Message Passing

Interprocess Communication for Information sharing

Introduction

- 1. Original sharing, or Shared-data approach

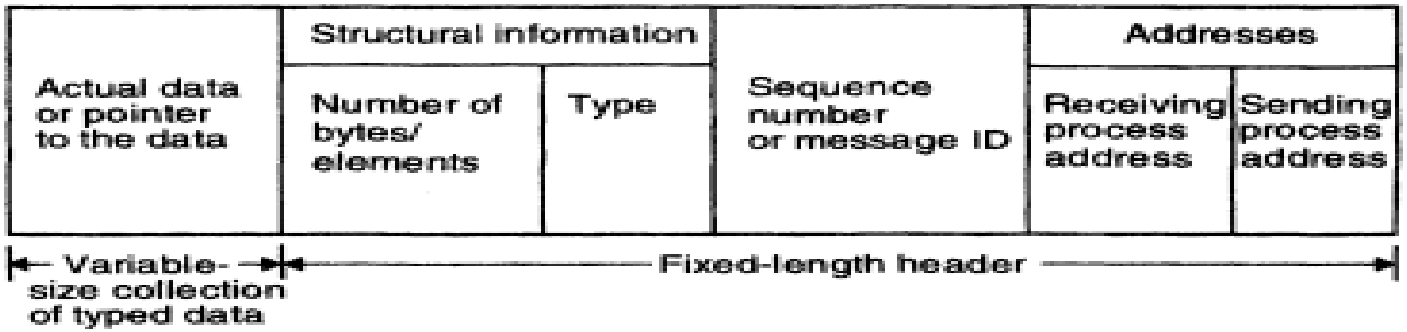
- 2. Copy sharing, or Message-passing approach

Desirable Features of a Good Message-Passing Systems-I

- 1. Simplicity
- 2. Uniform Semantics
  - Local Communication
  - Remote communication
- 3. Efficiency
- 4. Reliability
- 5. Correctness
- Issues related to correctnes:
  - Atomicity
  - Ordered delivery
  - Survivability
- 6. Flexibility
- 7. Security
- 8. Portability

ISSUES IN IPC BY MESSAGE PASSING-I

- A message is a block of information formatted by a sending process in such a manner that
- It is meaningful to the receiving process.
- It consists of a fixed-length header and
- A variable-size collection of typed data objects.
- As shown in Figure:



ISSUES IN IPC BY MESSAGE PASSING-II

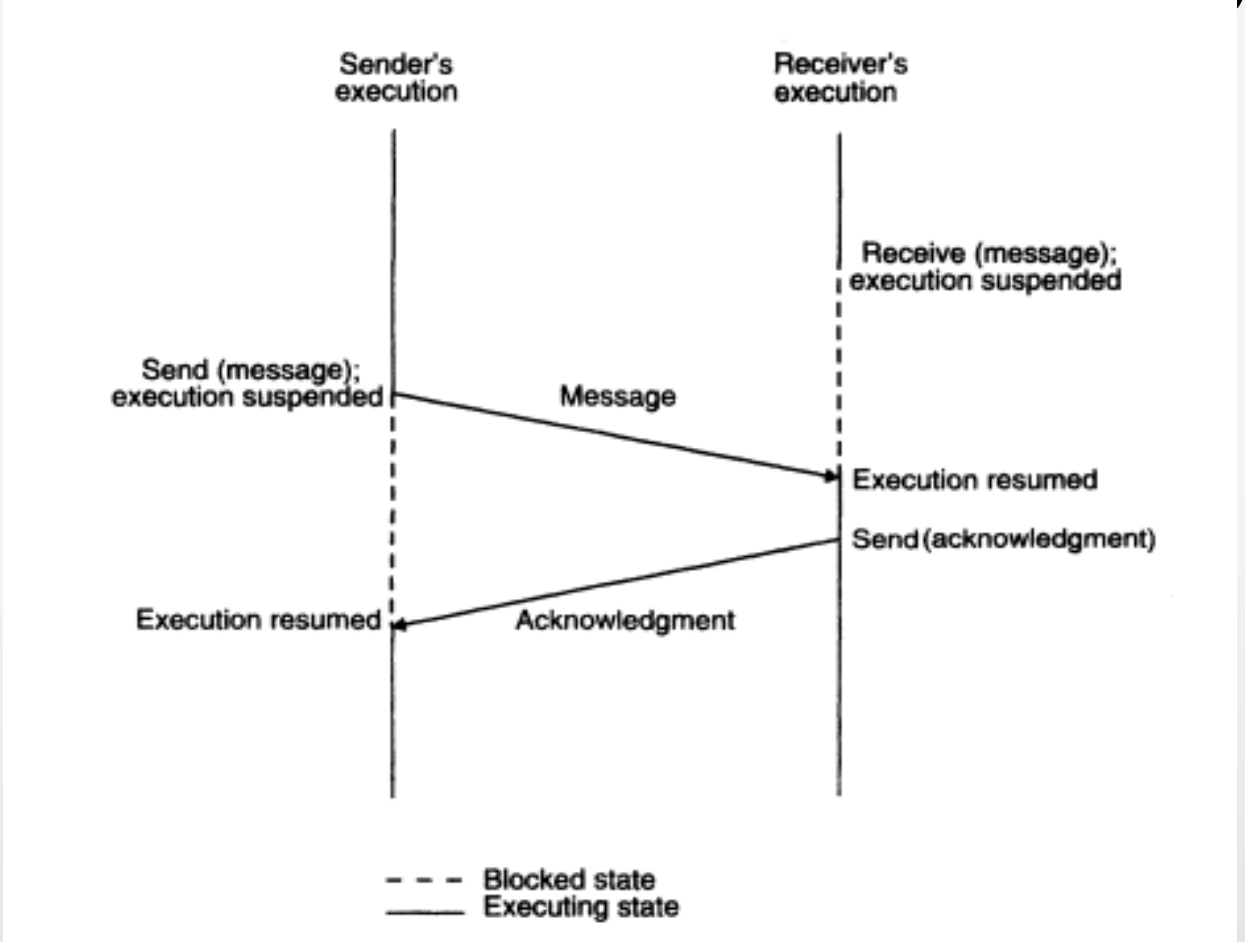
- In the design of an IPC protocol for a message-passing system, the following
- important issues need to be considered:
  - • Who is the sender?
  - • Who is the receiver?
  - • Is there one receiver or many receivers?
  - • Is the message guaranteed to have been accepted by its receiver(s)?
  - • Does the sender need to wait for a reply?
  - • What should be done if a catastrophic event such as a node crash or a communication link failure occurs during the course of communication?
  - • What should be done if the receiver is not ready to accept the message: Will the message be discarded or stored in a buffer? In the case of buffering, what should be done if the buffer is full?
  - • If there are several outstanding messages for a receiver, can it choose the order in which to service the outstanding messages?

SYNCHRONIZATION-I

- A central issue in the communication structure is the synchronization.
- The semantics used synchronization may be broadly classified as:-
  - → Blocking
  - → Non-Blocking
- An important issue in non-blocking receive primitive is how the receiving process knows that message is arrived in the message buffer.
  - → 1. Polling
  - → 2. Interrupt
- 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.

