Sandhya
Date
Page

* Parallel and Distributed Computing:-

In parallel Computing, all processors may have accen to a share memory to exchange information between processors.

In distributed Computing, each procenor has it's own private memory, where information is enchanged by passing message between processors (RPC, RMI).

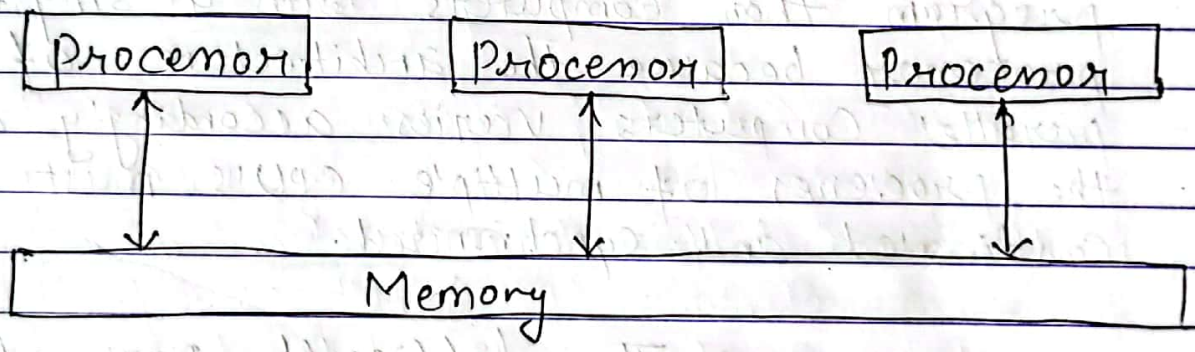


Fig: Parallel Computing

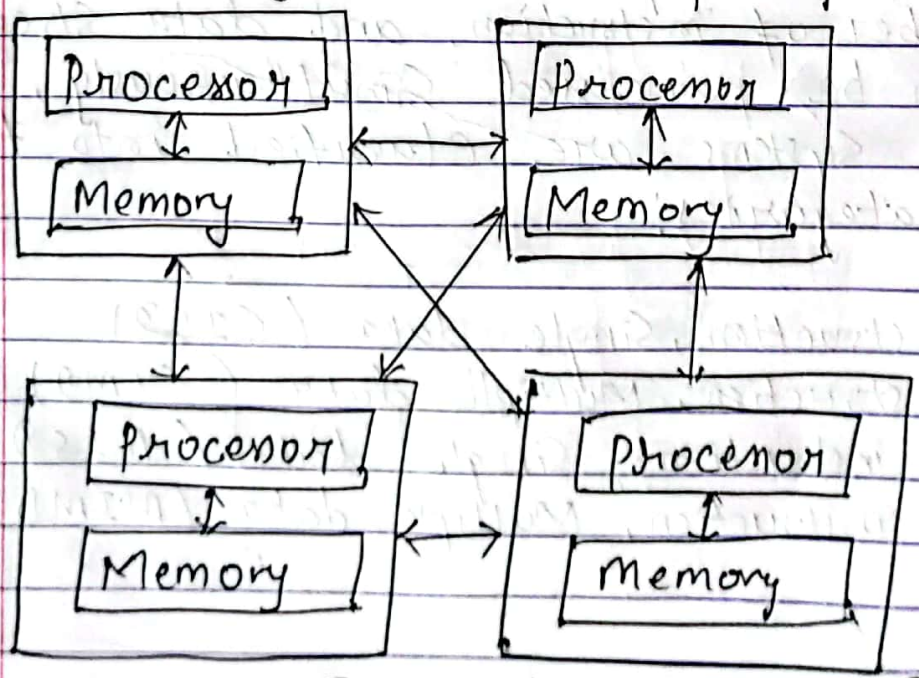


Fig: Distributed Computing

Why do we need parallel Computing?

- Serial Computing is too slow.
- Need for more Computing power (Eg. Data Mining, Search engine, cryptography).

* Parallel Procesing paradigms: Flynn's Taxonomy

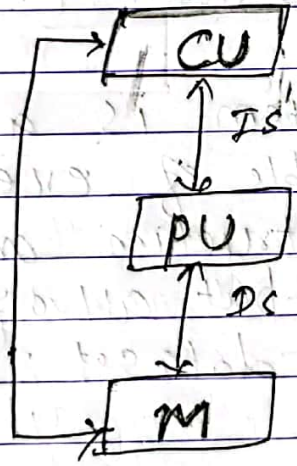
Parallel Systems are more difficult to program than Computers with a single protenor because the architecture of parallel Computers varies accordingly and the procenes of multiple CPU's must be coordinated and synchronized.

The difficult part of parallel procesing are CPUs. Based on the number of instruction and data streams that can be processed simultaneously, computing systems are classified into four major categories:

(i) Single instruction, Single data (SISD)
(ii) Single instruction, Multiple data (SIMD)
(iii) Multiple instruction, Single data (MISD)
(iv) Multiple instruction, Multiple data (MIMD)

(i) Single Instruction, Single data (SISD):

- An SISD Computing System is a uniprocesor machine which is capable of executing a Single Instruction operating on a Single data Stream. In SISD, machine, instructions are processed in a sequential manner and Computers adopting this model are popularly called Sequential Computers. All the instructions and data to be processed have to be stored in primary memory.

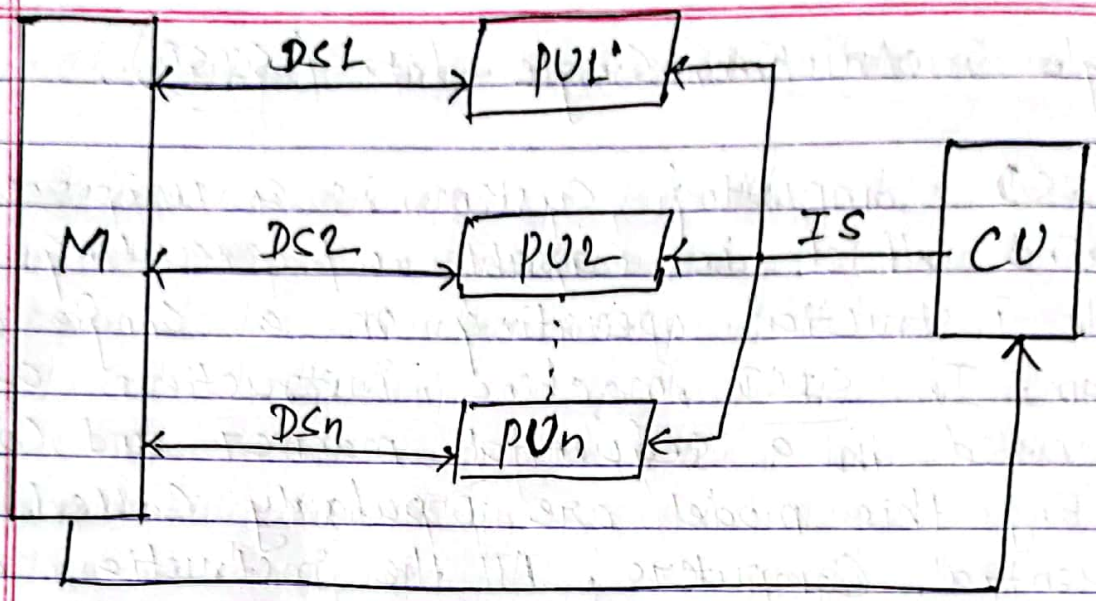


Where,

M = Memory
CU = Control Unit
PU = Procesing unit
IS = Instruction ~~unit~~ Strea
Ds = Data Stream

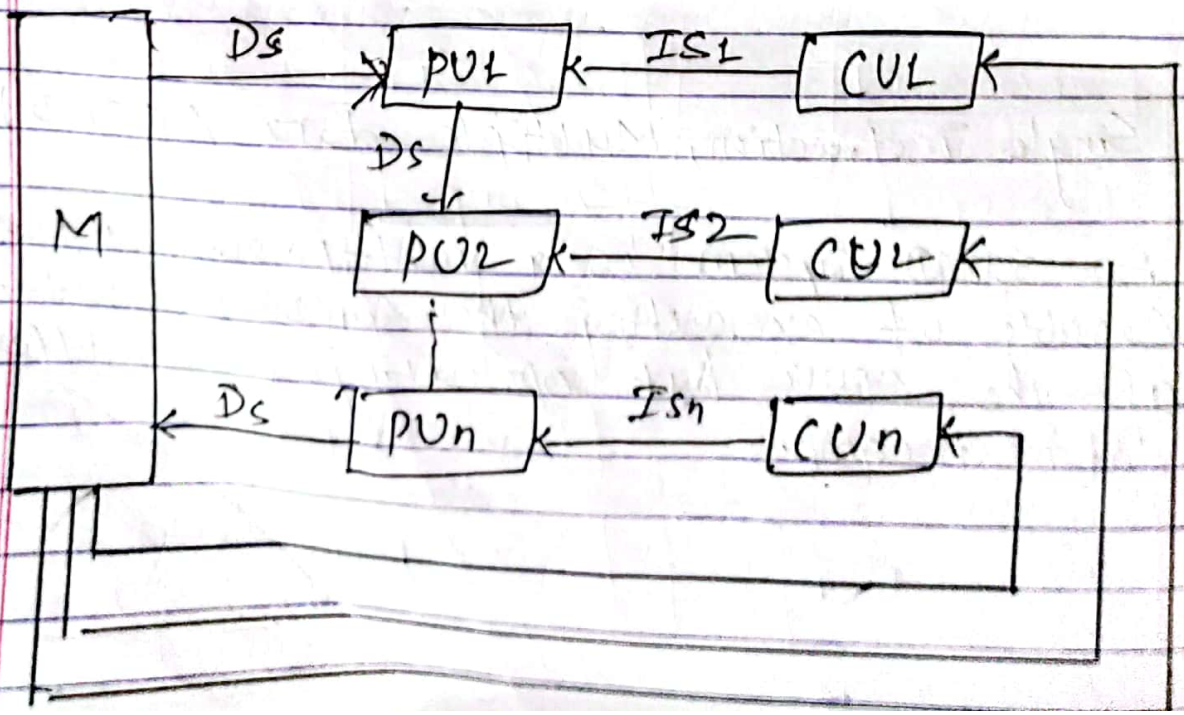
(ii) Single Instruction, Multiple data (SIMD):

- An SIMD System is a multiprocesor machine capable of executing the same Instruction on all the CPU's but operating on different data streams.



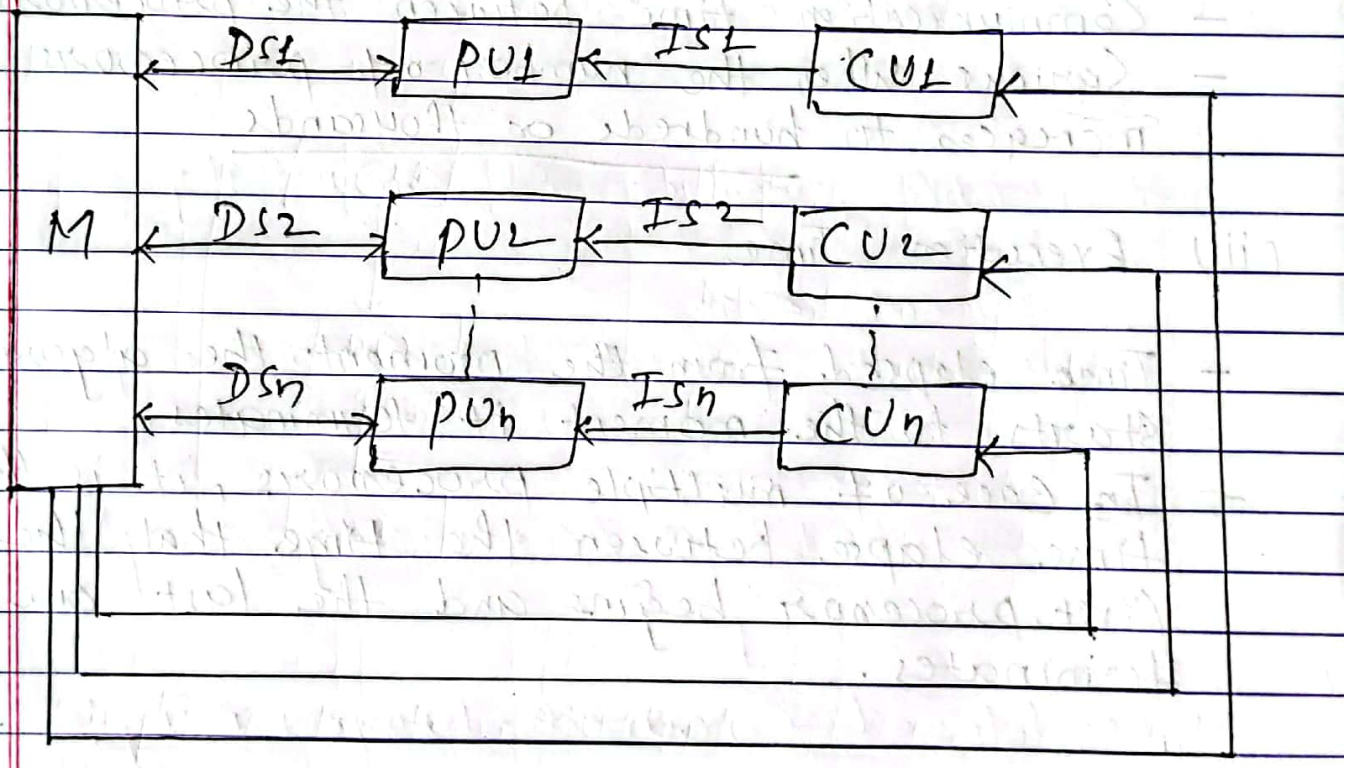
(iii) Multiple Instruction, Single data (MISD):

– An MISD Computing system is a multi-processor machine capable of executing multiple or different instruction on different procening elements (PEs) but all of them operating on the same data set.



(iv) Multiple Instruction, Multiple data (MIMD):

- An MIMD System is a multiprocenor machine which is capable of executing multiple instructions on multiple data sets. Each procesing elements in the MIMD model has separate instruction and data streams and work asynchronously.



* Issues related to parallelization that do not arise in sequential programming:

(i) Task Allocation
(ii) Communication Time
(iii) Execution Time
(iv) Speedup
(v) Communication Penalty

(i) Task Allocation (co-operative & independe

- Proper sequencing of the tasks when so of them are dependent and cannot be executed simultaneously
- Load balancing or Scheduling

(ii) Communication Time

- Communication time between the procenors
- Serious when the number of procenors increases to hundreds or thousands

(iii) Execution Time

- Time elapsed from the moment the algorith starts to the moment it terminates.
- In case of multiple procenors, it is the time elapse between the time that the first procenor begins and the last one terminates.

(iv) Speedup

- S = Running time of the best available sequential algorithm / Running time of the parallel algorithm
- It is not always possible to decompose the task.

- So maximum speed of N processor system in executing algorithm is,

$$S_N \leq \frac{1}{f + \frac{(1-f)}{N}} \leq \frac{1}{f}$$ Where,

Where, $f \rightarrow$ fraction of computation that must be done sequentially

$S_{max} = 1$, where $f = 1$ (no speedup that is every task must be done sequentially)

$S_{max} = 0$, where $f = 0$ (full speedup that is all computations must be parallel)

(V) Communication Penalty:

$$CP_i = \frac{E_i}{C_i}$$

Where,

E_i = total execution time spent by P_i

C_i = total time used for Communication by P_i

Sandhya
Date
Page

* Semantics of Concurrent Programming:-

- Concurent programming refers activation of more than one instructions at a time.

- In programming language theory Semantics is the field of concerned with the mathematical study of the meaning of the programming language.

E.g:
I eat rice.
(Syntactically Comect and Semantically Comect

I eat car.
(Syntactically Comect but Semantically wron

E.g: int a, a=2;
int b, b=3.1 (Semantically wrong)

* Models of Concurent Programming:

(i) Lewel of Granularity:

- Consider two agents (eg. bank teller) are accening a centralized database simultan ously

- Observe these activities at (deposit=100)

- Transaction level → activities are concurrent
- CPU cycle level → activities are sequential

(ii) Sharing the clock:

- When several processes operate concurrently, it is important to determine whether they share the same clock. e.g. P1, P2 is dependent on P1

(iii) Sharing the memory:

- Accen to share memory may have to be mutually exclusive.

