Introduction to Data Science

(Lecture 23)

Dr. Mohammad Pourhomayoun

Assistant Professor

Computer Science Department

California State University, Los Angeles

Map Reduce

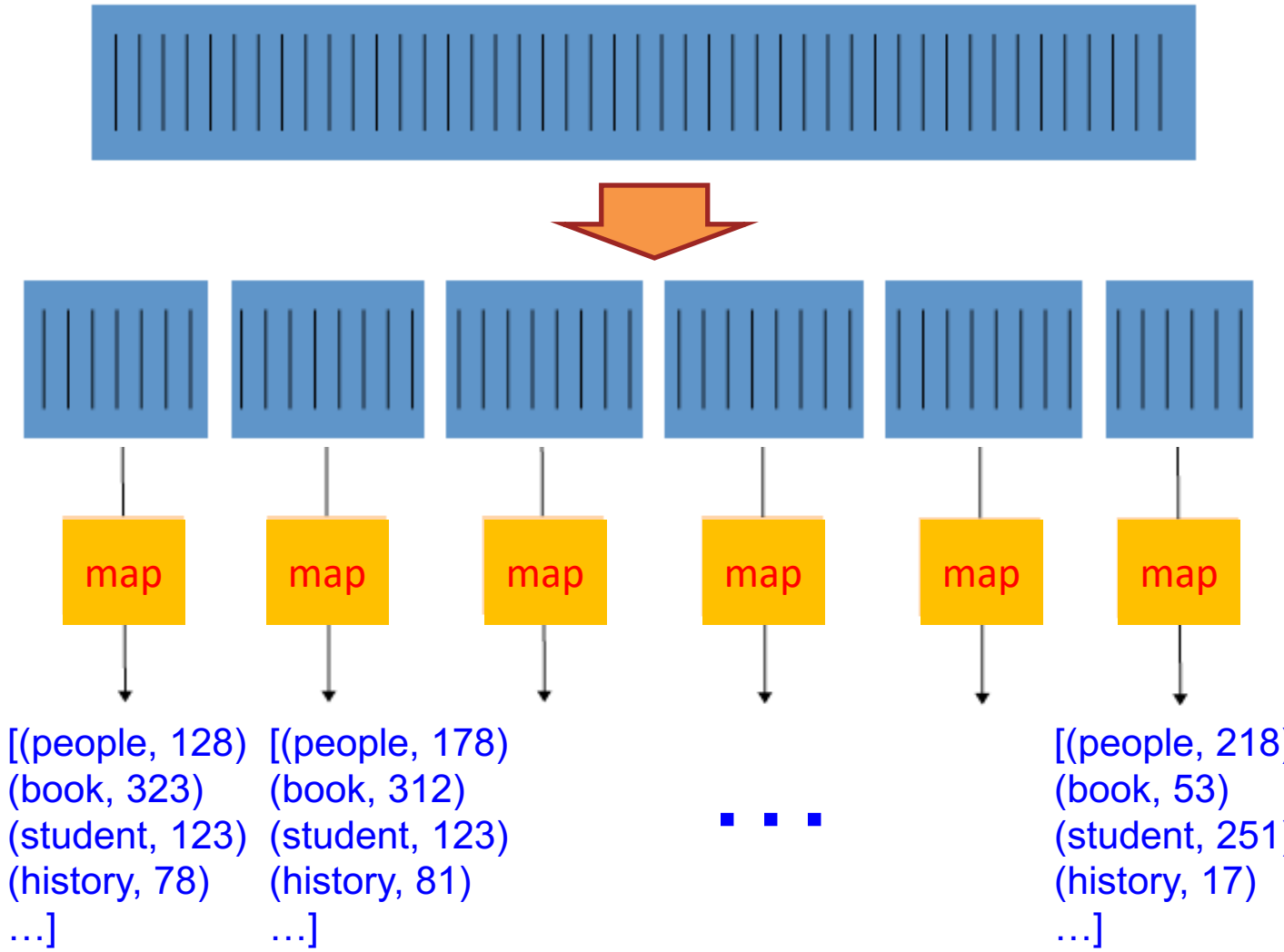
Map-Reduce

- **Map-Reduce** is a programming model for processing and generating big data sets with a parallel and distributed algorithm.

- **map function** processes input key/value pairs to generate a set of intermediate key/value pairs.
- **reduce function** merges all intermediate values associated with the same intermediate key.

