Queen Mary University of London

INTRODUCTION TO PARALLELISM

BIG DATA PROCESSING

Félix Cuadrado

felix.cuadrado@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

Module organization

- Lectures: Fridays 9:00-11:00. Mason LT
- Labs:
  - (BSc) Mondays 9:00-11:00
  - (MSc) Wednesdays 16:00-18:00
  - (BSc DAs) Fridays 11:00-13:00
- Lecturers
  - Felix Cuadrado, Ben Steer
- Demonstrators:
  - Bingqing Guo, Zico Pratama, Mosab Bazargani, Xindi Zhang

Module Assessment

- Exam: 65%
  - 4 questions, short answers, practical exercises
- Coursework: 35%
  - Lab quizzes: 15%
    - 5 quizzes, 3% each
    - Published week of labs, deadline 2 weeks after.
  - Individual coursework: 20%
    - Deadline: Week 10 (End of november)

Contents

- **Introduction to parallelism**
- Parallel applications
- The Map/Reduce Programming Model

Parallel Computing

- The use of a number of processors, working together, to perform a calculation or solve a problem.
- The calculation will be divided into tasks, sent to different processors.
  - **Processor coordination** will be required
- Processors can be different cores in the same machine, and/or different machines linked by a network

Why Parallelise?

- There are many problems we cannot solve by running them on a simple processor/machine. They are:
  - Too large (do not fit in one machine)
  - Take too long

Parallel computing can provide faster results, and it can even be cheaper

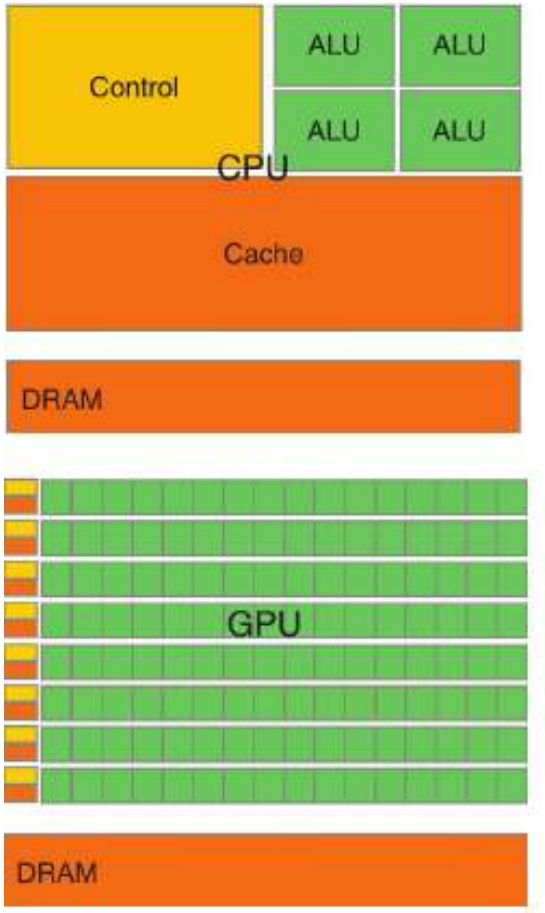
Sequential Program Execution

- Basic model based on Von Neumann architecture in the 40s
- One instruction is fetched, decoded and executed at a time
- As processor speed increases, more instructions can be executed in the same time

