Distributed Systems
“Inter-Process Communication (IPC)”

Mathieu Delalandre
University of Tours, Tours city, France
mathieu.delalandre@univ-tours.fr

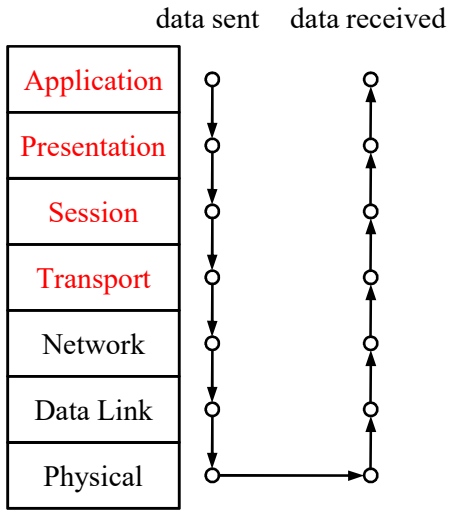
Lecture available at http://mathieu.delalandre.free.fr/teachings/dsystems.html

Inter-Process Communication in Distributed Systems

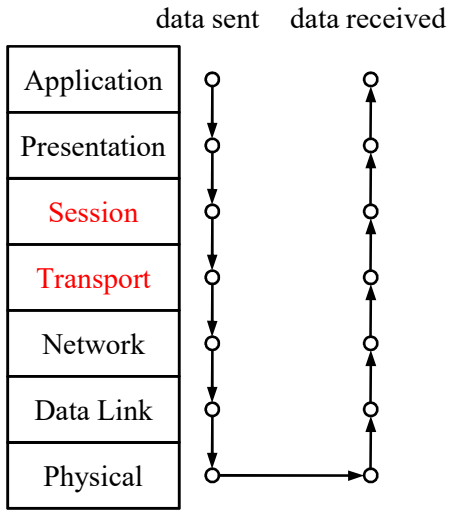
1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Introduction (1)

Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

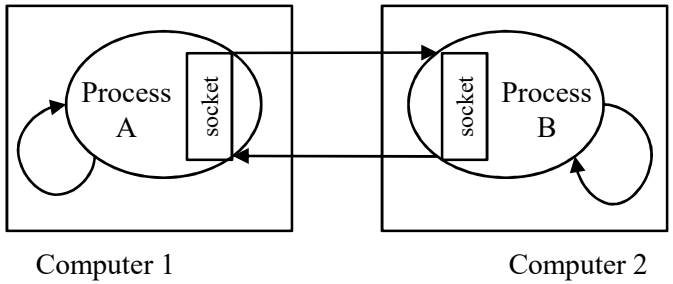


Introduction (2)



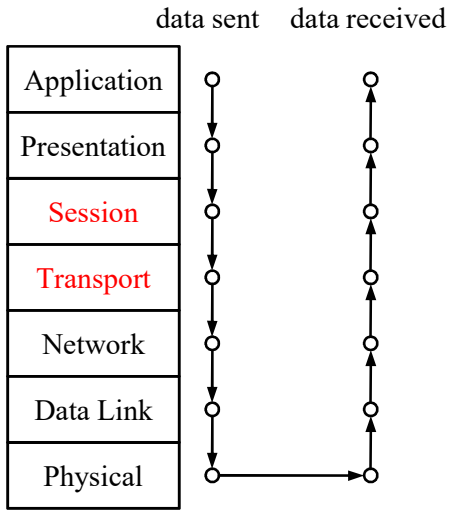
Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

Communication: we must fix the way to coordinate and to manage the communication between processes.



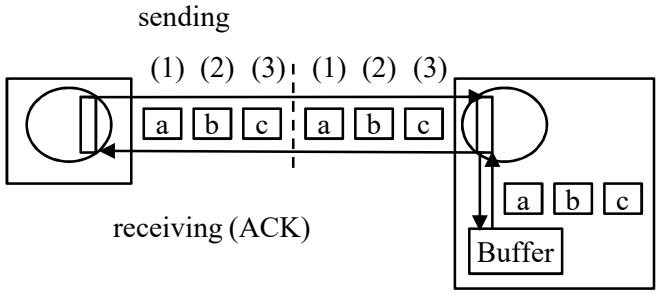
Socket: is a communication end point to which an application can write/read data over the underlying network.
It constitutes a standard transport interface for delivering incoming data packets between processes.

Introduction (3)

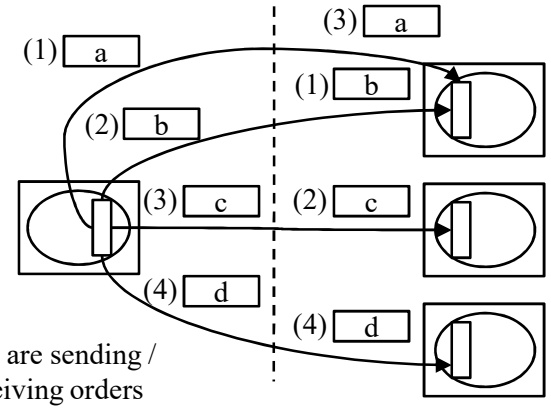


Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

Communication: we must fix the way to coordinate and to manage the communication between processes.

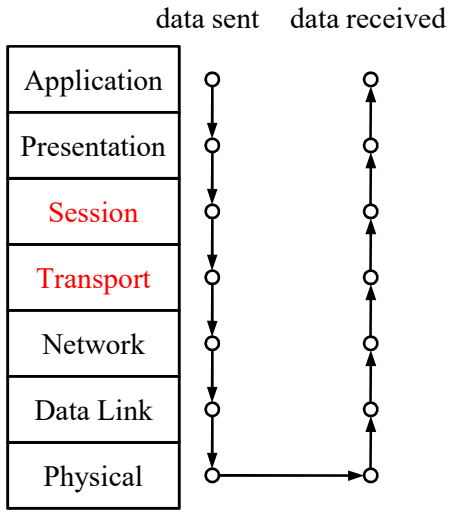


Stream-oriented communication (e.g. TCP) is a data communication mode in which the devices at the end points use a protocol to establish an end-to-end logical or physical connection before any data that may be sent.



Message-oriented communication (e.g. UDP) is a data transmission method in which each data packet carries information in a header that contains a destination address sufficient to permit the independent delivery of the packet to its destination via the network.

Introduction (4)

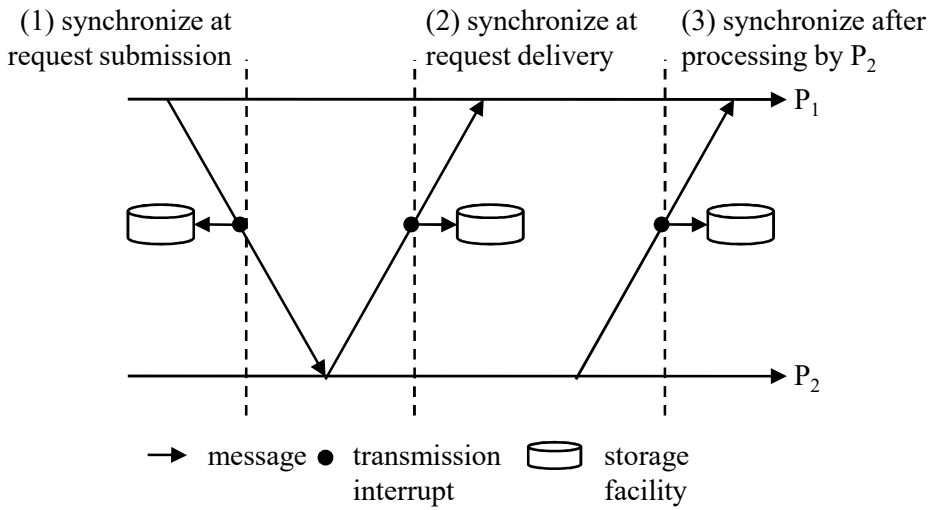


Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

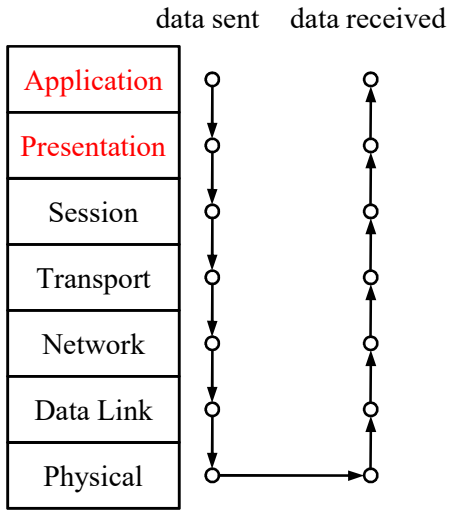
Communication: we must fix the way to coordinate and to manage the communication between processes.

Transient communication: with it, a message is stored by the communication system only as long as the sending and receiving applications are executing. In this case, the communication system consists of traditional store-and-forward routers.

Persistent communication: with it, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver. In this case, the middleware will store the message at one or several storage facilities. Various combinations of synchronization and persistence occur in practice:



Introduction (5)



Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

Communication: we must fix the way to coordinate and to manage the communication between processes.

Synchronous (asynchronous) communication requires that the sender is blocked until its request is known to be accepted. It is asynchronous otherwise.

Blocking (non-blocking) primitive returns to the invoking process after the processing for the primitive (whether in synchronous or asynchronous mode) completes. If the control returns back immediately after invocation, it is non-blocking.

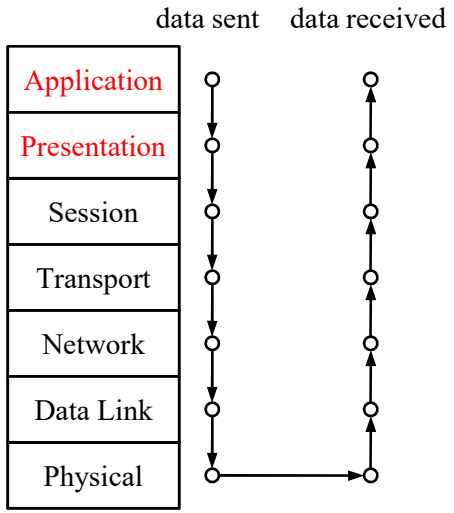
Quality of Service (QoS) refers requirements describing what is needed from the underlying distributed system to ensure services. QoS for stream-oriented communication concerns mainly timeliness, volume and reliability.

Failure model: communication suffers of failures (omissions, messages are not guaranteed to be delivered in sender order, processes can crash, etc.) that must be handled.

Group communication: a group is a collection of processes that act together in some system or user-specified way. The key property of a group is that when a message is sent to the group itself, all members of the group receive it. It is a form of one-to-many communication (one sender, many receivers).

