Quick Refresher

Recap:

Which Python library is commonly used for data manipulation and analysis?

A: NumPy
B: Django
C: Pandas
D: Matplotlib

```
import pandas as pd
```

Recap:

You have a "Date" column as strings. How do you convert it to a proper datetime format?

A: df['Date'] = pd.to_datetime(df['Date'])

B: df['Date'] = df['Date'].astype('datetime')

C: df['Date'] = convert_to_datetime(df['Date'])

D: df['Date'] = df['Date'].apply(datetime.strptime)

Recap:

How do you change the data type of a column from string to integer?

A: df['column'] = df['column'].to_int()

B: df['column'] = df['column'].convert_dtypes()

C: df['column'] = df['column'].astype('int64')

D: df['column'] = cast(df['column'], 'integer')

Recap:

You have a column with addresses like " 123 Main Street, New York ".
Which code snippet removes excess whitespace around the address?

A: df['Address'] = df['Address'].split()

B: df['Address'] = df['Address'].strip()

C: df['Address'] = df['Address'].remove(' ')

D: df['Address'] = df['Address'].clean()

Recap:

You need to extract the city name from addresses formatted like "123 Main Street, New York, NY". How could you achieve this?

A: df['City'] = df['Address'].split(',')[1]

B: df['City'] = df['Address'].split(' ')[-1]

C: df['City'] = df['Address'].strip().split(',')[1]

D: df['City'] = df['Address'].extract(', (.*),')

Recap:

Which of the following is a prerequisite for setting up a Hadoop cluster?

A: Python

B: Java

C: .NET Framework

D: Ruby

Recap:

You need to download the Hadoop distribution. Which command would you use?

A: curl

B: wget

C: scp

D: git clone https://github.com/apache/hadoop

Recap:

What is the purpose of the hadoop_env.sh file within a Hadoop installation?

A: Stores data blocks for HDFS

B: Contains the primary user interface for the cluster

C: Sets environment variables and customizations for Hadoop components

D: Manages the scheduling of Hadoop jobs

Recap:

What is the purpose of the hadoop_env.sh file within a Hadoop installation?

A: Stores data blocks for HDFS

B: Contains the primary user interface for the cluster

C: Sets environment variables and customizations for Hadoop components

D: Manages the scheduling of Hadoop jobs

Velocity == Mis
Missed windows

Veracity == In
Insights can be w

Volume == Sto
Limit on what ca

Variety == Co
Analysis become

Value == Wast
Leaving potential

RECAP

Big Data offers huge potential, BUT it requires the right skills and tools to harness

HADOOP!

hadoop

How does Hadoop help handle Big Data?

Designed for Massive Scalability

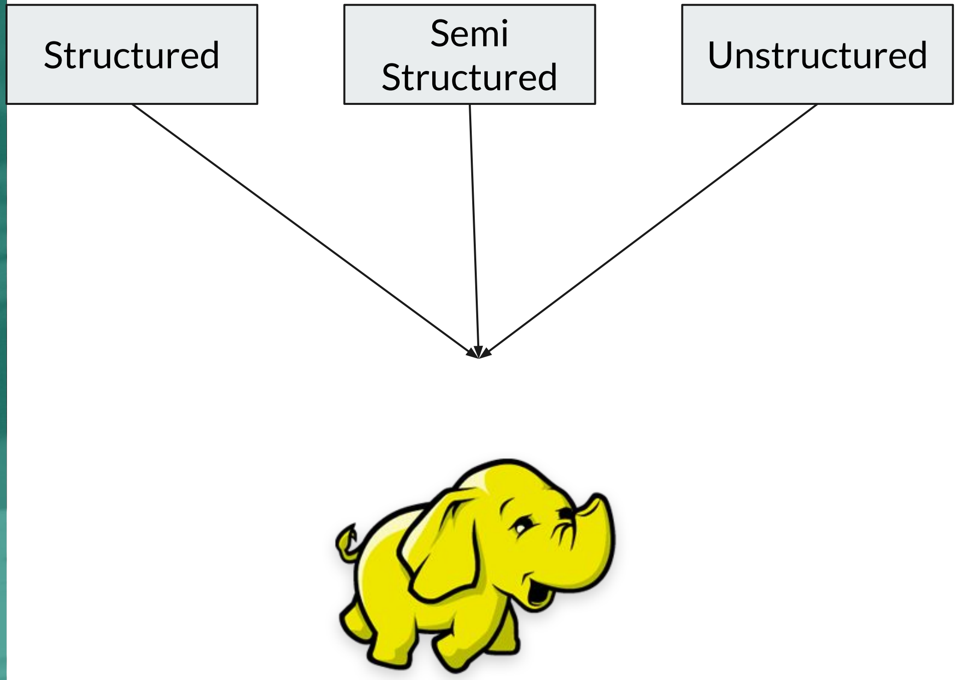
How does Hadoop help handle Big Data?

Designed for Massive Scalability

- Traditional databases and tools reach a storage limit.
- **Hadoop scales horizontally**: Add more commodity machines to increase capacity.
- Designed to handle petabytes and even exabytes of data.

How does Hadoop help handle Big Data?

Handles Diverse Data



How does Hadoop help handle Big Data?

Handles Diverse Data

- From Structure to Chaos, Hadoop Handles It All
- Hadoop doesn't require everything to be pre-organized for analysis.
- Analysis can happen on the raw data.

How does Hadoop help handle Big Data?

Real-Time Insights

How does Hadoop help handle Big Data?

Real-Time Insights

- Hadoop is designed for streaming data.
- This means insights as events happen, for immediate action.

How does Hadoop help handle Big Data?

Fault-Tolerant Big Data

What is “Fault Tolerance”?

How does Hadoop help handle Big Data?

Fault-Tolerant Big Data

What is “Fault Tolerance”?

- **The Redundant Alarm Clock:** Multiple alarms in case one doesn't go off.
- **Backup Route Planning:** If one road has unexpected traffic, switch to a different route.

How does Hadoop help handle Big Data?

Fault-Tolerant Big Data

- Hardware WILL fail, that's a given.
- Hadoop stores data redundantly across nodes.
- Computations are designed to continue even with failures.

What did we learn about Hadoop?

Recap: How Hadoop helps?

- Designed for Massive Scalability: Store and process truly huge and growing datasets.
- Handles Diverse Data: Structured, semi-structured, unstructured...
- Real-time Insights: Built for fast, continuous processing.
- Reliability/Fault Tolerance: Data is protected, and computations survive machine failures.

Origins of Hadoop

Origins of Hadoop

From Web Search to Big Data Backbone

Origins of Hadoop

From Web Search to Big Data Backbone

- **Inspiration**: Needed to index the entire web at massive scale.
    - Distributed Storage
    - Parallel Processing

Origins of Hadoop

From Web Search to Big Data

e entire

Original Research from Google that inspired Hadoop

-

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

d on

me

Origins of Hadoop

From Web Search to Big Data Backbone

- **Inspiration:** Needed to index the entire

d on
ame

la’s work
icture.