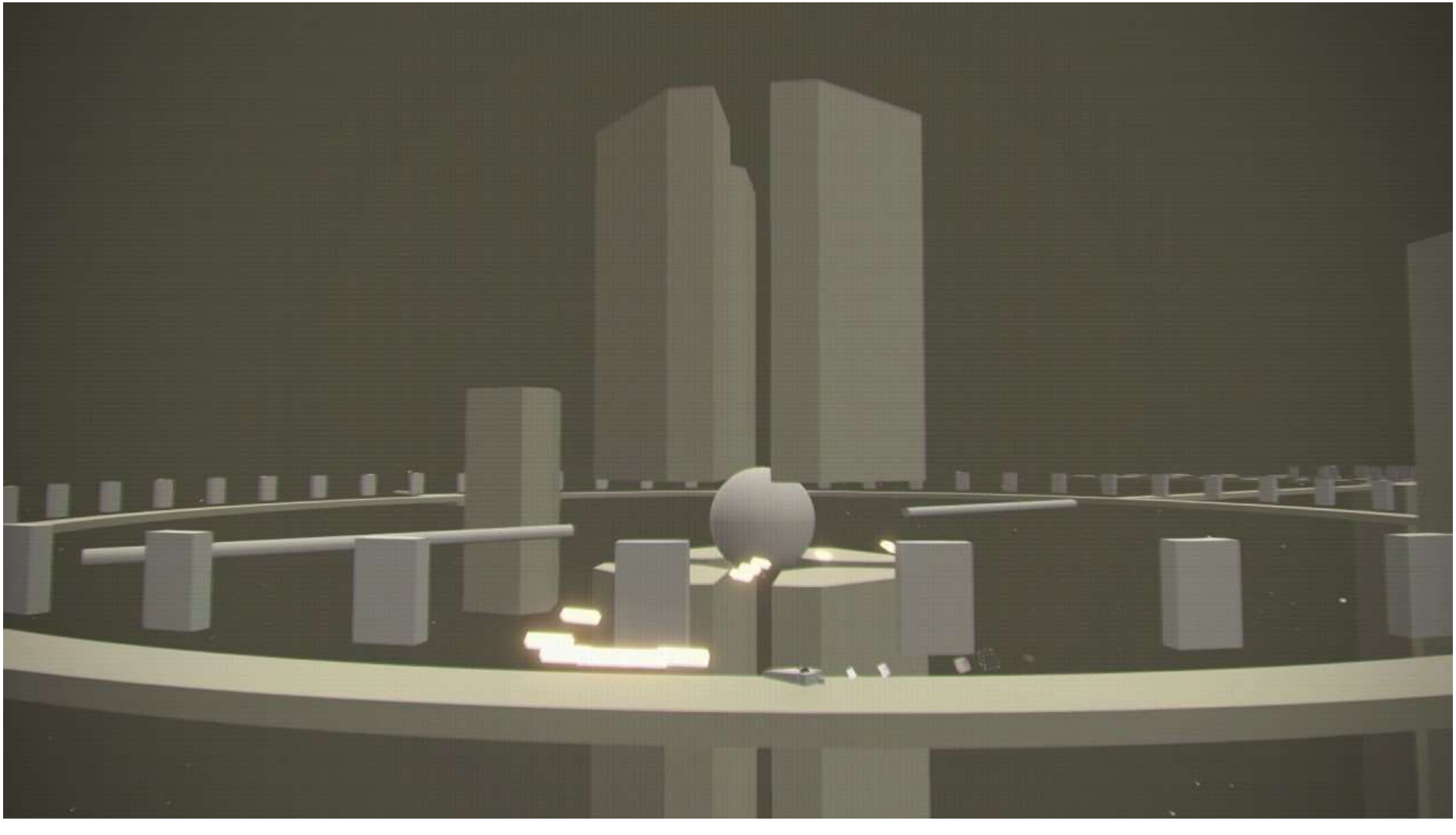
Single processor limitations

- We have roughly reached the practical limitations in the amount of computing power a single processor can have
  - We cannot make processors much faster, or much bigger
- According to Moore's law the number of transistors per chip will continue to increase
- But this means now we have chips containing an increasing amount of processors
  - Multicore chips

Machine clusters



Programming a cluster of machines



Parallel computing is hard

- Parallel Computing is generally very hard, because:
  - Many algorithms are hard to divided into subtasks (or cannot be divided at all)
  - The subtasks might use results from each other, so coordinating the different tasks might be difficult
- Some problem areas are much easier than others to parallelise

Easy parallelisation

Image processing: sharpening an image

1. We have a big photograph
2. We divide it into tiles (square patches), and sharpen each tile
3. Then we adjust the pixels at the edges so that they match up

This works because edge pixels are a small part of the total, and changing them does not affect the other pixels

Not so easy parallelisation

- Path search: Get best route from A to B
- How to divide the task in lesser ones? Find intermediate points
- How to select these intermediate points?
- Need to communicate among processes, what are the intermediate results
- How can we guarantee that the solution is the best one?

