Concurrent Programming

Aaron Smith

COMP 301

May 3, 2021

Motivating example

Baking cookies!

Baking cookies

Measure the dry ingredients

Measure the wet ingredients

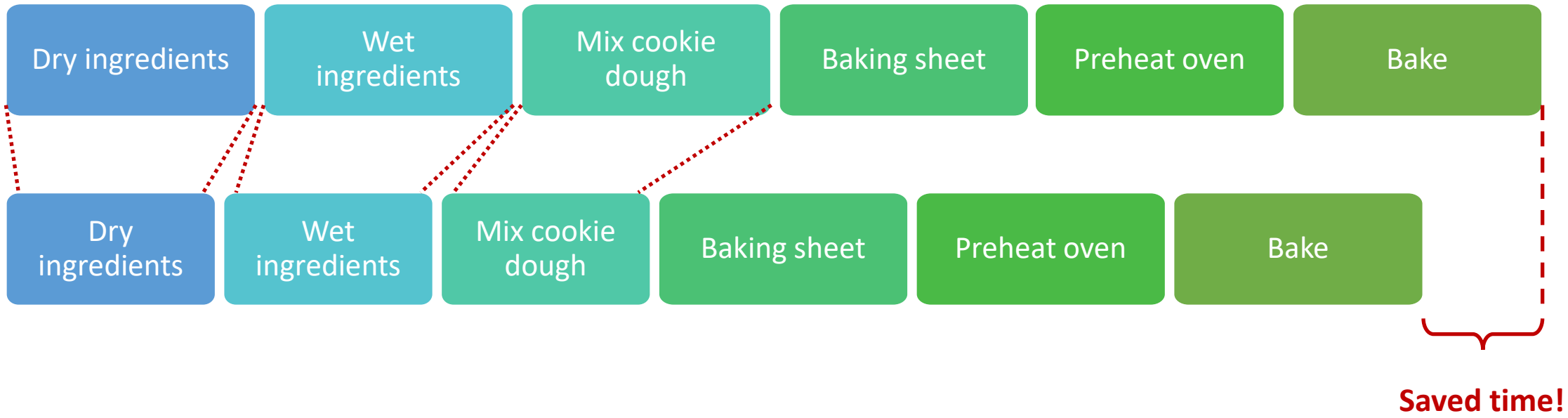
Combine the wet and dry ingredients

Put the cookie dough on the sheet

Preheat the oven

Bake the cookies

Improving the algorithm



You got a new mixer! Now you can mix the ingredients faster!

Improving the algorithm

You can preheat the oven while mixing the ingredients!

Improving the algorithm

You can have a friend help you!

How to speed up completing a task

Strategy	Baking cookies	Executing a program
Do the same steps, only faster	Upgrade your mixer	Upgrade your computer
Rearrange tasks so you can do one while waiting for the other to finish	Preheat while portioning on the baking sheet	**Asynchronous programming**
Get a friend to help you do tasks at the same time	Wet ingredients and dry ingredients at the same time	**Parallel programming**

Concurrent Computing

Poll Everywhere (1)

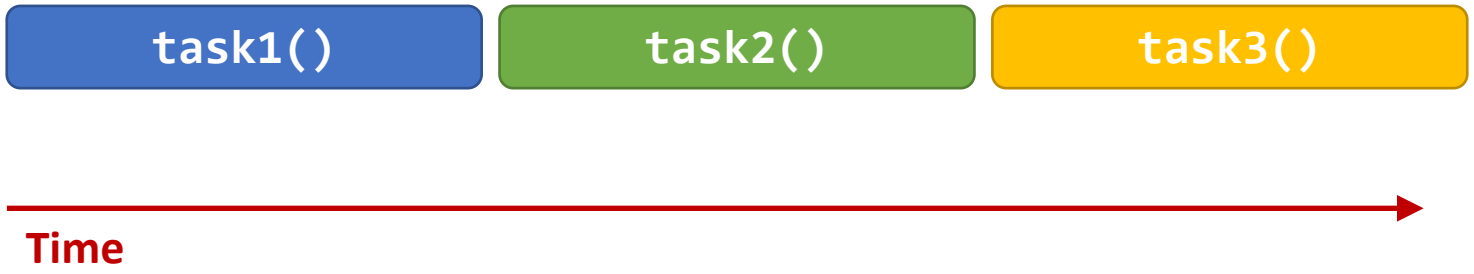
To answer, go to
https://pollev.com/onsmith

Prompt: Think of an example of an everyday experience that could be made faster by parallelizing. Do you need another person? Or can you parallelize the task by yourself?

