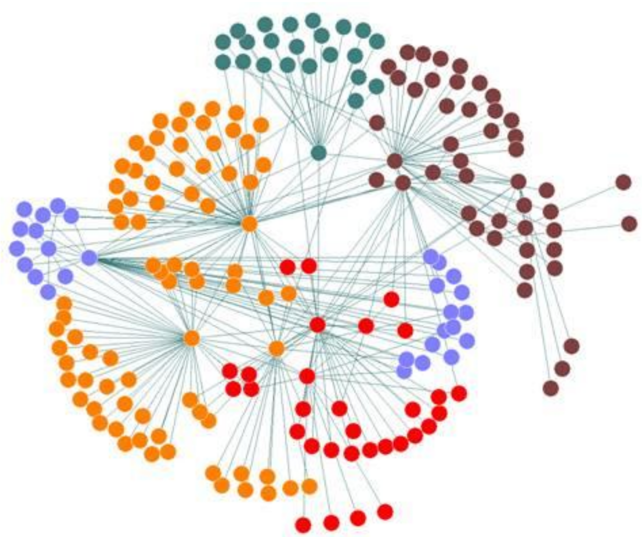
TEL AVIV UNIVERSITY

Algorithms and Applications in Social Networks



2019/2020, Semester B

Slava Novgorodov

Lesson #11

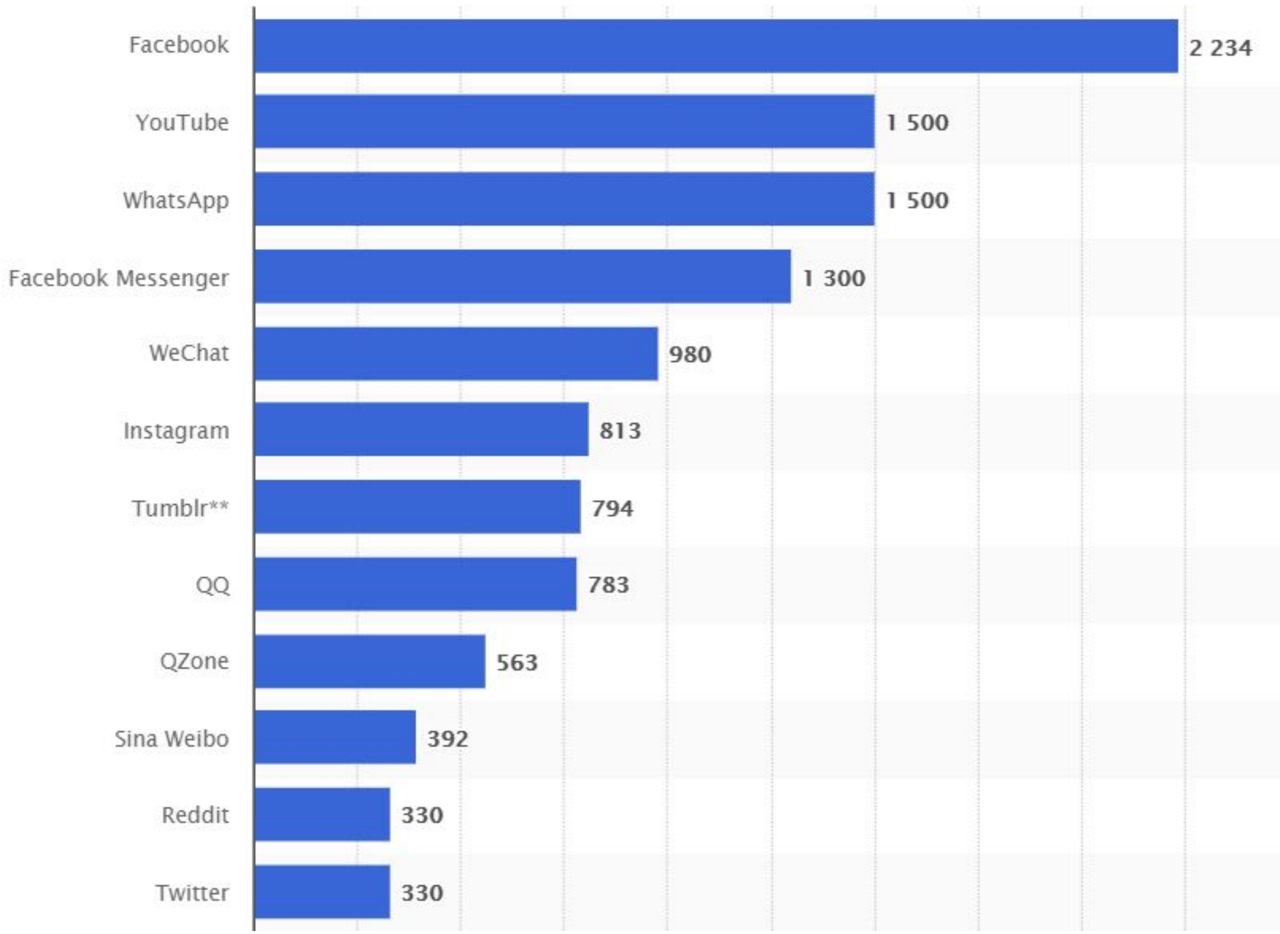
- Dealing with Large Scale Networks
- The Map/Reduce Approach
- Social Network Analysis Examples

