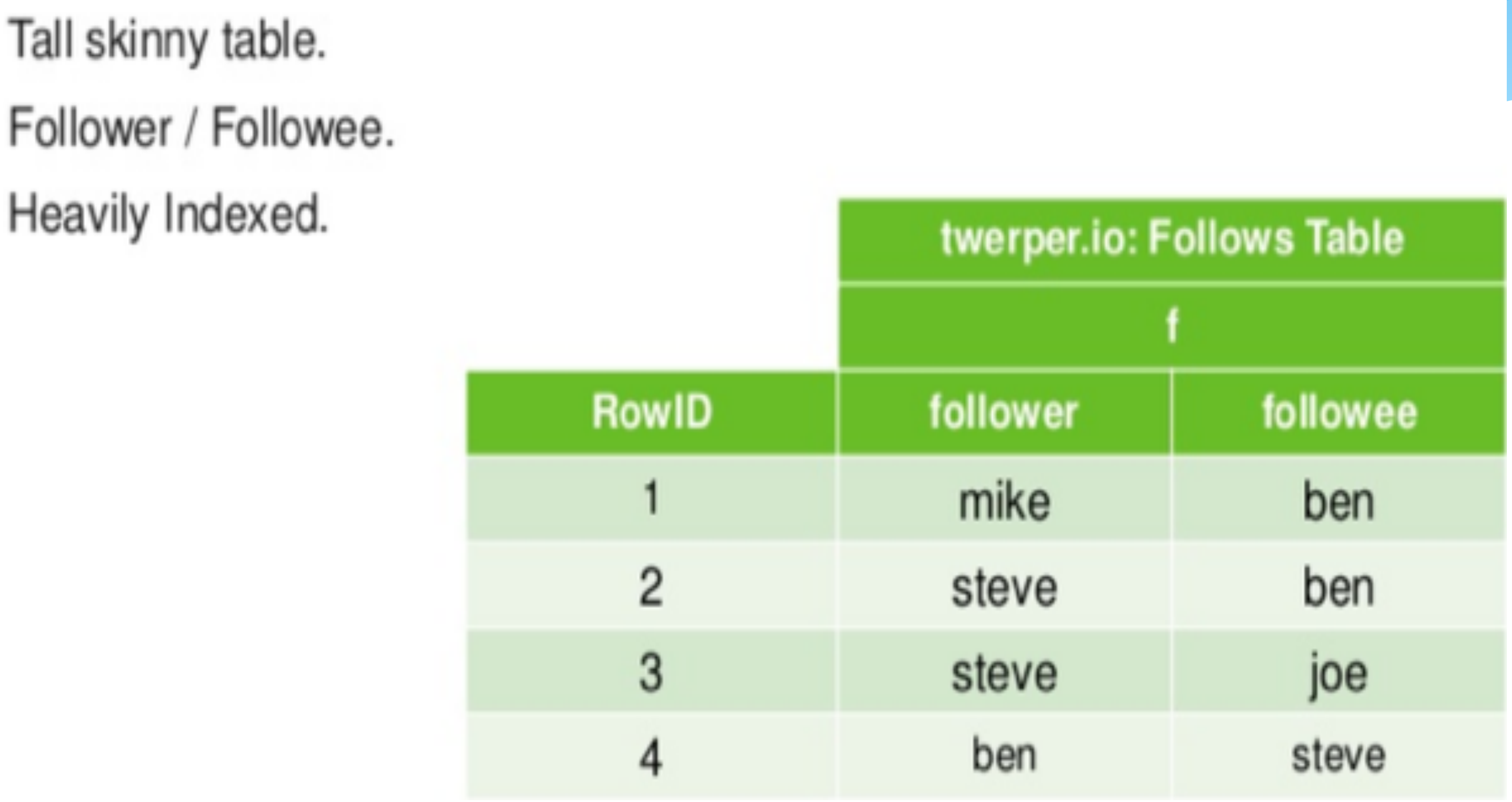
Twerper: The latest in social networking.

- Users and messages.
- Users post messages.
- Users follow users.

Application Needs:

- Relations: Does Twerper Mike follow Twerper Joe?
- BFFs: Are Mike and Joe “BFFs” (do they follow each other?)
- Popularity: How many followers does Mike have anyway?

How would you model this data in RDBMS?



