CS6423: Scalable Computing

Gregory Provan

Spring 2020

Lecture 2: Introduction to MapReduce

Plan for today

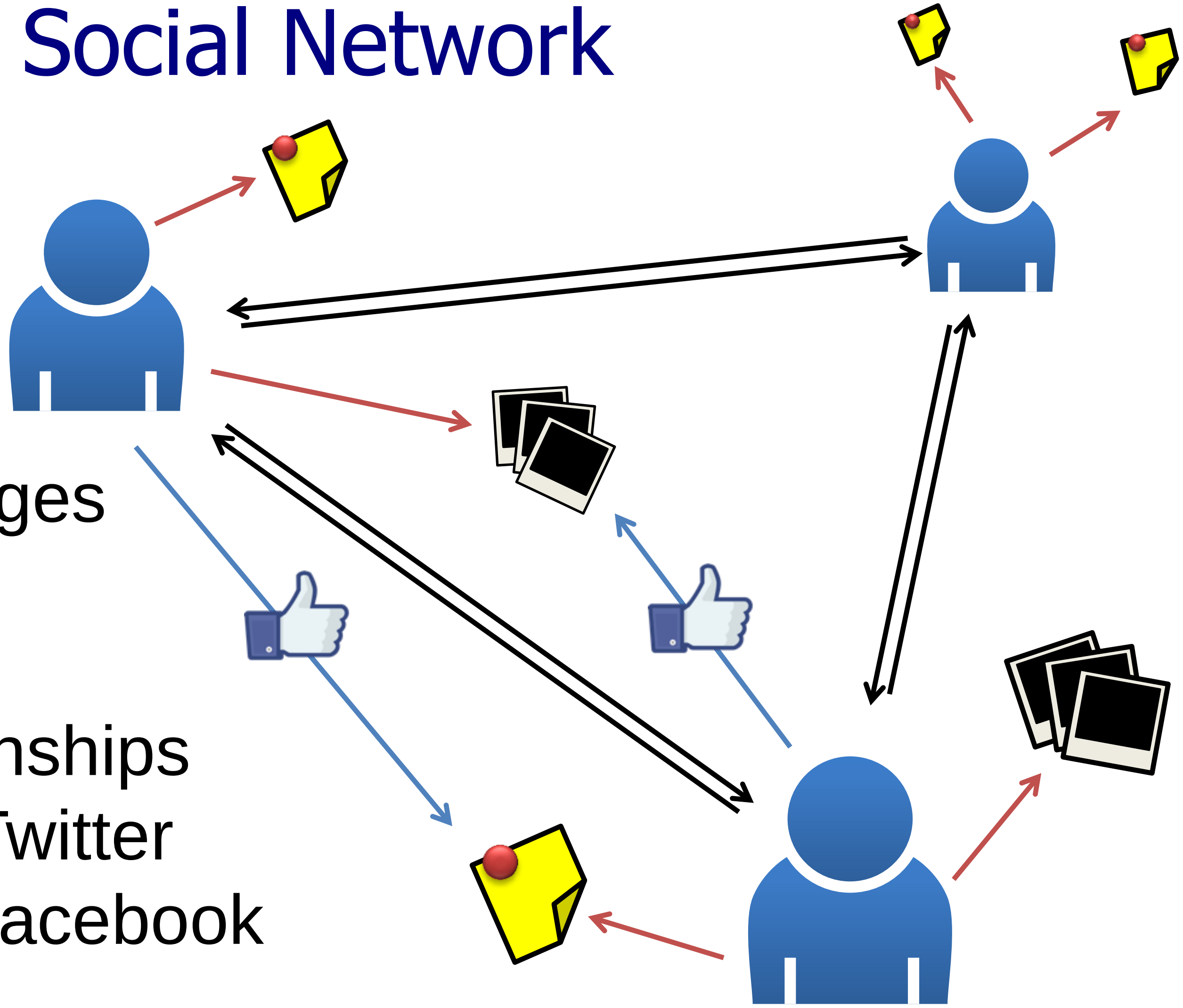
- Motivation NEXT
- MapReduce architecture
  - Data flow
  - Execution flow
  - Fault tolerance etc.

Modern Computing Needs

- Web-search requires traversing enormous graph
- Huge amounts of data to sift through
- Need new computing paradigm

- Examples
  - Social networks (Facebook)
  - Recommender systems (Amazon)

Social Network



Vertices
- Users
- Posts / Images

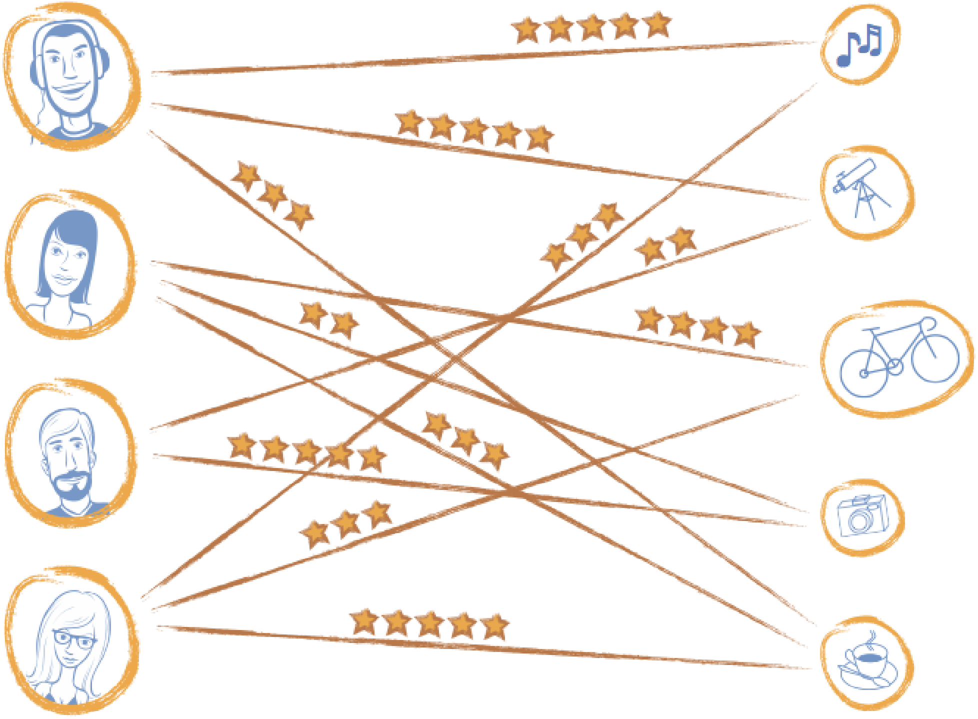
Edges
- Social Relationships
  - *Directed:* Twitter
  - *Undirected:* Facebook
- Likes

User - Item Graphs
(Recommender Systems)

