ВВЕДЕНИЕ В HADOOP …

Вовченко Алексей,

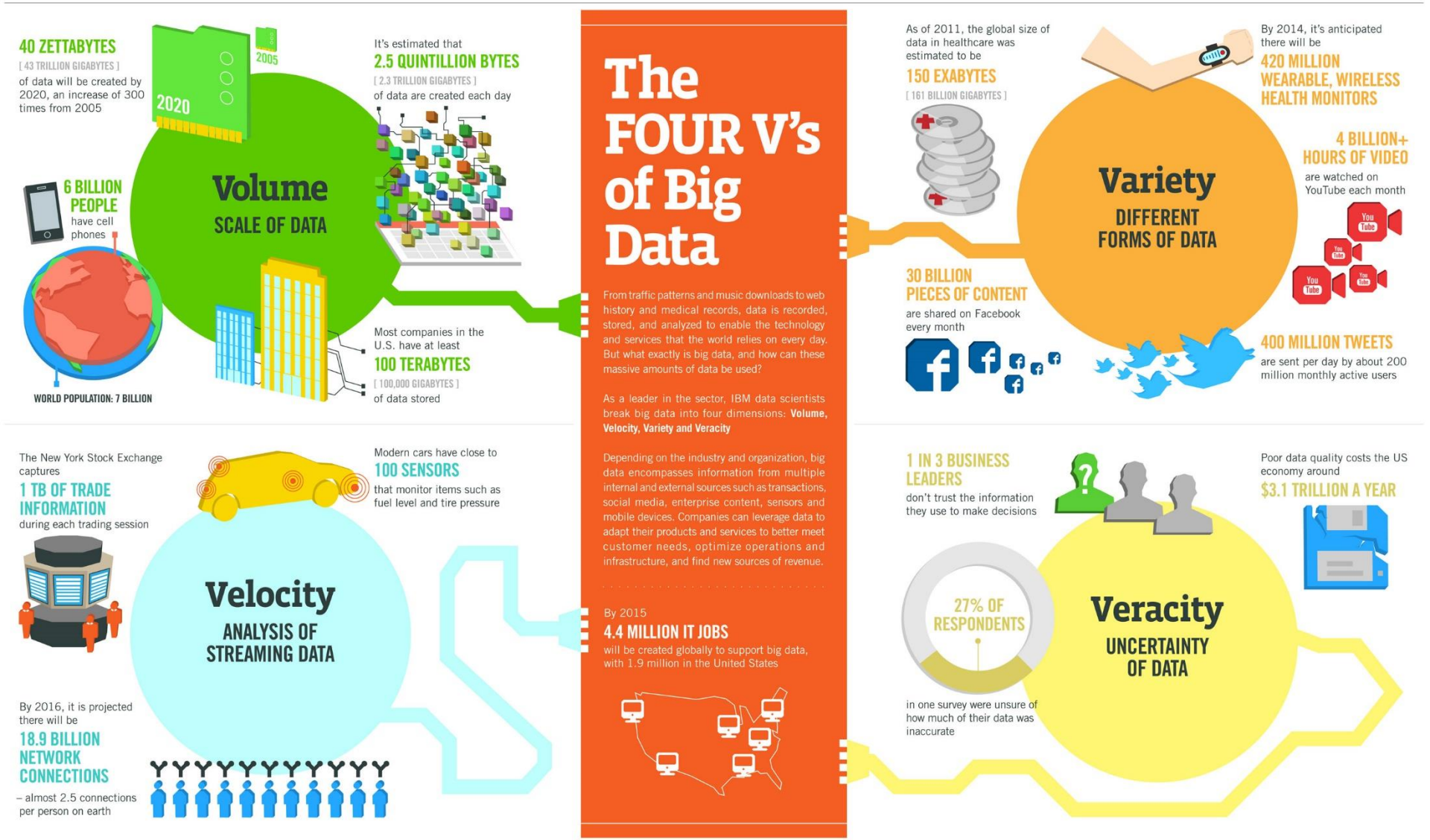
alexey.vovchenko@gmail.com

к.т.н., с.н.с.

ВМК МГУ, ИПИ РАН

ЧЕМ ХАРАКТЕРИЗУЮТСЯ БОЛЬШИЕ ДАННЫЕ?

THE FOUR V'S OF BIG DATA



Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

ЧЕМ ПЛОХИ ТРАДИЦИОННЫЕ АРХИТЕКТУРЫ (СУБД)?

ТРАДИЦИОННЫЕ АРХИТЕКТУРЫ

- Архитектура СУБД:
  - Дисковое хранение
  - Транзакции
  - Индексация
  - Журнализация
  - Многопоточность
  - Блокировки
- Производительность самых мощных серверов растет медленнее чем объемы данных

LARGE-SCALE DATA ANALYTICS

- MapReduce computing paradigm (E.g., Hadoop) vs. Traditional database systems

vs.

- **Many enterprises are turning to Hadoop**
  - Especially applications generating ***big data***
  - Web applications, social networks, scientific applications

BIG DATA

Scalability (petabytes of data, thousands of machines)

Flexibility in accepting all data formats (no schema)

Efficient and simple fault-tolerant mechanism

Commodity inexpensive hardware

vs.

ТРАДИЦИОННЫЕ АРХИТЕКТУРЫ

Performance (tons of indexing, tuning, data organization tech.)

Features:

- Provenance tracking
- Annotation management
- ....

ЧТО ВКЛЮЧАЕТ ПЛАТФОРМА БОЛЬШИХ ДАННЫХ?

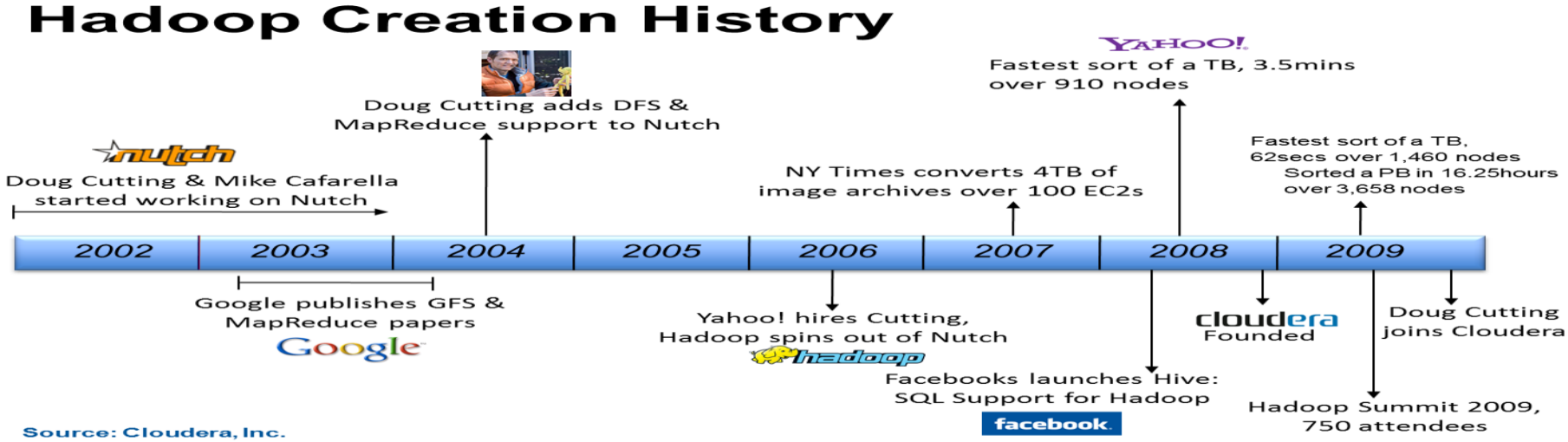
СРЕДСТВА ОБНАРУЖЕНИЯ НОВОЙ ИНФОРМАЦИИ

ПЛАН

- Введение в Hadoop
- Архитектура Hadoop
- Парадигма map-reduce
- Архитектура стека компонентов Hadoop
- Когда стоит применять Hadoop?

ИСТОРИЯ (© WIKIPEDIA)

- В 2005 году Дугом Каттингом для построения программной инфраструктуры распределённых вычислений для проекта Nutch — свободной программной поисковой машины на Java (на основе публикации сотрудников Google о вычислительной концепции MapReduce[3]) Проект был назван в честь игрушечного слонёнка ребёнка основателя
- В течение 2005—2006 годов Hadoop развивался усилиями двух разработчиков — Каттинга и Майка Кафарелы (*Mike Cafarella*) в режиме частичной занятости[3], сначала в рамках проекта Nutch, затем — проекта Lucene.
- В январе 2008 года Hadoop становится проектом верхнего уровня системы проектов Apache Software Foundation. В апреле 2008 года Hadoop побил мировой рекорд производительности — 1 ТБайт был обработан за 209 сек. на кластере из 910 узлов[5]. С этого момента начинается широкое применение Hadoop за пределами Yahoo — технологию для своих сайтов внедряют Last.fm,Facebook, The New York Times[6]…
- В сентябре 2009 года Каттинг переходит в калифорнийский стартап Cloudera, который, благодаря такому переходу, комментаторы характеризуют как «нового знаменосца Hadoop»
- В апреле 2010 года корпорация Google предоставила Apache Software Foundation права на использование технологии MapReduce, через три месяца после её защиты в патентном бюро США, тем самым избавив организацию от возможных патентных претензий[8].

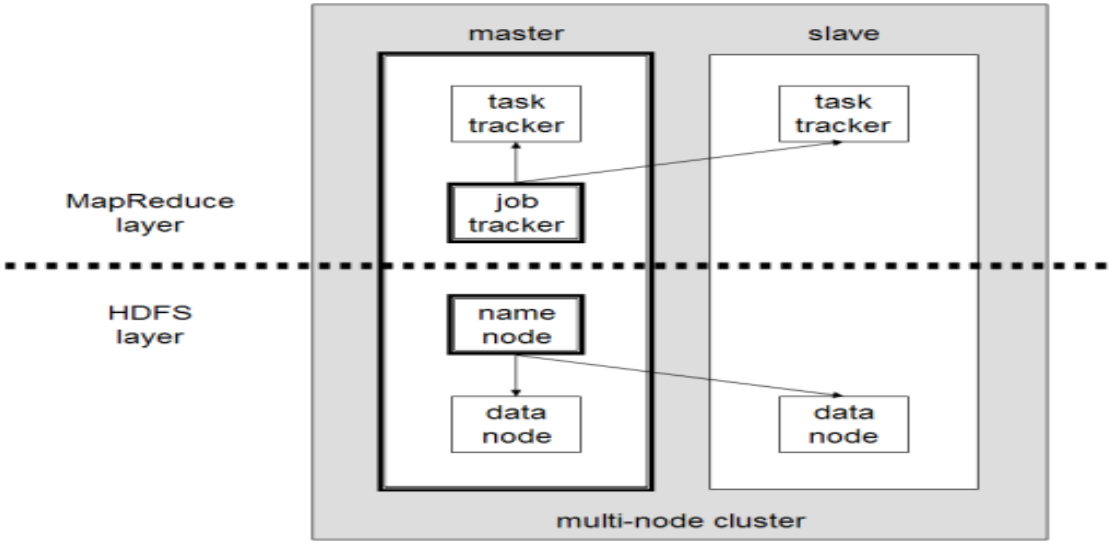


ПЛАН

- Введение в Hadoop
- **Архитектура Hadoop**
- Парадигма map-reduce
- Архитектура стека компонентов Hadoop
- Когда стоит применять Hadoop?

WHAT IS HADOOP

- **Hadoop framework consists on two main layers**
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



What is Hadoop?

The Apache Software Foundation

- **Open source project**
- **Written in Java**
- **Optimized to handle**
  - Massive amounts of data through parallelism
  - A variety of data (structured, unstructured, semi-structured)
  - Using inexpensive commodity hardware
- **Great performance**
- **Reliability provided through replication**
- **Not for OLTP, not for OLAP/DSS, good for Big Data**
- **Current version: 0.20.2**

Hadoop-related open source projects

PIG

jaql

ZooKeeper

Hadoop is not for all types of work

- Not to process transactions (random access)
- Not good when work cannot be parallelized
- Not good for low latency data access
- Not good for processing lots of small files
- Not good for intensive calculations with little data

Terminology review

Node 1

Terminology review

Node 1

Node 2

Terminology review

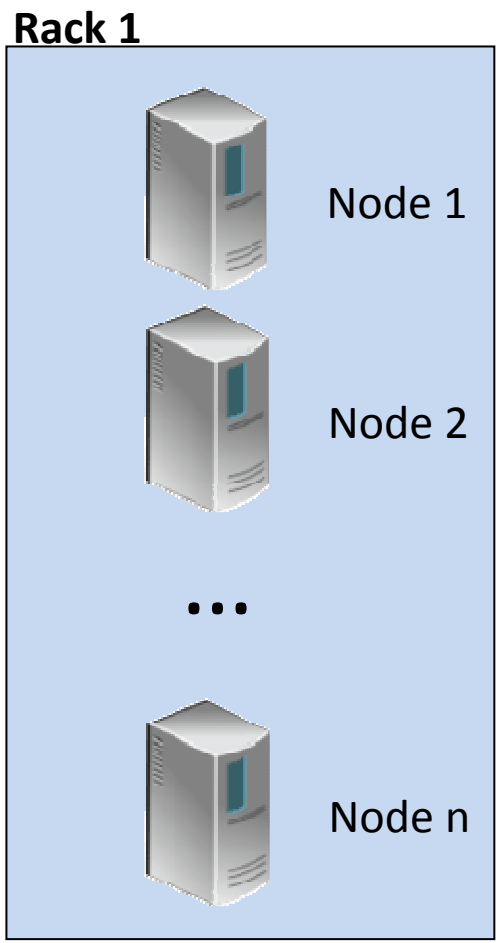
Node 1

Node 2

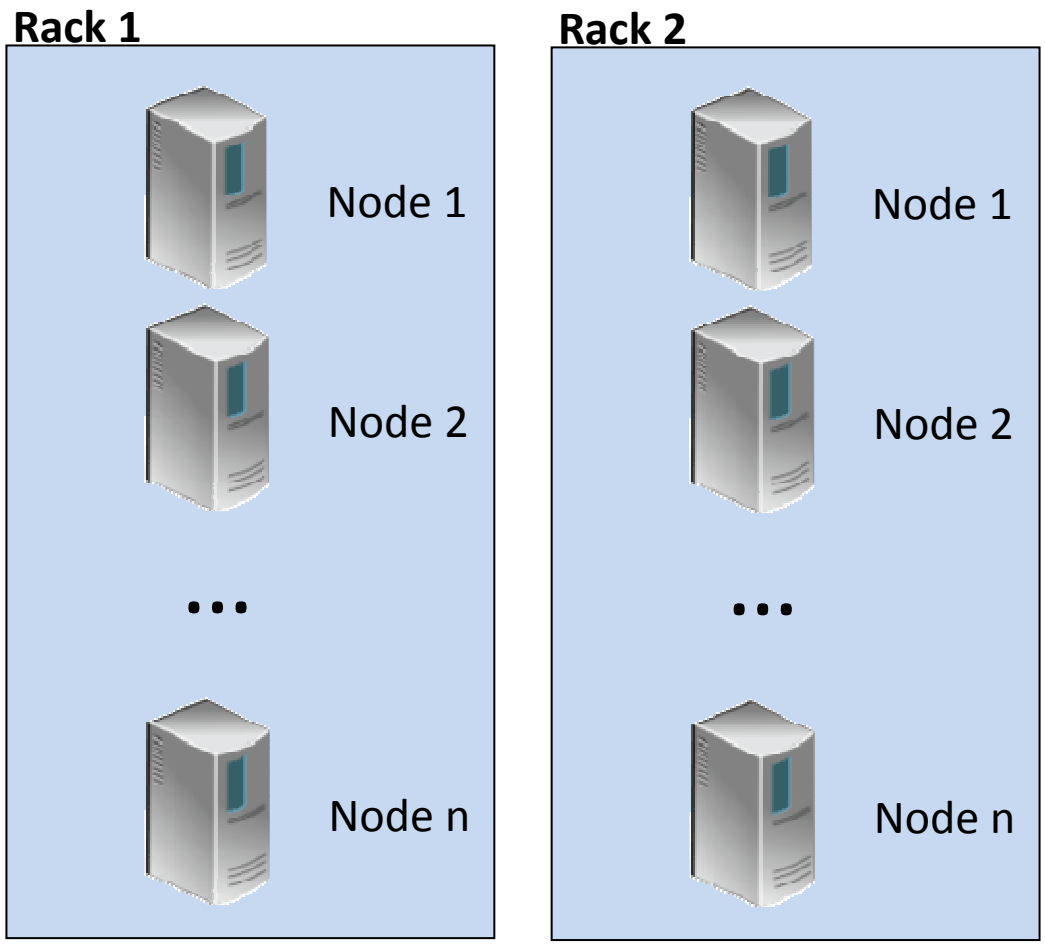
...

