MAP REDUCE ALGORITHMS

CLOUD COMPUTING

Dr. Atm Shafiul Alam

a.alam@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

Thanks to Dr. Gareth Tyson and Dr. Félix Cuadrado who created the original version of these slides

QUICK RECAP…

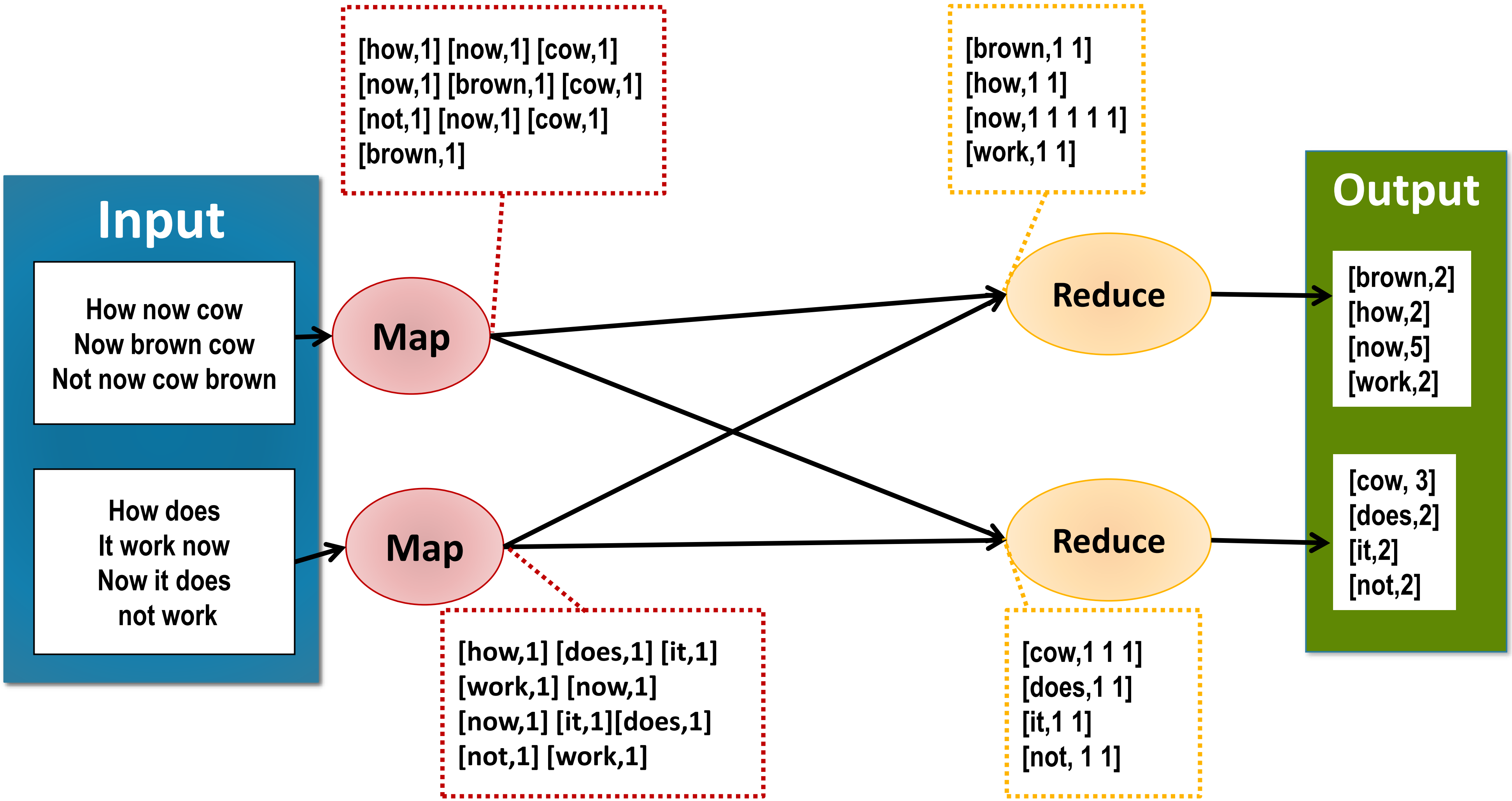
Yesterday we...

- Learnt more about the key components of Map Reduce
- Introduced the Combiner
- Learnt about Apache Hadoop
  - YARN
  - HDFS

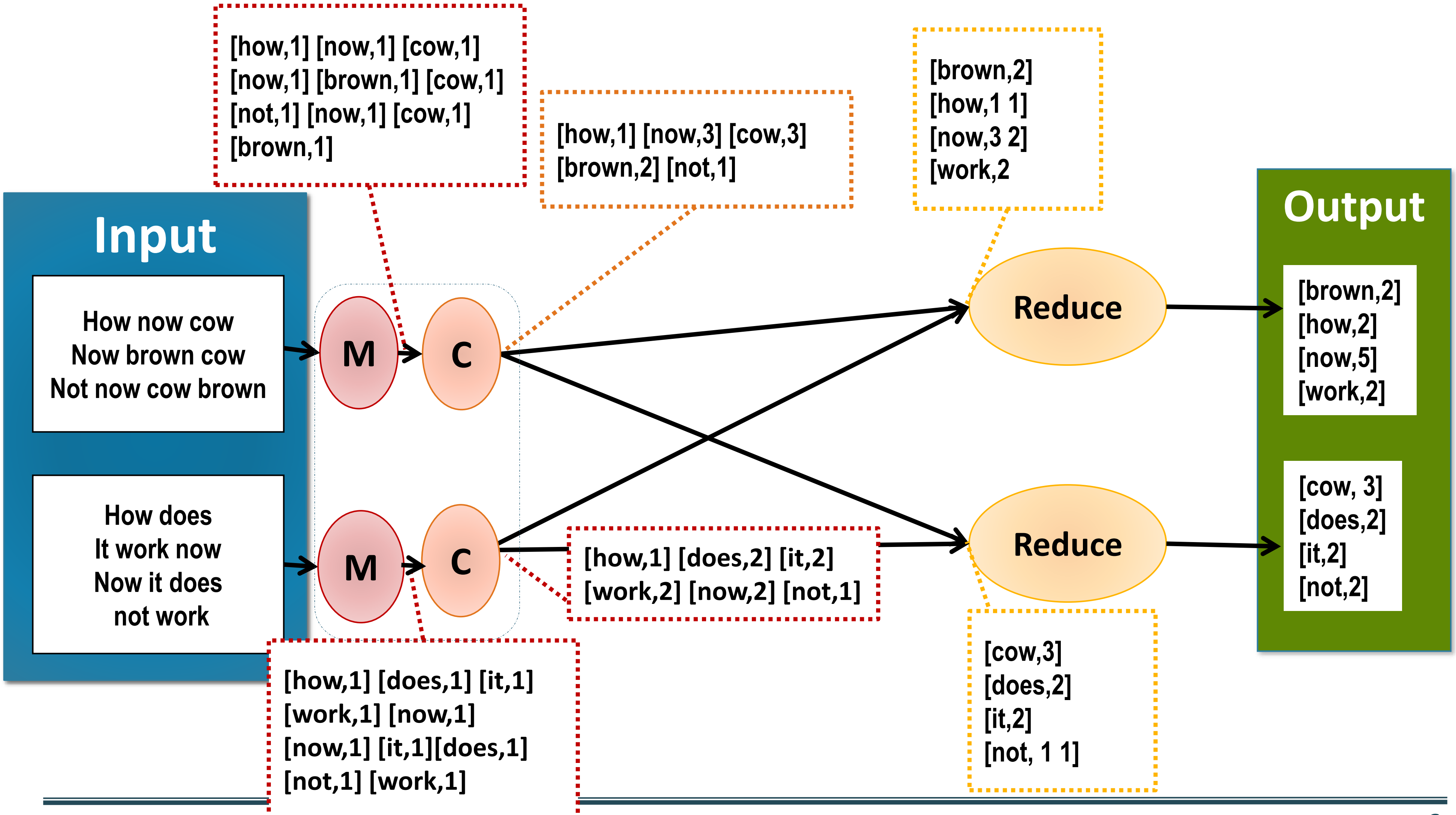
The Combiner

- The **combiner** acts as a **preliminary reducer**
- Runs on the mapper
- Reduces the number of emitted items
  - Improves efficiency

Word Count Example



Word Count Example with Combiner



Hadoop Architecture

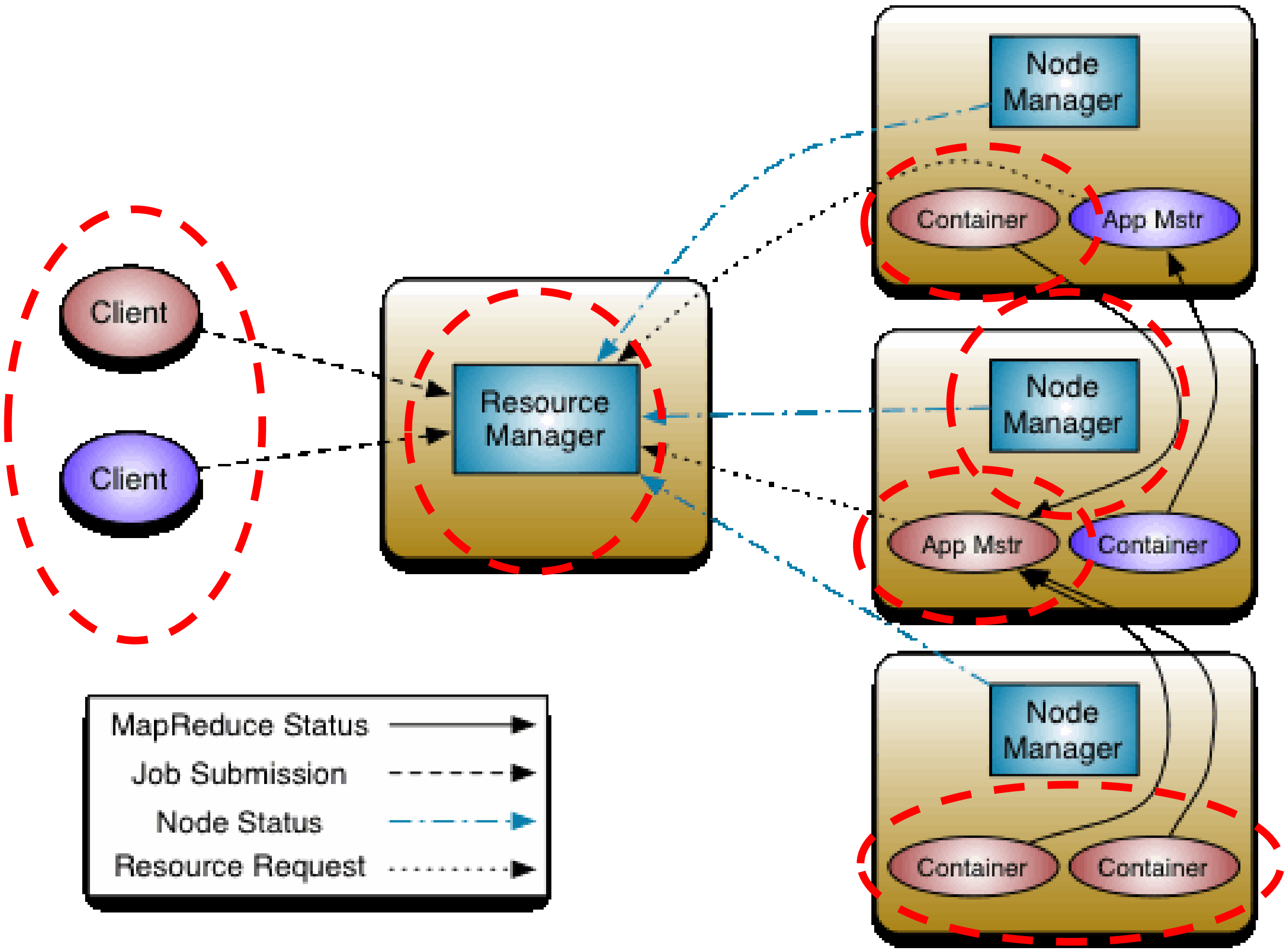
- Hadoop executes on a cluster of networked PCs
- Each node runs a set of daemons
  - ResourceManager
  - NodeManager

