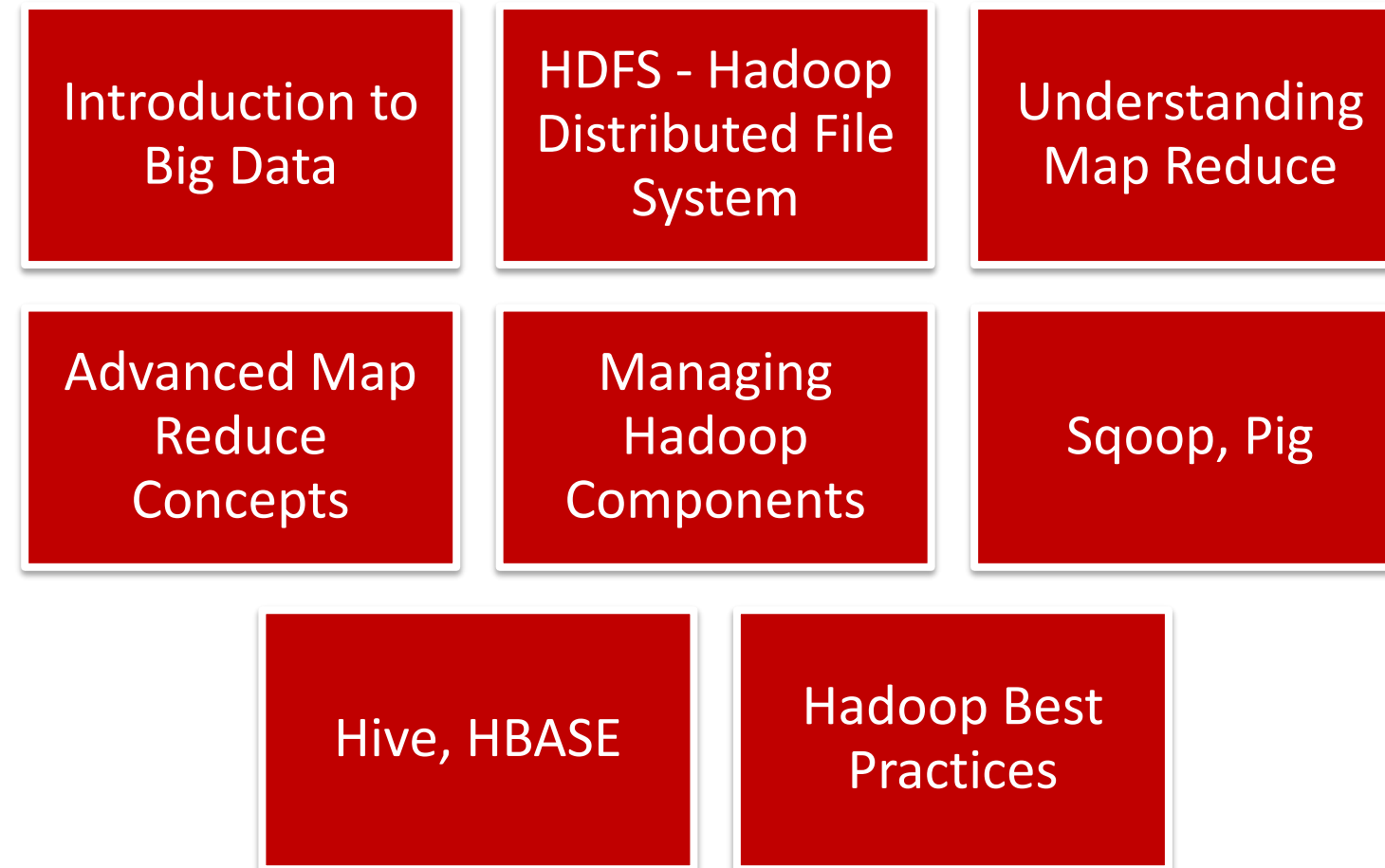


# BIGDATA BOOTCAMP

**1<sup>ST</sup> TO 4<sup>TH</sup> MAY, 2014**

**< bigdatabootcamp />**

# BIGDATA

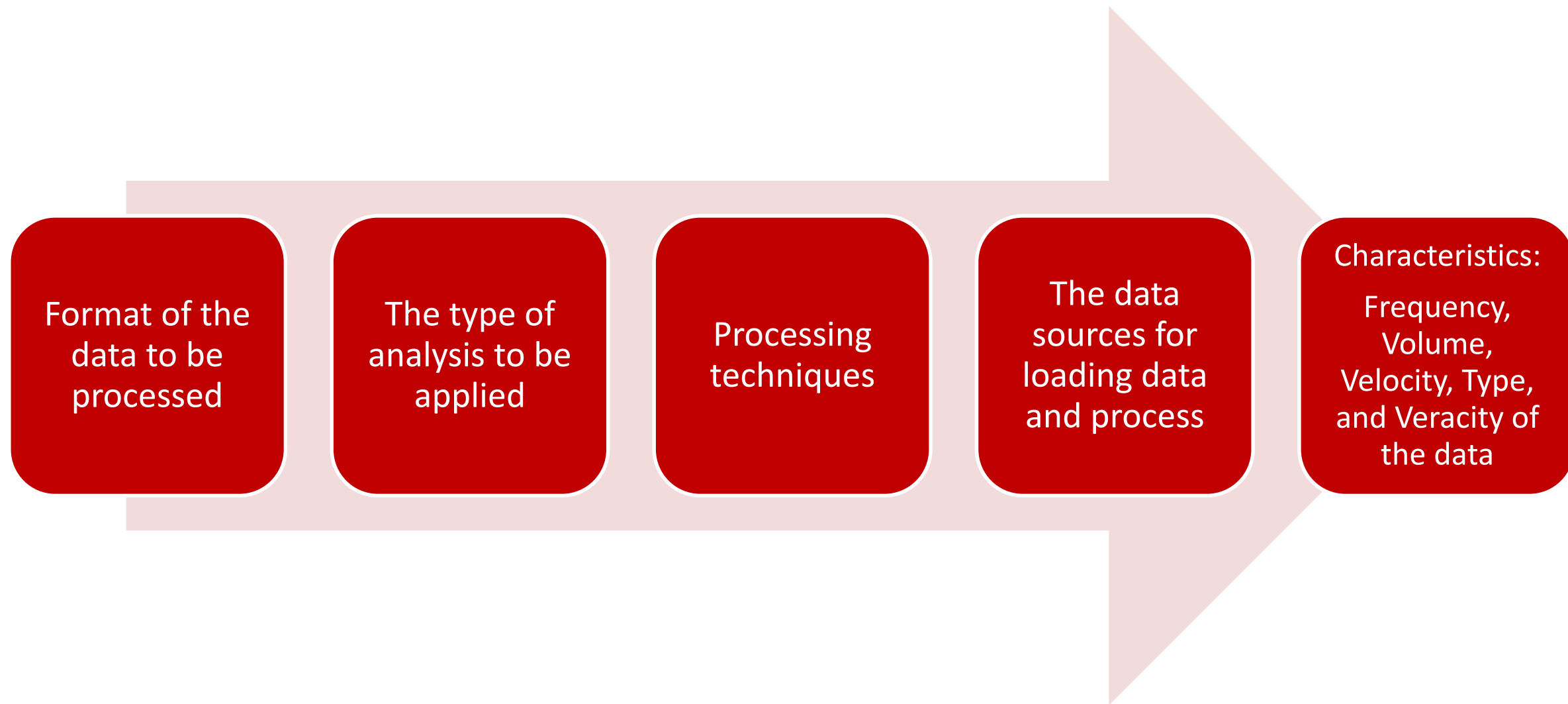


# AGENDA

WHAT IS BIGDATA?

# WHAT IS BIGDATA

CLASSIFY DATA BASED ON:

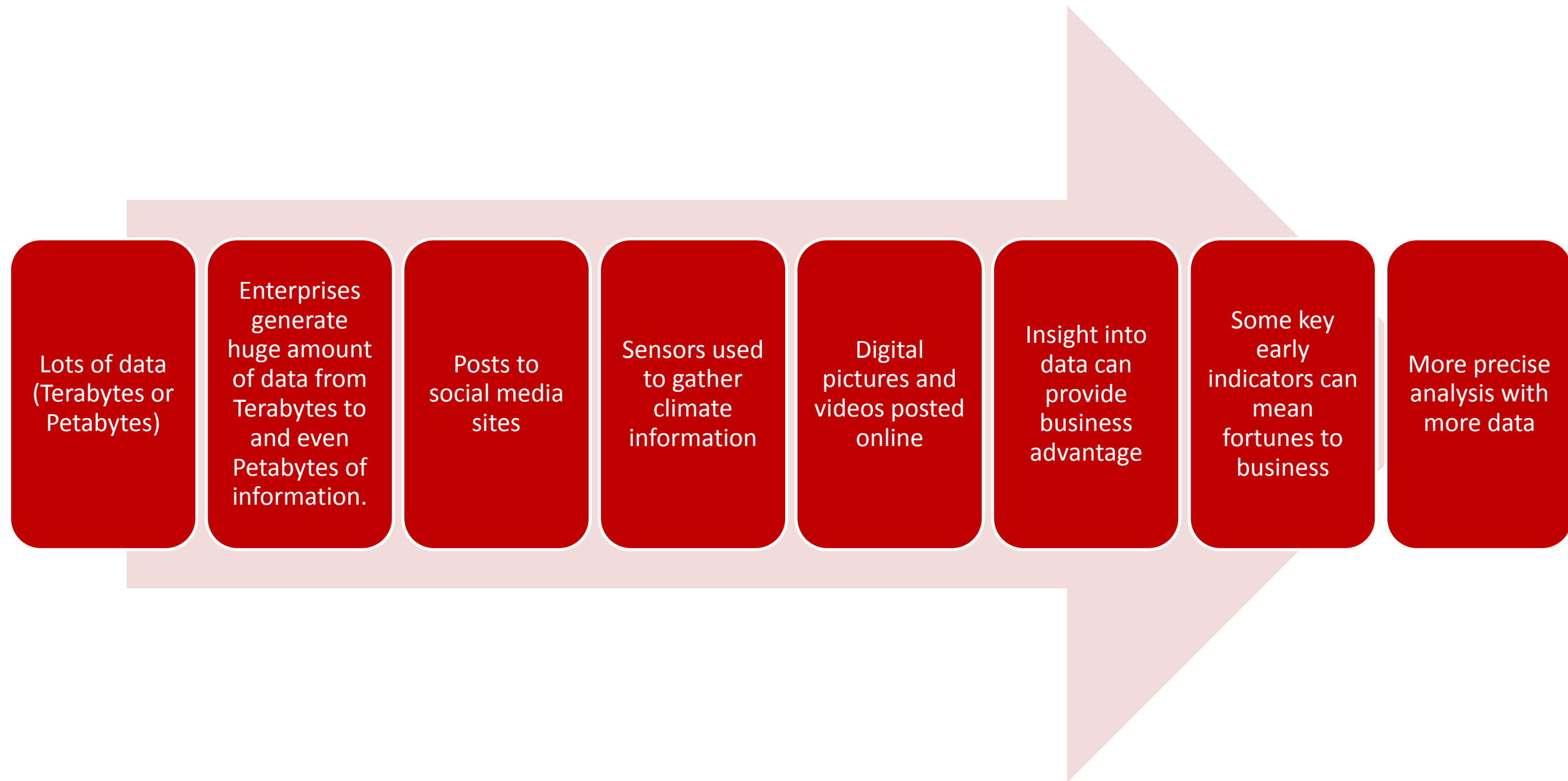




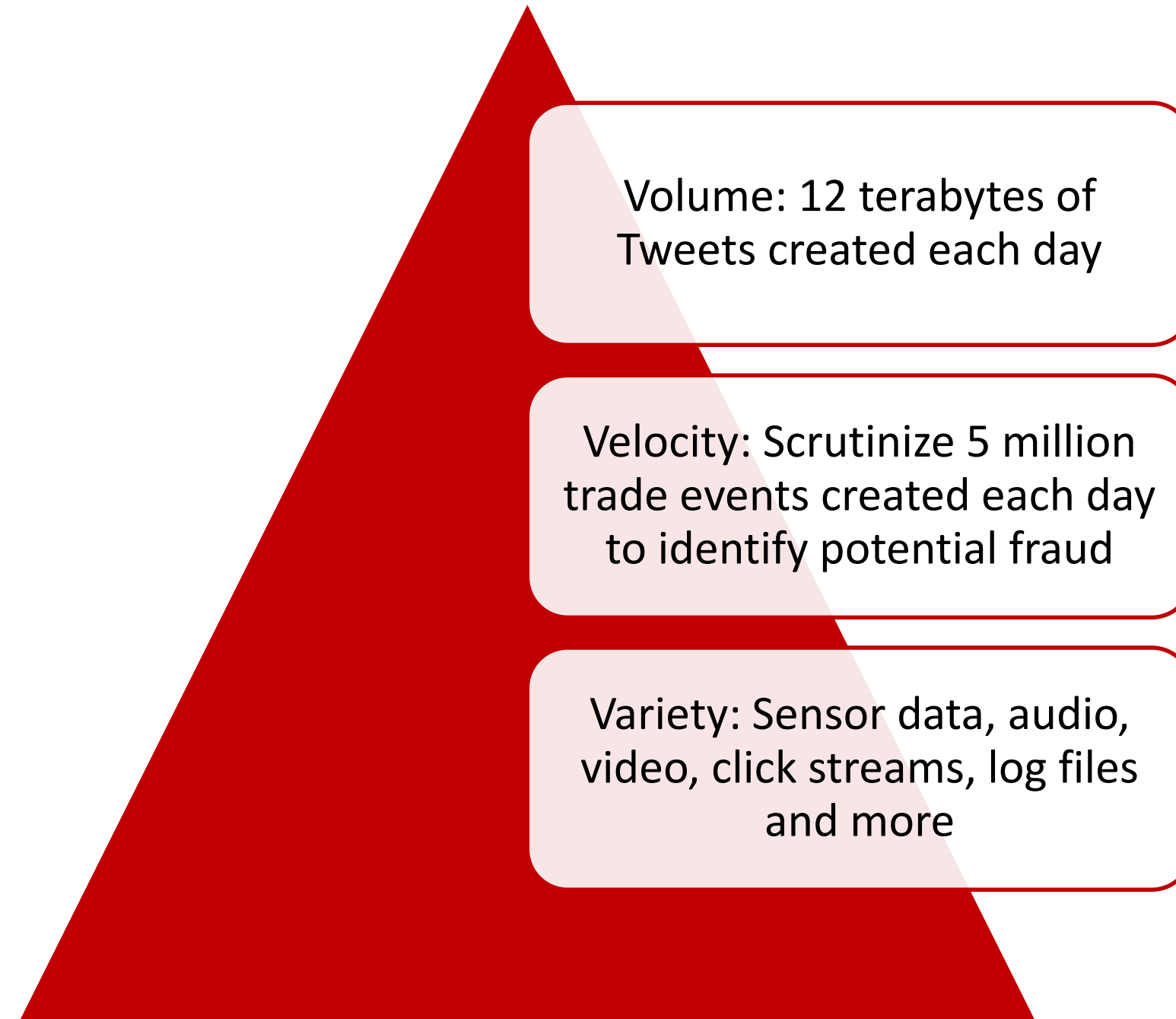
# IBM BIGDATA DEFINITION



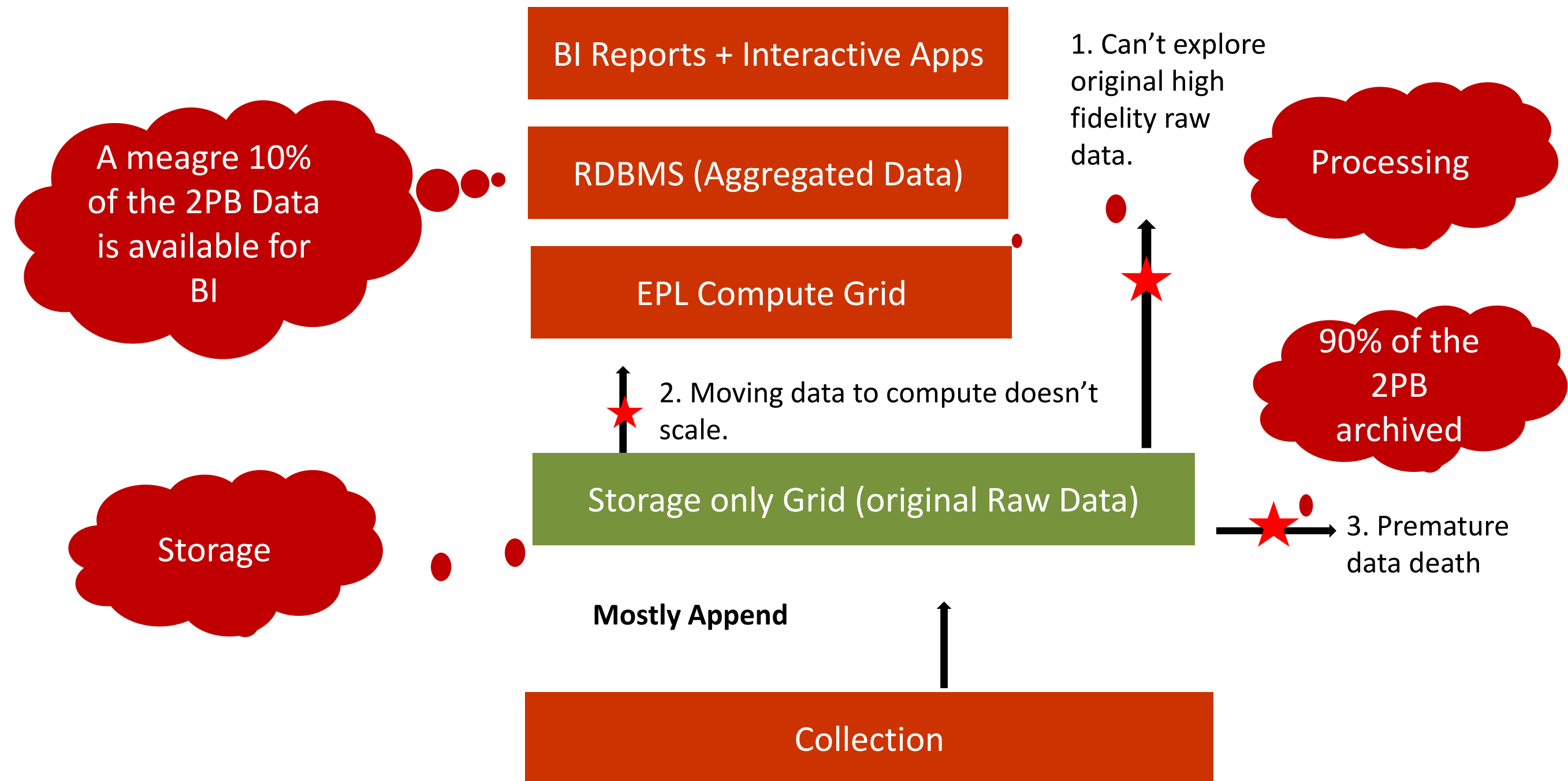
# WHAT IS BIGDATA



# THE 3 V's of BIGDATA

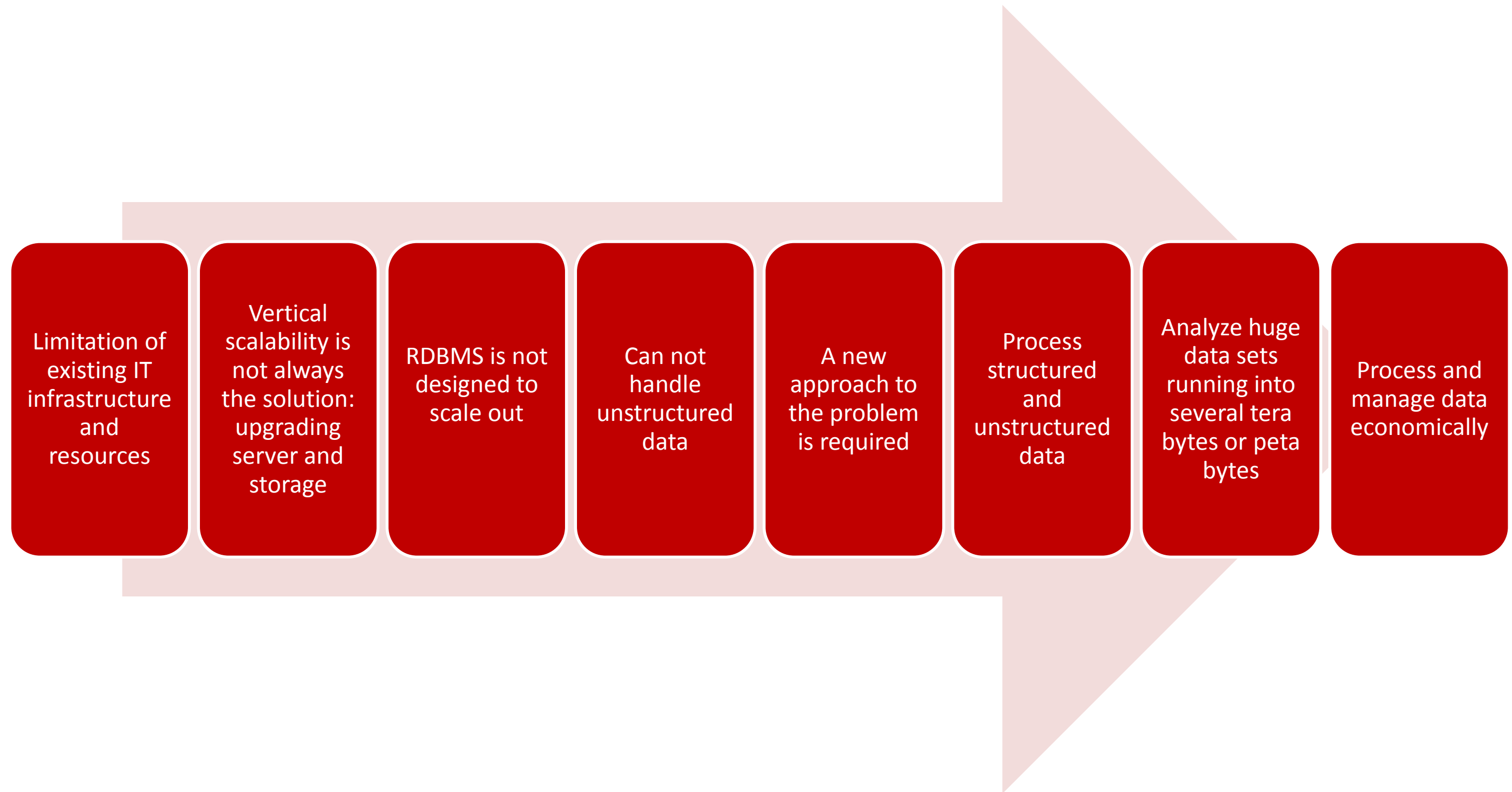


# LIMITATIONS OF EXISTING TECHNOLOGIES

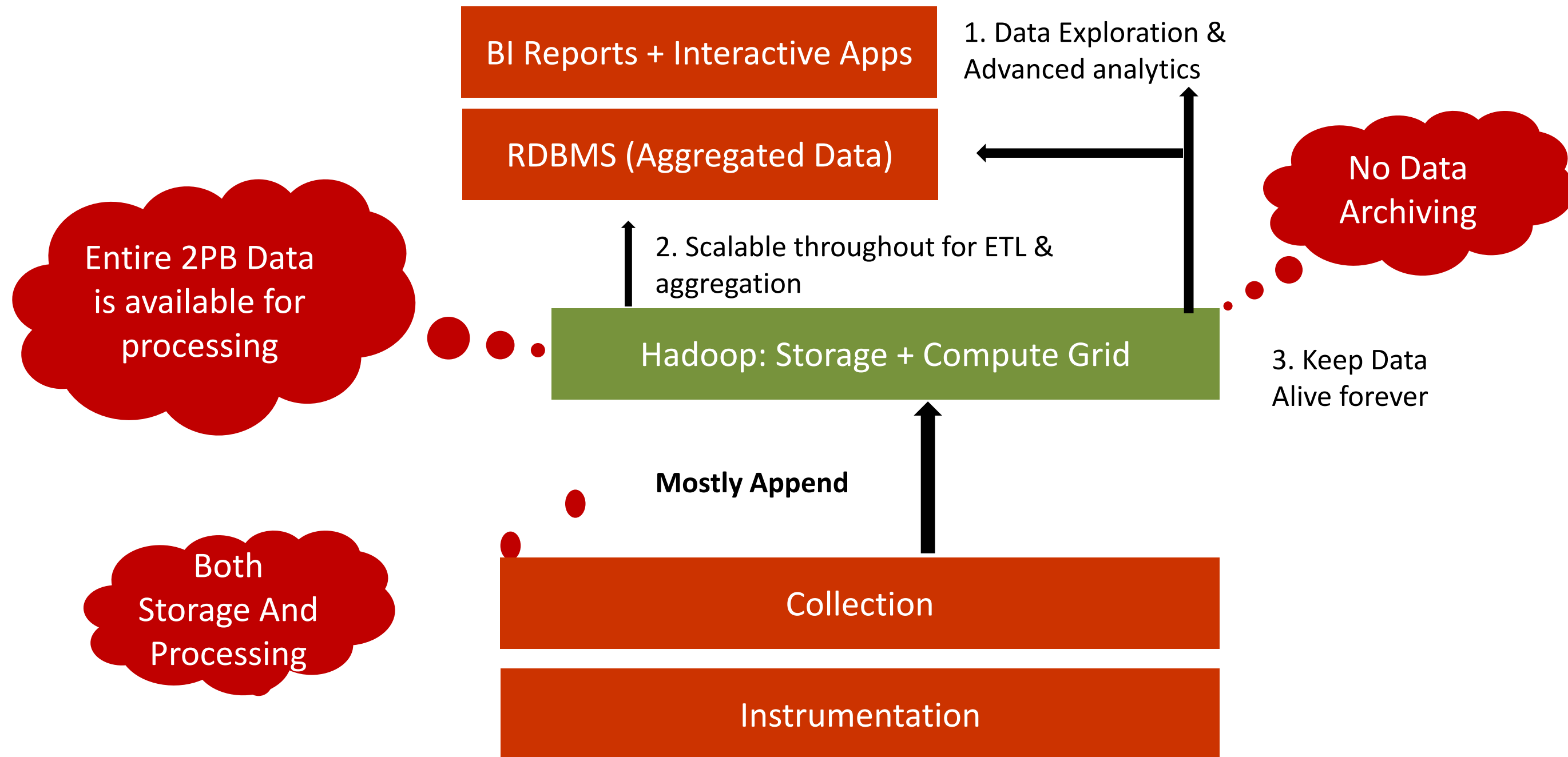




# WHY ALL THIS ?



# HADOOP ADVANTAGE



# STRUCTURED & UNSTRUCTURED

MAP THE FOLLOWING TO CORRESPONDING DATA TYPE:

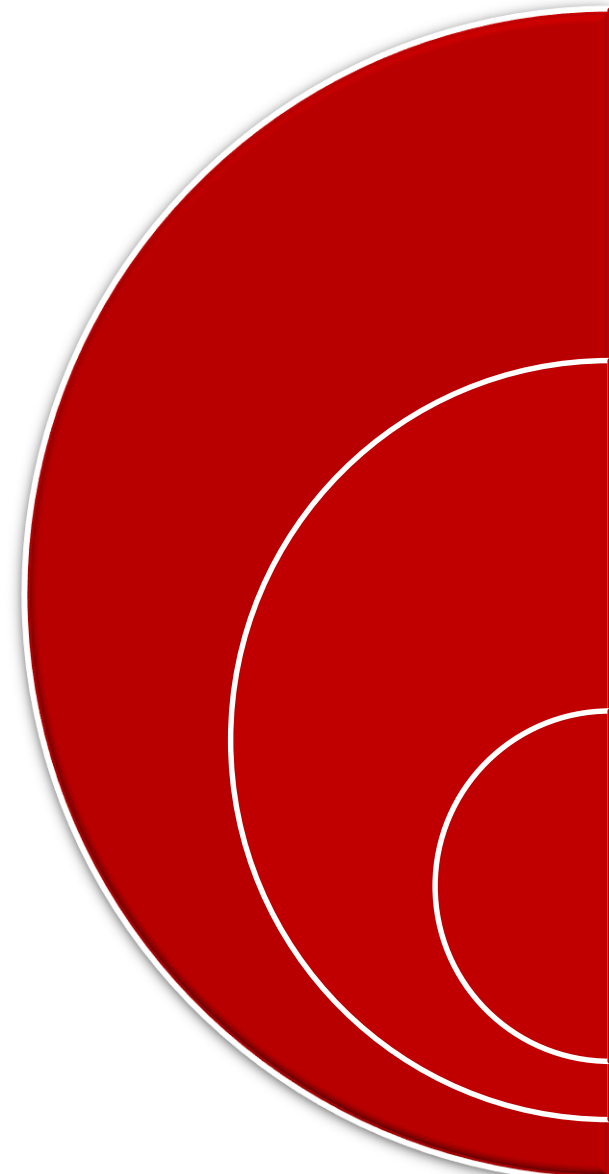
XML FILES

WORD DOCS, PDF FILES, TEXT FILES

EMAIL BODY HELLO THERE!

