getindata

Introduction to
Hadoop Ecosystem

Marek Wiewiórka - GetInData

What Is Big Data?

1. **Making data-driven decisions based on complete data**
    - Fast, automated, accurate
2. **Using technologies for collecting, storing and analyzing data**
    - Cost-efficient, scalable, extensible, reliable

Why Infrastructure For Big Data?

- **4.4 zettabytes in 2013 and is forecasting a tenfold growth by 2020 to 44 zettabytes**
- **A full 90 percent of all the data in the world has been generated over the last two years**
  - Stated in May 2013
- **Only 0.5% of all data is currently analyzed**
  - Around 23% of data useful if tagged and analyzed

How could
a good distributed system
look like ?

Linear Scalability

- **A big system of small machines - not a big machine**
  - Scale easier!

A big system of small animals!

Fault-tolerance

- **Partial failure shouldn’t break the system**
  - It shouldn’t break the system, but it will only degrades system proportionally
  - It shouldn't cause data-loss or computation-failure
  - A broken component can be easily replaced
- **Failures will happen, so expect them**
  - Write code that handles software and hardware failures
  - Use cheaper hardware!

Abstraction

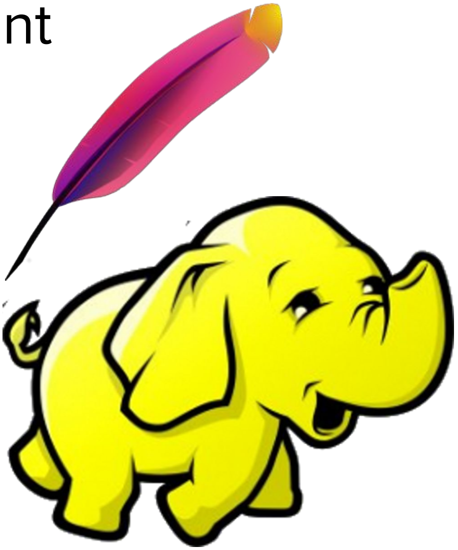
- **Hide all messy details related to distributed computing into a easy-to-use library**
- **Provide clean abstraction for users**
  - Use a high-level API to write your business code!

Performance

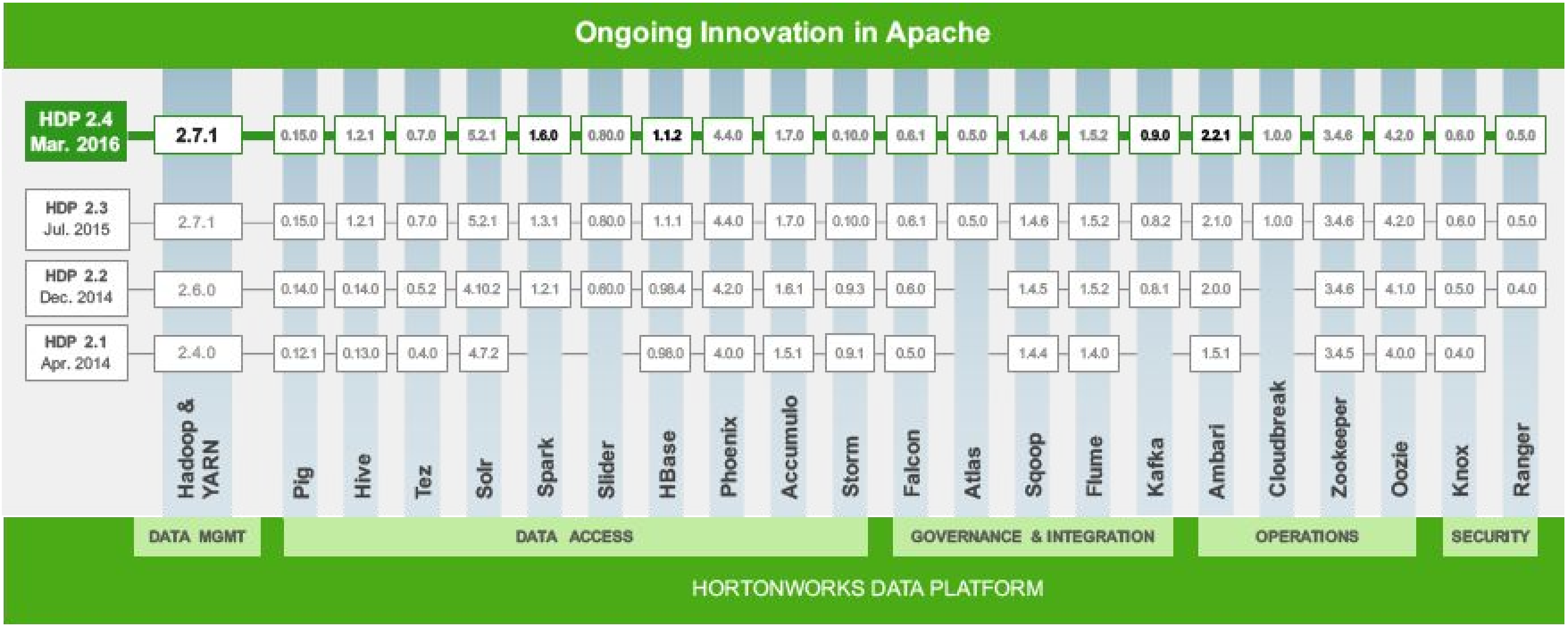
- **Move computation to data - not data to computation**
  - Save the network bandwidth!
- **Take advantage of decreasing costs of hardware**

Apache Hadoop

- **Follows the mentioned ideas**
- **Provides a battle-tested solution for large-scale computation**
- **Contains two core components**
  - HDFS - a distributed storage
  - YARN - a distributed resource management
- **Integrates with many other useful tools**
  - High-level libraries to process data



Hortonworks Data Platform



Ongoing Innovation in Apache

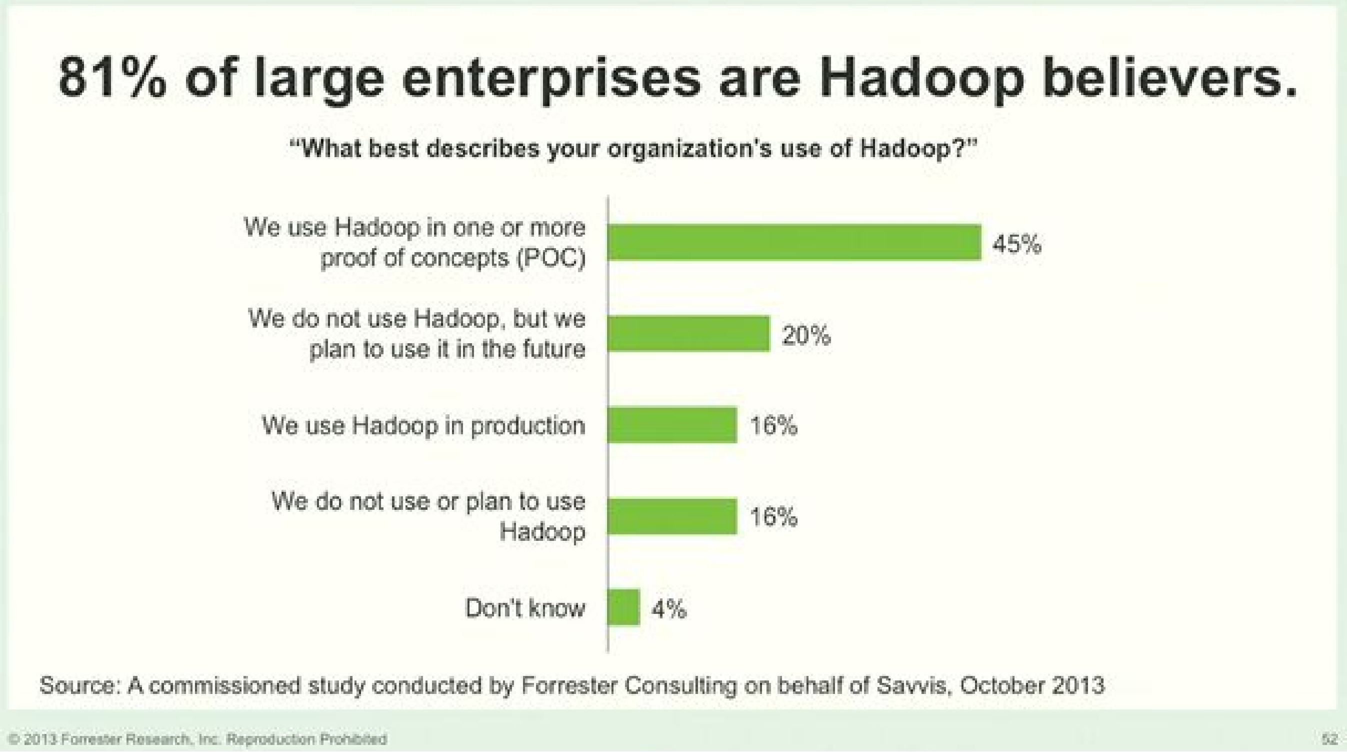
	DATA MGMT	DATA ACCESS									
	Hadoop & YARN	Pig	Hive	Tez	Solr	Spark	Slider	HBase	Phoenix	Accumulo	Storm
HDP 2.4 Mar. 2016	2.7.1	0.15.0	1.2.1	0.7.0	5.2.1	1.6.0	0.80.0	1.1.2	4.4.0	1.7.0	0.10.0
HDP 2.3 Jul. 2015	2.7.1	0.15.0	1.2.1	0.7.0	5.2.1	1.3.1	0.80.0	1.1.1	4.4.0	1.7.0	0.10.0
HDP 2.2 Dec. 2014	2.6.0	0.14.0	0.14.0	0.5.2	4.10.2	1.2.1	0.60.0	0.98.4	4.2.0	1.6.1	0.9.3
HDP 2.1 Apr. 2014	2.4.0	0.12.1	0.13.0	0.4.0	4.7.2			0.98.0	4.0.0	1.5.1	0.9.1

	GOVERNANCE & INTEGRATION					OPERATIONS				SECURITY	
	Falcon	Atlas	Sqoop	Flume	Kafka	Ambari	Cloudbreak	Zookeeper	Oozie	Knox	Ranger
HDP 2.4 Mar. 2016	0.6.1	0.5.0	1.4.6	1.5.2	0.9.0	2.2.1	1.0.0	3.4.6	4.2.0	0.6.0	0.5.0
HDP 2.3 Jul. 2015	0.6.1	0.5.0	1.4.6	1.5.2	0.8.2	2.1.0	1.0.0	3.4.6	4.2.0	0.6.0	0.5.0
HDP 2.2 Dec. 2014	0.6.0		1.4.5	1.5.2	0.8.1	2.0.0		3.4.6	4.1.0	0.5.0	0.4.0
HDP 2.1 Apr. 2014	0.5.0		1.4.4	1.4.0		1.5.1		3.4.5	4.0.0	0.4.0	

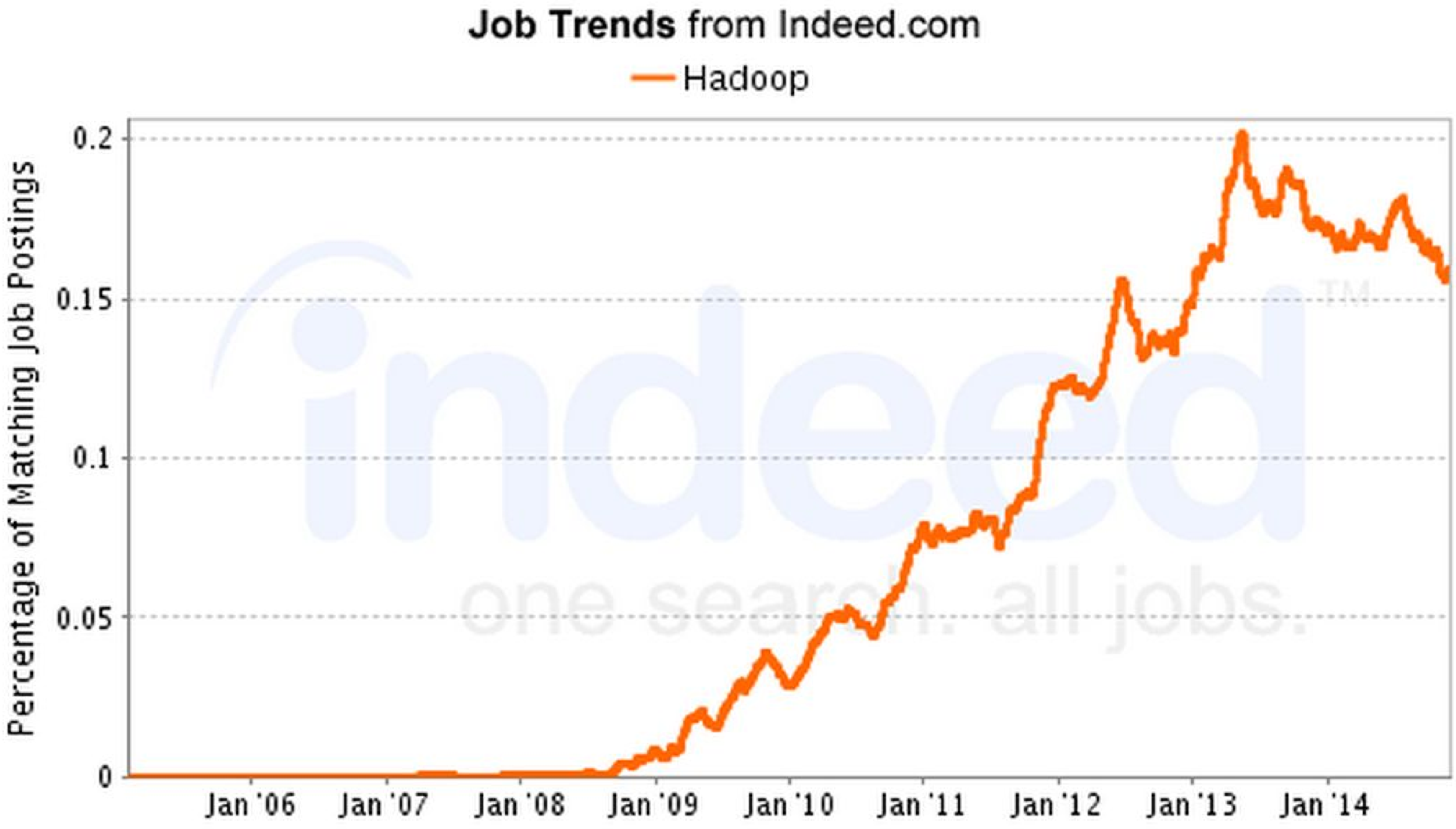
HORTONWORKS DATA PLATFORM

Hadoop Adoption

- **Market for Hadoop is expected to grow 25x times until 2020**



Hadoop Job Trends



Let's see how Hadoop faces Big Data challenges!

Chapter

HDFS

Hadoop Distributed File System

A Definition For Your Uncle

An easy to use software program

that runs on many inexpensive computers

and stores many files redundantly

and still works when some of the computers crash!

Dry Demo

Interacting with HDFS

Dry Demo

1. **List the content of home directory**

```
$ hdfs dfs -ls /user/jeff
```

2. **Upload a file from a local filesystem to HDFS**

```
$ hdfs dfs -put songs.txt /user/jeff
```

3. **Read the content of the file from HDFS**

```
$ hdfs dfs -cat /user/jeff/songs.txt
```

Dry Demo

4. **Create a subdirectory in your home directory**

   ```
   $ hdfs dfs -mkdir songs
   ```

5. **Move the file to the newly-created subdirectory**

   ```
   $ hdfs dfs -mv songs.txt songs/
   ```

6. **Remove the directory from HDFS**

   ```
   $ hdfs dfs -rmr songs
   ```

Uploading A File To HDFS

Question?

