

---

# **CS 444: Big Data Systems**

## **Chapter 4. Big Data Storage and Management**

Chase Wu

New Jersey Institute of Technology

Some of the slides were provided through the courtesy of Dr.  
Ching-Yung Lin at Columbia University

## Remind -- Apache Hadoop

---



The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than relying on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

- **Hadoop Common:** The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN:** A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

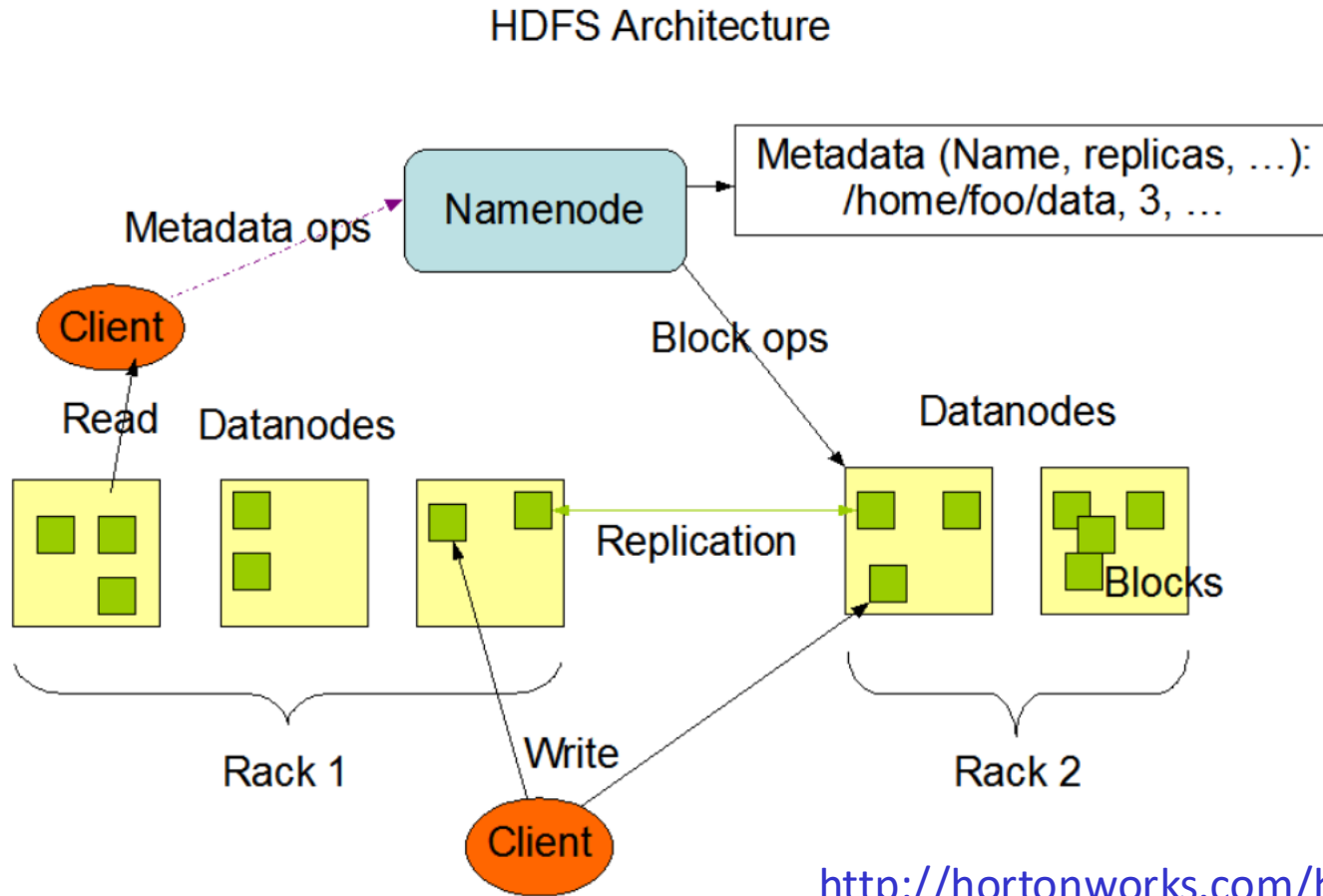
<http://hadoop.apache.org>

## Remind -- Hadoop-related Apache Projects

---

- [Ambari™](#): A web-based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig and Hive applications visually.
- [Avro™](#): A data serialization system.
- [Cassandra™](#): A scalable multi-master database with no single points of failure.
- [Chukwa™](#): A data collection system for managing large distributed systems.
- [HBase™](#): A scalable, distributed database that supports structured data storage for large tables.
- [Hive™](#): A data warehouse infrastructure that provides data summarization and ad hoc querying.
- [Mahout™](#): A Scalable machine learning and data mining library.
- [Pig™](#): A high-level data-flow language and execution framework for parallel computation.
- [Spark™](#): A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- [Tez™](#): A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- [ZooKeeper™](#): A high-performance coordination service for distributed applications.

# Remind -- Hadoop Distributed File System (HDFS)



<http://hortonworks.com/hadoop/hdfs/>

- **Namenode:** This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
- **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

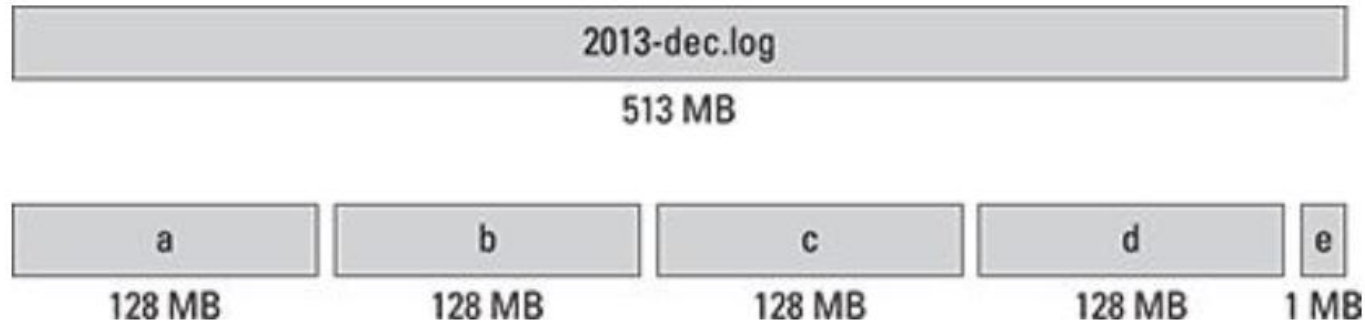
# Data Storage Operations on HDFS

---

- Hadoop is designed to work best with a modest number of extremely large files.
- Average file sizes → larger than 500MB.
- **Write Once, Read Often** model.
- Content of individual files cannot be modified, other than appending new data at the end of the file.
- What we can do:
  - Create a new file
  - Append contents to the end of a file
  - Delete a file
  - Rename a file
  - Modify file attributes like owner

# HDFS blocks

- File is divided into blocks (default: 64MB in Hadoop 1 and 128 MB in Hadoop 2) and duplicated in multiple places (default: 3, which could be changed to the required values according to the requirement by editing the configuration files `hdfs-site.xml`)



- Dividing into blocks is normal for a native file system, e.g., the default block size in Linux is 4KB. The difference of HDFS is the scale.
- Hadoop was designed to operate at the petabyte scale.
- Every data block stored in HDFS has its own metadata and needs to be tracked by a central server.

