Multiprocessing and MapReduce

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

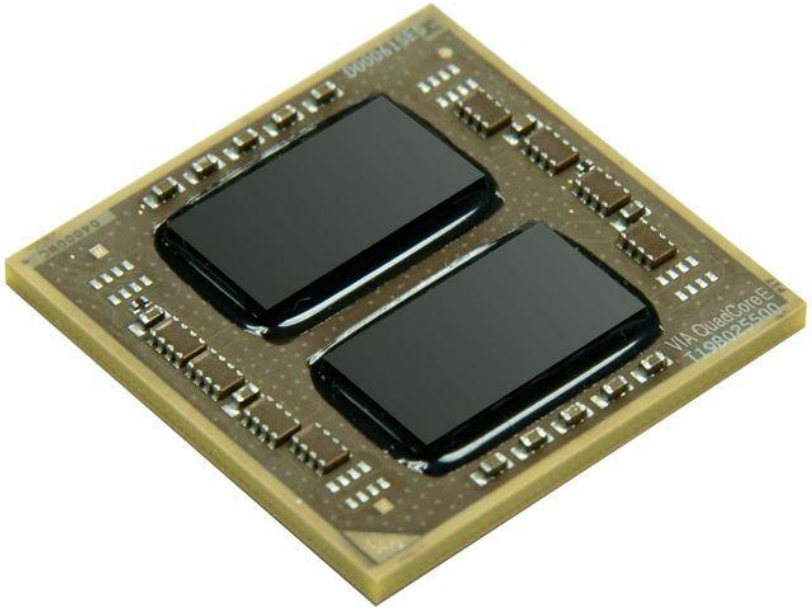
Announcements

- Exam on Friday
- Homework 5 check-in due Monday

Learning Objectives

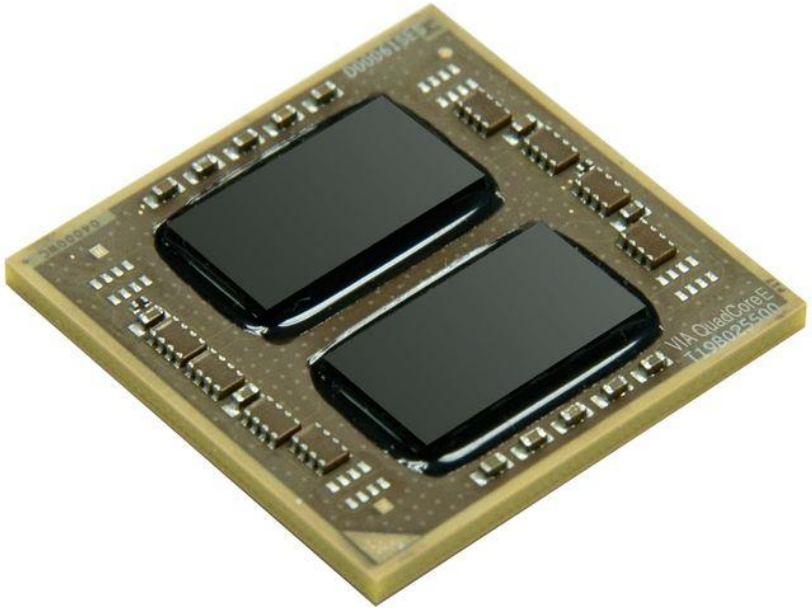
- To understand the benefits and challenges of multiprocessing and distributed systems
- To trace MapReduce algorithms on distributed systems and write small mapper and reducer functions

Computers today have multiple cores



Quad-core processor

Multiple Cores vs Multiple Processors



Quad-core processor

4-processor computer

Cores vs Processors

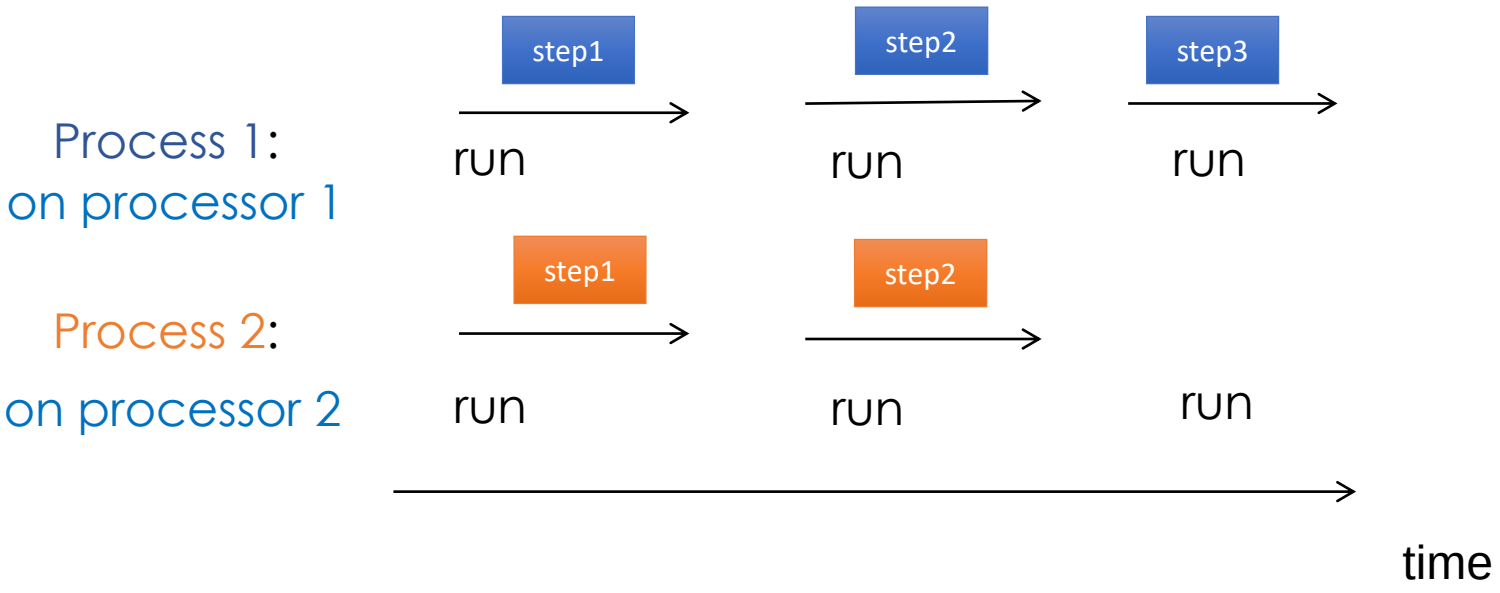
- Multiple cores share memory, faster to work together
- Multiple processors have their own memory, slower to share info
- For this class, let’s assume that these two are pretty much equal

How do you determine how to run programs?