[Ref]: Dean, Jeffrey & Ghemawat, Sanjay. (2004). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM.

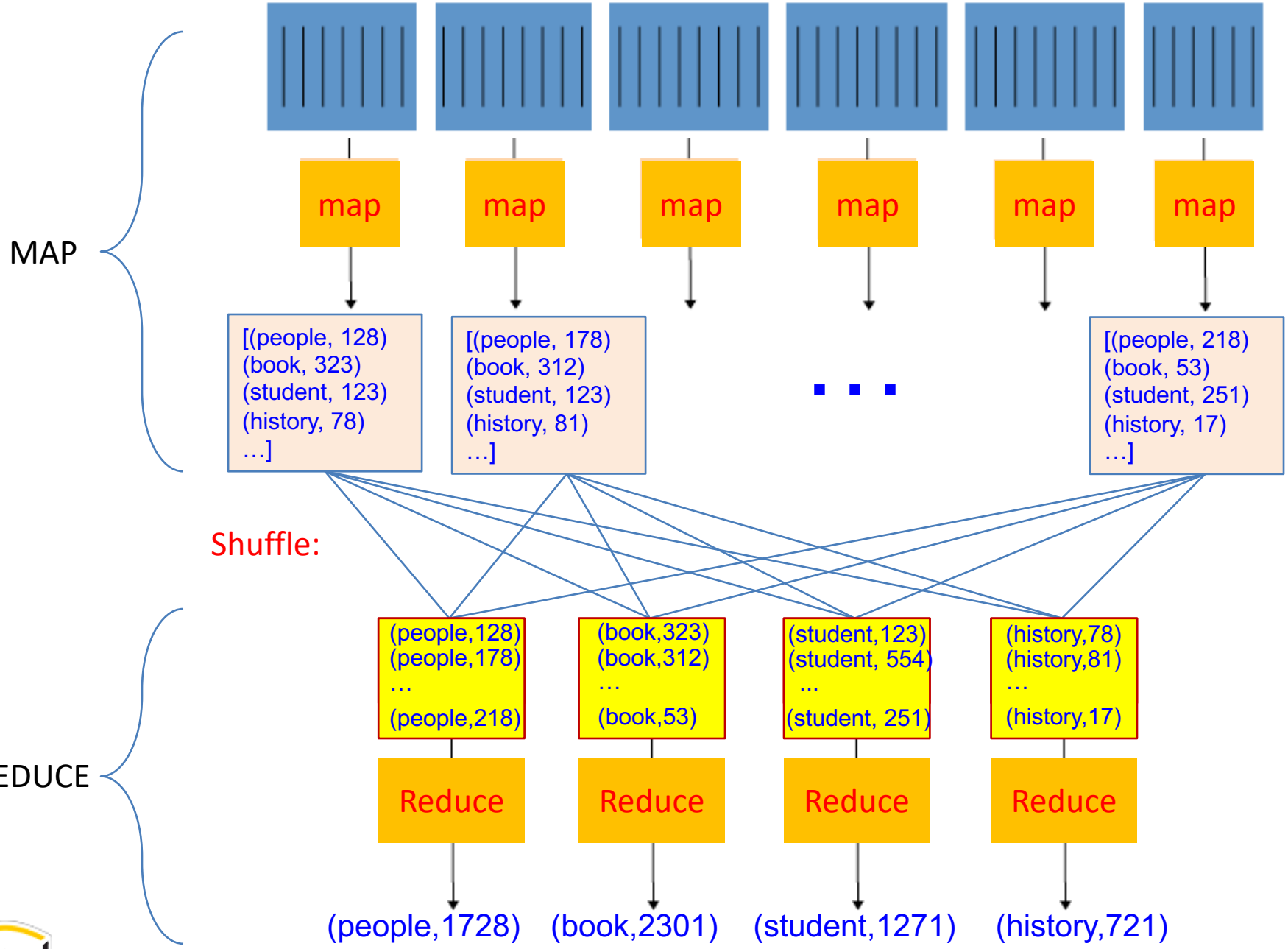
Example: Compute overall word frequency across 5M docs



You have millions of documents

Distribute the documents among k computers

For each document f returns a set of **(word, freq)** pairs



Map: Count **All words in Each chunk of data**

Reduce: Count the occurrences of **Each word in the Entire data**

MAP-REDUCE

- In this example:
  - **Map:** Counts **All** words in **Each** chunk of data
  - **Reduce:** Counts the occurrences of **Each** word in the **Entire** dataset

