CMSC 132:
Object-Oriented Programming II

Threads in Java

Department of Computer Science

University of Maryland, College Park

Problem

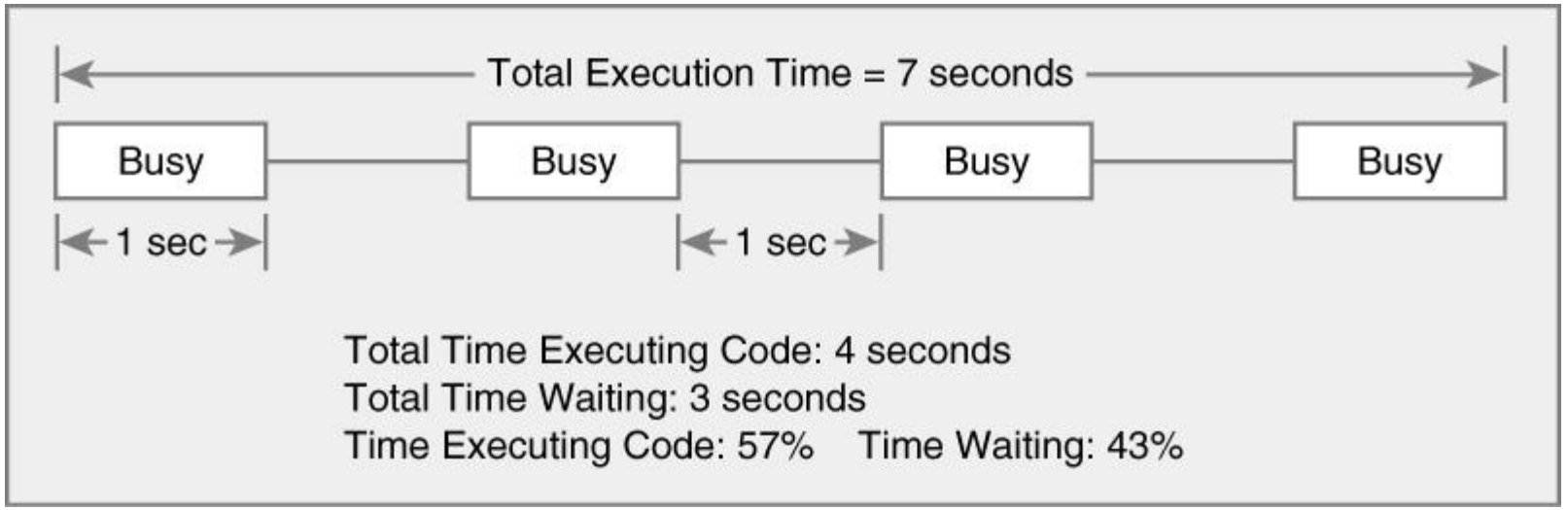
- Multiple tasks for computer
  - Draw & display images on screen
  - Check keyboard & mouse input
  - Send & receive data on network
  - Read & write files to disk
  - Perform useful computation (editor, browser, game)
- How does computer do everything at once?
  - Multitasking
  - Multiprocessing

Multitasking (Time-Sharing)

