Distributed Systems —:

- Collection of independent computers that appear to users of the system as a single computer.
- Aspects : H/W, S/W.
- Approaches :
  - Centralized : Master/Slave  (Control w/ master)
  - Decentralized : Peer-Peer 

Centralized : Client/Server Architecture (Web srvr, Login Sys, etc)
Decentralized : Peer Networks.

- Centralized System Characteristics :
  - Global Clock
  - Single Central Unit
  - Dependendent Component Failure
  - Scaling (Vertical Scaling on Central Server possible)
  - Easy to secure

Vertical Scaling : Enhance central system computing resources
- Dedicated Resources
- Cost efficient for small systems
- Vertical Scaling Limits
- Dependent on network connectivity
- Bottlenecks upon traffic spikes.

- Decentralized System
  - Independent Nodes
  - Behaviour of system : aggregation of node behaviour.
  - No clock
  - Multiple.
  - Dependent Component Failure at low extent

- High availability
- More autonomy & control over resources
-
- No control over independent nodes

× Advantages over Centralized OS
Economical : Cheaper costs of microprocessors than mainframes.
Speed : More nodes imply more computing power.
Reliability : System operational even on failure of nodes.
Incremental Growth : System resources extendable by adding nodes.
Inherent Distribution : Some applications involve spatially-separated machines

× Adv. over PCs
Data Sharing : Allow multiple user access to a common DB.
Device Sharing : Share expensive peripherals
Communication : Support human–human communication
Flexibility : Spread workload over machines in cost effective manner.

× Disadvantages :
Software : Increased complexity, low availability, incompatibility of current software or task for distributed systems
Networking : May saturate, cause problem (due to disconnectivity)
Security : Easier access to secret data.

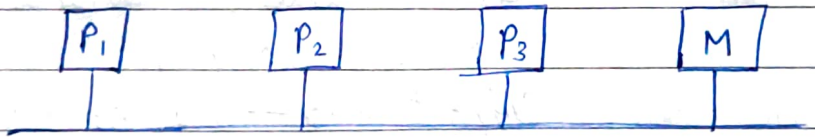
× Hardware Concepts :

- Utilize MIMD architectures (Parallel & Distributed Systems).
— PDC — Multiprocessors (Shared Memory) — Bus / Switched
Multicomputers (Private Memory) — Bus / Switched
Bus – Common Data Bus
Switched – Indep conn. b/w units.

Multiprocessors : Tightly Coupled
Multicomputers : Loosely Coupled

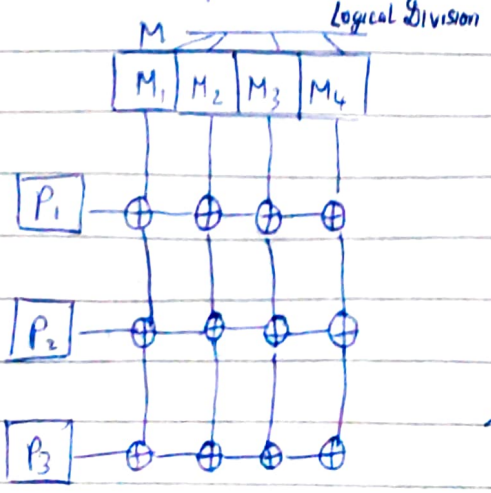
	TC	LC
Memory	Shared Memory	Distributed Memory
Data Rate	High Data Rate	Low Data Rate
Cost	High Cost	Low Cost
Memory Conflicts	Memory conflicts	Nomemory conflicts
	Low degree of interaction ⟷	High degree of interaction b/w tasks

× Bus - Based Multiprocessors :
- Overburden on Bus.

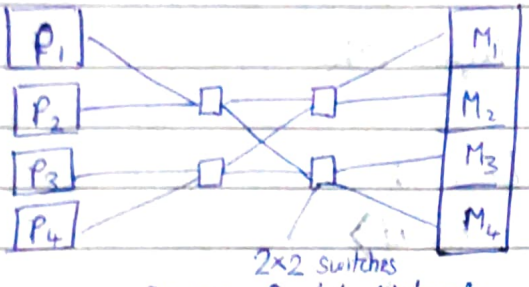


- Solution : Cache w/ processors.
- New problem : Inconsistency in same data on diff caches.
- Solution : Write - Through Cache (Immed. write on memory)
: Shoopy Cache (Monitor data of other caches to sync. data (problem : continuous monitoring))

× Switched Multiprocessors :



× Divide memory into chunks, connect memory & processors via switches to create paths b/w memory & processors.
× Switch count exponential in # of processors (or n×m)
× Used in hierarchical, NUMA sys.



