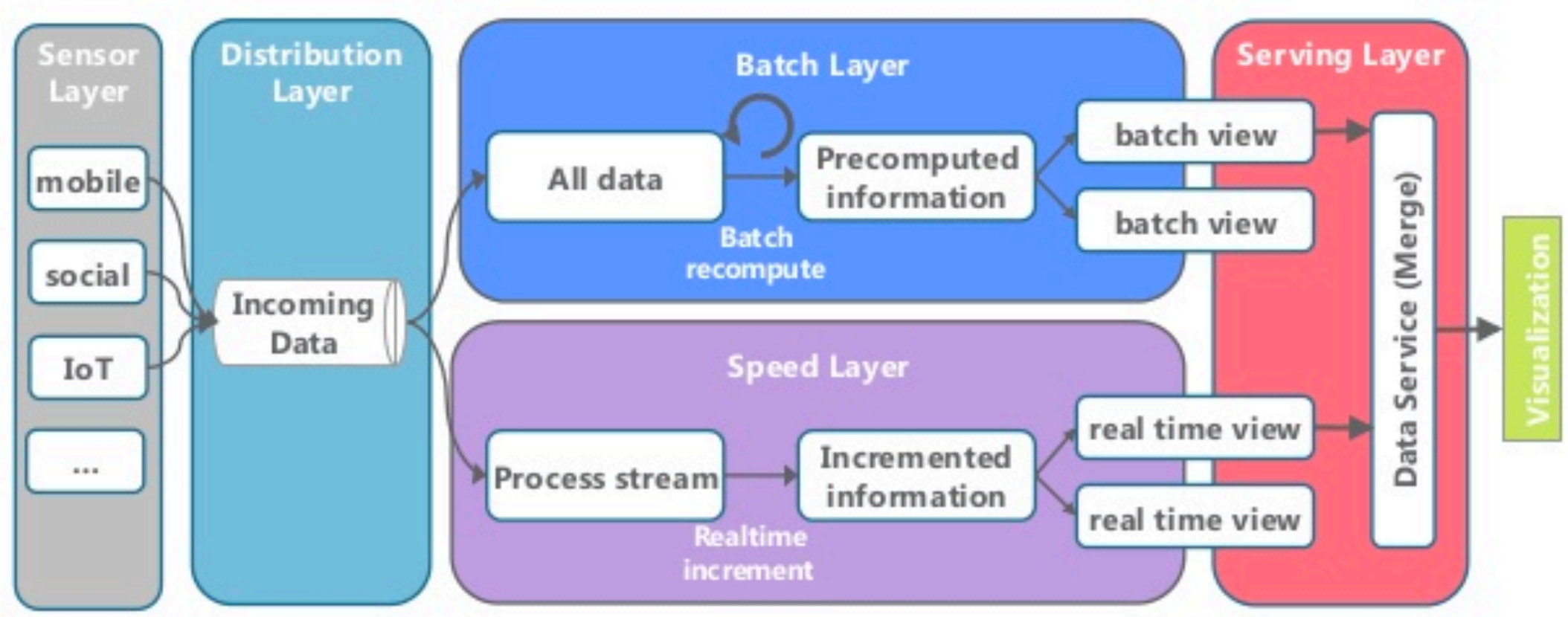
ETL of Big Data and Kafka

张春阳

张春阳
开课吧人工智能高级导师
清华大学（MEM）硕士
曾任新东方教育研究院人工智能部门部门主管 百度大数据算法工程师

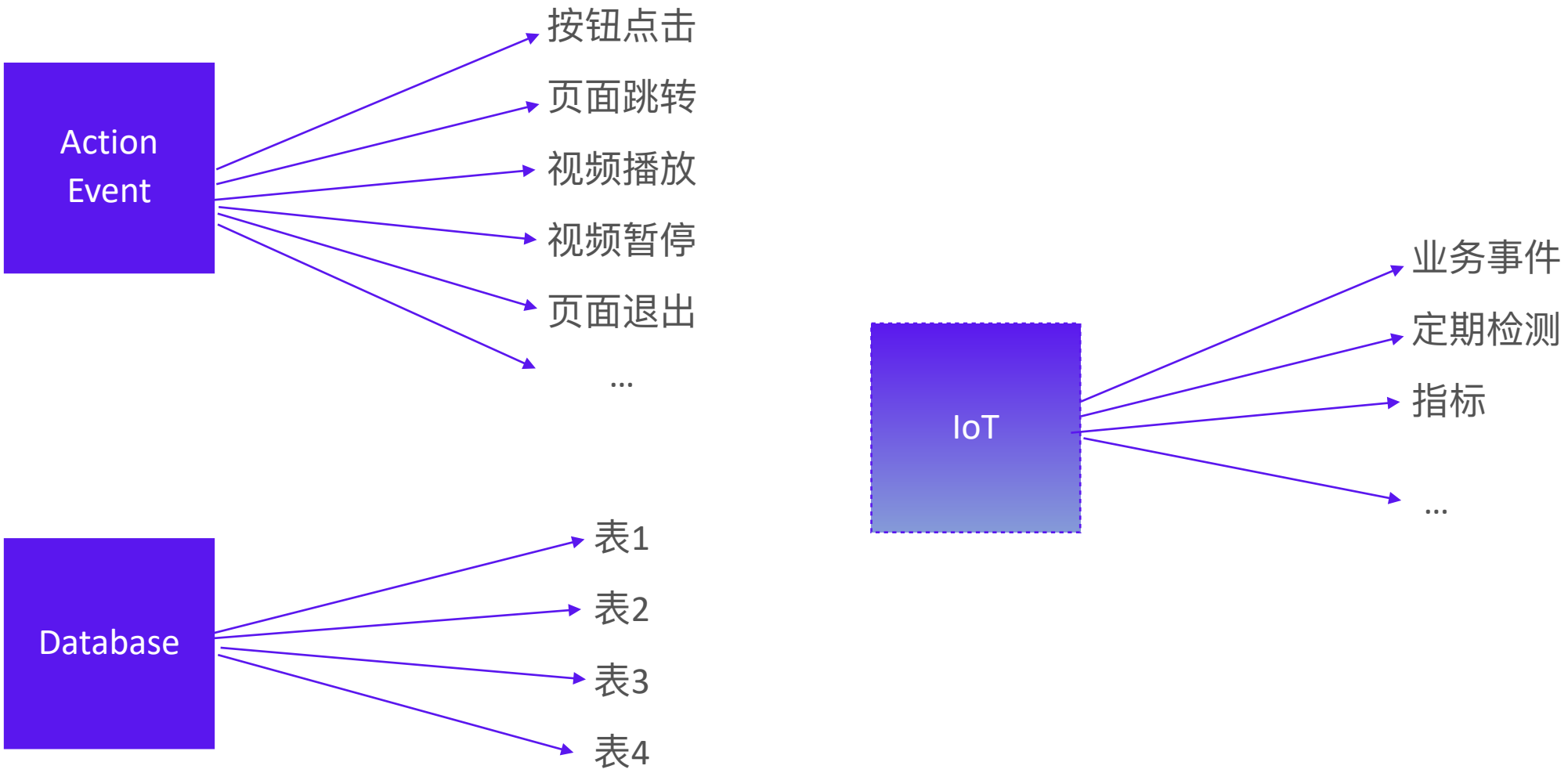
Review

Popular Lambda Architecture of Big Data

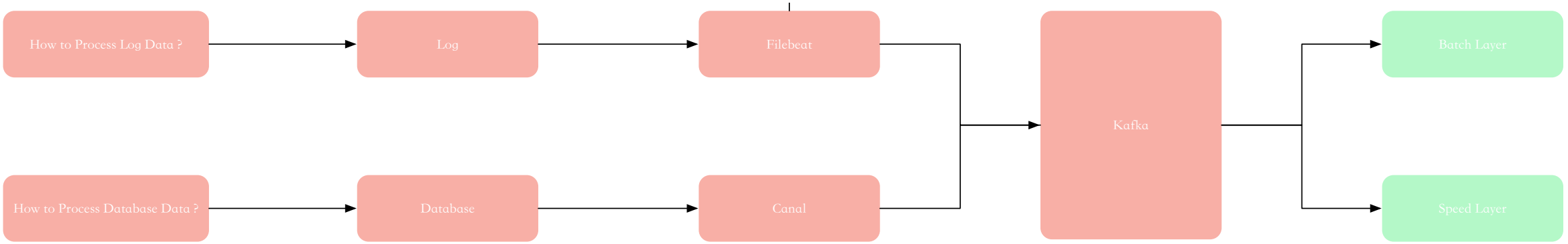


Adapted from: Marz, N. & Warren, J. (2013) Big Data. Manning.

What’s data in our System ?



Our Data Target



How to Process Database Data?

Optimize VirtualBox

headless start

sudo vi /etc/default/grub
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash 3"
sudo reboot

Keyless SSH

ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
cat ~/.ssh/id_rsa.pub append to other ~/.ssh/authorized_keys

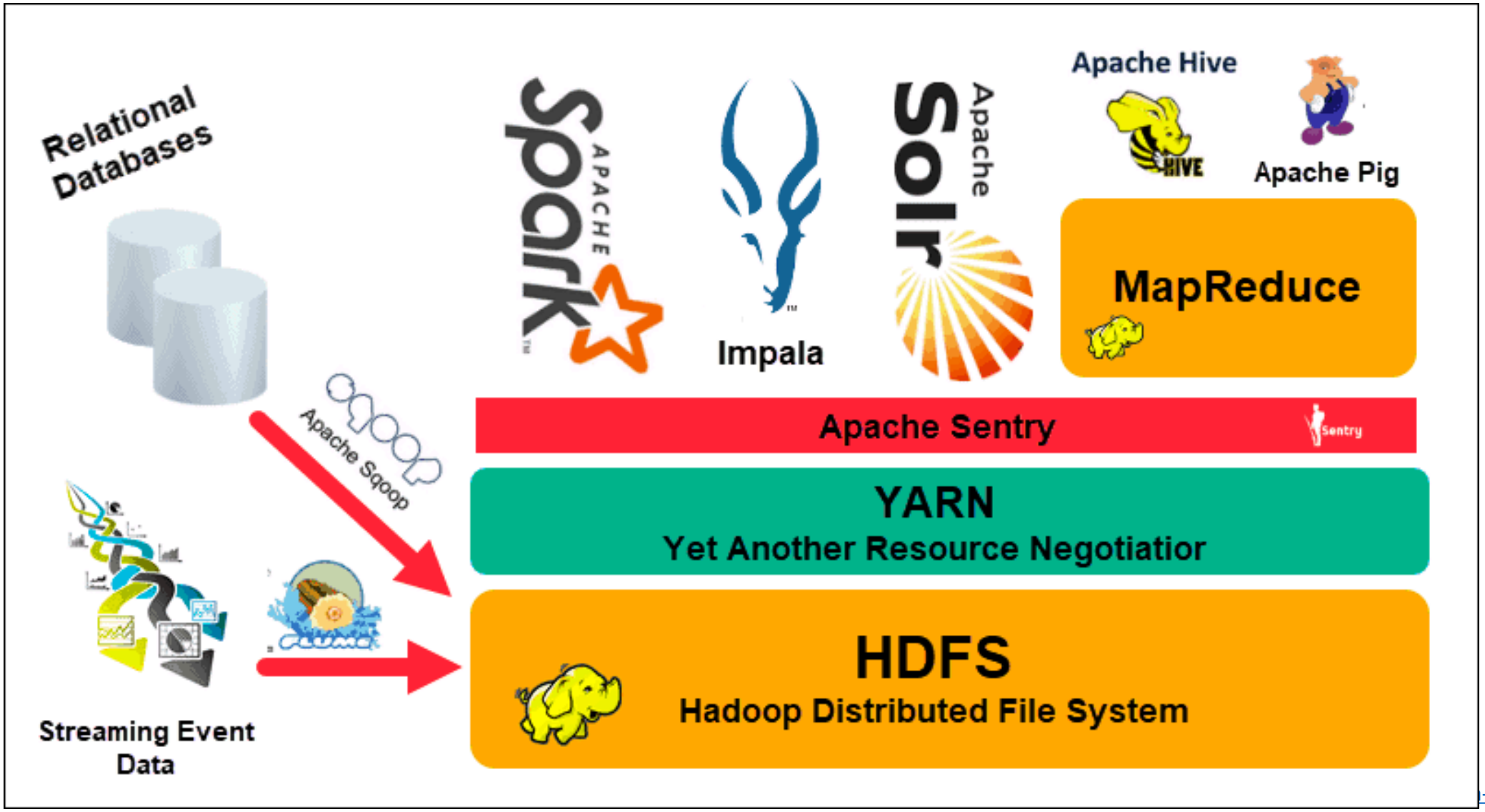
Start Service and Autostart

```
mkdir -p /opt/bigdata/services
scp services/* bigdata@bigdata-node2:/opt/bigdata/services
scp services/* bigdata@bigdata-node3:/opt/bigdata/services
bash /opt/bigdata/services/install.sh

systemctl start zookeeper.service
systemctl enable zookeeper.service
systemctl start kafka.service
systemctl enable kafka.service
systemctl start mysql.service
systemctl enable mysql.service
systemctl start kafka-manager.service(option)
```

Hadoop

Architecture of Hadoop



-RocketMQ-QuickStart

HDFS

HADOOP
DISTRIBUTED
FILE
SYSTEM

Deploy Hadoop

Hadoop Nodes

Hostname	Is Name Node	Is Data Node
bigdata-node1	Yes	Yes
bigdata-node2	No	Yes
bigdata-node3	No	Yes

Deploy Hadoop

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
ssh bigdata-node1/bigdata-node2/bigdata-node3

yum update yum
sudo apt install openjdk-8-jdk openjdk-8-jre -y
java -version

update-alternatives --display java
vim ~/.bashrc
source ~/.bashrc
hadoop version
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
export HADOOP_HOME="/opt/bigdata/hadoop"
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/bin
```

Master Node Configuration