SYNCHRONIZATION-II

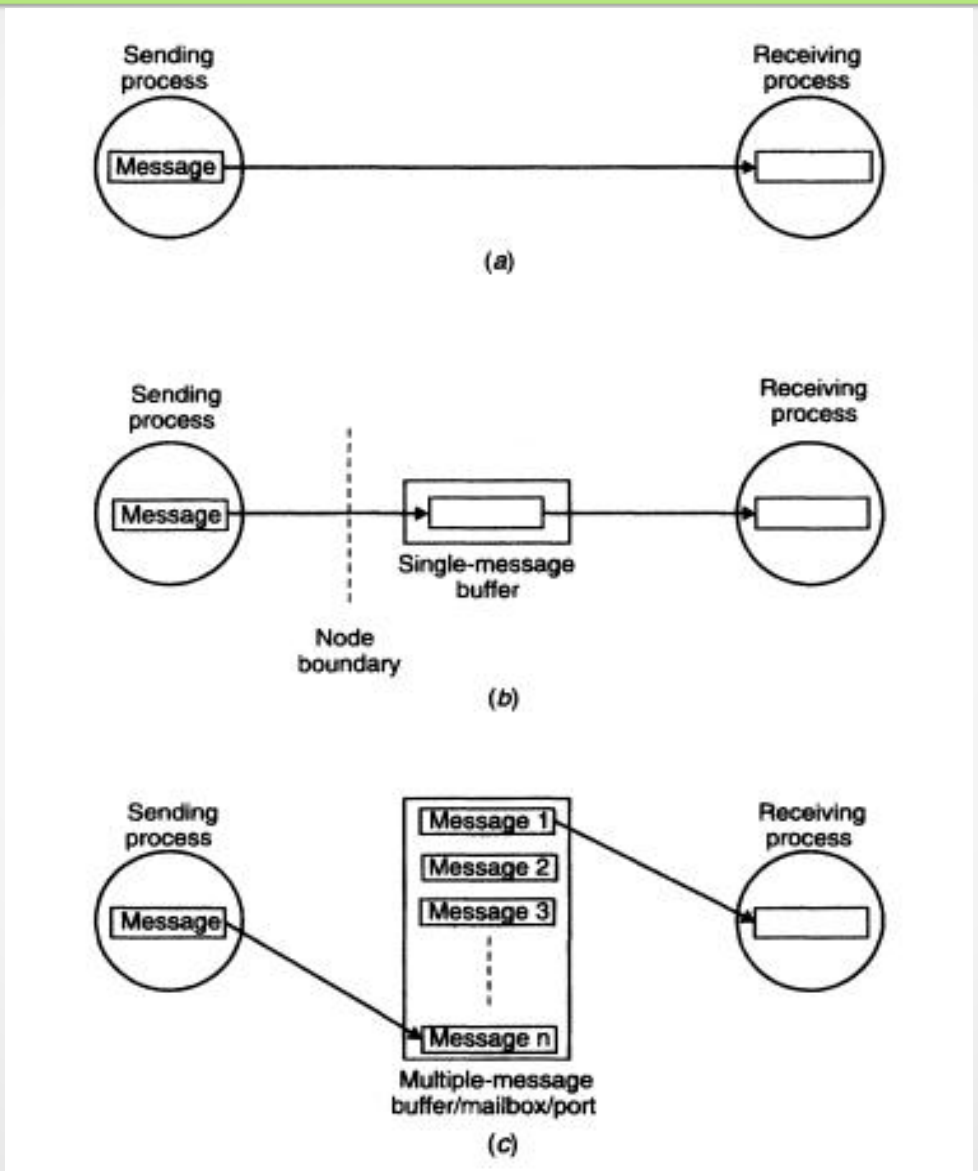
- Figure: Synchronous Mode of Communication with send and receive primitives having blocking type semanti



BUFFERING-I

- The synchronous and asunchronous mode of communication correspond respectively to the two extremes of buffering: a Null Buffer or No Buffering and a buffer with unbounded capacity.
- Other two commonly used buffering strategies are Single-buffering and finite bound or multiple message buffers.
- These four types of buffering strategies are;
  - → Null Buffer or No Buffering
  - → Single Message Buffer
  - → Unbounded-Capacity Buffer
  - → Finite Bound or Multiple Message Buffer

BUFFERING-II



MUlTIDATAGRAM MESSAGES

- MTU (maximum transfer unit)
- Each packet is known as Datagram
- Single Datagram Messages: messages less than MTU
- Multi Datagram Messages: messages larger than MTU

Encoding and Decoding of Message Data

- A message data should be meaningful to the receiving process. This implies that, ideally, the structure of the program object should be preserved while they are being transmitted from address space of the sending process to the address space of the receiving process. This obviously is not possible in a heterogeneous systems in which the sending and receiving processes are on different computers of different architectures. However, even in homogeneous systems, it is very difficult to achieve this goal mainly because of two reasons:
  - 1. An absolute pointer value loses
  - 2. Different program objects occupy varying amount of storage space.
- Due to above mentioned problem encoding and decoding is done.
- One of the following two representations may be used for encoding and decoding of a message data.
  - 1. In tagged representation
  - 2. In Untagged representation

PROCESS ADDRESSING

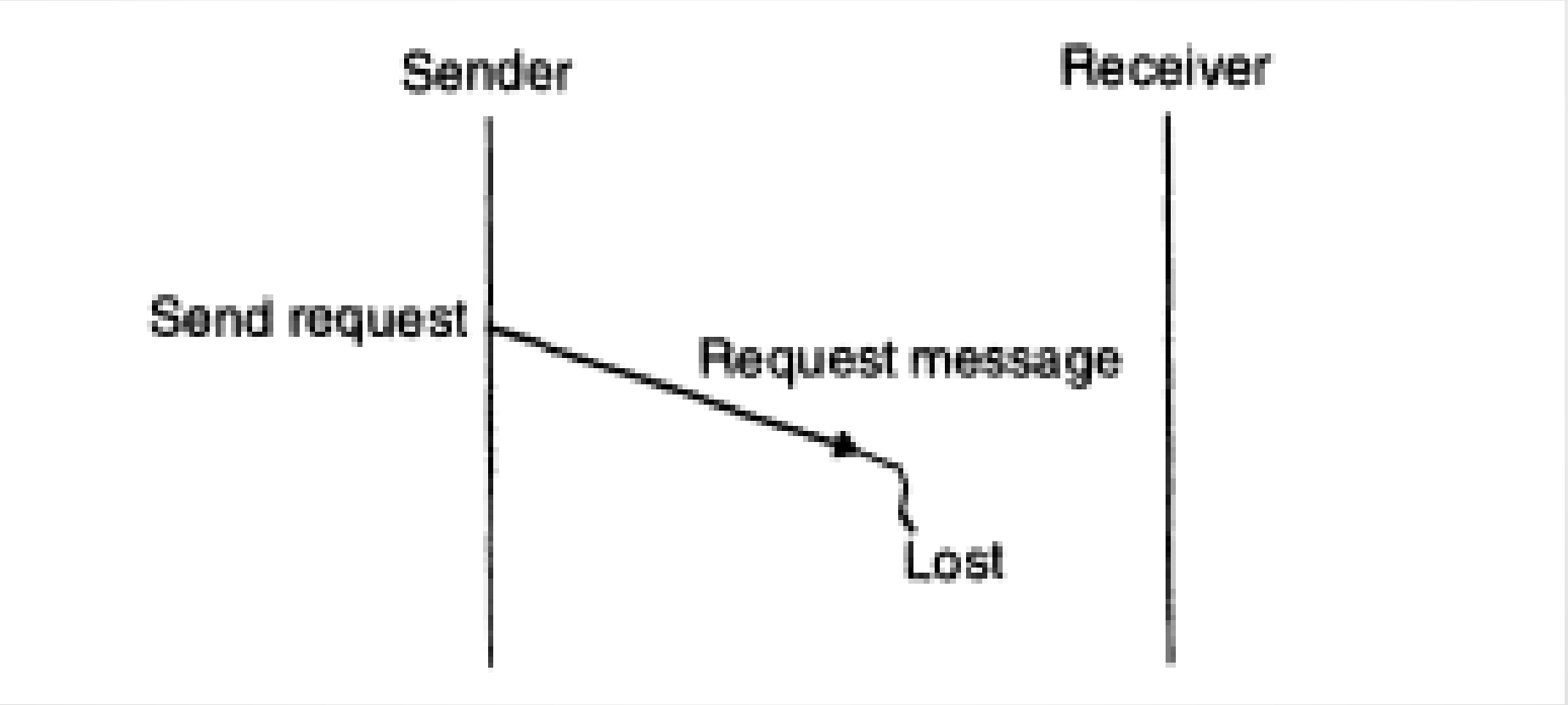
- Another important issue in message based communication is addressing(or naming)of the parties involved in an interaction.
- MPS usually supports two types of process addressing.
  - Explicit Addressing
    - send(process id,message)
    - receive(process id,message)
  - Implicit Addressing
    - send any(service id,message)
    - receive any(process id,message)
- Primitives for explicit and Implicit addressing of a process
- A simple method to identify a process is by combination of
- machine_id and local_id
  - (machine_id, local_id, machine_id)

