cloudera

INTRODUCTION TO HBASE

9/24/12 STL HUG 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

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

William Gibson


Google!
BETA
Search the web using Google
Google Search
I'm feeling lucky
More Google!
Copyright ©1999 Google Inc.

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, …

cloudera

ENTER APACHE HBASE

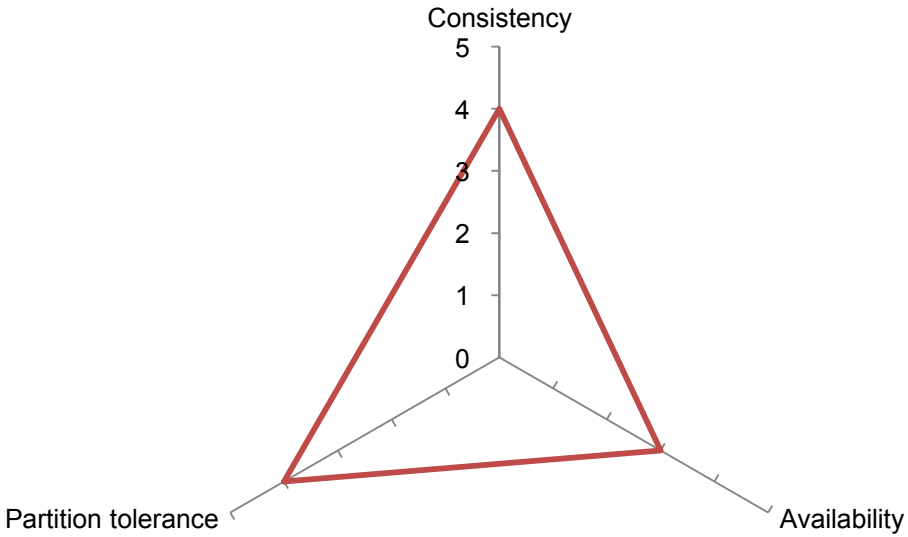
Low-latency, consistent, random read/write big data access

What is Apache HBase?

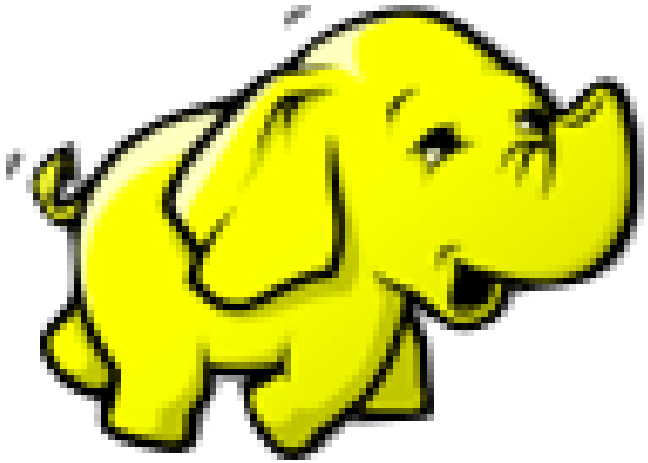
Apache HBase is an **open source**, **horizontally scalable**, **consistent, low latency, random access** big-data store built on top of **Apache Hadoop**

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.



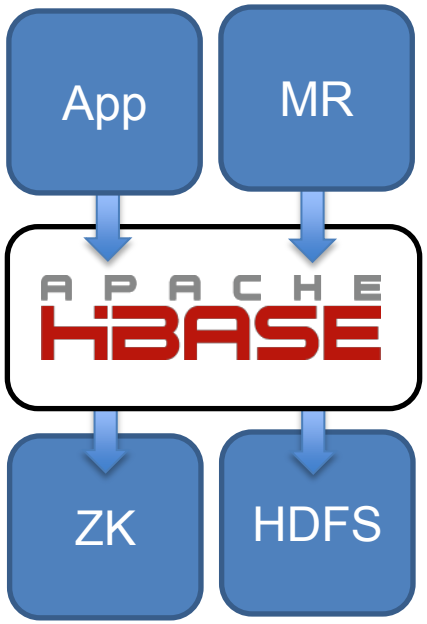
HBase depends on Apache Hadoop



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

HBase Depedencies

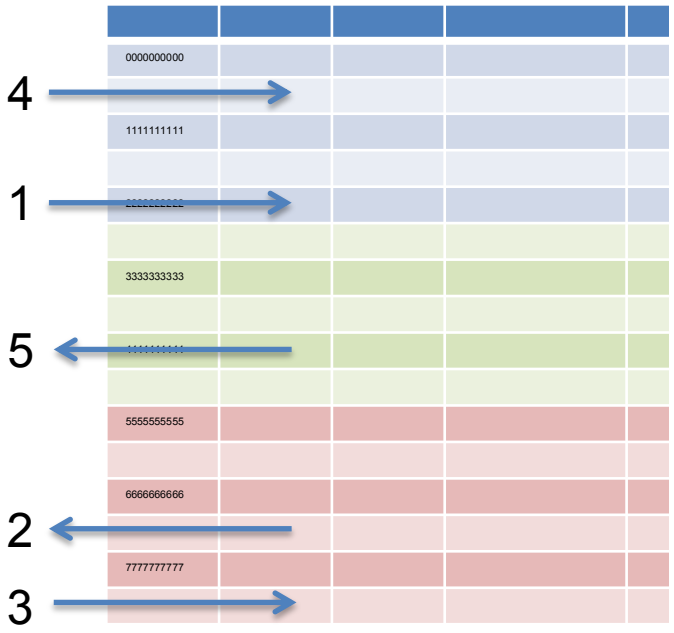
- **Apache Hadoop HDFS** for data **durability** and **reliability** (Write-Ahead Log)
- **Apache ZooKeeper** for distributed coordination
- **Apache Hadoop MapReduce** support built-in support for running MapReduce jobs



