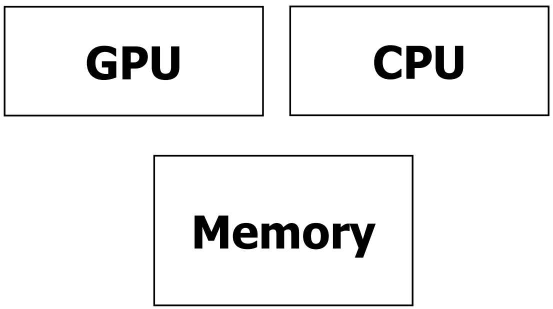
Map-Reduce (Part I)

Professor Wei-Min Shen

University of Southern California

Thanks for source slides and material to: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets (http://www.mmds.org)

Single Node Architecture



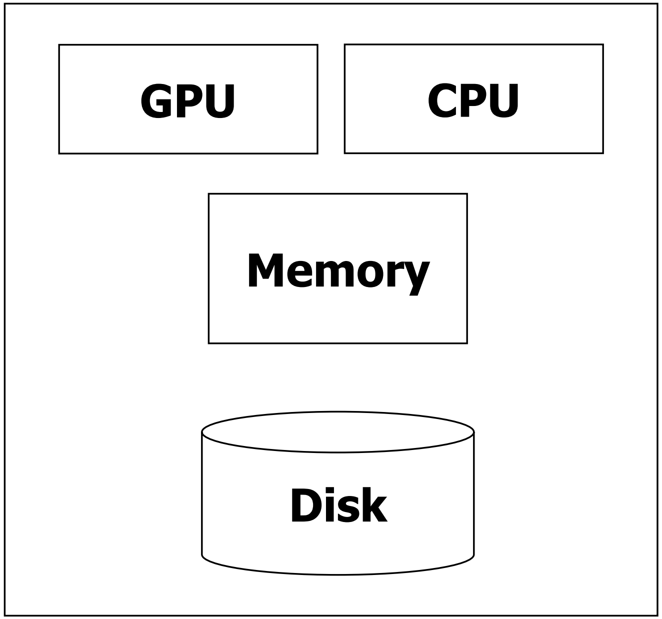
Machine Learning, Statistics

Computational Model of CPU/GPU and memory

What if the data can't fit in memory at the same time?

swap from memory to disk

Single Node Architecture



Machine Learning, Statistics

"Classical" Data Mining algorithm

Not sufficient !

communite to disk is slow

Motivation: Google Example

Crawling and indexing the web pages

- 10 billion web pages
- Average size of webpage = 20 KB

  ➪ 10 billion * 20KB = 200 TB
- The data is stored on a single disk and tends to be processed in CPU
- One computer reads 50 MB/sec from disk (disk read bandwidth)

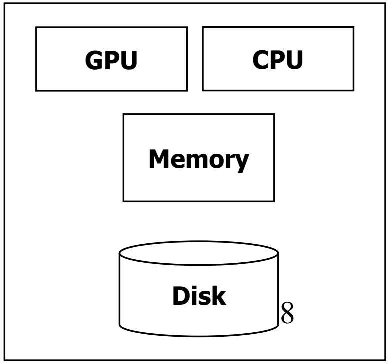
  ➪ Time to read = 4 million seconds ≈ 46 days
- Even longer to do useful things with the data

Motivation: Google Example

Crawling and indexing the web pages

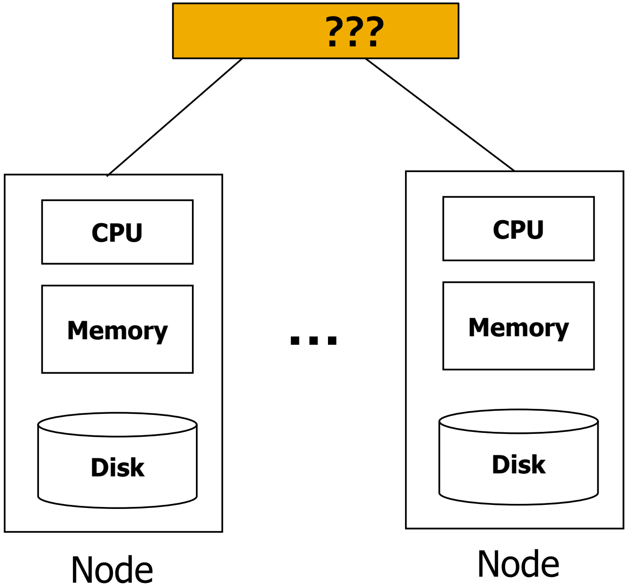
- Split the data into chunks
- Store and process the data in parallel in multiple disks and CPUs
- e.g., 1,000 disks and CPUs

