# Hadoop Distributed File System (HDFS)



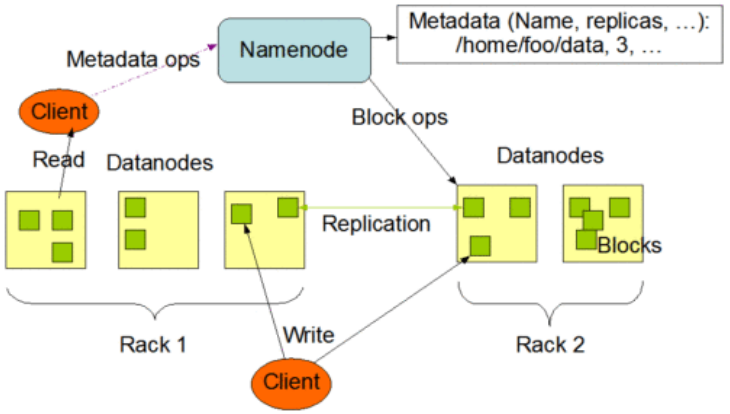
**Dr. Rajiv Misra**
**Dept. of Computer Science & Engg.**
**Indian Institute of Technology Patna**
**rajivm@iitp.ac.in**

# Preface

## Content of this Lecture:

- In this lecture, we will discuss design goals of HDFS, the read/write process to HDFS, the main configuration tuning parameters to control HDFS performance and robustness.





# Introduction

- Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm.

- An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data.

- A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! span 25,000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations worldwide report using Hadoop.

# Introduction

- Hadoop is an **Apache project**; all components are available via the Apache open source license.

- **Yahoo! has developed and contributed to 80%** of the core of Hadoop (**HDFS and MapReduce**).

- **HBase** was originally developed at **Powerset**, now a **department at Microsoft**.
- **Hive** was originated and developed at **Facebook**.
- **Pig, ZooKeeper, and Chukwa** were originated and developed at **Yahoo!**
- **Avro** was originated at **Yahoo!** and is being co-developed with **Cloudera**.

# Hadoop Project Components

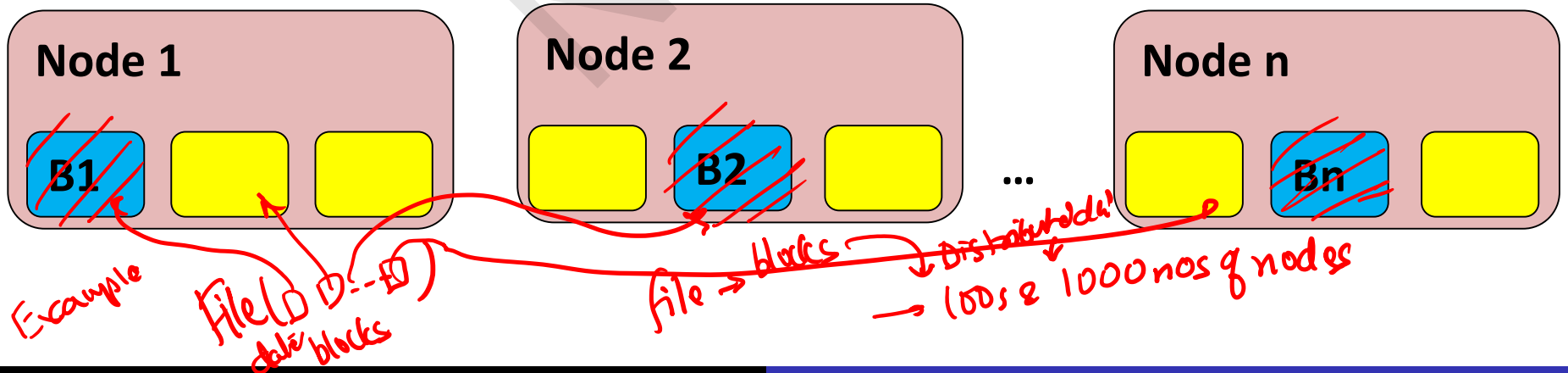
| | |
|---|---|
| HDFS | Distributed file system |
| MapReduce | Distributed computation framework |
| HBase | Column-oriented table service |
| Pig | Dataflow language and parallel execution framework |
| Hive | Data warehouse infrastructure |
| ZooKeeper | Distributed coordination service |
| Chukwa | System for collecting management data |
| Avro | Data serialization system |

# HDFS Design Concepts

- **Scalable distributed filesystem:** So essentially, as you add disks you get scalable performance. And as you add more, you're adding a lot of disks, and that scales out the performance.

- **Distributed data on local disks on several nodes.**

- **Low cost commodity hardware:** A lot of performance out of it because you're aggregating performance.



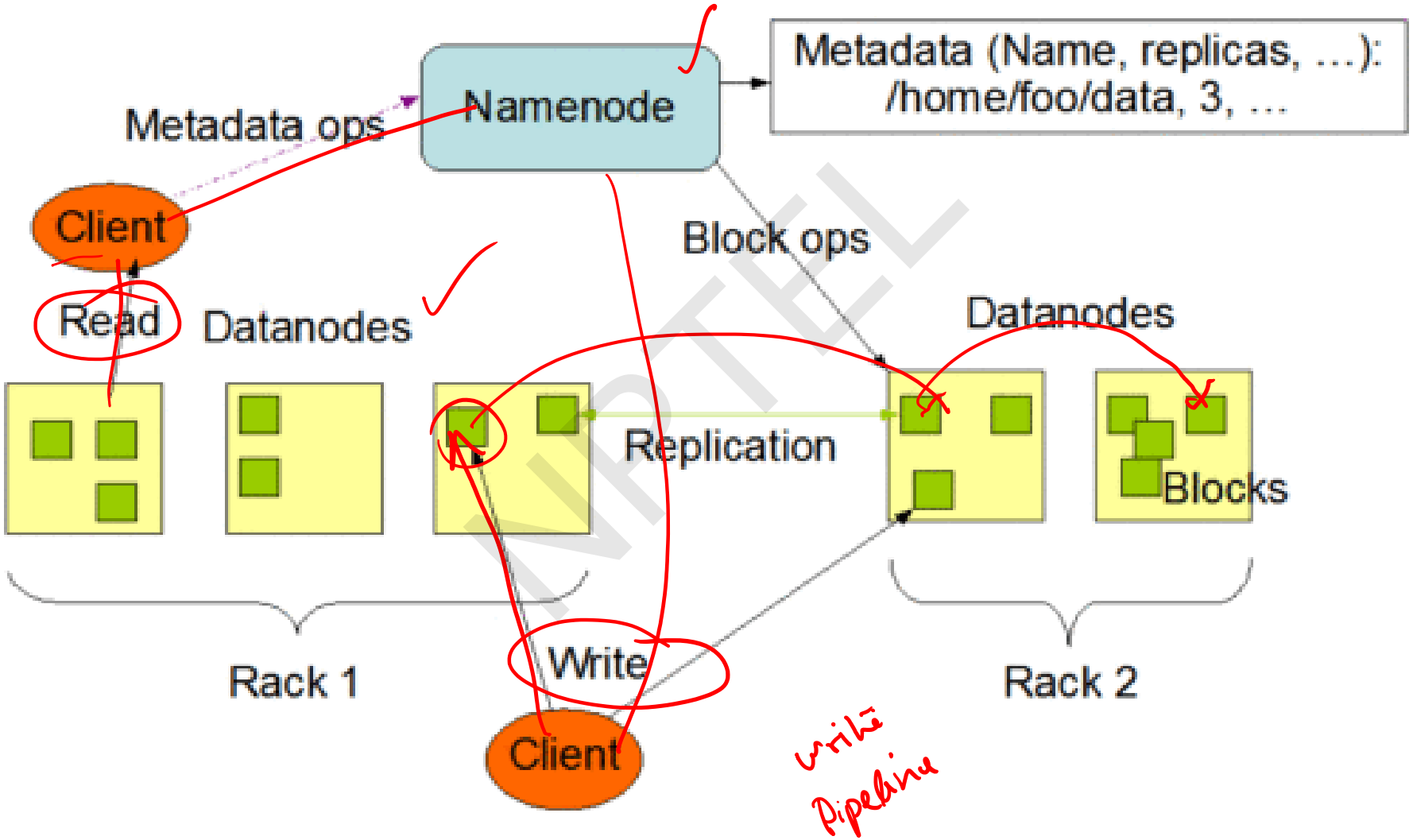
# HDFS Design Goals

- **Hundreds/Thousands of nodes and disks:**
  - It means there's a higher probability of hardware failure. So the design needs to handle node/disk failures.
- **Portability across heterogeneous hardware/software:**
  - Implementation across lots of different kinds of hardware and software.
- **Handle large data sets:**
  - Need to handle terabytes to petabytes.
- **Enable processing with high throughput**

# Techniques to meet HDFS design goals

- **Simplified coherency model:**
  - The idea is to write once and then read many times. And that simplifies the number of operations required to commit the write.
- **Data replication:**
  - Helps to handle hardware failures.
  - Try to spread the data, same piece of data on different nodes.
- **Move computation close to the data:**
  - So you're not moving data around. That improves your performance and throughput.
- **Relax POSIX requirements to increase the throughput.**

# Basic architecture of HDFS



# HDFS Architecture: Key Components

- **Single NameNode:** A master server that manages the file system namespace and basically regulates access to these files from clients, and it also keeps track of where the data is on the DataNodes and where the blocks are distributed essentially.
- **Multiple DataNodes:** Typically one per node in a cluster. So you're basically using storage which is local.
- **Basic Functions:**
  - Manage the storage on the DataNode.
  - Read and write requests on the clients
  - Block creation, deletion, and replication is all based on instructions from the NameNode.

# Original HDFS Design

- Single NameNode
- Multiple DataNodes
  - Manage storage- blocks of data
  - Serving read/write requests from clients
  - Block creation, deletion, replication

# HDFS in Hadoop 2.0

- HDFS Federation: Basically what we are doing is trying to have multiple data nodes, and multiple name nodes. So that we can increase the name space data. So, if you recall from the first design you have essentially a single node handling all the namespace responsibilities. And you can imagine as you start having thousands of nodes that they'll not scale, and if you have billions of files, you will have scalability issues. So to address that, the federation aspect was brought in. That also brings performance improvements.

- Benefits:
  - Increase namespace scalability
  - Performance
  - Isolation

# HDFS in Hadoop 2

- How its done
- Multiple Namenode servers
- Multiple namespaces
- Data is now stored in Block pools

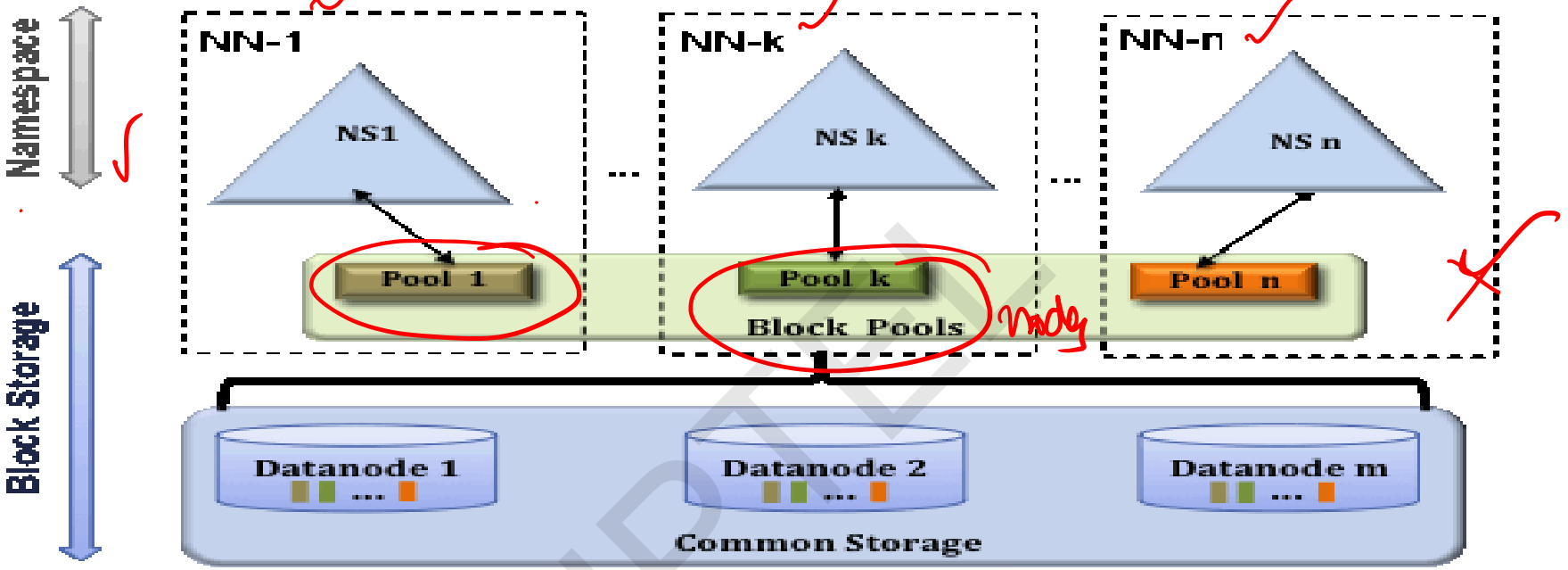
- So there is a pool associated with each namenode or namespace.
- And these pools are essentially spread out over all the data nodes.

# HDFS in Hadoop 2

- High Availability-
  Redundant NameNodes

- Heterogeneous Storage and Archival Storage
  - ARCHIVE, DISK, SSD, RAM_DISK

# Federation: Block Pools




- So, if you remember the original design you have one name space and a bunch of data nodes. So, the structure looks similar.

