

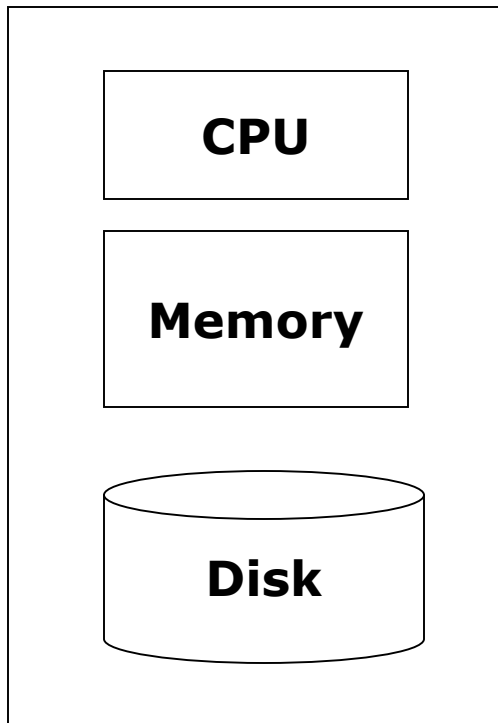
CS 345A  
Data Mining

---

MapReduce

# Single-node architecture

---



**Machine Learning, Statistics**

**“Classical” Data Mining**

---

# Commodity Clusters

---

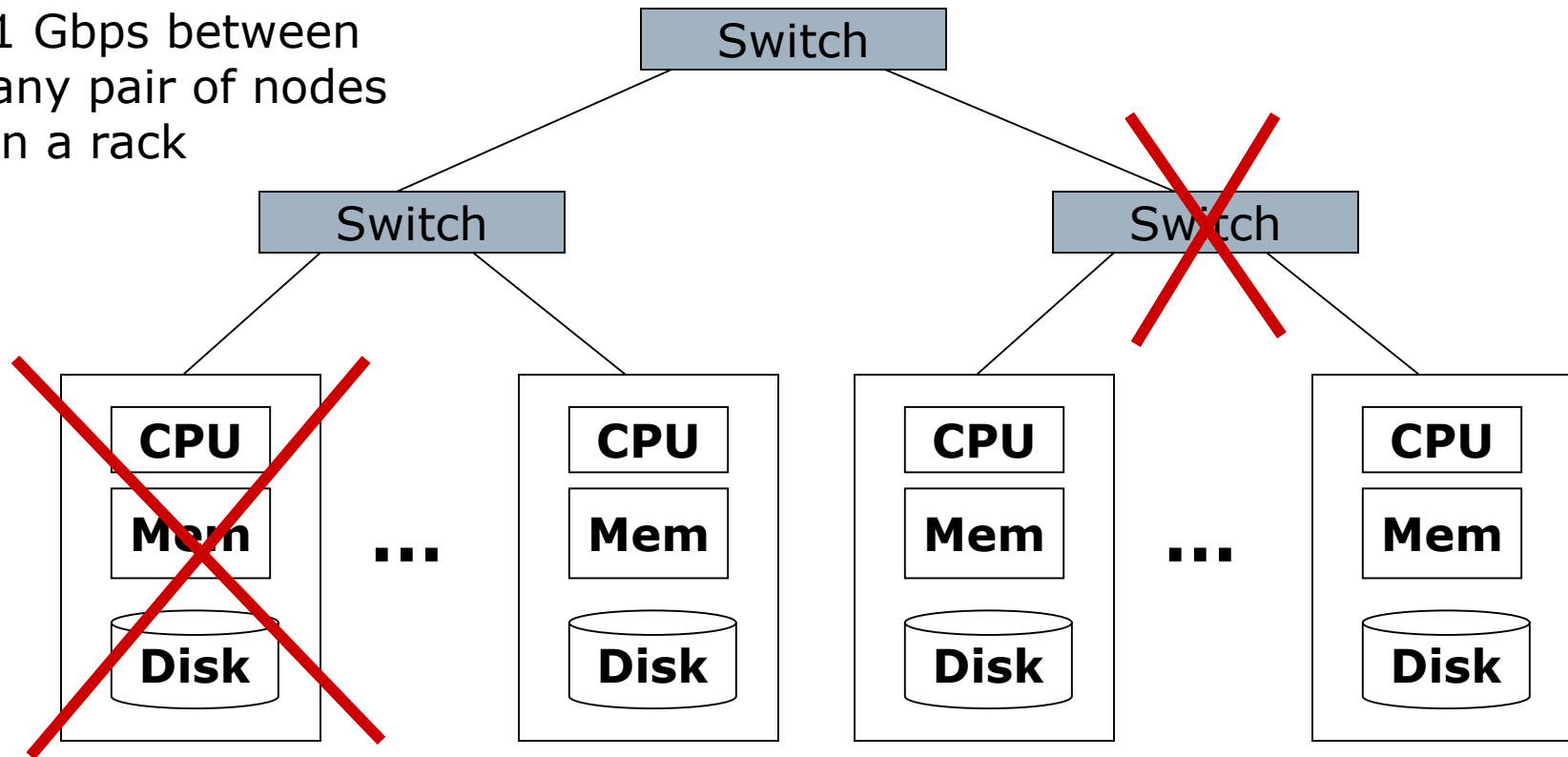
- Web data sets can be very large
    - Tens to hundreds of terabytes
  - Cannot mine on a single server (why?)
  - Standard architecture emerging:
    - Cluster of commodity Linux nodes
    - Gigabit ethernet interconnect
  - How to organize computations on this architecture?
    - Mask issues such as hardware failure
-

