Queen Mary University of London

HADOOP PROGRAMMING

BIG DATA PROCESSING

Félix Cuadrado

felix.cuadrado@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

Contents

- **Apache Hadoop**
- The Combiner
- MapReduce aggregate computation

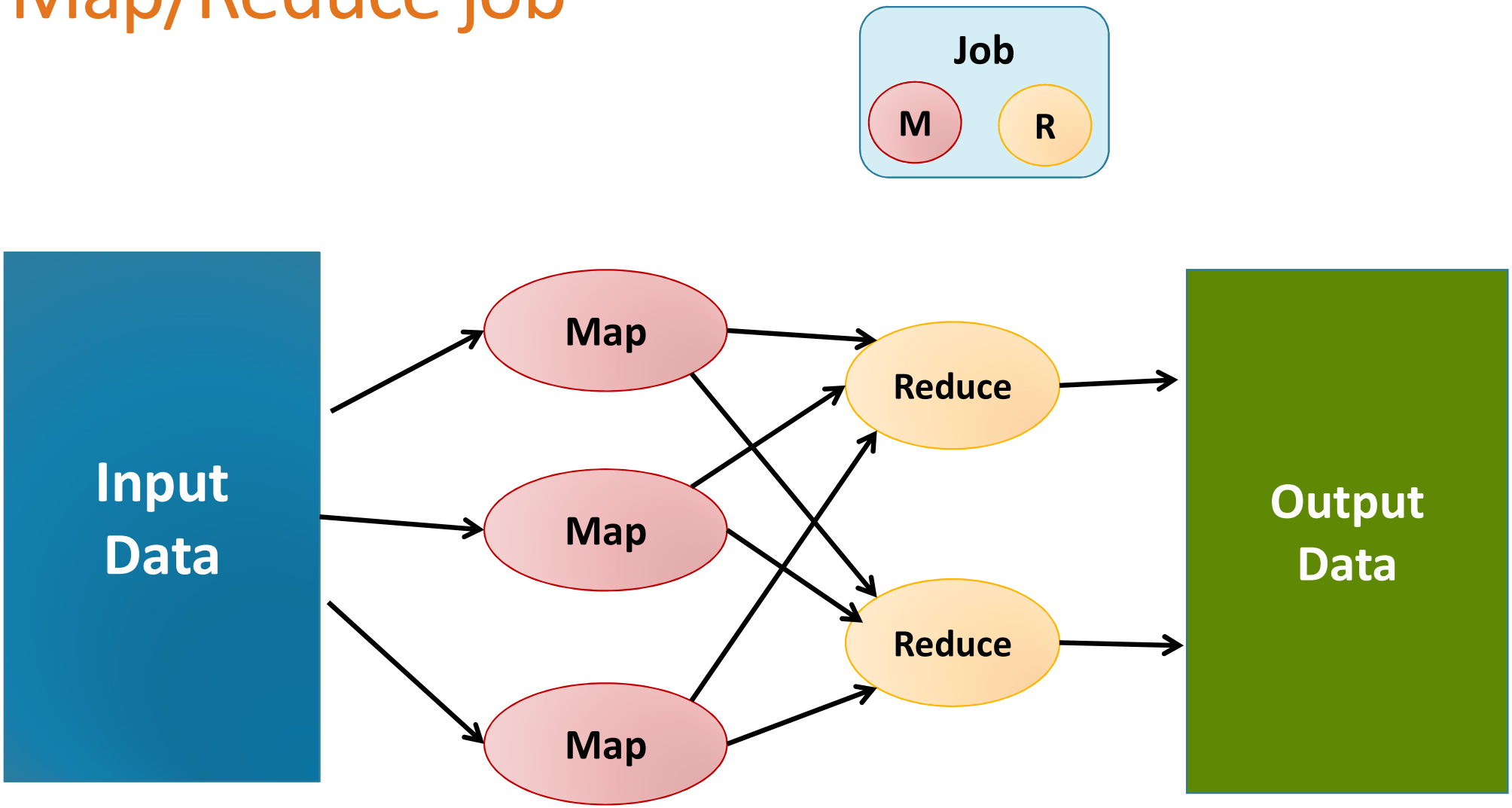
The Apache Hadoop project

- The brainchild of Doug Cutting (Yahoo)
- Open source project hosted at Apache
- Started in 2007 when code was spun out of Nutch
- Has grown into a large top-level project at Apache with significant ecosystem
  - V2 (YARN, 2013) structures it as a generic platform
  - Even third-party distros *a la* Linux (Cloudera, Hortonworks)