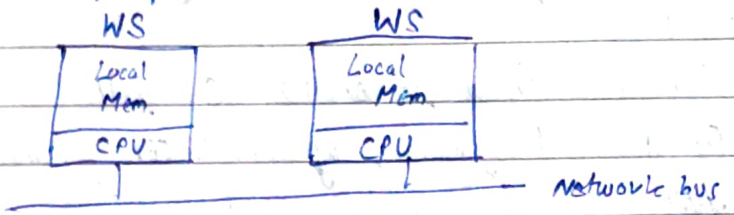
Omega Switch Network

- × Alter; reduce switch count by use of multi-level switches.
- × Switch count decided upon consideration of load balancing.

$\log_2 n$ layers. $n/2$ switches per layer.

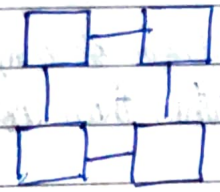
$\therefore \frac{n \log_2 n}{2}$ switches for n procs.

× Bus-Based Multicomputers:

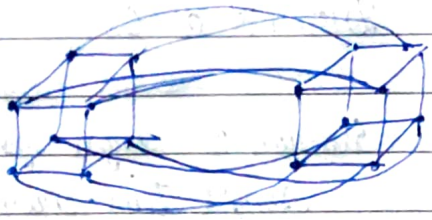


- Failure of bus ⇒ failure of system.

× Switched Multicomputers :



Grid-Based



Hypercube (4)

- NUMA-based arch.

× Software Concepts — :

- Sharing of peripherals. : Loosely Coupled (S/W) (No sync.)
- Execution of programmes in parallel. : Tightly Coupled (S/W) (Sync.)

× 8 H/W & S/W concept combinations possible.

Eg: Bus Based Multiprocessor w/ Loose coupled S/W.

Network OS —: (Loosely Coupled H/W)

× Workstations connected with LAN :
rlogin <machine id>
// ssh <machine id>

× Copy files across machines :
rcp <machine 1> <machine 2>:<file 2>
// scp <machine> <src> <dest>

- Global, shared file system accessibility.
- Concerns :
  - Require machines to share OS ? No
  - On different systems ; agreement on format & meaning of messages potentially exchanged required.
  - High degree of autonomy.

Distributed OS — (Tightly Coupled S/W)

× Illusion of single system : entire network of computers is a single time-sharing system rather than collection of machines.
× Single, global IPC mechanism.
× Same mechanism across machines.
× Global protection mechanism required.
× Same process mgmt. system across machines.
× Same filesystem required for compatibility & efficiency.

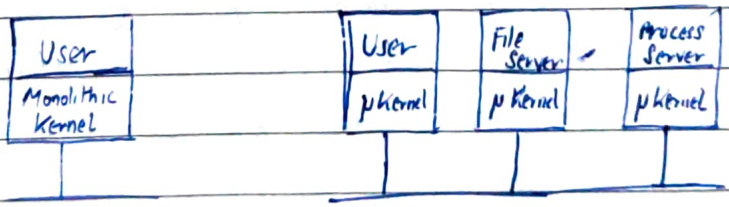
Network OS : LC H/W & LC S/W
Distributed OS : LC H/W & TC S/W
Parallel OS : TC H/W & TC S/W (Multiproc. Time Sharing OS)

	NOS	DOS	MPOS	
× Virt. Uniproc.	N	Y	Y	
× Same OS.	N	Y	Y	
× # Copies of OS	M	M	1	(M systems/proc.)
× Comm.	Shared Files.	Msg.	Shm.	
× Agreed Protocols (NW) Reqd?	Y	Y	N	
× Single Run Queue	N	N	Y	
× Well defined file sharing semantics?	N	Y	Y	

Distributed OS - Design Issues -

× Transparency : Achieve single-system image (user unaware of working).
- Location Transparency : Users unable to state location of resources.
- Replication Transparency : Users unable to tell # of existing copies
- Migration Transparency : Resources moveable w/o name change on user end.
- Concurrency Transparency : Multiple users may share resources automatically.
- Parallelism Transparency : Task execution in parallel w/o user knowledge.

