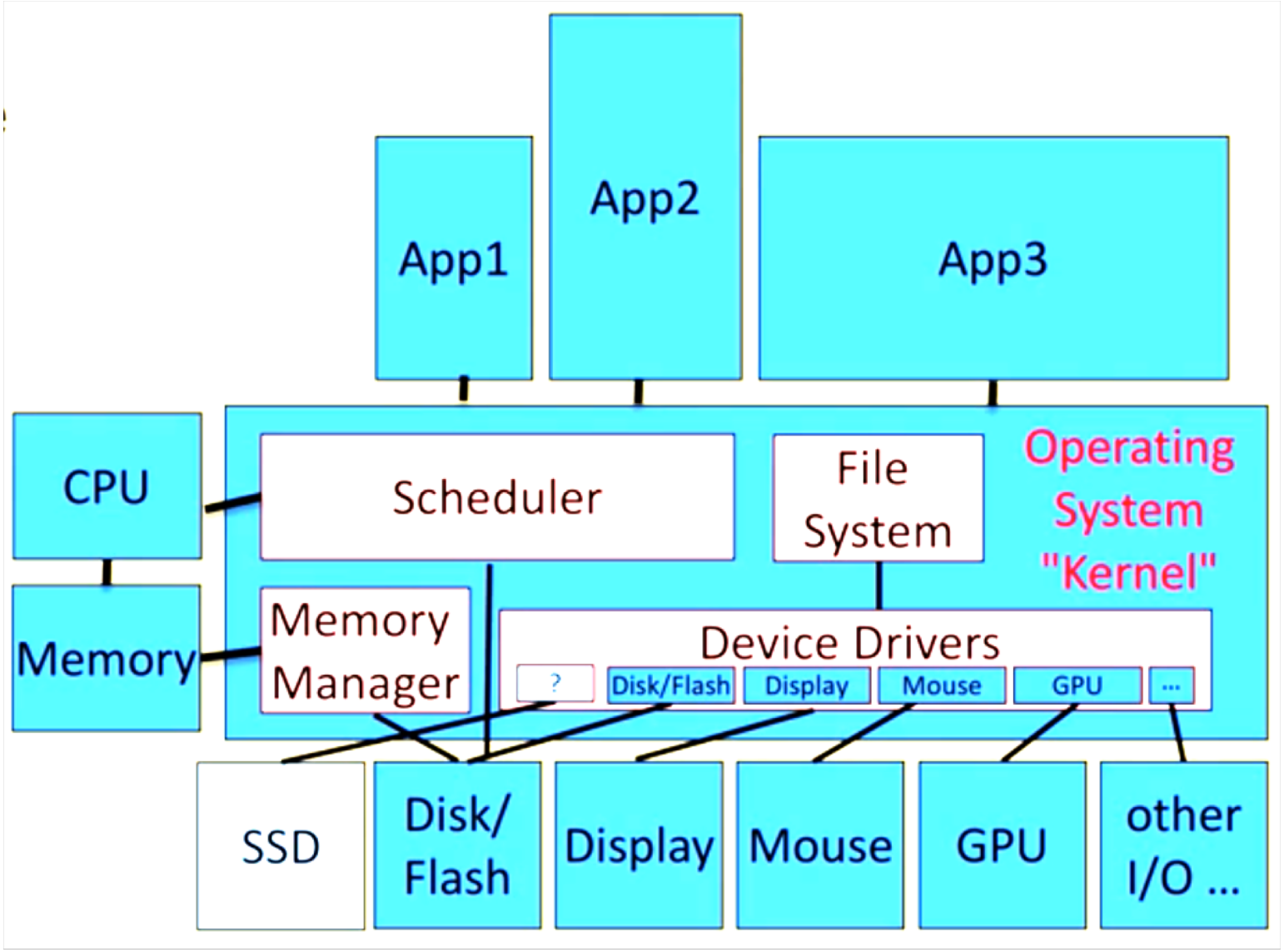
Midterm Review

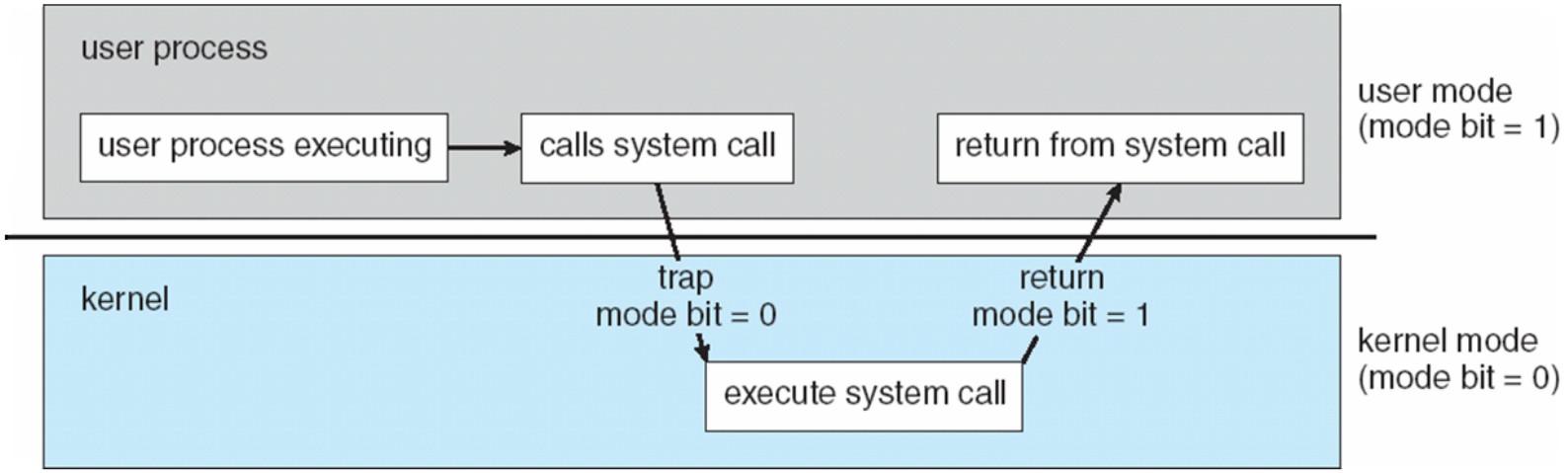
Overview OS

Operating Systems (OS)



