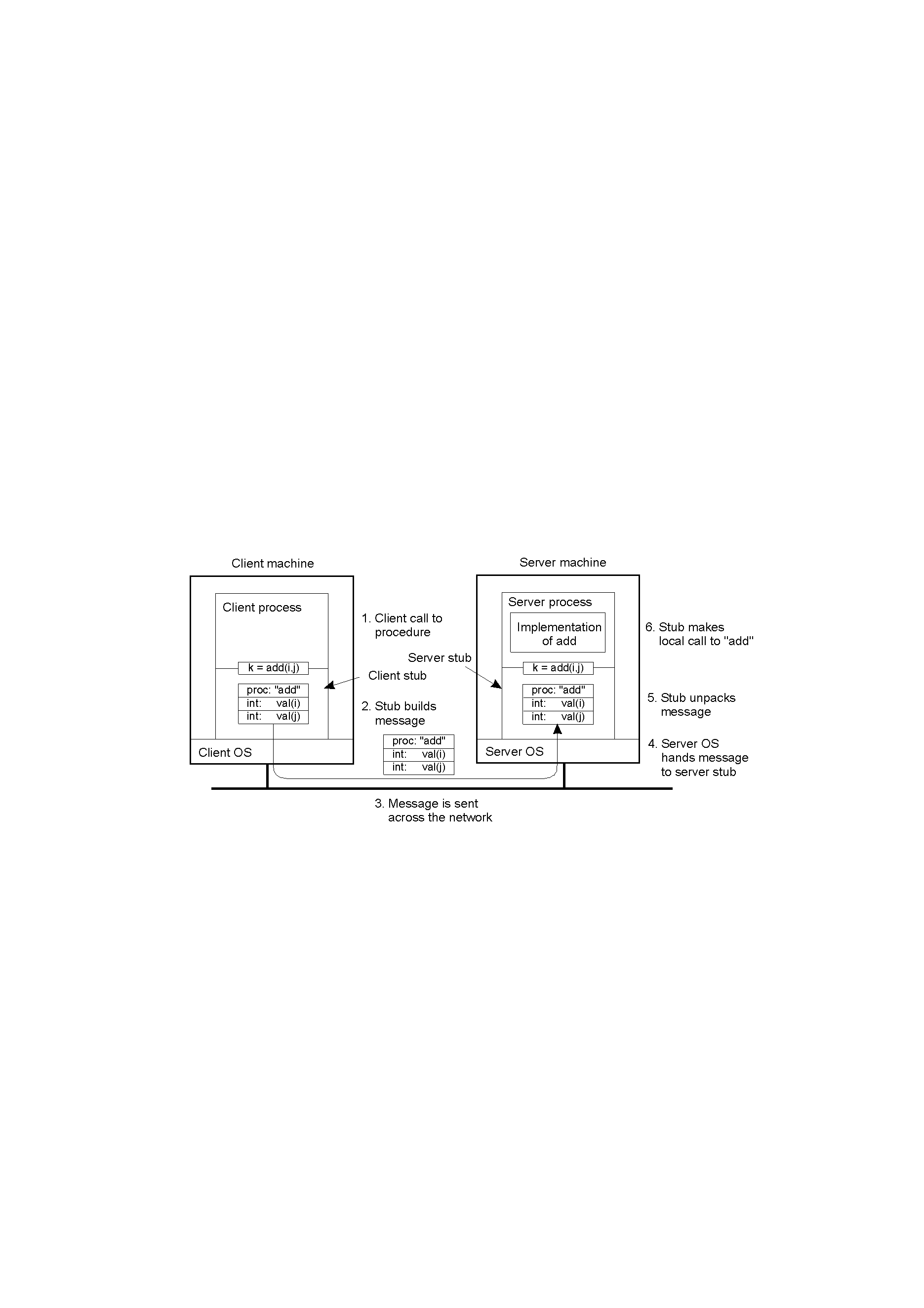
**Unit – II: Processes, Communication and Synchronization [1, 2, 3]**

