# CMSC 330: Organization of Programming Languages

Multithreaded Programming in Java

## Multiprocessors

- Description
  - Multiple processing units (multiprocessor)
  - From single microprocessor to large compute clusters
  - Can perform multiple tasks in parallel simultaneously

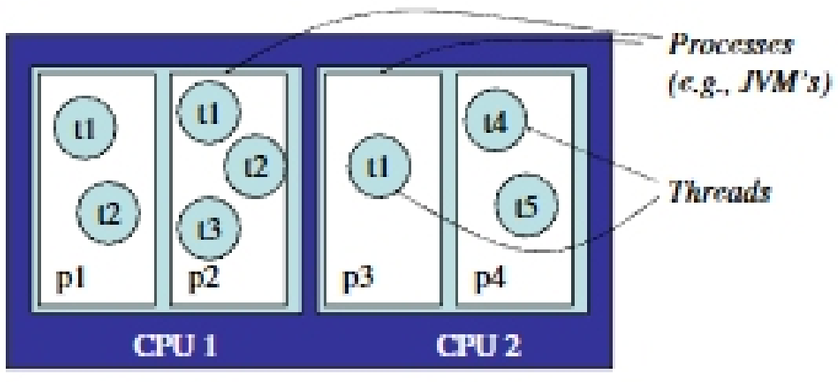
Dual-core AMD Athlon X2

32 processor Pentium Xeon

106K processor IBM BlueGene/L



## Computation Abstractions



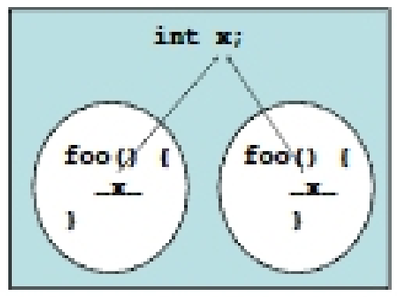

*A computer*



## Processes vs. Threads

```
int x;
foo() {
  x
}
```

```
int x;
foo() {
  x
}
```




*Processes do not share data*

*Threads share data within a process*

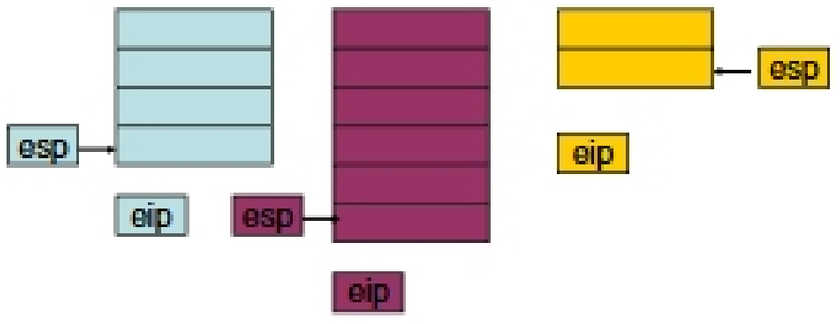


## So, What Is a Thread?

- Conceptually
  - Parallel computation occurring within a process
- Implementation view
  - Program counter and stack
  - Heap and static area shared among all threads
- All programs have at least one thread (main)



## Implementation View




- Per-thread stack and instruction pointer
  - Saved in memory when thread suspended
  - Put in hardware esp/eip when thread resumes

## Tradeoffs

- Threads can increase performance
  - Parallelism on multiprocessors
  - Concurrency of computation and I/O
- Natural fit for some programming patterns
  - Event processing
  - Simulations
- But increased complexity
  - Need to worry about safety, liveness, composition
- And higher resource usage



## Programming Threads

- Threads are available in many languages
  - C, C++, OCaml, Java, Ruby, …
- In many languages (e.g., C and C++), threads are a platform specific add-on
  - Not part of the language specification
  - Implemented as code libraries (e.g., pthreads)
- They're part of the Java language specification