Dealing with Large Scale Networks

Large Scale Networks

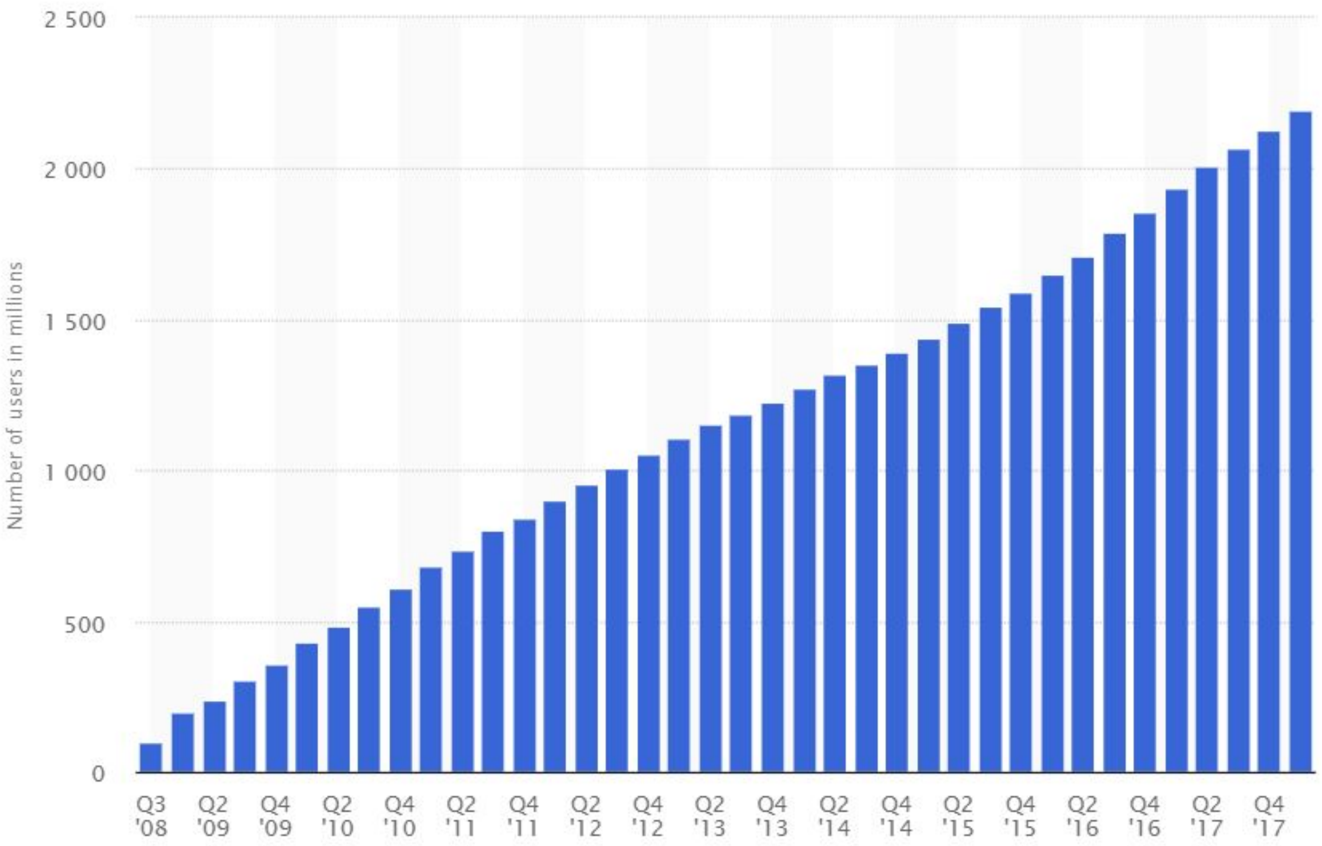
- The real online social networks are huge
- Other “constructed” social networks that involve people are also very big
- Need a scalable solution for analysis

Statistics



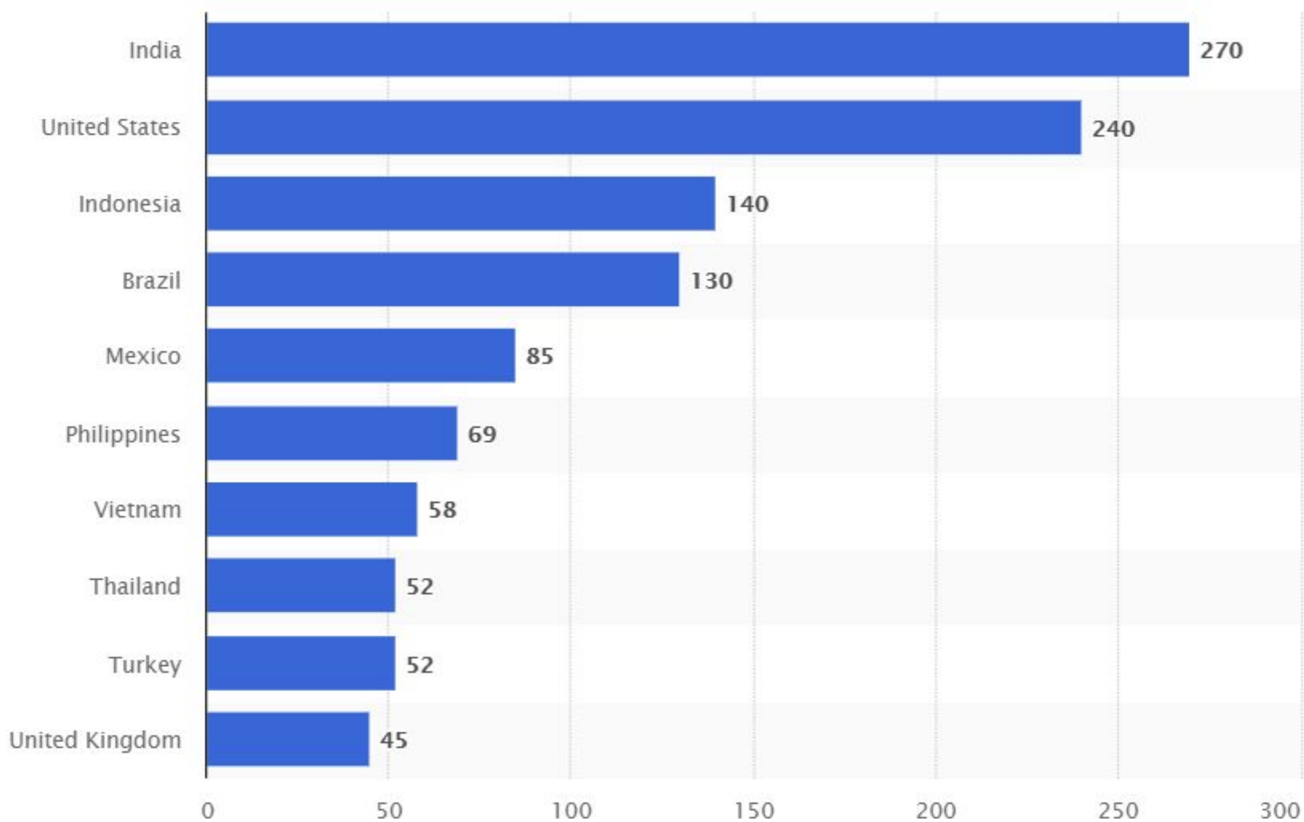
Top social networks – number of active users (in millions) – April 2018