  ⇨ Time to read = 4 million seconds / 1,000 = 4,000 seconds
- Cluster Computing



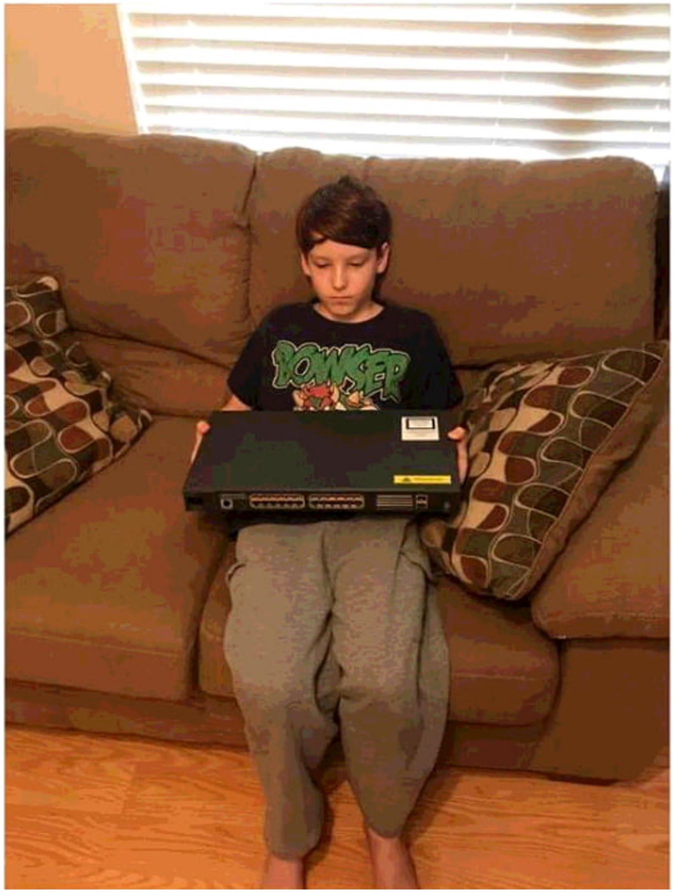
↑ single node architecture
↓
Cluster Architecture

Each **rack** contains 16-64 **nodes**
e.g., commodity Linux nodes

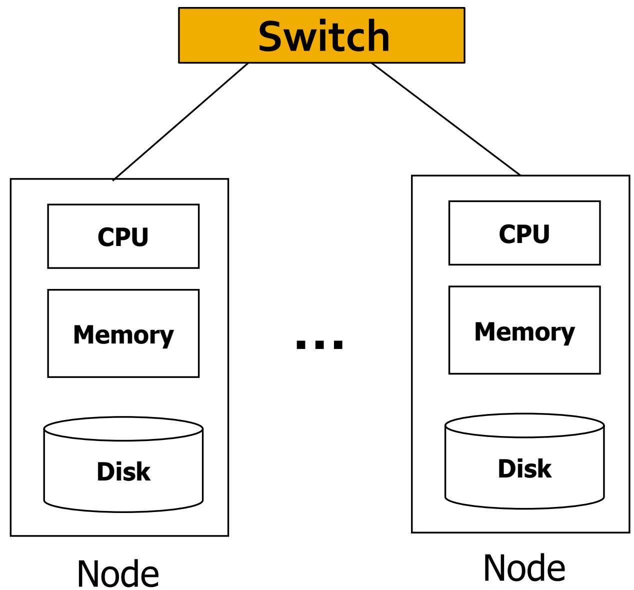


Cluster Architecture

My son said he wanted a switch for his birthday.

