USC Viterbi
School of Engineering

INF 553: Foundations and Applications of Data Mining

Map-Reduce

Dr. Anna Farzindar
farzinda@usc.edu

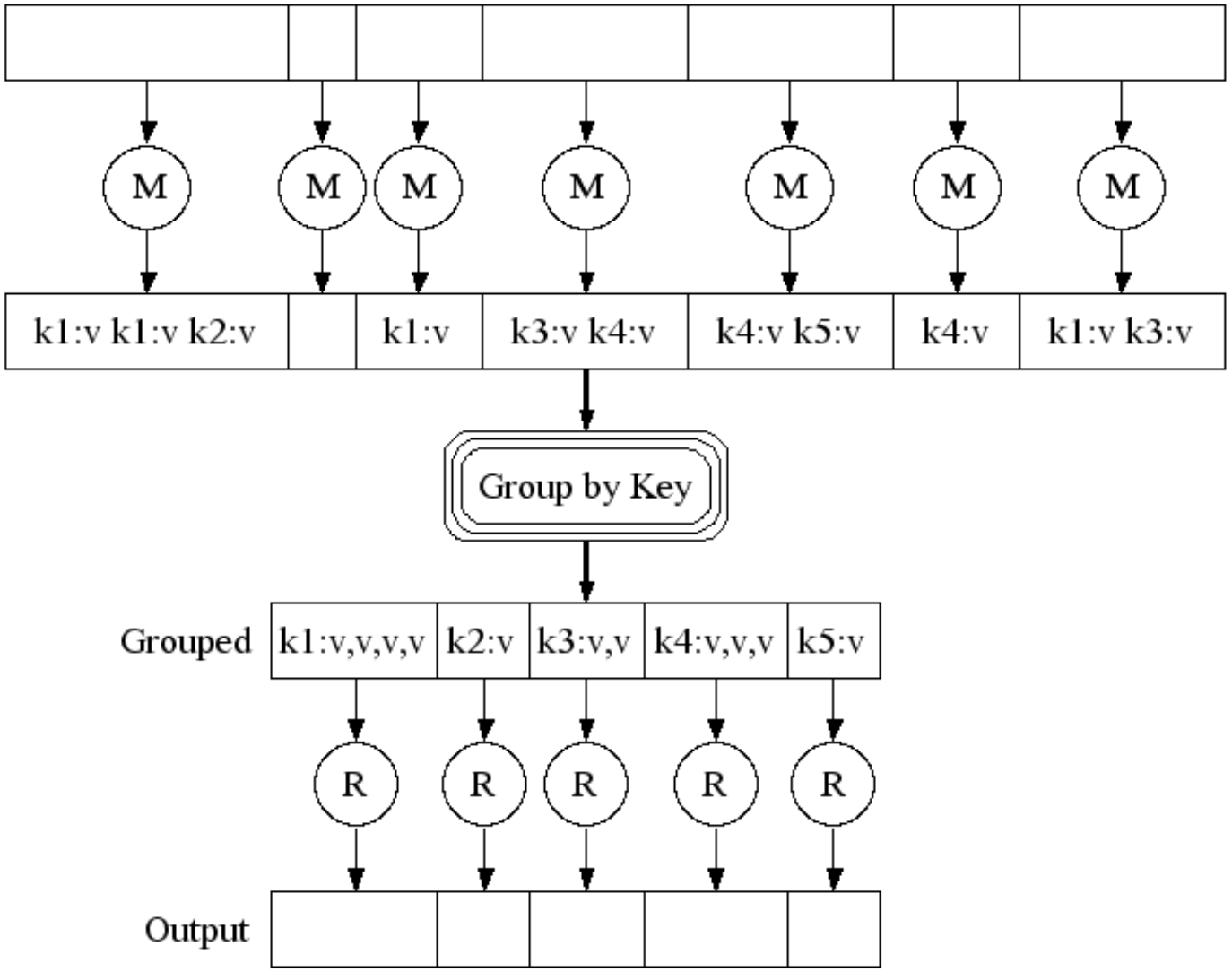
Thanks for source slides and material to: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets

Map-Reduce: A diagram

MAP:
Read input and produces 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



Example (Exercise 2.3.1 in book)

Design MapReduce algorithms to take a very large file of integers and produce as output:

- **(a) The largest integer**

Map task: ?

Reduce tasks?

Notes on example (a)
(a) The largest integer

- **Each Map task processes a chunk of the input file**
- **Map task produces (integer, 1) of the largest value in that chunk as key, value pair**
- **Grouping by key identifies duplicates**
- **How many Reduce tasks?**
- **Single reduce task: produces (integer, 1) of largest value**
- What about **multiple reduce tasks?**
  - In the shuffle step, keys could be **sorted by range**, so only look at **output** from the Reduce stage that has the **highest range of keys**.

Exercise 2.3.1 (cont.)

Design MapReduce algorithms to take a very large file of integers and produce as output:

- **(b) The average of all the integers**

Map task:

Reduce task:

Notes on example (b)

(b) The average of all the integers

- In this case, don't worry whether the integers are unique
- **Correction: Map task: produce (key, (num of integers, sum of integers)) as output**
  - **Value is the (num, sum) tuple**
  - **Key could be num or sum of integers or something else**
- **Single Reduce task: sum all sums of integers, sum num of integers from Reduce tasks, calculate average; emit (average, 1)**
  - If multiple values for one key, means there happened to be overlap on the value of the key
  - **Extract values from tuples and add to sums**
- Since each Map task summarizes large chunk of data by a single (key, (num, sum)) key-value pair, this should scale to use a **single Reducer** even with thousands of Map tasks.

Exercise 2.3.1 (cont.)

Design MapReduce algorithms to take a very large file of integers and produce as output:

- **(c) The same set of integers, but with each integer appearing only once**

Map task:

Reduce task:

Notes on example (c)

(c) The same set of integers, but with each integer appearing only once

- Does not require sorting of final output
- **Map task: emit (integer, 1) once only for each unique integer (e.g., maintain an array or list to track whether you have seen/emitted that integer before)**
- **Shuffle step will group together all values for the same integer: (integer, 1, 1, 1, 1…)**
  - Same integer may appear in multiple chunks that go to different Map tasks
  - Each integer key will only go to one Reduce task
- **Reduce task: eliminate duplicates (ignore list of 1's) for each integer key and emit (integer)**
- Combine the output from multiple reduce tasks (not required to be in any order.

Exercise 2.3.1 (cont.)