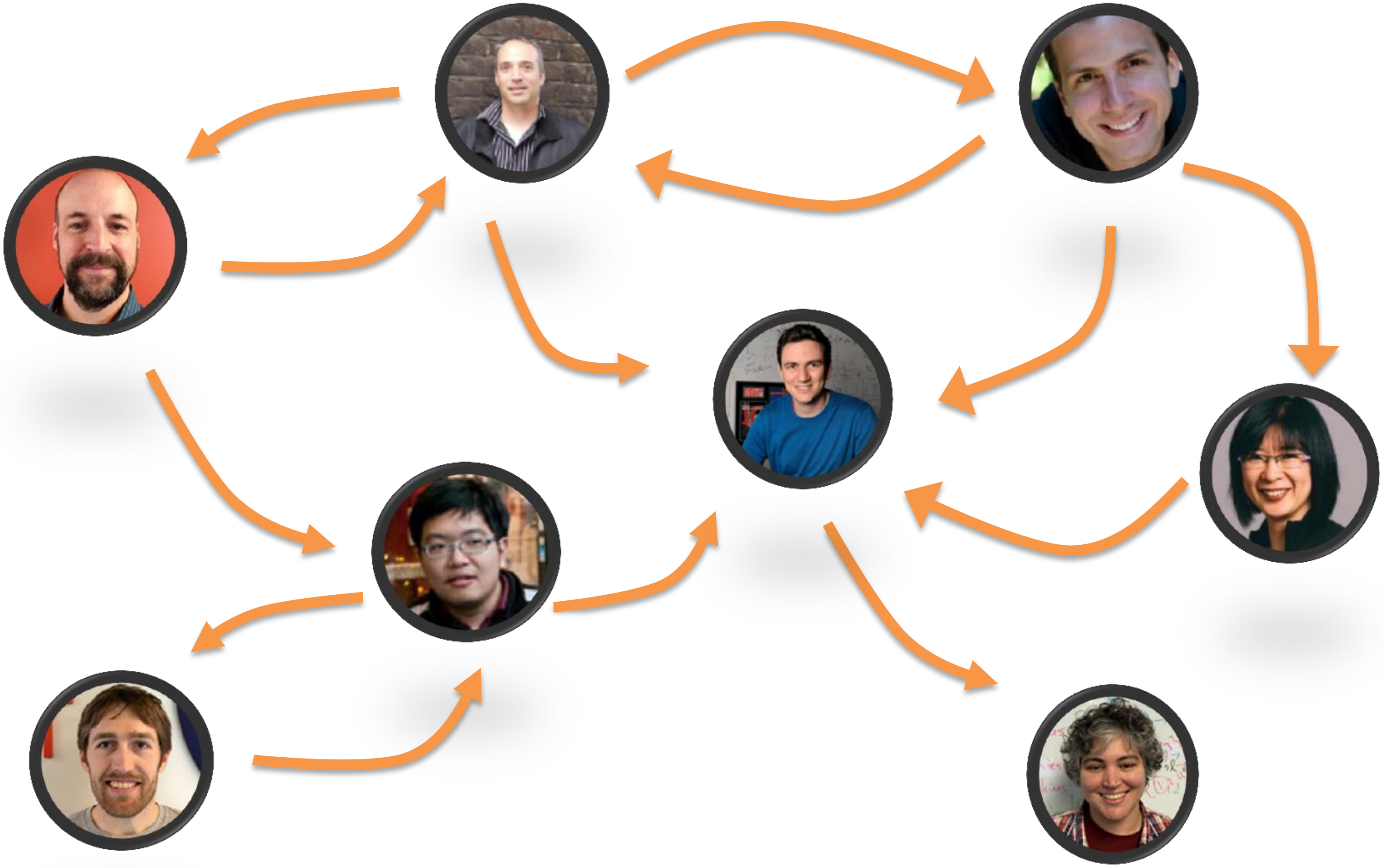
Bipartite Graphs

Vertices: *Users and Items*

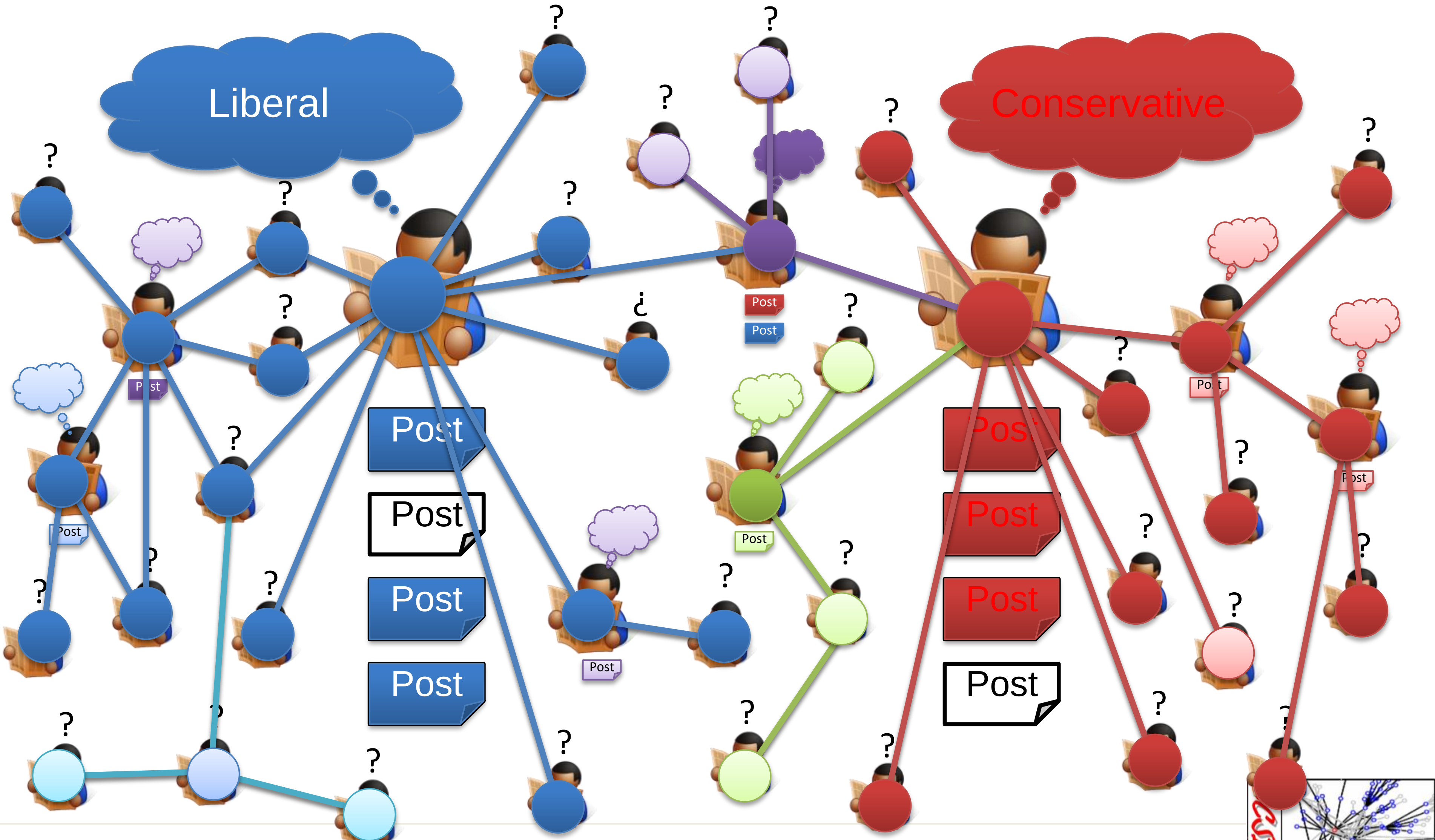
Edges: *Ratings*



Identifying Leaders

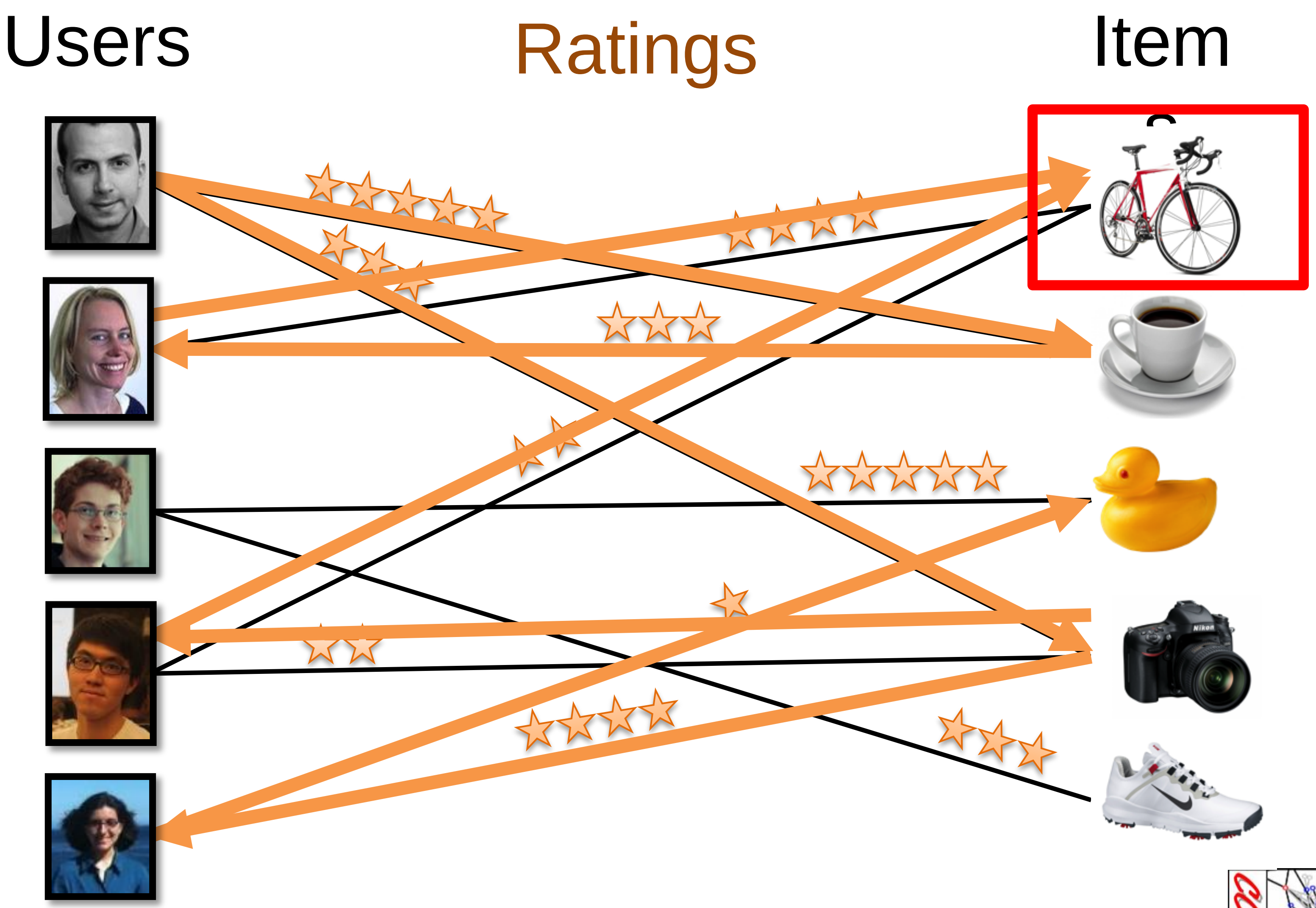


Predicting Behavior



Cork Complex Systems Lab

Recommending Products



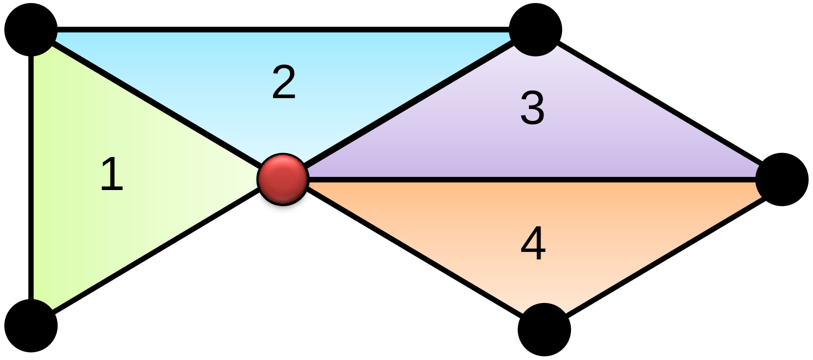
Users

Ratings

Items

