CLASS 3
Hadoop-Related Tools

CSCI 6830

BIG DATA ANALYTICS WITH HADOOP AND R

Hadoop-Related Tools

- **Ambari**
- **Avro**
- Cassandra
- **Chukwa**
- **Hbase**
- ***Hive***
- Mahout
- ***Pig***
- Spark
- Tez
- Zookeeper

Ambari

Ambari. What it does?

- Open-source tool for **monitoring** Hadoop cluster
- Ambari is distributed with HortonWorks Hadoop
- Simplified **deployment** and **maintenance** of hosts
- Intuitive Web interface allows easily **provision**, **configuration** and **testing** of all the Hadoop services and core components
- Ambari Blueprints API for **automating cluster installations** without user intervention
- **Manage** a Hadoop cluster via **Web interface** to:
  - Control the lifecycle of Hadoop services and components
  - Modify configurations
  - Manage the ongoing growth of your cluster.

What Ambari does? (more)

- **Monitor a Hadoop cluster**
  - Insight into the health of a cluster.
  - Alerts for watching Hadoop services
  - Visualizes cluster operational data in a simple Web interface
- **Integrate Hadoop with the Enterprise**
  - Provides a RESTful API for integration with existing tools:
    - Microsoft System Center Operations Manager, HP Operations Manager and Teradata Viewpoint

Hadoop components supported by Ambari (1)

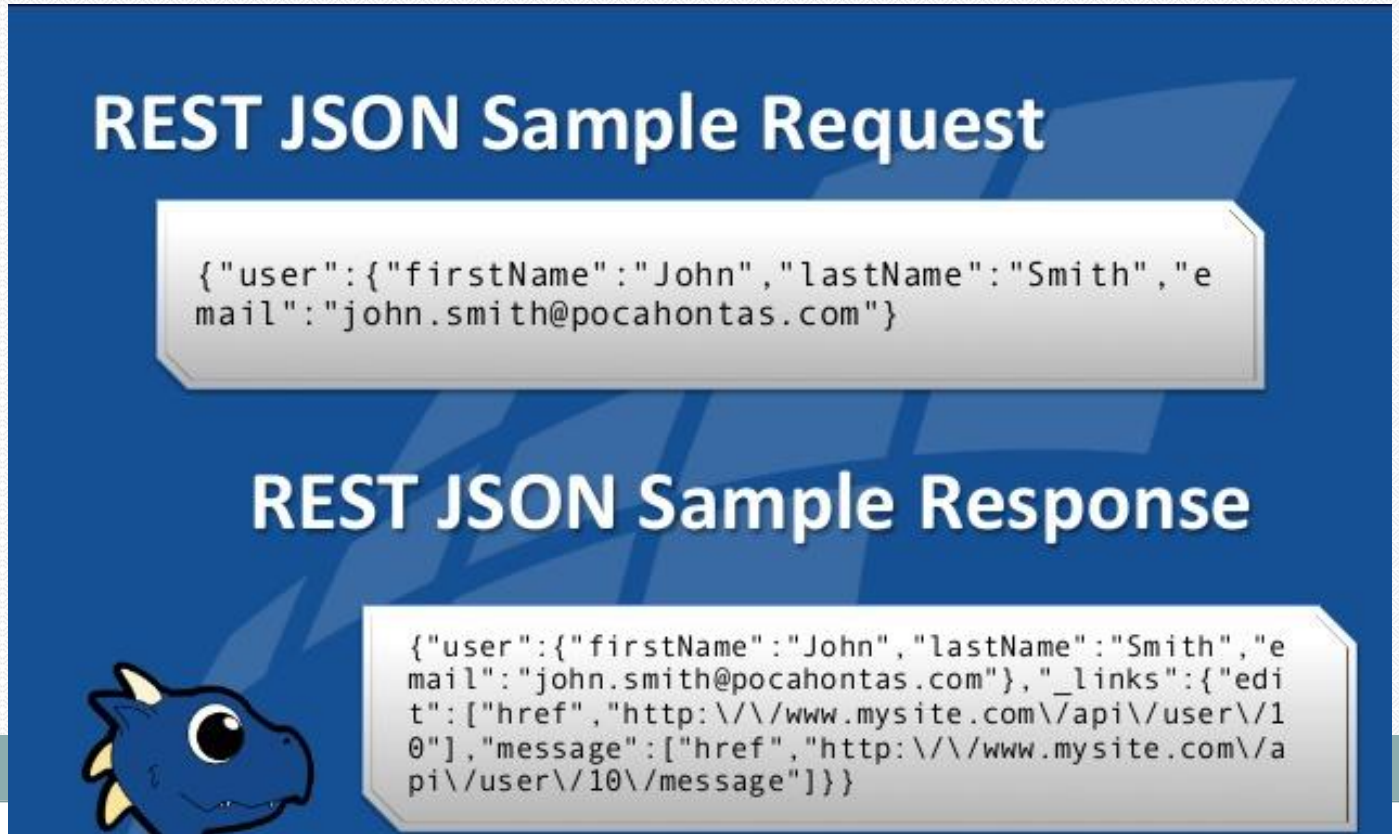
- **HDFS**. Abari allows management of NameNodes, Secondary NameNodes, and DataNodes.
- **MapReduce**. Ambari manages the default settings for MapReduce. These settings are used in YARN by MarReduce Application Manager.
- **Hive**. Data warehouse tool.
- **Pig**. A tool for creating data flow programs.
- **HCatalog**. Metadata storage for MapReduce, Hive & Pig.
- **HBase**. A column-oriented database which can be viewed as a multidimensional key value store.
- **ZooKeeper**. Centralized service for naming & distributed syncronization services.

Hadoop components supported by Ambari (2)

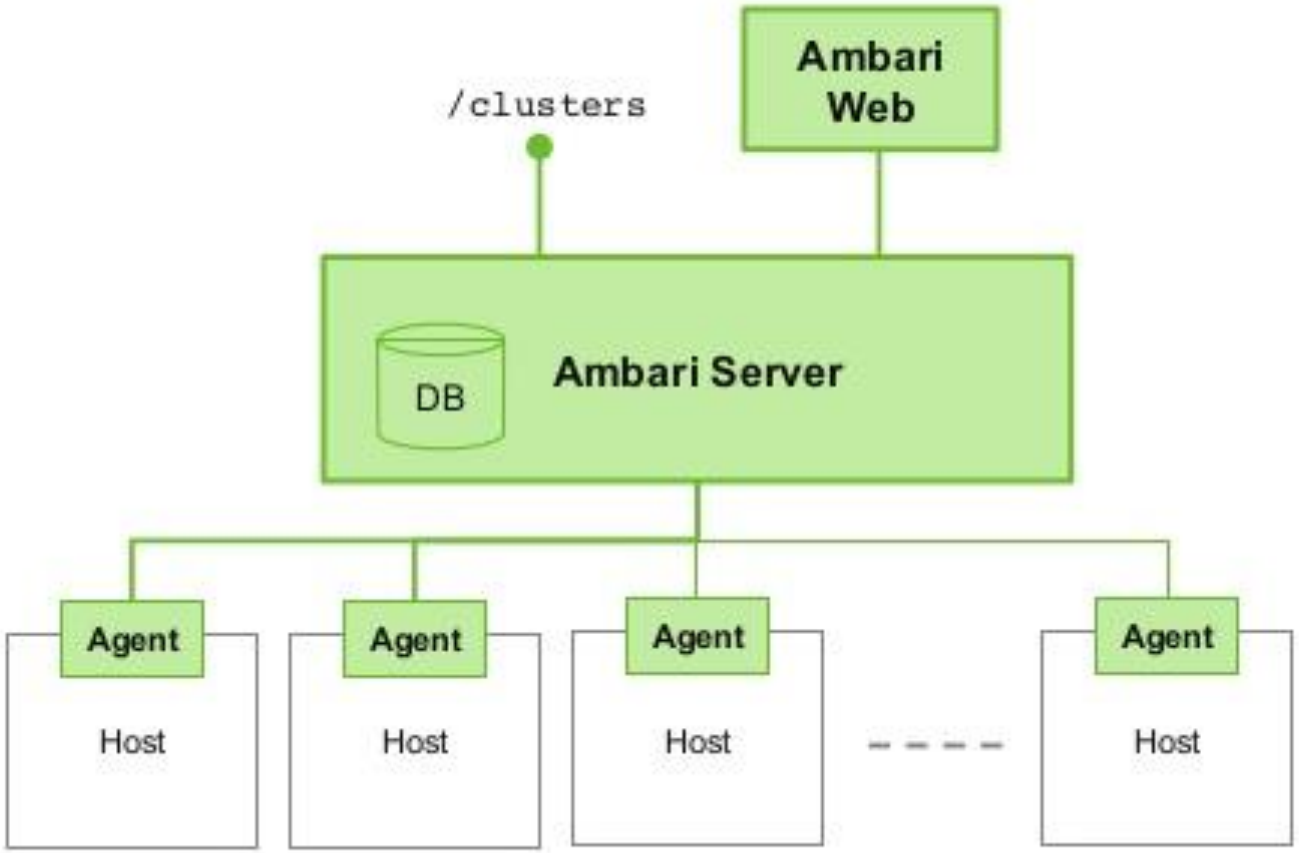
- **Oozie**. A workflow engine for Pig and MapReduce.
- **Sqoop**. Used for import/export from external systems to HDFS, e.g. RDBMS to flat files to Hive/Pig.
- Hadoop cluster provisioning.
  - Web interface to install Hadoop on a cluster.
  - Centralized configuration of Hadoop cluster.
- Hadoop cluster monitoring.
  - Uses Ganglia for metrics collection. A Ganglia monitor is installed on each remote host, which collects metrics and sends them back.
  - Uses Nagios for notifications. Nagios provides alerts on system states.
- REST-based API are used by:
  - Web-based dashboard
  - Application developers

Hold on, what is REST API?

- REST stands for **Re**presentational **S**tate **T**ransfer
- Lightweight API for accessing web services
  - And some other uses