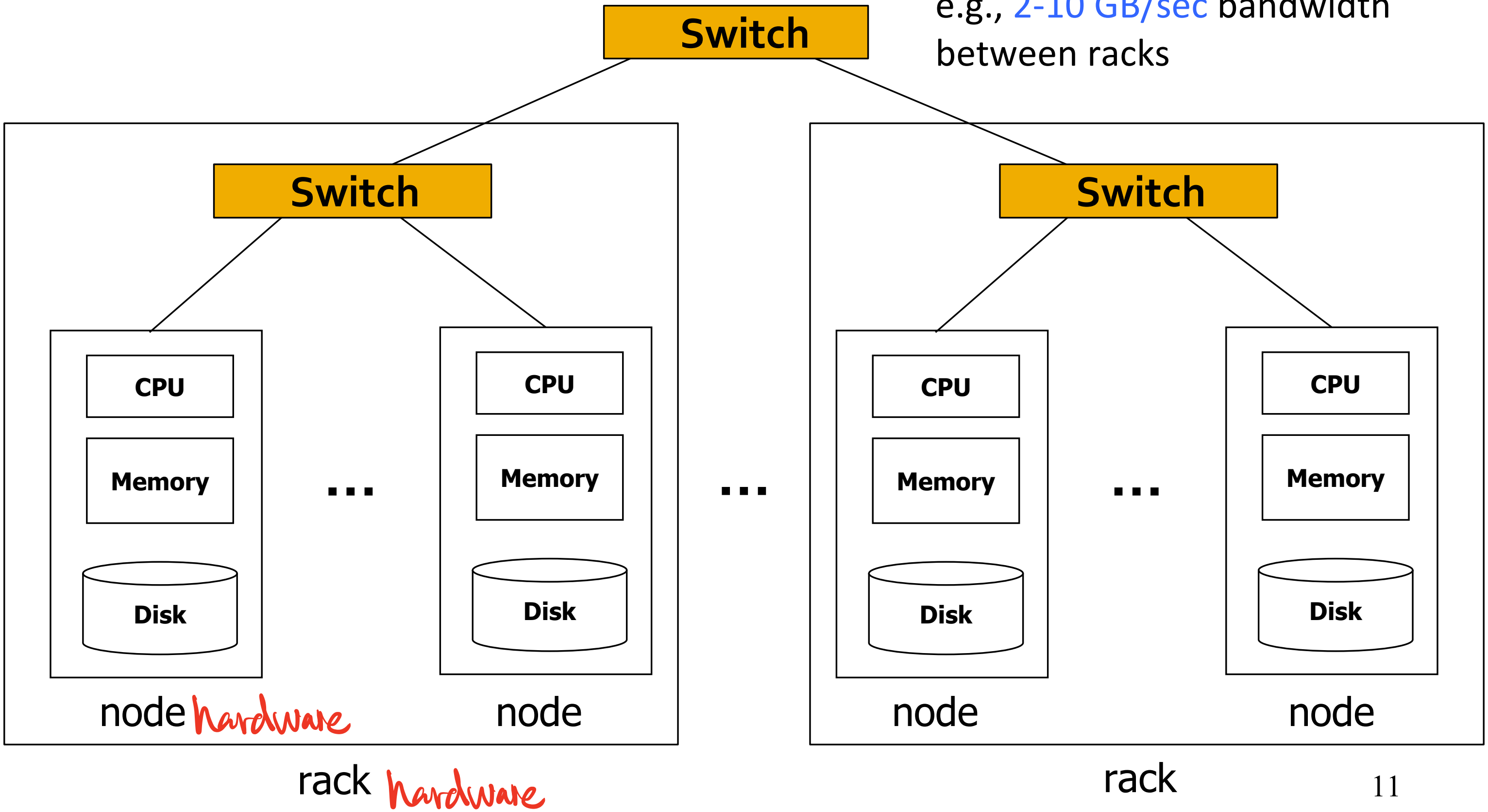


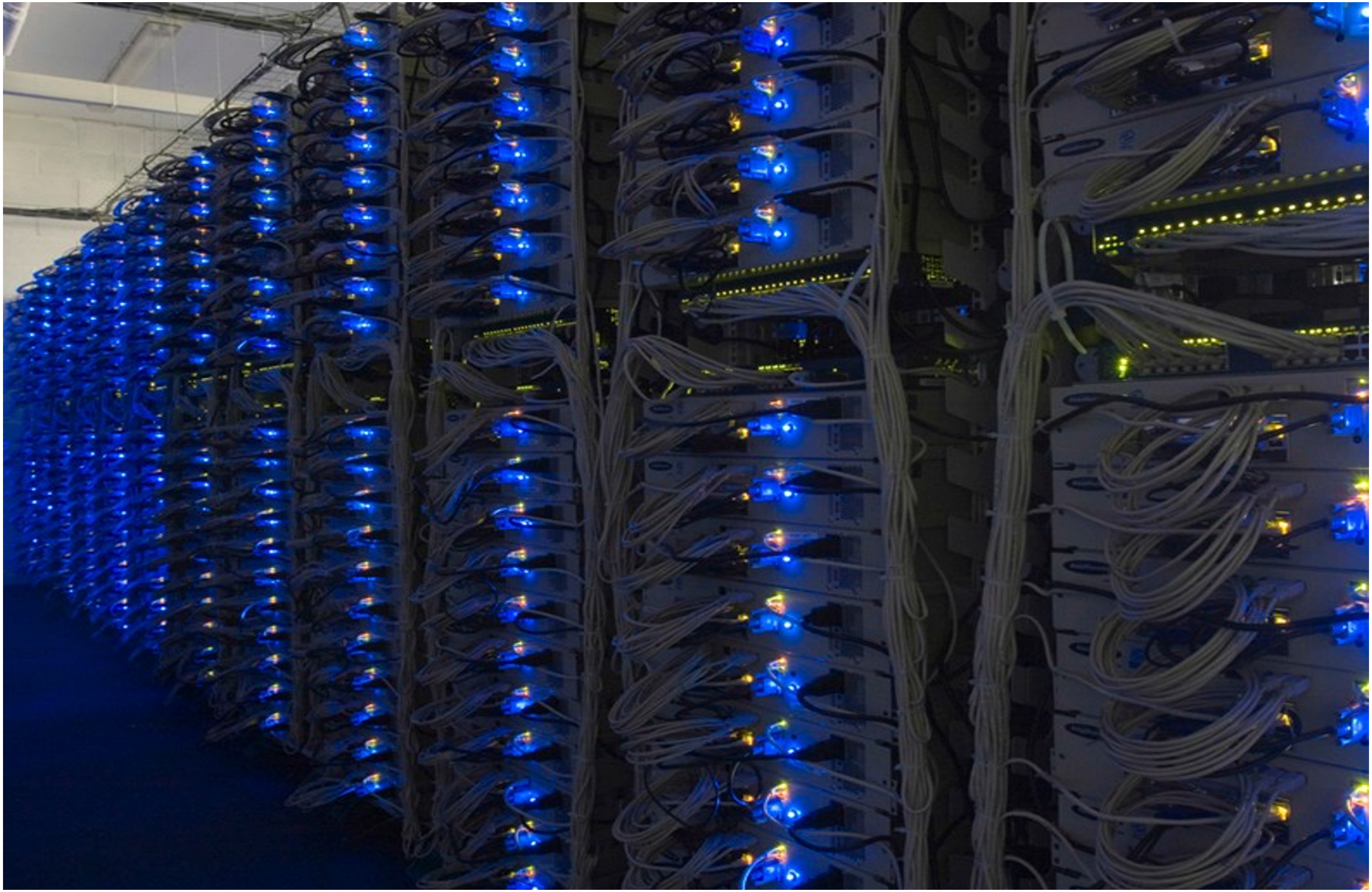
Switch - connect nodes
e.g., 1 GB/sec bandwidth between any pair of nodes in a rack



