

Review

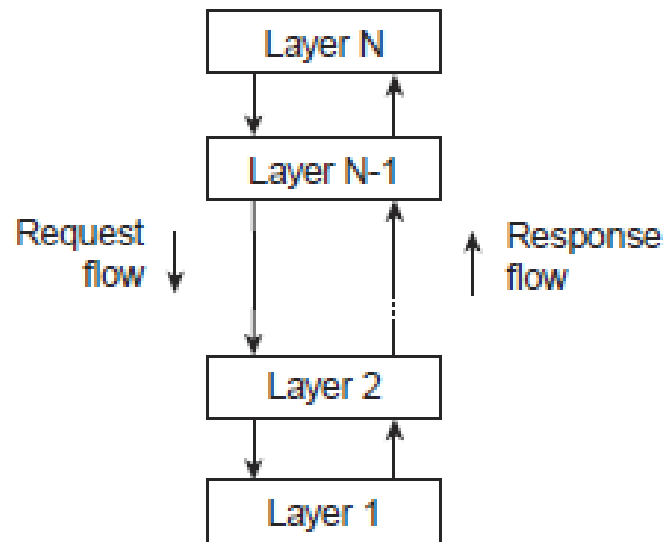
Review

- Distributed system models
 - Layered architectures – role of middleware
 - Client/server
 - Peer-to-peer
 - Multi-tier systems
 - Design principles

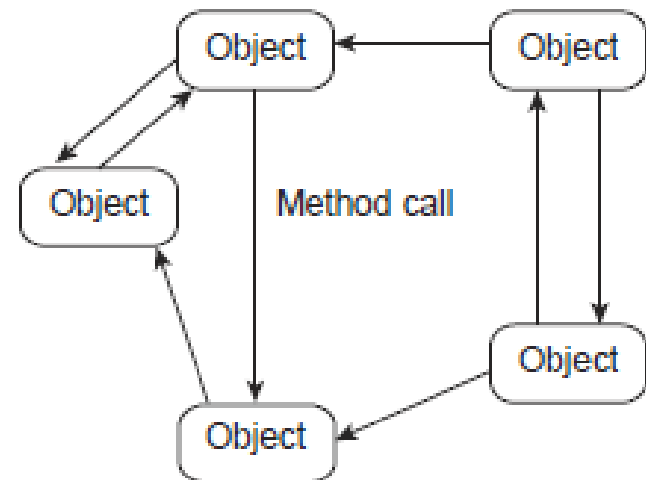
Architectural styles

Basic idea

Organize into **logically different** components, and distribute those components over the various machines.



(a)



(b)

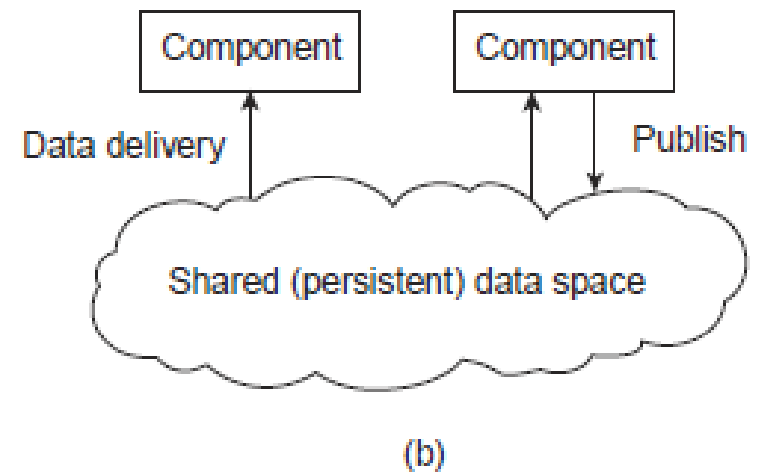
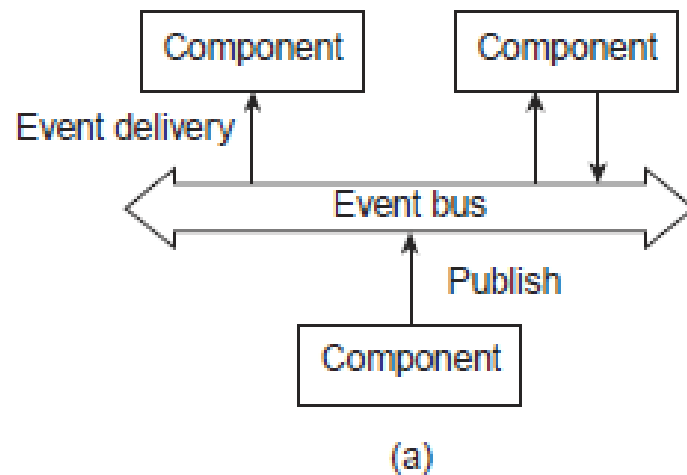
(a) Layered style is used for client-server system

(b) Object-based style for distributed object systems.

Architectural Styles

Observation

Decoupling processes in **space** (“anonymous”) and also **time** (“asynchronous”) has led to alternative styles.

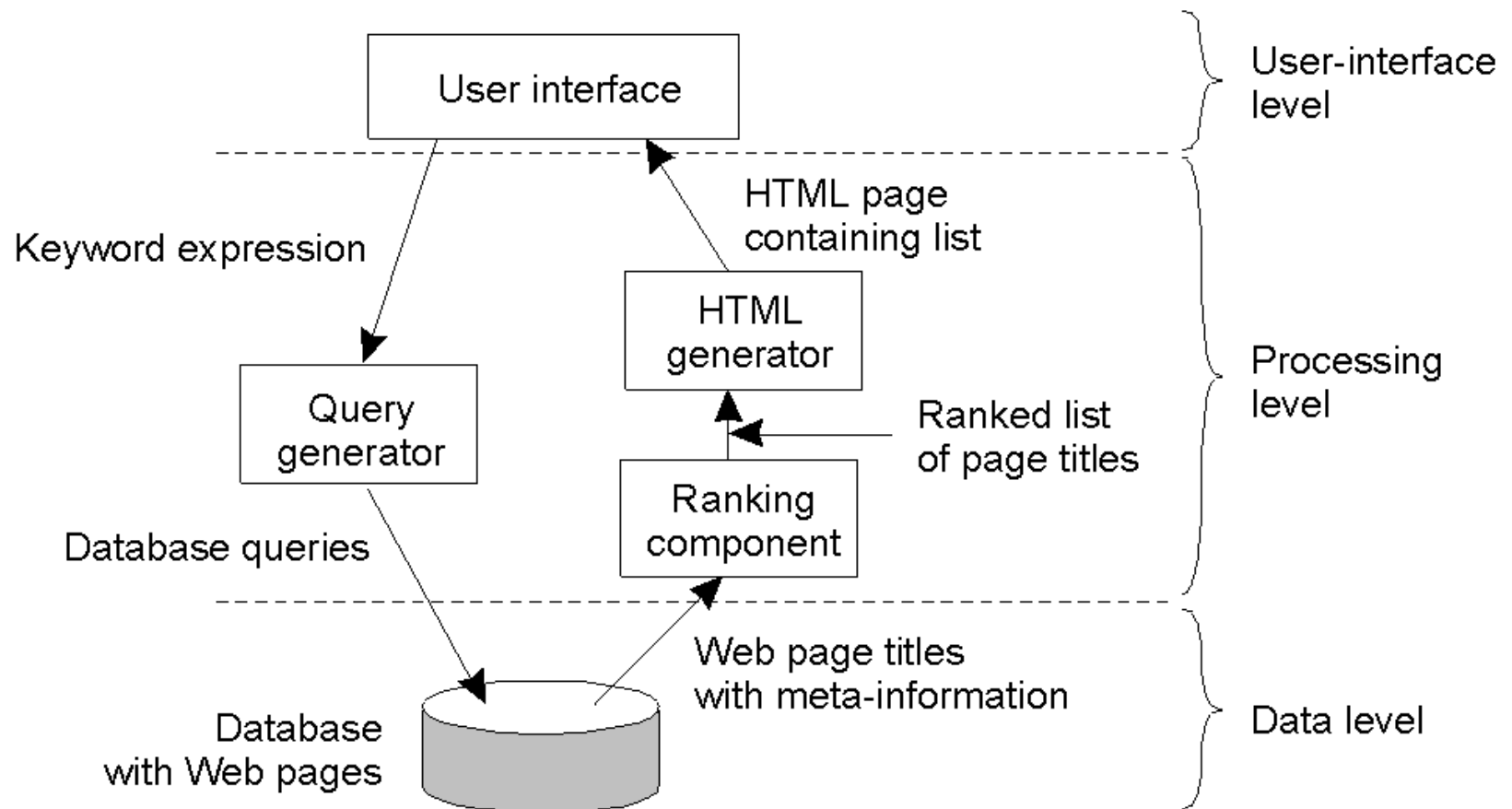


(a) Publish/subscribe [decoupled in **space**]

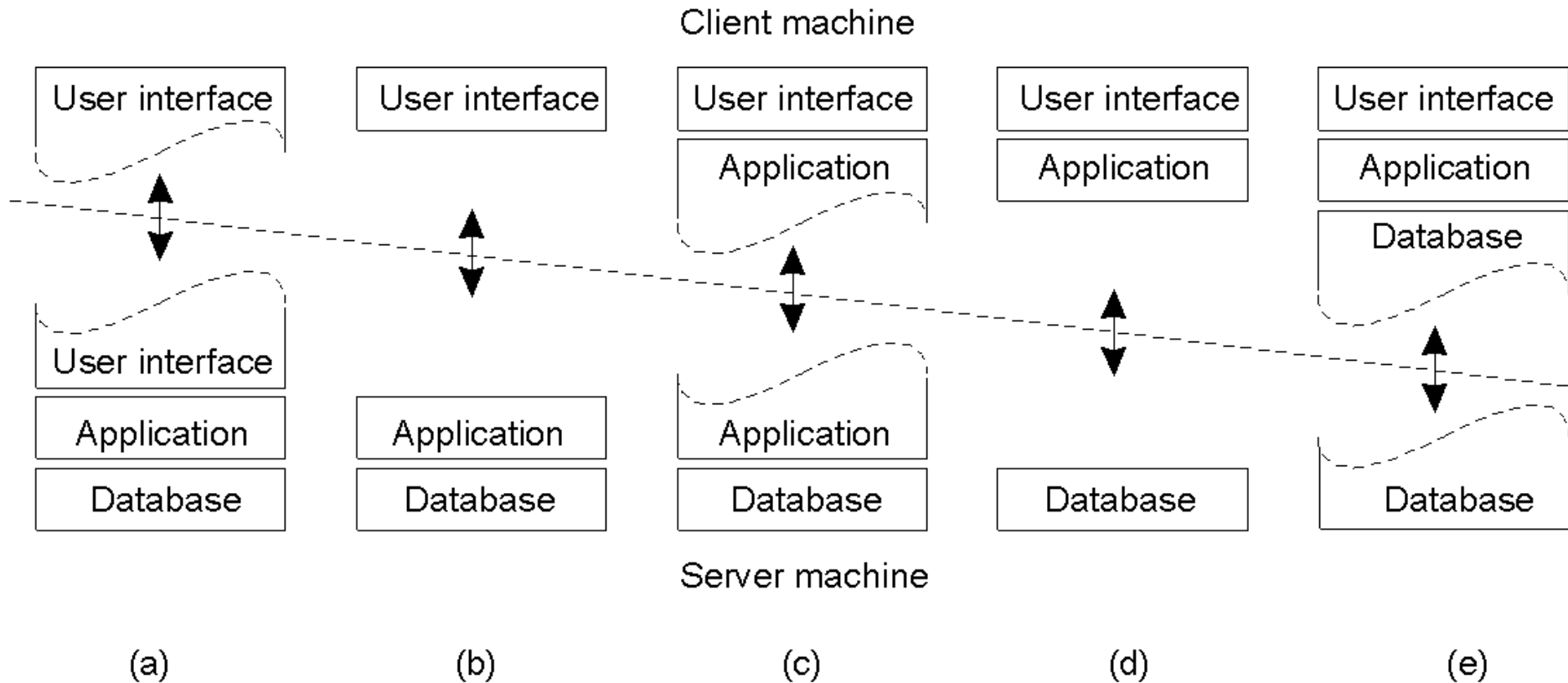
(b) Shared dataspace [decoupled in **space** and **time**]

