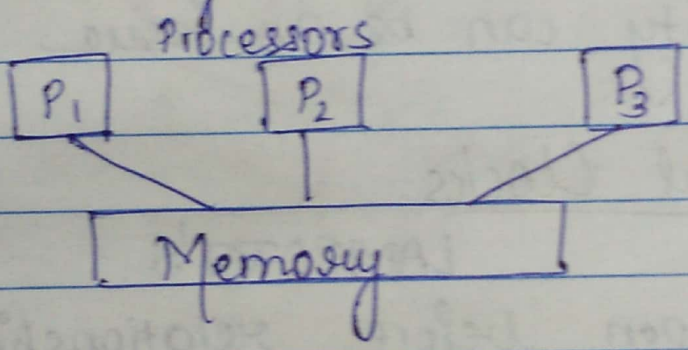
Resource:

1. Software / Hardware resource (Printer)
2. Local resource / Remote resource.
3. Consumable / Reusable resource (Interrupt messages) (Printer)

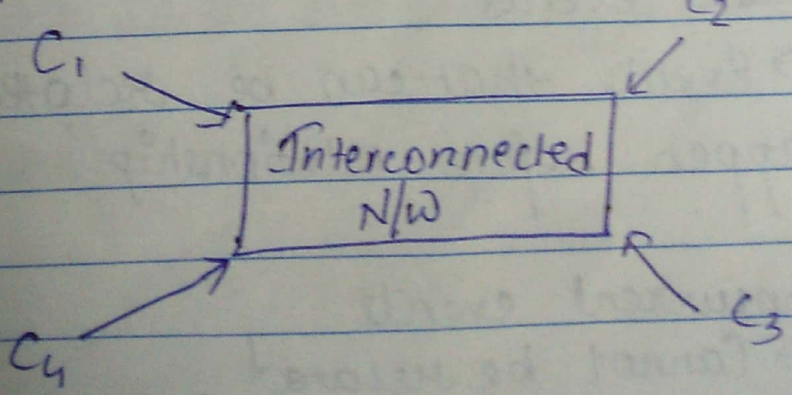
Parallel Systems:



If memory gets corrupted, all the processor would be affected.

Distributed Systems:

Each processor has its own memory block.



- Data backup is available in multiple systems
- Failure in one does not affect others.
- However, because all systems have their own memory, if memory of any one is affected, updated info is not available to all other systems.
- Another disadvantage - Non availability of a common synchronized clock.
- Compatibility of systems can be an issue.
- Security can be an issue.

Logical Clocks.

LAMPORT'S.

Happen before relationship.

$a \longrightarrow b$

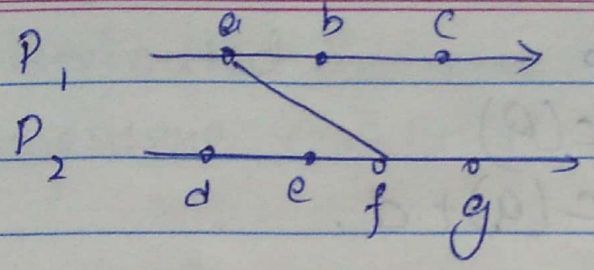
a Happened before b.

(a & b can belong to same or different processes)

(i) Causal events
→ Events that can be related with 'happen before relationship.'

(ii) Concurrent events
→ Cannot be related.

Date.
Page No.



$a \rightarrow b$ $\quad$ $a \rightarrow f$

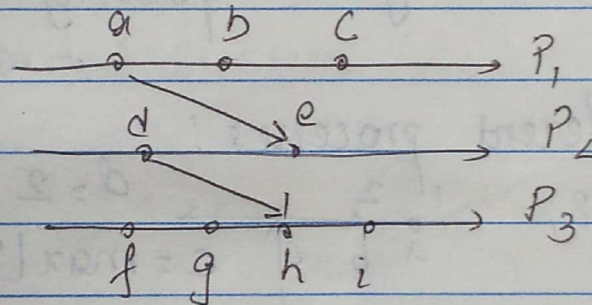
$b \rightarrow c$ $\quad$ $f \rightarrow g$

$\Rightarrow a \rightarrow c$ $\quad$ $\Rightarrow a \rightarrow g$

$\longrightarrow$ Causal events.

Concurrent events : a||d , b||d , c||e

Q



Causal :

$a \rightarrow b$, $b \rightarrow c$, $a \rightarrow c$

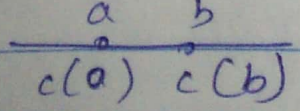
$a \rightarrow e$ $\quad$ $d \rightarrow e$ $\quad$ $d \rightarrow h$ $\quad$ $h \rightarrow i$

$d \rightarrow i$ $\quad$ $f \rightarrow g$ $\quad$ $g \rightarrow h$

Concurrent :

a||d , d||g , e||f , c||d . . . .

Timestamp = C.



Date.
Page No.

$a \rightarrow b$

$c(b) > c(a)$

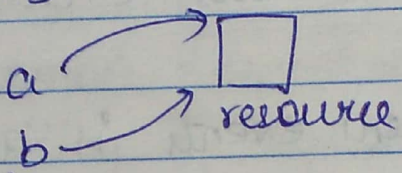
$c(b) = c(a) + d.$

In same process:

$a = 1$

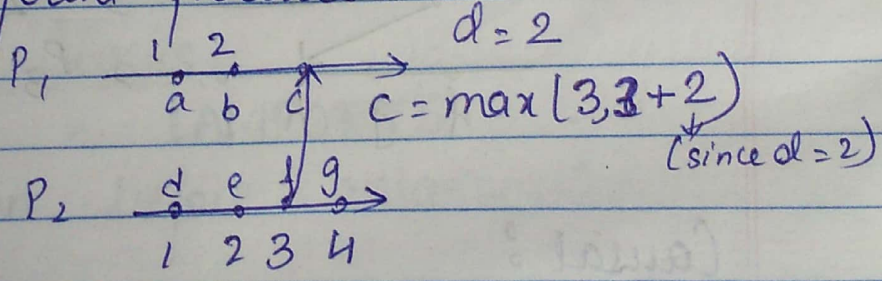
$d = 2$

$\Rightarrow b = 3$



a is given priority.

In different processes:



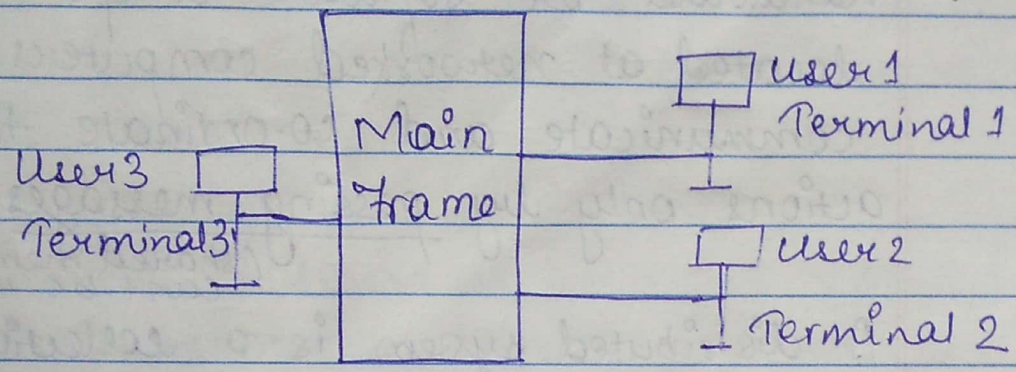
internet – E.g. - two LANs connected

Internet - world wide web

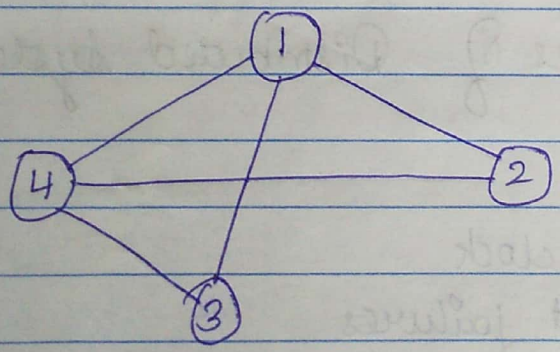
Date.
Page No.

Distributed Systems : Independent computers connected over a network.