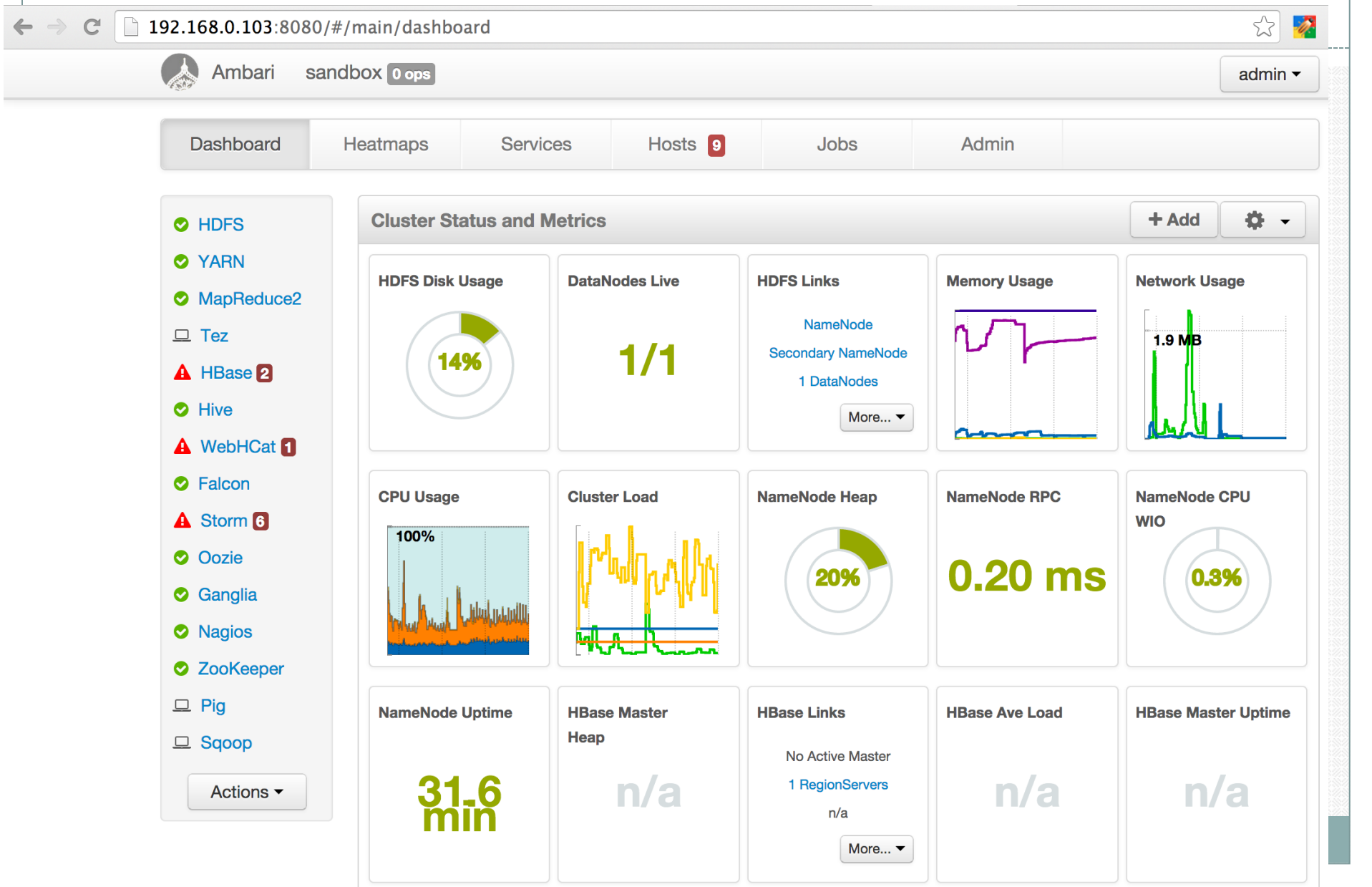
System Architecture



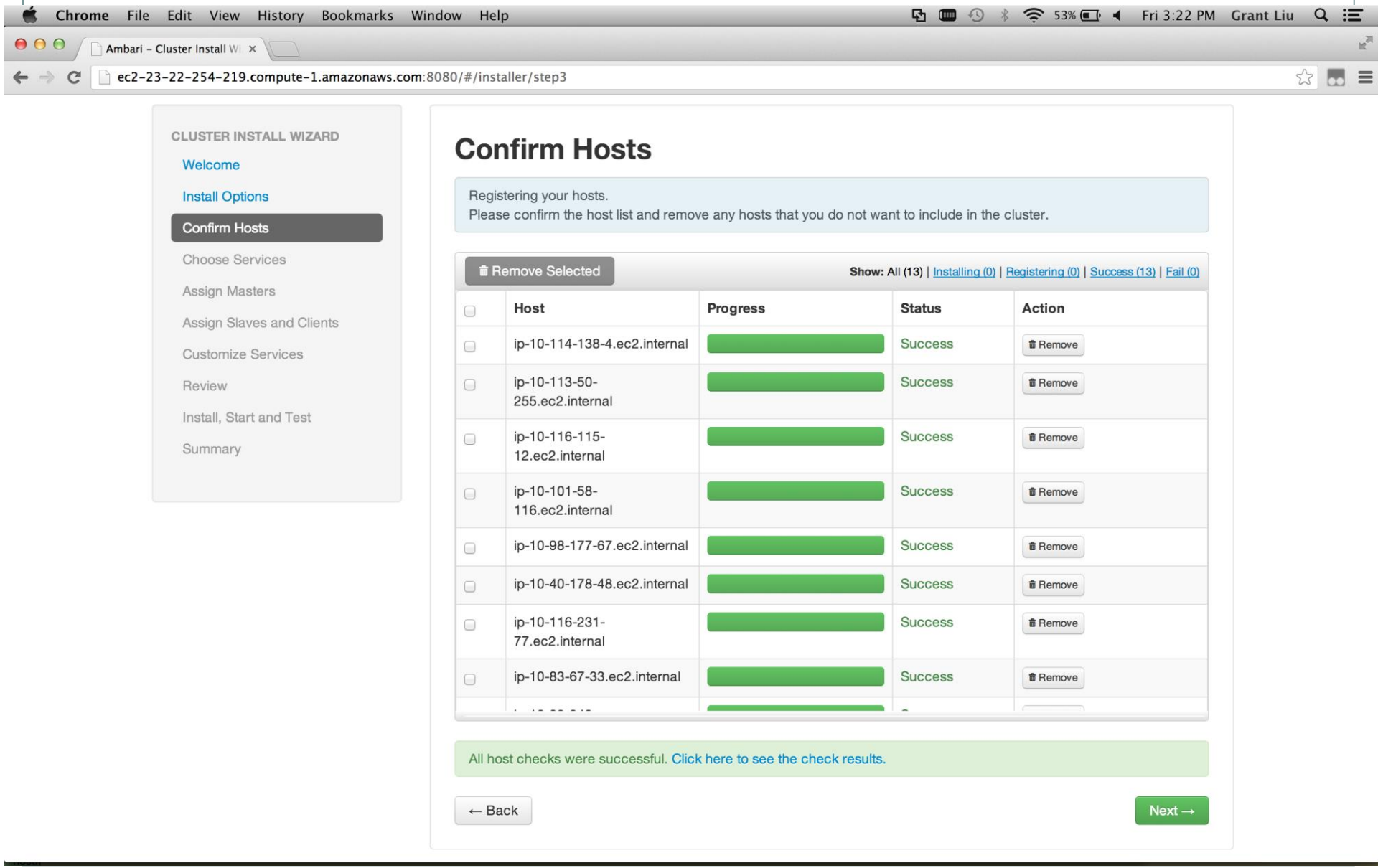
Ambari Features and Benefits

Feature	Benefit
Wizard-driven interface	Facilitates installation of Hadoop across any number of hosts
API-driven installations	Ambari Blueprints for automated provisioning
Granular service control	Precise management of Hadoop services and component lifecycles
Configuration change history	Ongoing management of Hadoop service configurations
RESTful APIs	Enables integration with enterprise systems
Extensible framework	Brings custom services under management via Ambari Stacks
Customizable user interface	Develop innovative user experiences via Ambari Views Framework
User Views	Advanced capabilities for cluster optimization and tuning for Hadoop DevOps

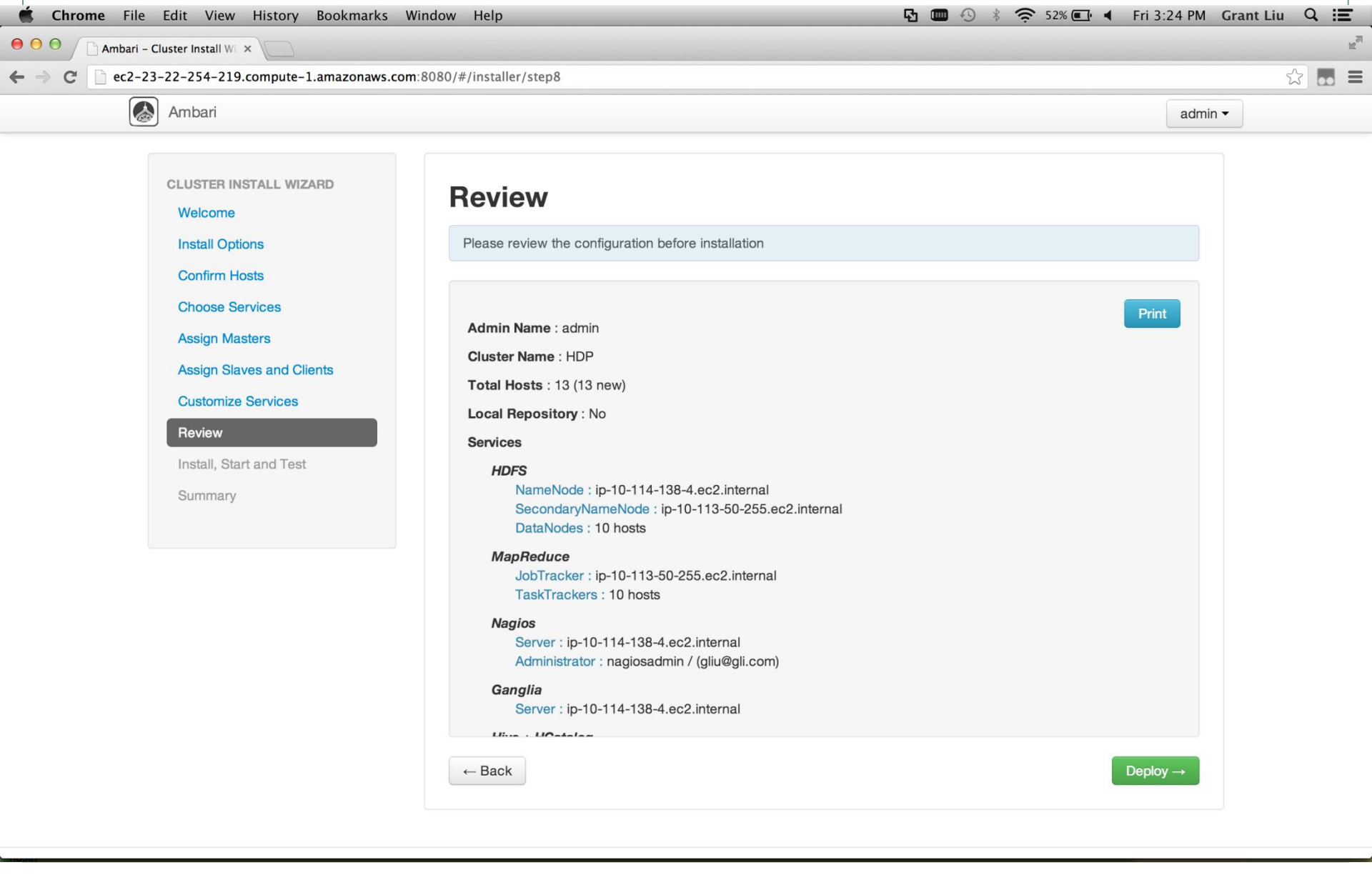
Ambari Interface: dashboard



Ambari: installing a cluster (step 3)



Ambari: installing a cluster (step 8)



Ambari's Competitors

- **CloudEra Manager**
- Syncfusion Big Data **Cluster Manager**
- **Chukwa**: a monitoring service
- **FailMon**: a hardware diagnostic tool
- **Ganglia**: a visual monitoring tool with history
- **JMX**: Hadoop core server and task state monitor
- **Karmasphere** (sold to FICO)
- **Nagios**: a monitoring and alert generation framework
- **X-Trace**. Generates data about distributed transactions, latency and bottlenecks, and flow of control in distributed systems.

