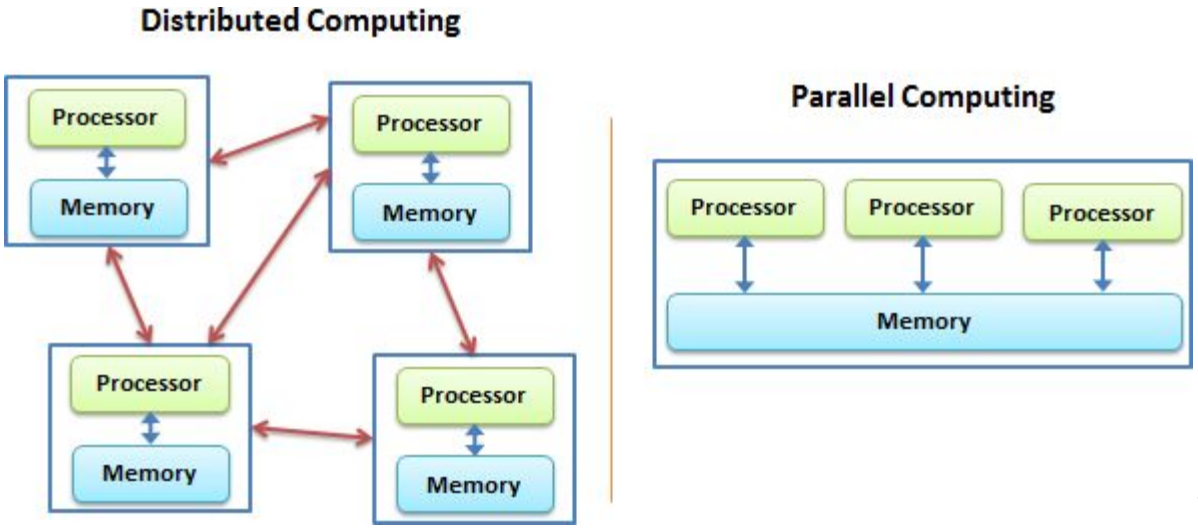
Parallel and Distributed Computing

Unit 1.1 :- PDC , A Perspective

Introduction

- In parallel computing, all processors may have access to a share memory to exchange information between processors
- In distributed computing, each processor has its own private memory, where information is exchanged by passing message between processors (RPC, RMI)



Why do we need parallel computing?

- Serial computing is too slow
- Need for more computing power (Eg:- Data mining, search engine, cryptography)

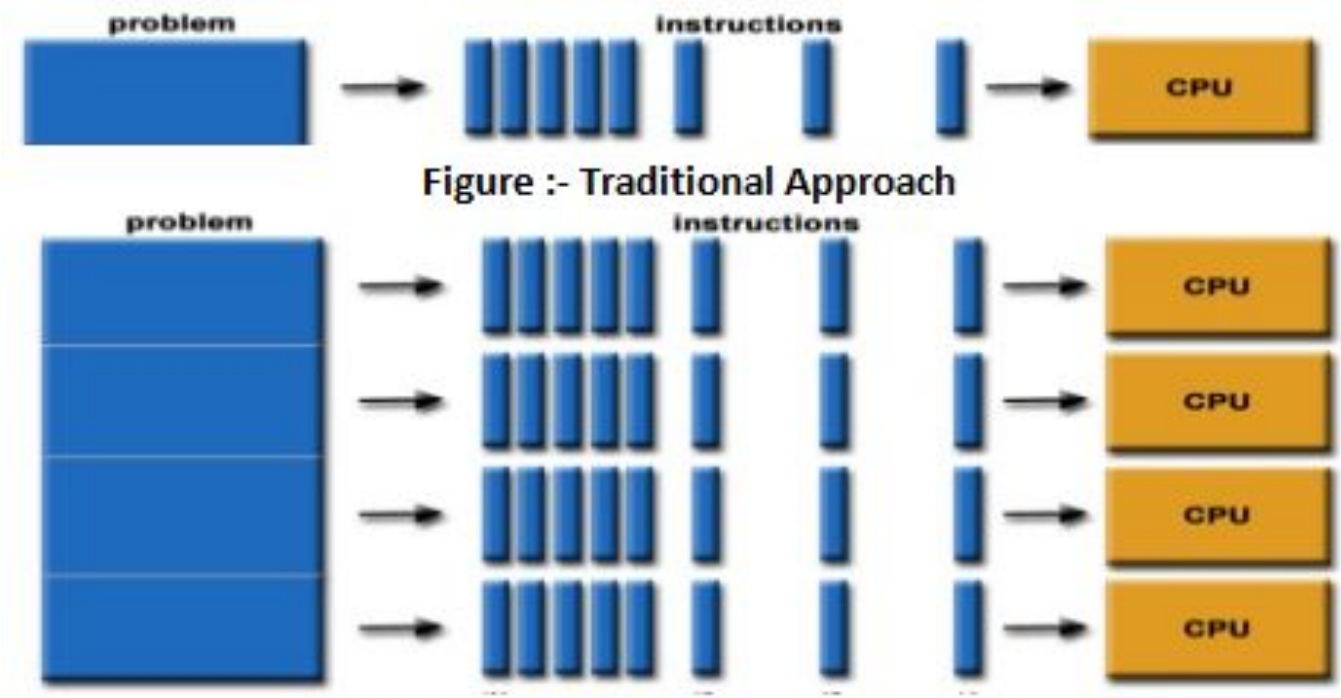
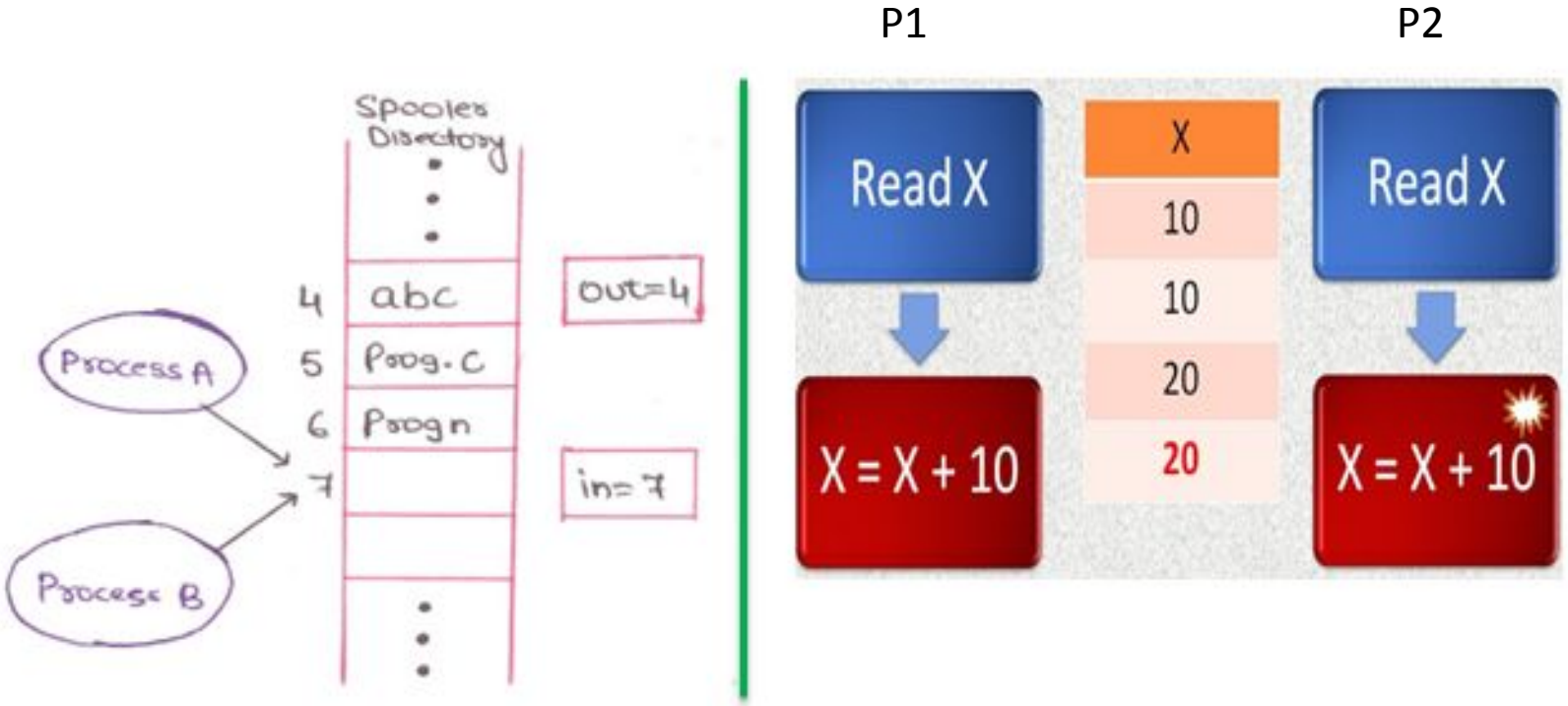


