CSE 4/587
Data Intensive Computing

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 07
Introduction to Hadoop

Announcements and Feedback

- Project Phase 1 is posted
    - Phases 2 and 3 are being finalized
    - Sign up as a team with one of the three TAs
    - Start early, phase 2 and 3 will be built on top of phase 1
- Starting next week, Tuesday office hours are moved to Thursday permanently.

Recap from Last Class

- Reviewed 3 different algorithms for analyzing our data
  - Linear regression
  - k-Nearest Neighbors
  - k-Mean
- Demonstration of each one briefly in Python (using JupyterLab, pandas, and scikit-learn)
  - Demo code is posted on the website

What happens when our data gets bigger?

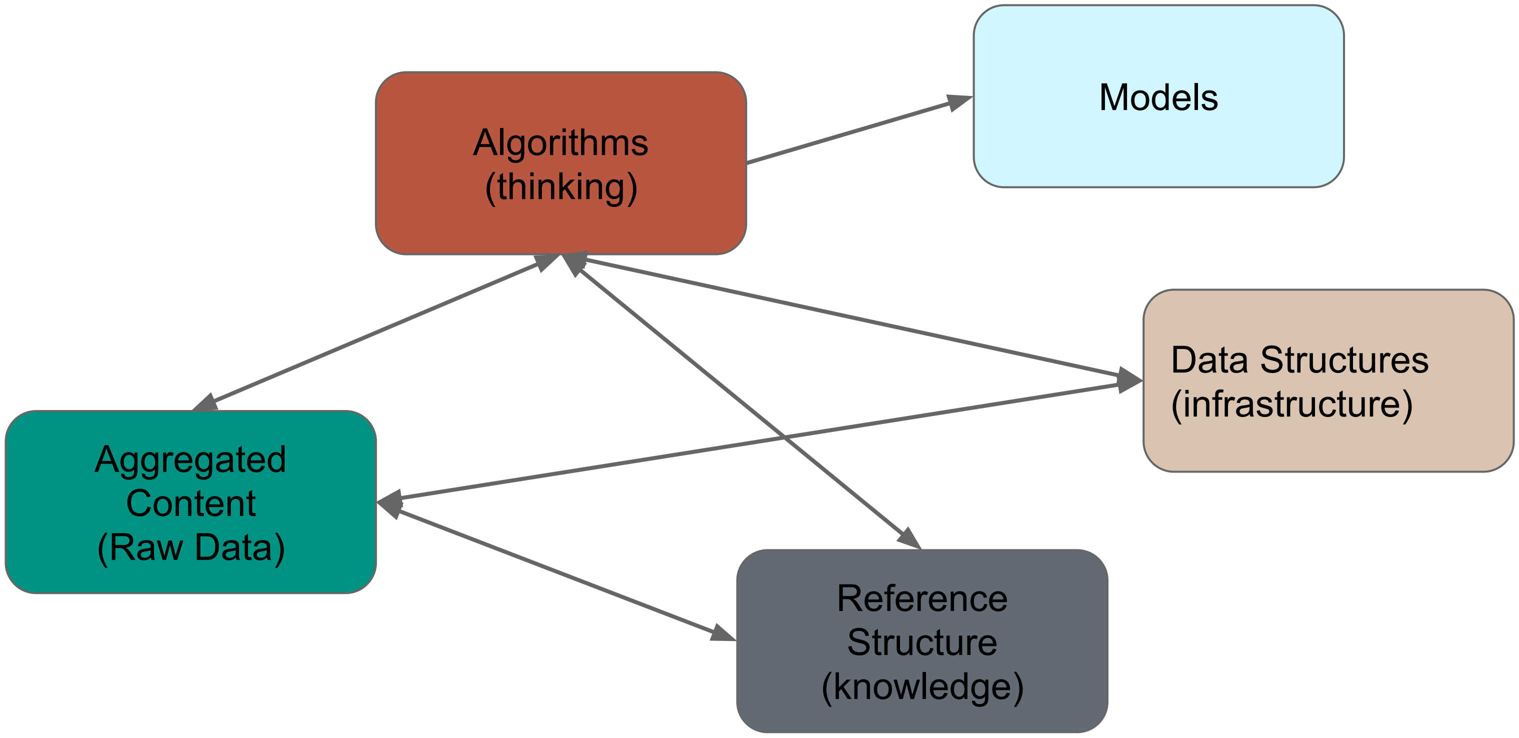
- Examples from last time were using a small dataset
  - Reading and analyzing the data could easily take place on our laptop
- What happens as our data gets bigger and bigger?
  - Algorithms are one concern…
  - …but even before we get to that…

What happens when our data gets bigger?