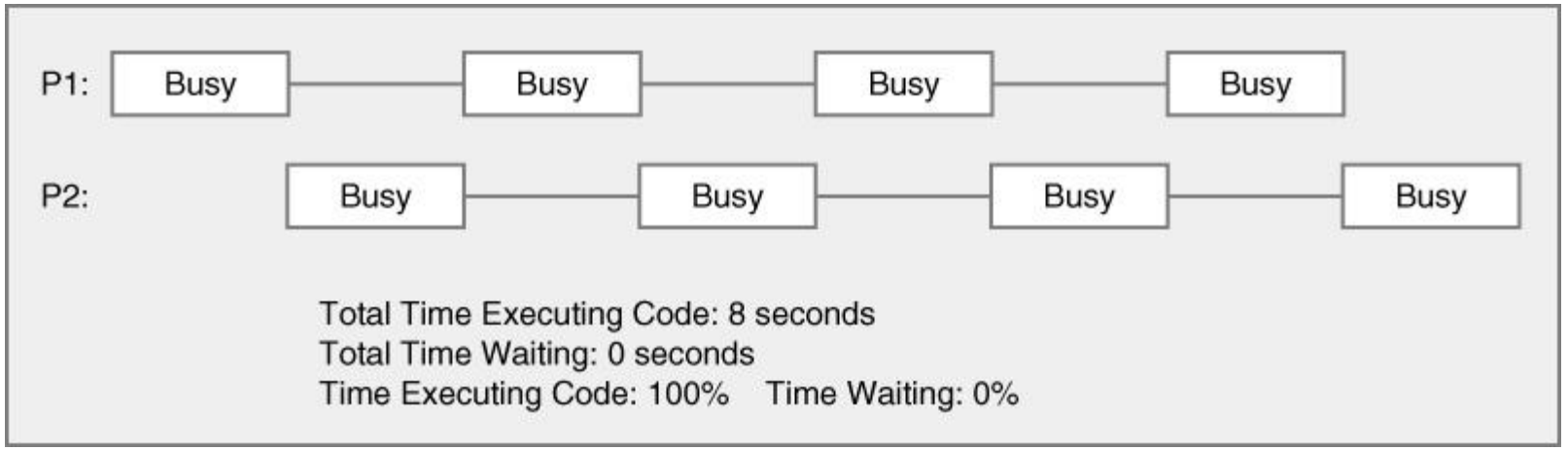
- **Approach**
  - **Computer does some work on a task**
  - **Computer then quickly switch to next task**
  - **Tasks managed by operating system (scheduler)**
- **Computer seems to work on tasks concurrently**
- **Can improve performance by reducing waiting**

Multitasking Can Aid Performance

- **Single task**



- **Two tasks**



Multiprocessing (Multithreading)

- **Approach**
  - **Multiple processing units (multiprocessor)**
  - **Computer works on several tasks in parallel**
  - **Performance can be improved**

Dual-core AMD Athlon X2

32 processor Pentium Xeon

4096 processor Cray X1

Beowulf computer cluster (Borg, 52-node cluster used by McGill University Image/Info from Wikipedia)

Perform Multiple Tasks Using…

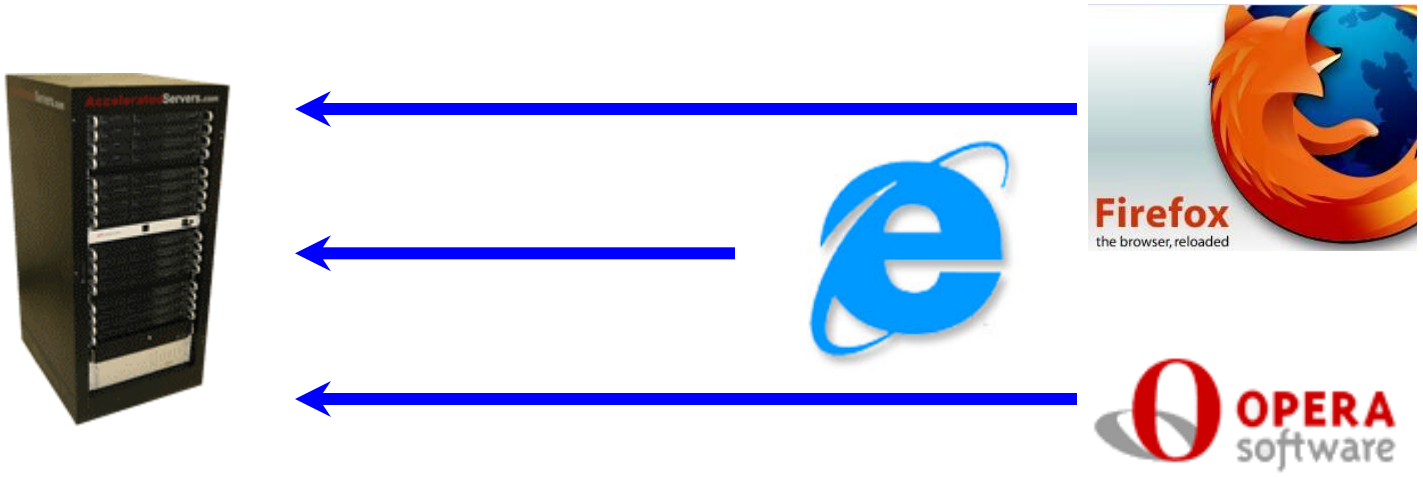
1. **Process**
   - Definition – executable program loaded in memory
   - Has own address space
     - Variables & data structures (in memory)
   - Each process may execute a different program
   - Communicate via operating system, files, network
   - May contain multiple threads

