MONASH University

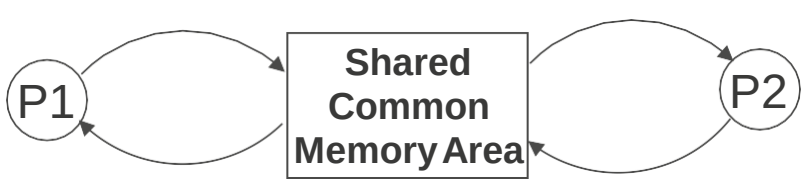
Information Technology

FIT5183: Mobile and Distributed Computing Systems (MDCS)

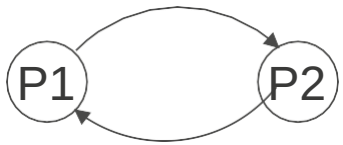
Lecture 2 – Part I
Inter-process Communications (IPC) & Remote Procedure Call (RPC)

Inter-process Communication (IPC)

- IPC basically requires information sharing among two or more processes. Two basic methods are:
  - Original sharing (Shared data approach)
  - Copy Sharing (**Message passing** approach)



Shared data approach



Message passing approach

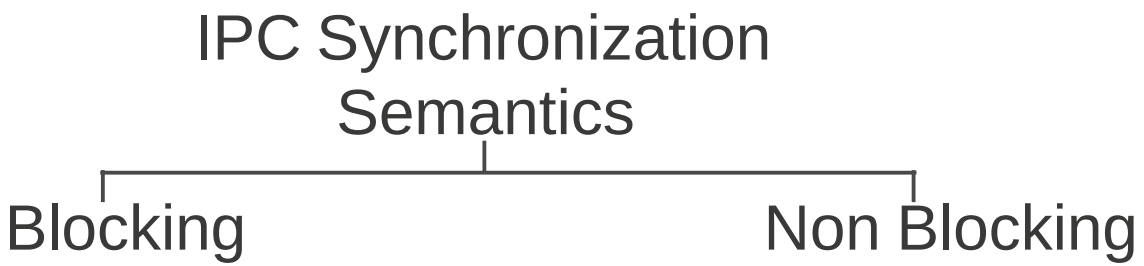
Message Passing System

- A *message-passing system* is a sub-system of a distributed system that provides a set of message-based IPC protocols and does so by shielding the details of complex network protocols and multiple heterogeneous platforms from programmers.
- It enables processes to communicate by exchanging messages
- Allows programs to be written by using simple communication primitives, such as *send* and *receive*.
- Serves as a suitable infrastructure for building other higher level IPC systems, such as RPC (Remote Procedure Call) and DSM (Distributed Shared Memory).

Issues in IPC by Message Passing

- A typical message structure
  - Header
    - Addresses
      - ✓✓Sender address
      - ✓✓Receiver address
    - Sequence number
    - Structural information
      - ✓✓Type
      - ✓✓Number of bytes
  - Message

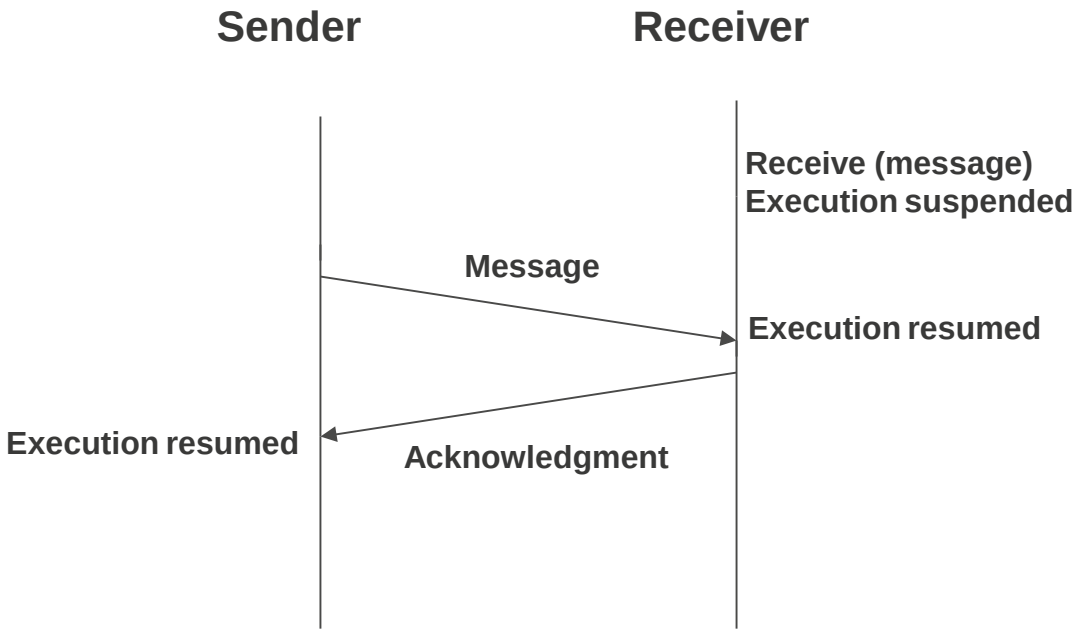
Synchronization in IPC



- Send primitive
  - Blocking
  - Non blocking
- Receive primitive
  - Blocking
  - Non Blocking
    - ✓✓**Polling**
    - ✓✓**interrupt**

Synchronous & Asynchronous Communication

- When both the send and receive primitives of a communication between two processes use blocking semantics, the communication is said to be Synchronous; otherwise it is asynchronous.



