CS 2113
Software Engineering

Lecture 12: Learning to Multi-task

Clone https://github.com/cs2113f16/lec-12.git

Professor Tim Wood - The George Washington University

This Time

- **More with GUIs**
  - New types of classes in Java
- **Threading**
  - Let your program do multiple things at once
- Last project

Button Events

- Something must implement ActionListener

```
class NewFrame extends JFrame implements ActionListener {
  public NewFrame (int width, int height)
  {
   // ...
   button.addActionListener(this);
   // ...
  }
  public void actionPerformed (ActionEvent a)
  {
    System.out.println ("ActionPerformed!");
  }
```

Multiple Buttons

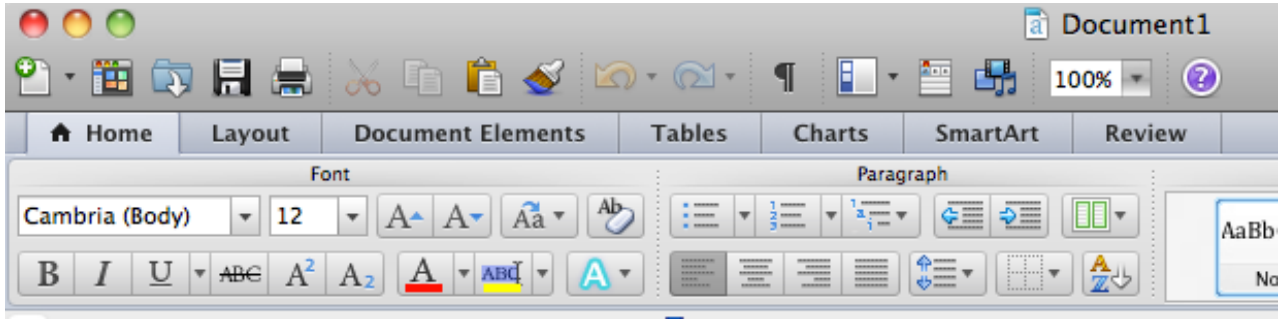
- In **`guis.buttons.ButtonTest`**
  - Make the hello and bye buttons change the text of the msg label.

- How can we differentiate between the two buttons?
  - Need to inspect the ActionEvent parameter you are passed
  - Use either:
    - a.getActionCommand() returns the clicked button's text label
    - a.getSource() returns a reference to the object that started the event (i.e., the JButton instance that was clicked)

`https://github.com/cs2113f16/lec-12.git`

…that can get messy

- when we have lots buttons!




```
public void actionPerformed (ActionEvent a)
{
  // Get the button string.
  String s = a.getActionCommand();

  if (s.equalsIgnoreCase ("Bold")) {
    // ...
  }
  else if (s.equalsIgnoreCase ("Italic")) {
    // ...
  }
  else if (s.equalsIgnoreCase ("Right Justify")) {
    // ...
  }
```

What else can we do?

- Having one giant event handler function is messy
  - Need to be careful that a change to one button won't break code for another
- Do we have an alternative?
- We want to use OOP principles!
  - Compartmentalize functionality
  - Reuse code instead of copy/pasting
  - Isolate and protect data

Options…

- We could create custom classes just for handling the events

```
class QuitButtonHandler implements ActionListener {
  public void actionPerformed (ActionEvent a)
  {
    System.out.println ("ActionPerformed!");
  }
}
```

```
class NewFrame extends JFrame {
  public NewFrame ()
  {
   // ...
   quitButton.addActionListener(new QuitButtonHandler);
   randButton.addActionListener(new RandomButtonHandler);
  }
```

Problem???