Avro

LANGUAGE-NEUTRAL DATA SERIALIZATION SYSTEM

Avro History

- Avro was a British aircraft manufacturer founded in 1910

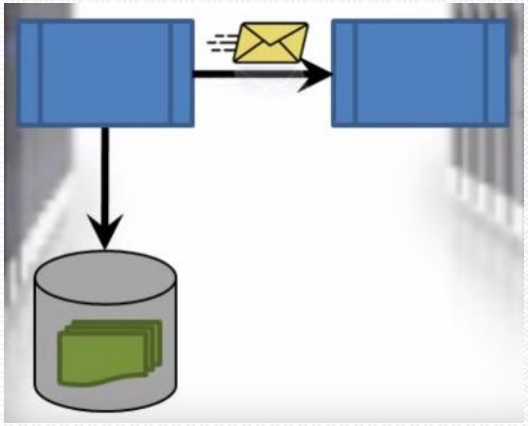
The A.V. Roe Type I Triplane, Roe's first successful aircraft

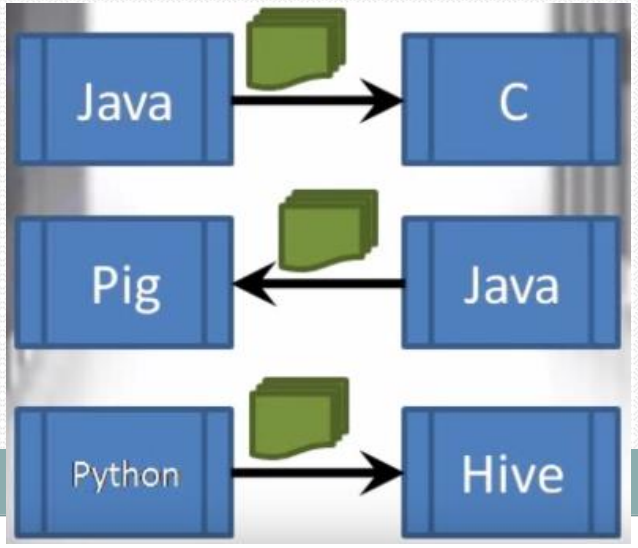
The Avro Lancaster

Avro Vulcan

Apache Avro

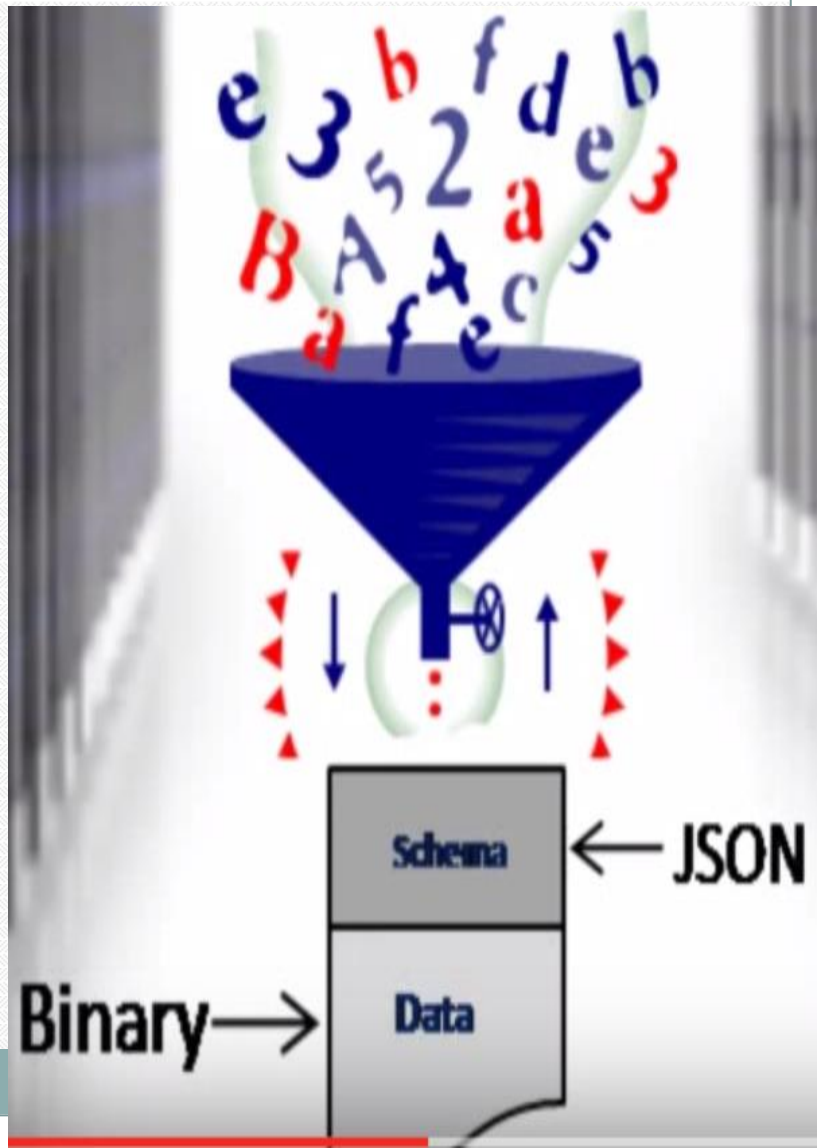
- Provides two important services for Hadoop
  - Data serialization
  - Data exchange
- Avro can serialize data into
  - Files
  - Messages
- Avro allows to exchange data between any language





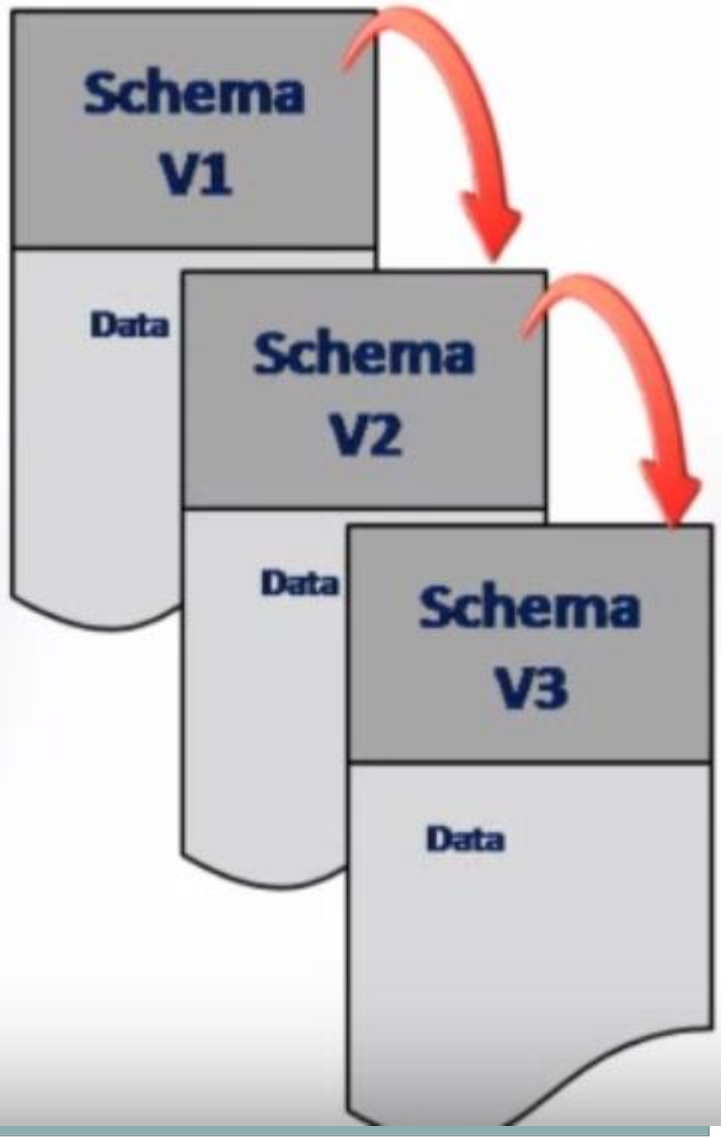
More on Avro

- Avro stores in compact and efficient format
  - Stores both definitions and data in one message or file
- Data definition is in JSON format
- Data is in binary format
- Avro files include markers
  - Allows efficient splitting for MapReduce processing



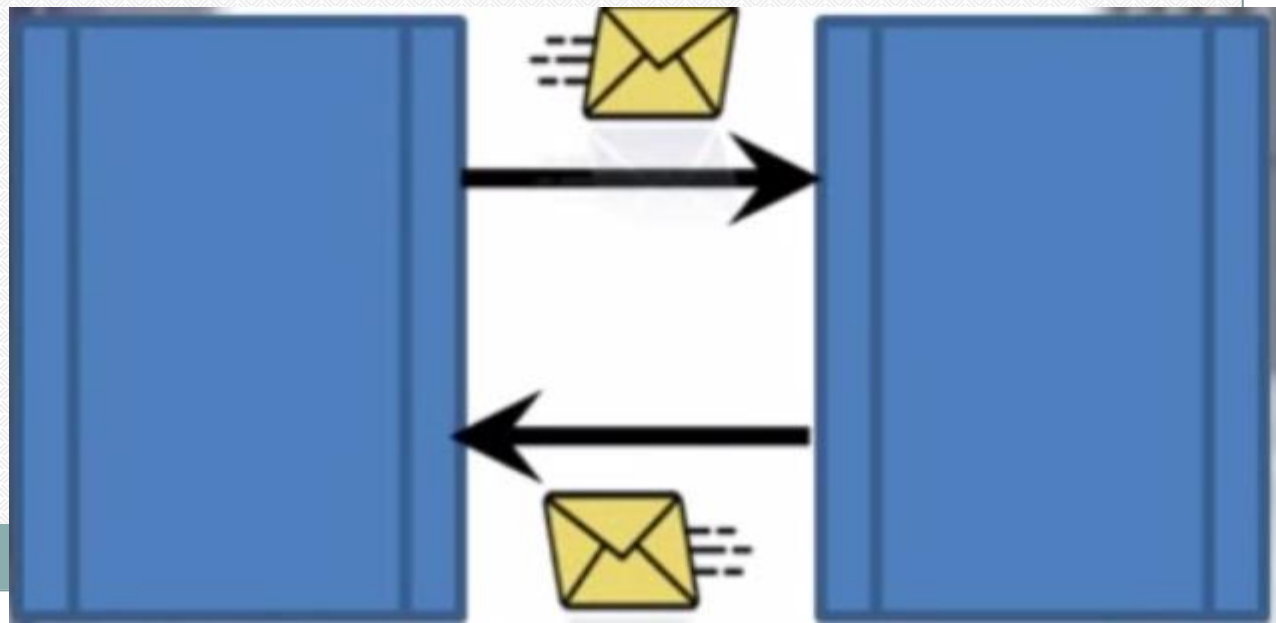
Avro Data Schemas Evolution

- Avro supports data schemas, which change over time
- Known as Schema Evolution
- Handles changes:
  - Missing fields
  - Added fields
  - Changed fields
- Old programs can read new data
- New programs can read old data



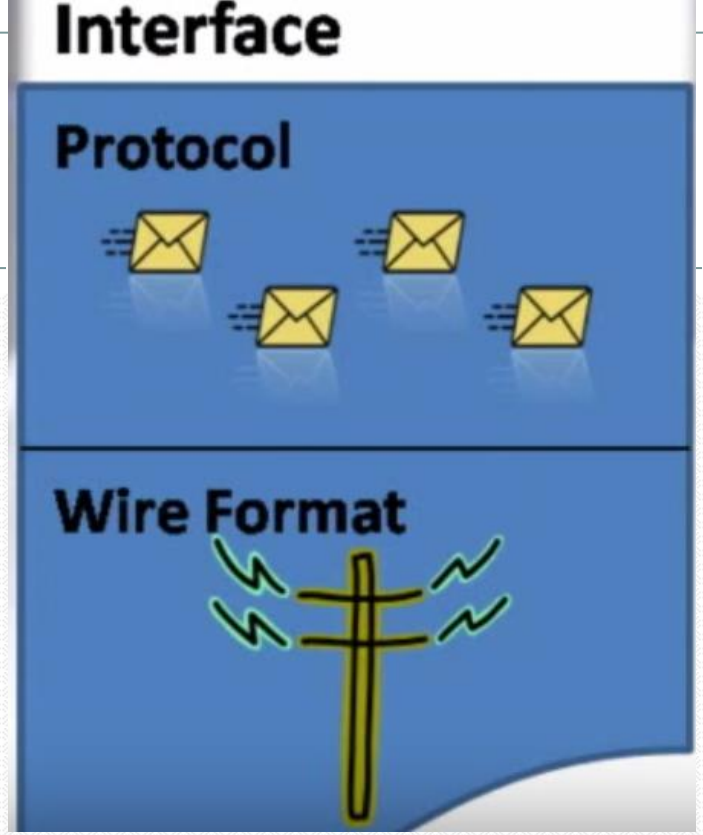
Avro Remote Procedure Calls (RPC)