Node n

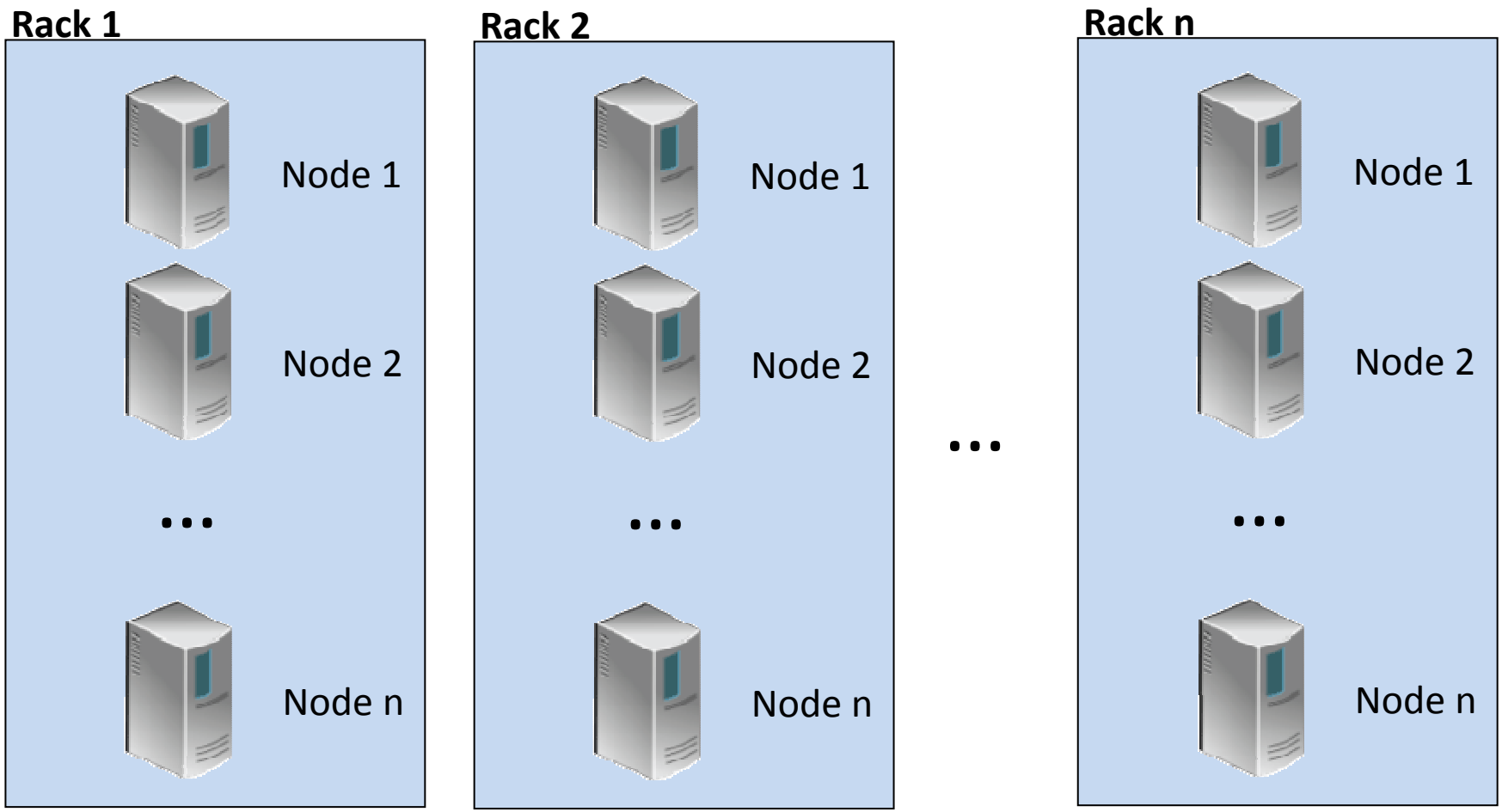
Terminology review



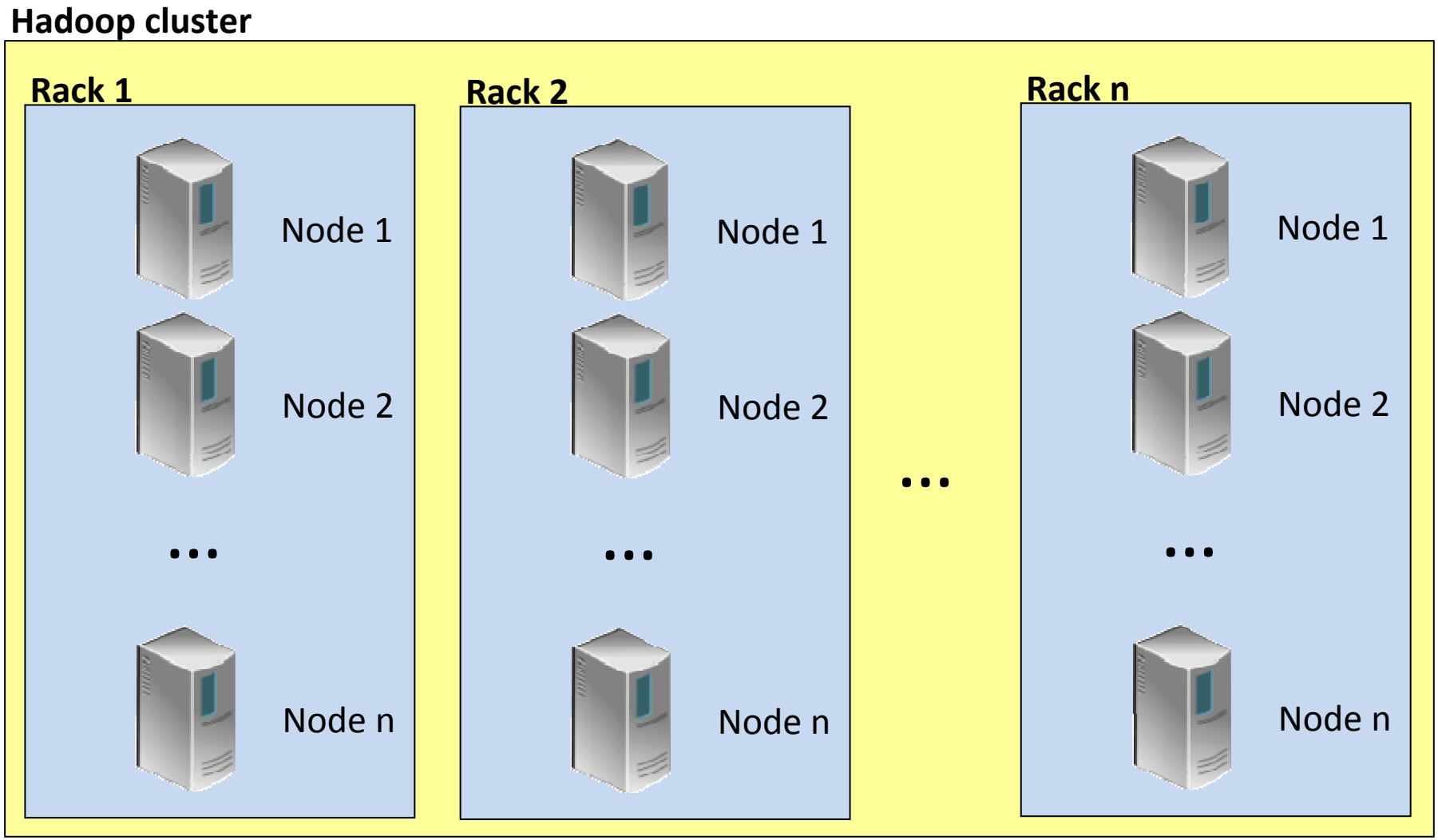
Terminology review



Terminology review



Terminology review

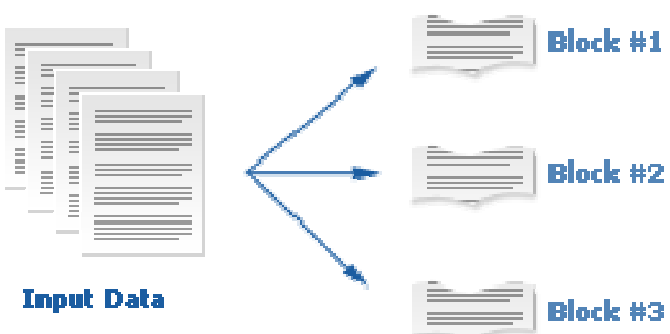


Hadoop architecture

- Two main components:
  - Hadoop Distributed File System (HDFS)
  - MapReduce Engine

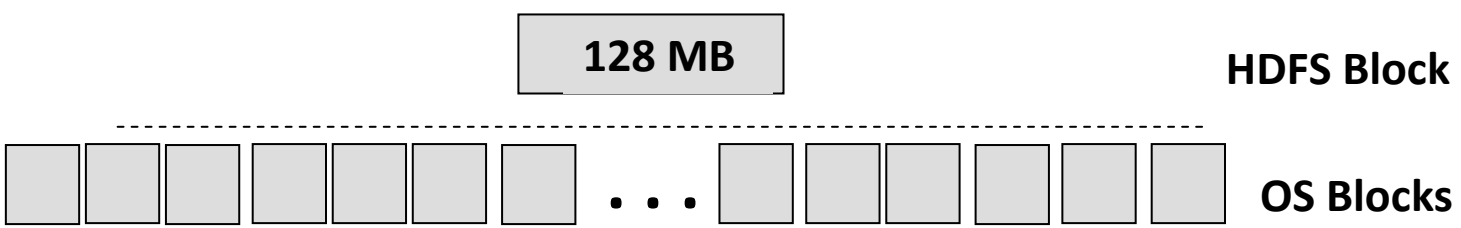
Hadoop distributed file system (HDFS)

- Hadoop file system that runs on top of existing file system
- Designed to handle very large files with streaming data access patterns
- Uses blocks to store a file or parts of a file



HDFS - Blocks

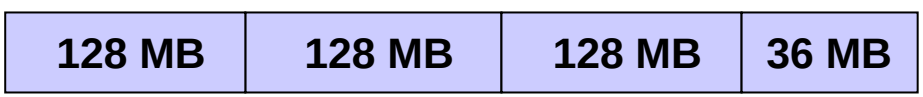
- File Blocks
  - 64MB (default), 128MB (recommended) – compare to 4KB in UNIX
  - Behind the scenes, 1 HDFS block is supported by multiple operating system (OS) blocks



HDFS - Blocks

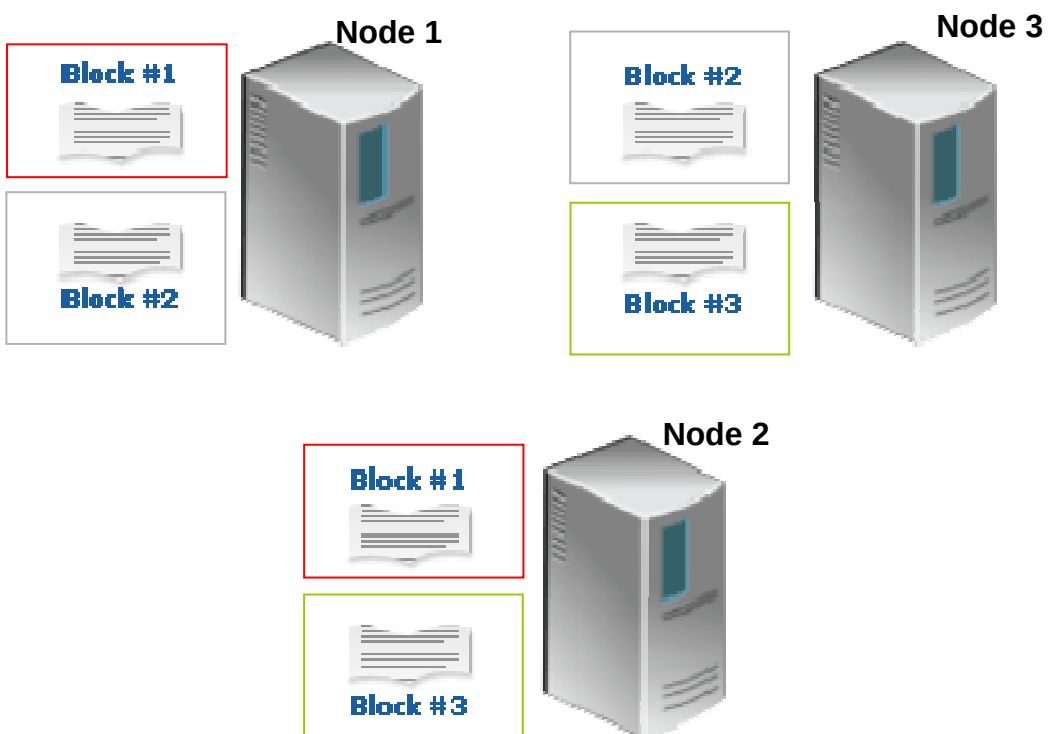
- **Fits well with replication to provide fault tolerance and availability**

- **Advantages of blocks:**
  - Fixed size – easy to calculate how many fit on a disk
  - A file can be larger than any single disk in the network
  - If a file or a chunk of the file is smaller than the block size, only needed space is used. Eg: 420MB file is split as:

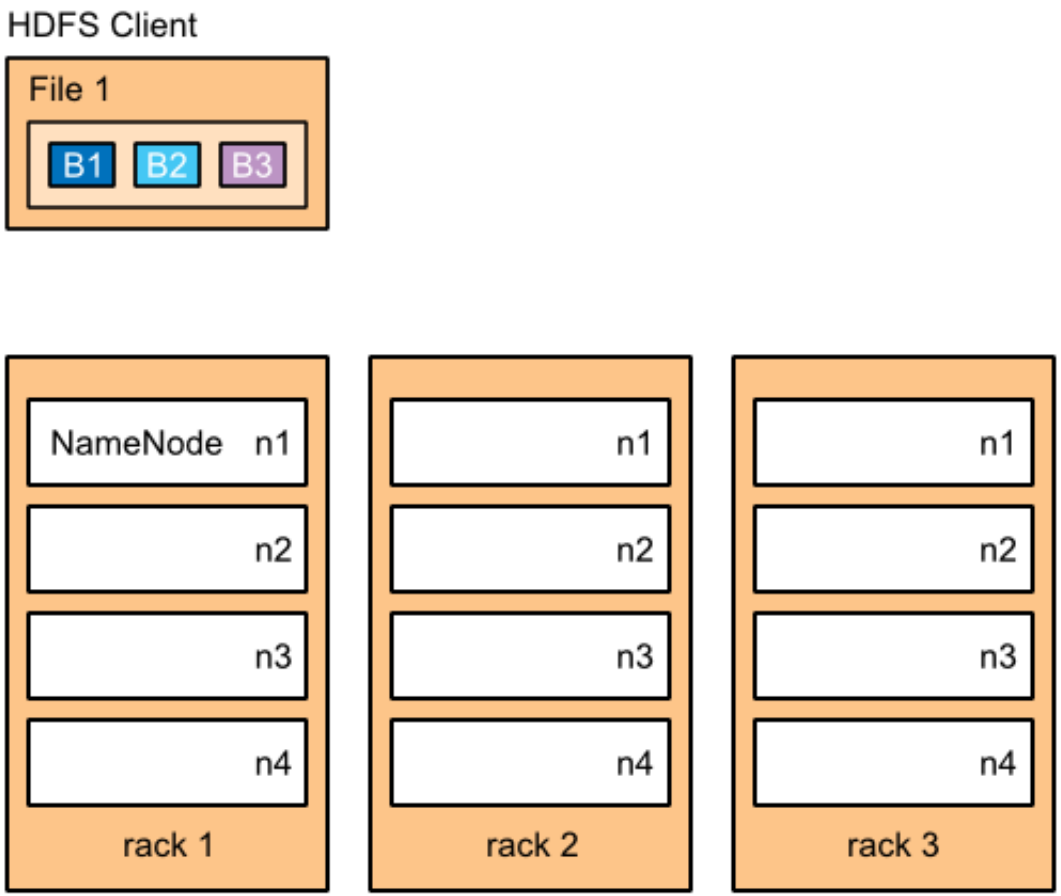


HDFS - Replication

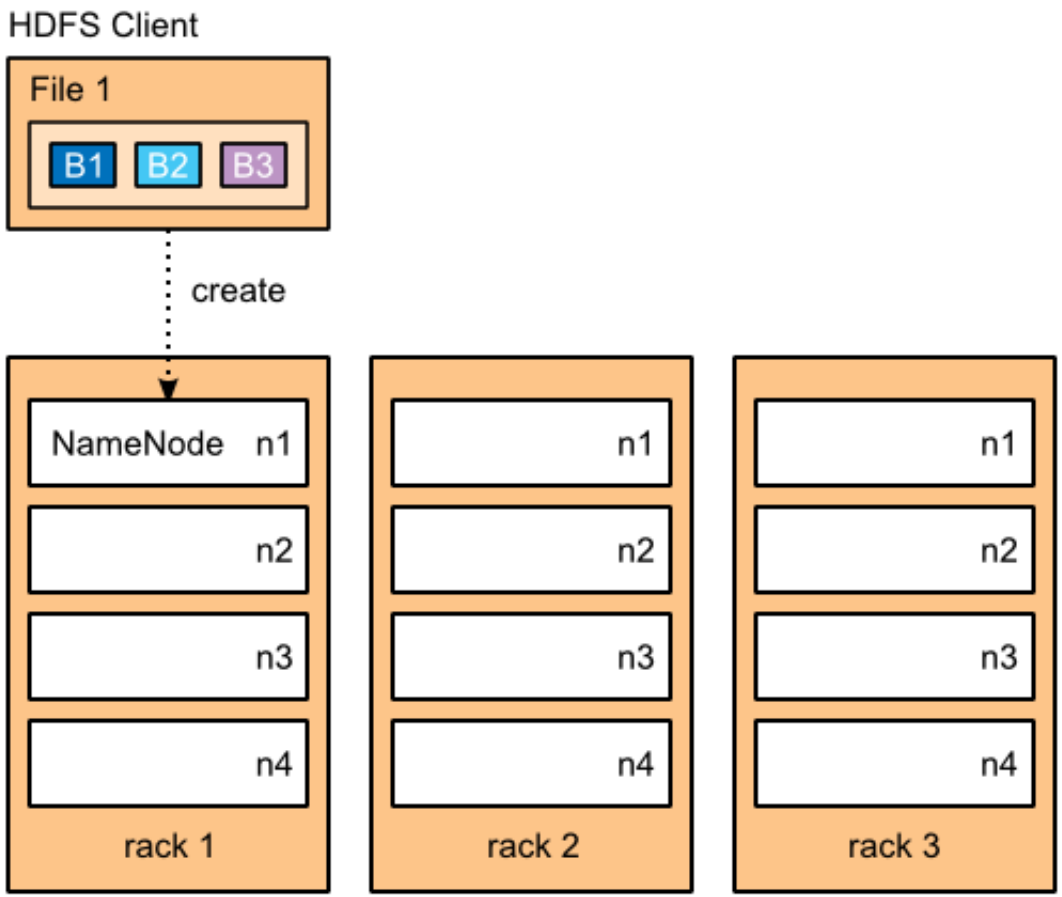
- Blocks with data are replicated to multiple nodes
- Allows for node failure without data loss