Multi-processing is the term used to describe running many tasks across many cores or processors

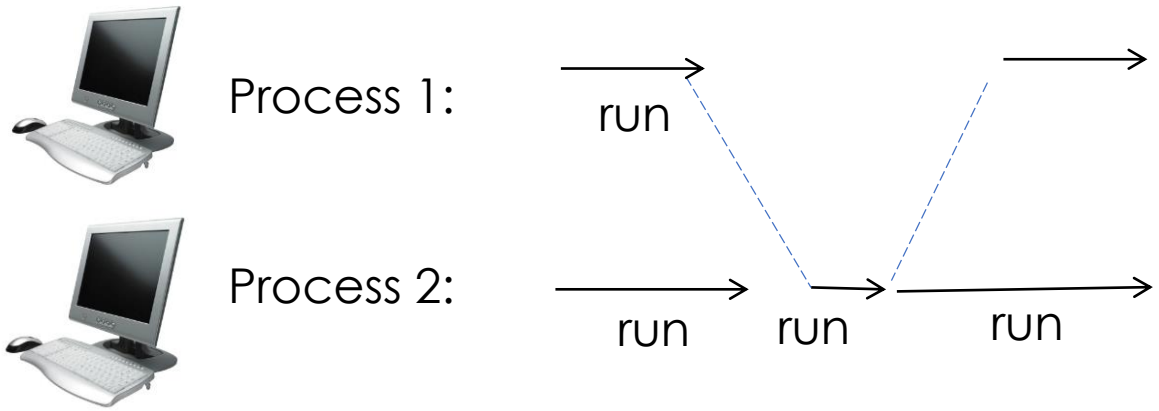
Multiple CPUs: Multiprocessing

If you have multiple CPUs, you may execute multiple processes in parallel (simultaneously) by running each on a different CPU.



Multiple Cores and Multiple Computers: Distributed Computing

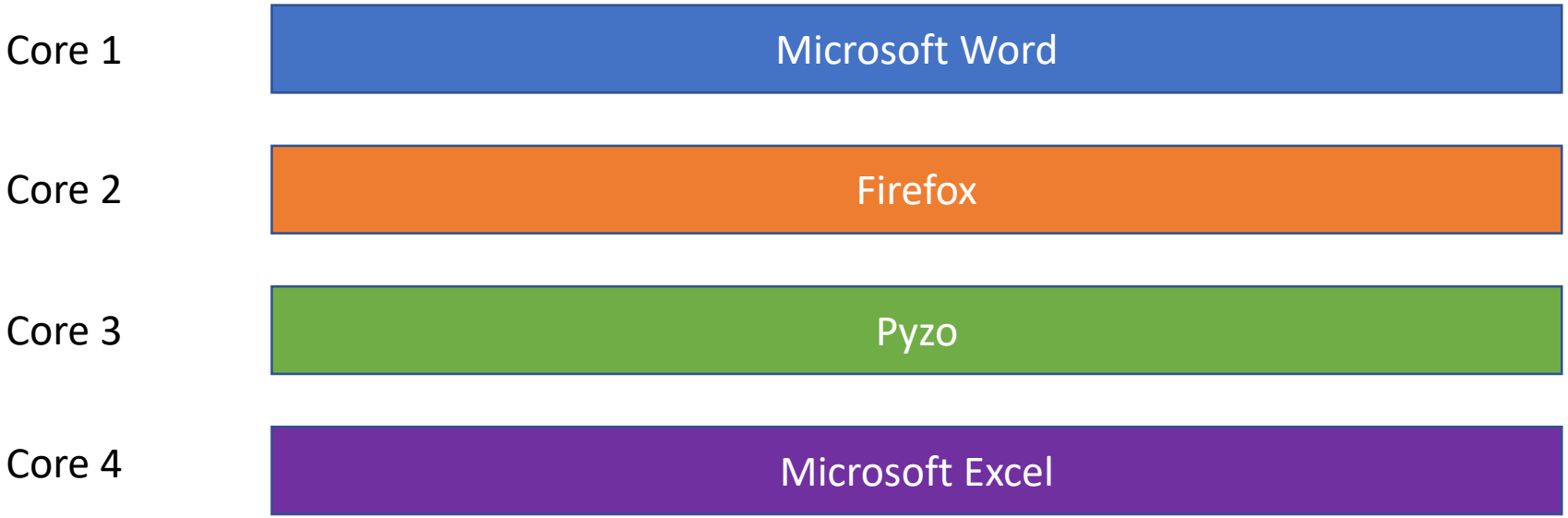
- If you have access to multiple machines, you can split the work up into many tasks and give each machine its own task
- The computers pass messages to each other to communicate information in order to put the tasks together



Multi-Processing

Run one task within each core

One task per core:



Multi-processing features

Just like multiple adders can run concurrently on a single core, multiple cores can all run concurrently

Multi-processing features

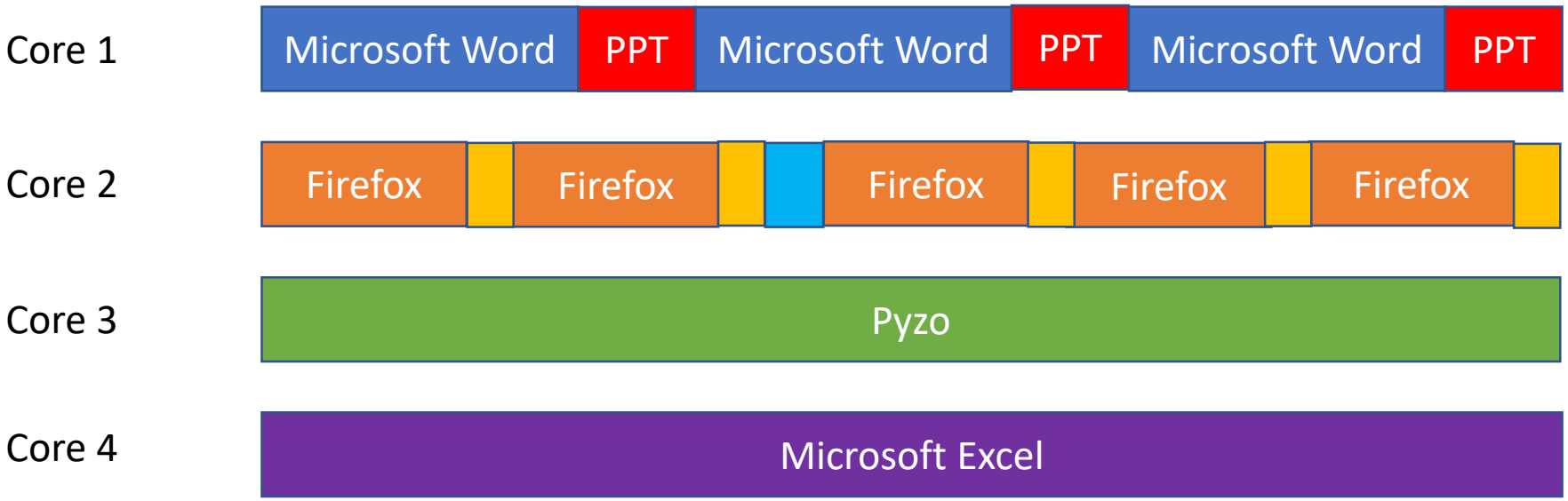
Just like multiple adders can run concurrently on a single core, multiple cores can all run concurrently

Just as single processors can multi-task, each core can multi-task

Multi-processing

Multi-processing allows a computer to run separate tasks within each core (how do you determine which tasks go on which core?)

