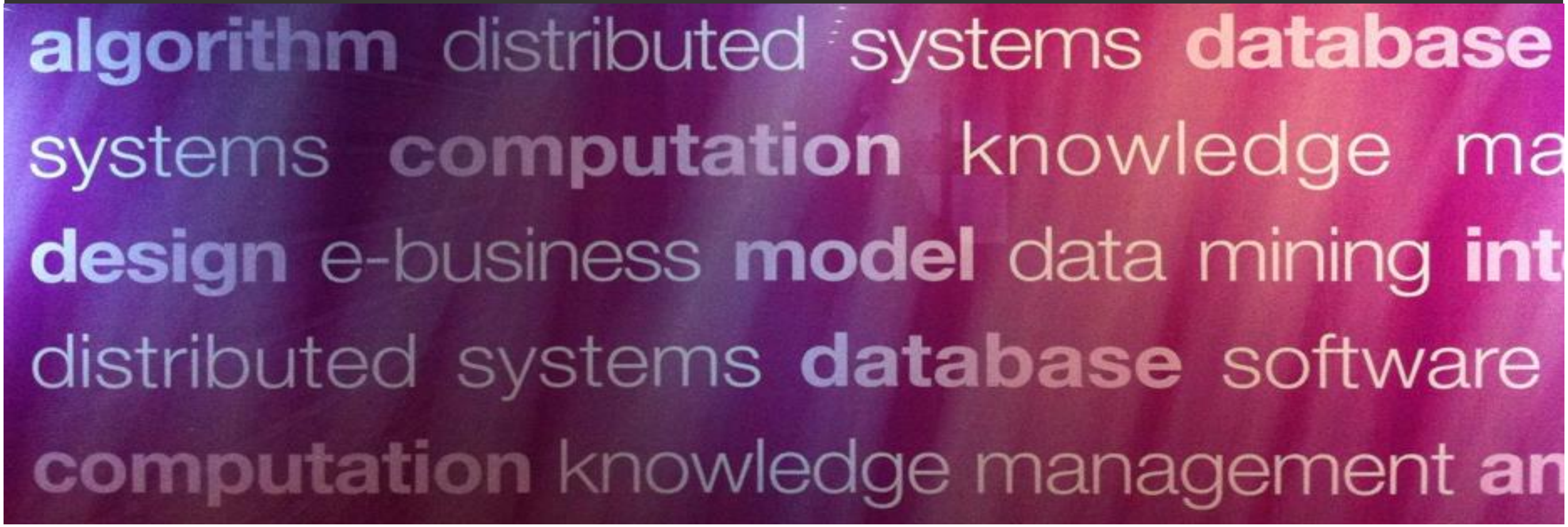
MONASH University

Information Technology

FIT3143 - LECTURE WEEK 4

INTER-PROCESS COMMUNICATIONS & REMOTE PROCEDURE CALL



Overview

1. Inter process communication (IPC)
2. Synchronous and asynchronous communication
3. Remote procedure call

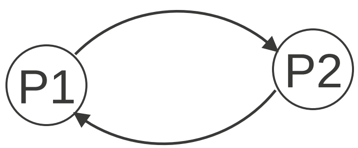
Learning outcome(s) related to this topic

- Compare and contrast different parallel computing architectures, algorithms and communication schemes using research-based knowledge and methods (LO2)

Inter-process Communication (IPC)

- IPC basically requires information sharing among two or more processes. Two basic methods-
  - 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).

Desirable features of a good message passing system

- Simplicity

- Uniform semantics
  - Same primitives for local and remote communication

- Efficiency
  - Reduce the number of message as far as possible
  - Some optimization normally adopted for efficiency include-
    - Avoiding the cost of establishing and terminating connections between the same pair of processes for each and every message exchange between them
    - Minimizing cost of maintaining connections
    - Piggybacking of acknowledgment of previous message with the next message.

Desirable features of a good message passing system

- Reliability
  - Lost and duplicate message handling

- Correctness
  - Atomicity
  - Ordered delivery
  - Survivability

- Flexibility
  - Can drop one or more correctness properties

Desirable features of a good message passing system

- Security
  - Authentication of sender and receiver
  - Encryption of messages

- Portability
  - Message passing system should itself be portable
  - The application written by using primitives of the IPC protocol should be portable.

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

Issues need to be considered for designing an IPC protocol

- Some of the main issues will be,

- Identity related
  - Who is the sender?
  - Who is the receiver?
- Network Topology related
  - 1 receiver or many?
- Flow control related
  - Guaranteed by the receiver?
  - Sender should wait for reply?
- Error control and channel management
  - Node crash……what to do?
  - Receiver not ready….what to do?
  - Several outstanding messages for the receiver

Synchronization in IPC

IPC Synchronization Semantics

Blocking

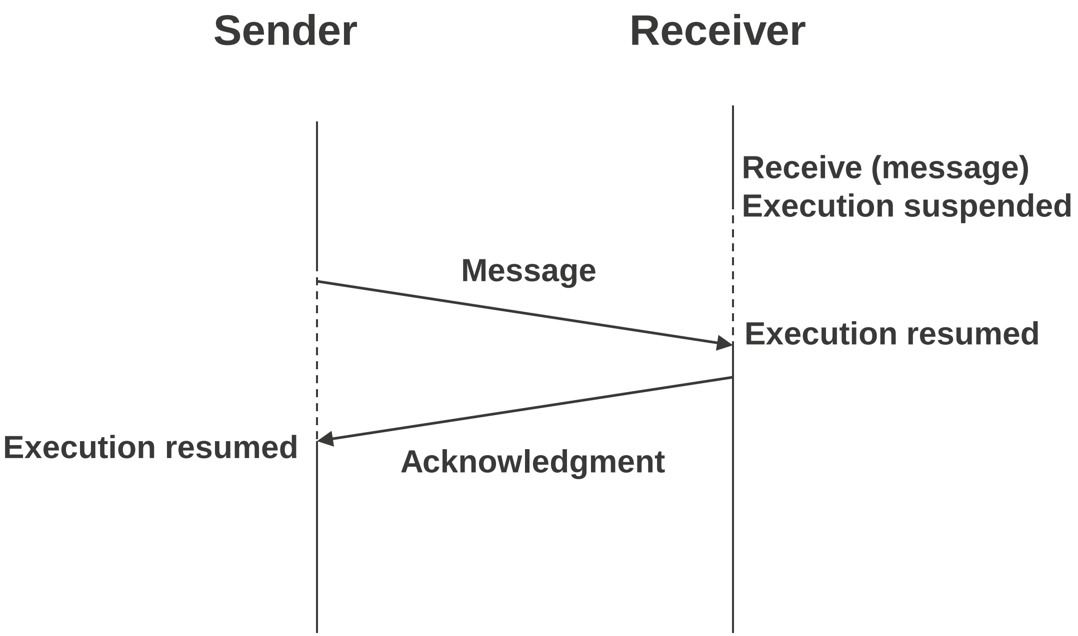
Non Blocking

- 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

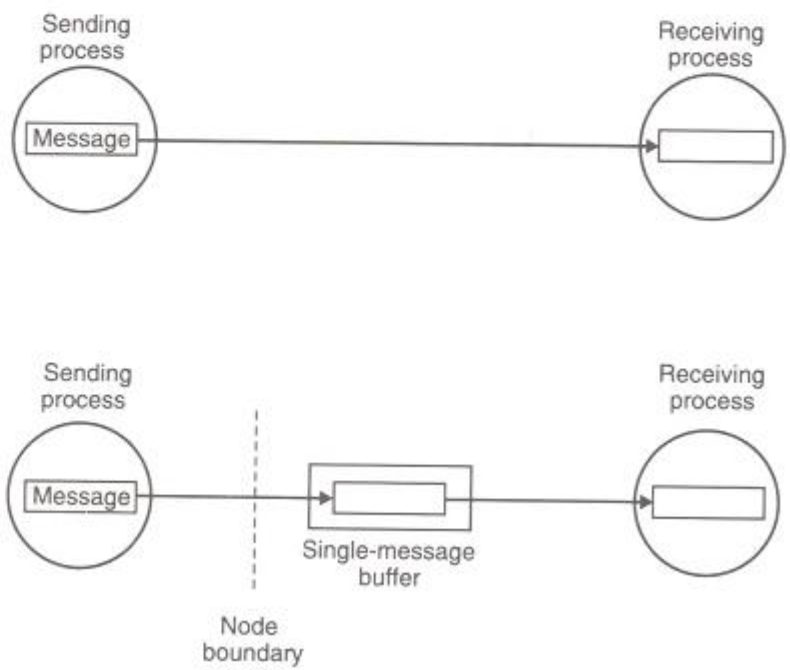
- **Asynchronous**
  – High concurrency
  – More flexible than synchronous
  – Lower deadlock risk than in synchronous communication (but beware)

