SWE404/DMT413
BIG DATA ANALYTICS

Lecture 2: Hadoop and HDFS

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

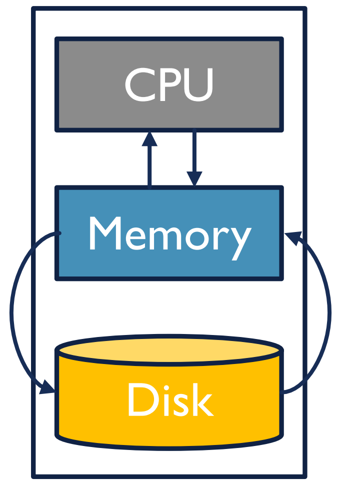
Office: A1-432

Office hour: 2pm-4pm Mon & Thur

hadoop

Traditional Data Processing

- Reading a word from disk vs. main memory: 10^5 slower.
- Normal disks only reach 150MB/s for sequential reads.
- IO Bounded: biggest performance bottleneck is reading / writing large files to disk.
  - 100 GBs ~ 10 minutes.
  - 200 TBs ~ 20,000 minutes = 13days.



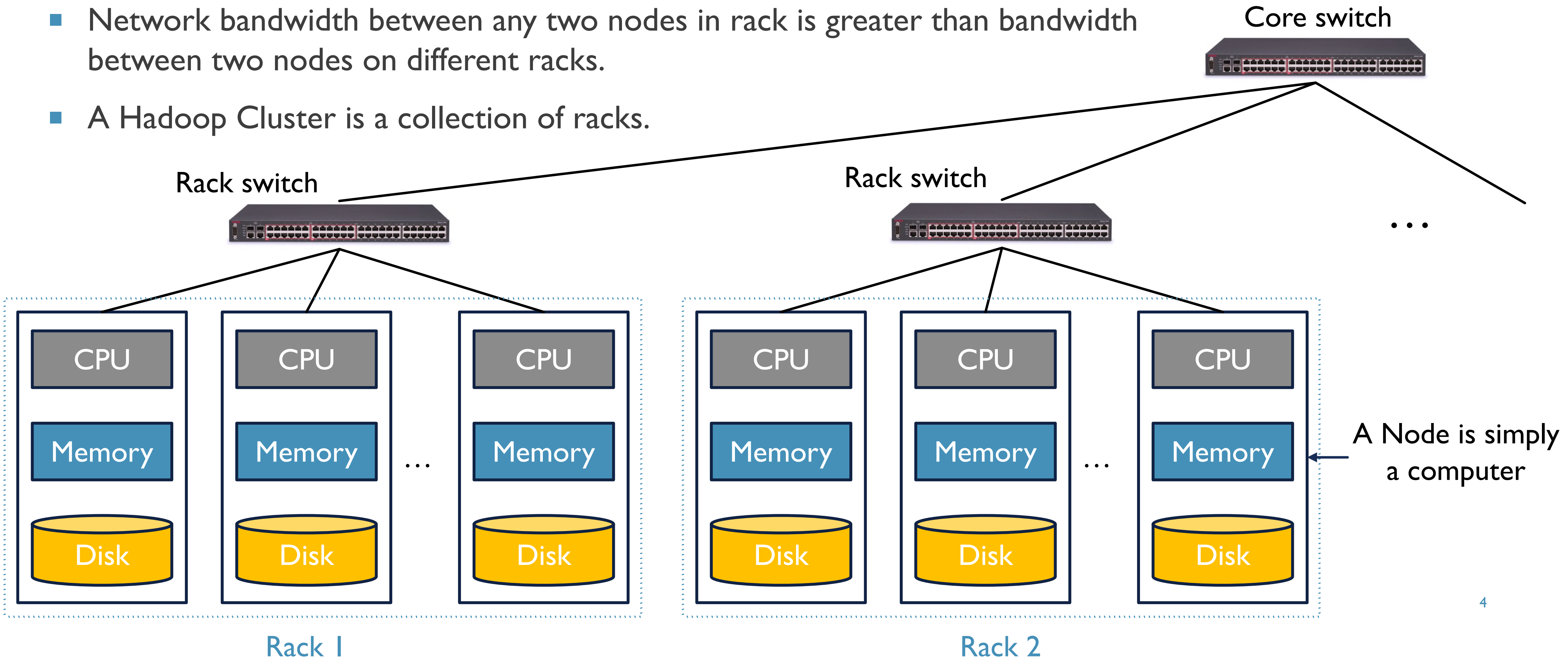
What is Hadoop?

Hadoop is named after a toy elephant belonging to Doug Cutting's son. Cutting is 'father' of Hadoop.

- Hadoop is a framework for storing data on large clusters of *commodity hardware* and running applications against that data.
  - No need for big and expensive servers.
  - Many affordable and easily available computers with single-CPU are tied together.
- A *cluster* is a group of interconnected computers (we call computers in the cluster as *nodes*) that can work together on the same problem.
- Using networks of affordable compute resources to acquire business insight is the key value proposition of Hadoop.

- A rack is a collection of 30 or 40 nodes that are physically stored close together and are all connected to the same switch.
- Network bandwidth between any two nodes in rack is greater than bandwidth between two nodes on different racks.
- A Hadoop Cluster is a collection of racks.

Switch: a networking hardware that connects devices on a computer network. (not Nintendo switch)



Challenges for IO Cluster Computing

ad-hoc: only provide specific solution for specific problem, can't be generalized.

- Node fail: averagely 1 in 1000 node fails a day.
  - Solution: duplicate data.
- Network bottleneck: typically 1-10 GB/s throughput.
  - Solution: bring computation to node, rather than data to node.
- Traditional distributed programming is often ad-hoc and complicated.
  - Solution: stipulate a programming system that can easily be distributed.