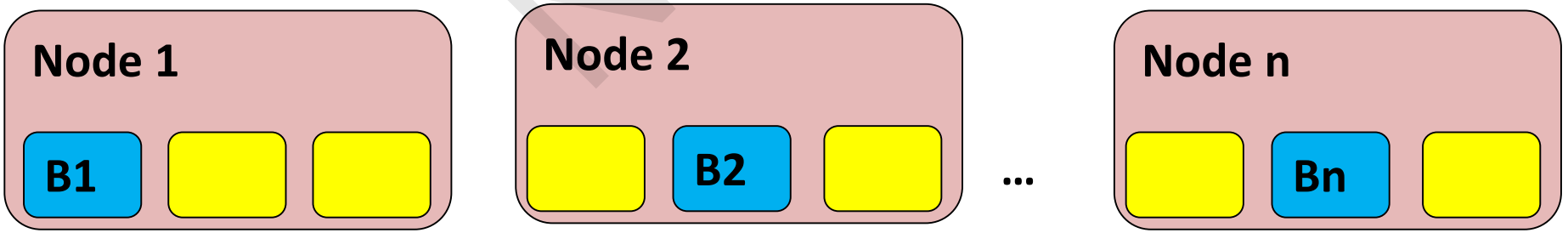
- You have a bunch of NameNodes, instead of one NameNode. And each of those NameNodes is essentially right into these pools, but the pools are spread out over the data nodes just like before. This is where the data is spread out. You can gloss over the different data nodes. So, the block pool is essentially the main thing that's different.

# HDFS Performance Measures

- Determine the number of blocks for a given file size,
- Key HDFS and system components that are affected by the block size.
- An impact of using a lot of small files on HDFS and system

# Recall: HDFS Architecture

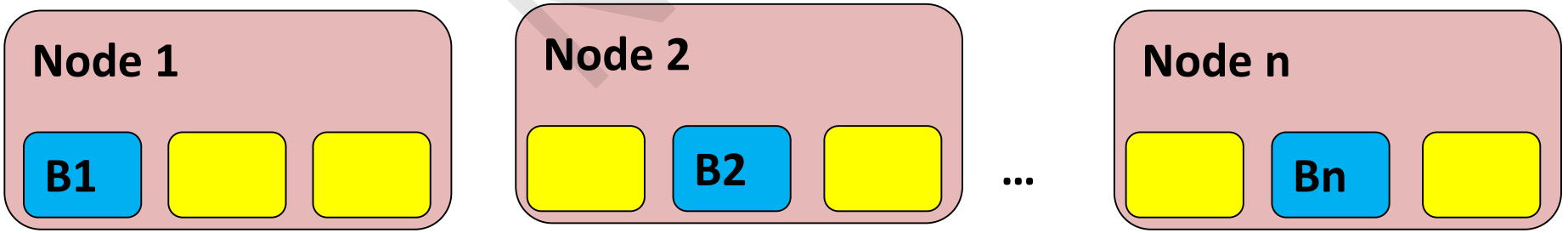
- Distributed data on local disks on several nodes



# HDFS Block Size

- Default block size is 64 megabytes.
- Good for large files!
- So a 10GB file will be broken into: 10 x 1024/64=160 blocks

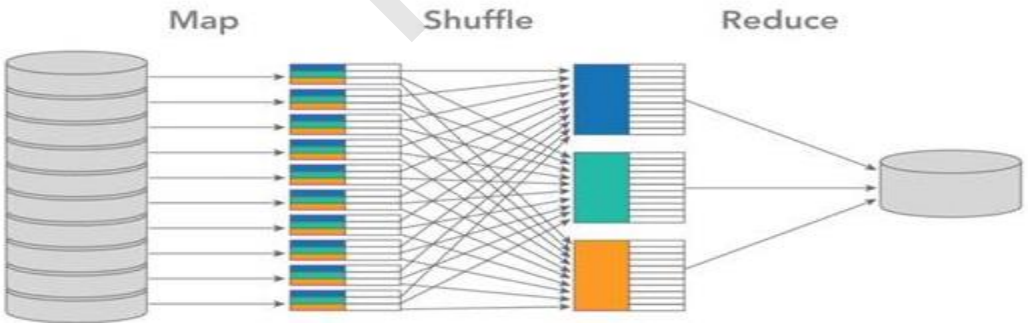




# Importance of No. of Blocks in a file

**NameNode memory usage:** Every block that you create basically every file could be a lot of blocks as we saw in the previous case, 160 blocks. And if you have millions of files that's millions of objects essentially. And for each object, it uses a bit of memory on the NameNode, so that is a direct effect of the number of blocks. But if you have replication, then you have 3 times the number of blocks.

**Number of map tasks:** Number of maps typically depends on the number of blocks being processed.



# Large No. of small files: Impact on Name node

- **Memory usage:** Typically, the usage is around 150 bytes per object. Now, if you have a billion objects, that's going to be like 300GB of memory.

- **Network load:** Number of checks with datanodes proportional to number of blocks

# Large No. of small files: Performance Impact

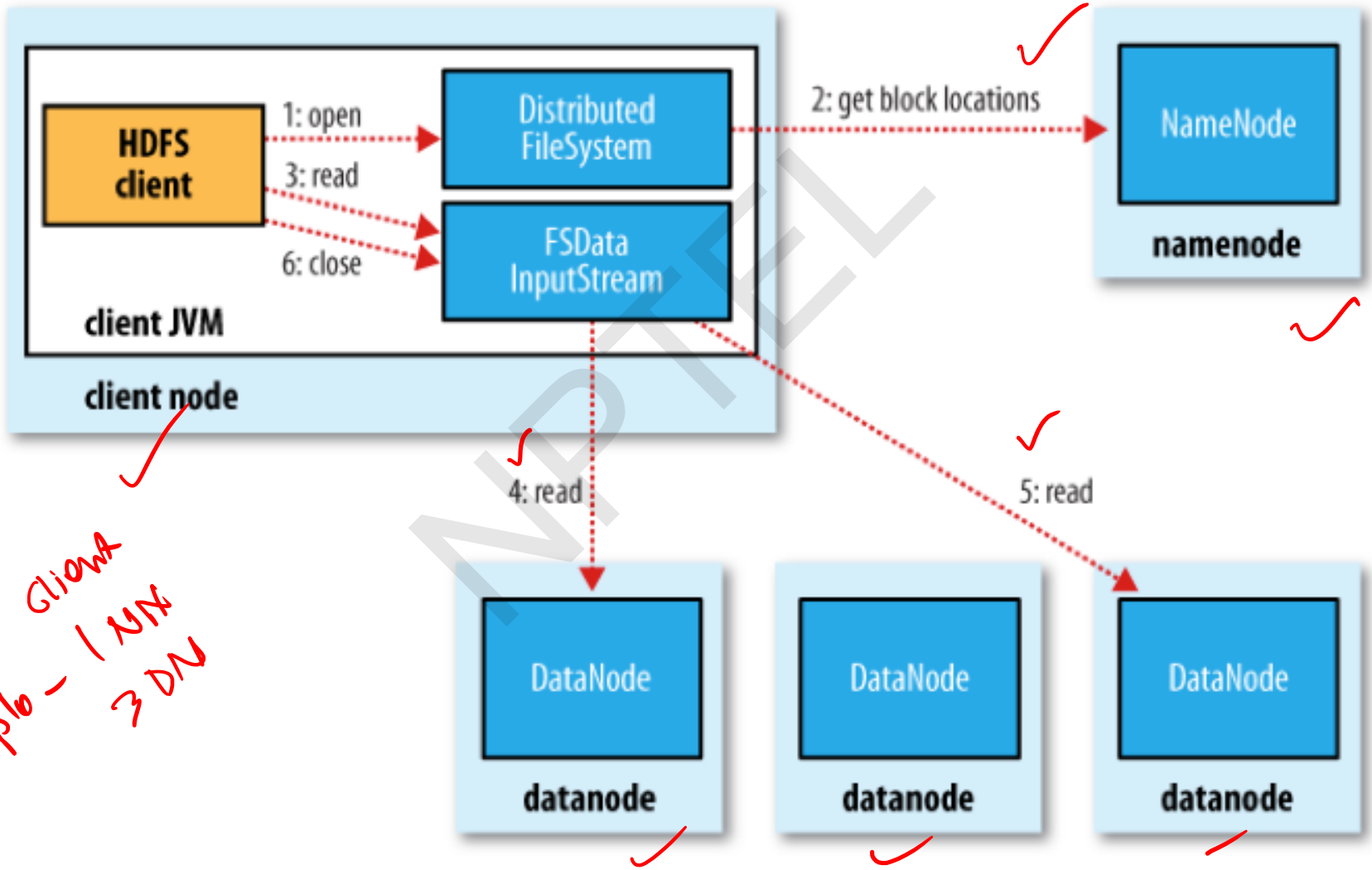
- **Number of map tasks:** Suppose we have 10GB of data to process and you have them all in lots of 32k file sizes? Then we will end up with 327680 map tasks.
- Huge list of tasks that are queued.
- The other impact of this is the map tasks, each time they spin up and spin down, there's a latency involved with that because you are starting up Java processes and stopping them.
- Inefficient disk I/O with small sizes

# HDFS optimized for large files

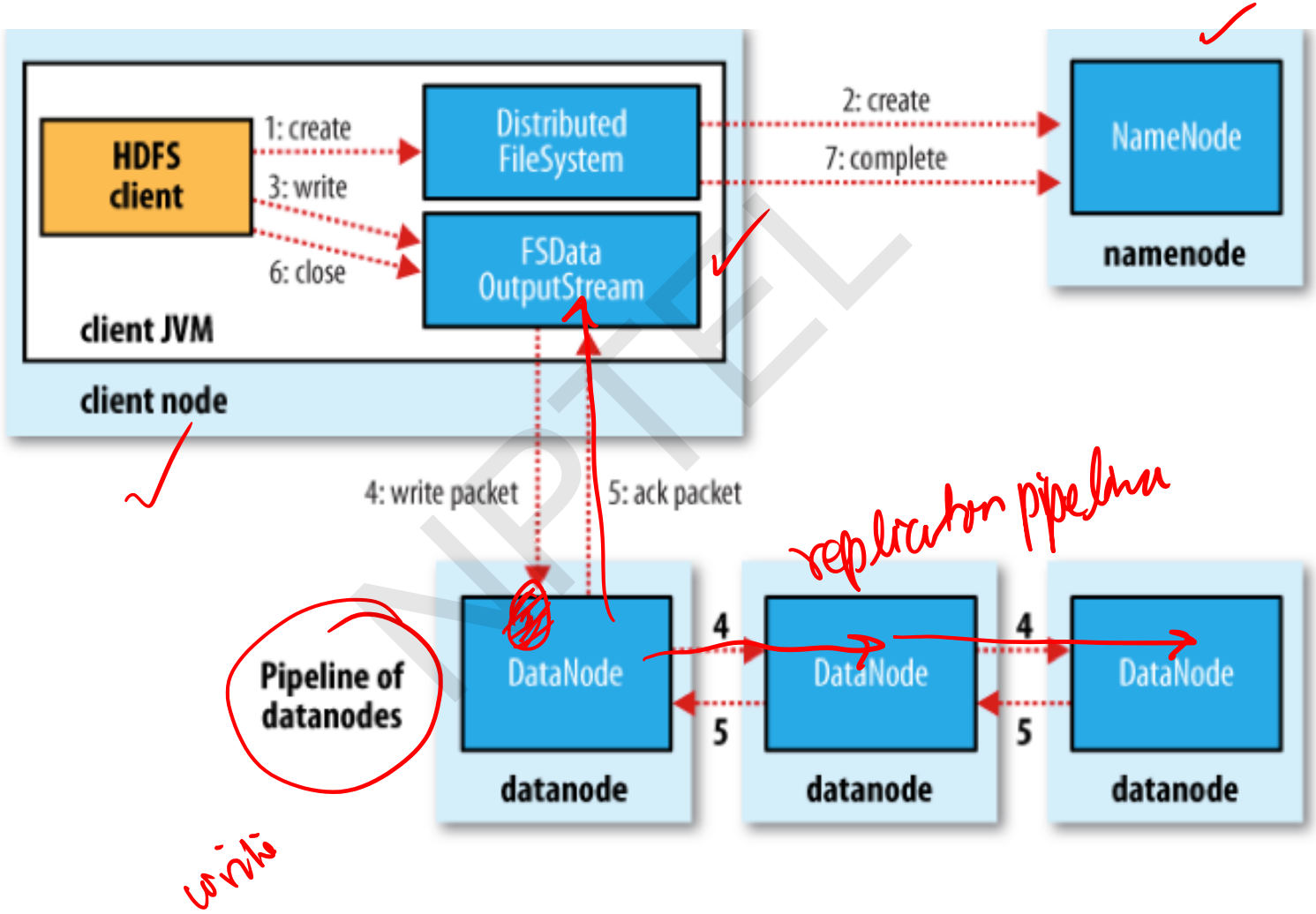
- Lots of small files is bad!
- **Solution:**
  - Merge/Concatenate files
  - Sequence files
  - HBase, HIVE configuration
  - CombineFileInputFormat

# Read/Write Processes in HDFS

# Read Process in HDFS



# Write Process in HDFS



# HDFS Tuning Parameters

# Overview

- Tuning parameters
- Specifically DFS Block size
- NameNode, DataNode system/dfs parameters.

# HDFS XML configuration files

- Tuning environment typically in HDFS XML configuration files, for example, in the hdfs-site.xml.
- This is more for system administrators of Hadoop clusters, but it's good to know what changes affect impact the performance, and especially if your trying things out on your own there some important parameters to keep in mind.
- Commercial vendors have GUI based management console

# HDFS Block Size

- Recall: impacts how much NameNode memory is used, number of map tasks that are showing up, and also have impacts on performance.
- Default 64 megabytes: Typically bumped up to 128 megabytes and can be changed based on workloads.
- The parameter that this changes dfs.blocksize or dfs.block.size.

# HDFS Replication

- Default replication is 3.
- Parameter: dfs.replication

- Tradeoffs:
  - Lower it to reduce replication cost
  - Less robust
  - Higher replication can make data local to more workers
  - Lower replication ➔ More space

# Lot of other parameters

- Various tunables for datanode, namenode.

- **Examples:**
- Dfs.datanode.handler.count (10): Sets the number of server threads on each datanode
- Dfs.namenode.fs-limits.max-blocks-per-file: Maximum number of blocks per file.

