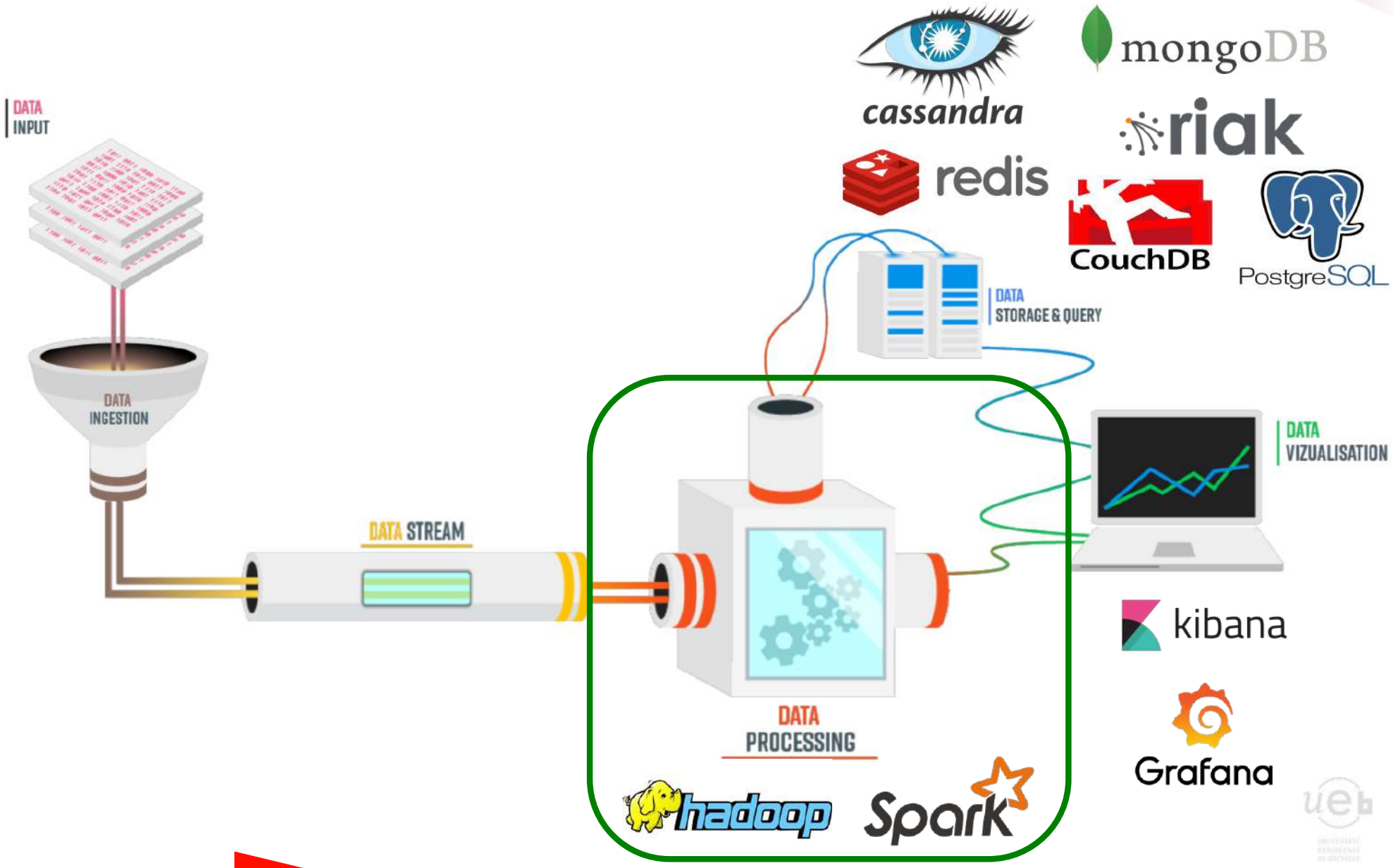
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Big Data Processing with MapReduce

The Big Data pipeline



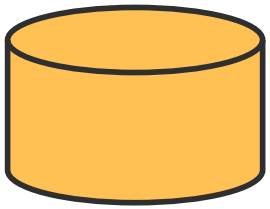
Outline

- Google File System
- MapReduce programming model
- Examples
- MapReduce system architecture
- Apache Hadoop
- Limitations

File systems overview

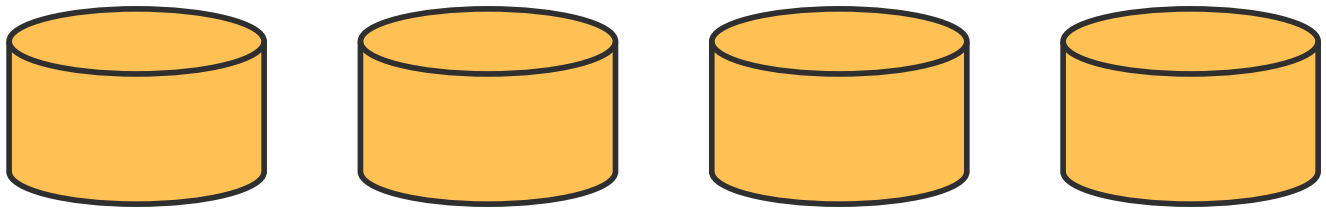
- Permanently stores data
  - Usually on top of a lower-level physical storage medium
- Divided into logical units called **"files"**
  - Addressable by a filename ("`foo.txt`")
  - Usually supports hierarchical nesting (directories)
- A file path = relative (or absolute) directory + file name

```
/dir1/dir2/foo.txt
```



Distributed file systems

- Support access to files on remote servers
- Must support **concurrency**
  - Locking, who “wins” concurrent writes, etc.
  - Must gracefully handle dropped connections
- Can offer support for replication and local caching



Motivation

- Google needed a good distributed file system
  - Redundant storage of massive amounts of data on cheap and unreliable computers
- Why not use an existing file system?
  - Google's problems are different from anyone else's
    - Different workload and design priorities
  - **GFS** is designed for Google apps and workloads
  - Google apps are designed for **GFS**

Assumptions: environment

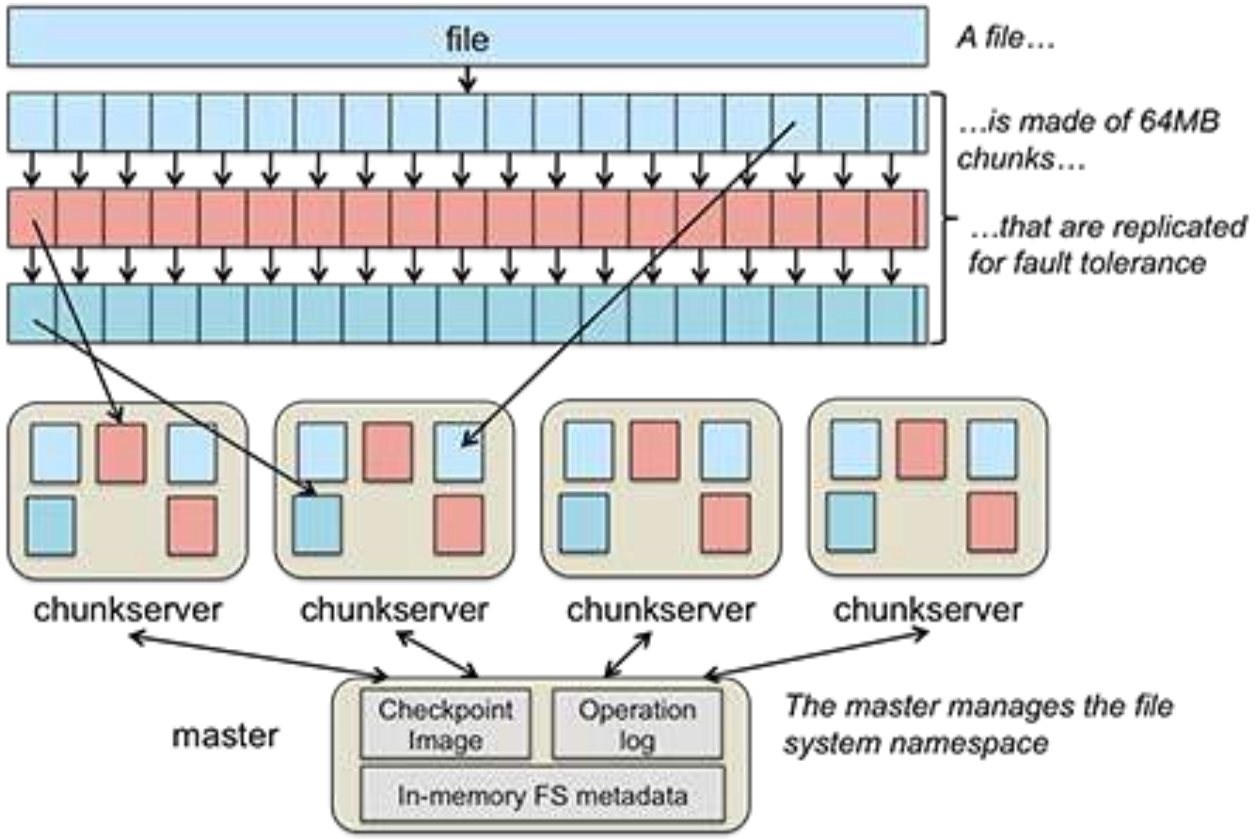
- Commodity hardware
  - Inexpensive

- High component failure rates
  - Inexpensive commodity components fail all the time
  - The norm rather than the exception

- Huge storage needs
  - Must support PBs of space

Google File Ssystem design principles

- Files stored as **chunks**
  - Fixed size (64MB)
- **Replication**
  - Each chunk replicated across 3+ chunkservers



- **No data caching:** little benefit due to large data sets

Spread the work over many machines

Good news: “easy” parallelization

- Reading the entire web with 1,000 machines ⇒ less than 3 hours

Bad news: programming work

- Communication and coordination
- Debugging
- Fault tolerance
- Management and monitoring
- Optimization