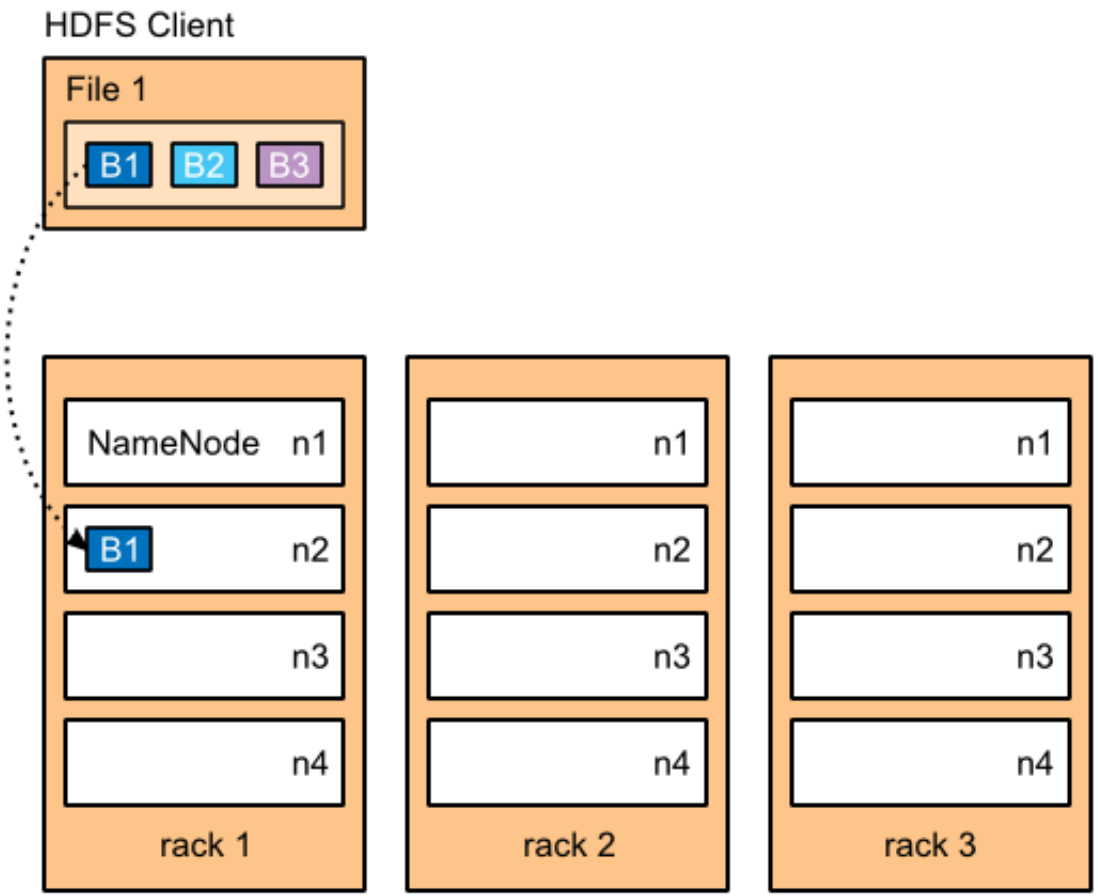
Writing a file to HDFS



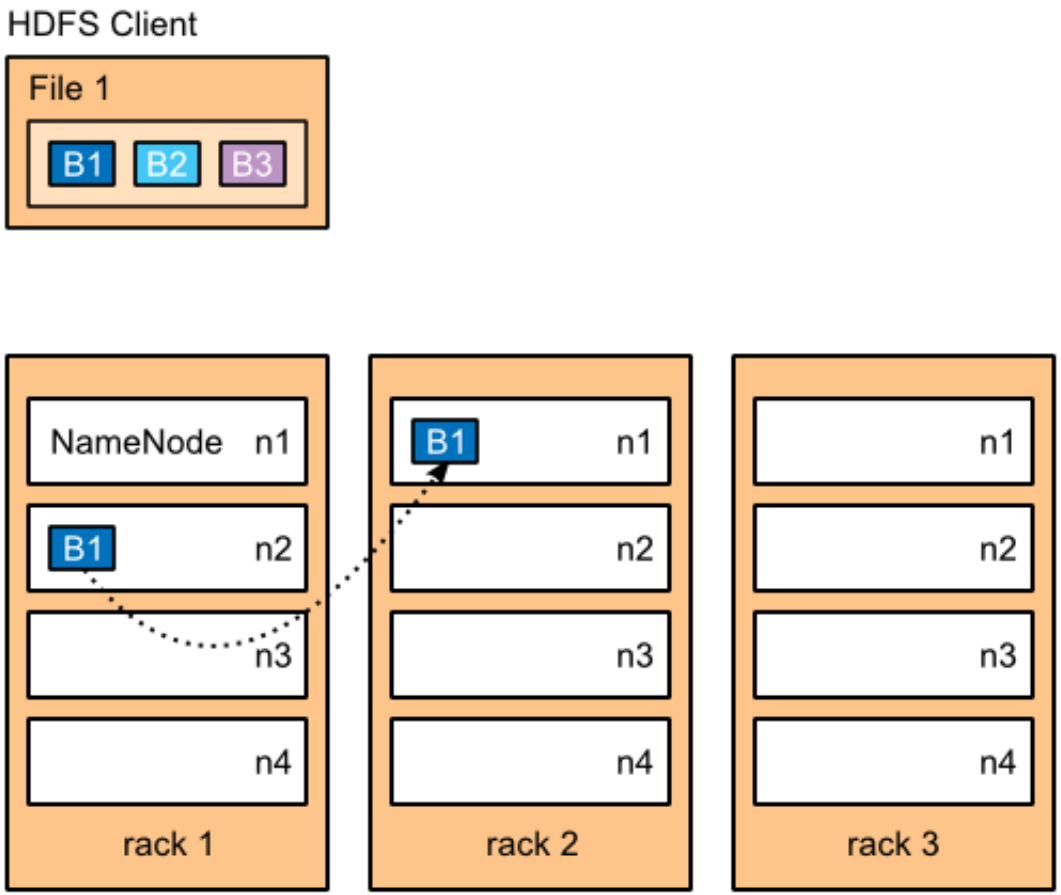
Writing a file to HDFS



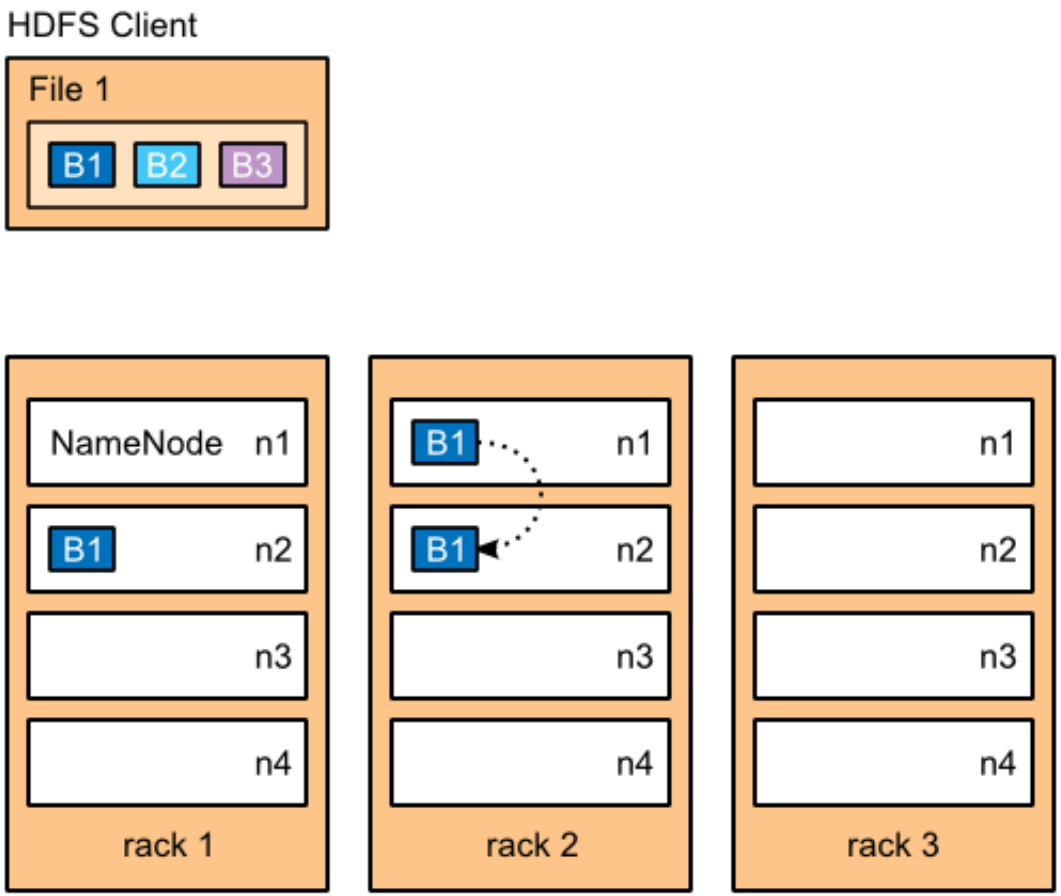
Writing a file to HDFS



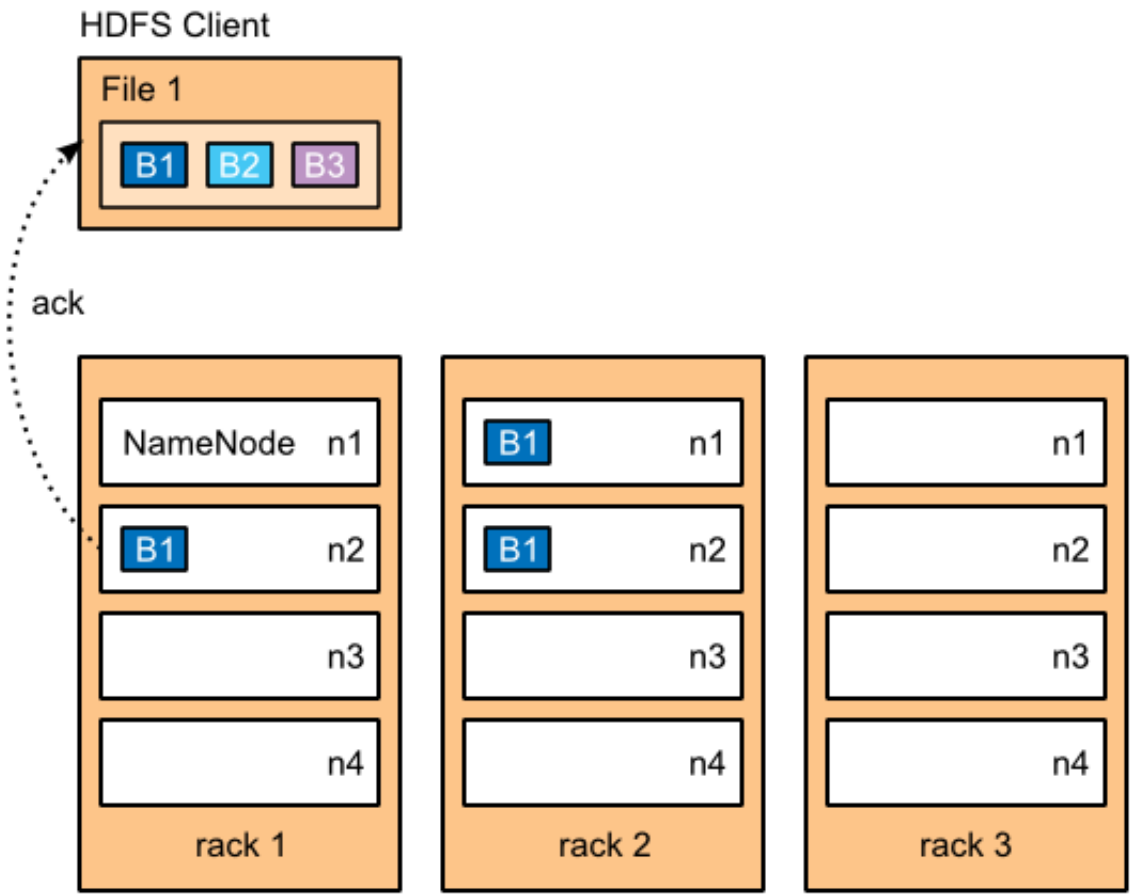
Writing a file to HDFS



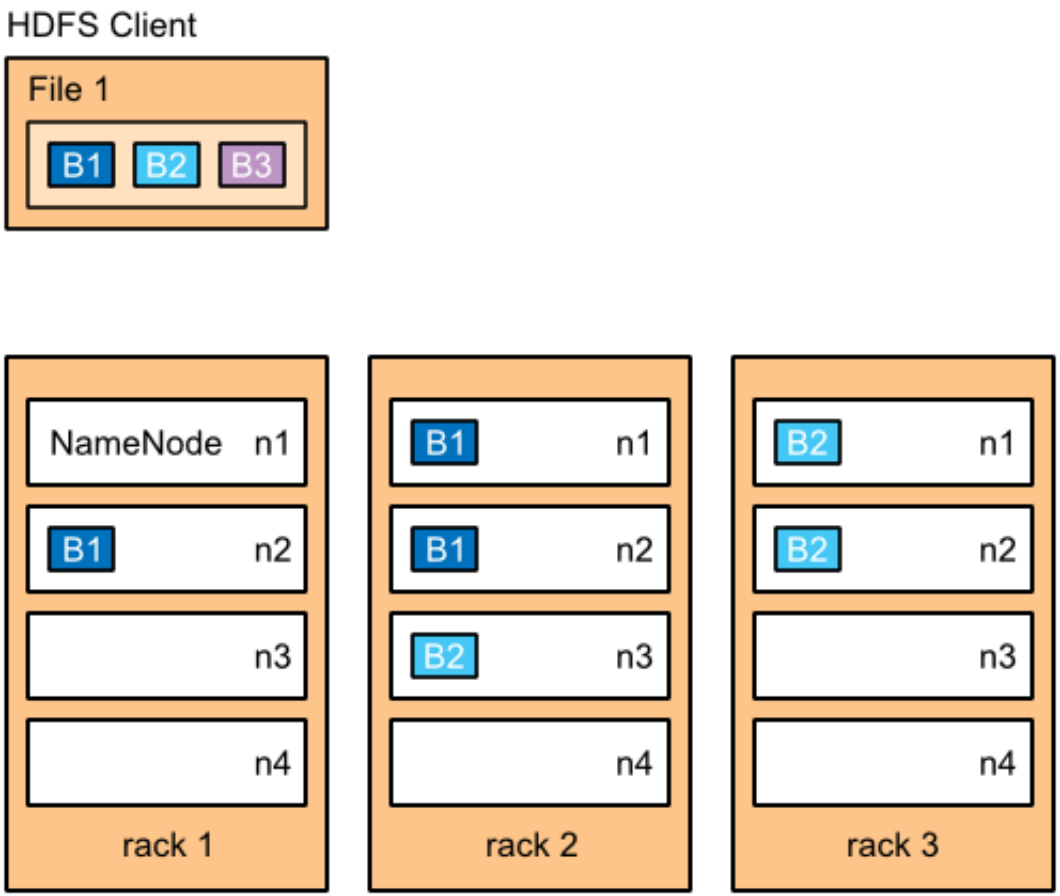
Writing a file to HDFS



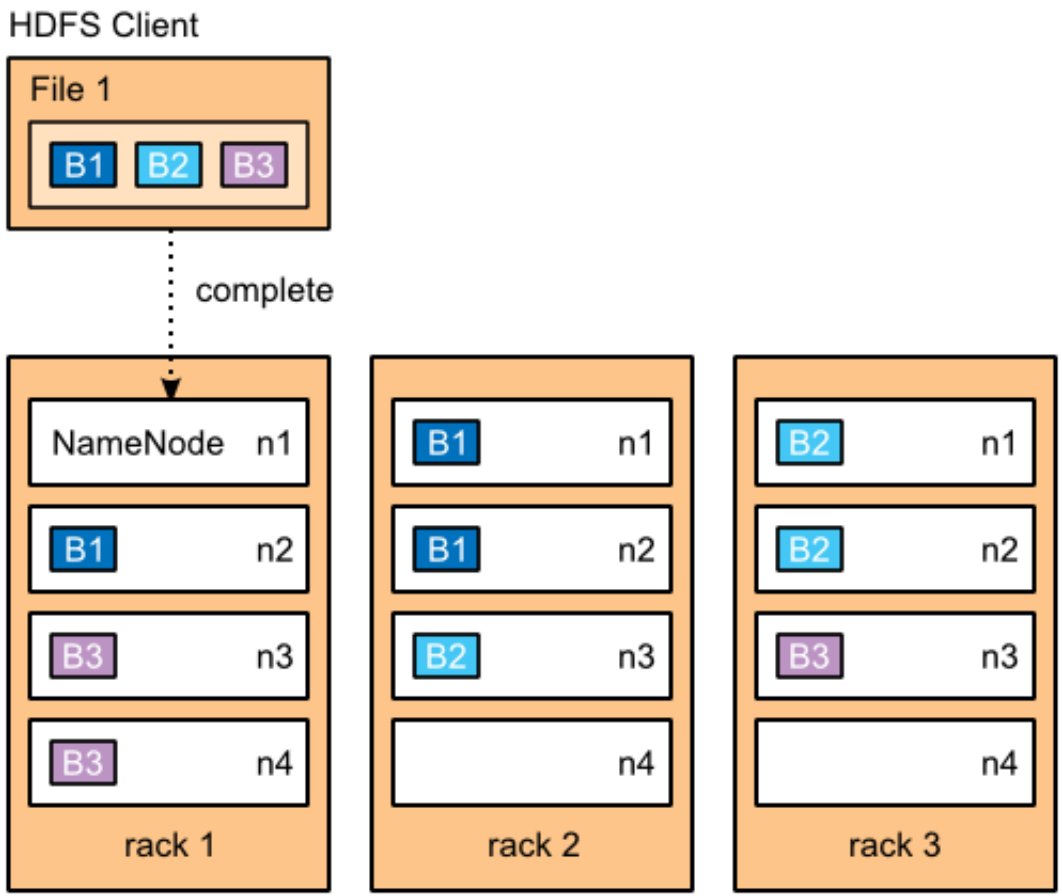
Writing a file to HDFS



Writing a file to HDFS



Writing a file to HDFS



HDFS Command line interface

- **File System Shell (fs)**
  - Invoked as follows:

```
hadoop fs <args>
```

- **Example:**
  - Listing the current directory in hdfs

```
hadoop fs –ls .
```

HDFS Command line interface

- **FS shell commands take paths URIs as argument**
  - URI format: `scheme://authority/path`
- **Scheme:**
  - For the local filesystem, the scheme is *file*
  - For HDFS, the scheme is *hdfs*

```
hadoop fs –copyFromLocal
        file://myfile.txt
        hdfs://localhost/user/keith/myfile.txt
```

- **Scheme and authority are optional**
  - Defaults are taken from configuration file core-site.xml

HDFS Command line interface

- **Many POSIX-like commands**
  - cat, chgrp, chmod, chown, cp, du, ls, mkdir, mv, rm, stat, tail
- **Some HDFS-specific commands**
  - copyFromLocal, copyToLocal, get, getmerge, put, setrep

MapReduce engine

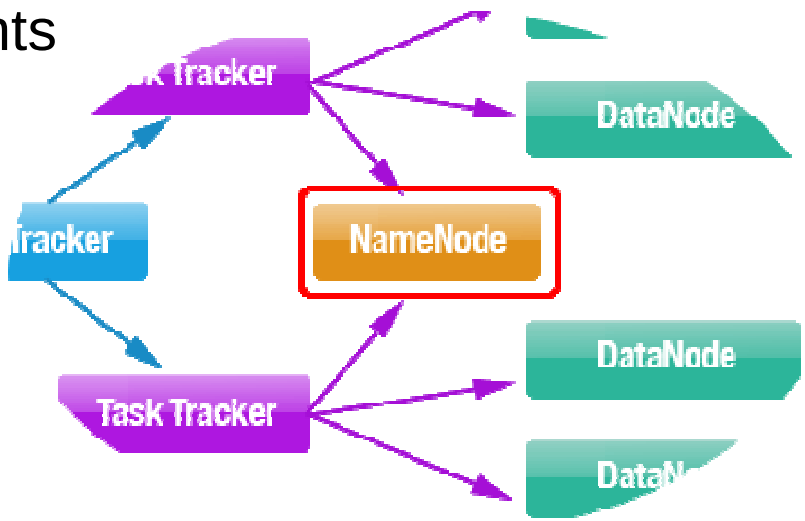
- Technology from Google
- A MapReduce program consists of map and reduce functions
- A MapReduce job is broken into tasks that run in parallel

Types of nodes - Overview

- **HDFS nodes**
  - NameNode
  - DataNode
- **MapReduce nodes**
  - JobTracker
  - TaskTracker
- **There are other nodes not discussed in this course**

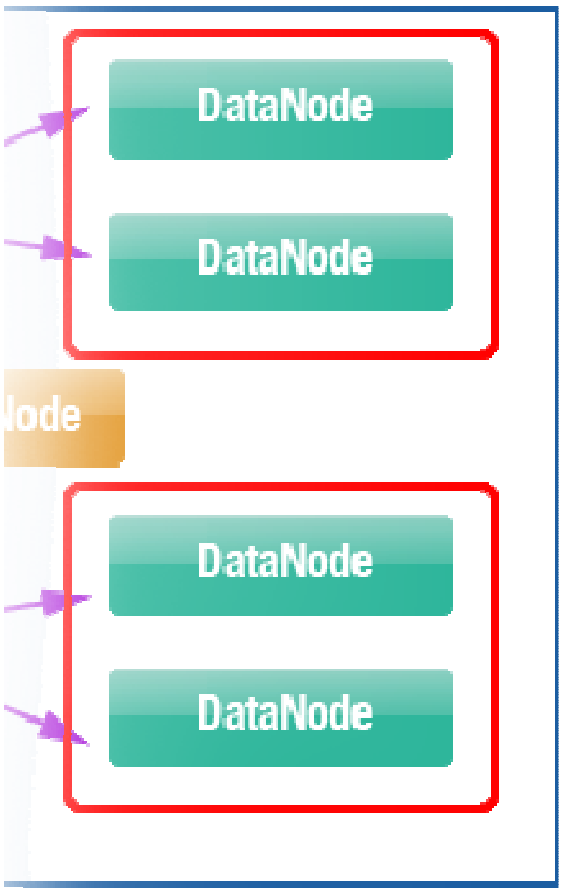
Types of nodes - NameNode

- **NameNode**
  - Only one per Hadoop cluster
  - Manages the filesystem namespace and metadata
  - Single point of failure, but mitigated by writing state to multiple filesystems
  - Single point of failure: Don't use inexpensive commodity hardware for this node, large memory requirements



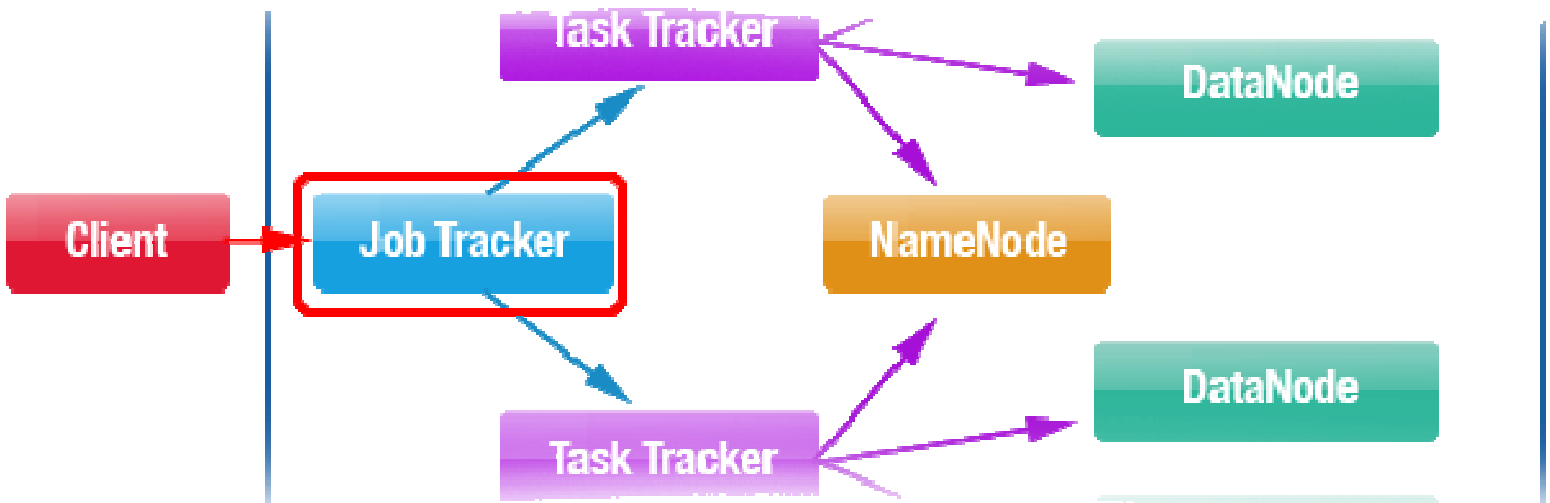
Types of nodes - DataNode

- **DataNode**
  - Many per Hadoop cluster
  - Manages blocks with data and serves them to clients
  - Periodically reports to name node the list of blocks it stores
  - Use inexpensive commodity hardware for this node



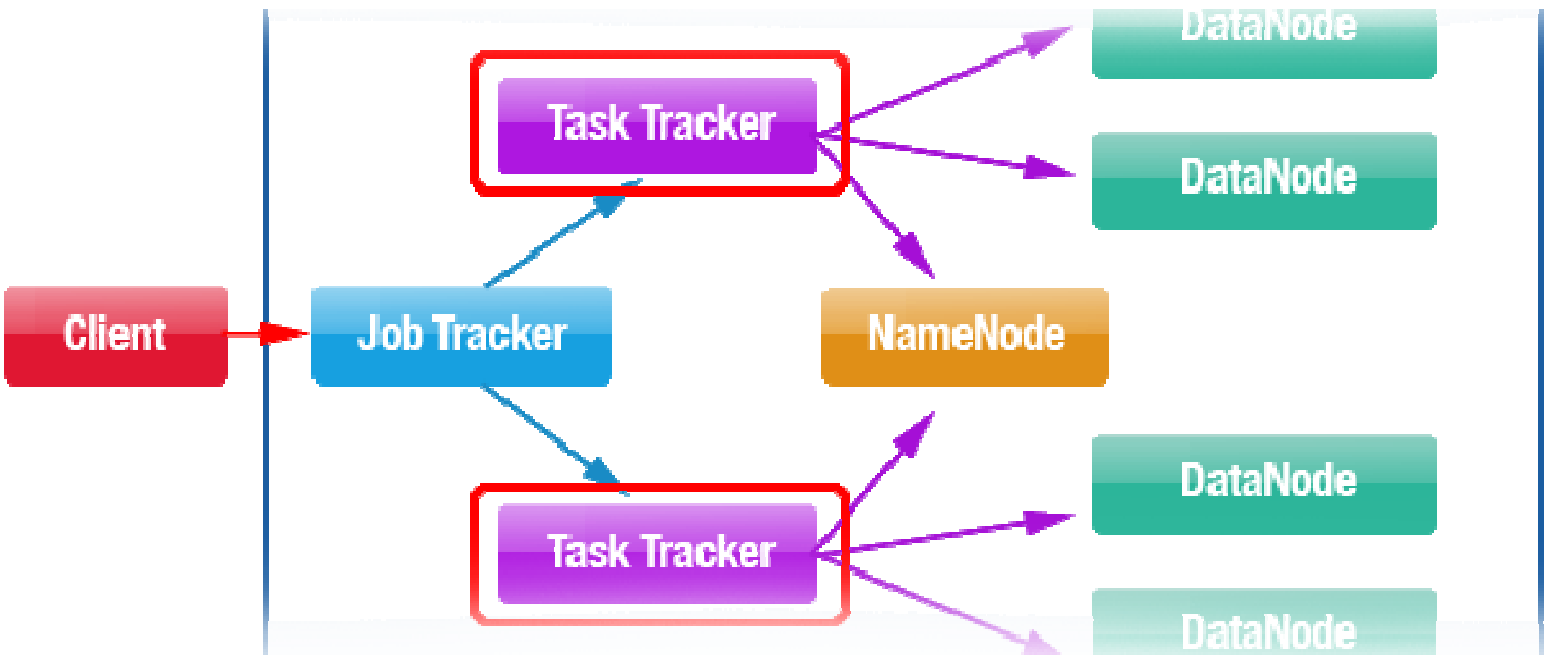
Types of nodes - JobTracker

- **JobTracker node**
  - One per Hadoop cluster
  - Receives job requests submitted by client
  - Schedules and monitors MapReduce jobs on task trackers



Types of nodes - TaskTracker

- **TaskTracker node**
  - Many per Hadoop cluster
  - Executes MapReduce operations
  - Reads blocks from DataNodes



What is a Map operation?

- Doing something to every element in an array is a common operation:

```
var a = [1,2,3];
for (i = 0; i < a.length; i++)
    a[i] = a[i] * 2;
```

What is a Map operation?

- Doing something to every element in an array is a common operation:

```
var a = [1,2,3];
for (i = 0; i < a.length; i++)
     a[i] = a[i] * 2;
```

- New value for variable **a** would be:

```
var a = [2,4,6];
```

What is a Map operation?

- Doing something to every element in an array is a common operation:

```
var a = [1,2,3];
for (i = 0; i < a.length; i++)
    a[i] = a[i] * 2;
```

This can be written as a function

- New value for variable **a** would be:

```
var a = [2,4,6];
```

What is a Map operation?

- Doing something to every element in an array is a common operation:

```
var a = [1,2,3];
for (i = 0; i < a.length; i++)
    a[i] = fn(a[i]);
```

Like this, where *fn* is a function defined as:
`function fn(x) {return x*2;}`

- New value for variable **a** would be:

```
var a = [2,4,6];
```

What is a Map operation?

- Doing something to every element in an array is a common operation:

```
var a = [1,2,3];
for (i = 0; i < a.length; i++)
     a[i] = fn(a[i]);
```

Now, all of this can also be converted into a “map” function

What is a Map operation?