Cluster Architecture

Backbone switch - connect racks
e.g., 2-10 GB/sec bandwidth
between racks





In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Cluster Computing Challenges I

- **Node failures**

  e.g., one server can stay up 3 years (1,000 days)

  1,000 servers in cluster ⇒ 1 failure/day

  1M servers in cluster ⇒ 1,000 failures/day

➪ Store data **persistently** and keep it available when nodes fail

➪ Deal with node failures **during a long running computation**

Cluster Computing Challenges II

- **Network bottleneck**

  e.g., network bandwidth = 1 GB/sec

  moving 10TB data takes approximately 1 day

➪ A framework that does not move data around so much while it's doing computation

but move computer to data

Cluster Computing Challenges III

- **Distributed/parallel programming is hard**

⇨ A simple model that hides most of the complexity

Map-Reduce

- **Map-Reduce addresses the challenges**
  - **Node failure**

    Store data redundantly on multiple nodes
  - **Network bottleneck**

    Move computation close to data to minimize data movement
  - **Distributed programming**

    Map function and Reduce functions

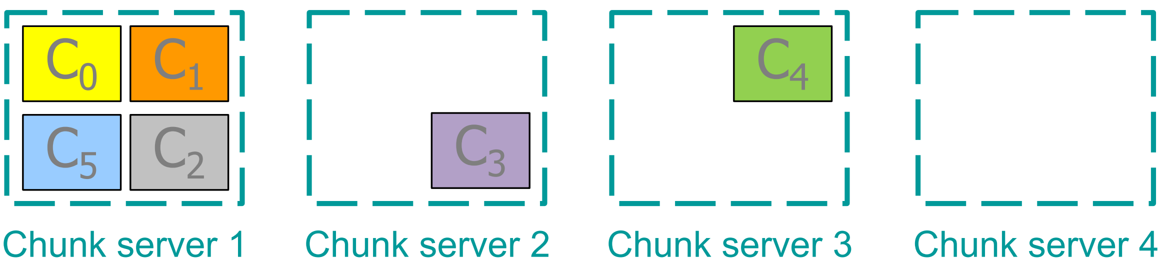
Redundant Storage Infrastructure

- **Distributed File System** too big to store
  - Store data multiple times across a cluster
  - Provide global file namespace
  - E.g., Google GFS; Hadoop HDFS

- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

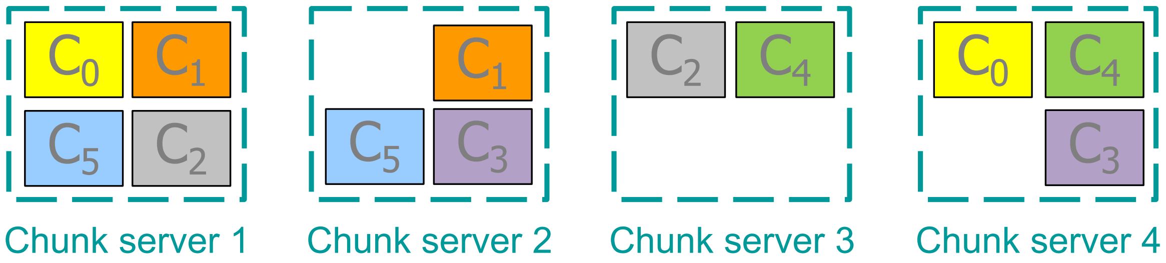
- Data is kept in "chunks" spread across machines (**chunk servers**)
  - Software … hardware
- Each chunk replicated on different machines
- E.g., 4 chunk servers , file 1 is split into 6 chunks



Not sufficient ! Need multiple copies of each chunk.

Distributed File System

- Data is kept in "chunks" spread across machines (**chunk servers**)
- Each chunk replicated on different machines
- E.g., 4 chunk servers , file 1 is split into 6 chunks

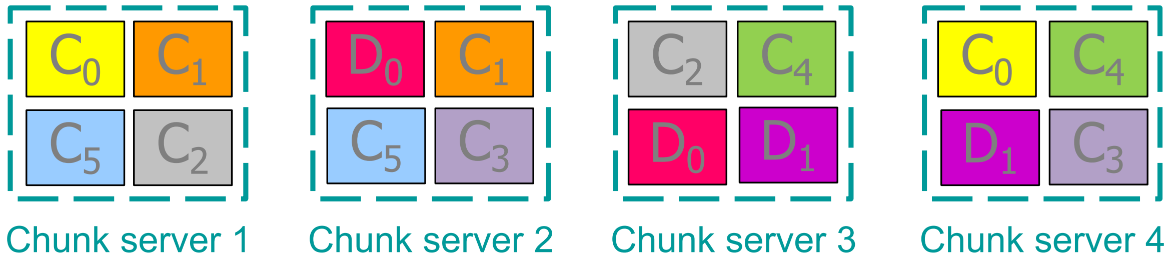


Each chunk is replicated twice, and the replicas of a chunk are never on the same chunk server.

