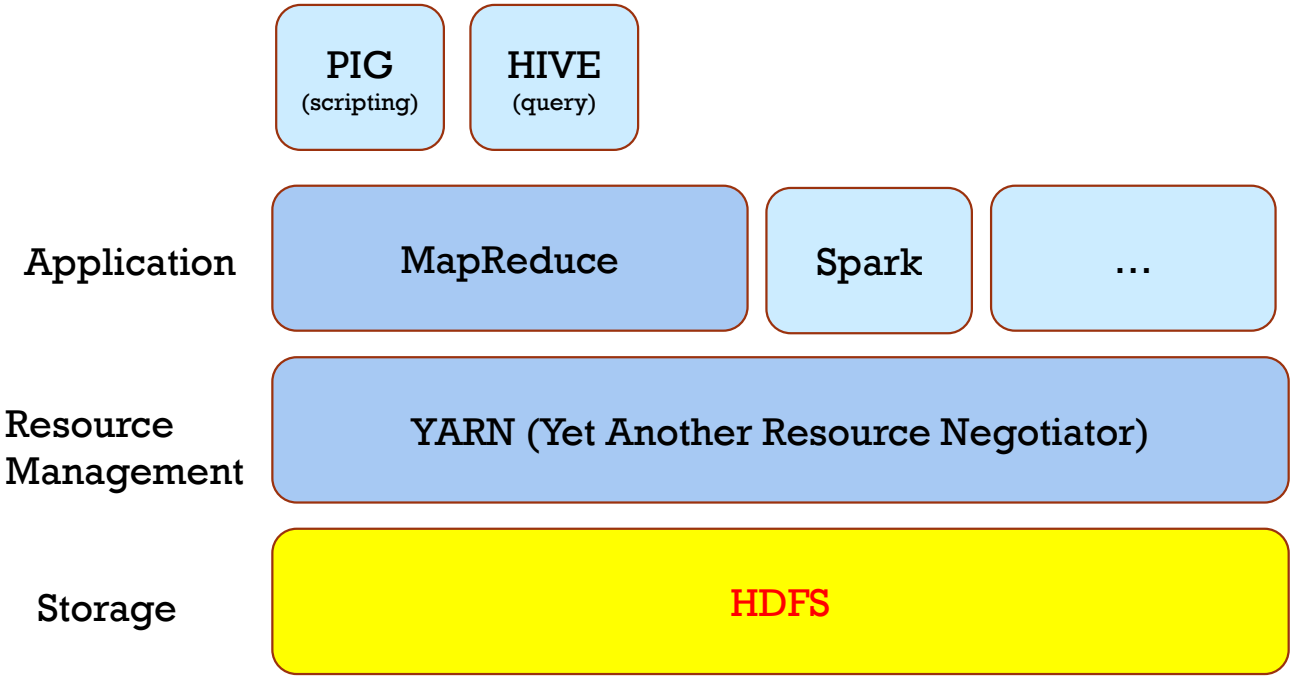
