cloudera

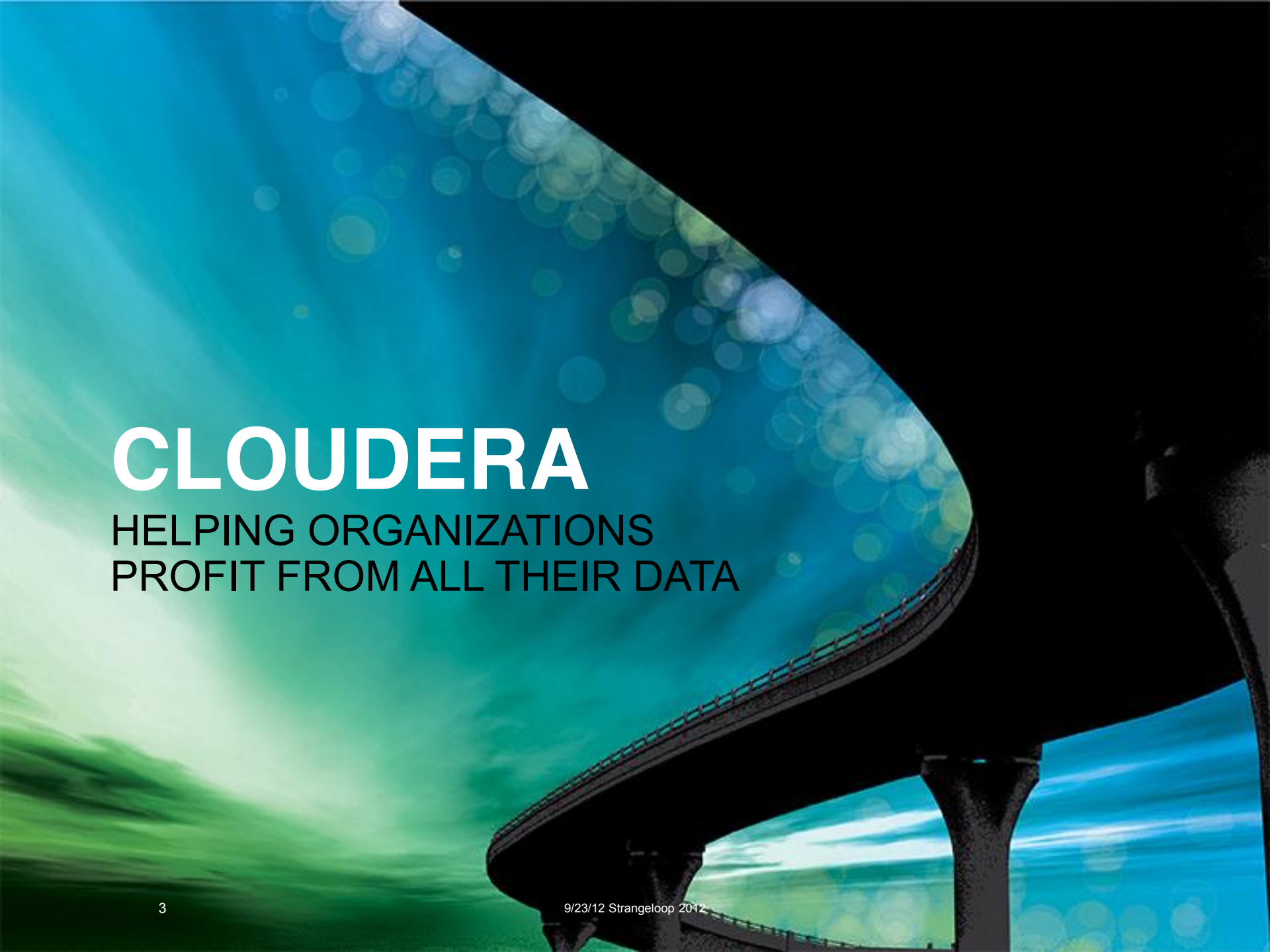
USING HBASE

9/23/12 Strangeloop 2012

Jonathan Hsieh,
jon@cloudera.com
@jmhsieh

Who Am I?

- **Cloudera:**
  - Software Engineer on the Platform Team
  - **Apache HBase** committer / PMC
  - **Apache Flume** founder / committer / PMC
  - **Apache Sqoop** committer / PMC
- **U of Washington:**
  - Research in Distributed Systems


CLOUDERA
HELPING ORGANIZATIONS
PROFIT FROM ALL THEIR DATA

“The future is already here — it's just not very evenly distributed.”

William Gibson

Inspiration: Google BigTable (2006)

- OSDI 2006 paper

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

- **Goal**: **Low latency**, **Consistent**, **random read/write** access to massive amounts of **structured data**.
  - It was the data store for Google's crawler web table, gmail, analytics, earth, blogger, …

Implementation: Apache HBase (2012)

- Web Application Backend
  - Inboxes
  - Catalogs
  - Search index server
  - Web cache
  - Social Media storage
- Monitoring Real-time Analytics
  - OpenTSDB
- A data storage layer for higher level platforms

Outline

- **Enter Apache HBase**
- The HBase Data Model
- System Architecture
- Real-World Applications
- Deployment and Operations
- Conclusions

cloudera

ENTER APACHE HBASE

Low-latency, consistent, random read/write big data access

What is Apache HBase?

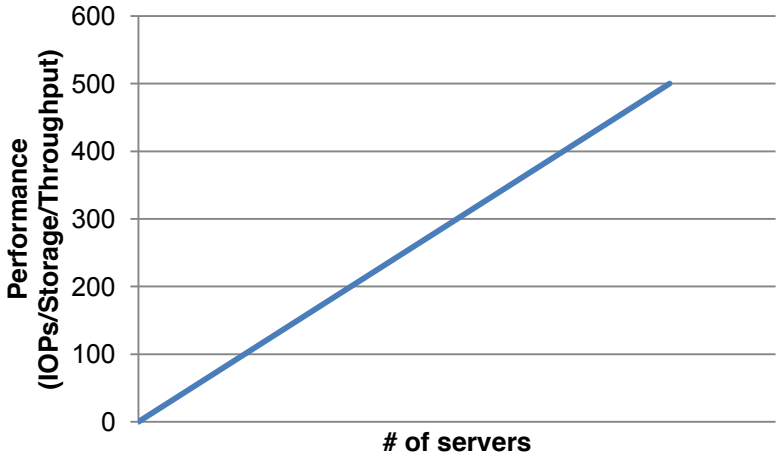
Apache HBase is an **open source**, **horizontally scalable**, **consistent, low latency, random access** data store built on top of **Apache Hadoop**

HBase is Open Source

- **Apache 2.0 License**
- A Community project with committers and contributors from diverse organizations
  - Facebook, Cloudera, StumbleUpon, Salesforce.com, Huawei, TrendMicro, eBay, HortonWorks, …
- Code license means anyone can modify and use the code.

HBase is Horizontally Scalable

- Adding more servers **linearly increases** performance and capacity
  - Storage capacity
  - Input/output operations



- Store and access data on 1-1000's **commodity servers**

- **Largest cluster**: ~1000 nodes, ~1PB
- **Most clusters**: 10-40 nodes, 100GB-4TB

Commodity Servers (circa 2012)

- 12x 1TB hard disks in a **JBOD** (Just a Bunch Of Disks) configuration
- 2 quad core CPUs, 2+GHz
- 24-96GBs of RAM (96GBs if you're considering HBase w/ MR)
- 2x 1Gigabit Ethernet
- $5k-10k / machine

What is Apache HBase?

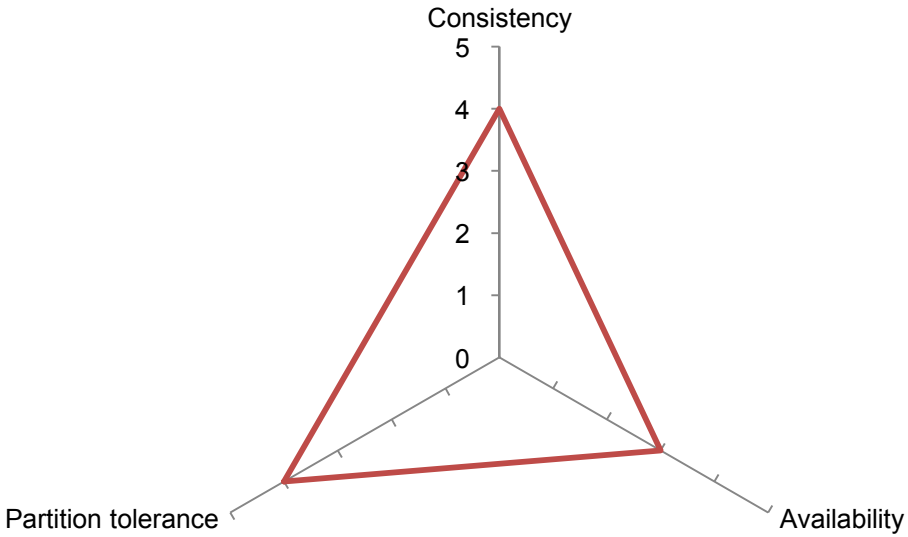
Apache HBase is an **open source**, **horizontally scalable**, **consistent, low latency, random access** big-data store built on top of **Apache Hadoop**

What is Apache HBase?

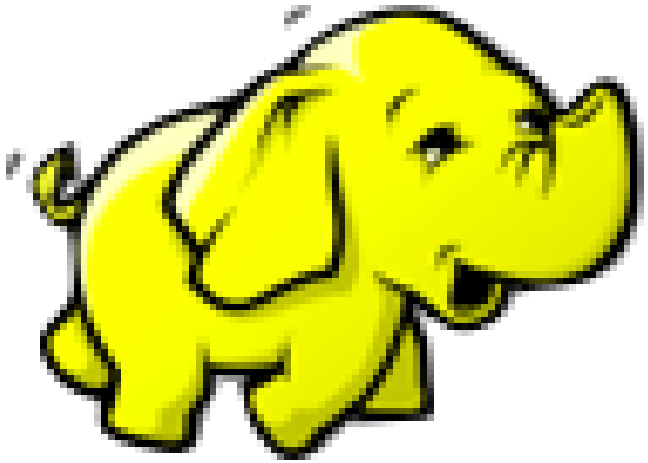
Apache HBase is an **open source**, **horizontally scalable**, **consistent, low latency, random access** big-data store built on top of **Apache Hadoop**

HBase is Consistent

- Brewer's CAP theorem
- **Consistency:**
  - DB-style ACID guarantees on **rows**
- **Availability:**
  - Favor recovering from faults over returning stale data
- **Partition Tolerance:**
  - If a node goes down, the system continues.



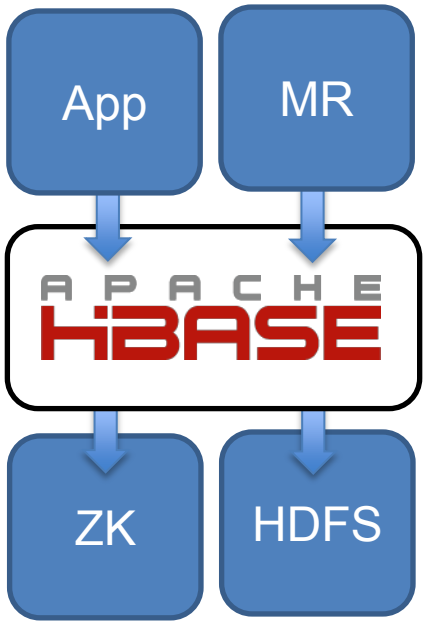
HBase depends on Apache Hadoop



Apache Hadoop is an **open source**, **horizontally scalable** system for **reliably storing** and **processing massive amounts** of data across many **commodity servers**.

HBase Depedencies

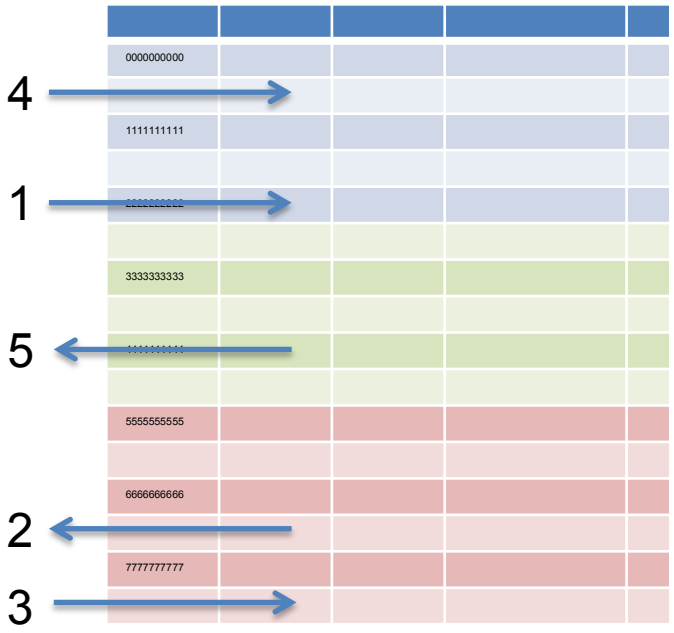
- **Apache Hadoop HDFS** for data **durability** and **reliability** (Write-Ahead Log)
- **Apache ZooKeeper** for distributed coordination
- **Apache Hadoop MapReduce** support built-in support for running MapReduce jobs



HBase does Low-latency Random Access

- **Writes**: 1-3ms, 1k-10k writes/sec per node
- **Reads:** 0-3ms cached, 10-30ms disk
  - 10-40k reads / second / node from cache
- **Cell size:** 0-3MB preferred

- Read, write and insert data anywhere in the table
  - No sequential write limitations



Do I need an HBase or some other “NoSQL” data store?

Did you try scaling your RDBMS vertically?

=>

- Upgrading to a beefier machines could be quick
  - (upgrade that m1.large to a m2.4xlarge)
- This is a good idea.

- What if this isn't enough?

Changed your RDBMS schema and queries?

- Remove text search queries (LIKE)
- Remove joins
  - Joins due to Normalization require expensive seeks
- Remove foreign keys and encode your own relations
  - Avoid constraint checks
- Just put all parts of a query in a single table.

- Lots of Full table scans?
  - Good time for Hadoop.
- Time to consider HBase

Need to scale RDBMS reads?

- Using DB replication to make more copies to read from
- Use Memcached

- Assumes 80/20 read to write ratio, this works reasonably well if can tolerate replication lag.

- Unfortunately, eventually you may need more writes.
  - Replication has diminishing returns with more writes.

Need to scale RDBMS writes?

- Let's shard and federate the DB
  - Loses consistency, order of operations.
- HA introduces operational complexity
- This is definitely a good time to consider HBase

We “optimized the DB” by discarding some fundamental SQL/relational databases features.

Outline

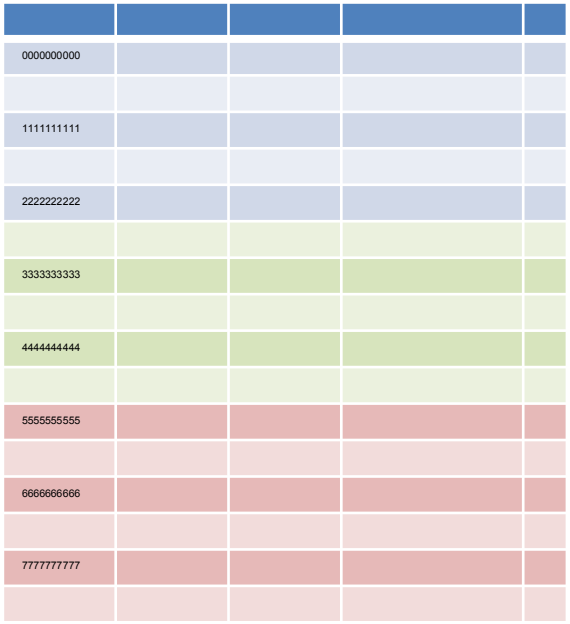
- Enter Apache HBase
- **The HBase Data Model**
- System Architecture
- Real-World Applications
- Deployment and Operations
- Conclusions

cloudera

THE HBASE DATA MODEL

Rows and columns, gets and puts.

Sorted Map Datastore



- It is a big table
- Tables consist of **rows**, each of which has a primary **row key**
- Each row has a set of **columns**
- Rows are stored in **sorted** order

Sorted Map Datastore
(logical view as “records”)

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	‘9ft’	‘CA’	‘Founder’	
tlipcon	‘5ft7’	‘CA’	‘PMC’ @ts=2011 ‘Committer’ @ts=2010	‘Committer’

Sorted Map Datastore

(logical view as “records”)

Implicit PRIMARY KEY in RDBMS terms

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	‘9ft’	‘CA’	‘Founder’	
tlipcon	‘5ft7’	‘CA’	‘PMC’ @ts=2011 ‘Committer’ @ts=2010	‘Committer’

Sorted Map Datastore

(logical view as “records”)

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	‘9ft’	‘CA’	‘Founder’	
tlipcon	‘5ft7’	‘CA’	‘PMC’ @ts=2011 ‘Committer’ @ts=2010	‘Committer’

Sorted Map Datastore
(logical view as “records”)

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	‘9ft’	‘CA’	‘Founder’	
tlipcon	‘5ft7’	‘CA’	‘PMC’ @ts=2011 ‘Committer’ @ts=2010	‘Committer’

A single cell might have different values at different timestamps

Sorted Map Datastore

(logical view as "records")

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	'9ft'	'CA'	'Founder'	
tlipcon	'5ft7'	'CA'	'PMC' @ts=2011 'Committer' @ts=2010	'Committer'

Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Sorted Map Datastore

(logical view as "records")

Implicit PRIMARY KEY in RDBMS terms

Column format family:qualifier

Data is all `byte[]` in HBase

Row key	info: height	info:state	roles:hadoop	roles:hbase
cutting	'9ft'	'CA'	'Founder'	
tlipcon	'5ft7'	'CA'	'PMC' @ts=2011 'Committer' @ts=2010	'Committer'

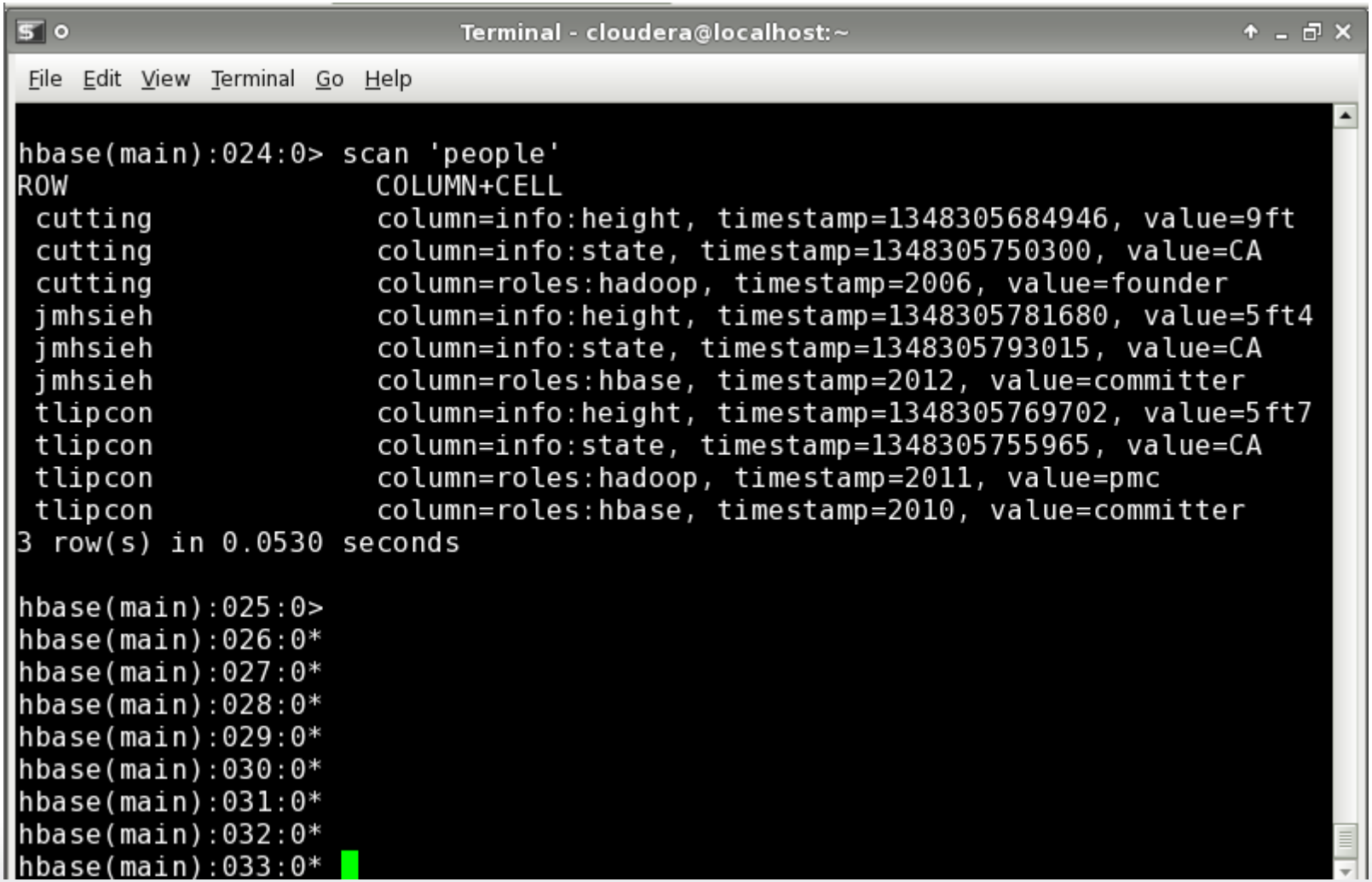
Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Anatomy of a Row