Figure :- Traditional Approach

Figure:- Parallel Computing Approach

Issues on Shared Resources

- Race Condition



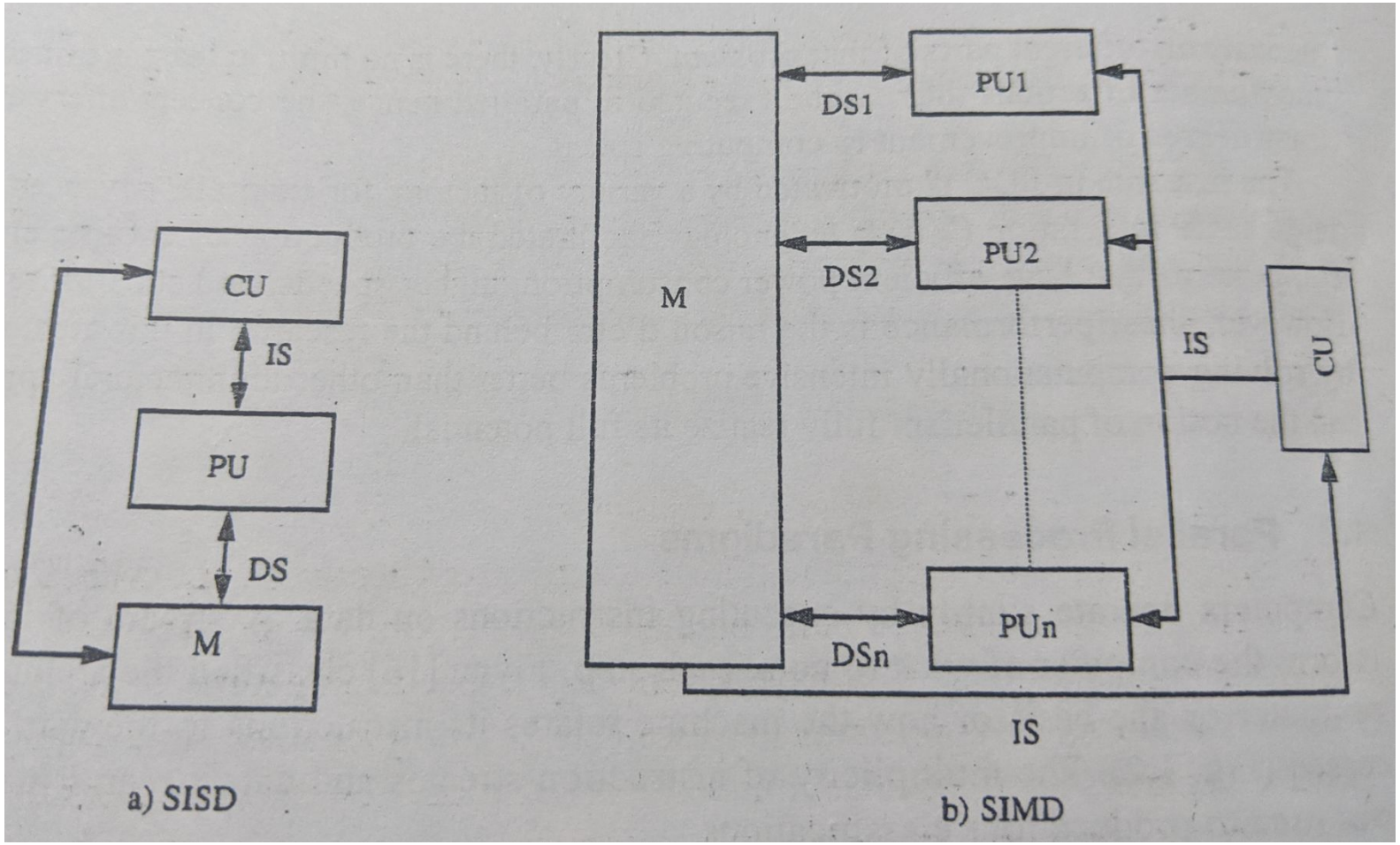
How to avoid Race Condition?