Doug Cutting & Mike Cafarella, co-created the Hadoop project.

Origins of Hadoop

From Web Search to Big Data Backbone

- **Inspiration:** Needed to index the entire

d on
ame

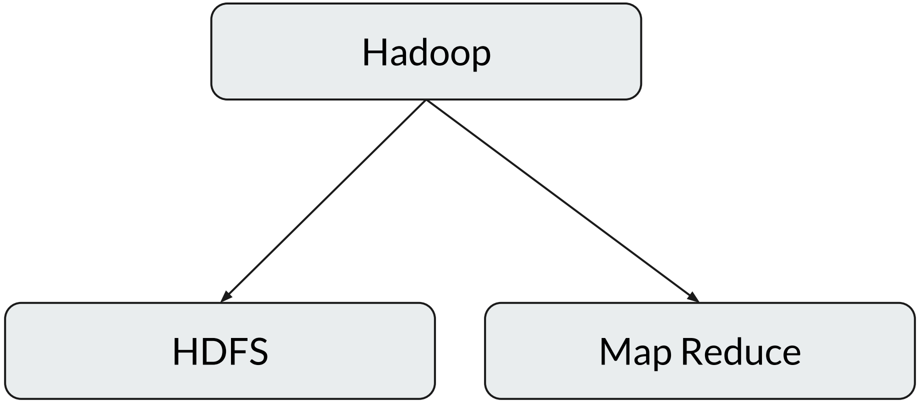
la's work
icture.

Doug Cutting serves as Chief Architect of Cloudera - one of the major Hadoop distributors.

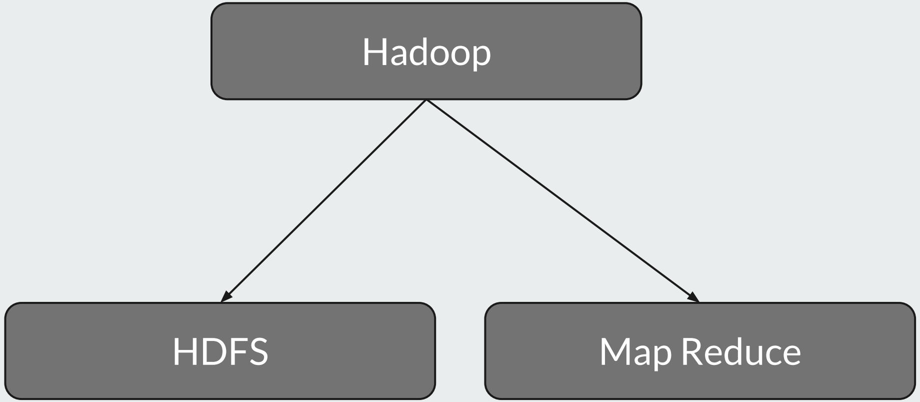
Core components of Hadoop

Core components of Hadoop

Hadoop Fundamentals: Storage & Processing

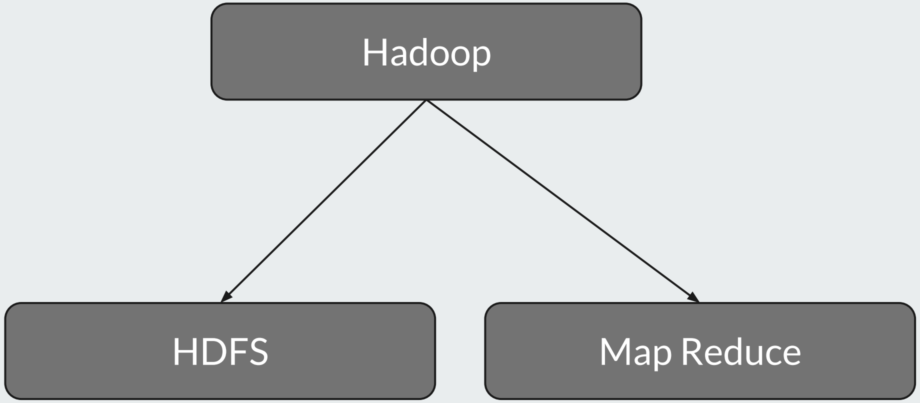


Hadoop Fundamentals: Storage & Processing



- **Storage** → **HDFS**: Distributed filesystem for reliable, scalable storage.

Hadoop Fundamentals: Storage & Processing



- **Storage** → **HDFS**: Distributed filesystem for reliable, scalable storage.
- **Processing** → **MapReduce**: Programming model for processing large datasets in parallel.

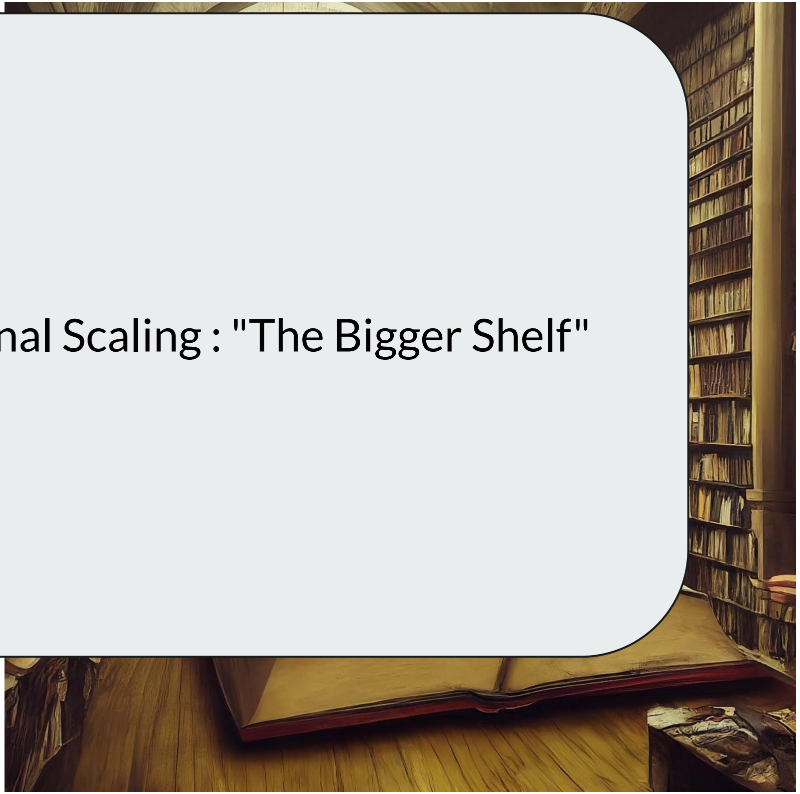
HDFS

HDFS
The 'Big File' Solution

Imagine a book so enormous, it wouldn't fit on ANY bookshelf. What do we do?

HDFS
The 'Big File' Solution

How about building a bigger shelf?