- Each row has a **primary key**
  - Lexicographically sorted byte[]
  - Timestamp associated for keeping **multiple versions of data** (MVCC for consistency)
- Row is made up of columns.
- Each (row,column) referred to as a **Cell**
  - Contents of a cell are all byte[]'s.
  - Apps must "know" types and handle them.
- Rows are **Strongly consistent**

Example in Hbase Shell

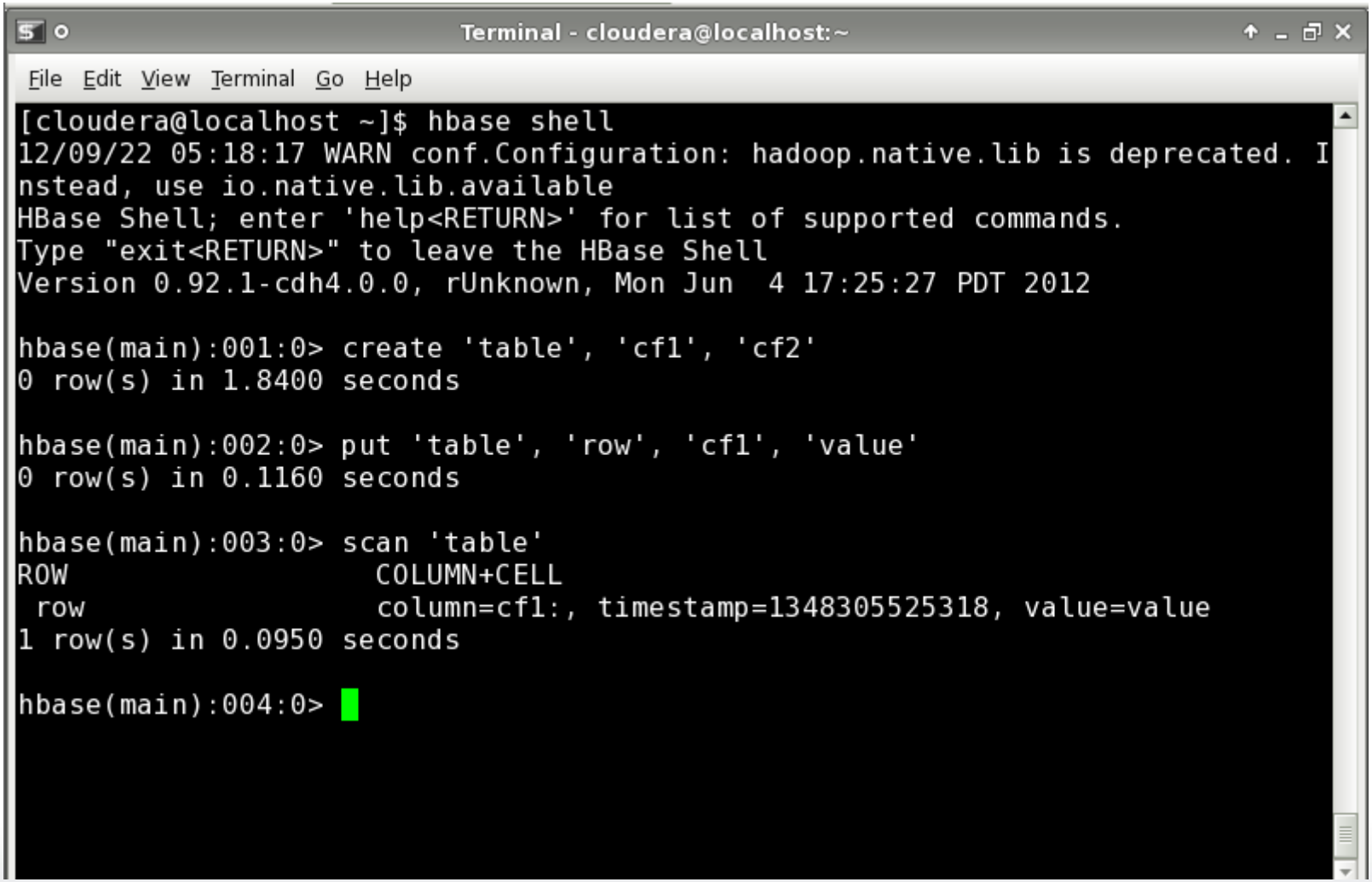


Access HBase data via an API

- Data operations
  - Get
  - Put
  - Scan
  - Increment
  - CheckAndPut
  - Delete

- Access via HBase shell, Java API, REST proxy

Shell: Create a table, add, and read a row



Java API

```
byte[] row = Bytes.toBytes(“row”);
byte[] col = Bytes.toBytes(“cf1”);
byte[] putVal = Bytes.toBytes(“your boat”);
Configuration config = HBaseConfiguration.create();

HTable table = new HTable(config, “table”);
Put p = new Put(row);
p.add(col, putVal)
table.put(p);

Get g = new Get(row);
Result r = table.get(g);
byte[] getVal = r.getValue(col);
assertEquals(putVal, getVal);
```

REST Server

- bin/hbase rest start –p 8070

- Browse:
- curl –H “Accept: application/json/” http://localhost:8070/...
  - http://localhost:8070/ tables
  - http://localhost:8070/people/schema schema
  - http://localhost:8070/people/tlipcon row

- More info here
  - http://wiki.apache.org/hadoop/Hbase/Stargate

Simple API. What is the catch?

With great power comes great responsibility.

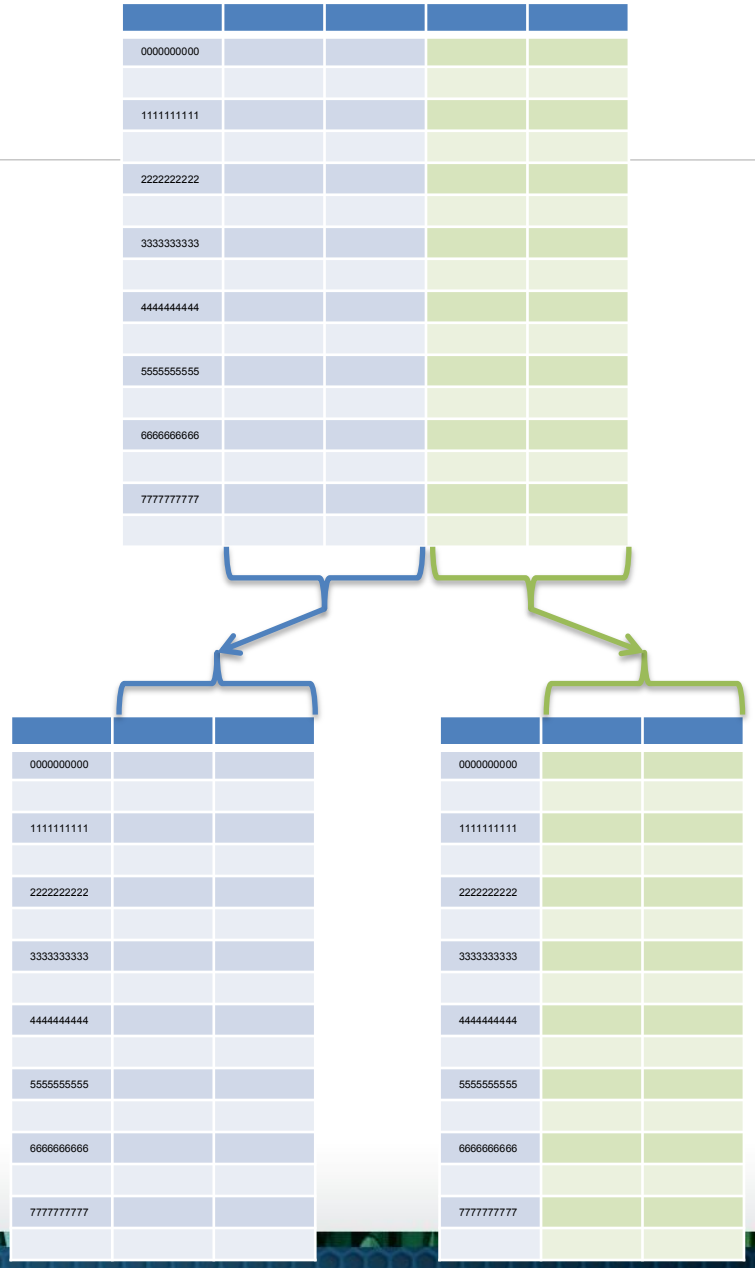
Cost Transparency

- Goal: Predictable latency of random read and write operations.
  - Now you have to understand some of the physical layout of your datastore.

- Efficiencies are based on **Locality** and your **schema.**
- Need to understand some physical concepts:
  - Column Families
  - Sparse Columns
  - Regions
- Your schema needs to consider these.

Column Families

- A **Column family** is a set of related columns.
- Group sets of columns that have similar access patterns
- Select parameters to tune read performance per column family



Physical Storage of Columns Families

Each column family is contained in its own file.

`info` Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` Column Family

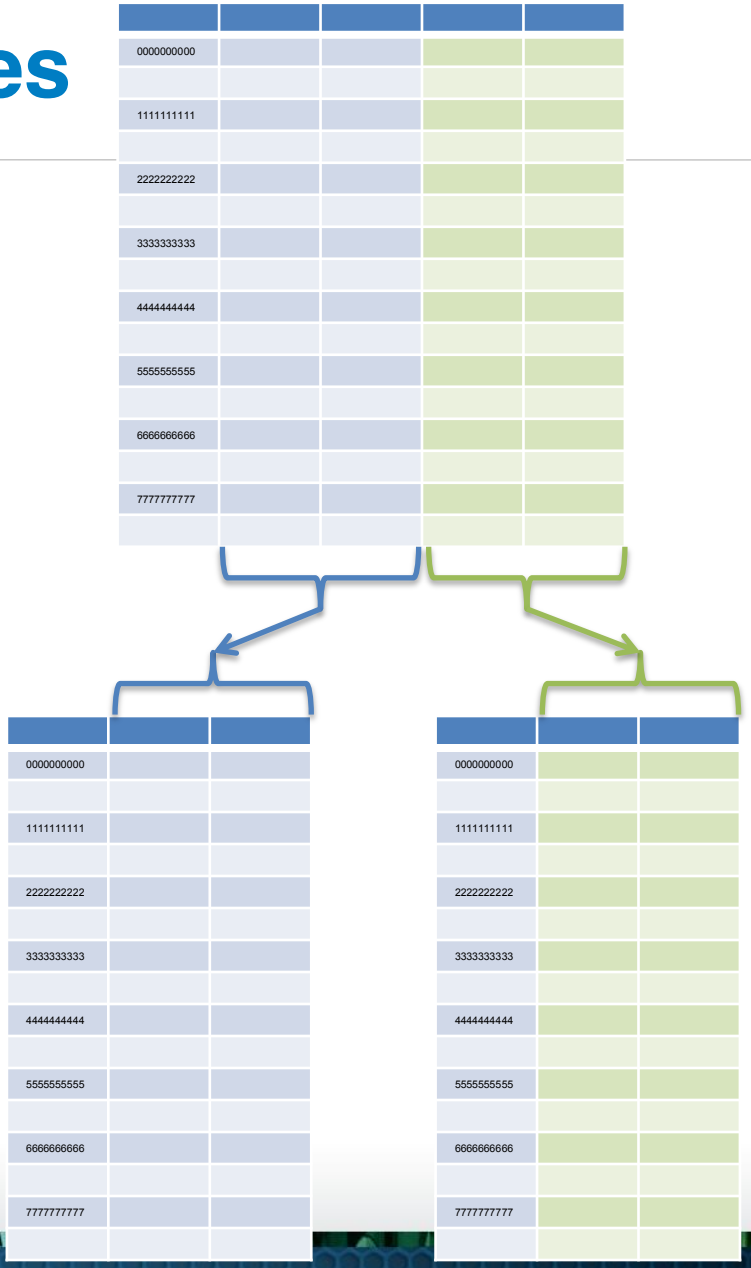
Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

Tuning Column Families

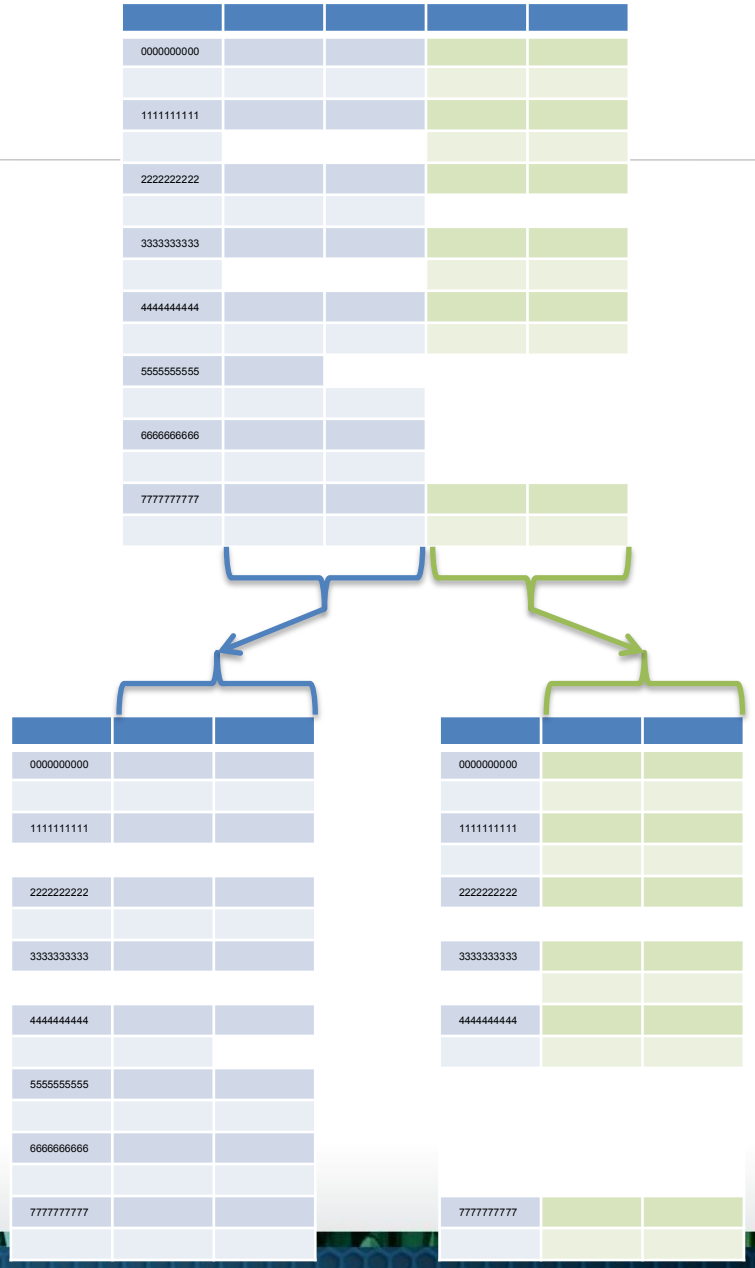
- Good for tuning **read performance**
  - Store related data together for better compression
  - Avoid polluting cache from another.
  - Derived data can have different retention policies
- Column family parameters
  - Block Compression (none, gzip, LZO, Snappy)
  - Version retention policies
  - Cache priority



cloudera

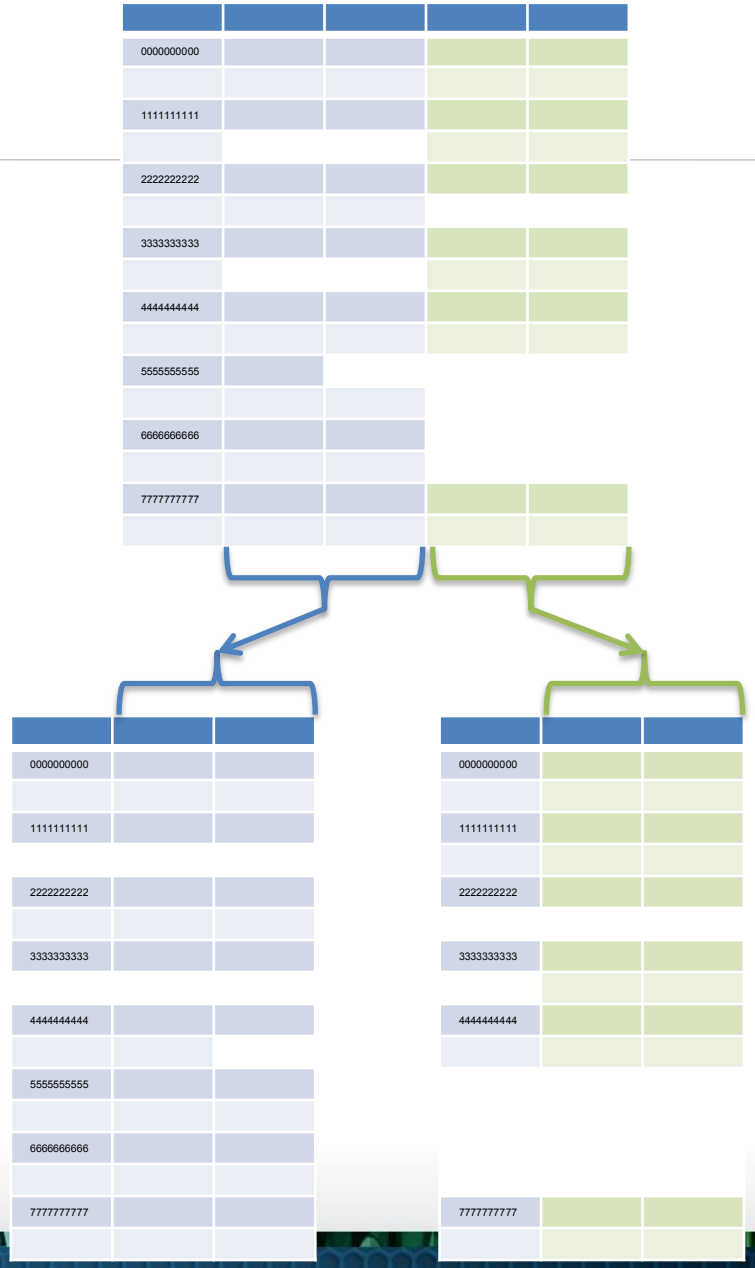
Sparse Columns

- Sparseness provides **schema flexibility**
  - Add columns later, no need to transform entire schema
  - If you find yourself adding columns to your db, HBase is a good model.