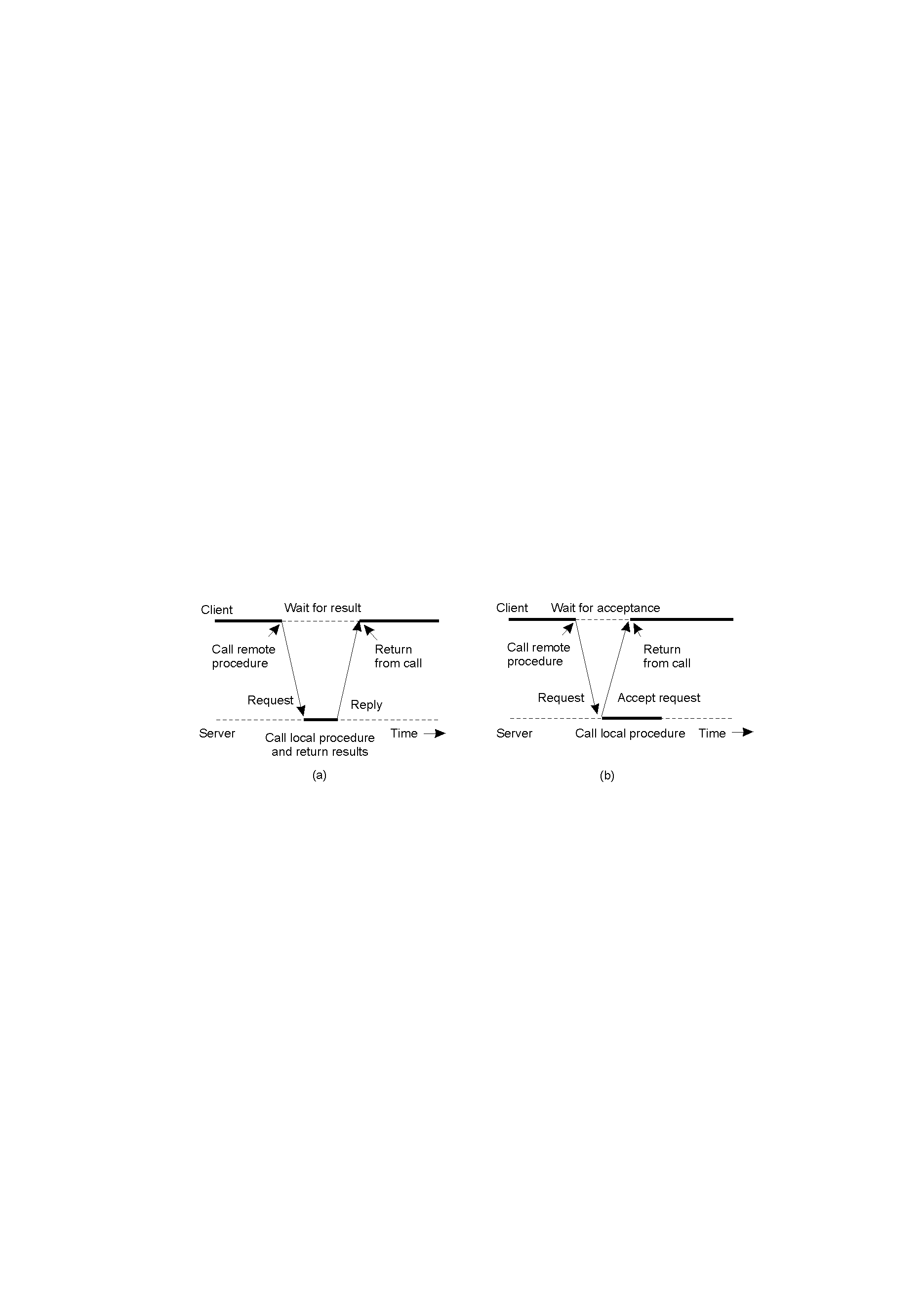
**Remote Procedure Call [1, 2, 3]**

* **When making a RPC:**
  + The calling environment is suspended.
  + Procedure parameters are transferred across the network to the environment where the procedure is to execute.
  + The procedure is executed **there.**
  + When the procedure finishes, the results are transferred back to the calling environment.
  + Execution resumes as if returning from a regular procedure call.