Many tasks in a core (multitasking):



Multi-processing features

Just like multiple adders can run concurrently on a single processor, multiple cores/processors can all run concurrently

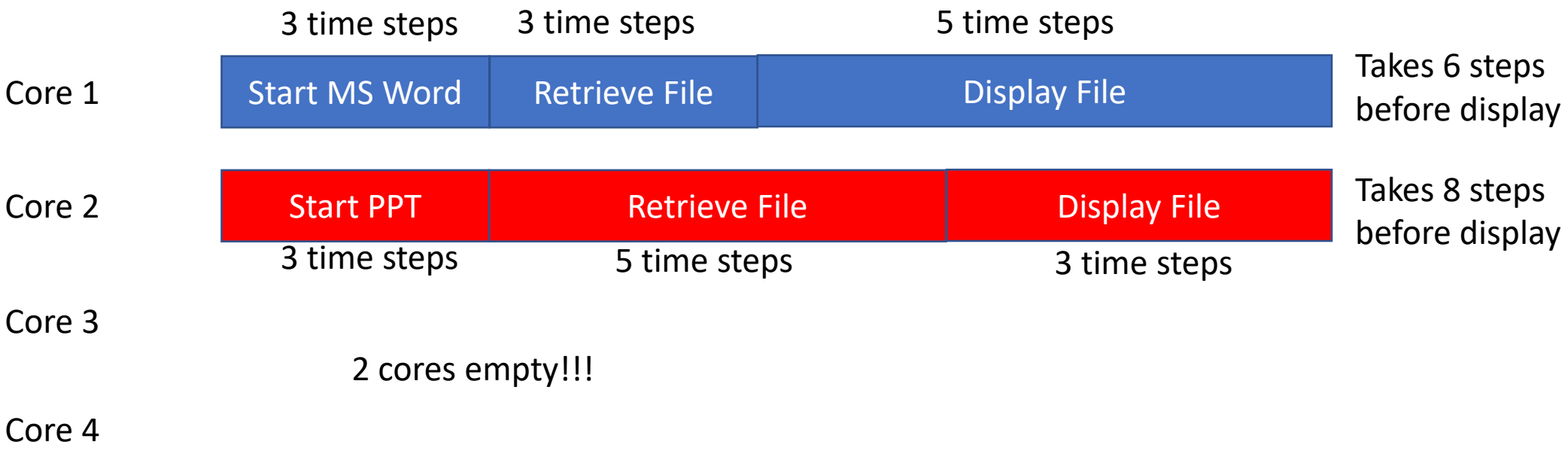
Just as single processors can multi-task, each core can multi-task

Just like a single processor with different circuits, we can pipeline tasks across processors

Multi-processing

Without pipelining on multiple cores

Leaves cores bored/not busy while taking extra time on one core

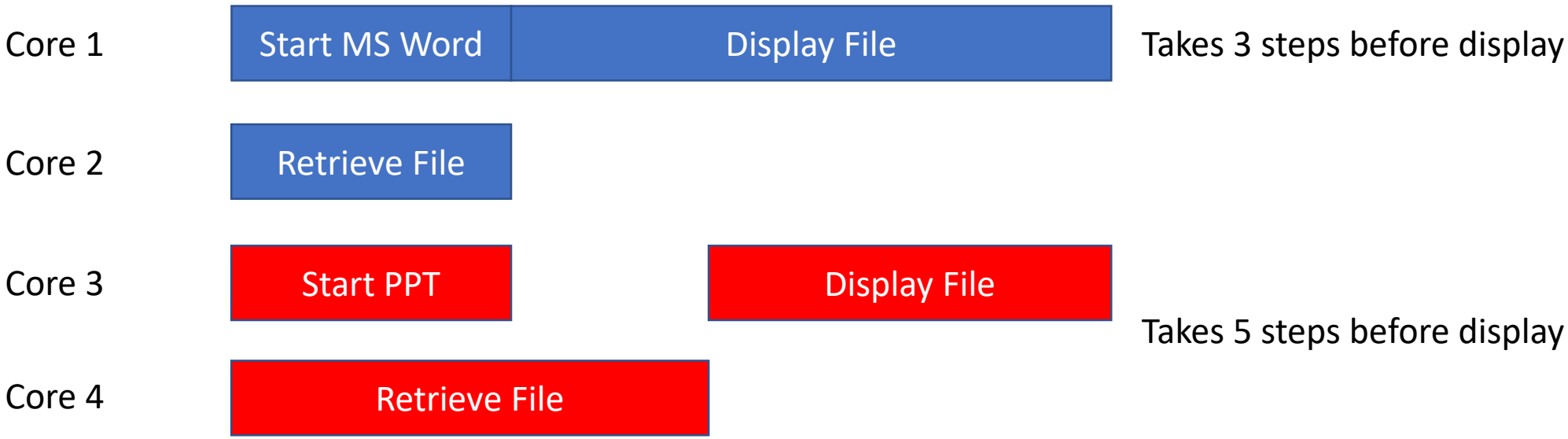


Multi-processing

With pipelining on multiple cores

Potentially takes less time to open programs, open data, etc

Requires that you send data between cores (expensive)



Writing Concurrent Programs

How can you write programs that can be split up and run concurrently?

Writing Concurrent Programs

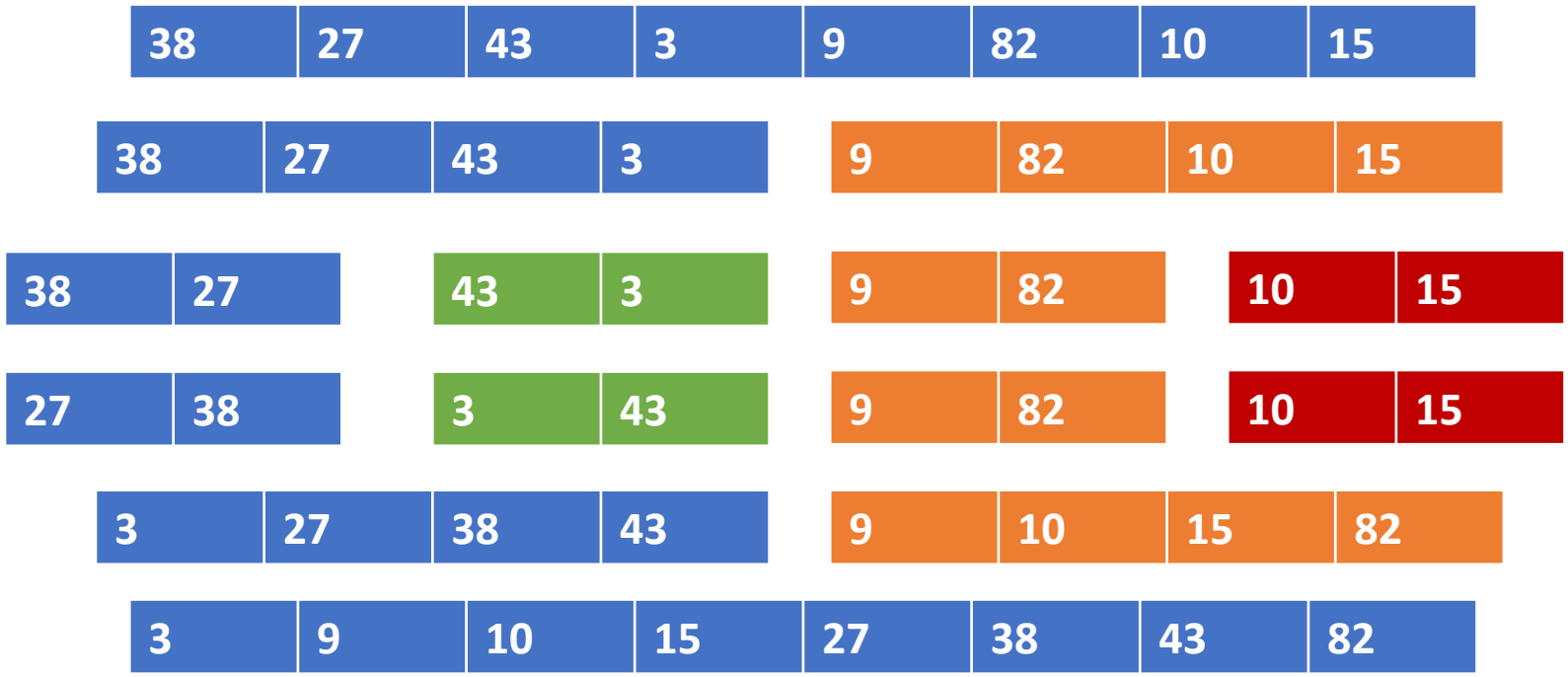
How can you write programs that can be split up and run concurrently?