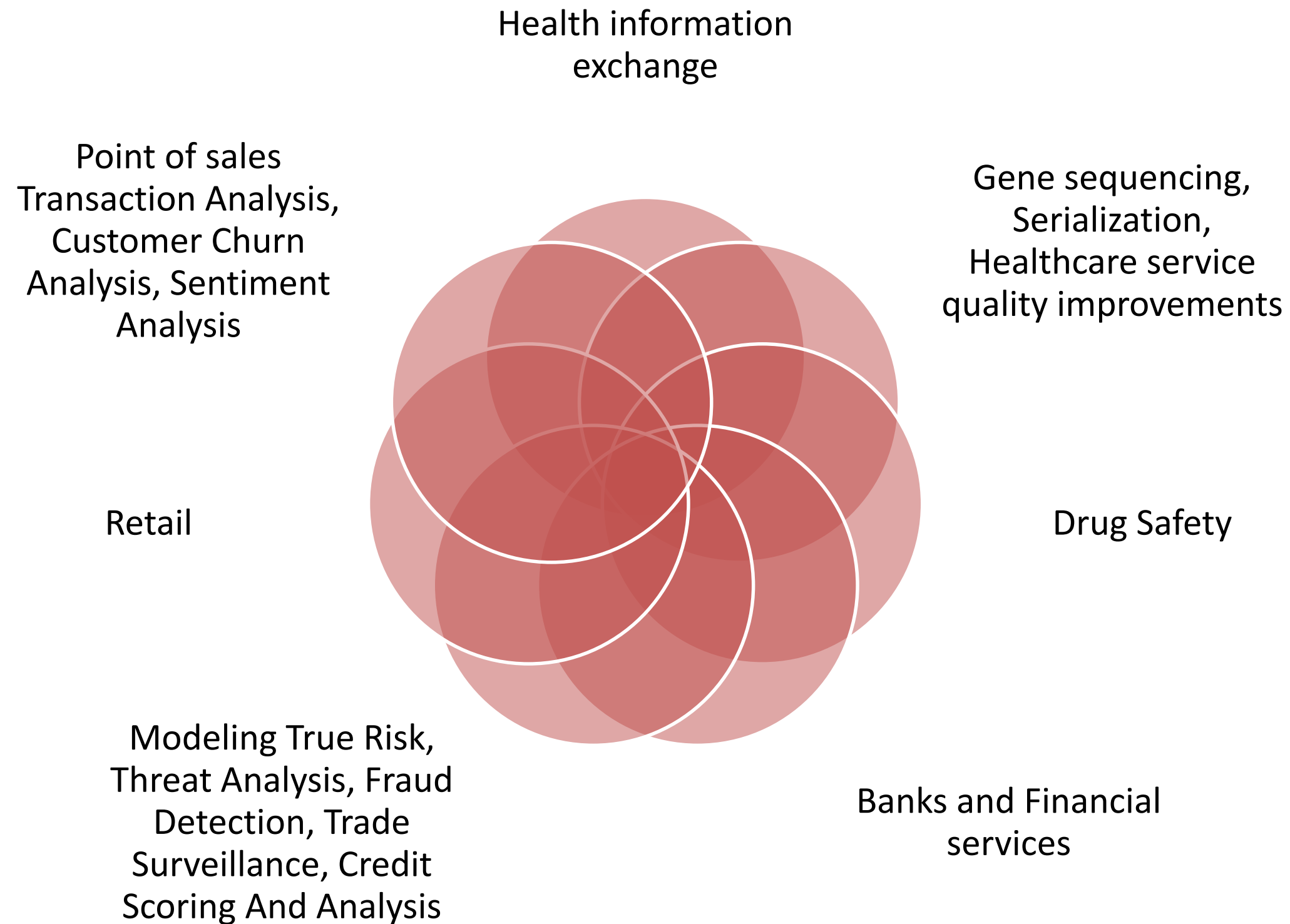
DATA FROM ENTERPRISE SYSTEMS  
(ERP, CRM etc.)

# SCENARIOS



Web and e-tailing: Recommendation Engines, Ad Targeting, Search Quality, Abuse and Click Fraud Detection	
Telecommunications: Customer Churn Prevention, Network Performance Optimization, Calling Data Record (CDR) Analysis, Analyzing Network to Predict Failure	
Government: Fraud Detection And Cyber Security, schemes, Justice, Healthcare & Life Sciences	Welfare

# SCENARIOS



# REFERENCES

Big Data

[http://en.wikipedia.org/wiki/Big\\_Data](http://en.wikipedia.org/wiki/Big_Data)

IBM's definition – Big Data Characteristics

<http://www-01.ibm.com/software/data/bigdata/>



# Hadoop

- Open source Apache Project
- Designed for massive scale
- Designed to run on commodity servers
- Design to recover from failure
- Parallel programming model
- Who is using it?

# Hadoop

- Yahoo! - Over 20,000 servers running Hadoop
- Largest Hadoop cluster is 4000 servers, 16PB raw storage
- Face book- 2000 servers
- 24 PB raw storage, 100TB raw log/day
- Other users: eBay and LinkedIn

# HADOOP – IT’S ABOUT SCALE AND STRUCTURE

Structured	Data Types	Multi and Unstructured
Limited, No Data Processing	Processing	Processing coupled with Data
Standards & Structured	Governance	Loosely Structured
Required On write	Schema	Required On Read
Reads are Fast	Speed	Writes are Fast
Software License	Cost	Support Only
Known Entity	Resources	Growing, Complexities, Wide
Interactive OLAP Analytics	Best Fit Use	Data Discovery
Complex ACID Transactions		Processing Unstructured Data
Operational Data Store		Massive Storage/Processing

# WHY NOT EXISTING EDW SOLUTIONS?

Amount of data that need to be analyzed has increased significantly in recent times.

Petabytes of data generated every day for organizations like face book, twitter, Google etc.

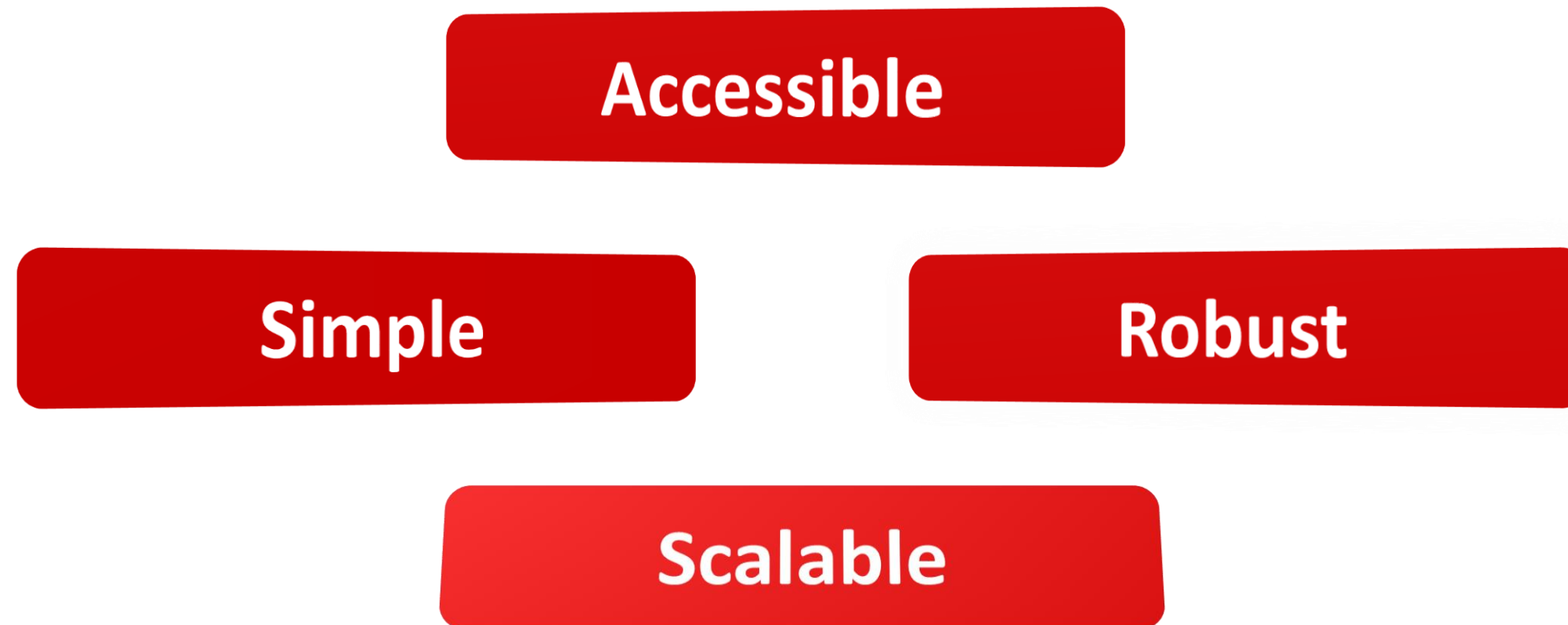
Cost of implementing DW solutions are very high.

The cost of commercially available solutions along with the cost of data processing and storage are significantly high.

Need to analyze unstructured data

Generated by different sources like web, social media, sensors and various media files.

# HADOOP – WHY IS IT DIFFERENT ?



# HADOOP COMPONENTS

## Components

1. HDFS – Storage
2. Map Reduce for processing

## Daemons

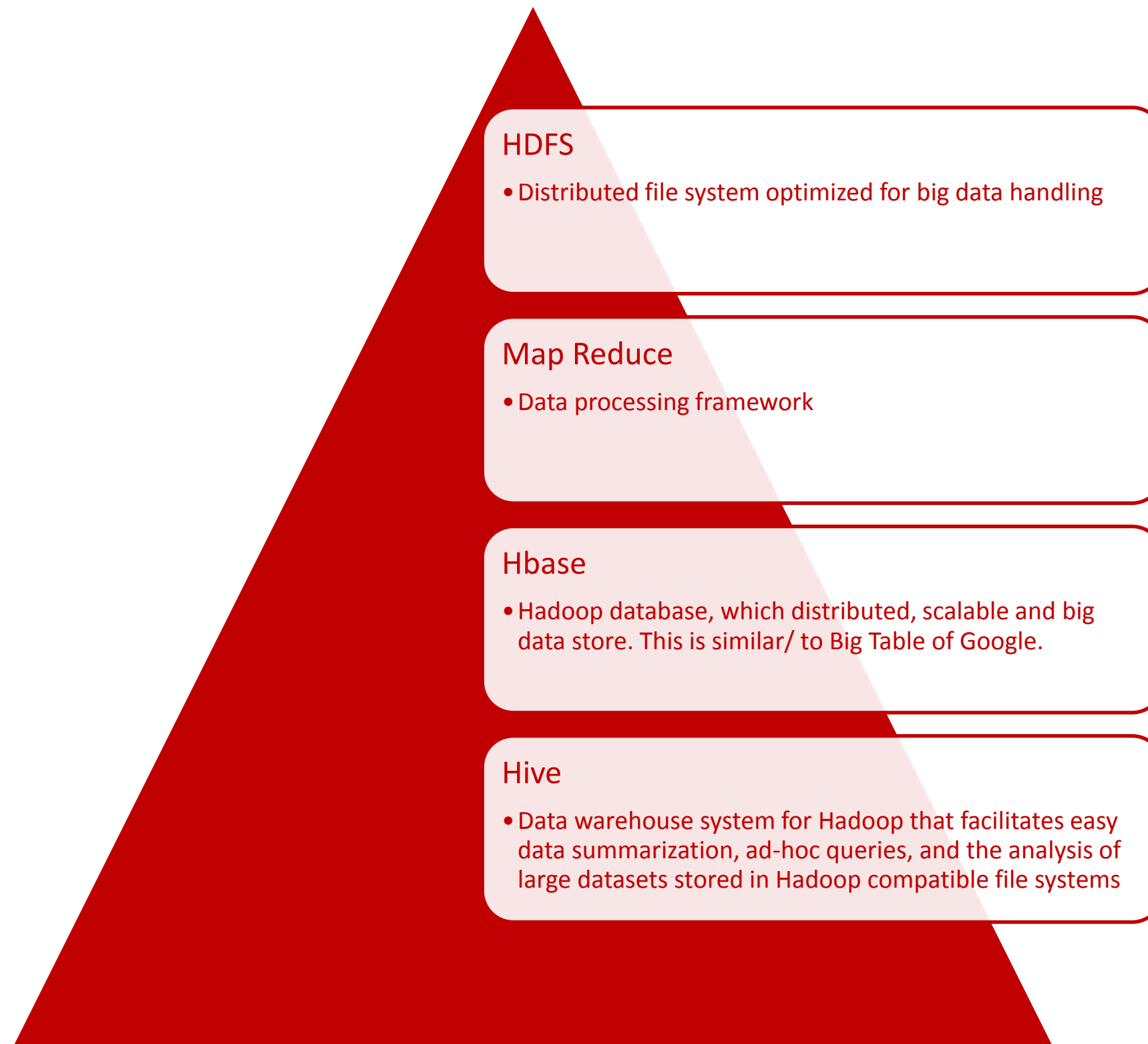
Name Node  
Secondary Name  
node  
Data Node  
Job Tracker  
Task Tracker

# HADOOP COMPONENTS

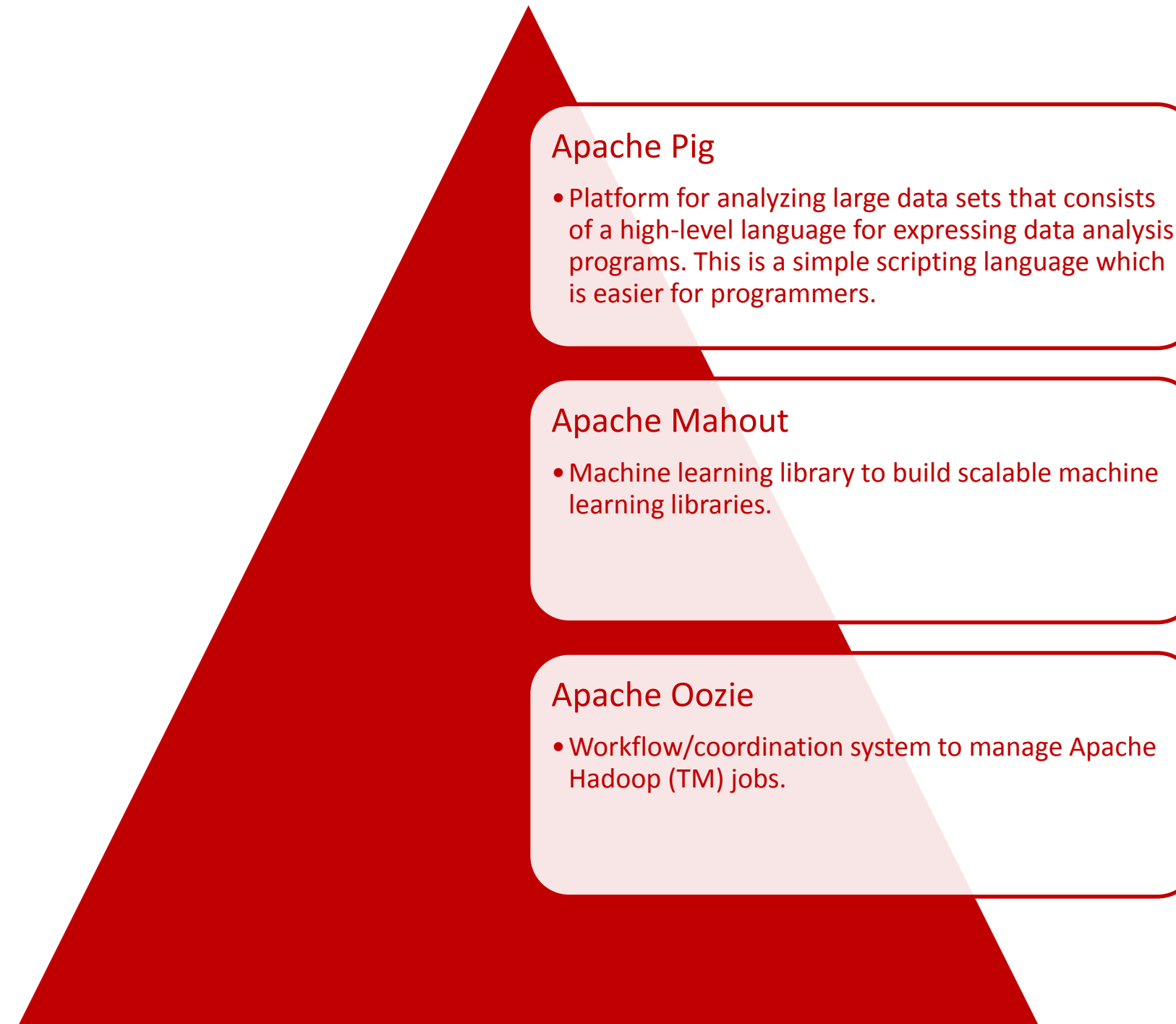
Apache Oozie (Workflow)		
Hive DW System	Pig Latin Data Analysis	Mahout Machine Learning
Map Reduce Framework		
H Base		
HDFS (Hadoop Distributed File System)		



# HADOOP COMPONENTS



# HADOOP COMPONENTS..



# WHY HADOOP?

## Financial Analysis

- Fraud Detection, Consumer spending patterns, Securities Analysis, sentiment analysis

## Application Log Analysis

- Application Behavior, User usage patterns

## Social Media

- Objectionable content or image filtering
- Content based user profiling for targeted advertisement

## Web/Content Indexing, Document clustering

## Scientific Simulation

## Bio Informatics Research

## Machine Learning / Translation

# AGENDA

HDFS

# HDFS

HDFS - Key Design Principles

HDFS - Building Blocks

Name Node & Data Node

Deployment modes

HDFS commands

# HDFS - Key Design Principles

- Storing large data files - Tera-bytes or Peta-bytes
- Partitioned and spread across multiple machines
- Inbuilt replication to tolerate failure of machines
- Can use commodity hardware
- Streaming data access
- Low latency data access



# HDFS - BUILDING BLOCKS

File is split into multiple chunks and stored

Each chunk is called BLOCK

HDFS block sizes are large-64 MB

Why large block size?

Hadoop does not access one record at a time, it has to process all data

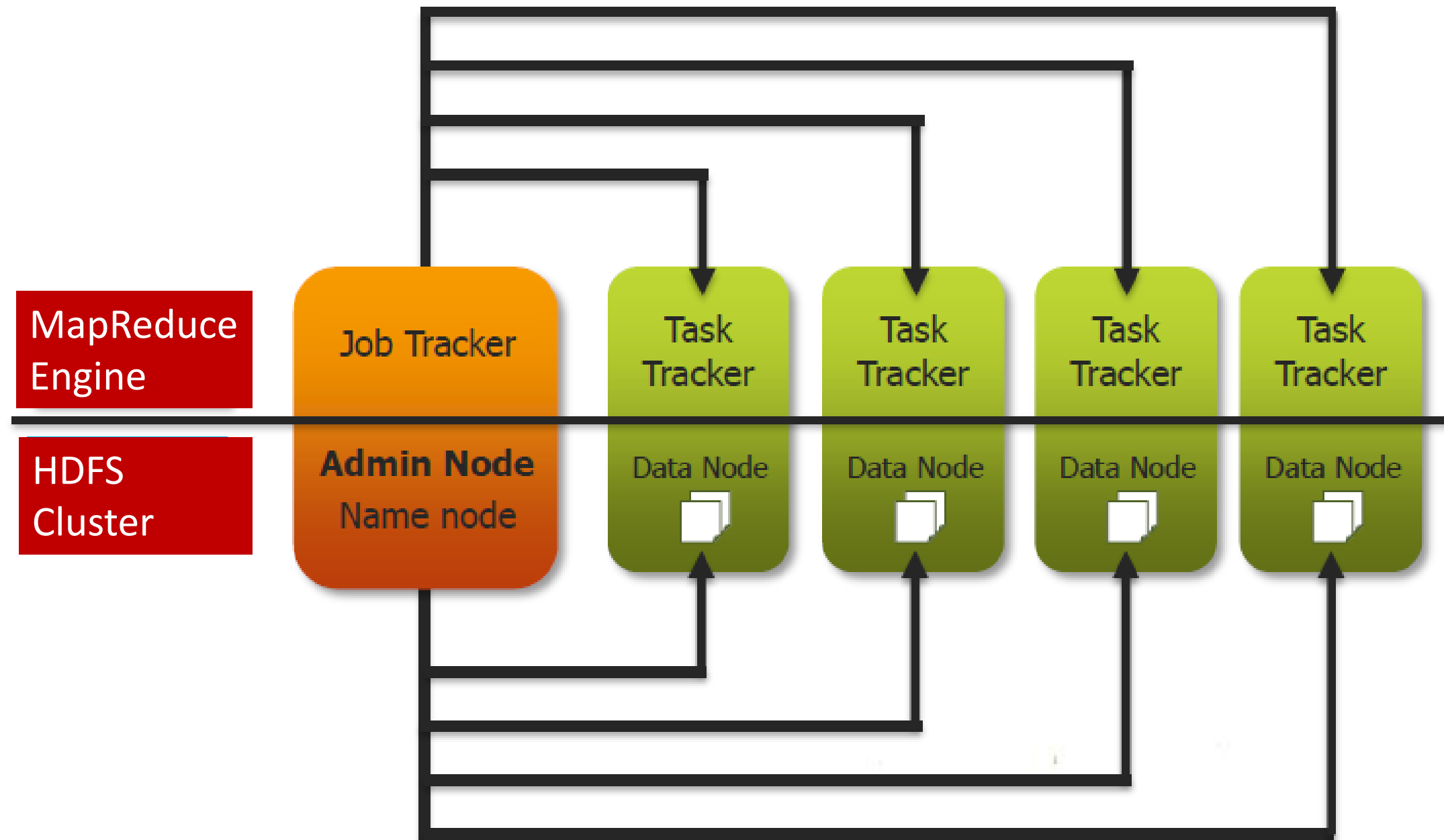
Keeping block size large keeps number of disk transfer to low

Map Reduce processes one block at a time

Blocks are replicated across multiple machines

By default it stores 3 copies of each block in separate machines at any point of time