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

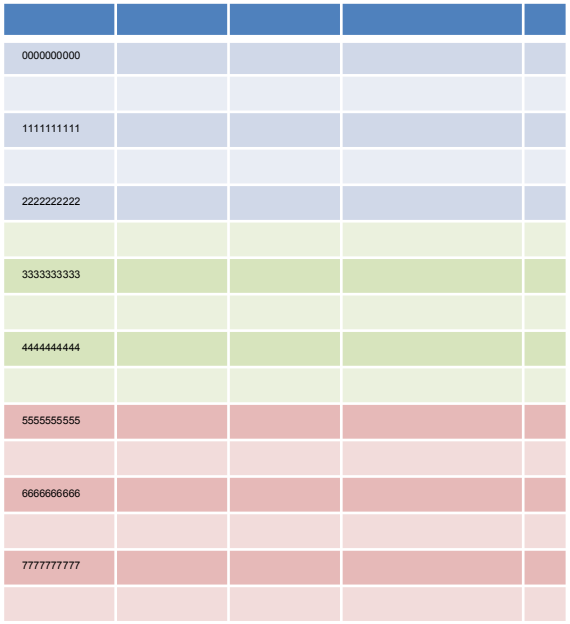


cloudera

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

Access HBase data via an API

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

- Access via HBase shell, Java API, REST proxy

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);
```

Simple API. What is the catch?

With great power comes great responsibility.

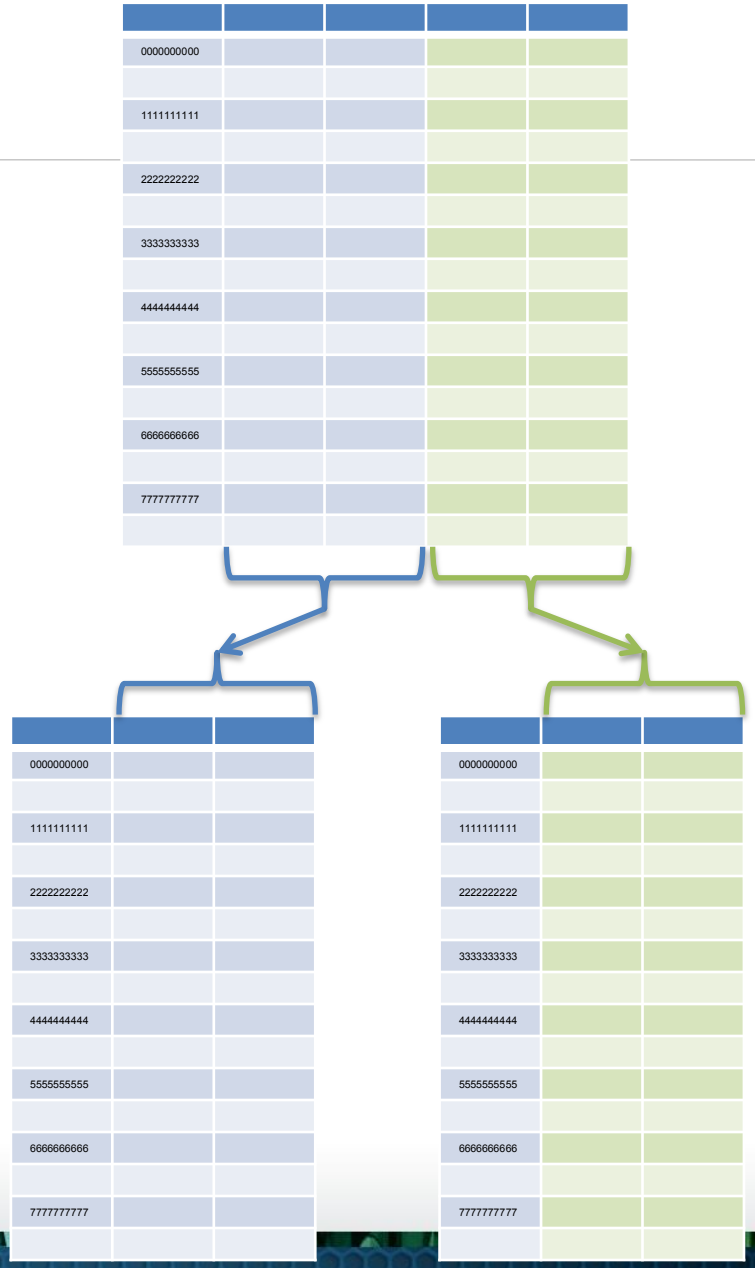
Cost Transparency

