CS425: Algorithms for Web Scale Data

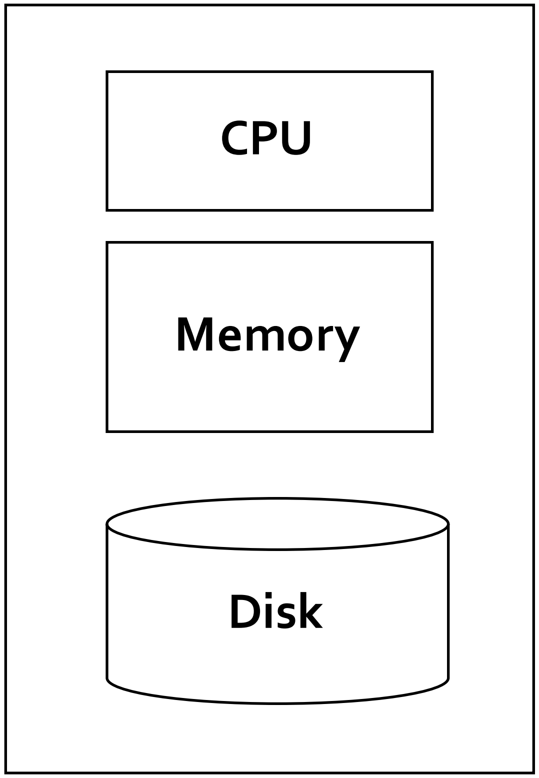
Lecture 5: MapReduce

Most of the slides are from the Mining of Massive Datasets book.
These slides have been modified for CS425. The original slides can be accessed at: www.mmds.org

MapReduce

- **Challenges of large scale computing:**
  - How to distribute computation?
  - Distributed/parallel programming is hard
  - Need to consider parallelism, efficiency, communication, synchronization, reliability.
- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Single Node Architecture



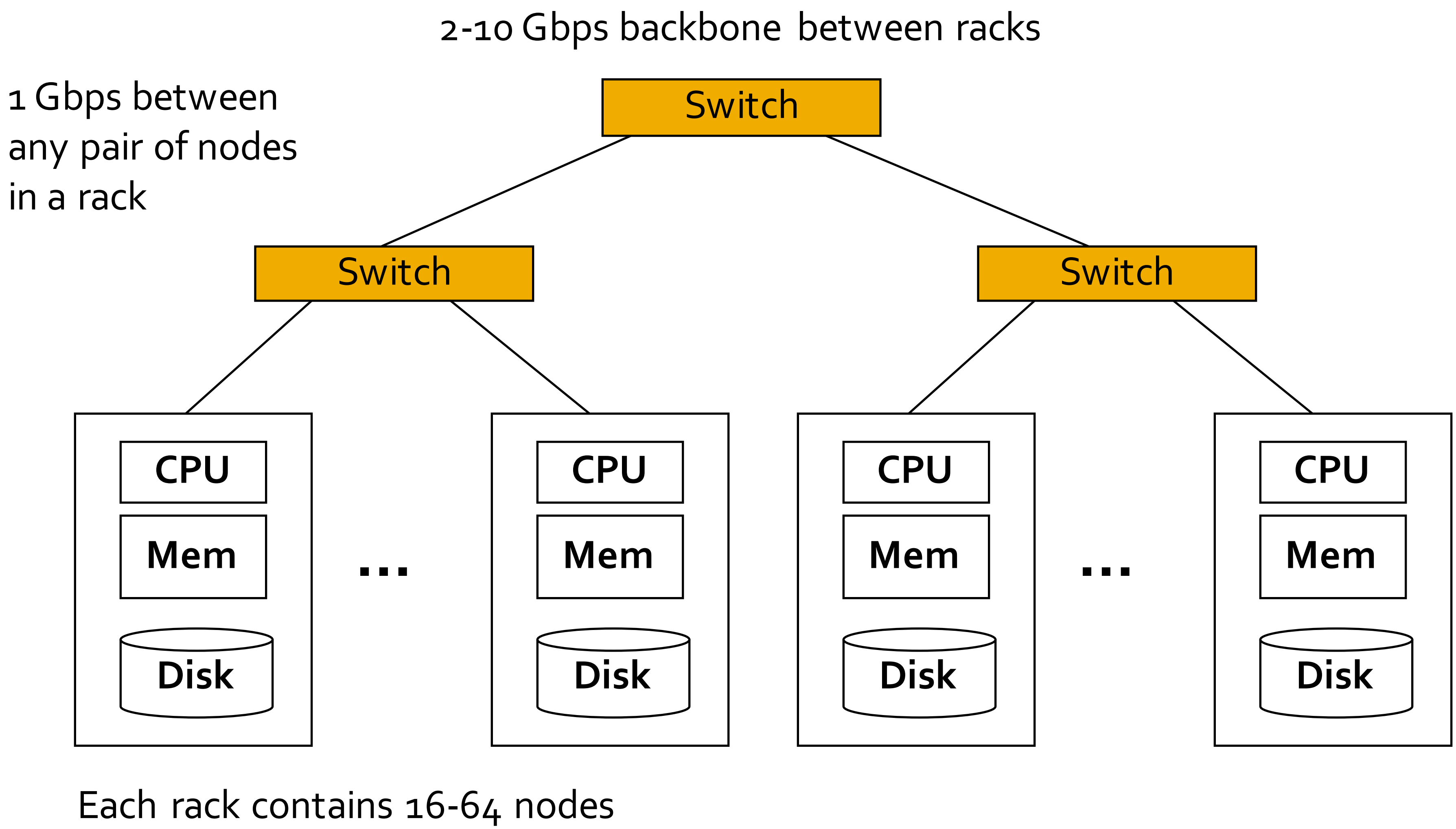
Machine Learning, Statistics

"Classical" Data Mining

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 80-160 MB/sec from disk
  - > 1 month to read the web
- Many hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



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

Large-scale Computing

- **Large-scale computing** for **data mining** problems on **commodity hardware**
- **Challenges:**
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Idea and Solution

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Map-reduce** addresses these problems
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - Map-Reduce

Storage Infrastructure

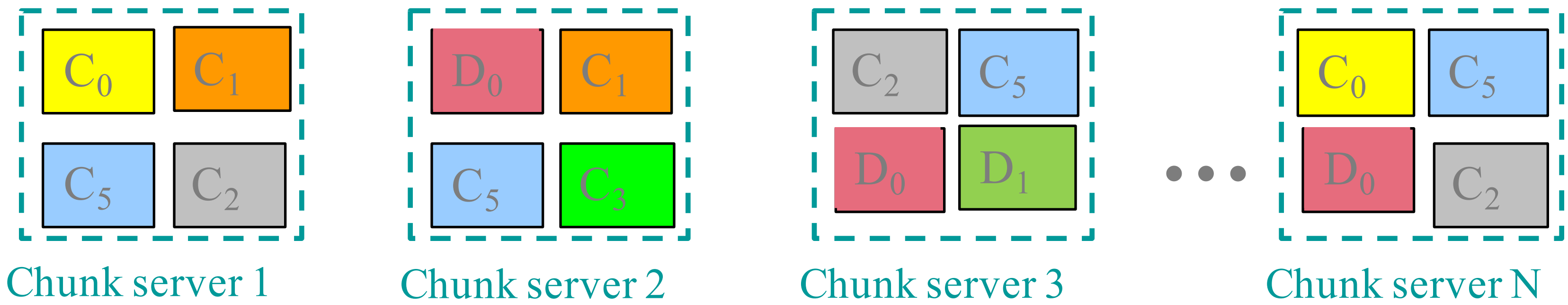
- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (TBs)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-128MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Distributed File System

- **Reliable distributed file system**
- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

