18CSE356T
Distributed Operating Systems

Unit III

CLR-3 : TO REALIZE THE NECESSITY OF SYNCHRONIZATION, CONSISTENCY AND FAULT TOLERANCE IN A DISTRIBUTED SYSTEM.

CLO-3 :

IMPLEMENT SYNCHRONIZATION OF DISTRIBUTED SYSTEMS USING VARIOUS ALGORITHMS.

TOPICS COVERED

- *Clock synchronization*
- *Event ordering*
- *Mutual exclusion*
- *Deadlock*
- *Election algorithm*

Why *SYNCHRONIZATION needed?*

- A distributed system - A collection of distinct processes that are spatially separated and run **concurrently.**
- In multiple concurrent processes, Sharing of resources may be ***cooperative or competitive***.
- Both cooperative and competitive sharing require adherence to certain **rules -** guarantee that correct interaction occurs.
- Rules - Implemented in the form of synchronization mechanisms .

SYNCHRONIZATION MECHANISMS

- Synchronization mechanisms that are suitable for distributed systems.
  - Clock synchronization
  - Event ordering
  - Mutual exclusion
  - Deadlock
  - Election algorithms

Clock synchronization

- An application - Have processes running concurrently on multiple nodes of the system, thus requires that the clocks of the nodes to be synchronized with each other.
  - For eg, a **distributed on-line reservation system**
  - The seat booked almost simultaneously from two different nodes be offered to the client who booked first, even if the time difference between the two bookings is very small.
  - NOT possible if clocks of the nodes of the system are not synchronized.

Clock synchronization (2)

It is the job of a distributed operating system designer to devise and use suitable algorithms for properly synchronizing the clocks of a distributed system.

How Computer Clocks Are Implemented

- A computer clock has three components
- -a quartz crystal that oscillates at a well-defined frequency,
- a *counter* register, and
- a *constant* register.
  - Constant register - Used to store a constant value that is decided based on the frequency of oscillation of the quartz crystal.
  - Counter register - Used to keep track of the oscillations of the quartz crystal.
  - *Clock tick* – An interrupt, reinitialized to the value in the

Clock synchronization (3)

Drifting of Clocks:

- Due to differences in the crystals, the rates at which two clocks run are normally different from each other.
- Difference may be small but accumulated over many oscillations leads to an observable difference in the times of the two clocks.
- Drift rate is approximately 10^-6, giving a difference of 1 second every 1,000,000 seconds, or 11.6 days.
- Computer clock must be periodically resynchronized with the real-time clock to keep it non-faulty.
- A clock is considered non-faulty if there is a bound on the amount of drift from real time for any given finite time interval.

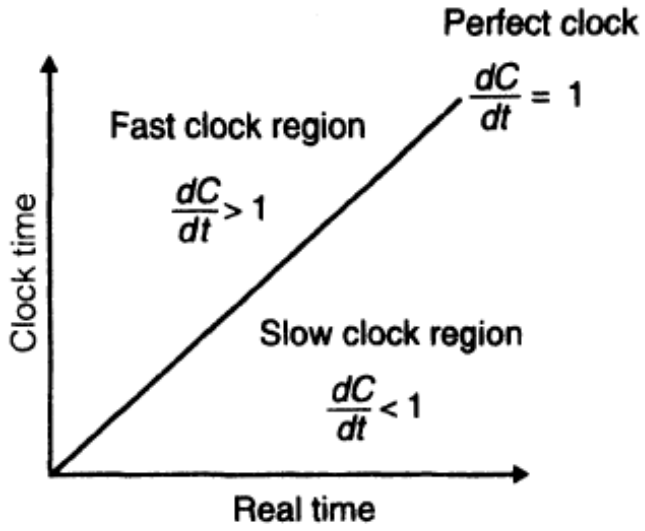
Drifting of Clocks

- Suppose that when the real time is *t,* the time value of a clock *p* is $C_p(t)$.
- If all clocks in the world were perfectly synchronized, we would have $C_p(t) = t$ for all *p* and all *t.*
- If the *maximum drift rate* allowable is p, a clock is said to be non-faulty if the following condition holds for it:

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

Drifting of Clocks

- From fig below, After synchronization with a perfect clock, slow and fast clocks drift in opposite directions from the perfect clock.
- Of two clocks, at a tim Δt after they were synchronized, the maximum deviation between the time value of the two $2\rho\Delta t$ cks will be



A distributed system requires the following types of clock synchronization:

1. Synchronization of the computer clocks with real-time (or external) clocks.

2. Mutual (or internal) synchronization of the clocks of different nodes of the system.

Issues

Clock Synchronization Issues:

- The difference in time values of two clocks is called *clock skew.*
- Errors occur mainly because of unpredictable communication delays during message passing used to deliver a clock signal or a clock message from one node to another.
- Time must never run backward because this could cause serious problems.

Clock synchronization algorithms

- Designed to gradually introduce such a change in the fast running clock instead of readjusting it to the correct time all at once.
  - **Distributed Algorithms**
  - **Centralized Algorithms**
    - *Passive Time Server Centralized Algorithm.*
    - *Active Time Server Centralized Algorithm*

Centralized Algorithms

- **Centralized Algorithms**
  - *Passive Time Server Centralized Algorithm.*
  - *Active Time Server Centralized Algorithm*

Passive *Time Server Centralized Algorithm*

- In the passive time server approach, the time server only responds to requests for time from other nodes.
- Each node periodically sends a message *("time* = ?") to the time server.
- When the time server receives the message, it quickly responds with a message *("time = T"),* where *T* is the current time in the clock of the time server node.
- Let us assume that when the client node sends the *"time*=?" message, its clock time is *T0,* and when it receives the *"time = T"* message, its clock time is *T1.*
- *Thus* ,time required for the propagation of the message *"time = T"* from the time server node to the client's node is *(T 1 – T0)/2.*
- When the reply is received at the client's node, its clock is readjusted to' *T+(T1 – T0)/2.*

Passive *Time Server Centralized Algorithm*

- *(T1 – T0)/2* is not a very good estimate, thus Several proposals have been made to improve this estimated value.

Two methods:

Method 1: Assumes the availability of some additional information.

- **ASSUMPTION:** the approximate time taken by the time server to handle the interrupt and process a *"time* =?" request message is known.
- Let this time be equal to *I*. Then a better estimate of the time taken for propagation of the message *"time = T*" from the time server node to the client's node would be *(T1 – T0-I)/2.*
- When the reply is received at the client's node, its clock is readjusted to

Passive *Time Server Centralized* Algorithm

- Method 2: Assumes that no additional information is available.
  - Proposed by Cristian [1989].
  - Several measurements of *T1* – *T0* are made.
  - Measurements exceeding some threshold value - Discarded.
  - The average of the remaining measurements is then calculated, and half of the calculated value is used as the value to be added to T.
  - Alternatively, minimum T1- *T0* taken as most accurate one, and half of this value is used as the value to be added to *T.*

Active Time Server Centralized Algorithm

- Time server periodically broadcasts its clock time *("time = T").*
- Receive the broadcast message and use the clock time in the message for correcting their own clocks.
- Each node has a priori knowledge of approximate time *(Ta)* required for the propagation of the message *"time = T"* from the time sever node to its own node.
- When broadcast message is received at a node, the node's clock is readjusted to the time *T+Ta.*
- ***DRAWBACK:***
  - Not fault tolerant. If broadcast message reaches too late - the clock of that node will be readjusted to an incorrect value.
  - Requires broadcast facility to be supported by the network.

Active Time Server Centralized Algorithm

Berkeley algorithm:

- Proposed by Gusella and Zatti [1989] for internal synchronization of clocks of a group of computers running the Berkeley UNIX.
- Time server periodically sends a message *("time* = ?") to all the computers in the group.
- Time server has a priori knowledge of the approximate time required for the propagation of a message from each node to its own node.

Active Time Server Centralized Algorithm

- Based on this knowledge, it first readjusts the clock values of the reply messages.
- Takes *fault-tolerant average* of the clock values of all the computers (including its own) - the time server chooses a subset of all clock values that do not differ from one another by more than a specified amount, and the average is taken only for the clock values in this subset.

Centralized algorithms - Drawbacks

1.They are subject to single-point failure. If the time server node fails, the clock synchronization operation cannot be performed. This makes the system unreliable.

2. From a scalability point of view it is generally not acceptable to get all the time requests serviced by a single time server. In a large system, such a solution puts a heavy burden on that one process.

Distributed Algorithms

- Externally synchronized clocks are also internally synchronized
- Each node's clock is independently synchronized with real time.
- All clocks of system remain mutually synchronized.
- Each node is equipped with real time receiver so that each node can be independently synchronized.

Distributed Algorithms

- Theoretically internal synchronization of clock is not required.
- in practice, due to inherent inaccuracy of real time clocks, different

real time clocks produce different time.

- Internal synchronization is performed for better accuracy.
- Types of internal Synchronization:
  - Global averaging
  - Localized Averaging

Global Averaging

- Each node broadcasts its local clock time in the form of a special "resync" message.
- After broadcasting the clock value , the clock process of a node waits for time T (T is calculated by algorithm).
- During this period, clock process collects the resync messages broadcast by other nodes.
- Clock process estimates the skew of its clock with respect to other nodes (when message received) .
- It then computes a fault-tolerant average of the estimated skews and use it to correct the local clock.
- Two commonly algorithms use to calculate the fault-tolerant average.

Global Averaging cont..

- 1st simplest algorithm is to take the average of estimated skews and use it as the correction for the local clock. And only below a threshold are taken into account, and other are set to zero before computing the average.

- 2nd Algorithm, each node limits the impact of faulty clocks by discarding the highest and lowest skews. And then calculating the average of the remaining skews. And use it to the correction for the local clock.

Localized Averaging Algorithm

- Global algorithm was only Suitable for small networks due to broadcast facility.
- Where node has direct communication to every other node (fully connected topology) .
- This algorithm attempt to overcome the drawbacks of the global averaging algorithm.
- Nodes of a distributed system are logically arranged in some kind of pattern such as Ring.
- Periodically each node exchange its clock time with is neighbors in the ring.
- And then sets its clock time by taking Average.
- Average is calculated by , its own clock time and clock times of its neighbors.

Event Ordering

- Keeping the clock synchronized within 5,10 milisec is expensive task.
- Lamport observed, Its not necessary to keep the clocks synchronized

in distributed system.

- It is sufficient to keep the all event be totally ordered.
- For Partial ordering of the events, Lamport defined a new relation called Happened-Before relation.

Happened Before Relation

- Happened Before Relation(Denoted by $\rightarrow$) on a set of events Satisfies Following Conditions.
- If a and b are two events in the same process and **a occurs before b**, then $a \rightarrow b$.
- If a is event of sending a message by one process and b is event of receipt of same message by another process, then $a \rightarrow b$. it follows **law of causality** because receiver can not receive message until sender sends it and propagation time from sender to receiver is always positive.
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. happened before relation follows **transitive relation**
- $a \rightarrow a$ is not possible. Happened before relation is **irrreflexive partial** ordering.

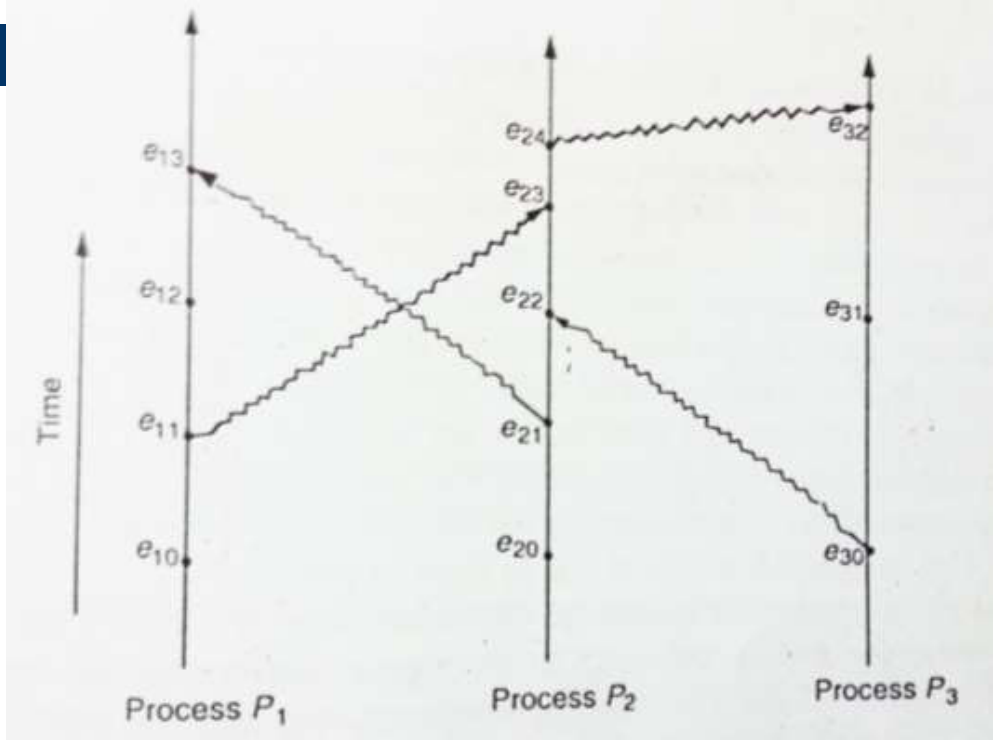
Happened Before relation cont..

- Event types;
  - Causal events
  - Concurrent Events:

- Causal Events:
- Two events a $\rightarrow$b is true if path exists between a and b by moving forward in time along process and message lines in the direction of arrows. These events are related by happened before relation.

- Concurrent events: two events are concurrent if no path exists between them and are not related by happened before relation.

Diagram



- Causal Events:
- e1o → e11 e20 →e24 e11 → e23 e21 →e13
- e30 → e24
- e11 → e32
- Concurrent Events:
- e12 and e20 e21 and e30 e10 and e30 e11 and e31 e12 and e32 e13 and e22

Logical Clocks Concept

- For determining the event a occurs before b, need common clock, or set of perfectly synchronized clocks.
- Neither of these available, Lamport provided a solution of this problem. Local clock concept.
- The logical clock concept is to associate a timestamp with each system event.
- Each process P_i, has a clock C_i.
- And assigned a number $C_i(a)$ to any event in that process.
- In fact, Logical clocks implemented by counters with no actual timing mechanism.
- Happened-before relation should be ordered, using these clocks.
- So that Logical clocks must satisfy the following condition:
- For any two events a and b, if $a \rightarrow b$, then $C(a) < C(b)$

Implementation of Logical Clocks

Happened Before algorithm with logical clocks must follows the following conditions:

- C1: If a and b are two events within the same process P_i and a occurs before b , then $C_i(a) < C_j(b)$.
- C2: If a is the sending message by process P_i, and b is the receipt of that message by process P_j, then $C_i(a) < C_j(b)$
- C3: A clock C_i associated with a process P_i, must always go forward, never backward, That is, corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting the value.

Implementation of Logical Clocks cont.

To meet conditions C1, C2, and C3, Lamport used the following implementation rules:

IR1(implementation Rule): Each process P_i increments C_i between two successive events.

IR2(implementation Rule): If P_i sending a message m by an event a, and the message m contains a timestamp $T_m = C_i(a)$, and other process P_j receive this message $C_i(a)$. And if $C_i(a)$ is greater than and equal to its present value then $C_i(a)$ value updated on it with +1 increment, otherwise C_j present value continue.

Implementation of Logical Clocks by Using Counters

- Two processes P1 and P2 each have counters C1 and C2.
- C1 values increase C1=0,1,2,3,4,5…
- C2 values increase C2=0,1,2,3,4,5…
- Counters act like logical clocks.
- Message having the incremented value of counter, that is sent by P1.
- On the other side P2 receive this message, and instead of simple increment of its counter, a check is made .
- Check is , If the incremented counter value of P2 is less than or equal to the timestamp in the received message from P1 .
- if C2 <= C1 ,e.g 3 <= 4 , C2 value will be updated with +1.

Implementation of Logical Clocks by Using Counters cont...

