香港城市大學
City University of Hong Kong

Lab 3

HADOOP

香港城市大學
City University of Hong Kong

- "Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**"

Hortonworks

HADOOP?

- Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**

  - It is an open source software.

HADOOP?

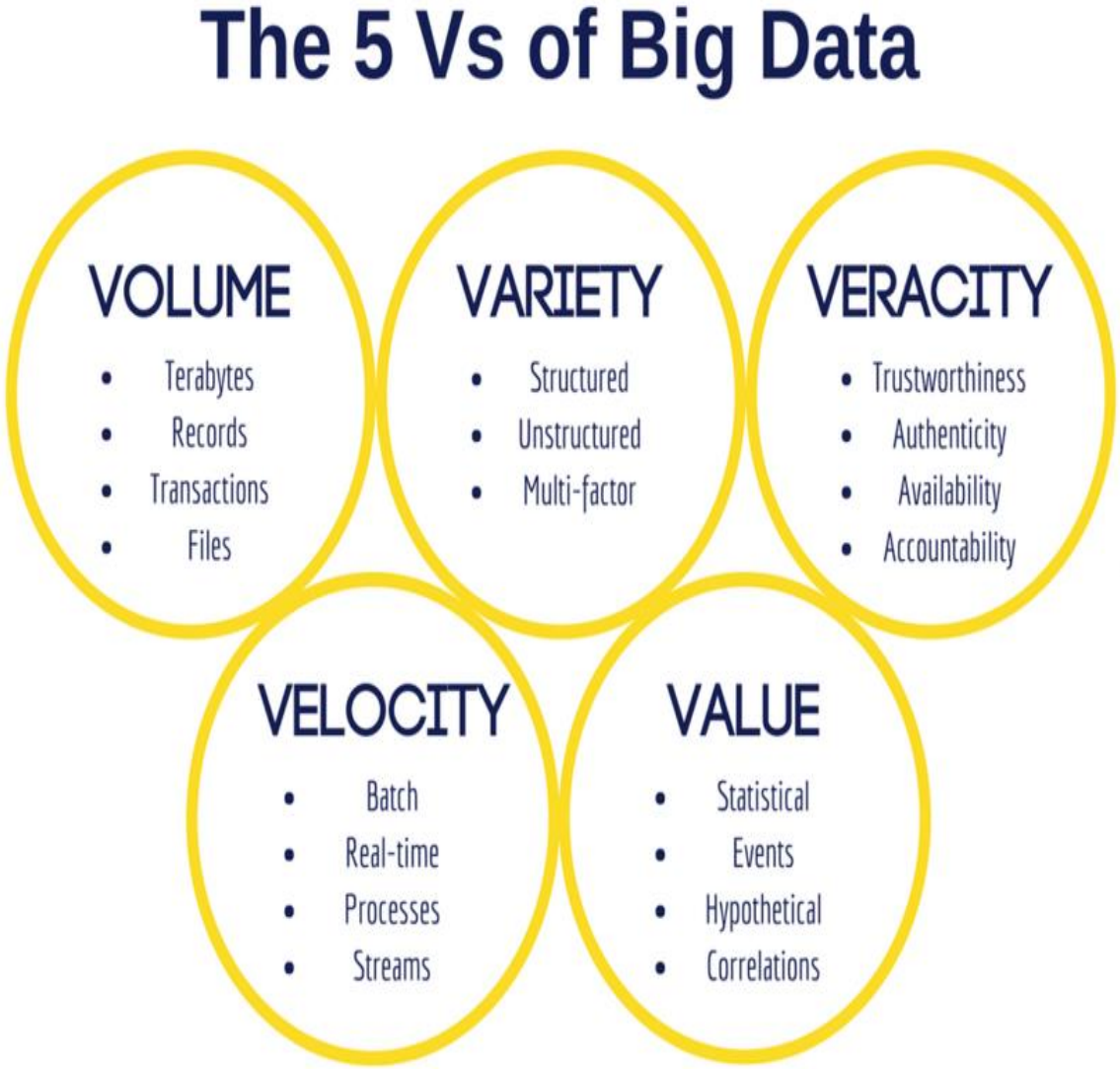
- Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**

  - More PCs can be added to the cluster, and their hard drive simply becomes part of the storage

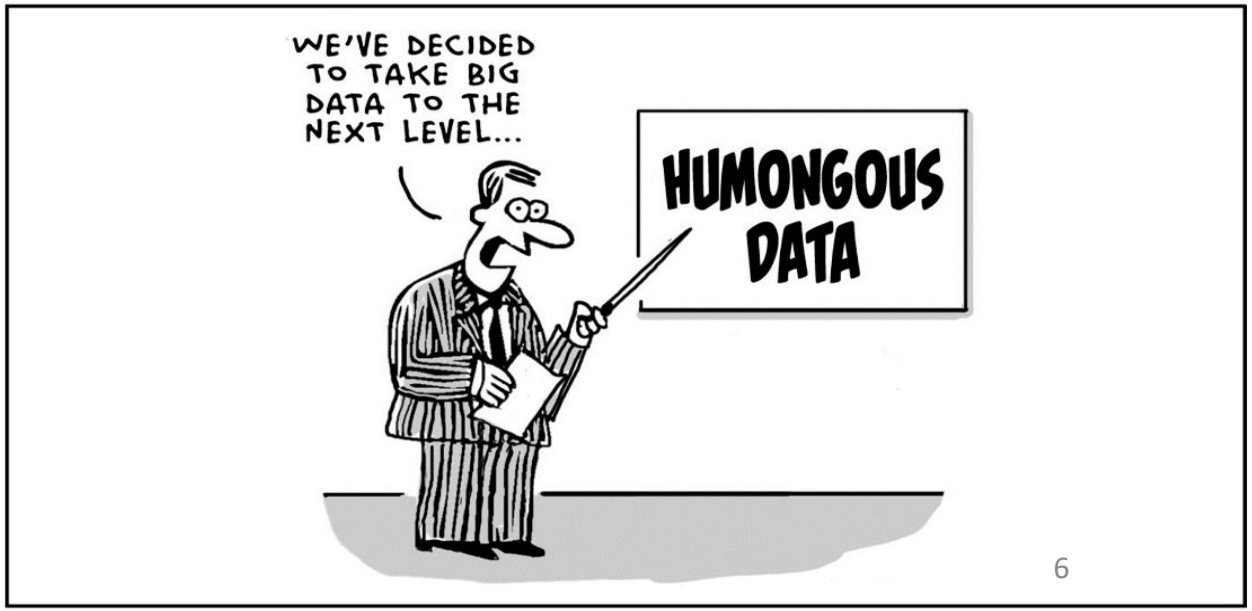
HADOOP?

- Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**

  - Data transformation and aggregation is done in a parallel manner. i.e. all CPUs of the PCs in the cluster work on the instruction for faster processing



HADOOP is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**



HADOOP?

- Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**
  - As many computers as possible can be added as the need arises.

HADOOP?

- Hadoop is an open source **software platform** for **distributed storage** and **distributed processing** of **very large datasets** on **computer clusters** built from **commodity hardware**

  - The hardware is readily available and can be gotten off the shelf.

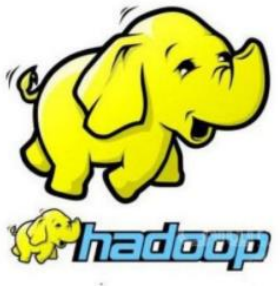
HISTORY

CityU 香港城市大學 City University of Hong Kong

- 2003 – 2004, Google published GFS (Google File System) which inspired distributed data storage.

- Yahoo while building open source web search engine “Nutch” came up with Hadoop in 2006 with Doug Cutting and Tom White as the heads of the team.

- Hadoop is the name of Doug Cutting’s son’s yellow stuffed elephant.



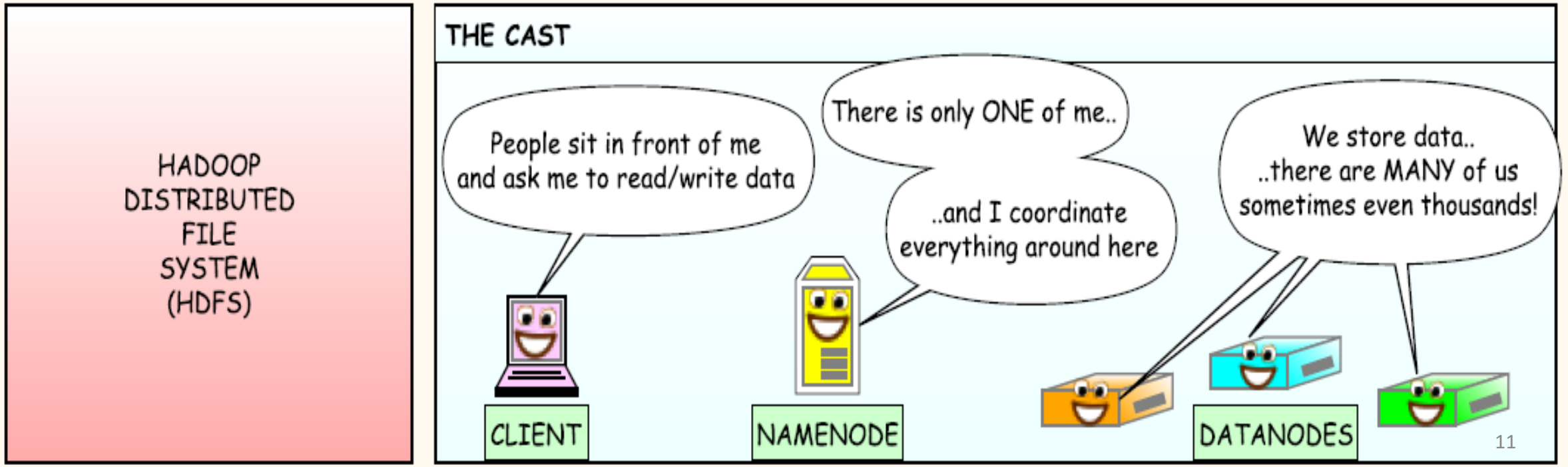
Why Hadoop?

香港城市大學
City University of Hong Kong

- Data is too big and generated rapidly in different formats.
- Processing time is faster.
- Vertical scaling is not fit for disk seek time
- Redundancy
- Horizontal scaling is linear
- Efficiency

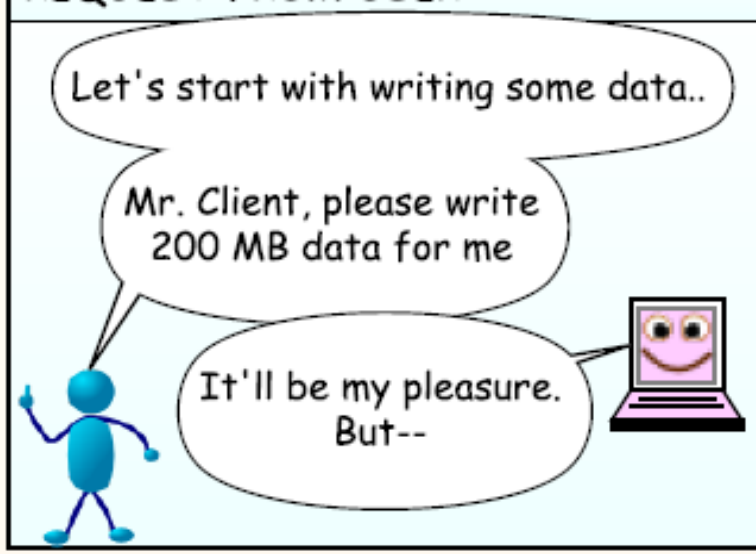
How it works

Writing/reading data in HDFS cluster
Fault tolerance in HDFS :
Detecting and Handling Failures.