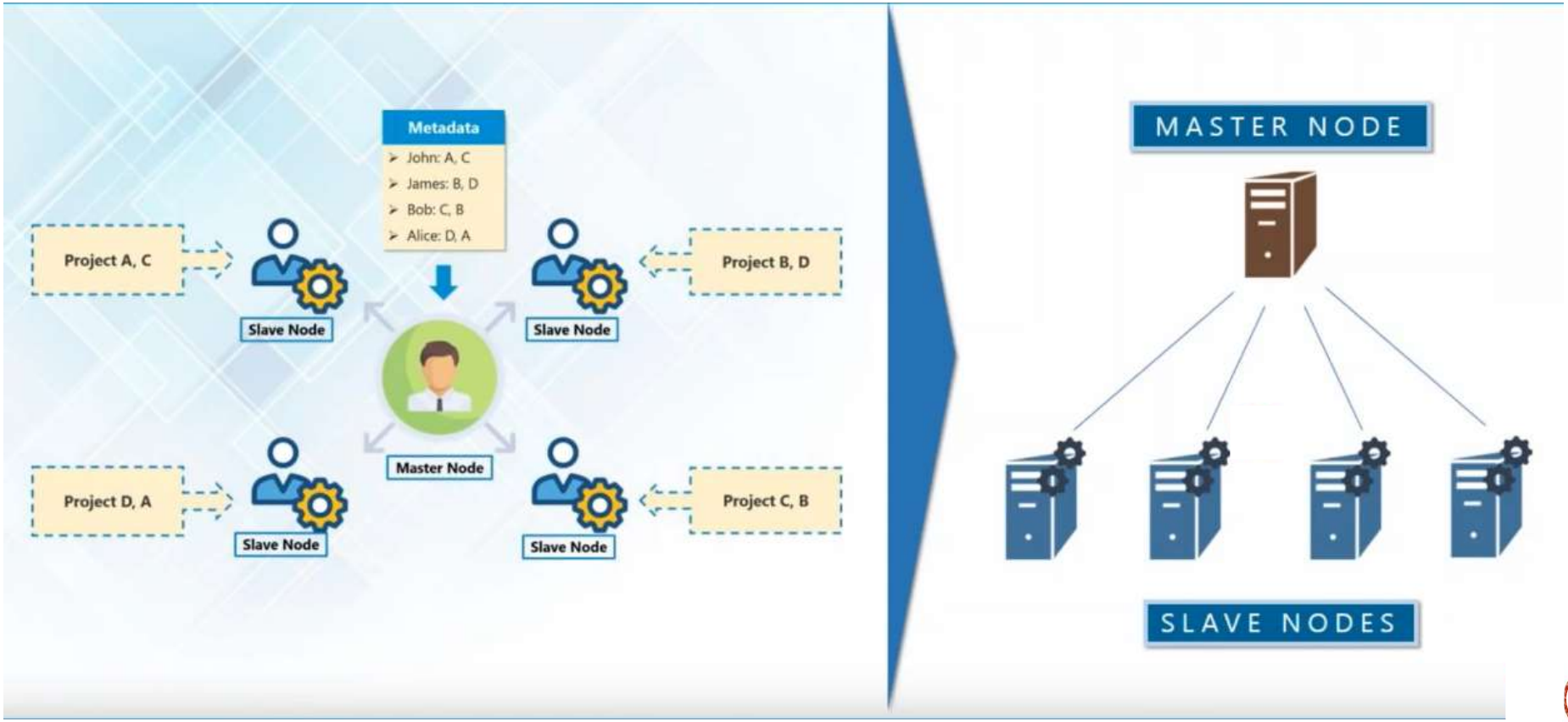
HDFS

