Levels of Concurrency

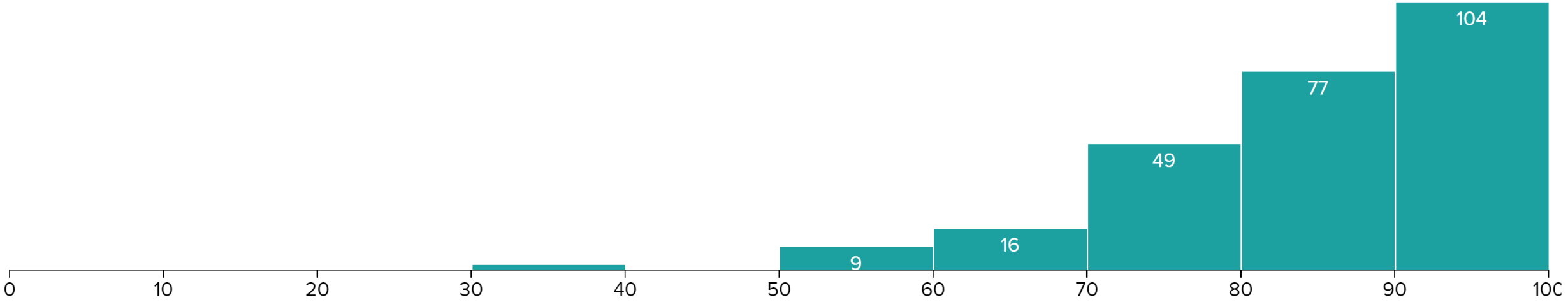
15-110 – Friday 10/22

Announcements - General

- Hw4 is due **Monday**
- Midsemester Feedback Forms are also due **Monday** [optional]
  - Over 2/3 of the class has provided feedback already – thank you!!
- Final exams dates are released!
  - 15-110's exam: **Thursday 12/09 8:30am-11:30am**
  - **DO NOT BOOK TRAVEL BEFORE THE EXAM**. We do not let students take the exam early.
- Academic Integrity reminder
  - When collaborating, **everyone should be actively participating**. One student should never be providing a solution to the others (even at a high level or just by talking)
  - Do not search online for solutions to problems, or even parts of solutions. Searching online is only okay for broader concepts and remembering names for functions.

Announcements – Quiz3

- Median: 86.5. **Well done!**



Learning Goals

- Define and understand the differences between the following types of concurrency: **circuit-level concurrency, multitasking, multiprocessing,** and **distributed computing**

- Create **concurrency trees** to increase the efficiency of complex operations by executing sub-operations at the same time

Unit Introduction

Scaling Up Computing

In the unit on Data Structures and Efficiency, we determined that certain algorithms may take a long time to run on large pieces of data.

In this unit, we'll address the following questions:

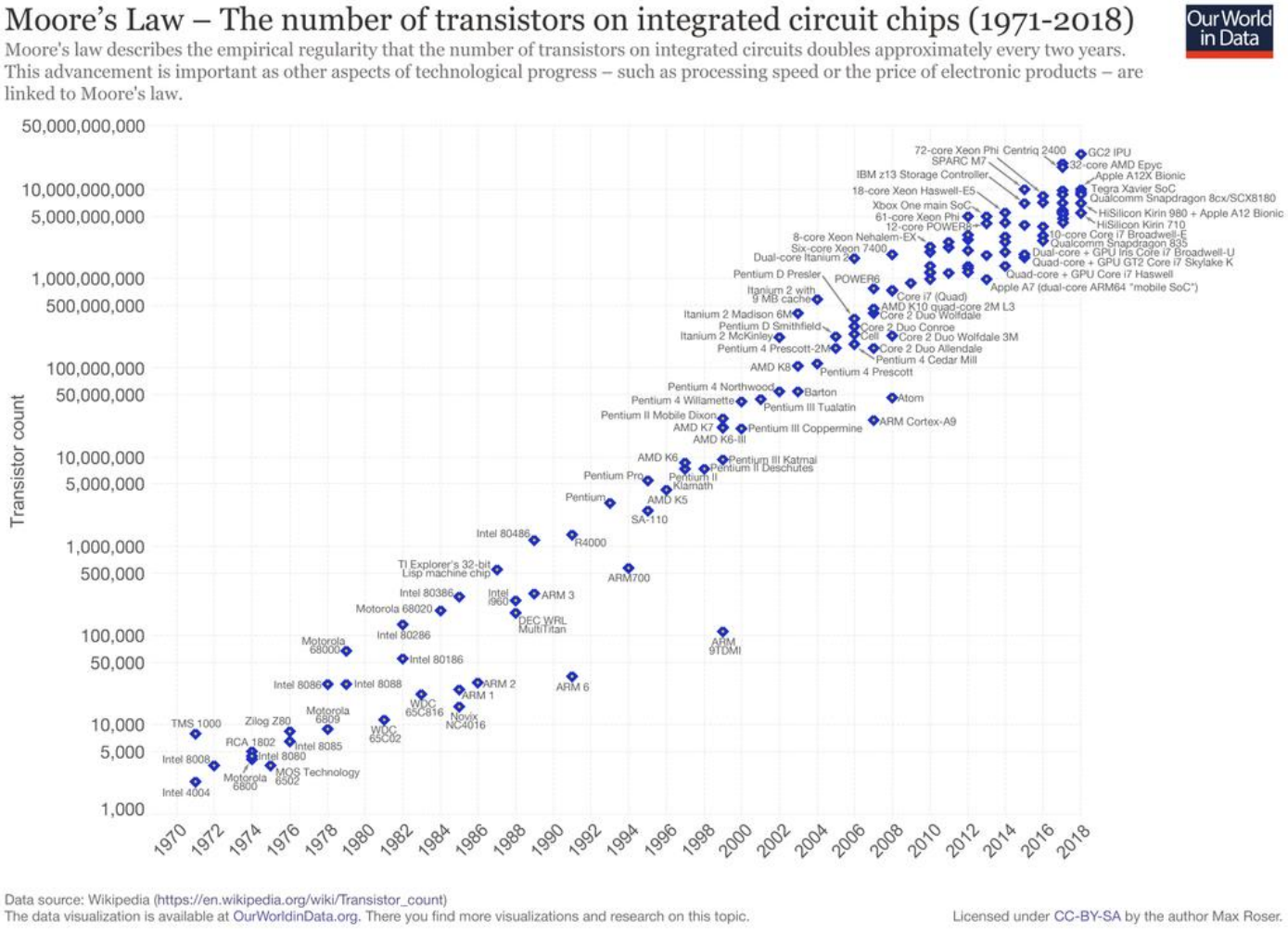
- How is it possible for complex algorithms on huge sets of data (like Google search) to run quickly?
- How can we write algorithms that require communication between multiple computers, instead of running individually?