- 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

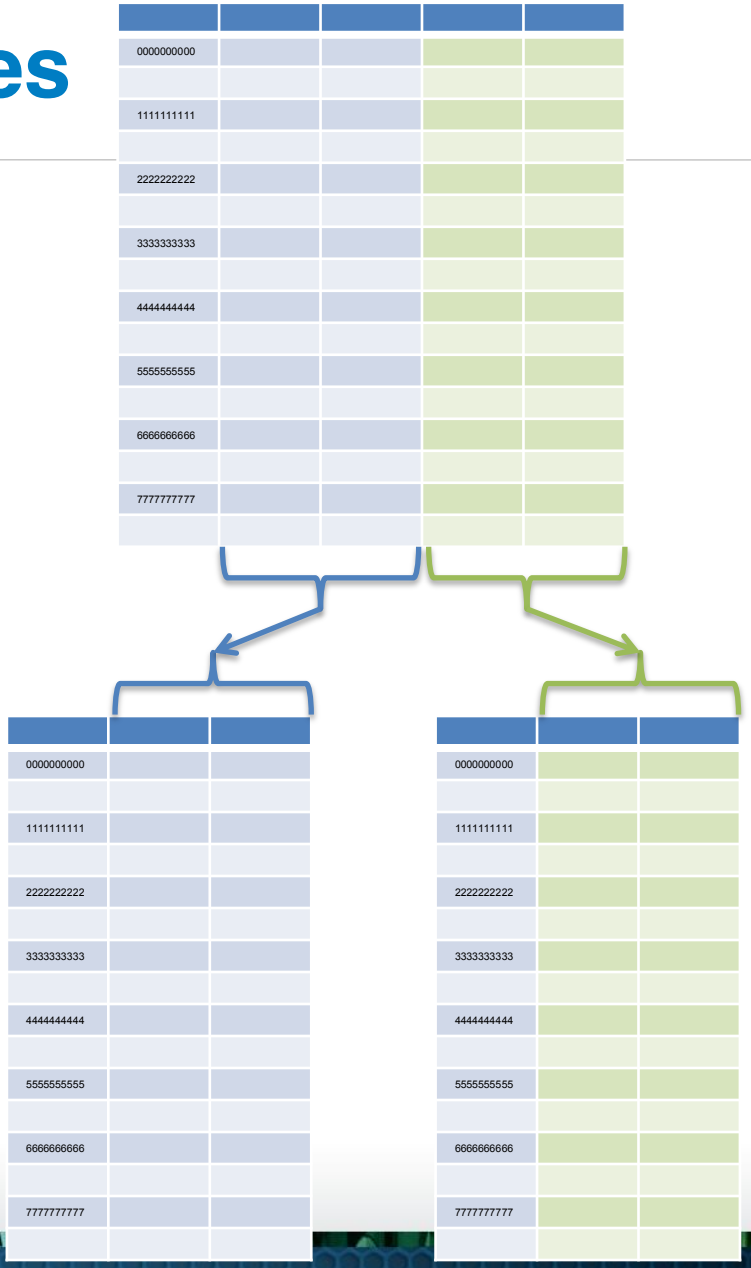
Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

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

Milliseconds since unix epoch

Tuning Column Families

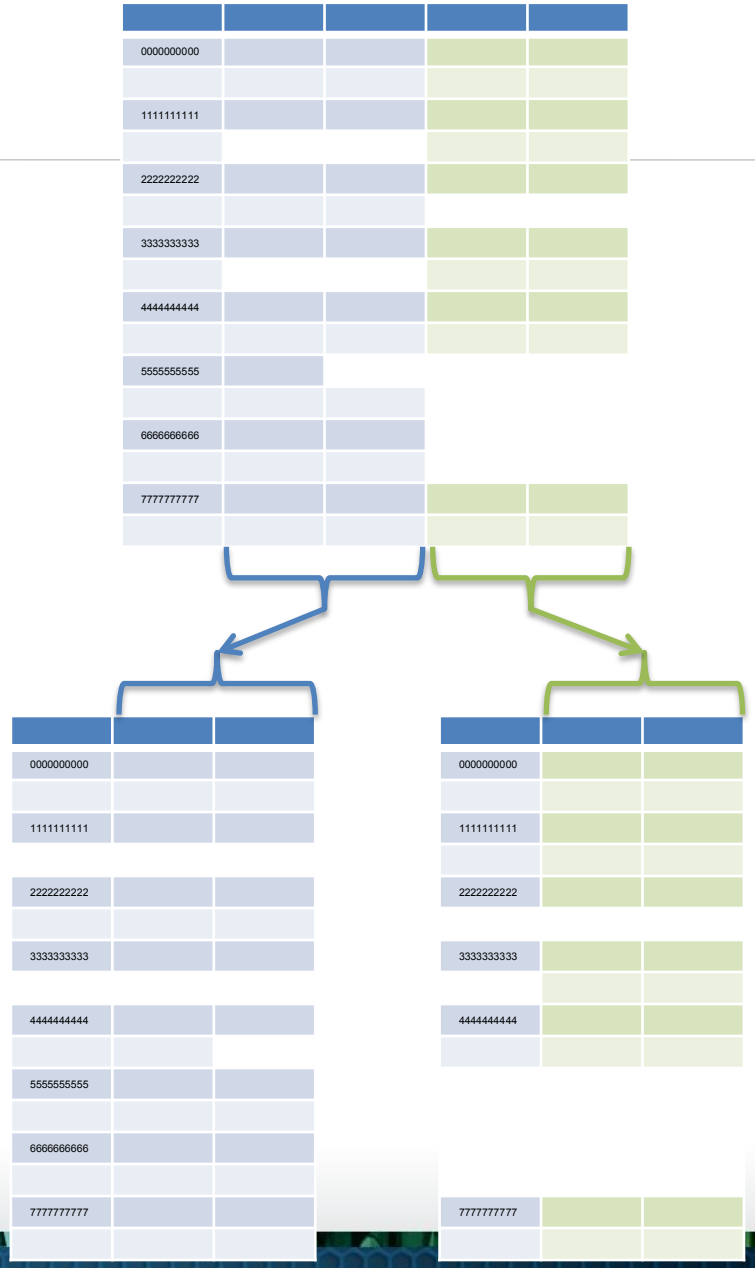
- Good for tuning **read performance**
  - Store related data together for better compression
  - Avoid polluting cache from another.
  - Derived data can have different retention policies
- Column family parameters
  - Block Compression (none, gzip, LZO, Snappy)