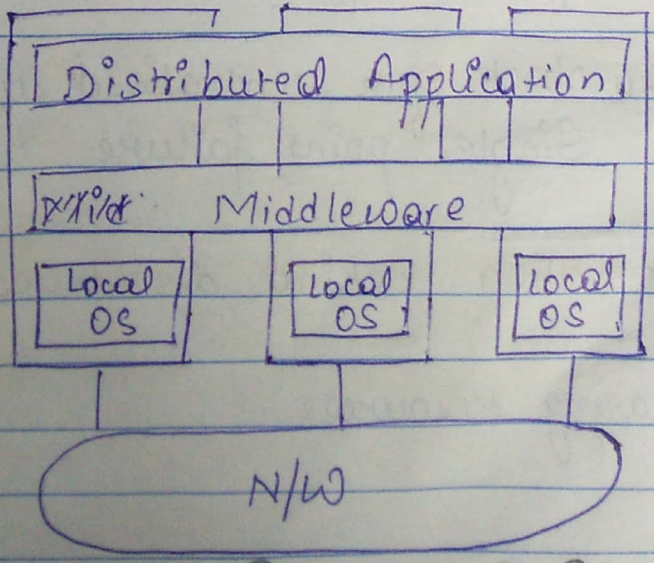
Centralised system :



Distributed system :



1, 2, 3, 4 → Independent computers



Schematic Diagram of Distributed System

Date.
Page No.

The heterogenity of different OS is masked by middle ware.

A distributed system is one in which hardware or software components located at networked computers communicate and co-ordinate their actions only by passing messages. (shared memory can't be used)

A distributed system is a collection of independent computers that appears to its users as a single coherent system

Consequences of Distributed System:

1. Concurrency
2. No global clock
3. Independent failures.

(i)
(ii)
(iii)

Drawback of a centralised system:
Single point failure

Motivation behind distributed system:

(i)

- Sharing resources

Page No.

Service - Service manages a collection of related resources and presents their functionality to users and applications for e.g file services.

The only access we have to the service is via the set of operations that it exports.

Server - A server is a running program on a networked computer that accepts requests from programs running on other computers to perform a service and respond appropriately. The requesting processors are referred to as clients.

Examples of distributed system :

(i) Internet
(ii) Intranet
(iii) Mobile & Ubiquitous computing

Challenges in a distributed system :

(i) Heterogenity - Network, hardware, OS, programming lang.

Middleware - www

Date.
Page No.

Masking Heterogenity :-

(a) Middleware is a software layer that provides a programming abstraction as well as masks the heterogenity of the underlying network, hardware, OS, prog. language.

E.g - CORBA (Common Object Request Broker)

The middleware provides a uniform computational model for use by the programmers of servers and distributed applications.

(b) Mobile Codes - E.g Java Applets

A mobile code can be sent from one computer to another and run at the destination

(c) Virtual Machine.

2. (d) ~~Openness~~

(ii) Openness - It is the characteristics which determine whether the system can be extended and reimplemented in various ways. The openness of a distributed system is determined primarily by the degree to which new resource sharing

Date.
Page No.

~~services~~ ~~abilities~~ can be added and made available ~~to~~ for use by a variety of client programs.

Openness can't be achieved unless the specification and documentation of key software interfaces of the components of a system are made available to s/w developers.

A distributed system may be extended in hardware level by addition of computers to the network and at s/w level by introduction of new services and reimplementation of old ones, enabling application programs to share resources.

(iii) Security

(a) Confidentiality - protection against disclosure to unauthorised users

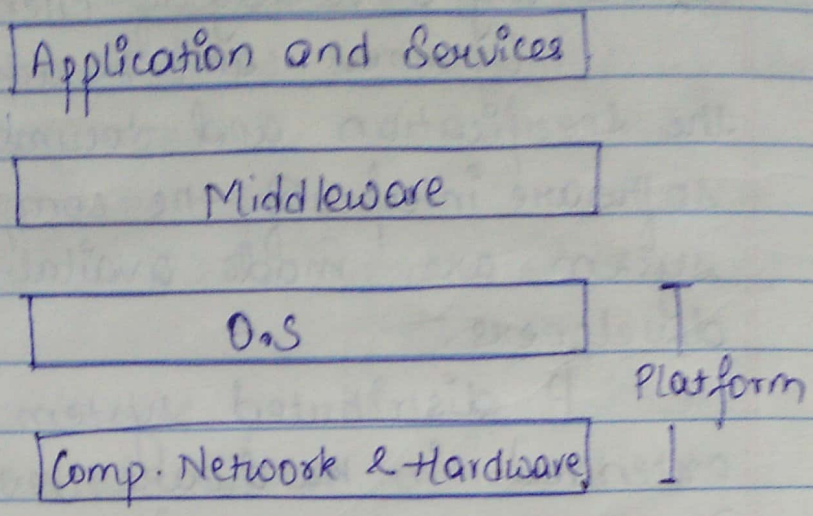
(b) Integrity - protection against alteration or corruption.

(c) Availability - protection against interference with means to access the resources.

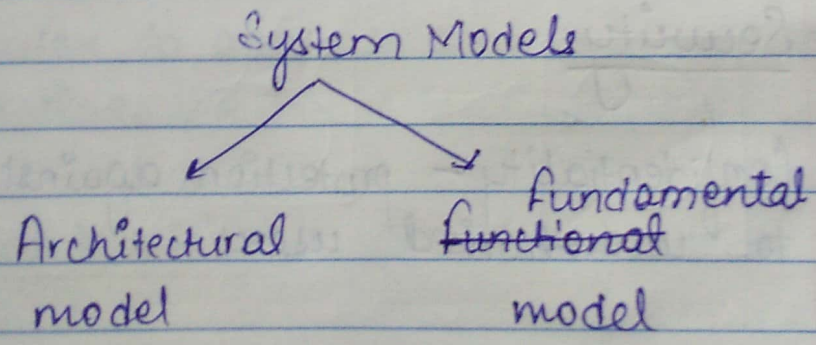
(iv) Scalability -

Date.
Page No.

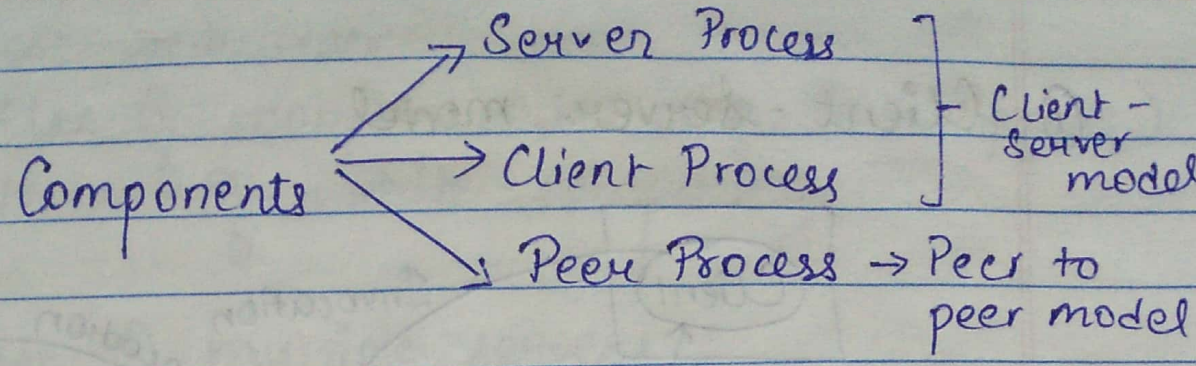
System Models.