HADOOP SOFTWARE ARCHITECTURE



CLIENT/SERVER ARCHITECTURE (MASTER/SLAVE ARCHITECTURE)



DISTRIBUTED FILE SYSTEM

- A client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer

- More complex than regular disk file systems
  - Network-based
  - High-level of fault-tolerance

- BIG DATA -- dataset outgrows the storage capacity of a single physical machine,
  - Data set is partitioned across a number of separate machines.

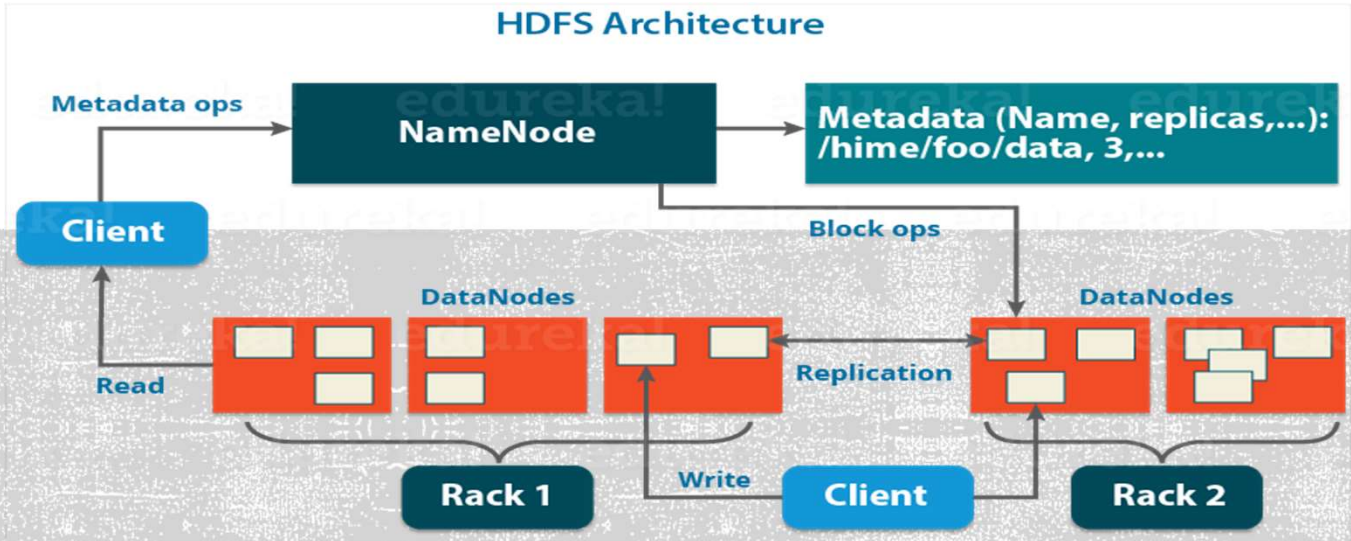
HDFS

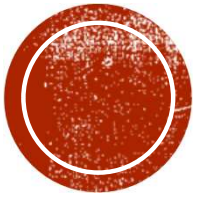
HDFS is a file system designed for storing **very large** files with **streaming data access** patterns, running on clusters on **commodity** hardware.

- **Very large files** - files are normally hundreds of megabytes, gigabytes, or terabytes in size.
- **Streaming data access** – a write-once, read many-times pattern
- **Commodity hardware** – node failure % is high. It has to tolerate to failures without disruption or loss of data.

HDFS DESIGN

- Data Blocks
- Namenode and Datanode
- Data flow
  - Read Operation
  - Write Operation
- Hadoop Replication Strategy
- Hadoop Basic File Operations
- Hadoop Supported File Systems





HDFS BLOCKS

HDFS BLOCKS

HDFS supports the concept of a block, but it is a much larger unit — 128 MB by default.

- Files in HDFS are broken into block-sized chunks, which are stored as *independent* units
- If the size of the file is less than the HDFS block size, the file does not occupy the complete block storage

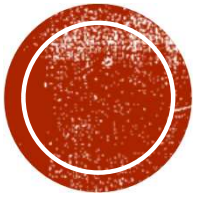
WHY HDFS BLOCK SIZE IS LARGE?

- To minimize the cost of seek time

- Target is to make the seek time 1% of the disk transfer time
  - Ie. 10ms average seek time and 100 MBps transfer rate → ? MB block size

BLOCK ABSTRACTION

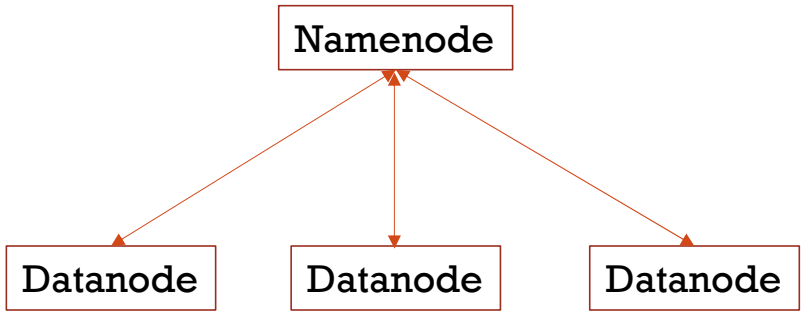
- Flexible
  - Blocks from a file can be stored on any available disks in the cluster
- Scalable
  - Can handle very large data set in a distributed environment
- Simple storage subsystem and management
  - Easy to determine the number of blocks that can be stored in a disk
  - Easy to deal with various failure modes
- Each block is replicated to achieve the desired level of fault tolerance and availability
  - The default replication factor is **three** machines, and can be changed to fit the needs



HOW DOES HDFS WORK?

