

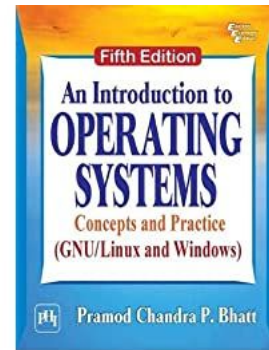
CS310 Operating Systems

Lecture 18: Inter Process Communication – Shared Memory

Ravi Mittal
IIT Goa

Reading

- Book: An introduction to Operating System: Concepts and Practices, Pramod Chandra Bhatt, 5th Edition,
 - Chapter 7, Section 7.3.4

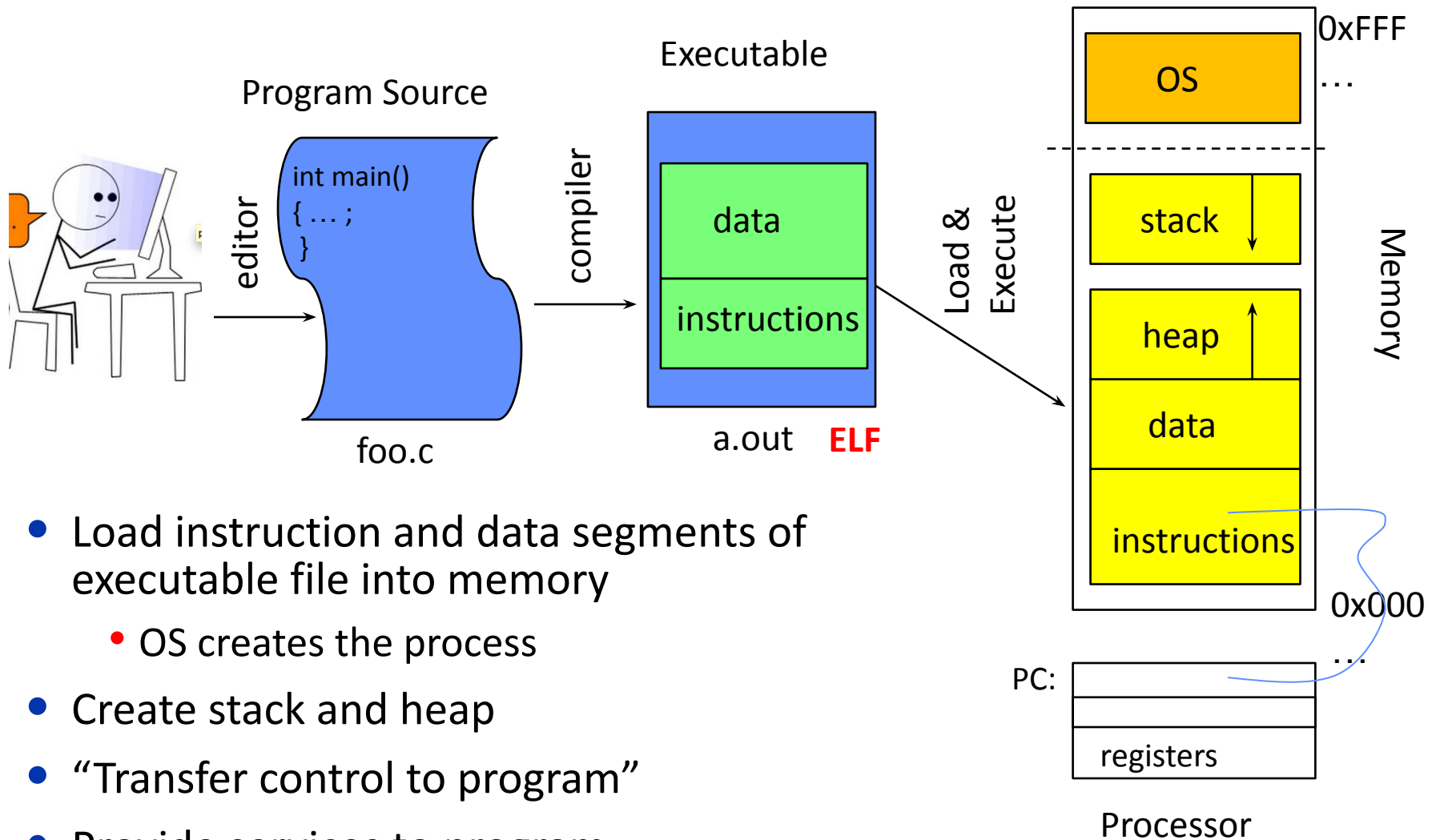


Today we will study

- Where are we in CS310?
- Shared Memory Concept
- Garbage Collection

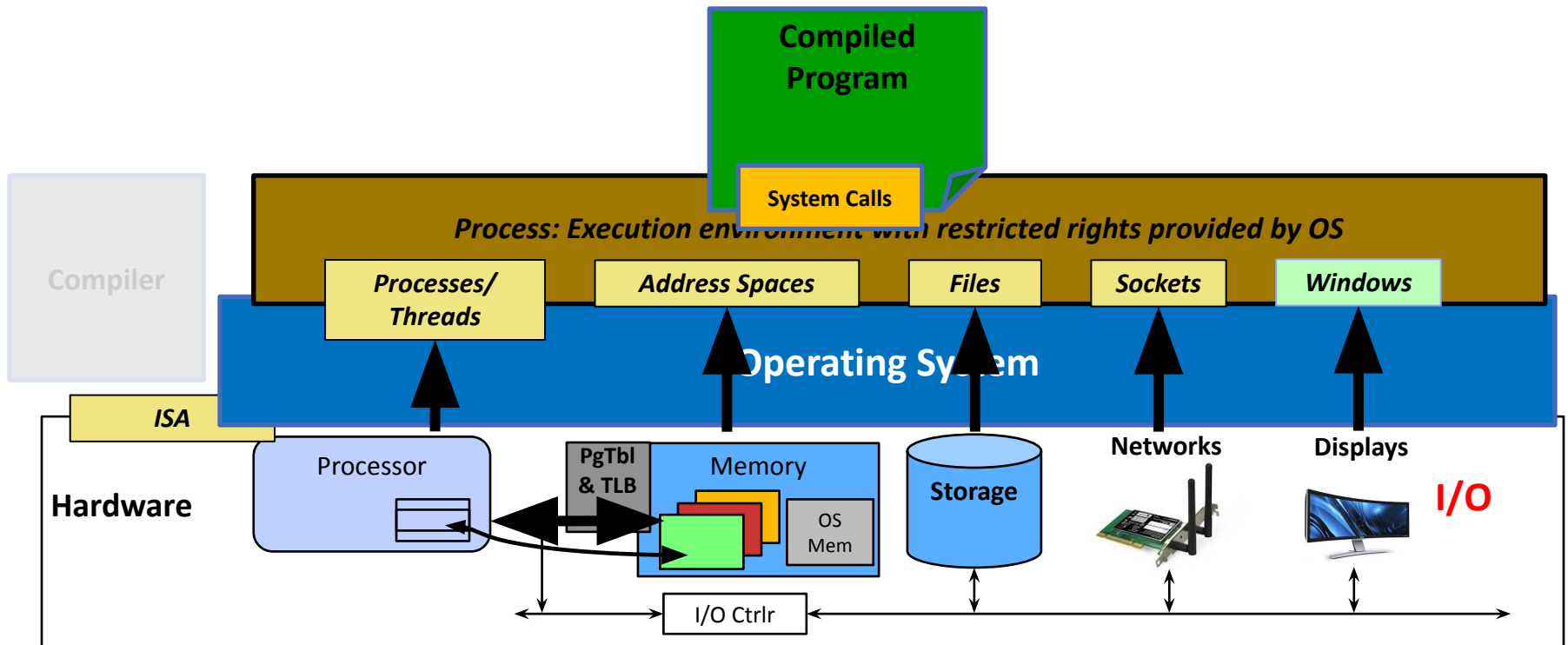
Where are we in CS310?

Recall: OS Bottom Line: Run Programs



ELF: Executable and Linking Format

Recall: Operating System



Topics

- OS Concepts: Top Level view
 - How does the OS work?
 - Multiprogramming Concept
- Process Concept, PCB, Context Switching
- How processes interact with OS for use of hardware – System Calls
- Threads
 - Concept
- Inter-process Communication
 - Message Passing and Shared Memory
- Concurrency Management
- File System, I/O
- Security
- Cloud Infrastructure

Two important Requirements

- When processes/threads interact with each other, they must satisfy the following requirements:
- Synchronization
 - Processes may need to be synchronized to enforce mutual exclusion
- Communication
 - Cooperating processes may need to exchange information