- Client - Server – Master - slave model
- Peer to peer
- Hybrid



Architectural Model →
- Simplication of components
- Placing the components
- Interconnection beroeen different components

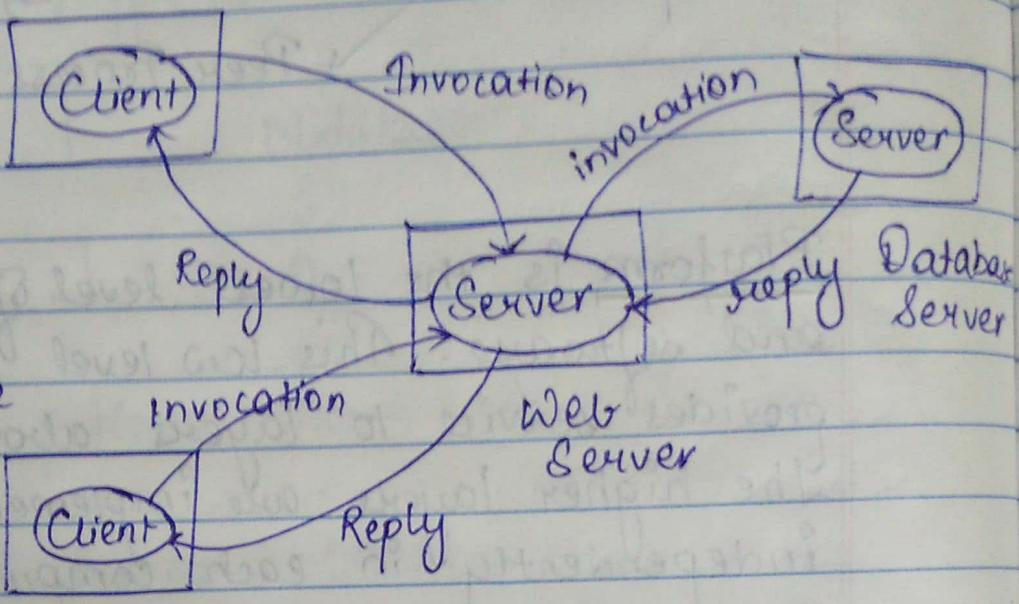


Platform is the lowest level of hardware and software. This low level layer provides service to layers above them. The higher layers are implemented independently in each computer bringing the system's programming up to a ~~layer~~ level that facilitates communication and co-ordination between processes.

Middleware is there to mask heterogenity and to provide a convenient programming model to application programmers. Middleware is represented by processes or objects in a set of computers that interact with each other to implement communication and resource sharing support for distributed applications.

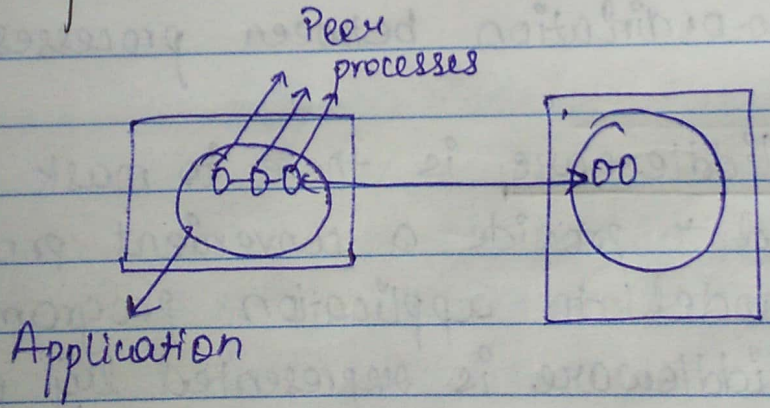
CORBA
JAVA RMI
Web Services
} E.g of middleware

(1) Client - Server model



Pros : Simple
Cons : less scalable

(2) Peer to peer model



Pros : Much more scalable
Cons : Much more complex

Variations in the model :

(1) Use of multiple server and caches to increment the performance and resilience.
(2) Hardware constrained machines for

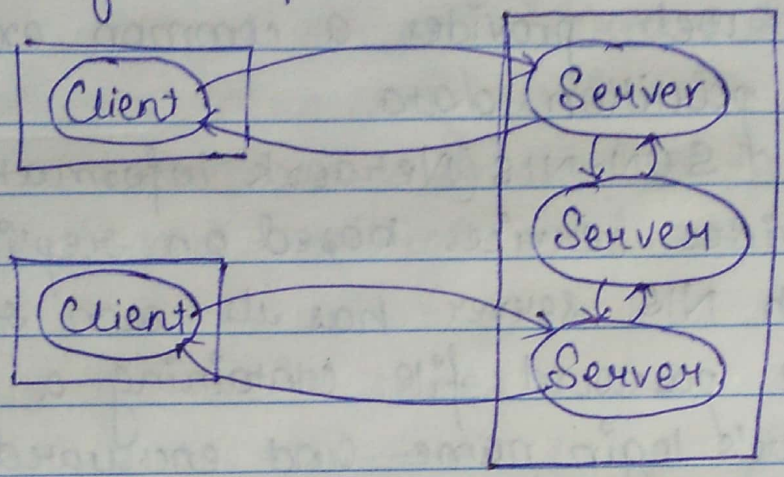
Date.
Page No.

cost reduction.

③ Use of mobile codes (E.g. Java Applets)

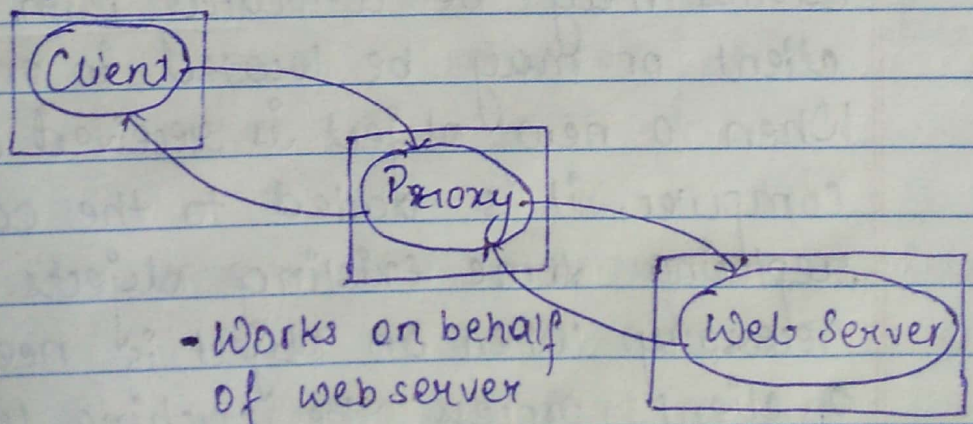
④ Use of mobile computing

(a) Use of multiple servers.



E.g SUN-NIS (Provides password services)

(b) Proxy server as cache



- Works on behalf of web server
- Performs caching (storing a copy of frequently visited websites to increase response time)
- Acts as a firewall

Multiple servers —

The servers may partition the set of objects on which the service is based and distribute between themselves or they may maintain replicated copies of them on several hosts.

The web provides a common example of partition data.

SUN-NIS (Network Information Service) provides services based on replicated data. Each NIS server has its own replica of the password file containing a list of user's login name and encrypted passwords.

Proxy servers —

Caches may be co-located with each client or may be located in proxy server. When a new object is received at a computer, it is added to the cache store, replacing some existing objects if necessary. When an object is needed by a client process the caching service first checks the cache and supplies the object if an updated copy is available. If not, an updated copy is fetched.

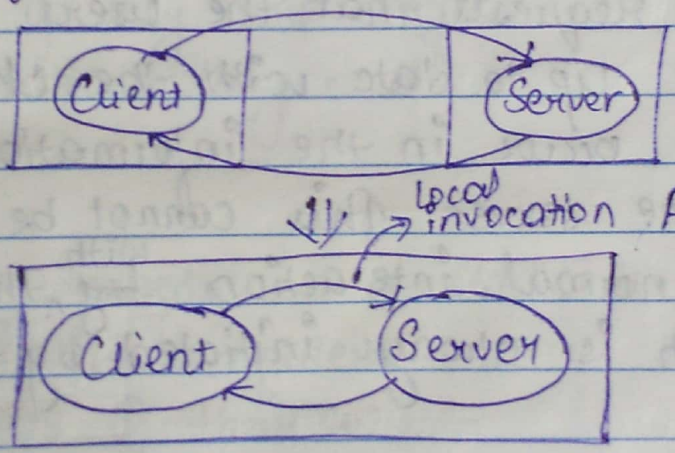
Web browsers maintain a cache of recently visited web pages and other

Date.
Page No.

web resources in the client's local file system. Using a special HTTP request, it checks with the original server that whether the cache pages are up to date or not and then only displays them.

Proxy servers increase availability and performance of service by reducing load. They may be used to access remote web servers through a firewall.

(c) Use of mobile codes



Advantage: Reduced delay

The example of mobile codes are applets. The user running a browser selects a link to an applet whose code is stored on a web server. On clicking the link, the code is downloaded to the browser and runs there. An advantage of running the downloaded code locally is that it can give good interactive response since

Fluctuations in bandwidth : jitter.