Contents

- Introduction to parallelism
- **Parallel applications**
- The Map/Reduce Programming Model

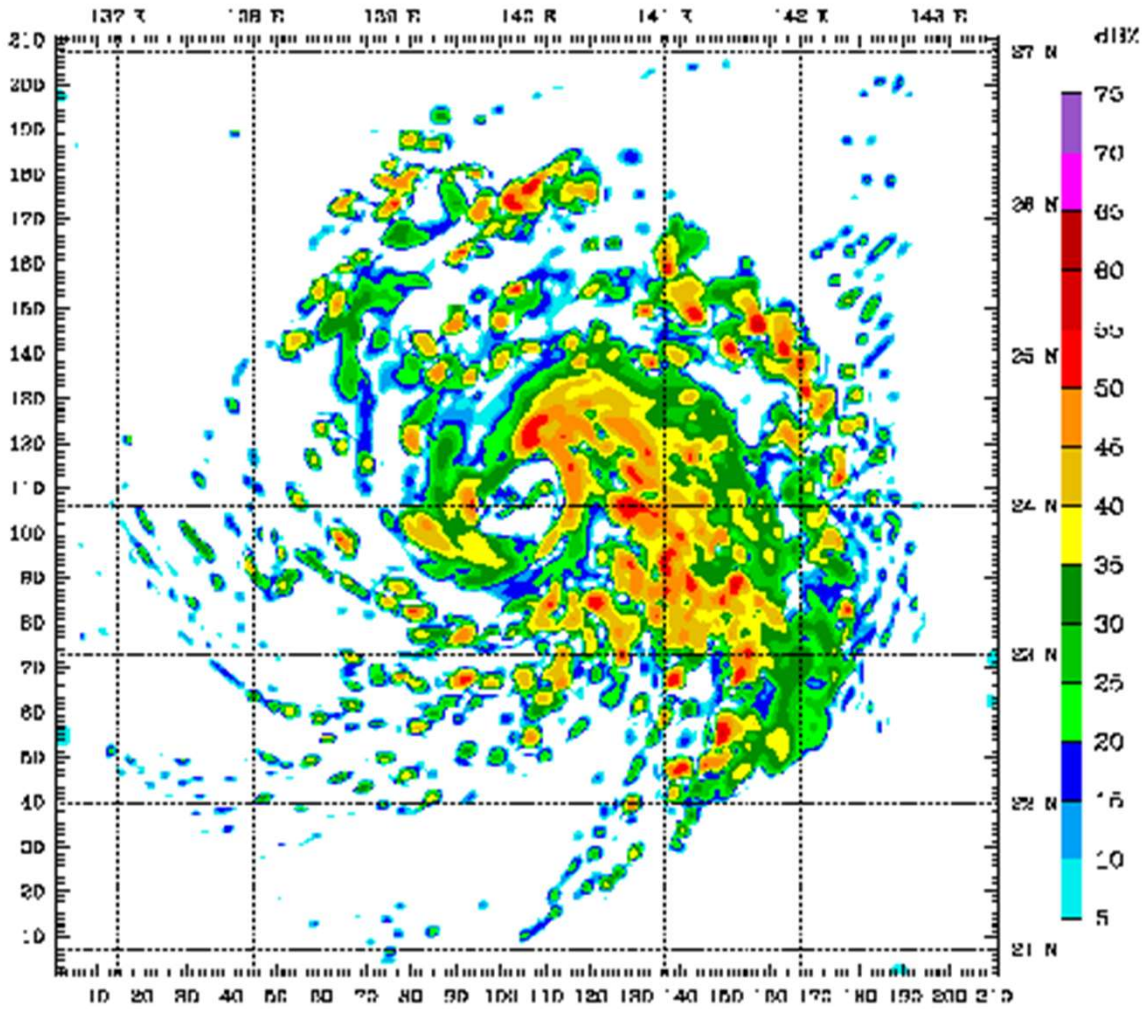
Applications of parallelism: Simulation

We often need to simulate physical events:

- What happens when lightning strikes a plane?
- This is important with modern planes, because they are made of plastic, and lightning goes through them (whereas it goes round metal planes)
- Planes are too expensive to conduct a large number of tests with them, but we have to know that they are safe to fly
- so simulation is a necessity: but it is computationally difficult

