CPSC 4770/6770

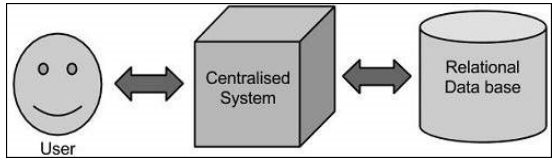
Distributed and Cluster Computing

Lecture 12: Hadoop Distributed File System

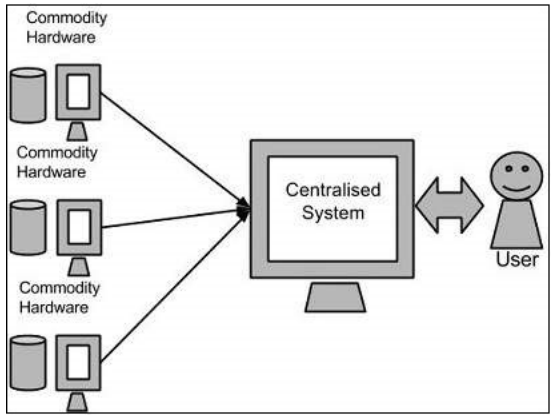
Overview

- 2002: Doug Cutting and Mike Carafella started a project to build an open-source search engine called Nutch. A component of this project was a web crawler that can crawl and index the Internet
- 2003: Google released a research paper on its in-house data storage system called Google File System (GFS)
- 2004: Google released another research paper on the programming approach to process data stored on GFS, called MapReduce
- 2005: Cutting and Carafelle rebuilt the underlying file management system and processing framework of Nutch based on the architectural design of Google's GFS and MapReduce
- 2006: The adaptations of Google's GFS and MapReduce were converted into a single open source project called **Hadoop**, which was sponsored by Yahoo and led by Doug Cutting
- 2007: Yahoo maintains a 1000-node production cluster
- 2008: Hadoop becomes the platform of Yahoo's web index. Hadoop wins record for world fastest system to sort one terabyte of data (209 seconds using a 910-node cluster). Hadoop becomes a top-level open source project of Apache Foundation. First Hadoop commercial distributor led by a former Google employee, Cloudera, is founded
- 2009: Hadoop sorts one terabyte of data in 62 seconds and one petabyte of data in 16.25 hours using a 3800-node cluster. Second Hadoop commercial distributor, MapR, is formed
- 2011: Yahoo spins off its own Hadoop commercial distributor, Hortonworks
- 2012: Apache Hadoop 1.0 is released

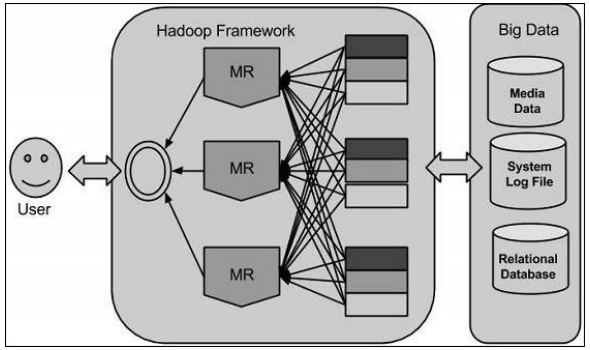
Tradition Approach



Googles Solution - MapReduce



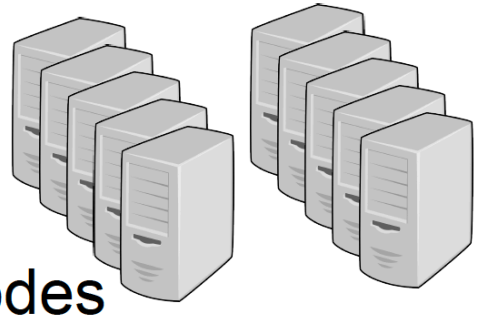
Hadoop



Why a Distributed File System?

- 1 node
- 4 Hard drive (I/O Channels)
- 100 MB/sec I/O bus per channel

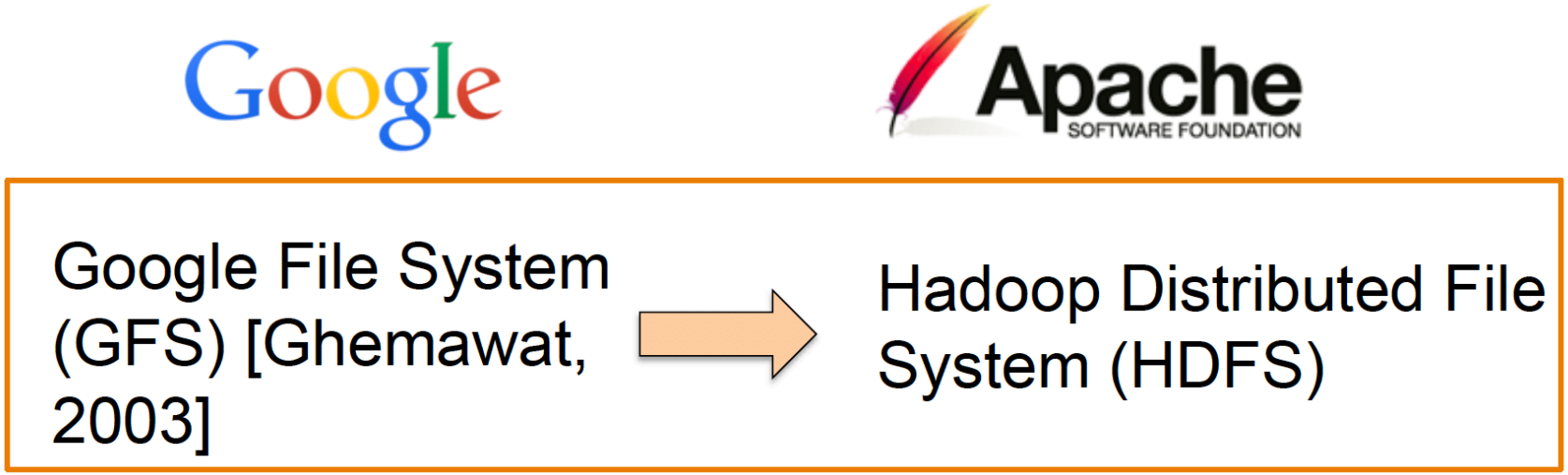
- 1TB file takes 45 minutes to read



- 10 nodes
- 4 Hard drives (I/O channels) per node
- 100 MB/sec I/O bus per channel

- 1TB file takes 4.5 minutes to read

Google and Apache Software Linkage



GFS and HDFS Terms

Google	hadoop HDFS
• GFS Master	• NameNode
• Chunkserver	• DataNode
• Client	• Client
• Chunks	• Blocks

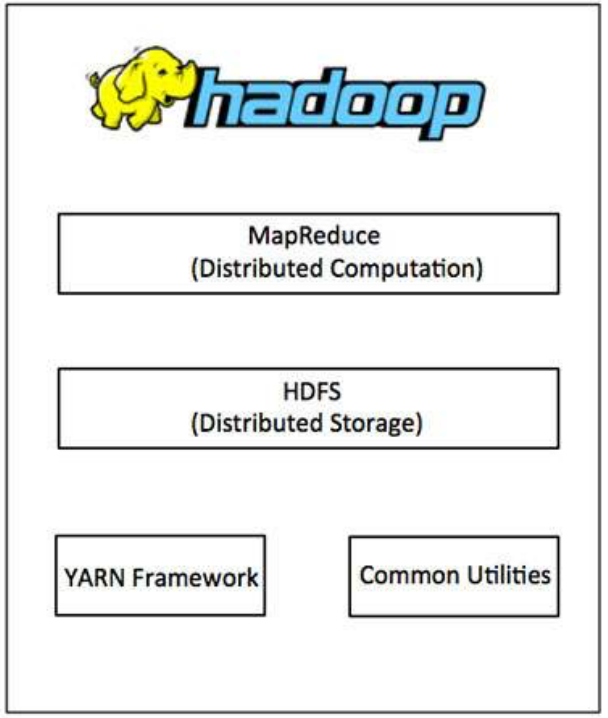
Master– Single (similar to JobTracker)

