Lecture 4.b

Client Server Communication

Dr. Gabriele Pierantoni

10.02.2021

Client/Server Communication

Some Basic Concepts - Program

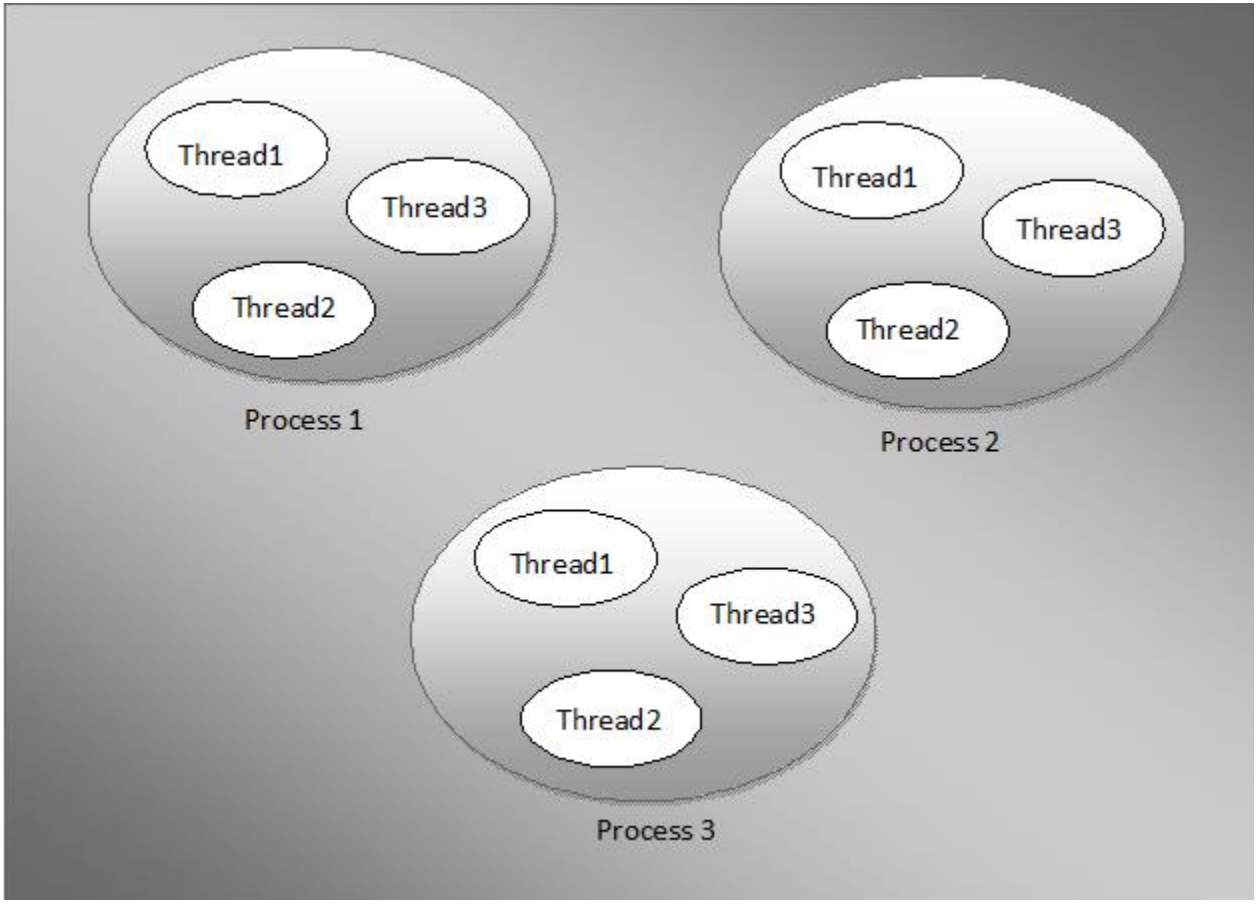
- A computer **program** is a collection of instructions that performs a specific task when executed by a computer.

- A **process** is an instance of a computer **program** that is being executed. It contains the **program** code and its current activity. Depending on the operating system (OS), a **process** may be made up of multiple threads of execution that execute instructions concurrently

- A process can have multiple threads. Each thread will have their own task and own path of execution in a process. For example, in a notepad program, one thread will be taking user inputs and another thread will be printing a document.