Buffering

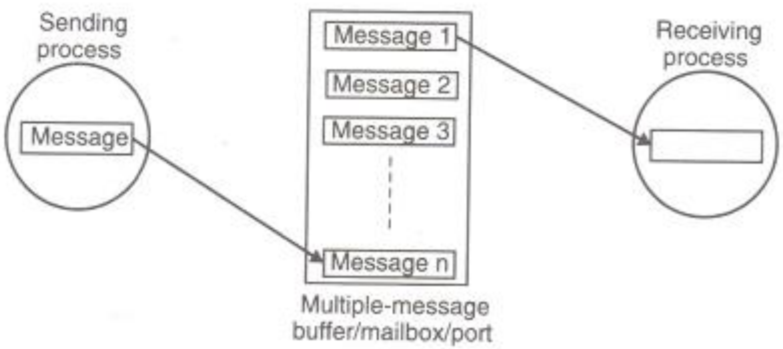
- Synchronous systems
  - Null buffer
  - Single message buffer

- Asynchronous systems
  - Unbounded capacity buffer
  - Finite message (multiple message buffer)

Buffering



Synchronous System

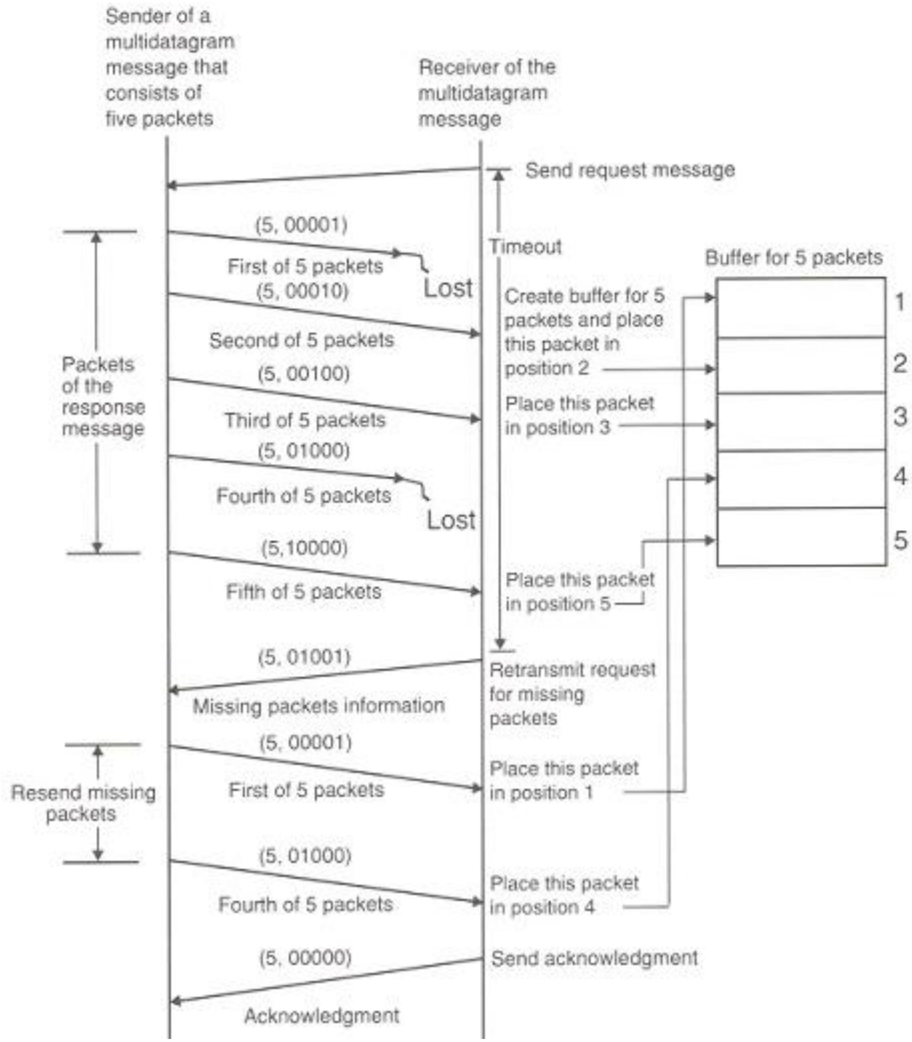


Asynchronous System

Multi-datagram Messages

- Almost all networks have an upper bound on the size of data that can be transmitted at a time. This is known as MTU (maximum transfer unit).
- Thus a message whose size is greater than MTU has to be fragmented- each fragment is sent in a packet. These packets are known as datagram.
- Thus messages may be single-datagram messages or multi-datagram messages
- Assembling and dissembling is the responsibility of message passing system.

Using Bitmap for multi datagrams



Encoding Decoding

Encoding/Decoding is needed if

- Sender and receiver have different architecture

Even for Homogeneous Encoding/Decoding is needed for

✓ Using an absolute pointer

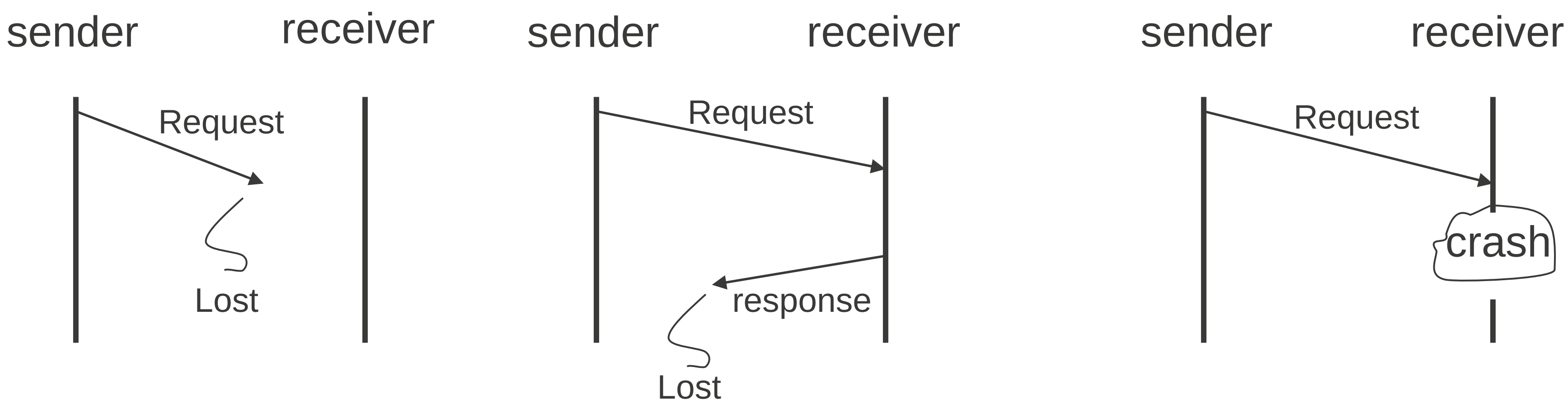
✓ To know which object is stored in where and how much storage it requires