Hadoop accomplishes with two key components: HDFS and MapReduce

MapReduce

- MapReduce is a framework for parallel computing.
- It is usually composed of three operations:
  - **Map:** each node applies the map function to the local data, and writes the output to a temporary storage.
  - **Shuffle:** nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same node.
  - **Reduce:** nodes now process each group of output data, per key, in parallel.

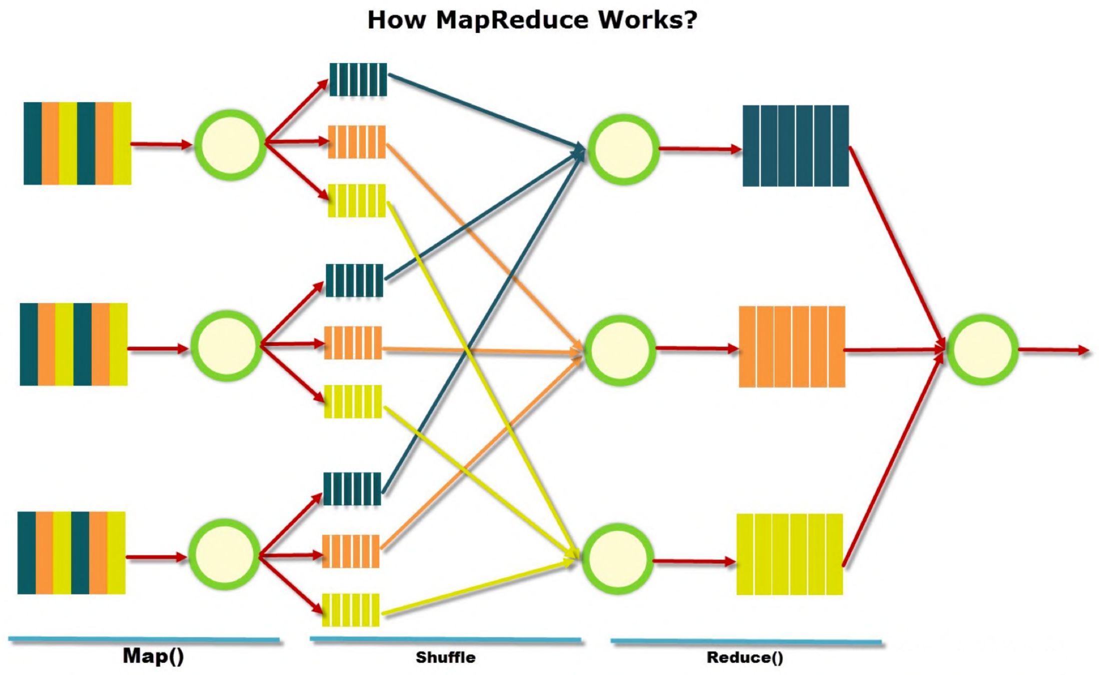


Image source: https://developerzen.com/introduction-to-mapreduce-for-net-developers-1030e070698a

Hadoop Key Features

- **Shared nothing architecture:** Hadoop is a cluster with independent machines (cluster with nodes), that every node perform its job by using its own resources.
- **Fault tolerance:** When data is sent to an individual node, that data is also replicated (usually x3) to other nodes in the cluster. In the event of failure, there is another copy available for use.
- **Commodity hardware:** Hadoop does not require a very high-end server with large memory and processing power. Hadoop runs on JBOD (just bunch of disk), so every node is independent in Hadoop.
- **Horizontal scalability:** We do not need to improve the power of nodes. As the data keeps on growing, we just keep adding nodes.

Hadoop Workflow

- Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs:
  - Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M.
  - These files are then distributed across various cluster nodes for further processing.
  - HDFS, being on top of the local file system, supervises the processing.
  - Blocks are replicated for handling hardware failure.
  - Checking that the code was executed successfully.
  - Performing the sort that takes place between the map and reduce stages.
  - Sending the sorted data to a certain computer.
  - Writing the debugging logs for each job.