Finding Communities

- Count triangles passing through each vertex:
-

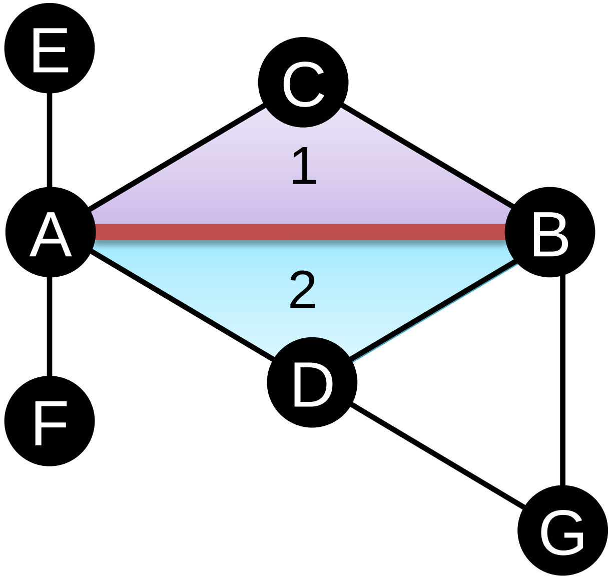


- Measure “cohesiveness” of local community

$$\text{ClusterCoeff[i]} = \frac{2 * \#\text{Triangles[i]}}{\text{Deg[i]} * (\text{Deg[i]} - 1)}$$

Counting Triangles

- Count triangles passing through each vertex by counting triangles on each edge:

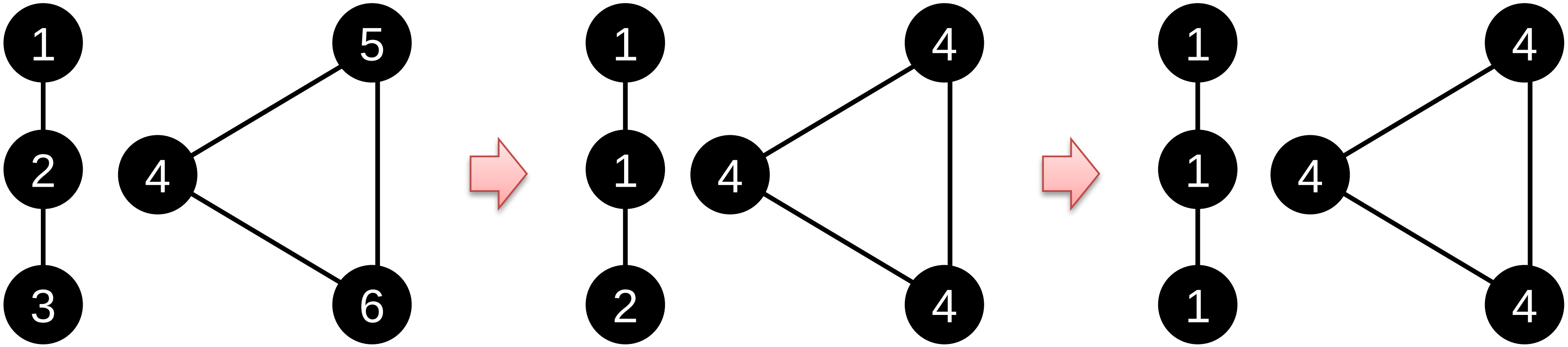


$$\{B, C, D, E, F\} \cap \{A, C, D, G\} = \{C, D\}$$

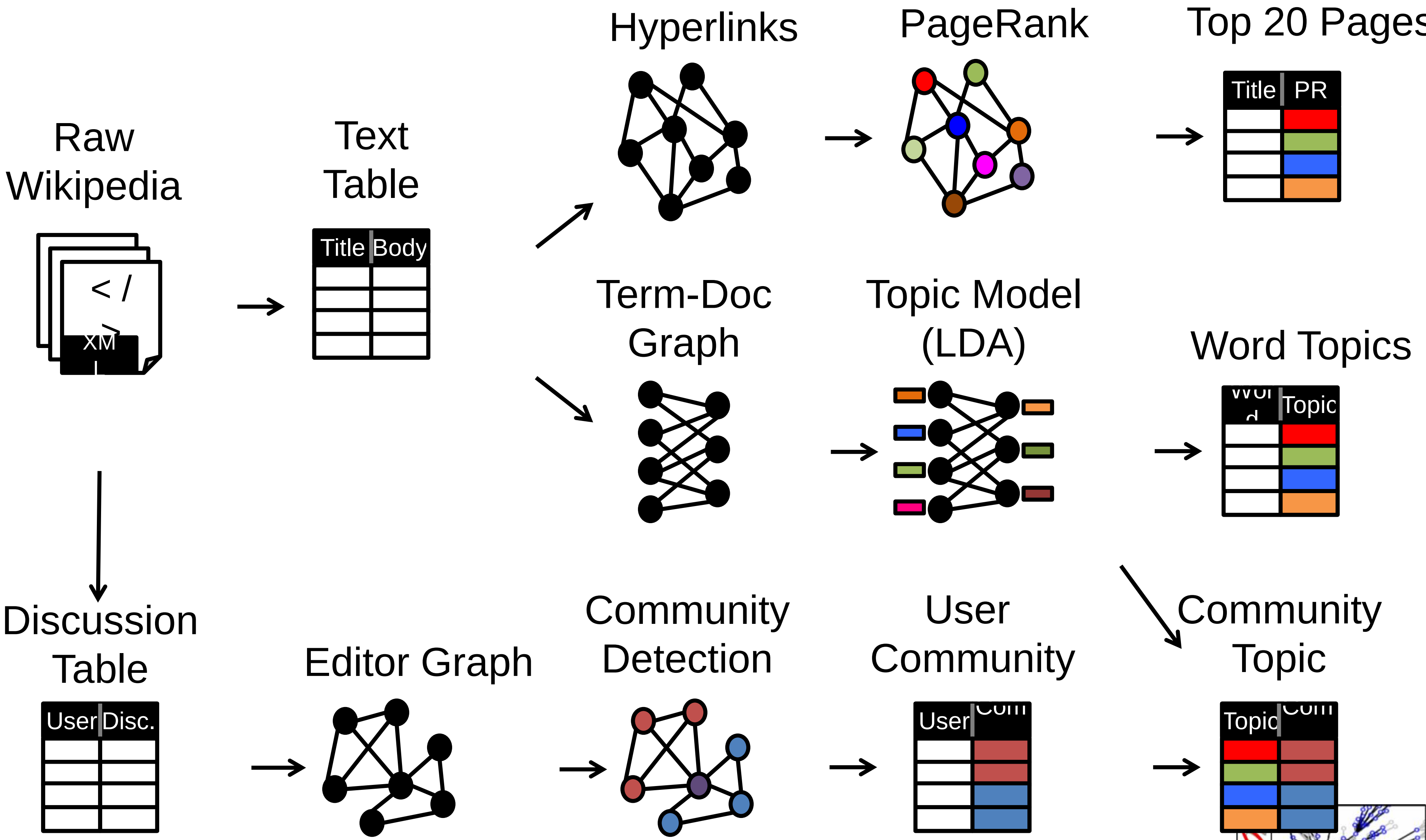
Connected Components

- Every vertex starts out with a unique component id (typically it's vertex id):

$$CC[i] = \arg \min_{\{i,j\} \in E} CC[j]$$



Putting it All Together



Raw Wikipedia

< / > XML

Text Table

Title | Body

Hyperlinks

PageRank

Top 20 Pages

Title | PR

Term-Doc Graph

Topic Model (LDA)

Word Topics

Word | Topic

Discussion Table

User | Disc.

Editor Graph

Community Detection

User Community

User | Com

Community Topic

Topic | Com

Cork Complex Systems Lab

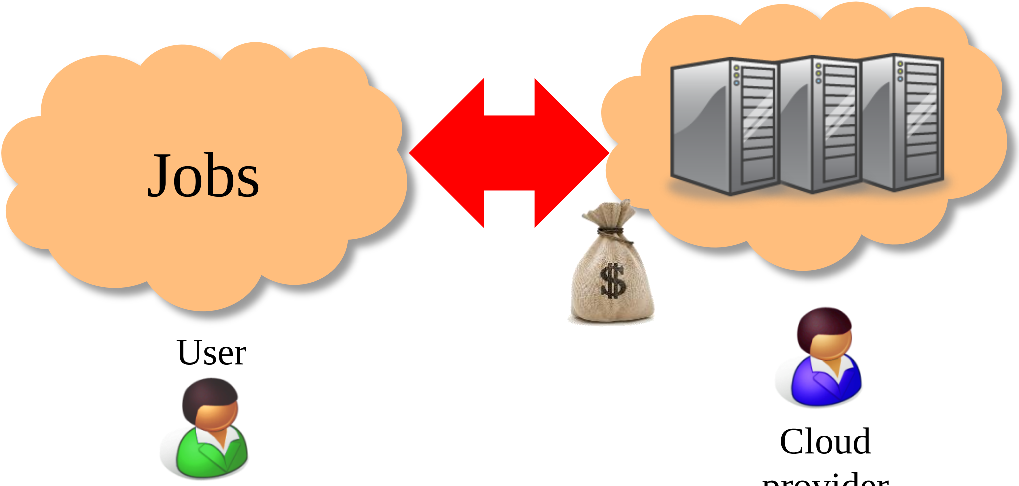
Many Other Graph Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Community Detection
  - Triangle-Counting
  - K-core Decomposition
  - K-Truss
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring
- Classification
  - Neural Networks

Topic 1: Modern Distributed Algorithms

- What is the cloud computing paradigm
- How to characterise its behaviour
- Cloud computing software
  - MapReduce