Slave – Multiple (similar to TaskTracker)

GFS Paper provides architecture but no C code.
HDFS implements this architecture in Java code base

Apache Hadoop Project

- **Hadoop Distributed File System (HDFS)**
- YARN (Yet Another Resource Negotiator)
- Hadoop MapReduce



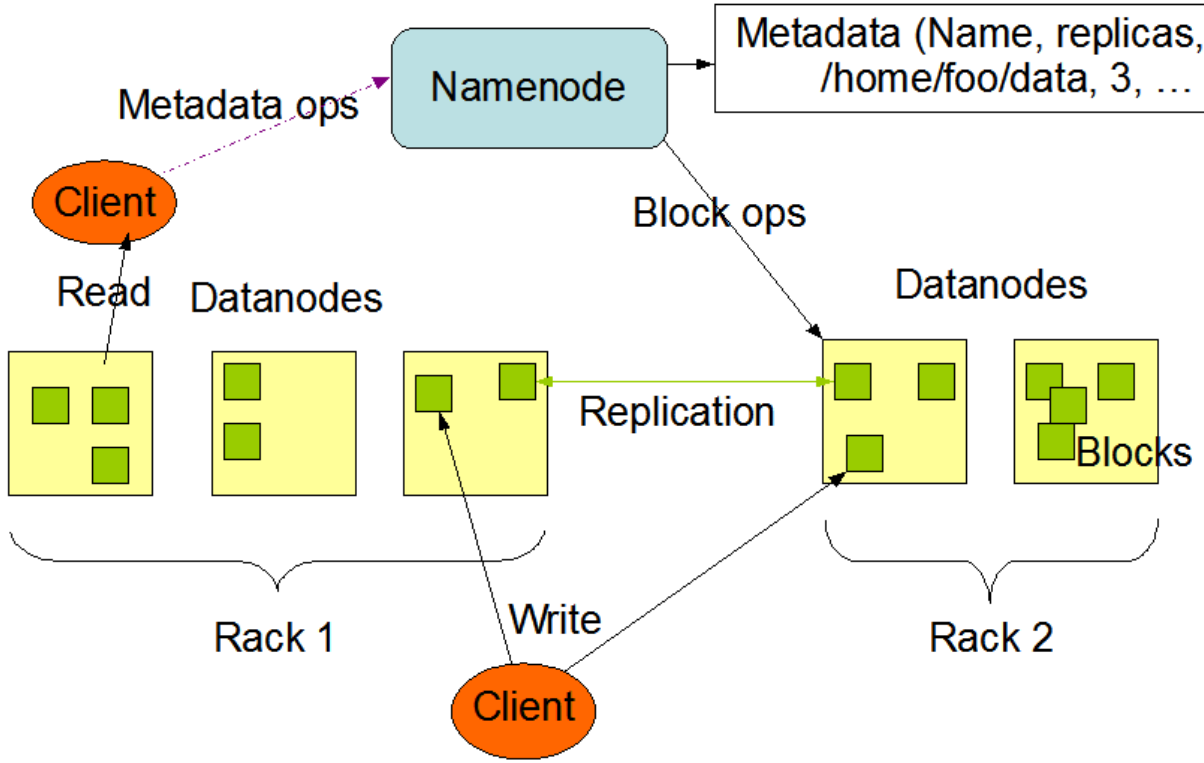
Design Assumptions and Goals

- Hardware failure is the norm rather than the exception
- Streaming data access
  - Not for general purpose applications
  - For batch processing rather than interactive use
  - For high throughput of data access rather than low latency of data access
- Large data sets (terabytes in size)
- Simple coherency model (write once read many)
- Moving computation is cheaper than moving data
- Portability across heterogeneous hardware and software platform

HDFS Architecture

- **Master/Slave Architecture**
  - Master: NameNode
  - Workers: DataNode

HDFS Architecture



NameNode

- manages the file system namespace
- regulates access to files by clients
- executes file system namespace operations
- determines the mapping of blocks to DataNodes
- keeps track of DataNodes status

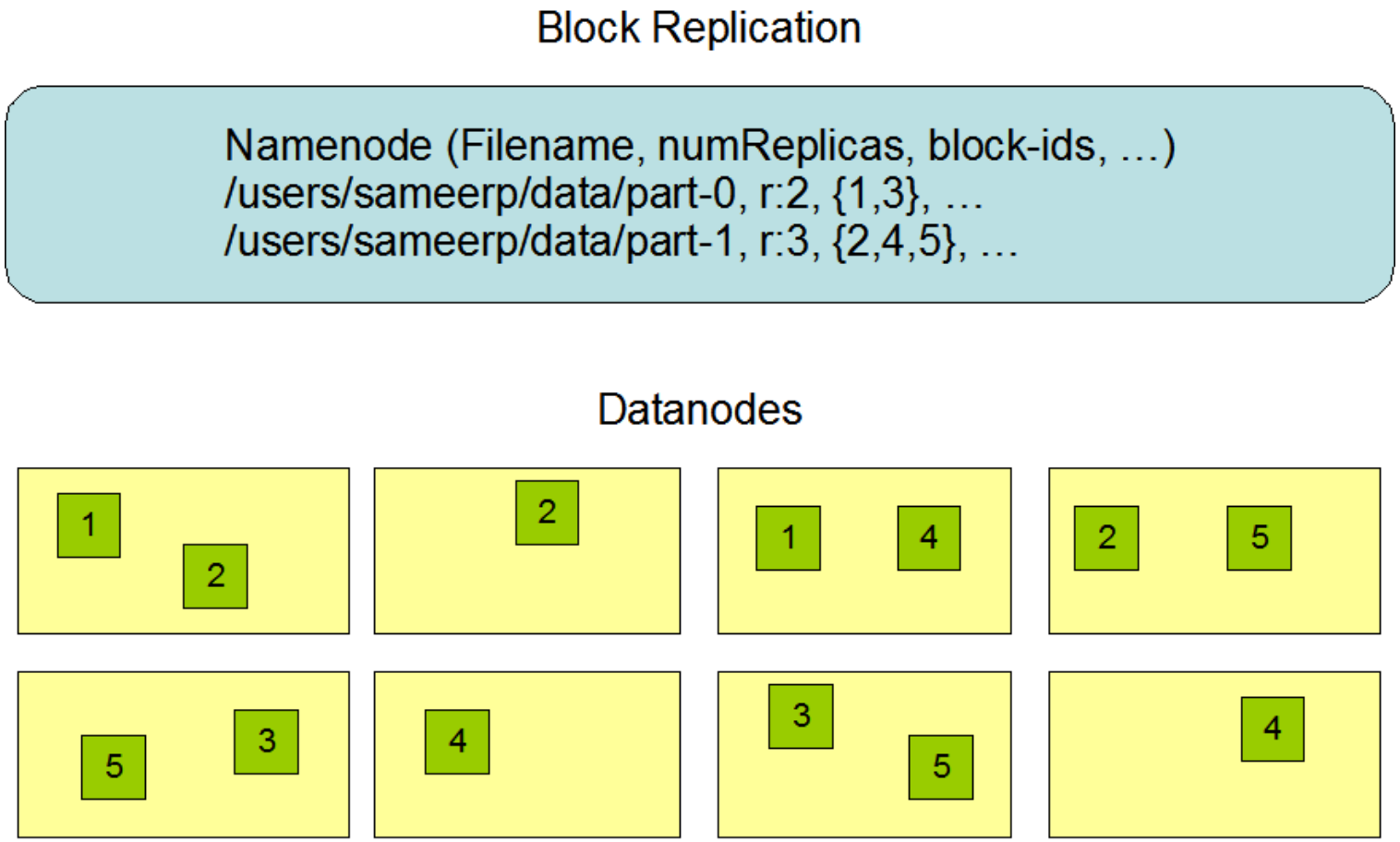
DataNode:

- one per node in the cluster
- manages storage attached to the node
- serves read and write requests clients
- sends regular heart-beat messageas back to the NameNode for verification

