1.00/1.001/1.002
Introduction to Computers and Engineering Problem Solving

Recitation 10
Threads

April 30 & May 1 2012

Processes vs. Threads

Both terms describe interleaving operations to make it appear that multiple things are happening in parallel

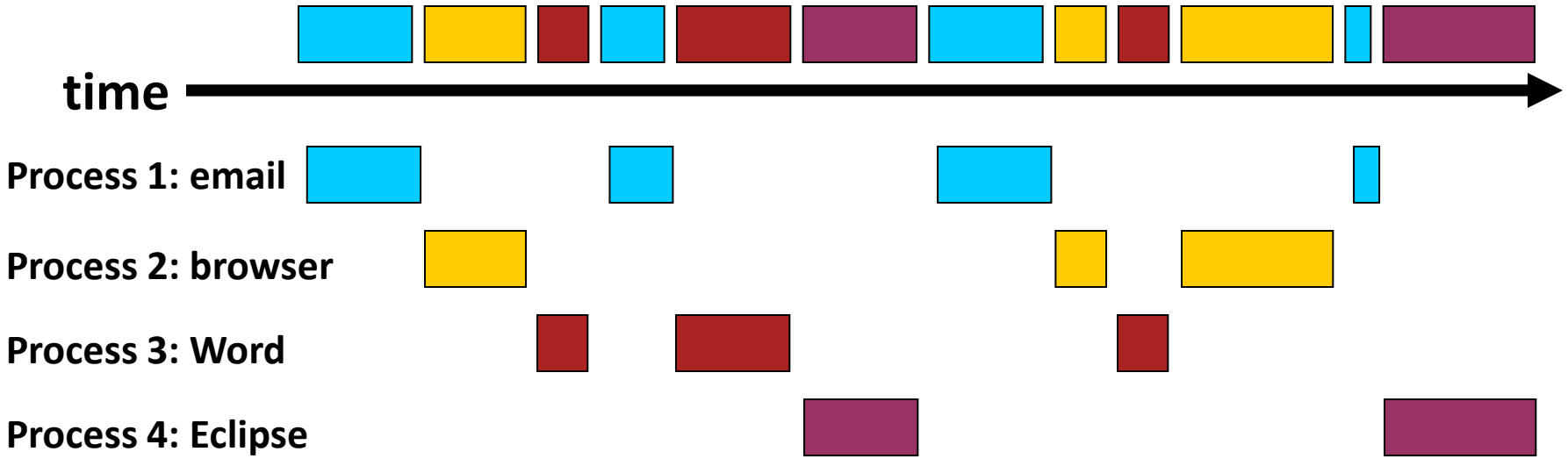
What is the difference?

First, What is an Operating System (OS)?

- The MOTHER of all programs.
- It helps standardize how other programs (applications) can use computer resources.
- It schedules which programs run when.