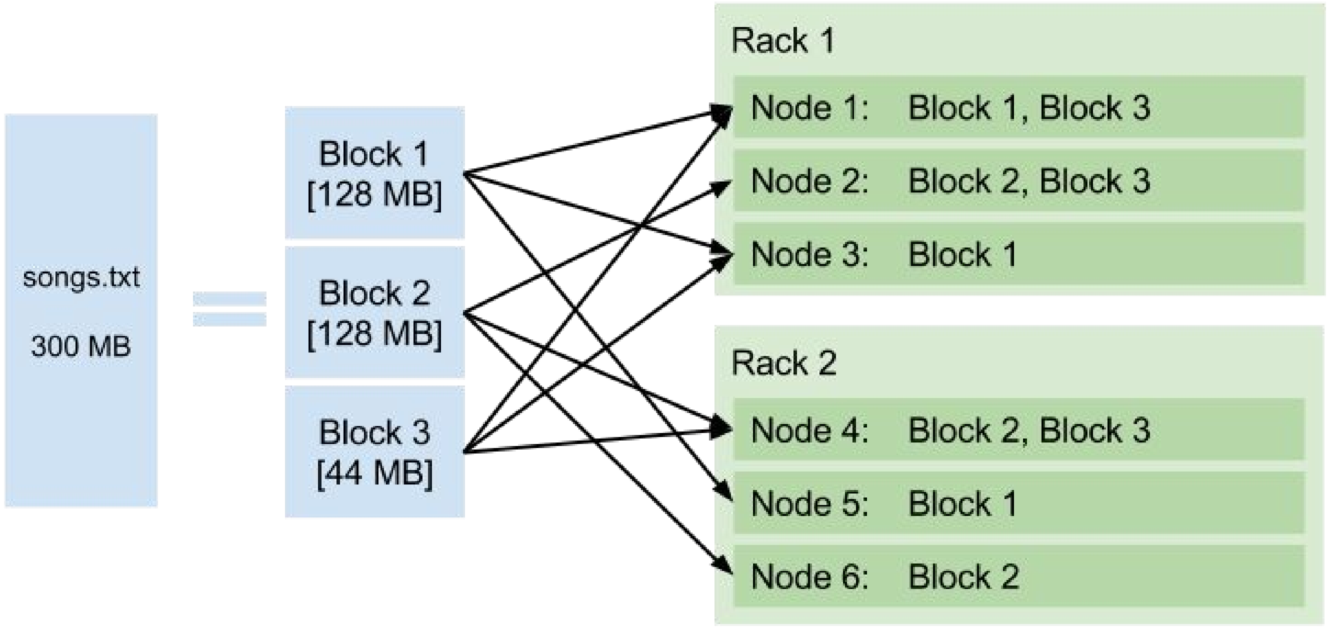
- **What happens when a file is uploaded to HDFS?**

```
$ hdfs dfs -put songs.txt /user/tiger
```

Uploading A File To HDFS

Answer!

- A file is split into smaller, but still large, blocks
- Each block is stored redundantly on multiple machines



Splitting A File Into Blocks

- **The default block size is 128MB**
  - Sometimes even 256MB is recommended
  - It’s much larger than in traditional filesystems
- **A file is just “sliced” into chunks after each 128MB (or so)**
  - It does NOT matter if it is text, binary, compressed or not
  - It does matter later - when reading the data
- **HDFS is data-agnostic**

Image source: http://pixgood.com/slicing-bread.html

Splitting A File Into Blocks

- **Writing a file with a non-default block size**

```
$ hadoop fs -D dfs.block.size=268435456 \
 -put songs.txt /user/jeff
```

Replicating Blocks

- **The default replication factor is 3**
  - It can be specified per a file or a directory
  - It can be dynamically changed any time
    - It will automatically add/remove appropriate replicas
- **Tradeoff between**
  - Reliability, availability, performance
  - Disk space

Replicating Blocks

- **Writing a file with a default replication factor**

```
$ hadoop fs -put songs.txt /user/jeff
```

- **Changing the replication factor for a file**

```
$ hadoop fs -setrep -w 8 /user/jeff/songs.txt
```

- **Writing a file with a custom replication factor**

```
$ hadoop fs -D dfs.replication=8 \
-put songs.txt /user/jeff
```

Fault-Tolerance

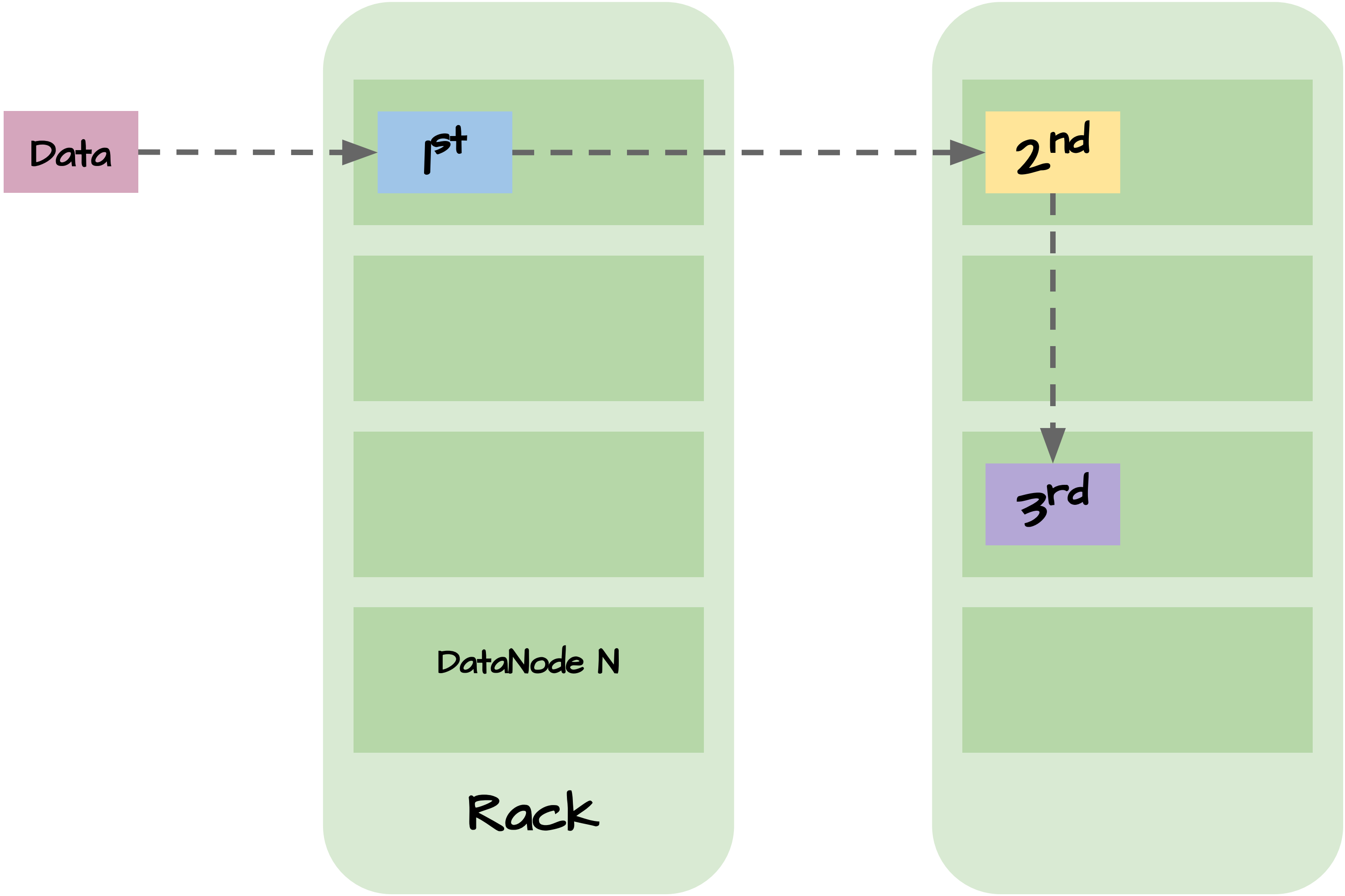
What happens when a node fails during write?

Fault-Tolerance

What happens when a node fails during write?

- **A write operation is finished successfully when at least `dfs.namenode.replication.min` nodes store each block**
  - Default is 1
- **Missing replicas will be re-created later asynchronously**

Default Block Replica Placement



Selecting Replicas

What if we have more than 3 replicas?

Selecting Replicas

What if we have more than 3 replicas?

- **Additional replicas are placed on random nodes**
  - No node with more than 1 replica of any block
  - If possible, no rack with more than 2 replicas of the same block
- **If can not assign enough nodes, then a block is under-replicated**
  - HDFS will recreate a missing replica when appropriate nodes appear

Reading A File From HDFS

Question?

- **What happens when a file is read from HDFS?**

```
$ hdfs dfs -cat /user/tiger/songs.txt
```

Reading A File From HDFS

(Incomplete) Answer!

- **Information about the file is needed!**
  - How was the file splitted into the blocks?
  - Where are these blocks located?
  - If a block is replicated multiple times, which replica to read from?

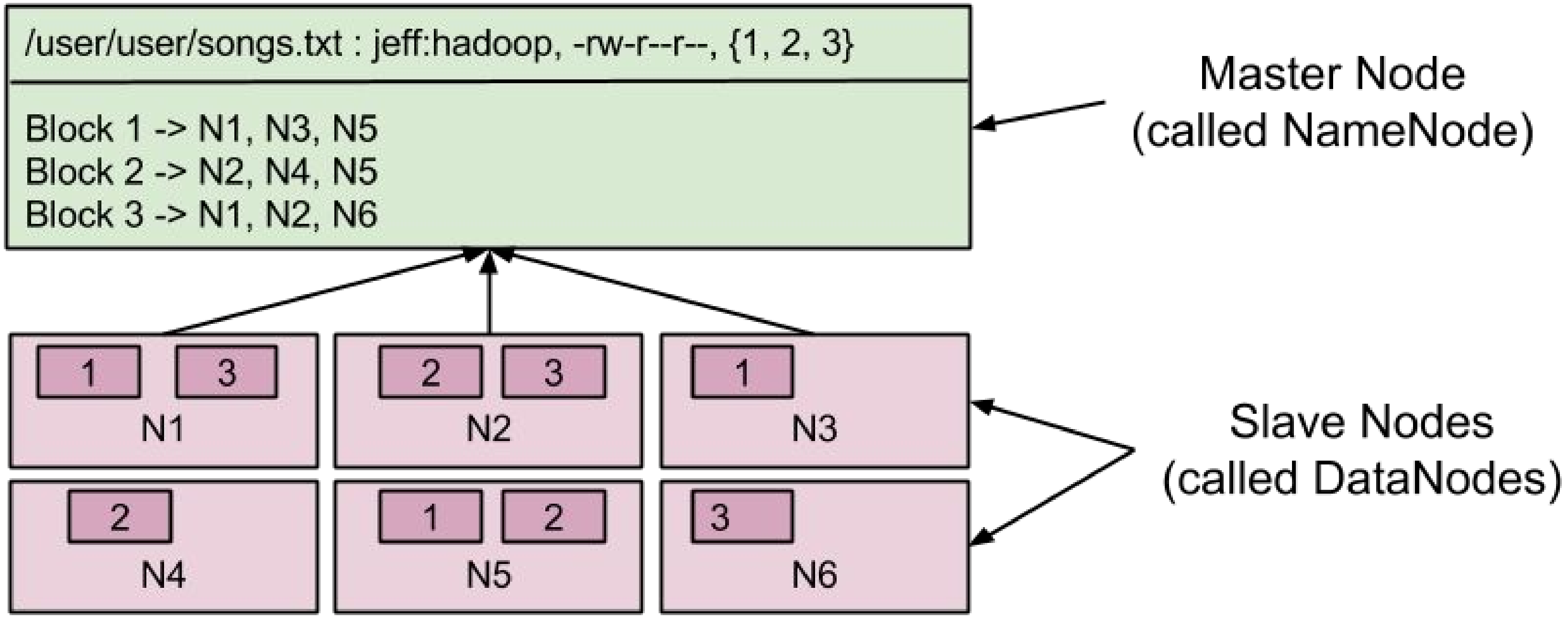
HDFS Metadata Information

- HDFS keeps information of each file and directory

PROPERTY	FILE	DIRECTORY
Full path	YES	YES
Replication factor	YES	
Last modification time	YES	YES
Last access time	YES	
Permissions and Ownership	YES	YES
Block size, List of blocks, File size	YES	
Namespace and diskspace quota		YES

Master And Slaves

- The Master manages metadata information
- The Slaves store blocks of data and serve them to the client



NameNode - The Master Daemon

- **The most important HDFS daemon**
- **Performs the metadata-related operations**
- **Keeps information in RAM (for fast response)**
  - The filesystem tree
  - Metadata for all files and directories
  - Names and locations of blocks
- **Metadata (but not all) is additionally stored on disks for reliability**

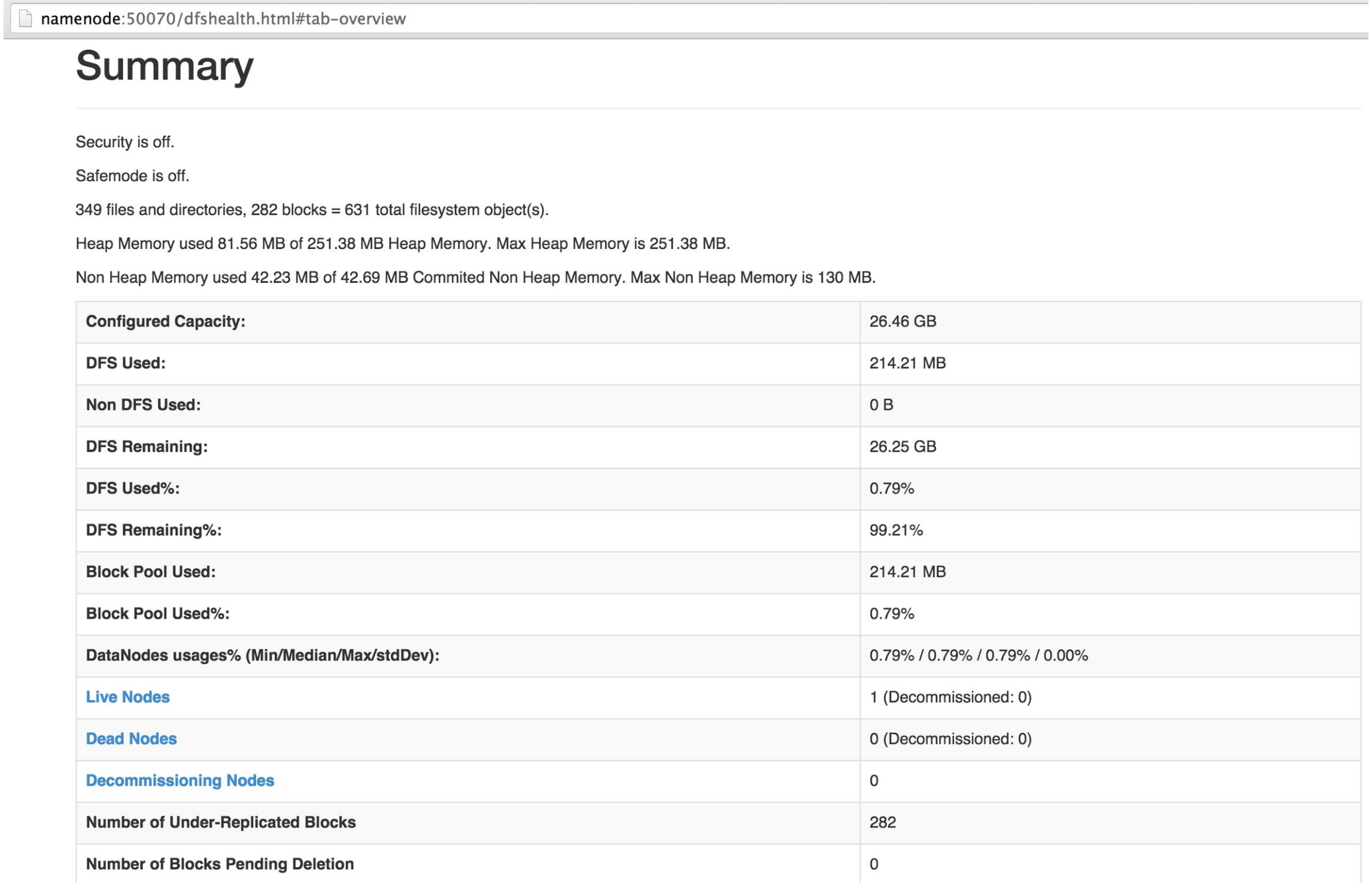
DataNode - A Slave Daemon

- **Stores and servers blocks of data**
- **A block is stored as a regular file on a local filesystem**
  - e.g. `blk_-992391354910561645` (and checksums in a separate file)
  - A block itself does not know which file it belongs to!
- **Sends a heartbeat message to the NN to say *“Hi! I am still alive*”**

Demo

HDFS Web UIs

NameNode Web UI



namenode:50070/dfshealth.html#tab-overview

Summary

Security is off.