- Mutual Exclusion
  - Interrupt Disabling (Busy Wait)
  - Locked Variable (Busy Wait)
  - Strict Alternation (Busy Wait)
  - Sleep and Wake Up (Busy Wait Solution)

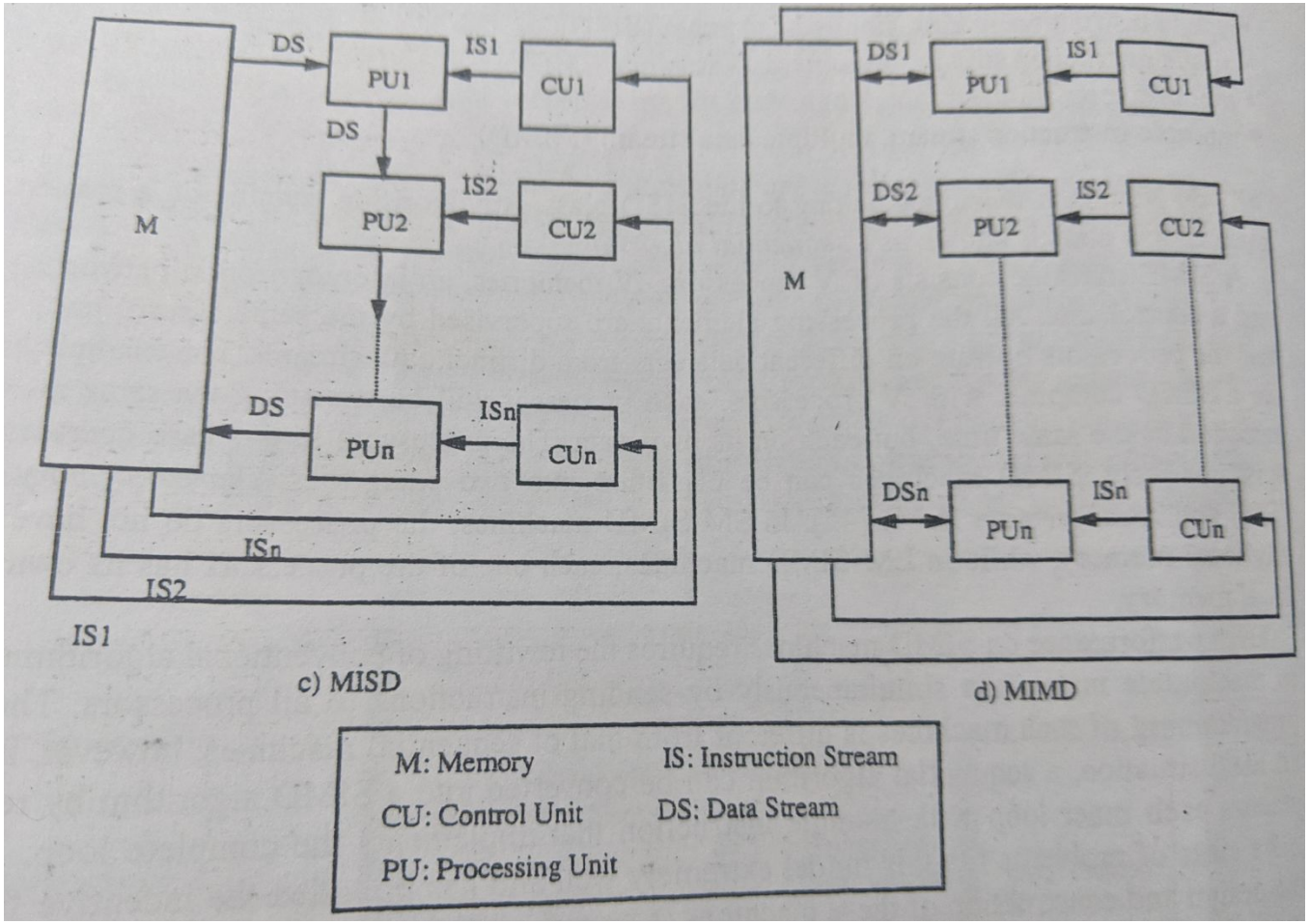
Parallel Processing Paradigms

- Flynn's Taxonomy
  - SISD (Single Instruction Single Data)
  - SIMD (Single Instruction Multiple Data)
  - MISD (Multiple Instruction Single Data)
  - MIMD (Multiple Instruction Multiple Data)

Parallel Processing Paradigms……



Parallel Processing Paradigms……



Issues

- Issues related to parallelization that do not arise in sequential programming
  - Task Allocation
    - Proper sequencing of the tasks when some of them are dependent and cannot be executed simultaneously
    - Load balancing or Scheduling
  - Communication Time
    - Communication time between the processors
    - Serious when the number of processors increases to hundreds or thousands

Performance Evaluation

- Execution Time
  - Time elapsed from the moment the algorithm starts to the moment it terminates
  - In case of multiple processors, it is the time elapse between the time that the first processor begins and the last one terminates
- Speedup
  - $S = \frac{Running\ time\ of\ the\ best\ available\ sequential\ algorithm}{Running\ time\ of\ the\ paralll\ algorithm}$
  - 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, 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)