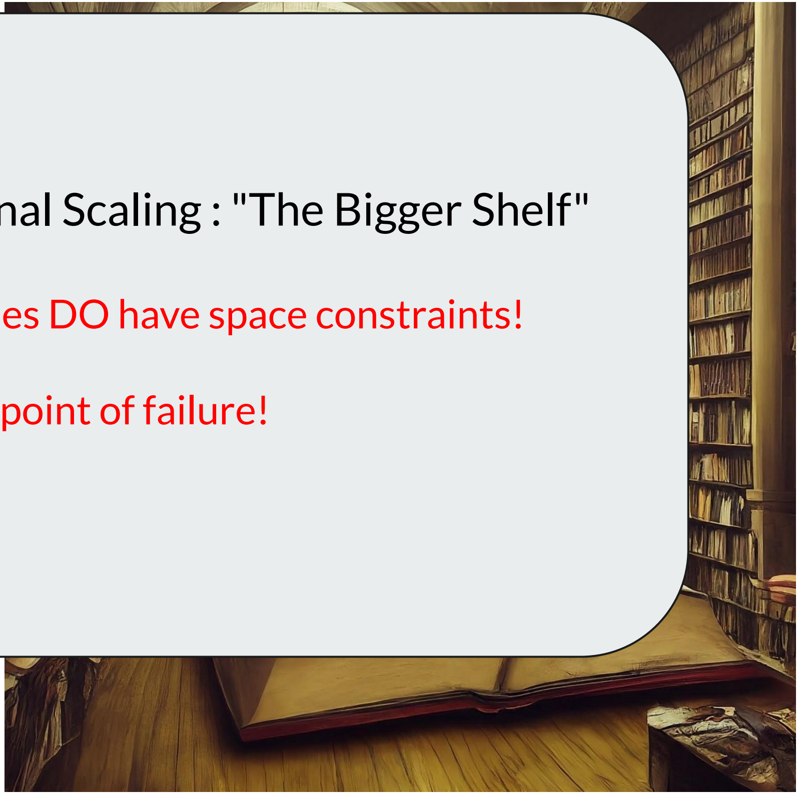


Traditional Scaling : "The Bigger Shelf"

HDFS

The 'Big File' Solution

How about building a bigger shelf?

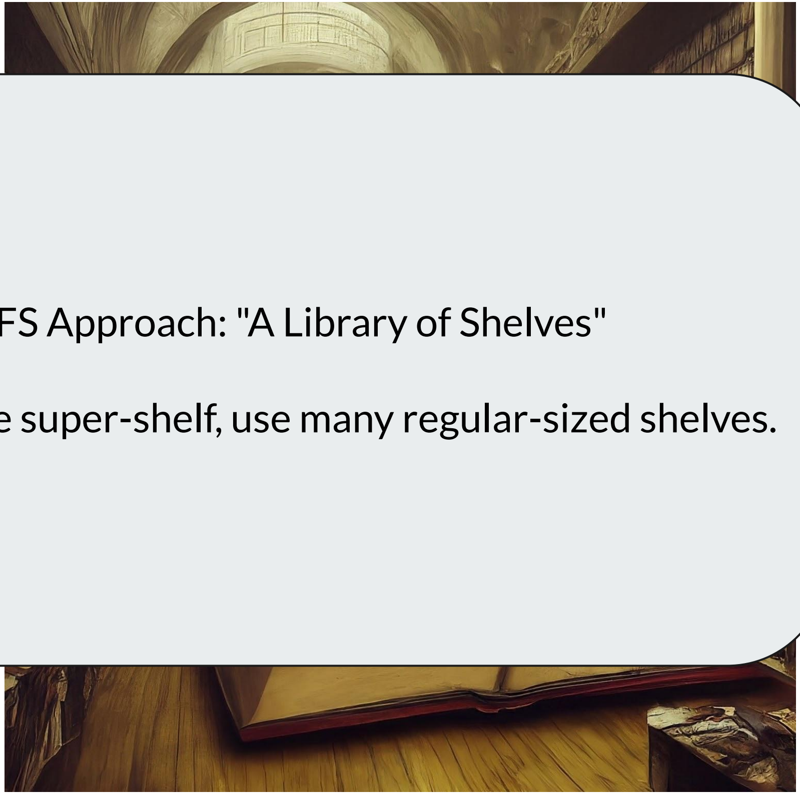


Traditional Scaling : "The Bigger Shelf"

- Libraries DO have space constraints!
- Single point of failure!

HDFS
The 'Big File' Solution

How about splitting the book and putting each chapter on a different shelf in a huge library?



HDFS Approach: "A Library of Shelves"

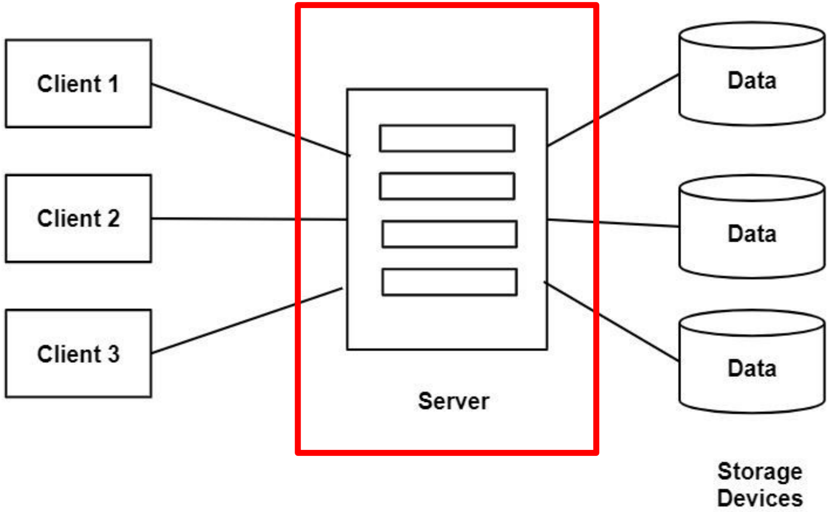
Instead of one super-shelf, use many regular-sized shelves.