Statistics



Facebook – number of active users (in millions) per quarter

Statistics



Facebook – number of active users (in millions) per country – April 2018

Statistics

Oberlo

How Many People Use Facebook?

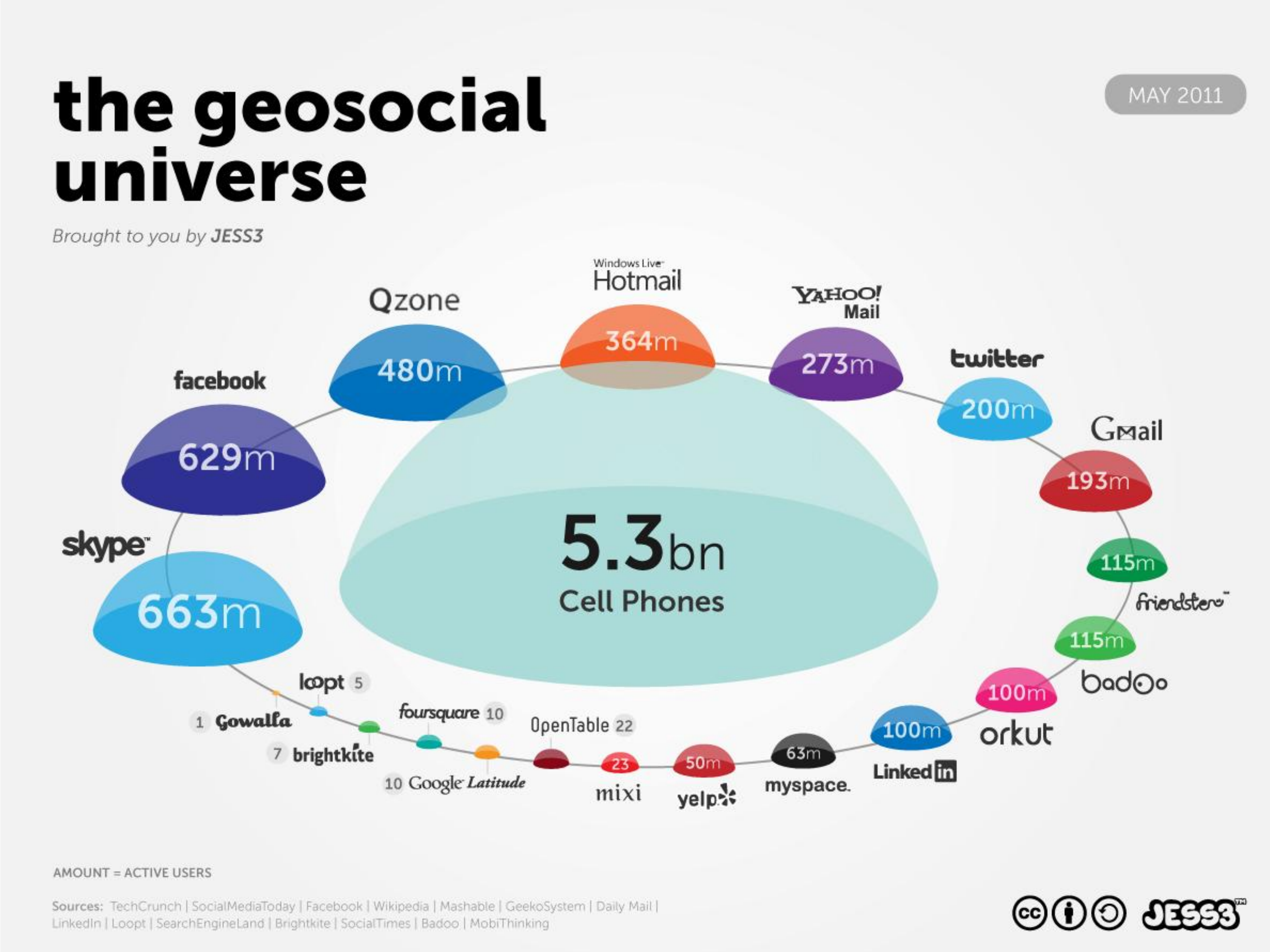
Facebook monthly active users (MAUs) – **MAUs were**