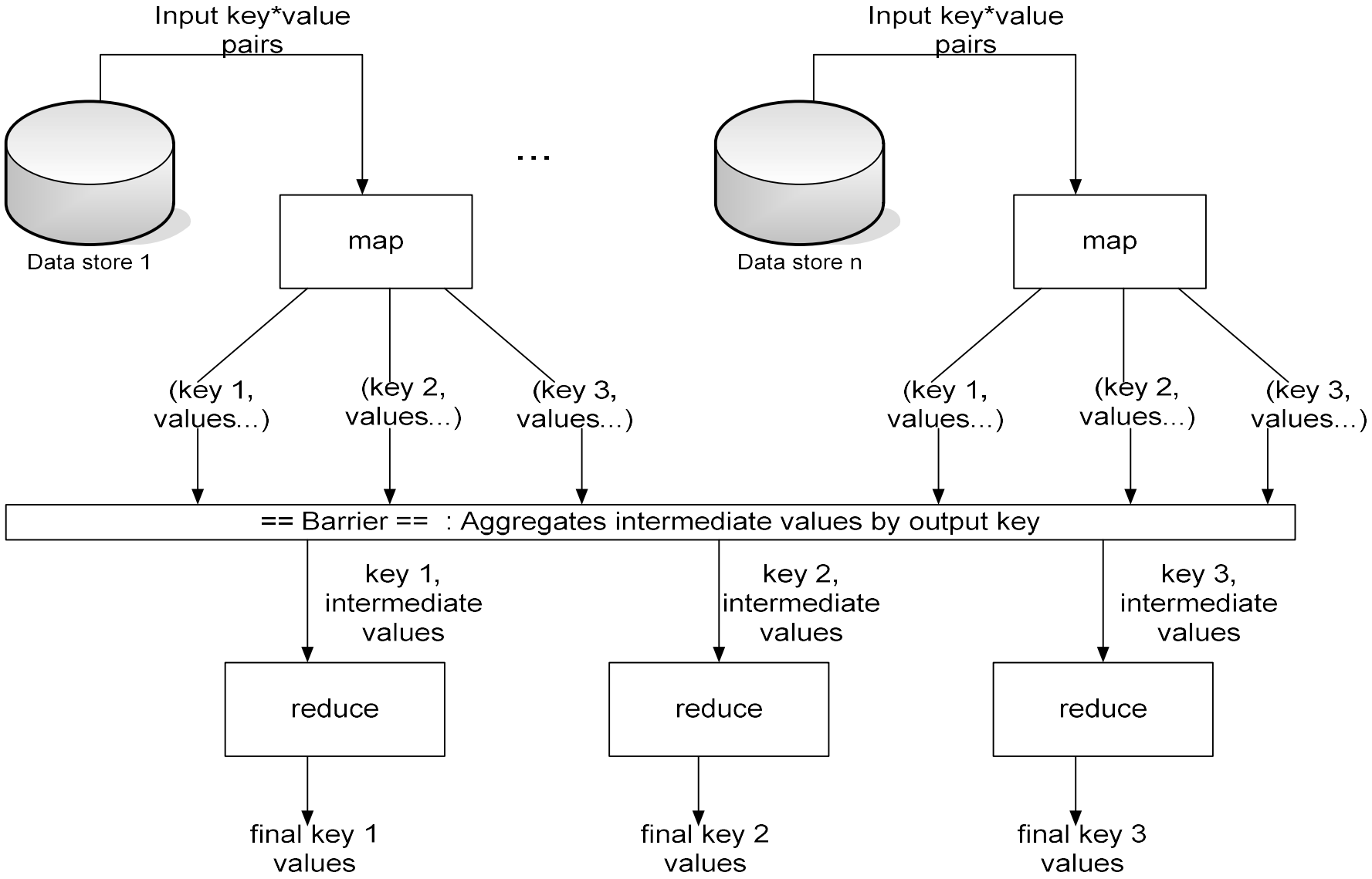
Hadoop Physical Requirements

- Designed to run in clusters of commodity PCs
  - Leverages heterogeneous capabilities
- Scales up to thousands of connected machines
- Suitable for Local Networks / DataCenters
  - Rack servers connected over a LAN
  - Clusters distributed over the Internet are not feasible
    - Network would become an enormous bottleneck