- **Full List:**

- http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

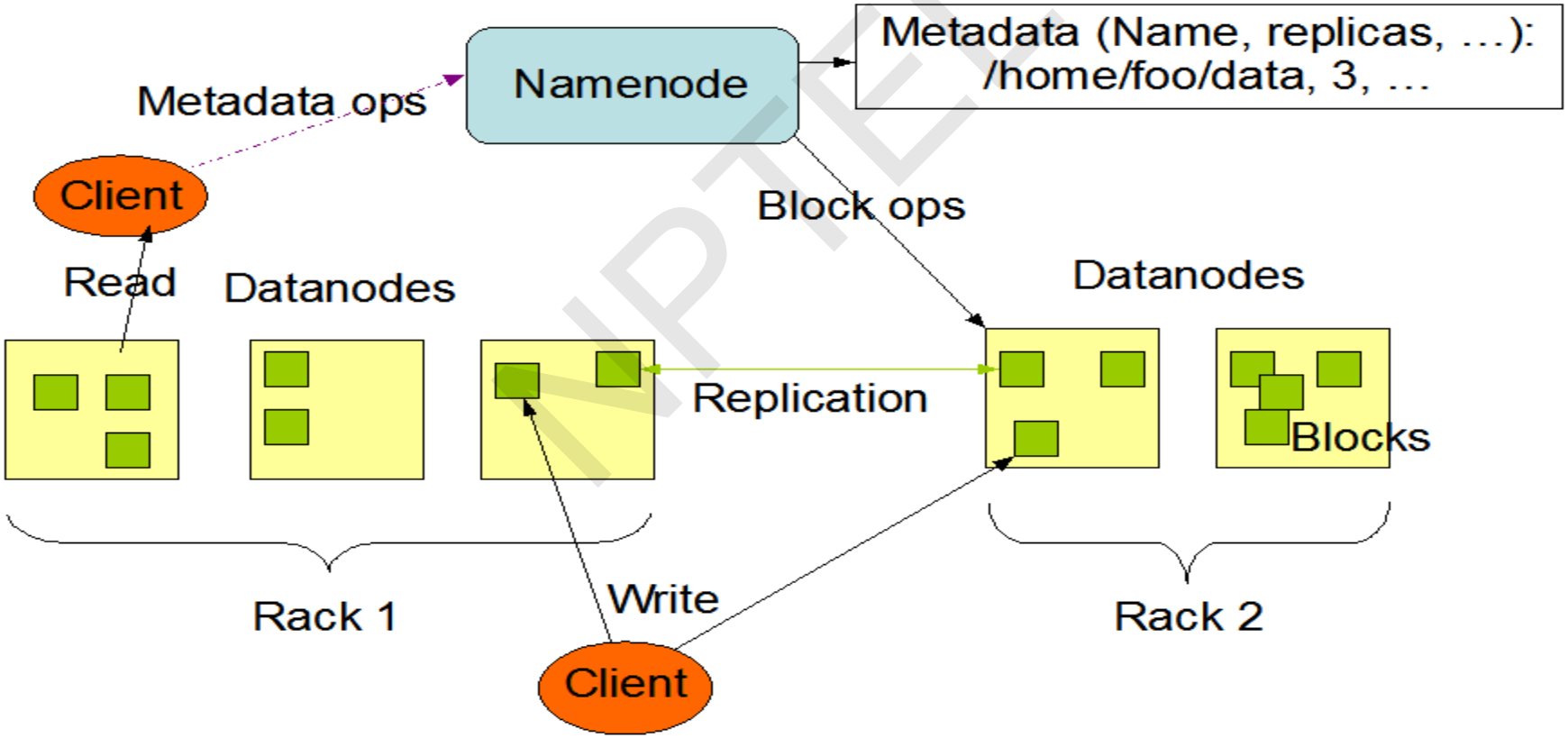
# HDFS Performance and Robustness

# Common Failures

- **DataNode Failures:** Server can fail, disk can crash, data corruption.
- **Network Failures:** Sometimes there's data corruption because of network issues or disk issue. So, all of that could lead to a failure in the DataNode aspect of HDFS. You could have network failures. So, you could have a network go down between a particular and the name node that can affect a lot of data nodes at the same time.
- **NameNode Failures:** Could have name node failures, disk failure on the name node itself or the name node itself could corrupt this process.

# HDFS Robustness

- NameNode receives heartbeat and block reports from DataNodes



# Mitigation of common failures

- **Periodic heartbeat: from DataNode to NameNode.**
- **DataNodes without recent heartbeat:**
  - Mark the data. And any new I/O that comes up is not going to be sent to that data node. Also remember that NameNode has information on all the replication information for the files on the file system. So, if it knows that a datanode fails which blocks will follow that replication factor.
  - Now this replication factor is set for the entire system and also you could set it for particular file when you're writing the file. Either way, the NameNode knows which blocks fall below replication factor. And it will restart the process to re-replicate.

# Mitigation of common failures

- Checksum computed on file creation.
- Checksums stored in HDFS namespace.
- Used to check retrieved data.
- Re-read from alternate replica

# Mitigation of common failures

- Multiple copies of central meta data structures.
- Failover to standby NameNode- manual by default.

# Performance

- Changing blocksize and replication factor can improve performance.
- **Example: Distributed copy**
- Hadoop distcp allows parallel transfer of files.

# Replication trade off with respect to robustness

- One performance tradeoff is, actually when you go out to do some of the map reduce jobs, having replicas gives additional locality possibilities, but the big trade off is the robustness. In this case, we said no replicas. Might lose a node or a local disk: can't recover because there is no replication.
- Similarly, with data corruption, if you get a checksum that's bad, now you can't recover because you don't have a replica.
- Other parameters changes can have similar effects.

# Conclusion

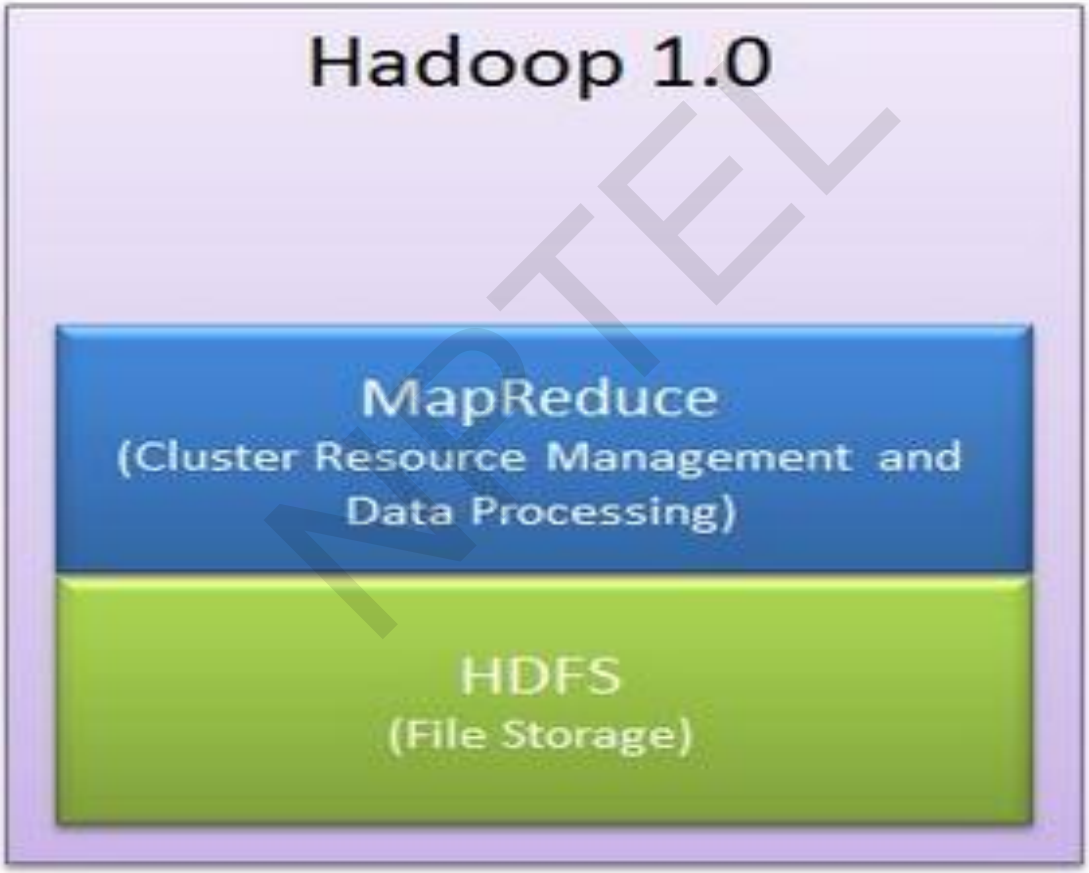
- In this lecture, we have discussed design goals of HDFS, the read/write process to HDFS, the main configuration tuning parameters to control HDFS performance and robustness.

# Hadoop MapReduce 1.0

**Dr. Rajiv Misra**
**Dept. of Computer Science & Engg.**
**Indian Institute of Technology Patna**
**rajivm@iitp.ac.in**

# What is Map Reduce

- MapReduce is the execution engine of Hadoop.

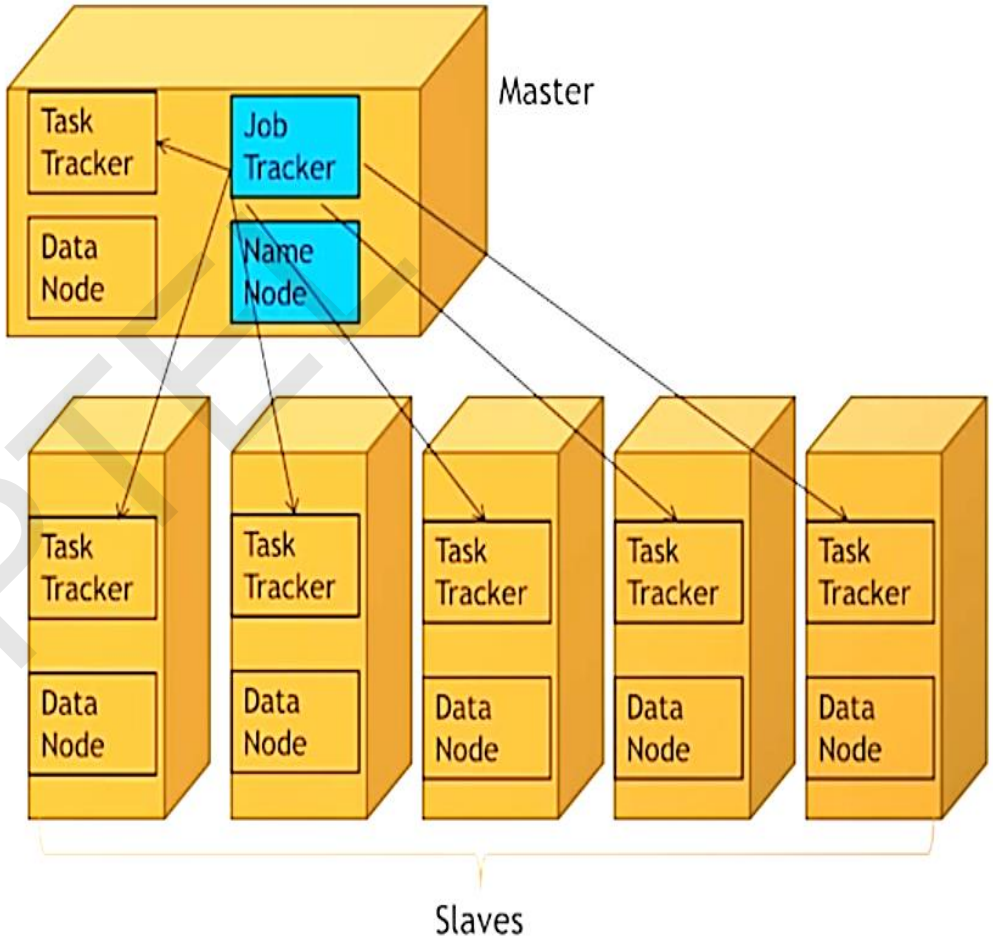


# Map Reduce Components

- The Job Tracker
- Task Tracker

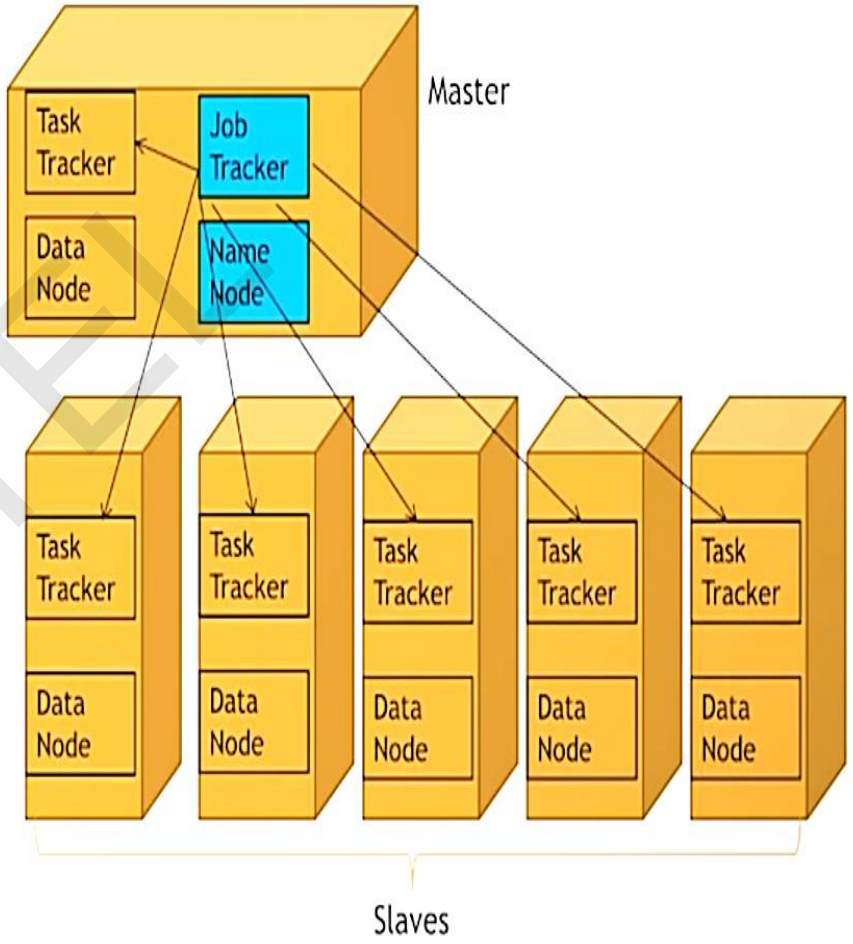
# The Job Tracker

- The Job Tracker is hosted inside the master and it receives the job execution request from the client.
- Its main duties are to break down the receive job that is big computations in small parts allocate the partial computations that is tasks to the slave nodes monitoring the progress and report of task execution from the slave.
- The unit of execution is job.