HDFS
The 'Big File' Solution

Formally -

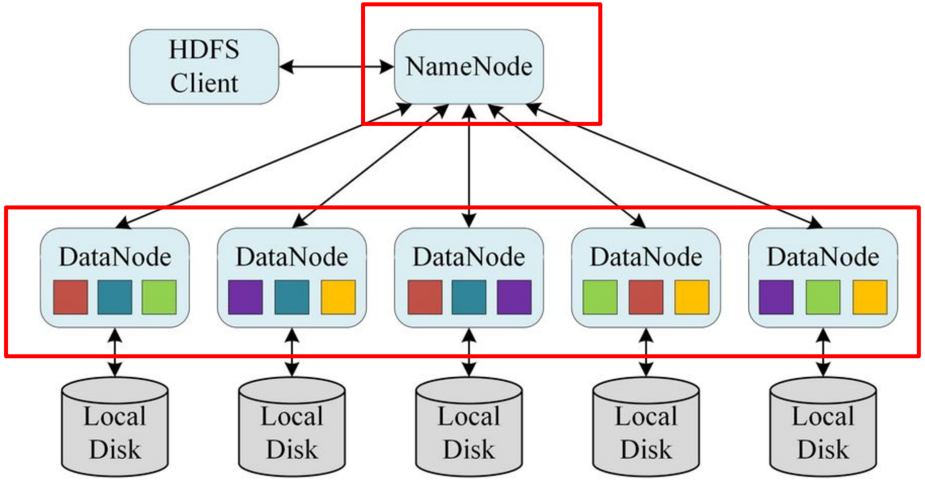
- **Designed for Scale:** Stores very large files across many machines (forming a **CLUSTER**).
- **Resilience Built-In**: Data is replicated across nodes (3x replication is common) for protection against failure.
- **Optimized for Big Files**: Works best with huge datasets; not a lot of smaller ones.