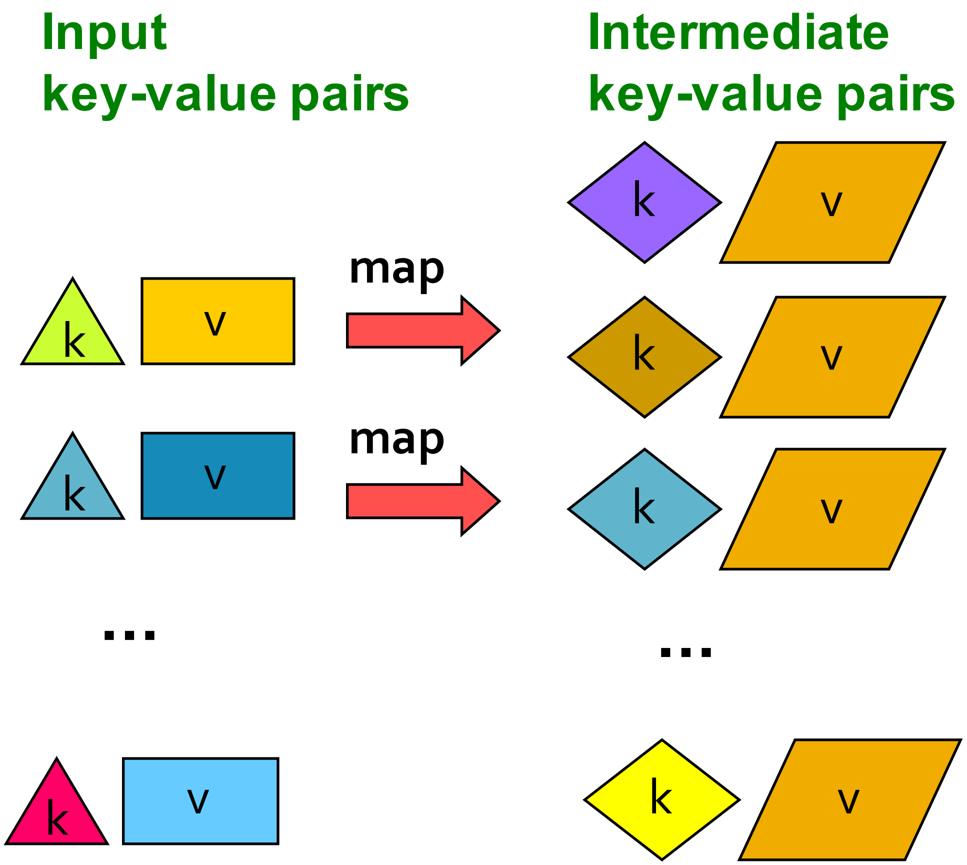
Chunk servers also serve as compute servers

MapReduce: Overview

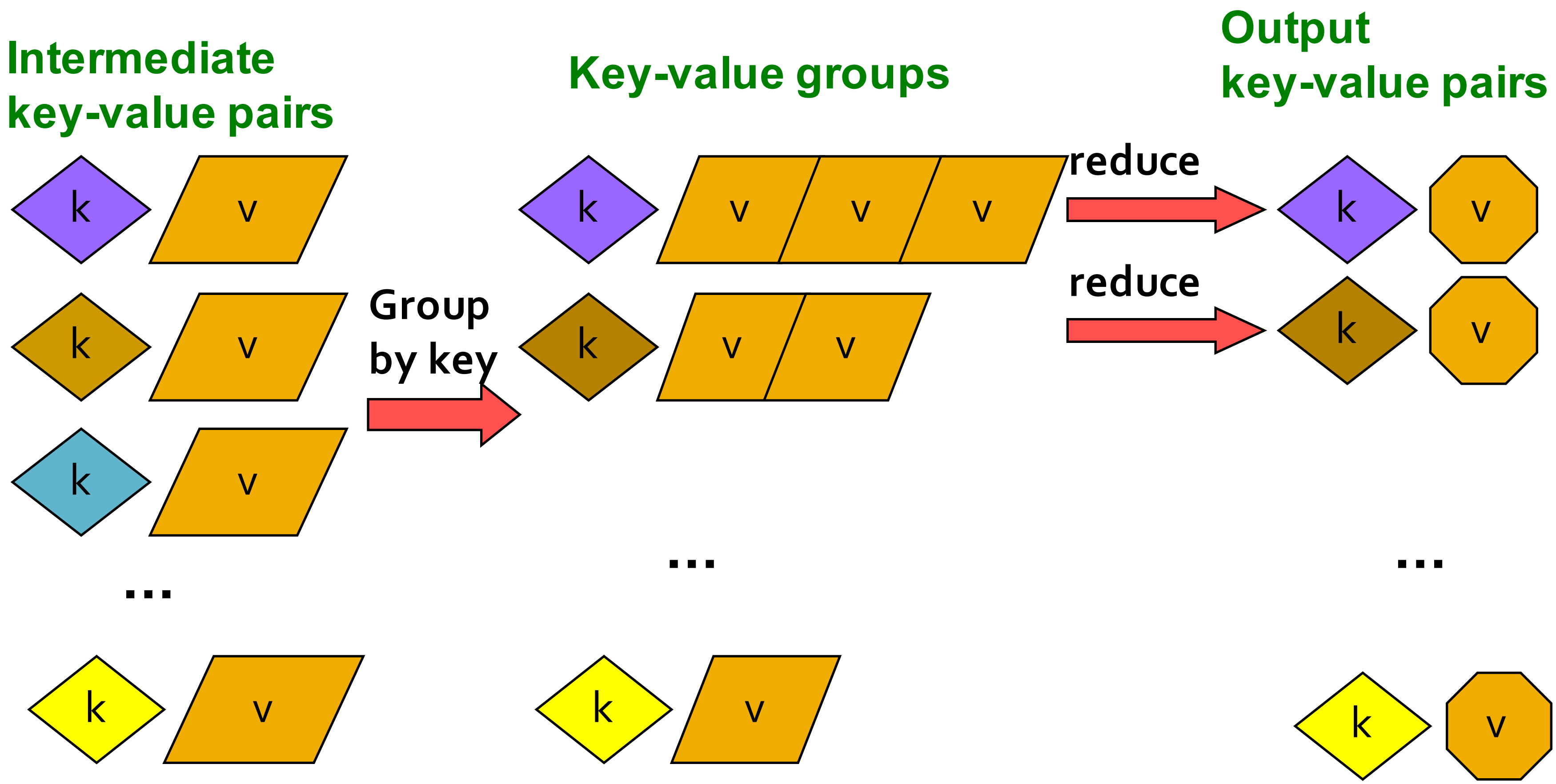
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

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

MapReduce: The Map Step



MapReduce: The Reduce Step



More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies 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 and processed in *v’* order**
    - There is one Reduce function call per unique key *k’*

Programming Model: MapReduce

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

MapReduce: Word Counting

Provided by the programmer

MAP:
Read input and produce 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, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Word Count Using MapReduce

```
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)
```

Map-Reduce: Environment

Map-Reduce environment takes care of:

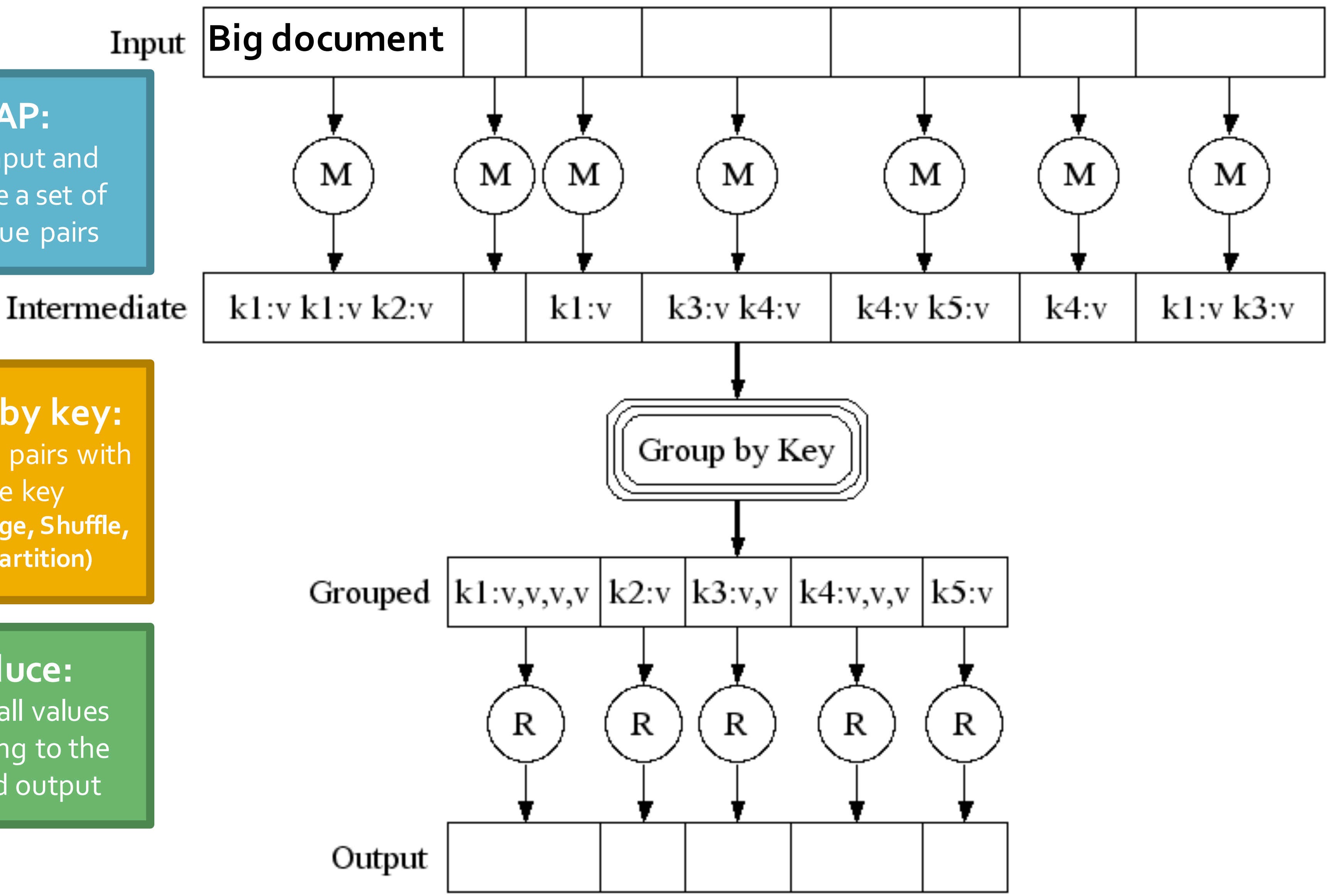
- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Map-Reduce: A diagram