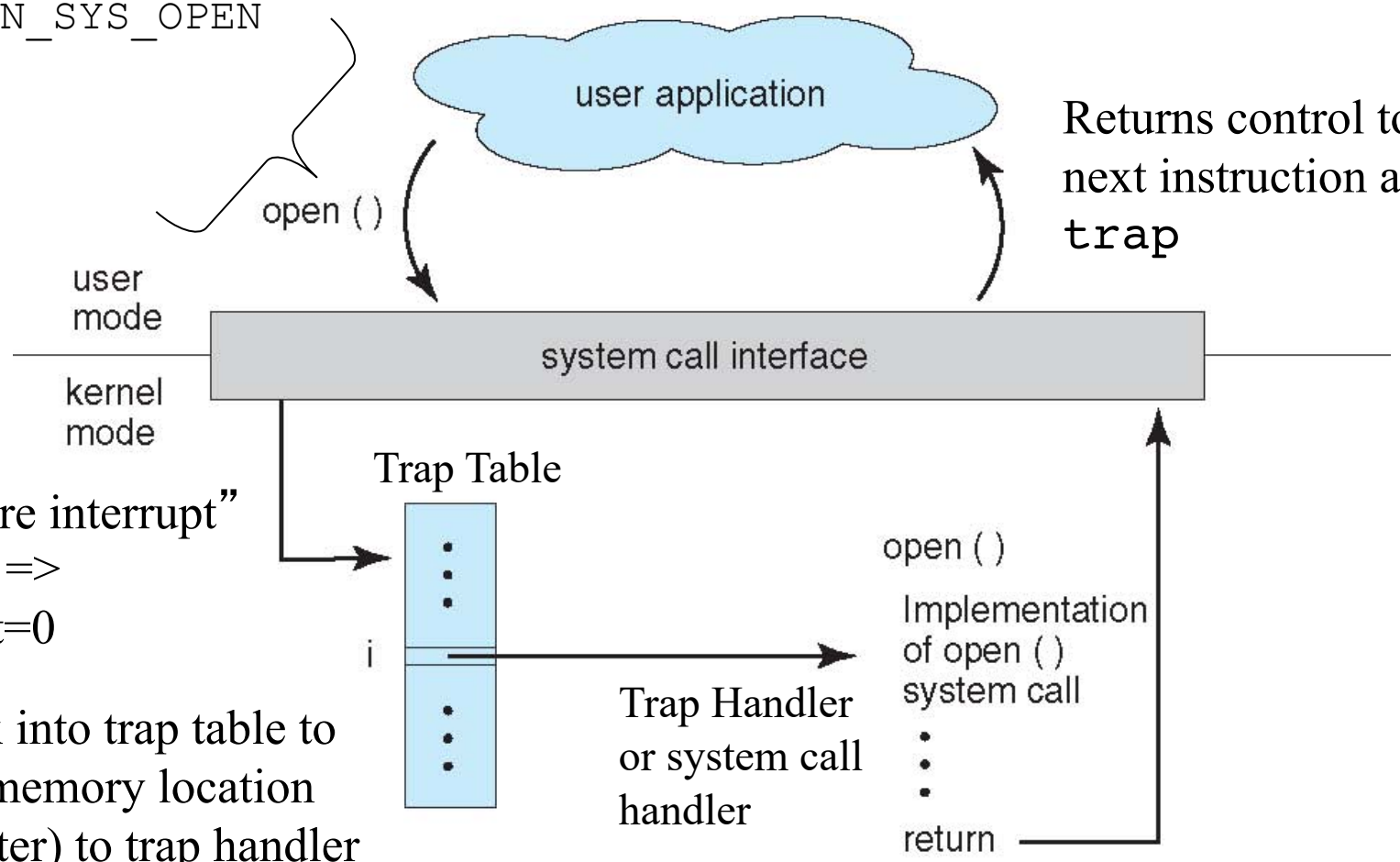
System Calls

- The `trap` instruction is used to switch from user to supervisor mode, thereby entering the OS
  - `trap` sets the mode bit to 0
  - On x86, use INT assembly instruction (more recently SYSCALL/SYSENTER)
  - mode bit set back to 1 on return
- Any instruction that invokes `trap` is called *a system call*
  - There are many different classes of system calls

