Hadoop Overview

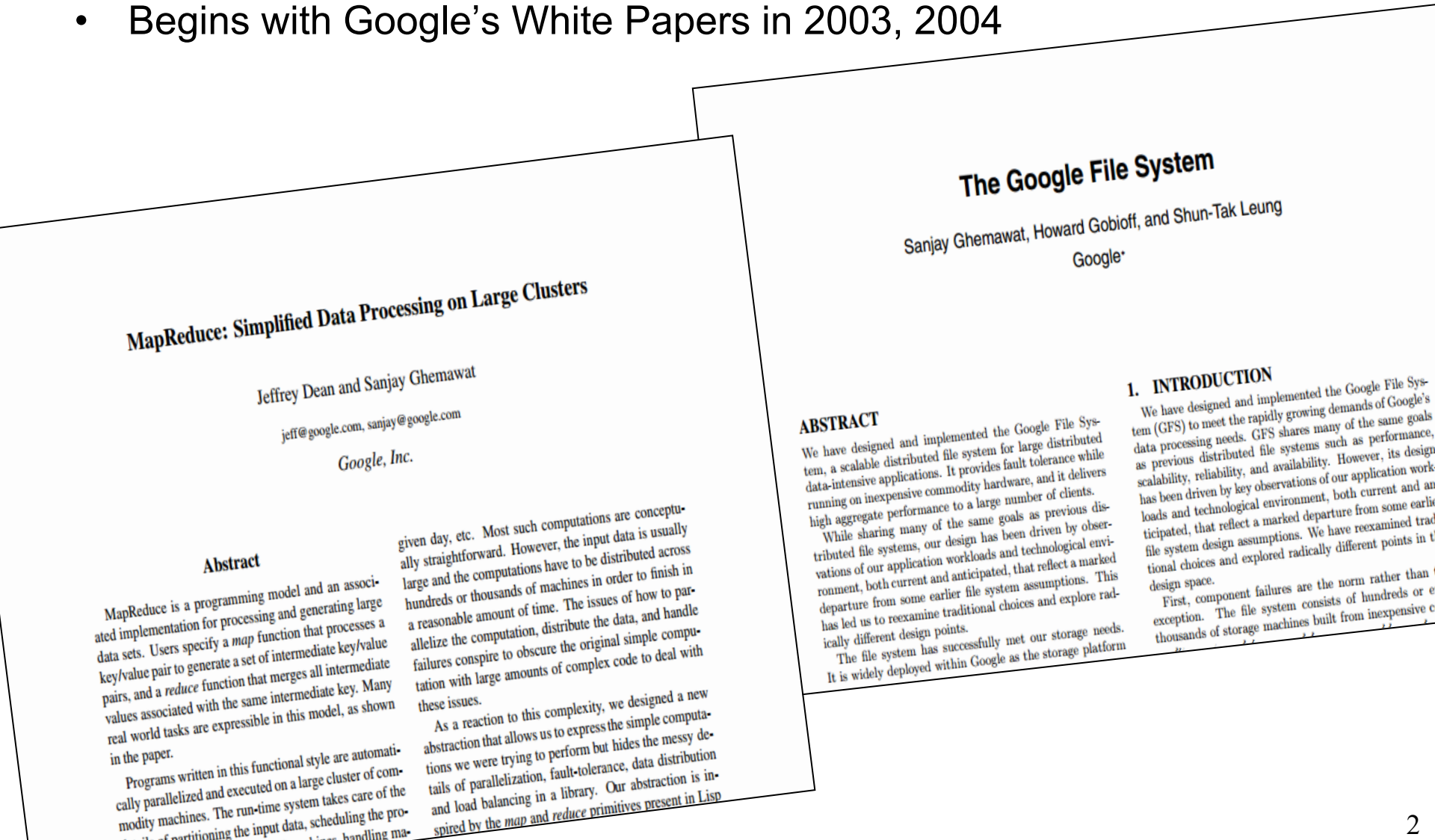
Why are talking about Hadoop?

- Contribution to the NoSQL movement
- Contribution to the Open Source movement
- Pathfinder to prove key NoSQL fundamentals
- Illustrates TRENDS in software
  - Certain Technologies come on the scene to address a particular problem
  - System Engineers want to learn the latest, best thing
  - Organizations want to adopt the latest, best thing
  - The demands of the marketplace create inflationary job movement
  - Then ....
    - New technologies come along and push out the old
    - The cycle repeats

Hadoop Overview

Hadoop -- History and evolution

- Begins with Google's White Papers in 2003, 2004



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform

1. INTRODUCTION

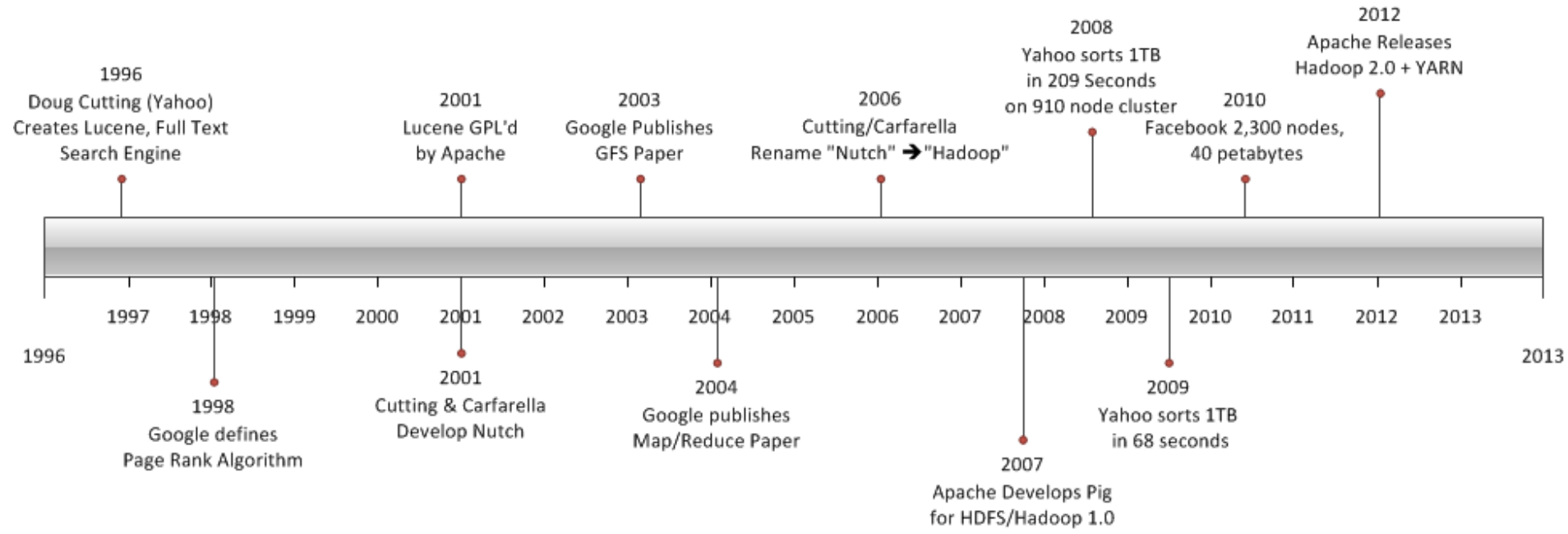
We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive

Hadoop's History

- Built upon Google's concepts
- Core = the Google File System
- Google: Set out to create the world's finest web search engine
- Quickly overwhelmed existing relational DBMS platforms
- Created their own solutions
- Built into Hadoop
- Very "hot" technology for the past 6 years
- Waning in popularity due to cloud-based possibilities

Hadoop – Timeline



Hadoop – Principles

- Parallelization (for throughput)
- Distribution of Data and Compute Tasks
- On cheap commodity hardware (physical, not virtual)
- Local dedicated disk
- Fault Tolerance
- Redundancy Factor
- Relaxed ACID compliance
- Chunks = Large Blocks
- Linear Scalability
- Write Once Read Many
- Open Source

stopped here on friday 04/17

Hadoop Provides

- A distributed filesystem (HDFS) that can store data across thousands of servers
- A means of running work (MapReduce & YARN) across those machines, running the work near the data