There are a lot of similar tasks (nuclear safety, for example) where direct testing is impossible or difficult

Weather prediction



Applications of parallelism: Prediction

- We would like to predict real-world events (such as the weather)
- This often involves very long and data intensive calculations, just like simulation does
- Challenging both for algorithm design and implementation

Applications: Data analysis

- Marketers analyze large amounts of data from Twitter in order to find out how their products are doing
- Google analyzes lots of web pages in order to support google search
- The LHC produces huge amounts of data, which need analysis
- Biologists read large quantities of DNA, and they have to work out what it means
- All these applications are computationally demanding, and they also use large amounts of data ("Big Data")

Data analysis (cont)

- Take Netflix and its approach to applying data analysis for maximizing the effectiveness of their video catalog.
- Traditionally, there is a limited subset of movie genres (let's say 100).
- Netflix manages more than 70,000 genres, and combines them with collected data about the millions of users that actively consume content through the service.
- With that information, they can provide recommendations considerably more effective than the previous standard
- http://www.theatlantic.com/technology/archive/2014/01/how-netflix-reverse-engineered-hollywood/282679/

Conclusions

- We now have access to massive quantities of data
  - Social networking and web/mobile apps traces
  - Automated data gathering (Sequencers in biology, particle accelerators in physics, automated cameras on astronomy)
- The data needs analysis for interpreting it
- This is the new “Big Data” market for parallel computing

Contents

- Introduction to parallelism
- Parallel applications
- **The Map/Reduce Programming Model**

Our first parallel program

- Task: count the number of occurrences of each word in one document
- Input: text document
- Output: sequence of: word, count

```
The      56
School   23
Queen    10
…
```

Program Input

QMUL has been ranked 9th among multi-faculty institutions in the UK, according to tables published today in the Times Higher Education.

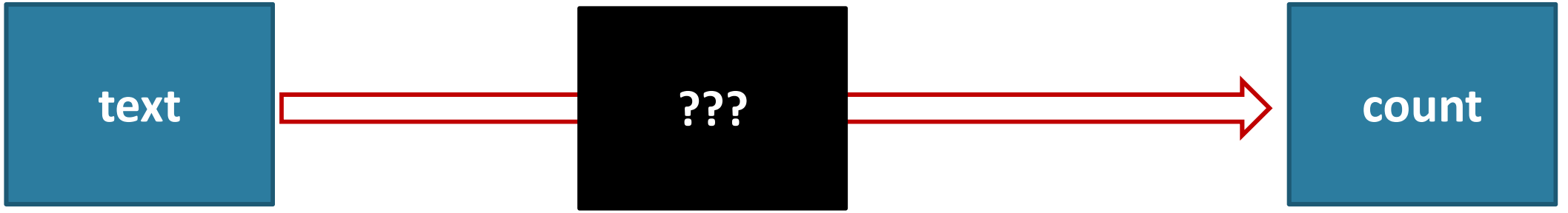
A total of 154 institutions were submitted for the exercise.

The 2008 RAE confirmed Queen Mary to be one of the rising stars of the UK research environment and the REF 2014 shows that this upward trajectory has been maintained.

Professor Simon Gaskell, President and Principal of Queen Mary, said: “This is an outstanding result for Queen Mary. We have built upon the progress that was evidenced by the last assessment exercise and have now clearly cemented our position as one of the UK’s foremost research-led universities. This achievement is derived from the talent and hard work of our academic staff in all disciplines, and the colleagues who support them.”

The Research Excellence Framework (REF) is the system for assessing the quality of research in UK higher education institutions. Universities submit their work across 36 panels of assessment. Research is judged according to quality of output (65 per cent), environment (15 per cent) and, for the first time, the impact of research (20 per cent).

How to solve the problem?



QMUL has been ranked 9th among multi-faculty institutions in the UK, according to tables published today in the Times Higher Education.

A total of 154 institutions were submitted for the exercise.