# HADOOP COMPONENTS



# NAME NODE & DATA NODE

## Name Node

Manages the file system namespace

Stores the metadata of all the file and directory details

Knows what data nodes and the blocks of files they store

Without Name Node, HDFS is unusable

To withstand failure write metadata to multiple disks or local disk and NFS

## Secondary Name Node

Is NOT an automatic failover system

Merges namespace with the edit log

Recover from Name Node failure

Use NFS or backup copy of metadata and switch secondary to primary

# Name Node & Data Node

Name Node & Data Node

Data Node

Store data blocks

Report to Name Node periodically about the blocks they store

# DEPLOYMENT MODES

## Standalone or local mode

- No daemons running
- Everything runs on single JVM
- Good for Development

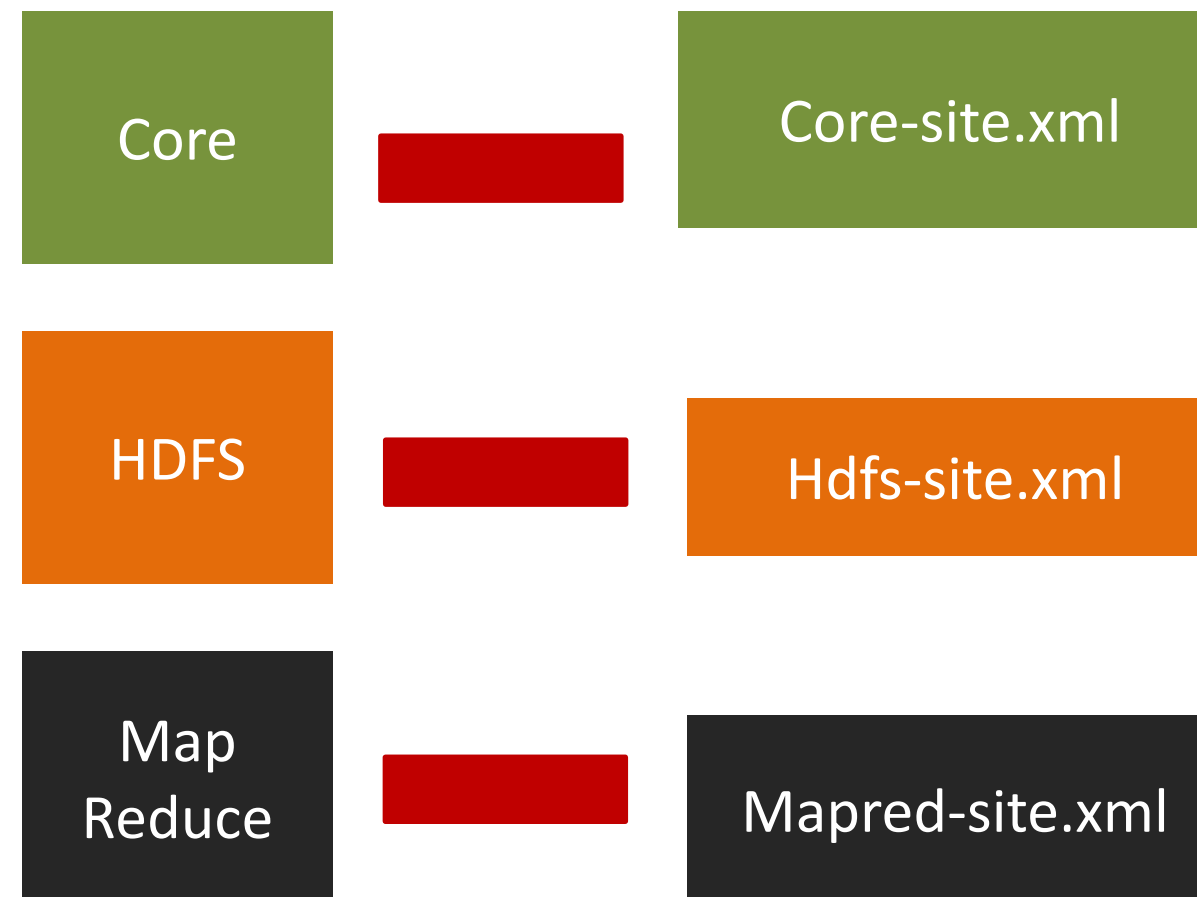
## Pseudo-distributed Mode

- All daemons running on single machine, a cluster simulation on one machine
- Good For Test Environment

## Fully distributed Mode

- Hadoop running on multiple machines on a cluster
- Production Environment

# HADOOP CLUSTER CONFIGURATION





# HADOOP CONFIGURATION FILES

hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and Map Reduce.
hdfs-site.xml	Configuration settings for HDFS daemons: the name node, the secondary name node, and the data nodes.
mapred-site.xml	Configuration settings for Map Reduce daemons: the job tracker, and the task trackers.
masters	A list of machines (one per line) that each run a secondary name node.
slaves	A list of machines (one per line) that each run a data node and a task tracker.
hadoop-metrics.properties	Properties for controlling how metrics are published in Hadoop
log4j.properties	Properties for system log files, the name node audit log, and the task log for the task tracker child process

# CONFIGURING HDFS

Property	Value	Description
dfs.name.dir	<value>/disk1/hdfs/name,/remote/h dfs/name</value>	The list of directories where the name node stores its persistent metadata. The name node stores a copy of the metadata in each directory in the list. (Comma separated directory names) {hadoop.tmp.dir}/dfs/name
dfs.data.dir	<value>/disk1/hdfs/data,/ disk2/hdfs /data</value>	A list of directories where the data node stores blocks. Each block is stored in only one of these directories. \${hadoop.tmp.dir}/dfs/data
fs.checkpoint.dir	<value>/disk1/hdfs/name secondary,/disk2/hdfs/name</value>	A list of directories where the secondary name node stores checkpoints. It stores a copy of the checkpoint in each directory in the list. \${hadoop.tmp.dir}/dfs/namesecondary

# DEFINING MAPRED-SITES.XML

Property	Value	Description
Mapred.job.tracker	<value>localhost: 8021</value>	The hostname and port that the job tracker's RPC server runs on. If set to the default value of local, then the Job tracker is run in-process on demand when you run a Map Reduce job (you don't need to start the job tracker in this case, and in fact you will get an error if you try to start it in this mode)
Mapred.local.dir	{hadoop.tmp.dir}/mapred/local	A list of directories where Map Reduce stores intermediate data for jobs. The \${hadoop.tmp.dir}/mapred/local data is cleared out when the job ends.
Mapred.system.dir	{hadoop.tmp.dir}/mapred/system	The directory relative to fs.default.name where shared files are stored, during a job \${hadoop.tmp.dir}/mapred/system run.
Mapred.tasktracker.map.tasks.maximum	2	The number of map tasks that may be run on a tasktracker at any one time.
Mapred.tasktracker.reduce.tasks.maximum	2	The number of reduce tasks that may be run on a tasktracker at any one time.

# HADOOP

Start Hadoop

Format Name Node

`hadoop name node -format`

Start dfs service

`./start-dfs.sh`

Start mapred service

`./start-mapred.sh`

Verify services are running

`Jps`

# BASIC HDFS COMMANDS

Creating Directories

```
hadoop fs-mkdir <dirname>
```

Removing Directories

```
hadoop fs -rm <dirname>
```

Copying files to HDFS from Local file system

```
hadoop fs -copyFromLocal <local dir/filename> <hdfs dirname>/< hdfs filename>
```

Copying files from HDFS to local file system

```
hadoop fs -copyToLocal <hdfs dirname>/< hdfs filename> <local dir/filename>
```

List files and Directories

```
hadoop fs -ls <dirname>
```

List the blocks that make up each file in the file system

```
hadoop fsck / -files -blocks
```

# AGENDA

HDFS –  
General Purpose Tools

# HDFS – General Purpose Tools

HDFS Web UI

Job Tracker Web UI

Hadoop Maintenance Activities

Running Hadoop in Cluster

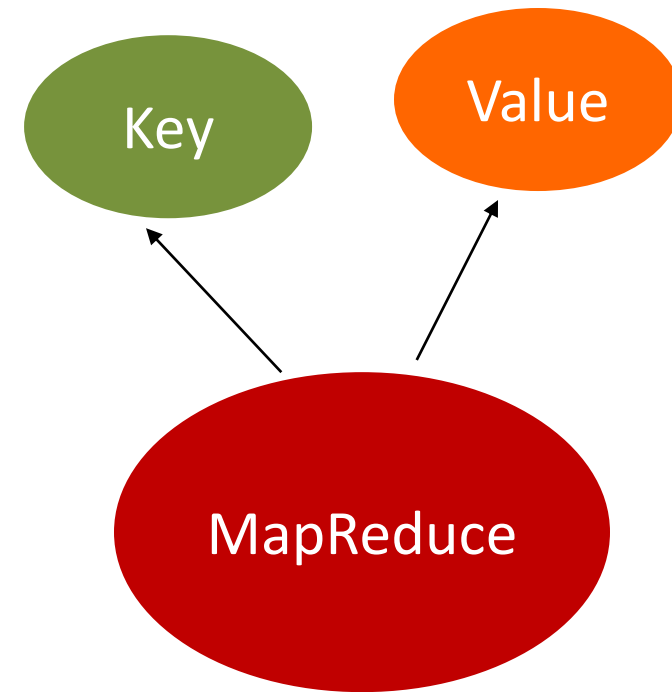
# MAP REDUCE

What is Map Reduce?

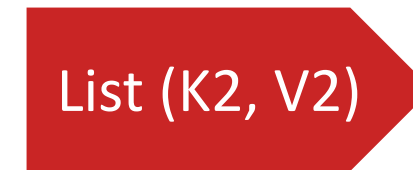
Map Reduce Workflow



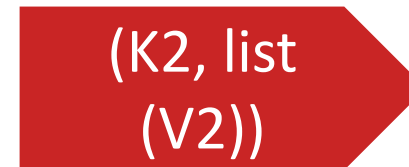
# MAP REDUCE



Map:



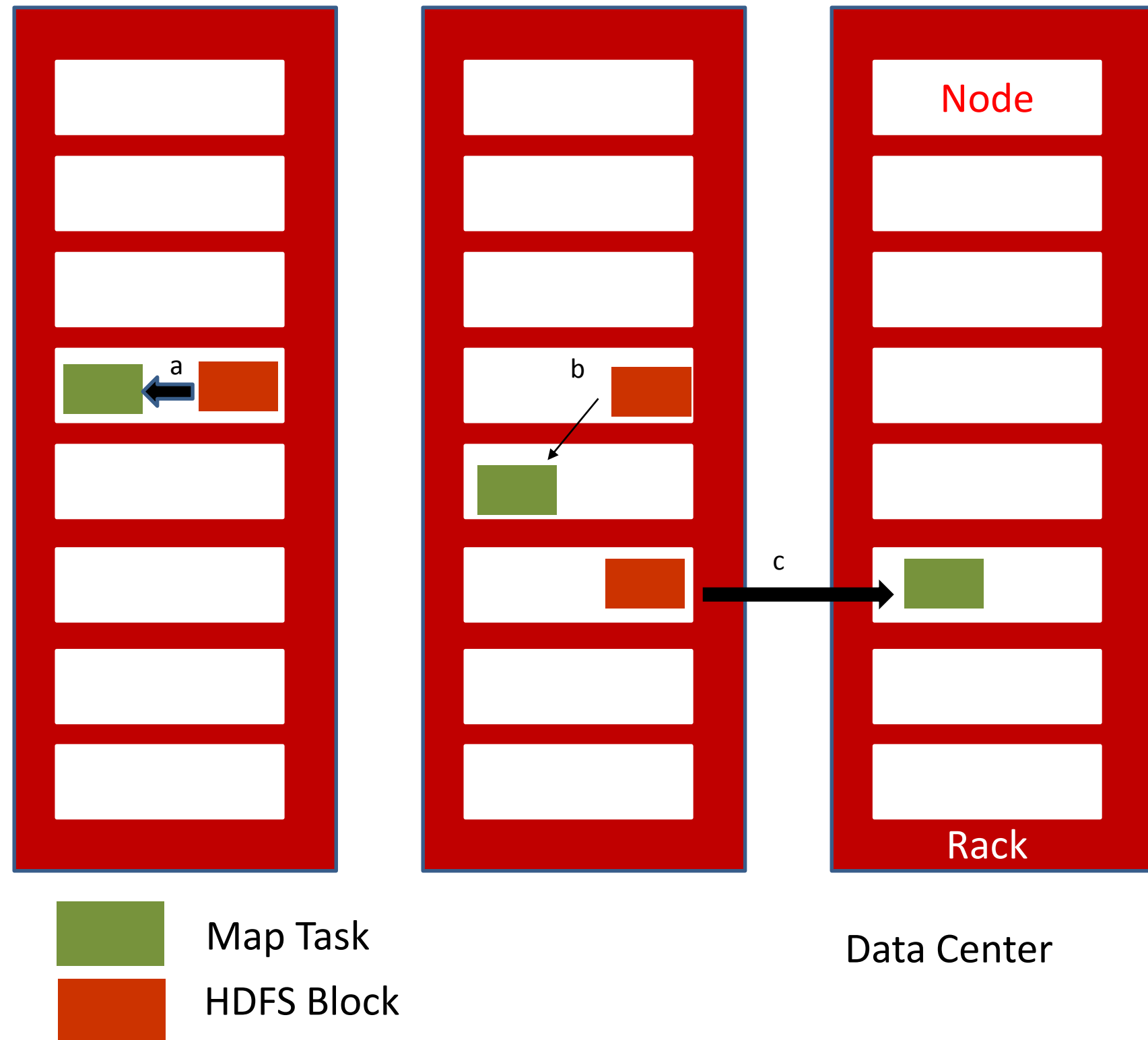
Reduce:



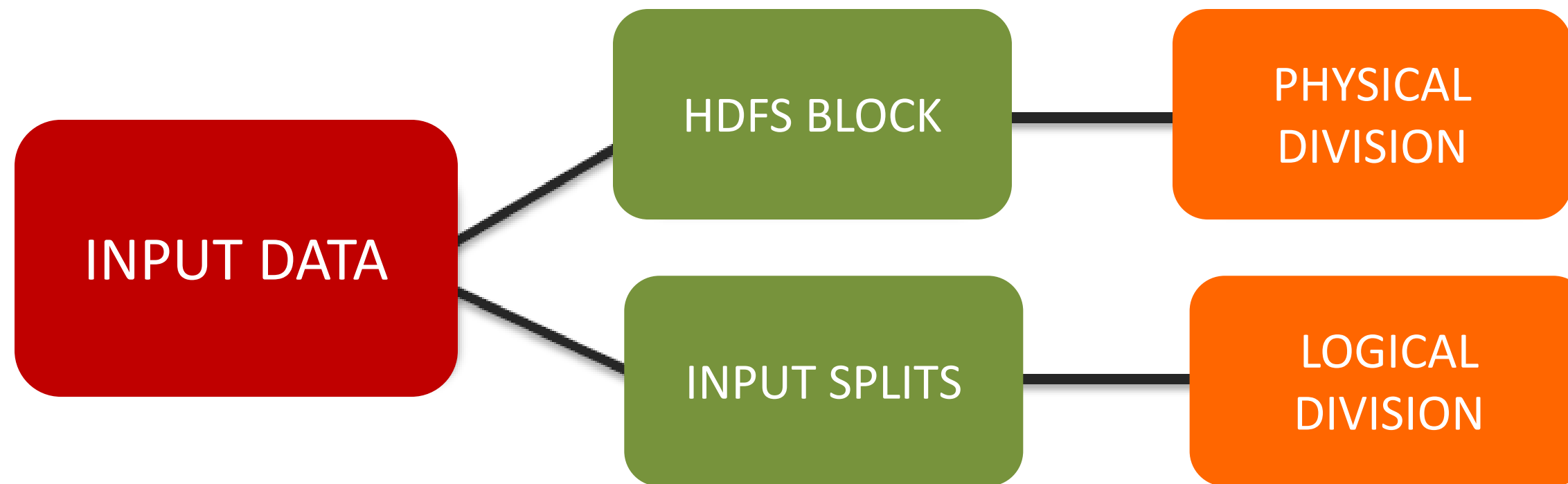
# MAP REDUCE - ADVANTAGE

✓ Two biggest Advantages:

- Taking processing to the data
- Processing data in parallel



# INPUT SPLITS



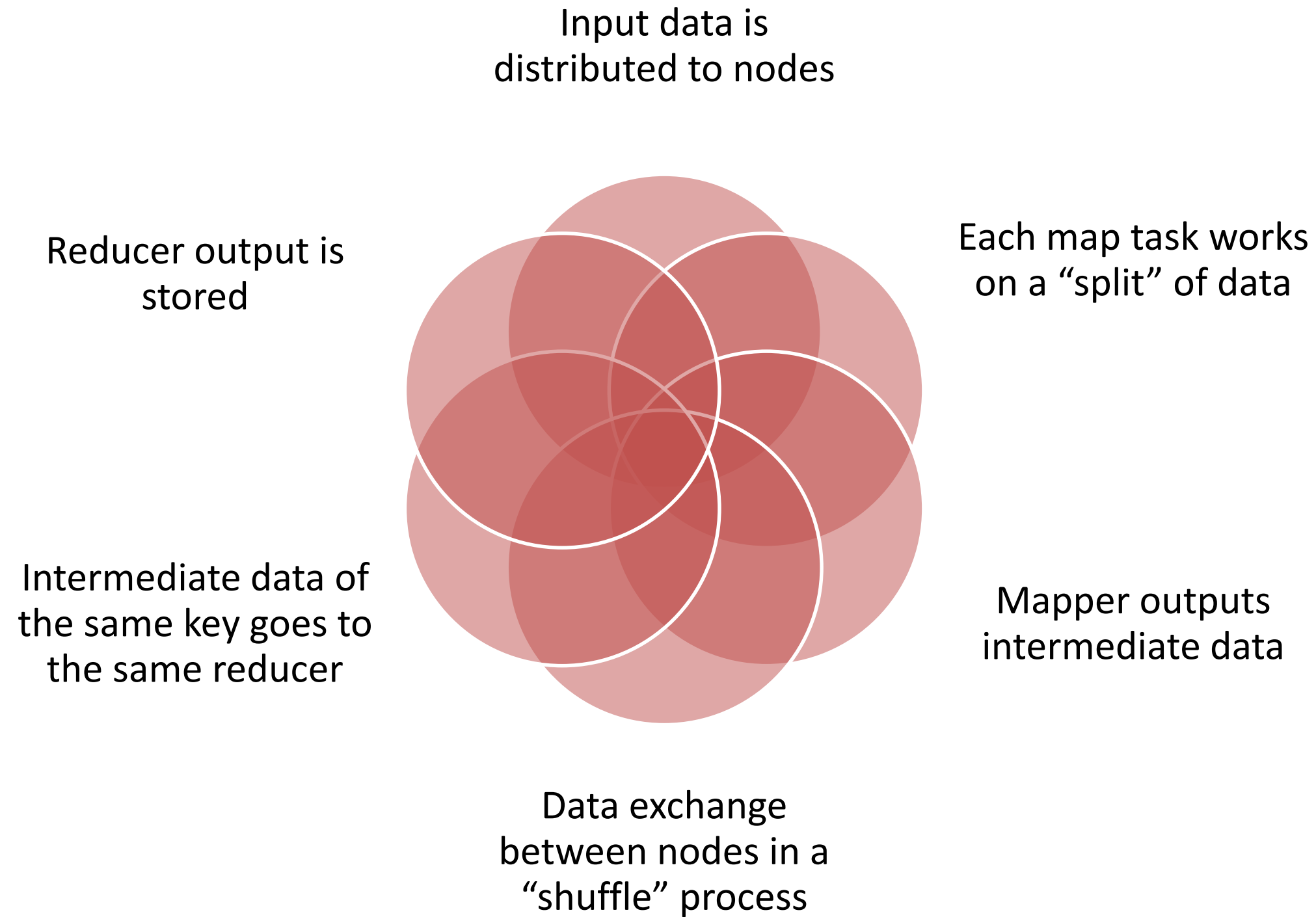
# MAP REDUCE

Logical records do not fit neatly into the HDFS blocks.

Logical records are lines that cross the boundary of the blocks.

First split contains line 5 although it spans across blocks.

# MAP REDUCE FLOW



# MAPPER

Reads data from input data split as per the input format

Denoted as Mapper<k1, v1, k2, v2>

k1, v1 are key value pair of input data

K2, v2 are key value pair of output data

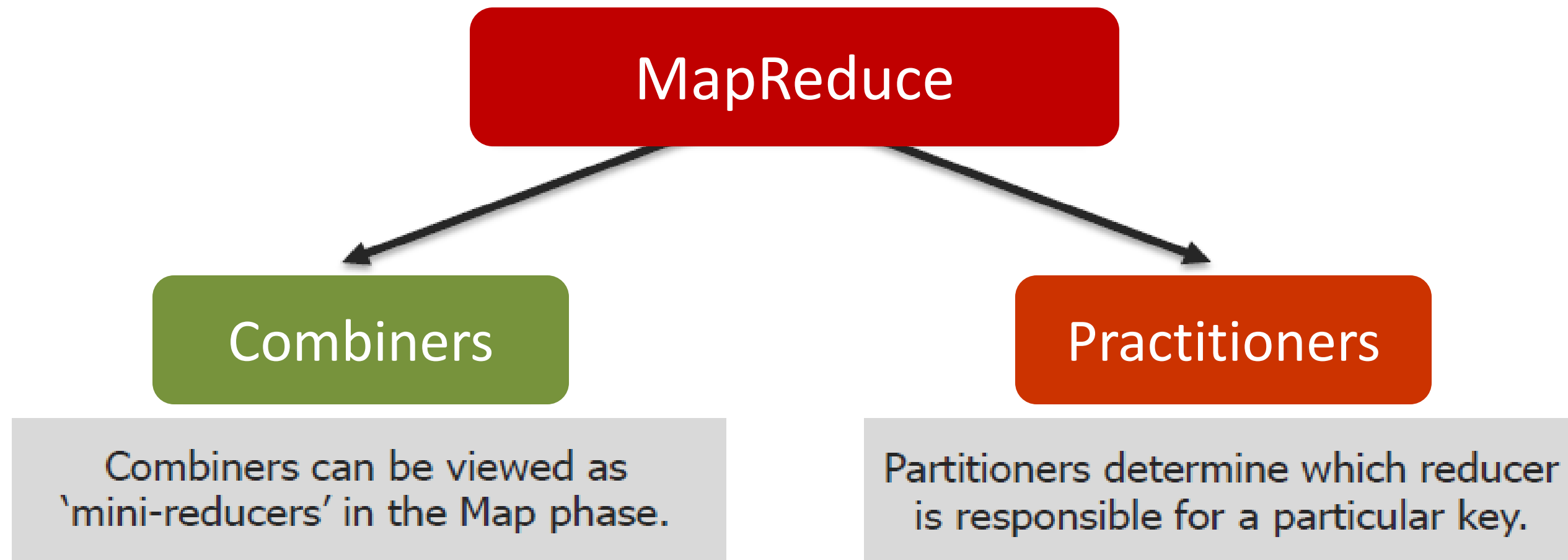
Hadoop will create k2 and List(v2)

And give the data to Reducer.