MapReduce Programming Model

- **Input** : a set of (in_key , in_value) pairs
- **Output**: a set of (out_key , out_value) pairs

- Programmer specifies two functions:

map (in_key, in_value) -> list (out_key, intermediate_value)

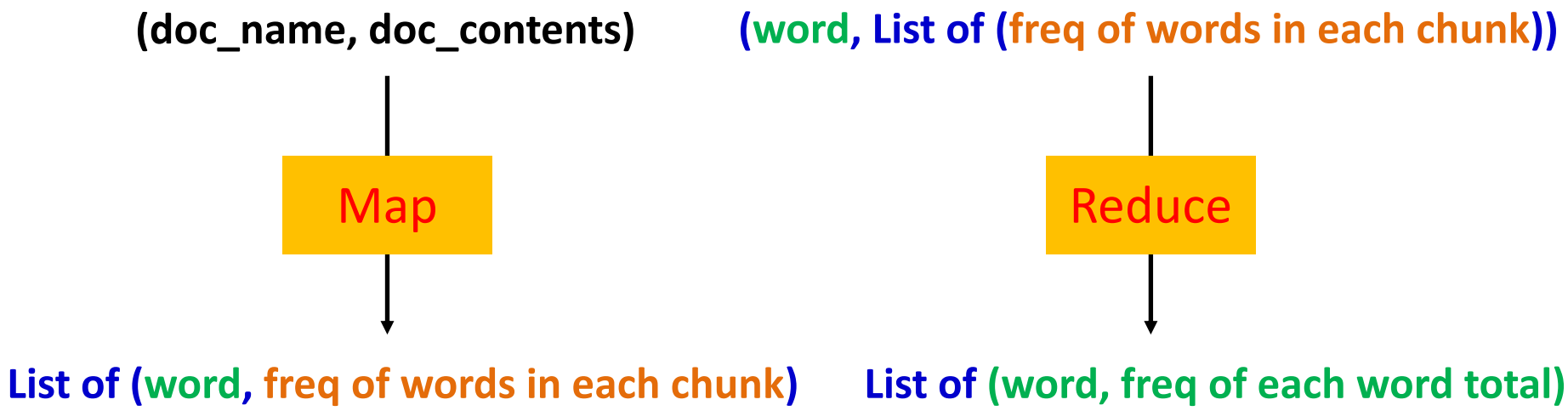
- Processes input (key,value) pairs
- Produces set of intermediate pairs

reduce (out_key, list(intermediate_value)) -> list (out_key, out_value)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

Inputs and Outputs

- Input : a set of (in_key,in_value) pairs
- Output: a set of (out_key,out_value) pairs



- **Very important to distinguish between (in_key, in_value) and (out_key, out_value)!!!**

```
map(String input_key, String input_value):
        # input_key: document name/id
        # input_value: document contents
Intermediate_values = {}
for each word w in input_value:
        Intermediate_values[w] += 1
```

```
reduce(String output_key, List of [intermediate_values]):
        # output_key: word
        # output_values: freq of each word
output_values = 0;
for each v in intermediate_values:
        output_values += v
return(output_key,output_values )
```

Map Reduce for Relational Database Operations

Example: Relational Join

- **Relational Join**: Stick the tuples of two relations together when they agree on common attributes (column names).
- Consider two database tables R(A, B) and S(B, C).
- **R JOIN S** : T(A, B, C) formed by joining rows (a, b) ϵ R and (b, c) ϵ S with matching b.
- Example: R(A,B) JOIN S(B,C) = {a,b,c | a,b is in R and b,c is in S}.

A	B
6	2
12	2
7	5

R

B	C
2	9
5	11
5	3
9	5

S

A	B	C
6	2	9
12	2	9
7	5	11
7	5	3

R JOIN S

The Map Function for Join

- Each tuple (a,b) in R is mapped to:

  key = b, value = (R,a).

  – Note: "R" in the value is just a bit to indicate "this value represents a tuple in R, not S."

- Each tuple (b,c) in S is mapped to:

  key = b, value = (S,c).

- After grouping by keys (shuffle), each reducer gets a key-list that looks like:

  (b, [(R,a_1), (R,a_2),…, (S,c_1), (S,c_2),…]).