Date.
Page No.

it doesn't suffer from delays or variability of bandwidth associated with network connections.

This was initiated by the client.

Push model – Initiated by server.

Some websites use functionality not found in standard browsers and require downloading of additional code. The additional code may communicate with the server. Consider an application that requests that the users should be kept up to date with the changes as they occur in the information source in the server. This cannot be achieved by normal interaction with the web server which is always initiated by the client.

Mobile Agents : It is a running program including both code and data. It travels from one computer to another carrying out a task in someone's behalf such as collecting information and eventually returning with the results.

The advantage of this model is a reduction in communication costs and time through the replacement of remote

Date.
Page No.

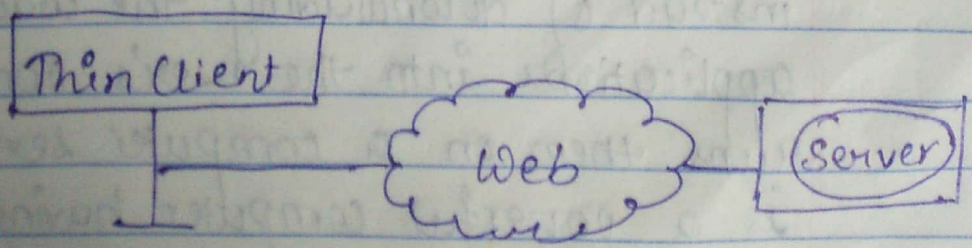
invokations with local ones.

Mobile agents just like mobile codes are a potential security threat to the resources in the computers that they visit The environment receiving a mobile agent should decide on which of the local resources it should be allowed to use based on the identity of the user on whose behalf the agent is acting.

Mobile agents can themselves be vulnerable. They may not complete their tasks if they are refused to access the information they need.

(d) Use of hardware constrained machines
— N/W computers
- No OS of their own
- Download OS and applications from a web service.

— Thin clients



E.g X-11 windows system in UNIX

UNIX - multiuser (can be used by multiple users simultaneously)

Date.
Page No.

Advantage : reduction of running cost.
Disadvantage : delay.

N/W Computers —
It downloads its OS and any application software needed by the user from a remote file server. Applications are run locally but the files are managed by a remote file server. Networked applications like web browsers can also be run using a network computer. The processor and memory capacities of a N/W computer may be constrained in order to reduce costs

Thin Clients —
It refers to a s/w layer that supports a window based UI on a computer that is local to the user while executing application programs on a remote computer. This architecture has the same low management and hardware costs as the network computer scheme. But, instead of downloading the codes of applications into the user's computer, it runs them on a computer server which is a powerful computer having the capacity to run large no. of applications

simultaneously. This computer server is typically a multiprocessor system or a cluster computer running a multiprocessor version of UNIX or Windows. The main drawback of this system is in delay experienced.

X-11 windows systems in UNIX is an example of thin client. It manages the display and interactive devices of the computer on which it runs. The clients need not be located in the same computer as the server. The server procedures are always invoked using RPC (Remote Procedure Call).

(2) Fundamental Models

(a) Interactive model
(b) Security model
(c) Failure model

Date.
Page No.

Fundar

- Descri
of dist

Associ

- Identi

- Comm
- Collec
under

3 m

(a) Inter
(b) Fail
(c) Sec

Inte

- Perfo
- Tha
of t
Pe

Date.
Page No.

Fundamental Models

- Describes the fundamental properties of distributed systems.

Associated with :

- Identification of main components
  - Processes
  - Comm. channels.
- Communication between the components
- Collective behavior of the components under certain external conditions.
  - Security
  - Failures

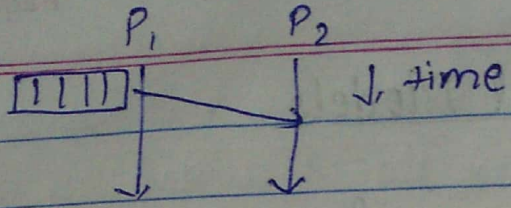
3 models —

(a) Interaction model
(b) Failure model
(c) Security model

Interaction model :

- Performance of communication channels
- Inability to have a single global notion of time.
  - Performance of comm. channel depends on — bandwidth, traffic, jitter, latency

Date.
Page No.



Latency - Time taken ~~to~~ seen by the first bit of the packet to reach from P_1 to P_2. It depends upon traffic, OS services

System architectures of a distributed system indicates that distributed systems are composed of many processes interacting in a complex way. For e.g multiple server processes or peer processes.

The behaviour of such processes along with their states can be described by a distributed algorithm. All of the activities in a distributed system is performed by interacting processes. Two significant factors affecting interacting processes are -

(i) Comm. channel performance
(ii) Impossibility of maintaining a single global notion of time.

Communication channel performance :-
Comm. over a computer network has the following performance characteristics relating to latency, bandwidth, jitter.

The delay b/w the start of a message transmission from one process and the

Date.
Page No.

beginning of its receipt by another process is referred to as latency. latency includes –

(a) The time taken by the first of a stream of bits transmitted through a network to reach its destination.

(b) Delay in accessing the network. The delay increases when latency increases.

(c) The time taken by OS comm. services at both the sending & receiving process. It varies according to the current load on OS.

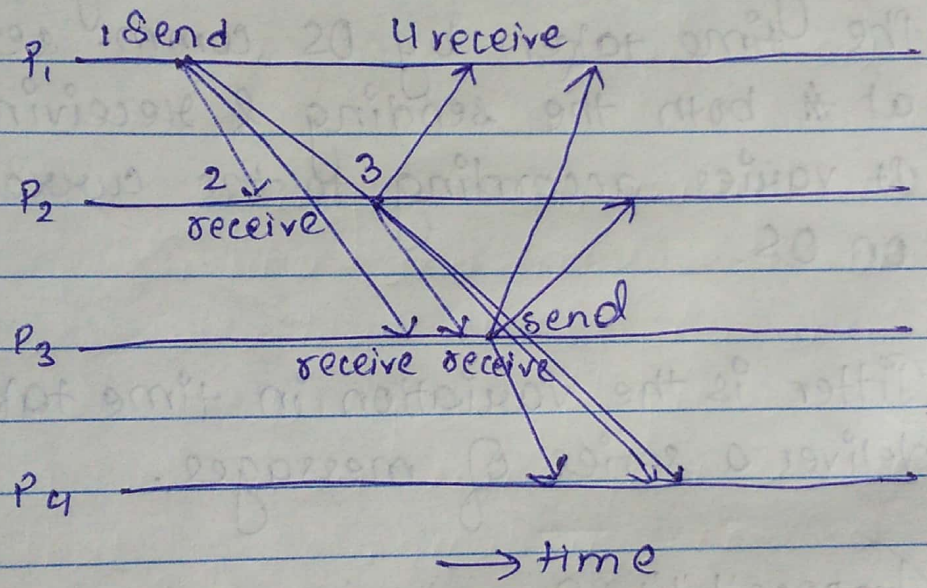
Jitter is the variation in time taken to deliver a series of messages.

Impossibility of maintaing a single notion of time :- Each computer in a distributed system has its own internal clock. Therefore, 2 processes running on different computers can associate time stamps with their events. Even if 2 processes read their clocks at the same time, their local clocks are going to supply different times. This is because comp. clocks drift from perfect time and their drift rates differ from one another. Clock drift rate

refers to the relative amount that a comp. clock differs from a perfect reference clock.

Interaction model

Synchronous Asynchronous



Failure model :

- Failure of processes
- Failure of communication channel

(i) Omission failure → Process, Comm. channel } Async.
(ii) Arbitrary failure
(iii) Timing failure → Synchronous system

Class of failure	Affects	Description.
Clock	Process	Process's local clock exceeds bounds on its rate of drift from real time.
Performance	Process	Process exceeds bounds on the interval b/w two steps.
Performance	Channel	Message transmission takes longer than usual stated bound.