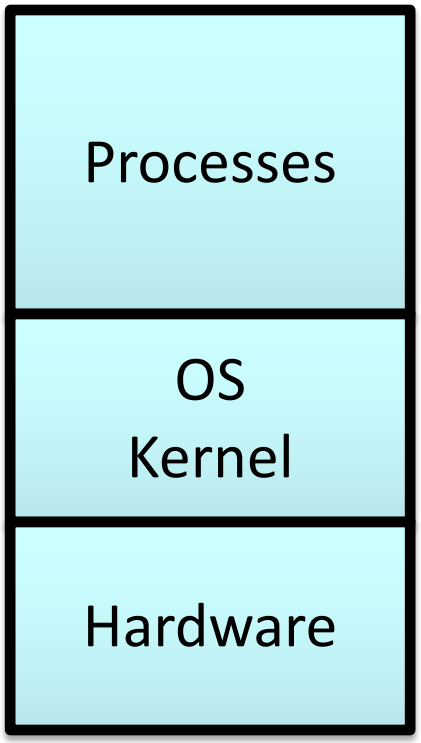


API – System Calls – OS Relationship

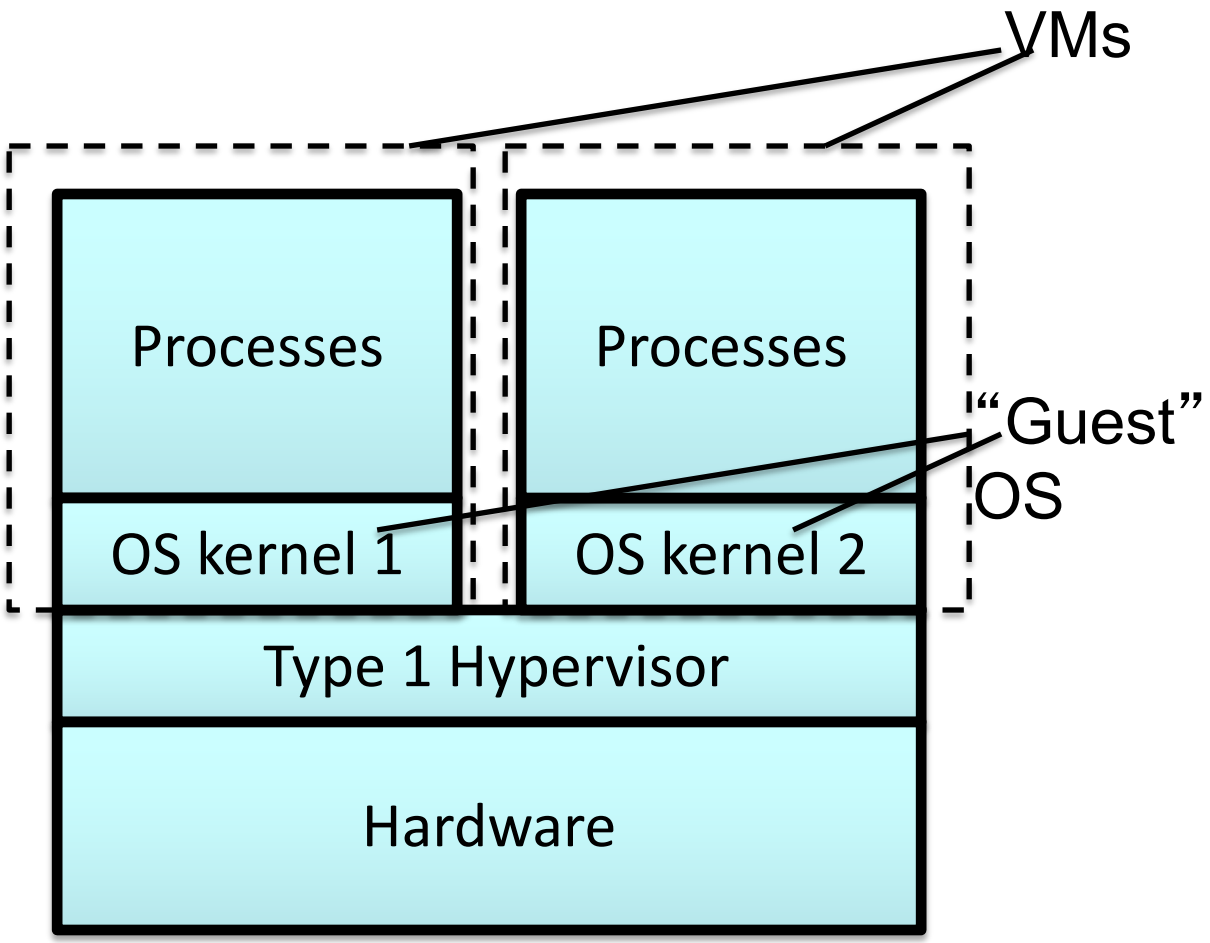
```
open() {
…
trap N_SYS_OPEN
…
}
```



Virtual Machines

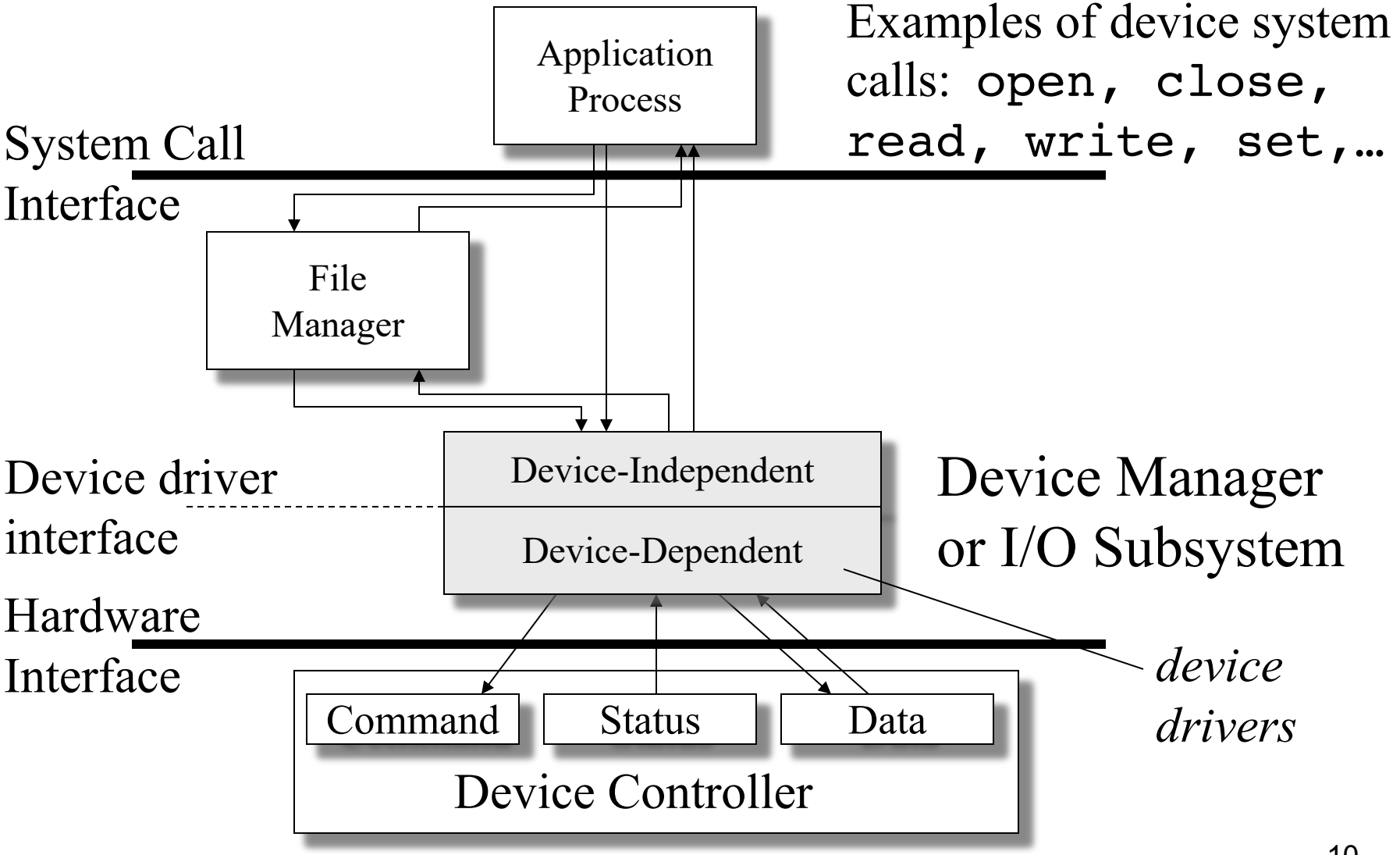


Traditional OS



A Type 1 *Hypervisor* provides a virtualization layer for guest OSs and resides just above the hardware.

Device Management Organization



Loadable Kernel Modules (LKM)

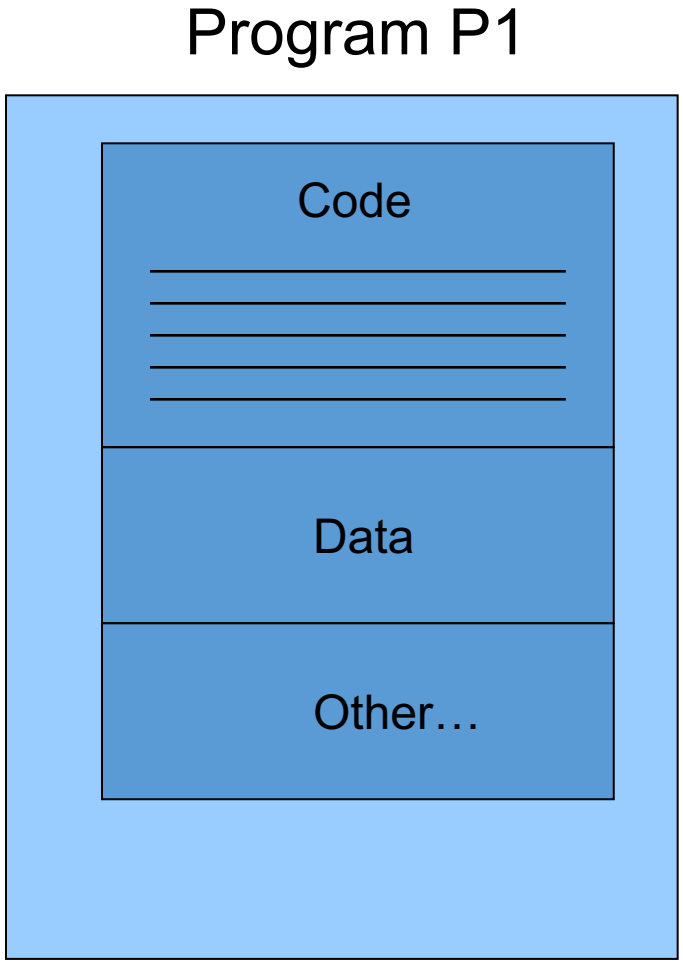
- LKM is an object file that contains code to extend a running kernel
- LKMs can be loaded and unloaded from kernel on demand at runtime
- LKMs offer an easy way to extend the functionality of the kernel without having to rebuild or recompile the kernel again
- LKMs are a simple and efficient way to create programs that reside in the kernel and run in privileged mode
- Most of the drivers are written as LKMs

Process

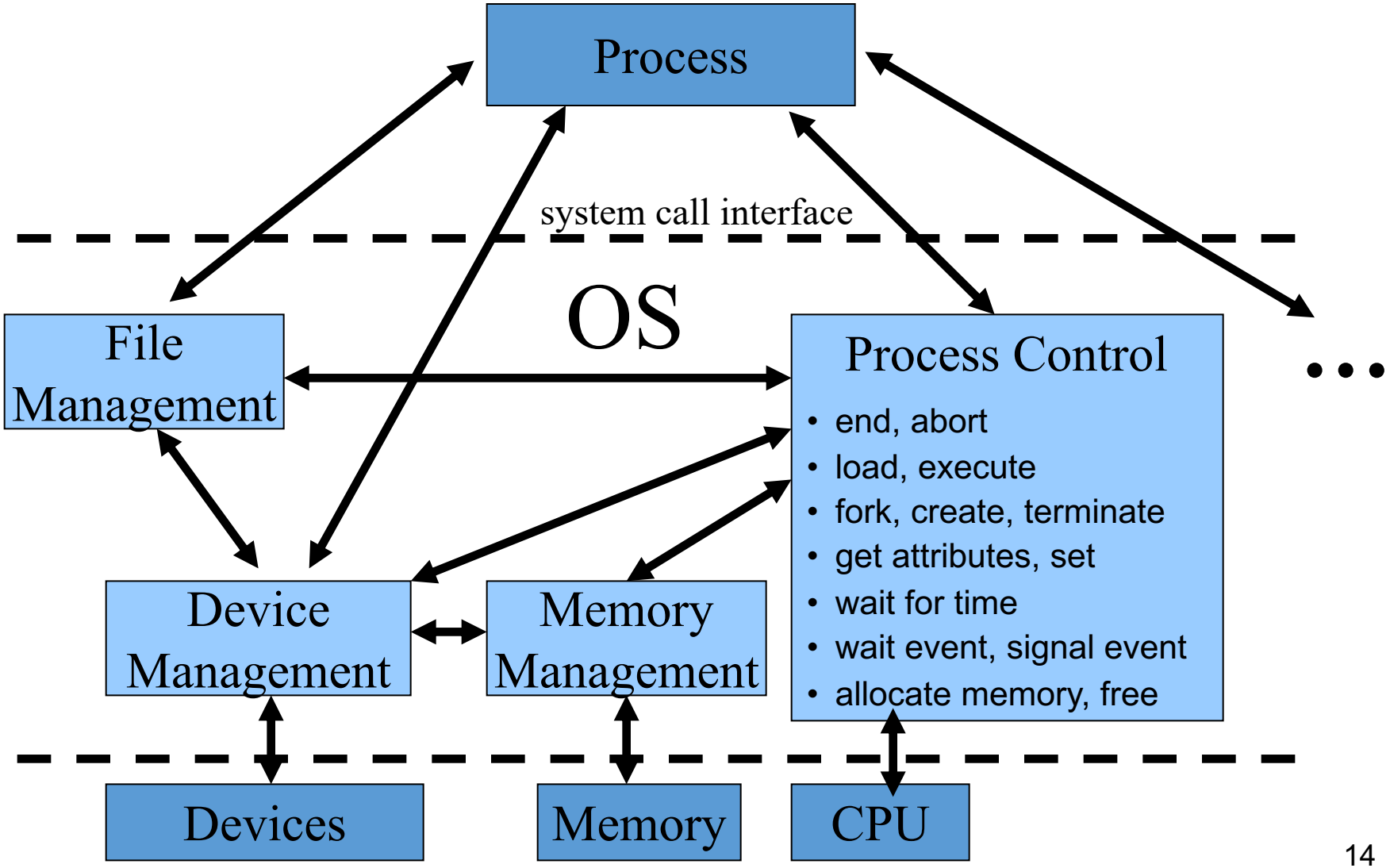
Process

- A software program consist of a sequence of code instructions and data stored on disk
- A program is a passive entity