```
sudo mkdir -p /data/hdfs/
sudo mkdir -p /data/yarn/
sudo mkdir -p /data/mr-history/
vim etc/hadoop/hadoop-env.xml
vim etc/hadoop/hdfs-site.xml
vim etc/hadoop/core-site.xml
vim etc/hadoop/yarn-site.xml
vim etc/hadoop/mapred-site.xml
sudo chown -R bigdata:bigdata /data
hdfs  namenode  -format
vim etc/hadoop/workers
mkdir /opt/bigdata/hadoop
scp -r hadoop/* bigdata@bigdata-node2:/opt/bigdata/hadoop/
scp -r hadoop/* bigdata@bigdata-node3:/opt/bigdata/hadoop/
```

Worker Node Configuration

sudo mkdir -p /data/hdfs/
sudo mkdir -p /data/yarn/
sudo mkdir -p /data/mr-history/
sudo chown -R bigdata:bigdata /data

sbin/start-all.sh
jps
http://192.168.56.101:9870/dfshealth.html#tab-datanode

HDFS Command

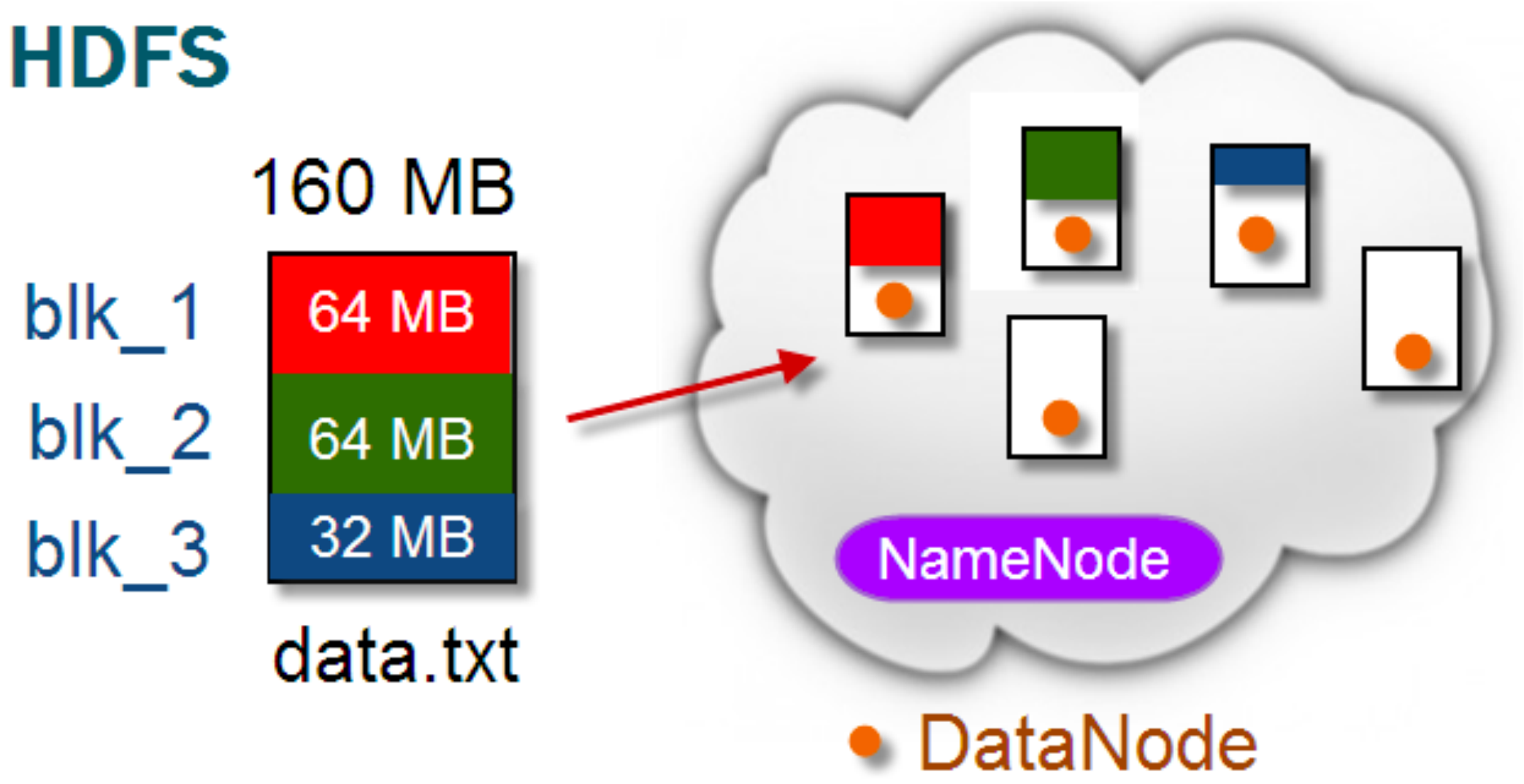
hadoop version
hadoop fs –mkdir /path/directory_name
hadoop fs –mkdir -p /path/directory_name
hadoop fs -ls /
hadoop fs -ls -R /
haoop fs -put <localsrc> <dest>
hadoop fs -get <src> <localdest>
hadoop fs -copyToLocal <hdfs source> <localdst>
hadoop fs –cat /path_to_file_in_hdfs
hadoop fs -mv <src> <dest>
hadoop fs -cp <src> <dest>

HDFS Command

hadoop fs -moveFromLocal <localsrc> <dest>
hadoop fs -moveToLocal <src> <localdest>
hadoop fs -tail [-f] <file>
hadoop fs –rm <path>
hadoop fs -expunge