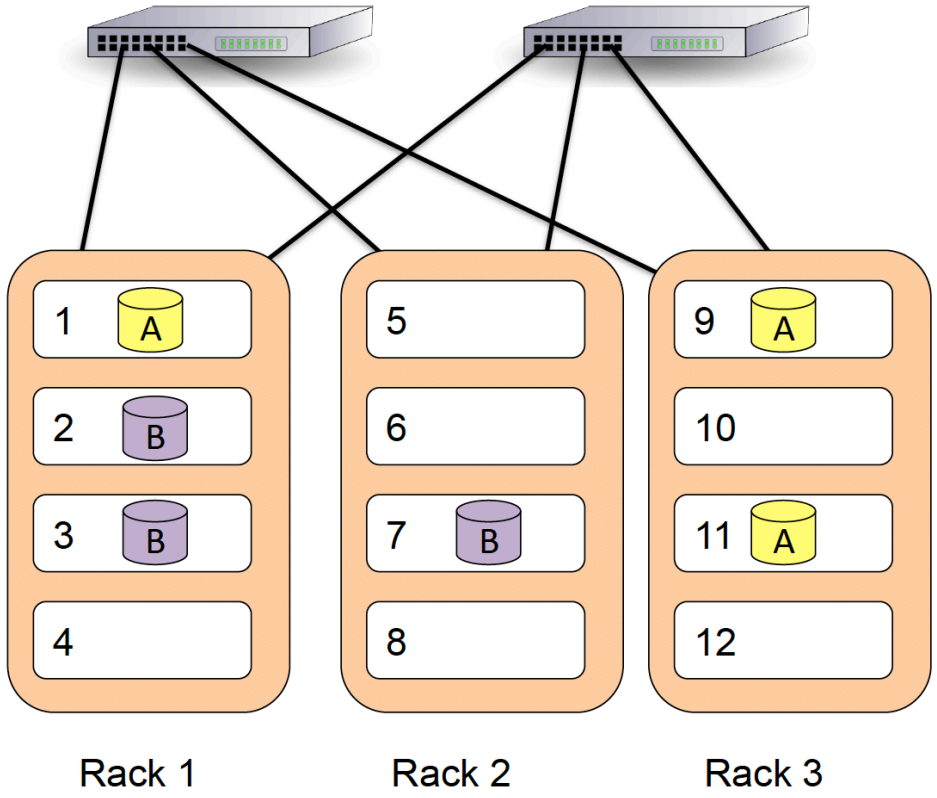
Files and Directories:

- HDFS file system namespace is exposed to users
- Operations include opening, closing, and renaming files and directories.
- HDFS allows user data to be stored in files

Block Replication



Rack Awareness and Replication Placement Policy



NameNode:

metadata
file1.txt =

Block A
DN: 1, 9, 11

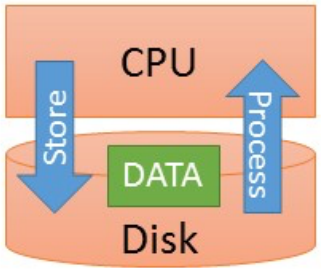
Block B
DN: 2, 3, 7

Rack Awareness
Rack 1: 1,2,3,4
Rack 2: 5,6,7,8
Rack 3: 9,10,11,12

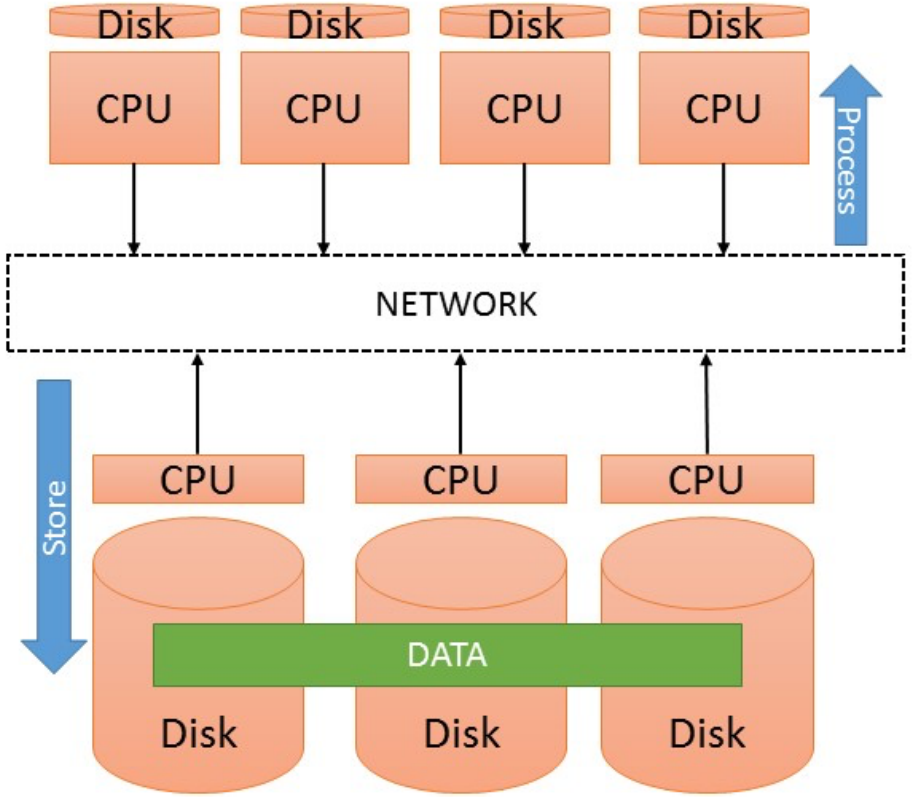
Placement Policy: HDFS default policy

- Default replication factor: 3
  - First replica on one node in the local rack,
  - Second replica on a different node in the local rack,
  - Third replica on a different node in a different rack.
- Replication factor greater than 3:
  - Random determination of placement
  - Rack limit: $\frac{replicas - 1}{racks + 2}$

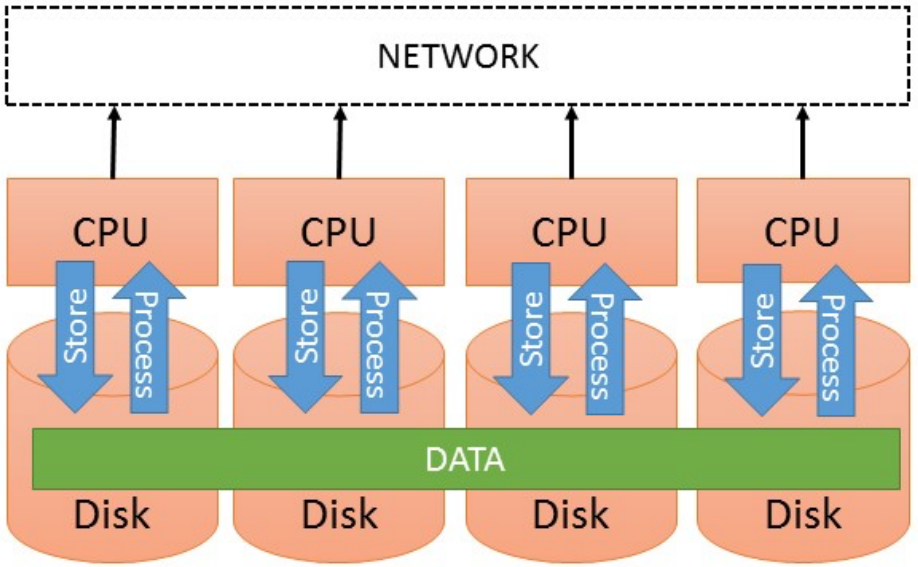
Comparing File System Models



Single computer storage/process model with data movements happen via internal connection

