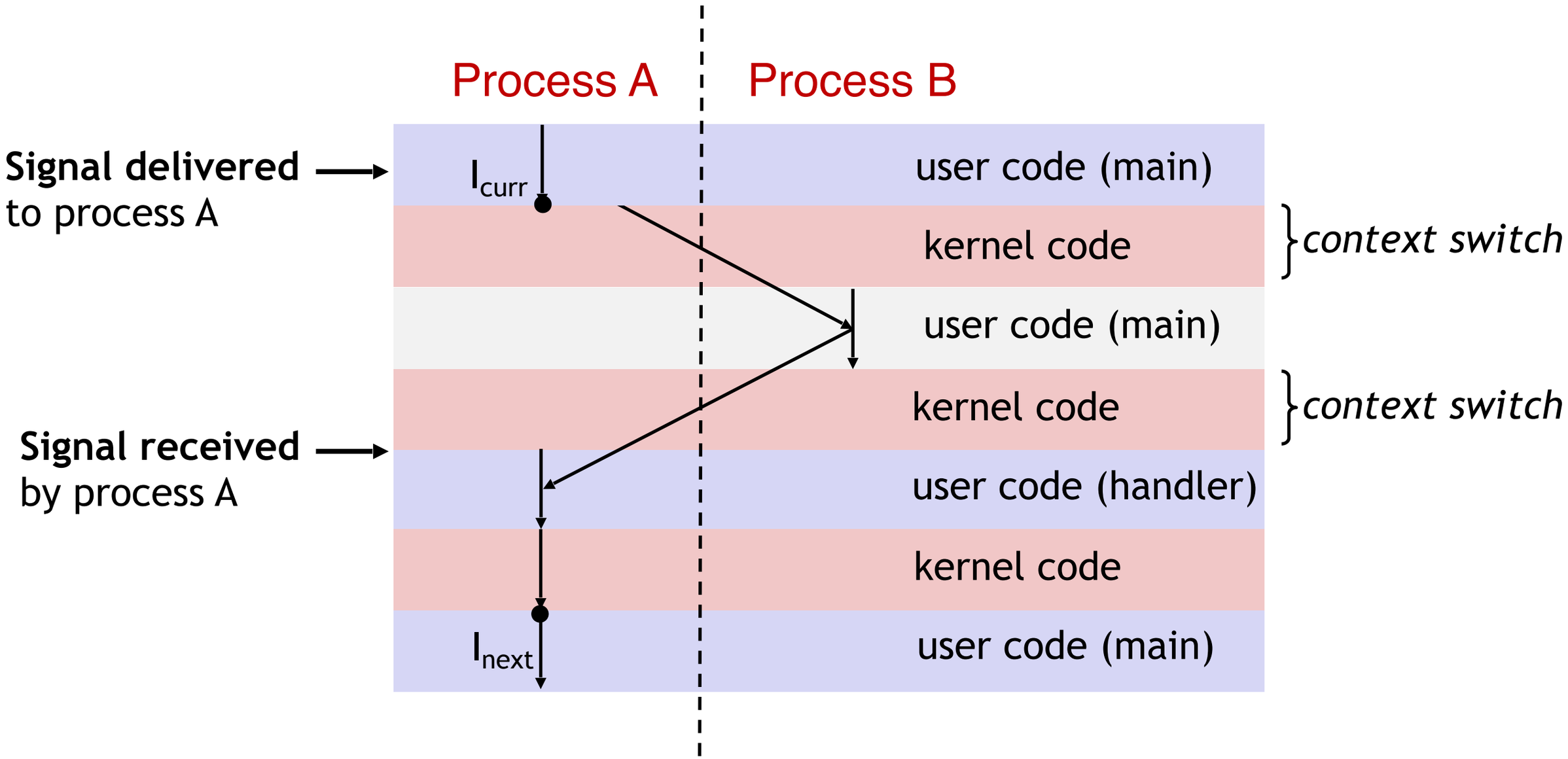
Threads

CS 475, Spring 2018
Concurrent & Distributed Systems

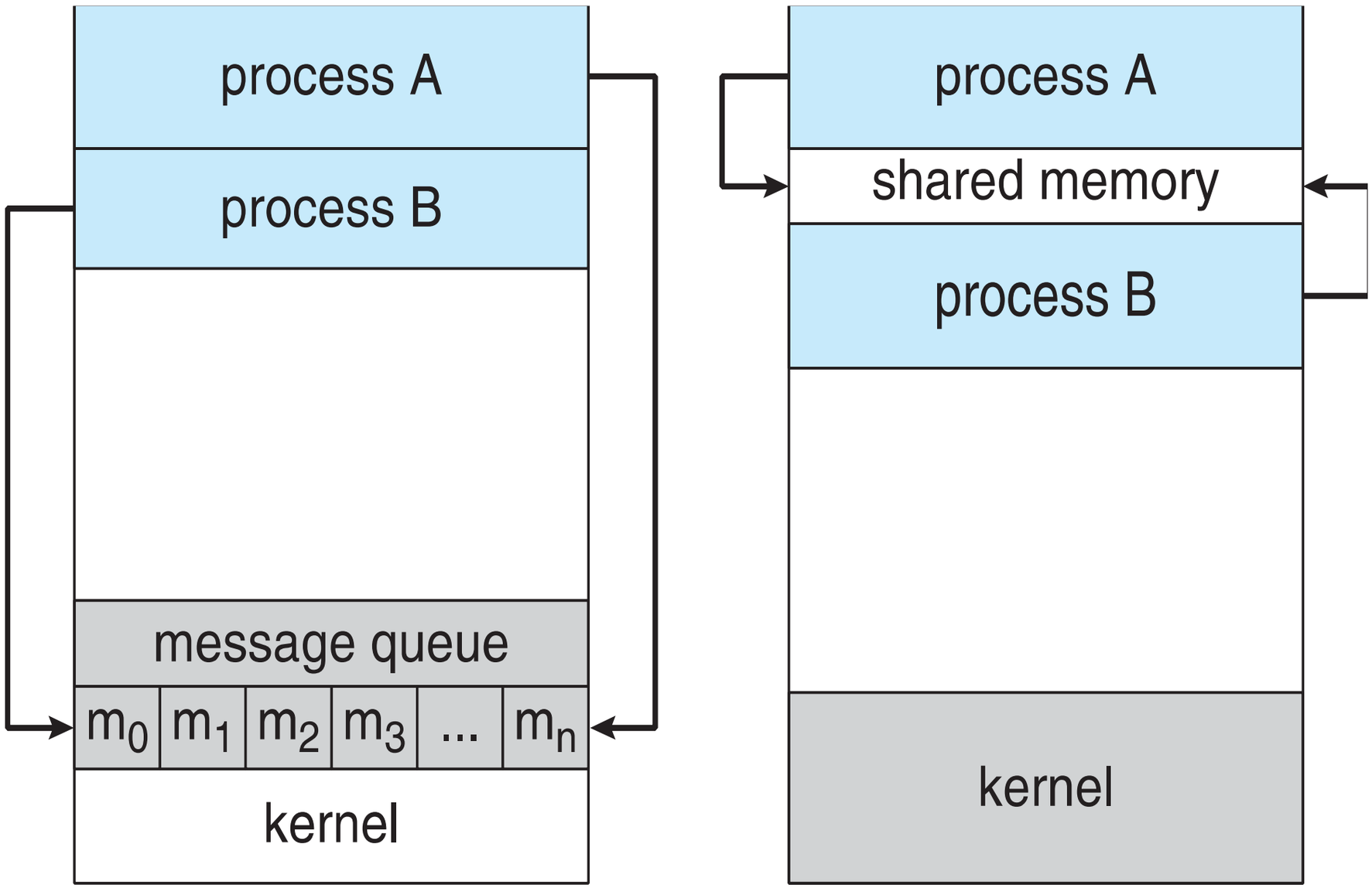
Review: Signals

- OS has a mechanism to **interrupt** regular execution of a process
- We've discussed this in the context of **preemption** - when a round-robin scheduler decides a processes' quantum is up
- Signals notify a process of some message (the signal) and then allow the process to do something

Review: Signal Handlers as Concurrency



Review: Message Passing & Shared Memory



Message Passing **Shared Memory**

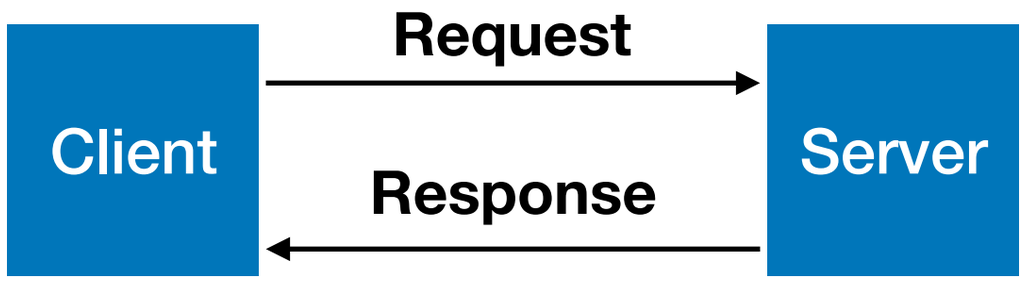
Review: IPC Summary

- Different models suit different needs
- Examples (real world):
  - Communication via postcard
  - Speaking in a room with two people
  - Instant messaging
  - Speaking in this classroom (with 60 people)

Announcements

- Additional readings:
  - Tannenbaum 3.1
