Hadoop Distributed File System

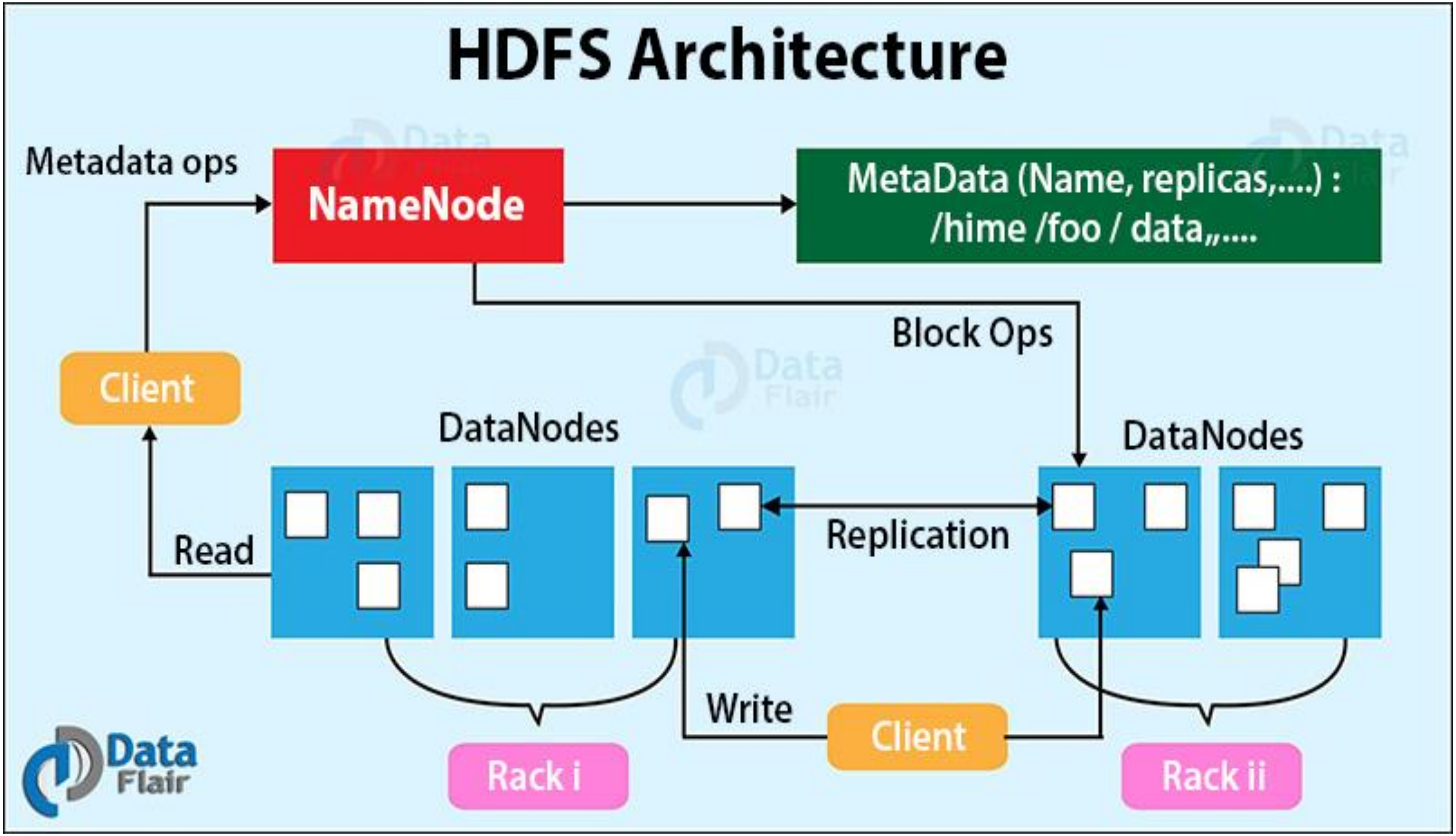
Dr. Amit Praseed

Assumptions behind HDFS

- Runs on commodity hardware – failure is common
- Works well with a number of large files
- Optimized for "Write once, read many times"
- Optimized for large streaming reads
- High throughput is more important than low latency

Notice the similarity with GFS!
This is because HDFS was designed and built based on GFS specifications

HDFS Architecture



HDFS Architecture

- Operates on top of an existing file system
- Files are stored as blocks
  - Default size is 64 MB
- Reliability through replication
- NameNode stores metadata and manages access
- No caching due to large file sizes

HDFS File Storage

- NameNode
  - Stores metadata – filename, location of blocks etc
  - Maintains metadata in memory
- DataNode
  - Stores file contents as blocks
  - Different blocks of same file are on different data nodes
  - Same block is replicated across data nodes

Failure and Recovery

- NameNodes keep track of DataNodes through periodic HeartBeat messages
- If no heartbeat is received for a certain duration, DataNode is assumed to be lost
  - NameNode determines which blocks were lost
  - Replicates the same on other DataNodes
- NameNode failure = File system failure
- Two options
  - Persistent backup and checkpointing
  - Secondary/backup NameNode

Balancing Hadoop Clusters

- Hadoop works best when data is evenly spread out
- Goal is to have all DataNodes filled up to a similar level
- Hadoop runs a balancer daemon
  - Redistributes blocks from over utilized DataNodes to underutilized ones
  - Runs in the background and can be throttled as necessary