The Map Function for Join

R =

A	B
6	2
12	2
7	5

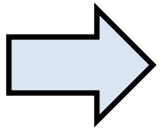
S =

B	C
2	9
5	11
5	3
9	5

Map on R:
- (2, (R,6))
- (2, (R,12))
- (5, (R,7))

Map on S:
- (2, (S,9))
- (5, (S,11))
- (5, (S,3))
- (9, (S,5))

Shuffle



(2, [(R,6), (R,12), (S,9)])
(5, [(R,7), (S,11), (S,3)])
(9, [(S,5)])

The Reduce Function for Join

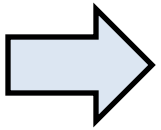
- After grouping by keys, each reducer gets a key-list that looks like:

$$(b, [(R,a_1), (R,a_2),\ldots, (S,c_1), (S,c_2),\ldots]).$$

- Reducer generates a tuple (a,b,c) for **each** pair of (R,a_i) and (S,c_j) on the list with key b.

The Reduce Function for Join

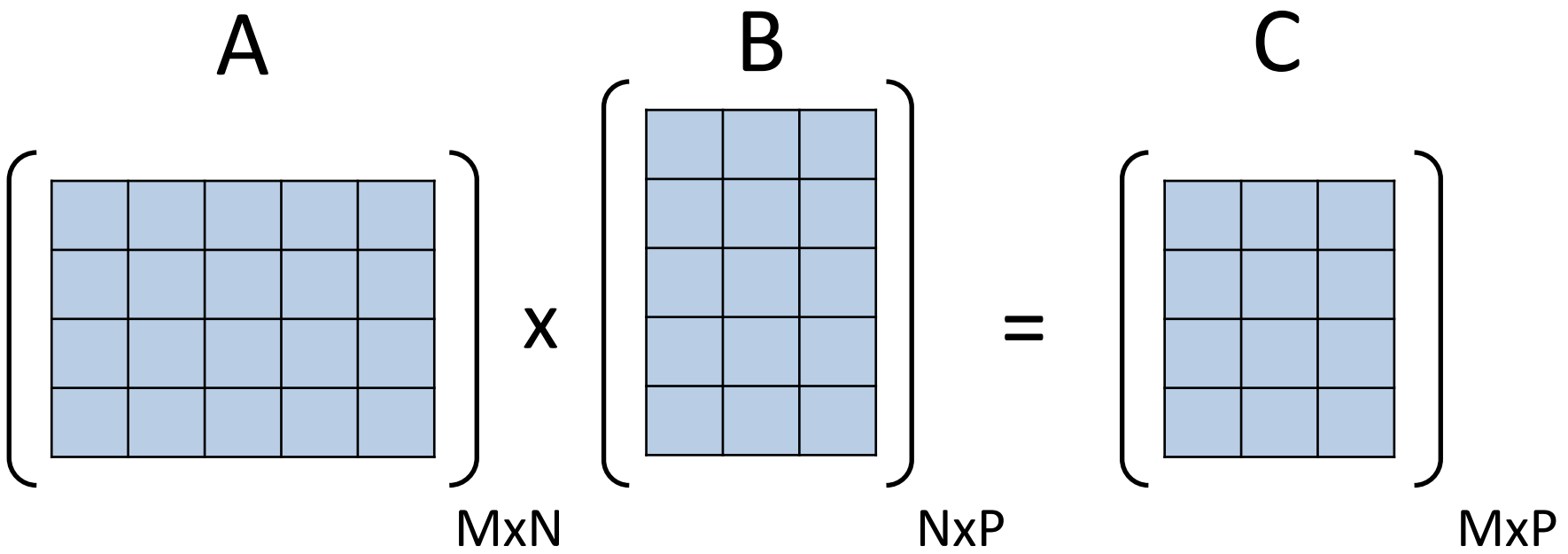
(2, [(R,6), (R,12), (S,9)])
(5, [(R,7), (S,11), (S,3)])
(9, [(S,5)])