# The Task Tracker

- Task tracker is the MapReduce component on the slave machine as there are multiple slave machines.
- Many task trackers are available in a cluster its duty is to perform computation given by job tracker on the data available on the slave machine.
- The task tracker will communicate the progress and report the results to the job tracker.
- The master node contains the job tracker and name node whereas all slaves contain the task tracker and data node.



# Execution Steps

**Step-1**The client submits the job to Job Tracker

**Step-2** Job Tracker asks Name node the location of data

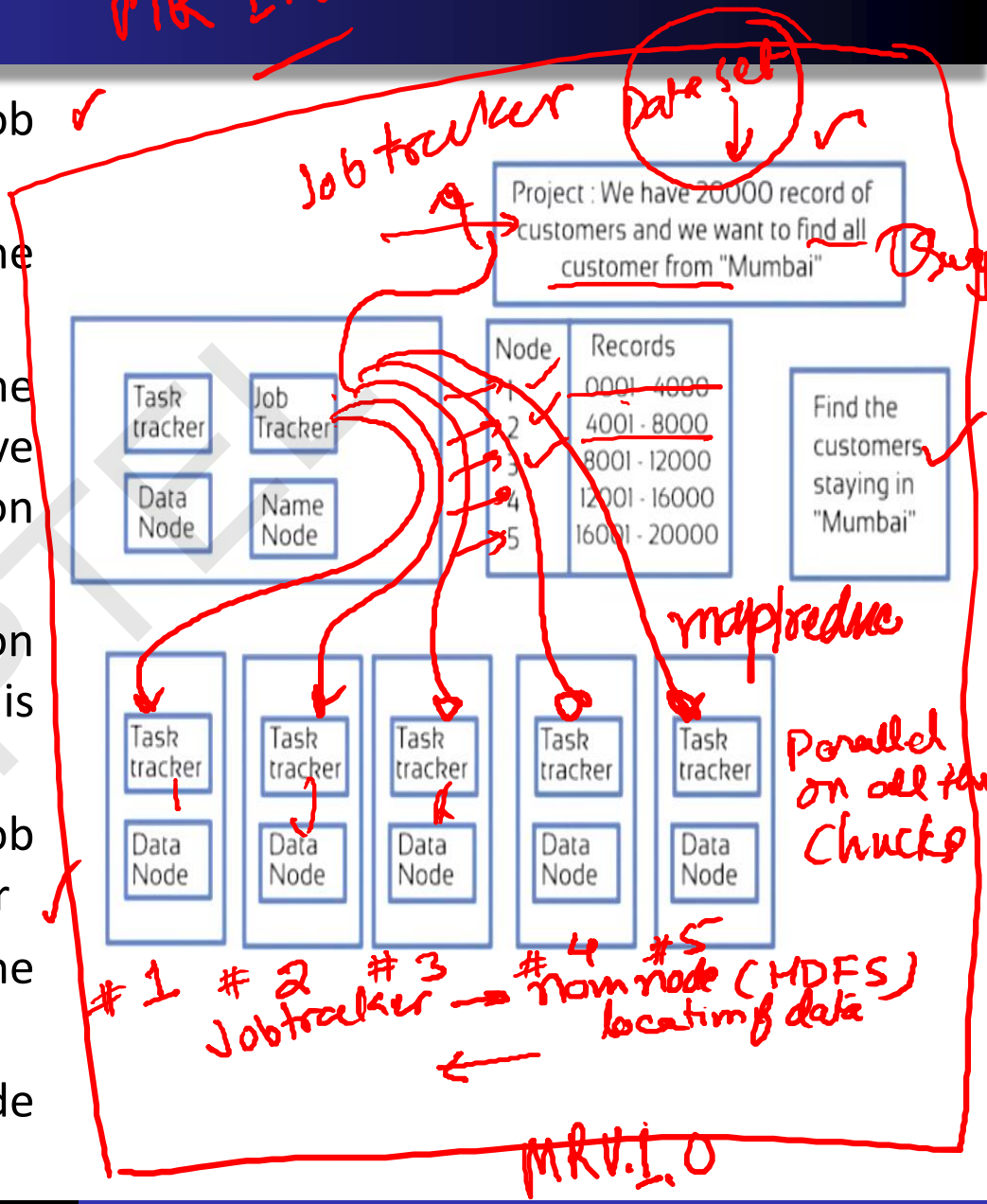
**Step-3** As per the reply from name node, the Job Tracker ask respective task trackers to execute the task on their data

**Step-4** All the results are stored on some Data Node and the Name Node is informed about the same.

**Step-5** The task trackers inform the job completion and progress to Job Tracker

**Step-6** The Job Tracker inform the completion to client

**Step-7** Client contacts the Name Node and retrieve the results



# Hadoop MapReduce 2.0

**Dr. Rajiv Misra**
**Dept. of Computer Science & Engg.**
**Indian Institute of Technology Patna**
**rajivm@iitp.ac.in**

# Preface

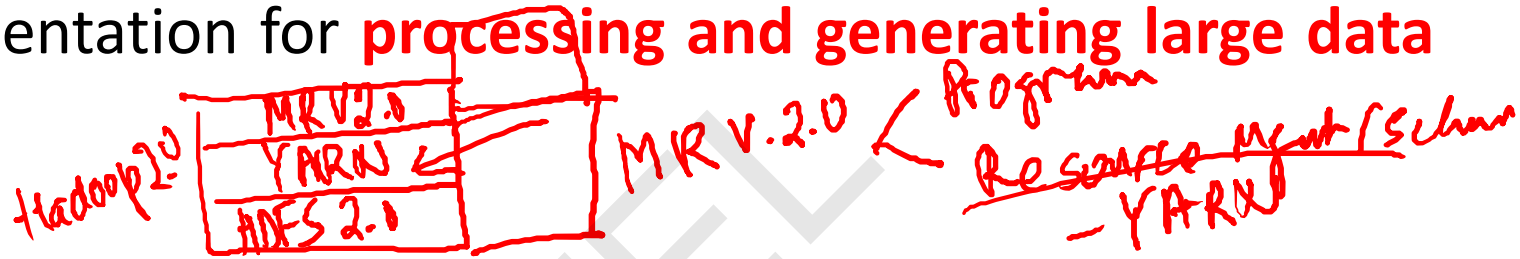
**Content of this Lecture:**

- In this lecture, we will discuss the '**MapReduce paradigm**' and its internal working and implementation overview.

- We will also see many examples and different applications of MapReduce being used, and look into how the '**scheduling and fault tolerance**' works inside MapReduce.

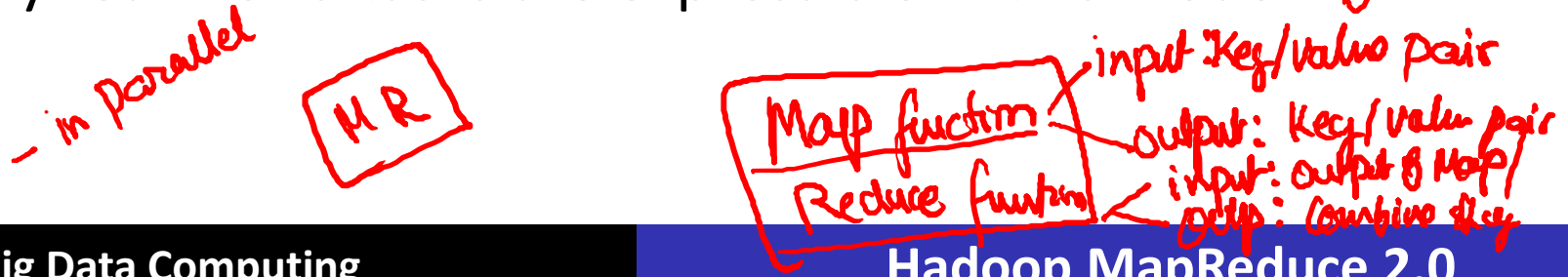
# Introduction

- **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.



# Contd…

- 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 details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication.
- This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.
- A **typical MapReduce computation processes** many terabytes of data on thousands of machines. Hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

# Distributed File System

## Chunk Servers

- File is split into contiguous chunks
- Typically each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

## Master node

- Also known as Name Nodes in HDFS
- Stores metadata
- Might be replicated

## Client library for file access

- Talks to master to find chunk servers
- Connects directly to chunkservers to access data

# Motivation for Map Reduce (Why)

- **Large-Scale Data Processing**
  - Want to use 1000s of CPUs
  - But don't want hassle of managing things
- **MapReduce Architecture provides**
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

# MapReduce Paradigm

# What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)

**Sum of squares:**

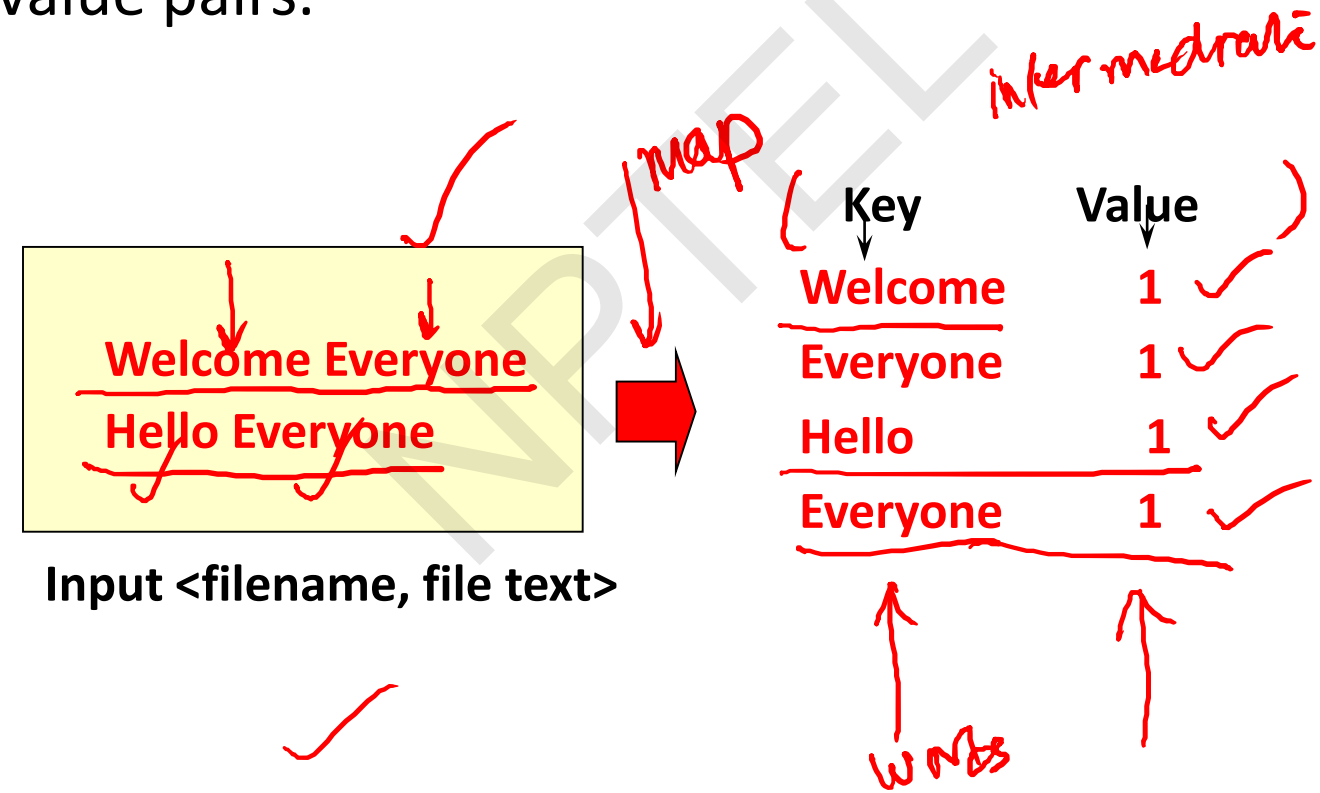
- **(map square '(1 2 3 4))**
  - **Output: (1 4 9 16)**

  **[processes each record sequentially and independently]**
- **(reduce + '(1 4 9 16))**
  - **(+ 16 (+ 9 (+ 4 1) ) )**
  - **Output: 30**

  **[processes set of all records in batches]**
- Let's consider a sample application: **Wordcount**
  - You are given a **huge** dataset (e.g., Wikipedia dump or all of Shakespeare's works) and asked to list the count for each of the words in each of the documents therein

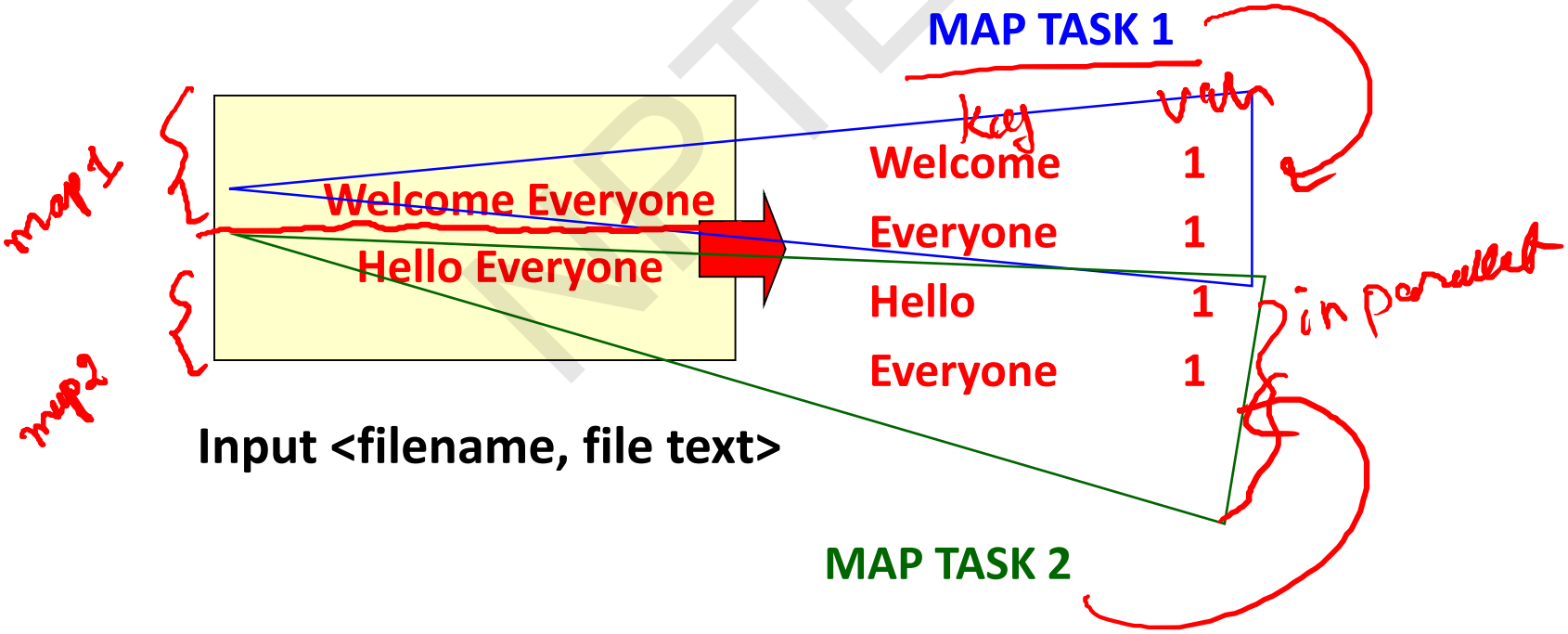
# Map

- Process individual records to generate intermediate key/value pairs.



