BIG DATA MANAGEMENT

Peeyush Taori

TA: Abhishek Jain, Hema Sri

ABOUT ME

- Passionate about Data Science, Big Data, and Capital Markets
- London Business School
- Indian School of Business
- Big Data and Data Science Consulting
- Hedge Fund Trading and Risk Management

PLAN FOR TODAY

1. Thought Experiment
2. Course Preview
3. Big Data – What and Why
4. Introduction to Hadoop
5. HDFS

THOUGHT EXPERIMENT

- Imagine you are the data architect of a small grocery store.
  - Keep track of sales and products.
  - ~ 10 MB data generated every day
- What can you do with this data?
  - Maximize profit, products to stock, who are buying.
  - Analysis can be done on a decent laptop
- Store is more recognized, and increasing footfall. Decide to open new store.
  - Combined data ~ 1GB
  - Faster computer (maybe a small server)
  - Move to RDBMS perhaps

THOUGHT EXPERIMENT

- Open stores in major cities (20 stores).
  - Data ~ 30 GB
  - Additional Questions: Competitor cost analysis, kind of ad campaigns, predict future sales & demand, cost optimization, supplier management
  - Need a high end cluster/RDBMS setup
- Go nationwide. Start a simple website – people post reviews.
  - Want to know what people are talking (Text analysis). How to do that.
  - Data ~ 1TB
  - Limitations of existing systems.
  - Need to store data, process it, and ensure it happens smoothly.
  - Enter Big Data.

THOUGHT EXPERIMENT

- Finally, open online portal.
  - Competitive, real-time pricing check
  - Stream and analyze data in real-time
  - Streaming data from multiple resources
  - Forecasting, Pricing analytics, gaining insights from data in real-time
- Big Data allows us to address these business questions efficiently

COURSE SUMMARY

Spark

- Spark Core
- Spark SQL
- Streaming
- GraphX

Hadoop

- Distributed File System (HDFS)
- Distributed Computing Engine (MapReduce) -- Not for now
- Cluster Manager (YARN)

Hive -- Next module

Pig – Next Module

AWS & Azure (Tutorials)

COURSE OBJECTIVES & GRADING

- Getting a good grade ☺
- Appreciate and understand major Big Data technologies
  - Conceptually and programmatically
- In Class Exercises (20%)
- Assignment (20%)
- Group Project (30%)
- Exams (30%)

RECOMMENDED TEXTBOOKS & SOFTWARE

Learning Spark: Lightning Fast Big Data Analysis. H. Karau & A. Konwinski. O'Reilly Media, 2015

Hadoop: The Definitive Guide (4th edition). T. White. O'Reilly Media, 2015

- We would be using Cloudera QuickStart
- Python as language of implementation
- Materials on LMS
  - Lecture slides
  - Recommended readings
  - Additional resources (Code snippets, detailed learning notes)

LEARNING FORMAT

Topic Introduction

- Motivation and Real Life Context
- Conceptual Understanding
- Architecture
- API and Data Types
- Toy Example
- Case Study

LET US BEGIN

RECORD THE TIME

Open (and possibly analyze) 1 GB file in Excel

Frequency table on a 3-4 GB file in R

Query on a 40 GB file in SQL

WHAT IS BIG DATA?

No standard definition

Characterized by 3 V

Volume

- Humongous amount of data available in the world
- Humanity since its inception till 2003 generated 5 exabytes. Same amount is now being generated every two days!!! And the pace is accelerating (Eric Schmidt, Google)

Velocity

- Speed at which data is being generated
- Today, we generate 4 Exabytes of data every day

Variety

- Structured, Unstructured (messages, text conversations, sensors, images, videos)

ONE INTERNET MINUTE

ONE INTERNET MINUTE

INTERNET MINUTE

UNTAPPED OPPORTUNITY?

- Industry projected size of ~187 Billion by 2019
- Big Data analysis critical to company strategy (Business Execs) KPMG Survey
- Huge shortage of Big Data professionals ~ 2,00,000 (McKinsey study)
- Issues?

THE CASE FOR BIG DATA

- Google

(https://www.google.com/intl/en/insidesearch/howsearchworks/thestory/index.html)

  - Indexes 60 trillion web pages worldwide
  - Data size: over 100,000,000 Gigabytes (100 Petabytes)
    - That's 100,000 iMacs (a terabyte each)
  - Processes over 20 Petabytes of data per day (http://techcrunch.com/2008/01/09/google-processing-20000-terabytes-a-day-and- growing/)
- Facebook

(http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes- ingested-every-day/)

  - 890M daily active users (745M daily active mobile)
  - 500 Terabytes of data ingested per day
    - E.g. 300M photos per day

BIG DATA AT WAL-MART

- 1 million customer transactions per hour
- 2.5 Petabytes of data collected per day

(Big Data: The Management Revolution, McAfee and Brynjolfsson, HBR, Oct 2012)

- Applications:

(http://blogs.wsj.com/cfo/2013/11/22/now-trending-big-data-at-walmart-_com/)

  - Set adwords pricing
  - See what's trending in social media
  - Buying patterns among customers
  - Analyze competitors' pricing in real-time
  - Changed shipping policy based on analysis – free shipping minimum reset from $45 to $50

HOW DO ORGANIZATIONS USE BIG DATA?

- Telecom : Vodafone generates ~4 TB CDR each day
  - Use for network quality
  - Customer churn
- GE: Big Data in industrial use
  - Predicting maintenance periods of key manufacturing machines
  - Optimizing delivery routes
  - Best way to deal with energy finance
- Ford: Data on ~ 4-5 Million vehicles
  - How do people drive vehicles
  - Key component issues
  - Design better vehicles

HOW DO ORGANIZATIONS USE BIG DATA?

- LinkedIn: People you may know
- AT&T: Patterns about usage and user movements
- Amazon: Recommendations
- MIT used location data from Macy's parking lot to estimate Thanksgiving sales – before Macy's could!
- Anything homegrown??
  - Aadhar project (Govt of India)

IOT & BIG DATA

BIG DATA

DATA AT THE SCALE OF THE WEB

What if you get TB or PB of data every day?

- How would you store it?
- Analyze?

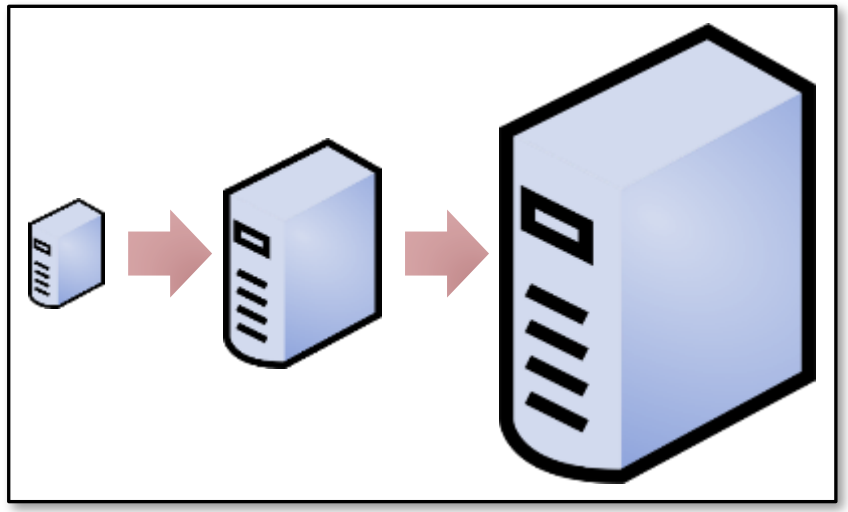
The rise of the World Wide Web increased the scale of databases dramatically. Imagine trying to process the text of millions of web pages (Yahoo, Google), or organize a billion personal profiles plus all of their connections (Facebook, LinkedIn)!

Companies like Amazon, Google, and Facebook had to solve problems where their data simply couldn't fit onto one machine.

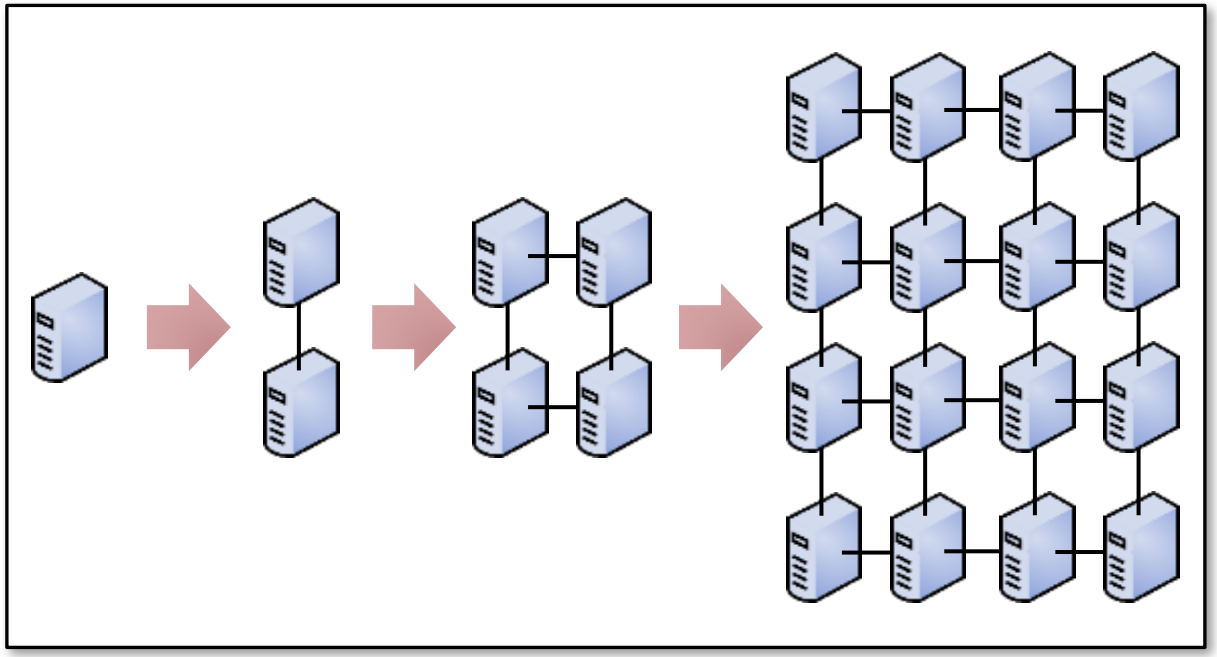
As a result, these companies became innovators in cluster-based computing, and their inventions were the models for Hadoop, NoSQL, and other big data technologies.