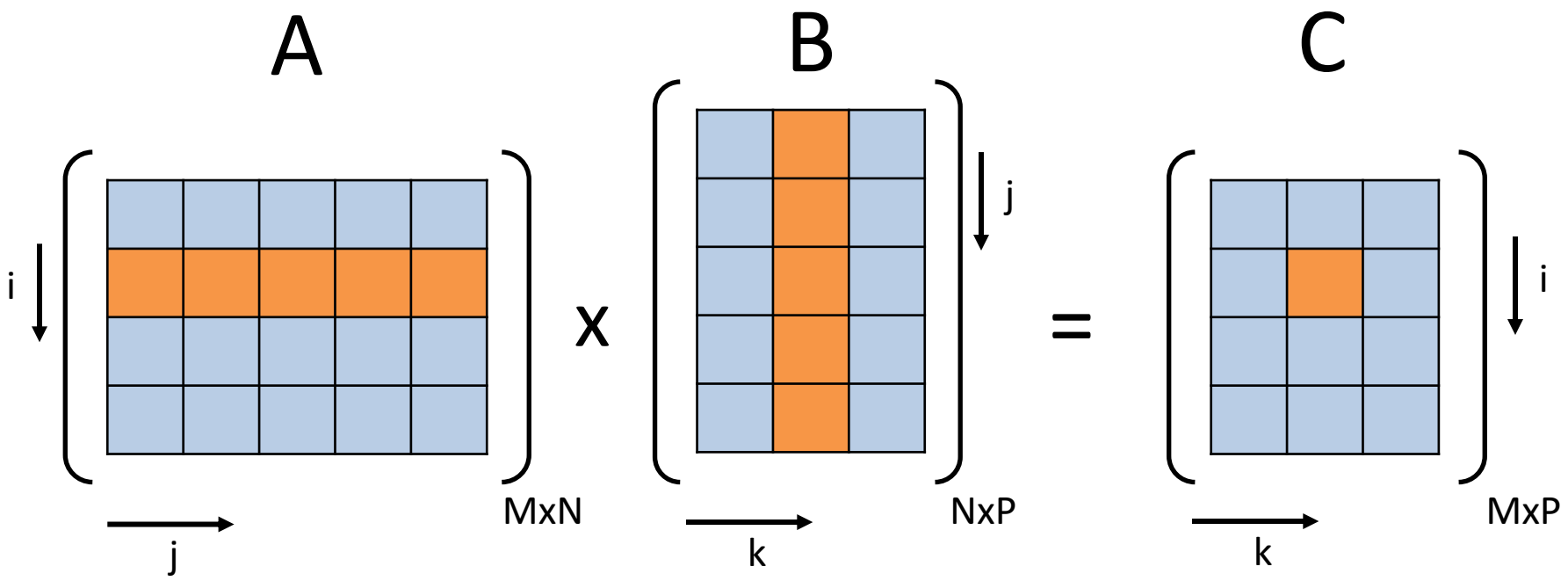
A	B	C
6	2	9
12	2	9
7	5	11
7	5	3

Map Reduce
for Matrix and Vector Multiplication

MapReduce for Matrix Multiplication



Matrix Multiplication



$$c_{ik} = \sum_{j} a_{ij} \times b_{jk}$$

Example

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

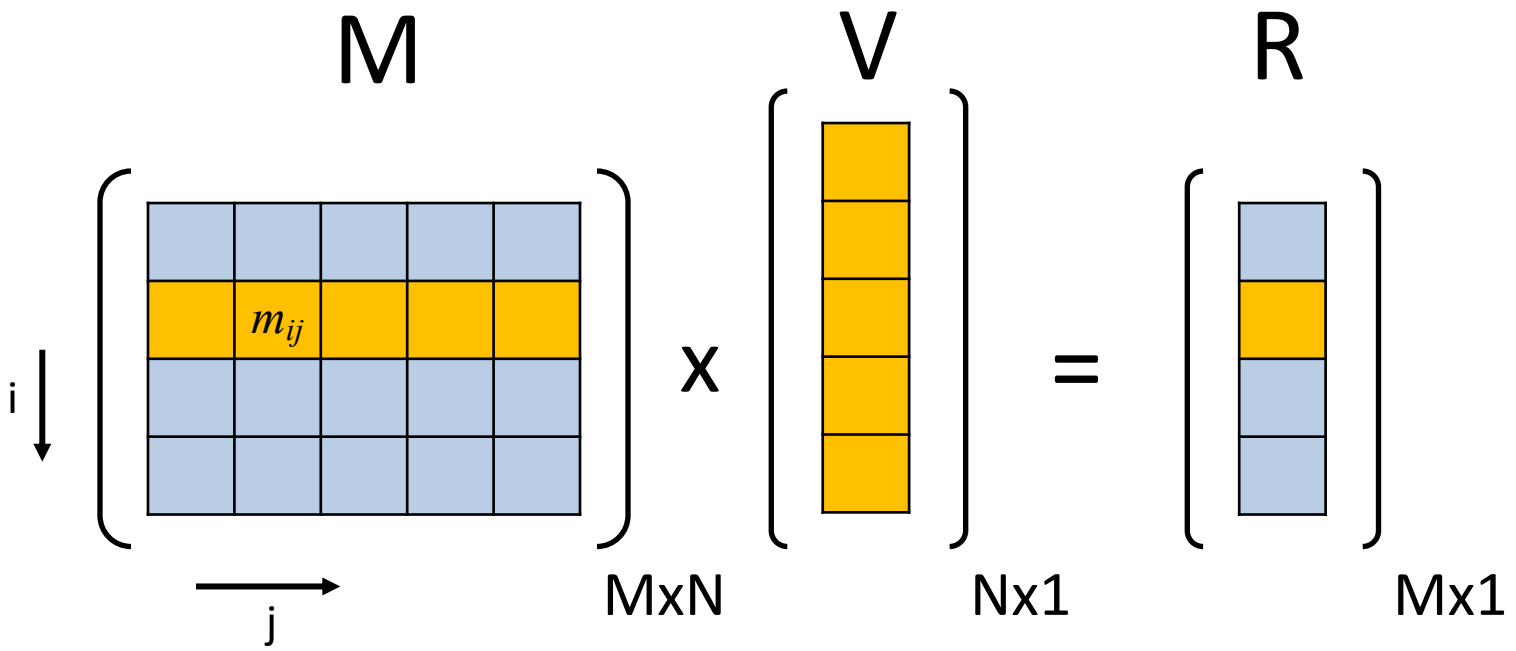
$$= \begin{bmatrix} 15 & 16 \\ 20 & 27 \end{bmatrix}$$