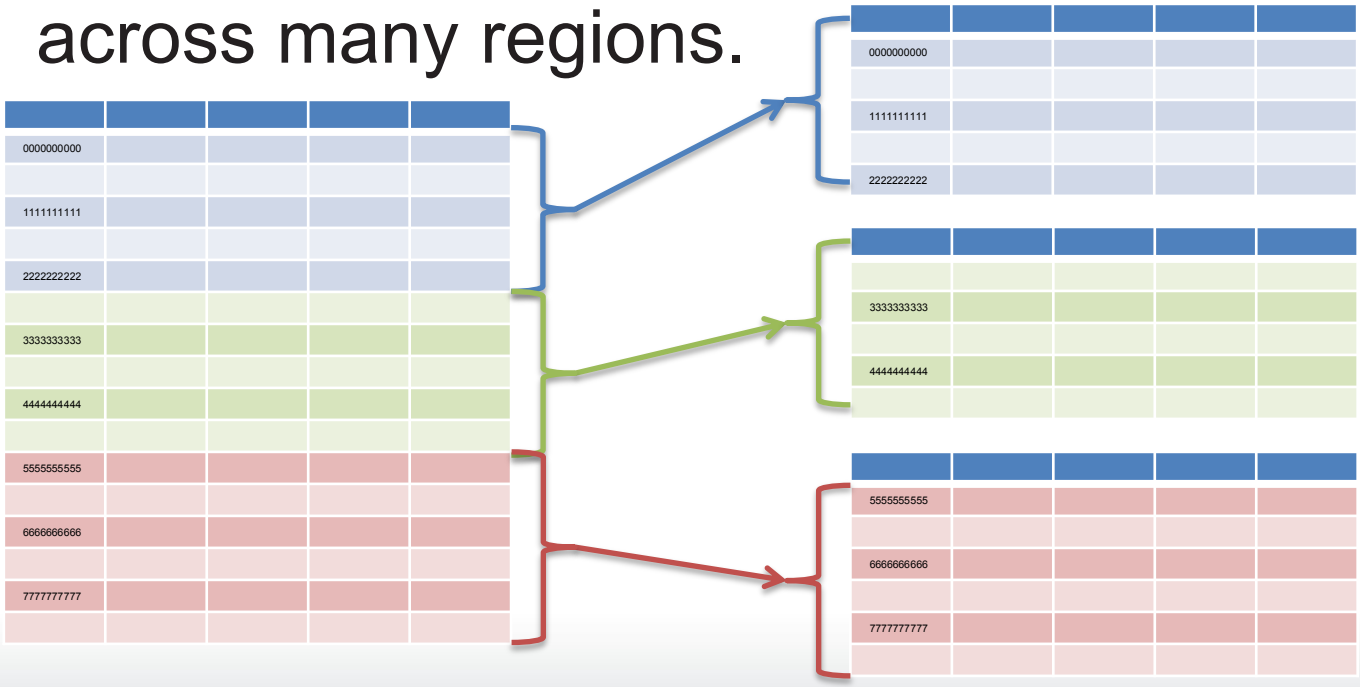
Sparse Columns

- Sparseness can improve performance
  - Null columns don't take space. You don't need to read what is not there.
  - If you have a traditional db table with lots of nulls, you data will probably fit well



Horizontal Scaling - Regions

- Tables are divided into sets of rows called **regions**
- **Scale read and write capacity** by spreading across many regions.



Regions: Tradeoffs

- Easier to scale cluster capacity
  - Auto sharding and load balancing capability
- Greater throughput and storage capacity
  - Horizontal scalability of writes and reads and storage
- Enough consistency for many applications
  - Per row ACID guarantees

- No built-in atomic multi-row operations
- No built-in consistent secondary indices
- No built-in global time ordering

SQL + HBase

- No built-in SQL query language and query optimizer
- There is work on integration Apache Hive (SQL-like query language)
  - Currently not the optimal, x5 slower than normal Hive+HDFS
- Apache Sqoop and HBase Integration
  - Copy RDMS tables from database to Hbase
  - Copy HBase Tables into RDMS
  - (there is some impedance mismatch)

HBase vs RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Column family level Authentication / Authorization
Indexes	On arbitrary columns	Row-key only*
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of “queries”/second

Outline

- Enter Apache HBase
- The HBase Data Model
- **System Architecture**
- Real-World Applications
- Deployment and Operations
- Conclusions

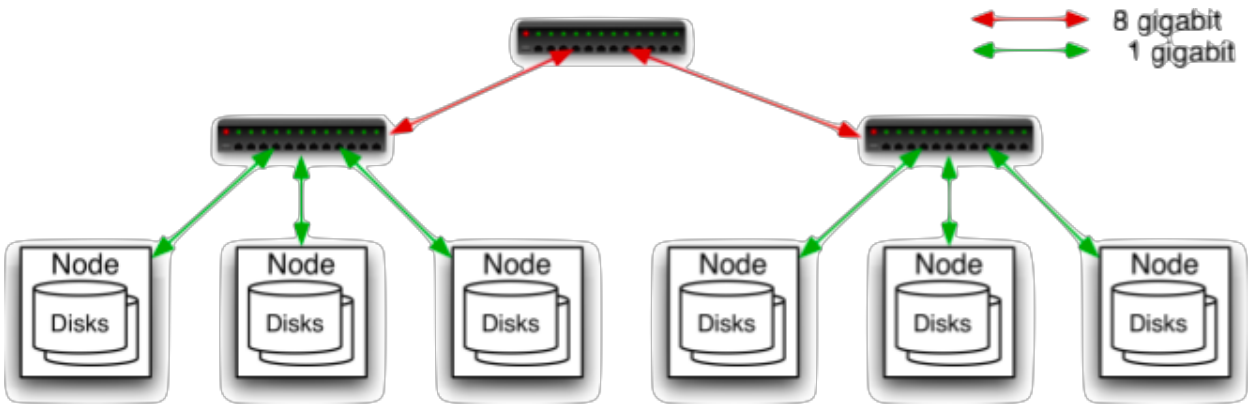
cloudera

SYSTEM ARCHITECTURE

Clusters, Nodes, and Dependencies

A Typical Look...

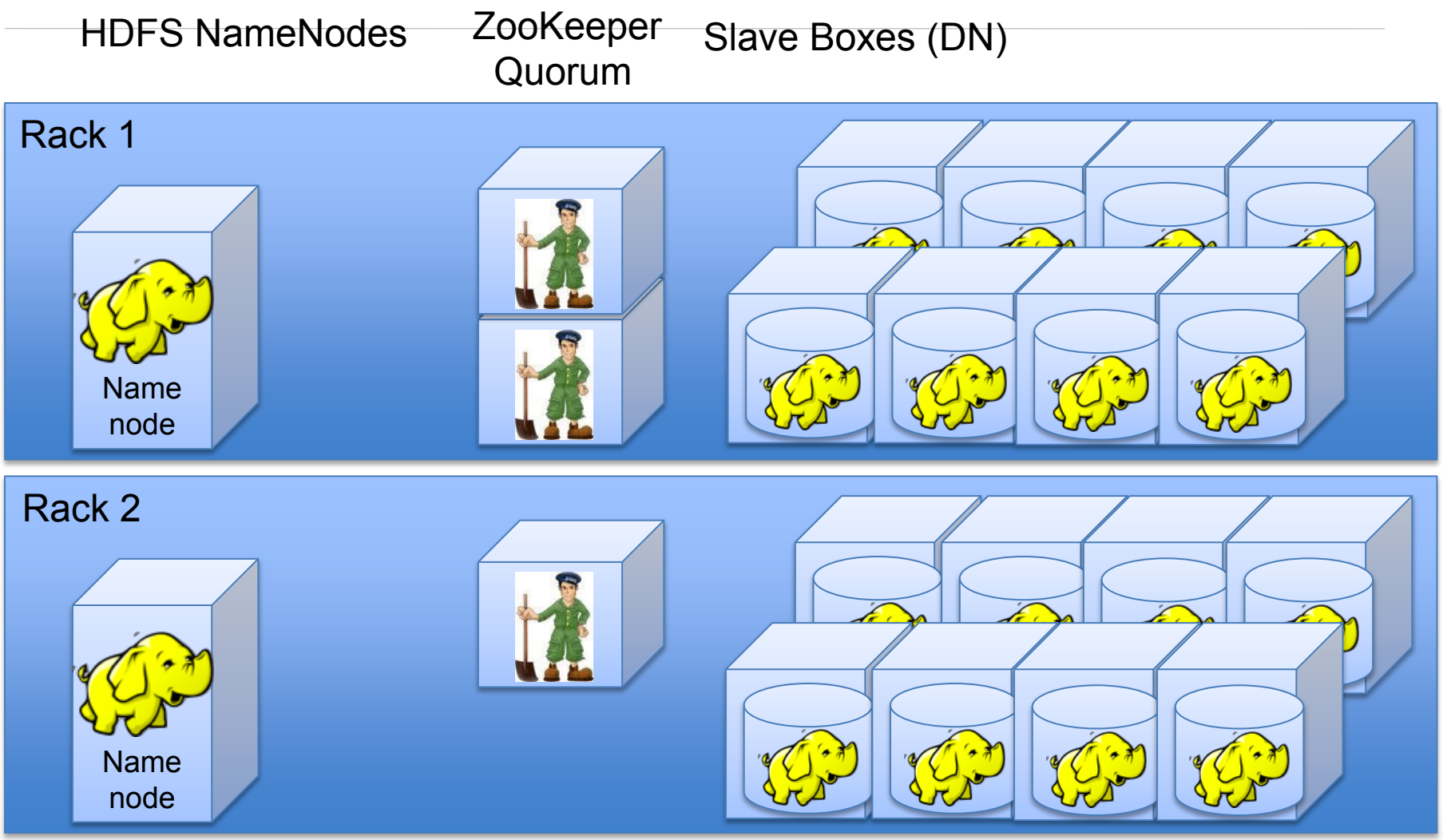
- 5-4000 commodity servers (8-core, 48 GB RAM, 12-24 TB, gig-E)
- 2-level network architecture
  - 20-40 nodes per rack



HBase Processes

- HDFS
  - Name node
  - Data node
- ZooKeeper
  - Quorum Peer
- HBase
  - Master
  - Region Server

HDFS Nodes (physically)



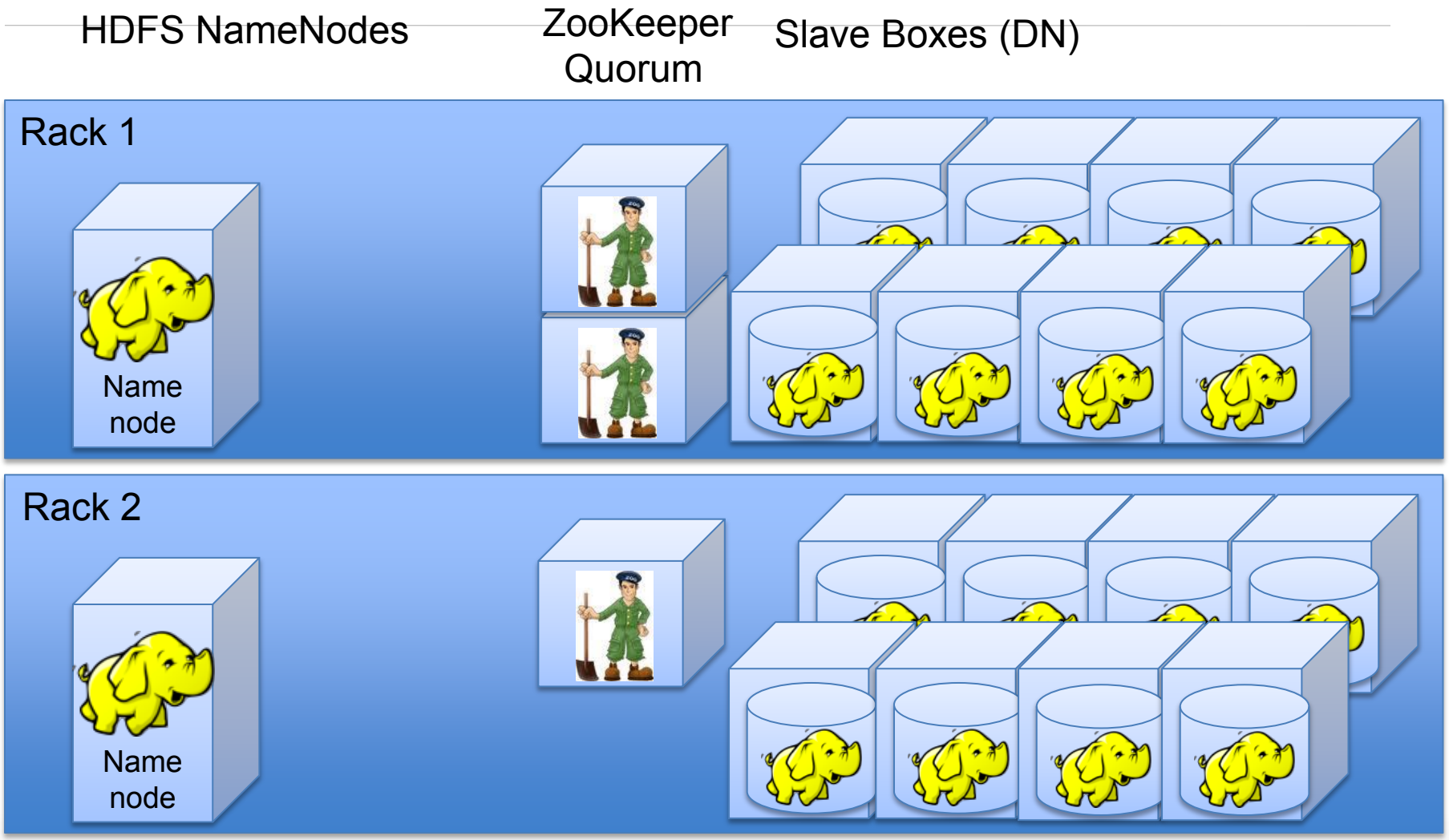
HDFS Name Node

- Stores file system metadata on disk and in memory
  - Directory structures, permissions
  - Modifications stored as an edit log
- Fault tolerant and Highly Available

HDFS Data nodes

- HDFS splits the files into 128MB (or 256MB) blocks
- **Data nodes** store and serve these Blocks
  - By default, pipeline writes to 3 different machines.
  - By default, local machine, machines on other racks.
  - Locality helps significantly on subsequent reads and computation scheduling.

HDFS Nodes (physically)

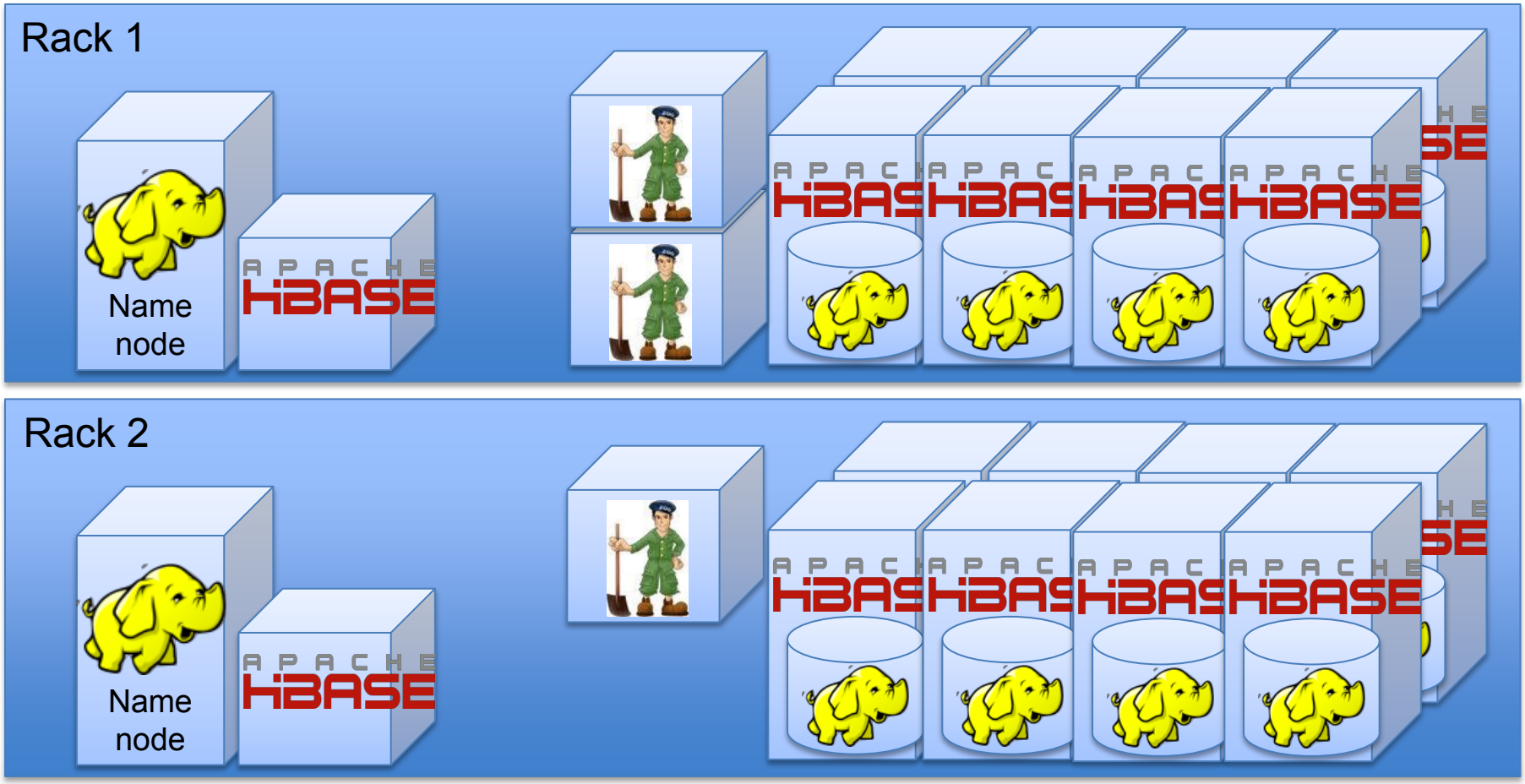


HBase + HDFS Nodes (physically)

HDFS NameNodes
HBase Masters

ZooKeeper Quorum

Slave Boxes (DN + RS)



HMaster and ZooKeeper

- HMaster
  - Controls which Regions are served by which Region Servers.
  - Assigns regions to new region servers when they arrive or fail.
  - Can have a hot standby master if active master goes down.
  - Metadata state kept in ZooKeeper
- Apache ZooKeeper
  - Highly Available System for coordination.
  - Generally 3 or 5 machines (always an odd number)
  - Uses consensus to guarantee common shared state.
  - Writes are considered expensive

Region Server

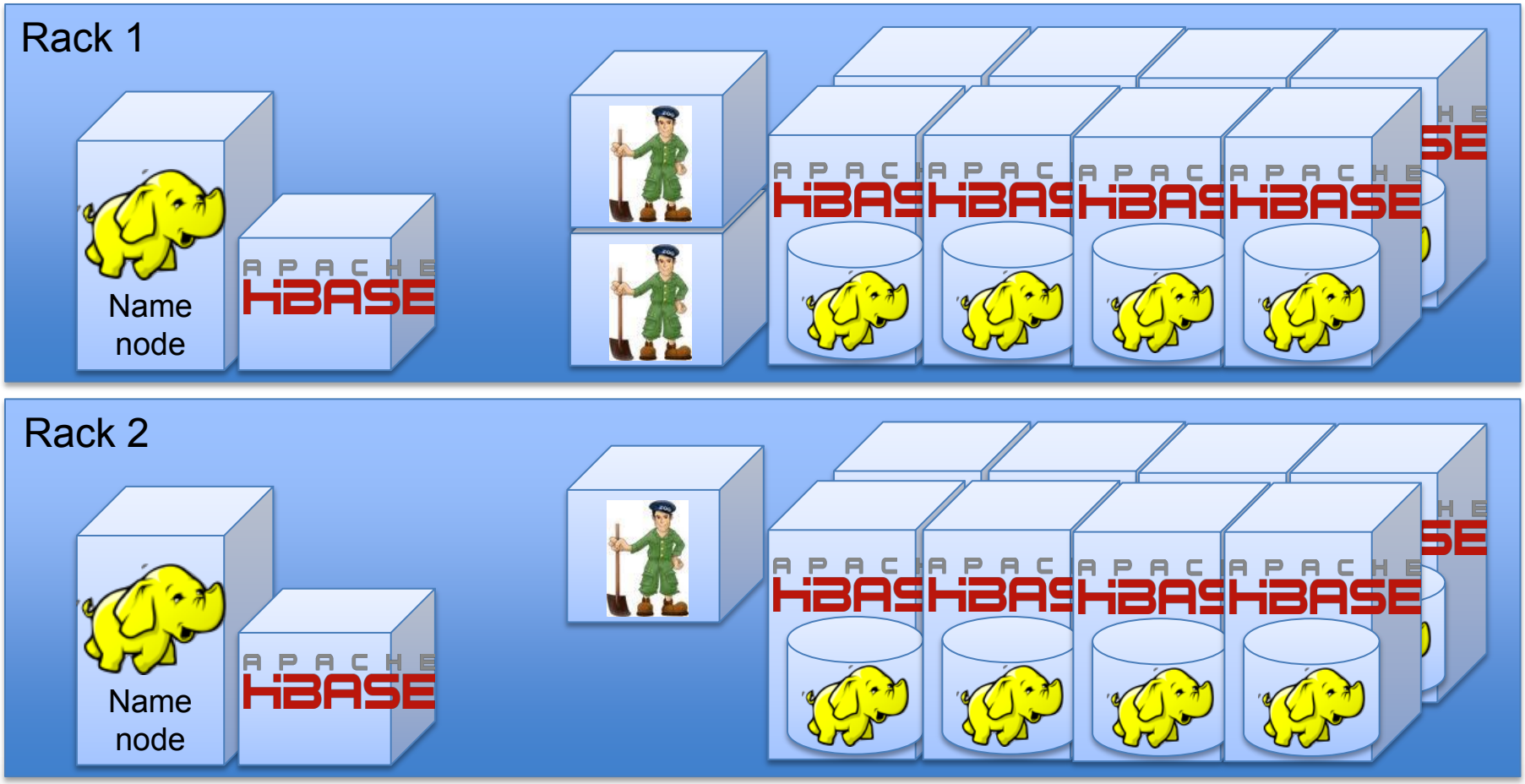
- Tables are chopped up into regions
- A region is only served by a single “region server” at a time.
- Region Server can serve multiple regions
  - Automatic load balancing if region server goes down.
- Co-locate region servers with data nodes.
  - Takes advantage of HDFS file locality
- Important that clocks are in reasonable sync. Use NTP!