- …like this, where `fn` is a function passed as an *argument*:

```
function map(fn, a) {
  for (i = 0; i < a.length; i++)
        a[i] = fn(a[i]);
}
```

- You can invoke this map function like this:

```
map(function(x){return x*2;}, a);
```

What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

```
function sum(a) {
  var s = 0;
  for (i = 0; i < a.length; i++)
       s += a[i];
  return s;
}
```

What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

```
function sum(a) {
  var s = 0;
  for (i = 0; i < a.length; i++)
    s += a[i];
  return s;
}
```

This can be written as a function

What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

```
function sum(a) {
  var s = 0;
  for (i = 0; i < a.length; i++)
    s = fn(s,a[i]);
  return s;
}
```

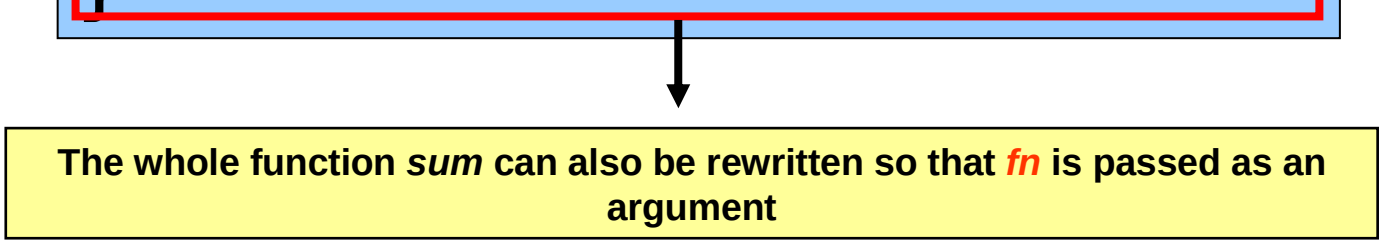
Like this, where function *fn* is defined so it adds its arguments:

```
function fn(a,b){
  return a+b;
}
```

What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

```
function sum(a) {
  var s = 0;
  for (i = 0; i < a.length; i++)
      s = fn(s, a[i]);
  return s;
}
```



What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

```
function reduce(fn, a, init) {
  var s = init;
  for (i = 0; i < a.length; i++)
      s = fn(s, a[i]);
  return s;
}
```

Like this… The function name was changed to *reduce*, and now it takes three arguments, a function, an array, and an initial value

What is a Reduce operation?

- Another common operation on arrays is to combine all their values:

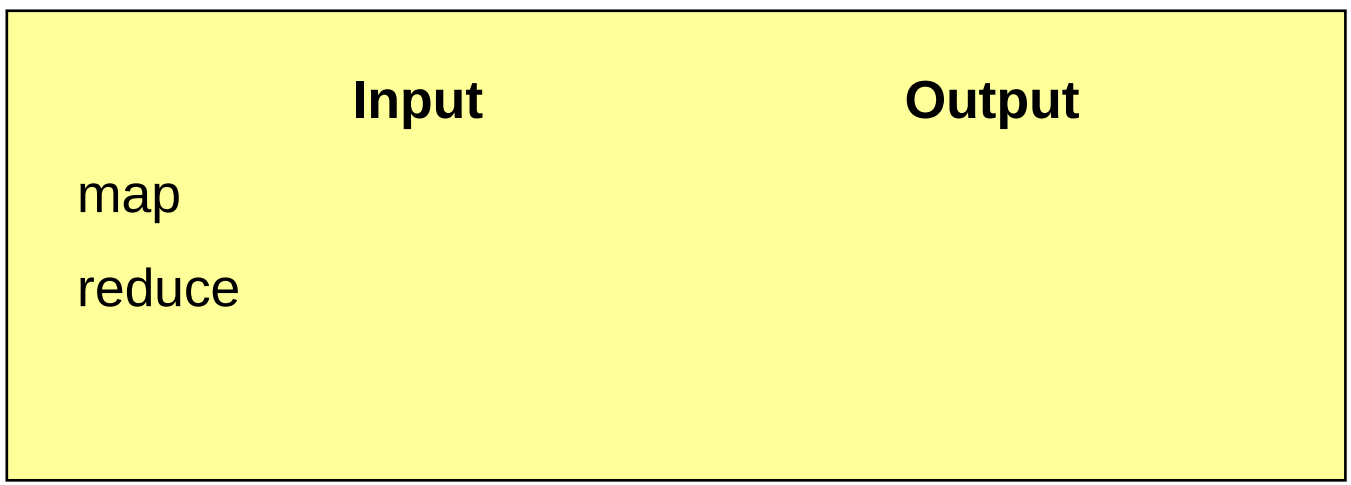
```
function sum(a) {
  var s = 0;
  for (i = 0; i < a.length; i++)
       s += a[i];
  return s;
}
```

as a reduce operation:

```
reduce(function(a,b){return a+b;},a,0);
```

Two Fundamental data types

- Key/value pairs
- Lists



Two Fundamental data types

- Key/value pairs
- Lists

	Input	Output
map	<k1, v1>	
reduce		

Two Fundamental data types

- Key/value pairs
- Lists

	Input	**Output**
map	<k1, v1>	list(<k2, v2>)
reduce		

Two Fundamental data types

- Key/value pairs
- Lists

	Input	**Output**
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	

Two Fundamental data types

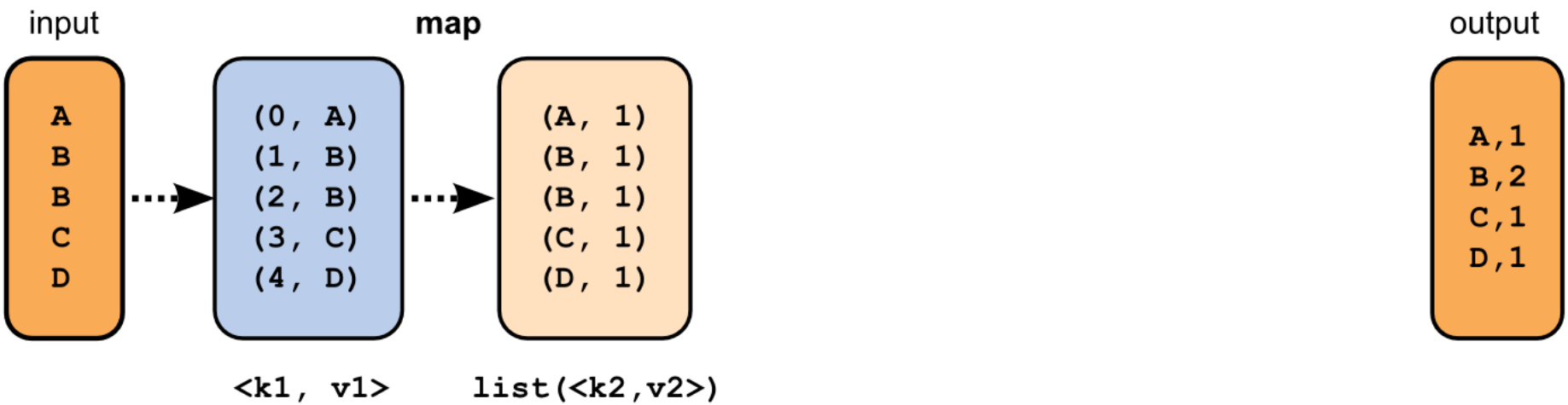
- Key/value pairs
- Lists

	Input	**Output**
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	list(<k3, v3>)

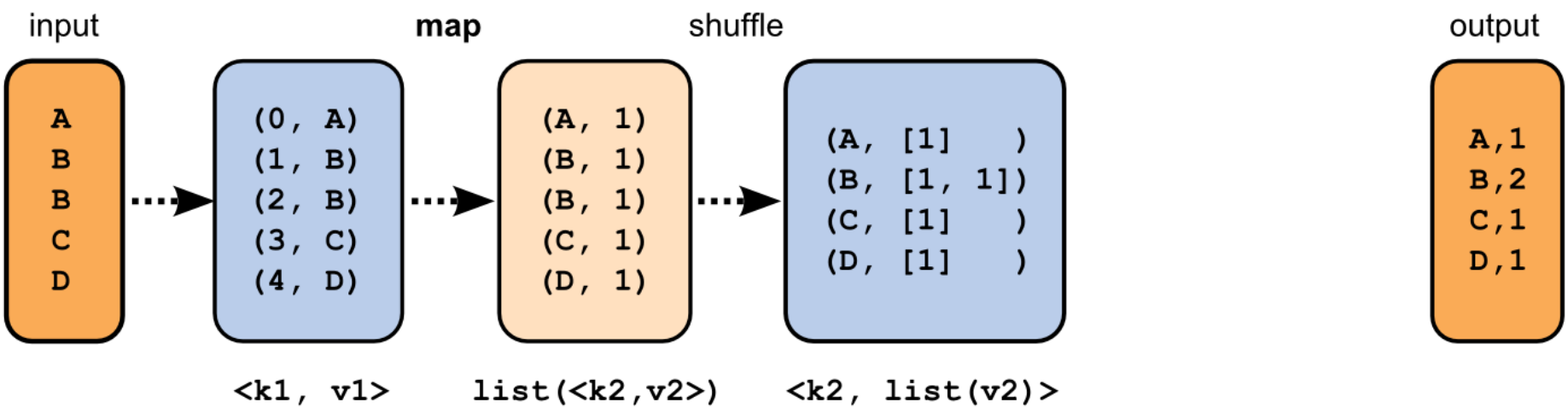
Simple data flow example

Simple data flow example

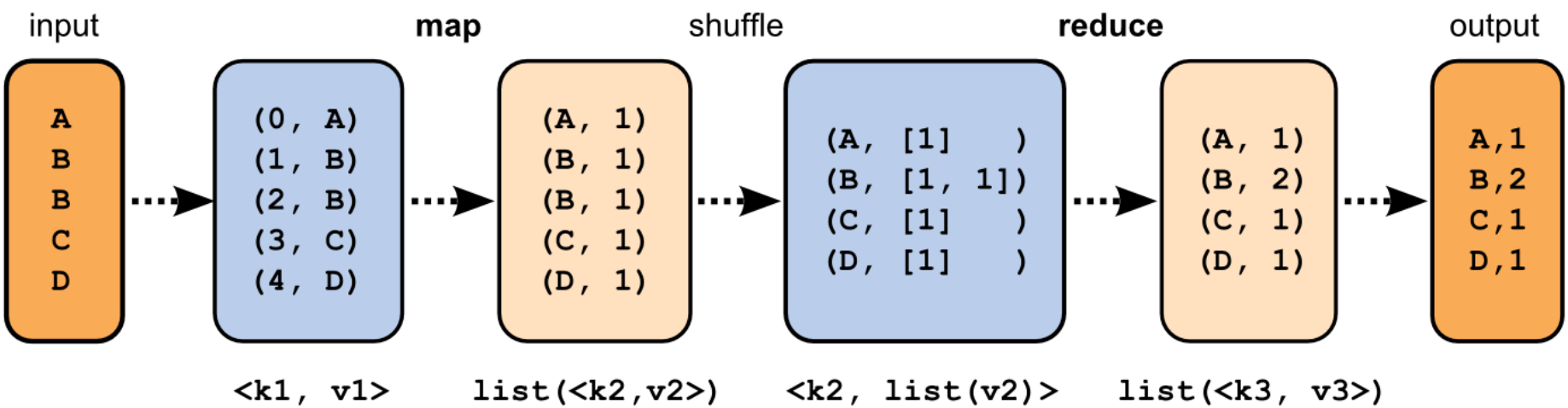
Simple data flow example



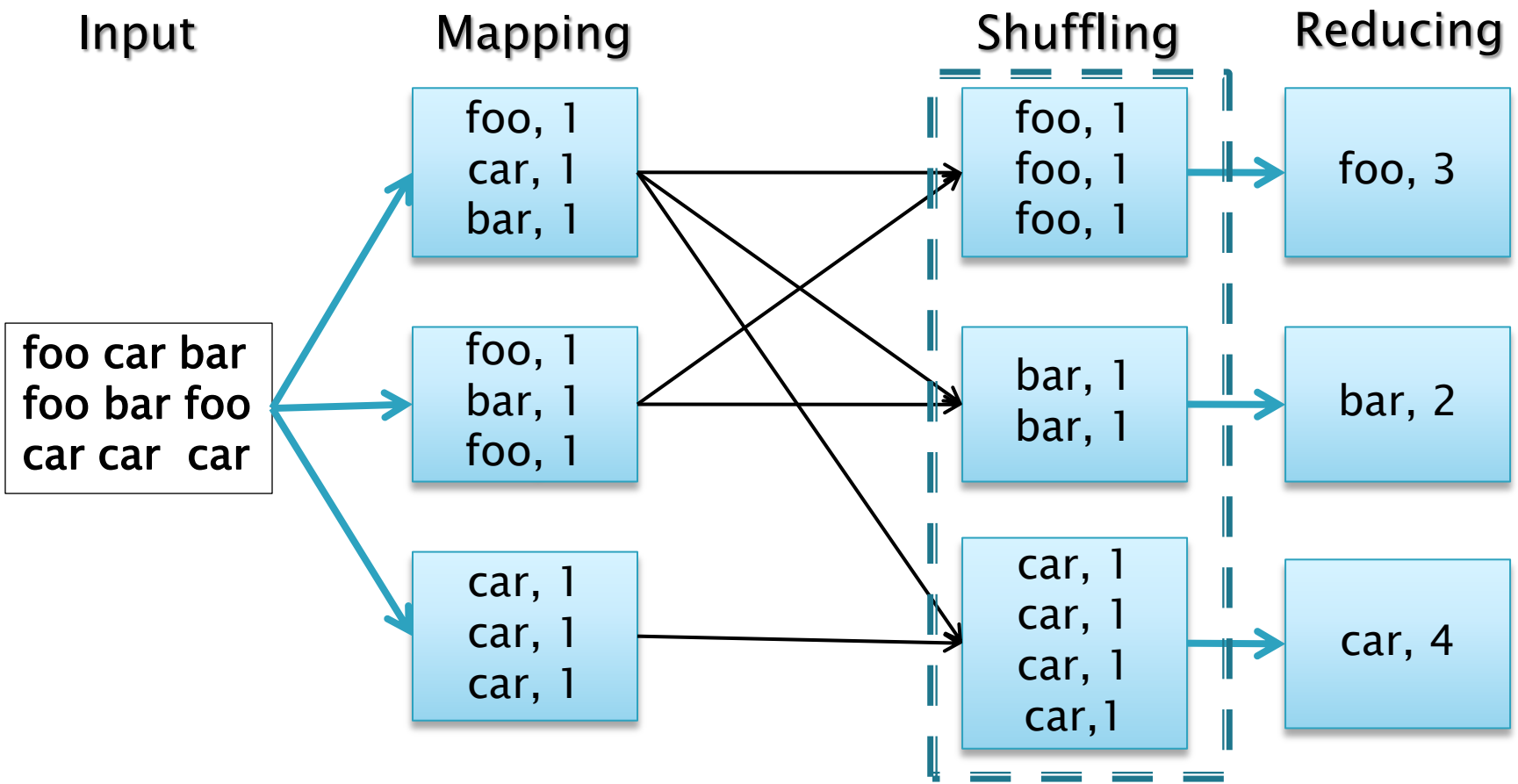
Simple data flow example



Simple data flow example



Word Count


Input
Mapping
Shuffling
Reducing
foo car bar
foo bar foo
car car car
foo, 1
car, 1
bar, 1
foo, 1
bar, 1
foo, 1
car, 1
car, 1
car, 1
foo, 1
foo, 1
foo, 1
bar, 1
bar, 1
car, 1
car, 1
car, 1
car,1
foo, 3
bar, 2
car, 4

WordCount Map

```
void Map (key, value) {
        for each word x in value:
            output.collect(x, 1);
}
```

WordCount Map

```
public static class Map
        extends Mapper<LongWritable, Text, Text, IntWritable>{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context )
      throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());

      while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
          context.write(word, one);
      }
```

WordCount Reduce

```
void Reduce (keyword, <list of value>){
        for each x in <list of value>:
                sum+=x;
        output.collect(keyword, sum);
}
```