Design MapReduce algorithms to take a very large file of integers and produce as output:

- **(d) The count of the number of distinct integers in the input**

Map task:

Reduce task:

Intuition for (d)

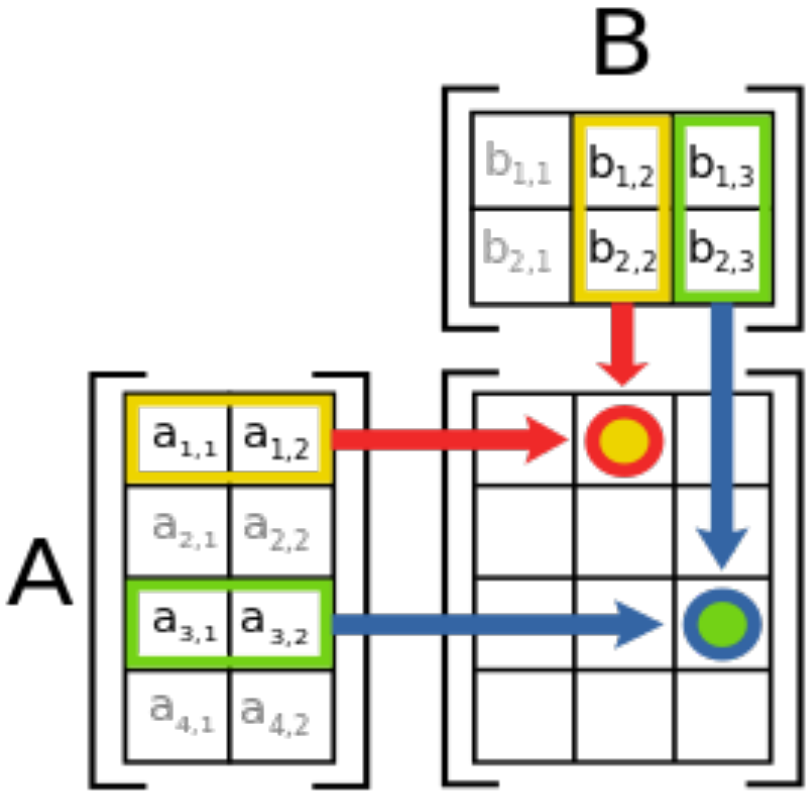
(d) The count of the number of distinct integers in the input

- **Need to eliminate duplicates and then count unique integers**
- Each **Map task** only knows about its chunk of data
  - It can **eliminate duplicates for that chunk (as in part (c))**
- **Shuffle step:** groups together any duplicates from multiple Map tasks before sending them to corresponding Reduce task
- **Reduce task** can eliminate duplicates for the keys (integers) that it processes (as in part (c))
- **But can Reduce task(s) also count the unique integers?**
  - **Can only count unique integers for the range of keys it processes**
- **Can we assume a single Reduce task in this case?**
  - No, because **input file is too large for a single machine,** and we don't know how many duplicates we will eliminate
- **If multiple Reduce tasks, then how do we count across?**

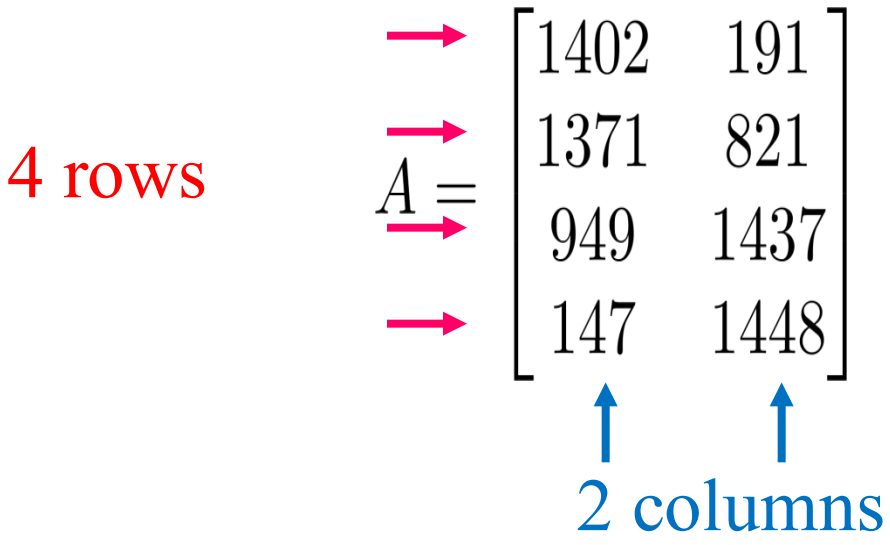
Notes on example (d)

- **Two stages needed,** since very large input file
- **First phase: eliminate duplicates in large input file**
  - **Map task 1:** just emit (integer, 1) for unique integers
  - **Sort by** key, so disjoint set or **range of integers** goes to each reducer
  - **Reduce task 1**: Eliminates duplicates, counts unique integers, emits count of unique integers for its range of inputs **(count, count)**
  - **Note that the Reduce tasks will process disjoint, non-overlapping sets of keys, so don't need to preserve the values**
- **Second phase: count unique numbers overall**
  - **Map task 2**: Each map task will get some number of counts as input from the previous Reduce phase
  - Adds them together and emits **(count, count)**
  - **Sort step:** only one Reducer task, so all (count, count) pairs sent there
  - **Reduce task 2**: single Reducer, **sums all counts from map tasks** and produces overall count (could be multiple values for a key).

Matrix Multiplication



Matrix: Rectangular array of numbers:



$\mathbb{R}^{4x2}$

Dimension of matrix

: number of rows x number of columns

Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

A11=
1402

A12=
191

A32=
1437

A41=
147

A43=
Undefined X

$A_{ij} =$ "i, j entry" in the i^{th} row, j^{th} column.

Example: Matrix-Vector Multiplication

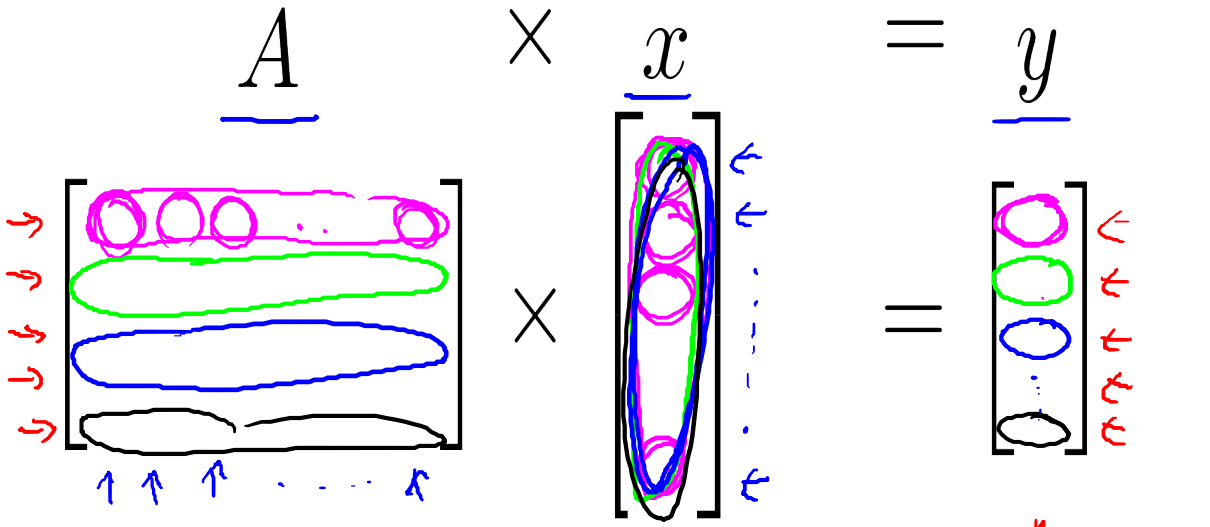
$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} =$$