Worse news: repeat this for every problem

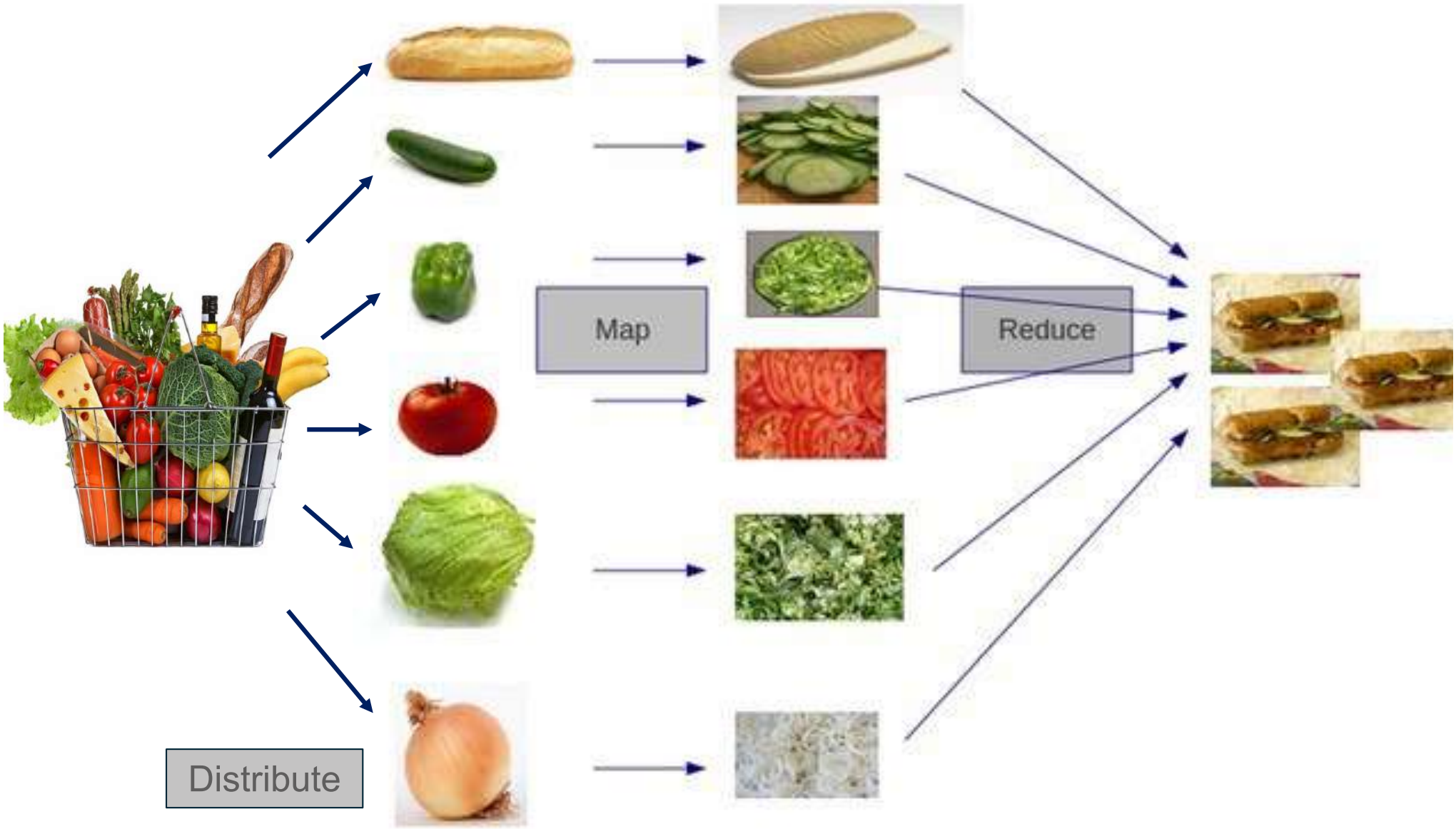
What is needed?

A simple programming model that applies to many data-intensive computing problems

Approach: hide messy details in a runtime library

- Automatic parallelization
- Load balancing
- Network and disk transfer optimization
- Handling of machine failures
- Robustness
- Improvements to core library benefit all users of library!

Sucha a model is… MapReduce



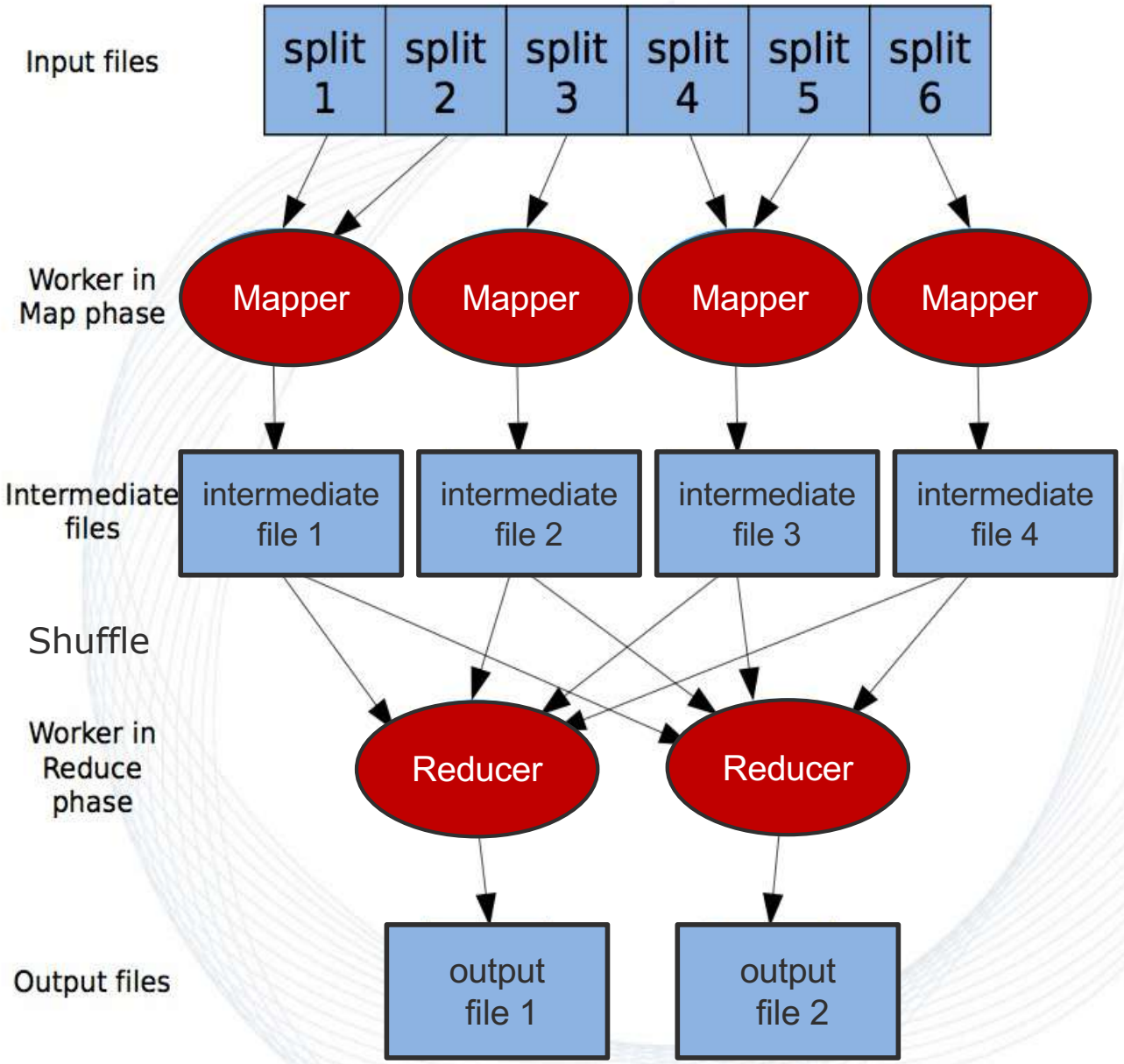
Such a model is… MapReduce

Typical problem solved by MapReduce

- Read a lot of data
- **Map:** extract something interesting from each record
- Shuffle (sort)
- **Reduce:** aggregate, summarize, filter or transform
- Write the results

Outline stays the same, **map** and **reduce** change to fit the problem

MapReduce at a glance



Everything is key / value

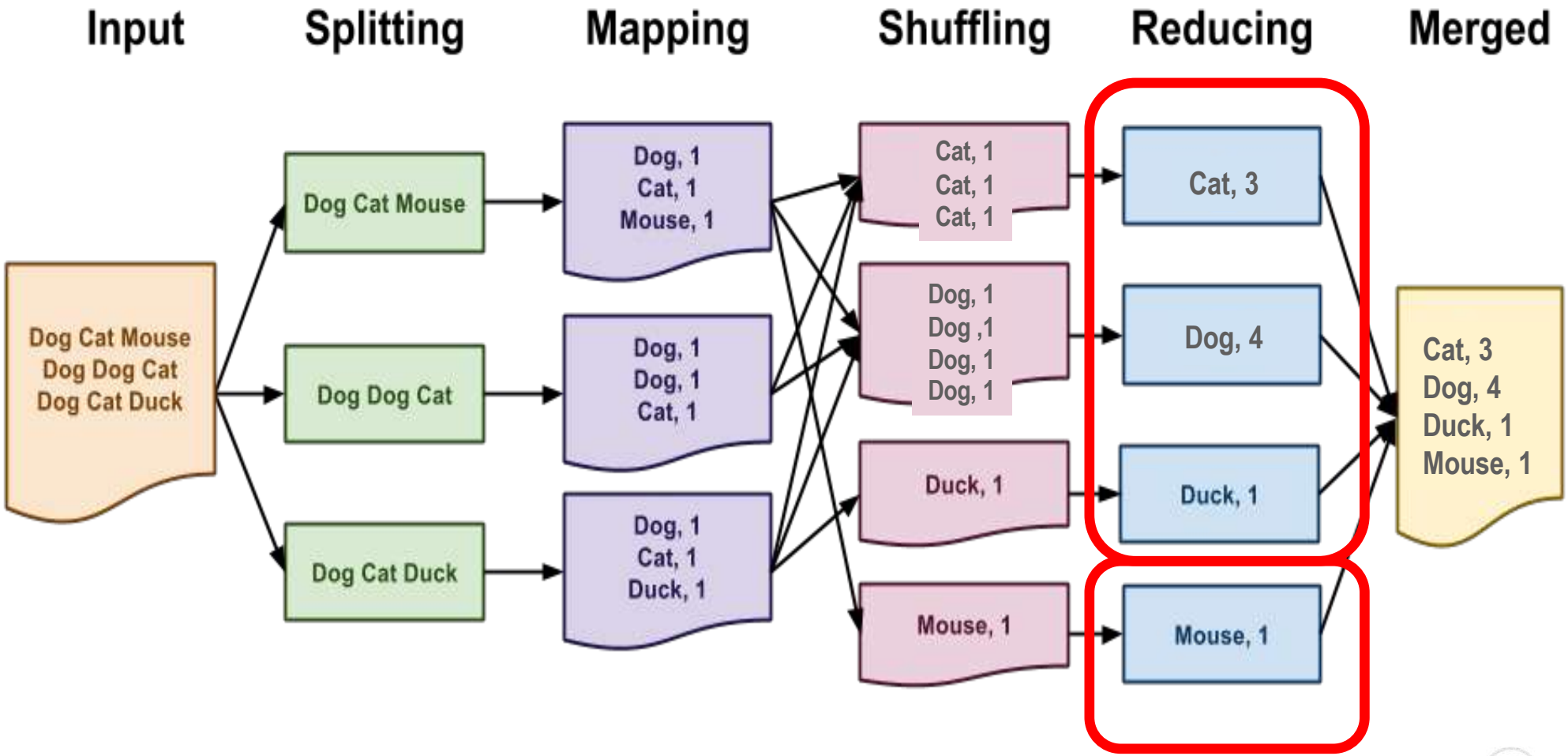
It is inspired by the Map and Reduce functions from functional programming