hadoop fs -chown [-R] [owner] [:[group]] <path>
hadoop fs -chgrp <group> <path>
hadoop fs -setrep <rep> <path>
hadoop fs –du –s /directory/filename
hadoop fs -df [-h] <path>
hadoop fsck <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]

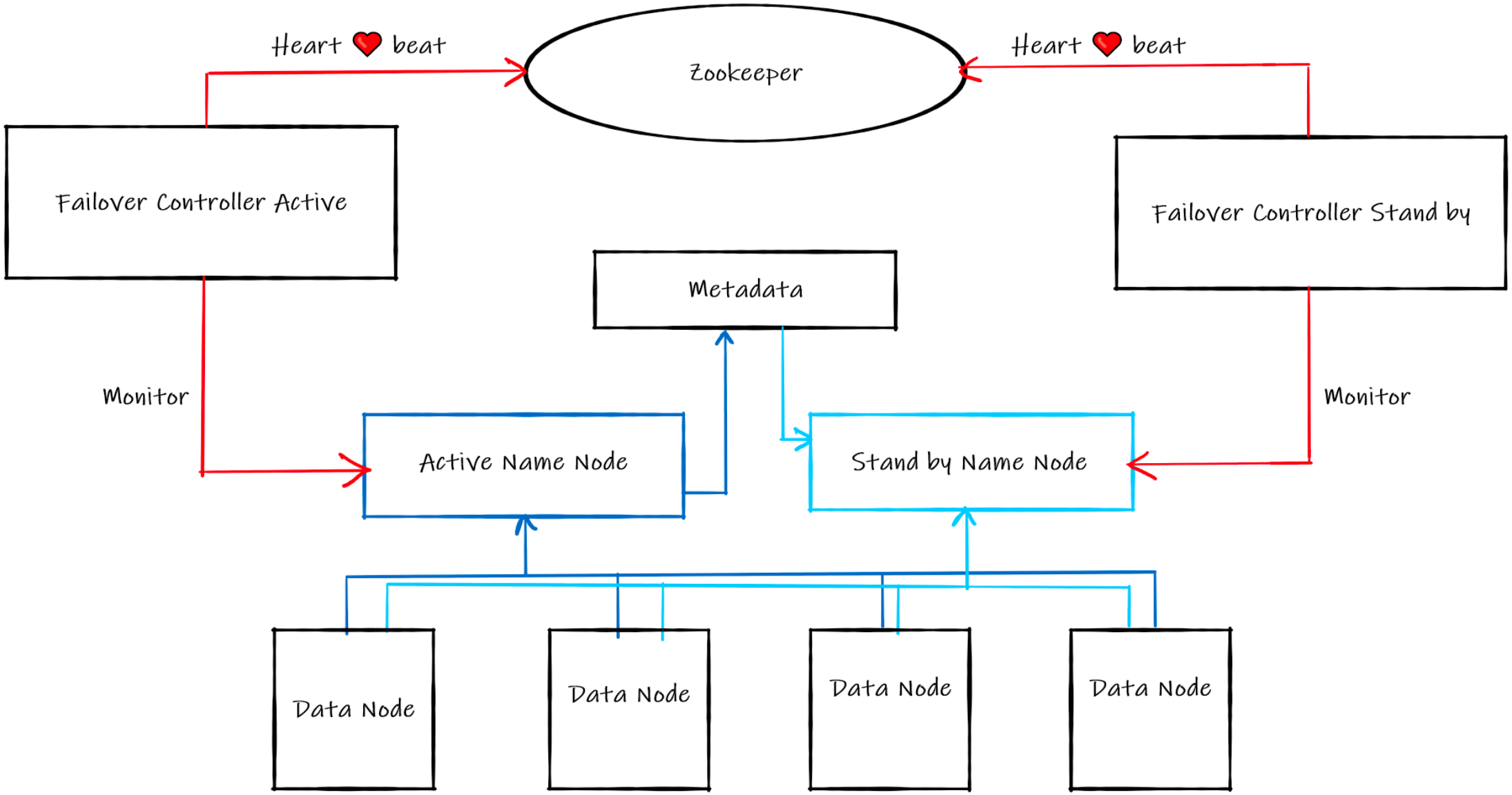
Hadoop Intro

HDFS

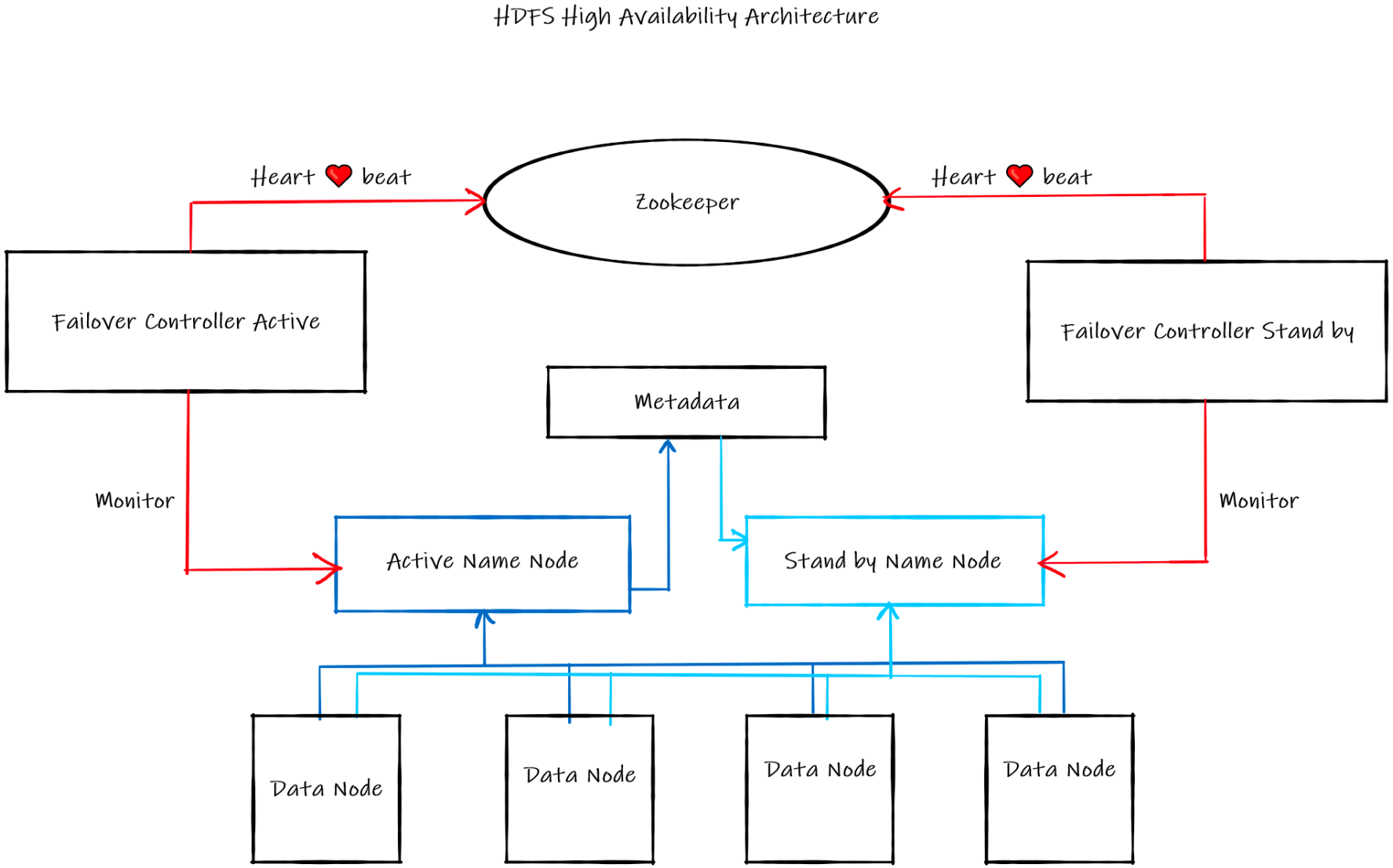

HDFS
160 MB
blk_1
blk_2
blk_3
64 MB
64 MB
32 MB
data.txt
NameNode
DataNode

HDFS High Availability Architecture

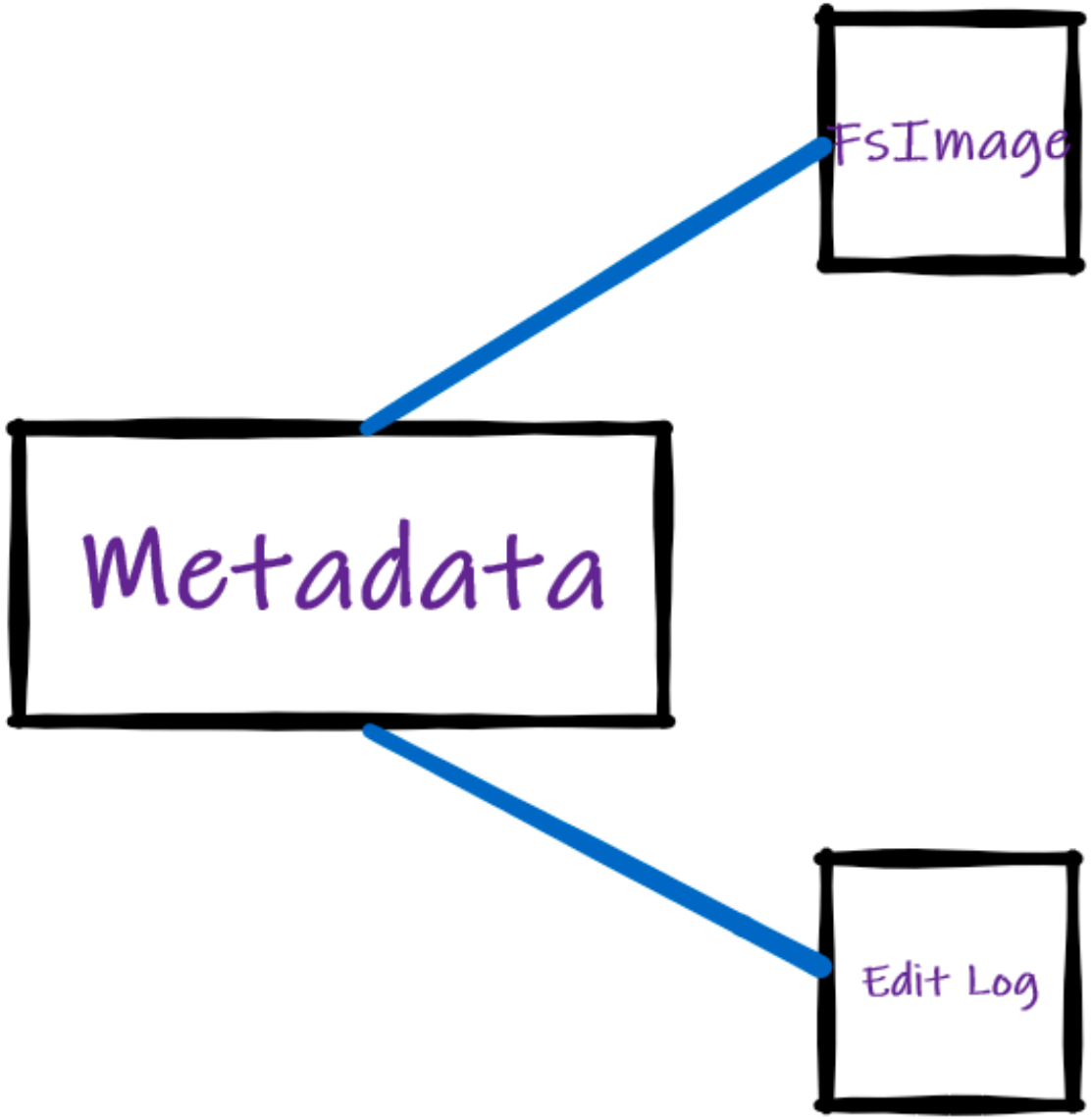
HD



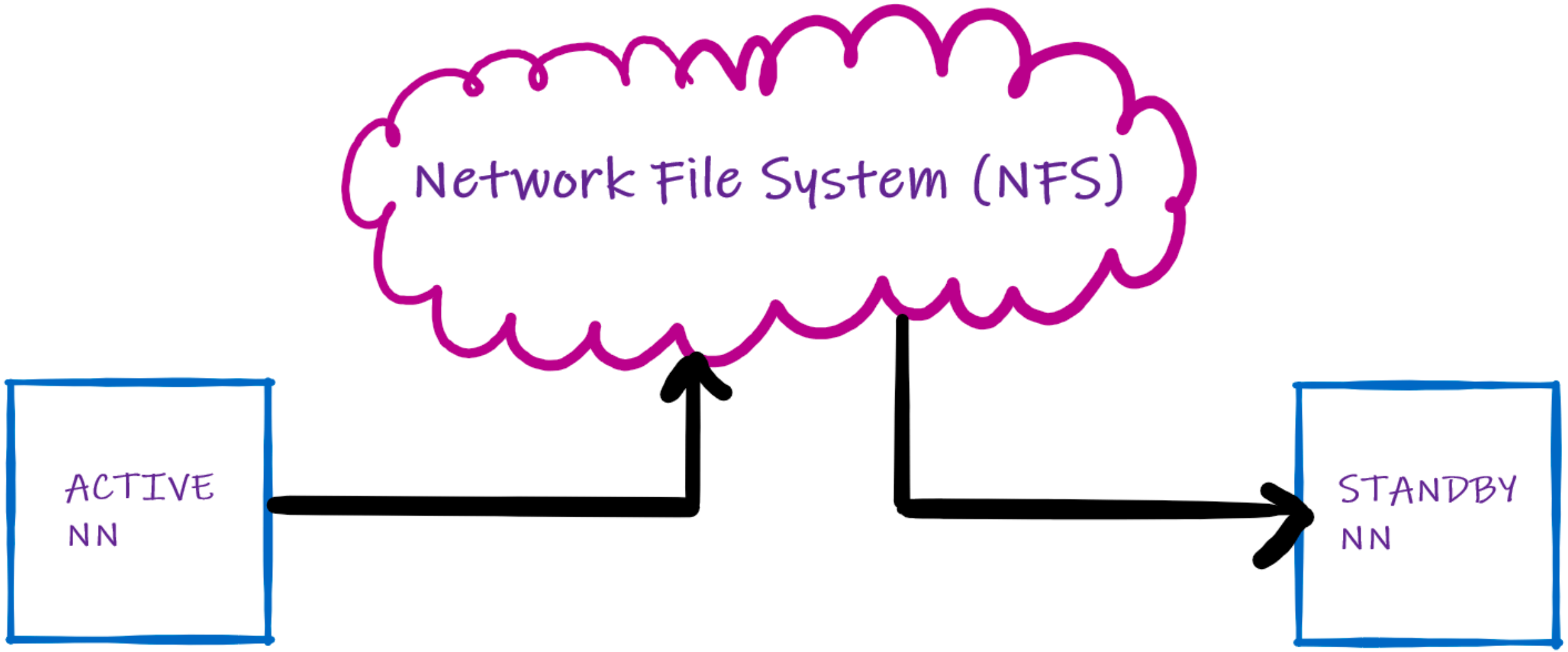
HDFS


HDFS High Availability Architecture
Heart beat
Zookeeper
Heart beat
Failover Controller Active