Perform Multiple Tasks Using…

2. **Thread**
   - Definition – sequentially executed stream of instructions
   - Shares address space with other threads
   - Has own execution context
     - Program counter, call stack (local variables)
   - Communicate via shared access to data
   - Multiple threads in process execute same program
   - Also known as “lightweight process”
   - Let’s see how memory is organized for a threaded environment

Motivation for Multithreading

1. **Captures logical structure of problem**
   - **May have concurrent interacting components**
   - **Can handle each component using separate thread**
   - **Simplifies programming for problem**
- **Example**



Web Server uses threads to handle …

Multiple simultaneous web browser requests

Motivation for Multithreading

2. **Better utilize hardware resources**
   - When a thread is delayed, compute other threads
   - Given extra hardware, compute threads in parallel
   - Reduce overall execution time
- **Example**

Multiple simultaneous web browser requests…

Handled faster by multiple web servers

Multithreading Overview

- Motivation & background
- Threads 
  - Creating Java threads
  - Thread states
  - Scheduling
- Synchronization
  - Data races
  - Locks
  - Deadlock

Programming with Threads

- **Concurrent programming**
  - **Writing programs divided into independent tasks**
  - **Tasks may be executed in parallel on multiprocessors**

- **Multithreading**
  - **Executing program with multiple threads in parallel**
  - **Special form of multiprocessing**

Creating Threads in Java

- **Need to specify work performed by thread**

- **Two approaches**
  1. **Runnable interface (Preferred Approach)**

     ```
     public interface Runnable {
         public void run();   // work ⇒ thread
     }
     ```

  2. **Extending Thread class**

     ```
     public class Thread extends Object { … }
     ```

Thread Class

```
public class Thread extends Object
                            implements Runnable {
    public Thread();
    public Thread(String name);   // Thread name
    public Thread(Runnable R);    // Thread ⇒ R.run()
    public Thread(Runnable R, String name);

    public void run();     // if no R, work for thread
    public void start();   // begin thread execution
    ...
}
```

Creating Threads in Java

- **Runnable Approach**
  1. Define class implementing Runnable interface

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

  2. Put work to be performed in run( ) method
  3. Create instance of the “worker” class
  4. Create thread to run it
     - Create Thread object
       - Pass worker object to Thread constructor
     - Or hand the worker instance to an executor
       - Alternative methods for running threads

Creating Threads in Java

- **Example**

```
public class MyT implements Runnable {
    public void run( ) {
        …   // work for thread
    }
}
Thread t = new Thread(new MyT( ));  // create thread
t.start();                          // begin running thread
…                                   // thread executing in parallel
```

- **See Message example in code distribution**

Alternative Thread Creation Approach

- **Thread Class Approach**
  - Extend Thread class and override run method
  - Not recommended
- **Example**

```
public class MyT extends Thread {
    public void run( ) {
        …                          // work for thread
    }
}
MyT t = new MyT( ) ;               // create thread
t.start( );                        // begin running thread
…                                  // thread executing in parallel
```

- **See Message example in code distribution**

Why Not Recommended?