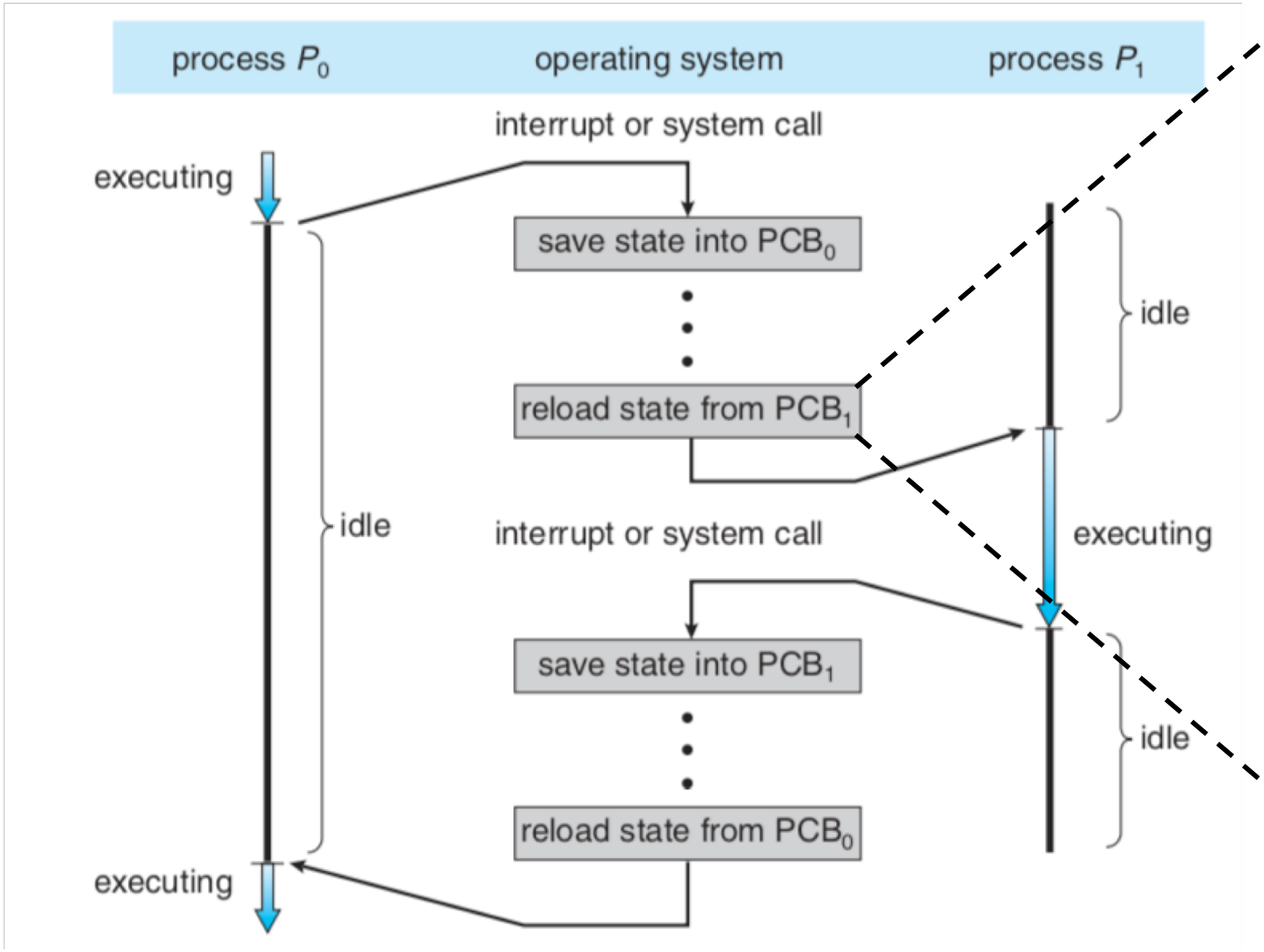
- A process is a program actively executing from main memory within its own address space



Process Management



Context Switching

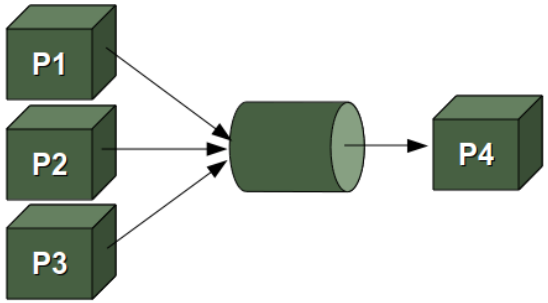


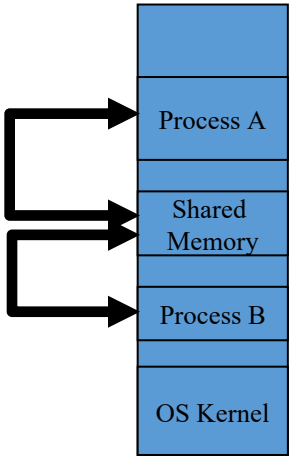
- Process state, e.g. ready, running, or waiting
- Accounting info, e.g. process ID
- Program Counter
- CPU registers
- CPU-scheduling info, e.g. priority
- Memory management info, e.g. base and limit registers, page tables
- I/O status info, e.g. list of open files

Interprocess Communication (IPC)

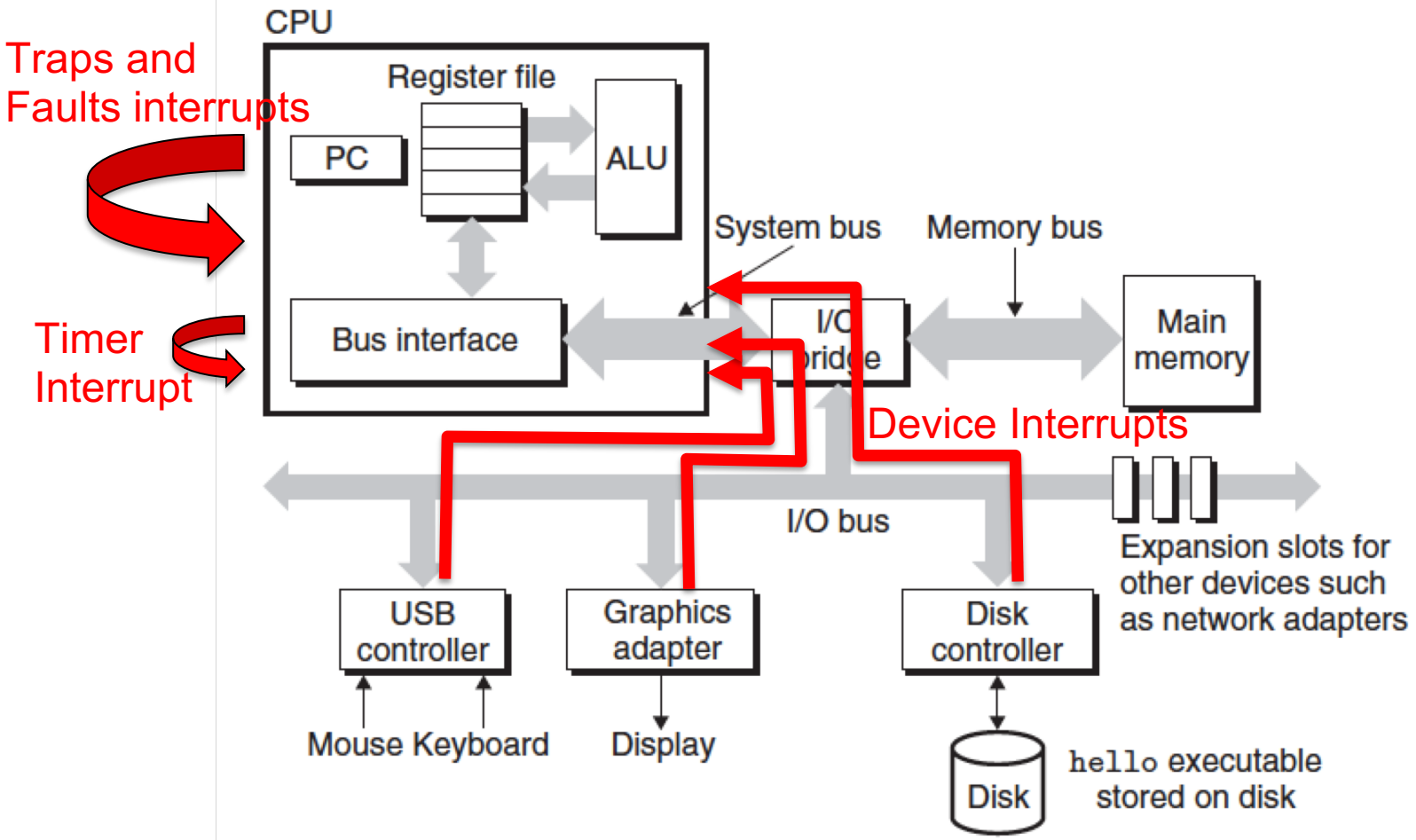
Interprocess Communication (IPC)

- Signals / Interrupts
  - Notifying that an event has occurred
- Message Passing
  - Pipes
  - Sockets
- Shared Memory
  - Race conditions
  - Synchronization
- Remote Procedure Calls