Concurrent vs sequential computing

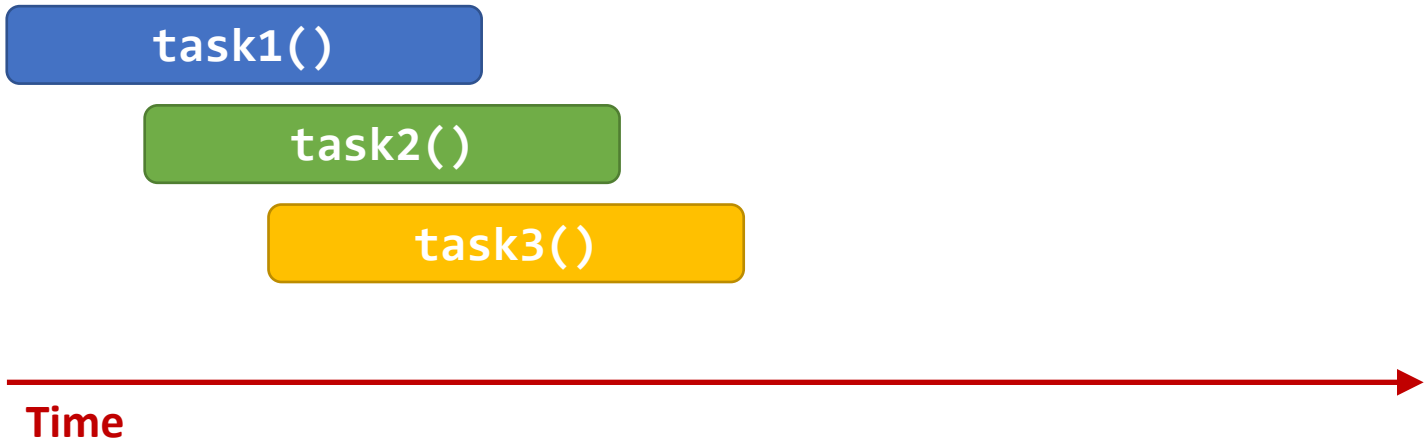
Definitions

Sequential computing – When a series of computations are executed ***one at a time***, such that each computation must finish before the next can begin



Definitions

Concurrent computing – When a series of computations are executed ***during overlapping time periods***



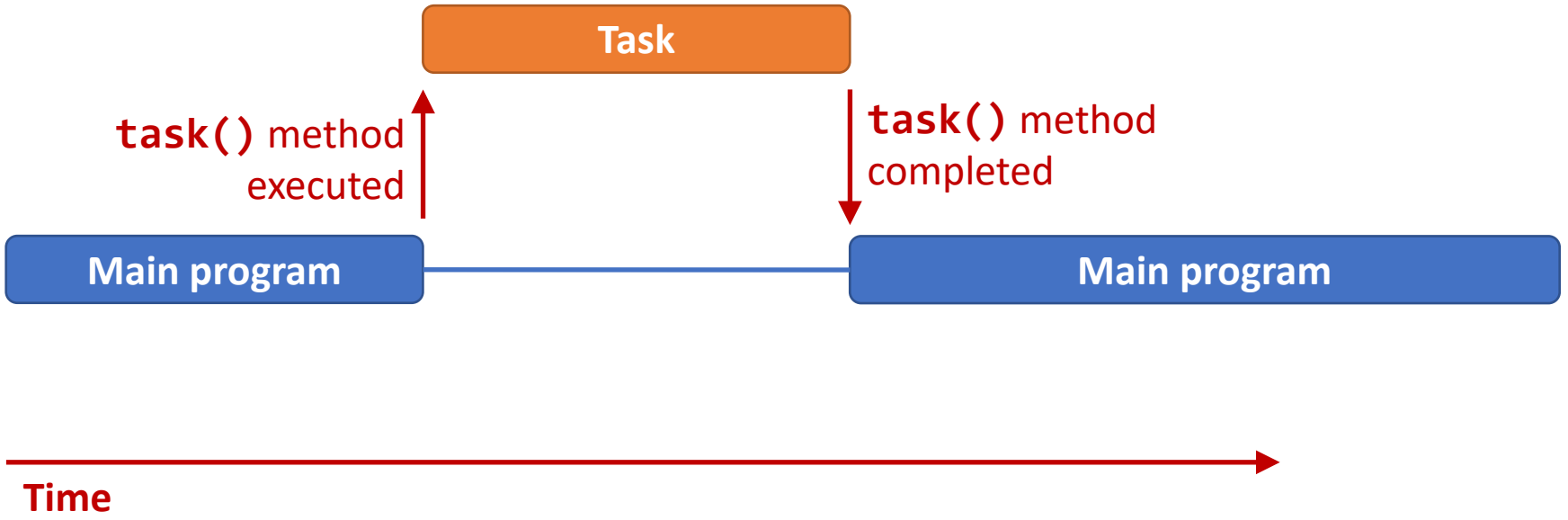
The tasks might be executed simultaneously…

