

MapReduce & Hadoop

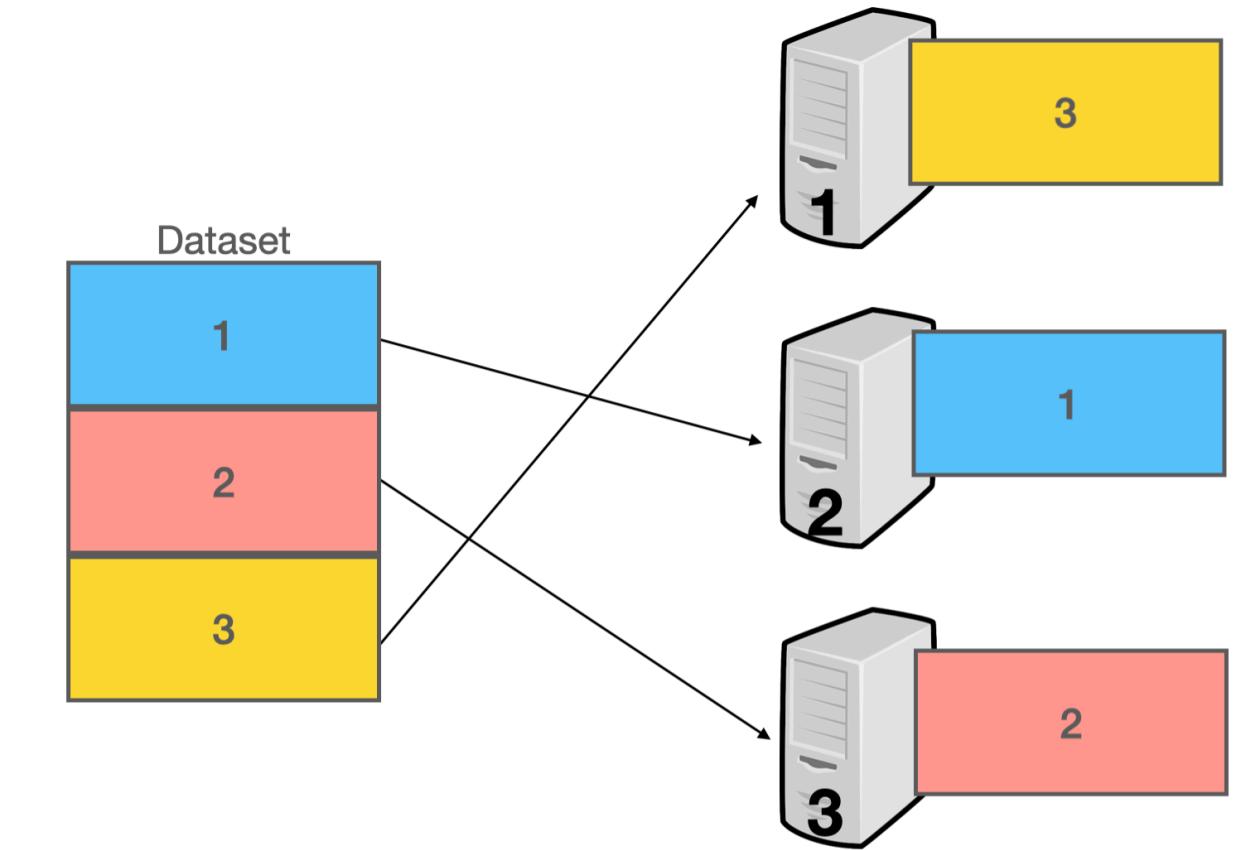
Big Data Management

19-09-2025

Previously...

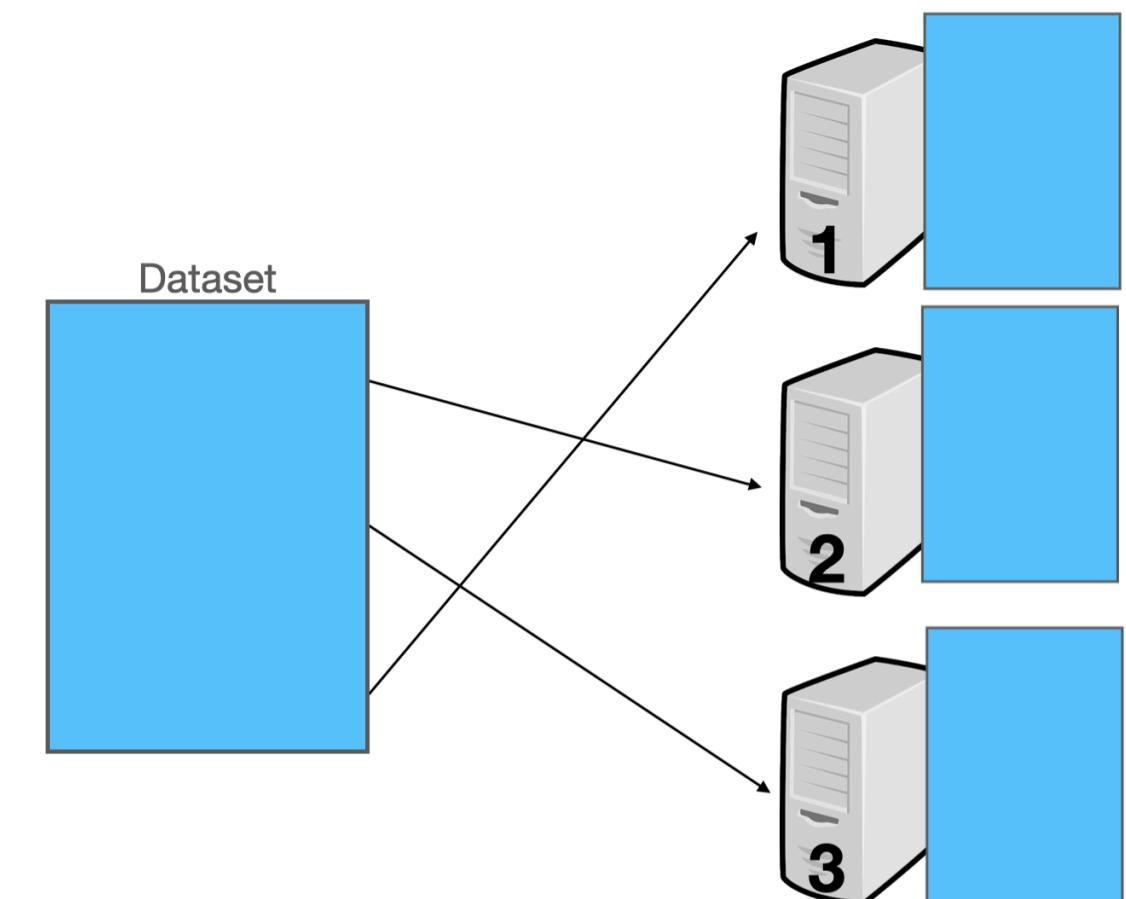
◆ Partitioning

- ◆ Splitting data into (typically disjoint) subsets —> **partitions**
 - ◆ hash vs. range partitioning
- ◆ Each partition is assigned to a different node (aka sharding)
- ◆ **Rebalancing** partitions when nodes are added



◆ Replication

- ◆ Keeping a **replica** (i.e., copy) of the same data on different nodes
- ◆ Most common strategy: **leader-follower**
- ◆ **Synchronous** vs. **asynchronous**
- ◆ Different consistency models for handling **replication lag**



From distributed systems to big data systems

- ♦ Working with distributed systems: totally different from writing software on a single computer
- ♦ Problems with distributed systems
 - ♦ Nondeterministic behaviour
 - ♦ Partial failures
 - ♦ Debugging becomes very hard
 - ♦ High probability of large variance in runtime/throughput performance



Need for new tools

Google cluster idea (2003)

- ◆ **Failures are the norm**
 - ◆ 1 server → may stay up 3 years (1,000 days)
 - ◆ 10,000 servers → expect to lose 10 per day
 - ◆ Ultra-reliable hardware doesn't really help (at large scales, even most reliable hardware fails, albeit less often)
 - ◆ Software still needs to be fault-tolerant
- ◆ **Data is growing:** either large files or billions of small files
- ◆ **Append-only** instead of overwriting
 - ◆ Random writes are practically nonexistent

Use of commodity machines!

Goal: throughput and not peak performance

Need for new tools

- ◆ Distributed File Systems (DFS)
 - ◆ Store petabytes of data in a cluster
 - ◆ Transparently handle fault-tolerance and replication
- ◆ Parallel Processing Platforms
 - ◆ Offer a programming model to allow developers to easily write distributed applications
 - ◆ Alleviate developer from handling concurrency, network communication, and machine failures
 - ◆ Move computation to data, not data to computation

In this lecture...

- ♦ Distributed file systems (DFS)
- ♦ The MapReduce distributed data processing paradigm
- ♦ Hadoop/MapReduce under the hood



In this lecture...

- ♦ **Distributed file systems (DFS)**
- ♦ The MapReduce distributed data processing paradigm
- ♦ Hadoop/MapReduce under the hood

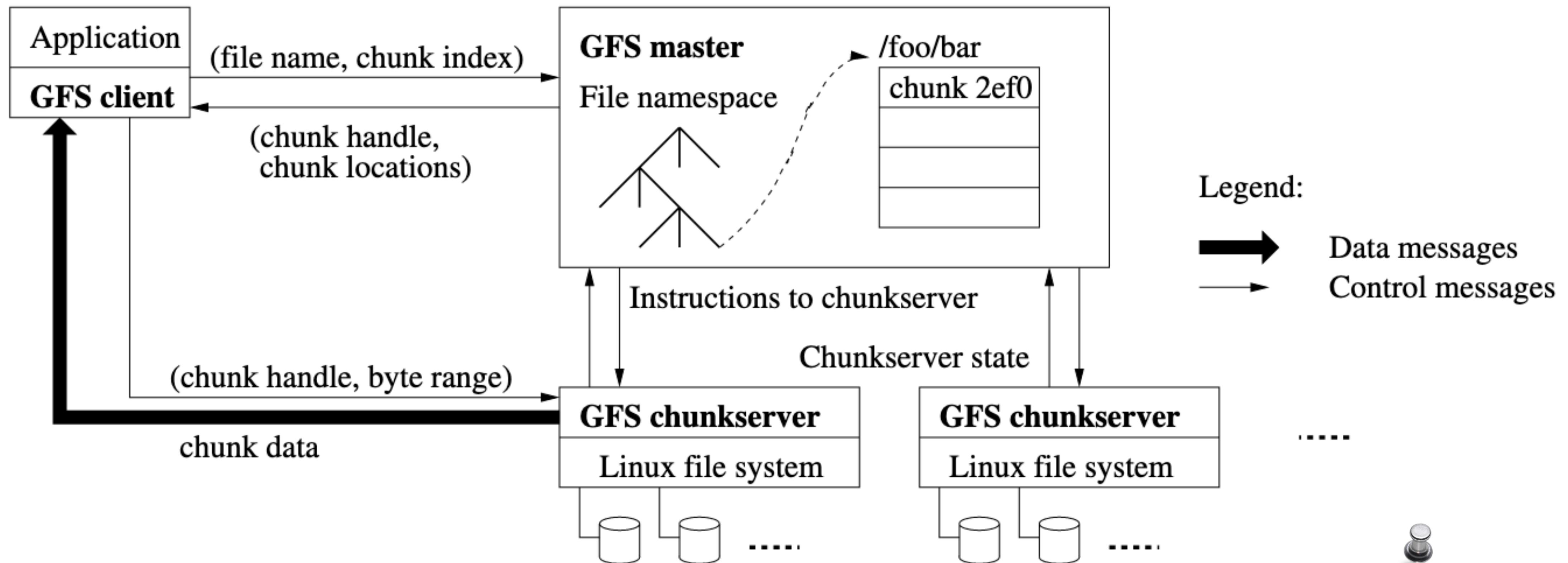


Distributed File System (DFS)

- ◆ Files should be:
 - ◆ Enormous
 - ◆ Rarely updated
- ◆ Stores very large data files **efficiently** and **reliably**
- ◆ Files partitioned into **fixed-sized chunks** (eg., 64MB)
 - ◆ Stored as Linux files
- ◆ Chunks are replicated (typically 3 times)

Google File System (GFS) architecture

- ◆ Single master, multiple chunk servers



Master is potential **single point of failure / scalability bottleneck** —>
Use **shadow masters**

Hadoop Distributed File System (HDFS)

- ♦ Distributed file systems manage the storage across a network of machines
- ♦ HDFS is Hadoop's flagship filesystem

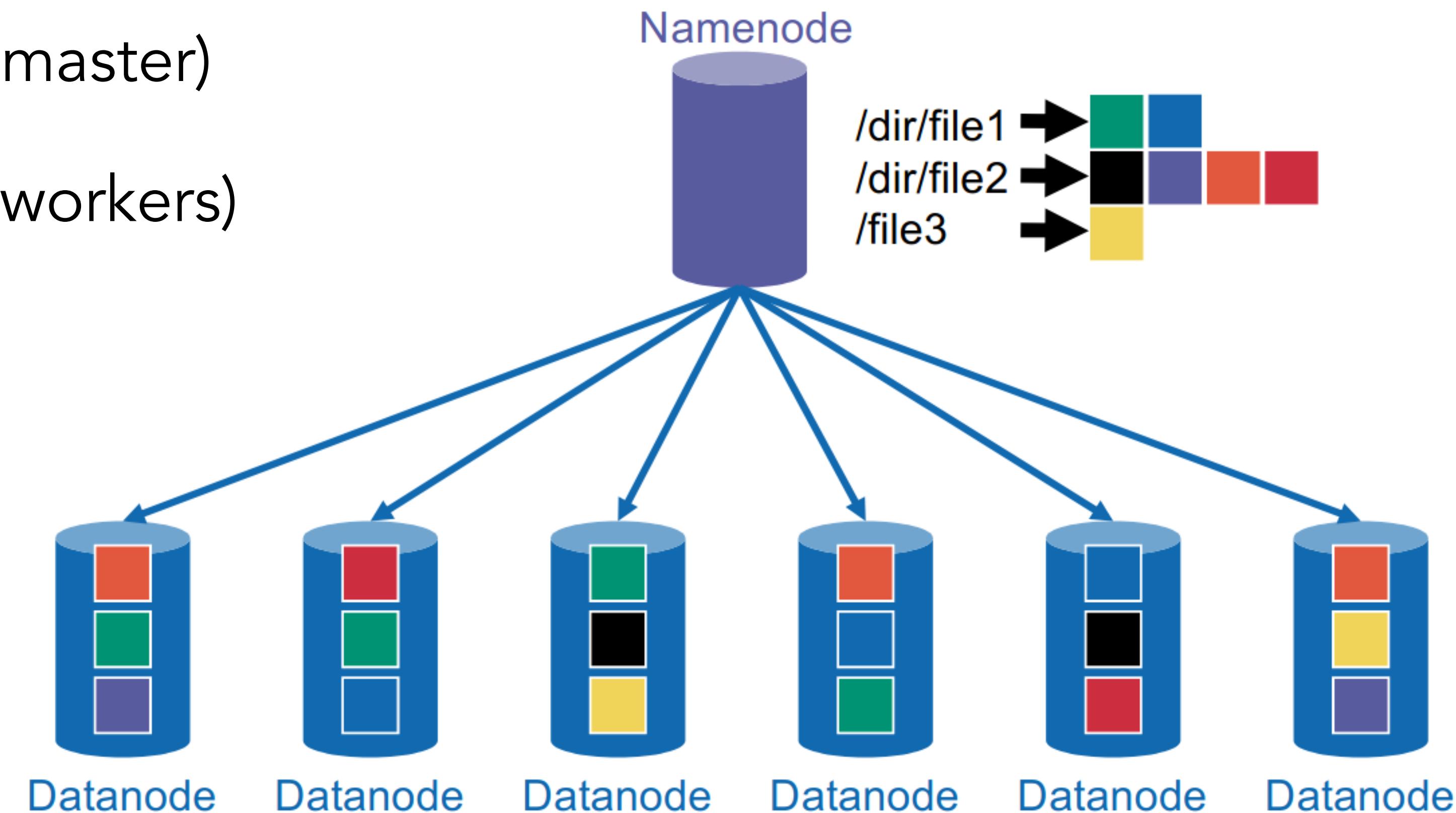
HDFS Block

- ◆ HDFS stores very large files
- ◆ Files are **split** into block-sized chunks (by default 64MB)
- ◆ With this block abstraction:
 - ◆ A file can be larger than a single disk in the network
 - ◆ HDFS blocks are larger than disk blocks
 - ◆ Storage system is simplified (metadata for blocks)
 - ◆ **Replication** for fault-tolerance and availability

HDFS architecture

- ◆ Two types of HDFS nodes

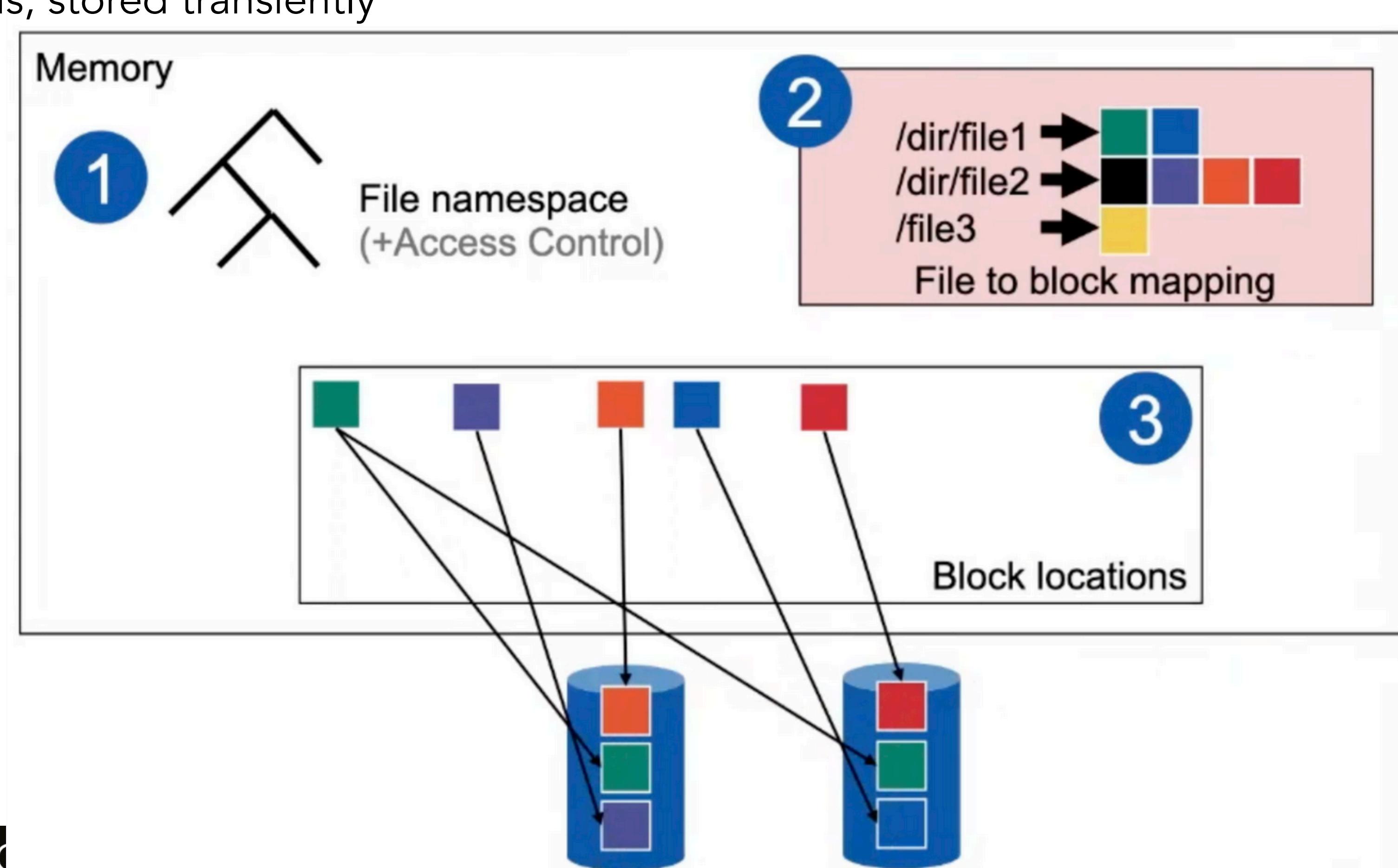
- ◆ One **namenode** (the master)
- ◆ Multiple **datanodes** (workers)