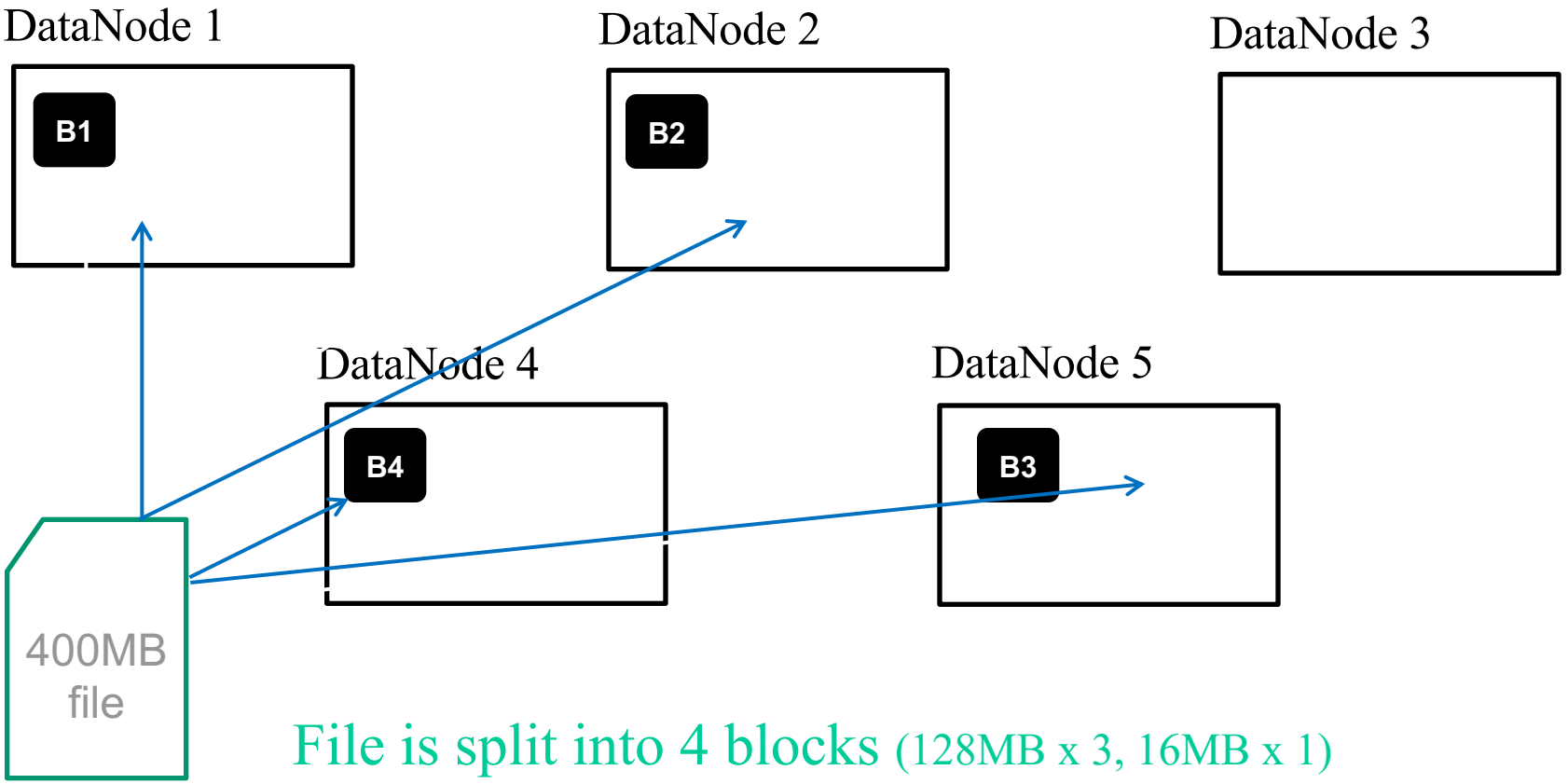
http://wiki.apache.org/hadoop/ProjectDescription

Key Features

Write Once Read Many

- Logical filesystem – sits on top of native OS
- Files split into Blocks
- Blocks spread across the cluster

400 MB file to write to the cluster

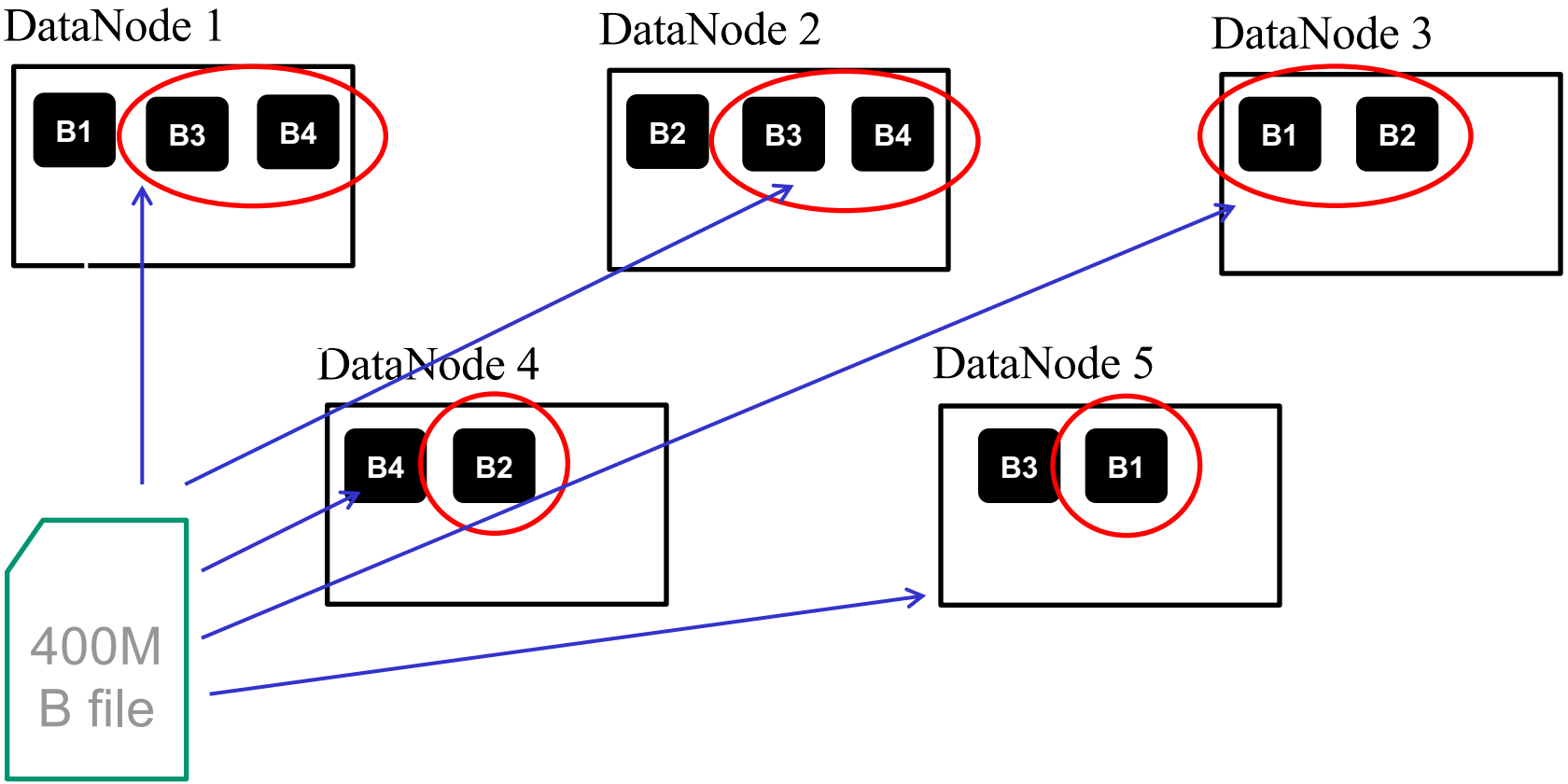


File is split into 4 blocks (128MB x 3, 16MB x 1)

Data Blocks copied to 3 different nodes

- **Fault-Tolerance**
- **Multiple copies provide performance boost**
- **Replication Level is configurable**
- **Full checksums**
- **Rack awareness**