1X1 + 3X5 = 16
4X1 + 0X5 = 4
2X1 + 1X5 = 7

3x2 2x1 3x1 matrix

Details:



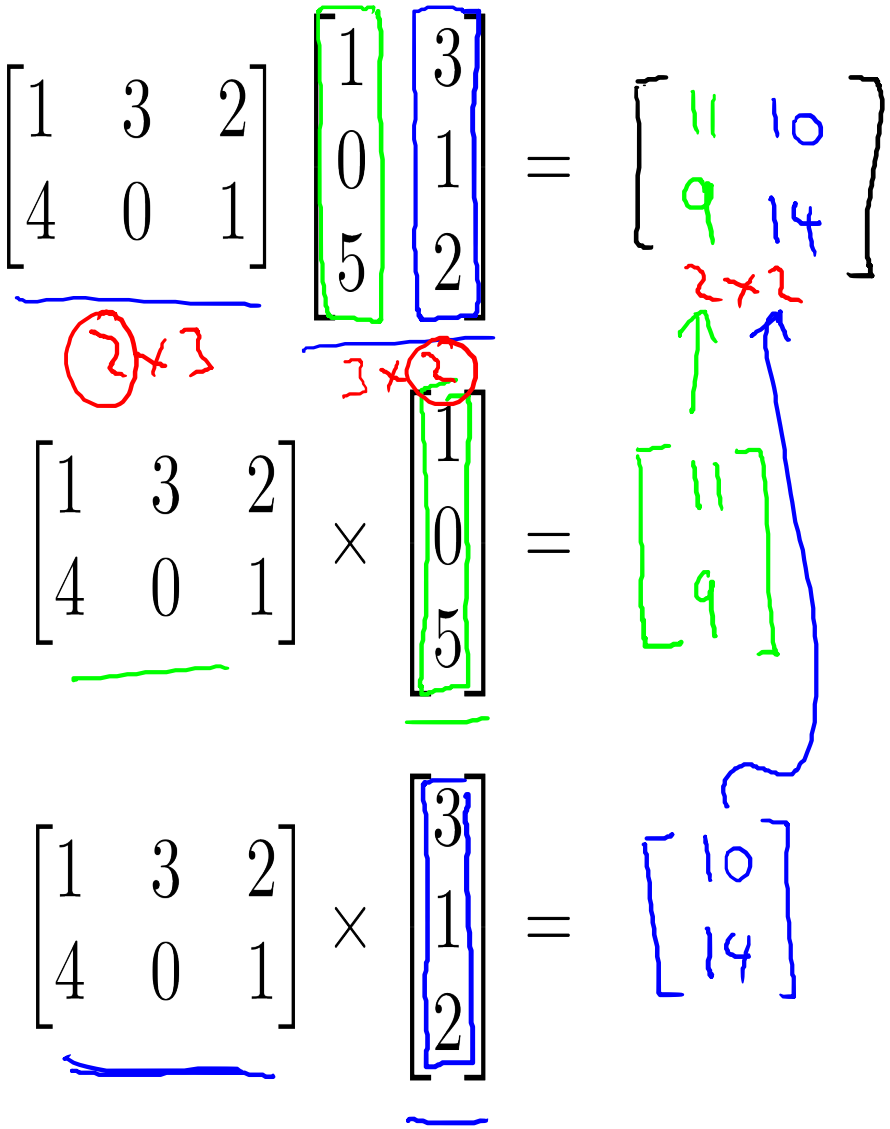
m x n matrix (m rows, n columns)

n x 1 matrix (n-dimensional vector)