Process Addressing

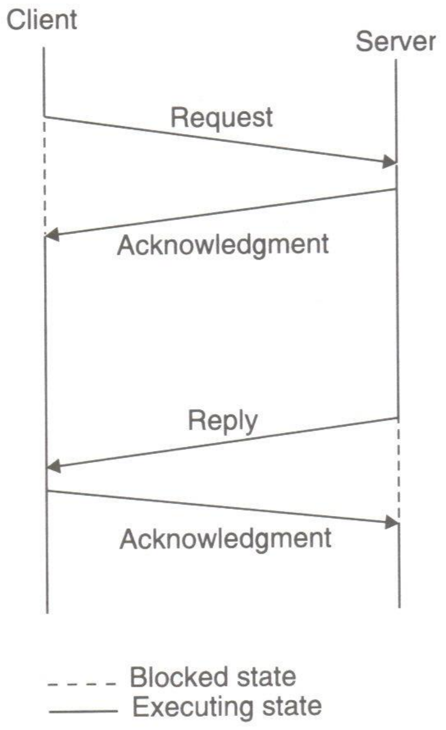
- Explicit Addressing
  - Send(process_ID, message)
  - Receive(process_ID, message)
- Implicit Addressing
  - Send_any(service_ID, message)
  - Receive_any(process_ID, message)

Failure Handling

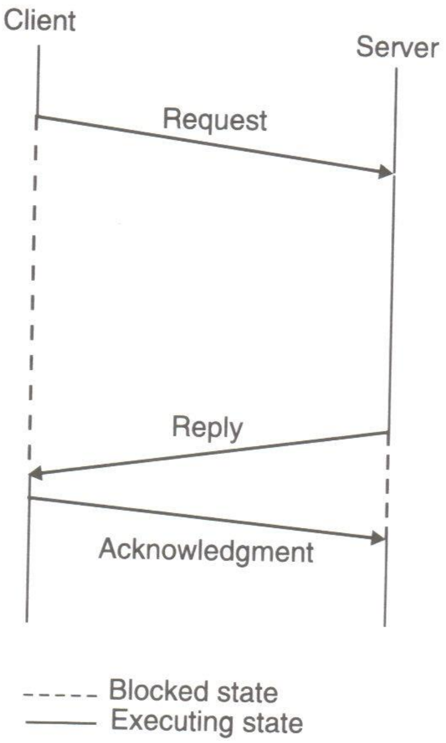
- Failure classification
  1. Loss of request message
  2. Loss of response message
  3. Unsuccessful execution of the request



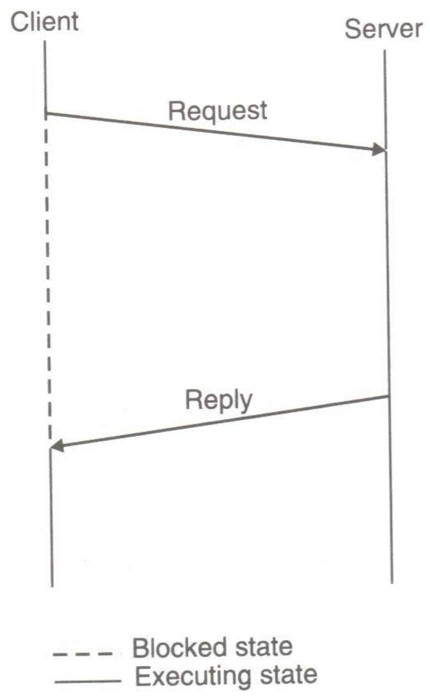
Reliable IPC protocol



4-message protocol

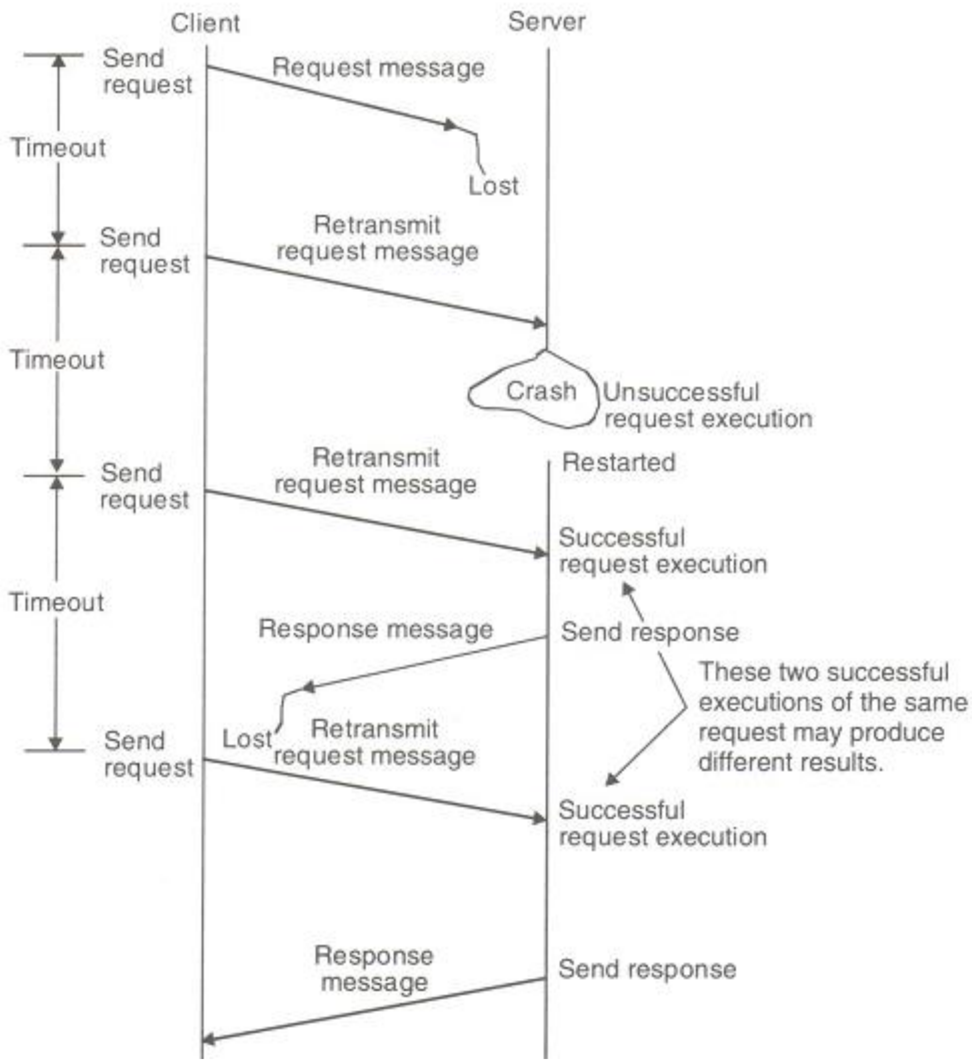


3-message protocol



2-message protocol

An Example of Fault Tolerant System



Idempotency

- An idempotent function will return the same result given the same input, even when executed multiple times. For example, an idempotent function is illustrated in Algorithm 1; a nonidempotent function is illustrated in Algorithm 2.

Algorithm 1 An example of an idempotent function.