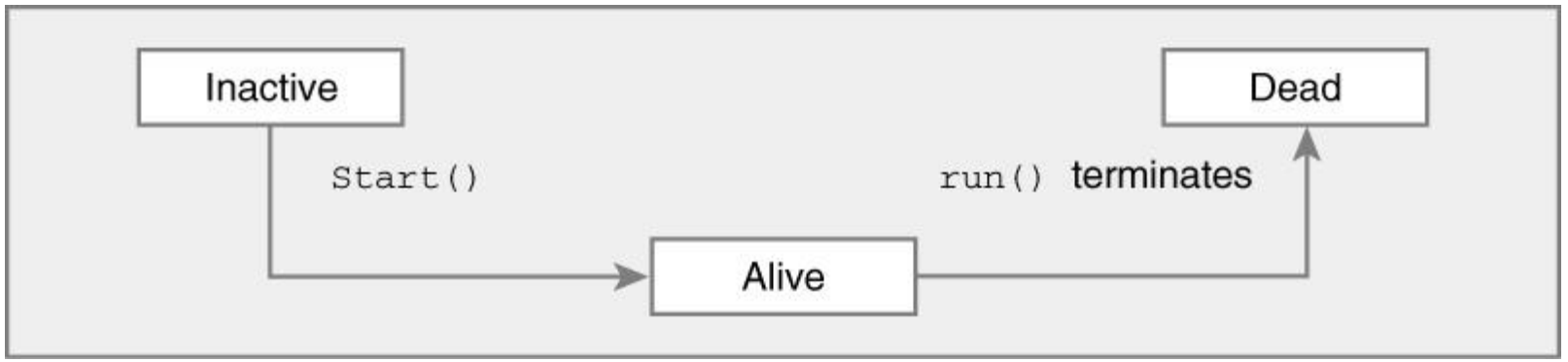
- Not a big problem for getting started
  - But a bad habit for industrial strength development
- Methods of worker and Thread class intermixed
- Hard to migrate to more efficient approaches
  - Thread Pools

More Thread Class Methods

```
public class Thread extends Object {
    …
    public static Thread currentThread()
    public String getName()
    public void interrupt()
    public boolean isAlive()
    public void join()
    public void setDaemon()
    public void setName()
    public void setPriority()
    public static void sleep()
    public static void yield()
}
```