2.60 billion

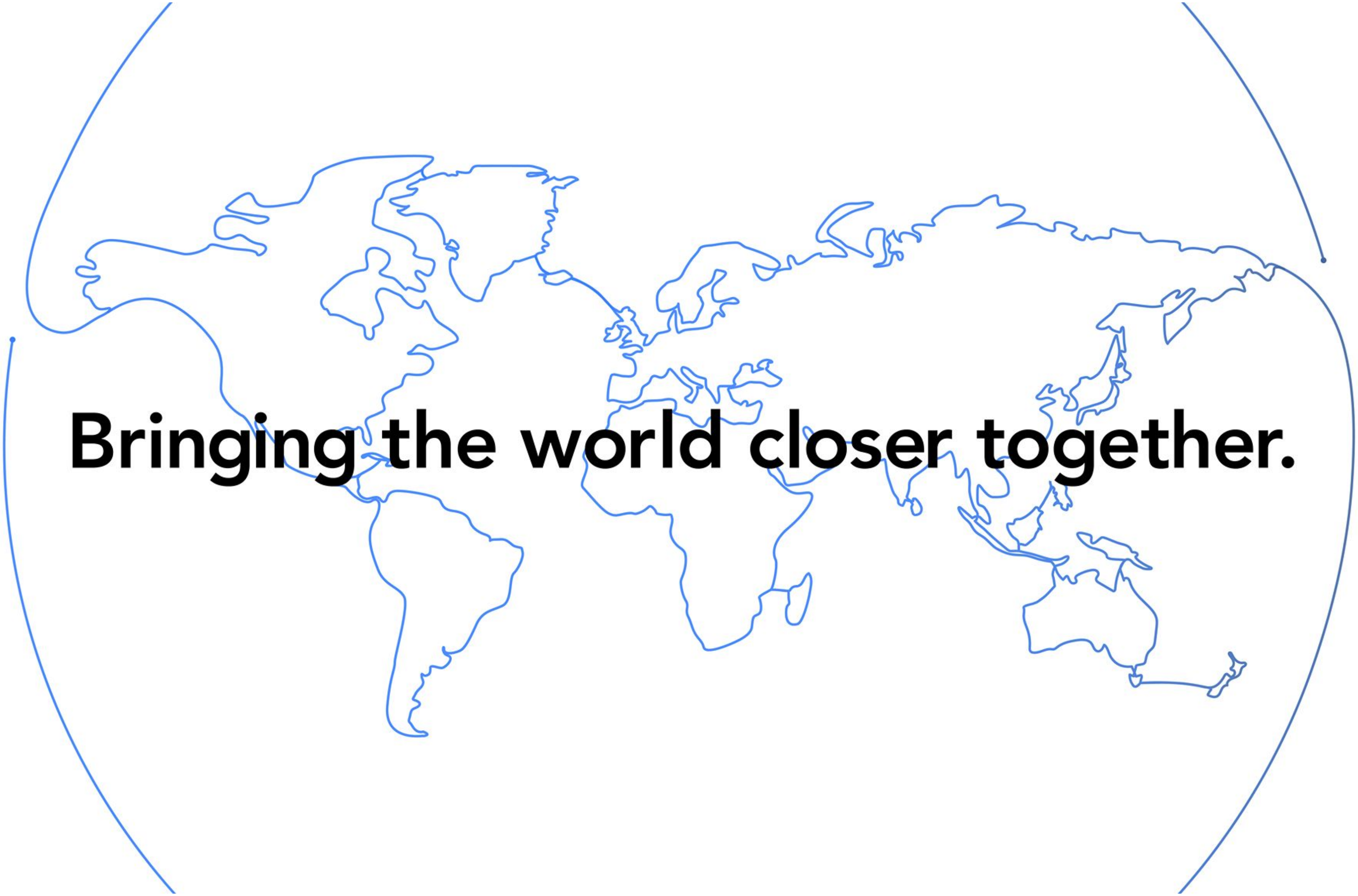
Facebook daily active users (DAUs) – **DAUs were**

1.73 billion

(Facebook, 2020)



May 2011 – size of major social services (in active users)


Bringing the world closer together.

Social Network Analysis Tools

- Small scale network analysis and visualization:

NetworkX

- Pros: has implementation of many of the known algorithms
- Cons: Not so good for large-scale data

Tools for Large-Scale analysis

- Apache Giraph
- GraphLab
- Pegasus
- MapReduce

Tools for Large-Scale analysis

- Apache Giraph
- GraphLab
- Pegasus
- **MapReduce**

The Map/Reduce Approach

Map Reduce

- A programming model for large-scale, parallel and distributed data processing

Map Reduce

- Publically presented by Google in 2004

MapReduce:
Simplified Data Processing on Large Clusters

Jeff Dean, Sanjay Ghemawat
Google, Inc.

OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150
https://research.google.com/archive/mapreduce-osdi04-slides/

Map Reduce

- MapReduce is useful for a wide range of applications:
  - Distributed Sorting
  - Web-graph analysis (PageRank, …)
  - Documents clustering
  - Inverted index construction
  - …

When to use Map Reduce?

- Problems that are ***huge***, but not ***hard***
- Problems that easy to parallelize (easily partitionable and combinable)
- You should only implement Map and Reduce!

Hadoop

- A collection of open-source implementations of parallel, distributed computation
- Started in 2006

- **HDFS** – open source implementation of GFS (Google File Syste)

(Few words about) HDFS

- Great for huge files (TBs…)
- Each file is partitioned to chunks (64MB+)
- Each file is replicated several times

M/R Approach

- Read the data
- **Map**: Extract information from each row
- Shuffle
- **Reduce**: Aggregate, filter, transform…
- Write the results

M/R Model

- Input: Files
- Each line in file: (key, value)

- M/R program:
  - Input: Bag of (input_key, value) pairs
  - Output: Bag of (output_key, value) pairs

Map Phase

- Input: Bag of (input_key, value) pairs
- Output: Bag of (intermidiate_key, value) pairs

- The system applies the map phase in parallel to all (input_key, value) pairs in the input file

Reduce Phase

- Input: Bag of (interm_key, **bag of values**) pairs
- Output: Bag of (output_key, values)

- The system groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- The “Hello, World!” of Map Reduce – **WordCout**

- Given a file with many rows, find how many times each word appears in the whole file