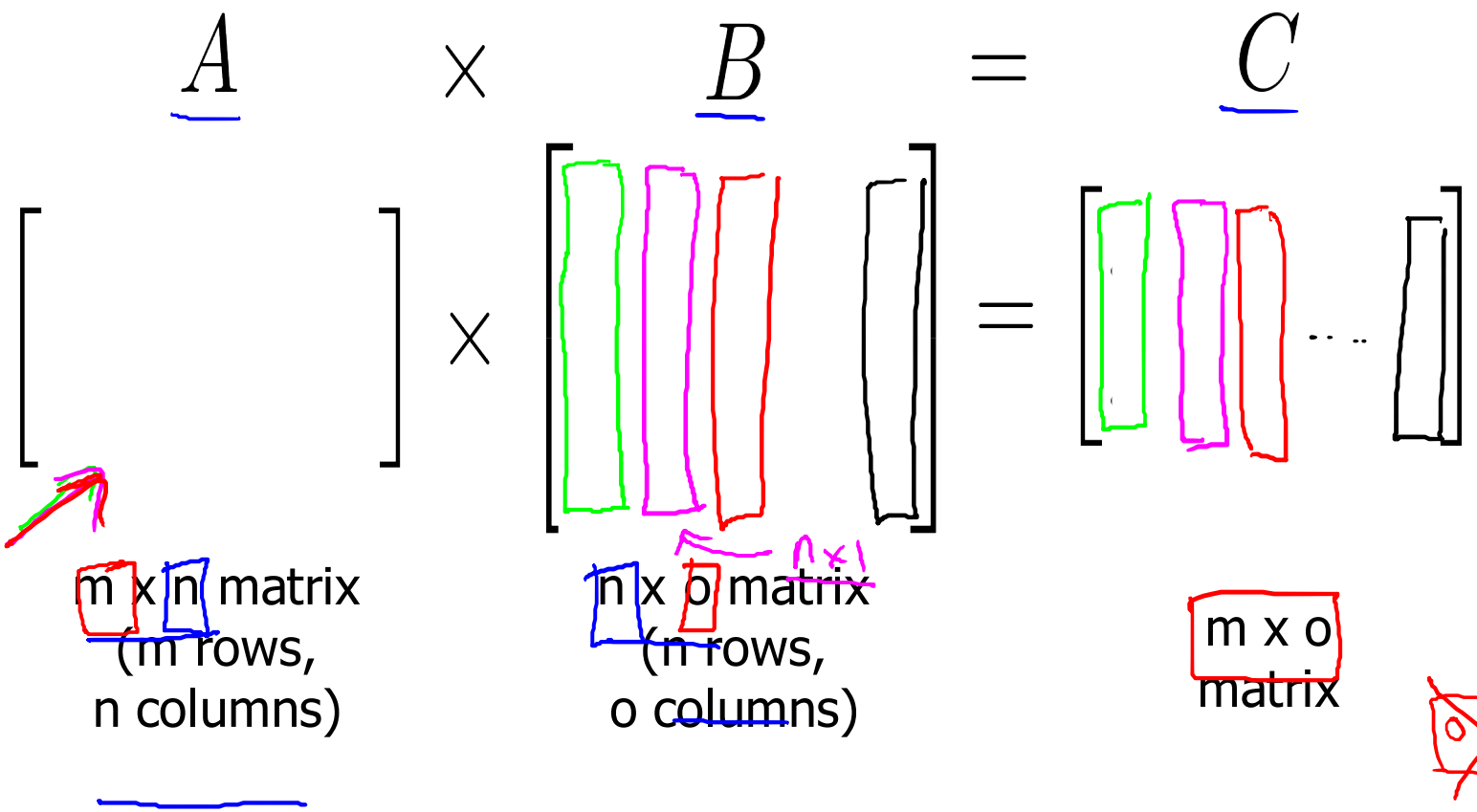
m-dimensional vector

To get y_i, multiply A's i^{th} row with elements of vector x, and add them up.

Example: Matrix Multiplication

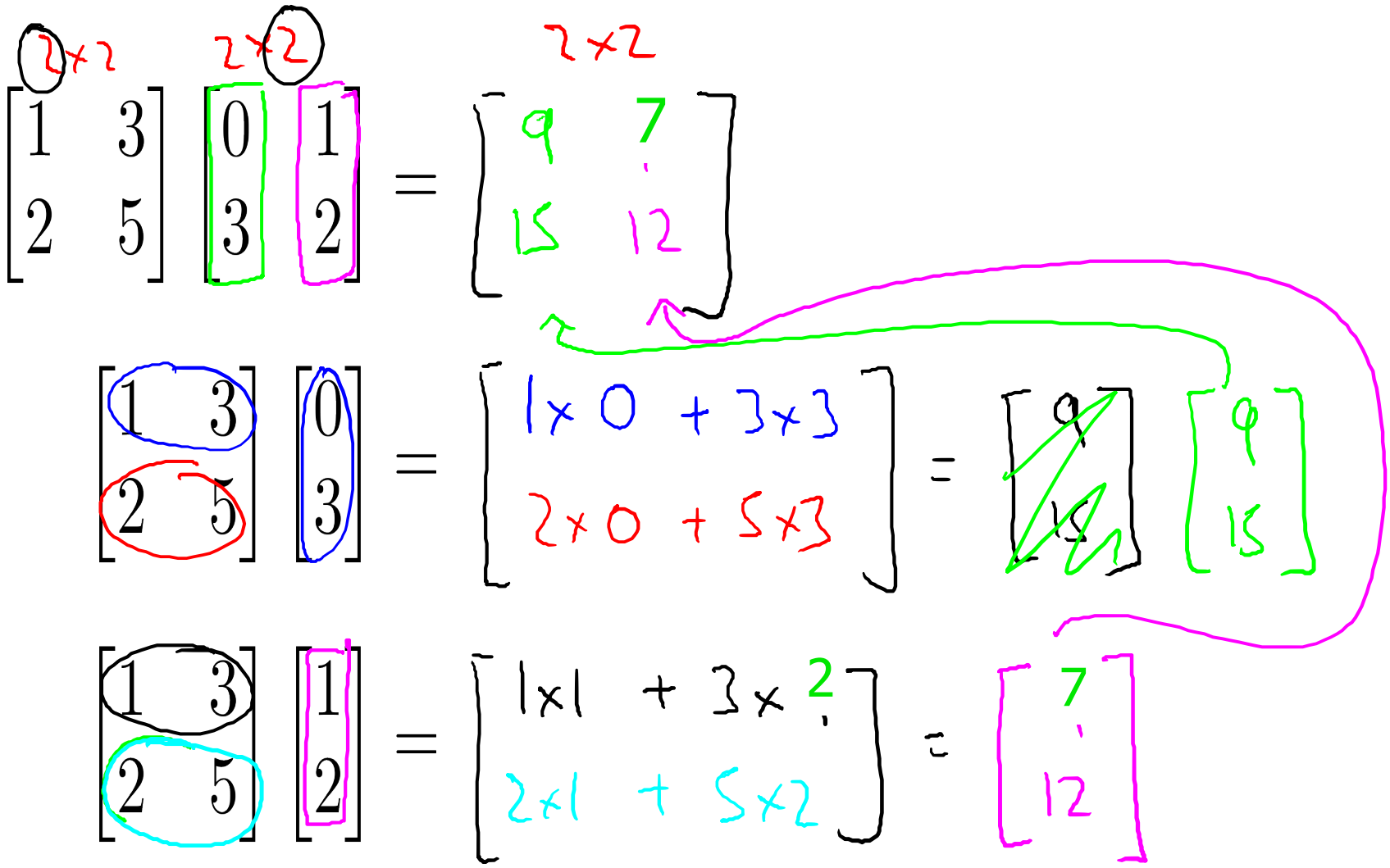


Details:



The i^{th} column of the matrix C is obtained by multiplying A
with the i^{th} column of B. (for i = 1,2,…,o)

Example


2×2
2×2
2×2
$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 9 & 7 \\ 15 & 12 \end{bmatrix}$
$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 1\times 0 + 3\times 3 \\ 2\times 0 + 5\times 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 15 \end{bmatrix} \begin{bmatrix} 9 \\ 15 \end{bmatrix}$
$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1\times 1 + 3\times 2 \\ 2\times 1 + 5\times 2 \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \end{bmatrix}$

Sparse matrix

- Sparse matrix
  - A sparse matrix is a matrix in which most of the elements are zero.

$$\begin{bmatrix} 1.1 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 1.9 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 2.6 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 7.8 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.5 & 2.7 & 0 & 0 \\ 1.6 & 0 & 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9 & 1.7 \end{bmatrix}$$

- Spars matrix representations (Row i , Column j, value)

Matrix Multiply Example

- Matrix representations (Row i , Column j, value)
- Matrix Multiply:

C = A X B

A has dimensions L x M

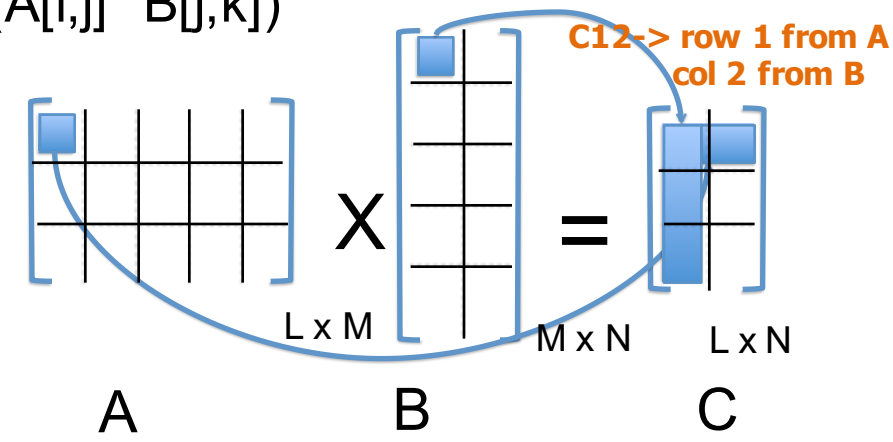
B has dimensions M x N

C has dimensions L x N

Matrix multiplication: $C[i,k] = Sum_j (A[i,j] * B[j,k])$

C11->row 1 from A
col 1 from B

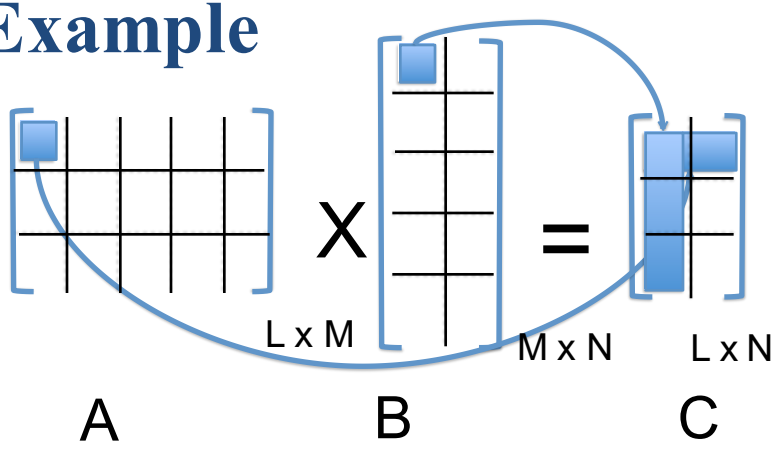
C12-> row 1 from A
col 2 from B



Key = (i,k) for k in 1..N
Value = A[i,j]

Key = (i,k) for i in 1..L
Value = B[j,k]

Matrix Multiply Example



C = A X B

A has dimensions L x M

B has dimensions M x N

C has dimensions L x N

Matrix multiplication: $C[i,k] = Sum_j (A[i,j] * B[j,k])$

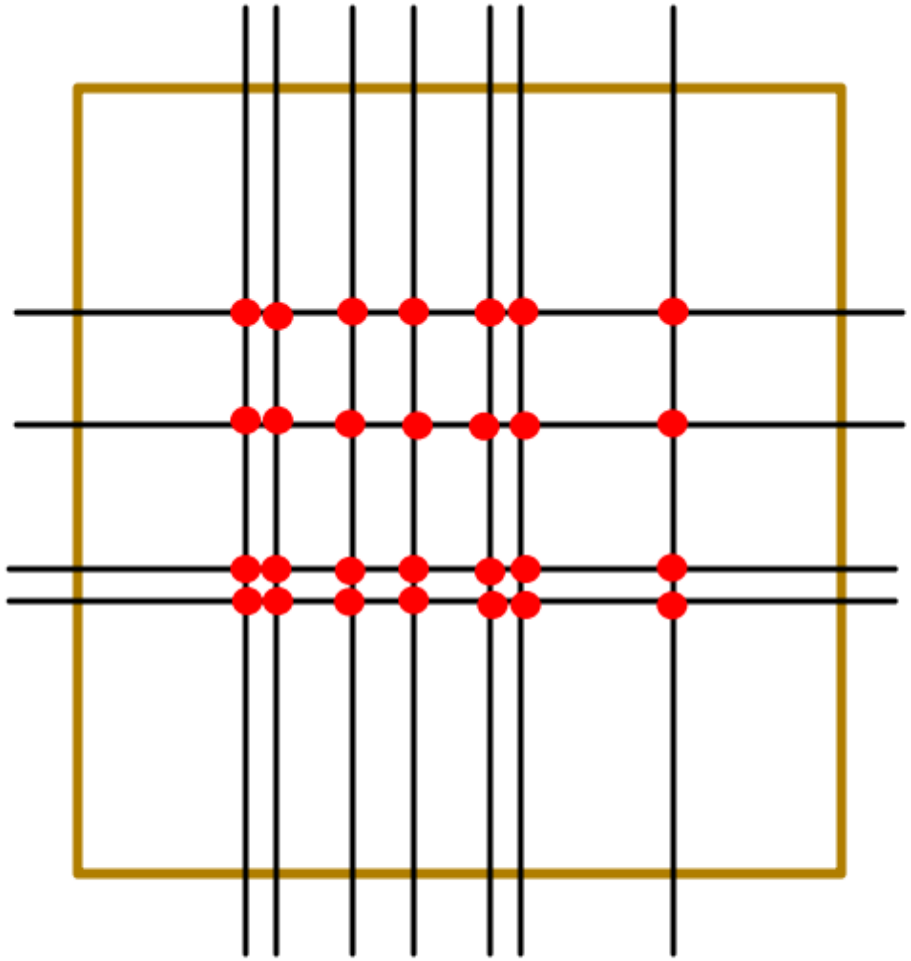
In the map phase:

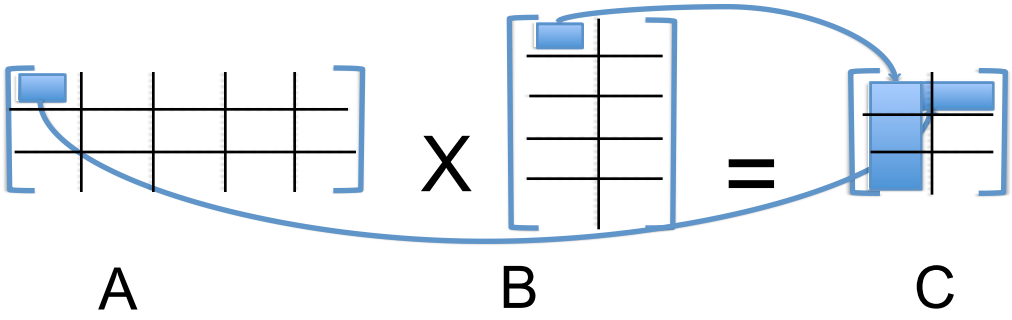
- for each element (i,j) of A, **emit ((i,k), A[i,j])** for **k in 1..N**
  - **Better: emit ((i,k)('A', i, k, A[i,j])) for k in 1..N**
- for each element (j,k) of B, **emit ((i,k), B[j,k])** for **i in 1..L**
  - **Better: emit ((i,k)('B', i, k, B[j,k])) for i in 1..L**

In the reduce phase One reducer per output cell, emit

- **key = (i,k)**
- **value = Sum_j (A[i,j] * B[j,k])**

The Responsibility of One Reducer





C[i,k] = Sum$_j$ (A[i,j] *B[j,k]], C is LxN

In the map phase:

- for each element (i,j) of A, **emit ((i,k)('A', i, k, A[i,j])) for k in 1..N**
- for each element (j,k) of B, **: emit ((i,k)('B', i, k, B[j,k])) for i in 1..L**

Example (what is needed at the reducer):

C[1,1] = A[1,1] * B[1,1] + A[1,2] * B[2,1] + A[1,3] * **B[3,1]** + A[1,4] * B[4,1] + A[1,5] * B[5,1]

C[1,2] = A[1,1] * B[1,2] + A[1,2] * B[2,2] + A[1,3] * B[3,2] + A[1,4] * B[4,2] + A[1,5] * B[5,2]

C[2,1] = A[2,1] * B[1,1] + A[2,2] * B[2,1] + A[2,3] * **B[3,1]** + A[2,4] * B[4,1] + A[2,5] * B[5,1]

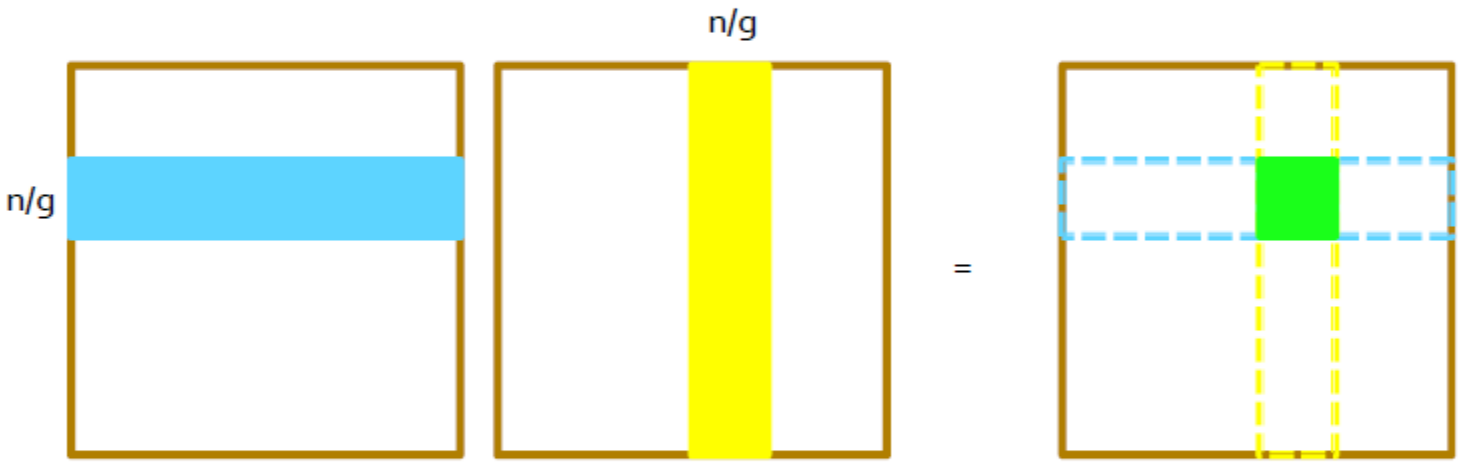
Map phase: For A[1,2], emit ('A', 1, k, A[1,2]) for k in 1..2

emit ((1,1)('A', 1, 1, A[1,2])), ((1,2)('A', 1, 2, A[1,2]))

For **B[3,1]**: emit ('B', i, 1, B[3,1]) for i in 1..3

emit ((1,1)('B', 1, 1,B[3,1])), ((2,1)('B', 2, 1,B[3,1])), ((3,1)('B', 3, 1,B[3,1]))

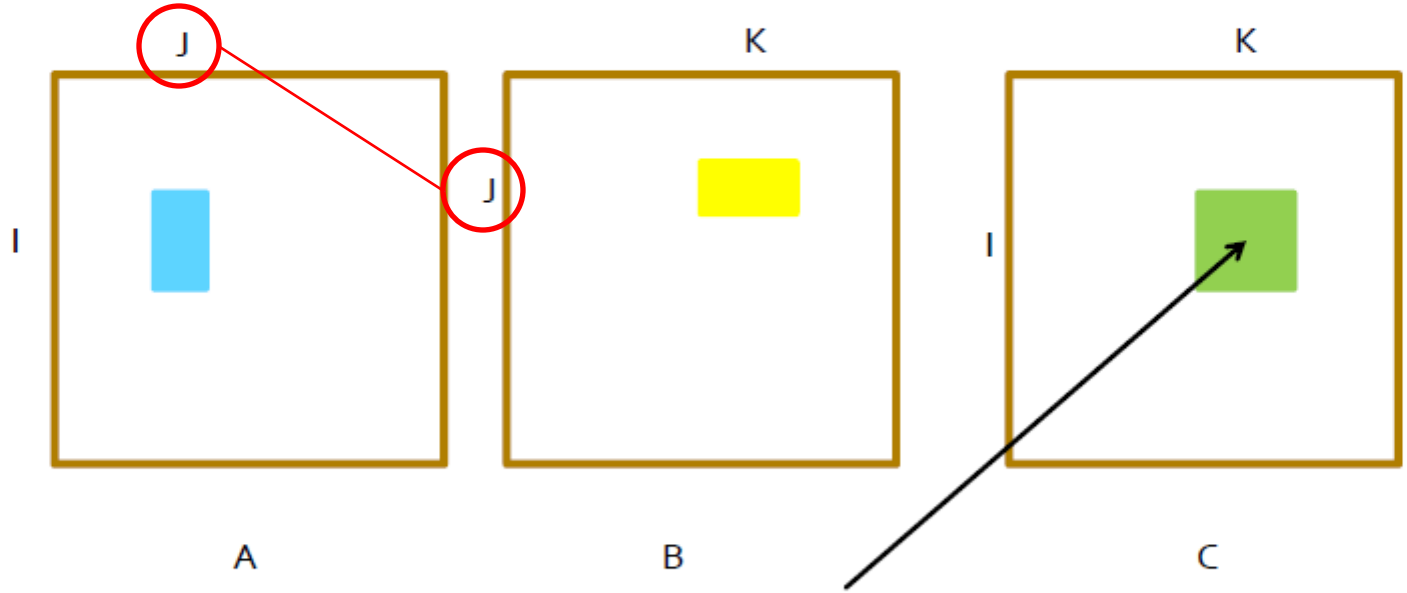
Picture of One Reducer



Two-phase Map Reduce Matrix Multiply

- **A better way:** use two map reduce jobs.
- Job 1: Divide both input matrices into rectangles.
  - Reducer takes two rectangles and produces partial sums of certain outputs.
- Job 2: Sum the partial sums.