Namenode

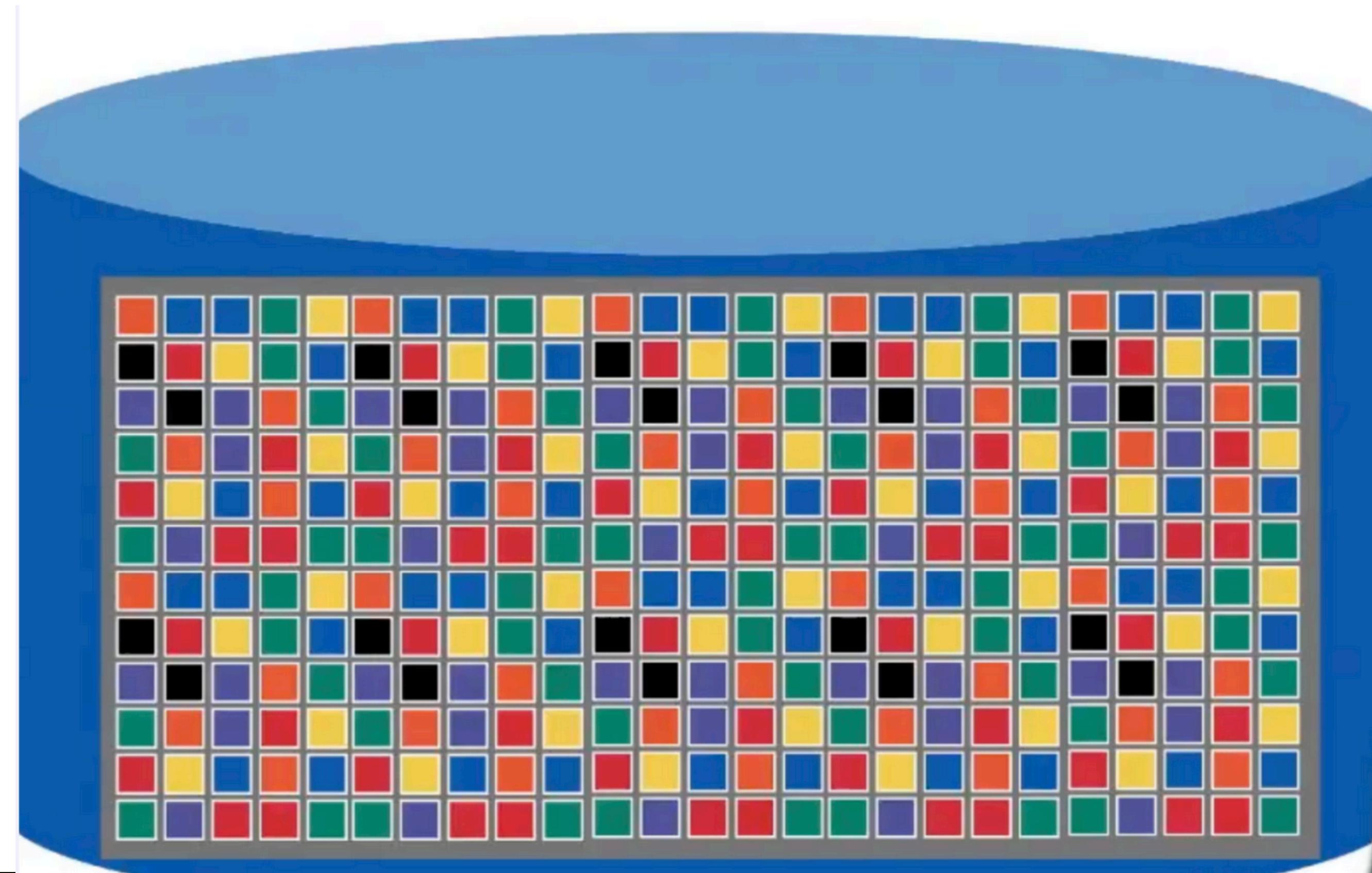
- ◆ Namenode manages the filesystem namespace

- ◆ File system tree and metadata, stored persistently
- ◆ Block locations, stored transiently

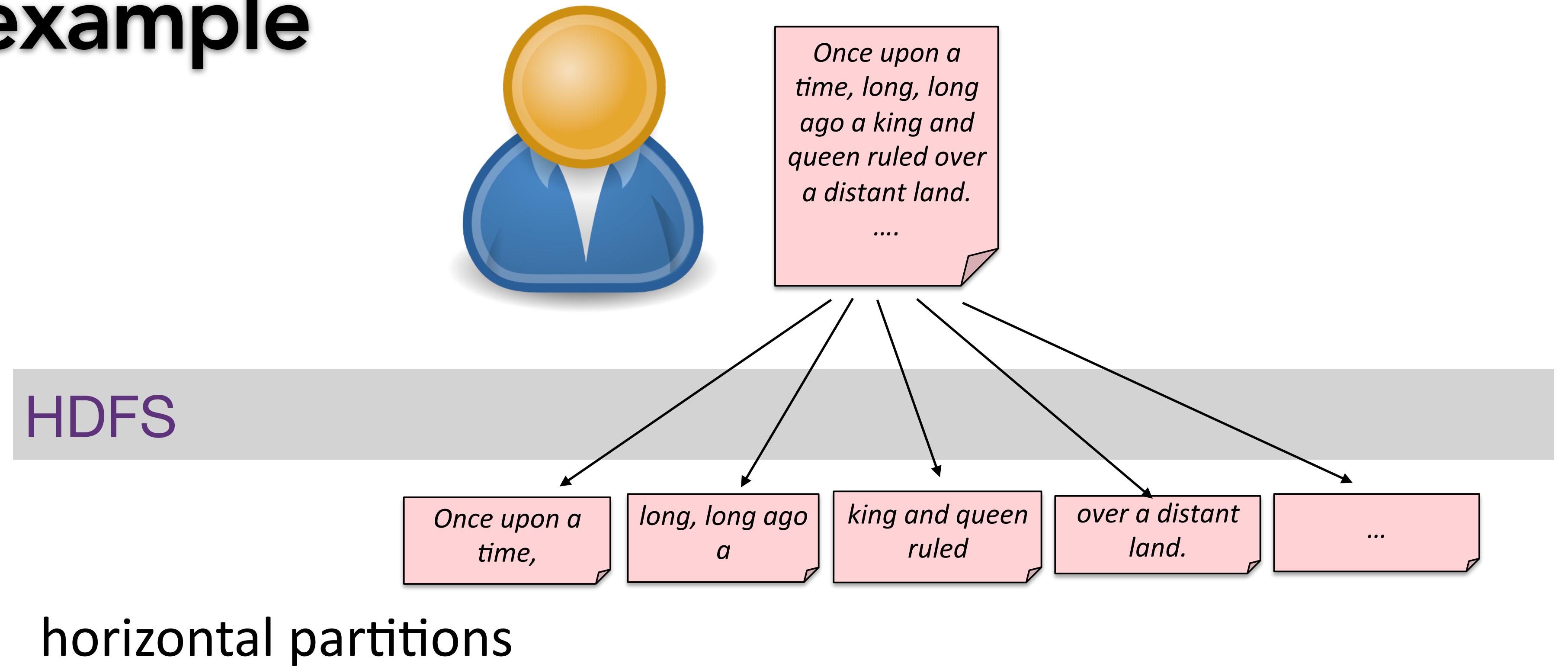


Datanode

- ◆ Datanodes store and retrieve data blocks
 - ◆ When requested from clients or namenode
 - ◆ Also send list of stored blocks periodically to namenode



HDFS example

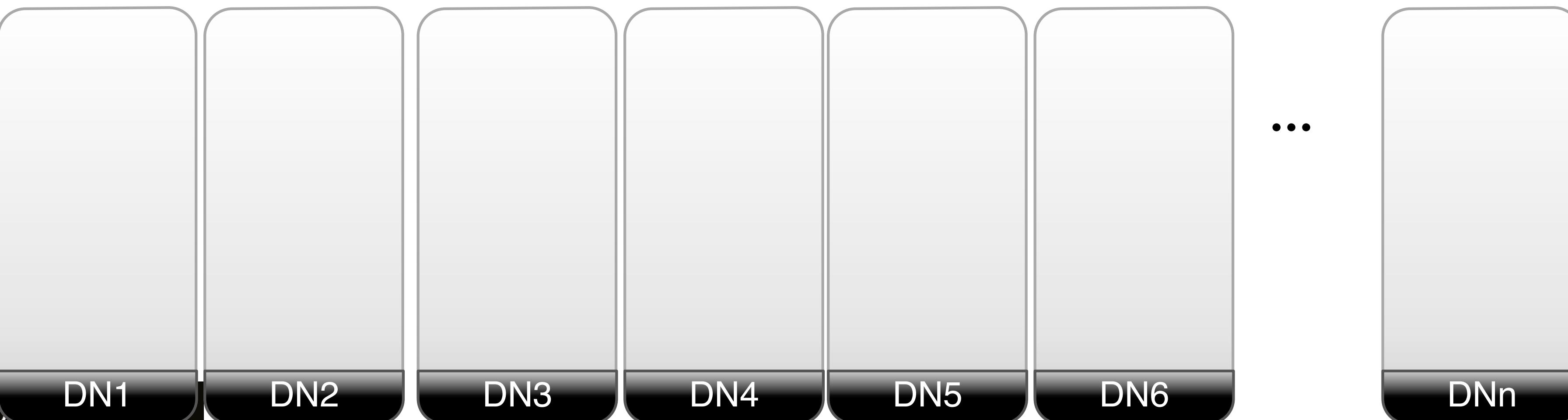
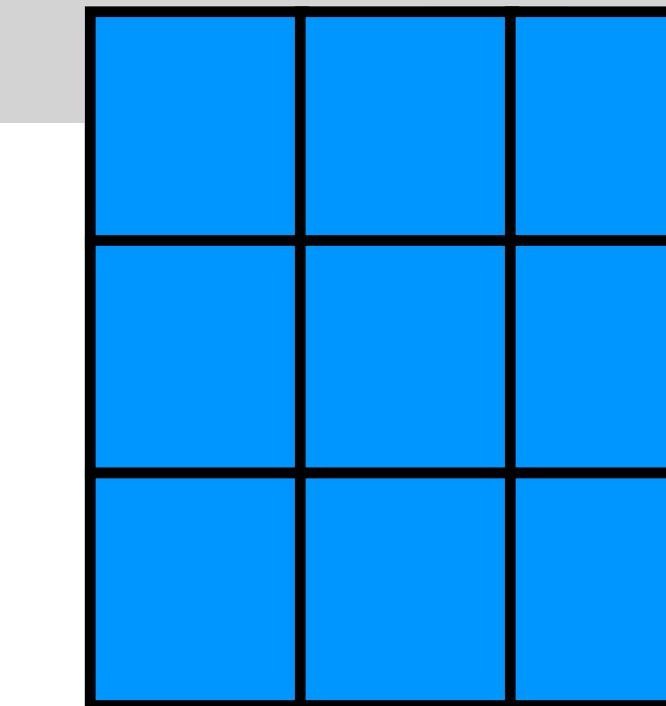


HDFS example

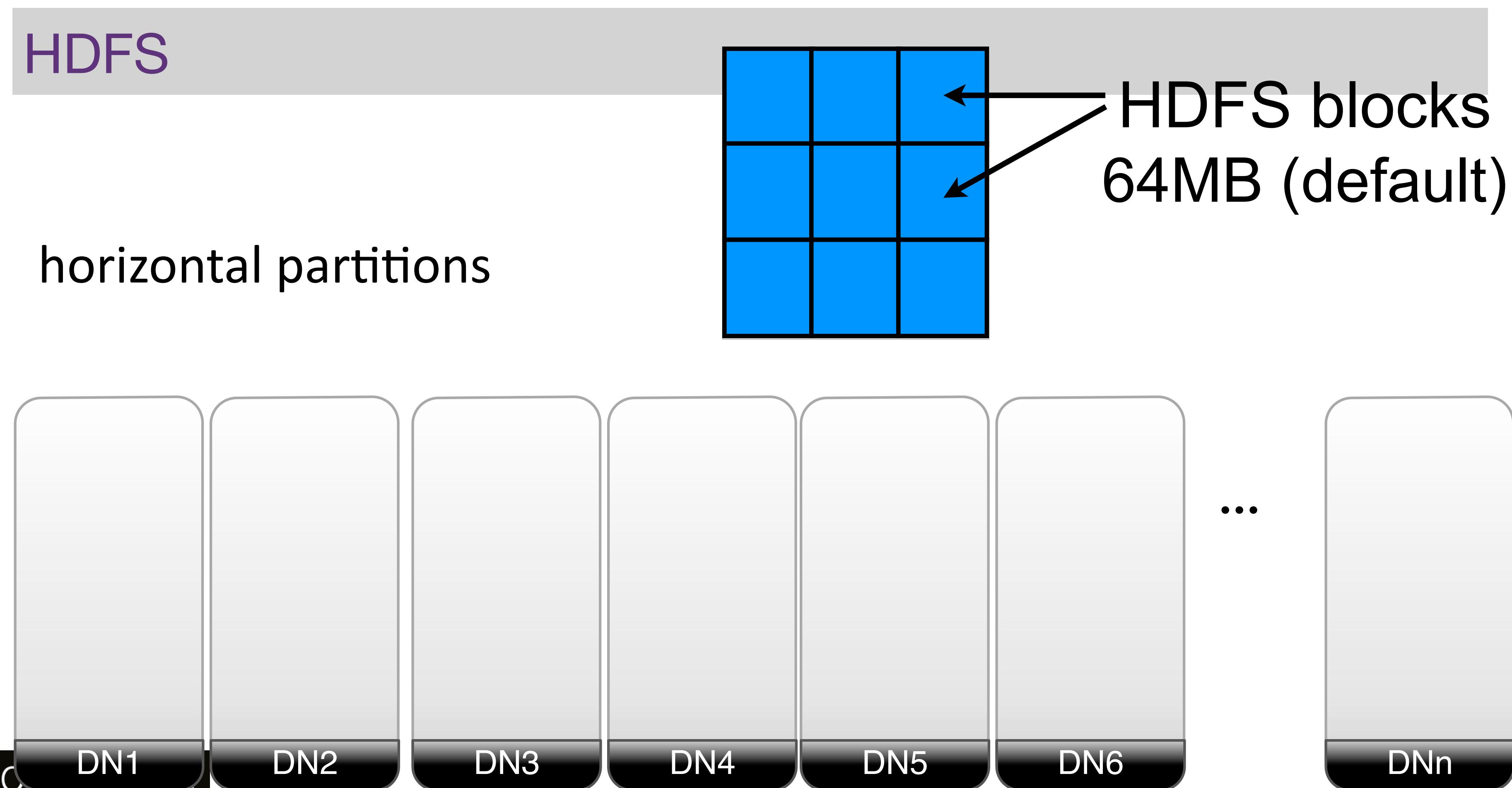
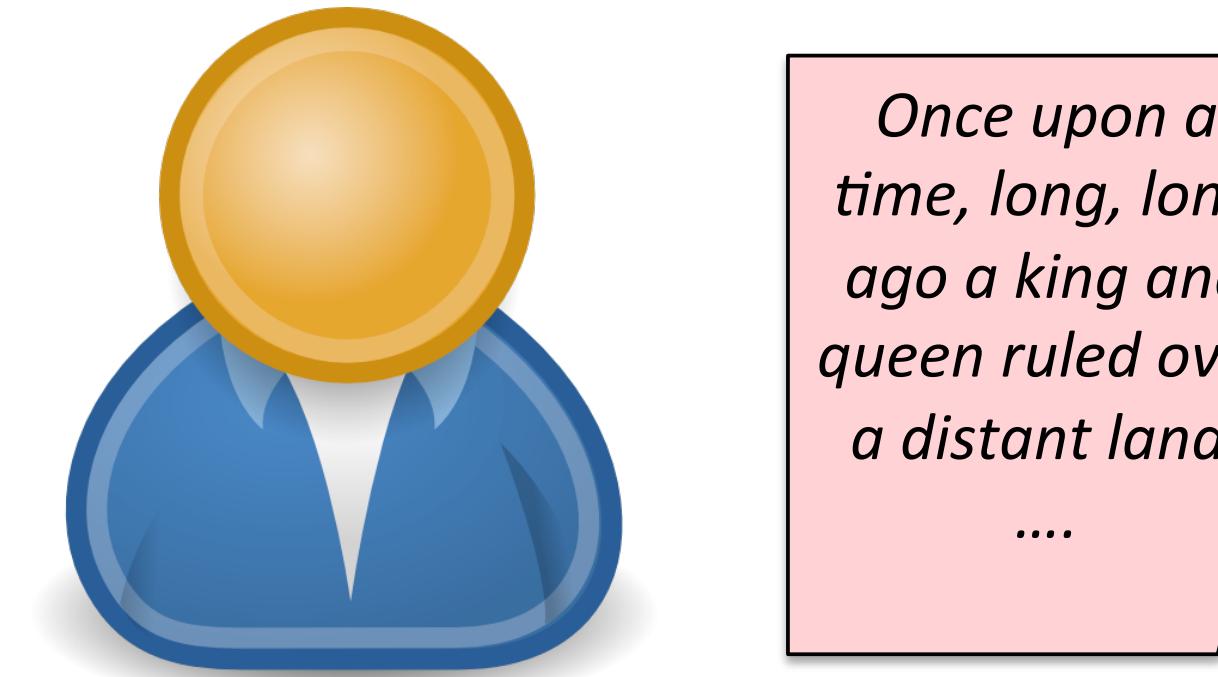