× Flexibility :



Monolithic (High Complexity, Low flexibility)

Microkernel (Divided Components, High Flexibility)

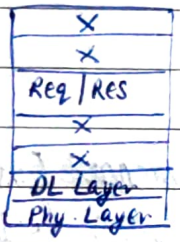
× - Reliability : Multiple components reduce chances of failure.
× - Availability : Influences reliability
× - Security : Risk of infection of all nodes from single node
× Performance : Optimize response time (↓), throughput (↑)
× Scalability : Ability to expand system resources (considering other factors & preserve performance.

Potential Bottlenecks —:

- Centralized Components (Decrease throughput & limit users)
- Centralized Database (Decrease throughput)
- Centralized Algorithms (Offline vs Online)

Comm. in DS —: OSI Model —:

- Nodes follow client-server architecture.
- Modified layer arch : Req/Res protocol



Nodes over same protocol & OS.
(Port, Addr, etc on H/W layer)

- Comm. services reducible to two syscalls :
  - send (dest, & msg) : send data message
  - receive (addr, & msg) : receive message

message components :
source, dest, opcode (operation code),
count (# bytes), offset (start offset),
extras (extra1, extra2), result, name (path of file),
data (for I/O).

- Issues ;
  - Addressing : Identification of process to deliver message
  - Solution - Two-part names (process & machine #)
    - Broadcasting to announce addresses (LAN)
      (Problem : Non-scalable, increases traffic)
    - DNS based resolver system (Name server for mapping high-level names to addr.).

- Blocking : On request send, block until completion. (similarly for receiver).
  Alter, nonblocking approach, return control to caller immed. before completion & resume execution.
- Reliability : Require acknowledgement for assuring message delivery.
  : Approaches : Send ACK for every response & request.
  : Use response as ACK for request.
- Buffering : Dispatch msgs to processes directly or cache msgs in kernel for increasing reliability, or use mailboxes.

Addressing : Via Machine #, Sparse Proc. Addr. or ASCII name
Blocking : Via Blocking Primitives, Non-blocking w/ copy to kernel or non-blocking w/ interrupt
Buffering : Unbuffered (discard extra), Unbuffered or Buffered (Mailbox)
Reliability : Unreliable, or Request-Ack-Reply-Ack or Req.-Reply-Ack

Distributed Clock Synchronization —:

+ Synchronization : Adjustment of clocks to show same time as another.
  : Possible in centralized & distributed systems.
+ Centralized System : Shared memory, common clock, clear event ordering (events timed w/ same clock). (sync. is easy).
+ Decentralized System : No shared mem, no common clock (multiple sprite clocks) (sync. is hard)
+ Requirement for sync : send & recv. data in sync. manner.
+ Clock Synchronization :
  - Sync. time of all systems in dist. environment.
  - Challenges :
    - Clock Skew (delay introduced in clocks)
  - Properties of dist sys :
    - Relevant info. scattered across machines
    - Processes make decisions based on local info.

- Single pt. of failure must be avoided.
- No common clock or precise time src. exists (global level).

2 => Collection of information in single location for processing is undesirable

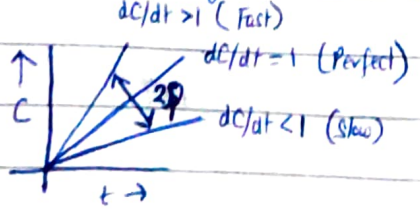
3 => Sys. should not go down on failure of single sys.

4 -> P_A & P_B request time => $T_{P_B} > T_{P_B}$

- Issues due to no sync. : make software fails to detect file updates of files across systems.
  - Challenge : on local clock, events occurring after another may nevertheless be assigned an earlier time.
- Types :

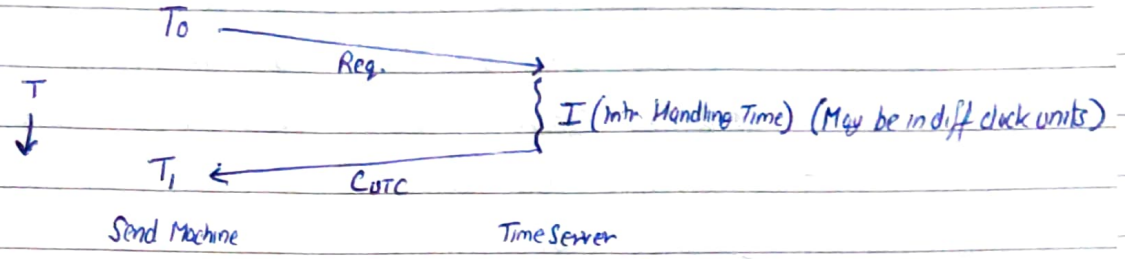
× Physical Clock Sync. :