Cloud Computing Concept



Jobs

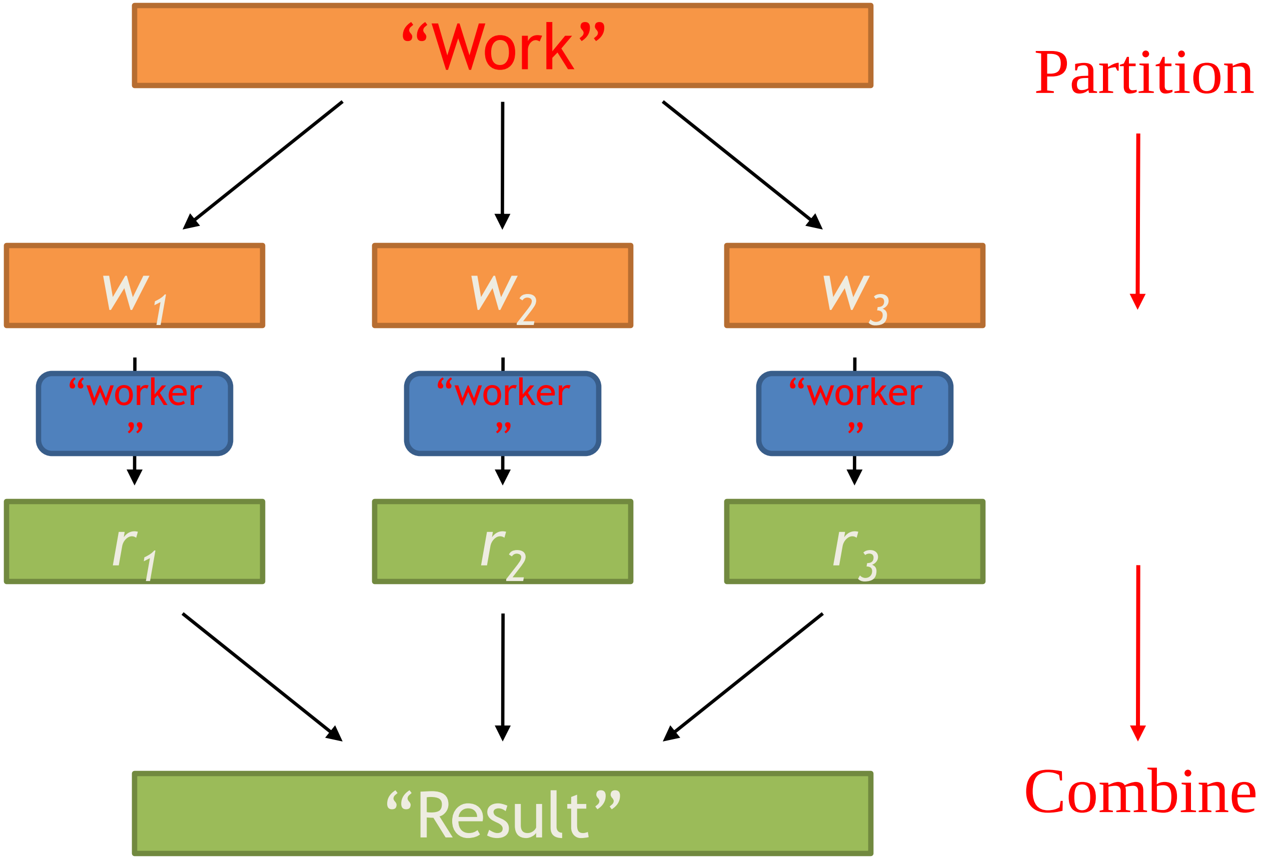
User

Cloud provider

New Paradigm: MapReduce

- Employ multiple CPUS
  - Parallel processing
- New programming paradigm
  - Based on functional programming
- Two-phase inference
  - Map
  - Reduce

Cloud Algorithm: MapReduce



"Work"

Partition

w_1 w_2 w_3

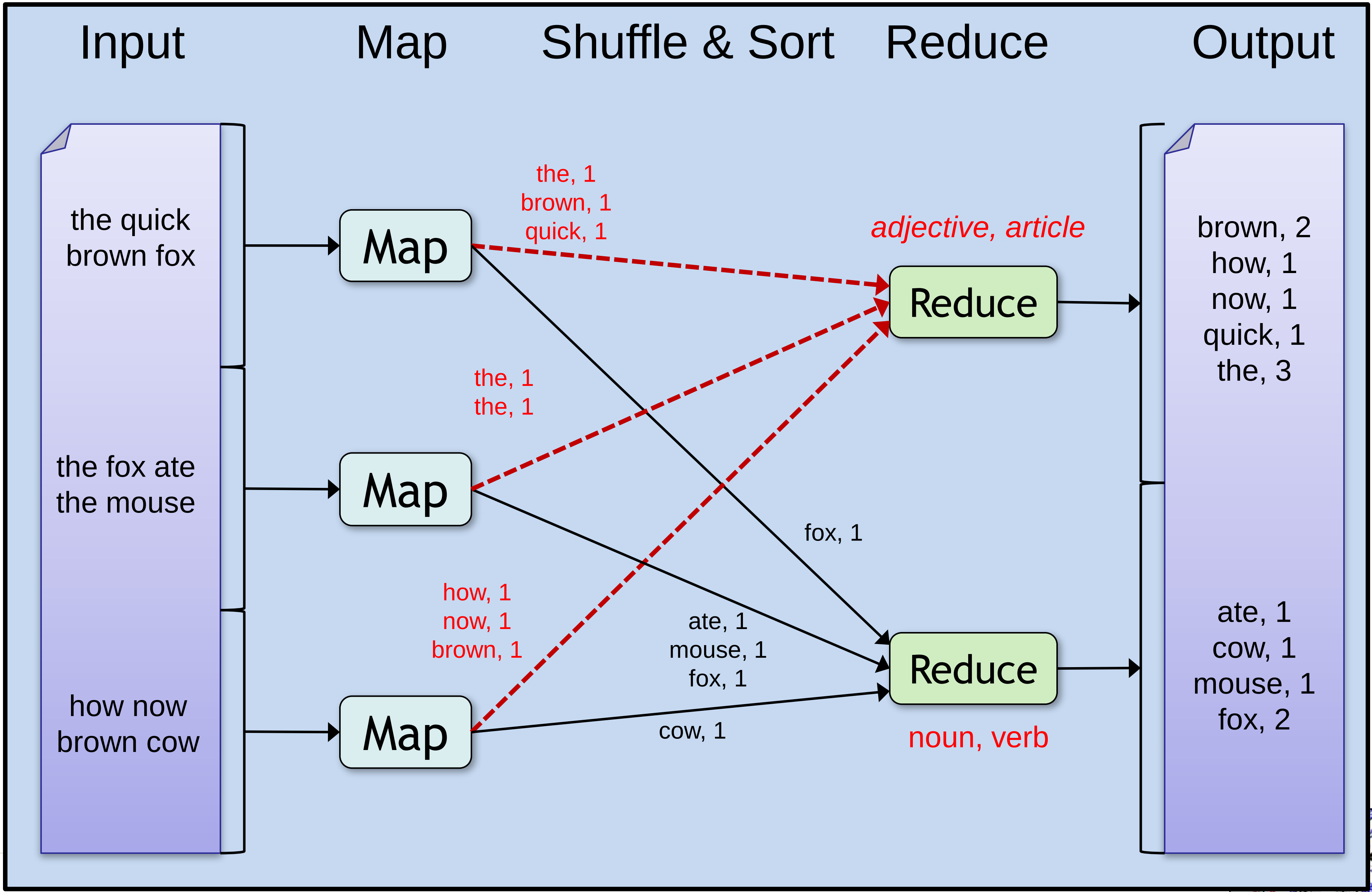
"worker" "worker" "worker"

r_1 r_2 r_3

Combine

"Result"