Creating Threads in Java

- **Note**
  - **Thread eventually starts executing only if start() is called**



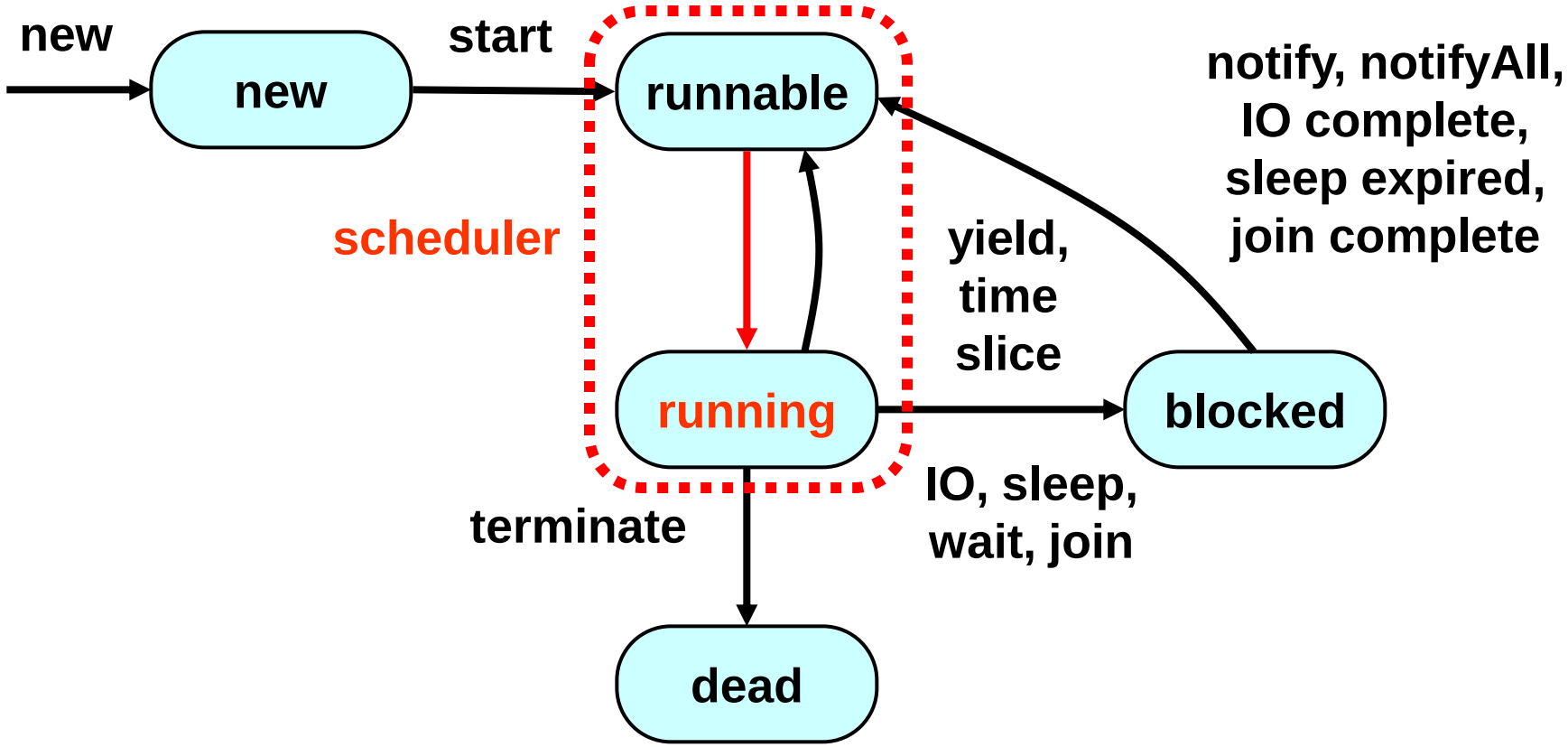
  - **Runnable is interface**
    - **So it can be implemented by any class**
    - **Required for multithreading in applets**
  - **Do not call the run method directly**

Threads – Thread States

- Java thread can be in one of these states
  - New – thread allocated & waiting for start()
  - Runnable – thread can begin execution
  - Running – thread currently executing
  - Blocked – thread waiting for event (I/O, etc.)
  - Dead – thread finished
- Transitions between states caused by
  - Invoking methods in class Thread
    - new(), start(), yield(), sleep(), wait(), notify()…
  - Other (external) events
    - Scheduler, I/O, returning from run()…
- In Java states defined by Thread.State