Performance Evaluation……

- Communication Penalty
  - $CP_i = \frac{E_i}{C_i}$
  - Where $E_i \rightarrow$ total execution time spent by P_i and $C_i \rightarrow$ total time used for communication by P_i

End of Session Unit 1.1

Unit 1.2 :- Semantics of Concurrent Programming

Introduction

- Concurrent programming refers activation of more than one instructions at a time
- In programing language theory, semantics is the field of concerned with the mathematical study of the meaning of the programming language
- Syntactic structure VS Semantic structure
- English Language
  - S + V + O (Grammatical pattern but language dependent)

Categories

- Axiomatic Semantic Definition
- Operational Semantic Definition
- Denotational Semantic Definition

Models of Concurrent Programming

- Level of Granularity
  - Consider two agents (eg bank teller)are accessing a centralized database simultaneously
  - Observe these activities at (deposit = 100)
    - Transaction Level
    - CPU Cycle Level
- Sharing the Clock
  - When several processes operate concurrently, it is important to determine whether they share the same clock (30 **8:30**) p1 , p2 is dependent on p1

Mutual Exclusion

- Sequence of actions where execution is individual
- No two processes are in their critical section
- Once a process starts executing the sequence of actions whose execution us indivisible, it may not be interrupted until the sequence is completed
- Busy wait VS Sleep and Wakeup
- Why Mutual Exclusion

Process P1	**Process P2**
a = getchar()	----------
----------	----------
----------	a = getchar()
----------	a = b
b = a	----------
putchar(b)	----------
----------	putchar(b)

- Here ***a*** is shared variable
- The value that P1 has read is lost

Mutual Admission

- Sequence of actions which must be executed by two processes simultaneously
- If one process is ready and the other is not, it must wait until both are ready
- Eg:-
  - Message Passing
    - Sender (Destination, Message)
    - Receiver (Source, Message)
- Sender and Receiver must be ready for the transfer to occur

Axiomatic Semantic Definition

- 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 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

Axiomatic Semantic Definition……

- Example
  - {x = 5} x = x + 1 {x = 6}

    P S Q

    TRUE After Execution Must Be TRUE
- {j = 3 AND k = 4} j = j + k {j=7 AND k = 4}
- {x ≥ 0} while (x ≠ 0) do x = x – 1 {x = 0}
- {x<0} **while (x ≠ 0) do x = x – 1 infinite** {x = 0}
- ***P ∧ S terminates → Q (after code run)***

Axiomatic Semantic Definition…….

- Example

?

$\{x = x + 1\}$ $x \leq N$

$x + 1 \leq N$

Axiomatic Semantic Definition……

- Partial Correctness
  - A program is partially correct with respect to pre condition an post condition, provided that if the program is started with the values that makes the pre condition true, the resulting value make the post condition true, when the program halts (if ever)
  - i.e. **if answer is returned, it will be correct**
- Total Correctness
  - The program terminates when started with values satisfying the pre condition, the program is called

Axiomatic Semantic Definition…….

- Sequence Statement Rule
  - Has the form
  - i.e. if H_1, H_2,……..,H_n have all been verified then H is also valid
  - Examples

$$\frac{x > 1\{x := x + 1\}x > 2 \qquad x > 2\{x := x + 1\}x > 3}{x > 1\{x := x + 1; x := x + 1\}x > 3}$$

Axiomatic Semantic Definition…….

$$\frac{x > 1\{x := x + 1\}x > 2 \qquad x > 0\{x := -x\}x < 0}{x > 1\{x := x + 1; x := -x\} \quad ?}$$

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

$$\frac{x > 1\{x := x + 1\}x > 2 \qquad x > 2 \rightarrow x > 0 \qquad x > 0\{x := -x\}x < 0}{x > 1\{x := x + 1; x := -x\}x < 0}$$

Axiomatic Semantic Definition……

- Alternation Statement Rule
  - **IF –THEN**

    {P and B} S {Q} P if(con) {s1} else s2.Q

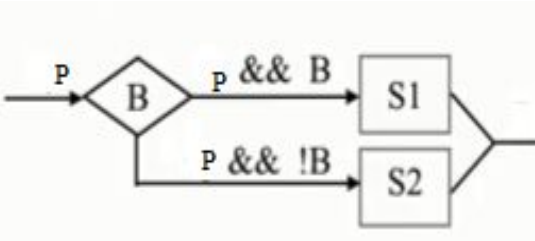
    {P and NOT B} ⊃ {Q}

    ∴{P} IF B THEN S END IF {Q}
  - **IF – ELSE**

    {P and B} S1 {Q}

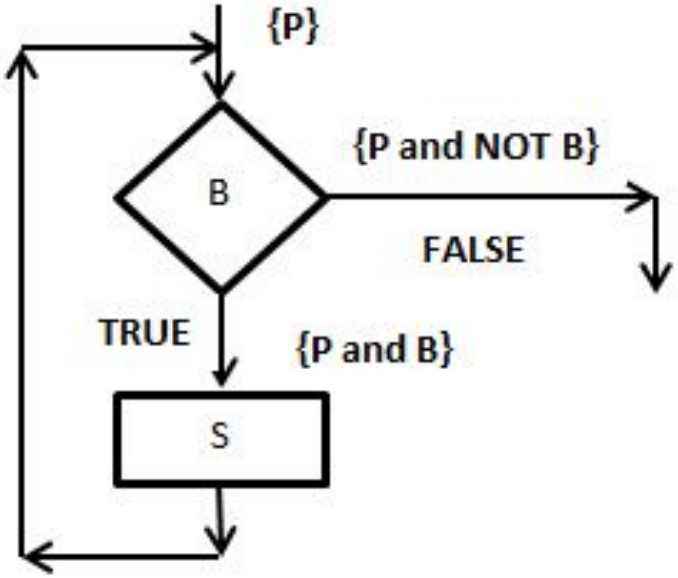
    {P and NOT B} S2 {Q}

    ∴{P} IF B THEN S1 ELSE S2 END IF {Q}



Axiomatic Semantic Definition…….

- LOOP
  - **WHILE**
    - $$\frac{\{P \text{ and } B\}S\{P\}}{\therefore\{P\}\text{while } B \text{ do } S \text{ END while}\{P \text{ and NOT } B\}}$$



Axiomatic Semantic Definition…….