Multitier Systems

■ Example: Internet Search Engines



Multitier System Alternatives



Review

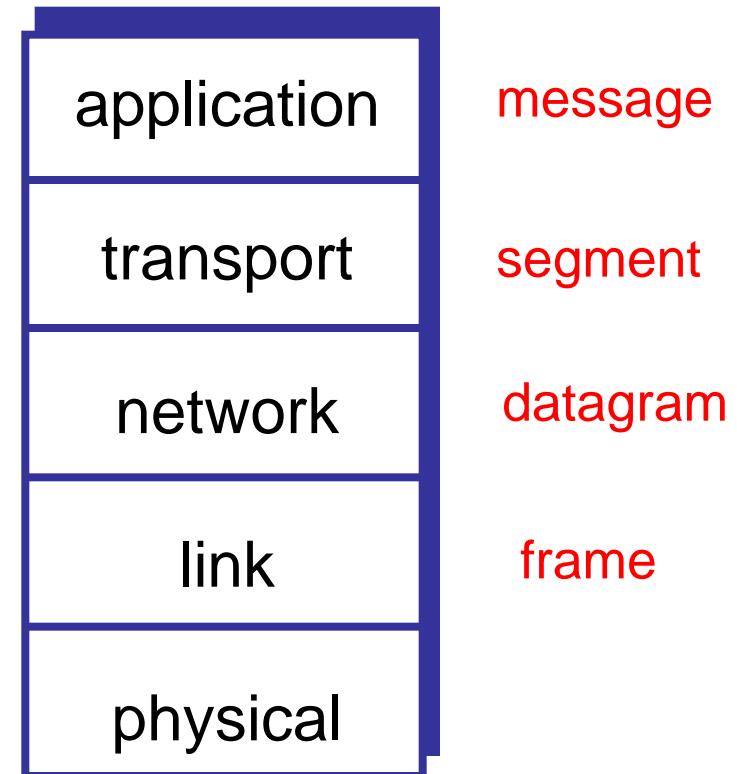
- Computer networks
 - Internet architecture
 - Layered protocols – ISO/OSI
 - Internet protocols – TCP/IP
 - Circuit, packet, virtual circuit switching
 - Routing

OSI Protocol Summary

<i>Layer</i>	<i>Description</i>	<i>Examples</i>
Application	Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.	HTTP, FTP, SMTP, CORBA IIOP
Presentation	Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.	Secure Sockets (SSL), CORBA Data Rep.
Session	At this level reliability and adaptation are performed, such as detection of failures and automatic recovery.	
Transport	This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports attached to processes, Protocols in this layer may be connection-oriented or connectionless.	TCP, UDP
Network	Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through routers. In a single LAN no routing is required.	IP, ATM virtual circuits
Data link	Responsible for transmission of packets between nodes that are directly connected by a physical link. In a WAN transmission is between pairs of routers or between routers and hosts. In a LAN it is between any pair of hosts.	Ethernet MAC, ATM cell transfer, PPP
Physical	The circuits and hardware that drive the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre optic circuits) or other electromagnetic signals (on radio and microwave circuits).	Ethernet base- band signalling, ISDN

Internet Protocol Stack

- **Application:** supporting network applications
 - ftp, smtp, http
- **Transport:** host-host data transfer
 - tcp, udp
- **Network:** routing of datagrams from source to destination
 - ip, routing protocols
- **Link:** data transfer between neighboring network elements
 - ppp, ethernet
- **Physical:** bits “on the wire”



Internet Transport Protocol Services

TCP service:

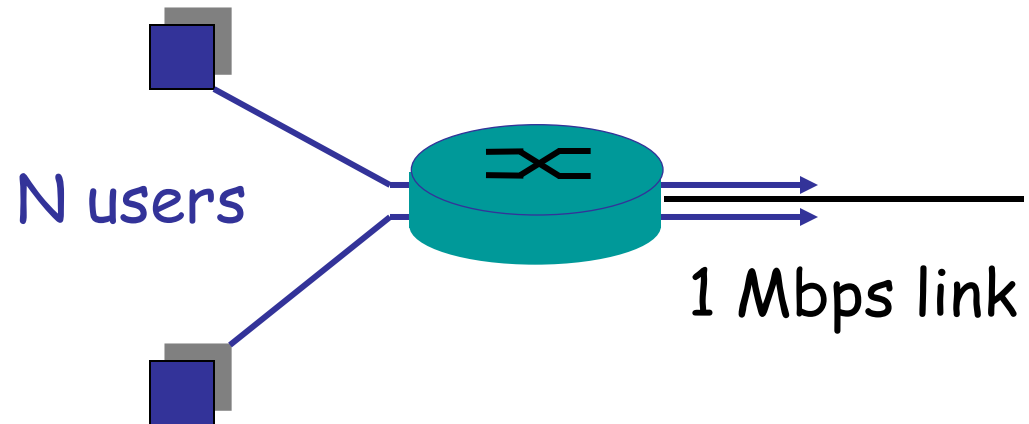
- *connection-oriented*: setup required between client, server
- *reliable transport* between sending and receiving process
 - Receipt of packets returned as acknowledgements
- *flow control*: sender won't overwhelm receiver
- *congestion control*: throttle sender when network overloaded
- *does not provide*: timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Packet Switching vs Circuit Switching

- Packet switching allows more users to use network!
- 1 Mbit link; each user:
 - 100Kbps when “active”
 - active 10% of time
- circuit-switching:
 - 10 users
- packet switching:
 - 35 users,
probability > 10 active less than .004



Packet Switching vs Circuit Switching (2)

- Packet switching is great for bursty data
 - resource sharing
 - no call setup
- It incurs **excessive congestion**: packet delay and loss
 - protocols needed for reliable data transfer, congestion control
- **How to provide circuit-like behavior?**
 - bandwidth guarantees needed for audio/video apps
 - still an unsolved problem, but solutions such as ATM have been developed

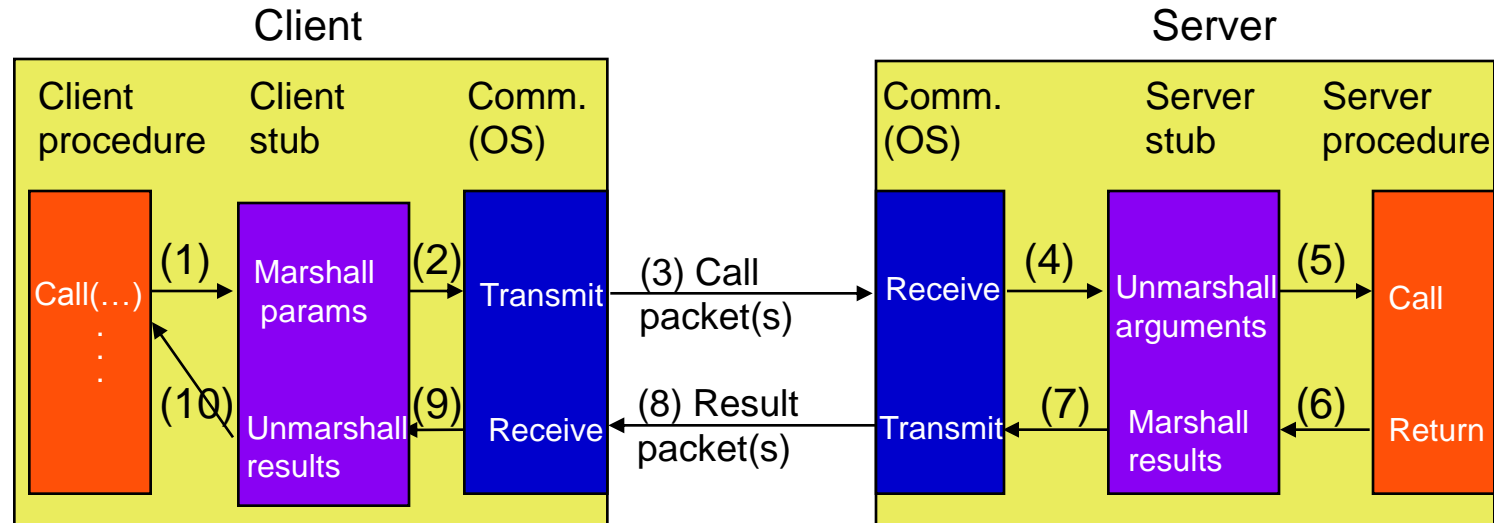
Routing Algorithms

- Two main classifications of routing algorithms
 - Global routing algorithm
 - e.g. Link-state (LS) algorithm
 - Connectivity of all nodes and all link costs serve as inputs
 - Decentralized routing algorithms
 - e.g. Distance-vector algorithm
 - No node has complete information about the costs of all network links

Review

- Distributed objects & remote invocation
 - RPC
 - RMI
 - Failure semantics of these
 - Message-based communication

RPC Operation (2)



1. Client procedure calls the stub as local call
2. Client stub builds the message and calls the local OS
3. Client OS sends msg to server OS
4. Server OS hands over the message to server stub
5. Server stubs unpacks parameters and calls server procedure
6. Server procedure does its work and returns results to server stub
7. Server stub packs results in a message and calls its local OS
8. Server OS sends msg to client OS
9. Client OS hands over the message to client stub
10. Client stubs unpacks result and returns from procedure call

RPC in Presence of Failures

- Five different classes of failures can occur in RPC systems
 - The client is unable to locate the server
 - The request message from the client to the server is lost
 - The reply message from the server to the client is lost
 - The server crashes after receiving a request
 - The client crashes after sending a request

Object Identity (OID) & Referencing

- OID is an invariant property of an object which distinguishes it logically and physically from all other objects.

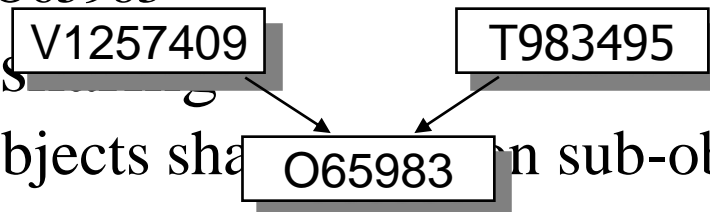
- OIDs are unique.
- OIDs are independent of state.

- Time invariance

- | | | |
|---|-----------------|-----------------|
| ● | OID:O1 | OID: O1 |
| → | VIN: “V1257409” | VIN: “V1257409” |
| → | Make: “Volvo” | Make: “Volvo” |
| → | ... | ... |
| → | Owner: O65983 | Owner: O97603 |

- Referential s

- Multiple objects share sub-objects



Object References

- Should encode:
 - Which object is being referenced
 - The location of the object
 - RMI specific details
 - Which protocol is used (TCP/UDP/HTTP)
 - How is the parameters being marshalled?
 - 📄 Binary, XML, JSON, etc.
- The object reference can even include an implementation handle
 - Specifies the location of the object proxy implementation, which serves the same role as the client stub in RPCs.
 - Enable clients to dynamically add support to protocols and marshalling mechanisms.
 - Can even enable a client to dynamically make use of objects that it wasn't aware of at compile time.

Message-Based Communication

- Lower-level interface to provide more flexibility
- Two (abstract) primitives are used to implement these
 - send
 - receive
- Issues:
 - Are primitives blocking or nonblocking (synchronous or asynchronous)?
 - Are primitives reliable or unreliable (persistent or transient)?

Synchronous/Asynchronous Messaging

■ Synchronous

- The sender is blocked until its message is stored in the local buffer at the receiving host or delivered to the receiver.

■ Asynchronous

- The sender continues immediately after executing a send
- The message is stored in the local buffer at the sending host or at the first communication server.