Safemode is off.

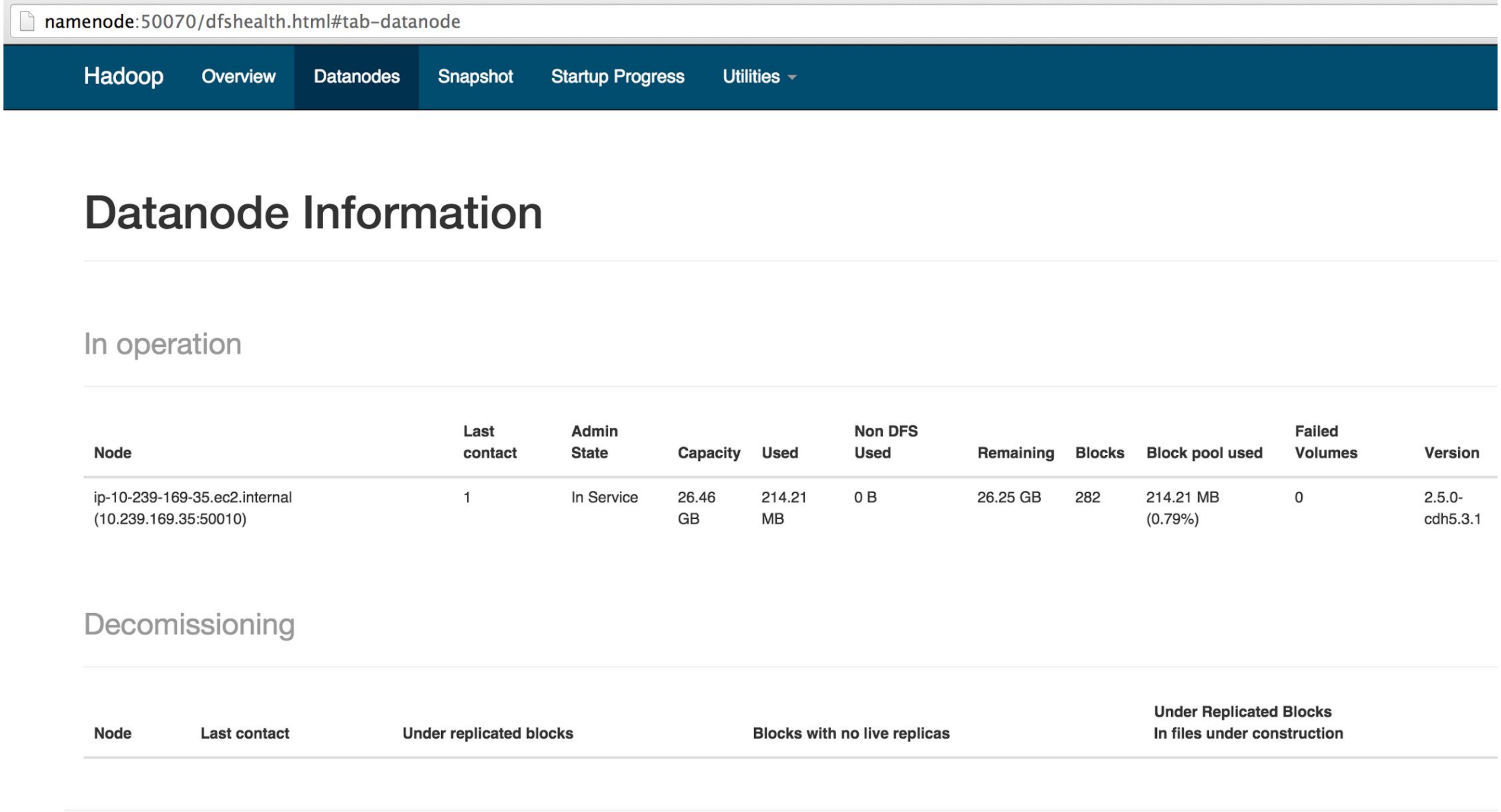
349 files and directories, 282 blocks = 631 total filesystem object(s).

Heap Memory used 81.56 MB of 251.38 MB Heap Memory. Max Heap Memory is 251.38 MB.

Non Heap Memory used 42.23 MB of 42.69 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

Configured Capacity:	26.46 GB
DFS Used:	214.21 MB
Non DFS Used:	0 B
DFS Remaining:	26.25 GB
DFS Used%:	0.79%
DFS Remaining%:	99.21%
Block Pool Used:	214.21 MB
Block Pool Used%:	0.79%
DataNodes usages% (Min/Median/Max/stdDev):	0.79% / 0.79% / 0.79% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Number of Under-Replicated Blocks	282
Number of Blocks Pending Deletion	0

DataNode Web UI



namenode:50070/dfshealth.html#tab-datanode

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-10-239-169-35.ec2.internal (10.239.169.35:50010)	1	In Service	26.46 GB	214.21 MB	0 B	26.25 GB	282	214.21 MB (0.79%)	0	2.5.0-cdh5.3.1

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Reading A File From HDFS

The Previous Question Again :)

- **OK, so what really happens when a file is read from HDFS?**

```
$ hdfs dfs -cat /user/jeff/songs.txt
```

Reading A File From HDFS

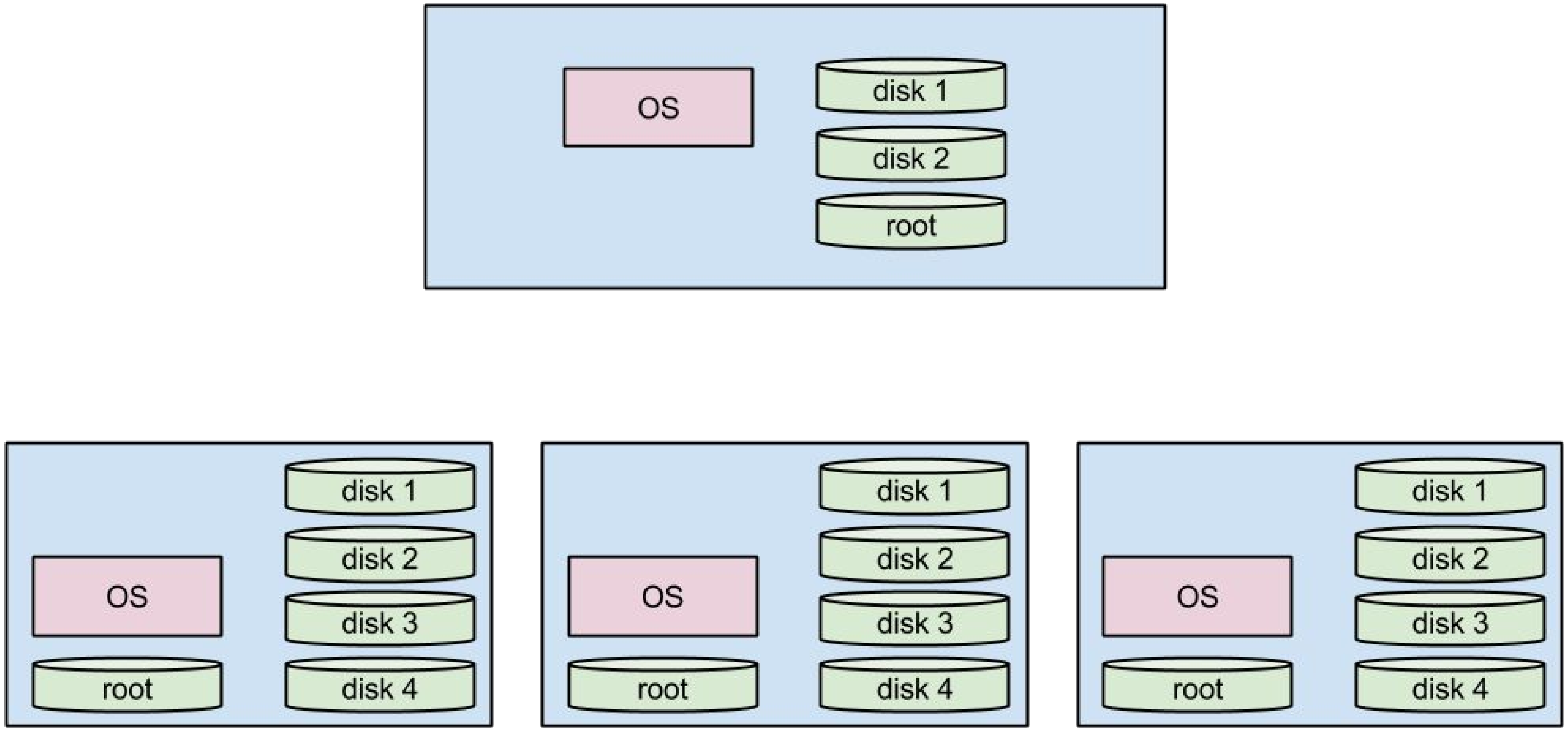
- **The NameNode knows:**
  - How a file is divided into blocks
  - Where these blocks are located
- **The NN automatically redirects a client to an appropriate DNs to read the content of a block**
  - The NN chooses a DN that is the “closest” to minimize the network transfer
- **Blocks of data are never sent through the NameNode**
  - To avoid the bottleneck!

Fault-Tolerance

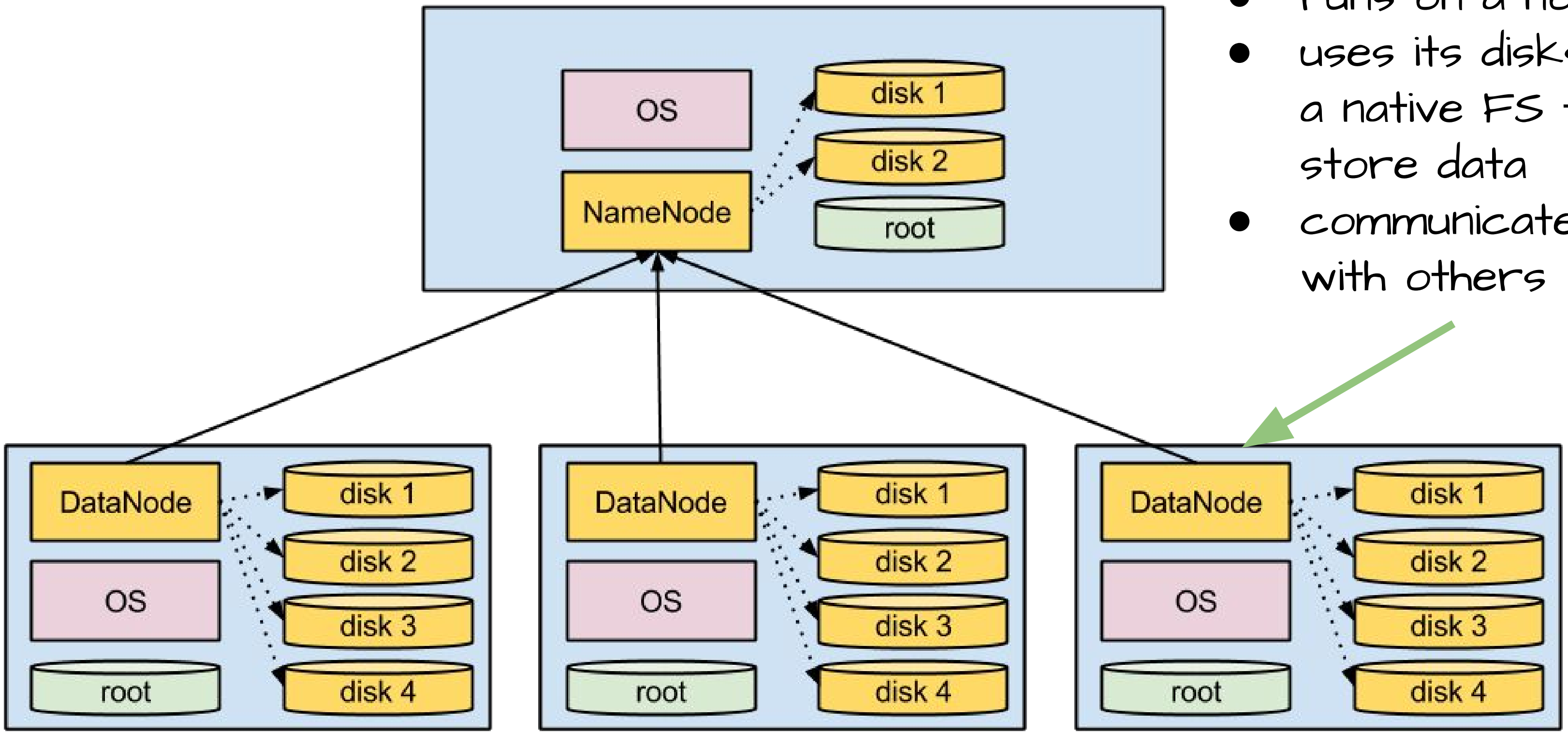
What happens when a DateNode fails during read?

- **A client is automatically and transparently redirected to other DataNode that has other replica of the same block**
  - User doesn’t write any custom code for that!

Cluster Without HDFS



Cluster With HDFS



A Java process that

- runs on a node
- uses its disks and a native FS to store data
- communicates with others

Technical Facts About HDFS

- **Runs as Java software installed on each node in a cluster**
  - e.g. if you kill DataNode, there is no HDFS on the node
- **Uses a native file system to store blocks as regular files**
  - e.g. ext3, ext4, xfs
- **Is data-agnostic**
  - Data in any format can be stored
- **Designed as master-slave architecture**

Fault-Tolerance

- **What happens when**
  - 1 DataNode
  - 3 DataNodes
  - A rack of DataNodes
  - NameNode

  crash(es)?