Synchronous mode with blocking send and receive primitives

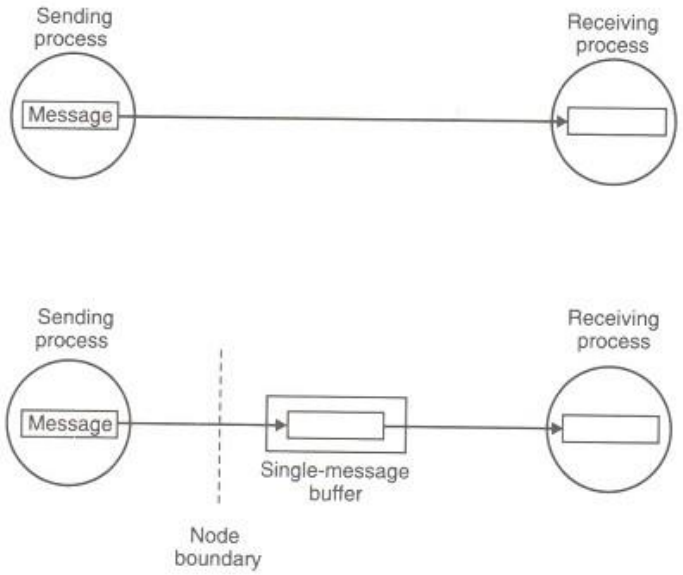
Synchronous vs. Asynchronous Communication

- **Synchronous**
  - Simple and easy to implement
  - Contributes to reliability
  - No backward error recovery needed

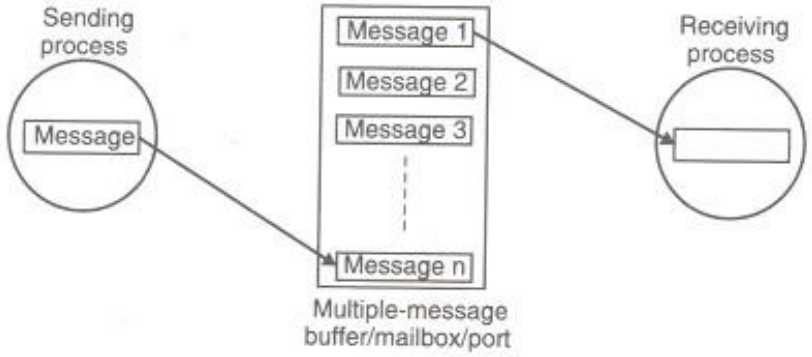
- **Asynchronous**
  - High concurrency
  - More flexible than synchronous
  - Lower deadlock risk than in synchronous communication (but beware)

Buffering

- Synchronous systems
  - Null buffer
  - Single message buffer
- Asynchronous systems
  - Unbounded capacity buffer
  - Finite message (multiple message buffer)