Users implement the interface of two primary functions

- **map(k, v) → <k', v'>***
- **reduce(k', <v'>*) → <k', v''>***

All **v'** with same **k'** are reduced together, and processed in **v'** order

Example 1: word count

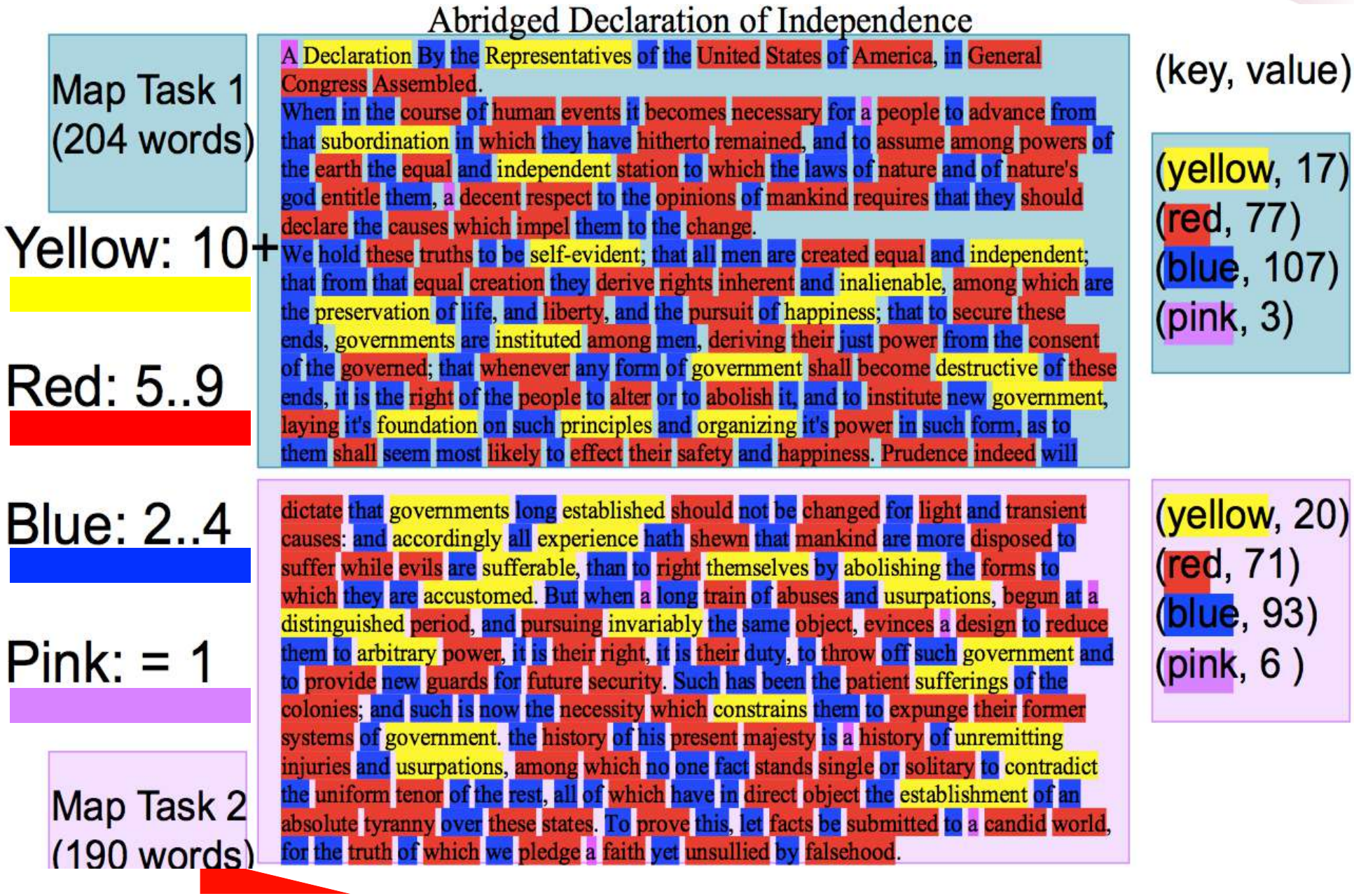


Example 2: word length count

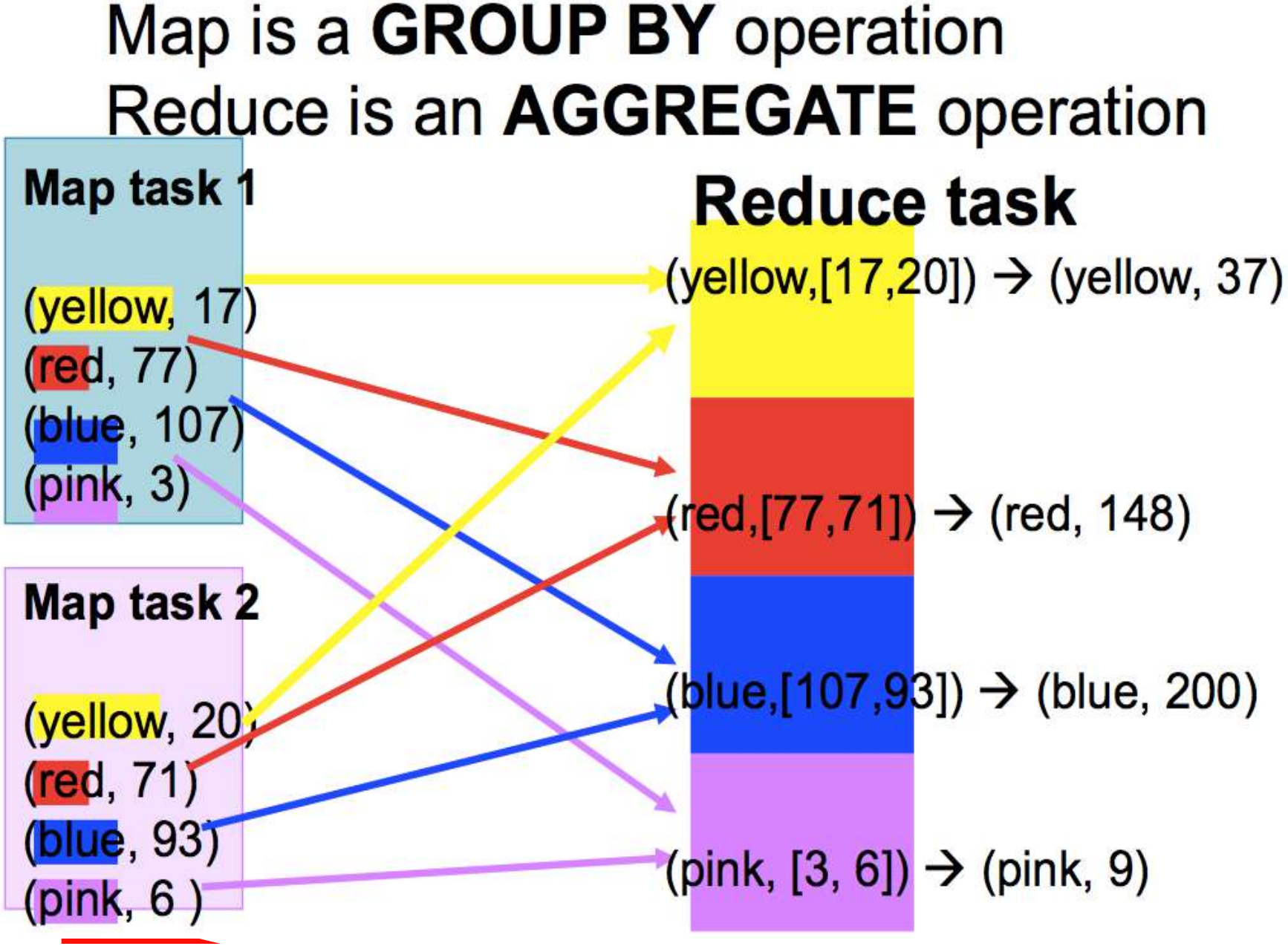
Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.
When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.
We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