- We could create custom classes just for handling the events

```
class QuitButtonHandler implements ActionListener {
  public void actionPerformed (ActionEvent a)
  {
    System.out.println ("ActionPerformed!");
  }
}
```

What if the event handler needs access to data from NewFrame?

```
class NewFrame extends JFrame {
  public NewFrame ()
  {
   // ...
   quitButton.addActionListener(new QuitButtonHandler);
   randButton.addActionListener(new RandomButtonHandler);
  }
```

What we really want:

- To isolate the event handler for each object

- To allow the event handlers to access the data of the class they are in

What we really want:

- To isolate the event handler for each object
  - but a single class can only implement the functions in an interface once!
- To allow the event handlers to access the data of the class they are in
  - but if we use separate classes for each event handler we won't be able to do this!
- Oh noes!
  - :(

Inner Classes

- Java to the rescue!
- Use an **Inner Class**

```
class myClass {
  class myInnerClass {
    void someFunc() {
    }
  }

}
```

Inner Classes

- Use an **Inner Class**

```
class myPanel {
   private JLabel myLabel;

   class eHandler1 implements ActionListener {
      myLabel.setText("Handler 1!");
   }

   class eHandler2 implements ActionListener {
   }

   class eHandler3 implements ActionListener {
   }
}
```

What can it do?

- Can an inner class touch its outer's privates?
  - **Yes** it can!
- Can an "outer" class call functions in the inner?
  - **Yes** it can!
- Can an inner class implement an interface?
  - **Yes** it can!
- Can an inner class extend another class?
  - **Yes** it can!

Sample Code

- Check out **guis.inner.InnerTest.java**
- Note:
  - The inner class can have: functions, data, and constructors
  - The inner class can access private data of its outer class
  - The outer class can call into the inner class
- **Todo:**
  - **Make the outer class print out the values of X and Y**

Worksheet!

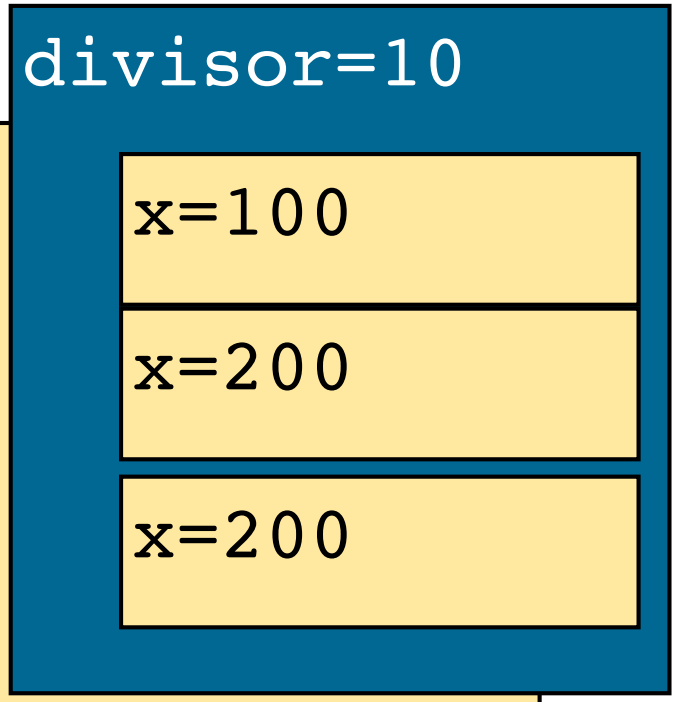
- What would this code print?

```
public class InnerQuiz {
   private int divisor;

   class MyInnerClass {
      private int x;

      public MyInnerClass(int xin) {
         x = xin;
      }
      public void print() {
         System.out.println(x/divisor);
      }
   }
   public InnerQuiz() {
      divisor = 1;
      MyInnerClass inny1 = new MyInnerClass(100);
      divisor = 10;
      MyInnerClass inny2 = new MyInnerClass(200);
      inny1.print();
      inny2.print();
   }
}
```

Worksheet!

- What would this code print?

```
public class InnerQuiz {
   private int divisor;

   class MyInnerClass {
      private int x;

      public MyInnerClass(int xin) {
         x = xin;
      }
      public void print() {
         System.out.println(x/divisor);
      }
   }
   public InnerQuiz() {
      divisor = 1;
      MyInnerClass inny1 = new MyInnerClass(100);
      divisor = 10;
      MyInnerClass inny2 = new MyInnerClass(200);
      inny1.print();
      inny2.print();
   }
}
```



Worksheet!