## Why replicate? How to set the replication number?

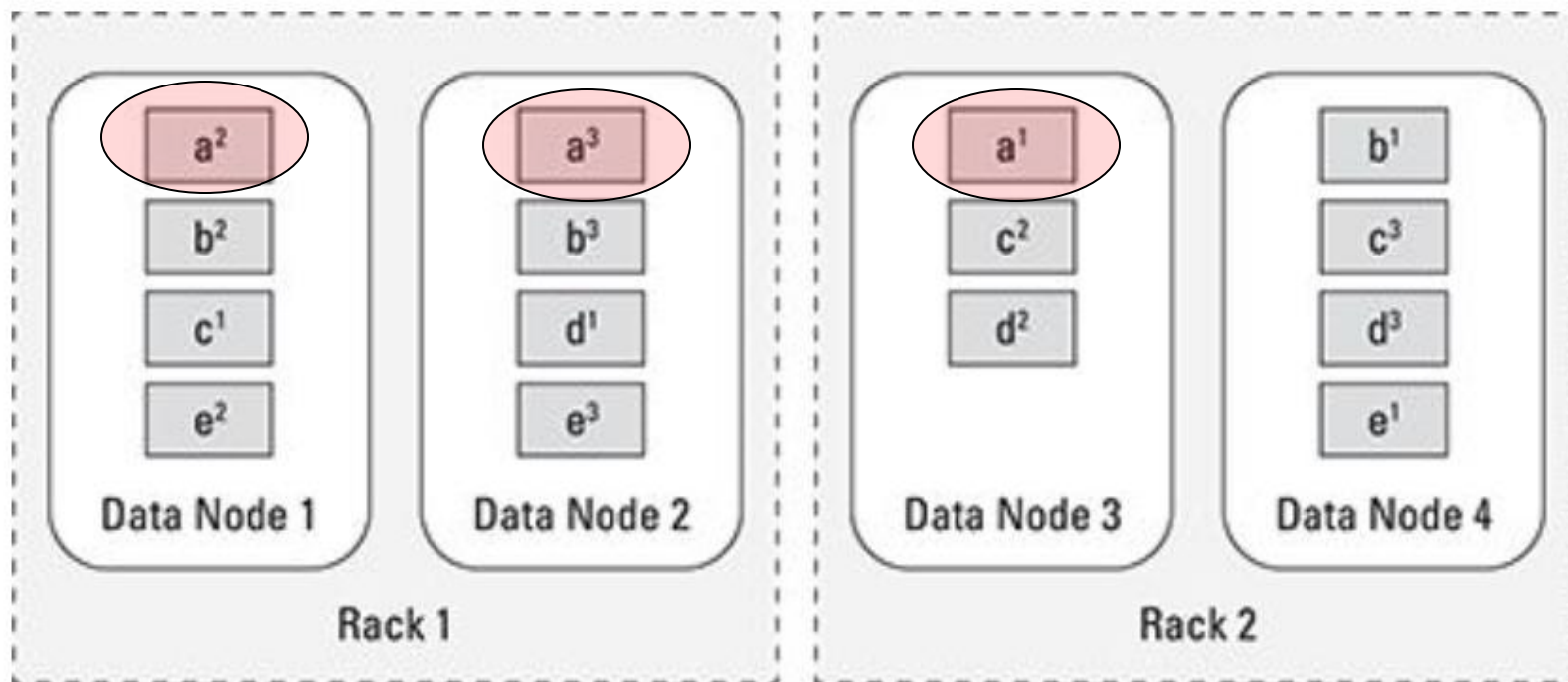
- Reliability
- Performance

## How to set the block size properly?

- Block in HDFS vs. sector in native file system
- Degree of parallelism

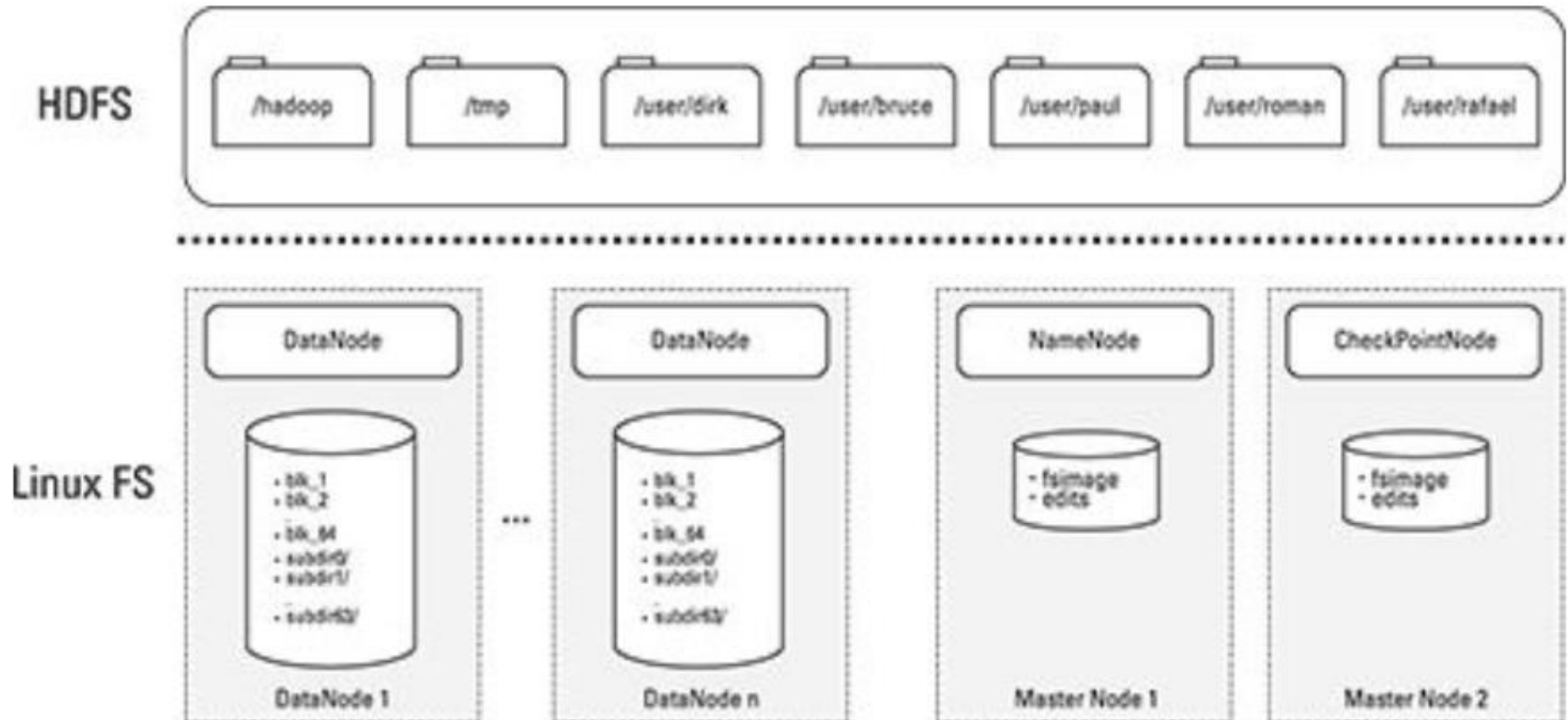
# HDFS block replica placement

- Replication patterns of data blocks in HDFS.

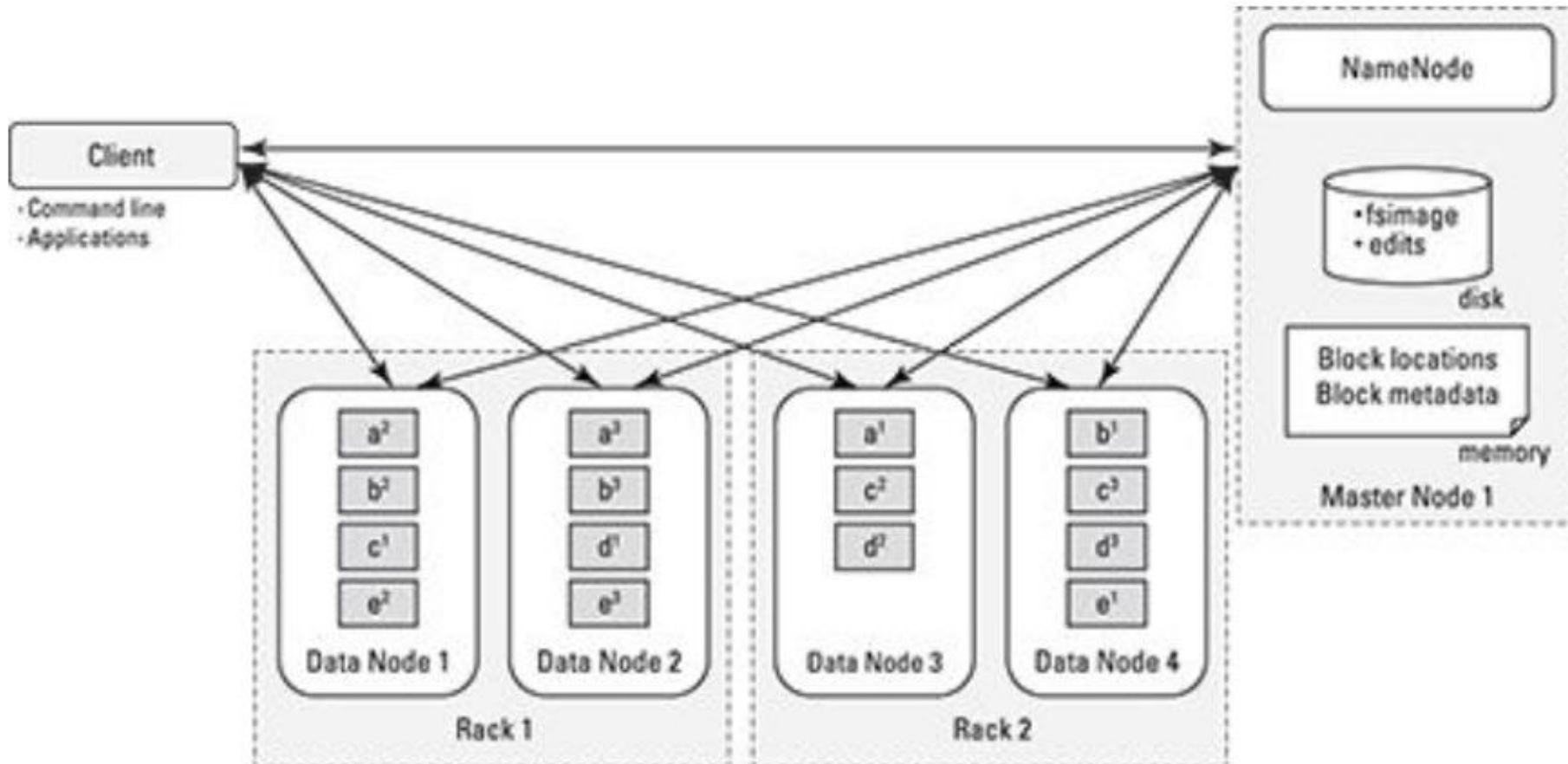


- When HDFS stores the replicas of the original blocks across the Hadoop cluster, it tries to ensure that the block replicas are stored at different failure points.
- Rack-aware replica placement** to improve data reliability, availability, and network bandwidth utilization, in a hierarchical (multi-rack, multi-node) architecture
  - NameNode places replicas of a block on multiple racks for improved fault tolerance.
    - tries to place at least one replica of a block in each rack, so that if a complete rack goes down, the system will be still available on other racks.

