

CS310 Operating Systems

Lecture 15: Inter Process Communication – Message Passing

Ravi Mittal
IIT Goa

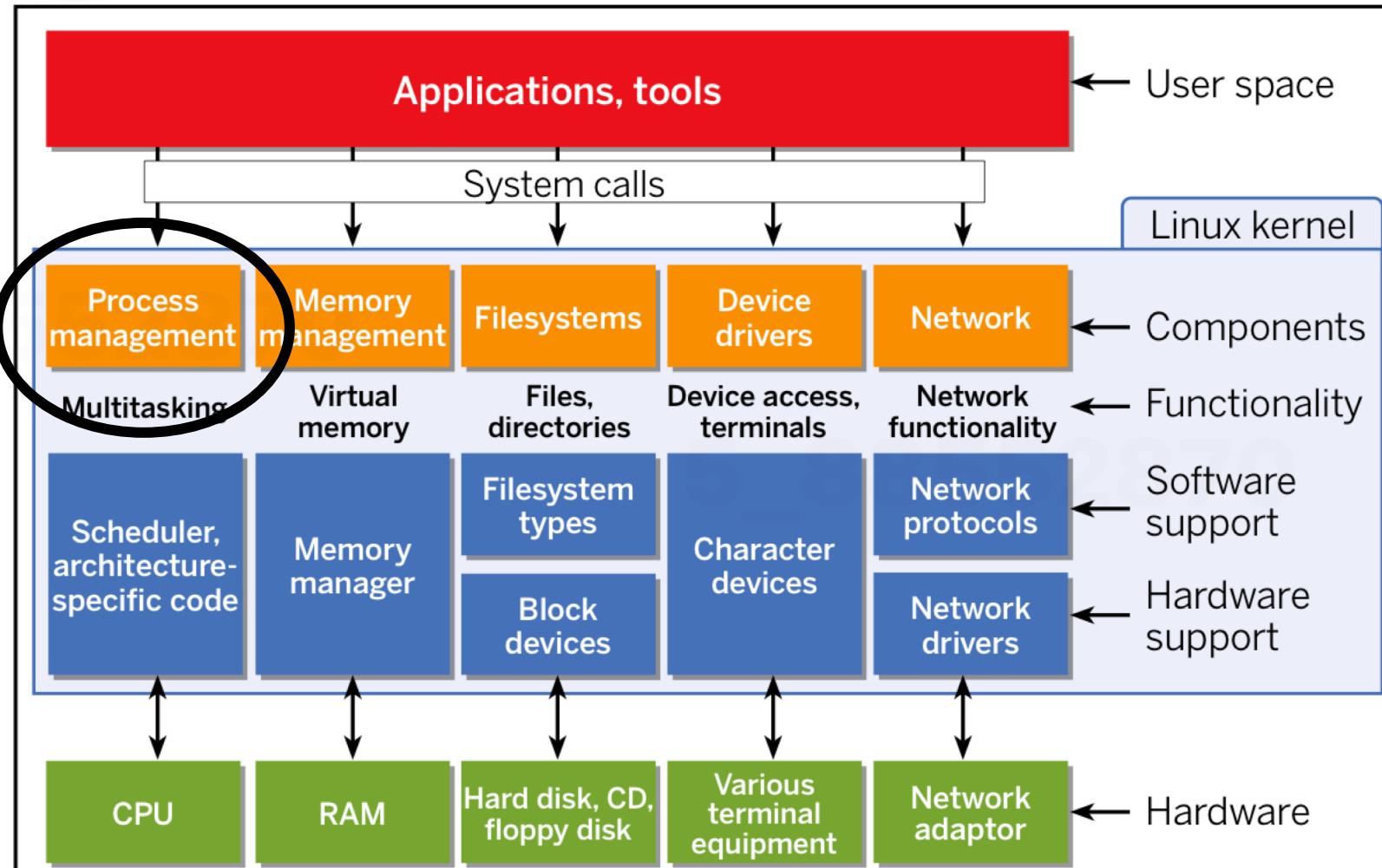
Reading

- Book: Operating Systems: Internals and Design Principles, William Stallings, Person Publishing
- Book: Operating System Concepts, 10th Edition, by Silberschatz, Galvin, and Gagne
 - Chapter 3.6
- Book: Modern Operating Systems, Andrew Tenenbaum, and Herbert Bos, 4th Edition, Pearson
 - Chapter 5.6

In today's class, we will study

- Concurrent Execution of Processes – Issues
- Inter Process Communication (IPC) – Introduction
- Message Passing Concept
 - Synchronous Vs Asynchronous
 - Blocking Vs Non-blocking
 - Direct vs Indirect Communication
 - Mailboxes and Ports
 - Message Format

Where are we ?



Concurrent Execution of Processes - Issues

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
 - However, OS has to share the system resources among these processes
 - Processes compete for system resources such as disk, file or printer.
- Cooperating processes
 - Results of one or more processes may be needed for another process, for
 - Information sharing
 - Computation Speedup
 - Modularity
 - Convenience