Linux Signals & Interrupts

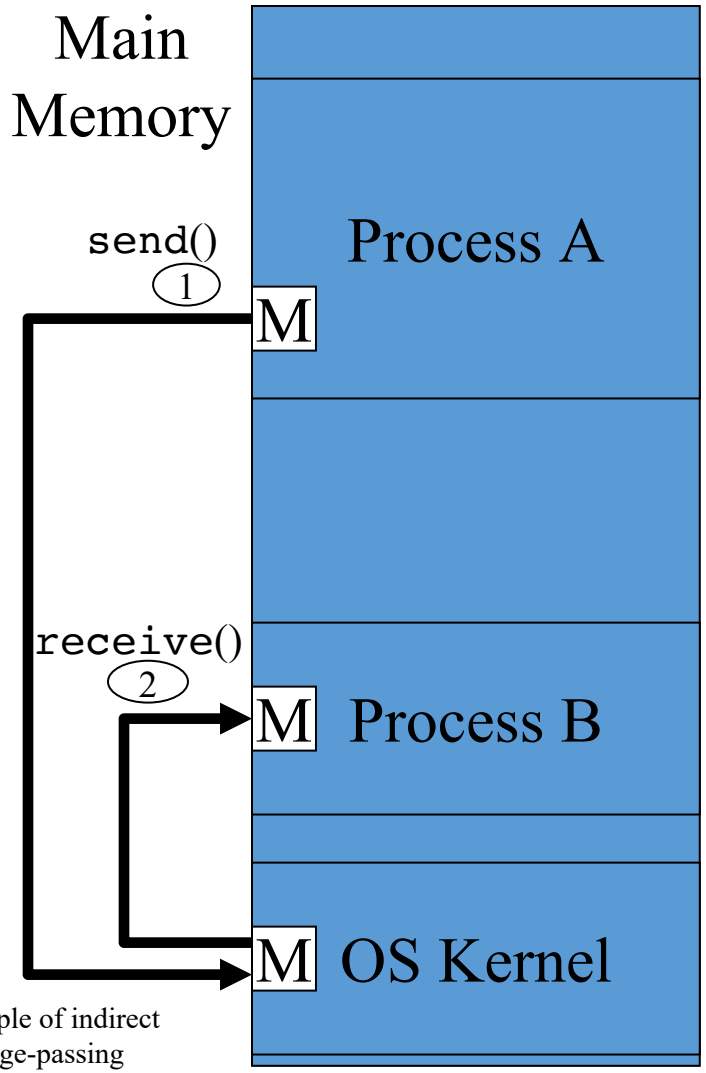


Signals

- Kernel-to-Process
- Process-to-Process

- Signals' issues
  - Blocking vs Non-blocking system calls
  - Synchronization vs Asynchronization
  - Race conditions

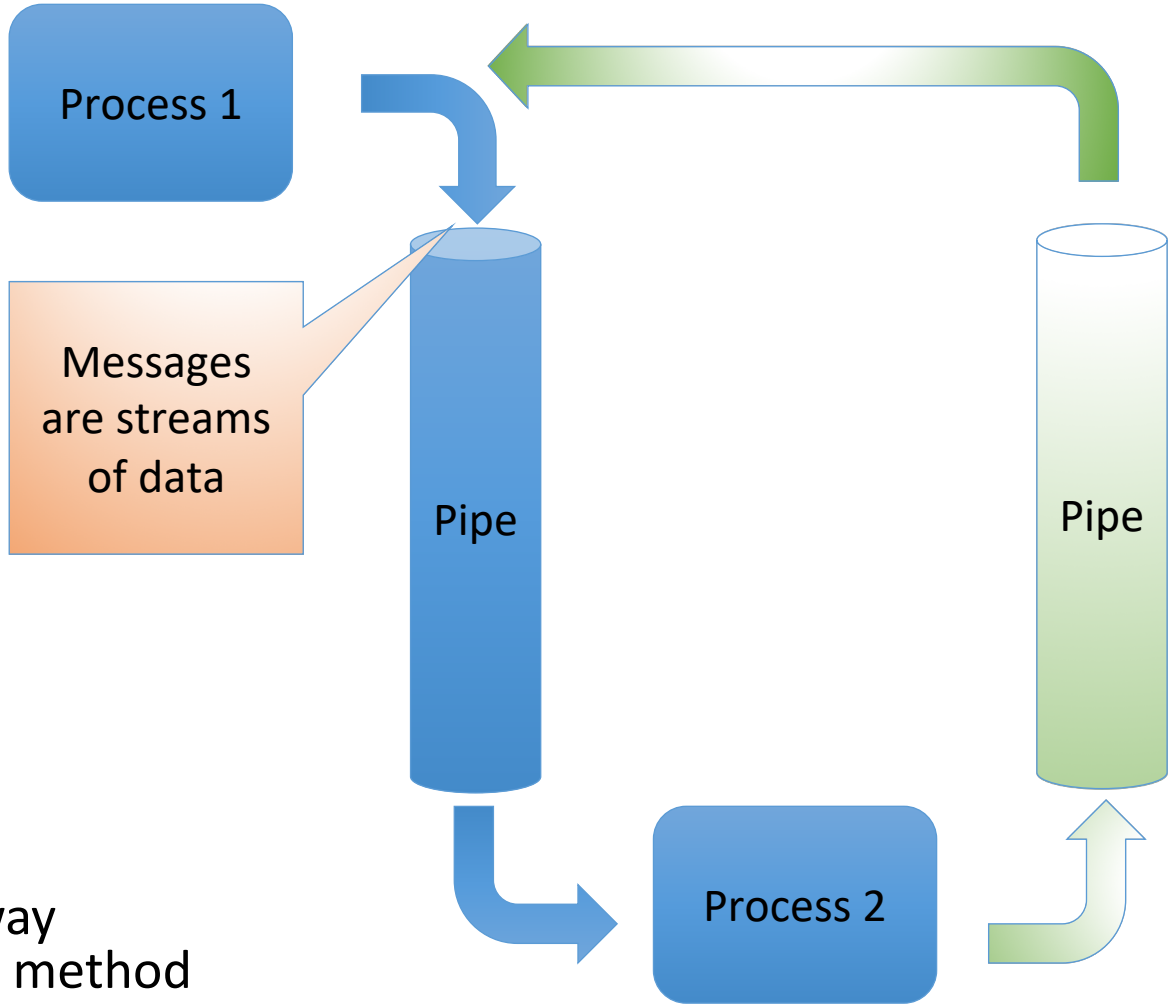
IPC Message Passing



Example of indirect message-passing

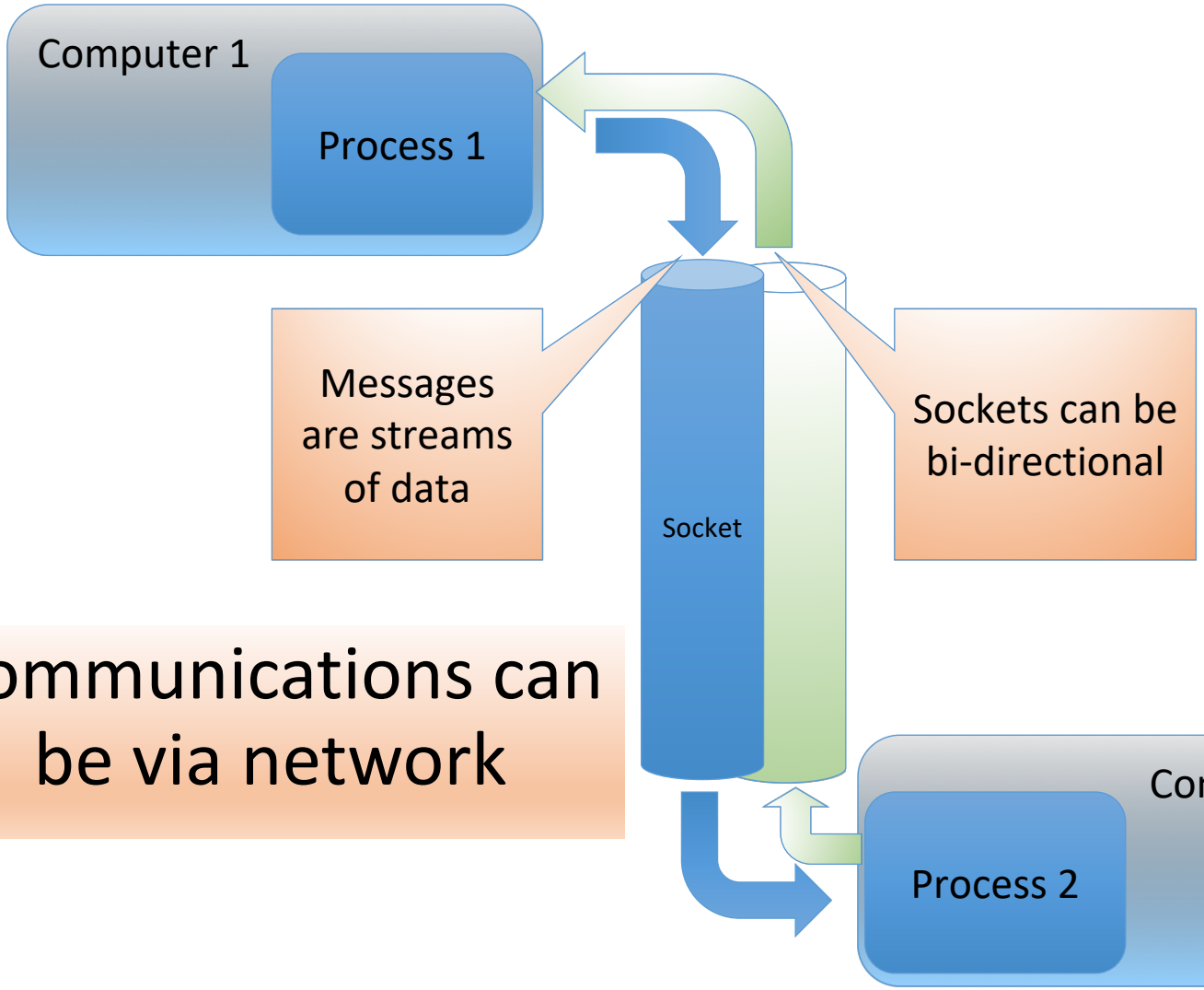
- `send()` and `receive()` can be blocking/synchronous or non-blocking/asynchronous
- Used to pass small messages
- Advantage: OS handles synchronization
- Disadvantage: Slow
  - OS is involved in each IPC operation for control signaling and possibly data as well
- Message Passing IPC types:
  - Pipes
  - UNIX-domain sockets
  - Internet domain sockets
  - message queues
  - remote procedure calls (RPC)