Apache Hadoop Ecosystem

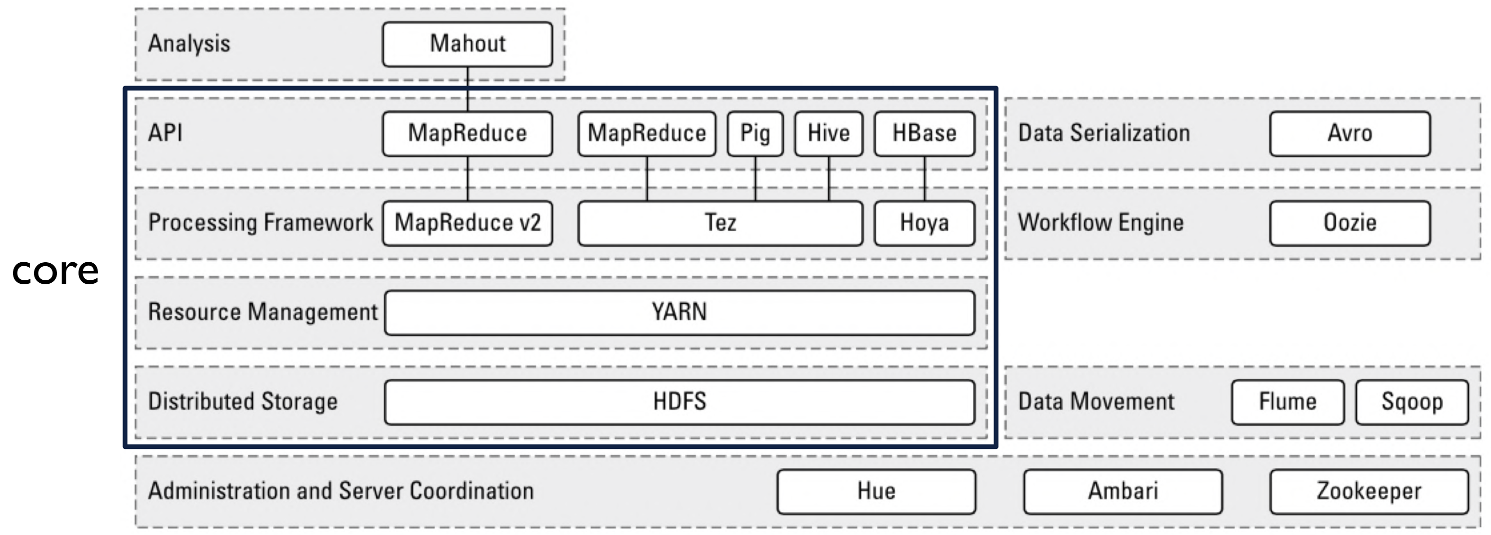


Image source: DeRoos, Dirk. *Hadoop for dummies*. John Wiley & Sons, 2014.

hadoop
HDFS

Traditional File System

- In the late 1990s, Google was facing the major challenge of having to be able to store and process all the pages on the Internet and Google users' web log data.
- At the time, Google was using a *scale-up* architecture model
  - increase system capacity by adding CPU cores, RAM, and disk to an existing server.
- Two major problems:
  - **Expense:** bigger servers with more storage was becoming incredibly expensive.
  - **Structural limitations:** the limits of what a scale-up architecture could sustain is easily reached.

Distributed File System

- Rather than scale *up*, Google engineers decided to scale *out* by using a cluster of smaller servers.
  - Continually add to if they needed more power or capacity.
  - Instead of "one big", use "many small".
- Google File System (GFS) was developed.
  - It was the inspiration for the engineers who first developed HDFS.

What You Can Do with Files in HDFS

- HDFS has a Write Once, Read Often model of data access.
- There's still lots you can do with HDFS files except modifying files:
  - create a new file,
  - append content to the end of a file,
  - delete a file,
  - rename a file,
  - modify file attributes like owner.

Data Blocks

- HDFS breaks a file down into a set of individual blocks and stores these blocks in different DataNodes in the Hadoop cluster.
- A file can be larger than any single disk in the network.
- The default block size is 128MB, but it can be customized.
  - 128MB is considered for data at the petabyte scale.
  - The block size in Linux is 4KB.
- Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's storage.

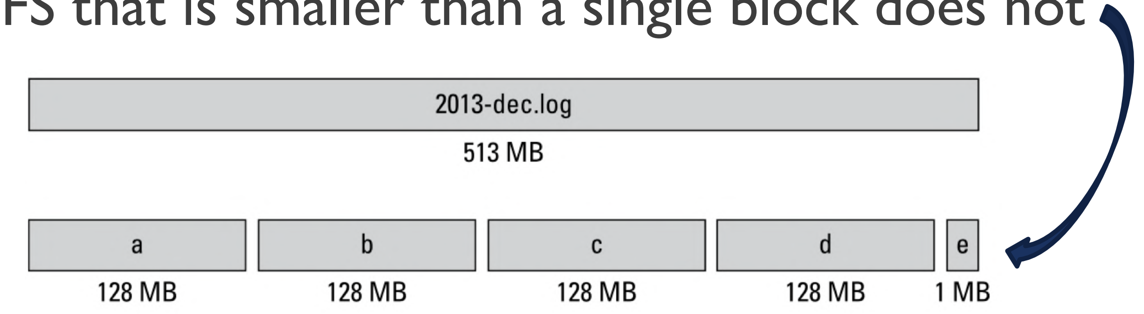
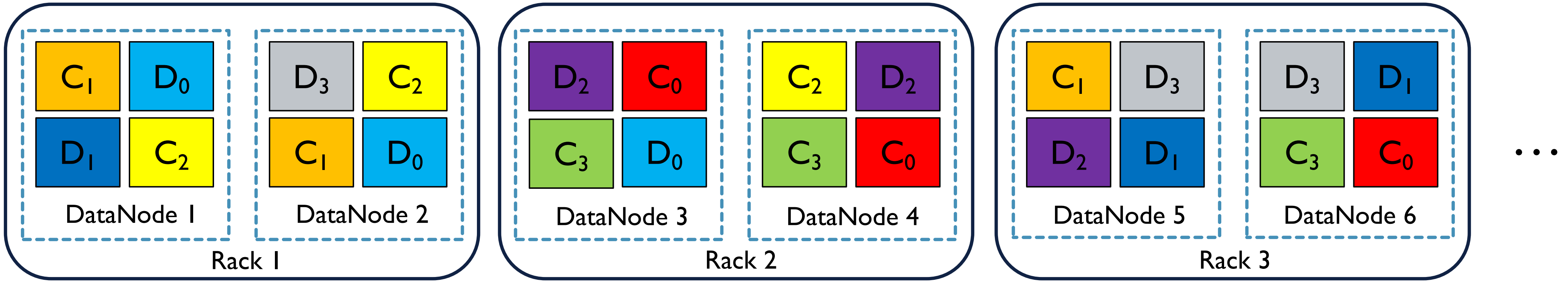


Image source: DeRoos, Dirk. *Hadoop for dummies.* John Wiley & Sons, 2014.