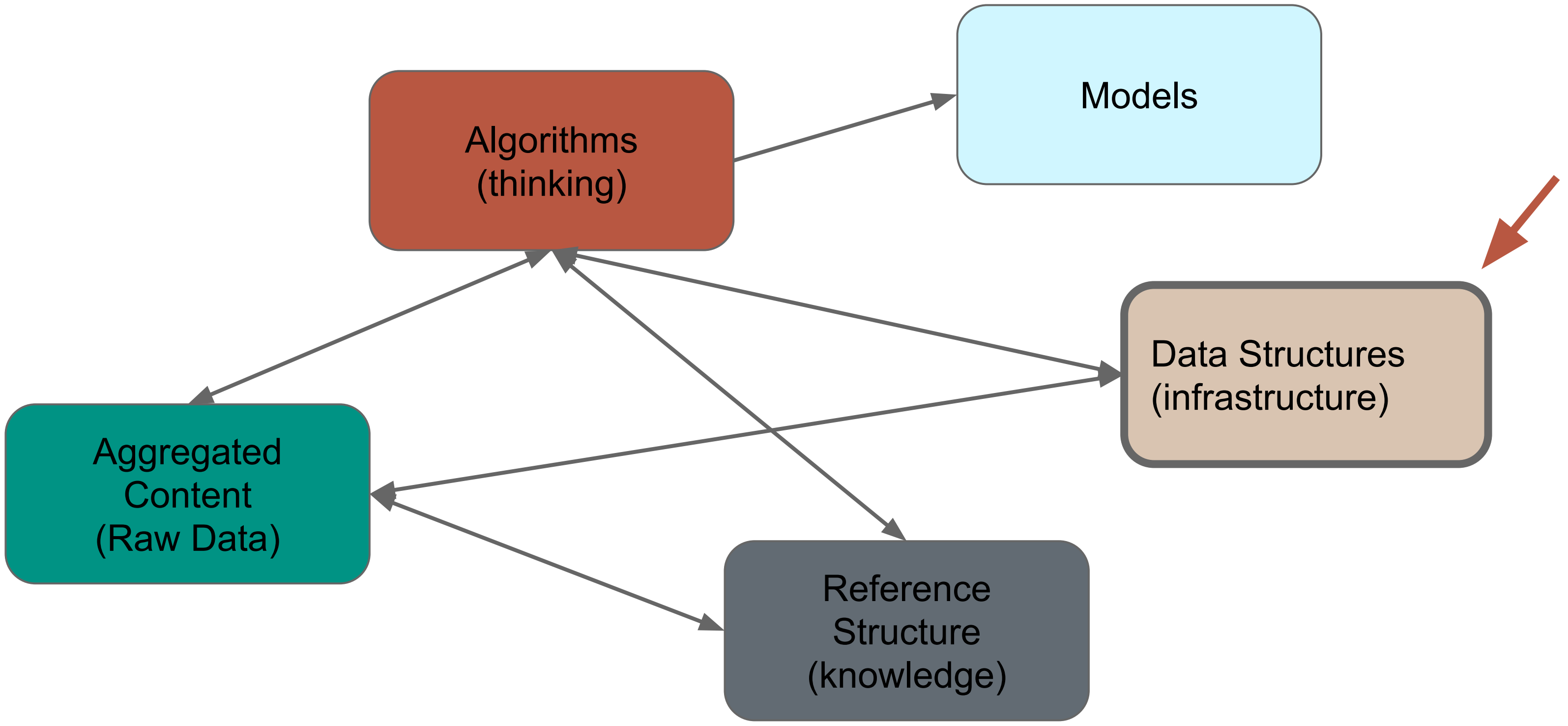
- Examples from last time were using a small dataset
  - Reading and analyzing the data could easily take place on our laptop
- What happens as our data gets bigger and bigger?
  - Algorithms are one concern…
  - …but even before we get to that…

How do we store and access all this data!?

Components of Data Intensive Computing



Components of Data Intensive Computing



Hadoop

- The internet introduced a new challenge with web logs
  - Large scale (petascale)
  - Unique characteristic: "Write Once, Read Many" (WORM)
- Google exploited this characteristic in its Google File System (GFS)
- Hadoop is the open source version of GFS
  - Distributed file system
  - Started as part of project Nutch and Lucene (focused on search)
  - Found to have wide application outside of search
  - Now its own Apache project

Hadoop Features

- Highly fault-tolerant
- High throughput
- Handles large data sets effectively
- Streaming access to file system data
- Can be built from commodity hardware
- Files browsable with any HTTP browser
- Initially provided a Java API — but now supports most programming languages

Key Aspects

- The key aspects of Hadoop we will discuss today are:
  - Architecture
  - Protocol - rules for operation
  - Data Organization
  - Robustness
  - API to access services
  - Software (MapReduce)

Architecture

Architecture: NameNode and DataNodes

- HDFS clusters consist of a single **NameNode** and multiple **DataNodes**

NameNode

A master server that manages the filesystem namespace, tracks metadata, and regulates client access to files.

DataNodes

Usually one per node in a cluster.