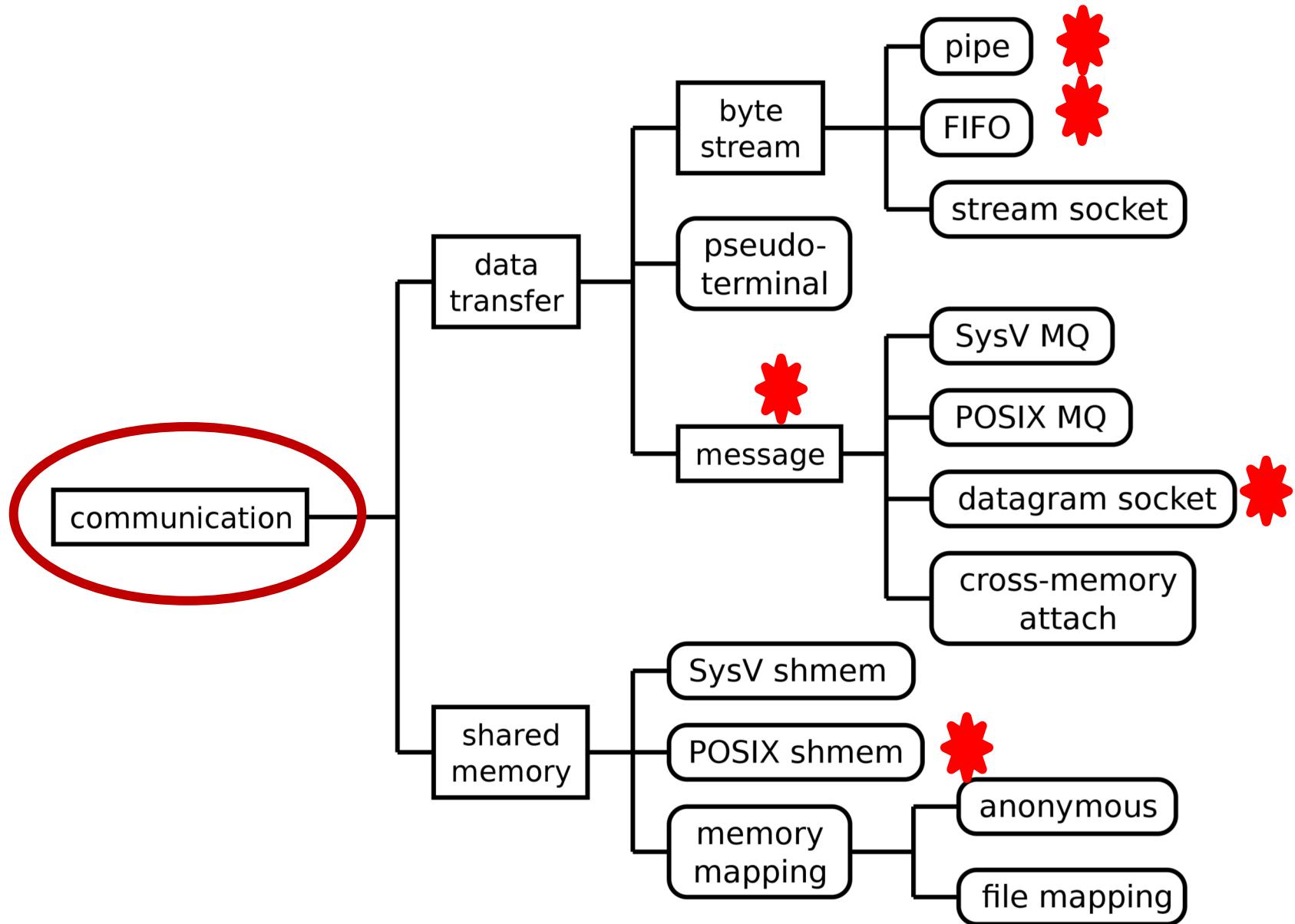
Process Interaction - Issues

1. What way one process can **pass information** to another process ?
 2. Two processes don't get into each other's way
 - Two competing processes in airways reservation system trying to grab the last seat
 3. Need for proper **sequencing** when there is a dependency
 - If process A produces data that B prints, process B must wait until A has produced data
- All of the above issues are also applicable to threads
 - Though information passing is easy as threads share the same address space

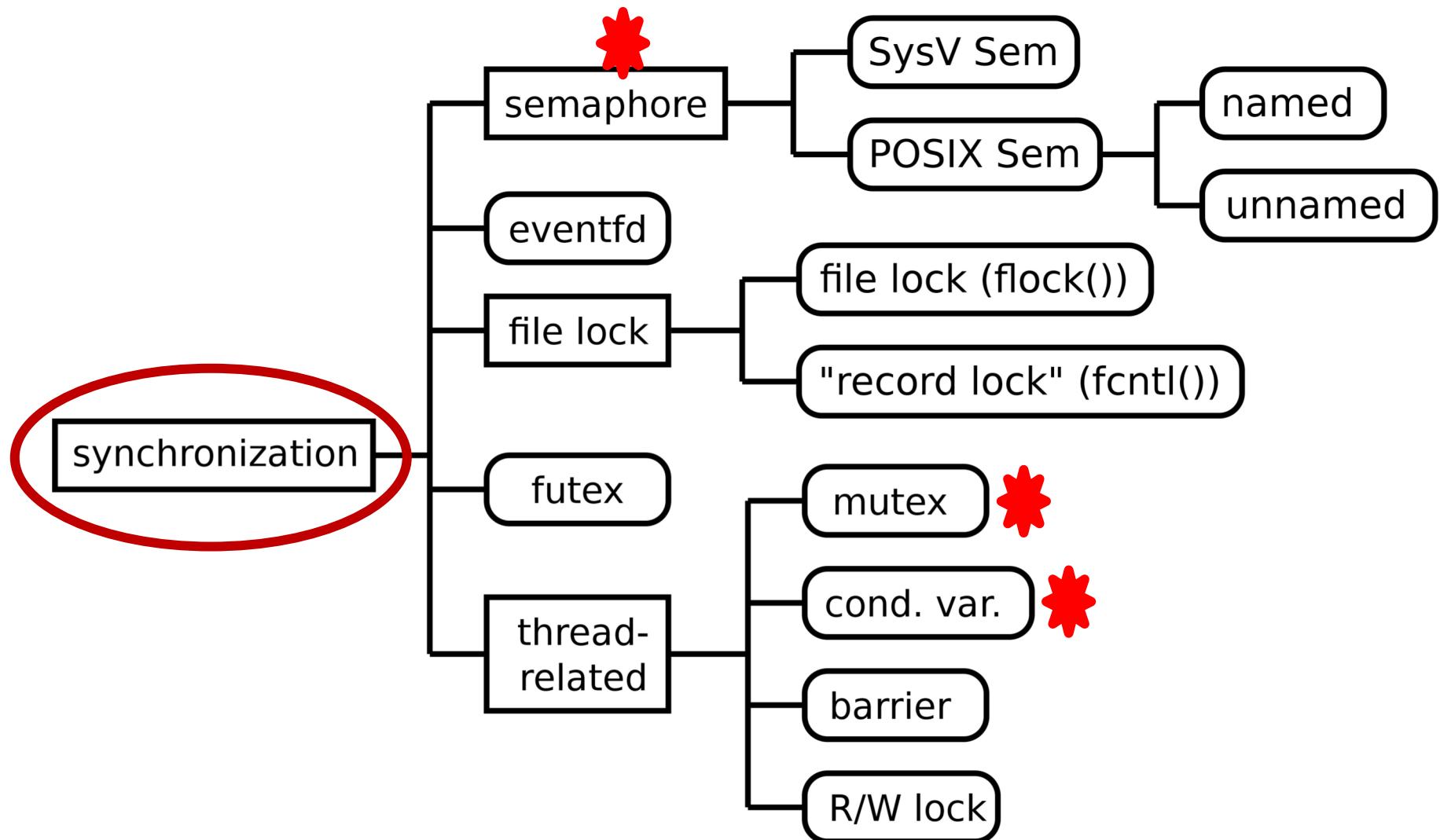
Requirements of Processes/Threads - Interaction