Some are naturally split apart like mergesort (one color per core):

Writing Concurrent Programs

How can you write programs that can be split up and run concurrently?

Some are naturally split apart like mergesort (one color per core):



1 split, n moved items into 2 lists

2 splits, n moved items into 4 lists

2 splits, n moved items into 8 lists

4 sorts, n items moved

2 sorts, n items moved

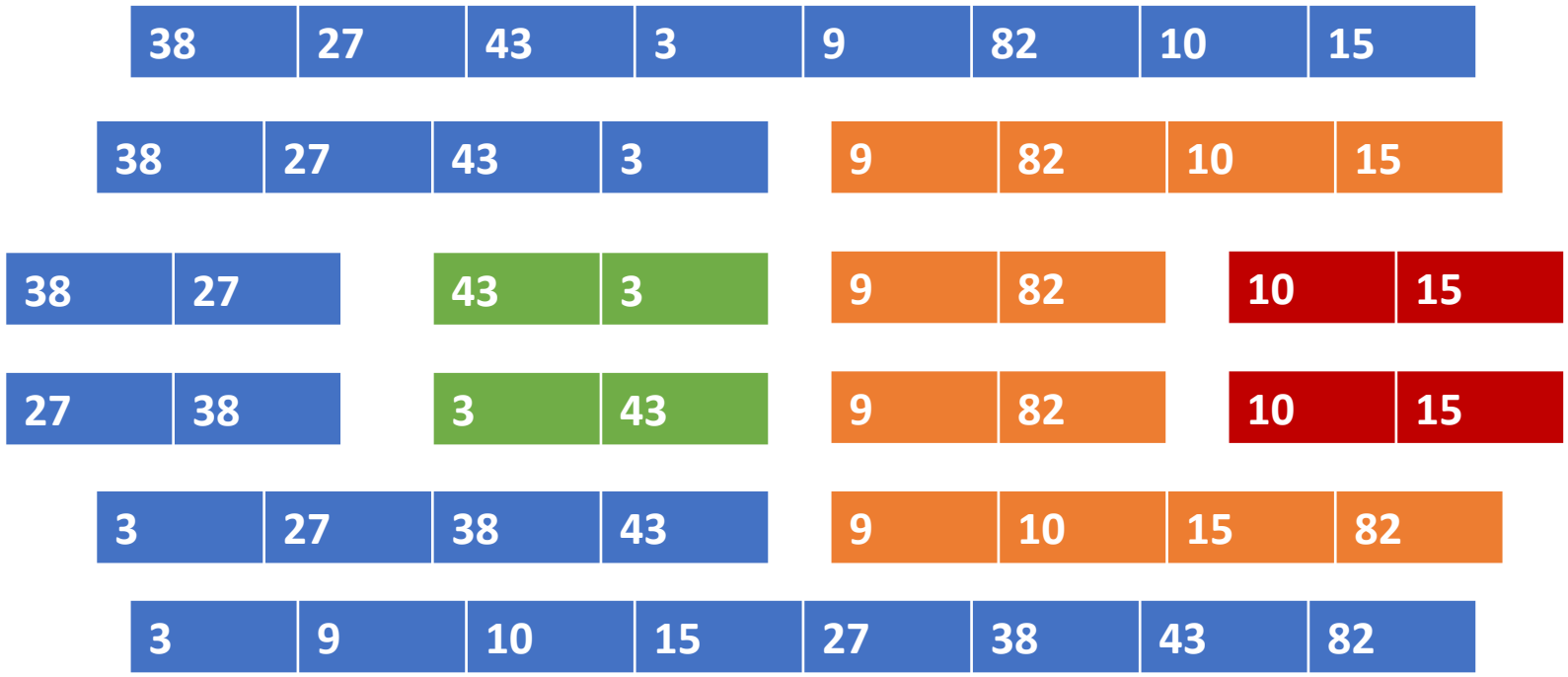
1 sort, n items moved

1 processor, n*2*log(n) moves

Writing Concurrent Programs

How can you write programs that can be split up and run concurrently?

Some are naturally split apart like mergesort (one color per core):



Each processor does n+(n/2)+(n/4)+… < 2n steps

Think About It

How could you parallelize a for loop? Can you do it in all for loops?

Think About It

How could you parallelize a for loop? Can you do it in all for loops?

```
for i in range(len(L)):
    print(L[i][0])
```

Pretty easy to parallelize

Each loop works on different data

```
for i in range(len(L)):
    L[i] = L[i-1]
```

Harder to parallelize

Each loop depends on the one before

Takeaways: Writing Concurrent Programs

How can you write programs that can be split up and run concurrently?