Etc.

Introduction (6)



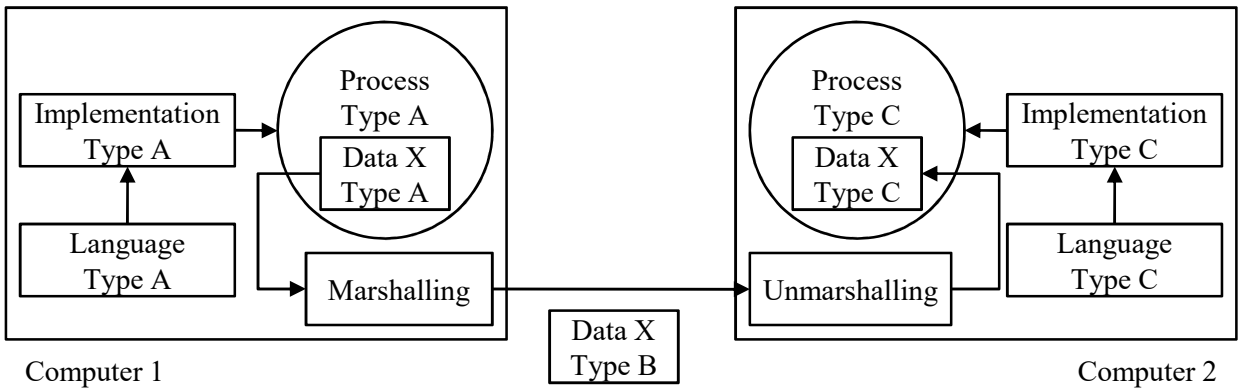
Inter-Process Communication (IPC) is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. There are two main aspects to consider: **communication** and **interoperability**.

Interoperability: when a communication is possible, the last problem is to make data readable between different systems.

IDL (Interface Definition Language): an open distributed system offers services according to standard rules that describes the syntax and semantics of these services. Such rules are formalized in protocols, and specified through interfaces described with an IDL.

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission.

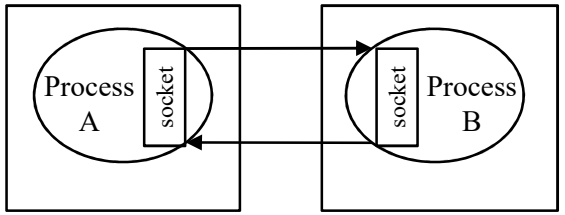
Unmarshmalling is the process if disassembling data on arrival to produce an equivalent collection of data items at the destination.



Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

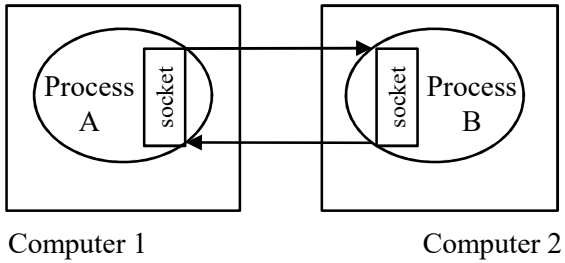
Socket Communication (1)



Sockets constitutes a mechanism for delivering incoming data packets to the appropriate application process, based on a combination of local and remote addresses and port numbers. Each socket is mapped by the operational system to a communicating application process.

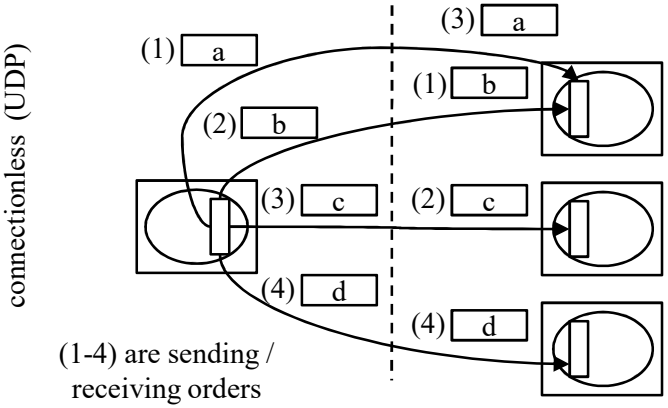
Socket is characterized by a unique combination of:
- a protocol transfer (UDP, TCP, etc.).
- the local socket address and port number.
- the remote socket address and port number.

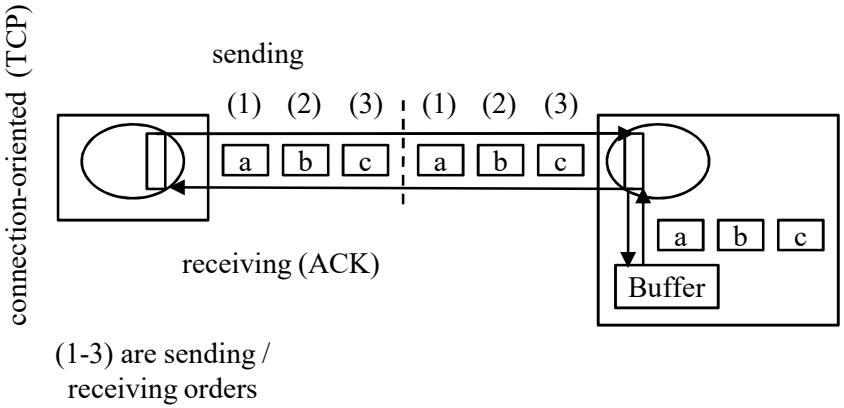
Socket Communication (2)



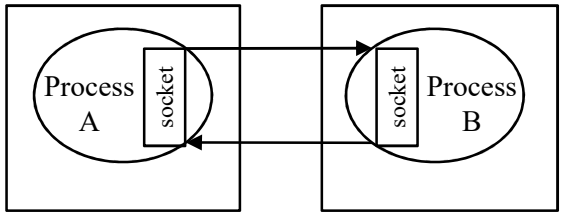
Connectionless-oriented communication (e.g. UDP) is a data transmission method in which each data packet carries information in a header that contains a destination address sufficient to permit the independent delivery of the packet to its destination via the network.

Connection-oriented communication (e.g. TCP) is a data communication mode in which the devices at the end points use a protocol to establish an end-to-end logical or physical connection before any data that may be sent.





Socket Communication (3)

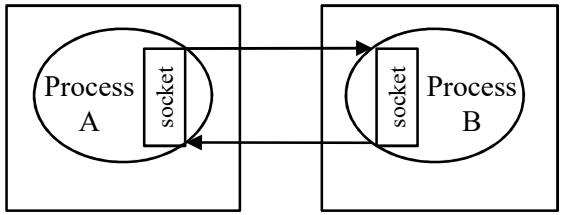


Connectionless-oriented communication (e.g. UDP) is a data transmission method in which each data packet carries information in a header that contains a destination address sufficient to permit the independent delivery of the packet to its destination via the network.

Connection-oriented communication (e.g. TCP) is a data communication mode in which the devices at the end points use a protocol to establish an end-to-end logical or physical connection before any data that may be sent.

	Protocols	**Data**	**Process(es)**		**Message**			**Mode**		
			sending	**receiving**	**checking**	**ordering**	**buffer**	**communi-cation**	**send**	**receive**
message-oriented	UDP	Packet	1	1/n	no/yes	no/yes	yes	half duplex	asynchronous / non-blocking	asynchronous / blocking
stream-oriented	TCP	Stream		1	yes	yes		full duplex	synchronous / non-blocking	synchronous / blocking

Socket Communication (4)



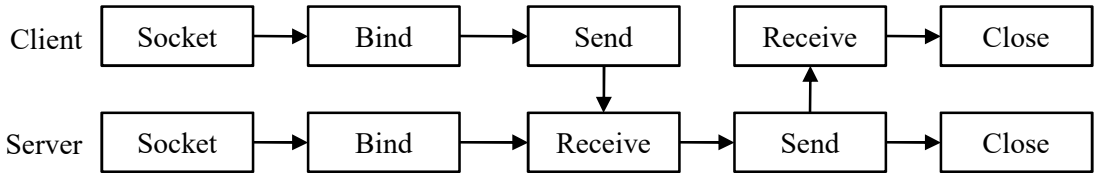
UDP sockets establish the UDP protocol.

Primitive	**Meaning**
Socket	Create a new communication end point
Bind	Attach a local address to the socket
Close	Release the connection

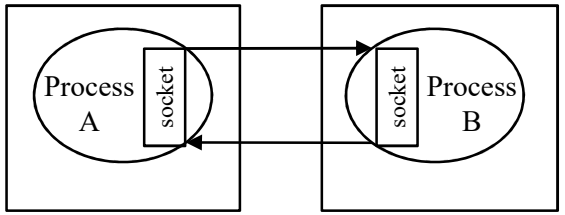
Common primitives

Primitive	**Meaning**
Send	Send some data over the connection
Receive	Receive some data over the connection

UDP primitives



Socket Communication (5)



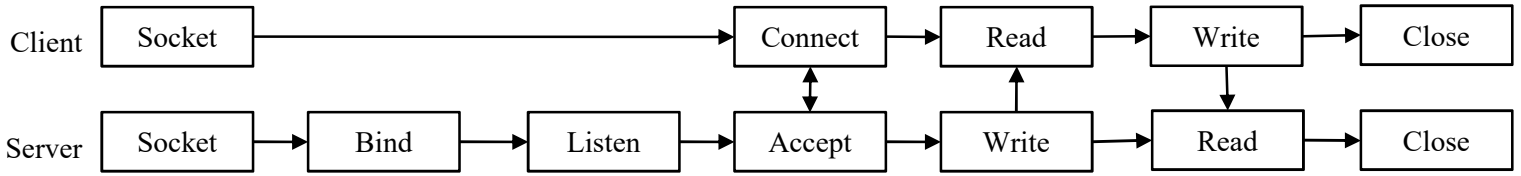
TCP sockets establish the TCP protocol.

Primitive	**Meaning**
Socket	Create a new communication end point
Bind	Attach a local address to the socket
Close	Release the connection

Common primitives

Primitive	**Meaning**
Listen	Announce willingness to accept connections
Accept	Block caller until a connexion request arrives
Connect	Actively attempt to establish a connection
Read	Read some data over the connection
Write	Write some data over the connection

TCP primitives



Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Stream-Oriented Communication
"Introduction" (1)

Stream oriented communication is a data communication mode whereby the devices at the end points use a protocol to establish an end-to-end logical or physical connection before any data that may be sent.

Continuous (representation) media: temporal relationships between data items are fundamental to interpret the data.
e.g. video, audio

Discrete (representation) media: temporal relationships between data items are not fundamental to interpret the data.
e.g. exe files

Streaming live data: data is captured in real time and sent over the network to recipients.