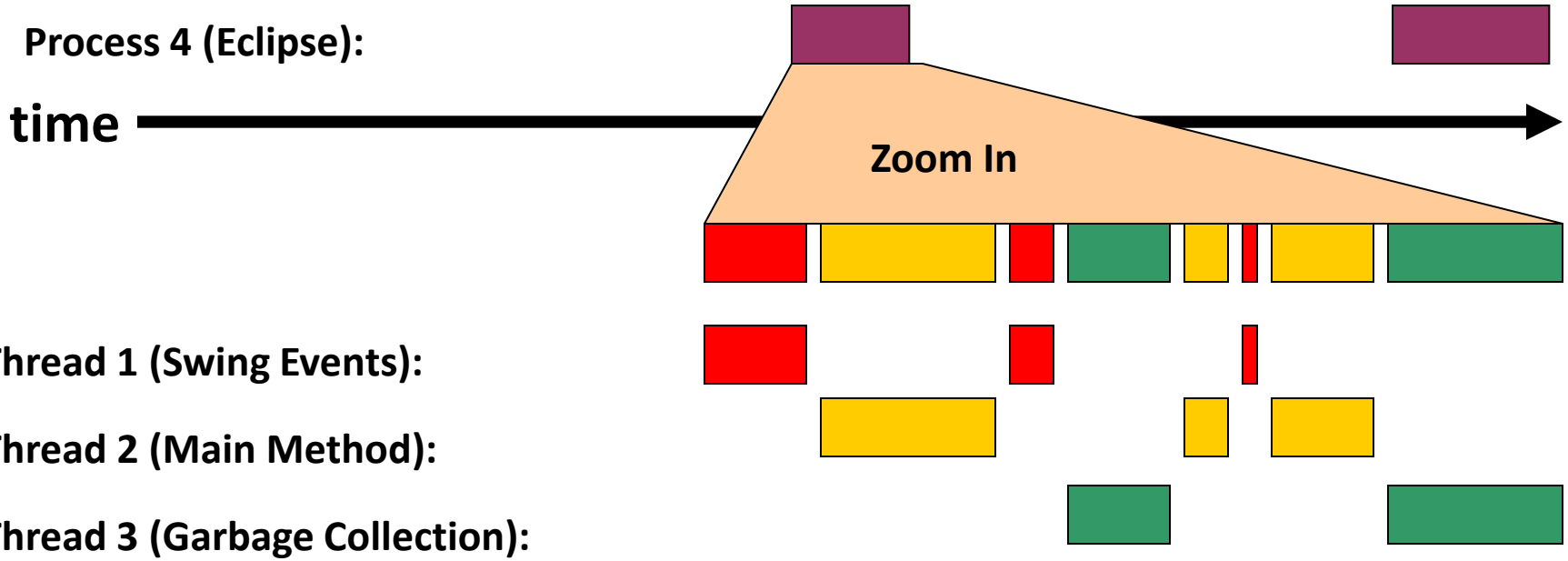
Processes

- An instance of a computer program being executed. Managed by the OS.
- Insular: Each process uses its own resources and has a reasonable amount of processing time. Most modern operating systems prevent communication between independent processes.
- Safe: Separation of memory spaces allows one program to continue running even if another program writes garbage into its memory space (and crashes). Insulation can also prevent spreading viruses.



Threads

- If a process is an operating system's way of interleaving programs, a thread is a program's way of interleaving sections of code.
- Not independent: Multiple threads can run within the same process and share resources such as memory while different processes don't share resources.
- On a multi-core or multiprocessor system (most computers today) the threads or tasks will actually run at the same time in parallel! On a single processor the processor switches between different threads frequently enough that the user perceives that the threads are running at the same time. In the diagram below for example, thread 2 and thread 3 can be running at the same time, so the yellow and green boxes can overlap.



Threads: Analogy

Cookbook:

- A set of cooking instructions

Cooking from the cookbook:

- One or more cooks following the cookbook

Cook:

- Multiple cooks can cook from the same cookbook
- Each cook is following a set of cooking instructions - not necessarily on the same page
- Can use the same utensils/food
- Can cook in parallel

Image removed due to copyright restrictions.

Computer program:

- A set of computer instructions

Process:

- An instance of a computer program running

Thread:

- Multiple threads can run in the same process
- Each thread executing a set of program instructions - not necessarily same section of code
- Can share the program resources
- Can execute in parallel (multi-core processor)

Image removed due to copyright restrictions.

Need to synchronize!

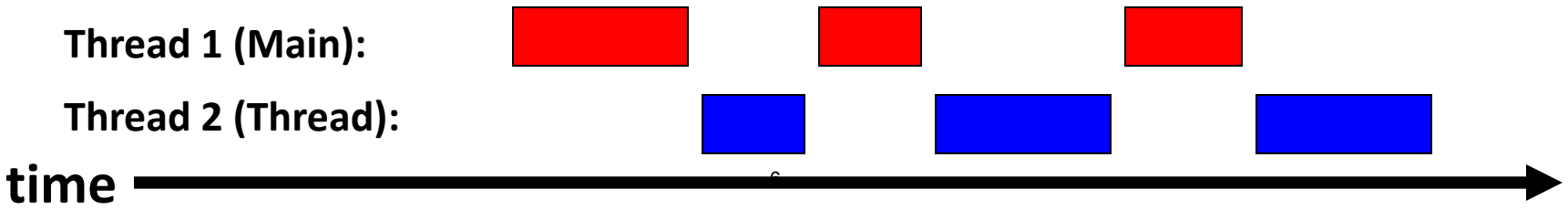
Creating your own Thread (extend class `Thread`)

```
public class MyThread extends Thread{
        public void run(){
                        // code executed in the Thread goes here}
        }

        public static void main(String[] args){
                Thread t = new MyThread()
                t.start(); // start the thread
                // code executed in Main goes here
        }
}
```