HBase + HDFS Nodes (physically)

HDFS NameNodes
HBase Masters

ZooKeeper Quorum

Slave Boxes (DN + RS)

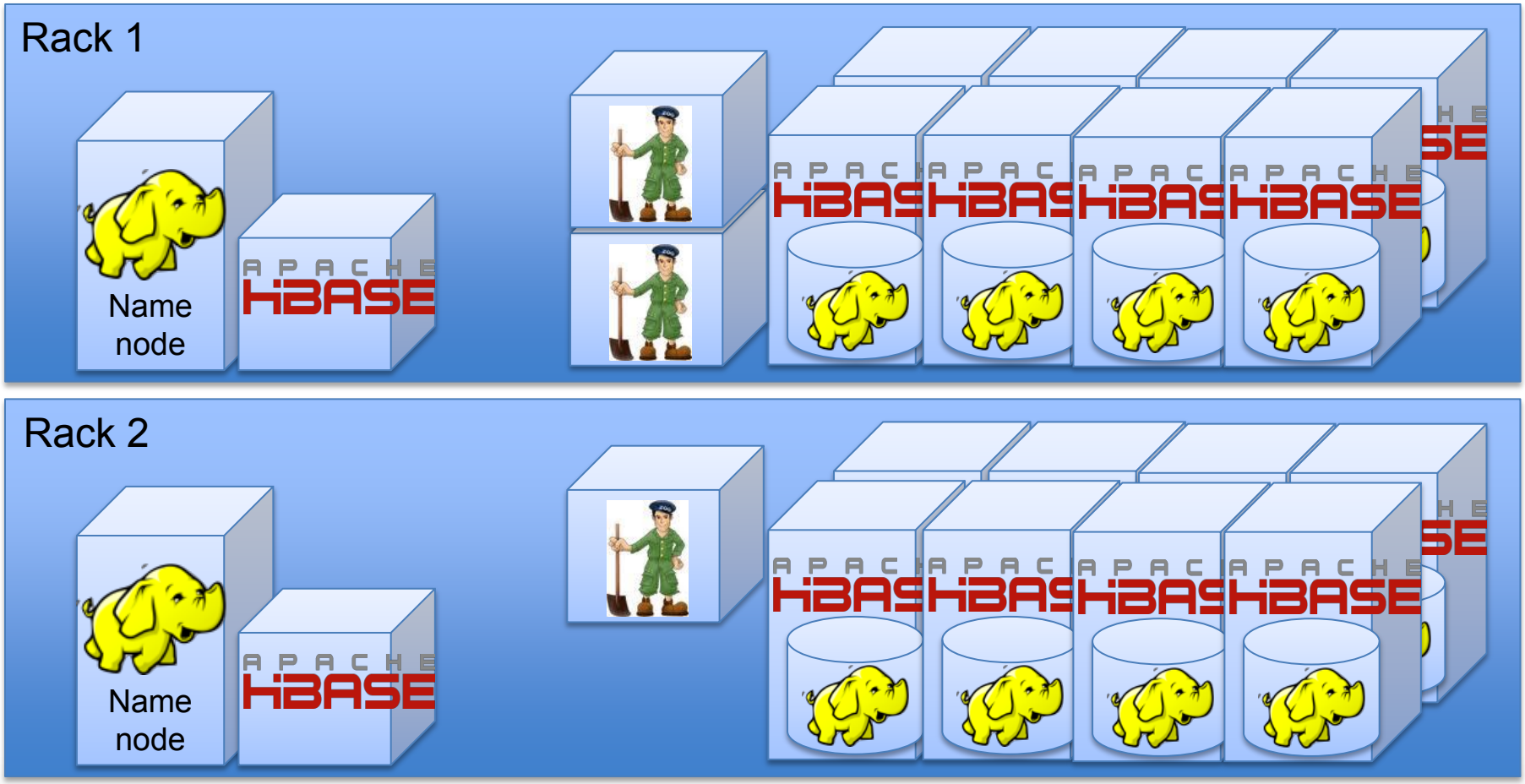


HBase + HDFS: No SPOF

HDFS NameNodes
HBase Masters

ZooKeeper Quorum

Slave Boxes (DN + RS)



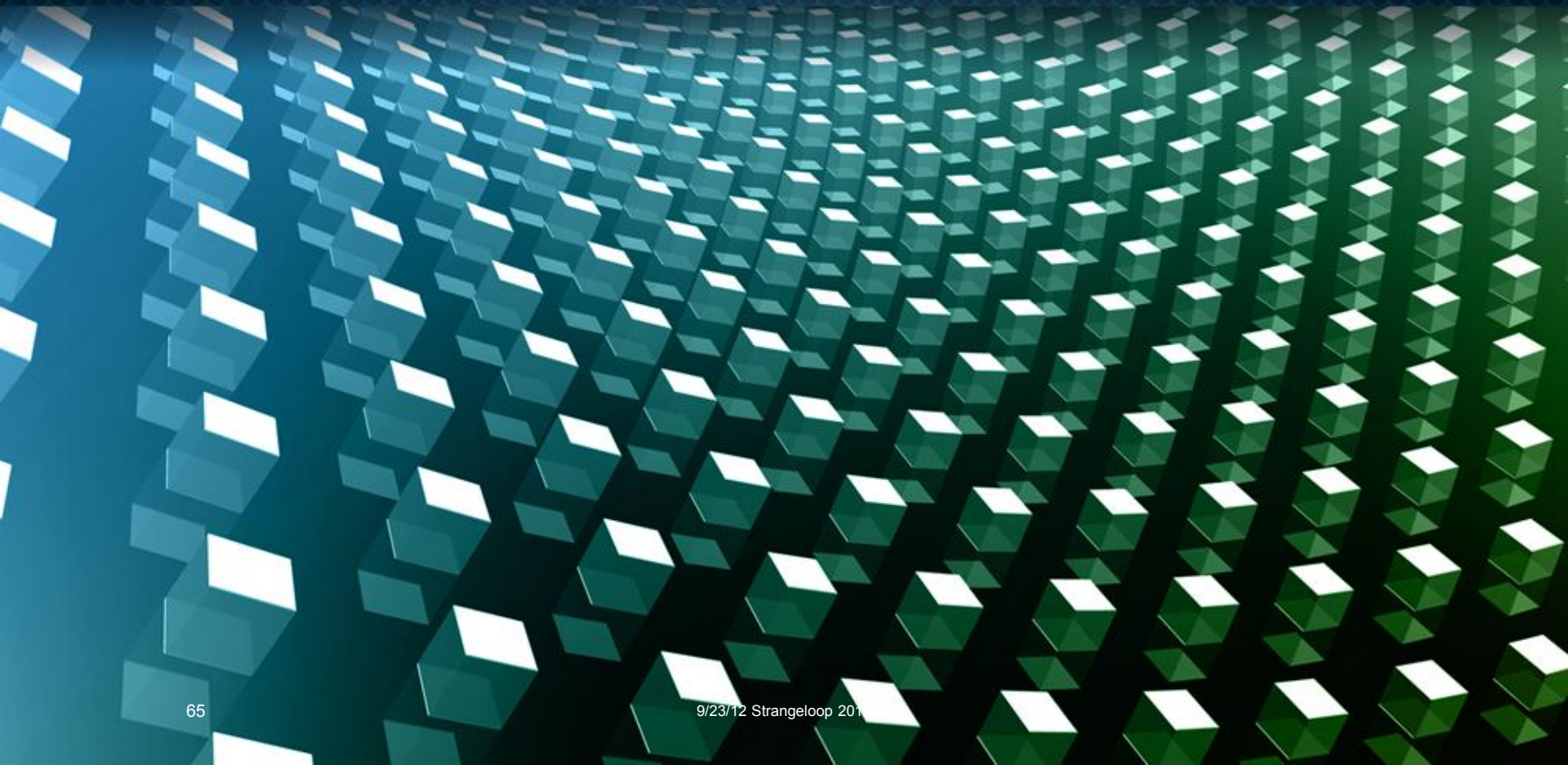
Outline

- Enter Apache HBase
- The HBase Data Model
- System Architecture
- **Real-World Applications**
- Deployment and Operations
- Conclusions

cloudera

REAL WORLD APPLICATIONS

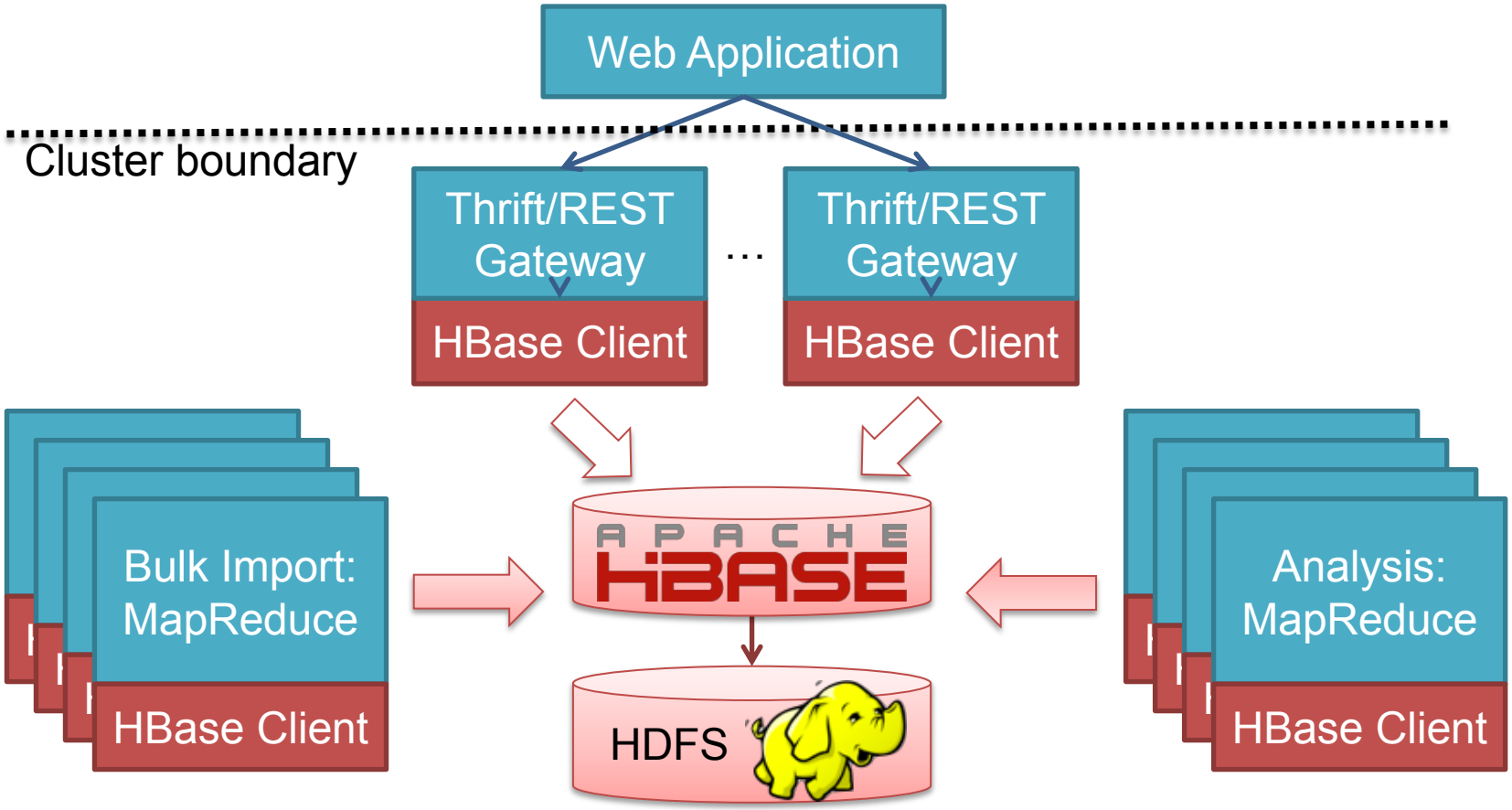
How and where is this infrastructure being used



cloudera

“YOU CAN’T SOLVE 21ST CENTURY PROBLEMS WITH 20TH CENTURY TECHNOLOGIES”

HBase Application Architecture

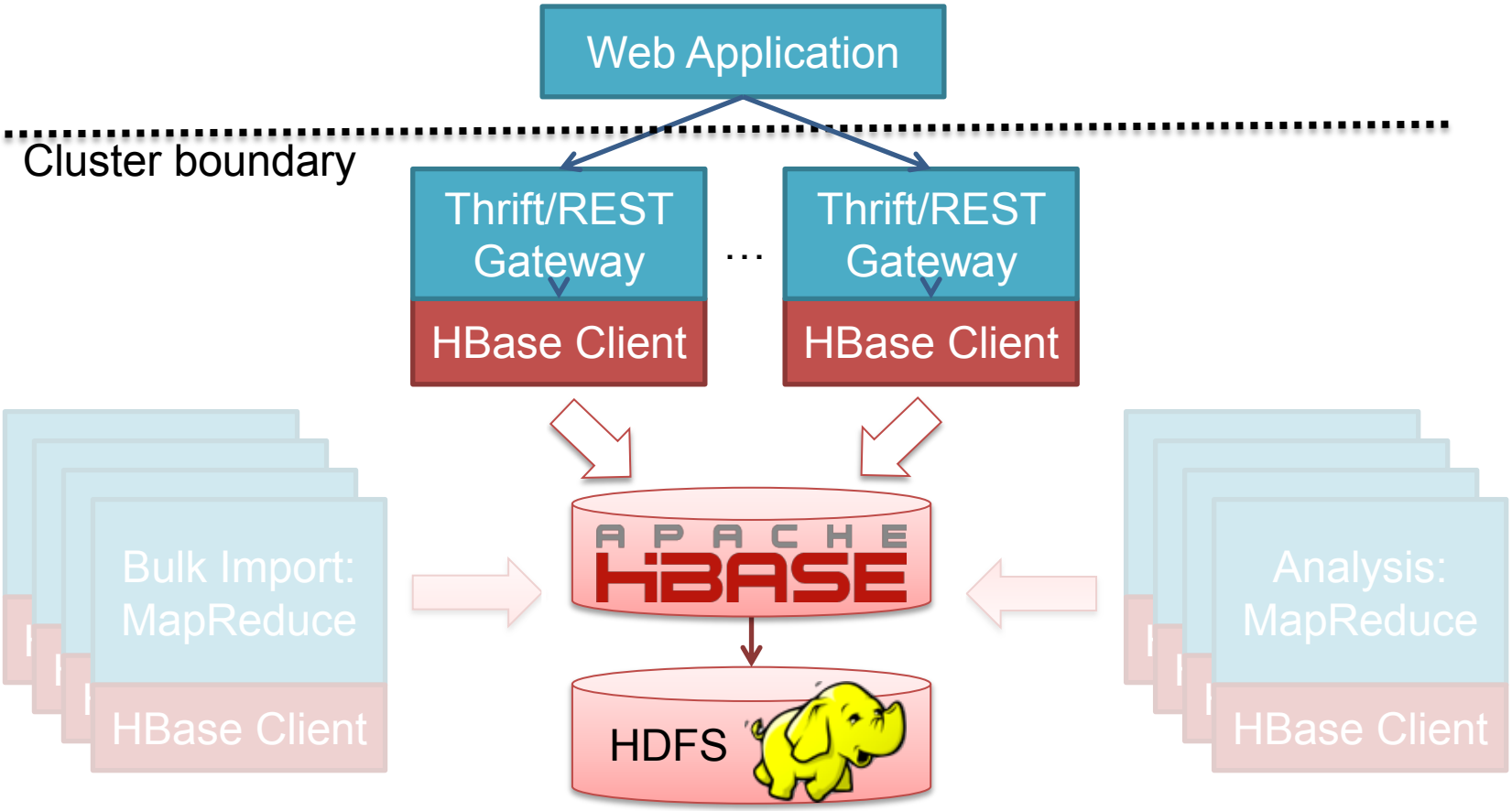


Example Apache HBase Applications

- Web Application Backends
  - Inboxes: **Facebook Messages**, Tumblr
  - Catalogs: Intuit Mint Merchant DB, Gap Inc. Clothing Database, OLCL (world library catalog)
  - URL Shortener: StumbleUpon http://su.pr,
  - Search Index: **eBay Cassini**, PhotoBucket, YapMap
- Massive datastore for Analysis
  - **Mozilla Socorro** (crash report DB), Yahoo! Web Crawl Cache
  - Mignify / Internet Memory project
- Monitoring Real-time Analytics
  - **OpenTSDB**, Sproxil, Sematext

More Info at http://www.hbasecon.com/agenda/

HBase Web Application



RDBMS: Data-centric schema design

- Entity relational model.
  - Design schema in “Normalized form”
  - Figure out your queries
  - DBA to sets primary secondary keys once query is known
- Issues:
  - Join latency and cost can be difficult to predict
  - Difficult/Expensive to change schema / add columns

HBase: Query-centric schema design

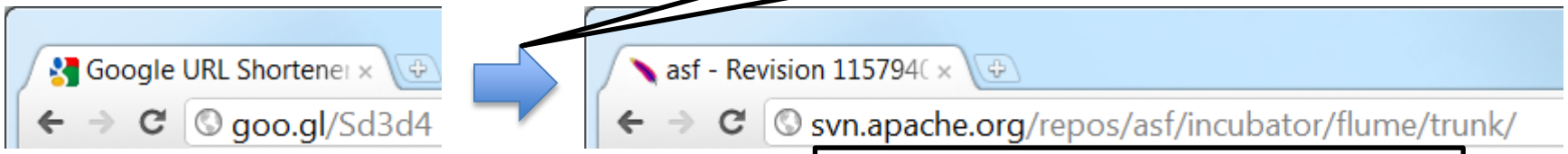
- Know your queries then design your schema
- Column-family oriented
  - Create these by knowing fields needed by queries
  - Its better to have a fewer than many
- App developers optimize the queries, not DBAs
- If you've done the relational DB query optimizations, you are mostly there already!

