CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Threads in Java

Department of Computer Science
University of Maryland, College Park

Daemon Threads

- **Java threads types**
  - User
  - Daemon
    - Provide general services
    - Typically, never terminate
    - To set as daemon thread, call setDaemon() before start()
- **Program termination**
  - All user threads finish
  - Daemon threads are terminated by JVM

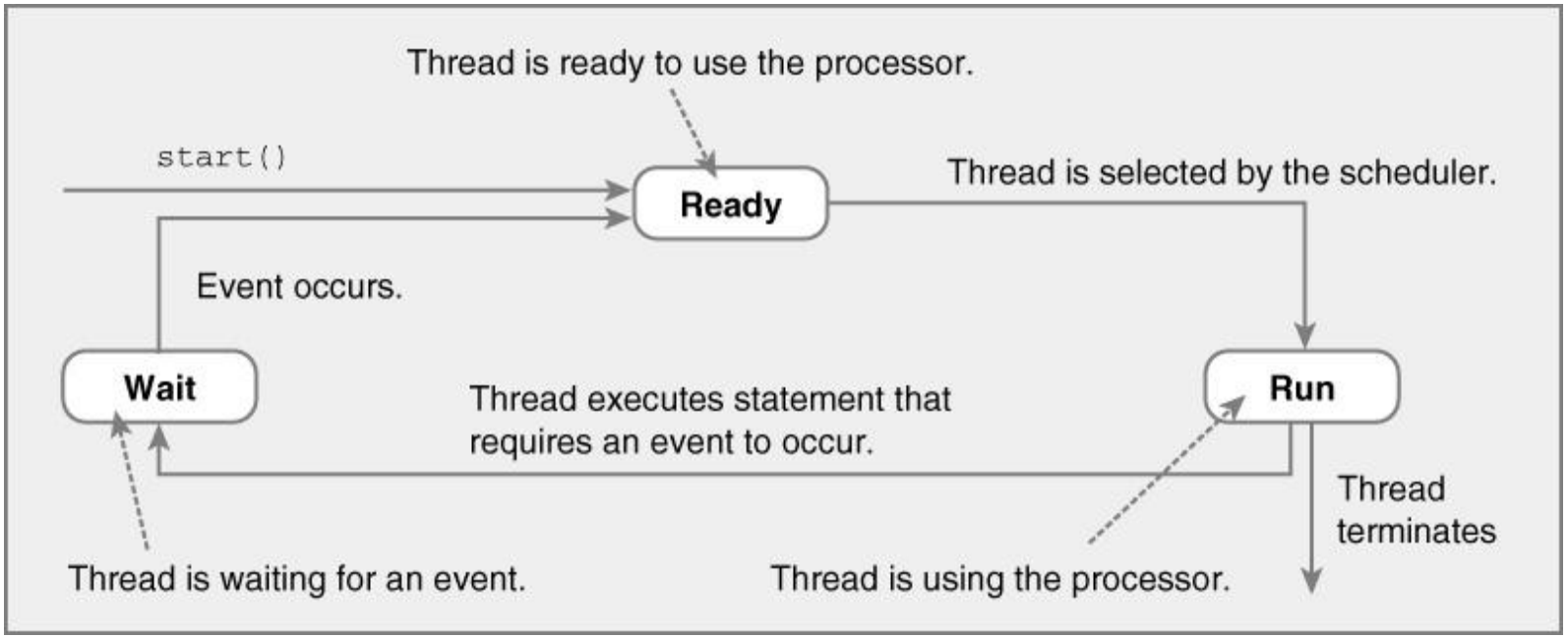
Threads - Scheduling

- Scheduler
  - Determines which runnable threads to run
    - When context switching takes place
  - Can be based on thread priority
  - Part of OS or Java Virtual Machine (JVM)

- Scheduling policy
  - **Non-preemptive** (cooperative) scheduling
  - **Preemptive** scheduling