Failover Controller Stand by
Metadata
Monitor
Monitor
Active Name Node
Stand by Name Node
Data Node
Data Node
Data Node
Data Node

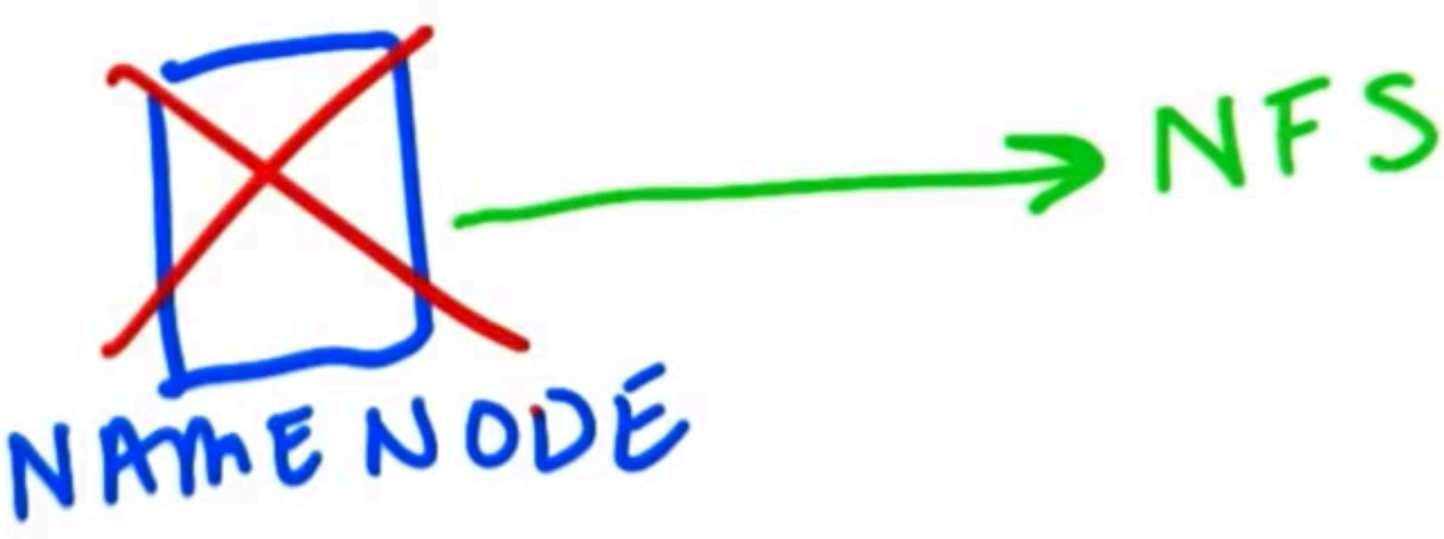
HDFS



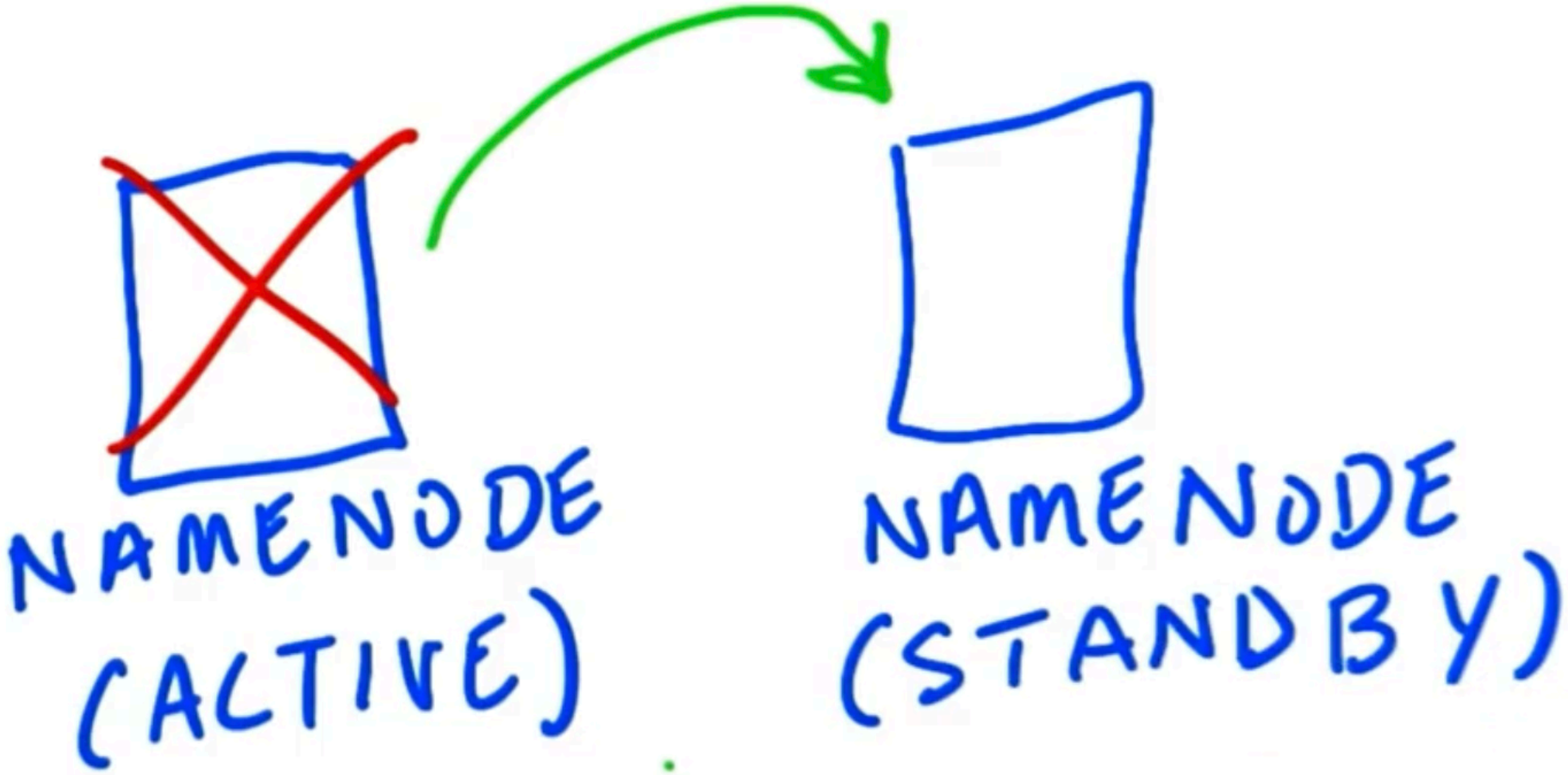
HDFS



Name Node High Availability

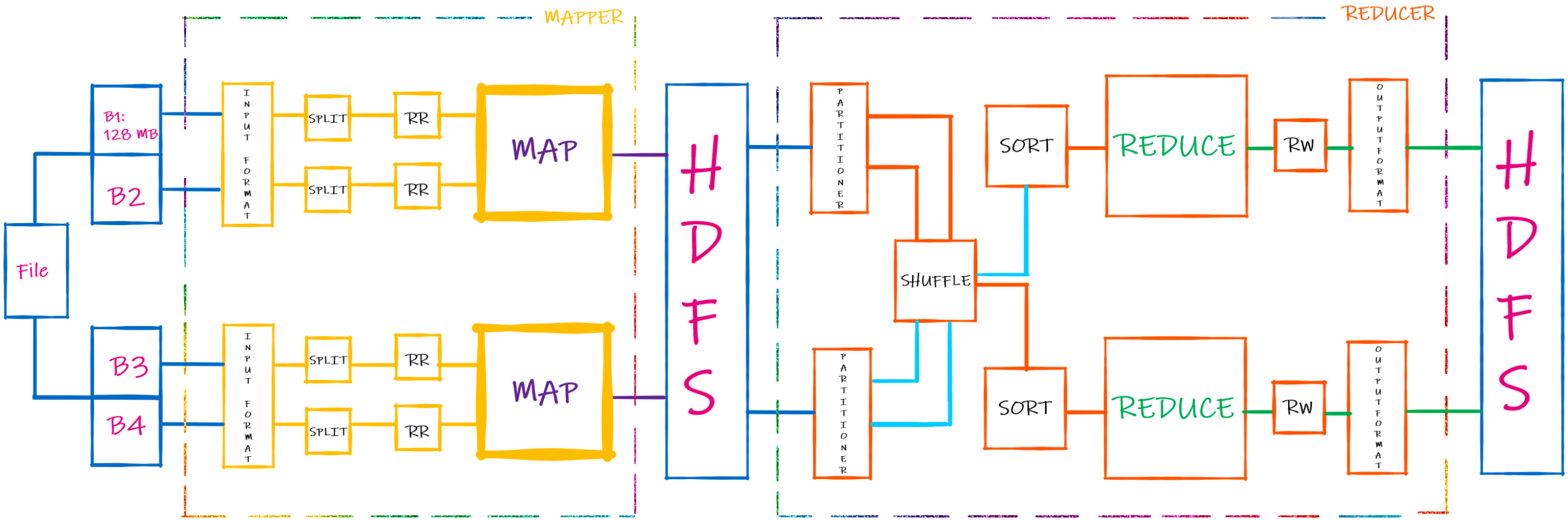


Active an Standby NameNode (Secondary NameNode)



Map Reduce

MapReduce

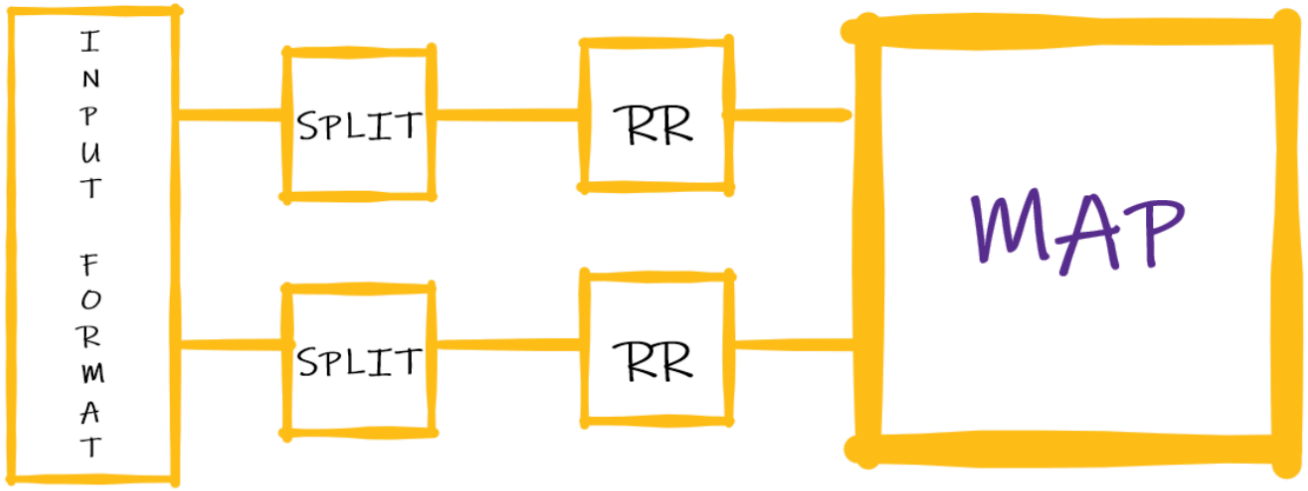

MAPPER
REDUCER
File
B1: 128 MB
B2
B3
B4
INPUT FORMAT
INPUT FORMAT
SPLIT
SPLIT
SPLIT
SPLIT
RR
RR
RR
RR
MAP
MAP
HDFS
PARTITIONER
PARTITIONER
SHUFFLE
SORT
SORT
REDUCE
REDUCE
RW
RW
OUTPUTFORMAT
OUTPUTFORMAT
HDFS

MapReduce