SCALING HORIZONTALLY

Instead of scaling up to bigger and bigger computers, the Web giants were the leaders in learning how to "scale out" by distributing their computing problems across multiple machines. They employ clusters of off-the-shelf computer hardware in huge data centers. The cost scales (almost) linearly with the power they need.



Scaling "up" to bigger systems means nonlinear increases in cost



Scaling "out" to bigger clusters of commodity-priced systems has almost a linear cost curve

HOW CAN YOU DISTRIBUTE A DATABASE?

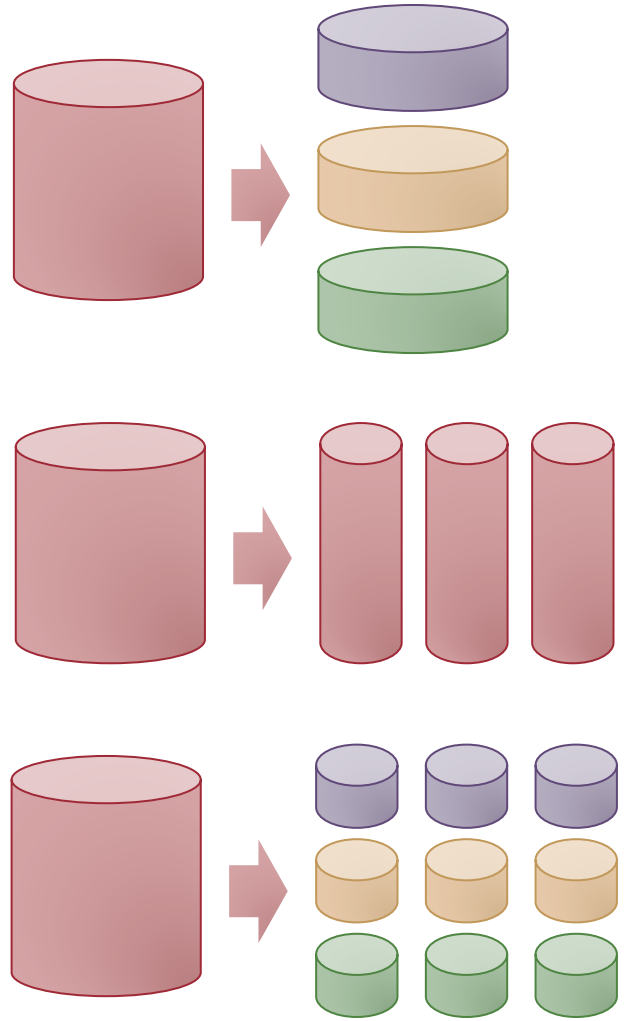
Partitioning or "**sharding**" = putting different parts of the data in different places

- When there's just too much data to store in one place, or, when logical subsets of the data could benefit from being optimized or managed differently.
- May be used for security.

Replication = making multiple copies of the same data in different places

- When one server becomes a bottleneck that slows down the system, and/or you want to speed up access time for the users.
- When you want to have insurance against one server or one data center going down.

Partitioning + Replication



SCHEDULING AND COORDINATION

A more vexing challenge is the complexity of how to break up your program into parts, and distribute those parts around the computer cluster, and make sure that the results are correctly combined when they finish.

You would have to have expertise in distributed systems, and put in a lot of hard work, to be able to do this. Therefore, it's not likely to be worth it except for the most important applications.

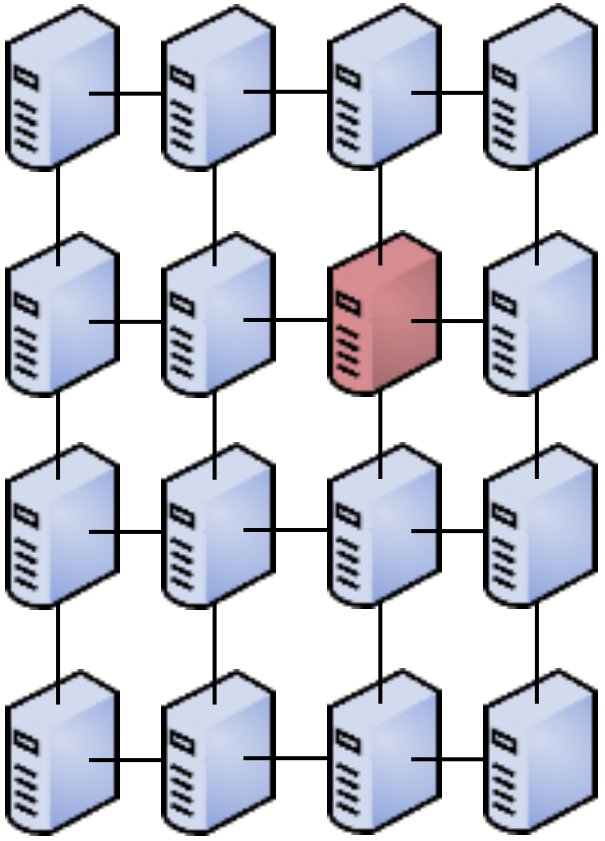
HANDLING FAILURE GRACEFULLY

For any individual computer, failure is quite rare.

But if you have hundreds of computers working together in a cluster for a long period of time, some of them will fail.

If you have a job that takes several hours to run, what happens when a node fails in the middle? Do you stop, and start again?

Systems that do Big Data processing on large clusters must be designed to accept failures in the cluster and keep on working. That requires replication of data, failure detection, and other complex functionality.

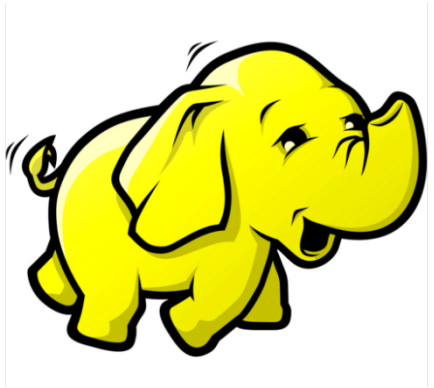


ENTER HADOOP

The business value of Hadoop is this: Hadoop solves the low-level problems of scheduling and coordinating work in a shared-nothing computer cluster, sends work to the nodes where data is stored so that it can be processed most efficiently, and detects and recovers from partial failure in the cluster.

Hadoop solves these fundamental problems for parallel processing, so you don't have to. You can focus on writing a small program to analyze data, and Hadoop will make sure the program is executed by the cluster.

WHAT IS HADOOP?



HADOOP

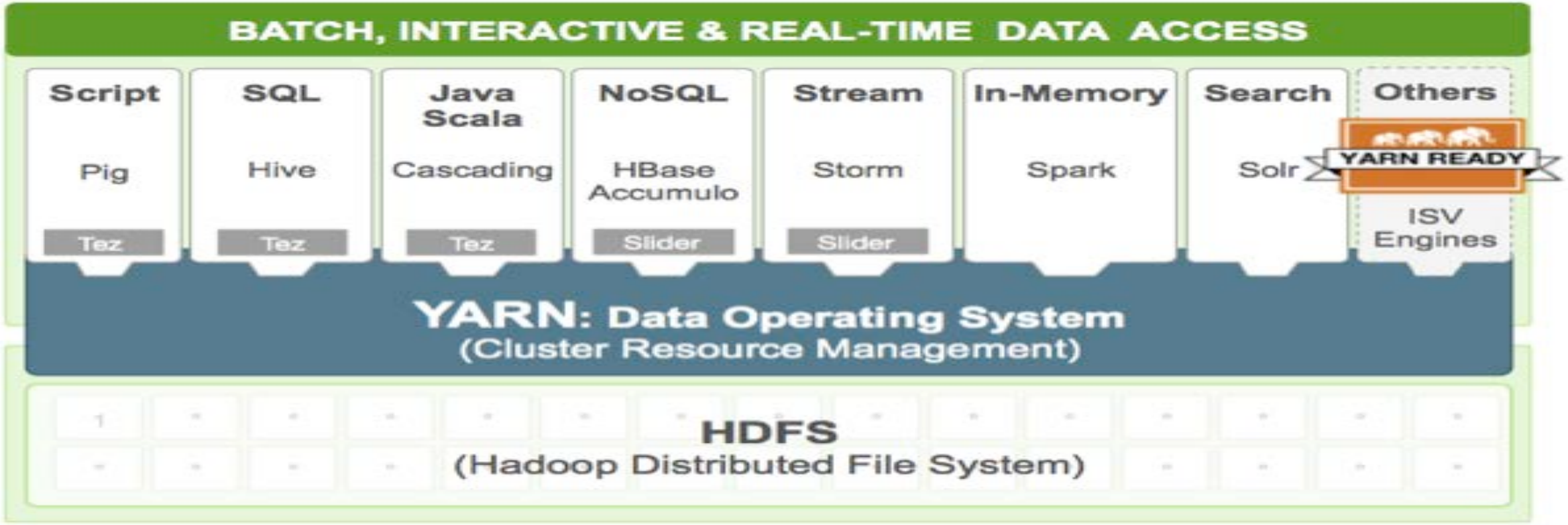
- Google published a white paper on MapReduce framework (2004)
- Doug Cutting & Mike Cafarella made Hadoop (2005)
- It is NOT Big Data, neither is it a database
- Platform (framework)
- Core Hadoop Components
  - Hadoop Distributed File System (HDFS)
  - Distributed Computing Engine (MapReduce)
  - Cluster Manager (YARN)
- Roughly 2.9 M lines of code

WHY HADOOP

- Inexpensive commodity hardware
- Free open source software
- Scalable
- Reliable
- Enables data archival and reporting (conventional uses)
- Enables cutting edge analytics

THE HADOOP ECOSYSTEM

HADOOP ECOSYSTEM



Source: http://hortonworks.com/hadoop/yarn/

MAJOR COMPONENTS OF HADOOP

HDFS = Hadoop Distributed Filesystem: A file system for storing big datasets on a cluster; it makes the cluster look like it's just one big disk. Behind the scenes, it replicates the data and protects against node failure.

MapReduce: a programming framework for parallel processing of data. The user defines two programs – a "mapper" and a "reducer" – and Hadoop coordinates execution throughout the cluster.

YARN and the rest of the ecosystem: YARN is the resource manager and scheduler in newer versions of Hadoop, and it allows other applications besides MapReduce to be executed within a Hadoop cluster. This has led to a proliferation of new Hadoop-based systems like Pig, Hive, HBase, and Spark, which we will see during this course.