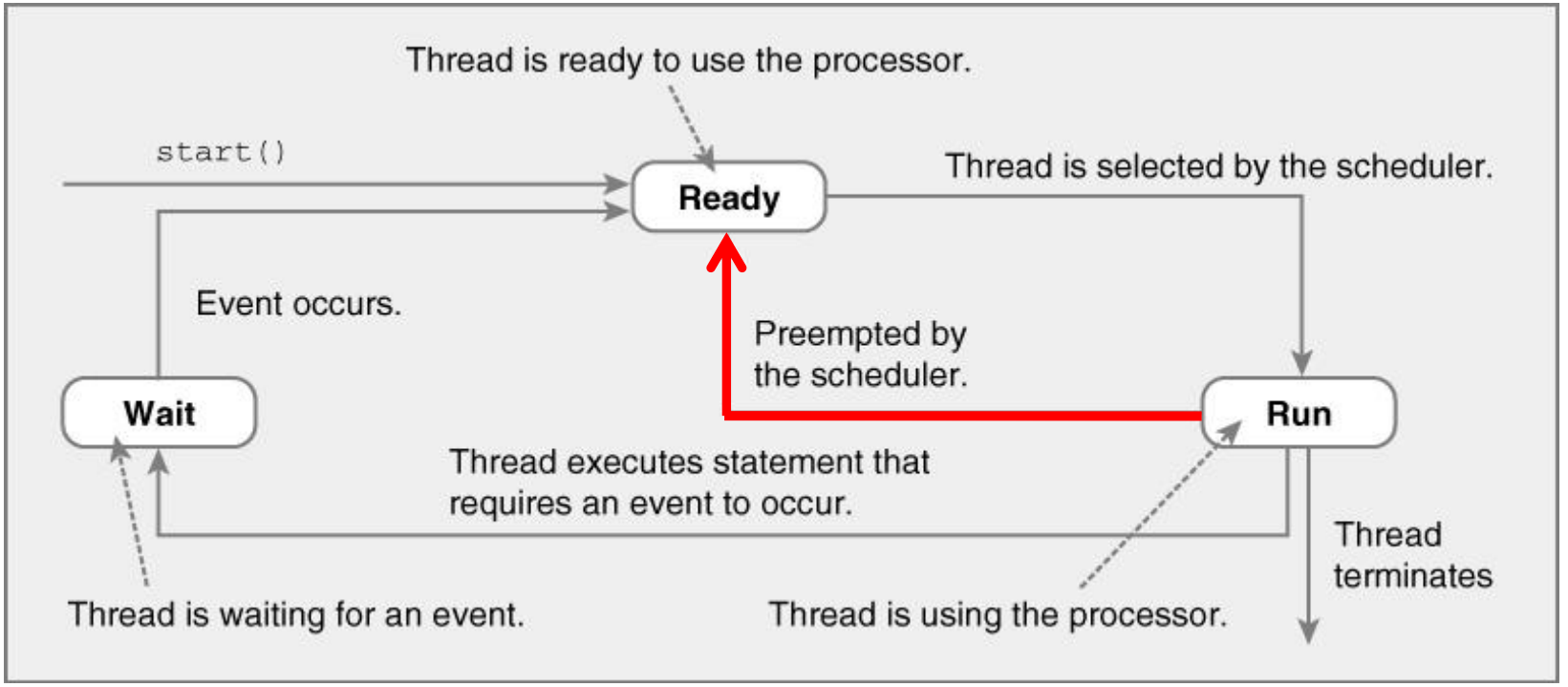
Threads - Non-preemptive Scheduling

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



Threads - Preemptive Scheduling

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



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
- Your code should work correctly regardless the scheduling policy in place

