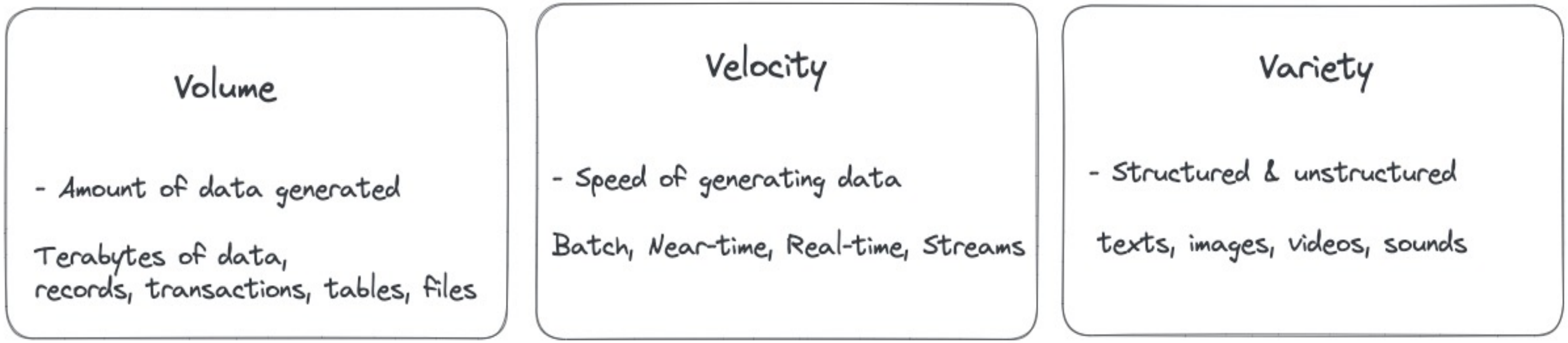
Big Data Course

- Key Concepts

The 3V's of Big Data



Volume

- Amount of data generated

Terabytes of data,
records, transactions, tables, files

Velocity

- Speed of generating data

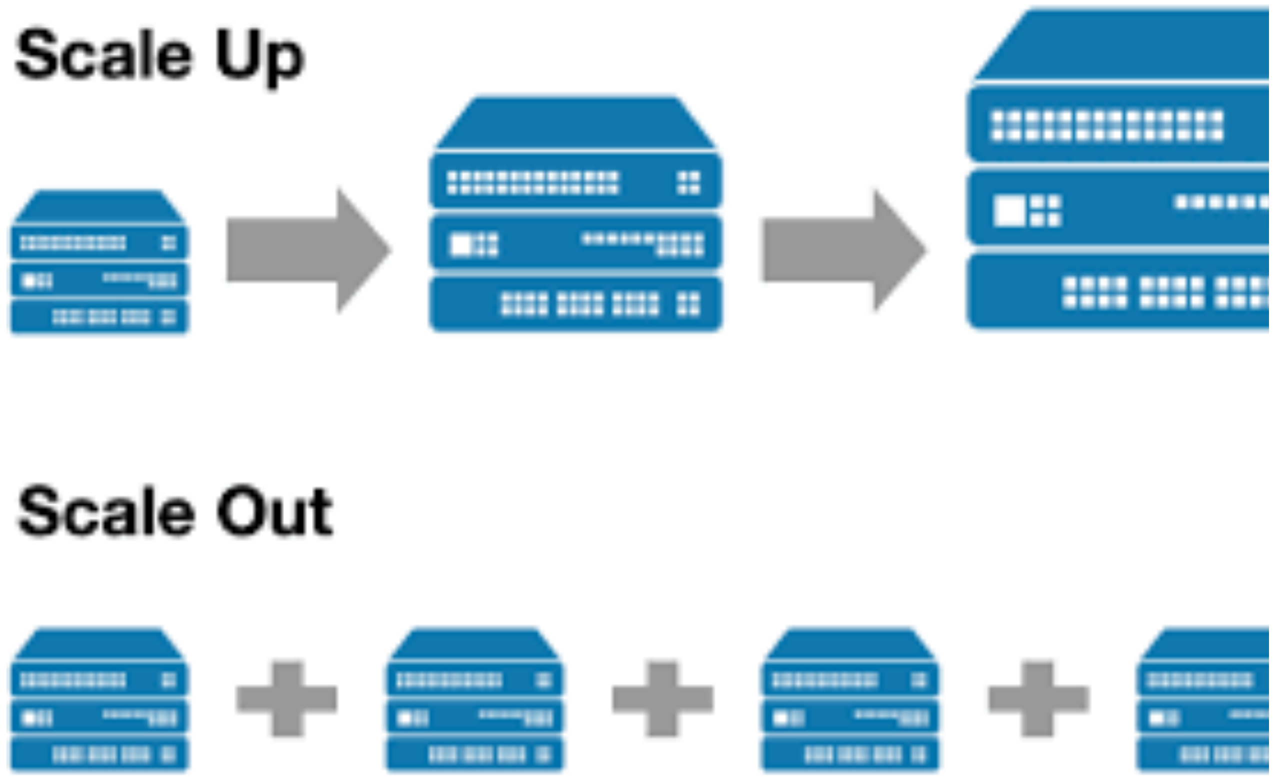
Batch, Near-time, Real-time, Streams

Variety

- Structured & unstructured

texts, images, videos, sounds

Scaling Up vs Scaling Out



Scaling up (vertical scaling)

Buying more expensive hardware e.g. more memory, better CPU

Scaling out (horizonal scaling) buying cheaper hardware

Distributed Storage

Advantages:

- **Scalability**—the primary motivation for distributing storage is to scale horizontally, adding more storage space by adding more storage nodes to the cluster.
- **Redundancy**—distributed storage systems can store more than one copy of the same data, for high availability, backup, and disaster recovery purposes.
- **Cost**—distributed storage makes it possible to use cheaper, commodity hardware to store large volumes of data at low cost.
- **Performance**—distributed storage can offer better performance than a single server in some scenarios, for example, it can store data closer to its consumers, or enable massively parallel access to large files.