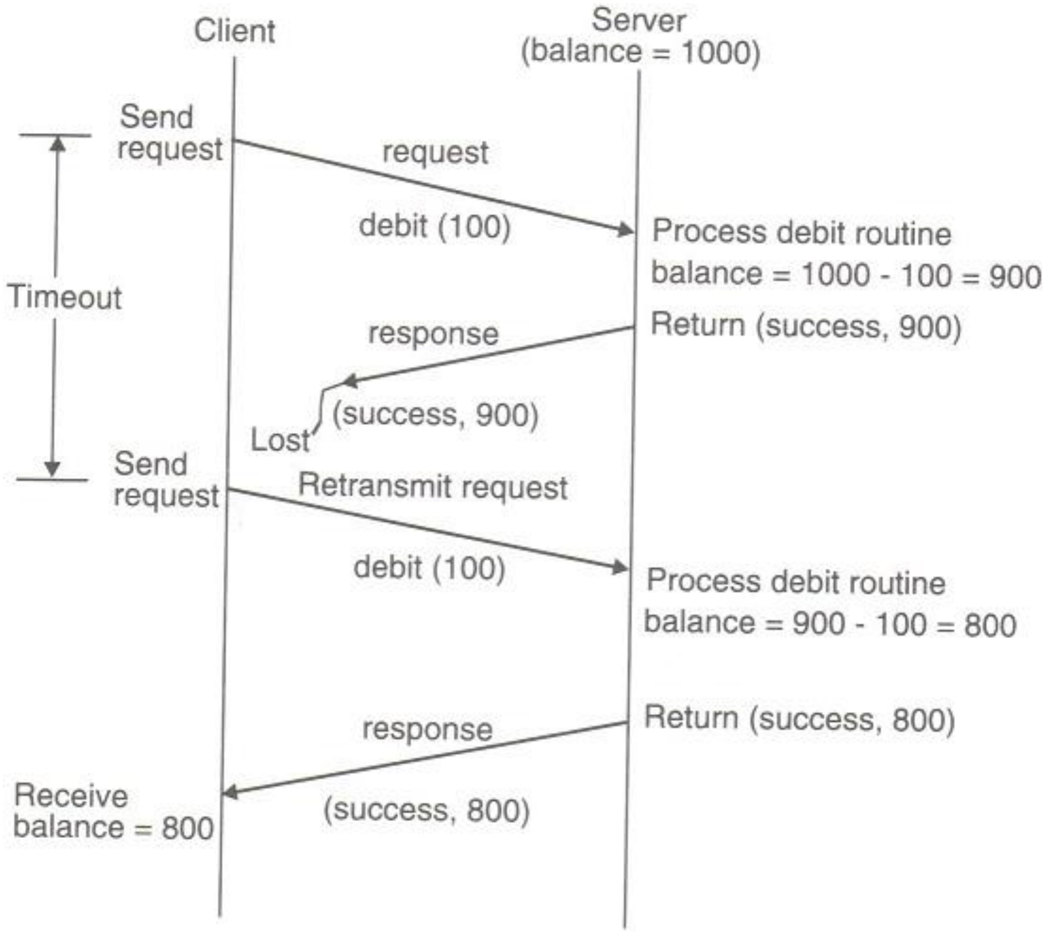
```
function GETSQRT(n)
    return SQRT(n)
end function
```

Algorithm 2 An example of a non-idempotent function. This function will return different values when given the same input. An illustration of the flow can be seen in Figure 2.7*.

```
function DEBIT(amount)
    if balance > amount then
        balance ← balance − amount
        return (”success”, balance)
    else
        return (”failure”, balance)
    end if
end function
```

**Note: Figure 2.7 refers to the following slide*

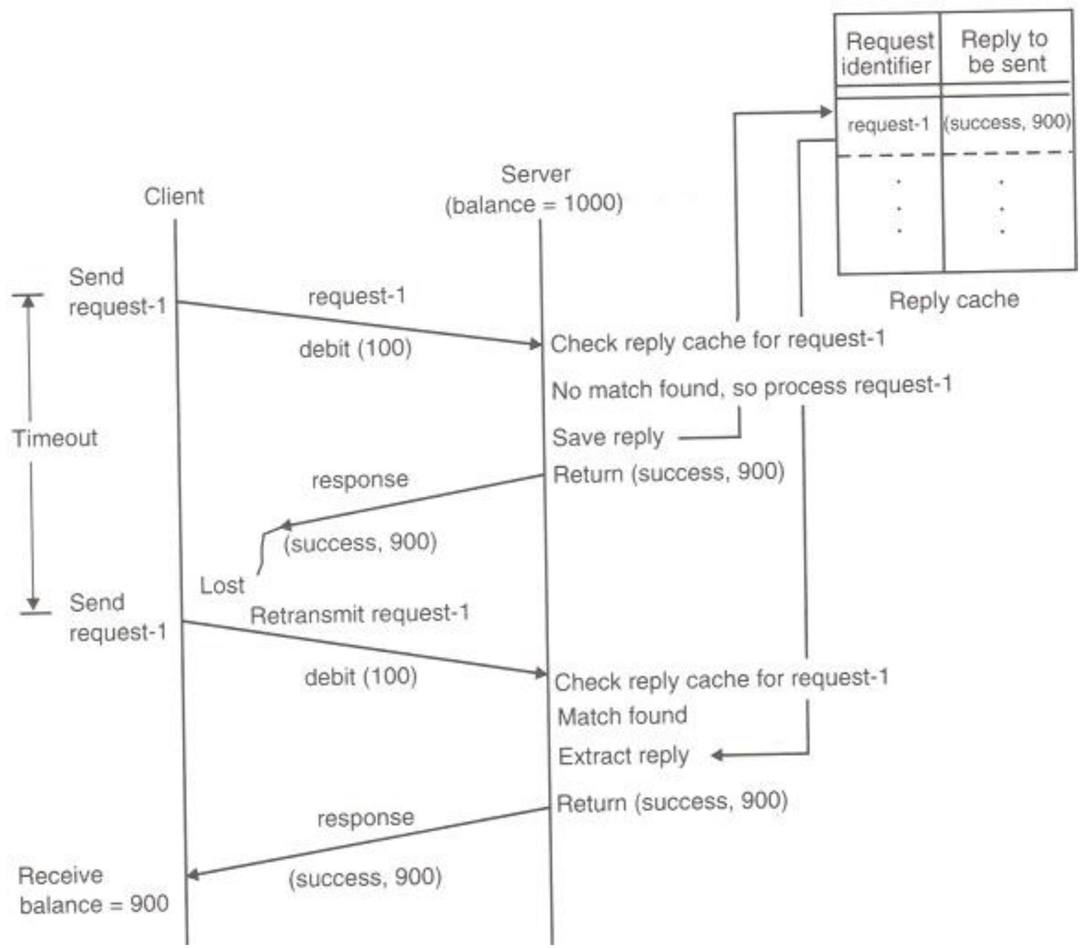
A Non-Idempotent Routine



Implementation of Idempotency

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

Implementation of Idempotency



Group Communication

- Group communication may be
  - One to many
  - Many to one
  - Many to many

- One to many
  - Group management
  - Group addressing
  - Buffered and unbuffered multicast
  - Send-to-all and Bulletin-Board semantics
  - Flexible reliability in multicast communication
  - Atomic multicast

Many to Many communication

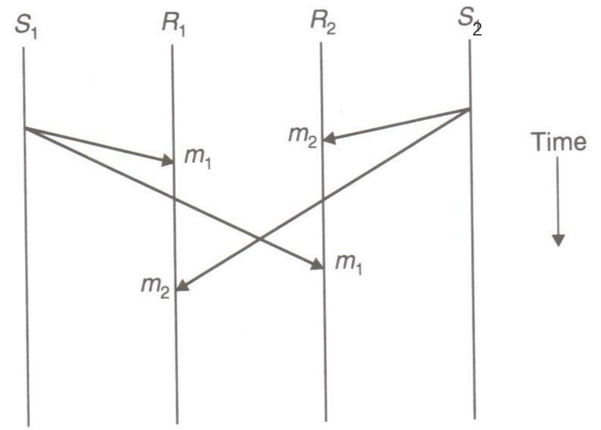
- The issues related to one-to-many and many-to-one communications also applies here

- In addition, *ordered message delivery* is an important issue. This is trivial in one-to-many or many-to-one communications.

- For example, two server processes are maintaining a single salary database. Two client processes send updates for a salary record. What happen if they reach in different order? (will sequencing of messages help in this case?)

Semantics for ordered delivery in many-to-many comm.