Synchronous System

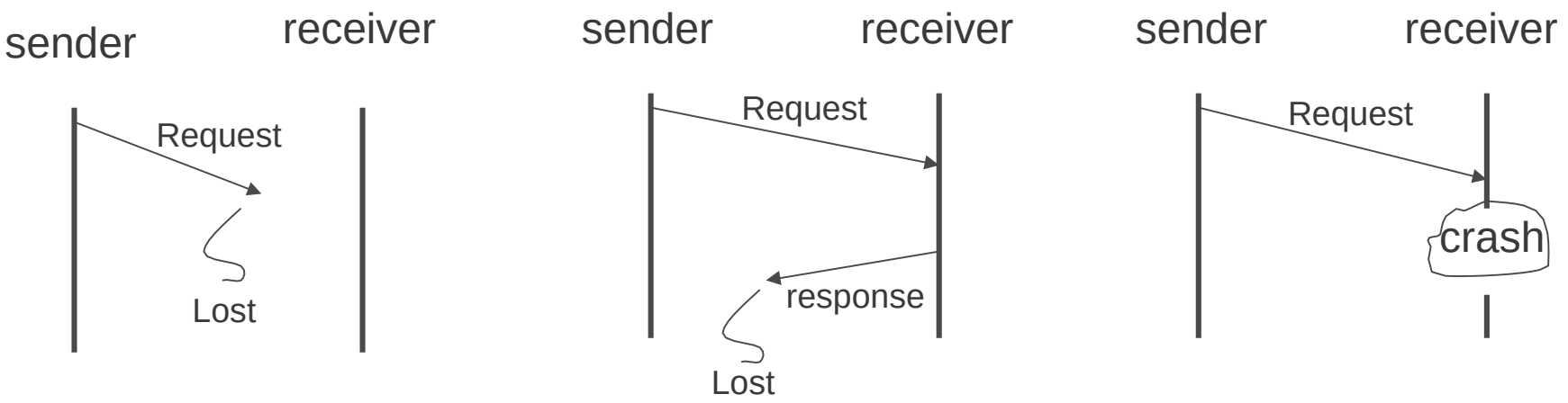


Asynchronous System

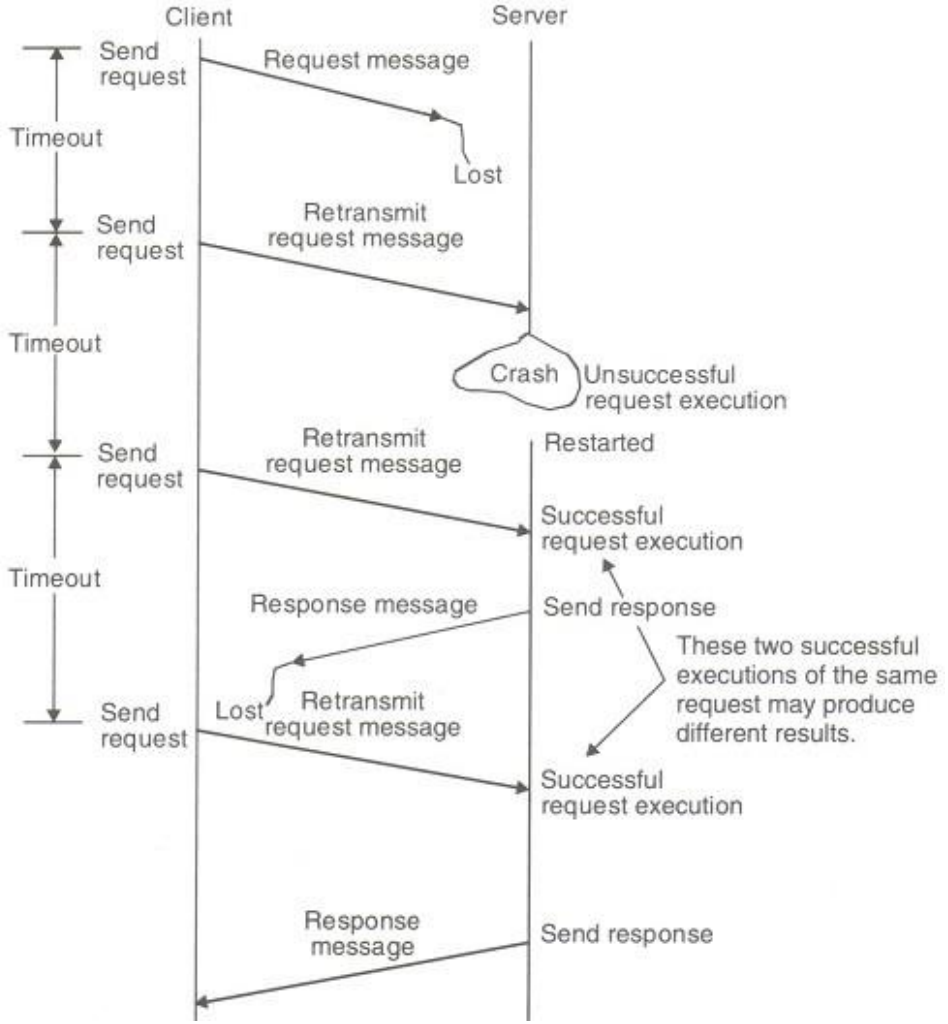
Failure Handling

- Failure classification

1. Loss of request message
2. Loss of response message
3. Unsuccessful execution of the request



An Example of Fault Tolerant System



Idempotency

What is the difference between the following two functions?

- getSqrt(n)

  { return sqrt(n)

  }

- Debit (amount)

  { if (balance>amount)

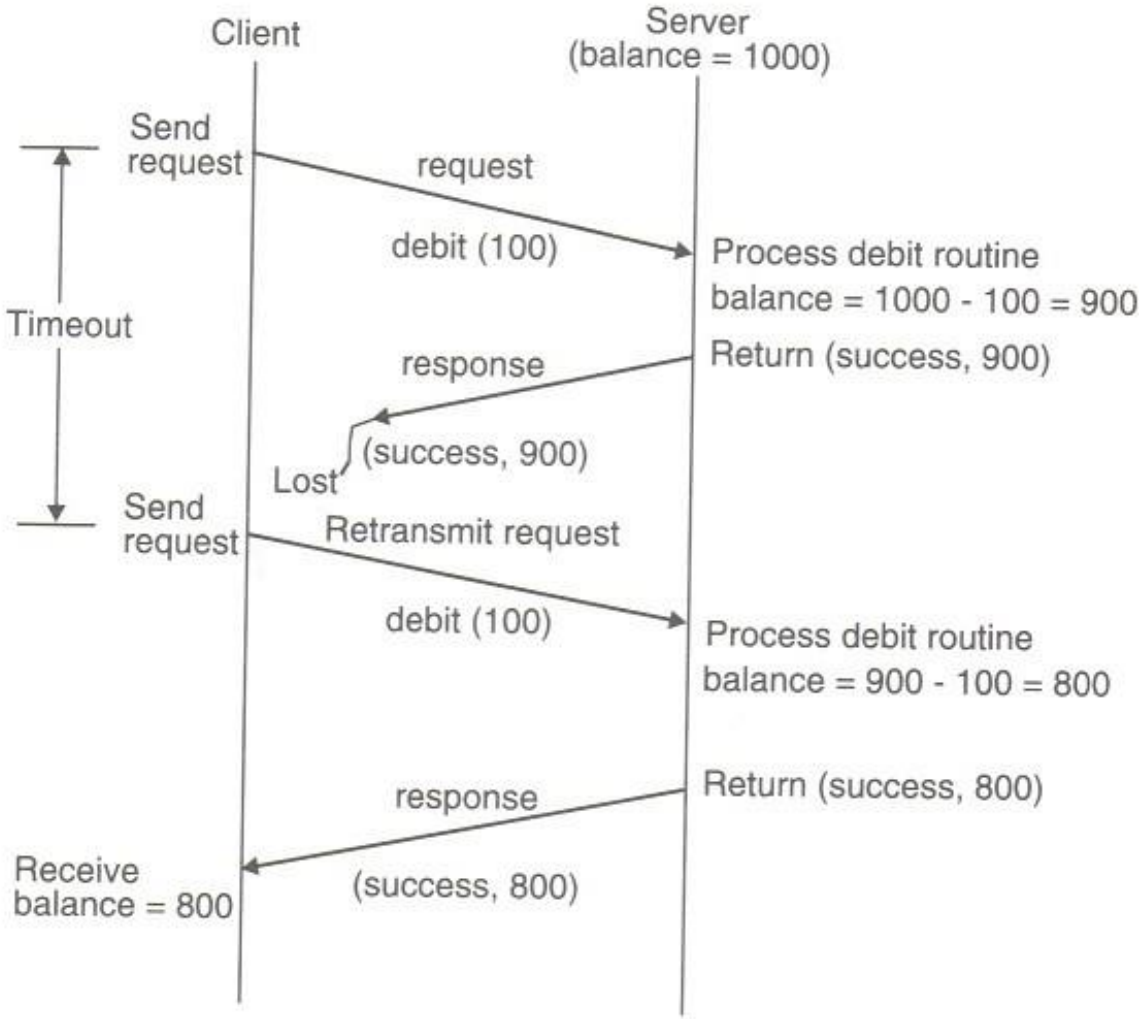
  balance= balance-amount

  return (“success”, balance)

  else return (“failure”, balance)