Moore's Law: Computers Keep Getting Faster

You've probably noticed that the computer you use now is much faster than the computer you used ten years ago. That's because of a technology principle known as **Moore's Law**.

Moore's Law basically states that the power of a computer doubles **every two years**. If you buy a computer designed in 2020, it should be **twice as powerful** as a computer made in 2018.

Note: Moore's Law is an observation, not an actual law of nature. But how does it work?



Transistors Provide Electronic Switching

Recall the lecture on gates and circuits. How does the computer send data to different circuits for different tasks?

This is accomplished using a **transistor,** a small device that makes it possible to switch electric signals. In other words, adding a transistor to a circuit gives the computer a **choice** between two different actions. Gates are partially made out of transistors.

When we make transistors smaller, we can decrease the distance between them (reducing communication time) and increase the number that fit on a chip. Smaller transistors also use less current. This makes the computer **faster**.

Moore's Law: Double the Transistors

A more precise statement of Moore's Law is that the number of transistors on a computer chip will double every two years. This provides the increase in computing power, and the speed-up.

Originally, engineers were able to double the number of transistors by making them smaller every year, to fit twice as many transistors on a single computer chip. But around 2010 it became physically impossible to make the transistors smaller at such a rapid rate (due to electronic leakage).

Now engineers attempt to follow Moore's Law by using **parallelization** instead. In other words, your computer may contain multiple processing units, and may run more than one block of instructions **at the same time**.

Levels of Concurrency

Concurrency and Parallelization

In general, when we refer to the term **concurrency**, we mean that multiple programs are running at exactly the same time.

We will also refer to **parallelization** as the process of taking an algorithm and breaking it up so that it can run across multiple concurrent processes at the same time.

In this lecture, we'll discuss four different levels at which concurrency occurs. Next time, we'll discuss broad approaches for implementing parallel algorithms.

Four Levels of Concurrency

The four levels of concurrency are:

Circuit-Level Concurrency: concurrent actions on a single CPU

Multitasking: seemingly-concurrent programs on a single CPU

Multiprocessing: concurrent programs across multiple CPUs

Distributed Computing: concurrent programs across multiple computers

A CPU Manages Computation

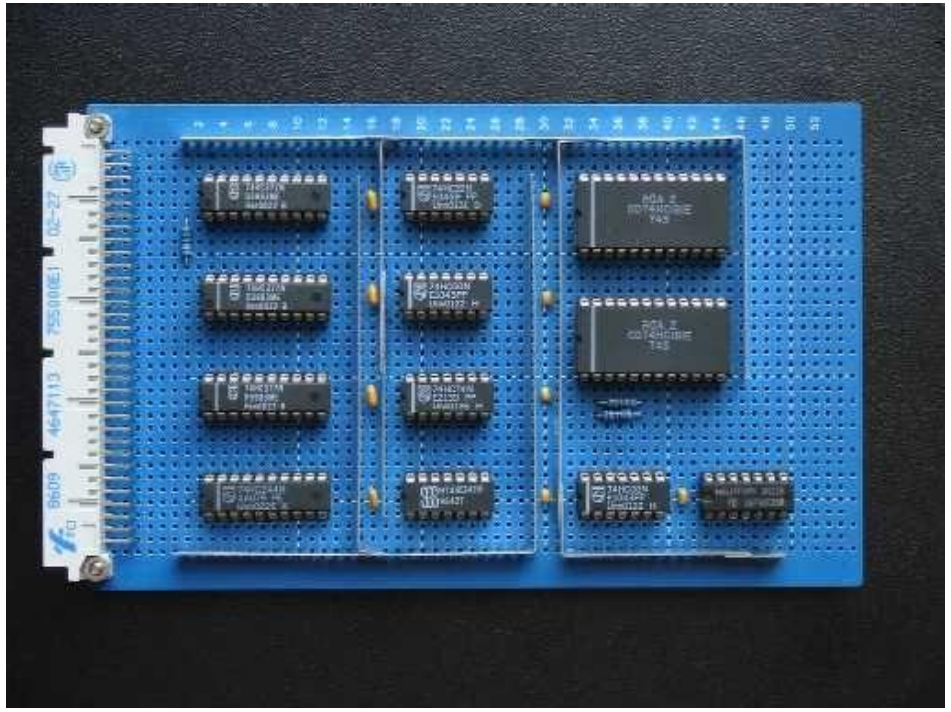
A **CPU** (or Central Processing Unit) is the part of a computer's hardware that actually runs the actions taken by a program. It's composed of a large number of circuits.