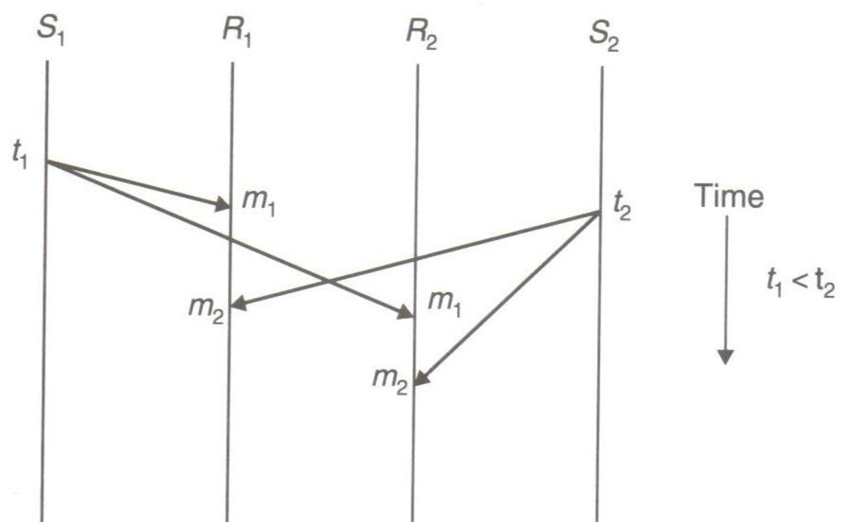
- No-ordering



No ordering constraint

Semantics for ordered delivery in many-to-many comm.

- Absolute ordering
  - All messages are delivered to all receiver processes in the exact order in which they were sent.
  - Using global timestamp as message identifiers with sliding window protocol



Absolute ordering semantic

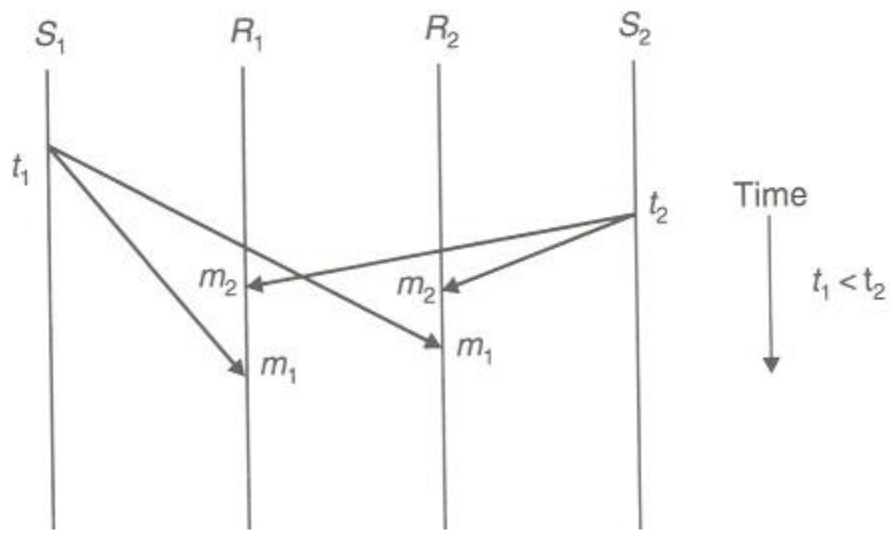
Semantics for ordered delivery in many-to-many comm.

- Consistent ordering
  - All messages are delivered to all receivers in the same order. However, this order may be different from the order in which messages were sent.

  Implementation
  - Make the communication appear as a combination of many-to-one and one-to-many communication [Chang and Maxemchuk]
  - Kernels of sending machines send messages to a single receiver (known as *sequencer*) that assigns a sequence number to each message and then multicast it.
  - Subject to single point of failure and hence has poor reliability.

  - A distributed algorithm - *ABCAST* in *ISIS* system [Birman and Renesse] (self study)

Semantics for ordered delivery in many-to-many comm.



Consistent ordering semantic

Semantics for ordered delivery in many-to-many comm.

- Causal ordering
  - **If the event of sending one message is causally related to the event of sending another message, the two messages are delivered to all receivers in correct order.**
  - **Two message sending events are said to be causally related if they are corelated by the *happened-before* relation.**

[The expression $a \rightarrow b$ is read "a happens before b" and means that all processes agree that first event a occurs, then afterward, event b occurs. The happens-before relation can be observed directly in two situations:

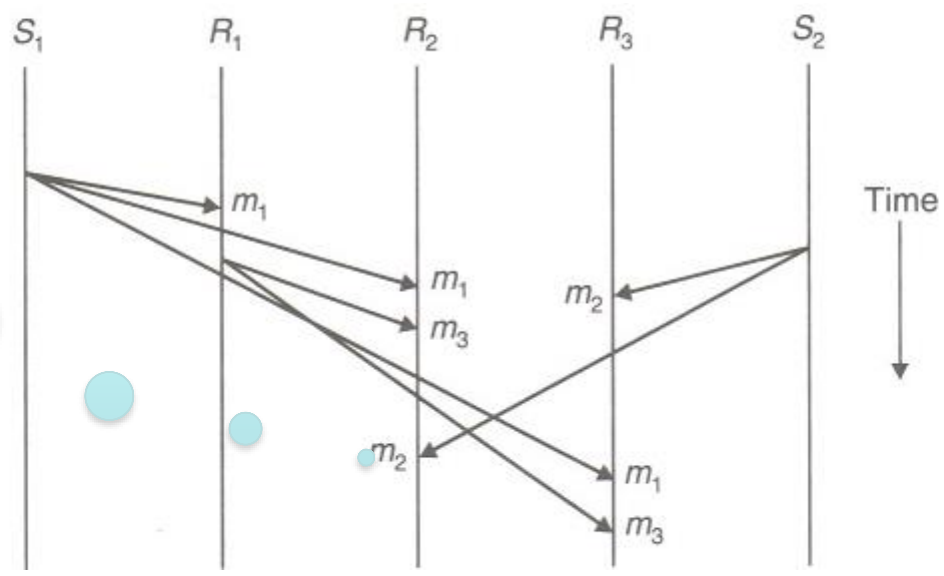
1. If a and b are events in the same process, and a occurs before b, then $a \rightarrow b$ is true.
2. If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then $a \rightarrow b$ is also true.

Happens-before is a transitive relation, so if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.]

Semantics for ordered delivery in many-to-many comm.

- One example of implementing Causal consistency is *CBCAST* in *ISIS* system [Birman et al].

3rd round
m2 is
held back



Causal ordering semantic

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

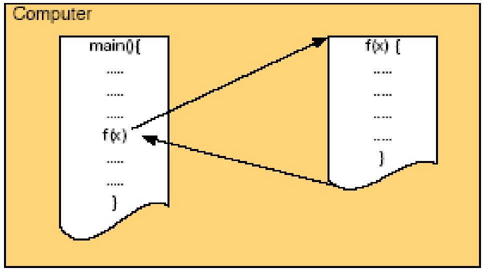
- While the RPC is not the universal panacea for all types of distributed applications but for a fairly large number of distributed applications.

- The RPC has become a widely accepted IPC mechanism in DS. Its 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

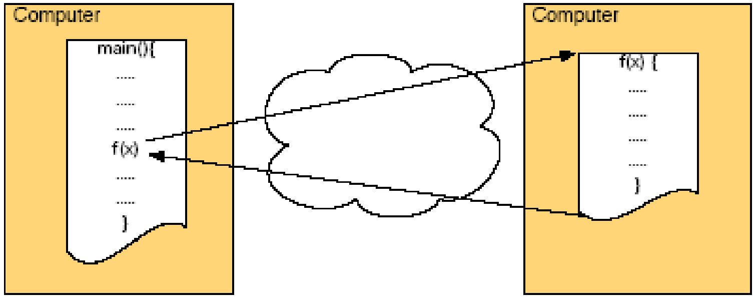
RPC model

- 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.