(iv) Pattern of interaction:

(a) Synchronization (defines a chronological order between events)

- Mutual Exclusion

- It is defined by an encapsulated sequence of actions whose execution is indivisible; once a process starts executing this sequence, it may not be interrupted or the sequence is completed.

- Mutual Admission

- It is defined by an encapsulated sequence

of actions which must be executed by two processes simultaneously if one pr is ready and the other is not; it must wait until both are ready to give the undivided attention. E.g. Message passing

(b) Communication (defines a transfer of information)

- Synchronous Communication

- All parties involved in the Communicati are present at the same time. E.g. pho Calling.

- Asynchronous Communication

- doesn't require that all parties involved the Communication to be present at the same time. E.g. E-mail

* Semantic definitions:

- A programming Language is defined by two features it's Syntax; which defines the set of legal sentences in the language and it's Semantics; which assigns meaning to legal sentences of the language.

Sandhya
Date
Page

Programming Language Semantics:

(1) Axiomatic Semantic Definition
(2) Operational Semantic Definition
(3) Denotational Semantic Definition

(1) Axiomatic Semantic Definition: 'Formal Logic'

- Defines the meaning of language construct by making statements in the form of axioms or inference rules

Based on the Hoare's program (Hoare Triple), which is in the form,

$\{P\}\ S\ \{Q\}$ Where, P → Pre-Condition
Q → Post-Condition
S → Statement

- If S statement is executed in state where P holds, then it terminates and Q holds.
- Pre-Condition describes the state of the program before execution (Like INPUT)
- Post Condition defines the state after the execution (Like OUTPUT)
- Here, P and Q are predicates that holds Boolean value either TRUE OR FALSE.

Eg: $\{x=5\}$ $x=x+1$ $\{x=6\}$

P S Q

TRUE After execution must be

- $\{J=3 \text{ AND } K=4\}$ $j=j+k$ $\{j=7 \text{ AND } K=$

- $\{x \geq 0\}$ while $(x \neq 0)$ do $x=x-1$ $\{x=0$

- $\{x<0\}$ while $(x \neq 0)$ do $x=x-1$ infinite $\{x=0\}$

PAS terminates → Q (after Code run)

- Partial Correctness:

- A Program is partially correct with respe to pre Condition and post Condition, pro that if the program is started with the values that makes the pre Condition true, the resulting value make the post Condi true, when the program halts (if ever) i.e. if answer is returned, it will be C

- Total Correctness:

- The program terminates when started with values satisfying pre conditio, the program is called totally correct i.e it requires that the algorithm terminates

- Sequence Statement Rule:

- It has the form:

$$\frac{H_1, H_2, \ldots\ldots, H_n}{H}$$

i.e. if $H_1, H_2, \ldots, H_n$ have all been verified then H is also valid.

E.g:

✓ $\frac{P \rightarrow Q, P}{Q}$

$\{P\}\ S1\ \{Q\}, \{Q\}\ S2\ \{R\}$

$\{P\}\ S1, S2\ \{R\}$

$\{x > 1\}\ x = x + 1\ \{x > 2\}$

$\{x > 2\}\ x = x + 1\ \{x > 3\}$

$\{x > 1\}\ x = x + 1;\ x = x + 1\ \{x > 3\}$

- Alternation Statement Rule:

- Defined by the following inference rule;

- IF - THEN

$\{P \text{ and } B\}\ S\ \{Q\}$

$\{P \text{ and NOT } B\} \supset \{Q\}$

$\therefore \{P\}$ IF B THEN S END IF $\{Q\}$

- IF - ELSE B → Condition;

$\{P \text{ and } B\}\ S1\ \{Q1\}$

$\{P \text{ and NOT } B\}\ S2\ \{Q\}$

$\therefore \{P\}$ IF B THEN S1 ELSE S2 END IF $\{$

- LOOP WHILE

$\{P \text{ and } B\}\ S\ \{P\}$

$\therefore \{P\}$ While B do - SEND While $\{P$ and NO

- Disjoint Parallel Program:

- Two programs S1 and S2 are said to disjoint if none of them can change variables accessed by the other
- Denoted as $[S1 \,||\, S2]$

E.g.

$S_1 : x = z$ - BUT NOT → $S_1 : x = y + 1$
$S_2 : y = z$ $S_2 : x = z$

Semantics;

$$\frac{\{P_1\}\ S_1\ \{Q_1\} \quad \{P_2\}\ S_2\ \{Q_2\}}{\therefore \{P_1 \text{ and } P_2\}\ [S_1 \,||\, S_2]\ \{Q_1 \text{ and } Q_2\}}$$

- Await Then Rule:

- Await T then B, where T is a Condition and B is a block of Code

- If Condition T holds then block B is executed else the process is suspended until T becomes true

Eg: the bank has an automated program that sends an email if the balance of customer is less than 1000.... here B is the block of Codes that Sends the email and has to await until the balance < 1000

Semantics;

$$\frac{\{P \text{ and } T\}\ B\ \{Q\}}{\therefore \{P\}\ \text{Await } T \text{ then } B\ \{Q\}}$$

(2) Operational Semantics: → program exe. steps.

- Focuses on how the state of the program is affected
i.e. how a computation is performed

- Meanings for program phrases defined in terms of the steps of computation they can take during program execution

E.g:
To execute $x = y$, first determine value of y, and assign it to x

- Uses the notation $\langle P, S \rangle$, means Semanti of program P at state S

E.g

$\langle z = x, x = y, y = z, [x \to 5, y \to 7, z \to 0] \Rightarrow$

$\langle x = y, y = z, [x \to 5, y \to 7, z \to 5] \rangle \Rightarrow \langle P_2, S_2$

$\langle y = z, [x \to 7, y \to 7, z \to 5] \rangle \Rightarrow$

$\langle \ , [x \to 7, y \to 5, z \to 5] \rangle$

- So before starting the program, the stat of the memory is $[x \to 5, y \to 7, z \to 0]$, and after the program has terminated it $[x \to 7, y \to 5, z \to 5]$, which is meanin of the program $z = x, x = y, y = z$ in th State $[x \to 5, y \to 7, z \to 0]$

- Syntax directed translation can also be represented as operational Semantics.

E.g:

Top → down
Left → Right

E = Expression
T = Term

Production	Semantic Rule
$E \rightarrow E + T$	{Print ('+')}
$E \rightarrow E - T$	{Print ('-')}
$E \rightarrow T$	
$T \rightarrow 0$	{Print ('0')}
$T \rightarrow 1$	{Print ('1')}
$T \rightarrow 9$	{Print ('9')}

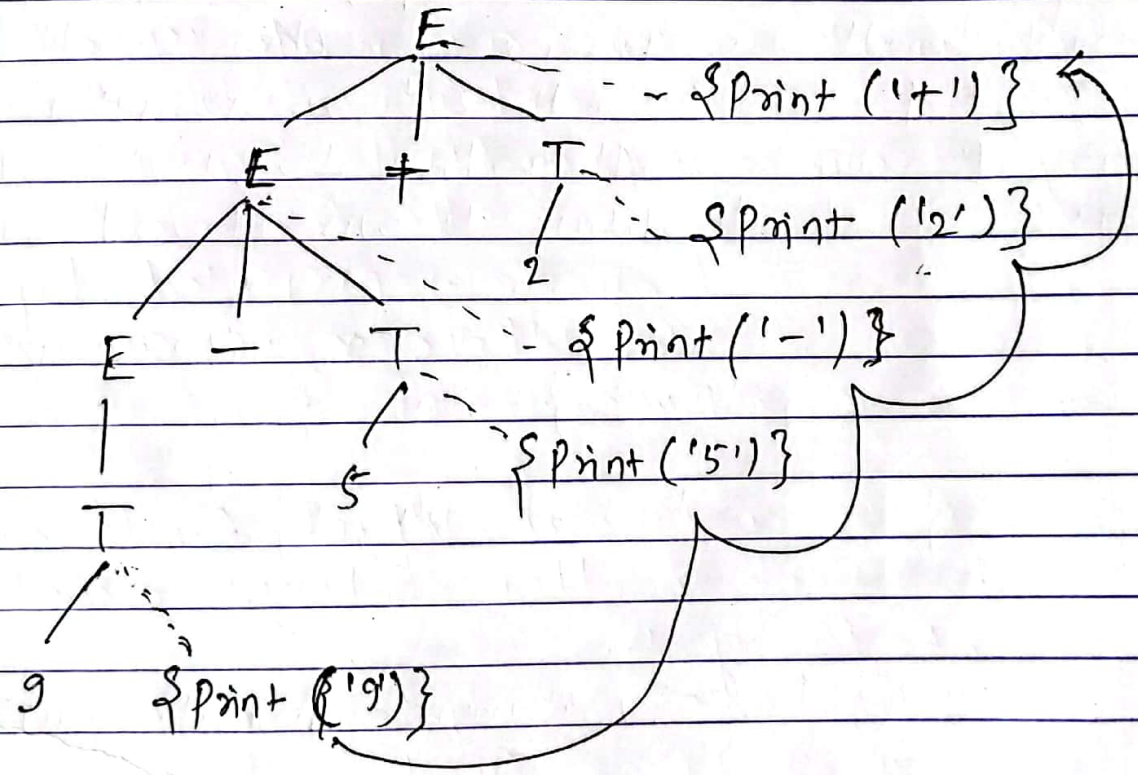


Fig: Action translating (9-5+2) into

95-2+

- Hennessy Milner Logic:

- Used to specify properties of a labe transition system
- Introduced by Mathew Hennessy and R Milner
- Syntax

Mapping it's construct to mathematical objects or functions.

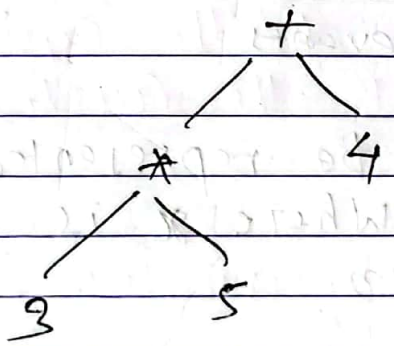
(3) Denotational Semantic Definitions:

- Each construct of the language is mapped into a mathematical object that defines it's meanings
- Idea of denotational semantics is to associate an appropriate mathematical object with each phrase of the language.

E.g: Syntax tree for 3*5+4 (postfix)

35*+4

35*4+



- Tradionally, the emphatic bracket ([[...]]), is used to represent the denotational semantics definition

E.g:

E1 + E2 = plus (Evaluate [[E1]], Evaluate [[E2]],)

- Also the ordered pair can be used to represent the program

E.g: fact(n) = if (n==0) then 1 }
else n * fact(n-1) }

Can be viewed as an ordered pair for denotational semantics as $\langle n, n! \rangle$,

$\{\langle 0,1\rangle, \langle 1,1\rangle, \langle 2,2\rangle, \langle 3,6\rangle, \langle 4,24\rangle, \ldots\}$

Communicating Sequential process:

- Concurrent systems are made up of processes where each process is defined by a sequence of events
- A process P can be represented by the notation $x \rightarrow Q$, where x is an event and Q is a process

  i.e. P as a process which first engage in the event x then behaves exactly as described by Q.
- Each process is defined with a particular alphabet which represent the event.
- Small letter (alphabet i.e. set of events), Capital letter (process)
- It is logically impossible for an alphabet to engage an event outside an alphabet

i.e. a machine designed to sell Chocbate Cannot deliver toy

E.g:

A Simple vending machine that serves a Single customer then stop can be represented as,

1. $VM_0 \overset{def}{=} (Coin \rightarrow (Candy \rightarrow stop))$ Single Custom
2. $VM_1 \overset{def}{=} (Coin \rightarrow (Candy \rightarrow VM_1))$ Infinite "
3. $VM_2 \overset{def}{=} (Coin \rightarrow (Candy \rightarrow VM_2)) \,||\, (notebill \rightarrow (toffee \rightarrow VM_2))$

↳ Vending machine that serves a Candy for a coin deposit & a toffee for a notebil deposit

(Event + Procen)

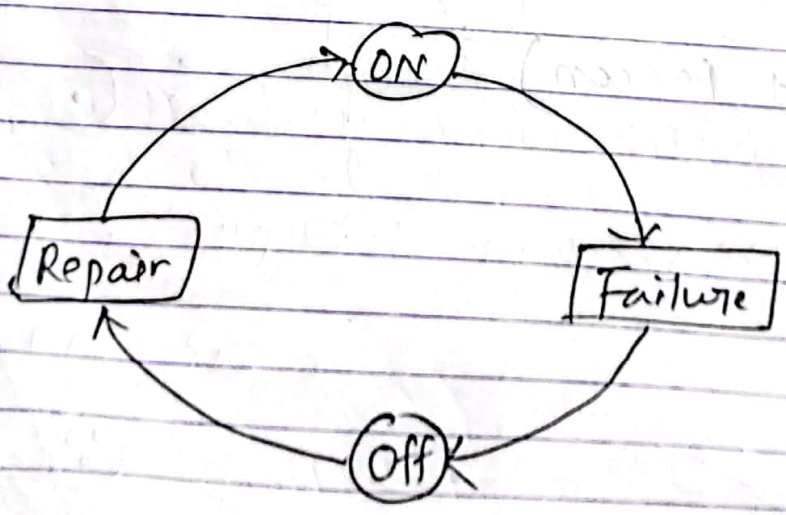
* Process Algebra:

- System behavior generally consists of proces and data
- Process are the control mechanism for th manipulation of the data
- Procen are dynamic and active, while the data are static and passive.

Petri Nets:

- It is the mathematical modeling langua for the description of the system i.e. used to model the functionality behavior of the system.

- Abstract model of information flow



- Petri Nets (PN) are represented through directed graph (bi-partite graph)

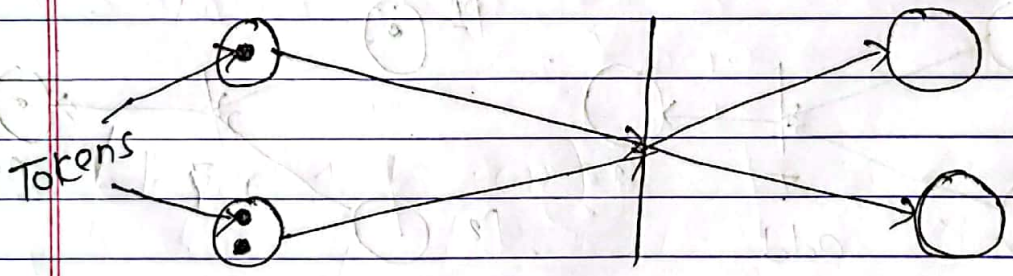
- Consists of two types of nodes

(1) Place (Buffer that holds something)
- Represented by circle

(2) Transition (Event / activity that takes place)
- Represented by Straight line (|), or rectangle (▭), or box (■)

- A place can be marked with a finite no. (possibly zero) of tokens.

- Token circulates in the system between places



Input places (pre condition) | Transition (T) | Output place (post Conditio

- A transition is called enabled, if for every arc from a place P to transition T, there exists a distinct token in the marking.

ie. a transition is fired whenever there is at least one token in all the input places.

- When it is fired, one token (by default) is removed from all the input places one token is added to all it's output places.