The CPU is made up of several parts. It has a **control unit**, which maps the individual steps taken by a program to specific circuits. It also has many **registers**, which store information and act as temporary memory.

CPUs Have Many Logic Units

For our purpose, the most interesting part is the **logic units**. These are a set of circuits that can perform basic arithmetic operations (like addition or multiplication).

Importantly, the CPU has many **duplicates** of these- it might have hundreds of logic units that all perform addition.



1: Circuit-Level Concurrency

The first level of concurrency happens within a single CPU, or **core**. Because the CPU has many arithmetic units, it can break up complex mathematical operations so that subparts of the operation run on separate logic units **at the same time**.

For example, if a computer needs to compute (2 + 3) * (5 + 7), it can send (2 + 3) and (5 + 7) to two different addition units **simultaneously**. Once it gets the results, it can then send them to the multiplication unit. This only takes **two time steps**, instead of three.

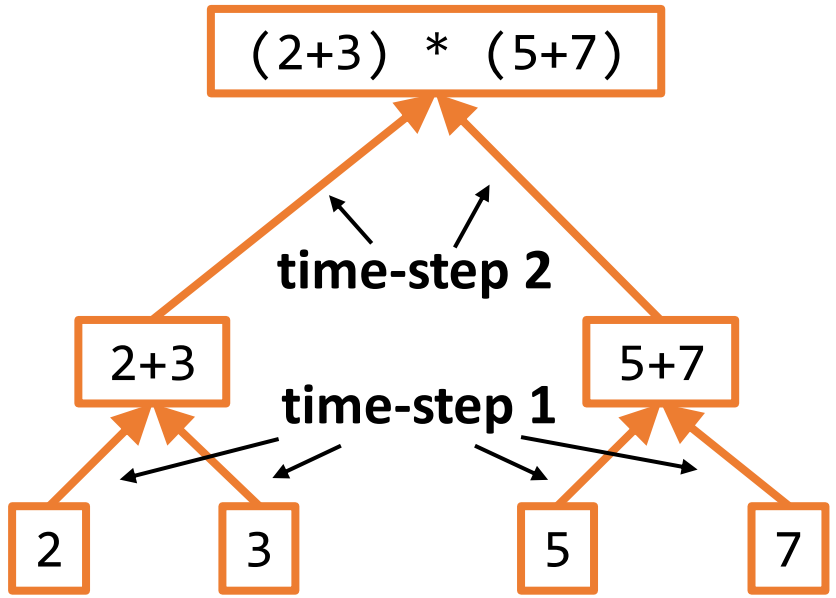
Concurrency Trees

A **concurrency tree** is a tree that shows how a complex operation can be broken down into the fewest possible time steps.

Actions which occur simultaneously are written as nodes at the same **level** of the tree. Nodes are on the same level when they are the same distance from the root.

The **total** number of steps is the number of non-leaf nodes in the tree. This example tree has three total steps.

The number of **time-steps** is the number of non-leaf **levels** in the tree. This example tree has two time-steps.