Failure Handling

- While distributed system may offer potential for parallelism, it is also prone to partial failure such as node crash or a communication link failure. Therefore, during interprocess communication, such failures may lead the following problems:
- 1. Loss of Request Message
- 2. Loss of Response Message
- 3. Unsuccessful Execution of the Request

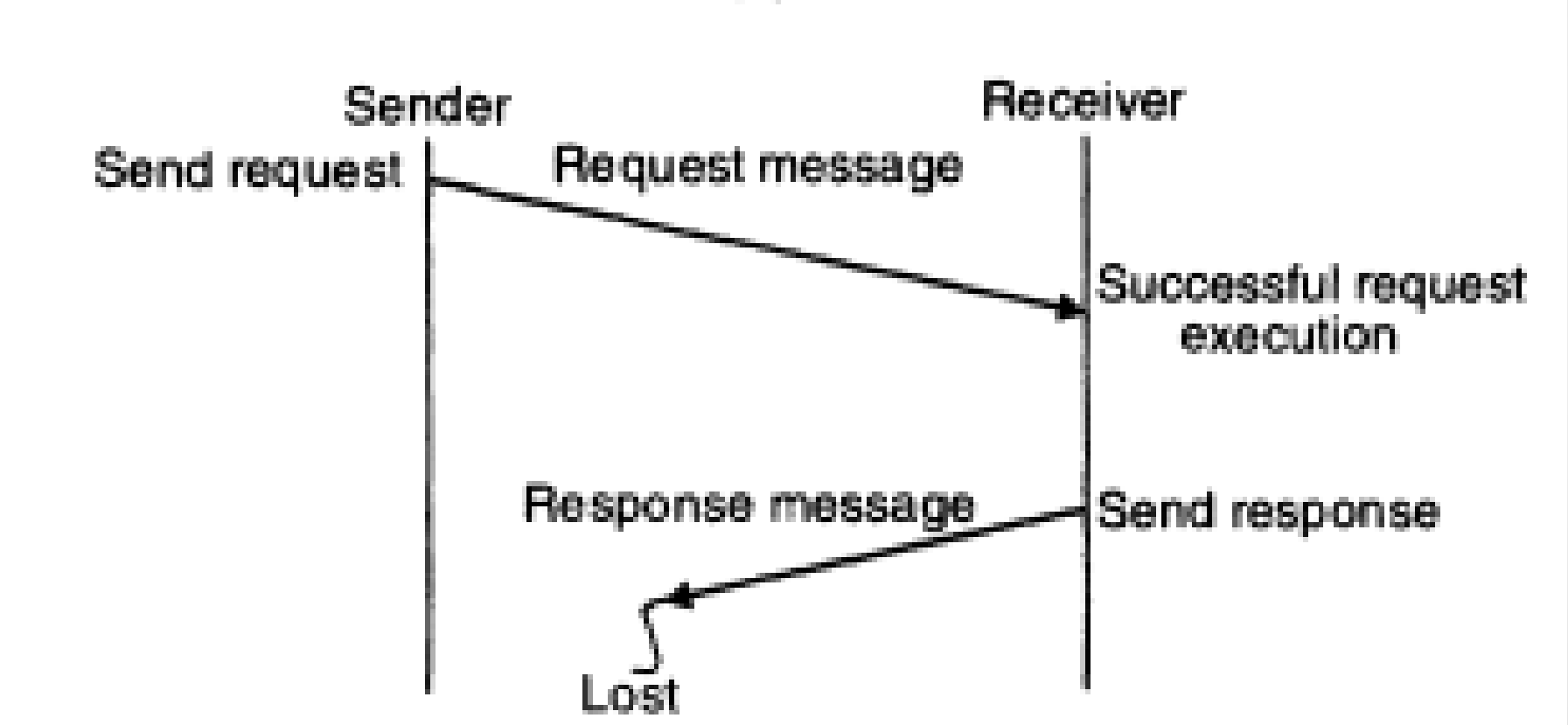
Failure Handling cntd...

- Request message is lost.



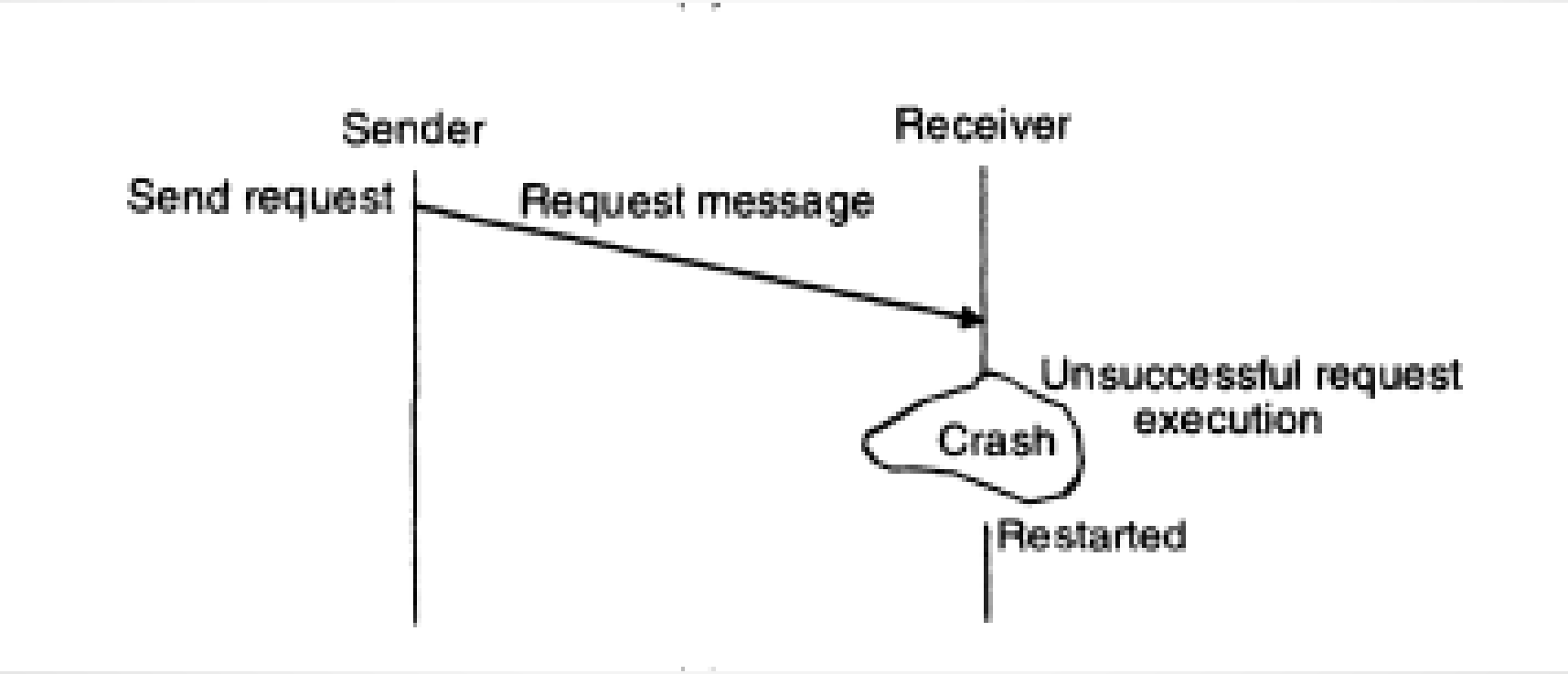
Failure Handling cntd...