Message Passing via Pipes

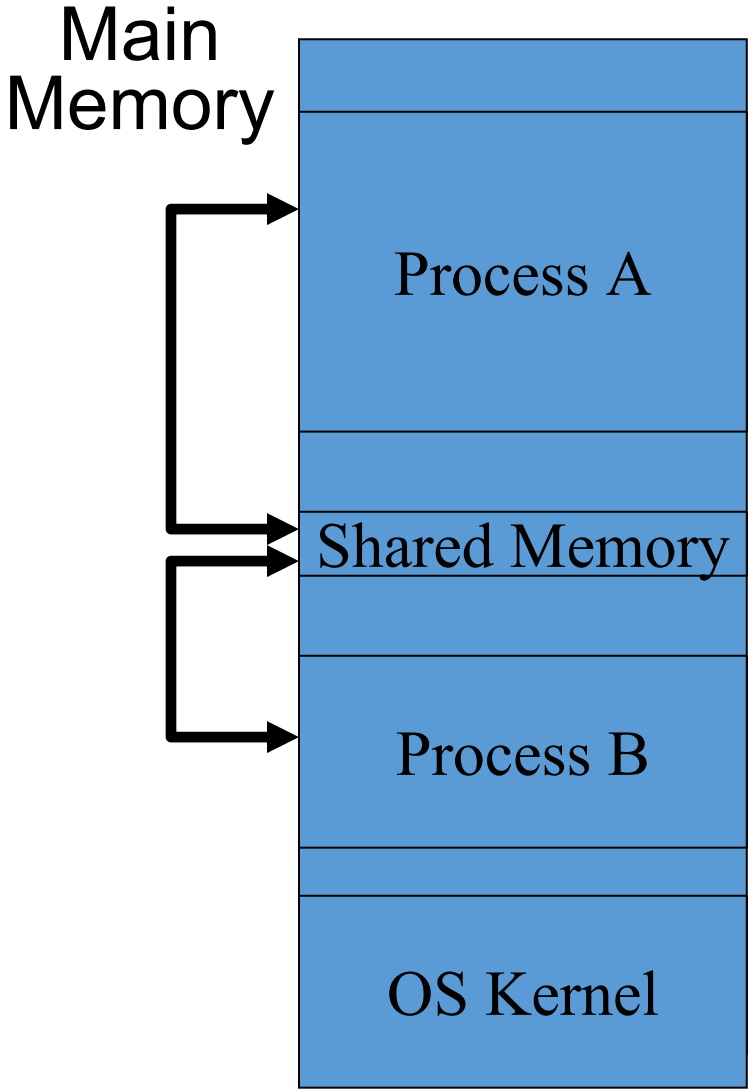


- Pipes are one way communication method

Message Passing via Sockets



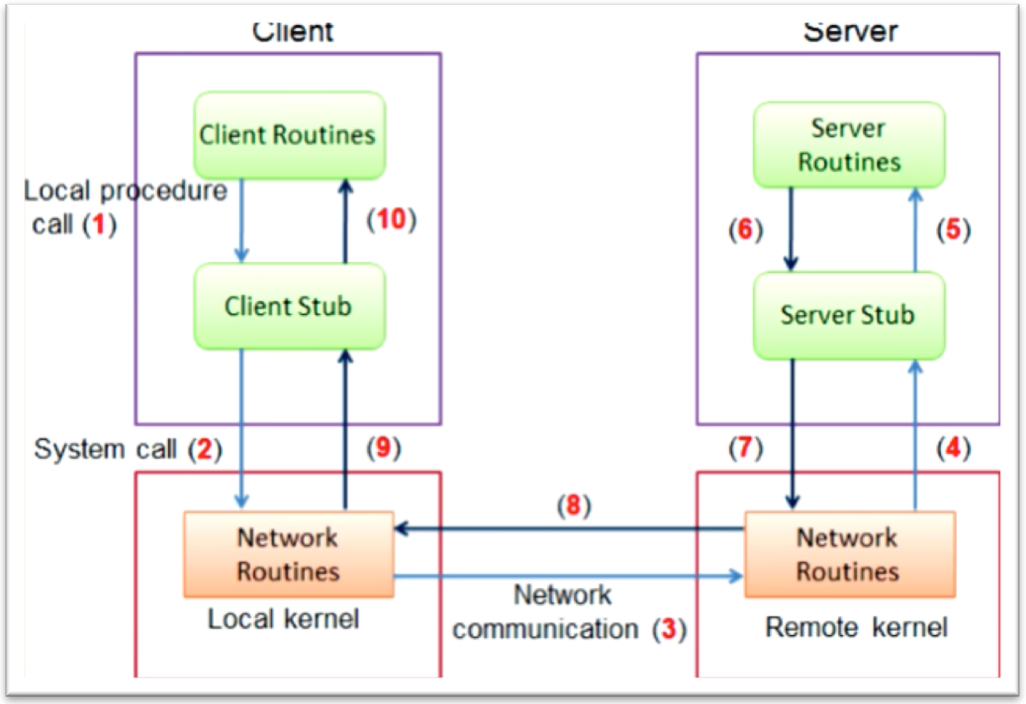
Shared Memory



- OS provides mechanisms for creation of a shared memory buffer between processes (both processes have address mapped to their process)
- Applies to processes on the same machine
- Problem: shared access introduces complexity
  - need to synchronize access

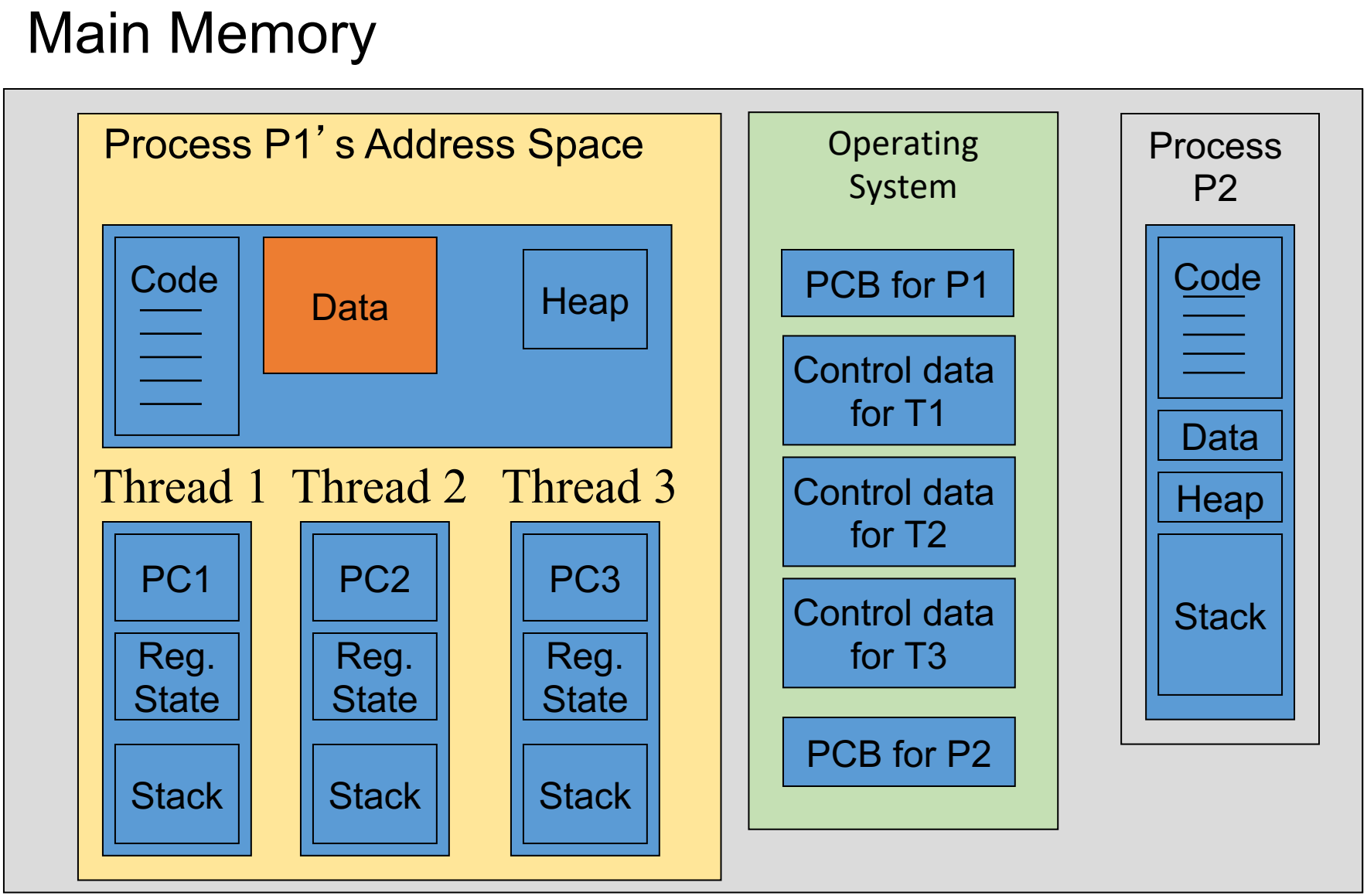
Remote Procedure Calls (RPC)