- Communication
 - Cooperating processes may need to exchange information
- Synchronization
 - Processes may need to be synchronized to enforce mutual exclusion

Communication (in Linux)



Synchronization (in Linux)

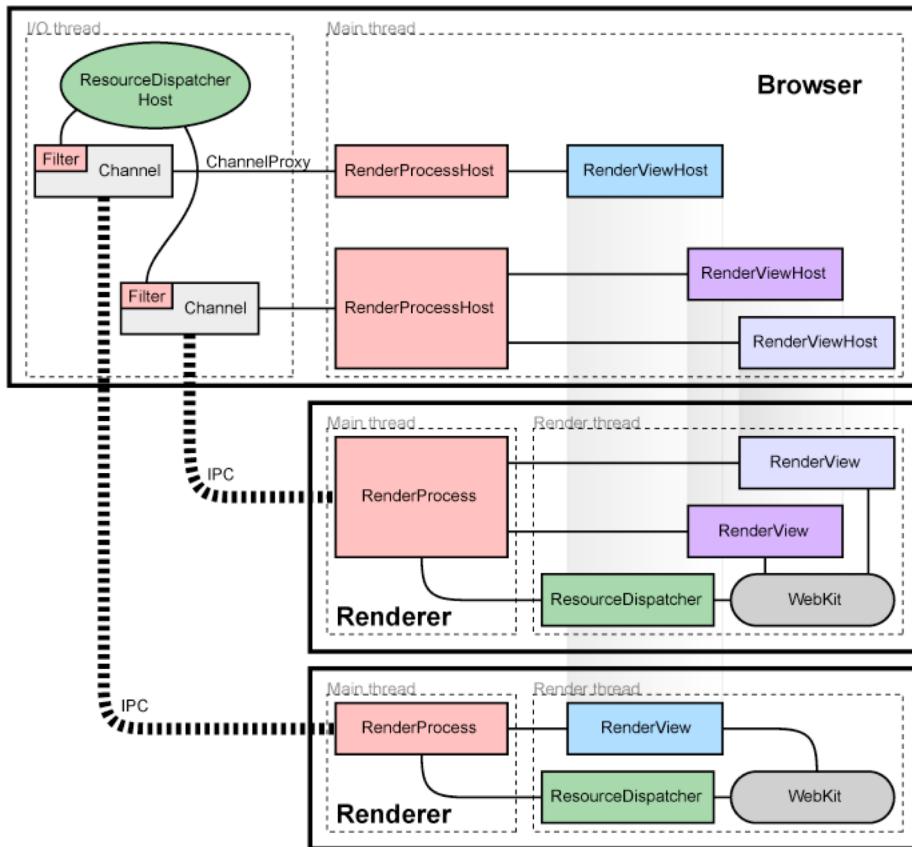


Inter-Process Communication (IPC) - Introduction

IPC Requirement

- Any use case of IPC that comes to your mind?
- Any big application?

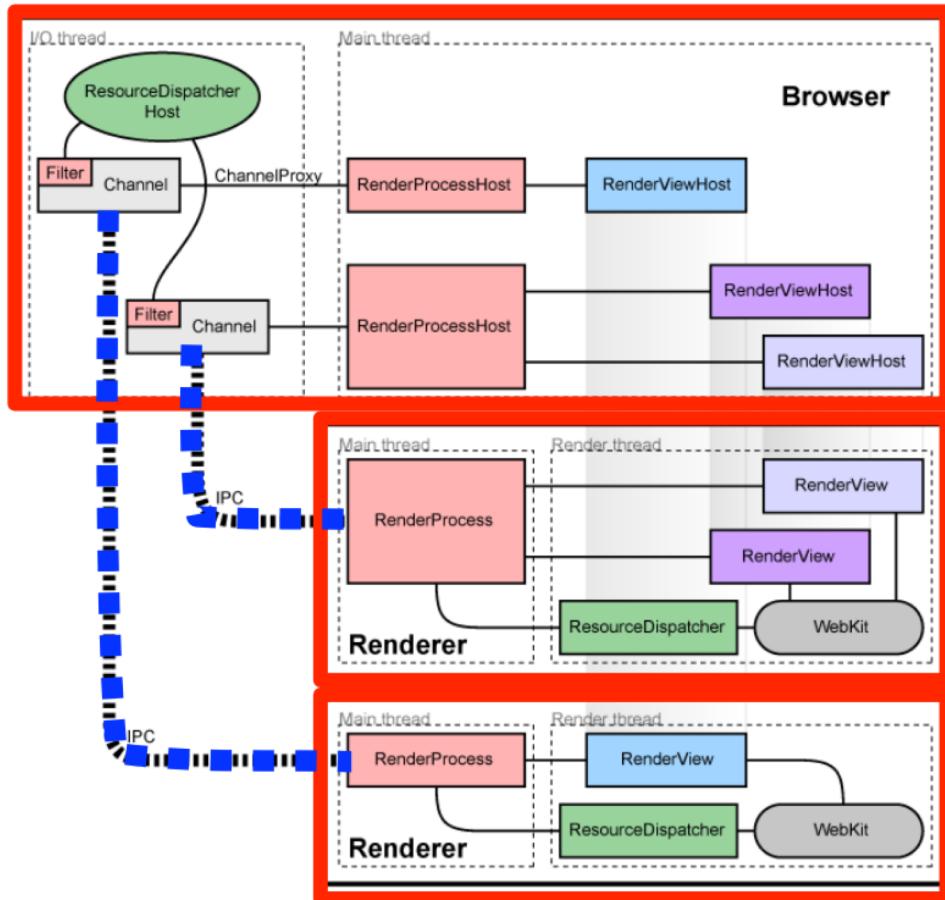
Google Chrome Architecture



- Separate processes for browser tabs to protect the overall application from bugs and glitches in the rendering engine
- Restricted access from each rendering engine process to others and to the rest of the system

