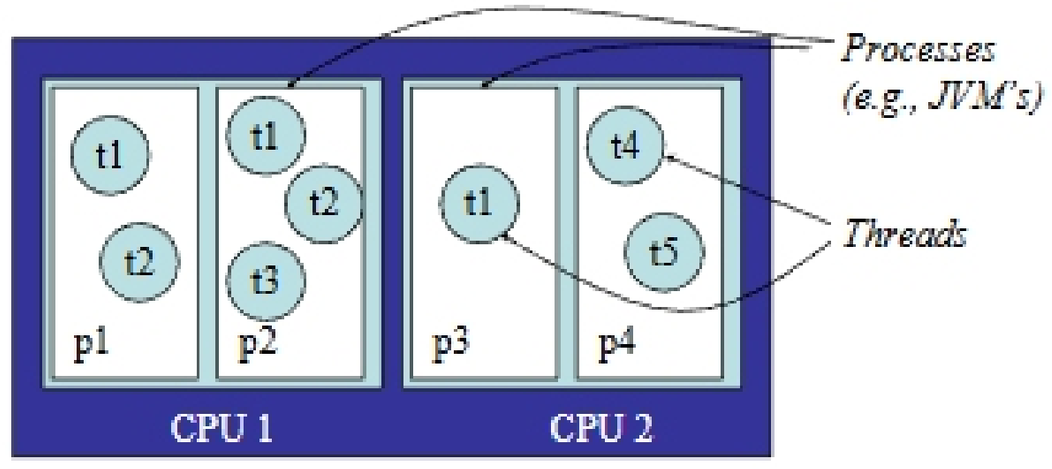
CMSC 330: Organization of Programming Languages

Threads

Computation Abstractions

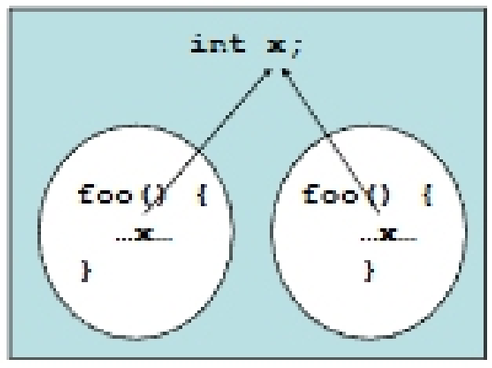


A computer

Processes vs. Threads

```
int x;
foo() {
…x…
}
```

```
int x;
foo() {
…x…
}
```



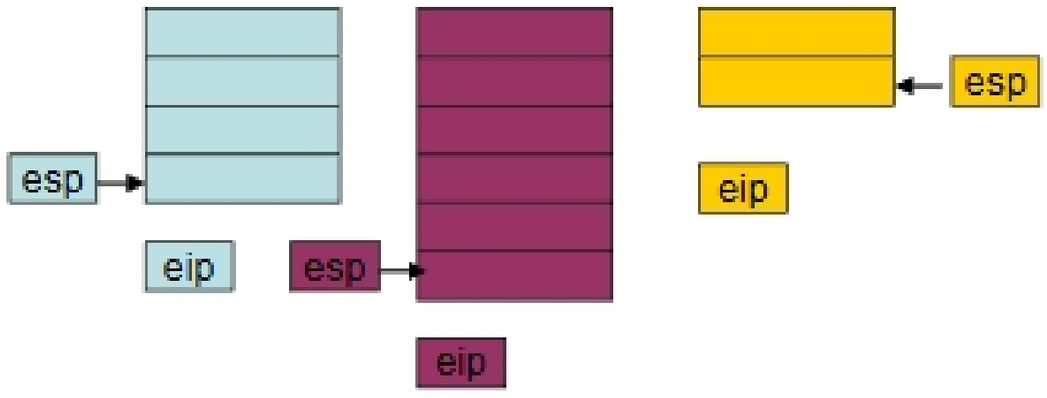
Processes do not share data

Threads share data within a process

So, What Is a Thread?

- **Conceptually**: it is a parallel computation occurring within a process
- **Implementation view**: it's a program counter and a stack. The heap and static area are 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

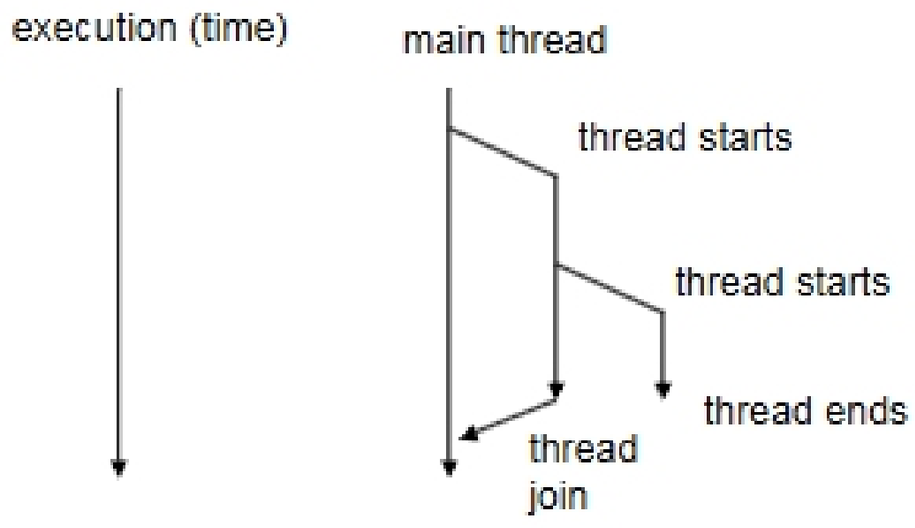
Programming Threads

- Threads are available in many languages
  - C, C++, Objective Caml, Java, SmallTalk ...
- In many languages (e.g., C and C++), threads are a platform specific add-on
  - Not part of the language specification
- 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);
}
```