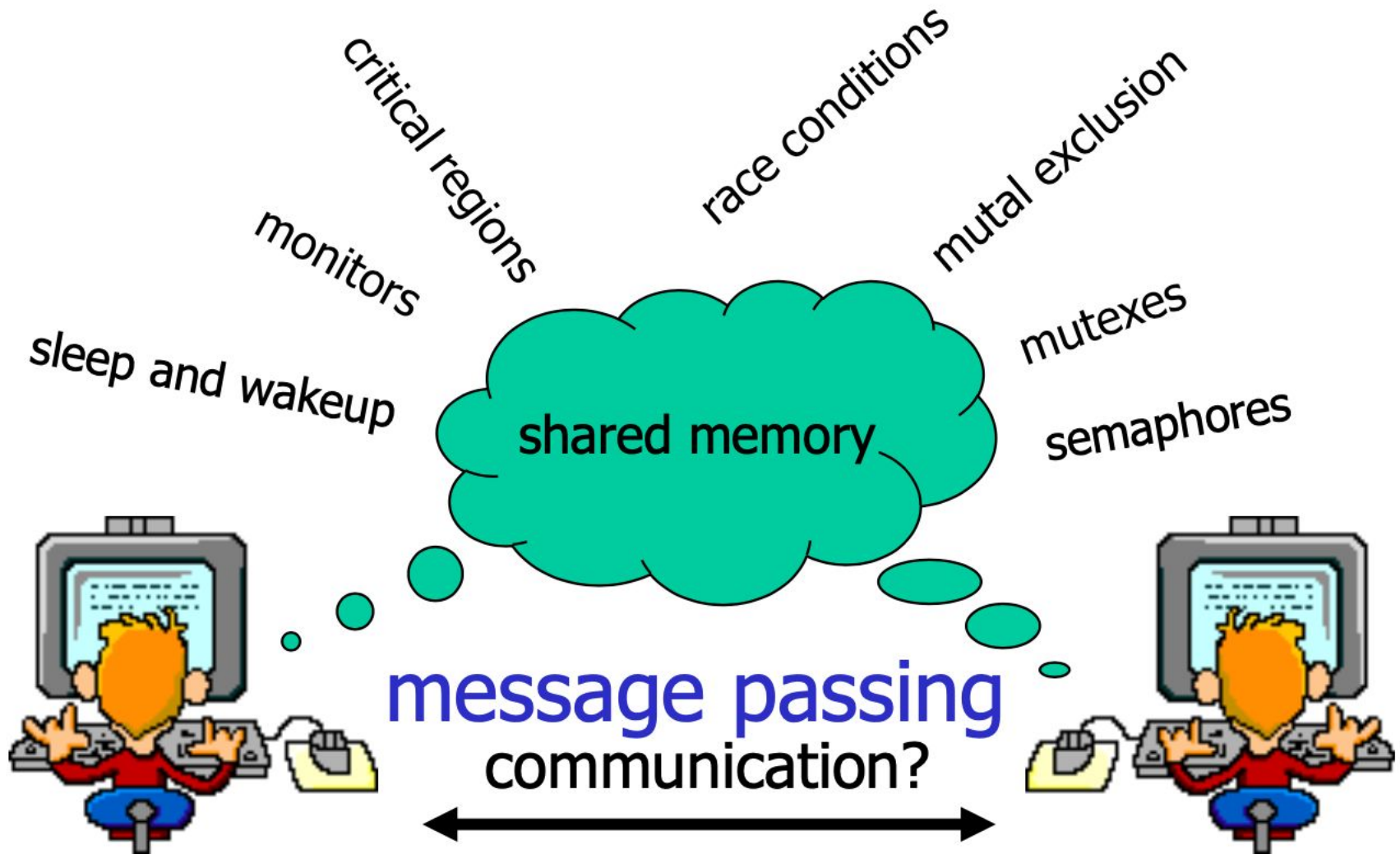
Concurrent Execution of Processes / Threads

- Concurrency is inherent feature of multiprogramming, multiprocessing, distributed systems, and client-server model of computation
- Implementation of systems using concurrent Threads/Processes require
 - Communication between Processes/Threads
 - Synchronization among processes/Threads
 - Sharing of common resources