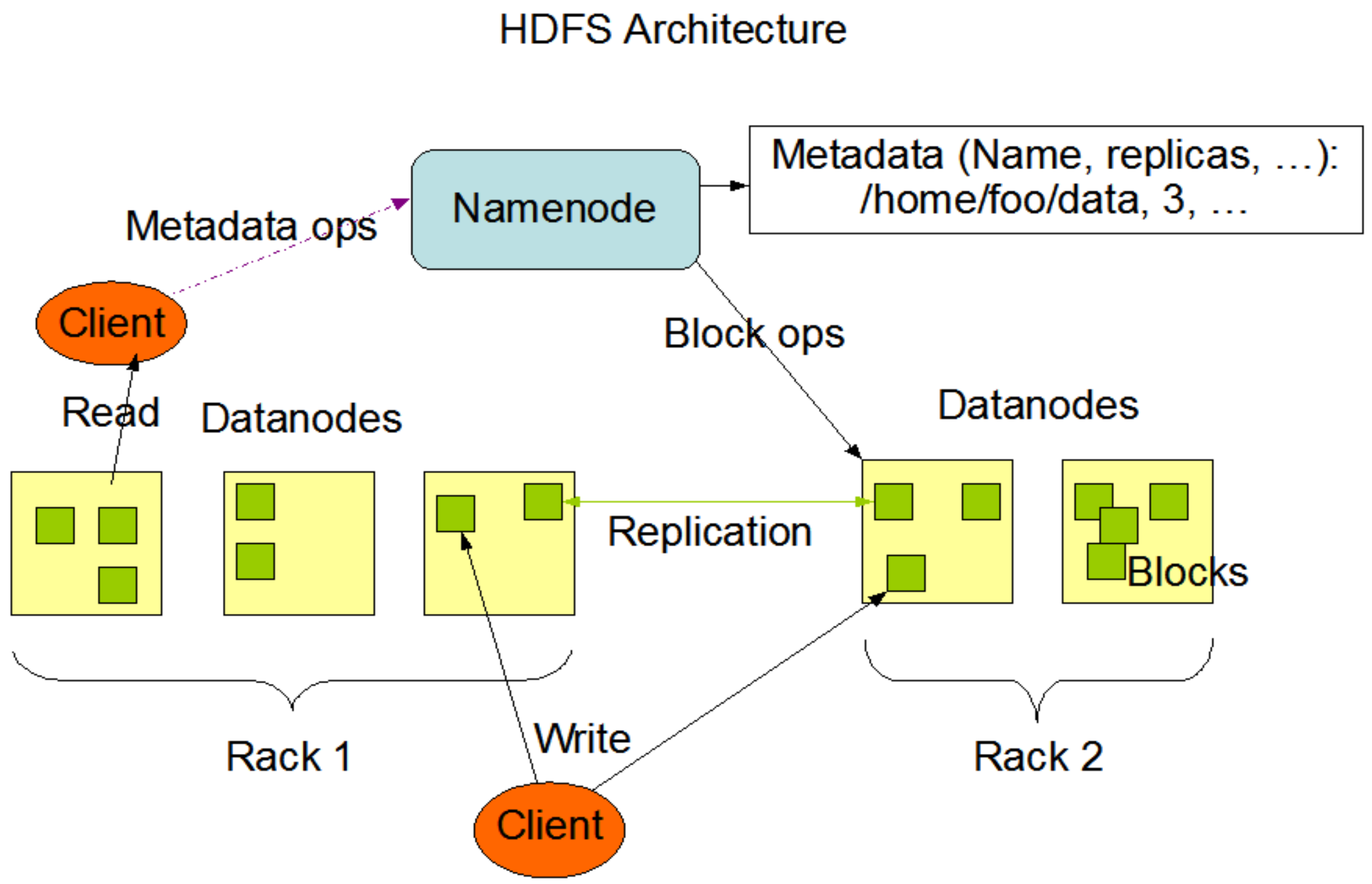
Distributed Storage Features

Most distributed storage systems have some or all of the following features:

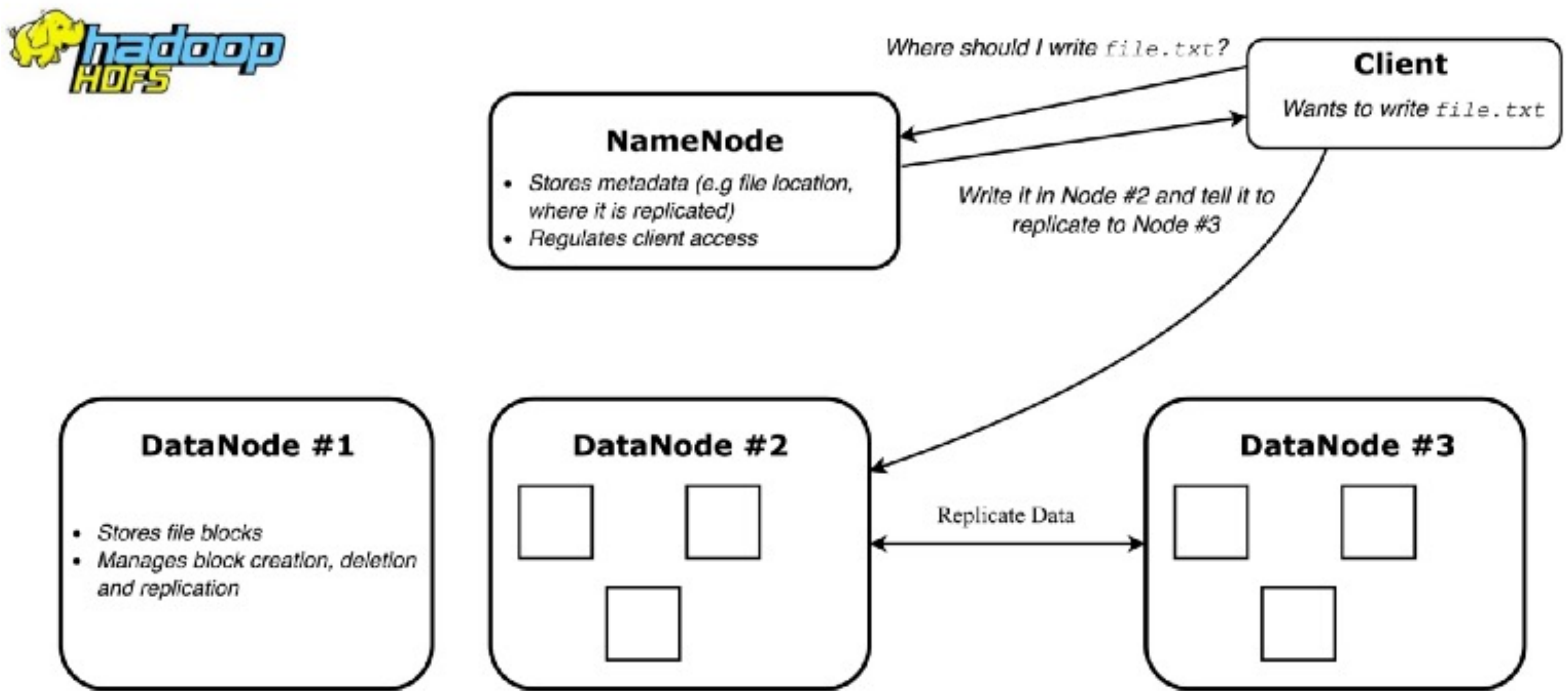
- **Partitioning**—the ability to distribute data between cluster nodes and enable clients to seamlessly retrieve the data from multiple nodes.
- **Replication**—the ability to replicate the same data item across multiple cluster nodes and maintain consistency of the data as clients update it.
- **Fault tolerance**—the ability to retain availability to data even when one or more nodes in the distributed storage cluster goes down.
- **Elastic scalability**—enabling data users to receive more storage space if needed, and enabling storage system operators to scale the storage system up and down by adding or removing storage units to the cluster.

HDFS Architecture

Hadoop Distributed File System – an example of distributed system technology



A Simpler Model:


hadoop HDFS
Where should I write file.txt?
Client
Wants to write file.txt
NameNode
• Stores metadata (e.g file location, where it is replicated)
• Regulates client access
Write it in Node #2 and tell it to replicate to Node #3
DataNode #1
• Stores file blocks
• Manages block creation, deletion and replication
DataNode #2
Replicate Data
DataNode #3

Architecture Overview

Its architecture consists mainly of NameNodes and DataNodes.

NameNodes are responsible for keeping metadata about the cluster, like which node contains which file blocks.

They act as coordinators for the network by figuring out where best to store and replicate files, tracking the system’s health.
DataNodes simply store files and execute commands like replicating a file, writing a new one and others.