Data Block Replication

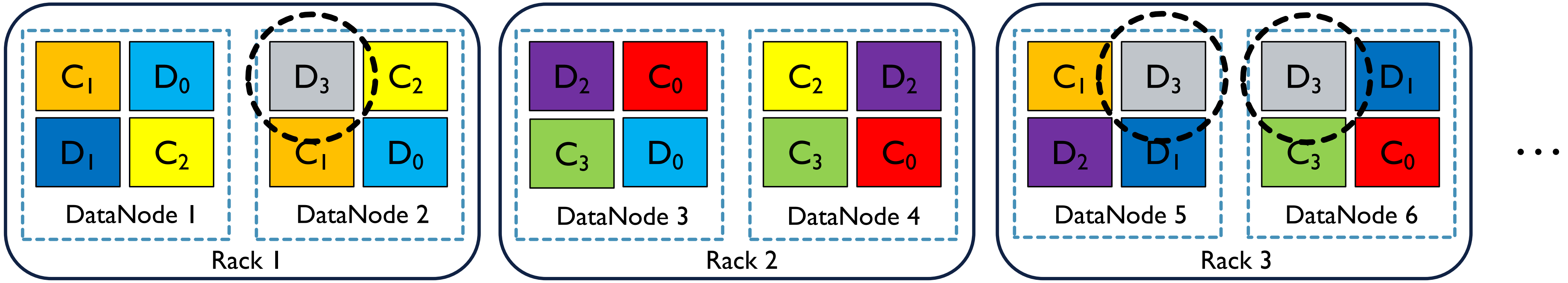
- HDFS is designed to store data on commodity hardware.
  - commodity = cheap = unreliable.
- Assuming that the individual nodes are unreliable, how can we make the cluster reliable?
  - Planning ahead for disaster, HDFS stores 3 copies of every data block in different places.
  - Bonus advantage: more opportunities for locating computation near the needed data.



C and D are two files with 4 blocks each

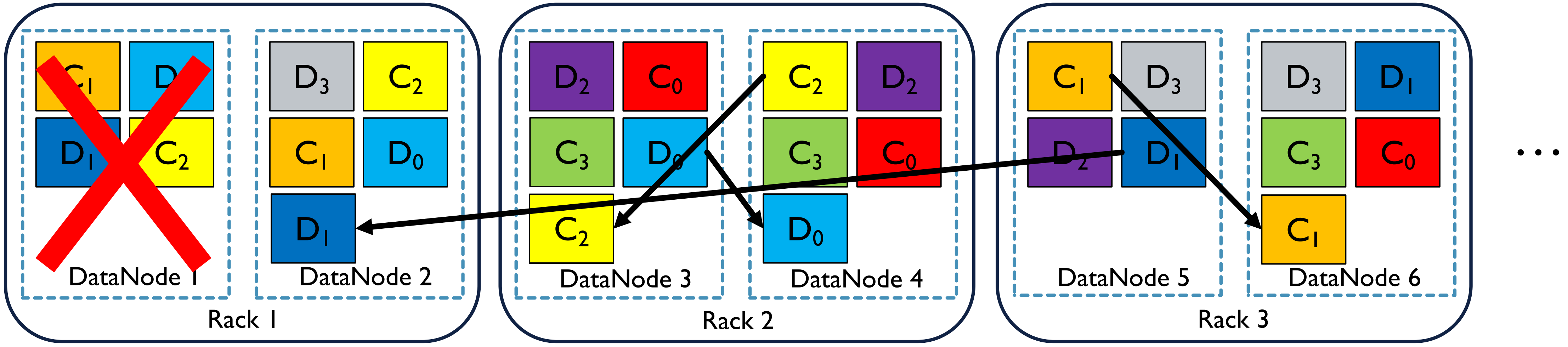
Block Placement

- The default HDFS replica placement policy follows:
  1. No DataNode contains more than one replica of any block.
  2. No rack contains more than two replicas of the same block.
- For example:
  - When D_3 is needed to be stored, DataNode 2 is chosen. The first copy of D_3 is stored there.
  - The remaining two copies of D_3 need to be stored in a different rack. So the second copy is stored on DataNode 5, Rack 3.
  - The final copy is stored on the same rack as the second copy, but not on the same DataNode, so it gets stored on DataNode 6.



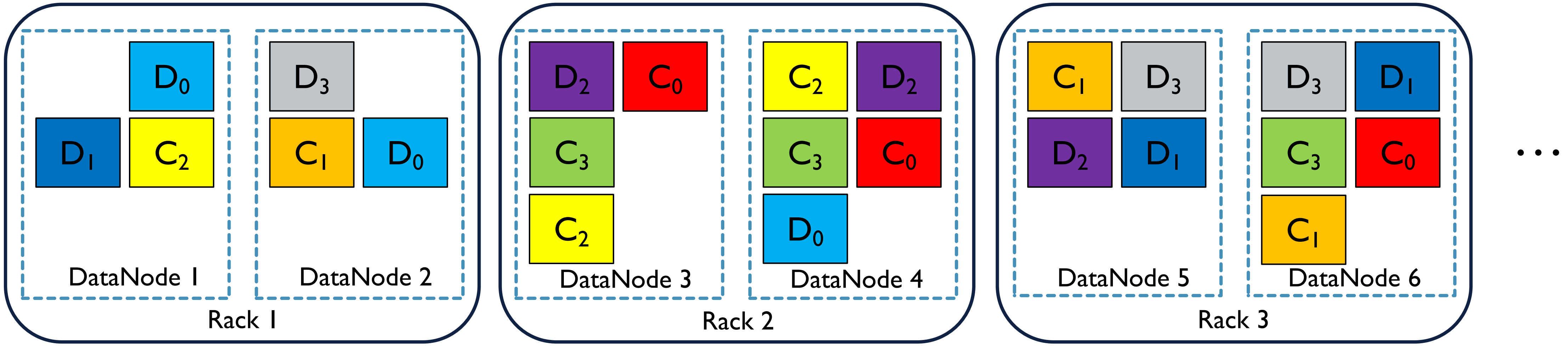
Data Block Replication

- When DataNode 1 fails, HDFS finds out that C_1, D_0, D_1, C_2 are *underreplicated*.
- It orders a new copy for them.
  - Still follow the block placement rule.



Data Block Replication

- When DataNode 1 comes back online after a few hours, HDFS finds out that C_1, D_0, D_1, C_2 are *overreplicated*.
- It orders one copy of them to be deleted.