Picture of First Job



For *i* in I and *k* in K, contribution is $\Sigma_{j \text{ in } J} A_{ij} \times B_{jk}$

Two-phase Map Reduce Matrix Multiply

- Farhan Khwaja
- **Idea: Multiply the appropriate values on the first MapReduce phase, Add on second MapReduce phase**
- $C[i,k] = Sum_j (A[i,j] * B[j,k])$
- C[1,1] = A[1,1] * B[1,1] + A[1,2] * B[2,1] + A[1,3] * B[3,1] + A[1,4] * B[4,1] + A[1,5] * B[5,1]
- C[1,2] = A[1,1] * B[1,2] + A[1,2] * B[2,2] + A[1,3] * B[3,2] + A[1,4] * B[4,2] + A[1,5] * B[5,2]
- 1st Map Function:

For each matrix element **A[ij] : emit(j , (A, i, A[i,j])**

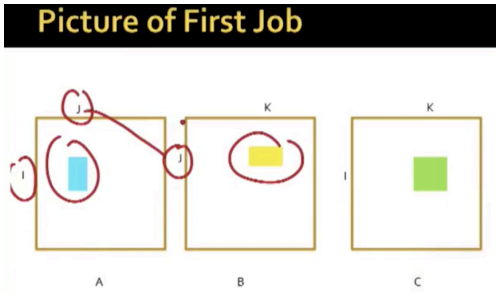
For each matrix element **B[jk] : emit(j , (B, k, B[j,k])**

Example (all will go to same reducer with key 2):

For A[1,2]: emit (2, (A, 1, A[1,2])

For B[2,1]: emit (2, (B, 1, B[2,1])

For B[2,2]: emit (2, (B, 2, B[2,2])



Two-phase Map Reduce Matrix Multiply

- 1st Reduce Function:

Each key j will have its own Reduce function, will produce all products for a given value of j

For each value of **(i,k)** which comes from A and B, i.e. **(A, i, A[ij])** and **(B, k, B[jk]):** **emit((i,k), (A[ij] * B[jk]))**

Example: Input from Map task

For A[1,2]: emit (2, (A, 1, A[1,2])

For B[2,1]: emit (2, (B, 1, B[2,1])

For B[2,2]: emit (2, (B, 2, B[2,2])

Reduce task for key j:

emit ((1,1), A[1,2] * B[2,1])

emit ((1,2), A[1,2] * B[2,2])

- C[1,1] = A[1,1] * B[1,1] + A[1,2] * B[2,1] + A[1,3] * B[3,1] + A[1,4] * B[4,1] + A[1,5] * B[5,1] **(Note: every term has key (1,1))**
- C[1,2] = A[1,1] * B[1,2] + A[1,2] * B[2,2] + A[1,3] * B[3,2] + A[1,4] * B[4,2] + A[1,5] * B[5,2] **(Note: every term has key (1,2)**

Two-phase Map Reduce Matrix Multiply

- 2nd Map Function:
  map(key,value):

#Let the pair of (**((i,k), (A[ij] * B[jk]))** pass through

- 2nd Reduce Function:

reduce(key,values):

#each (i,k) will have its own reduce function

- For each **(i,k)** we will add up the values, thus emitting:
  emit((i,k),Sum(values))

- Example: C[1,1] = A[1,1] * B[1,1] + A[1,2] * B[2,1] + A[1,3] * B[3,1] + A[1,4] * B[4,1] + A[1,5] * B[5,1]
- **(Note: every term has key (1,1), will go to same reducer).** √

More Examples: Relational Join

Employee

Name	SSN
Sue	999999999
Tony	777777777

Assigned Departments

EmpSSN	DepName
999999999	Accounts
777777777	Sales
777777777	Marketing

Emplyee ⋈ Assigned Departments

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

Relational Join in MapReduce: Before Map Phase

Key idea: Lump all the tuples together into one dataset

Employee

Name	SSN
Sue	999999999
Tony	777777777

Assigned Departments

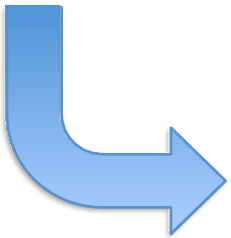
EmpSSN	DepName
999999999	Accounts
777777777	Sales
777777777	Marketing

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

What is this for?

Relational Join in MapReduce: Map Phase

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



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?

- Map Task: Emit (key, value) pair
- **Key is unique ID** in each table, used for join
- **Value is a tuple** with all fields from table (including key).

Relational Join in MapReduce: Reduce Phase

key=999999999, values=[(Employee, Sue, 999999999),
(Department, 999999999, Accounts)]

Sue, 999999999, 999999999, Accounts

key=777777777, values=[(Employee, Tony, 777777777),
(Department, 777777777, Sales),
(Department, 777777777, Marketing)]

Tony, 777777777, 777777777, Sales
Tony, 777777777, 777777777, Marketing

- **Group by Key:** groups together all values (tuples) associated with each key
- **Reduce task:** emit joined values (without table names).

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 Task

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)

Distributed Sort

- Sort a very large list of (firstName, lastName) pairs by lastName followed by firstName

- Map task:?
- Reduce task:?

Examples of outputs:

Smith Anne

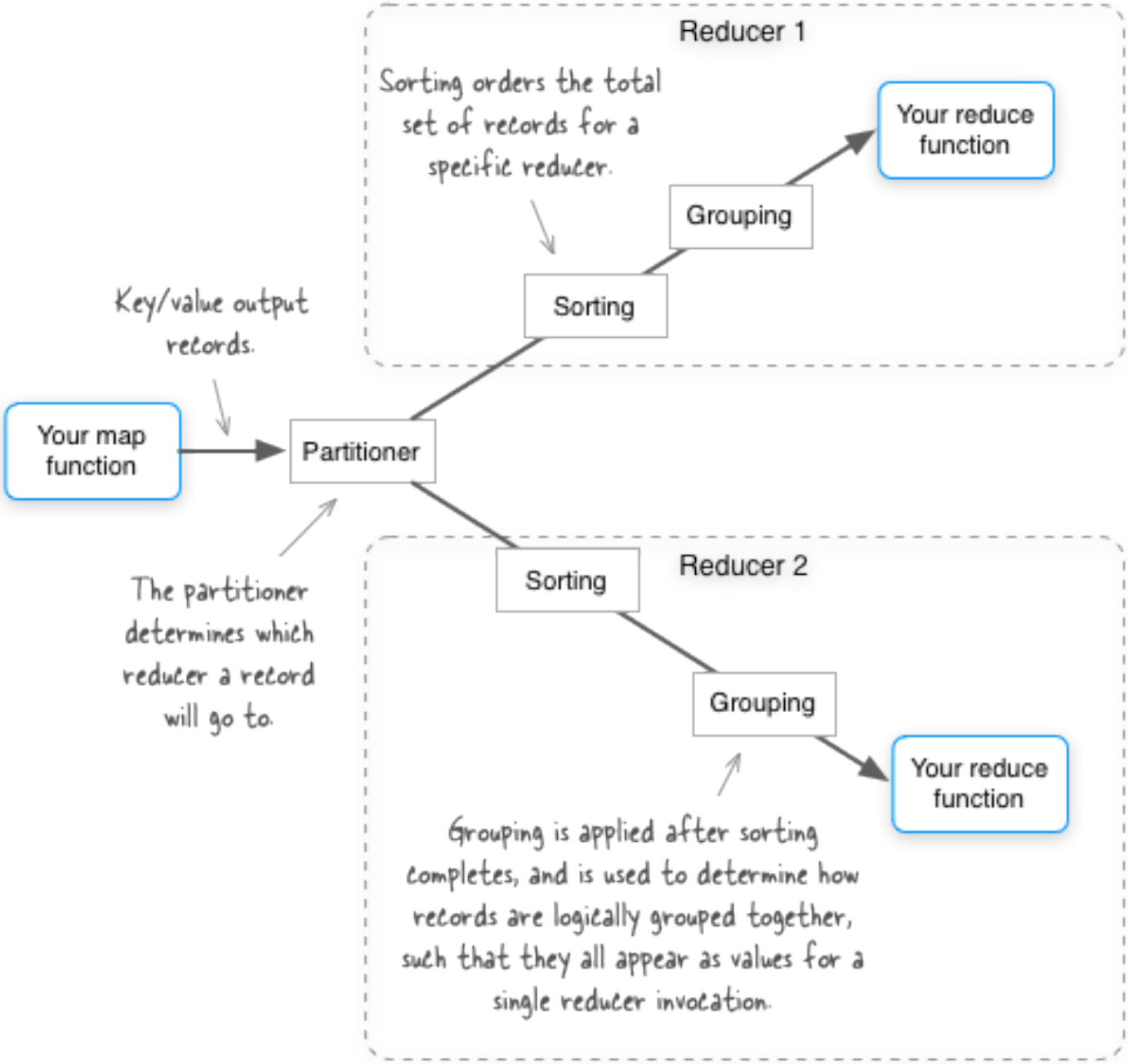
Smith John

Smith Ken

Distributed Sort explanation

Sort by lastName followed by firstName

- Map task
  - Emit (lastName, firstName)
- Group by keys:
  - Group together entries with same last name
  - Divide into non-overlapping alphabetical ranges
  - Keys are sorted in alphabetical order
- Reduce task
  - Processes one key at a time (in alphabetical order)
  - For each lastName key, if there are multiple firstName values, will emit (lastName, firstName) in alphabetical order
  - Merge output from all Reduce tasks.


Reducer 1
Sorting orders the total set of records for a specific reducer.
Your reduce function
Grouping
Sorting
Key/value output records.
Your map function
Partitioner
The partitioner determines which reducer a record will go to.
Sorting
Reducer 2
Grouping
Your reduce function
Grouping is applied after sorting completes, and is used to determine how records are logically grouped together, such that they all appear as values for a single reducer invocation.

Example: Join By Map-Reduce

- **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

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Map-Reduce Join

- **Use a hash function *h* from B-values to *1…k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

Cost Measures for Algorithms

- **In MapReduce we quantify the cost of an algorithm using**

1. *Communication cost* = **total I/O** of all processes
2. *Elapsed communication cost* = **max of I/O** along any path
3. (*Elapsed*) *computation cost* analogous, but count only **running time of processes**.

Note that here the big-O notation is not the most useful (adding more machines is always an option)

Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost =**
    - **input file size +**
    - **sum** of the sizes of all **files** passed from **Map processes to Reduce processes +**
    - the sum of the **output sizes** of the **Reduce** processes.
  - **Elapsed communication cost** is the **sum of the largest input** + output for any map process, plus the same for **any reduce** process.

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other
- Total cost tells what you **pay in rent** from your friendly neighborhood **cloud**
- Elapsed cost is **wall-clock** time using parallelism.

Cost of Map-Reduce Join

- **Total communication cost**
  = O(|R|+|S|+|R ⋈ S|)
- **Elapsed communication cost** = O(s)
  - Choose the number of Map processes so that the I/O limit ***s*** is respected
  - ***s*** isthe amount of input or output that any one process can have
  - ***s* could be:**
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. Cost.

Problems Suited for MapReduce

General Characteristics of Good Problems for MapReduce

Data set is truly "big"

- **Terabytes**, not tens of gigabytes
- Hadoop/MapReduce designed for terabyte/petabyte scale computation
- Most **real-world problems** process less than 100 Gbytes of input
  - Microsoft, Yahoo: median job under 14 GB
  - Facebook: 90% of jobs under 100 GB.

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

General Characteristics of Good Problems for MapReduce

Don't need fast response time

- When submitting jobs, Hadoop latency can be a minute
- **Not well-suited** for problems that require faster response time
  - online purchases, transaction processing
- A good pre-computation engine
  - E.g., pre-compute related items for every item in inventory.

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

General Characteristics of Good Problems for MapReduce

Good for applications that work in **batch mode**

- Jobs run without human interaction, intervention
- Runs over entire data set
  - Takes time to initiate, run; shuffle step can be time-consuming
- Does **not support real time** applications well: sensor data, real-time advertisements, etc.
- Does **not provide** good support for **random access** to datasets
  - Extensions: Hive, Dremel, Shark, Amplab.

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

General Characteristics of Good Problems for MapReduce

- Best suited for data that can be expressed as key-value pairs without losing context, dependencies
  - Graph data harder to process using MapReduce
  - Implicit relationships: edges, sub-trees, child/parent relationships, weights, etc.
  - Graph algorithms may need information about the entire graph for each iteration
    - Hard to break into independent chunks for Map tasks
  - Alternatives: Google's Pregel, Apache Giraph.

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

General Characteristics of Good Problems for MapReduce

Other problems/data ***not*** suited for MapReduce

- Tasks that need results of intermediate steps to compute results of current step
  - Interdependencies among tasks
  - Map tasks must be independent
- Some machine learning algorithms
  - Gradient-based learning, expectation maximization.

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

Problems NOT suitable for MapReduce

- **MapReduce is great for:**
  - Problems that require **sequential data access**
  - Large batch jobs (**not** interactive, real-time)
- **MapReduce is inefficient for problems where random (or irregular) access to data required:**
  - **Graphs**
  - Interdependent data
    - Machine learning
    - Comparisons of many pairs of items.

General Characteristics of Good Problems for MapReduce

Summary:

Good candidates for MapReduce:

- Jobs that have to **process huge quantities of data** and either **summarize or transform** the contents
- **Collected data** has elements that can easily be captured with an identifier **(key)** and corresponding **value.**

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy

MapReduce inside Google

Google

Googlers' hammer for 80% of our data crunching

- Large-scale web search indexing
- Clustering problems for Google News
- Produce reports for popular queries, e.g. Google Trend
- Processing of satellite imagery data
- Language model processing for statistical machine translation
- Large-scale machine learning problems
- Just a plain tool to reliably spawn large number of tasks
  - e.g. parallel data backup and restore

The other 20%? e.g. Pregel