```
map(in_key, in_value) -> list(intermediate_key, intermediate_value)
```

1. **Input Format —** It is a pointer to HDFS' file block location. The data is still not loaded into memory and currently sits as-is on HDFS itself. In our case the pointer in node 1's Input Format points to Block 1 and 2 similarly, in Node 2, it points to Block 3 and 4.
2. **Split —** At this point, the file is actually loaded into memory. The number of splits equals the number of blocks in that node. The split and the RecordReader work together.
3. **RR or RecordReader —** I'm sure you wondered how to turn a simple file into a (k,v) pair. Well, Google mentioned in their white paper, most of the processing is done via abstraction, which is great! The record reader simply does this processing of the data into (k,v) pair for us. What's more, there are multiple ways to achieve that. More details below.
4. **Map —** Finally, we arrive at the "map" function, wherein the actual processing happens. Whatever logic you'd like the function to perform, here is where it all happens. Post this, the resultant (k,v) pair is finally offloaded into HDFS again and the reducer task is kick-started.

Job Started: Map 0% Reduce 0%

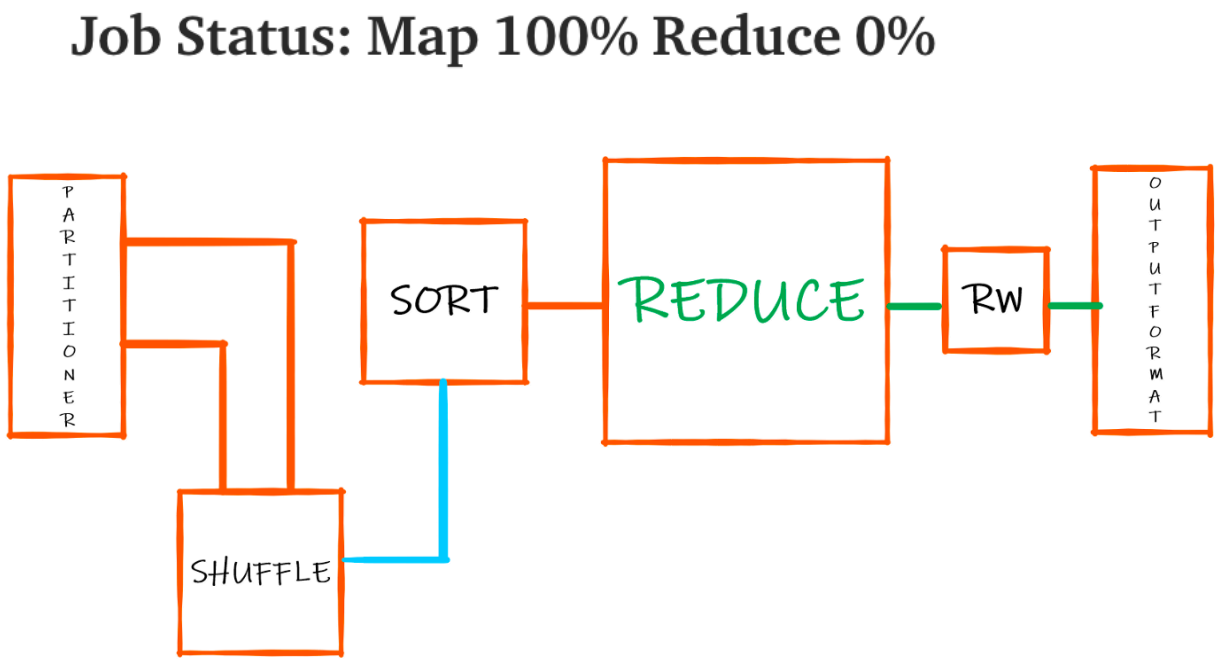


Mapper

MapReduce