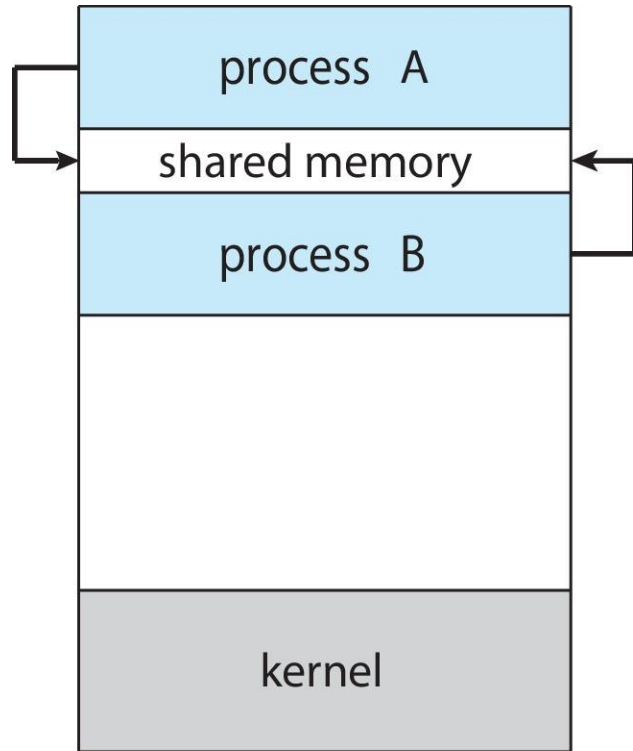
Interactions among processes

- **Competing processes:** Processes themselves do not share anything. But OS has to share the system resources among these processes “competing” for system resources such as disk, file or printer.
- **Cooperating processes :** Results of one or more processes may be needed for another process. Reasons for cooperation:
 - Information sharing
 - Computation Speedup
 - Modularity
 - Convenience

IPC – Big Picture

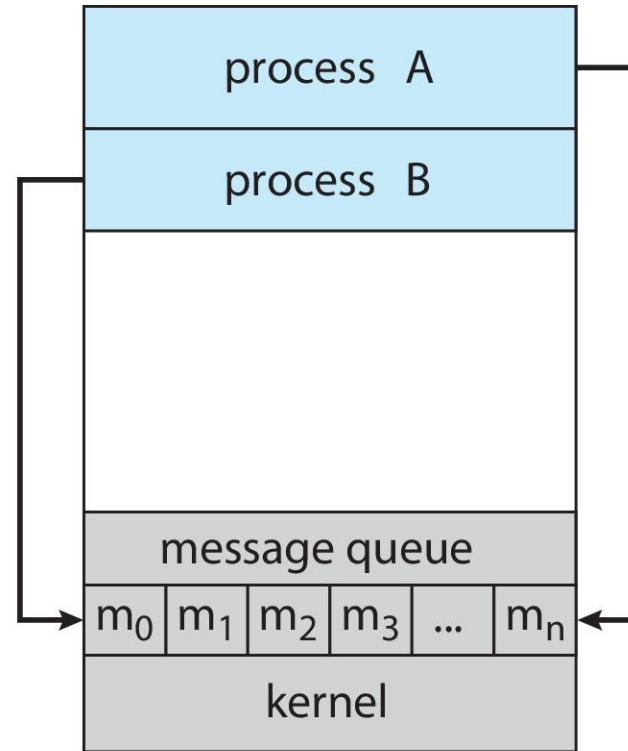


Models of IPC: Shared Memory, Message Passing



(a)