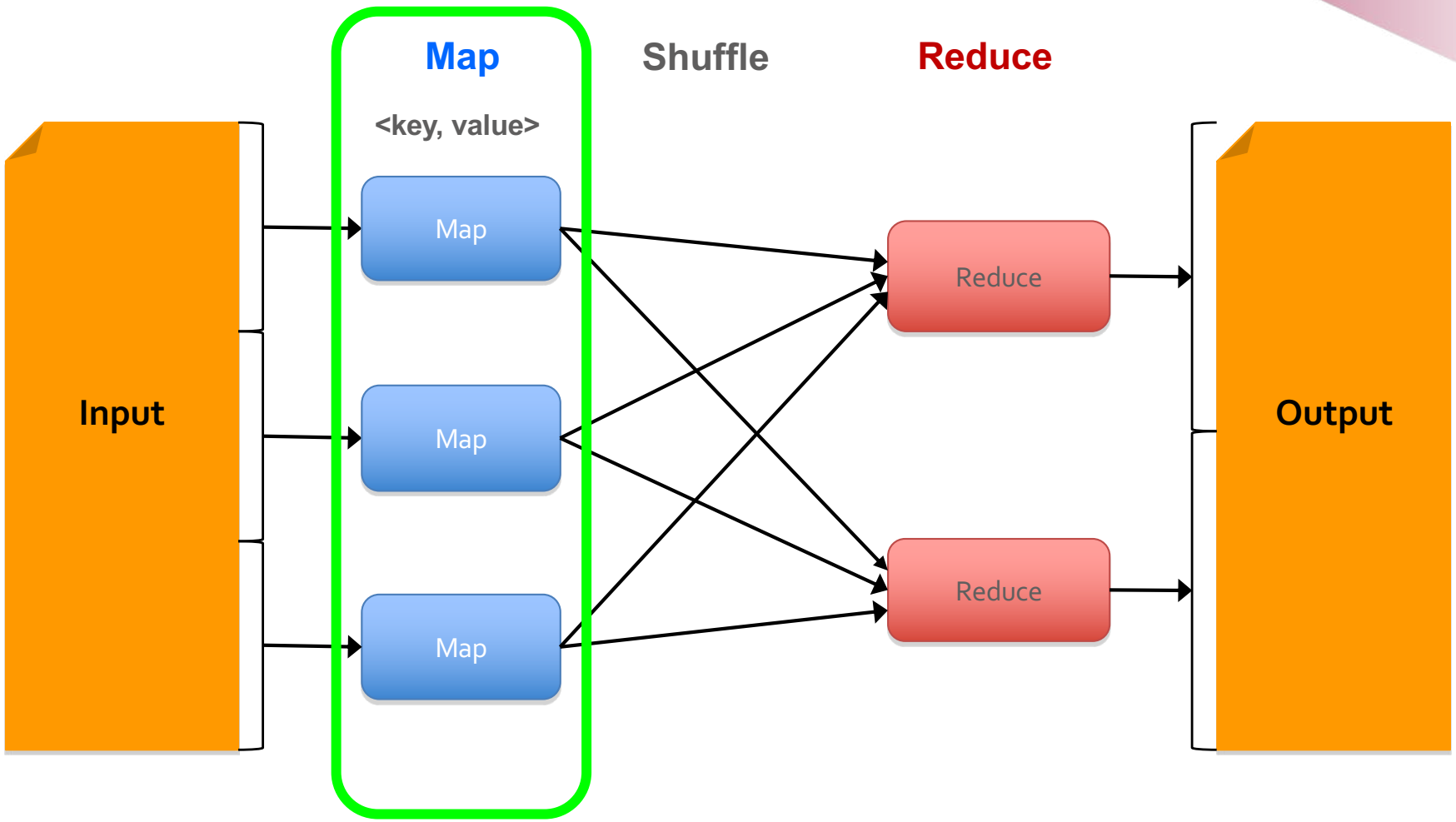
Example 2: word length count



Example 2: word length count



Zoom on the Map phase



Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs: (filename, line). Then:

map(k, v) → <k', v'>*

Combiner

For certain types of reduce functions (commutative and associative), we can execute the reduce function within the mappers

- SUM, COUNT, MAX, MIN ...

Example: word count

- Without Combiner

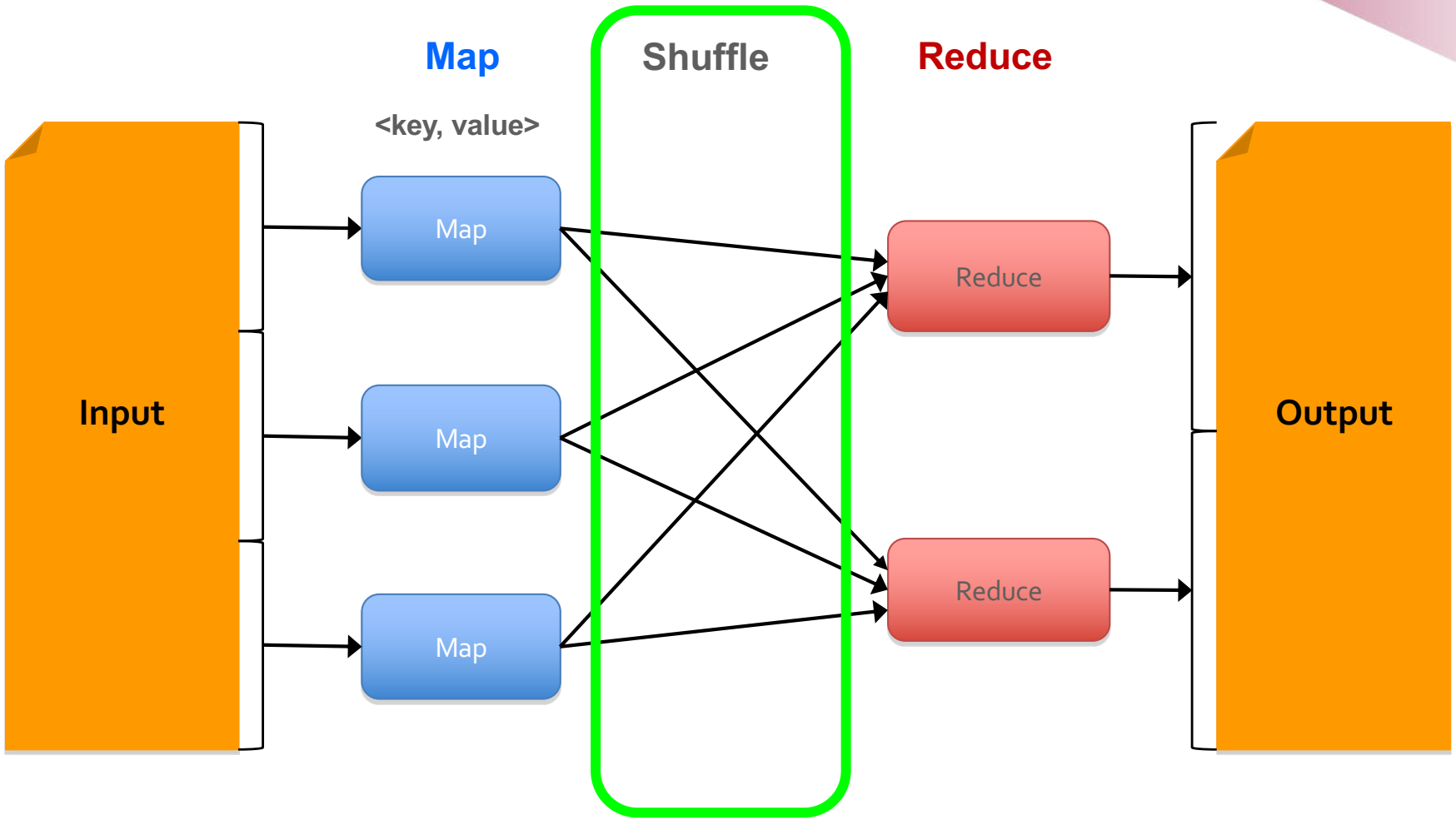
  <docid, {list of words}> => N records <word, 1>

- With Combiner

  <docid, {list of words}> => <word, N>

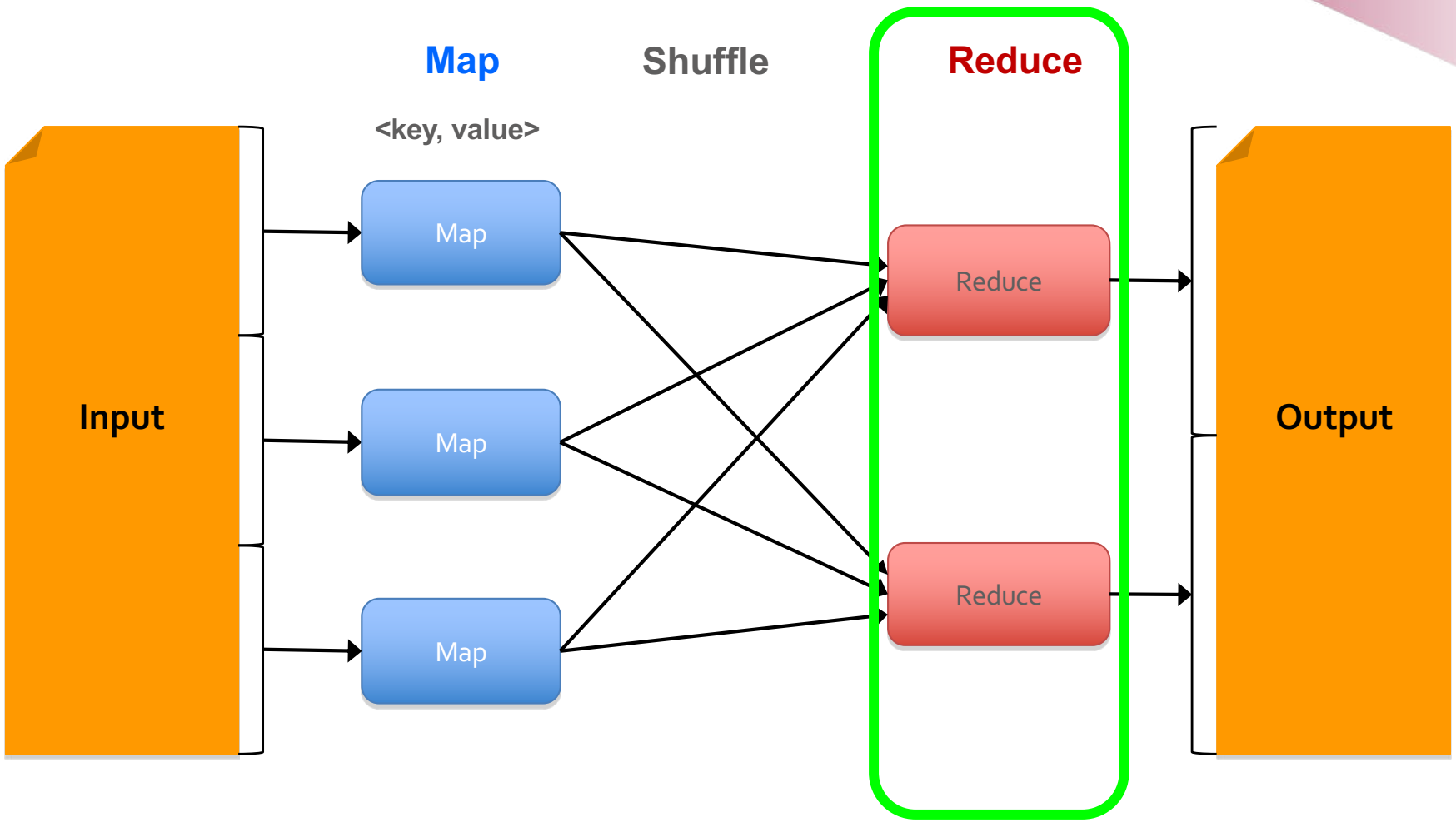
- N, the number of times the word appears in the mapper