Input:
this is first line
and this is second line
and another line

Output:
this, 2
is, 2
first, 1
line, 3
and, 2
second, 1
another, 1

Example – solution

- The “Hello, World!” of Map Reduce - **WordCout**

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Example – solution

- Map:

```
def mapfn(k, v):
    for w in v.split():
        yield w, 1
```

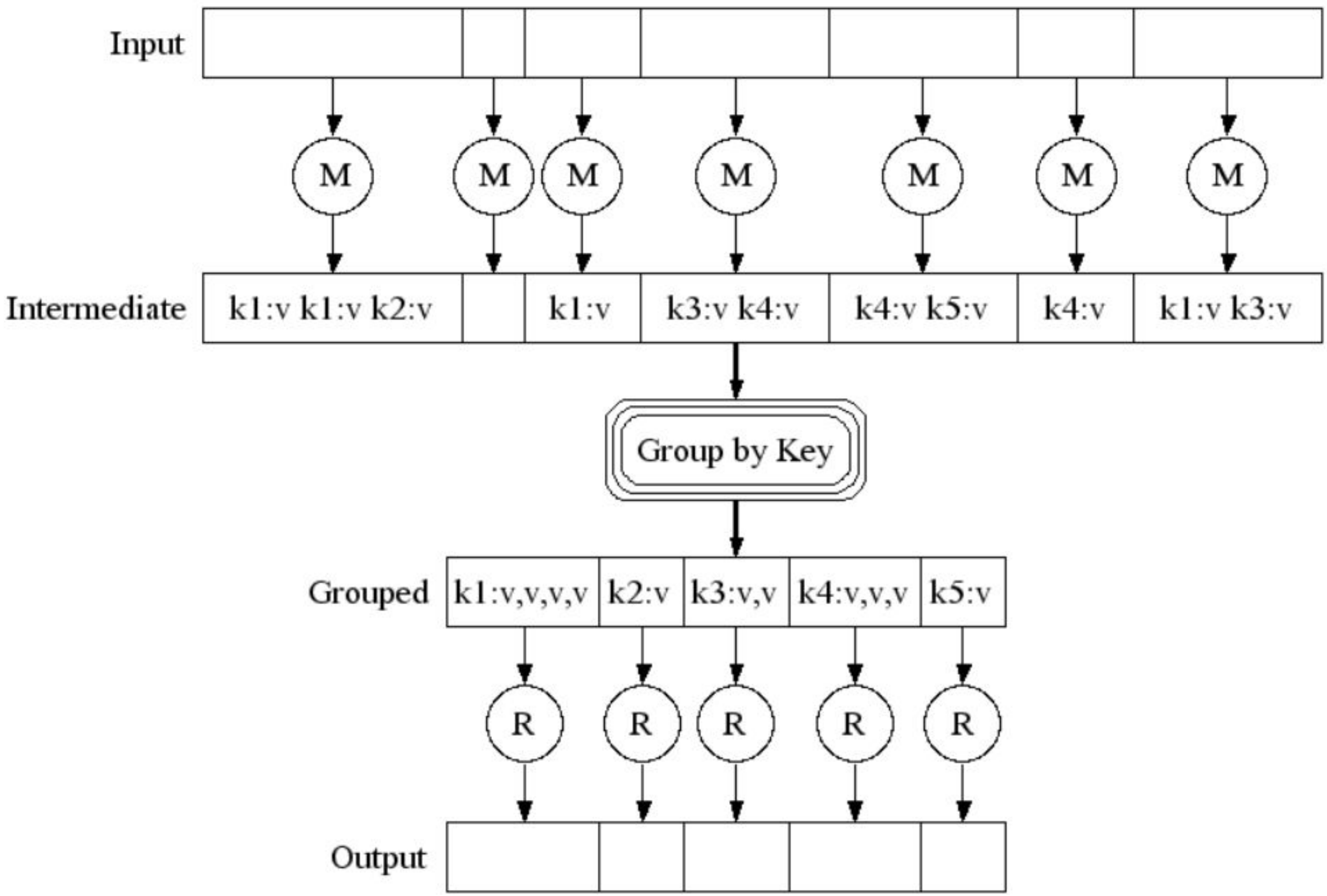
- Reduce:

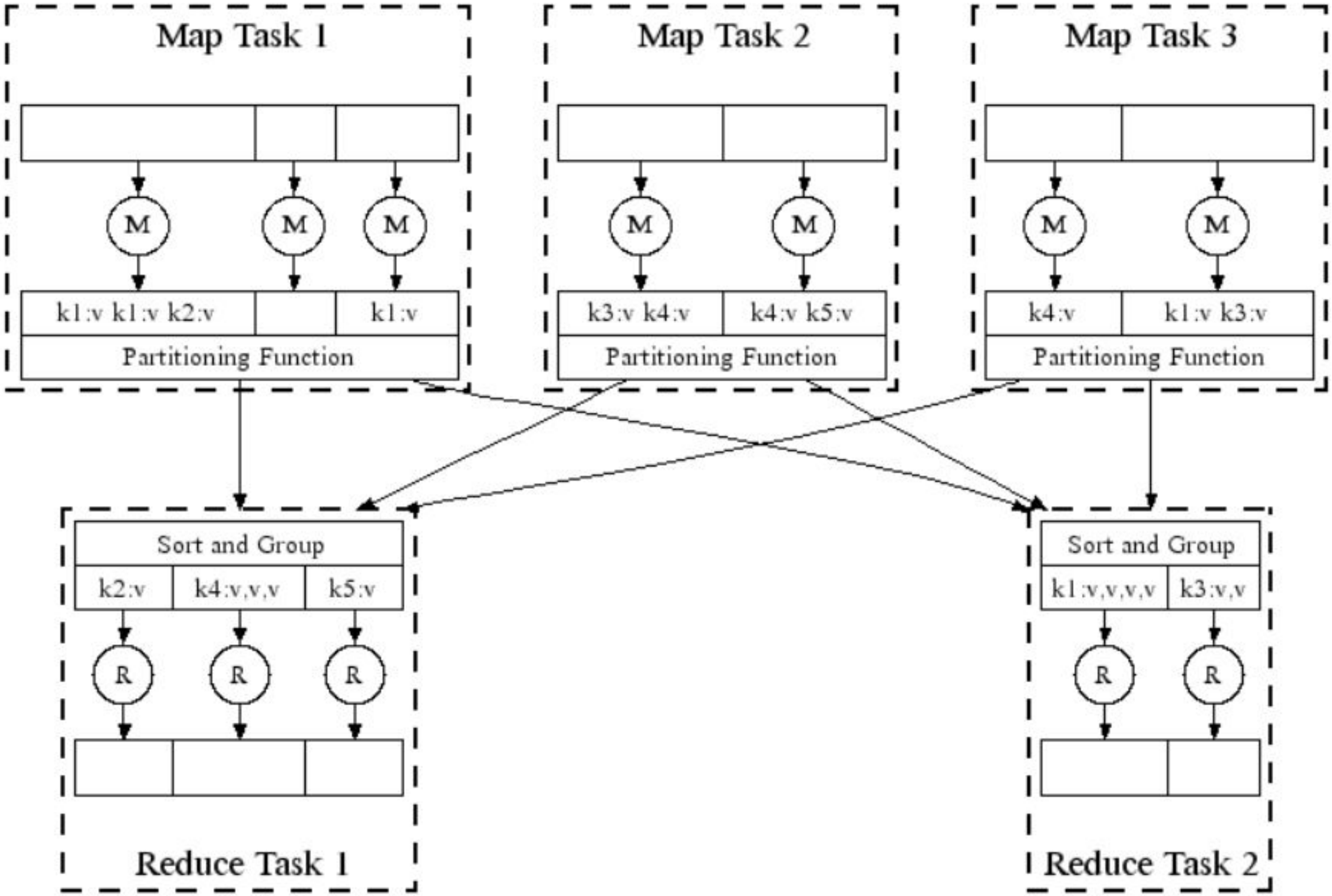
```
def reducefn(k, vs):
    result = sum(vs)
    return result
```