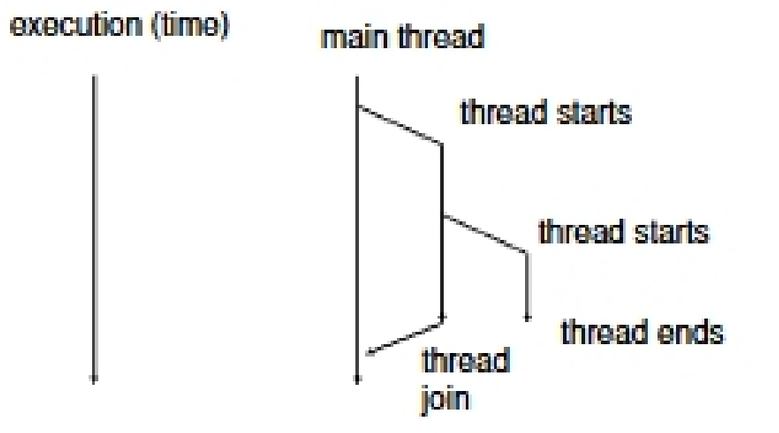


## Java Threads

- Every application has at least one thread
  - The "main" thread, started by the JVM to run the application's main() method
- main() can create other threads
  - Explicitly, using the Thread class
  - Implicitly, by calling libraries that create threads as a consequence
    - RMI, AWT/Swing, Applets, etc.



## Thread Creation






## Thread Creation in Java

- To explicitly create a thread:
  - Instantiate a Thread object
    - An object of class Thread or a subclass of Thread
  - Invoke the object's start() method
    - This will start executing the Thread's run() method concurrently with the current thread
  - Thread terminates when its run() method returns



## Running Example: Alarms

- Goal: let's set alarms which will be triggered in the future
  - Input: time t (seconds) and message m
  - Result: we'll see m printed after t seconds

## Example: Synchronous alarms

```
while (true) {
  System.out.print("Alarm> ");

  // read user input
  String line = b.readLine();
  parseInput(line); // sets timeout

  // wait (in secs)
  try {
    Thread.sleep(timeout * 1000);
  } catch (InterruptedException e) { }
  System.out.println("("+timeout+") "+msg);
}
```



## Making It Threaded (1)

```
public class AlarmThread extends Thread {
  private String msg = null;
  private int timeout = 0;

  public AlarmThread(String msg, int time) {
    this.msg = msg;
    this.timeout = time;
  }

  public void run() {
    try {
      Thread.sleep(timeout * 1000);
    } catch (InterruptedException e) { }
    System.out.println("("+timeout+") "+msg);
  }
}
```



## Making It Threaded (2)

```
while (true) {
  System.out.print("Alarm> ");

  // read user input
  String line = b.readLine();
  parseInput(line);
  if (m != null) {
    // start alarm thread
    Thread t = new AlarmThread(m,tm);
    t.start();
  }
}
```



## Alternative: The Runnable Interface

- Extending Thread prohibits a different parent
- Instead implement Runnable
  - Declares that the class has a void run() method
- Construct a Thread from the Runnable
  - Constructor Thread(Runnable target)
  - Constructor Thread(Runnable target, String name)



## Thread Example Revisited

```
public class AlarmRunnable implements Runnable {
  private String msg = null;
  private int timeout = 0;

  public AlarmRunnable(String msg, int time) {
    this.msg = msg;
    this.timeout = time;
  }

  public void run() {
    try {
      Thread.sleep(timeout * 1000);
    } catch (InterruptedException e) { }
    System.out.println("("+timeout+") "+msg);
  }
}
```



## Thread Example Revisited (2)

```
while (true) {
  System.out.print("Alarm> ");

  // read user input
  String line = b.readLine();
  parseInput(line);
  if (m != null) {
    // start alarm thread
    Thread t = new Thread(
                  new AlarmRunnable(m,tm));
    t.start();
  }
}
```