- What would this code print?

```
public class InnerQuiz {
   private int divisor;

   class MyInnerClass {
      private int x;

      public MyInnerClass(int xin) {
         x = xin;
      }
      public void print() {
         System.out.println(x/divisor);
      }
   }
   public InnerQuiz() {
      divisor = 1;
      MyInnerClass inny1 = new MyInnerClass(100);
      divisor = 10;
      MyInnerClass inny2 = new MyInnerClass(200);
      inny1.print();
      inny2.print();
   }
}
```

Each **MyInnerClass** instance has its own value for **x**, but they share the same **divisor** variable defined in the outer class.

Worksheet!

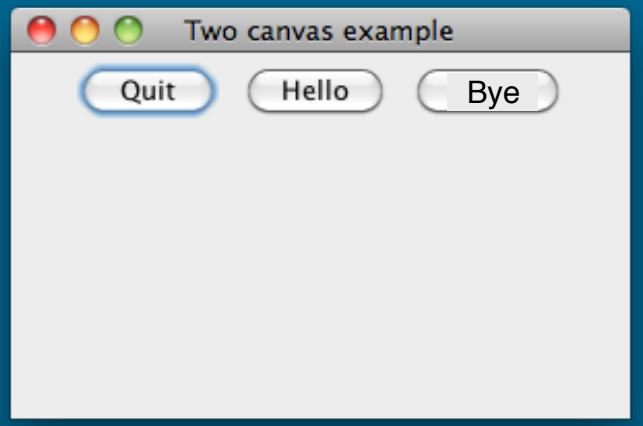
- What would this code print?

```
public class InnerQuiz {
   private int divisor;

   class MyInnerClass {
      private int x;

      pu

      }
      pu

      }
   }
   publi
      di
      My
      di
      MyInnerClass inny2 = new MyInnerClass(200);
      inny1.print();
      inny2.print();
   }
}
```

An Inner class can (**Circle ALL that apply**)
(a) Read and write public data in its outer class
(b) Read private data in its outer class
(c) Write private data in its outer class
(d) Have its own private data written from its outer class

Inner Class Event Handlers

- Look at **guis.inner.InnerEvents.java**
- We want to have:
  - **Quit**: quits
  - **Hello**: display "hello"
  - **World**: prints "bye"
  - (in the msg JLabel)



- The quit button currently uses an inner class
- **Your turn:**
  - Add two new inner classes for Hello and Bye
- **Elite Hacker:**
  - Combine your two inner classes into a single inner class

Types of Classes in Java

- A public/private class
  - Must have name equal to file
- A class with no privacy modifier
  - Only usable within that package
- A inner class inside of another class
  - Inner can access the outer and vice versa

- An anonymous inner class

```
quitB.addActionListener (
  new ActionListener() {
    public void actionPerformed (ActionEvent a)
    { System.exit (0); }
  }
);
```

… and a few other types too!