HDFS Architecture

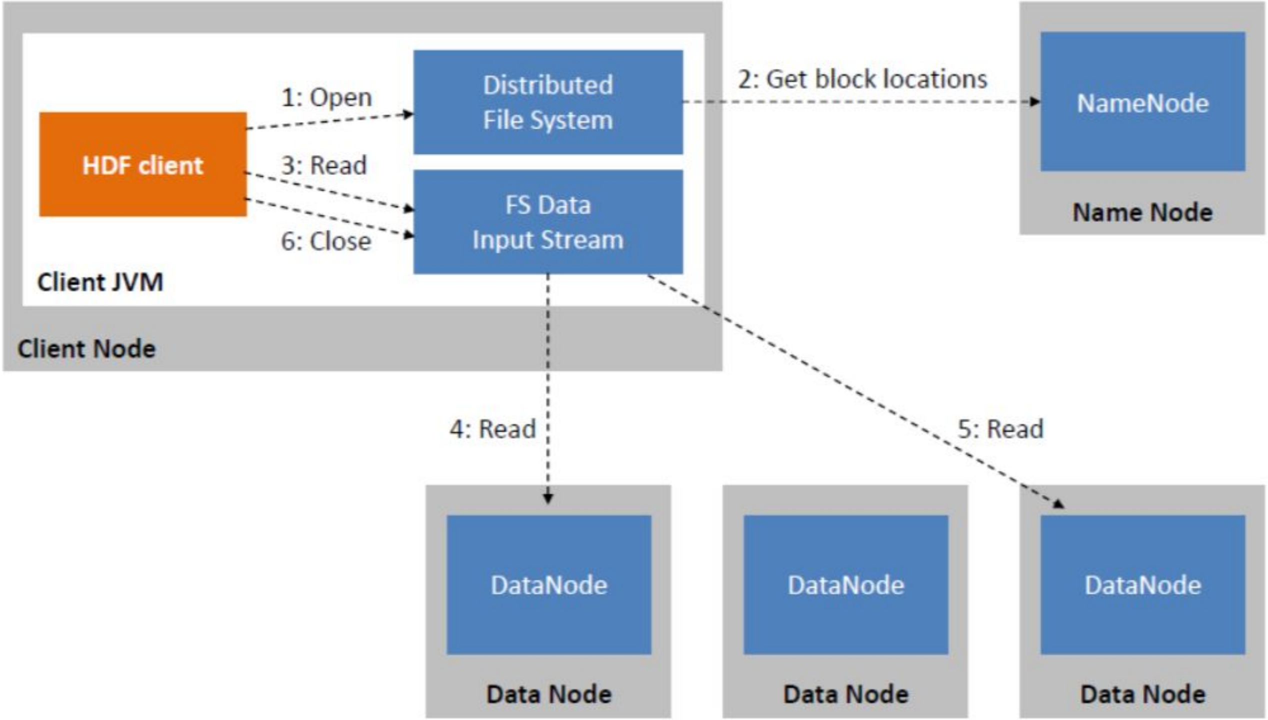


HDFS Architecture

Master - Slave Design



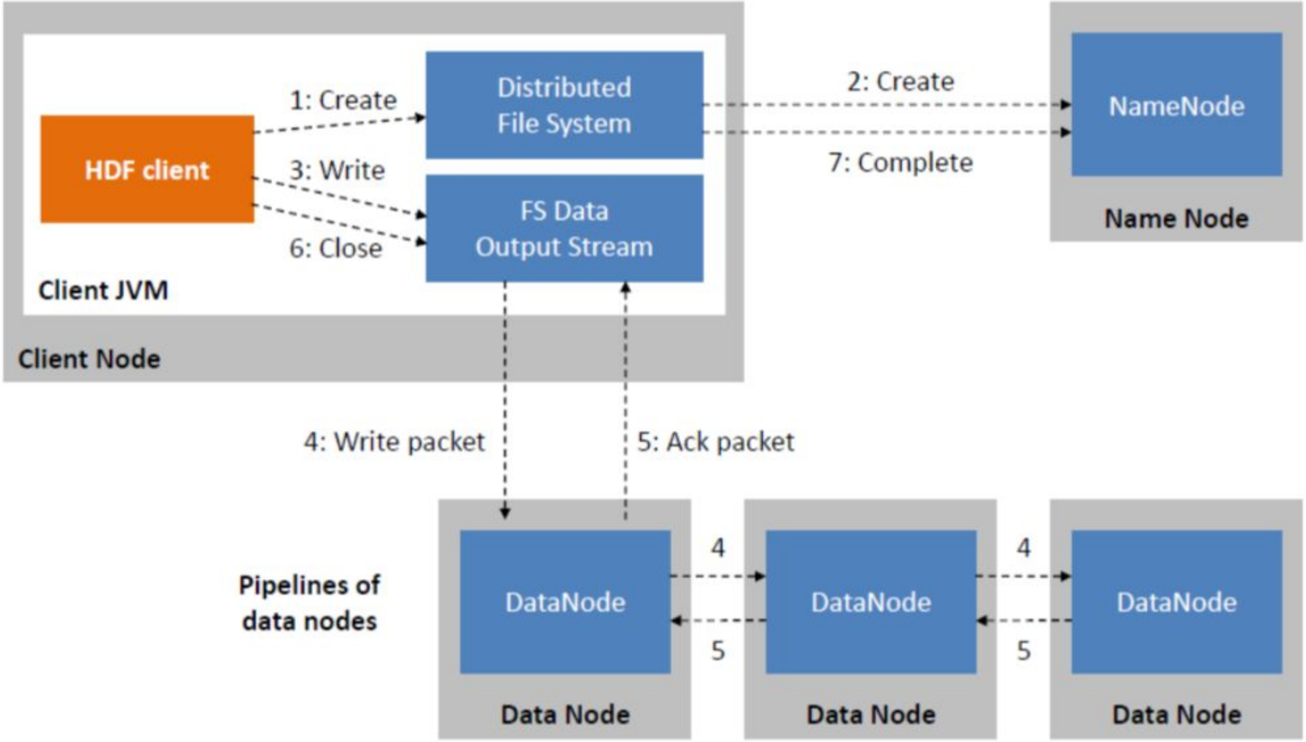
Read Operation in HDFS



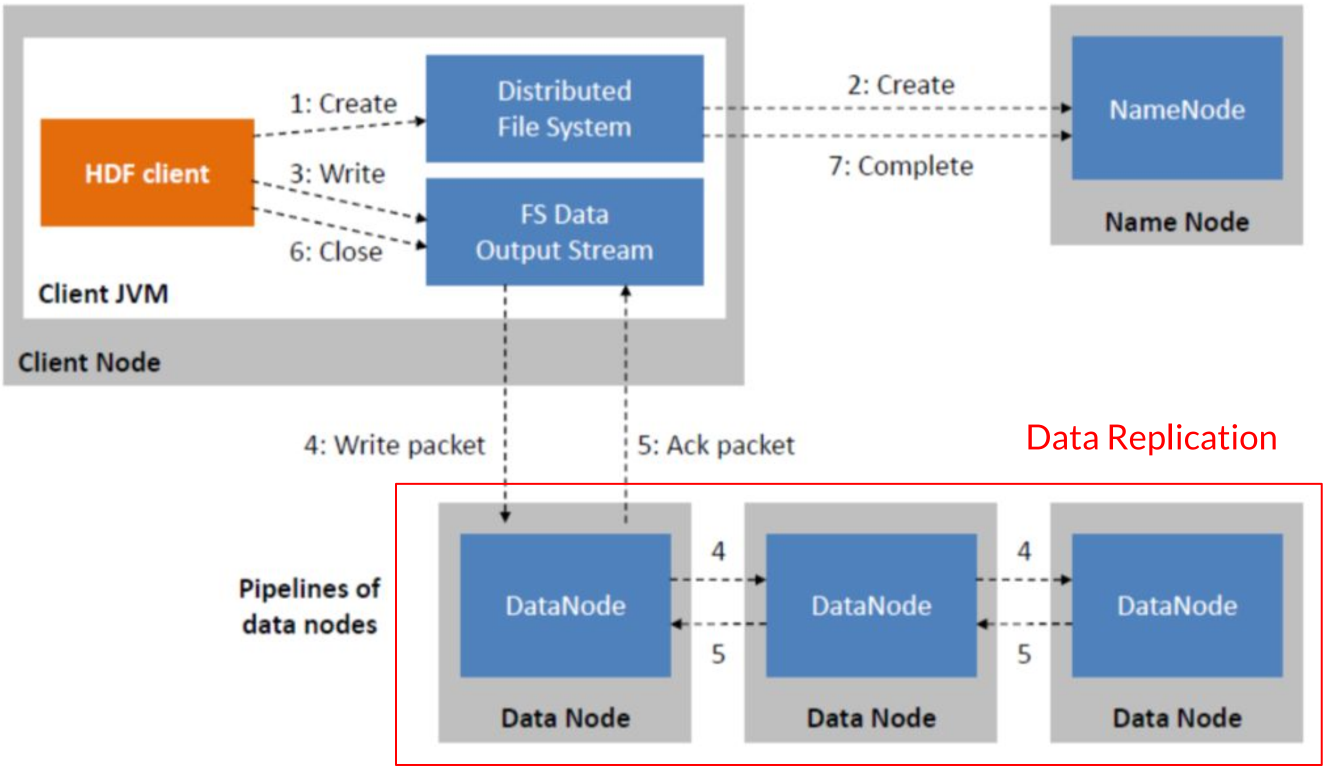
Read Operation in HDFS

- **Initiation:** The client sends a request to read a specific file from HDFS.
- **Consulting the Map**: The NameNode knows the filesystem structure.
  - It checks its metadata to find where the file's blocks are stored across different DataNodes.
- **Getting Directions**: The NameNode sends the client a list of DataNode addresses that hold the needed file blocks.
- **Direct Fetch**: Contact the DataNodes directly to retrieve the individual file blocks.
- **Reconstruction**: The client reassembles the received data blocks in the correct order to recreate the complete file.

Write Operation in HDFS



Write Operation in HDFS



Write Operation in HDFS

- **Initiation:** The client sends a request to the NameNode to create a new file in HDFS.

- **NameNode as Architect:**
  - It checks for file conflicts and permissions.
  - It creates a record in its metadata to keep track of the new file and its blocks.

- **Breaking it Down:** The client divides the data to be written into blocks (HDFS's storage unit).

Write Operation in HDFS

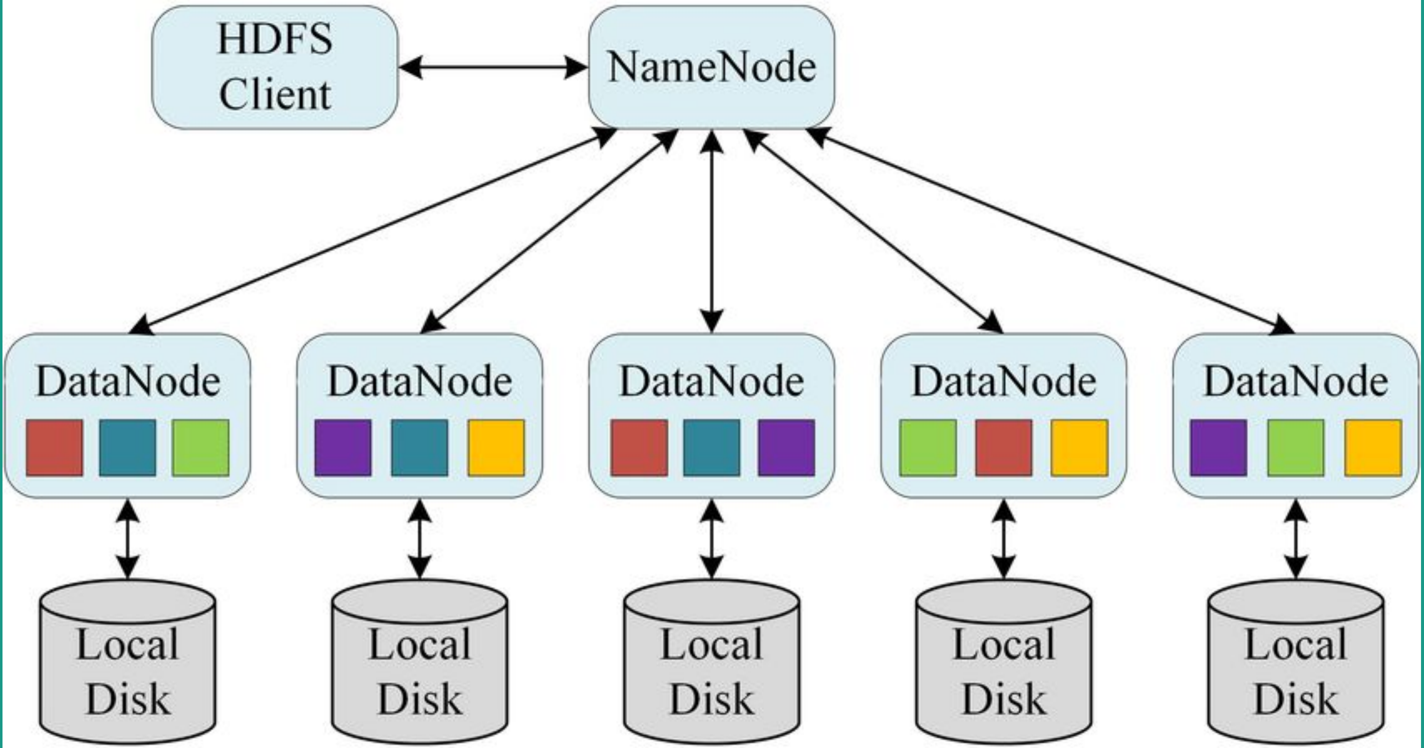
- **Strategic Placement:** The NameNode decides:
    - Which DataNodes will store each block (considering space, network).
    - How many replicas of each block to create (the replication factor).