- Response message is lost.



Failure Handling cntd...

- Receiver's computer crashed.

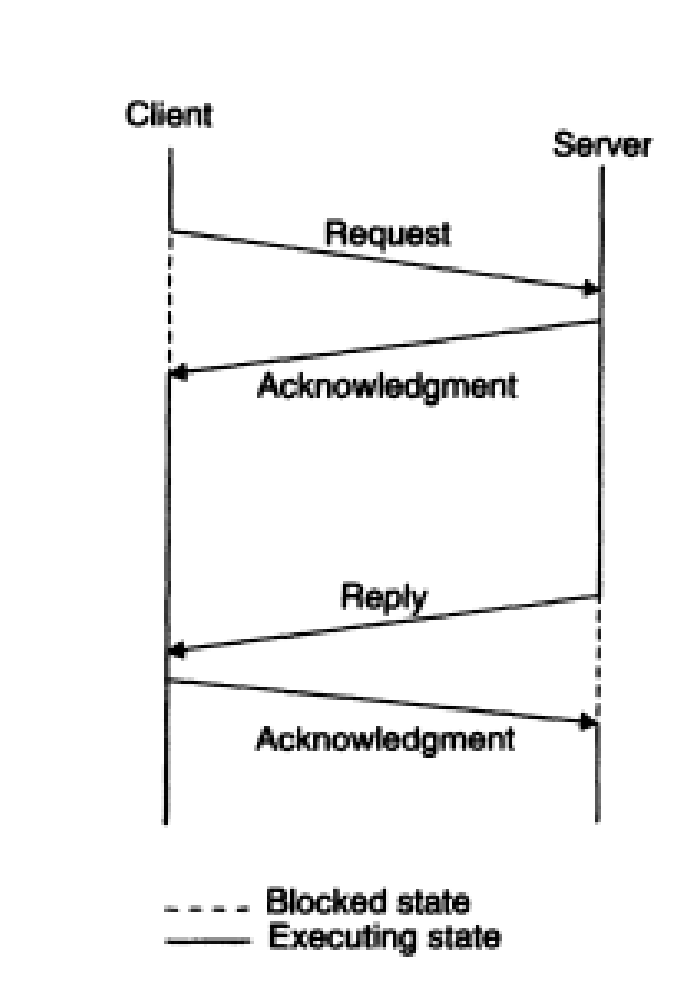


Failure Handling cntd...

- To cope up with these problems, a reliable IPC protocol of a MPS is normally designed based on the idea of internal retransmission of message after timeouts and the return of the acknowledgement message of the sending machine's kernel be the receiving machine's kernel.
- Based on the above idea, a fourway message reliable IPC protocol for client-server between two processes works as follows:
  - 1. Four-Message Reliable IPC Protocol
  - 2. Three-Message Reliable IPC Protocol
  - 3. Two-Message Reliable IPC Protocol

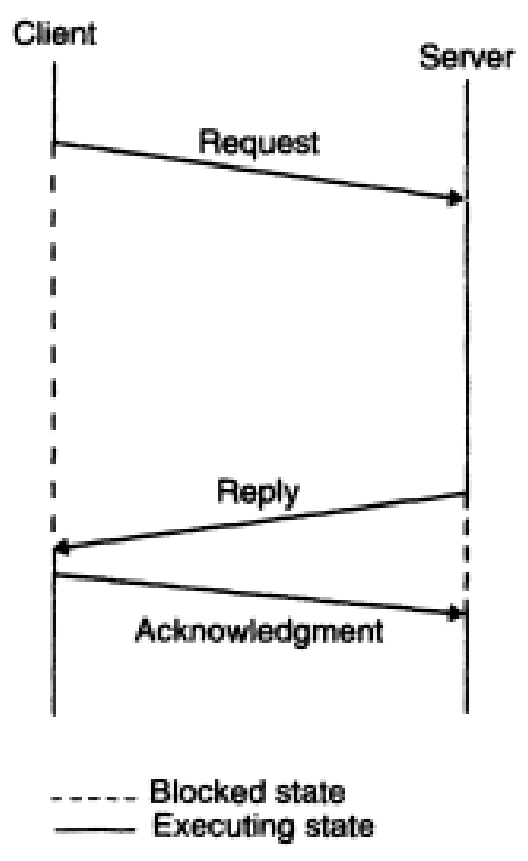
Failure Handling cntd...

- Four-message reliable IPC protocol



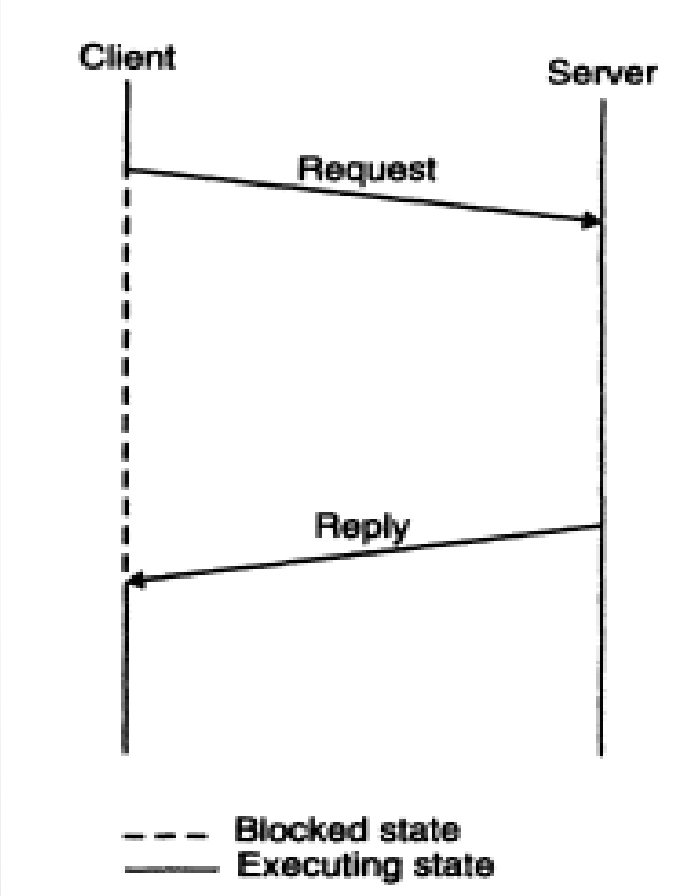
Failure Handling cntd...

- The three-message reliable IPC protocol for client-server communication between two processes.