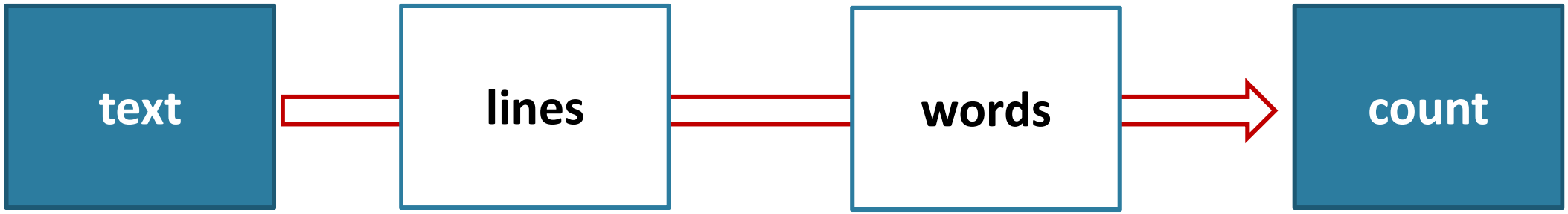
The 2008 RAE confirmed Queen Mary to be one of the rising stars of the UK research environment and the REF 2014 shows that this upward trajectory has been maintained.

Professor Simon Gaskell, President and Principal of Queen Mary, said: "This is an outstanding result for Queen Mary. We have built upon the progress that was evidenced by the last assessment exercise and have now clearly cemented our position as one of the UK's foremost research-led universities. This achievement is derived from the talent and hard work of our academic staff in all disciplines, and the colleagues who support them."

The Research Excellence Framework (REF) is the system for assessing the quality of research in UK higher education institutions. Universities submit their work across 36 panels of assessment. Research is judged according to quality of output (65 per cent), environment (15 per cent) and, for the first time, the impact of research (20 per cent).

```
The      56
School   23
Queen    10
….
```

How to solve the problem?



How to solve the problem on a single processor?

```
#input:text string with the complete text
words = text.split()
count = dict()
for word in words:
    if word in count:
        count[word] = count[word] + 1
    else:
        count[word] = 1
```

How to solve the problem on a single processor? Java version

```
//input: text String with the complete text
List[String] words = text.split();
Hashtable[String,Integer] count = new Hashtable();
for (String word: words){
  if(count.containsKey(word){
    count.put(word, count.get(word)+1);
  }
  else{
    count.put(word,1)
  }
}
```

Parallelizing the problem

- Splitting the load on subtasks:
  - Split sentences/lines into words
  - Count all the occurrences of each word
- …what do we do with the intermediate results?
  - Merge into single collection
  - Possibly requires parallelism too