- Note: TA office hours now posted
  - GTA: Arda Gumusalan
    Office Hours: Tuesdays, 10:00am-12:00pm, 4456 Engineering Building
  - UTA: Thanh Luu
    Office Hours:Â Tuesdays, 2:00pm-4:00pm, Thursdays, 10:00am-12:00pm, 4456 Engineering Building
- Note - schedule revision (7 lectures on threads -> 3)

Writing a Server

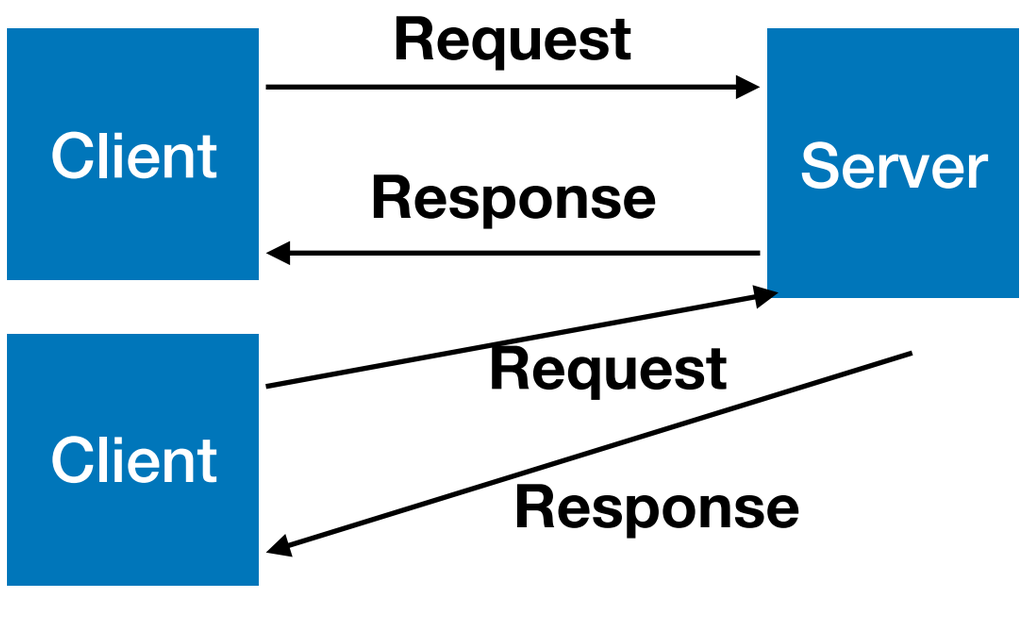



```
message next_produced;
while (true) {
    send(next_produced_command);
    receive(server_response);
}
```

```
message next_consumed;
while (true) {
    receive(next_consumed);
    send(response);
}
```

Assume talking over some messaging, like our example on Monday

Serving Multiple Clients

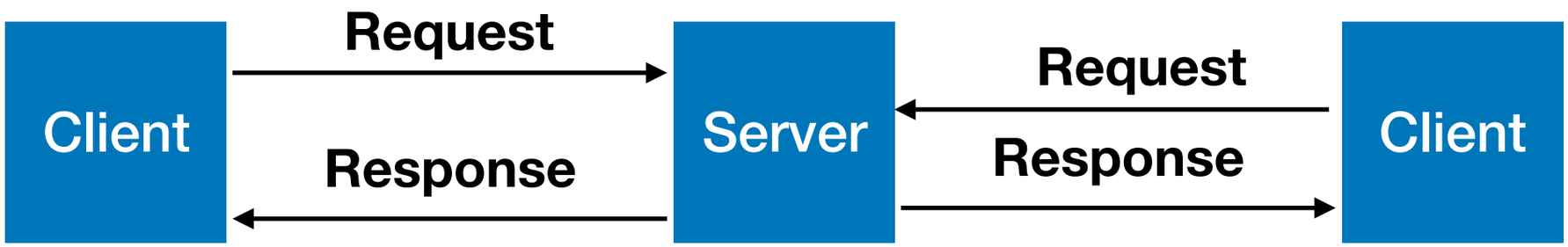



```
message next_produced;
while (true) {
    send(next_produced_command);
    receive(server_response);
}
```

```
message next_consumed;
while (true) {
    receive(next_consumed);
    send(response);
}
```