Schema design exercise

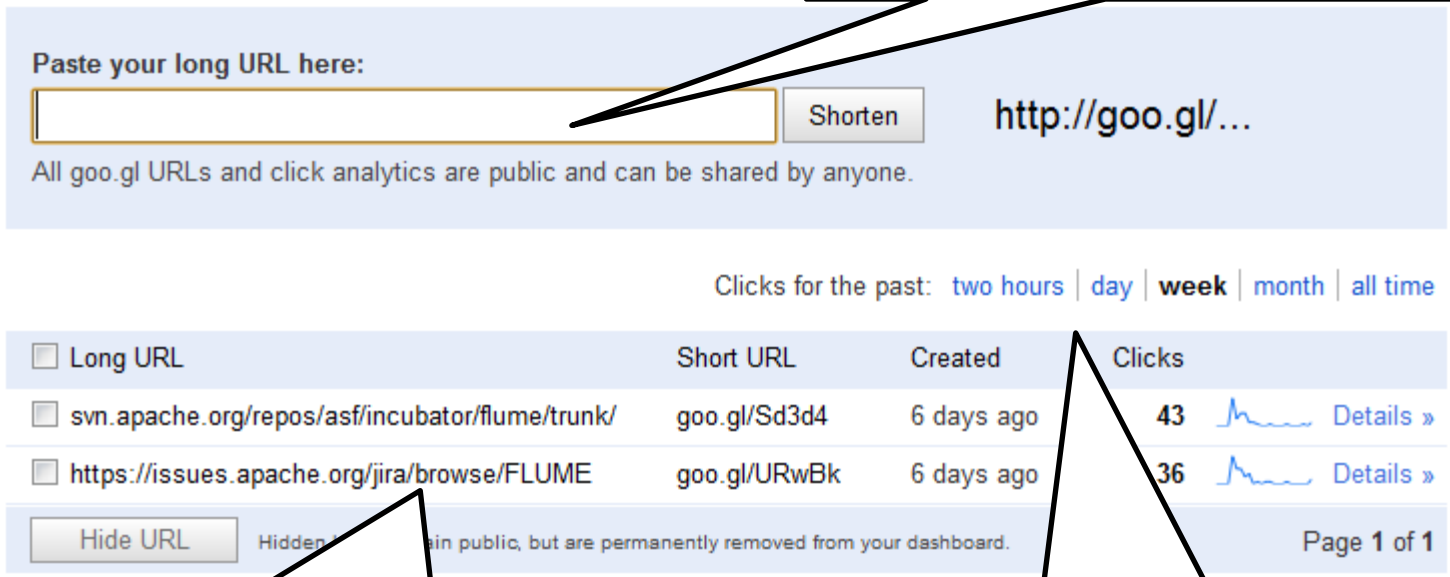
- URL Shortener
  - Bit.ly, goo.gl, su.pr etc.

Url Shortener Servic

Lookup hash, track click, and forward to full url



Enter new long url, generate store to users' mapping to short url



Look up all of a users shortened urls and display

Track historical click counts over time

Url Shortener schema

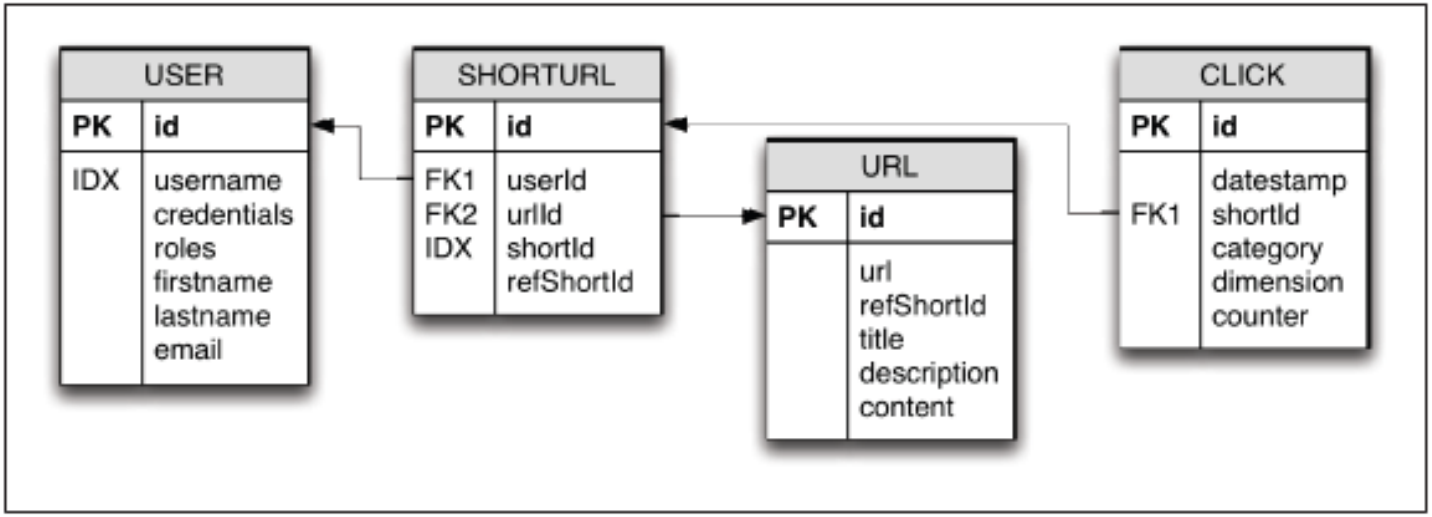


Figure 1-2. The Hush schema expressed as an ERD

- All queries have at least one join
- Constraints when adding new urls, and short urls.
- How do we delete users?

Url Shortener schema

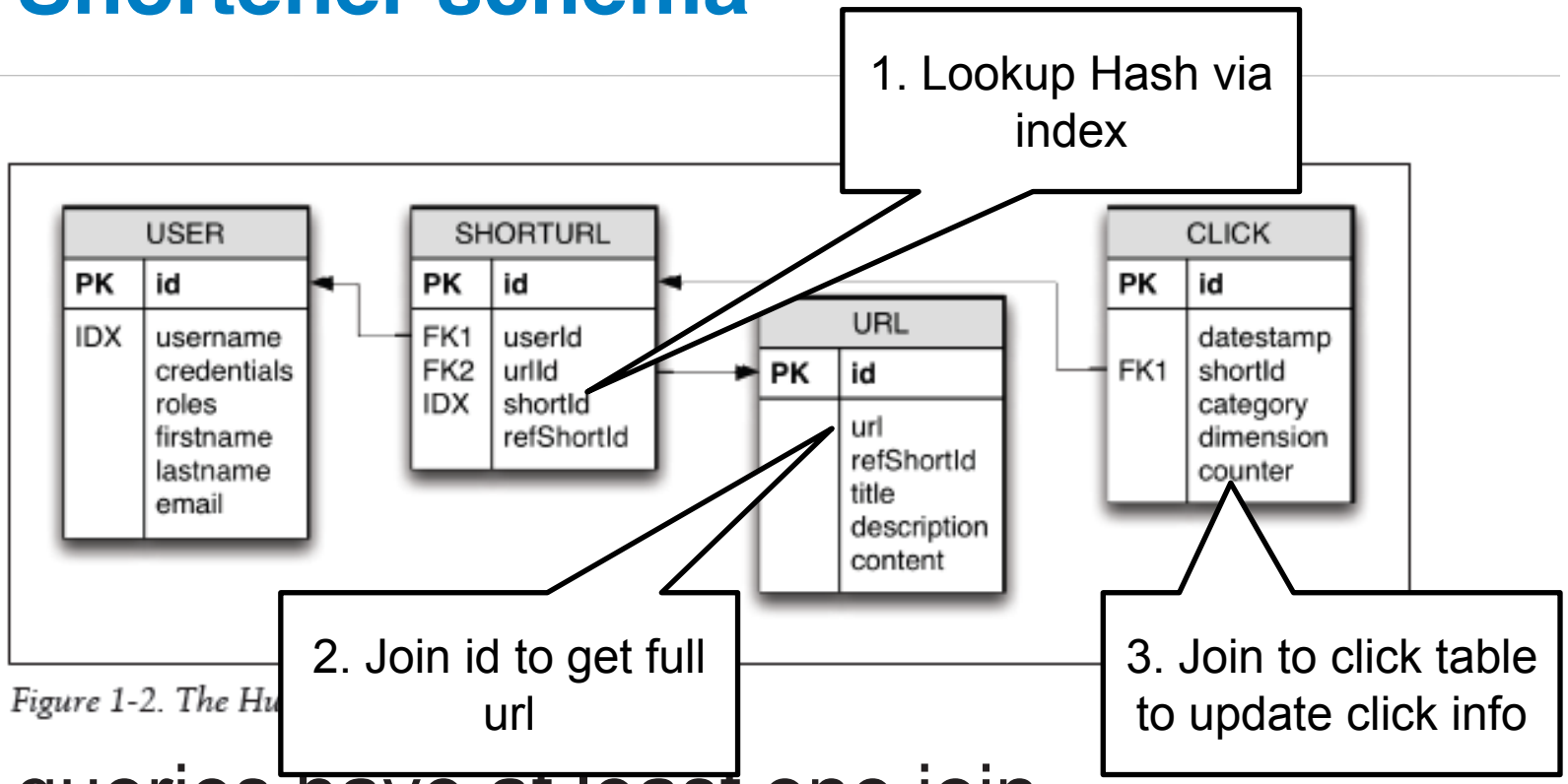


Figure 1-2. The Hu

- All queries have at least one join
- Constraints when adding new urls, and short urls.
- How do we delete users?

Url Shortener HBase schema

- All single queries
- Use c
  settin
  content column families.
- Using rowkey to group all of a user's shortened urls
- Consistency not guaranteed between tables.

1. Get url from url hash

2. Put to update click metrics

Table: shorturl		
Row Key:	shortId	
Family:	data:	Columns: [illegible], refShortId, userId, clicks
	stats-daily: **[ttl: 7days]**	Columns: YYYYMMDD, YYYYMMDD\x00<country-code>
	stats-weekly: **[ttl: 4weeks]**	Columns: YYYYWW, YYYYWW\x00<country-code>
	stats-monthly: **[ttl: 12months]**	Columns: YYYYMN, YYYYMM\x00<country-code>

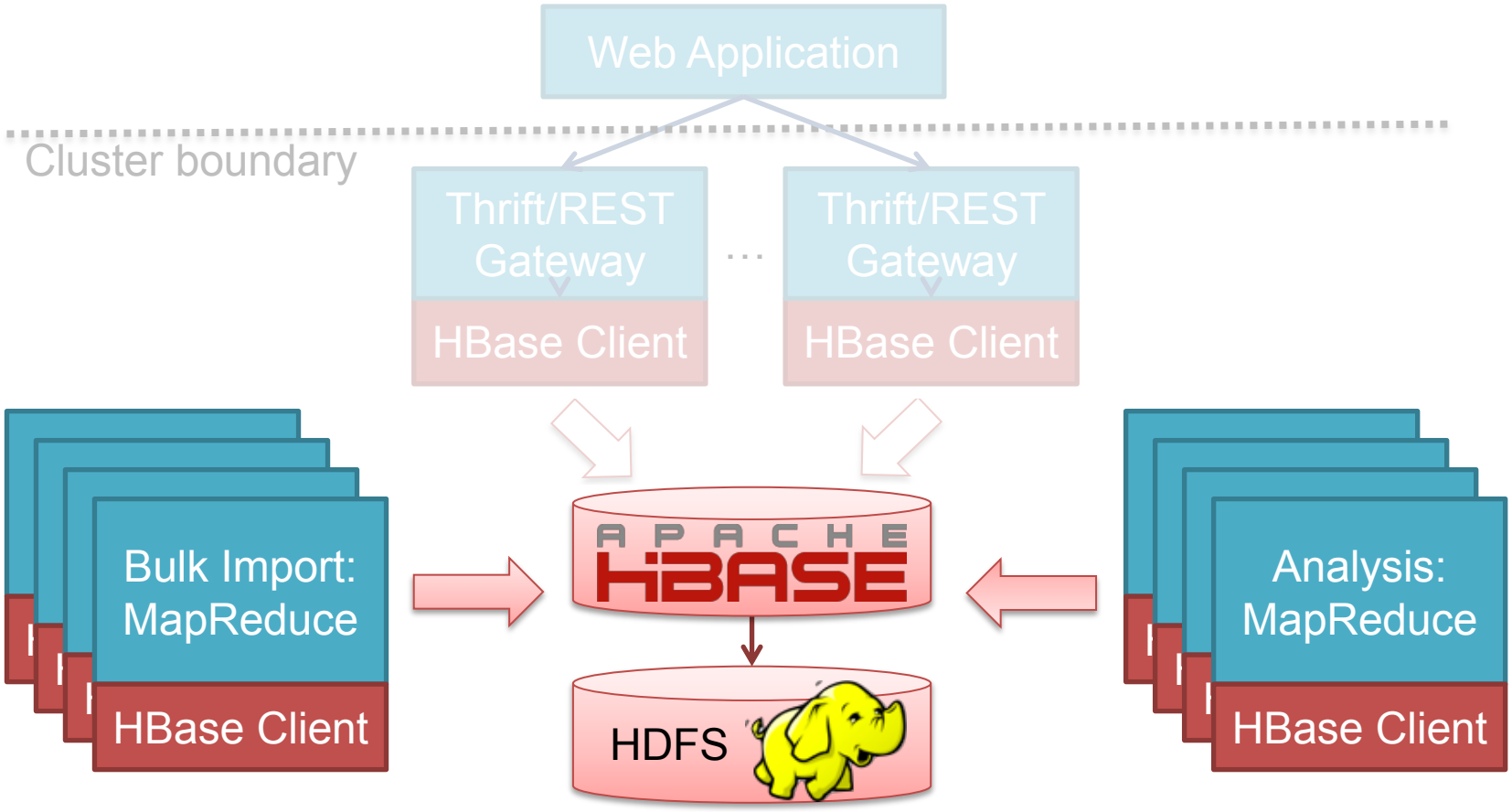
Table: url		
Row Key:	MD5(url)	
Family:	data: **[compressed]**	Columns: refShortId, title, description
	content: **[compressed]**	Columns: raw

Table: user-shorturl		
Row Key:	username\x00shortId	
Family:	data:	Columns: timestamp

Table: user		
Row Key:	username	
Family:	data:	Columns: credentials, roles, firstname, lastname, email

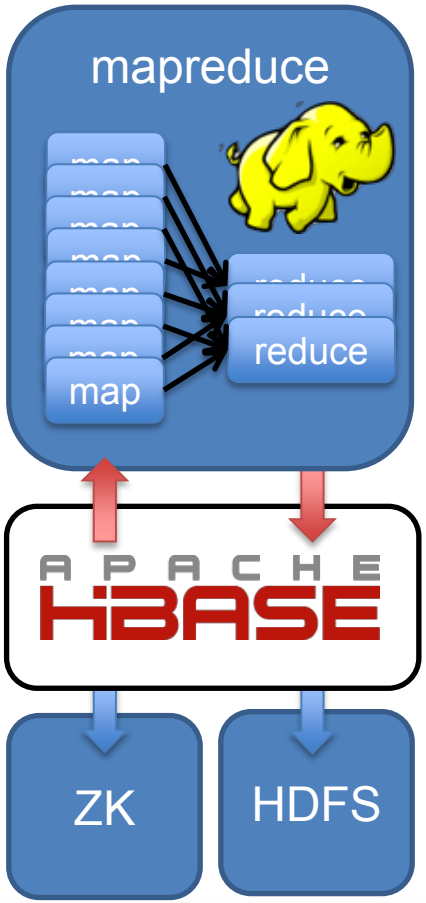
Figure 1-3. The Hush schema in HBase

HBase Analysis Application Architecture



Hadoop MapReduce and HBase

- Use to scalably **process** data into HBase
- Create new data sets from raw data.
  - ETL into DBs/HBase
- High throughput batch processing
  - You would not serve live traffic from MR query or directly from HDFS.
- Users just write a “map” function and a “reduce” function.



MapReduce Processes

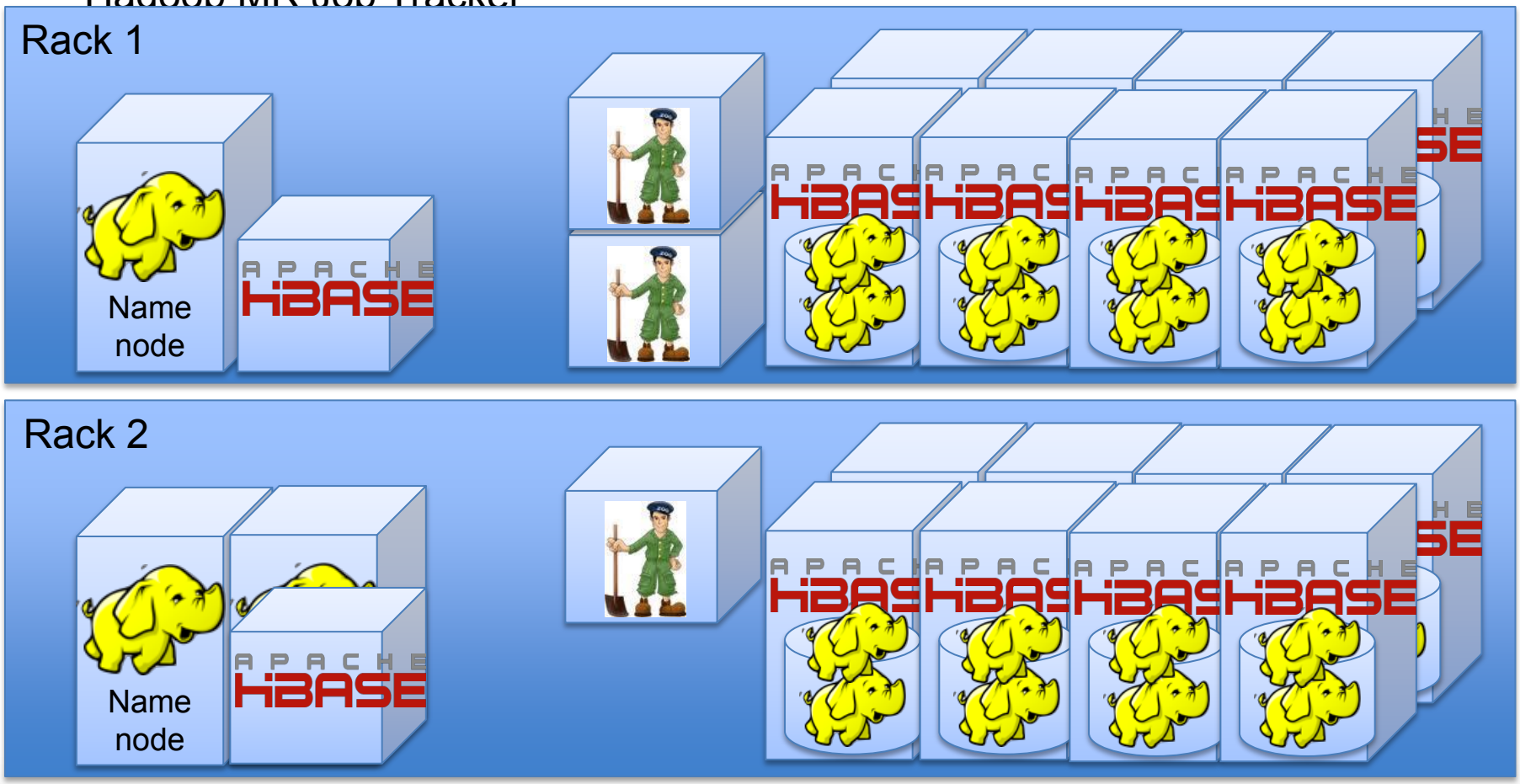
- Job Tracker
  - Schedules work and resource usage through out the cluster
  - Makes sure work gets done
    - Controls retry, speculative execution, etc.
- Task Trackers
  - These slaves do the “map” and “reduce” work
  - Co-located with data nodes

HBase + HDFS + MR Nodes (physically)

HDFS NameNodes
HBase Masters
Hadoop MR Job Tracker

ZooKeeper Quorum

Slave Boxes (DN + RS + TT)



Data Loading Patterns

- Random Writes
  - Uses Put API
  - Simple
  - Low latency
  - Less throughput (more HBase overhead)
- Use Case:
  - Real-time Web apps
  - Real-time serving of data
- Example:
  - Inbox, url shortener

- Bulk import
  - Use MapReduce to generate HBase native files, atomically add metadata.
  - High Latency
  - High Throughput
- Use Case:
  - Large scale analysis
  - Generating derived data
  - ETL
- Examples:
  - Delayed Secondary indexes
  - Building search index
  - Exploring HBase Schemas

Schema Design Exploration

- Applications optimized by designing the structure of data in HBase
  - MR, HDFS and HBase complement each other.
- Exploration Steps:
  - Save raw data to HDFS/HBase.
  - MR for data transformation and ETL-like jobs from raw data.
  - Use bulk import from MR to HBase.
  - Serve data from HBase

HBase vs just HDFS

	Plain HDFS/MR	HBase
Write pattern	**Append-only**	**Random write, bulk incremental**
Read pattern	**Full table scan, partition table scan**	**Random read, small range scan, or table scan**
Structured storage	**Do-it-yourself / TSV / SequenceFile / Avro / ?**	**Sparse column-family data model**
Max data size	**30+ PB**	**~1PB**

If you have neither random write nor random read, stick to HDFS!