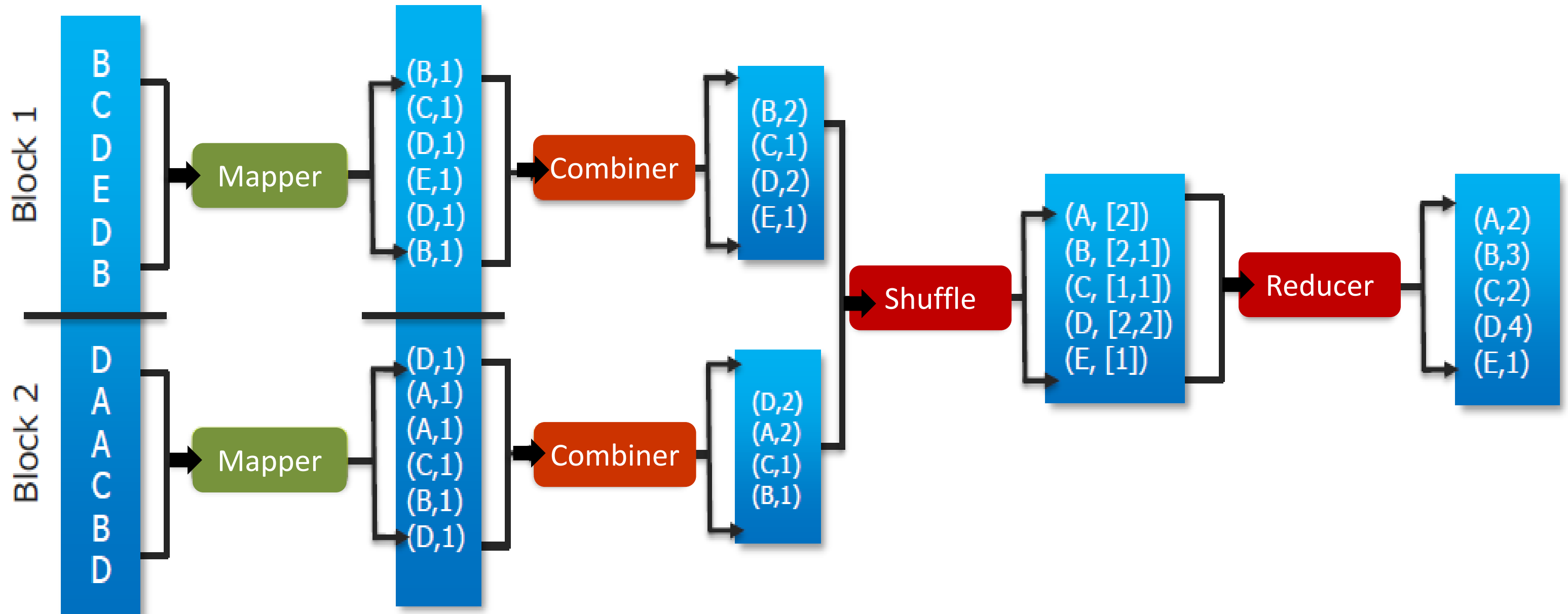
Package : org.apache.hadoop.mapreduce

# MAP REDUCE

Complete view of MapReduce, illustrating combiners and partitioners in addition to Mappers and Reducers

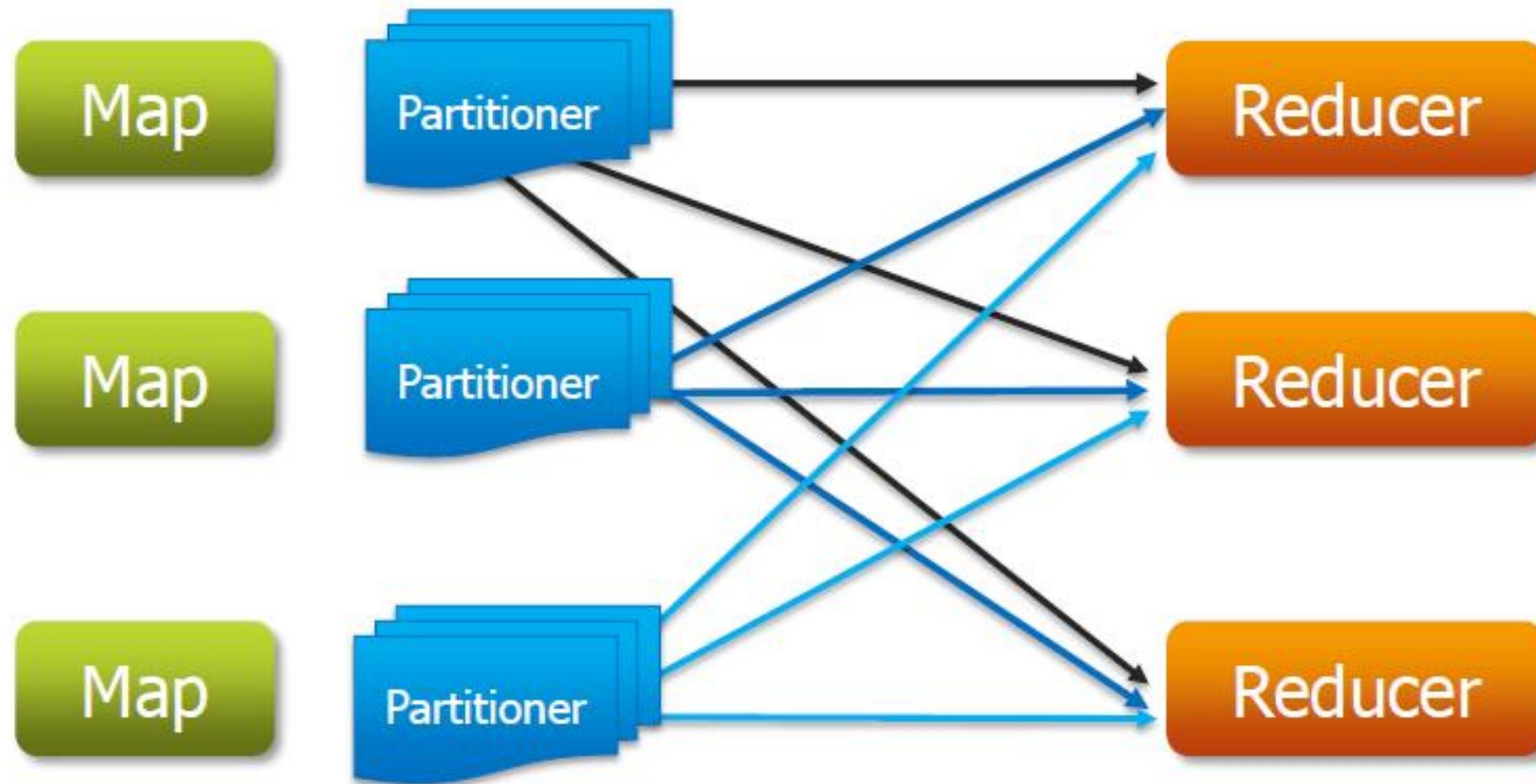


# MAPREDUCE - COMBINER





# MAPREDUCE PARTITIONER



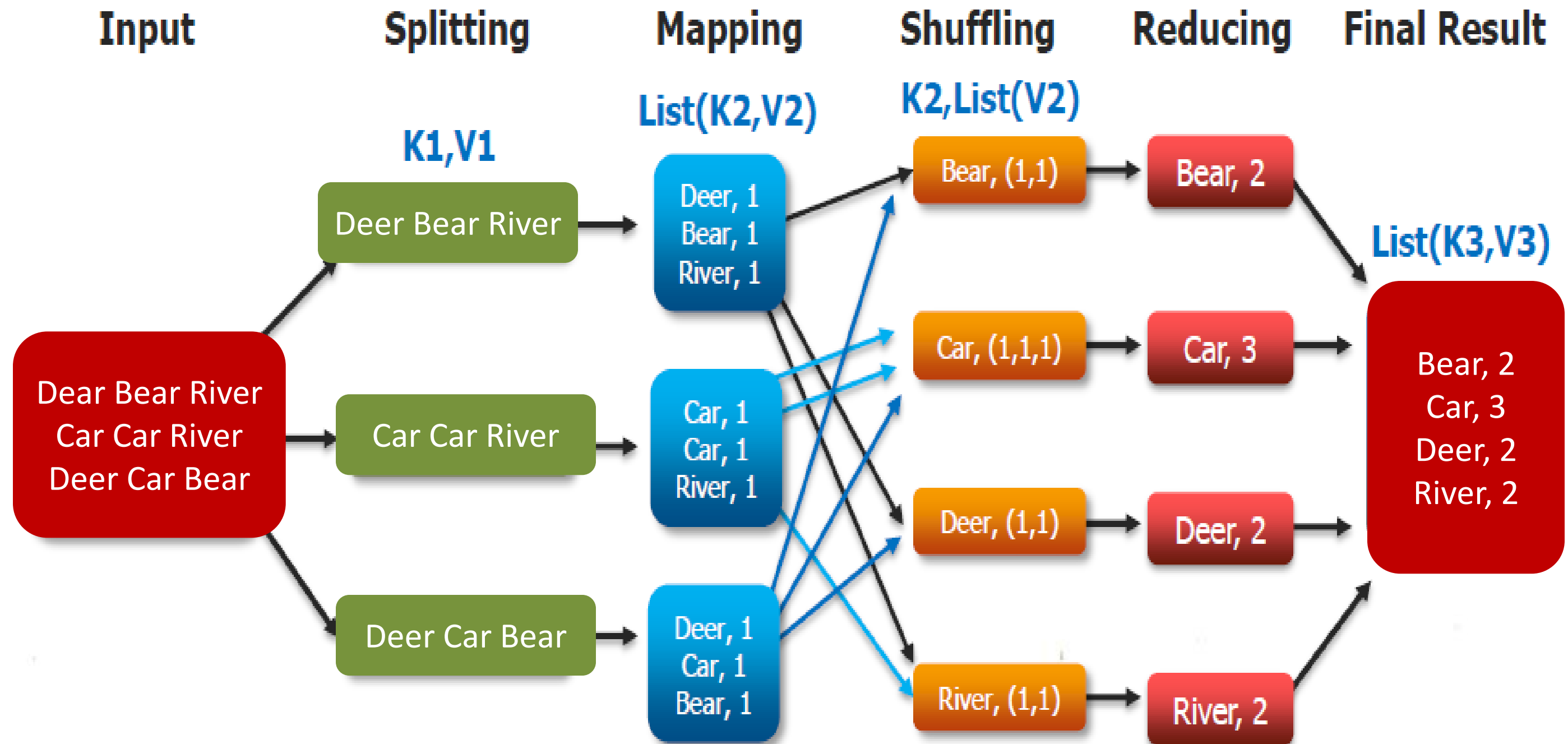
# MAPPER API

```
public class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable>  
<LogWritable, Text> key-value pair input to mapper  
<Text,IntWritable> key-value pair output of mapper
```

Override map ( ) method

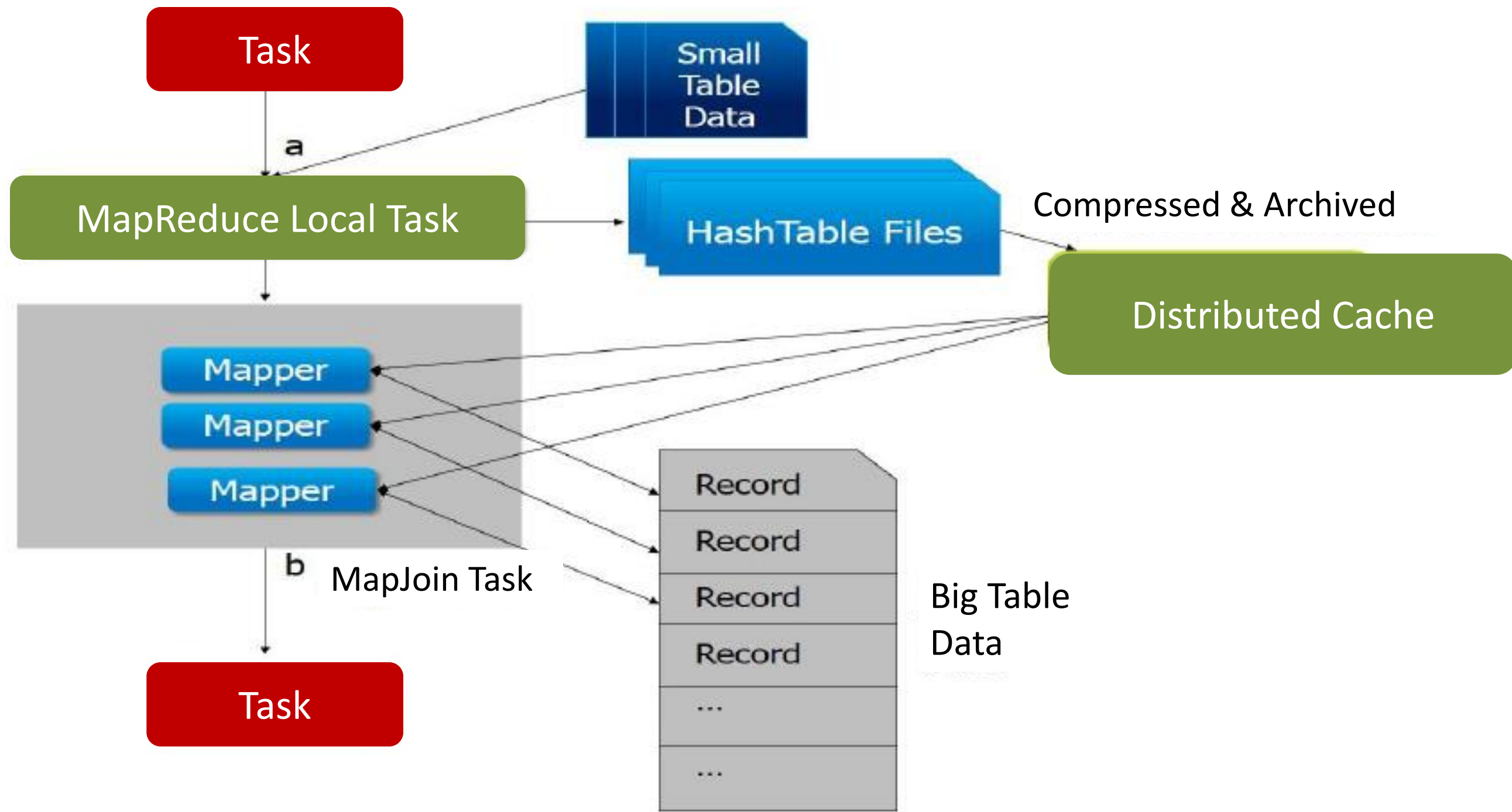
Public void map (Long Writable key, Text value, Context context) throws IO Exception,  
Interrupted Exception

# MAP REDUCE EXAMPLE

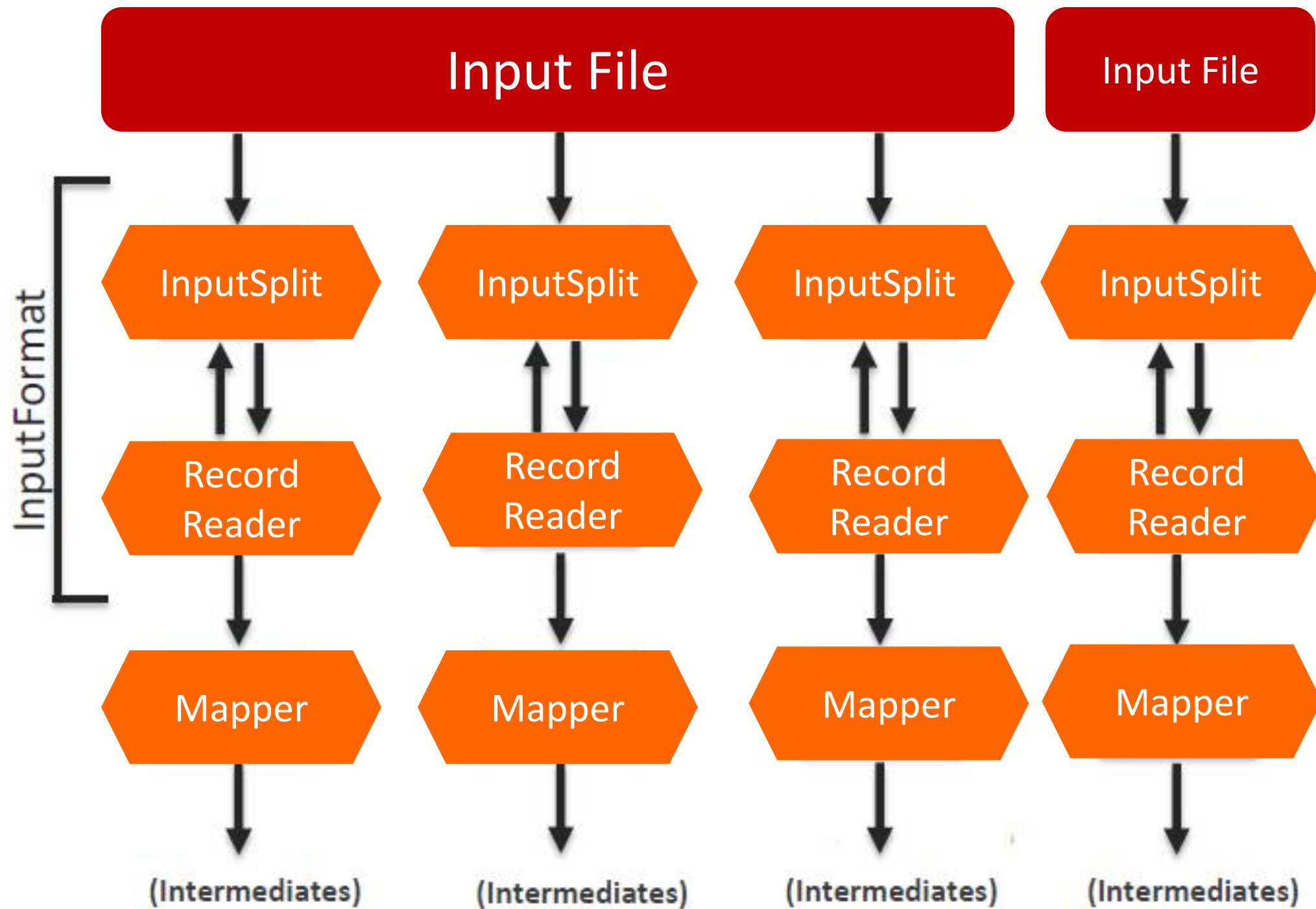


<http://stevekrenzel.com/finding-friends-with-mapreduce>

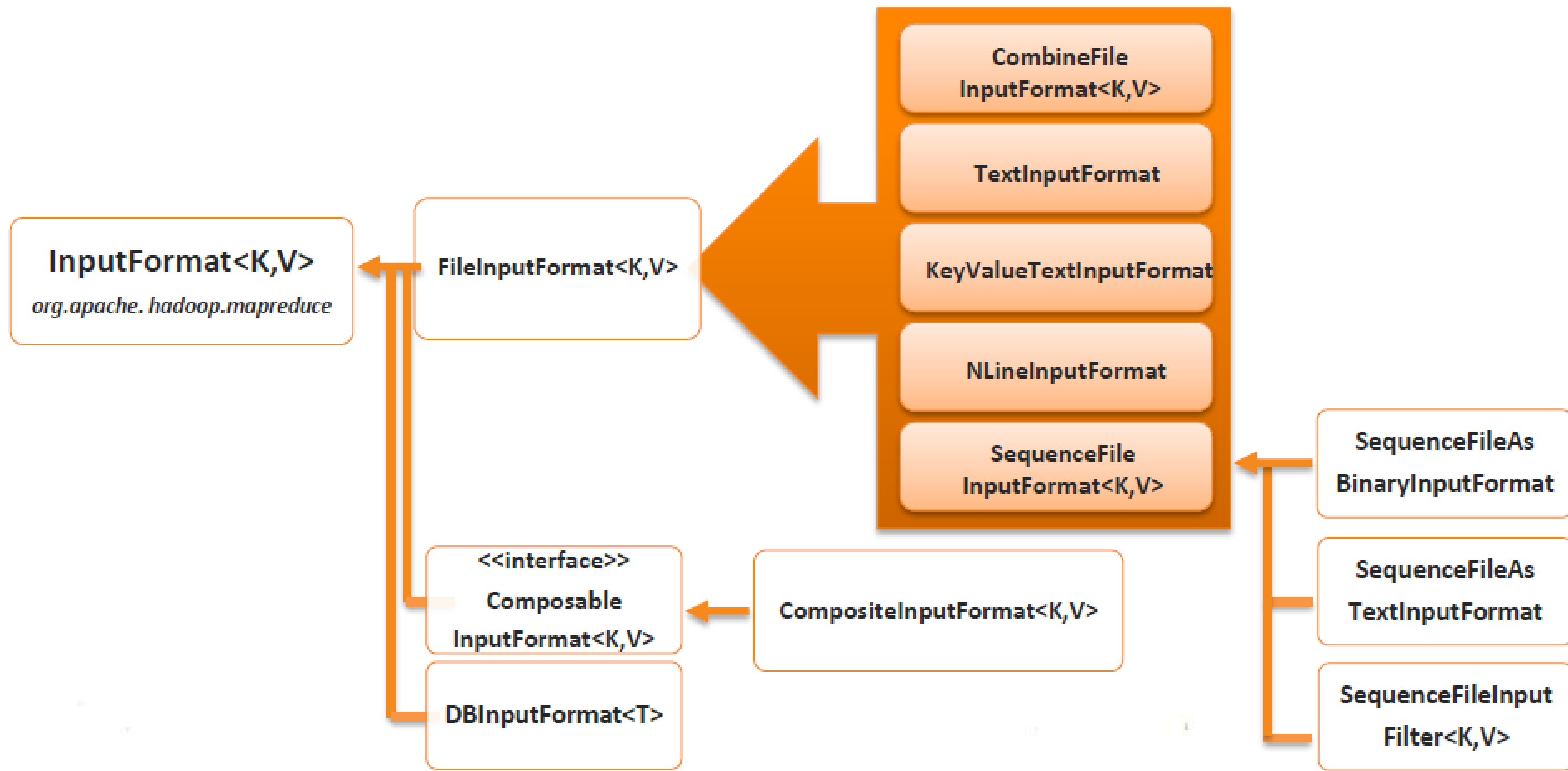
# MAP AND REDUCE SIDE JOINS



# INPUT FORMAT

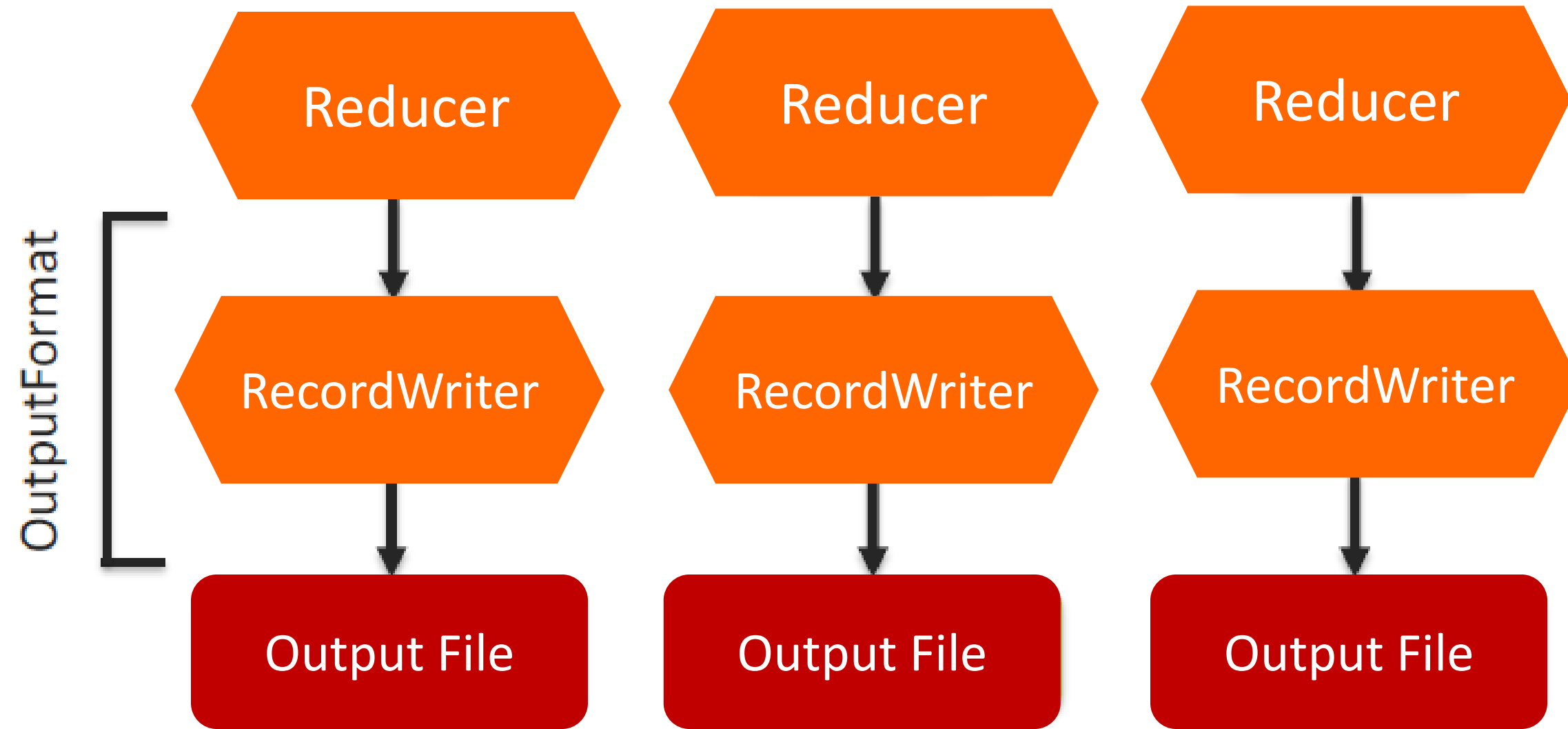


# INPUT FORMAT

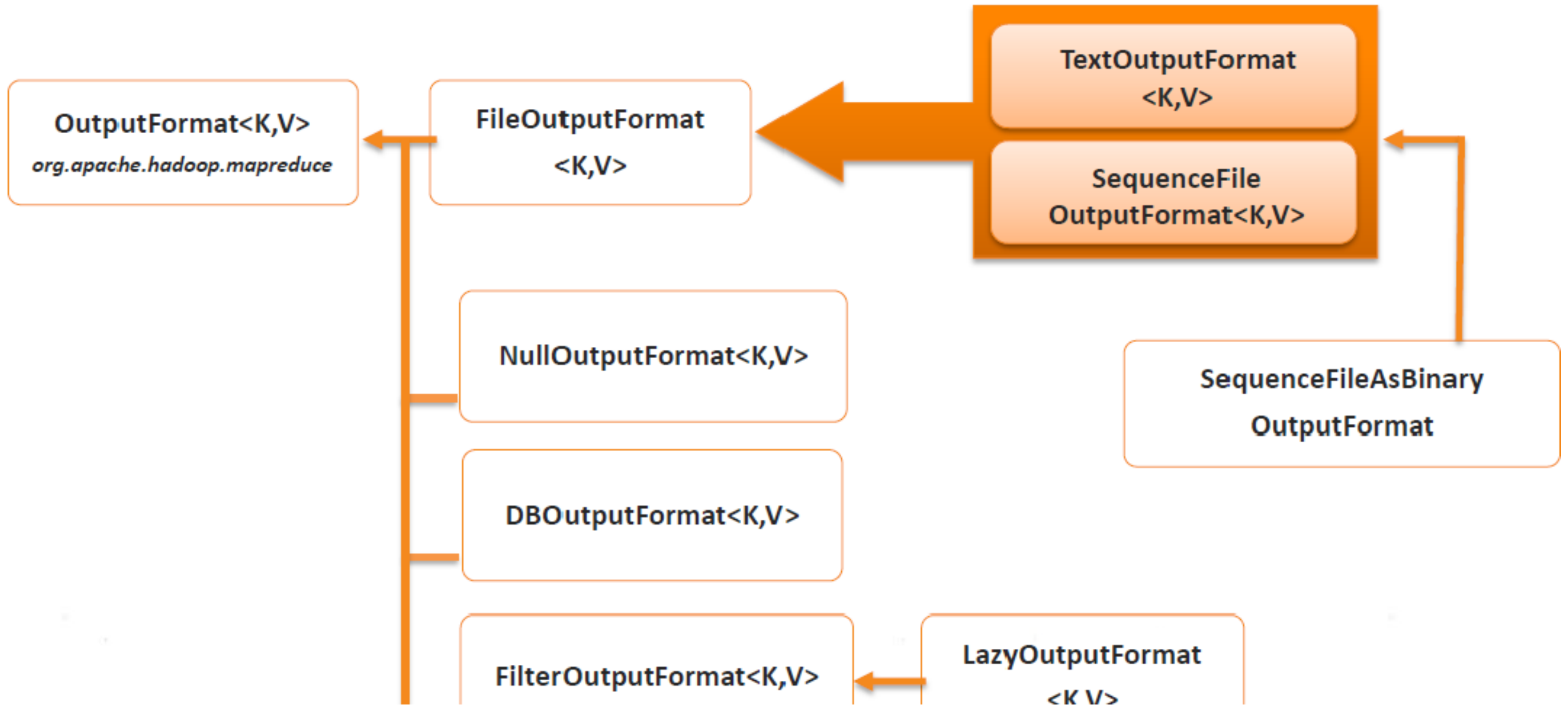




# OUTPUT FORMAT



# OUTPUT FORMAT



# MAP REDUCE

COUNTERS

DISTRIBUTED CAHCHE

Job start up – logical input split will be produced

Inout format – will decide what is the key and what is the value

# AGENDA

PIG

**PIG**

# PIG INSTALLTION

## Script:

Pig can run a script file that contains Pig commands.