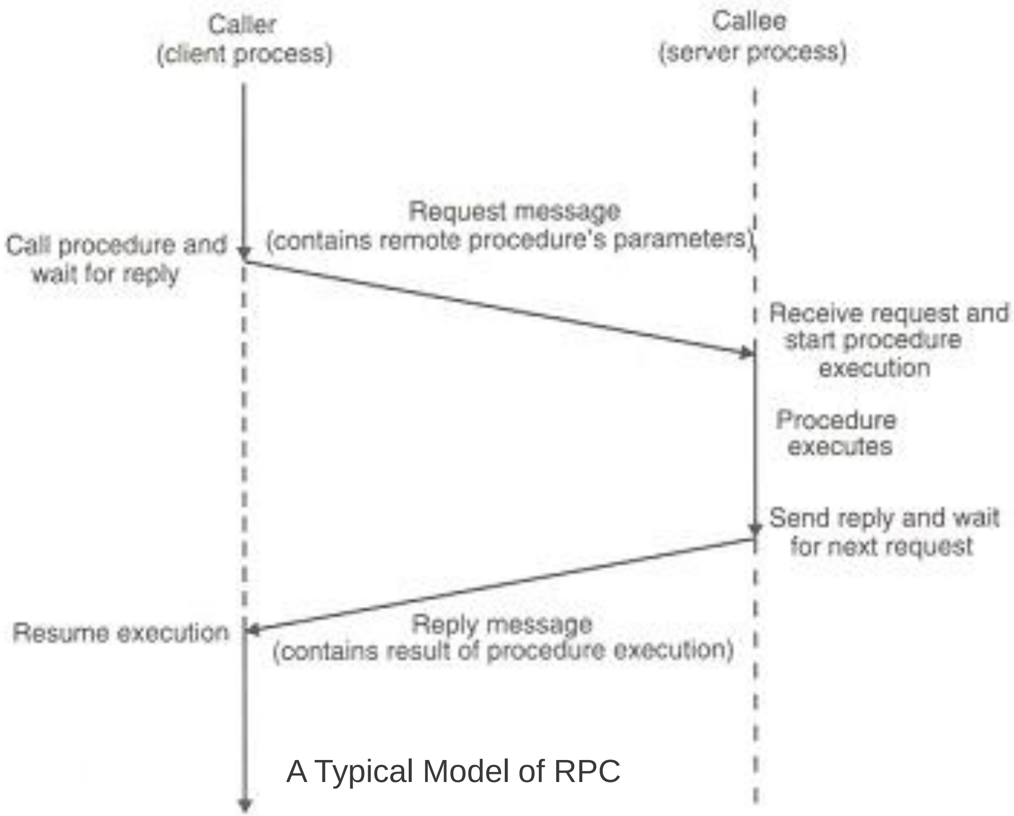
Local and Remote Procedure Call



Local Procedure Call



Remote Procedure Call

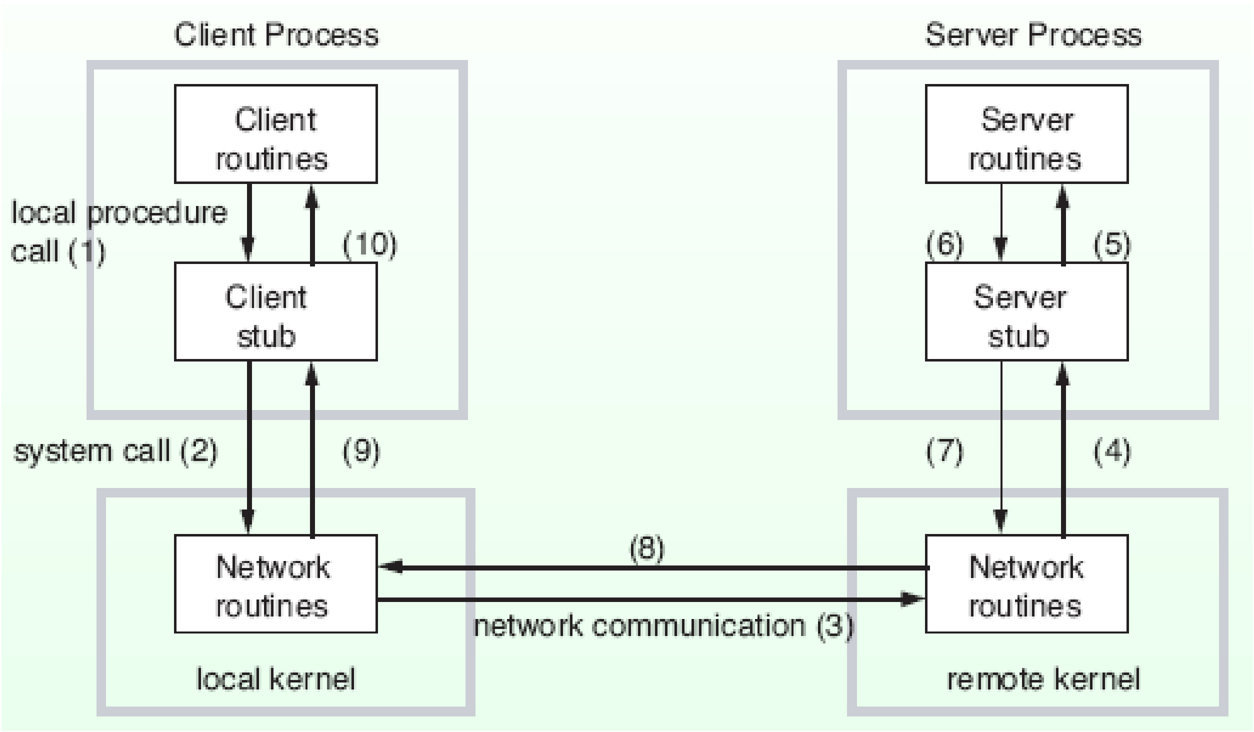


A Typical Model of RPC

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.

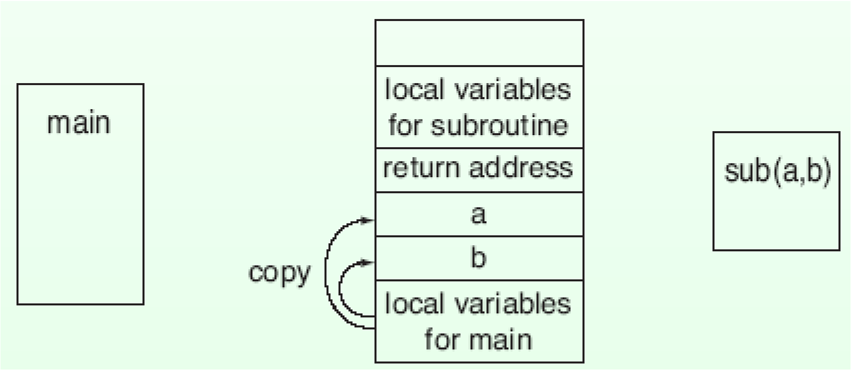
- Interface definition is similar to class definition in C++ and Java.

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

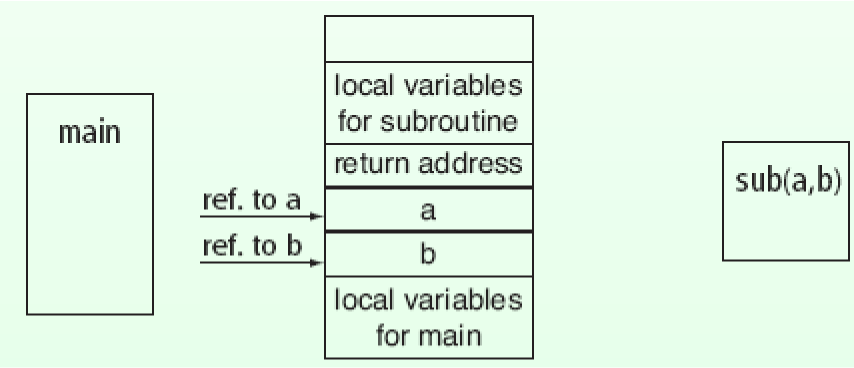
- The values of the arguments are copied to the stack and passed to the procedure.
- The called procedure may modify these, but the modifications do not affect the original value at the calling side.