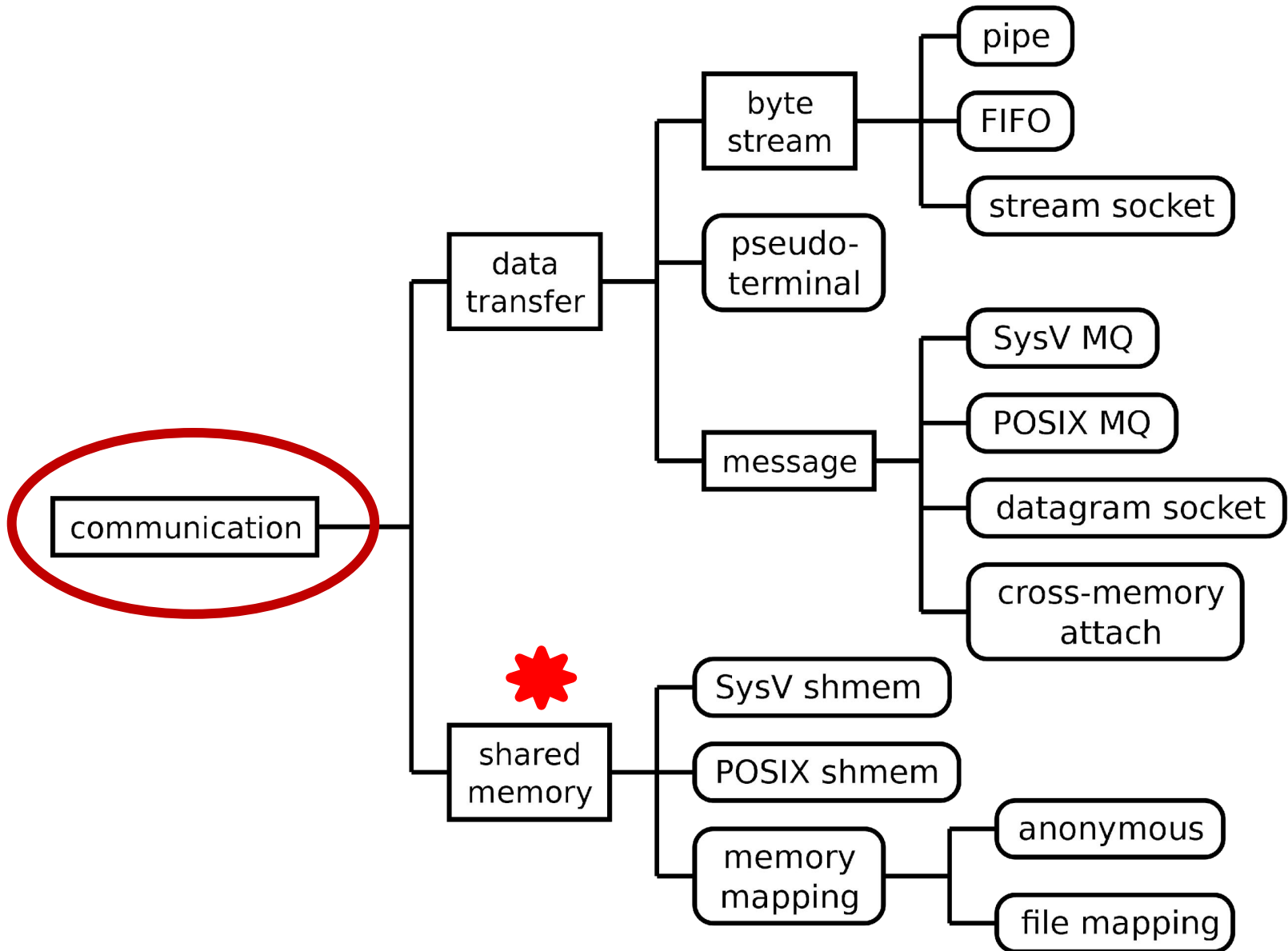
Shared memory



(b)

Message passing.

Communication



Models of IPC: Shared Memory, Message Passing

Both models are commonly used in operating systems

- **Shared Memory**

- Faster than message passing as message passing is implemented using system calls (generally slow)
- System Calls are needed only to establish shared memory regions
 - Once shared memory is created, access is the same as routine memory access – thus no help from OS is required

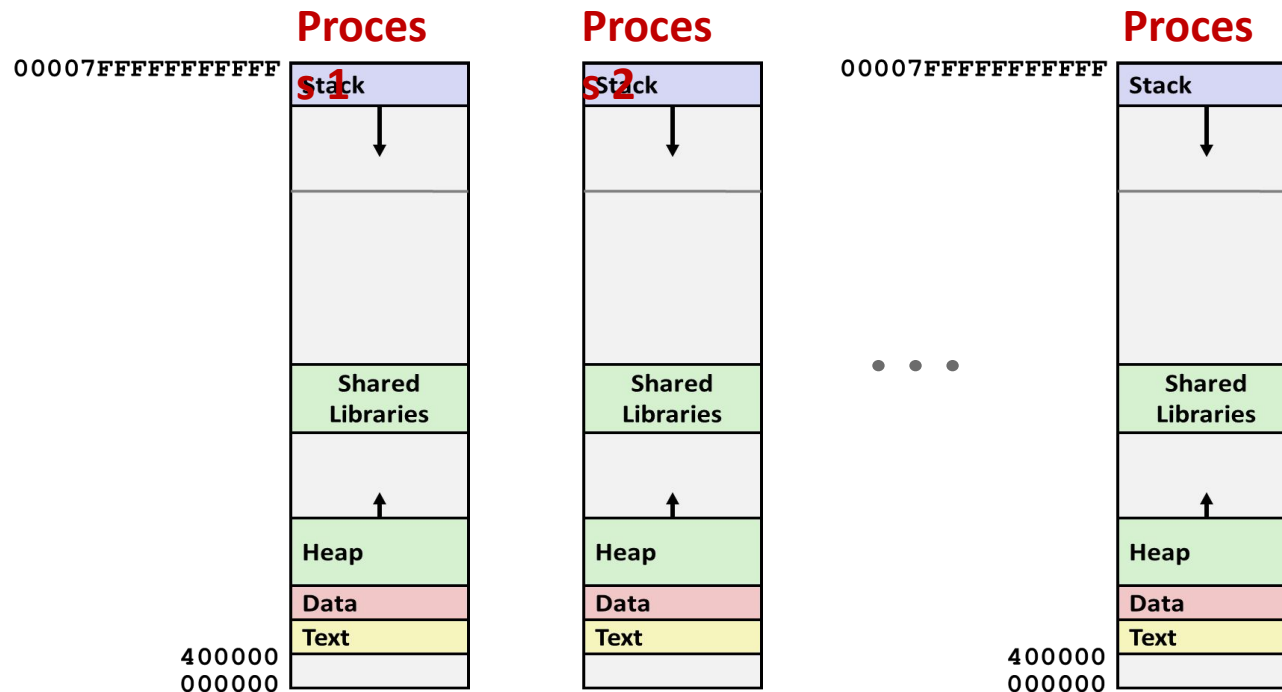
- **Message passing**

- Useful for exchanging small amount of data
- Easier to implement in distributed systems
- Usually implemented using system calls