Map/Reduce job



Synchronization and message passing



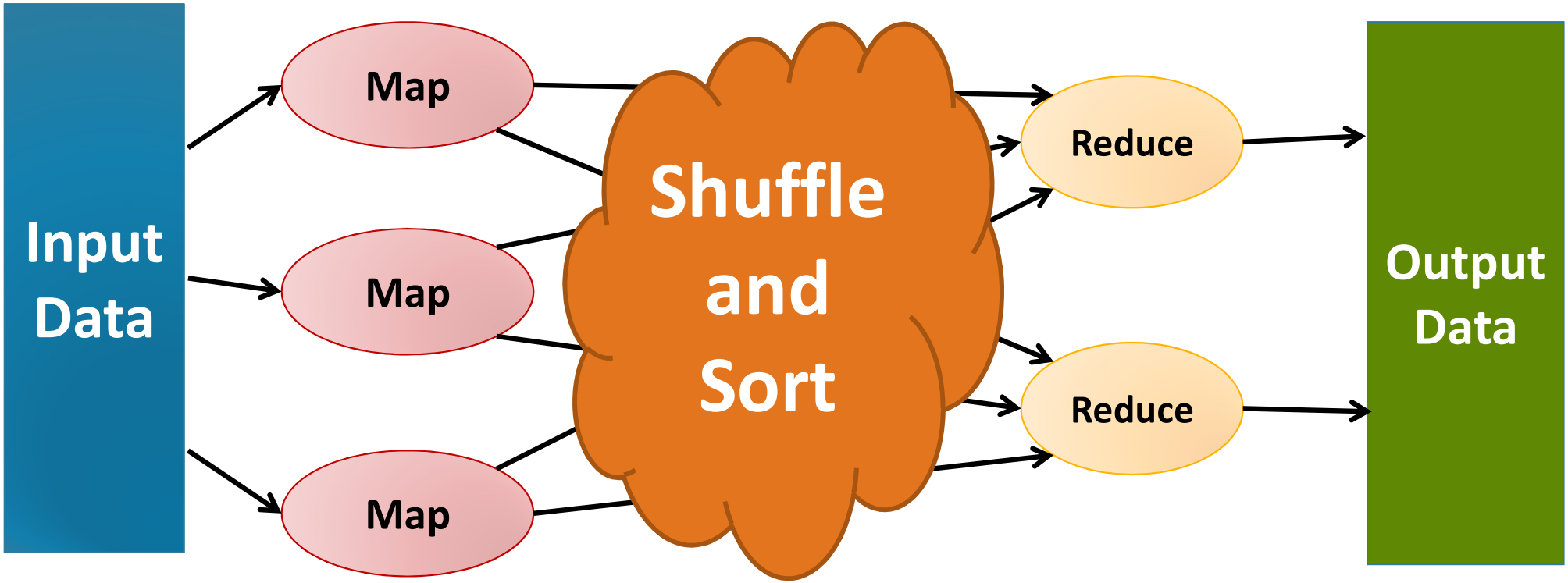
Hadoop job

- A Hadoop Job is packaged as a Jar file containing all the code for Mapper and Reducer functions
- The job is assigned a cluster-unique id
  - A set number of reattempts is managed for job tasks
- The file is replicated over the Hadoop nodes
  - Move computation to the data

Job execution: Complete MapReduce job flow

1. Split (logically) input data into computing chunks
2. Assign one chunk to a Map node
3. Run 1..* **Mappers**
4. Shuffle and Sort
5. Run 1..* **Reducers**
6. Results from Reducers create the job output

Shuffle and Sort



Shuffle and Sort – At each Mapper

1. All emitted Key-value pairs are **collected**
   - in-memory buffer (100MB default size), spills to HD
2. Key/Value Pairs are **partitioned** depending on target reducer
   - Partitioning aims at even split of keys
3. (Optionally) **Combiner** runs on each partition
4. Output is available to the Reducers through HTTP server threads

Shuffle and Sort – At each Reducer

1. The reducer **downloads** output from mappers
   - Potentially all Mappers are contacted
2. Partial values from each Mapper are **merged**
3. Keys are **sorted** and fed as input for the Reducer
   - List of (k, (v1,v2,v3,…)), sorted by k

HDFS

- HaDoop Distributed Filesystem
  - Shared storage among the nodes of the Hadoop cluster
- Storage for Input and output of MapReduce jobs
- HDFS is Tailored for MapReduce jobs
  - Large block size (128MB default nowadays)
    - But not too large, blocks define the minimum parallelization unit
  - HDFS is not a POSIX compliant Filesystem
    - Tradeoffs for improving data processing throughput

HDFS Data distribution

- Data distribution is a key element of the MapReduce model and architecture
- “Move computation to data” principle
- Blocks are replicated over the cluster
  - Default ratio is three times
  - Spread replicas among different physical locations
  - Improves reliability