  - Version retention policies
  - Cache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



cloudera

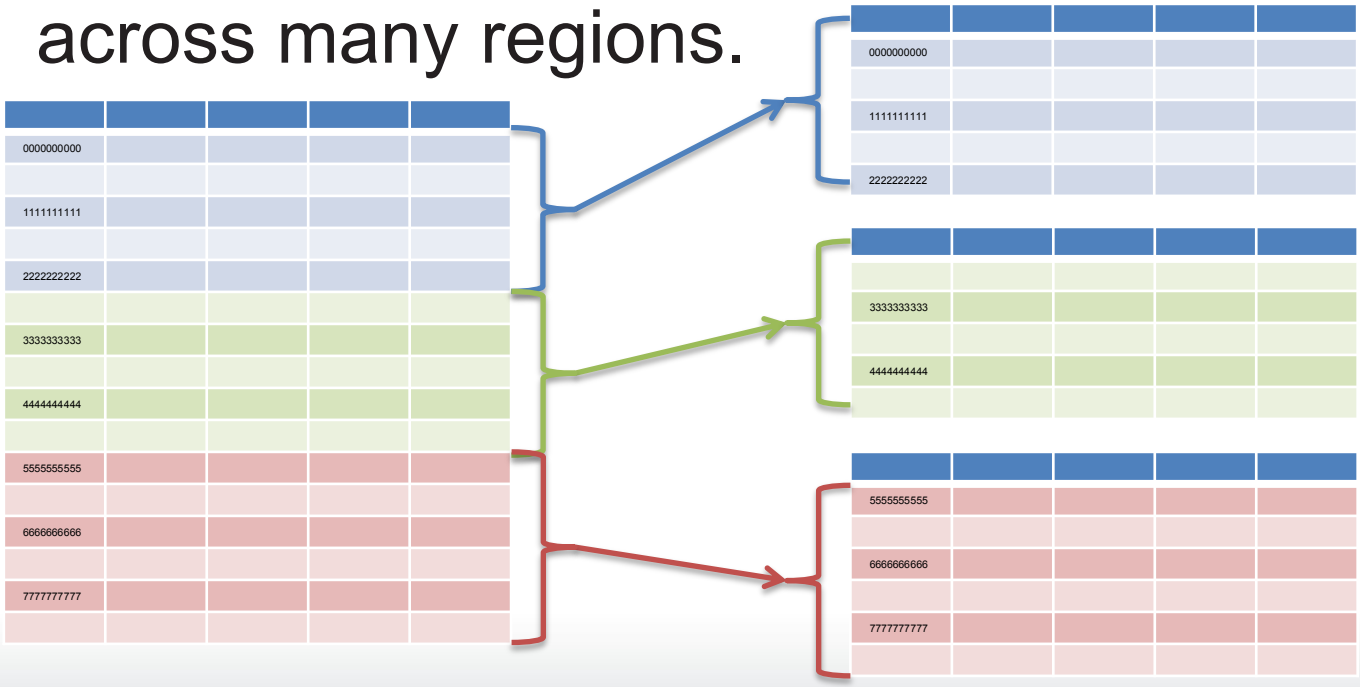
Sparse Columns

- 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.



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)

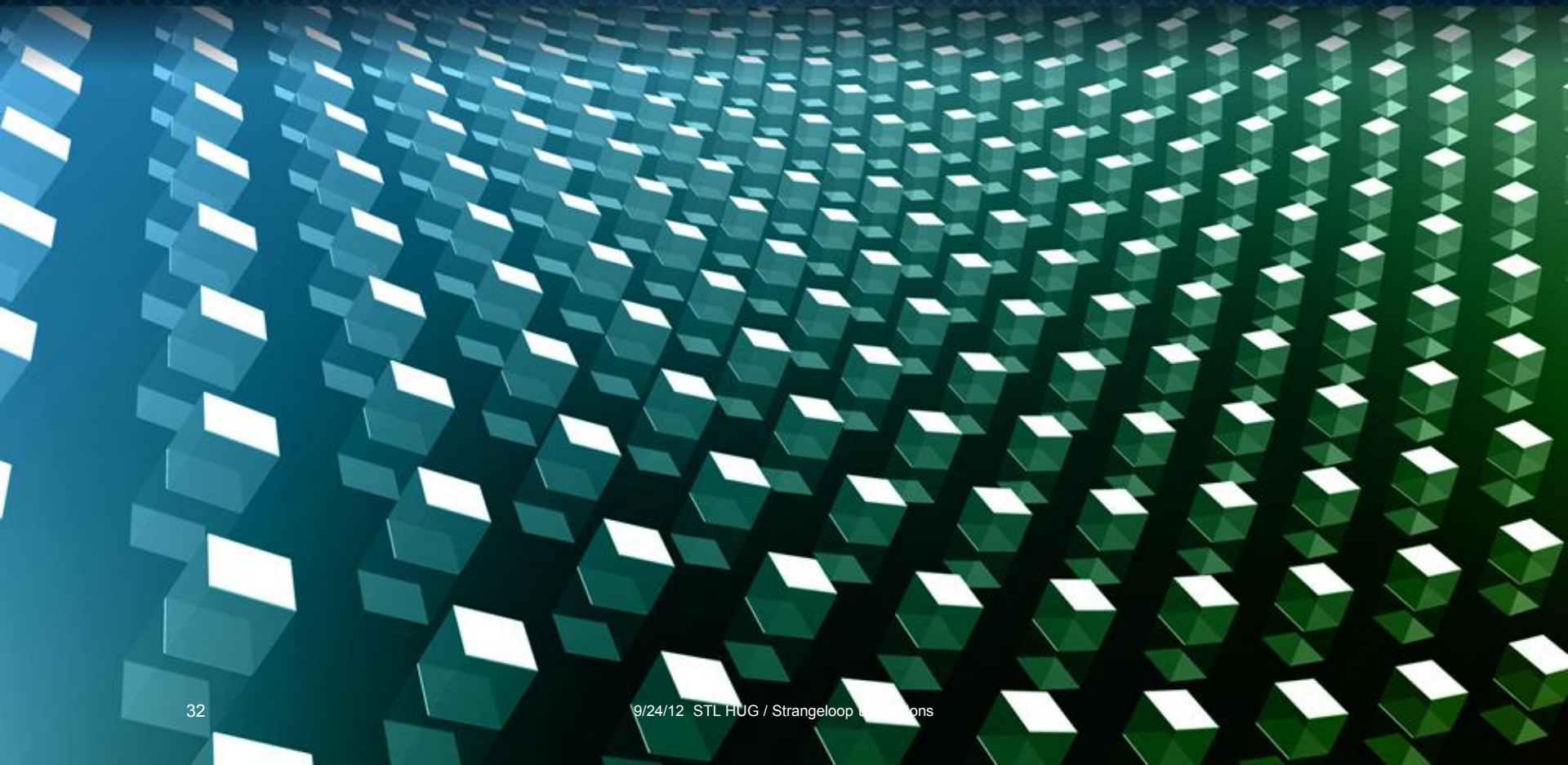
HBase vs RDBMS

	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

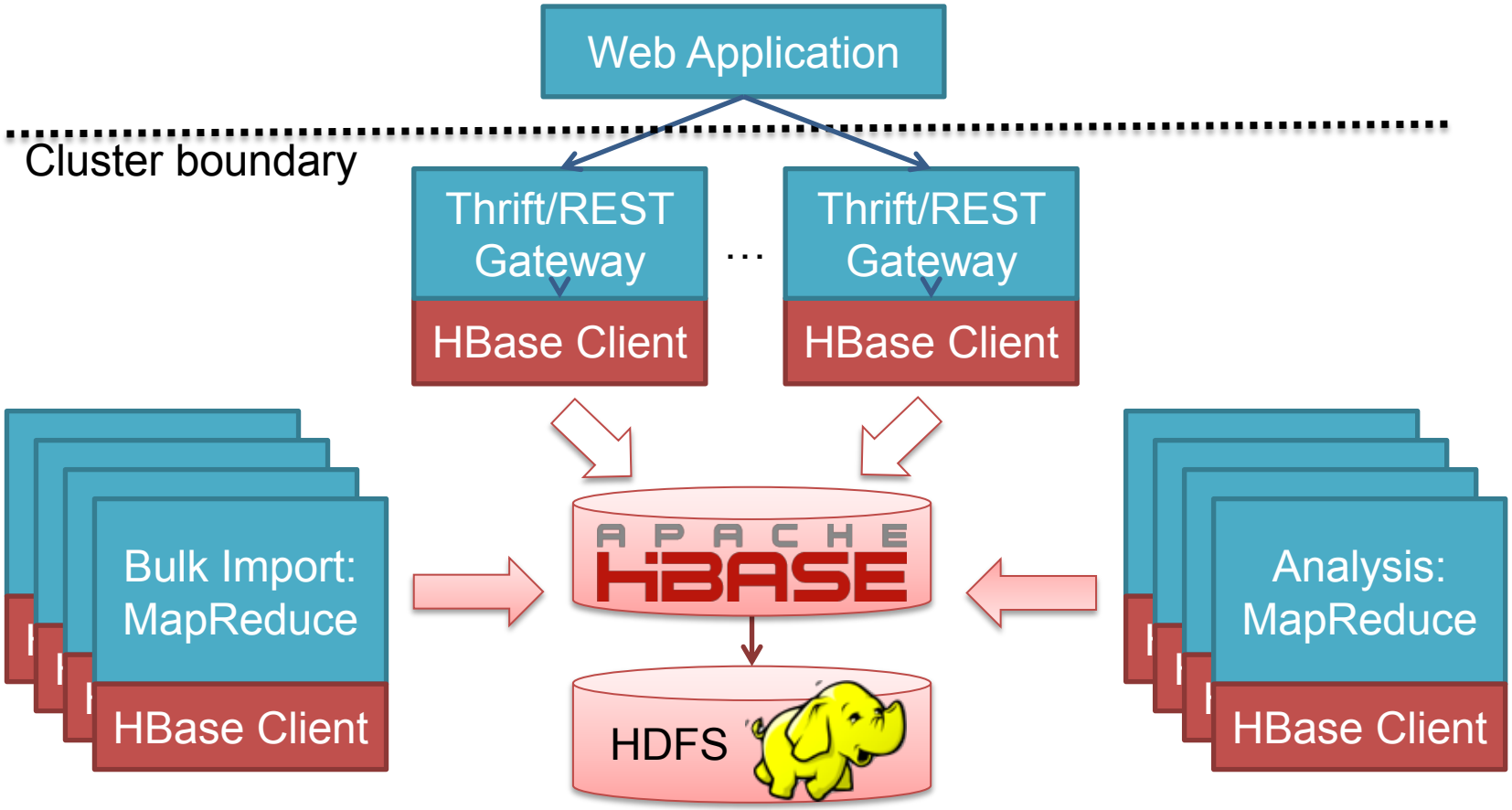
cloudera

REAL WORLD APPLICATIONS

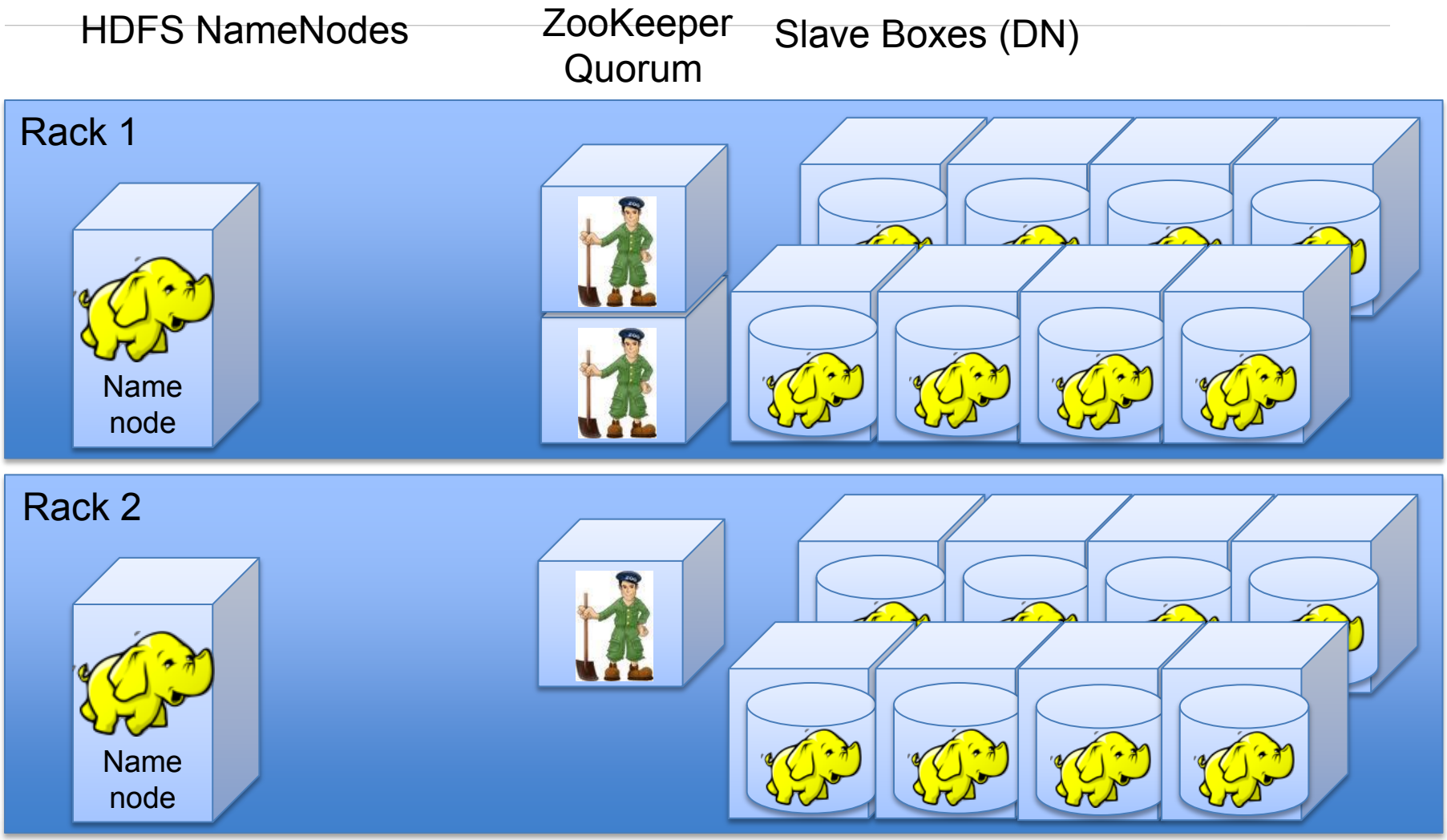