- Easy to pass data from program in one language to a program in another language
- Avro data exchange service is done by RPC



Avro RPC Interfaces

- Avro RPC interfaces are done in JSON
  - Protocol declaration
    - Defines messages
    - Messages are described with Avro data schemas
  - Wire format
    - How request and response are set, received, and buffered
    - Handshake protocol
    - Request and response format



Apache Avro

- Is a data serialization system
- Focuses on dynamic access, platform independence, and simple schema evolution
- Schema language – JSON
- Remote Procedure Call (RPC) framework to send data
- APIs for C, C++, C#, Java, JavaScript, Perl, PHP, Python, and Ruby
- Schema stored with data
- Arbitrary data types
- Does not have to generate and load code (optional)

Apache Avro

- Language-neutral data serialization system
- Addresses the major downside of Hadoop Writables: lack of language portability.
- Avro data format can be processed by many languages
  - C, C++, C#, Java, JavaScript, Perl, PHP, Python, and Ruby
- Avro data is described using a language-independent *schema*
- Avro schemas are usually written in JSON
- Data is usually encoded using a binary format

Avro Example Schema

```
{
  'type': 'record', 'name':'Widget',
  'fields': [
    {'name':'size','type':'long',
      'default':0},
    {'name':'density','type':'float'}
  ]
}
```

Avro Use Case

- Define schema for a customer record

```
{
    "namespace": "customer.avro",
    "type": "record",
    "name": "customer",
    "fields": [
        {"name": "name",    "type": "string"},
        {"name": "age",     "type": "int"},
        {"name": "address", "type": "string"},
        {"name": "phone",   "type": "string"},
        {"name": "email",   "type": "string"},
    ]
}
```

Avro Use Case (continued)

```
// compile the schema

java -jar /...../avro-tools-1.7.4.jar compile schema customer.avsc .

// create a couple of customers

Customer cust1 = new Customer("Dave",  33, "12 redmont street", "021234675", "davef@somedom@co.nz"  );
Customer cust2 = new Customer("Emily",  27, "13a lake cresent",  "022876234", "esuanders@somenz@co.nz"  );

// now serialize customers to disk

File file = new File("customers.avro");
DatumWriter<Customer>  userDatumWriter  = new SpecificDatumWriter<Customer>(Customer.class);
DataFileWriter<Customer>  dataFileWriter = new DataFileWriter<Customer>(customerDatumWriter);
dataFileWriter.create(cust1.getSchema(),  new File("customers.avro"));
dataFileWriter.append(cust1);
dataFileWriter.append(cust2);
dataFileWriter.close();
```

How good is Avro?

- Cyber Monday 2013 (one day)
  - More than 320,000 events per second
  - 7 Storm topologies consuming the events seconds from real time
  - 2TB of data saved to Hadoop
- 2014 preparation:
  - x2 number of events per second to ~640,000

Avro

- Avro specification precisely defines the binary format that all implementations must support.
- Avro has rich *schema resolution* capabilities
  - Schema used to read data need not be identical to the schema that was used to write the data.
- *Avro datafile* has a metadata section where the schema is stored
  - Makes the file self-describing

Avro Primitive Data Types

```
{ "type": "null" }
```

Type	Description	Schema
`null`	The absence of a value	`"null"`
`boolean`	A binary value	`"boolean"`
`int`	32-bit signed integer	`"int"`
`long`	64-bit signed integer	`"long"`
`float`	Single-precision (32-bit) IEEE 754 floating-point number	`"float"`
`double`	Double-precision (64-bit) IEEE 754 floating-point number	`"double"`
`bytes`	Sequence of 8-bit unsigned bytes	`"bytes"`
`string`	Sequence of Unicode characters	`"string"`

Avro Complex Data Types

Type	Description	Schema example
`array`	An ordered collection of objects. All objects in a particular array must have the same schema.	`{ "type": "array", "items": "long" }`
`map`	An unordered collection of key-value pairs. Keys must be strings and values may be any type, although within a particular map, all values must have the same schema.	`{ "type": "map", "values": "string" }`
`record`	A collection of named fields of any type.	`{ "type": "record", "name": "WeatherRecord", "doc": "A weather reading.", "fields": [ {"name": "year", "type": "int"}, {"name": "temperature", "type": "int"}, {"name": "stationId", "type": "string"} ] }`

Avro Complex Data Types (2)

`enum` A set of named values.

```
{
  "type": "enum",
  "name": "Cutlery",
  "doc": "An eating utensil.",
  "symbols": ["KNIFE", "FORK", "SPOON"]
}
```

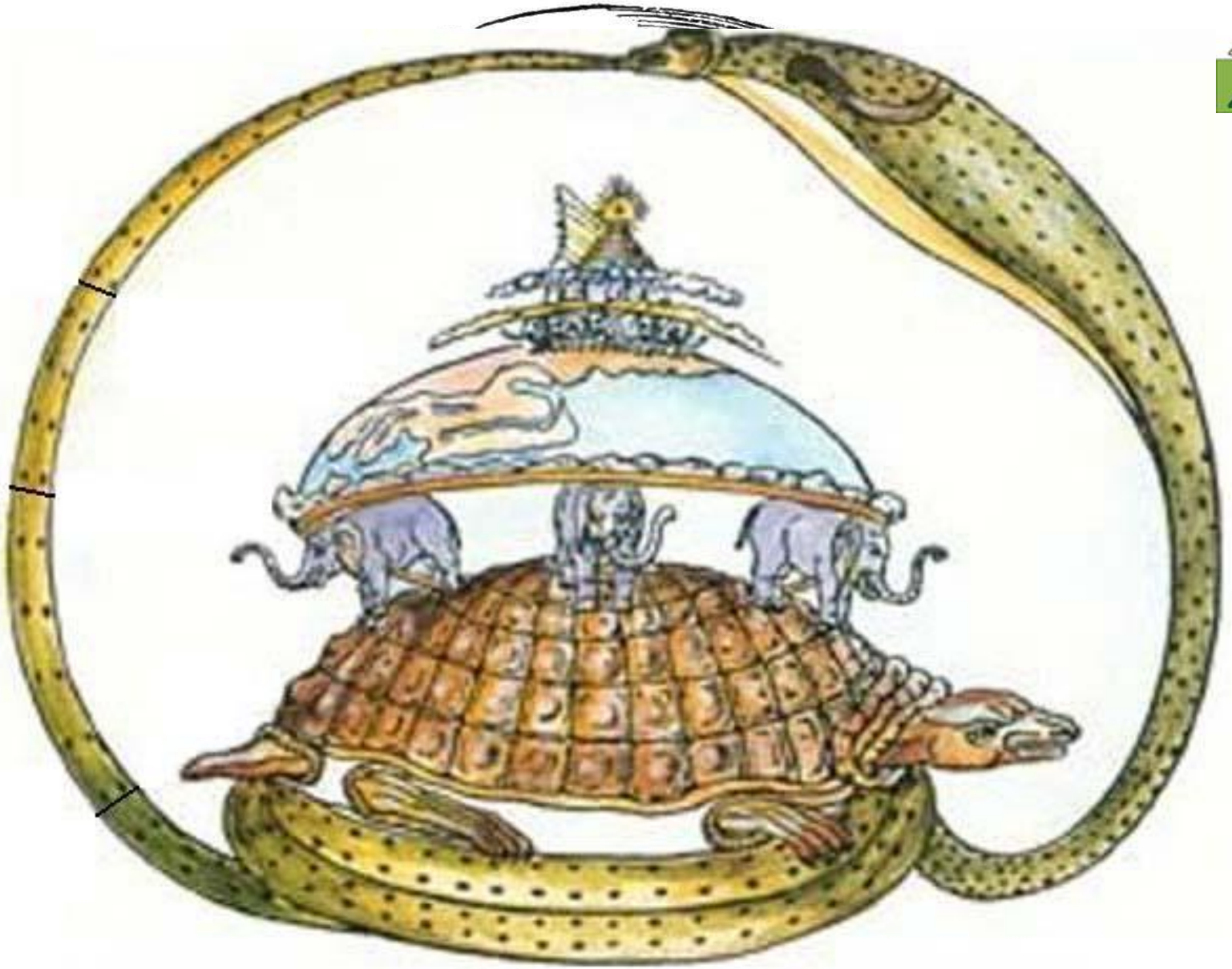
`fixed` A fixed number of 8-bit unsigned bytes.

```
{
  "type": "fixed",
  "name": "Md5Hash",
  "size": 16
}
```

Apache Chuckwa

- LARGE-SCALE LOG COLLECTION AND ANALYSIS
- ALSO USED FOR DISPLAYING MONITORING AND ANALYZING RESULTS
  - To make the best use of this collected data.

Chukwa's Name Origin



Chukwa

- Large-scale log collection and analysis
- Logs are on remote machine
- Hard to collect/access logs from thousand of machines
- Hard to correlate information from different system
- Unable to extract useful information from terabytes of data
- No easy way to detect failures on thousand of machines

Chukwa's Goals

- Collect
  - Arbitrary log files (unknown format)
  - Known log files (well define format)
  - Handle log rotation
  - Latency should be in minutes but not in hours
  - Scale to large cluster
- Store large volume of data–all data in one place
- Advanced log analytics and data mining
- Reporting framework