# HDFS is a User-Space-Level file system

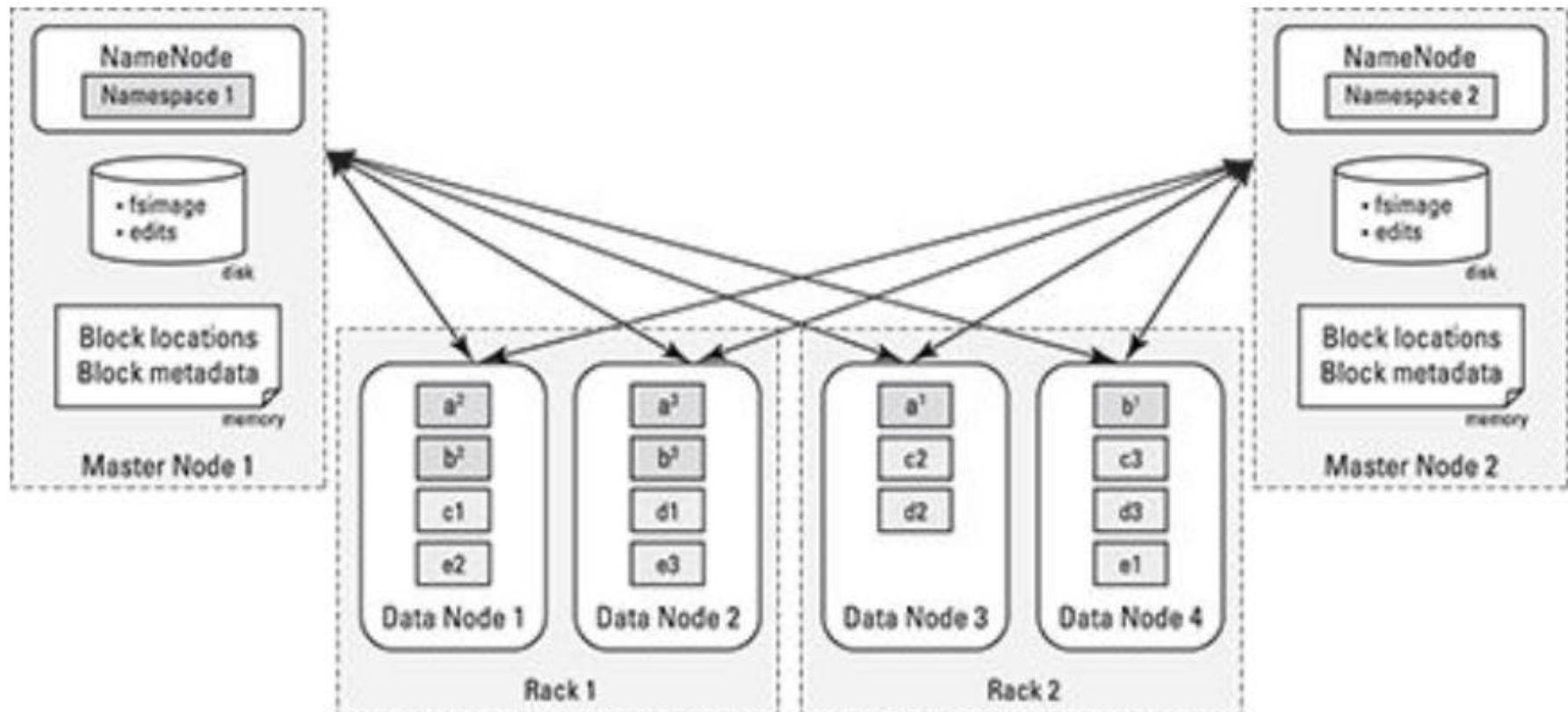


# Interaction between HDFS components



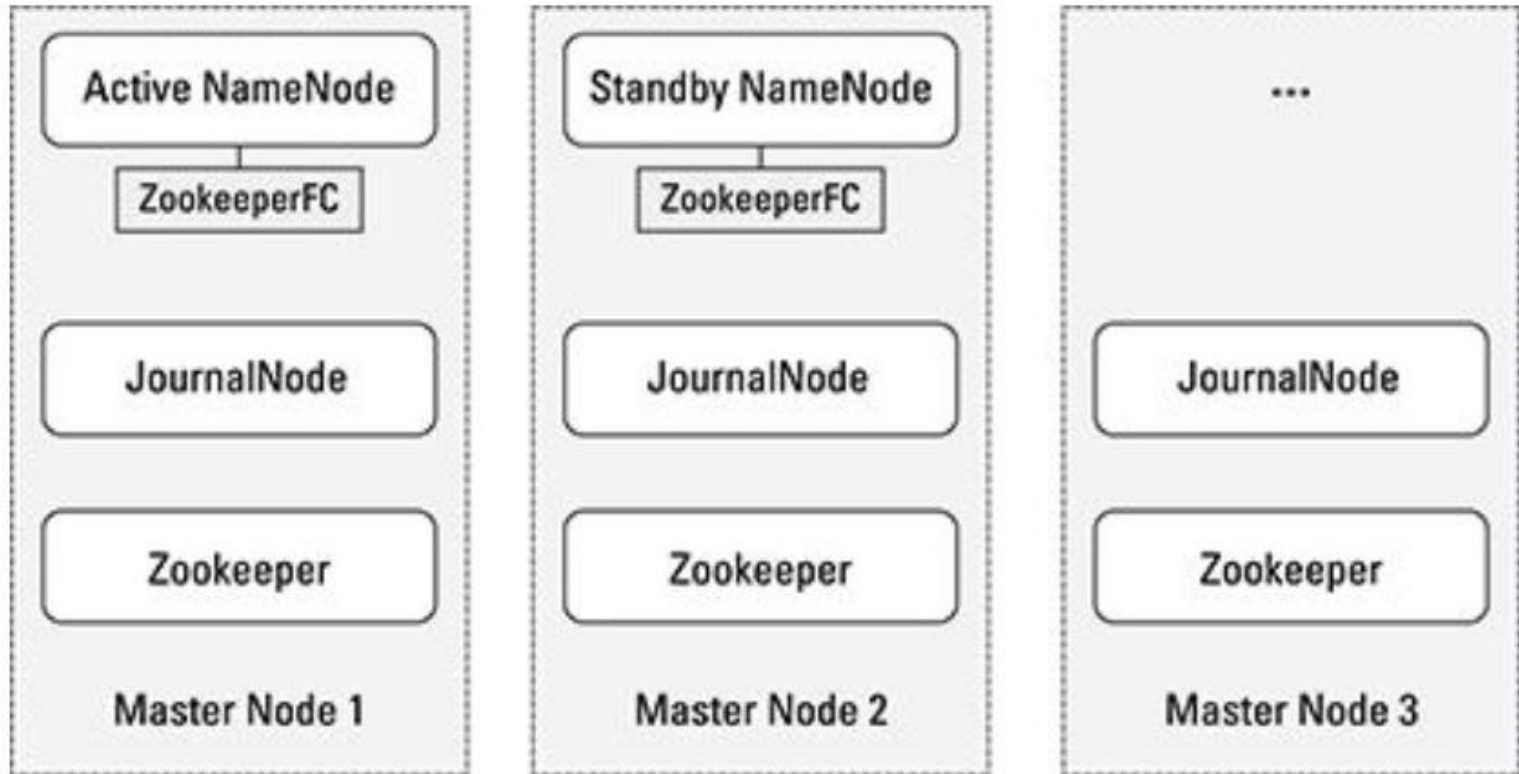
# HDFS Federation

- Before Hadoop 2.0
  - NameNode was a single point of failure and operation limitation.
  - Hadoop clusters could hardly scale beyond 3,000 or 4,000 nodes.
- In Hadoop 2.x
  - Multiple NameNodes can be used (HDFS High Availability feature – one is in an **Active** state, and the other one is in a **Standby** state).



## High Availability of the NameNodes

- Active NameNode
- Standby NameNode – keeping the state of the block locations and block metadata in memory -> HDFS checkpointing responsibilities.



- **JournalNode** – if a failure occurs, the Standby Node reads all completed journal entries to ensure the new Active NameNode is fully consistent with the state of the cluster.
- **Zookeeper** – provides coordination and configuration services for distributed systems.

**Table 5-1**

**Hadoop Codecs**

<i>Codec</i>	<i>File Extension</i>	<i>Splittable?</i>	<i>Degree of Compression</i>	<i>Compression Speed</i>
Gzip	.gz	No	Medium	Medium
Bzip2	.bz2	Yes	High	Slow
Snappy	.snappy	No	Medium	Fast
LZO	.lzo	No, unless indexed	Medium	Fast

# HDFS Commands

---

- All Hadoop commands are invoked by the bin/hadoop script.

```
hadoop [--config confdir] [COMMAND]
[GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

- **% `hadoop fsck / -files -blocks`:**  
➔ list the blocks that make up each file in HDFS.
- For HDFS, the schema name is hdfs, and for the local file system, the schema name is file.
- A file or directory in HDFS can be specified in a fully qualified way, such as:  
hdfs://namenodehost/parent/child or hdfs://namenodehost
- The HDFS file system shell command is similar to Linux file commands, with the following general syntax: ***hadoop fs or hdfs dfs -file\_cmd***
- For instance, mkdir runs as:  
\$hadoop fs -mkdir /user/directory\_name

## Some useful commands for HDFS

Many are identical to Linux commands.

COMMAND	DESCRIPTION
-ls	List files with permissions and other details
-mkdir	Creates a directory named path in HDFS
-rm	To Remove File or a Directory
-rmr	Removes the file that identified by path / Folder and subfolders
-rmdir	Delete a directory
-put	Upload a file / Folder from the local disk to HDFS
-cat	Display the contents for a file
-du	Shows the size of the file on hdfs.
-dus	Directory/file of total size
-get	Store file / Folder from HDFS to local file
-getmerge	Merge Multiple Files in an HDFS
-count	Count number of directory, number of files and file size
-setrep	Changes the replication factor of a file
-mv	HDFS Command to move files from source to destination
-moveFromLocal	Move file / Folder from local disk to HDFS
-moveToLocal	Move a File to HDFS from Local
-cp	Copy files from source to destination
-tail	Displays last kilobyte of the file
-touch	create, change and modify timestamps of a file
-touchz	Create a new file on HDFS with size 0 bytes
-appendToFile	Appends the content to the file which is present on HDF
-copyFromLocal	Copy file from local file system
-copyToLocal	Copy files from HDFS to local file system
-usage	Return the Help for Individual Command
-checksum	Returns the checksum information of a file
-chgrp	Change group association of files/change the group of a file or a path
-chmod	Change the permissions of a file
-chown	change the owner and group of a file
-df	Displays free space
-head	Displays first kilobyte of the file
-Create Snapshots	Create a snapshot of a snapshottable directory
-Delete Snapshots	Delete a snapshot of from a snapshottable directory
-Rename Snapshots	Rename a snapshot
-expunge	create new checkpoint
-Stat	Print statistics about the file/directory
-truncate	Truncate all files that match the specified file pattern to the specified length
-find	Find File Size in HDFS

# Big Data Management (File- and System-based)

---

- File-based Data Management: store encoded data in a single file (encoding format)
  - JSON
  - XML
  - CSV
  - Hierarchical Data Format (HDF4/5)
  - Network Common Data Form (netCDF)

# Big Data Management (File- and System-based)

---

- System-based Data Management (NoSQL Database)
  - RDBMS
    - SQL/MySQL for highly-structured (table) data
  - Key-Value Store
    - The most primary unit for building other NoSQL databases
  - Document Store
    - **mongoDB** for unstructured, dynamic data
  - Tabular Store
    - **HBase** for table-structured data
    - Cassandra for semi-structured data (each row can have different columns)
  - Object Database
  - Graph Database
    - Property graphs
    - Resource Description Framework (RDF) graphs

# Commonly Used Data Encoding Formats

## JSON (JavaScript Object Notation, .json)

- An open-standard language-independent data format
- Use text to transmit data objects: **attribute-value pairs** and array data types
- Used for asynchronous browser-server communication

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "gender": {
    "type": "male"
  }
}
```

# Commonly Used Data Encoding Formats

---

## XML (Extensible Markup Language, .xml)

- Use **tag pairs** to describe structured data and to serialize objects
- XML supports comments, but JSON does not