SELECTED COMPONENTS

Hadoop has attracted a large number of new projects that build on its technology. Some of them translate high-level abstractions into MapReduce jobs. Others bypass MapReduce entirely, and build different functionality on top of HDFS and YARN.

Most of them have funny names.

PIG & HIVE

Pig and Hive are two tools that allow the user to write data processing programs at higher levels of abstraction, and automatically translate those commands into MapReduce jobs.

Pig provides a "data flow" language, PigLatin. You can write scripts that link together several transformations – filters, joins, groupings – which would be very complex to code as MapReduce jobs. Pig's execution environment translates those instructions into MapReduce.

Hive, invented by Facebook, is one of the oldest "SQL on Hadoop" technologies. It allows you to use Hadoop as a kind of relational database, with one exception: the schema is defined at query time, not at the time the data was written to HDFS. So you can change the schema and do different types of queries later.

FLUME & SQOOP

Flume and Sqoop are two projects that help users to transfer data into HDFS.

Flume is designed to store streaming data like application logs from distributed systems into HDFS.

Sqoop is an increasingly popular tool for transferring data from relational databases into HDFS, and back. This allows you to use Hadoop to analyze data in existing systems, without transforming it too much.

SPARK

Apache Spark is a tool for interactive and multi-step analyses on a Hadoop cluster.

When a particular batch process requires several MapReduce steps, the time Hadoop takes to store the output of each step to disk can make processing very slow. Spark solves this problem by keeping data in-memory as much as possible, so several MapReduce steps can be carried out in rapid sequence.

HBASE

HBase was one of the first databases to use HDFS for interactive querying, delivering results in near-real time.

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

HDFS

HDFS is the Hadoop distributed filesystem. It splits large files up into blocks and stores them on different nodes in the cluster to enable parallel processing.

Each computer in the cluster has its own (Linux) operating system and file system. HDFS works on top of the underlying file system.

HDFS replicates data. This is part of a strategy to protect against node failure.

HDFS is aware of the topology of the data center, and tries to place data in ways that take advantage of it.

Master/Slave architecture

HDFS

- File System for storing very large files across computers (just like our individual File Systems on Windows/Mac/Linux)
- Data organized into files and directories
- Files are divided into blocks and these blocks are stored on individual machines
- Remember, HDFS is not storage. It is a filesystem.

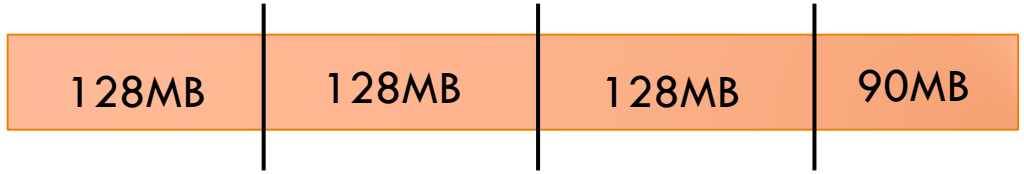
HDFS "BLOCKS"

In HDFS, massive files are split up into blocks. Blocks are up to 128MB by default, although different Hadoop distributions may have different defaults.

a big input file

HDFS "BLOCKS"

In HDFS, massive files are split up into blocks. Blocks are up to 128MB by default, although different Hadoop distributions may have different defaults.



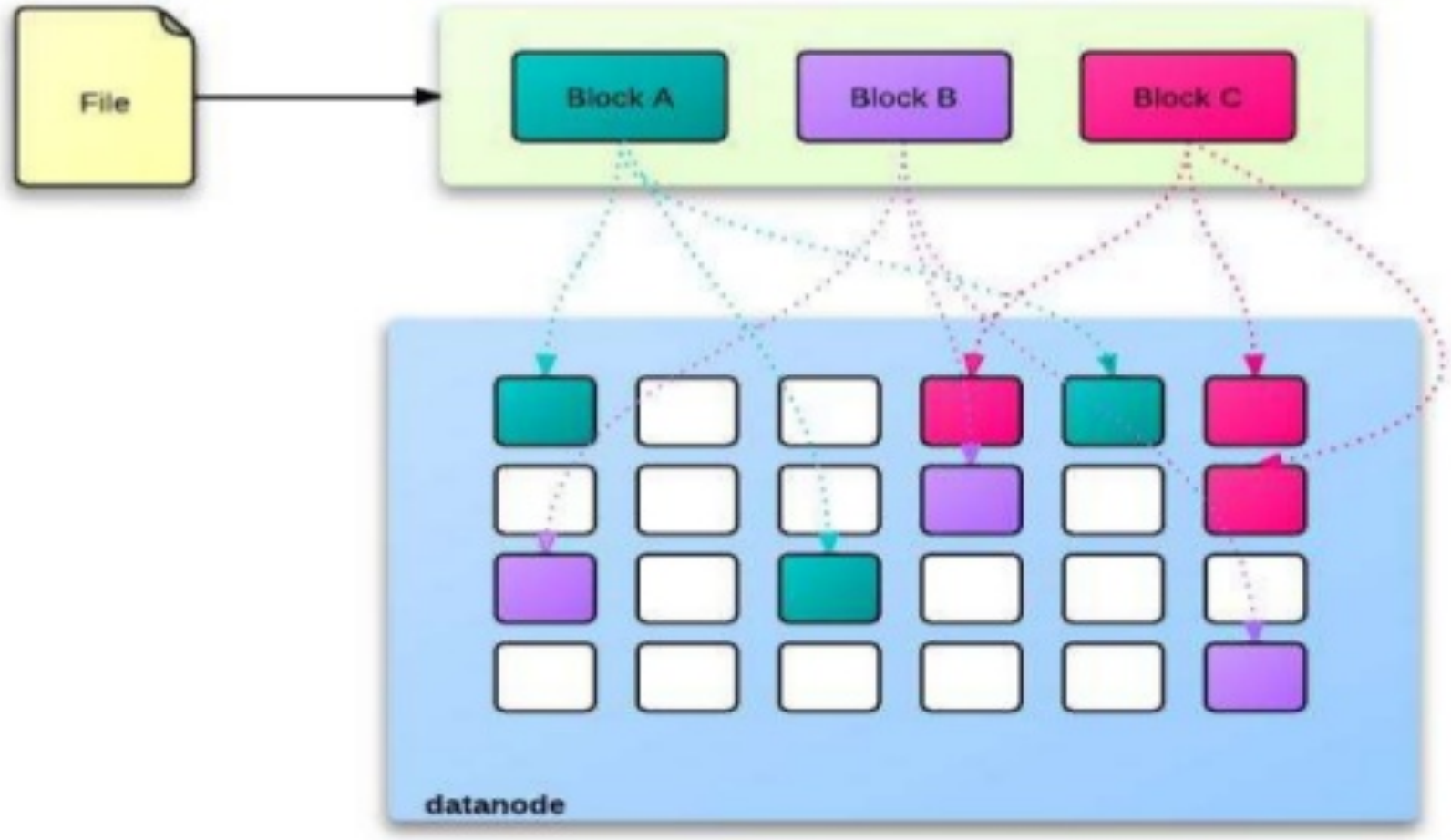
If there isn't 128MB of data, however, a block may be smaller.

Standard-sized blocks give HDFS two advantages:

Data can be read in a stream from the disk. This minimizes hard drive "seeking" so that data can be read from the disk much faster. Time to transfer > time to seek

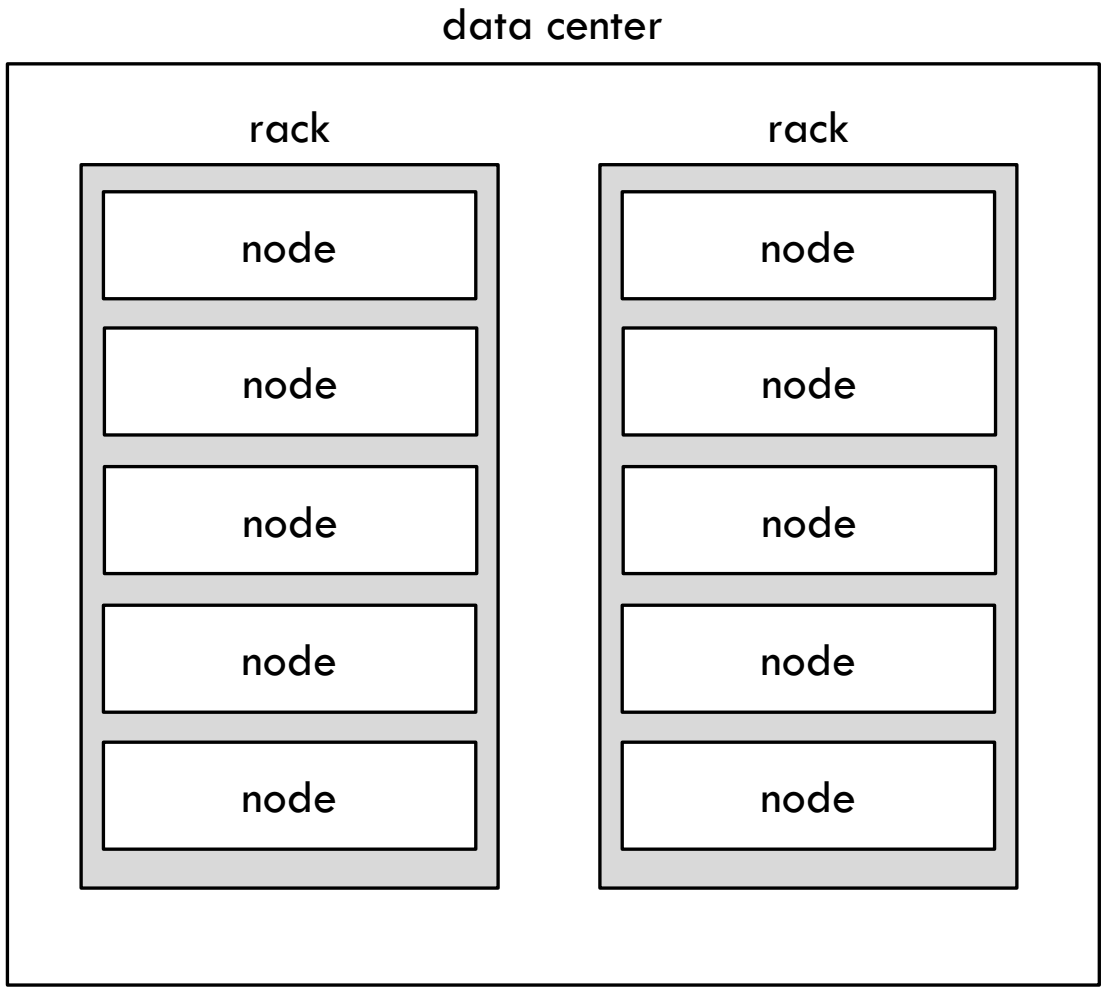
Hadoop can easily calculate how many blocks can be fit on a node. These blocks are large enough to read quickly from disk, small enough to fit easily where needed.

