

Multithreading in Java



CMSC 132

**Department of Computer Science
University of Maryland, College Park**

Overview

- **Multitasking**
- **Multiprocessing**
- **Motivation for multithreading**
- **Threads**
 - **Creating Java threads**
 - **Thread states**
 - **Scheduling**

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

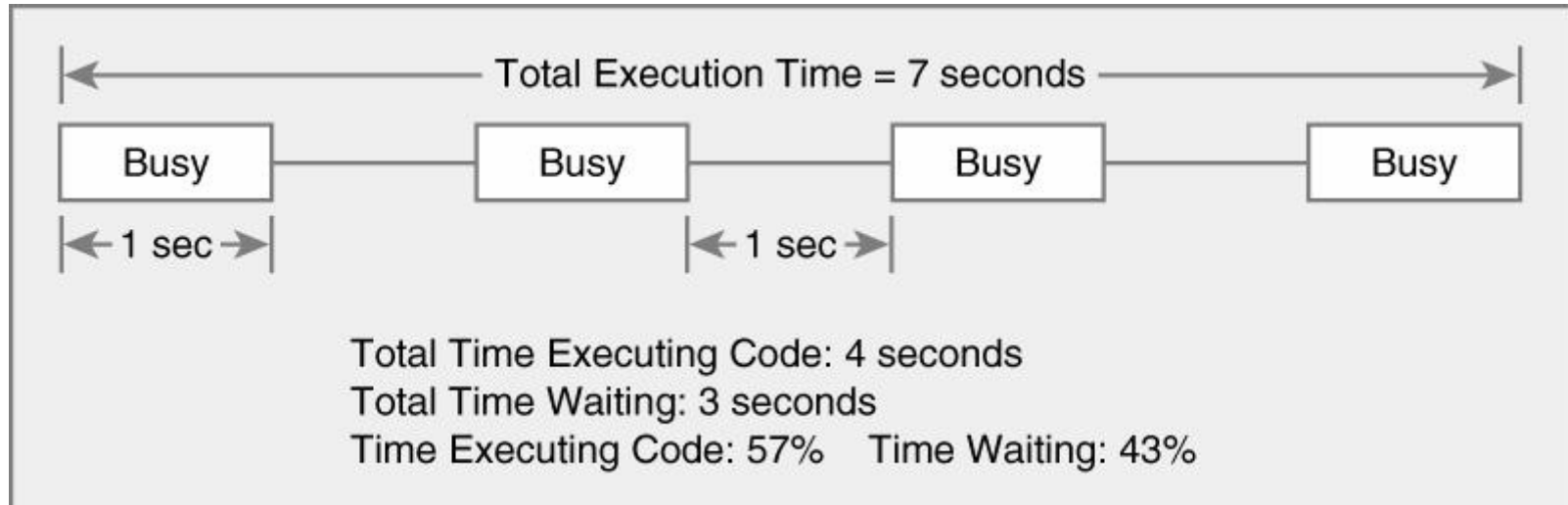
Multitasking (Time-Sharing)