Data Blocks

- As a Hadoop user, you have no idea which of the DataNode has the pieces of the file you need to process. All you see is a listing of files in HDFS.
- You don't know the complexity of how the file blocks are distributed, and you don't *need* to know.
- Actually, the DataNodes themselves don't even know what's inside the data blocks they're storing.
- Who's controlling?

NameNode: The Central Metadata Server for HDFS

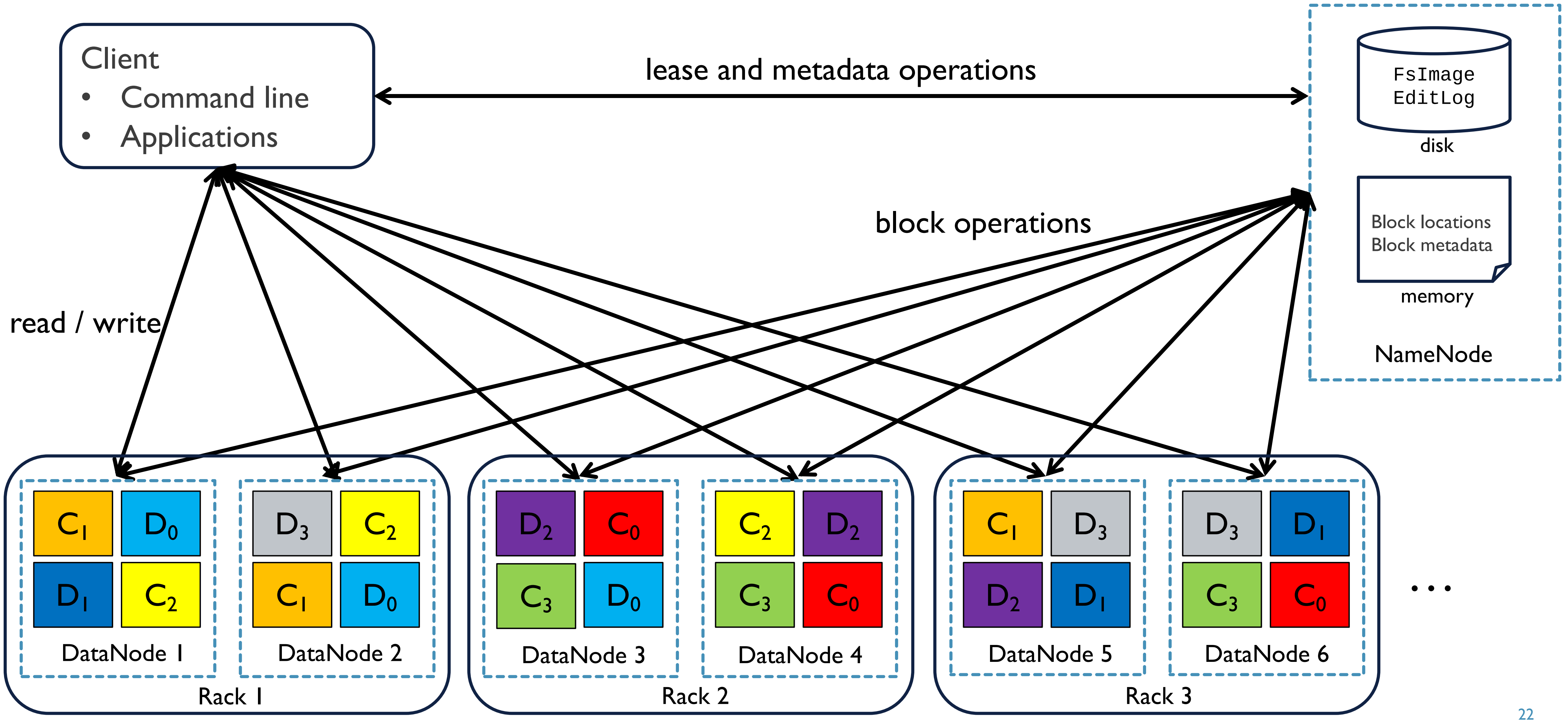
metadata: a set of data that describes and gives information about other data.

- The NameNode acts as the address book for HDFS because it knows:
  - which blocks make up individual files,
  - where each of these blocks and their replicas are stored.
- NameNode maintains two files:
  - `FsImage`: all mapping information dealing with the data blocks and their corresponding files.
  - `EditLog`: any data changes since the last checkpoint.

NameNode Startup and Operation

daemon: a computer program that runs as a background process

- To load all the information that the NameNode needs after it starts up, the following happens:
  - The NameNode loads the `FsImage` file into memory.
  - The NameNode loads the `EditLog` and re-plays the journaled changes to update the block metadata that's already in memory.
  - The DataNode daemons send the NameNode block reports.
- After the startup process, the NameNode has a complete picture of all the data stored in HDFS, and it's ready to receive application requests from Hadoop clients.



Interaction between HDFS components

Heartbeats and Block Report

How does the NameNode know that each DataNode works well?

- The DataNode daemons send the NameNode heartbeats (a quick signal) every three seconds (configurable), indicating they're active.
  - The NameNode marks DataNodes without recent (10 minutes) Heartbeats as dead and does not forward any new IO requests to them.
  - Any data that was registered to a dead DataNode is not available to HDFS any more.
- Every six hours (configurable), the DataNodes send the NameNode a block report outlining which file blocks are on their nodes.
- This way, the NameNode always has a current view of the available resources in the cluster.

Writing Data: Writer's Lease

1. The client sends a request to the NameNode to create a new file and is granted a *lease* for creating new file blocks in the cluster from the NameNode.
   - The writer's lease prevent other clients from writing the file, but not prevent reading.
     - HDFS implements a single-writer, multiple-reader model. A file may have many concurrent readers.
   - The writing client periodically renews the lease by sending a heartbeat to the NameNode.
   - The lease duration is bound by a *soft limit* and a *hard limit*.
     - If the soft limit expires and the client fails to close the file or renew the lease, another client can preempt the lease.
     - If after the hard limit expires (one hour) and the client has failed to renew the lease, HDFS automatically closes the file on behalf of the writer, and recover the lease.