Zoom on the Shuffle phase



After the map phase is over, all the intermediate values for a given output key are **shuffled (sorted)** together into a list

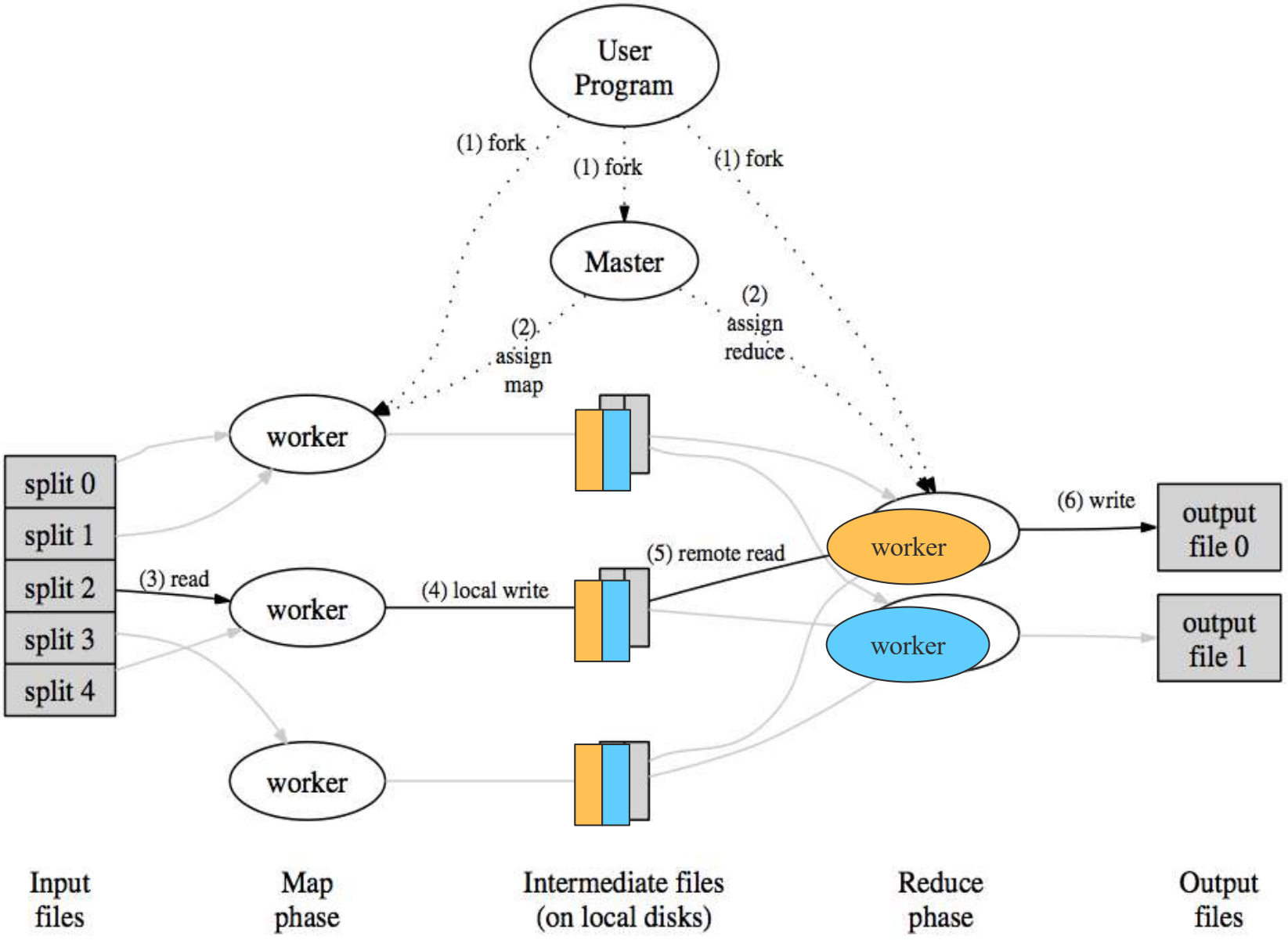
Zoom on the Reduce phase



reduce() combines those intermediate values into one or more final values per key (usually only one)

reduce(k', <v'>*) → <k', v''>*

Architectural overview



One master, many workers

- Master partitions input file into M splits, by key
- Master creates a map task for each split
- Master assigns the M map tasks to workers (=servers), keeps track of their progress
- Workers compute and write their output to local disk, partitioned into R regions
  - using a partition function: e.g., hash(key) mod R
- Master assigns R reduce tasks to the workers
- Reduce workers read corresponding regions from the map workers' local disks, compute and write results

Often: 1 split = 64 MB

why ?

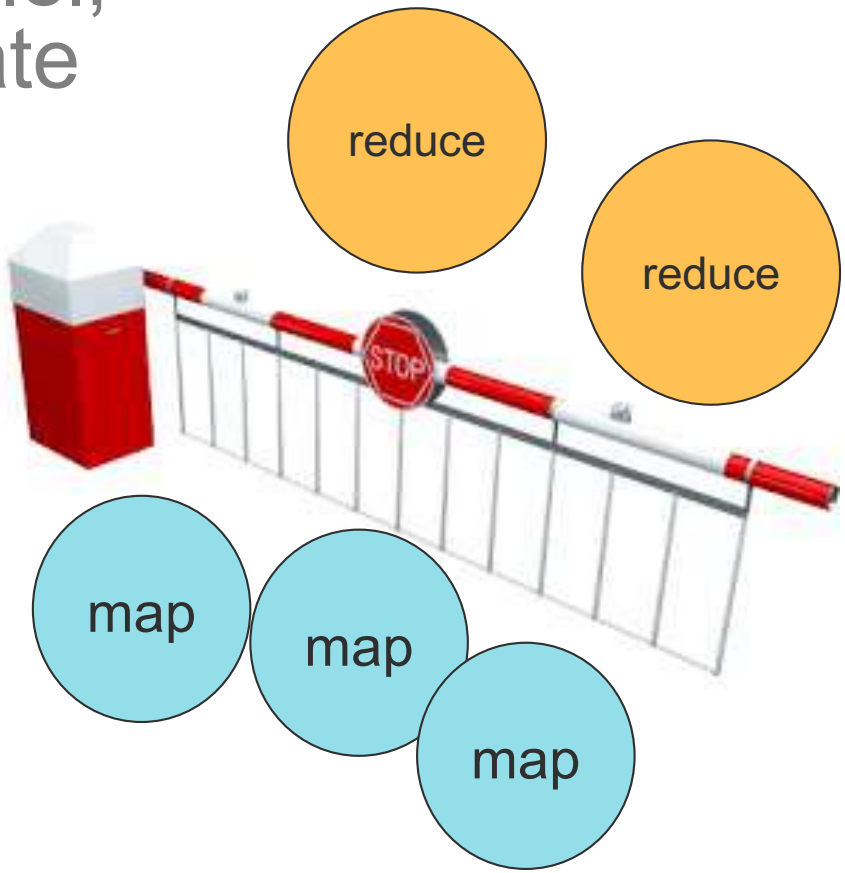
M tasks = 200,000; R tasks = 4,000; workers = 2,000

Parallelism

map() functions run in parallel, creating different intermediate values from the input data