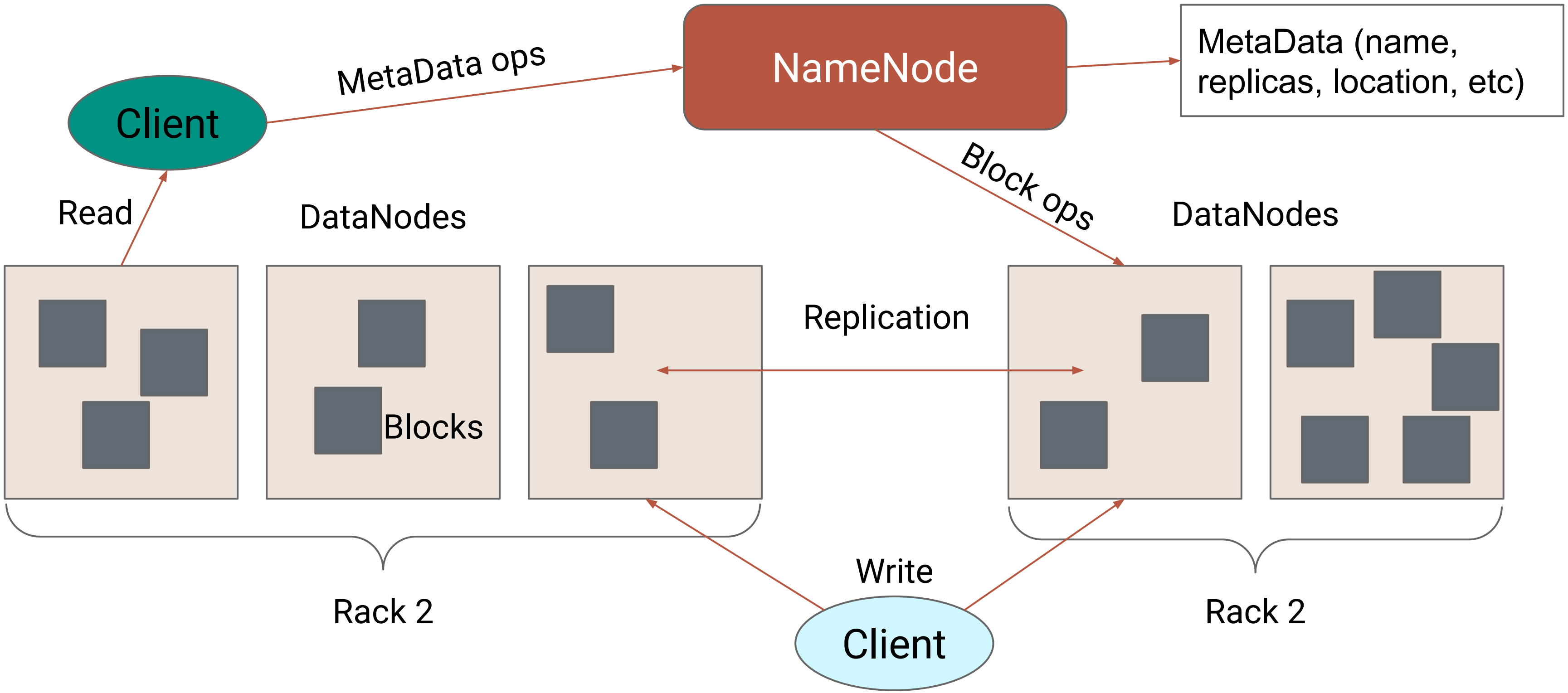
Manages storage attached to their node.

Serves read/write requests, file creation/deletion, and replication.

Architecture: Files

- HDFS exposes a filesystem namespace, and allows user data to be stored in files.
- A file is split into one or more blocks, and sets of blocks are stored in the DataNodes
  - All blocks in a file are the same size (except the last)
  - These blocks may be replicated and/or migrated

Architecture



File System

- Hierarchical file system with directories and files
  - Create, remove, move, rename, etc
- NameNode maintains the file system
- Any metadata changes to the file system recorded by the NameNode
- The replication factor for each file is also stored by the NameNode