Data Blocks are copied to three nodes



Looks just like a conventional filesystem…

bpreachuk — root@sandbox:~ — ssh — 112×29

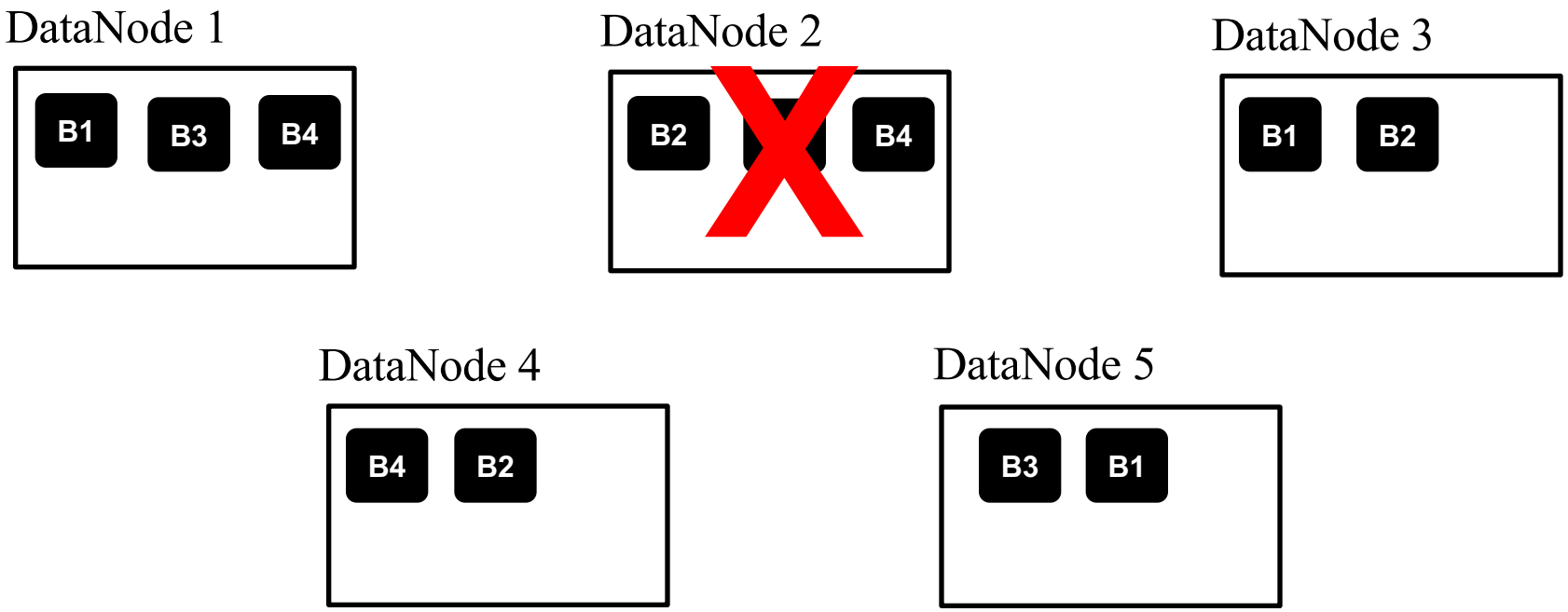
```
[root@sandbox ~]# hadoop fs -ls /apps
Found 5 items
drwxrwxrwx   - falcon hdfs          0 2014-04-21 07:15 /apps/falcon
drwxr-xr-x   - hdfs   hdfs          0 2014-04-21 07:16 /apps/hbase
drwxr-xr-x   - hdfs   hdfs          0 2014-04-21 07:17 /apps/hive
drwxr-xr-x   - tez    hdfs          0 2014-04-21 07:17 /apps/tez
drwxr-xr-x   - hcat   hdfs          0 2014-04-21 07:23 /apps/webhcat
[root@sandbox ~]#
[root@sandbox ~]#
[root@sandbox ~]# hadoop fs -ls -R /apps/hive/warehouse/sample_07
-rw-r--r--   1 hue hdfs      46055 2014-04-22 07:21 /apps/hive/warehouse/sample_07/sample_07.csv
[root@sandbox ~]#
[root@sandbox ~]#
[root@sandbox ~]# hadoop fs -ls  /apps/hive/warehouse
Found 5 items
drwxr-xr-x   - root hdfs          0 2014-08-23 00:18 /apps/hive/warehouse/generic_dw.db
drwxr-xr-x   - hue  hdfs          0 2014-04-22 07:21 /apps/hive/warehouse/sample_07
drwxr-xr-x   - hue  hdfs          0 2014-04-22 07:21 /apps/hive/warehouse/sample_08
drwxr-xr-x   - root hdfs          0 2014-08-22 14:55 /apps/hive/warehouse/tpcds_bin_partitioned_orc_5.db
drwxr-xr-x   - root hdfs          0 2014-08-22 15:26 /apps/hive/warehouse/tpcds_text_5.db
[root@sandbox ~]#
```

HDFS – Fault-Tolerance

Node failure is **Expected**

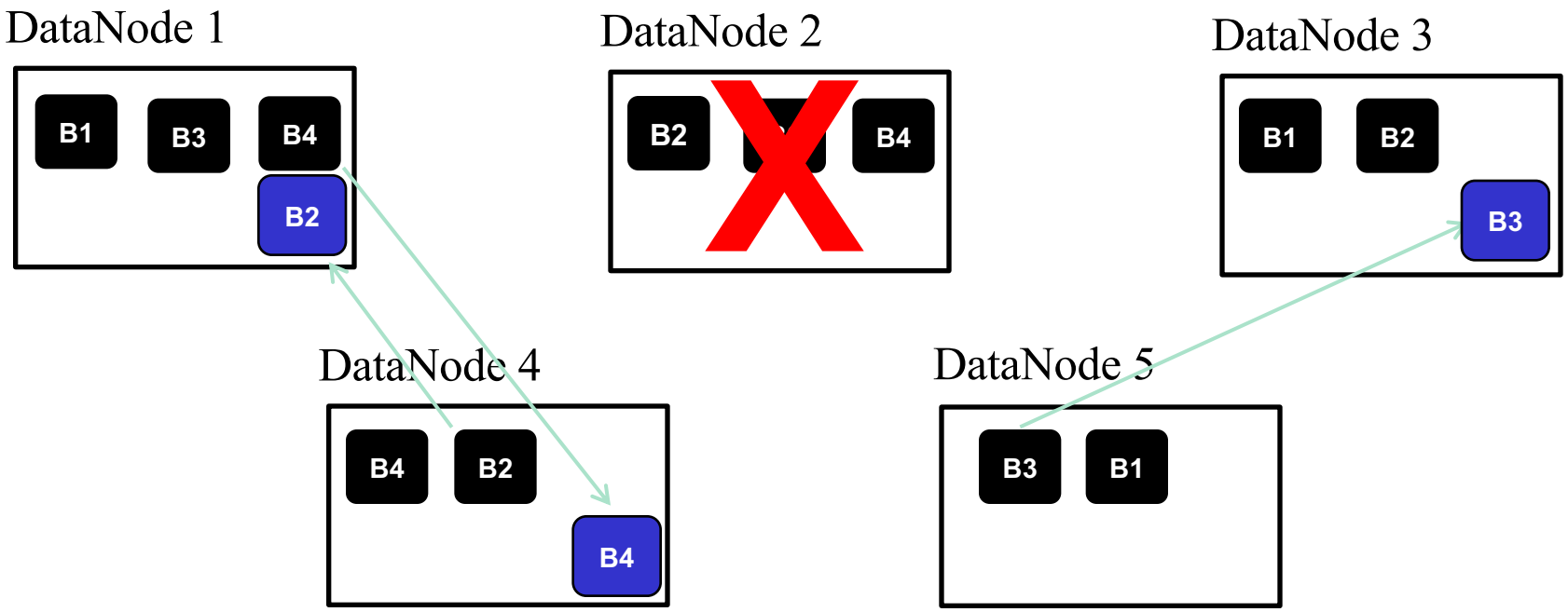
- Cluster detects missing/under-replicated blocks
- Cluster detects failed/unreachable nodes
- HDFS re-replicates data to maintain 3-copy protection level