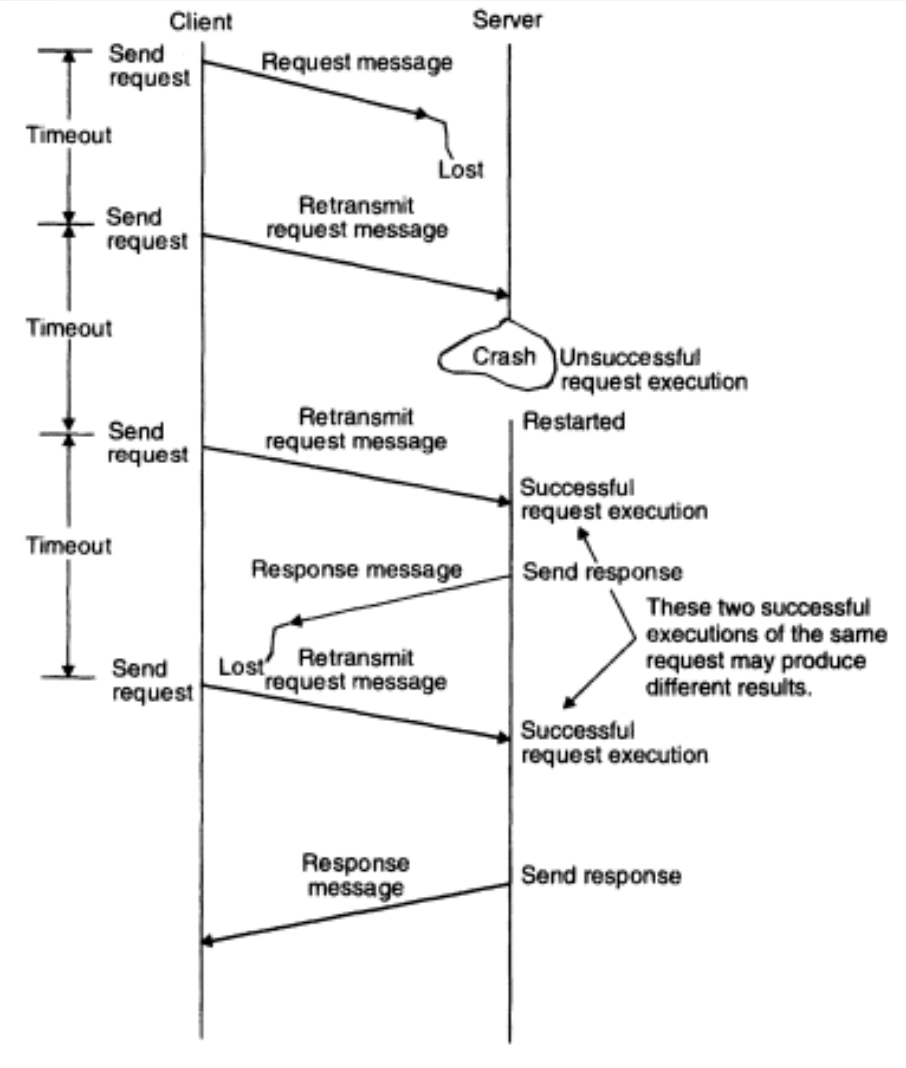


Failure Handling cntd...

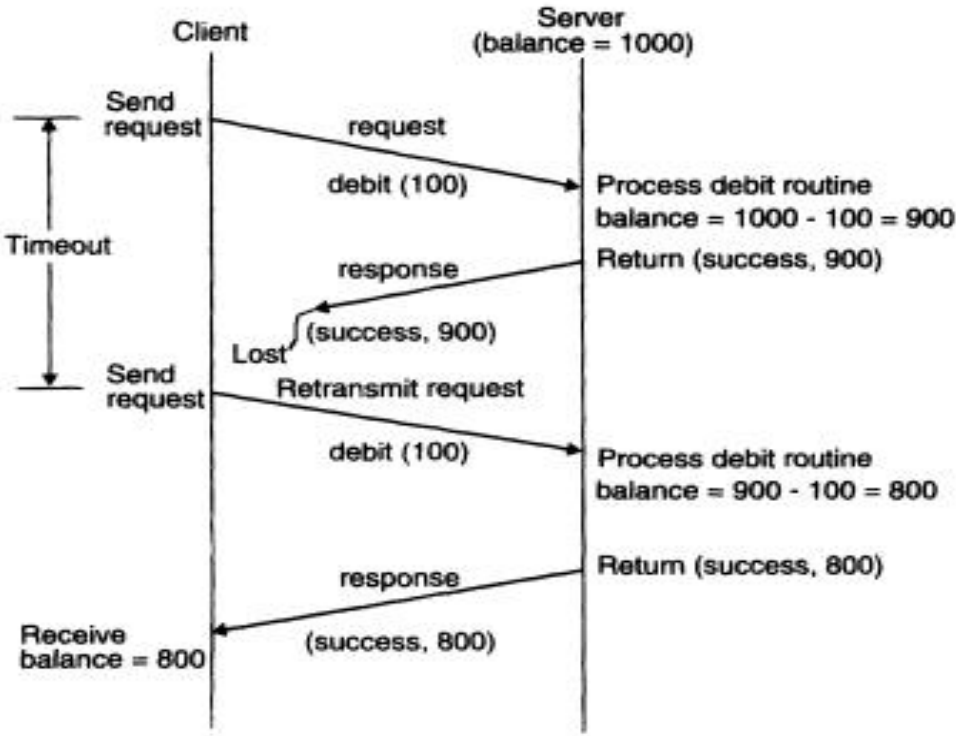
- The two-message IPC protocol used in many systems for client-server communication between two processes.



Failure Handling cntd... (example)