Transient/Persistent Messaging

■ Transient

- The sender puts the message on the net and if it cannot be delivered to the sender or to the next communication host, it is lost.
- There can be different types depending on whether it is asynchronous or synchronous

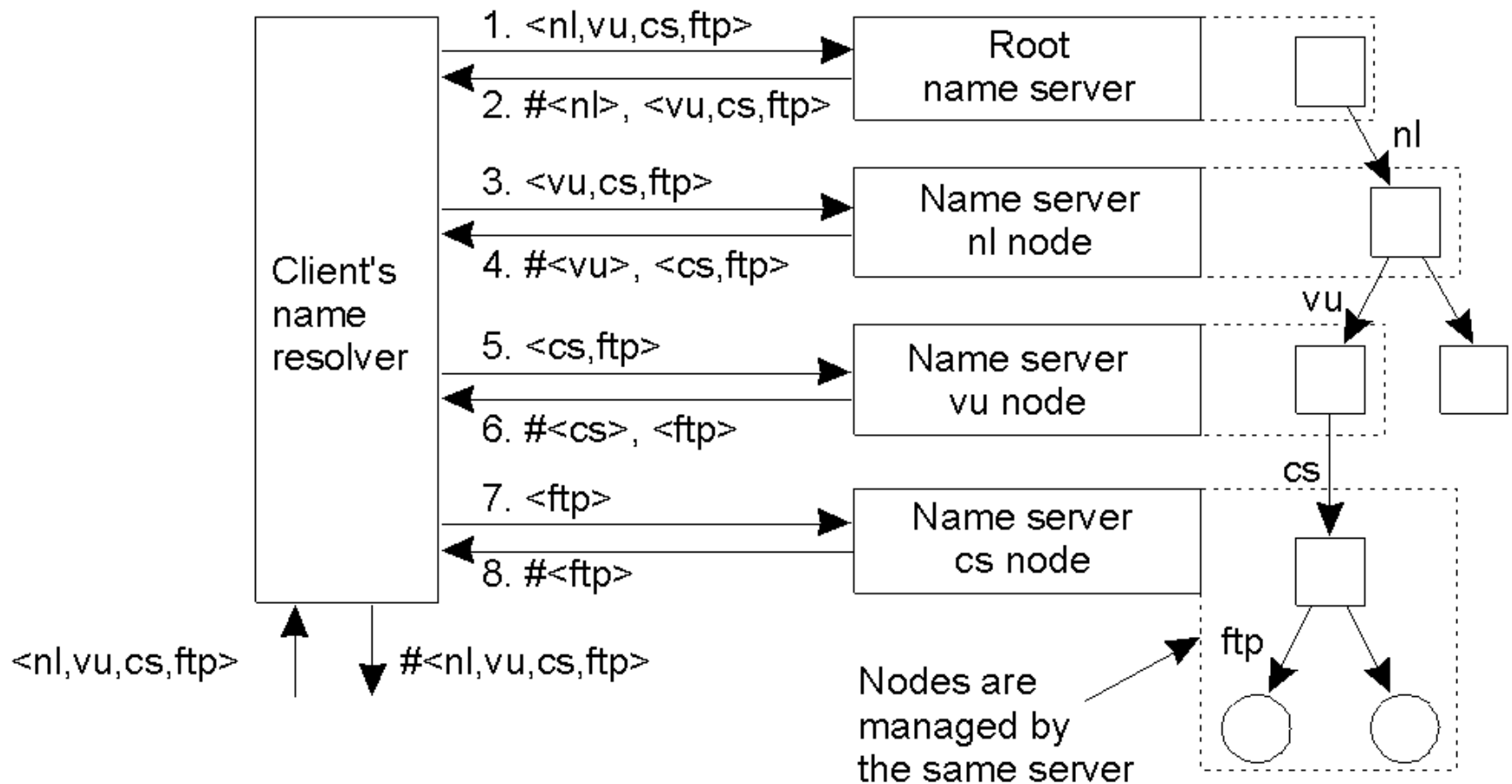
■ Persistent

- The message is stored in the communication system as long as it takes to deliver the message to the receiver

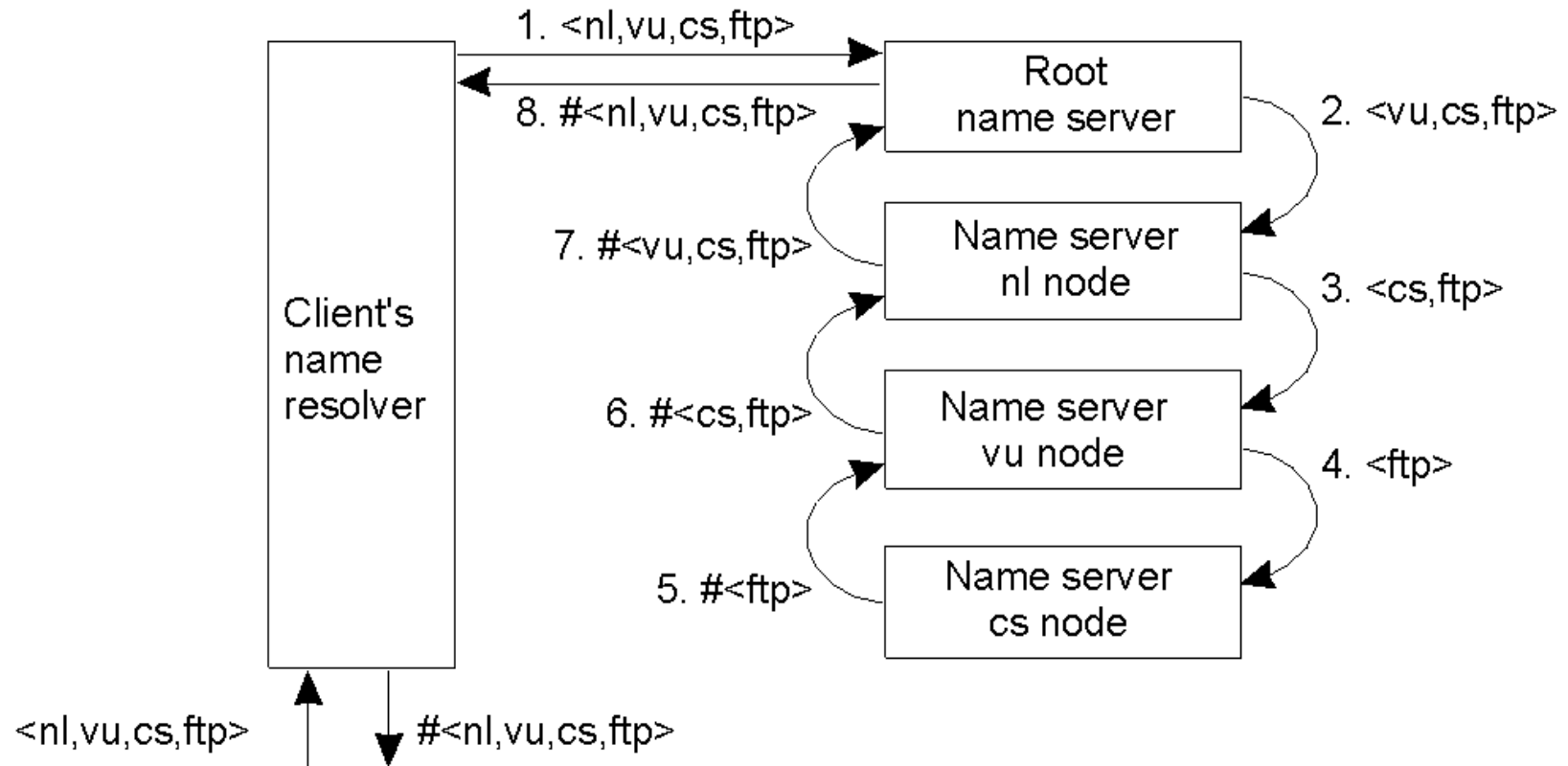
Review

- Distributed name systems
 - Naming and name resolution
 - Name services
 - Directory services

Iterative Navigation Example

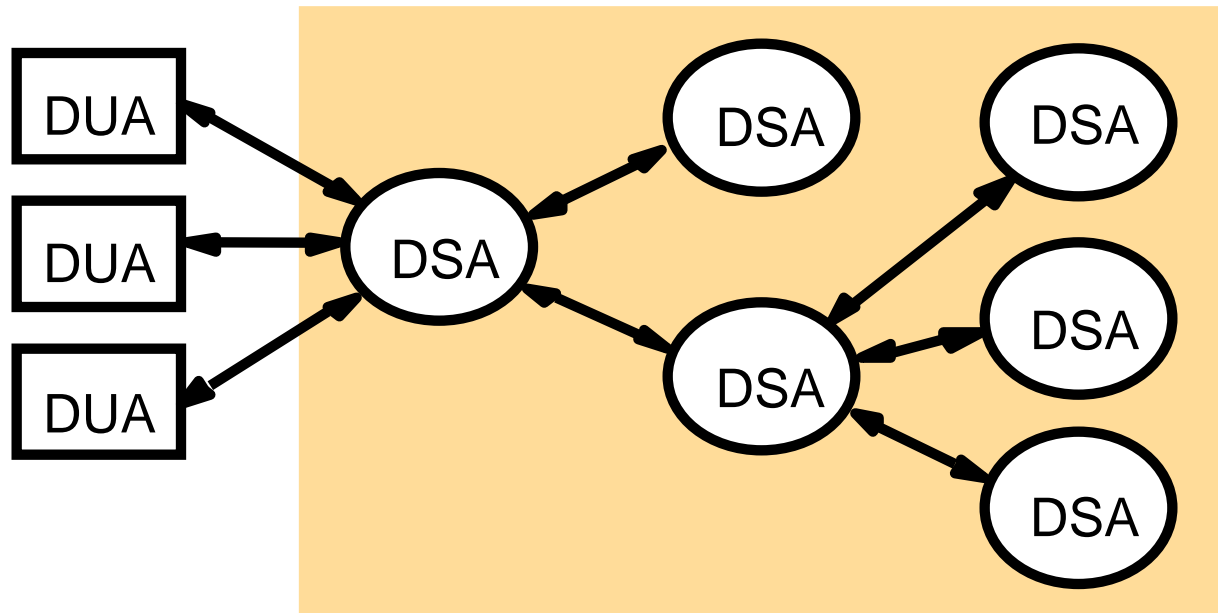


Recursive Navigation Example



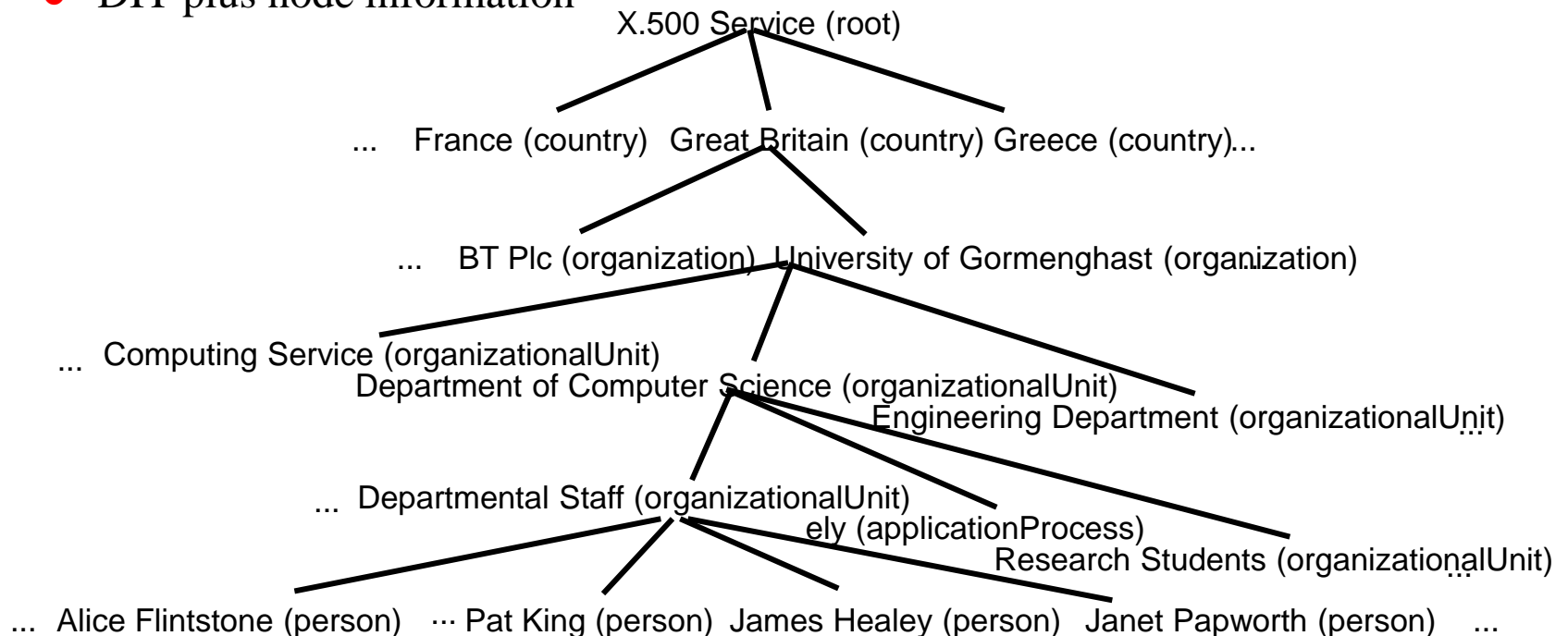
X.500 Directory Service

- Geared towards satisfying descriptive queries
 - Provide attributes of other users and system resources
- Architecture
 - Directory user agents (DUA)
 - Directory service agents (DSA)