How and where is this infrastructure being used



HBase Application Architecture



HDFS Nodes (physically)

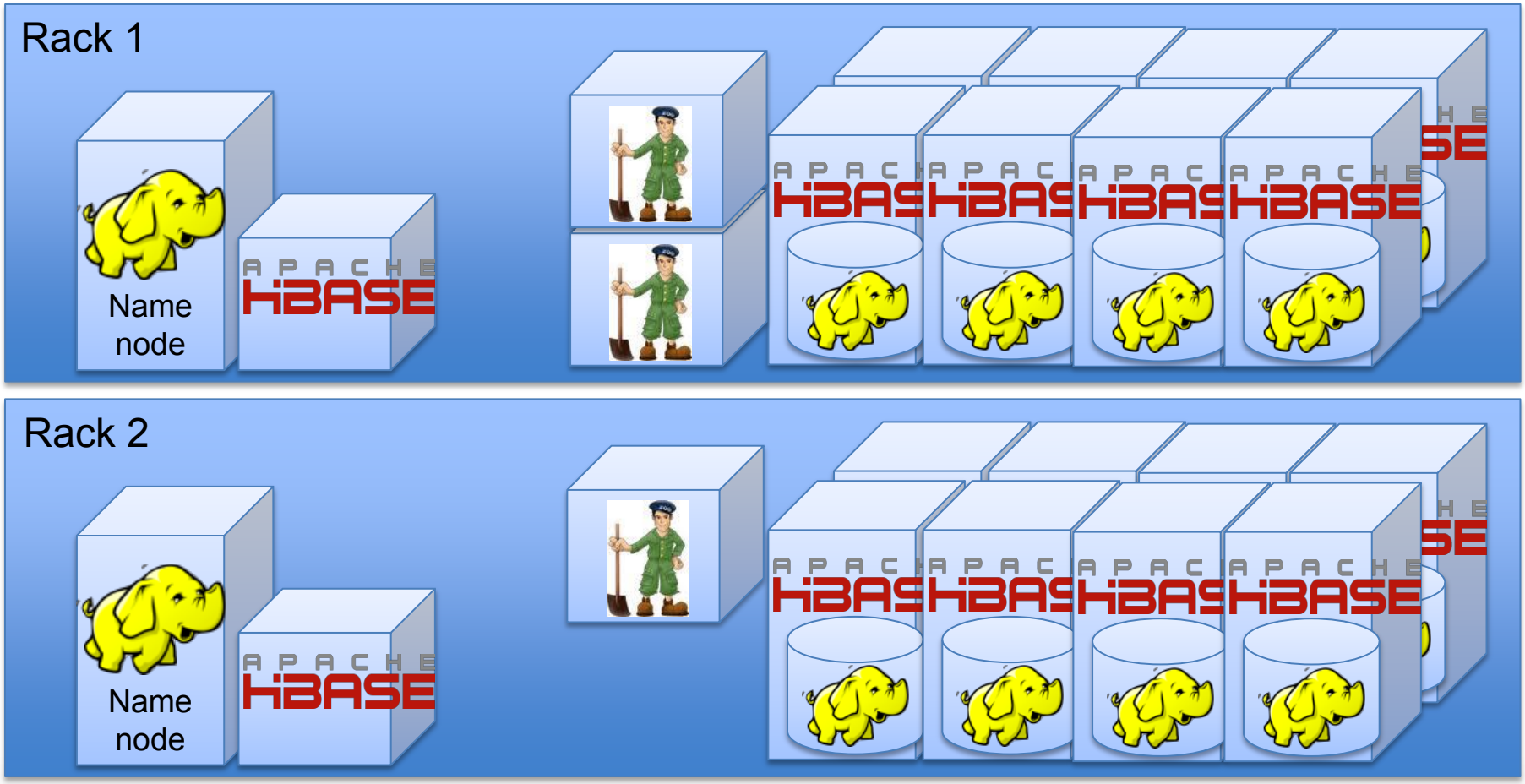


HBase + HDFS Nodes (physically)

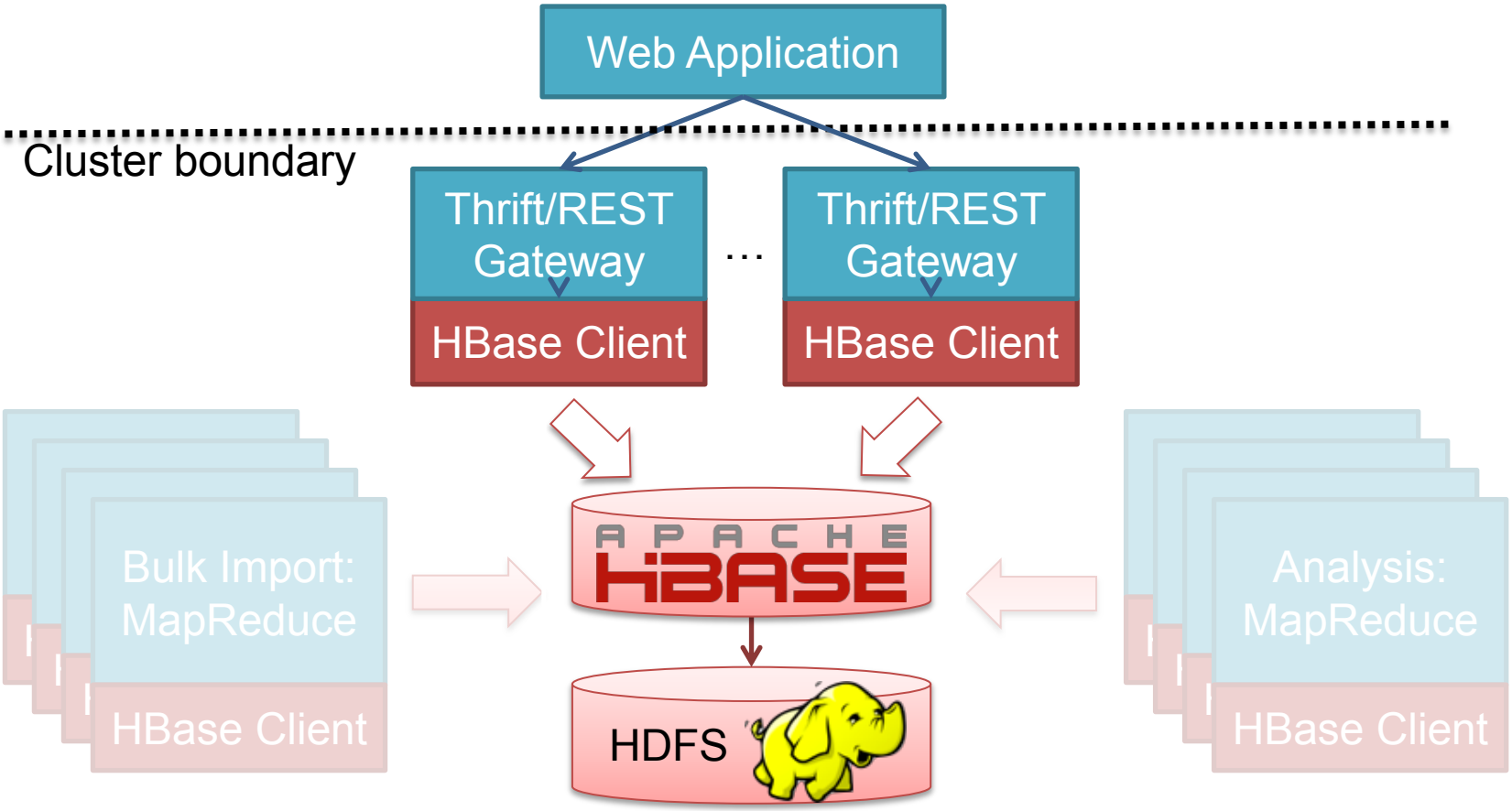
HDFS NameNodes
HBase Masters

ZooKeeper Quorum

Slave Boxes (DN + RS)



HBase Web Application

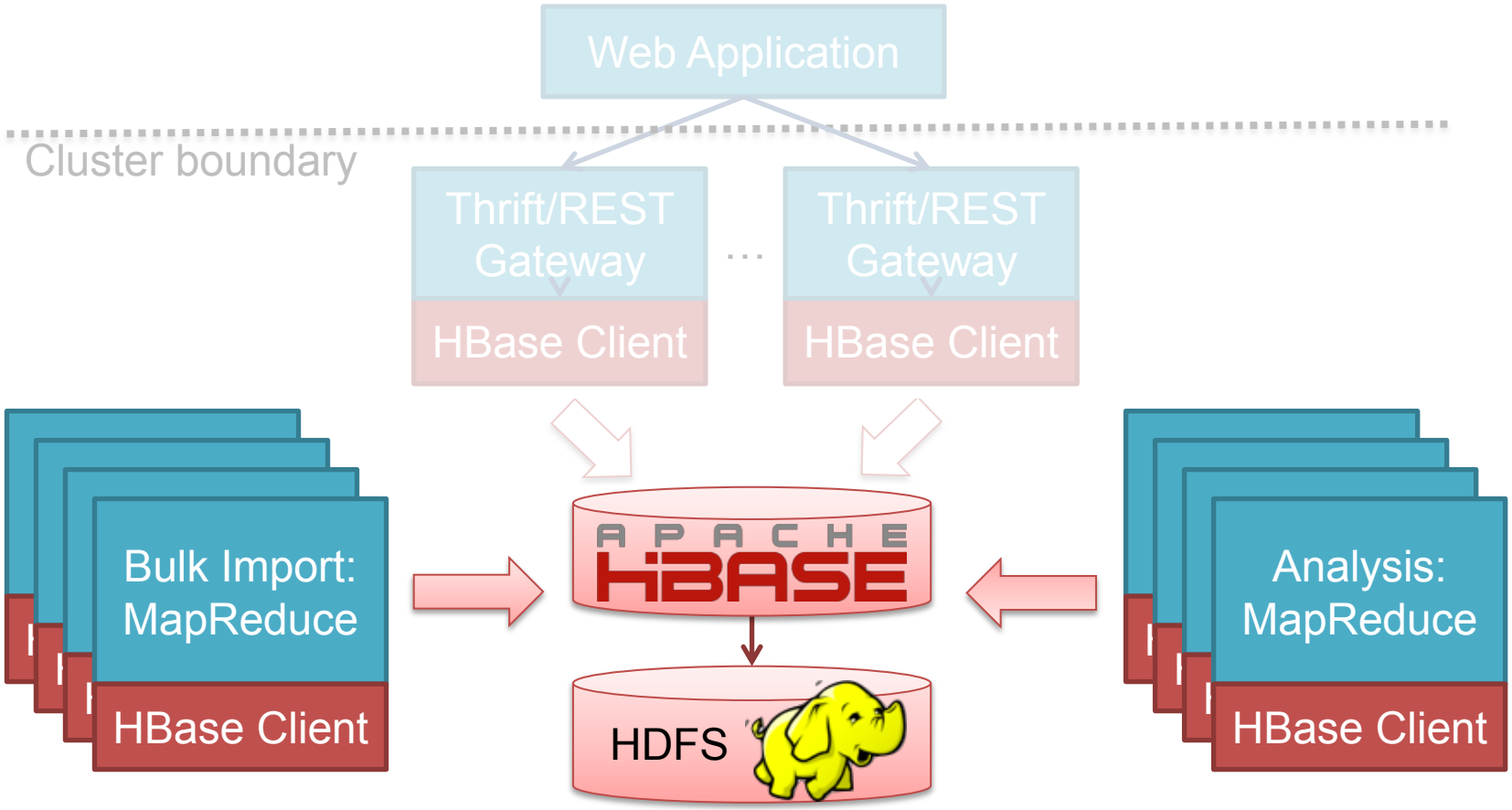


Example Apache HBase Applications

- 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,
- Monitoring Real-time Analytics
  - **OpenTSDB**, Sproxil, Sematext