Some are naturally split apart like mergesort (one color per core)

Sometimes loops are also easy to split, but sometimes not

Many programs are not easy to split

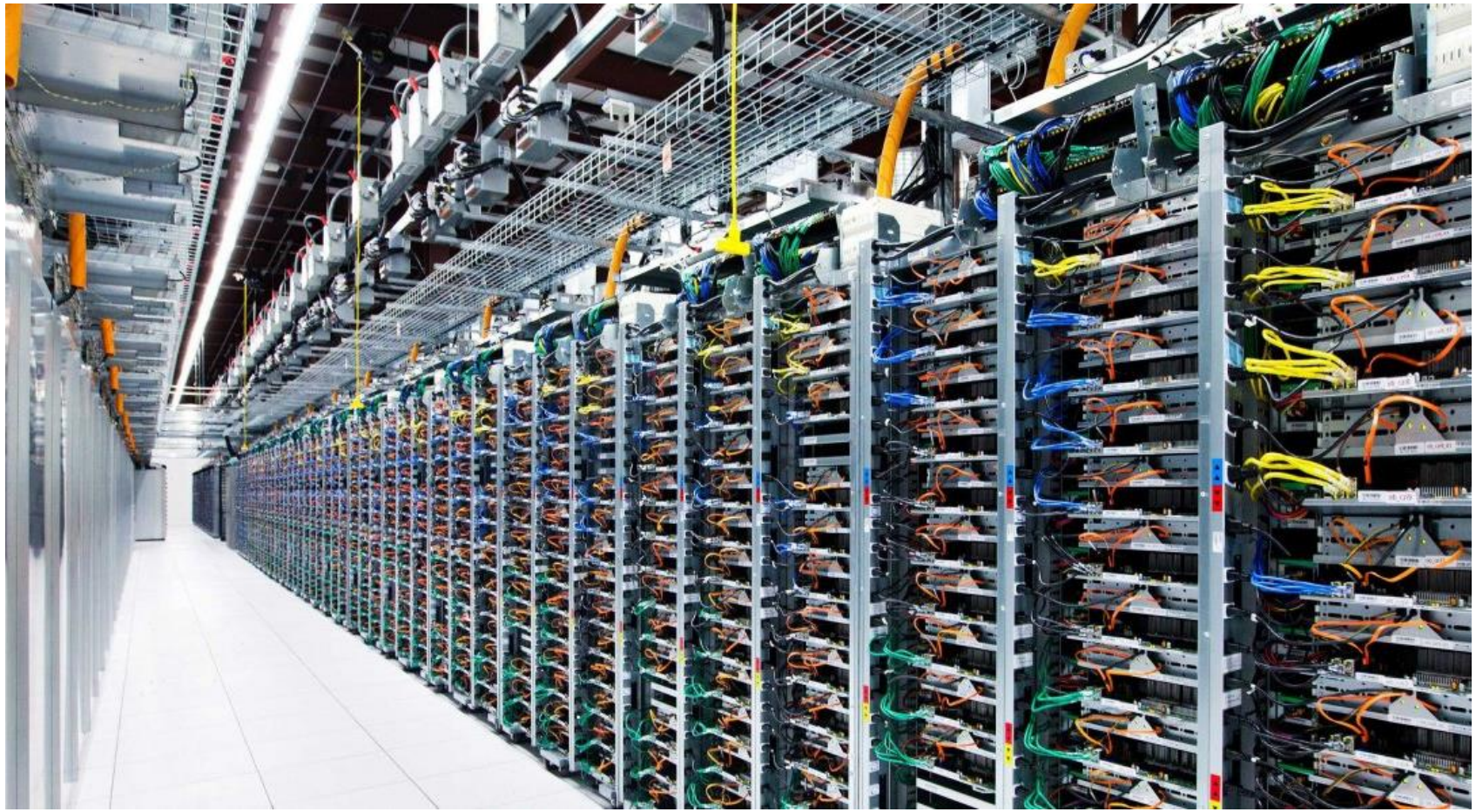
- Programmers spend a lot of time thinking about parallel code
- It is very error prone and time-consuming
- It still happens every day!

Scaling more than multiple cores

What does Google do with all of their data? Are they restricted to one computer (maybe with many cores)?

No!

Massive Distributed Systems (many networked computers)



Designing Distributed Programs

How do we get around the difficulty of writing parallel programs when working on distributed systems?

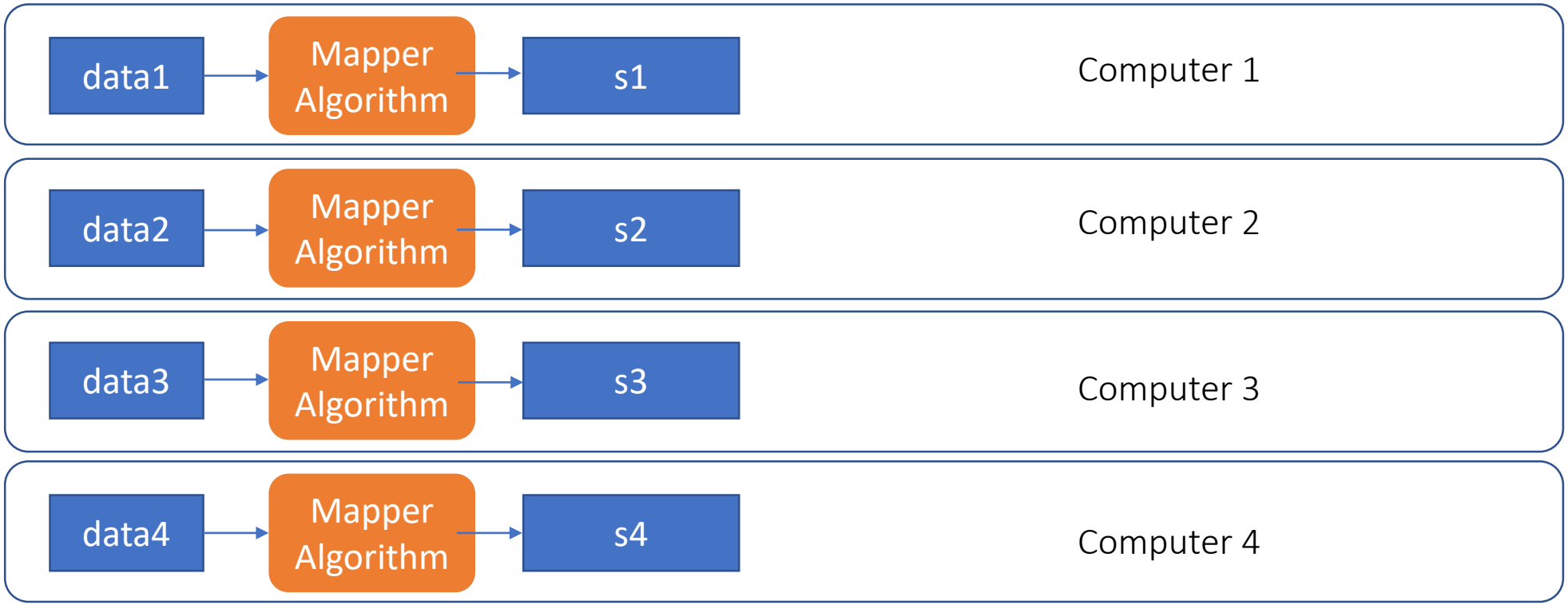
Sometimes we can come up with an algorithm that IS easily dividable.