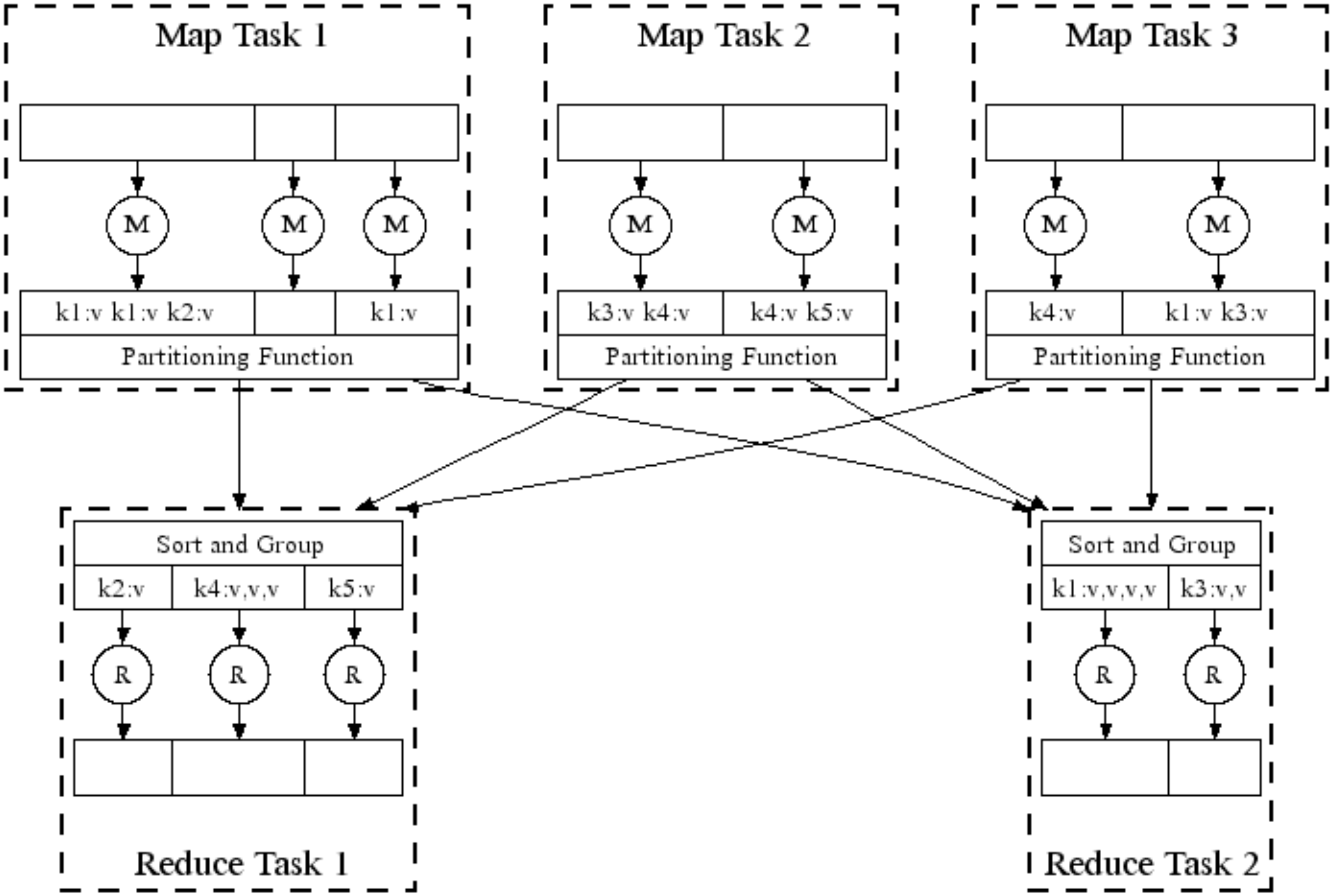
MAP:
Read input and produce a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



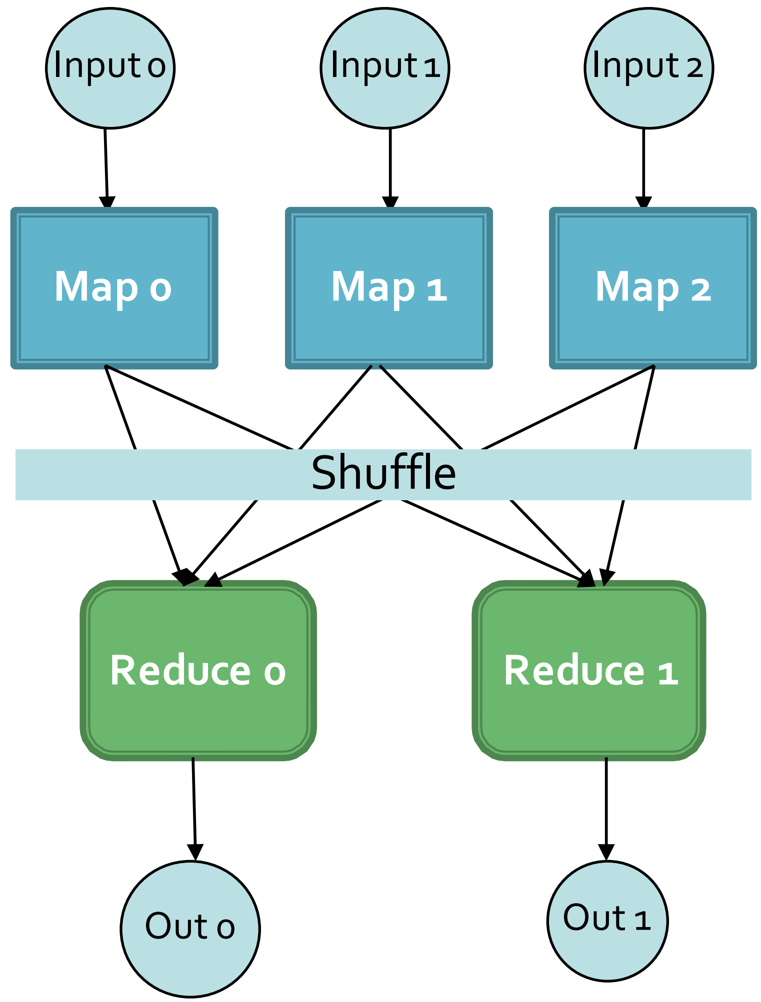
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k’v’-pairs with a given k’ are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



Data Flow

- **Input and final output are stored on a distributed file system (FS):**
  - 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 MapReduce task**