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

HBase Analysis Application Architecture



HBase and MR examples

- Massive data store for Analysis
  - **Mozilla Socorro** (crash report DB)
  - Yahoo! Web Crawl Cache
  - Search Index: **eBay Cassini**, PhotoBucket, YapMap
  - Mignify / Internet Memory project

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

Data Access 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

cloudera

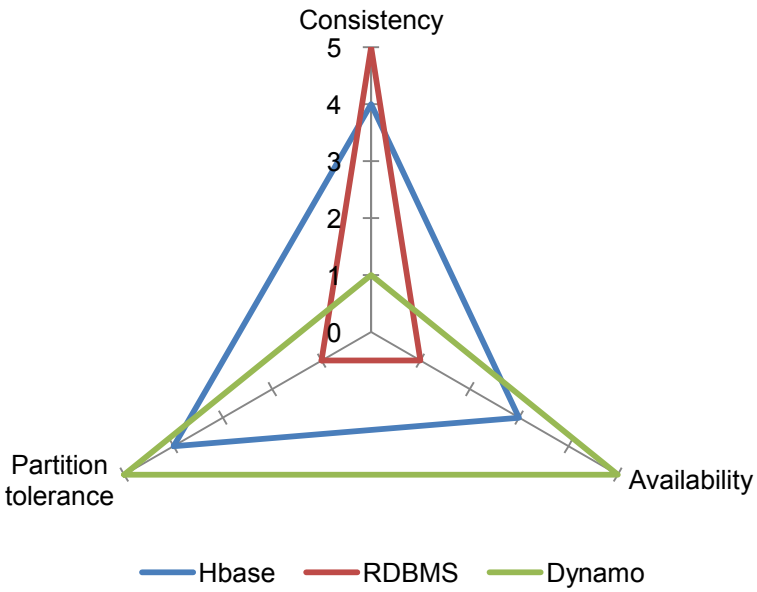
CONCLUSIONS