Thread

Main



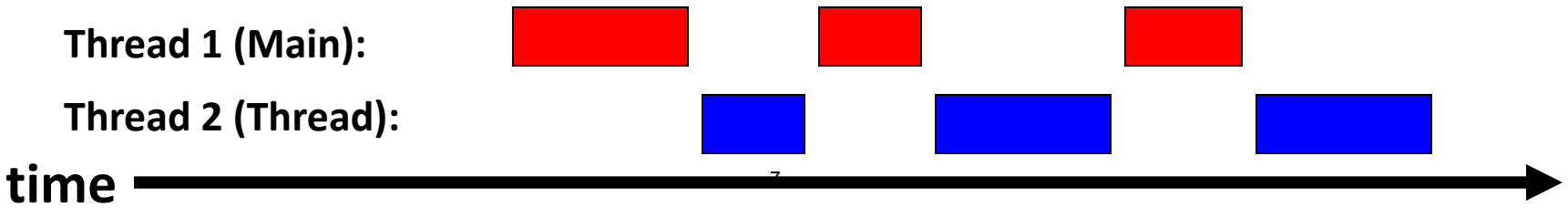
Creating your own Thread (implement interface `Runnable`)

```
public class MyThread implements Runnable{
    public void run(){
        // code executed in the Thread goes here
    }

    public static void main(String[] args){
        Thread t = new Thread(new MyThread());
        t.start(); // start the thread
        // code executed in Main goes here
    }
}
```

Thread

Main



class `Thread` instance methods

If you create a new thread, i.e.:

```
Thread t = new Thread();
```

These are some methods you can call in the Main thread (OR any thread but thread `t`):

`t.start()`

start thread t by executing the code in method `run()`

`t.isAlive()`

returns true or false, whether thread t still exists.

`t.join()`

Halt calling thread execution until thread t completes.

Thread.sleep

static void **sleep** (long millis)

- Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

```
public class SleepMessages{
   public static void main(String args[]){
      String importantInfo[] = {
         "1.00 Final Review Session May 18",
         "1.00 Final May 18", "Need to return Phidgets"};

      for (int i = 0; i < importantInfo.length; i++) {
         //Pause for 4 seconds
                  try{
                  Thread.sleep(4000);
                  }Catch(Exception e){
                        System.err.println(e);
                  }
         //Print a message
         System.out.println(importantInfo[i]);
      }
   }
}
```

These sleep times are not guaranteed to be precise because they are limited by the facilities provided by the underlying OS. Also, the sleep period can be terminated by interrupts. You cannot assume that invoking sleep will suspend the thread for precisely the time period specified.

Threads Example (Bad one)

```
public class Model {
        private int count;
        public void increment() {count++;}
        public static void main(String [] ars){
                Model m = new Model();
                Thread a = new MyThread(m);
                Thread b = new MyThread(m);
                a.start(); b.start();
                try{ a.join(); b.join(); }
                catch(Exception e){
                        System.err.print(e);
                }
                System.out.println(m.count);
        }
}
```

```
public class MyThread extends Thread{
        private Model m;

        public MyThread(Model m){
        this.m = m;
        }

        public void run(){
                for(int i=0;i!=10000;i++)
                m.increment();
        }
}
```

Try to run the program a few times? Do you always see 20,000? Why?

Thread Synchronization

The application of particular mechanisms to ensure that two concurrently executing threads do not execute specific portions of a program at the same time preventing problems such as below:

Thread Interference: happens when two operations running in different threads but acting on the same data, *interleave*. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

Memory consistency: occurs when different threads have inconsistent views of what should be the same data like the bank account example in lecture.

Java has **synchronized** keyword to avoid such problems:

```
public synchronized void compute() {
    // body of method
}
```

- compute() cannot be interrupted by another synchronized method acting on the same object.
- If a second thread attempts to execute another synchronized method on the same object, the second thread will wait until the first synchronized method exits.

Threads Example (With synchronized)

```
public class Model {
        private int count;
        public synchronized void increment()
                {count++;}
        public static void main(String [] ars){
                Model m = new Model();
                Thread a = new MyThread(m);
                Thread b = new MyThread(m);
                a.start(); b.start();
                try{ a.join(); b.join(); }
                catch(Exception e){
                        System.err.print(e);
                }
                System.out.println(m.count);
        }
}
```

```
public class MyThread extends Thread{
        private Model m;

        public MyThread(Model m){
        this.m = m;
        }

        public void run(){
                for(int i=0;i!=10000;i++)
                m.increment();
        }
}
```

Threads and Swing

- With very few exceptions, the Swing classes expect to have their methods called only from the event thread.
- Sometimes you want to update the information in a GUI from another thread (e.g., the main thread, a sensor thread, the CustomThread you just wrote).
- To do this create an object that describes a task to be performed in the event thread at some future time.