- LOOP (Example)

$$\frac{x > 0 \wedge x < 10\{x := x + 1\}x > 0}{x > 0\{\textbf{while}\ x < 10\ \textbf{do}\ x := x + 1\}x \geq 10 \wedge x > 0}$$

X = 10…… x>=10

Axiomatic Semantic Definition……

- Disjoint Parallel Program
  - Two programs S_1 and S_2 are said to be disjoint if none of them can change the variables accessed by the other
  - Denoted as $[S_1 \,||\, S_2]$
  - Example
    - $S_1 : x = z$ $\quad S_1: x = y + 1$
    - $S_2 : y = z$ $\quad S_2: x = z$
  - Semantics of this statement can be denoted as

$$\{P_1\} \quad S_1 \quad \{Q_1\}$$
$$\{P_2\} \quad S_2 \quad \{Q_2\}$$

Axiomatic Semantic Definition…….

- 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
  - **Example 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 code that sends the email and has to await until the balance < 1000**
  - Semantically, it can be denoted as

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

Operational Semantics

- 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
- Example
  - To execute x = y, first determine the value of y, and assign it to x
- Uses the notation <P, S>, means semantics of

Operational Semantics……

- So before starting the program, the state of the memory is [x → 5, y → 7, z → 0], and after the program has terminated, it is [x → 7, y → 5, z → 5], which is meanin
  x=y, y=z in the state **[x –**
- **Syntax directed trans**
  represented as operatio
- Eg

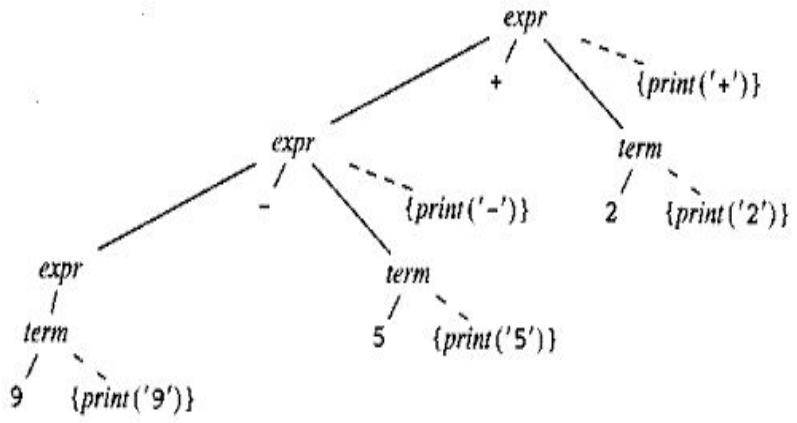


Fig. ... Actions translating 9-5+2 into 95-2+.

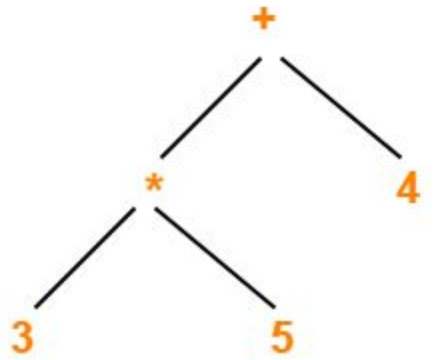
Production	**Semantic Rules**
expr → expr + term	*{print ('+')}*

Operational Semantics……

- **Hennessy Milner Logic**
  - Used to specify properties of a labeled transition system
  - Introduced by Mathew Hennessy and Rabin Milner
  - Syntax
    - $\varphi \rightarrow tt$ {always true}
    - $\varphi \rightarrow ff$ {always false}
    - $\varphi \rightarrow \varphi_1 \wedge \varphi_2$ {conjunction}
    - $\varphi \rightarrow \varphi_1 \vee \varphi_2$ {disjunction}
    - $\varphi \rightarrow [L]\varphi$ {after ***every*** execution of event L, the result has property φ}

Denotational Semantic Definitions

- Each construct of the language is mapped into a mathematical object that defines its meaning
- Idea of denotational semantics is to associate an appropriate mathematical object with each phrase of the language
- Example
  - Syntax tree for 3*5+4



Denotational Semantic Definitions……

- Traditionally, the emphatic bracket (), is used to represent the denotational semantics definition
- Eg:

  E1 + E2 = plus (Evaluate Evaluate)

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

  Eg:

  fact(n) = if(n==0) then 1 else n*fact(n-1)

  can be viewed as in ordered pair for denotational

Denotational Semantic Definitions……

- Example (denotational specification of language of non negative integer number)

  Number → Digit | Number Digit

  Digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

  Value $[\![ND]\!] = plus(times(10, value[\![N]\!], digit[\![D]\!])$

  value$[\![D]\!] = digit[\![D]\!]$

  digit$[\![0]\!] = 0$

  digit$[\![1]\!] = 1$

  ……

  digit$[\![9]\!] = 9$

- Eg
- $value[\![65]\!] = plus(times(10, value[\![6]\!]), digit[\![5]\!])$

  $= plus(times(10, digit[\![6]\!]), digit[\![5]\!])$

  $= plus(times(10, 6), digit[\![5]\!])$

  $= plus(60, digit[\![5]\!])$

  $= plus(60,5)$

  $= 65$

Denotational Semantic Definitions……

- **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 engages 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

Denotational Semantic Definitions……

- **Communicating Sequential Process**
- Example
  - A simple vending machine that serves a single customer then stop can be represented as,
    1. $VM_0 \stackrel{\text{def}}{=} (coin \rightarrow (candy \rightarrow stop))$
    2. $VM_1 \stackrel{\text{def}}{=} (coin \rightarrow (candy \rightarrow VM_1))$
    3. $VM_2 \stackrel{\text{def}}{=} (coin \rightarrow (candy \rightarrow VM_2)) || (notebill \rightarrow (toffee \rightarrow VM_2))$

End of Session
Unit 1.2

Unit 1.3 :- Process Algebra

Process Algebra

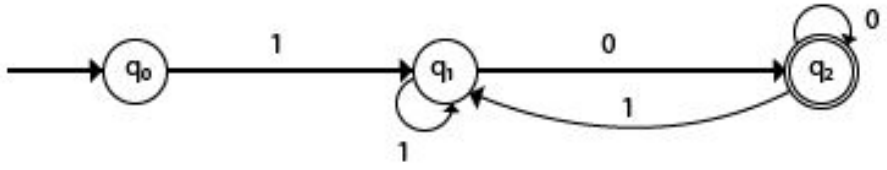
- System behavior generally consists of process and data
- Process are the control mechanism for the manipulation of the data
- Process are dynamic and active, while the data are static and passive
- System behavior tends to be composed of several processes that are executed concurrently, where those processes exchange data in order to influence each other's behavior

Process Algebra.....