Execution Overview

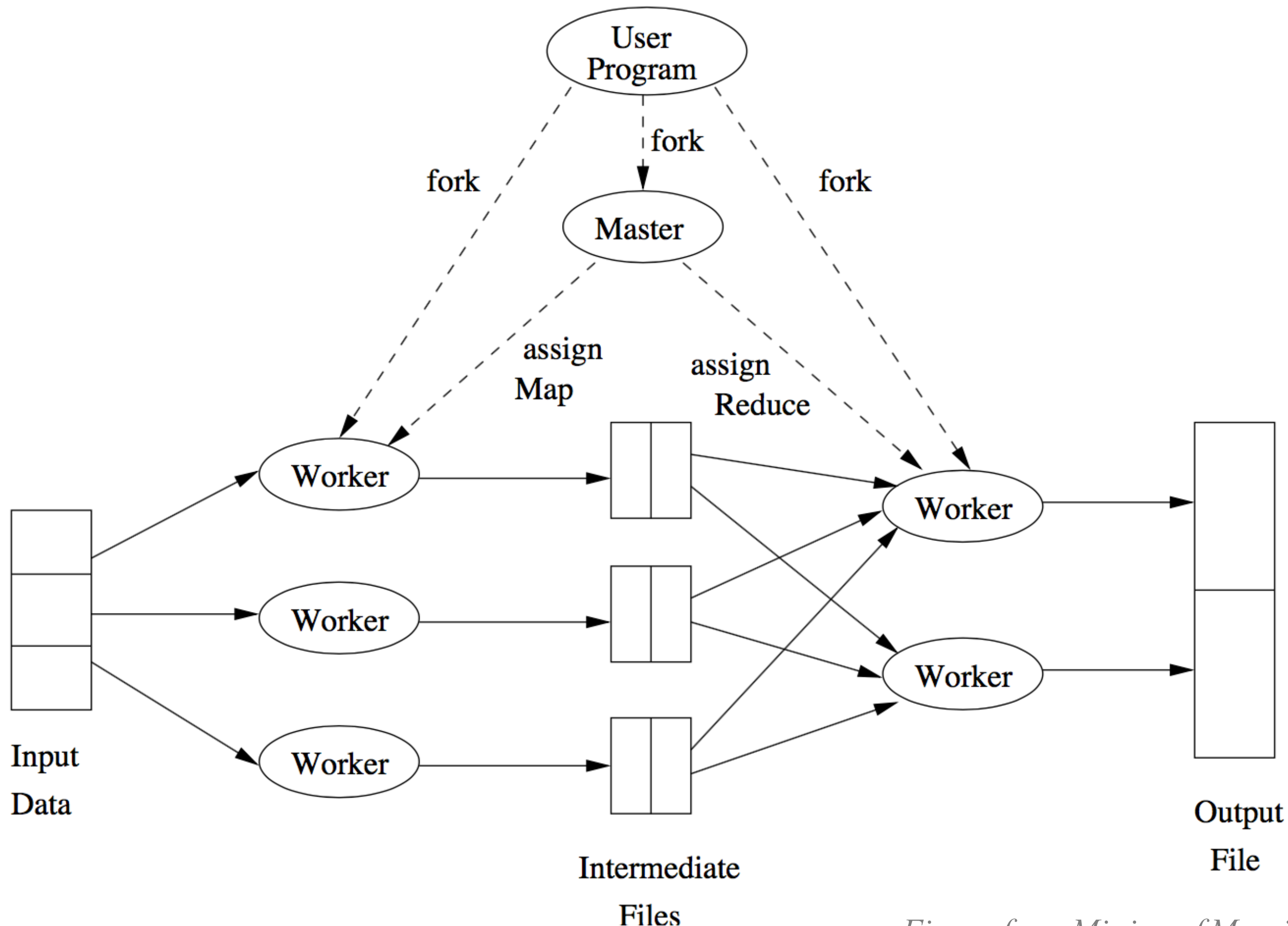


Figure from Mining of Massive Datasets textbook

Coordination: Master

- Master node takes care of coordination:
  - **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

Dealing with 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
  - Reduce task is restarted
- **Master failure**
  - MapReduce task is aborted and client is notified

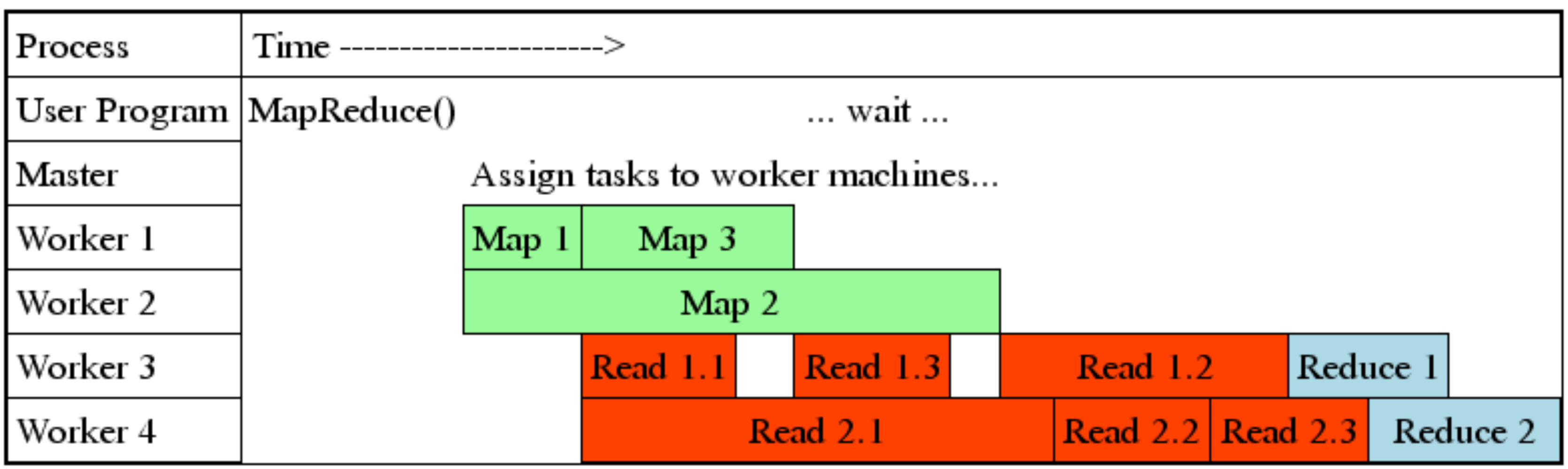
How many Map and Reduce jobs?

- *User defines M* map tasks, *R* reduce tasks
- **Rule of a thumb:**
  - Make *M* much larger than the number of nodes in the cluster
    - One DFS chunk per map is common
    - Improves dynamic load balancing and speeds up recovery from worker failures
  - **Usually *R* is smaller than *M***
    - Each mapper generates a file per reducer
    - Output is spread across *R* files

can use –getmerge to merge all files at the end

Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks >> machines
  - Minimizes time for fault recovery
  - Better dynamic load balancing