HDFS STORAGE

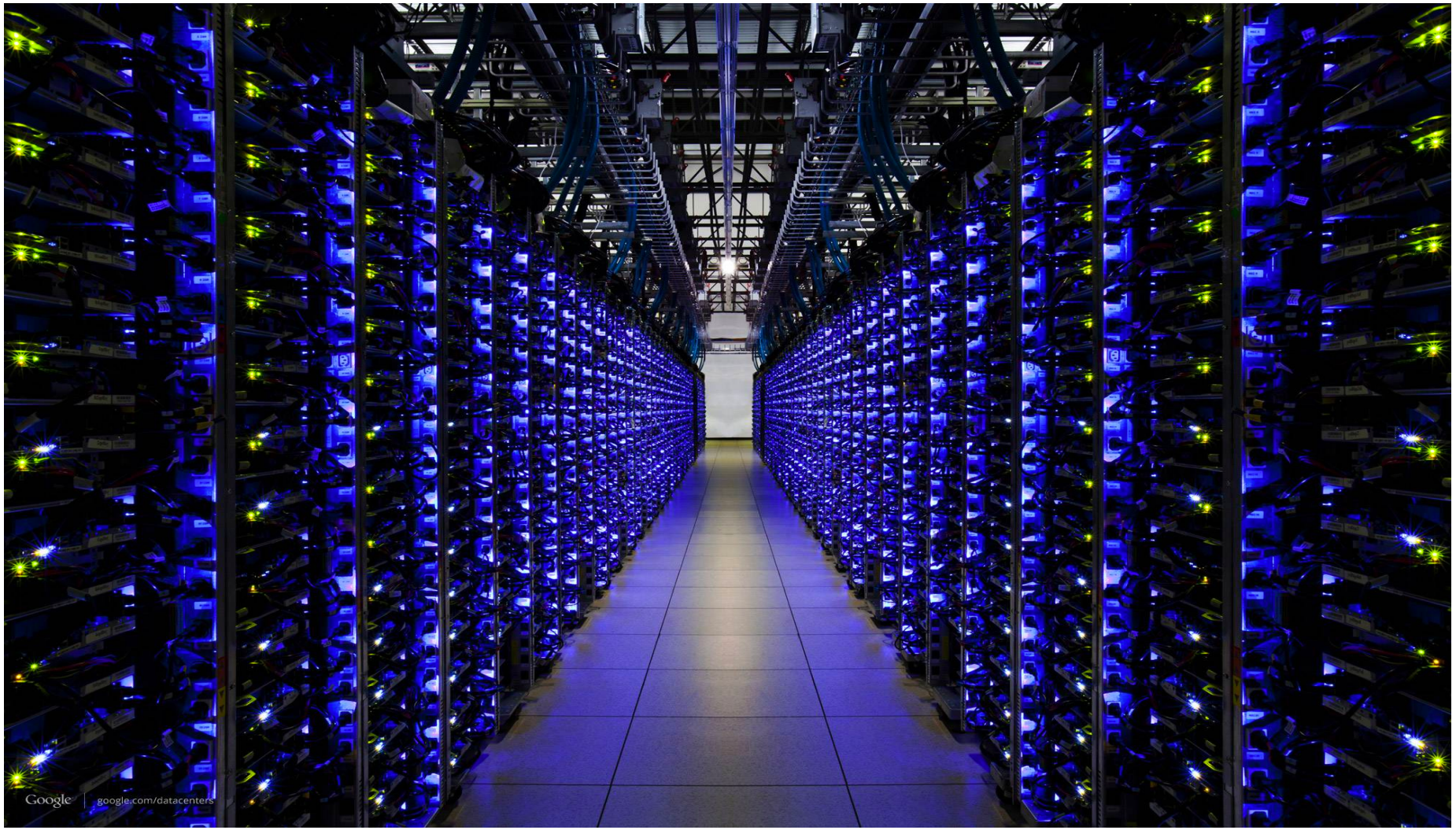


REPLICATION OF DATA

HDFS by default makes three replications of each block. Replication provides backup in case one copy is on a node that fails. Also, when a job must be run on a data block, HDFS can decide which copy is on a node that's less busy.



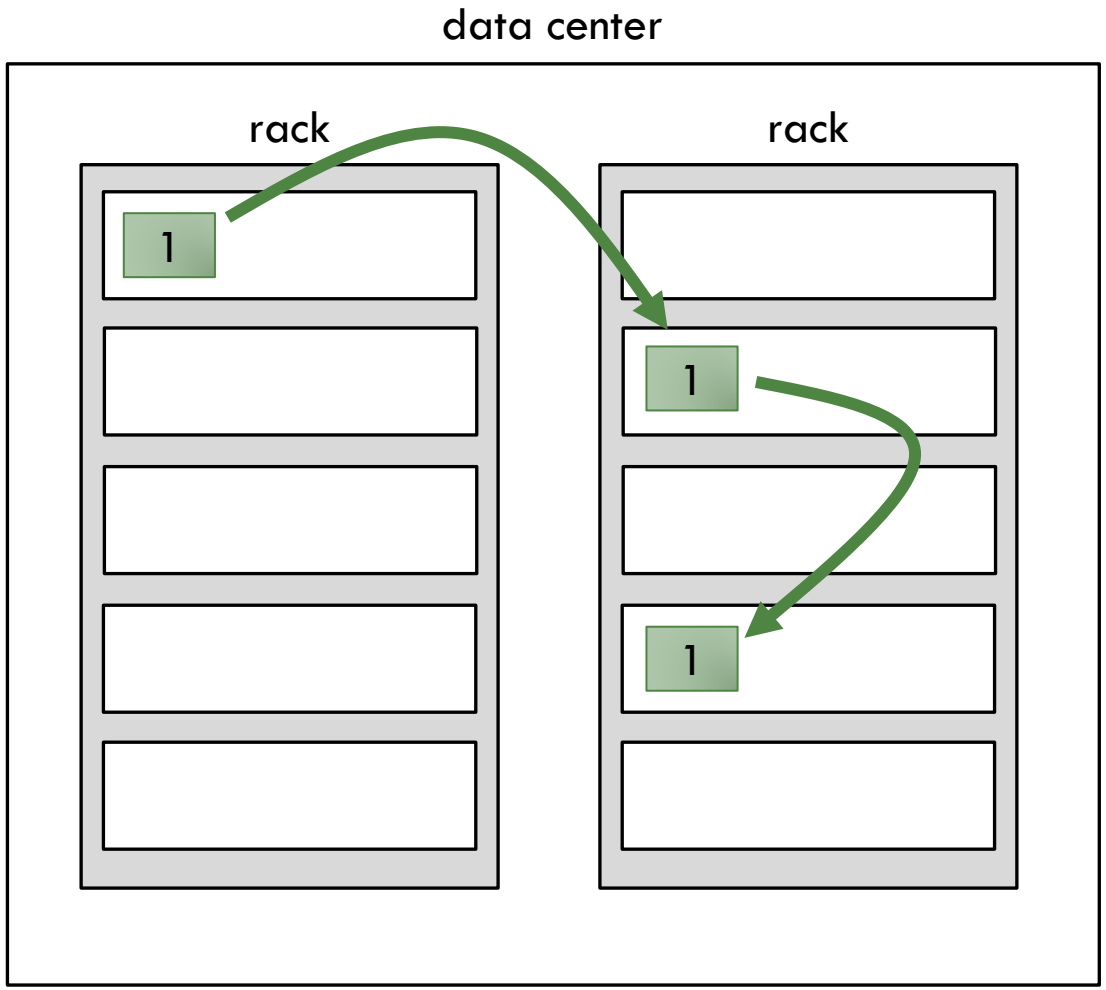
SERVER RACK AT GOOGLE DATA CENTRE



REPLICATION OF DATA

HDFS by default makes three replications of each block. Replication provides backup in case one copy is on a node that fails. Also, when a job must be run on a data block, HDFS can decide which copy is on a node that's less busy.

HDFS is aware of the topology of the network. The typical replication pattern is: first write to a randomly-selected node; then write to a node in a different rack; then write to a third node in the second rack.