- Systems consist of phy. clocks used to timestamp system events.
- Techniques : UTC (Universal Coordinated Time)
  Cristian's Algorithm
  Berkeley's Algorithm
- Terminology :
  H : Timer causing interrupt H times / second.
  C : value of clock
  t : UTC time
  $C_p(t)$: value of clock at machine p.
- Ideal Case : $C_p(t) = t \ \forall p$, i.e. $dC/dt = 1$
  ρ : deviation in time derivative. (max. drift rate)
- Allowed range : $1-\rho \leq \frac{dC}{dt} \leq 1+\rho$ (Permissible error)



- Cristian's Algorithm :
  - Issue RPC to time server & issue time.
  - Send request to time server in max $\delta/2\rho$ secs. ($\delta < 2\rho$)

- Time server replies w/ current UTC.
- Machine measures time diff. of sending & receiving reply from Time server & uses this difference for finding new time.



- Best estimate for message propogation time : $(T_1 - T_0)/2$.

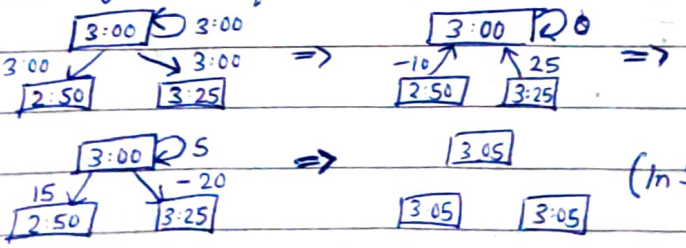
New time = $T_{NEW} = T_{SERVER} + \frac{(T_1 - T_0)}{2}$
(I)

T_{SERVER} = Time replied by UTC server.

- Problems : $C_{UTC} < C_p(t)$ (Faster clock at machine end)
  Solution : Gradual time changes instead of immediate.
  : I (Propogation time) is non-zero. (for reply reception)
  Solution : Measure propogation at sender end by estimate.

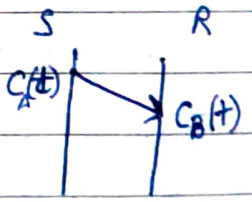
- Berkeley's Algorithm :
  - TS asks machine their clock time as well as notifies self time to machines. (In periodic fashion).
  - Machines reply to server w/ diff. in time w/ TS.
  - Server computes average time of clocks including itself.
  - TS shared adj. value of time w/ all machines, including itself.

Eg :



- Delay of propogation accounted by using S (wait time) for considering updates & receiving time values.

× Logical Clock Sync. :
- Focus over order of occurrence, instead of time of occurrence. (for non-interacting machines, sync is not important).
- Time can only move forward.
- Logical time based on order of events, obtained via intr to update clock. Problem: More intr => higher overhead.
- Common algorithm: Lamport's Algorithm.
- Lamport's Algorithm —:
  - Follows the happened-before concept.
    $A \rightarrow B$: A before B
    Logical conclusion => $T(A) < T(B)$, $C_A(T) < C_B(T)$
  - Concurrent Events: Events across different processes not in comm. where nothing may be said about which event occurred first.
  - Each machine receives msgs, carrying timestamp of sender.
  - On message arrival :—:
    If recvr's clk < msg timestamp
    set sys clk = msg timestamp + 1
    else do nothing



  - Clock always advanced, never rewinded, to preserve relative ordering of events.
  - Sender :
    Time = Time + 1
    Timestamp = Time
    send (msg, timestamp)
    Receiver :
    (msg, timestamp) = recv()
    Time = max(Time, timestamp) + 1
  - Fractional timestamps maintained for preventing concurrent events on systems where such events are not allowed.
    $a \rightarrow b \Rightarrow C_a(t) < C_b(t)$