Streaming stored data: data is not captured in real-time, but stored in a remote computer.

Simple stream consists of only a single sequence of data.

Complex stream consists of several related simple streams called substreams. The relations between substreams in a complex stream is often time independent.

Stream-Oriented Communication
"Introduction" (2)

Stream oriented communication is a data communication mode whereby the devices at the end points use a protocol to establish an end-to-end logical or physical connection before any data may be sent.

End-to-end delay refers to the time taken for a packet to be transmitted across a network from source to destination.

$$d_{end-end} = N\left(d_{trans} + d_{prop} + d_{proc}\right)$$

$d_{end\text{-}to\text{-}end}$	**end-to-end delay**
N	is the number of router switch.
d_{trans}	**Transmission delay** is the amount of time required to push all of the packet's bits into the wire, this is the delay caused by the data-rate of the link.
d_{prop}	**Propagation delay** is the amount of time it takes for the head of the signal to travel from the sender. It depends of the link length and the propagation speed.
d_{proc}	**Processing delay** is the time it takes routers to process the packet header.

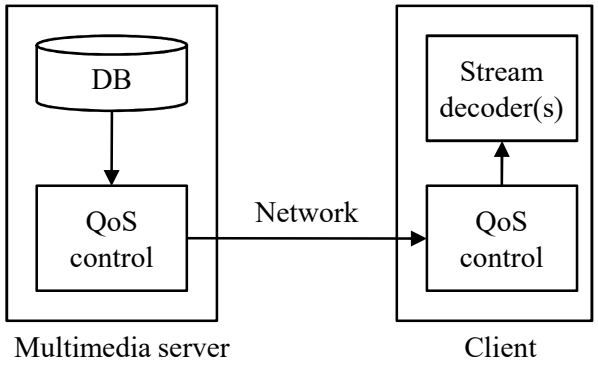
Transmission mode: timing is crucial in continuous data stream. To capture timing aspects, a distinction must be made between different transmission modes.

synchronous	$d_{end\text{-}to\text{-}end} < max$
isochronous	$min < d_{end\text{-}to\text{-}end} < max$
asynchronous	$d_{end\text{-}to\text{-}end}$ is free

Jitter is the $d_{end\text{-}to\text{-}end}$ delay variance.

Stream-Oriented Communication
"Quality of Service (QoS)" (1)

Quality of Service (QoS) refers requirements describing what is needed from the underlying distributed system to ensure services. QoS for stream-oriented communication concerns mainly timeliness, volume and reliability.



Main requirements of QoS

(Minimum) bit rate	is the rate at which data should be transported.
Maximum session delay	is the delay until a session has been set up (i.e. when an application can start sending data).
Min/max values of the $d_{end\text{-}to\text{-}end}$ delay	are the bounded the $d_{end\text{-}to\text{-}end}$ delay $min < d_{end\text{-}to\text{-}end} < max$.
The $d_{end\text{-}to\text{-}end}$ delay jitter	is the $d_{end\text{-}to\text{-}end}$ delay variance.
The round trip delay	is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received.

Stream-Oriented Communication
"Quality of Service (QoS)" (2)

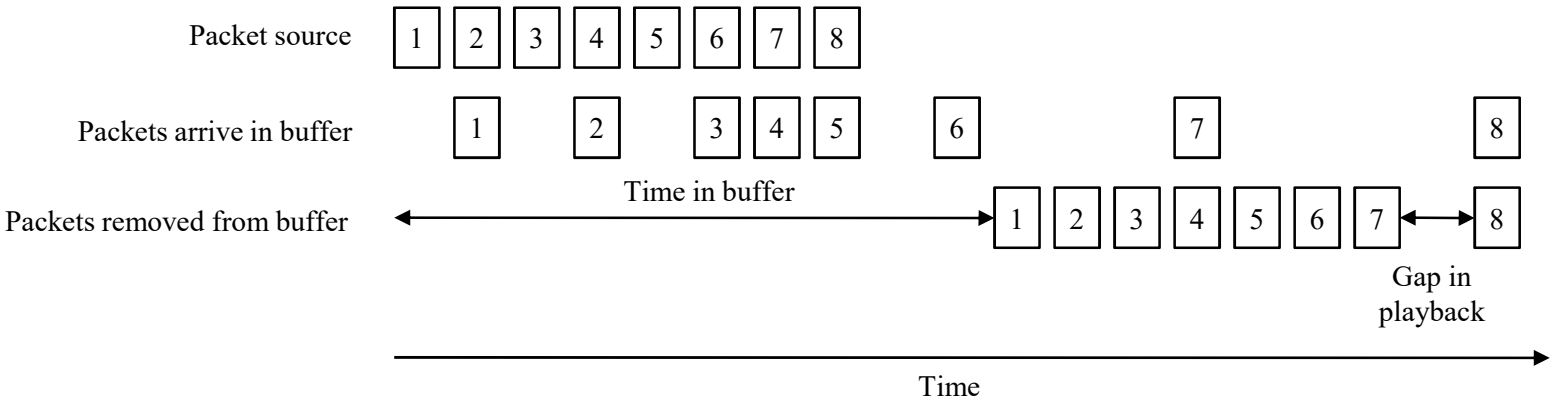
Quality of Service (QoS) refers requirements describing what is needed from the underlying distributed system to ensure services. QoS for stream-oriented communication concerns mainly timeliness, volume and reliability.

Enforcing QoS means than a distributed system can try to conceal as much as possible of the lack of QoS. There are several mechanisms that it can deploy.

(1) Internet provides a mean for differentiating classes of data.

Expedited forwarding	Specifies that a packet should be forwarded by the current router with an absolute priority.
Assured forwarding	By which traffic is divided into four subclasses, along with three ways to drop packets if network is congested.

(2) Buffering can help in getting data across the receivers.

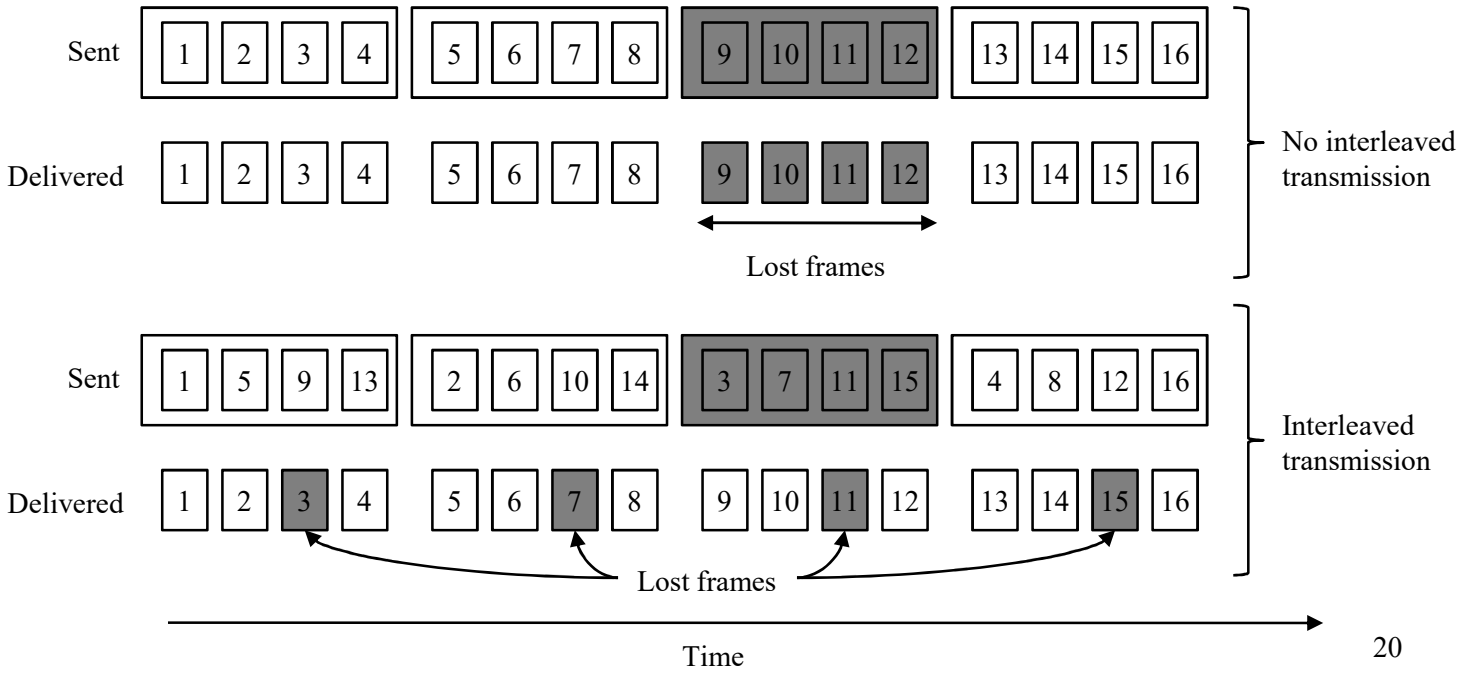


Stream-Oriented Communication
“Quality of Service (QoS)” (3)

Quality of Service (QoS) refers requirements describing what is needed from the underlying distributed system to ensure services. QoS for stream-oriented communication concerns mainly timeliness, volume and reliability.

Enforcing QoS means than a distributed system can try to conceal as much as possible of the lack of QoS. There are several mechanisms that it can deploy.

(3) To compensate packet lost, **Forward Error Correction (FEC)** can be applied (any k of the n received packets is enough to reconstruct k packets). To support gap resulting of packet lost, **interleaved transmission** can be used.



Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Primitives for Communication (1)

Primitives for communication: all the message-oriented communication in a distributed system is based on message passing; that refers the operations to send and to receive messages. It involves a set of primitives for communication (e.g. UDP, RPC, MPI, etc.).

Primitives for Communication (2)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication.

Send primitive: the send primitives has at least two parameters (a) the destination (b) the buffer in the user space, containing the data to be sent.

Receive primitive: similarly, the receive primitive has at least two parameters (a) the source from which the data is to be received (2) the user buffer into which the data is to be received.

Primitives for Communication (3)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication.

Blocking primitives: a primitive is blocking if control returns to the invoking process after the processing for the primitive (whether in synchronous or asynchronous mode) completes.

Non-blocking primitives: a primitive is non-blocking if control returns back to the invoking process immediately after invocation, even though the operation has not completed.

Primitives for Communication (4)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication.