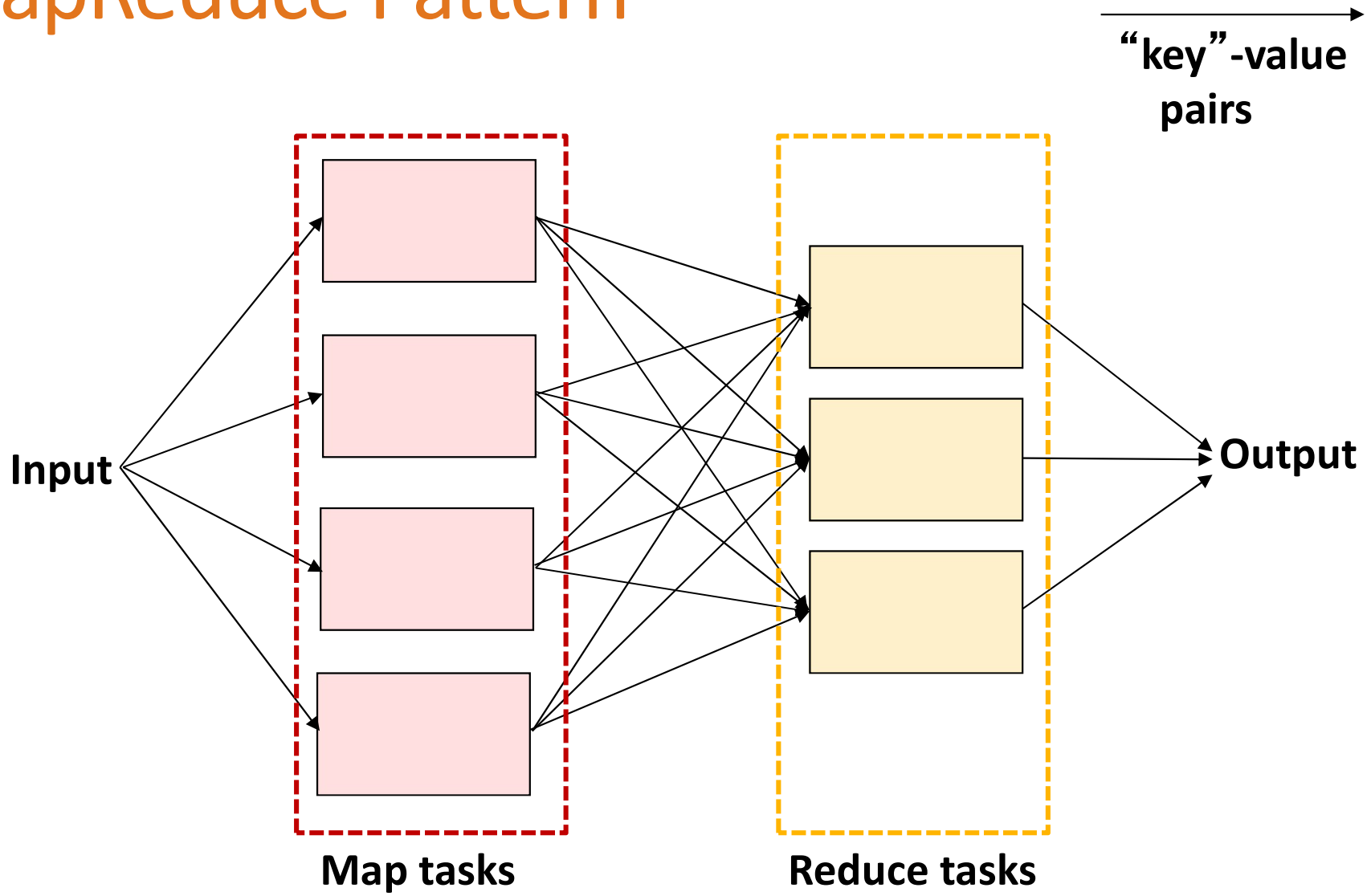
MapReduce

- *"A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."*

  Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters", Google Inc.

- More simply, MapReduce is:
  - A parallel programming model and associated implementation.

MapReduce Pattern



MapReduce Programming Model

- Process data using special **map**() and **reduce**() functions
  - The **map**() function is called on every item in the **input** and **emits** a series of **intermediate key/value** pairs
  - All the emitted **values** for a given key **are grouped** together
  - The **reduce**() function is called on every unique key, and the collected values. Emits a partial result bthat is added to the **output**

Example: word count (Pythonish pseudocode)

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

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

Example: word count (Javaish pseudocode)

```
public void Map (String filename,
                                    String text) {

  List[String] words= text.split();
  for (String word: words){
    emit(word, 1)
    }

}
```