  Computing

  - NameNode
  - SecondaryNameNode
  - DataNode

  Storage

Job Execution Architecture (YARN)



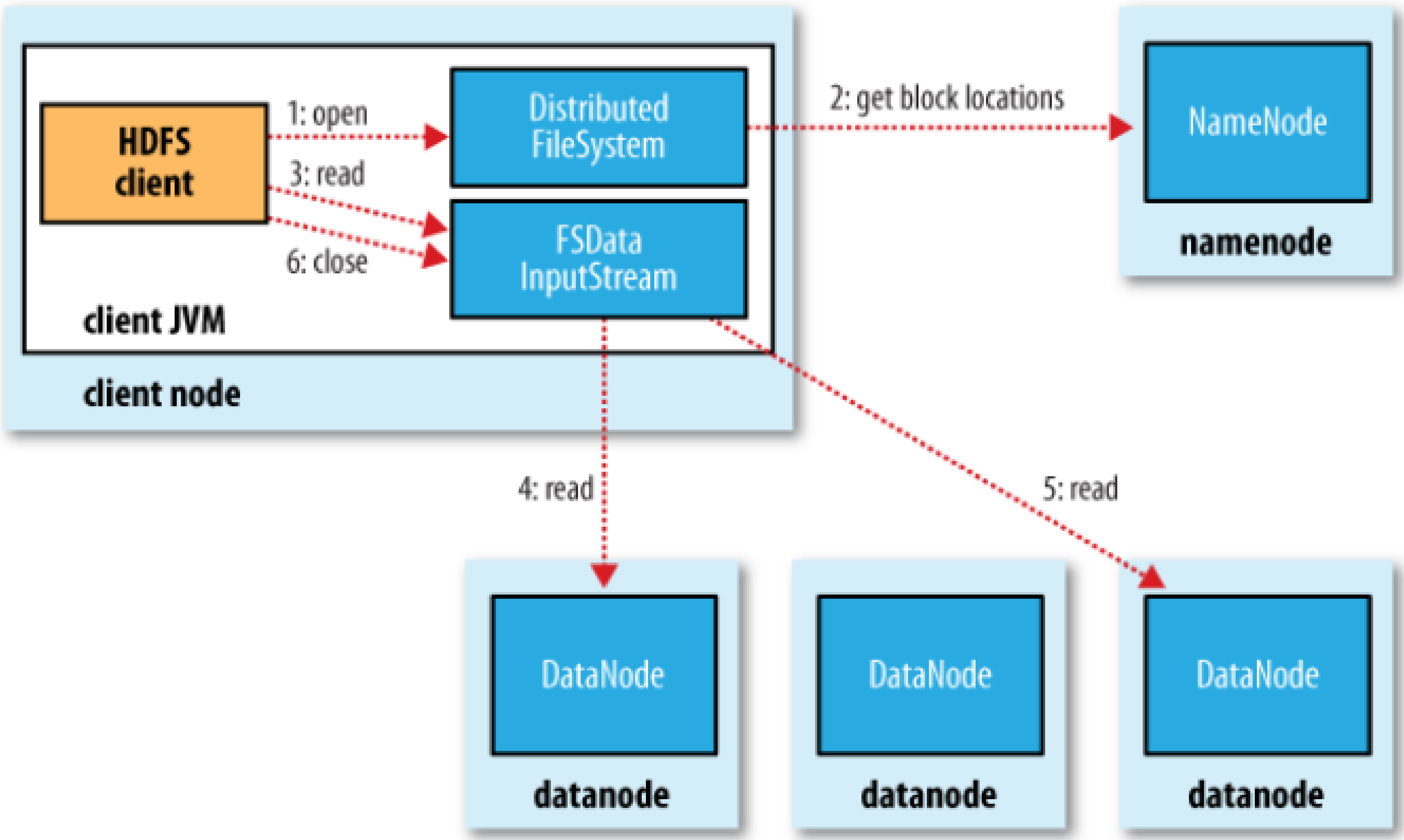
HDFS

- HaDoop Distributed Filesystem
  - Shared storage among the nodes of the Hadoop cluster
- Storage for input **and** output of MapReduce jobs
- Need to copy any data from **your** file system to the **HDFS**

Hadoop Storage Daemons

- DataNode (1..* per cluster)
  - Stores blocks from the HDFS
- NameNode (1 per cluster)
  - Keeps index table with (all) the locations of each block
- Secondary Namenode (1 per cluster)
  - Stores backup copy of index table

HDFS File Read operation



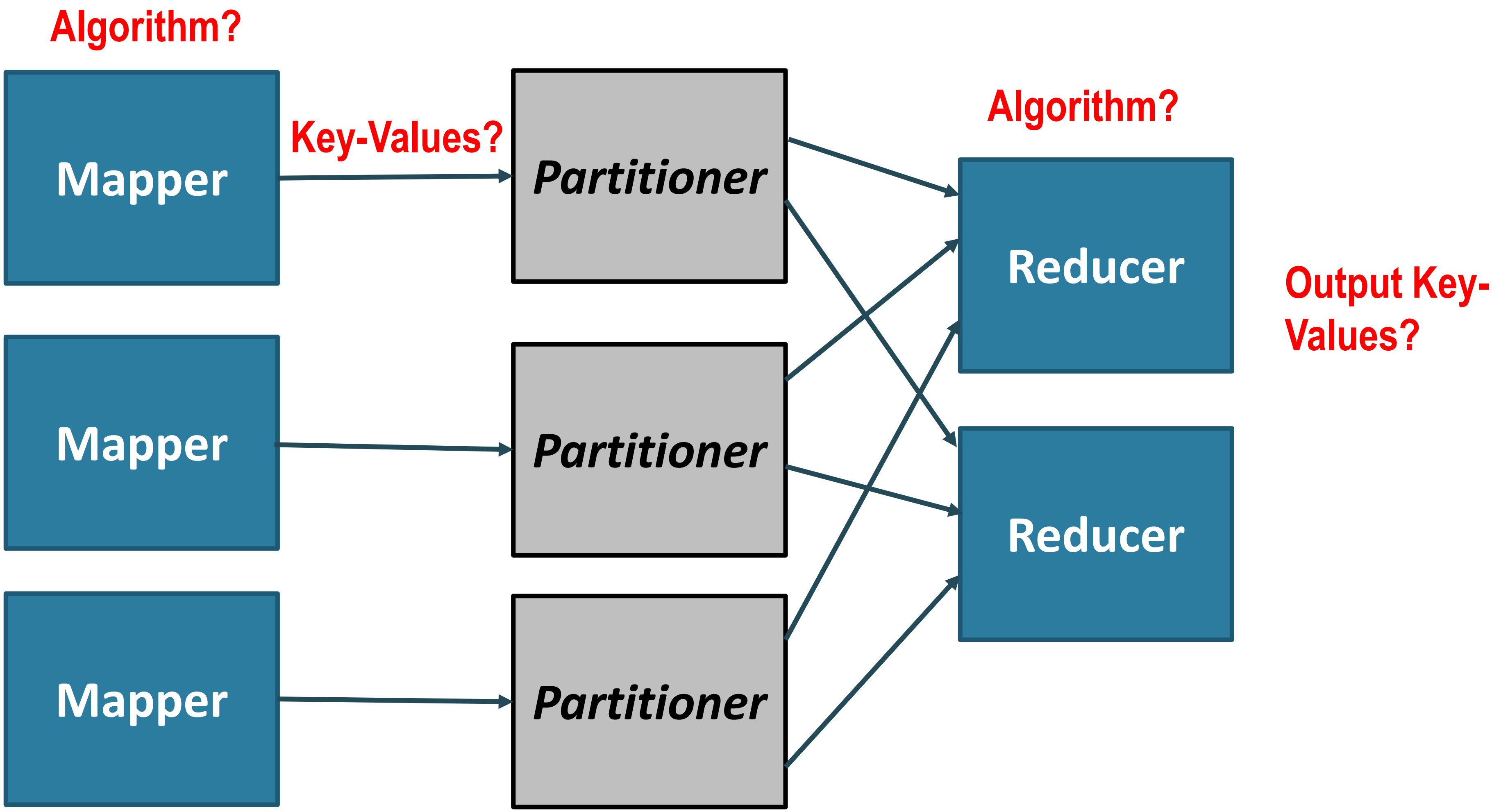
Today's contents

- **Numerical Summarization pattern**
- Inverted index pattern
- Data filtering
- Data joins

Numerical **Summarization**

- **Goal:** Calculate aggregate statistical values over a large dataset
  - Extract features from the dataset elements, compute the same function for each feature
- **Motivation:** Provide a top-level view of large input datasets to identify trends anomalies...
- **Examples:**
  - Count occurrences
  - Maximum / minimum values
  - Average / median / standard deviation

Decision points...