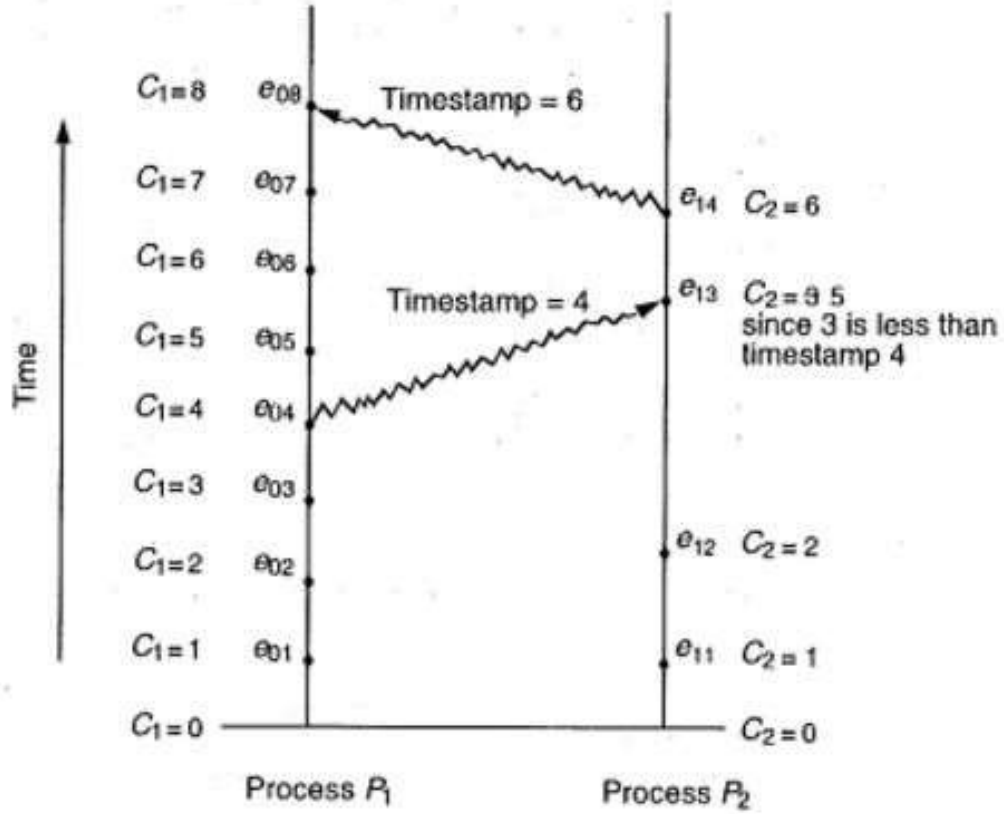
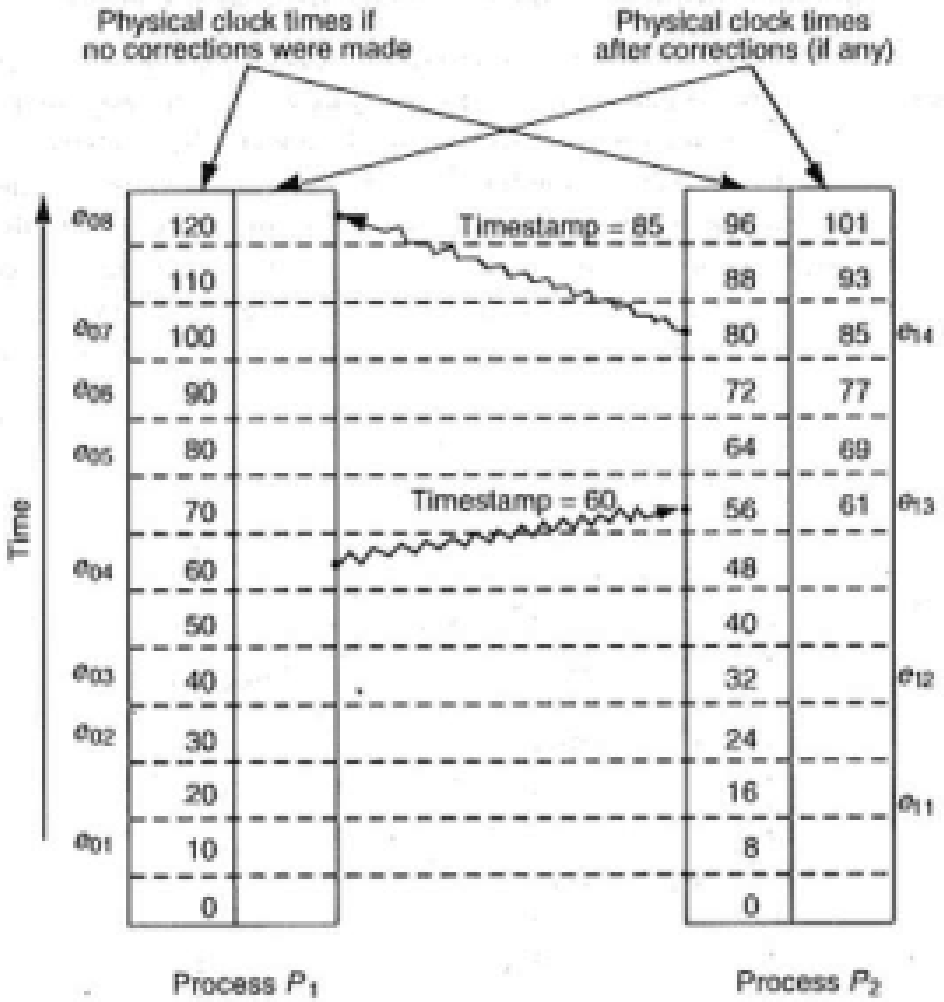


Fig. 6.4 Example illustrating the implementation of logical clocks by using counters.

Implementation of Logical Clocks by Using Physical Clocks

- Each process has a physical clock associated with it.
- Each clock run at a constant rate.
- However, The rates at which different clocks run are different.
- When clock of process P1 has ticked 10 times, The clock of process P2 has ticked only 8 times.
- If the incremented counter value of P2 is less than or equal to the timestamp in the received message.
- if C2 <= C1 ,e.g 56<= 60 , C2 value will be updated with +1.

Implementation of Logical Clocks by Using Physical Clocks cont..



Total Ordering of Events

- Happened-before has only partial ordering.
- Two events a or b not related by Happened-before , if same timestamps are associated with them.
- P1 and P2 clocks are same for example 100 .
- Nothing can be said about the order.
- Additional information required.
- Lamport purposed Total ordering of processes.
- He used process identity numbers.
- Timestamp associated with them will be 100.001 and 100.002.
- Here , process identity numbers of processes P1 and P2 are 001 and 002.

Mutual Exclusion

- A file must not be simultaneously updated by multiple processes.
- Such as printer and tape drives must be restricted to a single process at a time.
- Therefore , Exclusive access to such a shared resource by a process must b ensured.
- This exclusiveness of access is called mutual exclusion between processes.

Critical Section

- The sections of a program that need exclusive access to shared resources are referred to as critical sections.
- For mutual exclusion , Ways are introduced, to prevent processes from executing concurrently with their associated critical section.

Following requirement must satisfy for mutual exclusion:

1. Mutual exclusion: At anytime only one process should access the resource. A process can get another resource first releasing its own.
2. No starvation: if every process that is granted the resource eventually release it, every request must be eventually granted.

Starvation of process

- Indefinite postponement of a process because it requires some resource, but the resource is never allocated to this process. It is sometimes called livelock.

- 6.Mana.13.starvation (auckland.ac.nz)

Three approaches to implement mutual exclusion

- Centralized

  Approach

- Distributed

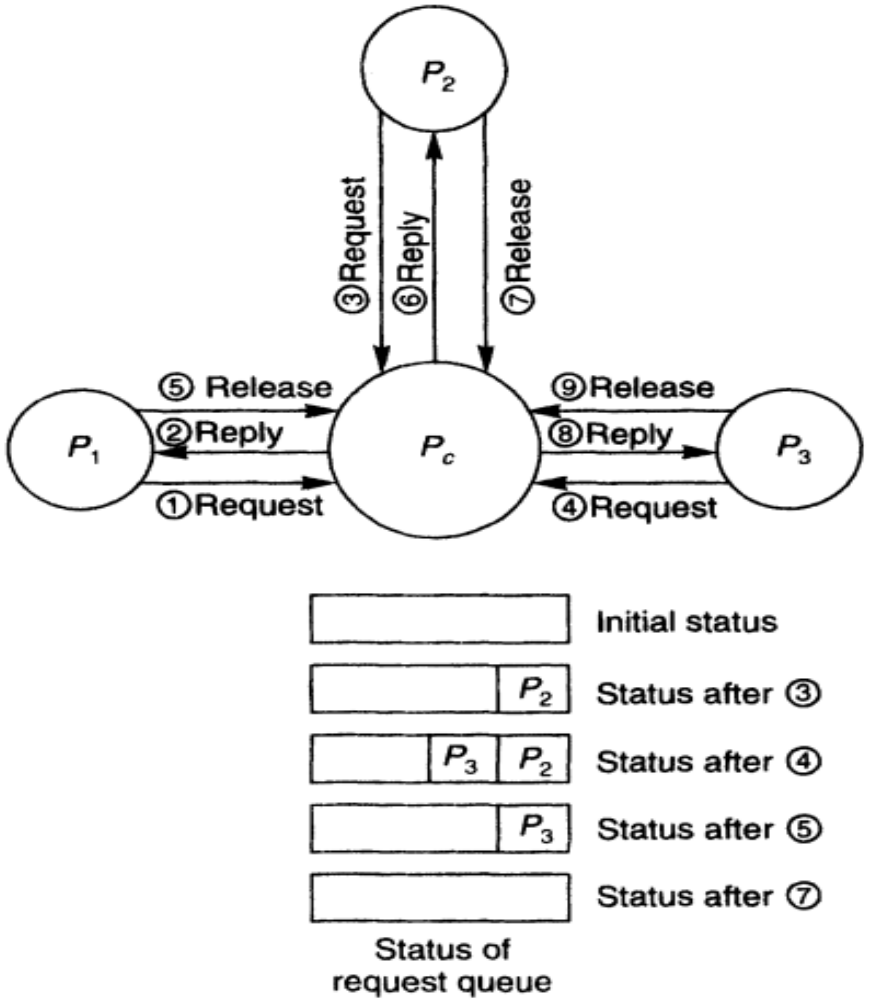
  Approach

- Token Ring

Centralized Approach

- One process in the system is selected as coordinator.
- That coordinates the entry to the critical sections.
- Each process that wants to enter a critical section must first seek permission from the coordinator.
- If two processes simultaneously ask for permission to enter the critical section, Coordinator grants permission according with some scheduling algorithms.
- When process exits, it notify the coordinator so that the coordinator can grant permission to another process.
- This algorithm (in centralized) ensures mutual exclusion, because at a time , the coordinator allows only one process to enter in critical section.
- This algorithm (in centralized) ensures no starvation, because use of FCFS (First come First served) .

Centralized Approach cont..



Centralized Approach cont..

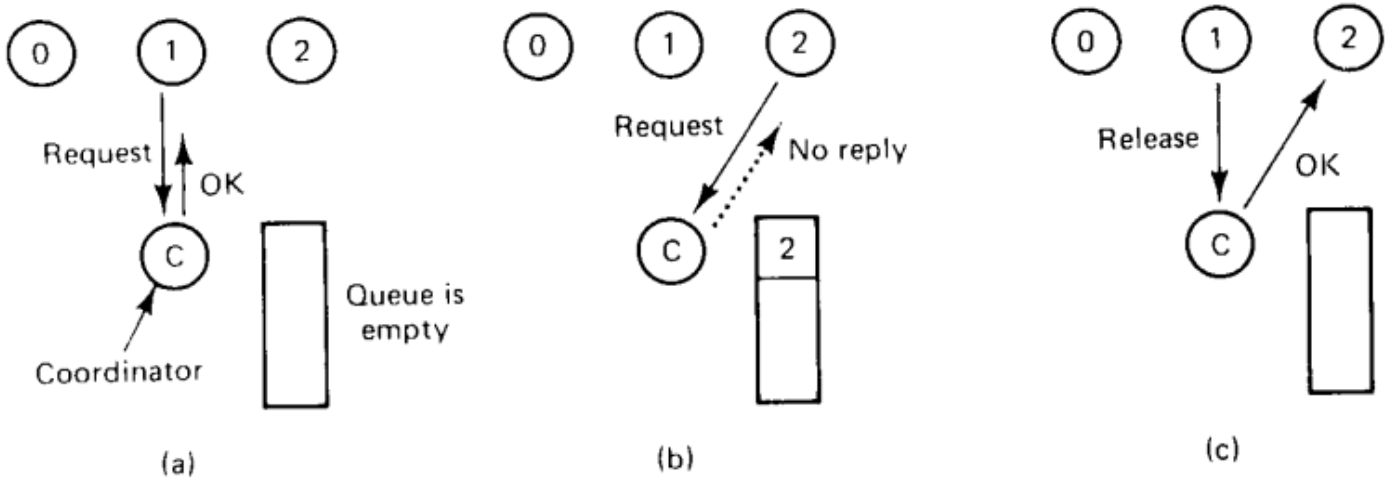


Fig. 3-8. (a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted. (b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply. (c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

Centralized Approach cont..

- Benefits:
  - Easy to implement
  - Requires only three messages per critical section (request, reply, release)

    .
- Drawbacks:
  - Single coordinator
  - Single point of failure
  - Performance down due to bottleneck in large systems.

Distributed Approach

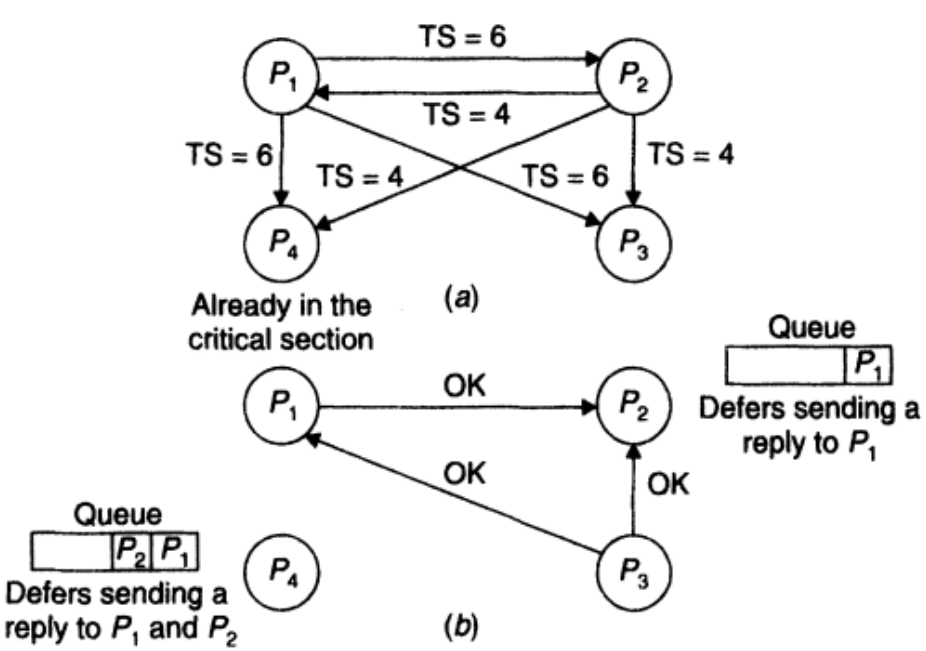
- Decision making for mutual exclusion is distributed across entire system.
- All process that want to enter critical section co-operate with each other before taking decision.
- Ricart and Agarwala's algorithm is used, based upon Lamport's event ordering scheme to assign unique time stamp to each event.
- When a process wants to enter critical section it sends request message to all process. Message contains:
  - Process identifier of process
  - Name of critical section that process wants to enter.
  - Unique time stamp generated by process for request message.
- On receiving request message a process immediately sends reply message or defers reply based upon following rules.