Example Concurrency Tree

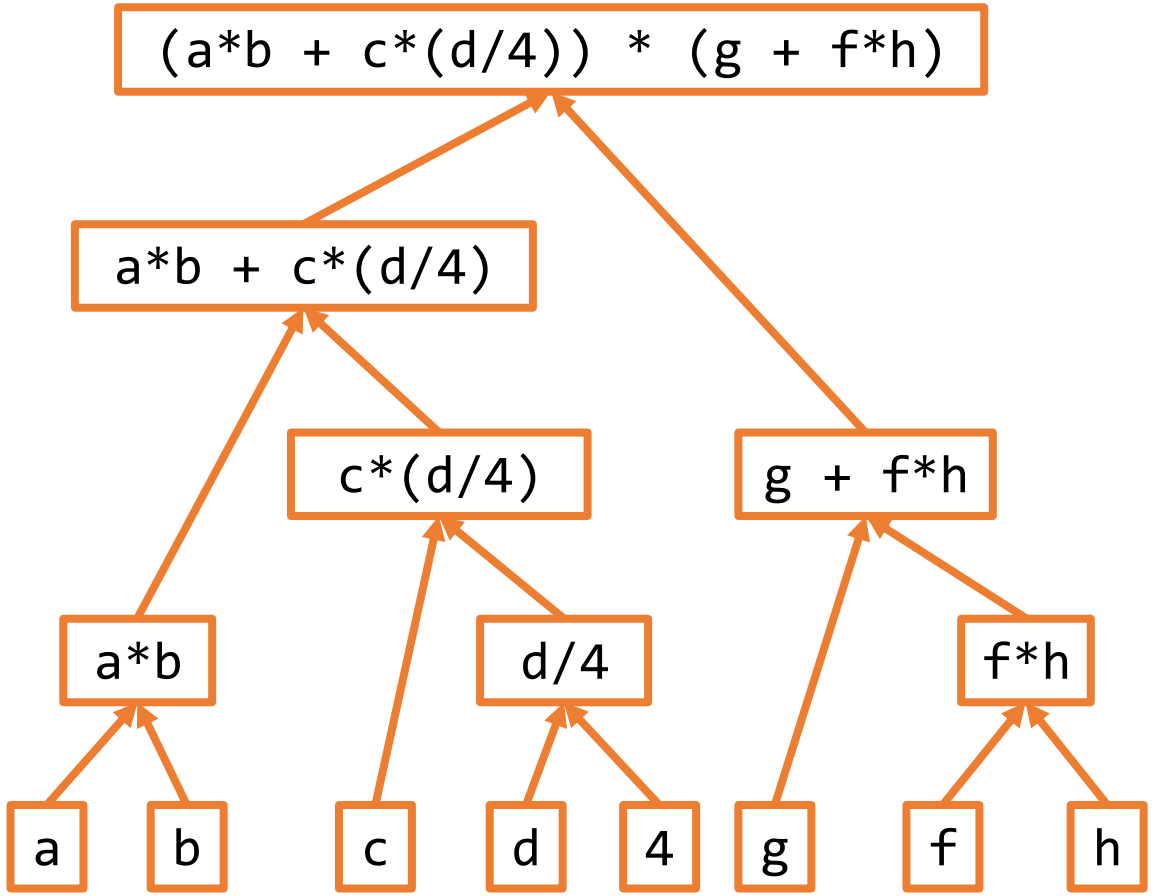
For example, let's make a concurrency tree for `(a*b + c*(d/4)) * (g + f*h)`

In the first time-step, we can compute `a*b`, `d/4`, and `f*h`.

The next time-step contains the operations that **required** those computations to be done already – `c*(d/4)` and `g + f*h`.

In general, the operations at each level could not be done any earlier in the process.

This tree has **seven total steps** and **four time-steps**.



Activity: Count Equation Steps

Consider the following equation:

```
((a*b + 1) - a) + ((c/2) * (d*e + f))
```

How many **total steps** does it take to compute this equation?

How many **time-steps** does it take to compute this equation?

Hint: If you aren't sure, try drawing a concurrency tree!

2: Multitasking

The second level of concurrency is **multitasking**.

This level is very different from the others in that it doesn't actually run multiple actions at the same time. Instead, it creates the **appearance** of concurrent actions.

CPU Schedulers Arrange Programs

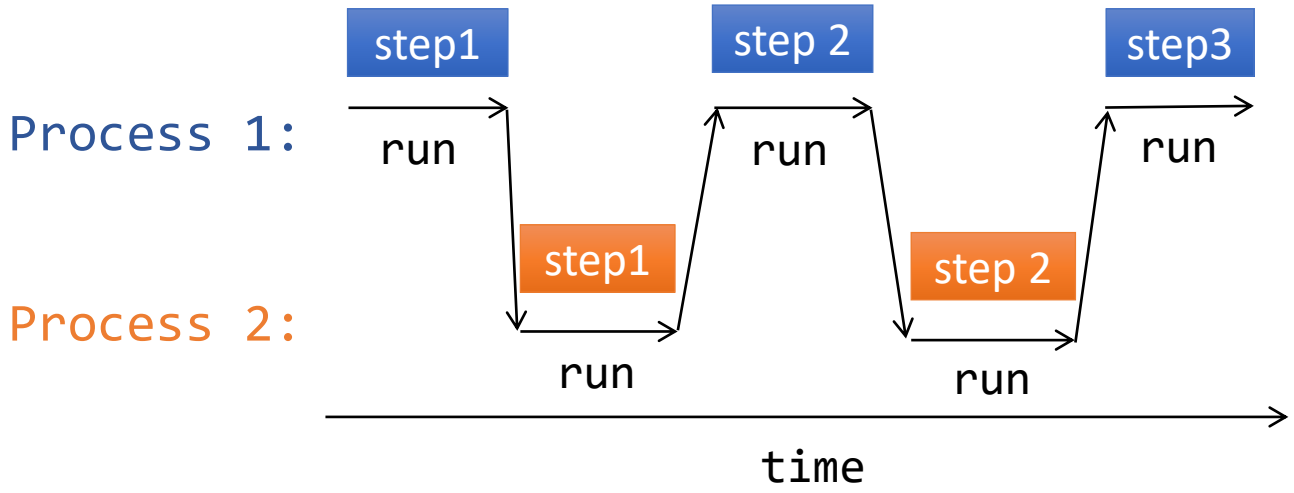
Multitasking is accomplished by a part of the operating system called a **scheduler**. This is a component that decides which program action will happen next in the CPU.

When your computer is running multiple applications at the same time – like your browser, and a word editor, and Pyzo – the scheduler decides which program gets to use the CPU at any given point.