- All threads of the same process share memory of that process. As threads of the same process share the same memory, communication between the threads is fast.

Some Basic Concepts – Program, Processes and Threads



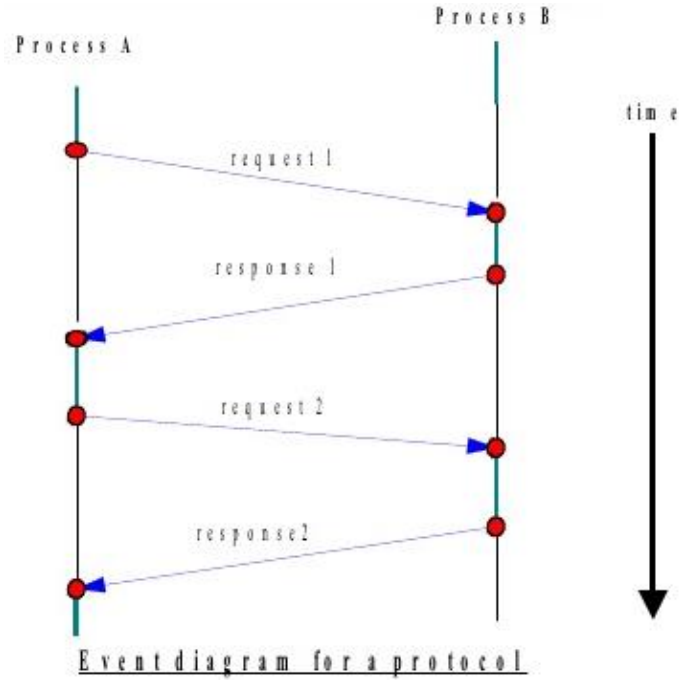
Inter-Process Communication (IPC)

Definition:

- **IPC = a set of methods for the exchange of data among multiple processes or computers**
- **IPC provides mechanism for processes to communicate and to synchronize their actions**

IPC types:

- **Message Passing Communication**
- **Request/Reply Communication**
- **Transaction Communication**

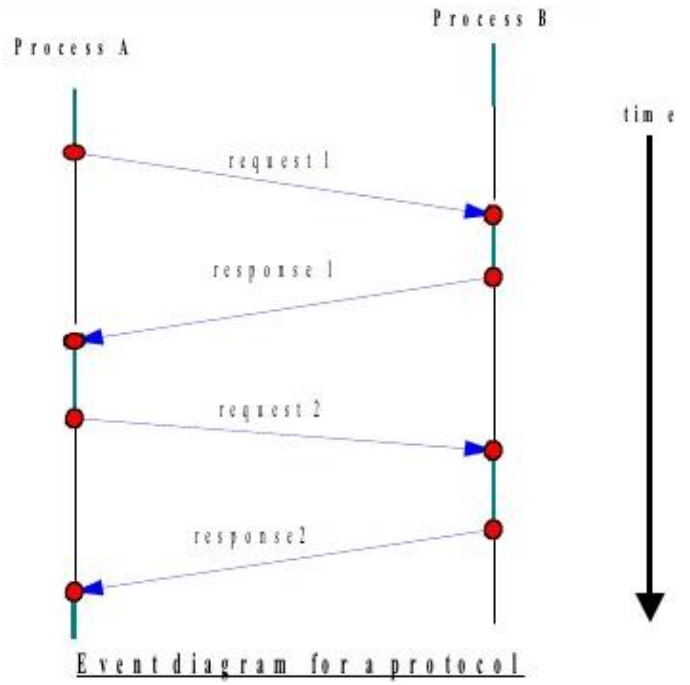


Event diagram for a protocol

Inter-Process Communication (IPC)

IPC types:

Request/Reply Communication: one of the basic methods computers use to communicate with each other, in which the first computer sends a request for some data and the second computer responds to the request. Browsing a web page is an example of request–response communication.