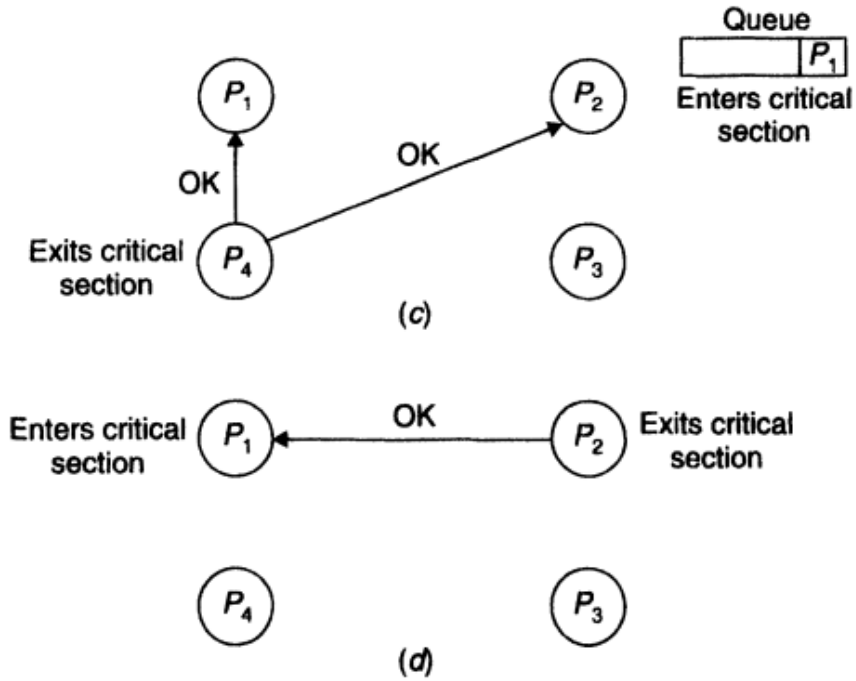
Distributed Approach Cont..

- If receiver process is in critical section, it places request message in queue and defers sending a reply.
- If receiver process is not in critical section and waiting to enter, it compares it's own time stamp with received time stamp. If received time stamp is lower than it' own, it means that process made request earlier, than it sends reply to that process otherwise places it in queue and defers sending reply.
- If receiver process is neither in critical section nor waiting, it simply sends reply message.
- A process enters critical section if it receives message from all process, executes in critical section and than sends reply message to all processes in queue.

Distributed Approach Cont.

.





Distributed Approach Cont.

.

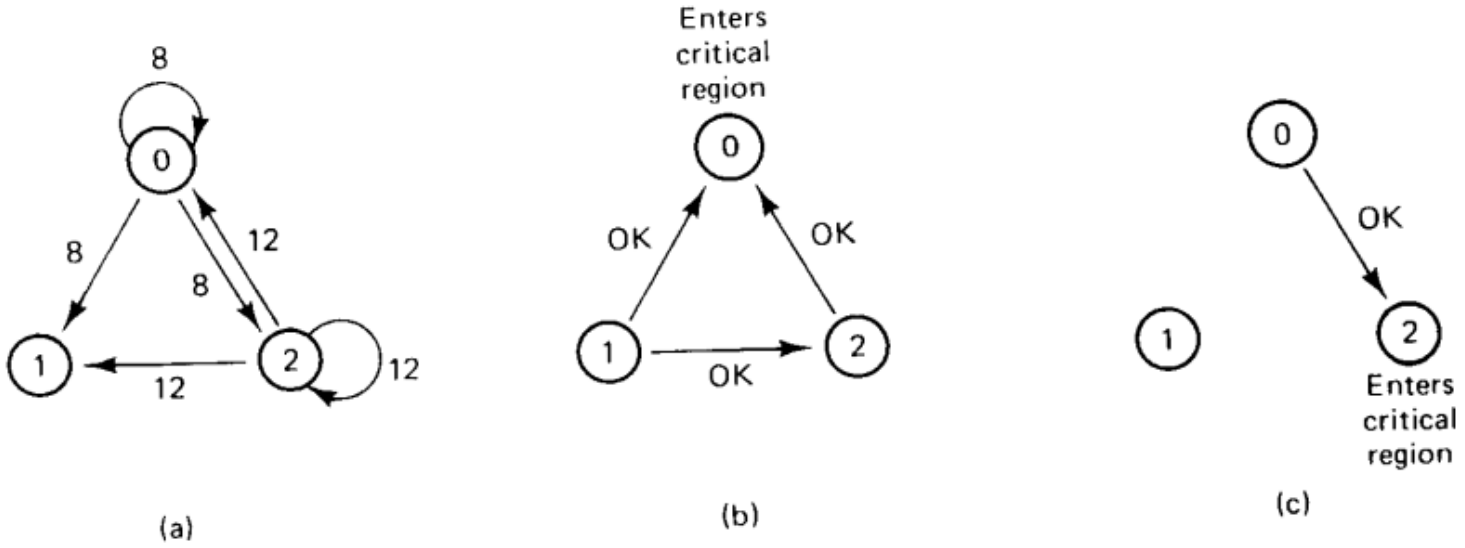


Fig. 3-9. (a) Two processes want to enter the same critical region at the same moment. (b) Process 0 has the lowest timestamp, so it wins. (c) When process 0 is done, it sends an *OK* also, so 2 can now enter the critical region.

Drawbacks and solutions

1)System having n processes is liable to n points of failure and blocks in case of single process failure.

▪Solution: receiver sends permission denied message when it is in critical section, else it sends OK. if receiver did not receive reply it keeps on trying after time-out, until it gets reply, else it considers that process has crashed.

2) Each process must know identity of process participating in group.

- When a process joins it must receive names of all other processes and name of new process must be distributes to all members of group.
- Suitable only for small group due to large no of updates of creation and deletion.

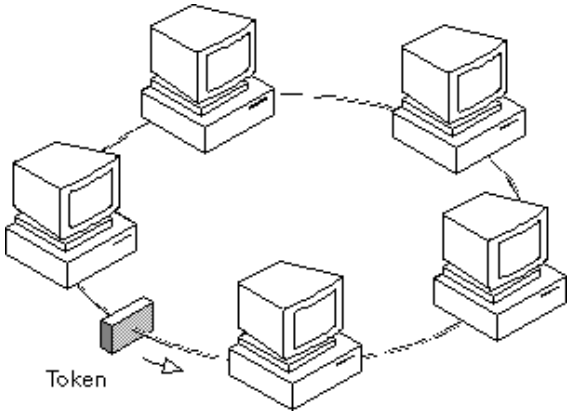
3) To much wait if there are large no of nodes. If there are n nodes and single message travels through network then each process will wait for 2(n-1) messages. Suitable for small group.

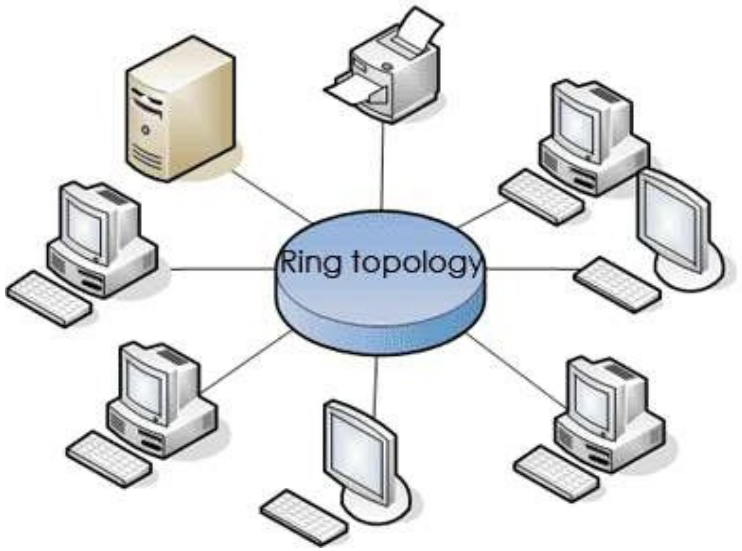
▪Best solution: Majority consensus rather than consensus of all processes is proposed in literature.

Token passing Approach

- Mutual Exclusion is achieved by circulating single token among processes.
- Token is special type of message that entitles its holder to enter critical section. Process are arranged in logical ring structure.
- Token can be moved clockwise or anti-clockwise.
- When a process gets token it can enters critical section and after finishing work, exits from critical section and passes token to neighbor process.
- Process can enter critical section one time in each turn with token.
- If process is not interested to enter in critical section, it simply passes token to neighbor, if no one is interested token keeps on circulating in ring.
- Waiting time varies from 0 to n-1 messages depending upon receipt of

Diagram





Diagram

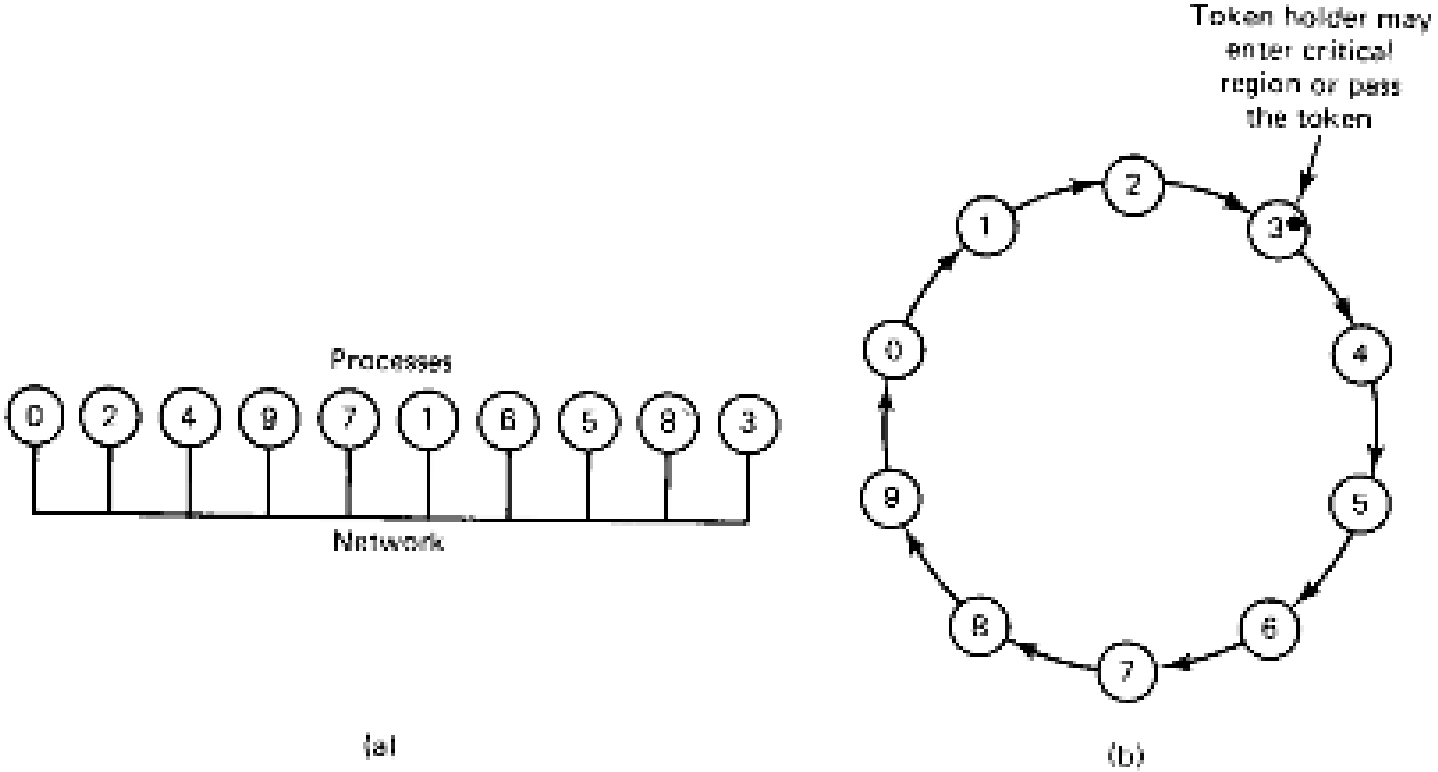


Fig. 3-10. (a) An unordered group of processes on a network. (b) A logical ring constructed in software.

Drawbacks and Solutions:

1) **Process failure**: Single process failure causes logical ring to break.

- Solution: exchange of Ack(acknowledgement) after message Receipt with neighbor. If no Ack comes, means problem with process.
- Current ring information with each process, using this each process skips its faulty neighbor process and when that process recovers, it informs it's previous neighbor or process.

2**) Lost Token**: Token is lost, new token message must be generated.

- Solution: Designate one process as monitor process.
- Monitor periodically circulates "who has token" message. All the process passes message to neighbor, except the process which has token. It send its process identifier in special field with token to neighbor.
- When message returns to monitor after one circulation, if there is no entry in that field means token has been lost, generates new token and circulates in ring

3) **Monitor process and "who has token" message is lost.**

- Solution: more than one monitor process. And if one fails election has to be done for selection of next monitor who will generate token.

Example videos for each algorithm

- Centralized Approach
  - https://www.youtube.com/watch?v=Ze6JI0pvjXM
- Distributed Approach
  - https://www.youtube.com/watch?v=4OYEgJb5vfI
- Token Ring
  - https://www.youtube.com/watch?v=KSBVEAKfKus

Comparison of the three 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

Fig. 3-11. A comparison of three mutual exclusion algorithms.

Election Algorithms

- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instant of time there is a single coordinator for all processes in the system.
- Election algorithms are based on following assumptions
    1. Each process in system has unique priority no.
    2. Whenever an election is held, process having highest priority no among the currently running processes is elected as coordinator.
    3. On recovery, a failed process can take appropriate actions to rejoin the set of active process.
- **Types:**
    - Bully Algorithm
    - Ring Algorithm

Bully Algorithm

- This algorithm was proposed by Garcia-Molina [1982].
- In this algorithm it is assumed that every process knows the priority number of every other process in the system.

Bully Algorithm

- When a process(say Pi) sends a request message to coordinator and does not receive a reply within a fixed time out period, it assumes that co-ordinator has failed.
- It initiates election by sending an election message to every process with higher priority number than itself.
- If Pi does not receive response in time out means it has highest priority.
- It will become coordinator and will inform all processes through message.
- If Pi receives reply means other processes with higher priority are available and does not take any decision and waits for message of coordinator.
- When Process Pj receives an election message. It informs sender that is alive and now Pj will do election and will become coordinator if it has highest priority number.
- Else Pj will initiate message and this will continue.

Bully Algorithm Example

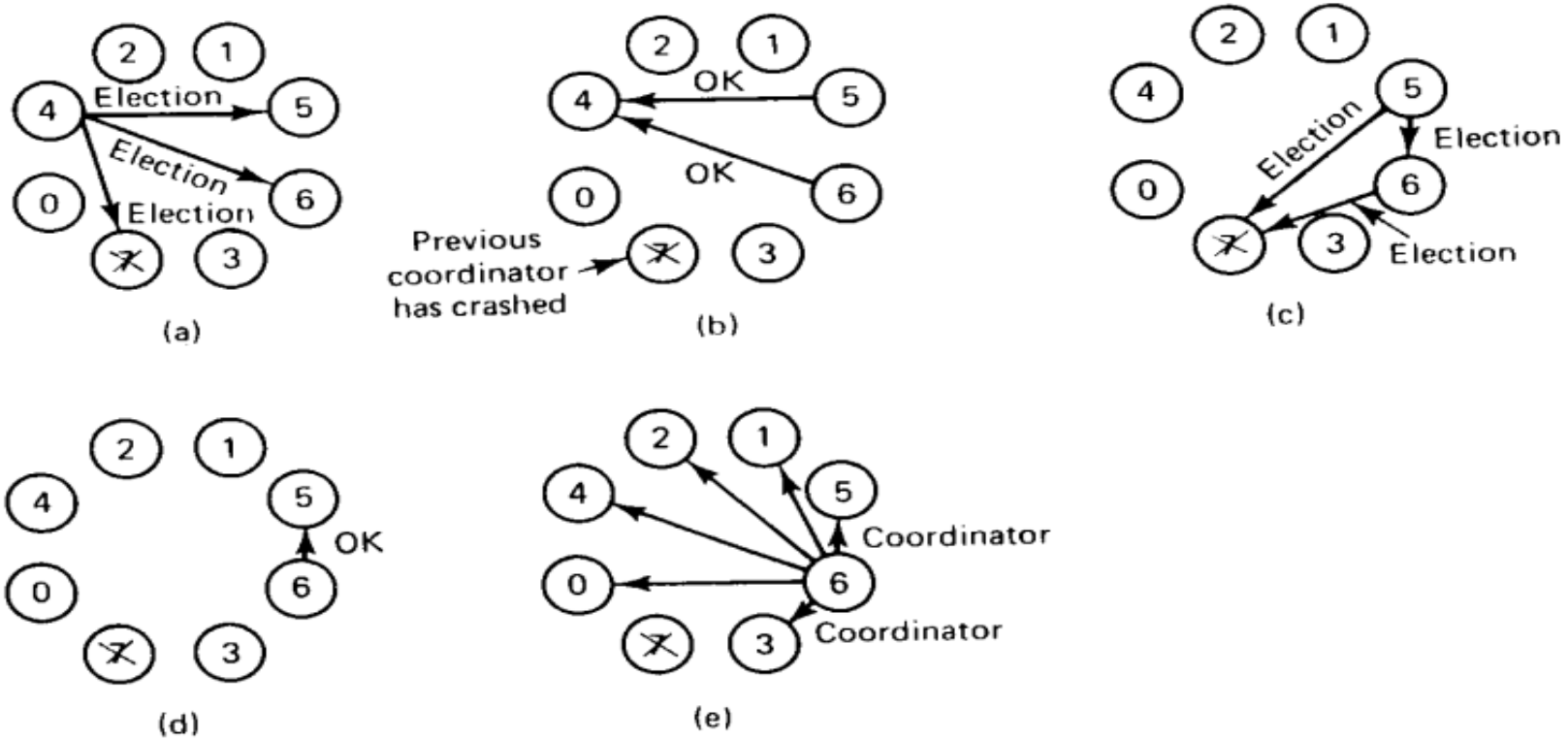


Fig. 3-12. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

Bully Algorithm Example

https://www.youtube.com/watch?v=uzYIQ02NDiY

Ring Algorithm

- Processes are arranged in logical ring.
- Every process in the system knows structure of ring.
- If successor of sender process is down, it can be skipped until active member is achieved.