# Cluster Architecture

---

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack



Each rack contains 16-64 nodes

---

# Stable storage

---

- First order problem: if nodes can fail, how can we store data persistently?
  - Answer: Distributed File System
    - Provides global file namespace
    - Google GFS; Hadoop HDFS; Kosmix KFS
  - Typical usage pattern
    - Huge files (100s of GB to TB)
    - Data is rarely updated in place
    - Reads and appends are common
-

# 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

- a.k.a. 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
-

# Warm up: Word Count

---

- ❑ We have a large file of words, one word to a line
  - ❑ Count the number of times each distinct word appears in the file
  - ❑ Sample application: analyze web server logs to find popular URLs
-

# Word Count (2)

---

- ❑ Case 1: Entire file fits in memory
  - ❑ Case 2: File too large for mem, but all `<word, count>` pairs fit in mem
  - ❑ Case 3: File on disk, too many distinct words to fit in memory
    - `sort datafile | uniq -c`
-



# Word Count (3)

---

- To make it slightly harder, suppose we have a large corpus of documents
  - Count the number of times each distinct word occurs in the corpus
    - `words (docs/*) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one to a line
  - The above captures the essence of MapReduce
    - Great thing is it is naturally parallelizable
-

# MapReduce

---

- Input: a set of key/value pairs
  - User supplies two functions:
    - $\text{map}(k,v) \rightarrow \text{list}(k_1,v_1)$
    - $\text{reduce}(k_1, \text{list}(v_1)) \rightarrow v_2$
  - $(k_1,v_1)$  is an intermediate key/value pair
  - Output is the set of  $(k_1,v_2)$  pairs
-

# 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; value: an iterator over counts
```

```
  result = 0
```

```
  for each count v in values:
```

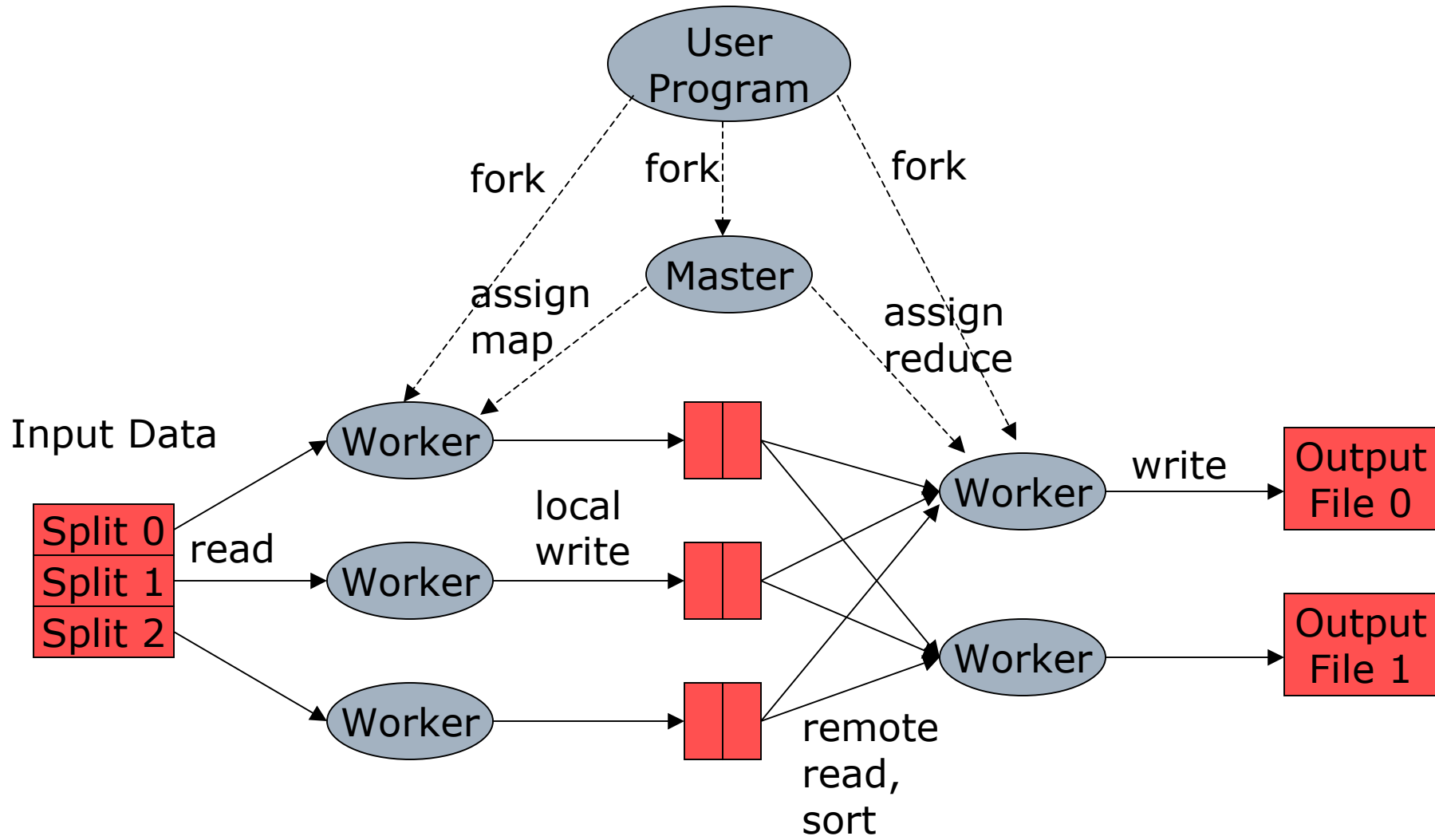
```
    result += v
```