HDFS

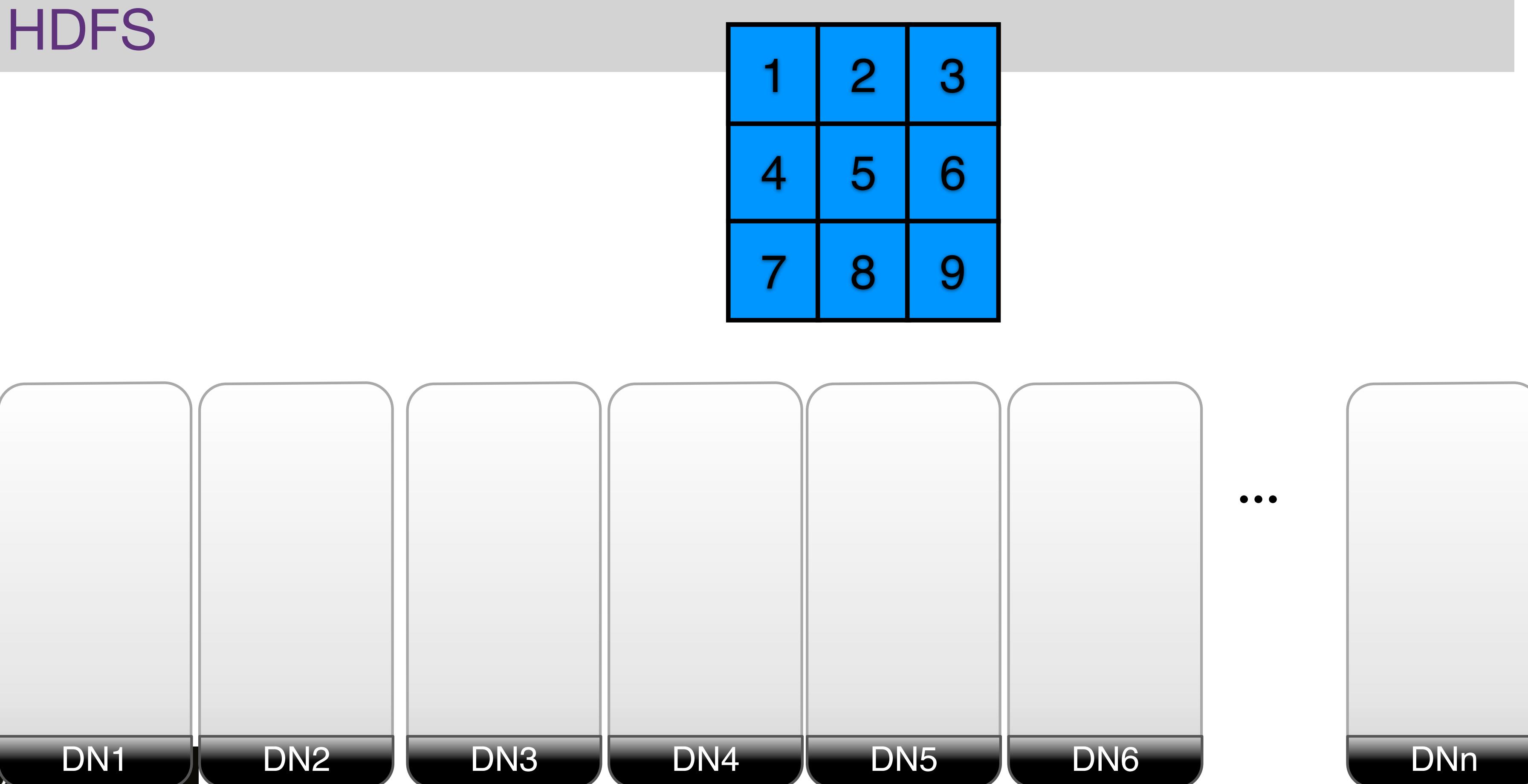
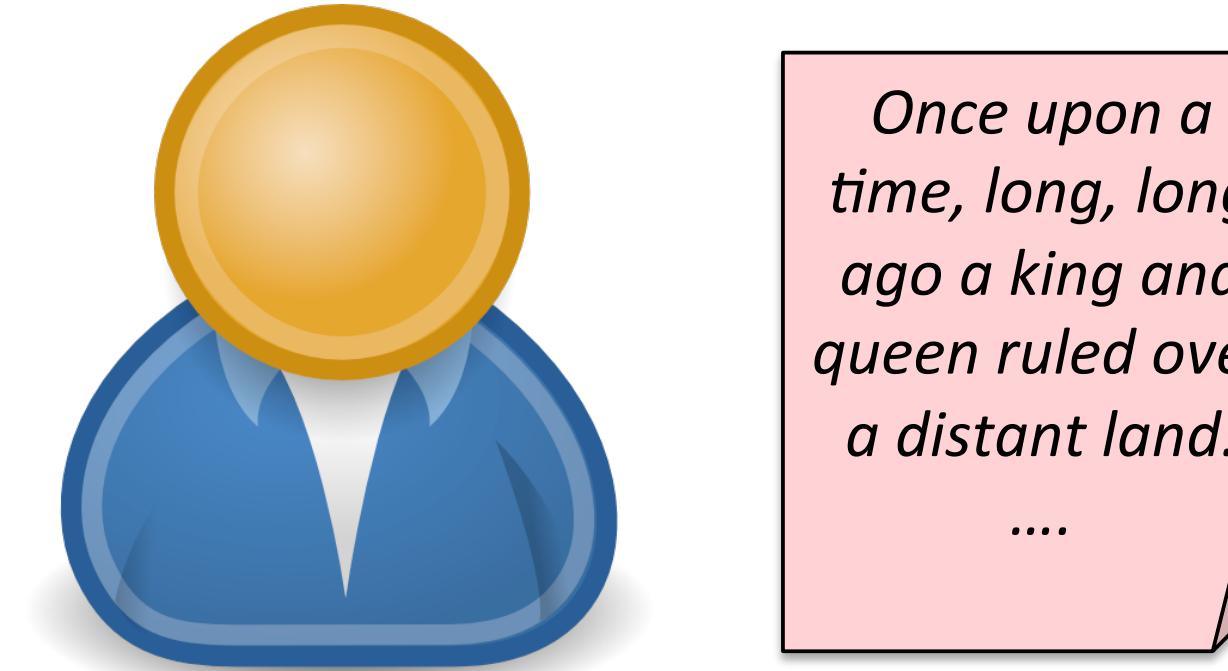
horizontal partitions



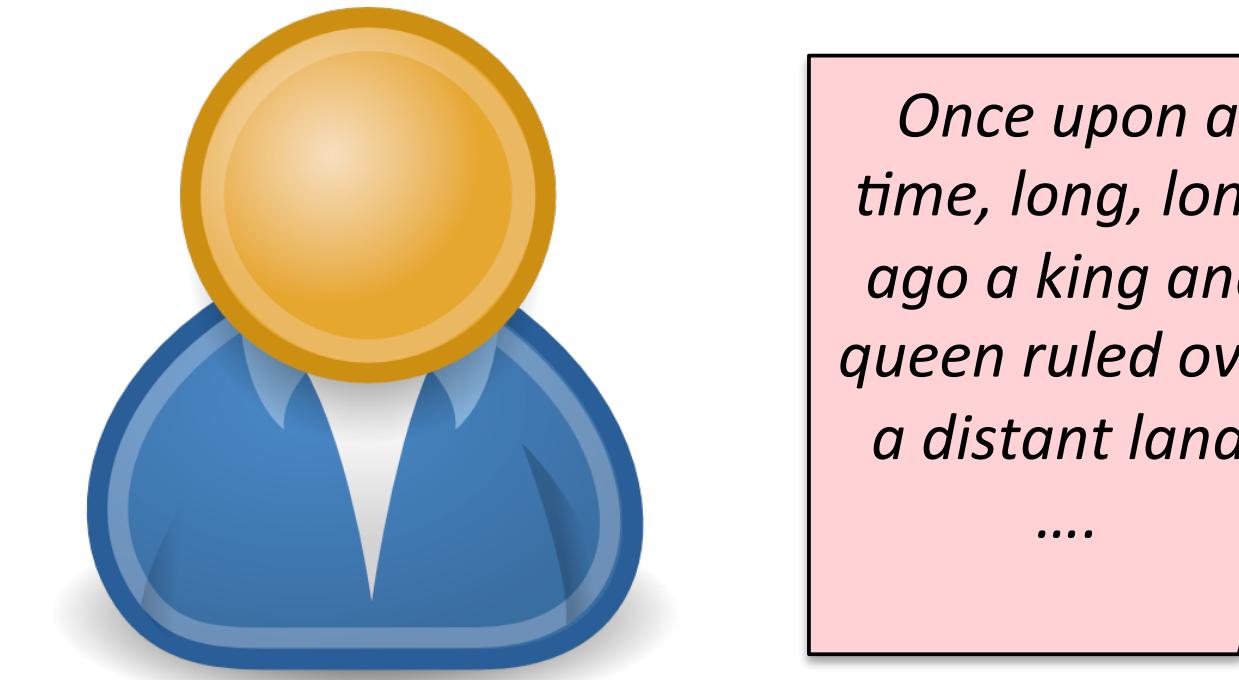
HDFS example



HDFS example



HDFS example

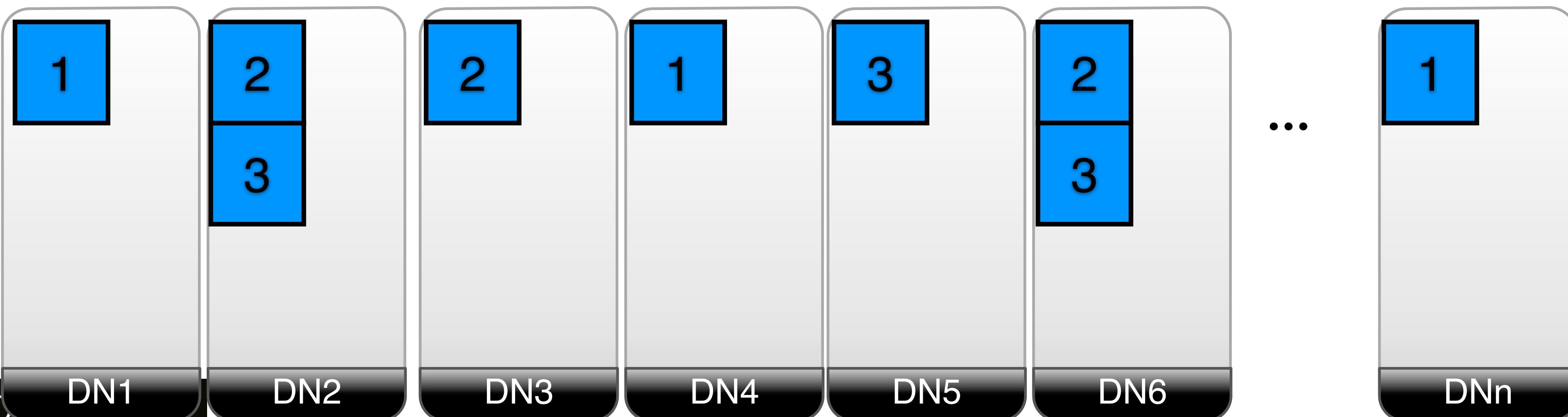


Once upon a time, long, long ago a king and queen ruled over a distant land.

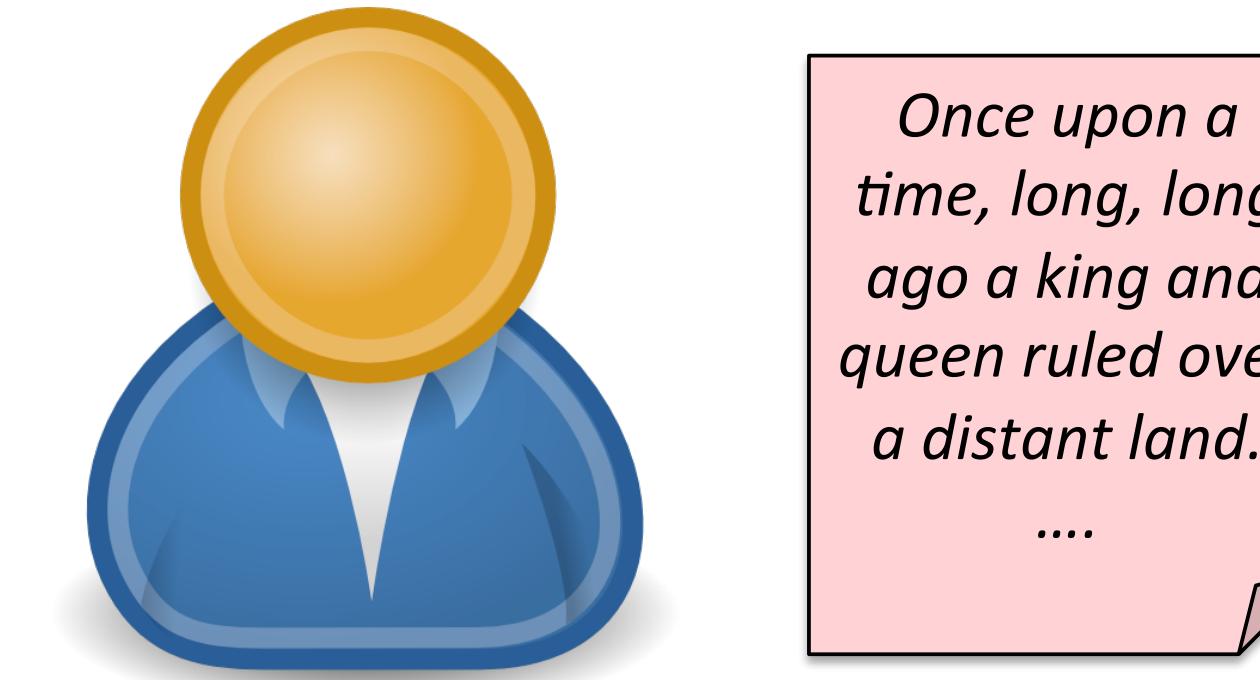
....

HDFS

4	5	6
7	8	9

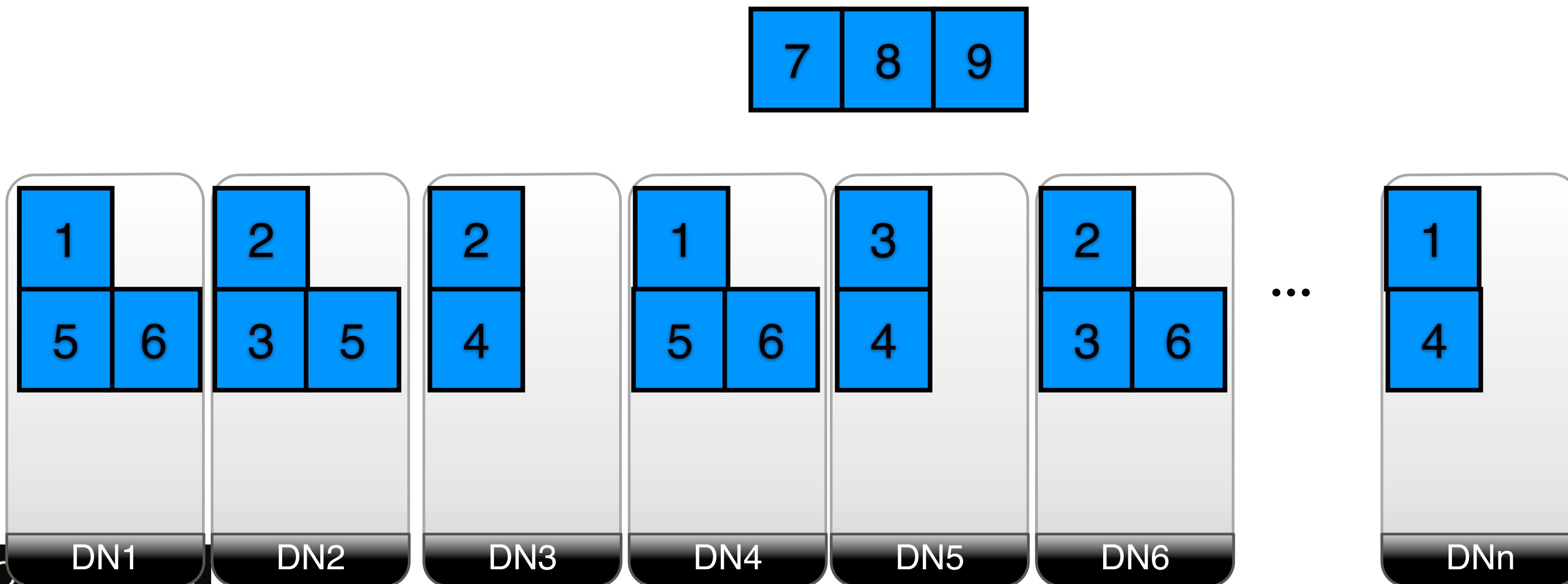


HDFS example

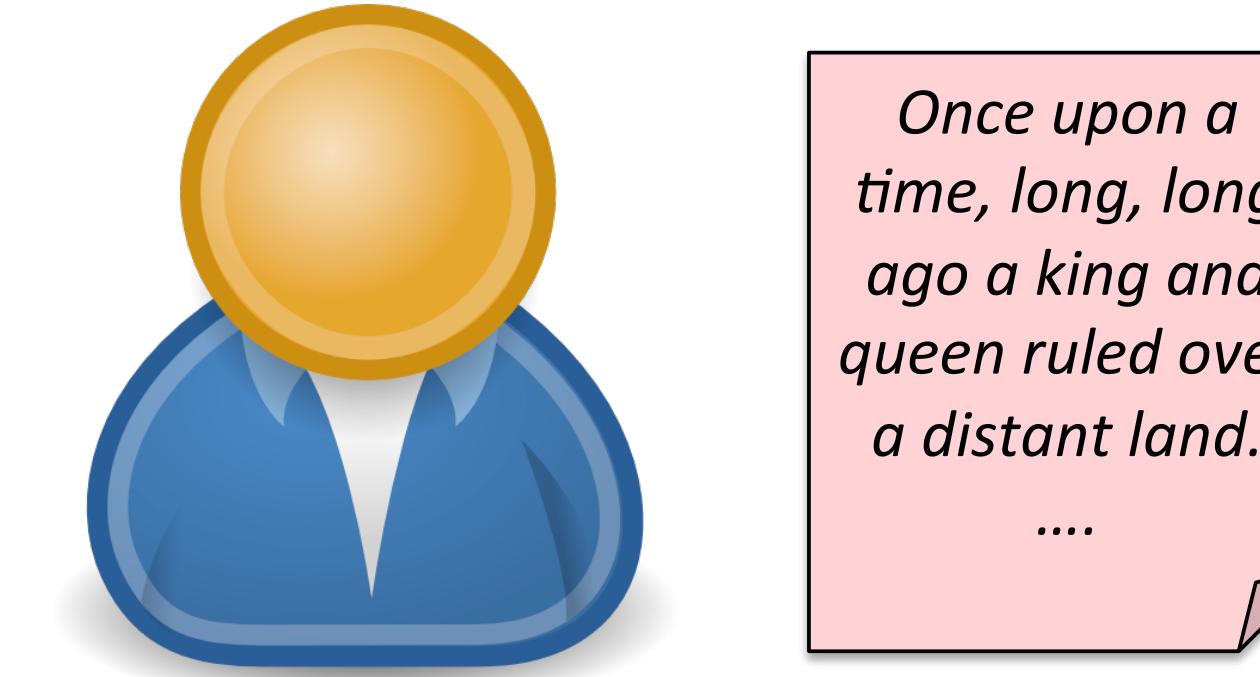


*Once upon a
time, long, long
ago a king and
queen ruled over
a distant land.
....*