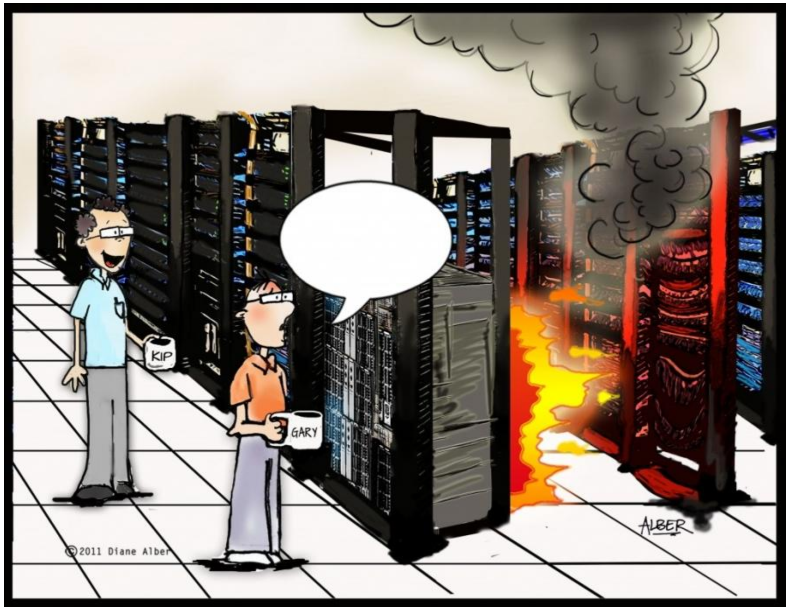
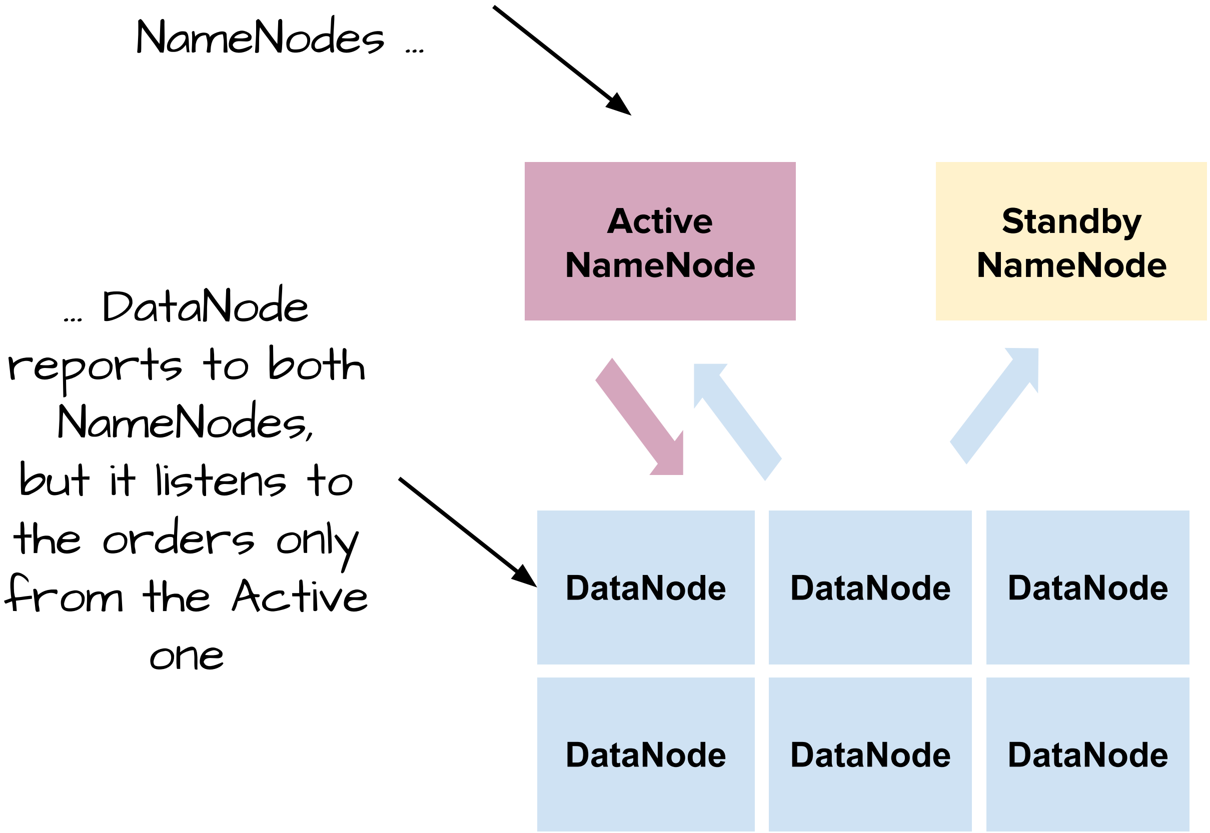


Image source:
http://kipandgary.com/blog/wp-content/uploads/2011/11/virtualized-server-1024x791(pp_w894_h690).jpg

NameNode HA (Manual Failover)

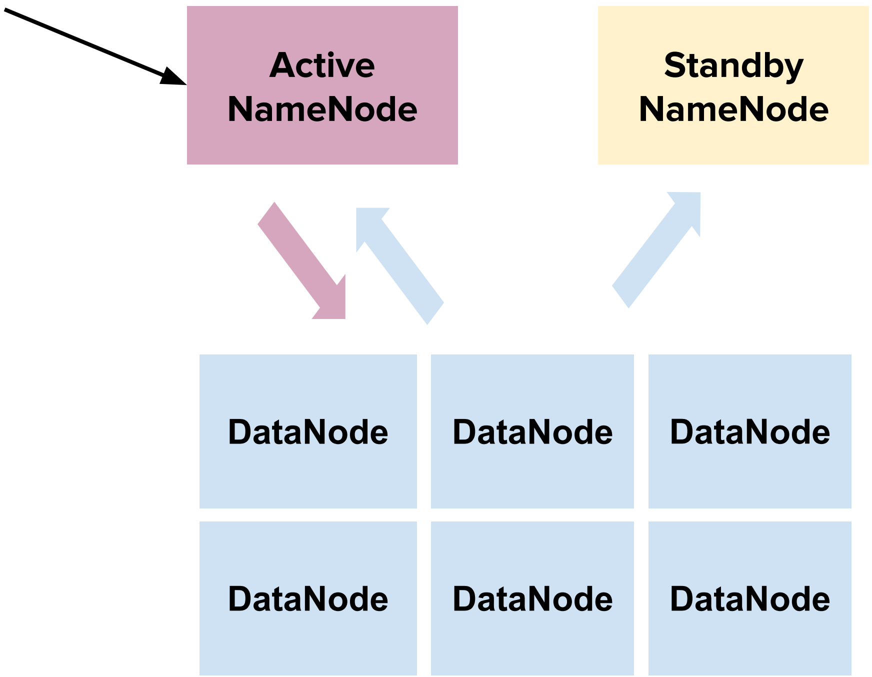
There are TWO NameNodes ...



... DataNode reports to both NameNodes, but it listens to the orders only from the Active one

NameNode HA (Manual Failover)

Only the Active NameNode responds to HDFS requests e.g. create, read, write, delete, list



Active NameNode

Standby NameNode

DataNode

DataNode

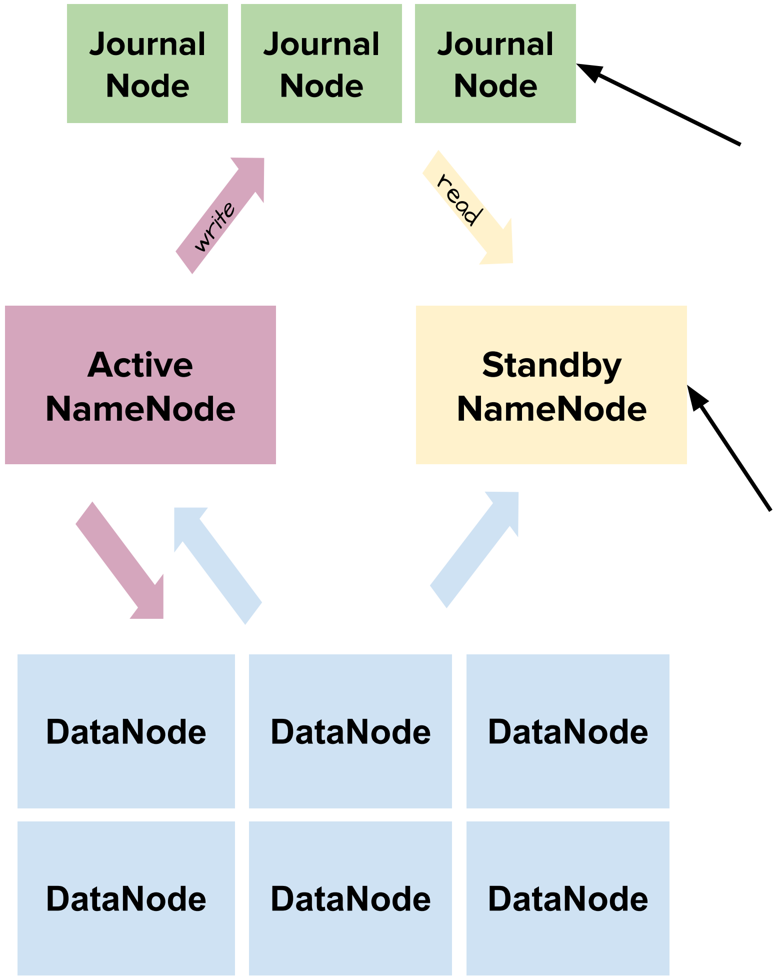
DataNode

DataNode

DataNode

DataNode

NameNode HA (Manual Failover)



Journal Node

Journal Node

Journal Node

Active NameNode

Standby NameNode

DataNode

DataNode

DataNode

DataNode

DataNode

DataNode

1. The state is shared on a quorum of Journal Nodes

2. The Standby simultaneously reads and applies the edits to be in-sync

Answers!

1. NameNode
2. DataNodes
3. Java
4. Apache Foundation
5. Fault-tolerance
6. Commodity hardware
7. Scalability
8. Heterogenous nodes
9. Triplicated block in HDFS

Editing Files In HDFS

Question

- **How would you modify the content of a file in HDFS?**

Read-Write Access To File

Answer

- **HDFS does not support a random write operation**
  - You can only append to the end of the file
- **If you need to modify the content of a file, then**
  1. Download a file from HDFS
  2. Make modifications in a local filesystem
  3. Remove the file from HDFS
  4. Upload a new version of a file to HDFS

Properties of HDFS

- **Assumes streaming access**
  - Reading and writing data from the beginning to the end
  - “Write once - read many times” pattern
  - The loose analogy to CD-ROM
- **Focuses more on throughput than latency**
  - The analogy to a big truck (not Ferrari)
- **Likes very large files**
  - Why? :)

Bad Use-Cases For HDFS

- **Low-latency requests**
  - File servers, RDBMS, NoSQL are better
- **Random read or random write requests**
  - RDBMS, NoSQL are better
- **An extremely high number of small files**

HDFS Block

Question

- **Why a block does not know which file it belongs to?**

HDFS Block

Answer

- **Design decision for simplicity and performance**
- **Filename, permissions, ownership might change**
  - It would require updating all block replicas that belong to a file

HDFS Metadata

Question

- **Why the NameNode does NOT store information about block locations on disks?**

HDFS Metadata

Answer

- **Design decision for simplicity**
- **Locations of block replicas may change over time**
  - e.g. balancing the cluster (moving blocks between nodes)

Exercise

Uploading Data To HDFS

http://bit.ly/1SqouAq

Pages 1-6

Bonus!

Accessing HDFS

Question

- **What are the biggest annoyances when accessing HDFS using the `hdfs dfs` command?**

Accessing HDFS

Answer

- **It is slow!**
  - JVM startup takes time
  - It loads a bunch of JAR files that you might not need
- **No tab-completion**
- **Problematic to integrate with own code e.g. Java, Python**

Snakebite

- **Python library for interacting with HDFS**
- **Both a client library and a command line tool**
  - No all commands are supported
- **Much faster than Hadoop CLI**
  - Uses Protocol Buffers to communicate to the NameNode
- **Tab completion included!**
- **Open-sourced at Github by Spotify**
- **Try it now!**

```
$ snakebite /use<TAB>
```

Accessing HDFS

- **Java API**
  - Native way, because HDFS is written in Java
- **HttpFs and WebHDFS**
  - HTTP REST interface via `wget` or `curl`
  - WebHDFS scales better
  - HttpFs supports HA

Hadoop
Deployment Modes

Hadoop Deployment Modes

- **Local mode**
- **Pseudo-distributed mode**
- **Fully-distributed mode**

Local Mode

- Everything runs in single process
- Useful for testing

Pseudo-Distributed Mode

- **A single-node Hadoop cluster**
- **Each daemon runs in its own JVM**
  - All daemons run on the same machine
- **HDFS is used for storing data**
  - Replication factor is set to 1
  - Different filesystem can be used e.g. local or S3
- **Suitable for learning Hadoop and experimenting**

Hadoop Sandbox

- **All components pre-installed in pseudo-distributed mode**
  - Configuration adjusted for running on one machine
- **All Hadoop daemons already started**
- **Perfect to experiment with Hadoop**
- **Delivered by vendors to download**
  - Obviously, for free!

Fully-Distributed Mode

- **A multi-node Hadoop cluster**
  - Scales to thousands of nodes
- **Each daemon runs in its own JVM on a separate machine**
  - For small clusters, master daemons can be collocated
- **HDFS is used for storing data**
  - Different filesystem can be used e.g. S3
- **Suitable for production use**

Exercise

Interacting with HDFS using CLI and HUE

Chapter

YARN - Yet Another Resource Negotiator

Managing Computational Resources

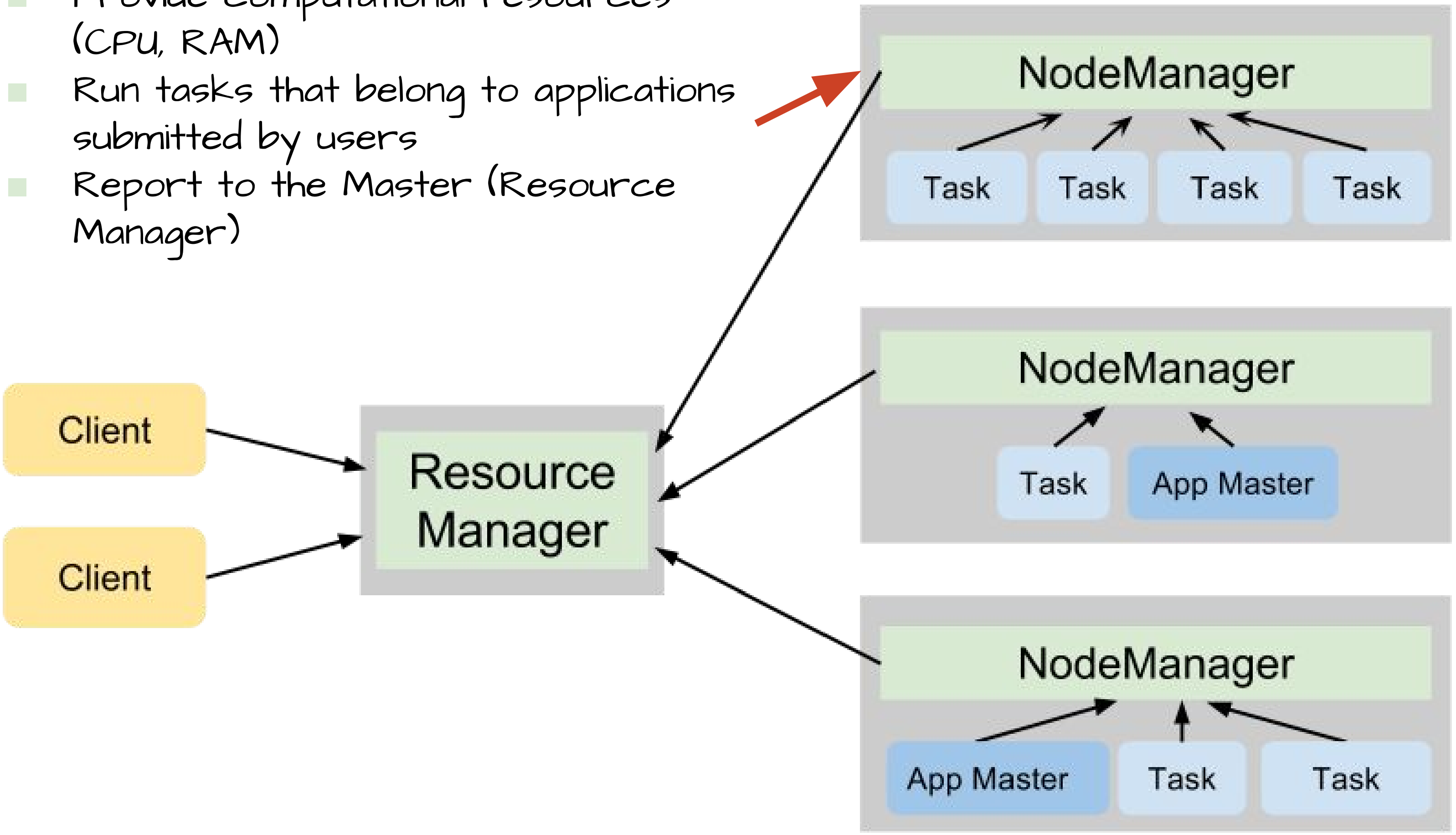
- **How to manage computational resources (e.g. memory, processor) on the cluster?**