This particular implementation is in Python (as the rest of the lecture).

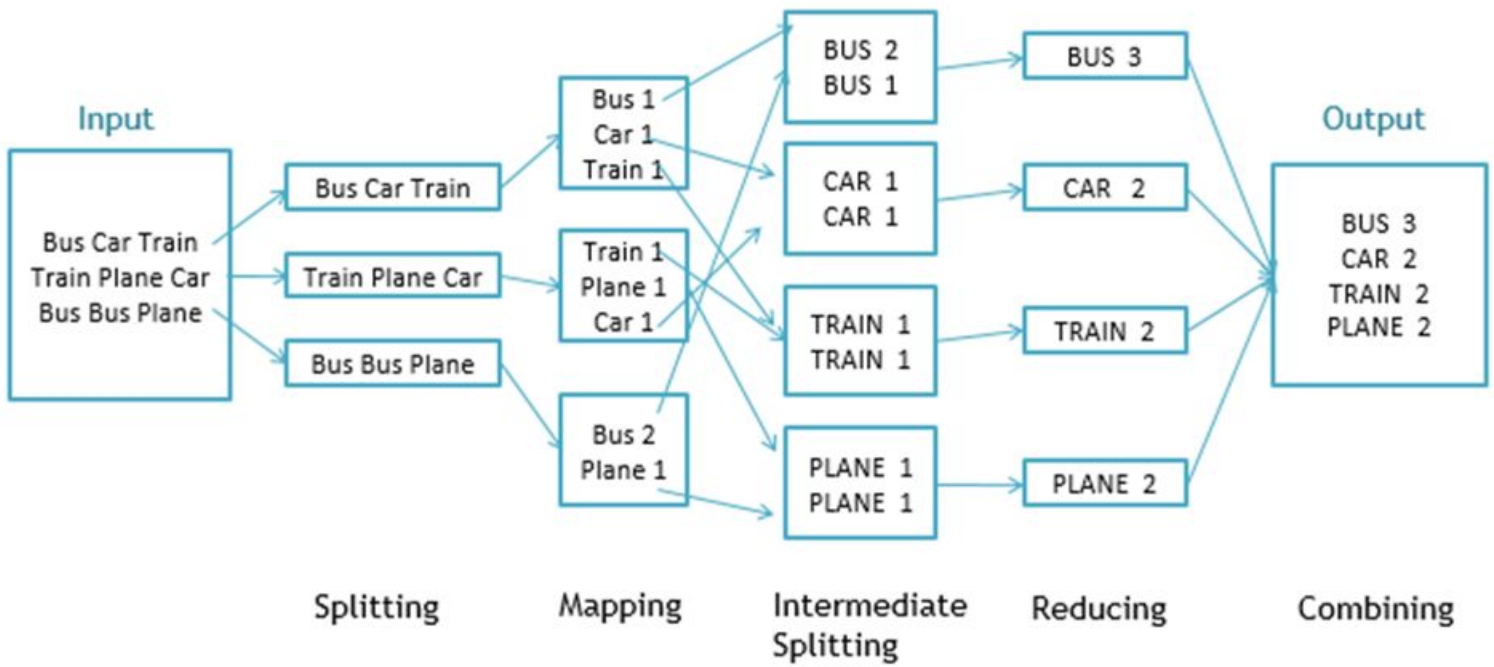
Java, Scala and other languages are also supported.

It's not important to remember the syntax, remember the pseudo-code!