X.500 Name Tree

- Directory Information Tree (DIT)
 - Every node of the tree stores extensive information
- Directory Information Base (DIB)
 - DIT plus node information



Discovery Services

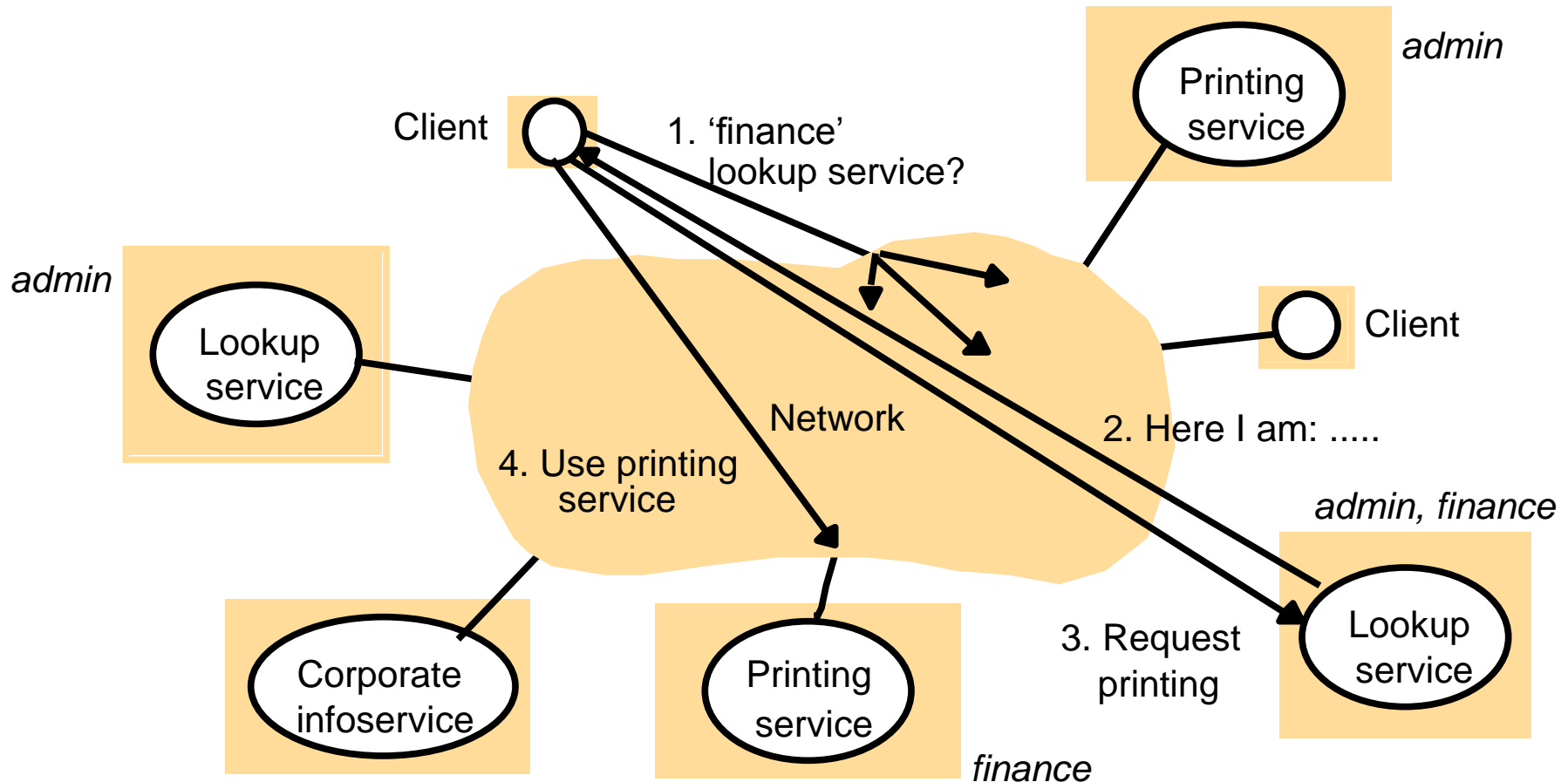
- Directory services that allow clients to query available services in a spontaneous networking environment
 - e. g., which are the available color printers
 - services enter their data using registration interfaces
 - structure usually rather flat, since scope limited to (wireless) LAN
- JINI (<http://www.sun.com/jini/>)
 - Now known as Apache River
 - JAVA-based discovery service
 - clients and servers run JVMs
 - ☞ communication via Java RMI
 - ☞ dynamic loading of code

JINI Discovery Related Services

- Lookup service
 - Holds information regarding available services
- Query of lookup service by Jini client
 - Match request
 - Download object providing service from lookup service
- Registration of a Jini client or Jini service with lookup service
 - Send message to well-known IP multicast address, identical to all Jini instances
 - Limit multicast to LAN using time-to-live attribute
 - Use of leases that need to be renewed periodically for registering Jini services

JINI Discovery Scenario

- new client wishes to print on a printer belonging to the finance group



Review

- Distributed file systems
 - Functions and architecture
 - NFS (V.3 and V.4)
 - AFS
 - Caching and sharing

Caching in Sun NFS (2)

■ Client caching

- Caching of read , write, getattr, lookup and readdir operations
- Potential inconsistency: the data cached in client may not be identical to the same data stored on the server
- Timestamp-based scheme used in polling server about freshness of a data object (presumption of synchronized global time, e.g., through NTP)
 - T_c : time cache entry was last validated
 - $Tm_{\text{client/server}}$: time when block was last modified at the server as recorded by client/server
 - t : freshness interval
 - Freshness condition: at time T

$$\text{☞ } [(T - T_c) < t] \vee [Tm_{\text{client}} = Tm_{\text{server}}]$$

☞ if $(T - T_c) < t$ (can be determined without server access), then entry presumed to be valid

☞ if not $(T - T_c) < t$, then Tm_{server} needs to be obtained by a **getattr** call

☞ if $Tm_{\text{client}} = Tm_{\text{server}}$, then entry presumed valid; update its T_c to current time, else obtain data from server and update Tm_{client}


Caching in Sun NFS (3)

■ Client caching - write operations

- Mark modified cache page as “dirty” and schedule page to be flushed to server (asynchronously)

- Flush happens with closing of file, or when sync is issued,


- When asynchronous block input/output (bio) daemon is used and active

-  when read, then read-ahead: when read occurs, bio daemon sends next file block

-  when write, then bio daemon will send block asynchronously to server

- Bio daemons: performance improvement reducing probability that client blocks waiting for

-  read operations to return, or

-  write operations to be committed at the server

Andrew File System (AFS)

- Started as a joint effort of Carnegie Mellon University and IBM
- Today basis for DCE/DFS: the distributed file system included in the Open Software Foundations' Distributed Computing Environment
- Some UNIX file system usage observations, as pertaining to caching
 - Infrequently updated shared files and local user files will remain **valid for long periods of time** (the latter because they are being updated on owners workstations)
 - Allocate large local disk cache, e. g., 100 Mbyte. This is sufficient for the establishment of a **working set** of the files used by one user and will ensure that files are still in this cache when needed next time

Andrew File System (AFS)

- Some UNIX file system usage observations, as pertaining to caching (continued)
 - Assumptions about typical file accesses (based on empirical evidence)
 - usually **small files**, less than 10 Kbytes
 - reads much more common than writes (appr. 6: 1)
 - usually **sequential access**, random access not frequently found
 - user-locality: most files are used by **only one user**
 - **burstiness** of file references: once file has been used, it will be used in the nearer future with high probability
- Design decisions for AFS
 - **Whole-file serving**: entire contents of directories and files transferred from server to client (AFS-3: in chunks of 64 Kbytes)
 - **Whole file caching**: when file transferred to client it will be stored on that client's local disk

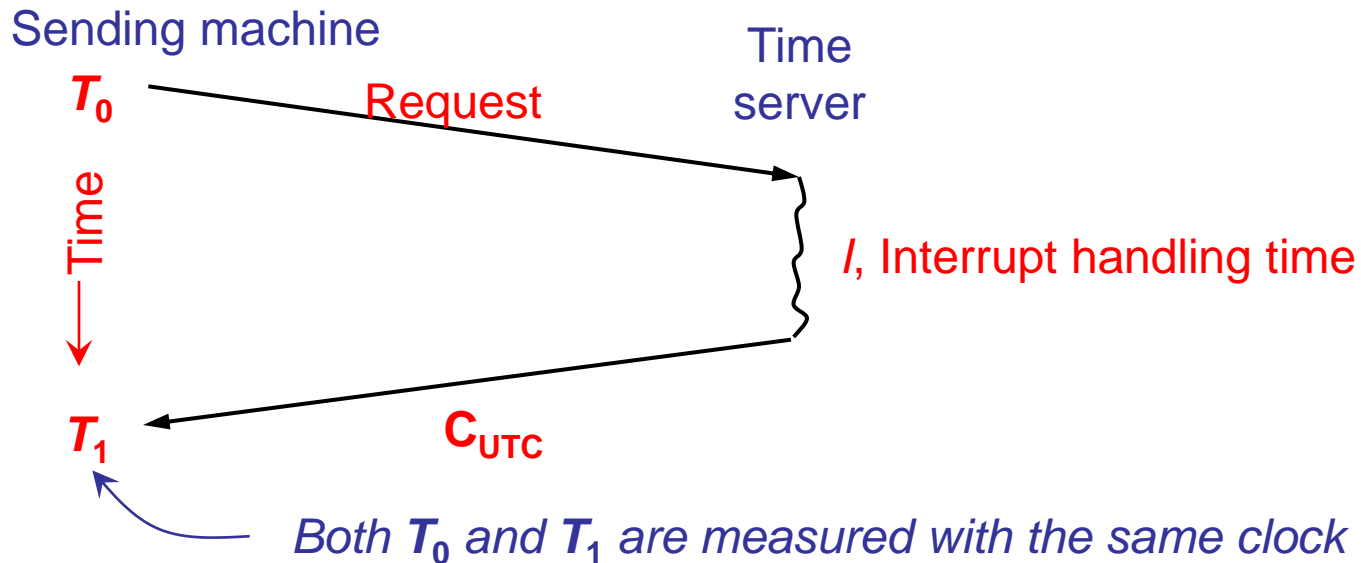
Review

■ Synchronization

- Time and clocks
 - Physical clock synchronization
 - Logical clock synchronization
- Establishing global state and election protocols
- Transactions and concurrency control

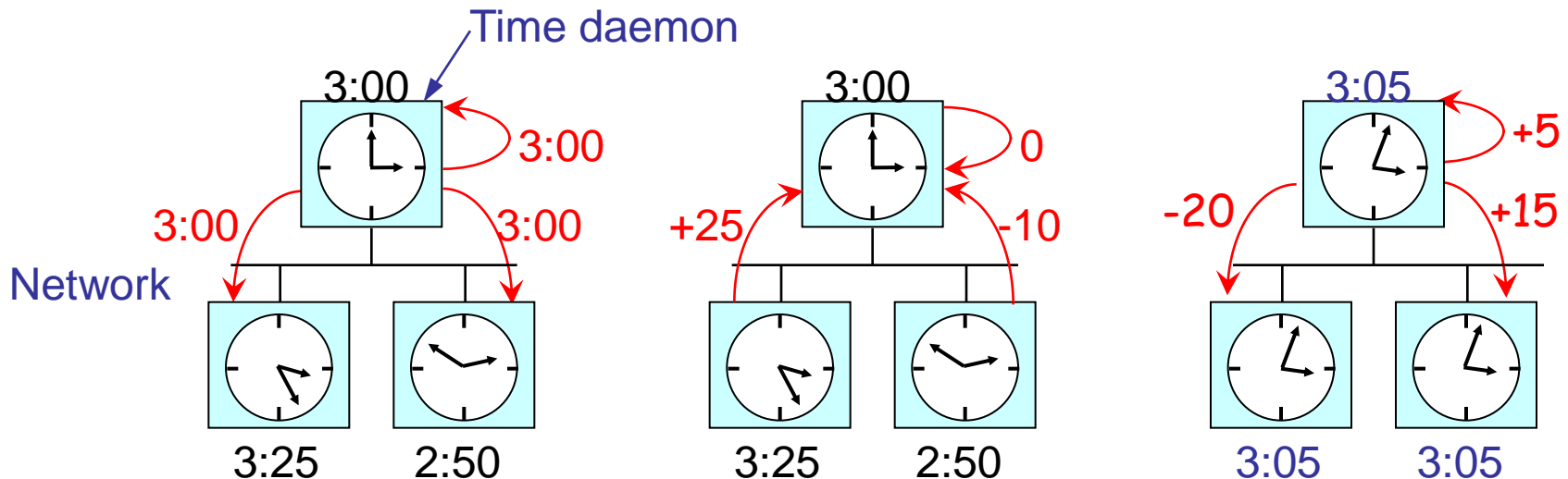
Cristian's Algorithm

- One time server (WWV receiver); all other machines stay synchronized with the time server.
- Cannot set T_1 to C_{UTC} because time must never run backwards. Changes are introduced gradually by adding more or less seconds for each interrupt.
- The propagation time is included in the change.
 - Estimated as: $(T_1 - T_0 - I)/2$ (can be improved by continuous probing & averaging)