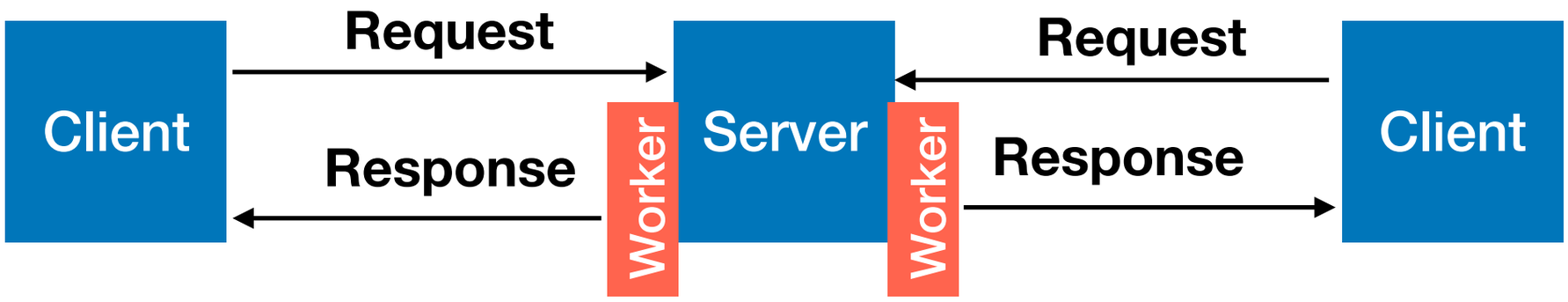
Assume talking over some messaging, like our example on Monday

Serving Multiple Clients



How can our server respond to multiple clients at once?

Serving Multiple Clients



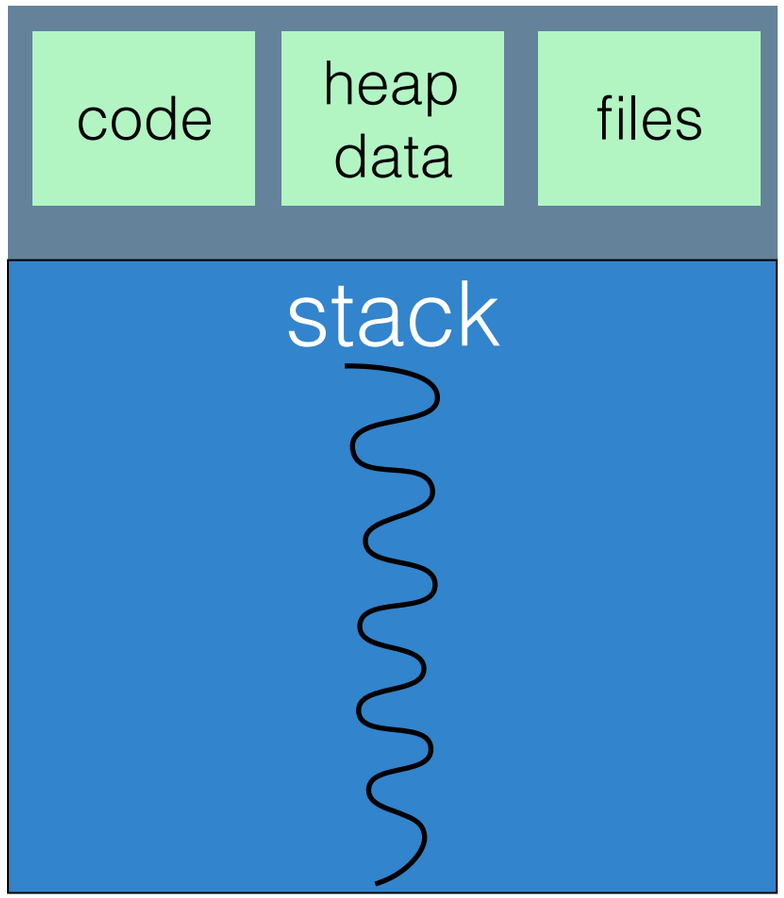
Server receives request, fork()'s a worker, lets worker handle request
Server can immediately receive another request (as soon as fork())

Workers might communicate with parent process with shared memory

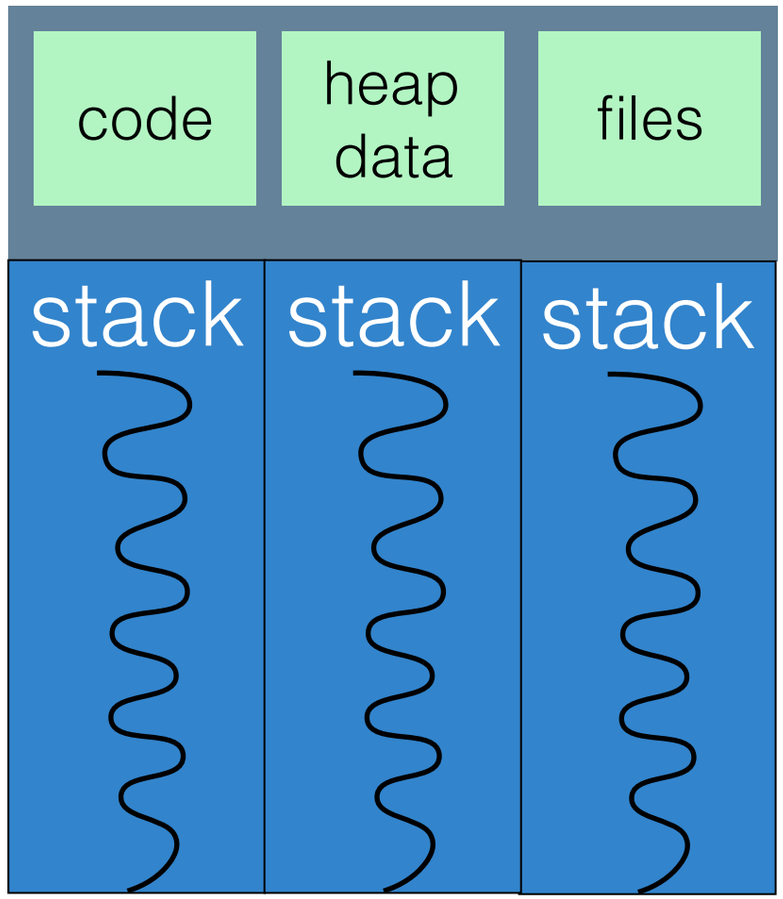
Serving Multiple Clients

- This architecture used to be **incredibly common**
- Apache HTTPD (once run almost every website, now ~40%) introduced in 1995
- HTTPD serviced each request in a process (keeping a pool of those workers and reusing them)
- Upsides:
  - Easy to program
  - If nothing needs to be shared between workers, fast
- Downsides:
  - Consumes relatively a lot of memory: web servers became RAM dependent instead of CPU dependent