Distributed File System

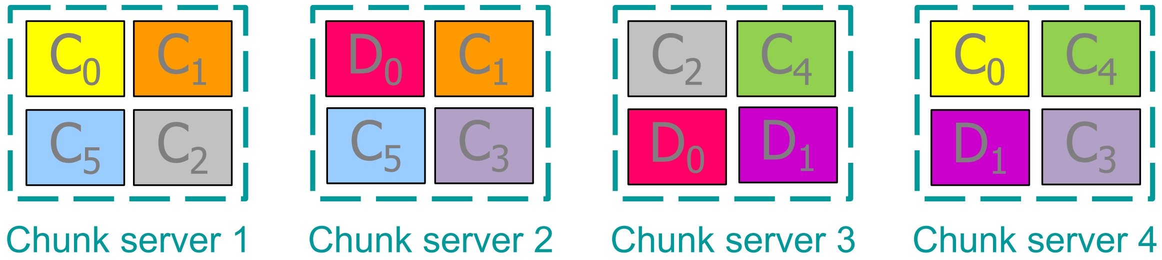
- Data is kept in “chunks” spread across machines (**chunk servers**)
- Each chunk replicated on different machines
- E.g., 4 chunk servers , file 1 is split into 6 chunks

  Another file 2 has 2 chunks, D_0 and D_1



Distributed File System

- Data is kept in "chunks" spread across machines (**chunk servers**)
- Each chunk replicated on different machines



Chunk servers also serve as compute servers

Bring computation to data!

2. Components of Distributed File System

- (1) **Chunk servers**
    - File is split into contiguous chunks (typically 16-64 MB)
    - Each chunk is replicated (usually 2x or 3x)
    - Try to keep replicas in different racks
        each rack is a server
        - In case that the switch on a rack can fail and entire rack becomes inaccessible

Components of Distributed File System

- **Master node**
  - a.k.a. Name Node in Hadoop HDFS
  - Stores metadata about where files are stored

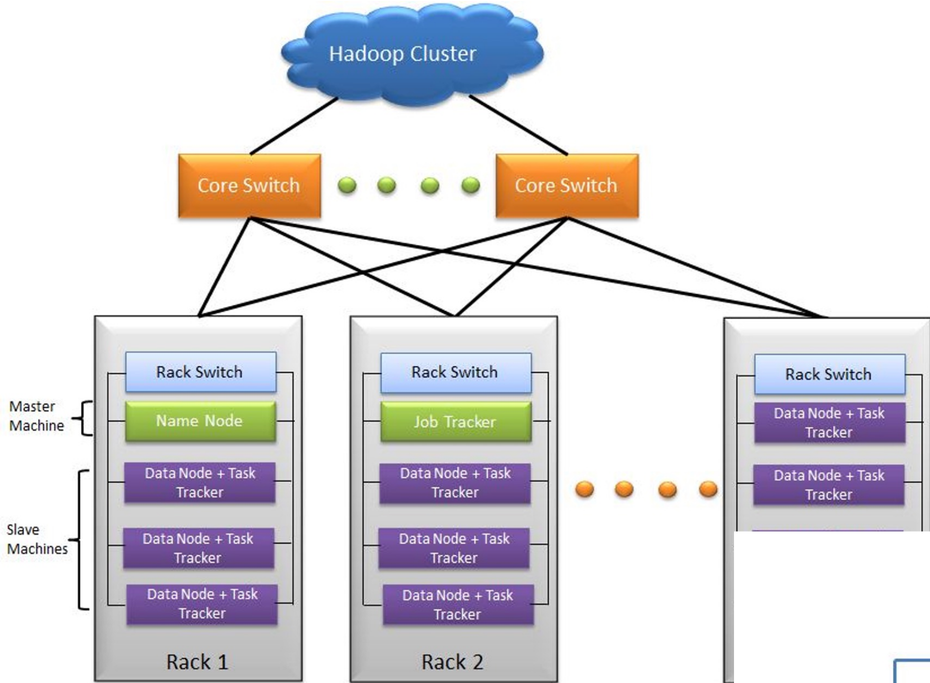
    e.g., it will know that file-1 is divided into 6 chunks, the locations of each of the 6 chunks and the locations of the replicas
  - Might be replicated
    - Otherwise it might become a single point of failure