Multi-Tasking

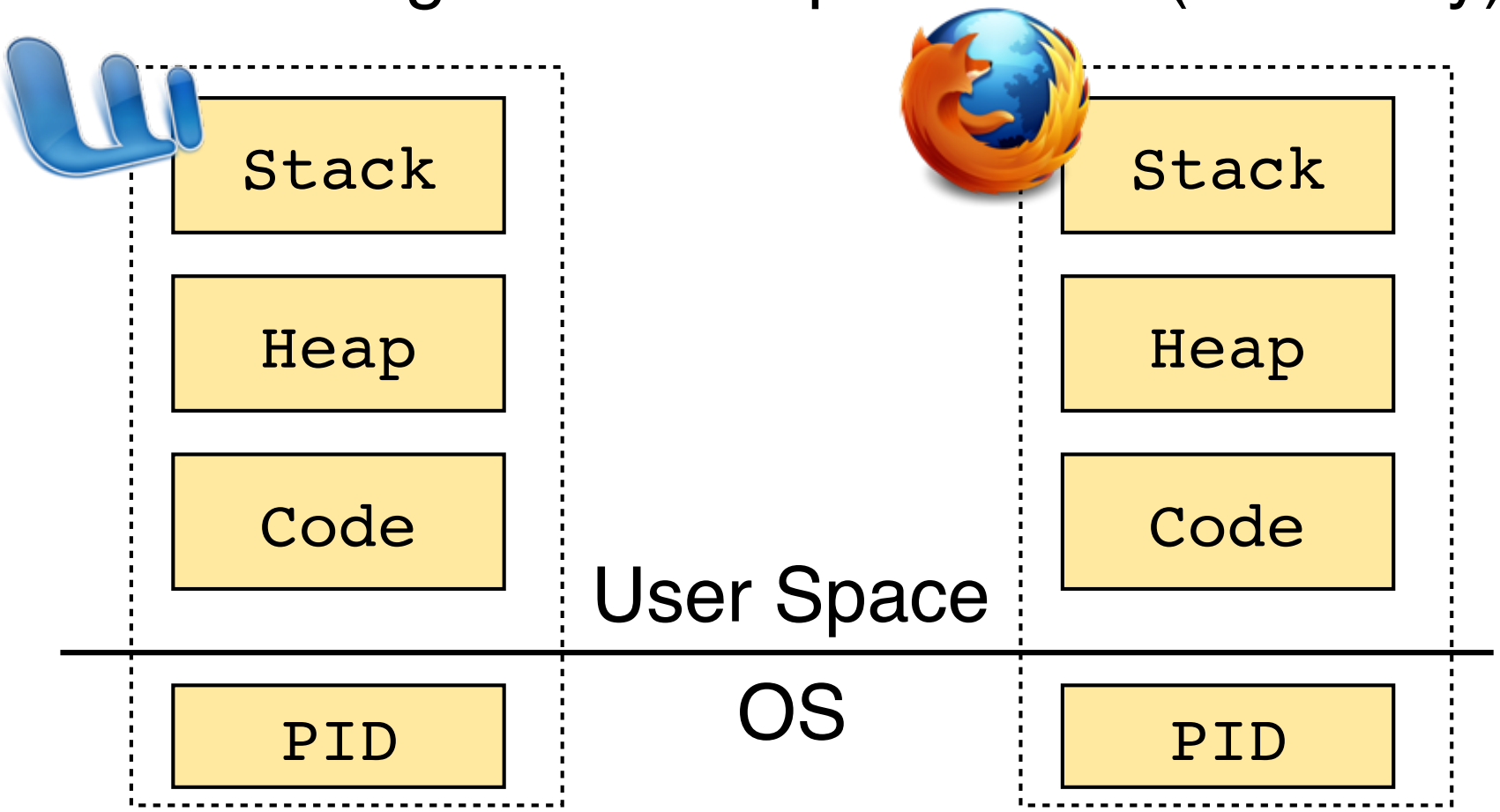
What is different?