Word Count Example



Input

Map

Shuffle & Sort

Reduce

Output

the quick
brown fox

the fox ate
the mouse

how now
brown cow

Map

Map

Map

the, 1
brown, 1
quick, 1

the, 1
the, 1

how, 1
now, 1
brown, 1

fox, 1

ate, 1
mouse, 1
fox, 1

cow, 1

adjective, article

Reduce

Reduce

noun, verb

brown, 2
how, 1
now, 1
quick, 1
the, 3

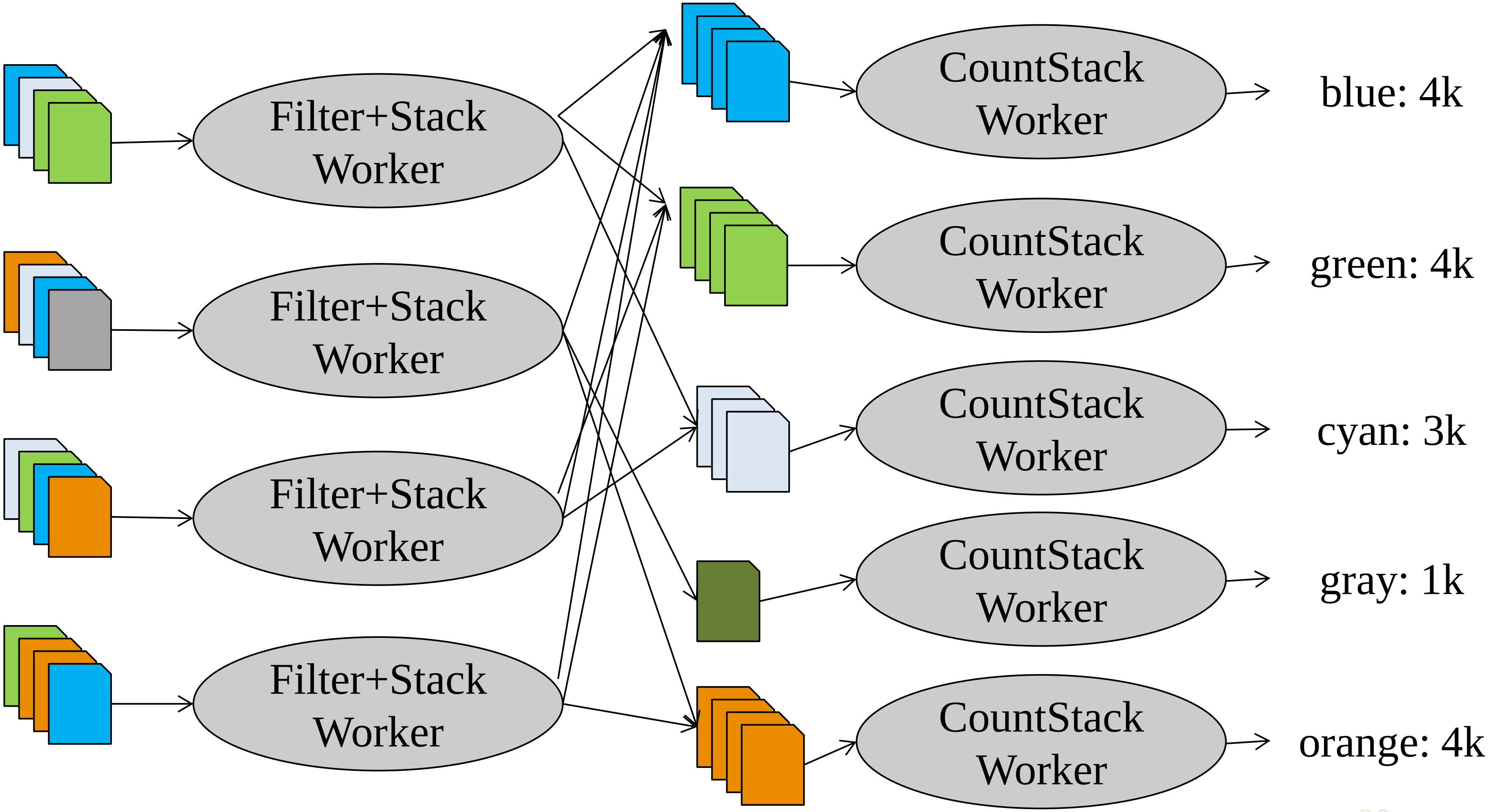
ate, 1
cow, 1
mouse, 1
fox, 2

Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

What is the common theme of all of these problems?

Abstracting digital data flows



Filter+Stack Worker

Filter+Stack Worker

Filter+Stack Worker

Filter+Stack Worker

CountStack Worker

CountStack Worker

CountStack Worker

CountStack Worker

CountStack Worker

blue: 4k

green: 4k

cyan: 3k

gray: 1k

orange: 4k

Abstracting once more

- There are two kinds of workers:
  - Those that take input data items and produce output items for the "stacks"
  - Those that take the stacks and aggregate the results to produce outputs on a per-stack basis

- We'll call these:
  - map: takes (item_key, value), produces one or more (stack_key, value') pairs
  - reduce: takes (stack_key, {set of value'}), produces one or more output results – typically (stack_key, agg_value)

Why MapReduce?

- Scenario:
  - You have a huge amount of data, e.g., all the Google searches of the last three years
  - You would like to perform a computation on the data, e.g., find out which search terms were the most popular
- How would you do it?
- Parallel programming
  - The computation isn't necessarily difficult, but parallelizing and distributing it, as well as handling faults, is challenging
- Idea: A programming language!
  - Write a simple program to express the (simple) computation, and let the language runtime do all the hard work

Plan for today

- Motivation ✓
- MapReduce architecture
  - Data flow
  - Execution flow
  - Fault tolerance etc.

NEXT

What is MapReduce?

- A famous distributed programming model
- In many circles, considered *the* key building block for much of Google's data analysis
  - A programming language built on it: Sawzall, http://labs.google.com/papers/sawzall.html
  - *… Sawzall has become one of the most widely used programming languages at Google. … [O]n one dedicated Workqueue cluster with 1500 Xeon CPUs, there were 32,580 Sawzall jobs launched, using an average of 220 machines each. While running those jobs, 18,636 failures occurred (application failure, network outage, system crash, etc.) that triggered rerunning some portion of the job. The jobs read a total of 3.2x10^{15} bytes of data (2.8PB) and wrote 9.9x10^{12} bytes (9.3TB).*
  - Other similar languages: Yahoo's Pig Latin and Pig; Microsoft's Dryad
- Cloned in open source: Hadoop, http://hadoop.apache.org/

The MapReduce programming model

- Simple distributed functional programming primitives
- Modeled after Lisp primitives:
  - `map` (apply function to all items in a collection) and
  - `reduce` (apply function to set of items with a common key)
- We start with:
  - A user-defined function to be applied to all data,
    `map`: (key,value) → (key, value)
  - Another user-specified operation
    `reduce`: (key, {set of values}) → result
  - A set of *n* nodes, each with data
- All nodes run `map` on all of their data, producing new data with keys
  - This data is collected by key, then shuffled, and finally `reduced`
  - Dataflow is through temp files on GFS

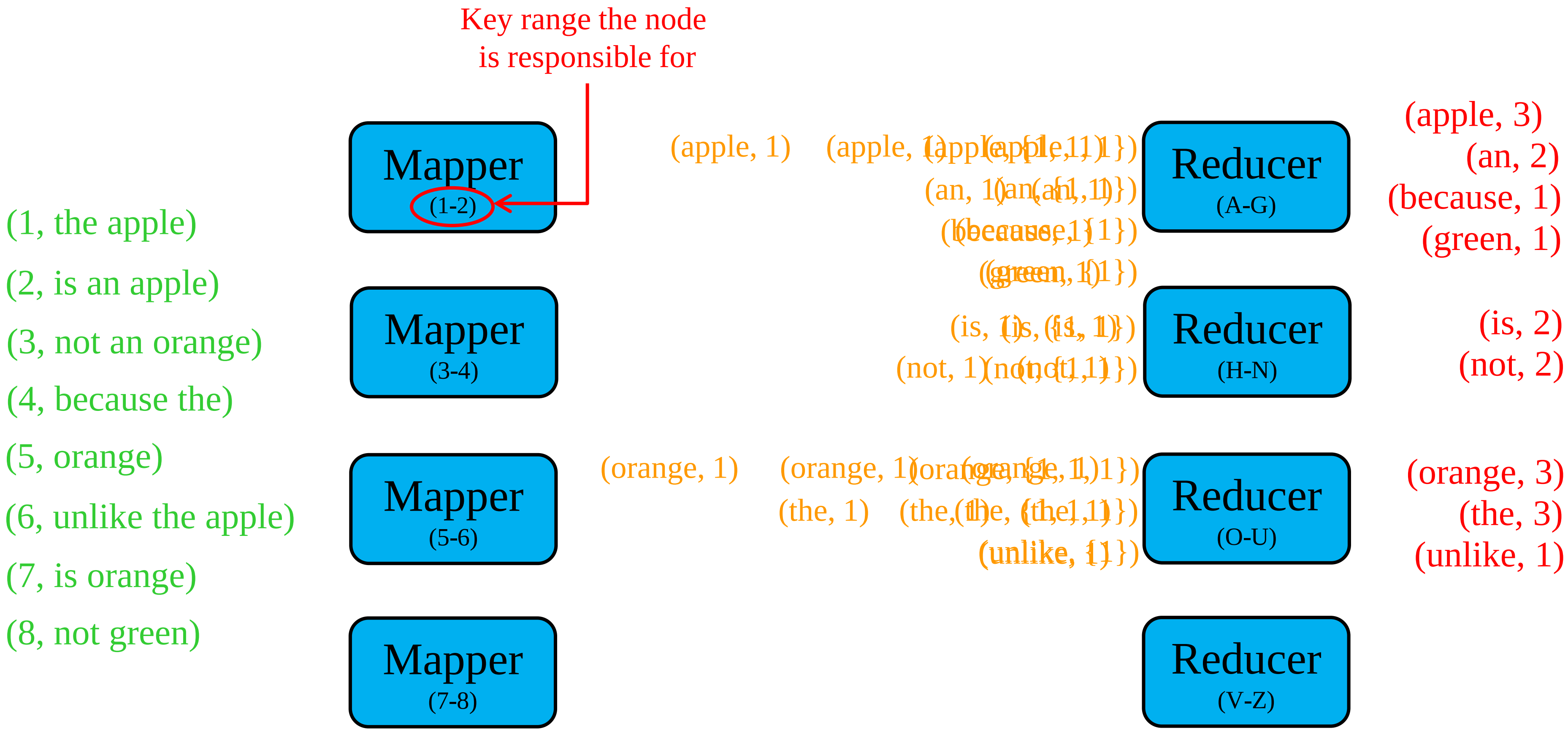
Simple example: Word count

```
map(String key, String value) {
  // key: document name, line no
  // value: contents of line
  for each word w in value:
    emit(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(key, result)
}
```