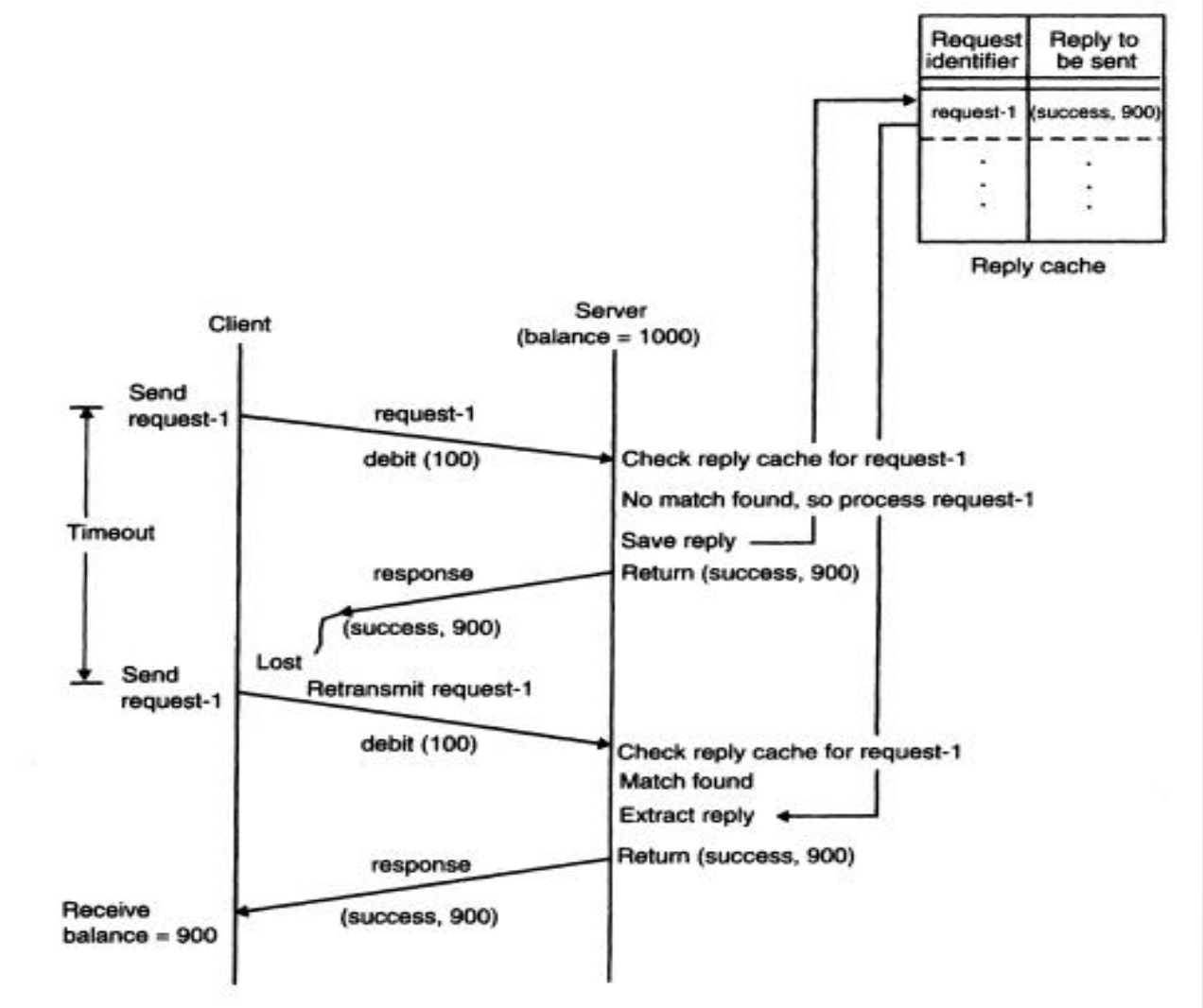


Idempotency and Handling of Duplicate Request Messages

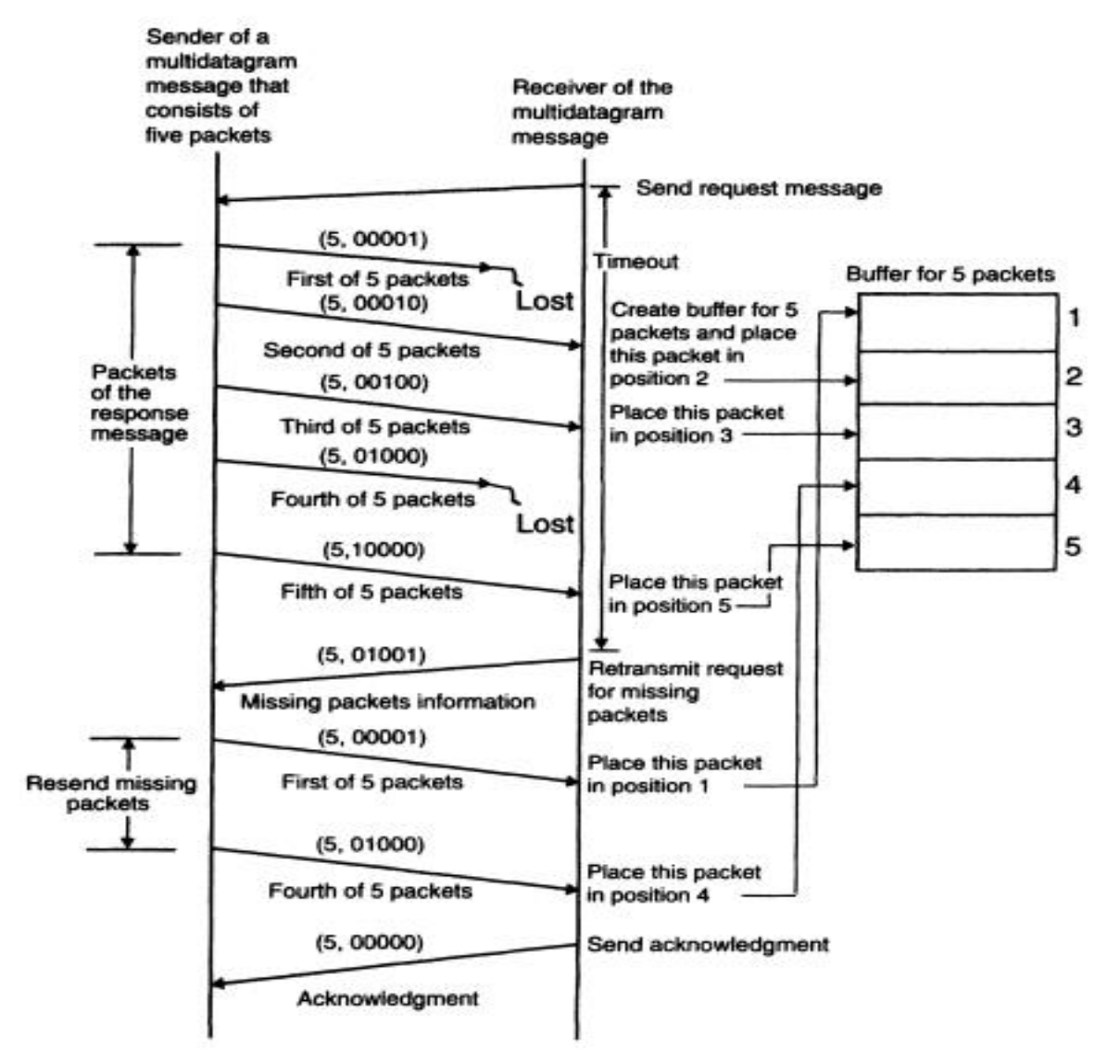



```
debit (amount)
{
    if (balance >= amount)
    {  balance = balance - amount;
       return ("success", balance);
    }
    else return ("failure", balance);
}
```

Idempotency and Handling of Duplicate Request Messages .....(cntd)



Keeplng Track of lost and Out-of-Sequence Packets in Multldatagram Messges



Group Communication

Depending on single or multiple senders and receivers, the following are the three types of group communications;

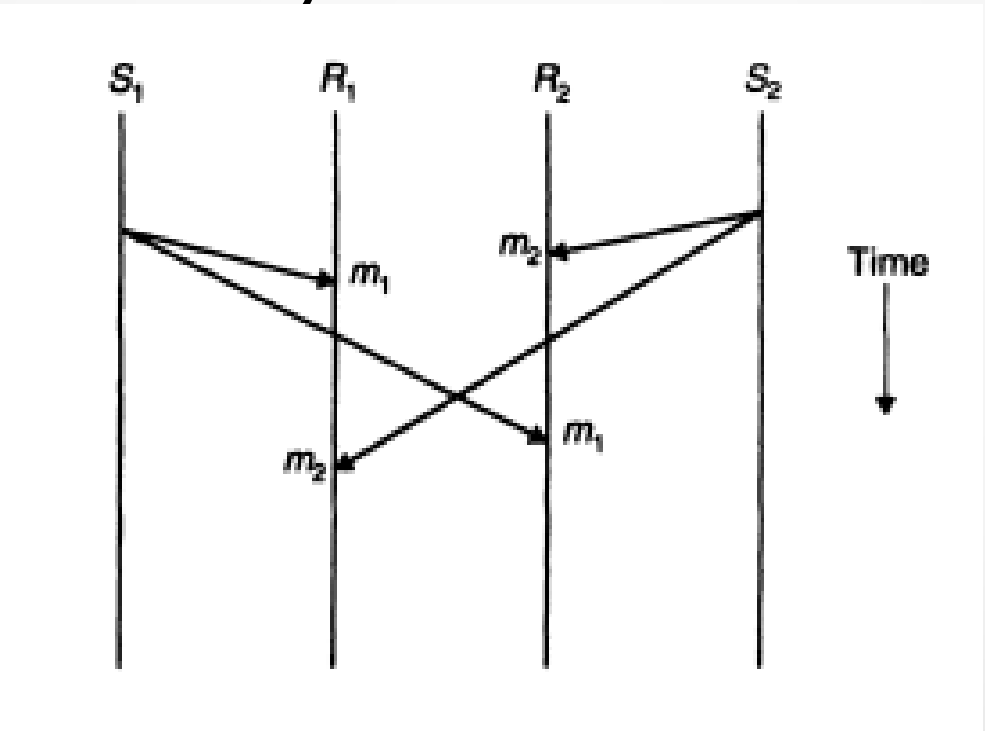
1. One-to-Many
2. Many-to-One
3. Many-to-Many

Group Communication...cntd

- 1. One-to-Many
  - Group Management
  - Group Addressing
  - Message Delivery to Receiver Process
  - Buffered and Unbuffered Multicast
  - Send-to-All and Bulletin-Board Semantics
  - Flexible Reliability in Multicast Communication
  - Atomic Multicast

Group Communication...cntd

- 2. Many-to-One
- 3. Many-to-Many: No ordering constraint for message delivery



Group Communication...cntd

- The commonly used semantics for ordered delivery of multicast messages are
  - Absolute ordering,
  - Consistent ordering, and
  - Causal ordering.