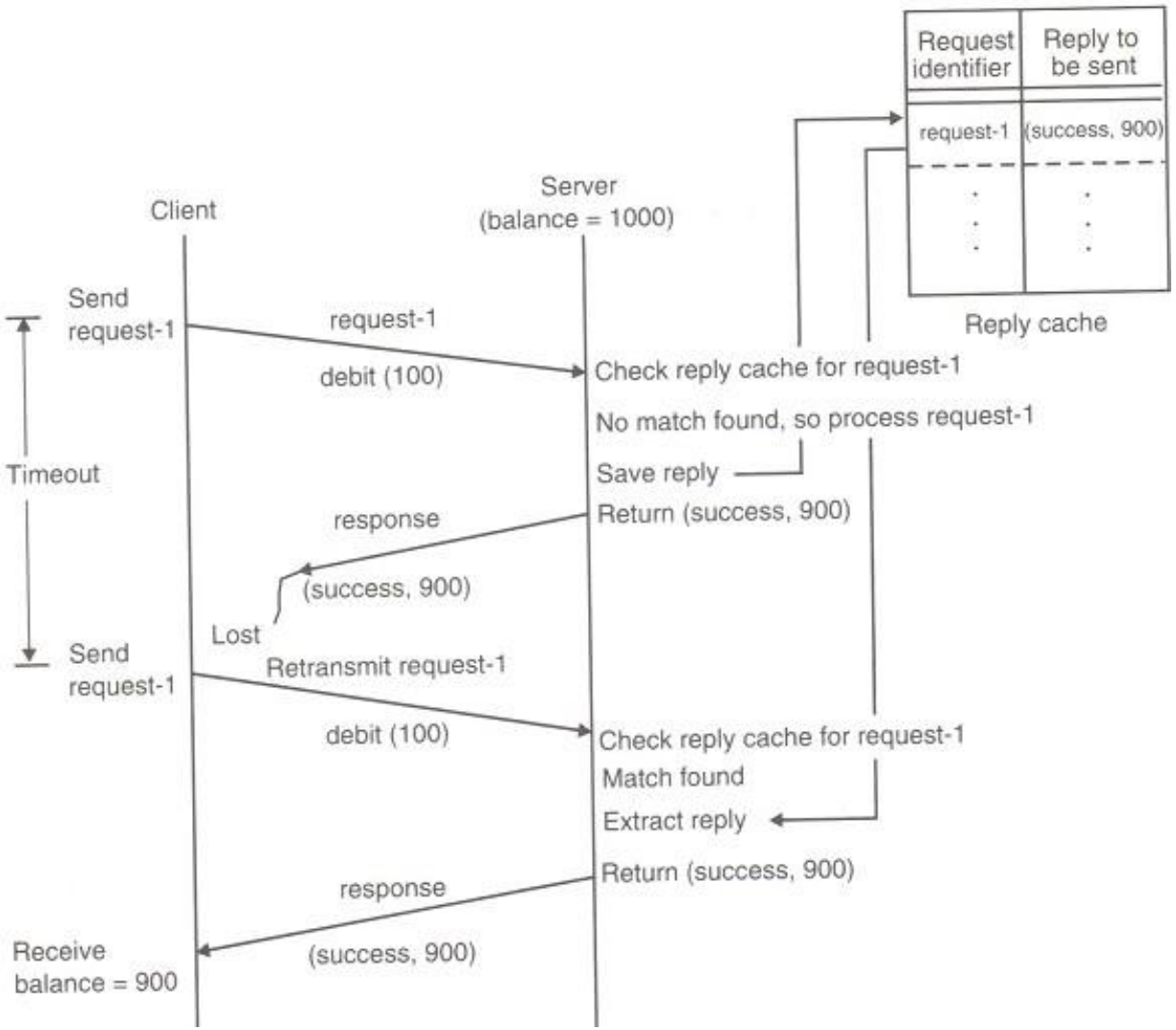
  }

A Non-Idempotent Routine



Implementation of Idempotency

- How to implement Idempotency?
  - Adding sequence number with the request message
  - Introduction of ‘Reply cache’


Request identifier
Reply to be sent
request-1
(success, 900)
Reply cache
Client
Server
(balance = 1000)
Send request-1
request-1
debit (100)
Check reply cache for request-1
No match found, so process request-1
Save reply
Timeout
Return (success, 900)
response
(success, 900)
Lost
Send request-1
Retransmit request-1
debit (100)
Check reply cache for request-1
Match found
Extract reply
Return (success, 900)
response
(success, 900)
Receive balance = 900

Remote Procedure Call (RPC)

- The IPC part of a distributed application can be adequately and efficiently handled by using an IPC protocol based on message passing system.
- However, an independently developed IPC protocol is tailored specifically to one application and does not provide a foundation on which to build a variety of distributed applications.
- Therefore, a need was felt for a general IPC protocol that can be used for designing several distributed applications.
- The RPC facility emerged out of this need.

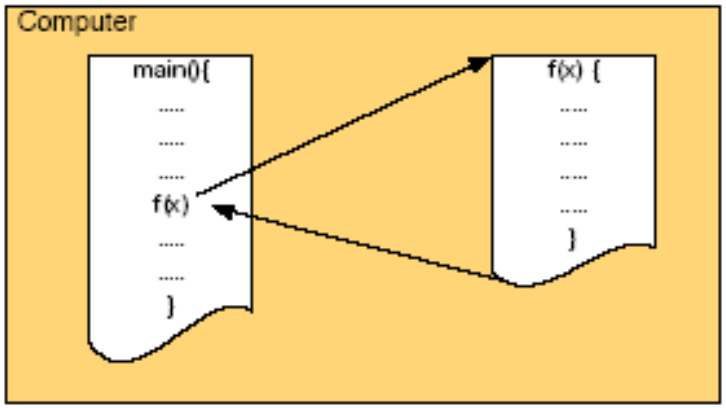
Remote Procedure Call

- While the RPC is not the universal panacea for all types of distributed applications but for a fairly large number of distributed applications.
- The RPC has become a widely accepted IPC mechanism in DS. It features –
  - Simple call syntax.
  - Familiar semantics.
  - Specification of **a well defined interface**.
  - Ease of use.
  - Generality. "In single-machine computations procedure calls are often the most important mechanism for communication between the parts of the algorithm" [Birrell and Nelson].
  - Its efficiency

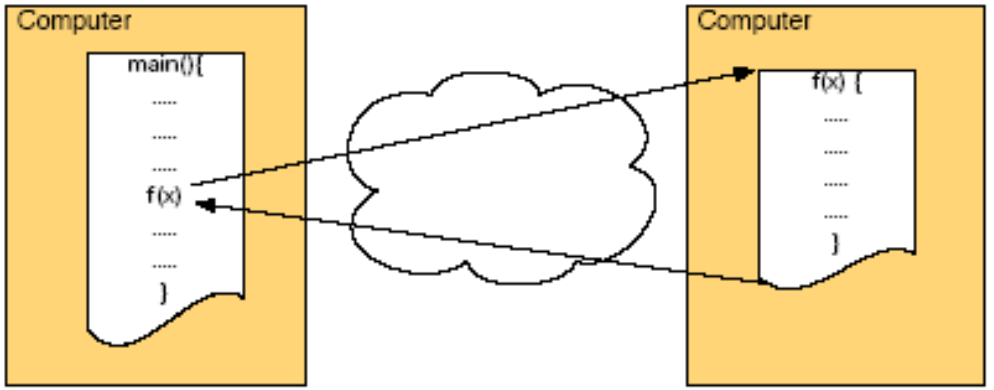
RPC model

- RPC model is similar to “Procedure call” model.
- Procedure call is same as function call or subroutine call
- Local Procedure Call - The caller and the callee are within a single process on a given host.
- Remote Procedure Call (RPC) - A process on the local system invokes a procedure on a remote system. The reason we call this a “procedure call” is because the intent is to make it appear to the programmer that a local procedure call is taking place.