Ex:
```
JPanel component = …
Runnable update = new Runnable() {
        public void run() {
        component.doSomething();    // Task
};
```

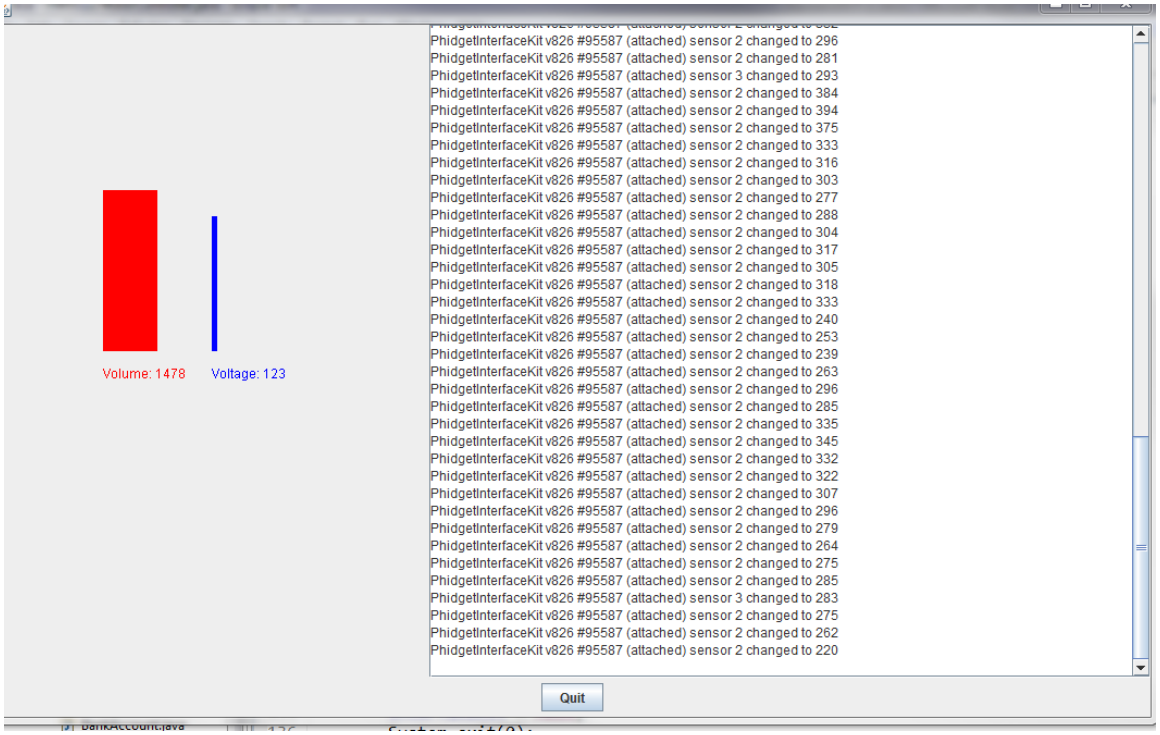
- Then pass that object to the event thread using a synchronized method that places it in the event thread's queue.

```
SwingUtilities.invokeLater( update );
```

invokeLater() is a synchronized static method in the SwingUtilities class in the javax.swing package. It inserts the task in the event queue.

- Swing will execute the task when it can.

Homework 9: Dam Management Threads and Sensors



- Control reservoir volume and generated voltage from sensors
- Create threads that random inject/withdraw reservoir water
- Display results (shown above) using Swing

Homework 9: Dam Management Threads and Sensors

- Multiple threads
  - Water injection/withdraw threads
  - Sensor change event is a separate thread
  - Swing runs in its own thread
  - Main thread

- Use of *Synchronization* keyword
  - Figure out which method(s) are called concurrently from multiple threads
  - Interleave calls to these method(s) using synchronization

- Use *invokeLater()* method to "post" jobs to Swing dispatching queue from non-Swing thread such as the sensor thread

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving

Spring 2012