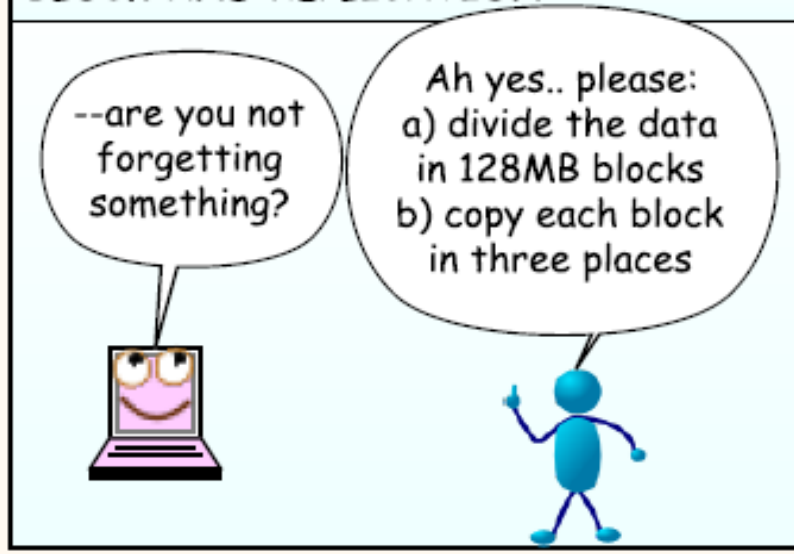


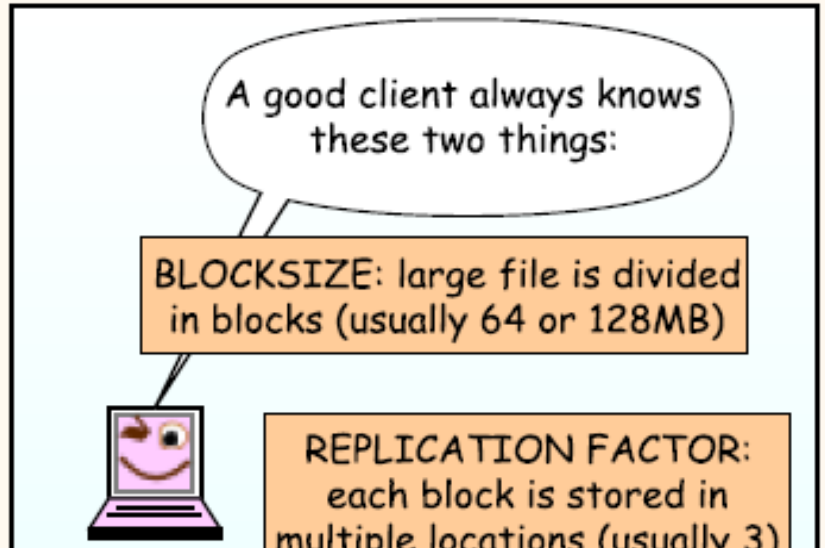
WRITING DATA IN HDFS CLUSTER

REQUEST FROM USER

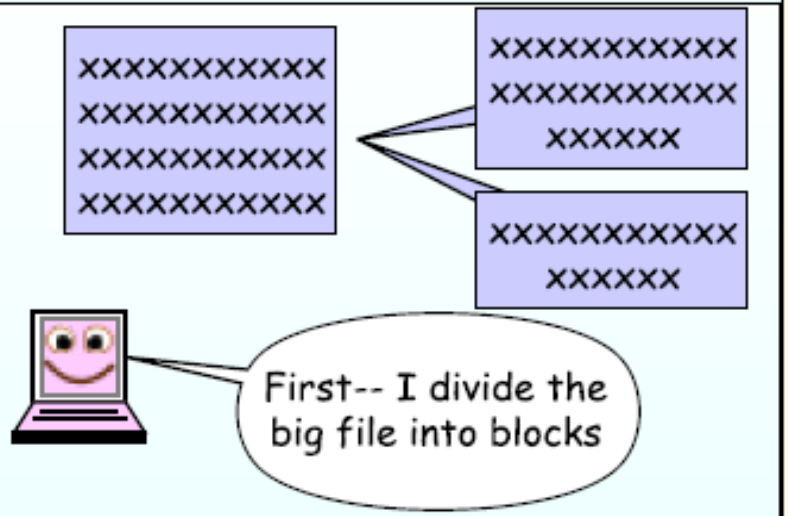


BLOCK AND REPLICATION

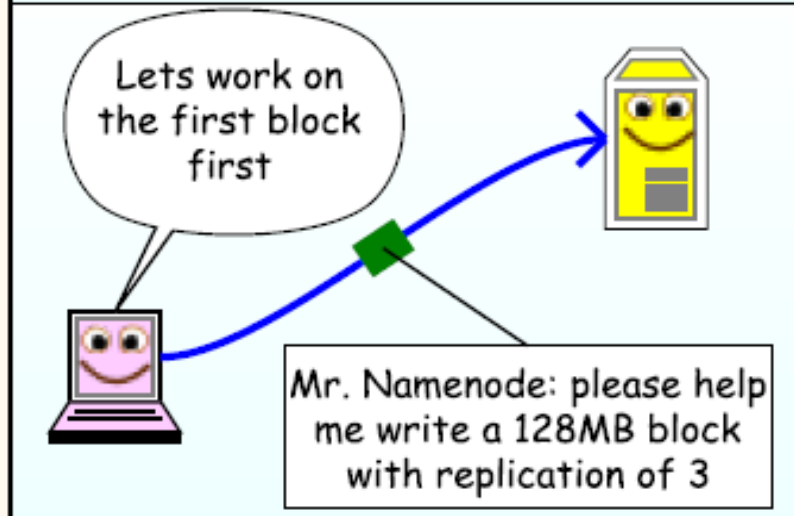




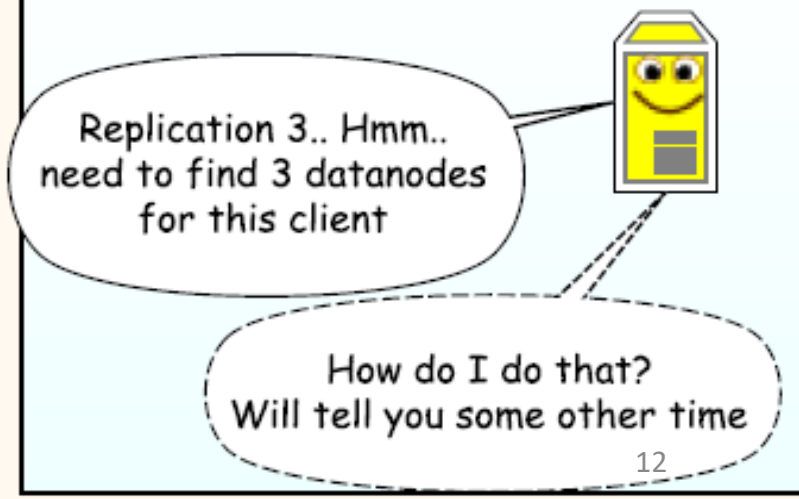
DIVIDE FILE INTO BLOCKS



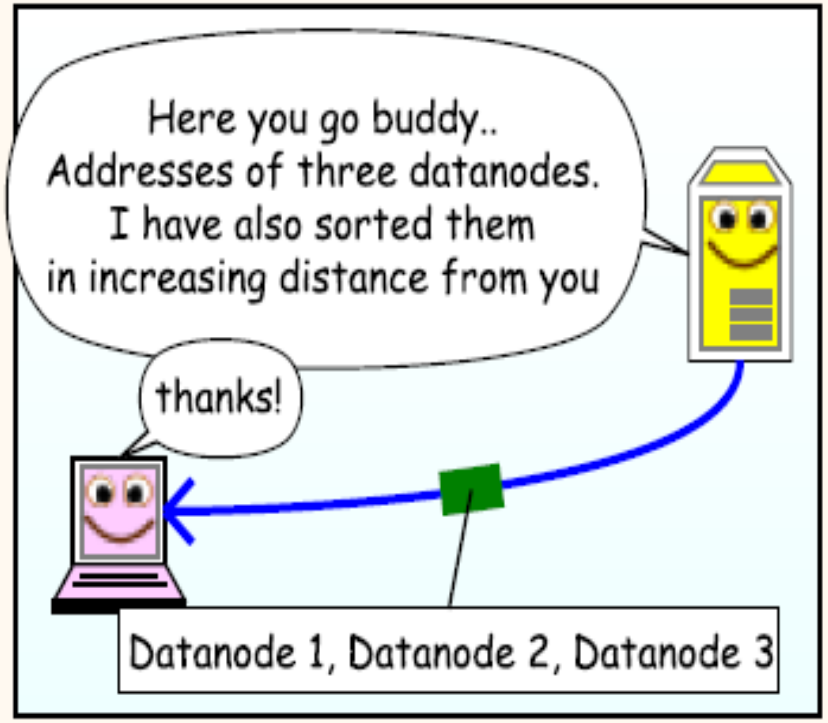
ASK NAMENODE

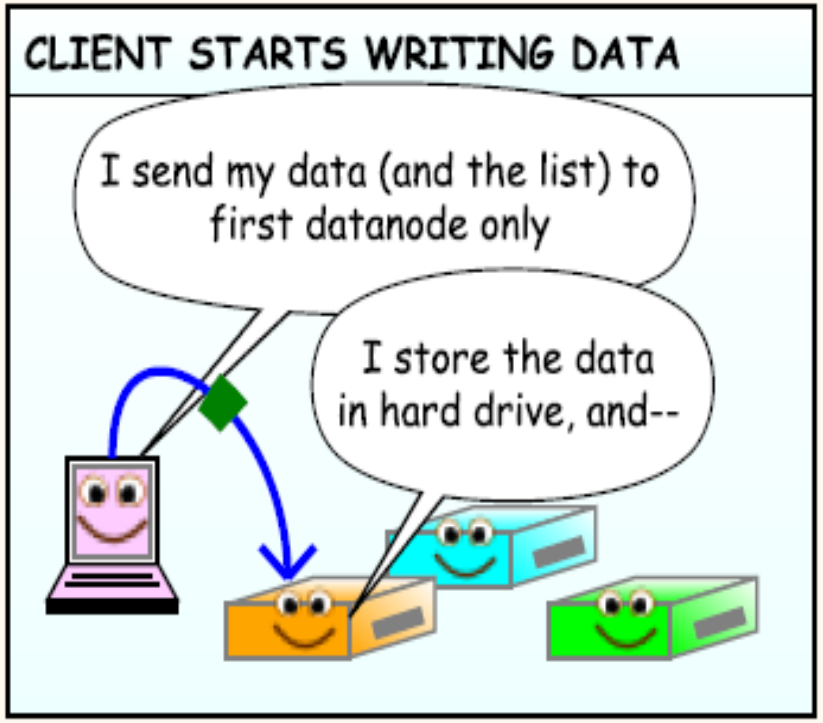


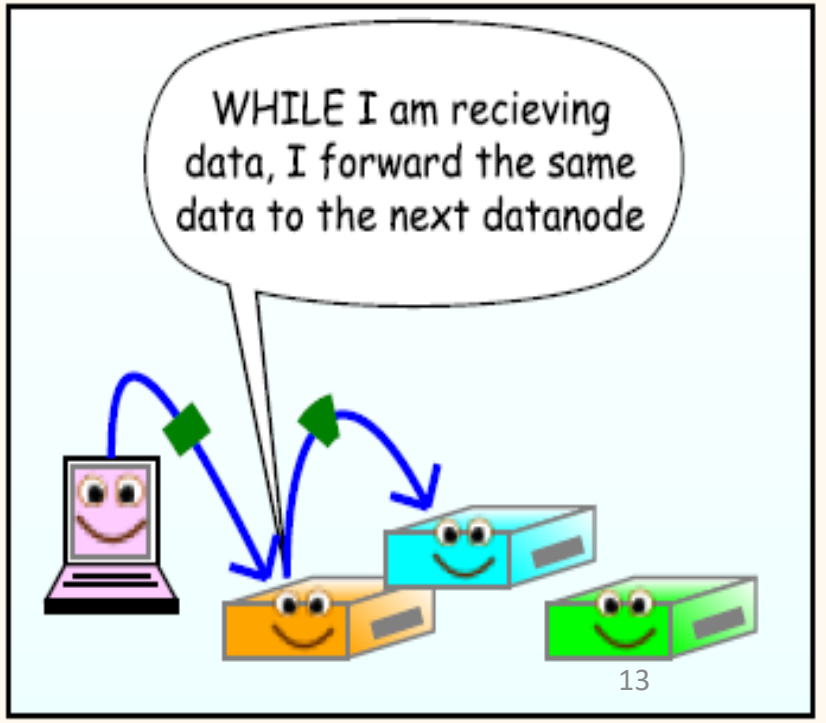
NAMENODE ASSIGNS DATANODES

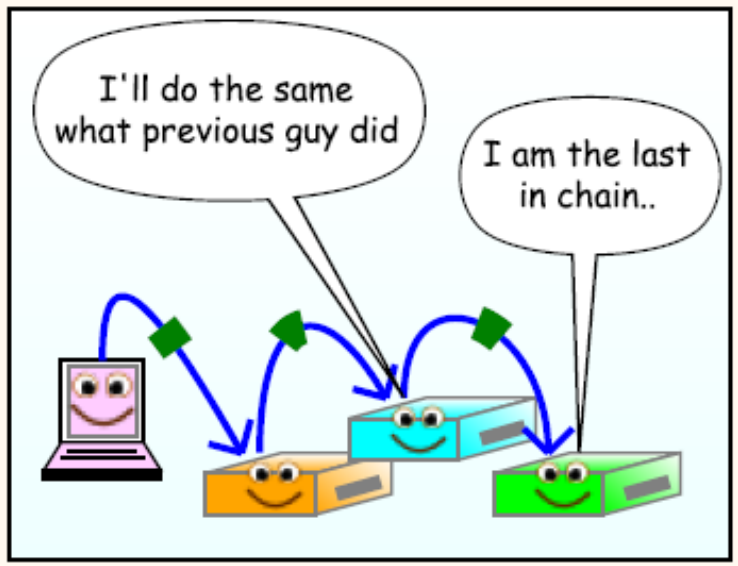


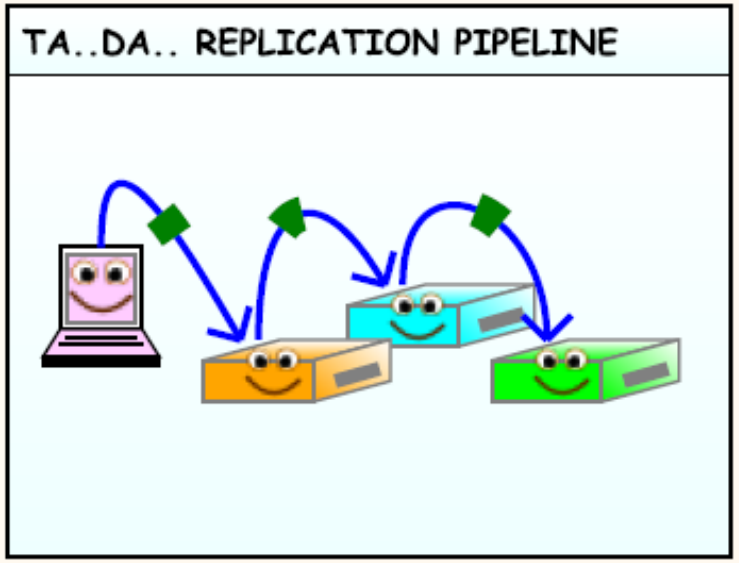
How it works

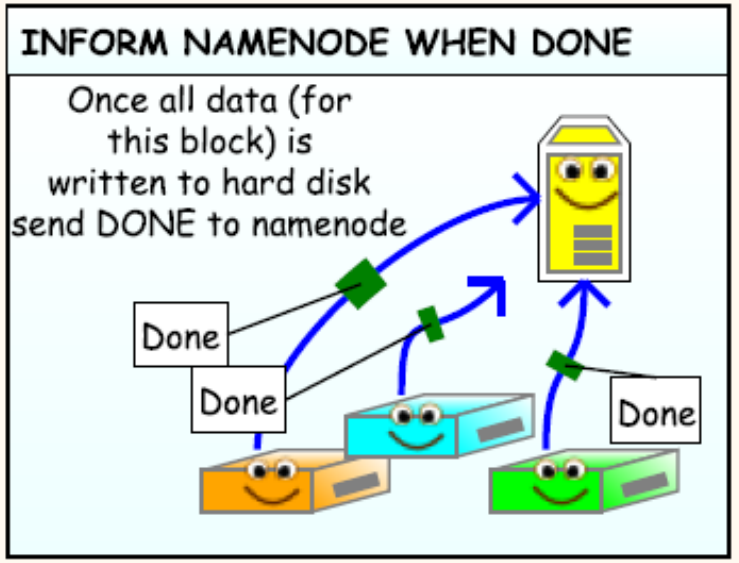

Here you go buddy..
Addresses of three datanodes.
I have also sorted them
in increasing distance from you
thanks!
Datanode 1, Datanode 2, Datanode 3