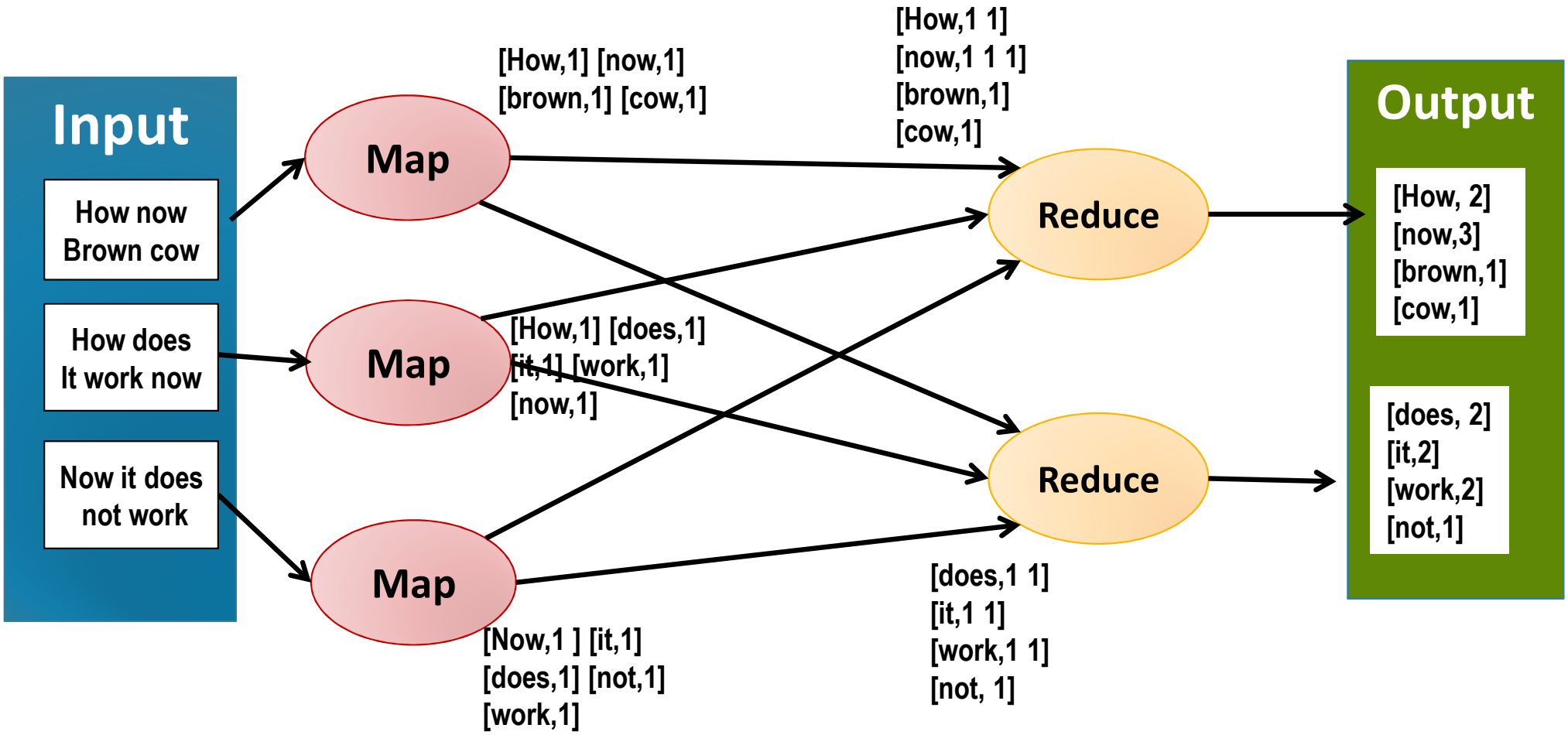
Example: word count (Javaish pseudocode)

```
public void Reduce (String key,
                    List[Integer] values) {
  int sum = 0;
  for (Integer count: values){
    sum+=count;
    }
  emit(key, sum);
}
```

How MapReduce parallelises

- Input data is partitioned into processable chunks
- One Map job is executed per chunk
  - All can be parallelised (depends on number of nodes)
- One Reduce Job is executed for each distinct key emitted by the Mappers
  - All can be parallelised (partitioned ‘evenly’ among nodes)
- Computing nodes first work on Map jobs. After all have completed, a synchronization step occurs, and they start running Reduce jobs