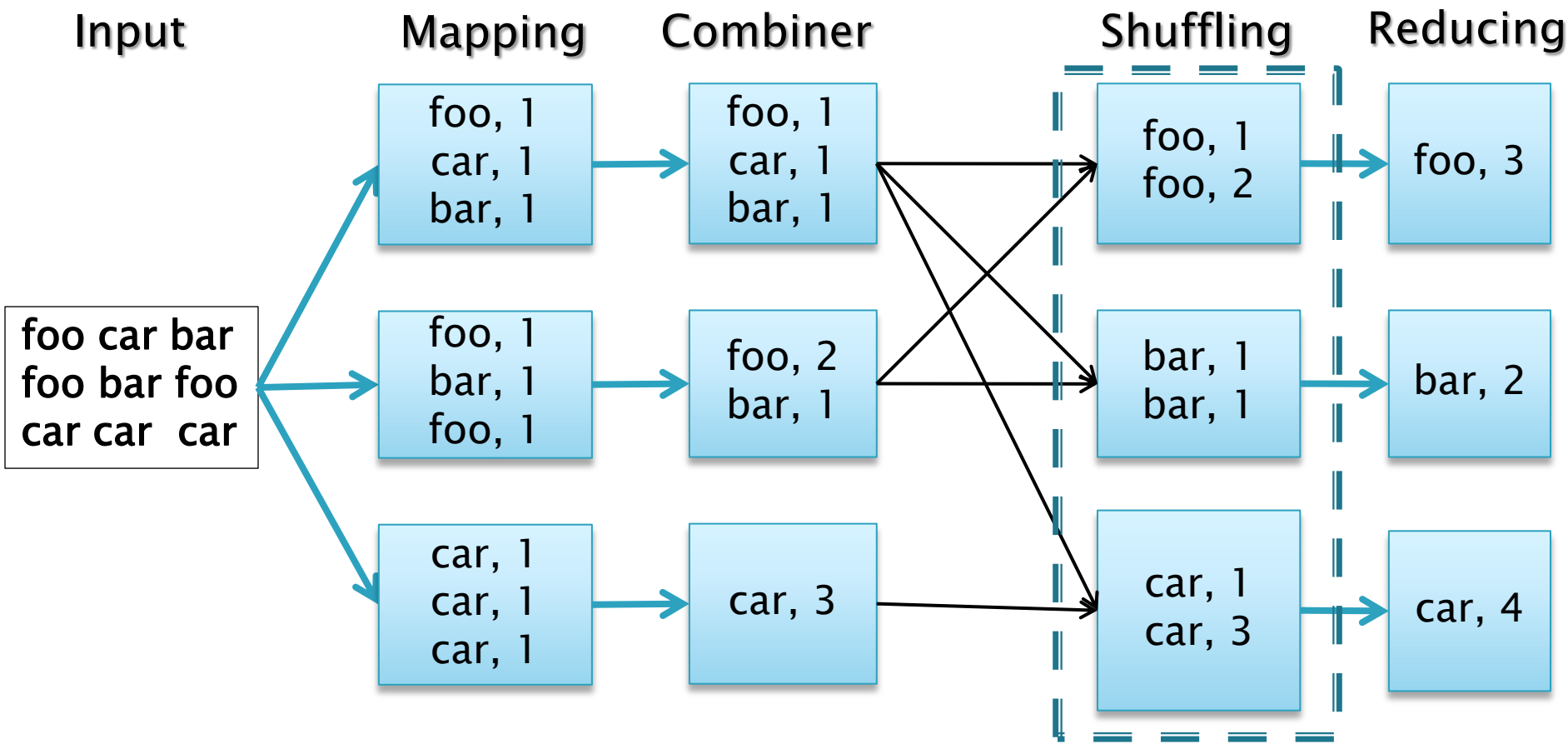
WordCount Reduce

```
public static class Reduce
      extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context) throws …..{
        int sum = 0; // initialize the sum for each keyword
        for (IntWritable val : values) {
                sum += val.get();
        }
        result.set(sum);

        context.write(key, result);
  }
}
```

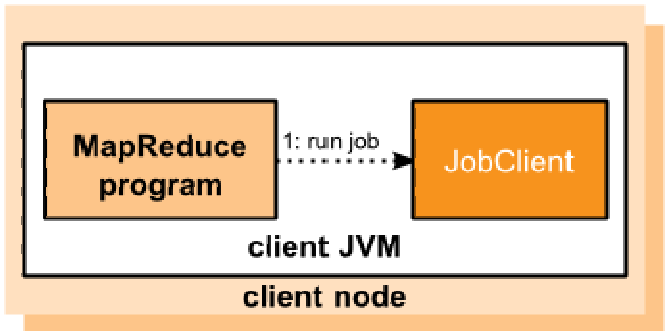
Optimizations

- Reduce phase can only be activated until all map tasks finish
  - It will be a waste if there is a extremely long map task
- “Combiner” functions can run on same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth

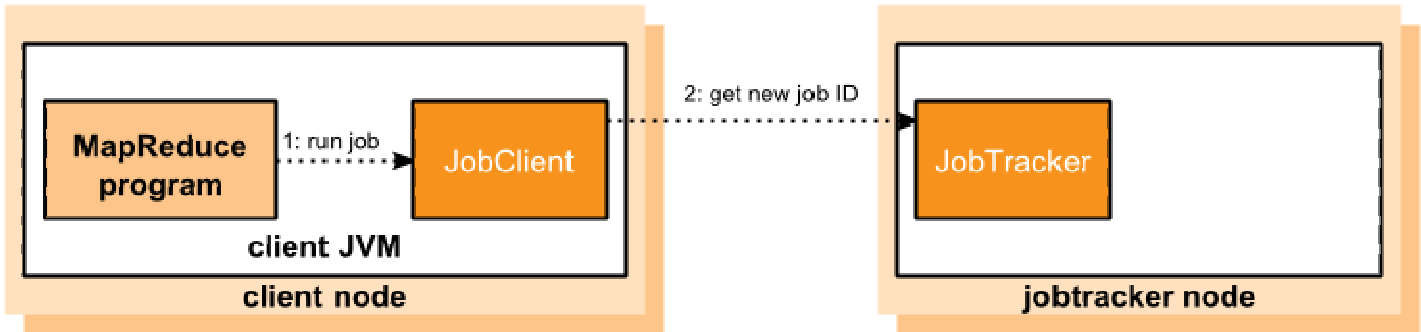
Word Count with Combiner



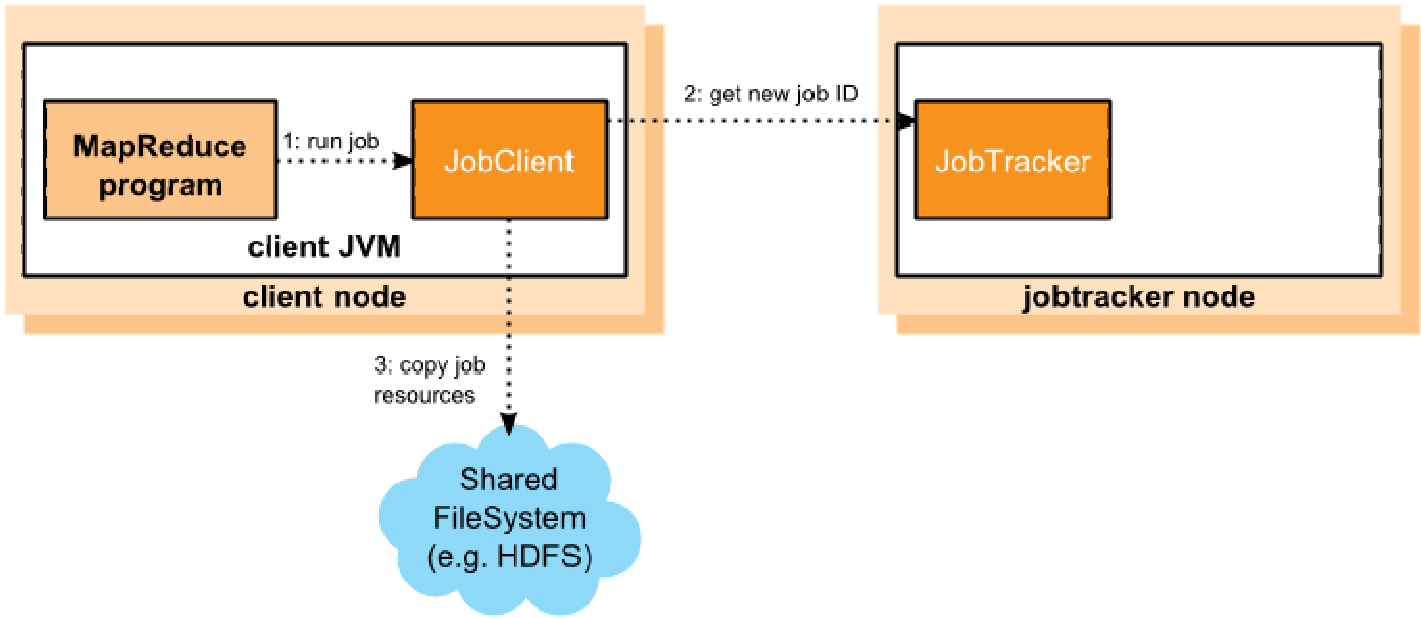
Submitting a MapReduce job



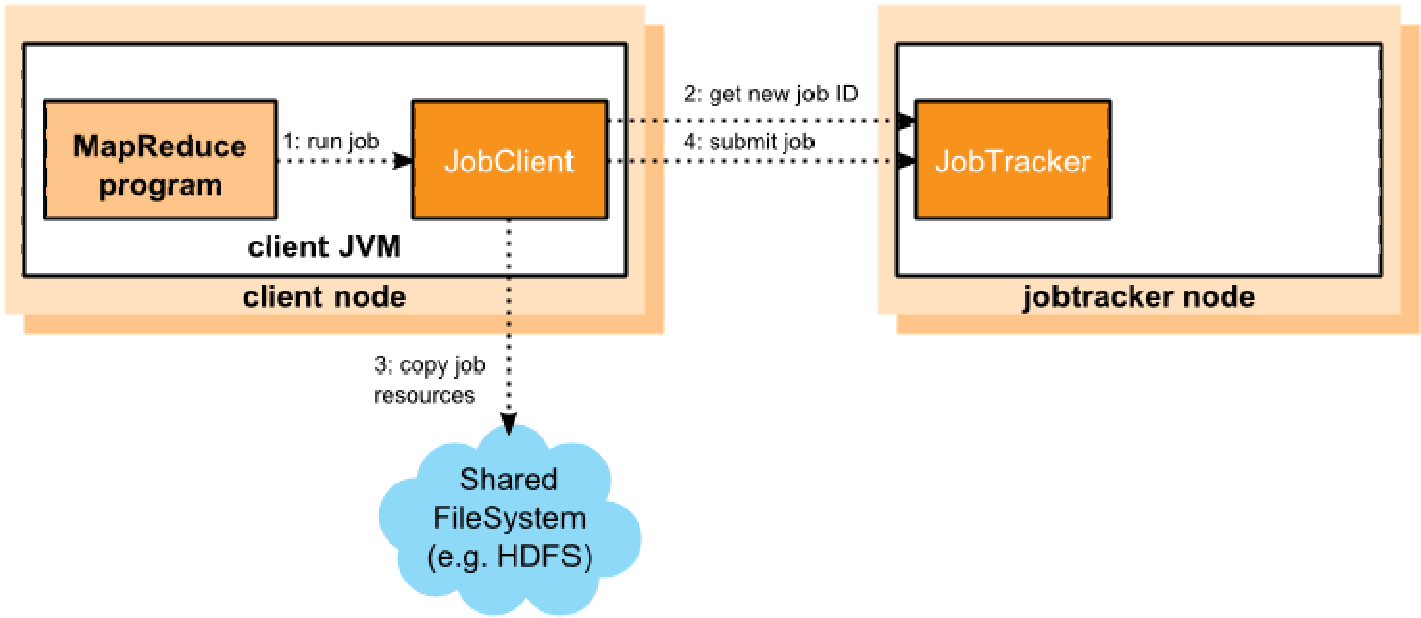
Submitting a MapReduce job



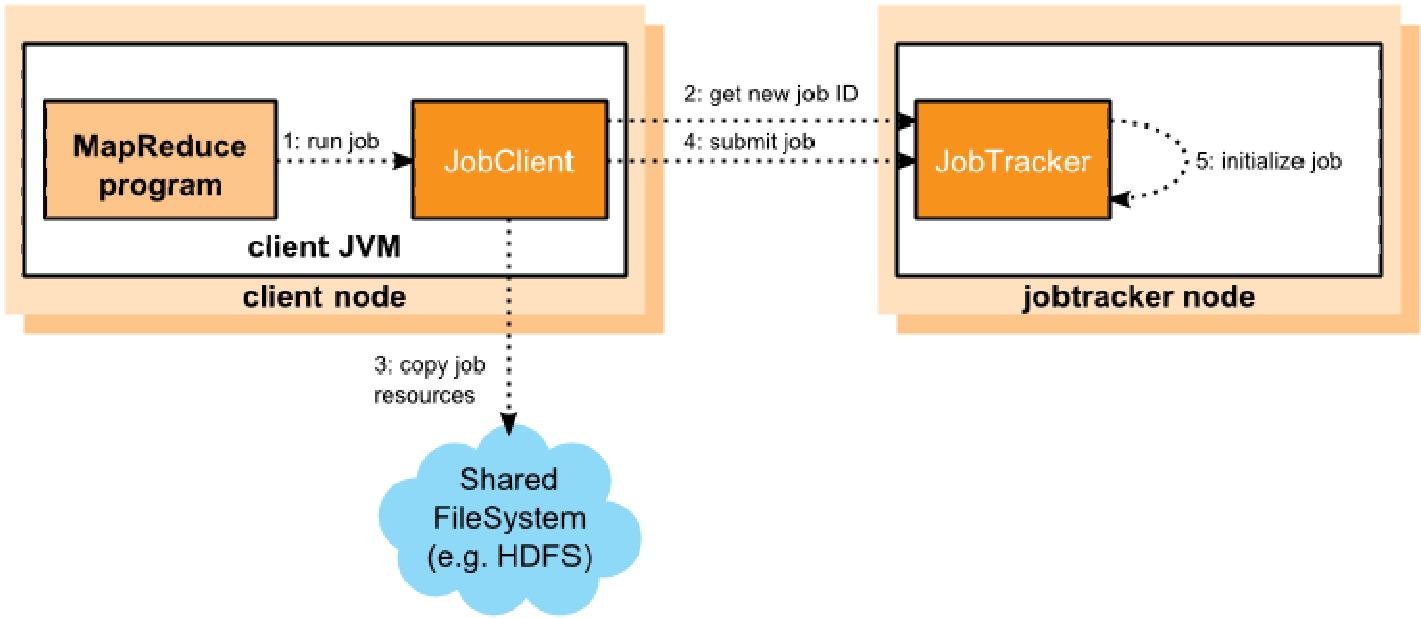
Submitting a MapReduce job



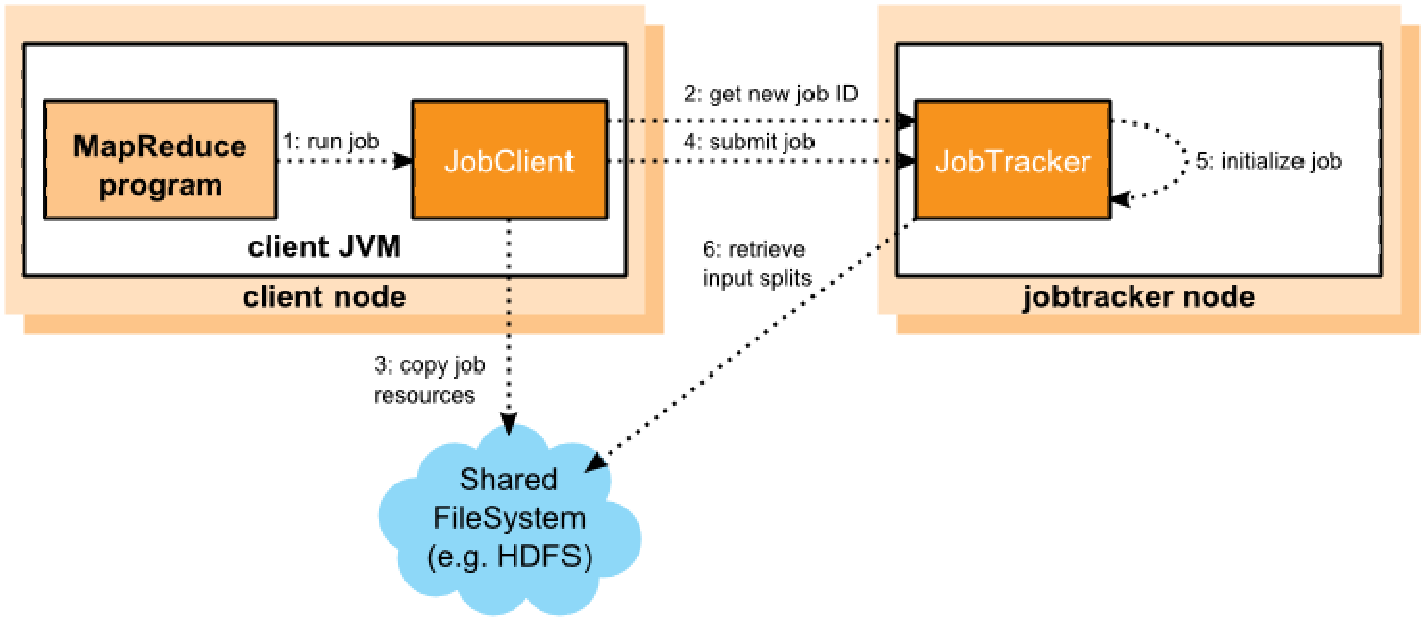
Submitting a MapReduce job



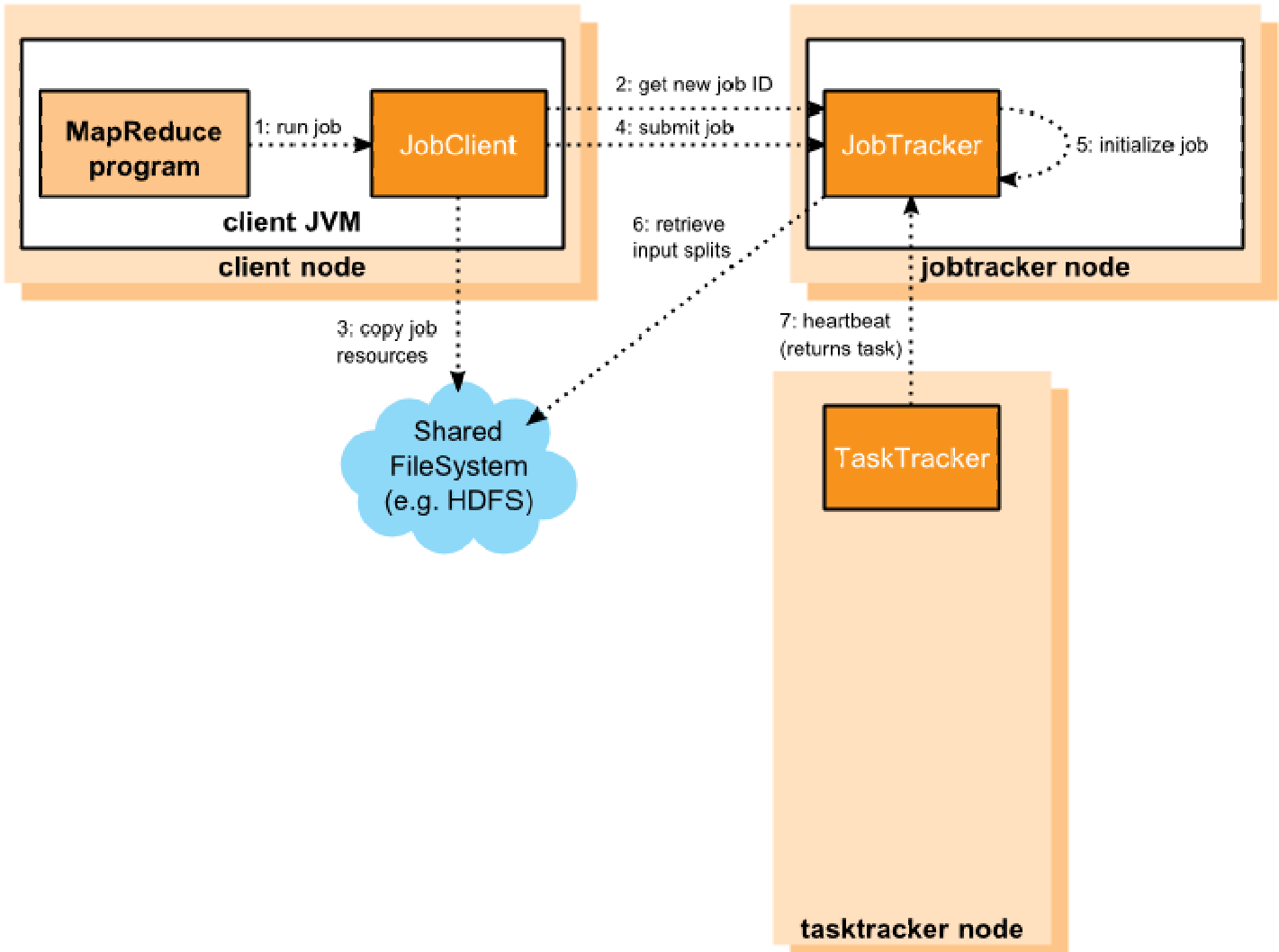
Submitting a MapReduce job



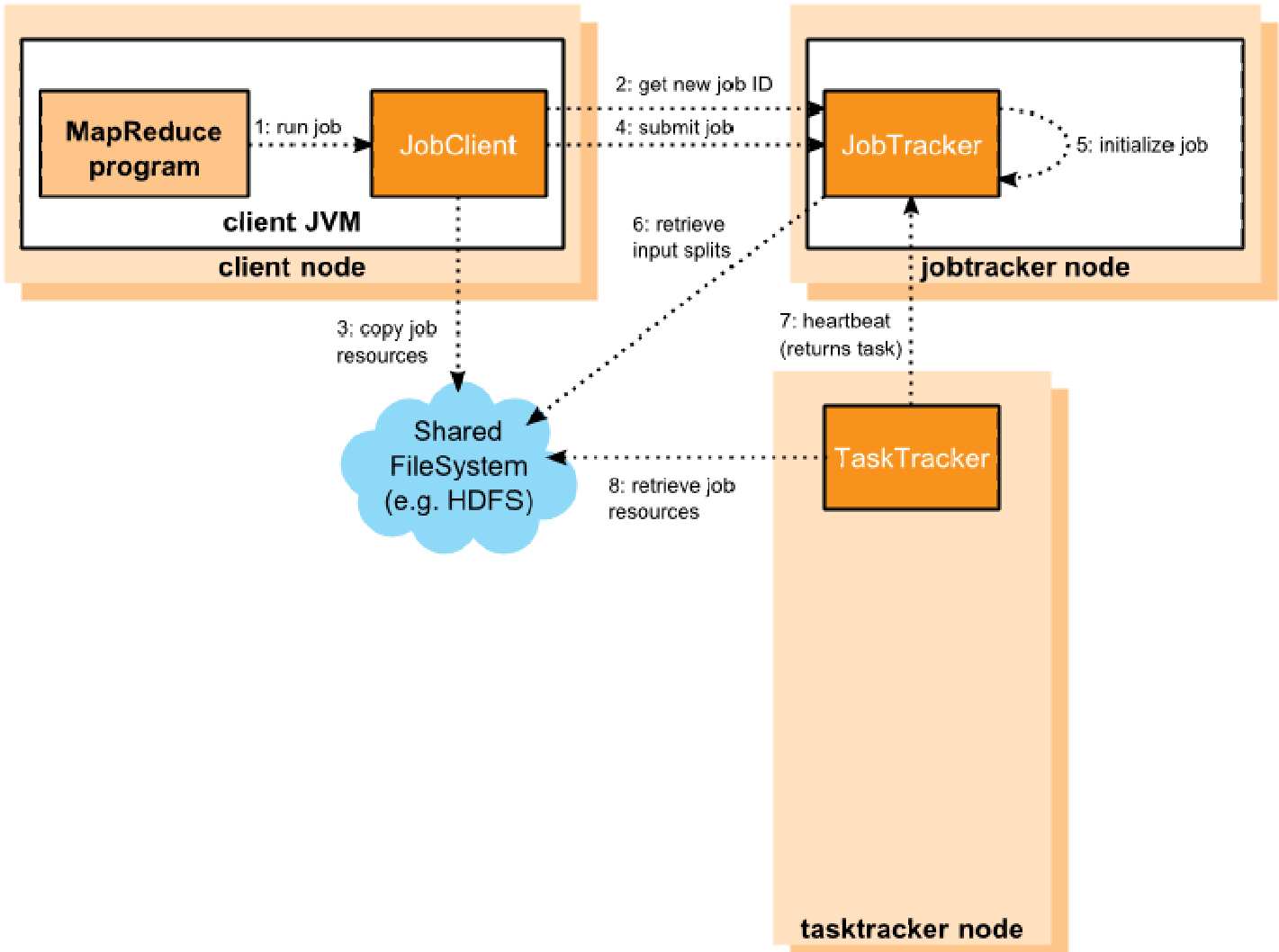
Submitting a MapReduce job



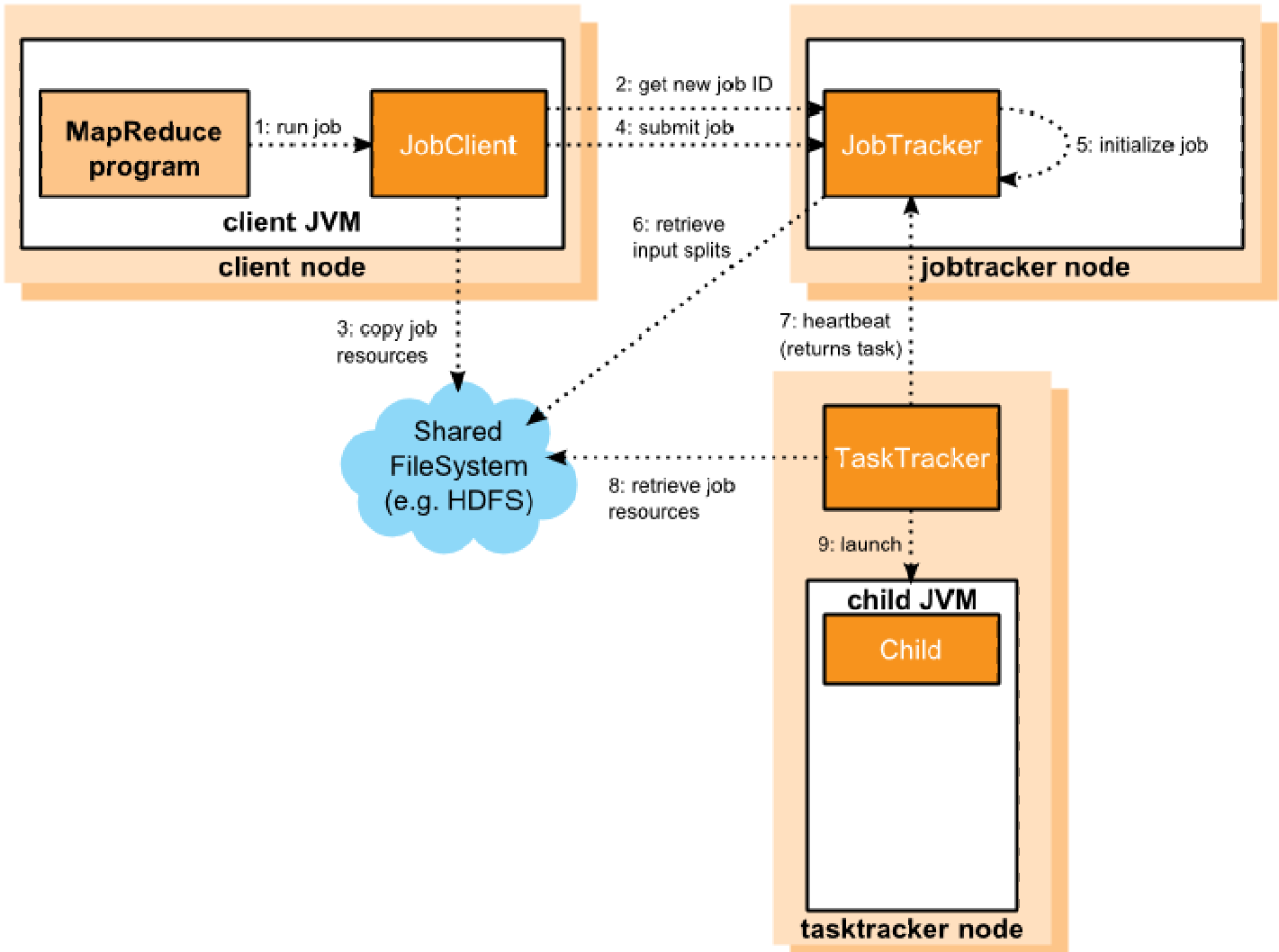
Submitting a MapReduce job



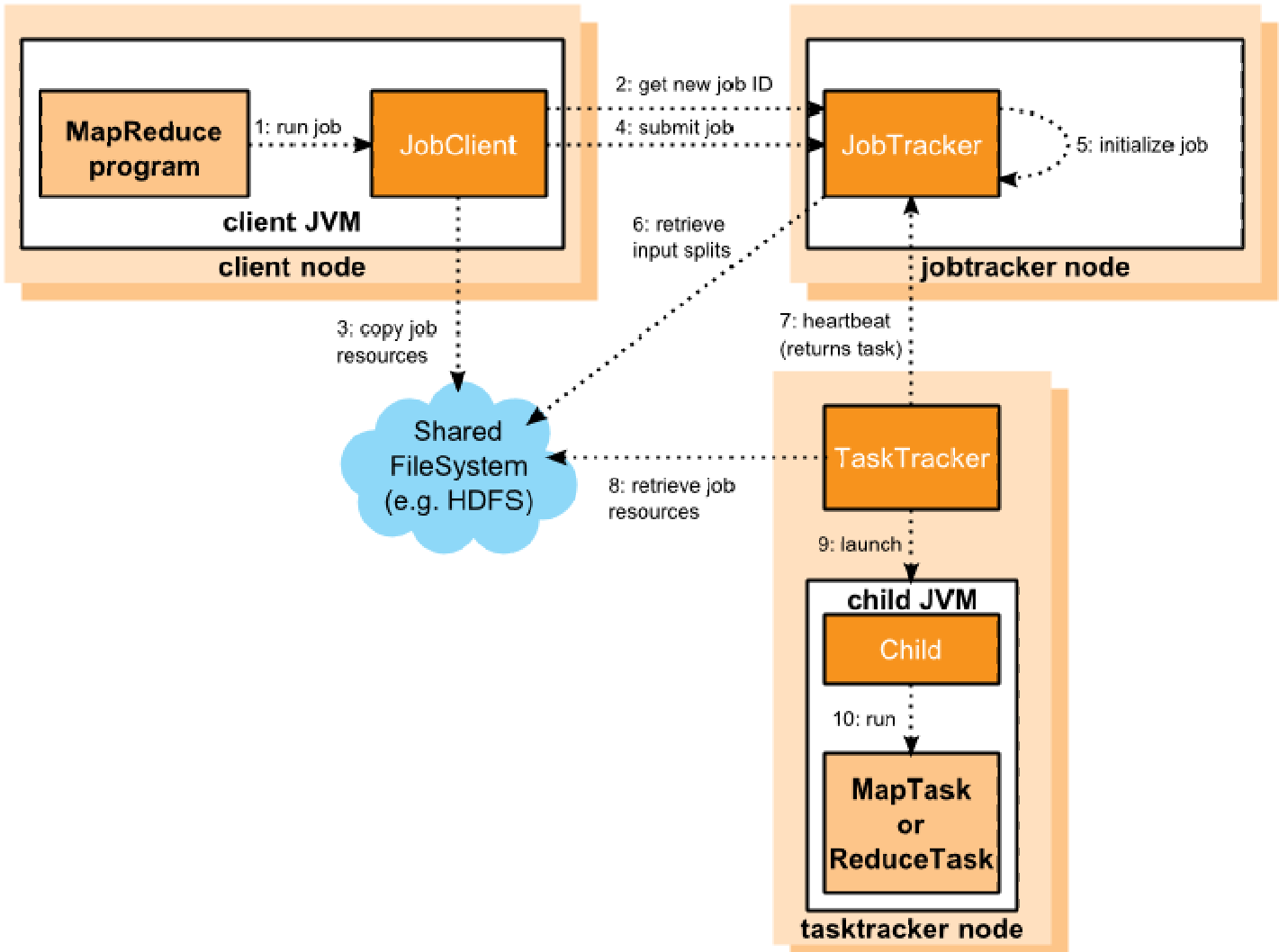
Submitting a MapReduce job



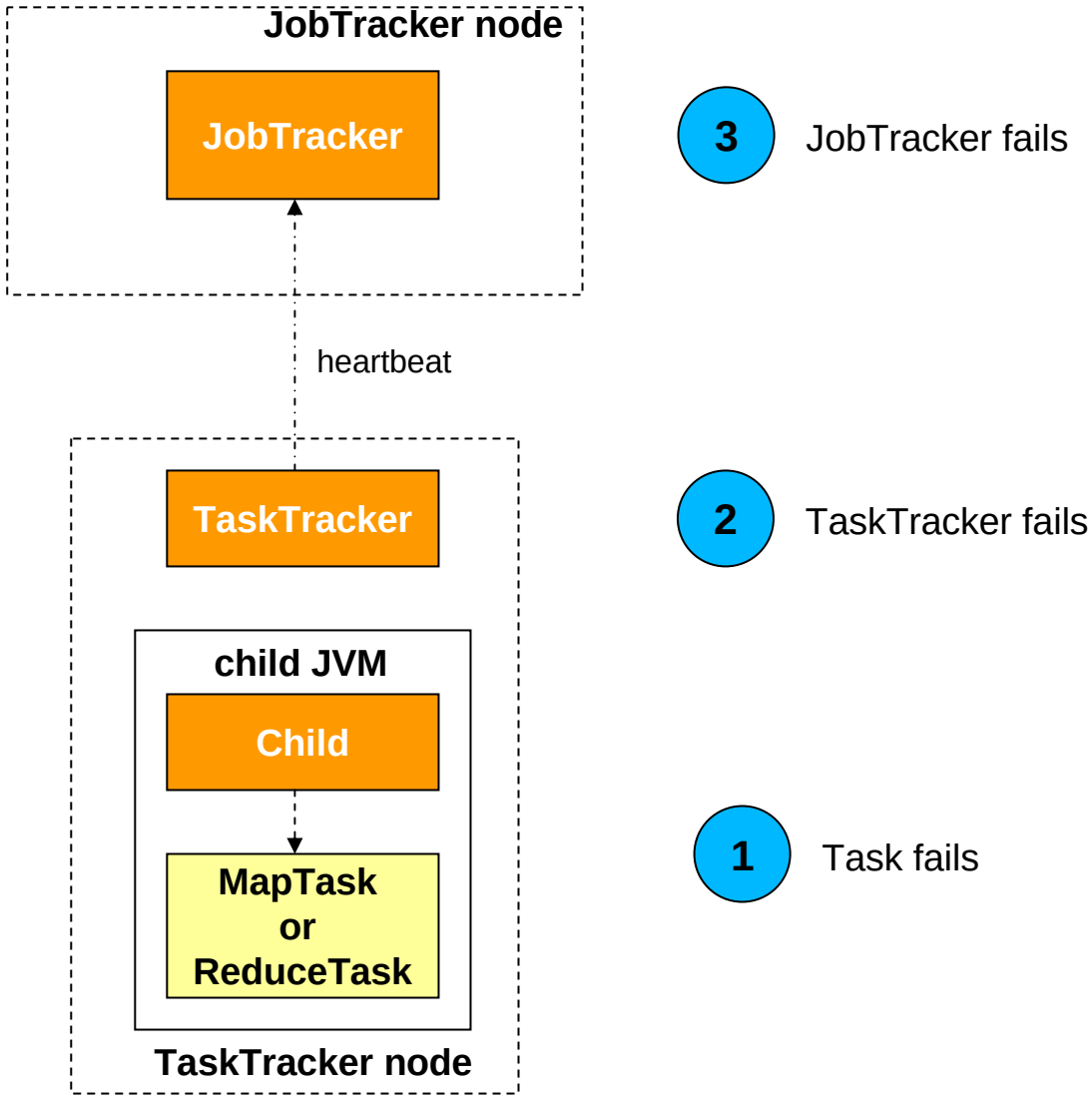
Submitting a MapReduce job



Submitting a MapReduce job

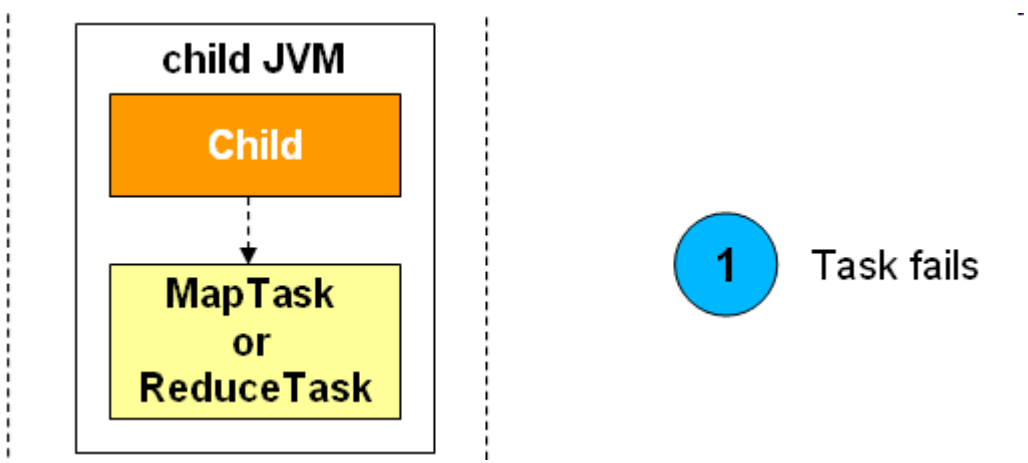


Fault tolerance



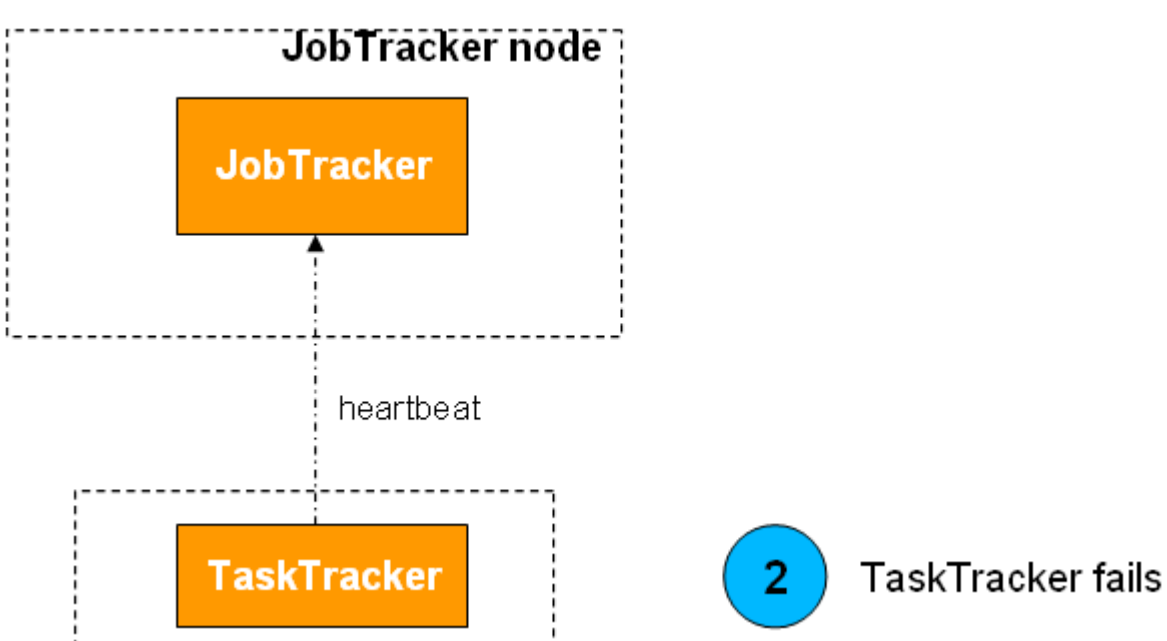
Fault tolerance

- **Task Failure**
  - If a child task fails, the child JVM reports to the TaskTracker before it exits. Attempt is marked failed, freeing up slot for another task.
  - If the child task hangs, it is killed. JobTracker reschedules the task on another machine.
  - If task continues to fail, job is failed.



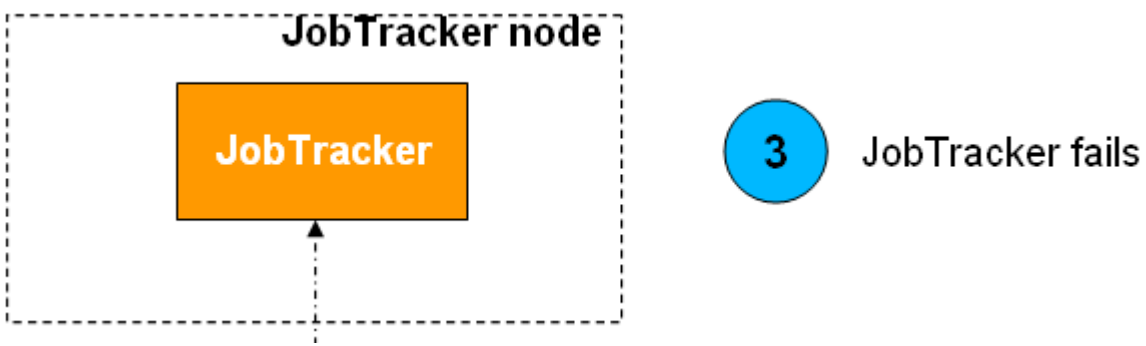
Fault tolerance

- **TaskTracker Failure**
  - JobTracker receives no heartbeat
  - Removes TaskTracker from pool of TaskTrackers to schedule tasks on.



Fault tolerance

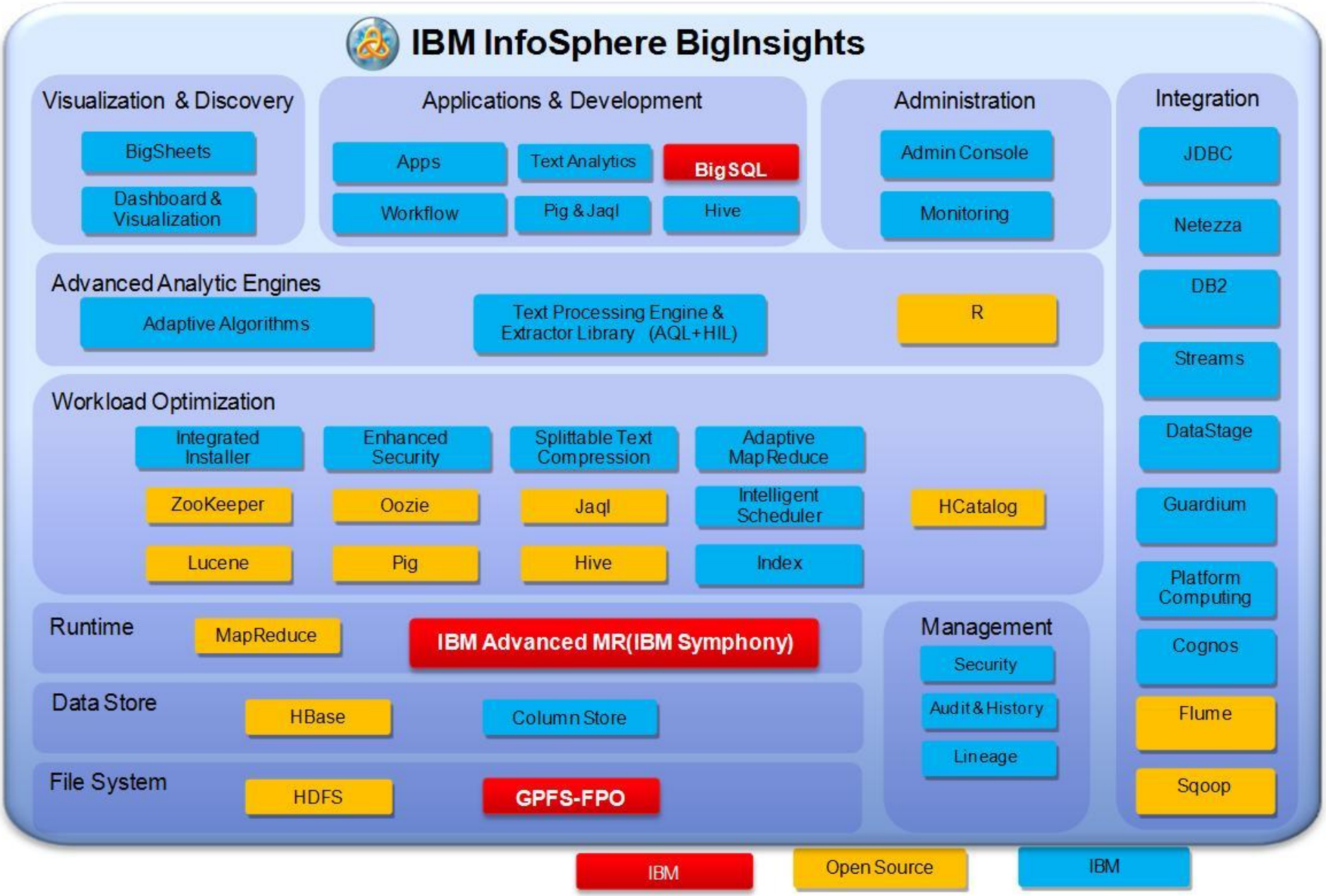
- **JobTracker Failure**
  - Singe point of failure. Job fails



ПЛАН

- Введение в Hadoop
- Архитектура Hadoop
- Парадигма map-reduce
- **Архитектура стека компонентов Hadoop**
- Когда стоит применять Hadoop?

BIGINSIGHTS –HADOOP STACK (IBM)



HADOOP STACK

- Avro™: A data serialization system.
- Cassandra™: A scalable multi-master database with no single points of failure.
- Chukwa™: A data collection system for managing large distributed systems.
- HBase™: A scalable, distributed database that supports structured data storage for large tables.
- Hive™: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- Mahout™: A Scalable machine learning and data mining library.
- Pig™: A high-level data-flow language and execution framework for parallel computation.
- ZooKeeper™: A high-performance coordination service for distributed applications.
- Oozie is a job coordinator and workflow manager for jobs executed in Hadoop, which can include non-MapReduce jobs

ПЛАН

- Введение в Hadoop
- Архитектура Hadoop
- Парадигма map-reduce
- Архитектура стека компонентов Hadoop
- **Когда стоит применять Hadoop?**

ТОЧЕЧНАЯ ОБРАБОТКА АКТИВНЫХ ДАННЫХ?

- Самый частый сценарий - это хранение «активных» данных —
  - информации о пользователях,
  - списка товаров,
  - комментариев к статьям
  - и т.д.
- всего, что может часто меняться. В этом случае мы хотим иметь возможность обрабатывать данные точечно — извлекать нужный объект по индексу, обрабатывать его и загружать обратно.
- Пусть очень много данных так что отдельная СУБД не справляется

ТОЧЕЧНАЯ ОБРАБОТКА АКТИВНЫХ ДАННЫХ?

- Не Hadoop
- Именно такой функционал предоставляет большинство СУБД (причём как реляционных, так и NoSQL).