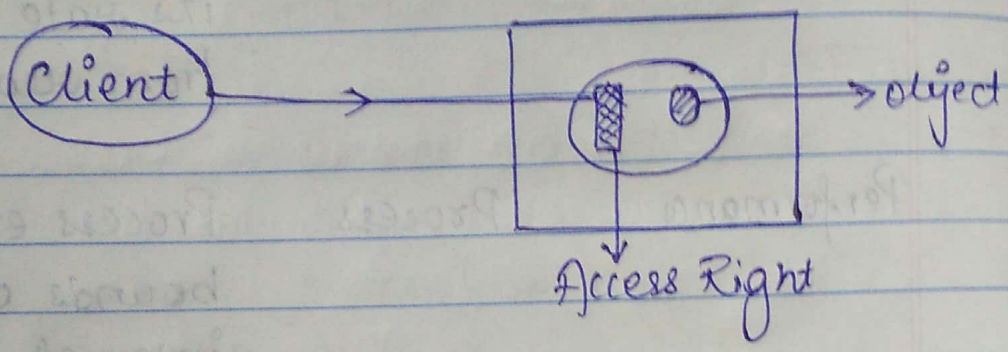
Security Model:

→ Access rights
(i) Threats to process
(ii) Threats to comm. channel
(iii) Denial of service.

Processes interact by sending messages. Due to the fact the network and comm. services used by the processes is open, the messages are exposed to attack.

Date.
Page No.

Server and peer processes again expose their interfaces enabling invocations to be sent to them by an process.



The enemy or adversary is capable of sending any message to any process and reading or copying any message between a pair of processes. The attack may come from a computer that is legitimately connected to a computer or from the one that is connected in unauthorised manner. The threats from an enemy may be -

(i) Threats to processes
(ii) Threats to comm. channels
(iii) Denial of service.

Threats to processes :
A process designed to handle incoming requests may receive a message from any other process and may not

Date.
Page No.

necessarily determine the identity of the sender. Although comm. protocols like IP does include the address of the source computer in each process, it is not difficult for the enemy to generate a message with a forged source address. This lack of reliable knowledge of the source of a message is a potential threat to correct functioning of both servers and clients.

Threats to comm. channel:
An enemy can copy, alter or inject messages as they travel across the networks. This is a threat to the privacy and integrity of information as it travels through the network. Another form of attack is to save copies of messages and replay them at a later time making it possible to reuse the message over and over again. These threats can be defeated by secure channels.

Distributed OS

Transparency → Location
Transparency → Failure
Transparency → Access

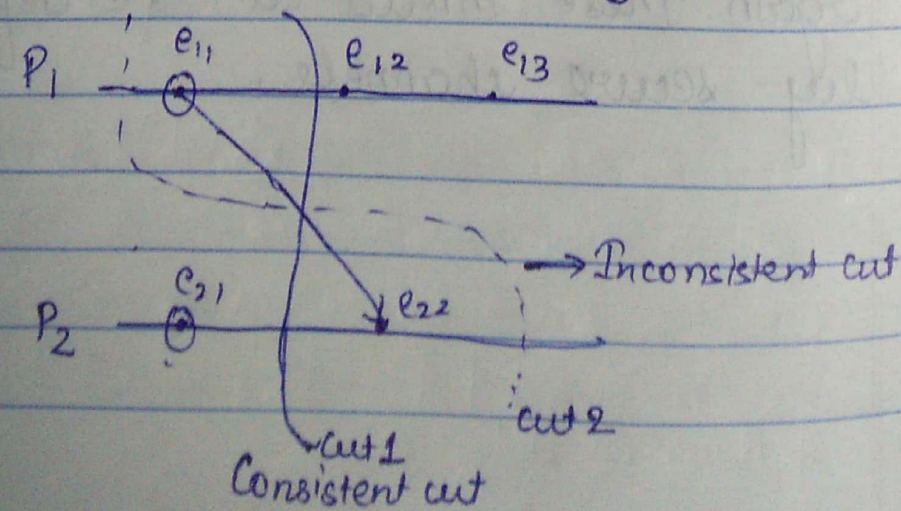
Issues in distributed OS -

(i) Global knowledge
(ii) Naming
(iii) Scalability
(iv) Compatibility
(v) Process Synchronization
(vi) Resource management
(vii) Security

Theoretical foundation of Distributed Systems

Limitations -

1. Absence of global clock
2. Absence of shared memory.



Page No.

Cut 1 → e_{11}, e_{21} Cut 2 → e_{21}, e_{22}

Lamport's logical clock :

- Only when event occurs.
- Happened before relationship.

$a \rightarrow b$

(a happened before b)

- Causal ordering of events

Condition 1 If $a \rightarrow b$ in a process P_i,

$C_i(a) < C_i(b)$

Condition 2 If 'a' is a sending event in P_i and 'b' is corresponding receiving event in P_j,

$C_i(a) < C_j(b)$

If $a \rightarrow b$ in process P_i

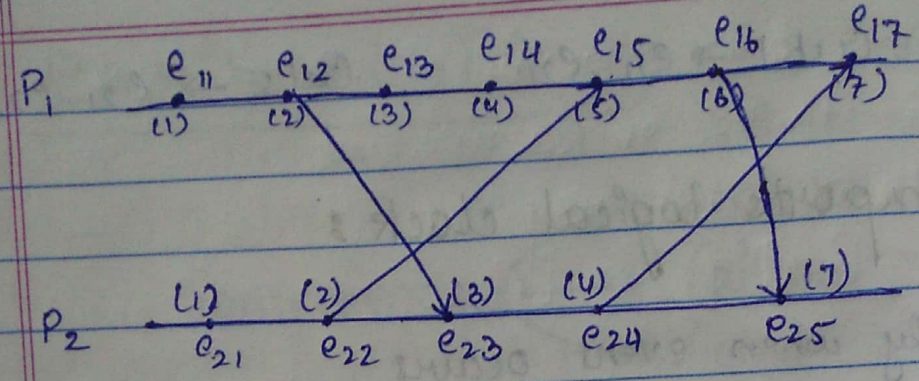
IR1 : $C_i(b) = C_i(a) + 1$;

IR2 : $C_i(b) = \max(C_i(b), t_{m+1})$;

(time stamp for message passing)

$e_{11} // e_{21}$ → Concurrent processes (no relationship)

Date.
Page No.



$$C_{(e_{25})} = \max(C_{(e_{25})}, t_m + 1)$$

$$= \max(5, 7)$$

$$= 7$$

Limitation:

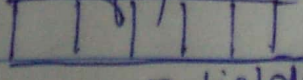
- Absolute ordering of events not possible.

$P_i \prec P_j$ if $i < j$
↓
happened before

Virtual time is an approximation of global and physical time. It advances along progression of events and is therefore discrete. If no event occurs in the system, virtual time stops.

Vector Clocks

n = no. of processes

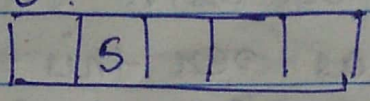


n fields for each process

Date. _____
Page No. _____

$P_i \longrightarrow C_i[i]$

If n = 5 :



$C_2 = 5$

$I.R_1$: $C_i[i] = C_i[i] + d$; $\quad d > 0$

$I.R_2$: $\forall k,\ C_j[k] = max\ [C_j[k], t_m[k]]$;

d = no. of events between a & b

Let 'n' be the no. of processes in a distributed system. Each process P_i is equipped with a clock C_i which is an integer vector of length 'n'. The clock C_i can be thought of as a function that assigns a vector $C_i(a)$ to any event 'a'. $C_i(a)$ is referred to as the timestamp of event 'a' at P_i. $C_i[i]$, the i^{th} entry of C_i corresponds to P_i's own logical time.

$C_i[j]$, where $j \neq i$ is the P_i's best guess of the logical time at P_j.

In other words, the j^{th} entry of C_i indicates the time of occurrence of the last event at P_j which happened before the current point of time at P_i. The implementation rule for vector clock are as follows :

Date.
Page No.