- Goal: Given a set of documents, count how often each word occurs
  - Input: Key-value pairs (document:lineNumber, text)
  - Output: Key-value pairs (word, #occurrences)
  - What should be the intermediate key-value pairs?

Simple example: Word count



Key range the node is responsible for

(1, the apple)

(2, is an apple)

(3, not an orange)

(4, because the)

(5, orange)

(6, unlike the apple)

(7, is orange)

(8, not green)

Mapper (1-2)

Mapper (3-4)

Mapper (5-6)

Mapper (7-8)

(apple, 1) (apple, 1) (apple, 1) (apple, {1, 1, 1})

(an, 1) (an, 1) (an, {1, 1})

(because, 1) (because, {1})

(green, 1) (green, {1})

(is, 1) (is, 1) (is, {1, 1})

(not, 1) (not, 1) (not, {1, 1})

(orange, 1) (orange, 1) (orange, 1) (orange, {1, 1, 1})

(the, 1) (the, 1) (the, 1) (the, {1, 1, 1})

(unlike, 1) (unlike, {1})

Reducer (A-G)

Reducer (H-N)

Reducer (O-U)

Reducer (V-Z)

(apple, 3)
(an, 2)
(because, 1)
(green, 1)

(is, 2)
(not, 2)

(orange, 3)
(the, 3)
(unlike, 1)

1 Each mapper receives some of the KV-pairs as input

2 The mappers process the KV-pairs one by one

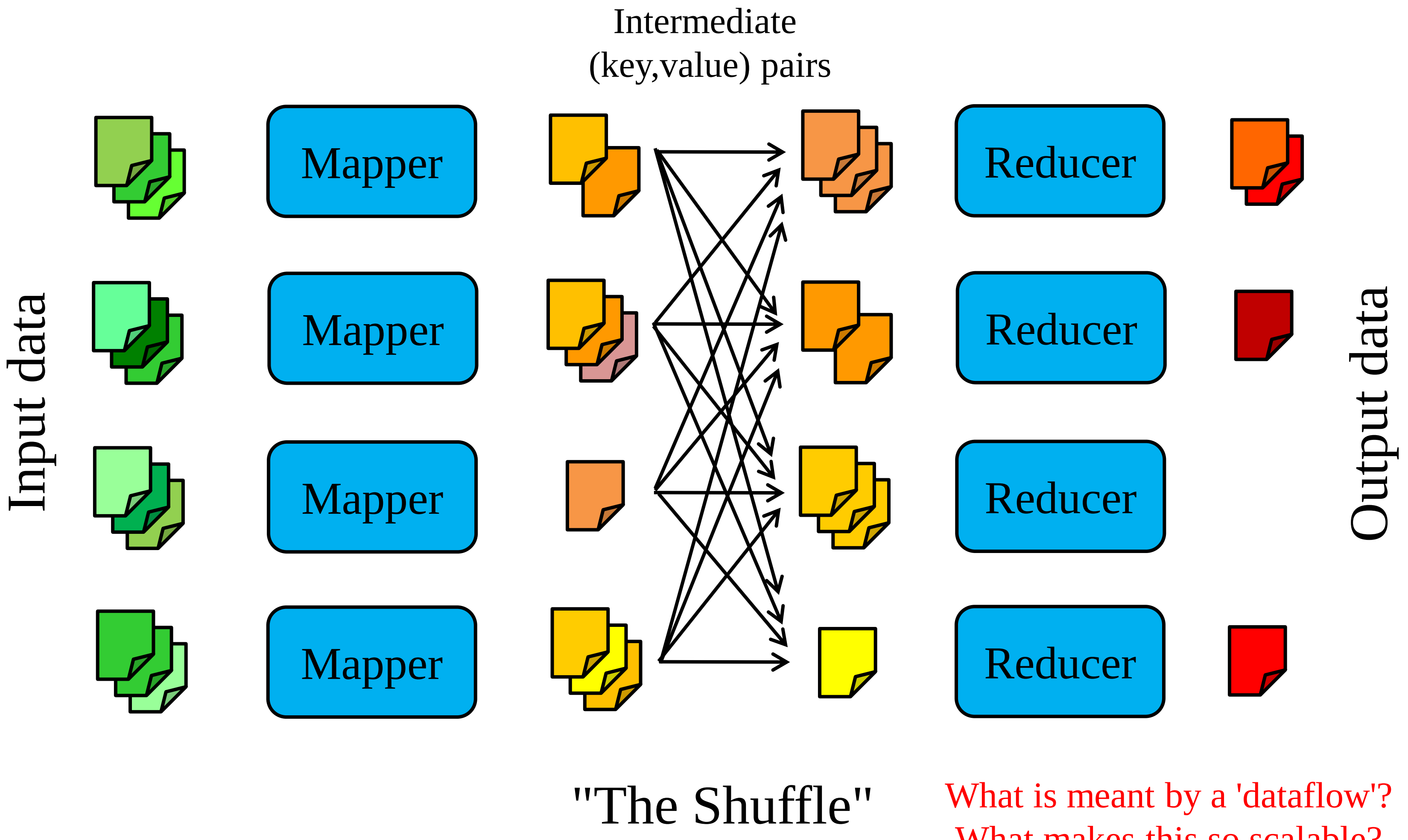
3 Each KV-pair output by the mapper is sent to the reducer that is responsible for it