Concepts:

NameNode

- It manages the file system namespace and regulates access to files by clients
- keeps track of the location of data blocks in the cluster

DataNode

- Responsible for storing data blocks
- Communicate with the NameNode to report the status of data blocks and perform block replication and recovery

Blocks

- Files in HDFS are broken down into blocks, typically 128MB or 256MB in size
- Blocks are replicated across multiple DataNodes for fault tolerance and high availability
- The replication factor is configurable (default is 3) but can be anywhere between 1 and the number of DataNodes in the cluster

Rack

- A group of servers
- HDFS tries to place replicas of blocks on different racks to improve fault tolerance and reduce the impact of network congestion

Metadata

- HDFS stores metadata about files and directories in the file system
- This metadata includes information such as file names, permissions, modification times, and block locations
- The metadata is stored in memory on the NameNode and is periodically written to disk for durability

Client

- Clients are applications that use HDFS to read or write dataBlocks are replicated across multiple DataNodes for fault tolerance and high availability
- They communicate with the NameNode to locate the blocks of data they need and communicate directly with the appropriate DataNodes to perform I/O operations

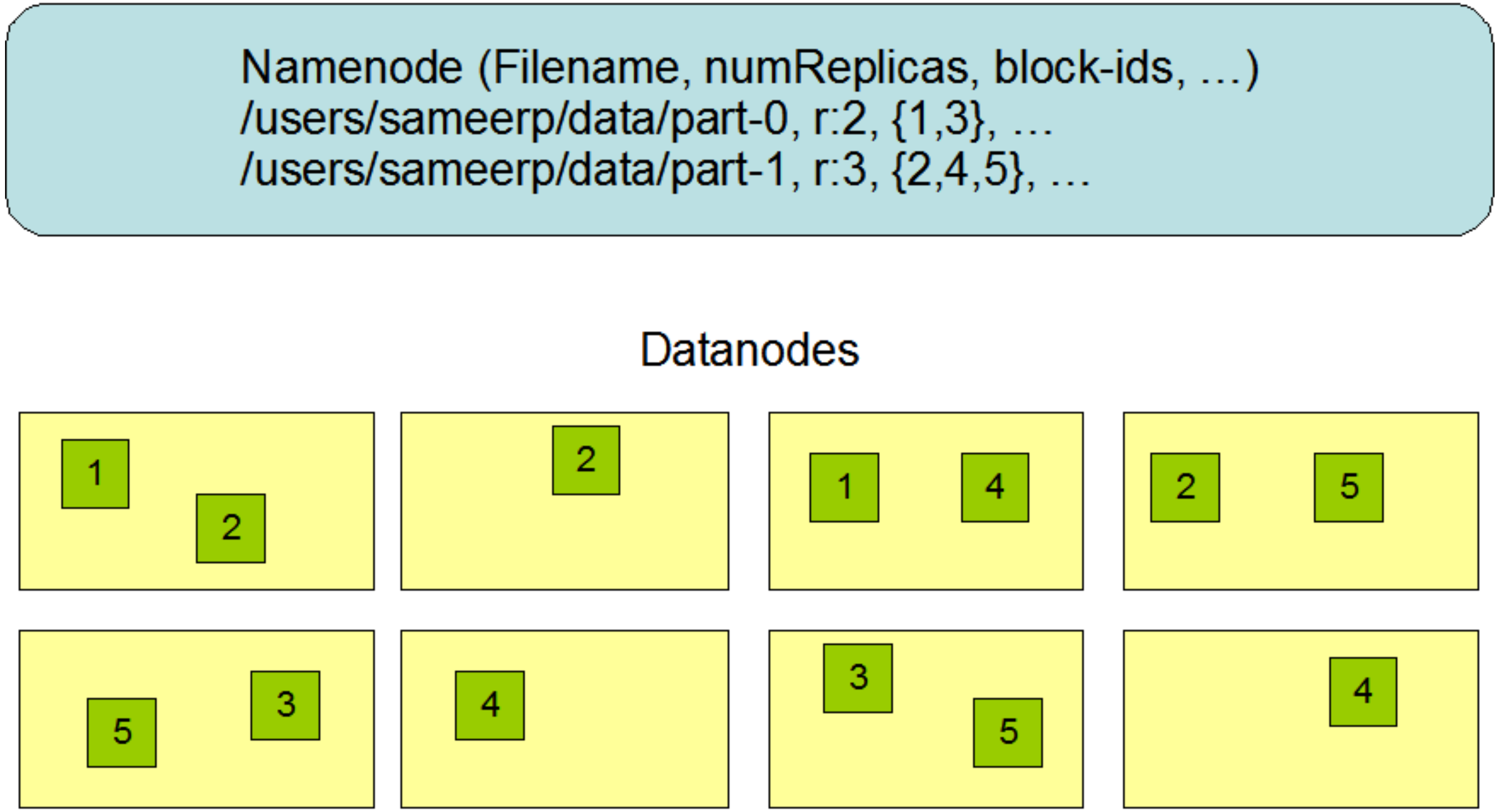
Replication

- HDFS uses replication to ensure fault tolerance and data availability
- By default, each data block is replicated three times across different DataNodes in the cluster
- The replication factor can be adjusted based on the storage capacity and reliability of the cluster
- HDFS also supports block-level checksums to detect and recover from data corruption.

Fault Tolerance

- HDFS is designed to be highly fault-tolerant, with built-in mechanisms to detect and recover from failures
- If a DataNode fails, the NameNode will replicate the lost blocks to other nodes in the cluster
- If the NameNode itself fails, a standby NameNode can take over within seconds to ensure uninterrupted operation of the file system
- HDFS also supports data integrity checks and automatic data recovery to ensure that data remains consistent even in the event of hardware or software failures.

Block Replication



Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Distributed Computing

Scalability

- Scalability is the ability of a distributed computing system to handle increasing workloads as more nodes are added
- Distributed systems must be designed to scale horizontally, adding more nodes to the system to handle additional tasks

Parallelism

- Parallelism is the ability of a distributed computing system to divide a task into smaller sub-tasks that can be processed concurrently on different nodes
- Parallel processing can improve performance and reduce the time required to complete a task