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber>
    <type>home</type>
    <number>212 555-1234</number>
  </phoneNumber>
  <phoneNumber>
    <type>fax</type>
    <number>646 555-4567</number>
  </phoneNumber>
  <gender>
    <type>male</type>
  </gender>
</person>
```

# Commonly Used Data Encoding Formats

## CSV (Comma-Separated Values, .csv)

- A delimited data format
- Fields/columns are separated by the **comma** character
- Records/rows are terminated by **newlines**
- All records have **the same number of fields in the same order**
- Any field may be quoted

Year	Make	Model	Description	Price
1997	Ford	E350	ac, abs, moon	3000.00
1999	Chevy	Venture "Extended Edition"		4900.00
1999	Chevy	Venture "Extended Edition, Very Large"		5000.00
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.00

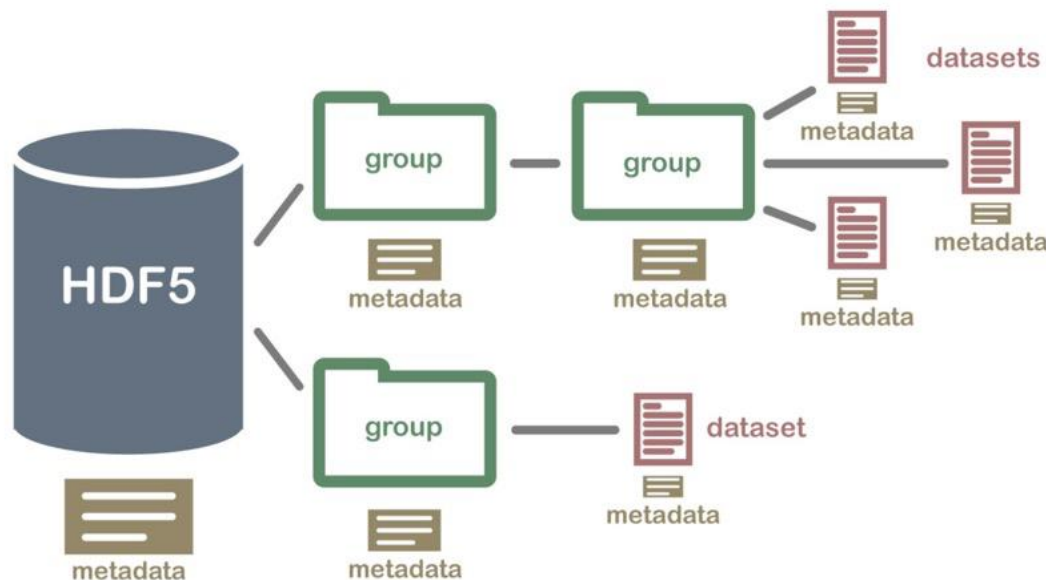
The above table of data may be represented in CSV format as follows:

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""",",",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""",",",5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

# Commonly Used Data Encoding Formats

## Hierarchical Data Format (HDF4/5, .hdf)

- A set of file formats (**HDF4**, **HDF5**) designed to store and organize large amounts of data
- Supported by many commercial and non-commercial software platforms, including Java, MATLAB, Scilab, Octave, Mathematica, IDL, Python, R, Fortran, and Julia.
- HDF5 simplifies the file structure to include only two major types
  - **Datasets**, which are **multidimensional arrays of a homogeneous type**
  - **Groups**, which are container structures that can hold datasets and other groups

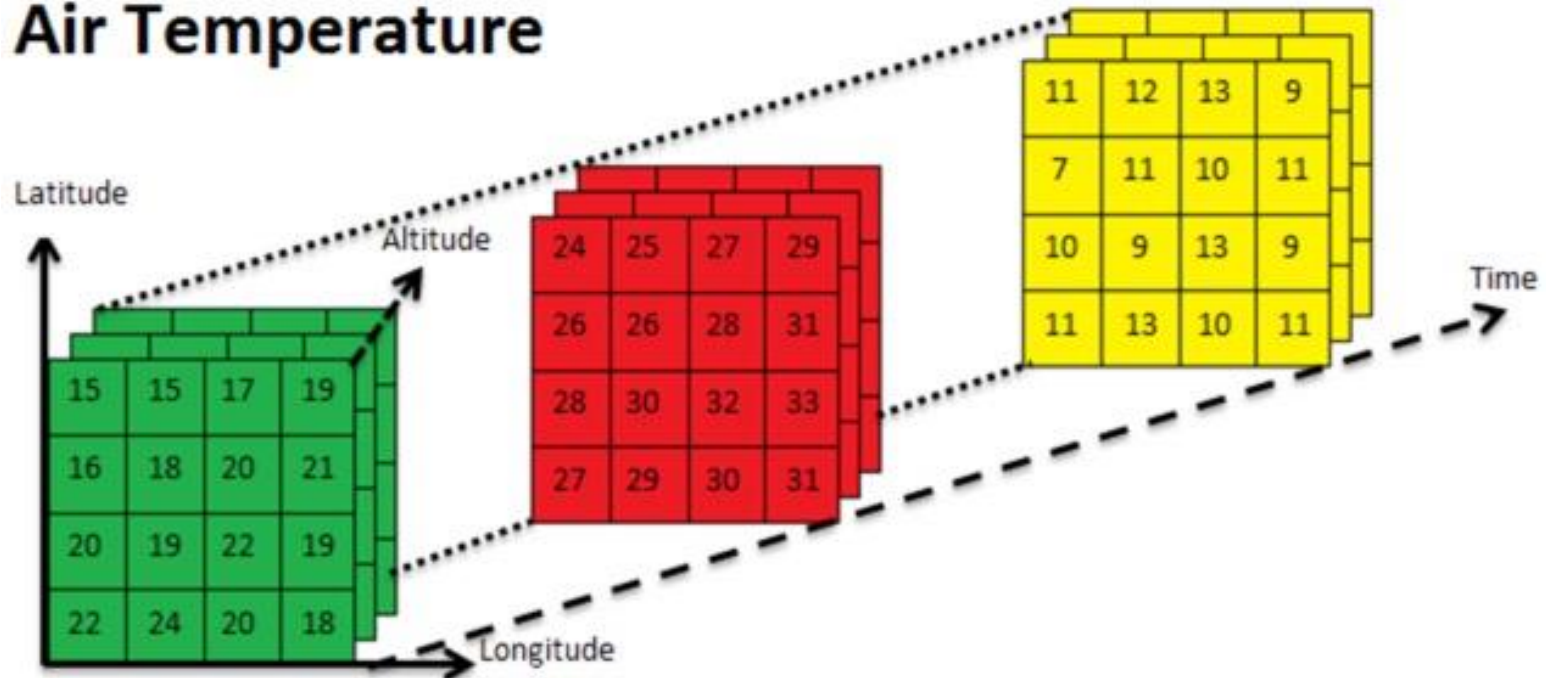


# Commonly Used Data Encoding Formats

## Network Common Data Form (netCDF, .nc)

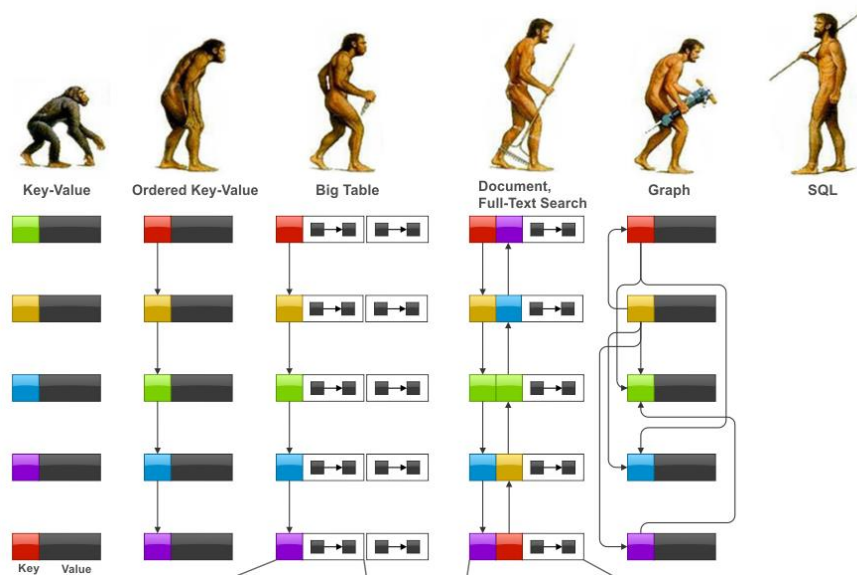
- A set of self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data
- Starting with version 4.0, the netCDF API allows the use of the HDF5 data format
- An extension of netCDF for parallel computing called Parallel-NetCDF (or PnetCDF) has been developed by Argonne National Laboratory and Northwestern University

### Air Temperature



# System-based NoSQL Databases: Key-Value Store

- Considered as **the most primary and the simplest version of all NoSQL databases**
- Use a one-way mapping from the key to the value to store a basic data item



Only provide some simple operations:

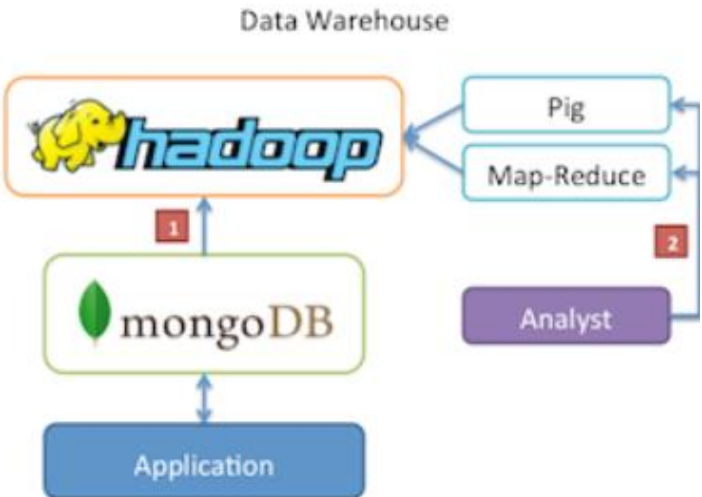
- *Get(key)*, which returns the value associated with the provided *key*.
- *Put(key, value)*, which associates the *value* with the *key*.
- *Multi-get(key1, key2, ..., keyN)*, which returns the list of values associated with the list of *keys*.
- *Delete(key)*, which removes the entry for the *key* from the data store.

## Example Data Represented in a Key-Value Store

Key	Value
...	
"BMW"	{"1-Series", "3-Series", "5-Series", "5-Series GT", "7-Series", "X3", "X5", "X6", "Z4"}
"Buick"	{"Enclave", "LaCrosse", "Lucerne", "Regal"}
"Cadillac"	{"CTS", "DTS", "Escalade", "Escalade ESV", "Escalade EXT", "SRX", "STS"}
...	

# NoSQL: Document Store

The following diagram highlights the components of a MongoDB insert operation:



```

db.users.insert ( ← collection
{
  name: "sue", ← field: value
  age: 26, ← field: value
  status: "A" ← field: value
}
)
    
```

} document

The components of a MongoDB insert operations.

The following diagram shows the same query in SQL:

```

INSERT INTO users ← table
( name, age, status ) ← columns
VALUES ( "sue", 26, "A" ) ← values/row
    