- The desktop I owned in 1995?
  - 133 Mhz Pentium CPU

- The laptop I own now?
  - 2.2 Ghz Intel i7 4 core CPU

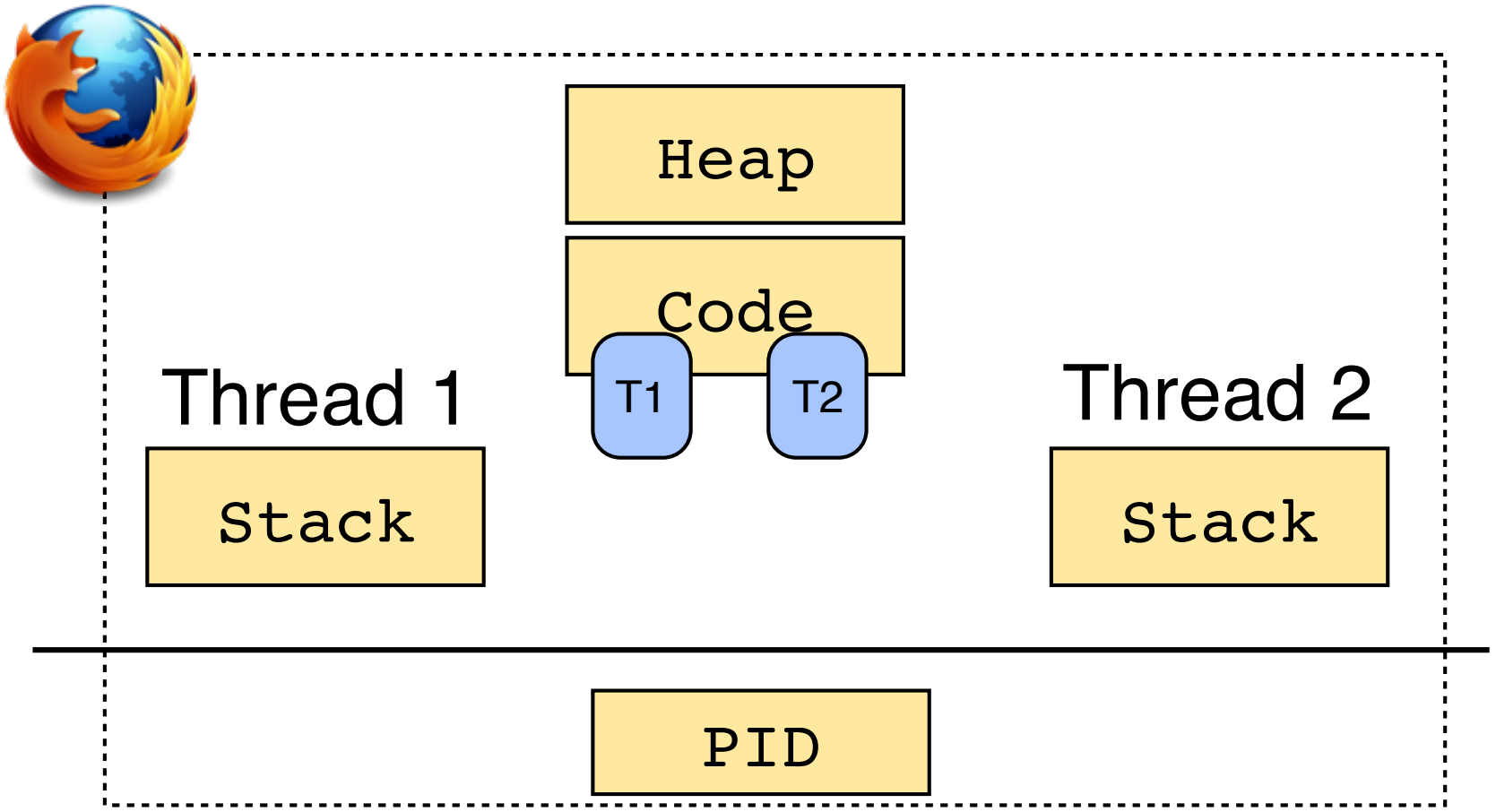
Threads and Processes

- Operating system schedules **processes**
- Each process has own resources and code
- Switching the active process is (relatively) slow



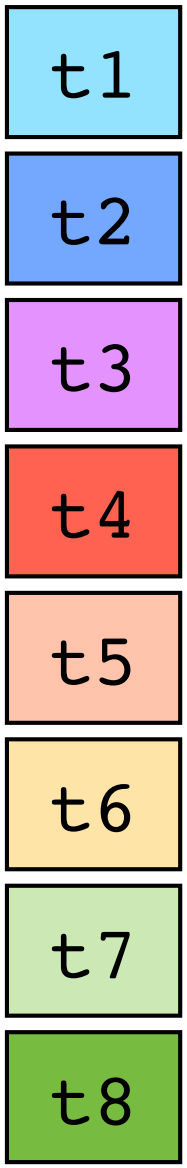
Threads and Processes

- Threads allow **concurrency** within one app
- Fast to switch between
- Threads share the same memory space