Contents

- Apache Hadoop
- **The Combiner**
- MapReduce aggregate computation

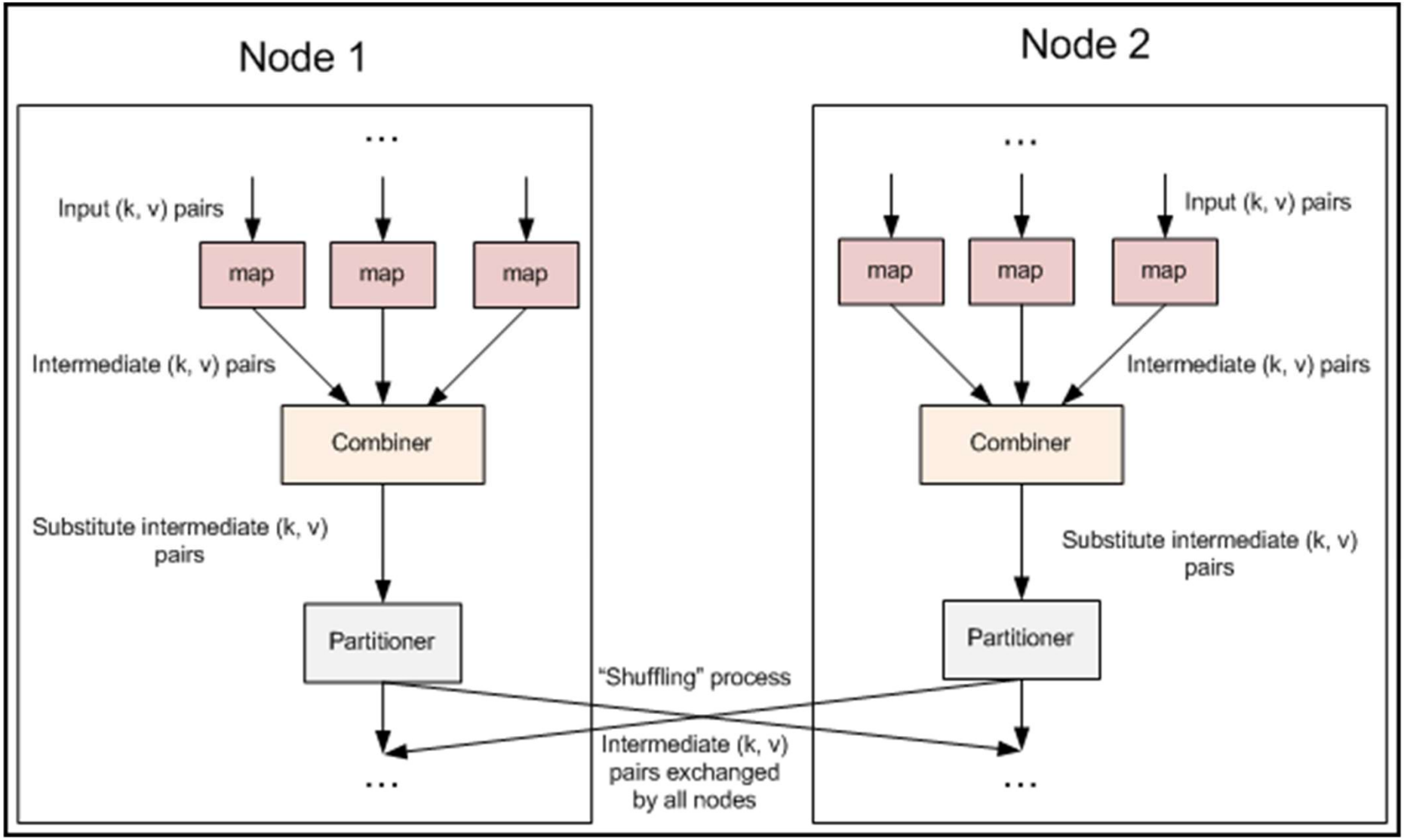
The cost of communications

- Parallelising Map and Reduce jobs allow algorithms to scale close to linearly
- One potential bottleneck for MapReduce programs is the cost of Shuffle and Sort operations
  - Data has to be copied over network communications
  - All the keys emitted by the mappers
  - Sorting large amounts of elements can be costly
- Combiner is an additional optional step that is executed before these steps

The Combiner

- The combiner acts as a preliminary reducer
- It is executed at each mapper node just before sending all the key value pairs for shuffling
- Reduces the number of emitted items
  - Improves efficiency
- It cannot be mandatory (the algorithm must work correctly if the combiner is not invoked)