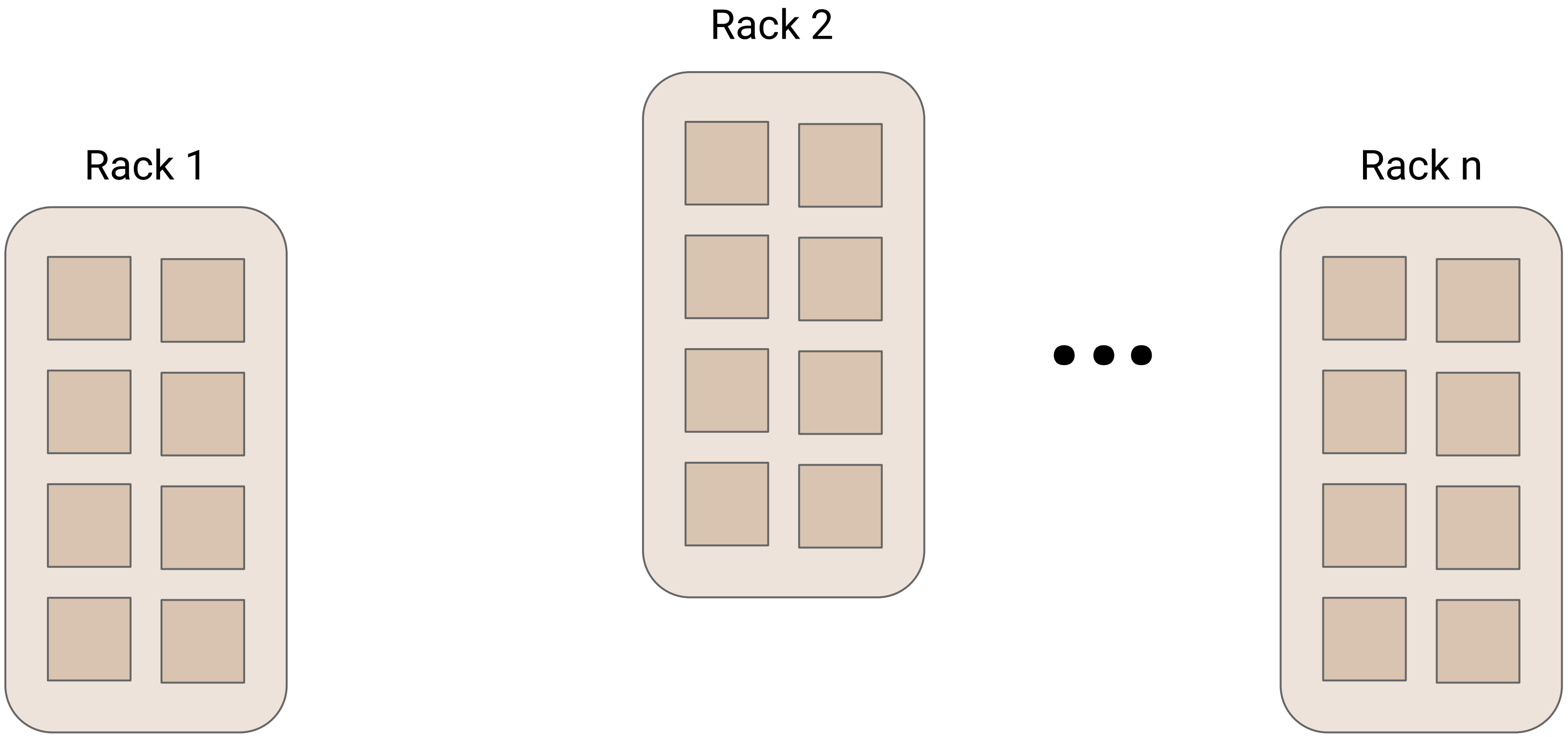
Data Replication

- HDFS is designed to store very large files across machines in a cluster
- Each file is a sequence of blocks
  - All blocks in a file are the same size (except the last block)
  - Blocks are replicated for fault tolerance
  - Block size and replica are configurable per file
- The NameNode receives a **heartbeat** and **BlockReport** from each DataNode
  - This report contains information about all the blocks on the DataNode