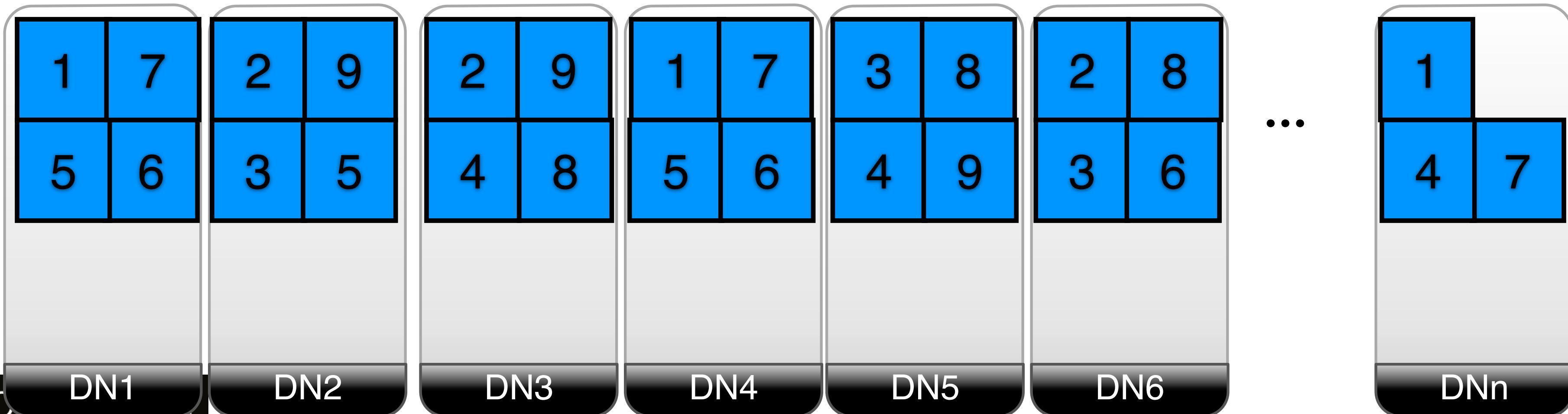
HDFS



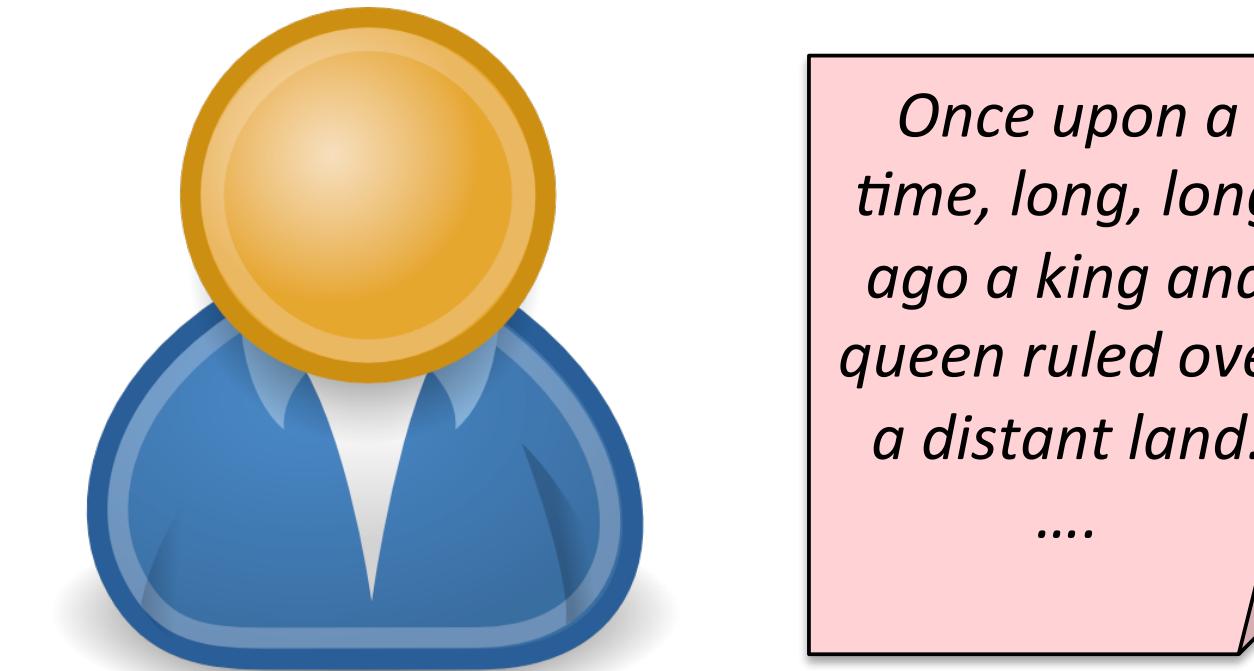
HDFS example



HDFS



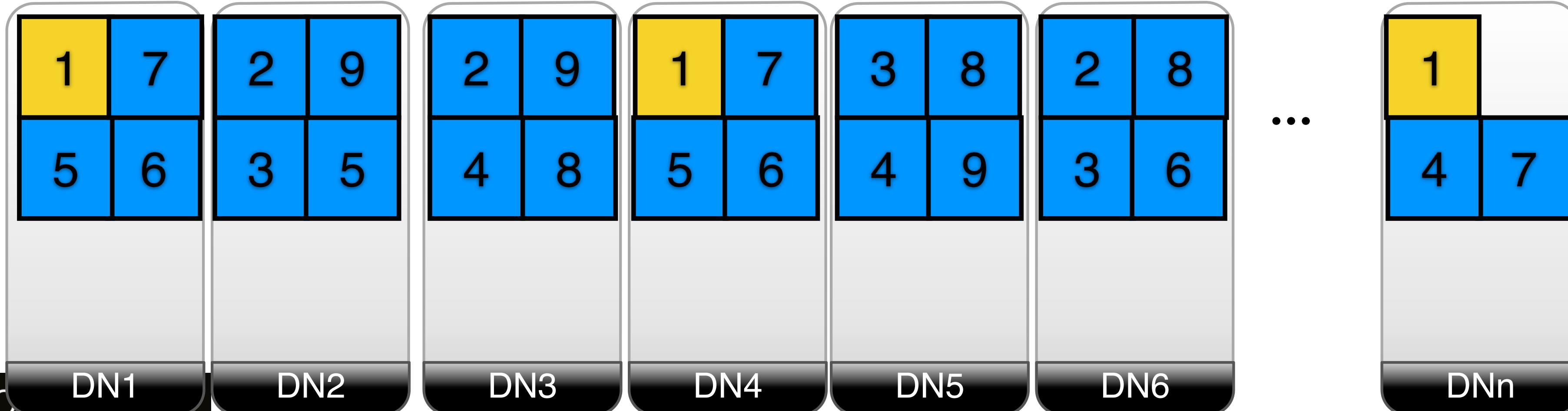
HDFS example



HDFS

partitioning

replication (by default factor 3)



HDFS example

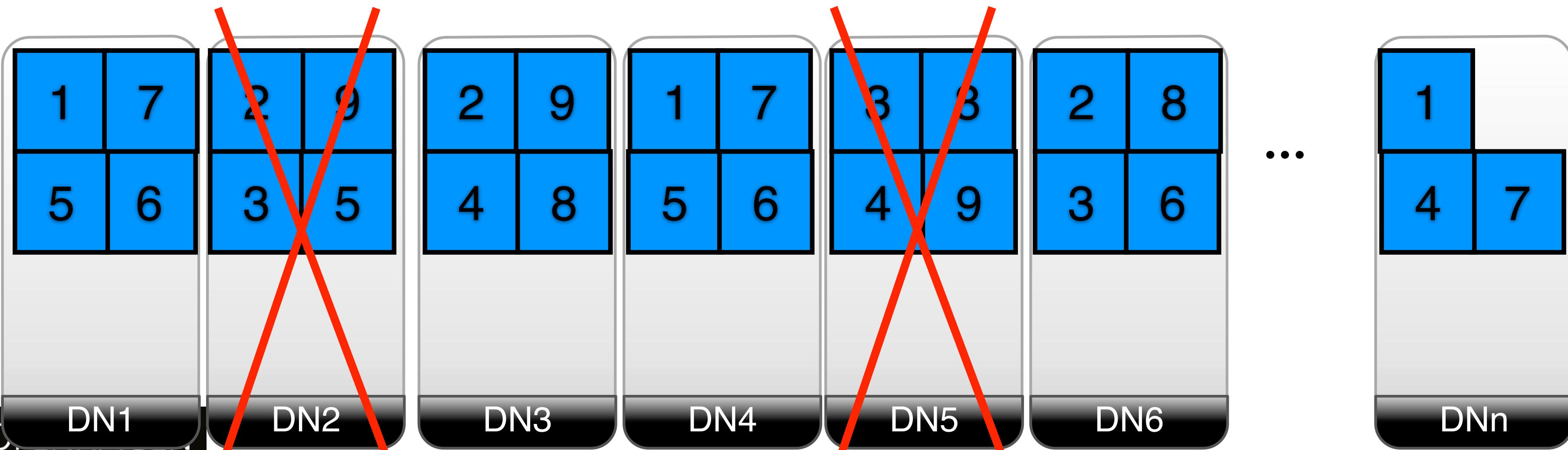


Fault-tolerance



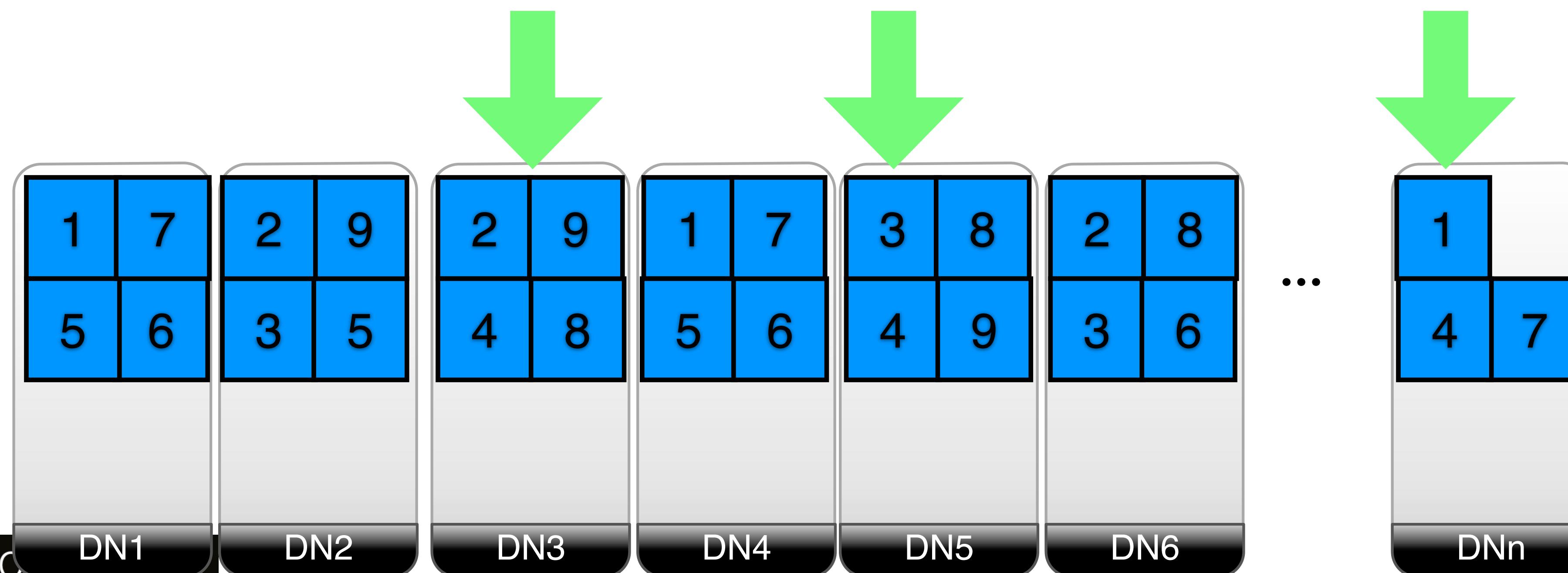
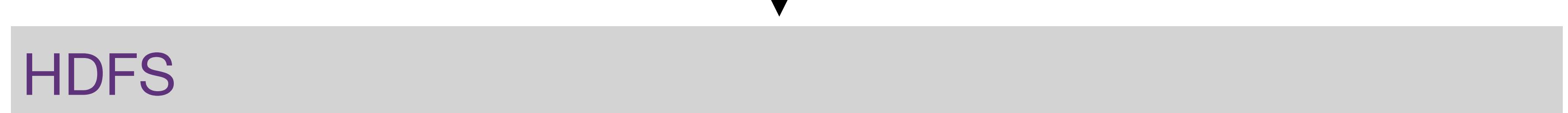
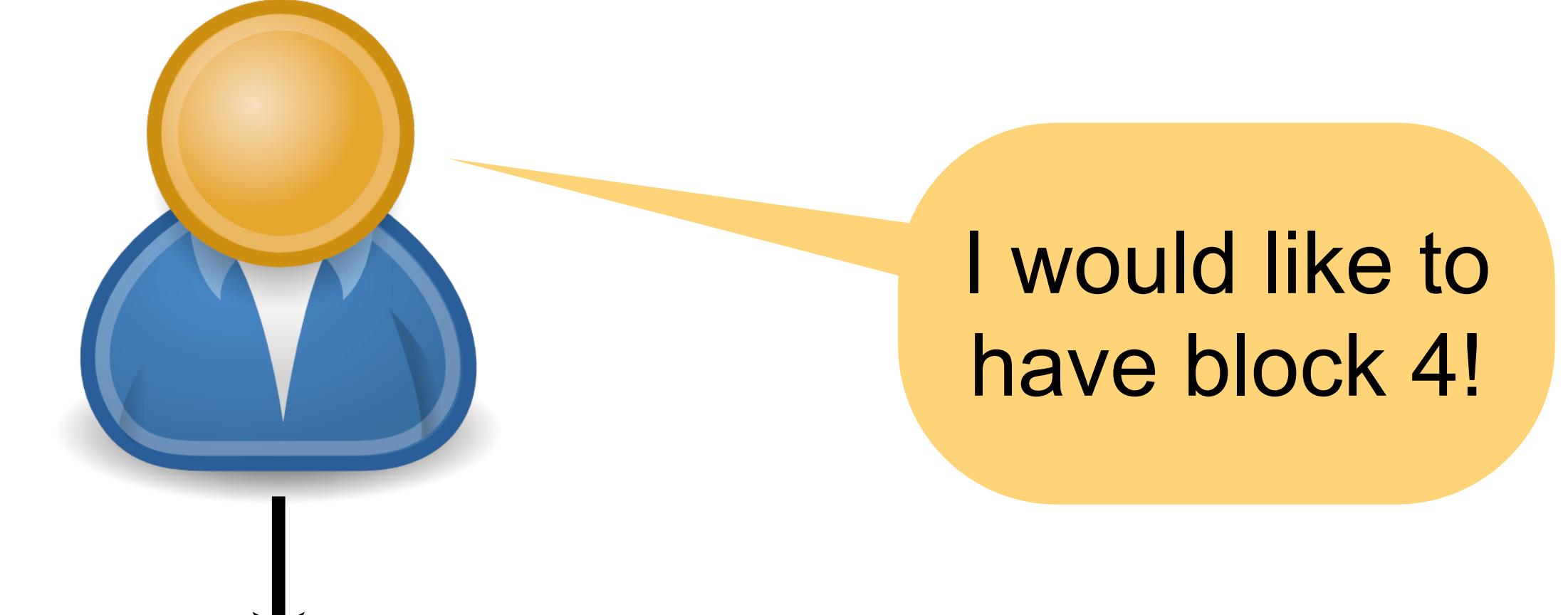
Once upon a
time, long, long
ago a king and
queen ruled over
a distant land.
....

HDFS



HDFS example

Load balancing



HDFS example

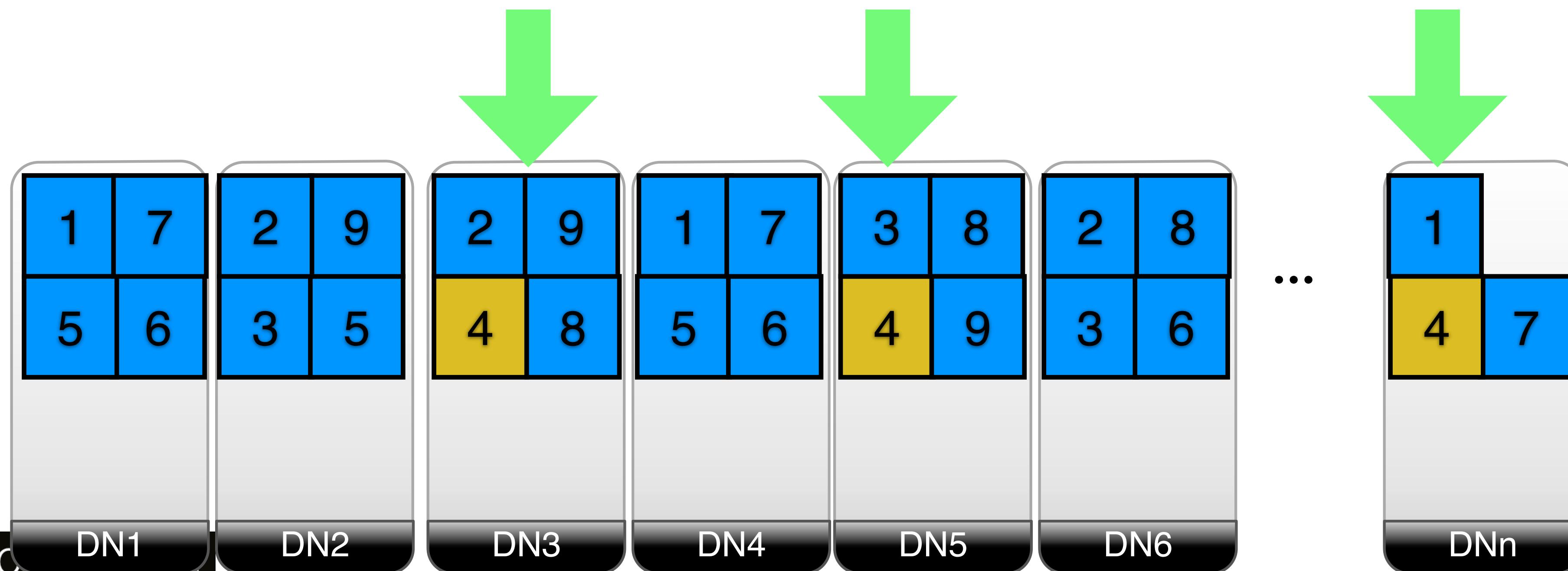


Load balancing

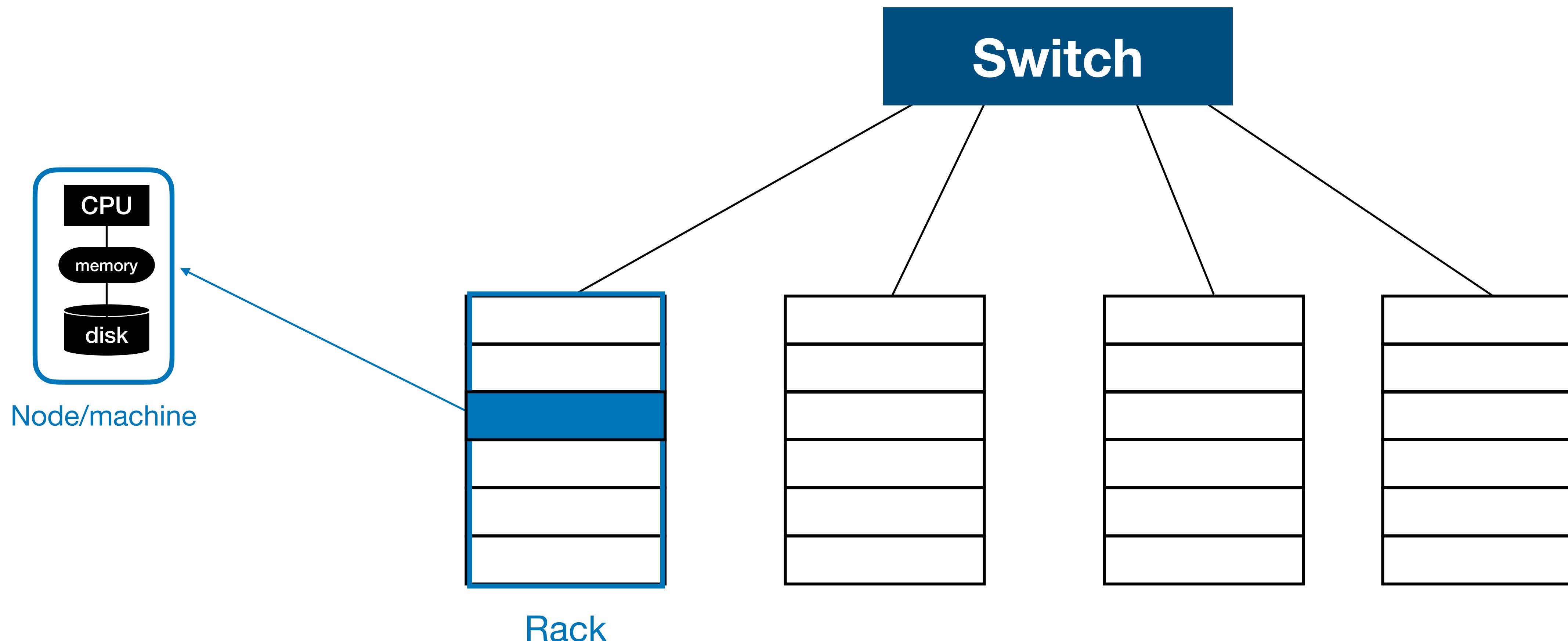


I would like to have block 4!

HDFS



Recall: Typical cluster configuration



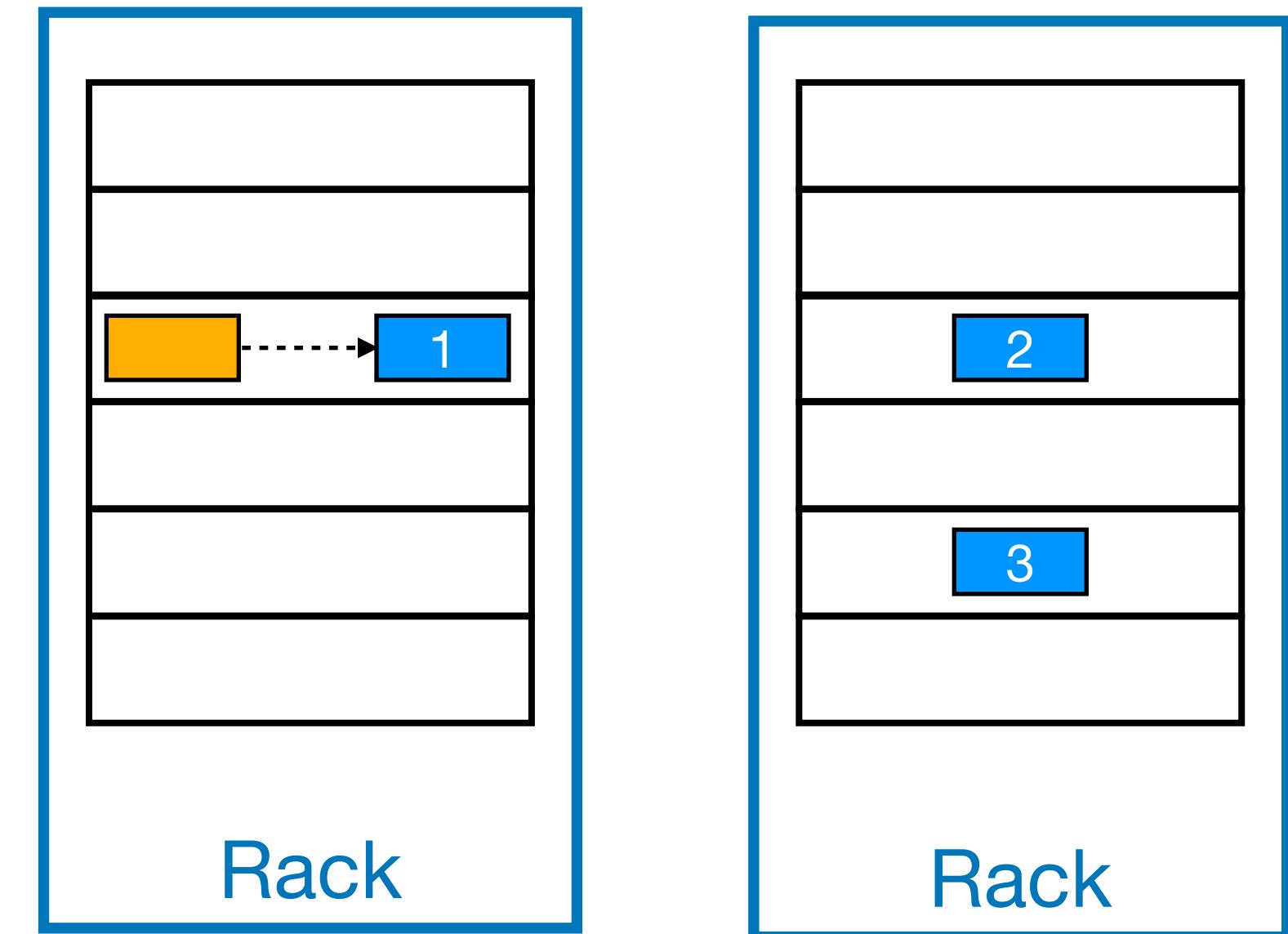
Replicas placement

- ♦ Goal

- ♦ Reliability
 - ♦ Write and read bandwidth

- ♦ Block distribution
 - ♦ Default strategy

- ♦ First replica: on the client node
 - ♦ Second: random, off rack
 - ♦ Third: same rack as second, different node
 - ♦ More: random



Today's lecture

- ◆ Distributed file systems (HDFS)
- ◆ **The MapReduce distributed data processing paradigm**
- ◆ Hadoop/MapReduce under the hood



What is MapReduce and Hadoop?