reduce() functions also run parallel, each working on a different output key

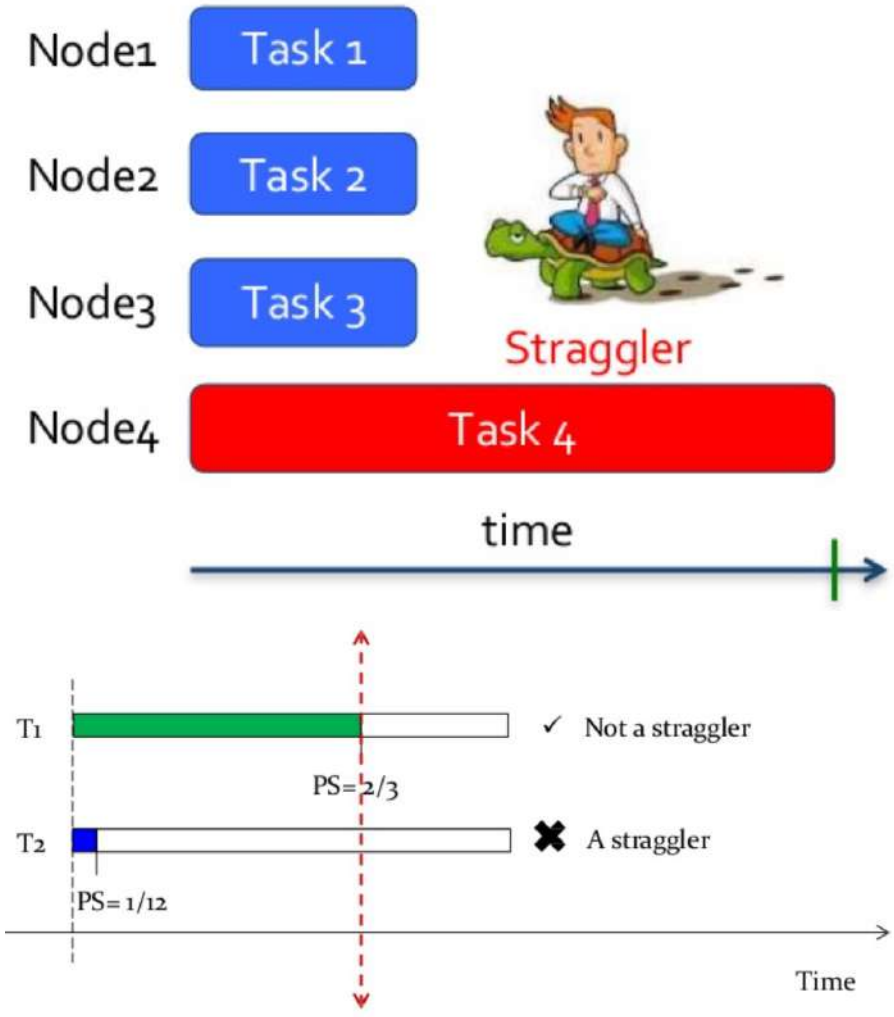
All values are processed independently



Bottleneck: reduce phase can't start until map phase is completely finished

Stragglers

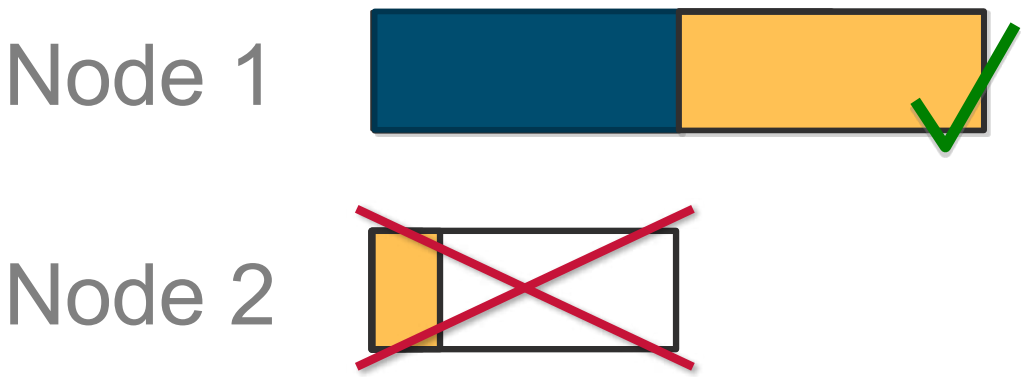
- Nodes that take unusually long time to complete one of the tasks
- Reasons
  - Bad disk forces frequent correctable errors (30MB/s to 1MB/s)
  - The cluster scheduler has scheduled other tasks on that machine
- Stragglers are a main reason for slowdown: a MR job is dominated by the slowest task



Solution: backup tasks - pre-emptive backup execution of the last few remaining in-progress tasks

Speculation in MapReduce

Slow nodes (stragglers) → run backup tasks



MapReduce widely used at Google

- distributed grep
- distributed sort
- term-vector per host
- document clustering
- machine learning
- web access log stats
- web link-graph reversal
- inverted index construction
- statistical machine translation

MapReduce widely used at Google

Google Official Blog

Insights from Googlers into our products, technology, and the Google culture

Sorting 1PB with MapReduce

Posted: Friday, November 21, 2008

At Google we are fanatical about organizing the world's information. As a result, we spend a lot of time finding better ways to sort information using MapReduce, a key component of our software infrastructure that allows us to run multiple processes simultaneously. MapReduce is a perfect solution for many of the computations we run daily, due in large part to its simplicity, applicability to a wide range of real-world computing tasks, and natural translation to highly scalable distributed implementations that harness the power of thousands of computers.

In our sorting experiments we have followed the rules of a standard terabyte (TB) sort benchmark. Standardized experiments help us understand and compare the benefits of various technologies and also add a competitive spirit. You can think of it as an Olympic event for computations. By pushing the boundaries of these types of programs, we learn about the limitations of current technologies as well as the lessons useful in designing next generation computing platforms. This, in turn, should help everyone have faster access to higher-quality information.

We are ... able to sort 1TB (stored on the Google File System as 10 billion 100-byte records in uncompressed text files) on 1,0... computers in 68 seconds. By comparison, the previous 1TB sorting record is 209 seconds on 910 computers.

200 seconds

Sometimes ... than a terabyte, so we were curious to find out what happens when you sort more and gave one petabyte (PB) a try. One petabyte is a thousand terabytes, or, to put this amount in perspective, it is 12 times the amount of archived web data in the U.S. Library of Congress as of May 2008. In comparison, consider that the aggregate size of data processed by all instances of MapReduce at Google was on average 20PB per day in January 2008.

It took six hours and two minutes to sort 1PB (10 trillion 100-byte records) on 4,000 computers. We're not aware of

Google

google.com/+google

News and updates on Google's products, technology and more

Suivre +1

+ 12 650 192

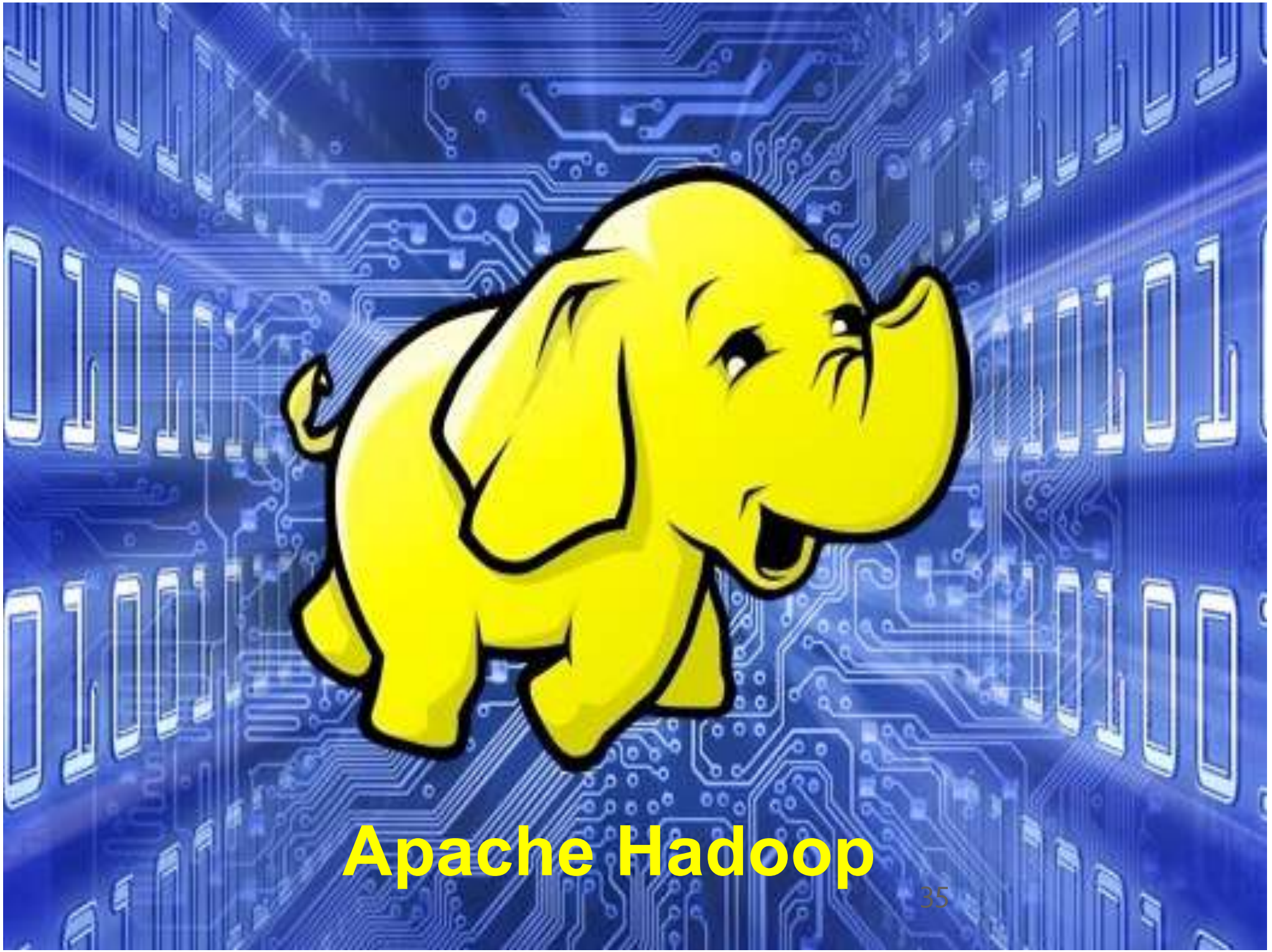
Labels

Archive

Feed

MapReduce limitations

- Not efficient for real-time processing
- Very limited queries
  - Difficult to write more complex tasks
  - Need multiple map-reduce operations
  - Solutions: declarative query languages ☺
- No support for iterative processing
- Barrier between Map and Reduce


Apache Hadoop

What is Hadoop?