Example: `pig script.pig` runs the commands in the Local file `script.pig`

## Grunt:

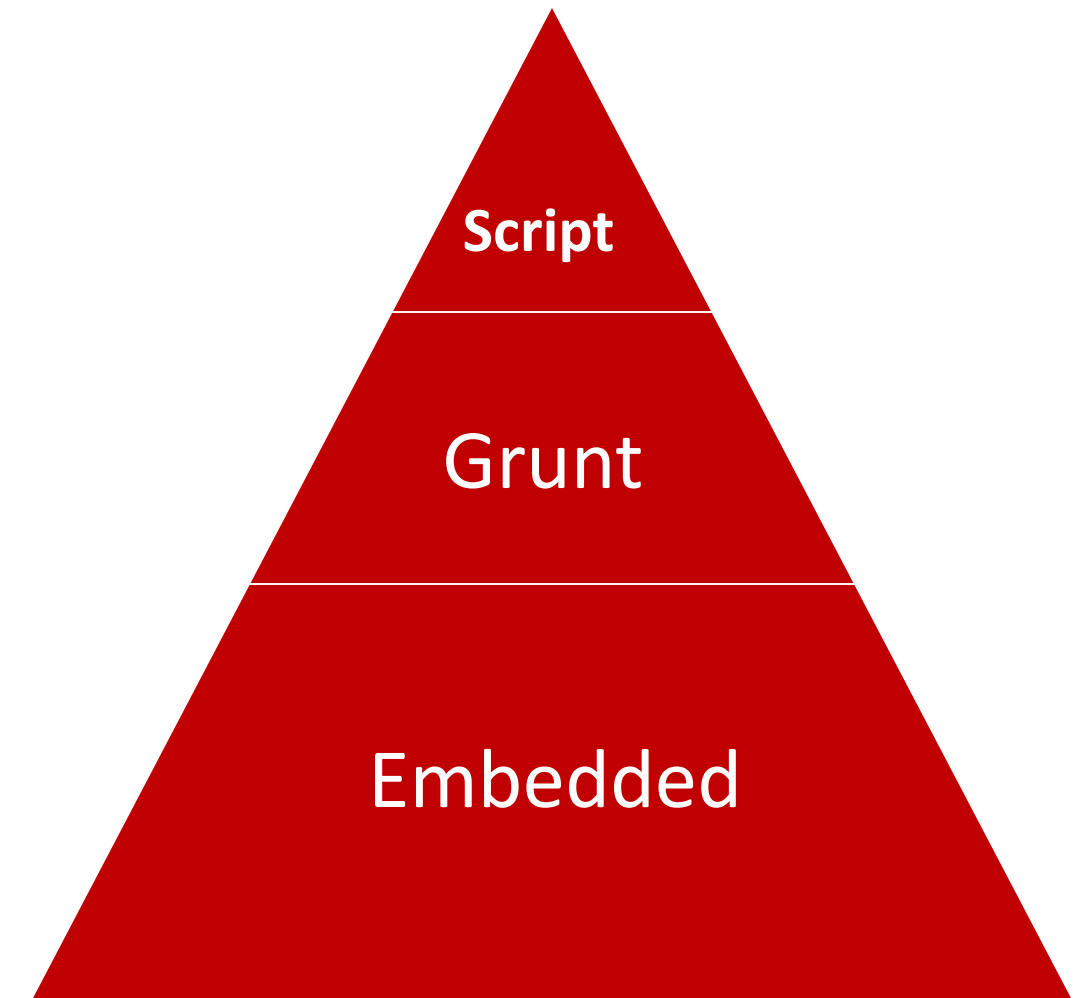
Grunt is an interactive shell for running Pig commands.

It is also possible to run Pig scripts from within Grunt

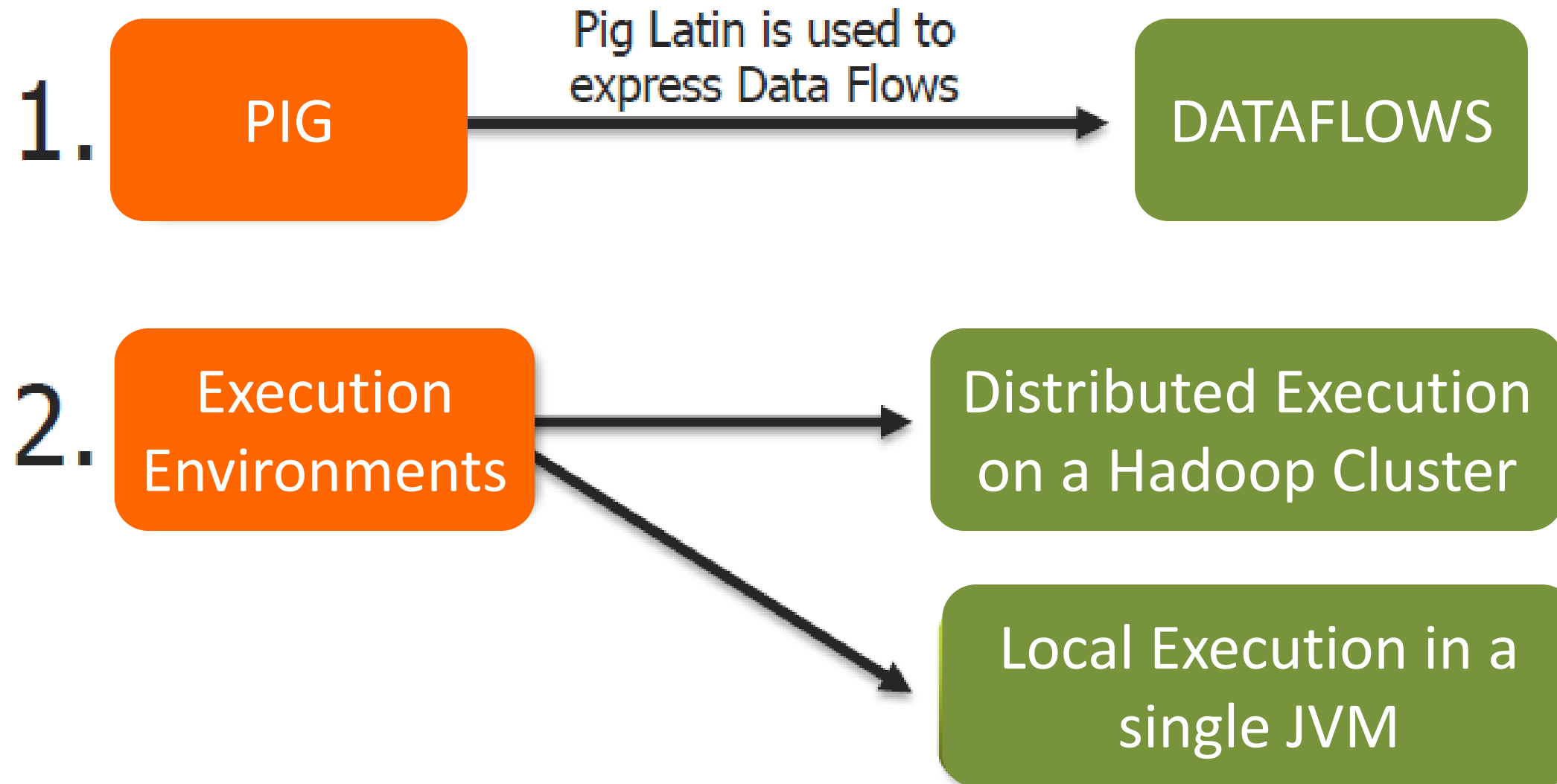
Using `run` and `exec` (execute).

## Embedded:

Embedded can run Pig programs from Java, much like you



# PIG COMPONENTS





# PIG

## WHERE NOT TO USE

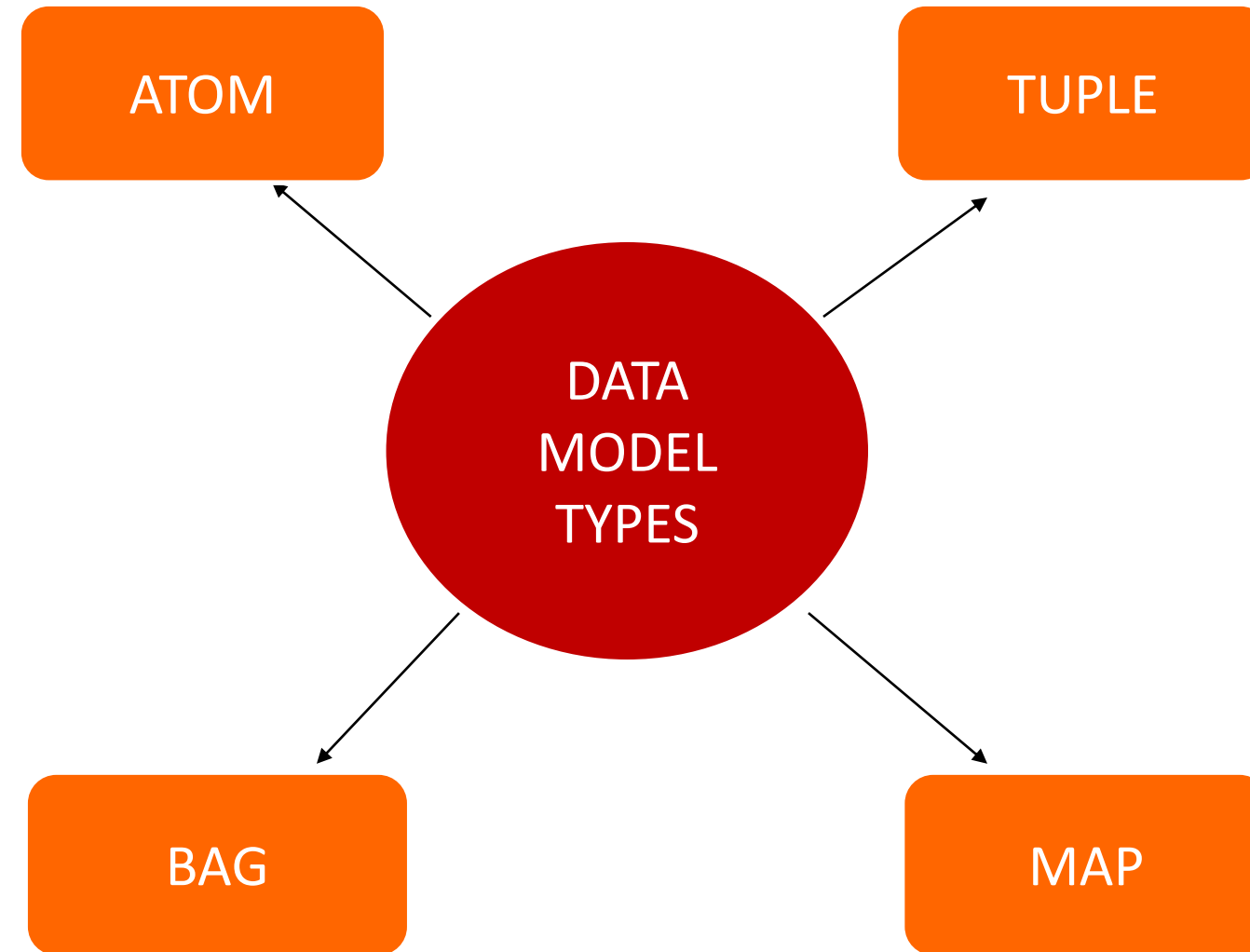
- COMPLETELY UNSTRUCTURED DATA
- ITS IS SLOW COMPARED TO CMPILED MAP REDUCE
- COMPLEX FILES – VIDEO, AUDIO

## WHERE TO USE

- Web logs
- Search platforms
- Adhoc queries
- Quick proto typing

# DATA TYPES

## PIG DATA TYPES



## PIG DATA TYPES

Pig Data Type	Implementing Class
Bag	org.apache.pig.data.DataBag
Tuple	org.apache.pig.data.Tuple
Map	java.util.Map<Object, Object>
Integer	java.lang.Integer
Long	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Chararray	java.lang.String

# PIG DATA TYPES

A **Data Atom** is a simple atomic data value. It is stored as a string but can be used as either a string or a number

Examples of data atoms are '[apache.org](http://apache.org)' and '1.0'.

A **Tuple** is a data record consisting of a sequence of "fields". Each field is a piece of data of any type (data atom, tuple or data bag). We denote tuples with < > bracketing. An example of a tuple is <[apache.org](http://apache.org),1.0>.

## What is a data bag?

A **Data Bag** is a set of tuples (duplicate tuples are allowed). You may think of it as a "table", except that Pig does not require that the tuple field types match, or even that the tuples have the same number of fields! (It is up to you whether you want these properties.) We denote bags by { } bracketing. Thus, a data bag could be {<[apache.org](http://apache.org),1.0>, <[flickr.com](http://flickr.com),0.8>}

## What is a data map?

A **Data Map** is a map from keys that are string literals to values that can be any data type. Think of it as a HashMap<String,X> where X can be any of the 4 pig data types. A Data Map supports the expected get and put interface. We denote maps by [ ] bracketing, with ":" separating the key and the value, and ";" separating successive key value pairs. Thus, a data map could be [ 'apache' : <'search', 'news'> ; 'cnn' : 'news' ]. Here, the key 'apache' is mapped to the tuple with 2 atomic fields 'search' and 'news', while the key 'cnn' is mapped to the data atom 'news'.

# PIG - FLOW

## PIG LATIN PROGRAM

- It is made up of a series of operations or transformations that are applied to the input data to produce output.



# PIG RELATIONAL OPERATIONS

Category	Operator	Description
Loading and Storing	LOAD STORE DUMP	Loads data from the file system or other storage into a relation . Saves a relation to the file system or other storage. Prints a relation to the console.
Filtering	FILTER DISTINCT FOREACH...GENERATE STREAM	Removes unwanted rows from a relation. Removes duplicate rows from a relation. Adds or removes fields from a relation. Transforms a relation using an external program.
Grouping and Joining	JOIN COGROUP GROUP CROSS	Joins two or more relations. Groups the data in two or more relations. Groups the data in a single relation. Creates the cross product of two or more relations.
Sorting	ORDER LIMIT	Sorts a relation by one or more fields. Limits the size of a relation to a maximum number of tuples.
Combining and Splitting	UNION SPLIT	Combines two or more relations into one. Splits a relation into two or more relations.

# PIG

10 Lines of Pig ~ 200 lines of java code

Load

Store

Filter

Join

Order

Sort

**High level built in operators to manage the data in PIG.**

**Run time executions converts to Map reduce and execute it.**

Adhoc way of creating and executing map reduce jobs on very large data sets

Rapid Development, No java coding, Developed by Yahoo

# PIG

Pig – is high level declarative language

It is a data flow language



# Processing in Hadoop

1. Load emp.data : load – specify the file name for processing.
2. Process (filter based on the data, order)
3. Dump / store
4. Dump : dumps the result to the console
5. Store : stores results to the file

loaddata = load '/emp.data'

Filtereddata = filter loaddata

Ordereddata = order filtereddata

Dump ordereddata

Store ordereddata Store results to HDFS

# PIG

You can mix with the map reduce programm.

It works on top of Hadoop. We can create complex jobs. Large volumes – quick process.

Time sensitive data loads

Processing many data sources

Analytic insight through sampling

Pig supports many relational operators.

Ability to perform sampling. = first 100

Pig can call UDF = User defined functions

Pig is slow compared to map reduce – in execution.

# PIG - Uses

Processing Web logs

Data processing for search algorithms

Support for adhoc queries

Used for ETL – Transforming of the data

PIG Latin commands

Execution environments : Distributed execution on Hadoop cluster

No need to install any thing extra on the cluster

# PIG EXECUTION

DUMP of STORE command will generate Logical plans

All the commands in the script will be executed.

# PIG LATIN FILE LOADERS

## PIG LATIN FILE LOADERS

- ☐ PigStorage – Loads and Stores data that is delimited by something
- ☐ TextLoader – Loads data line by line (delimited by the newline character)
- ☐ CSVLoader – Loads CSV files
- ☐ XML Loader – Loads XML files

# SQOOP

- What is Sqoop?
- Install & configure sqoop
- Generate Code
- Sqoop Import
- Sqoop Export

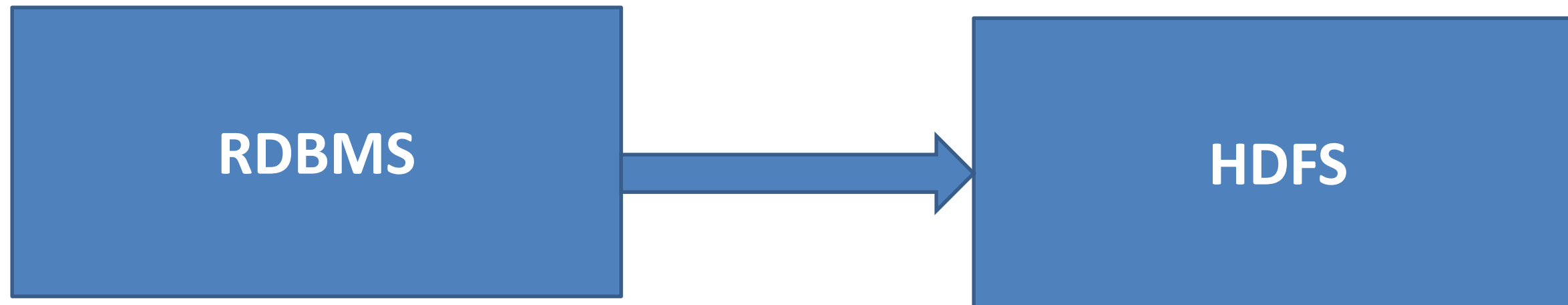
# SQOOP

## Sqoop Overview

Is a JDBC implementation, so it works with most of the databases

Integrates with Hive

Generates code that can be used for map reduce applications



# SQOOP

Define sqoop home

`SQOOP_HOME=<sqoop installation path>`

Set Path

`PATH=$PATH:$SQOOP_HOME/bin`

Copy mysql driver jar file to sqoop library location

`/home/ubuntu/sqoop-l. 4.0-incu bating/lib`



## SQOOP CODE GEN

- ❑ Generate class as per the database schema
- ❑ Sqoop codegen -connect jdbc:mysql://local host/big —table <table-name> -username <username> —password <password> -class-name <class name>
- ❑ Can use with map reduce programs

# SQOOP IMPORT

## Sqoop import

**--connect jdbc: mysql://local host/<database name>**

**--table <table-name>**

**--username <user-name>**

**-password <password>**

**-m 1**

**-target-dir output/sqoop**

**Where, m is no of map program to be run**

## SQOOP EXPORT

Sqoop export

--connect jdbc:mysql://local host/<database-name>

--table <table-name>

-username <user-name>

-password <password>

-export-dir <directory>

Where, m is no of map program to be run

# AGENDA

## HIVE

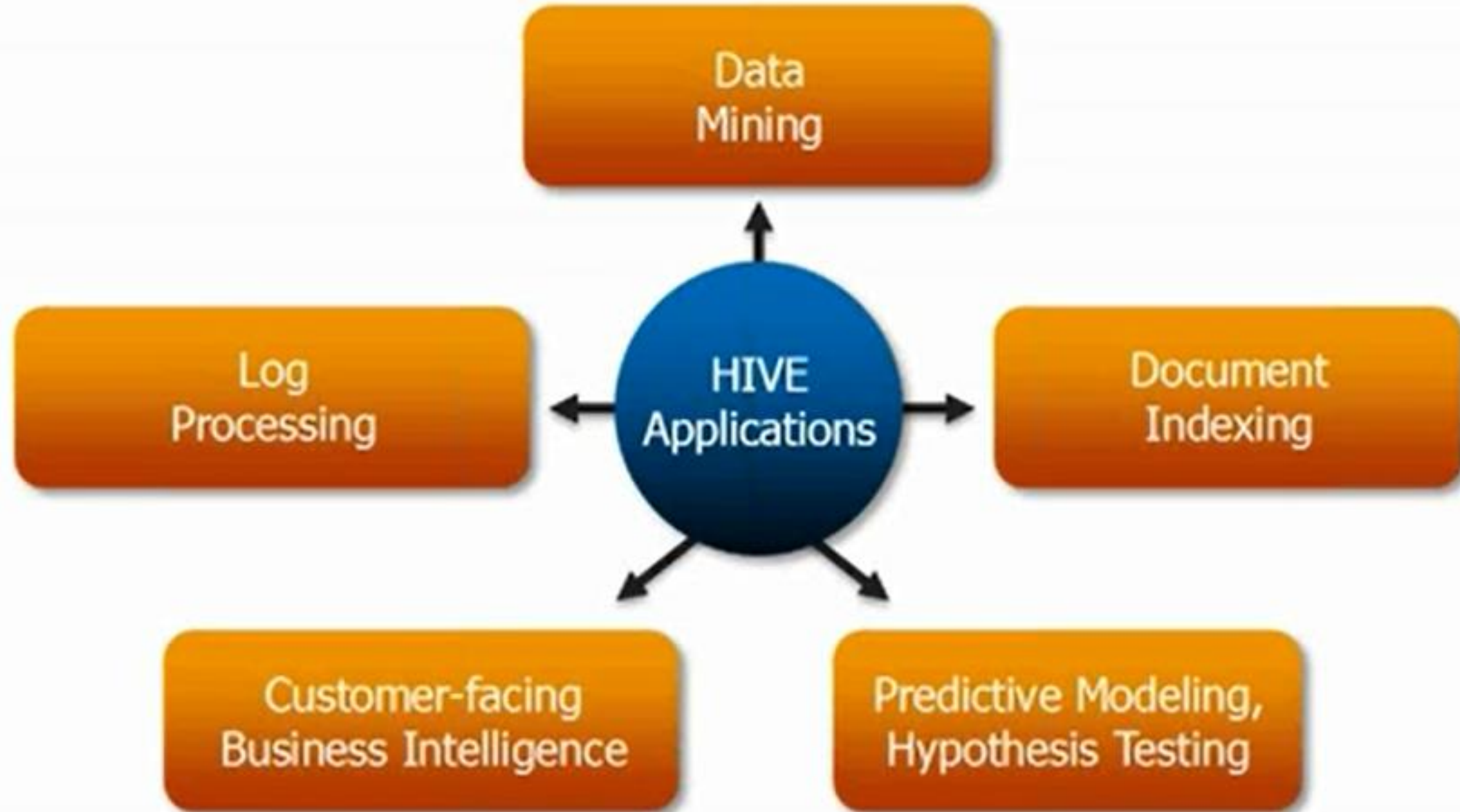
# HIVE

- Data warehousing package built on top of hadoop
- Used for data analysis
- Targeted towards users comfortable with SQL
- It is similar to SQL and call HiveQL
- For managing and querying structured data
- Abstracts complexity of Hadoop
- No need learn Java and Hadoop APIs
- Developed by Face book and contributed to community
- Hive automatically stores and manages data for users

# HIVE

- ✓ Data Warehousing package built on top of Hadoop
- ✓ Used for data analysis
- ✓ Targeted towards users comfortable with SQL
- ✓ It is similar to SQL and called HiveQL
- ✓ For managing and querying structured data
- ✓ Abstracts complexity of Hadoop
- ✓ No need learn java and Hadoop APIs
- ✓ Developed by Facebook and contributed to community
- ✓ Facebook analyzed several Terabytes of data everyday using Hive

# WHERE TO USE HIVE





## PIG/HIVE

Features	Hive	Pig
Language	SQL-like	PigLatin
Schemas/Types	Yes (explicit)	Yes (implicit)
<b>Partitions</b>	<b>Yes</b>	<b>No</b>
Server	Optional (Thrift)	No
User Defined Functions (UDF)	Yes (Java)	Yes (Java)
Custom Serializer/Deserializer	Yes	Yes
DFS Direct Access	Yes (implicit)	Yes (explicit)
Join/Order/Sort	Yes	Yes
Shell	Yes	Yes
Streaming	Yes	Yes
Web Interface	Yes	No
JDBC/ODBC	Yes (limited)	No