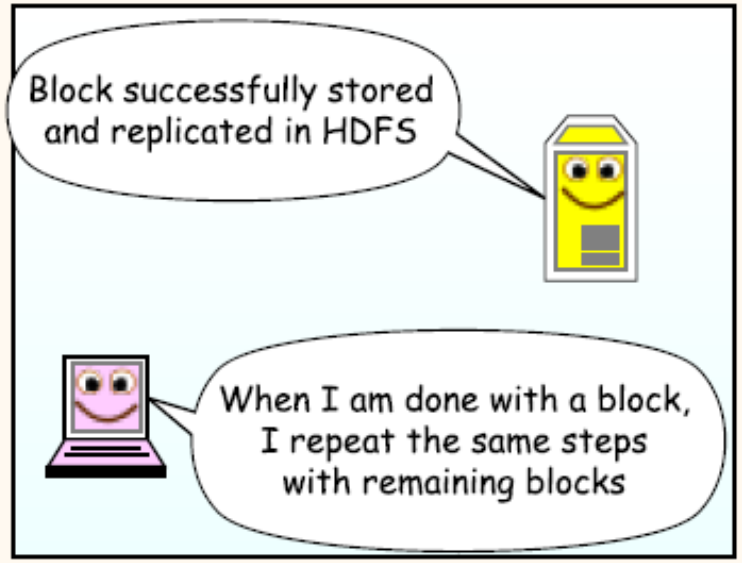

CLIENT STARTS WRITING DATA
I send my data (and the list) to
first datanode only
I store the data
in hard drive, and--

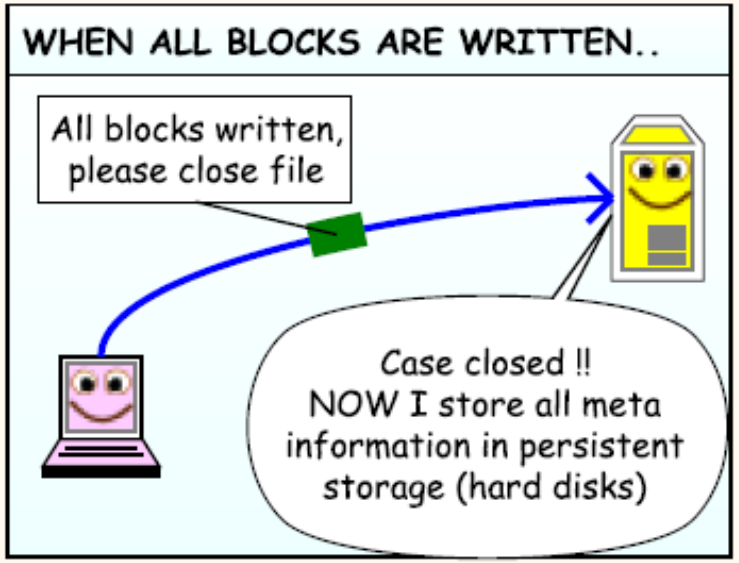

WHILE I am recieving
data, I forward the same
data to the next datanode

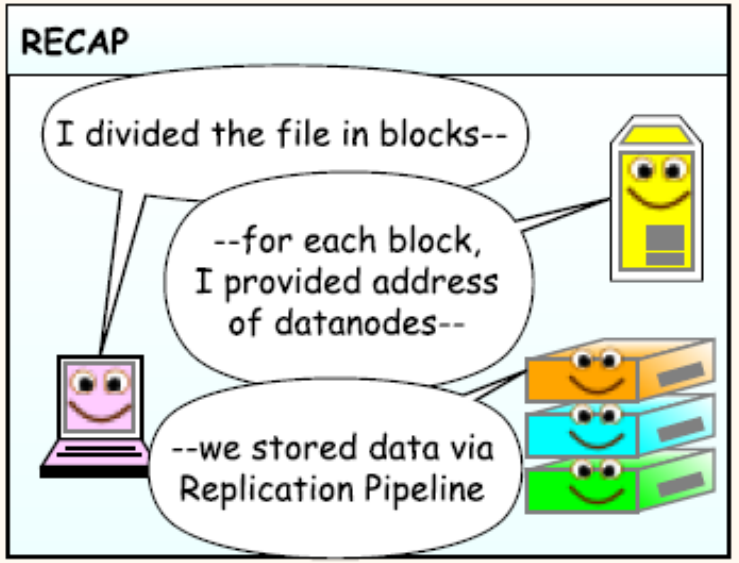

I'll do the same what previous guy did
I am the last in chain..


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

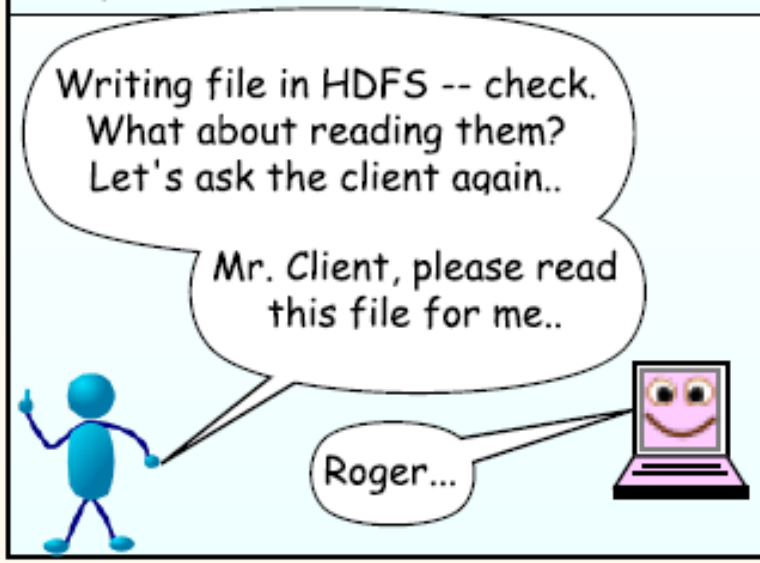

Block successfully stored and replicated in HDFS
When I am done with a block, I repeat the same steps with remaining blocks


WHEN ALL BLOCKS ARE WRITTEN..
All blocks written, please close file
Case closed !! NOW I store all meta information in persistent storage (hard disks)