Components of Distributed File System

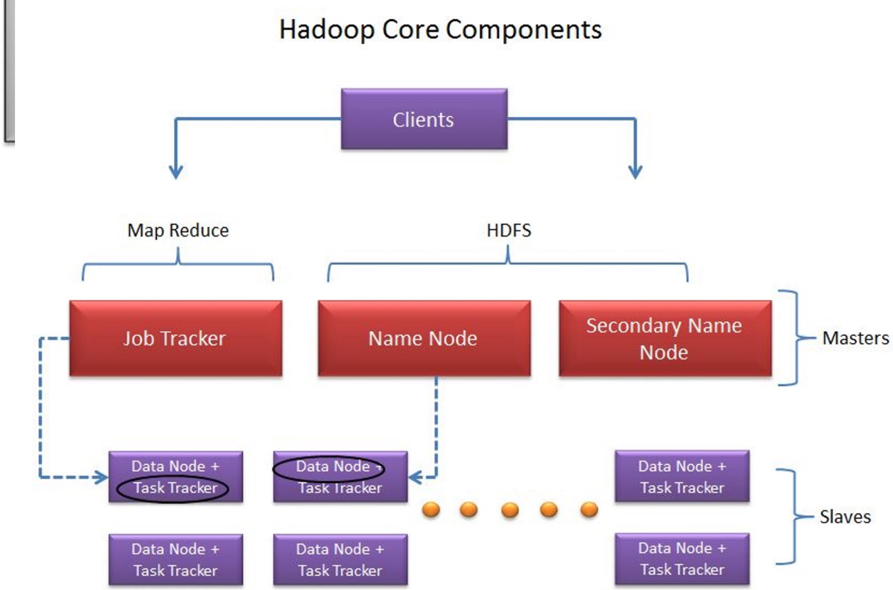
13)

- **Client library for file access**
  - Talks to master to find chunk servers that store the chunks

    then
  - Connects directly to chunk servers to access data without going through the master node



Hadoop Cluster



http://saphanatutorial.com/hadoop-cluster-architecture-and-core-components/

Programming Model: Map-Reduce

- **Warm-up task**
  - We have a huge text document
  - Count the number of times each distinct word appears in the file
  - Sample application:
    - Analyze web server logs to find popular URLs

- A thought experiment: ☺ ☺ ☺
  - You and your friends are given today's New York Times newspaper
  - How would you do the above tasks?

Task: Word Count

- **Case 1**
  - File is too large for memory, but all <word, count> pairs fit in memory
  - Solution:
    - Use a hash table (word -> count) to store the number of times that word appears
    - Make a simple sweep through the file, and will have the word count pairs for every unique word.

Task: Word Count

- **Case 2**
  - Even the <word, count> pairs do not fit in memory
  - Solution:
    - **Map-Reduce**

Map-Reduce: Overview

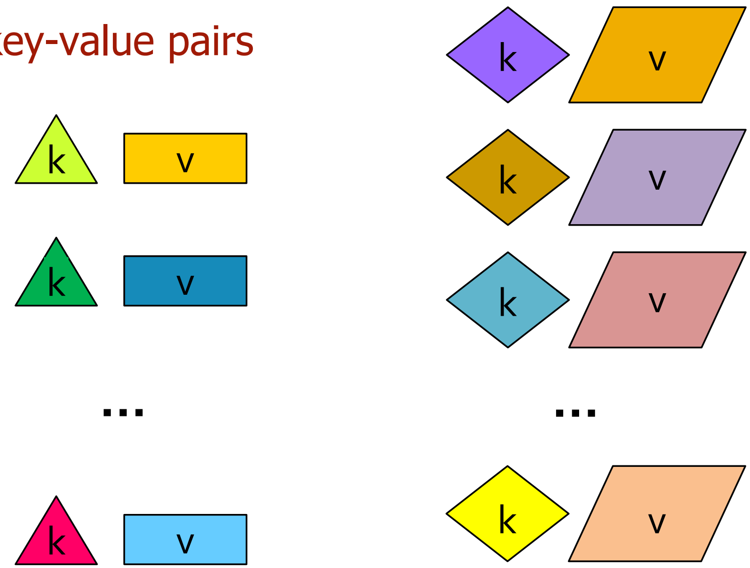
- **Map**
  - Divide the file into many "records"
  - Extract something (e.g., word) from each record (as key)
  - Output one or multiple things for each record
- **Group by key**
  - Sort and shuffle
- **Reduce**
  - Aggregate, summarize, filter or transform
  - Output the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