Processes vs Threads

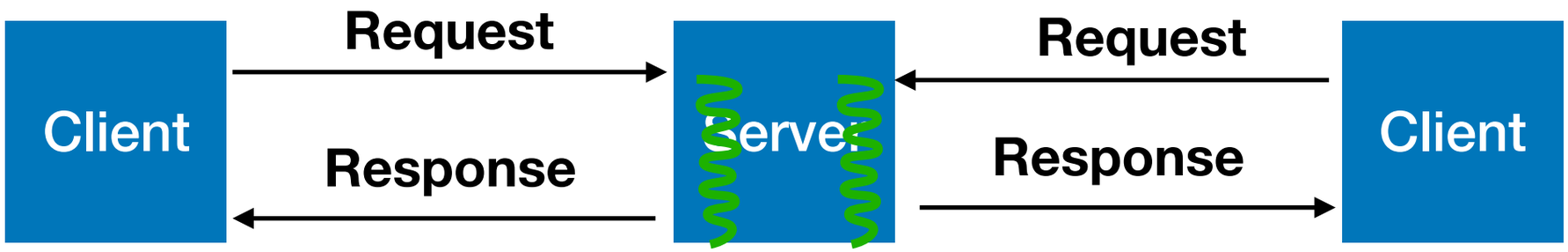


Single-Threaded Process



Multi-Threaded Process

Serving Multiple Clients



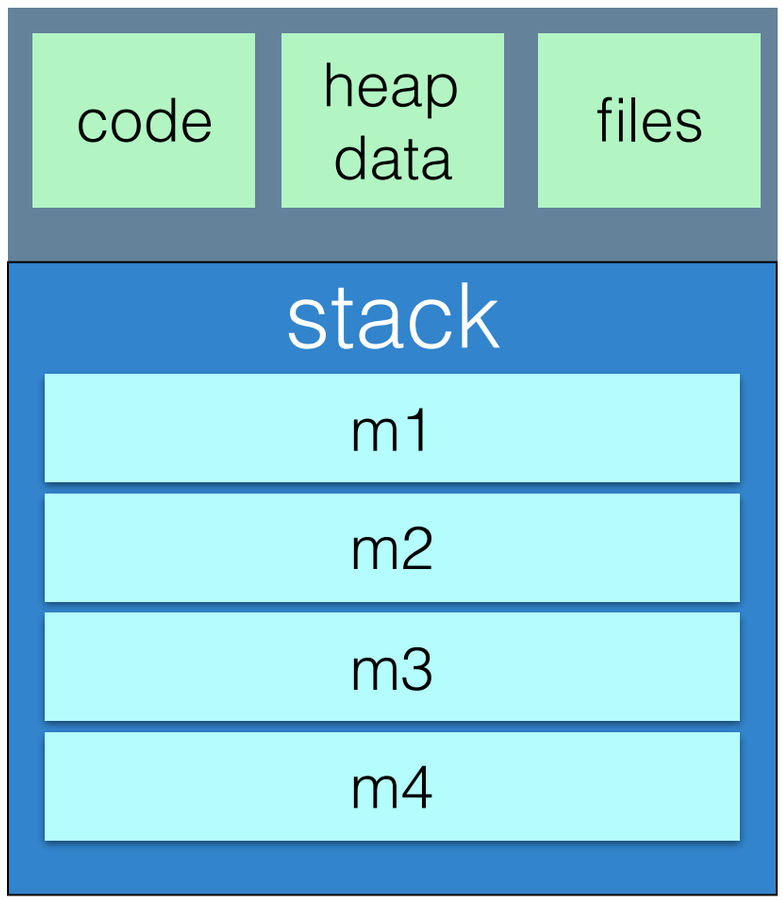
“But what about the isolation/fault tolerance? Now one client can crash the whole server!”

In the real world, you might still have *n* worker processes, but then each worker process has many threads.

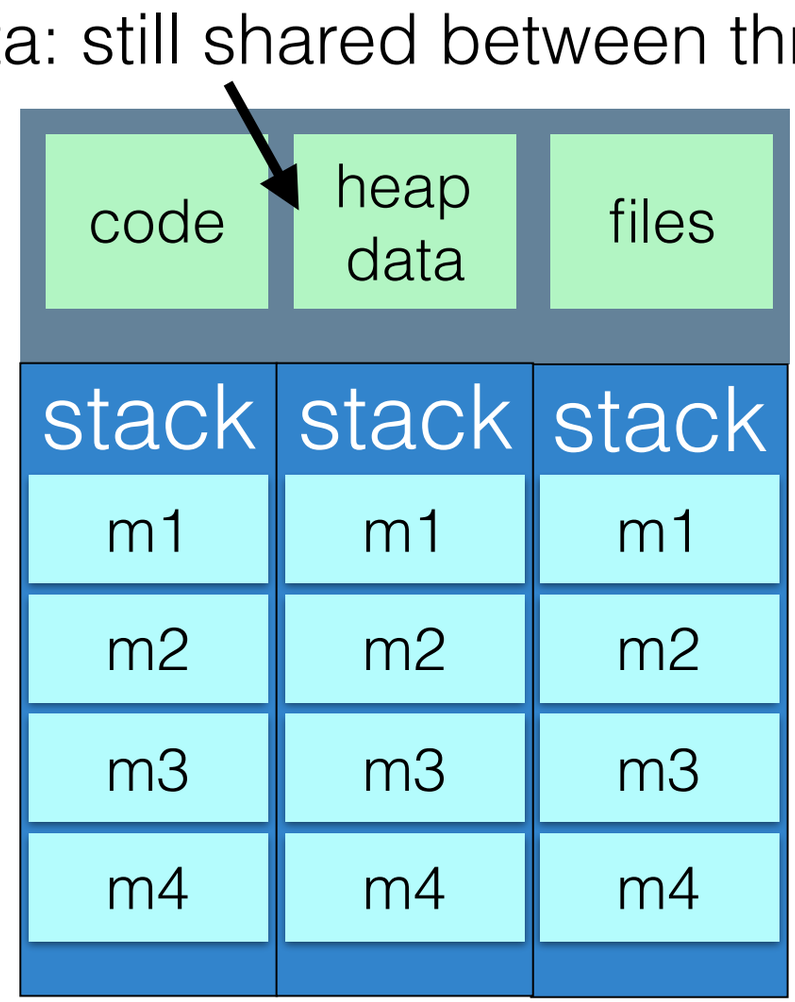
What do we use threads for?