HDFS NAMENODE AND DATANODES

- A HDFS cluster has two types of node – a **namenode** and a number of **datanodes**,
  - Operate in a **master-worker** pattern



HDFS NAMENODE AND DATANODES

- **Namenode**
  - Maintains the file system tree and the metadata for all the files and directories in the tree
    - Store persistently on the local disk in the form of two files: the *namespace image* and the *edit log*
  - Services block location requests
    - The entire (block, datanode) mapping table is stored in memory, for efficient access

- **Datanode**
  - Stores data
  - Maintains the block location information

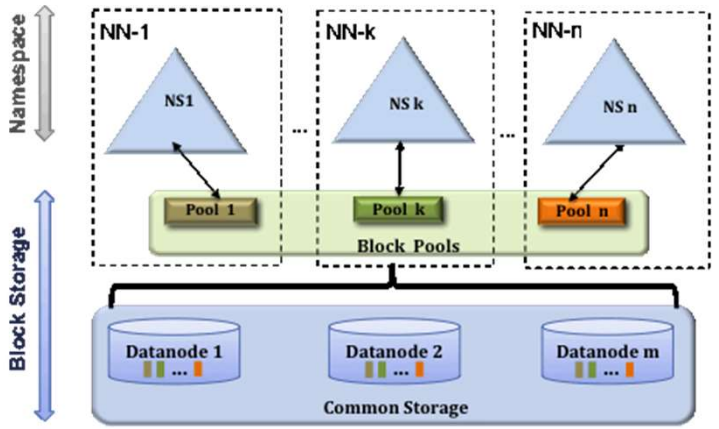
NAMENODE DESIGN ISSUES AND CHALLENGES

- Namenode is the main component to make filesystem operational
  - If it fails, the whole cluster will not function
- It maintains a single namespace and metadata of all the blocks for the entire of cluster
  - Not scalable
    - the entire of namespace and block metadata in memory
  - Poor isolation
    - no way to separate a group of works from one another

NAMENODE FEDERATION

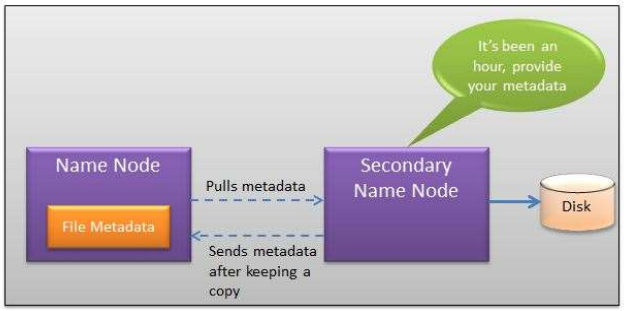
SCALABILITY

- HDFS Federation is introduced in the 2.x release to address scalability issue by adding more namenode(s)
- Each manages a portion of filesystem namespace which independent from one another
- Each data node will register with each namenode to store blocks for different namespace



NAMENODE FAULT TOLERANCE

Simple solution - **secondary** namenode



- Periodically (hourly) pulls a copy of the file metadata from the active namenode to save to its local disk