Call-by-Reference

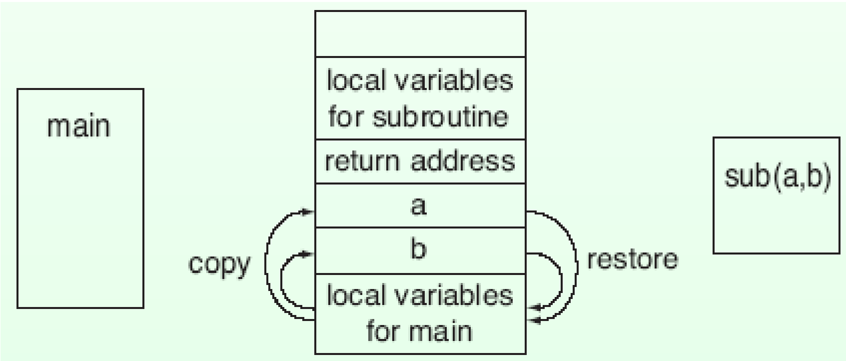
- The memory addresses of the variables corresponding to the arguments are put into the stack and passed to the procedure.

- 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.

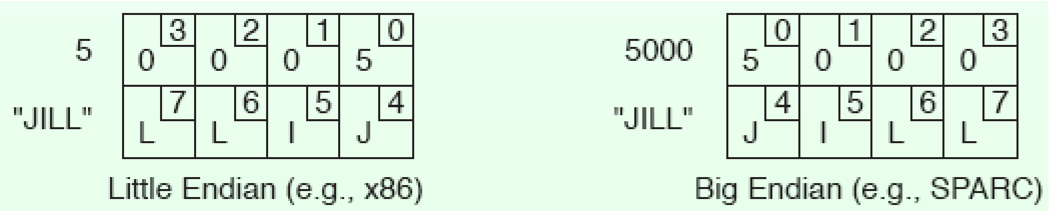


Parameter Passing in RPC

- 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.
- Do we need to convert the values of the arguments into a standard format to transmit over the network?

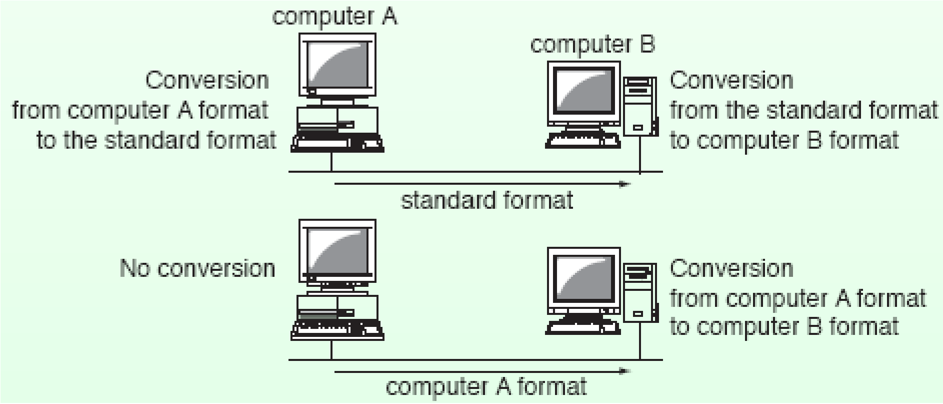
Parameter Passing in RPC

- Reasons to convert the values of the arguments into a standard format to transmit over the network
  - Different machines use different character codes. E.g., IBM main frames use EBCDIC, while PCs use ASCII.
  - Representation of numbers may differ from machine to machine.
  - Big endian and little endian



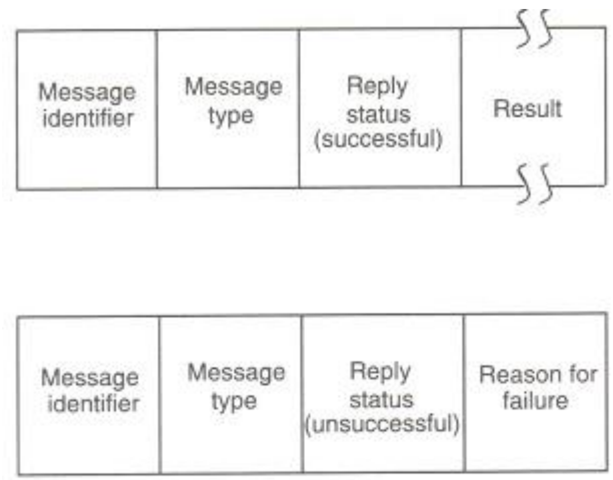
Parameter Passing in RPC

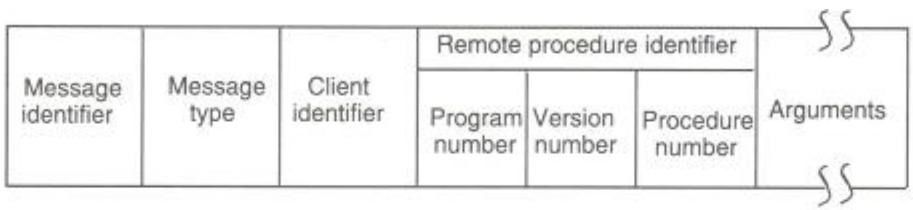
- If a standard format is not used, two message conversions are necessary.
    - If format information is attached to the message, only one conversion at the receiver will suffice.
    - However, the receiver must be able to handle many different formats.



RPC Messages

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





A typical RPC Call message format

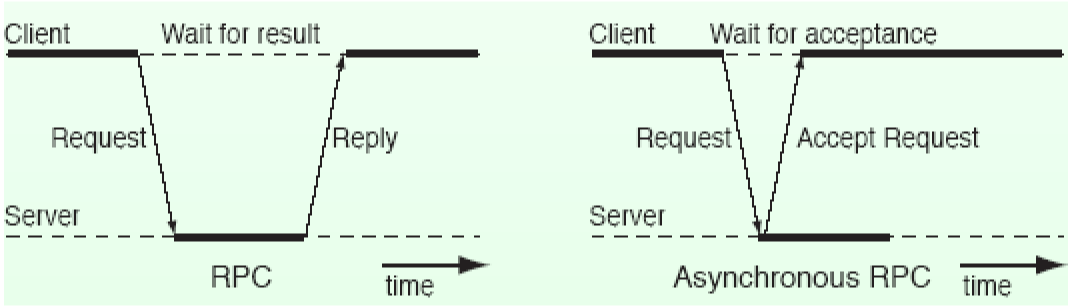
A typical RPC Reply message format

(successful and unsuccessful)

Variations of RPC

Asynchronous RPC

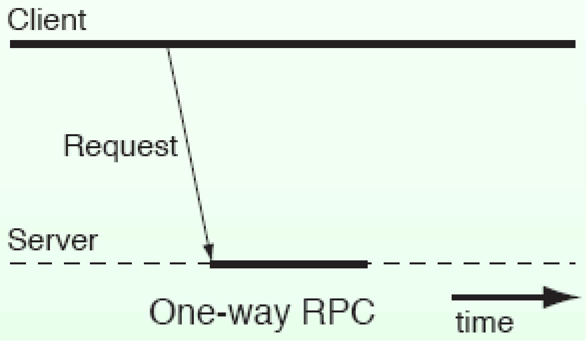
- RPC (When a client requests a remote procedure, the client wait until a reply comes back in RPC.
- If no result is to be returned, unnecessary wait time overhead.
- In asynchronous RPC, the server immediately sends accept message when it receives a request.



Call-Back RPC

One-way RPC

- In one-way RPC, the client immediately continues after sending the request to the server.