Shared Memory Concept

Shared Memory Concept

- Requires communicating processes to establish a region of shared memory
- How to create shared memory region – when each process has it's own address space?

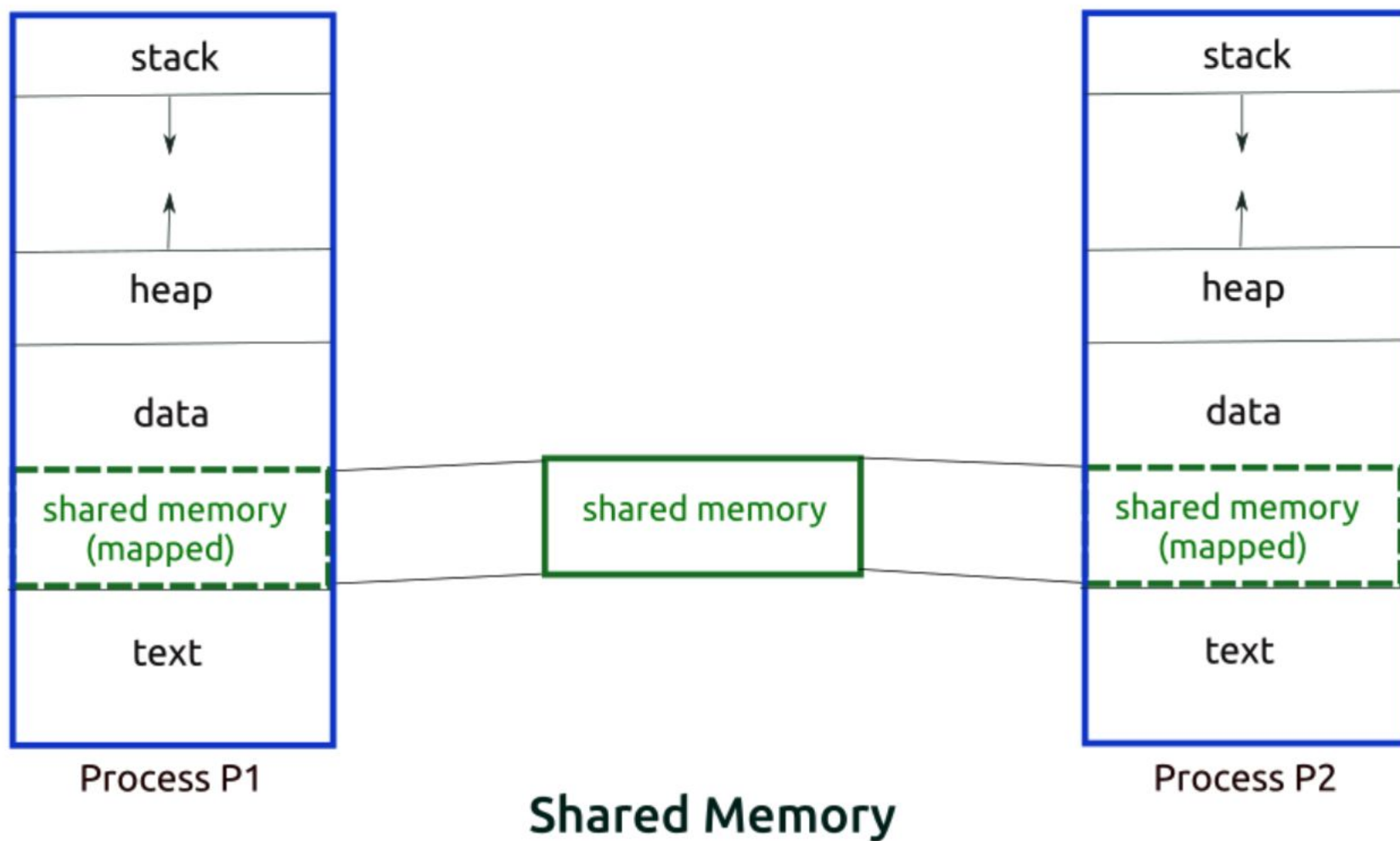


Shared Memory Concept

- A shared-memory region resides in the address space of the process creating the shared-memory segment
- Any process that wish to communicate using this shared memory segment must attach it to their address space
- Shared memory requires that two or more processes agree to remove the restriction of non-sharing in their address space
- They can exchange information by reading and writing data into the shared areas
 - Data and location are determined by processes
 - Not under control of OS
- Processes are responsible for ensuring that they are not writing the same location simultaneously
 - Producer Consumer Algorithm