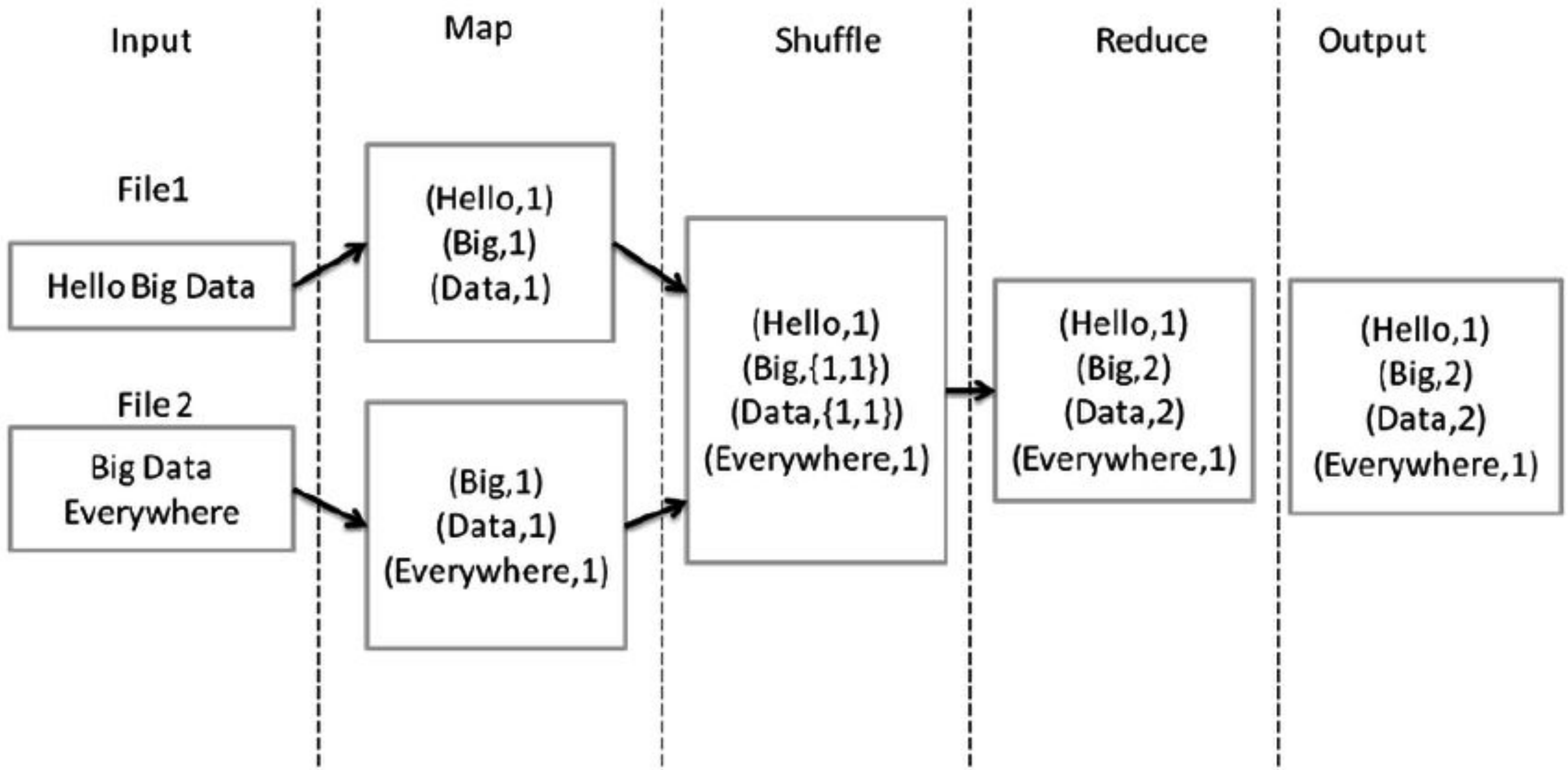

Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output


Map Task 1
M
M
M
k1:v k1:v k2:v
k1:v
Partitioning Function
Map Task 2
M
M
k3:v k4:v
k4:v k5:v
Partitioning Function
Map Task 3
M
M
k4:v
k1:v k3:v
Partitioning Function
Sort and Group
k2:v
k4:v,v,v
k5:v
R
R
R
Reduce Task 1
Sort and Group
k1:v,v,v,v
k3:v,v
R
R
Reduce Task 2