- ◆ **MapReduce**: A programming model
 - ◆ Functional style
 - ◆ Users specify Map and Reduce functions
 - ◆ Expressive
- ◆ **Hadoop**: An associated implementation
 - ◆ Automatically parallelized
 - ◆ Automatic partitioning, execution, failure handling, load balancing, communication
 - ◆ Can handle very large datasets in clusters of commodity computers

MapReduce in a nutshell (I)

- ◆ Framework
 - ◆ Read lots of data
 - ◆ **Map**: process a data item / record
 - ◆ **Sort and shuffle**
 - ◆ **Reduce**: aggregate, summarize, filter, transform
 - ◆ Write results
- ◆ Map and Reduce are **user-specified** → used to model a given problem
 - ◆ Input is a **user-defined function**
 - ◆ Sort and shuffle is taken care by the system itself

MapReduce in a nutshell (II)

- ♦ Simple API

- ♦ **Map** (*key, value*) —> {*ikey, ivalue*}
- ♦ **Reduce** (*ikey, {ivalue}*) —> (*key', value'*)

- ♦ **Map phase**: independent processes (mappers) which run **in parallel**

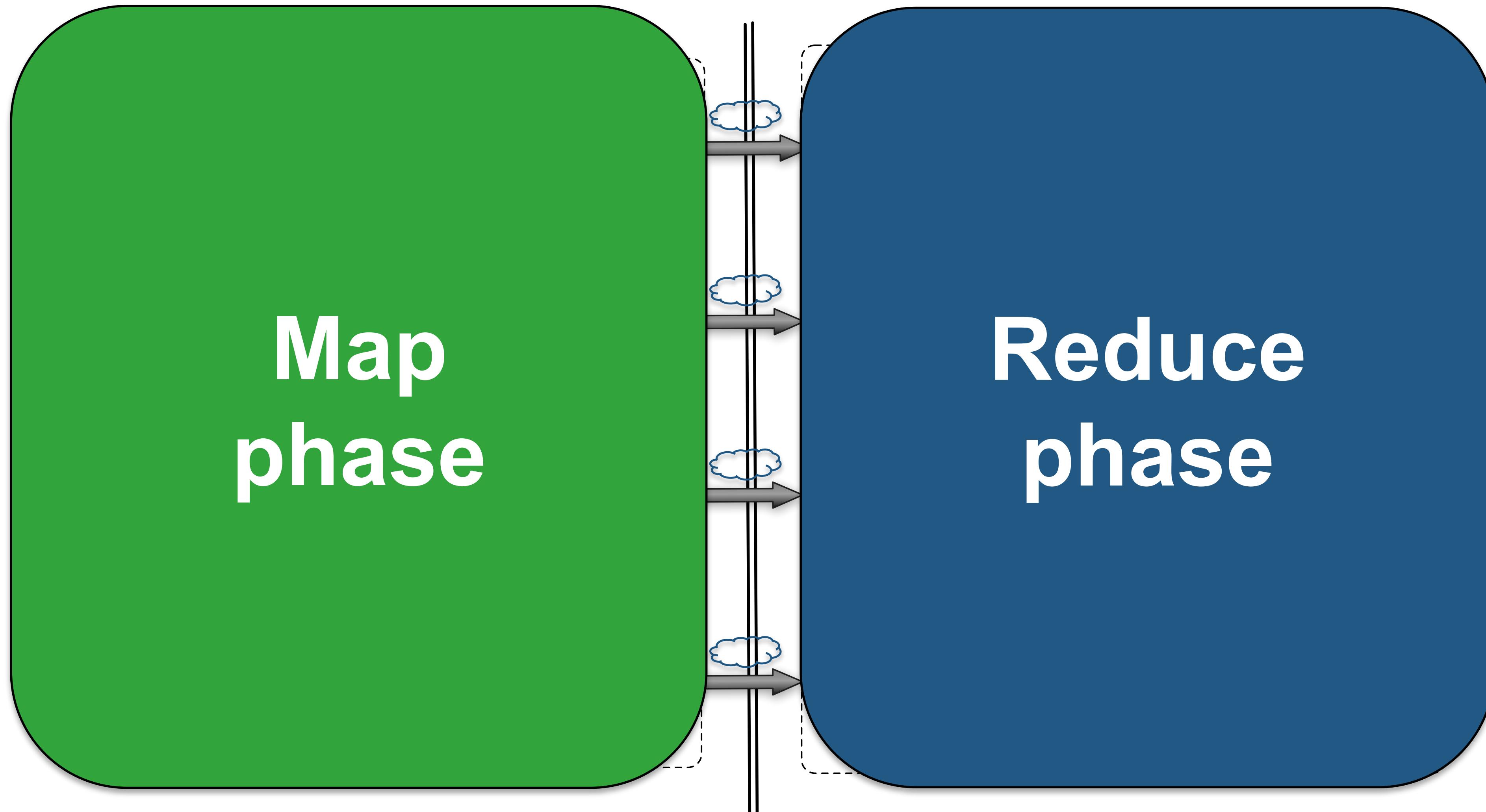
- ♦ operate on the chunks (blocks) of input data
- ♦ output intermediate results

- ♦ **Shuffle phase**: Intermediate results are shuffled through the network

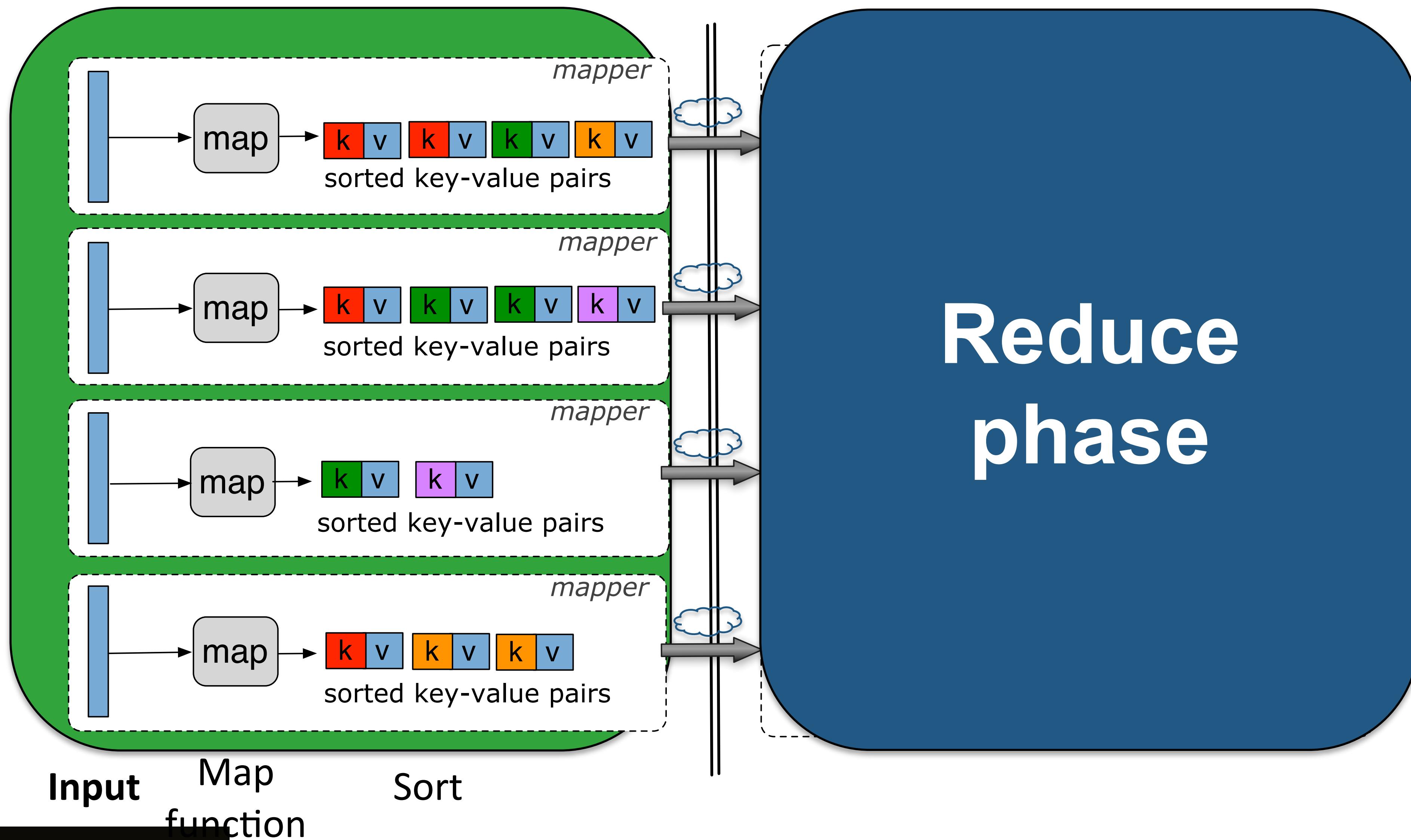
- ♦ **Reduce phase**: independent processes (reducers) which run **in parallel**

- ♦ group intermediate results of the map phase
- ♦ operate on the groups
- ♦ output final results

MapReduce illustration

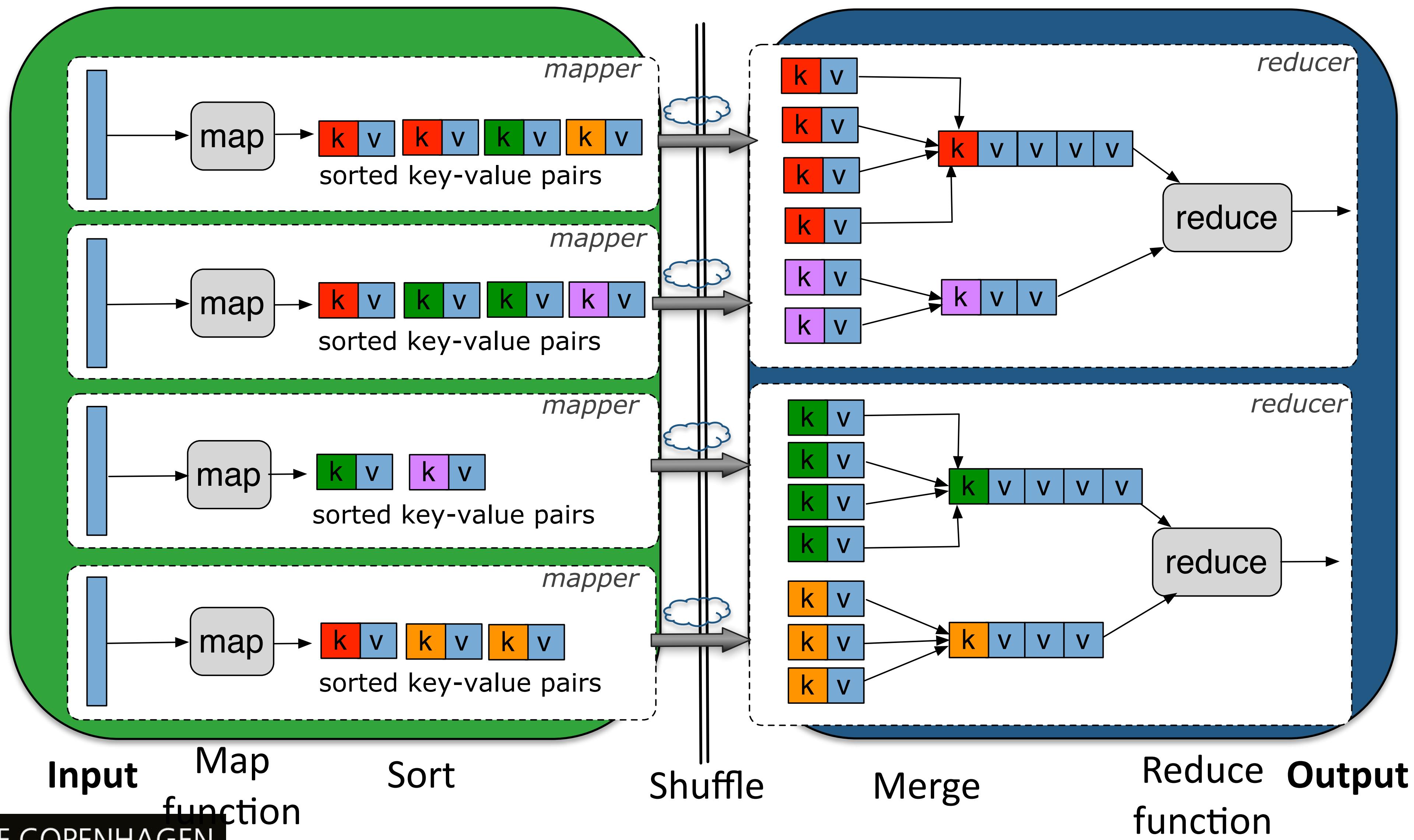


MapReduce illustration



Reduce
phase

MapReduce illustration



Google Use Case: Web Index



WIKIPEDIA

The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export

Create a book
Download
Printable version



WIKIPEDIA

The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

Apache Hadoop

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help improve it or discuss these issues on the talk page.
• This article contains content that is written like an advertisement. (October 2013)
• This article appears to contain a large number of buzzwords. (October 2013)

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.^[3]

The core of Apache Hadoop consists of a storage part (Hadoop Distributed File System (HDFS)) and a processing part (MapReduce). Hadoop splits files into large blocks and distributes them amongst the nodes in the cluster. To process the data, Hadoop MapReduce transforms each code for nodes to process in parallel, based on the data each node needs to process. This approach takes advantage of manipulating the data that they have on hand—to allow the data to be processed faster and more efficiently than conventional supercomputer architecture that relies on a parallel file system where computation and data are connected.^[5]

The base Apache Hadoop framework is composed of the following modules:

- Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
- Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing high-throughput access bandwidth across the cluster;
- Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters.

MapReduce

From Wikipedia, the free encyclopedia

MapReduce is a programming model and an associated implementation for processing large data sets. Similar approaches have been very well known since 1995 with the Message Passing Interface (MPI).

A MapReduce program is composed of a Map() procedure (method) that performs a key-value mapping and a Reduce() method that performs a summary operation (such as counting the number of occurrences of a word). The "infrastructure" or "framework" orchestrates the processing by marshalling the work among available machines in between the various parts of the system, and providing for redundancy and fault tolerance.

The model is inspired by the map and reduce functions commonly used in functional programming forms.^[7] The key contributions of the MapReduce framework are not the actual algorithmic primitives, but rather the distributed execution engine that optimizes the execution engine once. As such, a single-threaded implementation of the MapReduce model is usually only seen with multi-threaded implementations.^[8] The use of this mode of computation makes it easier to implement fault tolerance features of the MapReduce framework come into play.

MapReduce libraries have been written in many programming languages, with the first one being Java. It is part of Apache Hadoop. The name MapReduce originally referred to the programming model, but it is also used as a big data processing model,^[10] and development on Apache Mahout had re-implemented the MapReduce capabilities.^[11]

Contents [hide]

- 1 Overview
- 2 Logical view
 - 2.1 Examples
- 3 Dataflow
 - 3.1 Input reader

Print/export
Create a book
Download as PDF

NASA Topics Missions Galleries NASA TV Follow NASA Downloads About NASA Audiences Search

Apache Hadoop

Chandra X-ray

Apr 10, 2019

Black Hole Image Makes History; NASA Telescopes Coordinated Observations

A black hole and its shadow have been captured in an image for the first time, a historic feat by an international network of radio telescopes called the Event Horizon Telescope (EHT). EHT is an international collaboration whose support in the U.S. includes the National Science Foundation.

f t in p +

IT-UNIVERSITETET I KØBENHAVN

UDDANNELSER EFTERUDDANNELSER FORSKNING ERHVERVSSAMARBEJDE OM ITU ENG

Master i it-ledelse har nu åbent for optag. Søg frem til 22. maj 2023

LÆS MERE HER

UDDANNELSER PÅ ITU

Vælg uddannelse

IT-Universitetet udbyder it-uddannelser inden for spil, digitalt design, data science, softwareudvikling og business.

Vælg efteruddannelse

IT UNIVERSITY OF COPENHAGEN

Google Use Case: Web Index

Google big data X | 

All Images News Videos Books More Tools

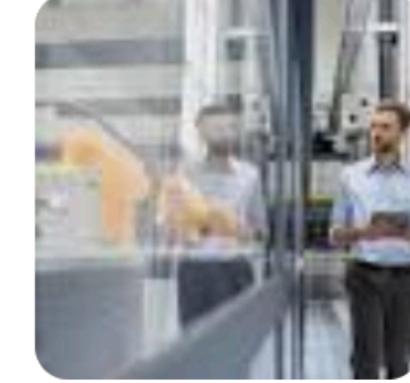
About 18.300.000 results (0,45 seconds)


S The Stack

[The birth of “Big Code” and how to deliver it with remote ex...](#)

Big Data is big, obviously. But we only use the expression because it is intended to convey the now the mostly-forgotten notion that some...

18 hours ago


F Forbes

[From Big Data To Smart Data: How Manufacturers Can Dri... Value From Industrial Data](#)

Heiko Claussen is SVP of AI at AspenTech, responsible for the company's industry 4.0 strategy, industrial AI research and data science.

21 hours ago


MF menafn

[Big Data Analytics In Energy Market To Reach \\$ 36.76 Bn ...](#)

Big Data Analytics in Energy Market Size The big data analytics in energy market forecast is offered along with information related to ke.

1 day ago

Google Use Case: Web Index

Term	<DocumentIDs, Count>
Big data	<32,10> <45, 7> <59, 3>
....
MapReduce	<12, 50> <3, 20>
....