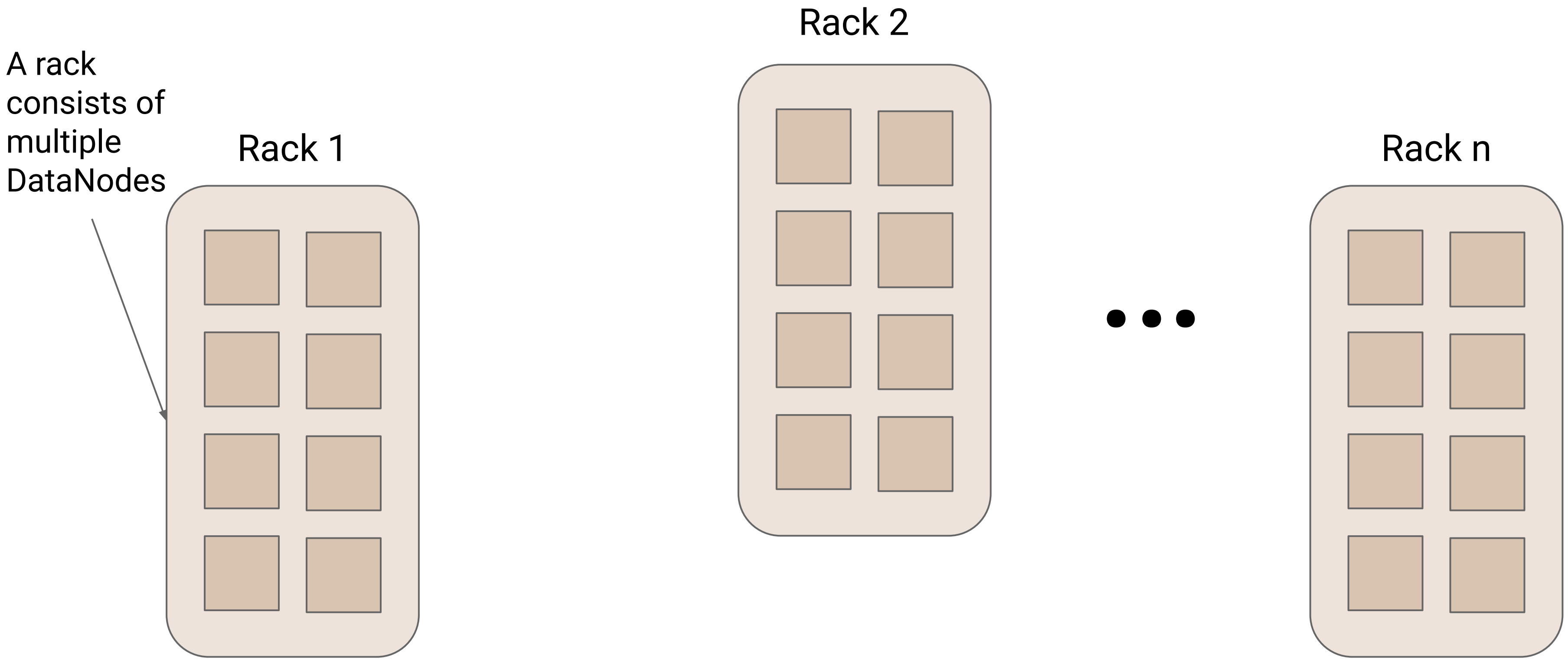
Replica Placement

- Replica placement is critical to reliability and performance
- Optimizing replica placement is what distinguishes HDFS from other distributed file systems
- First, what does a typical cluster look like…