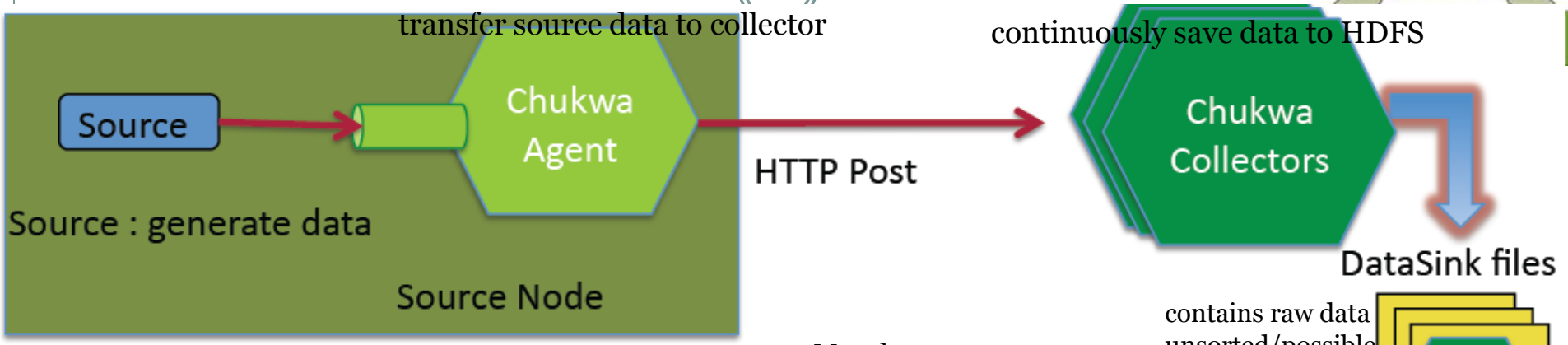
Chukwa is good for

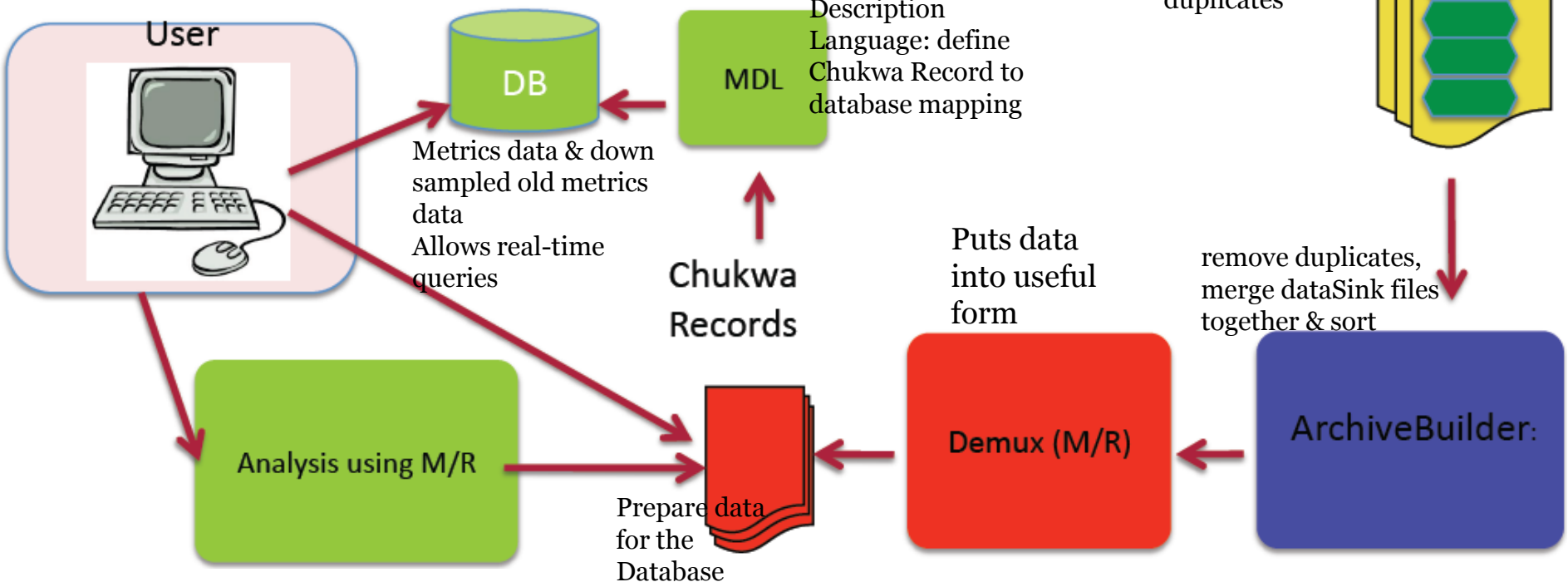
- Collect application logs: log4j integration
- Collect new source of data by implementing the Adaptor interface
- Extract additional information by using an existing parser or by extending or writing your own.

Chukwa Advantages

- Scalable & light log collection pipeline
- Scalable log processing pipeline
- All your data in one place
- Cross system analysis
- Native M/R & Pig integration
- Open source – Apache 2.0
- http://hadoop.apache.org/chukwa/

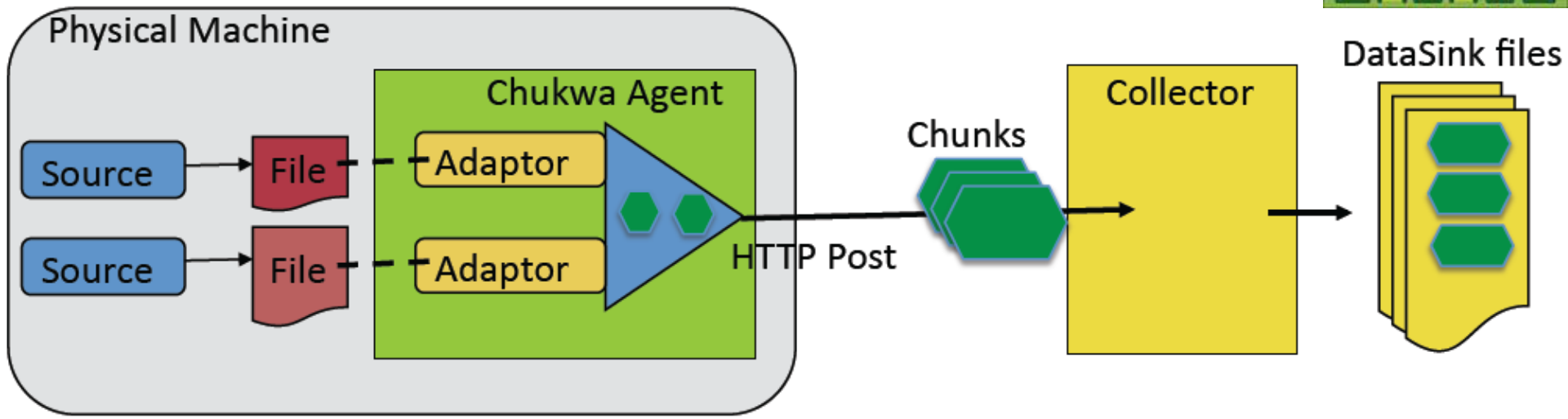
Chukwa Architecture



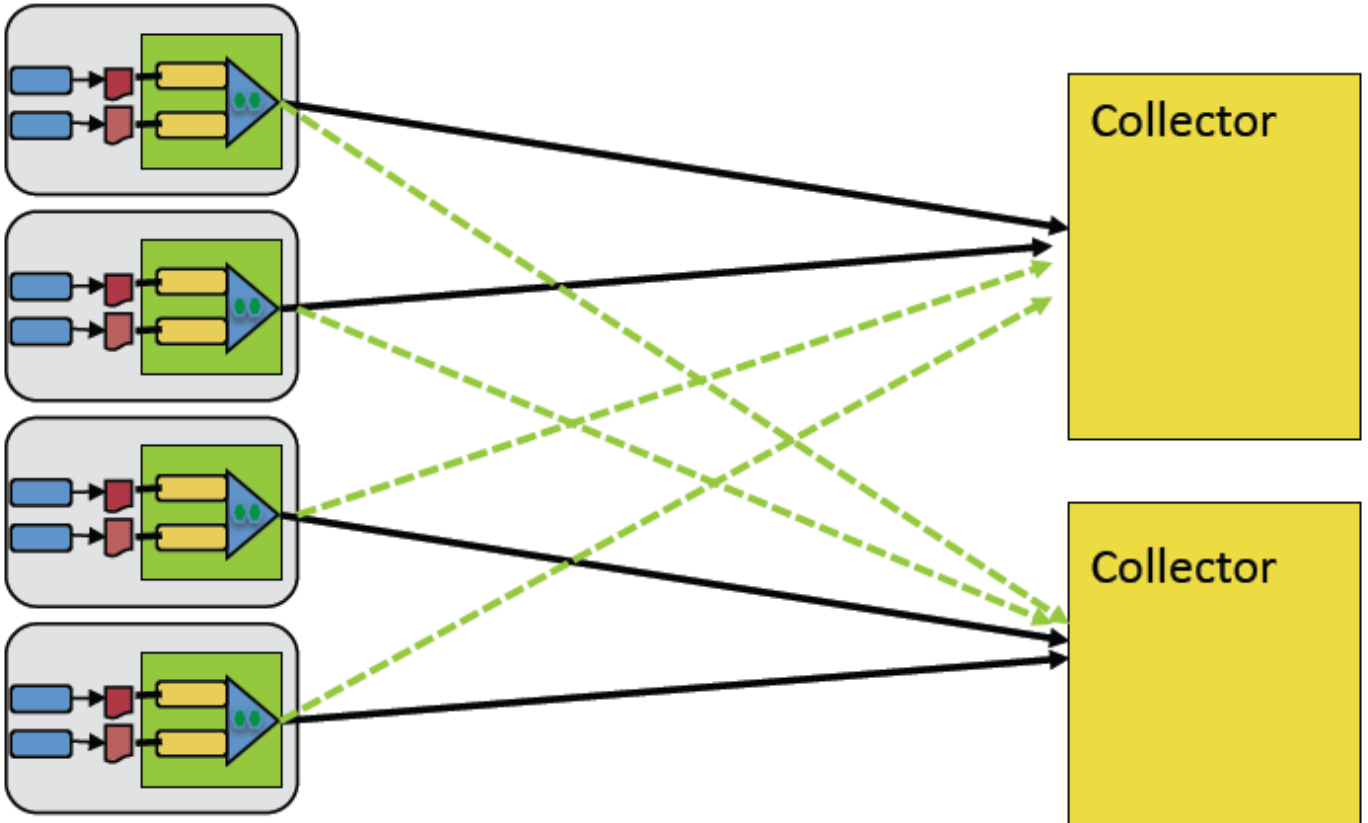


Chukwa Interface

Data Collection by Chukwa Agent



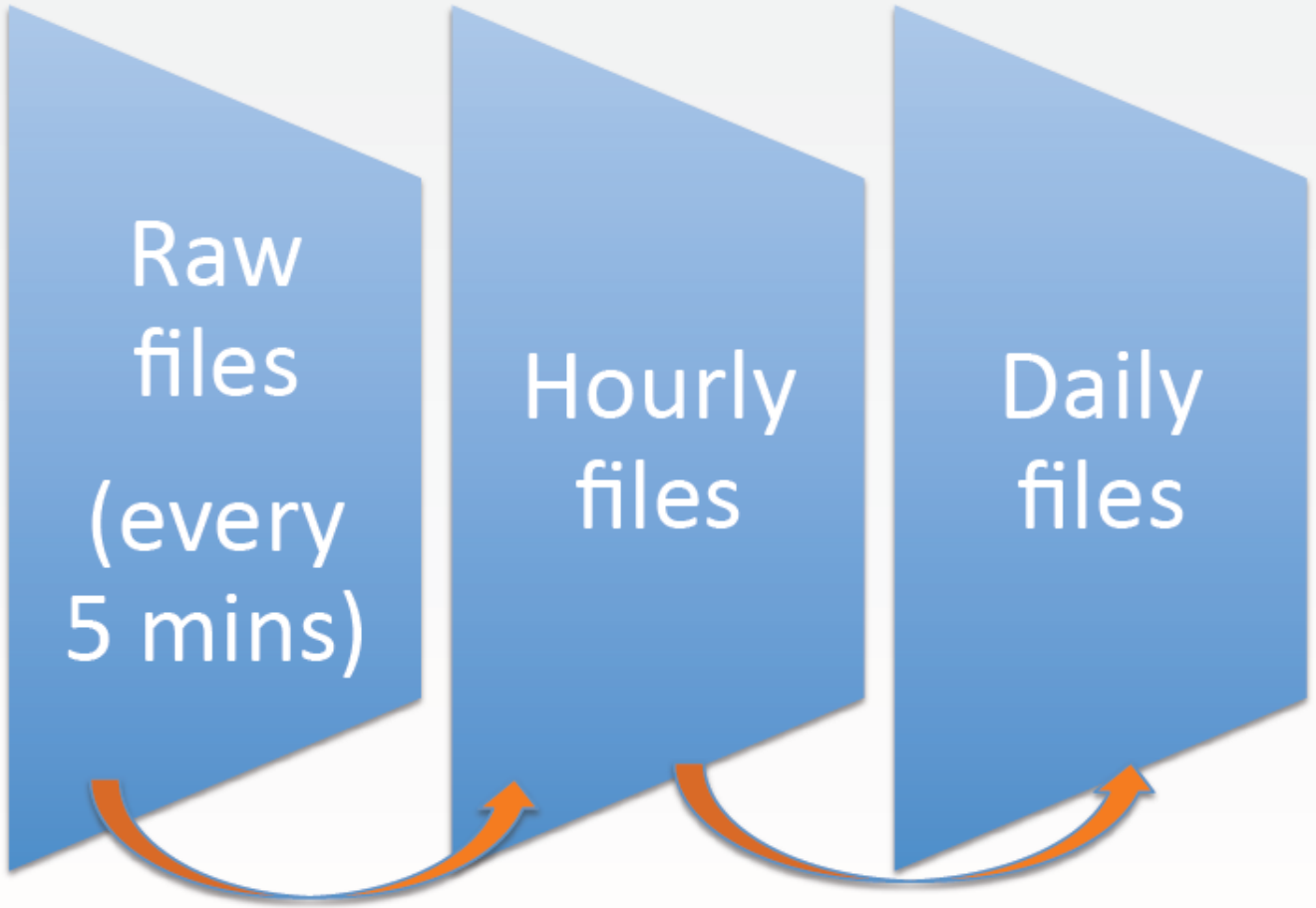
Chukwa Automatic Failover & Load Balancing