Multitasking with a Scheduler

When multiple applications are running at the same time, the scheduler can make them **seem** to run at the same time by breaking each application's process into steps, then alternating between the steps rapidly.

If this alternation happens quickly enough, it looks like true concurrency to the user, even though only one process is running at any given point in time.

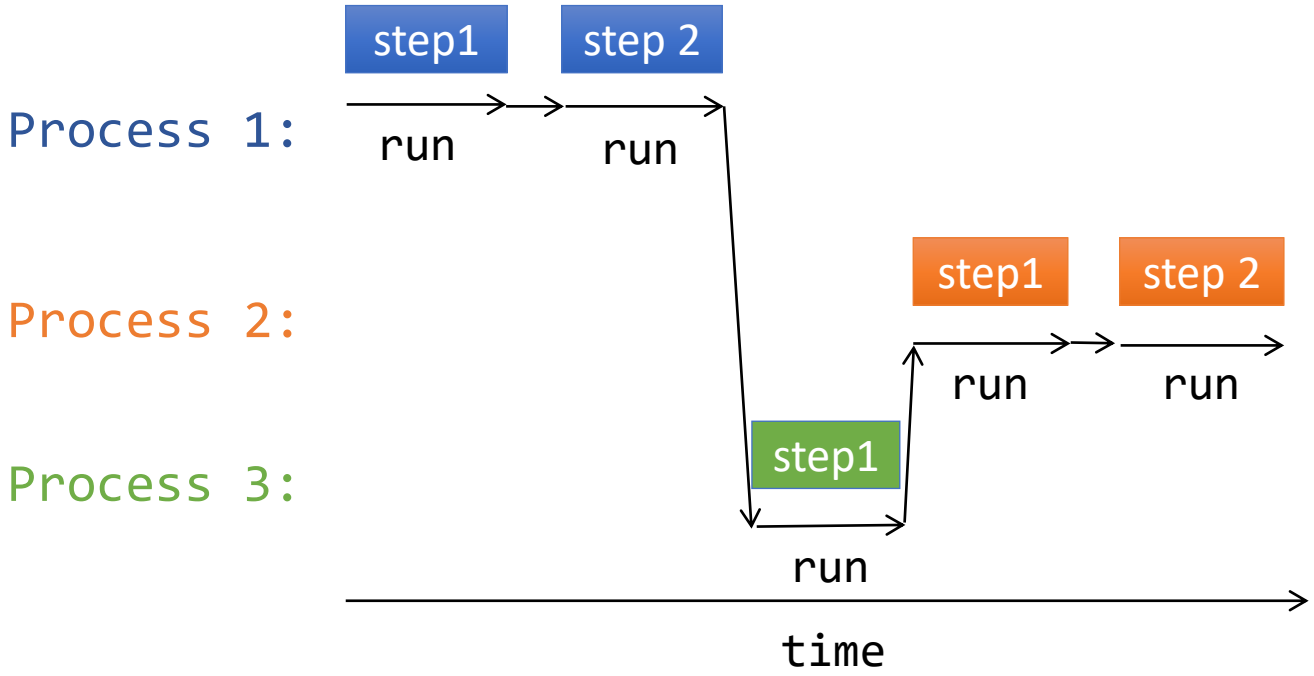


Schedulers Can Choose Any Order

When two (or more) processes are running at the same time, the steps don't need to alternate perfectly.

The scheduler may choose to run several steps of one process, then switch to one step of another, then run all the steps of a third. It might even choose to put a process on hold for a long time, if it isn't a priority.

In general, the scheduler chooses which order to run the steps in to **maximize throughput** for the user. Throughput is the amount of work a computer can do during a set length of time.