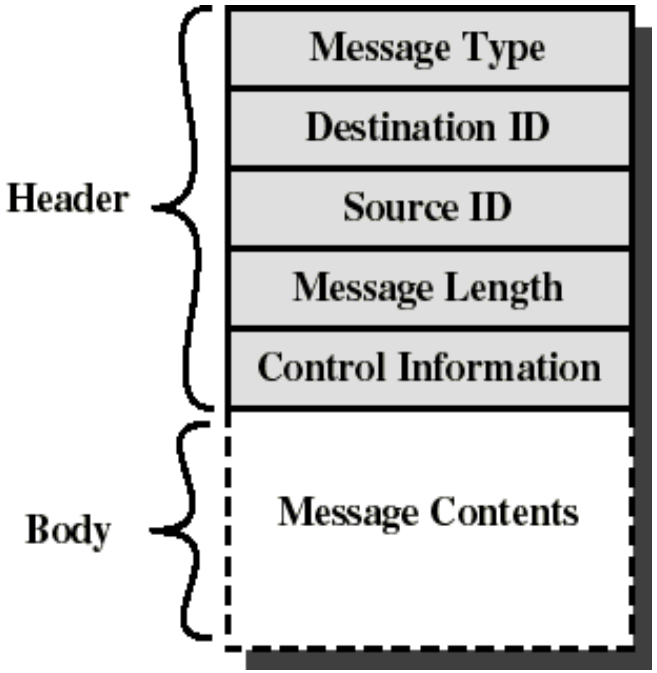
Event diagram for a protocol

Inter-Process Communication: Message Passing (1)

Message

- **consists of header and body**
- **<u>header:</u>**

  control information
  - **what to do if run out of buffer space.**
  - **sequence numbers.**
  - **priority**
- **<u>body:</u>**
  - **message contents**



Message passing

- **Processes communicate with each other using fixed or variable sized messages without using shared variables**
- **It represents the lowest level of inter-process communication between processes either inside the same computer or in distributed systems**