<https://www.chromium.org/developers/design-documents/multi-process-architecture>

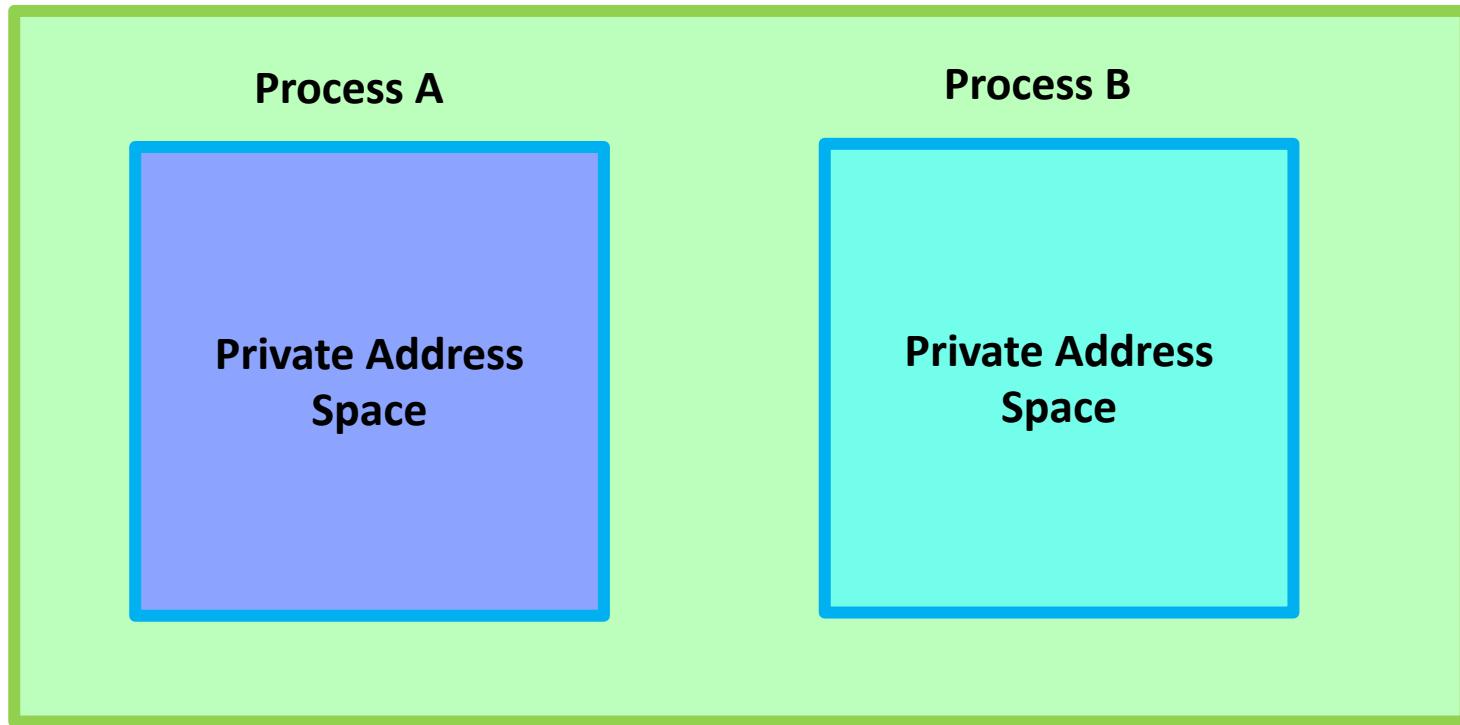
Google Chrome Architecture



- A named pipe is allocated for each renderer process for communication with the browser process
- Pipes are used in asynchronous mode to ensure that neither end is **blocked waiting** for the other

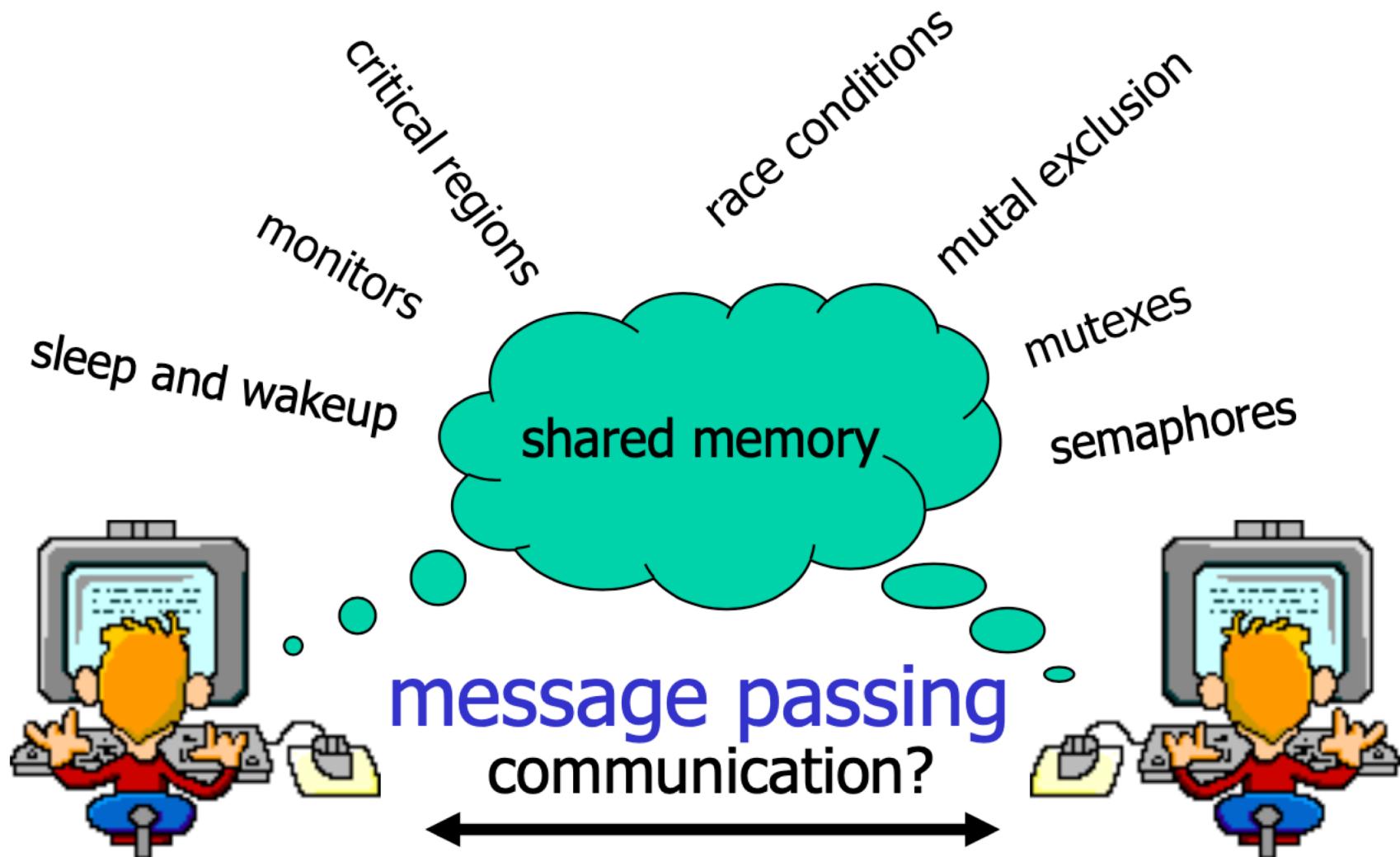
<https://www.chromium.org/developers/design-documents/multi-process-architecture>