Local and Remote Procedure Call

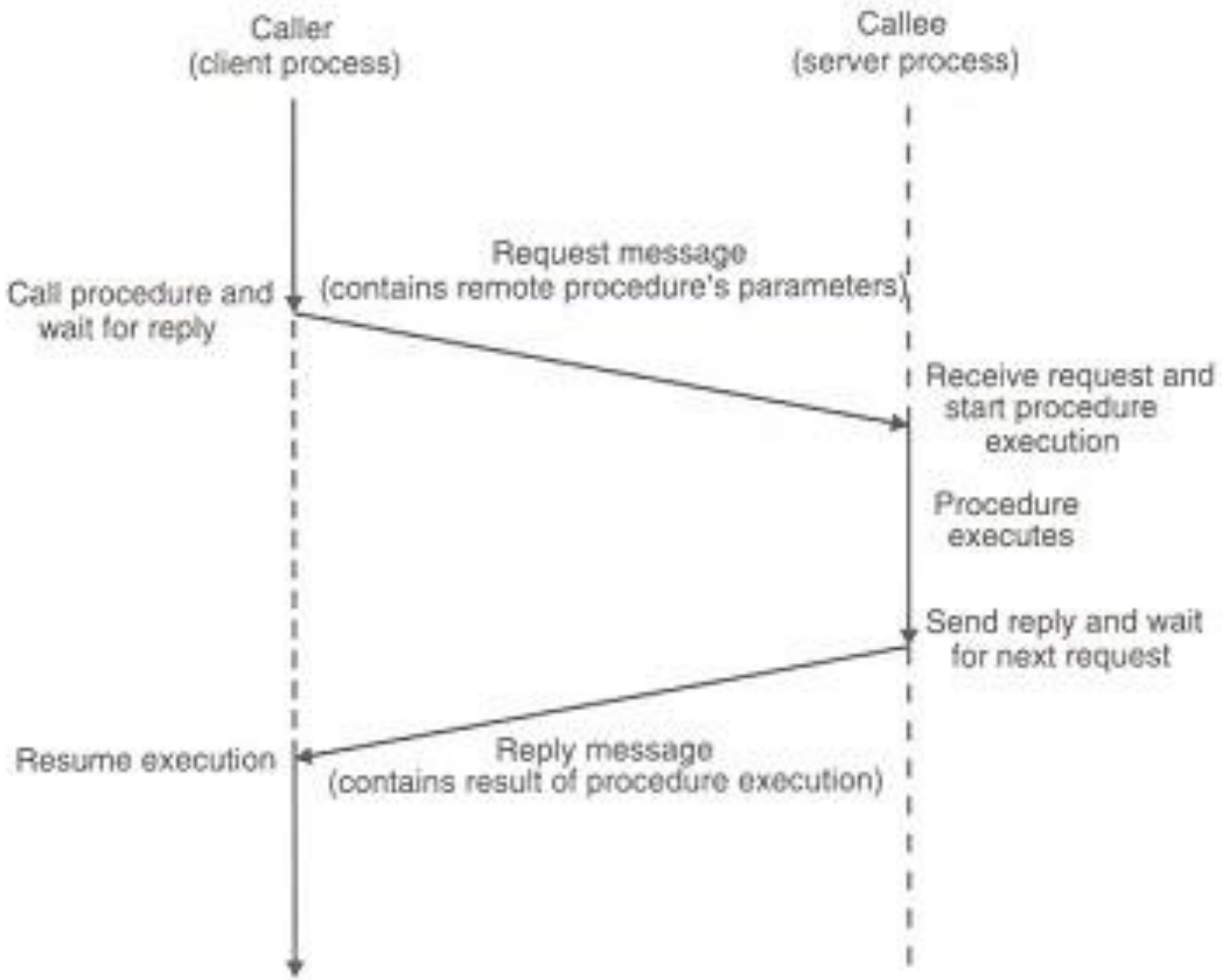


Local Procedure Call



Remote Procedure Call

A Typical Model of RPC



Design Issues for RPC

- The style of programming promoted by RPC – programming with interfaces.
- The call semantics associated with RPC.
- The key issue of transparency and how it relates to remote procedure calls.