Combiner


Node 1
Node 2
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner

Word count combiner

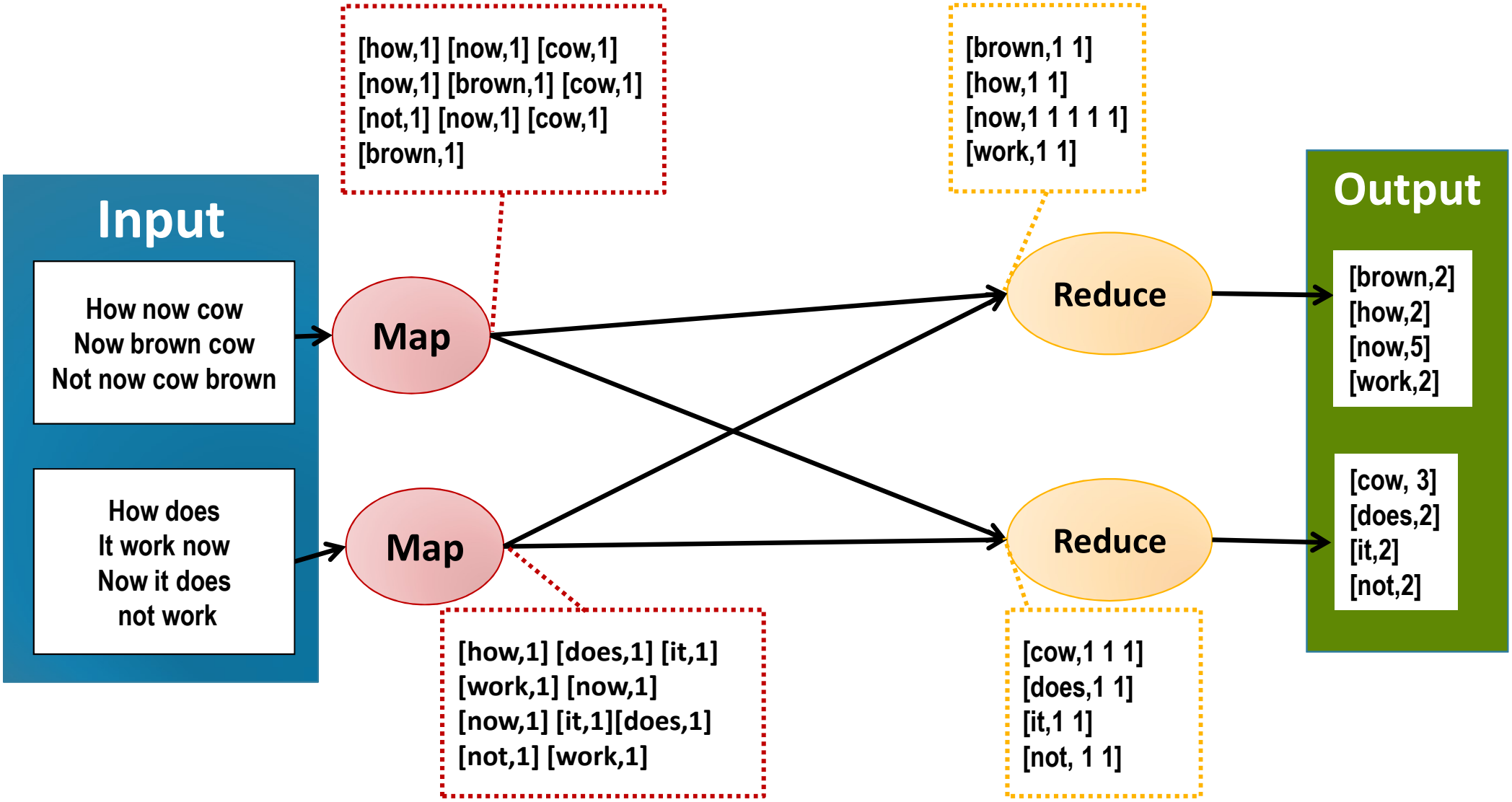
```
def mapper(_,text):
    words = text.split()
    for word in words:
        emit(word, 1)
def reducer(key, values):
    emit(key, sum(values)

def combiner(key, values):
    emit(key, sum(values)
```