Example 1;

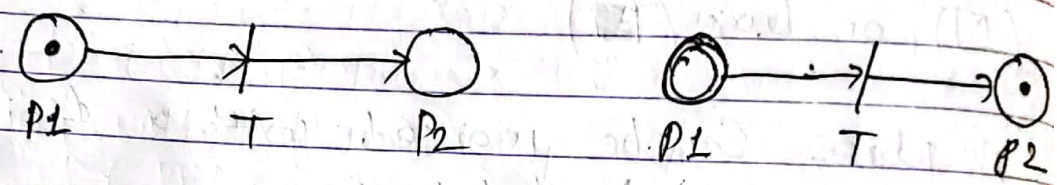
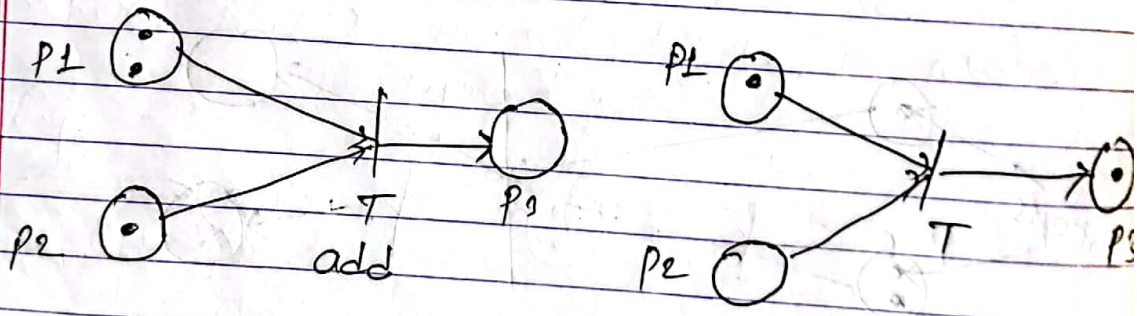


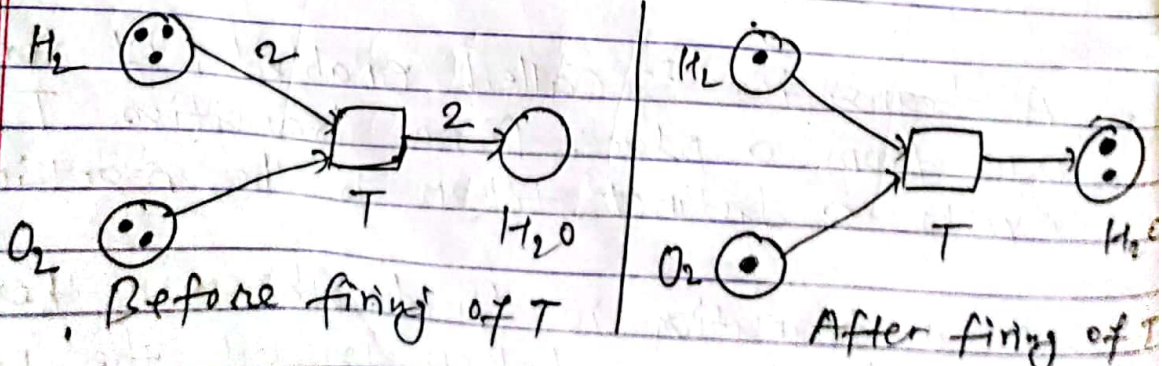
Fig: Before Firing of T Fig: After firing

Example 2;



Before firing of T After firing of T

Example 3;



Before firing of T After firing of T

Fig: Petri net showing an equation $2H_2 + O_2 \rightarrow 2H_2O$

Example 4;

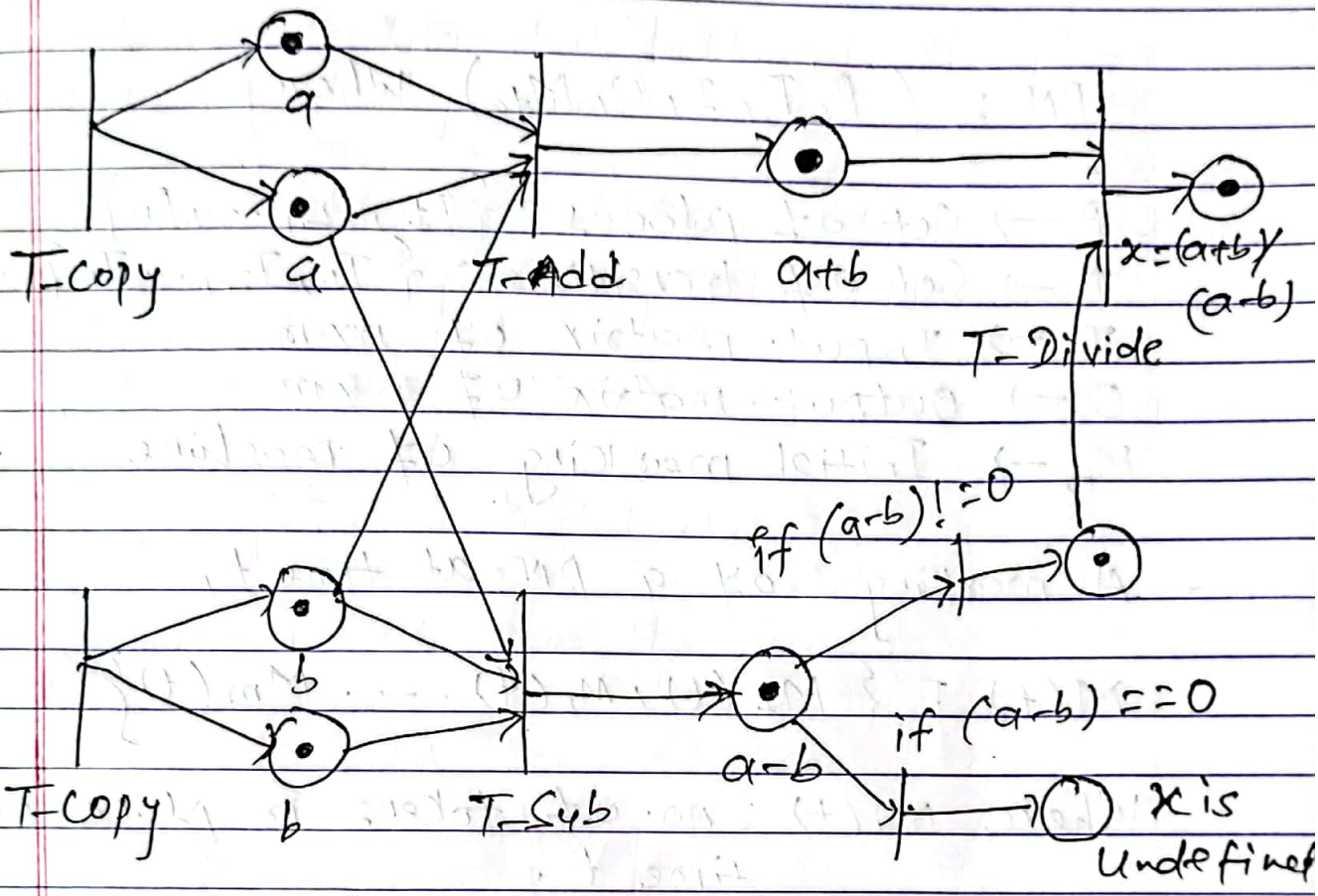


Fig: Petri net showing a data flow Computation of $x = \frac{a+b}{a-b}$

Sandhya
Date
Page

Formal definition:

$PN = (P, T, I, O, M_{to})$ Where,

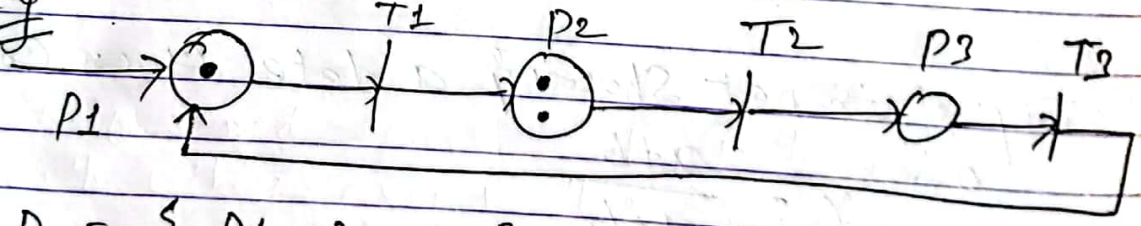
$P \rightarrow$ Set of places $= \{P_1, P_2, \ldots, P_n\}$
$T \rightarrow$ Set of transitions $= \{T_1, T_2, \ldots T_n\}$
$I \rightarrow$ Input matrix of $n \times m$
$O \rightarrow$ Output matrix of $n \times m$
$M_{to} \rightarrow$ Initial marking of machine

- A marking of a PN at time t,

$M(t) = \{M_1(t), M_2(t) \ldots M_m(t)\}$

Where, $M_i(t)$ = no. of tokens in place i at time t.

Eg:



$P = \{P_1, P_2, P_3\}$
$T = \{T_1, T_2, T_3\}$

$$I = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{array}{c} \begin{matrix} P_1 & P_2 & P_3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \qquad O = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

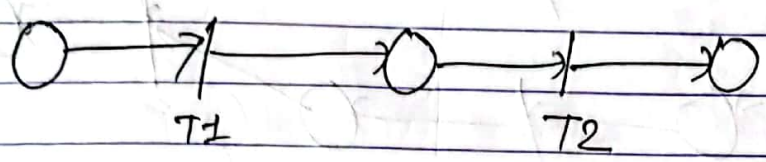
$M_{to} = (1 \; 2 \; 0)$ — no. of tokens
$P_1 \; P_2 \; P_3$

SDCCC

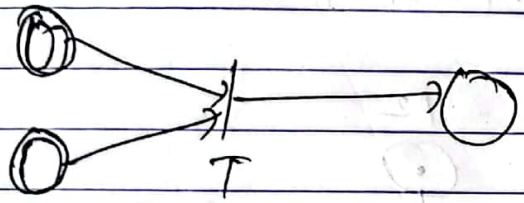
Sandhya
Date
Page

* Variations in PN:

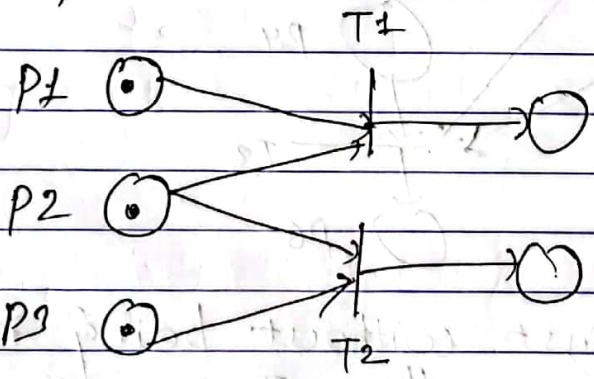
1. Sequential Action:



2. Dependency:

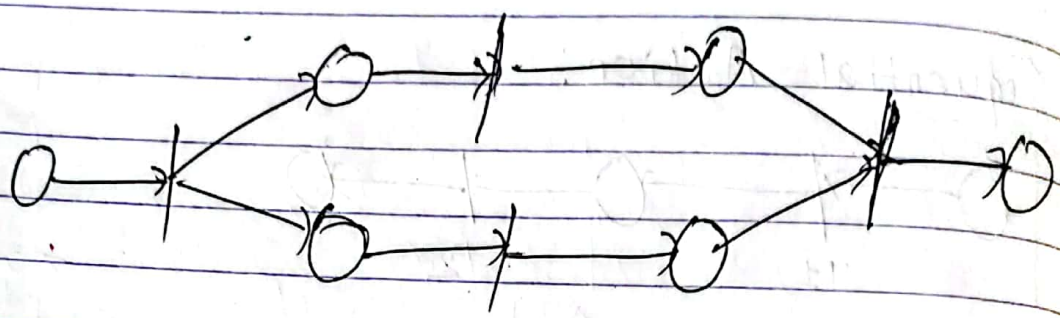


3. Conflict:

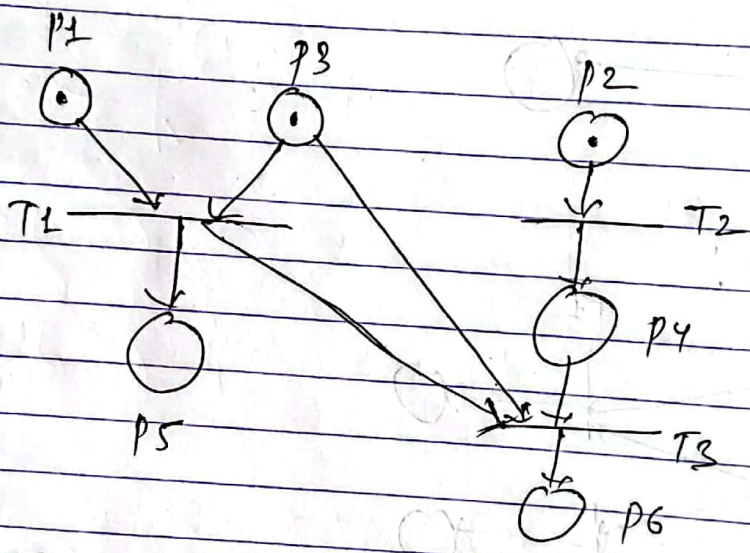


- P1 and P2 have tokens but not P3 then T1 is fired
- P2 and P3 have tokens but not P1 then T2 is fired
- If all have tokens which one to fire then Conflict (solution may be priority)

4. Concurrency:



5. Confusion: i.e. Conflict + Concurrency



- T1 occured first without being in Conf with other events then T2 occurred.
- T2 occured first and then T1, T3 got Conflict

- Conflict $(T_1, C_1) = \phi$ where $C_1 = \{P_1, P_2, P_3\}$
- Conflict $(T_1, C_2) = \{T_2\}$ where $C_2 = \{P_1, P_3, P$

Sandhya
Date
Page

* Pure and Simple Net:

* Properties of Petri Nets:

1. Reachability
2. Boundness
3. Liveness
4. Reversibility

} RBLR

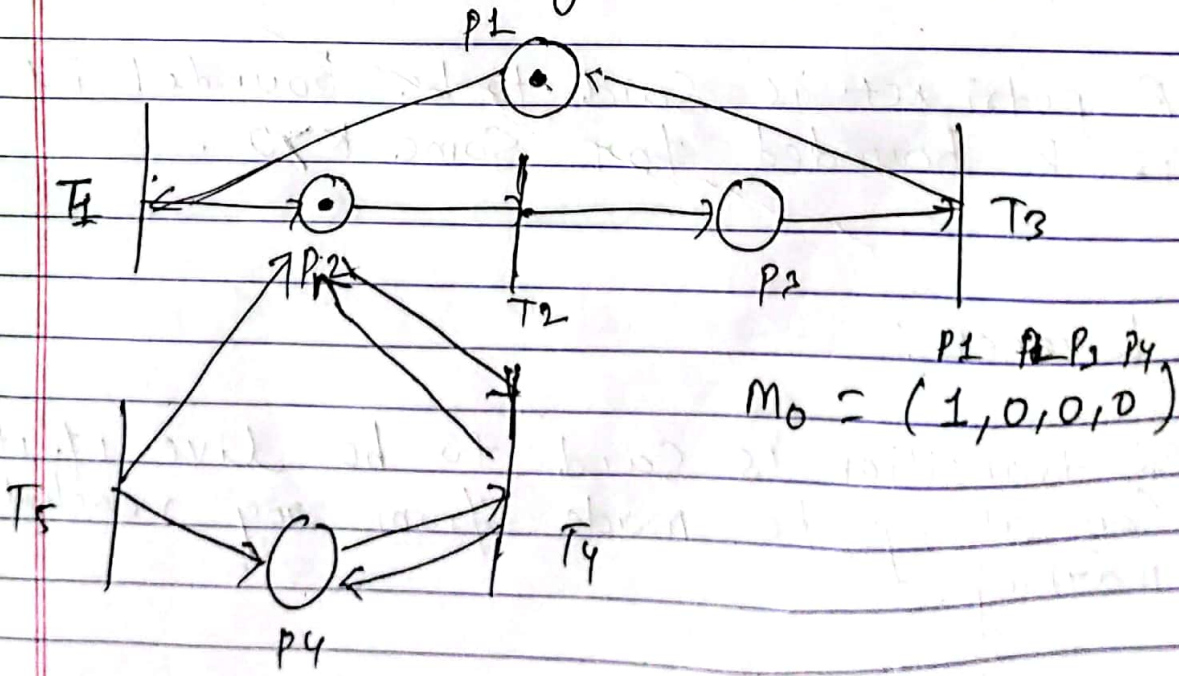
1. Reachability:

- A marking M is said reachable from another marking M', if there exists a sequence, σ, of transitions such that

$$M' [\sigma > M$$

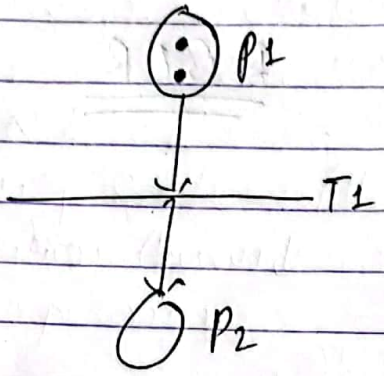
i.e. $M' t_1 M'' t_2 M''' t_3 \ldots t_n M$, where $\sigma = t_1, t_2, \ldots t_n$

$R(M_0)$ → Set of markings, reachable from the initial marking M_0



$R(M_0) = \{(0,1,0,0), (0,0,1,0), (0,0,0,1)$

E.g.