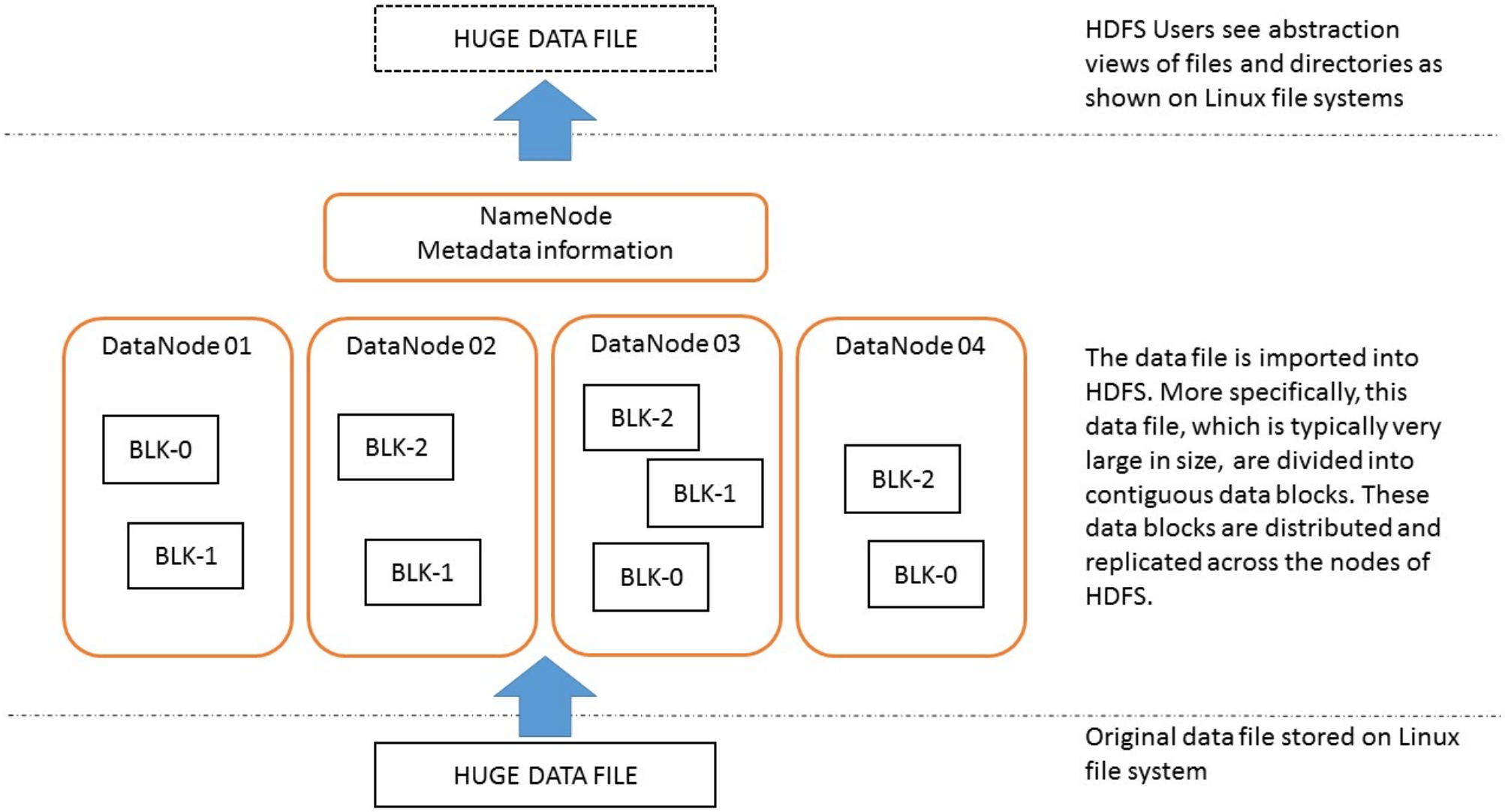


Remote storage model with large amount of data stored on specialized storage system. Data must be moved across the network to computers running the actual applications.



Data-locality storage model where the applications are co-located with the data to be processed. Data are still being transferred across the network to facilitate applications that use massive amount of data, but the demand for network is much less due to the processing steps that already happened on each individual computers prior to the actual data movement.

HDFS

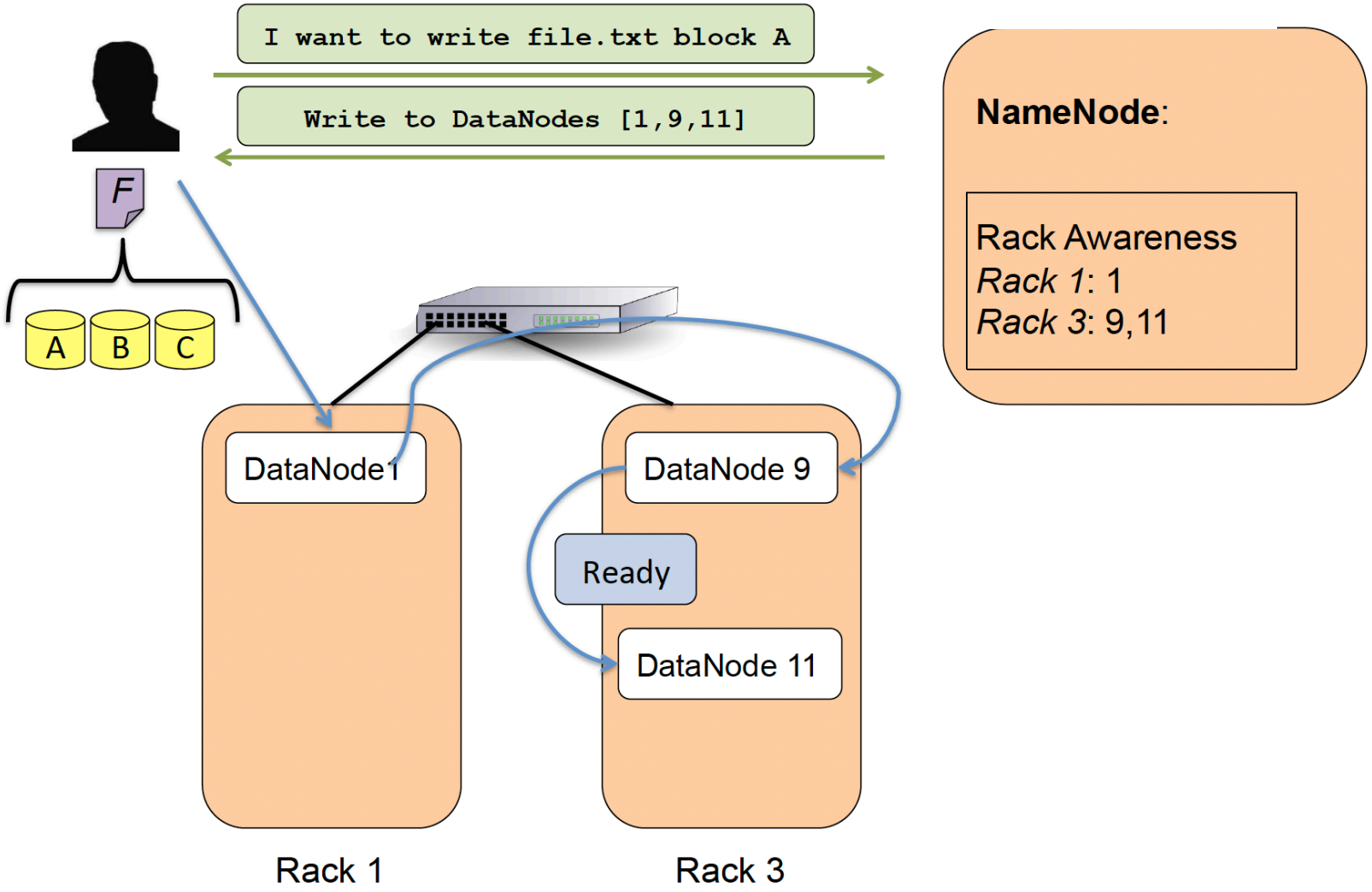


HDFS Users see abstraction views of files and directories as shown on Linux file systems