Wordcount example

Map (*key, value*) \rightarrow {*ikey, ivalue*}

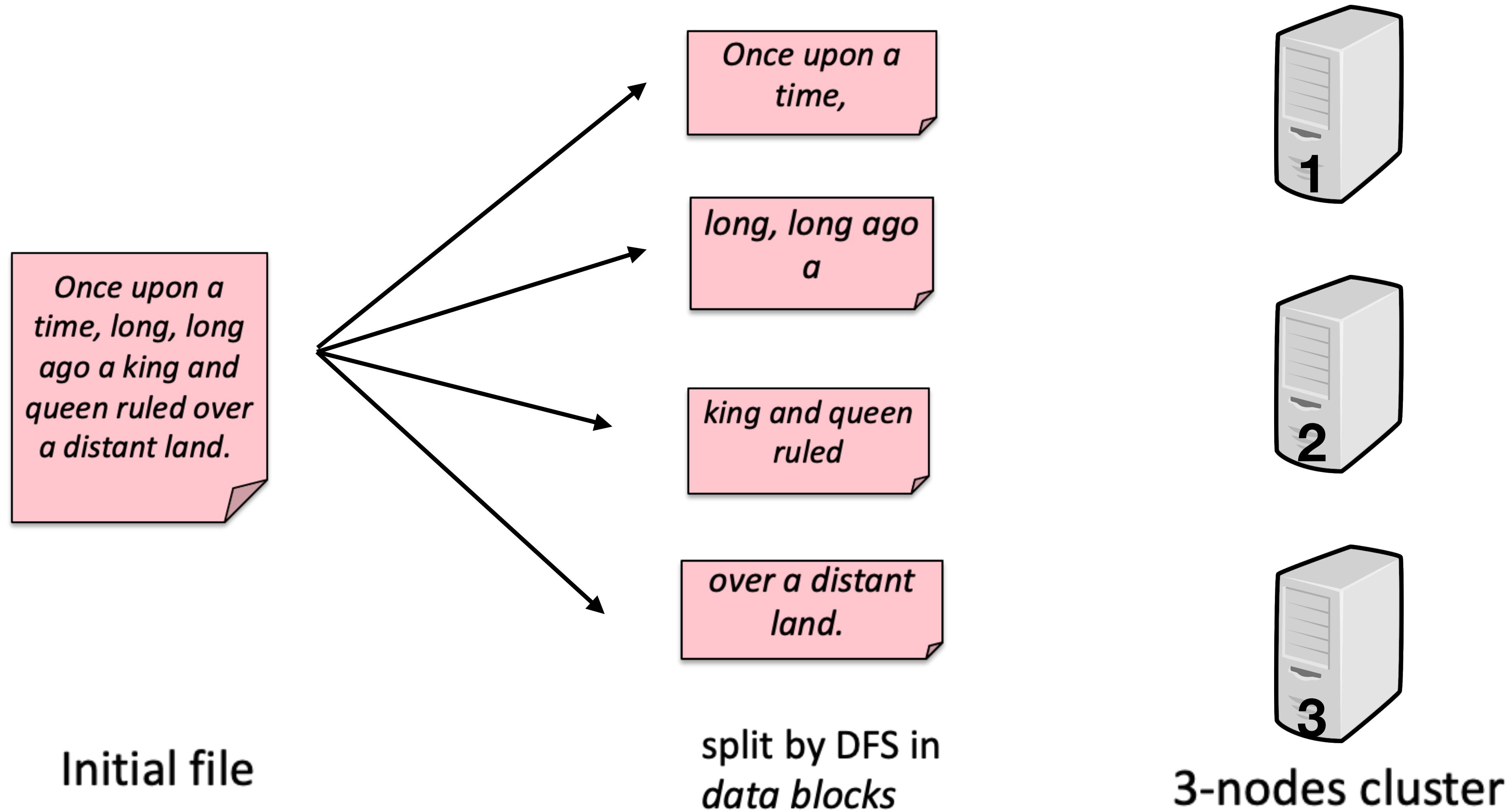
Reduce (*ikey, {ivalue}*) \rightarrow (*key', value'*)



Map (*docID, text*) \rightarrow {*word, 1*}

Reduce (*word, {1, 1, ...}*) \rightarrow (*word, count*)

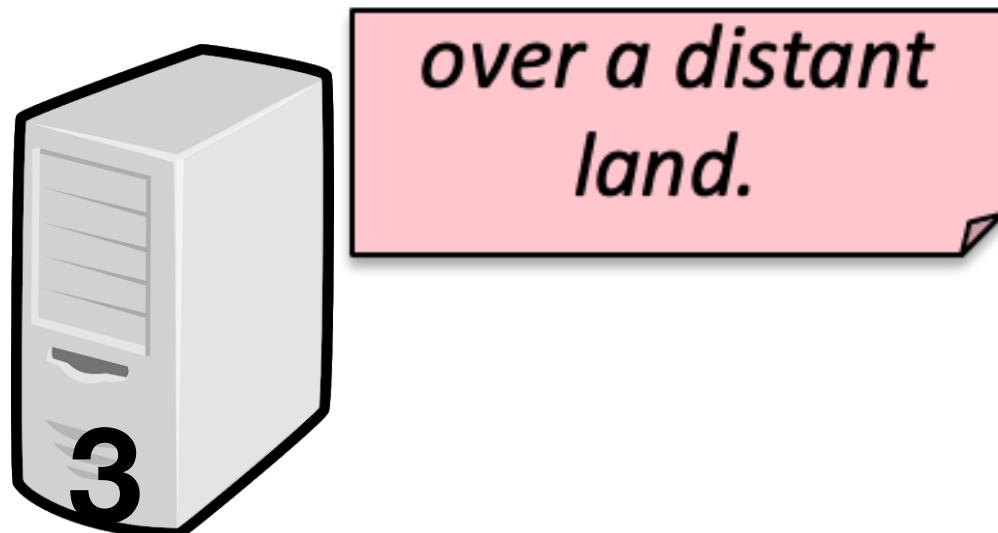
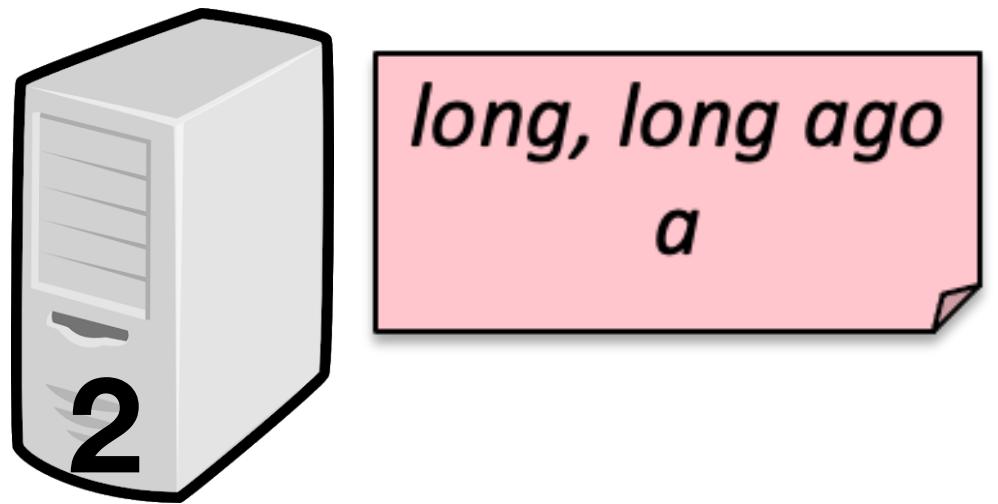
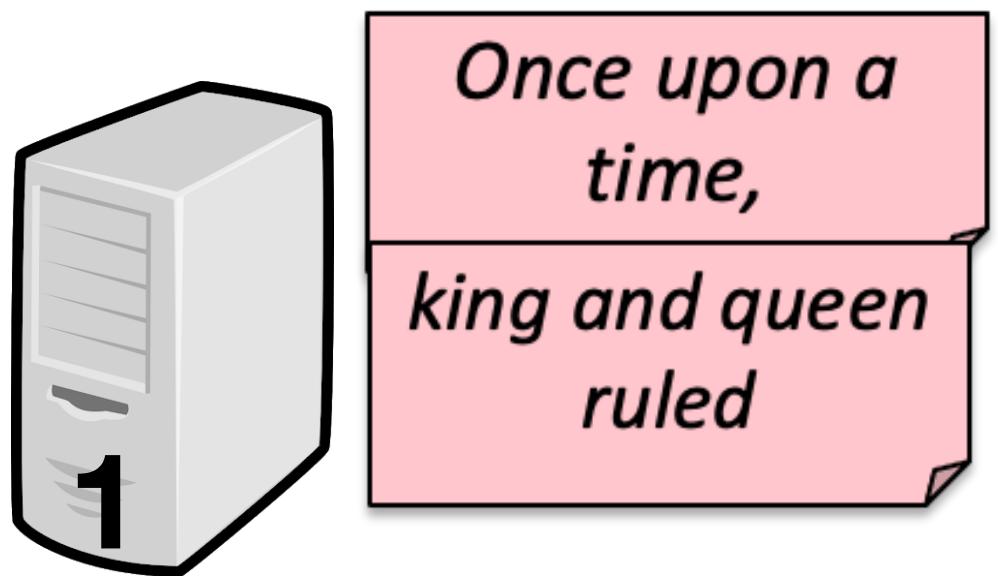
Wordcount example



Wordcount example

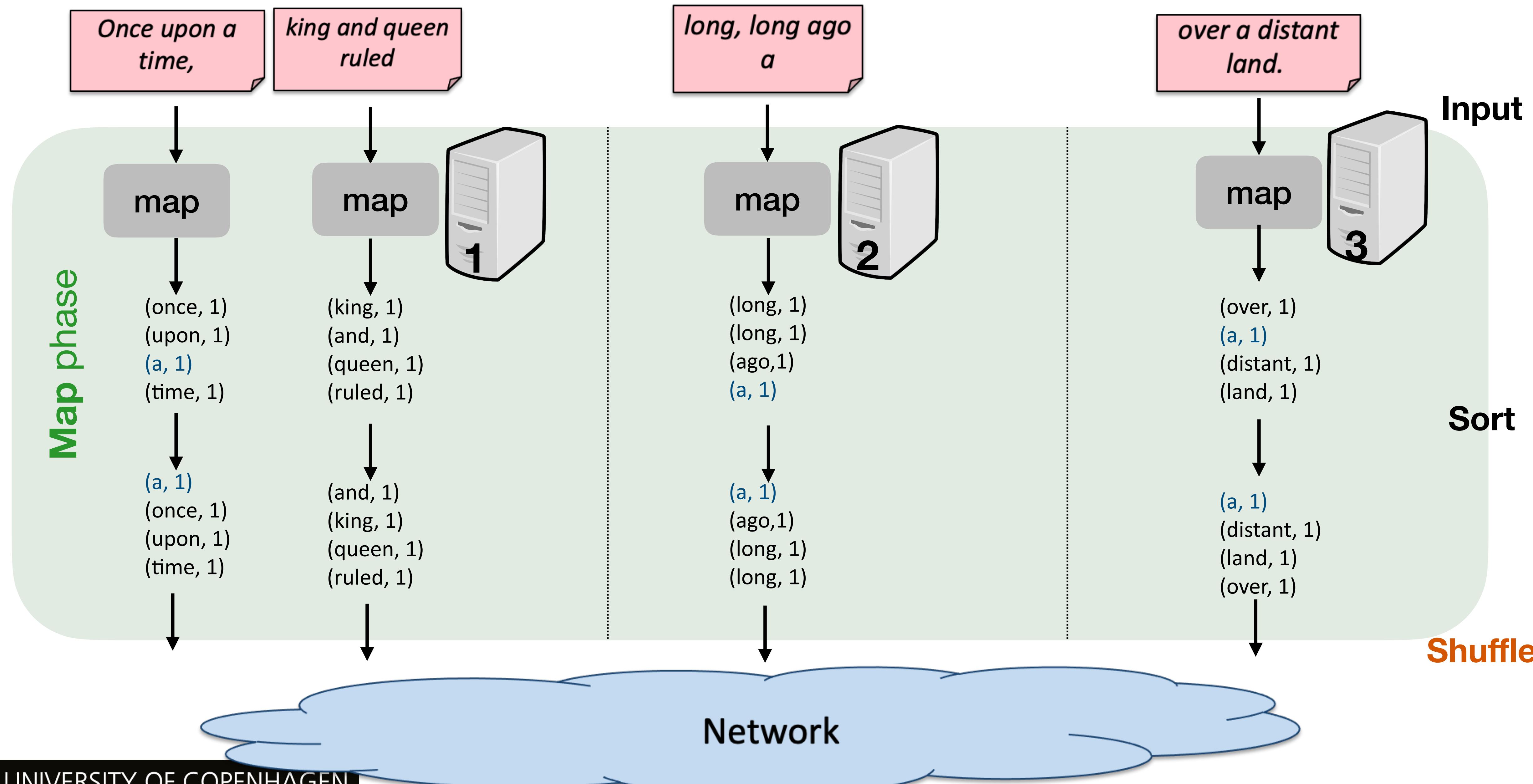
Once upon a time, long, long ago a king and queen ruled over a distant land.

Initial file

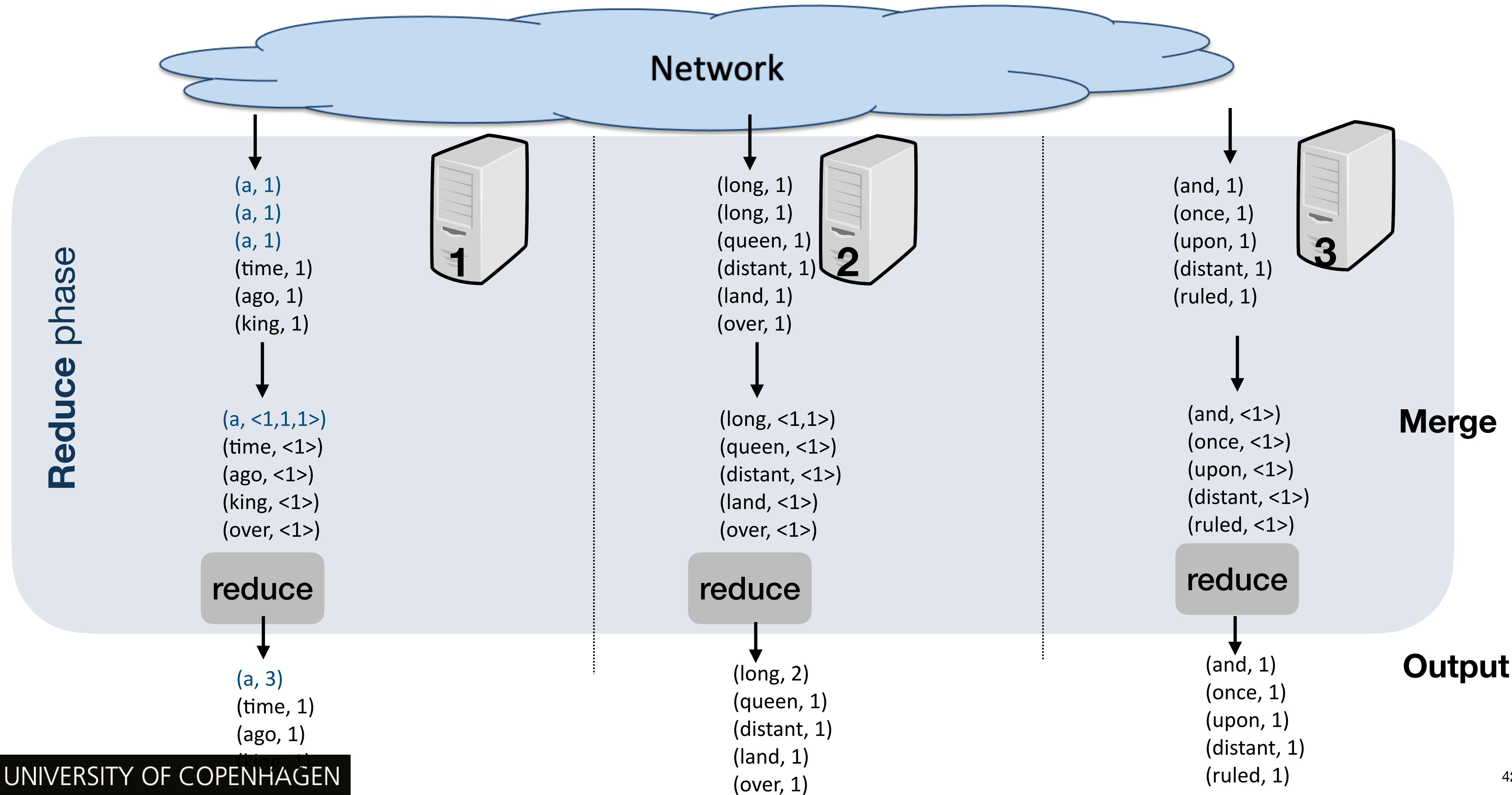


3-nodes cluster

Wordcount example (Map phase)



Wordcount example (Reduce phase)



WordCount example: pseudocode

Map(Integer key, String text)

foreach word w **in** text:

emit (w, 1)

Reduce(String word, Iterator<Integer> counts)

int result = 0

foreach count **in** counts:

 result += count

emit (word, result)

Hadoop/MapReduce Wordcount code :)

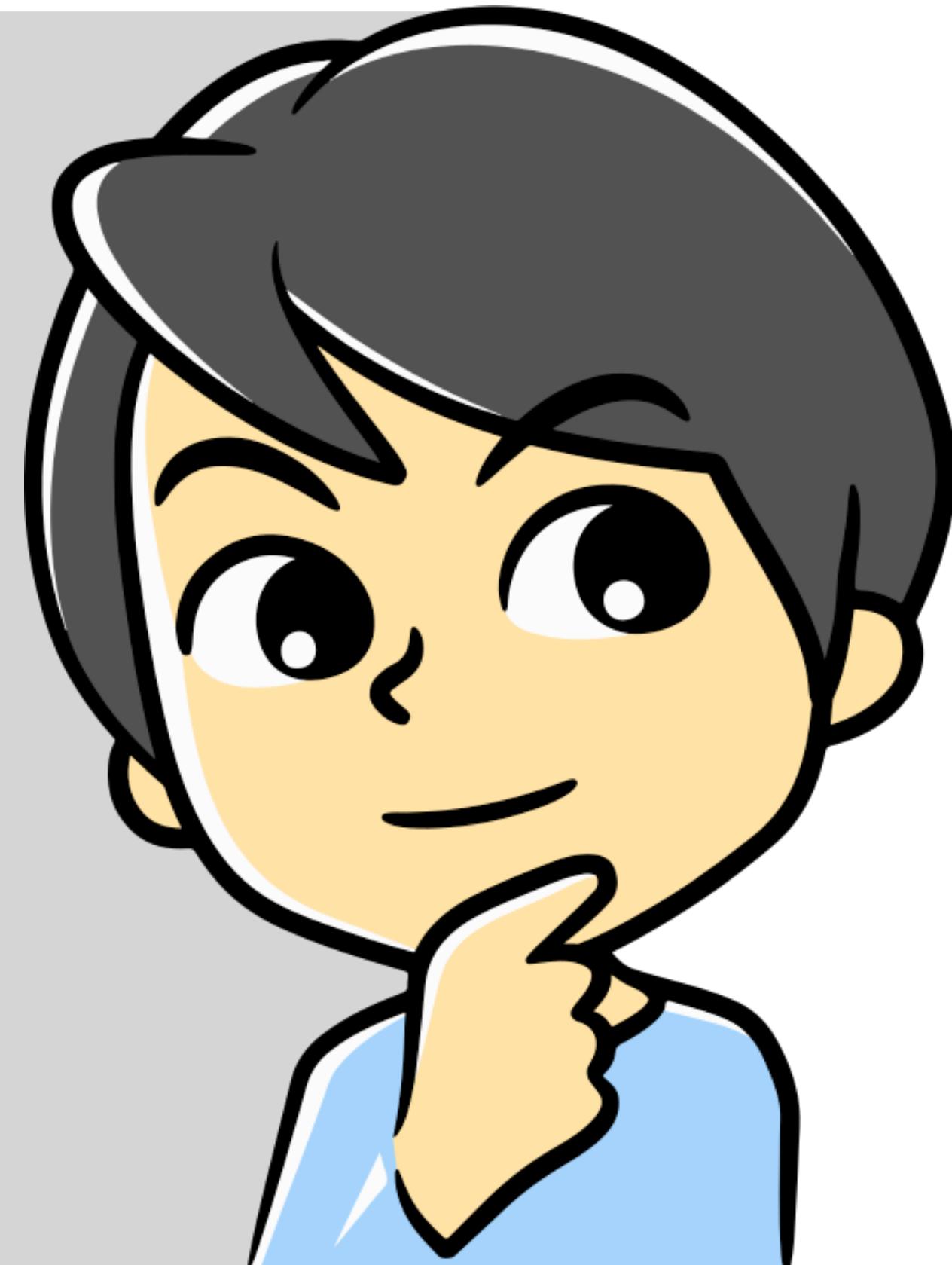
```
1 package org.myorg;
2
3 import java.io.IOException;
4 import java.util.*;
5
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.conf.*;
8 import org.apache.hadoop.io.*;
9 import org.apache.hadoop.mapreduce.*;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
14
15 public class WordCount {
16
17     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
18         private final static IntWritable one = new IntWritable(1);
19         private Text word = new Text();
20
21         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
22             String line = value.toString();
23             StringTokenizer tokenizer = new StringTokenizer(line);
24             while (tokenizer.hasMoreTokens()) {
25                 word.set(tokenizer.nextToken());
26                 context.write(word, one);
27             }
28         }
29     }
30
31     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
32
33         public void reduce(Text key, Iterable<IntWritable> values, Context context)
34             throws IOException, InterruptedException {
35             int sum = 0;
36             for (IntWritable val : values) {
37                 sum += val.get();
38             }
39             context.write(key, new IntWritable(sum));
40         }
41     }
42
43     public static void main(String[] args) throws Exception {
44         Configuration conf = new Configuration();
45
46         Job job = new Job(conf, "wordcount");
47
48         job.setOutputKeyClass(Text.class);
49         job.setOutputValueClass(IntWritable.class);
50
51         job.setMapperClass(Map.class);
52         job.setReducerClass(Reduce.class);
53
54         job.setInputFormatClass(TextInputFormat.class);
55         job.setOutputFormatClass(TextOutputFormat.class);
56
57         FileInputFormat.addInputPath(job, new Path(args[0]));
58         FileOutputFormat.setOutputPath(job, new Path(args[1]));
59
60         job.waitForCompletion(true);
61     }
62
63 }
```



Only 63 lines for a distributed/
parallel application!