```
reduce(intermediate_key, list(intermediate_value) -> list(out_key,
out_value)
```

1. **Partitioner —** The intermediate (k,v) pair is again loaded into memory, as-is. And using these intermediate keys, a *grouping function* is applied to the dataset. We'll understand this better in the example in the next section. If you're feeling lucky — here's a custom partitioner.
2. **Shuffle —** This is grouping across nodes. Basically, common keys are now "shuffled" across nodes.
3. **Sort —** The data is now sorted based on keys.
4. **Reduce —** Finally, we arrive at the "reduce" function, wherein the actual aggregation of data happens. Whatever aggregation you'd like the function to perform, here is where it all happens.
5. **Output Format —** The resultant (k,v) pair is finally offloaded into HDFS again via the RecordWriter (RW — more details below) and the job is completed.



Job Status: Map 100% Reduce 100%

A Simple Example

Imagine you're Elon Musk, the CEO of Tesla Motors. You've got the following data on current global sales of Tesla cars (in millions) over the year. Naturally, you're happy, you smoked some and immediately went on Twitter:

> Tesla stock price is too high imo

Okay, that was a mistake. Then you weep or not. Anyway, I digress.

Tesla Car Sales Dataset (in Millions)

Map

```
Country,Sales(M)
USA,1
Russia,1
UK,1
France,1
China,1
Russia,1
UK,1
France,1
China,1
USA,1
```