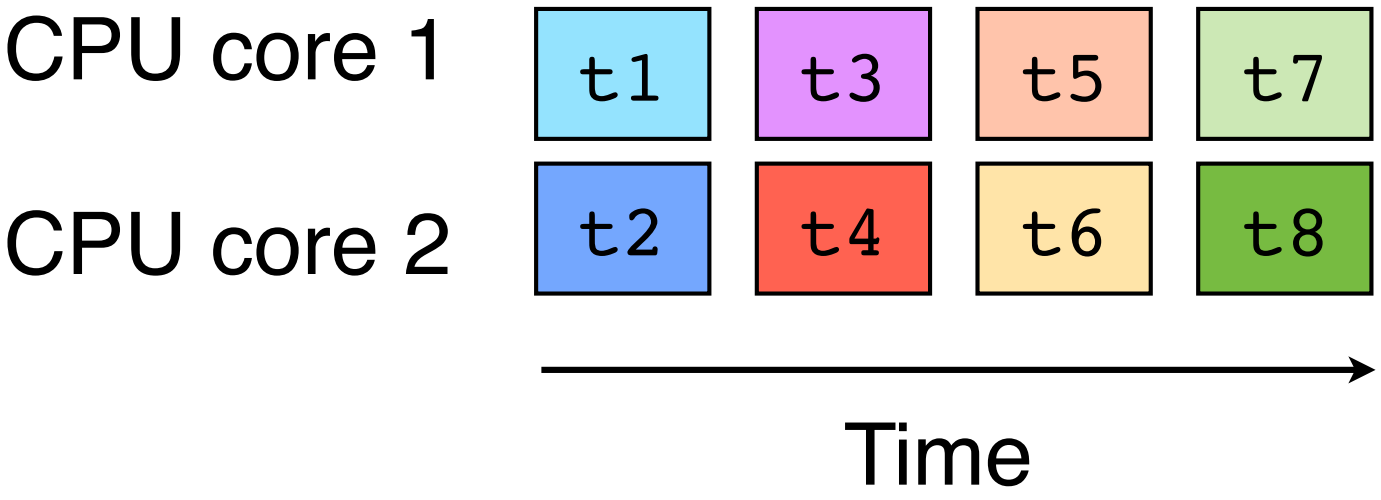
Multi-tasking

- If we start 8 threads, will they all **run at once?**



Multi-tasking

- If we start 8 threads, will they all **run at once?**
  - **No!**



- Can only run one thread per CPU core at a time

Multi-threading

- Use threads whenever you want to do (at least) two things at the same time
  - Perform a calculation without freezing the GUI
  - Read from disk and write to the network at the same time
  - Run ten long running simulations at the same time
  - Read from the network without pausing the GUI
- Your program starts with one thread for the **`main()`** method!
- Create a new thread and give it something to **`run()`** whenever you need parallelism
  - but don't directly call **run()** yourself! (Use **start()**)

Threads in Java

- **A Thread is a class that knows how to run**
  - Java uses the **`Runnable`** interface for this

```
public class MyTask implements Runnable {

    private int x;

    public MyTask(int i) {
        x = i;
    }
    public void run() {
        // THIS IS THE CODE TO RUN IN NEW THREAD
        // it will be called automatically when the thread starts
    }

    public static void main(String[] args) {

        int numIterations = 10;
        int numThreads = 5;

        for (int i = 0; i < numThreads; i++) {
            Thread t = new Thread(new MyTask(numIterations));
            t.setName("Thread " + i);
            t.start();
            System.out.println("When will this appear?");
        }
    }
}
```