■ Approach

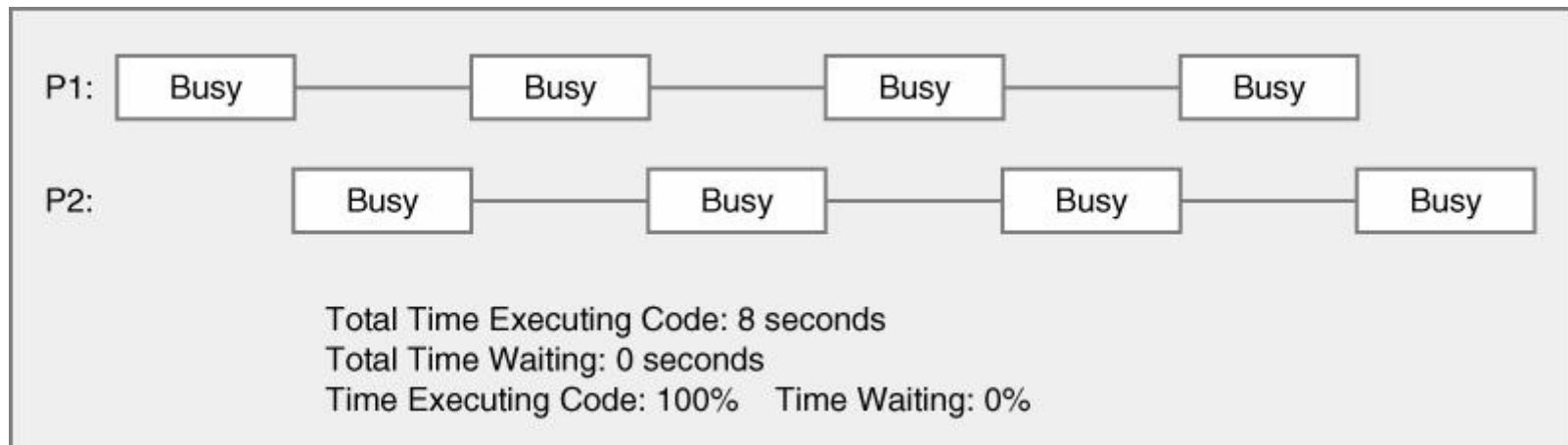
- Computer does some work on a task
 - Computer then quickly switch to next task
 - Tasks managed by operating system (scheduler)
- Computer **seems** to work on tasks concurrently
- Can improve performance by reducing waiting

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

Perform Multiple Tasks Using...

■ 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...

■ Thread

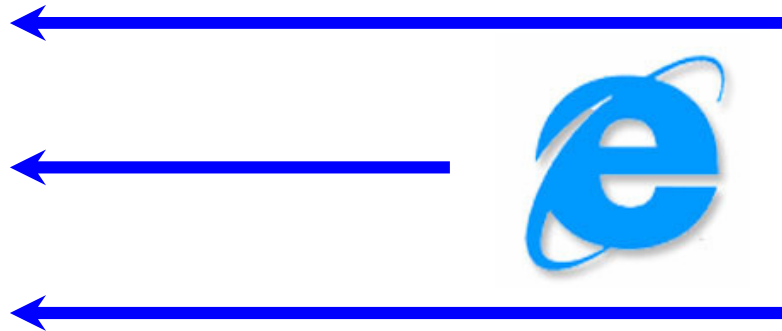
- Definition – sequentially executed stream of instructions
- Shares address space with other threads
- Has own execution context
 - Program counter, call stack (local variables)
- Communicate via shared access to data
- Multiple threads in process execute same program
- Also known as “lightweight process”

Motivation for Multithreading

- **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 ...