- **Data Transfer:** Each DataNode:
    - Stores the received block
    - Forwards it to the next DataNode in the pipeline
    - Sends an acknowledgment back to the previous node/client

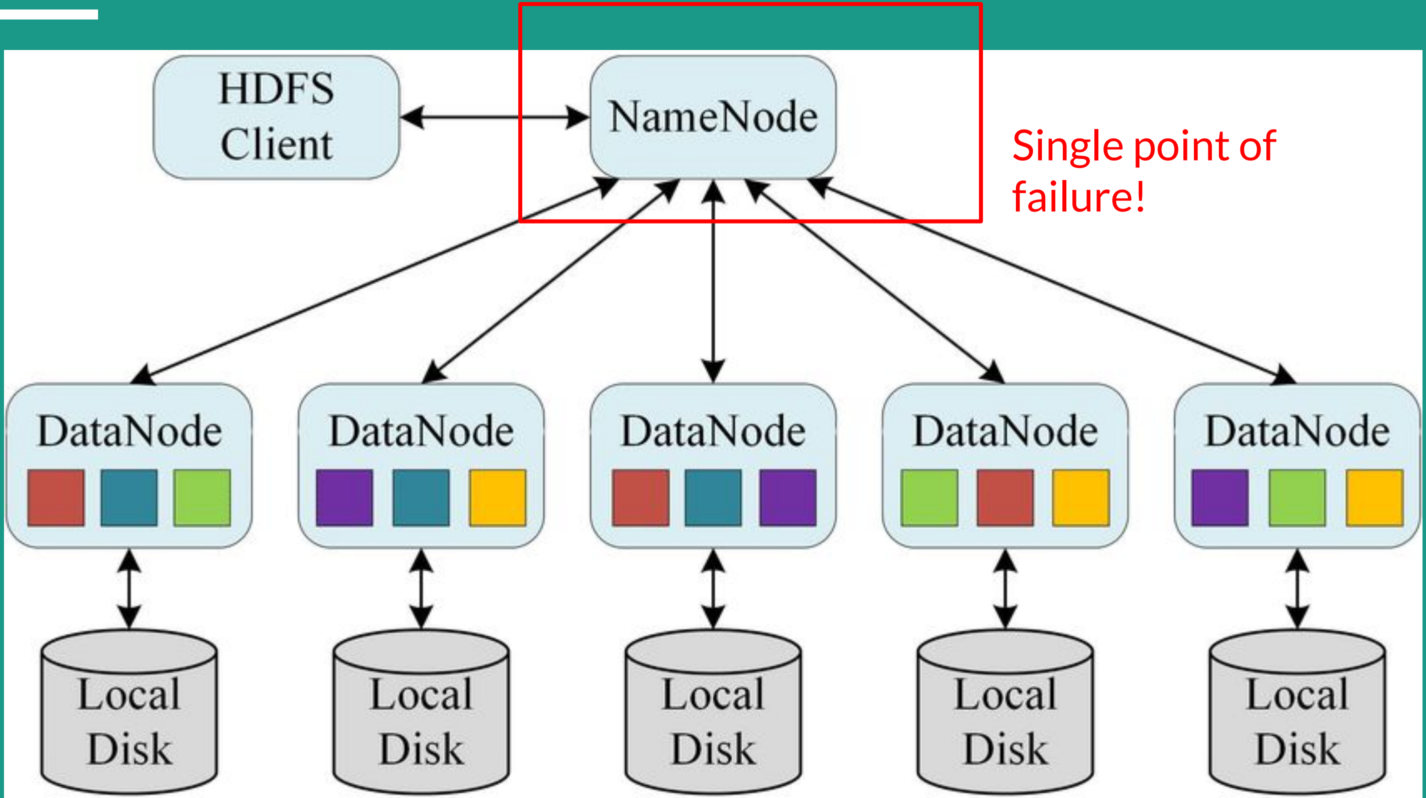
- **Completion:** After all blocks are written, the client notifies the NameNode.

- **Record Keeping:** The NameNode updates its metadata, reflecting the final locations of all blocks of the new file.

Find the weak spot!