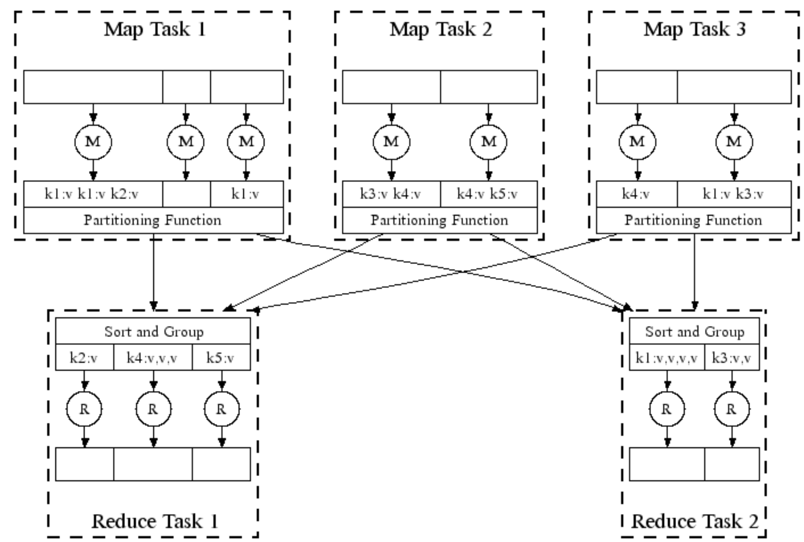


Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

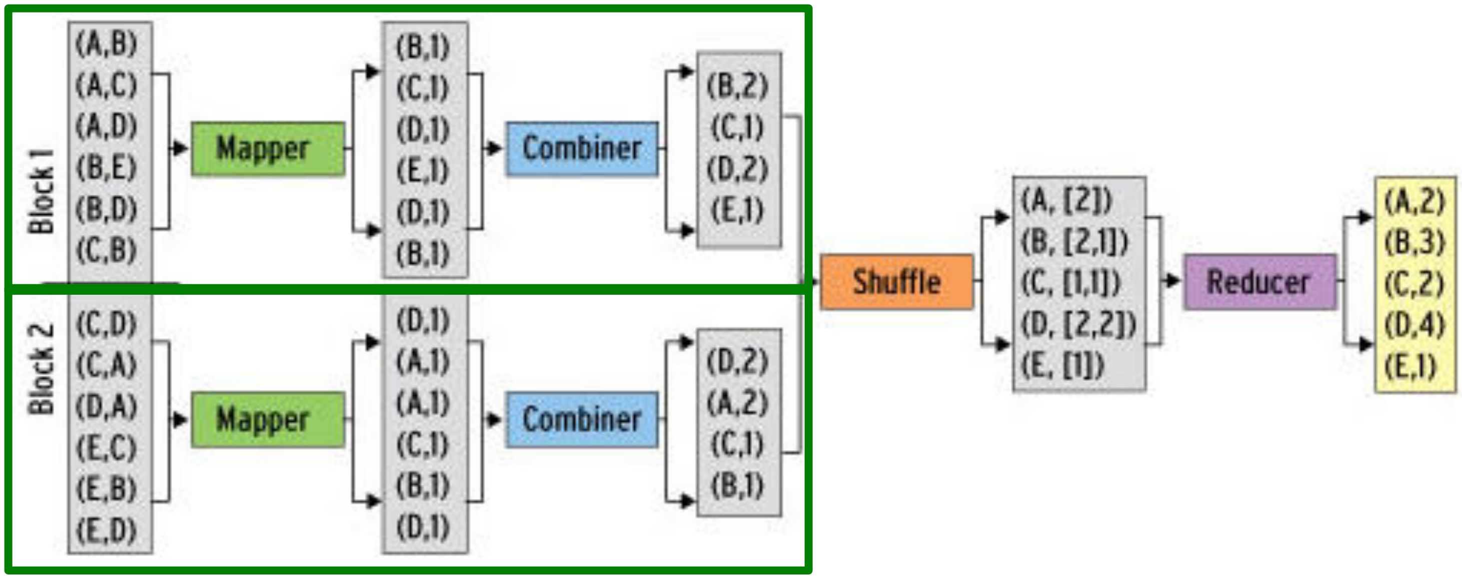
Refinement: Combiners

- Often a Map task will produce many pairs of the form *(k,v_1), (k,v_2), …* for the same key *k*
  - E.g., popular words in the word count example
- **Can save network time by pre-aggregating values in the mapper:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative



Refinement: Combiners

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - **hash(key) mod *R***
- **Sometimes useful to override the hash function:**
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

Problems Suited for Map-Reduce

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host

- **Map:**
  - Emit <host name, size>
- **Reduce:**
  - Sum up the sizes

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

Example Application: Join

Join Operation

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	B	C
a_3	b_2	c_1
a_3	b_2	c_2
a_4	b_3	c_3

Join with MapReduce: Map

- Map:
  - For each input tuple **R(a, b)**:
    Generate <**key** = **b**, **value** = **('R', a)**>
  - For each input tuple **S(b, c)**:
    Generate <**key** = **b**, **value** = **('S', c)**>

Think of 'R' and 'S' as bool variables that indicate where the pair originated from

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

Key-value pairs

<b_1, (R, a_1)>
<b_1, (R, a_2)>
<b_2, (R, a_3)>
<b_3, (R, a_4)>
<b_2, (S, c_1)>
<b_2, (S, c_2)>
<b_3, (S, c_3)>

Join with MapReduce: Shuffle & Sort

Output of Map

$<b_1, (R, a_1)>$

$<b_1, (R, a_2)>$

$<b_2, (R, a_3)>$

$<b_3, (R, a_4)>$

$<b_2, (S, c_1)>$

$<b_2, (S, c_2)>$

$<b_3, (S, c_3)>$

Input of Reduce

$<b_1, [(R, a_1); (R, a_2)]>$

$<b_2, [(R, a_3); (S, c_1); (S, c_2)]>$

$<b_3, [(R, a_4); (S, c_3)]>$

Join with MapReduce: Reduce

- Reduce:
  - Input: <b, value_list>
  - In the value list:
    - Pair each entry of the form (‘R’, a) with each entry (‘S’, c), and output:

      <a, b, c>

Input of Reduce

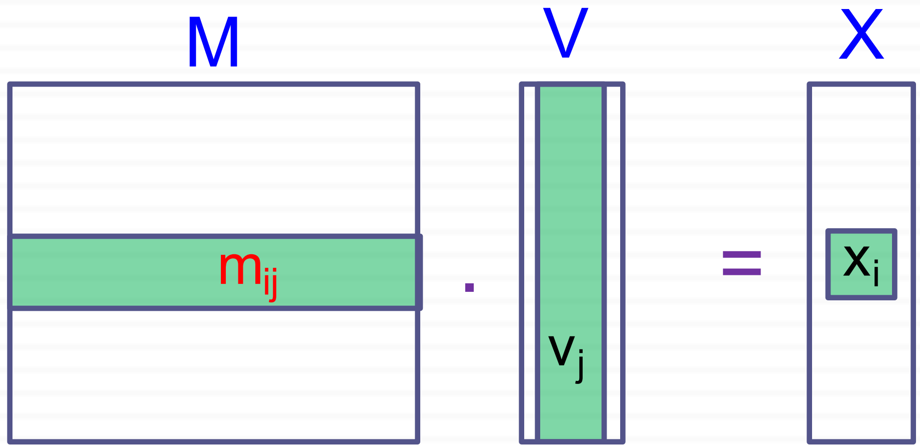
$<b_1, [(R, a_1); (R, a_2)]>$
$<b_2, [(R, a_3); (S, c_1); (S, c_2)]>$
$<b_3, [(R, a_4); (S, c_3)]>$

Output of Reduce

$<a_3, b_2, c_1>$
$<a_3, b_2, c_2>$
$<a_4, b_3, c_3>$

Example Application: Matrix-Vector Multiplication

Matrix-Vector Multiplication



$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

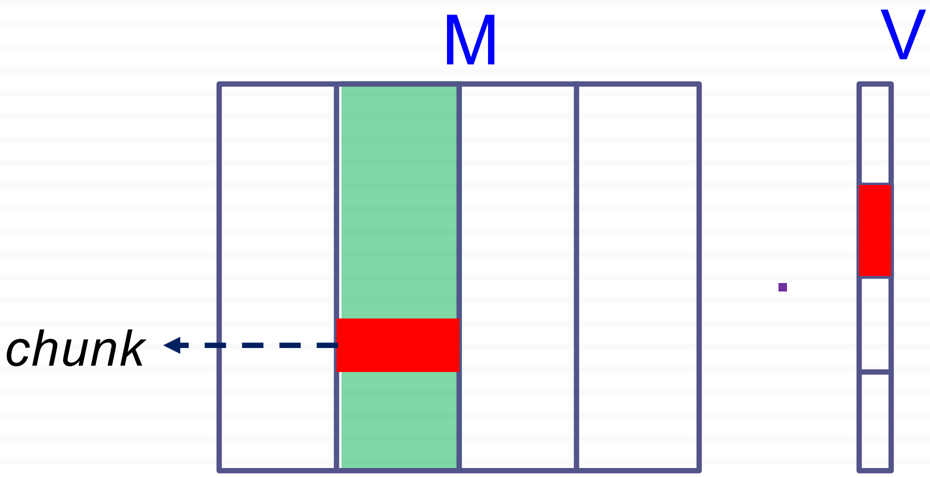
Simple Case: Vector fits in memory

- For simplicity, assume that n is not too large and V fits into main memory of each node.
- First, read V into an array accessible from Map tasks
- **Map**:
  - For each m_{ij}, generate <**key** = **i**, **value** = $\mathbf{m_{ij} * v_j}$>
- **Reduce**:
  - Input: <**key** = **i**, **values** = [$\mathbf{m_{i1} * v_1}$; $\mathbf{m_{i2} * v_2}$; … ; $\mathbf{m_{in} * v_n}$]
  - Sum up all values, and output <**key** = **i**, **value** = **sum**>

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

- What if V does not fit into main memory?
  - Still works, but very slow.