- Client makes a call to a function and passes parameters
- The client has linked a stub for the function. This stub will marshal (packetize) the data and send it to a remote server
- The network transfers the information to a server
- A service listening to the network receives a request
- The information is unmarshalled (unpacked) and the server's function is called.
- The results are returned to be marshalled into a packet
- The network transfers the packet is sent back to the client
- TCP/IP used to transmit packet



Threads

Threads


Main Memory
Process P1's Address Space
Code
Data
Heap
Thread 1
Thread 2
Thread 3
PC1
PC2
PC3
Reg. State
Reg. State
Reg. State
Stack
Stack
Stack
Operating System
PCB for P1
Control data for T1
Control data for T2
Control data for T3
PCB for P2
Process P2
Code
Data
Heap
Stack

Threads

- Process vs Thread
- Kernel vs User-space threads
  - Many-to-One model
  - One-to-One model
  - Many-to-Many model

Benefits vs Thread Safety

- Benefits
  - Responsiveness
  - Resource sharing
  - Low context-switching overhead
  - Scalability

- Thread safety
  - Thread safe
    - If the code behaves correctly during simultaneous or concurrent execution by multiple threads
  - Reentrant
    - If the code behaves correctly when a single thread is interrupted in the middle of executing the code reenters the same code

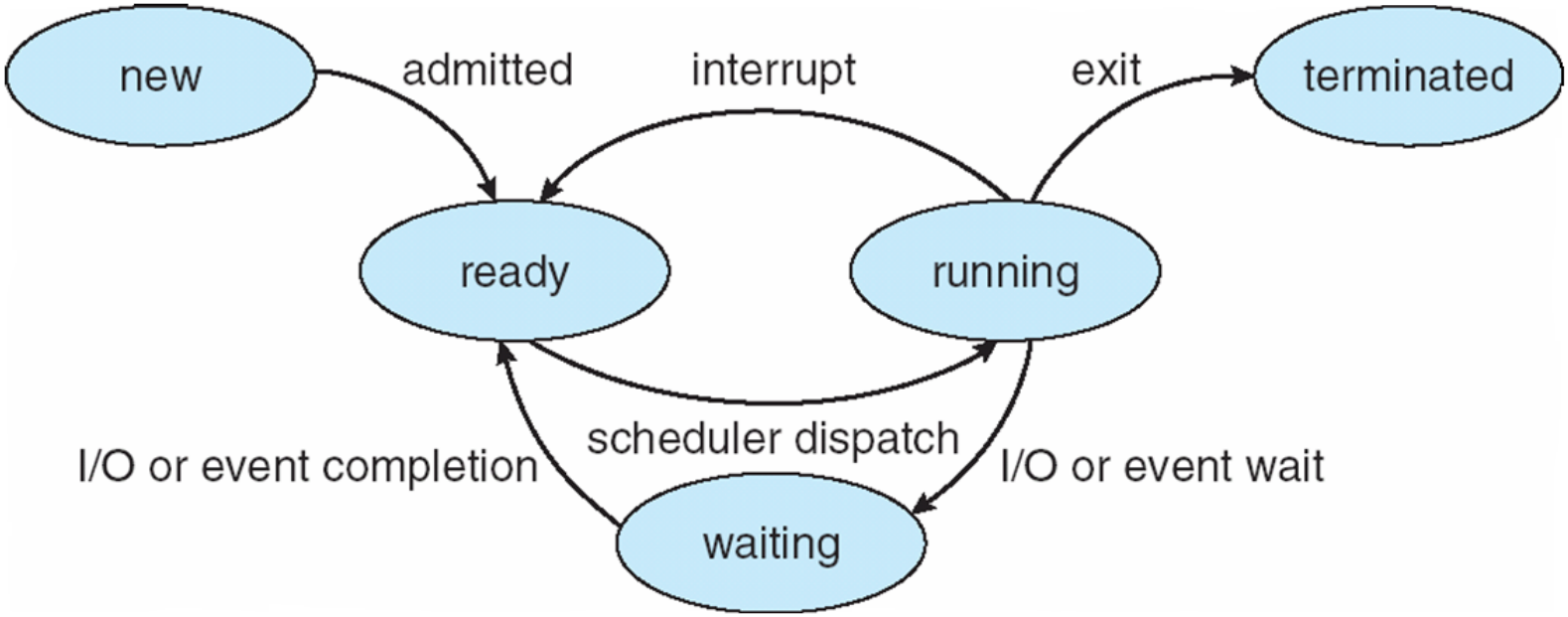
Synchronization

- Problems without synchronization
  - Race condition
    - Critical section
  - Deadlock
  - Starvation
- Synchronization mechanisms
  - Mutual exclusion
    - Atomic Test & Set
  - Semaphore
  - Monitor
  - Conditional variables

Synchronization

- Producer – Consumer (Bounded Buffer) problem
  - Problem description?
  - Solution?
- Reader – Writer problem
  - Problem description?
  - Solution?
- Dining Philosophers problem
  - Problem description?
  - Solution?

Diagram of Process State



Also called “blocked” state

Deadlock

Conditions

The following 4 conditions must hold simultaneously for deadlock to arise:

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

Deadlock

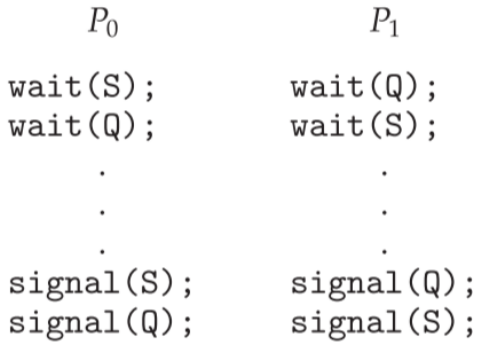
- Detection
- Prevention
- Avoidance

Question **1**

Not yet answered

Points out of 1.00

Below are two processes using the semaphores initialized as Q=1 and S=1. Which of the following is true regarding the execution of these processes?



Select one or more:

- [ ] a. P1 is subject to starvation
- [ ] b. P0 and P1 will always complete their execution
- [ ] c. P1 will always run before P0
- [ ] d. P0 will always run before P1
- [X] e. P0 and P1 can result in deadlock
- [ ] f. P0 is subject to starvation

Question **1**

Not yet answered

Points out of 5.00

```
int temp;

void swap(int *y, int *z)
{
    int local;
    local = temp;
    temp = *y;
    *y = *z;
    *z = temp;
    temp = local;
}
```

Select the best answer regarding the given code snippet.

Select one:

- a. The code is re-entrant and thread safe
- b. The code is thread safe but not re-entrant
- c. The code is neither thread safe nor re-entrant
- d. The code is re-entrant but not thread safe

Question **1**

Not yet answered

Points out of 5.00

```
int temp;

void swap(int *y, int *z)
{
    int local;
    local = temp;
    temp = *y;
    *y = *z;
    *z = temp;
    temp = local;
}
```

Select the best answer regarding the given code snippet.

Select one:

- a. The code is re-entrant and thread safe
- b. The code is thread safe but not re-entrant
- c. The code is neither thread safe nor re-entrant
- d. The code is re-entrant but not thread safe

Sample Question

Multiple Choice Questions: Choose one option that answers the question best.

1. Advantages of threads over processes include

A. lower context switch time
B. no possibility of race conditions
C. sharing of heap and stack
D. smaller code size
E. All of the above

Sample Question

Multiple Choice Questions: Choose one option that answers the question best.