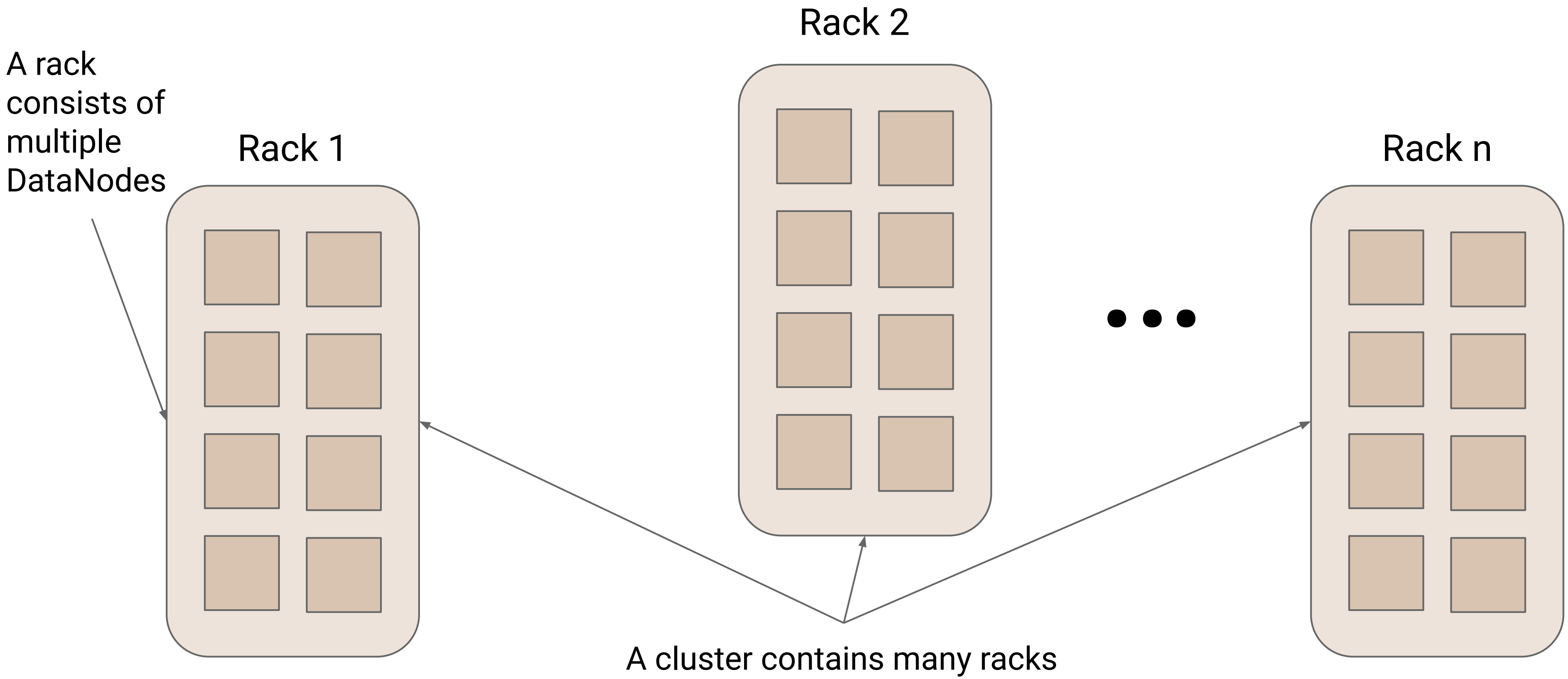
Replica Placement



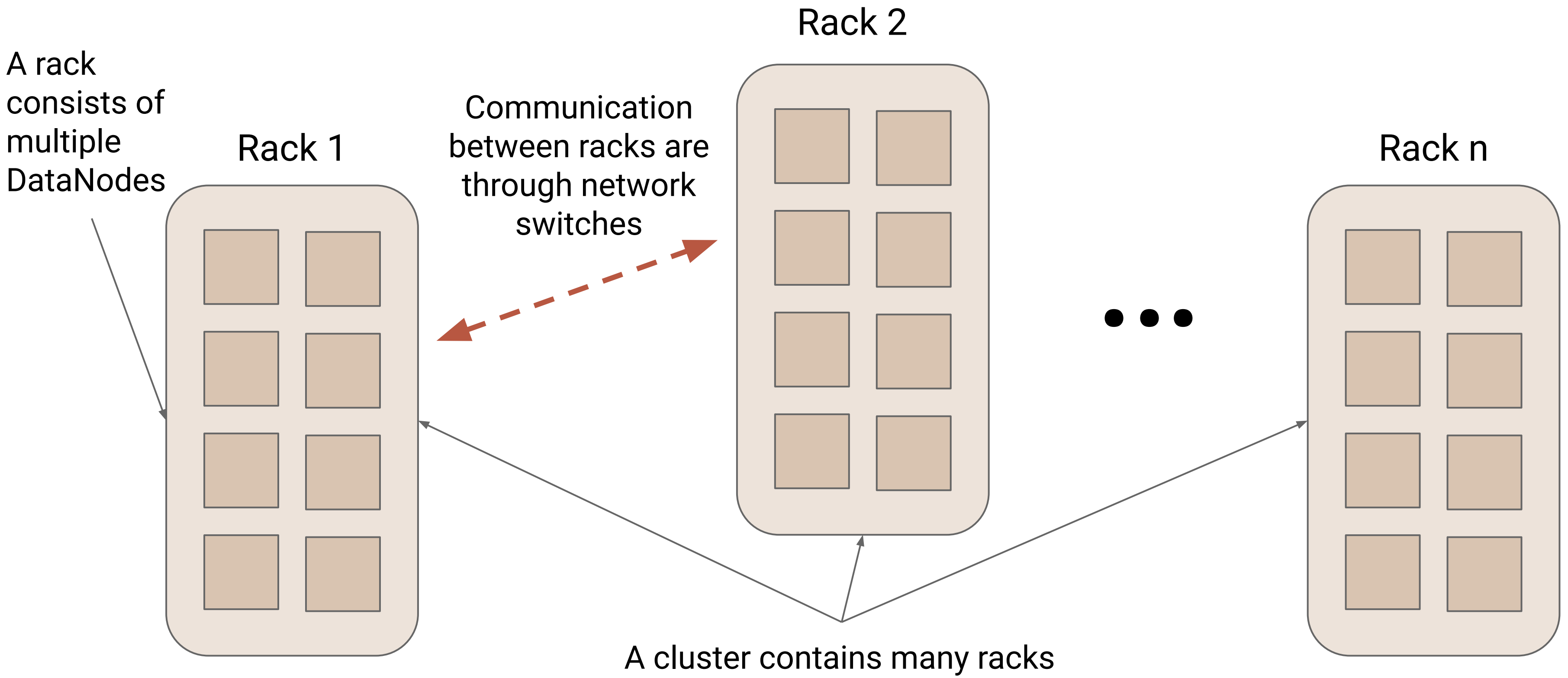
Replica Placement



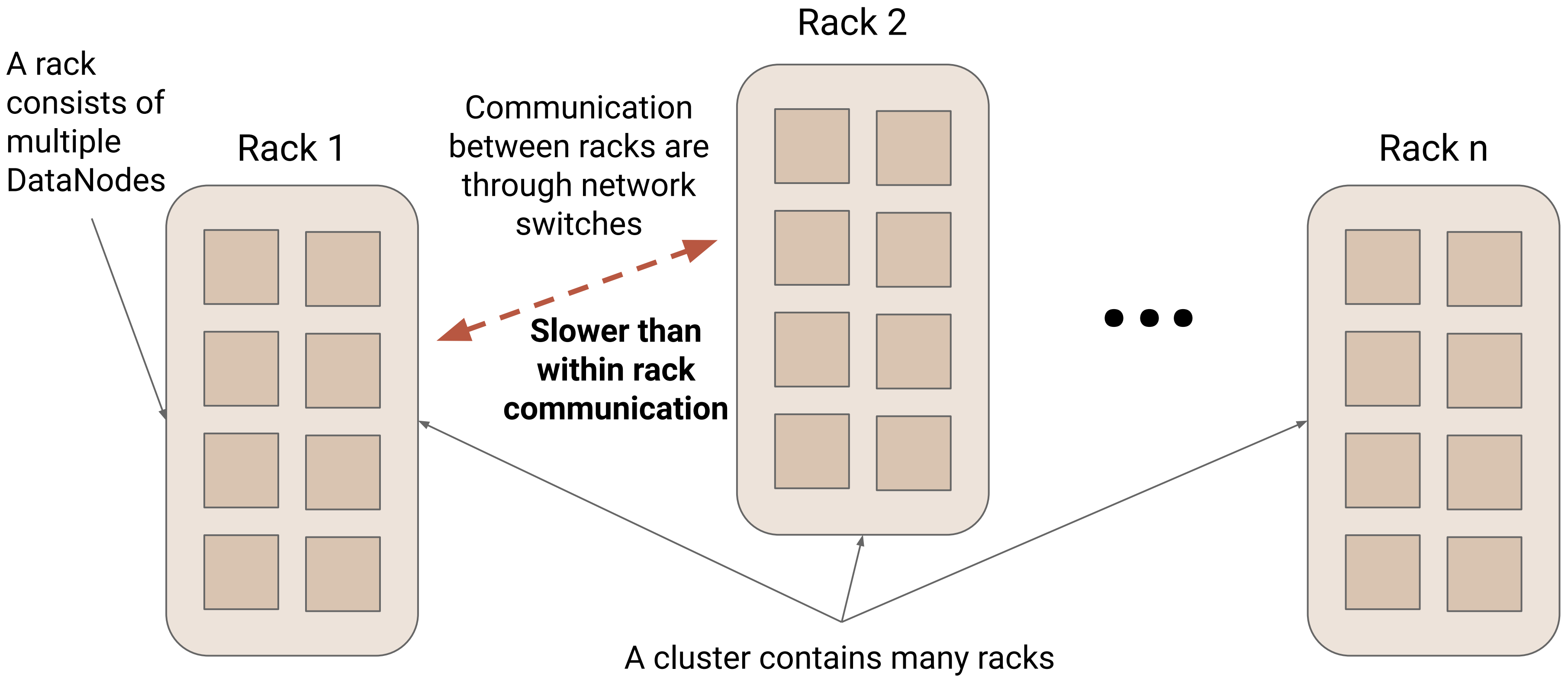
Replica Placement



Replica Placement



Replica Placement



Replica Placement

Rack Aware Placement

- Goal: Improve reliability, availability, and network bandwidth utilization
- NameNode determines the rack id for each DataNode
- Replicas are typically placed on unique racks
    - Simple scheme but non-optimal
    - Writes are expensive
    - Typical replication factor is 3

Replica Placement

Rack Aware Placement

- Goal: Improve reliability, availability, and network bandwidth utilization
- NameNode determines the rack id for each DataNode
- Replicas are typically placed on unique racks
  - Simple scheme but non-optimal
  - Writes are expensive
  - Typical replication factor is 3
- **Improvement:** Place one on a node in the local rack, one on a node in a remote rack, and another on a different node in that same remote rack.
- For a file this means ⅓ of the replicas on one node, ⅔ on one rack, and the other ⅓ distributed across all other racks. *(not an even distribution)*

Replica Selection

- When a client attempts to perform a read, HDFS tries to minimize bandwidth consumption and latency.
  - If there is a replica on the reader's node, then that is preferred
  - Otherwise try for the same rack
  - An HDFS cluster may span multiple data centers, a replica in the local data center is preferred over a remote data center

Startup (Safemode)

- On initial startup of a NameNode, the system enters a safe mode
  - During safe mode no replication occurs
- Each DataNode checks in with a heartbeat and a BlockReport
- Each Block in the BlockReport has a minimum number of replicas to be considered "safely replicated"
- The NameNode waits for a certain percentage of blocks to reach the threshold for safely replicated before leaving safe mode and replicating any blocks which are lacking

File System MetaData

- The HDFS namespace is stored by NameNode
- NameNode uses a transaction log called the EditLog to record every change that occurs to file system MetaData
  - Creating a new file
  - Changing the replication factor
  - Deletion
- EditLog is stored in the NameNode's local file system
- Entire namespace including block mapping is stored in FsImage file in the NameNodes local filesystem.