- Run multiple tasks seemingly at once
  - Update UI
  - Fetch data
  - Respond to network requests
- Process creation: heavyweight, thread creation: lightweight
- Improve responsiveness, scalability
- Concurrency + Parallelism

Threads: Memory View



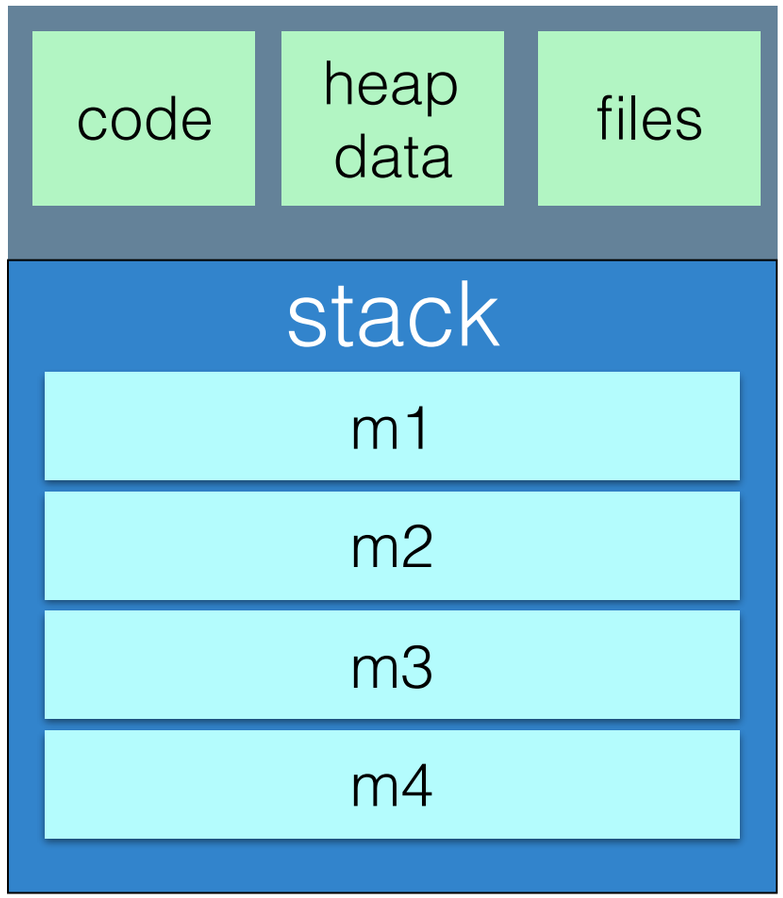
Single-Threaded Process



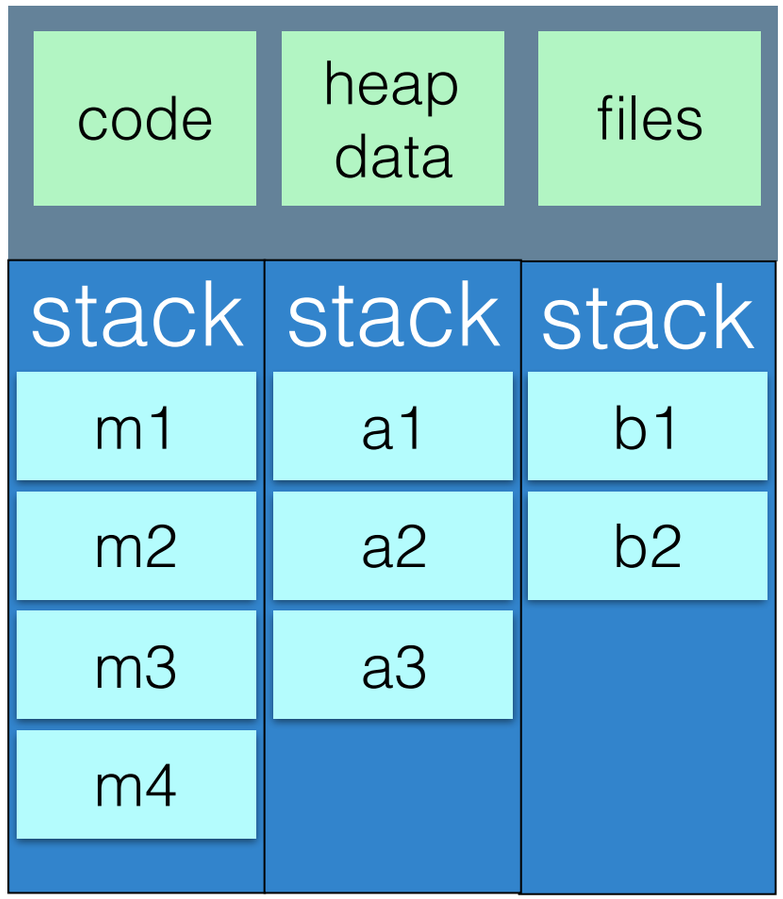
Multi-Threaded Process

Each thread might be executing the same code, but with different local variables (and hence doing different stuff)

Threads: Memory View



Single-Threaded Process



Multi-Threaded Process

Each thread might be executing totally different code, too

Processes vs Threads

- How threads and processes are similar
  - Each has its own logical control flow.
  - Each can run concurrently.
  - Each is context switched.
- How threads and processes are different
  - Threads share code and data, processes (typically) do not.
  - Threads are somewhat less expensive than processes.
  - Process control (creating and reaping) is (ballpark!) twice as expensive as thread control.

How to split up the work

- **Data parallelism** - distribute subsets of same data across multiple cores, perform same operation on each
- **Task parallelism** - distribute multiple threads, each thread performs a different operation
- In either case, there is some need to coordinate and share data between threads

Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- S is serial portion
- N processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