```

The components of a SQL INSERT statement.



## Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.



## Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

# NoSQL: Graph Database

## Graph Models

- Labeled-Property Graphs
  - Represented by a set of nodes, relationships, properties, and labels
  - Both nodes of data and their relationships are named and can store properties represented by key/value pairs
- RDF (Resource Description Framework: Triplestore) Graphs



Apache TinkerPop™ is a graph computing framework for both graph databases (OLTP: Online Transactional Processing) and graph analytic systems (OLAP: Online Analytical Processing).

## Amazon Neptune

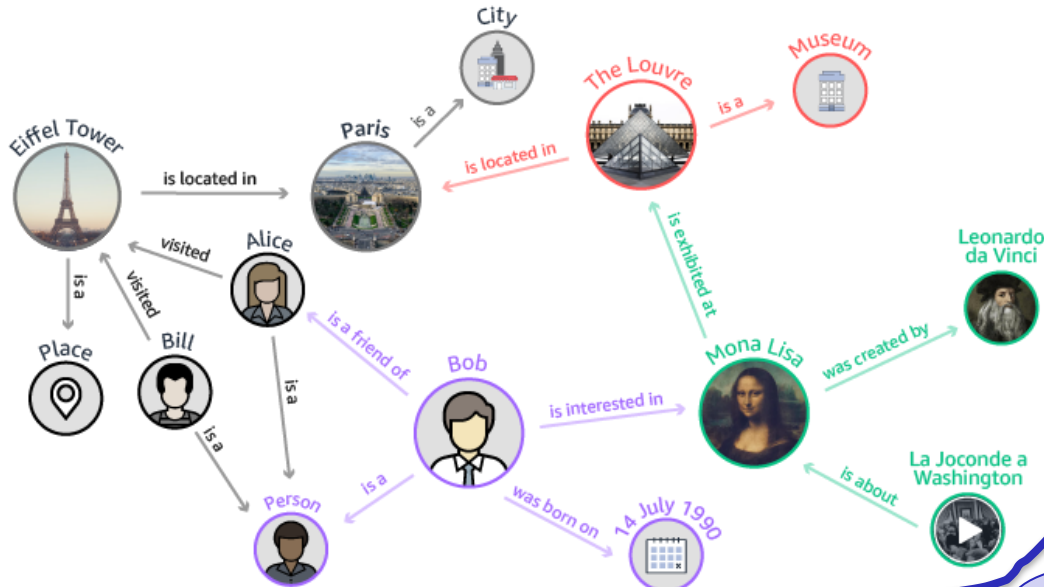
Fast, reliable graph database built for the cloud

- [ArangoDB](#) - OLTP Provider for ArangoDB.
- [Bitsy](#) - A small, fast, embeddable, durable in-memory graph database.
- [Blazegraph](#) - RDF graph database with OLTP support.
- [CosmosDB](#) - Microsoft's distributed OLTP graph database.
- [ChronoGraph](#) - A versioned graph database.
- [DSEGraph](#) - DataStax graph database with OLTP and OLAP support.
- [GRAKN.AI](#) - Distributed OLTP/OLAP knowledge graph system.
- [Hadoop \(Spark\)](#) - OLAP graph processor using Spark.
- [HGraphDB](#) - OLTP graph database running on Apache HBase.
- [Huawei Graph Engine Service](#) - Fully-managed, distributed, at-scale graph query/analysis service that provides a visualized interactive analytics platform.
- [IBM Graph](#) - OLTP graph database as a service.
- [JanusGraph](#) - Distributed OLTP and OLAP graph database with BerkeleyDB, Apache Cassandra and Apache HBase support.
- [JanusGraph \(Amazon\)](#) - The Amazon DynamoDB Storage Backend for JanusGraph.
- [Neo4j](#) - OLTP graph database (embedded and high availability).
- [neo4j-gremlin-bolt](#) - OLTP graph database (using Bolt Protocol).
- [OrientDB](#) - OLTP graph database
- [Apache S2Graph](#) - OLTP graph database running on Apache HBase.
- [Sqlg](#) - OLTP implementation on SQL databases.
- [Stardog](#) - RDF graph database with OLTP and OLAP support.
- [TinkerGraph](#) - In-memory OLTP and OLAP reference implementation.
- [Titan](#) - Distributed OLTP and OLAP graph database with BerkeleyDB, Apache Cassandra and Apache HBase support.
- [Titan \(Amazon\)](#) - The Amazon DynamoDB storage backend for Titan.
- [Titan \(Tupl\)](#) - The Tupl storage backend for Titan.
- [Unipop](#) - OLTP Elasticsearch and JDBC backed graph.

### How is a graph stored?

- Linked list
- Adjacency matrix

# NoSQL: Graph Database Use Cases



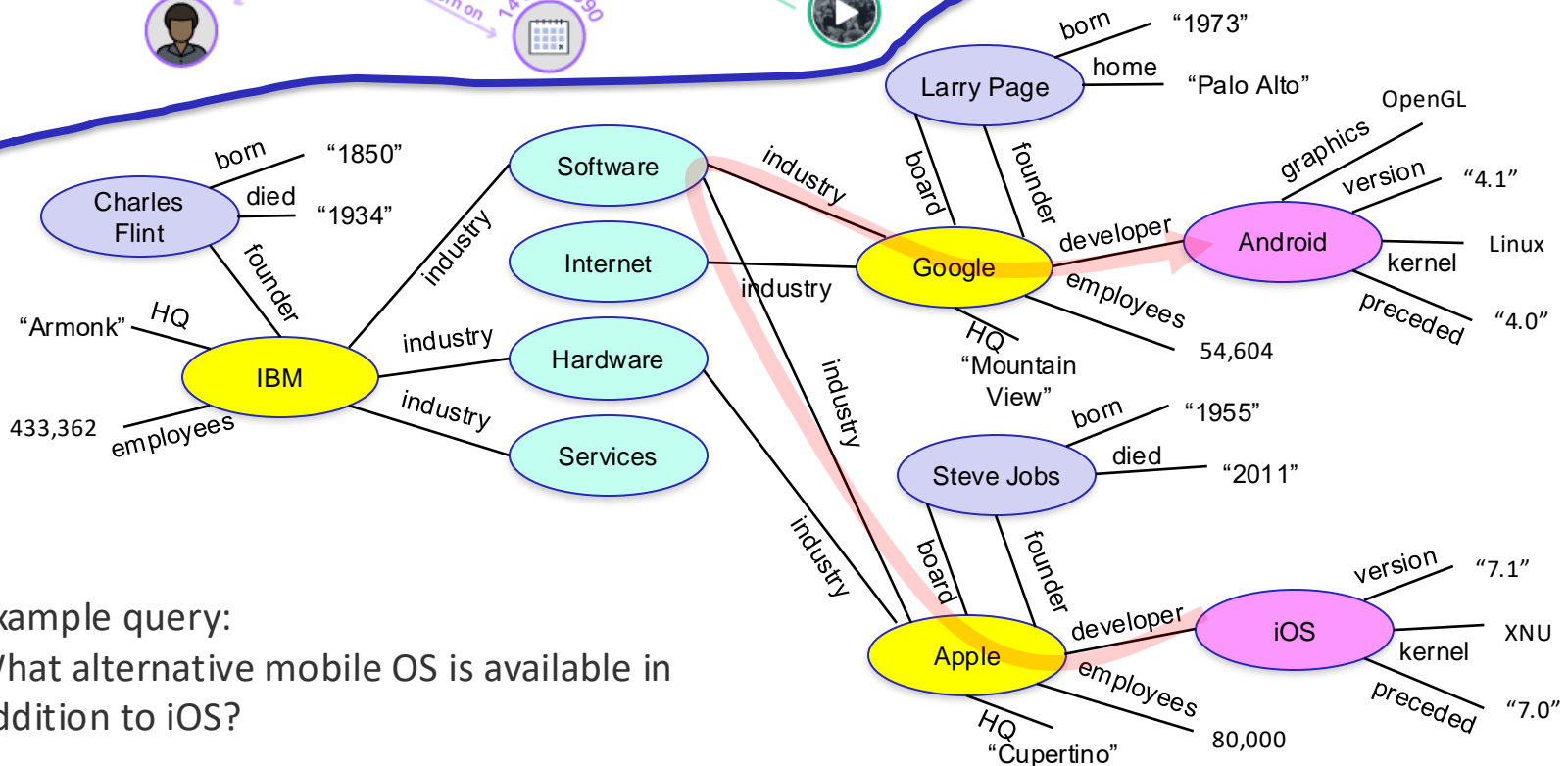
## Knowledge Graphs

Example query:

I'm interested in The Mona Lisa.

Help me find other artworks

- by Leonardo da Vinci, or
- located in The Louvre.



Example query:

What alternative mobile OS is available in addition to iOS?

# What is the fundamental challenge for RDB on Linked Data?

In Relational DB, relationships are *distributed*. It takes a long time to **JOIN** to retrieve a graph from data

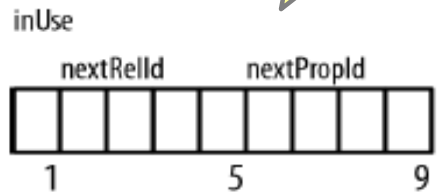
User					
UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order	
OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem		
OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

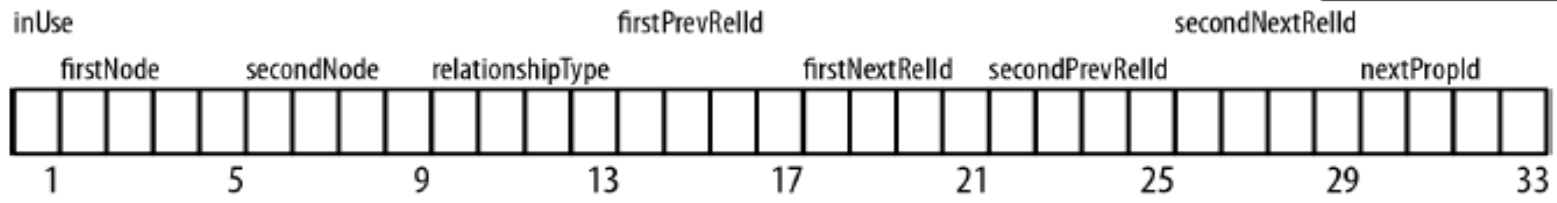
Product		
ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Native Graph DB stores nodes and relationships directly. It makes retrieval efficient.



Example query:  
What is the backlog of the product purchased by User A?

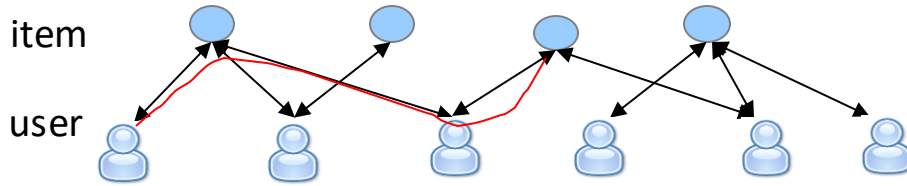
## Relationship (33 bytes)



Retrieving multi-step relationships is a **'graph traversal'** problem

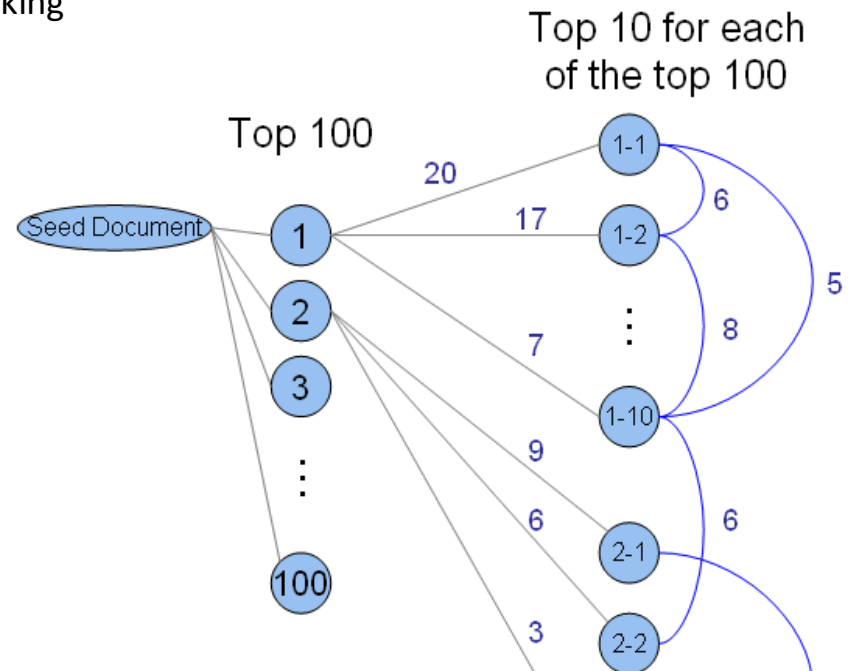
Cited "Graph Database" O'liey 2013

# Preliminary datastore comparison for Recommendation & Visualization



People who bought this also bought that..

Recommendation ==> 2-hop traversal & ranking

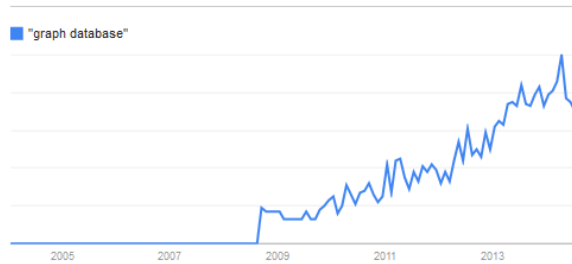


Document Visualization ==> 4-hop traversal & rankings

# Google Trends on Relational vs Graph Databases

Trends of search interest on **Graph Database** and **Relational Database**, realtime from Google (Google Trend normalizes Y-axis to the highest value in a chart to 100%):

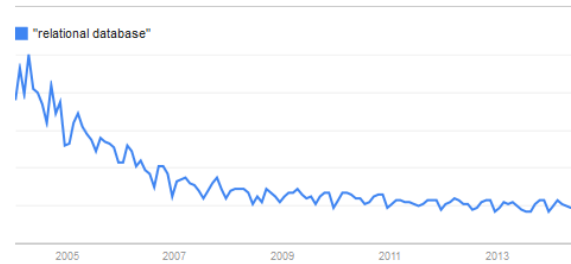
Interest over time. Web Search. Worldwide, 2004 - present.



Google™

[View full report in Google Trends](#)