Recommended Solution

- **YARN**

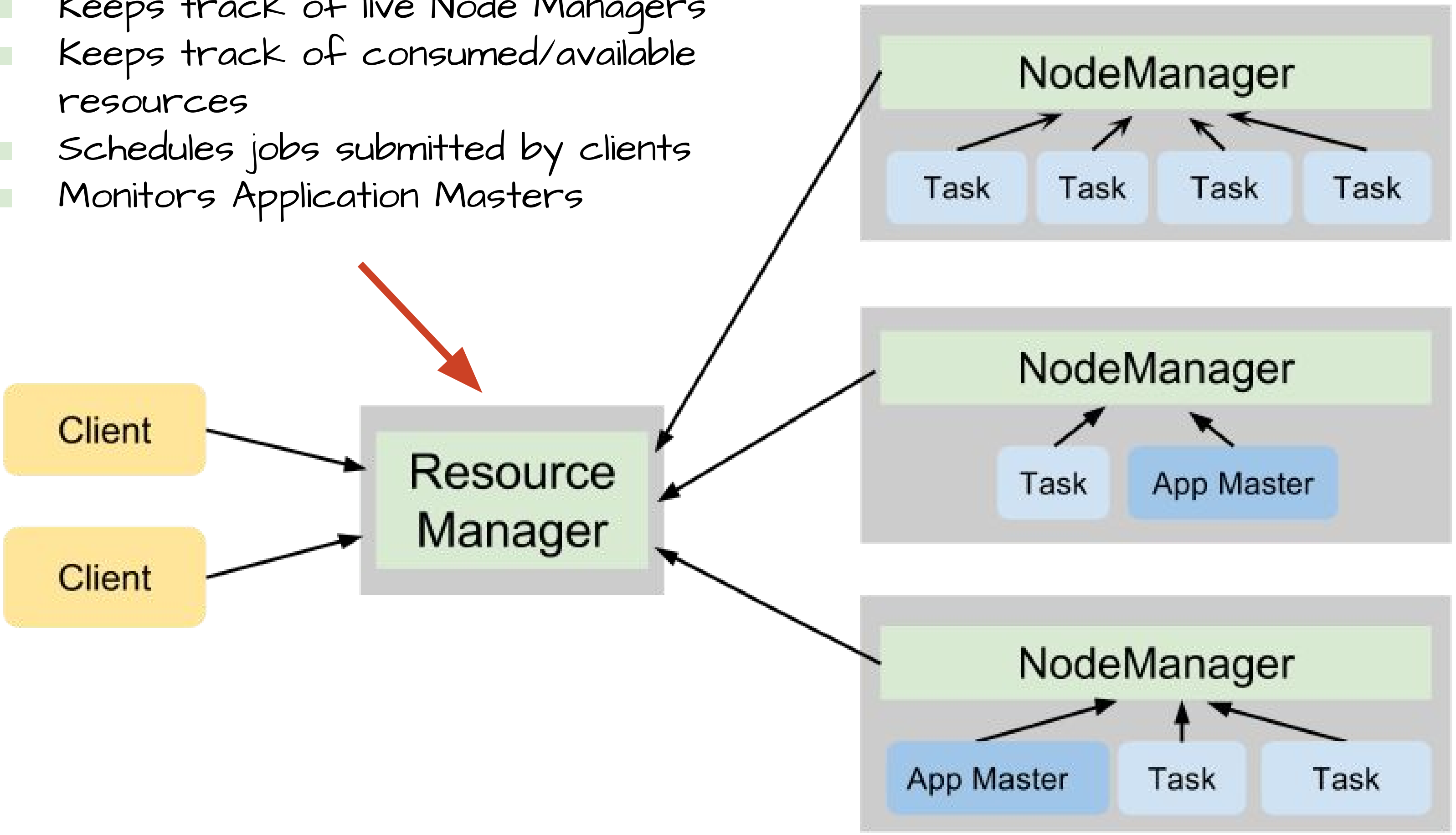
Node Managers (Slaves)

- Provide computational resources (CPU, RAM)
- Run tasks that belong to applications submitted by users
- Report to the Master (Resource Manager)

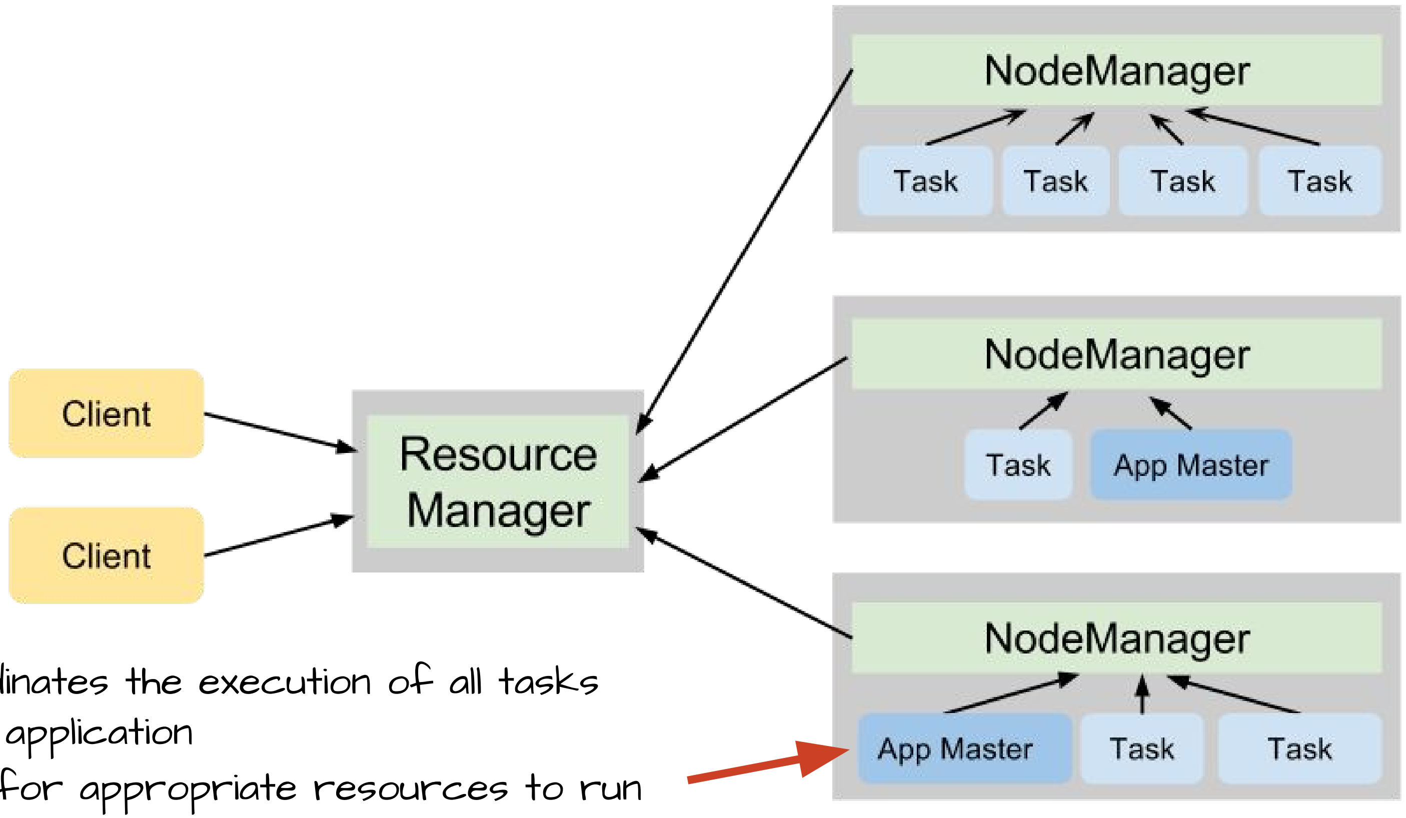


Resource Manager (Cluster Master)

- Keeps track of live Node Managers
- Keeps track of consumed/available resources
- Schedules jobs submitted by clients
- Monitors Application Masters

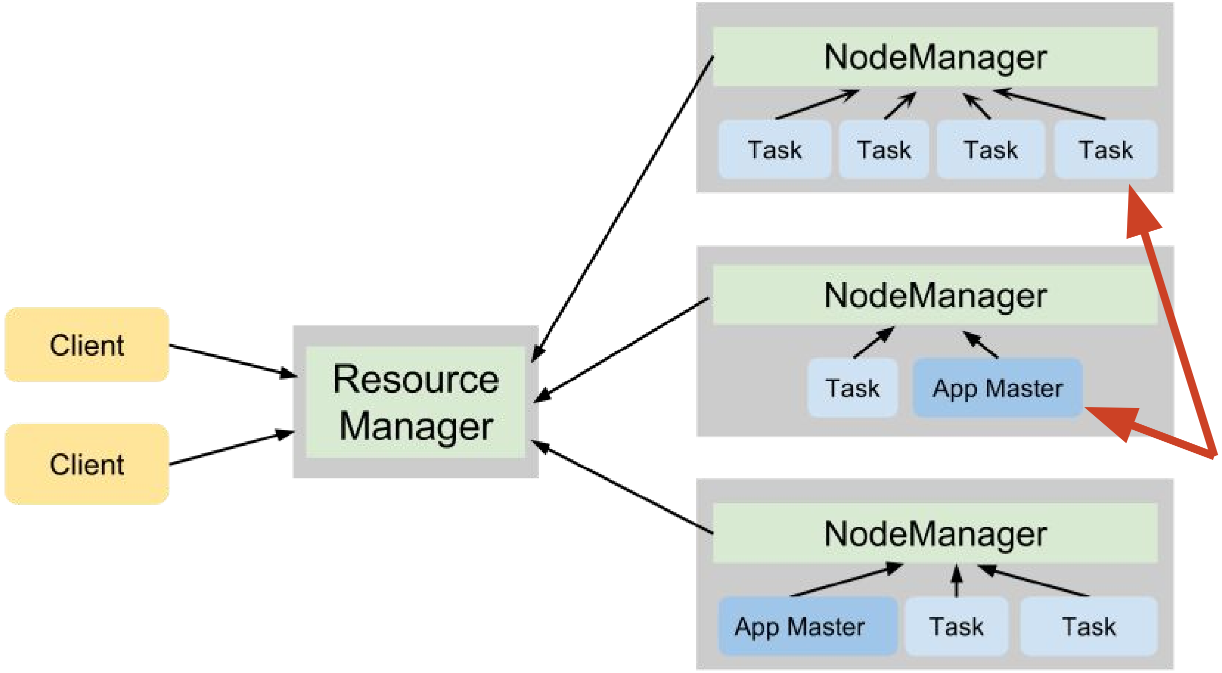


Application Master



- Coordinates the execution of all tasks within application
- Asks for appropriate resources to run tasks
- Runs on the Node Manager

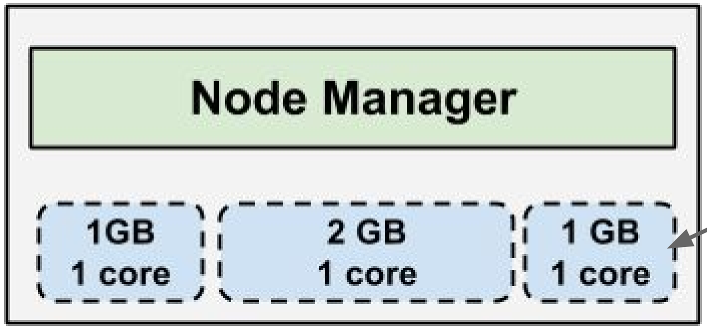
Containers



- NodeManagers offer resources in form of containers
- Run different types of tasks in containers (e.g. regular tasks or Application Masters)
- Can have different sizes (in terms of RAM and number of CPU cores)

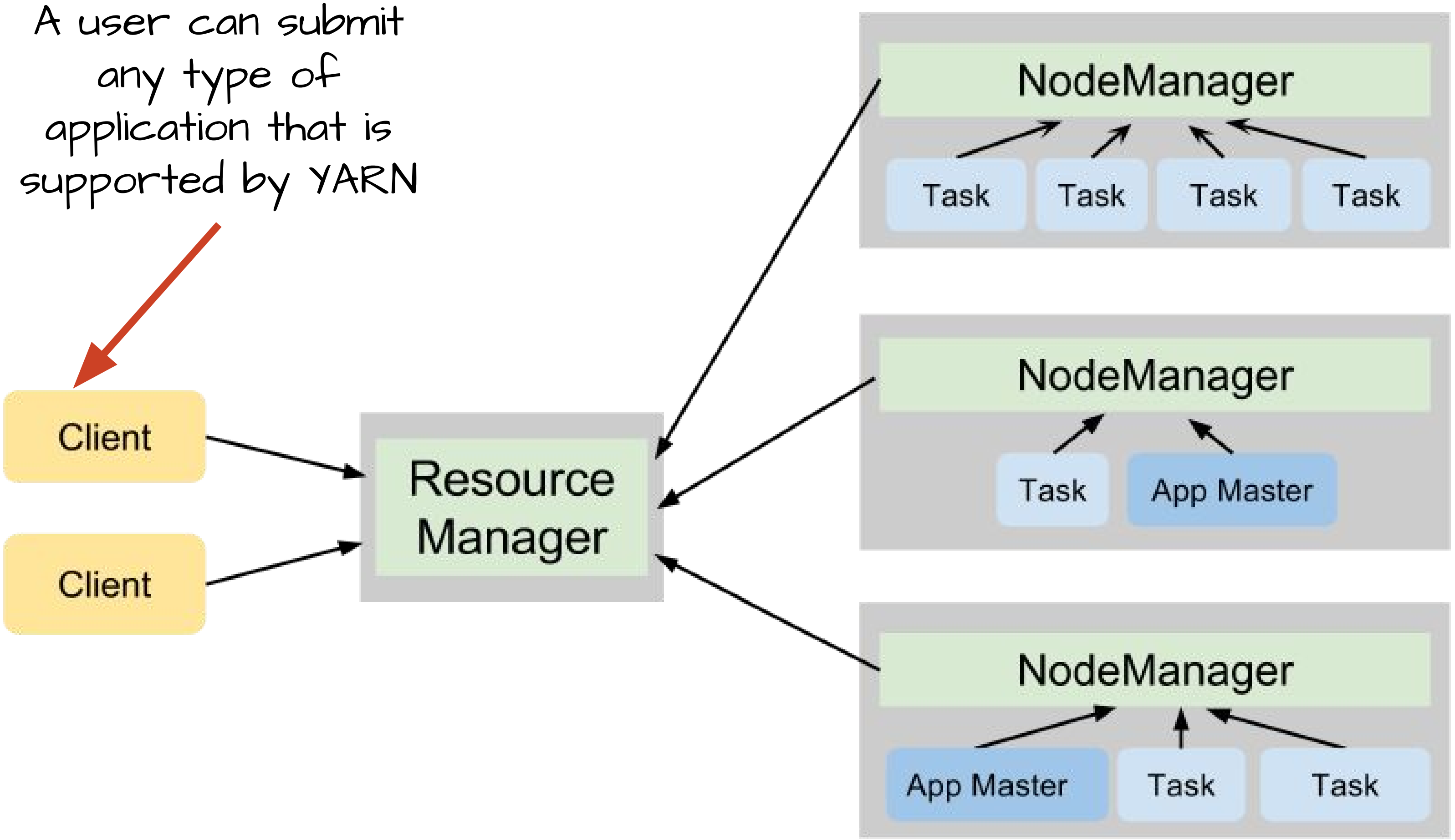
Container

- **A combination of memory, CPU, disk and network IO**
  - e.g. 2GB RAM, 1 CPU, 1 disk
- **Currently only memory and CPU (YARN-3) are supported**