One way to handle these specific problems is an algorithm called MapReduce

- invented at Google
- allows for a lot of concurrency in the map step

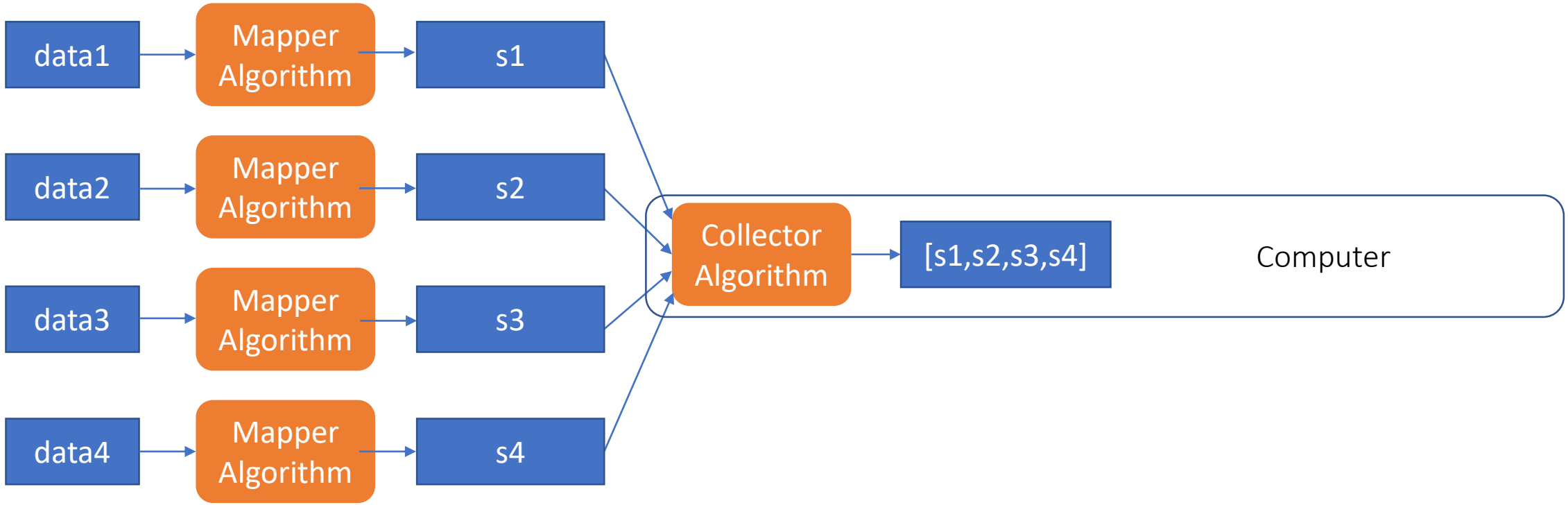
MapReduce Algorithm

Divide data into pieces and run a mapper function on each piece. The mapper returns some summary information (s1,s2,s3,s4) about the data. Each piece can be run on it's own computer.



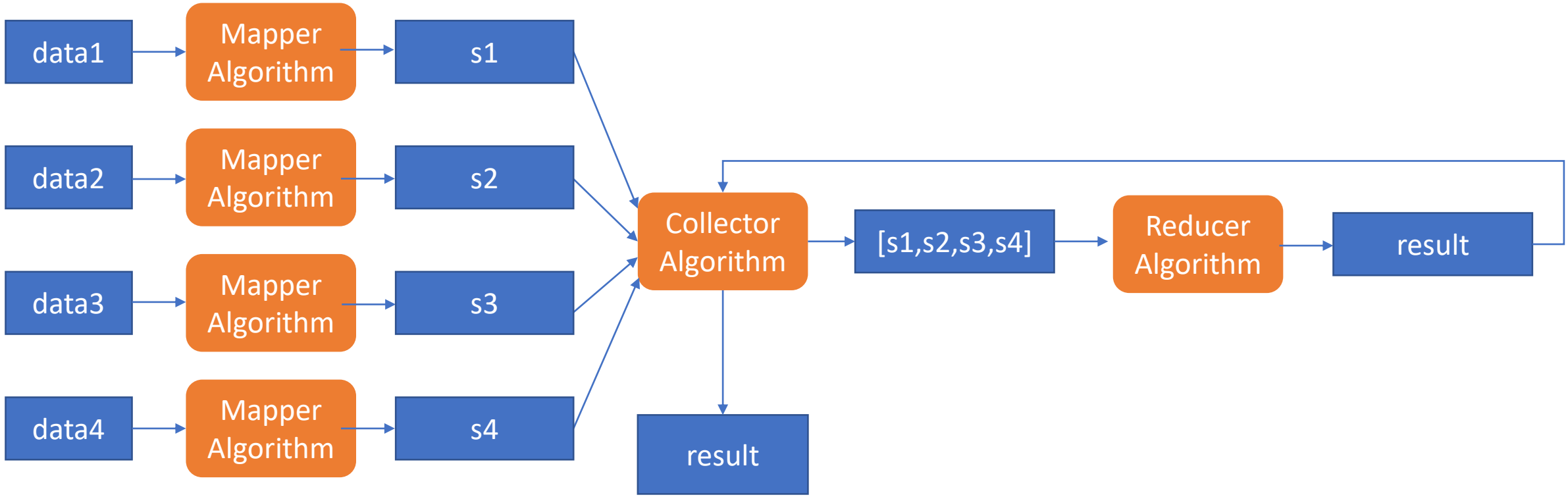
MapReduce Algorithm

The collector takes the summary information s from each computer and makes a list. The collector can run on another computer or one of the same computers.



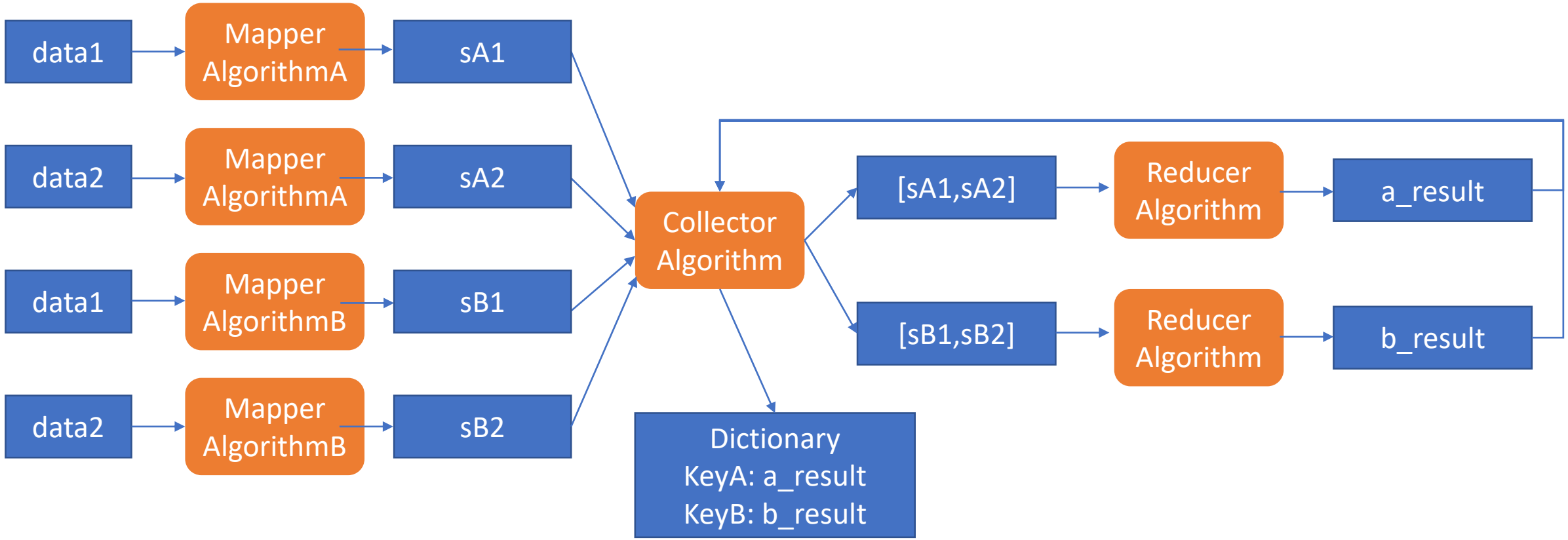
MapReduce Algorithm

The collector takes the summary information s from each computer and makes a list. The list is given to the reducer algorithm which takes the list and returns a result. Typically the collector outputs the result at the end.