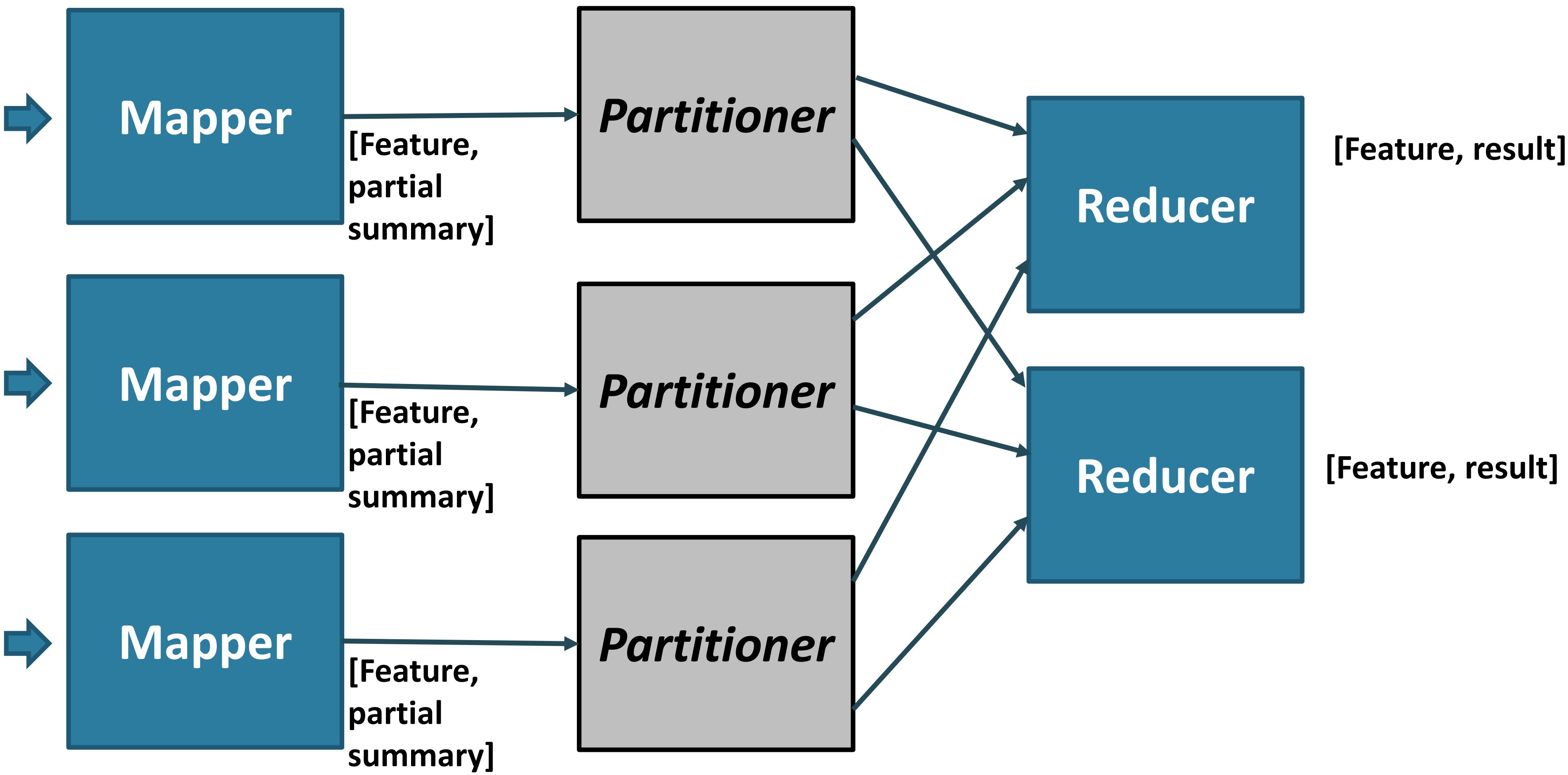
Sample dataset: China's Air Quality sensors

location	aqi	type	essential	pm25	pm10	co	no2	o31	o38	so2	ts
海淀区万柳	325	严重污染	细颗粒物(PM2.5)	275	0	1.8	71	174	90	60	16-03-16 10:35:
海淀区万柳	325	严重污染	细颗粒物(PM2.5)	275	0	1.8	71	174	90	60	16-03-16 09:50:
海淀区万柳	303	严重污染	细颗粒物(PM2.5)	253	263	1.6	67	165	78	56	16-03-16 09:35:
海淀区万柳	311	严重污染	细颗粒物(PM2.5)	261	267	1.8	88	139	55	60	16-03-16 08:35:
海淀区万柳	323	严重污染	细颗粒物(PM2.5)	273	293	2.2	146	70	35	54	16-03-16 07:35:
海淀区万柳	299	重度污染	细颗粒物(PM2.5)	249	251	1.8	110	70	26	48	16-03-16 06:35:

Sample numerical summarization questions

- Compute what is the **maximum** PM2.5 registered for each location provided in the dataset
- Return the **average** AQI registered each week
- Compute for each day of the week the **number of locations** where the PM2.5 index exceeded 150

Numerical Summarization Structure



Writing Map and Reduce functions

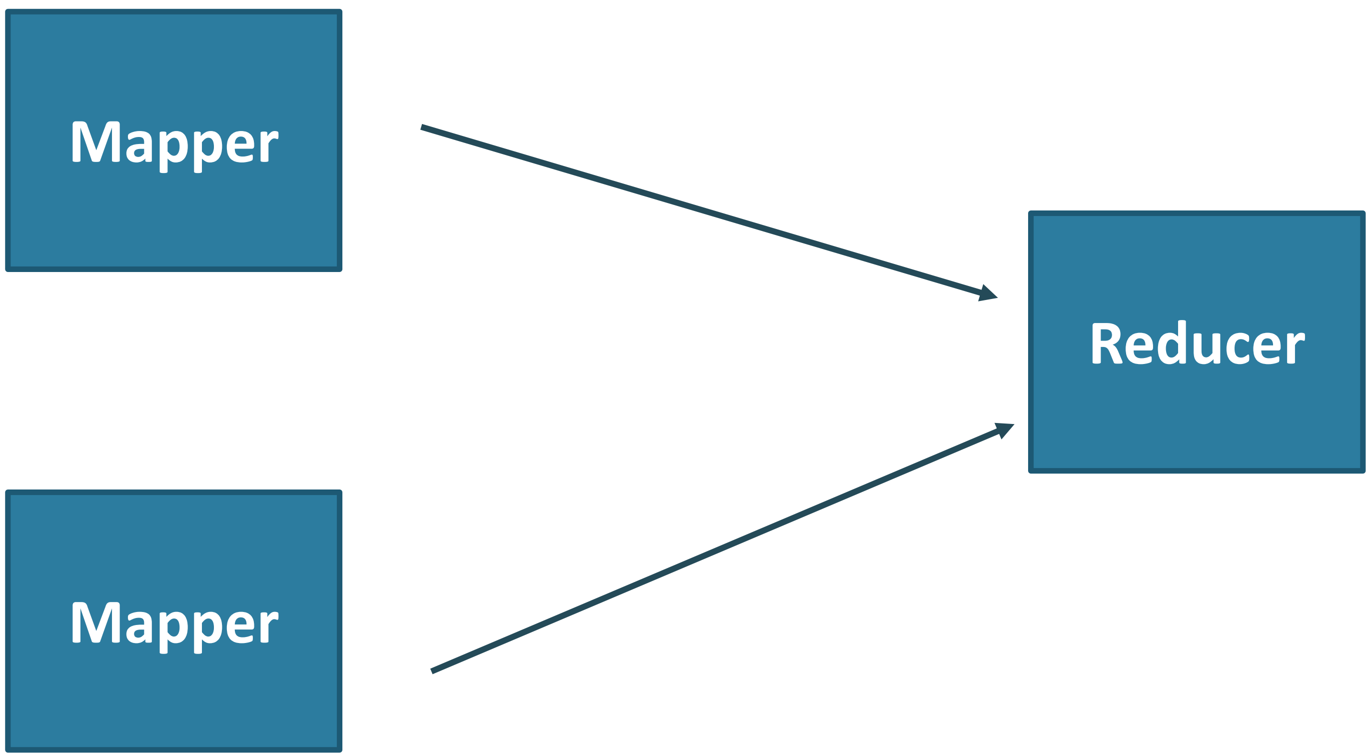
- **Mapper**
  - Find features in Input (e.g. words)
  - Set partial aggregate value for the features in that iteration (i.e. what key-value pairs)
- **Reducer**
  - Compute final aggregate result from all the intermediate values (the output)
- **Combiner?**

Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100

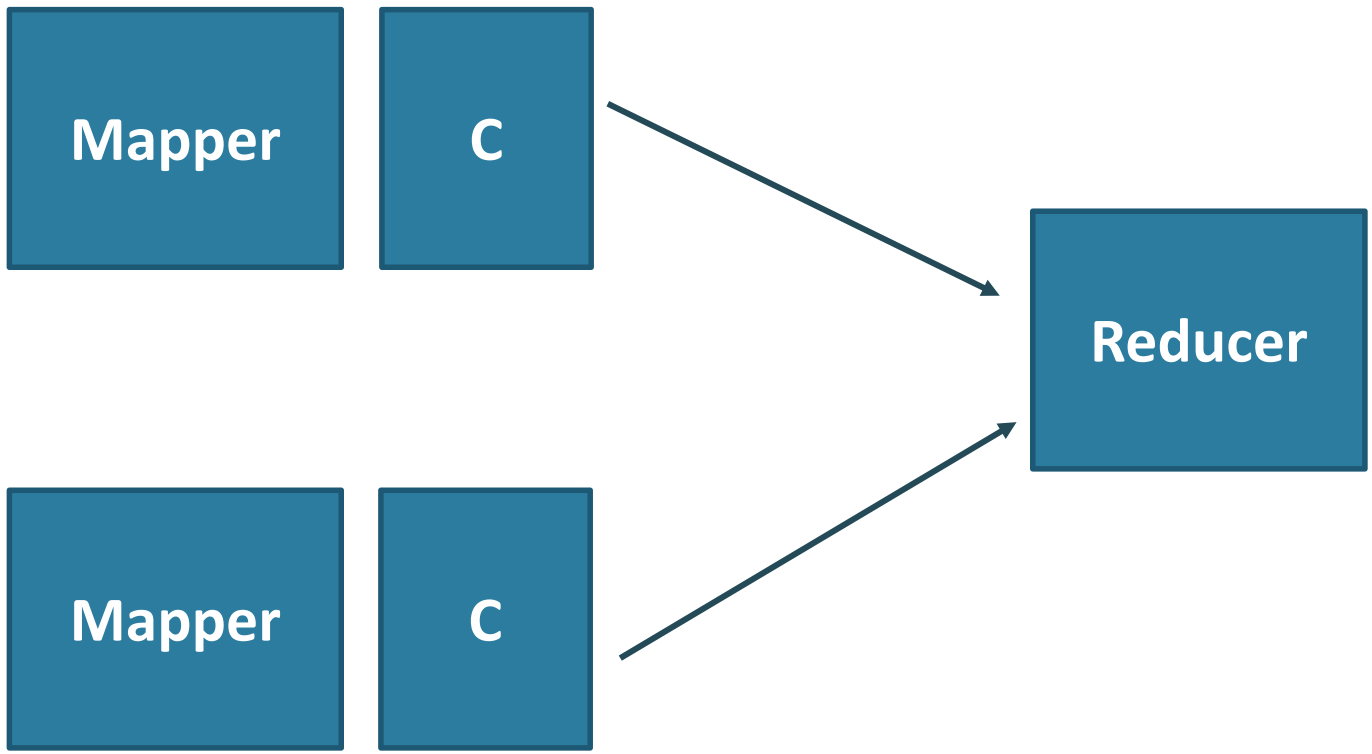


Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100

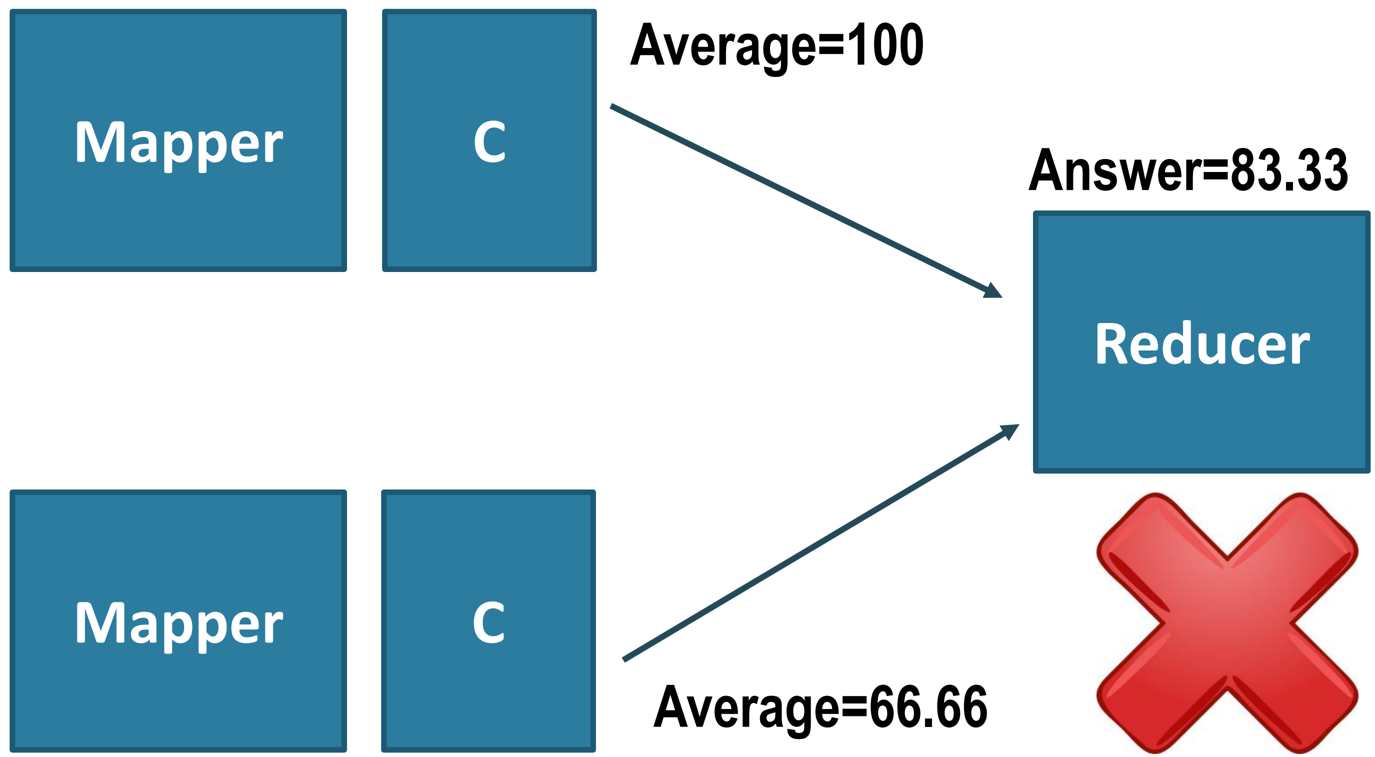


Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100



Combining Averages

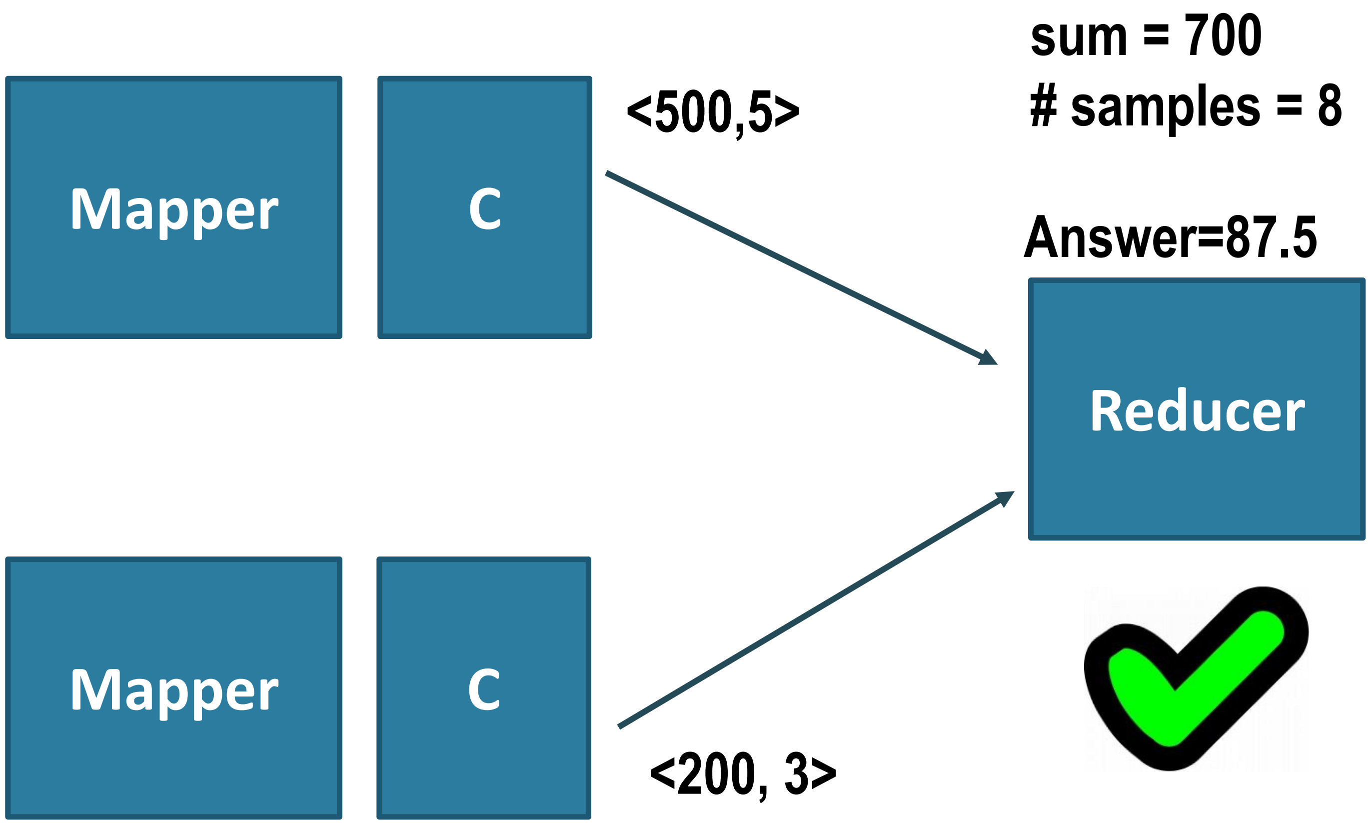
- Average is NOT an **associative operation**
  - Cannot be executed partially with the Combiners
- Solution: Change Mapper results
  - Emit aggregated quantities, and number of elements
  - **Mapper:** For input entries (100,100,20),
    - Emit (100,<u>1</u>),(100,<u>1</u>), (20,<u>1</u>)
  - **Combiner:** adds aggregates and number of elements
    - Emits (220,3)
  - **Reducer**
    - Adds aggregates and computes average

Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100



When should you use a combiner?

- When you can pre-aggregate (pre-reduce) data on the mapper without changing the outcome
- Remember – combiners must be **optional**
  - Outcome must still be the same without the combiner
- Combiner simply reduces network traffic…
  - Reduces number of emitted values

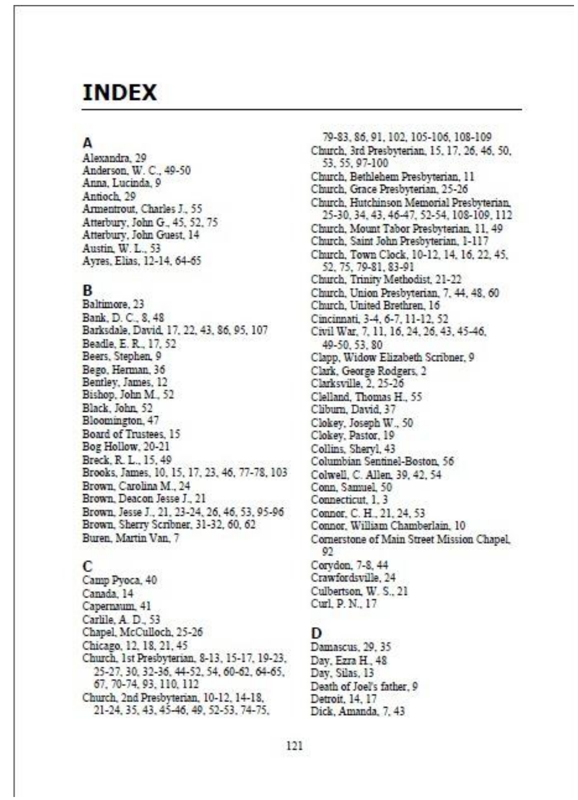
Contents

- Numerical Summarization pattern
- **Inverted index pattern**
- Data filtering
- Data joins

Inverted index

- **Goal:** Generate index from a dataset to allow faster searches for specific features