$M_0 = (2,0)$

$R(M_0) = \{(1,1), (0,2)\}$

2. Boundness:

- Number of tokens in a place be bounded
- A place P is said to be K-bounded, if the number of tokens in P never exceed K, i.e. $M(P) \leq K$
- A petri net is said to be K-bounded if all places are K-bounded.
- A petri net is said to be bounded if it is K bounded for some $K > 0$.

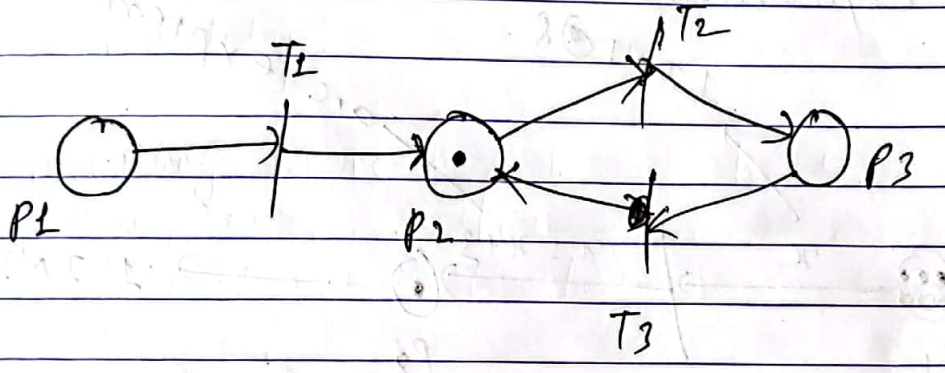
3. Liveness:

- A transition is said to be live if it can always be made from any reachable marking.

i.e. $\forall M \in R(M_0)), \exists M' \in R(M)$ such that

$M'[t\rangle$

- A transition is deadlocked if it can never fire.
- A petri net is live if all it's transitions are live.



T1 → Dead
T2 → Live
T3 → Live

4. Reversibility:

- A petri net is said reversible, if the initial marking remains reachable from any reachable marking.

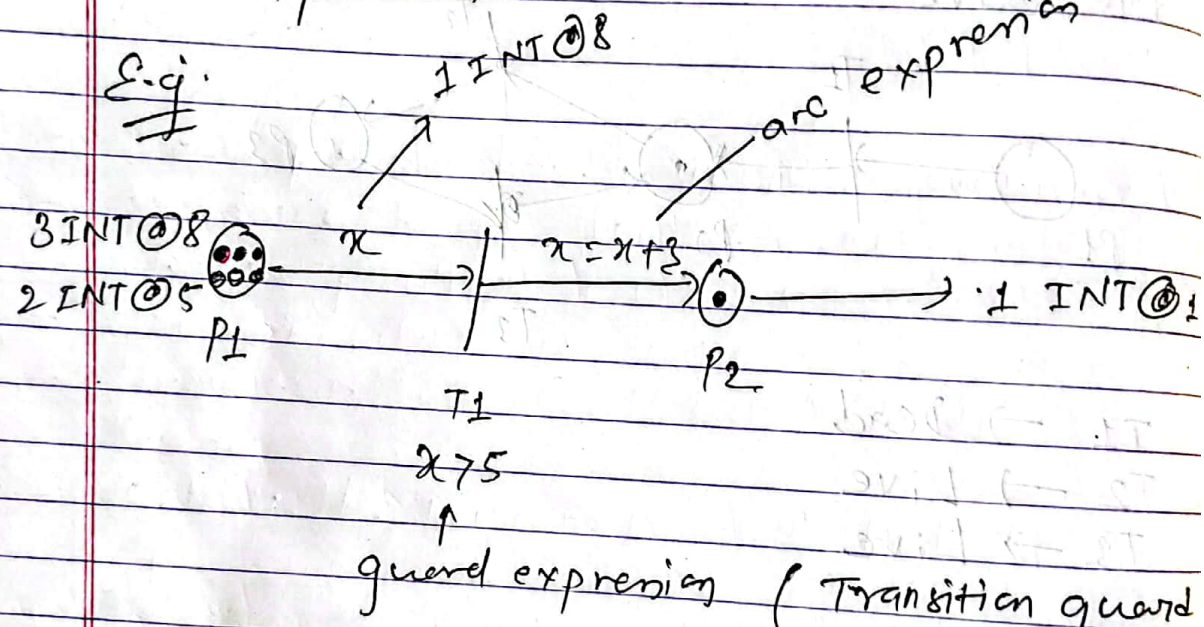
i.e. $M_0 \in R(M), \forall M \in R(M_0)$

* High Level Petri Net:

- Consists of following features

1. Petri Net
2. Marking Scheme (Like Color petrinet)
3. Transition Guard
4. Arc Expression

E.g.



Sandhya
Date
Page

* Complexity Theory :-

- Estimates the amount of Computational resources (time and Storage) need to Solve a problem.

Three basic aims of Complexity theory are:

(i) Introducing a notation to Specify Complexity

- Introduce a mathematical notation to Specify the functional relationship between the input Size of the problem and Consumption and Computational resources. E.g. Notation like Big.oh (O), Big Omega (Ω) Big theta (Θ).

(ii) Choice of machine model to standardize the measure

- Specify the underlying machine model to prescribe the measures for the Consumption of resources
- So that machine Cannot exceed the resour prescribed by the writer of the algorith E.g Turing machine (TM)

(iii) Refinement of the measures of parallel Compu

- How fast Can we Solve the problem, when a large number of processors are put together to work in parallel.

* Turing Machine as a basis of Consequ

1. Simplicity of abstraction
2. Dual nature of TM
3. Inclusion of non-determinism
4. Realization of unbounded parallelism
5. Provision of an easy abstract measure resources

* Distributed Computing:

Major obstacles in distributed computing is uncertainty due to differing procesor speed and occasional failure of components

- A system consists of n procesors $P_1, P_2 \ldots P_n$ and network, net.

Leader Election:

- In distributed computing, leader election is the procesn of designating a single procesor as the leader of some task distributed among several computer.

- The existence of leader can simplify co-ordination among the procesors and is helpful in acheiving fault tolerance and saving resources.

Algorithm; 'Election'

1. Bully Algorithm
2. Ring Algorithm

1. Bully Algorithm:

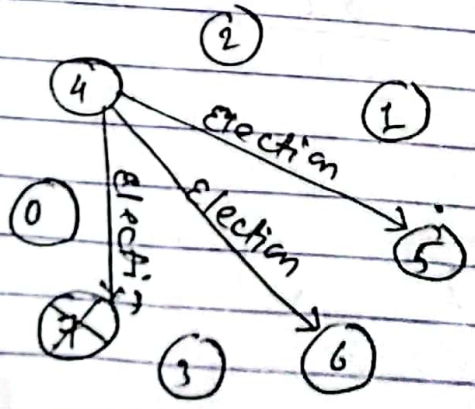
- The biggest guy in town always win.
- When a process notices that the Coordinator is not responding to request, it initiates as election.

Procedure:

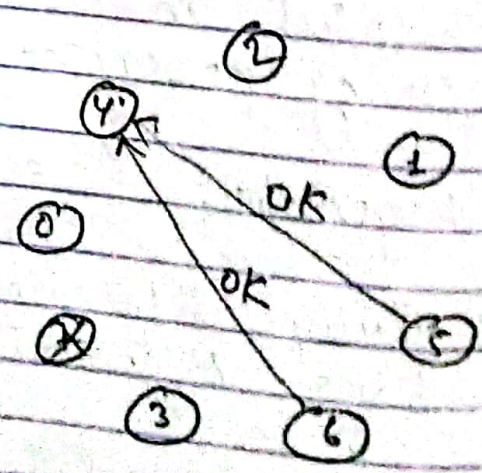
- Each process has a unique processor ID
- Processes know the ID's and address of every other processes
- Communication is assumed reliable.
- Process initiates election if the Coordinator failed
- Select the processor with highest ID.

E.g:

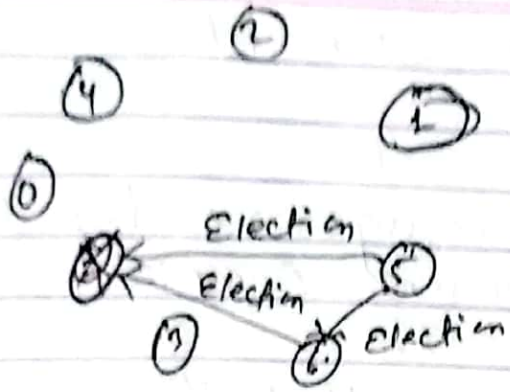
=>



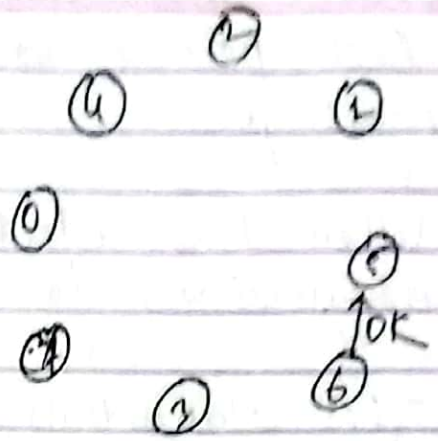
Since 7 is crashed it doesn't received the message.



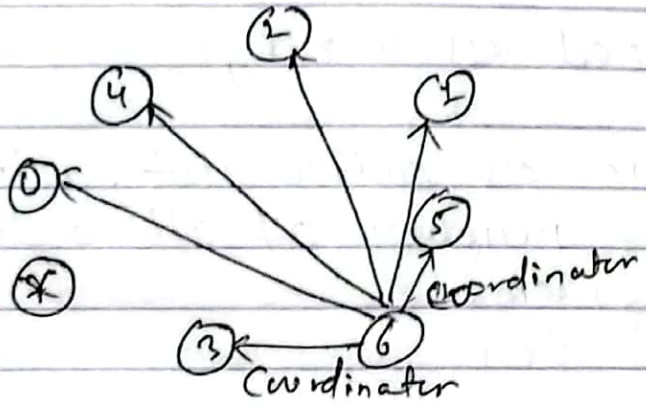
Process 5 and 6 re to the message sin process 7 previous Coordinator has crashed



Both procen 5 and 6 hold the election.



procen 6 is a winner or new coordinator.



Procen 6 is announcing.

2. Ring Algorithm:

- If any procen notices that the current Co-ordinator has failed, it starts an election by sending message to the first neighbor of the ring.

- The election message contains the node's identifier and is forwarded around the ring.

- Each procen adds it's own identifier to the message.

- When the election message reaches the originator, the election is complete.

- The algorithm assumes that the link between the procenes are unidirectional and every procen can manage to the procen on it's right only.

- System is oreganized as a ring.

- Uses data structure as active list, a list that has priority number of all active processes in the System.

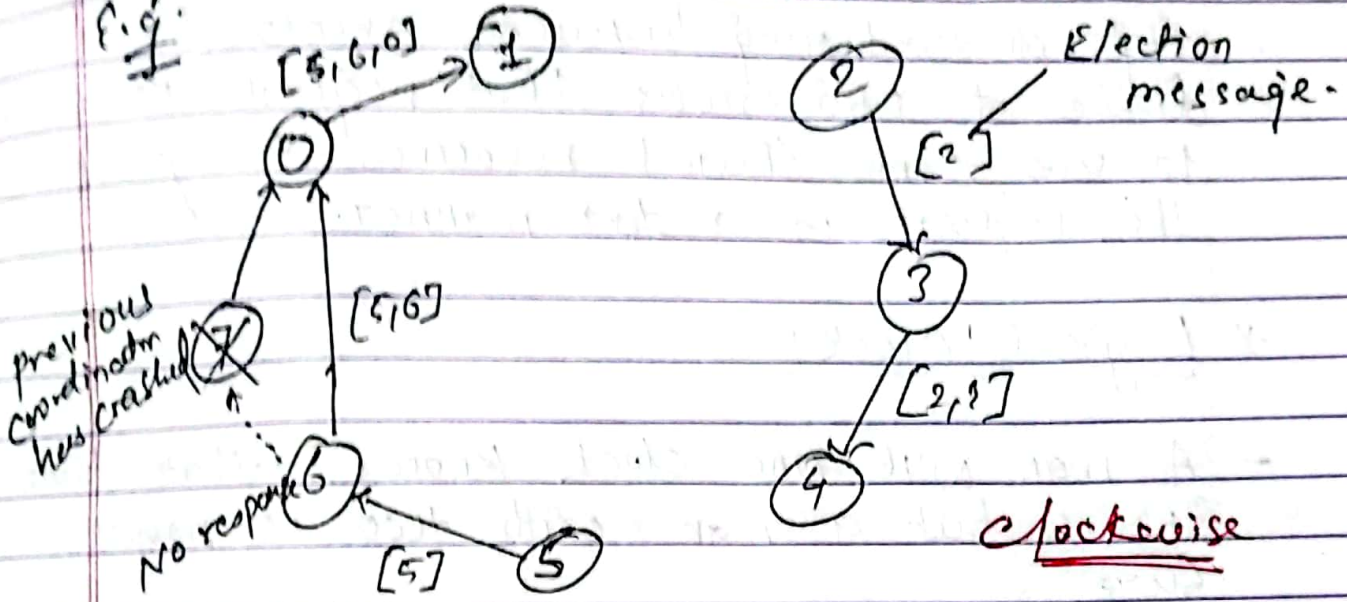
Procedure:

- If procen P1 detects a coordinator failur it creates a new active list which is empty initially. It sends election message to it's neighbor on right side and adds number, to it's active list.

- If procen, P2 receives message, in the same way it adds it's number in active list and forwarded the message to it's neighbor

- If a process receives it's own message, then it detects the process with highest number as a new leader.

E.g:



Step: 1 Initially, process 2 and 5 are initiating the election

Step: 2 They forward the election message to it's neighbor with it's ID's and so on until the circuit builds.

Step: 3 The active list for 2 and 5 will be

2 → [2, 3, 4, 5, 6, 0, 1, 2]

5 → [5, 6, 0, 1, 2, 3, 4, 5]

Step: 4 Process 6 has highest number. So, Both select 6 as a new coordinator or new leader.

* Ordering of events:

- Ordering between the events in the system.
- Eg: an ordering between events by different processors that request permission to use some shared resource to grant the request in a fair manner.

* Logical Clock:

- A man with one clock knows what time it is but a man with two is never sure
- Assign logical time stamps to events based on orderings among them
- If event a happened before event b, then it can be represented as

  $a \rightarrow b$ OR $a \lessdot b$

- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- If $a \lessdot b$ then $T(a) < T(b)$ where T is timestamp.
- But if $T(a) < T(b)$ then can we guarantee $a \lessdot b$.

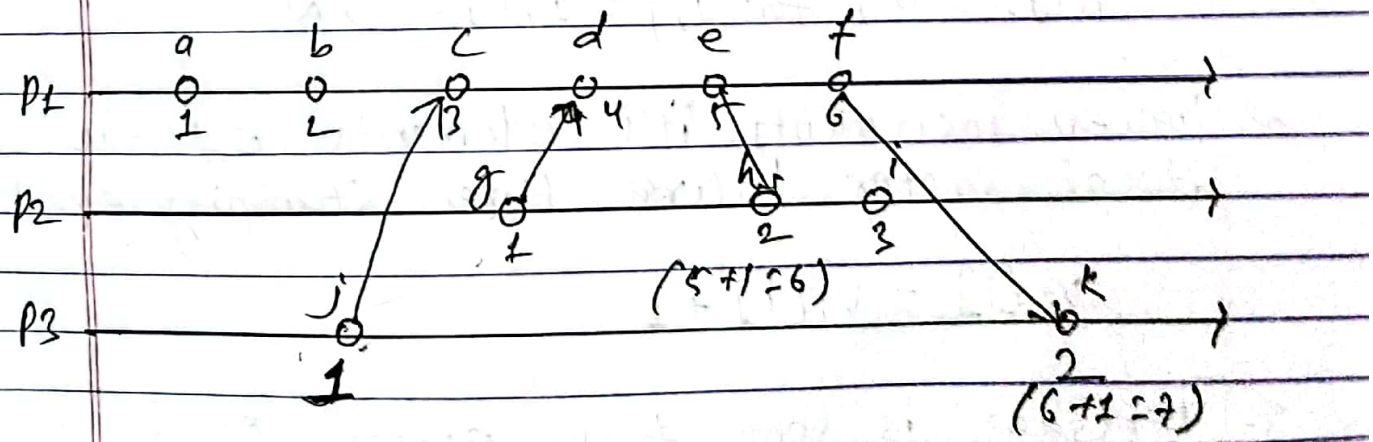
- Assign a clock to each events, if $a \to b$, then Clock(a) < Clock(b)

- If a and b occur on different process that do not exchange message, then neither $a \to b$ nor $b \to a$ are true, then events are called Concurrent.