Interest over time. Web Search. Worldwide, 2004 - present.

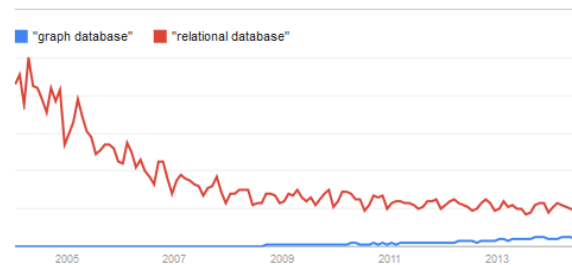


Google™

[View full report in Google Trends](#)

Comparison of relative amounts of searches on **Relational Database** and **Graph Database**:

Interest over time. Web Search. Worldwide, 2004 - present.



Google™

[View full report in Google Trends](#)

**“By 2025, graph technologies will be used in 80% of data and analytics innovations, up from 10% in 2021, facilitating rapid decision making across the enterprise.”**

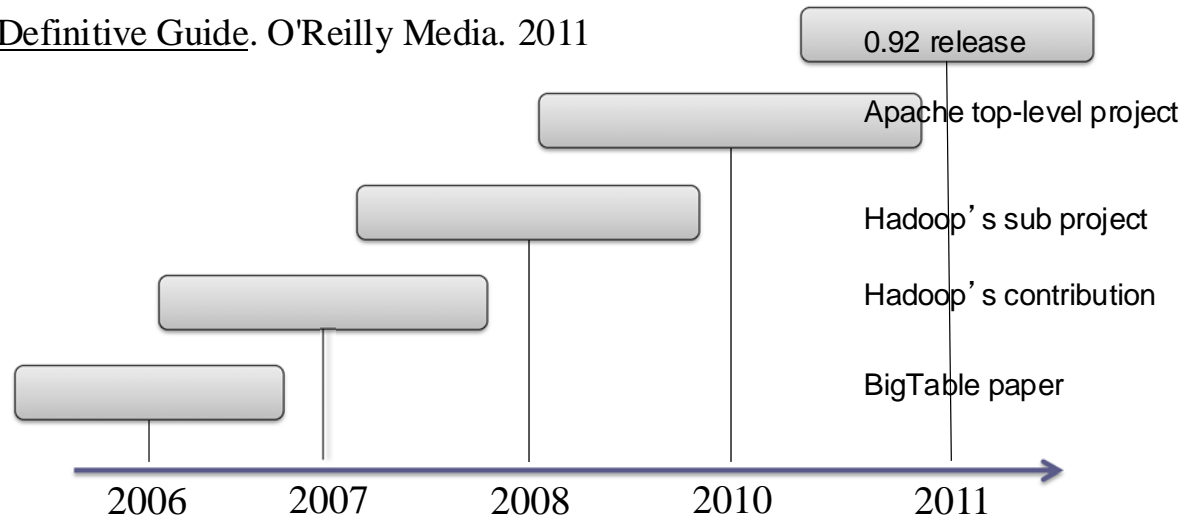
**- Gartner "Market Guide: Graph Database Management Solutions" Merv Adrian, Afraz Jaffri 30 August 2022**



- **HBase** is modeled after Google's BigTable and **written in Java, and is developed on top of HDFS**
- It provides a fault-tolerant way of **storing large quantities of sparse data**
  - Small amounts of information caught within a large collection of empty or unimportant data, e.g.,
    - Finding the 50 largest items in a group of 2 billion records
    - Finding the non-zero items representing less than 0.1% of a huge collection
- HBase features compression, in-memory operation, and Bloom filters on a per-column basis
- An HBase system comprises a set of tables
  - Each table contains **rows and columns**, much like a traditional database.
  - An HBase column represents an attribute of an object
  - Each table must have an element defined as a **Primary Key**, and all access attempts to HBase tables must use this Primary Key

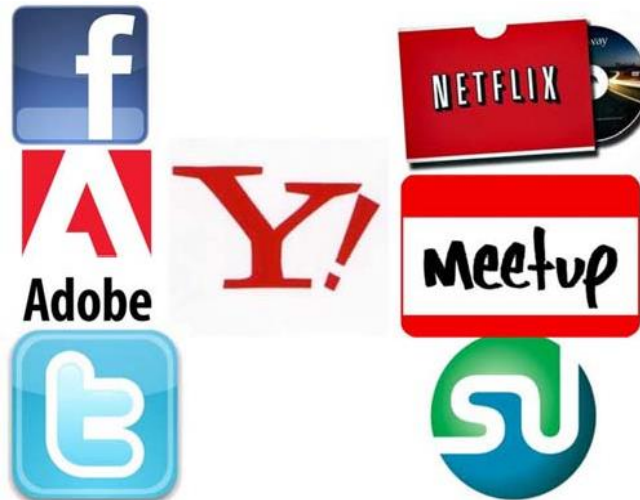
# HBase History

Source: Lars, George. HBase The Definitive Guide. O'Reilly Media. 2011



## Who Uses HBase?

- Here is a very limited list of well-known names
  - Facebook
  - Adobe
  - Twitter
  - Yahoo!
  - Netflix
  - Meetup
  - Stumbleupon
  - You????



# When to use HBase?

---

- **Not suitable for every problem**
  - Compared to RDBMs, it has VERY simple and limited APIs
- **Good for large amounts of data**
  - 100s of millions or billions of rows
  - If data is too small all the records will end up on a single node leaving the rest of the cluster idle
- **Must have enough hardware!!**
  - At the minimum 5 nodes
    - There are multiple management daemon processes: Namenode, HBaseMaster, Zookeeper, etc....
    - HDFS won't do well on anything under 5 nodes anyway; particularly with a block replication of 3
    - HBase is memory and CPU intensive
- **Carefully evaluate HBase for mixed workloads**
  - Client request (interactive, time-sensitive) vs. Batch processing (MapReduce)
    - SLAs on client requests would need evaluation
  - HBase has intermittent but large I/O access
    - May affect response latency!
- **Two well-known use cases**
  - Lots and lots of data (already mentioned)
  - Large amounts of clients/requests (usually cause a lot of data)
- **Great for single random selects and range scans by key**
- **Great for variable schema**
  - Rows may drastically differ
  - If your schema has many columns and most of them are null

# When NOT to use HBase?

---

- **Bad for traditional RDBMS retrieval**

- Transactional applications
- Relational analytics
  - 'group by', 'join', and 'where column like', etc....

- **Currently bad for text-based search access**

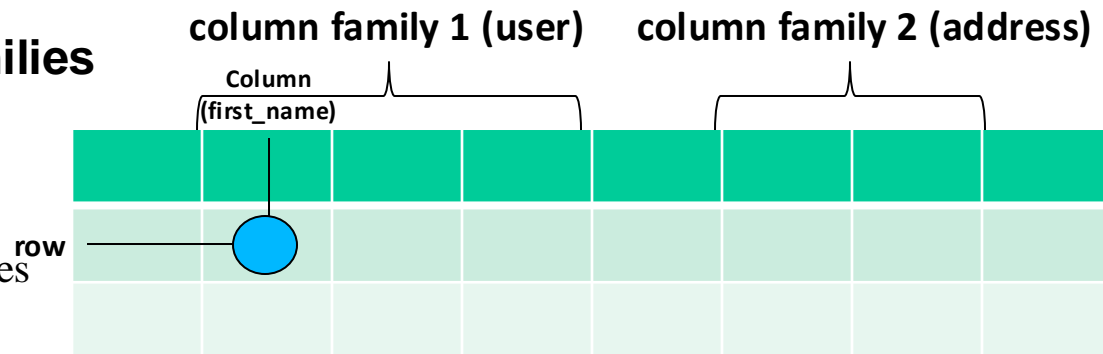
- There is work being done in this arena
  - HBasene: <https://github.com/akkumar/hbasene/wiki>
  - HBASE-3529: 100% integration of HBase and Lucene based on HBase' coprocessors
- Some projects provide solutions that use HBase
  - Lily=HBase+Solr <http://www.lilyproject.org>

# HBase Data Model

- **Data is stored in Tables**
- **Tables contain rows**
  - Rows are referenced by a unique (Primary) key
    - Key is an array of bytes – good news
    - Anything can be a key: string, long and your own serialized data structures
- **Rows made of columns**
- **Data is stored in cells**
  - Identified by “row x column-family:column”
  - Cell’s content is also an array of bytes

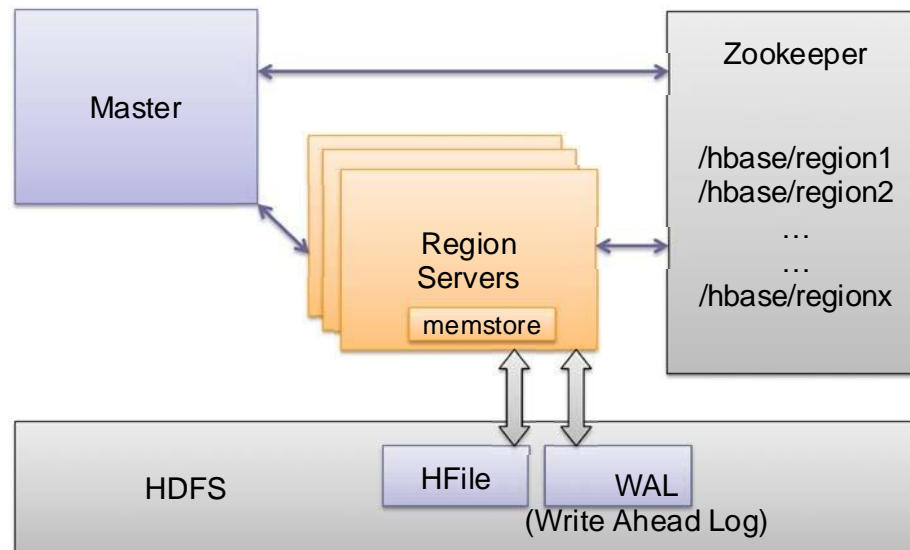
## HBase Families

- **Columns are grouped into families**
  - Labeled as “family:column”
    - Example “user:first\_name”
  - A way to organize your data
  - Various features are applied to families
    - Compression
    - In-memory option
    - **Stored together - in a file called HFile/StoreFile**
- **Family definitions are static**