```
Country,Sales(M)
UK,1
USA,1
China,1
UK,1
USA,1
China,1
UK,1
USA,1
China,1
UK,1
```

Reduce

Country,Sales(M): Partition
USA,1
USA,1
Russia,1
Russia,1
UK,1
UK,1
France,1
France,1
China,1
China,1

Country,Sales(M): Partition
UK,1
UK,1
UK,1
UK,1
USA,1
USA,1
USA,1
China,1
China,1
China,1

Country,Sales(M): Shuffle
Russia,1
Russia,1
France,1
France,1

Country,Sales(M): Shuffle
USA,1
USA,1
USA,1
USA,1
USA,1
UK,1
UK,1
UK,1
UK,1
UK,1
UK,1
China,1
China,1
China,1
China,1
China,1

Reduce

```
Country,Sales(M): Sort
France,1
France,1
Russia,1
Russia,1
```

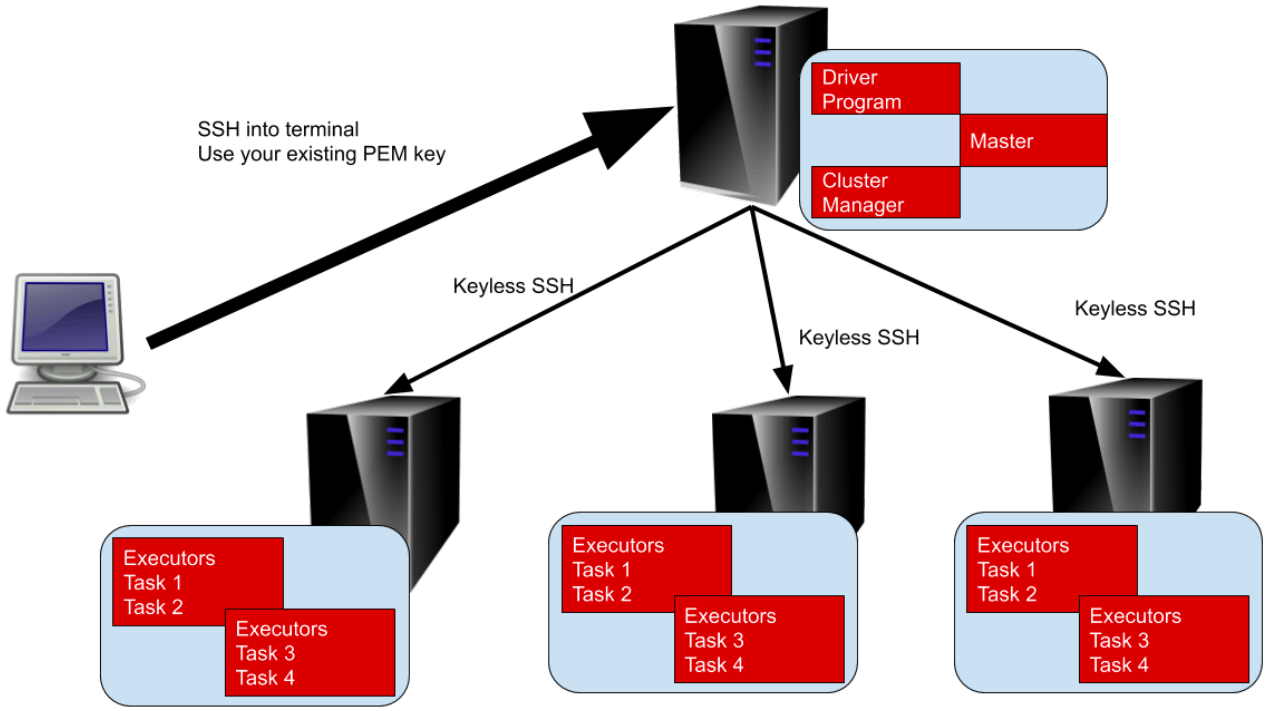
```
Country,Sales(M): Sort
China,1
China,1
China,1
China,1
China,1
USA,1
USA,1
USA,1
USA,1
USA,1
UK,1
UK,1
UK,1
UK,1
UK,1
UK,1
```

```
Country,Sales(M): Reduce
France,2
Russia,2
```

```
Country,Sales(M): Reduce
China,5
USA,5
UK,6
```

Spark

Hadoop Nodes



Hostname	Is Master Node	Is Worker Node
bigdata-node1	Yes	Yes
bigdata-node2	No	Yes
bigdata-node3	No	Yes

Deploy HBase

export HBASE_HOME=/opt/bigdata/hbase
export PATH=$HBASE_HOME/bin:$PATH

vim ~/.bashrc
source ~/.bashrc

sudo mkdir -p /data/hbase/tmp
sudo mkdir -p /data/hbase/hfiles
sudo mkdir -p /data/zookeeper
sudo chown -R bigdata:bigdata /data/

vim /opt/bigdata/hbase/conf/hbase-env.sh
source conf/hbase-env.sh
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
vim /opt/bigdata/hbase/conf/hbase-site.xml
vi regionservers
mkdir /opt/bigdata/hbase
scp -r hbase/* bigdata@bigdata-node2:/opt/bigdata/hbase/
scp -r hbase/* bigdata@bigdata-node3:/opt/bigdata/hbase/
bin/start-hbase.sh

http://192.168.56.101:16010/master-status

Assignment

`Setup Hadoop environment and implement`

1、减分项：抄袭作业；原封不动提交老师代码；作业提交超时。
2、加分项：代码有注释；代码运行结果正确；代码格式规范；在作业的基础上有对比实验或者自己总结的项目心得。
3、满分作业要求：
a.提交了非抄袭代码。
b.符合讲师的作业目标要求。
c.代码有合理的注释。
d.代码书写整洁规范、代码逻辑严谨。
e.有模型实验结果的对比总结或者从项目中总结的心得体会。
f.代码能够运行成功。