In case of error/exception, the agent pick a new collector from a predefined list and resend the same list of chunks

Failover

Chukwa File rotation for Archive



Chukwa Sample Uses

- Admin: Billing, accounting, provisioning
- Development: software log analysis and debugging
- Ops: Hardware failures and performance
- Users: Want estimate of current resources

Apache Hbase

DISTRIBUTED COLUMN-ORIENTED DATA STORE BUILT ON TOP OF HDFS

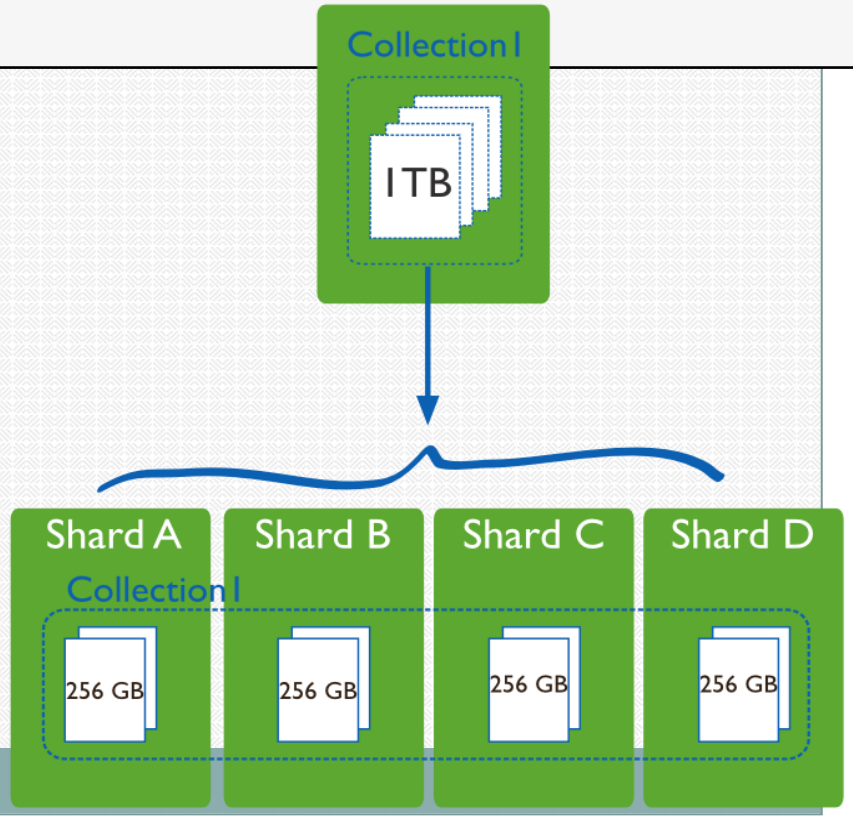
HBase

- Is a distributed column-oriented data store built on top of HDFS
- Provides random, real time access to data in Hadoop
- Was created to store very large tables
- Great for multi-structured or sparse data.
- Possible to query data for a particular point in time
- Pros:
  - Good for semi-structured data
  - Quick access to data

HBase

Characteristic	Benefit
Fault tolerant	•**Replication** across the data center •**Atomic and strongly consistent** row-level operations •**High availability** through automatic failover •**Automatic sharding and load balancing** of tables

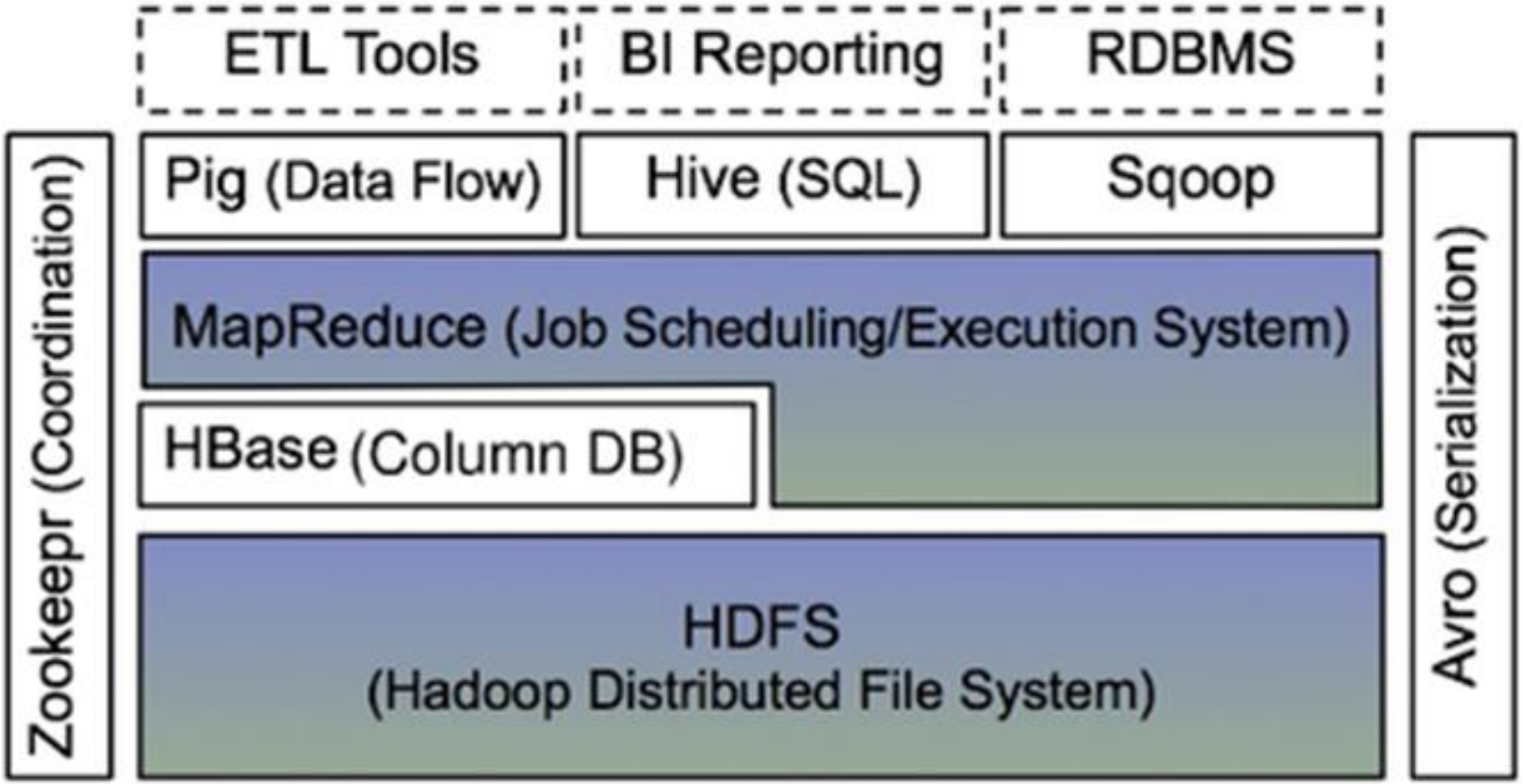
- Sharding is a type of database partitioning
  - Separates very large databases the into smaller, faster, more easily managed parts.
  - Smaller parts are called **data shards**.
  - The word **shard** means **a small part of a whole**.



Characteristic	Benefit
Fast	•**Near real time lookups** •**In-memory caching** via **block cache** and **bloom filters** • **Bloom Filters** provide a lightweight in-memory structure to reduce those N disk reads to only the files likely to contain that Row (N-B). •**Server side processing** via filters and co-processors
Usable	•**Data model** accommodates wide range of use cases •**Metrics exports** via File and Ganglia plugins •**Easy Java API** •Thrift API •REST API

HBase: Part of Hadoop's Ecosystem

- HBase is built on top of HDFS
- HBase files are internally stored in HDFS

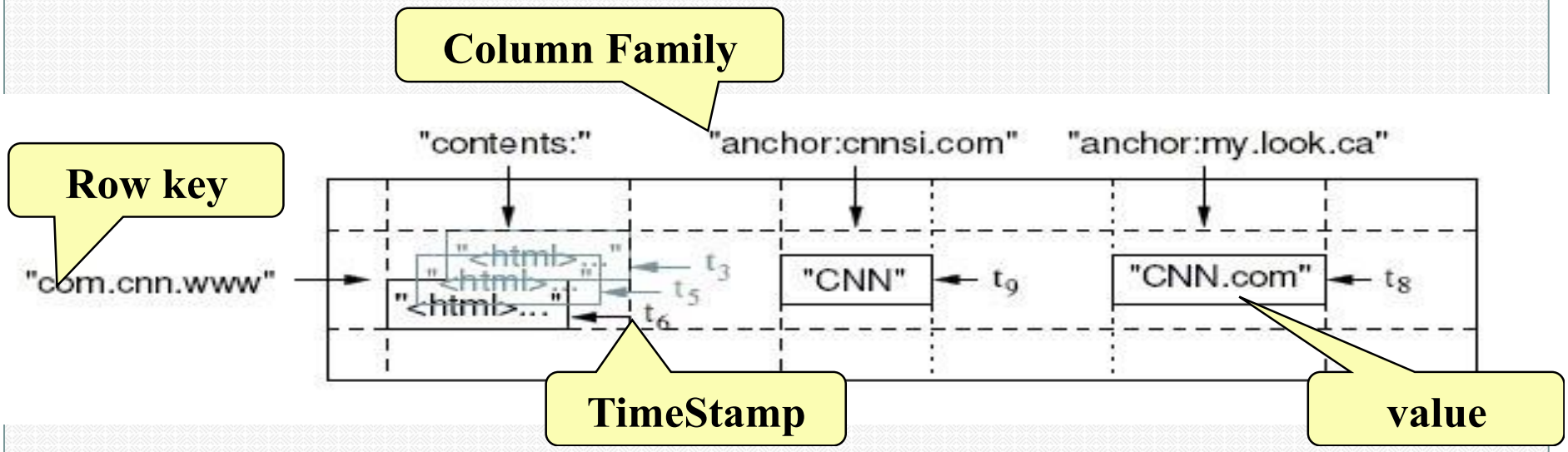


HBase Features

- ***HBase*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)
- HBase updates are done by creating new versions of values

HBase Data Model

- **HBase is based on Google's Bigtable model**
  - **Key-Value pairs**



HBase Logical View. Example.

Table name: webtable

Row Key	Time Stamp	Column Family Contents	ColumnFamily anchor	ColumnFamily people
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"	
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"	
"com.cnn.www"	t6	contents:html = "<html>…"		
"com.cnn.www"	t5	contents:html = "<html>…"		
"com.cnn.www"	t3	contents:html = "<html>…"		

Example continued…

- Table “webtable” contains
  - Two **rows** (com.cnn.www and com.example.www)
  - And three **column families**
    - **contents**, **anchor**, and **people**.
- In the first row (com.cnn.www),
  - Column family **anchor** contains two columns (anchor:cssnsi.com, anchor:my.look.ca)
  - Column family **contents** contains one column (contents:html).

Example continued …

- This example contains:
  - 5 versions of the row with the row key com.cnn.www
  - 1 version of the row with the row key com.example.www
  - Column **contents:html** contains the entire HTML of a given website.
  - Column family **anchor** contains 2 columns with:
    - link to an external site
    - Attribute of that link
  - **People** column family represents people associated with the site.