1. The client procedure calls a **client stub** passing parameters in the normal way.
2. The client stub ***marshals the parameters,*** builds the message, and calls the local OS.
3. The client's OS sends the message **(using the transport layer)** to the remote OS.
4. The server remote OS gives **transport layer** message to a **server stub**.
5. The server stub **demarshals the parameters** and calls the desired server routine.
6. The server routine does work and returns result to the server stub via normal procedures.
7. The server stub ***marshals the return values*** into the message and calls local OS.
8. The server OS **(using the transport layer)** sends the message to the client's OS.
9. The client's OS gives the message to the client stub
10. The client stub ***demarshals the result***, and execution returns to the client.



* **Asynchronous RPC [1, 2, 3]**



**Figure:**

1. The interconnection between client and server in a traditional RPC
2. The interaction using asynchronous RPC

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)

**Message Oriented Transient Communications: - [1, 2, 3]**

* **Message oriented transient communications :-**

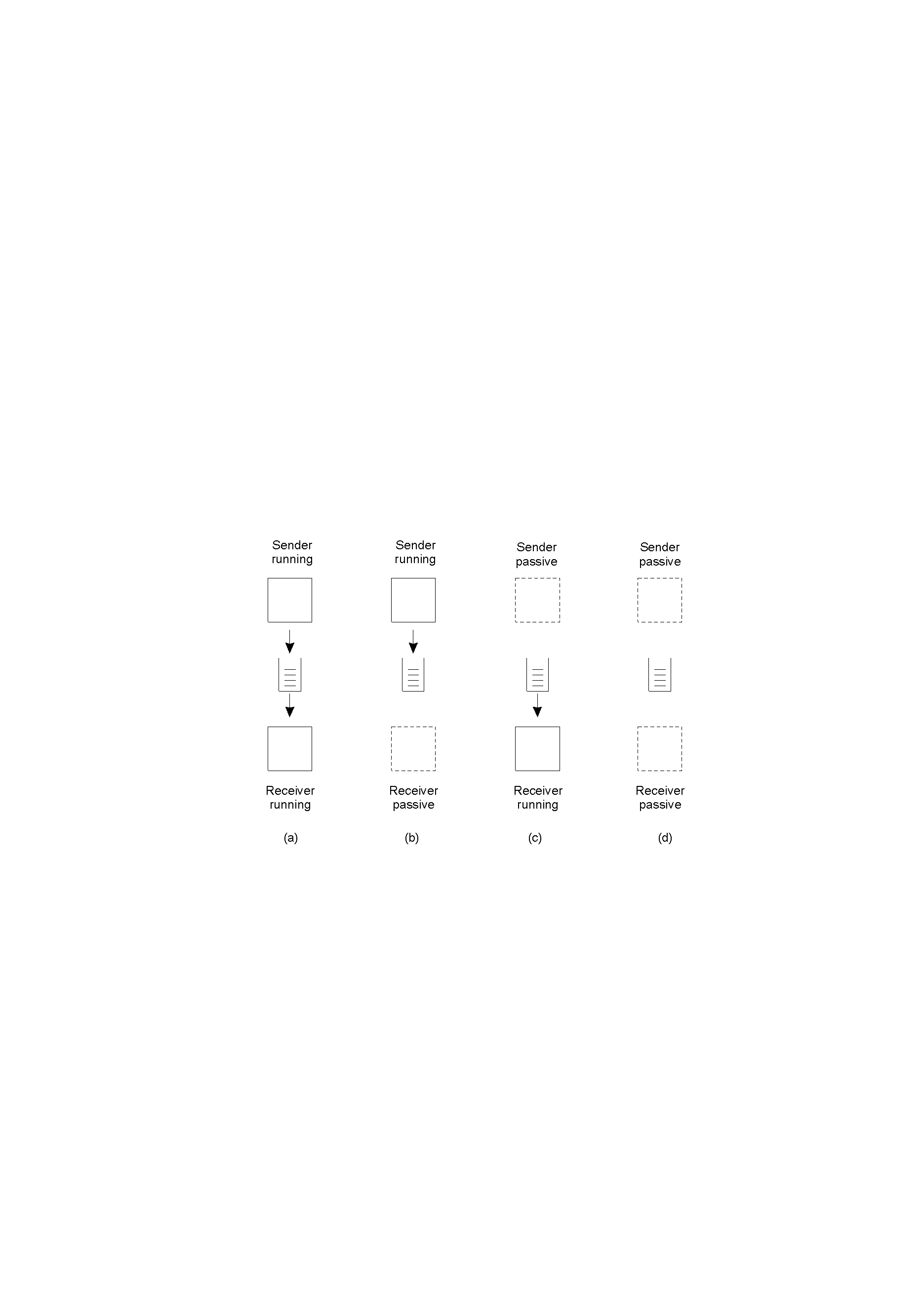
Berkeley Sockets

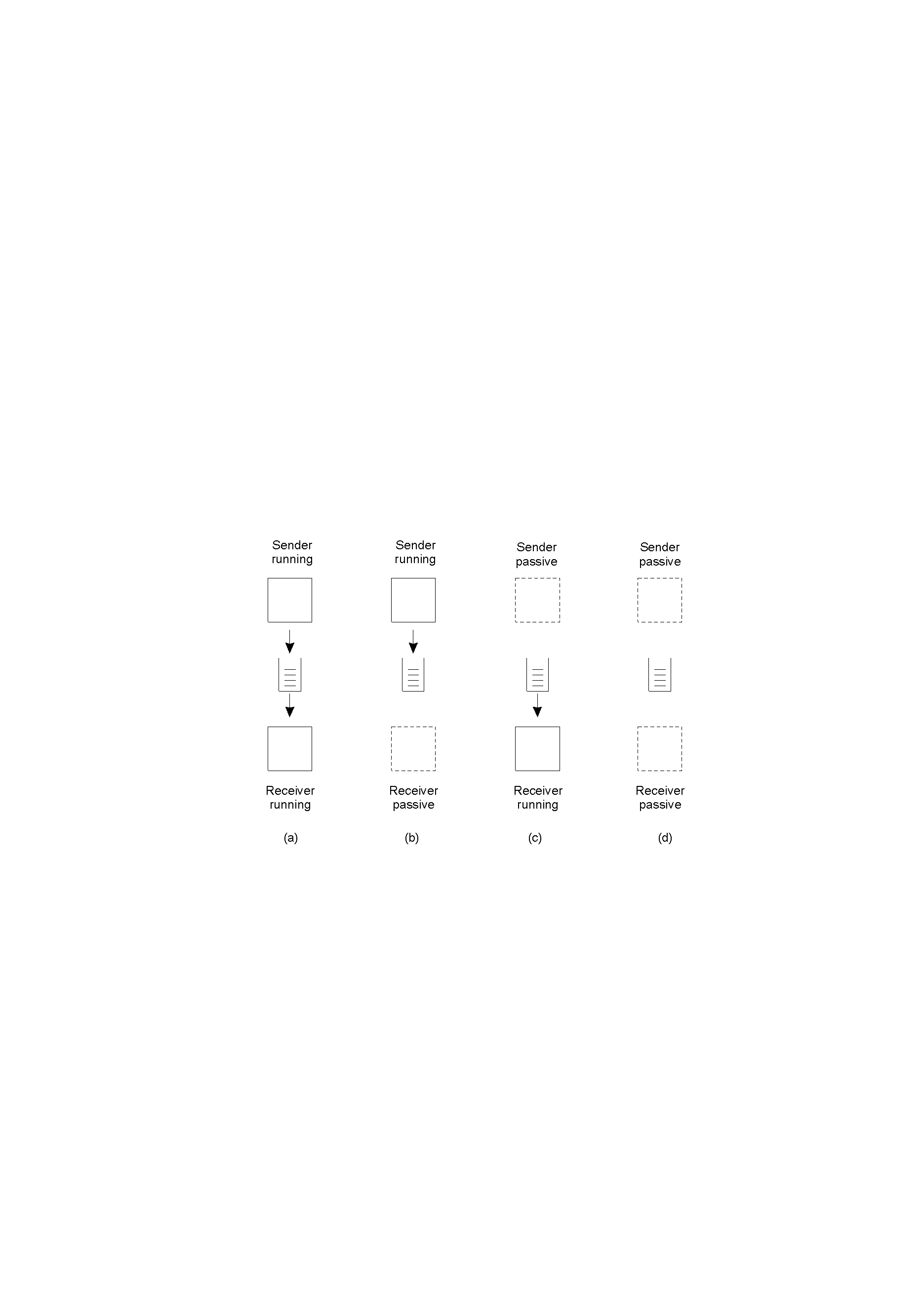
Socket primitives for TCP/IP.