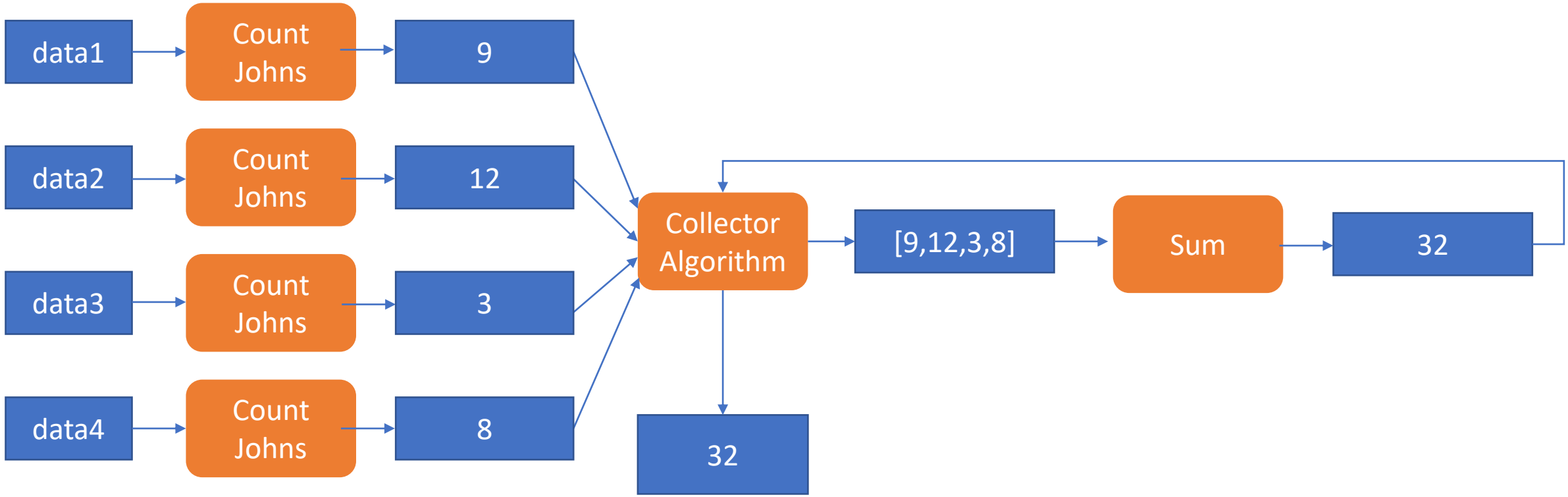
MapReduce Algorithm

Since the mapper can be any function, sometimes we have different mappers do different things and collect all results together. For example searching for many different words. In that case, the collector makes a list per algorithm, and outputs a dictionary of results.



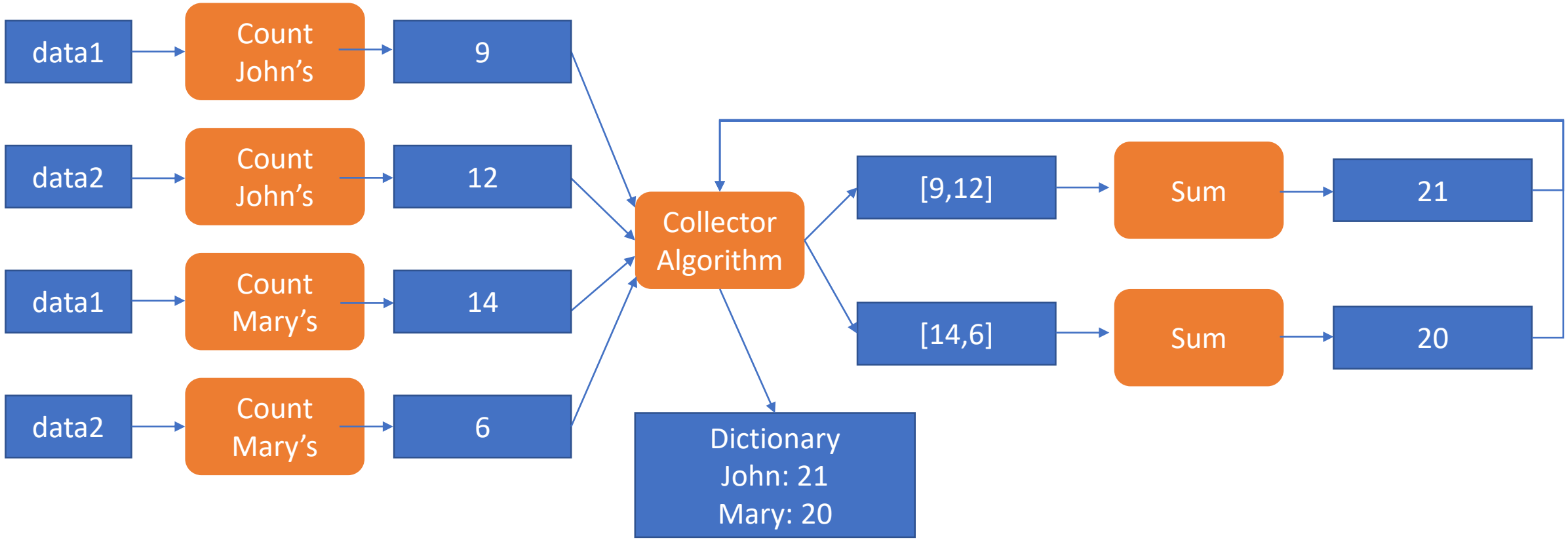
Example: Count Number of John's in Phonebook

Divide the phone book into parts data1,data2,data3,data4. Each mapper counts the number of John's and output as s1,s2,s3,s4 respectively. The collector gets all results, forms a list, and gives it to the reducer to sum the result.