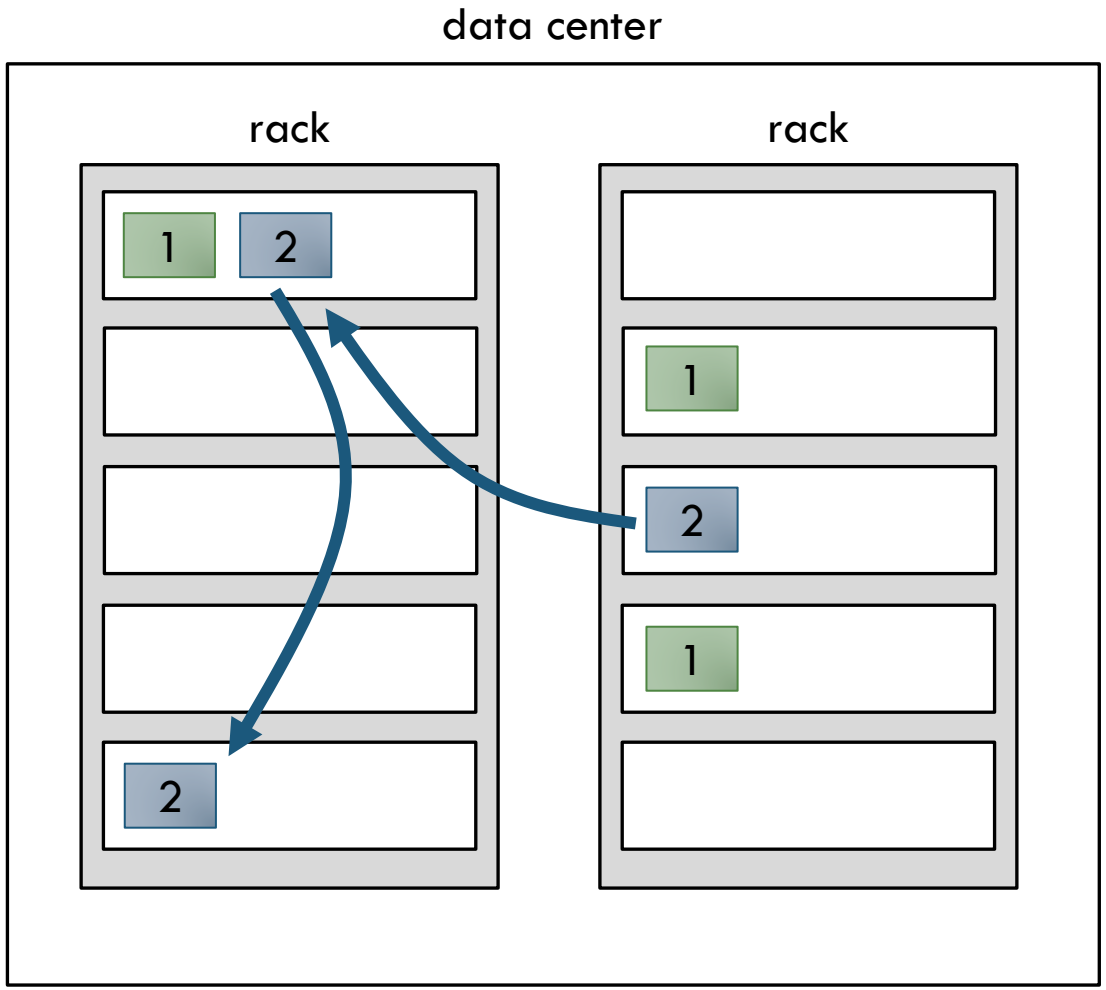


REPLICATION OF DATA

Communication between computers in a rack is faster than communication between racks.

Communication between racks in a data center is faster than communication between data centers.

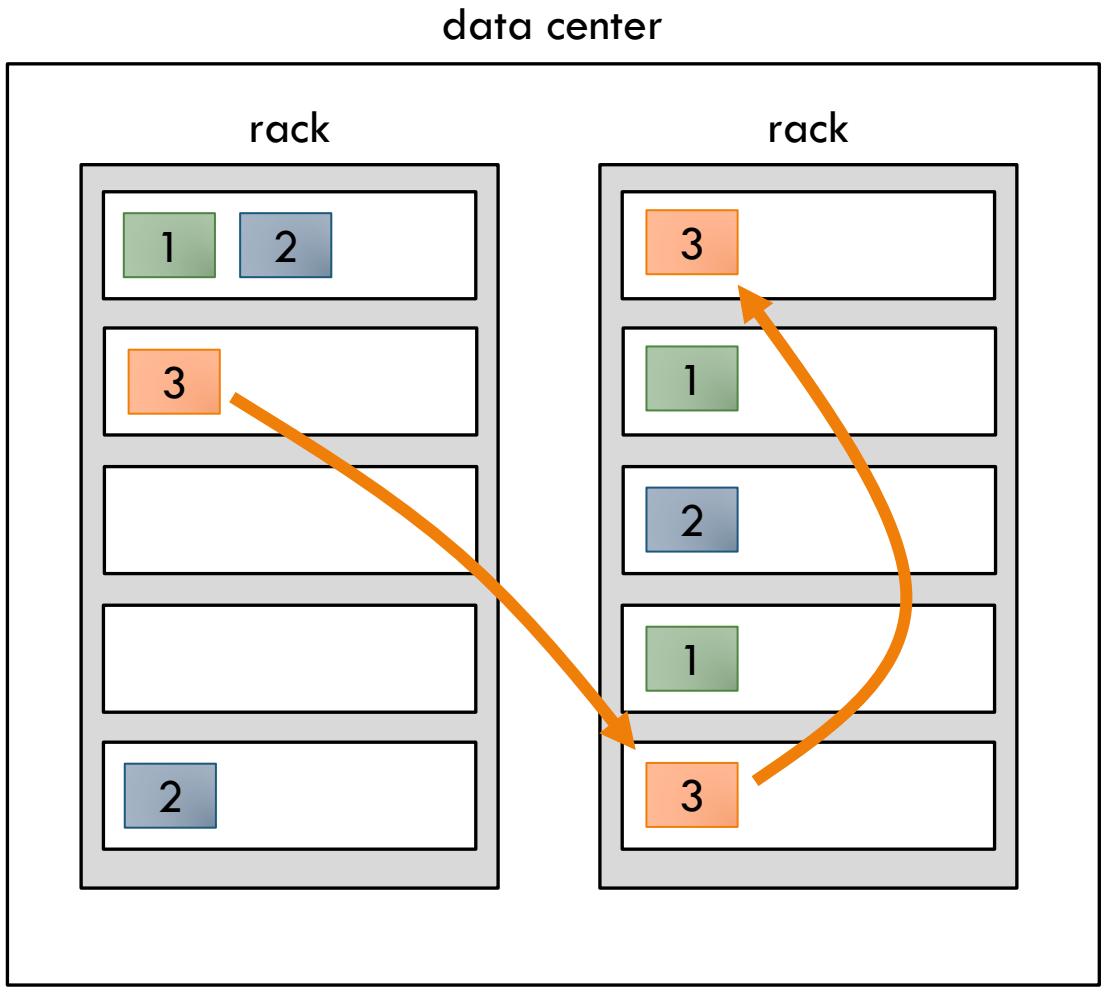
By this strategy, data is only copied from rack to rack once, minimizing the network burden.



REPLICATION OF DATA

In exchange for this small cost, we gain reliability – the data will survive if one node fails or even if an entire rack fails.

It also becomes possible to read the data from an alternative node if one node is busy or even if the entire rack is busy.



ARCHITECTURE

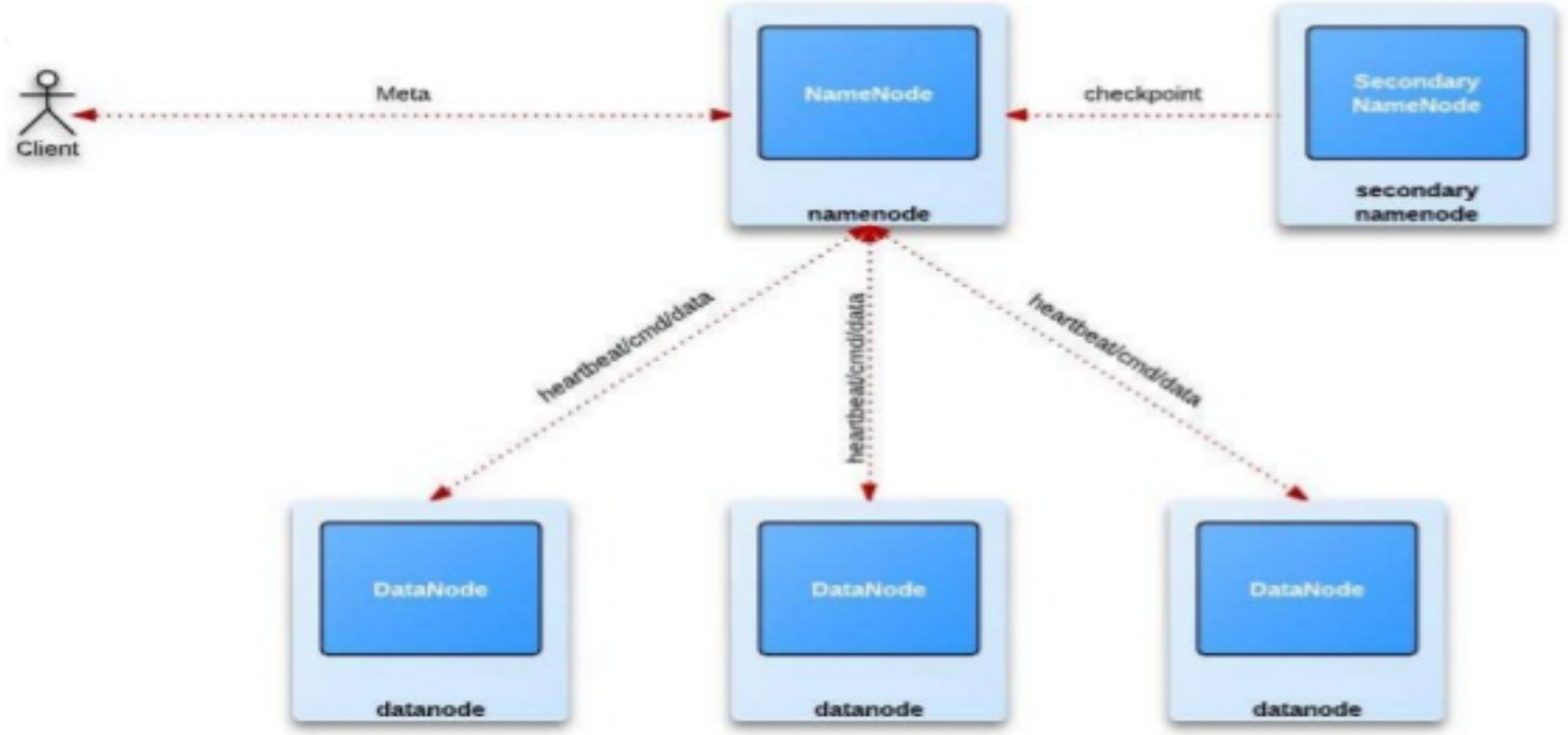
Master/Slave architecture

- Single namenode - Manages Metadata
- Multiple datanodes – They actually store the data
- Secondary Namenode – Backup or checkpoint system if namenode goes down
- Exposes file system namespace
- Internally data stored into blocks

Namenode

- Most important machine in HDFS
- Stores metadata
- Executes file system namespace operations
- Internal mapping of blocks
- Replication factor

HDFS ARCHITECTURE



NAMENODE

Contains FileSystem Meta Data

- File System Names
- Blocks Info
- Replicas Info

Interacts with

- Client
- Datanode
- Secondary Namenode

SECONDARY NAMENODE

Serves as a kind of backup

However, only does checkpointing and has NON REALTIME copy of namenode (every 5 min by default)

In event of namenode going down, this has to be manually brought up

DATANODE

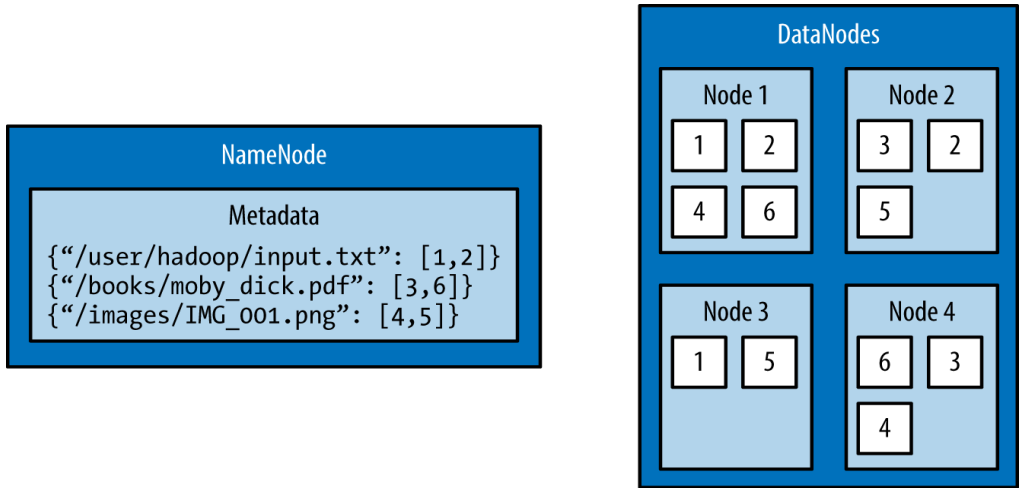
Stores blocks of data

Data is stored on local filesystem. Each block of data is stored in a separate file.