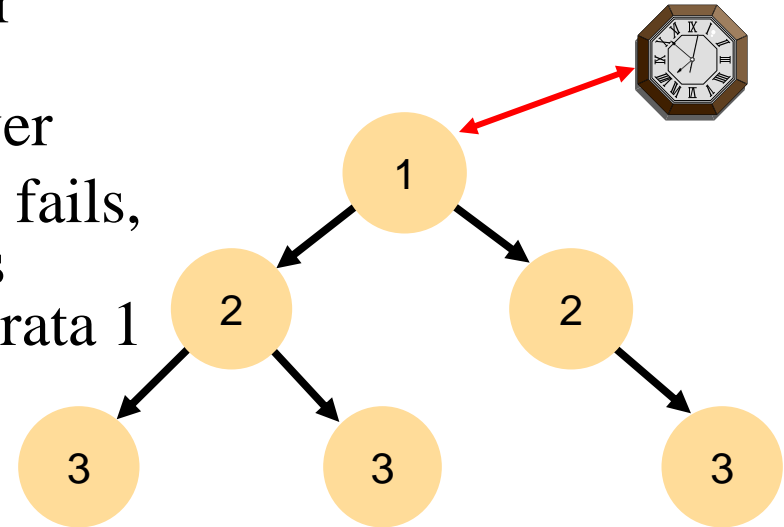
Berkeley Algorithm

- Suitable when no machine has a WWV receiver
- The time server (a time daemon) is active:
 - Time daemon polls every machine periodically to ask what time is there
 - Based on the answers, it computes an average time
 - Tells all other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved
- The time daemon's time is set manually by operator periodically



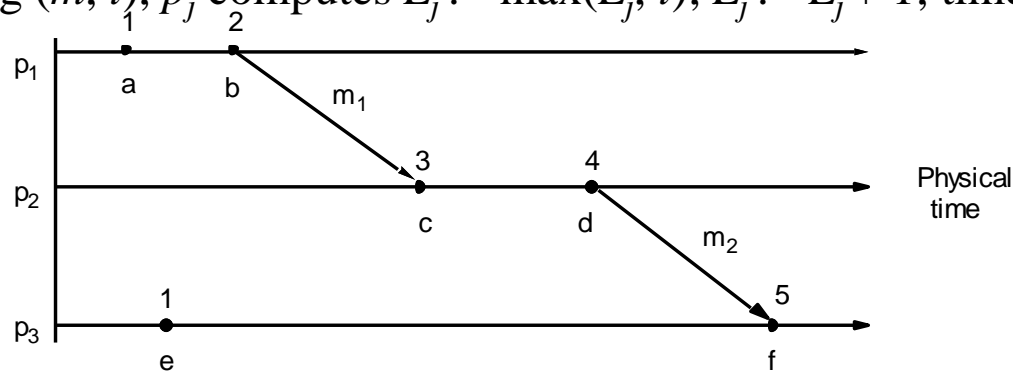
The Internet Network Time Protocols (NTP)

- Layered client-server architecture, based on UDP message passing
- Synchronization at clients with higher strata number less accurate due to increased latency to strata 1 time server
- Failure robustness: if a strata 1 server fails, it may become a strata 2 server that is being synchronized through another strata 1 server
- Modes:
 - Multicast
 - One computer periodically transmits time
 - Procedure call
 - Similar to Cristian's algorithm
 - Symmetric
 - To be used where high accuracy is needed



Lamport's Algorithm

- Capturing happens-before relation
- Each process p_i has a local monotonically increasing counter, called its **logical clock** L_i
- Each event e that occurs at process p_i is assigned a Lamport timestamp $L_i(e)$
- Rules:
 - L_i is incremented before event e is issued at p_i : $L_i := L_i + 1$
 - When p_i sends message m , it adds $t = L_i$: (m, t) [this is event $send(m)$]
 - On receiving (m, t) , p_j computes $L_j := \max(L_j, t)$; $L_j := L_j + 1$; timestamp event $receive(m)$

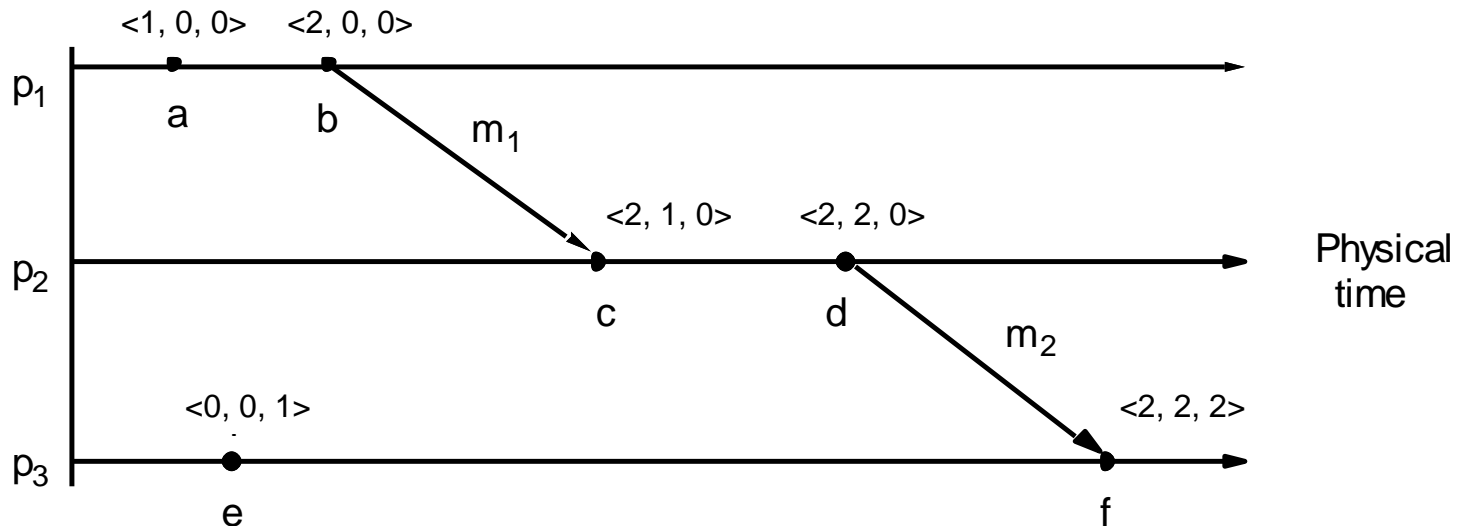


Vector Timestamp

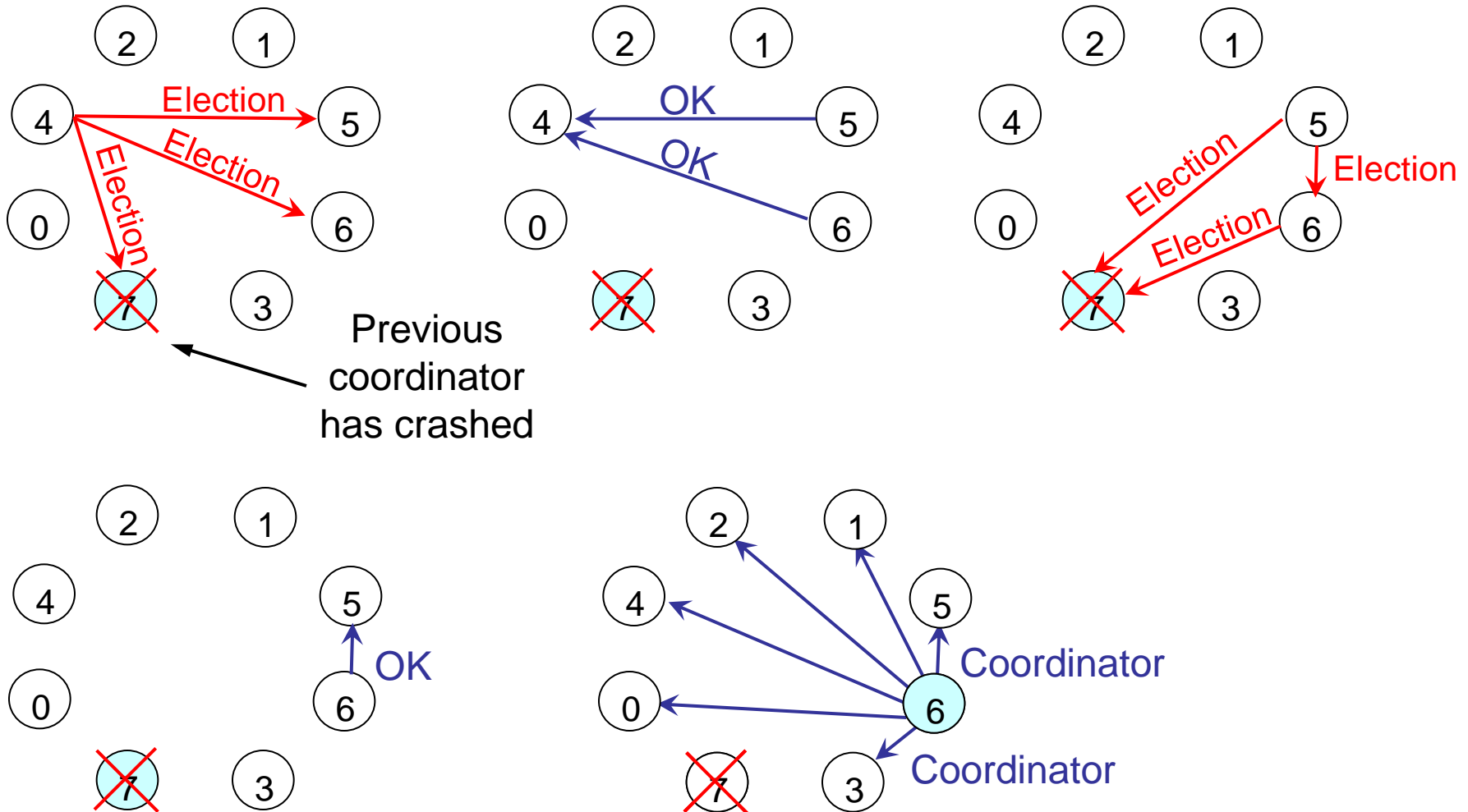
- With Lamport timestamps:
 - $e \rightarrow f$ then $\text{timestamp}(e) < \text{timestamp}(f)$
 - But $\text{timestamp}(e) < \text{timestamp}(f)$ does not imply that $e \rightarrow f$
 - Does not capture causality
- Vector timestamp:
 - All hosts use a vector of logical clocks, where the i -th element is the clock value for the i -th host
 - Timestamp elements for other hosts are initially set to zero