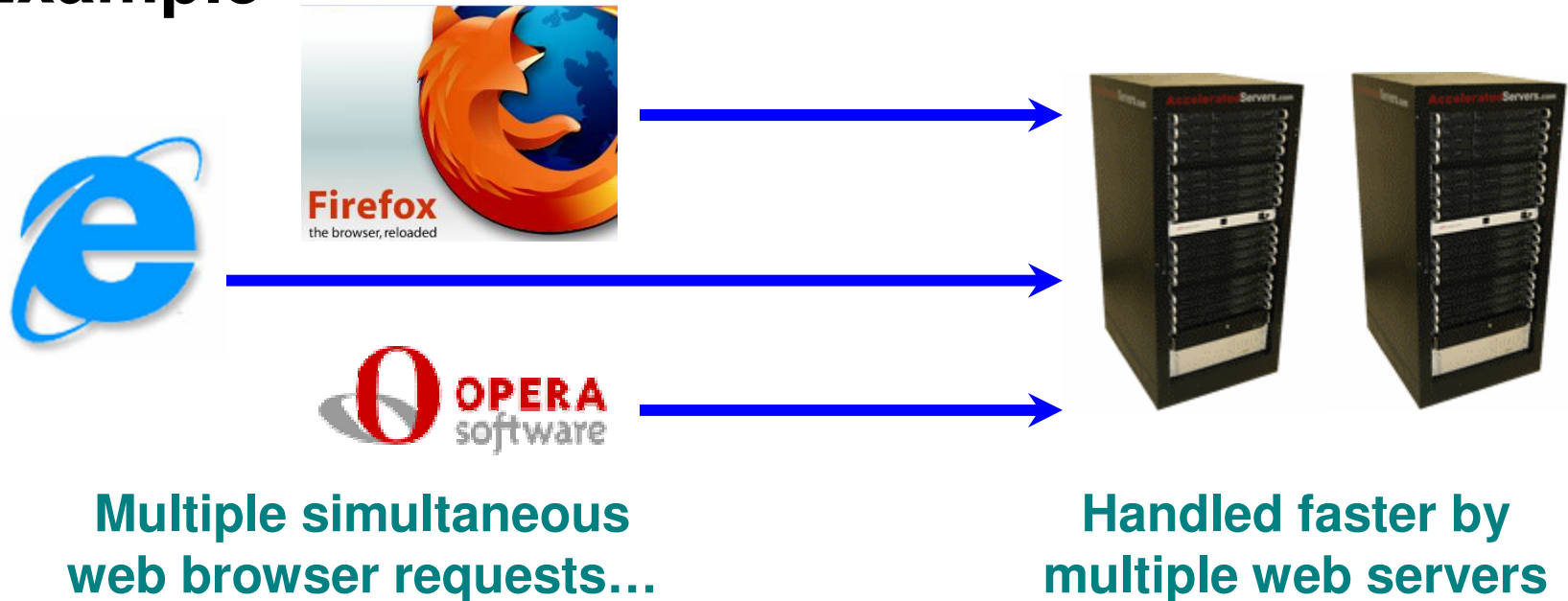
Multiple simultaneous
web browser requests

Motivation for Multithreading

■ Better utilize hardware resources

- When a thread is delayed, compute other threads
- Given extra hardware, compute threads in parallel
- Reduce overall execution time

■ Example



Multithreading Overview

- **Motivation & background**

- **Threads**

- **Creating Java threads**
- **Thread states**
- **Scheduling**

- **Synchronization**

- **Data races**
- **Locks**
- **Wait / Notify**

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

Thread Class

```
public class Thread {  
  
    public Thread(Runnable R); // Thread ⇒ R.run()  
    public Thread(Runnable R, String name);  
  
    public void start(); // begin thread execution  
    ...  
}
```

More Thread Class Methods

```
public class Thread {  
    ...  
    public String getName();  
    public void interrupt();  
    public boolean isAlive();  
    public void join();  
    public void setDaemon(boolean on);  
    public void setName(String name);  
    public void setPriority(int level);  
  
    public static Thread currentThread();  
  
    public static void sleep(long milliseconds);  
    public static void yield();  
}
```

Creating Threads in Java

- You have to specify the work you want the thread to do
- Define a class that implements the Runnable interface

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

- Put the work in the run method
- Create an instance of the worker class and create a thread to run it
 - or hand the worker instance to an executor

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
```

Java Thread Example

```
public class ThreadExample implements Runnable {  
    public void run() {  
        for (int i = 0; i < 3; i++)  
            System.out.println(i);  
    }  
    public static void main(String[] args) {  
        new Thread(new ThreadExample()).start();  
        new Thread( new ThreadExample()).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

main (): thread 1, thread 2, println Done

thread 1: println 0, println 1, println 2

thread 2: println 0, println 1, println 2

Alternative (Not Recommended)

■ Directly extend Thread class

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

Why Not Recommended?