Node Failure is Expected



When a node fails (or is unreachable)…

HDFS re-replicates data



Maintains 3-copy protection level

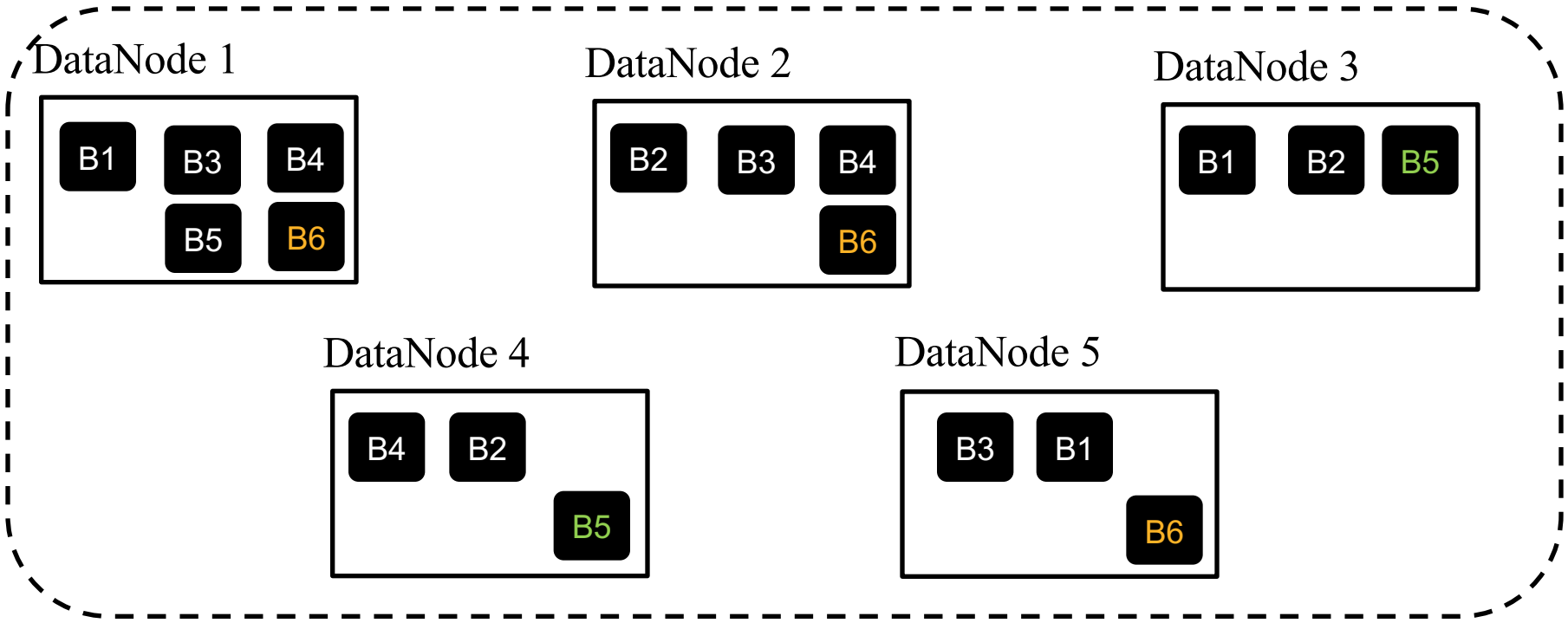
How does the cluster know:

- What to replicate
- Where data is stored?
- If a block has been corrupted?
- If a block is over or under-replicated?
- If a data node has failed or is unreachable?

The NameNode(s)

- Critical High-Performance Node
- Contains all metadata for data blocks
- Keeps metadata in-memory
- Controls re-replication of missing data

HDFS – NameNode



NameNode

StoreSales.txt - blocks B1, B2, B3, B4
Promos.csv - blocks B5, B6

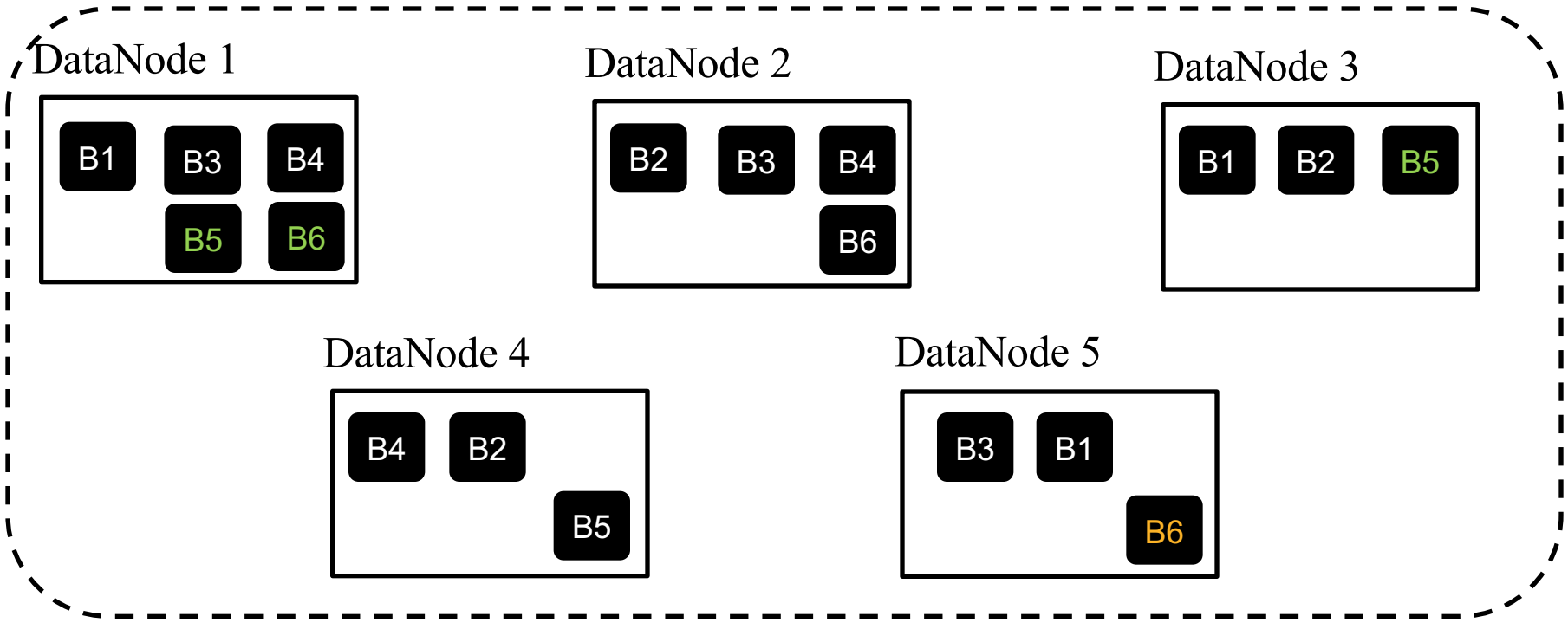
- Critical High-Performance Node
- Contains all metadata for data blocks
- Keeps metadata in-memory
- Controls re-replication of missing data

HDFS - Communications

- **Data Node Heartbeats**
- **Data Node Block Reports**
- **Rack Awareness**

If Name Node is down… Hadoop is down!