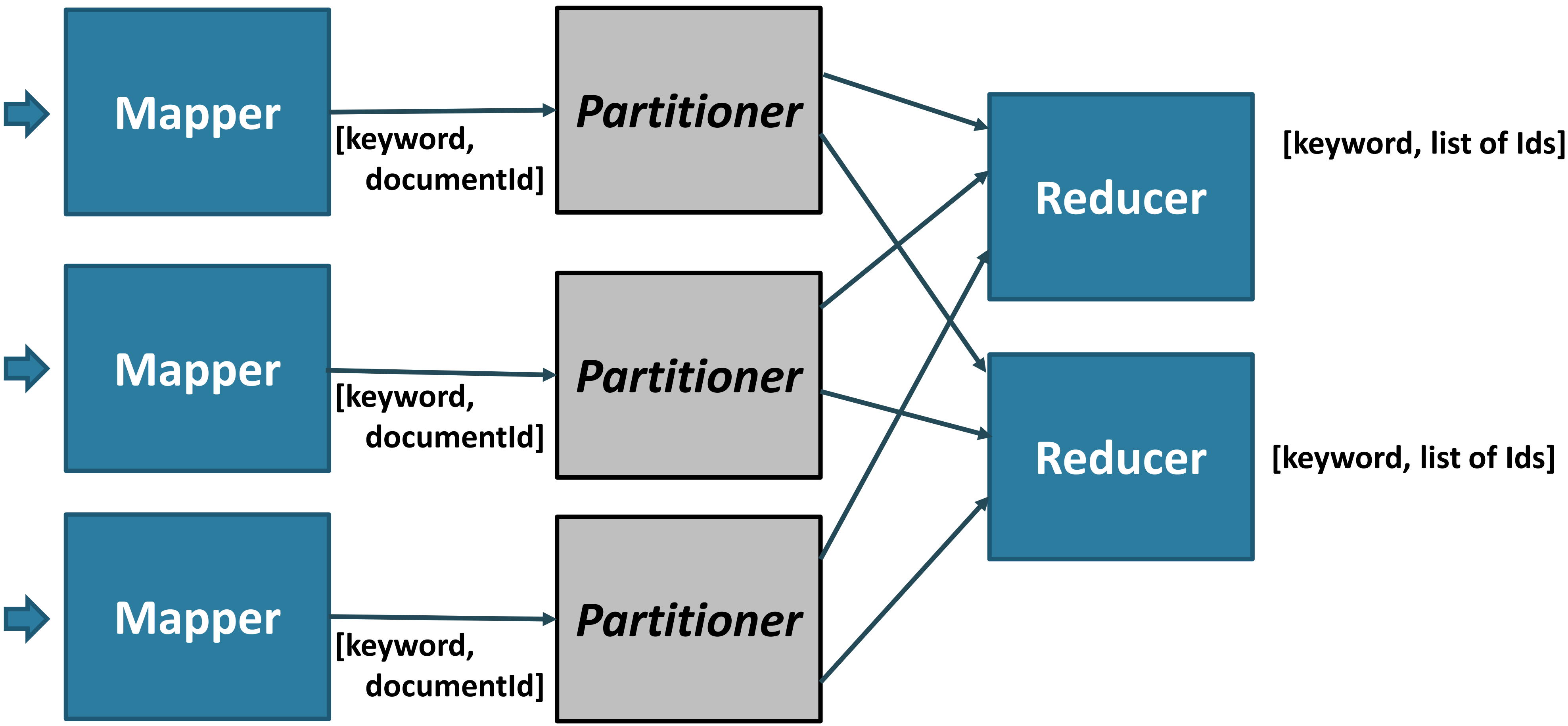
- **Motivation:** improve search efficiency
- Examples:
  - Building index from a textbook.
  - Finding all websites that match a search term



INDEX

121

Inverted Index Structure



Writing Map and Reduce functions

- **Mapper**
  - Find features in Input (set of keywords to index)
  - Emit [keyword, document identifier] as [key, value]
- **Reducer**
  - Identity function (emits the list of results provided in shuffle and sort)
  - i.e., Receive a set of identifiers for each keyword and simply concatenate them
- **Combiner?**

Inverted Index Mapper

```
public void map (String docId, String text){

   String[] features = findFeatures(text);

      for(String feature: features){
            emit(feature, docId);
      }
}
```

Inverted Index Reducer

```
public void reduce (String feature,
                                  String[] docIds){

    emit(feature, formatNicely(docIds))
     //formatNicely() combines indexes into list
     //that is easy to read
}
```

Some questions…

- How many key-value pairs will be fed into each mapper?
  - Each mapper will get a roughly even share of the input keys
- How many key-value pairs will be emitted by each mapper?
  - Based on the algorithm - one key will be emitted per “feature” identified
  - Divided by #mappers (e.g. /10 if there are 10 mappers)
- How many keys will be fed into each reducer?
  - Each reducer will get a roughly even share of the keys

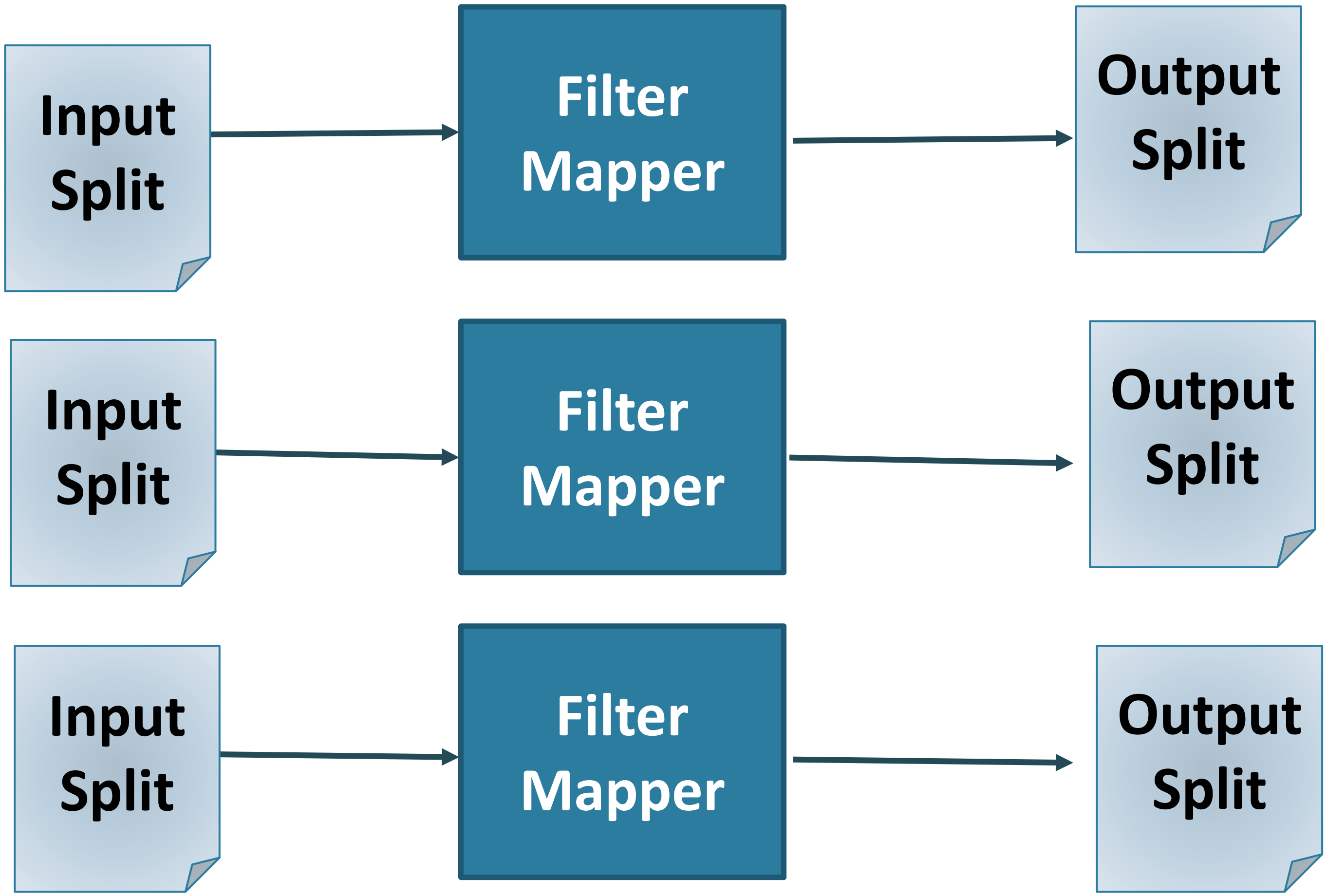
Contents

- Numerical Summarization pattern
- Inverted index pattern
- **Data filtering**
- Data joins

Filtering

- **Goal:** Filter out records/fields that are not of interest for further computation.
- *Speedup* the actual computation – thanks to a reduced size of the dataset
- Examples:
  - Distributed grep (text search).
  - Tracking a thread of events (logs from the same user)
  - Data cleansing
- Mapper only job

Filtering Structure



Top ten elements

- **Goal:** Retrieve a small number of records (top 10), relative to a ranking function
  - Focus on the most important records of the input dataset
- **Motivation:** frequently the interesting records are those ranking first according to a ranking function
- Examples:
  - Build top 10 sellers view (e.g. from Alibaba)
  - Find unusual outliers in the data
  - Most profitable items

Example: Top 10 music singles

Entered (week ending)	Weeks in top 10	Single	Artist
			Singles in 2015
21 April 2016	23	"One Dance" (#1)	Drake featuring Wizkid & Kyla
10 September 2015	22	"What Do You Mean?" ‡	Justin Bieber
5 November 2015	18	"Sorry"‡	Justin Bieber
26 November 2015	17	"Love Yourself" ‡	Justin Bieber
31 March 2016	16	"Cheap Thrills" (#3)	Sia featuring Sean Paul
26 May 2016	16	"Too Good"	Drake featuring Rihanna
25 August 2016	16	"Closer" (#7)	The Chainsmokers featuring Halsey
29 September 2016	16	"Say You Won't Let Go"	James Arthur
10 March 2016	15	"I Took a Pill in Ibiza" (#4)	Mike Posner
12 May 2016	15	"This Is What You Came For" (#5)	Calvin Harris featuring Rihanna
14 July 2016	15	"Dancing on My Own"	Calum Scott
6 October 2016	15	"Starboy"	The Weeknd featuring Daft Punk
3 November 2016	15	"Rockabye"	Clean Bandit featuring Sean Paul & Anne-Marie
5 November 2015	14	"Hello"‡	Adele
11 February 2016	13	"7 Years" (#2)	Lukas Graham
25 February 2016	13	"Lush Life" (#6)	Zara Larsson
15 December 2016	13	"Human"	Rag'n'Bone Man
11 February 2016	12	"Work" (#9)	Rihanna featuring Drake
4 August 2016	12	"Cold Water"	Major Lazer featuring Justin Bieber & MØ
4 February 2016	11	"Fast Car"	Jonas Blue featuring Dakota
24 March 2016	11	"Work from Home"	Fifth Harmony featuring Ty Dolla Sign
14 July 2016	11	"Don't Let Me Down"	The Chainsmokers featuring Daya

Queen Mary University of London

Example: Top 10 student marks

Student	Mark
Zoë	98
John	96
Fred	90
Alfred	86
Melissa	84
Emily	70
Faisal	67
Elizabeth	65
Phil	63
Michael	60
Archibald	60
Emma	56
Betty	52
Sean	49

Top Ten Map and Reduce functions

- **Mapper**
  - Emits null, value with (ranking, record)
- **Reducer**
  - Sort all values by ranking, emit k times null, value with (ranking, record)
- **Combiner**
  - Same as Reducer

Top Ten Mapper

```
public void map (String studentId, double grade){
    emit(null, new Pair(studentId, grade));
}
```