Lamport's Logical Clock:

- Due to absence of a global clock in a distributed OS, Lamport's Logical Clock is needed.

- Let us assume the following example



- Each message carries a time stamp of the sender's clock

- When a message arrives

1. If receiver's clock < (message_time_stamp)
   set system clock to (message_time_stamp + 1)

2. Else do nothing.

Bad ordering:

event

(e,h) = 5 < 2
(f,k) = 6 < 2 } wrong

(j,c) = 1 < 3 Right

Vector clock:

Rules:

1. Vector initialization to 0 at each process

$V_i[j] = 0$, for $i, j = 1, 2, \dots \rightarrow N$

2. Process increments it's elements of the local vector before time stamping event.

$V_i[i] = V_i[i] + 1$

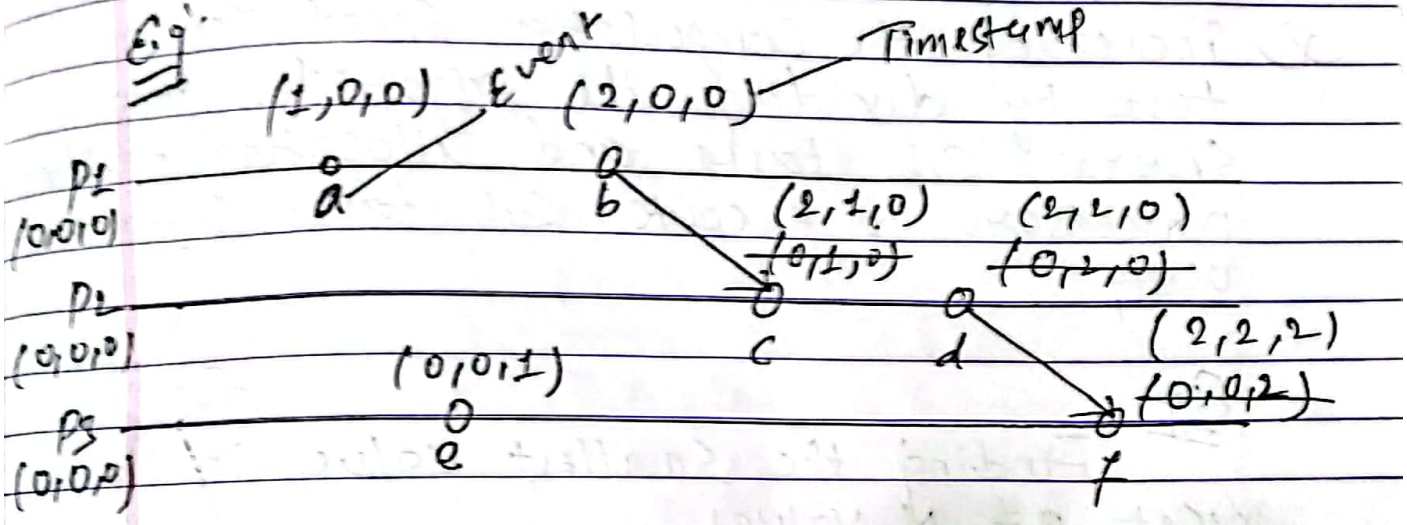
3. Message is sent from process P_i with V_i attached to it.

4. When P_i receives message, Compare vector elements, and set local vector to higher of two values.

$V_j[i] = \max\{V_i[i], V_j[i]\}$

For any two events e and e'

1. If $e \rightarrow e'$ then $V(e) < V(e')$
2. If $V(e) < V(e')$ then $e \rightarrow e'$



* PRAM Models:

(Parallel Random Access Machine Models)

- Parallel Can be defined as a technique for increasing the Computation Speed for a task by dividing the algorithm into several sub tasks and allocating multiple processors to execute sub tasks simultaneously.

Eg:

Finding the smallest value in the set of N values

Algorithm 1

- All possible comparisons of the pairs of the elements from set of number and carried out simultaneously each processor executing one operation of Comparison.

Algorithm 2

- Do in parallel (Actual no. of processor required

$$(P) = \frac{n(n-1)}{2}$$

- P_i derives the pair (i_1, i_2) of indices that corresponds to a different i:

L: Array

- P_i reads value $L(i_1)$ and $L(i_2)$
- If $L(i_1) > L(i_2)$ then P_i sends negative outcome to P_{i_1} else P_i sends negative outcome to P_{i_2}.
- At this stage the only active processors is P_j, $1 \le j \le h$, which did not receive a negative outcome.
- P_j reads the value of $L(j)$ and write it into the output cell.

Eg: $L = \{2, 6, 4, 8\}$, No. of elements $(n) = 4$

- So, no. of processors $(P) = \frac{n(n-1)}{2} = \frac{4*3}{2} = 6$

$P_1 \longrightarrow \langle 1,2 \rangle \longrightarrow$ -ve outcome to P_2

$P_2 \longrightarrow \langle 1,3 \rangle \longrightarrow$ " " " P_3

$P_3 \longrightarrow \langle 1,4 \rangle \longrightarrow$ " " " P_4

$P_4 \longrightarrow \langle 2,3 \rangle \longrightarrow$ " " " P_2

$P_5 \longrightarrow \langle 2,4 \rangle \longrightarrow$ " " " P_4

$P_6 \longrightarrow \langle 3,4 \rangle \longrightarrow$ " " " P_4

Here, Among all processor, P_4 didn't get -ve outcome 2 times $= L(2)$

Here, processor P_i is choose as

$$i = \frac{(i_2-1)(i_2-2)}{2} + i_1$$

E.g: Assignment of processor for Comparison index,

$(1,2) \longrightarrow \frac{(2-1)\times(2-2)}{2} + 1 = 1 \longrightarrow P_1$

$(1,3) \longrightarrow \frac{(3-1)\times(3-2)}{2} + 1 = 2 \longrightarrow P_2$

$(1,4) \longrightarrow \frac{(4-1)\times(4-2)}{2} + 1 = 4 \longrightarrow P_4$

$(2,3) \longrightarrow \frac{(3-1)\times(3-2)}{2} + 2 = 3 \longrightarrow P_3$

$(2,4) \longrightarrow \frac{(4-1)\times(4-2)}{2} + 2 = 5 \longrightarrow P_5$

$(3,4) \longrightarrow \frac{(4-1)\times(4-2)}{2} + 3 = 6 \longrightarrow P_6$

* PRAM Model:

1. EREW (Exclusive Read Exclusive Write)

- Every memory cell can be read or written to by only one procesor at a time.

2. CREW (Concurrent Read Exclusive Write)

- Multiple procesors can read a memory cell but only one can write at a time.

3. ERCW (Exclusive Read Concurrent Write)

- Never Considered.

4. CRCW (Concurrent Read Concurrent Write)

- Multiple procesors can read and write

(Issues: on concurrent write it can be assumed that all the procesors are writing the same value otherwise, priority can be assigned to solve.

* Optimality and Efficiency of parallel algorithm:

- Let A be a problem of size N.
- Assumed that A can be solved on a PRAM by a parallel algorithm P_A in time $T(N)$ by employing $P(N)$ processors.

Workdone:

$$W(N) = T(N) * P(N)$$

Speedup:

$$S = \frac{t(N)}{T(N)}$$ where $t(N)$ → sequential time

Efficiency:

$$E = \frac{\text{Speedup}}{P(N)}$$

Sandhya
Date
Page

* Basic PRAM algorithms:

1. Computing Prefix Sums
2. List Ranking
3. Parallel Sorting Algorithm

1. Computing Prefix Sums:

Given, array, $A = \{a_0, a_1, a_2, \ldots, a_{n-1}\}$

We have to compute,

$\{a_0, (a_0 + a_1), (a_0 + a_1 + a_2), \ldots (a_0 + a_1 + \ldots + a_{n-1})\}$

Eg:

$A = \{5, 3, -6, 2, 7, 10, -2, 8\}$

Output should be:

$\{5, 8, 2, 4, 11, 21, 19, 27\}$

Algorithm:

- Input an array $[a_1, a_2, \ldots, a_n]$
- If $(n == 1)$ then $S_1 \leftarrow a[1]$

i = index

- Else

For j=0 to logn - 1 do

For i = 2^j + 1 to n do in parallel

Procesor P_i

1. Obtains $a[i-2^j]$ from P_{i-2^j} through shared memory

2. $a[i] = a[i-2^j] + a[i]$

E.g.

	1	2	3	4	5	6	7	8
A = {	5,	3,	-6,	2,	7,	10,	-2,	8 }

n = 8 logn = $\log_2 8$ = 3

∴ J = {0, 1, 2}

Case I: J = 0

For i = 2^0 + 1 to 8

= 1 + 1 to 8 = 2 to 8.

(ii) $a[2] = a[2-2^0] + a[2]$

$= a[1] + a[2] = 5+3 = 8$

(iii) $a[3] = a[3-2^0] + a[3]$

$= a[2] + a[3] = 3-6 = -3$

(iii) $a[4] = -6+2 = -4$

(iv) $a[5] = 2+7 = 9$

(v) $a[6] = 7+10 = 17$

(vi) $a[7] = 10-2 = 8$

(vii) $a[8] = 8-2 = 6$

So, after 1st iteration new array is

1	2	3	4	5	6	7	8

$A = \{5, 8, -3, -4, 9, 17, 8, 6\}$

Case : II $J = 1$

For $i = 2^1 + 1$ to 8

$= 3$ to 8

(i) $a[3] = a[3-2^1] + a[3] = a[1] + a[3] = 5+3 = 8$

(ii) $a[4] = 8-4 = 4$

(iii) $a[5] = 9-3 = 6$

(iv) $a[6] = 17-4 = 13$

(v) $a[7] = 8+9 = 17$

(vi) $a[8] = 17+6 = 23$

So, after 2nd iteration new array is,

$A = \{5, 8, 2, 4, 6, 13, 17, 23\}$

Case III : $j = 2$

For $i = 2^j + 1$ to n

$= 2^2 + 1$ to $8 = 5$ to 8

(i) $a[5] = a[5-2^2] + a[5] = a[1] + a[5] = 5$

$= 11$

(ii) $a[6] = a[3] + 8 = 21$

(iii) $a[7] = 17 + 2 = 19$

(iv) $a[8] = 23 + 4 = 27$

∴ Final Array, $A = \{5, 8, 2, 4, 11, 21, 19, 27\}$

* List Ranking :-

- The problem is that, given a Singly linked list L, with N objects, for each node compute the distance to the end of the list.

- If d denotes the distance

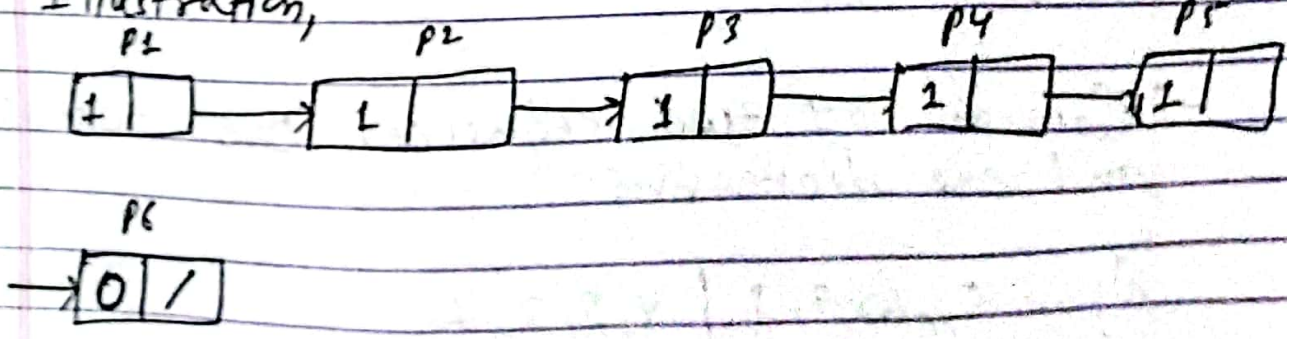
$$node.d = \begin{cases} 0, & \text{if } node.next = nil \\ node.next.d + 1, & \text{otherwise} \end{cases}$$

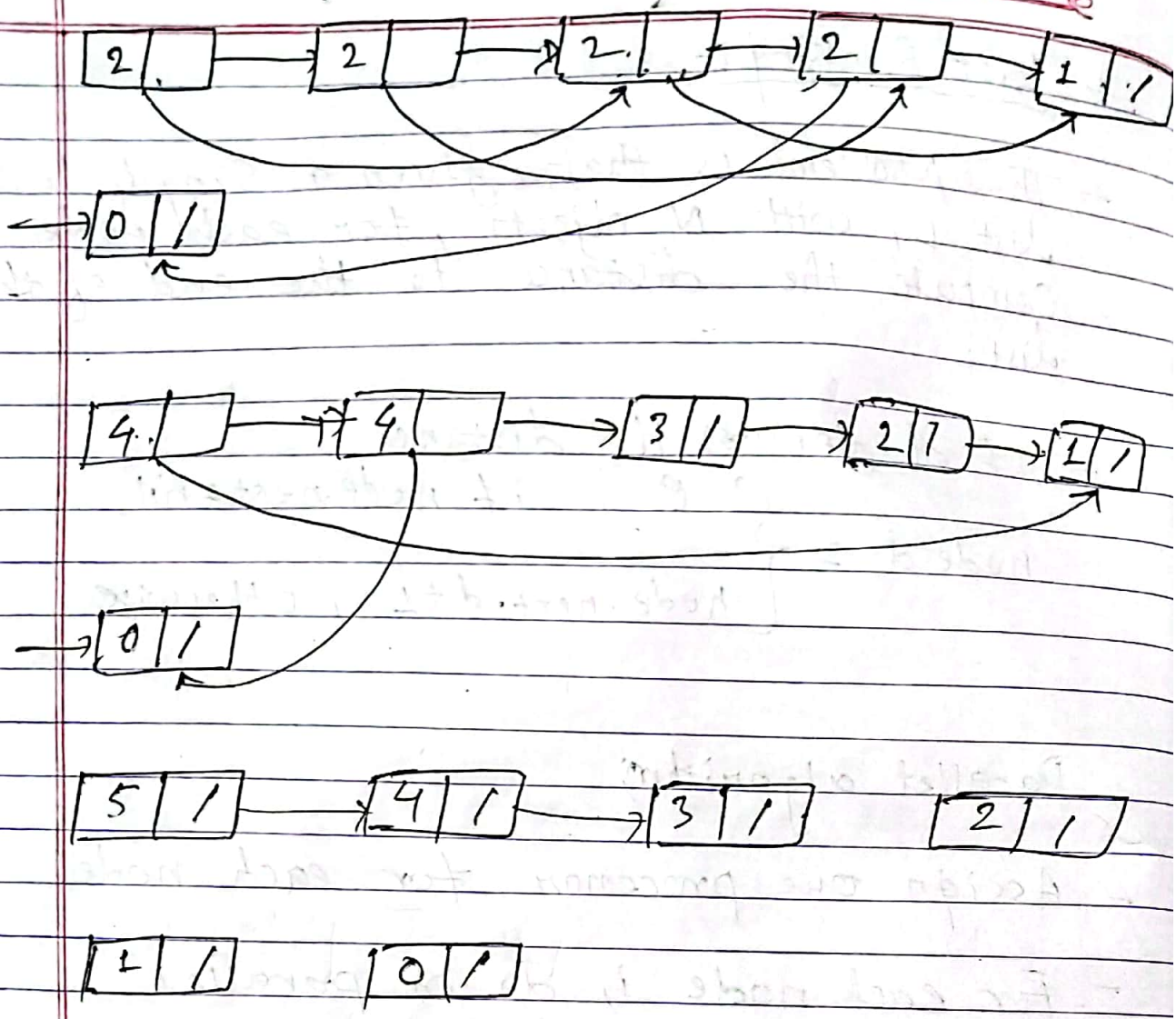
Parallel algorithm;

- Assign one processor for each node
- For each node i, do in parallel

$$\{ i.d = i.d + i.next.d \quad i.next = i.next.next \}$$

Illustration;





* Parallel Sorting Algorithm:

- Bitonic Sort

- Consists of two sequences, one increasing and one decreasing.