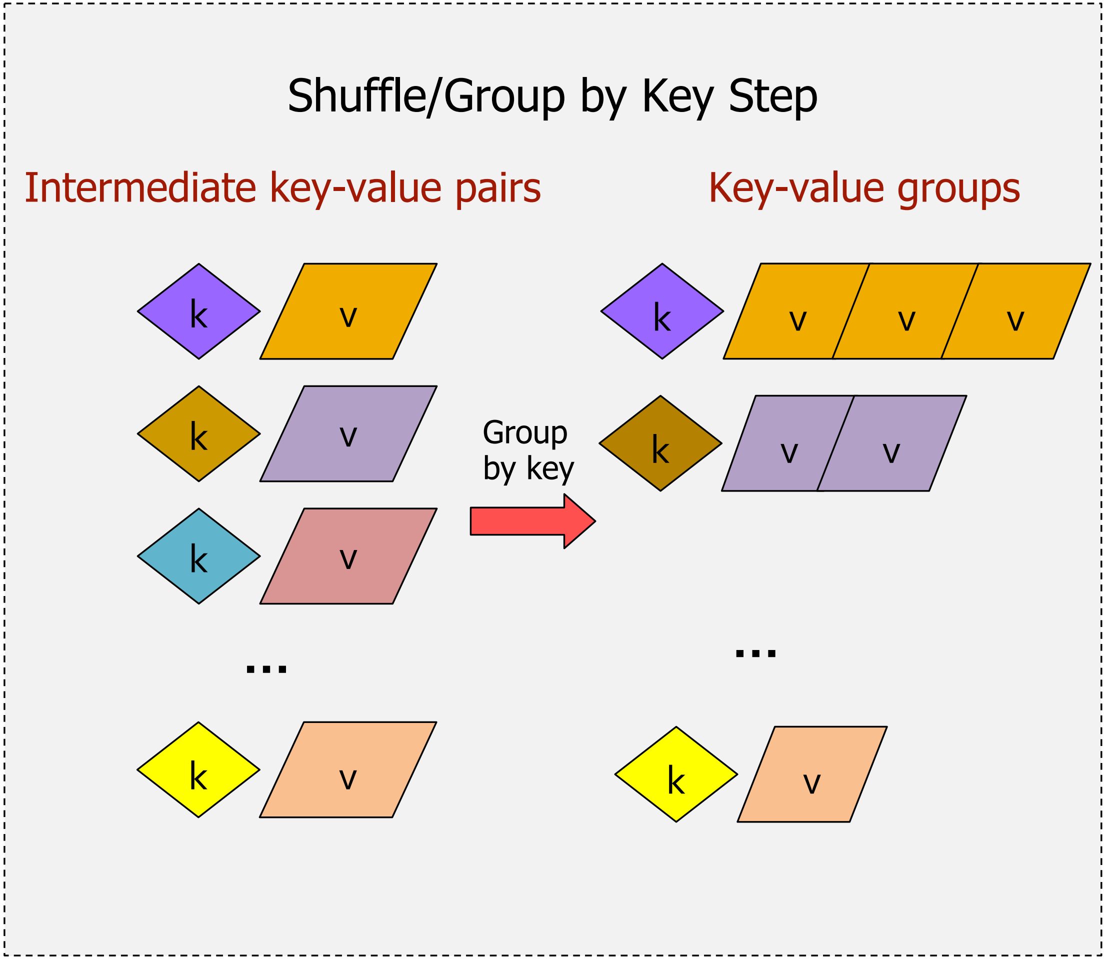
Map-Reduce: The <u>Map</u> Step

Intermediate key-value pairs

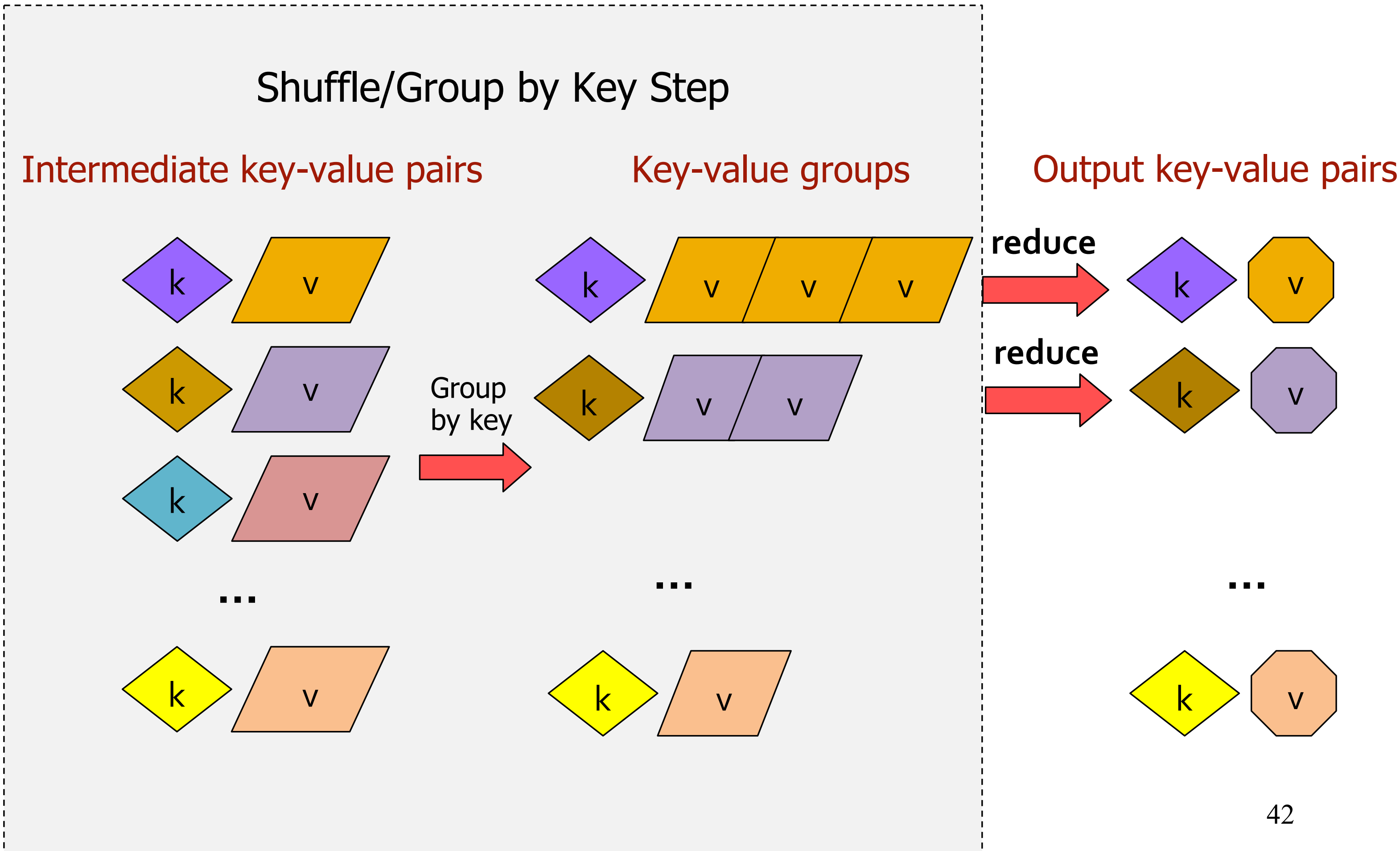
Input: key-value pairs



Map-Reduce: The Reduce Step



Map-Reduce: The Reduce Step



More formally…

- **Input:** a set of key-value pairs
- Programmer need to specify two methods:
  - **Map(k, v)** → <k’, v’>*
    - Takes a key-value pair and outputs a set of key-value pairs

      E.g., key is the filename, value is a single line in the file
    - There is one Map call for every (k, v) pair

  - **Reduce(k’, <v’>*)** → <k’, v’’>*
    - **All values v’ with same key k’ are reduced together**
    - There is one Reduce function call per unique key k’

eg 1.

Map-Reduce: Word Count

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 2)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Map-Reduce: Word Count

Provided by the programmer

Provided by the programmer

if file is too big

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 2)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Map-Reduce: Word Count
pseudo-code

```
map(key, value):
  // key: document name; value: text of the 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(key, result)
```

eg2:

Map-Reduce example 2

Build an inverted index

- (ID, content) => (content, List[IDs])
- Application: Search Engines, supporting full text searches

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)
…

Map task:

What intermediate (key, value) pairs produced?

Reduce task:

?

Map-Reduce example 2

Build an inverted index

- (Location, content) => (content, location)
- Application: Search Engines, supporting full text searches

Input (key, value):

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)
…

Map task:

For each word in input value, emit (word, tweet_ID) as intermediate (key, value) pair

Reduce task

Reduce function emits key and list of tweet_IDs associated with that key

eg3:

Map-Reduce example 3

Social Network Analysis: Count Friends

- In a social network (Facebook, Instagram, etc.), how many friends does each person have?

Input (key, value)	**Desired Output**
(Jim, Sue)	(Jim, 3)
(Sue, Jim)	(Lin, 2)
(Lin, Joe)	(Sue, 1)
(Joe, Lin)	(Kai, 1)
(Jim, Kai)	(Joe, 1)
(Kai, Jim)	
(Jim, Lin)	
(Lin, Jim)	

Map task: ? Reduce task: ?

Map-Reduce example 3

Social Network Analysis: Count Friends