Serve read and write requests

NameNode replicates datanode blocks in event of failure

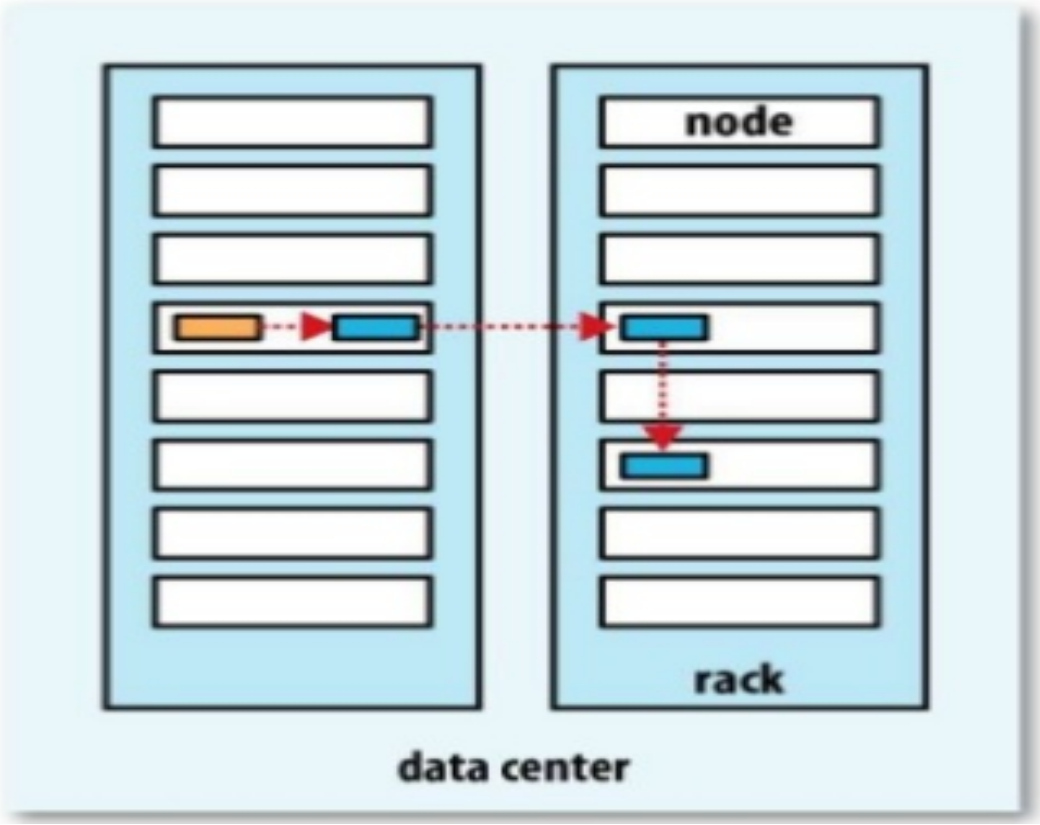
Client directly gets data from datanote, not from namenode. Why??



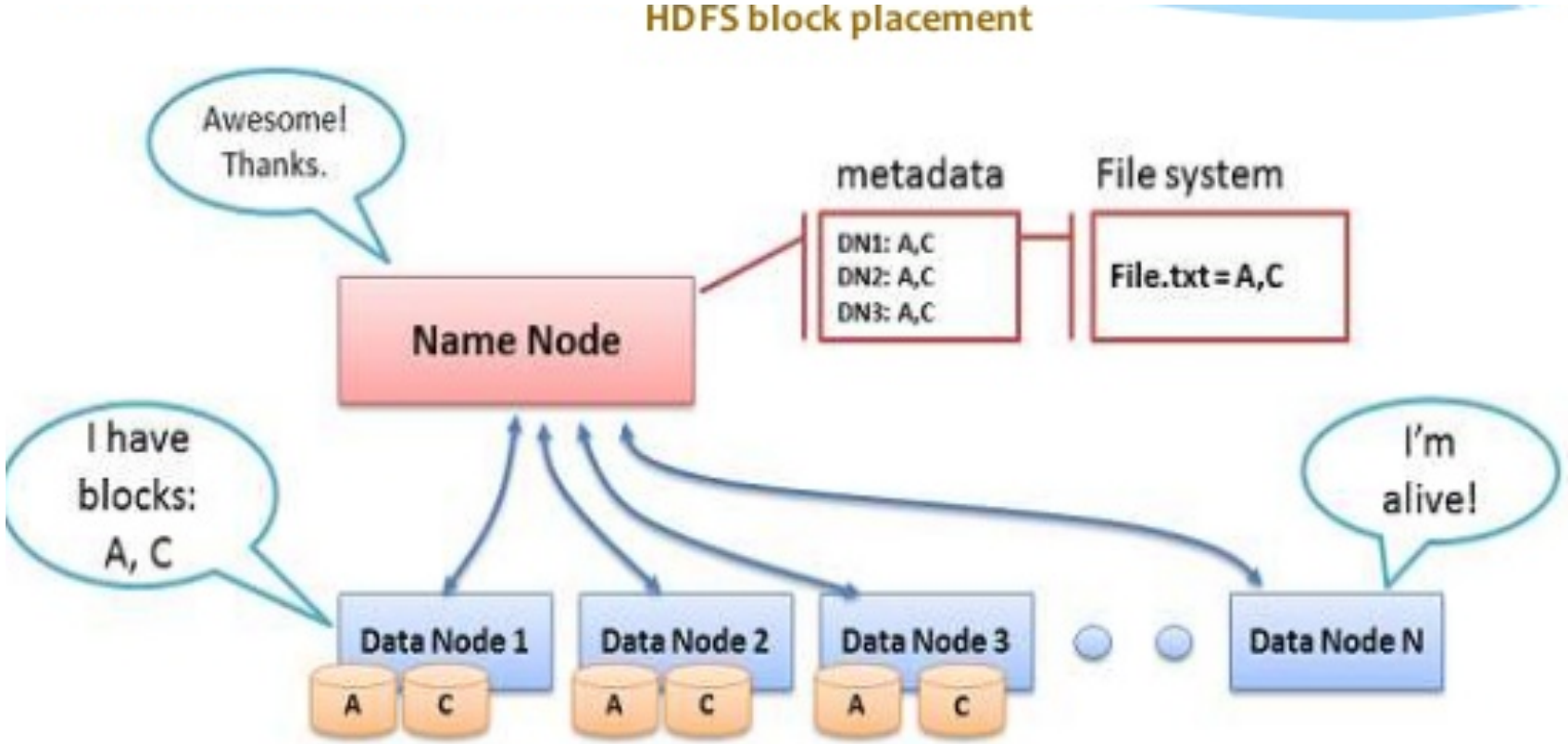
DATANODE

On Cluster Level

- replication = 3
  - First replica local with Client
  - Second & Third on two nodes of same remote rack