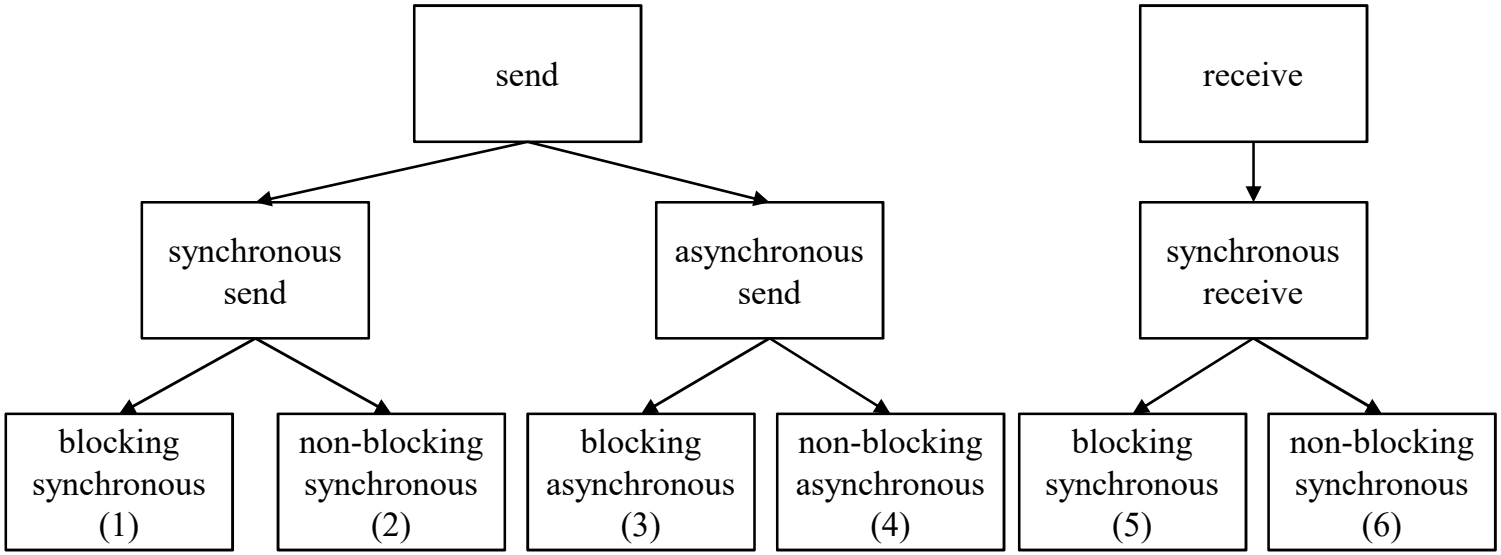
Synchronous primitives: a send (or a receive) primitive is synchronous if both handshake with each other.

Asynchronous primitives: a send primitive is said to be asynchronous if control returns back to the invoking process after the data item to be sent has been copied out of the user-specified buffer. The asynchronous receive could be presented as a specific setting of the synchronous case (i.e no reply).

Primitives for Communication (5)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication.

There are therefore four versions of the send primitive (1) blocking synchronous (2) non-blocking synchronous (3) blocking asynchronous (4) non-blocking asynchronous. For the receive primitive, there are two versions (5) blocking synchronous (6) non-blocking synchronous.

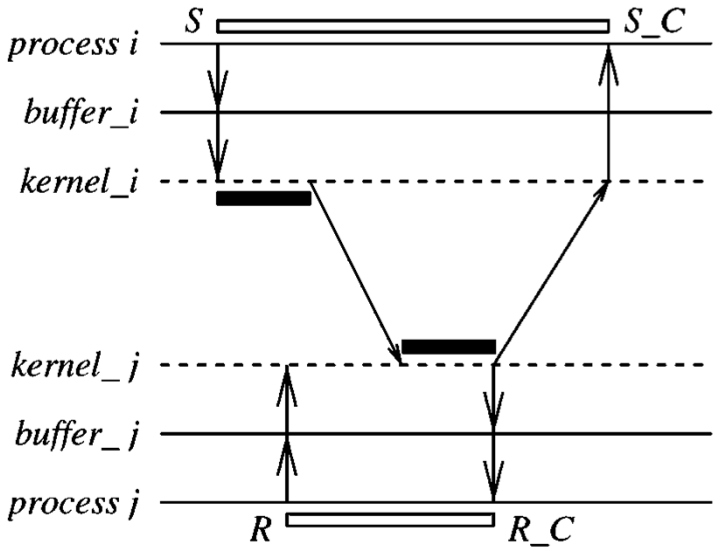


Primitives for Communication (6)

The following are some definitions of primitives for communication:

(1) Blocking synchronous send: the data gets copied from the user buffer to the kernel buffer and is then sent over the network. After the data is copied to the receiver's system buffer, an acknowledgement back to the sender causes control to return to the process and completes the send.

(5) Blocking synchronous receive: the receive call blocks until the data expected arrives and is written in the specified user buffer. Then control is returned to the user process.



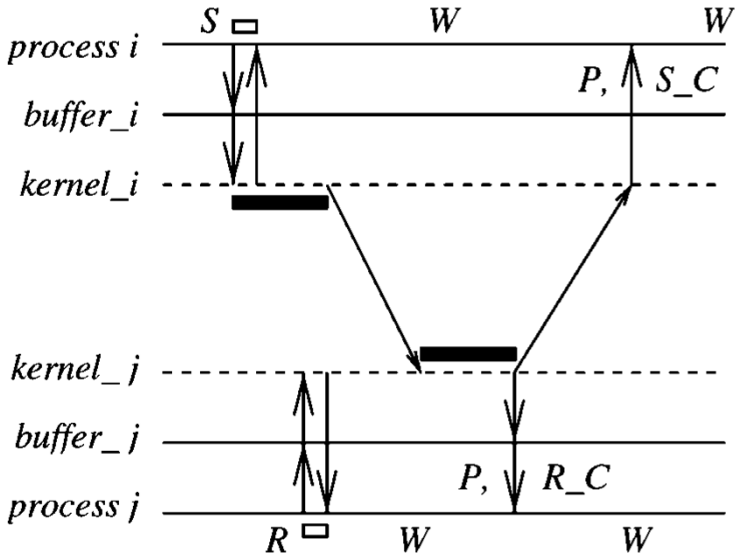
▬	Duration to copy data from or to user buffer		
▭	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

Primitives for Communication (7)

The following are some definitions of primitives for communication:

(2) Non-blocking synchronous send: control returns back to the invoking process as soon as the copy of data from the user buffer to the kernel buffer is initiated. A parameter in the non-blocking call also gets set with the handle of a location. The user process can invoke the blocking wait operation on the returned handle.

(6) Non-blocking synchronous receive: the receive call will cause the kernel to register the call and return the handle of a location that the user process can later check for the completion. This location gets posted by the kernel after the expected data arrives and is copied to the user-specified buffer.

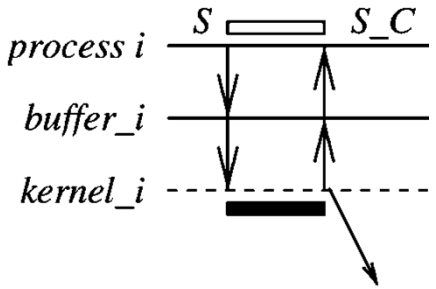


▬	Duration to copy data from or to user buffer		
▭	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

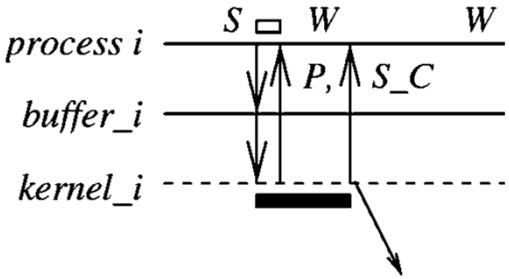
Primitives for Communication (8)

The following are some definitions of primitives for communication:

(3) Blocking asynchronous send: the user process that invokes the send is blocked until the data is copied from the user's buffer to the kernel buffer.



(4) Non-blocking asynchronous send: the user process that invokes the send is blocked until the transfer of the data from the user's buffer to the kernel buffer is initiated. The asynchronous send completes when the data has been copied out of the user's buffer.



▬	Duration to copy data from or to user buffer		
▭	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

Inter-Process Communication in Distributed Systems

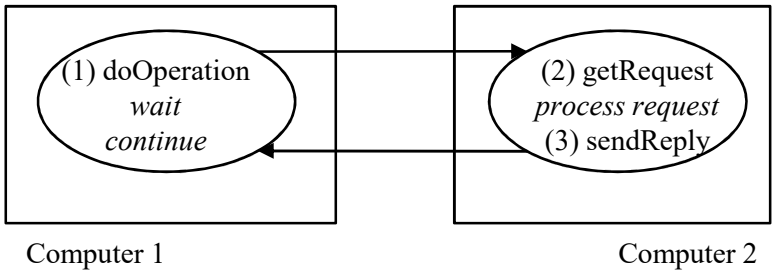
1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Request-Reply Protocols (1)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Communication operations: the Request-Reply protocol is based on trio of communication operations.

Operations	**Meaning**
doOperation	is used by a process to invoke a remote operation. The process sends a request message and invokes a receive to get a reply message.
getRequest	is used by the remote process to get the request from the caller process.
sendReply	is used by the remote process to send the reply to the caller process.



Request-Reply Protocols (2)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Message structure: is related to the information to be transmitted.

Field	**Type**	**Description**
messageType	integer or boolean	request (= 0) or reply (=1)
MessageId	integer	is a message identifier, to check that a reply message is the result of the current request. It is composed of two parts: - a request id: an increasing sequence of integers of the sending process. - an identifier for the sender process (e.g. port and internet address).
Data	any	Na

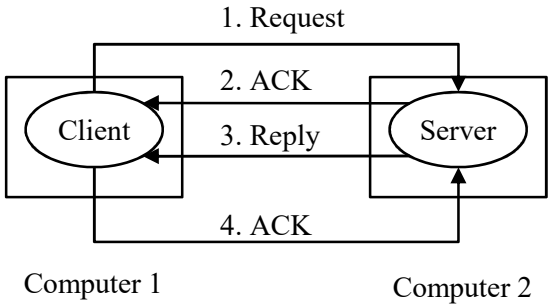
Request-Reply Protocols (3)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Failure model: the Request-Reply protocol suffers of same communication failures (omissions, messages are not guaranteed to be delivered in sender order, processes can crash, etc.). Different aspects must be considered.

Exchange protocols: three protocols, with different semantics in the presence of communication failures, are used to implement various types of request-reply protocols.