- При масштабировании основная проблема здесь возникает с максимальным объёмом данных, хранимым в базе. Однако, даже если используемая СУБД не поддерживает распределённую структуру (IBM Warehouse, MySQL(Facebook)), проблема легко решается за счёт разбиения на секции на уровне приложения.
- Возможно использование специальных параллельных баз данных (IBM Netezza, Teradata, e.t.c.)

ПОТОКОВАЯ ОБРАБОТКА В РЕАЛЬНОМ ВРЕМЕНИ?

- Иногда, упор делается не на хранении, а на обработке данных.
- Пусть нам нужно в реальном времени получать твиты, анализировать их и выводить результат на динамически генерируемом графике.
- Пусть у нас миллионы ежедневных твитов

ПОТОКОВАЯ ОБРАБОТКА В РЕАЛЬНОМ ВРЕМЕНИ?

- Не Hadoop
- Объем не большая проблема. 1 млн. твитов - это не больше 140 миллионов символов или 280Мб при использовании кодировки UTF16.
- Проблема может возникнуть при анализе этих твитов в реальном времени.
- Большинство потоковых алгоритмов (будь то сентиментный анализ твитов, сбор куммулятивной статистики или онлайн машинное обучение) использует для своей работы небольшие, независимые друг от друга кусочки данных.
- Это позволяет легко распараллелить обработку, просто добавив больше вычислительных узлов и поставив перед ними балансировщик нагрузки.
  - В простейшем случае в качестве балансировщика может выступать брокер сообщений (такой как RabbitMQ или ZeroMQ),
  - в более сложных можно использовать готовые фреймворки для потоковой обработки, такие как Storm.
  - При этом основной код, непосредственно выполняющий обработку данных, практически не меняется по сравнению с односерверной версией.
- IBM Streams

ПАКЕТНАЯ ОБРАБОТКА ИСТОРИЧЕСКИХ ДАННЫХ?

- Представим, что у нас есть данные обо всех покупках в крупной сети супермаркетов за последние пол года.
- Мы хотим проанализировать эти данные: посчитать среднюю эффективность каждого супермаркета, эффект от проведённых акций, корреляцию между купленными товарами и множество других метрик.
- Как организовать работу с данными таким образом, чтобы вычисление этих параметров занимало резонное время?

ПАКЕТНАЯ ОБРАБОТКА ИСТОРИЧЕСКИХ ДАННЫХ?

- Hadoop
- Мы можем загрузить все данные в распределённую базу Oracle и работать с ними так же, как с активными.
  - Но в этом случае сервер приложений будет последовательно забирать данные из базы и последовательно обрабатывать каждую запись, что крайне неэффективно.
- Мы также можем настроить конвеер для потоковой обработки, распределяя нагрузку между серверами приложений.
  - Но рано или поздно мы упрёмся в канал связи между обрабатывающими узлами и узлами данных.
- Единственный способ распределить нагрузку и не переполнить канал связи — *минимизировать передвижение данных* между узлами.
- А для этого необходимо максимум вычислений производить *локально* на тех машинах, где лежат обрабатываемые данные. Именно **принцип локальности данных** лежит в основе парадигмы MapReduce и всего Hadoop.

REFERENCES

1. Chuck Lam. Hadoop in Action
2. Tom White. Hadoop: The Definitive Guide, 3rd Edition
3. IBM Course: “Hadoop Fundamentals” (BD001)
   - http://bigdatauniversity.com/