Containers are dynamically created and deleted

Applications



A user can submit any type of application that is supported by YARN

Demo

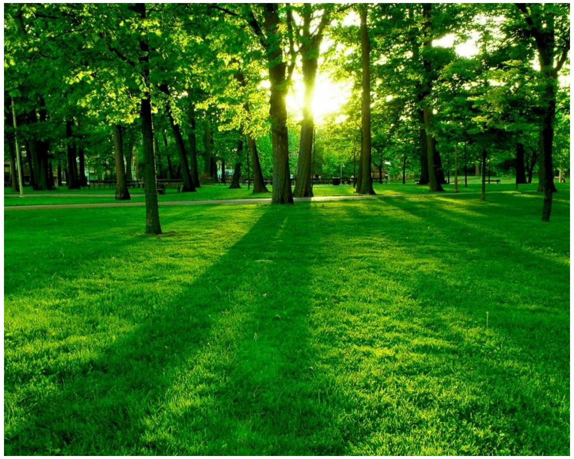
YARN Web UIs

Generic Approach

- **The RM, NMs and containers don’t care about the type of an application or a task**
  - They only focus on consuming computing resources
- **All application-specific code is located inside the AM**
- **A single YARN cluster can run various workloads**
  - MapReduce, Spark, Giraph, Storm (in-progress)

Benefits Of Generic Approach

- **Lower operational costs**
  - Only one "do-it-all" cluster must be maintained
- **Higher cluster utilization**
  - Temporarily unused resources can be used by another framework
- **Reduced data motion**
  - No need to move data between YARN cluster and other cluster

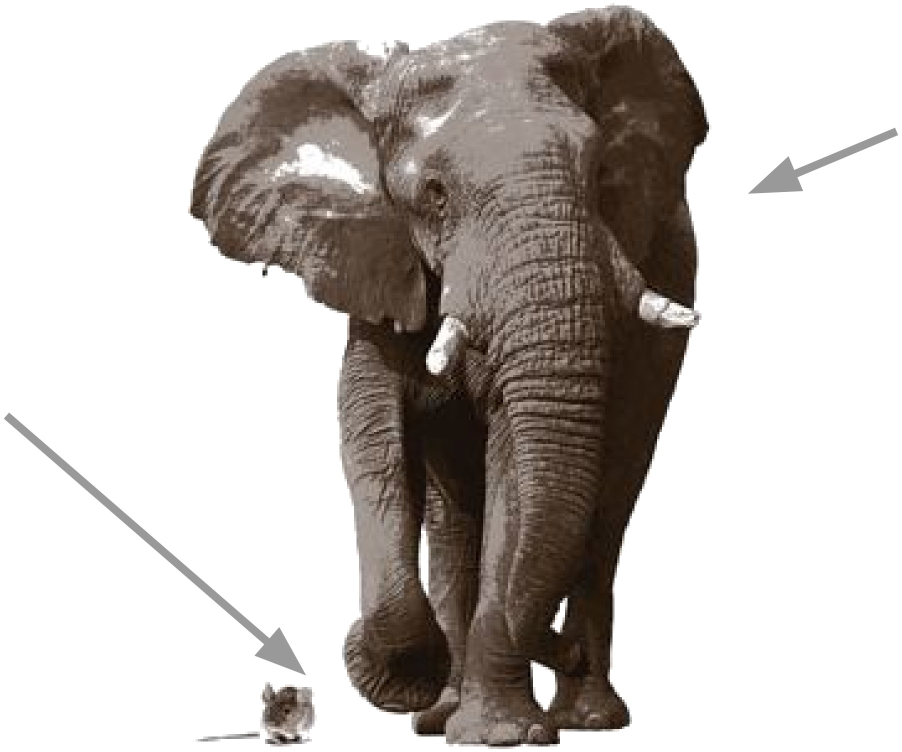


Frameworks Powered By YARN

- **MapReduce**
- **Apache Spark**
- **Apache Tez**
- **Cloudera Impala**
  - Fast SQL on Hadoop
- **Apache Giraph**
  - A graph processing framework
- **Apache Storm**
  - Real-time stream processing
- **Find more!**
  - http://wiki.apache.org/hadoop/PoweredByYarn
  - or implement

Sending Computation To Data

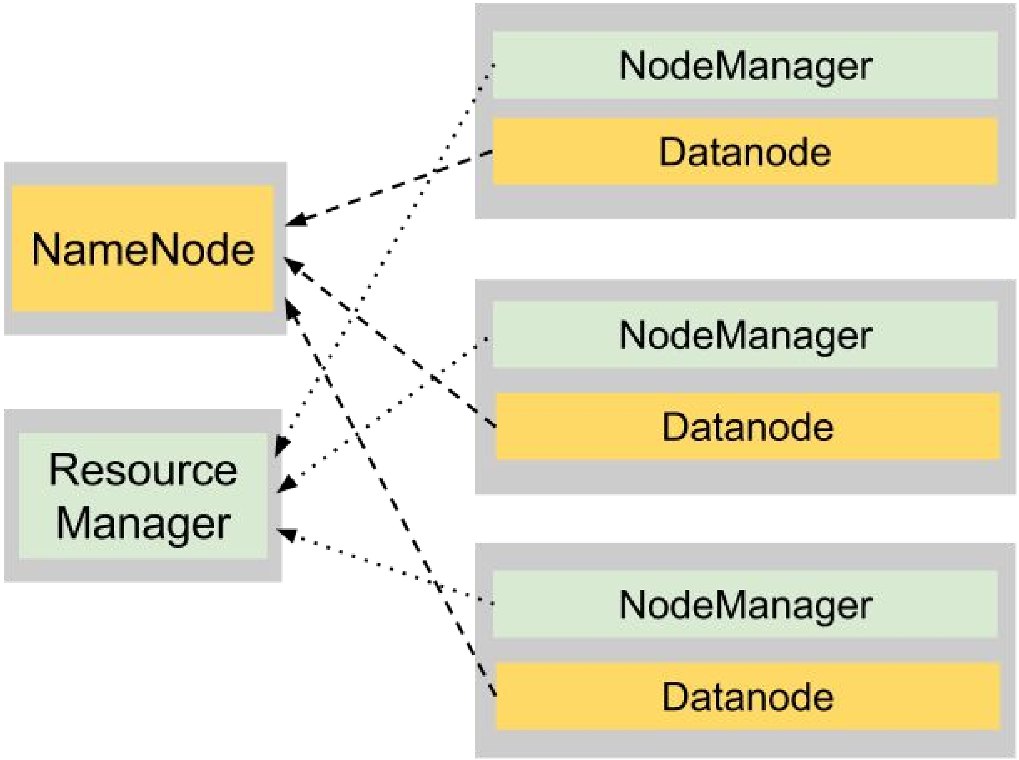
- **It is more efficient to send computation to data, isn't?**



Large volume of data

Computation e.g. a jar file

HDFS + YARN = Core Hadoop



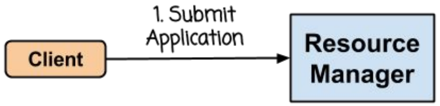
1. NMs should be collocated with DNs

2. The RM tries to schedule tasks on a node which is the closest to the data

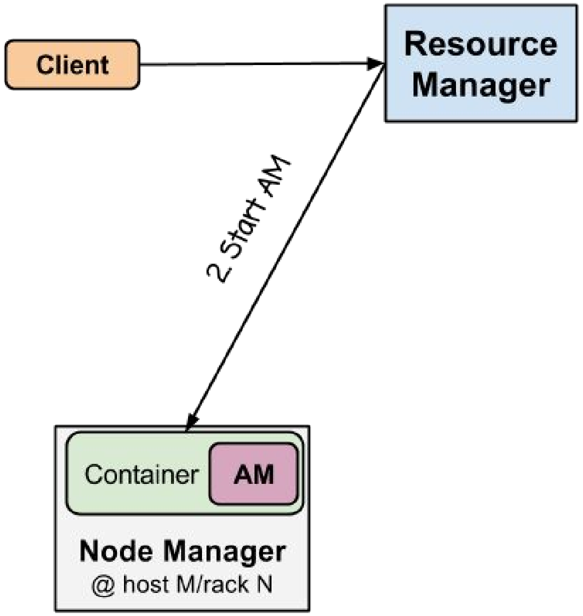
3. Large volumes of data don't have to be sent over the network

Application Submission in YARN

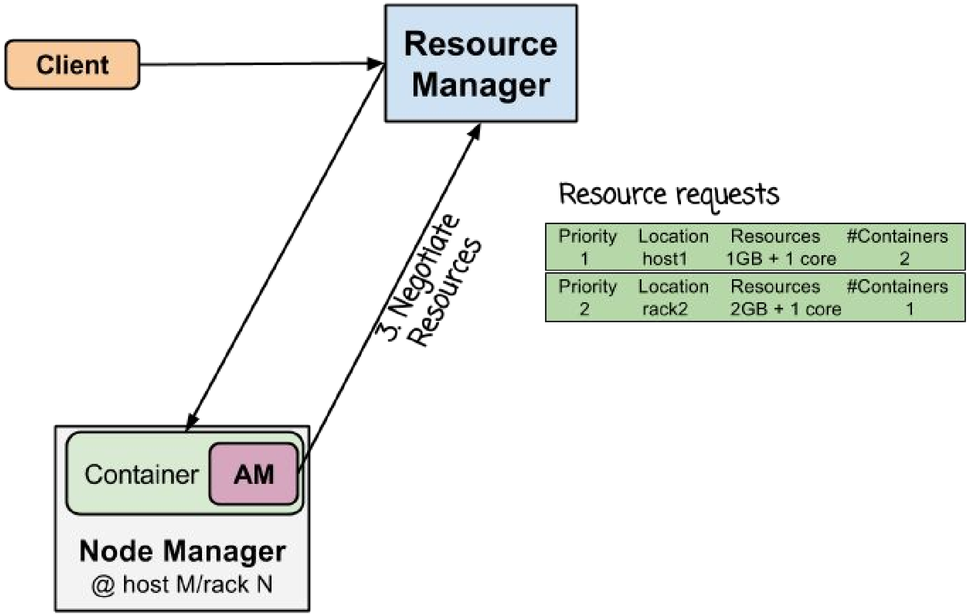
Application Submission In YARN



Application Submission In YARN



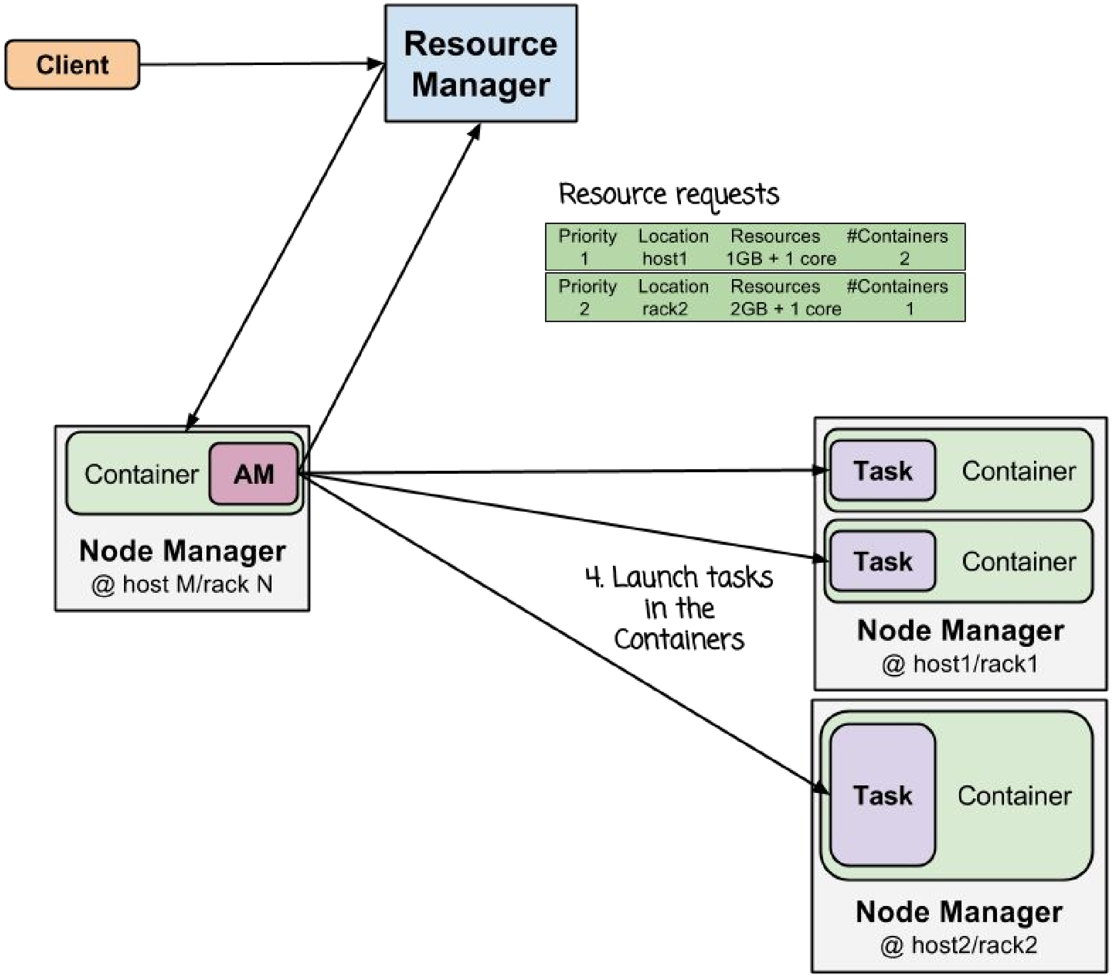
Application Submission In YARN



Resource requests

Priority	Location	Resources	#Containers
1	host1	1GB + 1 core	2
Priority	Location	Resources	#Containers
2	rack2	2GB + 1 core	1

Application Submission In YARN

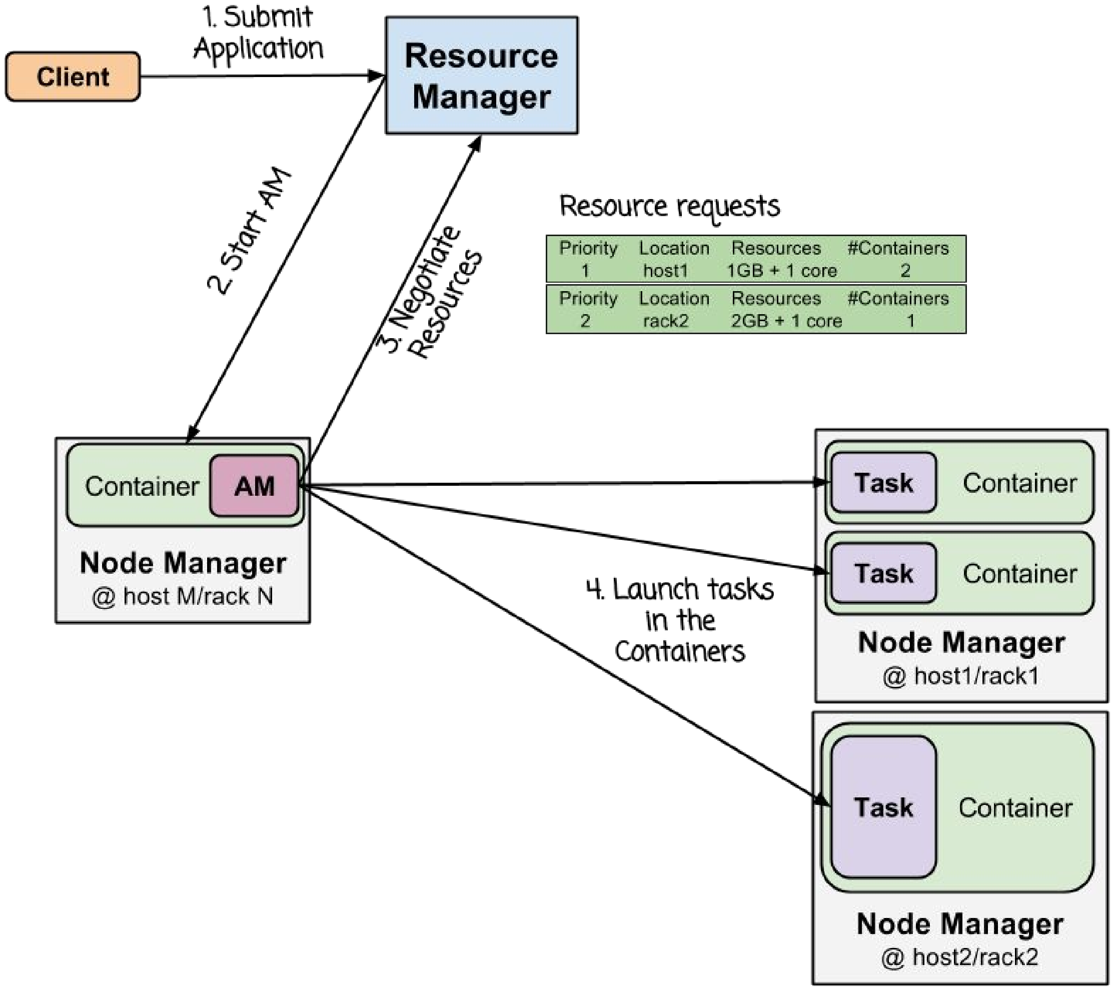


Resource requests

Priority	Location	Resources	#Containers
1	host1	1GB + 1 core	2
Priority	Location	Resources	#Containers
2	rack2	2GB + 1 core	1