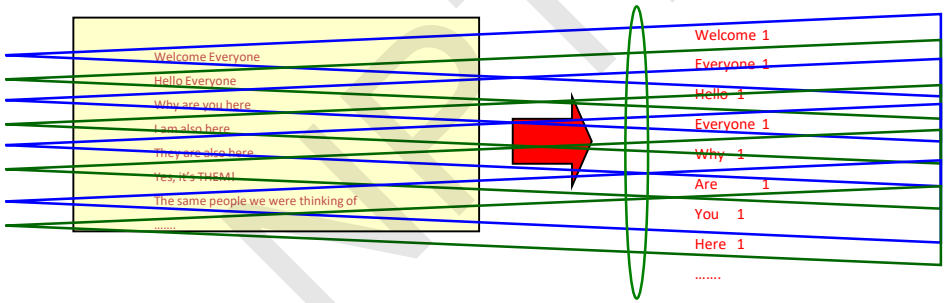
# Map

- **Parallelly** Process individual records to generate intermediate key/value pairs.



# Map

- **Parallelly** Process **a large number** of individual records to generate intermediate key/value pairs.

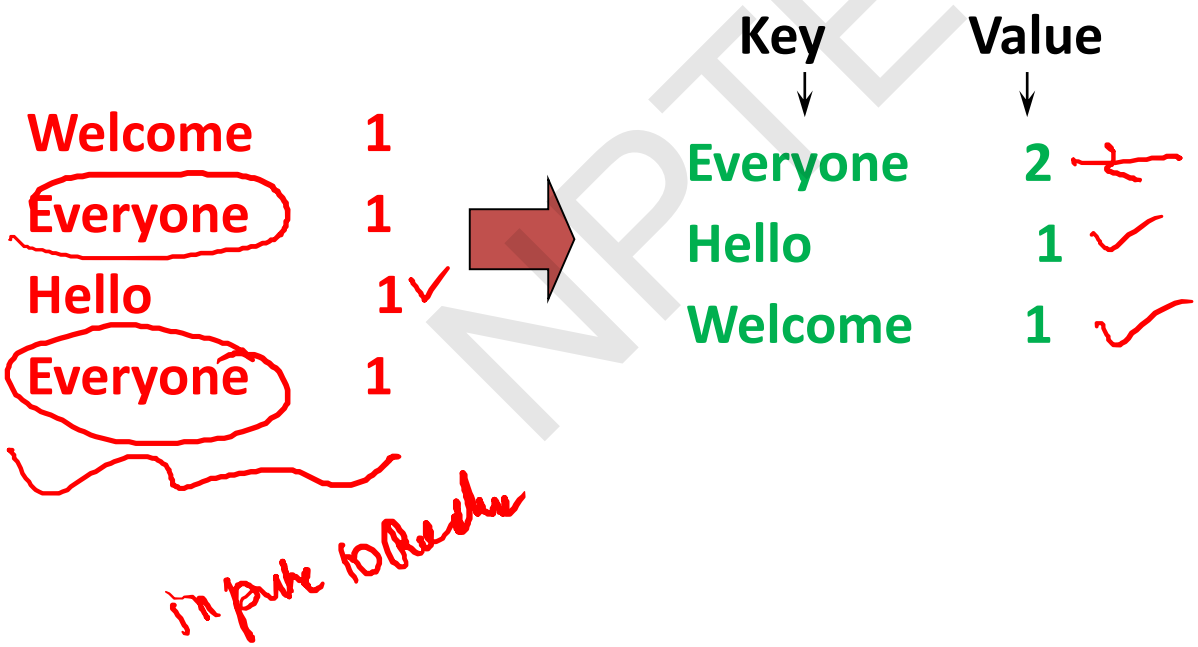



**Input <filename, file text>**

**MAP TASKS**

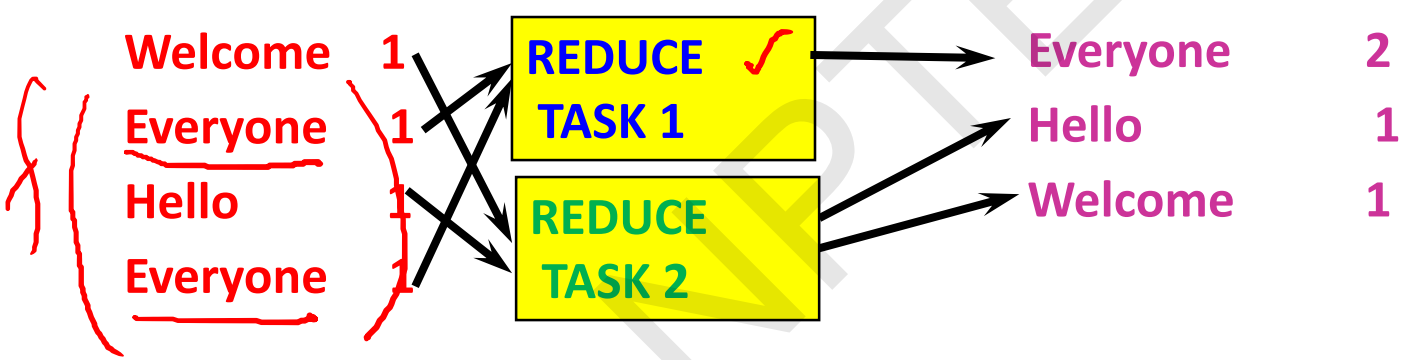
# Reduce

- Reduce processes and merges all intermediate values associated per **key**



# Reduce

- Each key assigned to one Reduce
- Parallelly Processes and merges all intermediate **values by partitioning keys**




- Popular: ***Hash partitioning***, *i.e.*, key is assigned to
  - reduce # = hash(key)%number of reduce tasks

# Programming Model

- The computation takes a set of **input key/value pairs**, and produces a set of **output key/value pairs.**

- The user of the MapReduce library expresses the computation as two functions:

(i) The Map

(ii) The Reduce

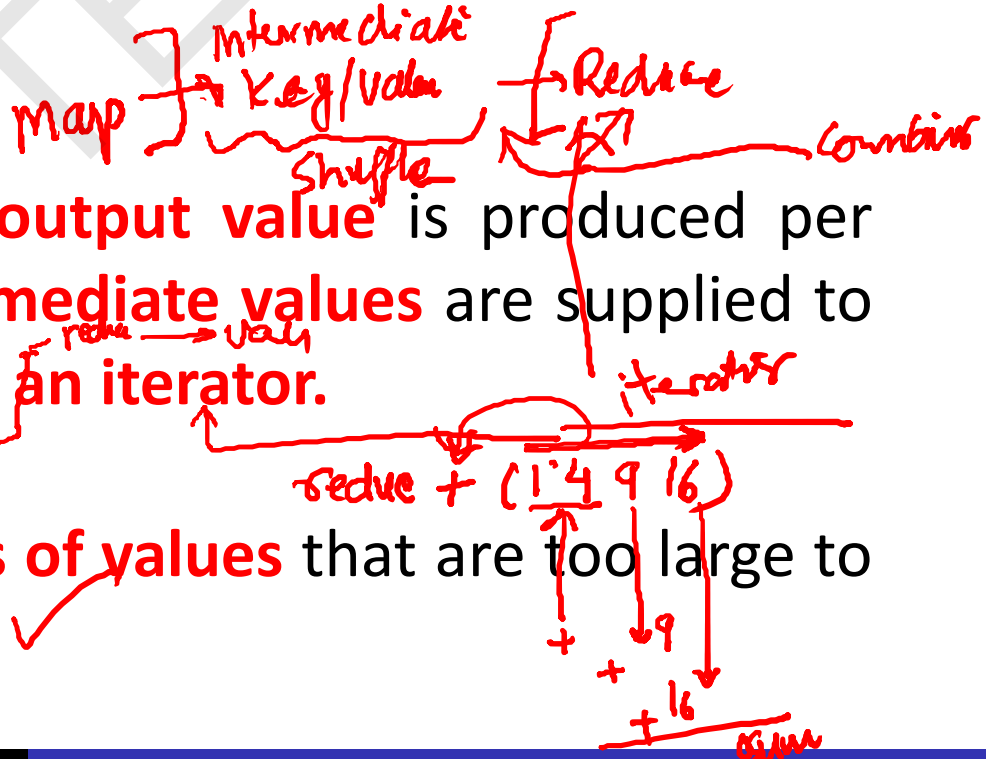
# (i) Map Abstraction

- Map, written by the user, takes an input pair and produces a set of **intermediate key/value pairs.**

- The MapReduce library groups together all intermediate values associated with the same **intermediate key *'I'*** and passes them to the **Reduce function.**

# (ii) Reduce Abstraction

- **The Reduce function,** also written by the user, accepts an **intermediate key *'I'*** and a set of values for that key.
- It merges together these values to form a **possibly smaller set of values.**
- Typically just **zero or one output value** is produced per Reduce invocation. The **intermediate values** are supplied to the user's reduce function via **an iterator.**
- This allows us to **handle lists of values** that are too large to fit in memory.



# Map-Reduce Functions for Word Count

```
map(key, value):
// key: document name; value: text of document
  for each word w in value:
        emit(w, 1)

reduce(key, values):
// key: a word; values: an iterator over counts
        result = 0
        for each count v in values:
              result += v
        emit(key, result)
```

# Map-Reduce Functions

- **Input:** a set of key/value pairs
- User supplies two functions:

  map(k,v) → list(k1,v1)

  reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- **Output** is the set of (k1,v2) pairs

# MapReduce Applications

# Applications

- Here are a few simple applications of interesting programs that can be easily expressed as **MapReduce computations.**
- **Distributed Grep:** The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency:** The map function processes logs of web page requests and outputs (URL; 1). The reduce function adds together all values for the same URL and emits a (URL; total count) pair.
- **ReverseWeb-Link Graph:** The map function outputs (target; source) pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: (target; list(source))

# Contd…

- **Term-Vector per Host:** A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word; frequency) pairs.

- The map function emits a (hostname; term vector) pair for each input document (where the hostname is extracted from the URL of the document).

- The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final (hostname; term vector) pair

# Contd…

- **Inverted Index:** The map function parses each document, and emits a sequence of (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (word; list(document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

- **Distributed Sort:** The map function extracts the key from each record, and emits a (key; record) pair. The reduce function emits all pairs unchanged.

# Applications of MapReduce

**(1) Distributed Grep:**

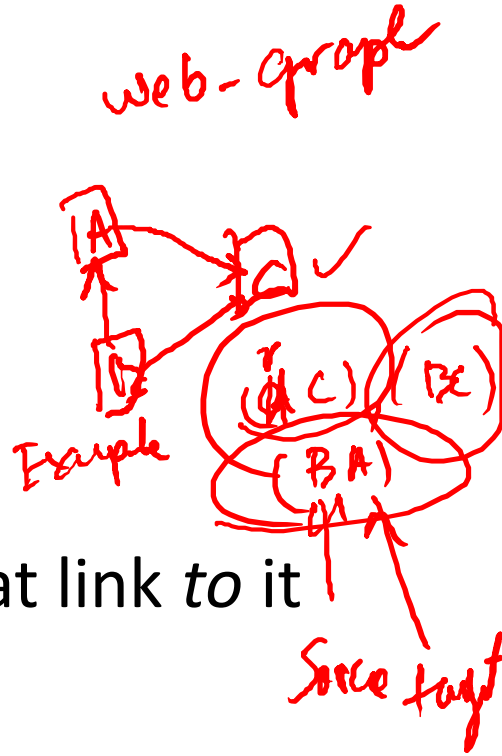
- Input: large set of files
- Output: lines that match pattern