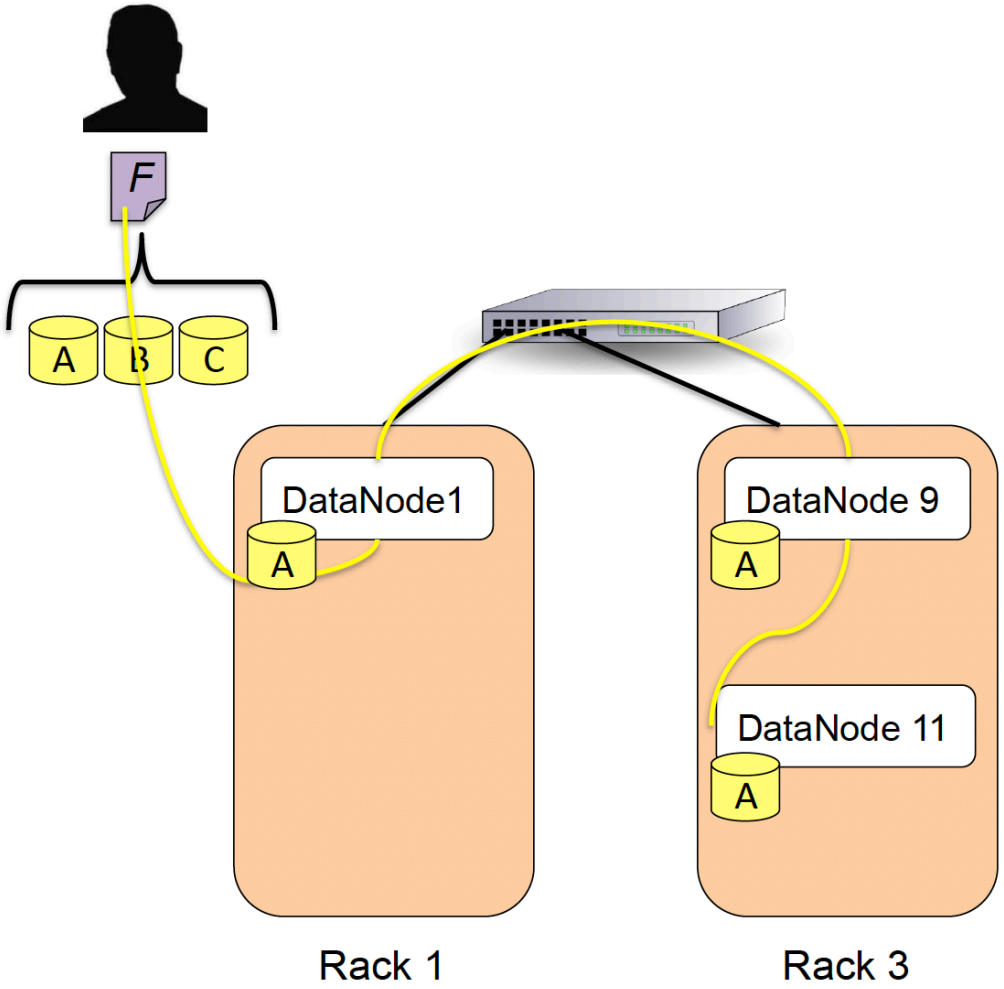
The data file is imported into HDFS. More specifically, this data file, which is typically very large in size, are divided into contiguous data blocks. These data blocks are distributed and replicated across the nodes of HDFS.

Original data file stored on Linux file system

HDFS Writes [Prep]



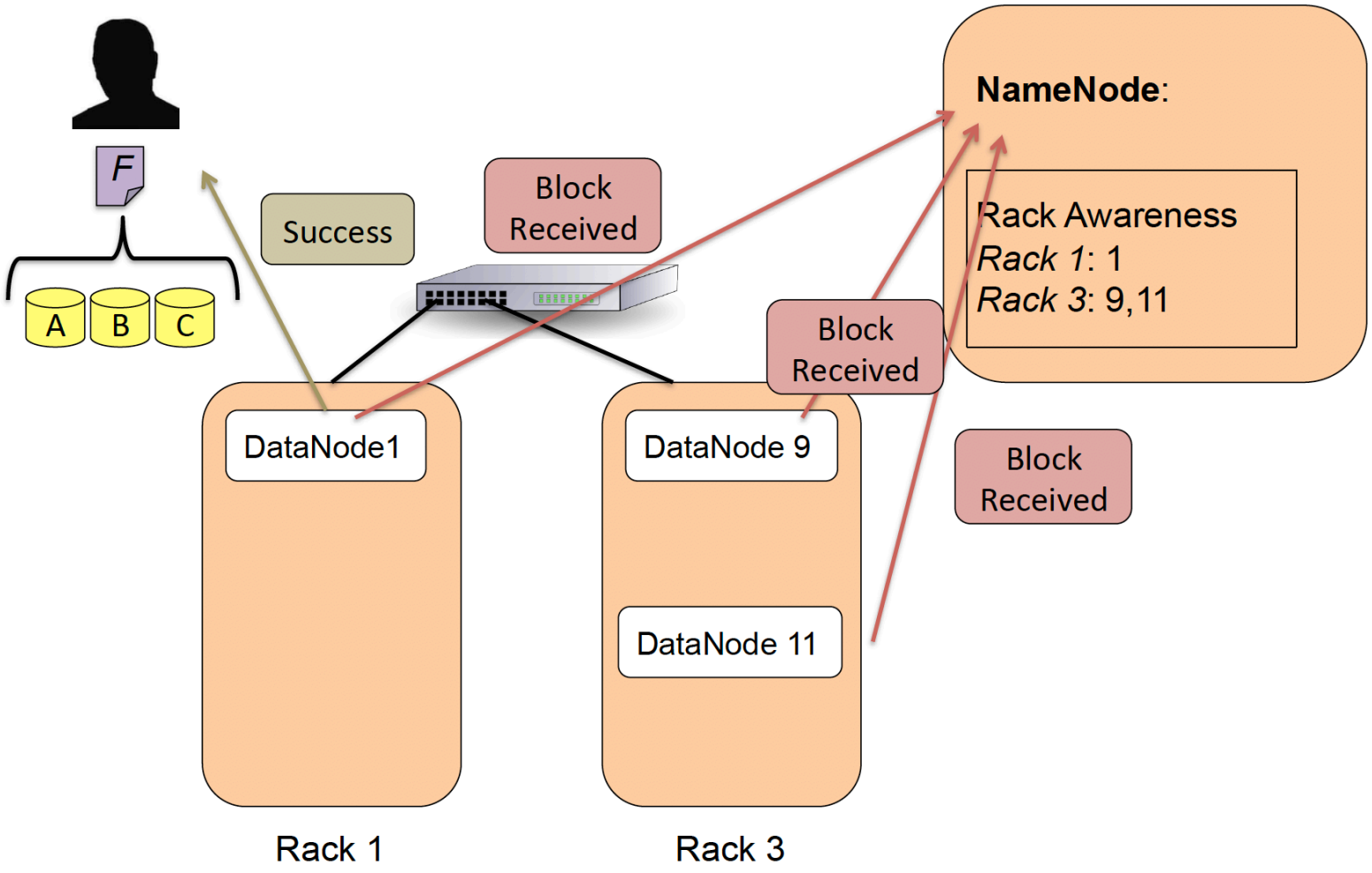
HDFS Writes [Pipeline]