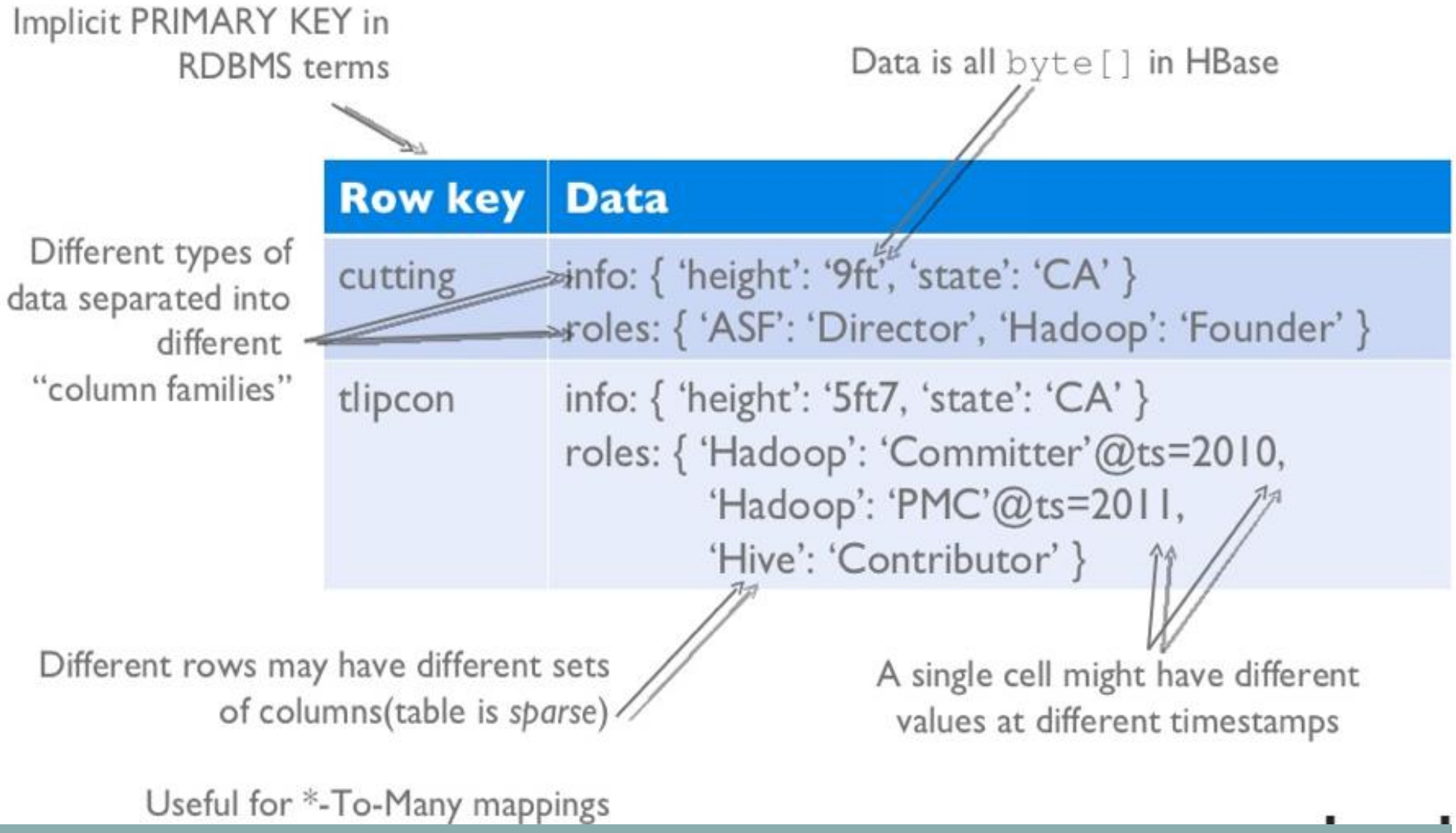
Explanation

- Cells in this table that appear to be empty do not exist in HBase.
- This makes HBase "sparse."
- A tabular view is not the only way to look at data

```
{
  "com.cnn.www": {
    contents: {
      t6: contents:html: "<html>..."
      t5: contents:html: "<html>..."
      t3: contents:html: "<html>..."
    }
    anchor: {
      t9: anchor:cnnsi.com = "CNN"
      t8: anchor:my.look.ca = "CNN.com"
    }
    people: {}
  }
  "com.example.www": {
    contents: {
      t5: contents:html: "<html>..."
    }
    anchor: {}
    people: {
      t5: people:author: "John Doe"
    }
  }
}
```

- This is the same information as in the table
- This is a multidimensional map

HBase Logical View



HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*

PERSON TABLE

row key	personal_data		demographic		...
PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2 - Census Data in Column Families

Each column family consists of one or more *Columns*

APACHE HBASE

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - ***familyName:columnName***

Column family named "Contents"

Column family named "anchor"

Column named "apache.com"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

Version number for each row

- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		value
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Notes on Data Model (Cont'd)

- The ***version number*** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version number are unique within each key
- Table can be very sparse
  - Many cells are empty
- ***Keys*** are indexed as the primary key

Has two columns [cnnsi.com & my.look.ca]

Row Key	Time Stamp	ColumnFamily `contents`	ColumnFamily `anchor`
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

HBase Physical Model

HBase Physical Model

- Each column family is stored in a separate file (called ***HTables***)
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values: *<key, column family, column name, timestamp>*

Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Table 5.2. ColumnFamily `anchor`

Row Key	Time Stamp	Column Family `anchor`
"com.cnn.www"	t9	`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8	`anchor:my.look.ca` = "CNN.com"

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

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

cloudera

Column Families

- Different sets of columns may have different properties and access patterns
- Configurable by column family:
  - Compression (none, gzip, LZO)
  - Version retention policies
  - Cache priority
- CFs stored separately on disk: access one without wasting IO on the other.

HBase Regions

- Each HTable (column family) is partitioned horizontally into ***regions***
  - Regions are counterpart to HDFS blocks

Table 5.3. ColumnFamily `contents`

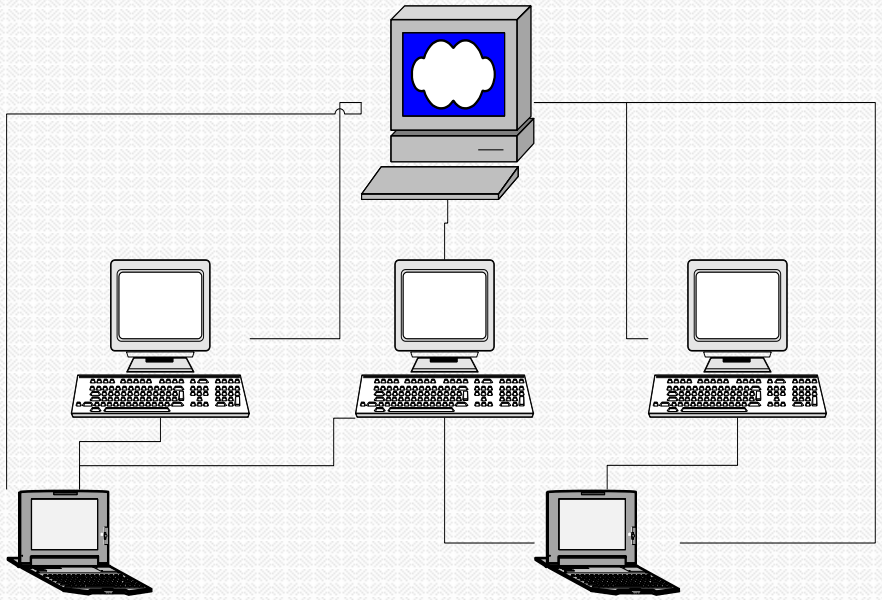
Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Each will be one region

HBase Architecture

Three Major Components

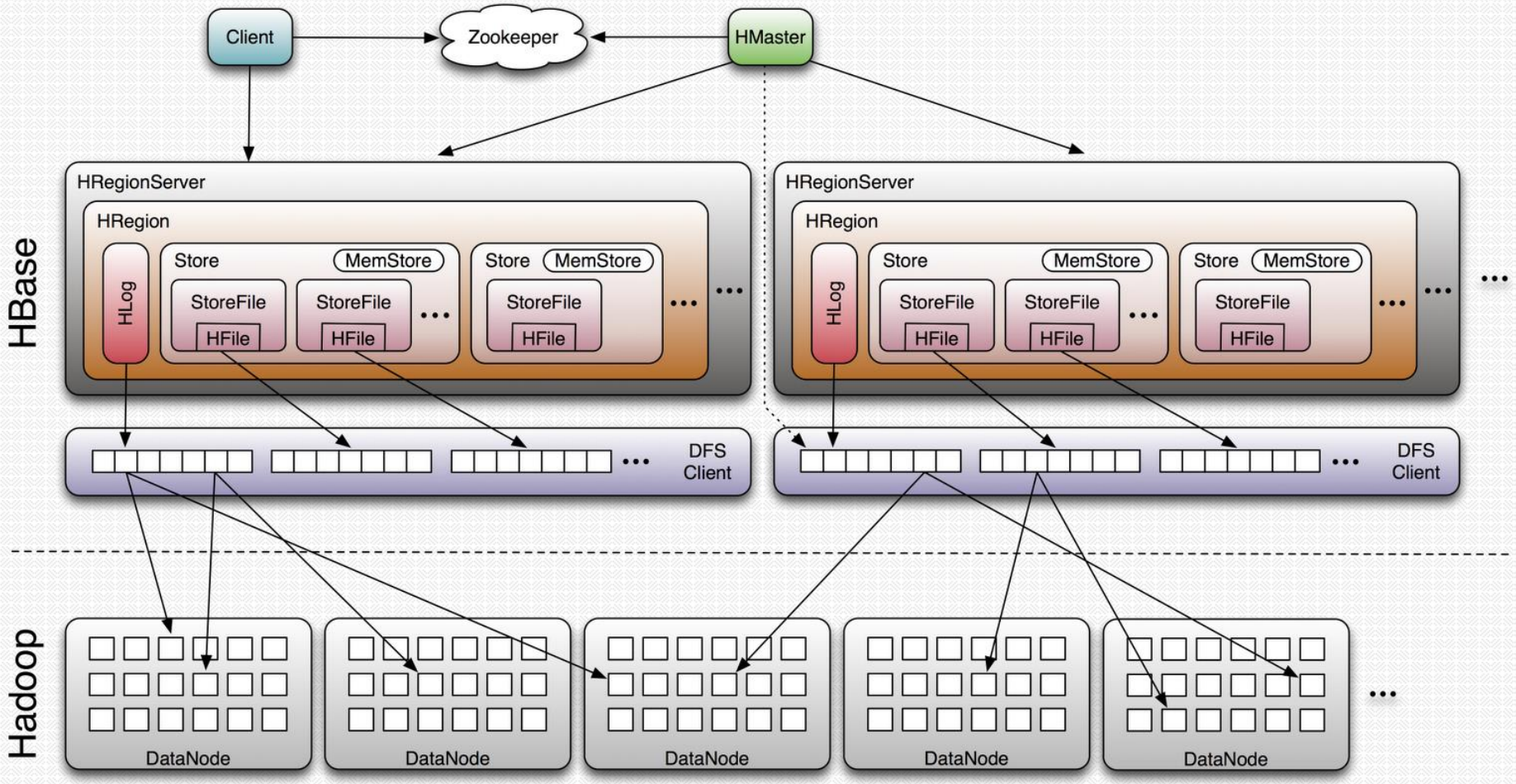
- The HBaseMaster
  - One master
- The HRegionServer
  - Many region servers
- The HBase client