Writing Data: Replication Pipelining

2. The NameNode allocates a block with a unique block ID and determines a list of DataNodes to host replicas of the block.
   - The replica selection algorithm follows the block placement rule.
   - The DataNodes form a *pipeline*, the order of which minimizes the total network distance from the client to the last DataNode.
   - Each DataNode in the pipeline
     - receives the data in packets
     - writes each packet to its local repository
     - transfers that packet to the next DataNode.

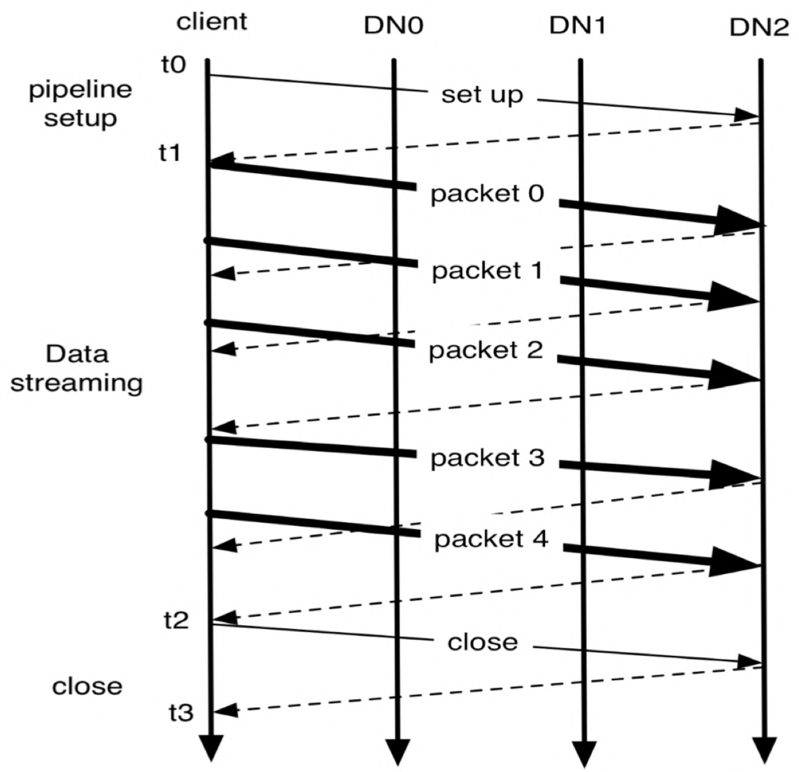


Image source: K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, 2010, pp. 1-10.

Writing Data: Acknowledge

3. After the DataNode daemons acknowledge the file blocks have been created, the client application closes the file and notifies the NameNode, which then closes the open lease.
   - The dash line in the figure is acknowledge.

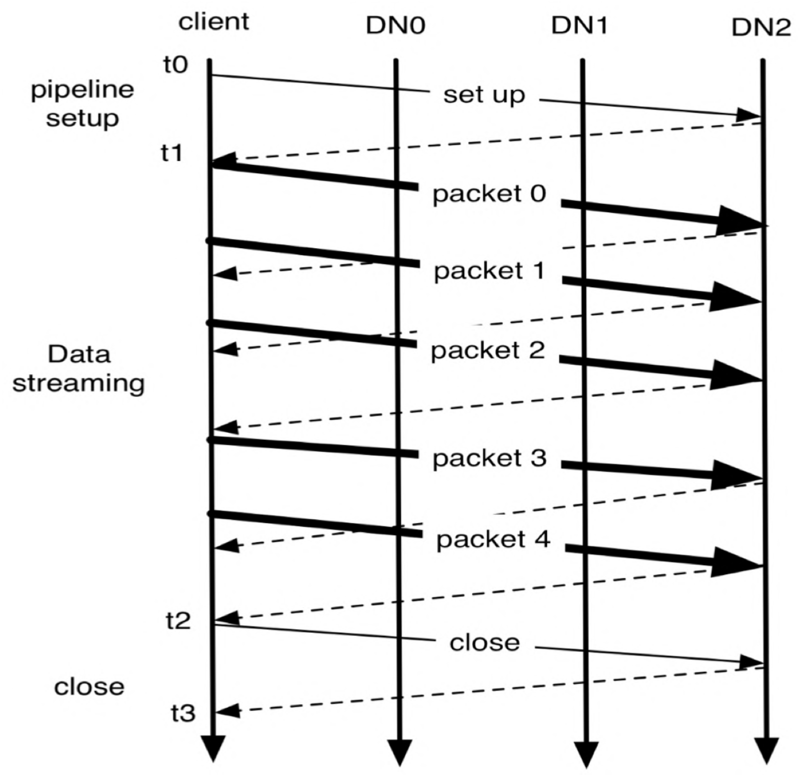


Image source: K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, 2010, pp. 1-10.

Reading Data

The client sends a request to the NameNode for a file.

- The NameNode determines the list of blocks and the locations of each block replica.
- The locations of each block are ordered by their distance from the reader.
- When reading the content of a block, the client tries the closest replica first. If the read attempt fails, the client tries the next replica in sequence.

Data Integrity

- It is possible that a block of data fetched from a DataNode arrives corrupted.
  - faults in a storage device, network faults, buggy software...
- The HDFS client software implements *checksum checking* on the contents of HDFS files.
  - When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace.
  - When a client retrieves file contents it verifies that the data matches the checksum.
  - If not, then the client can opt to retrieve that block from another DataNode that has a replica of that block.