Eg. 5, 6, 7, 8 | 4, 3, 2, 1

- It is a sequence of elements $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ there exists $\langle a_0, a_1, \ldots a_i \rangle$ is increasing and $\langle a_{i+1}, a_{i+2}, \ldots a_{n-1} \rangle$ is decreasing

- If $a_{n/2}$ is the beginning of the decreasing sequence S, then

$L(S) = \{ \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots \min(a_{n/2-1}, a_{n-1}) \}$

$R(S) = \{ \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots \max(a_{n/2-1}, a_{n-1}) \}$

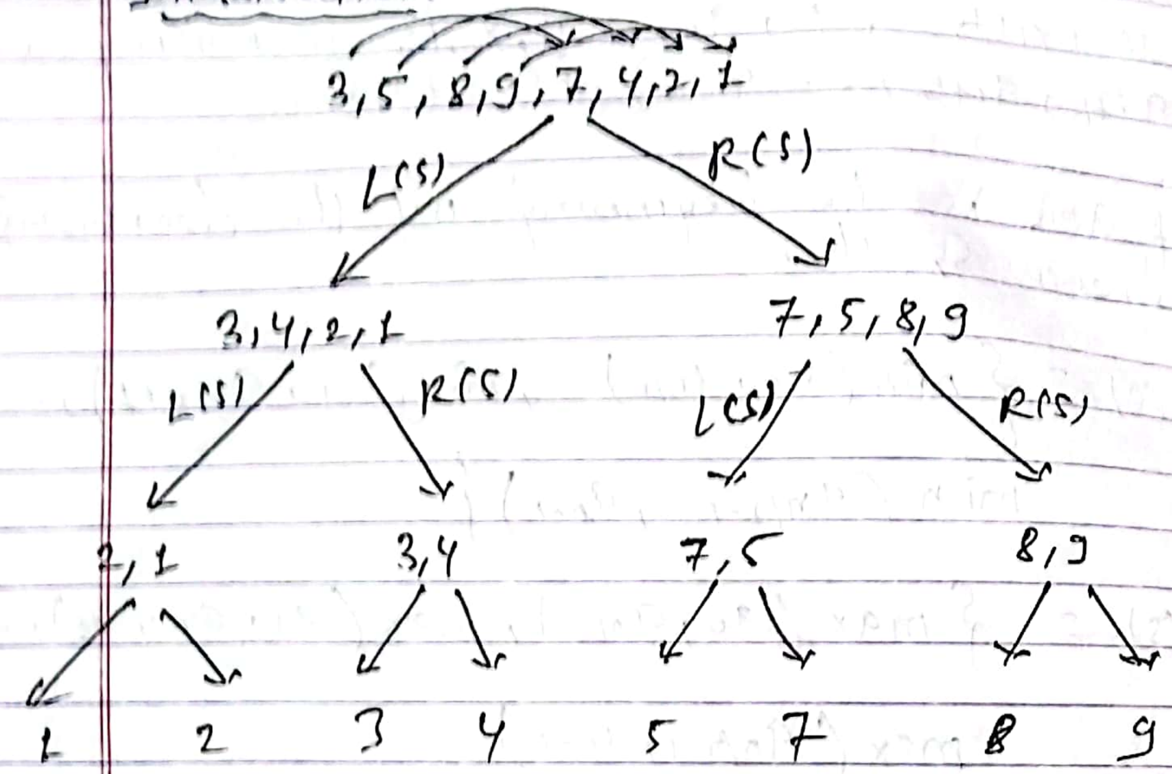
Algorithm;

- Input → a bitonic sequence S
- If S is the length of 1 then STOP
- Else

  Form L(S) and R(S)

- Do in parallel
  - L(S) ← Recursive bitonic merge L(S)
  - R(S) ← Recursive bitonic merge R(S)
  - Concatenate L(S) and R(S)

Illustration:



* Primality Testing: 'Randomized Algorithm'

$a^{n-1} \% n == 1 \rightarrow$ prime

else

not prime

$a < n$

* BSR (Broadcasting with Selective Reduction):

- Broadcasting with selective reduction is a model of parallel computation in which N procesors share M memory locations.

- At each memory location, a subset of the incoming broadcast data is selected (a/c to one or more appropriate selection criteria) and reduced to one value (using an appropriate reducing operator), which is stored in the memory.

- Broadcast instruction consists of three phases:

1. Broadcast phase:

- Allows all of the N procesors to write concurrently to all of the M memory locations.

- Each procesor P_i, $1 \leq i \leq N$ produced a record containing two fields, a tag t_i and datum d_i, to be stored.

2. Selection phase:

- After the data are received at each memory locations U_j, $1 \leq j \leq m$, a switch S_j will select the receiving data d_i by comparing tag value t_i by using a selection rule,

$$\theta \in [<, \leq, =, >, \geq, \neq]$$

3. Reduction phase:

- Selected data are reduces to a single value using reduction rule,

$$R \{\Sigma, \pi, \wedge, \vee, \oplus, \overset{\max}{\wedge}, \overset{\min}{\cup}\}$$

* Types of BSR:

1) One Criterion BSR:

- Each broadcast datum has to pass a single test being allowed to participate in the reduction process.

E.g: Sorting, parenthesis Matching, Optimal Sum Subsegment.

Sorting:

- Number of processors needed = N (number of elements in array is N)
- Rank of elements x_j can be expressed as,

$$r_j = \sum 1 \mid x_i \leq x_j$$

- When all the elements in the array are distinct, every datum x_j in it's position r_j are in the sorted array

Illustration: $A = \{3, 2, 6, 5\}$
$P_1\ P_2\ P_3\ P_4$

For 3 → $3 \leq 3, 2 \leq 3$, ∴ rank = 2
For 2 → $2 \leq 2$, ∴ rank = 1
For 6 → $3 \leq 6, 2 \leq 6, 6 \leq 6, 5 \leq 6$, ∴ rank = 4
For 5 → ~~5 ≤ 5, 4 ≤ 5~~ 4 $3 \leq 5, 2 \leq 5, 5 \leq 5$
∴ rank = 3

Hence the sorted array is {2, 3, 5, 6}.

- In case of duplicate,

E.g. $A = \{3, 4, 1\}$

(i) $A[1] = 3$

$t_1 = 3 - \frac{1}{1} \leq t_1 = 3 - \frac{1}{1}$ → True.

$t_2 = 4 - \frac{1}{2} \leq t_1 = 3 - \frac{1}{1}$ → False

$t_3 = 3 - \frac{1}{3} \leq t_1 = 3 - \frac{1}{1}$ → False

∴ Rank = True value = 1st.

(ii) $A[2] = 4$

$t_1 = 3 - \frac{1}{1} \leq t_2 = 4 - 1/2$ → True

$t_2 = 4 - 1/2 \leq t_2 = 4 - 1/2$ → True

$t_3 = 3 - 1/3 \leq t_2 = 4 - 1/2$ → True

∴ Rank = 3rd

(iii) A[2] = 3

$t_2 = 3 - \frac{1}{1} \leq t_3 = 3 - 1/3 \rightarrow$ True

$t_2 = 4 - 1/2 \leq t_3 = 3 - 1/3 \rightarrow$ False

$t_3 = 3 - 1/3 \leq t_3 = 3 - 1/3 \rightarrow$ True

$\therefore$ Rank = 2nd.

At last → A = {3, 3, 4}

2) Two criterion BSR:

- BSR algorithm where broadcast data are tested for their satisfaction of two criteria before being allowed to take part in reduction procen.

E.g.

Counting inversions in a permutation.

- Given a set, S, a permutation π of S is a set S', containing all elements of S, but in different order

- Inversions in permutations are those numbers of pairs, which are in disorder

E.g
No. of inversions in $\{1, 6, 2, 9, 5, 3\} : 3$
They are $\{6,2\}, \{6,5\}, \{9,5\}$

Rules:

$$y_j = \sum 1 \; / \; \{i<j\} \wedge \{\pi(i) > \pi(j)\}$$

3) Three criterion BSR algorithm:

- Have to pass three criterion to go to reduction procen
- E.g: vertical segment visibility

Rules:

$$\text{Lefttop}(\ell t_j) = \wedge i \; / \; \underbrace{x_i < x_j}_{1} \; \underbrace{\wedge \; t_i \geq t_j}_{2} \; \underbrace{\wedge \; b_i \leq t_j}_{3}$$

$$\text{Right top}(r t_j) : \wedge i / \; x_i \geq x_j \wedge t_i \geq t_j \wedge b_i \leq t_j$$

Where,

$x_i < x_j$ = defines the coordinates either lies left or right.

$t_i \geq t_j$ = decides the top is always great than the center line.

b_i & t_j : decides the bottom to decide left top.

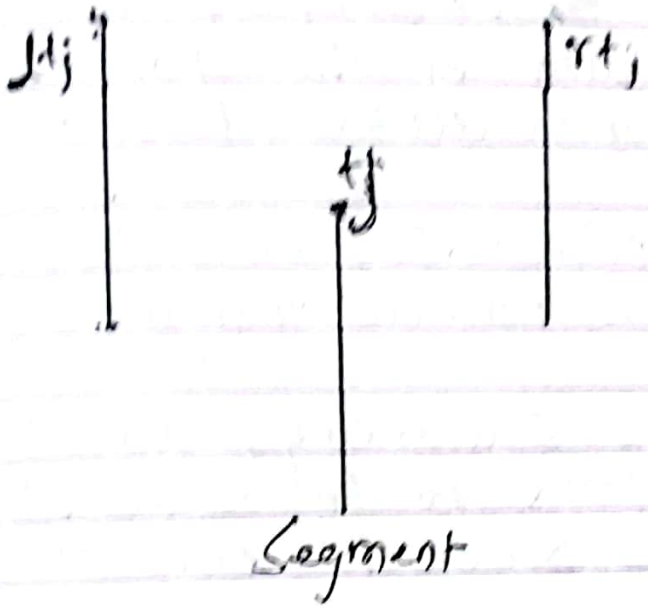


Fig: 3 criterion BSP

* Task Scheduling:-

- After program partitioning tasks must be optimally scheduled on the processors such that the program execution time is minimized.

- Scheduling techniques can be classified as deterministic and non-deterministic

- In deterministic scheduling, all the information about tasks to be scheduled is entirely known prior to execution time.

- In non-deterministic, some information may not be known before program execution (Eg. conditional branches)

* Scheduling System Models:

- It consists of:

1. Parallel program tasks
2. Target machine
3. Schedule
4. Performance criteria

1. Parallel program tasks:

- Can be defined as the system $(T, <, D_{ij}, A_i)$ where,

a. $T = \{t_1, t_2, \ldots, t_n\}$ is a set of tasks to be executed.

b. $<$ is a partial order defined on T, i.e. $t_i < t_j$ means t_i must be Completed before t_j can start execution.

c. D_{ij} is an $n \times n$ matrix of Communication data, where $D_{ij} \geq 0$ is the amount of data required to be transmitted from task t_i to t_j.

d. A_i is a vector of the amount of Computations i.e $A_i > 0$ is the number of instructions required to execute t_i.

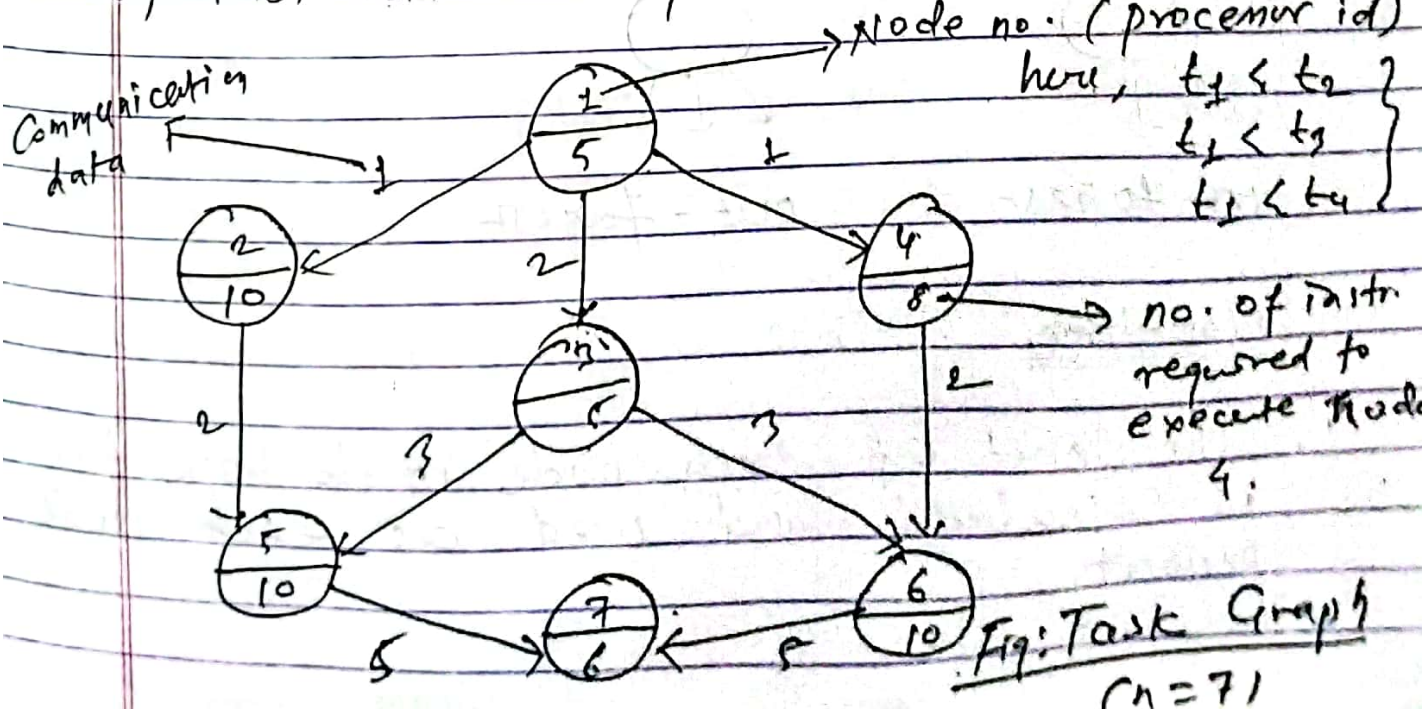


Fig: Task Graph (n = 7)

* Scheduling tree structure task graphs:

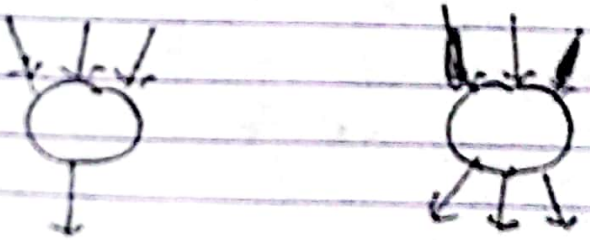
A. Scheduling in-forests / out-forests without Communication
B. Scheduling in-forests / out-forests with Communication