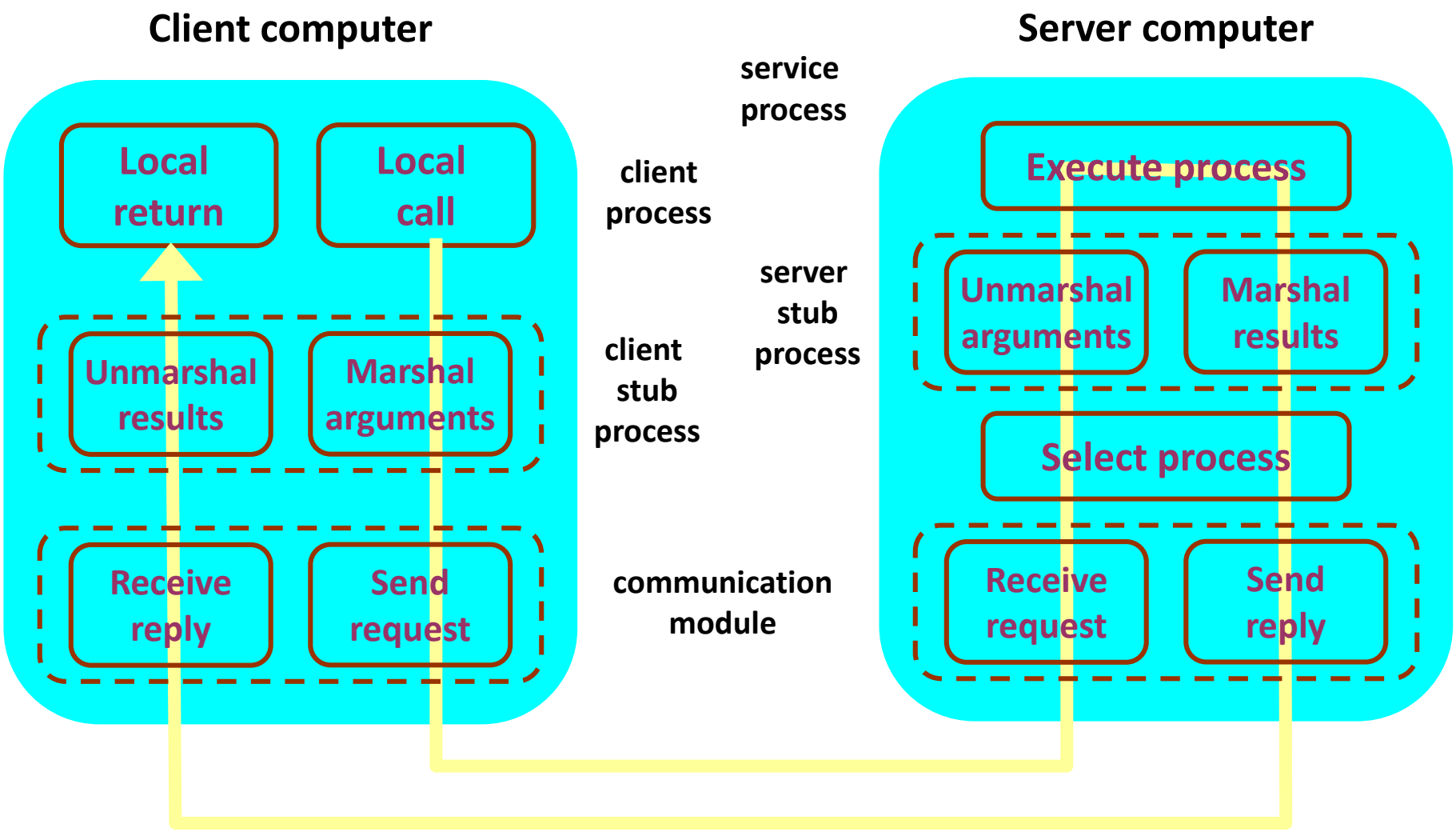
Inter-Process Communication: RPC

Remote Procedure Call (RPC)

In distributed computing, a **remote procedure call (RPC)** is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. This is a form of client–server interaction (caller is client, executor is server), typically implemented via a request–response message-passing system.

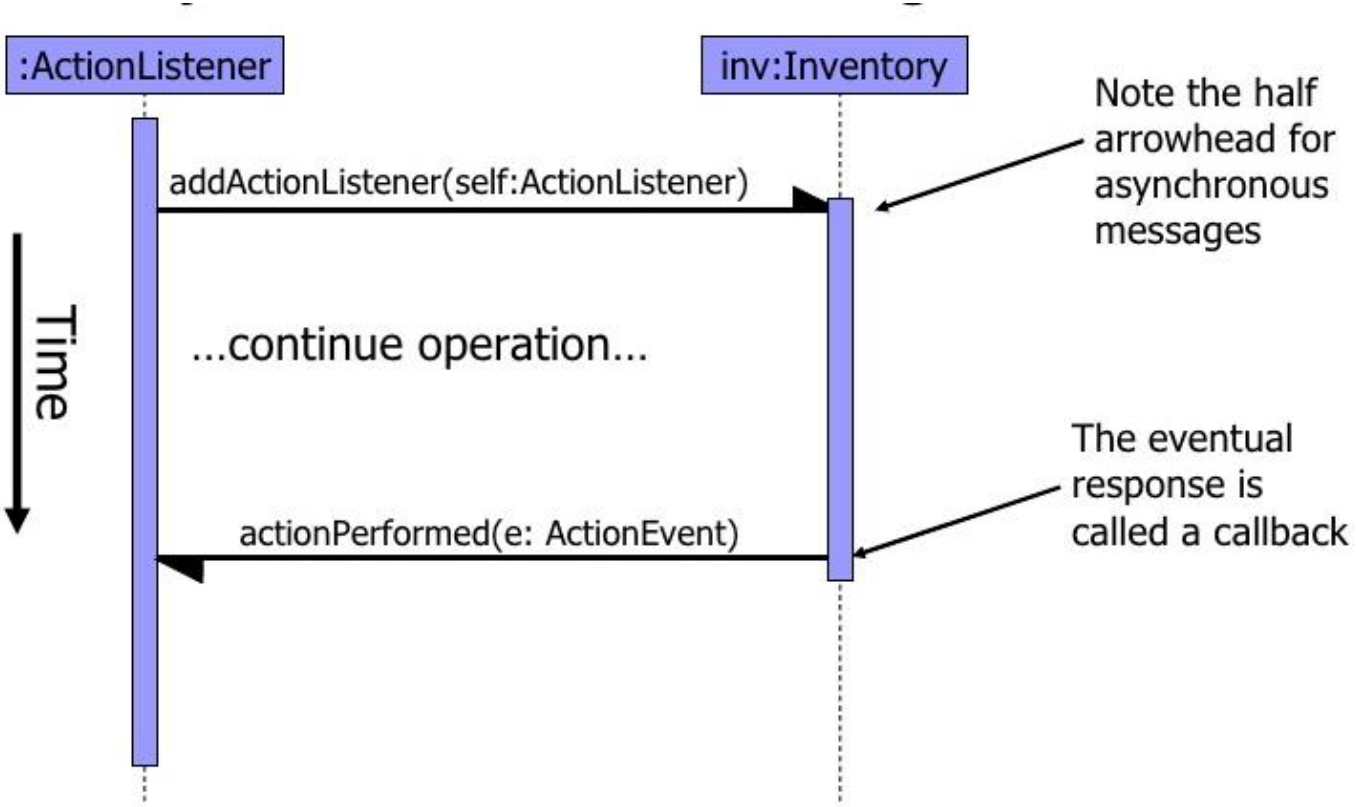
Inter-Process Communication: Request - Reply (2)