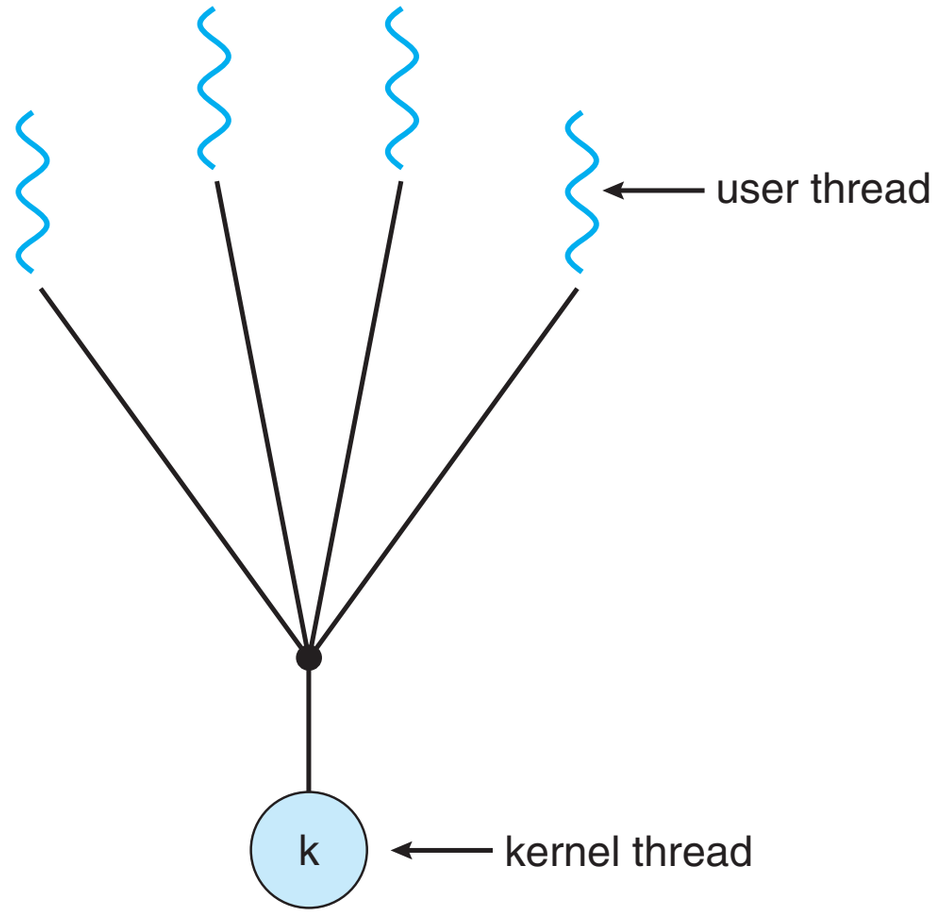
- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As N approaches infinity, speedup approaches 1 / S
- Serial portion of an application has disproportionate effect on performance gained by adding additional cores

User Threads vs Kernel Threads

- **User threads** - management done by user-level threads library
  - POSIX Pthreads
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Many-to-one Mappings

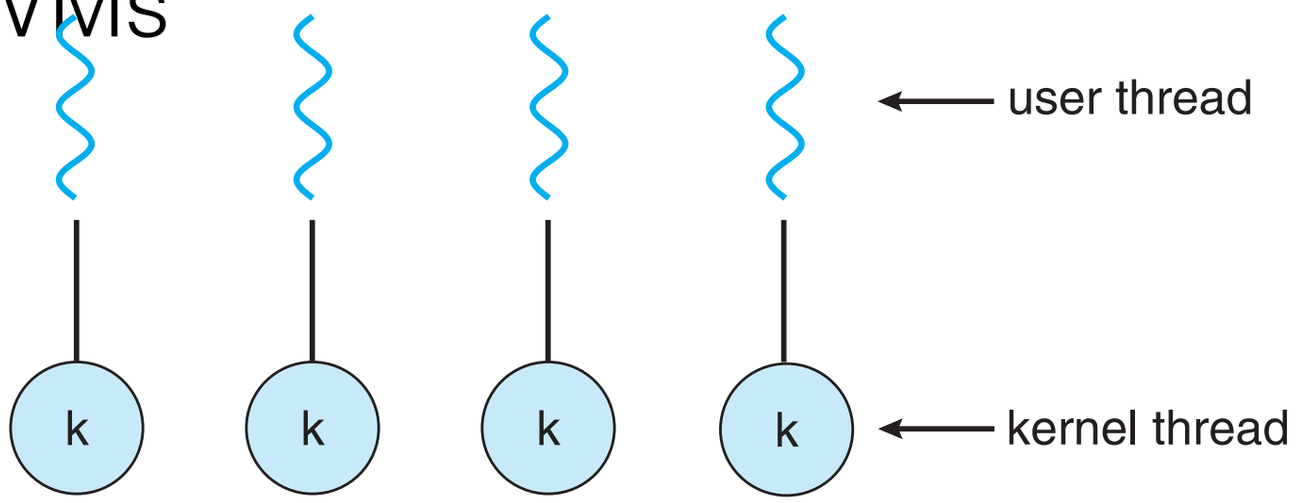
- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads
  - Terrible old JVMs



One-to-one Mappings

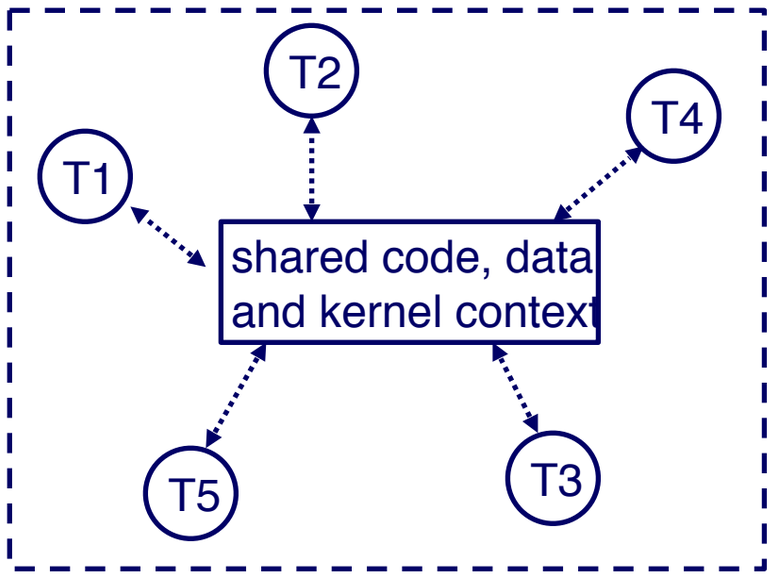
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead

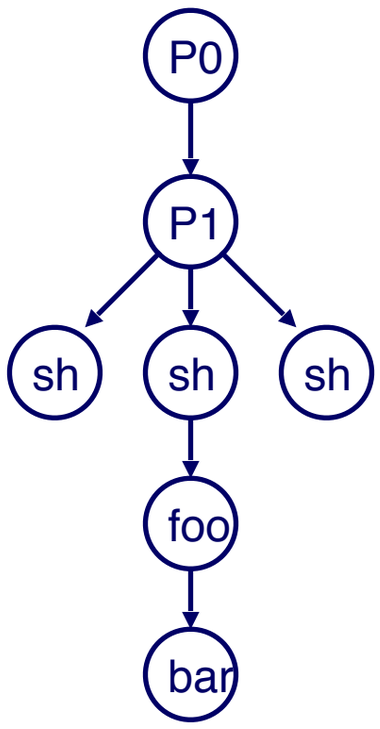
- Examples
  - Windows, Linux, Mac
  - Oracle + OpenJDK JVMs