Threads – Thread States

- State diagram



Running is a logical state → indicates runnable thread is actually running

Daemon Threads

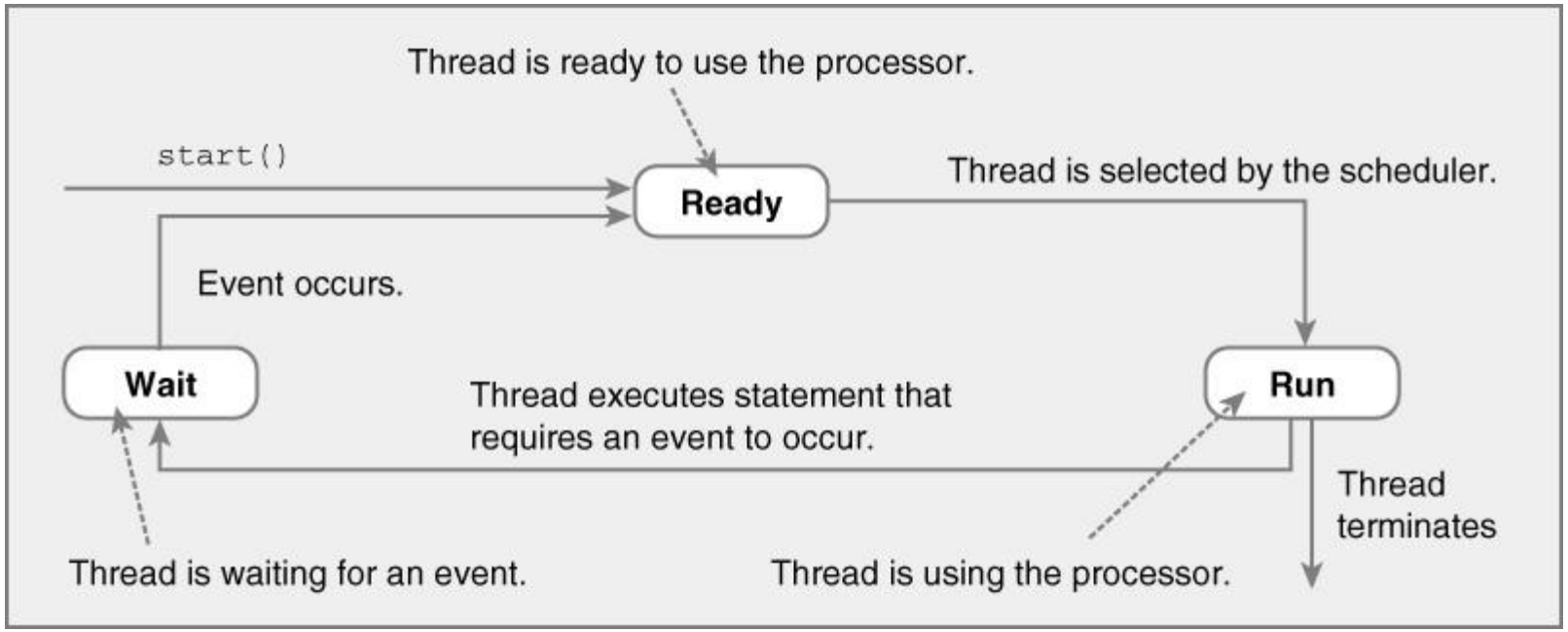
- **Java threads types**
  - **User**
  - **Daemon**
    - **Provide general services**
    - **Typically never terminate**
    - **Call setDaemon() before start()**
- **Program termination**
  1. **All user threads finish**
  2. **Daemon threads are terminated by JVM**

Threads – Scheduling

- **Scheduler**
  - **Determines which runnable threads to run**
  - **Can be based on thread priority**
  - **Part of OS or Java Virtual Machine (JVM)**
- **Scheduling policy**
  - **Nonpreemptive (cooperative) scheduling**
  - **Preemptive scheduling**