|  |  |
| --- | --- |
| **Primitive** | **Meaning** |
| Socket | Create a new communication endpoint |
| Bind | Attach a local address to a socket |
| Listen | Announce willingness to accept connections |
| Accept | Block caller until a connection request arrives |
| Connect | Actively attempt to establish a connection |
| Send | Send some data over the connection |
| Receive | Receive some data over the connection |
| Close | Release the connection |

* **Message oriented persistent communications :- [1, 2, 3]  
  Message-Queuing Model**

**Basic idea:** applications communicate by inserting messages in specific queues





* **Four combinations for loosely-coupled communications using queues.**

|  |  |
| --- | --- |
| **Primitive** | **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 messages, and remove the first. Never block. |
| Notify | Install a handler to be called when a message is put into the specified queue |

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)

**Physical Clock Synchronization [1, 2, 3]**

**Physical Clocks (1)**

**Physical Clocks (2) [1, 2, 3]**

**Global Positioning System (1)**

**Global Positioning System (2) [1, 2, 3]**

Real world facts that complicate GPS

1. It takes a while before data on a satellite’s position reaches the receiver.
2. The receiver’s clock is generally not in synch with that of a satellite..

**Clock Synchronization Algorithms**

**Network Time Protocol.**

**Assume delay is about symmetric**

1. Skew = [(T2-T1)+(T3-T4)]/2.
2. Take many (8) pair of (skew,delay) and use one with best (least) delay.
3. Clocks have “strata” - lowest stratum is best – only adjust clock if your stratum is higher.

**The Berkeley Algorithm (1) [1, 2, 3]**

**The Berkeley Algorithm (2) [1, 2, 3]**

**The Berkeley Algorithm (3)**

**Clock Synchronization in Wireless Networks (1) [1, 2, 3]**

**Clock Synchronization in Wireless Networks (2) [1, 2, 3]**

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)

**Logical Clock Synchronization:- [1, 2, 3]**

**Lamport’s Logical Clocks (1)**

**The "happens-before" relation *→* can be observed directly in two situations:**

* If *a* and *b* are events in the same process, and *a* occurs before *b*, then *a → b* is true.
* 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 → b*

**Lamport’s Logical Clocks (2) [1, 2, 3]**

**Lamport’s Logical Clocks (3) [1, 2, 3]**

**Lamport’s Logical Clocks (4) [1, 2, 3]**

**Lamport’s Logical Clocks (5) [1, 2, 3]**

**Updating counter C*i* for process P*i***

1. Before executing an event P*i* executes   
   C*i* ← C*i* + 1.
2. When process P*i* sends a message m to P*j*, it sets *m*’s timestamp *ts (m)* equal to C*i* after having executed the previous step.
3. Upon the receipt of a message *m*, process P*j*adjusts its own local counter as   
   C*j* ← max{C*j* , *ts (m)*}, after which it then executes the first step and delivers the message to the application.

**Example: Totally Ordered Multicasting [1, 2, 3]**

**Vector Clocks (1)**

**Vector Clocks (2) [1, 2, 3]**

**Vector clocks are constructed by letting each process P*i* maintain a vector VC*i* with the following two properties:**

1. VC*i* causality [ *i* ] is the number of events that have occurred so far at P*i*. In other words, VC*i* [ *i* ] is the local logical clock at process P*i* .
2. If VC*i* [ *j* ] = k then P*i* knows that k events have occurred at P*j*. It is thus P*i*’s knowledge of the local time at P*j* .