4 The reducers sort their input by key and group it

5 The reducers process their input one group at a time

Cork Complex Systems Lab

MapReduce dataflow

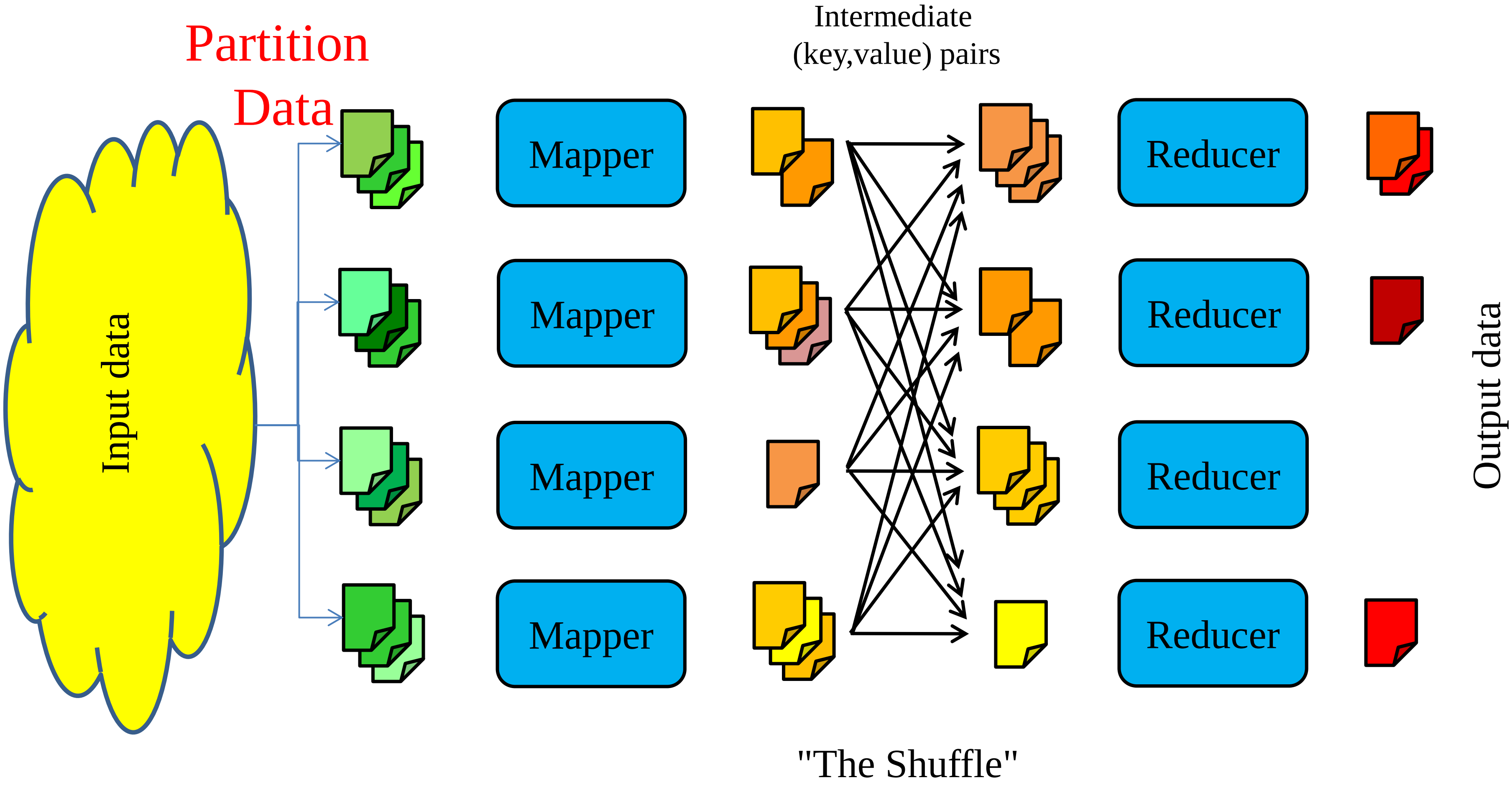


What is meant by a 'dataflow'?
What makes this so scalable?

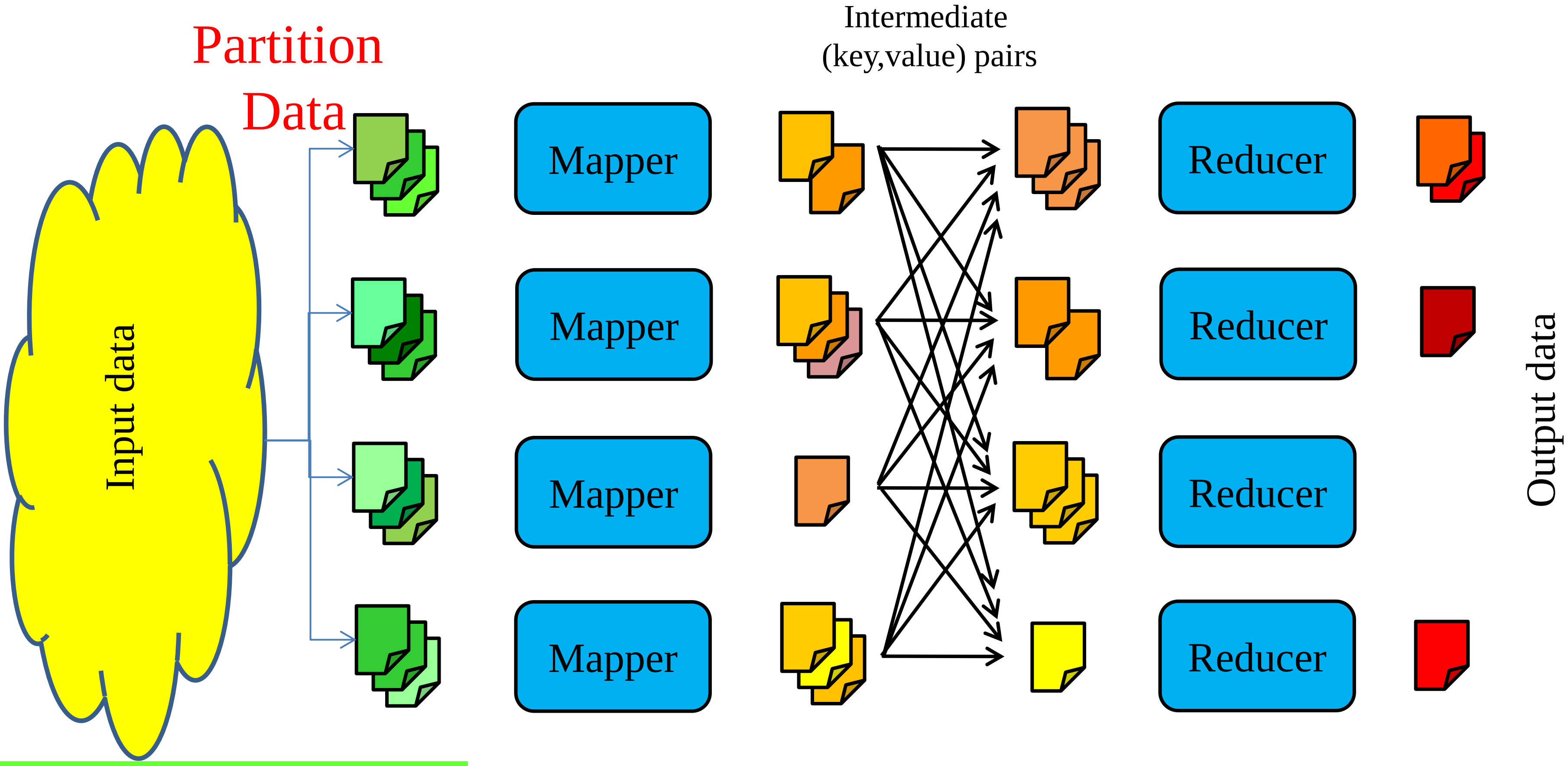
MapReduce Algorithm Classes

- Aggregation/Enumeration
  - Word count
  - Count URL access events
- Graph analytics
  - FaceBook friendship analysis
  - PageRank
- Machine Learning
  - Bayesian learning
  - clustering

MapReduce Approach



MapReduce Input Data: Graph Analytics



"The Shuffle"

Webgraph,
FaceBook Social Network

More examples

- Distributed grep – all lines matching a pattern
  - Map: filter by pattern
  - Reduce: output set
- Count URL access frequency
  - Map: output each URL as key, with count 1
  - Reduce: sum the counts
- Reverse web-link graph
  - Map: output (target,source) pairs when link to target found in souce
  - Reduce: concatenates values and emits (target,list(source))
- Inverted index
  - Map: Emits (word,documentID)
  - Reduce: Combines these into (word,list(documentID))