HDFS BLOCK PLACEMENT

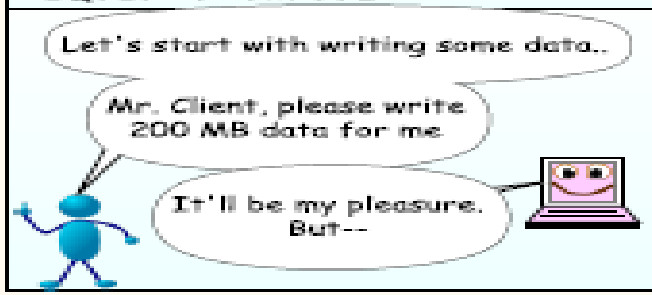


HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

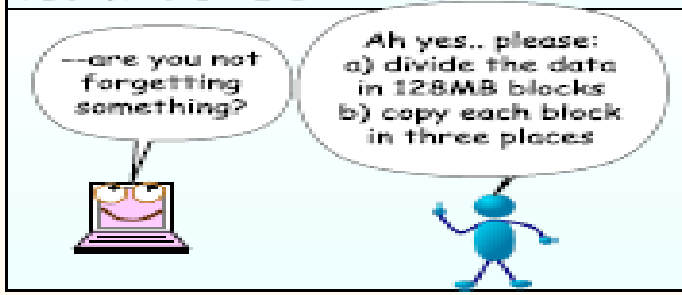
THE CAST

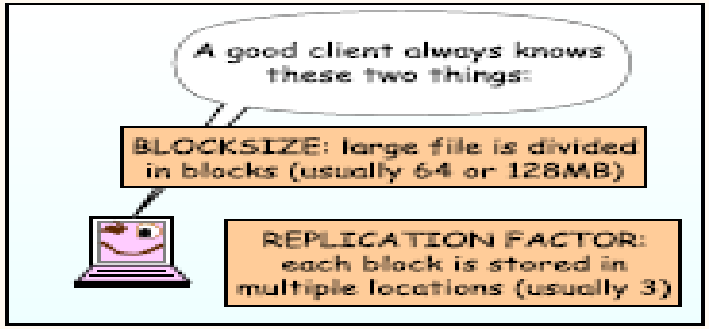
WRITING DATA IN HDFS CLUSTER

REQUEST FROM USER

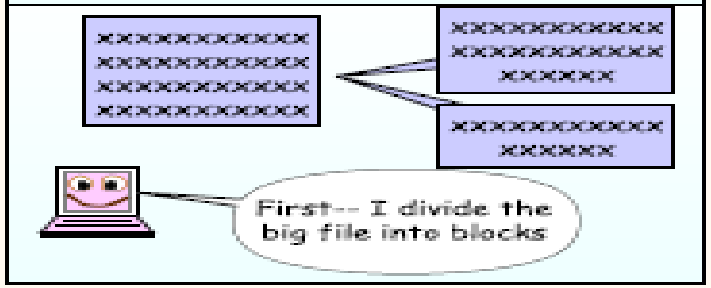


BLOCK AND REPLICATION

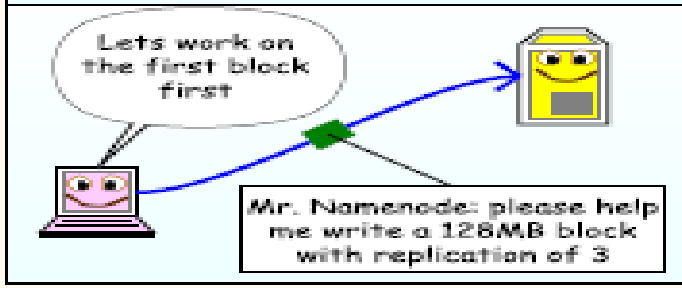




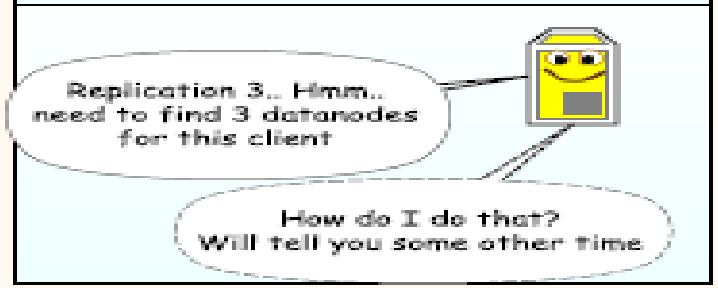
DIVIDE FILE INTO BLOCKS

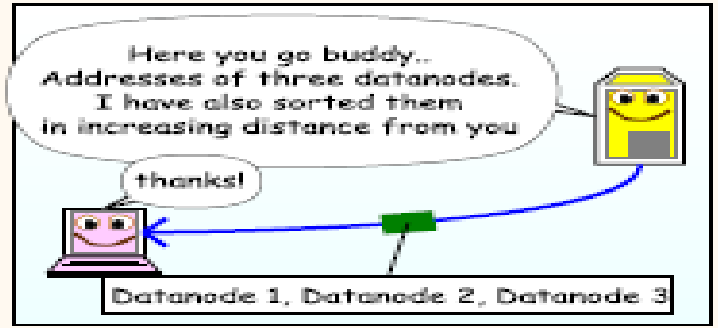


ASK NAMENODE

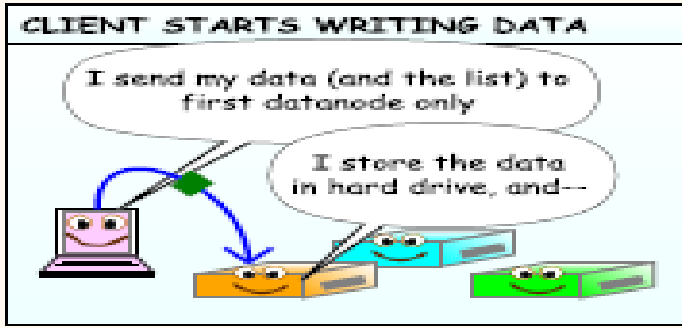


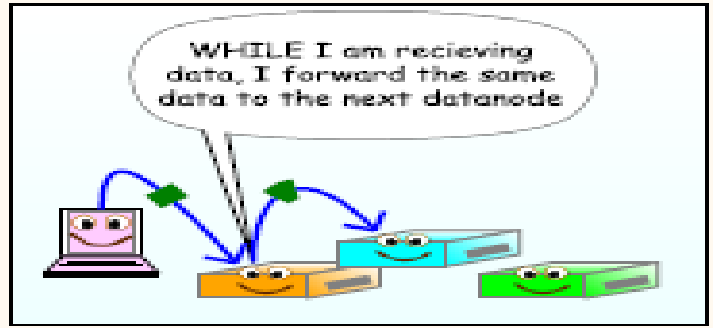
NAMENODE ASSIGNS DATANODES

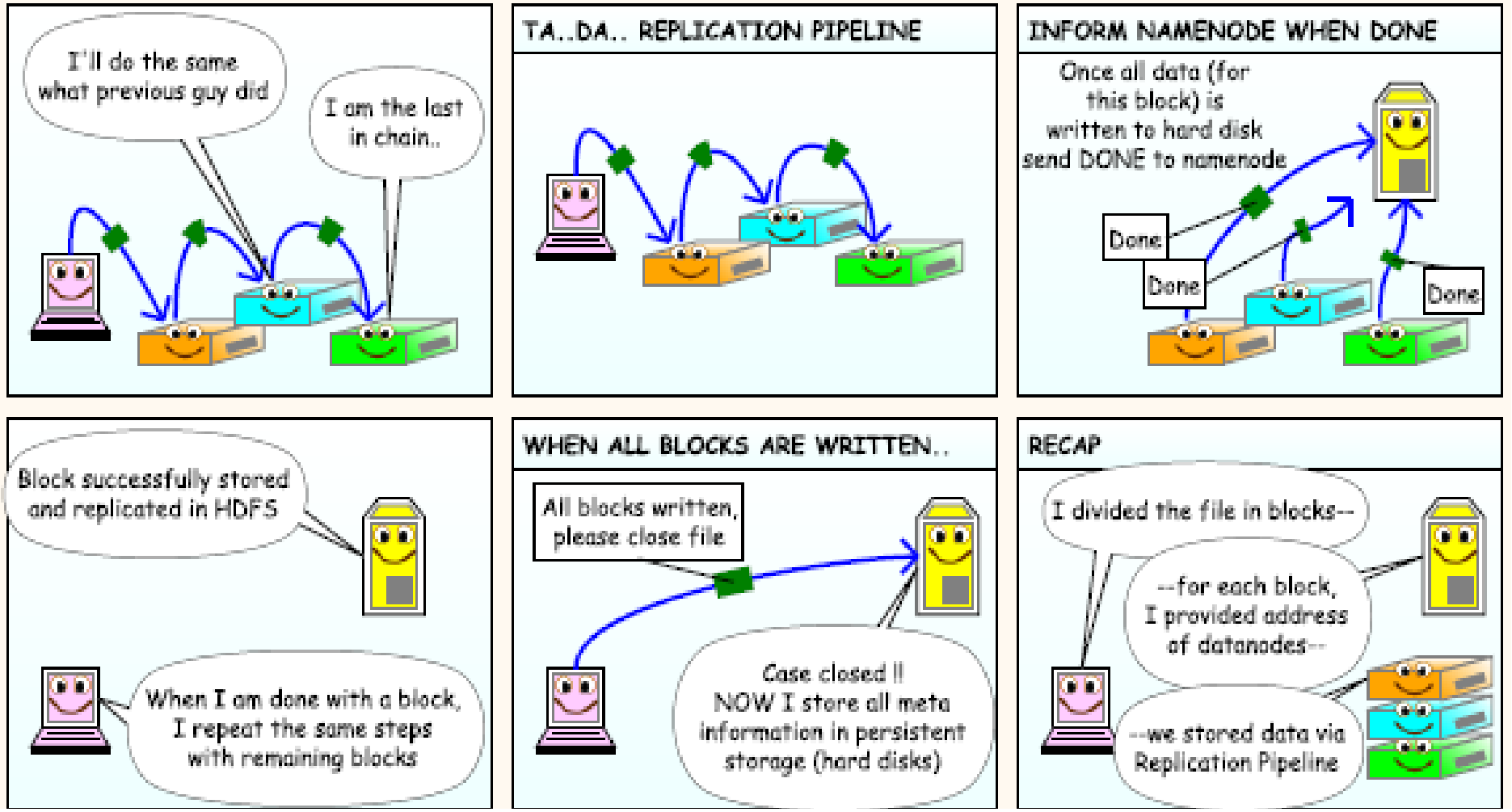


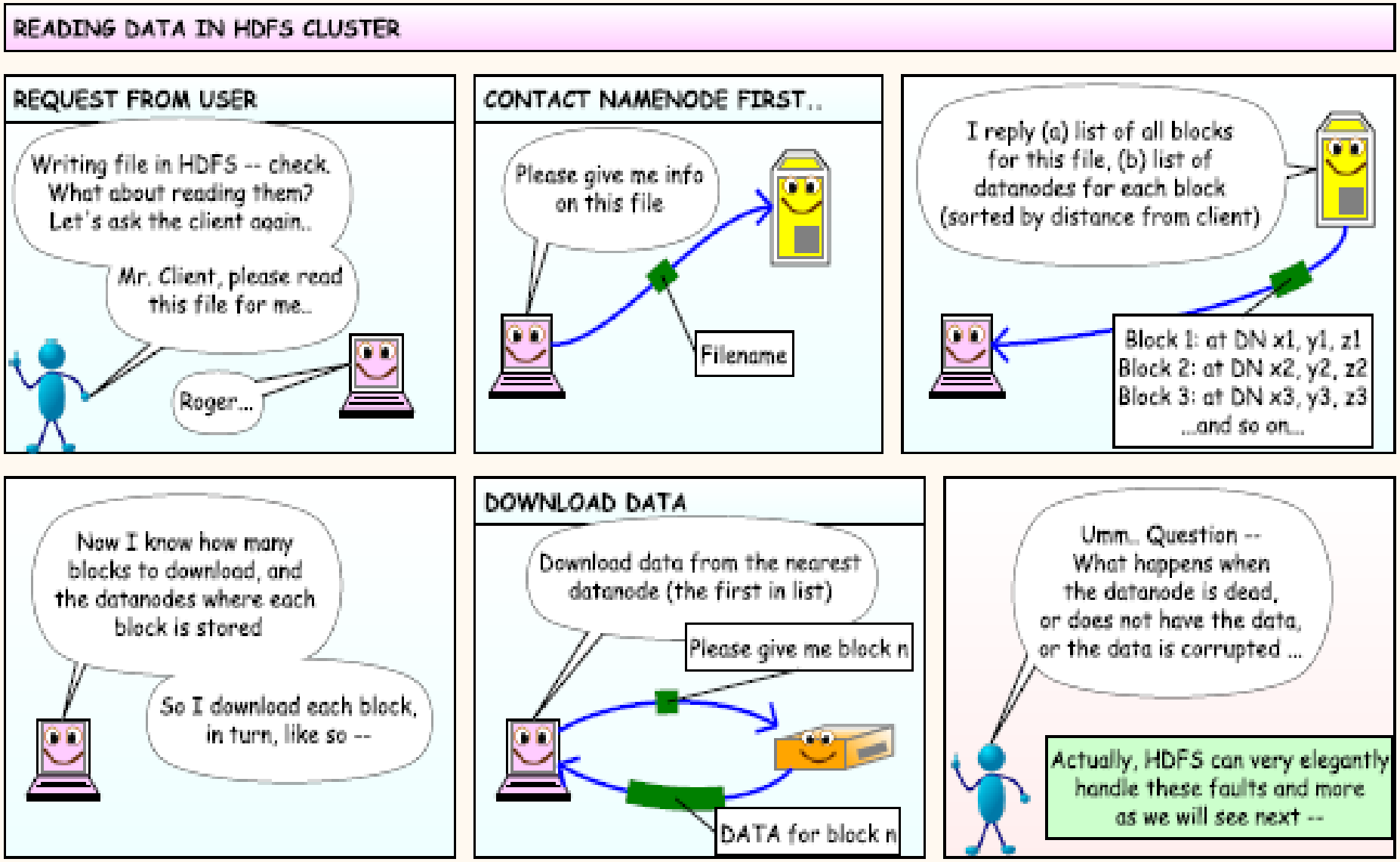


CLIENT STARTS WRITING DATA



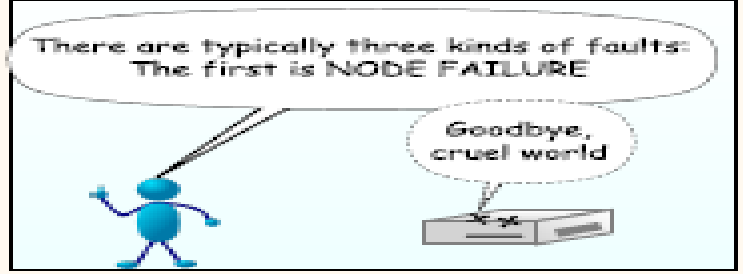



I'll do the same what previous guy did
I am the last in chain..
TA..DA.. REPLICATION PIPELINE
INFORM NAMENODE WHEN DONE
Once all data (for this block) is written to hard disk send DONE to namenode
Done
Done
Done
Block successfully stored and replicated in HDFS
When I am done with a block, I repeat the same steps with remaining blocks
WHEN ALL BLOCKS ARE WRITTEN..
All blocks written, please close file
Case closed !! NOW I store all meta information in persistent storage (hard disks)
RECAP
I divided the file in blocks--
--for each block, I provided address of datanodes--
--we stored data via Replication Pipeline


READING DATA IN HDFS CLUSTER
REQUEST FROM USER
Writing file in HDFS -- check. What about reading them? Let's ask the client again..
Mr. Client, please read this file for me..
Roger...
CONTACT NAMENODE FIRST..
Please give me info on this file
Filename
I reply (a) list of all blocks for this file, (b) list of datanodes for each block (sorted by distance from client)
Block 1: at DN x1, y1, z1
Block 2: at DN x2, y2, z2
Block 3: at DN x3, y3, z3
...and so on...
Now I know how many blocks to download, and the datanodes where each block is stored
So I download each block, in turn, like so --
DOWNLOAD DATA
Download data from the nearest datanode (the first in list)
Please give me block n
DATA for block n
Umm.. Question -- What happens when the datanode is dead, or does not have the data, or the data is corrupted ...
Actually, HDFS can very elegantly handle these faults and more as we will see next --