Vector Timestamp

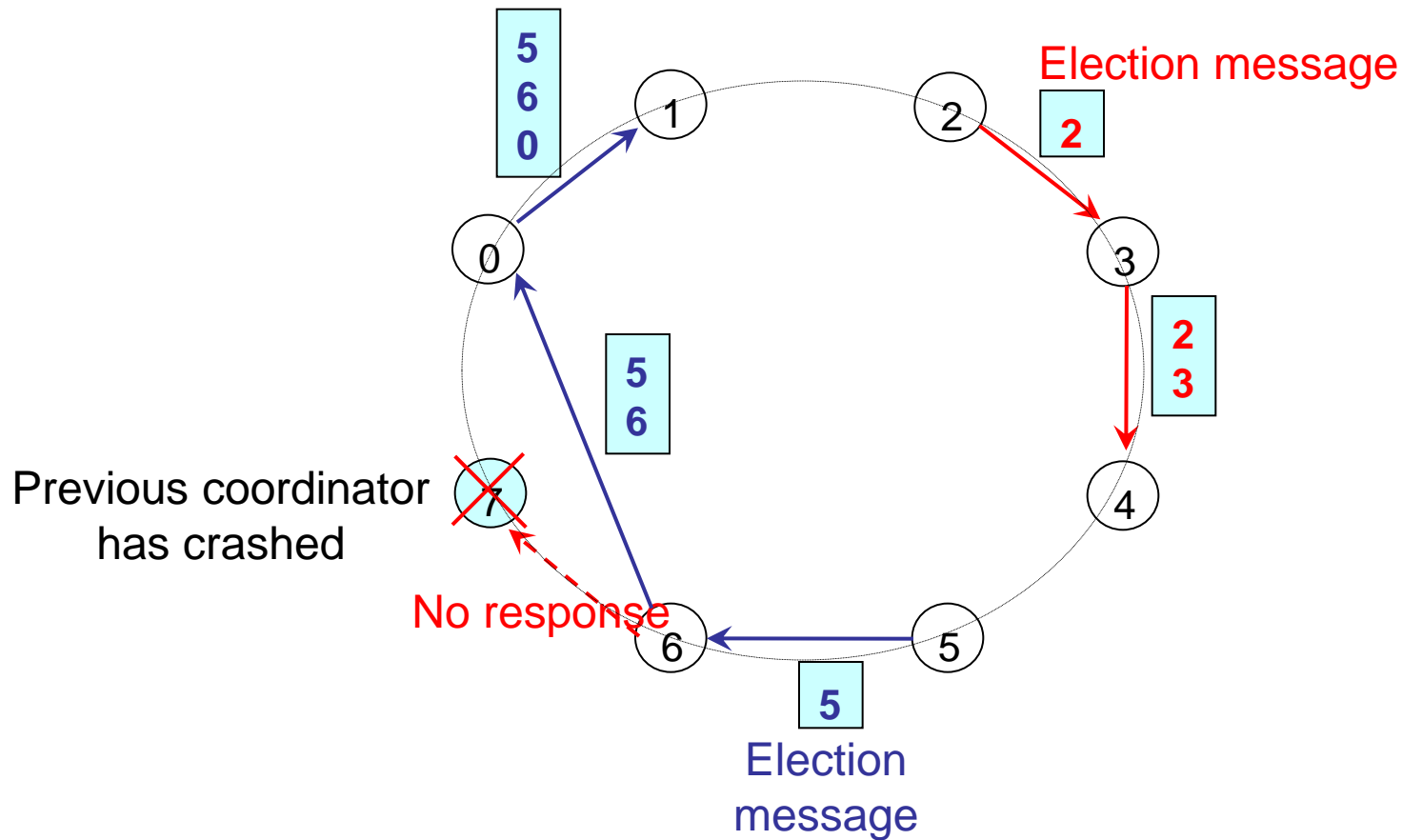
- For every non-communication event on host i
 - Increment the i -th element in the vector
- Send message sends the entire vector
- On receiving a message from host j (on host i), update vector j by:
 - Incrementing the i -th element
 - For the remaining elements, take the max value between vector i and j



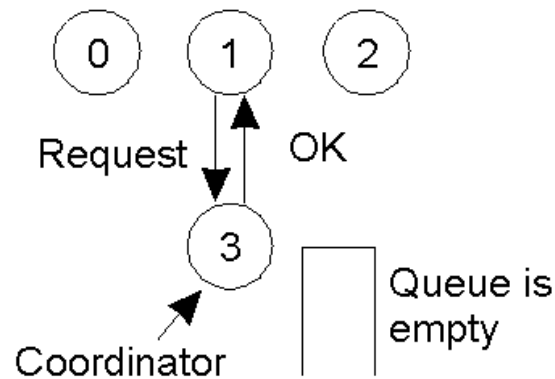
Bully



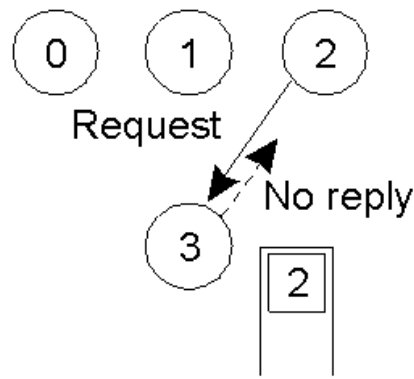
Ring



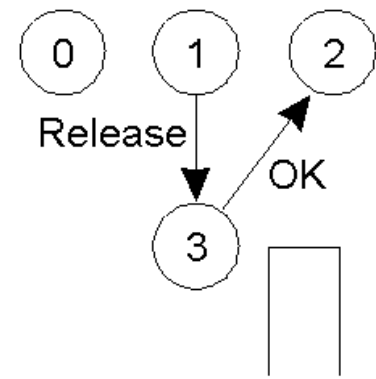
A Centralized Algorithm (Central Server Algorithm)



(a)



(b)

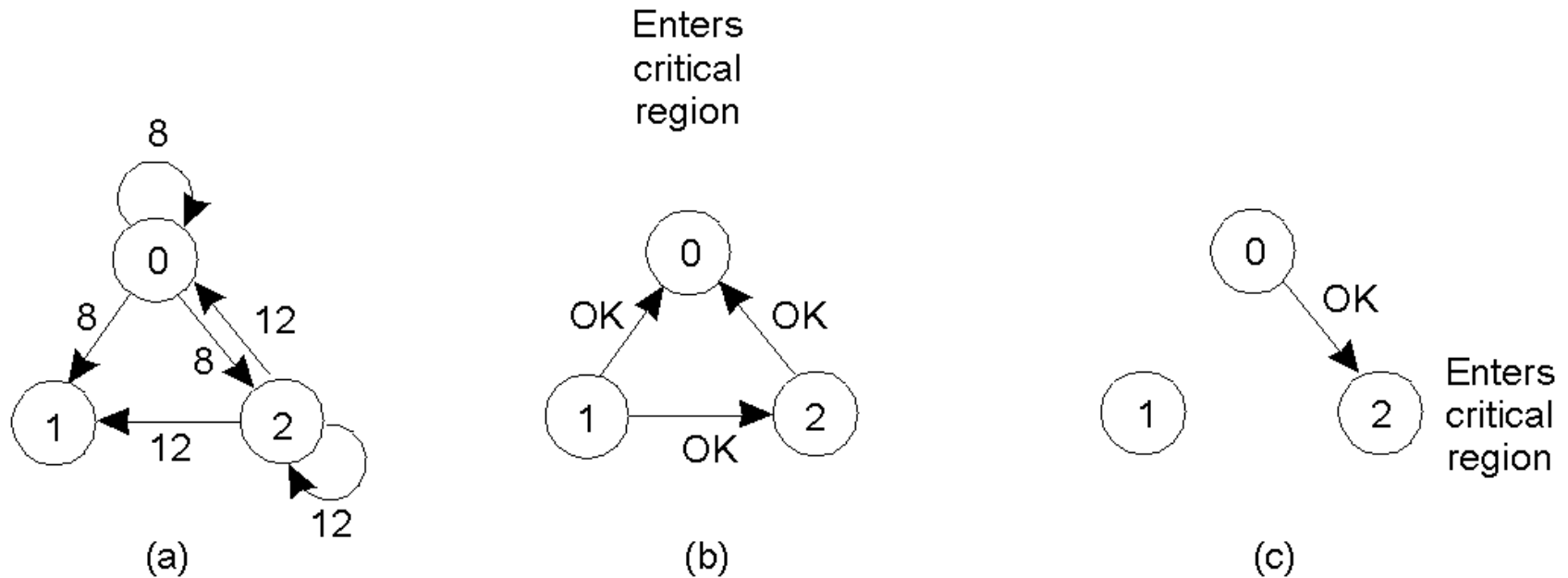


(c)

One process is elected as the **coordinator**:

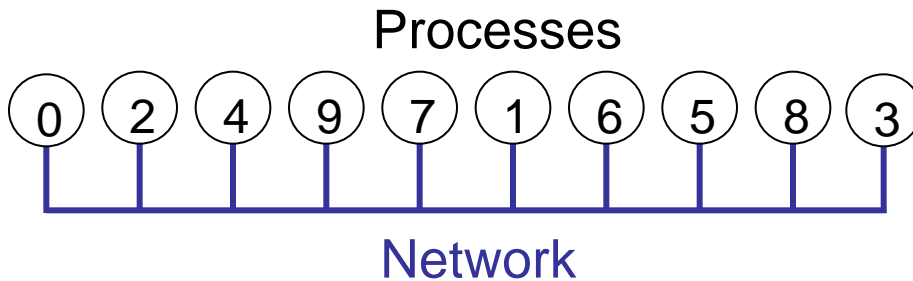
- a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
- b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
- c) When process 1 exits the critical region, it tells the coordinator, when then replies to 2

Distributed

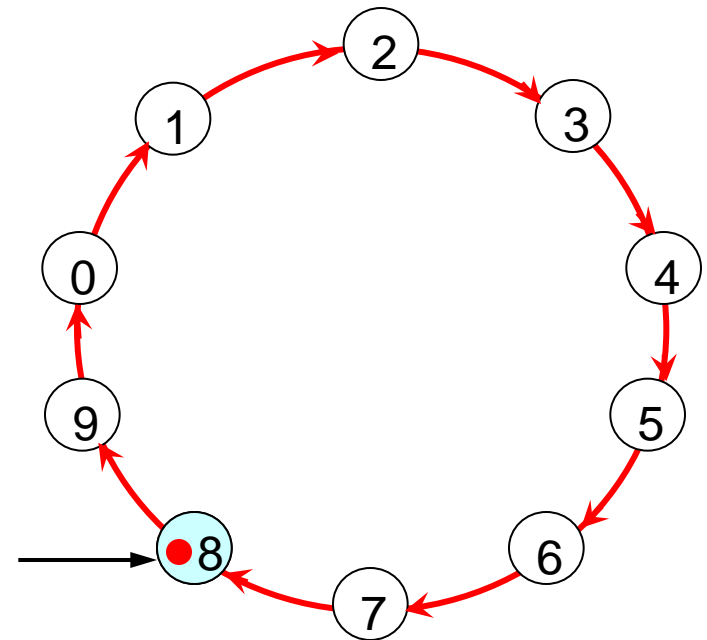


- a) Two processes want to enter the same critical region at the same moment.
- b) Process 0 has the lowest timestamp, so it wins.
- c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

A Token Ring Algorithm



Token holder may enter critical region or pass the token



Comparison of the 3 Algorithms

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	$2(n - 1)$	$2(n - 1)$	Crash of any process
Token ring	1 to ∞	0 to $n - 1$	Lost token, process crash

- **Centralized Algorithm:** Simplest and most efficient. A request and a grant to enter, and a release to exit. Only 2 message times to enter.
- **Decentralized Algorithm:** Requires $n-1$ request messages, one to each of the other processes, and an additional $n-1$ grant messages. Entry takes $2(n-1)$ message times assuming that messages are passed sequentially over a LAN
- **Token Ring Algorithm:** If every process constantly wants to enter a critical region (each token pass will result in one entry). The token may sometimes circulate for hours without anyone being interested in it (number of messages per entry is unbounded). Entry delay varies from 0 (token just arrived) to $n-1$ (token just departed)

Deadlock Management

■ Prevention

- Guaranteeing that deadlocks can never occur in the first place. Check transaction when it is initiated. Requires no run time support.
- All resources which may be needed by a transaction must be predeclared

■ Avoidance

- Detecting potential deadlocks in advance and taking action to insure that deadlock will not occur. Requires run time support.
- Order either the objects or the sites and always request locks in that order.