Threads + Processes

- No “hierarchy” of threads associated with a process - more of a pool





Thread Libraries

- Thread library provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

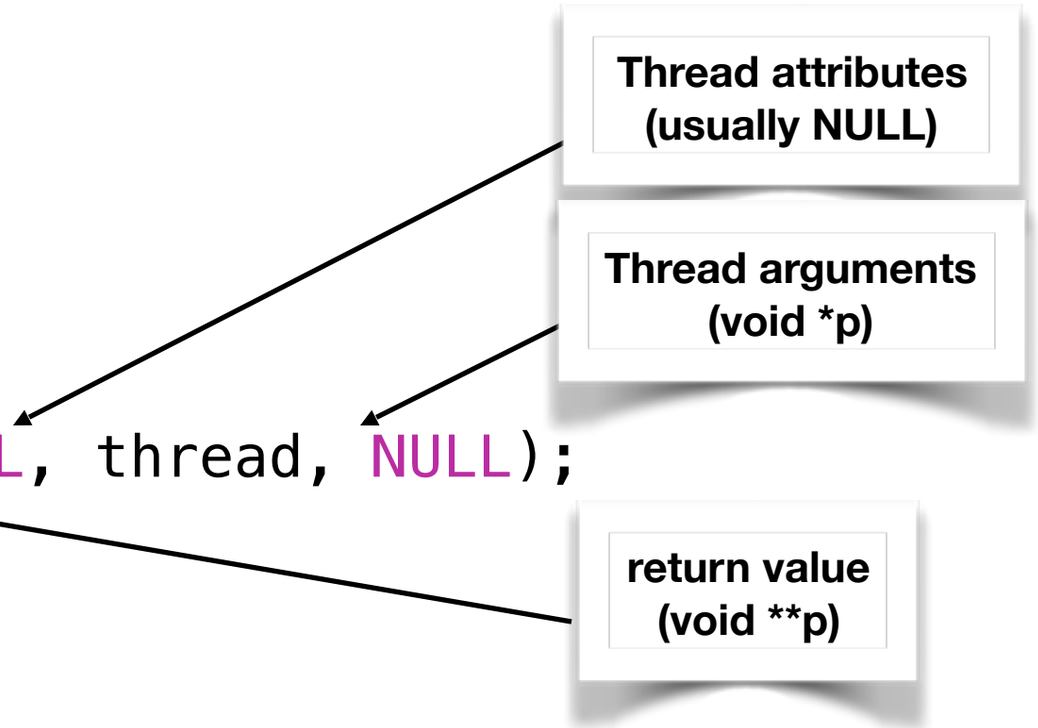
- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- Specification, not implementation
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Pthreads

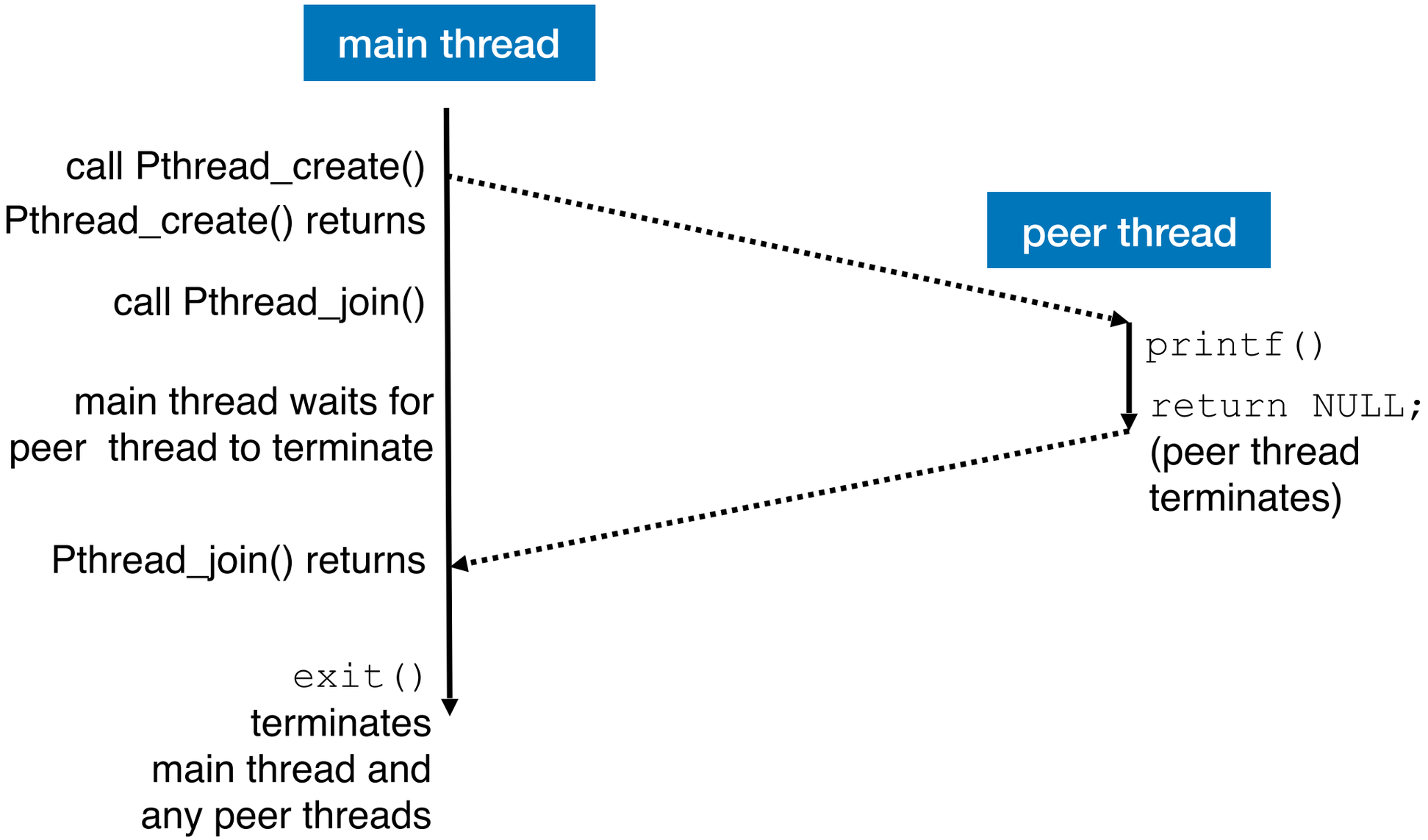
- Pthreads: Standard interface for ~60 functions that manipulate threads from C programs.
- Creating and reaping threads.
  - pthread_create
  - pthread_join
- Determining your thread ID
  - pthread_self
- Terminating threads
  - pthread_cancel
  - pthread_exit
  - exit [terminates all threads] , ret [terminates current thread]
- Synchronizing access to shared variables
  - pthread_mutex_init
  - pthread_mutex_[un]lock
  - pthread_cond_init
  - pthread_cond_[timed]wait

Pthreads Example

```c
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

void *thread(void *vargp);

int main() {
    pthread_t tid;

    Pthread_create(&tid, NULL, thread, NULL);
    Pthread_join(tid, NULL);
    exit(0);
}

/* thread routine */
void *thread(void *vargp) {
    printf("Hello, world!\n");
    return NULL;
}
```



Pthreads Example



Pthreads

- Threads have state just like processes (and are scheduled by the OS too)
- Ready
- Running
- Blocked
- Terminated