Ring Algorithm

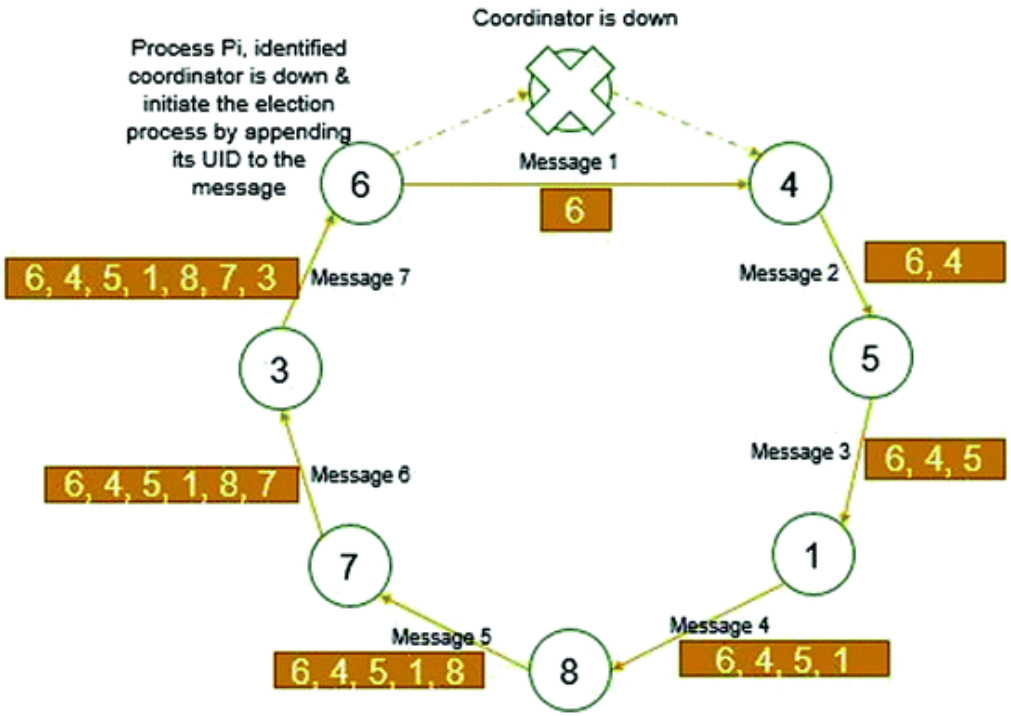
The algorithm works as follows.

- When Pi requests coordinator and it is down it will wait for time out, it initiates election by sending election message to successor.
- Message contains priority no of process Pi.
- Each successor will add its priority no on receipt.
- It will circulate and reaches Pi.

Ring Algorithm

- Pi recognizes its own message due to first priority no.
- It selects the highest priority no processor.
- It then circulates coordinator message to all active processes to tell them.
- It deletes coordinator message after complete round when message comes.
- When pj comes after recovery it tells its neighbour its process identity and message will circulate until it reaches coordinator and will receive reply to let it know the coordinator.

Ring Algorithm Example



Ring Algorithm Example

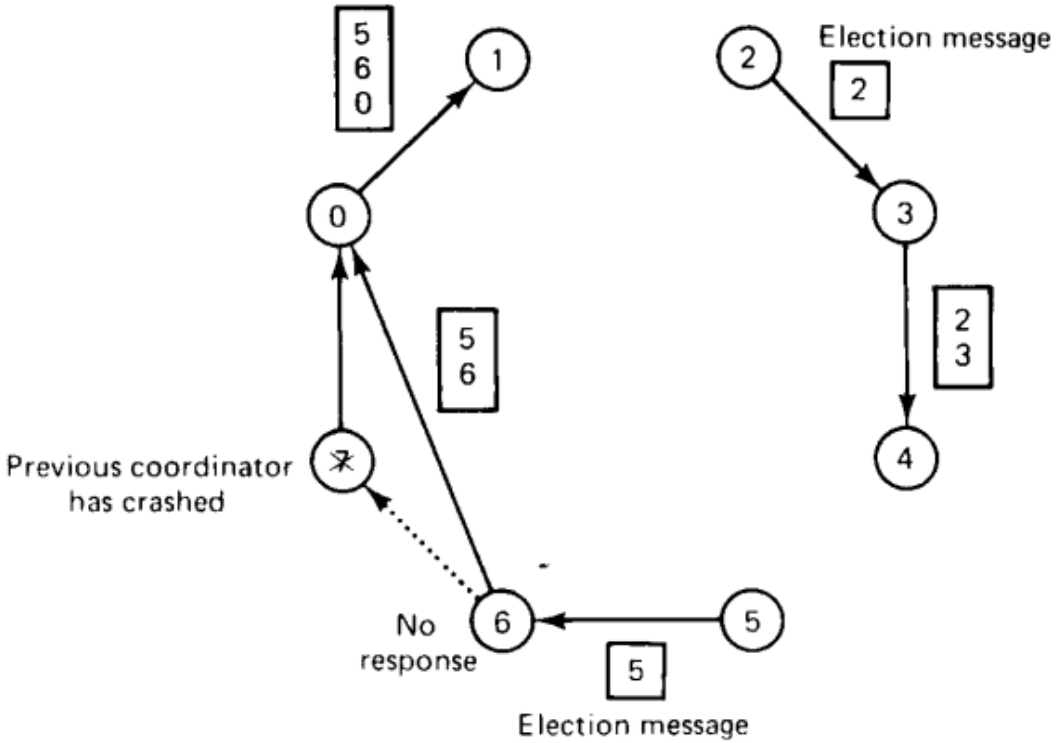


Fig. 3-13. Election algorithm using a ring.

Ring Algorithm Example

https://www.youtube.com/watch?v=VBCD2h9VLDk

Discussion of Two Election Algorithms

Bully Algorithm:

•When lowest priority no process detects coordinator failure in a system of n processes n-2 elections are performed, leaving failed coordinator and sender.

•In worst case complexity is O(n^2).

•best case n-2 messages are required. Complexity O(n)

Ring Algorithm:

•election requires 2(n-1) messages.

•n-1 messages for one round rotation of election message. n-1 messages for one round rotation of coordinator.

Atomic transaction

- An atomic transaction is **an indivisible and irreducible series of database operations such that either all occurs, or nothing occurs**. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright.

Now look at a modern banking application that updates an online data base in place. The customer calls up the bank using a PC with a modem with the intention of withdrawing money from one account and depositing it in another. The operation is performed in two steps:

1. Withdraw(amount, account1).
2. Deposit(amount, account2).

If the telephone connection is broken after the first one but before the second one, the first account will have been debited but the second one will not have been credited. The money vanishes into thin air.

Transaction Model

Various types of storage media are distinguished by their relative speed, capacity, and resilience to failure.

- Volatile storage.
- Nonvolatile storage.
- Stable storage.

Transaction Processing Systems (TPS)

- Operations on a database are usually carried out in the form of transactions.
- Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtin

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Transaction Primitives

Transaction Primitives

Programming using transactions requires special primitives that must either be supplied by the operating system or by the language runtime system. Examples are:

1. BEGIN_TRANSACTION: Mark the start of a transaction.
2. END_TRANSACTION: Terminate the transaction and try to commit.
3. ABORT_TRANSACTION: Kill the transaction; restore the old values.
4. READ: Read data from a file (or other object).
5. WRITE: Write data to a file (or other object).

NESTED TRANSACTION

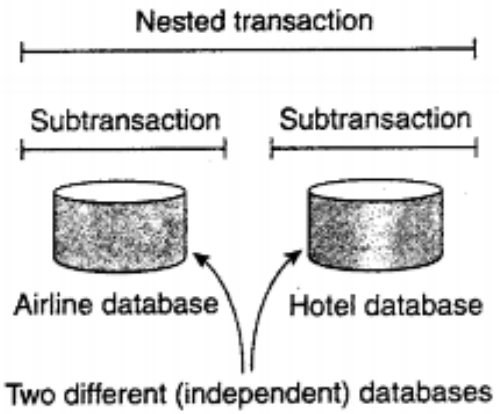
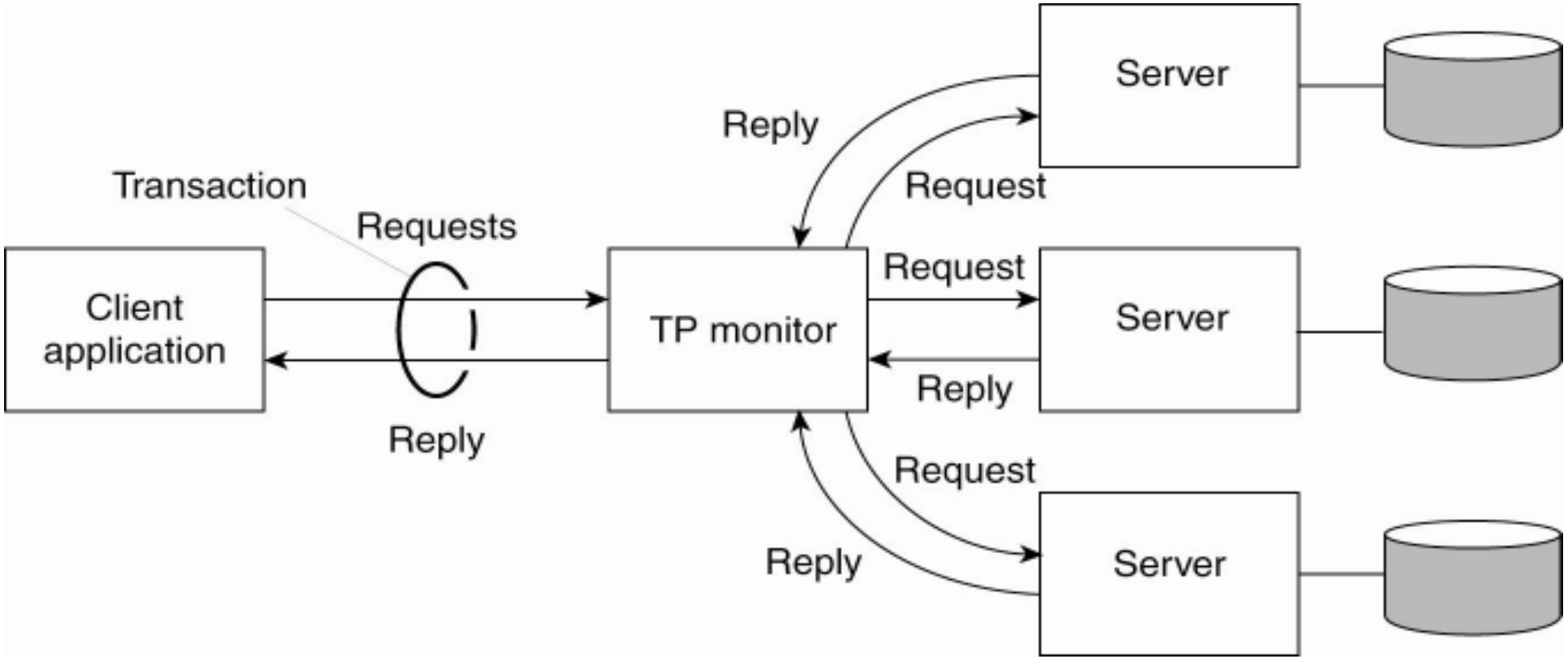


Figure 1-9. A nested transaction.

- The top-level transaction may fork off children that run in
  parallel with one another, on different machines, to gain performance or simplify programming.
- Each of these children may also execute one or more sub transactions, or fork off its own children.
- Provide a natural way of distributing a transaction across multiple machines.
- Follow a logical division of the work of the original transaction.

For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three sub transactions. Each of these sub transactions can be managed separately and independent of the other two.

Transaction Processing Systems (TPS)



The role of a TP monitor in distributed systems

ACID Properties

- The characteristic feature of a transaction is either all of these operations are executed or none are executed.
- This all-or-nothing property of transactions is one of the four characteristic properties that transactions have.
  - 1. Atomic: To the outside world, the transaction happens indivisibly.
  - 2. Consistent: The transaction does not violate system invariants.
  - 3. Isolated: Concurrent transactions do not interfere with each other.
  - 4. Durable: Once a transaction commits, the changes are permanent.

Transaction Properties

Properties of Transactions

Transactions have four essential properties. Transactions are:

1. Atomic: To the outside world, the transaction happens indivisibly.
2. Consistent: The transaction does not violate system invariants.
3. Isolated: Concurrent transactions do not interfere with each other.
4. Durable: Once a transaction commits, the changes are permanent.

These properties are often referred to by their initial letters, **ACID**.

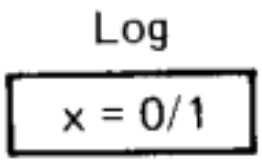
Write-ahead log

- Write-ahead logging , called as intention list
- The changes are first recorded in the log, which must be written to stable storage, before the changes are written to the database.
- In a system using WAL, all modifications are written to a log before they are applied. Usually both redo and undo information is stored in the log.

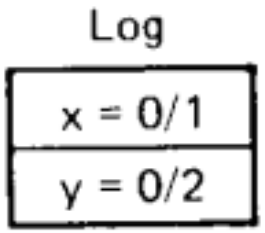
Write-ahead log

```
x = 0;
y = 0;
BEGIN_TRANSACTION
    x = x + 1;
    y = y + 2;
    x = y * y;
END_TRANSACTION
```

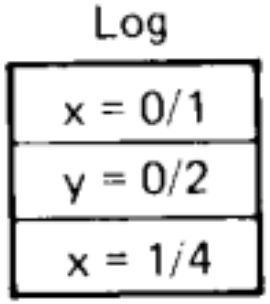
(a)



(b)



(c)



(d)

Fig. 3-19. (a) A transaction. (b)–(d) The log before each statement is executed.

Commit Protocol

Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The *two-phase commit* (2PC) protocol is widely used

Commit Protocol

Two Phase Commit Protocol (2PC)

- Assumes fail-stop model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the local sites at which the transaction executed
- Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Commit Protocol

Phase 1: Obtaining a Decision

- Coordinator asks all participants to *prepare* to commit transaction T_i.
  - C_i adds the records <prepare *T*> to the log and forces log to stable storage
  - sends prepare *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <no *T*> to the log and send abort *T* message to C_i
  - if the transaction can be committed, then:
  - add the record <ready *T*> to the log
  - force *all records* for *T* to stable storage
  - send ready *T* message to C_i

Commit Protocol

Phase 2: Recording the Decision

- *T* can be committed of C_i received a ready *T* message from all the participating sites: otherwise *T* must be aborted.
- Coordinator adds a decision record, <commit *T*> or <abort *T*>, to the log and forces record onto stable storage. Once the record stable storage it is irrevocable (even if failures occur)
- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.