# HIVE COMPONENTS

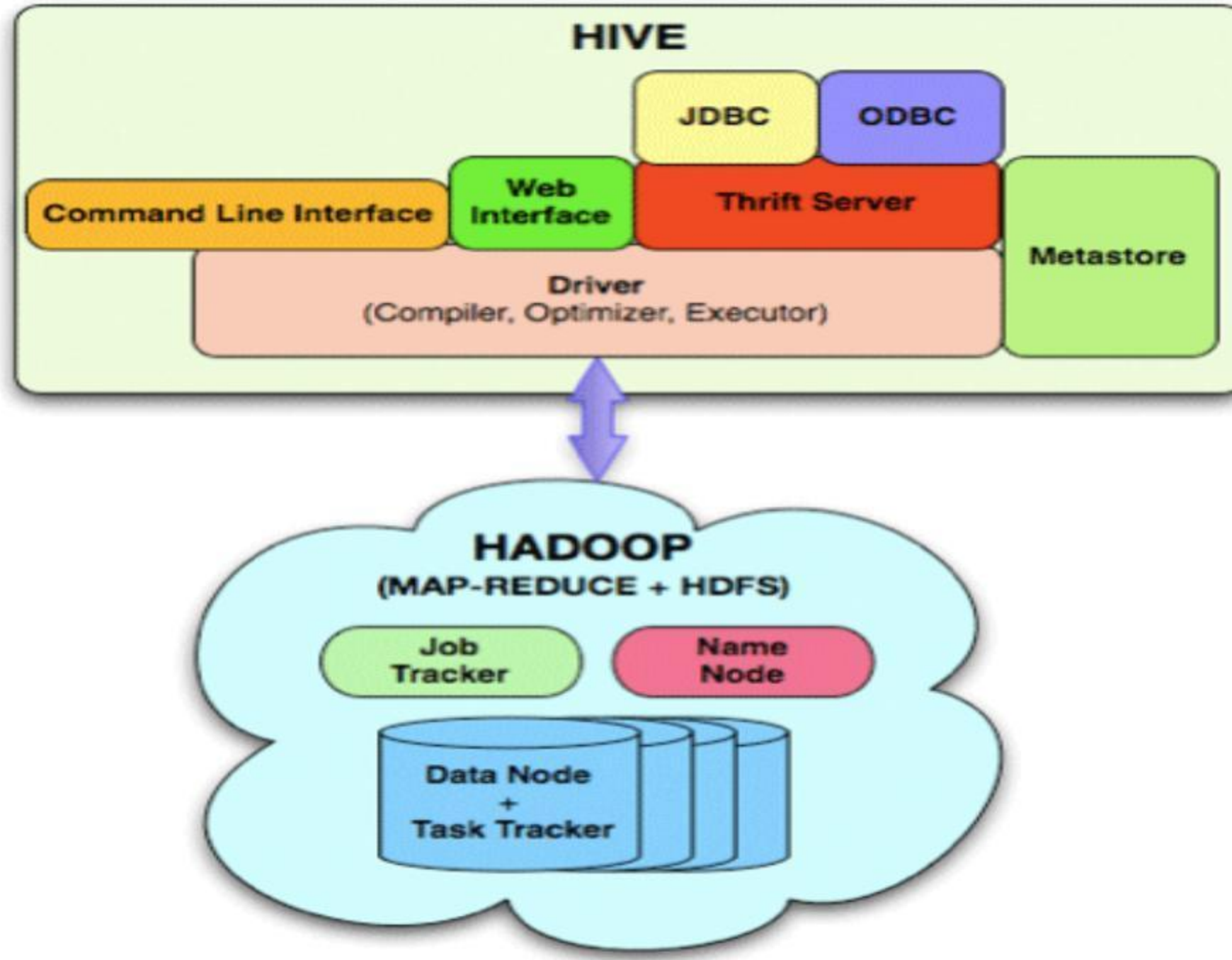
**Hive has 3 main components:**

**Serializers/Deserializers** - This component has the framework libraries that allow users to develop serializers and deserializers for their own data formats. This component also contains some built in serialization/deserialization families.

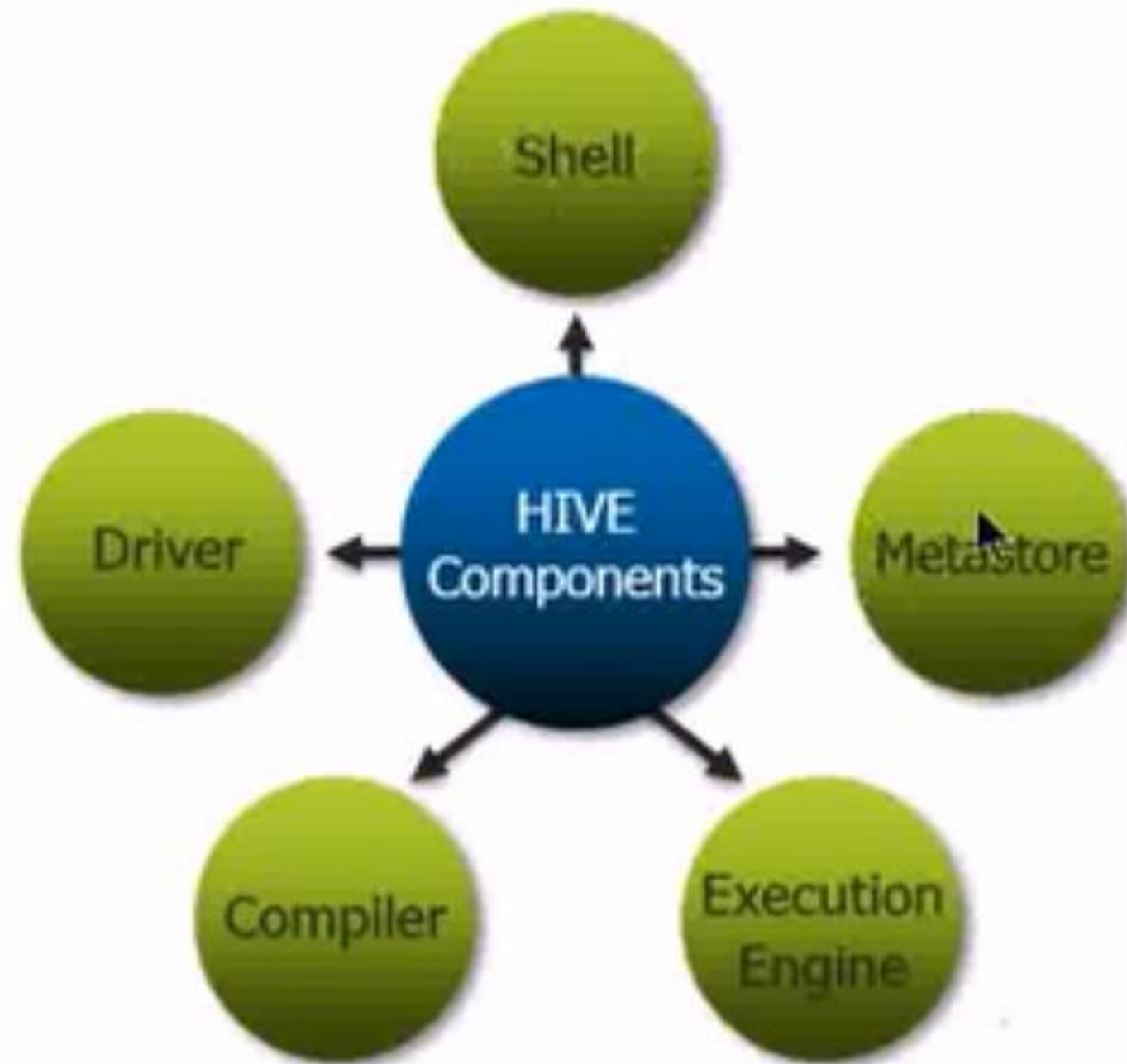
**MetaStore (trunk/metastore)** - This component implements the metadata server, which is used to hold all the information about the tables and partitions that are in the warehouse.

**Query Processor (trunk/ql)** - This component implements the processing framework for converting SQL to a graph of map/reduce jobs and the execution time framework to run those jobs in the order of dependencies.

# HIVE COMPONENTS



# HIVE COMPONENTS



# HIVE COMPONENTS

DRIVER – PARSES – MANAGES SHOW

COMPILER – CREATES EXECUTION PLAN, CREATES DAG OF MAP REDUCE JOBS

EXECUTION ENGINE – USES HADOOP TO EXECUTE THE JOBS

META STORE : DERBY DATABASE TO STORE META DATA

<https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties>

# HIVE DATA MODELS

## Hive Data Models

- Databases
  - Namespaces
- Tables
  - Schemas in namespaces
- Partitions
  - How data is stored in HDFS.
  - Grouping data bases on some column
  - Can have one or more columns
- Buckets or Clusters
  - Partitions divided further into buckets bases on some other column.
  - Use for data sampling

# HIVE DATA TYPES

## Hive Data Types - Simple Types

- TINYINT-1 byte integer.
- SMALLINT - 2 byte integer
- INT - 4 byte integer
- BIGINT-8 byte integer
- BOOLEAN - TRUE/FALSE
- FLOAT - single precision
- DOUBLE - Double precision
- STRING - sequence of characters in a specified character set

# HIVE DATA TYPES

## Hive DataType – Complex Types

- Structs

Column c of type struct {a INT; INT} the a field is accessed by the expression c.a

- Maps (key-value tuples)

A map M comprising of a mapping from 'group' -> gid the value can be accessed using M['group']

- Arrays (indexable lists)

An array A having elements [a,b,c] returns 'b'

# HIVE

## External Tables

- Create the table in another hdfs location and not in warehouse directory
- Not managed by hive

```
CREATE EXTERNAL TABLE external table (dummy STRING)
```

```
LOCATION '/user/tom/external table';
```

- Hive does not delete the table (or hdfs files) even when the tables are dropped. It leave the table untouched and only metadata about the tables are deleted



# HIVE

## LOAD Data

- Load the data into the table

```
LOAD DATA LOCAL INPATH '/home/ubuntu/work/data/txn.csv'
```

```
OVERWRITE INTO TABLE txnrecords;
```

- Describing metadata or schema of the table

```
Describe txnrecords;
```

# HIVE

## Managing Tables

- Loading Data

- LOAD DATA LOCAL INPATH /<path>/<filename> INTO TABLE <table name>
- LOAD DATA LOCAL INPATH /<path>/<filename> INTO TABLE <table name> PARTITION (designation='developers') *only Loads a specific partition*
- SHOW tables;
- Show partitions <table name>;
- Describe <table name>;

# HIVE

## Create database and table

- **Create database**

Create database retail;

- **Use database**

Use retail;

- **Create table for storing transactional records**

```
create table txnrecords (txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state STRING, spend By STRING )
```

Row format delimited

Fields terminated by ',' stored as text file;

# HIVE

## Managing Tables

- **Altering a table**

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

```
ALTER TABLE tabl ADD COLUMNS (c1 INT COMMENT 'a new int column', c2 STRING DEFAULT 'defval');
```

- **Dropping a table or a partition**

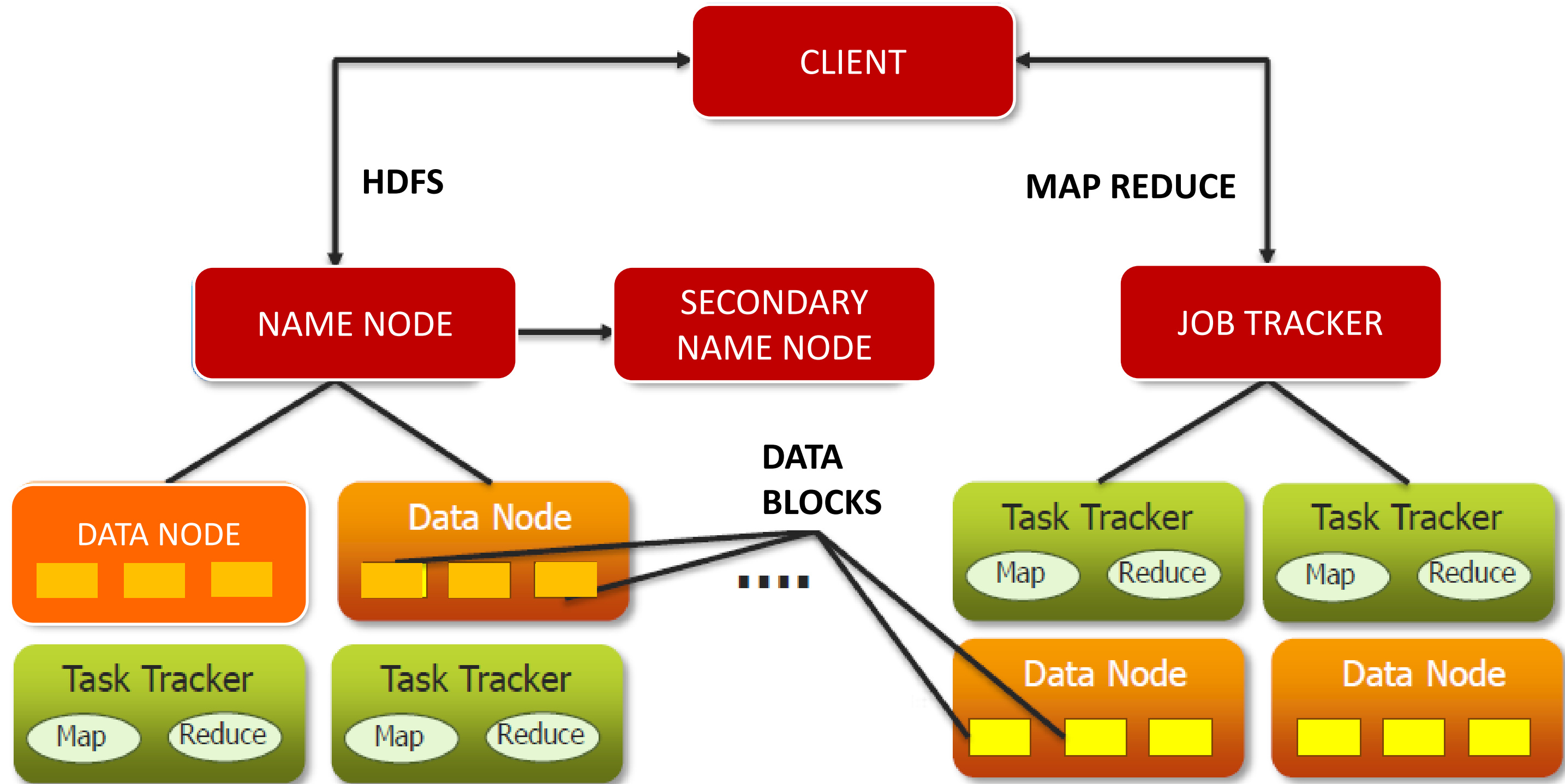
```
DROP TABLE pv_users;
```

```
ALTER TABLE pv_users DROP PARTITION (ds='2008-08-08')
```

# AGENDA

## HADOOP 2.0

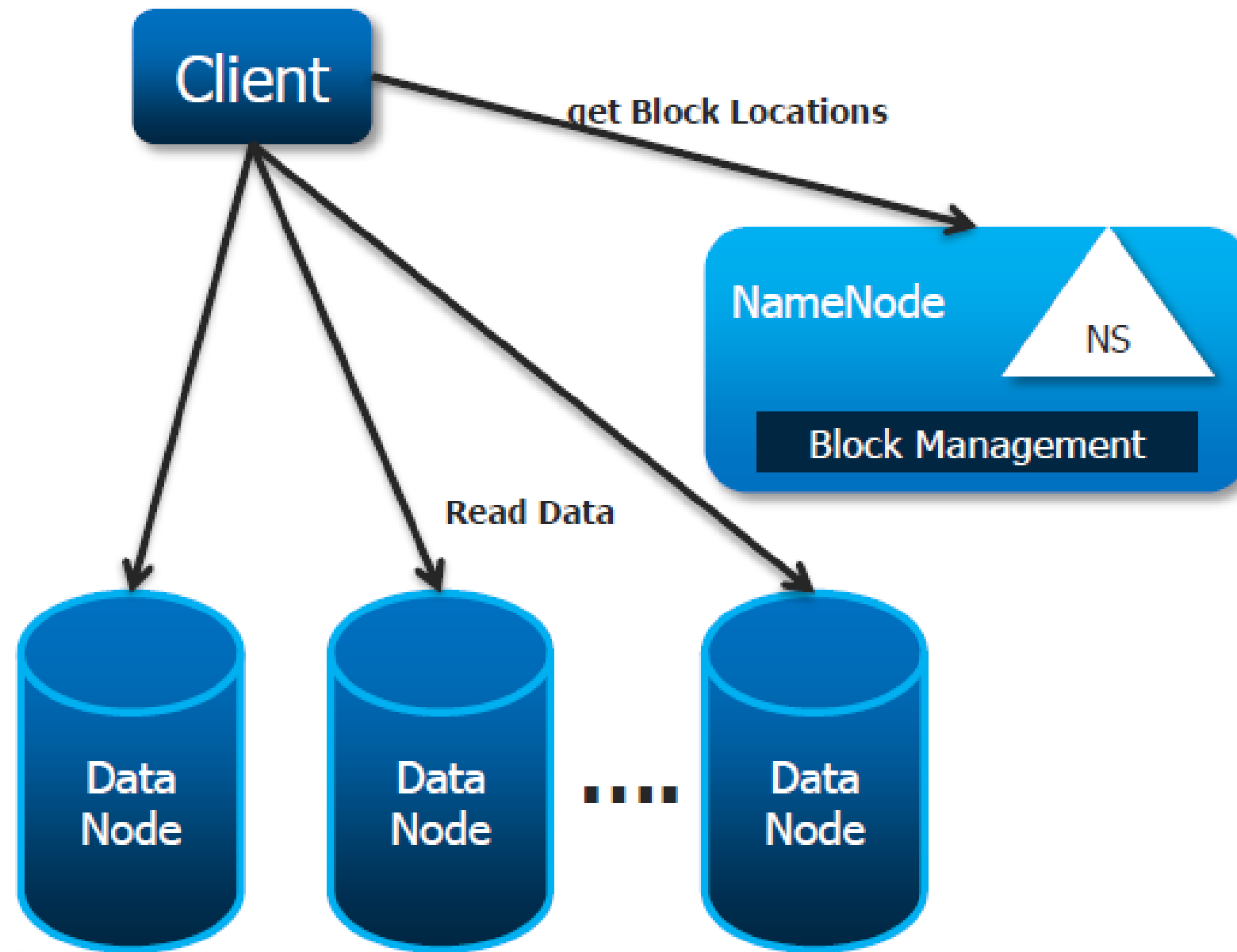
# HADOOP 1.0 SUMMARY



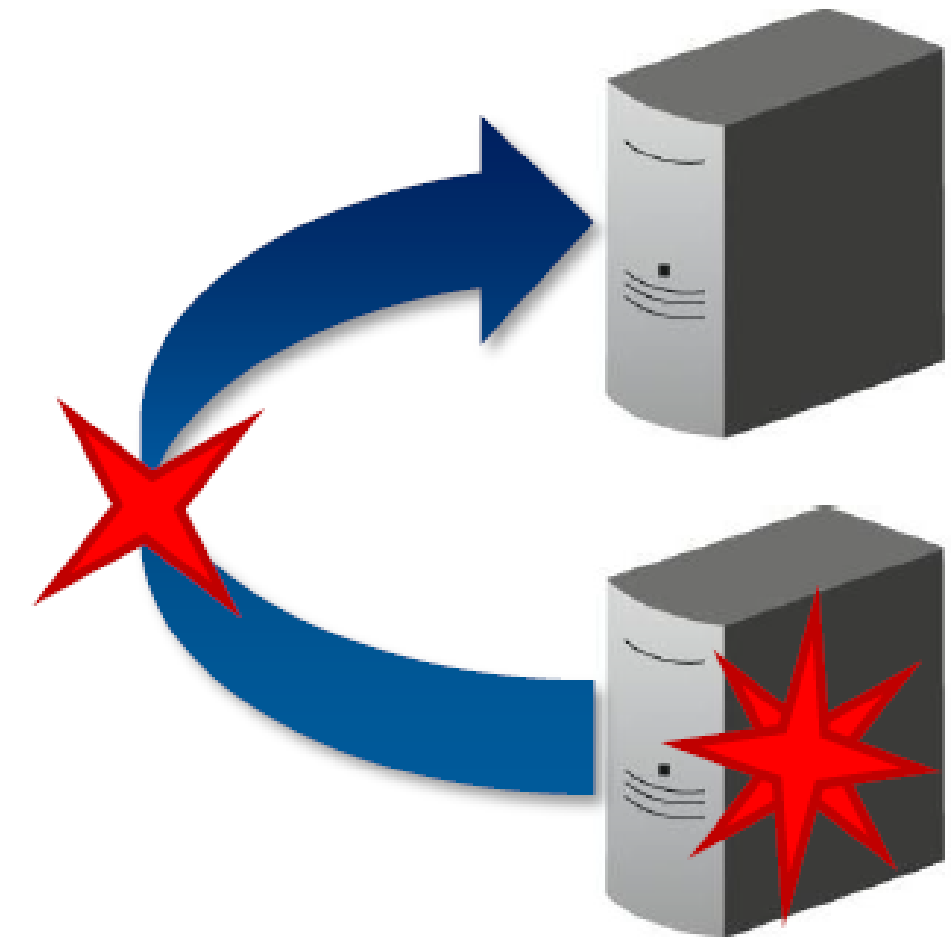
## HADOOP 1.0 challenges

Problem	Description
NameNode – No Horizontal Scalability	Single NameNode and Single Namespaces, limited by NameNode RAM
NameNode – No High Availability (HA)	NameNode is Single Point of Failure, Need manual recovery using Secondary NameNode in case of failure
Job Tracker – Overburdened	Spends significant portion of time and effort managing the life cycle of applications
MRv1 – Only Map and Reduce tasks	Humongous Data stored in HDFS remains unutilized and cannot be used for other workloads such as Graph processing etc.