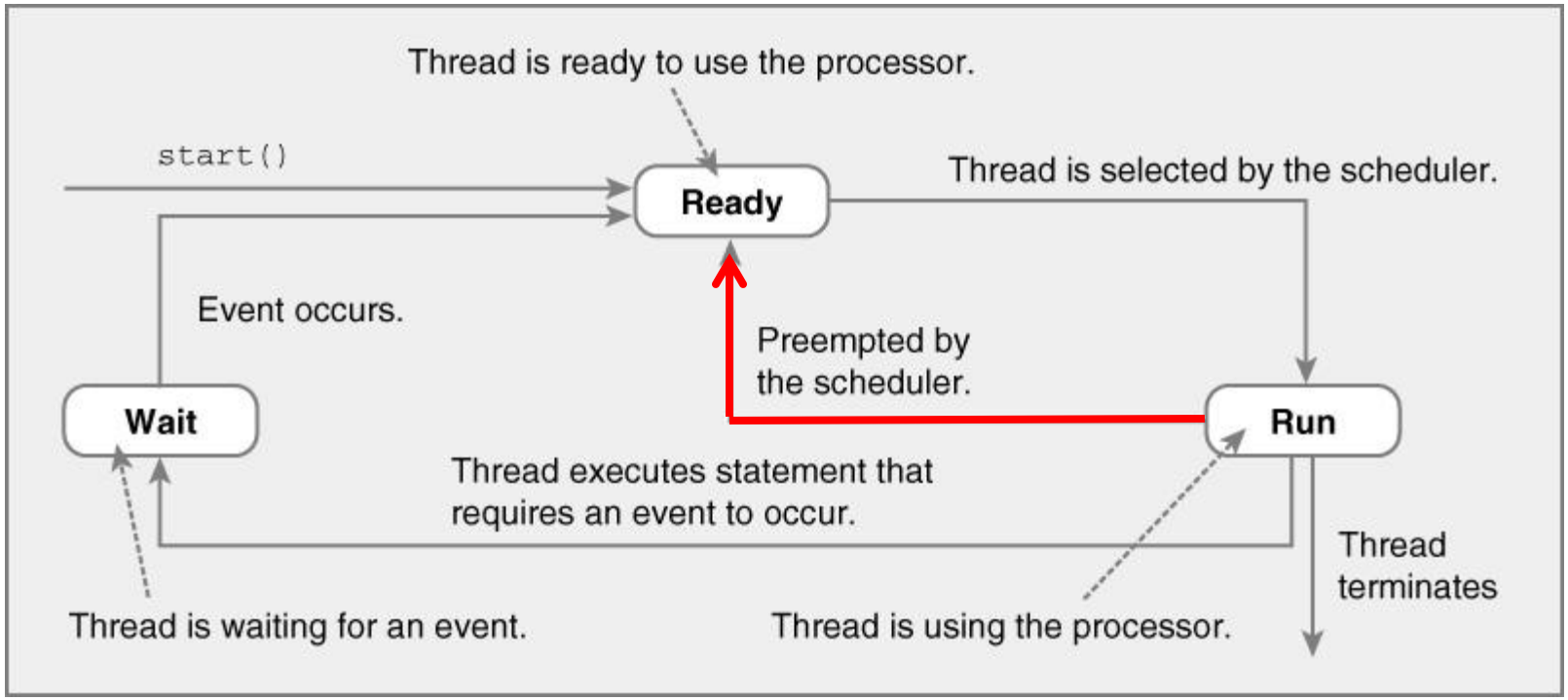
Threads – Non-preemptive Scheduling

- **Threads continue execution until**
  - **Thread terminates**
  - **Executes instruction causing wait (e.g., IO)**
  - **Thread volunteering to stop (invoking yield or sleep)**



Threads – Preemptive Scheduling

- **Threads continue execution until**
  - **Same reasons as non-preemptive scheduling**
  - **Preempted by scheduler**