Commit Protocol

Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <commit *T*> record: site executes redo (*T*)
- Log contains <abort *T*> record: site executes undo (*T*)
- Log contains <ready *T*> record: site must consult C_i to determine the fate of *T*.
  - If *T* committed, redo (*T*)
  - If *T* aborted, undo (*T*)
- The log contains no control records concerning *T* replies that S_k failed before responding to the prepare *T* message from C_i
  - since the failure of S_k precludes the sending of such a response C_1 must abort *T*
  - S_k must execute undo (*T*)

Commit Protocol

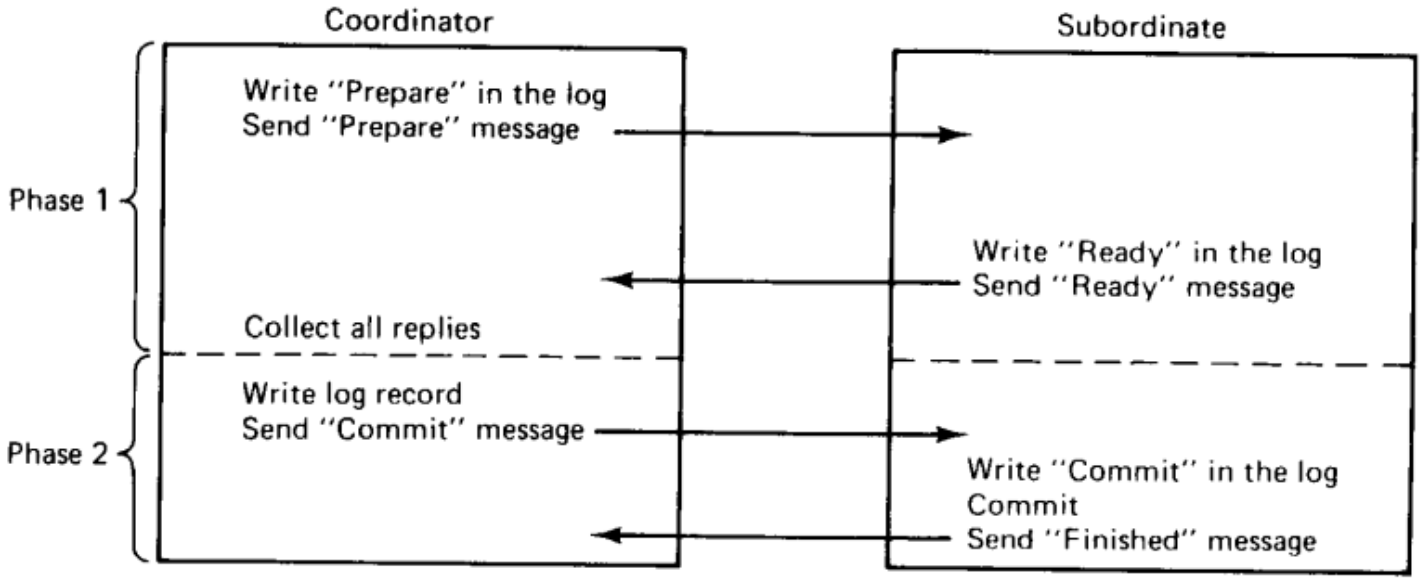


Fig. 3-20. The two-phase commit protocol when it succeeds.

Concurrency Control

- Locking
- Optimistic concurrency control
- Timestamps

Concurrency Control-Locking

- Oldest and most widely used CC algorithm
- A process before read/write ➔ requests the scheduler to grant a lock
- Upon finishing read/write ➔ the lock is released
- In order to ensure serialized transaction Two Phase Locking (2PL) is used
- Two phase locking:
  - In the first (growing) phase of the transaction, new locks are only acquired, and in the second (shrinking) phase, locks are only released.
  - A transaction is not allowed acquire *any* new locks, once it has released any one lock.

Concurrency Control-Locking

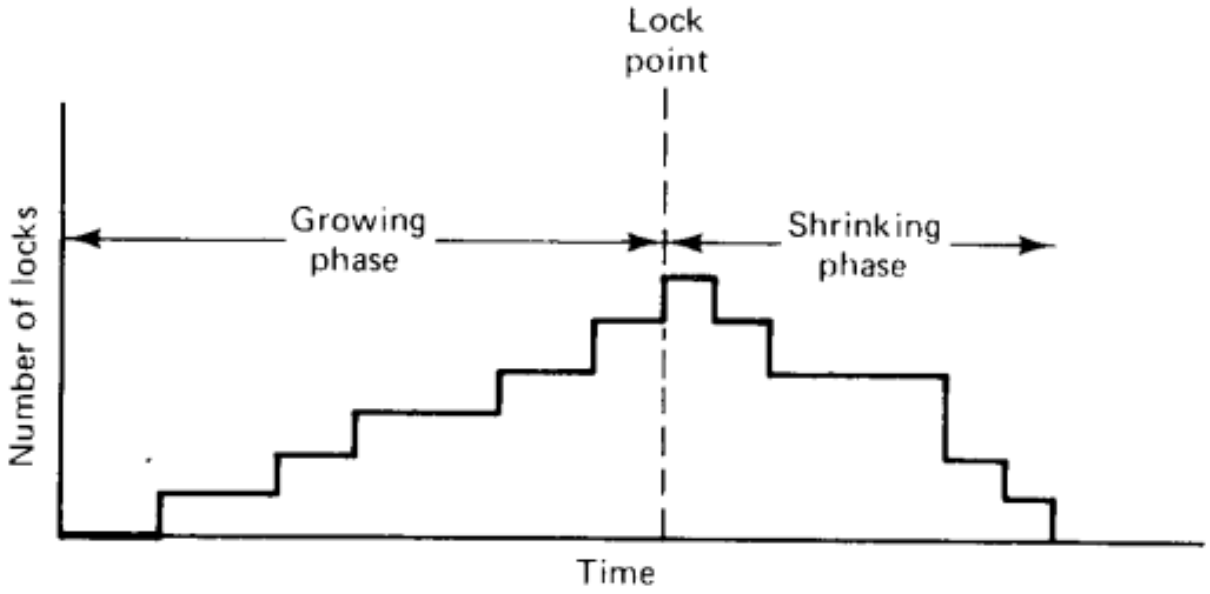


Fig. 3-21. Two-phase locking.

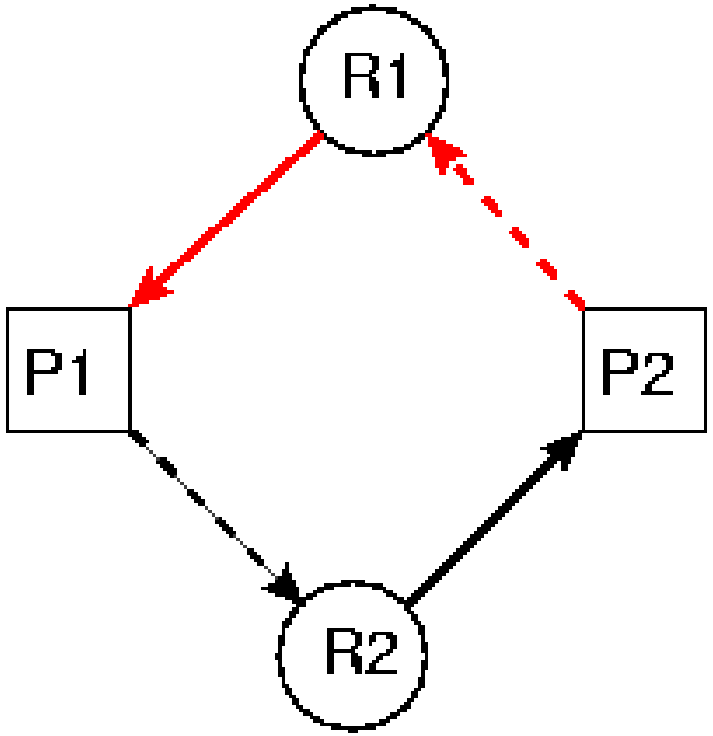
Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

Example:

- System has 2 tape drives.
- P1 requests for one tape drive and the system allocates tape drive 1.
- P2 request for one tape drive and the system allocates tape drive 2.
- P1 request for one more tape drive and enter in waiting state.
- P2 request for one more tape drive and it also enter in waiting state.

Deadlock Example



R1 is held by

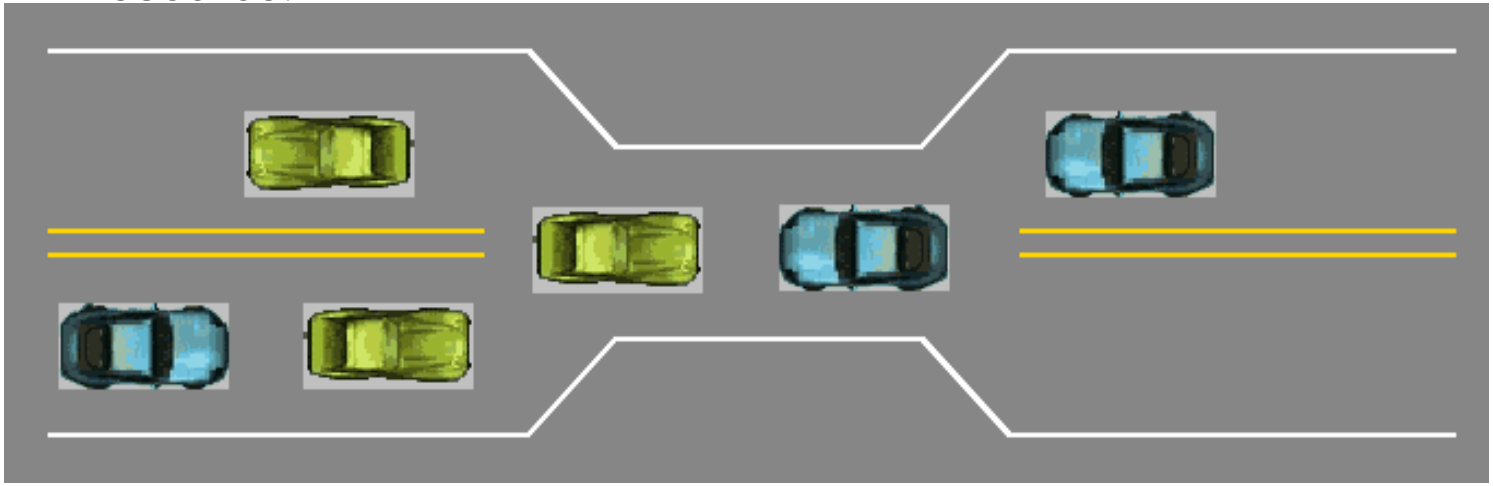
is waiting for R1

R2 is held by

is waiting for R2

Deadlock example (Bridge Crossing)

- Traffic only in one direction.
- Each section of bridge can be viewed as a resource.



- If a deadlock occurs. It can be resolved if one car back up.
- Called preempt resources and rollback.
- Several cars may have to be backed up if a deadlock occurs.

Deadlock cont..

Process use a resources in following sequence of events.

•Request: Process first makes a request for resources. If not available must wait. When available, request satisfied. But number of requests may not exceeded the total number of available units of resources.

•Allocate: System allocate the resources. It maintain a table. Where records of resources, and which resource is allocated to which process. Also having queue for waiting process.

•Release: After the process finished using the allocated resource, it release the resource to the system. System update the allocation.

Deadlock cont..

- If some of the processes entered in the waiting state, will never change state. Because requested resources are held by other waiting processes . This situation is called **deadlock**.
- Resource can be physical objects (Disk drive , tape, CPU, RAM) or logical objects (files, tables, database) .
- **Preemptable**, meaning that the **resource** can be taken away from its current owner (and given back later). An example is memory.
- **Non-preemptable**, meaning that the **resource** cannot be taken away. An example is a printer.

Necessary Conditions for Deadlock

Deadlock occurs if the following conditions hold simultaneously:

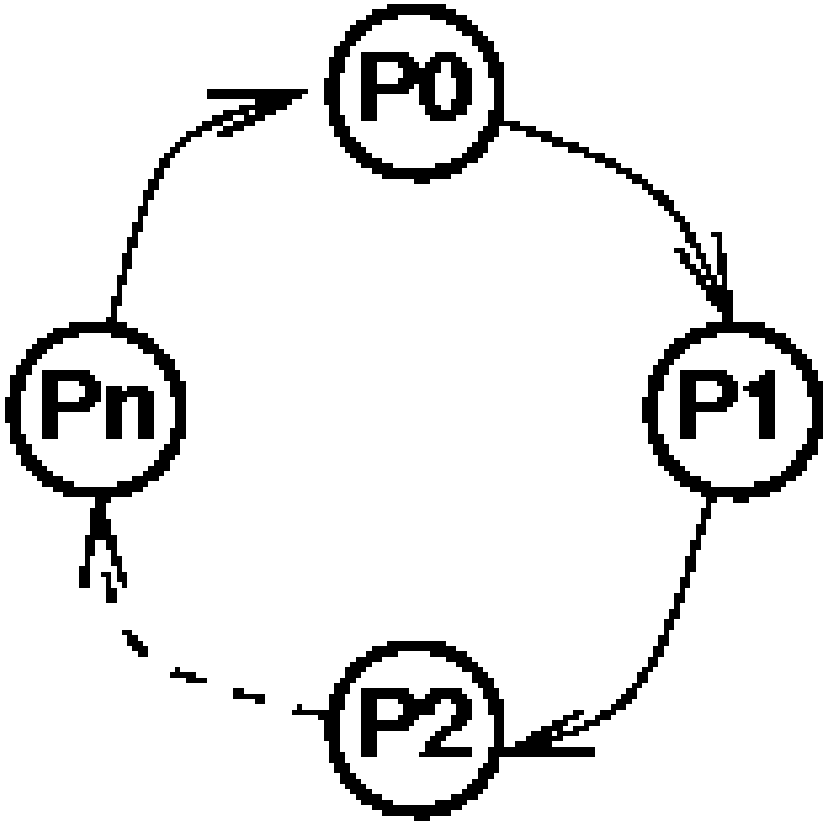
- Mutual exclusion condition
- Hold and wait condition
- No-Preemption condition
- Circular-wait condition

Necessary Conditions for Deadlock cont…

Deadlock occurs if the following conditions hold simultaneously:

- **Mutual exclusion:** only one process can use a resource at a time.
- **Hold and wait:** Processes are allowed to request for new resources without releasing the resources that they are currently holding.
- **No-preemption:** If a process has a resource, cannot be preempted.
- **Circular-wait:** Two or more processes from a circular chain.