- Example

  DFA (Deterministic Finite Automata)

  1(0 + 1)*0 → Process Algebra



Process Graph

Process Algebra.....

- Is a labeled transition system in which one state is selected as a root state
- If the labeled transition system contains an edge $S \xrightarrow{a} S'$, then the process graph can evolve from state S to S' by the execution of action (event) ***a***
- For the mathematical reasoning, process graph are expressed algebraically called process algebra
- So process algebra is a mathematical framework in which system behavior is expressed in the form of algebraic terms
- Process algebra studies two types of action
  - Observable Action
  - Unobservable Action

Process Algebra……

- **Observable Action**
  - Can occur within a process, if the process is ready to perform the action and some controlling environments allows it to occur
  - Explicit
- **Unobservable Action**
  - Occur without any event (explicit)
  - Implicit
  - Denoted by τ (tow)

Process Algebra……

- **Example memory = 2items q0, q1(1+1), q2(2-1)**

- The buffer is operated through a two place buffer
- **IN** → putting one item in the buffer
- **OUT** → taking one item from the buffer
- It is impossible to put one item into a full buffer and impossible to take one item from empty buffer
- Initially buffer is empty, i.e. only one operation (IN) is possible

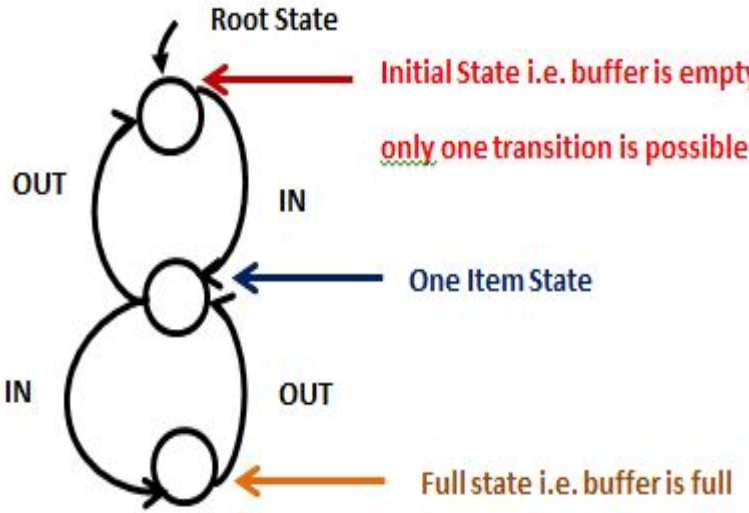


Figure: - Transition System representing a two place buffer

Process Algebra……

- **Example m1 (1), m2(1) int a[2] first int b[1],c[1] second**

- Q0 b-emp c-emp
- Q1 b-ful c-emp
- Q2 b-em c-ful
- Q3 b-ful c-full

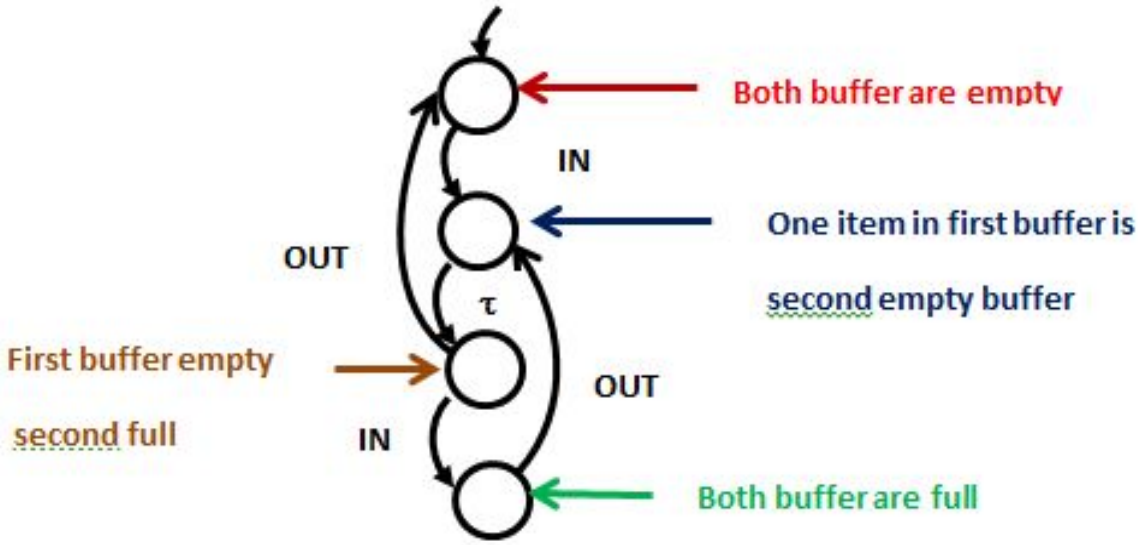


Figure: - Transition System representing a two one place buffer

Transition System

- Formally it can be defined as , A' = (A, S, $\rightarrow$, S_0), where

  A : alphabet

  S : set of states

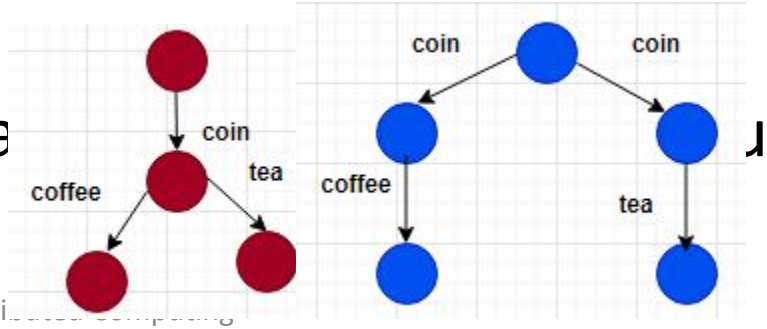
  $\rightarrow$: transition $\quad S \times (A \cup \{\tau\}) \rightarrow S$