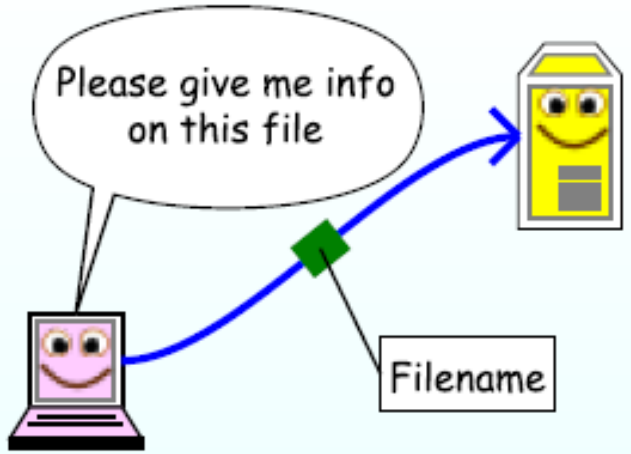

RECAP
I divided the file in blocks--
--for each block, I provided address of datanodes--
--we stored data via Replication Pipeline

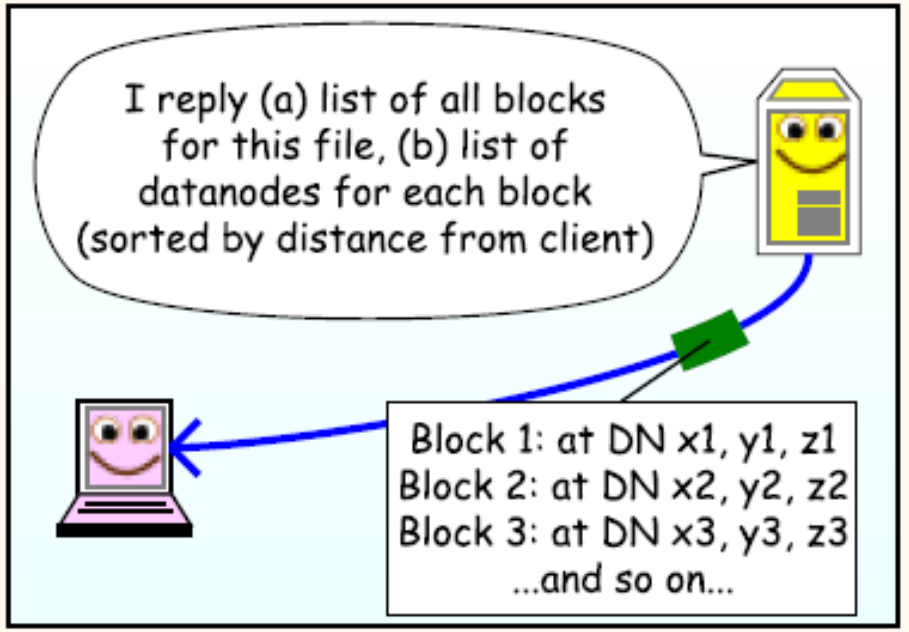
READING DATA IN HDFS CLUSTER

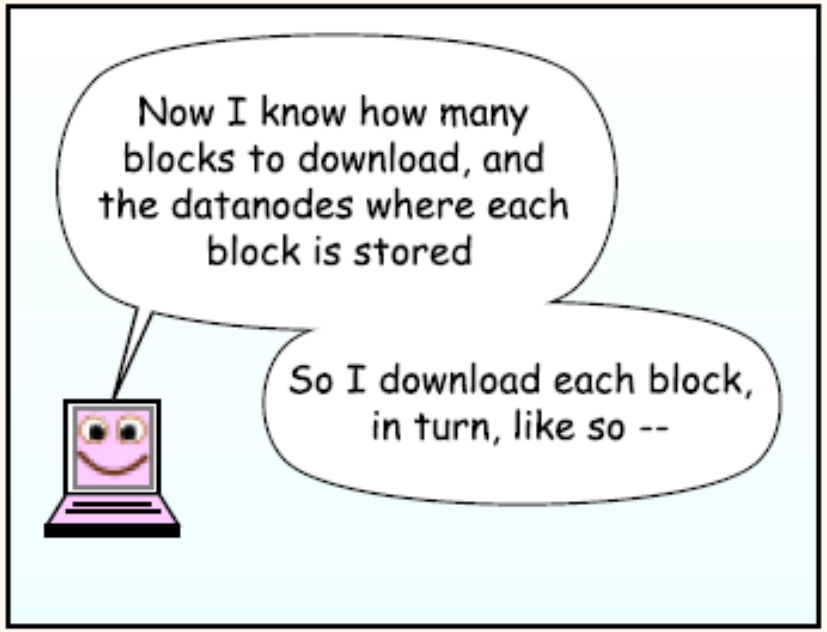
REQUEST FROM USER



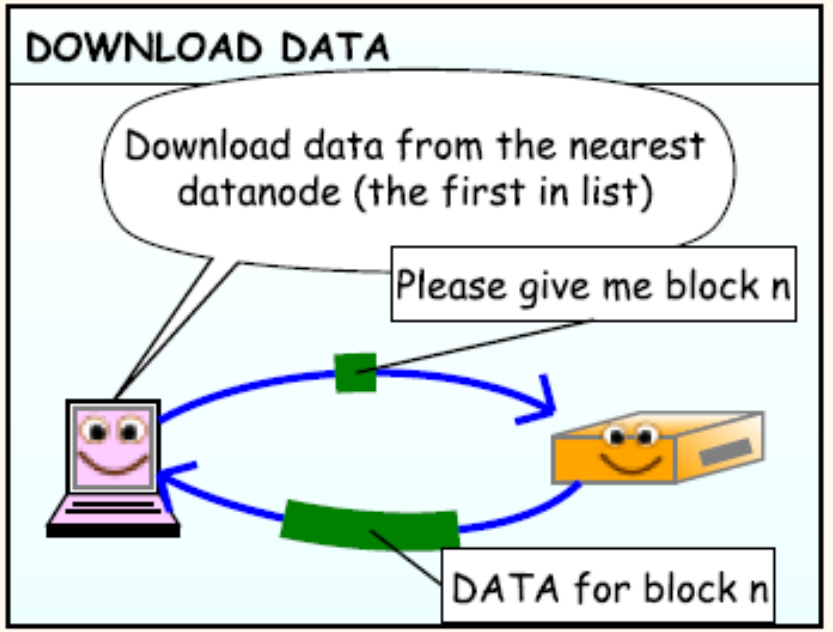
CONTACT NAMENODE FIRST..

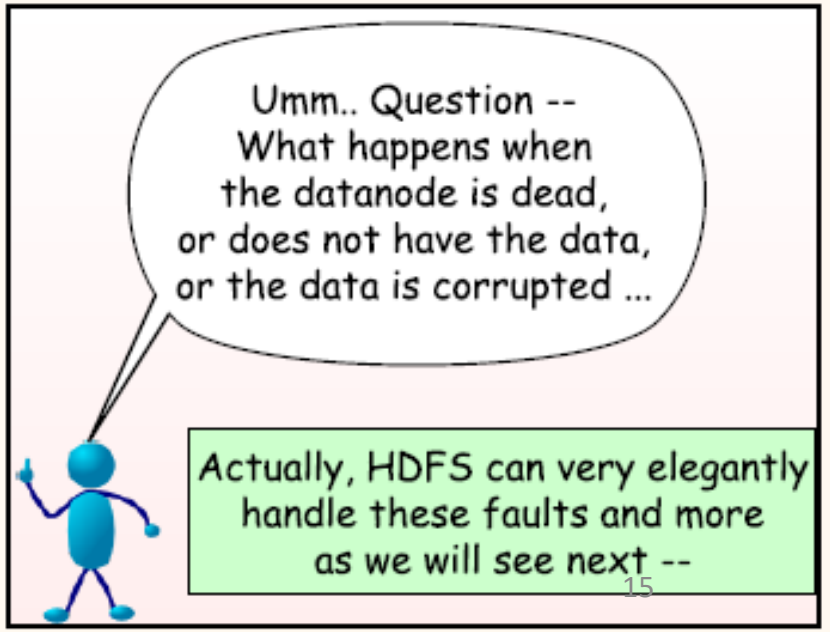




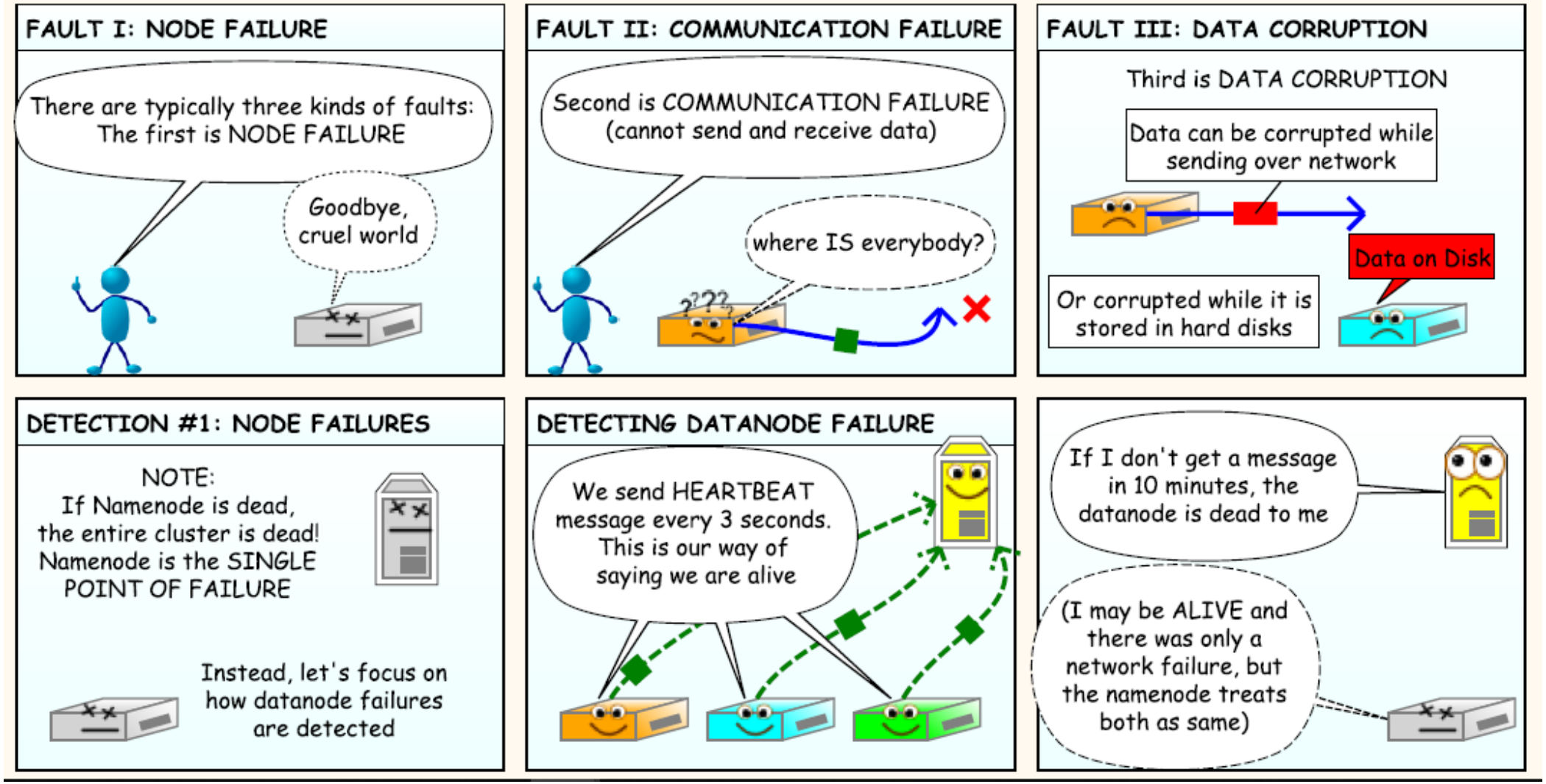


DOWNLOAD DATA



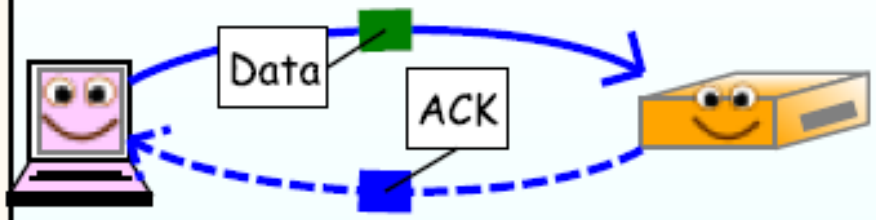


FAULT TOLERANCE IN HDFS. PART I: TYPES OF FAULTS AND THEIR DETECTION



DETECTION #2: NETWORK FAILURES

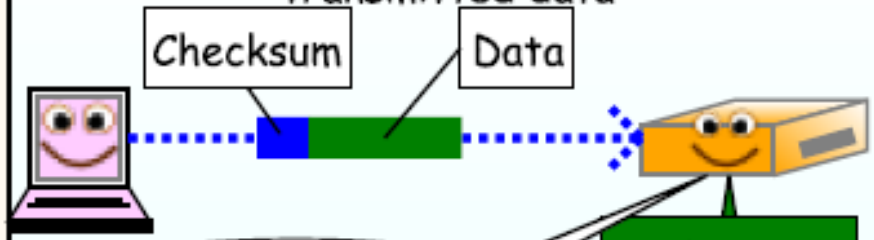
Whenever data is sent,
an ACK is replied by the reciever