Combiner rules

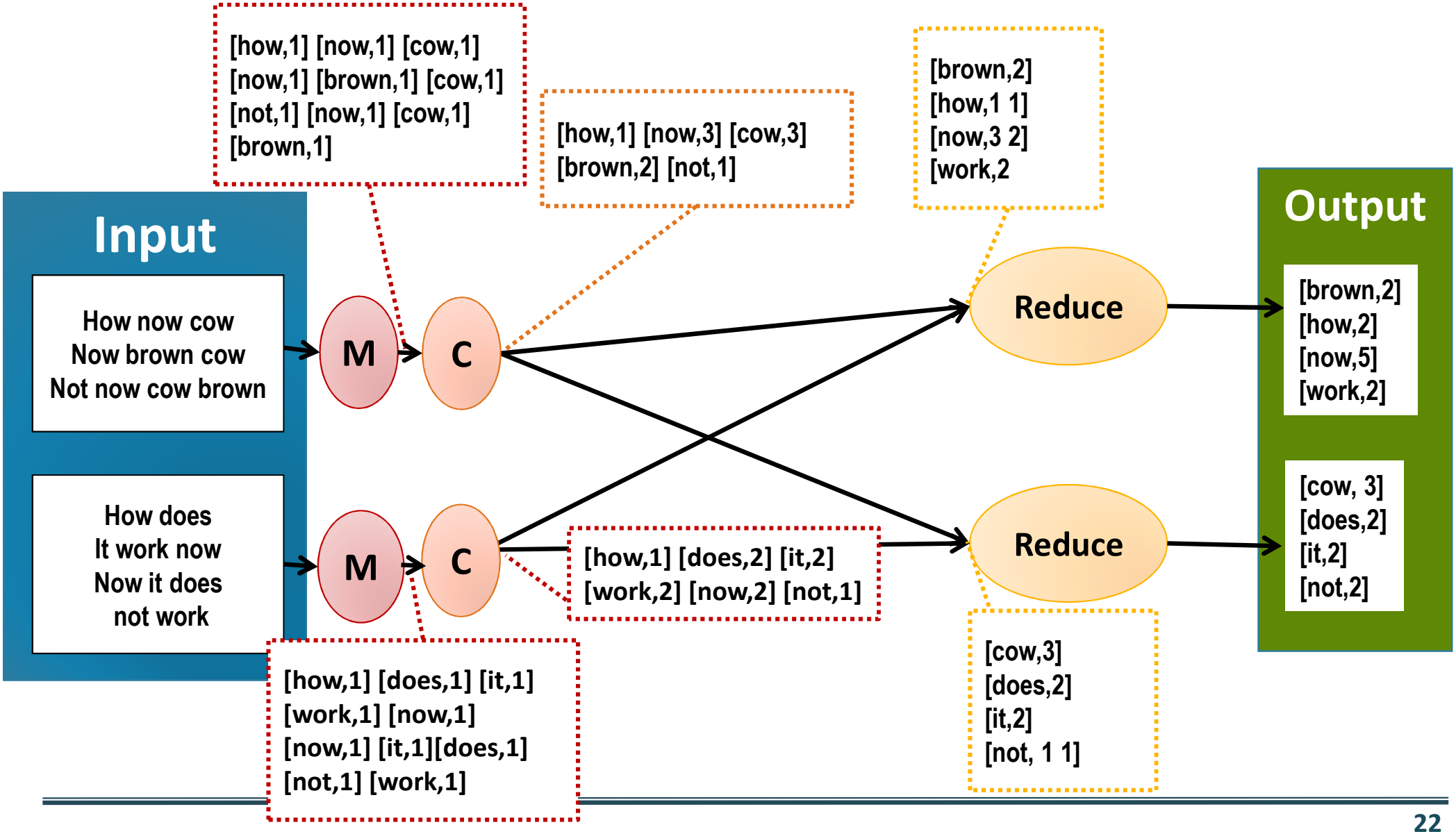
- The combiner has the same structure as the reducer (same signature) but must comply with these rules

1. Idempotent - The number of times the combiner is applied can't change the output
2. Transititive - The order of the inputs can't change the output
3. Side-effect free - Combiners can't have side effects (or they won't be idempotent).
4. Preserve the sort order - They can't change the keys to disrupt the sort order
5. Preserve the partitioning - They can't change the keys to change the partitioning to the Reducers

Word Count Example



Word Count Example with Combiner



Contents

- Apache Hadoop
- The Combiner
- **MapReduce aggregate computation**

Numerical Summarisation

- Goal: Calculate aggregate statistical values over a dataset
- Extract features from the dataset elements, compute the same function for each feature
- Examples:
  - Count occurrences
  - Maximum / minimum values
  - Average / median / standard deviation