Shared Memory Model - Steps

- Initiate shared memory mechanism with the help of kernel
- Kernel identifies “safe area” that is attached to both processes
- Attached shared area is used in consistent manner
- When finished, shared data area is detached from all processes which it was attached
- Information regarding the shared memory is deleted from the kernel
- Note that
 - shared memory segment is created by the kernel and mapped to the data segment of the address space of a requesting processes
 - A process can use the shared memory just like any other global variable in its address space



Shared Memory – Fast access

- Processes can read and write shared memory region (segment) directly as ordinary memory access (like they are accessing memory variables directly without kernel help)
 - During this time, kernel is not involved.
 - It is fast – there is no Kernel access
- Race condition can occur

Procedure for using Shared Memory

- A system wide key is used for identifying shared memory segments
 - Find this key (how?)
 - Use `ftok()` system call to generate key using a file path and bits of project id
 - This info is also be known to other process so they can also generate the key
- Use `shmget()` to allocate a shared memory
 - Get shared memory id (`shmid`) as return value
- Use `shmat()` to attach a shared memory to an address space
 - Use `shmid` as one of the parameter
 - Any process that has `shmid` can now attach
- Once communication is over
- Use `shmdt()` to detach a shared memory from an address space
- Use `shmctl()` to deallocate a shared memory

Shared Memory: key generation (1/3)

- To use shared memory, include the following:
 - `#include <sys/types.h>`
 - `#include <sys/ipc.h>`
 - `#include <sys/shm.h>`
- A key is a value of type `key_t`.
- There are three ways to generate a key:
 - Do it yourself
 - Use function `ftok()`
 - Ask the system to provide a private key

Shared Memory: key generation (2/3)

- Do it yourself: use

```
key_t  SomeKey;
```

```
SomeKey = 1234;
```

- Use `ftok()` to generate one for you:

- `key_t = ftok(char *path, int ID);`

- path is a path name (e.g., `“./”`)

- ID is an integer (e.g., `‘a’`), or Project Id bits

- Function `ftok()` returns a key of type `key_t`:

- `SomeKey = ftok(“./”, ‘x’);`

- Keys are global entities. If other processes know your key, they can access your shared memory.

- Ask the system to provide a private key using `IPC_PRIVATE`

Shared Memory: key generation (3/3)

- First step is to setup shared memory setup in kernel
- Key is generated using `ftok()` function

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok (const char *pathname, int proj_id);
```

- The *pathname* is an existing file in the filesystem
- The last eight bits of *proj_id* are used; these must not be zero
- A System V IPC key is returned

Asking for shared memory (1/4)

- Include the following:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

- Use `shmget()` to request a shared memory:

```
• shm_id = shmget(
key_t key,          /* identity key */
int size,           /* memory size */
int flag);          /* creation or use */
```

- `shmget()` returns a shared memory ID.
- The flag, for our purpose, is either `0666 (rw)` or `IPC_CREAT | 0666`. Yes, `IPC_CREAT`

Asking for shared memory (2/4)

- The following creates a shared memory of size

`struct Data` with a private key

`IPC_PRIVATE`. This is a creation

`(IPC_CREAT)` and permits read and write

`(0666)` .

```
struct Data { int a; double b; char x; };  
int          ShmID;
```

```
ShmID = shmget(  
    IPC_PRIVATE, /* private key */  
    sizeof(struct Data), /* size */  
    IPC_CREAT | 0666); /* cr & rw */
```

Asking for shared memory (3/4)

- The following creates a shared memory with a key based on the current directory