Call Semantics

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

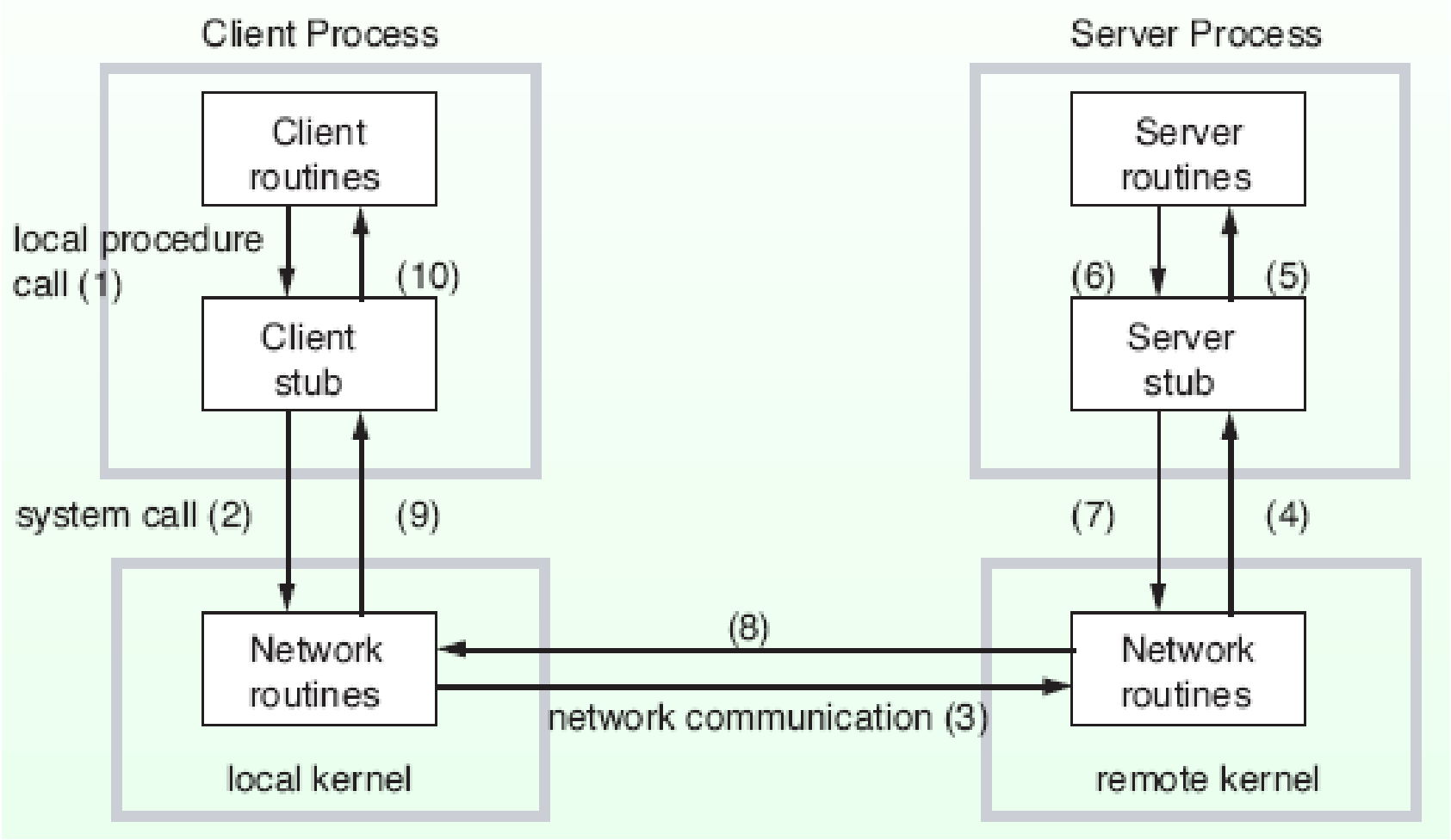
Transparency

- The network needs to be made invisible, so that everything looks just like ordinary procedure calls.
- All networking should be done by the RPC implementation, such as connecting to the remote machine.

Implementing RPC mechanism

- To achieve the goal of *semantic transparency*, the implementation of an RPC mechanism is based on the concept of *stubs*
- *Stubs* provide a perfectly normal (local) procedure call abstraction
- To hide the distance and functional details of underlying network, an RPC communication package (known as *RPCRuntime)* is used.
- Thus RPC implementation involves five elements-
  - The client
  - The client stub
  - The RPCRuntime
  - The server stub
  - The server

RPC in Detail



Stubs

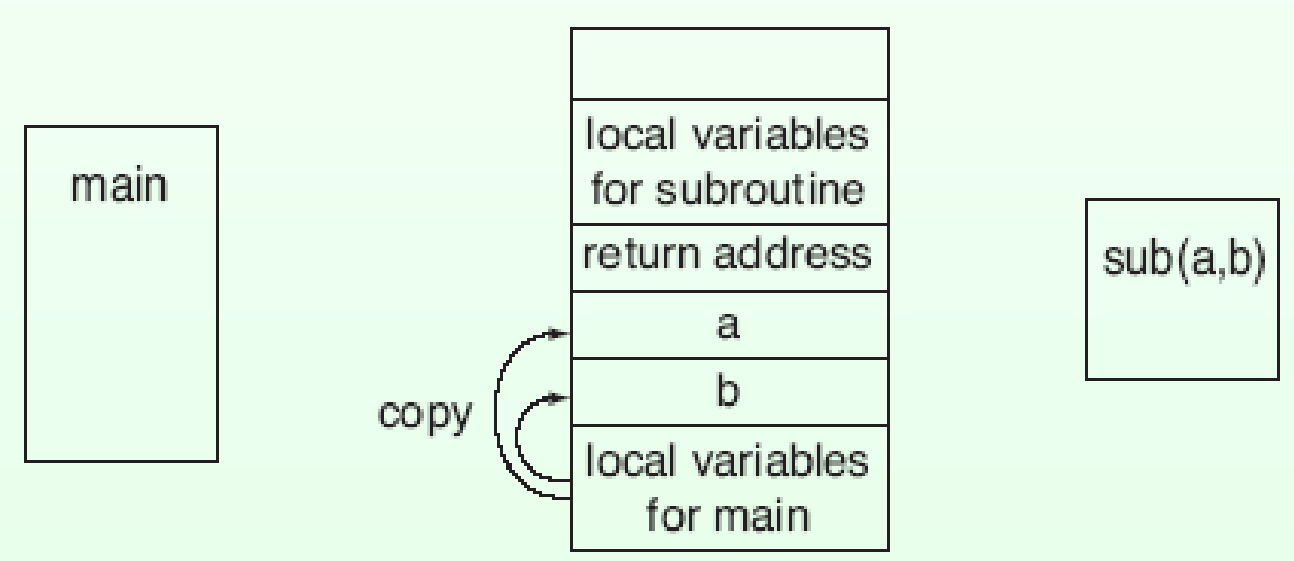
- Client and server stubs are generated from interface definition of server routines by development tools.
- Interface definition is similar to class definition in C ++ and Java.

Parameter Passing Mechanisms

- When a procedure is called, parameters are passed to the procedure as the arguments. There are three methods to pass the parameters.
  - call-by-value
  - call-by-reference
  - call-by-copy/restore

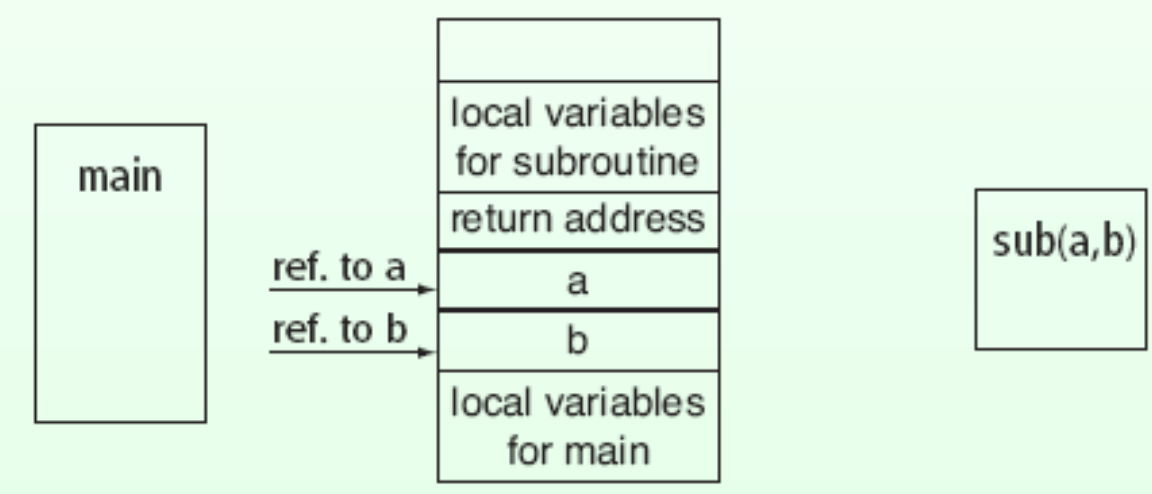
Call by Value

- The values of the arguments are copied to the stack and passed to the procedure.
- The called procedure may modify these, but the modifications do not affect the original value at the calling side.