# NAME NODE DOES NOT SCALE



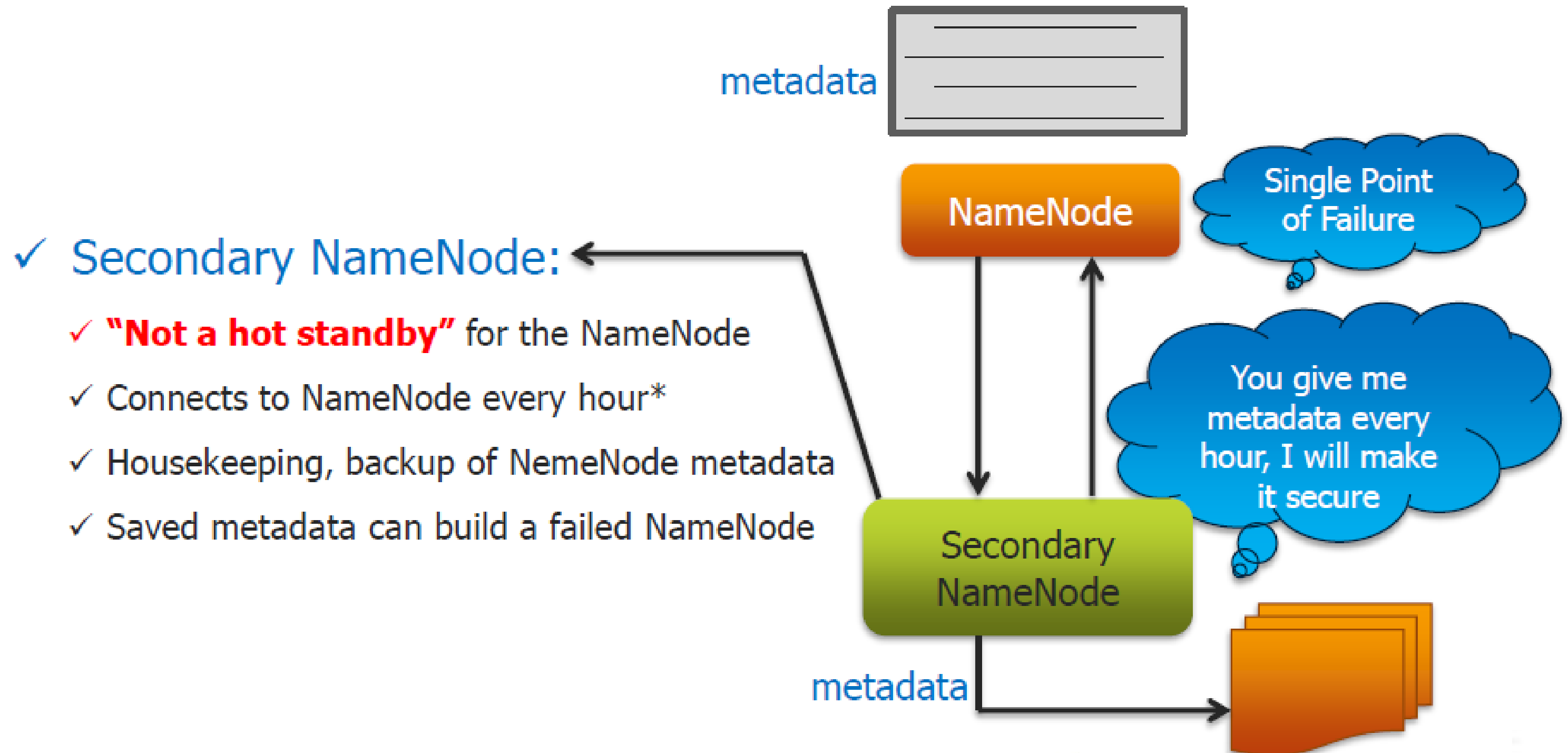
**NameNode - No Horizontal Scale**



**NameNode - No High Availability**



# NAME NODE – SINGLE PPOINT OF FAILURE



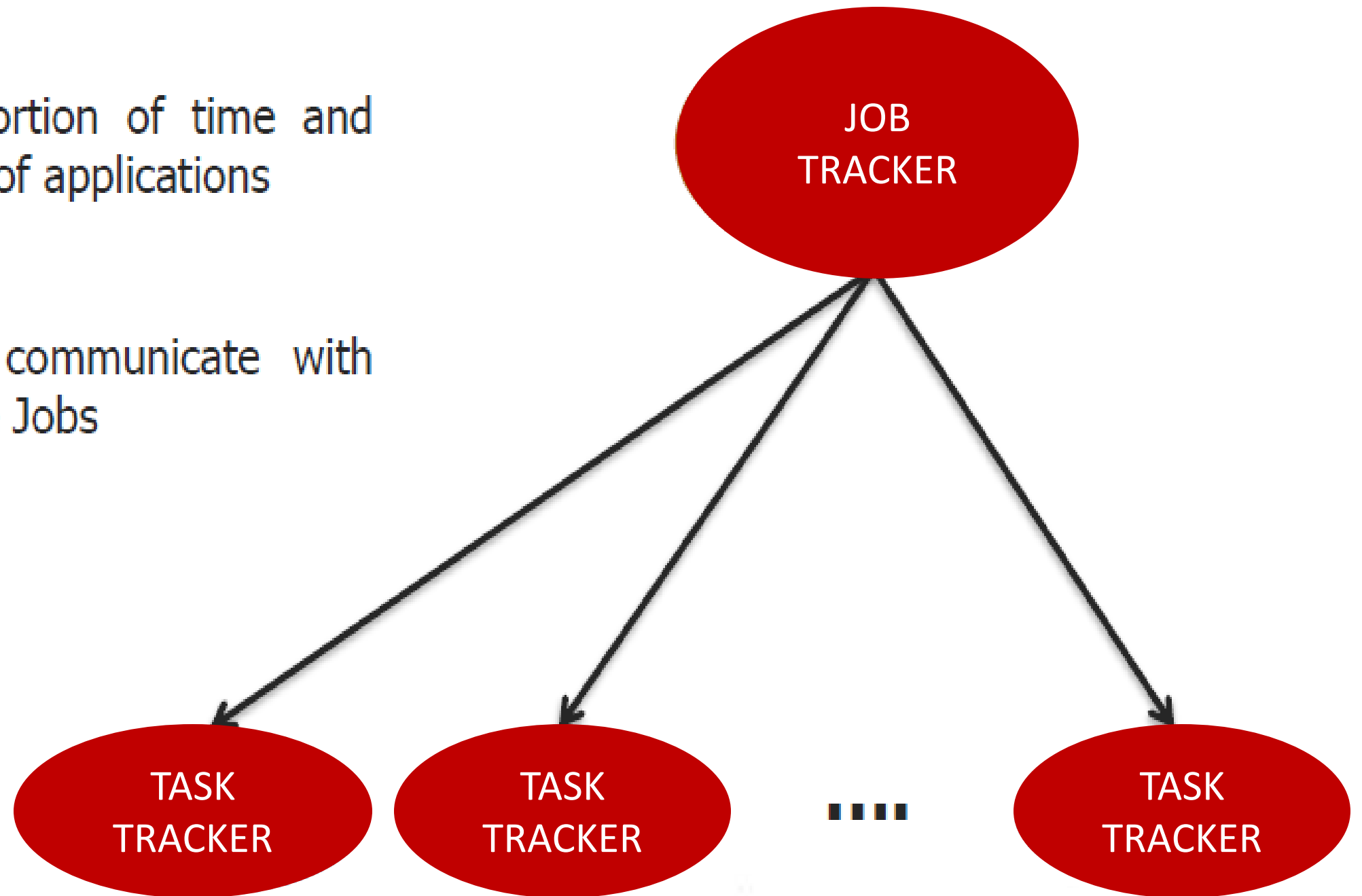
# JOB TRACKER - LOAD

## CPU

- ✓ Spends a very significant portion of time and effort managing the life cycle of applications

## Network

- ✓ **Single** Listener Thread to communicate with thousands of Map and Reduce Jobs



# HADOOP 2.0 – NEW FEATURES

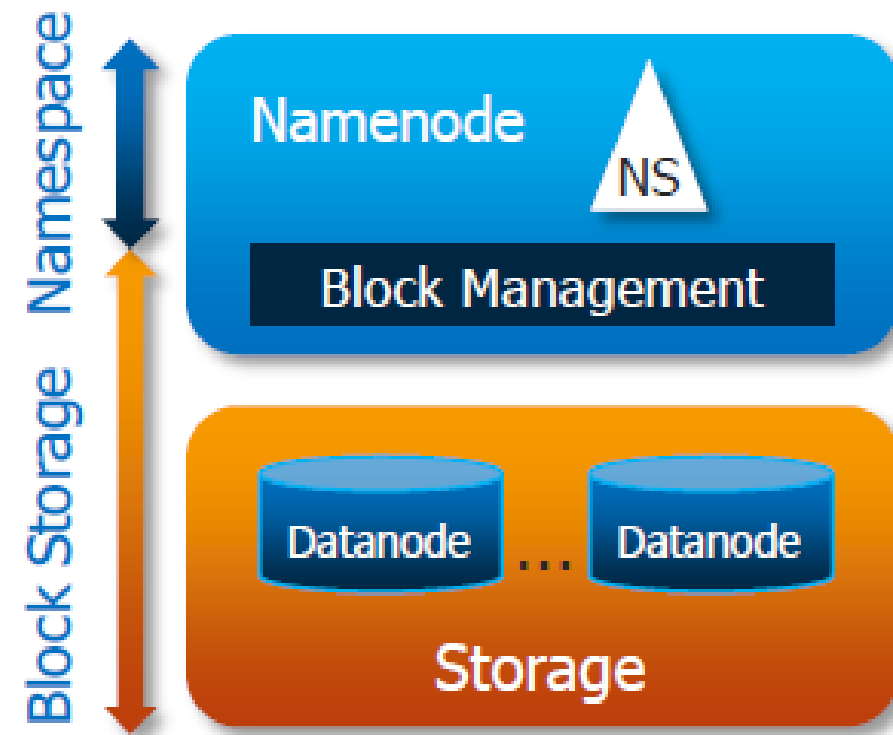
PROPERTY	HADOOP 1.0	HADOOP 2.0
Federation	One Namenode and Namespaces	Multiple Namenode and Namespaces
High Availability	Not present	Highly Available
YARN - Processing Control and Multi-tenancy	JobTracker, Task Tracker	Resource Manager, Node Manager, App Master, Capacity Scheduler

## Other important Hadoop 2.0 features

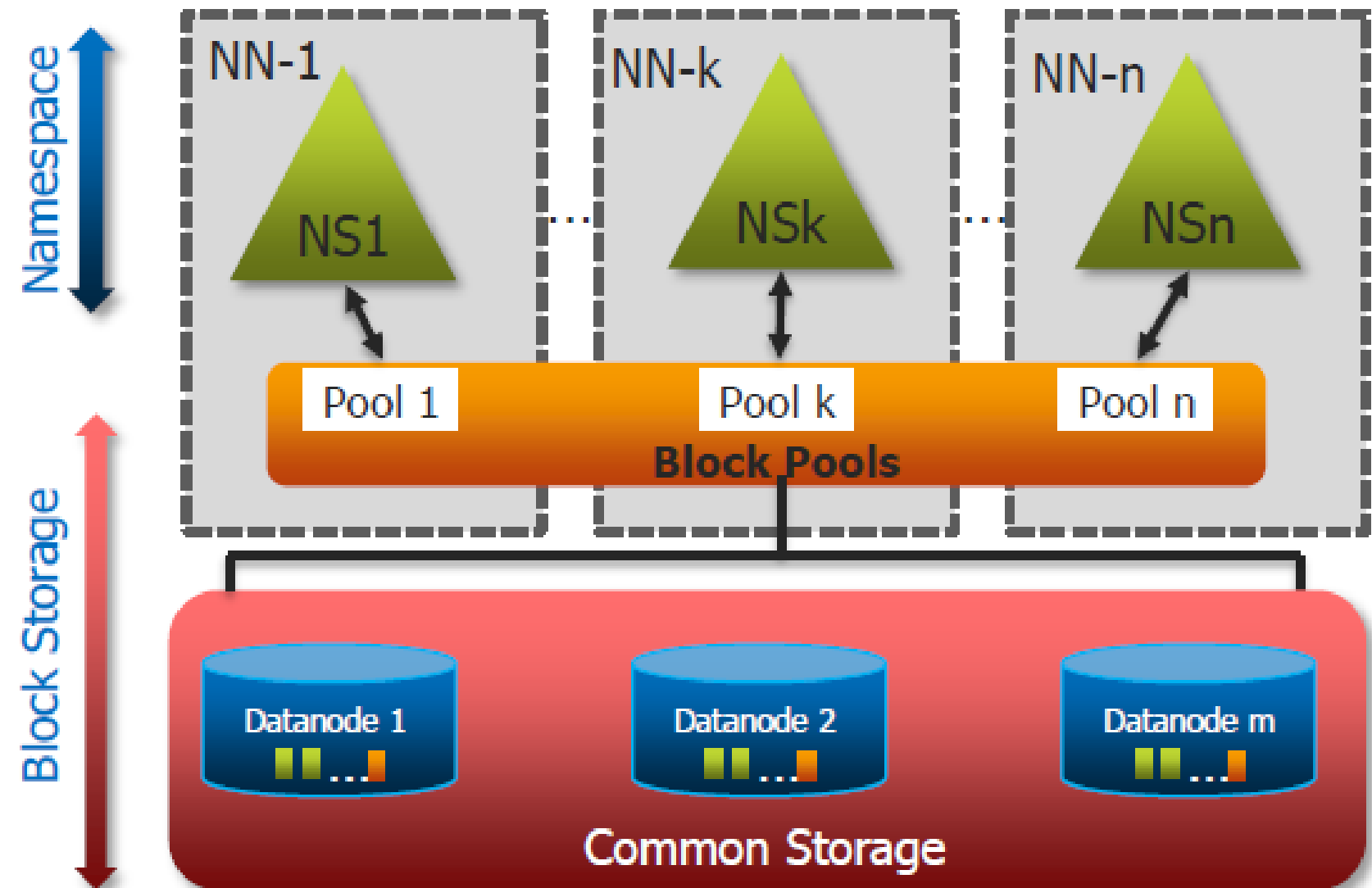
- ✓ HDFS Snapshots
- ✓ NFSv3 access to data in HDFS
- ✓ Support for running Hadoop on MS Windows
- ✓ Binary Compatibility for MapReduce applications built on Hadoop 1.0
- ✓ Substantial amount of Integration testing with rest of the projects (such as PIG, HIVE) in Hadoop ecosystem

# HADOOP 2.0

## HADOOP 1.0



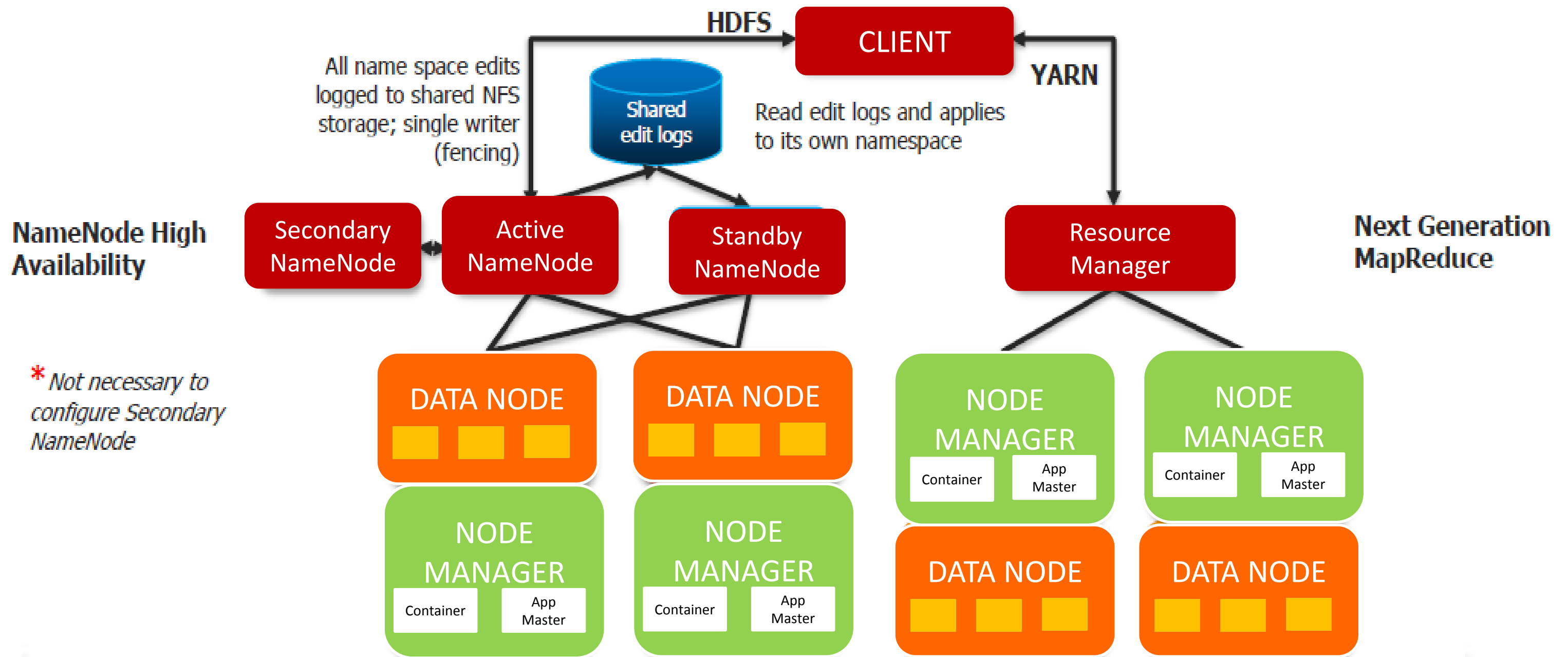
## HADOOP 2.0



<http://hadoop.apache.org/docs/stable2/hadoop-project-dist/hadoop-hdfs/Federation.html>

# HADOOP 2.0

## HDFS HIGH AVAILABILITY



# HADOOP – 2.0

High Availability in  
Hadoop 2.0

Automatic Fail Over  
To Standby Name  
Node

Active  
NameNode

Standby  
NameNode

Secondary  
Name Node

Automatic Fail Over  
To Standby Name  
Node

Secondary  
Name Node

NameNode

Edit logs

FSImage

Meta-Data

NameNode recovery in  
Hadoop 1.0

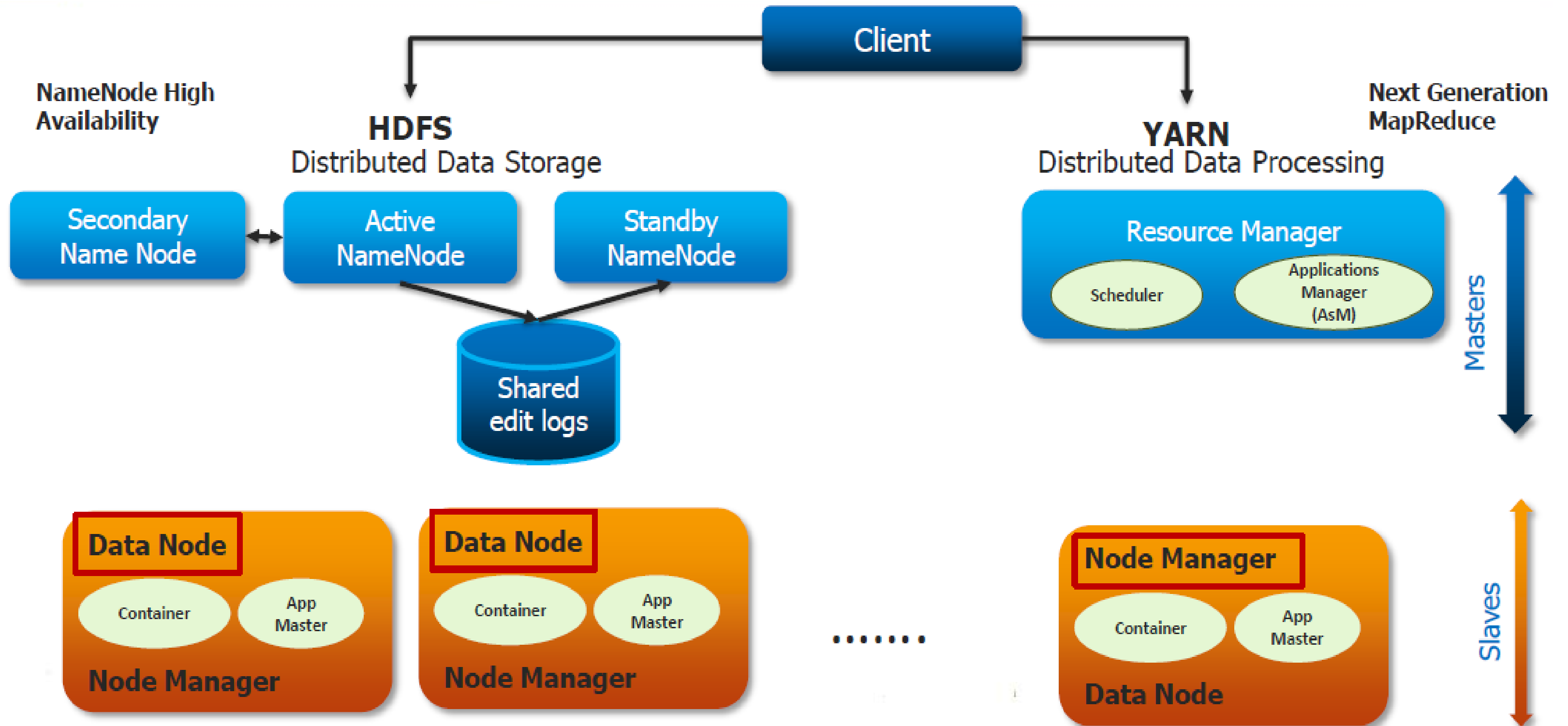
# HADOOP 2.0

## **How does HDFS Federation help HDFS Scale horizontally?**

Reduces the load on any single NameNode by using the multiple, independent NameNode to manage individual parts of the file system namespace

In order to scale the name service horizontally, HDFS federation uses multiple independent Namenodes. The Namenodes are federated, that is, the Namenodes are independent and don't require coordination with each other.

# HADOOP 2.0





# AGENDA

## HBASE

# WHAT PIG AND HIVE IS NOT

- We can analyse data / not good for real time and not good for row level update.
- In Hive, we do only processing, Analyse message in real time, Real time data input
- HDFS does not provide individual record look up
- Data model is different

# NO SQL DATABASES AND HBASE

=====

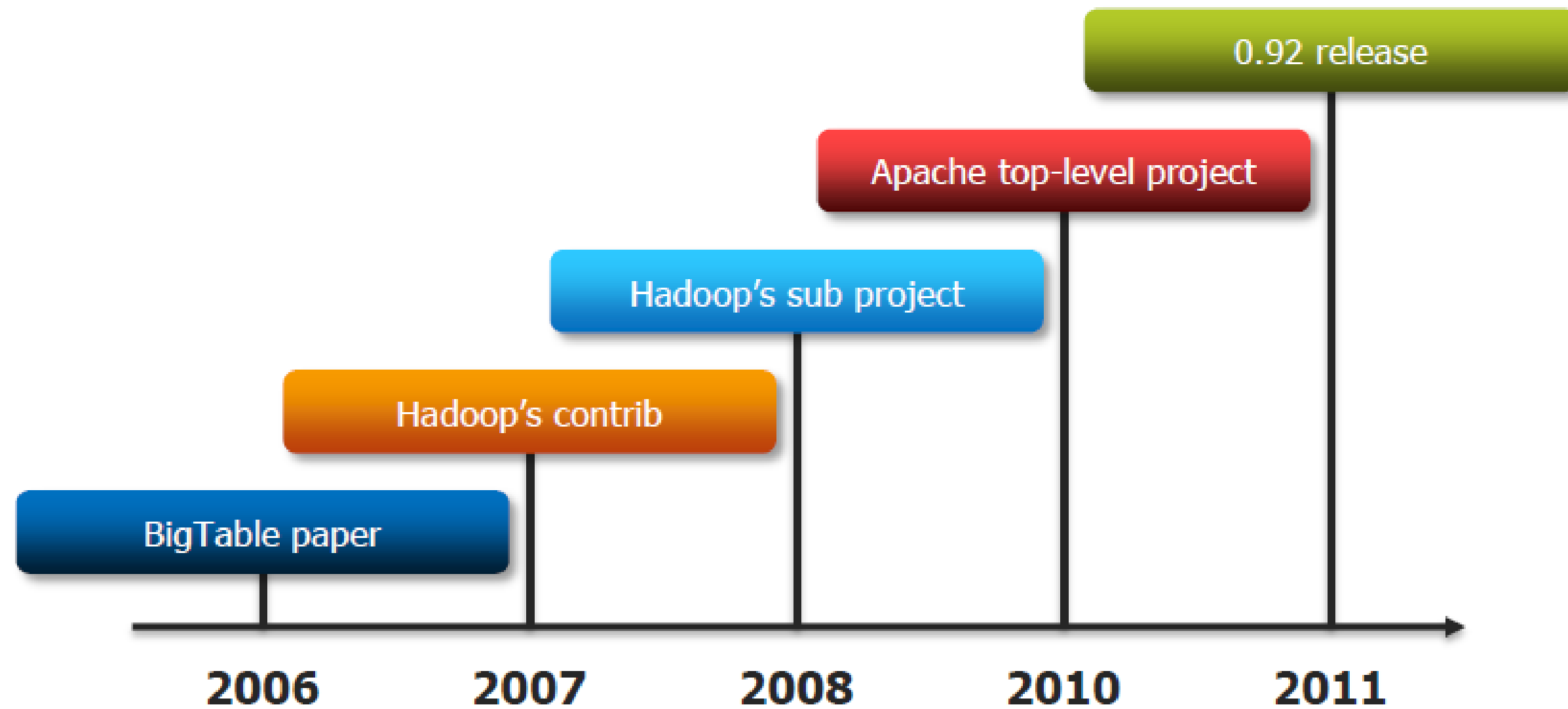
## WHAT WE NEED



# WHAT COULD BE A GOOD SOLUTION

- ☐ Distributed
- ☐ Sorted
- ☐ Sparse Data store – thinly distributed – with lot of gaps
- ☐ Automatic Sharing
- ☐ 150 kinds of No SQL databases.

# HBASE



# HBASE CHARACTERISTICS

✓ HBase is a key/value store. Specifically it is:

SPARSE

DISTRIBUTED

MULTI  
DIMENSIONAL

SORTED  
MAP

CONSISTENT

# HBASE & RDBMS

HBASE	HBASE
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema.
Good with sparse tables,	Not optimized for sparse tables.
Joins using MR –not optimized	Optimized for joins.
Tight integration with MR	Not really...
Horizontal scalability –just add hardware	Hard to shard and scale
Good for semi-structured data as well as structured data	Good for structured data

# HBASE

HBASE has Column Family – A column family should have one column at least

Each row will be identified by a unique row

Data is stored with time stamp

To retrieve the data we need to know :

Table Name => Row Id => Column Family => Column => Time Stamp

HBASE versions the data

Hbase rowkey uniquely identifies a row and we should add unique id here.



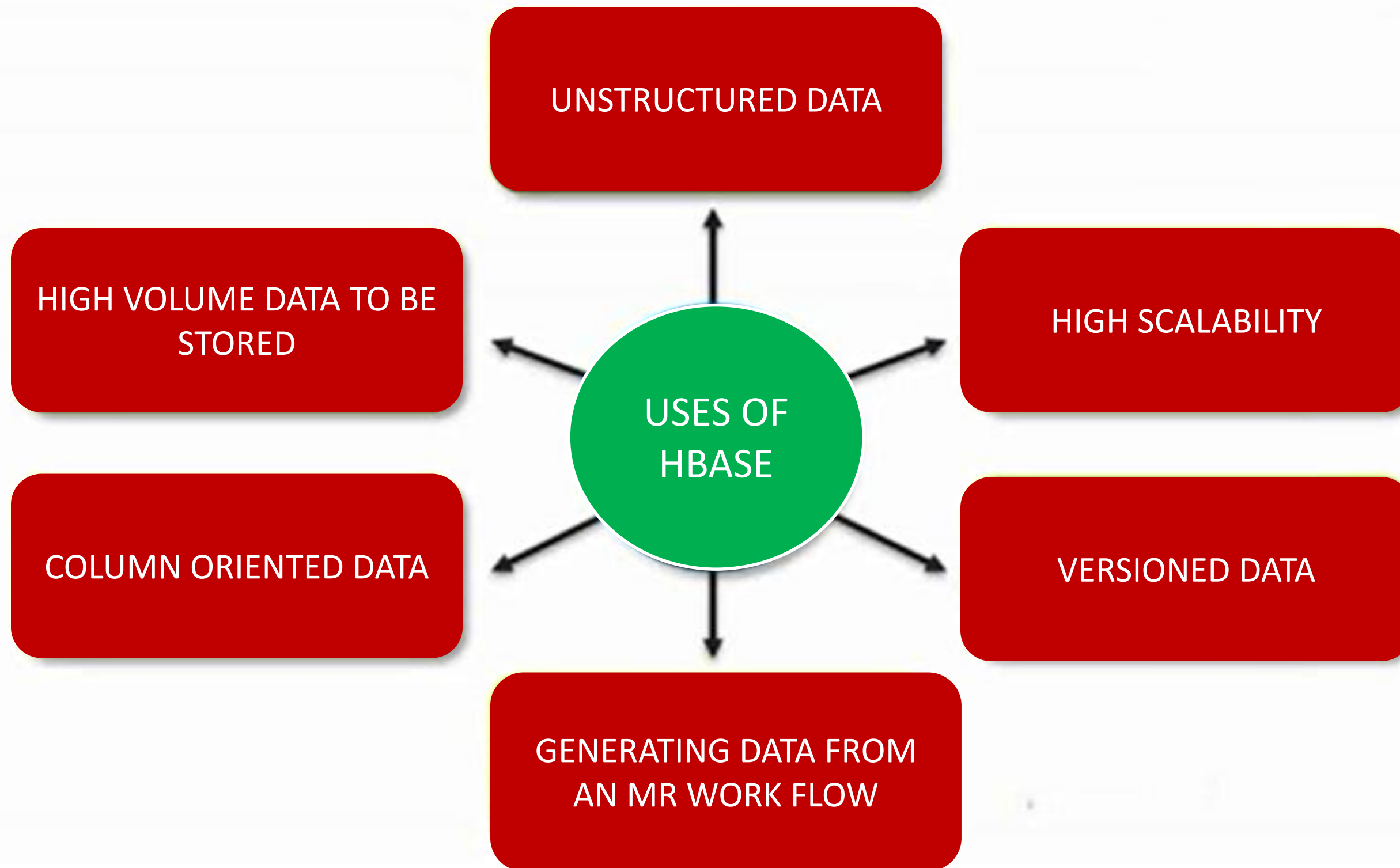
# HBASE

Table row keys are also byte arrays so almost anything can serve as a row key.