Block Scanner

- Each DataNode runs a block scanner that periodically scans its block replicas and verifies that stored checksums match the block data.
  - If a client reads a complete block and checksum verification succeeds, it informs the DataNode. The DataNode treats it as a verification of the replica.
- Whenever a read client or a block scanner detects a corrupt block, it notifies the NameNode.
  - The NameNode does not delete the corrupted replica immediately, until the good replica count reaches the replication factor (e.g. x3).
  - This policy aims to preserve data as long as possible. So even if all replicas of a block are corrupt, the policy allows the user to retrieve its data from the corrupt replicas.

Rebalancing

- Over time, data is likely to become unevenly distributed across the racks and DataNodes.
- This uneven distribution can have a detrimental impact on performance because the demand on individual DataNodes will become unbalanced.
  - Nodes with little data won't be fully used.
  - Nodes with many blocks will be overused.
- A cluster is balanced if for each DataNode the following condition is met:

$$\left|\frac{used\ space\ at\ the\ node}{total\ capacity\ of\ the\ node} - \frac{used\ space\ in\ the\ cluster}{total\ capacity\ of\ the\ cluster}\right| > threshold$$

- It iteratively moves replicas from DataNodes with higher utilization to DataNodes with lower utilization.
  - Also follow the block placement rule.

Decommissioing

- The cluster administrator has a list of the host addresses of nodes that are permitted to register.
- When a DataNode is removed from the list, the process is:
  - The DataNode is marked for decommissioning.
  - It will not be selected as the target of replica placement, but it will continue to serve read requests.
  - The NameNode starts to schedule replication of its blocks to other DataNodes.
  - Once the NameNode detects that all its blocks are replicated, the node is safely removed from the cluster without jeopardizing any data availability.

Checkpoint

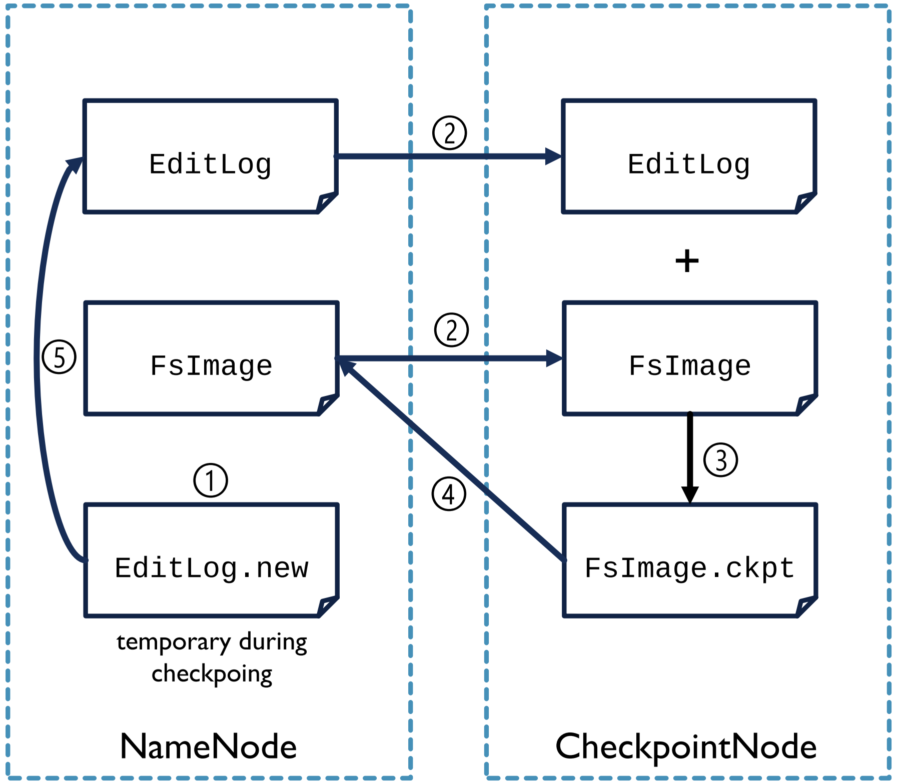
We have block replicas to protect DataNodes. How can we protect the NameNode?

- Creating periodic checkpoints is one way to protect the file system metadata.
- The system can start from the most recent checkpoint if all other persistent copies of `FsImage` or `EditLog` are unavailable.
- A checkpoint can be triggered
  - at a given time interval expressed in seconds,
  - or after a given number of filesystem transactions have accumulated.
- If both of these properties are set, the first threshold to be reached triggers a checkpoint.

CheckpointNode

The process of checkpoint:

1. The NameNode creates a new file `EditLog.new` to accept the journaled file system changes.
2. As a result, `EditLog` accepts no further changes and is copied to the CheckpointNode, along with the `FsImage` file.
3. The CheckpointNode merges these two files, creating a file named `FsImage.ckpt`.
4. The CheckpointNode copies the `FsImage.ckpt` file to the NameNode. The NameNode overwrites the file `FsImage` with `FsImage.ckpt`.
5. The NameNode renames the `EditLog.new` file to `EditLog`.