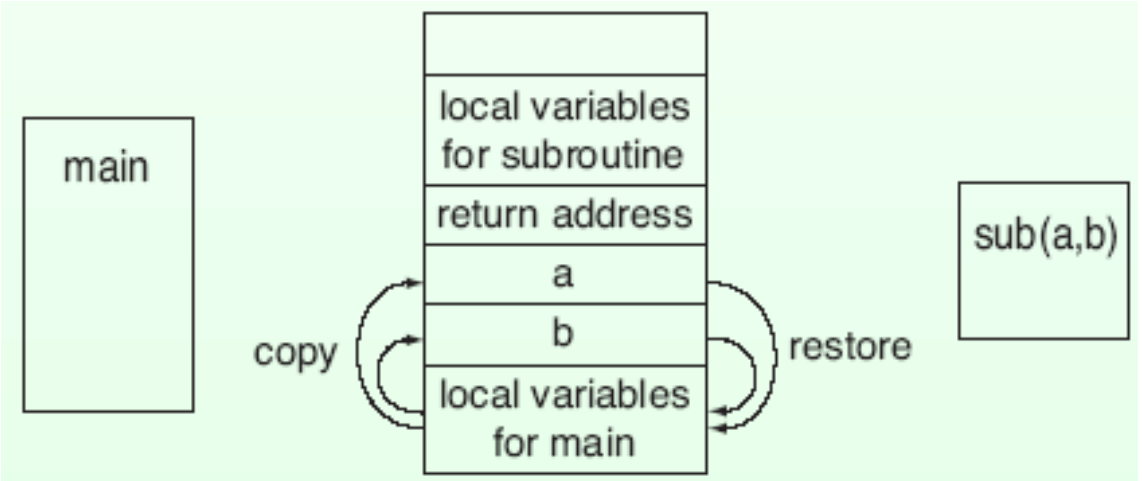
Call-by-Reference

- The memory addresses of the variables corresponding to the arguments are put into the stack and passed to the procedure.
- Since these are memory addresses, the original values at the calling side are changed if modified by the called procedure.



Call-by-Copy/Restore

- The values of the arguments are copied to the stack and passed to the procedure.
- When the processing of the procedure completes, the values are copied back to the original values at the calling side.
- If parameter values are changed in the subprogram, the values in the calling program are also affected.



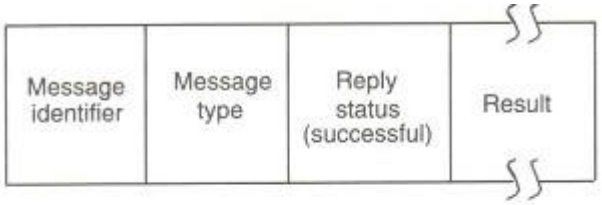
Parameter Passing in RPC

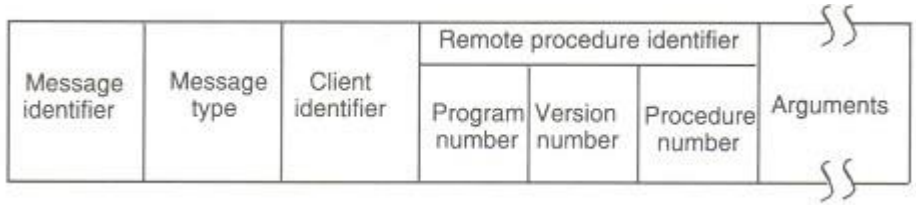
- Which parameter passing mechanisms are possible?
  - It is possible to implement all of the three mechanisms if you wish. Usually call-by-value and call-by-copy/ restore are used.
  - Call-by-reference is difficult to implement. All data which may be referenced must be copied to the remote host and the reference to the copied data is used.

- We need to convert the values of the arguments into a standard format to transmit over the network
  - Different machines use different character codes.
  - Representation of numbers may differ from machine to machine.