- In a social network (Facebook, Instagram, etc.), how many friends does each person have?

Input (key, value)

focus on the first one

Input (key, value)						Output
(Jim, Sue)	MAP	Jim, 1	SHUFFLE /GROUP	Jim, (1, 1, 1)	REDUCE	(Jim, 3)
(Sue, Jim)		Sue, 1		Lin, (1, 1)		(Lin, 2)
(Lin, Joe)		Lin, 1		Sue, (1)		(Sue, 1)
(Joe, Lin)		Joe, 1		Kai, (1)		(Kai, 1)
(Jim, Kai)		Jim, 1		Joe, (1)		(Joe, 1)
(Kai, Jim)		Kai, 1				
(Jim, Lin)		Jim, 1				
(Lin, Jim)		Lin, 1				

eg4:

Map-Reduce example 4

Integers divisible by 7

- Design a Map-Reduce algorithm that takes a very large file of integers and produces as output all unique integers from the original file that are evenly divisible by 7

- The large file of integers cannot fit in the memory of node

Map task:

Each Map task gets a chunk of the file of integers and processes it …

Reduce task:

?

Map-Reduce example 4

Integers divisible by 7

- Design a Map-Reduce algorithm that takes a very large file of integers and produces as **output all unique integers** from the original file that are **evenly divisible by 7**
- The large file of integers cannot fit in the memory of node

```
map(key, value_list):
      for v in value_list:
            emit(v, 1)
```

X

Map-Reduce example 4
Integers divisible by 7

- Design a Map-Reduce algorithm that takes a very large file of integers and produces as **output all unique integers** from the original file that are **evenly divisible by 7**

- The large file of integers cannot fit in the memory of node

Solution:

```
map(key, value_list):
      // Eliminate duplicates
      for a unique v in value_list:
            emit(v, 1)

reduce(key, values):
      if (v % 7) == 0 :
            emit (key, 1)
```

Map-Reduce example 4
Integers divisible by 7

- Design a Map-Reduce algorithm that takes a very large file of integers and produces as **output all unique integers** from the original file that are **evenly divisible by 7**
- The large file of integers cannot fit in the memory of node

Solution 2: better

```
map(key, value_list):
        for v in valuelist:
                if (v % 7) == 0 :
                        emit(v, 1)

reduce(key, values):
        // Eliminate duplicates
        emit (key, 1)
```

Question: Why check whether divisible by 7 in the Map task rather than the Reduce task?

Reduce communication: send less data over network