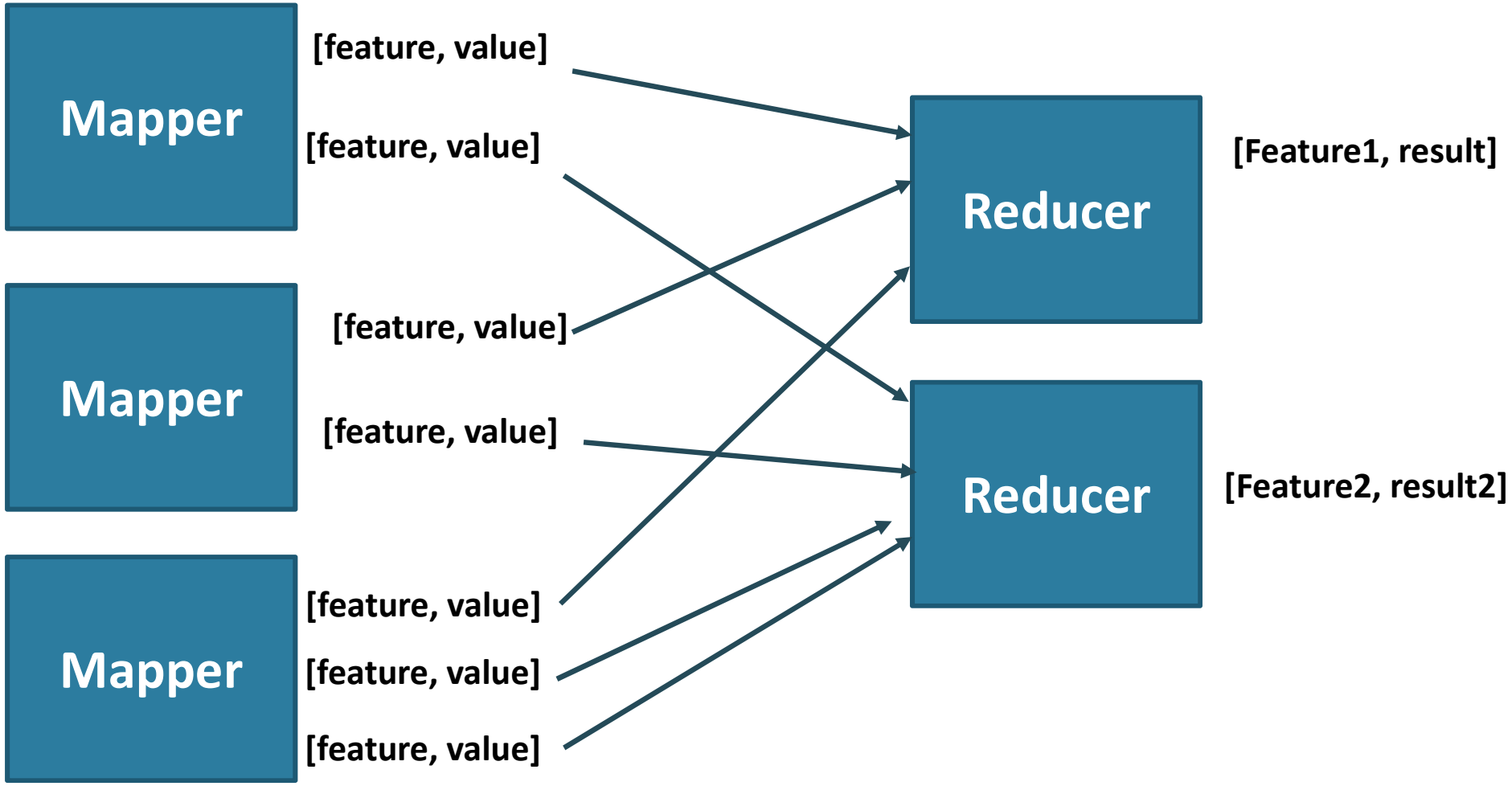
Sample dataset: China's Air Quality sensors

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 summarisation 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 Summarisation Structure



Writing Map and Reduce functions

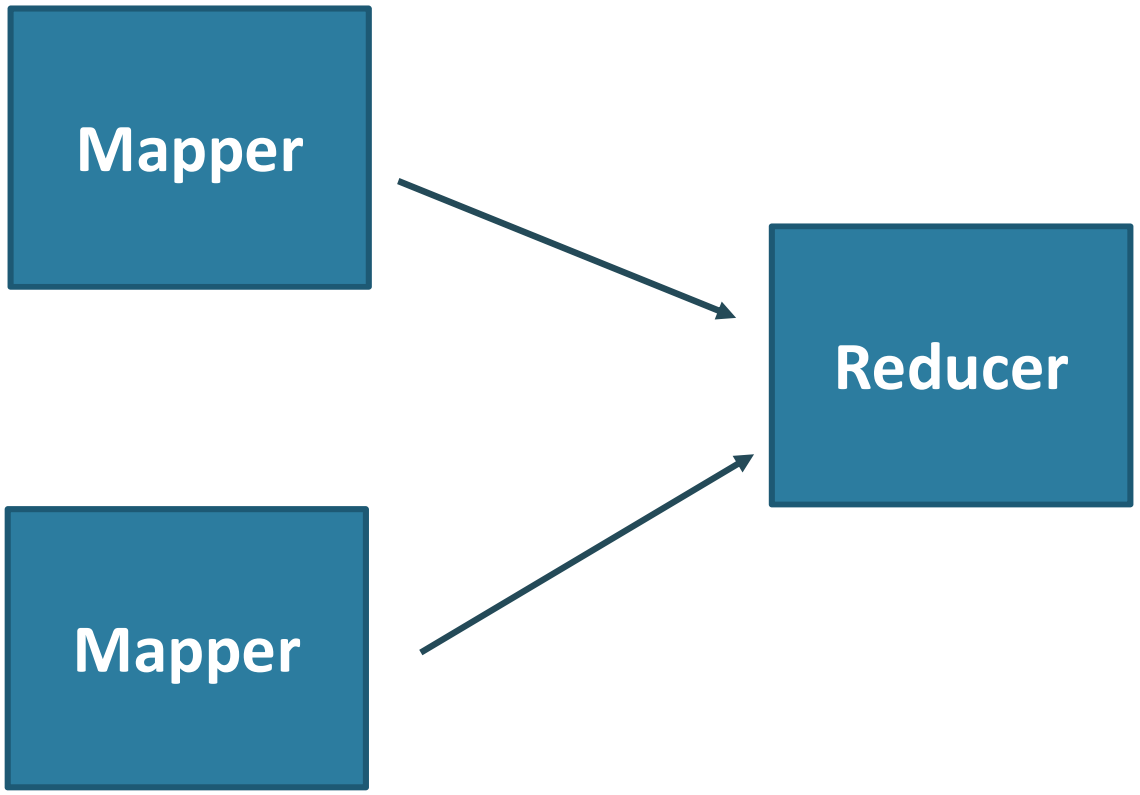
- Mapper
  - Find features in Input. For each one:
  - Emit Key = feature. Value = partial aggregate value
- Reducer
  - Compute final aggregate result from all the intermediate values for that feature
- Combiner?