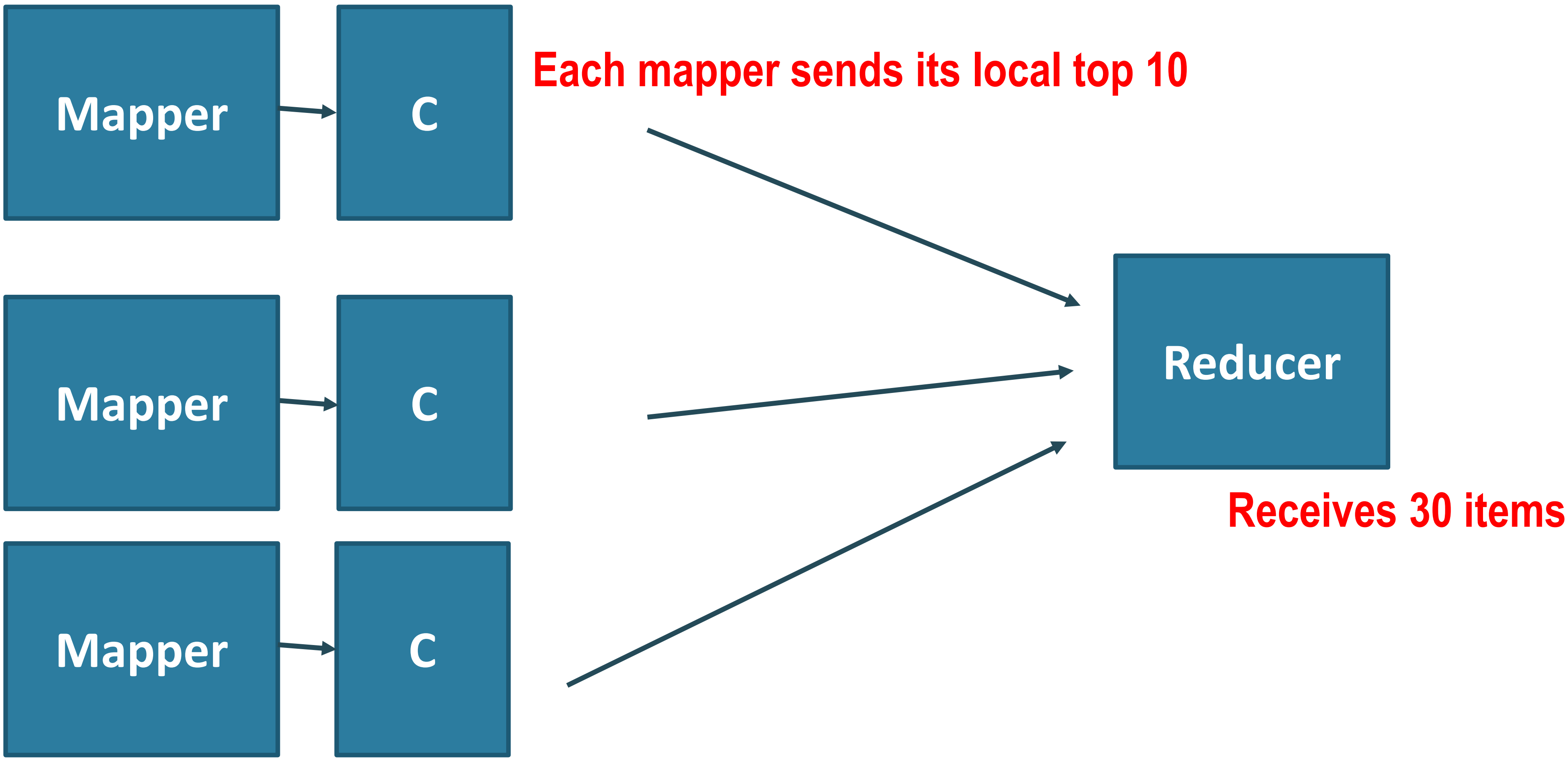
Use of null key means that everything will go into single reduce() function call

Top Ten Reducer

```
public void reduce (null, Pair[] studentResults){
    list rankedStudents =
            sortResultsByGrade(studentResults)
    list topTen = rankedStudents.getTop(10);
    int rank = 1;

    for(Pair student: topTen)
        emit(rank, student.getId +
                        student.getGrade())
        rank++;
    }
}
```

Top Ten Structure



Top Ten Performance

- How many Reducers?
  - Performance issues?
- What happens if we don’t use Combiners?
- Performance depends greatly on the number of elements (to a lesser extend on the size of data)
- Minimum requirement:
  - The ranking data for a whole input split must fit into memory of a single Mapper

Contents

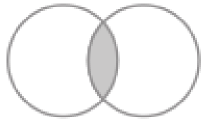
- Numerical Summarization pattern
- Inverted index pattern
- Data filtering
- **Data joins**

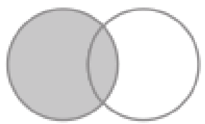
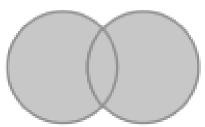
Join Definition

- **Goal:** combine together related data
- E.g. Combine Amazon user purchase log from *AND* the logs of the web servers
- What else would you use join patterns for in this context?
    - Relate purchase habits to demographics
    - Send reminders to inactive users
    - Recommendation systems

Types of Joins

- **Inner:** compare all tuples in relations L and R, and produce a result if a join predicate is satisfied.



- **Outer:** Doesn't require both tuples to match based on a join predicate. Instead, can retain a record from L or R even if no match exists.
  - Left outer 
  - Full outer 

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
        JOIN `course` on user.course = course.id;
```

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
  INNER JOIN `course` on user.course = course.id;
```

Alice	HTML5
Bob	CSS3
Carol	CSS3
David	MySQL

Emma missing because she has no course

Relational Database Joins

user

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

course

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

```
SELECT user.name, course.name FROM `user`
  RIGHT JOIN `course` on user.course = course.id;
```

Alice	HTML5
Bob	CSS3
Carol	CSS3
NULL	Java
NULL	Python
David	MySQL

Joining datasets in Hadoop

1. **Replication join**—An outer **map-side** join where one of the datasets is *small enough to be kept in memory.*

2. **Repartition join**—A **reduce-side** join for joining two or more *large datasets*.

3. **Semi-join**—A **map-side** join where one out of several large datasets is filtered so that it fits in memory.

Replication Joins

- **Idea:** Replicate smallest dataset to all the map hosts using Hadoop's distributed cache.
- **Map:**
  1. Use the initialization method of each map task to load the small dataset into a hashtable
  2. Use the key from each record of the large dataset to look up the small dataset hashtable
  3. Join between the large dataset record and all of the records from the small dataset that match the join value.
- No Reducer is needed

Replication Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

Map Task 1

Create HashTable from course

Map Task 2

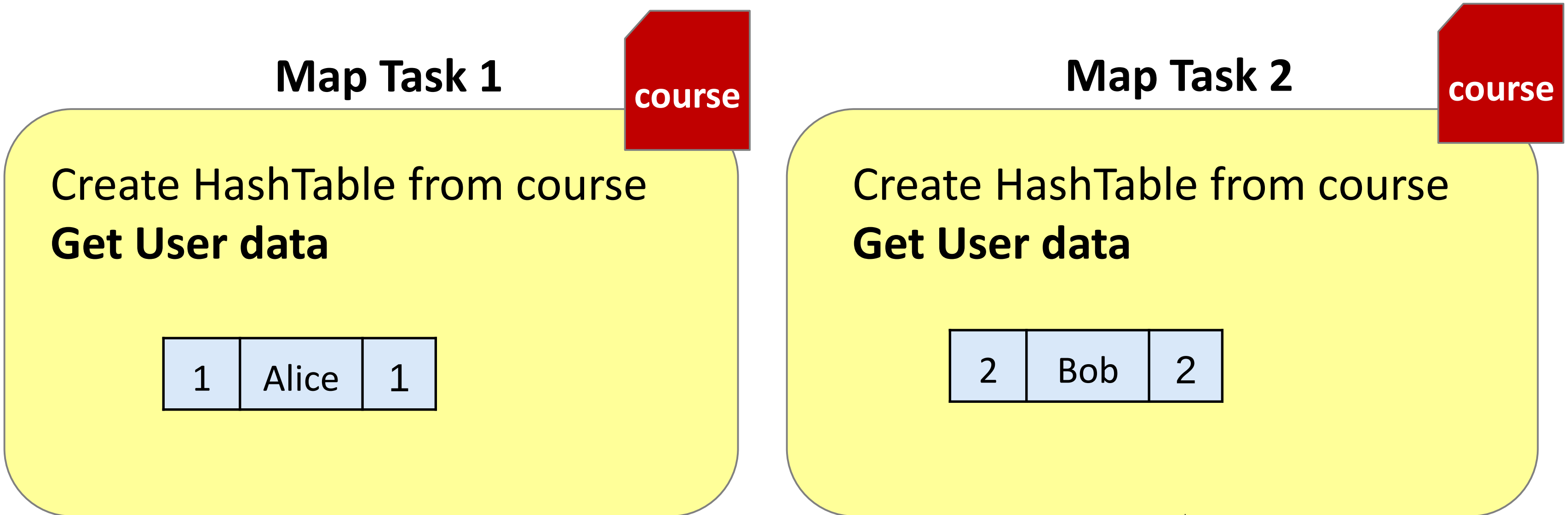
Create HashTable from course

course

Read courses from distributed cache and make local to all the mappers

Replication Joins (II)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL



user

course

id	name	course
1	Alice	1
3	Carol	2
5	Emma	NULL

Input split 1

id	name	course
2	Bob	2
4	David	5
6	John	NULL

Input split 2

Replication Joins (III)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

user

course

Map Task 1

course

Create HashTable from course

Get User data

Lookup in Hashtable

1	Alice	1

id	name	course
3	Carol	2
5	Emma	NULL

Map Task 2

course

Create HashTable from course

Get User data

Lookup in Hashtable

2	Bob	2

id	name	course
4	David	5
6	John	NULL

Replication Joins (IV)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

user

course

Map Task 1

course

Create HashTable from course
Get User data
Lookup in Hashtable
Join and Emit

Alice	HTML5

id	name	course
3	Carol	2
5	Emma	NULL

Map Task 2

course

Create HashTable from course
Get User data
Lookup in Hashtable
Join and Emit

Bob	CSS3

id	name	course
4	David	5
6	John	NULL

Replication Joins (IV)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

user

course

Map Task 1

course

Create HashTable from course

Get User data

Lo

Jo

Map Task 2

course

Create HashTable from course

Get User data

Works well if `course` is small and `user` is big

3	Carol	2
5	Emma	NULL

4	David	5
6	John	NULL

Repartition join

- **Idea:** Process both datasets in Mappers, emit the join key as the Map out key
- Perform the join at the reducer among all the elements
- Mapper will "tag" records from different files with same join key