Protocols	**1. request**	**2. ACK**	**3. reply**	**4. ACK**
Request (R)	√			
Request -Acknowledge Request (RA)	√	√		
Request Reply (RR)	√		√	
Request Reply-Acknowledge Reply (RRA)	√		√	√



Request-Reply Protocols (4)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Failure model: the Request-Reply protocol suffers of same communication failures (omissions, messages are not guaranteed to be delivered in sender order, processes can crash, etc.). Different aspects must be considered.

Timeouts: when a communication fails, the doOperation uses a timeout when it is waiting to get the reply of the remote process. There are various options that can do a doOperation after a timeout.

Description
To return immediately from doOperation with an indication that the communication has failed.
To send the request message repeatedly until either it gets a reply or else it is reasonably sure that the delay is due to lack of response of the remote process rather than a lost message.

Request-Reply Protocols (5)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Failure model: the Request-Reply protocol suffers of same communication failures (omissions, messages are not guaranteed to be delivered in sender order, processes can crash, etc.). Different aspects must be considered.

Discarding duplicate request messages: when a request message is retransmitted, the receiver may repeat the operation. The receiver recognizes successive messages using the request identifier, and filters out the duplicates.

Request-Reply Protocols (6)

The Request-Reply protocol is one of the basic protocol that computers use to communicate to each other. When using request-reply, the first computer requests some data and the second computer responds to the request.

Failure model: the Request-Reply protocol suffers of same communication failures (omissions, messages are not guaranteed to be delivered in sender order, processes can crash, etc.). Different aspects must be considered.

Lost reply message: if the remote process receives a duplicate request, it needs to resend the reply. We must distinguish two operation cases:

- *idempotent operation* can be performed repeatability with the same effect as if it has been performed exactly once.

- if false, it is not an idempotent operation and an history must be used. History refers to a data structure that contains the record of (reply) messages. A problem associated with the use of history is the extra memory cost.

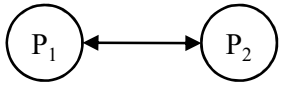
Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

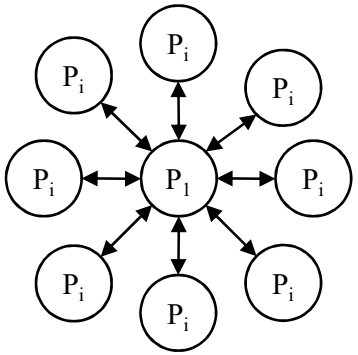
Group Communication (1)

Group communication: a group is a collection of processes that act together in some system or user-specified way. It is a form of one-to-many communication (one sender, many receivers), and is contrasted with point-to-point communication.

Point to point communication
(2 processes)



Group communication
(n client(s), 1 or n server(s))



Group Communication (2)

Group communication: a group is a collection of processes that act together in some system or user-specified way. It is a form of one-to-many communication (one sender, many receivers), and is contrasted with point-to-point communication.

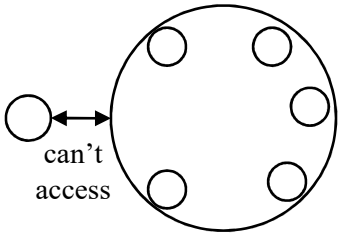
Design issue: group communication has many as the same design possibility as regular message passing, but there are a large number of additional choices that must be made, including.

<table>
<tr><th rowspan="4">Group communication</th><th>Access</th><th>Organization</th><th>Membership</th><th>Addressing</th><th>Atomicity</th><th>Message Ordering</th><th>Group Overlapping</th></tr>
<tr><td rowspan="2">Closed</td><td rowspan="2">Peer</td><td rowspan="2">Server</td><td>Multicast</td><td rowspan="2">no</td><td rowspan="3">properties</td><td rowspan="2">no</td></tr>
<tr><td>Unicast</td></tr>
<tr><td rowspan="2">Open</td><td rowspan="2">Hierarchical</td><td rowspan="2">Distributed</td><td>Broadcast</td><td rowspan="2">yes</td><td rowspan="2">yes</td></tr>
</table>

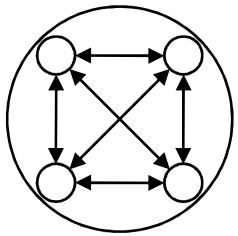
Group Communication (3)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

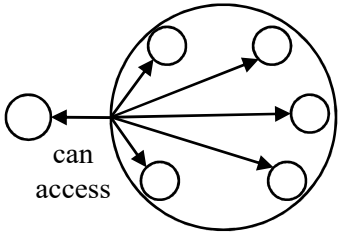
A closed group: only the members of the group can send to the group.



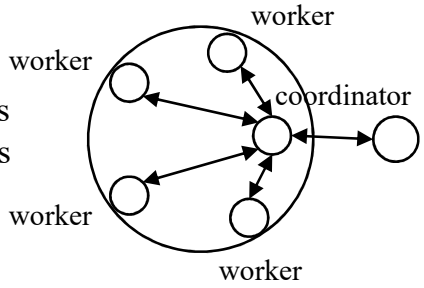
A peer group: all the processes are equal, no one is the "boss" and all decisions are made collectively.



An open group: any process in the system can send to the group.



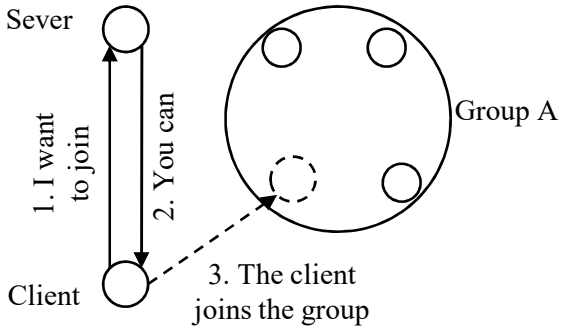
A hierarchical group: one process is the coordinator and all the others are workers.



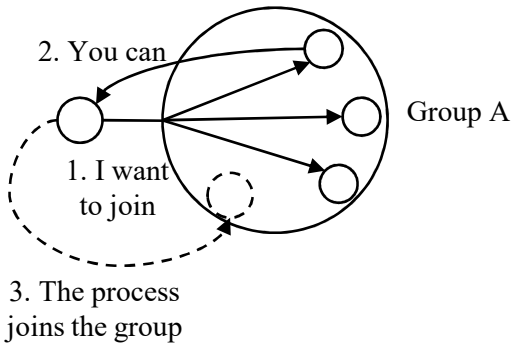
Group Communication (4)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
				Unicast			
	Open	Hierarchical	Distributed	Broadcast	yes		yes

Server membership: the group server maintains a database about the groups and their membership. Requests are sent to the server to delete, to create, to leave or to join a group.



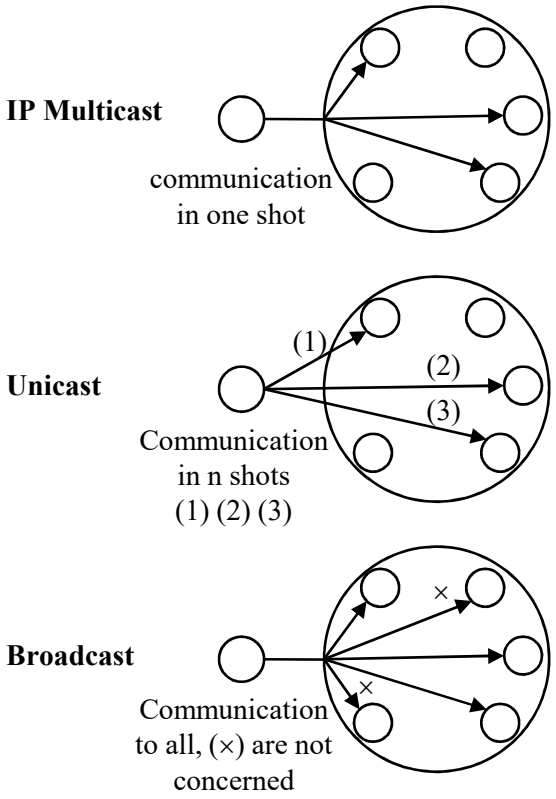
Distributed membership: an outsider can send a message to all the group members announcing its presence.



Group Communication (5)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

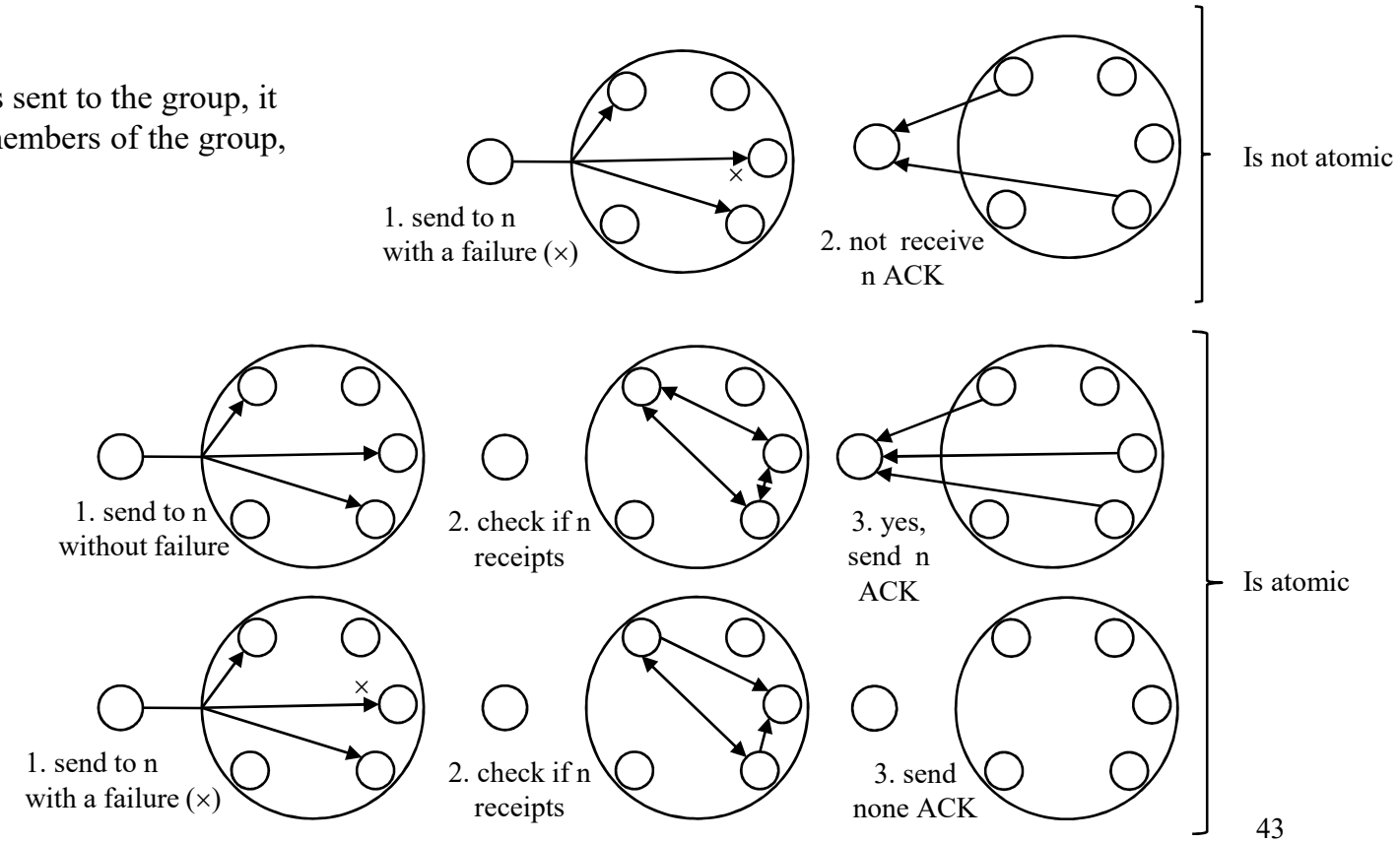
Addressing: in order to send a message to a group, a process must have some way of specifying which group it means. Group must be addressed just as processes do. Three implementation methods exist.