General Case: Vector does not fit into memory

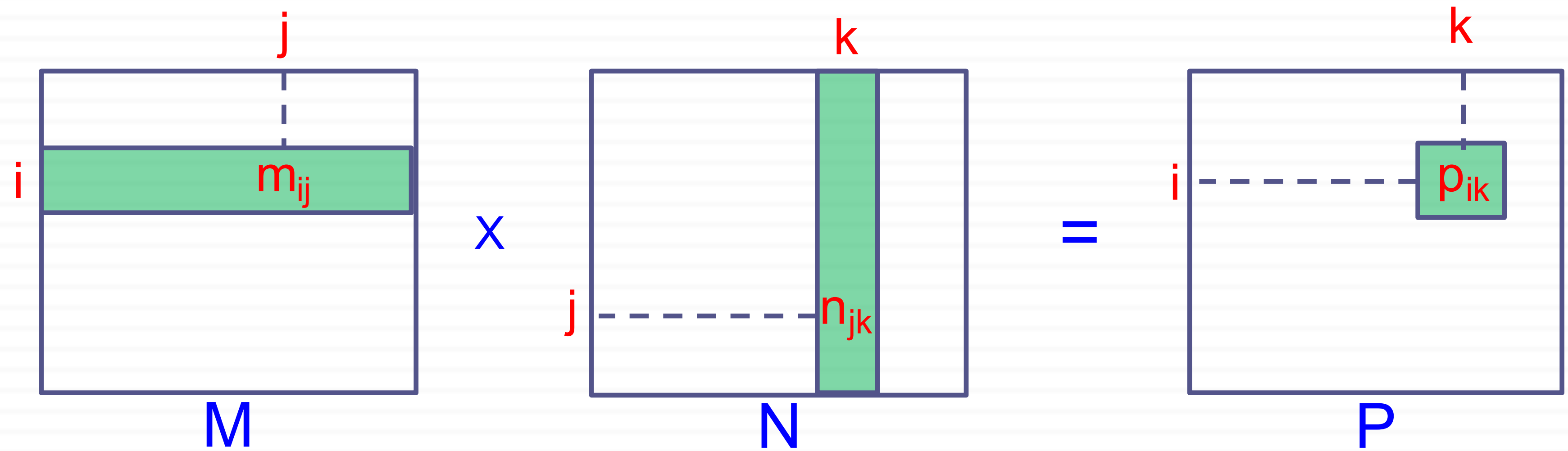


- *Vertical stripes for M*
- *Horizontal stripes for V*
- *Each stripe stored in a file.*

- Each map task:
  - is assigned a chunk of one of the M files.
  - reads one stripe of V completely, and stores in local node's memory.
- Map and reduce function definitions same as in previous slide.

Example Application: Matrix-Matrix Multiplication 2 Map-Reduce Steps

Matrix – Matrix Multiplication



$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Two-Step MapReduce

- Two matrices in two separate input files

- Two steps of MapReduce:
  - Step 1: “Join” the matrix entries that need to be multiplied with each other
    - *Similar to the Join operation we implemented using MapReduce*

  - Step 2: Accumulate all results and compute the output matrix entries

First MapReduce Step

- Objective: "Join" $\mathbf{m_{ij}}$ and $\mathbf{n_{jk}}$ entries
  - In this case, "join" corresponds to multiplication

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- **Map:**
  - For each $\mathbf{m_{ij}}$ value of matrix **M**

    Generate <**key** = **j**, **value** = **("M", i, $\mathbf{m_{ij}}$)** >

  - For each $\mathbf{n_{jk}}$ value of matrix **N**

    Generate <**key** = **j**, **value** = **("N", k, $\mathbf{m_{jk}}$)** >

Example: Map Output

$$
\begin{array}{cc}
 & j \\
i & \begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & k \\
j & \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
\end{array}
$$

Map output:

$<1, (M, 2, m_{21})>$ $<1, (N, 2, n_{12})>$

$<1, (M, 4, m_{41})>$ $<1, (N, 4, n_{14})>$

$<3, (M, 1, m_{13})>$ $<3, (N, 2, n_{32})>$

$<3, (M, 4, m_{43})>$ $<3, (N, 4, n_{34})>$

$<4, (M, 2, m_{24})>$ $<4, (N, 2, n_{42})>$

Intuition 1: Joining entries with same j values

Matrix M (rows i, columns j):

0	0	m_{13}	0
m_{21}	0	0	m_{24}
0	0	0	0
m_{41}	0	m_{43}	0

Matrix N (rows j, columns k):

0	n_{12}	0	n_{14}
0	0	0	0
0	n_{32}	0	n_{34}
0	n_{42}	0	0

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Map output:

<1, (M, 2, m_{21})>	<1, (N, 2, n_{12})>
<1, (M, 4, m_{41})>	<1, (N, 4, n_{14})>
<3, (M, 1, m_{13})>	<3, (N, 2, n_{32})>
<3, (M, 4, m_{43})>	<3, (N, 4, n_{34})>
<4, (M, 2, m_{24})>	<4, (N, 2, n_{42})>

Intuition 1: Joining entries with same j values

$$
\begin{array}{c|cccc}
 & & & j & \\
\hline
 & 0 & 0 & m_{13} & 0 \\
 & m_{21} & 0 & 0 & m_{24} \\
i & 0 & 0 & 0 & 0 \\
 & m_{41} & 0 & m_{43} & 0
\end{array}
\qquad
\begin{array}{c|cccc}
 & & k & & \\
\hline
 & 0 & n_{12} & 0 & n_{14} \\
 & 0 & 0 & 0 & 0 \\
j & 0 & n_{32} & 0 & n_{34} \\
 & 0 & n_{42} & 0 & 0
\end{array}
$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Map output:

<1, (M, 2, m_{21})>	<1, (N, 2, n_{12})>
<1, (M, 4, m_{41})>	<1, (N, 4, n_{14})>
<3, (M, 1, m_{13})>	<3, (N, 2, n_{32})>
<3, (M, 4, m_{43})>	<3, (N, 4, n_{34})>
<4, (M, 2, m_{24})>	<4, (N, 2, n_{42})>

Intuition 2: Partial sums

$$\begin{array}{ccccc} & & & j & \\ & 0 & 0 & m_{13} & 0 \\ i & m_{21} & 0 & 0 & m_{24} \\ & 0 & 0 & 0 & 0 \\ & m_{41} & 0 & m_{43} & 0 \end{array}
\qquad
\begin{array}{ccccc} & & & k & \\ & 0 & n_{12} & 0 & n_{14} \\ j & 0 & 0 & 0 & 0 \\ & 0 & n_{32} & 0 & n_{34} \\ & 0 & n_{42} & 0 & 0 \end{array}$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

<1, (M, 2, m_{21})>	<1, (N, 2, n_{12})>
<1, (M, 4, m_{41})>	<1, (N, 4, n_{14})>
<3, (M, 1, m_{13})>	<3, (N, 2, n_{32})>
<3, (M, 4, m_{43})>	<3, (N, 4, n_{34})>
<4, (M, 2, m_{24})>	<4, (N, 2, n_{42})>

m_{21} . n_{12} will contribute to the partial sum of p_{22}

Intuition 2: Partial sums

j

	0	0	m_{13}	0
i	m_{21}	0	0	m_{24}
	0	0	0	0
	m_{41}	0	m_{43}	0

k

	0	n_{12}	0	n_{14}
j	0	0	0	0
	0	n_{32}	0	n_{34}
	0	n_{42}	0	0

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

<1, (M, 2, m_{21})>
<1, (M, 4, m_{41})>
<3, (M, 1, m_{13})>
<3, (M, 4, m_{43})>
<4, (M, 2, m_{24})>

<1, (N, 2, n_{12})>
<1, (N, 4, n_{14})>
<3, (N, 2, n_{32})>
<3, (N, 4, n_{34})>
<4, (N, 2, n_{42})>

m_{24} . n_{42} will contribute to the partial sum of p_{22}

Intuition 2: Partial sums

$$\begin{array}{cc} & j \\ i & \begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \end{array} \qquad \begin{array}{cc} & k \\ j & \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} \end{array}$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

<1, (M, 2, m_{21})>	<1, (N, 2, n_{12})>
<1, (M, 4, m_{41})>	<1, (N, 4, n_{14})>
<3, (M, 1, m_{13})>	<3, (N, 2, n_{32})>
<3, (M, 4, m_{43})>	<3, (N, 4, n_{34})>
<4, (M, 2, m_{24})>	<4, (N, 2, n_{42})>

$m_{21} \cdot n_{14}$ will contribute to the partial sum of p_{24}

First MapReduce Step: Reduce

Reduce input: <1, [(M, 2, m_{21}); (M, 4, m_{41}); (N, 2, n_{12}); (N, 4, n_{14})] >
<3, [(M, 1, m_{13}); (M, 4, m_{43}); (N, 2, n_{32}); (N, 4, n_{34})] >
<4, [(M, 2, m_{24}); (N, 2, n_{42})] >

Reduce(key, value_list):

Put all entries in value_list of form **(M, i, $m_{i,key}$)** into **L_M**

Put all entries in value_list of form **(N, k, $n_{key,k}$)** into **L_N**

for each entry **(M, i, $m_{i,key}$)** in **L_M**

  for each entry **(N, k, $n_{key,k}$)** in **L_N**

    output <**key** = **(i, k)**; **value** = $m_{i,key} \cdot n_{key,k}$ >

Example: Reduce Output

Reduce input:

$<1, [\ (M, 2, m_{21}); (M, 4, m_{41}); (N, 2, n_{12}); (N, 4, n_{14})\ ] >$

$<3, [\ (M, 1, m_{13}); (M, 4, m_{43}); (N, 2, n_{32}); (N, 4, n_{34})\ ] >$

$<4, [\ (M, 2, m_{24}); (N, 2, n_{42})\ ] >$

Reduce output:

$<$ **(2, 2)**, $(m_{21}.n_{12})$ $>$

$<$ **(2, 4)**, $(m_{21}.n_{14})$ $>$

$<$ **(4, 2)**, $(m_{41}.n_{12})$ $>$

$<$ **(4, 4),** $(m_{41}.n_{14})$ $>$

$<$ **(1, 2)**, $(m_{13}.n_{32})$ $>$

$<$ **(1, 4)**, $(m_{13}.n_{34})$ $>$

$<$ **(4, 2)**, $(m_{43}.n_{32})$ $>$

$<$ **(4, 4),** $(m_{43}.n_{34})$ $>$

$<$ **(2, 2)**, $(m_{24}.n_{42})$ $>$

Second MapReduce Step: Map

Map input:

< (2, 2), ($m_{21}.n_{12}$) >
< (2, 4), ($m_{21}.n_{14}$) >
< (4, 2), ($m_{41}.n_{12}$) >
< (4, 4), ($m_{41}.n_{14}$) >

< (1, 2), ($m_{13}.n_{32}$) >
< (1, 4), ($m_{13}.n_{34}$) >
< (4, 2), ($m_{43}.n_{32}$) >
< (4, 4), ($m_{43}.n_{34}$) >

< (2, 2), ($m_{24}.n_{42}$) >

- **Map:**

  for each **(key, value)** pair in the input

  generate **(key, value)**
- Identity function
- The system will most likely assign the map tasks on the same node as the reduce that produced these outputs. Hence, no communication cost.

Second MapReduce Step: Reduce

Reduce input:

$< (2, 2), (m_{21}.n_{12}) >$ $< (1, 2), (m_{13}.n_{32}) >$
$< (2, 4), (m_{21}.n_{14}) >$ $< (1, 4), (m_{13}.n_{34}) >$
$< (4, 2), (m_{41}.n_{12}) >$ $< (4, 2), (m_{43}.n_{32}) >$
$< (4, 4), (m_{41}.n_{14}) >$ $< (4, 4), (m_{43}.n_{34}) >$

$< (2, 2), (m_{24}.n_{42}) >$

- **Reduce(key, value_list):**

```
sum = 0
foreach v in value_list
    sum += v
output <key, sum>
```

Example: MapReduce Step 2 - Reduce

$$\begin{array}{cc} & j \\ i & \begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \end{array} \qquad \begin{array}{cc} & k \\ j & \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} \end{array}$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Reduce input:

$< (2, 2), (m_{21}.n_{12}) >$	$< (1, 2), (m_{13}.n_{32}) >$
$< (2, 4), (m_{21}.n_{14}) >$	$< (1, 4), (m_{13}.n_{34}) >$
$< (4, 2), (m_{41}.n_{12}) >$	$< (4, 2), (m_{43}.n_{32}) >$
$< (4, 4), (m_{41}.n_{14}) >$	$< (4, 4), (m_{43}.n_{34}) >$

$< (2, 2), (m_{24}.n_{42}) >$

Example: MapReduce Step 2 - Reduce

$$
\begin{array}{cc}
 & j \\
i & \begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & k \\
j & \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
\end{array}
$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Reduce input:

$< (2, 2), (m_{21}.n_{12}) >$ $< (1, 2), (m_{13}.n_{32}) >$

$< (2, 4), (m_{21}.n_{14}) >$ $< (1, 4), (m_{13}.n_{34}) >$

$< (4, 2), (m_{41}.n_{12}) >$ $< (4, 2), (m_{43}.n_{32}) >$

$< (4, 4), (m_{41}.n_{14}) >$ $< (4, 4), (m_{43}.n_{34}) >$

$< (2, 2), (m_{24}.n_{42}) >$

Example: MapReduce Step 2 - Reduce

$$
\begin{array}{c|cccc|}
 & & & j & \\
 & 0 & 0 & m_{13} & 0 \\
i & m_{21} & 0 & 0 & m_{24} \\
 & 0 & 0 & 0 & 0 \\
 & m_{41} & 0 & m_{43} & 0
\end{array}
\qquad
\begin{array}{c|cccc|}
 & & & k & \\
 & 0 & n_{12} & 0 & n_{14} \\
 & 0 & 0 & 0 & 0 \\
j & 0 & n_{32} & 0 & n_{34} \\
 & 0 & n_{42} & 0 & 0
\end{array}
$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Reduce input:

< (2, 2), ($m_{21}.n_{12}$) >
< (2, 4), ($m_{21}.n_{14}$) >
< (4, 2), ($m_{41}.n_{12}$) >
< (4, 4), ($m_{41}.n_{14}$) >

< (1, 2), ($m_{13}.n_{32}$) >
< (1, 4), ($m_{13}.n_{34}$) >
< (4, 2), ($m_{43}.n_{32}$) >
< (4, 4), ($m_{43}.n_{34}$) >

< (2, 2), ($m_{24}.n_{42}$) >

Example: MapReduce Step 2 - Reduce

$$
\begin{array}{c c}
 & j \\
i & \begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\end{array}
\qquad
\begin{array}{c c}
 & k \\
j & \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
\end{array}
$$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Reduce input:

< **(2, 2)**, ($m_{21}.n_{12}$) >
< **(2, 4)**, ($m_{21}.n_{14}$) >
< **(4, 2)**, ($m_{41}.n_{12}$) >
< **(4, 4)**, ($m_{41}.n_{14}$) >

< **(1, 2)**, ($m_{13}.n_{32}$) >
< **(1, 4)**, ($m_{13}.n_{34}$) >
< **(4, 2)**, ($m_{43}.n_{32}$) >
< **(4, 4)**, ($m_{43}.n_{34}$) >

< **(2, 2)**, ($m_{24}.n_{42}$) >

Summary: Two-Step MapReduce Algorithm

- **Step1: Map (input):**

  For each m_{ij} value of matrix **M**

  generate <**key** = **j**, **value** = **("M", i, m_{ij})** >

  For each n_{jk} value of matrix **N**

  generate <**key** = **j**, **value** = **("N", k, m_{jk})** >

- **Step1: Reduce(key, value_list):**

  for each entry **(M, i, $m_{i,key}$)** in value_list

  for each entry **(N, k, $n_{key,k}$)** in value_list

  output <**key** = **(i, k)**; **value** = $m_{i,key} \cdot n_{key,k}$>

- **Step2: Map (key, value):**

  generate (**key**, **value**)

- **Step2: Reduce(key, value_list):**

  sum ← accumulate the values in value_list

  output (**key**, **sum**)

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example Application: Matrix-Matrix Multiplication Single Map-Reduce

Single-Step MapReduce: Intuition

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- To compute $\mathbf{p_{ik}}$, we need $\mathbf{m_{ij}}$ and $\mathbf{n_{jk}}$ values for all $\mathbf{j}$.

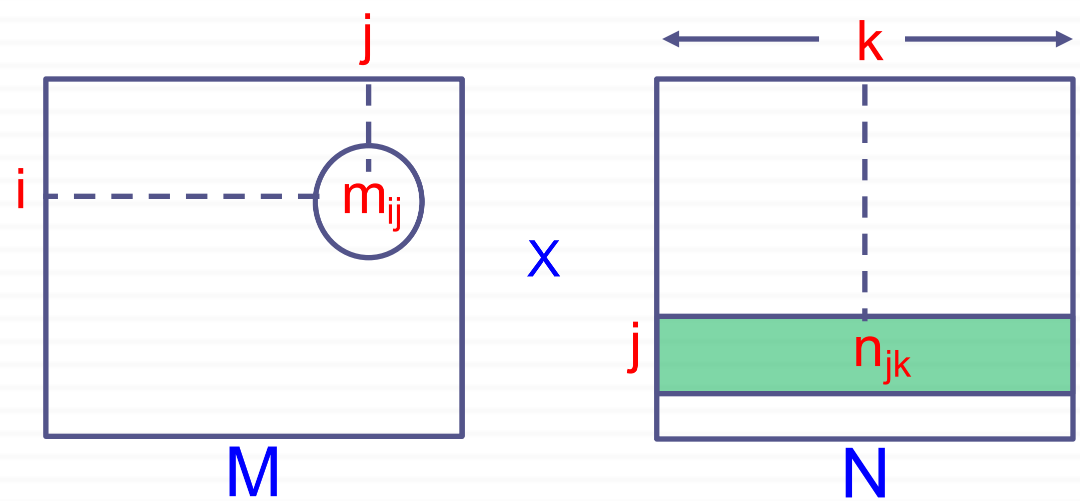
- In other words:

  $\mathbf{m_{ij}}$ entry is needed to compute $\mathbf{p_{ik}}$ values for all $\mathbf{k}$.