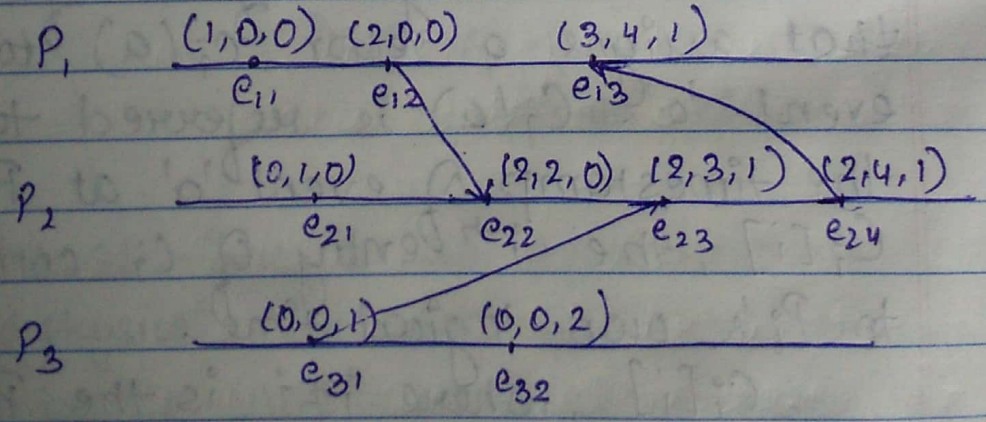
$I.R_1$ → C_i is incremented between any two successive events in process P_i as per this equation :

$$C_i[i] = C_i[i] + d$$

$I.R_2$ → If event 'a' is the sending of the message 'm' by process P_i, then message 'm' is assigned a vector time stamp $t_m = C_i(a)$

On receiving the same message 'm' by process P_j, C_j is updated as follows :

$$C_j[k] = \max(C_j[k], t_m[k])$$



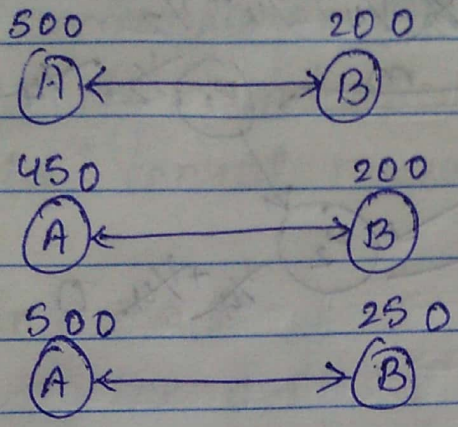
Ce_{22} : $\max((0,2,0),(2,0,0))$

$= (2,2,0)$

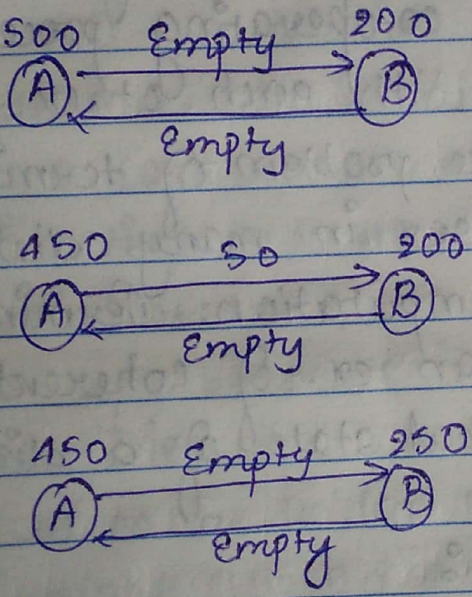
(2,3,0)
(0,0,1)

Date.
Page No.

Global state and inconsistencies :



Consistent Global State :



Termination Detection :

- Termination of a computation
- For getting a consistent global state.
- Assign weightage to each "active" process

Process → Active → $0 < w \leq 1$

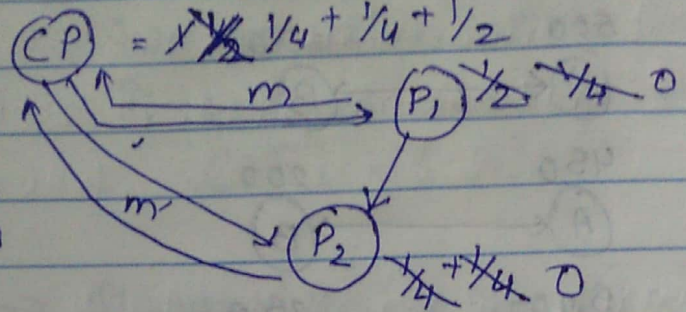
Process → Inactive → $w = 0$

$\Sigma w = 1$

Date.
Page No.

Co-ordinator Process

Initially only this is active hence w=1. As it passes messages, the weight gets distributed among processes such that $\Sigma_i w = 1$.



A distributed computation generally consists of a set of co-operating processes which communicate with each other by exchanging messages. The problem of termination detection arises in many distributed algos and computation. Termination detection is an ex. of coherent view or consistent global state of a distributed syste

System Model:
A process may be active or inactive. Only active processes can send messages. An active process can become idle at any time. On the other hand, an idle process can become active on receiving a "computati message". These messages are related to the underlying or computation being performed by co-ordinator process. A computation is said to be terminated if and only

Date.
Page No.

if all the processes are idle and there are no messages in transit. The messages sent by termination detection algo are called "control messages".

Basic idea :
One of the co-operating processes monitors computation and is called "controlling agent". Initially, all other processes are idle and controlling agent's weight = 1. Computation starts when controlling agent sends a computation message to one of the inactive processes. Any time a process sends a message, the process's weight gets splits up between itself and the receiving process. Messages carry the weight ~~for~~ the ~~re~~ to be added to the receiving process. The weight received along with a message is added to the ~~to~~existing weight of the receiving process.

The condition $0 < w \leq 1$ is true for all active processes including controlling agent and to each message in transit. The weights assigned are such that at any time, $\sum w = 1$.
On finishing computation, a process sends back its weight to the controlling agent.

Date:
Page No.

When the weight of the controlling agent is once again = 1, it concludes that the computation has terminated.

B(DW) → computation message
C(DW) → control message

Huan's termination detection algo :-

Rule 1: An active process on controlling agent has weight W may send a computation message to a process P by: $W_1 + W_2 = W$, $W_1 > 0$, $W_2 > 0$
$W = W_1$
Send $B(W_2)$ to P

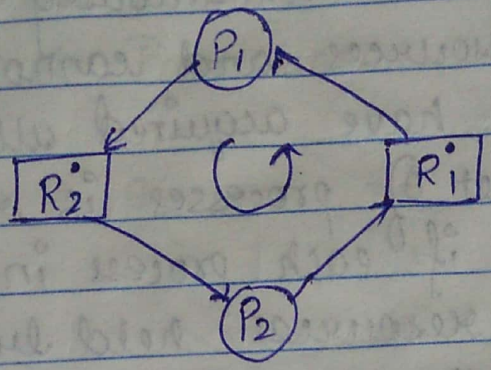
Rule 2: On receipt of B(DW), a process P with weight W
$W = W + DW$
If P is idle, it becomes active.

Rule 3: An active process with weight W on becoming idle
Send C(W) to control agent.
$W = 0$

Rule 4: On receiving C(DW) the controlling agent →
$W = W + DW$
If $W == 1$ <Terminate>

Date.
Page No.

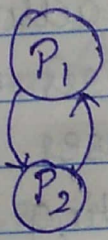
Distributed Deadlock Detection



Resource Allocation Graph

⇓

Wait-for Graph



(P_1 is waiting for P_2 and P_2 for P_1)

(i) Resource deadlock
(ii) Communication deadlock

System Model:

(i) The system uses reusable resources only.
(ii) The resources are accessed in mutually exclusive fashion.
(iii) There ~~are~~ is only one instance of diff. resources.

States → Running/Active → all the resources are present
→ Idle → waiting for necessary resources to begin execution

Resource Deadlock :

Processes can simultaneously wait for several resources and cannot proceed until they have acquired all those resources.