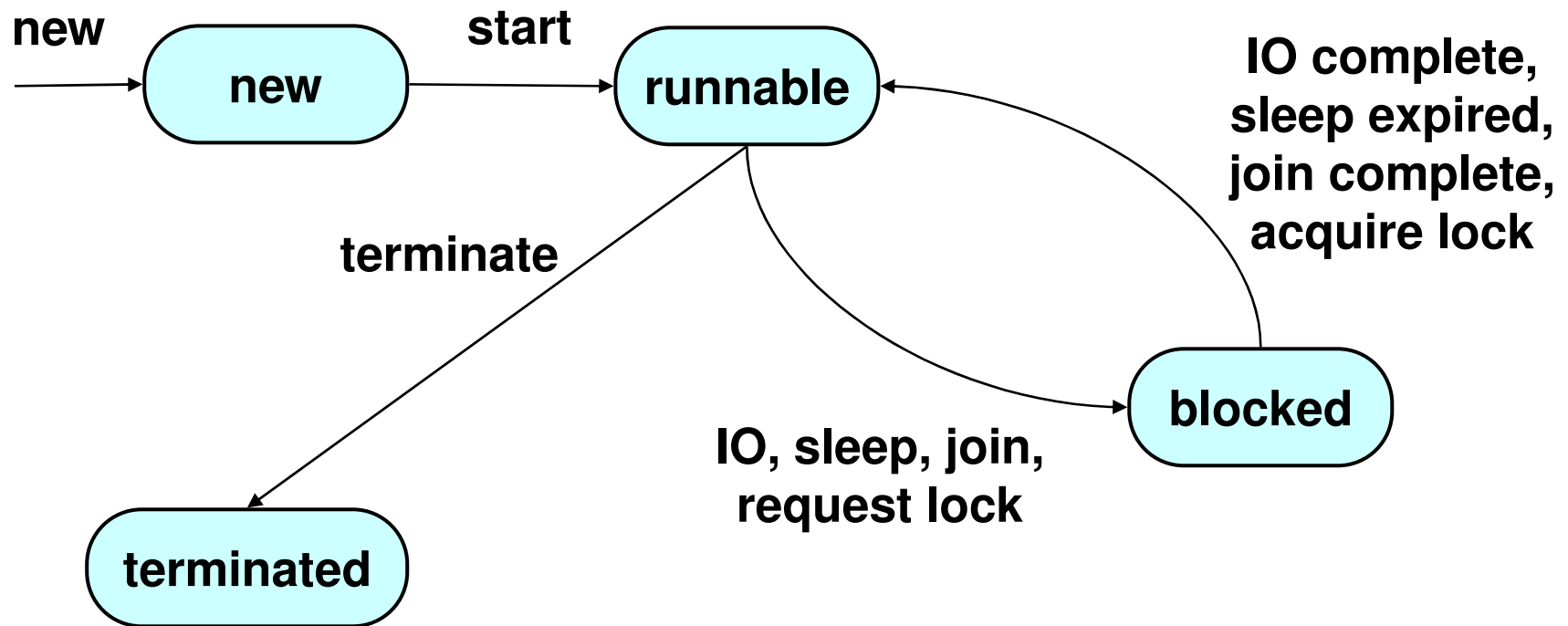
- **Not a big problem for getting started**
 - **but a bad habit for industrial strength development**
- **The methods of the worker class and the Thread class get all tangled up**
- **Makes it hard to migrate to Thread Pools and other more efficient approaches**

Threads – Thread States

- **Java thread can be in one of these states**
 - **New** – thread allocated & waiting for start()
 - **Runnable** – thread can execute
 - **Blocked** – thread waiting for event (I/O, etc.)
 - **Terminated** – thread finished
- **Transitions between states caused by**
 - **Invoking methods in class Thread**
 - **start(), yield(), sleep()**
 - **Other (external) events**
 - **Scheduler, I/O, returning from run()...**

Threads – Thread States

■ State diagram



Threads – Scheduling

■ Scheduler

- Determines which runnable threads to run
- Can be based on thread **priority**
- Part of OS or Java Virtual Machine (JVM)
- Many computers can run multiple threads simultaneously (or nearly so)

Thread Scheduling Observations

- Order thread is selected is **indeterminate**
 - Depends on scheduler, timing, chance
- Scheduling is not guaranteed to be fair
- Some schedules/interleavings can cause unexpected and bad behaviors
- Synchronization
 - can be used to control thread execution order

Daemon Threads

■ Java threads types

- User

- Daemon

 - Provide general services

 - Typically never terminate

 - Call `setDaemon()` before `start()`

■ Program termination

- If all non-daemon threads terminate, JVM shuts down