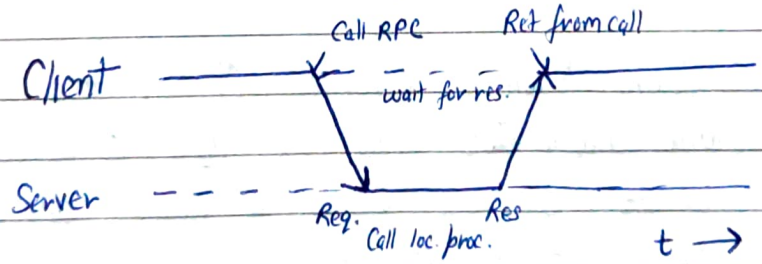
× Synchronized Clocks —
- Atmost once msg. delivery. (Msg. w/ same timestamp or lower are ignored).

- G = CT (Current) − Max Lifetime (TTL) − Max ClockSkew (Permissible Limit) (Delay)

- Using G, determine if packet is to be kept or discarded.
  Msg. w/ max G is ignored from received msgs.

x Remote Procedure Calls —:

- Executing procedure / method located in remote location.
- Conventional procedure call: Stack frames w/ local vars & ret. addr pushed to stack, & removed when completion.
- Caller executes fn. / procedure, & is suspended until completion & result return.
- Remote procedure call: Caller process executes remote proc. & is suspended until fn completes & returns.
- Parameters passed to machine where procedure is executed.
- Communication via IPC (message passing). (Loosely coupled machines)



- Parameter passing to RPC via stubs: client & server side.
  - Takes care of packing / unpacking of params.
  - Message passing is also handled.
  - Packed data by client unpacked at server.
  - Param. marshalling: Param. packing by client.
  - Client Stubs: Builds msg w/ params for local OS for execution
  - Server Stubs: Unpack params., for execution, send result to client.
  - Client - Pack ①, Server - Unpack ②, Server - Pack ③, Client - Unpack ④

- Variants: synchronous and asynchronous.

- RPC execution steps :
  - Client invokes client stub in normal way.
  - Client stub builds message, calls local OS. (traps to kernel)
  - Kernel sends message to remote kernel.
  - Remote kernel dispatches message to server stub
  - Server stub unpacks params., calls server.
  - Server performs tasks, returns result to stub.
  - Server stub packs result in message, traps to kernel.
  - Remote kernel sends message to client kernel.
  - Client kernel returns recvd msg. to client stub.
  - Stub unpacks result, returns to client.

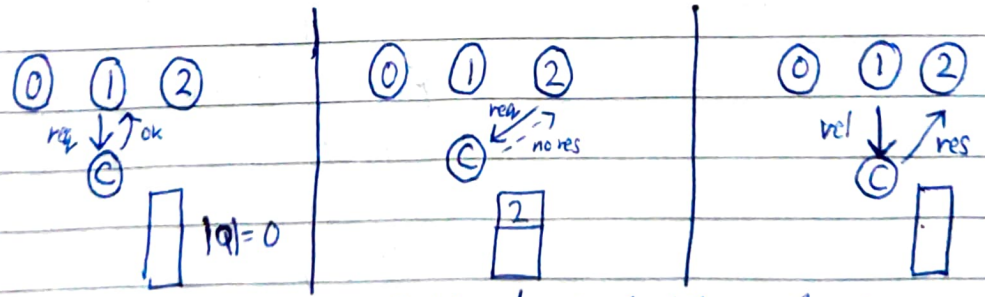
- Issues :
  - Data Representation : Formats & protocols
  - RPC call failure / duplication :
  - Binding : Client unaware of server for procedure call.

- Data Representation in Parameter Passing —:
  1. Value parameters may be in different formats (big-endian, etc)
  2. Call by reference / pointer not possible w/o copying data (addr. of remote machine unavailable).
  - Soln (1) : Client & Server must agree on : message fmt, fmt of complex data structures & transport protocol. Use external data representation (universal) for data representation (param marshalling).
- Call Failure / Duplication —:
  - Messages should be acted on only once (instead of atmost once).
  - For ensuring only one invocation, use concept of ACK.
  - Client sends req. to server, server ACK client to prevent client to sent multiple requests. (Server responds exactly once, client sends until ACK) (Server ignores successive msgs.)