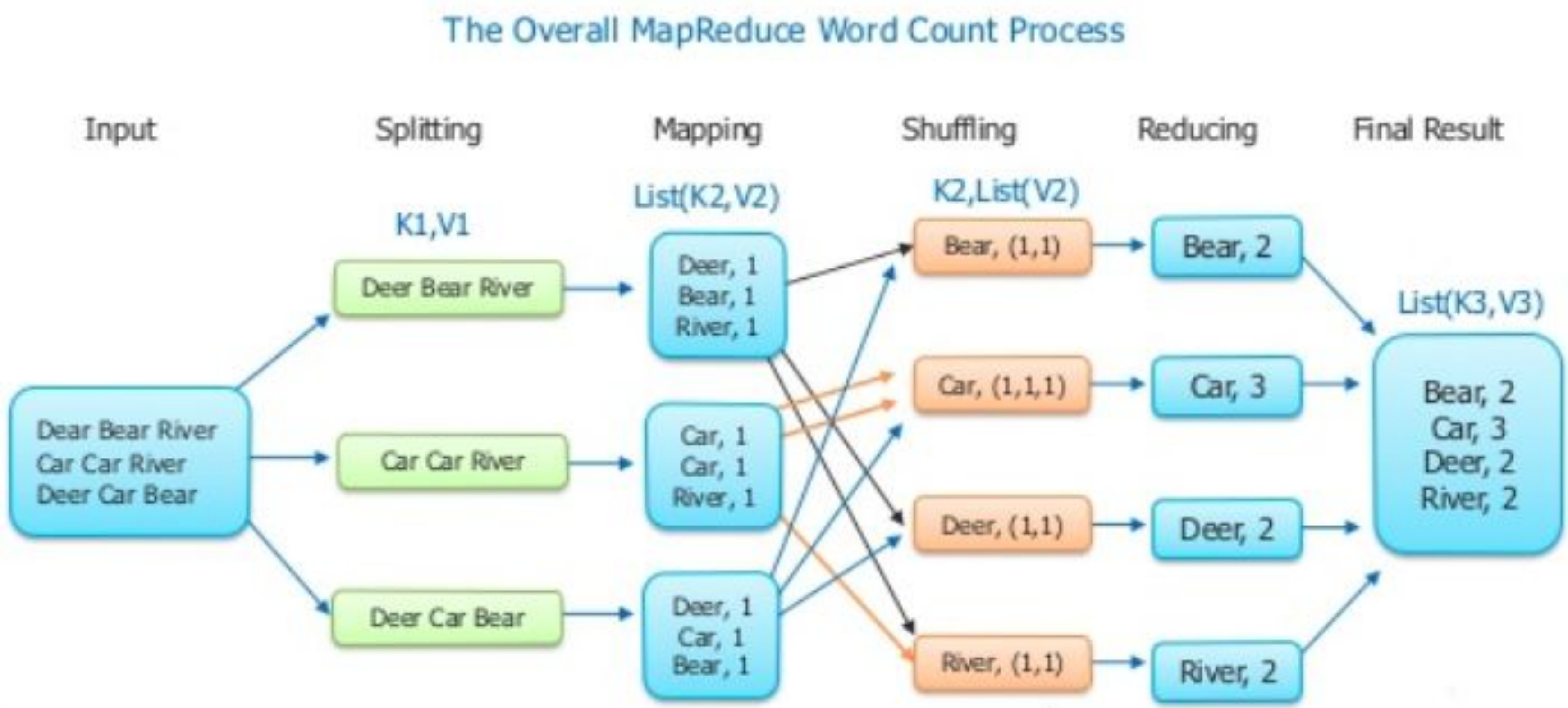
WordCount Flow in M/R



WordCount Flow in M/R



WordCount Flow in M/R



Another WordCount

```
This is a line
Also this
```

Map

```
map("This is a line") =
    this, 1
    is, 1
    a, 1
    line, 1
map("Also this") =
    also, 1
    this, 1
```

Reduce

```
reduce(a, {1}) =
    a, 1
reduce(also, {1}) =
    also, 1
reduce(is, {1}) =
    is, 1
reduce(line, {1}) =
    line, 1
reduce(this, {1, 1}) =
    this, 2
```

Result:

```
a, 1
also, 1
is, 1
line, 1
this, 2
```

Summary

- Map Reduce is a programming model for scalable data processing
- The input is a file, each line is processed separately
- User needs to implement **Map** and **Reduce**

- Technical details left for other courses:
  - Workers vs Tasks, HDFS, fault tolerance, translation other languages to MapReduce, …

Social Network Analysis Examples

Social Networks

- Social network may be huge…
- Need an efficient way to perform computation

- **Solution:** MapReduce

Social Networks

- Representation:
  - Adjacency Matrix vs Neighbors list?

- As Map Reduce takes text files and works line by line, better to have each line as a separate node:

A -> B C D
B -> A C D E
C -> A B D E
D -> A B C E
E -> B C D

Example #1

- **Task:** Find all incoming links
- **Input:**

  A -> B C

  B -> D E

  C -> A E

  D -> A E

  E -> D

- **Output:**

  A -> [‘C', ‘D']

  B -> [‘A’]

  C -> [‘A’]

  E -> [‘B', ‘C', ‘D’]

  D -> [‘B', ‘E']

Example #1 - solution

Map:

```
def mapfn(k, v):
	d = v.split("->")
	pages = set(d[1].strip().split(" "))
	for w in pages:
		yield w, d[0].strip()
```

Reduce:

```
def reducefn(k, vs):
	return vs
```

Example #2

- **Task:** Find all mutual friends of all **pairs of users**
- **Input:**

  A -> B C D
  B -> A C D E
  C -> A B D E
  D -> A B C E
  E -> B C D

- **Output:**

  ('A', 'B') -> {'C', 'D'}
  ('A', 'C') -> {'D', 'B'}
  ('A', 'D') -> {'B', 'C'}
  ('A', 'E') -> {'B', 'C', 'D'}
  ('B', 'C') -> {'A', 'D', 'E'}
  ('B', 'D') -> {'A', 'C', 'E'}
  ('B', 'E') -> {'C', 'D'}
  ('C', 'D') -> {'A', 'B', 'E'}
  ('C', 'E') -> {'B', 'D'}
  ('D', 'E') -> {'B', 'C'}

Example #2 - solution

Map:

```
def mapfn(k, v):
        d = v.split("->")
        friends = set(d[1].strip().split(" "))
        for f1 in friends:
            for f2 in friends:
                if f1 < f2:
                    key = d[0].strip()
                    yield (f1, f2), key
```

Reduce:

```
def reducefn(k, vs):
        return vs
```

Example #3

- **Task:** Find all mutual friends of all **current friends**
- **Input:**

A -> B C D
B -> A C D E
C -> A B D E
D -> A B C E
E -> B C D

- **Output:**

('A', 'D') -> {'B', 'C'}
('A', 'C') -> {'D', 'B'}
('A', 'B') -> {'D', 'C'}
('B', 'C') -> {'D', 'A', 'E'}
('B', 'E') -> {'D', 'C'}
('B', 'D') -> {'A', 'C', 'E'}
('C', 'D') -> {'A', 'B', 'E'}
('C', 'E') -> {'D', 'B'}
('D', 'E') -> {'B', 'C'}

Example #3 - solution

Map:

```
def mapfn(k, v):
	d = v.split("->")
	friends = set(d[1].strip().split(" "))
	for w in friends:
		first = d[0].strip()
		second = w
		if first > second:
			temp = first
			first = second
			second = temp
		yield (first, second), friends
```

Reduce:

```
def reducefn(k, vs):
	ret = vs[0]
	for s in vs:
		ret = ret.intersection(s)
	return ret
```

Example #4

- **Task:** Find all unique triangles in the network
- **Input:**

  A -> B C F

  B -> A

  C -> A D

  D -> C E F

  E -> D F

  F -> A D E

- **Output:**

  (D, E, F)

Example #4 - solution

- **Task:** Find all unique triangles in the network
- **Input:**

  A -> B C F
  B -> A
  C -> A D
  D -> C E F
  E -> D F
  F -> A D E

Idea: Generate triangles and count (if equals to 3)

- **Output:**

  (D, E, F)

Formalize at home

More Riddles

Riddle #1

There are 101 cities, every city connected to other 100 cities, 50 with in-bound connection and 50 with outbound connection

Prove that from every city to another you can go using maximum 2 edges

Riddle #1 - hint

There are 101 cities, every city connected to other 100 cities, 50 with in-bound connection and 50 with outbound connection

Prove that from every city to another you can go using maximum 2 edges

Hint: go in the negative direction…

Riddle #1 - solution

There are 101 cities, every city connected to other 100 cities, 50 with in-bound connection and 50 with outbound connection

Prove that from every city to another you can go using maximum 2 edges

Solution - In class

Thank you!

Questions?