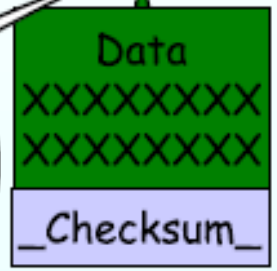
If the ACK is not received (after several retries), the sender assumes that the host is dead, or the network has failed

DETECTION #3: CORRUPTED DATA

Checksum is sent along with transmitted data

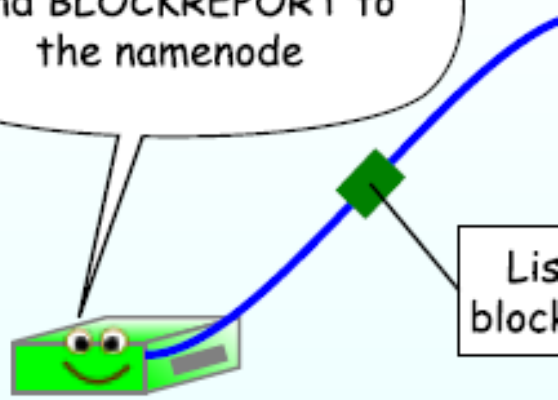


Moreover, when I store data in hard disks, I also store the checksum



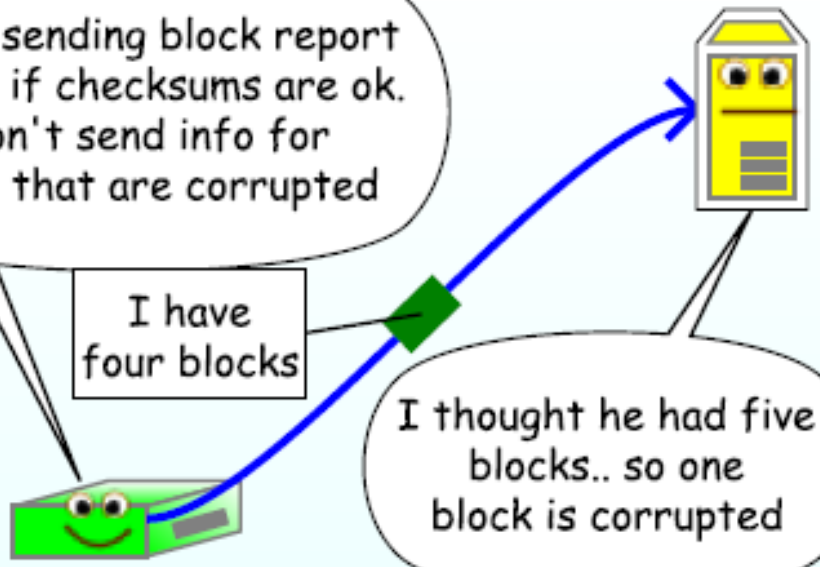
DETECTING CORRUPTED HARD

Periodically, all datanodes send BLOCKREPORT to the namenode



Before sending block report I check if checksums are ok. I don't send info for blocks that are corrupted