Simple Pseudo Code for Matrix Multiplication (general approach)

```
for i = 1 to n do
      for j = 1 to n do
            for k = 1 to n do
                  C[i,j] = C[i,j] + A[i,k] x B[k,j]
            endfor
       endfor
 endfor
```

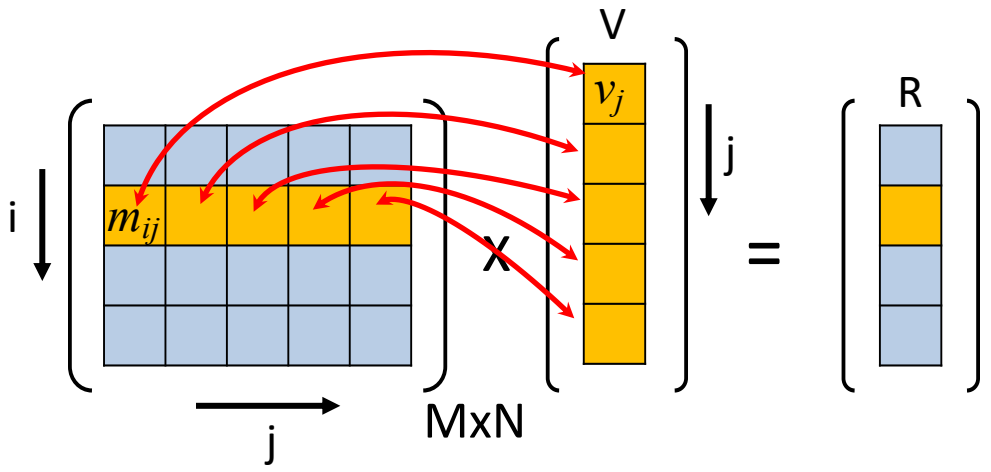
Special Case: Matrix-Vector Multiplication

- Map-Reduce originally developed by **Google** in order to compute the **PageRank** vector.