Reflection time

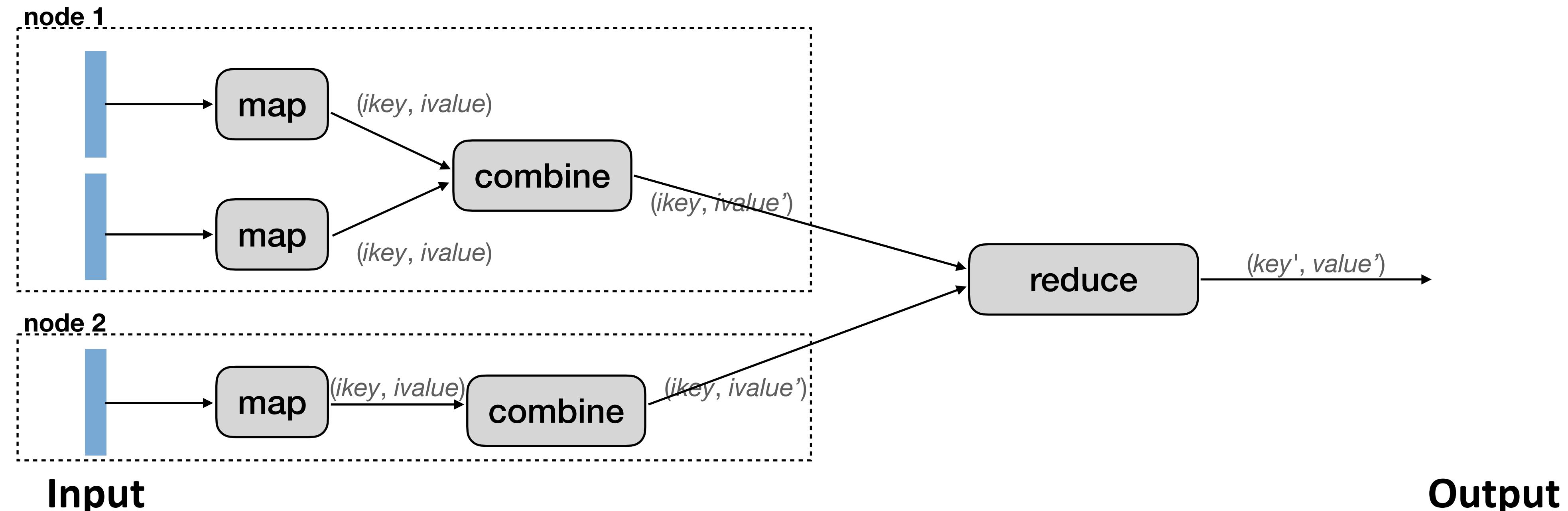
- ♦ MapReduce is powerful enough to run basic SQL queries
 - ♦ Although that's not its original intention
- ♦ Compute the total price of the products for each product type available in store 10.
 - ♦ prodId, prodName, prodType, price, storeId
 - ♦ select prodType, sum(price)
from products
where storeId = 10
groupby prodType
- ♦ How you would implement it in MapReduce?



Combiner

Local reducer

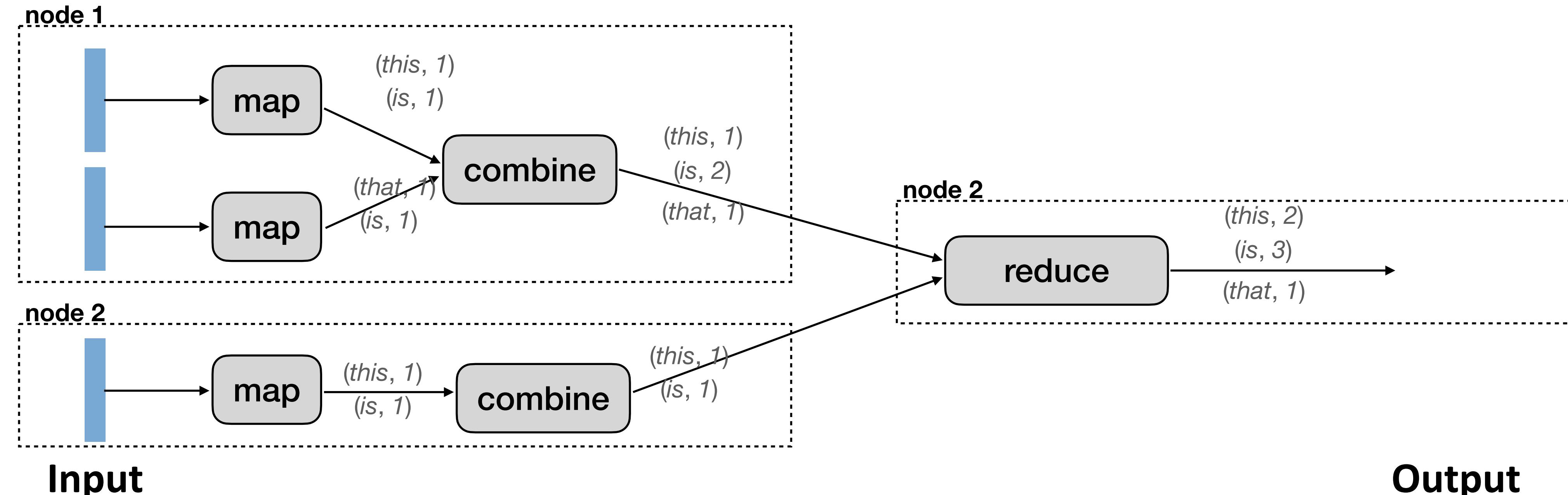
- ◆ Pre-aggregate in Map phase
 - ◆ Reduces network bandwidth
 - ◆ Reduces work on Reduce phase
 - ◆ Reduce functions needs to be associative and commutative



Combiner

Local reducer

- ◆ Pre-aggregate in Map phase
 - ◆ Reduces network bandwidth
 - ◆ Reduces work on Reduce phase
 - ◆ Reduce functions needs to be associative and commutative



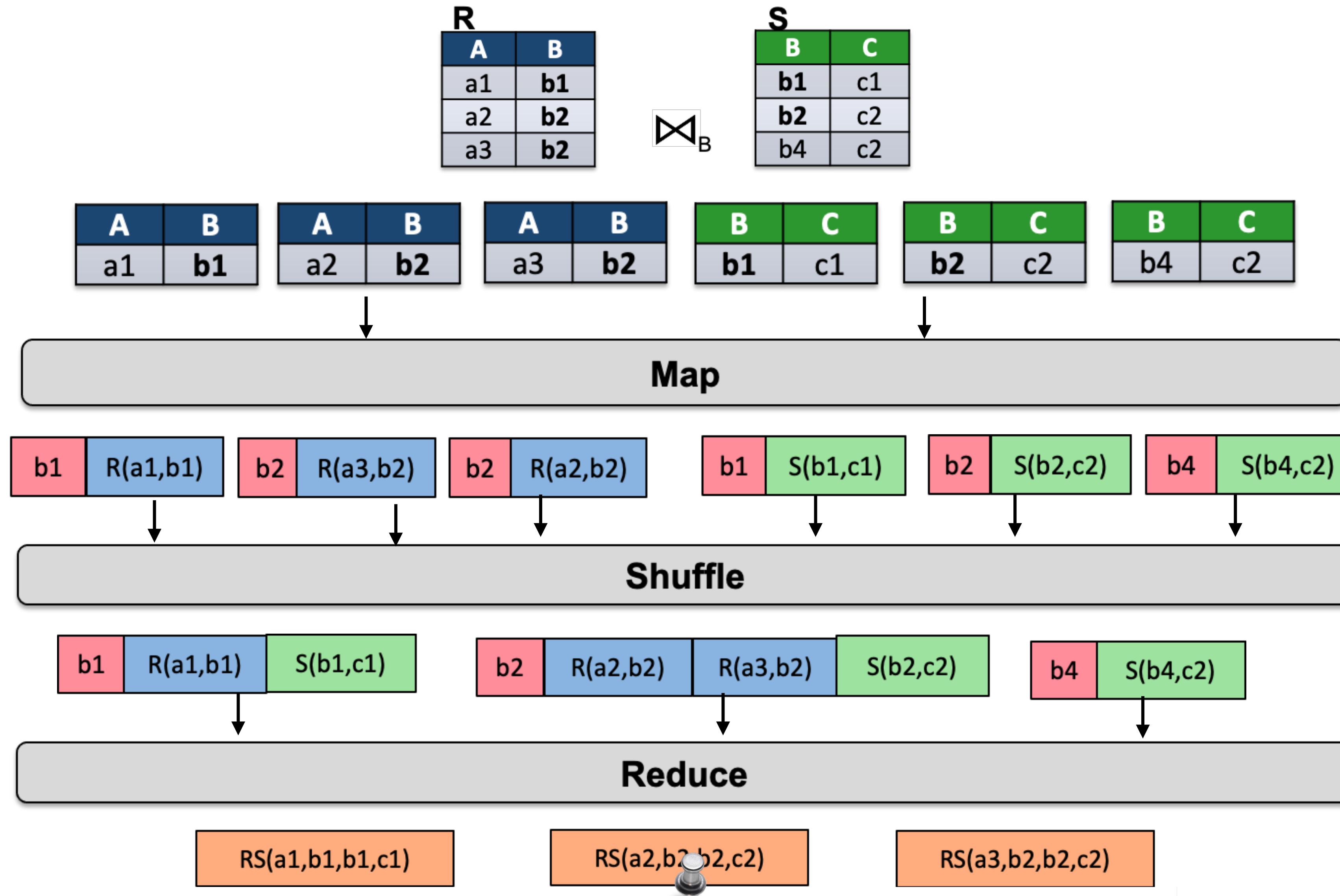
SQL in MapReduce

- ♦ MapReduce is powerful enough to run basic SQL queries
 - ♦ Although that's not its original intention
- ♦ Easy case: single table, no sorting
 - ♦ Map: selection, projection w/o duplicate elimination
 - ♦ MapReduce framework: grouping
 - ♦ Reduce: aggregation, having, duplicate elimination

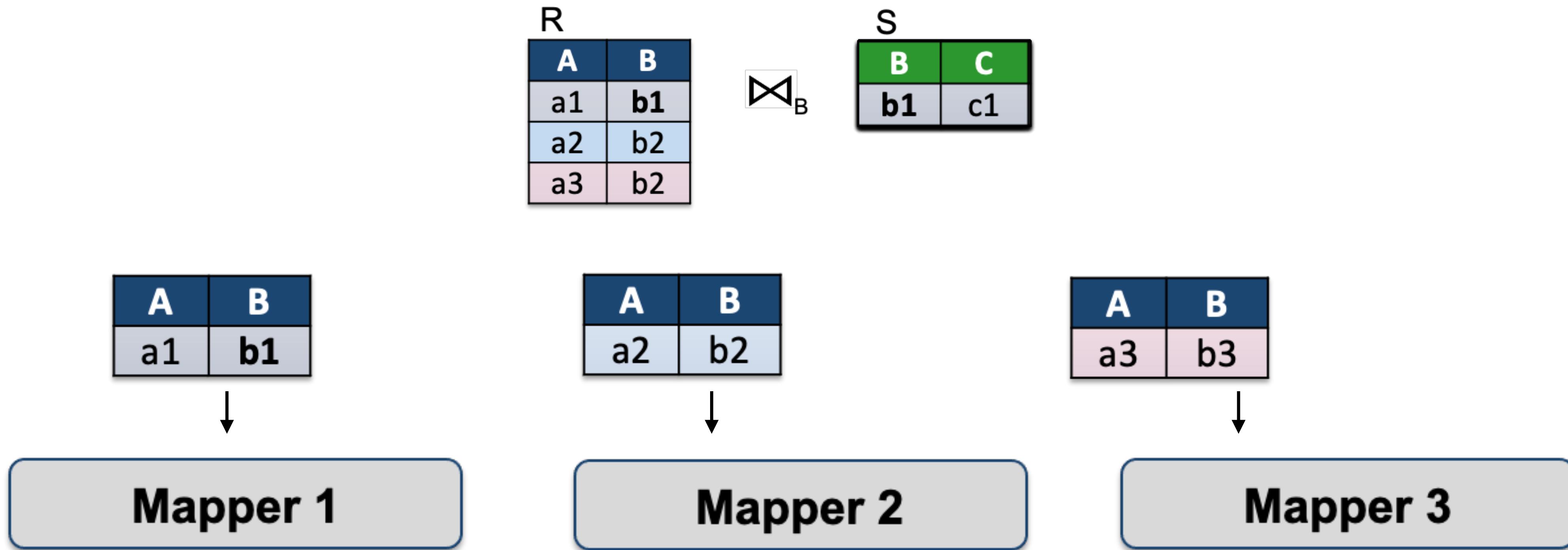
Joins in MapReduce

- ❖ Two main strategies:
 - ❖ **symmetric hash join (repartition join)**
 - ❖ **replicated join (broadcast join)**

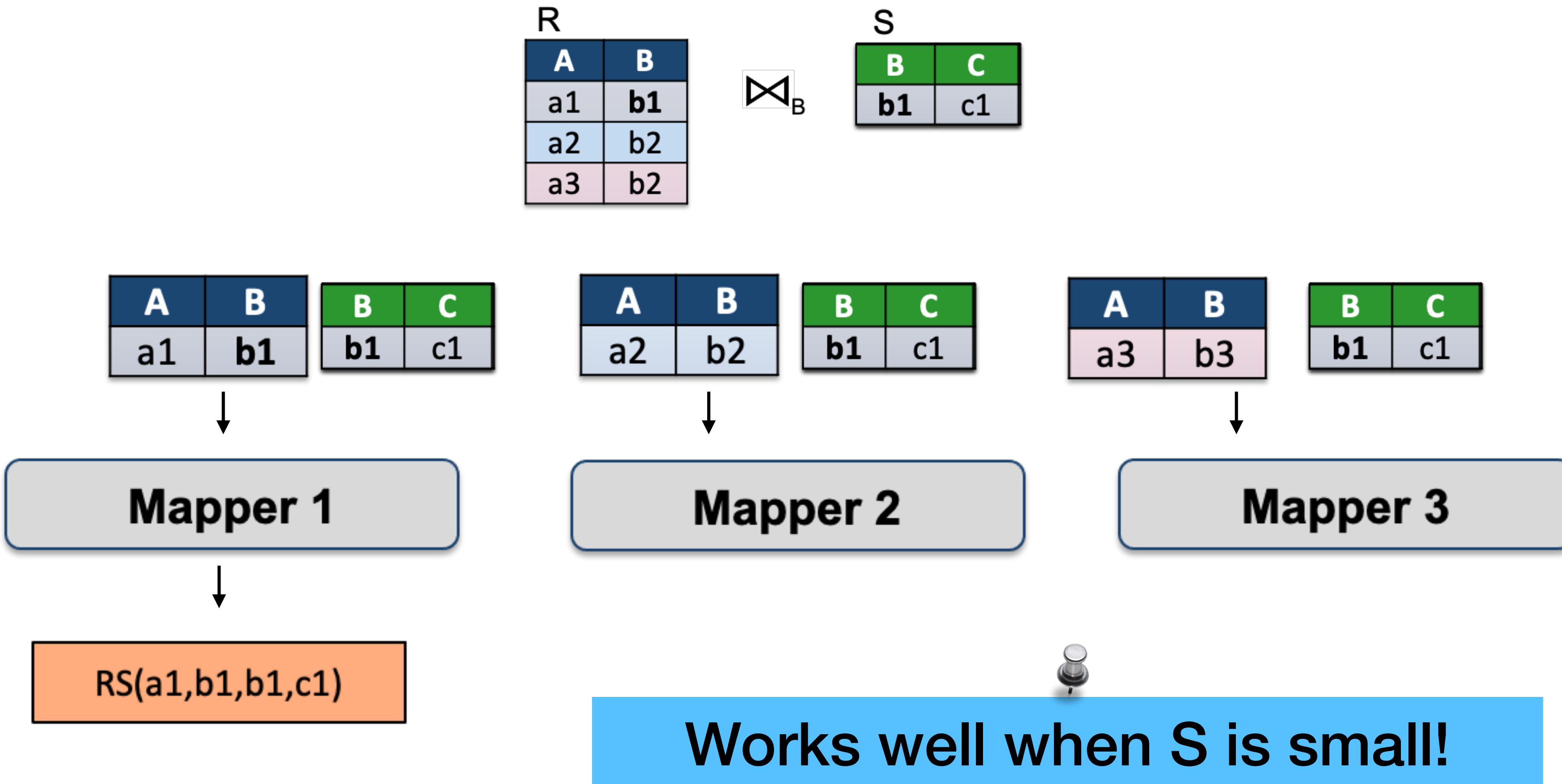
Repartition join



Broadcast join



Broadcast join



Today's lecture

- ◆ Distributed file systems (HDFS)
- ◆ The MapReduce distributed data processing paradigm
- ◆ **Hadoop/MapReduce under the hood**



Parallelization in MapReduce

- ♦ Application of user-defined function (UDF) f in `map()` on a data item often not influenced by computations on other data items
- ♦ **Embarassingly parallel**
- ♦ Implies that we can re-order or parallelize the execution
- ♦ Note: not true when f has side-effects or state



This property forms the basis for MapReduce!