```
  emit(result)
```

---

# Distributed Execution Overview

---



# Data flow

---

- Input, final output are stored on a distributed file system
    - Scheduler tries to schedule map tasks “close” to physical storage location of input data
  - Intermediate results are stored on local FS of map and reduce workers
  - Output is often input to another map reduce task
-

# Coordination

---

- Master data structures
    - Task status: (idle, in-progress, completed)
    - Idle tasks get scheduled as workers become available
    - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
    - Master pushes this info to reducers
  - Master pings workers periodically to detect failures
-

# Failures

---

## □ Map worker failure

- Map tasks completed or in-progress at worker are reset to idle
- Reduce workers are notified when task is rescheduled on another worker

## □ Reduce worker failure

- Only in-progress tasks are reset to idle

## □ Master failure

- MapReduce task is aborted and client is notified
-

# How many Map and Reduce jobs?

---

- M map tasks, R reduce tasks
  - Rule of thumb:
    - Make M and R much larger than the number of nodes in cluster
    - One DFS chunk per map is common
    - Improves dynamic load balancing and speeds recovery from worker failure
  - Usually R is smaller than M, because output is spread across R files
-



# Combiners

---

- Often a map task will produce many pairs of the form  $(k, v_1), (k, v_2), \dots$  for the same key  $k$ 
    - E.g., popular words in Word Count
  - Can save network time by pre-aggregating at mapper
    - $\text{combine}(k_1, \text{list}(v_1)) \rightarrow v_2$
    - Usually same as reduce function
  - Works only if reduce function is commutative and associative
-

# Partition Function

---

- Inputs to map tasks are created by contiguous splits of input file
  - For reduce, we need to ensure that records with the same intermediate key end up at the same worker
  - System uses a default partition function e.g.,  $\text{hash}(\text{key}) \bmod R$
  - Sometimes useful to override
    - E.g.,  $\text{hash}(\text{hostname}(\text{URL})) \bmod R$  ensures URLs from a host end up in the same output file
-

# Exercise 1: Host size

---

- Suppose we have a large web corpus
  - Let's look at the metadata file
    - Lines of the form (URL, size, date, ...)
  - For each host, find the total number of bytes
    - i.e., the sum of the page sizes for all URLs from that host
-

# Exercise 2: Distributed Grep

---

- Find all occurrences of the given pattern in a very large set of files

# Exercise 3: Graph reversal

---

- Given a directed graph as an adjacency list:

src1: dest11, dest12, ...

src2: dest21, dest22, ...

- Construct the graph in which all the links are reversed
-

# Exercise 4: Frequent Pairs

---

- Given a large set of market baskets, find all frequent pairs
  - Remember definitions from Association Rules lectures

# Hadoop

---

- An open-source implementation of Map Reduce in Java
  - Uses HDFS for stable storage

- Download from:

<http://lucene.apache.org/hadoop/>

---

# Reading

---

- Jeffrey Dean and Sanjay Ghemawat,  
**MapReduce: Simplified Data Processing  
on Large Clusters**

<http://labs.google.com/papers/mapreduce.html>

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, **The Google File System**

<http://labs.google.com/papers/gfs.html>

---