A. Scheduling in-forests / out-forests without Communication:

- Task graph is either,
- In-forest → each node has at most one immediate successors

or,

- Out-forest → each node has at most one immediate predecessors



in-forest out-forest

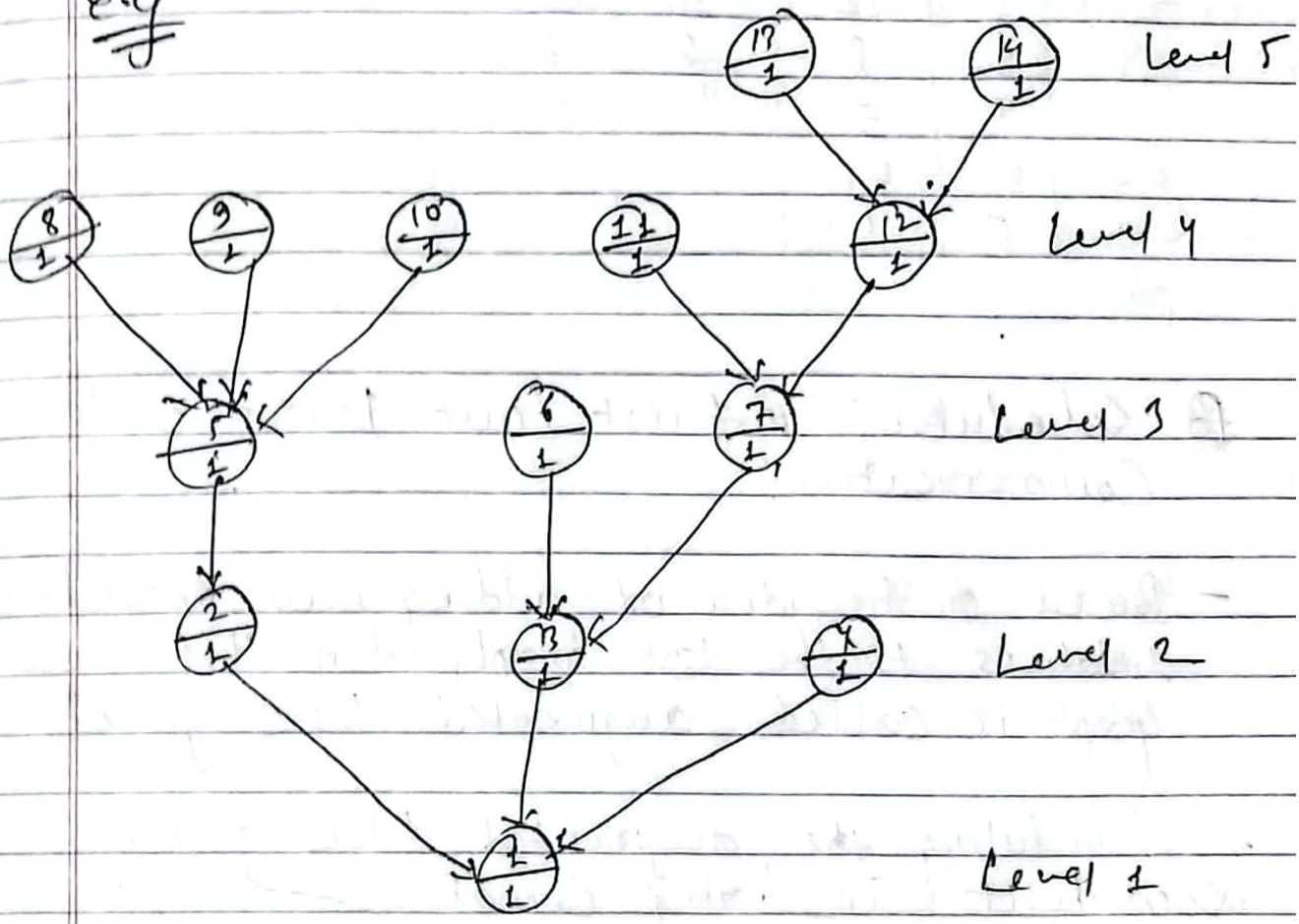
Algorithm;

1. The level of each node in the task graph is calculated and used as each node priority.

General strategy → used the highest level first

2. Whenever a processor becomes available, assign the unexecuted ready task with the highest priority.

E.g:



Node	Priority	Node	Priority
14	5	7	3
13	5	6	3
12	4	5	3
11	4	4	2
10	4	3	2
9	4	2	2
8	4	1	1

Fig: Task Priority

Task Scheduling:

	P1	P2	P3
0	14	13	11
1	12	10	9
2	8	6	7
3	5	3	4
4	2		
5	1		

- Scheduling in-forest / out-forest with Communication:

- Based on the idea of adding new precedence relations to the task graph, then this task graph is called augmented task graph.
- Scheduling the augmented task graph without considering Communication is equivalent to scheduling the original task graph with Communication.

Node-depth:

- The depth of a node is defined as the length of the longest path from any node with depth zero to that node.

depth(u) → 1 + max { depth(v) } ∀v ∈ predecessor

depth(q) → 0 if predecessor(u) = ϕ

operation swap-all:

- Given a schedule f, the operation swap_all(f, x, y), where x and y are two tasks in f scheduled to start at time t, on processors i and j.

- The effect of this operation is to swap all the task pairs scheduled on processors i and j is the scheduled at time t_1, $\forall t_1$, $t_1 \geq t$.

Algorithm;

1. Given an in-forest $G = (V, E)$, identify the set of siblings $S_1, S_2, \ldots S_k$ where S_i is the set of all nodes in V with a common child (S_i).

2. $A_1 \leftarrow A$

3. For every set S_i

   Pick node $u \in S_i$ with the maximum depth

   $A_1 \leftarrow A_1 - (v, \text{child}(S_i))$ $\forall v \in S_i$ and $v \neq u$

   $A_1 \leftarrow A_1 \cup (v, u)$ $\forall v \in S_i$ and $v \neq u$.

4. Obtain the schedule f by applying previous algorithm on the augmented in-forest $f = (V, A_1)$

5. For every set S_i in the original in-forc G_1 if node u (with the maximum depth) is scheduled in f in the time slot immediately before child (S_j), but in a different procenor, then apply the operati Swap_all.

E.g.:

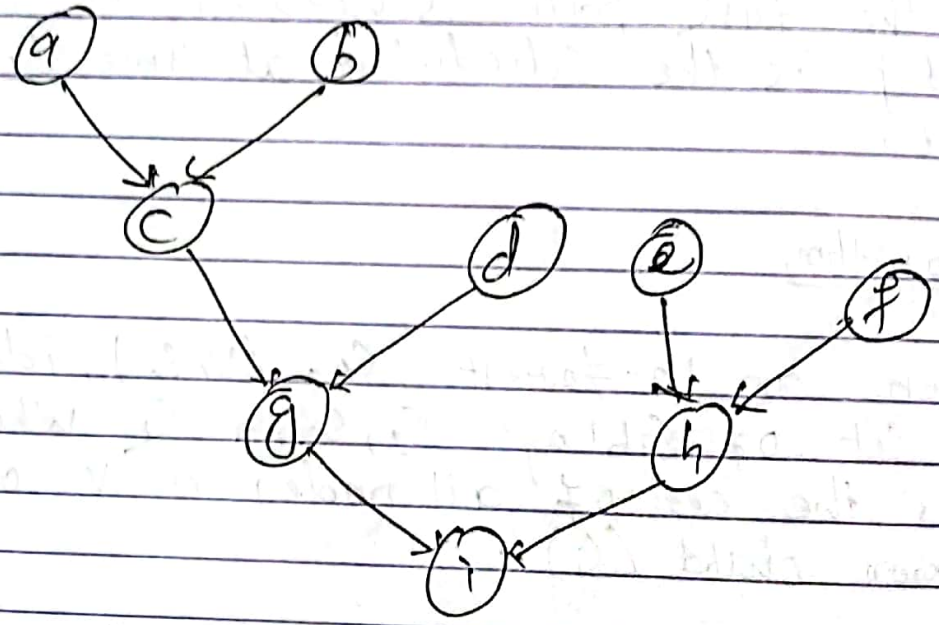


Fig: original task graph

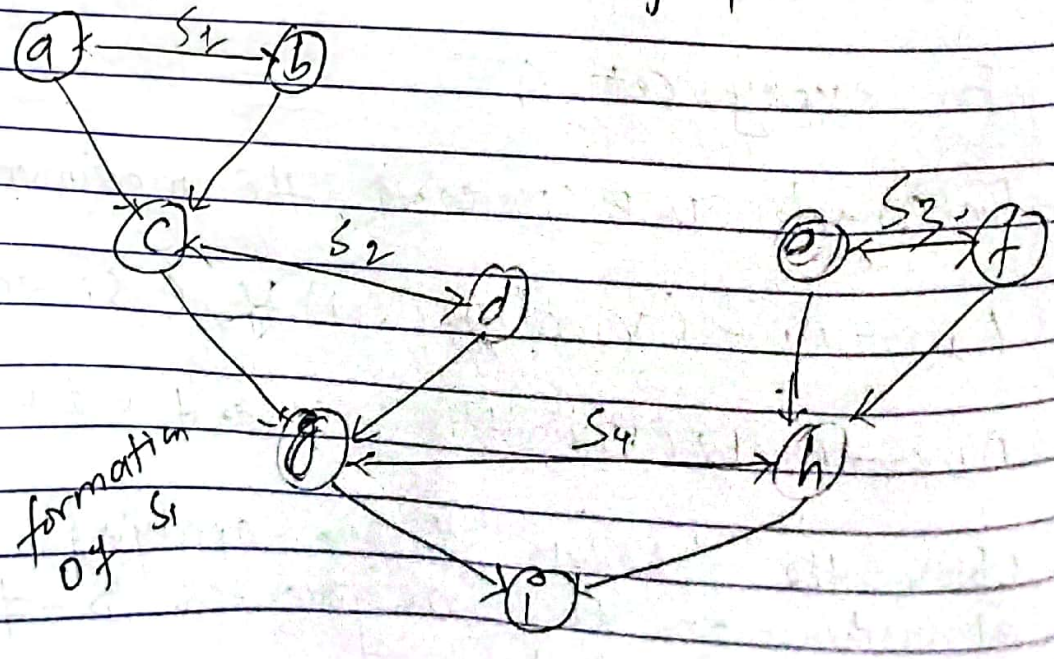


Fig: formation of S_1

- Depth of a and b in S_1 and e and f in S_3 are same.
- Depth of c is greater than d in S_2.
- Depth of g > h in S_4.

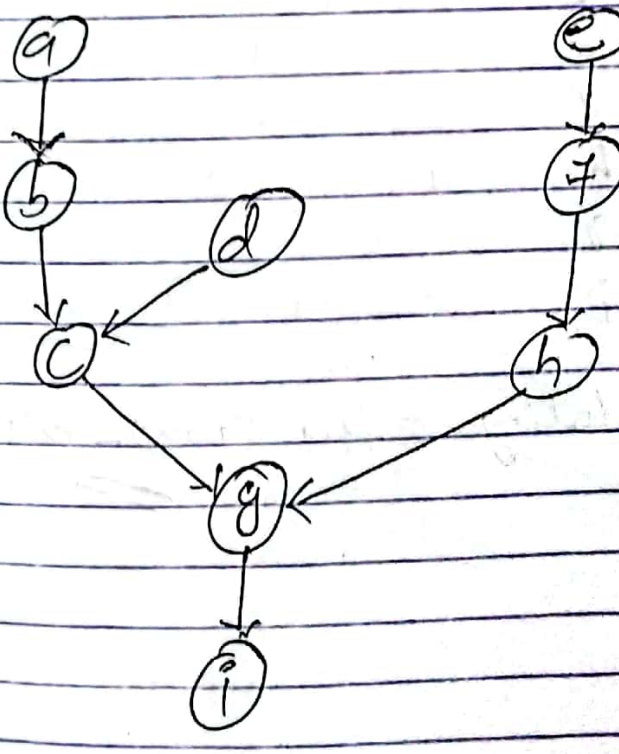
$S_1 = \{a, b\} \rightarrow b$

$S_2 = \{c, d\} \rightarrow c$

$S_3 = \{e, f\} \rightarrow f$

$S_4 = \{g, h\} \rightarrow g$

So, the augmented task graph is;



E.g:

Node	Level
a	6
b	4
c	3
d	4
e	5
f	4
g	2
h	3
i	1

Fig: Task Priority

Time	P_1	P_2
0	a	e
1	b	d
2	c	f
3		h
4		g
5		i

Fig: Task scheduling after swap-all

Time	P_1	P_2
0	a	e
1	b	d
2	f	c
3	h	
4	g	
5	i	

Fig: Task schedul

* Scheduling Interval Ordered Task:

- Used to indicate that the task graph which describes the precedence relation among the tasks in an interval order

1. Without Communication:

- The number of all successors of each node is used as each node's priority.

- Whenever a processor becomes available, assign it with the unexecuted ready task with the highest priority.

Ex:

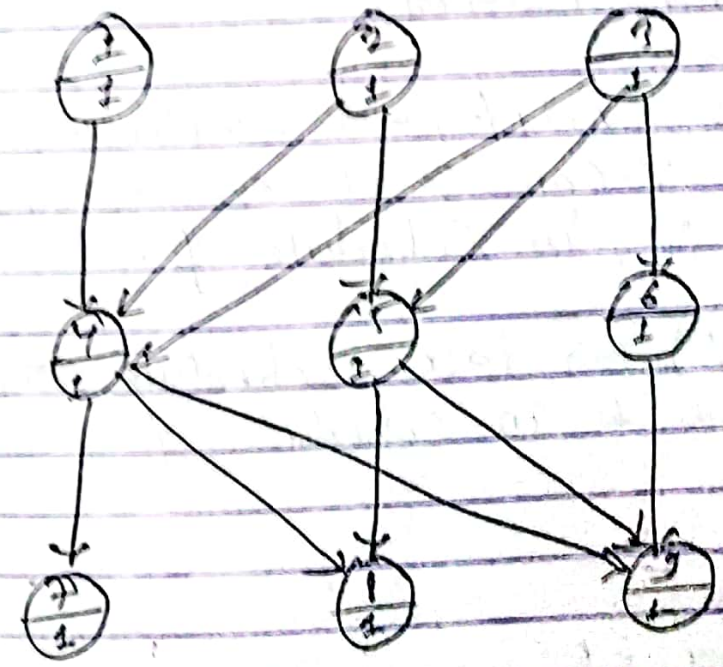


Fig: Task graph

Sandhya
Date
Page

Node	no. of successors
1	4
2	5
3	6
4	3
5	2
6	1
7	0
8	0
9	0

P1	P2	P3
3	2	1
4	5	6
7	8	9

Task Scheduling

Task Priority

2. With Communication:

- Start_time (v, i, f) → earliest time at which task v can start execution on procesor Pi, in Schedule f.
- task (i, t, f) → task schedule on procesor Pi at time t in Schedule f

Algorithm:

- The number of all successors of each node is used as each node's priority
- Nodes with highest priority are scheduled first

- Each task V is assigned to procesor P_i, with the earliest time start.

- If start_time (V_i, i, f) = start_time (V_j, f) $1 \leq i, j \leq m$, task V is assigned to processor P_i if task $(i$, start time $(V_i, i, f) - 1, f)$ has the smaller priority

- Communication delay is Considered as one unit of time

Eg:

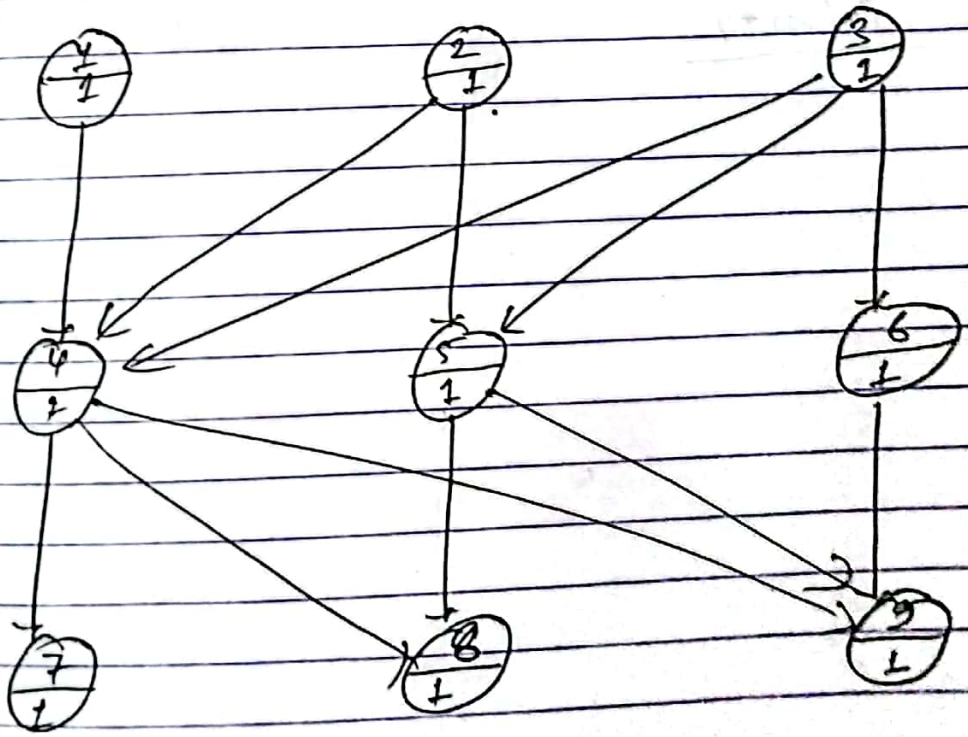


Fig: Task graph

Node	no. of successors
1	4
2	5
3	6
4	3
5	2
6	1
7	0
8	0
9	0

P1	P2	P3
3	2	1
6		
4	5	
7		
	8	9

Task scheduling

Task Priority

* Two Processor Scheduling:

- In this case, the no. of processors is limited to only two.