  S_0 : initial (root) state

- **Conventions**
  - a, b, c, …….are used to indicate elements of A
  - u, v, ……are used to indicate $A \cup \{\tau\}$
  - w is used to indicate $(A \cup \{\tau\})^*$ aauvv-w
  - ow is used to indicate A*……abbbbccaaa1b1
  - Example
  - $S \xrightarrow{w} S'$ if there exists $S = S1 \xrightarrow{u1} S2 \xrightarrow{u2} S3 \rightarrow \cdots \xrightarrow{un} Sn = S'$

Transition System……

- Reachable States
  - Let A' = (A, S, $\rightarrow$, S_0) be a transition system
  - Then s' $\in$ S is a reachable state of A' if there exists $w = u_1 u_2 \ldots\ldots u_n \in (A \cup \{\tau\})^*$ such that
    w
- $S_0 \longrightarrow s'$
  - Set of all reachable states at A' is denoted by reach(A')
  - Transition system can a… …ugh graphs
  - Example



Observation Bisimilarity

- Let $A_i' = (A_i, S_i, \rightarrow_i, S_{0i})$, i = 1, 2 be two transition systems
- A_1' and A_2' are observation bisimilar, denoted by $A_1' \cong A_2'$ if there is a relation $B \subseteq S_1 \times S_2$ such that,
  - $(S_{01}, S_{02}) \in B$
  - If $(S_1, S_2) \in B$ and then there is S_2' such that $(S_1', S_2') \in B$ and
  - If $(S_1, S_2) \in B$ andthen there is S_1' such that $(S_1', S_2') \in B$ and

Observation Bisimilarity……

- Example
  - Consider the following two processes

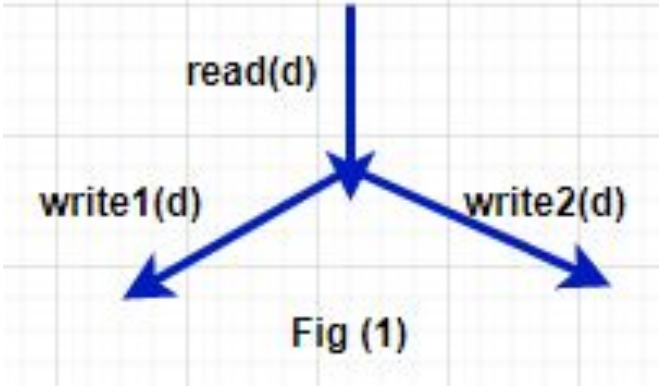


Fig (1)

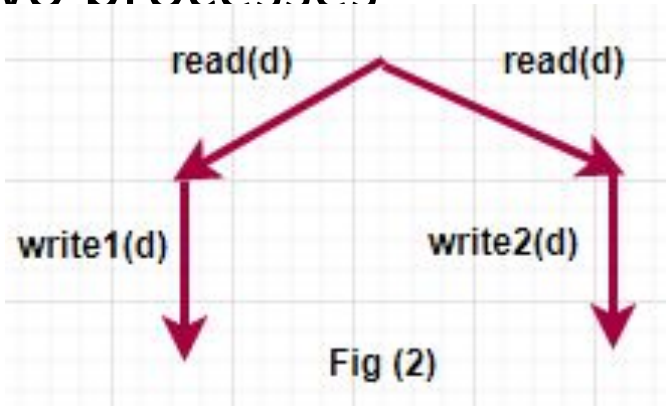


Fig (2)

  - In fig(1), the process read the datum d and decides whether

Observation Bisimilarity……

- Example
  - Here they are traced equivalent but not are bisimilar because
    - In fig(1), in case of crash of disc 1 after executing read(d) and write1(d), there is a second chance to write on disc 2 using write2(d)
    - But in fig(2), in case the disc 1 crashed after executing read(d) and write1(d) then it doesn't have any option to write the datum

Process Terms

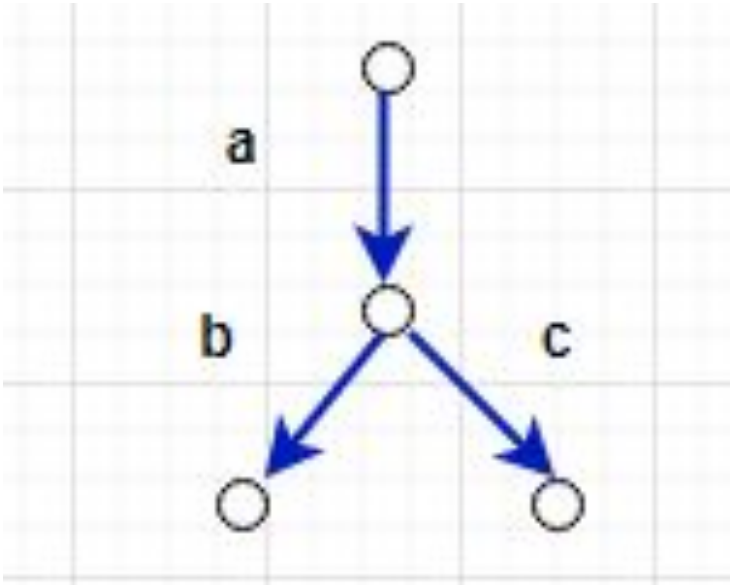
- Let ***act*** be defined as A $\cup$ {τ}
- The set of process terms, ***proc***, whose element will be denoted by P, Q, R and are described as

1. P $\rightarrow$ a.P (***Prefix***) $\Rightarrow$ denotes a process that behaves like P after executing event ***a***
2. P $\rightarrow$ stop{A} (***Deadlock***) $\Rightarrow$ denotes a process which doesn't perform any action
3. P $\rightarrow$ P + Q (***Choice***) $\Rightarrow$ denotes a process that behaves like either P or Q
4. P $\rightarrow$ P || Q (***Parallelism***) $\Rightarrow$ where P and Q

Process

Terms.....