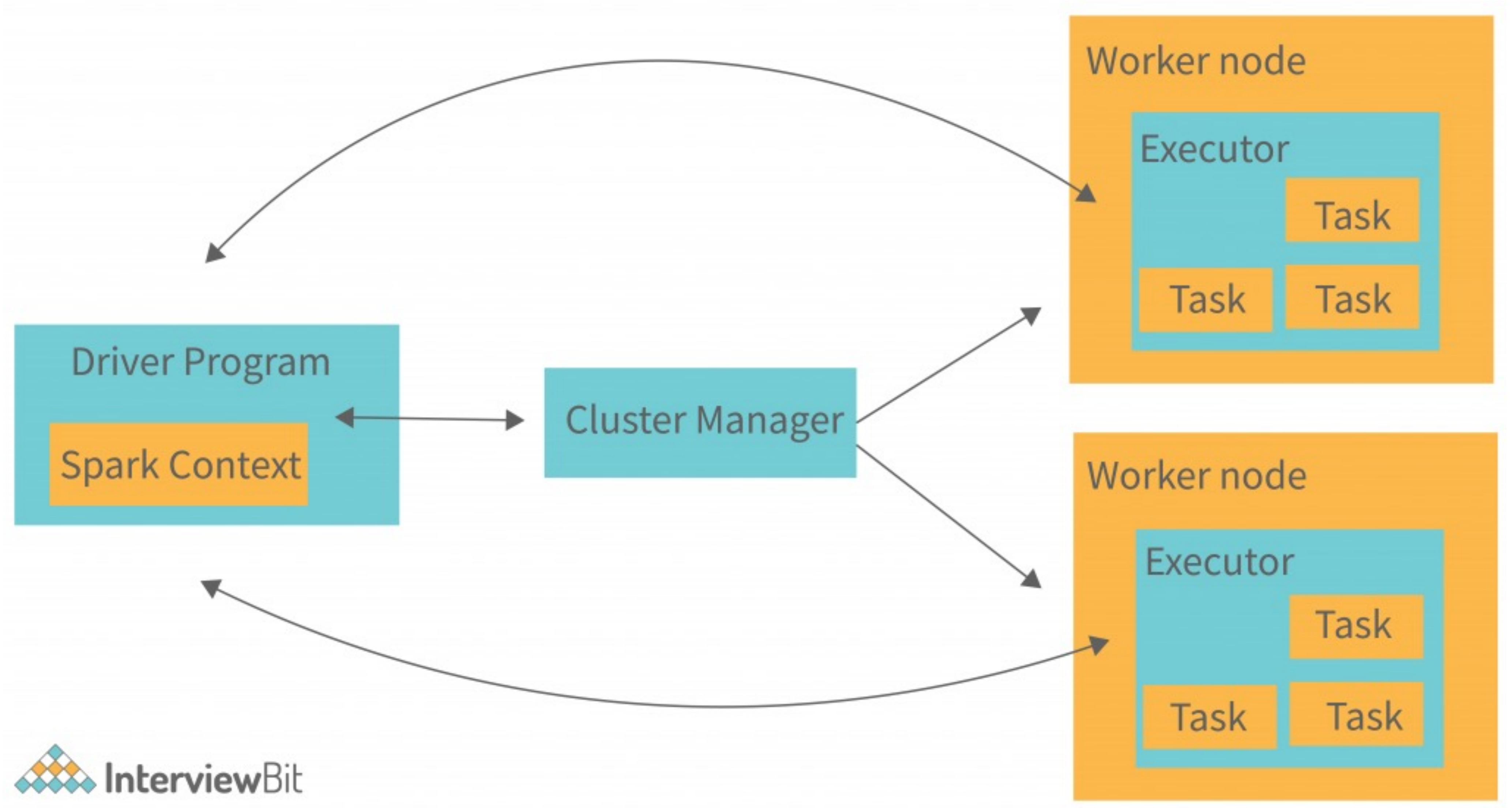
Fault Tolerance

- Fault tolerance is the ability of a distributed computing system to continue operating in the presence of failures
- Fault tolerance can be achieved through redundancy, replication, and self-healing mechanisms

Consistency

- In a consistent system, all nodes see and return the same information simultaneously. In order to ensure that all nodes have the same data, they need to exchange messages and work in synchronization.
- However, in speaking of data communications between nodes, some difficulties may arise. For example, messages' delivery may fail, or messages may get lost, or some nodes may be unavailable at some point.
- Generally, the weaker the required level of consistency, the faster the system can work – but at the same time the higher chances that it won't return the latest dataset.

Spark Architecture



Spark

- Spark is a distributed computing system designed to process large amounts of data across a cluster of computers.
- It operates on a master-worker architecture, in which a master node coordinates tasks and assigns them to worker nodes.

Spark Driver

- The driver is the process where the main method runs. First it converts the user program into tasks and after that it schedules the tasks on the executors.

Spark Executors

- Executors are worker nodes' processes in charge of running individual tasks in a given Spark job.
- They are launched at the beginning of a Spark application and typically run for the entire lifetime of an application. Once they have run the task they send the results to the driver.

Application Execution Flow

1. A standalone application starts and instantiates a SparkContext instance (and it is only then when you can call the application a driver).

2. The driver program ask for resources to the cluster manager to launch executors.

3. The cluster manager launches executors.

4. The driver process runs through the user application. Depending on the actions and transformations over RDDs task are sent to executors.

5. Executors run the tasks and save the results.

6. If any worker crashes, its tasks will be sent to different executors to be processed again

   Spark automatically deals with failed or slow machines by re-executing failed or slow tasks. For example, if the node running a partition of a map() operation crashes, Spark will rerun it on another node; and even if the node does not crash but is simply much slower than other nodes, Spark can preemptively launch a “speculative” copy of the task on another node, and take its result if that finishes.

7. With SparkContext.stop() from the driver or if the main method exits/crashes all the executors will be terminated and the cluster resources will be released by the cluster manager.