  $\mathbf{n_{jk}}$ entry is needed to compute $\mathbf{p_{ik}}$ values for all $\mathbf{i}$.

- Intuition: Send each input matrix entry to all reducers that need it.

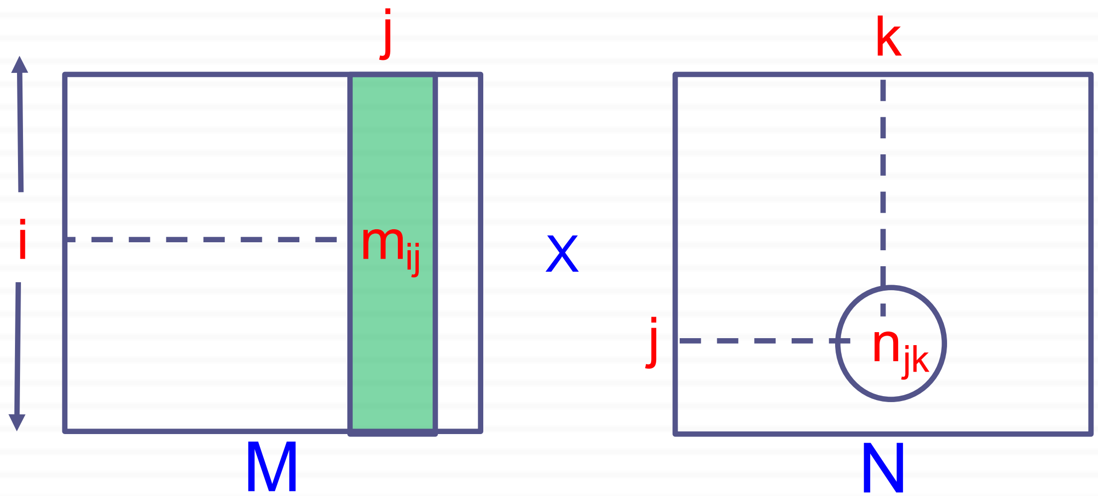
An Entry of Matrix M



$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- Each $\mathbf{m_{ij}}$ needs to be paired with all entries in row j of matrix N

An Entry of Matrix N



$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- Each $\mathbf{n_{jk}}$ needs to be paired with all entries in column j of matrix M

Map Operation

- *Reminder:*

  m_{ij} *entry is needed to compute* p_{ik} *values for all* k.

  n_{jk} *entry is needed to compute* p_{ik} *values for all* i.

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- Map:

  for each $\mathbf{m_{ij}}$ entry from matrix **M**:

      for **k**=1 to n

          generate <**key** = **(i, k)**, **value** = (**'M', j,** $\mathbf{m_{ij}}$) >

  for each $\mathbf{n_{jk}}$ entry from matrix **N**:

      for **i**=1 to n

          generate <**key** = **(i, k)**, **value** = (**'N', j,** $\mathbf{n_{jk}}$) >

Example: Map Output for Matrix M Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}
$$

Map output:

$<(1,1), (M, 3, m_{13})>$

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix M Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}
$$

Map output:

<(1,1), (M, 3, m_{13}) >
<(1,2), (M, 3, m_{13}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix M Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}
$$

Map output:

<(1,1), (M, 3, m_{13}) >
<(1,2), (M, 3, m_{13}) >
<(1,3), (M, 3, m_{13}) >
<(1,4), (M, 3, m_{13}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix M Entries

$$\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

Map output:

<(1,1), (M, 3, m_{13}) >   <(2,1), (M, 1, m_{21}) >
<(1,2), (M, 3, m_{13}) >   <(2,2), (M, 1, m_{21}) >
<(1,3), (M, 3, m_{13}) >   <(2,3), (M, 1, m_{21}) >
<(1,4), (M, 3, m_{13}) >   <(2,4), (M, 1, m_{21}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix M Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \end{bmatrix}
$$

Map output:

<(1,1), (M, 3, m_{13}) >	<(2,1), (M, 1, m_{21}) >	<(2,1), (M, 4, m_{24}) >	
<(1,2), (M, 3, m_{13}) >	<(2,2), (M, 1, m_{21}) >	<(2,2), (M, 4, m_{24}) >	…
<(1,3), (M, 3, m_{13}) >	<(2,3), (M, 1, m_{21}) >	<(2,3), (M, 4, m_{24}) >	
<(1,4), (M, 3, m_{13}) >	<(2,4), (M, 1, m_{21}) >	<(2,4), (M, 4, m_{24}) >	

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix N Entries

$$\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

Map output:

<(1,2), (N, 1, n_{12}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix N Entries

$$\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Map output:

<(1,2), (N, 1, n_{12}) >
<(2,2), (N, 1, n_{12}) >
<(3,2), (N, 1, n_{12}) >
<(4,2), (N, 1, n_{12}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix N Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \end{bmatrix}
$$

Map output:

<(1,2), (N, 1, n_{12}) > <(1,4), (N, 1, n_{14}) >
<(2,2), (N, 1, n_{12}) > <(2,4), (N, 1, n_{14}) >
<(3,2), (N, 1, n_{12}) > <(3,4), (N, 1, n_{14}) >
<(4,2), (N, 1, n_{12}) > <(4,4), (N, 1, n_{14}) >

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Map Output for Matrix N Entries

$$
\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix}
\cdot
\begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix}
=
\begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}
$$

Map output:

<(1,2), (N, 1, n_{12}) >
<(2,2), (N, 1, n_{12}) >
<(3,2), (N, 1, n_{12}) >
<(4,2), (N, 1, n_{12}) >

<(1,4), (N, 1, n_{14}) >
<(2,4), (N, 1, n_{14}) >
<(3,4), (N, 1, n_{14}) >
<(4,4), (N, 1, n_{14}) >

<(1,2), (N, 3, n_{32}) >
<(2,2), (N, 3, n_{32}) >
<(3,2), (N, 3, n_{32}) >
<(4,2), (N, 3, n_{32}) >

…

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Summary: Map Operation

m_{ij}

<**key**=(i,1), **value** = (M, j, m_{ij})>

⋮

<**key**=(i,n), **value** = (M, j, m_{ij})>

n_{jk}

<**key**=(1,k), **value** = (N, j, n_{jk})>

⋮

<**key**=(n,k), **value** = (N, j, n_{jk})>

Reduce Operation

- Input:

  key = **(i,k)**, **value_list** = [… **(M, j, m_{ij})**; … **(N, j, n_{jk})** …]

  an entry exists for any non-zero m_{ij} or n_{jk}

- Objective: Multiply m_{ij} and n_{jk} values with matching j values, and sum up all products to compute p_{ik}

- **Reduce(key, value_list)**

  put all entries of form (M, j, m_{ij}) into L_M
  sort entries in L_M based on j values
  put all entries of form (N, j, n_{jk}) into L_N
  sort entries in L_N based on j values

  sum ← 0
  for each pair (M, j, m_{ij}) in L_M and (N, j, n_{jk}) in L_N
  sum += m_{ij} . n_{jk}
  output (key, sum)

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Example: Reduce

$$\begin{bmatrix} 0 & 0 & m_{13} & 0 \\ m_{21} & 0 & 0 & m_{24} \\ 0 & 0 & 0 & 0 \\ m_{41} & 0 & m_{43} & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & n_{12} & 0 & n_{14} \\ 0 & 0 & 0 & 0 \\ 0 & n_{32} & 0 & n_{34} \\ 0 & n_{42} & 0 & 0 \end{bmatrix} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

Reduce input: **key** = **(4, 2)**, **value_list** = { **(M, m_{41}, 1)**; **(M, m_{43}, 3)**; **(N, n_{12}, 1)**; **(N, n_{32}, 3)**; **(N, n_{42}, 4)** }

Reduce output: **key** = **(4, 2)**, **value =** $m_{41} . n_{12} + m_{43} . n_{32}$

Summary: Single-Step MapReduce Algorithm

- **Map(input):**

  for each m_{ij} entry from matrix **M**:

  for **k**=1 to n

  generate <**key** = **(i, k)**, **value** = (**'M', j, m_{ij}**) >

  for each n_{jk} entry from matrix **N**:

  for **i**=1 to n

  generate <**key** = **(i, k)**, **value** = (**'N', j, n_{jk}**) >

- **Reduce(key, value_list)**

  sum ← 0

  for each pair **(M, j, m_{ij})** and **(N, j, n_{jk})** in value_list

  sum += m_{ij} . n_{jk}

  output **(key, sum)**

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

Next Lecture

- No analysis of communication costs and computation costs so far.

- Next lecture:
  - Complexity Analysis
  - Improved Algorithms