  - Created with table, should be rarely added and changed
  - Limited to a small number of families
    - unlike columns that you can have millions of

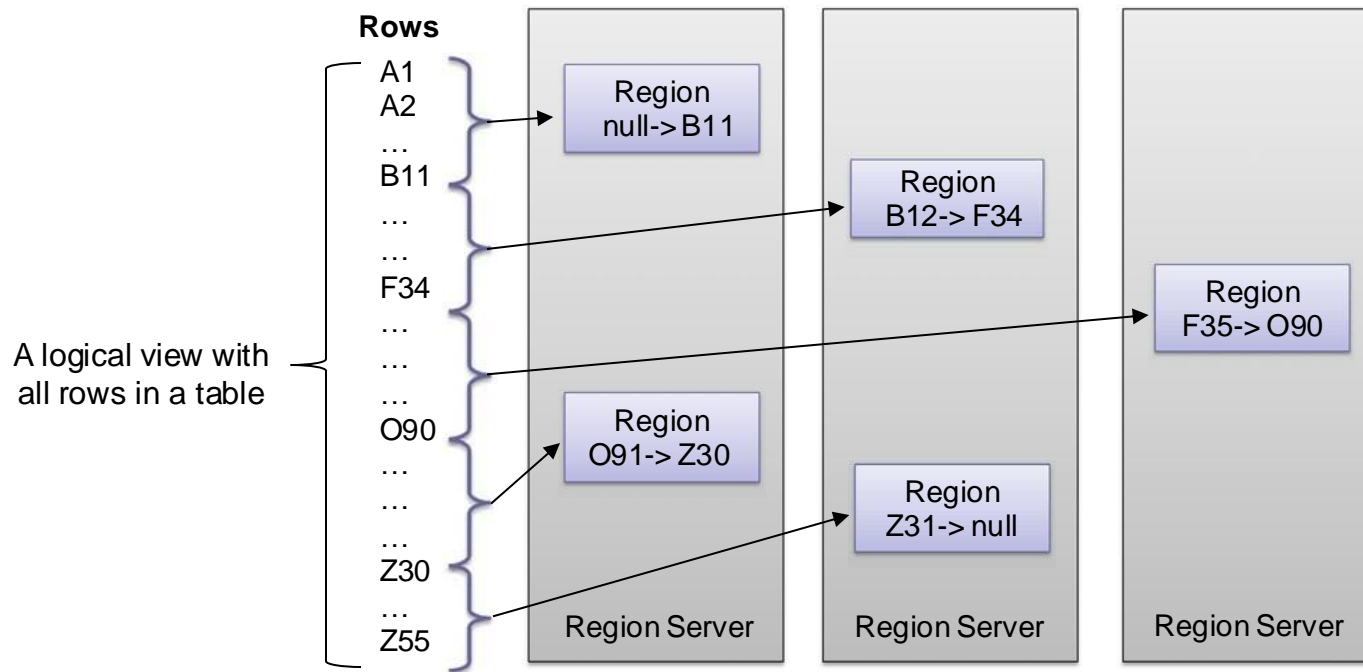


# HBase Distributed Architecture

- **Table is made of regions**
- **Region – a range of rows stored together**
  - Single shard, used for scaling
  - Dynamically split as they become too big and merged if too small
- **Region Server – serves one or more regions**
  - A region is served by only 1 Region Server
- **Master Server – daemon responsible for managing HBase cluster, or Region Servers**
- **HBase stores its data into HDFS**
  - Relies on HDFS's high availability and fault-tolerance features
- **HBase utilizes Zookeeper for distributed coordination**



# Row Distribution Between Region Servers



- **Regions per server depend on hardware specs. With today's hardware, it's common to have:**
  - 10 to 1000 regions per Region Server
  - Managing as much as 1GB to 2GB per region
    - How many rows per region? Depending on the size of each row and the size of a region
- **Benefits of splitting data into regions allows**
  - **Fast recovery** when a region fails
  - **Load balancing** when a region server is overloaded
    - May be moved between region servers
  - Splitting is **fast**
    - Reads from an original file while asynchronous process performs a split
  - Background processing: all of these happen automatically without user's involvement

# HBase Data Storage

---

- **Data (vertically along column families) is stored in files called HFiles/StoreFiles**
  - Usually saved in HDFS
- **HFile is basically a key-value map**
  - Keys are sorted lexicographically
- **When data is added, it's written to a log called Write Ahead Log (WAL) and is also stored in memory (memstore)**
- **Flush: when in-memory data exceeds maximum value, it is flushed to an HFile**
  - Data persisted to HFile can then be removed from WAL
  - Region Server continues serving read-writes during the flush operations, writing values to the WAL and memstore
- **HBase periodically performs data compaction**
  - ❖ Why?
    - ✓ To control the number of HFiles
    - ✓ To keep the cluster well balanced
  - Minor Compaction: Smaller HFiles are merged into larger HFiles (n-way merge)
    - Fast - Data is already sorted within files
    - Delete markers not applied
  - Major Compaction:
    - For each region, merge all the files within a column-family into a single file
    - Scan all the entries and apply all the deletes as necessary

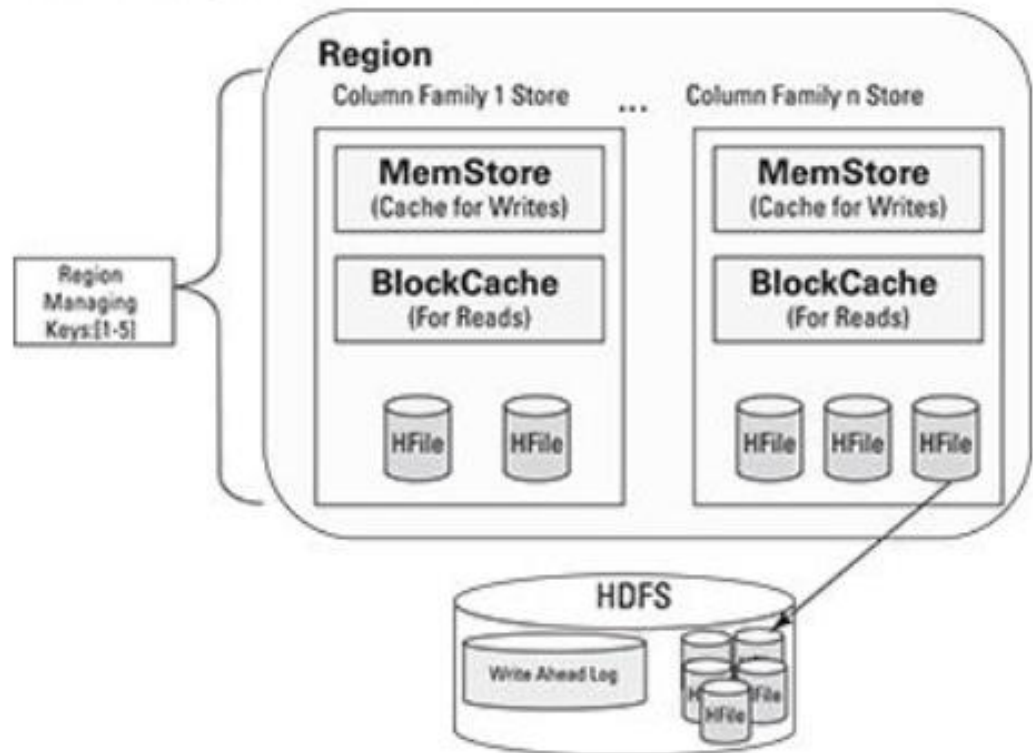
# HBase Architecture

## Logical Architecture

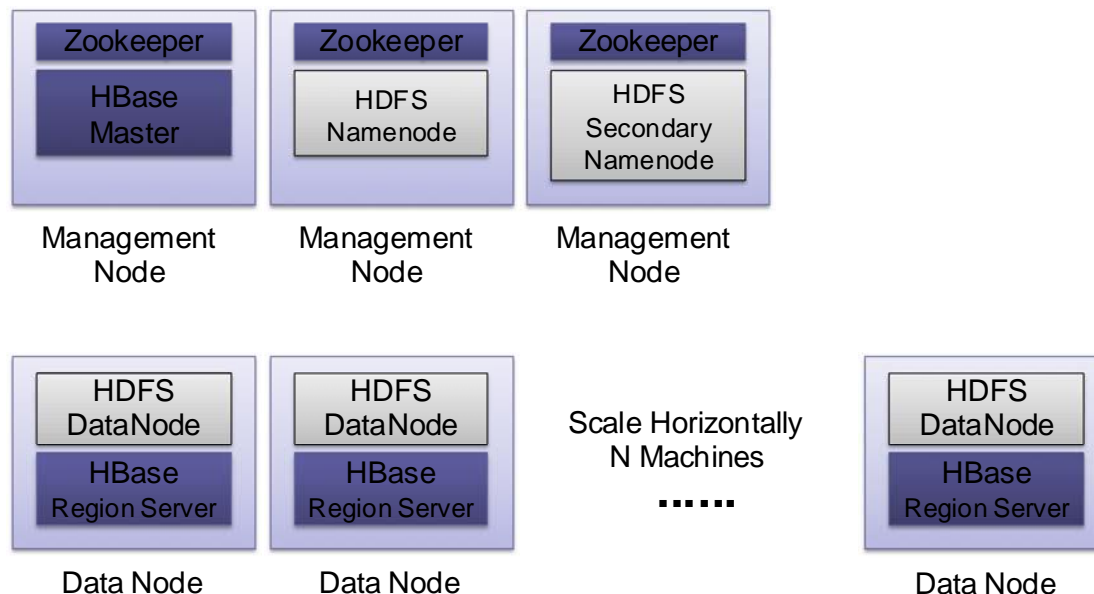


**Zookeeper Ensemble for HBase Coordination Services and Fault Recovery**

## HBase Regions in Detail



# HBase Deployment on HDFS

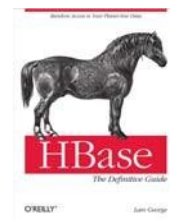


## Resources

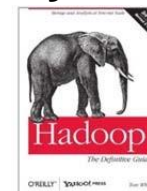
- Home Page
  - <http://hbase.apache.org>
- Mailing Lists
  - <http://hbase.apache.org/mail-lists.html>
  - Subscribe to User List
- Wiki
  - <http://wiki.apache.org/hadoop/Hbase>
- Videos and Presentations
  - <http://hbase.apache.org/book.html#other.info>

## Books

- HBase: The Definitive Guide by Lars George
  - Publication Date: September 20, 2011
- Apache HBase Reference Guide
  - Comes packaged with HBase
  - <http://hbase.apache.org/book/book.html>
- Hadoop: The Definitive Guide by Tom White
  - Publication Date: May 22, 2012
  - Chapter about Hbase



APACHE  
HBASE



# Characteristics of Data in HBase

Sparse data

**Table 12-1 Traditional Customer Contact Information Table**

<i>Customer ID</i>	<i>Last Name</i>	<i>First Name</i>	<i>Middle Name</i>	<i>E-mail Address</i>	<i>Street Address</i>
00001	Smith	John	Timothy	John.Smith@xyz.com	1 Hadoop Lane, NY 11111
00002	Doe	Jane	NULL	NULL	7 HBase Ave, CA 22222

Multiple versions of data for each cell

**Row Key** *Column Family: {Column Qualifier:Version:Value}*

00001 CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'}

ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}

00002 CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe',

ContactInfo: {'SA': 1383859185577:'7 HBase Ave, CA 22222'}

HDFS lacks **random read and write access**. This is where HBase comes into picture. It's a **distributed, scalable, big data store**, modeled after Google's BigTable. It stores data as key/value pairs.

## *Creating a table*