Group Communication (6)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

Atomicity: is when a message is sent to the group, it must arrive correctly to all the members of the group, or at none of them.



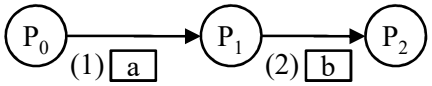
Group Communication (7)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

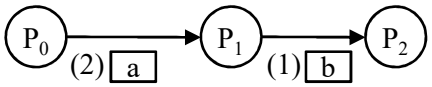
Message ordering: in basic group communication, messages are delivered to processes in arbitrary order. In many cases, this lack of ordering is not satisfactory. Group communication has to deal with message ordering.

Concurrent: two events (e.g. message exchange) are concurrent because they neither can influence each other. e.g. P_0 can deliver a message to P_1 before or after P_2

P_0 delivers before P_1

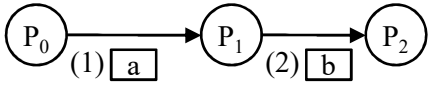


P_1 delivers before P_0



Causal: two events (e.g. message exchange) are causally related if the nature of behavior of the second one might have been influenced in any way by the first one.

P_0 must deliver before P_1



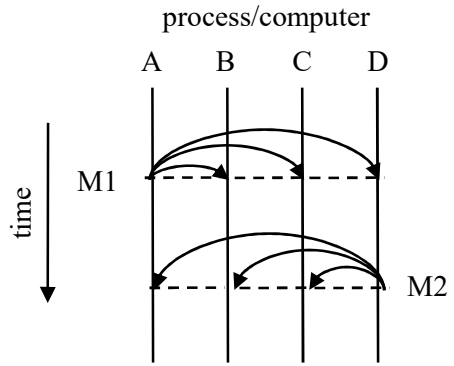
(1-2) are sending / receiving orders

Group Communication (8)

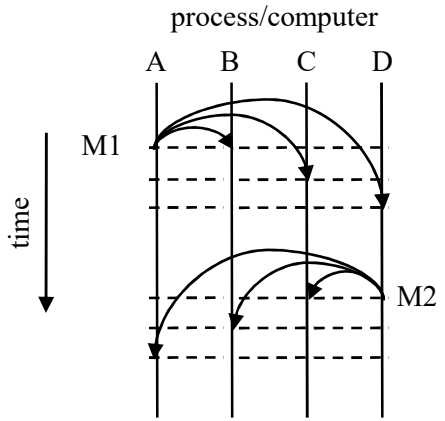
Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
				Unicast			
	Open	Hierarchical	Distributed	Broadcast	yes		yes

Message ordering: in basic group communication, messages are delivered to processes in arbitrary order. In many cases, this lack of ordering is not satisfactory. Group communication has to deal with message ordering.

A synchronous system is one in which events happen strictly sequentially (ie. zero time) (e.g. IP Multicast, broadcast).



A loosely synchronous system is one in which events take a finite amount of time but all events appear in the same order to all parties.

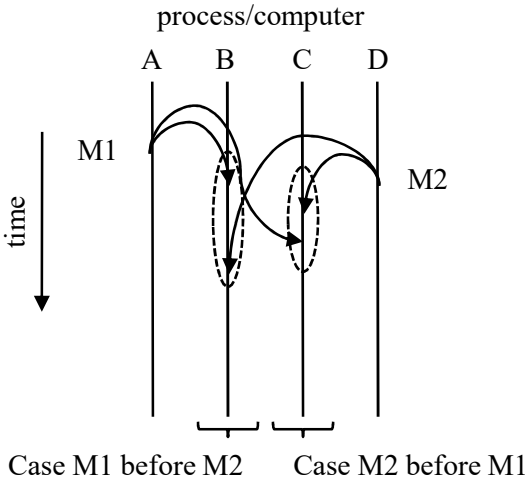


Group Communication (9)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

Message ordering: in basic group communication, messages are delivered to processes in arbitrary order. In many cases, this lack of ordering is not satisfactory. Group communication has to deal with message ordering.

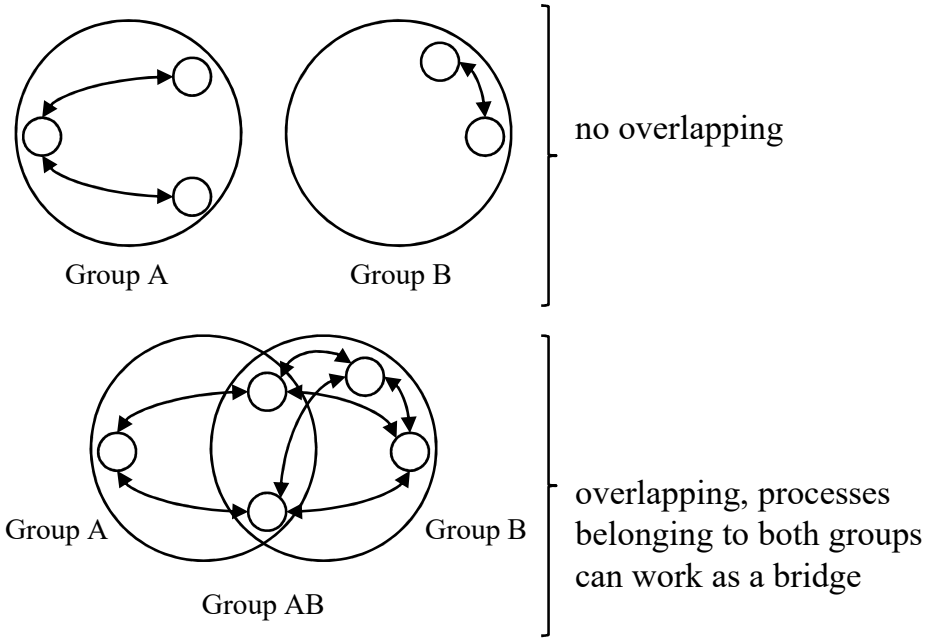
A virtually synchronous system is one in which the ordering constraint has been relaxed, but in such a way that under carefully selected circumstances, it does not matter.



Group Communication (10)

Group communication	Access	Organization	Membership	Addressing	Atomicity	Message Ordering	Group Overlapping
	Closed	Peer	Server	Multicast	no	properties	no
	Open	Hierarchical	Distributed	Unicast	yes		yes
				Broadcast			

Group overlapping: when a process can be member of multiple groups, we're discussing of group overlapping. One must take care, although there is a message ordering within each group, there is not necessarily any coordination between the groups.



Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

The Message Passing Interface (MPI) (1)

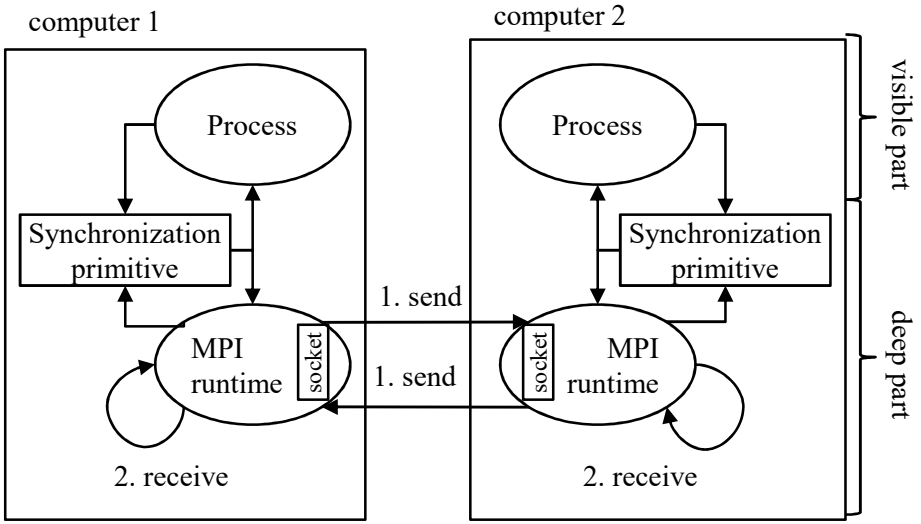
Message Passing Interface (MPI) is a standardized and portable message-passing system. It defines the syntax and semantics of a core of library routines (i.e. primitives) useful to write message-passing programs. MPI

- is tailored for transient communication.
- makes direct use of underlying network.
- assumes communication takes place within a group of known processes.
- has a static runtime system.

The Message Passing Interface (MPI) (2)

Message Passing Interface (MPI) is a standardized and portable message-passing system. It defines the syntax and semantics of a core of library routines (i.e. primitives) useful to write message-passing programs. MPI
- is tailored for transient communication.
- makes direct use of underlying network.
- assumes communication takes place within a group of known processes.
- has a static runtime system.