- Map – ***Emits a line if it matches the supplied pattern***

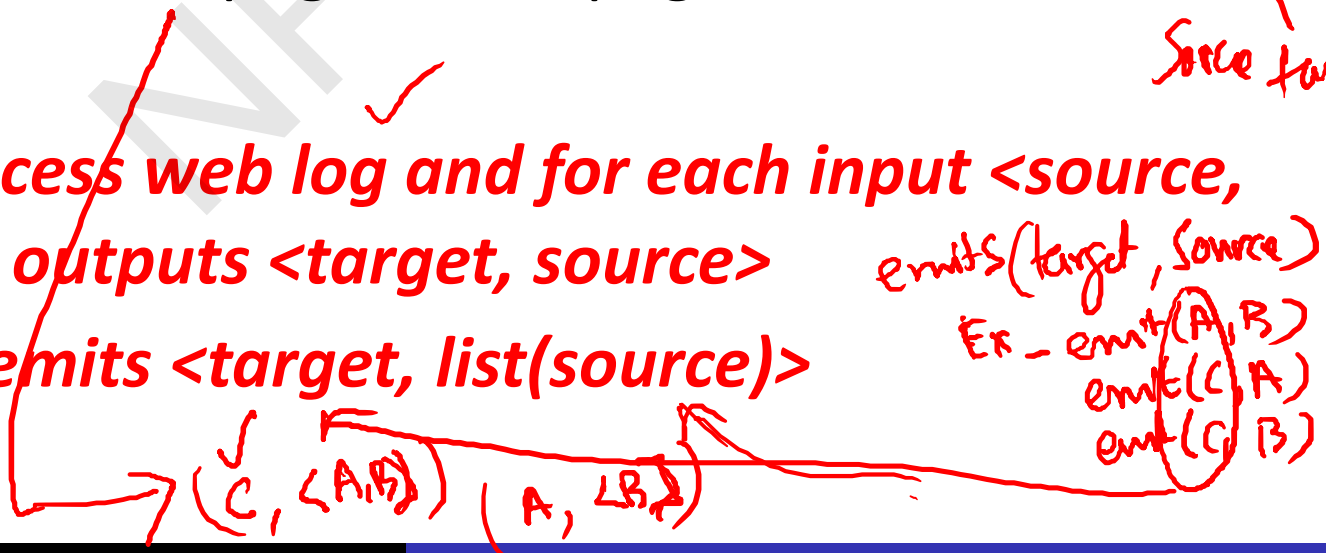
- Reduce – ***Copies the intermediate data to output***

# Applications of MapReduce

## (2) Reverse Web-Link Graph:

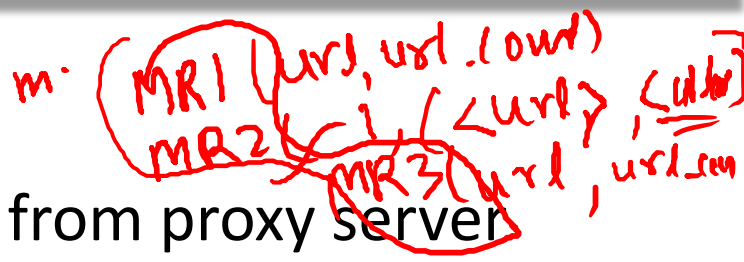



- **Input:** Web graph: tuples (a, b) where (page a → page b)
- **Output:** For each page, list of pages that link *to* it
- Map – ***process web log and for each input <source, target>, it outputs <target, source>***
- Reduce - ***emits <target, list(source)>***



# Applications of MapReduce

## (3) Count of URL access frequency:



- Input: Log of accessed URLs, e.g., from proxy server
- Output: For each URL, % of total accesses for that URL

- Map – ***Process web log and outputs <URL, 1>***
- Multiple Reducers - ***Emits <URL, URL_count>***

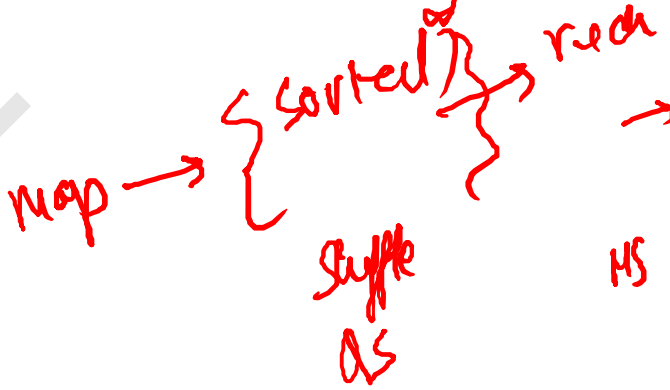
(So far, like Wordcount. But still need %)

- Chain another MapReduce job after above one
- Map – ***Processes <URL, URL_count>*** and outputs ***<1, (<URL, URL_count> )>***
- 1 Reducer – Does two passes. In first pass, sums up all ***URL_count's*** to calculate overall_count. In second pass calculates %'s

***Emits multiple <URL, URL_count/overall_count>***

# Applications of MapReduce

**(4) Map task's output is sorted (e.g., quicksort)**
**Reduce task's input is sorted (e.g., mergesort)**



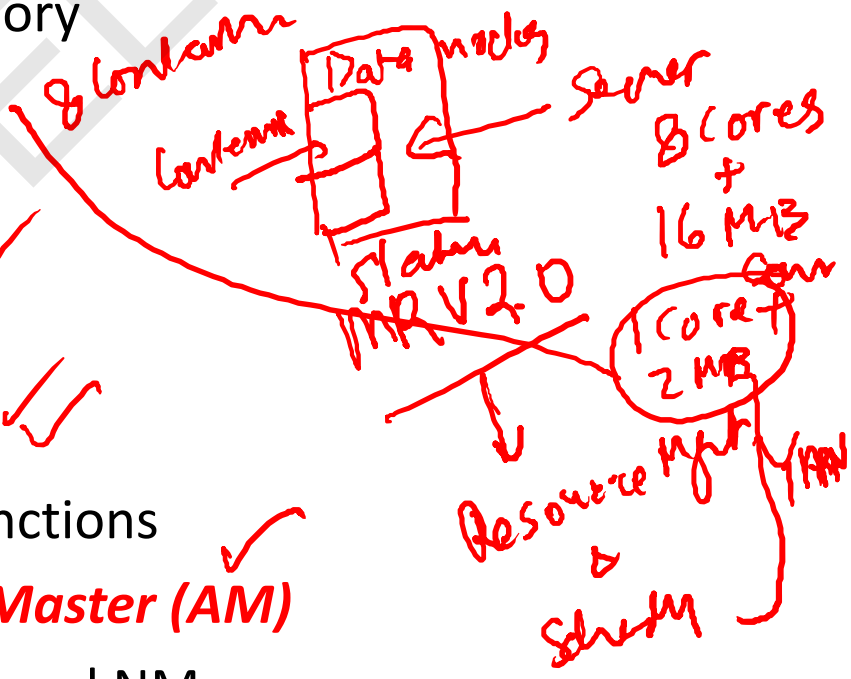

**Sort**

- Input: Series of (key, value) pairs
- Output: Sorted <value>s

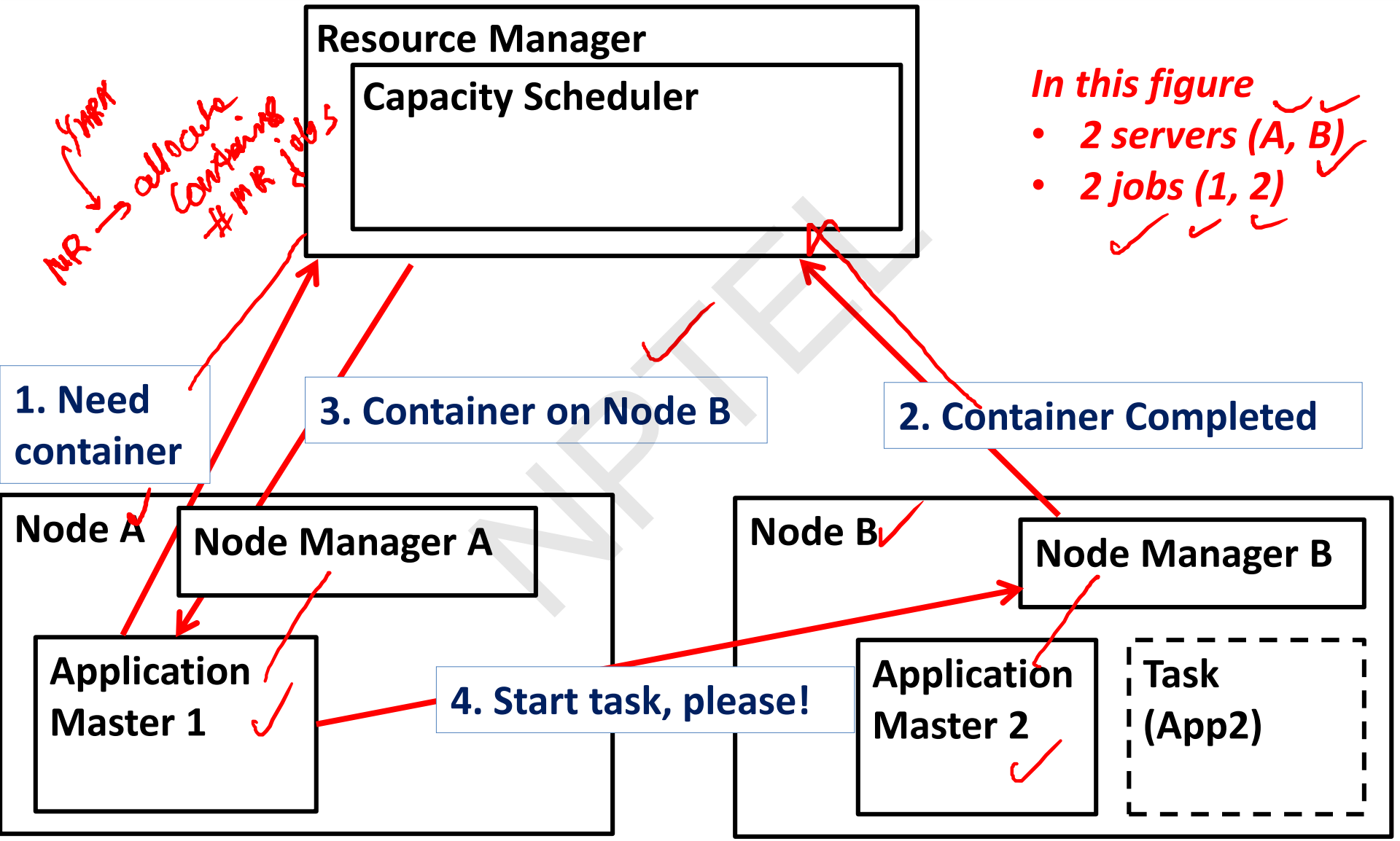
- Map – ***<key, value> → <value, _> (identity)***
- Reducer – ***<key, value> → <key, value> (identity)***
- Partitioning function – partition keys across reducers based on **ranges** (can't use hashing!)
  - Take data distribution into account to balance reducer tasks

# The YARN Scheduler

- Used underneath Hadoop 2.x +
- YARN = Yet Another Resource Negotiator
- Treats each server as a collection of ***containers***
  - Container = fixed CPU + fixed memory
- Has 3 main components
  - **Global *Resource Manager (RM)***
    - Scheduling
  - **Per-server *Node Manager (NM)***
    - Daemon and server-specific functions
  - **Per-application (job) *Application Master (AM)***
    - Container negotiation with RM and NMs
    - Detecting task failures of that job



# YARN: How a job gets a container



# MapReduce Examples

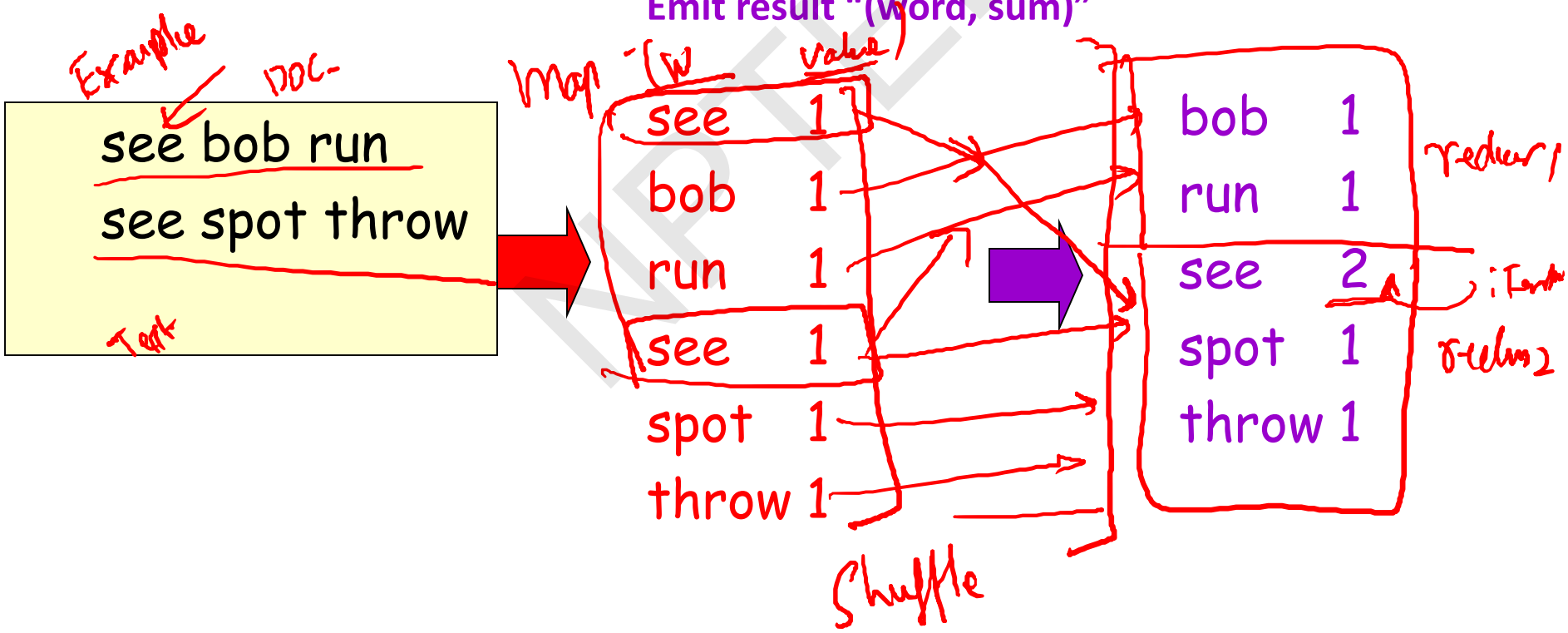
# Example: 1 Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)