4. Launch tasks in the Containers

Application Submission In YARN



Resource requests

Priority	Location	Resources	#Containers
1	host1	1GB + 1 core	2
Priority	Location	Resources	#Containers
2	rack2	2GB + 1 core	1

Easier Processing Of Big Data

- **How to easily query terabytes of data on hundreds of machines?**

Back To Facebook In ~2008

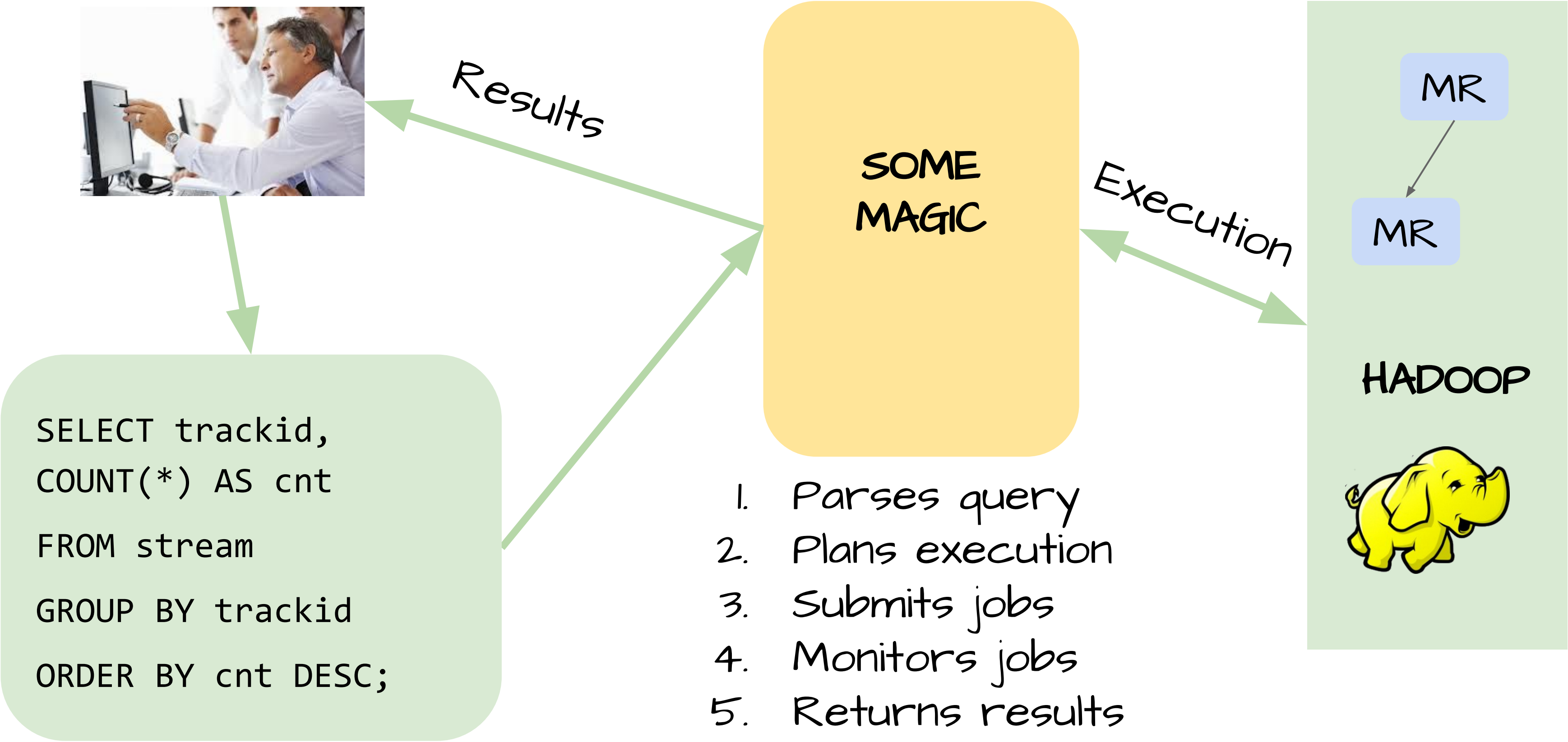
Their reality

- **Hadoop used at production**
- **Many analyst with SQL and RDBMS skills**
  - Not good at Java and MapReduce algorithms
- **Many BI and dashboarding tools integrated with SQL**

Their conclusion

- **Smooth migration from SQL to Hadoop is needed !!!**

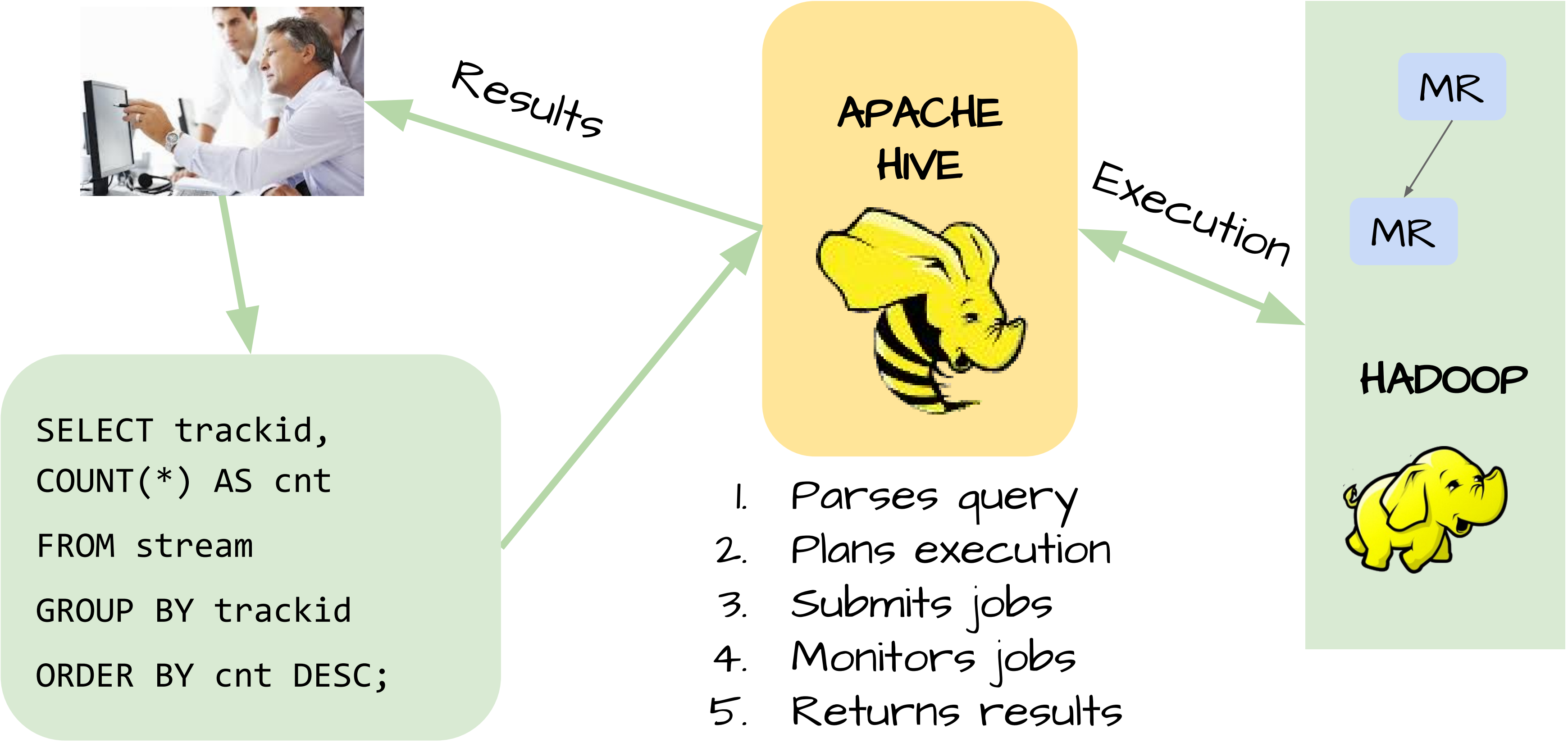
SQL On Hadoop



```
SELECT trackid,
COUNT(*) AS cnt
FROM stream
GROUP BY trackid
ORDER BY cnt DESC;
```

1. Parses query
2. Plans execution
3. Submits jobs
4. Monitors jobs
5. Returns results

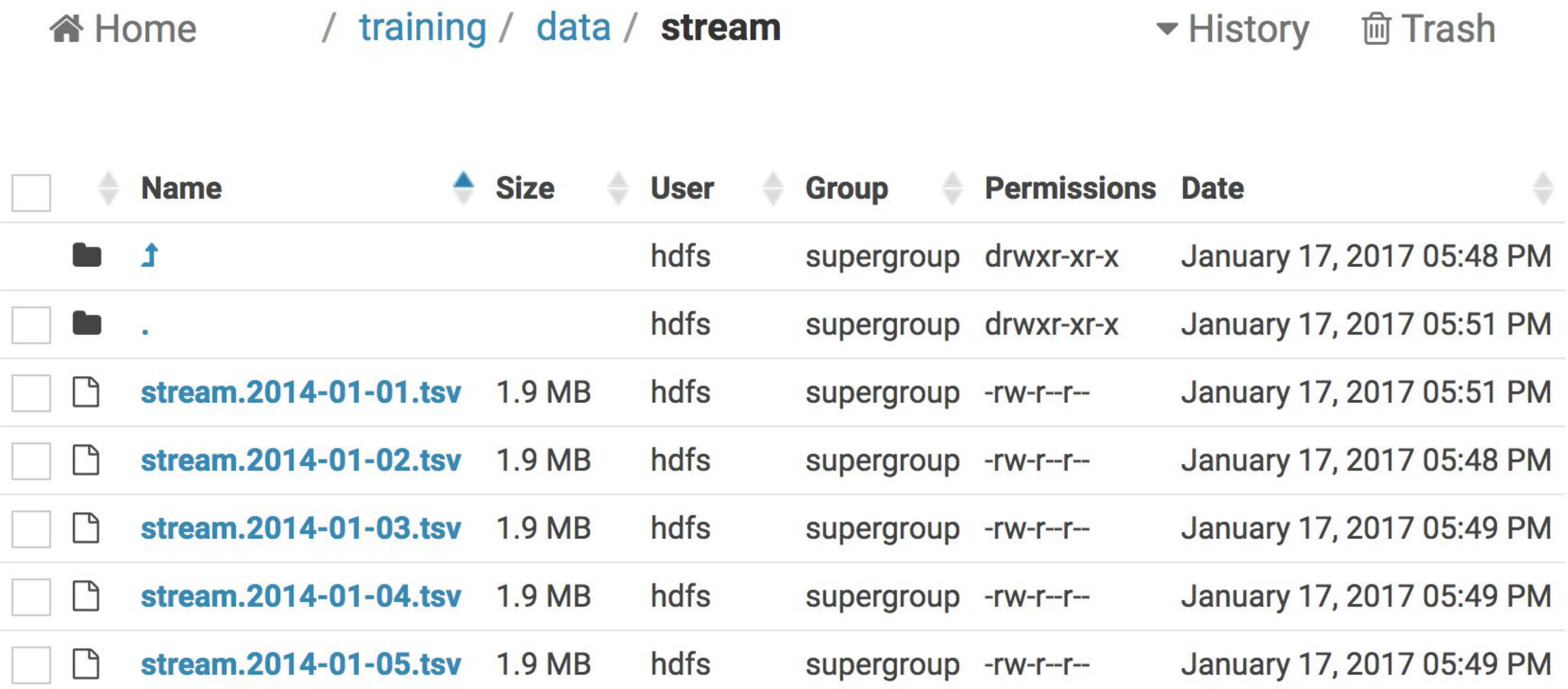
Apache Hive



```
SELECT trackid,
COUNT(*) AS cnt
FROM stream
GROUP BY trackid
ORDER BY cnt DESC;
```

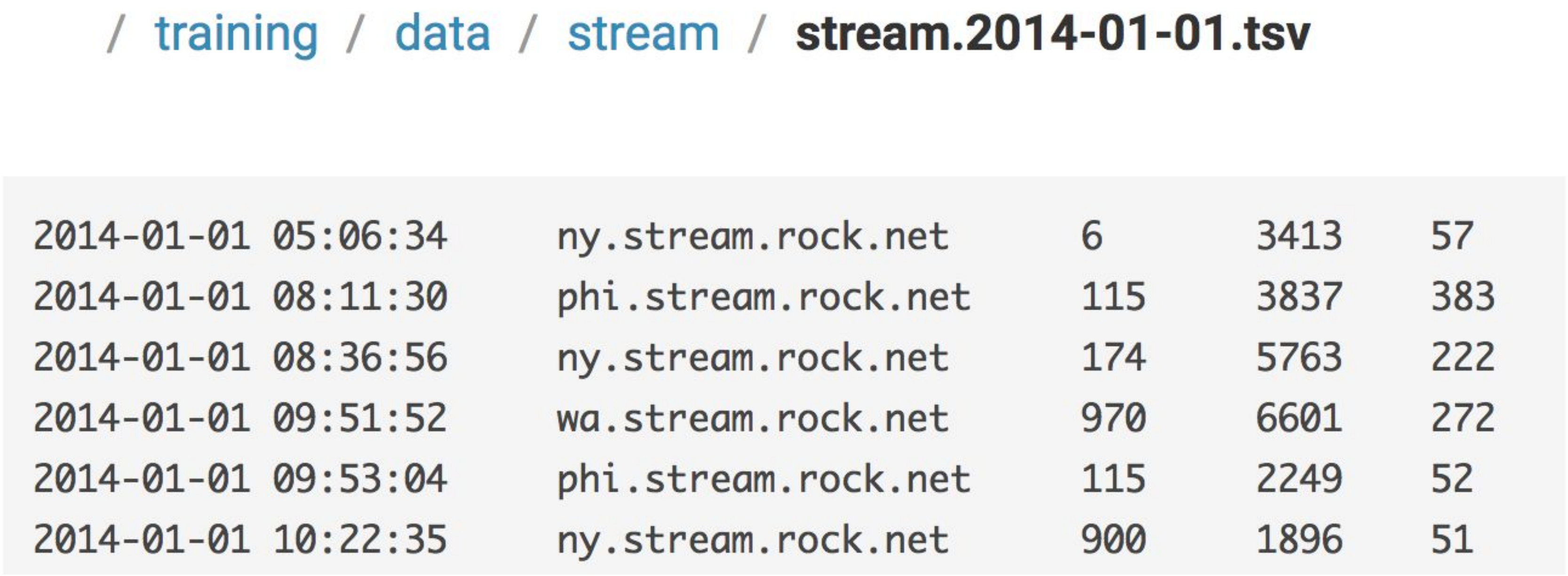
1. Parses query
2. Plans execution
3. Submits jobs
4. Monitors jobs
5. Returns results