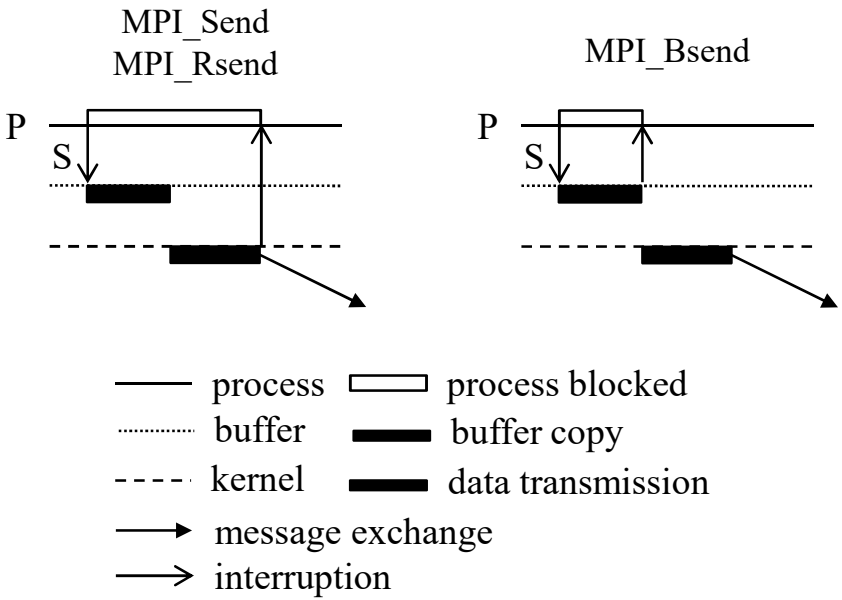
A run-time system is a software component that provides an abstraction layer to hide the complexity of services offered by the OS. The run-time systems can be used for drawing text, Internet connection, etc.



The Message Passing Interface (MPI) (3)

Message Passing Interface (MPI) is a standardized and portable message-passing system. It defines the syntax and semantics of a core of library routines (i.e. primitives) useful to write message-passing programs. MPI
- is tailored for transient communication.
- makes direct use of underlying network.
- assumes communication takes place within a group of known processes.
- has a static runtime system.

		Blocking	**Non-blocking**
Synchronous		MPI_Ssend	MPI_Issend
Asynchronous	**Generic**	MPI_Send	MPI_Isend
	Ready	MPI_Rsend	MPI_Irsend
	Buffered	MPI_Bsend	MPI_Ibsend

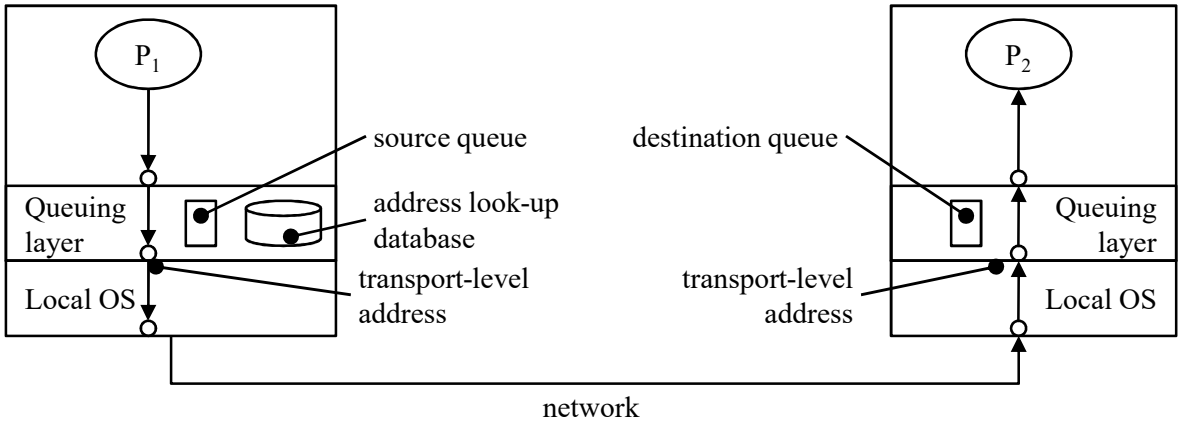


Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Message Queuing Systems (1)

Message Queuing Systems provide extensive support for persistent asynchronous communication. The essence of such system sis that they offer intermediate-term storage capacity for message (i.e. the queues), without requiring either the sender or receiver to be active during the message transmission. It permits communication loosely coupled in time, an important difference with Socket/MPI based communication is that message transfers with queuing can take minutes.



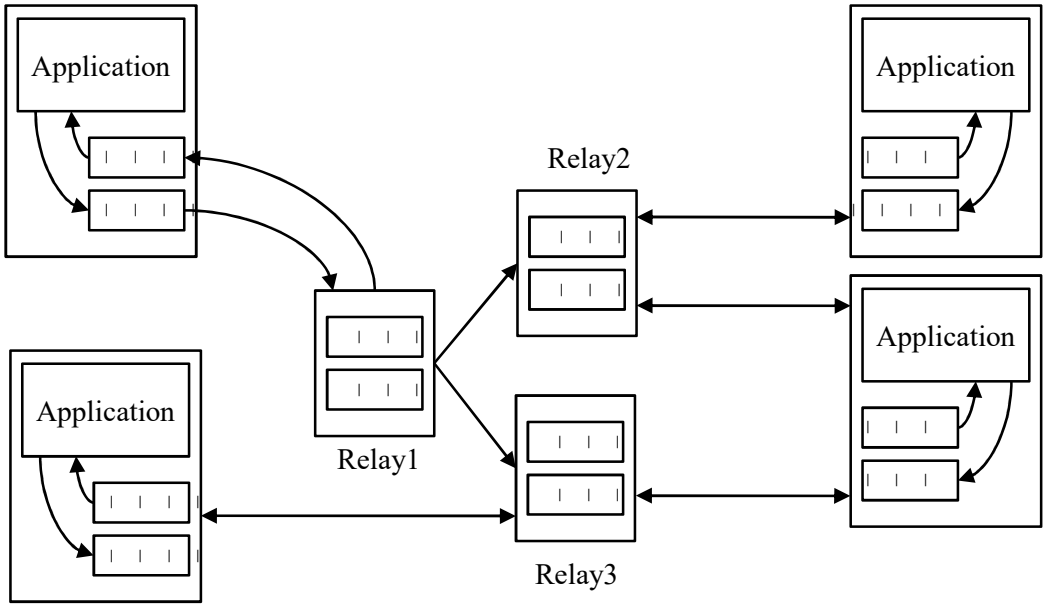
System queues: two queue levels, the source and destination queues. In principle, within a local OS each application has its own queue, but it is also possible for multiple applications to share a single queue.

Database of queue names: a queuing system must maintain a database of queue names to network locations.

Message Queuing Systems (2)

Message Queuing Systems provide extensive support for persistent asynchronous communication. The essence of such system sis that they offer intermediate-term storage capacity for message (i.e. the queues), without requiring either the sender or receiver to be active during the message transmission. It permits communication loosely coupled in time, an important difference with Socket/MPI based communication is that message transfers with queuing can take minutes.

Relay queue manager: there are special queue managers that operate as a relay; they forward incoming messages to other queue managers.



Message Queuing Systems (3)

Message Queuing Systems provide extensive support for persistent asynchronous communication. The essence of such system sis that they offer intermediate-term storage capacity for message (i.e. the queues), without requiring either the sender or receiver to be active during the message transmission. It permits communication loosely coupled in time, an important difference with Socket/MPI based communication is that message transfers with queuing can take minutes.

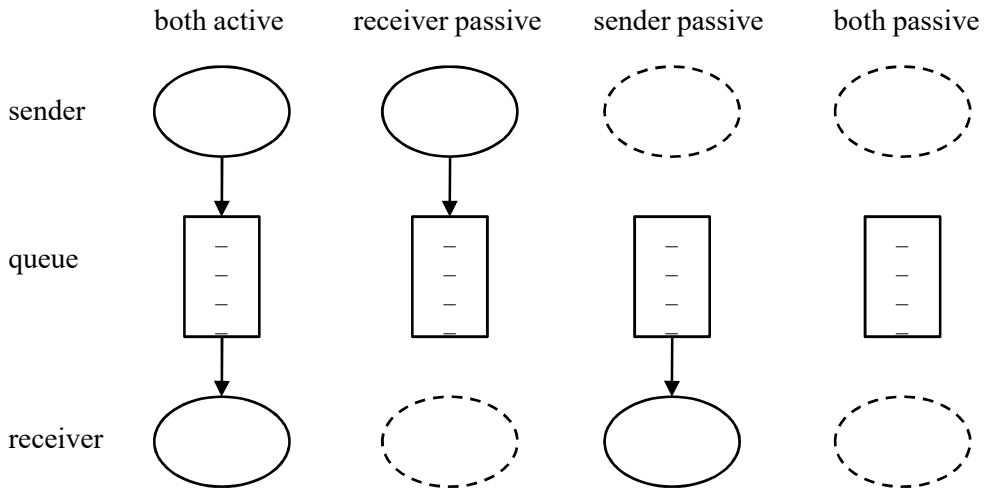
Queue manager: queues are managed by queue managers, they interact directly with the application that is sending or receiving a message. Interface offered by a queue manager can be extremely simple:

Primitives	**Meaning**
Put	append a message to a specified queue
Get	block until the specified queue is nonempty, and remove the first message
Poll	check a specified queue for message, and remove the first, never block
Notify	install a handler to be called when a message is put into the specified queue

Message Queuing Systems (4)

Message Queuing Systems provide extensive support for persistent asynchronous communication. The essence of such system sis that they offer intermediate-term storage capacity for message (i.e. the queues), without requiring either the sender or receiver to be active during the message transmission. It permits communication loosely coupled in time, an important difference with Socket/MPI based communication is that message transfers with queuing can take minutes.

Loosely-coupled communication modes: once a message has been deposited in a queue, it will remain there while its sender or receiver executing, or removed. This gives us four combinations in respect to the execution mode.

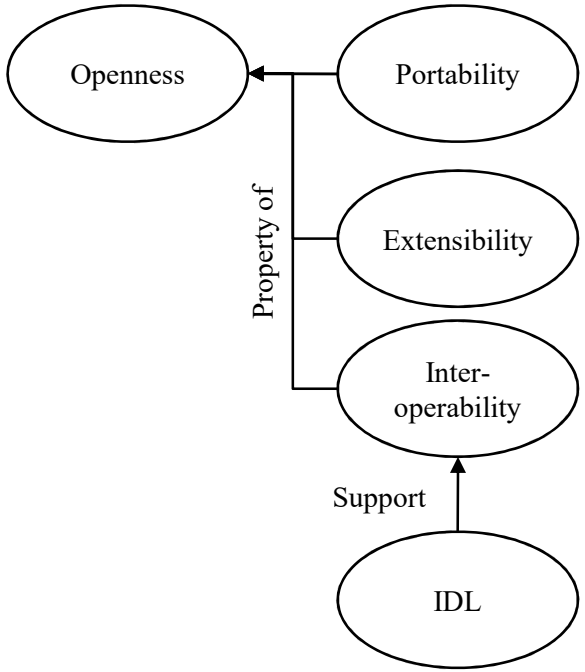


Inter-Process Communication in Distributed Systems

1. Introduction
2. Socket Communication
3. Stream-Oriented Communication
4. Message-Oriented Communication
   - 4.1. Primitives for Communication
   - 4.2. Request-Reply Protocols
   - 4.3. Group Communication
   - 4.4. The Message Passing Interface (MPI)
   - 4.5. Message Queuing Systems
5. Interoperability

Interoperability (1)

Openness: the openness in a distributed system is the characteristic that determines whether the system can be extended and re-implemented in a various way. It is characterized by the degree to which new resources-sharing and services can be added and made available for use by a variety of applications.