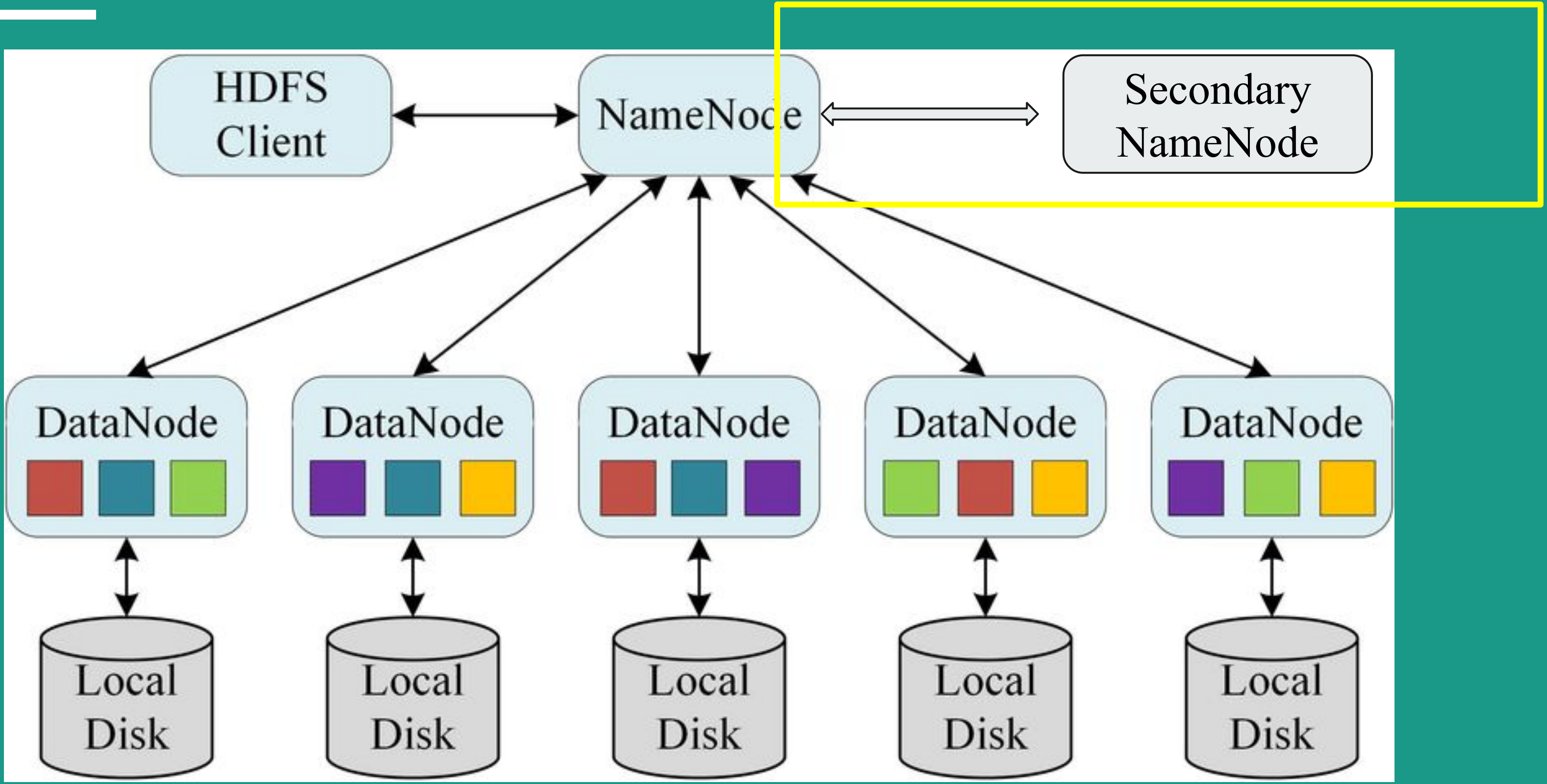


Find the weak spot!



Single point of failure!

Fix the weak spot!



Secondary NameNode

Not Just a Backup

- It's NOT a live replica.
  - The primary purpose isn't immediate failover.
- **Periodic Checkpoints**: It fetches copies of the NameNode's metadata
- **Key to Recovery:** In case of failure, these checkpoints drastically speed up rebuilding the NameNode's state.

Sample hdfs config

/etc/hadoop/conf/hdfs-site.xml

```
<property>
   <name>dfs.blocksize</name>
   <value>268435456</value>
</property>

<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>

<property>
   <name>dfs.namenode.http-address</name>
   <value>itracXXX.cern.ch:50070</value>
</property>
```

Sample hdfs config

/etc/hadoop/conf/hdfs-site.xml

```
<property>
   <name>dfs.blocksize</name>
   <value>268435456</value>
</property>

<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>

<property>
   <name>dfs.namenode.http-address</name>
   <value>itracXXX.cern.ch:50070</value>
</property>
```

NameNode Address

Sample hdfs config

`/etc/hadoop/conf/hdfs-site.xml`

```
<property>
   <name>dfs.blocksize</name>
   <value>268435456</value>
</property>
```

Replication Factor

```
<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>
```

```
<property>
   <name>dfs.namenode.http-address</name>
   <value>itracXXX.cern.ch:50070</value>
</property>
```

Sample hdfs config

/etc/hadoop/conf/hdfs-site.xml

Blocksize

```
<property>
   <name>dfs.blocksize</name>
   <value>268435456</value>
</property>

<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>

<property>
   <name>dfs.namenode.http-address</name>
   <value>itracXXX.cern.ch:50070</value>
</property>
```

Sample hdfs config

`/etc/hadoop/conf/hdfs-site.xml`

```
<property>
   <name>dfs.blocksize</name>
   <value>268435456</value>
</property>

<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>

<property>
   <name>dfs.namenode.http-address</name>
   <value>itracXXX.cern.ch:50070</value>
</property>
```

Full set of configs at - https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

Sample hdfs config

Step 8: Review the config files

```
[ ] !ls ~/input
```

```
capacity-scheduler.xml  hadoop-policy.xml  hdfs-site.xml    kms-acls.xml  mapred-site.
core-site.xml           hdfs-rbf-site.xml  httpfs-site.xml  kms-site.xml  yarn-site.xm
```

Given below are the list of files that you have to edit to configure Hadoop.

1. core-site.xml : The core-site.xml file contains information such as the port number used for Hadoop ins file system, memory limit for storing the data, and the size of Read/Write buffers.
2. hdfs-site.xml: The hdfs-site.xml file contains information such as the value of replication data, the nam of your local file systems. It means the place where you want to store the Hadoop infra.
3. yarn-site.xml: This file is used to configure yarn into Hadoop.
4. mapred-site.xml: This file is used to specify which MapReduce framework we are using.

Sample hdfs config

Step I: Name Node Setup

```
[ ] !/usr/local/hadoop-3.4.0/bin/hdfs namenode -format

WARNING: /usr/local/hadoop-3.4.0/logs does not exist. Creating.
2024-04-08 18:30:29,602 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = c95ab9dac78e/172.28.0.12
STARTUP_MSG:   args = [-format]
```

Sample hdfs config

Step II: Verifying Hadoop dfs

Add following to /usr/local/hadoop-3.4.0/etc/hadoop/hadoop-env.sh

```
export HDFS_NAMENODE_USER="root"
export HDFS_DATANODE_USER="root"
export HDFS_SECONDARYNAMENODE_USER="root"
```

Sample hdfs config

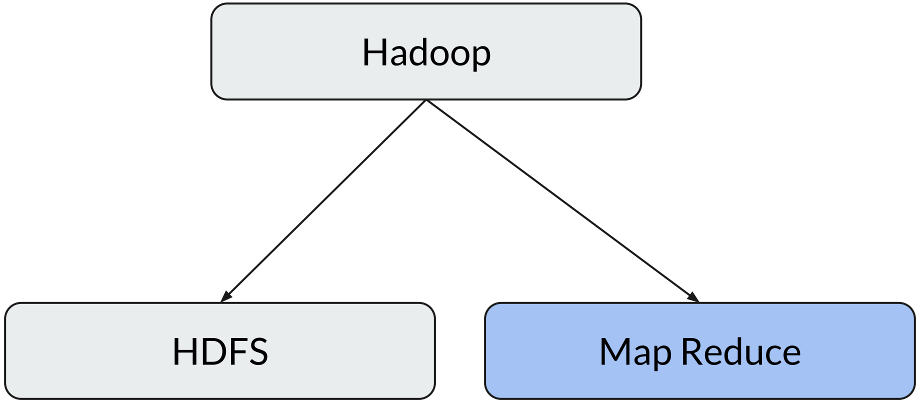
Use below command to verify that hdfs is running fine before we start working on it.

```
[ ]  !/usr/local/hadoop-3.4.0/sbin/start-dfs.sh

     Starting namenodes on [c95ab9dac78e]
     c95ab9dac78e: Warning: Permanently added 'c95ab9dac78e' (ED25519) to the list of known hosts.
     Starting datanodes
     Starting secondary namenodes [c95ab9dac78e]
```