HDFS – Standby NameNode



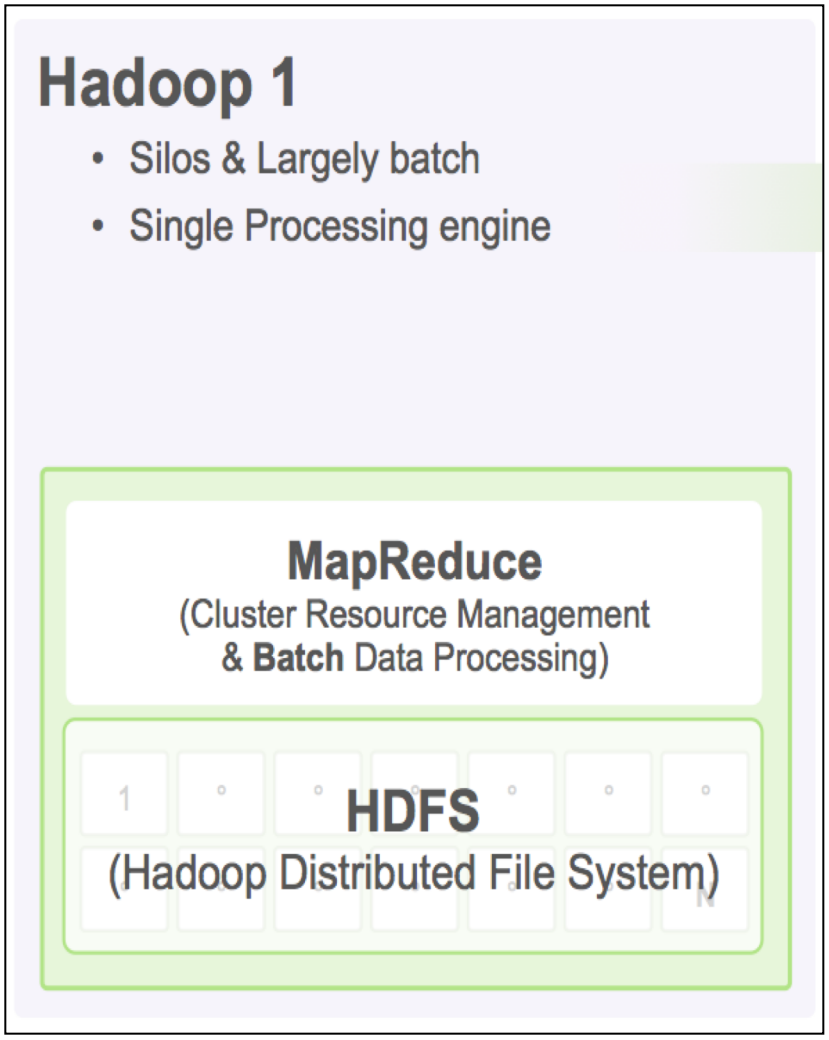
NameNode

StoreSales.txt - blocks B1, B2, B3, B4
Promos.csv - blocks B5, B6

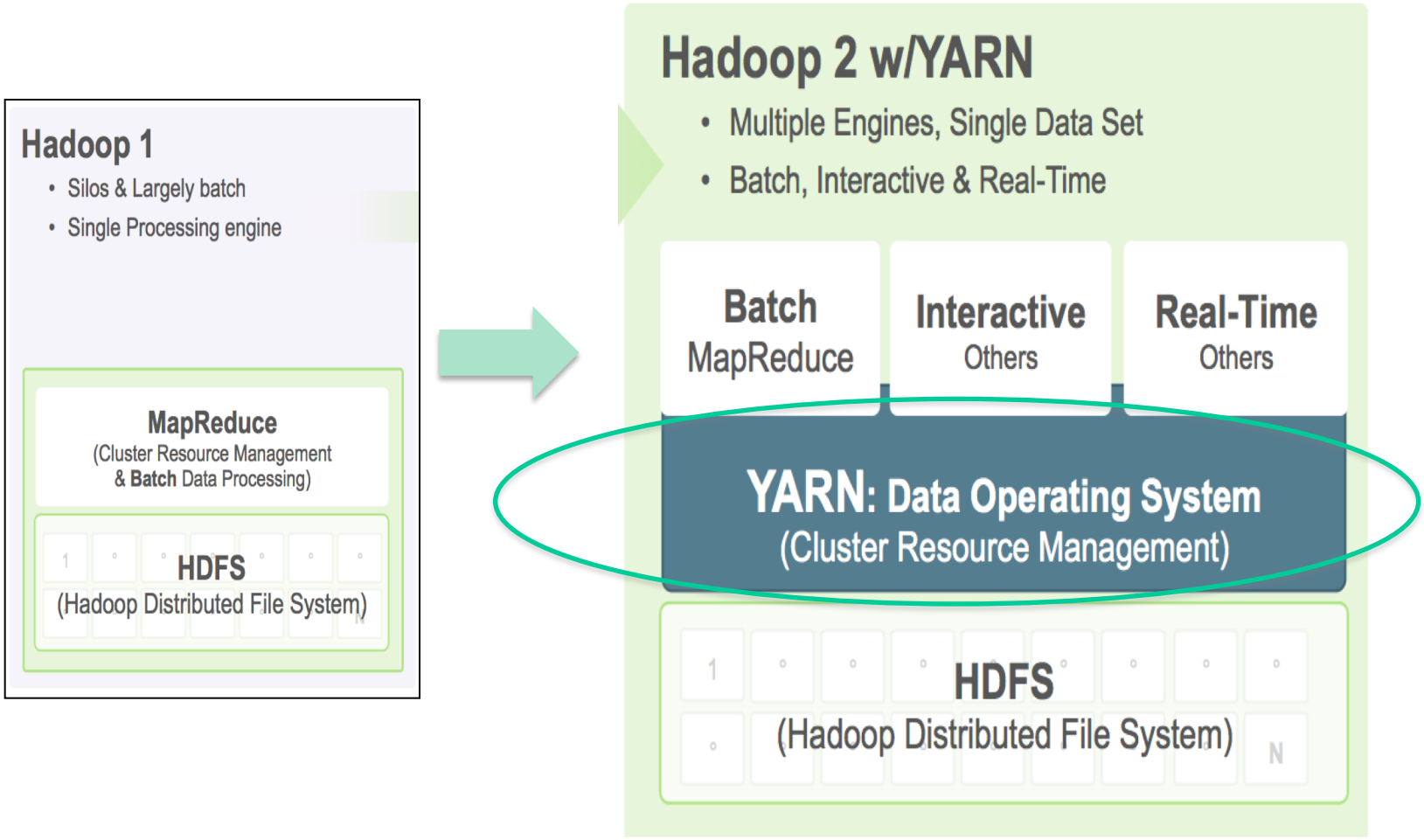
Standby NameNode

Redundant Name Space
Metadata Checkpoints to disk

MapReduce – Hadoop 1.0



YARN/Tez – Hadoop 2.0



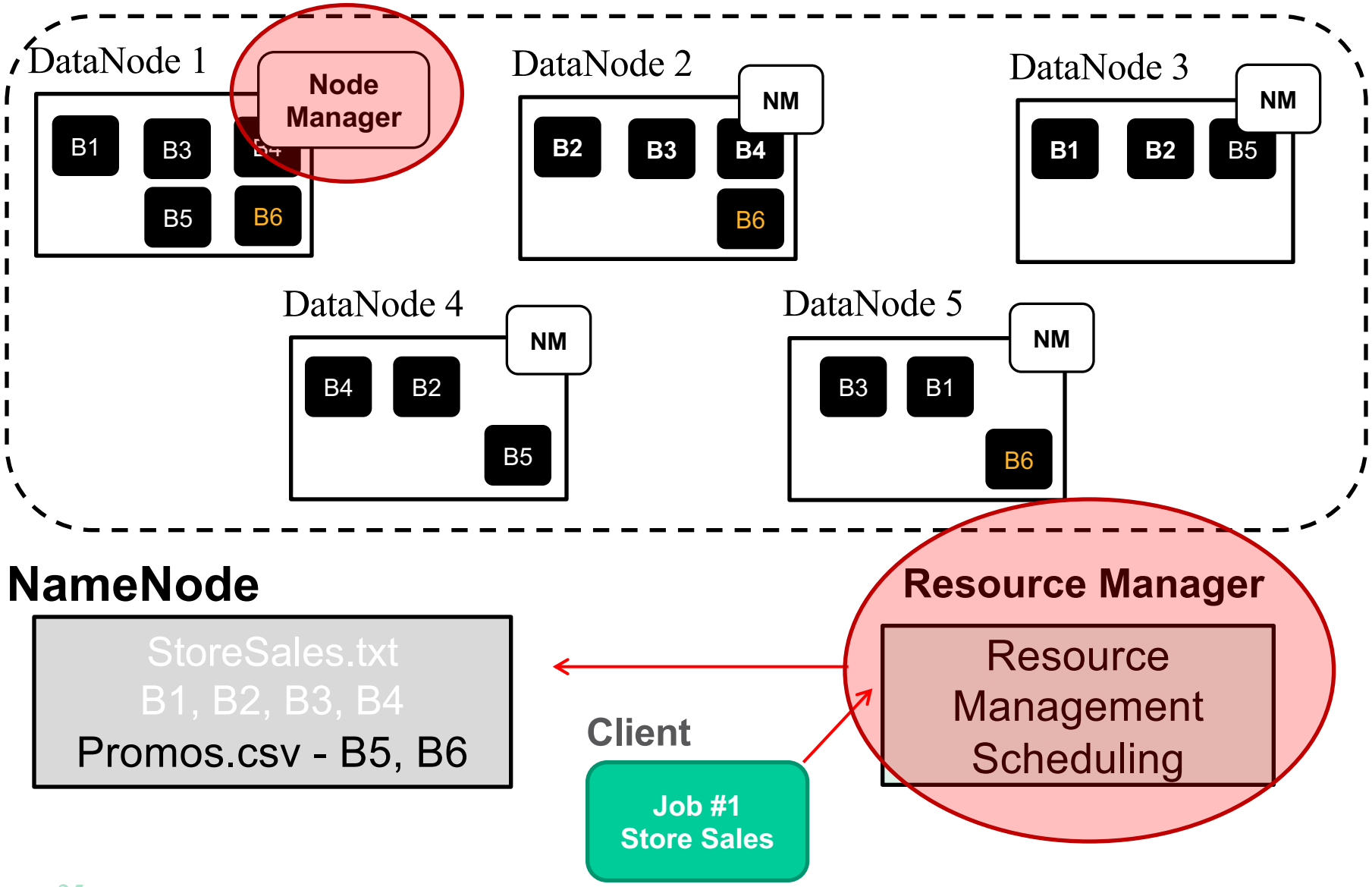
The Data Operating System

- More ways to process data than just MapReduce - Tez
- Improved Scalability & cluster Resource Utilization
- Mixed Workloads – Batch, Streaming, Online