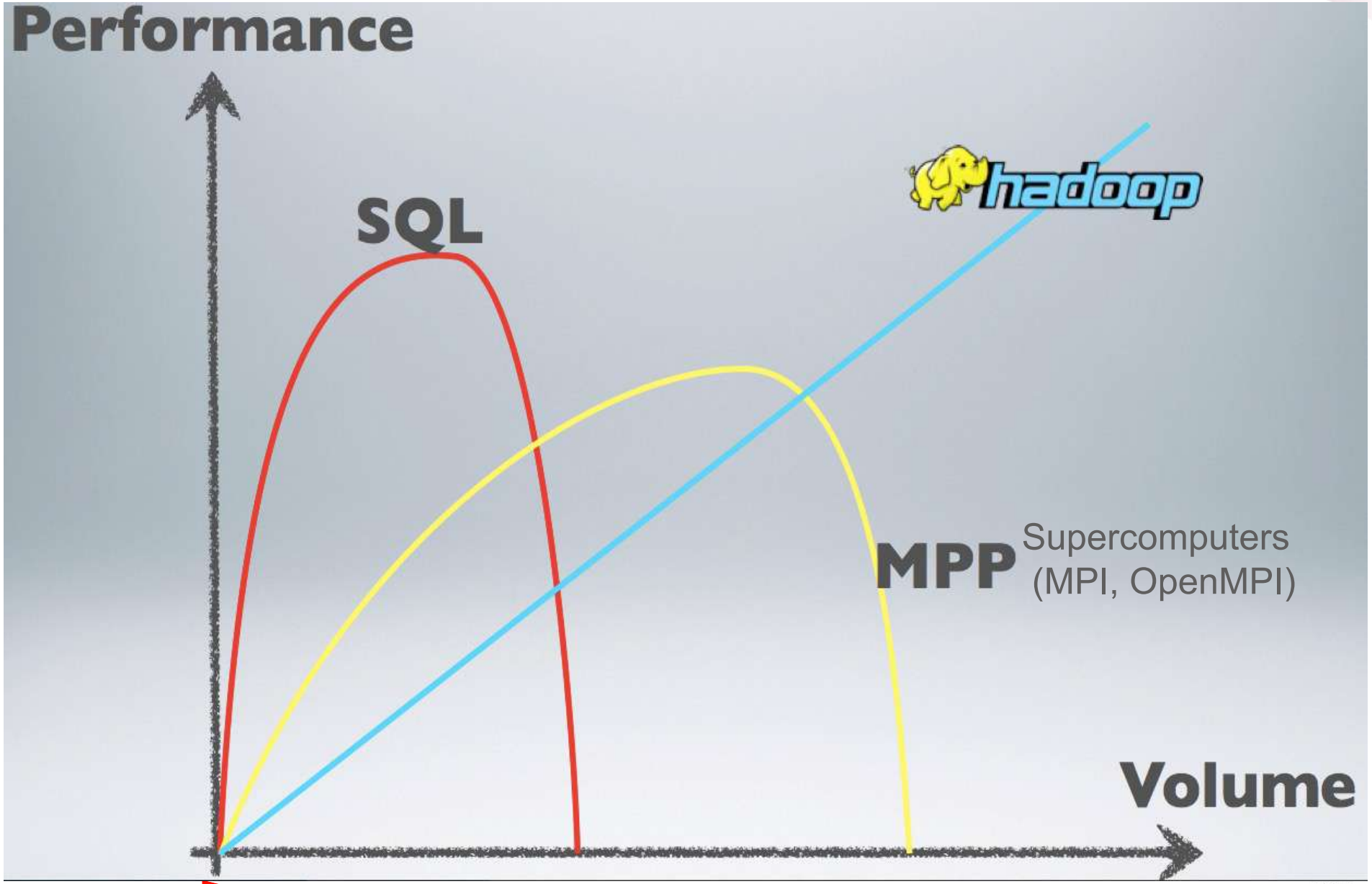
Open source implementation of MapReduce

- Top-level Apache project
- Developed in Java

Platform for data storage and processing

- Scalable
- Fault tolerant
- Distributed
- Any type of complex data

Why?


Performance
SQL
hadoop
MPP
Supercomputers
(MPI, OpenMPI)
Volume

The Hadoop 1.0 stack

HBase
(Columnar Database)

Pig
(Data Flow)

Hive
(SQL)

Cascading
(Java)

MapReduce
(Distributed Programming Framework)

HDFS
(Hadoop Distributed File System)

Hadoop MapReduce

Master-Slave architecture

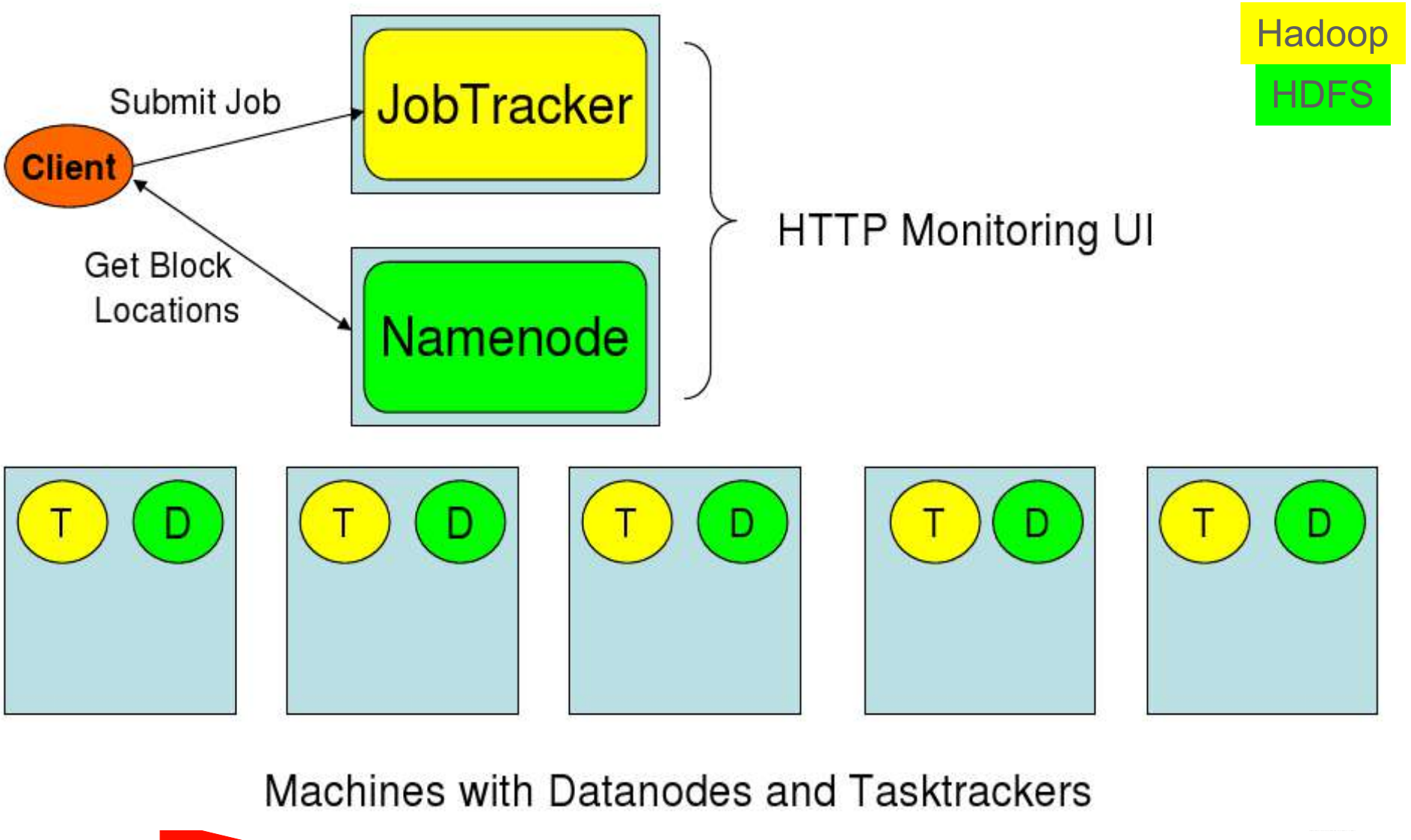
Job Tracker *(Master)*

- Accepts MapReduce jobs submitted by users
- Assigns Map and Reduce tasks to TaskTrackers
- Re-executes tasks upon failure

Task Trackers *(Workers)*

- Run Map and Reduce tasks
- Fixed number of map and reduce **slots** allocated by the administrator to each node according to its resources
  - Cannot run more map tasks than map slots at any given moment, even if no reduce tasks are running

Putting everything together: HDFS and Hadoop deployment



Hadoop MapReduce Client

- Define
  - Mapper class
  - Reducer class
  - “Launching” program

- Language support
  - Java
  - C++
  - Python

Word Count example in Hadoop

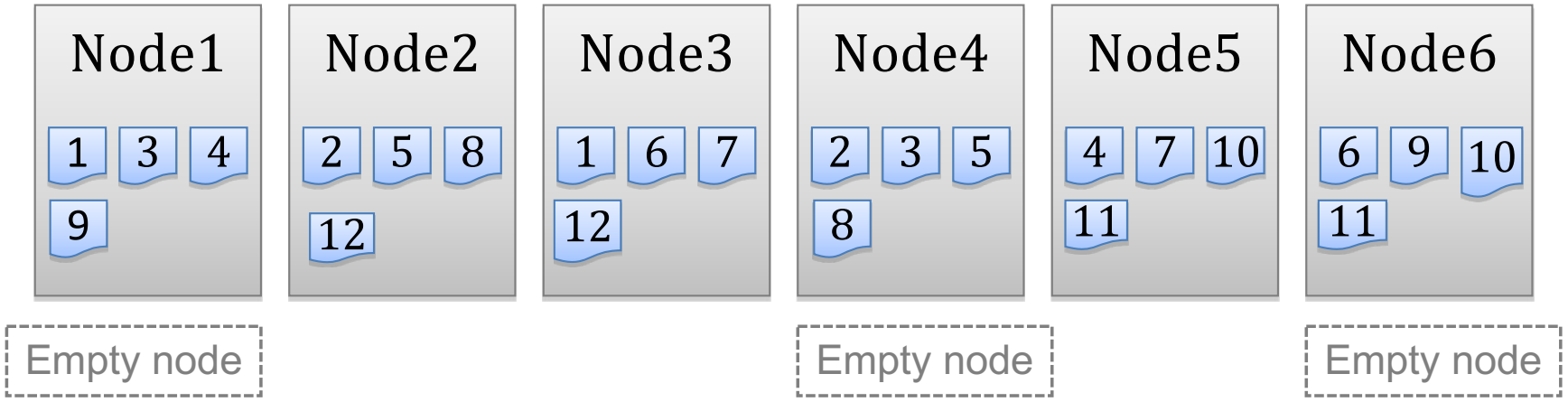
```
public void map(WritableComparable key, Writable value, OutputCollector output, Reporter reporter) throws
      IOException {

   String line = ((UTF8)value).toString();

   StringTokenizer itr = new StringTokenizer(line);

   while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
           output.collect(word, one);
   }
}


public void reduce(WritableComparable key, Iterator values, OutputCollector output,Reporter reporter) throws
      IOException {
   int sum = 0;

   while (values.hasNext()) {
     sum += ((IntWritable) values.next()).get();
    }

    output.collect(key, new IntWritable(sum));
}
```

Data locality

- Data locality is exposed in the map task scheduling
  - Tasks are assigned to nodes where the needed data chunks are already present (in HDFS)
- JobTracker schedules map tasks considering
  - Node-awareness
  - Rack-awareness
  - Non-local map tasks