```
hbase(main):002:0> create 'CustomerContactInfo', 'CustomerName', 'ContactInfo'  
0 row(s) in 1.2080 seconds
```

Table name

Column  
family 1

Column  
family 2

# HBase Example

---

## Entering Records

```
hbase(main):008:0> put 'CustomerContactInfo', '00001', 'CustomerName:FN', 'John'  
0 row(s) in 0.2870 seconds
```

```
hbase(main):009:0> put 'CustomerContactInfo', '00001', 'CustomerName:LN', 'Smith'  
0 row(s) in 0.0170 seconds
```

```
hbase(main):010:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'T'  
0 row(s) in 0.0070 seconds
```

```
hbase(main):011:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'Timothy'  
0 row(s) in 0.0050 seconds
```

```
hbase(main):012:0> put 'CustomerContactInfo', '00001', 'ContactInfo:EA', 'John.Smith@xyz.com'  
0 row(s) in 0.0170 seconds
```

```
hbase(main):013:0> put 'CustomerContactInfo', '00001', 'ContactInfo:SA', '1 Hadoop Lane, NY 11111'  
0 row(s) in 0.0030 seconds
```

```
hbase(main):014:0> put 'CustomerContactInfo', '00002', 'CustomerName:FN', 'Jane'  
0 row(s) in 0.0290 seconds
```

```
hbase(main):015:0> put 'CustomerContactInfo', '00002', 'CustomerName:LN', 'Doe'  
0 row(s) in 0.0090 seconds
```

```
hbase(main):016:0> put 'CustomerContactInfo', '00002', 'ContactInfo:SA', '7 HBase Ave, CA 22222'  
0 row(s) in 0.0240 seconds
```

# HBase Example

---

## Scan Results

```
hbase(main):020:0> scan 'CustomerContactInfo', {VERSIONS => 2}
ROW      COLUMN+CELL
00001    column=ContactInfo:EA, timestamp=1383859183030, value=John.Smith@xyz.com
00001    column=ContactInfo:SA, timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
00001    column=CustomerName:FN, timestamp=1383859182496, value=John
00001    column=CustomerName:LN, timestamp=1383859182858, value=Smith
00001    column=CustomerName:MN, timestamp=1383859183001, value=Timothy
00001    column=CustomerName:MN, timestamp=1383859182915, value=T
00002    column=ContactInfo:SA, timestamp=1383859185577, value=7 HBase Ave, CA 22222
00002    column=CustomerName:FN, timestamp=1383859183103, value=Jane
00002    column=CustomerName:LN, timestamp=1383859183163, value=Doe
2 row(s) in 0.0520 seconds
```

## Using the *get* Command to Retrieve Entire Rows and Individual Values

```
(1) hbase(main):037:0> get 'CustomerContactInfo', '00001'
COLUMN          CELL
ContactInfo:EA  timestamp=1383859183030, value=John.Smith@xyz.com
ContactInfo:SA  timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
CustomerName:FN  timestamp=1383859182496, value=John
CustomerName:LN  timestamp=1383859182858, value=Smith
CustomerName:MN  timestamp=1383859183001, value=Timothy
5 row(s) in 0.0150 seconds
```

```
(2) hbase(main):038:0> get 'CustomerContactInfo', '00001',
    {COLUMN => 'CustomerName:MN'}
COLUMN          CELL
CustomerName:MN  timestamp=1383859183001, value=Timothy
1 row(s) in 0.0090 seconds
```

```
(3) hbase(main):039:0> get 'CustomerContactInfo', '00001',
    {COLUMN => 'CustomerName:MN',
    TIMESTAMP => 1383859182915}
COLUMN          CELL
CustomerName:MN  timestamp=1383859182915, value=T
1 row(s) in 0.0290 seconds
```

# Create HBase table in Java

```
public static void main(String[] args) throws IOException {

    // Instantiating configuration class
    Configuration con = HBaseConfiguration.create();

    // Instantiating HbaseAdmin class
    HBaseAdmin admin = new HBaseAdmin(con);

    // Instantiating table descriptor class
    HTableDescriptor tableDescriptor = new
    HTableDescriptor(TableName.valueOf("emp"));

    // Adding column families to table descriptor
    tableDescriptor.addFamily(new HColumnDescriptor("personal"));
    tableDescriptor.addFamily(new HColumnDescriptor("professional"));

    // Execute the table through admin
    admin.createTable(tableDescriptor);
    System.out.println(" Table created ");
}
}
```

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "ExampleReadWrite");
job.setJarByClass(MyReadWriteJob.class); // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500); // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false); // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
    sourceTable, // input table
    scan, // Scan instance to control CF and attribute selection
    MyMapper.class, // mapper class
    null, // mapper output key
    null, // mapper output value
    job);
TableMapReduceUtil.initTableReducerJob(
    targetTable, // output table
    null, // reducer class
    job);
job.setNumReduceTasks(0);

boolean b = job.waitForCompletion(true);
if (!b) {
    throw new IOException("error with job!");
}
```

# HBase Table Mapper and Reducer

JAVA

```
public static class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "cf".getBytes();
    public static final byte[] ATTR1 = "attr1".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException,
        InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);    // we can only emit Writables...
        context.write(text, ONE);
    }
}
```

JAVA

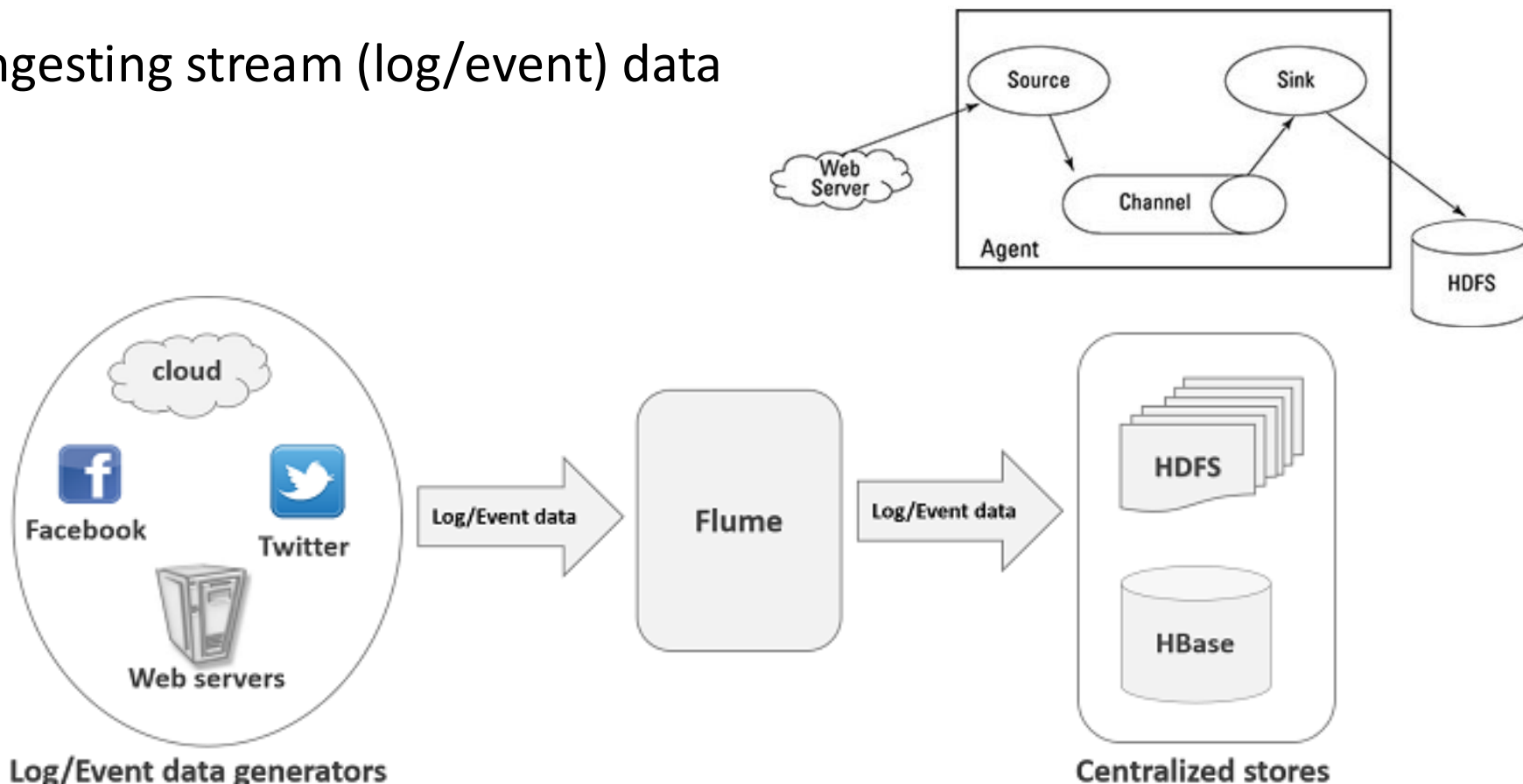
```
public static class MyTableReducer extends TableReducer<Text, IntWritable,
    ImmutableBytesWritable> {
    public static final byte[] CF = "cf".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
        InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.add(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

# Ingesting Data into HDFS/HBase – Apache Flume

## Ingesting stream (log/event) data



### Flume Features:

- Ingest log data from multiple web servers into a centralized store (HDFS, HBase) efficiently
- Import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart, along with the log files
- Support a large set of sources and destinations types
- Support multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Can be scaled horizontally

---

**Questions?**