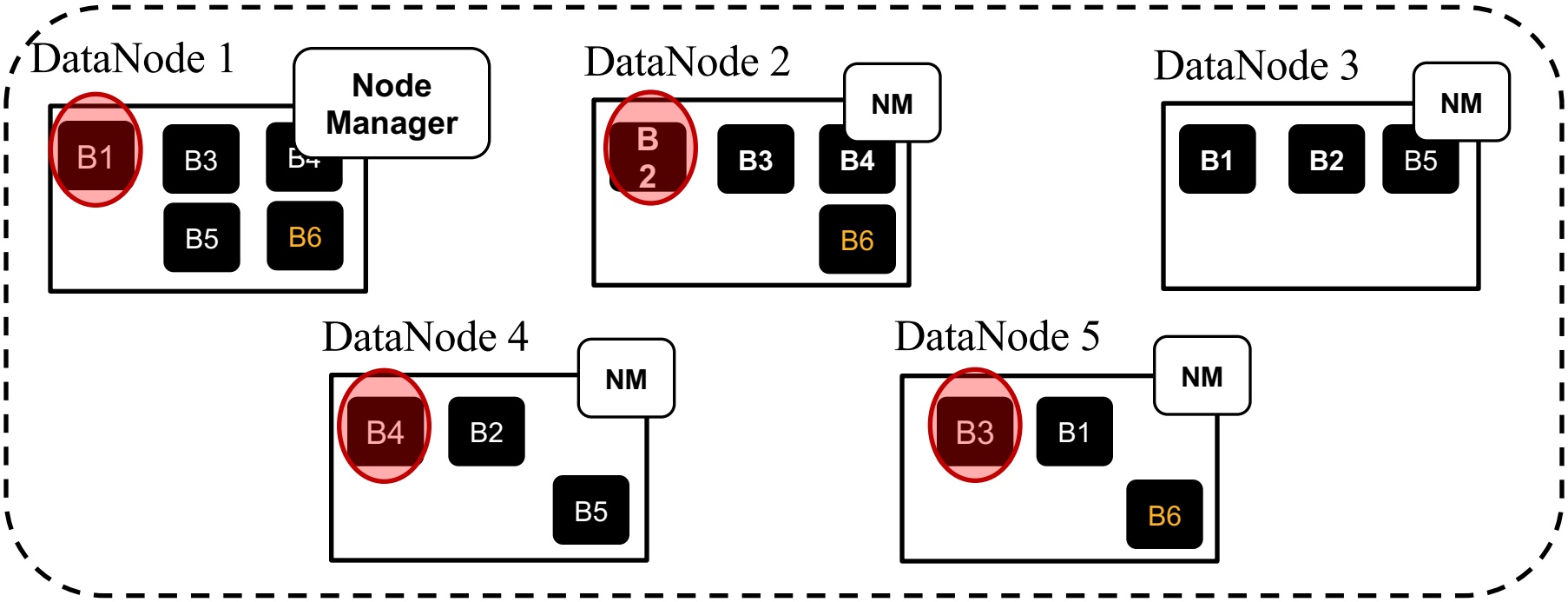
So how do we run 'work' against a Hadoop cluster?

- Jobs are split into Tasks
- Tasks are sent to the Data ***
- Tasks are performed in parallel

Hadoop Job Execution



Hadoop Job Execution



NameNode

StoreSales.txt
B1, B2, B3, B4
Promos.csv - B5, B6

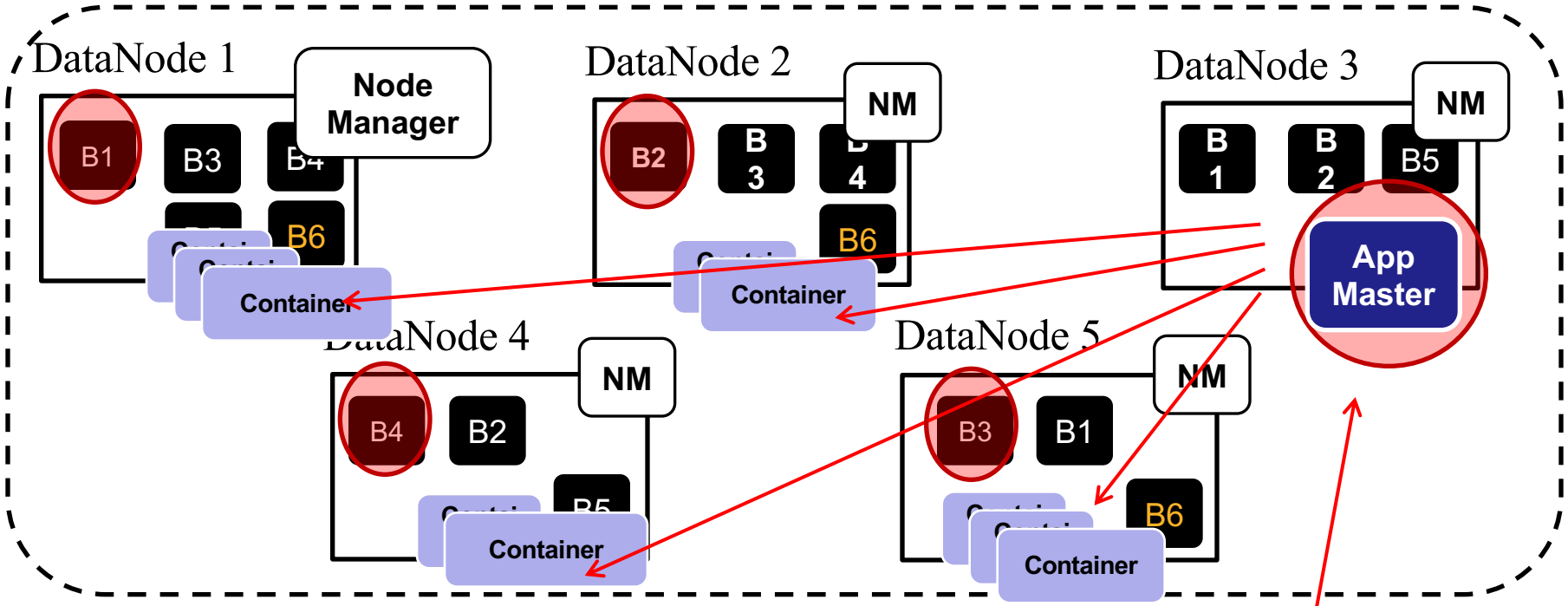
Resource Manager

Resource Management Scheduling

Client

Job #1 Store Sales

Hadoop Job Execution



NameNode

StoreSales.txt - B1, B2, B3, B4
Promos.csv - B5, B6

Client

Job #1
Store Sales

Resource Manager

Resource Management Scheduling

Job Execution – High Points

Resource Manager

- **Application Management and Scheduling (at least one, but most users set up redundancy for HA)**

Node Managers

- **Worker Nodes (1 per DataNode)**

Application Masters

- **Controller for each YARN application (1 per Job)**

Containers

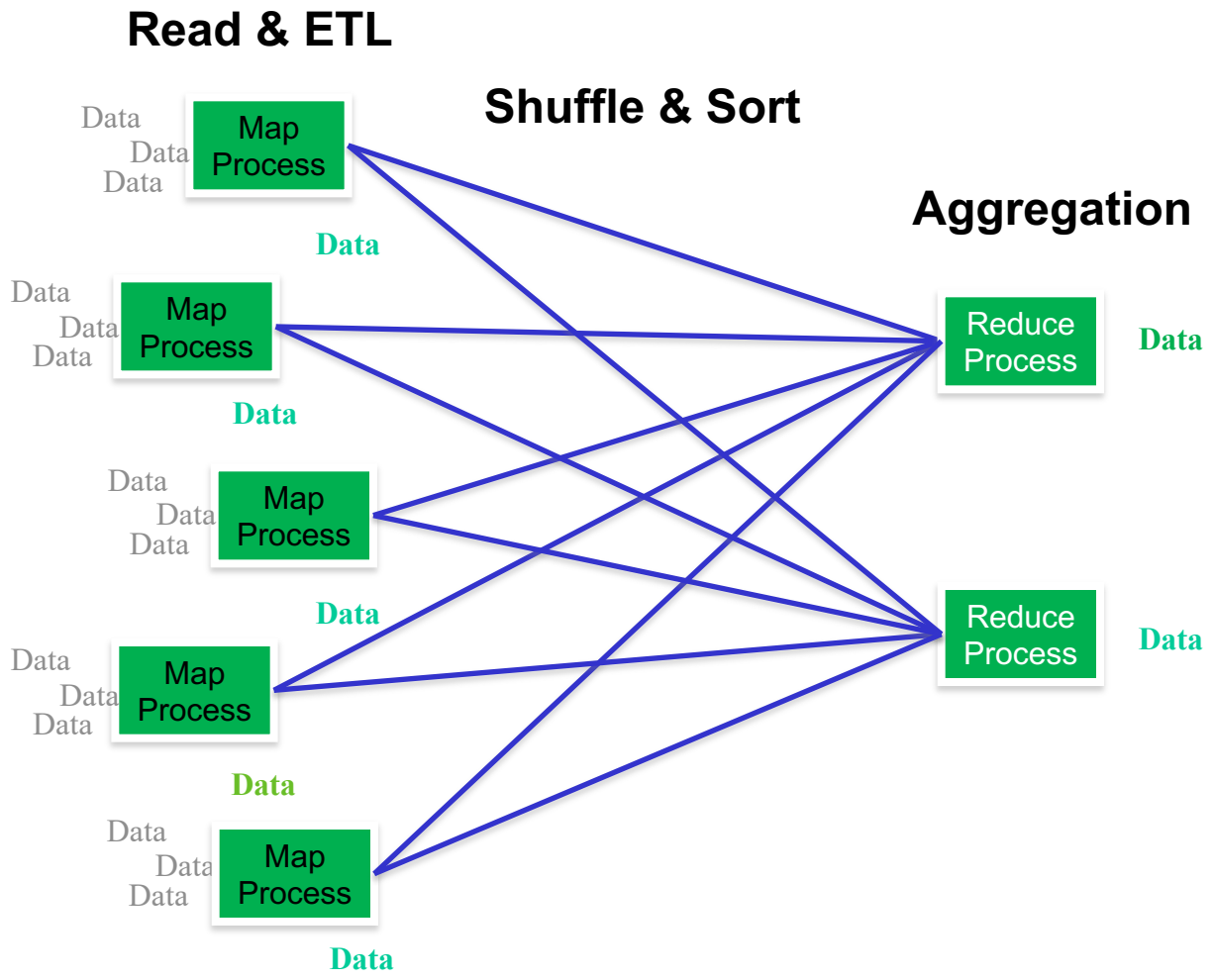
- **Generic Resource Unit (1 per Task)**