Core components of Hadoop

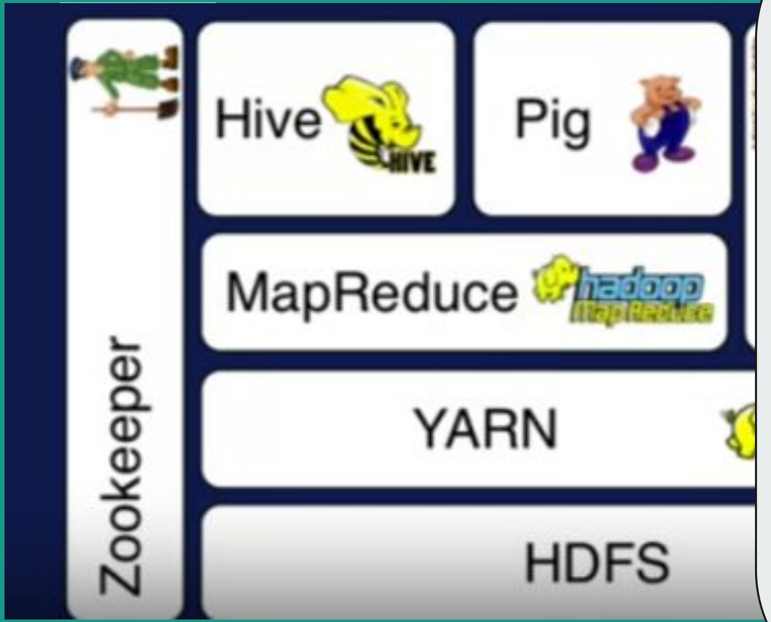
Hadoop Fundamentals: Storage & Processing



The Hadoop Zoo

Welcome to the Zoo!

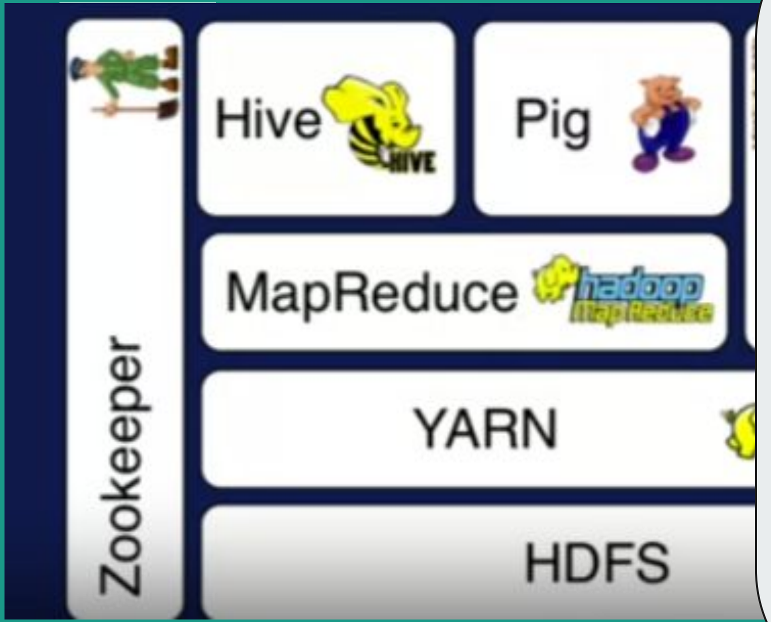
Welcome to the Zoo!



Fun Questions:

- Why is this a Zoo?
- Why is there a Zookeeper?

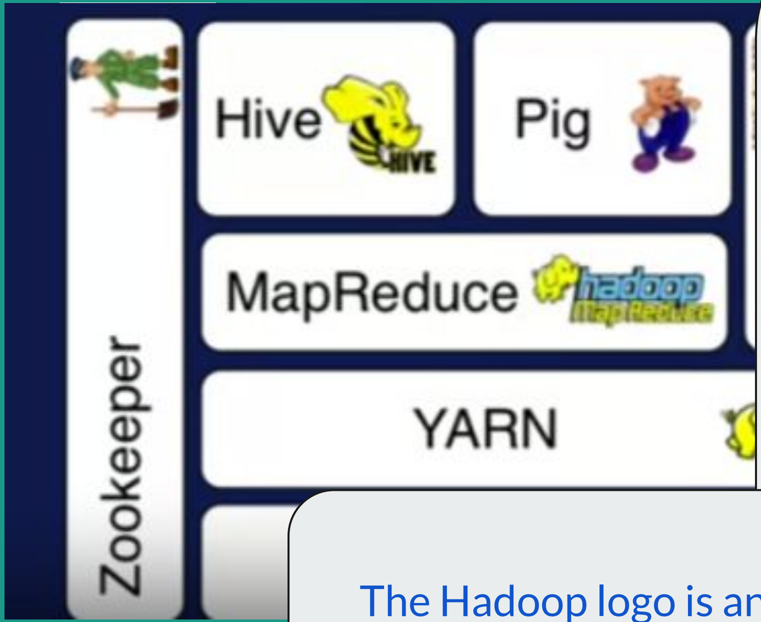
Welcome to the Zoo!



Fun Questions:

- Why is this a Zoo?
- Why is there a Zookeeper?
- Why is Hadoop logo an elephant?

Welcome to the Zoo!



Fun Questions:

- Why is this a Zoo?
- Why is there a Zookeeper?
- Why is Hadoop logo an elephant?
- Tool names like Hive, Pig ...

The Hadoop logo is an elephant because Doug Cutting named it after his son's toy elephant.

More Funny Names!

Ruby: A popular language named after the precious gemstone.

Python: Not just a snake reference; fans of the Monty Python comedy group might have influenced the choice.

Groovy: A Java-based language aimed at being dynamic and "groovy".

Django: A web framework named after the guitarist Django Reinhardt, highlighting its creative focus.

Wi-Fi: ?

More Funny Names!

Ruby: A popular language named after the precious gemstone.

Python: Not just a snake reference; fans of the Monty Python comedy group might have influenced the choice.

Groovy: A Java-based language aimed at being dynamic and "groovy".

Django: A web framework named after the guitarist Django Reinhardt, highlighting its creative focus.

Wi-Fi: Doesn't actually stand for anything! It was a marketing choice, aiming to sound similar to "Hi-Fi" (high-fidelity).