Rules for Applications

- HBase isn't really worth the effort until you have 5-10 machines.
- No Performance Isolation guarantees in HBase (yet)
  - Start conservative
  - Isolate applications and workloads by having more clusters.
    - If scale warrants, separate real-time applications into separate HBases.
    - Separate your Batch MR workloads from your realtime workloads if possible.

Examples

- Facebook
  - Many clusters, one application (Facebook Message)
  - Pods of 100-200 HBase Region servers
  - Easier for maintenance, and standardized hardware deploy in many data centers.

Outline

- Enter Apache HBase
- The HBase Data Model
- System Architecture
- Real-World Applications
- **Deployment and Operations**
- Conclusions

cloudera

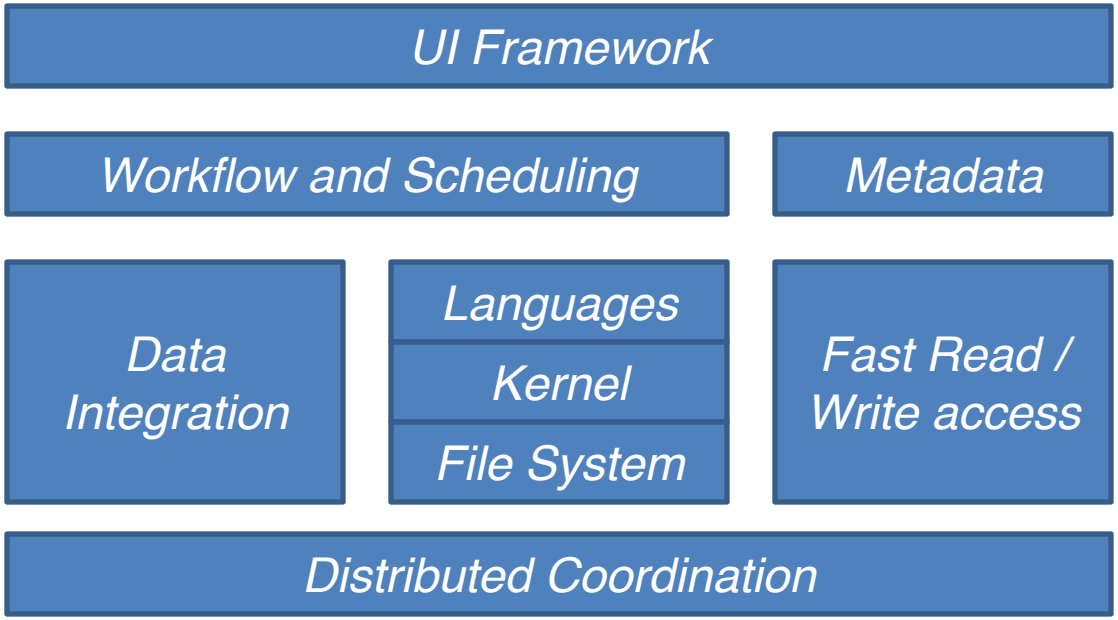
Monitoring, Benchmarking, Tuning, and Troubleshooting

DEPLOYMENT AND OPERATIONS

HBase Operations

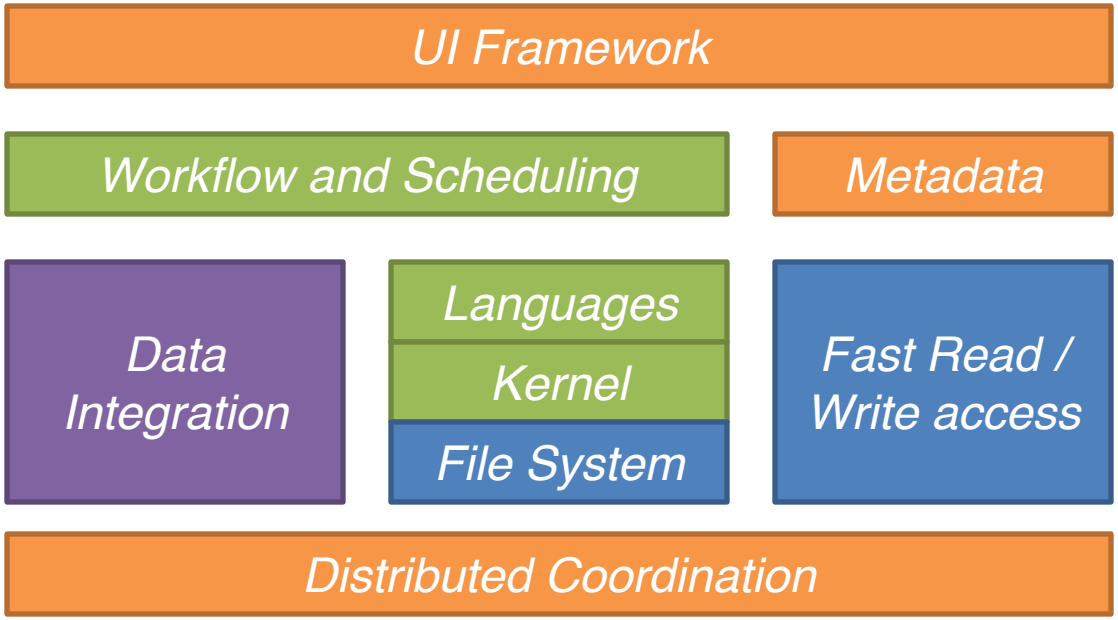
- Choosing a distribution
- Monitoring
- Performance and tuning
- Troubleshooting
- Backups and Disaster recovery

The Hadoop Big Data Stack



- Hadoop is the core of the Stack.
  - The kernel of a cluster operating system
- Each “friend” is a distributed service that has a similar workstation tool.

CDH4's Hadoop stack



- Storage : HDFS, HBASE
- Processing : MR, MR2, Pig, Hive, Mahout, Oozie
- Data Integration: Flume, Sqoop
- Coordination : ZooKeeper, Avro, Bigtop, (Hive), Hue

Distribution Options

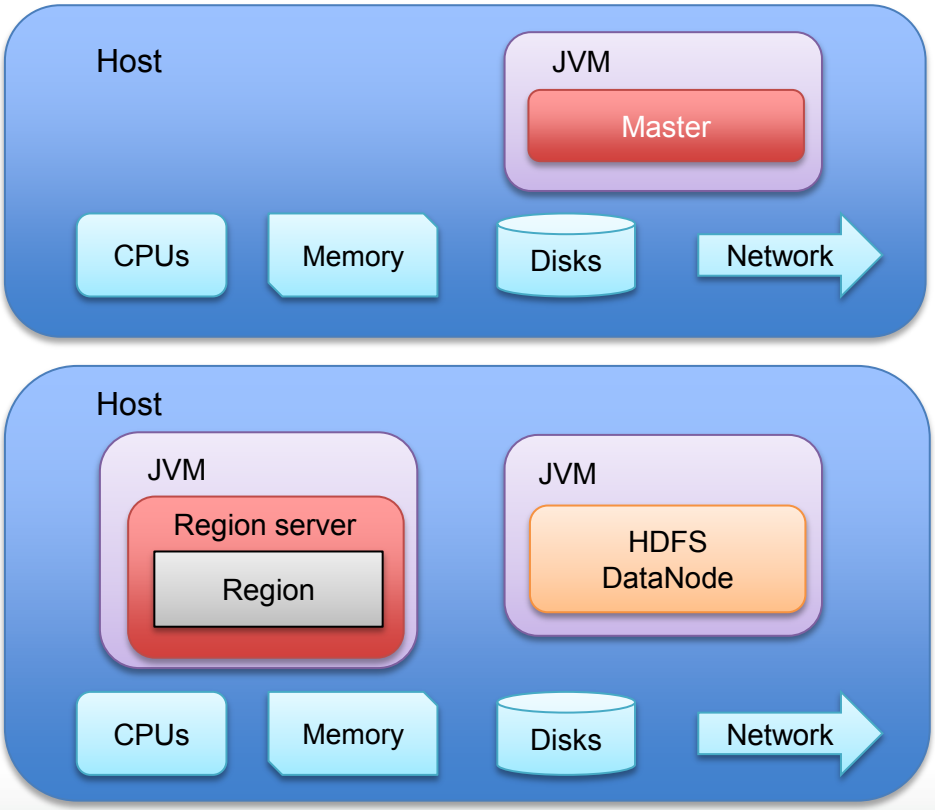
- Use Tarballs
  - Build your own deployment or use puppet/chef
  - Apache Whirr for EC2
- Use Packages (RPMs/Debs)
  - Apache Bigtop (Hadoop stack integration and packaging)
- Use Vendor Packages
  - Cloudera's Distribution including Apache Hadoop (CDH)
  - Cloudera Manager Free Edition
  - HortonWorks Data Platform

Monitoring + Metrics

- "If you cannot measure it, you cannot improve it"

- System Metrics
  - SAR
  - JVM GC
- HBase Metrics
  - Get / put latencies
  - Operations / sec
  - Hbase Health – hbck
  - HBase's Web UI

- Integrate with existing tools
  - Ganglia / Nagios
  - JMX / JConsole
  - Cloudera Manager



Ganglia .sourceforge.net

bolt Cluster Report for Sat, 22 Sep 2012 06:48:34 -0700

Get Fresh Data

Metric load_one | Last day | Sorted descending | Physical View

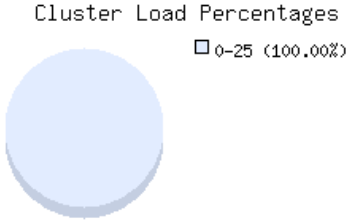
Grid > bolt > --Choose a Node

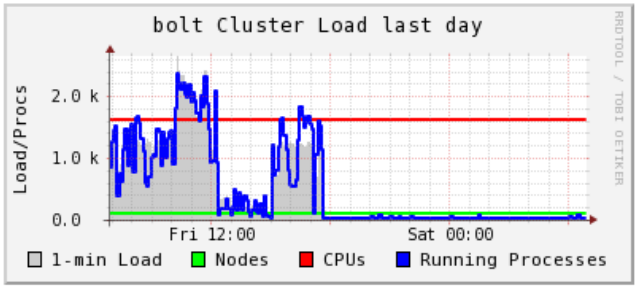
Overview of bolt

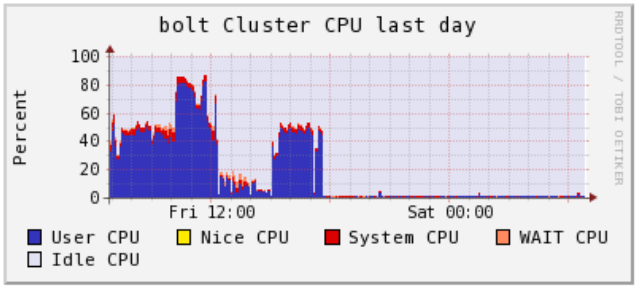
CPUs Total: **1616**
Hosts up: **102**
Hosts down: **0**

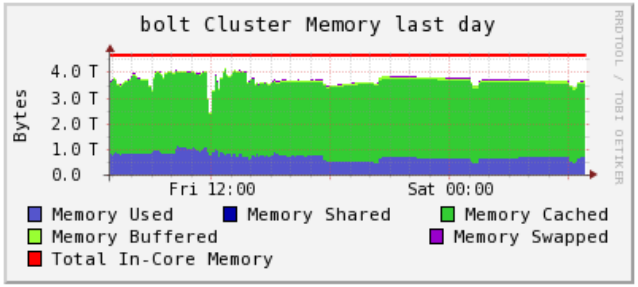
Avg Load (15, 5, 1m):
1%, 1%, 1%
Localtime:
2012-09-22 06:48

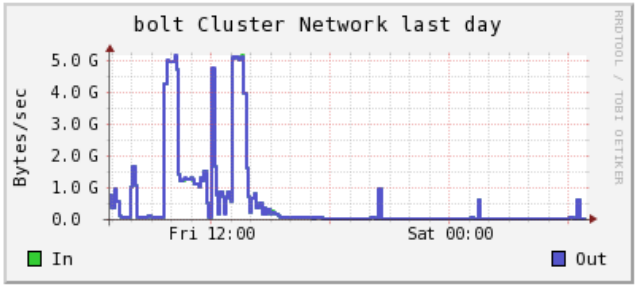
Cluster Load Percentages



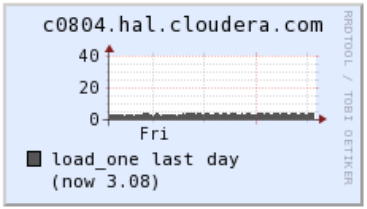


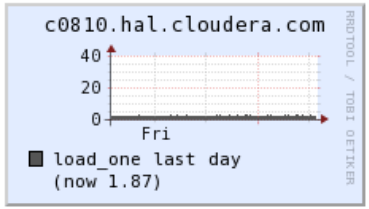


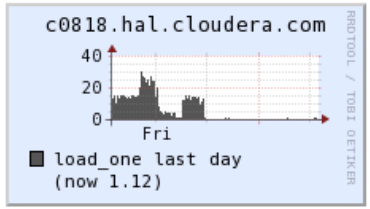


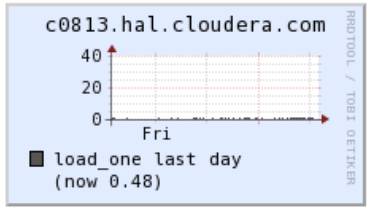


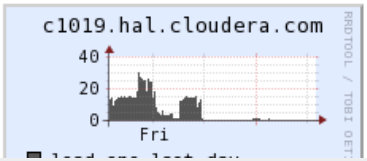
Show Hosts: yes ◉ no ○ | bolt **load_one** last **day** sorted **descending** | Columns 4 | Size small

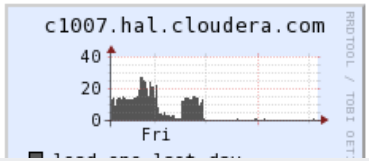


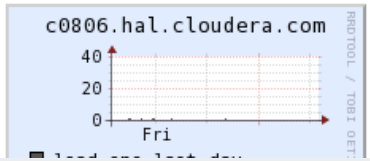


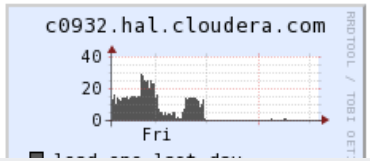












Ganglia .sourceforge.net

bfd-a Cluster Report for Sat, 22 Sep 2012 10:08:47 -0700

Get Fresh Data

Metric load_one ▾ Last month ▾ Sorted descending ▾ Physical View

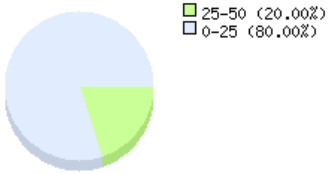
Grid > bfd-a > --Choose a Node ▾

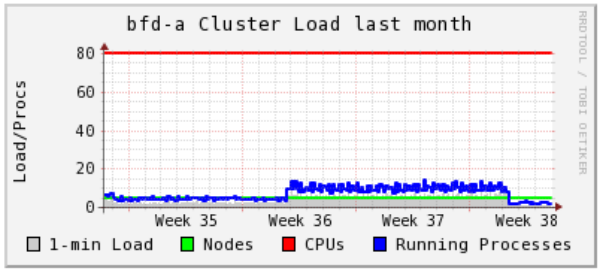
Overview of bfd-a

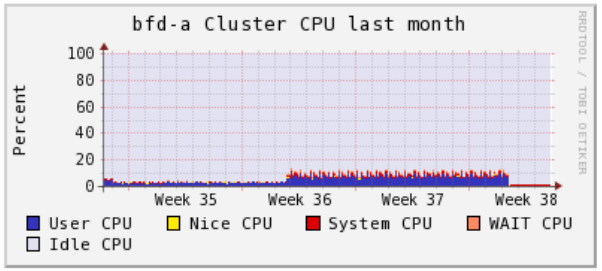
CPUs Total: **80**
Hosts up: **5**
Hosts down: **0**

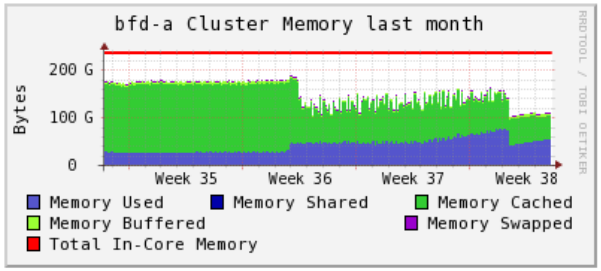
Avg Load (15, 5, 1m):
10%, 11%, 6%
Localtime:
2012-09-22 10:08

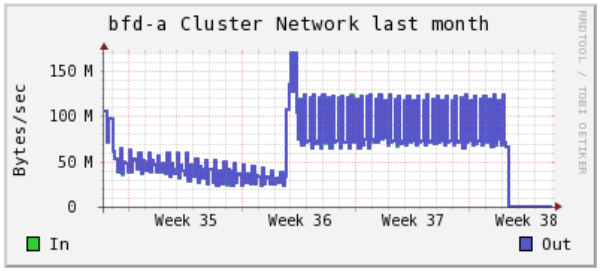
Cluster Load Percentages

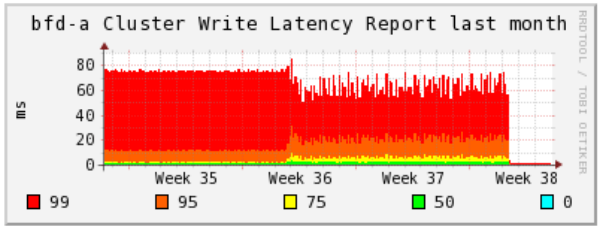


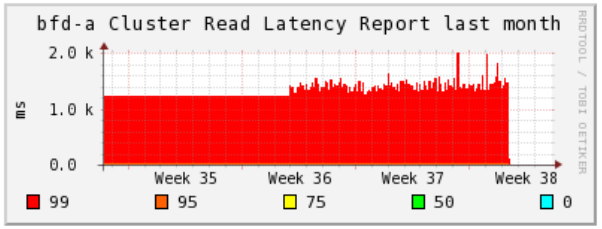




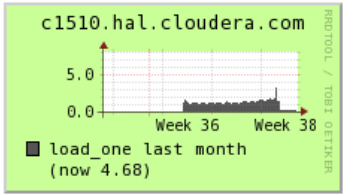


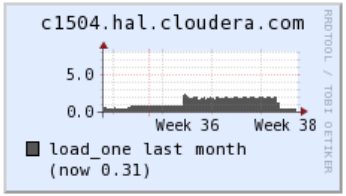


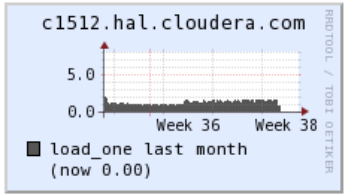


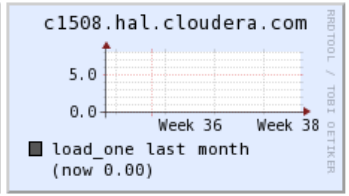


Show Hosts: yes ◉ no ○ | bfd-a **load_one** last **month** sorted **descending** | Columns 4 ▾ Size small ▾









HBase Performance Benchmarking

- Kinds of workloads:
  - Writes / Reads / Mixed workloads
  - Sequential vs Random
  - Query Distribution (Caching)
  - MR workload
- MR based benchmarks can be biased by MR overhead.