HBase Components

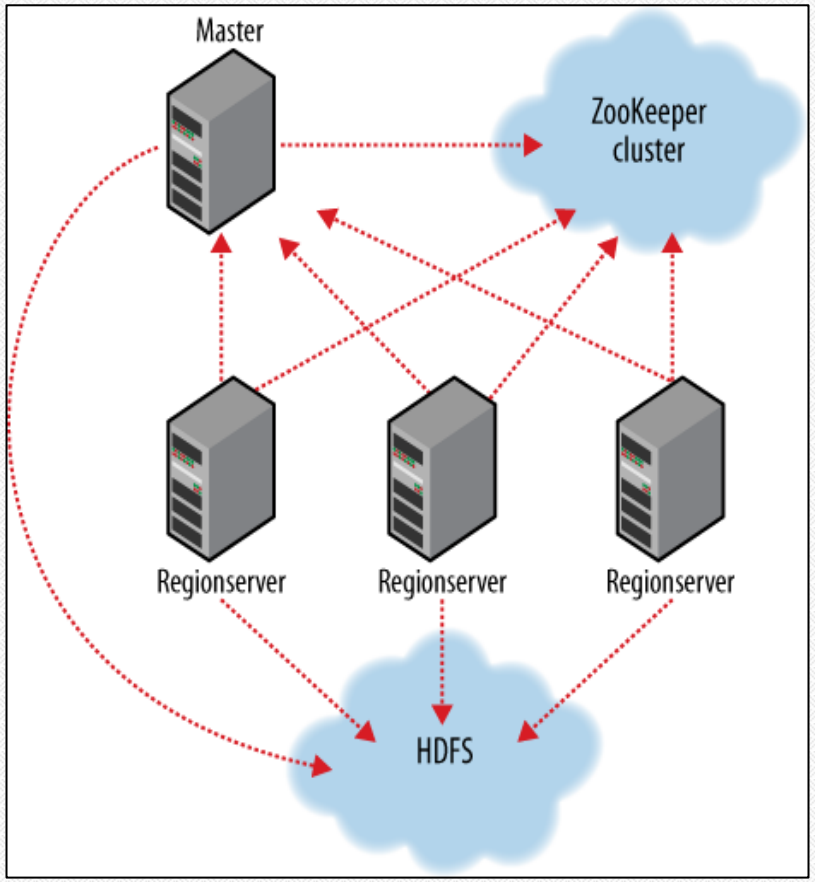
- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (***using a log***)
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

ZooKeeper

- HBase depends on ZooKeeper
- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper
- HMaster and HRegionServers register themselves with ZooKeeper



Creating a Table


```
HBaseAdmin admin= new HBaseAdmin(config);
HColumnDescriptor []column;
column= new HColumnDescriptor[2];
column[0]=new
  HColumnDescriptor("columnFamily1:");
column[1]=new
  HColumnDescriptor("columnFamily2:");
HTableDescriptor desc= new
  HTableDescriptor(Bytes.toBytes("MyTable"));
desc.addFamily(column[0]);
desc.addFamily(column[1]);
admin.createTable(desc);
```

Operations On Regions: Get()

- Given a key → return corresponding record
- For each value return the highest version

```
Get get = new Get(Bytes.toBytes("row1"));
Result r = htable.get(get);
5.8.1.2. Default Get Example ue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

- Can control the number of versions you want

```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of
```

Operations On Regions: Scan()


```
HTable htable = ...      // instantiate HTable

Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));                    // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  (char)0));  // stop key is exclusive
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
```

APACHE HBASE

Get()

Select value from table where key='com.apache.www' AND label='anchor:apache.com'

Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	"APACHE"
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

APACHE HBASE

Scan()

Select value from table where anchor='cnnsi.com'

Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	"APACHE"
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

Operations On Regions: Put()

- Insert a new record (with a new key), Or
- Insert a record for an existing key

Implicit version number (timestamp)

```
Put put = new Put(Bytes.toBytes(row));
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
htable.put(put);
```

Explicit version number

```
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

Operations On Regions: Delete()

- Marking table cells as deleted
- **Multiple levels**
  - Can mark an entire column family as deleted
  - Can make all column families of a given row as deleted

- **All operations are logged by the RegionServers**
- **The log is flushed periodically**

HBase: **Joins**

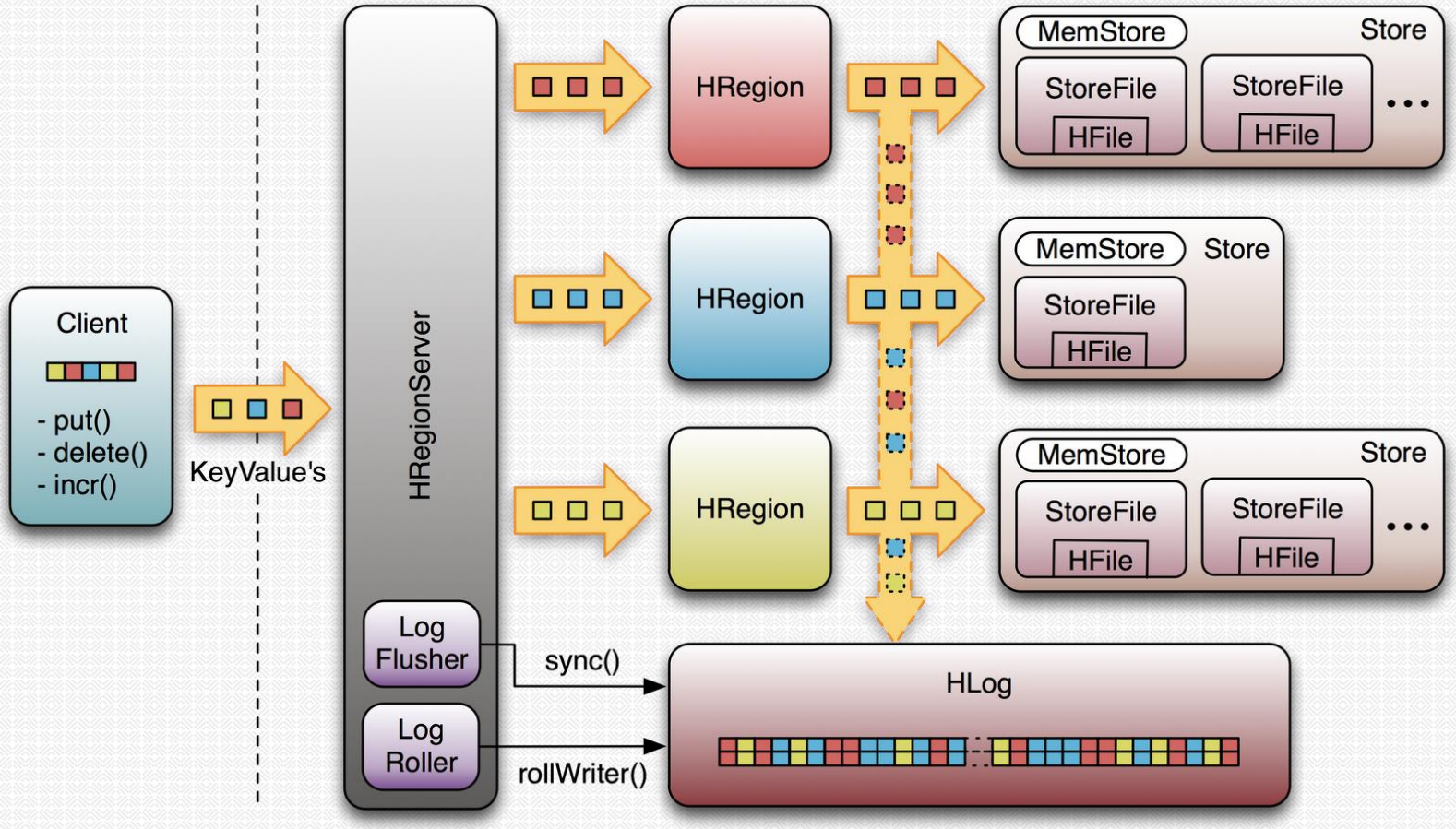
- HBase does not support joins
- Can be done in the application layer
  - Using scan() and get() operations

Altering a Table

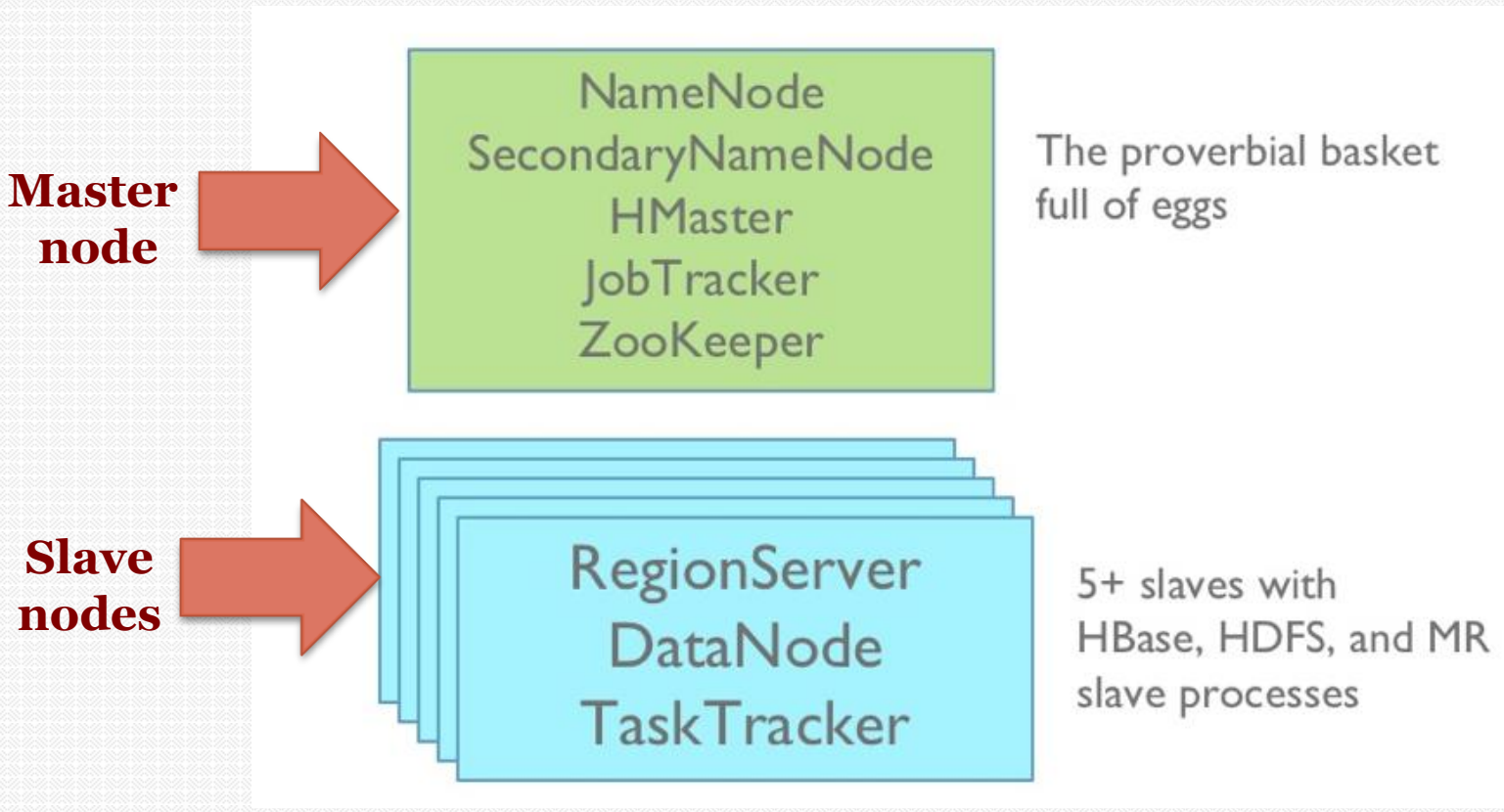

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);      // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);    // modifying existing ColumnFamily

admin.enableTable(table);
```

Disable the table before changing the schema

Logging Operations



HBase Deployment



HBase vs. 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

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

When to use HBase

- You need random write, random read, or both (*but not neither*)
- You need to do many thousands of operations per second on multiple TB of data
- Your access patterns are well-known and simple