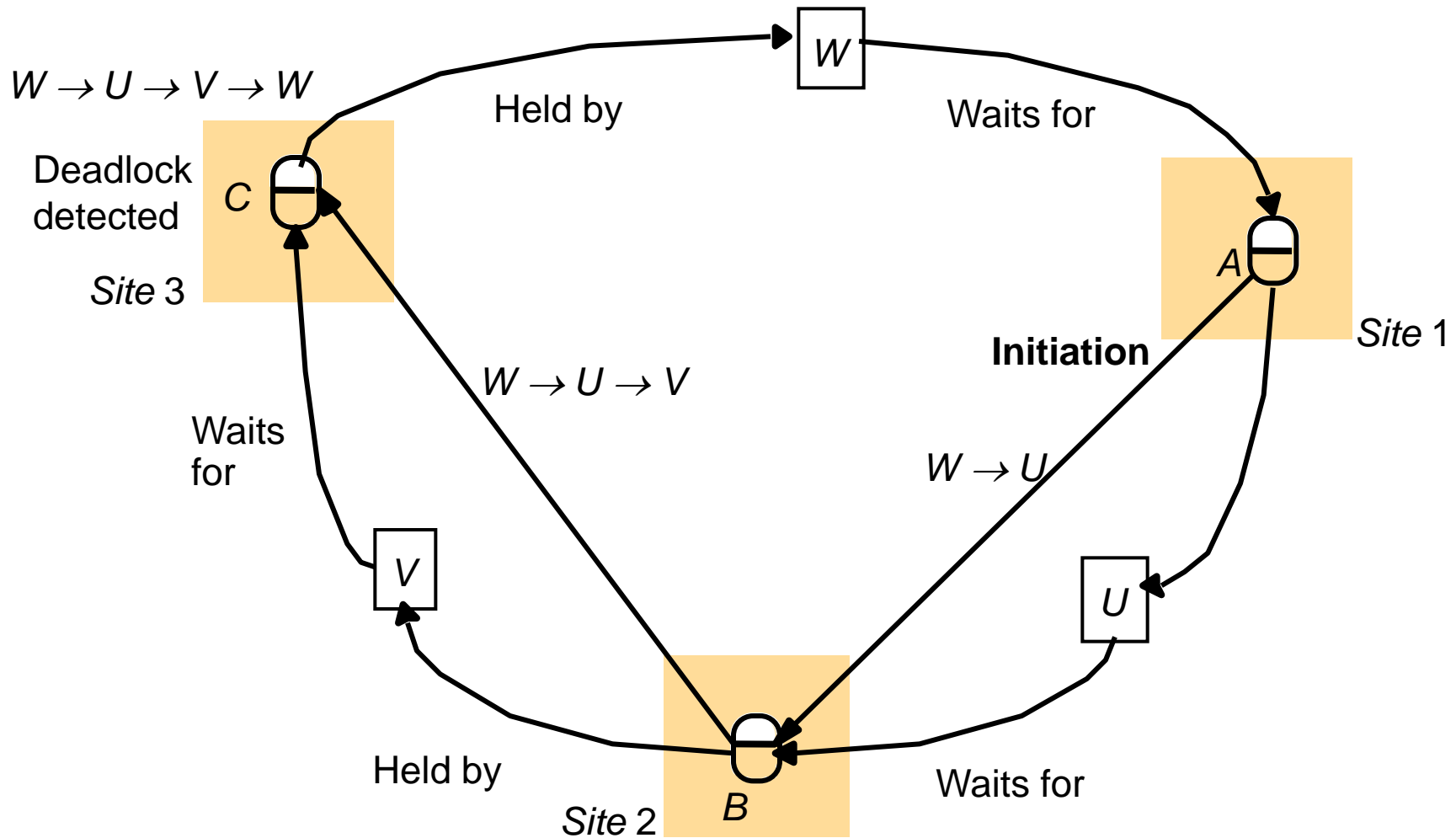
■ Detection and Recovery

- Allowing deadlocks to form and then finding and breaking them. As in the avoidance scheme, this requires run time support.

Distributed Deadlock Detection

- Sites cooperate in detection of deadlocks.
- One example:
 - The local WFGs are formed at each site and passed on to other sites. Each local WFG is modified as follows:
 - ☆ Since each site receives the potential deadlock cycles from other sites, these edges are added to the local WFGs
 - 🕒 The edges in the local WFG which show that local transactions are waiting for transactions at other sites are joined with edges in the local WFGs which show that remote transactions are waiting for local ones.
 - Each local deadlock detector:
 - looks for a cycle that does not involve the external edge. If it exists, there is a local deadlock which can be handled locally.
 - looks for a cycle involving the external edge. If it exists, it indicates a **potential** global deadlock. Pass on the information to

Probing

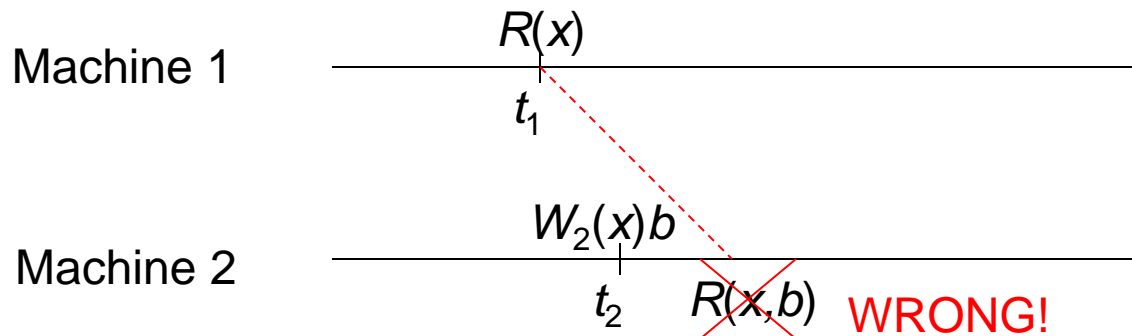


Review

- Replication
 - Replica consistency models
 - Update propagation
 - Replication protocols
 - Primary-based
 - Write-all

Strict Consistency

- Any *read*(*x*) returns a value corresponding to the result of the most recent *write*(*x*).



- Relies on absolute global time; all writes are instantaneously visible to all processes and an absolute global time order is maintained.
- Cannot be implemented in a distributed system

P1:	$W(x)a$	
P2:		$R(x)a$
Strictly consistent		

P1:	$W(x)a$	
P2:	$R(x)NIL$	$R(x)a$
Not strictly consistent		

Linearizability

- The result of the execution should satisfy the following criteria:
 - Read and write by all processes were executed in some serial order and each process's operations maintain the order of specified;
 - If $ts_{op_1}(x) < ts_{op_2}(y)$ then $op_1(x)$ should precede $op_2(y)$ in this sequence. This specifies that the order of operations in interleaving is consistent with the real times at which the operations occurred in the actual implementation.
- Requires synchronization according to timestamps, which makes it expensive.
- Used only in formal verification of programs.

Sequential Consistency

- Similar to linearizability, but no requirement on timestamp order.
- The result of execution should satisfy the following criteria:
 - Read and write operations by all processes on the data store were executed in some sequential order;
 - Operations of each individual process appear in this sequence in the order specified by its program.

■ The

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

Sequentially consistent

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

Not sequentially consistent

ity.

Epidemic Protocols

■ Anti-entropy

- Node P picks another node Q at random and exchanges updates with Q.
- P pushes and/or pulls updates to/from Q.
- Number of rounds to propagate a single update to all nodes is $O(\log N)$
 - A round is the period where every node would have at least once initiated an exchange with another node.

■ Gossip

- Node P has just been updated with data item x
- It tries to push the update to a random node Q
 - If Q has already heard the rumor, then P may lose interest in spreading the rumor (assume with probability $1/k$).
- Probabilistic protocol:
 - $s = e^{-(k+1)(1-s)}$ will remain susceptible but will never receive the update.
 - For $k=4$, then s is less than 0.007

■ How do you remove data?

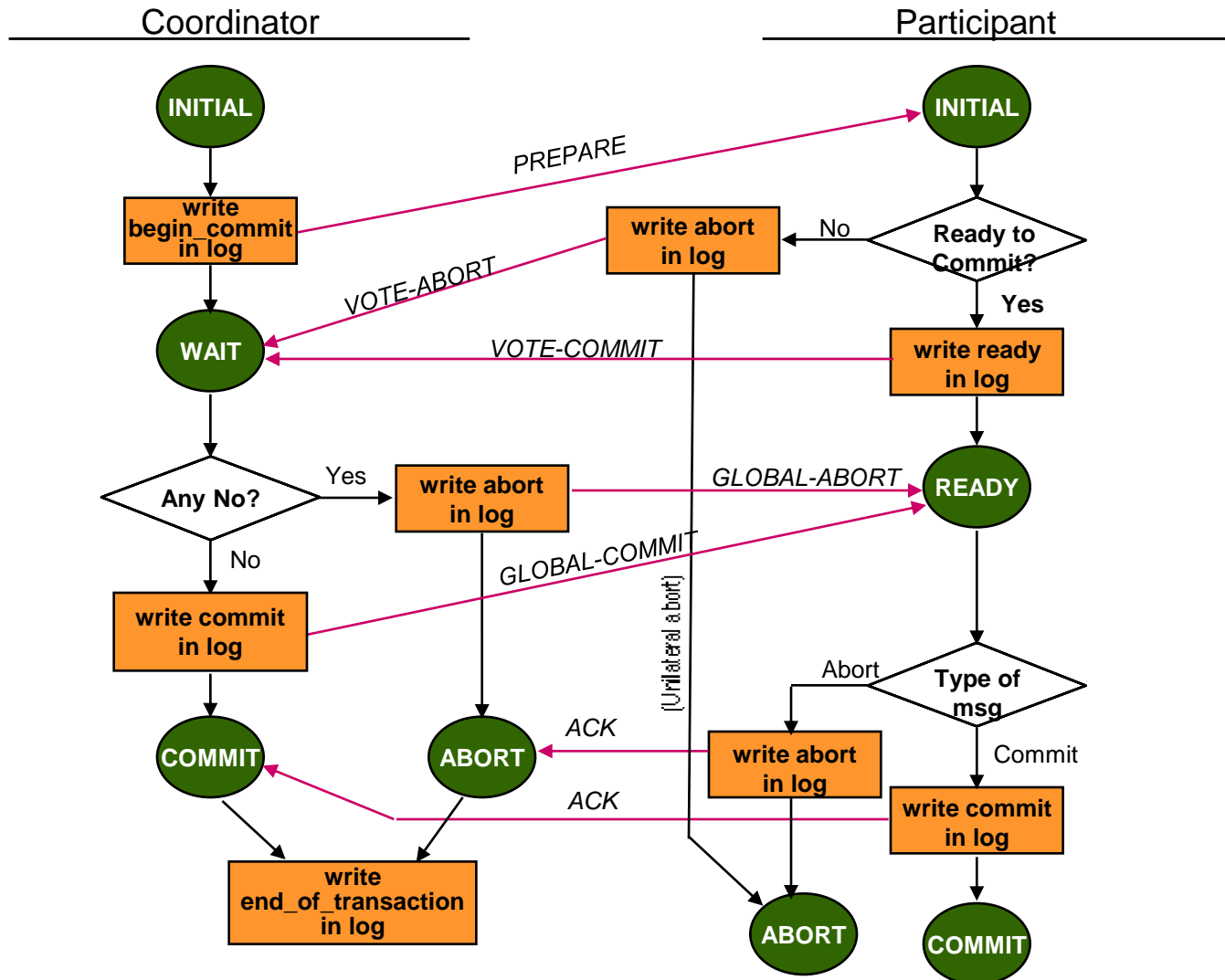
Review

- Fault-tolerance
 - Reliability and availability
 - Redundancy and process groups
 - Transactional dependability
 - Commit protocols

Fault Tolerance of Process Groups

- A system is *k fault tolerant* if it can survive faults in k components and still meets its specification.
- If failures are safe (silent), then $k+1$ processes are sufficient to get k fault tolerance.
- In case of arbitrary failures, $2k+1$ processes are required (since k failing processes can all generate the same result by chance)
 - This assumes that each process reaches its decision independently
 - What if processes gang up to produce wrong results? General problem is having a process group reach an *agreement*.
 - In this case you need $2k+1$ correctly functioning processes, for a total of $3k+1$ processes.
- In both cases we assume that communication failures do not occur.