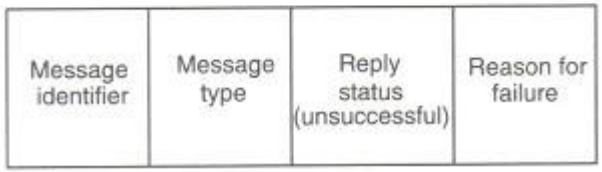
RPC Messages

- Generally two types of messages
  - Call messages
  - Reply messages





A typical RPC Call message format

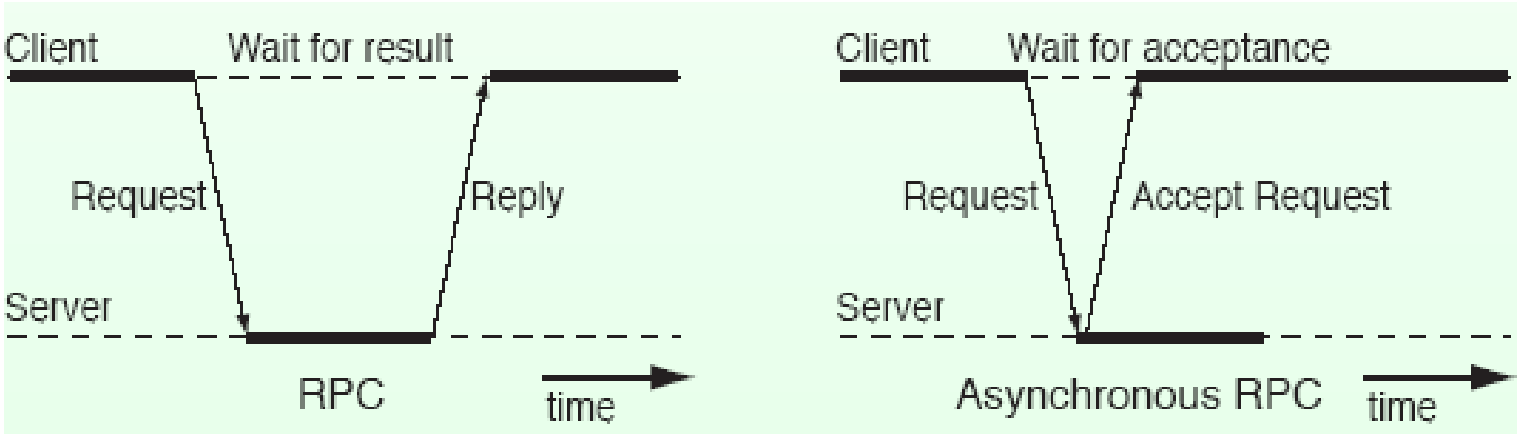


A typical RPC Reply message format (successful and unsuccessful)

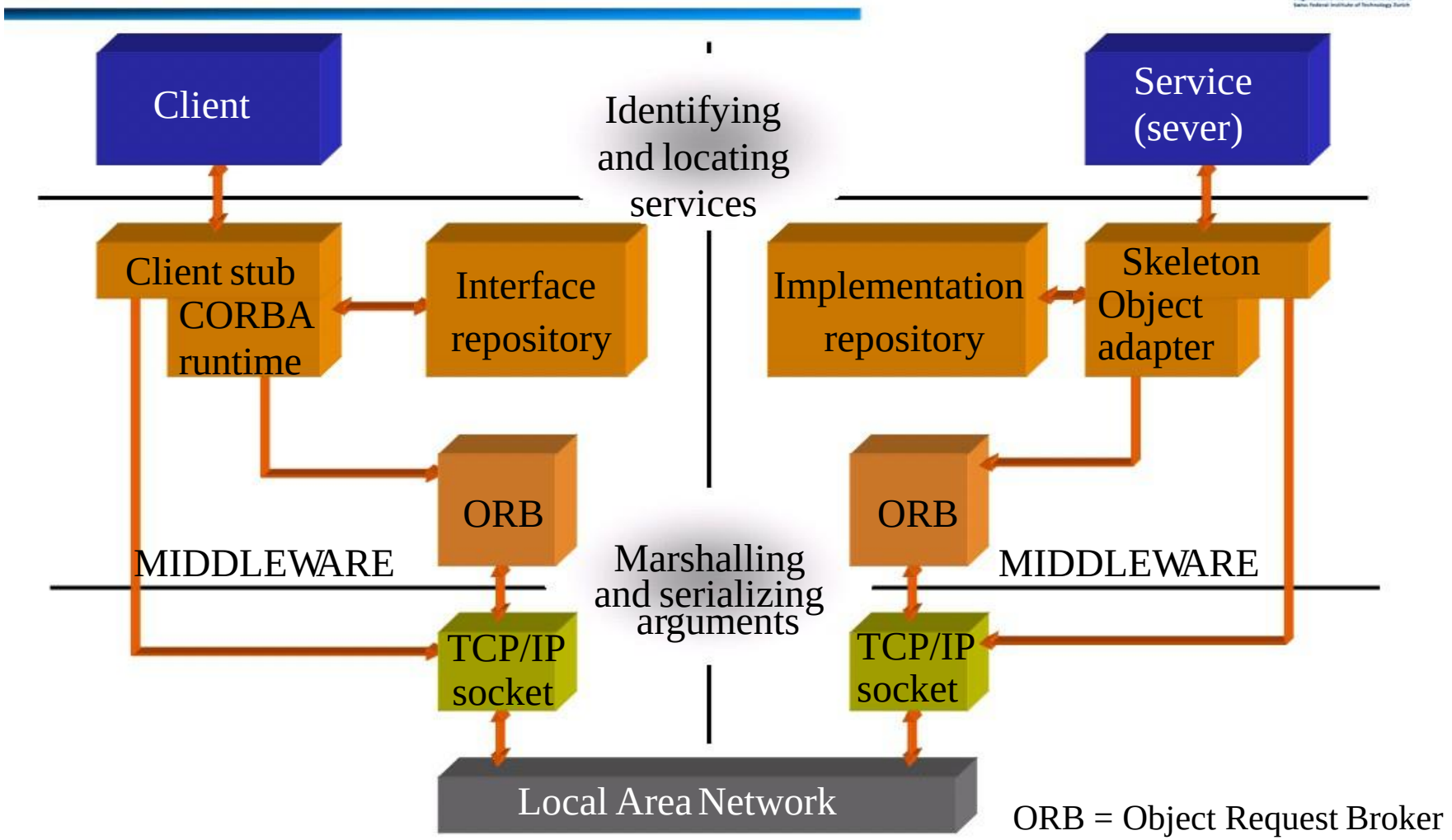
Variations of RPC

Asynchronous RPC

- RPC (When a client requests a remote procedure, the client wait until a reply comes back in RPC.
- If no result is to be returned, unnecessary wait time overhead.
- In asynchronous RPC, the server immediately sends accept message when it receives a request.



RPC between Objects – the CORBA Example



ORB = Object Request Broker

Summary

- What is the purpose of IPC?
  - information sharing among two or more processes
- Differences between Synchronous and Asynchronous Communications?
  - When both the send and receive primitives of a communication between two processes use blocking semantics, the communication is said to be Synchronous; otherwise it is asynchronous
- List the Types of Failure in IPC

  Loss of request message, Loss of response message, Unsuccessful execution of the request
- How to implement Idempotency?
  - Adding sequence number with the request message and Introduction of ‘Reply cache’
- Name an all propose IPC protocol?
  - Remote Procedure Call (RPC)

References

1. Birman, K. P. and Renesse, R. V. Reliable Distributed Computing with the ISIS Toolkit. IEEE Computer Society Press, 1994.
2. Birrell, A. D. and Nelson, B. J. Implementing remote procedure calls. ACM Trans. Comput. Syst. 2(1), 39-59, 1984.
3. Wilbur, S. and Bacarisse, B. Building distributed systems with remote call, Software Engineering Journal, 2(5), 148-159, 1987.
4. R. Gimson. Call buffering service. Technical Report 19, Programming Research Group, Oxford University, Oxford University, Oxford, England, 1985.