DataNode

- A DataNode stores data in files in its local file system
- DataNode has no knowledge about the HDFS file system
- It stores each block in a separate file
- It does not create all files in the same directory
    - It uses heuristics to determine the optimal number of files per directory
- Upon starting, it generates a list of all blocks and send the report to the NameNode

Data Organization

Data Blocks

- HDFS supports write-once-read-many (WORM) with reads at streaming speeds
- A typical block size is 64MB (or even 128MB).
- A file is chopped into 64MB chunks and stored.

Staging

A client request to create a file does not reach NameNode immediately

1. An HDFS client caches the data into a temporary file
2. Once the data reaches the block size, the client contacts the NameNode
3. NameNode inserts the filename into its hierarchy and allocates a data block
4. The NameNode responds to the client with the identity of the DataNode for the block, as well as destinations for the replicas
5. The client writes the block and flushes the file from its local memory

Staging (cont.)

6. When the file is closed the client sends the remaining data to the DataNode
7. The client then informs the NameNode that the write is complete
8. The NameNode commits the file creation op to a persistent store
    a. If the NameNode dies before the commit, the file is lost

Data Pipelining

- During the write process, the data blocks are flushed in small pieces (4K) to the first replica.
- That replica in turn copies it to the next replica, etc.

Protocol

Communication Protocol

- All HDFS communication protocols are layered on top of TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the NameNode machine
  - It talks to the NameNode machine using ClientProtocal
- The DataNodes talk to the NameNode using DataNodeProtocol
- RPC abstraction wraps both of these protocols
- NameNode is just a server, never initiates requests. Just responds.

Robustness

Possible Failures

- The primary objective of HDFS is to store data reliably in the presence of failures
- Three common failures that it must handle are:
  - DataNode failure
  - Network partition
  - NameNode failure

DataNode Failure and Heartbeat

- A crashed DataNode or a network partition can cause a subset of DataNodes to lose connectivity with the NameNode
- NameNode detects this by the absence of a heartbeat
    - NameNode marks these DataNodes, and does not send requests to them
    - Data registered to the failed DataNode is not available to the HDFS
    - Death of a DataNode may cause some blocks to require more replication

Re-Replication

- Sometimes Blocks in the system may fall below the required replication factor
- This can occur for a number of reasons
  - A DataNode has become unavailable
  - A replica may become corrupted
  - A hard disk on a DataNode may fail
  - The replication factor may have been increased

Data Integrity

- What if a block of data fetched from a DataNode arrives corrupted
  - Fault in a storage device
  - Network faults
  - Buggy software…our favorite :)
- An HDFS client creates a checksum for every block of its file and stores it in the HDFS namespace
- When another client retrieves the contents of a file it verifies that they match…if not it must retrieve the block from another replica

MetaData Disk Failure

- FsImage and EditLog are central data structures of HDFS
  - Corruption of these files can cause an entire HDFS instance to become non-functional.
- A NameNode can be configured to maintain multiple copies of these files
  - These copies are updated synchronously
  - MetaData is not data intensive
- The NameNode is a potential single point of failure
  - This currently requires manual intervention

Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes
- A scheme may more data from one DataNode to another if the free space on a DataNode is falling below a certain threshold
- A scheme may dynamically create and place additional replicas and rebalance other data if there is sudden high demand for a particular file
- These types of rebalancing are not yet implemented

API

FS Shell, Admin, and Browser Interface

- HDFS organizes its data in files and directories
- Command line interface called the FS shell
  - Syntax similar to bash and csh
  - ie: /bin/hadoop dfs -mkdir /foo
- There is also a DFSAdmin interface
- A browser can be used to view the namespace

Space Reclamation

- When a file is deleted, HDFS moves it to a trash directory for a configurable amount of time
- A client can request for the file to be recovered during this time
- After the specified time the file is deleted along with replicas, and all space is reclaimed
- This will also occur automatically if the replication factor is reduced

Software

Software System

- MapReduce requires a distributed file system, and an engine to distribute, coordinate, monitor, and gather results
- Hadoop provides the file system, and the engine through its JobTracker and TaskTracker system
  - JobTracker is simply a scheduler
  - TaskTracker is assigned tasks to run on its node
    - Each task runs in its own JVM

Job Tracker

- Scheduler service in the Hadoop system
- Client appliction is sent to the JobTracker
  - It talks to the NameNode
  - Locates TaskTrackers near the data (which has already been populated)
- Moves scheduled work to the TaskTracker
  - JobTracker is updated via heartbeat
  - Failure of a task is detected through a missing heartbeat

TaskTracker

- Accepts tasks (Map, Reduce, Shuffle, etc) from JobTracker
- Each TaskTracker ahs a number of slots for tasks
  - These are execution slots available on the machine or rack
- It spawns a JVM for each task
- Indicates the number of available slots through the heartbeat message with the JobTracker