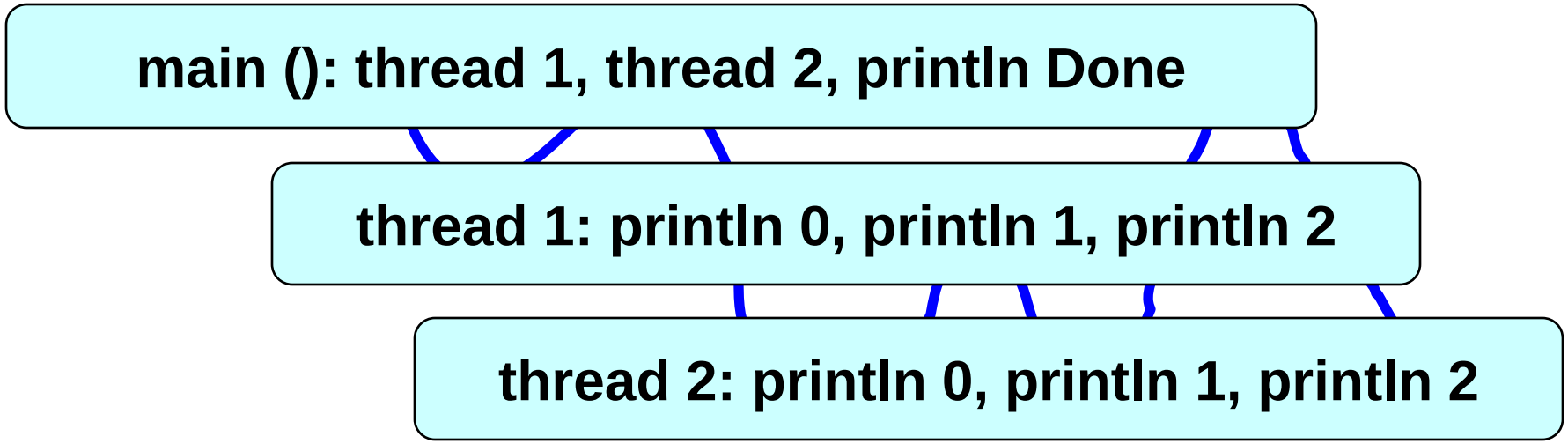


Java Thread Example

```
public class ThreadNoJoin extends Thread {
   public void run() {
      for (int i = 0; i < 3; i++) {
         try {
            sleep((int)(Math.random() * 5000)); // 5 secs
         } catch (InterruptedException e) { e.printStackTrace(); }
          System.out.println(i);
       }
   }
   public static void main(String[] args) {
      Thread t1 = new ThreadNoJoin();
      Thread t2 = new ThreadNoJoin();
      t1.start();
      t2.start();
      System.out.println("Done");
   }
}
```

Java Thread Example – Output

- **Possible outputs**
  - 0,1,2,0,1,2,Done // thread 1, thread 2, main()
  - 0,1,2,Done,0,1,2 // thread 1, main(), thread 2
  - Done,0,1,2,0,1,2 // main(), thread 1, thread 2
  - 0,0,1,1,2,Done,2 // main() & threads interleaved



Thread Class – join( ) Method

- **Can wait for thread to terminate with join( )**

- **Method prototype**
  - **public final void join( )**
    - **Returns when thread is done**
    - **Throws InterruptedException if interrupted**

Java Thread Example (Join)

```
public class ThreadJoin extends Thread {
        public void run() {
           for (int i = 0; i < 3; i++) {
              try {
                 sleep((int)(Math.random() * 5000)); // 5 secs
              } catch (InterruptedException e) { e.printStackTrace(); }
              System.out.println(i);
           }
        }

        public static void main(String[] args) {
           Thread t1 = new ThreadJoin();
           Thread t2 = new ThreadJoin();
           t1.start();
           t2.start();
           try {
              t1.join();
              t2.join();
           } catch (InterruptedException e) { e.printStackTrace(); }
           System.out.println("Done");
        }
}
```

About Join

- **Important: You will limit the concurrency level if you do not start/join correctly**
- **Suppose you want to run many threads concurrently. Start them all and then execute the join for each one. Do not start one thread, then join on that thread, start the second one, join on that thread, etc.**
- **Feel free to use arrays, sets, etc. to keep track of your threads**

Thread Scheduling Observations

- Order thread is selected is indeterminate
  - Depends on scheduler
- Scheduling may not be fair
  - Some threads may execute more often
- Thread can block indefinitely (starvation)
  - If other threads always execute first

About Threads

- **Common mistake → calling the run() method. If you want to run a thread you must execute start() and not call the run() method; the run() method is called for you.**
- **Thread.sleep → Suppose you have a thread object reference (t1) and invoke t1.sleep(2000). Which thread will be sleeping for 2 seconds? It will not be t1.**

Thread Example

- Swing uses a single-threaded model
- Long computations in the EDT freezes the GUI
- Example: Progress Bar Example