Circular wait



Deadlock Modeling

- Directed graph (consists of node and edges)
- Path (a to b)
- Cycle (first and last node same)
- Knot (A knot contain one or more cycles)
- This graphs consist of nodes

(a,b,c,d,e,f)

- Set of edges

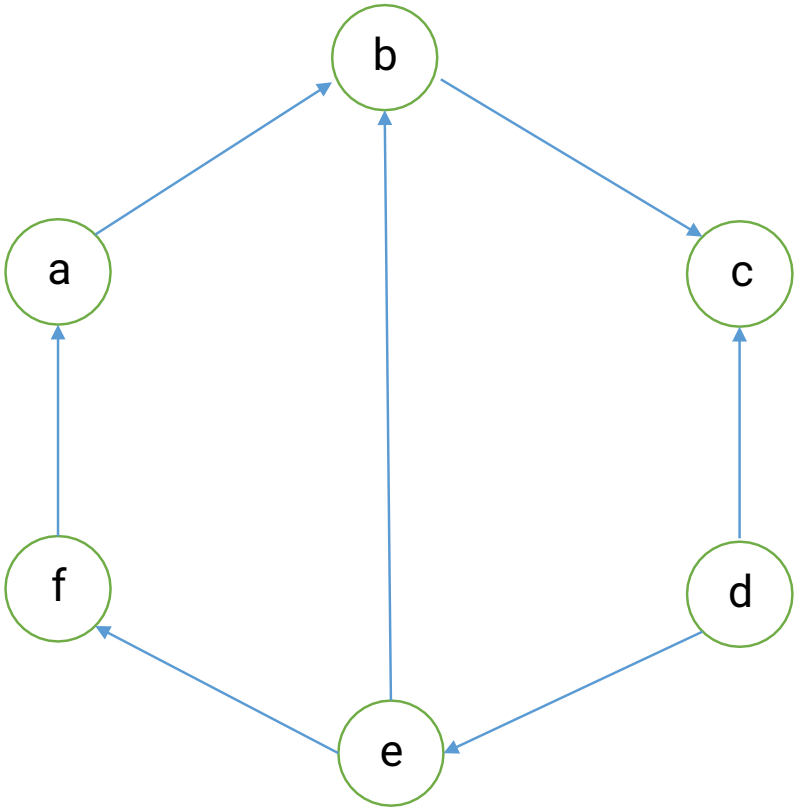
{(a,b),(b,c),(c,d),(d,e),(e,f),(f,a),(e,b)}

- Two cycles

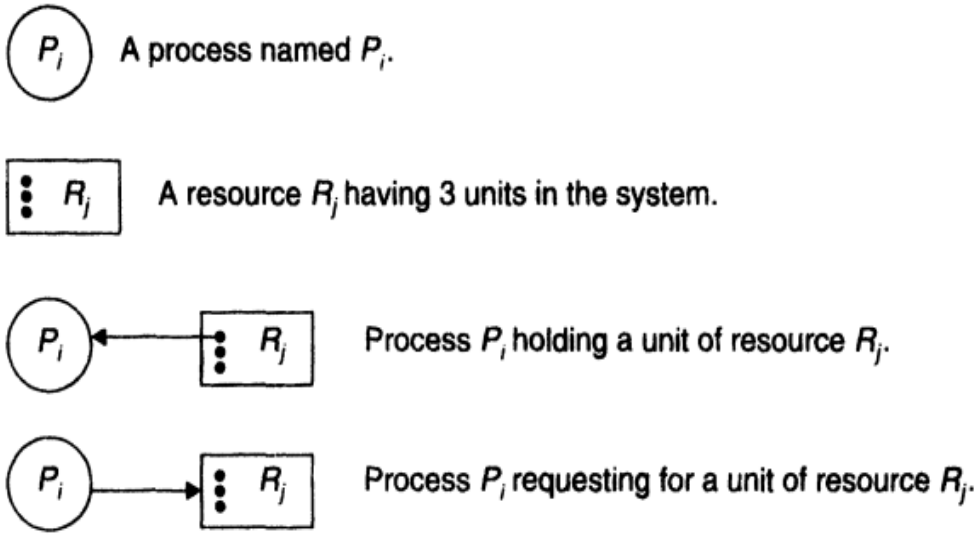
(a,b,c,d,e,f,a) and (b,c,d,e,b)

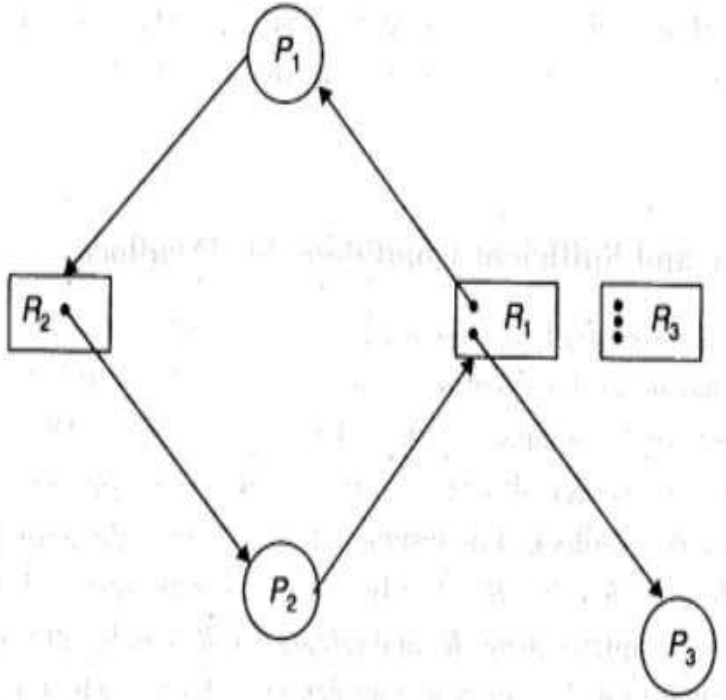
- One knot

(a,b,c,d,e,f) that contain two cycle.



Deadlock Modeling cont..





Deadlock Modeling cont..

- Process nodes (circle) P1,P2
- Resource nodes (rectangle) R1,R2,R3
- Units (bullets)

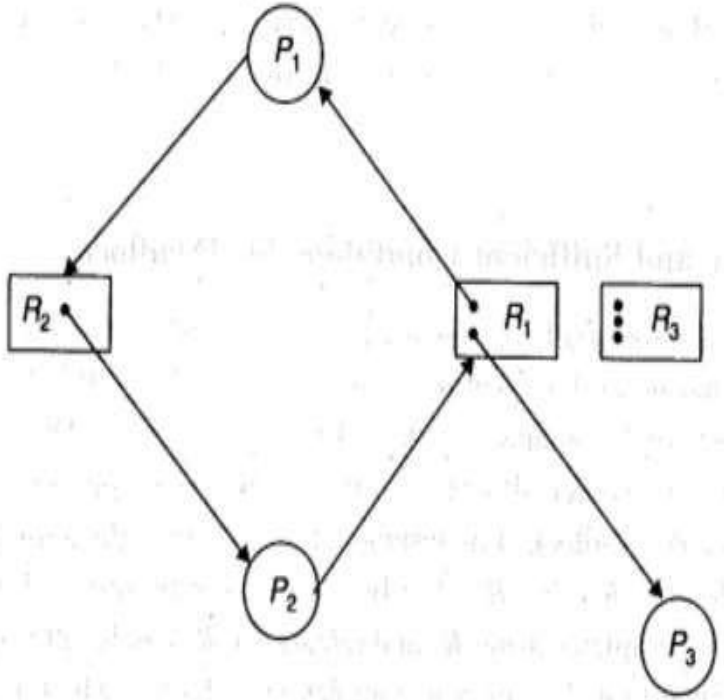
Resources have more than one units

- Assignment edges

Resource node to process node (R1,P1), (R1,P3), (R2,P

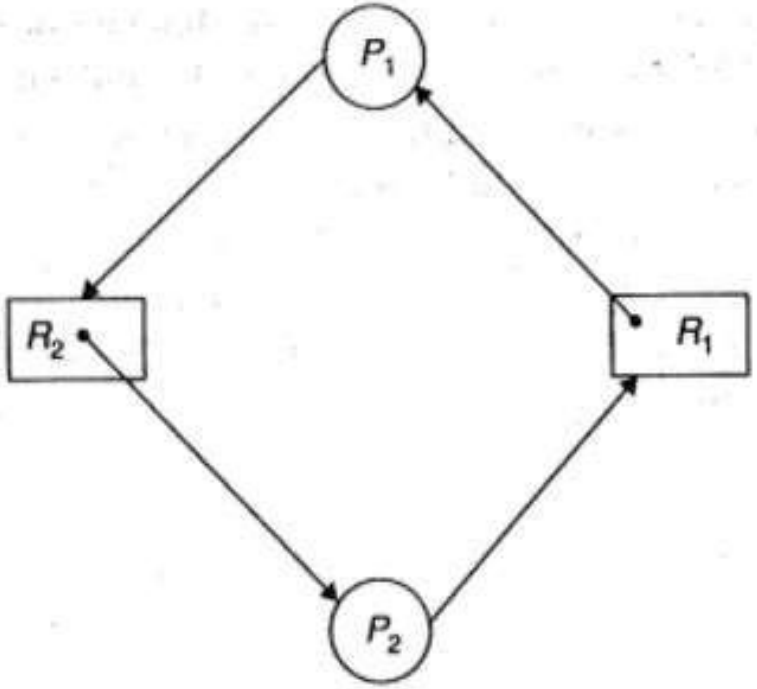
- Request edges

Process node to a resource node (P1,R2) and (P2,R1)



Necessary and Sufficient Conditions for Deadlock

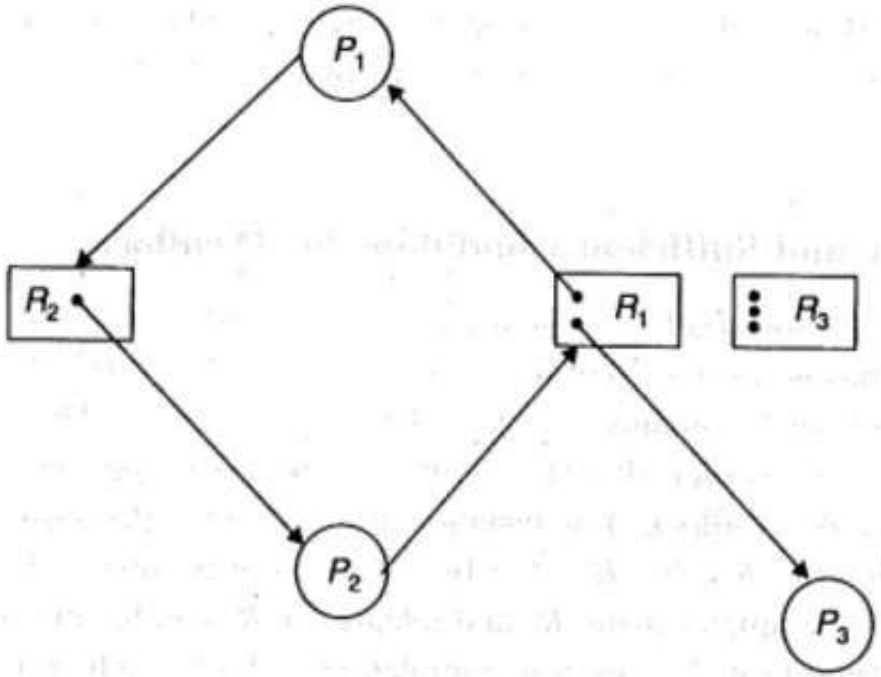
- A cycle is necessary condition for a deadlock.



Necessary and Sufficient Conditions for Deadlock cont..

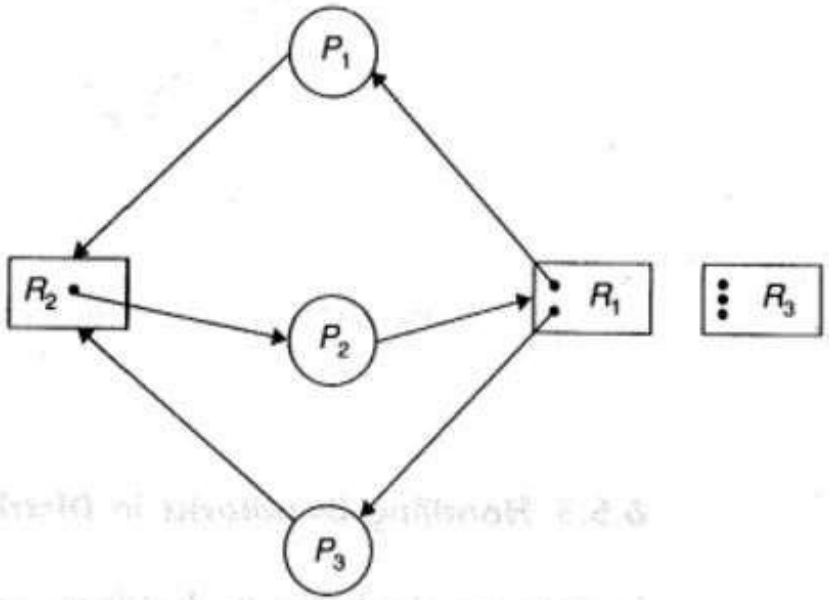
- A cycle is necessary condition for a deadlock.
- The presence of cycle is necessary condition for deadlock but not sufficient.
- This diagram contains cycle but does not represent a deadlock.

- When P3 completes its processing. It release R1. R1 assigned to P2.
- P2 complete its job and release R2,R1.
- R2 allocated to P1.



Necessary and Sufficient Conditions for Deadlock cont..

- If processes forming a cycle , and one or more resources Ri have more than one unit, and a knot will be there, deadlock occurs.
- P3 request for R2. (P3,R2) added to the graph.
- This graph has two cycles (P1,R2,P2,R1,P1) and (P3,R2,P2,R1,P3) in knot (P1,R2,P3,R1, P1).

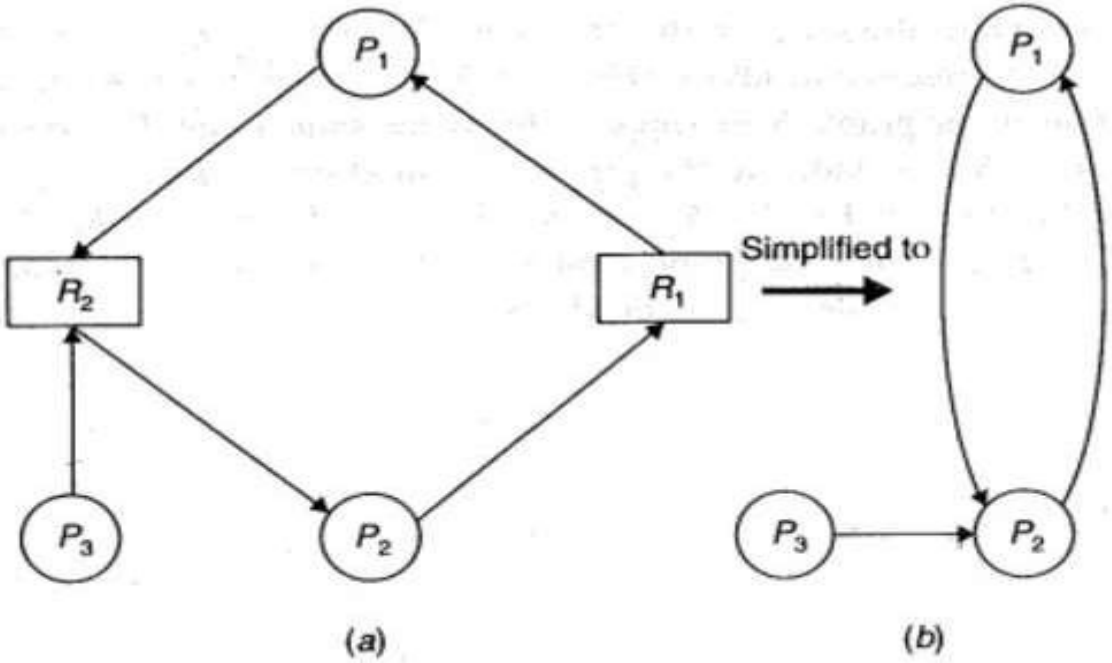


Necessary and Sufficient Conditions for Deadlock cont..

- Necessary and sufficient condition for deadlock can be summarized as follows:
- A cycle is necessary for deadlock.
- If there is a only single unit in one Resource in cycle.
- If more than one unit are there in cycle, then a knot is sufficient for a deadlock.

Wait-for graph (WFG)

- When each resource has one unit , simplified form of graph is used.
- Conversion from allocation graph to a WFG, removing the resource nodes.
- This simplification based on , a resource can always be identified by its current owner (process holding it).



Handling Deadlocks

- Prevention
- Avoidance
- Detection and recovery

Deadlock Prevention

- Designing the system in a way that deadlock become impossible.
- Prevention is different from avoidance because no runtime testing (dynamically) of potential allocations need be performed.
- Prevent the one of the four necessary conditions:
  - Mutual exclusion condition
  - Hold and wait condition
  - No-Preemption condition
  - Circular-wait condition

Deadlock Prevention cont..

Mutual Exclusion

•Unfortunately it is not possible to prevent deadlocks by denying mutual-exclusion.

•Cannot be prevented for all resources. Some resources are intrinsically non-sharable for example printer, and some are sharable like CPU.