BackupNode

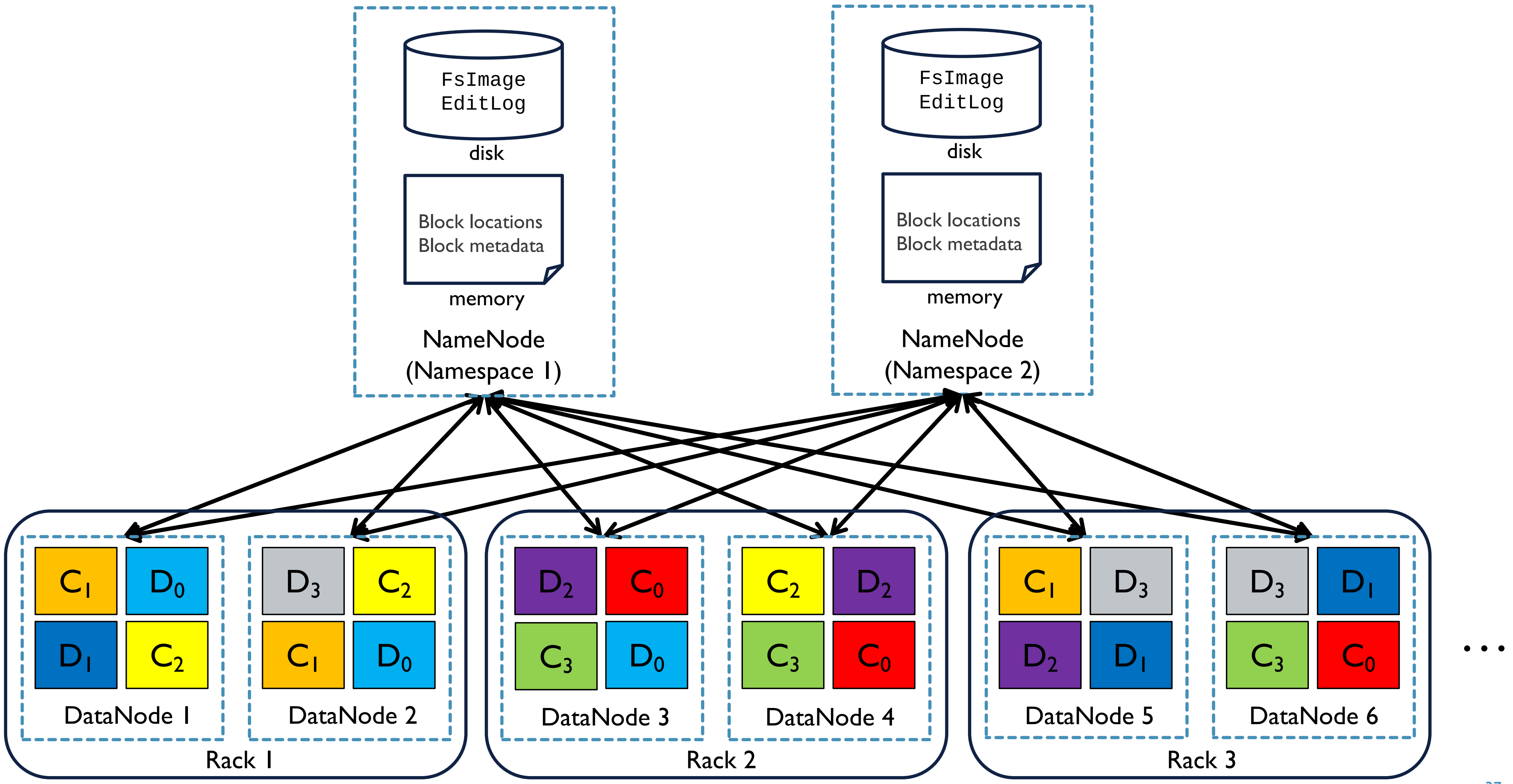
- The BackupNode provides the same functionality as the Checkpoint Node, but is synchronized with the NameNode.
- The BackupNode maintains a real-time backup of the NameNode's state.
- As a result of keeping the block metadata in memory, the BackupNode is far more efficient than the CheckpointNode at performing the checkpointing task.
  - `FsImage` and `EditLog` don't need to be transferred and merged. These changes are already merged in memory.

High Availability

- CheckpointNode and BackupNode protect against data loss, but they do not provide high availability of the filesystem.
  - The whole cluster would be unavailable until the NameNode was recovered.
- Hadoop 2 remedied this situation by adding support for HDFS *High Availability (HA)*.
  - A pair of NameNodes in an *active-standby* configuration.
  - When the active NameNode fails, the standby takes over its duties to continue servicing client requests without a significant interruption.

Federation

- Before Hadoop 2, the NameNode placed *limits* on the degree to which they could scale.
  - Few clusters were able to scale beyond 3,000 or 4,000 nodes.
  - The NameNode needs to maintain records for every block of data stored in the cluster.
  - It becomes increasingly difficult for the NameNode to scale up as the Hadoop cluster scales out.
- The solution to expanding Hadoop clusters indefinitely is to *federate* the NameNode.
  - Multiple NameNode instances run on their own, responsible only for the file blocks in its own name space.
  - NameNodes do not communicate with one another.



A Hadoop cluster with two NameNodes serving a single cluster

Benefits of Federation

- **Namespace Scalability**: Federation adds namespace horizontal scaling.
  - Large deployments or deployments using lot of small files benefit from namespace scaling by allowing more NameNodes to be added to the cluster.
- **Performance**: File system throughput is not limited by a single NameNode.
  - Adding more NameNodes to the cluster scales the file system read/write throughput.
- **Isolation**: A single NameNode offers no isolation in a multi user environment.
  - For example, an experimental application can overload the NameNode and slow down production critical applications.
  - By using multiple Namenodes, different categories of applications and users can be isolated to different namespaces.

Conclusion

After this lecture, you should know:

- The purpose of HDFS.
- How data is stored in HDFS.
- What are the jobs of DataNode and NameNode.
- How HDFS deals with failures.
- How to write/read data with HDFS.

Thank you!

Reference (recommend for further reading):

- **The official website**: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
- **The paper**: K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, 2010, pp. 1-10.
- **The book**: Chapter 4, DeRoos, Dirk. *Hadoop for dummies*. John Wiley & Sons, 2014.

Acknowledgement: Thankfully acknowledge slide contents shared by Prof. Yongxuan Lai and Dr. Ye Luo

XIAMEN UNIVERSITY MALAYSIA 厦門大學馬來西亞分校

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY

厦门大学计算机科学系 Computer Science Department of Xiamen University