Tolerant of Task Failures

In fact, Task failures are expected

- Speculative Execution – run a second task
- Task Resubmission

MapReduce breaks a large problem into sub-solutions



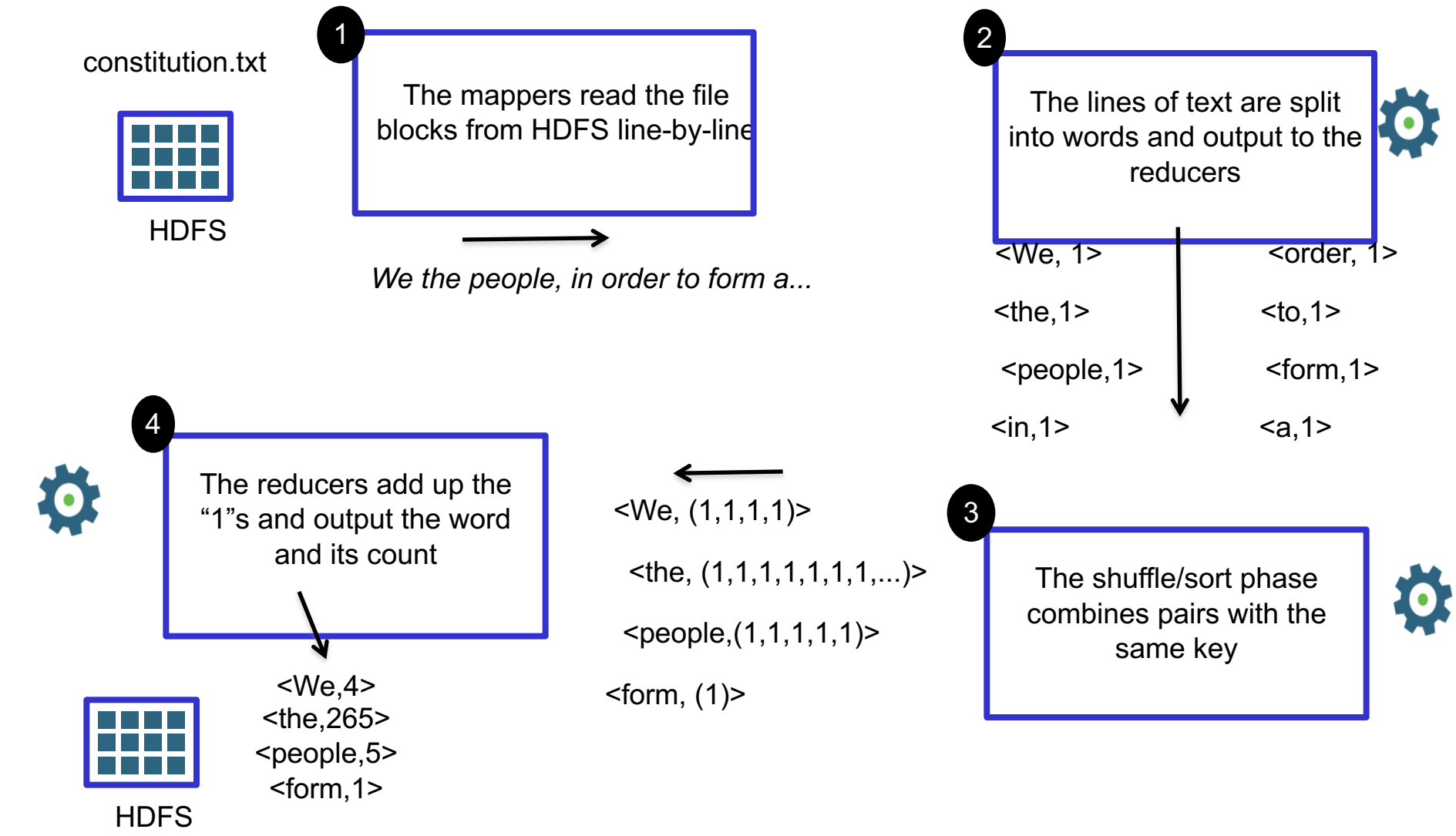
Map (SQL Analogy: Select – From - Where)

- Read data one Key-Value Pair at a time
- Apply computation
- Emit <key, value> pairs

Reduce (SQL Analogy: Group By - Having - Aggregation - Sort)

- Read Map Task results
- Apply computation
- Emit <key, value> pairs – usually 1 row/key