Threads in Java

- In Java, make a new thread by instantiating the class java.lang.Thread
- Pass it an object that implements *Runnable*
- *When you call thread.start(), the run() method of your runnable is called, from a new thread*
- *join() waits for a thread to finish*

```
Thread t = new Thread(new Runnable() {
 @Override
   public void run() {
    //This code will now run in a new thread
   }
});
t.start();
```

Threads in Java

- JVM manages threads (maybe uses Pthreads underneath)
- Each Java app gets at least one thread: `main`
  - Plus, likely a `finalizer` thread
  - Plus, the JVM itself makes a ton of threads that you can't see
    - JIT compiler, garbage collector mainly
- Fun tip: look at what threads are running in a Java app using the command-line `jstack` program

Threads in Java

```
public static void main(String[] args) throws InterruptedException {
  Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
        //This code will now run in a new thread
        System.out.println("Hello from the thread!");
      }
    });
    t.start();
    System.out.println("Hello from main!”);
    t.join();
}
```

What is the output of this code?

#1 Hello from the thread!
Hello from main!

This is a race condition

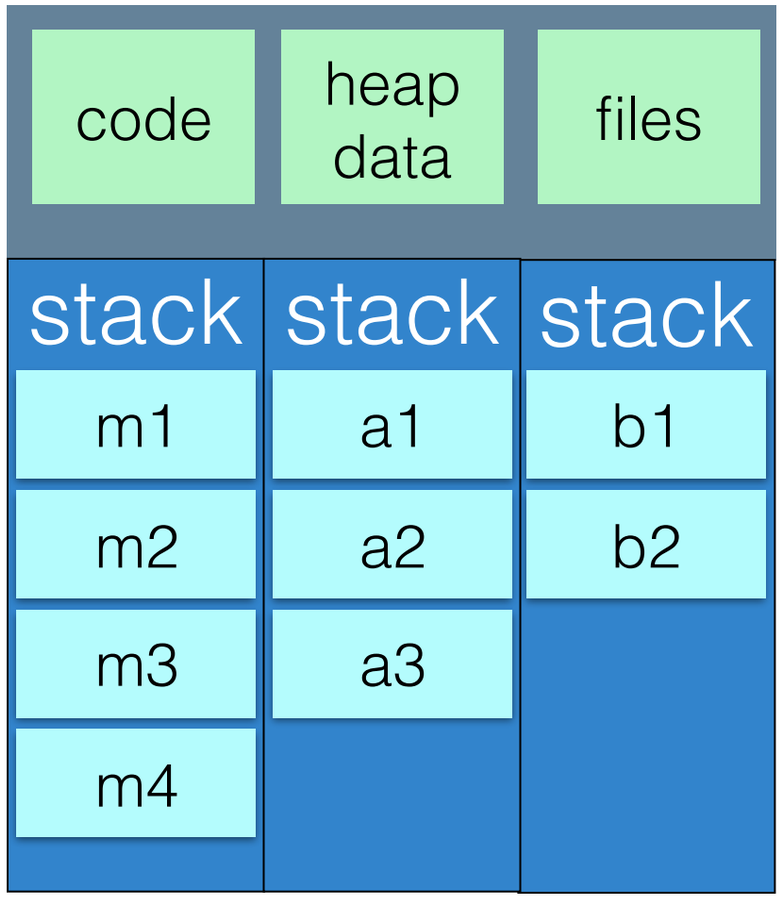
#2 Hello from main!
Hello from the thread!

Thread Communication

- Threads execute separate logical segments of code
- How do they talk to each other?

```
public static void main(String[] args) throws InterruptedException {
  Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
        //This code will now run in a new thread
        System.out.println("Hello from the thread!");
      }
    });
    t.start();
    System.out.println("Hello from main!”);
    t.join();
}
```

Shared Variables in Threads



Multi-Threaded Process

Thread Communication

- Same two high level options as processes: shared memory or message passing
- Shared memory:
  - Things are shared by default!
- Message passing:
  - Programmer manually says what to share
- We will focus on the simple shared memory approach, but keep in mind other options too

Race Conditions

```
static int i = 0;
public static void increment()
{
    i = i + 1;
}
```

This is one possible interleaving

Thread 1	Thread 2
increment()	increment()
read i = 0	
write i = 1	read i = 0
	write i =1

Race Conditions

	Person A	Person B
12:30	Look in fridge. Out of milk.	
12:35	Leave for store.	
12:40	Arrive at store.	Look in fridge. Out of milk.
12:45	Buy milk.	Leave for store.
12:50	Arrive home, put milk away.	Arrive at store.
12:55		Buy milk.
1:00		Arrive home, put milk away. Oh no!

Critical Section Problem

- Each thread/process has some *critical section* of code that:
  - Changes shared variables, files etc
  - When one thread/process is in a critical section, no other may be in the same critical section
  - Critical section problem: design a protocol to solve this
  - Each process/thread asks for permission to enter critical section, does its work, then exits the section

Critical Section in increment()

```
static int i = 0;
public static void increment()
{
    enterSection();
    i = i + 1;
    exitSection();
}
```

Only one thread can read/write i at once

But how to implement enterSection() and exitSection()?

Solution to Critical-Section Problem

- Need to guarantee **mutual exclusion**
  - If one thread/process is executing its critical section, no other can execute in their critical sections
- Need to guarantee **progress:**
  - If no process is executing in its critical section, and some other would like to, then some process must be allowed to continue
- Need to guarantee **bounded waiting:**
  - If some process wants to enter its critical section, it must eventually be granted access
- **Next time!**

Thread-safe code

- Libraries that are thread-safe promise that their functionality will be correct even if they are called from multiple threads
- Data structures are a common sore point
- Java provides a suite of thread-safe data structures in java.util.concurrent

Thread-safe code

```
HashSet hs = new HashSet();
void thread1()
{
    hs.add("A");
}
void thread2()
{
    hs.add(“B");
}
```

After thread1() is called in one thread, thread2() is called in another, HashSet may have one, both, or neither of A,B

```
HashSet hs =
    ConcurrentHashMap.newKeySet();
void thread1()
{
    hs.add("A");
}
void thread2()
{
    hs.add("B");
}
```

After thread1() is called in one thread, thread2() is called in another, HashSet will have both A,B

Checkpoint

Go to socrative.com and select “Student Login” (works well on laptop, tablet or phone)

Room Name: CS475

ID is your @gmu.edu email

Remember: this is a checkpoint for you, it is only graded for attendance

Assignment 1 Discussion

- http://www.jonbell.net/gmu-cs-475-spring-2018/homework-1/
- https://autolab.cs.gmu.edu/courses/CSTEST/assessments

Roadmap

- Next week: Synchronization
  - Sharing memory safely
  - Bad professor joke:

    Knock, knock.
    Race condition.
    Who's there?