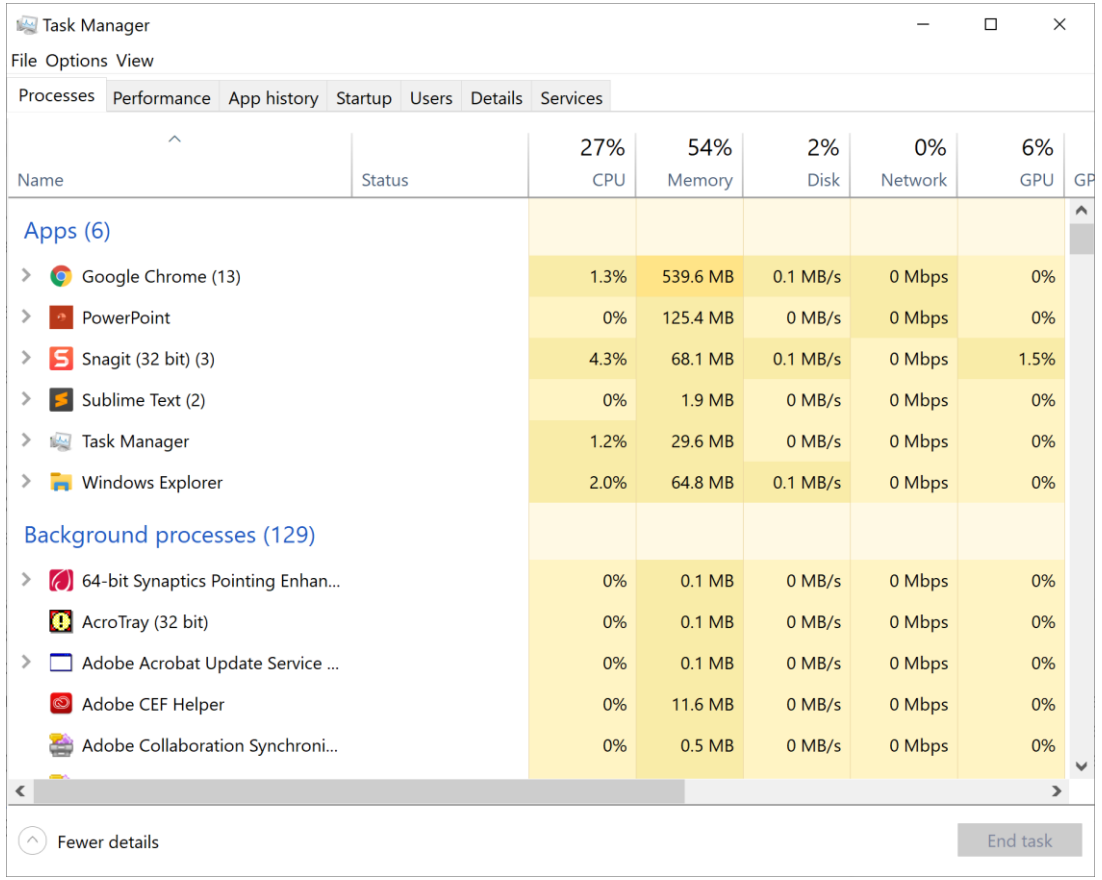


Your Computer Multitasks

Your computer uses multitasking to manage all of the applications you run, as well as the background processes needed to make your operating system work.

You can see all the applications your computer's scheduler is managing by going to your process manager (Task Manager on Windows, Activity Monitor on Macs). You can even see how much time each process gets on the CPU!

You do: open your process manager now to see how much CPU time each application takes



3: Multiprocessing

The third level of concurrency, **multiprocessing**, can run multiple applications **at the exact same time** on a single computer.

To make this possible, we put **multiple CPUs** inside a single computer, then run different applications on different CPUs at the same time.

By multiplying the number of actions we can run at a point in time, we multiply the speed of the computer.

Multiple Processor vs. Multi-Core

Technically there are two ways to put several CPUs into a single machine.

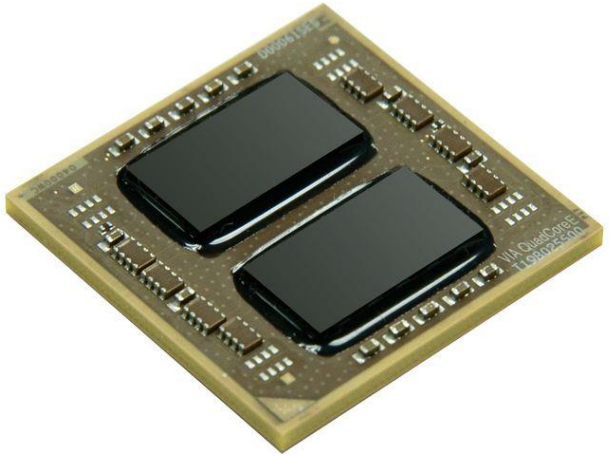
The first is to insert more than one processor chip into the computer. This is called **multiple processors**.

The second is to put multiple 'cores' on a single chip. Each core can manage its own set of actions. This is called **multi-core**.

There are slight differences between these two approaches in terms of how quickly the CPUs can work together and how they access memory. For this class, we'll treat them as the same.