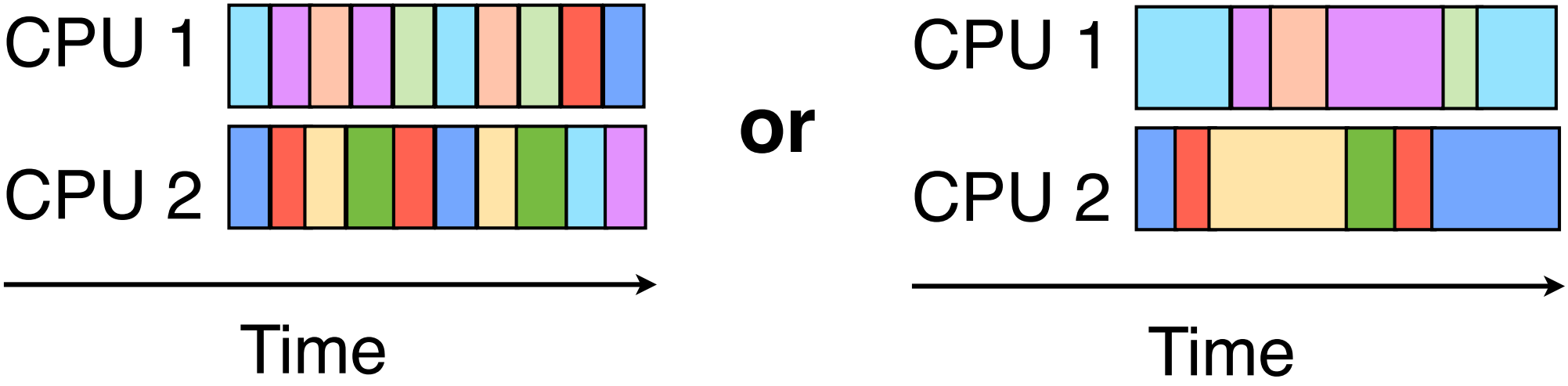
Who decides the order?

- Take a look at **`threads.scheduling.ManyThreads.java`**

- What happens when you run the code?
- Do you get the same ordering as your neighbor?
- What if you change the number of iterations?
- Or the number of threads?
- Is it the same every time you repeat?

Scheduling Threads

- Ordering may be very random



- Threads can explicitly relinquish the CPU
- **Scheduler** can interrupt and pick someone else
- Do not assume a particular ordering!!

Web Server

- Server Pseudocode:

```
Create a ServerSocket
while (true):
    Wait for client to connect
    Setup new socket for clie
    start new thread to do
```

```
Read request from
client
Prepare response
Send response to
client
close socket for
client
```

- What happens if preparin
  slow? What if there are r

Web Server

- Server Pseudocode:

```
Create a ServerSocket
while (true):
    Wait for client to connect
    Setup new socket for client
    Read request from client
    Prepare response
    Send response to client
    close socket for client
```

```
serverSocket server = new
ServerSocket(portnum);
```

```
Socket sock =
serverSocket.accept();
```

```
// Use BufferedReader
and PrintWriter classes
```

- What happens if preparing a response is very slow? What if there are many clients?

Multi-Threaded Server

- Main thread
  - Creates server sockets and waits for clients to connect
  - Start a new worker thread to process each client
- Worker thread
  - Needs a new class that implements Runnable
  - run() method processes a single request from the client

```
Create a ServerSocket
while (true):
    Wait for client to connect
    Setup new socket for client
    Read request from client
    Prepare response
    Send response to client
    close socket for client
```