- Binding client & server via port # :
  1 - Either at fixed port addresses.
  2 - Or via a match making technique.
  1) Server assigns fixed port address (unchanged) for service, & announces for use by client. (Leads to rigidity in port #).
  2) Matchmaker approach : server allocs port to service (any random). Client requests matchmaker for resolving port for reqd service & send to server for execution.

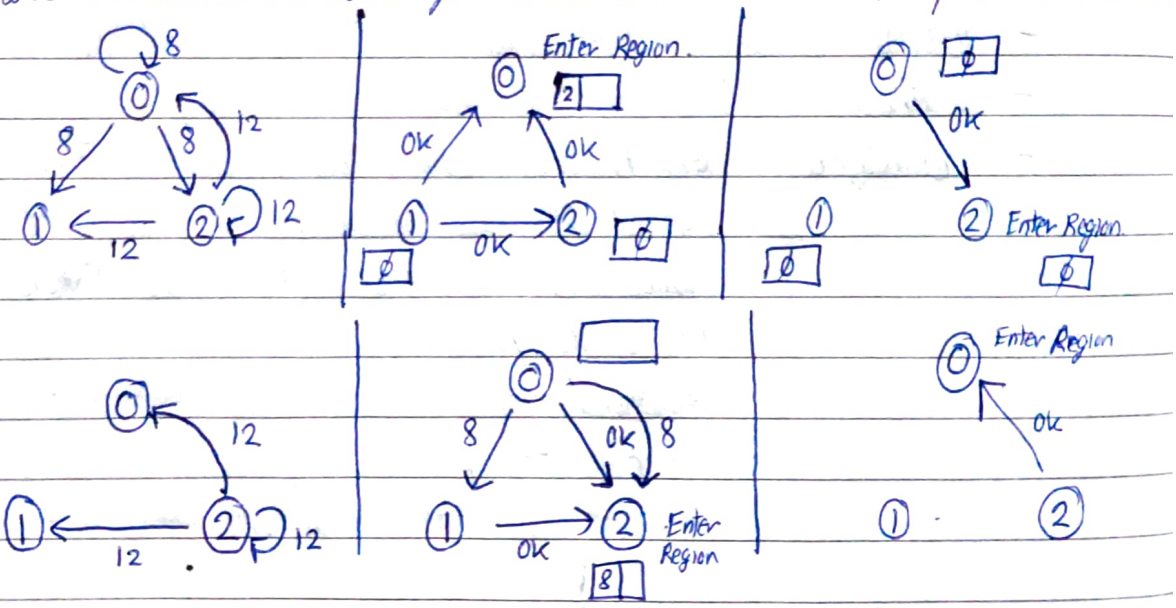
× Mutual Exclusion —:

- Provides a control over critical sections for cooperating processes.
- Methods : Centralized, Distributive, Token Ring.
- Centralized : One process / machine acts as coordinator.
  : Request for critical section sent to coordinator, which queues request (FIFO) for dispatch.
  : On single process request w/ empty queue, grant access, otherwise make process wait until resource freed. (no reply to requesting process, requesting process goes to wait & requests status periodically) until coordinator sends response).



  : Issues : Single point of failure (coordinator)
  : Performance bottleneck.
- Distributive : Workson principle of ordering of events & packages.
  : Build messages w/ critical section name, PID, current time
  : All systems actively part in algorithm, utilize reliable communication.

- When process requires access to critical region, broadcast declaration
- If receiver, processes not willing to access region sends OK.
- If receiver process in critical region, request queued to notify resource being freed.
- If receiver process wants to enter critical region but has not yet done so, compares message timestamp with self broadcasted message:
  - If time of broadcast message is lower, request queued for future notification
  - If time is higher, receiver sends OK to allow process to proceed.
- Process enters critical region once # OKs = # peer nodes.



- : Issues : Excessive transfer of data.
  : Dependency on multiple systems for acknowledgement. (Single point failure → n point failure)
  : All processes involved in decision making
  : Group membership maintainance.
  : Solutions : OK & DENY messages.
  : Time based waiting to prevent waiting for dead processes
  : Wait for majority or specific processes.

- Token Ring : Assign tokens to grant privilege for critical section.
: Pass token to successive nodes for use when work completed.
: Excessive passing may be unreliable (token may be lost) or may waste resources.
: Issues : Dependency on token holder.
: Multiple point failures. (successive node, token holding node)

	Msgs per entry/exit	Delay before entry (# msgs)	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