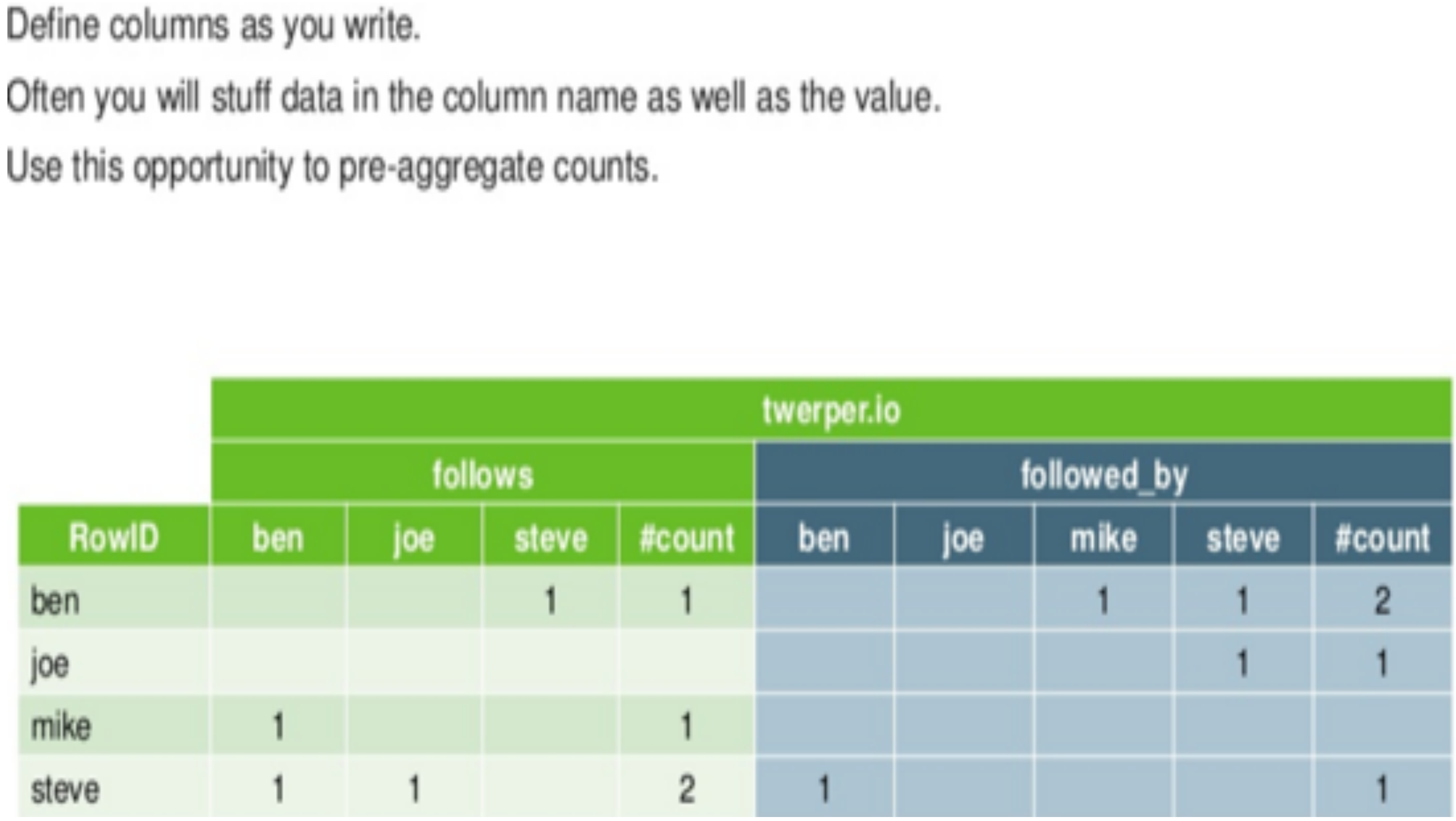
Tall skinny table.

Follower / Followee.

Heavily Indexed.

	twerper.io: Follows Table	
	f	
RowID	follower	followee
1	mike	ben
2	steve	ben
3	steve	joe
4	ben	steve

Schema is Tailored to the Question you are trying to ask.



Define columns as you write.