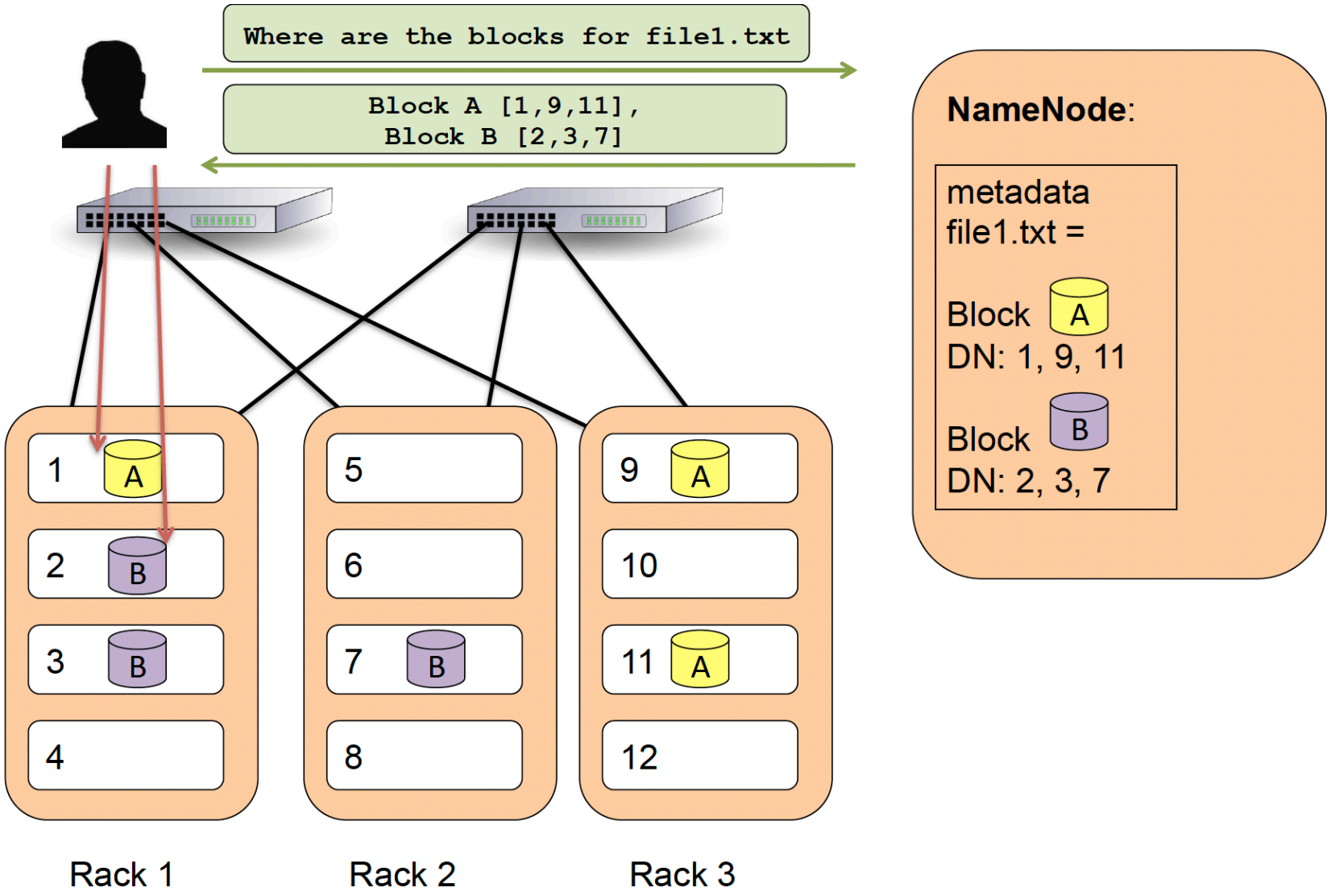
NameNode:

Rack Awareness
Rack 1: 1
Rack 3: 9,11

HDFS Writes [Ack]

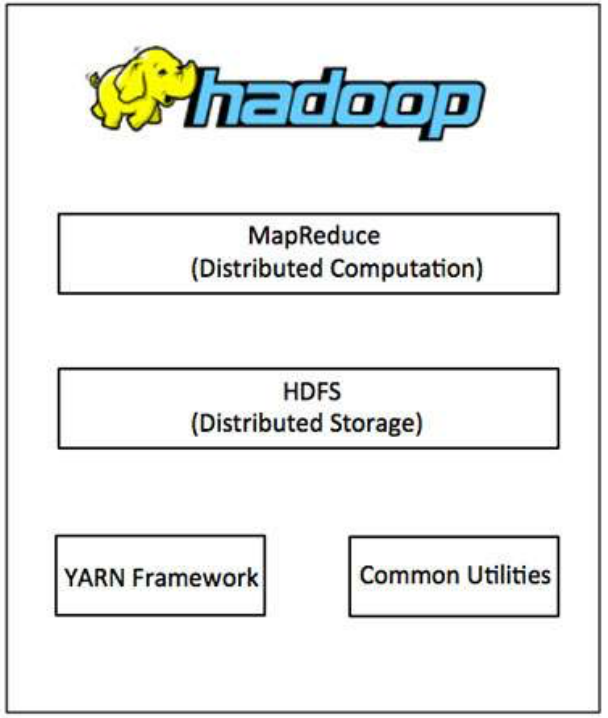


HDFS Reads


Where are the blocks for file1.txt
Block A [1,9,11],
Block B [2,3,7]
NameNode:
metadata
file1.txt =
Block A
DN: 1, 9, 11
Block B
DN: 2, 3, 7
1
A
2
B
3
B
4
5
6
7
B
8
9
A
10
11
A
12
Rack 1
Rack 2
Rack 3

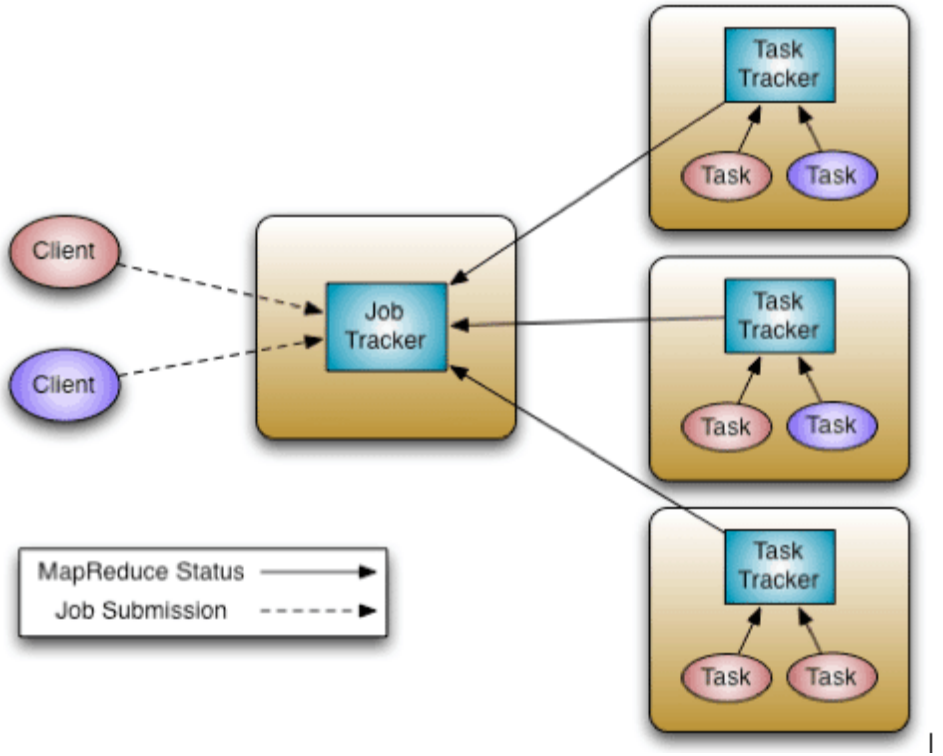
Apache Hadoop Project

- Hadoop Distributed File System (HDFS)
- **YARN (Yet Another Resource Negotiator)**
- Hadoop MapReduce