reduce(key, values):
// key: a word; values: an iterator over counts
        result = 0
        for each count v in values:
            result += v
        emit(key, result)
```

# Count Illustrated

map(key=url, val=contents):
For each word *w* in contents, emit (w, "1")
reduce(key=word, values=uniq_counts):
Sum all "1"s in values list
Emit result "(word, sum)"

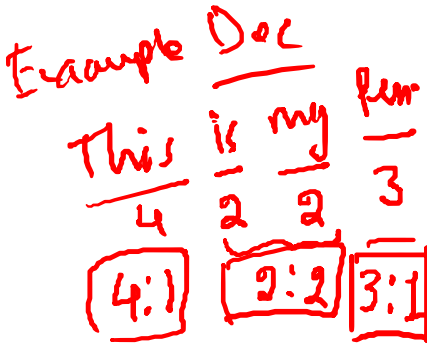


# Example 2: Counting words of different lengths

- **The map function takes a value and outputs key:value pairs.**

- For instance, if we define a map function that takes a string and outputs the length of the word as the key and the word itself as the value then

- map(steve) would return 5:steve and
- map(savannah) would return 8:savannah.

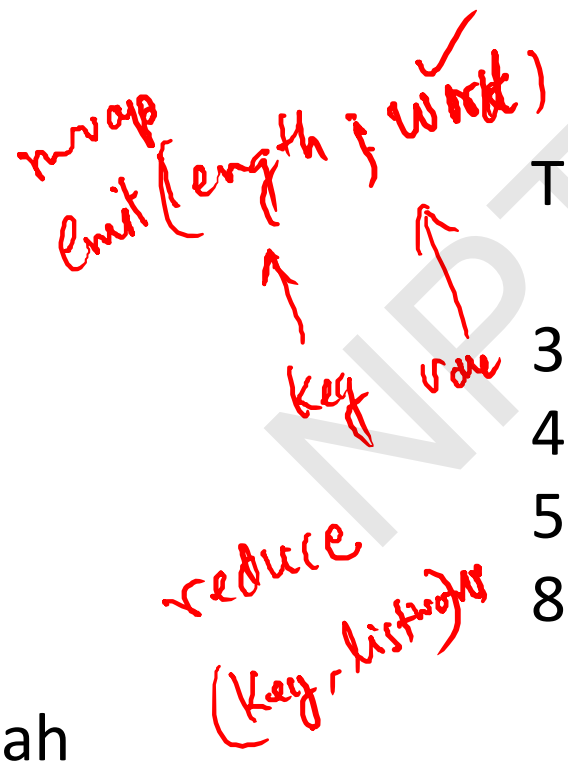



This allows us to run the map function against values in parallel and provides a huge advantage.

# Example 2: Counting words of different lengths

Before we get to the reduce function, the mapreduce framework groups all of the values together by key, so if the map functions output the following **key:value pairs:**

3 : the
3 : and
3 : you
4 : then
4 : what
4 : when
5 : steve
5 : where
8 : savannah
8 : research




They get grouped as:

3 : [the, and, you]
4 : [then, what, when]
5 : [steve, where]
8 : [savannah, research]

# Example 2: Counting words of different lengths

- Each of these lines would then be passed as an argument to the reduce function, which accepts a key and a list of values.
- In this instance, we might be trying to figure out how many words of certain lengths exist, so our reduce function will just count the number of items in the list and output the key with the size of the list, like:

3 : 3

4 : 3

5 : 2

8 : 2

# Example 2: Counting words of different lengths

- The reductions can also be done in parallel, again providing a huge advantage. We can then look at these final results and see that there were only two words of length 5 in the corpus, etc...

- **The most common example of mapreduce is for counting the number of times words occur in a corpus.**

# Example 3: Word Length Histogram

## 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 3: Word Length Histogram

## Abridged Declaration of Independence

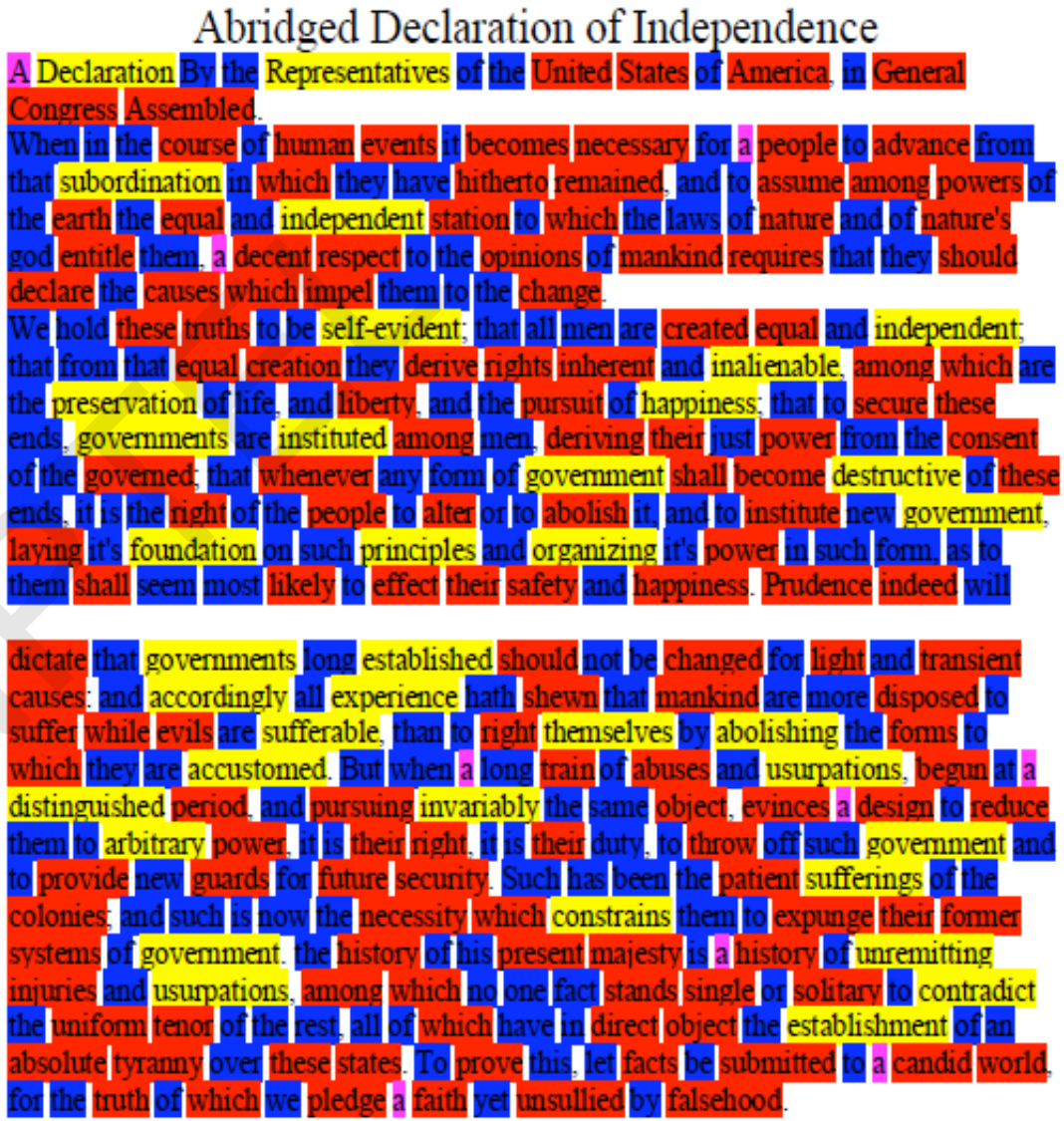
A Declaration By the Representatives of the United States of America, in General Congress Assembled. When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent 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.



**How many "big", "medium" and "small" words, are used ?**

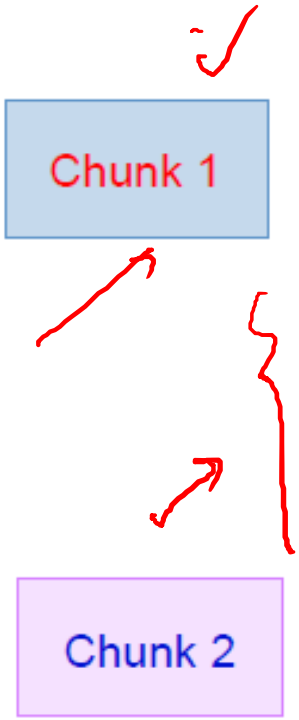
# Example 3: Word Length Histogram

- Big = Yellow = 10+ letters
- Medium = Red = 5..9 letters
- Small = Blue = 2..4 letters
- Tiny = Pink = 1 letter



# Example 3: Word Length Histogram

Split the document into chunks and process each chunk on a different computer



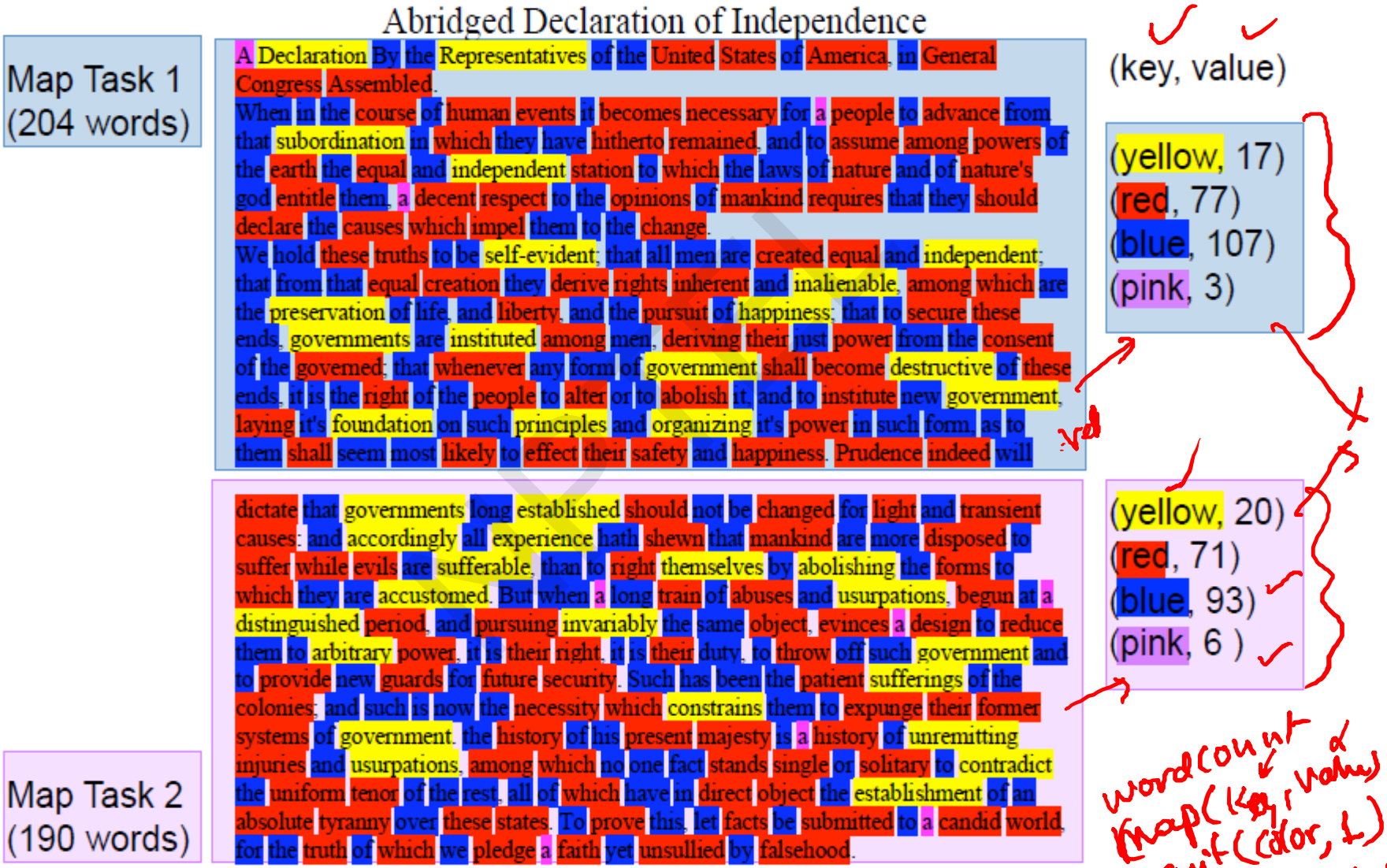

## Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.

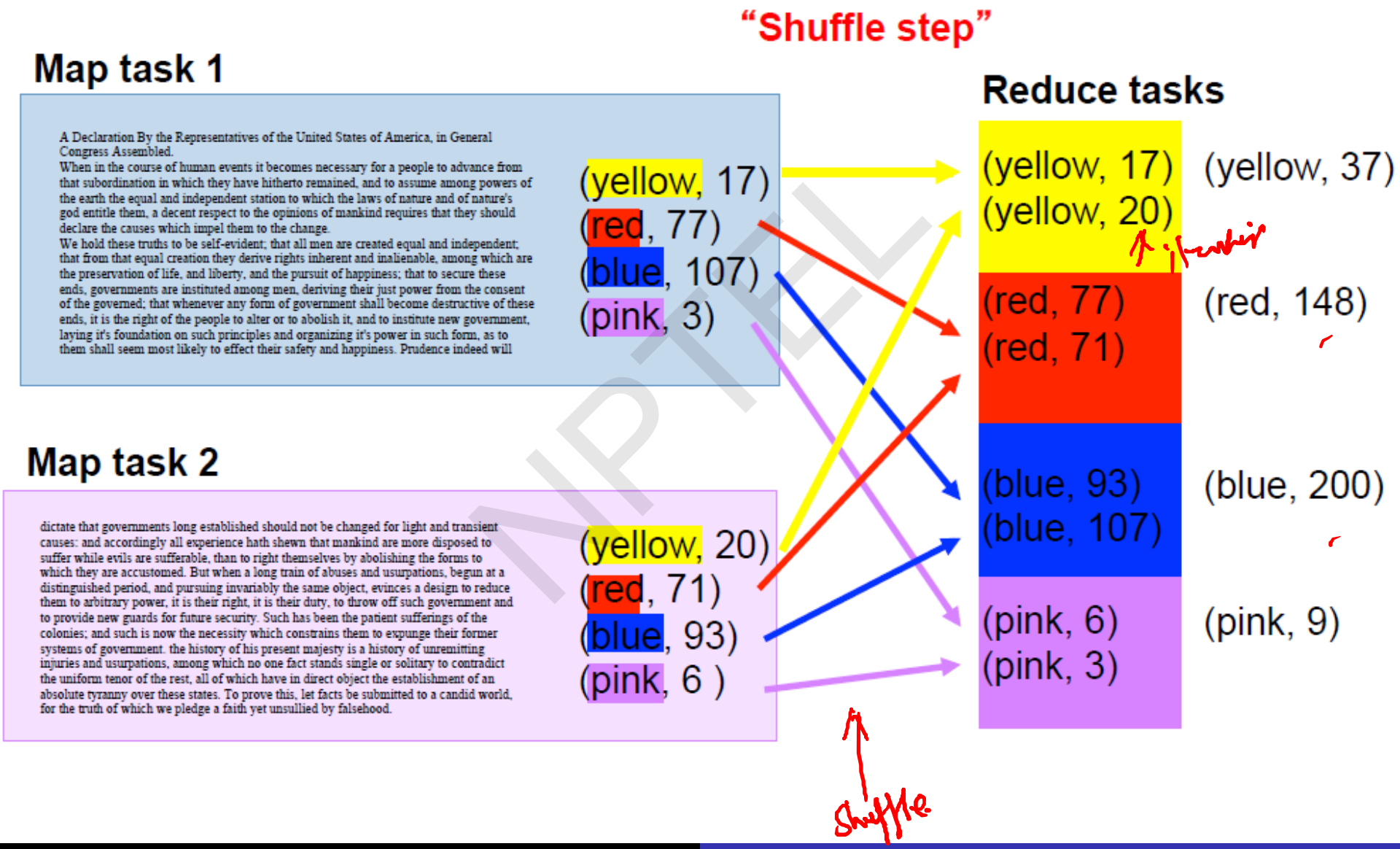
When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the 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 3: Word Length Histogram



# Example 3: Word Length Histogram



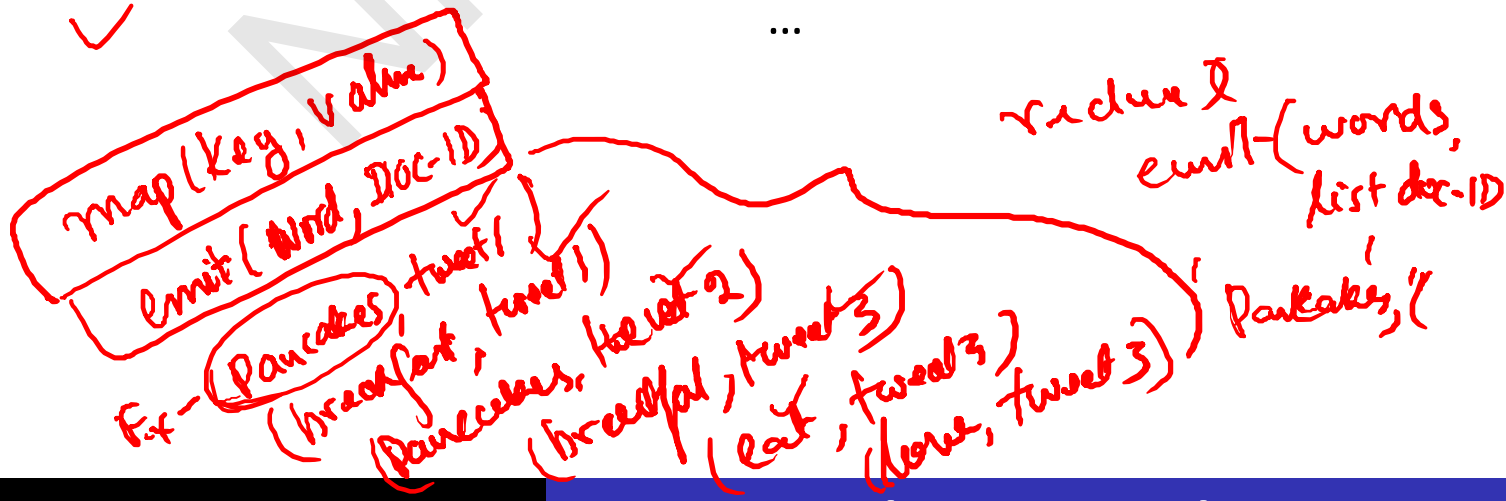
# Example 4: Build an Inverted Index

**Input:**

tweet1, ("I love pancakes for breakfast")
tweet2, ("I dislike pancakes")
tweet3, ("What should I eat for breakfast?")
tweet4, ("I love to eat")

**Desired output:**

"pancakes", (tweet1, tweet2)
"breakfast", (tweet1, tweet3)
"eat", (tweet3, tweet4)
"love", (tweet1, tweet4)
…



# Example 5: Relational Join

**Employee**

| Name | SSN |
|---|---|
| Sue | 999999999 |
| Tony | 777777777 |

**Assigned Departments**

| EmpSSN | DepName |
|---|---|
| 999999999 | Accounts |
| 777777777 | Sales |
| 777777777 | Marketing |

SSN = EMPSSN

Emplyee ⋈ Assigned Departments

| Name | SSN | EmpSSN | DepName |
|---|---|---|---|
| Sue | 999999999 | 999999999 | Accounts |
| Tony | 777777777 | 777777777 | Sales |
| Tony | 777777777 | 777777777 | Marketing |

# Example 5: Relational Join: Before Map Phase

**Employee**

| Name | SSN |
|---|---|
| Sue | 999999999 |
| Tony | 777777777 |

**Assigned Departments**

| EmpSSN | DepName |
|---|---|
| 999999999 | Accounts |
| 777777777 | Sales |
| 777777777 | Marketing |