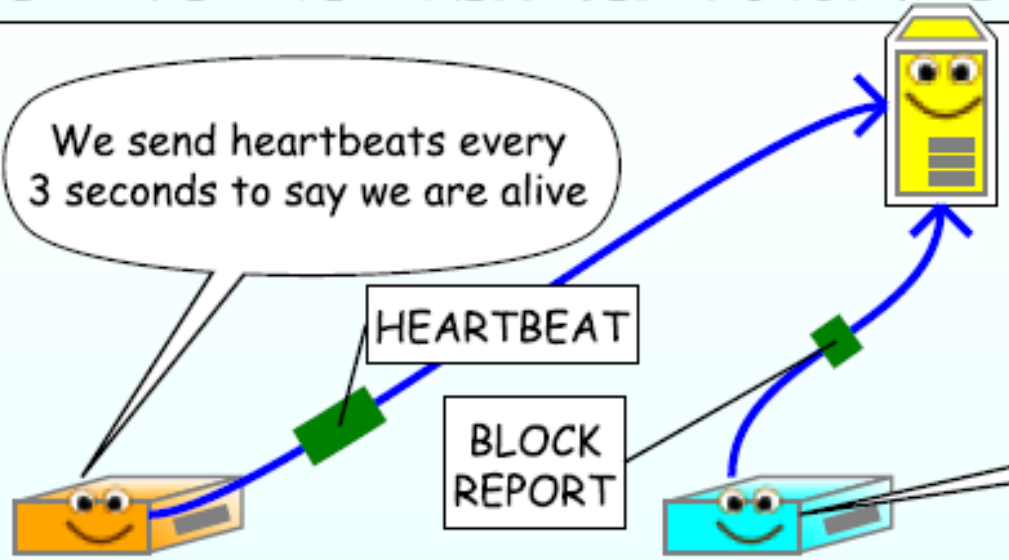
I have four blocks



I thought he had five blocks.. so one block is corrupted

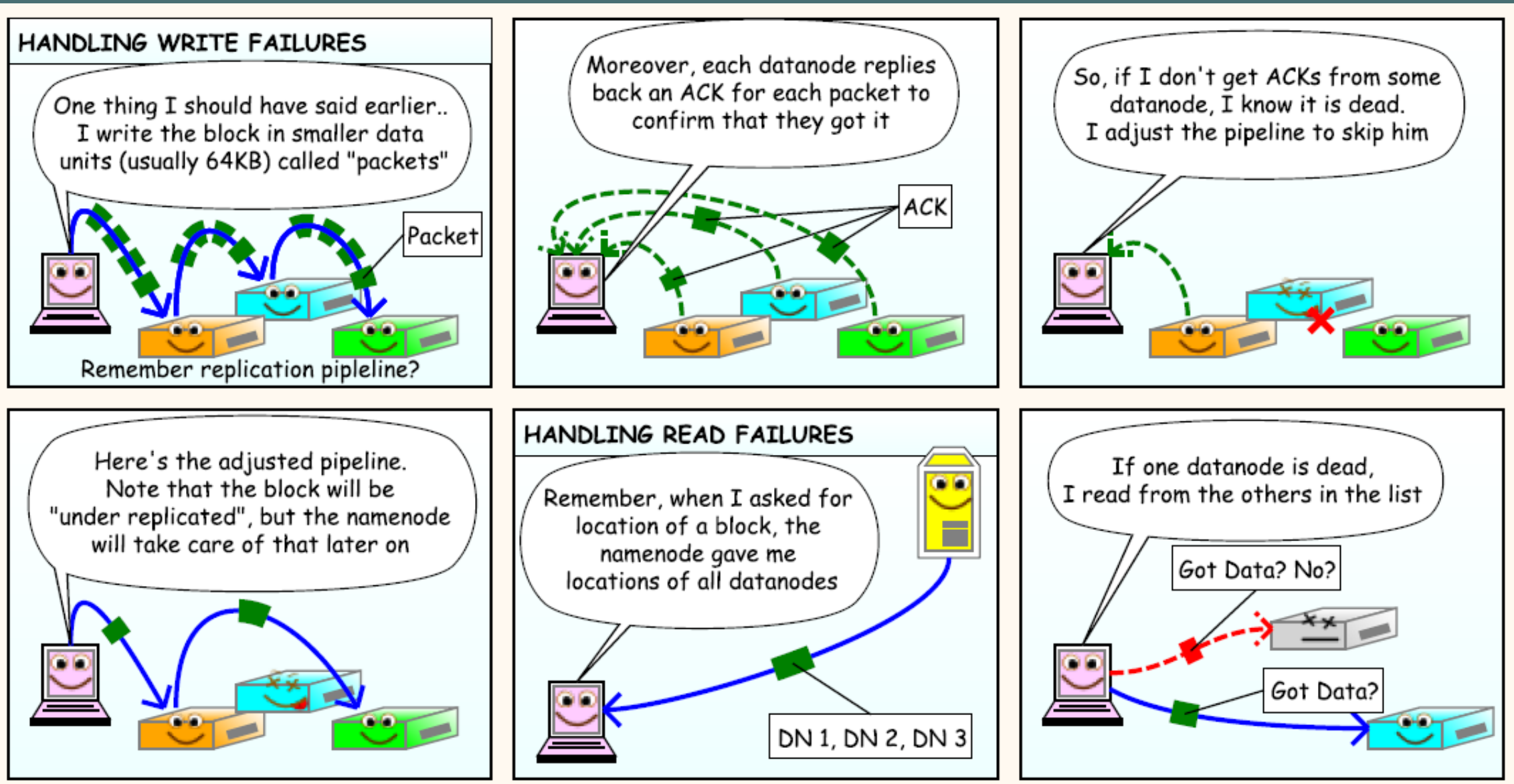
RECAP: HEARTBEAT MESSAGES AND BLOCK REPORTS

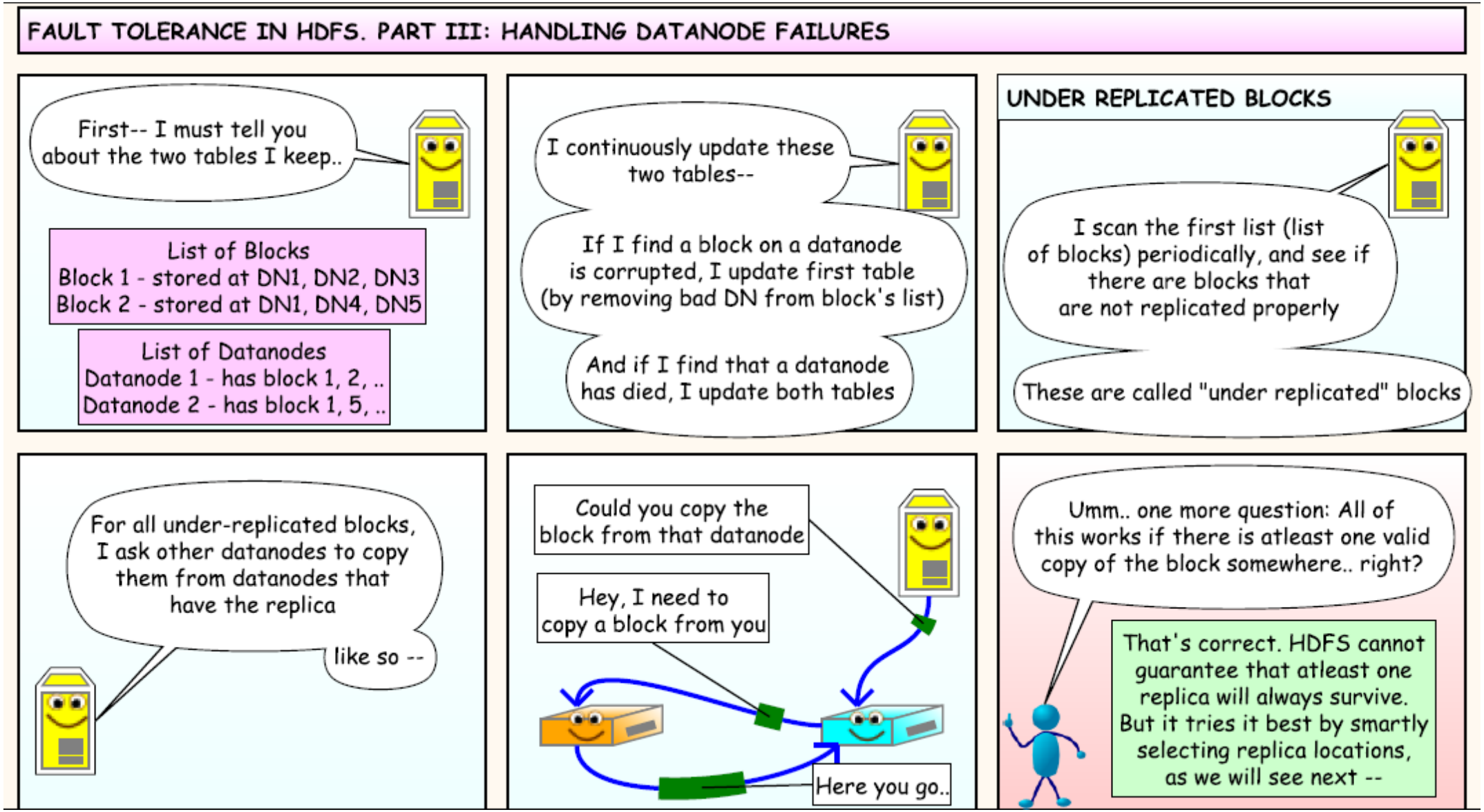
We send heartbeats every 3 seconds to say we are alive



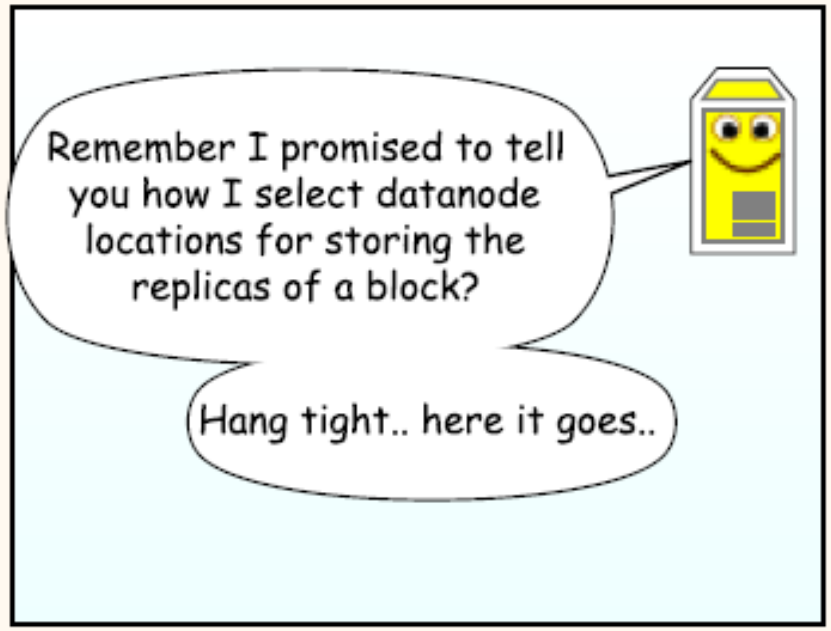
We send block re
and we skip blo
that are corrup

(which is ho
namenode wil
which blocks a


HANDLING WRITE FAILURES
One thing I should have said earlier.. I write the block in smaller data units (usually 64KB) called "packets"
Packet
Remember replication pipeline?
Moreover, each datanode replies back an ACK for each packet to confirm that they got it
ACK
So, if I don't get ACKs from some datanode, I know it is dead. I adjust the pipeline to skip him
Here's the adjusted pipeline. Note that the block will be "under replicated", but the namenode will take care of that later on
HANDLING READ FAILURES
Remember, when I asked for location of a block, the namenode gave me locations of all datanodes
DN 1, DN 2, DN 3
If one datanode is dead, I read from the others in the list
Got Data? No?
Got Data?


FAULT TOLERANCE IN HDFS. PART III: HANDLING DATANODE FAILURES
First-- I must tell you about the two tables I keep..
List of Blocks
Block 1 - stored at DN1, DN2, DN3
Block 2 - stored at DN1, DN4, DN5
List of Datanodes
Datanode 1 - has block 1, 2, ..
Datanode 2 - has block 1, 5, ..
I continuously update these two tables--
If I find a block on a datanode is corrupted, I update first table (by removing bad DN from block's list)
And if I find that a datanode has died, I update both tables
UNDER REPLICATED BLOCKS
I scan the first list (list of blocks) periodically, and see if there are blocks that are not replicated properly
These are called "under replicated" blocks
For all under-replicated blocks, I ask other datanodes to copy them from datanodes that have the replica
like so --
Could you copy the block from that datanode
Hey, I need to copy a block from you
Here you go..
Umm.. one more question: All of this works if there is atleast one valid copy of the block somewhere.. right?
That's correct. HDFS cannot guarantee that atleast one replica will always survive. But it tries it best by smartly selecting replica locations, as we will see next --

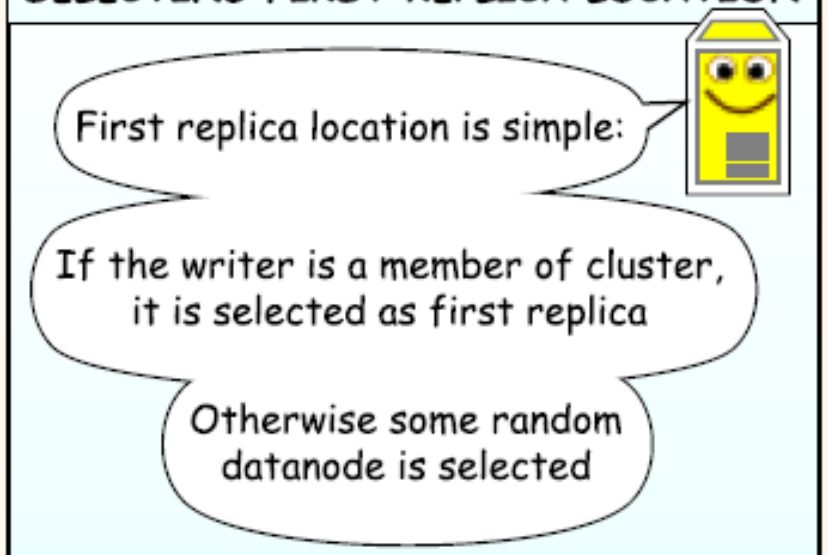
REPLICA PLACEMENT STRATEGY



RACKS AND DATANODES

The cluster is divided into RACKS
Each rack has multiple datanodes

SELECTING FIRST REPLICA LOCATION

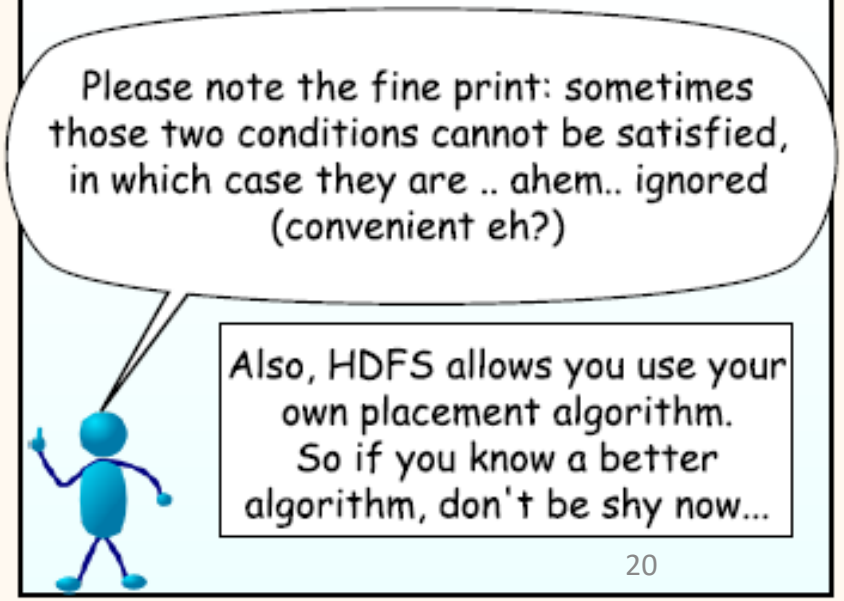


NEXT TWO REPLICA LOCATIONS

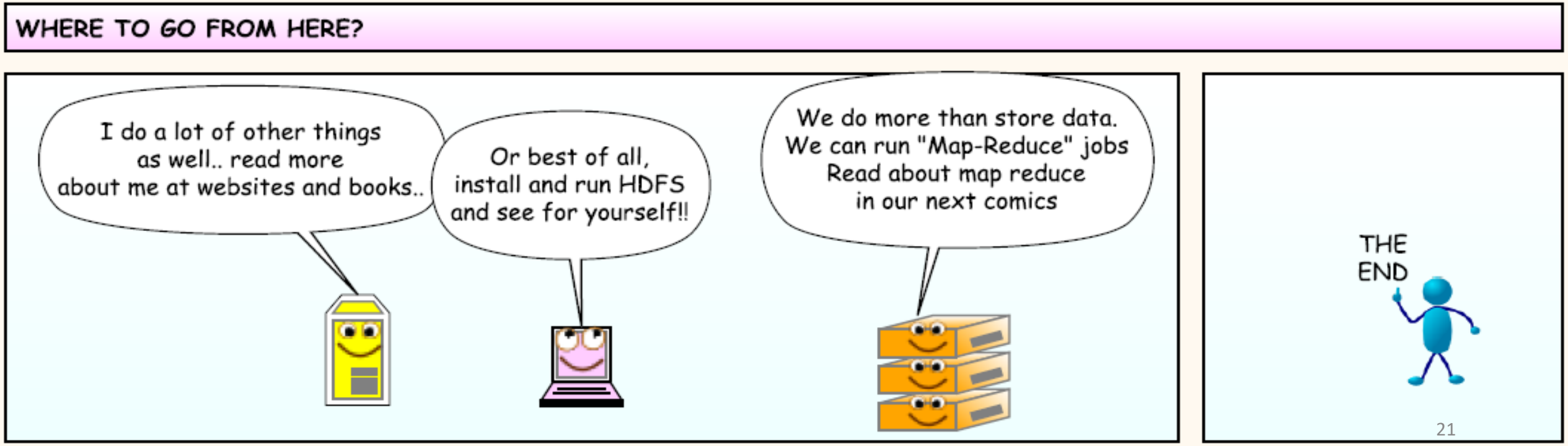
Pick a different rack than first replica's
Select two different datanode on that rack

SUBSEQUENT REPLICA LOCATIONS

Pick any random datanode,
if it satisfies these two conditions:

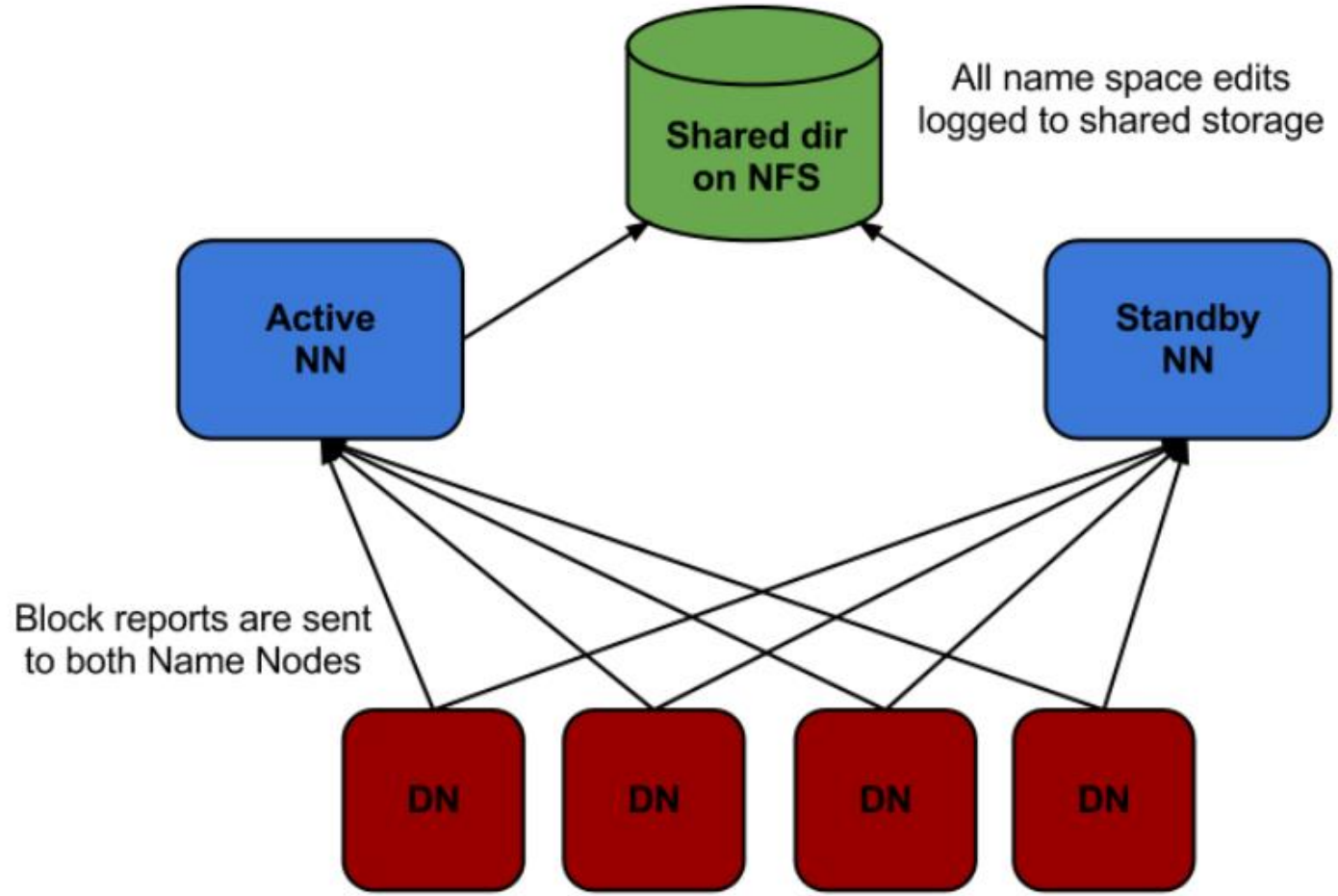


How it works



How it works

- The new version of Hadoop runs two redundant Namenodes to handle the Single Point of Failure.
- Hence, the two namenodes receive information from the datanodes concurrently.
- If the client fails to reach a namenode, it can reach another namenode.



Source: https://blog.cloudera.com/

Hadoop Ecosystem

Hadoop Modules

- Hadoop Distributed File System (HDFS)
- Yet Another Resource Negotiator (YARN)
- -A framework for job scheduling and cluster resource management.
- MapReduce

CityU
香港城市大學
City University of Hong Kong

HDFS

Hadoop Distributed File System

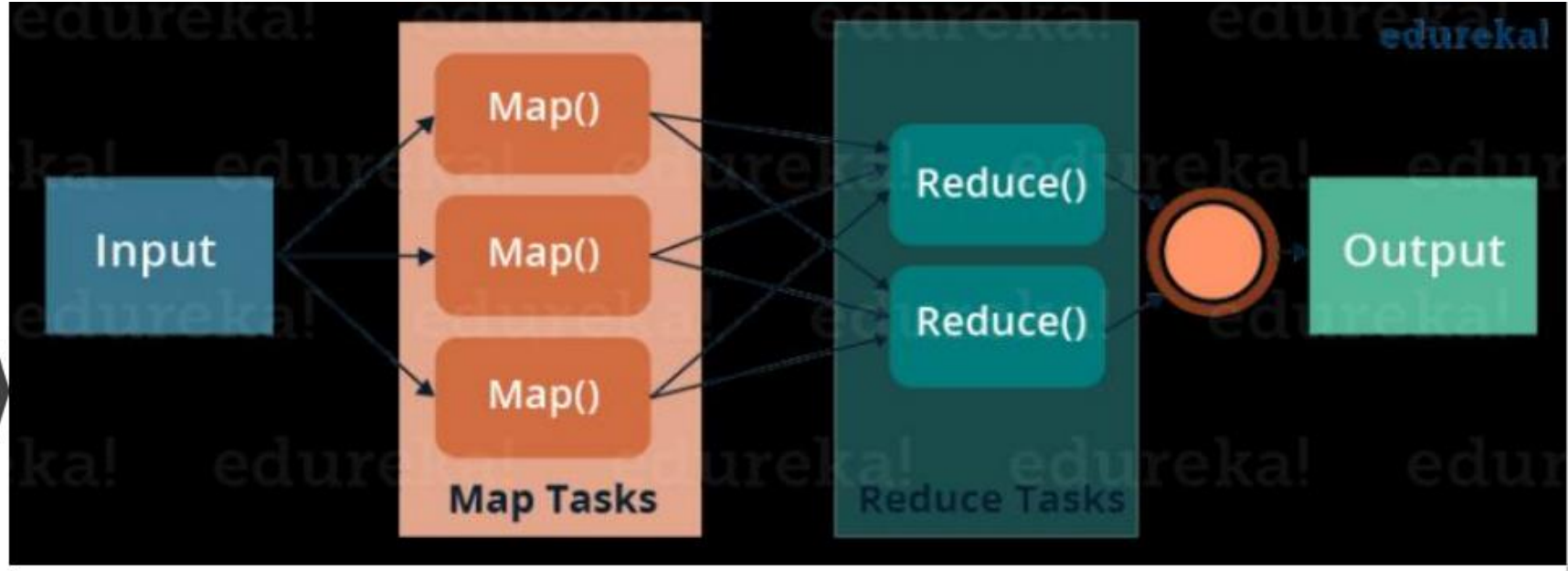
- A distributed file system that provides high-throughput access to application data.
- It is the system that allows to distribute the storage of big data across the cluster of computer.
- All hard drive in the cluster look like one giant system and maintains redundant copies of the data

MAPREDUCE

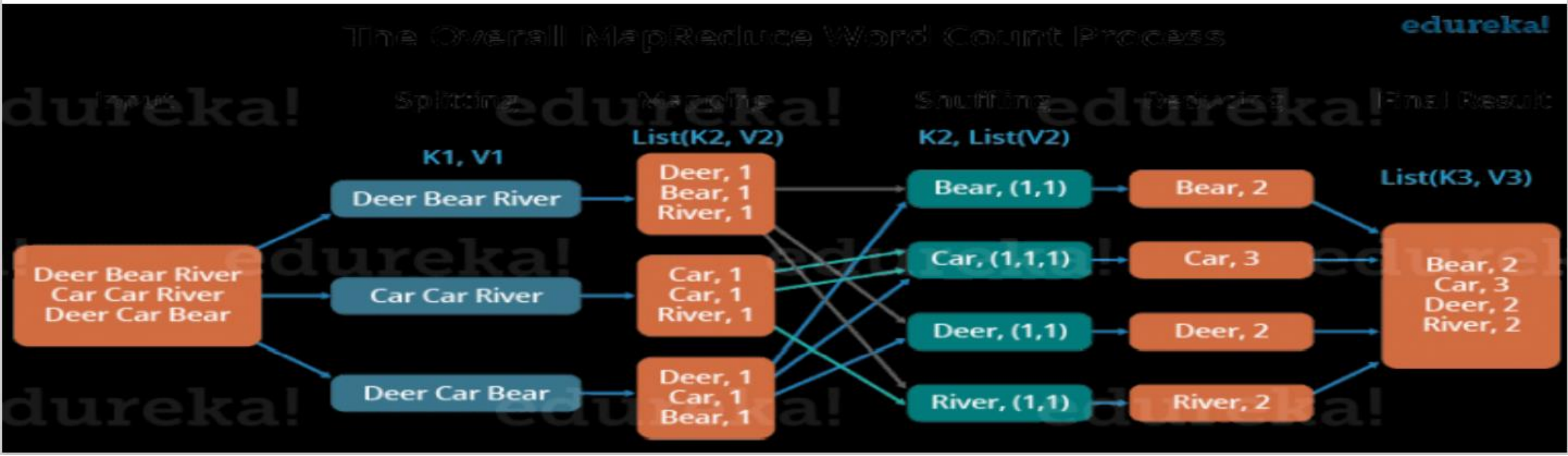
香港城市大學
City University of Hong Kong

- A YARN-based system for parallel processing of large data sets.
- This is the programming model that allows data processing across the entire cluster.
- Mappers: creates the input data, usually a file or directory. The input data is processed to create as few or as many outputs as needed, which is afterward passed to the reducers
- Reducers: process data from the mapper into something usable. Output from the reducer is saved in the HDFS.
- It’s a phase known as “shuffle and sort phase”

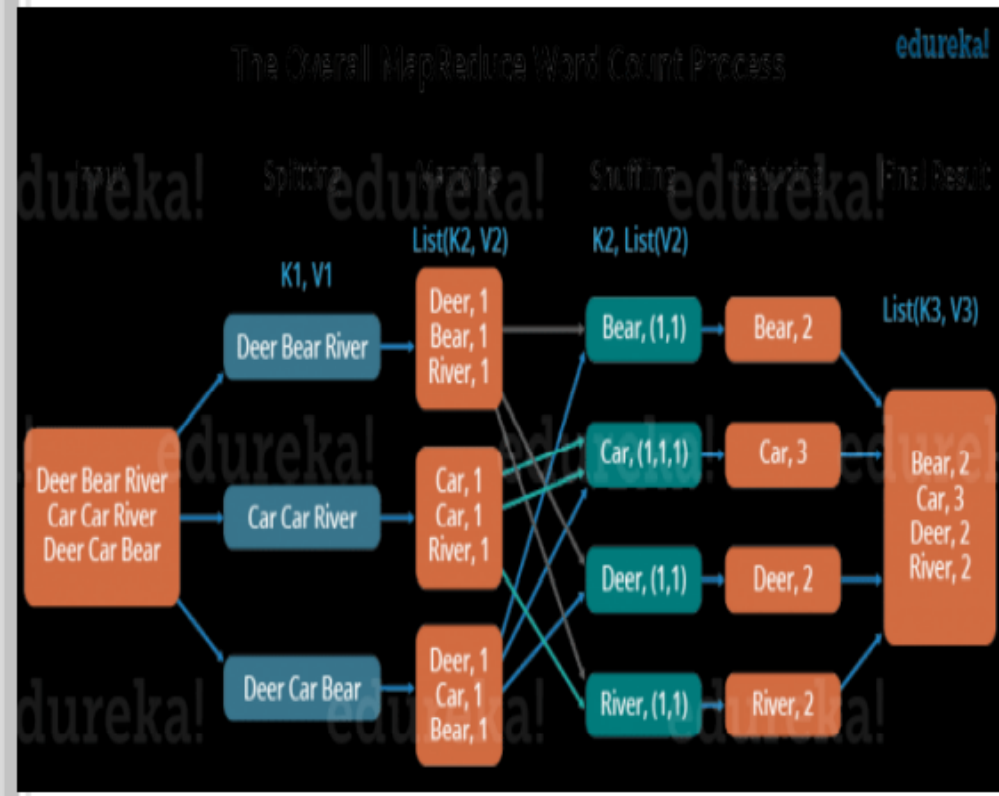
How it works



Source: https://www.edureka.co/blog/mapreduce-tutorial/



- First, the input is divided into three and distributed among the map nodes (mappers).
- Next, the words in each of the mapper is tokenized and each token or words is given a hardcoded value (1). A hardcoded value of 1 given because each word will appear at least once.
- Afterwards, a list of key-value pair will be created where the key is the individual words and value is one. Hence, for the third line (Dear Car, Bear) there exist 3 key-value pairs – Dear, 1; Car, 1; Bear, 1. same is done on all the mapper nodes.
- After mapper phase, sorting and shuffling occurs and all the tuples with the same key are sent to the corresponding reducer.
- At the end of sorting and shuffling, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Each Reducer then counts the values present in its list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.