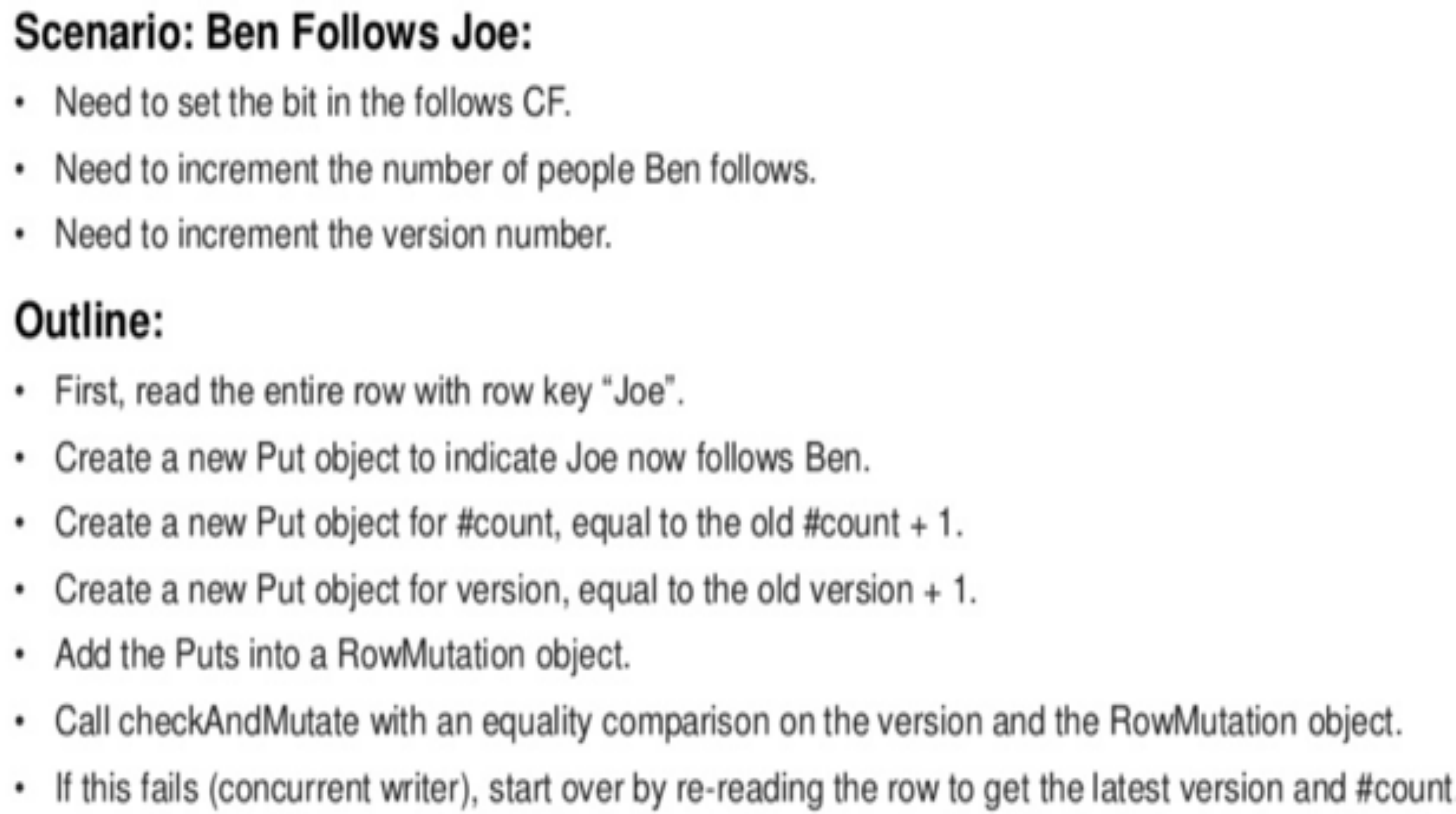
Often you will stuff data in the column name as well as the value.

Use this opportunity to pre-aggregate counts.

	twerper.io								
	follows				followed_by				
RowID	ben	joe	steve	#count	ben	joe	mike	steve	#count
ben			1	1			1	1	2
joe								1	1
mike	1			1					
steve	1	1		2	1				1

Data Application Lab

Check and Mutate



Scenario: Ben Follows Joe:

- Need to set the bit in the follows CF.
- Need to increment the number of people Ben follows.
- Need to increment the version number.

Outline:

- First, read the entire row with row key “Joe”.
- Create a new Put object to indicate Joe now follows Ben.
- Create a new Put object for #count, equal to the old #count + 1.
- Create a new Put object for version, equal to the old version + 1.
- Add the Puts into a RowMutation object.
- Call checkAndMutate with an equality comparison on the version and the RowMutation object.
- If this fails (concurrent writer), start over by re-reading the row to get the latest version and #count

Interview Time

- When to say use HBase?
  - When you already know what type of query you are going to run.
  - Queries are simple, data is large.

- When to say use SQL?
  - When you don't know all the queries you are going to run.
  - When you need complicated joins.

Reference

- https://hbase.apache.org/
- issues.apache.org/jira/browse/hbase
- http://download.bigbata.com/ebook/manning/books/Manning.HBase.in.Action.pdf
- http://hbase.apache.org/book.html#faq

Data Application Lab

THE END