Deadlock Prevention cont..

Hold and wait (Collective Requests)

•This method denies the hold-and-wait condition.

•We must guarantee that whenever a process requests a resource, it does not hold any other resources.

Following policies maybe used to ensure this:

•1. A process must request all of its resources before it begins execution. If all the needed resources are available , they are allocated to it. If one or more resources are not available , none will be allocated and process would just wait.

Deadlock Prevention cont..

Problems in Hold and wait (Collective Requests)

- Low resource utilization because a process may hold many resources but all of them not used.
- It may cause starvation because whenever a process made a request , resources are not available.

Deadlock Prevention cont..

No-Preemption condition (resources will be preempted)

- A preemptable resource is one whose state can be easily saved and restored later.
- A resource temporarily taken away from the process without harm to the computation performed by this process.
- CPU, Main Memory are preemptable resources.
- If the resources are preemptable, deadlocks can be prevented.

Two policies:

1. when a process requests for a resource, and resource is not available. Its already held resources taken away from it. And the process is blocked. The process is unblocked when the resources (requested by it, and preempted from it) available and allocated to it.

Deadlock Prevention cont..

No-Preemption condition (resources will be preempted)

2.When a process requests a resource that is not currently available, the system checks, if the requested resource is currently held by another process that is already blocked and waiting for other resources. If so , the requested resource is taken away from the waiting process and give to the requesting process, otherwise condition-1 applied.

Deadlock Prevention cont..

Circular wait condition (ordered requests)

- Each resource (type) is assigned a global number to impose total ordering.
- Impose a total ordering of all resource types , and require that each process can only requests resources in an increasing order of enumeration.
- Process should not request a resource with a number lower than is already held resource.
- If process request = j and process already held resource = i then, j > i
- Single request will be enough for the same resource type.
- It has proven that with this, graph can never have circular-wait.
- If tape drive =1 , disk drive = 5, printer = 12, after getting tape drive process can request for the disk drive, after disk drive process can request for the printer. But it a process have the number =5, it cannot request for the number=1 for tape drive.

Deadlock Avoidance

- The system dynamically considers every request and decides whether it is safe to grant it at this point.
- Deadlock avoidance use some advance knowledge of the resource usage of processes.

Following steps are for deadlock avoidance:

- When a process request for resource, even if the resource is available for allocation. It is not immediately allocated to that process.
- Advance knowledge used to perform analysis to decide whether process's request is safe or unsafe.
- When request is safe, resource allocated, otherwise deferred.

Deadlock Avoidance contd..

Safe State:

- A system is considered in safe state , if there exists some ordering of processes in which run all requests lead to completion .

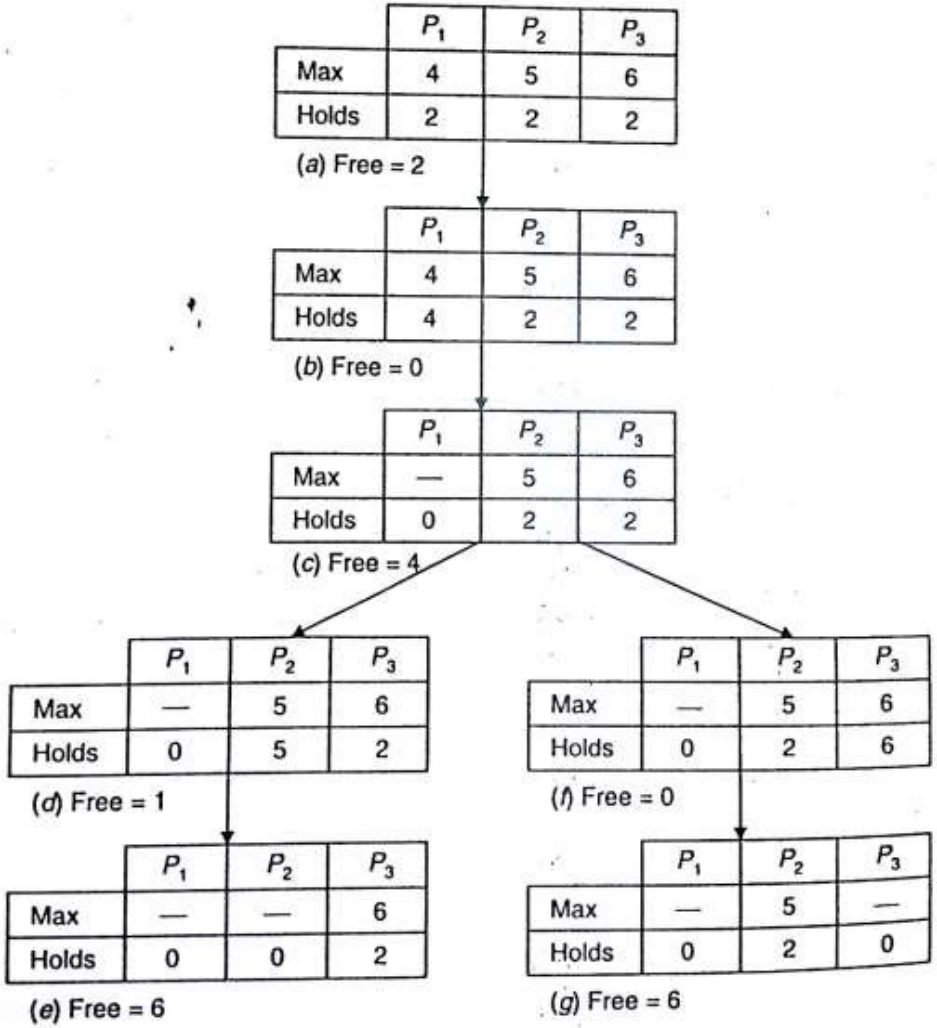
Safe Ordering (sequence):

- Any ordering of processes, that can guarantee the completion of all the process is called safe sequence.

Deadlock Avoidance contd..

- In a system there are 8 units of particular resource type.
- Three processes P1,P2 and P3 are competing.

Deadlock Avoidance contd..



	P_1	P_2	P_3
Max	4	5	6
Holds	2	2	2

(*a*) Free = 2

	P_1	P_2	P_3
Max	4	5	6
Holds	4	2	2

(*b*) Free = 0

	P_1	P_2	P_3
Max	—	5	6
Holds	0	2	2

(*c*) Free = 4

	P_1	P_2	P_3
Max	—	5	6
Holds	0	5	2

(*d*) Free = 1

	P_1	P_2	P_3
Max	—	—	6
Holds	0	0	2

(*e*) Free = 6

	P_1	P_2	P_3
Max	—	5	6
Holds	0	2	6

(*f*) Free = 0

	P_1	P_2	P_3
Max	—	5	—
Holds	0	2	0

(*g*) Free = 6

Deadlock Avoidance contd..

- If resource allocation is not done carefully , the system move from a safe state to unsafe state.

	P_1	P_2	P_3
Max	4	5	6
Holds	2	2	2

(*a*) Free = 2

	P_1	P_2	P_3
Max	4	5	6
Holds	2	3	2

(*b*) Free = 1

Deadlock Avoidance contd..

It is important to note the following remarks about safe and unsafe states:

- The initial state in which no resources are yet allocated is called safe state.
- Safe state, the system can guarantee that all processes can be run to completion.
- An unsafe state is not a deadlock state, but it may lead to a deadlock state, the system cannot guarantee that all processes can be run to completion.

Deadlock Avoidance contd..

Due to following reason , avoidance rarely used:

- Assumption of advance knowledge is available, however in practice, processes rarely know in advance what their maximum resource needs will be.
- Number of processes not known in advance, because dynamically varies. (user login , logout)
- Number of units change dynamically (machine crash).
- Restriction on resource allocation , cause degrades the system performance.

Deadlock Detection

- Uses algorithm that keeps examining state of system to determine whether a deadlock has occurred.
- When deadlock has occurred system takes action to recover.
- Algorithms are same for centralized as well as distributed.
- Uses resource allocation graph and searching for cycle or not depending upon single or multiple units of resources.

Following steps are needed to construct Weight For Graph(WFG).

1. Construct separate Resource allocation graph for each site. Resource node exists for all local resources and process node exists for all processes that are either holding or waiting for resource.
2. Construct WFG by removing resources and collapsing edges.
3. Take union of all WFG of all sites and construct single global WFG.

Deadlock Detection contd..

Two sites, site 1 with 2 resources and site 2 with 1 resource.

- P1 is holding R1 and requesting for R3.
- P2 is holding R2 and requesting for R1.
- P3 is holding R3 and requesting for R2

Union of WFG of two sites will show either deadlock exists or not. Local WFG does not contain cycle while Global has.

Difficulties: Maintaining WFG of each site is difficult.

Most important feature of deadlock detection algorithm is correctness which depends on these properties.

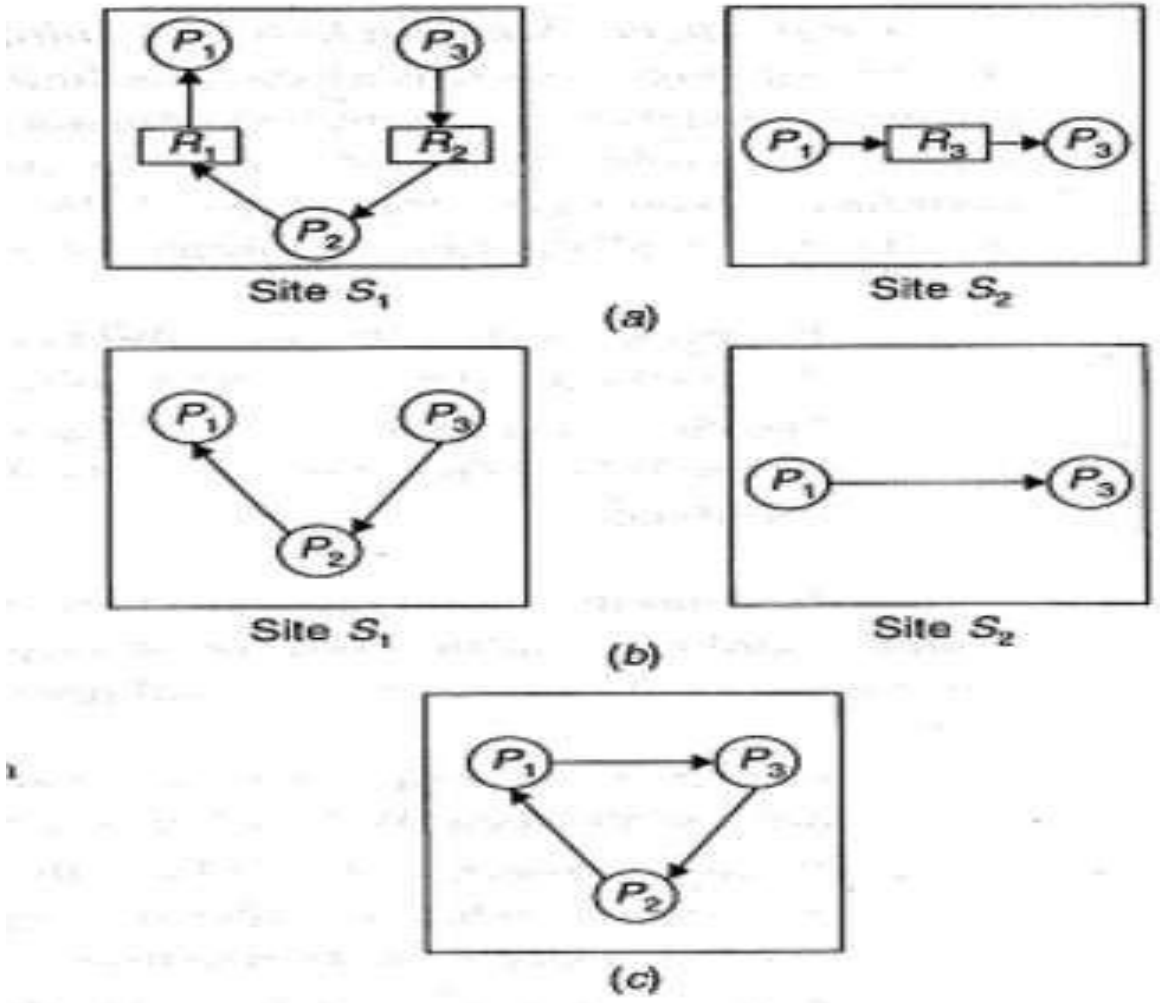
- **Progress Property:**

Deadlocks must be detected in finite amount of time.

- **Safety Property:**

Detected deadlock must exist. Message delays and out of date WFG cause false cycles to be detected. It results in detection of deadlocks that do not exist called **phantom deadlocks.**

Diagram


P_1
P_3
R_1
R_2
P_2
Site S_1
(a)
P_1
R_3
P_3
Site S_2
P_1
P_3
P_2
Site S_1
(b)
P_1
P_3
Site S_2
P_1
P_3
P_2
(c)

Deadlock Detection Cont..

Three commonly used techniques for organizing WFG in a distributes system are..

- Centralized Algorithms
- Distributed Algorithms
- Hierarchical Algorithms

Centralized Approach

- Local coordinator at each site that maintains WFG of its local resources.
- Central coordinator that is responsible for constructing union of all WFG.
- Central coordinator constructs global WFG from information received form local coordinator of all sites.

Deadlock detection is performed as follows.

1. If cycle exists in local WFG of any site, it represents local deadlock. Such deadlocks are detected and resolved by local coordinator.
2. Deadlocks involving resources at two or more sites get reflected as cycles in global WFG. Such deadlocks are detected and handled by central coordinator.
3. In centralized approach messages are sent form local to central coordinator to transfer information as follows.

Centralized Approach Cont..

- **Continuous Transfer**: local coordinator send message whenever an new edge is added or deleted
- **Periodic Transfer:** local coordinator sends message after fix time when no of changes have occurred.(in order to reduce messages)
- **Transfer on Request:** on request of central coordinator. Central coordinator invokes cycle detection algorithm periodically and requests information from each site just before invoking algorithm.

Drawbacks:

- Failure of central coordinator. Back up can resolve issue.
- Central coordinator will become performance bottle neck in large system having to many sites.
- It may detect false deadlocks or phantom.

Centralized Approach Cont..

3 processes (P1,P2,P3) compete for 3 resources (R1,R2,R3)

- Step 1: P1 requests for R1 and R1 is allocated to it.
- Step 2: P2 requests for R2 and R2 is allocated to it.
- Step 3: P3 requests for R3 and R3 is allocated to it.
- Step 4: P2 requests for R1 and waits for it.
- Step 5: P1 requests for R2 and waits for it.
- Step 6: P1 releases R1 and R1 is allocated to P2.
- Step 7: P1 requests for R3 and waits for it.

Centralized Approach Cont..

- m1: from site S1 to add edge (R1,P1)
- m2: from site S1 to add edge (R2,P2)
- m3: from site S2 to add edge (R3,P3)
- m4: from site S1 to add edge (P2,R1)
- m5: from site S1 to add edge (P3,R2)
- m6: from site S1 to delete edges (R1,P1) and (P2,R1) and edge (R1,P2)
- m7: from site S2 to add edge (P1,R3)

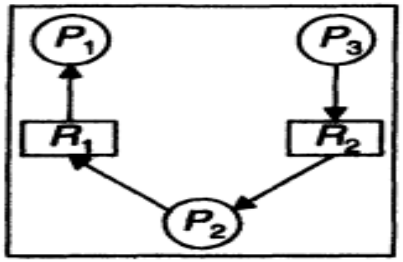
If all these messages are in order then no problem.

If message 7 from site 2 comes earlier than message 6 from site 6 than central coordinator will incorrectly conclude that deadlock has occurred and will start to recover. In this example we followed continuous transfer same problem will also be with other two methods.

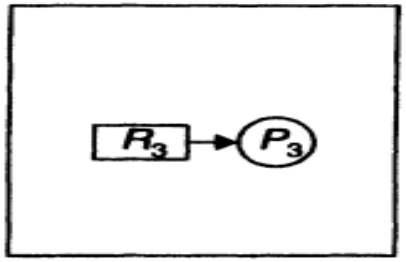
Solution: assign unique global timestamp with message.