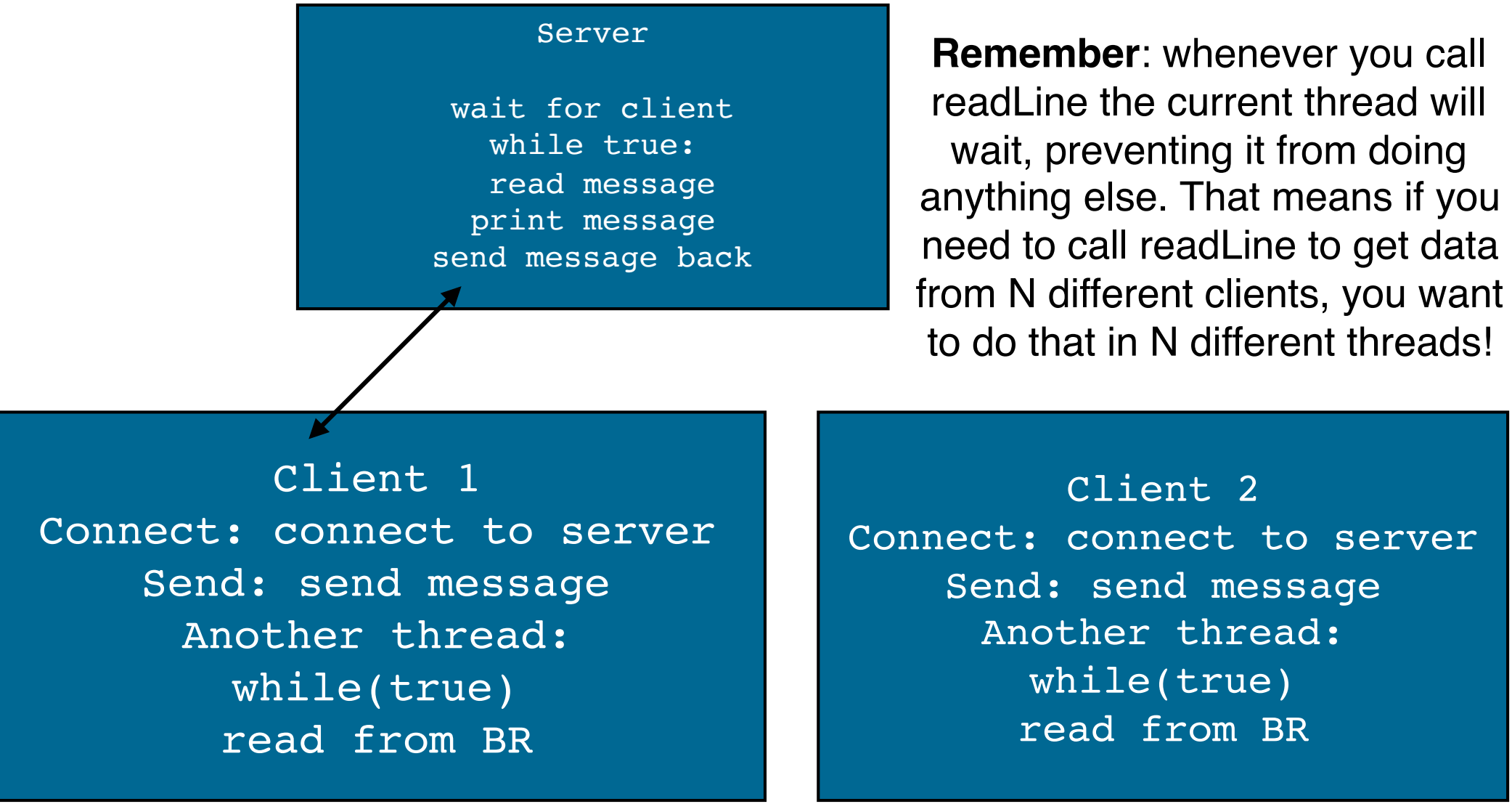
Do these in a thread!

Threaded Web Server

- Work with a neighbor
- Look at the **threads.web.SlowWebServer** class
  - How does it work?
  - Why is it slow?
  - Have one person run it and then both connect at the same time with a web browser
  - Use the **FrameBrowser** class to load the page and measure the time it takes. Does the time depend on whether another request is active?
- Make the Web Server use threads
  - Create and start a new thread for every new user

Threads in Networking

- Suppose you want to build a chat server



Remember: whenever you call readLine the current thread will wait, preventing it from doing anything else. That means if you need to call readLine to get data from N different clients, you want to do that in N different threads!