Word Count Example



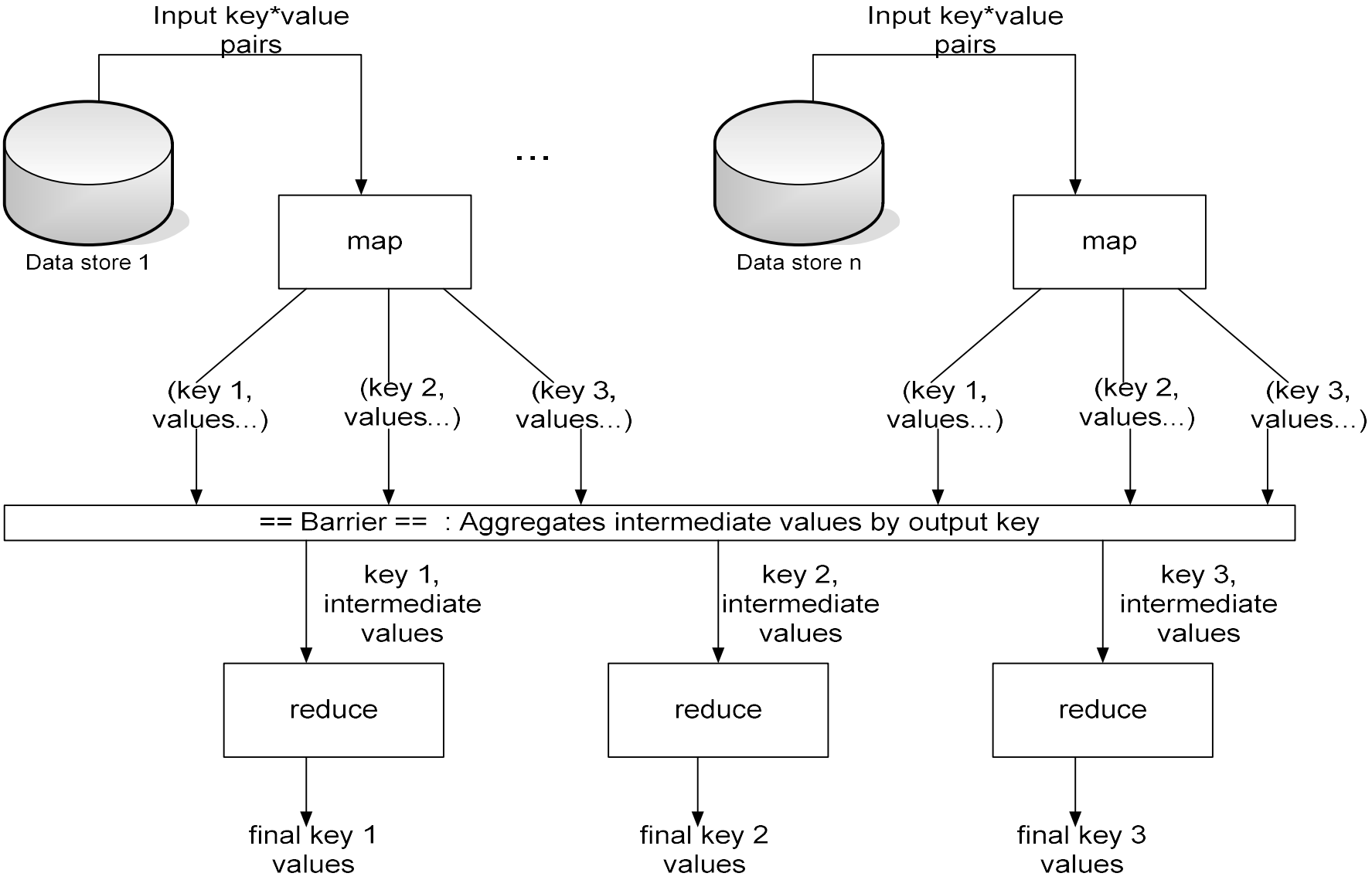
A Brief History

- Inspiration from functional programming (e.g., Lisp)
  - map() function
    - Applies a function to each **individual** value of a sequence to create a **new list** of values
    - Example: square x = x * x
      map square [1,2,3,4,5] returns [1,4,9,16,25]
  - reduce() function
    - Combines all elements of a sequence using a binary operator
    - Example: sum = (each elem in arr, total +=)
      reduce [1,2,3,4,5] returns 15 (the sum of the elements)

MapReduce Benefits

- High level parallel programming abstraction
- Framework implementations provide good performance results
- Scalability close to linear with increase in cluster size
- Greatly reduces parallel programming complexity
- However, it is not suitable for every parallel programming algorithm!

Synchronization and message passing



Shuffle and Sort steps

- Every key-value item generated by the mappers is collected
  - items are transferred over the network
- Same key items are grouped into a list of values
- Data is partitioned among the number of Reducers
- Data is copied over the network to each Reducer
- The data provided to each Reducer is sorted according to the keys

MapReduce Runtime System

- Partitions input data
- Schedules execution across a set of machines
- Handles load balancing
- Shuffles, partitions and sorts data between Map and Reduce steps
- Handles machine failure transparently
- Manages inter process communication