WordCount in MapReduce

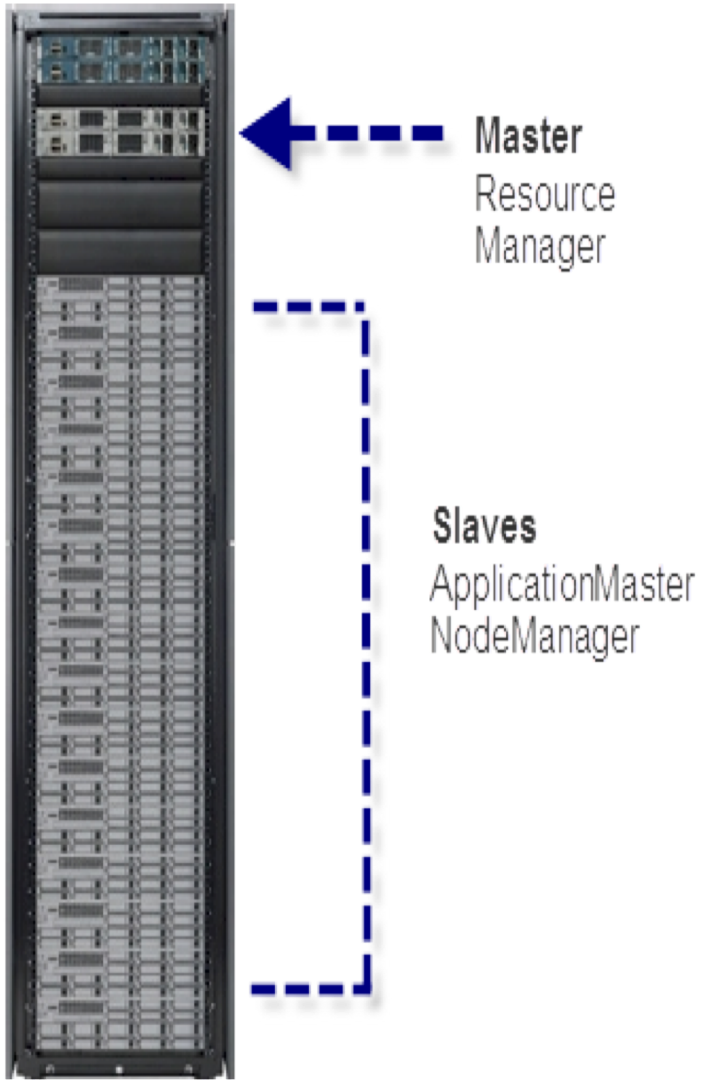


YARN

ResourceManager allocates cluster resources, starts ApplicationMaster

ApplicationMaster divides job into tasks, assigns task to NodeManagers

NodeManager runs on all slave nodes, starts and monitors task



stopped here mon apr 20

1. Coming Up Lectures
2. Exam
    - Review on Monday
    - Exam on Wednesday
    - Accommodations

BIG Improvements over MapReduce

Mixed Workloads

- **Batch, Interactive, and Real-Time processing**

Greater Task Parallelism

- **No longer successive Maps & Reduce loops**

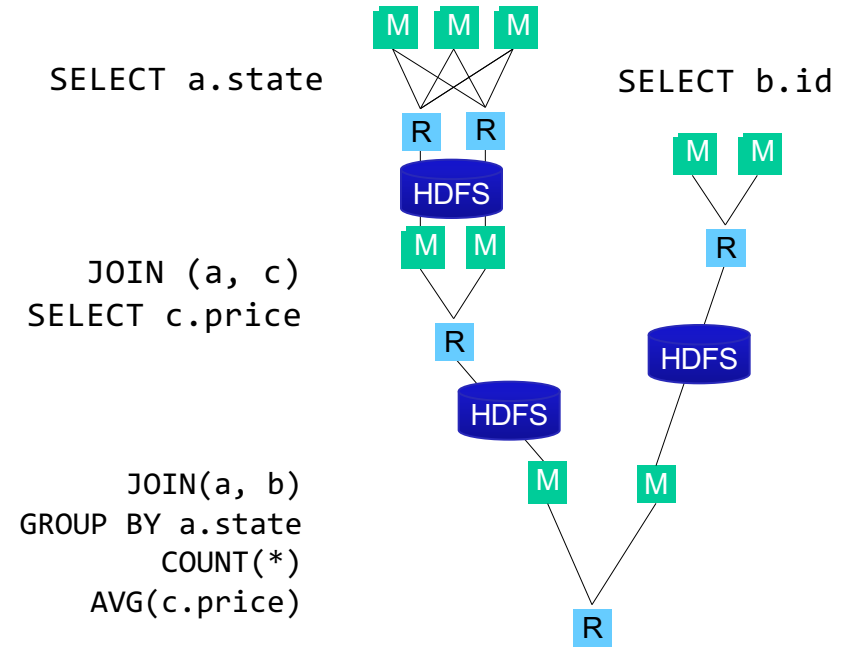
In-Memory & Caching Optimizations

- **No writes or spill to disk unless necessary**
  - Reduced process overhead (startup costs)
  - Data remains in-memory

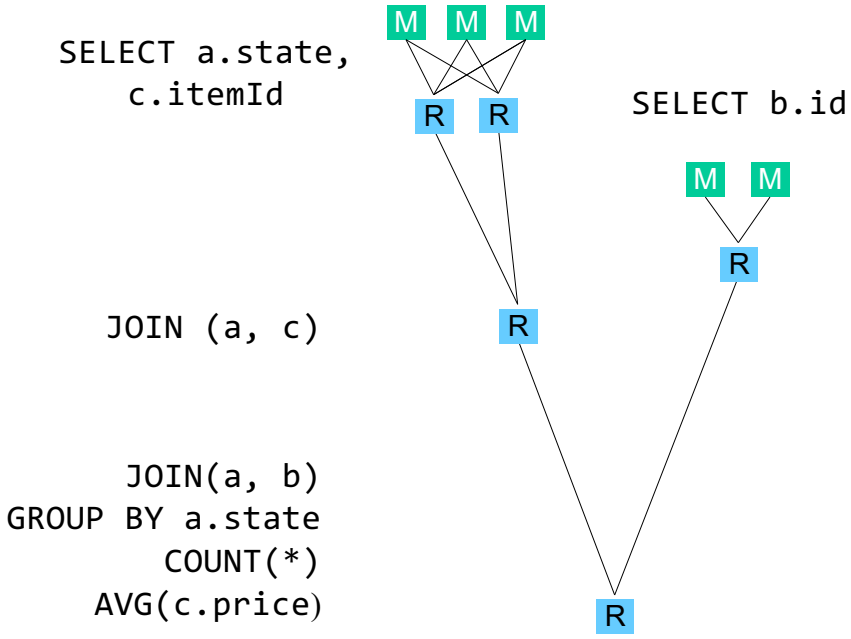
```
SELECT a.state, COUNT(*), AVG(c.price)
FROM a
   INNER JOIN b ON (a.id = b.id)
   INNER JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```

Tez avoids unneeded writes to HDFS

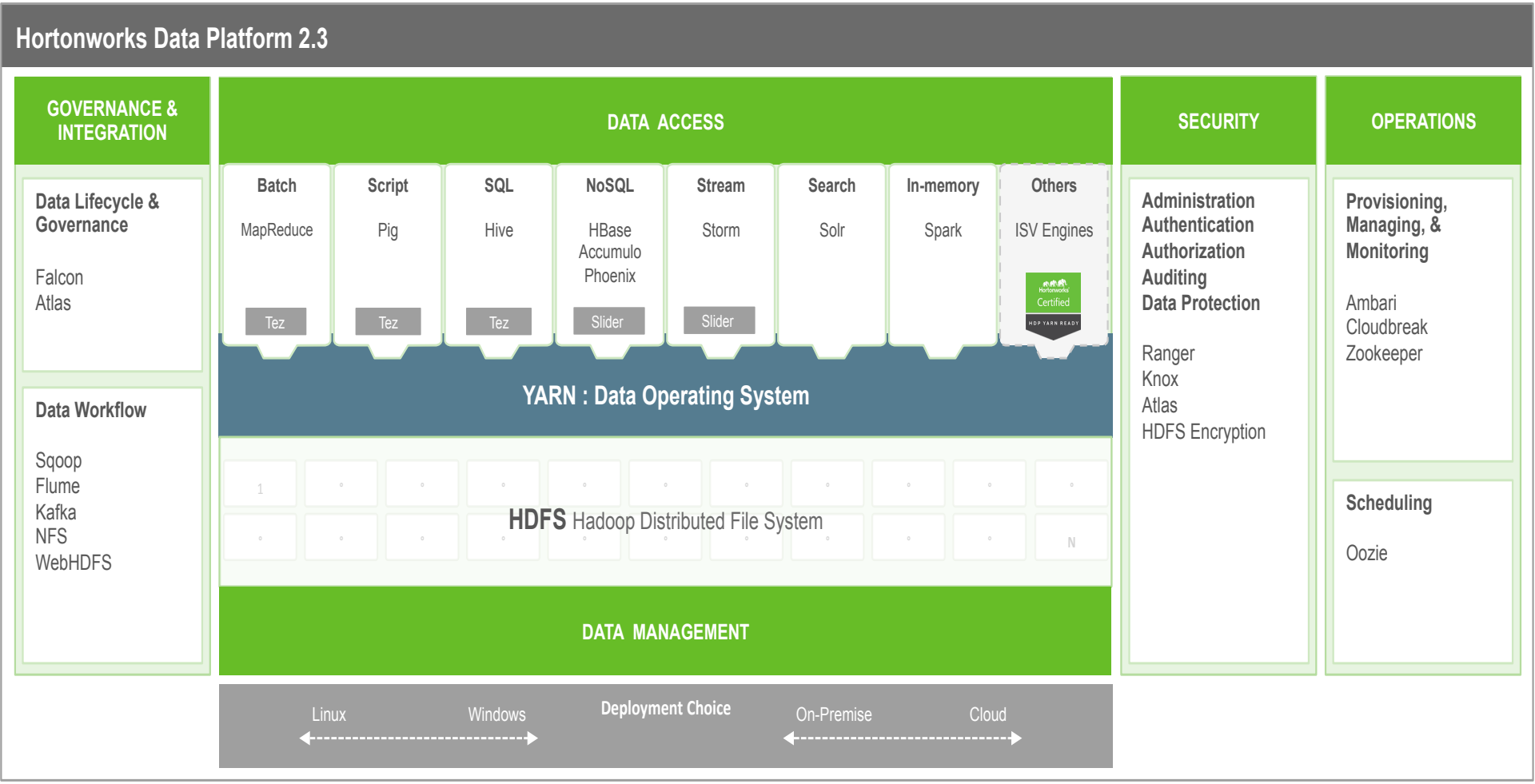
Hive – MapReduce



Hive – Tez



The Hadoop Ecosystem



The Hadoop Ecosystem

HDFS – The Hadoop Distributed File System.

Map/Reduce – A programming model for large datasets, Parallel & Distributed.

Pig – A data processing engine using scripting language. Uses M/R & Tez.

Hive – A data processing engine using SQL-like language. Uses M/R & Tez.

Hcatalog – Provides a relational view of HDFS data .

YARN – An OS-like layer that manages resources.

SPARK – Data analytics engine that uses in-memory data stores.

Tez – Workload optimizer for HDFS.

Kafka – Message Broker (Publish/Subscribe -- similar to MQSeries, MSMQ).

Hbase – A wide-column store NoSQL database engine.

Flume – Ingestion of streaming data (e.g. logs).

Sqoop – Ingestion of data from relational sources.

Solr – full text search

Oozie – Job scheduler.

Ambari – Sysadmin console

Ranger – Security manager

https://hadoopecosystemtable.github.io/

Getting Started with Hadoop

- Apache
  - https://hadoop.apache.org/
- Cloudera (Hortonworks)
  - https://www.cloudera.com/downloads/hortonworks-sandbox.html

Case Studies

Story # 1 – CMS Reporting for Reimbursement

Story # 2 – Sepsis