Group Communication...cntd

- Absolute Ordering

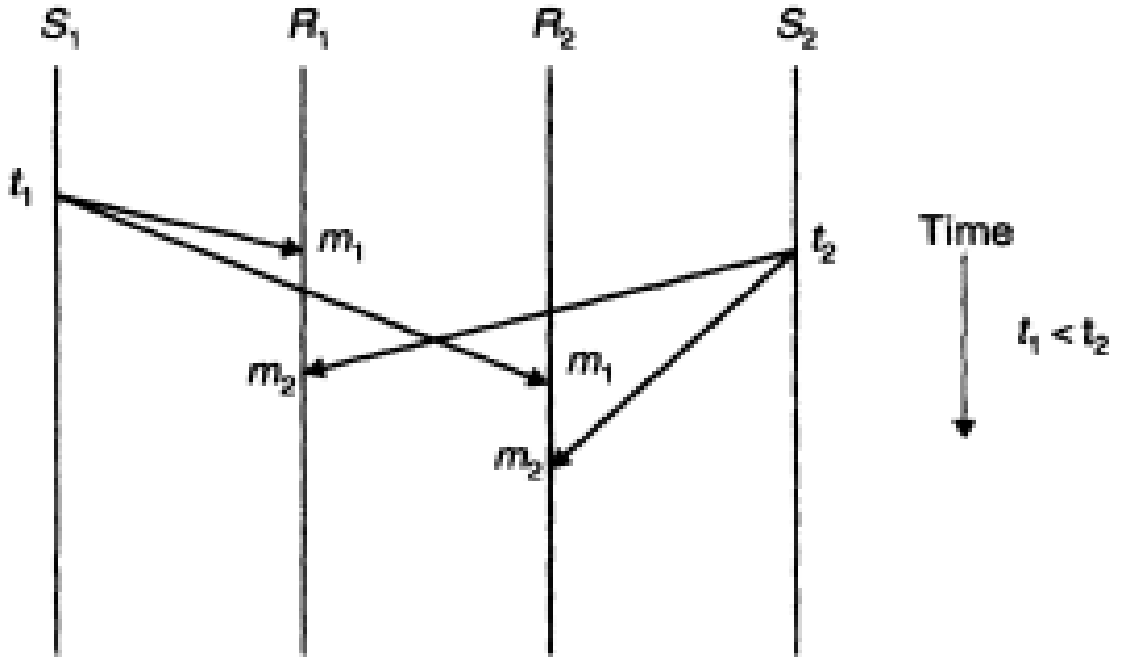


Fig. 3.15 Absolute ordering of messages.

Group Communication...cntd

- Consistent Ordering

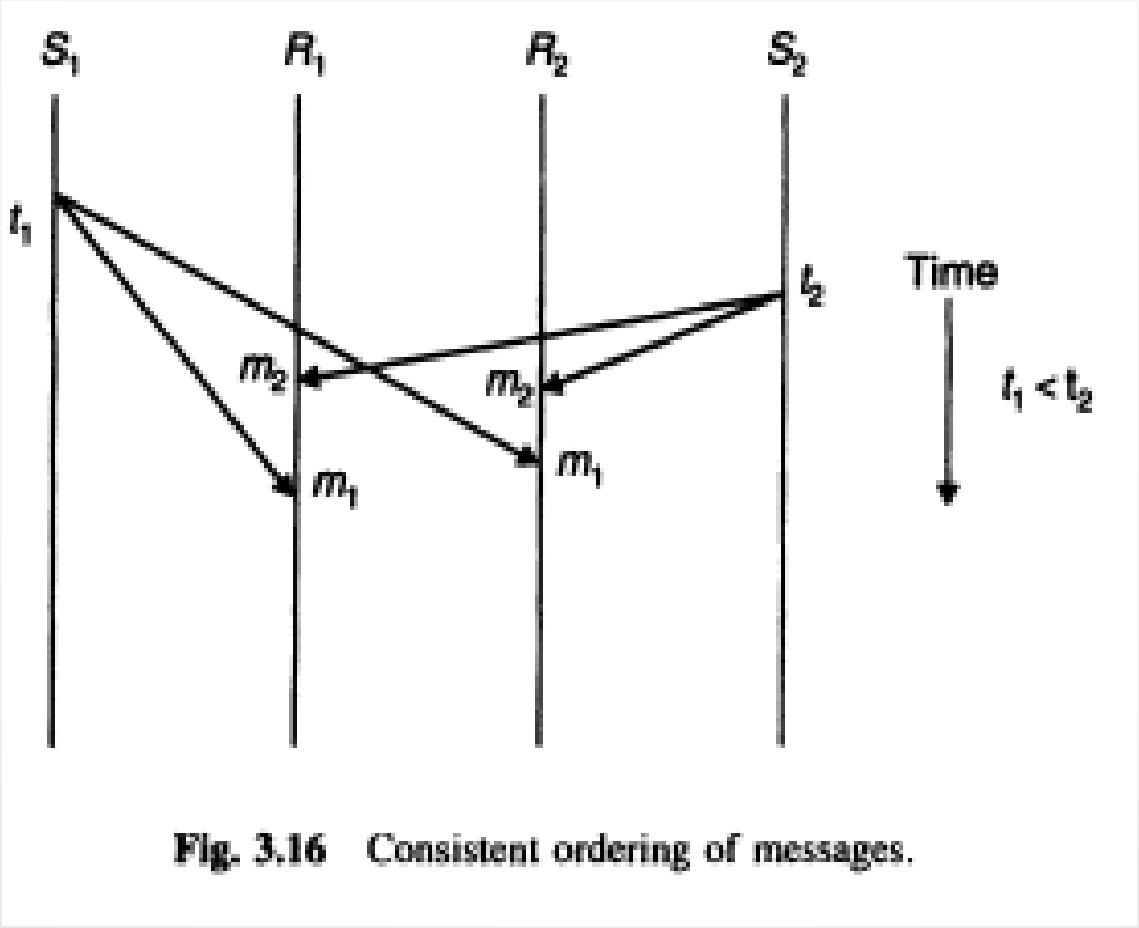


Fig. 3.16 Consistent ordering of messages.

Consistent Ordering

The sequencer-based method for implementing consistent-ordering semantics is subject to single point of failure and hence has poor reliability. A distributed algorithm for implementing consistent-ordering semantics that does not suffer from this problem is the ABCAST protocol of the ISIS system:

1. The sender assigns a temporary sequence number to the message and sends it to all the members of the multicast group.
2. A member (i) calculates its proposed sequencenumber
   by using the function
   max(Fmax , P max) + 1 + i/N
3. it selects the largest one as the final sequence number for the message and sends it to all members in a commit message.
4. On receiving the commit message, each member attaches the final sequence number to the message.
5. Committed messages with final sequence numbers are delivered to the application programs in order of their final sequence numbers.