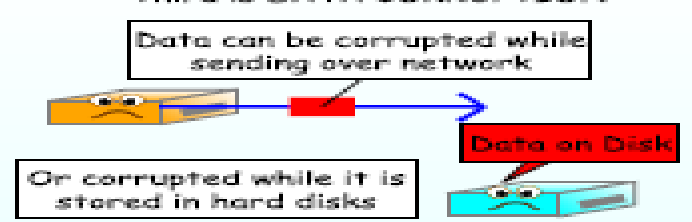
FAULT TOLERANCE IN HDFS. PART I: TYPES OF FAULTS AND THEIR DETECTION

FAULT I: NODE FAILURE



FAULT II: COMMUNICATION FAILURE

FAULT III: DATA CORRUPTION

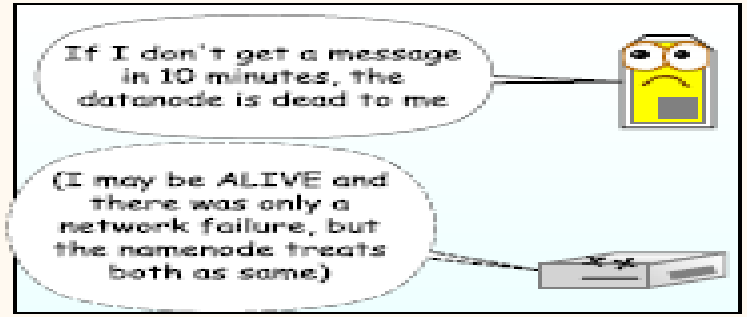


DETECTION #1: NODE FAILURES

NOTE:
If Namenode is dead, the entire cluster is dead! Namenode is the SINGLE POINT OF FAILURE

Instead, let's focus on how datanode failures are detected

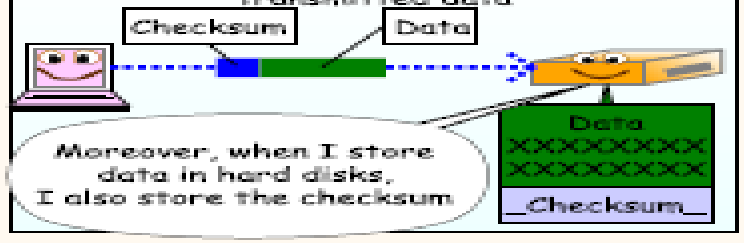
DETECTING DATANODE FAILURE



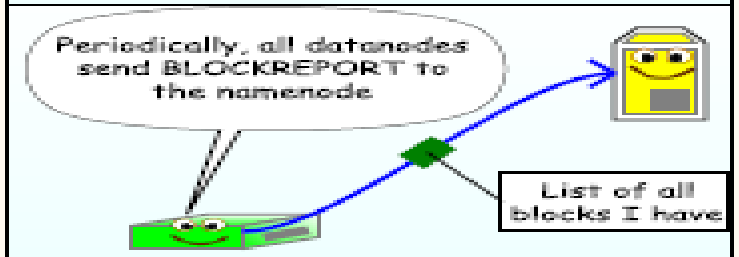
DETECTION #2: NETWORK FAILURES

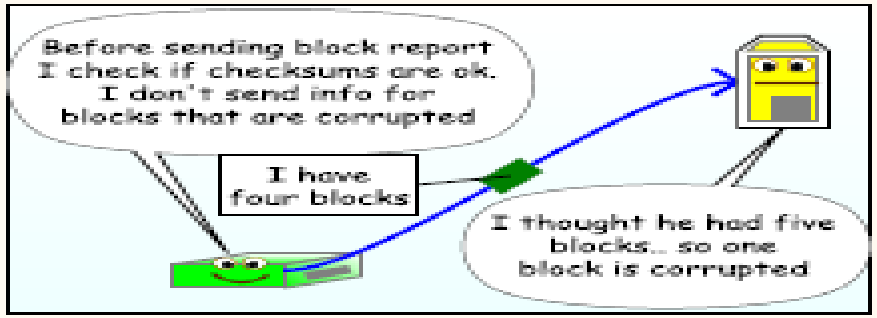
If the ACK is not received (after several retries), the sender assumes that the host is dead, or the network has failed

DETECTION #3: CORRUPTED DATA

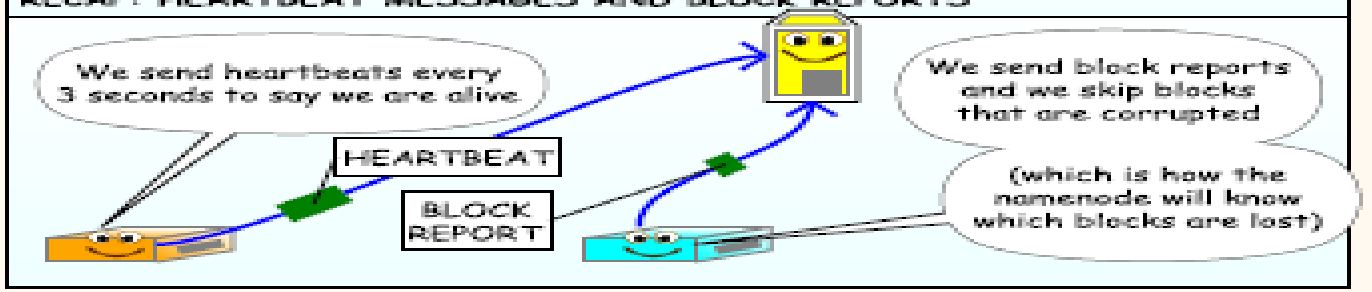


DETECTING CORRUPTED HARD DRIVES





RECAP: HEARTBEAT MESSAGES AND BLOCK REPORTS



HDFS

Good for

- Large Files
- Streaming Data Access

Not so good for

- Small files
- Random Access

INTERACTING WITH HDFS

To the user, HDFS looks just like a UNIX file system.

We can use command line tools, and familiar commands like “ls” and “mkdir” to work with files – we never have to see the “blocks”.

TIME TO MOVE TO VM ....

VM UNIX COMMANDS

`echo hello world`	**Print "hello world" to the standard output.**
`Pwd`	Print the working directory. We're in "~" which we know is the user's home directory, but this will give you the full address: /home/cloudera
`Ls`	List files and folders in the working directory.
`ls -l`	The same, but with details.
`ls -a`	The same, but with hidden files included.
`ls -al`	The same, but with details *and* hidden files.
`ls /`	List files in the top level directory of the file system.
`cd /`	Change directory to "/".
`cd home` `cd cloudera`	From the top level, cd into the "home" directory and then the "cloudera" subdirectory. These are *relative* moves because you didn't enter the full address of either directory.
`mkdir testdir`	Make a directory called "testdir" within the current working directory.
`cd /home/cloudera/testdir`	Move into testdir by providing the *absolute* location of it. You could also have just used "cd testdir".
`cd ~`	Move back to the home directory.