*Key idea: Lump all the tuples together into one dataset*

Employee, Sue, 999999999
Employee, Tony, 777777777
Department, 999999999, Accounts
Department, 777777777, Sales
Department, 777777777, Marketing

*What is this for?*

# Example 5: Relational Join: Map Phase

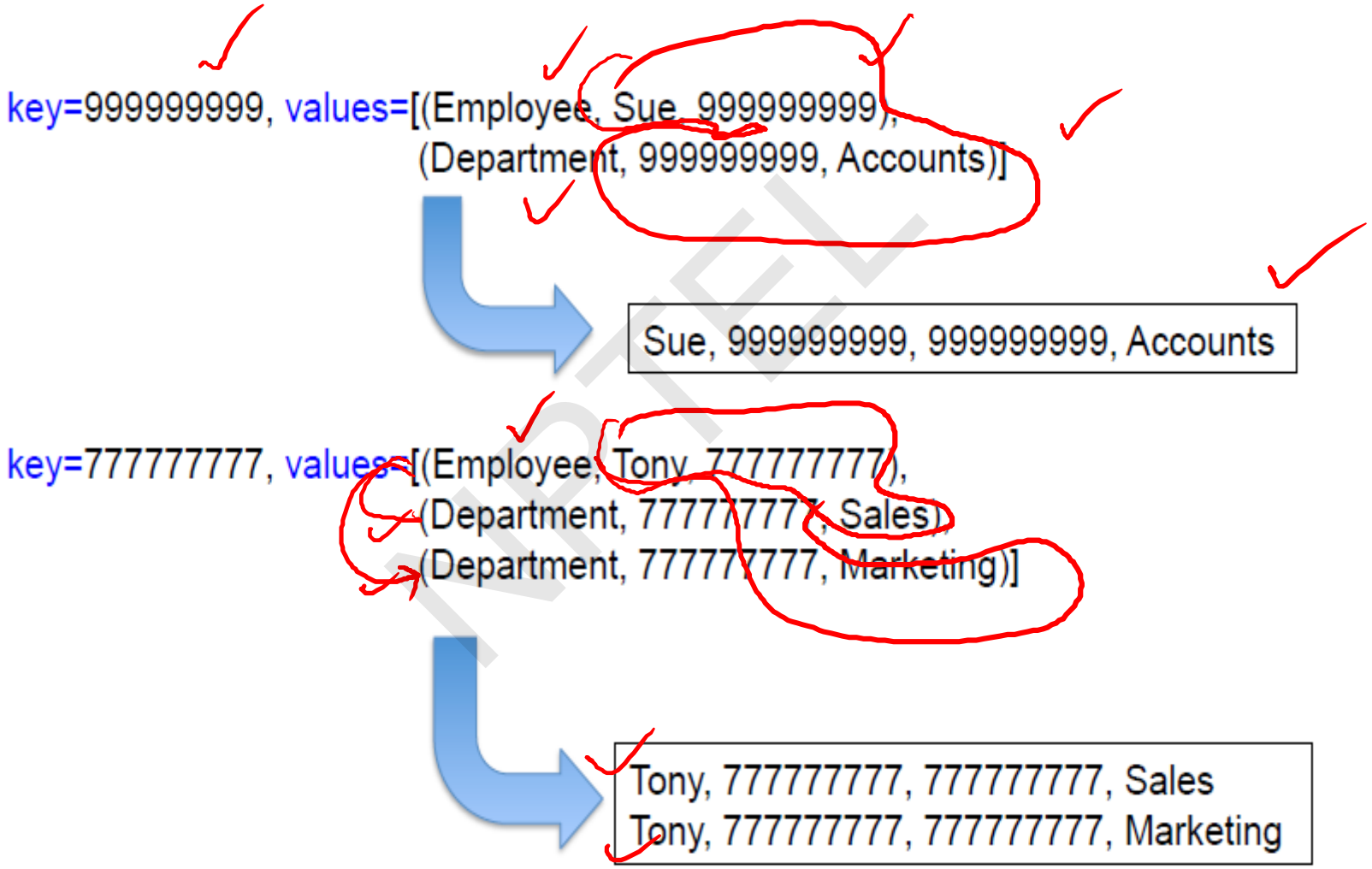
Employee, Sue, 999999999
Employee, Tony, 777777777
Department, 999999999, Accounts
Department, 777777777, Sales
Department, 777777777, Marketing

Join

key=999999999, value=(Employee, Sue, 999999999)
key=777777777, value=(Employee, Tony, 777777777)
key=999999999, value=(Department, 999999999, Accounts)
key=777777777, value=(Department, 777777777, Sales)
key=777777777, value=(Department, 777777777, Marketing)

*why do we use this as the key?*

# Example 5: Relational Join: Reduce Phase



# Example 5: Relational Join in MapReduce, again

Order(orderid, account, date)

1, aaa, d1
2, aaa, d2
3, bbb, d3

LineItem(orderid, itemid, qty)

1, 10, 1
1, 20, 3
2, 10, 5
2, 50, 100
3, 20, 1

*Map*

tagged with relation name

Order

1, aaa, d1 → 1 : "Order", (1,aaa,d1)
2, aaa, d2 → 2 : "Order", (2,aaa,d2)
3, bbb, d3 → 3 : "Order", (3,bbb,d3)

Line

1, 10, 1 → 1 : "Line", (1, 10, 1)
1, 20, 3 → 1 : "Line", (1, 20, 3)
2, 10, 5 → 2 : "Line", (2, 10, 5)
2, 50, 100 → 2 : "Line", (2, 50, 100)
3, 20, 1 → 3 : "Line", (3, 20, 1)

*Reducer for key 1*

"Order", (1,aaa,d1)
"Line", (1, 10, 1)
"Line", (1, 20, 3)

(1, aaa, d1, 1, 10, 1)
(1, aaa, d1, 1, 20, 3)

# Example 6: Finding Friends

- Facebook has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine).
- They also have lots of disk space and they serve hundreds of millions of requests everyday. They've decided to pre-compute calculations when they can to reduce the processing time of requests. **One common processing request is the "You and Joe have 230 friends in common" feature.**
- When you visit someone's profile, you see a list of friends that you have in common. This list doesn't change frequently so it'd be wasteful to recalculate it every time you visited the profile (sure you could use a decent caching strategy, but then we wouldn't be able to continue writing about mapreduce for this problem).
- We're going to use mapreduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

# Example 6: Finding Friends

- Assume the friends are stored as **Person->[List of Friends],** our friends list is then:

- A -> B C D
- B -> A C D E
- C -> A B D E
- D -> A B C E
- E -> B C D

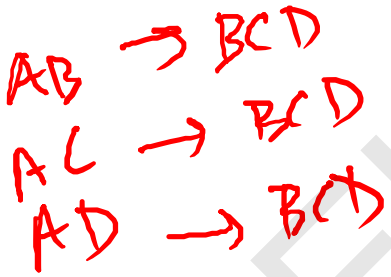
# Example 6: Finding Friends

**For map(A -> B C D) :**

(A B) -> B C D

(A C) -> B C D

(A D) -> B C D




**For map(B -> A C D E)** : (Note that A comes before B in the key)

(A B) -> A C D E

(B C) -> A C D E

(B D) -> A C D E

(B E) -> A C D E

# Example 6: Finding Friends

**For map(C -> A B D E) :**

(A C) -> A B D E

(B C) -> A B D E

(C D) -> A B D E

(C E) -> A B D E

**For map(D -> A B C E) :**

(A D) -> A B C E

(B D) -> A B C E

(C D) -> A B C E

(D E) -> A B C E

**And finally for map(E -> B C D):**

(B E) -> B C D

(C E) -> B C D

(D E) -> B C D

# Example 6: Finding Friends

- Before we send these key-value pairs to the reducers, we group them by their keys and get:

(A B) -> (A C D E) (B C D)

(A C) -> (A B D E) (B C D)

(A D) -> (A B C E) (B C D)

(B C) -> (A B D E) (A C D E)

(B D) -> (A B C E) (A C D E)

(B E) -> (A C D E) (B C D)

(C D) -> (A B C E) (A B D E)

(C E) -> (A B D E) (B C D)

(D E) -> (A B C E) (B C D)

# Example 6: Finding Friends

- Each line will be passed as an argument to a reducer.
- The **reduce function will simply intersect the lists of values** and output the same key with the result of the intersection.
- For example, **reduce((A B) -> (A C D E) (B C D))**

  will **output (A B) : (C D)**
- **and means that friends A and B have C and D as common friends.**

# Example 6: Finding Friends

- The result after reduction is:
- (A B) -> (C D)
- (A C) -> (B D)
- (A D) -> (B C)
- (B C) -> (A D E)
- (B D) -> (A C E)
- (B E) -> (C D)
- (C D) -> (A B E)
- (C E) -> (B D)
- (D E) -> (B C)

Now when D visits B's profile, we can quickly look up (B D) and see that they have three friends in common, (A C E).

# Reading

Jeffrey Dean and Sanjay Ghemawat,

**"MapReduce: Simplified Data Processing on Large Clusters"**

**http://labs.google.com/papers/mapreduce.html**