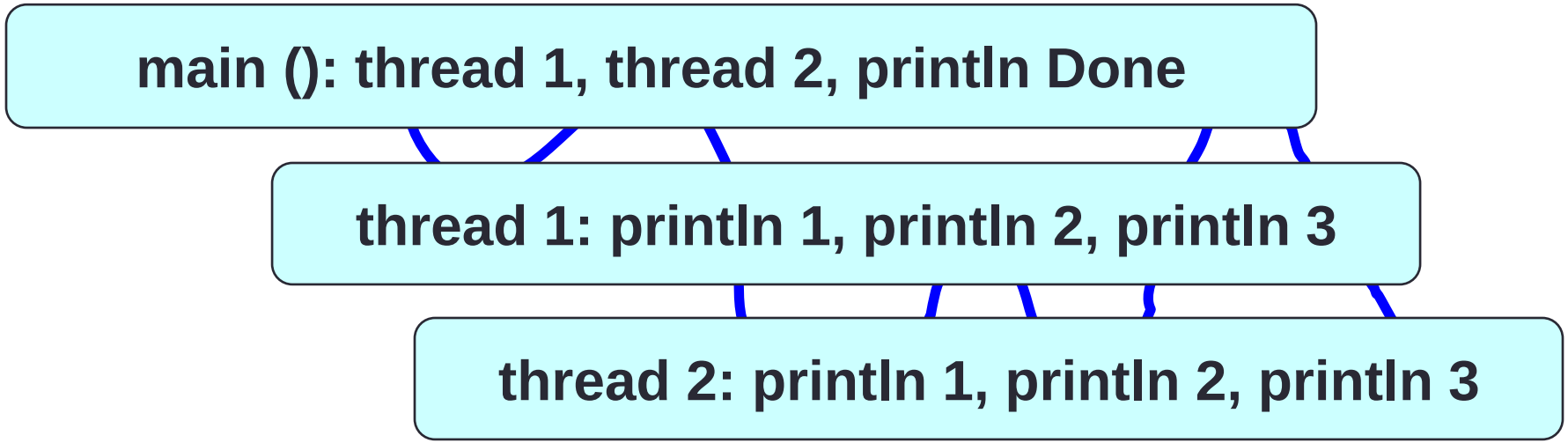
Java Thread Example

```
public class ThreadNoJoin extends Thread {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            try {
                Thread.sleep((int)(Math.random() * 3000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(i);
        }
    }

    public static void main(String[] args) {
        Thread thread1 = new ThreadNoJoin();
        Thread thread2 = new ThreadNoJoin();

        thread1.start();
        thread2.start();

        System.out.println("Done");
    }
}
```

- **Example:** ThreadNoJoin.java
- To understand this example better, let's assume we want to make a sandwich where one person (thread1) looks for bread, and another for tomato (thread2)

Java Thread Example - Output

- Possible outputs
  - 1,2,3,1,2,3,Done // thread 1, thread 2, main()
  - 1,2,3,Done,1,2,3 // thread 1, main(), thread 2
  - Done,1,2,3,1,2,3 // main(), thread 1, thread 2
  - 1,1,2,2,3,Done,3 // main() & threads interleaved



Thread Class - join( ) Method

- Can wait for thread to terminate with join( )
- Method prototype
  - **public final void join( )**
    - Blocks thread executing join(). For example, if t1 is a thread, t1.join() will block the thread executing t1.join() (current thread). Current thread execution can continue once t1 thread has finished
    - Returns when thread is done
    - Throws InterruptedException if interrupted
- **Example:** ThreadJoin.java

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.**
- The following is WRONG as you are executing code sequentially and not concurrently

```
t1.start()
t1.join()
t2.start()
t2.join()
```

- Feel free to use arrays, sets, etc., to keep track of your threads

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.

Terminating Threads

- A thread ends when the run() method ends
- Sometimes we may need to stop a thread before it ends
  - For example, you may have created several threads to find a problem solution and once one thread finds it, there is no need for the rest
- How to stop thread?
  - **Using stop() method** - **WRONG!** This is a deprecated method. Using it can lead to problems when data is shared
  - **Use interrupt() method**

Terminating Threads

- **Using interrupt() method**
  - This method does not stop the thread. Instead, it notifies the thread that it should terminate. The method sets a boolean variable in the thread and that value can be checked by the thread (by using the method interrupted())
  - It is up to the thread to terminate or not
  - public void run() {

```
        while(!Thread.interrupted()) {
            // work
         }
         // Release resources, cleaning tasks
    }
```

- **Example:** InterruptExample.java

Thread Example

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