Repartition Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

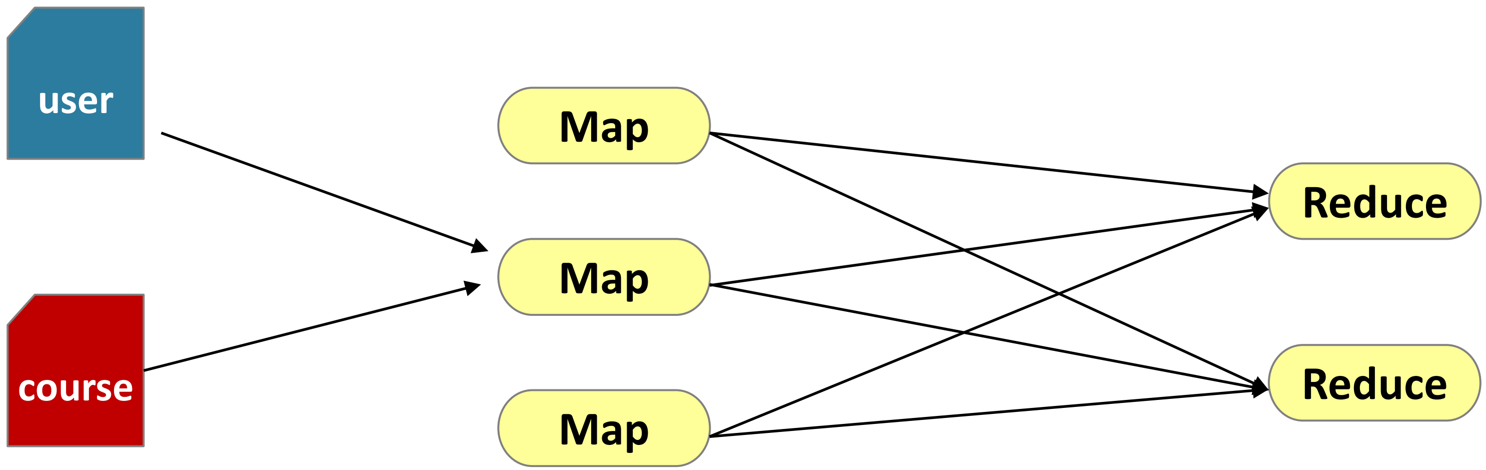
user

course

Repartition Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

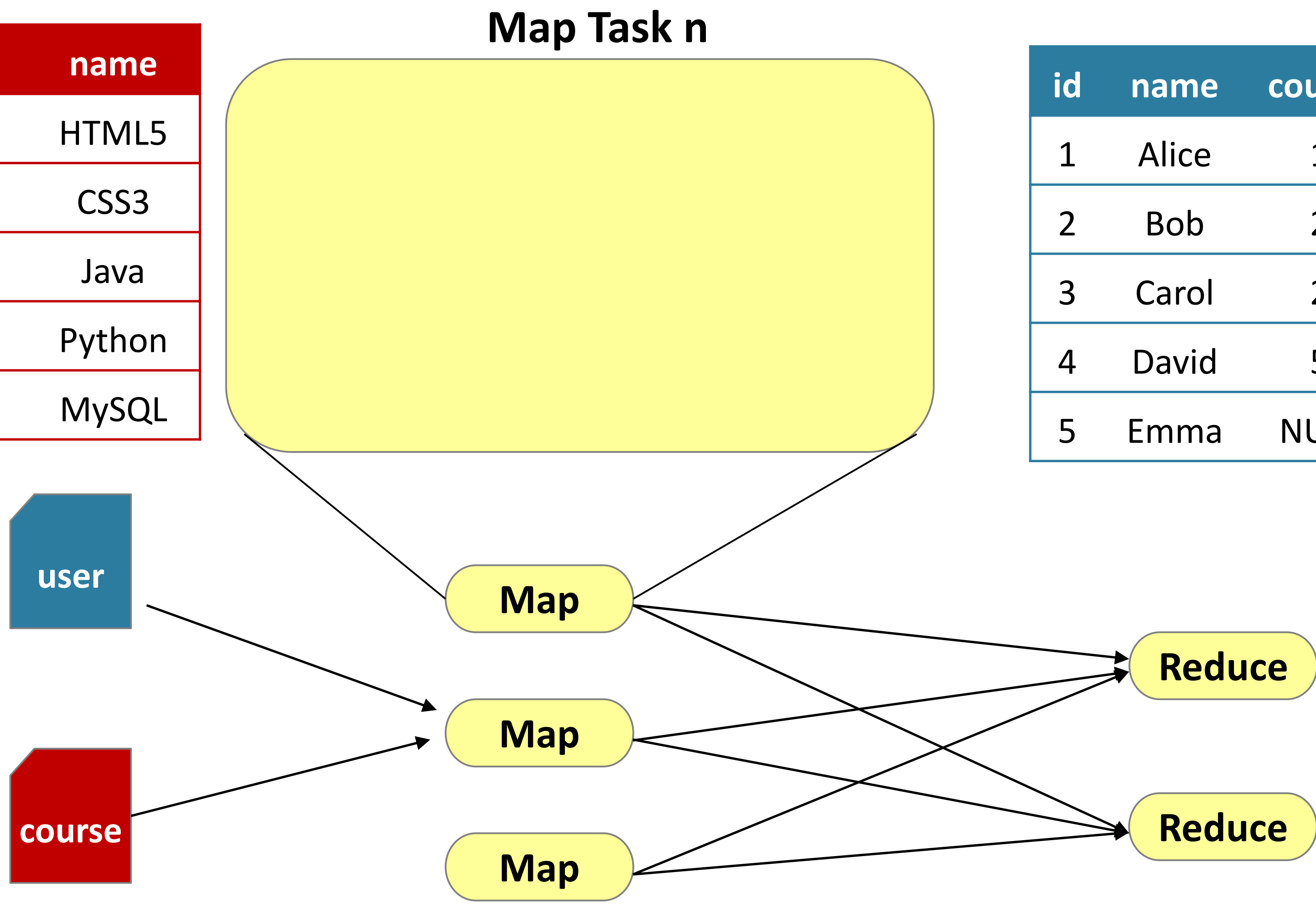


Based on Fig 4.2 From “Hadoop in Practice” by Alex Holmes

Repartition Joins (I)

id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL

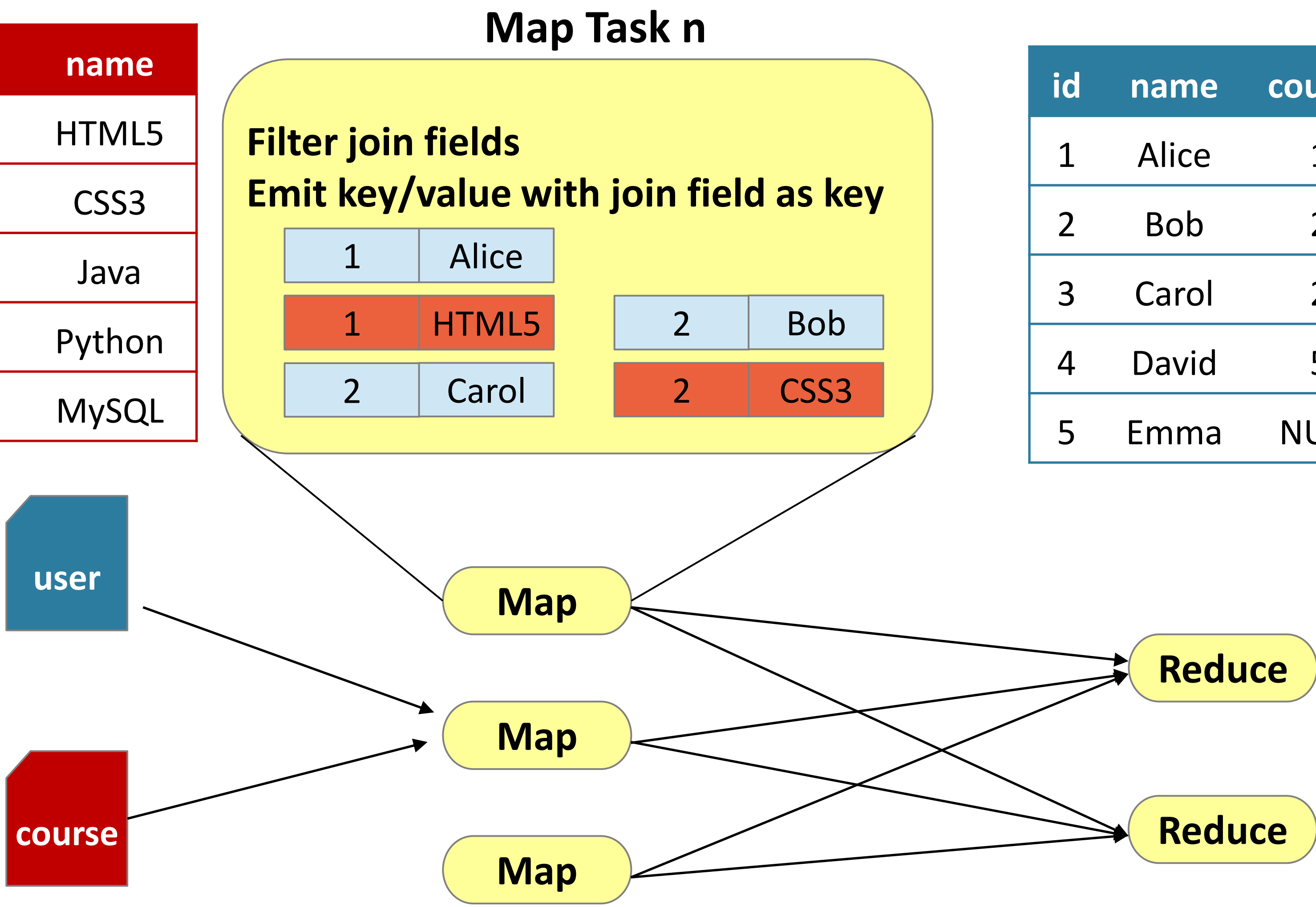


Based on Fig 4.2 From “Hadoop in Practice” by Alex Holmes

Repartition Joins (II)