Multiple Processors

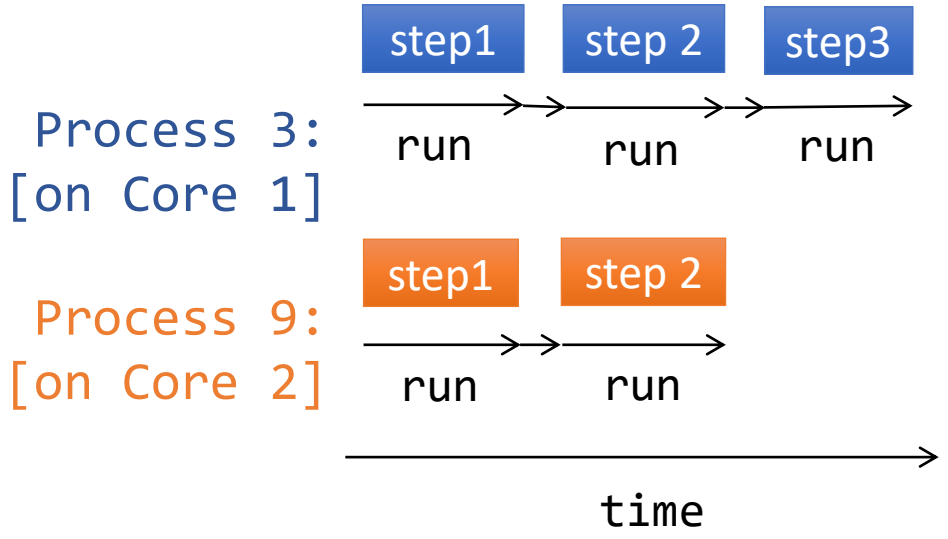
Multi-Core



Scheduling with Multiprocessing

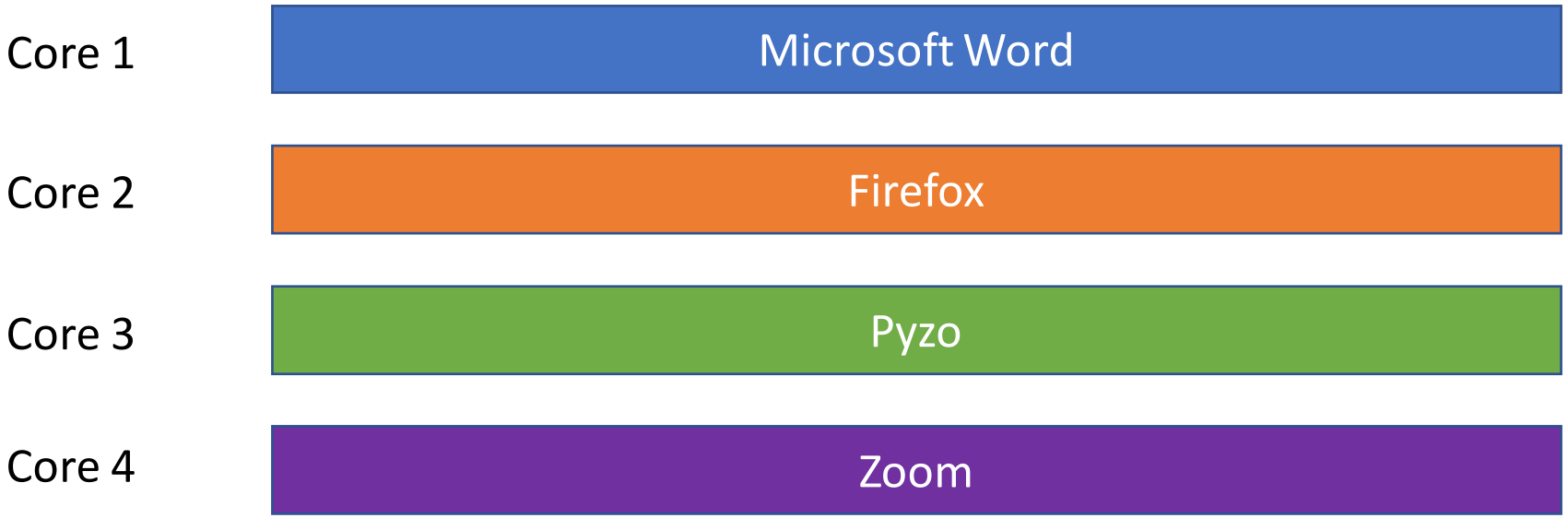
When we use multiple cores and multiprocessing, we can run our applications simultaneously by assigning them to different cores.

Each core has its own scheduler, so they can work independently.



Simplified Scheduling

Here's a simplified visualization of scheduling with multiprocessing, where we condense all of the steps of an application into one block.



Multiprocessing and Multitasking

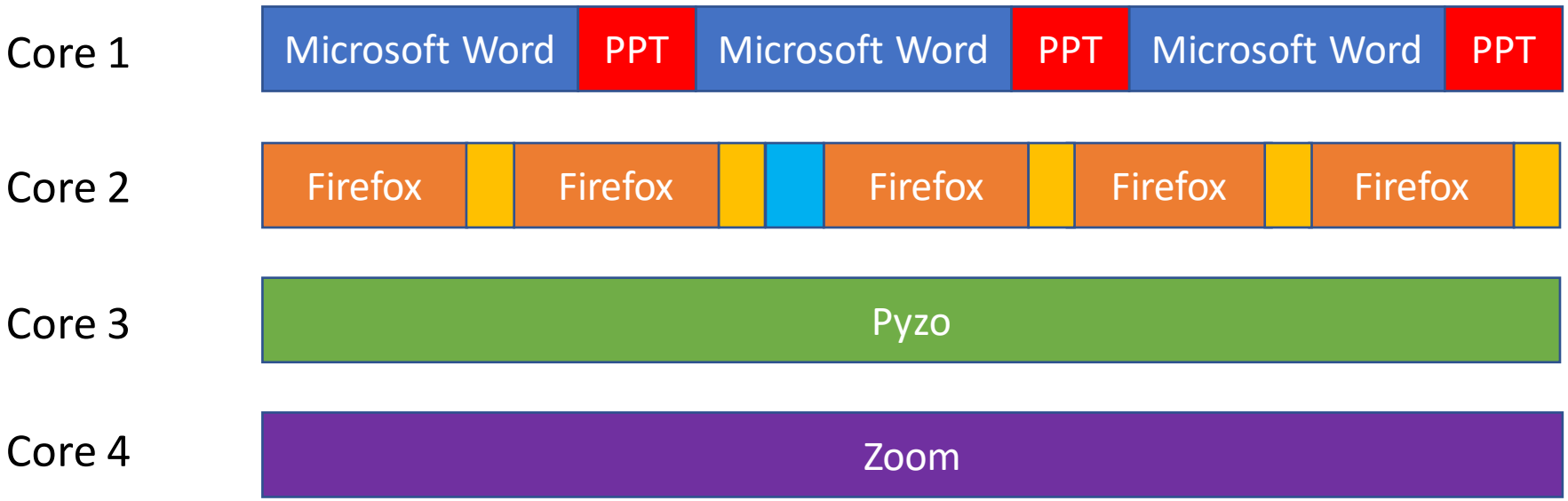
The number of cores we have on a single computer is usually still limited. Most modern computers use somewhere between 2-8 cores. If you run more than 2-8 applications at the same time, the cores use **multitasking** to make them appear to run concurrently.