Performance benchmarks suites

- HBase:Performance Evaluation
  - Micro benchmarks (but uses MR)
  - Test some basic hbase workloads (random/sequential reads/writes)

```
hbase org.apache.hadoop.hbase.PerformanceEvaluation
```

- Yahoo!'s Cloud Serving Benchmark (YCSB)
  - http://research.yahoo.com/Web_Information_Management/YCSB
  - One harness many workloads, many Datastores.
  - Different query and write distributions: Uniform and Zipf
  - Records latencies and throughput.
  - Make sure to get updated versions that presplit HBase tables!

```
java -classpath `hbase classpath` com.yahoo.ycsb.Client  -db
   com.yahoo.ycsb.db.HBaseClient
```

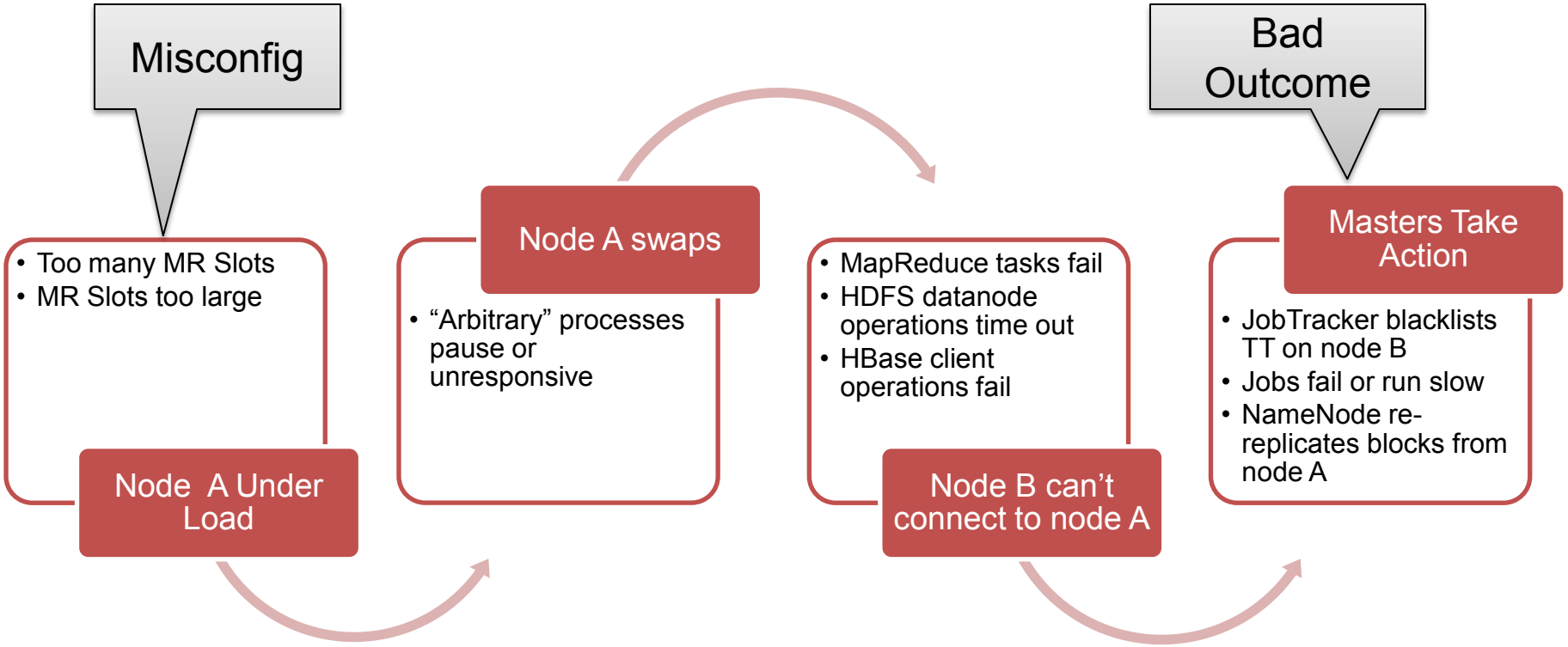
- HDFS: DFSIO
  - An aggregate HDFS IO throughput test. (uses MR)

```
hadoop jar $HADOOP_HOME/hadoop-*-test.jar TestDFSIO
```

Configuring HBase

- HBase depends on other systems so you need to understand enough of each.
- hdfs-site.xml, hbase-site.xml, zoo.cfg, hbase-env.sh
- System settings
  - Don’t Over provisioning host memory
    - 1GB / DN, 1-2GB/MR Task, 8-16 GB/RS
  - Bump up ulimit to 32k+
  - Bump up ZK max connections to 100+

Troubleshooting: Memory Over-subscription

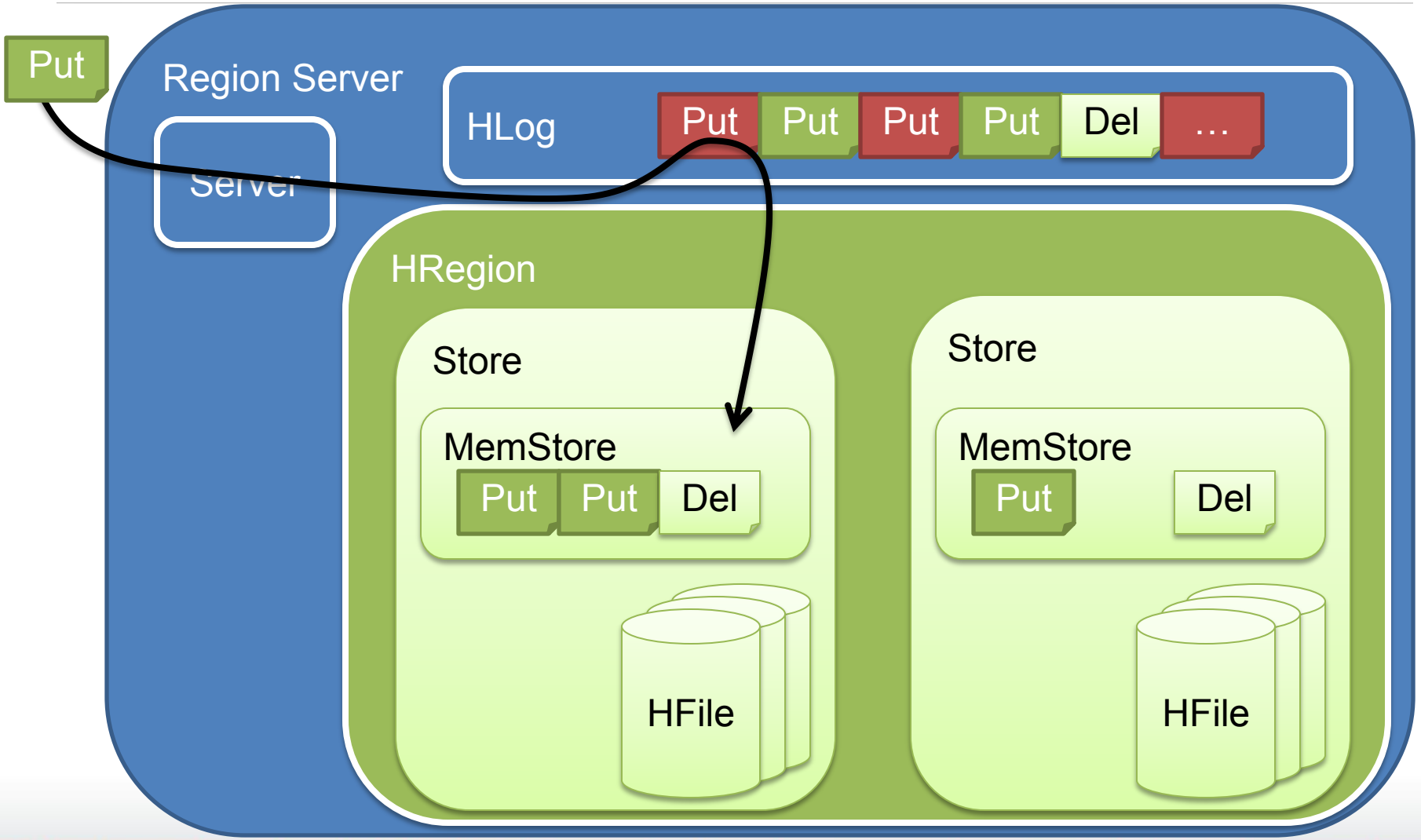


Isolate the problem, pull the thread, fix the problem.

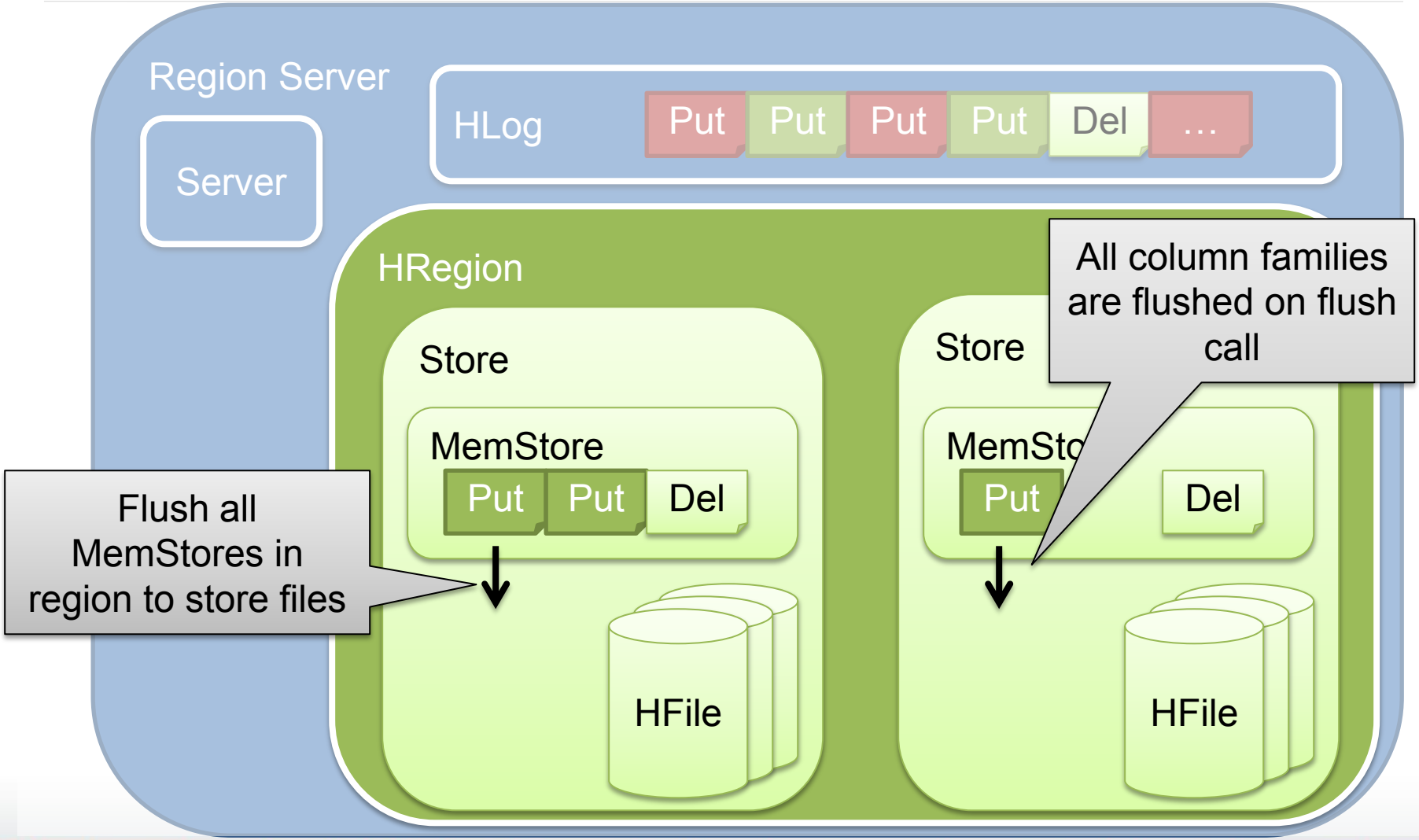
Tuning HBase for writes

- Scaling writes requires a bit of architecture understanding
  - Puts
  - Flushes
  - Compactions
  - Splits (sharding, triggers compaction)