…or maybe the computer is taking turns switching back and forth between them

Synchronous and asynchronous programming models

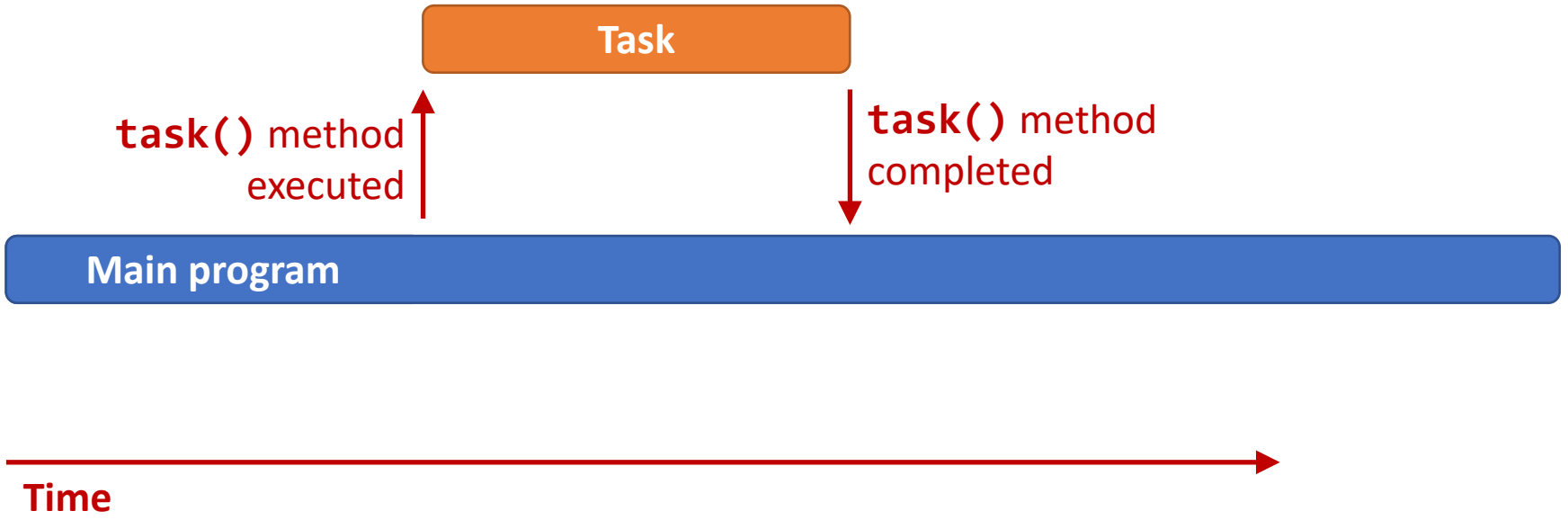
Definitions

Synchronous programming – A model of programming where a task may be started, and the program ***waits for it to complete*** before continuing on



Definitions

Asynchronous programming – A model of programming where a task may be started, but the program ***continues on without waiting*** for it to complete



Threads and multithreading

Definitions

A "**thread**" is an abstraction for *executing a program*

- To execute any program, your operating system creates a thread

A thread encapsulates the following information

1. **Instruction pointer**: the current point of execution
2. **Call stack**: which methods are currently executing
3. **Memory**: the contents of memory, including the heap

A single program can run *multiple threads at the same time*

- Separate **instruction pointer**
- Separate **call stack**
- **Shared memory**

Java `Runnable` interface

- The **`Runnable`** interface is built-in to Java
- A **`Runnable`** object represents a task that can be performed
- The task is performed by calling the **`run()`** method
- The task might be performed ***synchronously*** or ***asynchronously***

```
public interface Runnable {
  void run();
}
```

What Java syntax shortcut might be used to create a new **`Runnable`** object?

Creating a `Runnable` object

```
public class Multithreader {
  public static void main(String[] args) {
    Runnable task1 =
        () -> {
          for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
          }
        };
  }
}
```

Poll Everywhere (2)

To answer, go to
https://pollev.com/onsmith

```
public class Multithreader {
  public static void main(String[] args) {
    Runnable task1 =
        () -> {
          for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
          }
        };
  }
}
```

Prompt: What will this program print?

Running a `Runnable` object *synchronously*

This program will execute **sequentially** and **synchronously**

```
public class Multithreader {
  public static void main(String[] args) {
    Runnable task1 =
        () -> {
          for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
          }
        };

    System.out.println("Printing 1 to 10");
    task1.run();
    System.out.println("Done");
  }
}
```

No concurrency here; we simply created a **`Runnable`** object and executed its **`run()`** method

Java `Thread` class

Java's **`Thread`** class

- Built-in to Java
- Represents a ***thread of execution***
- Must be given a **`Runnable`** object to execute in the constructor
- Has a special **`start()`** method for executing the **`Runnable`**
- Executes the **`Runnable`** ***asynchronously (in parallel)***

```
Thread thread = new Thread(task1);
thread.start();
```

Running a `Runnable` object *asynchronously*

This program will execute **concurrently** and **asynchronously**

```
public class Multithreader {
  public static void main(String[] args) {
    Runnable task1 =
        () -> {
          for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
          }
        };

    System.out.println("Printing 1 to 10");
    Thread thread = new Thread(task1);
    thread.start();
    System.out.println("Done");

  }
}
```

The **`Runnable`**'s **`run()`** method will execute ***at the same time as the rest of the program***

What happens at this line?

Poll Everywhere (3)

To answer, go to
https://pollev.com/onsmith

```
public class Multithreader {
  public static void main(String[] args) {
    Runnable task1 =
        () -> {
          for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
          }
        };

    System.out.println("Printing 1 to 10");
    Thread thread = new Thread(task1);
    thread.start();
    System.out.println("Done");

  }
}
```

Prompt: What will this program print?