**Vector Clocks (3) [1, 2, 3]**

**Steps carried out to accomplish property 2 of previous slide:**

1. Before executing an event P*i* executes   
   VC*i* [ *i* ] ← VC*i* [*i* ] + 1.
2. When process P*i* sends a message m to P*j*, it sets *m*’s (vector) timestamp *ts (m)* equal to VC*i* after having executed the previous step.
3. Upon the receipt of a message m, process P*j* adjusts its own vector by setting   
   VC*j* [*k* ] ← max{VC*j* [*k* ], *ts (m)*[*k* ]} for each *k*, after which it executes the first step and delivers the message to the application.

**Enforcing Causal Communication**

**Atomic Multicast [1, 2, 3]**

**Want all nodes in group G to process all messages in same order**

1. Each node maintains a Lamport logical clock C.
2. When node n sends a message m to G it includes a timestamp ts(m) = C.
3. When node n' receives m, it updates its clock C' and sends a timestamped ACK a', with ts(a')=C'.
4. Node n collects all ACKs and takes the maximum timestamp T(m)=max{ts(a)} for all a, and sends a commit message c containing T(m) to G.
5. Node n' can deliver m when it receives c and it knows T(m) < T(m') for any other message m'.

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)

**Mutual Exclusion: [1, 2, 3]**

1. **Overview**

* Token: Passing special message between processes.
* Token based solution:

There is only one token available & whoever has that token is allowed to access the resource.

* Properties of Token based solution:

1. Avoid Starvation

2. Avoid Deadlock.

* Permission Based Approach: Process wanting to access resource first requires permission of other processes.
* Drawback:

Token lost, if the process holding it crashes.

**A Centralized Algorithm (1) [1, 2, 3]**

**A Centralized Algorithm (2) [1, 2, 3]**

**A Centralized Algorithm (3) [1, 2, 3]**

**A Centralized Algorithm (4) [1, 2, 3]**

* It is easy to implement & requires only three messages per resource(Request, Grant, Release)
* Drawback:

1. Coordinator is a single point of failure.

2. Single Coordinator may become performance bottleneck

**A Distributed Algorithm**

* When process wants to access shared resource ,it sends message to all processes in a group.
* Message consist of name of resource, process no & current time.

**A Distributed Algorithm (1) [1, 2, 3]**

**Three different cases:**

1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

**A Distributed Algorithm (2) [1, 2, 3]**

**A Distributed Algorithm (3) [1, 2, 3]**

**A Distributed Algorithm (4) [1, 2, 3]**

**A Token Ring Algorithm [1, 2, 3]**

**A Comparison of the Four Algorithms [1, 2, 3]**

**Global Positioning Of Nodes (1) [1, 2, 3]**

**Global Positioning Of Nodes (2)**

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)

**Election Algorithms [1, 2, 3]**

**The Bully Algorithm [1, 2, 3]**

1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. P’s job is done.