Source: https://www.edureka.co/blog/mapreduce-tutorial/

HADOOP ECOSYSTEM

HADOOP-RELATED PROJECTS

- Pig (T4)
- -A high-level data-flow language and execution framework for parallel computation.

- Hive
- -A data warehouse infrastructure that provides data summarization and ad hoc querying.

- Impala
- -Impala is the open source, native analytic database for Apache Hadoop.

- Hbase
- -A scalable, distributed database that supports structured data storage for large tables.

- Spark (T5)
- -A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL(Extract-Transform-Load), machine learning, stream processing, and graph computation.

SCENARIO

香港城市大學
City University of Hong Kong

Task:

```
Key         value
.t^#\|v$2\ 0AAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHH
75@~?'WdUF 1IIIIIIIIIJJJJJJJJJKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
w[o||:N&H, 2QQQQQQQQQRRRRRRRRRSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
....
```

For the Key: 10 of random characters,
For the Value: 80 random characters,
we want to Sort it by the key:

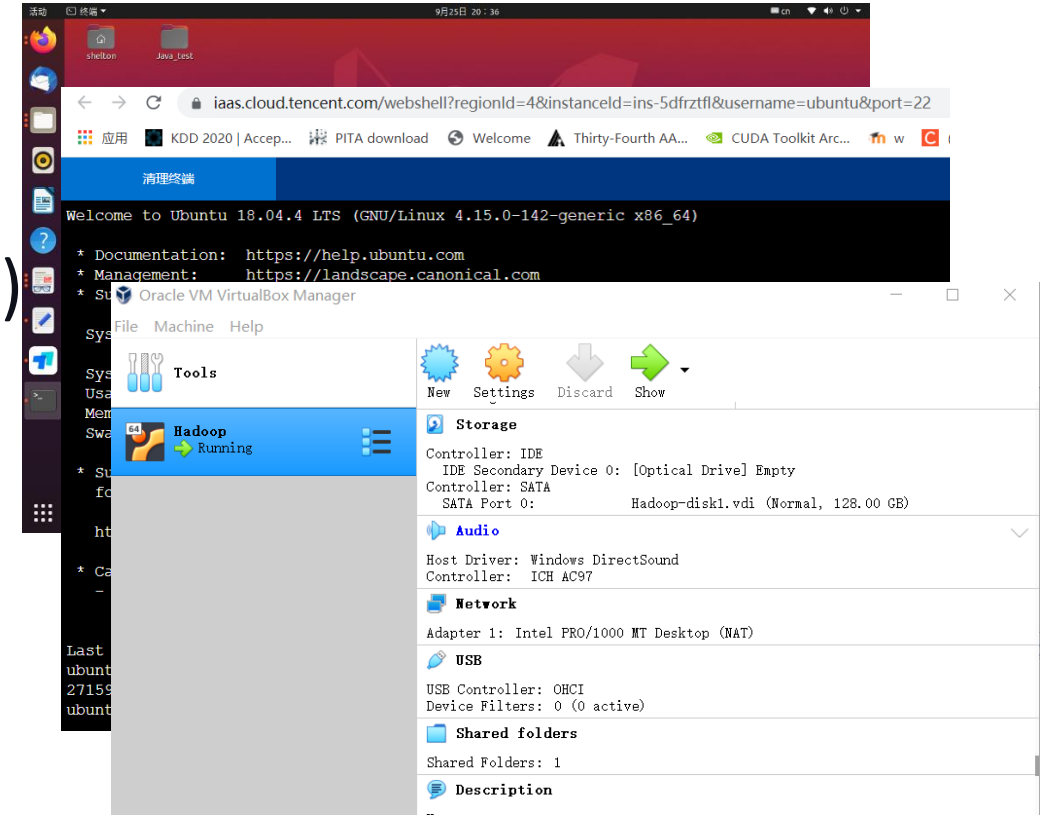
Assuming we have ten billion of key and value here(1 TeraByte),
How long will it take to sort all the data?

SCENARIO

- http://sortbenchmark.org/
- April 2008, Hadoop with 910 nodes became the fastest system to sort 1 TeraByte data in 209 seconds.
- 2016, 44.8 TB/min

SCENARIO

- One Linux PC(ubuntu, centos and so on)
- Linux cluster
- Virtual machine(Virtual Box)
- Rent a server on Cloud
- (such as Amazon Web Services,
- Tencent Cloud, and so on)



- In fact, Installing and Setting Hadoop are more challenging.

COMMANDS USED IN HDFS

香港城市大學
City University of Hong Kong

HDFS Command	Command Usage	Command Example	Function
version	version	hadoop version	prints the Hadoop version
mkdir	mkdir <path>	hdfs dfs -mkdir /test_dir	takes path URI's as an argument and creates directories.
put	put <localSrc> <dest>	hdfs dfs -put /home/dataflair/Desktop/sample /user/dataflair/dir1	copies the file or directory from the local file system to the destination within the DFS.
tail	hdfs dfs -tail <filename>	hdfs dfs -tail document.txt	shows the last 1KB of the file on console
ls	hdfs dfs -ls	hdfs dfs –ls	displays a list of the contents of a directory specified by path provided by the user, showing the names, permissions, owner, size and modification date for each entry.
cat	hdfs dfs –cat <filename>	hdfs dfs –cat /user/dataflair/dir1/sample	displays the contents of the filename on console
rm	hdfs dfs –rm <path>	hdfs dfs –rm /user/dataflair/dir1/sample	removes the file or empty directory(-rm –r not empty) present on the path provided by the user.

LAB3 HADOOP

- Platform: Windows/mac + Oracle VM VirtualBox
- Linux system: Ubuntu 14.04.1 LTS
- Hadoop version:2.6.0
-

```
bitnami@linux:~$ hadoop version
Hadoop 2.6.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r e3496499ecb8d22
0fba99dc5ed4c99c8f9e33bb1
Compiled by jenkins on 2014-11-13T21:10Z
Compiled with protoc 2.5.0
From source with checksum 18e43357c8f927c0695f1e9522859d6a
This command was run using /usr/local/hadoop-2.6.0/share/hadoop/common/hadoop-co
mmon-2.6.0.jar
```

- Java version:

```
bitnami@linux:~$ java -version
java version "1.8.0_40"
Java(TM) SE Runtime Environment (build 1.8.0_40-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.40-b25, mixed mode)
```

The latest version of Hadoop is 3.3.1
However,
Those basic commands are still available.

Lets Practice - HDFS

- Step 1 (Setup Hadoop machine)
  - Please, setup your Hadoop machine environment according to the lab 3 setup guide given in the CANVAS
- Step 2 (Login into your VM)
  - Login into your machine with Hadoop and open up a terminal (e.g. ctrl+alt+t)
- Step 3 (Basic HDFS commands)
  - Input the following basic HDFS commands

HDFS Commands

- Execute start-dfs.sh and jps

```
bitnami@linux:~$ start-dfs.sh
Starting namenodes on [0.0.0.0]
0.0.0.0: starting namenode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-
bitnami-namenode-linux.out
localhost: starting datanode, logging to /usr/local/hadoop-2.6.0/logs/hadoop-
bitnami-datanode-linux.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.6.0
/logs/hadoop-bitnami-secondarynamenode-linux.out
bitnami@linux:~$ jps
2451 Jps
2345 SecondaryNameNode
2186 DataNode
2061 NameNode
```

HDFS Commands

- Execute: hdfs dfs -usage

```
bitnami@linux:~$ hdfs dfs -usage
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
```

Transfer data to HDFS

```
bitnami@linux:~$ mkdir test_dir
bitnami@linux:~$ touch test_dir/test_file.txt
bitnami@linux:~$ hdfs dfs -put test_dir
bitnami@linux:~$ hdfs dfs -ls
Found 11 items
drwxr-xr-x   - bitnami supergroup          0 2015-05-14 02:39 .sparkStaging
drwxr-xr-x   - bitnami supergroup          0 2015-08-15 23:52 ex_data
drwxr-xr-x   - bitnami supergroup          0 2015-04-25 08:36 input
drwxr-xr-x   - bitnami supergroup          0 2015-08-11 18:50 max-temp
drwxr-xr-x   - bitnami supergroup          0 2015-05-15 18:38 max-temp-workflow
drwxr-xr-x   - bitnami supergroup          0 2015-08-11 18:57 max-temp2
drwxr-xr-x   - bitnami supergroup          0 2015-05-18 12:30 oozie-bitn
-rw-r--r--   1 bitnami supergroup         41 2015-08-13 17:44 sample.txt
drwxr-xr-x   - bitnami supergroup          0 2015-05-18 12:27 share
drwxr-xr-x   - bitnami supergroup          0 2015-08-13 20:46 temp
drwxr-xr-x   - bitnami supergroup          0 2015-08-28 15:42 test_dir
bitnami@linux:~$ hdfs dfs -ls test_dir
Found 1 items
-rw-r--r--   1 bitnami supergroup          0 2015-08-28 15:42 test_dir/test_file.txt
```

Your Task

- Delete test_dir/test_file.txt from HDFS
- Remove test_dir from HDFS
- Create test_dir2 on HDFS
- Copy test_file.text to test_dir2 on HDFS

Nano Editor

香港城市大學
City University of Hong Kong

- Install directly with “sudo apt install nano”. Put password when requested
- ^ means CTRL e.g., ^G means CTRL + G
- M means ALT e.g., M6 means ALT + 6
- To open a file, nano filename
- You can edit immediately
- To copy text,
  - Move cursor to beginning of text, press ALT + a to set selection mark,
  - Move cursor to end of text using arrow keys to highlight
  - To cancel selection, CTRL + 6
- To copy, ALT + 6
- To cut, CTRL + k
- To paste, CTRL + u
- To save, CTRL + o
- To exit nano, CTRL + x

VIM Editor

香港城市大學
City University of Hong Kong

- There are two modes, Command and Insert Mode
- To open file, vim filename. Now you are in command mode
- To edit, press “i”. Now you are in insert mode
- To go back to command mode, press ESC
- To see line numbers, :set number
- To move, use h, j, k, l for ←, ↓, ↑, → respectively.
- To navigate,
  - w for start of next word,
  - e for end of the word
  - b for beginning of the word
- To save and quit press :wq
- To quit without saving, press :q!

香港城市大學
City University of Hong Kong