IPC Communication Model

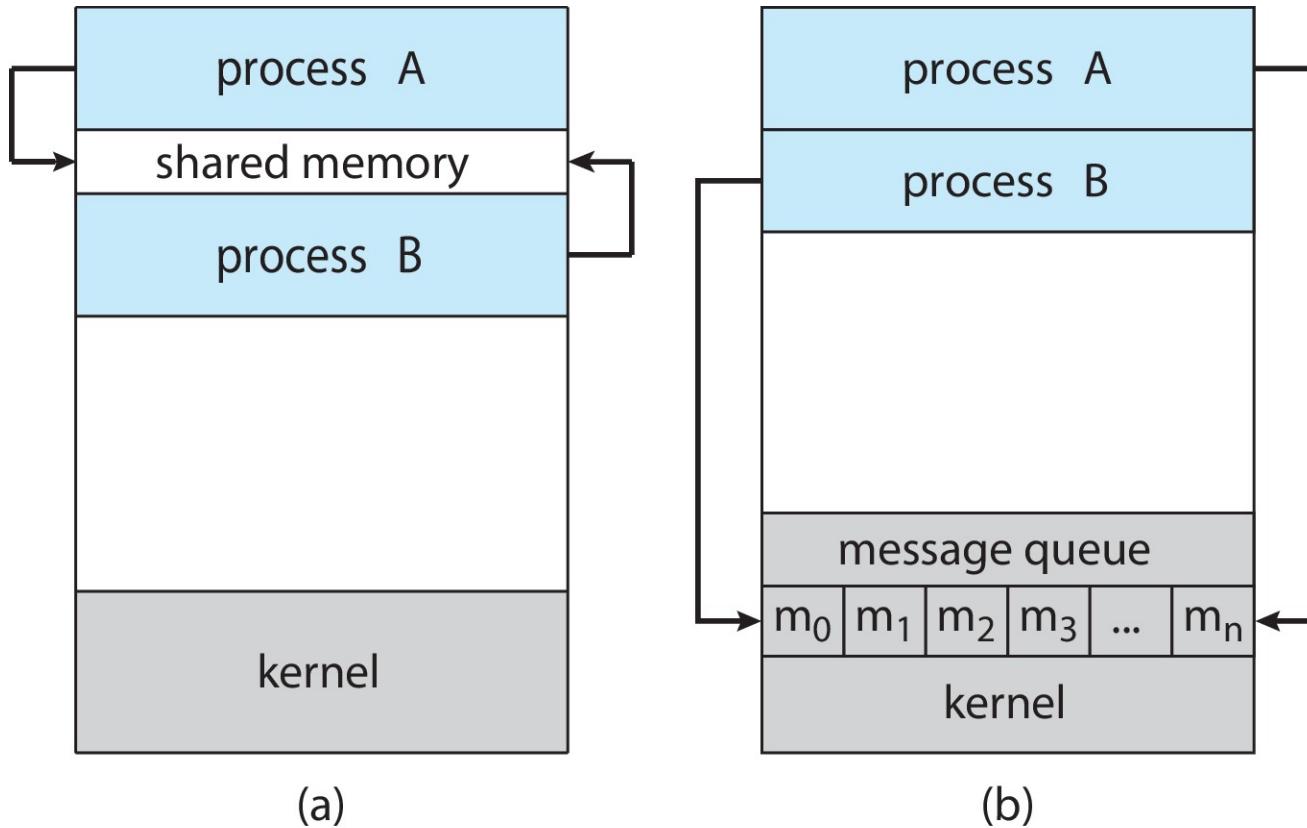


- Each process has it's own address space
- No process can write/read from other process's address space

IPC – Big Picture



Models of IPC: Shared Memory, Message Passing



Shared memory

Message passing.