DLCR

Sandhya
Date
Page

* Checkpointing in Parallel and Distributed System:

1. Fault Detection:

- Procen of recognizing that an error has occurred in the system.

2. Fault Location:

- After the error in the system is detected, the location of the part in the system that cause the error is identified.

3. Fault Containment:

- After the fault is located, the faulty part is isolated to prevent the possible propagation.

4. Fault Recovery:

- Procen of restoring the operational stages of the system

* Checkpointing using task duplication:

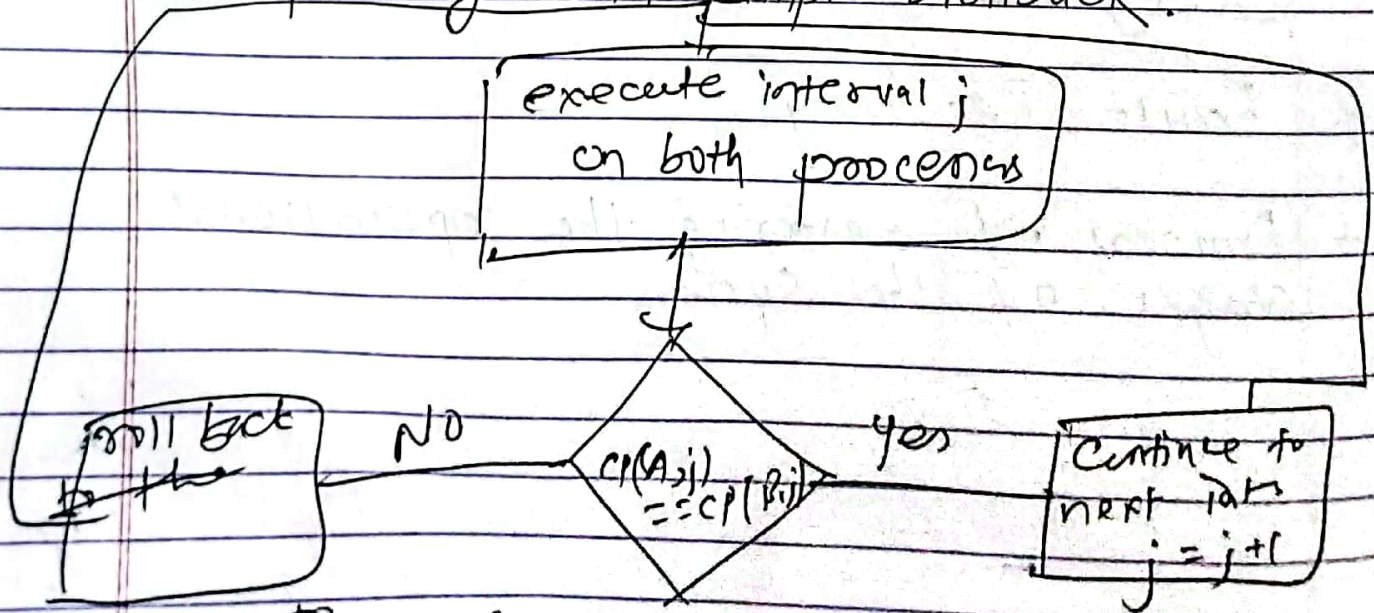
- Task duplication is analogous to shadowing techniques for recovery in database transaction system.

- Compares the state of same task that are executed in parallel on different processors.

- Here fault detection is achieved by duplicating the task into two or more processors and comparing the states of the processors at the checkpointing.

- Matching states is an indication of a correct execution.

Checkpointing with simple rollback:

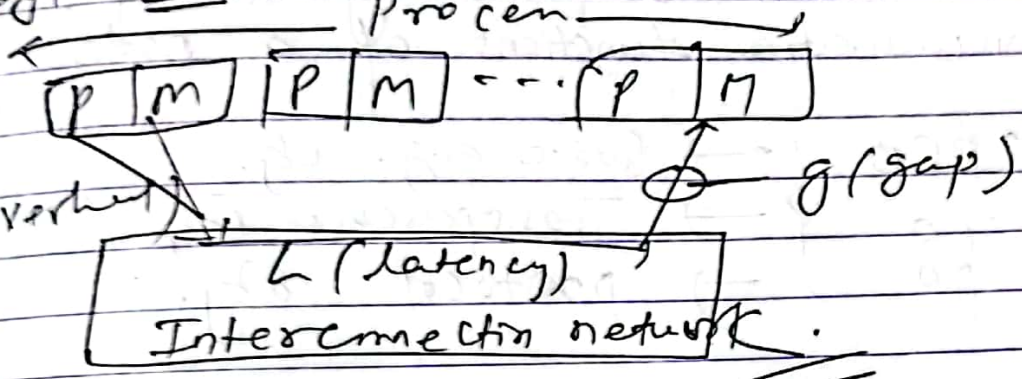


processors to the beginning of interval j

* ~~Communication Model~~ :- 'Model → A, B, C'

* Models of parallel Computation.

Log model



Open distributed System :-

Eg Telecommunication System →

(1) Components - distributed

① End user viewpoint (content view of user)

② Enterprise viewpoint (needs of users info)

③ Computational viewpoint (designer or programming

④ Information viewpoint (enterprise info).

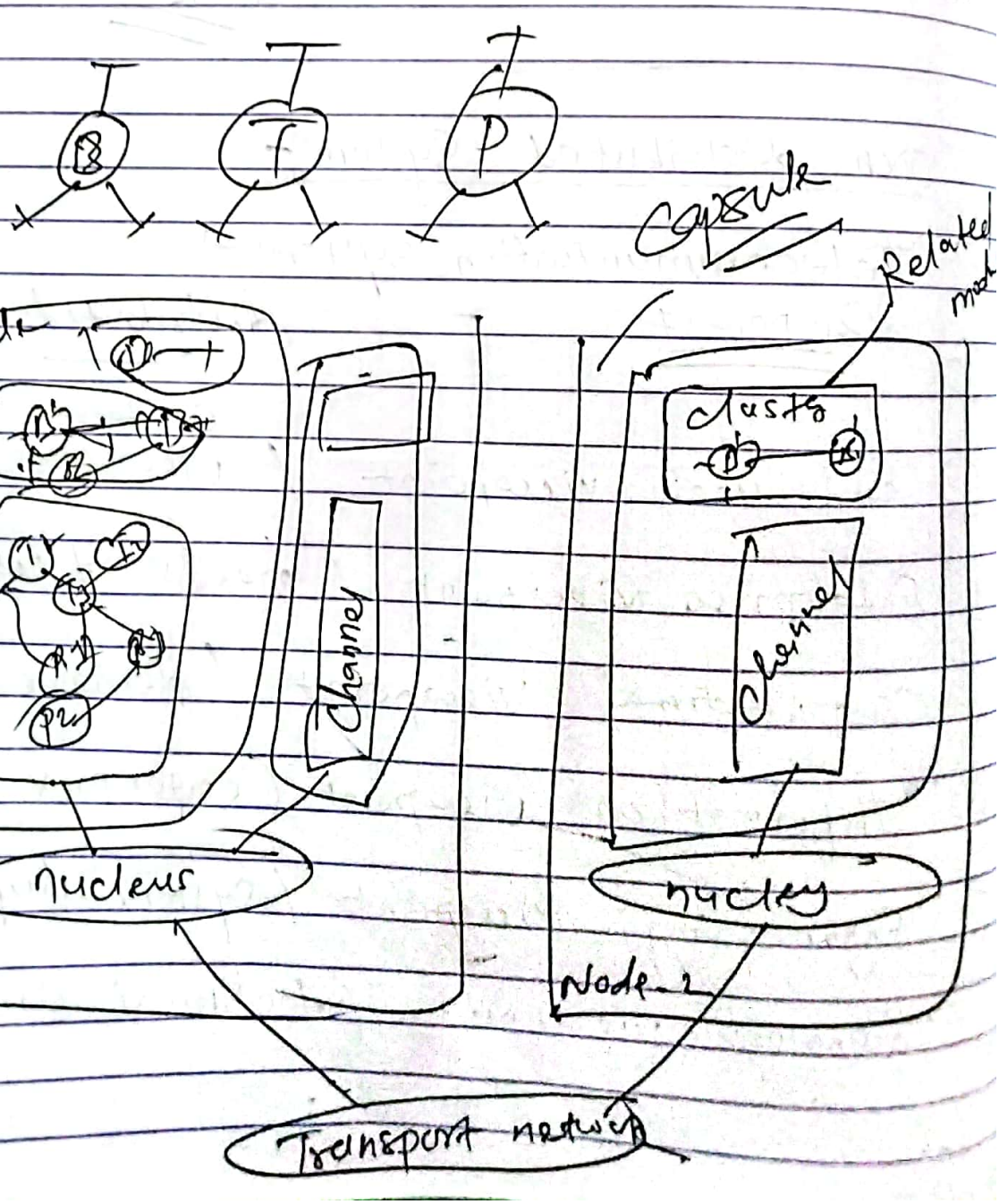
⑤ Egineering viewpoint (System support)

⑥ Technology viewpoint (technical component)

Engineering model;

→ A nucleus is an object that provides access to basic processing storage, and communication functions of a node for use by

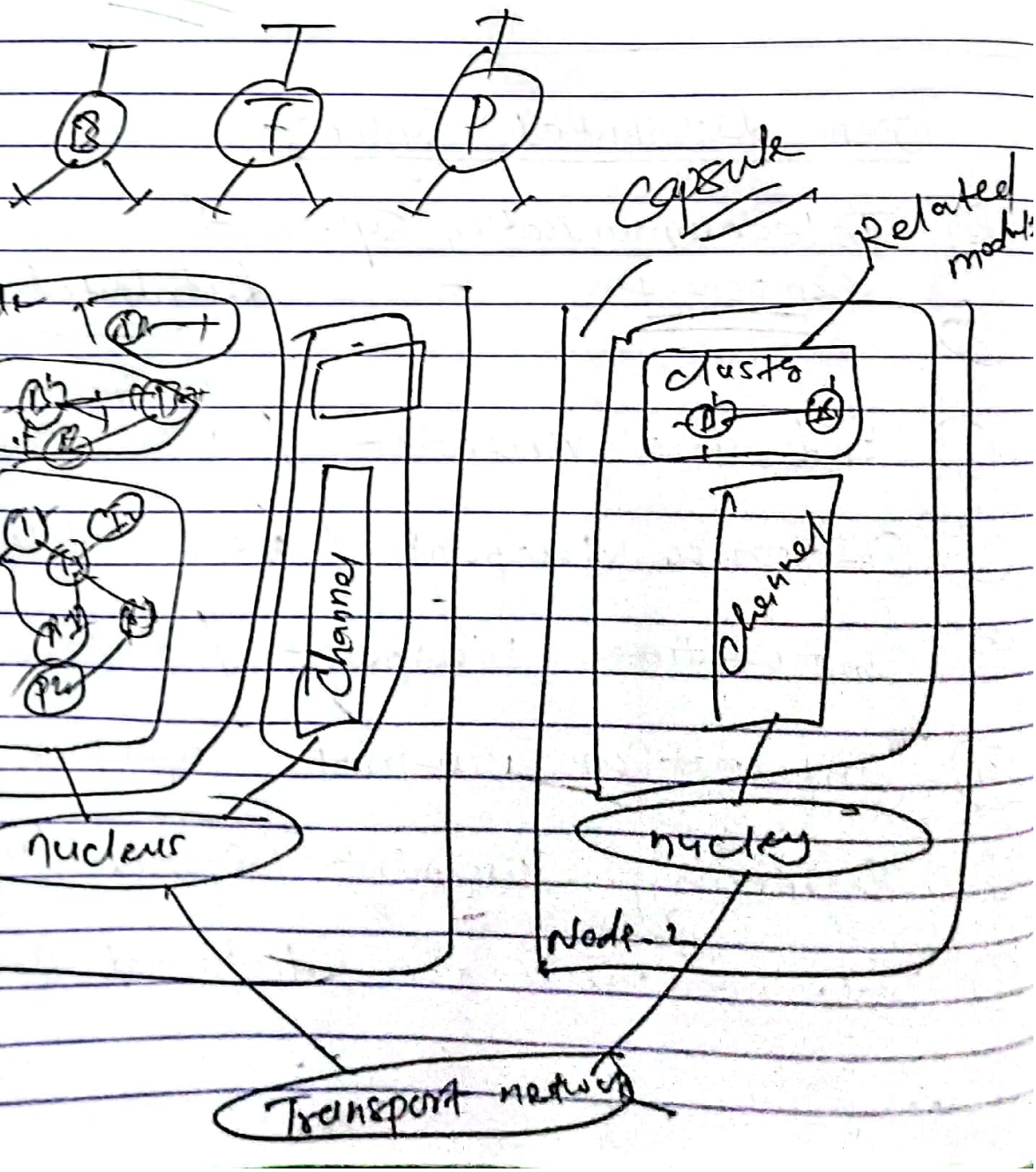
BEO → Basic eng. obj.
TO → Transparence Obj. (Sp. purpose model)
PO → protocol Obj.



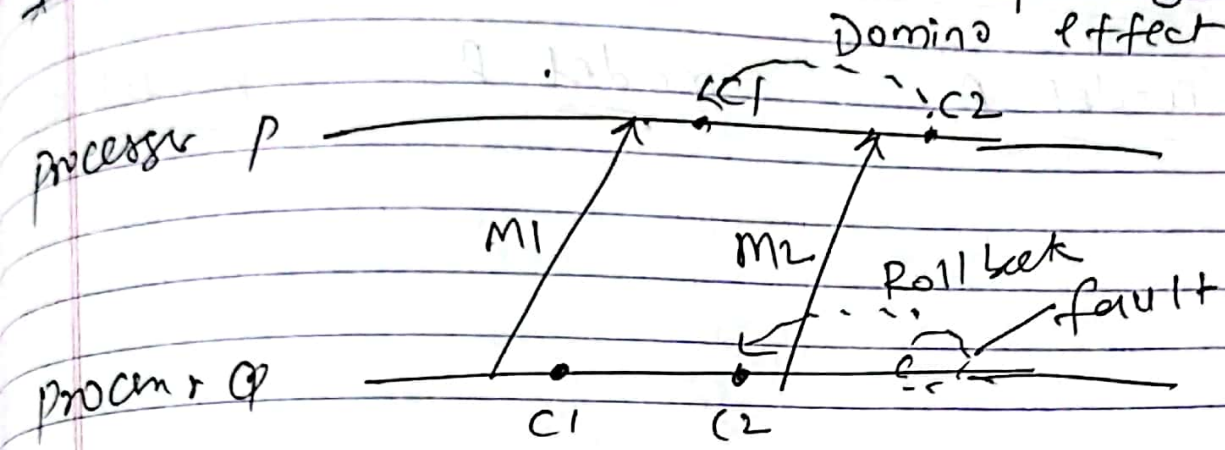
Engineering model;

→ A nucleus is an object that provides access to basic processing storage, and communication functions of a node for use by

- BEO → Basic eng. obj.
- TO → Transparency Obj. (Sp. purpose model)
- PO → protocol Obj.



* Techniques for Consistent Checkpointing



↳ When a fault is detected in one of the processes, the process is rolled back to its last saved checkpoint

↳ In order to maintain consistent checkpoint, some other processes that communicates with this task need to be rolled back, called

'domino effect'