```
struct Data { int a; double b; char x;};  
int      ShmID;  
key_t    Key;
```

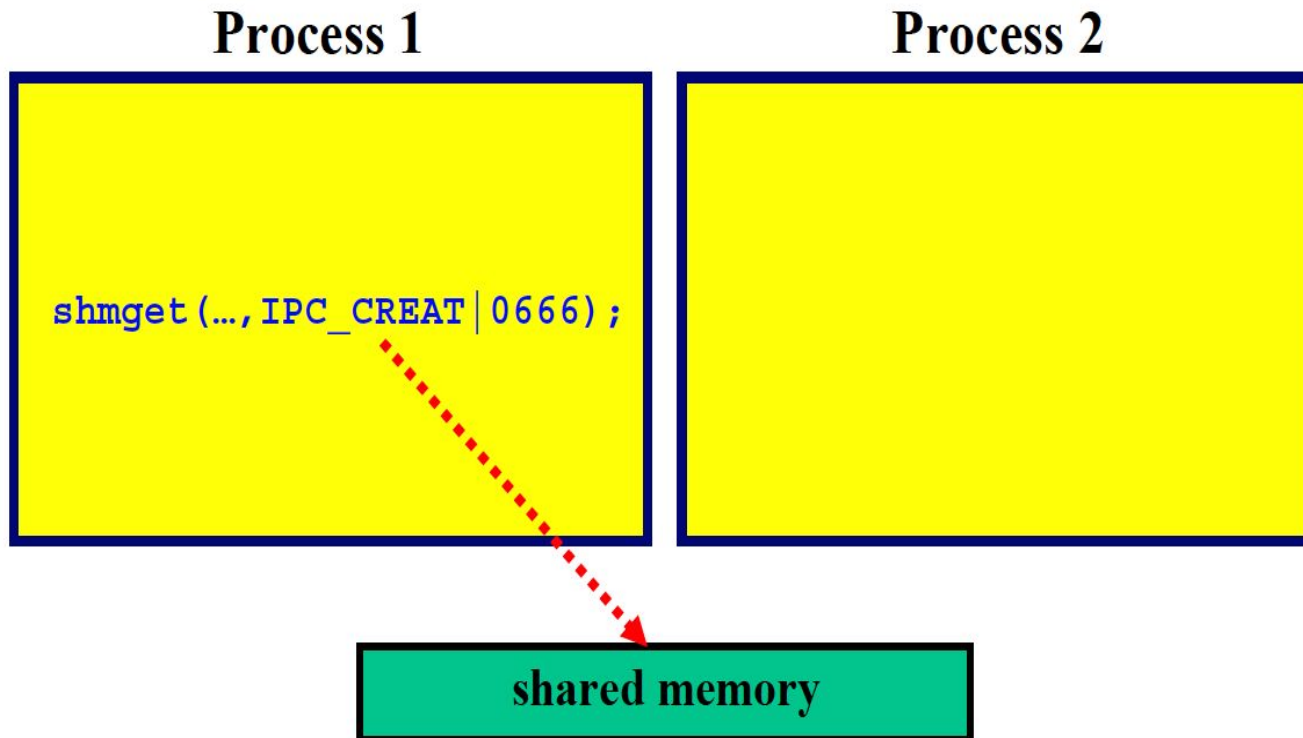
```
Key = ftok("./", 'h');  
ShmID = shmget(  
    Key,          /* a key */  
    sizeof(struct Data),  
    IPC_CREAT | 0666);
```

```
}
```

Asking for shared memory (4/4)

- The process that creates shared memory
 - It uses flag `IPC_CREAT | 0666`
- Other processes that accesses a created one shared segment using `0666`
- If the return value is negative (Unix convention), the request was unsuccessful, and no shared memory is allocated

After the Execution of `shmget()`



Shared memory is allocated; but, is not part of the address space

Attaching Shared Memory (1/2)

- Use `shmat()` to attach an existing shared memory to an address space:

```
shm_ptr = shmat(  
    int shm_id,          /* ID from shmget() */  
    char *ptr,          /* use NULL here */  
    int flag);          /* use 0 here */
```

- `shm_id` is the shared memory ID returned by `shmget()`
- Use `NULL` and `0` for the second and third arguments, respectively
- `shmat()` returns a void pointer to the memory.
- If unsuccessful, it returns a negative integer

Process 1

```
Shmget(..., IPC_CREAT | 0666);  
ptr = shmat(.....);
```

ptr

Process 2

```
ptr = shmat(.....);
```

ptr

shared memory

Now processes can access the shared memory

Detaching or removing the shared memory

- To detach a shared memory, use

```
shmdt(shm_ptr);
```

`shm_ptr` is the pointer returned by `shmat()`.

- After a shared memory is detached, it is still there. You can re-attach and use it again
- To remove a shared memory, use

```
shmctl(shm_ID, IPC_RMID, NULL);
```

where `shm_ID` is the shared memory ID returned by `shmget()`.

- After a shared memory is removed, it no longer exist

Garbage Collection Concept

Where does it happen in OS?

Do we need to do it for ourselves ?

Lecture Summary

- Shared memory provides a fast IPC mechanism