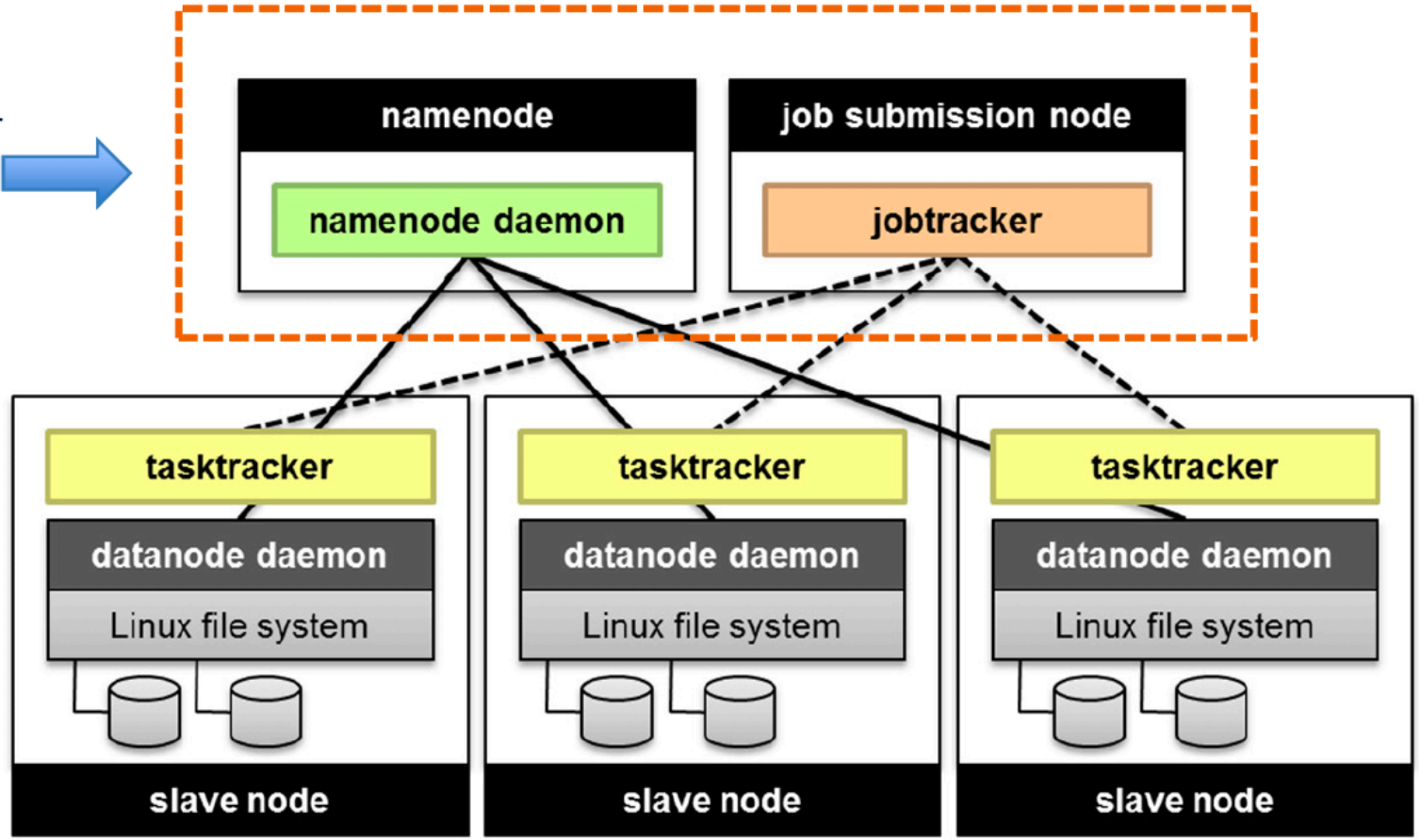
First Resource Manager/Job Controller on Hadoop MapReduce System

- Specifically tied to the MapReduce programming model
- Data-aware (via NameNode interactions)
- Task-aware (via TaskTracker interactions)
- Rely on **heartbeat** messages, which also contain slot availability information from TaskTrackers:
  - System health
  - Execution availability
  - Job progress
- Manage job execution process
- Rerun tasks lost due to node failures – Speculative execution
- Single point of failure



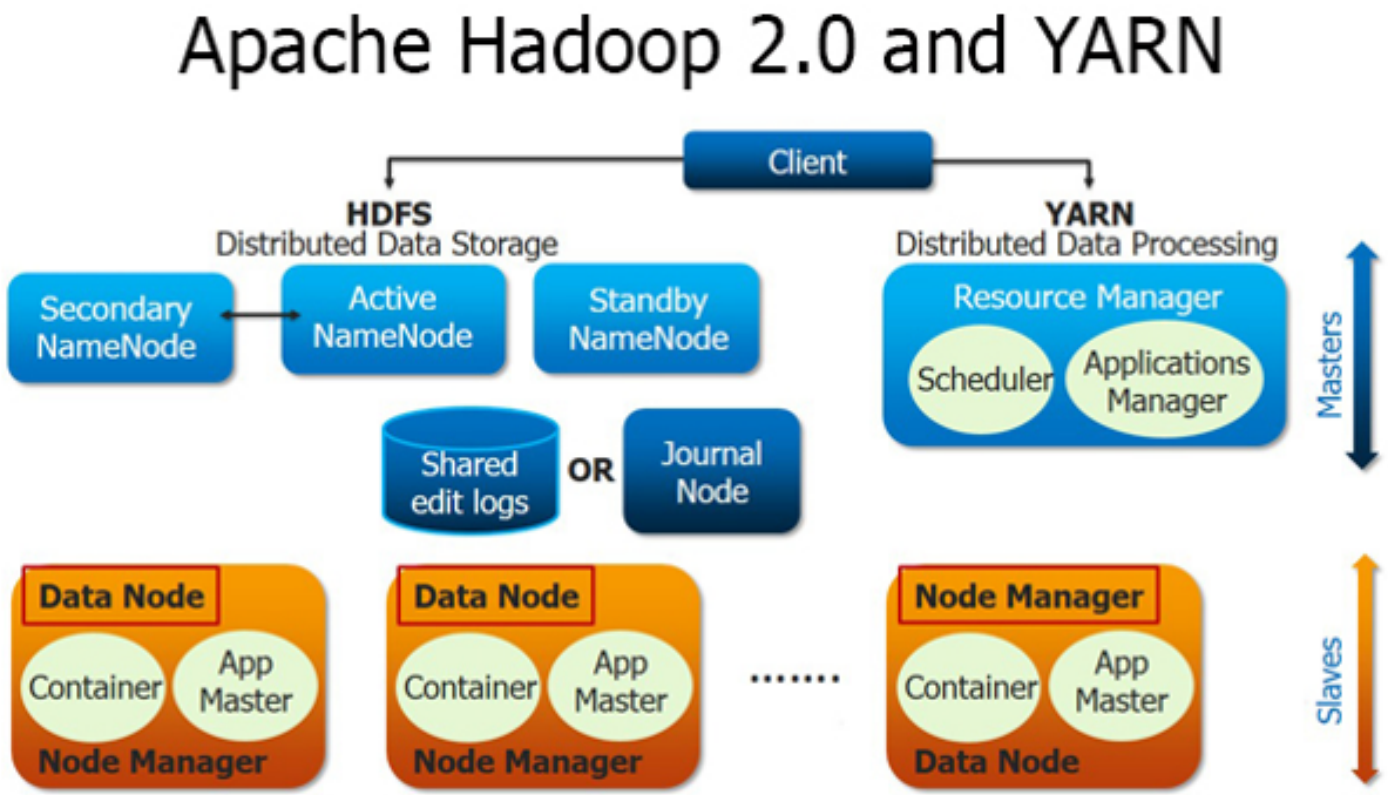
First Resource Manager/Job Controller on Hadoop MapReduce System (Cont.)