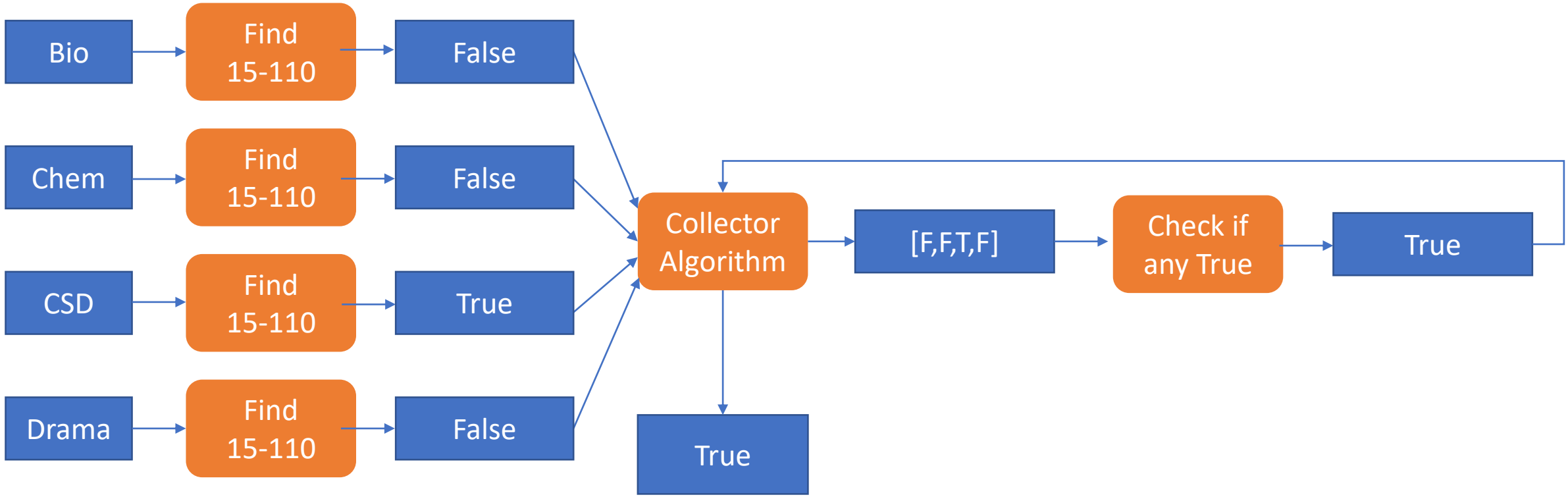
Example: Count John's and Mary's

Divide up the phonebook the same way. We run two different mappers on the same data (count John's and count Mary's). The collector keeps track of which answer goes to which mapper, makes separate lists for each, and then gives each list to a reducer. It outputs a dictionary of the results.



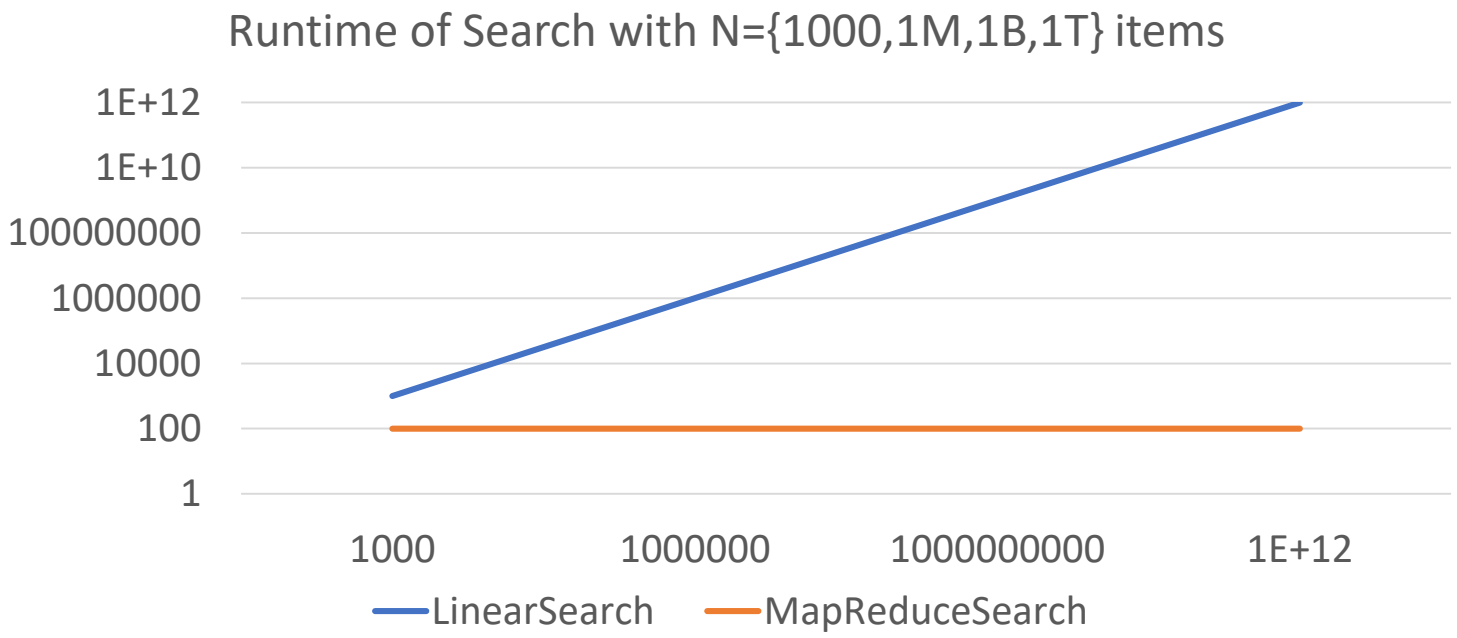
Example: Find 15-110 in course descriptions

Divide the course descriptions into parts - data1,data2,data3,data4. Each mapper checks if 15-110 is in there. The collector gets all results into a list, and the reducer checks if any are True. If yes, return True, if not return False.



Why Does MapReduce Run Quickly?

Suppose we had an n problem such as counting all the John's in a file. If I ran the computation like usual, it would take me O(n) time. If I broke the file into n/100 pieces (each file was 100 long), then it would run in O(1).



Takeaways from MapReduce

If we can find an algorithm that works on a small portion of our data (and that doesn't need any other part of the data too), then we can write a mapper function

Once we have a lot of mappers run, we can combine that data together using a reducer function.

You can even parallelize multiple mappers at the same time!

Takeaways of Multi-processing

- Multi-processing and distributed systems help reduce the runtime of programs by splitting up the work between cores, processors, or computers
- A goal is also to make them fault-tolerant - when a computer fails, the entire system doesn't fail.
- We do this by re-running only the computation on the failed computer and by backing up the same data across multiple machines so that the data isn't lost

Upsides of Multiprocessing

When using multiple machines, you can get much better performance than by using a single machine alone

- This is how Google gets search results so fast- by using hundreds of computers at once!

On a single machine, concurrency makes it possible to never waste time, thereby increasing the 'throughput' of the computer

- Throughput is the amount of work a computer can do in a given time period
- Example: while your computer is waiting for you to select an option in a pop-up menu, it might be handling work in another program in the background

Downsides of Multiprocessing

- It can be expensive to transfer a lot of data between different cores or computers

    The data has to move across more, longer wires

- Writing programs that run concurrent is much more complex, which can lead to more bugs
- Debugging concurrent software can be very very difficult, since behavior changes over multiple iterations!

    It’s like when we debug random programs.

    The randomness here is inherent in the scheduler.