Central coordinator send message to each site if some one has time stamp earlier than received message, site with minimum will reply and other site with negative

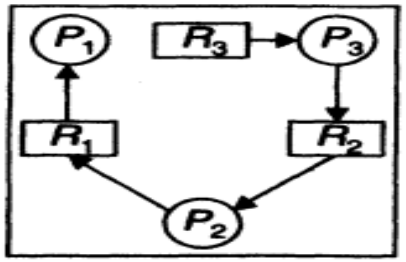
Diagram



Resource allocation graph of the local coordinator of site S_1

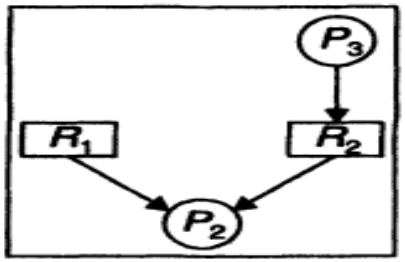


Resource allocation graph of the local coordinator of site S_2

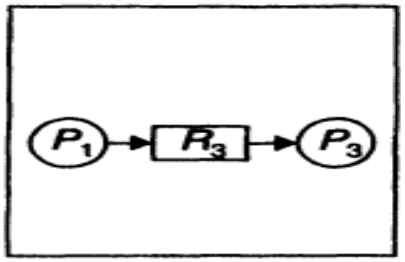


Resource allocation graph maintained by the central coordinator

(a)

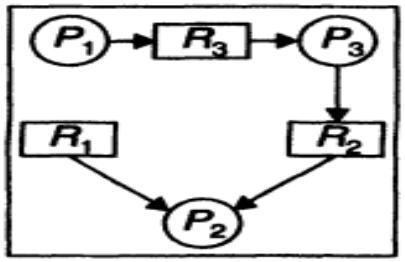


Site S_1

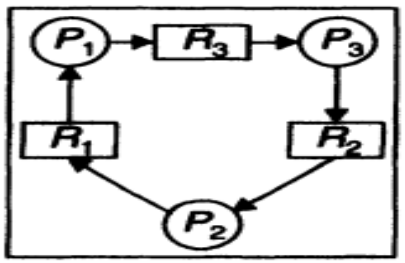


Site S_2

(b)



Central coordinator



Central coordinator

(c)

Hierarchical Approach

Drawbacks of Centralized approach

•Most WFG cycles are very short.

•90% of all deadlock cycles involve only two processes.

•So centralized approach seems less attractive because of time and overhead involved in assembling local WFG at central coordinator.

•Deadlock should be detected by a site located as close as possible to sites involved in cycle to minimize communication cost.

Solution:

•Hierarchical approach solves this issue.

Hierarchical Approach

- Uses logical hierarchy(tree) of deadlock detectors.
- Deadlock detectors are called controllers.
- Each controller is responsible for deadlocks that involves sites falling within its range.
- Unlike centralized approach instead of maintaining WFG at a single site, it is distributed over no of controllers.

WFG is maintained on following rules.

- Each controller that forms a tree of hierarchy tree maintains local WFG of single site.
- Each non-leaf controller maintains WFG that is union of WFGs of its immediate children in hierarchy tree.

Diagram

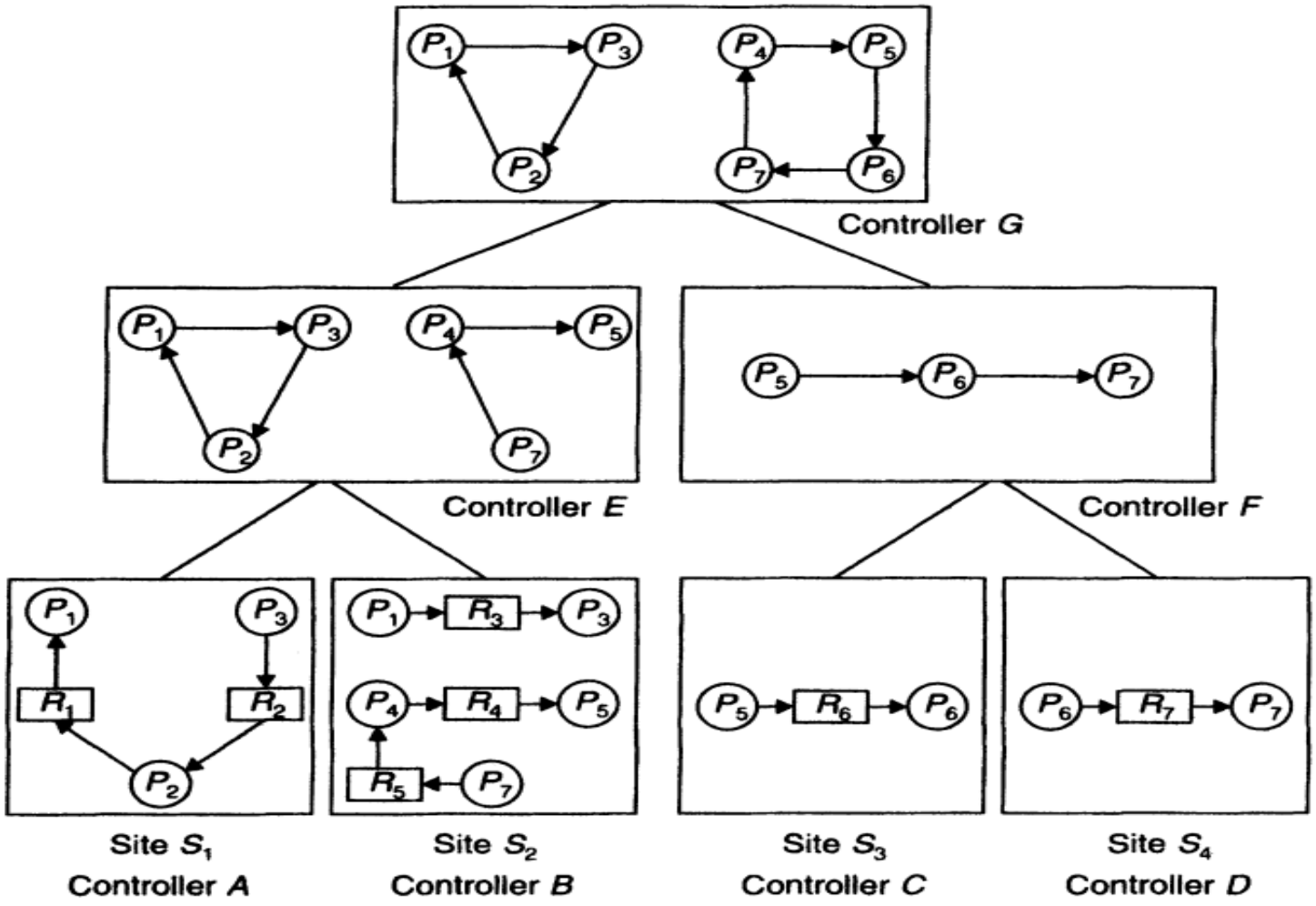


Fig. 6.17 Hierarchical deadlock detection approach.

Fully Distributed Approaches for deadlock detection

- Each site of system shares equal responsibility for deadlock detection.

Types:

- WFG based Distributed algorithm for deadlock detection.
- Probe based Distributed algorithm for deadlock detection.

WFG based Distributed algorithm for deadlock detection.

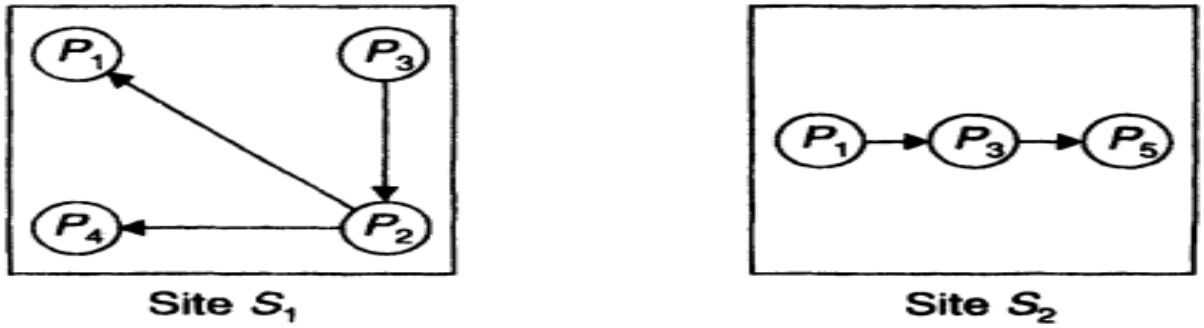
Each site maintains its own WFG.

•Extra node P_{ex} is added to local WFG of each site, and this node is connected to WFG of corresponding site in following manner.

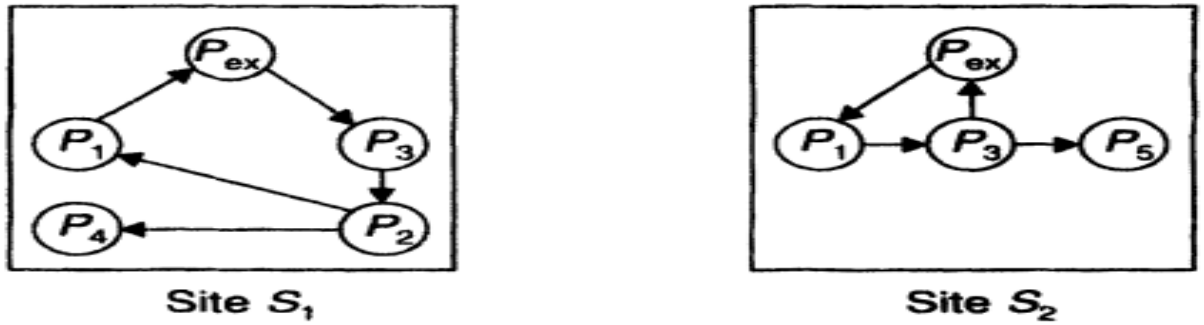
1) An edge(Pi, P_{ex}) is added if processes Pi is waiting for resource in another site held by any other process.

2) An edge(P_{ex},Pj) is added if Pj is a process of another site that is waiting for a resource currently held by process of this site.

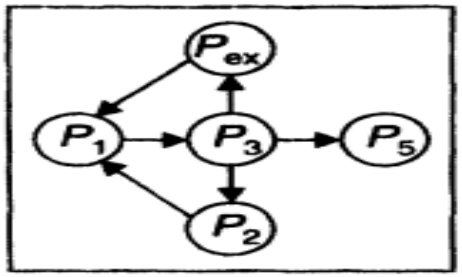
Example



(a)



(b)



(c)

Example illustrating the WFG-based fully distributed deadlock detection algorithm: (*a*) local WFGs; (*b*) local WFGs after addition of node P_{ex}; (*c*) updated local WFG of site S_2 after receiving the deadlock detection message from site S_1.

WFG based Distributed algorithm for deadlock detection cont...

1) in the WFG of site S1, edge (P1,Pex) is added because process P1 is waiting for a resource in site S2, that is held by process P3, and edge(Pex,P3) IS ADDED BECAUSE process P3 is process of site S2, that is waiting to acquire a resource currently held by process P2 of site S1.

2) in the WFG of site S2, edge (P3,Pex) is added because process P3 is waiting for a resource in site S1, that is held by process P2, and edge(Pex,P1) IS Added Because process P3 is process of site S2, that is waiting to acquire a resource currently held by process P2 of site S1.

Deadlock Handling:

•If local WFG contains a cycle that does not involve node Pex, a deadlock that involves only local process of that site has occurred. Such deadlocks are handled locally.

•If local WFG contains a cycle that involves node Pex, there is possibility of distributed deadlock that involves process of multiple sites. For confirmation deadlock distribution algorithm is invoked by site whose WFG contains a cycle involving node Pex.

Probe based Distributed algorithm for deadlock detection

- Also called Chandy-Misra-Hass (or CMH) algorithm
- Best algorithm for detecting global deadlocks in distributed system.
- Algorithm allows a process to request multiple resources at a time.
- When a process requests for a resource or resources fails to get requested resource and times out, it generates special probe message and sends it to process holding the requested resources.

Probe message contains following fields.

- The identifier of process just blocked.
- The identifier of process sending message.
- The identifier of process to whom the message is being sent.

Probe based Distributed algorithm for deadlock detection cont...

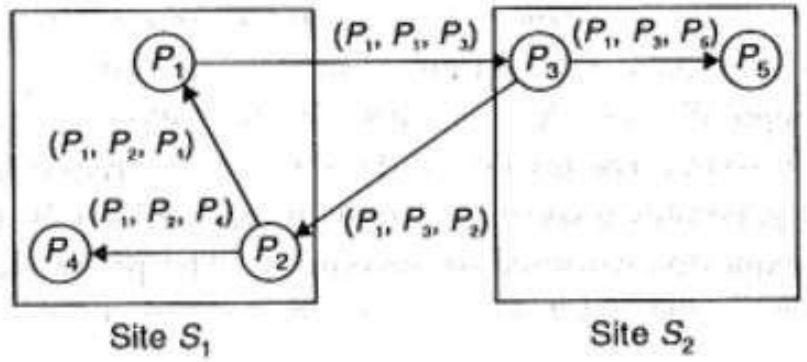
- On receiving a probe message, receipt checks to see if it itself is waiting for any resource or resources. If not this means that recipient is using the resource requested by the process that sent probe message to it. Receiver simply ignores probe message..
- On the other hand if receipt is waiting for any resource it passes the probe message to the process or processes holding the resource for which it is waiting . However probe message is forwarded.

Receipt modifies its field in following manner.

- The first field is left unchanged.
- The receipt changes the 2nd field to its own process identifier.
- 3rd field is changed to identifier of process that will be new receipt of this message.

Every new receipt of probe message repeats this procedure. If probe message returns back to its original sender a cycle exists and system is deadlocked.

Probe based Distributed algorithm for deadlock detection cont...



Attractive Features:

- Easy to implement, since message is of fix length and requires few steps for computation.
- Low overhead
- No need of graph construction
- False deadlocks are not detected.
- No specific structure required among processes.

Ways for recovery from deadlock

- Asking for operator intervention
- Termination of process or processes.
- Rollback of processes.

1)Asking for operator intervention: system assist the operator in decision making for recovery by providing him list of processes involved in deadlock. It is handled manually. Console is used to continuously monitor smooth running. Every site has an operator to handle deadlock.

Drawbacks:

- Not used in modern systems.
- Not suitable for distributed environment because each site operator will take some action for recovery.
- If operator of single site is informed. It may favor it's own processes.
- Operator of one site may not have rite to interfere with other

Ways for recovery from deadlock

2) Termination of process or processes:

•Simplest way to recover deadlock automatically by killing one or more processes and reclaim the resources held by them

•Algorithms of this type analyze resource requirements and interdependencies of processes involved in deadlock cycle and then select a set of processes which if killed can break cycle.

3) **Rollback of processes:**

•killing of processes restarting from start is expensive specially when the process has already run for long time.

•To reclaim a resource from process that were selected for being killed rollback the process to point where resource was not allocated to the process.

•Processes are check pointed periodically. Process state memory image and list of resources held by it are written to a file at regular interval. So in case of deadlock process can be restarted from any of check points.

Issues In Recovery From Deadlock

1. Selection of victim
   - Minimization of recovery cost
   - Prevention of starvation
2. Use of transaction mechanism

•**Selection of victim:** Deadlock is broken by killing or rolling back one or more processes. These process are called victims.

Minimization of recovery cost:

•Processes whose termination/rollback will incur minimum recovery cost must be selected.

•Unfortunately it is not possible to have universal cost function.

•Each system should determine its own cost function to select victim.

Factors:

1.Priority of processes.

2.Nature of processes(interactive or batch) No and types of resources held by processes

3.Length of service already received and expected length of service further

4.needed by processes.

5.Total no of processes that will be affected.

Issues In Recovery From Deadlock cont…

Prevention of starvation:

- Same processes may be repeatedly selected as victim on the basis minimization of recovery cost and may never complete called starvation.
- Raise the priority of processes every time it is victimized.
- Include no of times a process is victimized as parameter to cost function.

Use of Transaction mechanism:

- Rerunning a process from rollback state may not always be safe.
- Operations performed by that process may be non idempotent.
- It must be used with only those processes which will not cause ill effects.