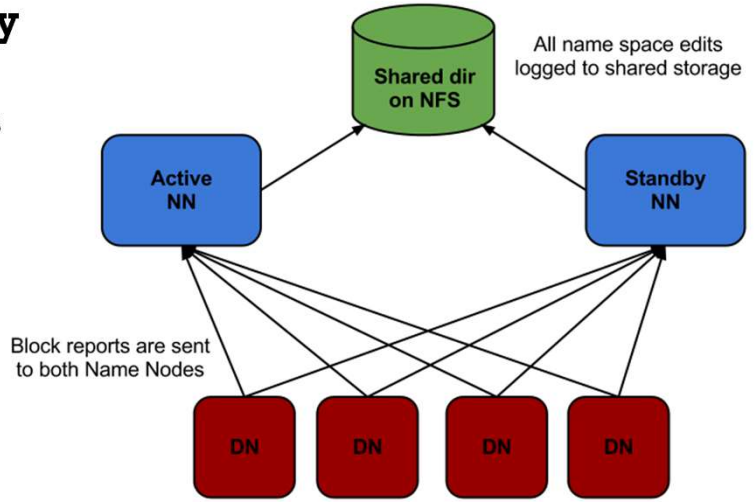
x Secondary namenode might take too long to come up online (30 minutes+)

1. Load the namespace image into the memory
2. Replay its edit log
3. Receive enough block reports from the datanodes to leave safemode

NAMENODE FAULT TOLERANCE

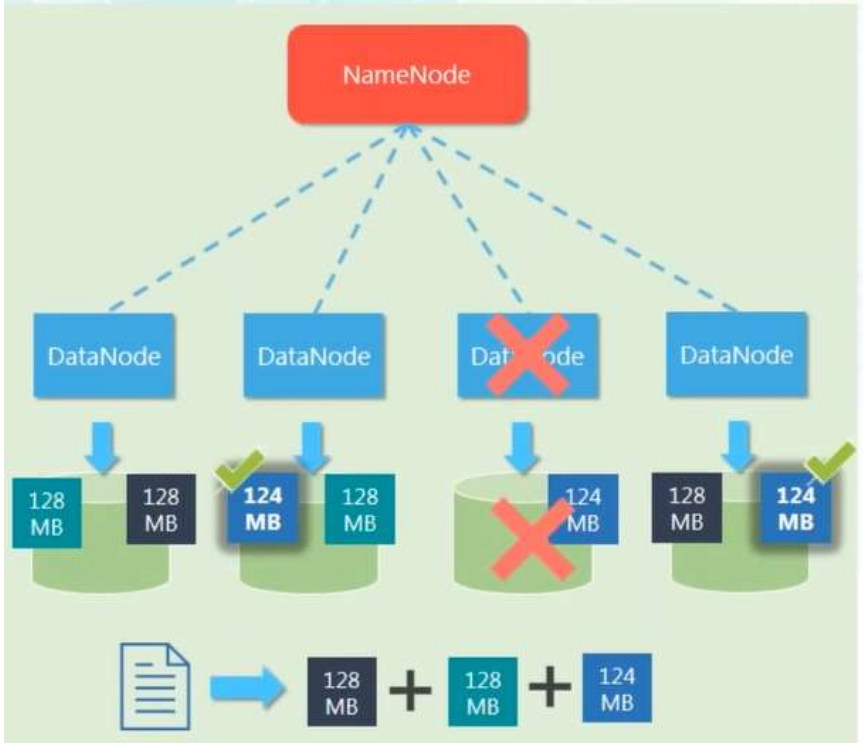
- HIGH AVAILABILITY (HA) CONFIGURATION -

- A pair of namenodes are configured as **active-standby**
- The standby takes over whenever the active one fails
- The namenodes use highly available shared storage to share the *edit log*
  - The active writes, and the standby reads to keep it in sync
- Datanodes must send block reports to *both* namenodes
- Client must be configured to handle namenode failover



WHAT HAPPENS IF DATANODE FAILS?

FAULT TOLERANCE – REPLICATION FACTOR



Solution:

Each data blocks are replicated (thrice by default) and are distributed across different DataNodes

APPLICATIONS NOT SUITABLE FOR HDFS

- Applications that require low-latency access, as opposed to high throughput of data

- Applications with a large number of small files
  - Costly metadata
  - In average, each file, directory, and block takes about 150 bytes. For a HDFS that maintains 1 million files with one block each will need memory – 1,000,000*150*2 = 300 MB

- Applications with multiple writers, or modifications arbitrary offsets in the file
  - Files in HDFS written by a single writer, with writes always made at the end of the file