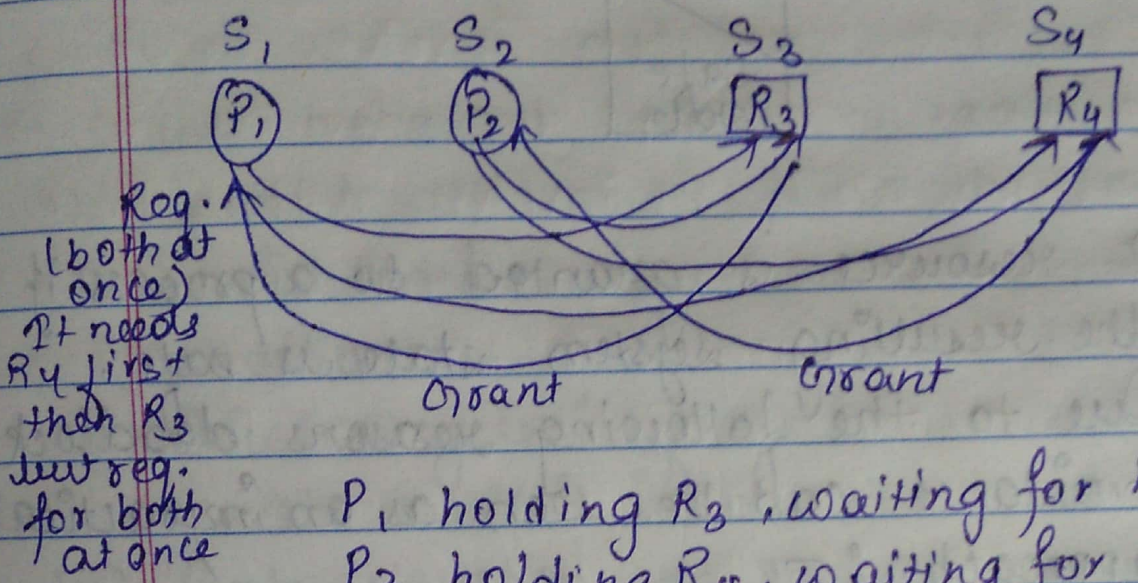
A set of processes is resource deadlocked if each process in the set requests resources held by another process in the set. It must receive all of the requested resources before it can become unblocked.

Communication Deadlock :

In this, a process waits to communicate with other processes among a set of processes. A waiting process can unblock on receiving a communication from any of these processes. A set of processes is said to be communication deadlocked if each process in the set is waiting to communicate with another process in the set and no process in the set ever initiates any further comm. until it receives the comm. for which it is waiting.

Deadlock Handling Strategies :-

(a) Deadlock Prevention



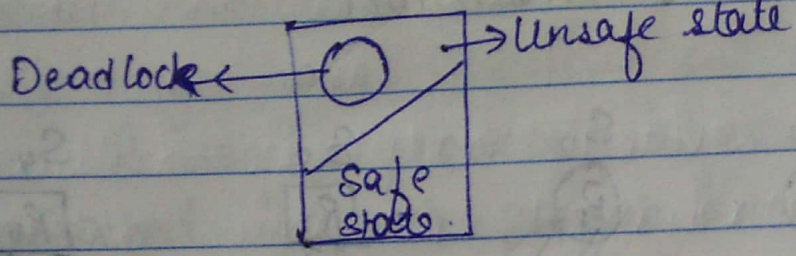
P_1 holding R_3, waiting for R_4.
P_2 holding R_4, waiting for R_3
$\Rightarrow$ Deadlock

Deadlock prevention technique is achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by pre-empting a process that holds a needed resource.

In the former method a process requests or releases a remote resource by sending a request message to the site where the resource is located. However, this method can lead to deadlock.

Date.
Page No.

(b) Deadlock Avoidance
- Always avoid unsafe state



A resource is granted to a process if the resulting system state is safe. Due to the following reasons deadlock avoidance can be seen as an impractical approach :-

(i) Every site has to maintain global state info. For this, huge storage requirements and extensive comm. costs makes deadlock avoidance an impractical approach.

(ii) The process for checking a safe global state must be mutually exclusive.

(iii) For large no. of processes and resources, it will incur very high computational cost to check for a safe state.

(c) Deadlock Detection
- Cycle in WFG (Wait for Graph)

Date.
Page No.

Must ensure —— (i) Progress → no undetected deadlocks
(ii) Safety → no 'phantom' deadlocks

This technique requires to examine if there is a cyclic weight. Deadlock detection in distributed system has 2 favourable conditions —

(i) Once a cycle is formed in a WFG, it persists until it is detected and broken.
(ii) Cycle detection can proceed concurrently with the normal activities of the system. Therefore there is no negative impact on the system throughput. As a result, in a distributed system, deadlock detection is the most seeked after method for deadlock handling.

Issues in deadlock detection and ~~removal~~ resolution:

Deadlock detection involves 2 issues —
(a) Maintenance of WFG.
(b) Search for WFG for the presence of cycles.

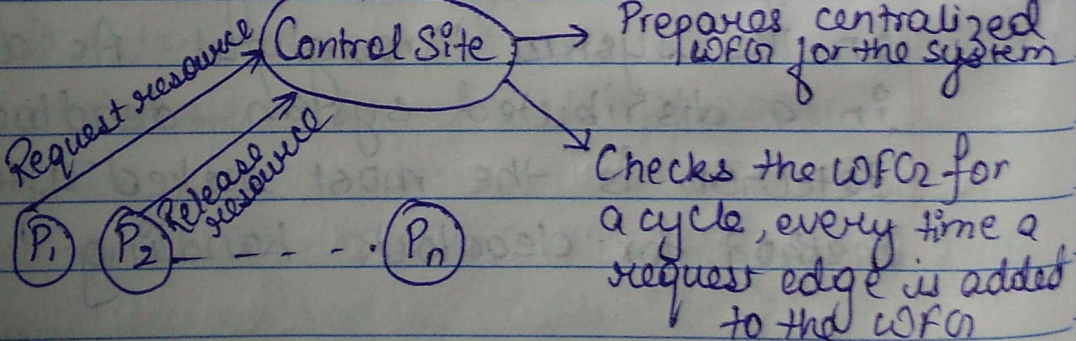
A correct deadlock detection algo must satisfy the following 2 conditions

Date.
Page No.

(i) Progress - The algo must detect all existing deadlocks in finite time
(ii) Safety - No false deadlocks to be detected.

Deadlock resolution involves breaking existing wait-for dependencies in the system WFG. It involves rolling back one or more processes that are deadlocked and assigning their resources to blocked processes in deadlock so that they can resume execution.

Centralized Deadlock Detection Algorithm:



Pros : Simple. Easy to implement
Cons : (i) Single point failure
(ii) Comm. links near the control site are congested.
(iii) Delay

Improvization

↳ Sites maintain their local info / states, periodically saves the status of control site.

In a centralized deadlock detection algo a designated control site maintains WFG of entire system and checks for cycles. All sites request and release resources by sending request resource and release resource messages to the control site. On receiving such messages the control site correspondingly updates its WFG. The control site checks the WFG for cycles whenever a request edge is added to the WFG.

Although this algo is conceptually simple and easy to implement, yet it is highly inefficient since all resource acquisition and releases must always go to the control site even for local resources. This produces large delays, large comm. overheads and congestion of comm. links near control site. Moreover, there are chances of single point failure.

As an improvisation to the algo, each site may maintain its resource

Date.
Page No.

status locally and is supposed to send the info periodically to the control site. The control site, on receiving local info from different sites, constructs the global WFG. However, due to inherent comm. delays and lack of perfectly synchronized clocks, the control site may get an inconsistent view of the system and thereby detect false deadlock.

Ho - Rammoorthy Algorithm

(i) Two phase
(ii) Single phase

Two Phase:
Periodically takes "resource status" from each site. Thereby, global WFG is constructed.

Single Phase:

Two tables → Resource
Two tables → Process

In this algo, every site maintains a status table that contains status of all processes initiated at that site. The status

Date.
Page No.

of a process includes all resources blocked and all resources being waited upon. Periodically, a designated site requests the status table from all sites to construct the WFG. If there is no cycle in the WFG, then the system is free from deadlock. Otherwise, the designated site again requests status tables from all the sites and constructs the WFG using only those transactions which are common to both the reports.

If the same cycle is detected again, the system is declared deadlocked. Although, by getting two consecutive reports, the designated site reduces the probability of getting an inconsistent view, but does not eliminate such a possibility.

Single Phase:
It requires only 1 status report from each site. However each site maintains 2 status table —

(a) Resource status table
(b) Process status table.

The resource status table at a site keeps track of the transactions that has

Date.
Page No.

locked or waiting for the resources stored at that site

The process status table keeps track of resources locked by or waited by all the ~~transac~~ processes at that site. Periodically a designated site requests both ~~all~~ the tables from every site to construct WFG.

The global WFG is constructed using only those transactions for which the entry in the resource table matches the corresponding entry in the resource table. If a cycle is formed, deadlock is declared. This algo does not detect false deadlocks since it eliminates the inconsistency in state info by using only those info which is common to both the tables. This is faster and requires fewer messages as compared to two phase algo. However, it demands more storage requirement as every site has to maintain 2 tables and exchange of bigger messages.