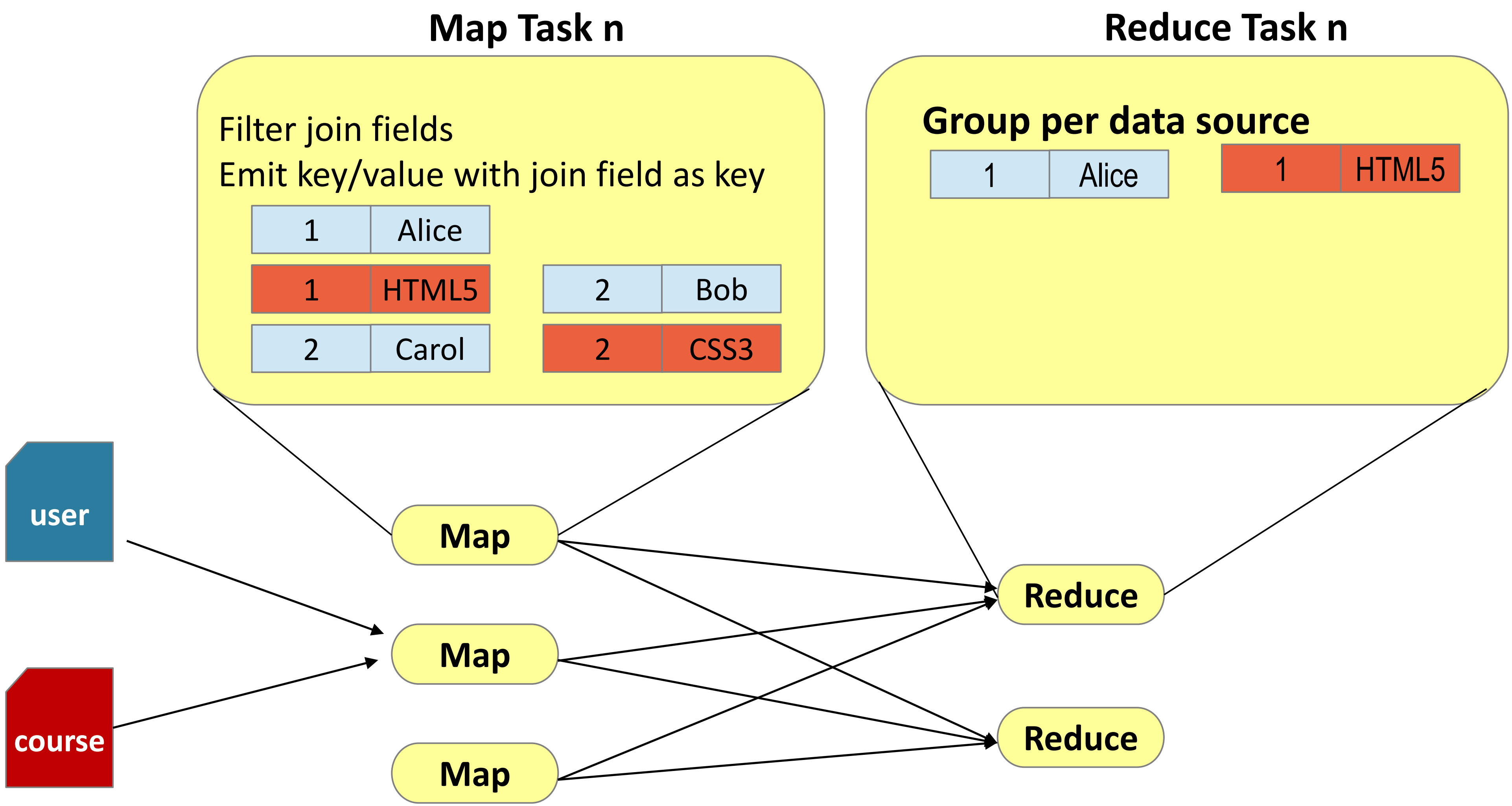
id	name
1	HTML5
2	CSS3
3	Java
4	Python
5	MySQL

id	name	course
1	Alice	1
2	Bob	2
3	Carol	2
4	David	5
5	Emma	NULL



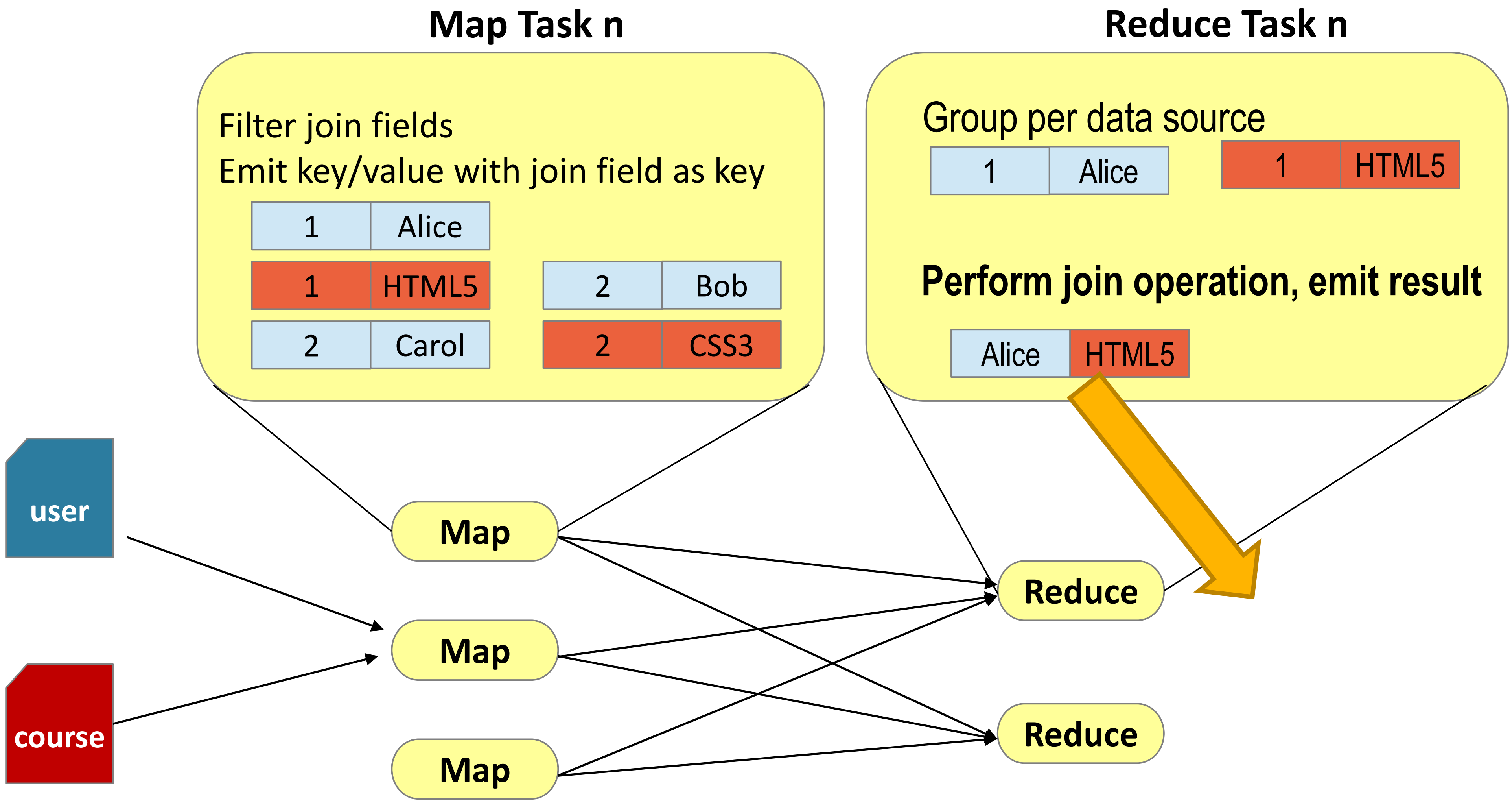
Based on Fig 4.2 From “Hadoop in Practice” by Alex Holmes

Repartition Joins (III)

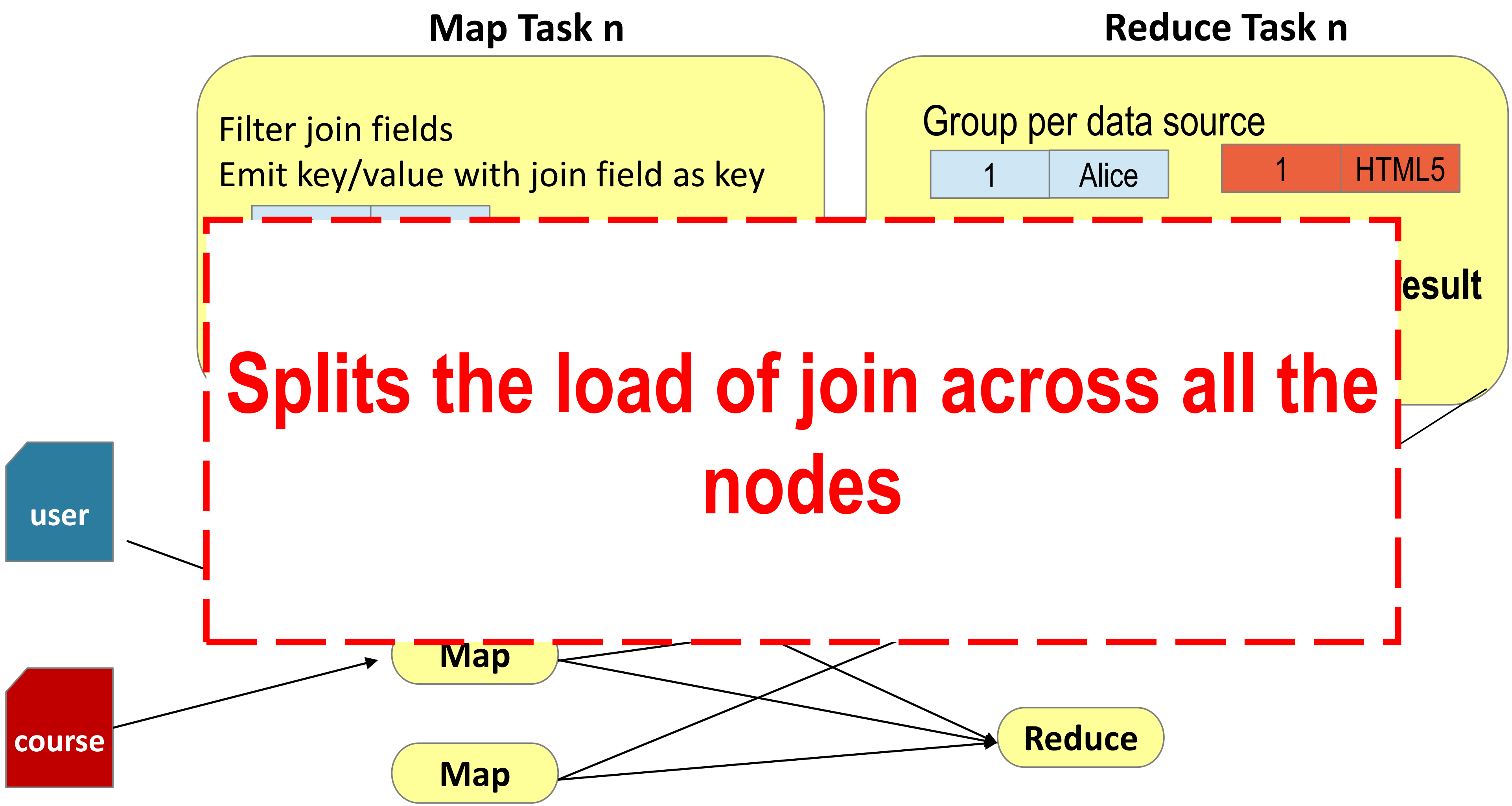


Based on Fig 4.2 From “Hadoop in Practice” by Alex Holmes

Repartition Joins (IV)



Repartition Joins (IV)



Based on Fig 4.2 From “Hadoop in Practice” by Alex Holmes

Summary

- Numerical Summarization pattern
- Inverted index pattern
- Data filtering
- Data joins