Portability characterizes to what extent an application developed for a distributed system A can be executed, without modifications, on a different distributed system B.

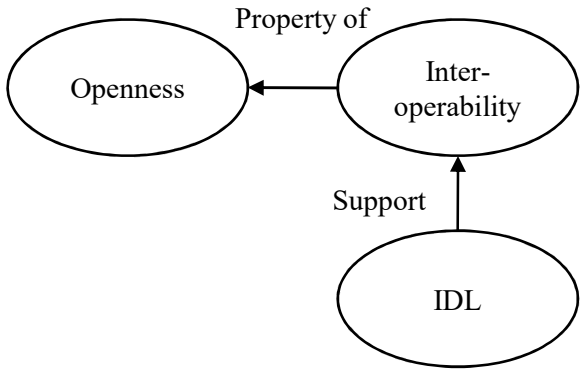
Extensibility for and opened distributed system concerns how it should be easy to configure the system out of different components, and how it should be easy to add new components or replacing the existing ones.

Interoperability characterizes the extend by which two implementations of systems or components from different manufacturers, can co-exist and work together.

IDL (Interface Definition Language): an open distributed system offers services according to standard rules that describes the syntax and semantics of these services. Such rules are formalized in protocols, and specified through interfaces described with an IDL.

Interoperability (2)

Interoperability characterizes the extend by which two implementations of systems or components from different manufacturers, can co-exist and work together.

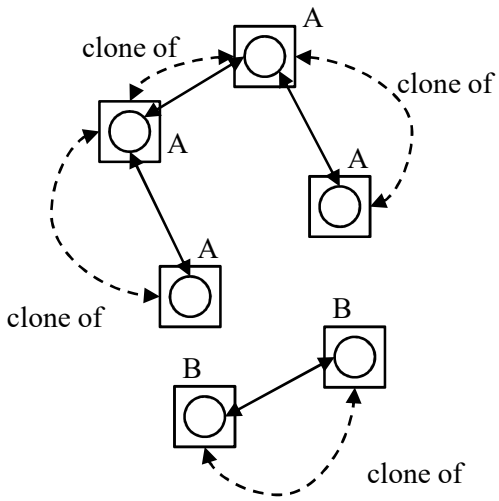


	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	no
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

Interoperability (3)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

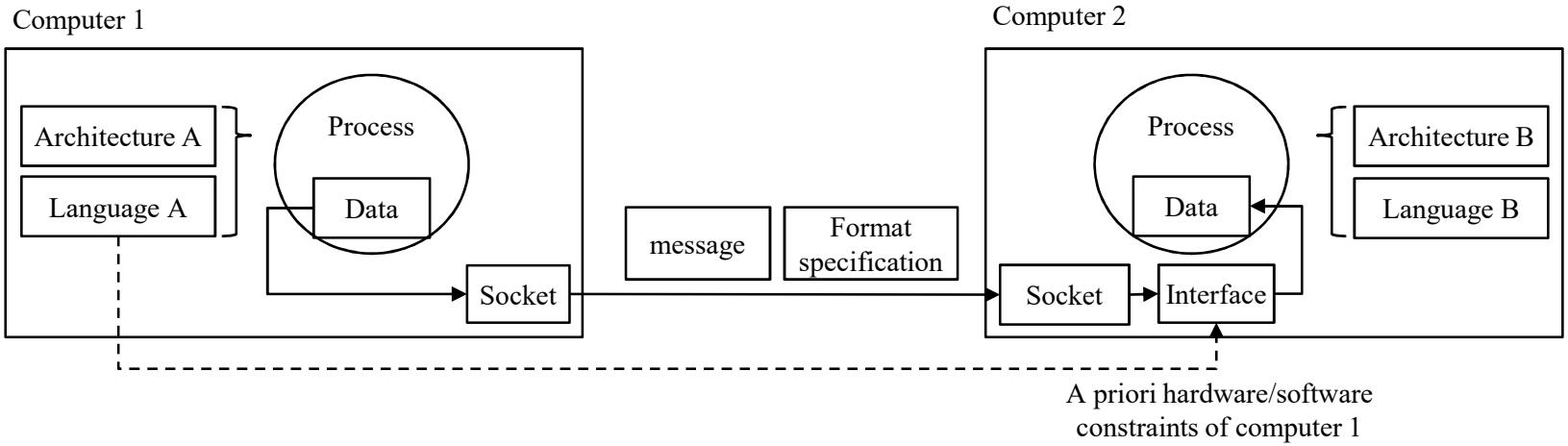
Cluster computing: the underlying hardware consists of a collection of similar computers, closely connected by means of a high-speed local network. In addition, each node runs the same OS.



Interoperability (4)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

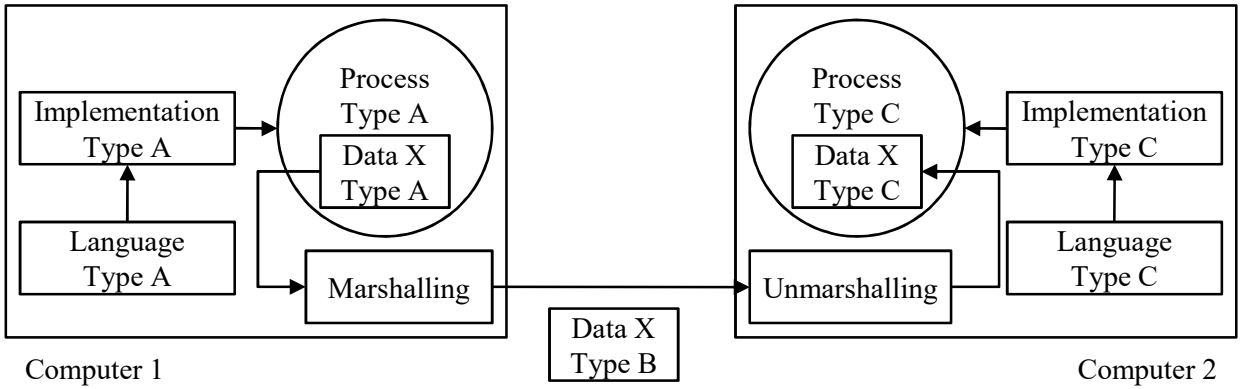
Implementing remote interfaces: the values are transmitted in the sender's format, together with an indication of the format used, and the recipient converts values if necessary.



Interoperability (5)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

External data representation is interested with standard for the representation of data structures and primitive values.



Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission.

Unmarshmalling is the process if disassembling data on arrival to produce an equivalent collection of data items at the destination.

Interoperability (6)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

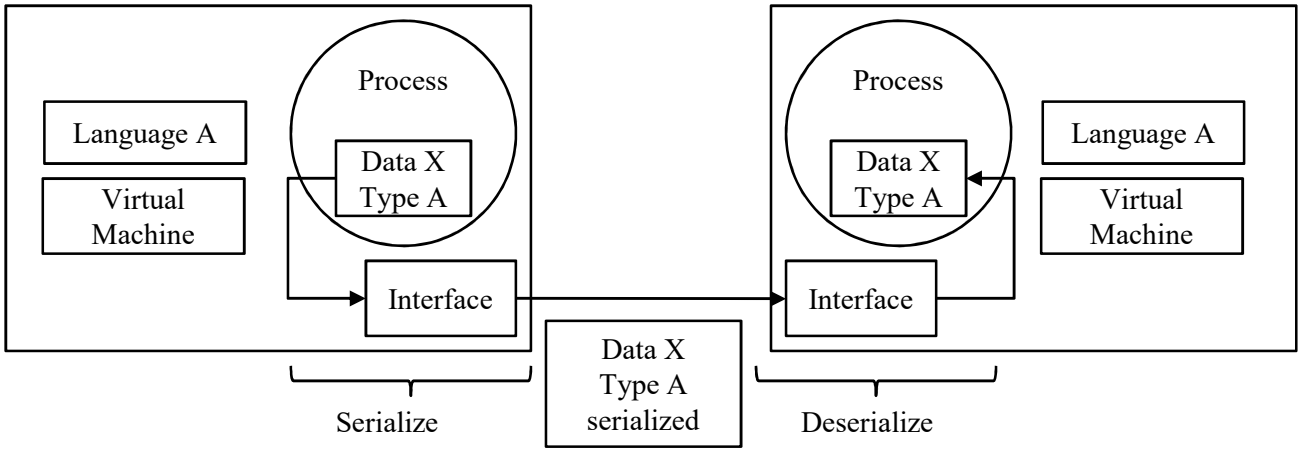
External data representation is interested with standard for the representation of data structures and primitive values.
e.g. XML Schema

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions        >
        <xsd:element name= "person" type ="personType" />
                <xsd:complexType name="personType">
                        <xsd:sequence>
                                <xsd:element name = "name"  type="xs:string"/>
                                <xsd:element name = "place"  type="xs:string"/>
                                <xsd:element name = "year"  type="xs:positiveInteger"/>
                        </xsd:sequence>
                        <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
                </xsd:complexType>
</xsd:schema>
```

Interoperability (7)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

Serialization refers to the activity of flattening an object or a connected set of objects into a serial form that is suitable for storing on disk or transmitting in a message.



Interoperability (8)

	Hardware	Software	Scalability	IDL
Cluster of computers	Same	Same	Medium	no
Implementing remote interfaces	Different	Different	Low	yes
External data representation	Different	Different	High	yes
Serialization	Different	Same	Medium	yes

Serialization refers to the activity of flattening an object or a connected set of objects into a serial form that is suitable for storing on disk or transmitting in a message.

e.g. Person p = new Person("Smith", "London", 1934);

Fields			**Description**	
1	Person		Class name	
2	42L		Serial version UID	
3	3		Number of variables	
4	int years		Types of variables	
5	String name			
6	String place			
7	1394		value	
8	5	Smith	length	value
9	6	London	length	value

Reflexion: serialization is based on the reflexion, the abilities in OOP to enquire about the properties of a class (type and names of instance variables and methods). Object can be also created from their names.

Handles: when an object is serialized, all the objects that it references are serialized together to ensure the deserialization.

Transient: serialization / deserialization are full automatic processes, but can be tuned by programmers by implementing their own methods. Part(s) of objects can be protected from serialization / deserialization processes as transient members.