NameNode and JobTracker can be placed on the same physical machine

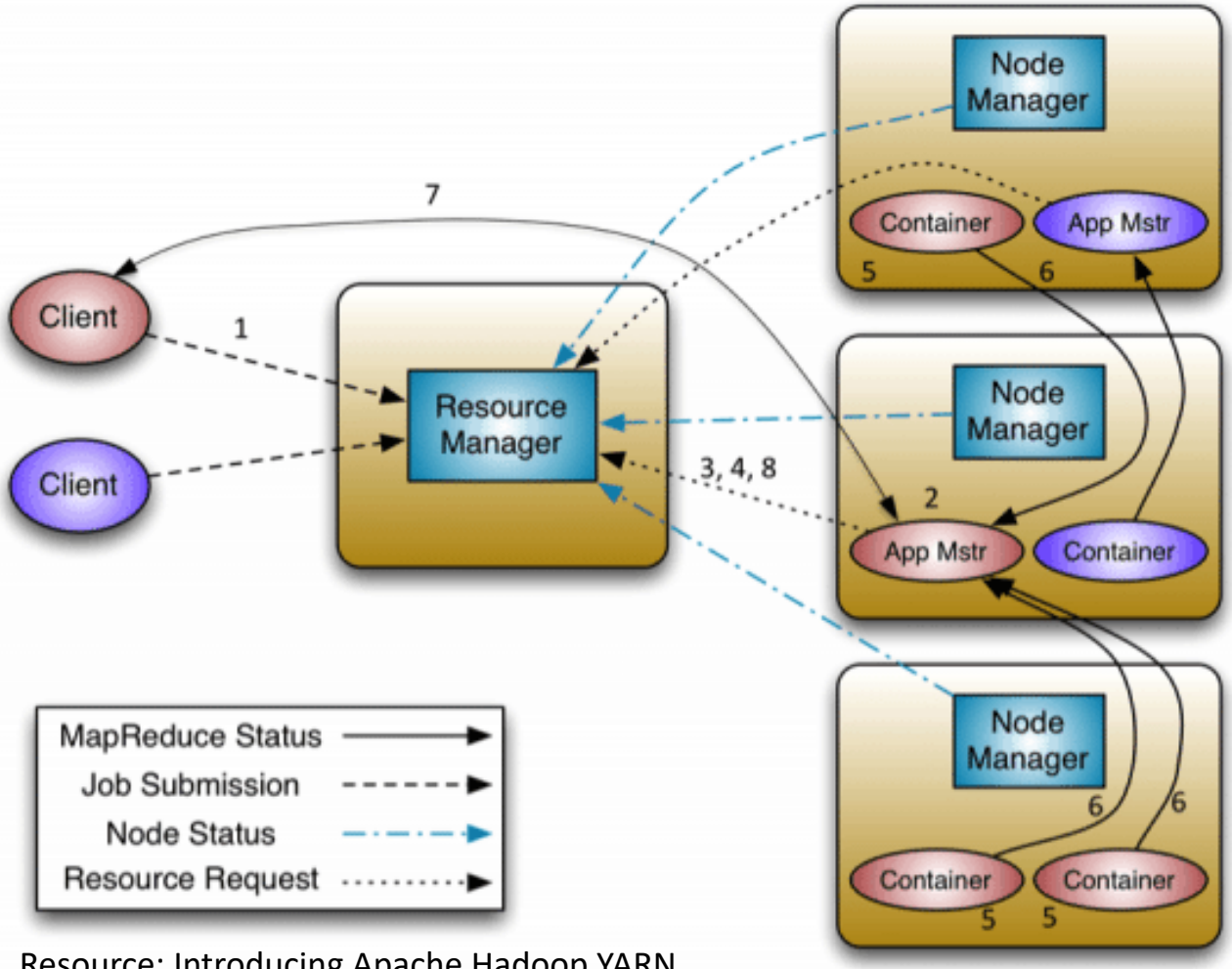


Hadoop YARN Distributed Application Management Framework

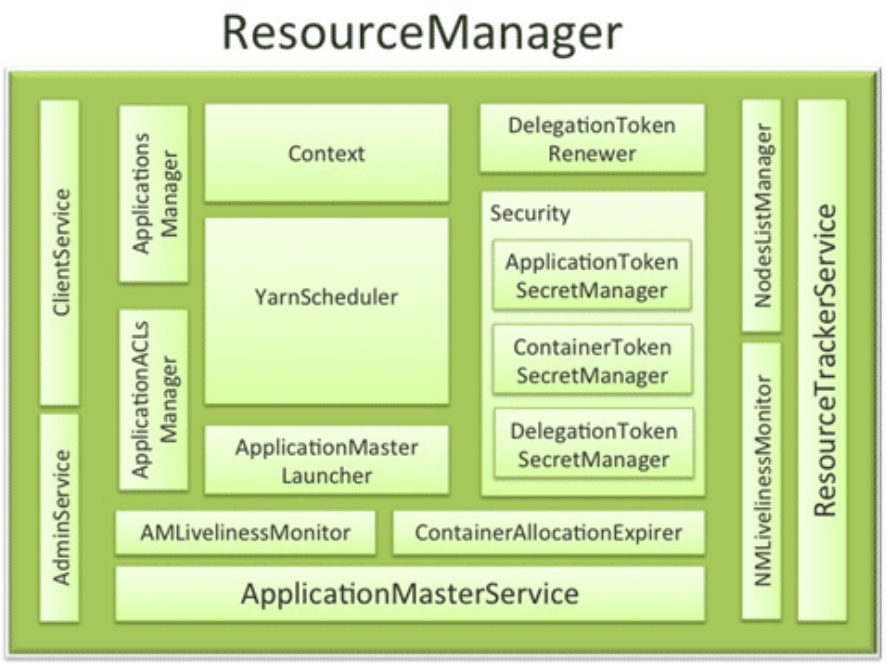
- Conceptual Design/Differences
  - Pure scheduler: limited to arbitrating available resources in the system
  - Pluggable scheduler: multiple scheduling algorithms
  - Job management is handle by ApplicationMaster
  - Resource Model:
    - Resource-name (hostname, rackname)
    - Memory (MB)
    - CPU (cores)
  - ResourceRequest:
    - resource-name, priority, resource-requirement, number-of-containers
    - Container: the resource allocation

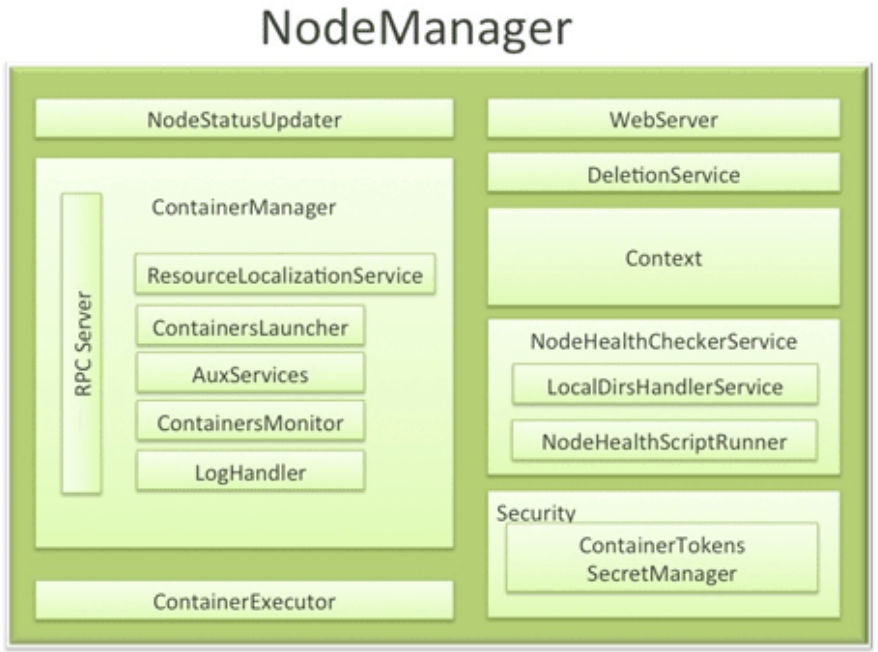


How Does Hadoop YARN Work?



Resource: Introducing Apache Hadoop YARN
(https://hortonworks.com/blog/introducing-apache-hadoop-yarn/)





Hadoop YARN Working Steps

- Step 1: A client program submits the application
- Step 2: Resource Manager negotiates a container to start the Application Master and then launches the Application Master
- Step 3: The Application Master, on boot-up, registers with the Resource Manager. This allows the client to query the Resource Manager for details to directly interact with its Application Master
- Step 4: Application Master negotiates resource containers via the resource-request protocol
- Step 5: After successful allocations, the Application Master launches the container by providing the container launch specification to the Node Manager. This includes command line to launch, environment variables, local resources (jars, shared-objects, ...), and security-related tokens.
- Step 6: The application code executing within the container then provides logging info back to its ApplicationMaster via an application-specific protocol.
- Step 7: During the application execution, the client that submitted the program communicates directly with the Application Master to get status, progress, updates via an application-specific protocol.
- Step 8: Upon completion, the Application Master deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.