Message Passing - Concepts

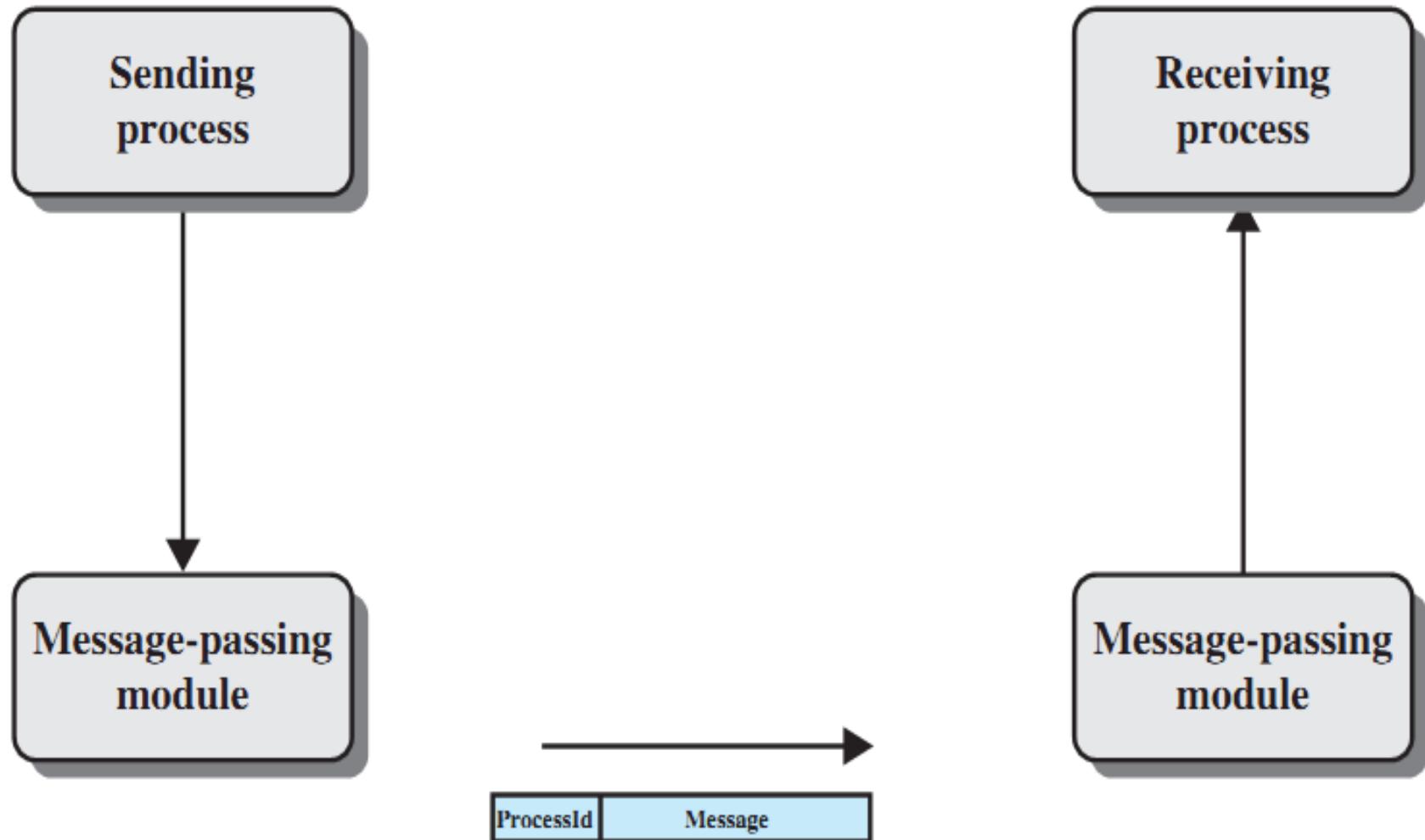
IPC: Message-Passing Systems

- No sharing of address space
- Very useful in distributed systems where the communicating processes reside on different computers connected by a network
- There are many forms of message Passing systems
 - Our focus is only in general concepts
- IPC facility provides two operations:
 - `send(destination, message)`
 - `receive(source, message)`
- A process sends information to another process (destination) in the form of message – by executing `send` primitive
- A process receives information by executing the `receive` primitive, indicating the source and the message

IPC: Message-Passing Systems: Issues

- Synchronous vs Asynchronous
- Direct vs Indirect
- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or b-directional?

Basic Message Passing Primitives



Message Passing – Local or distributed

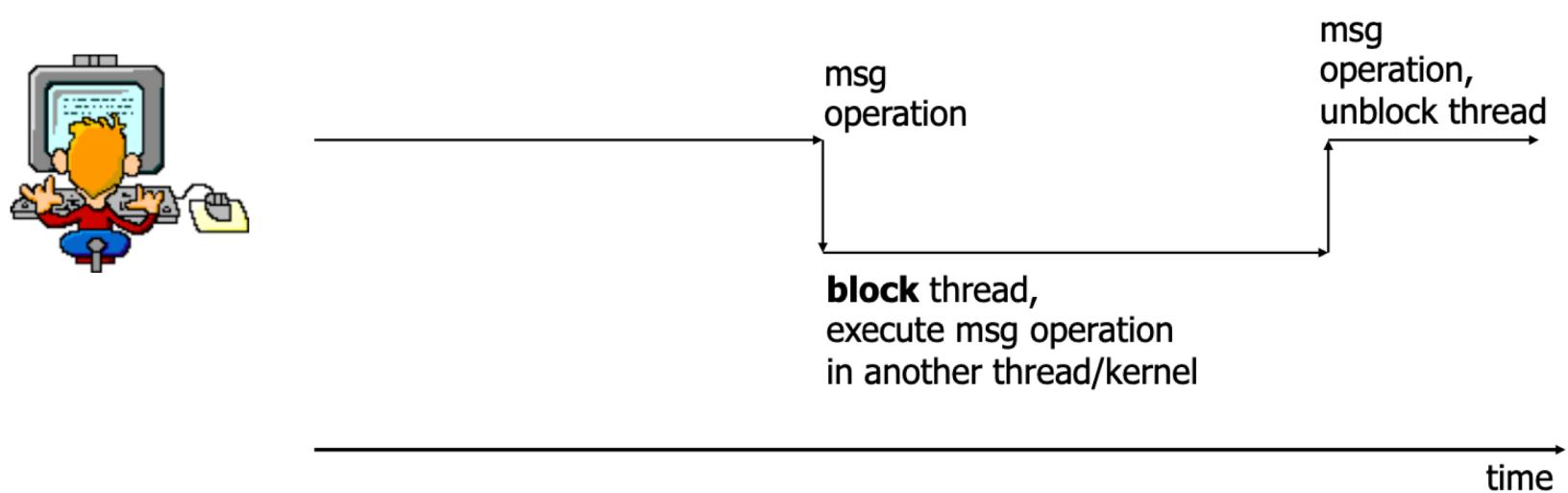
- In a Message passing system
 - Processes may inside the same computer
 - Processes may reside in a networked/distributed system
- We will later study a popular method of distributed communication
 - Remote Procedure Call (RPC)