Open Issues: data locality

Data locality is crucial for Hadoop's performance

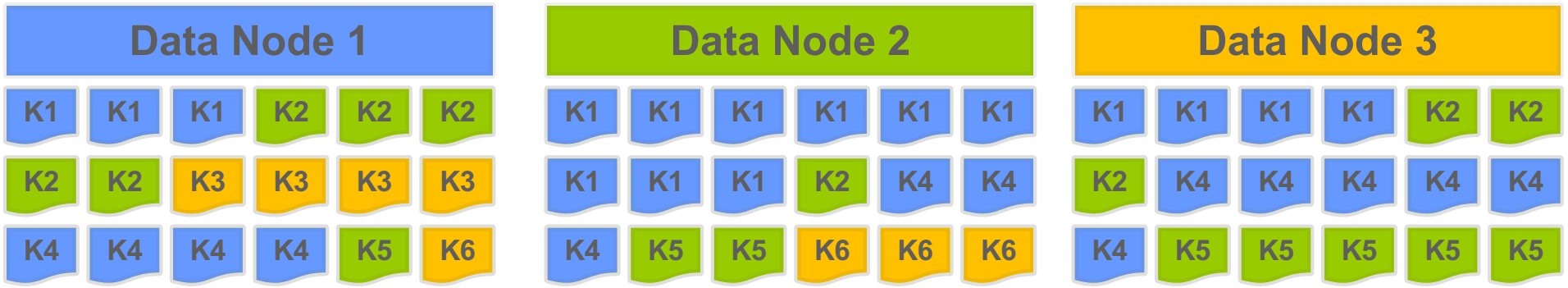


The simplicity of map tasks scheduling leads to **non-local maps execution** (25%)

Open Issues – data skew

- The current Hadoop hash partitioning for Reduce works well when the keys are equally frequent and uniformly stored in the data nodes
- In the presence of partitioning skew
  - Variation in Intermediate Keys frequencies
  - Variation in Intermediate Keys distribution among different Data Nodes
- Native blindly hash-partitioning is inadequate and will lead to
  - Network congestion
  - Unfairness in reducers inputs → Reduce computation skew
  - Performance degradation

Open Issues – data skew



Hash code: (Intermediate-key) Modulo ReduceID

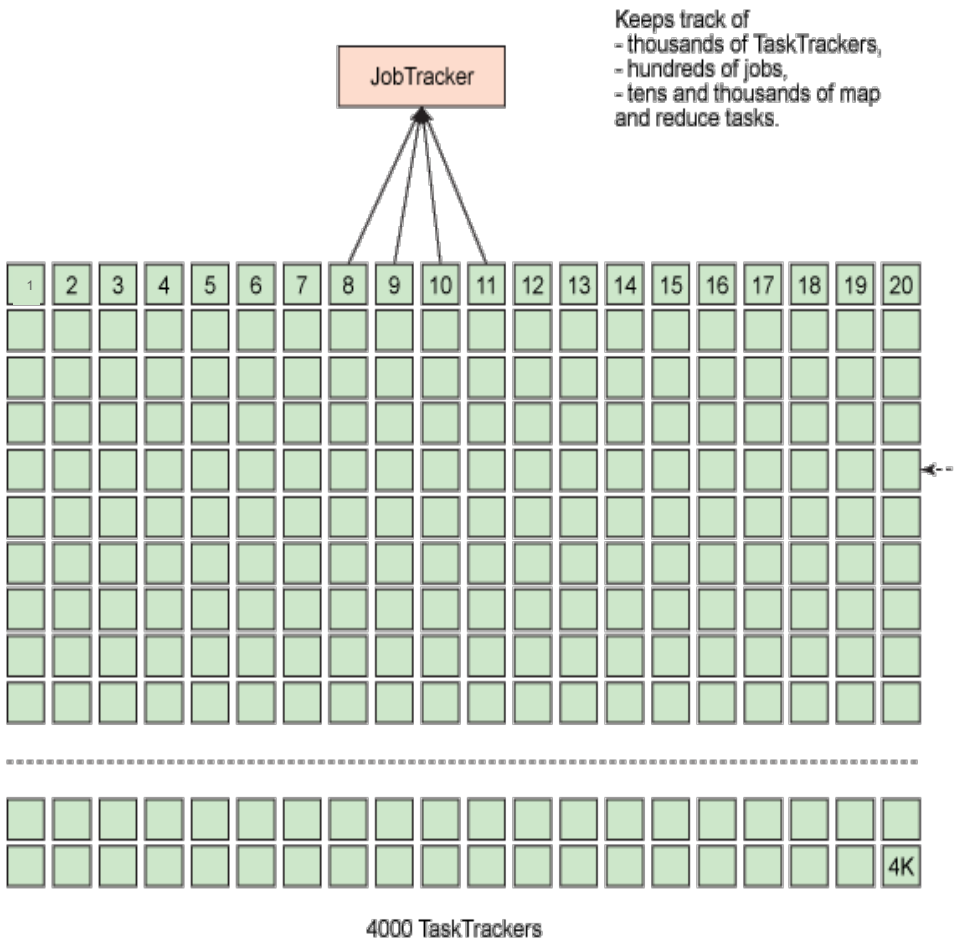
	Data Node 1	Data Node 2	Data Node 3	
Total Out Data Transfer	11	15	18	Total 44/54
Reduce Input	29	17	8	

Open Issues – single JobTracker

JobTracker has **two distinct responsibilities**

- Cluster resource management
- Task coordination

Scalability bottleneck caused by having a single JobTracker



Limitations of Hadoop v1

- Due to this scalability issue, **several smaller and less-powerful clusters** had to be created and maintained

- **Slots**: harm the cluster utilization because when all map slots are taken (and we still want more), we cannot use any reduce slots, even if they are available, or vice versa

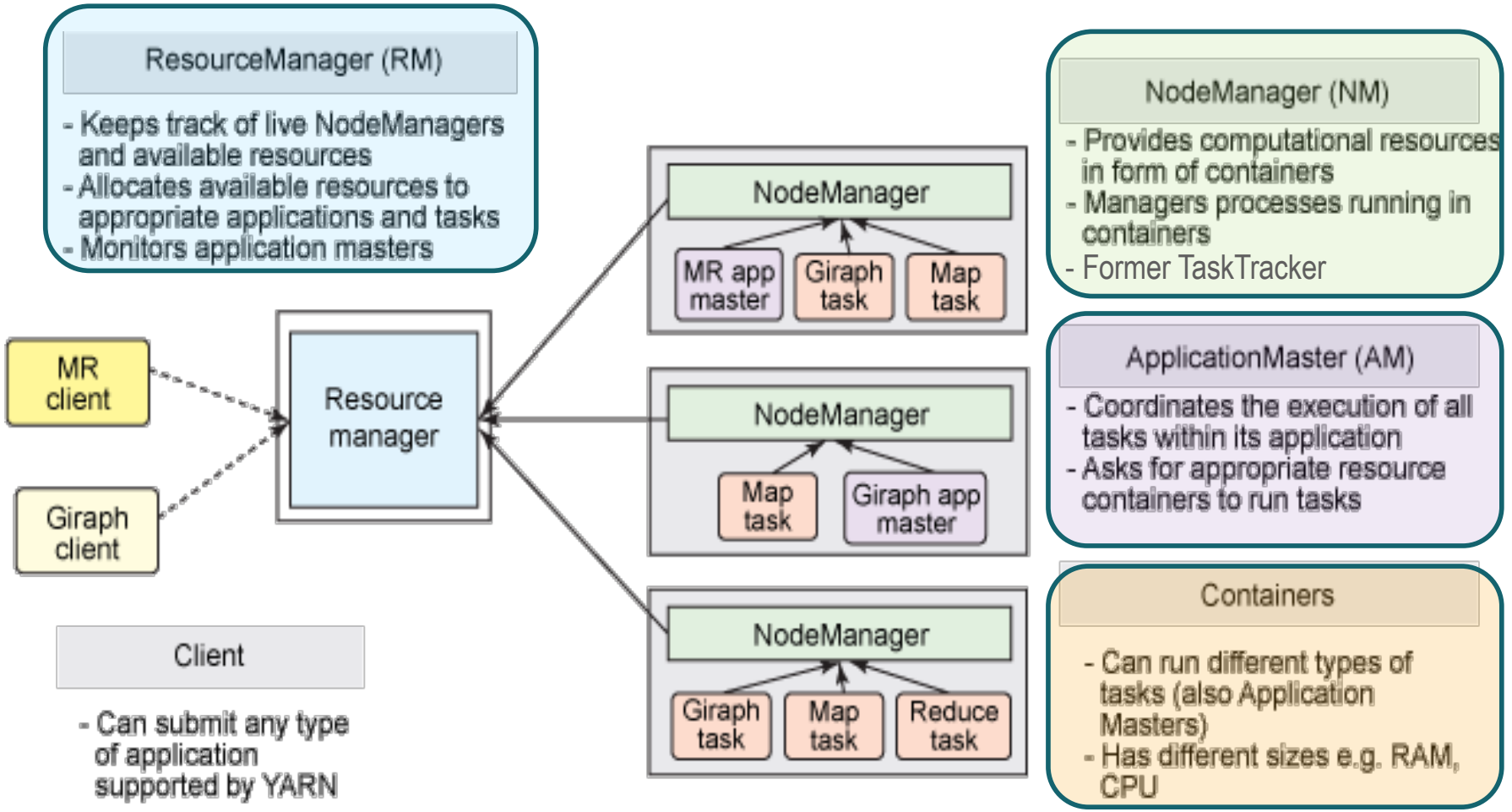
- Hadoop was designed to run **MapReduce jobs only**
  - Need to support other programming paradigms besides MapReduce

Idea

Reduce the responsibilities of the single JobTracker and **delegate** some of them to the many TaskTrackers

- New design by separating the responsibilities of the JobTracker into two distinct processes
  - **Resource Manager -** tracks live nodes and available resources in the cluster and assigning tasks to them
  - **Application Master** - for each submitted job, dedicated and short-living process controls the execution of tasks within that job only
- Thus, the coordination of a job's life cycle is spread across all of the available machines in the cluster:
  - More jobs can run in parallel, scalability dramatically increased

YARN: The next generation of Hadoop's compute platform



One cluster that can run any distributed application

Running Hadoop

Multiple options:

- On your local machine
  - **Standalone:** single Java process (good for debugging)
  - **Pseudo distributed:** each Hadoop daemon runs in a separate Java process
- Fully distributed
  - Clusters ranging from a few nodes to extremely large clusters with thousands of nodes

Bibliography

- Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM 51, 1 (January 2008), 107-113.*
- Sherif Sakr, Anna Liu, and Ayman G. Fayoumi. 2013. The family of mapreduce and large-scale data processing systems. *ACM Comput. Surv. 46, 1, Article 11 (July 2013), 44 pages*
- http://hadoop.apache.org/core/docs/current/api/
- http://hadoop.apache.org/core/docs/current/hdfs_design.html