**The Bully Algorithm (1) [1, 2, 3]**

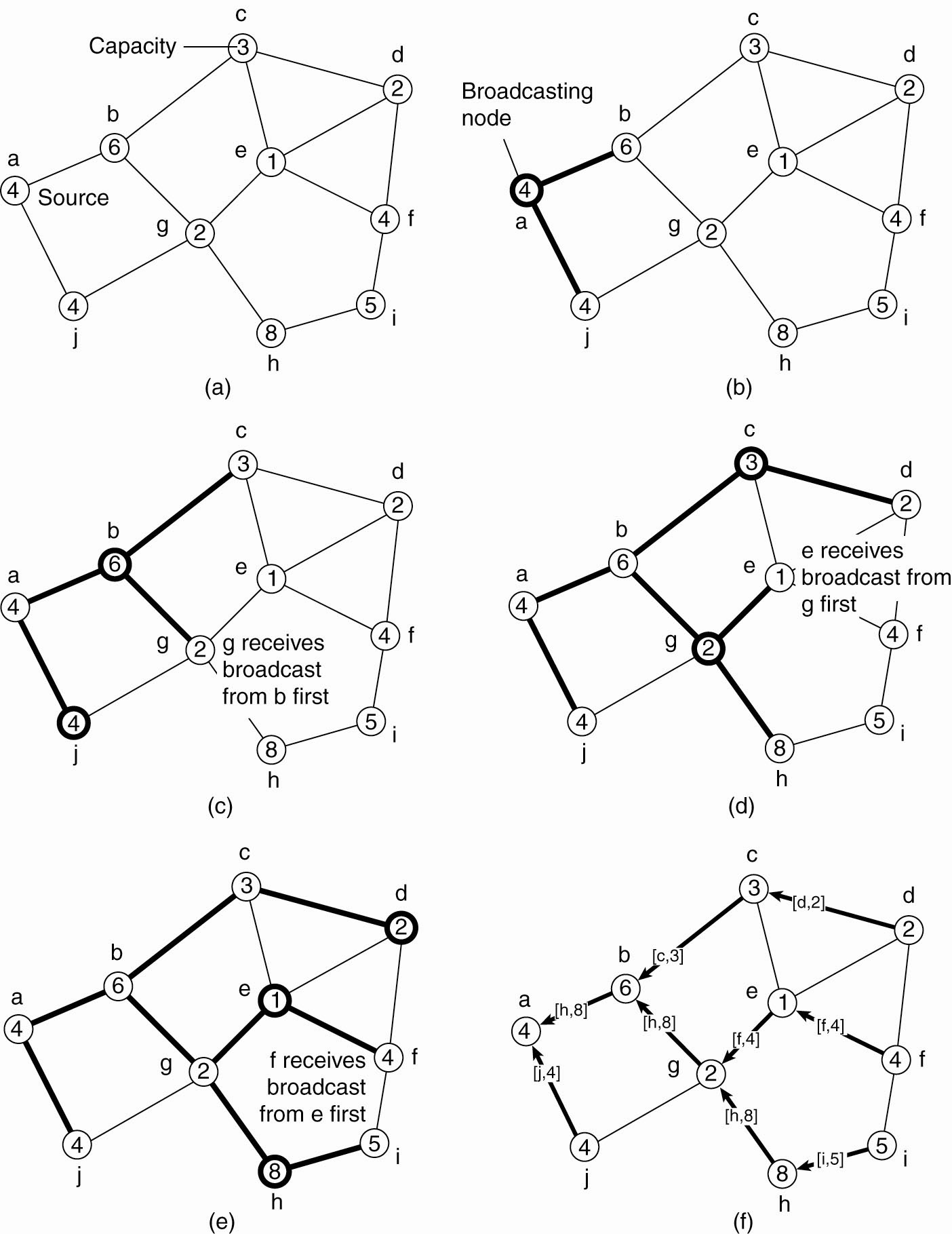
**The Bully Algorithm (2) [1, 2, 3]**

**A Ring Algorithm [1, 2, 3]**

**Elections in Wireless Environments (1) [1, 2, 3]**

**Elections in Wireless Environments (2) [1, 2, 3]**

**Elections in Wireless Environments (3) [1, 2, 3]**

****

**Elections in Large-Scale Systems (1) [1, 2, 3]**

**Requirements for super peer selection:**

1. Normal nodes should have low-latency access to super peers.
2. Super peers should be evenly distributed across the overlay network.
3. There should be a predefined portion of super peers relative to the total number of nodes in the overlay network.
4. Each super peer should not need to serve more than a fixed number of normal nodes.

**Elections in Large-Scale Systems (2) [1, 2, 3]**

**Text & Reference Books**

1. Distributed Systems Principles and Paradigms- A. S. Tanenbaum (2nd Edition), (Pearson Education)

2. Distributed Operating Systems - P. K. Sinha (PHI)

3. Distributed Systems – Concepts & Design by George Coulouris, Jean Dollimore, (Pearson Education)