Strings to binary representations of longs or even serialized data structures.

Varied schema – column family can have empty values for one row.

## HBASE – WHEN TO USE



# WHEN NOT TO USE

When you have few millions

When strict typing

# HBASE



- ✓ Facebook **monitored their usage and figured out what they really needed.**
- ✓ What they needed was a system that could handle two types of data patterns:
  - ✓ A short set of temporal data that tends to be volatile
  - ✓ An ever-growing set of data that rarely gets accessed

# Hbase implemenations



A number of applications including people search rely on HBase internally for data generation.



Uses HBase to power their Messages infrastructure  
<http://sites.computer.org/debull/A12june/facebook.pdf>



We use HBase as a real time data storage and analytics platform.



Uses HBase to store document fingerprint for detecting near-duplications. We have a cluster of few nodes that runs HDFS, mapreduce, and HBase.



Uses an HBase cluster containing over a billion anonymized clinical records.



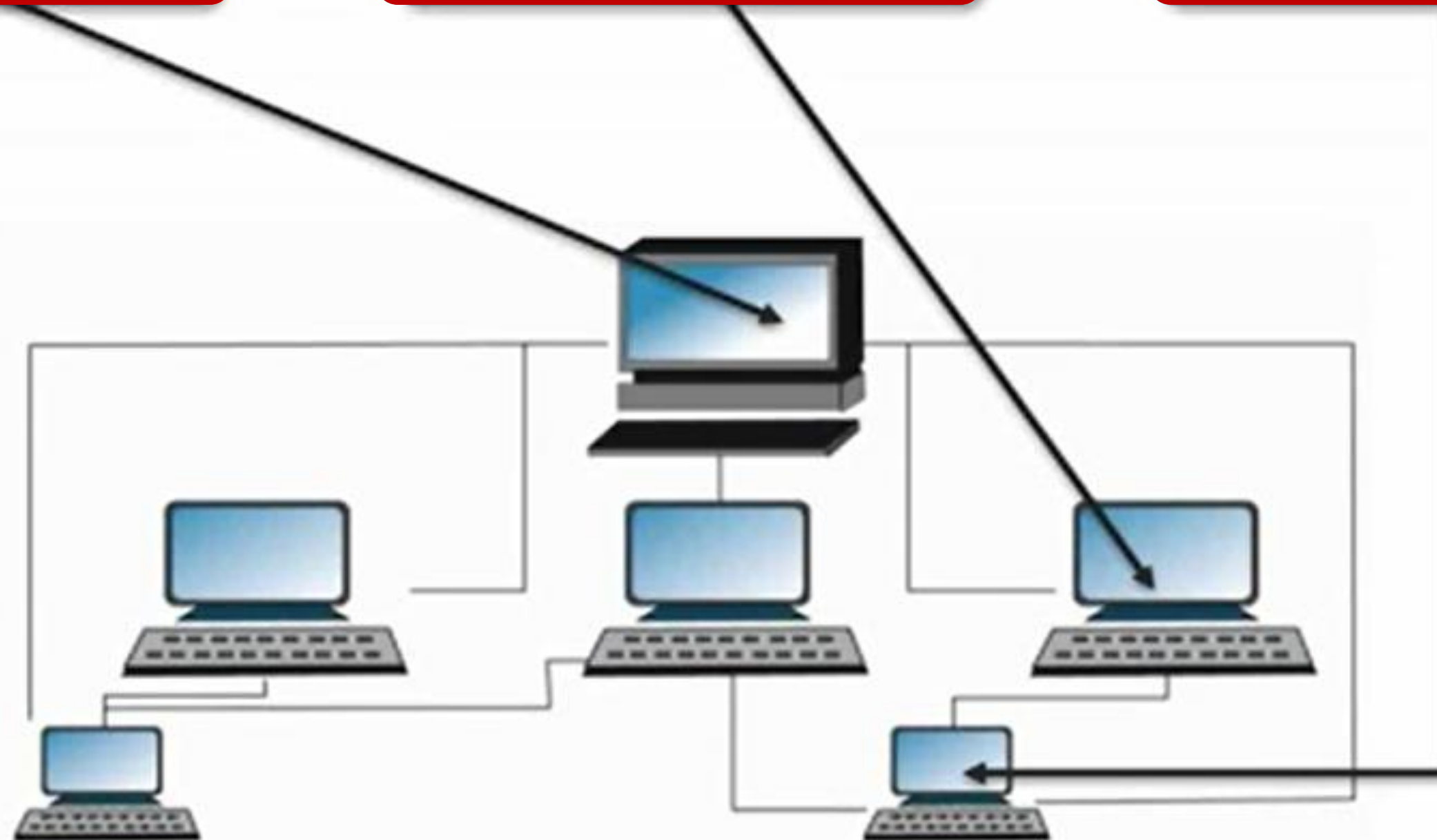
Uses HBase as a foundation for cloud scale storage for a variety of applications.

The diagram illustrates the HBase architecture components and their interactions:

- THE HBASEMASTER**: The central master node, represented by a desktop computer icon.
- THE HREGIONSERVER**: The server nodes, represented by desktop computer icons.
- THE HBASECLIENT**: The client nodes, represented by laptop icons.

Connections shown in the diagram:

- The **HBASEMASTER** is connected to all **HREGIONSERVER** nodes.
- The **HBASECLIENT** is connected to all **HREGIONSERVER** nodes.
- The **HREGIONSERVER** nodes are connected to each other, forming a network.





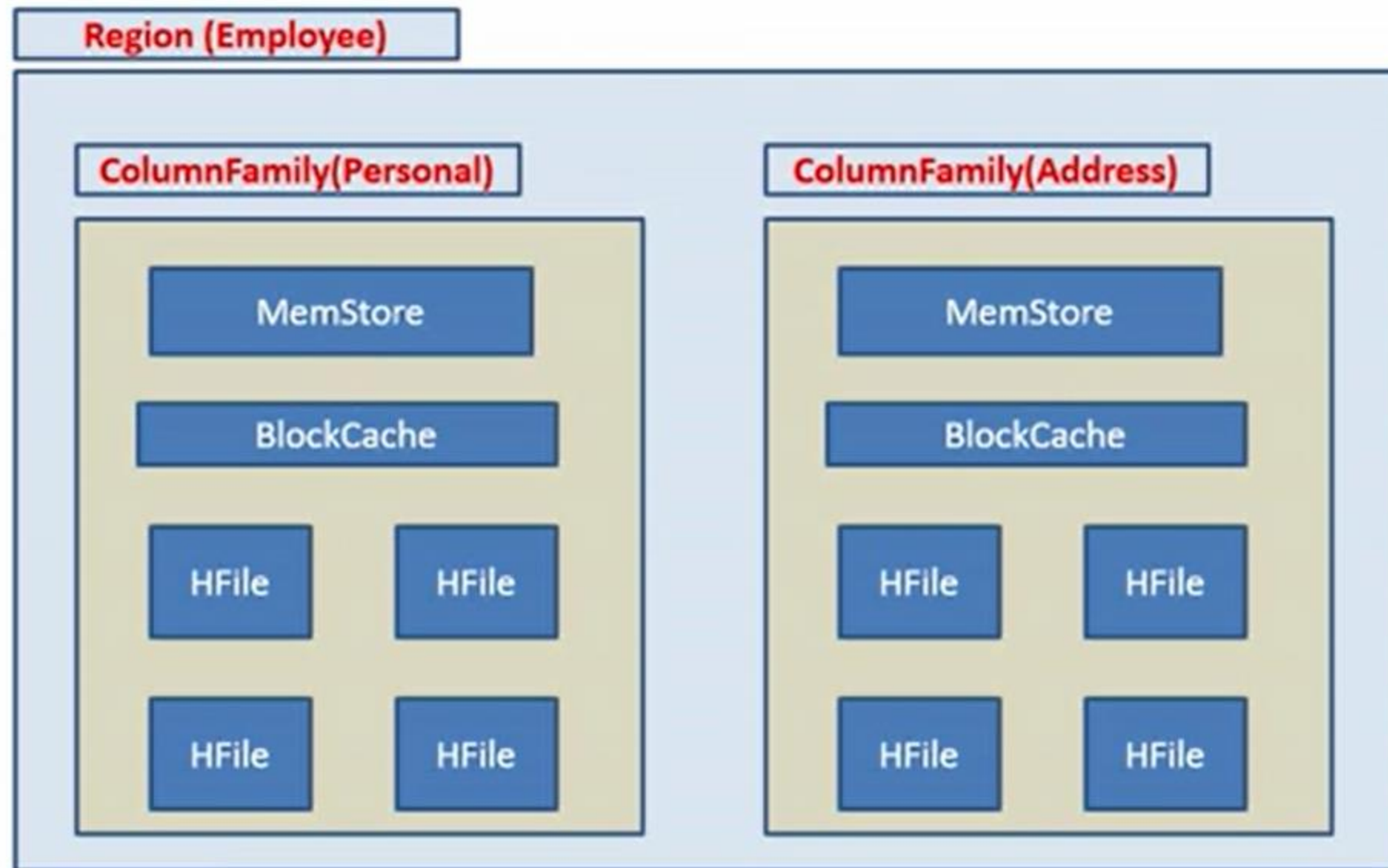
# HBASE Storage

## HBASE - Storage

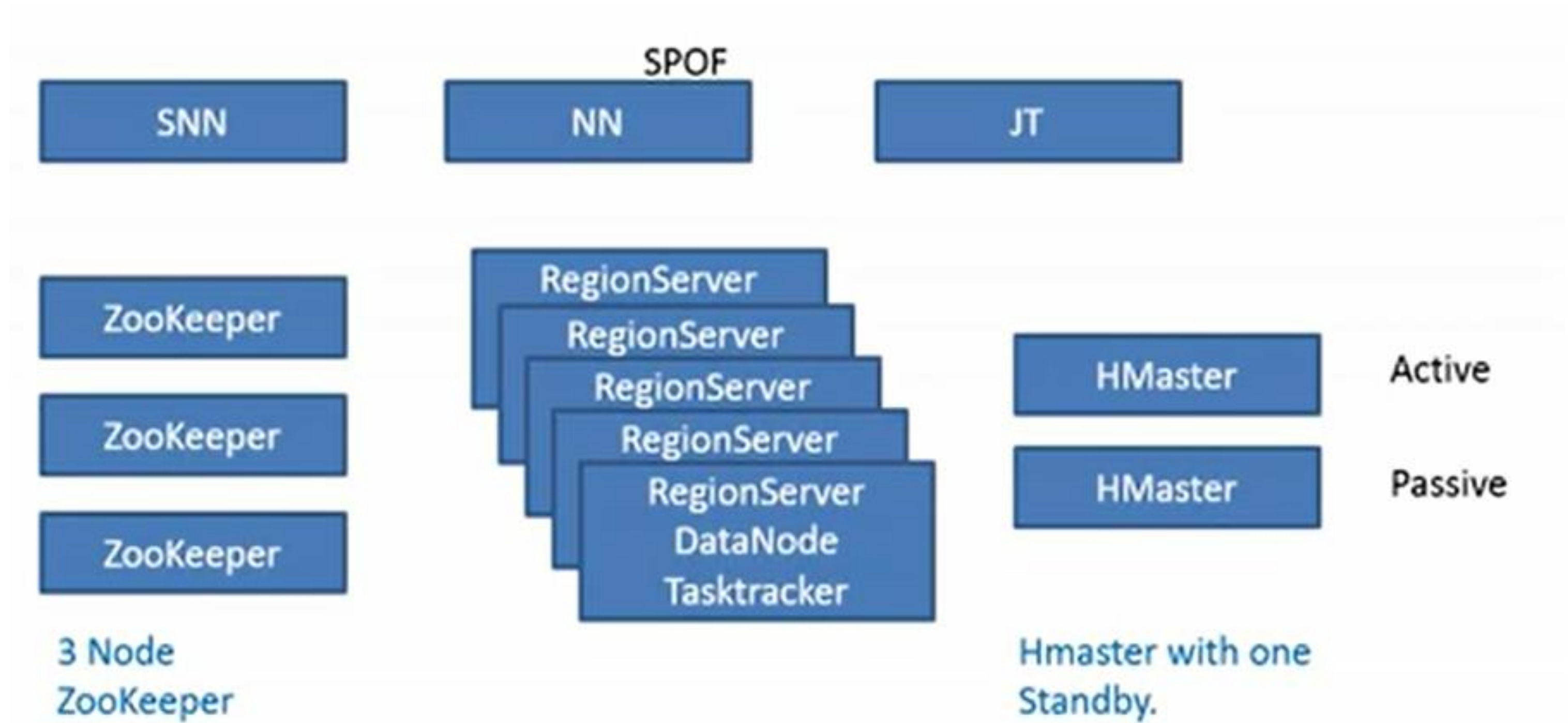
Region Size by Default is 256 mb

Hbase-site.xml hbase.hregion.max.filesize (Upperbound of Region Size is 4 gb)

A region in the **Employee** table



# HBASE Architecture





## REGION SERVERS

- ❑ HBase stores rows of data in tables.
- ❑ Tables are split into chunks of rows called “regions”.
- ❑ Those regions are distributed across the cluster, hosted and made available to client processes by the RegionServer process.
- ❑ A region is a continuous range within the key space, meaning all rows in the table that sort between the region’s start key and end key are stored in the same region.
- ❑ Regions are non-overlapping,
- ❑ A region is only served by a single region server at any point in time, which is how HBase guarantees strong consistency within a single row#.

# Apache Flume

- ❑ A Data collection service for Hadoop
- ❑ For distributed systems
- ❑ Open source
- ❑ Scalable
- ❑ Reliable
- ❑ Manageable
- ❑ Fault tolerant

# Apache Flume

**Flume uses Agents which have**

## **A Source**

- Listen for events
- Write events to channel

## **A channel**

- Queue event data as transactions

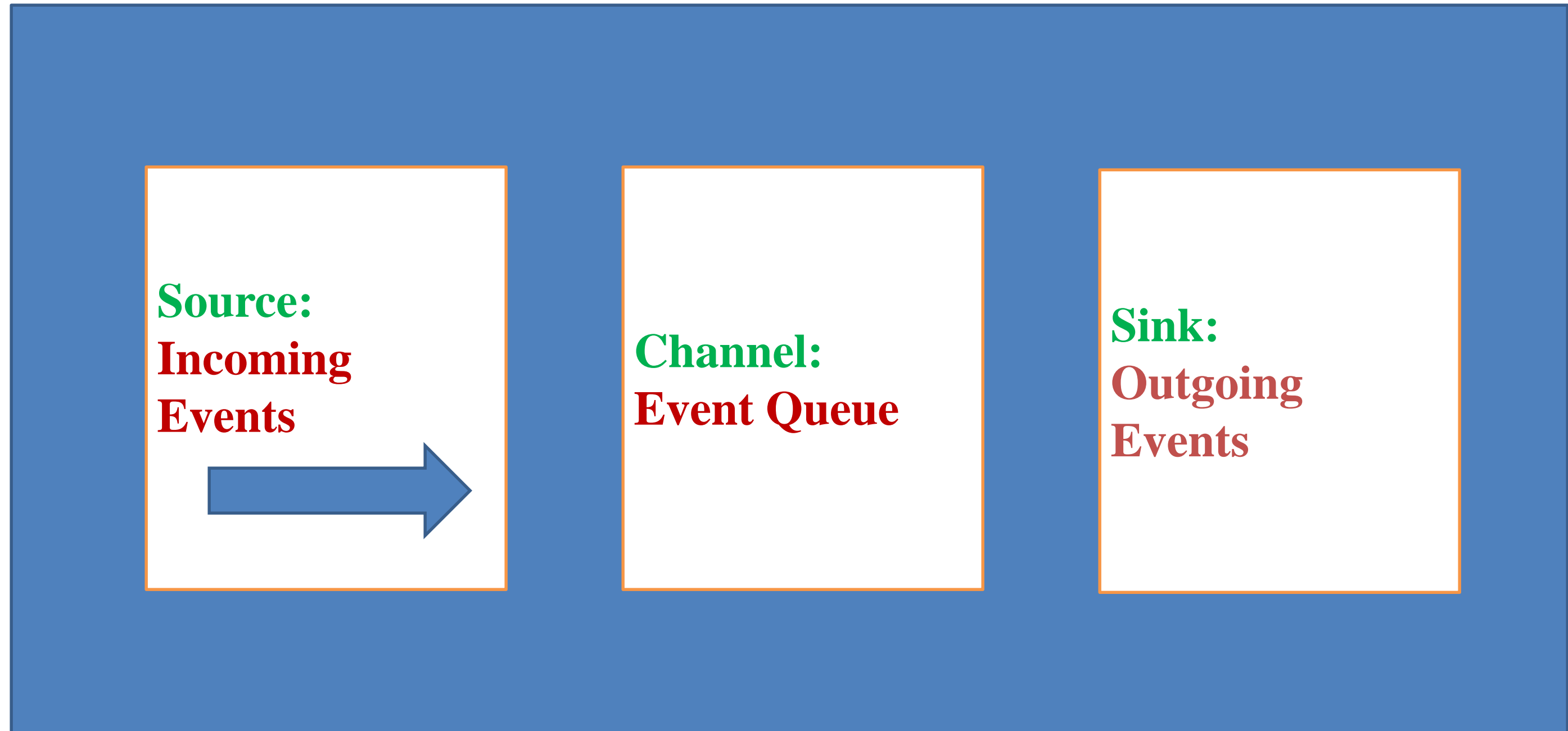
## **A sink**

- Write event data to target ie HDFS
- Remove event from queue

# Apache Flume

A single agent showing its parts

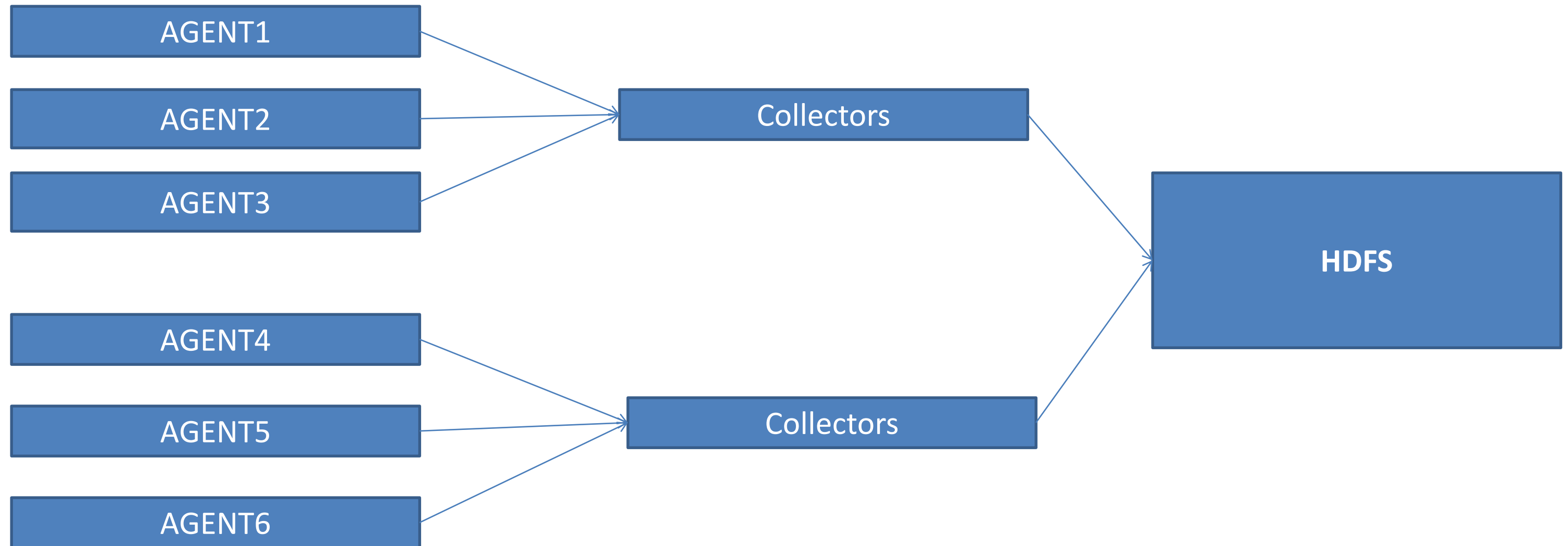
Generally one Agent for a given data type



# Apache Flume

AGENTS CAN BE CHAINED INTO FLOWS

For data serialization we use Avro



# AGENDA

## **HADOOP DEVELOPMENT CONSIDERATIONS**

# What to use when?

## ☐ Map Reduce

- ☐ For large and recurring Jobs
- ☐ For Speed and control
- ☐ Processing binary data
- ☐ Reuse of existing data libraries

## ☐ Pig

- ☐ For ad-hoc analysis

## ☐ Hive

- ☐ SQL style analysis and data warehousing

## Use Combiner

- ☐ Combiners are like mini-reducers
- ☐ Reduces network time
- ☐ The reducer class can itself be used as combiner
- ☐ Use it if network time is significantly higher after mapper is complete - can monitor using network monitoring tools



## USE Compression

- ❑ **Compression provides two benefits**
- ❑ **Reduce space needed to store files**
- ❑ **Speeds up data transfer across network and to or from disk**
- ❑ **Use compression type that supported splitting like bzip2 or LZO**
- ❑ **Use Sequence file format which supports compression and splitting**
- ❑ **AVRO files also support compression and splitting**

# CHAIN YOUR JOBS

- ☐ The processing of data to arrive at final output may need multiple steps
- ☐ Design multiple jobs to process every stage with intermediate outputs
- ☐ In case of job fails, the process needs to be restarted from the last successful job run
- ☐ It alleviates the risk. of running the complete process once again
- ☐ Store and take backup of intermediate outputs
- ☐ These output can be reused even if some jobs are re-designed

## **Deploy on Cloud or leverage cloud services**

- ☐ **Hadoop jobs may require hundreds of nodes**
- ☐ **Provisioning the nodes in house may be time consuming affair**
- ☐ **Cloud provides resources on demand**
- ☐ **Build a cluster on cloud and then use it for processing several jobs**
- ☐ **Amazon AWS provides Elastic Map Reduce (EMR) platform for running Hadoop jobs**

# CHOOSE RIGHT HARDWARE AND STORAGE

## ☐ Profile your workload type

- ☐ Memory intensive

- ☐ time Intensive

- ☐ CPU Intensive

## ☐ High end hardware for namenode or job tracker node

- ☐ To increase availability

## ☐ Shared Storage for Name Node

- ☐ To withstand server failure

# NODE CONFIGURATIONS

**Hadoop Heapsize determines the memory of deamons**

**Mapd.child.java.opts decidse the memory for each map or reduce tasks**

**If namenode, secondary namenode and job tracker are run on same machines, it will need more than 3 GB RAM to run these daemons**

**Namenode should have large memory available depending on number of files on the HDFS**

# MONITOR INFRASTRUCTURE

❑ Find out bottlenecks

❑ Monitor the following parameters

- ● Disk Performance - iostat
- ● Network Performance - ethtool, ping

❑ Use Monitoring tools

Ganglia

Nagios

## IMPORT DATA TO HDFS

- ☐ Do not use data directly from Database using DBInputFormat
- ☐ Reading from database or from other file systems can be time consuming and can effect the processing time significantly
- ☐ Import data to HDFS using Sqoop
- ☐ Export results back to Database

## DO PERIODIC MAINTENANCE

- ☐ Backup Name Node periodically
- ☐ Keep checking HDFS Health System
- ☐ Add nodes or remove nodes appropriately
- ☐ Remove temporary files
- ☐ Take back up of output files or intermediate files
- ☐ Reconfigure the system based on number of nodes added or removed



## FIRST TEST YOUR JOB ON SMALL SET OF DATASET

- ☐ Check if the map reduce jobs are working properly
- ☐ Do sampling and analyze the output
- ☐ Optimize the code
- ☐ Develop combiner and custom partitioner to optimize the performance
- ☐ Always write unit test cases to test map and reduce programs
- ☐ This would save significant amount of time if there are bugs in the code or the code does not provide the results intended