You can check how many cores your own computer has! If you're on Windows, go back to the process manager and switch to the tab 'Performance'. If you're on a Mac, go to About This Mac > System Report > Hardware.

You do: look up how many cores your computer has!

Scheduling with Multiprocessing and Multitasking

Here's a simplified view of what scheduling might look like when we combine multiprocessing with multitasking.



4: Distributed Computing

The final level of concurrency, **distributed computing**, goes beyond using a single machine.

If we have access to several computers (each with its own set of CPUs), we can network them together and use them all to perform advanced computations by assigning different subtasks to different computers.

By multiplying the number of computers that are working on a single problem, we can multiply the speed of a difficult computation.

Scheduling with Distributed Computing

Each computer in the network can take a single task, break it up into further subtasks, and assign those subtasks to its cores. This makes it possible for us to attempt to solve problems which would take a long time to solve on a single processor.

Task 1

Core 1	Subtask 1-1
Core 2	Subtask 1-2
Core 3	Subtask 1-3
Core 4	Subtask 1-4

Task 2

Core 1	Subtask 2-1
Core 2	Subtask 2-2
Core 3	Subtask 2-3
Core 4	Subtask 2-4

Task 3

Core 1	Subtask 3-1
Core 2	Subtask 3-2
Core 3	Subtask 3-3
Core 4	Subtask 3-4

Task 4

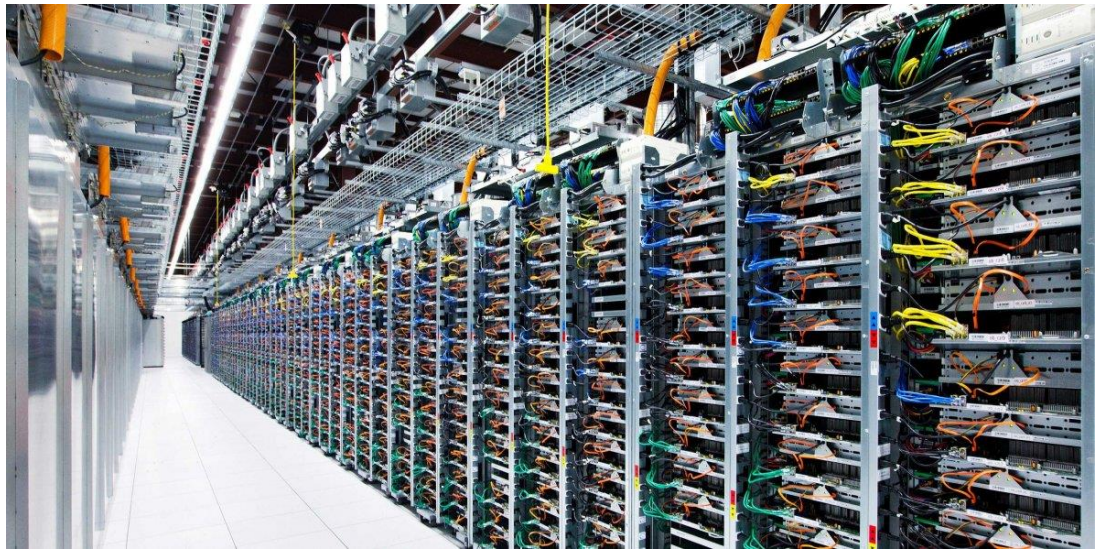
Core 1	Subtask 4-1
Core 2	Subtask 4-2
Core 3	Subtask 4-3
Core 4	Subtask 4-4

Companies Use Distributed Computing

Distributed computing is used by big tech companies (like Google and Amazon) both to manage thousands of customers simultaneously and to process complex actions quickly.

This is where the term 'server farm' comes from- these companies will construct large buildings full of thousands of computers which are all networked together and ready to process information.

A **supercomputer** is very similar to distributed computing. It's a computer with a *huge* number of processors connected together. The main difference is that all the processors are located in the same place.



Distributed Computing Must Be Fault Tolerant

When using distributed computing, it's very important that algorithms are designed to be **fault tolerant**.

The probability that a computer randomly crashes while running a program is low (maybe 1 in 10,000). But server farms regularly run far more than 10,000 computers at the same time.

Algorithms that run on distributed systems must be designed to have checks in place to make sure that no work is left unfinished. Typically, storage is also backed up on multiple machines, to make sure no data is lost if a single machine goes down.

Learning Goals

- Define and understand the differences between the following types of concurrency: **circuit-level concurrency, multitasking, multiprocessing,** and **distributed computing**

- Create **concurrency trees** to increase the efficiency of complex operations by executing sub-operations at the same time

Feedback: https://bit.ly/110-f21-feedback