Key takeaways

- **Consistent, low latency, random access to structured data on top of Hadoop**
- Apache HBase is not a Relational Database!
- In production at 100's of TB scale at several large enterprises.
- HBase complements and depends upon Hadoop.

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)



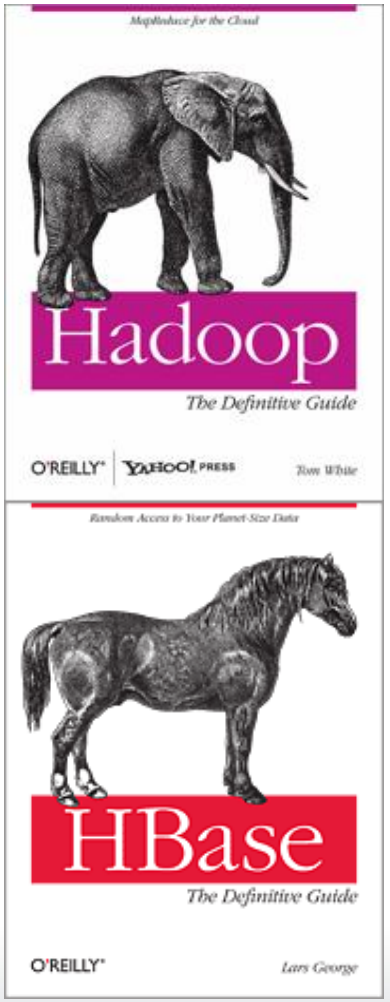
HBase vs just HDFS

	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
Hive (SQL) performance	Very good	4-5x slower
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!