2. Which of the following is FALSE about IPC via pipes?

A. Basic primitives are send() and receive().
B. Communication can be blocking or non-blocking.
C. Pipes can be anonymous or named.
D. Pipes can be used for only one-way communication.
E. IPC via pipes is slower than IPC using shared memory.

Sample Question

Multiple Choice Questions: Choose one option that answers the question best.

3. Which of the following is NOT required for a system to be in deadlock?

A. mutual exclusion

B. no preemption

C. acquire and hold

D. circular dependency

Sample Question

Short Answer Questions: Consider the following program code for the next question (Question 1).

```
int ret = fork();
if (ret == 0){
   ret = fork();
   printf("Hello \n");
}
printf("World \n");
return 0;
```

1. How many times the following word will be printed?

   A. "Hello": ____ times

   B. "World": ____ times

Sample Question

Short Answer Questions: Consider the following program code for the next question (Question 1).

```
int ret = fork();
if (ret == 0){
    ret = fork();
    printf("Hello \n");
}
printf("World \n");
return 0;
```

1. How many times the following word will be printed?

A. "Hello": 2 times

B. "World": 3 times

Sample Question

Short Answer Questions: Consider the following program code for the next question (Question 2).

```
int t;

void swap(int *x, int *y)
{
    int s;
    s = t;
    t = *x;
    *x = *y;
    *y = t;
    t = s;
}
```

2. Is there a possibility of a race condition updating variable t in the code above?

Sample Question

Short Answer Questions: Consider the following program code for the next question (Question 2).

```
int t;

void swap(int *x, int *y)
{
    int s;
    s = t;
    t = *x;
    *x = *y;
    *y = t;
    t = s;
}
```

2. Is there a possibility of a race condition updating variable t in the code above? YES

Sample Question

Short Answer Questions:

3. Mark each term with the letter of the correct statement.

A. A piece of code functions correctly during simultaneous or concurrent execution by multiple threads.	C. A function defined in the Linux kernel to copy data from kernel-space.
B. A function defined in the Linux kernel to copy data to kernel-space.	D. The procedure for replacing the currently executing process with another.

Thread-safe	**copy_to_user()**

Sample Question

Short Answer Questions:

3. Mark each term with the letter of the correct statement.

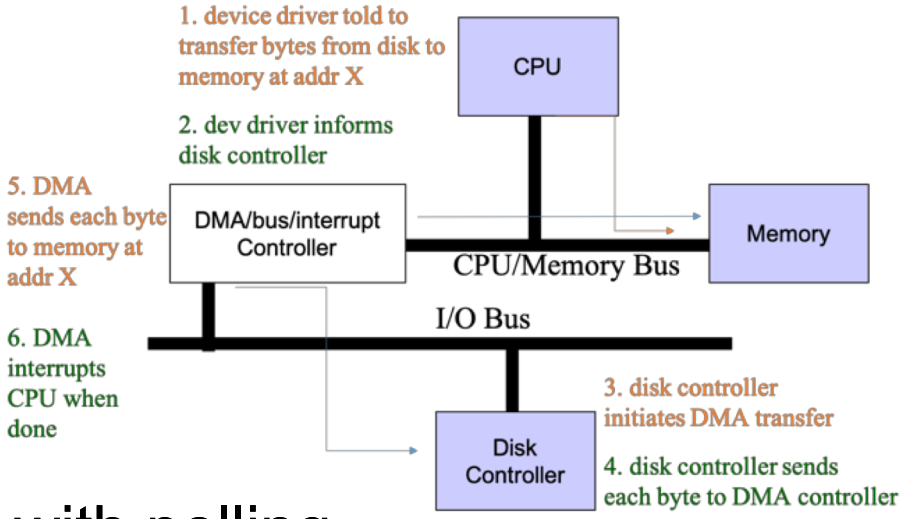
A. A piece of code functions correctly during simultaneous or concurrent execution by multiple threads.	C. A function defined in the Linux kernel to copy data from kernel-space.
B. A function defined in the Linux kernel to copy data to kernel-space.	D. The procedure for replacing the currently executing process with another.

Thread-safe	**copy_to_user()**
A	C

Sample Question

Short Answer Questions:

4. What is the I/O strategy of the device manager illustrated the image below?



A. direct I/O with polling

B. direct I/O with interrupts

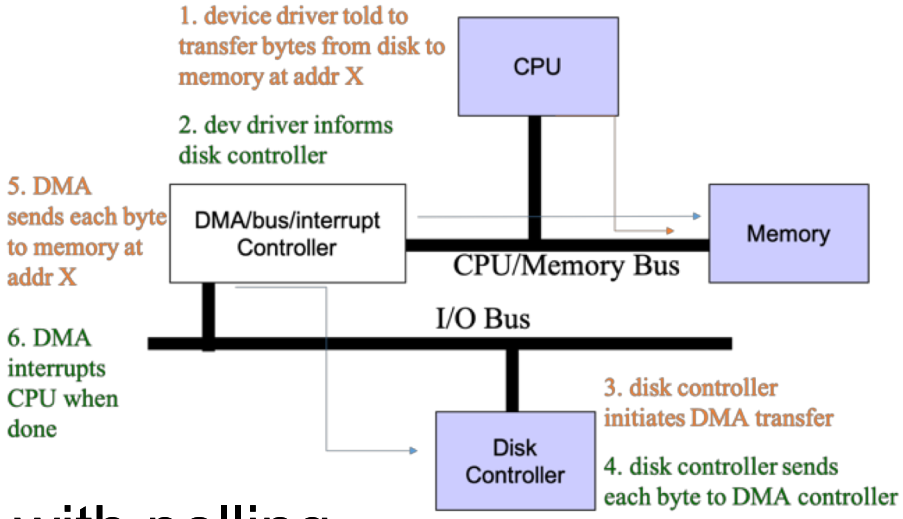
C. DMA with interrupts

D. hardware interrupts

Sample Question

Short Answer Questions:

4. What is the I/O strategy of the device manager illustrated the image below?



A. direct I/O with polling

B. direct I/O with interrupts

C. DMA with interrupts

D. hardware interrupts

Sample Question

1. Consider three processes, P, Q and R with the following code:
   - P: ps1; ps2; ps3; ps4;
   - Q: qs1; qs2; qs3; qs4;
   - R: rs1; rs2; rs3; rs4;

These processes have the following synchronization constraint:

a) rs1 must be the first statement to execute
b) process R should be the last process to exit
c) ps3 must execute after qs2
d) qs3 must execute after ps4 and rs3
e) rs4 must execute after either ps4 or qs4 (or both) have executed

Using Condition Variables, provide updated codes for P, Q and R that satisfy these constraints.

Sample Question

Process P	Process Q	Process R
<ps1>	<qs1>	<rs1>
<ps2>	<qs2>	<rs2>
<ps3>	<qs3>	<rs3>
<ps4>	<qs4>	<rs4>

Sample Question

Process P	Process Q	Process R
wait(&s1)	wait(&s1)	
<ps1>	<qs1>	<rs1>
		signal(&s1)
		signal(&s1)
<ps2>	<qs2>	<rs2>
<ps3>	<qs3>	<rs3>
<ps4>	<qs4>	<rs4>

Sample Question

Process P	Process Q	Process R
wait(&s1)	wait(&s1)	
<ps1>	<qs1>	<rs1>
		signal(&s1)
		signal(&s1)
<ps2>	<qs2>	<rs2>
wait(&s2)	signal(&s2)	
<ps3>	<qs3>	<rs3>
<ps4>	<qs4>	<rs4>

Sample Question

Process P	Process Q	Process R
wait(&s1)	wait(&s1)	
<ps1>	<qs1>	<rs1>
		signal(&s1)
		signal(&s1)
<ps2>	<qs2>	<rs2>
wait(&s2)	signal(&s2)	
	wait(&s3)	
<ps3>	<qs3>	<rs3>
		signal(&s3)
<ps4>	<qs4>	<rs4>
signal(&s3)		

Sample Question

Process P	Process Q	Process R
wait(&s1)	wait(&s1)	
<ps1>	<qs1>	<rs1>
		signal(&s1)
		signal(&s1)
<ps2>	<qs2>	<rs2>
wait(&s2)	signal(&s2)	
	wait(&s3)	
<ps3>	<qs3>	<rs3>
		signal(&s3)
		wait(&s4)
<ps4>	<qs4>	<rs4>
signal(&s3)	signal(&s4)	
signal(&s4)		

Sample Question

Process P	Process Q	Process R
wait(&s1)	wait(&s1)	
<ps1>	<qs1>	<rs1>
		signal(&s1)
		signal(&s1)
<ps2>	<qs2>	<rs2>
wait(&s2)	signal(&s2)	
	wait(&s3)	
<ps3>	<qs3>	<rs3>
		signal(&s3)
		wait(&s4)
<ps4>	<qs4>	<rs4>
signal(&s3)	signal(&s4)	wait(&s5)
signal(&s4)	signal(&s5)	
signal(&s5)		