Synchronization

- The communication of a message between two processes implies some level of synchronization between the two
- What happens to a process after it issues a send or receive primitive?
- When a send primitive is executed in a process: two possibilities
 - Sending process is blocked until the message is received
 - Sending process is Not blocked until the message is received

Blocking send and receive

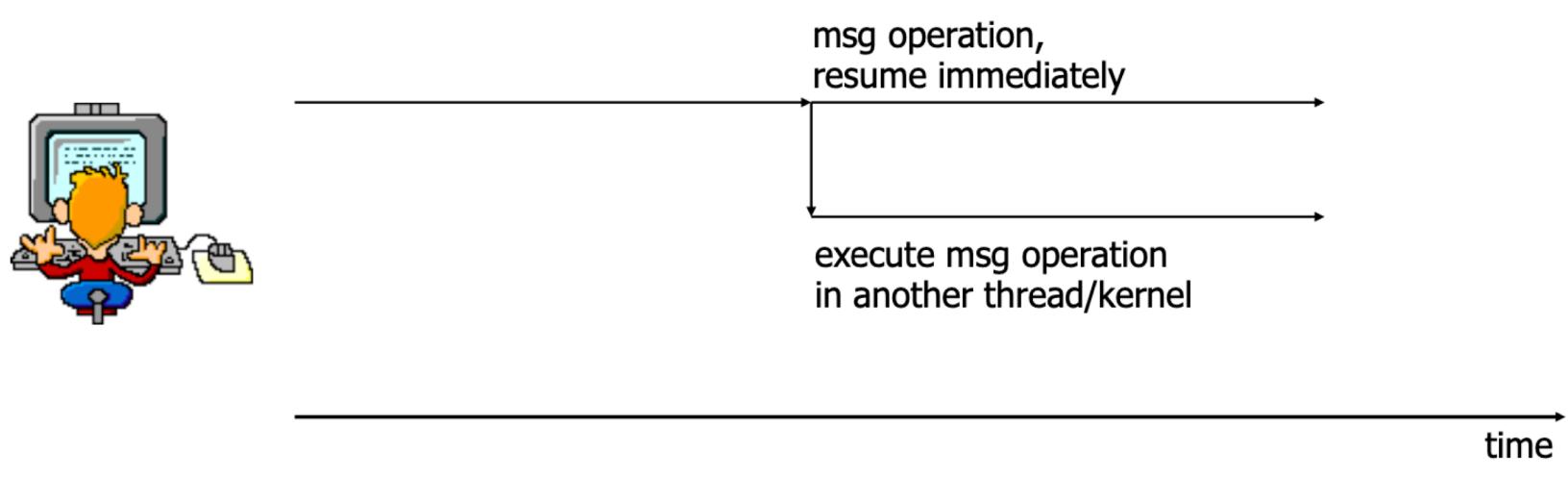
- **Blocking send** has the sender block until the message is received
- **Blocking receive** has the receiver block until a message is available



- Process/Thread is blocked until message primitive has been performed
- May be blocked for a very long time

Non-blocking send and receive

- **Non-blocking send** has the sender send the message and continue
- **Non-blocking receive** has the receiver retrieves a valid message or null



- Process /Thread gets control back immediately
- Process/Thread cannot reuse buffer for message before message is received

Blocking Vs Non-blocking Message Passing

- **For the sender:** it is more natural **not to be blocked** after issuing send:
 - can send several messages to multiple destinations
 - but sender usually expect acknowledgment of message receipt (in case receiver fails)
- **For the receiver:** it is more natural **to be blocked** after issuing receive:
 - the receiver usually needs the information before proceeding
 - but could be blocked indefinitely if sender process fails before send.

Blocking Vs Non-blocking Message Passing

There are really 3 combinations here that make sense:

1. Blocking send, Blocking receive
2. Nonblocking send, Nonblocking receive
3. Nonblocking send, Blocking receive
 - Most popular
 - Example: Server process that provides services/resources to other processes. It will need the expected information before proceeding.

Link Capacity – Buffering

- Links are associated with **queues** at both ends of the communication link.
- Queues are implemented with
 - **Zero capacity – 0 messages**
 - Sender must wait for receiver
 - **Bounded capacity** – finite length of **n** messages. At most n messages can reside in it
 - Sender must wait if link full
 - **Unbounded capacity** – infinite length (very large capacity)
 - Sender never waits

IPC Requirements

- If P and Q wish to communicate, they need to:
 - Establish communication link between them
 - Exchange messages via **send/receive**

Direct and Indirect Communication

- Direct communication (Direct Addressing):
 - When a specific process identifier is used in send primitive for source/destination i.e. the names of both the sender and the receiver are known
 - In many scenarios it is not possible to know source ahead of time. For example: connection to a print server

Indirect communication (Indirect Addressing):

- Messages are sent to or received from a shared mailbox or ports
 - Shared data structure - queues
- A mailbox can be viewed as an object into which messages can be placed by processes and from which messages can be removed
- Two processes can communicate only if they have a shared mailbox

Direct Communication



- The process that wants to communicate must explicitly name the recipient or sender of the communication
- The send() and receive() primitives are defined as:
 - `send(P, message)` : send a message to process P
 - `receive(Q, message)` : receive a message from Q
- Properties of communication link:
 - The processes need to know only each other's identity to communicate
 - A link is established automatically between every pair of processes that want to communicate
 - A link is associated with exactly one pair of communicating processes
 - Between each pair there exists exactly one link

Indirect Communication



- Messages are directed to and received from **mailboxes** (also referred to as ports)
 - Each mailbox has a **unique id**
 - Example: POSIX message queues use an integer value to identify a mailbox
 - Processes can communicate only if they share a **mailbox**
- A process can communicate with another process via a number of different mailboxes
- Two processes can communicate only through a shared mailbox

Indirect Communication

- Primitives:

`send(A, message)` : send a message to mailbox A

`receive(A, message)` : receive a message from mailbox A

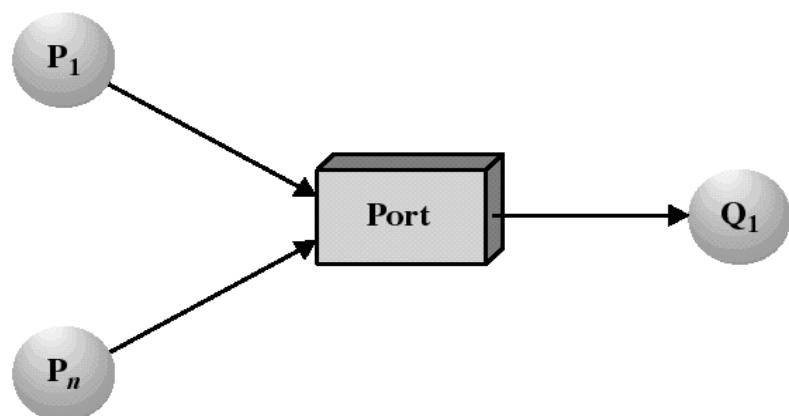
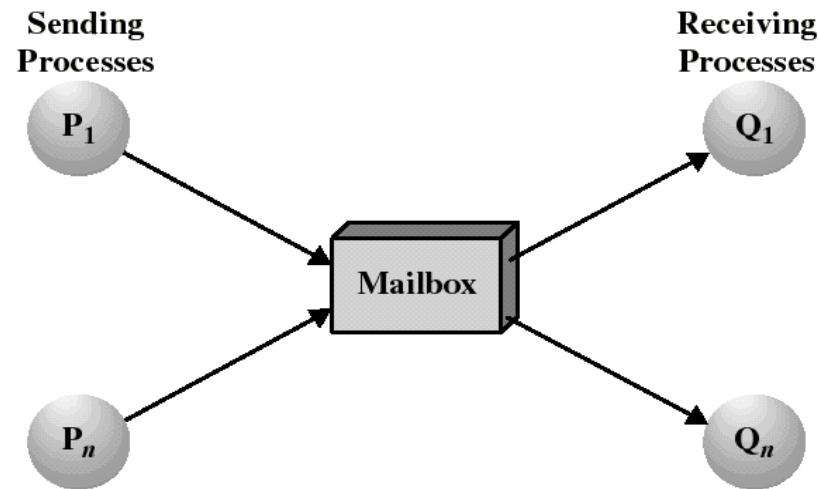
- Link established only if both processes share a common mailbox
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional
- Operations
 - Create a new mailbox (port)
 - Send and receive messages through mailbox
 - Destroy a mailbox

Indirect Communication

- Consider that processes P1, P2, and P3 share mailbox A
- Process P1 sends a message to A, while both P2 and P3 execute a `receive()` from A
- Who gets the message? The answer depends on which of the following methods are used?
 - Allow a link to be associated with at most two processes
 - Allow only one process at a time to execute a `receive()` operation
 - Allow the system to select arbitrarily which process will receive the message
 - Example: round robin – where processes take turns to receive messages
 - The system must notify to the sender about the receiver

Mailboxes and Ports

- A mailbox can be private to one sender/receiver pair
- A mailbox can also be shared among several senders and receivers (many – to – many)
- Port: is a mailbox associated with one receiver and multiple senders
 - used for client/server applications: the receiver is the server
 - A port is usually own and created by the receiving process
 - The port is destroyed when the receiver terminates

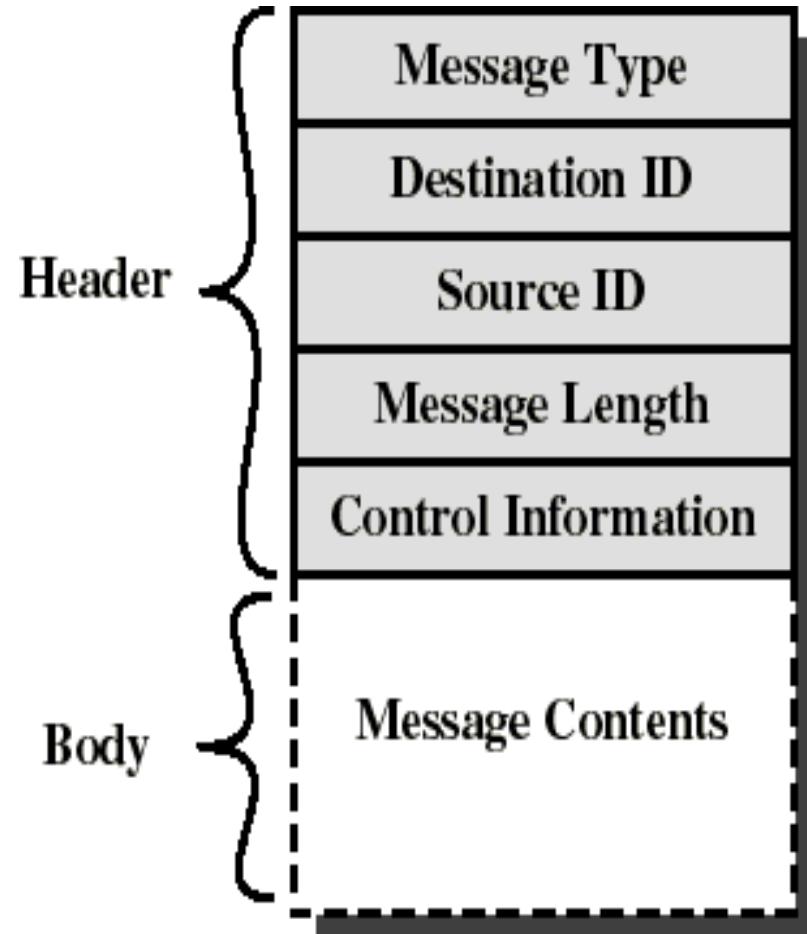


Mailboxes

- Who owns and creates mailbox ?
- Ex: If you have a mailbox in front gate of your home, it is surely owned and managed by you
- Mostly mailboxes are owned and created by the receiving process
 - Generally with the help of the OS
- Alternatively, mailboxes can be created and owned by the OS
- When a process is destroyed, the associated port is also destroyed

Message format

- Consists of header and body of message
- In Unix: no ID, only message type
- Control info:
 - what to do if run out of buffer space
 - sequence numbers
 - priority
- Queuing discipline: usually FIFO but can also include priorities.



Lecture Summary

- Two processes may communicate by exchanging messages with one another using message passing
- In the lecture we have looked in to various aspects of message passing mechanism
 - Synchronous and Asynchronous sending and receiving
 - Buffering – Zero capacity, Bounded Capacity, Unbounded Capacity
 - Direct and Indirect Communication
 - Mail boxes and Ports

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