Group Communication...cntd

- Causal Ordering

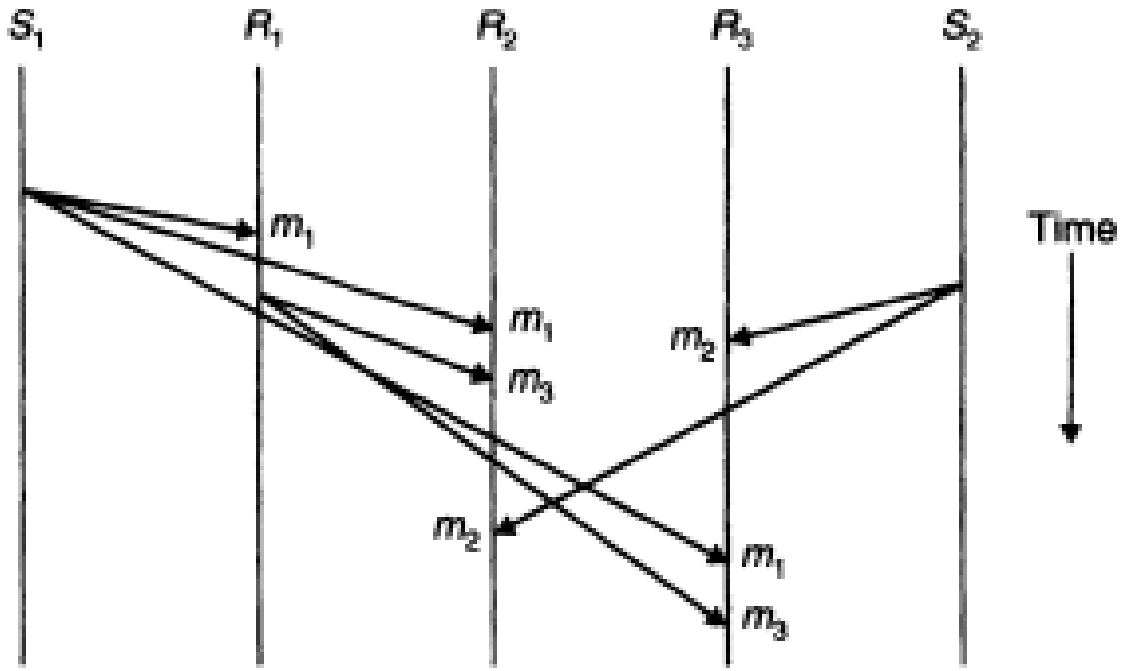


Fig. 3.17 Causal ordering of messages.

Causal Ordering

1. Each member process of a group maintains a vector of n components, where n is the total number of members in the group. Each member is assigned a sequence number from 0 to n, and the ith component of the vectors corresponds to the member with sequence number i. In particular, the value of the ith component of a member's vector is equal to the number of the last message received in sequence by this member from member i.

2. To send a message, a process increments the value of its own component in its own vector and sends the vector as part of the message.

3. When the message arrives at a receiver process's site, it is buffered by the runtime system. The runtime system tests the two conditions given below to decide whether the message can be delivered to the user process or its delivery must be delayed to ensure causal-ordering semantics. Let S be the vector of the sender process that is attached to the message and R be the vector of the receiver process. Also let i be the sequence number of the sender process. Then the two conditions to be tested are

$$S[i] = R[i] + 1 \quad \text{and} \quad S[j] \leq R[j] \quad \text{for all } j \neq i$$

The first condition ensures that the receiver has not missed any message from the sender. This test is needed because two messages from the same sender are always causally related. The second condition ensures that the sender has not received any message that the receiver has not yet received. This test is needed to make sure that the sender's message is not causally related to a message missed by the receiver.

Causal Ordering

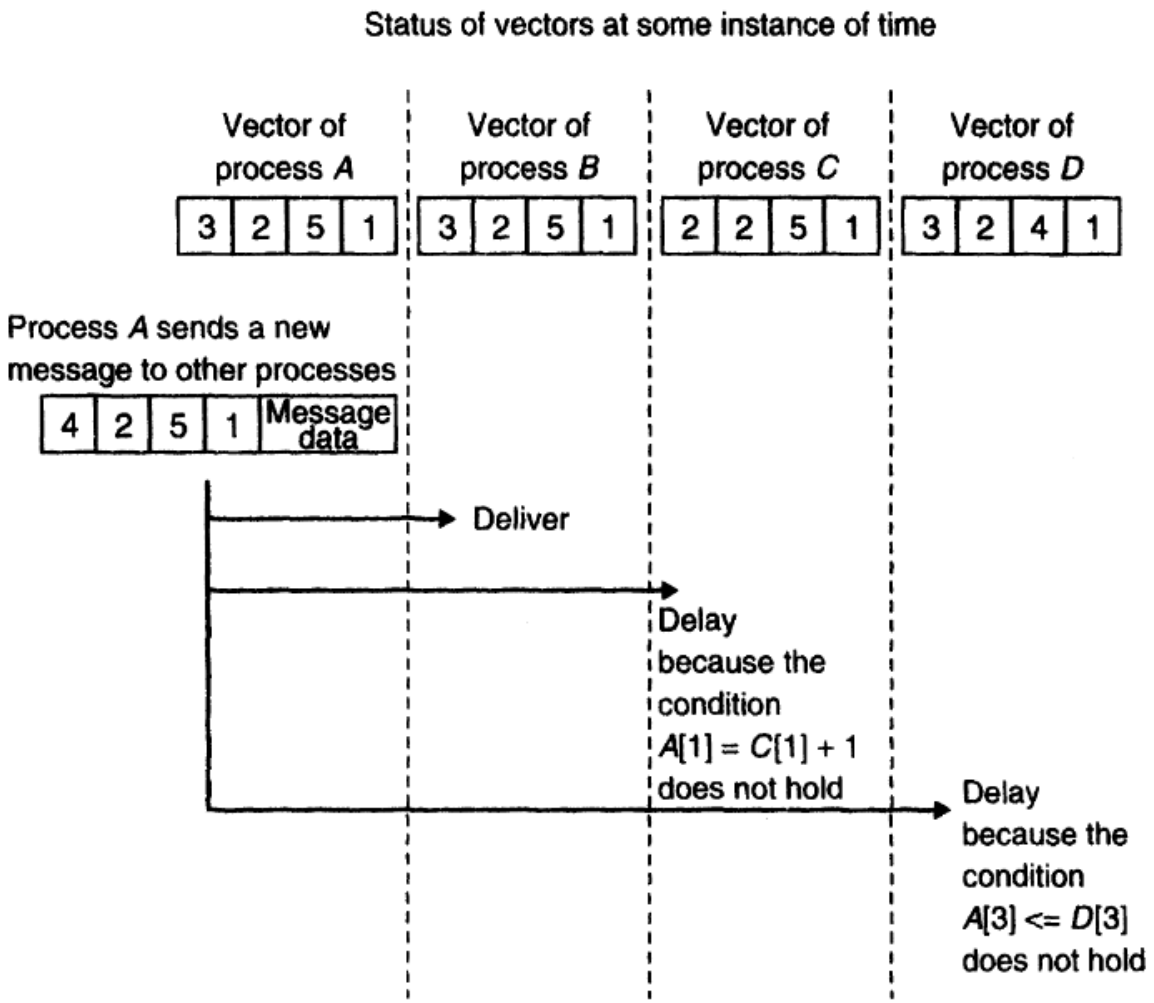


Fig. 3.18 An example to illustrate the CBCAST protocol for implementing causal ordering semantics.