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

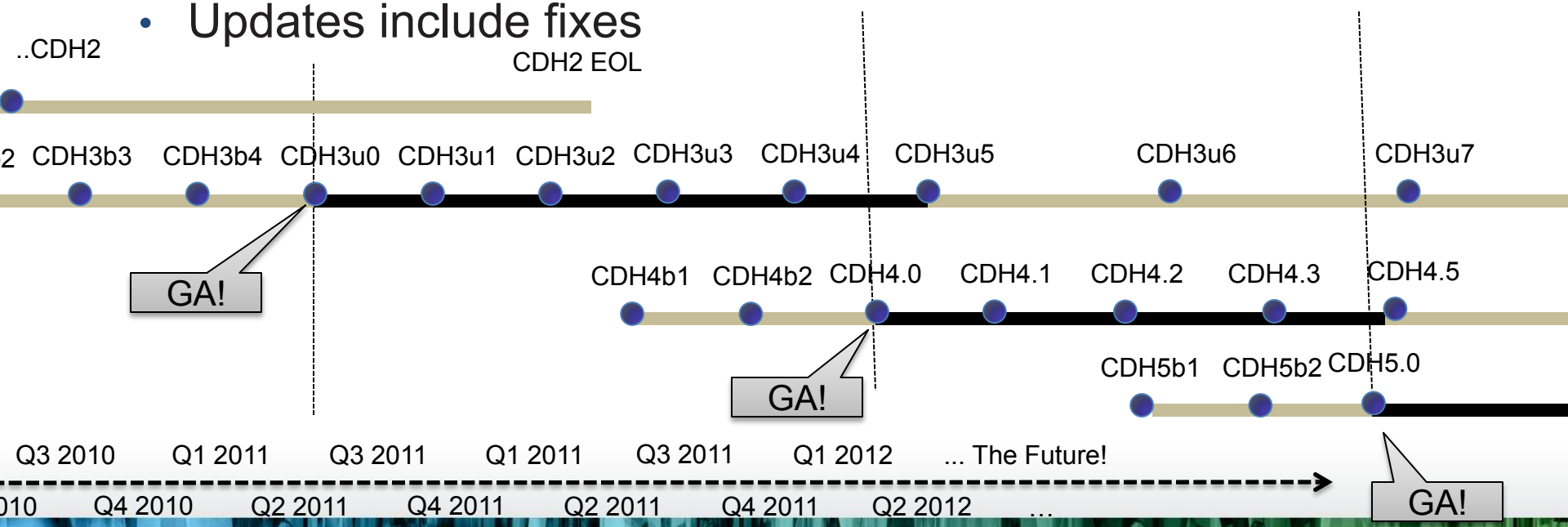
jon@cloudera.com

@cloudera

QUESTIONS?

Predictable Release Schedule

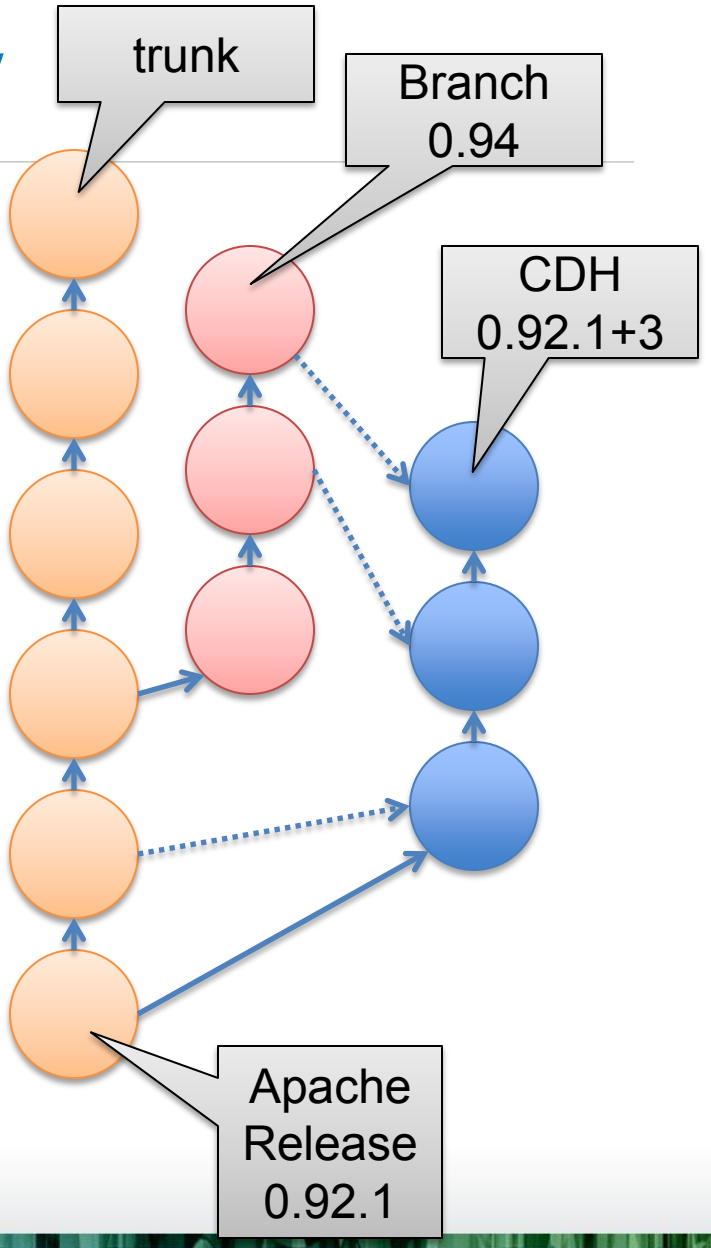
- Regular release and updates
  - Skip updates without penalty
- Compatibility policy
  - Only major releases break compatibility
  - Updates can include new features
  - Updates include fixes



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

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`