Computing averages

Input: Row with moduleid, studentid, grade

Goal: Compute module average

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

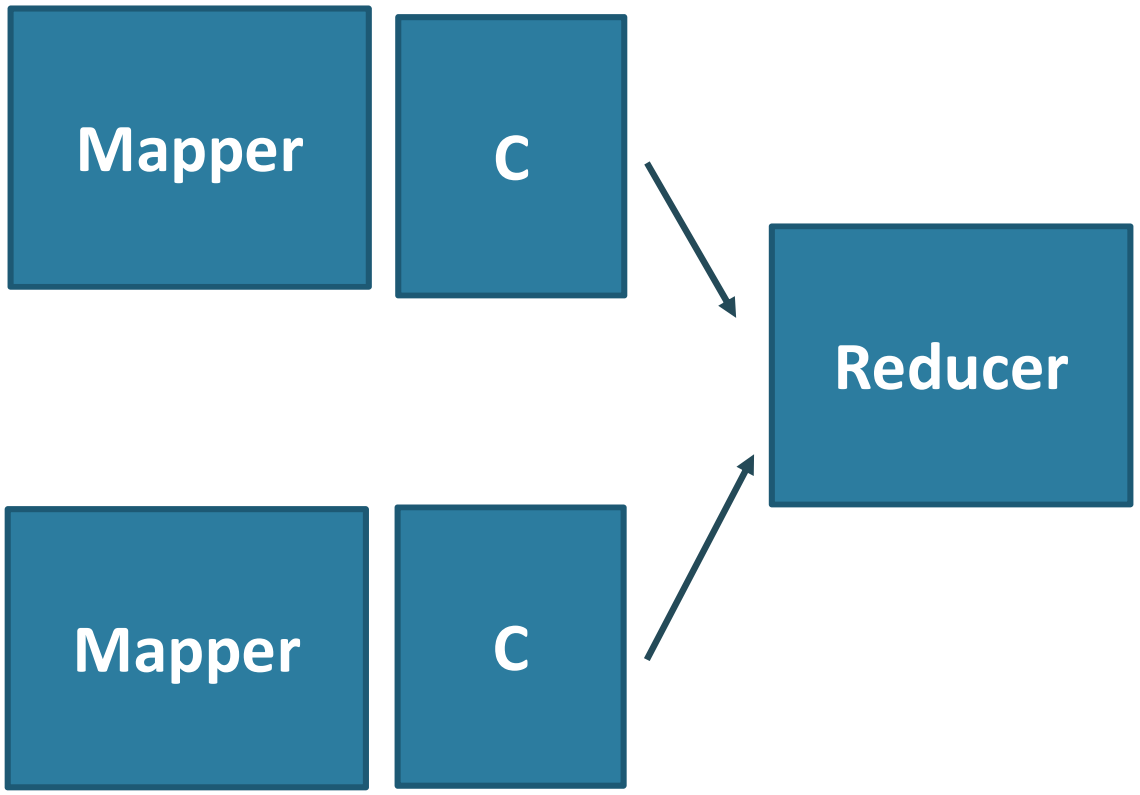


Computing averages

Input: Row with moduleid, studentid, grade

Goal: Compute module average

BigDataProc	ec03847293847	100
DataMining	ec29347298347	100
BigDataProc	ec23894283472	100
BigDataProc	ec23489209348	100
BigDataProc	ec23492834343	100
BigDataProc	ec34948758493	0
BigDataProc	ec56456456545	100
BigDataProc	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 mark values 100 100 20
    - Emit (100,1), (100,1), (20,1)
  - Combiner: adds aggregates and number of elements
    - Emits (220,3)
  - Reducer
    - Adds aggregates and computes average