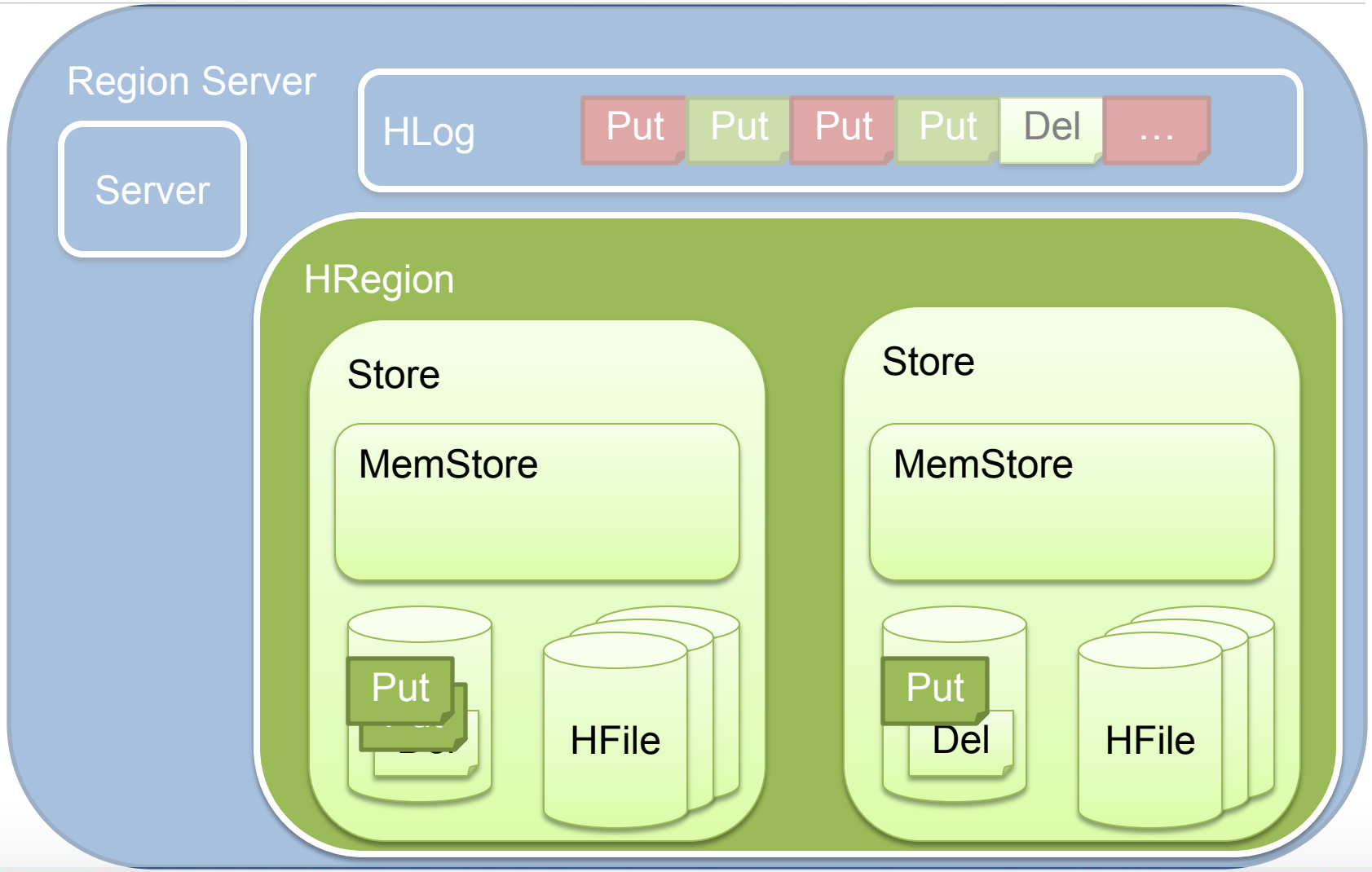
HBase Write Path



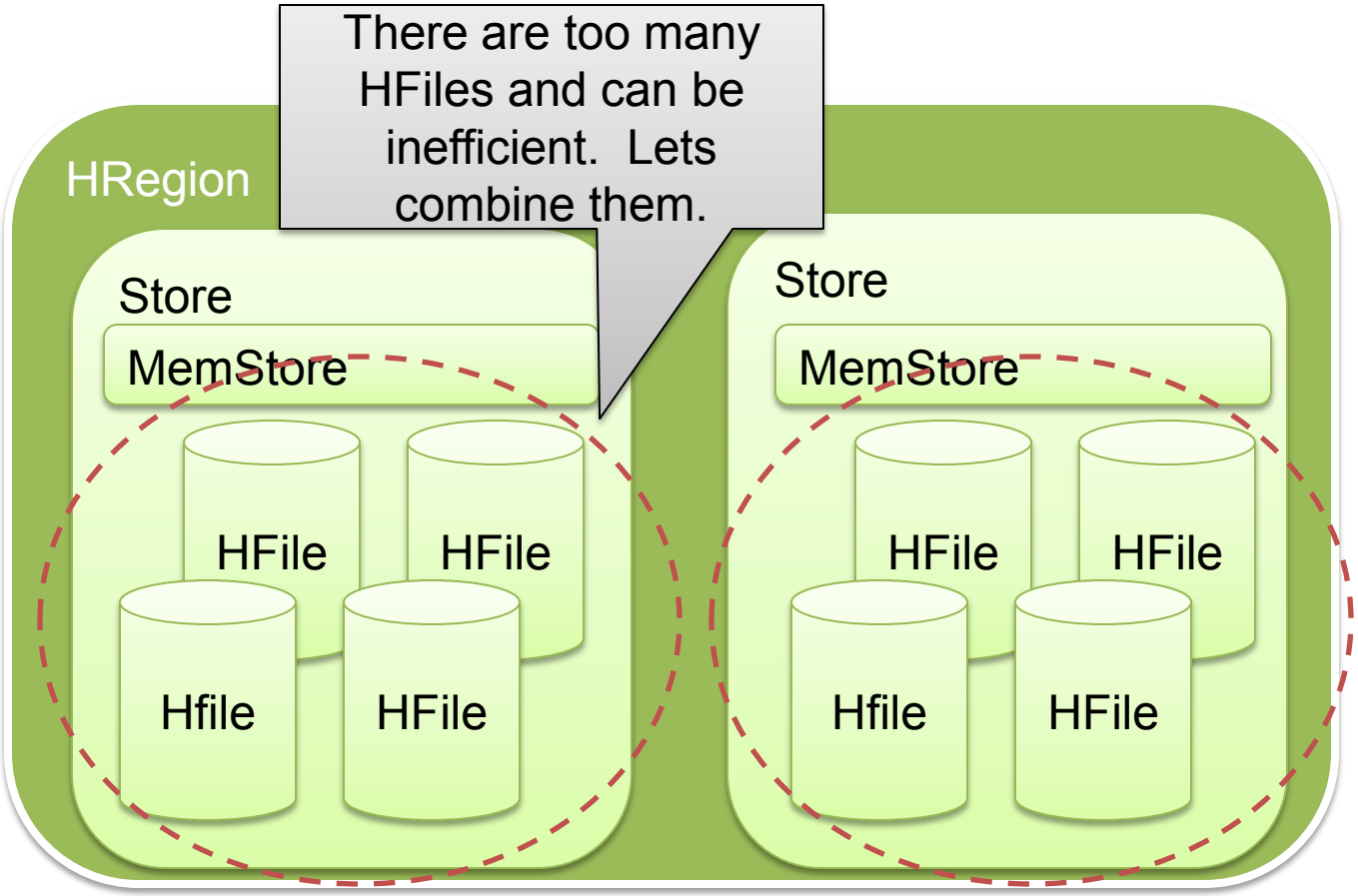
HBase Write Path: Flush



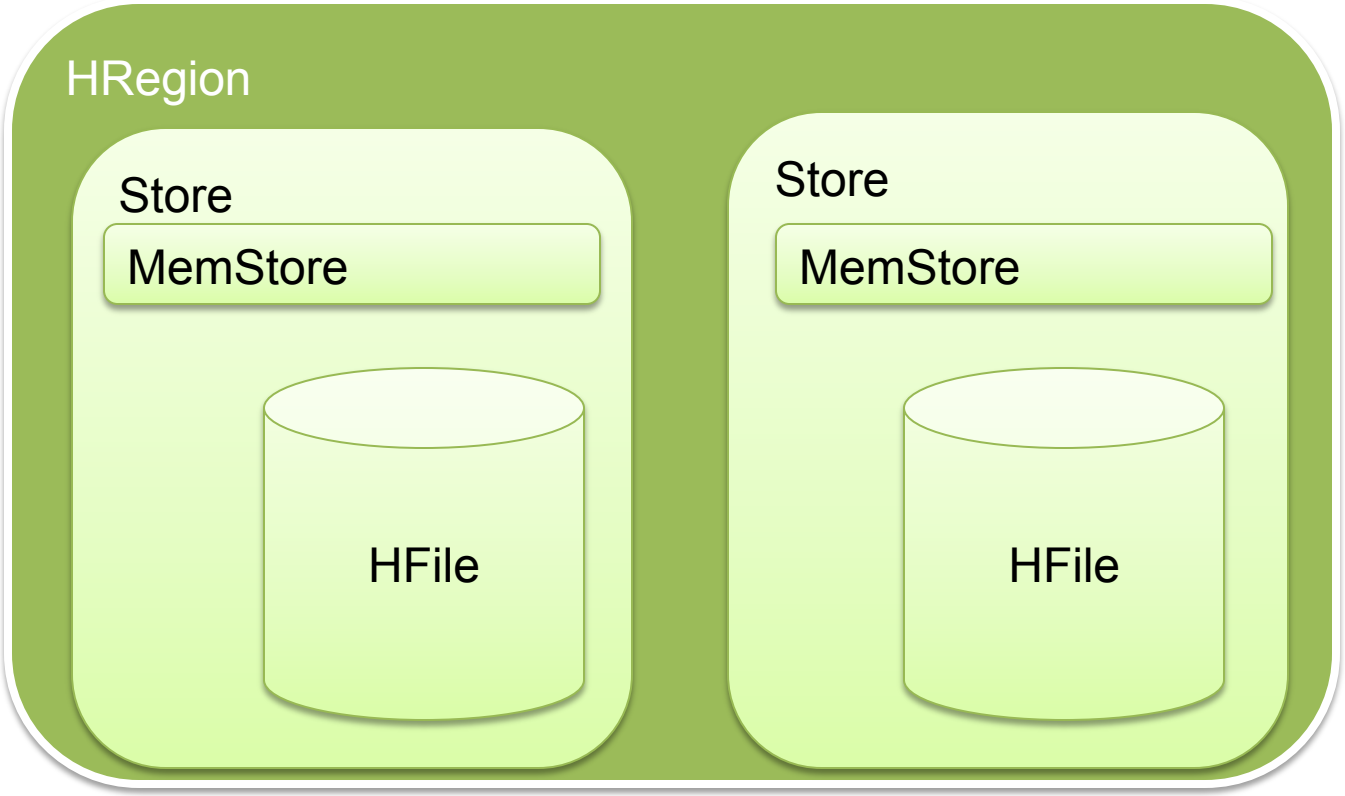
HBase Write Path: Flush



HBase Compaction



HBase Compaction



HBase Regions: operational advice

- Pre-split your regions
  - Creating a table by default creates one region that splits.
  - If you know your key distribution presplit so that writes and reads can be load balanced.
- Compaction Storms
  - GC during compaction == long pause / unavailbility
  - Schedule Major compactions at off times
  - Schedule splits at off times (Triggers major compact)
  - Tune the size of memstores / max files before majore compact
- JVM GC tuning
  - Overly large HBase heaps
    - Stop-the-world GC's @ 8-10s/GB
  - MSLAB mitigation (Avoid fragmentation old gen)
  - Debugging GC:
    - `-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps`
- Favor more machines over beefier machines (More IO)

Outline

- Enter Apache HBase
- The HBase Data Model
- System Architecture
- Real-World Applications
- Deployment and Operations
- **Conclusions**

cloudera

CONCLUSIONS

Key takeaways

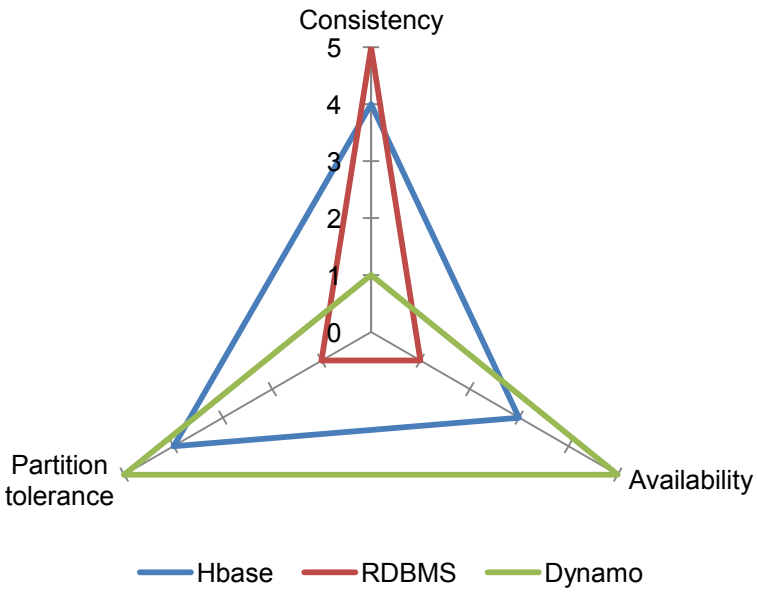
- Apache HBase is not a RDBMS! There are other scalable databases.
- Query-centric schema design, not data-centric schema design.
- In production at 100's of TB scale at several large enterprises.
- If you are restructuring your SQL DB to optimize it, you may be a candidate for HBase.
- HBase complements and depends upon Hadoop.
- New features focused for enterprise needs.

HBase vs RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only*
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of “queries”/second

HBase vs other “NoSQL”

- Favors **Strong Consistency** over **Availability** (but availability is good in practice!)
- **Great Hadoop integration** (very efficient bulk loads, MapReduce analysis)
- **Ordered range partitions** (not hash)
- **Automatically shards/scales** (just turn on more servers, really proven at petabyte scale)
- **Sparse column storage** (not key-value)



HBase vs just HDFS

	Plain HDFS/MR	HBase
Write pattern	**Append-only**	**Random write, bulk incremental**
Read pattern	**Full table scan, partition table scan**	**Random read, small range scan, or table scan**
Structured storage	**Do-it-yourself / TSV / SequenceFile / Avro / ?**	**Sparse column-family data model**
Max data size	**30+ PB**	**~1PB**

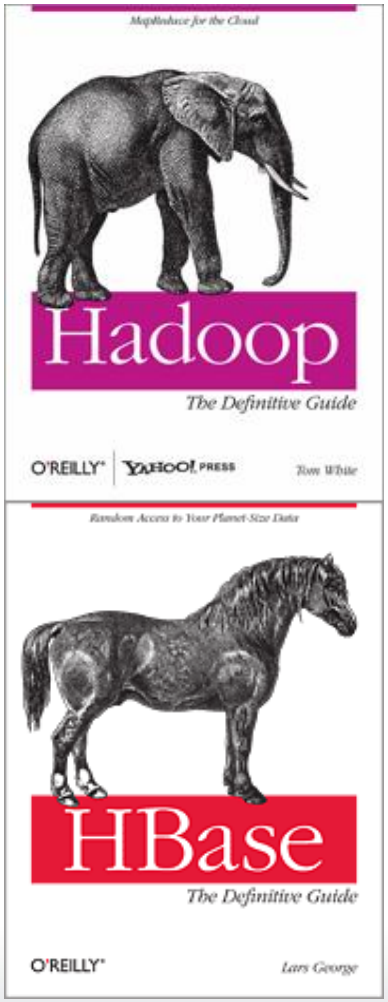
If you have neither random write nor random read, stick to HDFS!

Deployment and Operations

- HBase interacts with many systems.
  - This is powerful –
    - Each component excels at a particular function
    - Live and in production at top 100 websites
  - This can be challenging–
    - Many knobs to tune
  - Isolated instances to make managing easier
- Monitor monitor monitor

More resources?

- Download Hadoop and HBase!
  - CDH - Cloudera's Distribution including Apache Hadoop http://cloudera.com/
  - http://hadoop.apache.org/
- Try it out! (Locally, VM, or EC2)

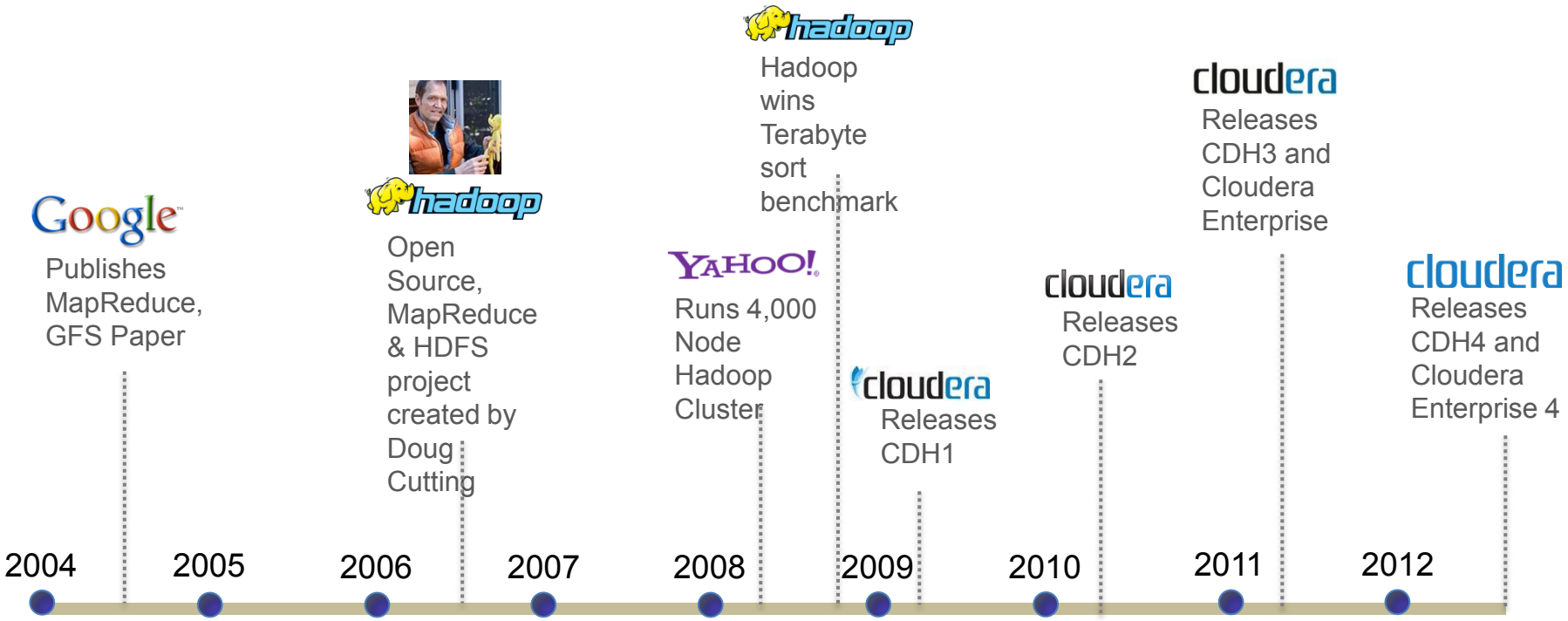


cloudera

jon@cloudera.com
@jmhsieh

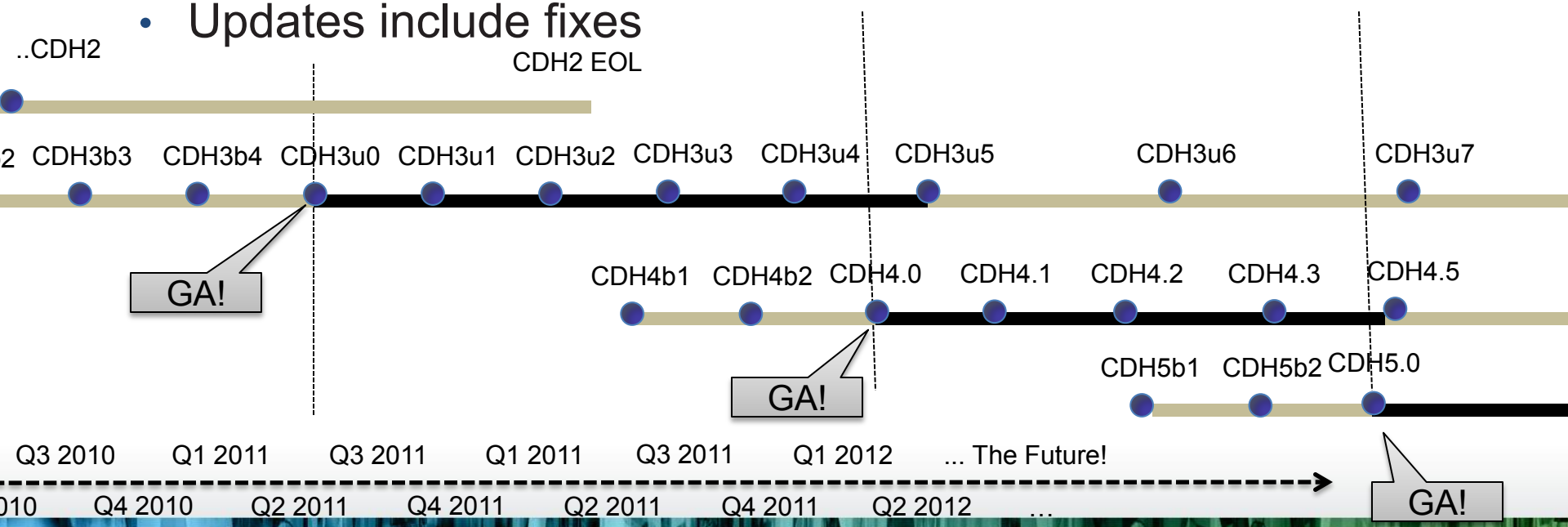
QUESTIONS?

Origin of Apache Hadoop and CDH



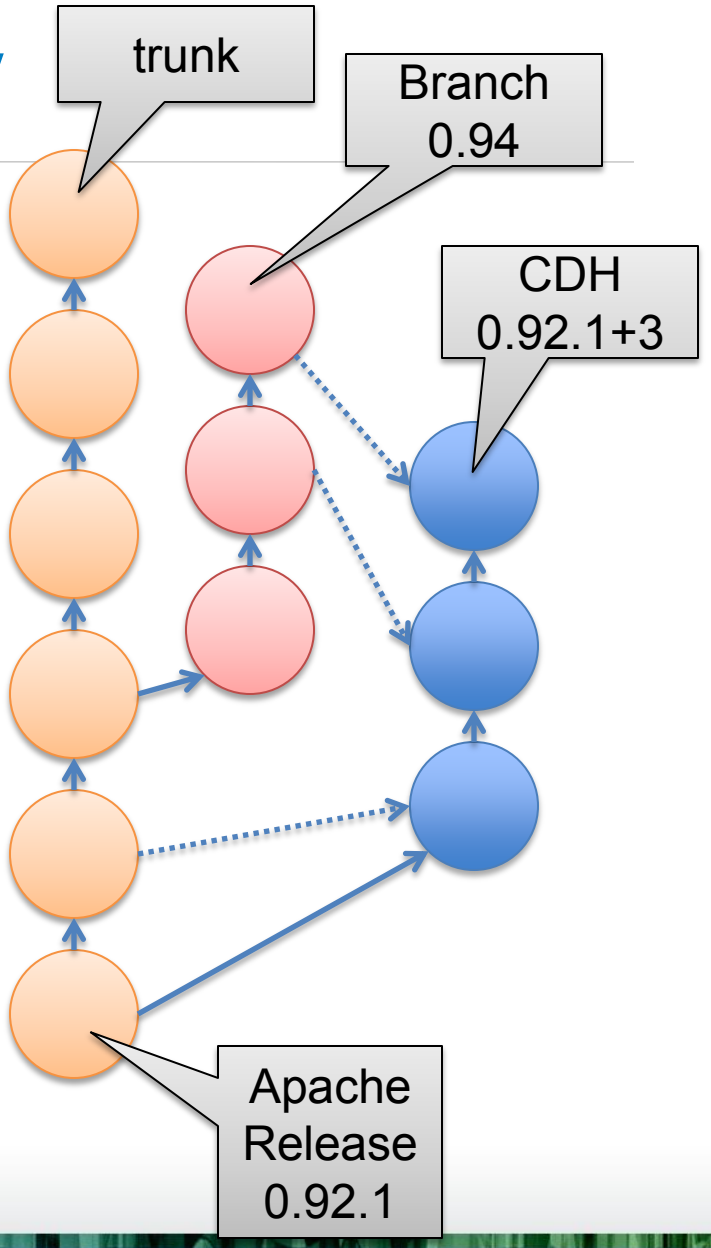
Predictable Release Schedule

- Regular release and updates
  - Skip updates without penalty
- Compatibility policy
  - Only major releases break compatibility
  - Updates can include new features
  - Updates include fixes



Open source methodology

- What's different from Apache?
- All components have code committed **upstream** first then backports code to CDH on top of a "pristine apache release"
- All patches available in tarballs




cloudera
CHANGING
THE WORLD
ONE PETABYTE
AT A TIME

Install HBase on your laptop

- Download and untar

```
wget
  http://apache.osuosl.org/hbase/hbase-
  0.92.2/hbase-0.92.2.tar.gz
tar xvfz hbase-0.92.2.tar.gz
cd hbase-0.92.2
bin/start-hbase.sh
```

- Verify:

```
bin/hbase shell
```

Browse `http://localhost:60010`