2PC Protocol Actions



Problem With 2PC

- Blocking

- Ready implies that the participant waits for the coordinator
- If coordinator fails, site is blocked until recovery
- Blocking reduces availability

- Independent recovery is not possible

- However, it is known that:

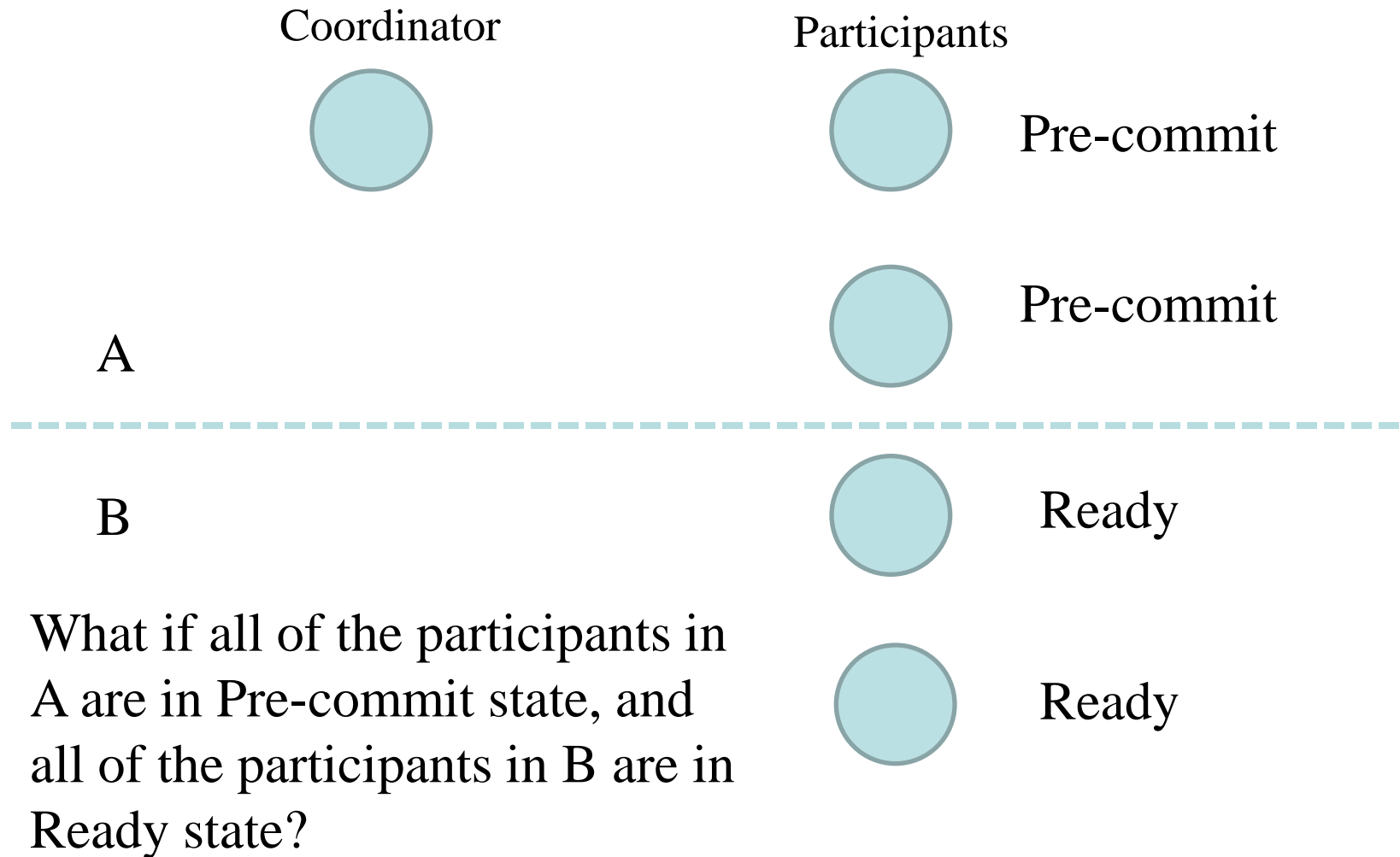
- Independent recovery protocols exist only for single site failures; no independent recovery protocol exists which is resilient to multiple-site failures.

- So we search for these protocols – 3PC

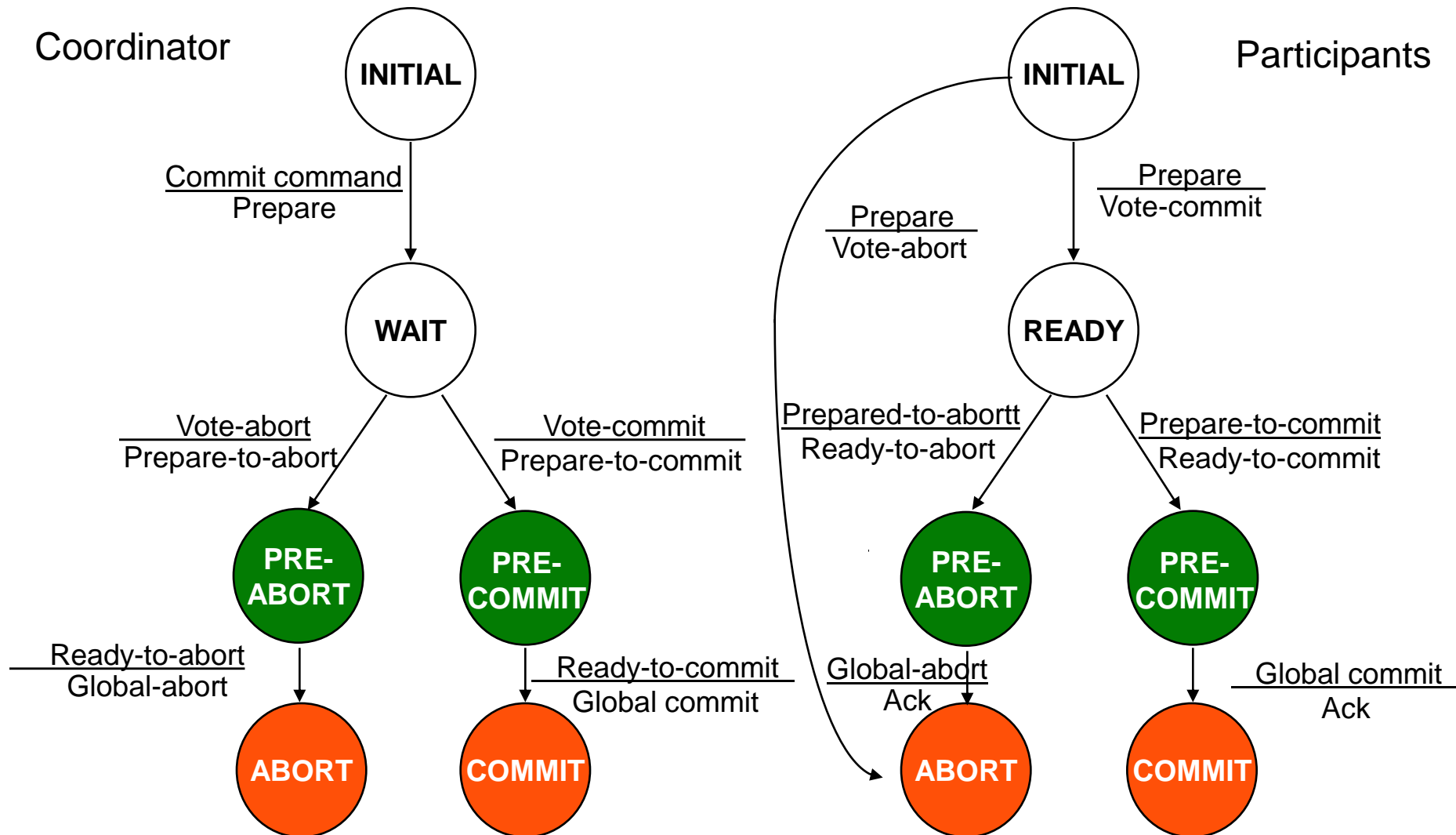
Three-Phase Commit

- 3PC is non-blocking.
- A commit protocols is non-blocking iff
 - it is synchronous within one state transition, and
 - its state transition diagram contains
 - no state which is “adjacent” to both a commit and an abort state, and
 - no non-committable state which is “adjacent” to a commit state
- Adjacent: possible to go from one state to another with a single state transition
- Committable: all sites have voted to commit a transaction
 - e.g.: COMMIT state

Transactions and Network Partitions



State Transitions in Quorum Protocols



Review

■ Security

- Fundamental security problems
- Cryptography
- Secure channels
 - Mutual authentication
 - Message confidentiality
 - Message integrity
- Access control
- Security Management
 - Key management
 - Authorization management

Types of Threats

■ Interception

- An unauthorized party has gained access to a service or data

■ Interruption

- Services or data become unavailable, unusable, destroyed and so on.

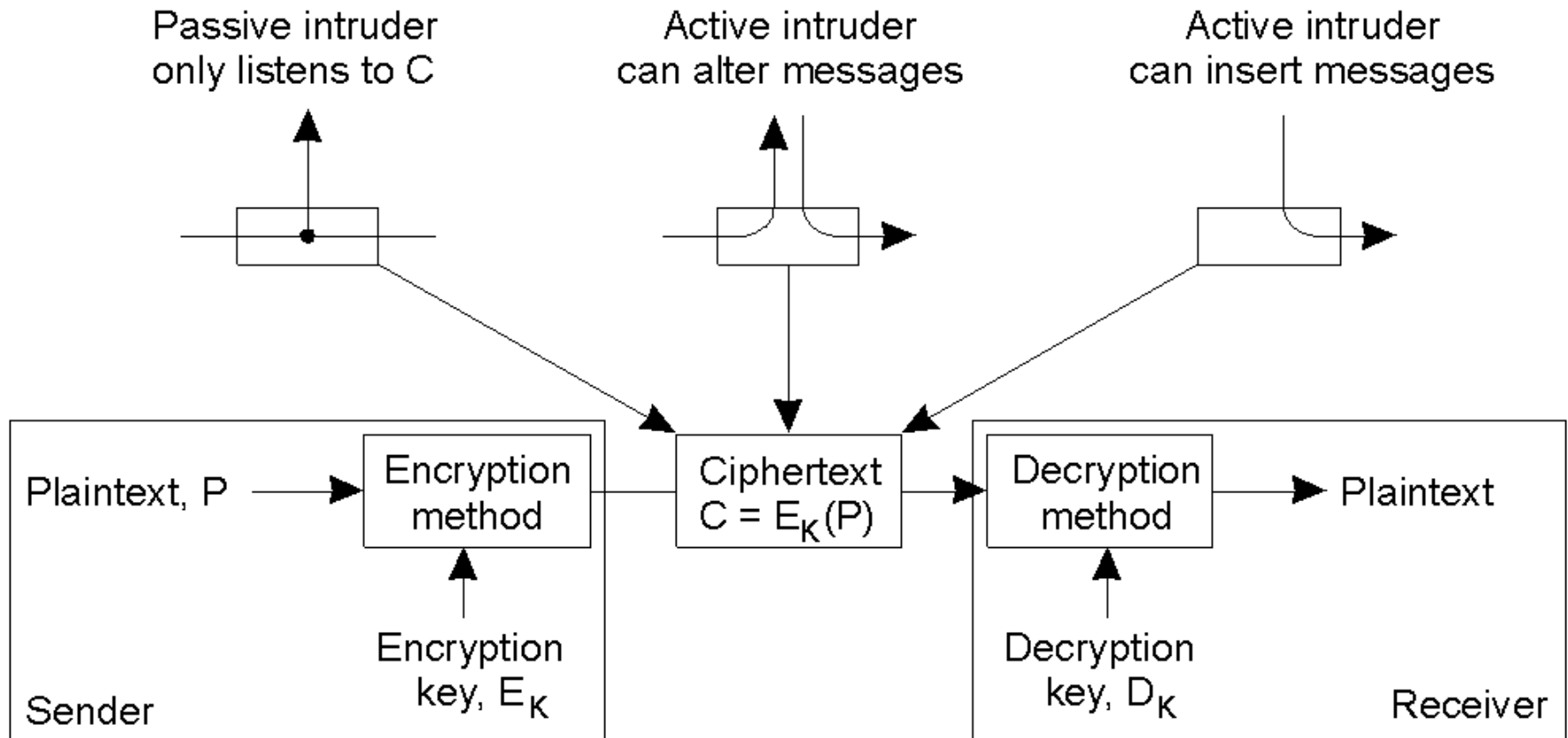
■ Modification

- Unauthorized changing of data or tampering with a service so that it no longer adheres to its original specifications

■ Fabrication

- Refers to the situation in which additional data or activity are generated that would normally not exist

Cryptography



The three different attacks that we need to protect against, and for which encryption helps.

Methods of Attack

- **Eavesdropping**
 - Obtaining copies of messages without authority
- **Masquerading**
 - Sending or receiving messages using the identity of another principal without their authority
- **Message tampering**
 - Intercepting messages and altering their contents before passing them on to the intended recipient
 - Man-in-the-middle attack
- **Replaying**
 - Storing intercepted messages and sending them out at a later time
- **Denial of service**
 - Flooding a communication channel or a system resource with messages in order to deny access for others