Some special types of RPC

- Callback RPC
- Broadcast RPC
- Batch-mode RPC
- Lightweight RPC

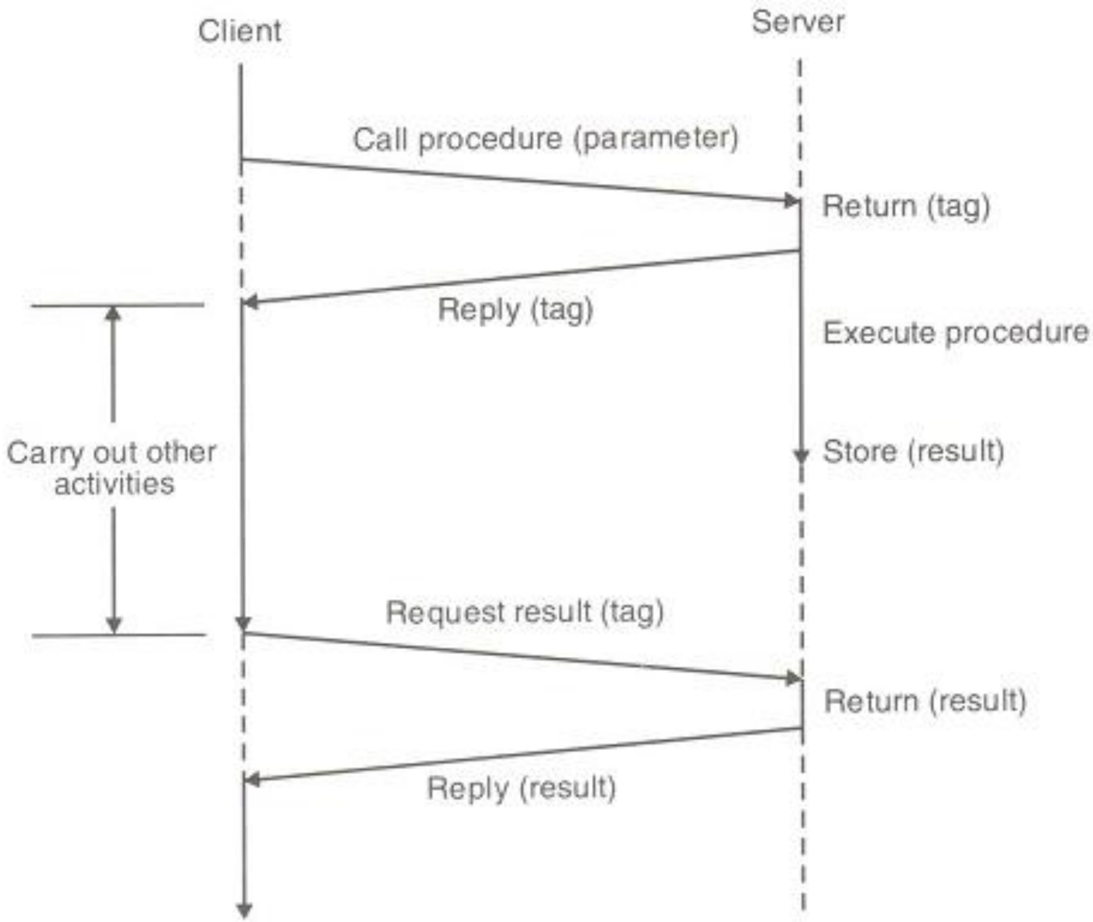
Optimizations for better Performance

- In Six Different Ways
  - Concurrent access to multiple servers
  - Serving multiple requests simultaneously
  - Reducing per-call workload of servers
  - Reply caching of idempotent remote procedures
  - Proper selection of timeout values
  - Proper design of RPC protocol specification

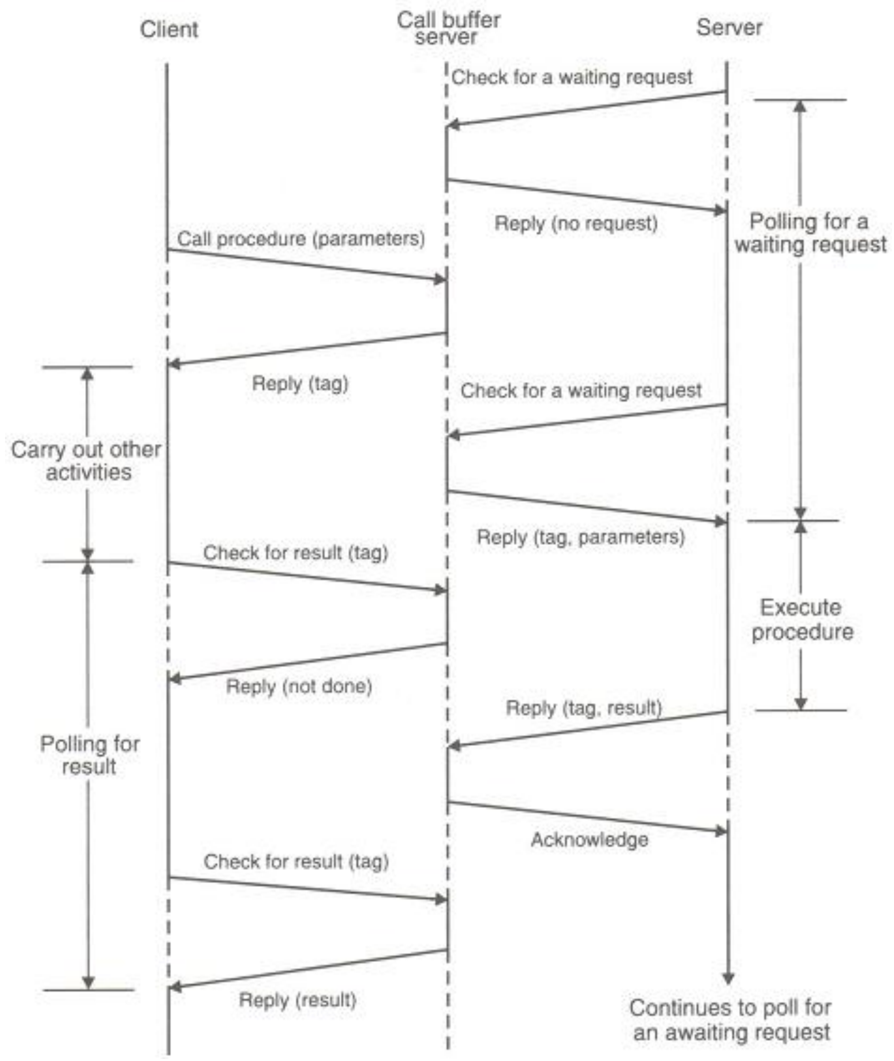
Concurrent Access to Multiple Server

- One of the following three may be adopted:
  - Threads
    - Use of Threads in the implementation of a client process where each thread can independently make remote procedure calls to different servers.
    - Addressing in underlying protocol should be rich enough to provide correct routing of responses.
  - Early reply approach [Wilbur and Bacarisse]
    - A call is split into two separate RPC calls- one passing parameters and other requesting the result
    - Server must hold the result causing congestion or unnecessary overhead.
  - Call buffering approach [Gimson]
    - Clients and servers do not interact directly but via a call buffer server
    - A variant of this approach was implemented in MIT (Mercury Communication System)

Early Reply Approach



Call Buffering Approach



Serving Multiple Requests Simultaneously

- Following types of delays are common-
  – A server, during the course of a call execution, may wait for a shared resource
  – A server calls a remote function that involves computation or transmission delays

- So the server may accept and process other requests while waiting to complete a request.

- Multiple-threaded server may be a solution.

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’
- Three main types of Group Communications?
  - One to many, Many to one, Many to many
- One of the greatest challenges in Many to Many?
    - Ordered Delivery
- Name an all propose IPC protocol?
  - Remote Procedure Call (RPC)
- Name a few ways to optimise RPC?
  - Concurrent Access to Multiple Servers, Serving Muliple Requests Concurrently, Reducing Call Workload per Server
- Three different techniques for implementing Concurrent Access to Multiple Servers?
  - Threads, Early Reply, Call Buffering

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.