Message Passing (2)

asynchronous communication (or non-blocking)

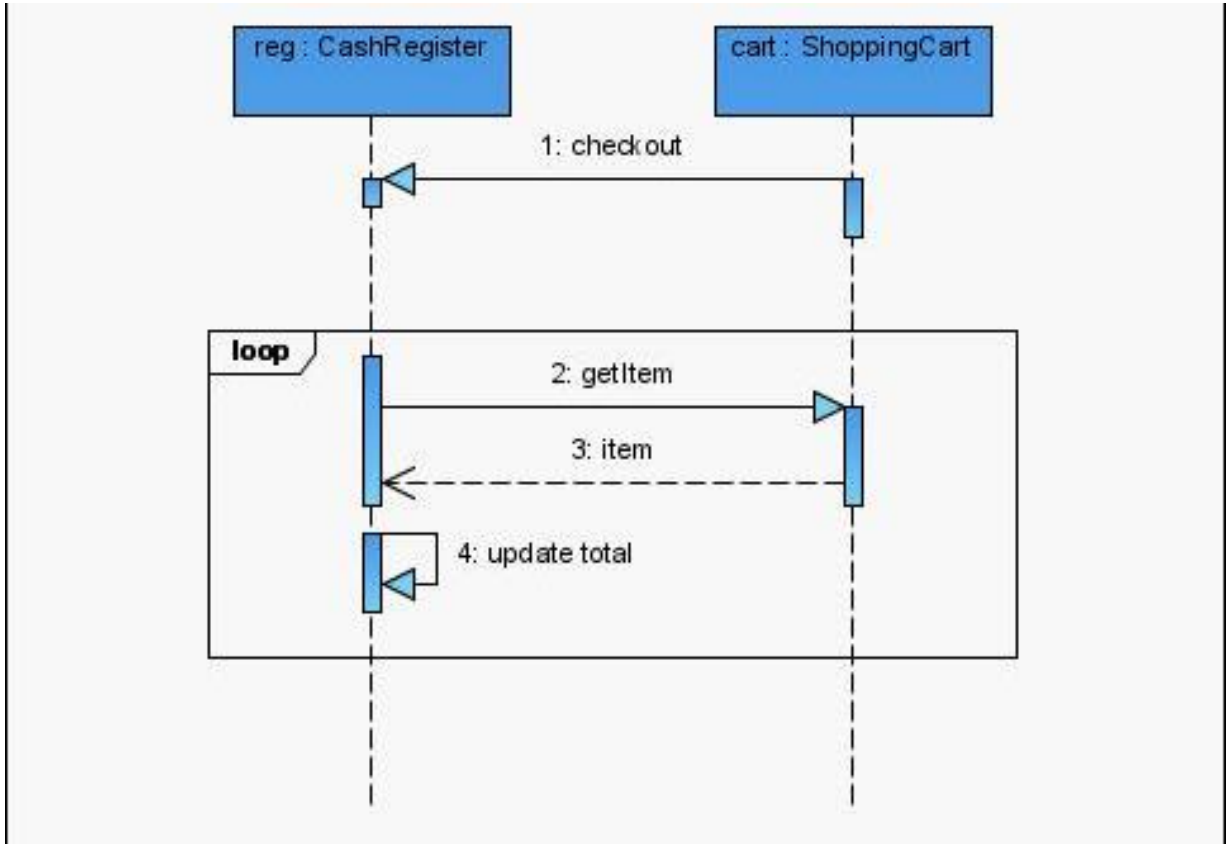
- **non-blocking send - sender sends the message and continues**
- **non-blocking receive - receiver receives a valid message or null**



Message Passing (3)

synchronous communication (or blocking)