Map for Matrix-Vector Multiplication

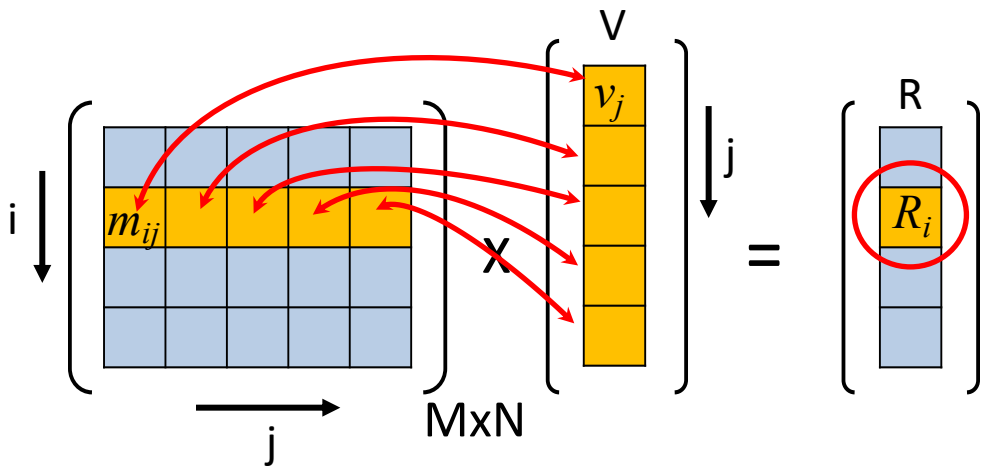
- **Map Function:**
  - N mappers process row i of the matrix at a time
  - Mapper j Maps $((i, j), m_{ij})$ to $(i, m_{ij}\, v_j)$
  - Note that We assumed that each mapper can load vector v.



Reduce for Matrix-Vector Multiplication

- **Reduce function:**
  - M reducers calculate elements of vector R
  - Reducer i receives $(i, [m_{i1}\, v_1, \ldots m_{iN}\, v_N])$, sums all values of the list of a key i, and produces (i, R_i):

$$R_i = \sum_j m_{ij} \times v_j$$



Map Reduce
for Matrix to Matrix Multiplication
(Optional)

MapReduce for Matrix-Matrix Multiplication (Optional: Not in Final Exam)

- Multiply matrix $\mathrm{M} = [m_{ij}]$ by $\mathrm{N} = [n_{jk}]$.
- $\mathrm{P} = \mathrm{M}$ x N:

$$\mathrm{P} = [p_{ik}], \text{ where } p_{ik} = \Sigma_j \, m_{ij} \text{ x } n_{jk}.$$

- In many applications, Large matrices are very sparse (mostly 0's). Thus, to save memory, we can represent a sparse matrix only using its nonzero elements m_{ij} in the form of tuples (i, j, m_{ij}). zero elements are not represented at all!

- The are two main approaches for Matrix Multiplication using MapReduces: **Single Pass** and **Double Pass**.

Single-Pass MapReduce for Matrix Multiplication (Optional: Not in Final Exam)

- **Map function:**
  - maps each m_{ij} to **key = (i,k), value = (M,j,m_{ij})**, for all k
  - maps each n_{jk} to **key = (i,k), value = (N,j,n_{jk})**, for all i
  - Similar to join, M and N here are bits indicating which matrix the value comes from.

Single-Pass MapReduce for Matrix Multiplication (Optional: Not in Final Exam)

- **Reduce function:**
  - For each $(\mathbf{M}, j, m_{ij})$ on the list for **key** (i,k) find the $(\mathbf{N}, j, n_{jk})$ with the **same** j.
  - Multiply m_{ij} by n_{jk} and then sum up to generate $((i,k), p_{ik})$:

$$p_{ik} = \sum_{j} m_{ij} \times n_{jk}$$

  - $((i,k), p_{ik})$ means that the element (i,k) of the output matrix has value of p_{ik}!

Double-Pass MapReduce for Matrix Multiplication (Optional): The First Pass

- **Map function:**
  - maps each m_{ij} to **key = j, value = $(\mathbf{M}, i, m_{ij})$**
  - maps each n_{jk} to **key = j, value = $(\mathbf{N}, k, n_{jk})$**

- **Reduce function:**
  - for key j, pair each **$(\mathbf{M}, i, m_{ij})$** on its list with each **$(\mathbf{N}, k, n_{jk})$**, and generate: key = (i, k), value = m_{ij} x n_{jk}.

Double-Pass MapReduce for Matrix Multiplication (Optional): The Second Pass

- **Map function:**
  - Pair each key (i,k) with the list of products m_{ij}x n_{jk} for all j:

$$((i,k), [m_{i1}\text{x } n_{1k} , m_{i2}\text{x } n_{2k} , \dots , m_{iN}\text{x } n_{Nk}])$$

- **Reduce function:**
  - For each (i,k), add up all the elements on the list, and generates $\boldsymbol{((i,k),p_{ik})}$:

$$p_{ik} = \sum_j m_{ij} \times n_{jk}$$

Thank You!

Questions?