~~Deadlock~~

Distributed Deadlock Detection Algorithm

1. Path Pushing

Obermarch Algorithm (Initiated for distributed databases)

(i) Transmit deadlock info in the form of paths.
(ii) Designated sites called "External" or "Ex"

path [Ex → T_1 → T_2 → Ex] loop is formed but since the loop is formed using Ex, so we don't call this as a deadlock

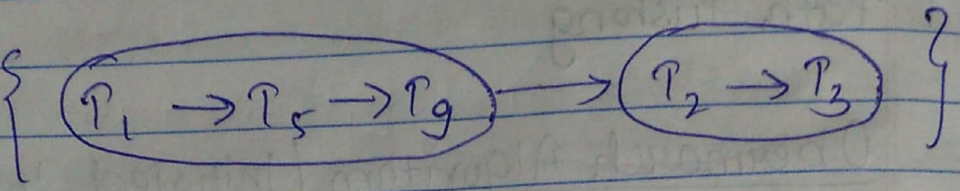
Transaction → Sub transactions
(1 transaction runs in machine A so it is not necessary all the subtransactions also run in the same machine because ~~the~~ we are in DS).

Assumption: Each transaction has utmost 1 subtransaction.

$T_1 \rightarrow T_s \rightarrow T_g$
(T_1 is dependent upon S and S is on g)
Here, WFG is termed as TWF Graph
(Transaction Wait For Graph)

Date.
Page No.

We can add info and pass it onto transaction whose lexical ordering is more than the current site.



Global info clubbed with local transformat" and sent to all the coming transactions. But this may lead to redundancy so we ensure that they are not in more ordering.

Obermarch Algorithm:

It has 2 interesting features -

(i) The non-local portion of the global TWF graph at a site is abstracted by a distinguished node called an external node, which helps in determining potential multisite deadlock without requiring a huge global TWF graph to be stored at each site.

(ii) Transactions are totally ordered which reduces no. of messages and consequently decreases deadlock detection overhead. It also ensures that exactly one transaction in each cycle detects deadlock.

Date.
Page No.

Algorithm:

1. A site waits for deadlock info from other sites
2. On receiving deadlock info, it attaches it's local info to the local TWF graph to build an updated TWF graph. It then detects cycles and breaks, only those that don't contain 'Ex'.
3. For cycles containing 'Ex', a site transmits them to all other sites, if and only if it is higher in lexical ordering than the other sites.

Drawback: Detects phantom deadlocks.
$n(n-1)/2$ msgs are to be transmitted to detect a deadlock

n = Total no. of transactions.

Message size : O(n) } In order of n
Delay : O(n) }

2. Edge Chasing Algorithm

Chandy - Misra - Haas algorithm:

Probe message is sent along the edges.

3 int (i, j, k)

- i: initiator of deadlock detection
- j: intermediate process or sender
- k: destination for which this msg was transmitted

Date:
Page No.

→ If probe msg comes back to the initiator then there will be a deadlock.

Every process p_i maintains an array

dependent$_i$[j] = true;

↓

Boolean array. | F | F | F | F | F | F | F | F |

When algo starts, all are false

P_j, P_k single site → locally depend

P_j, P_k diff. site → dependent

A process P_j is said to be dependent upon another process P_k if there exists a sequence of processes from $P_j = P_{i1}, P_{i2}, \ldots P_k$ such that each process P_k in the sequence is blocked and each process except the P_j holds a resource for which the previous in the sequence is waiting.

P_j is locally dependent upon P_k if P_k and P_j are at the same site.

Algorithm:

If P_i is locally dependent upon itself then <declare deadlock> else for all P_j and P_k such that —

(a) P_j is locally dependent upon P_j

Date. _____
Page No. _____

(b) P_j is waiting on P_k
(c) P_j and P_k are on diff. sites,

send Probe(i, j, k) to home site of P_k.
On receipt of Probe (i, j, k) the site takes following actions —
(d) P_k is blocked
(e) $dependent_k[i]$ is false
(f) P_k has not replied to all request of P_j.
Then,

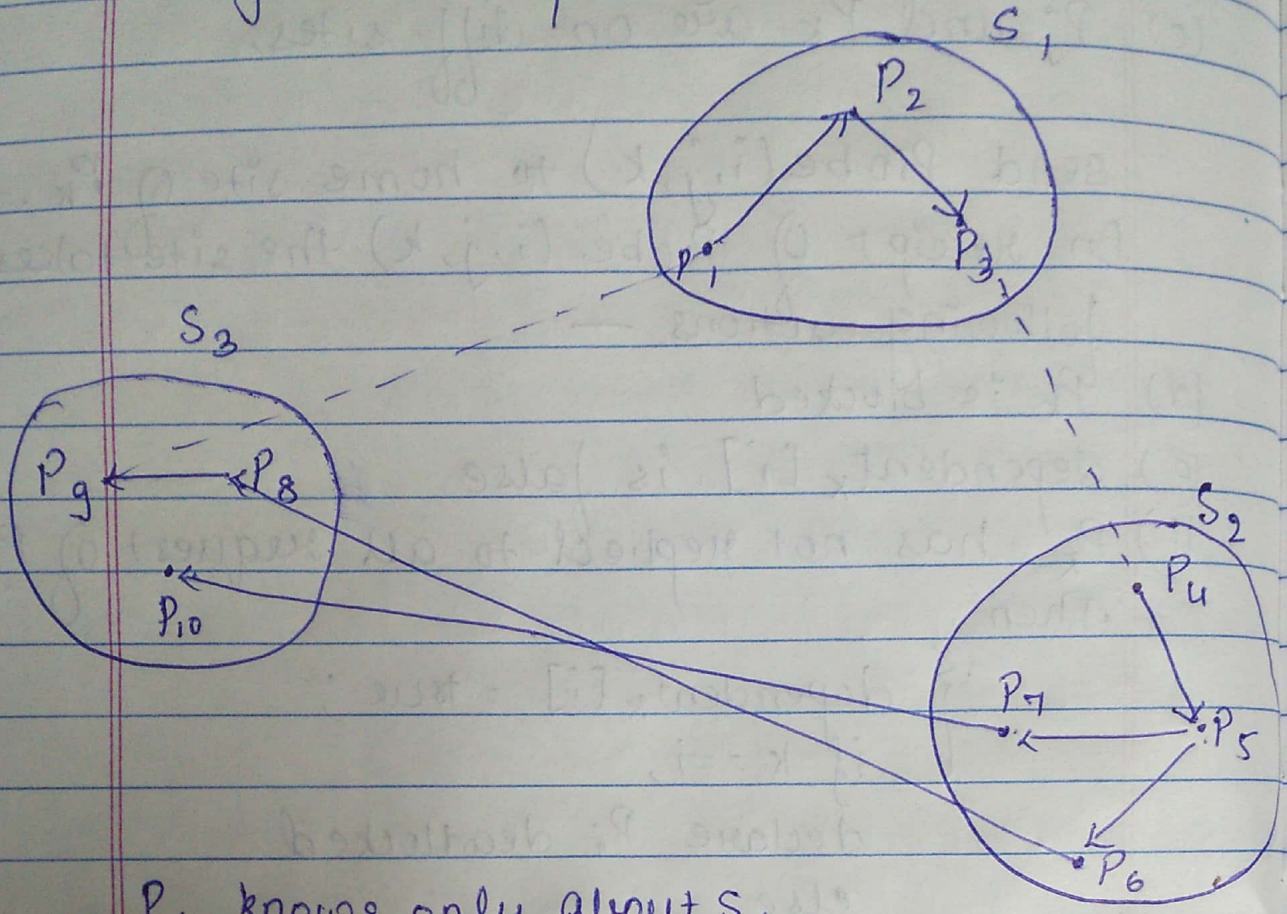
```
{ dependent_k[i] = true;
  if k == i,
  declare P_i deadlocked
  else
  for all P_m and P_n such that
```

(a') P_k is locally dependent upon P_m
(b') P_m is waiting on P_n,
(c') P_m and P_n are on diff sites
send Probe (i, m, n) to home site of P_n
(initiator remains same)

}

Page No.

Diagramatic explanation:



P_1 knows only about S_1,
so it will send probe msg to S_2
probe (1, 3, 4)
↓ ↓ ↓ to
by through

P_3 is waiting for P_4, so sends a probe message $probe_7(1, 7, 10)$ $Probe_6(1, 6, 8)$

{Probe a (1, a, 1)}

10 will not send probe msg because that is not waiting for any other process.