HDFS Directory



□		Name	Size	User	Group	Permissions	Date
	📁	↑		hdfs	supergroup	drwxr-xr-x	January 17, 2017 05:48 PM
□	📁	.		hdfs	supergroup	drwxr-xr-x	January 17, 2017 05:51 PM
□	📄	stream.2014-01-01.tsv	1.9 MB	hdfs	supergroup	-rw-r--r--	January 17, 2017 05:51 PM
□	📄	stream.2014-01-02.tsv	1.9 MB	hdfs	supergroup	-rw-r--r--	January 17, 2017 05:48 PM
□	📄	stream.2014-01-03.tsv	1.9 MB	hdfs	supergroup	-rw-r--r--	January 17, 2017 05:49 PM
□	📄	stream.2014-01-04.tsv	1.9 MB	hdfs	supergroup	-rw-r--r--	January 17, 2017 05:49 PM
□	📄	stream.2014-01-05.tsv	1.9 MB	hdfs	supergroup	-rw-r--r--	January 17, 2017 05:49 PM

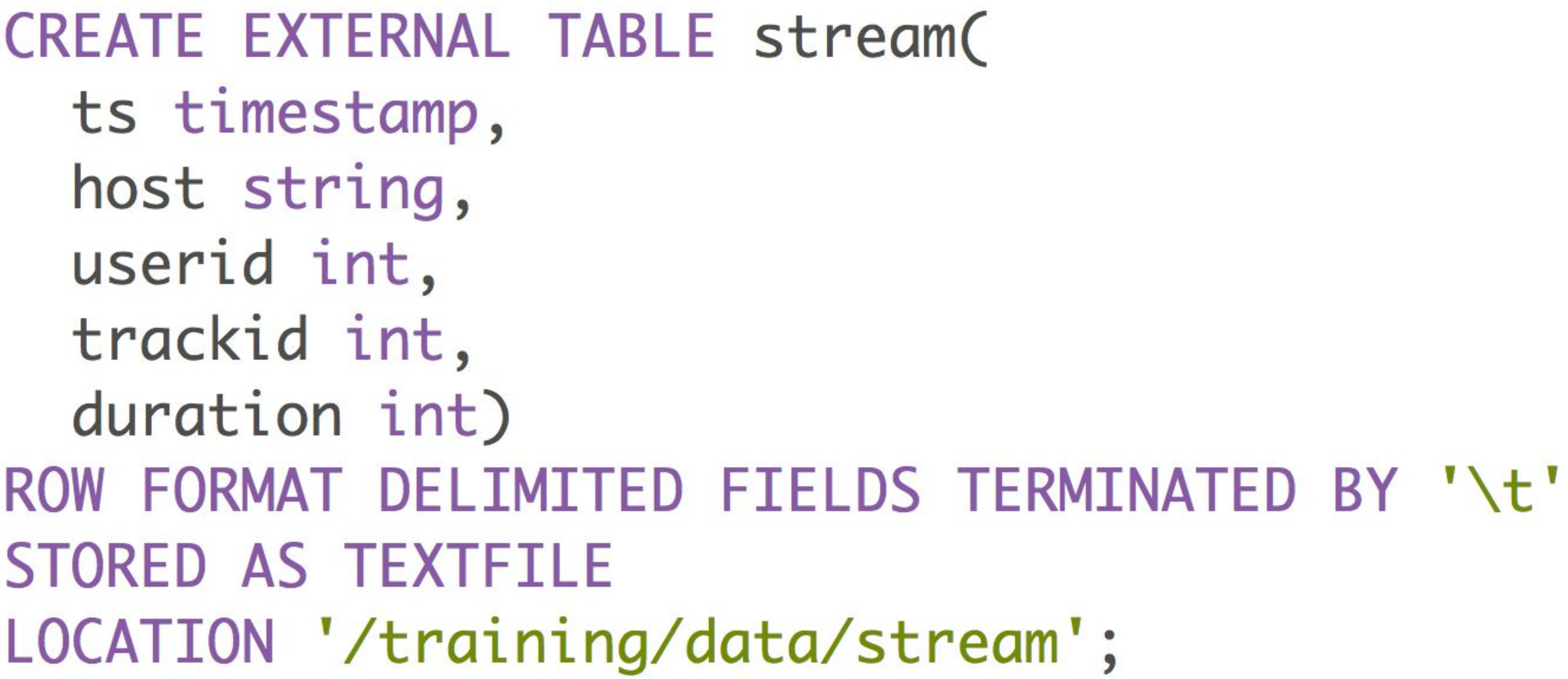
HDFS File



/ training / data / stream / **stream.2014-01-01.tsv**

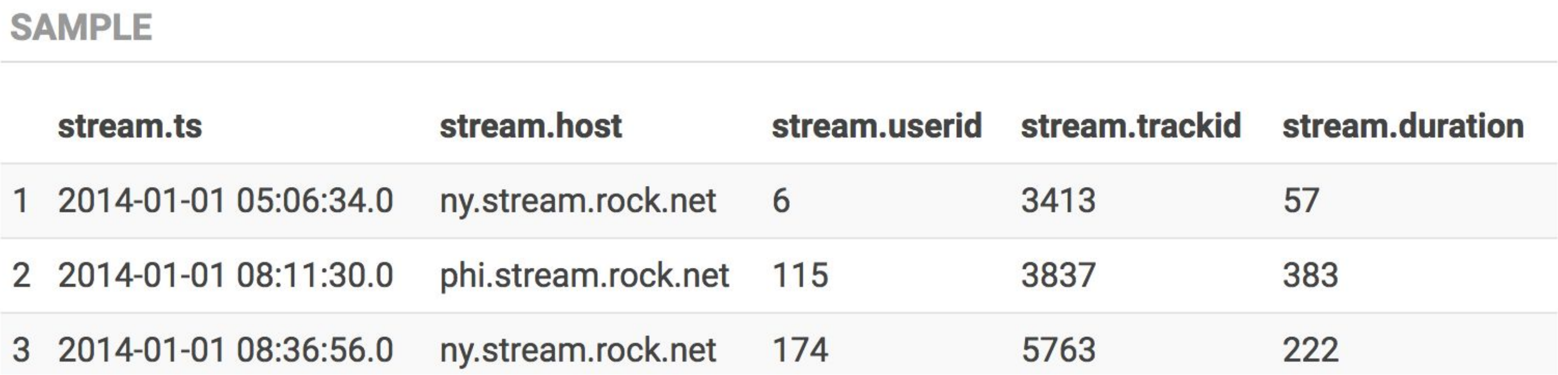
```
2014-01-01 05:06:34     ny.stream.rock.net      6       3413    57
2014-01-01 08:11:30     phi.stream.rock.net     115     3837    383
2014-01-01 08:36:56     ny.stream.rock.net      174     5763    222
2014-01-01 09:51:52     wa.stream.rock.net      970     6601    272
2014-01-01 09:53:04     phi.stream.rock.net     115     2249    52
2014-01-01 10:22:35     ny.stream.rock.net      900     1896    51
```

Creating Hive Table



```
CREATE EXTERNAL TABLE stream(
  ts timestamp,
  host string,
  userid int,
  trackid int,
  duration int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/training/data/stream';
```

Hive Table Abstraction



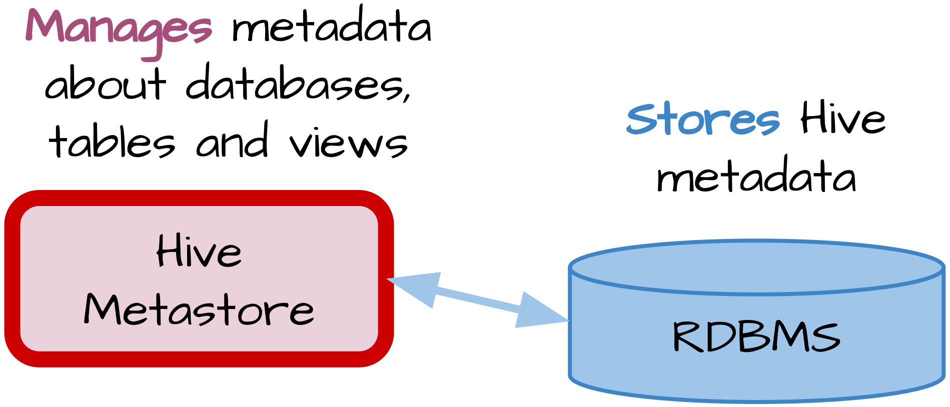
SAMPLE

	stream.ts	stream.host	stream.userid	stream.trackid	stream.duration
1	2014-01-01 05:06:34.0	ny.stream.rock.net	6	3413	57
2	2014-01-01 08:11:30.0	phi.stream.rock.net	115	3837	383
3	2014-01-01 08:36:56.0	ny.stream.rock.net	174	5763	222

Hive Metadata

- **For each table, we need to know**
  - The schema of the table (columns and types)
  - Location of the table in HDFS
  - Format of the data e.g. binary, text
  - How to convert the content of files into rows and columns

Hive Metastore



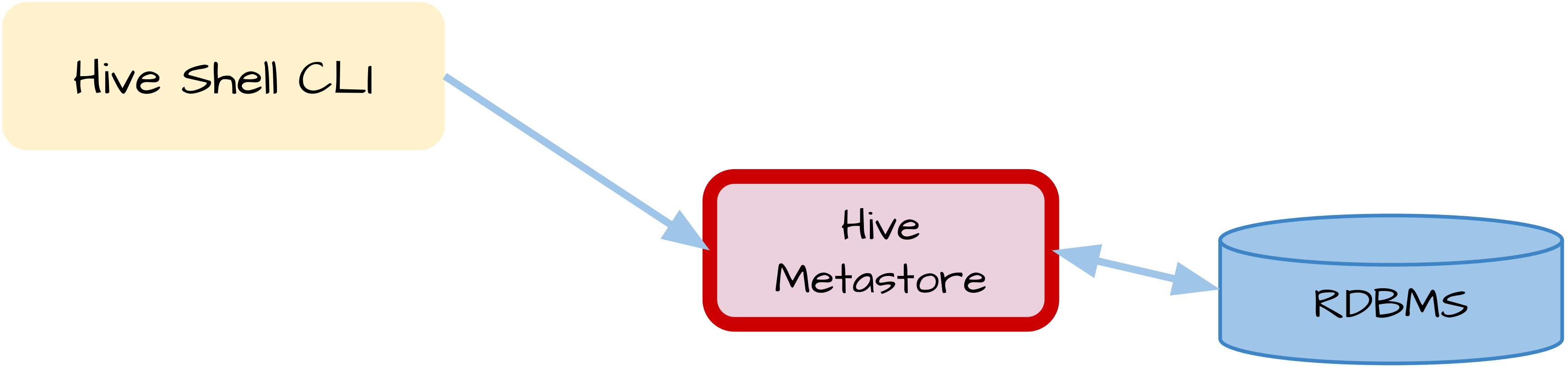
Manages metadata about databases, tables and views

Hive Metastore

Stores Hive metadata

RDBMS

Hive Shell CLI

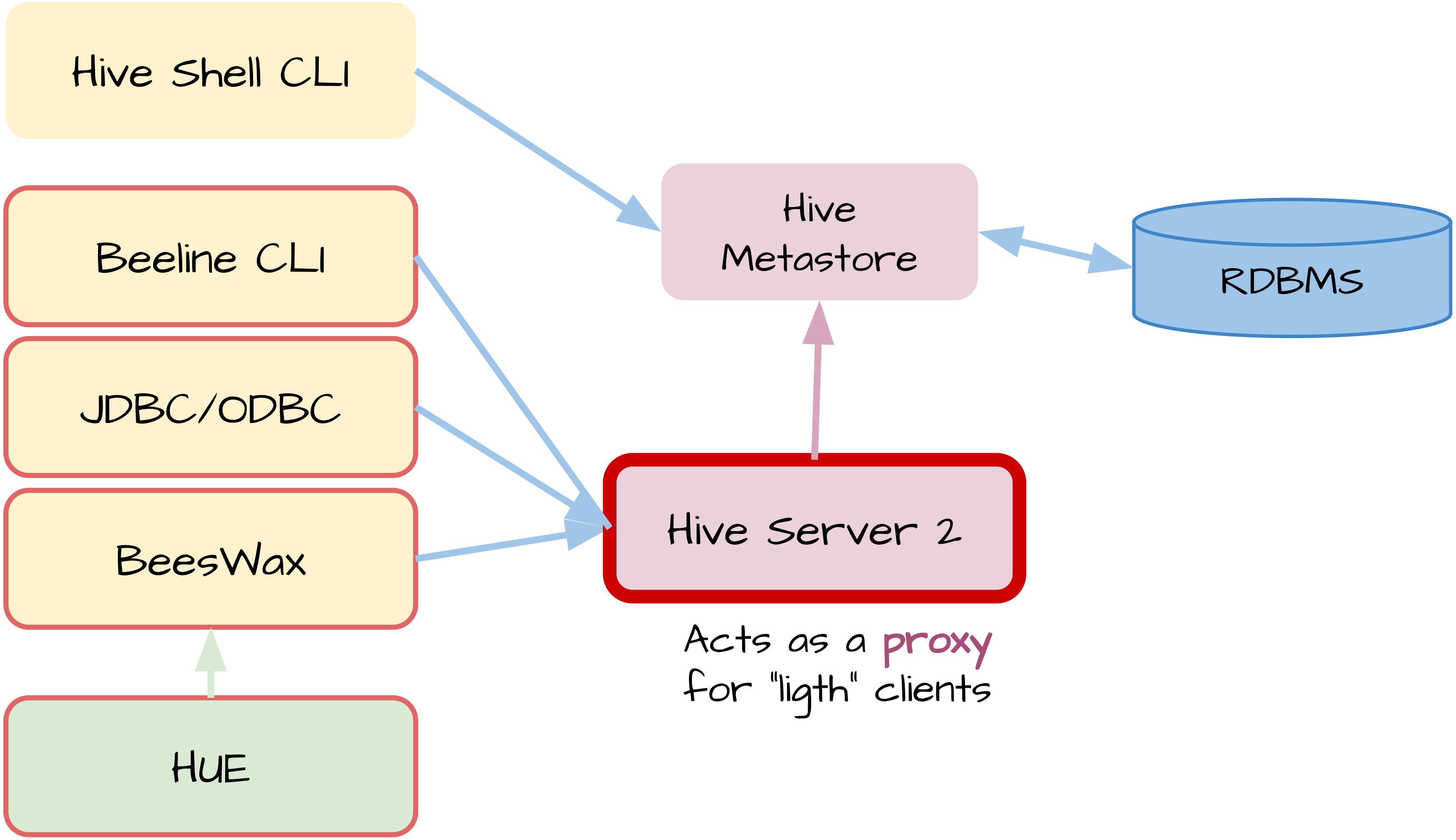


Hive Shell CLI

Hive Metastore

RDBMS

Hive Server 2



Acts as a proxy for "ligth" clients

Hive vs RDBMS

	Hive	RDBMS
Scale	Petabytes	Terabytes
Execution Engine	MapReduce, Tez, Spark	Developed during last decades
Latency	High	Low
Indexes	Limited support	Supported
Transactions	Single-table	Yes
Record-level INSERT, UPDATE, DELETE	Experimental	Yes

Exercise

Interacting With YARN Web UI

http://bit.ly/1SqouAq

Page 7