Hadoop/MapReduce under the hood

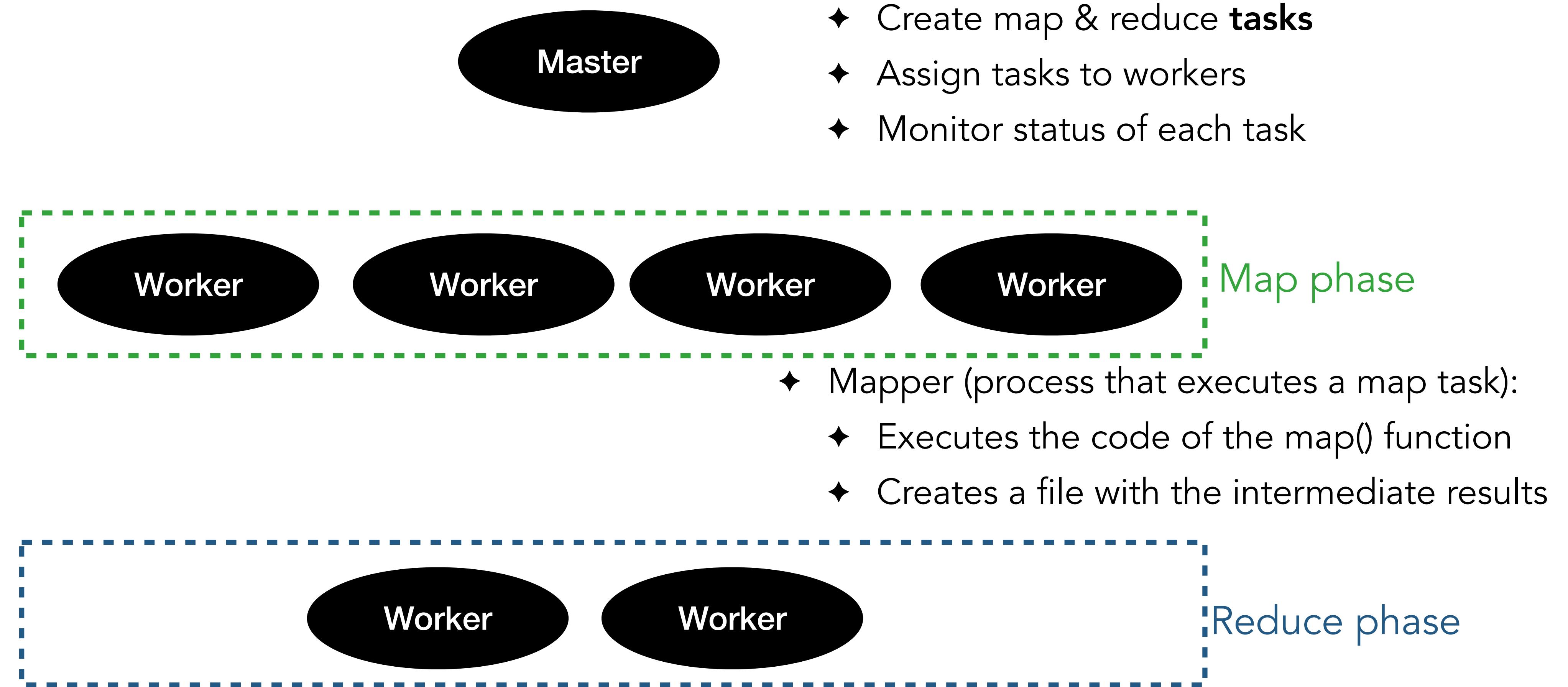
Once upon a time,

long, long ago a

king and queen ruled

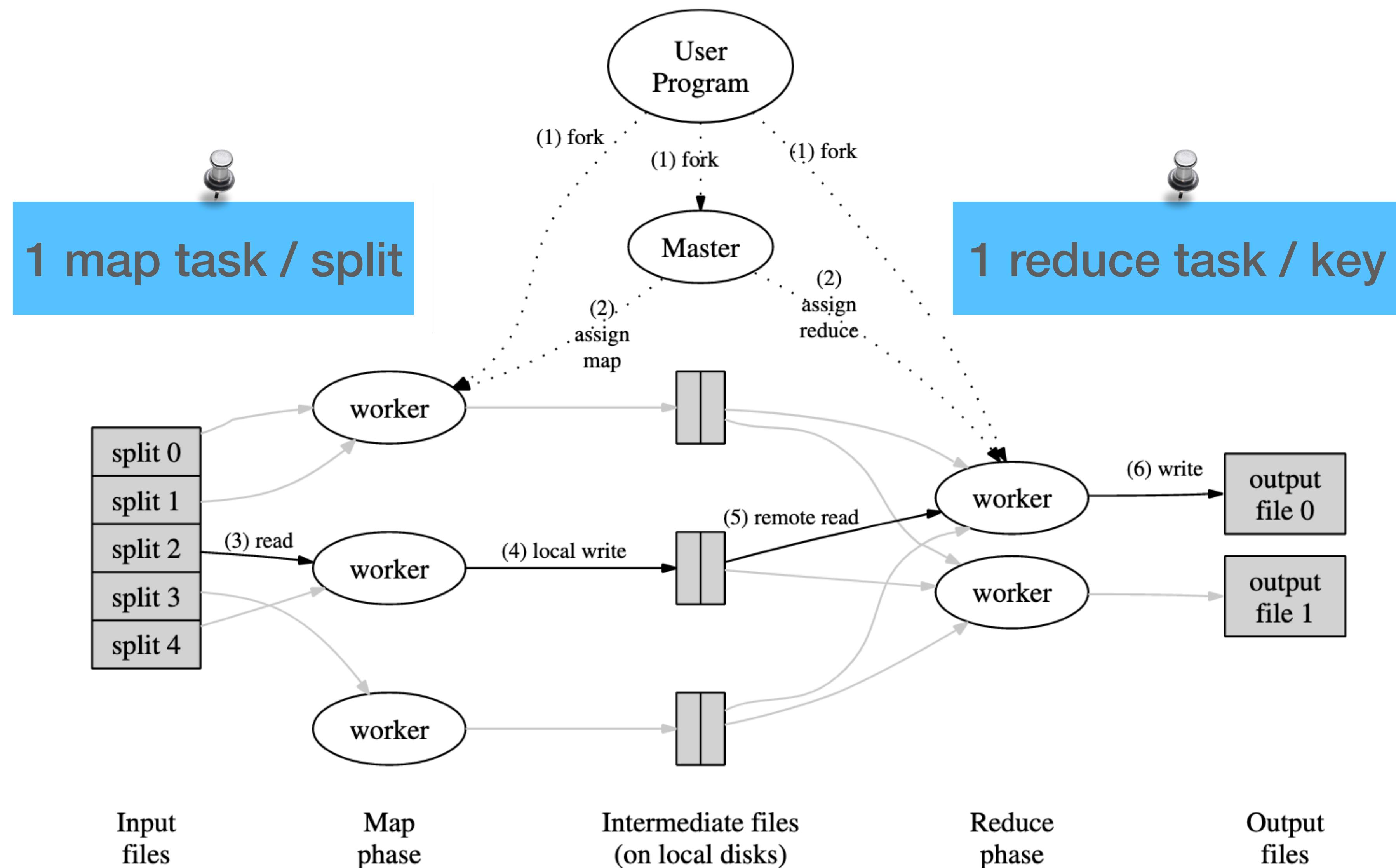
over a distant land.

Input file splits



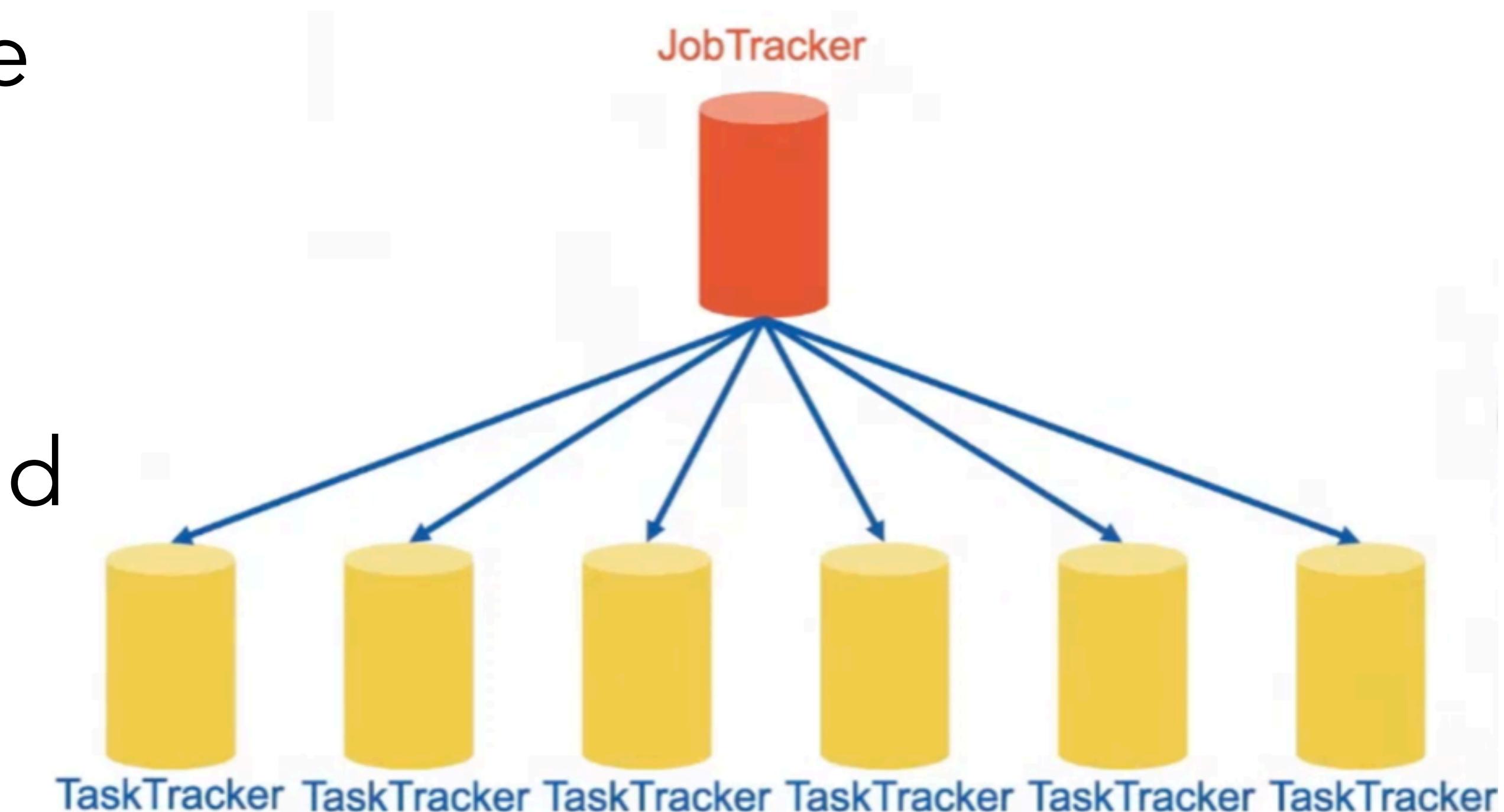
- ◆ Master:
 - ◆ Create map & reduce **tasks**
 - ◆ Assign tasks to workers
 - ◆ Monitor status of each task
- ◆ Mapper (process that executes a map task):
 - ◆ Executes the code of the map() function
 - ◆ Creates a file with the intermediate results
- ◆ Reducer (process that executes a reduce task):
 - ◆ Executes the code of the reduce() function
 - ◆ Writes output in DFS

Google MapReduce execution overview



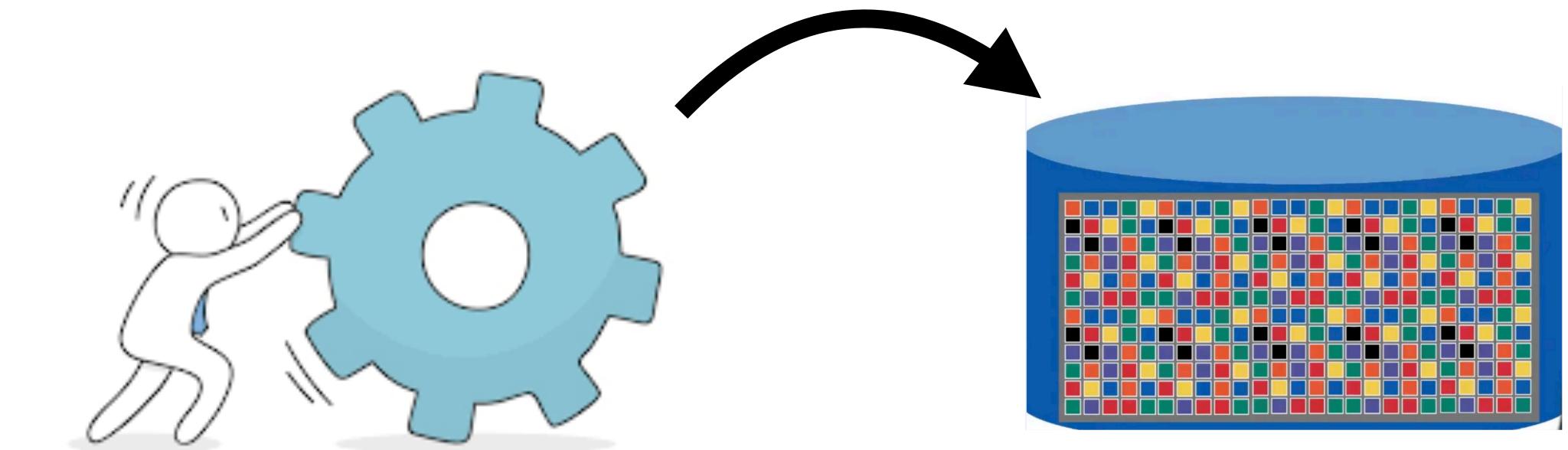
Hadoop architecture

- ◆ **Jobtracker:** client communication, job scheduling, resource management, lifecycle coordination
- ◆ **Tasktracker:** task execution and management



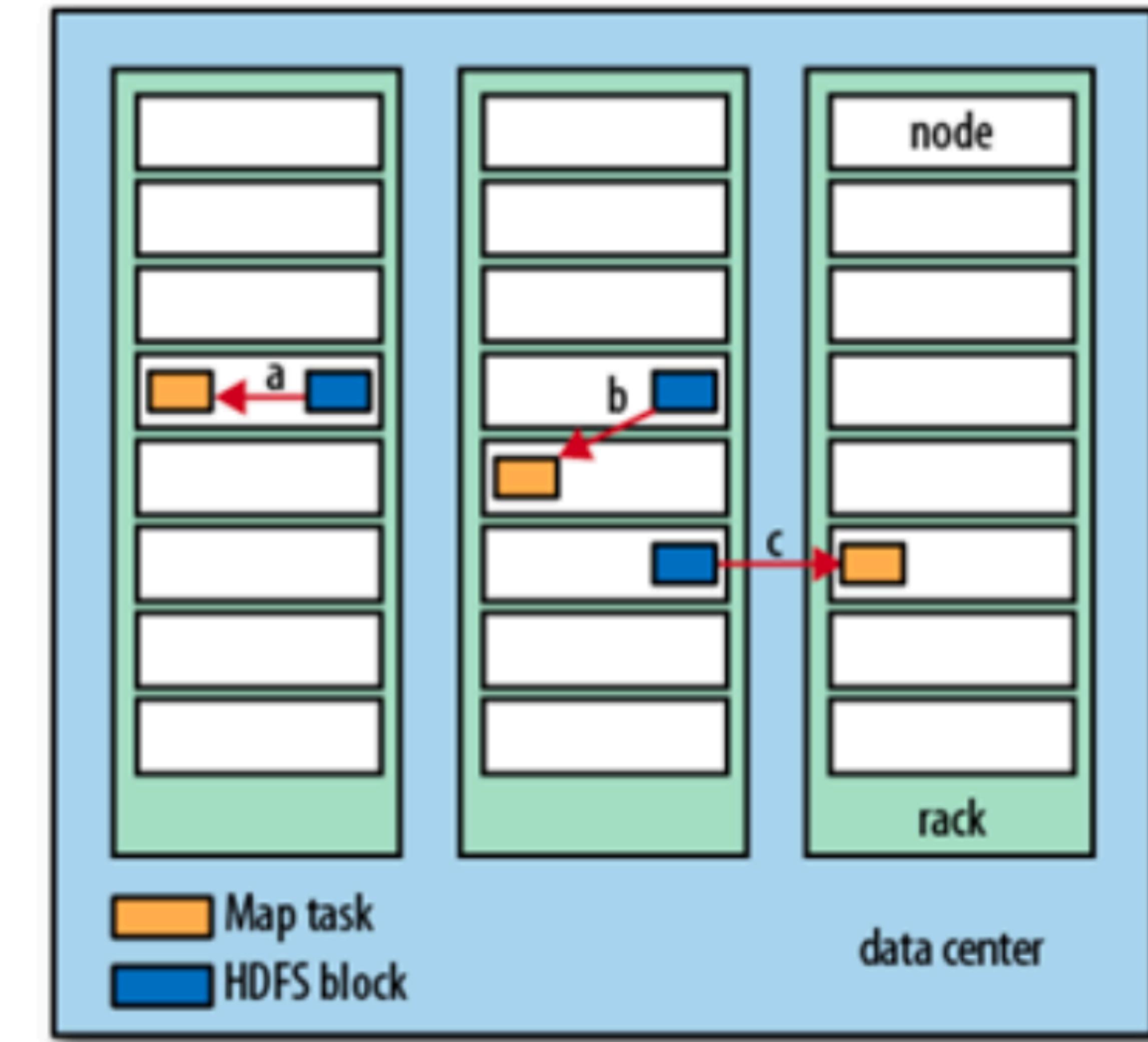
Data locality

- ♦ Move **computation to data**
 - ♦ Small code, large data
 - ♦ Reduces network bandwidth
- ♦ Goal: run map task on machine that stores input chunk
 - ♦ Then map task reads data locally (**local fetch**)
 - ♦ Thousands of machines can read at local disk speed; read rate not limited by network

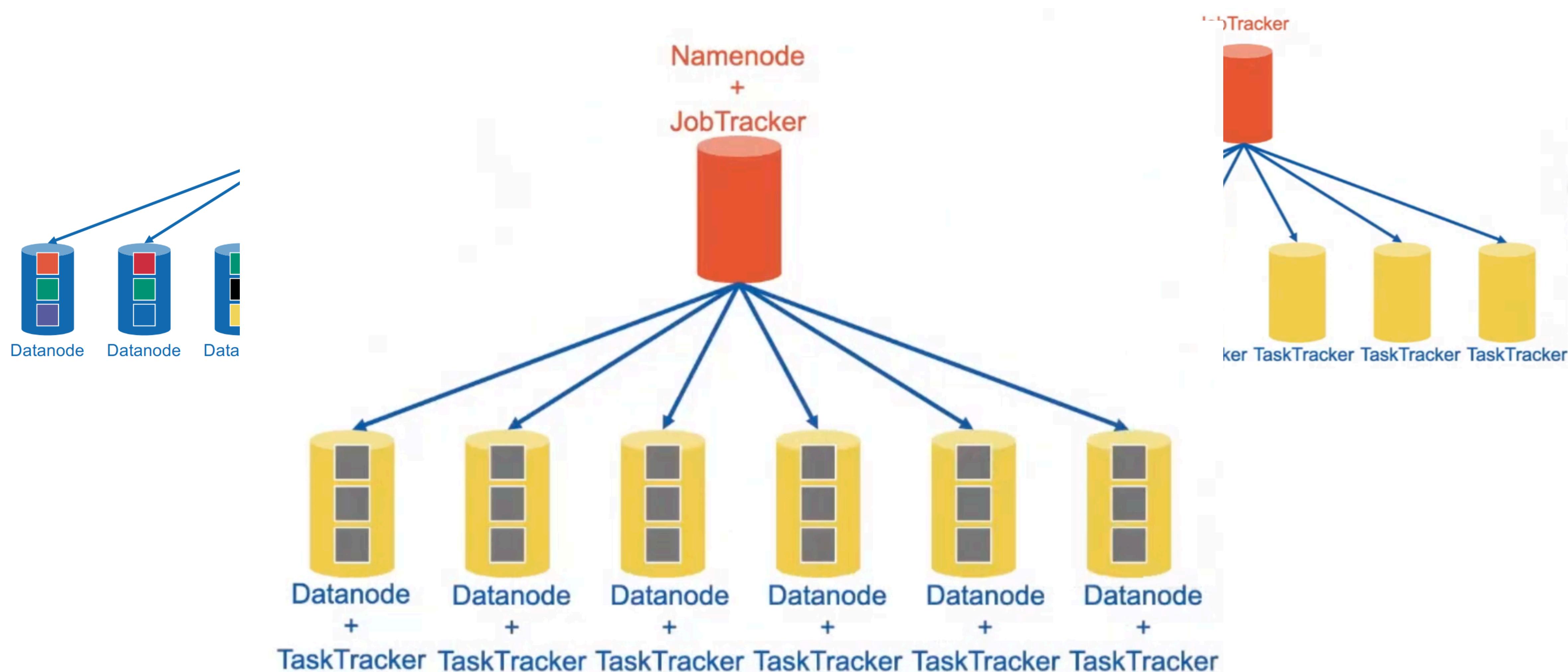


Different types of map tasks based on locality

- ◆ (a) Local map task
- ◆ (b) rack-local map task
- ◆ (c) off-rack map task



Hadoop infrastructure



Hadoop/MapReduce Summary

- ◆ **Distributed File System:**
 - ◆ Large files split into fixed-size (large) chunks
 - ◆ Chunks replicated 3 times
- ◆ **MapReduce:** programming model for distributed computation
 - ◆ Users code map() and reduce() functions
- ◆ **Hadoop:** Open-source implementation of MapReduce
 - ◆ Parallelizes map and reduce functions
 - ◆ Handles nodes failures

In the next lecture ...

- ♦ Dataflow engines paradigm
- ♦ Introduction to Apache Spark



Readings

- ♦ Chapter 2 from “Mining of Massive Datasets” book
- ♦ GFS paper:
 - ♦ Sanjay Ghemawat et al.: The Google file system. SOSP 2003: 29-43 [[pdf](#)]
- ♦ MapReduce paper:
 - ♦ Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150 [[pdf](#)]