- **blocking send - sender is blocked until the message is received**
- **blocking receive - receiver is blocked until a message is available**



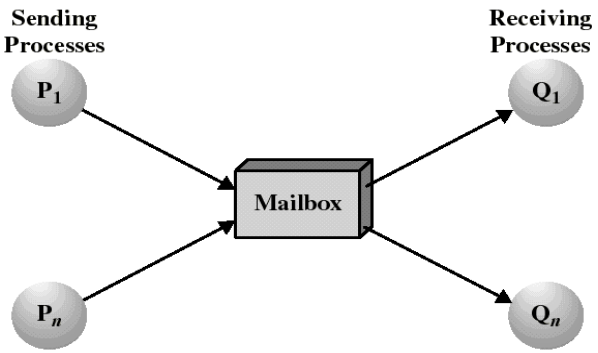
Message Passing (4)

Direct communication

- When a specific process identifier is used for source/destination
- Processes must name each other explicitly:

  send(P, message) – send a message to *process P*

  receive(Q, message) – receive a message *from processQ*

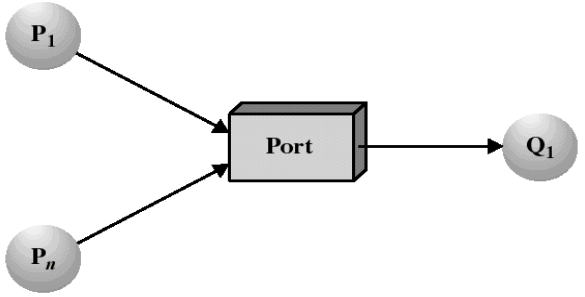


Indirect communication

- Senders put messages in a mailbox or a port, receivers pick them up

  send(A, message) – send a message to *mailbox A*

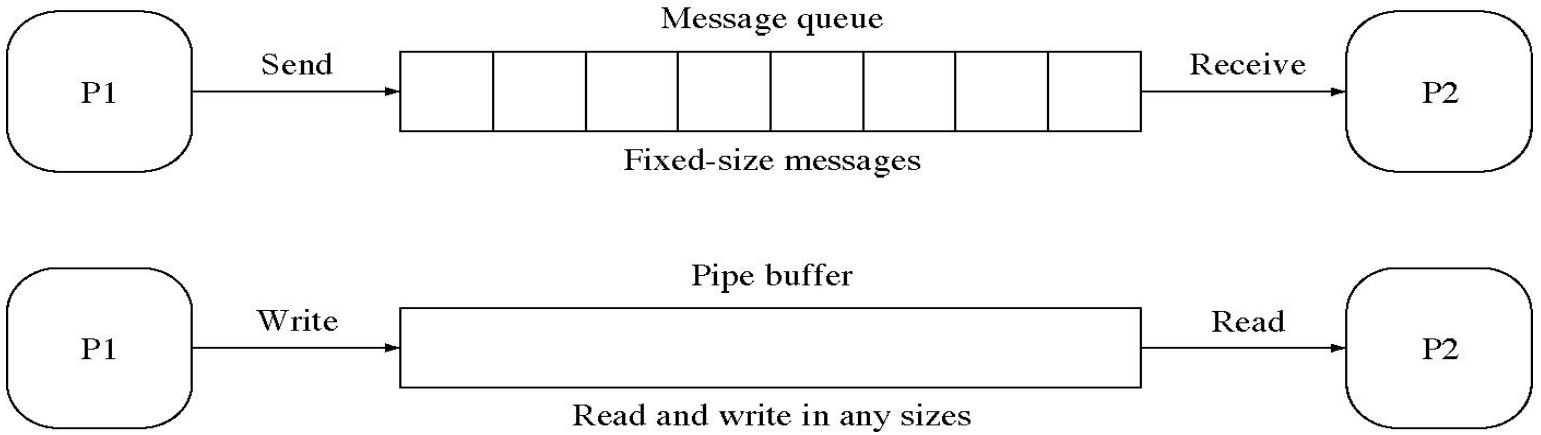
  receive(A, message) – receive a message from *mailbox A*



Message Passing (4)

Pipe

- **unidirectional communication**
- **implemented with finite size, FIFO byte stream buffer maintained by the kernel**
- **one process writes data into tail end of pipe while another process reads from head end of the pipe**



Socket

- **communication end point of a communication link managed by the transport services**
- **most popular message passing API**