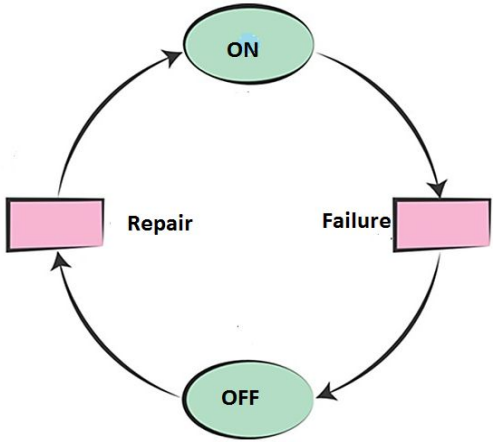
- **Example**



- **S = a.{b. stop{a,b,c} + c.stop{a,b,c}}**

Petri Nets

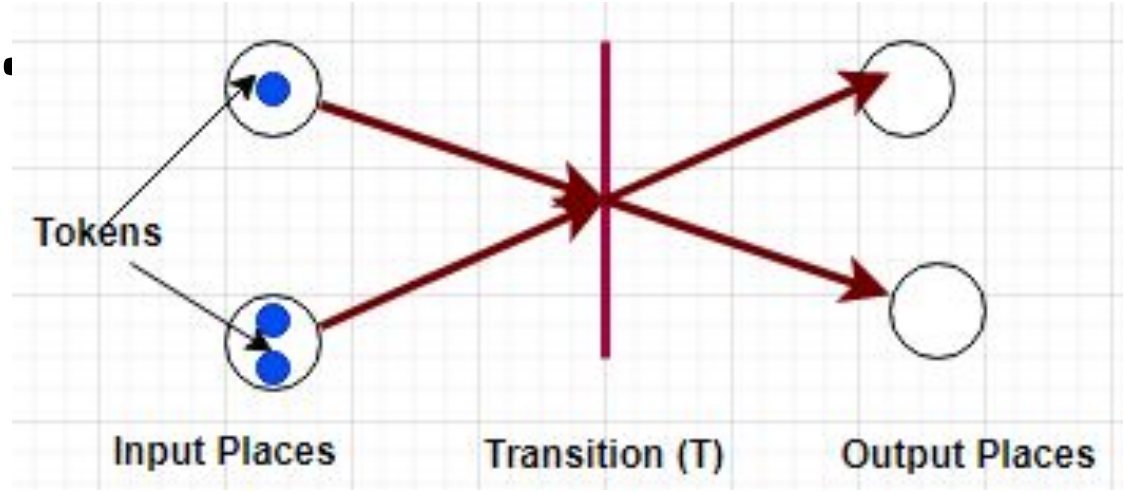
- Developed by Carl Adam Petri in 1962 as his PhD research
- Is the mathematical modeling language for the description of the system
- i.e. used to model the functionality behavior of the system
- Abstract model of information flow



Petri Nets……..

- 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 number (possibly zero) of tokens
- Token circulates in the system between places

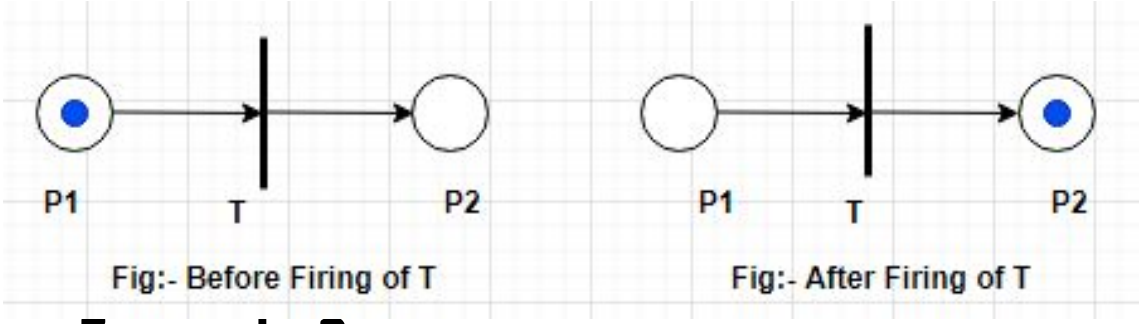
Petri Nets.....



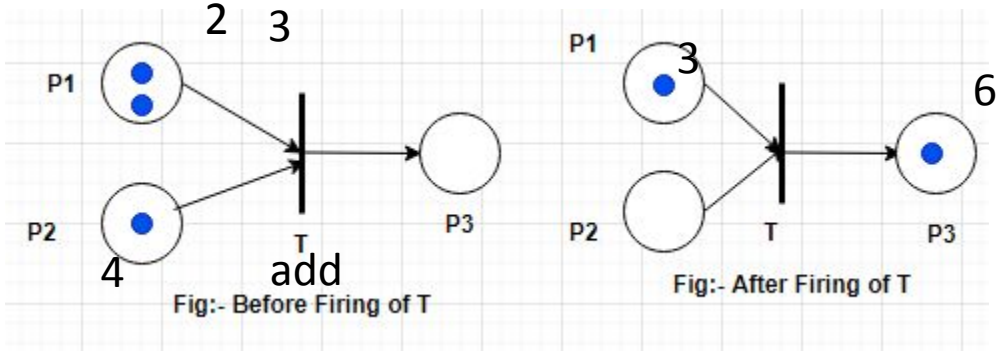
- A transition is called enabled, if for every arc from a place P to transition T, there exists a distinct token in the marking

Petri Nets.....

- Input Places → Pre condition
- Output Places → Post condition
- **Example 1**



- **Example 2**



Petri Nets.....

- **Example 3**

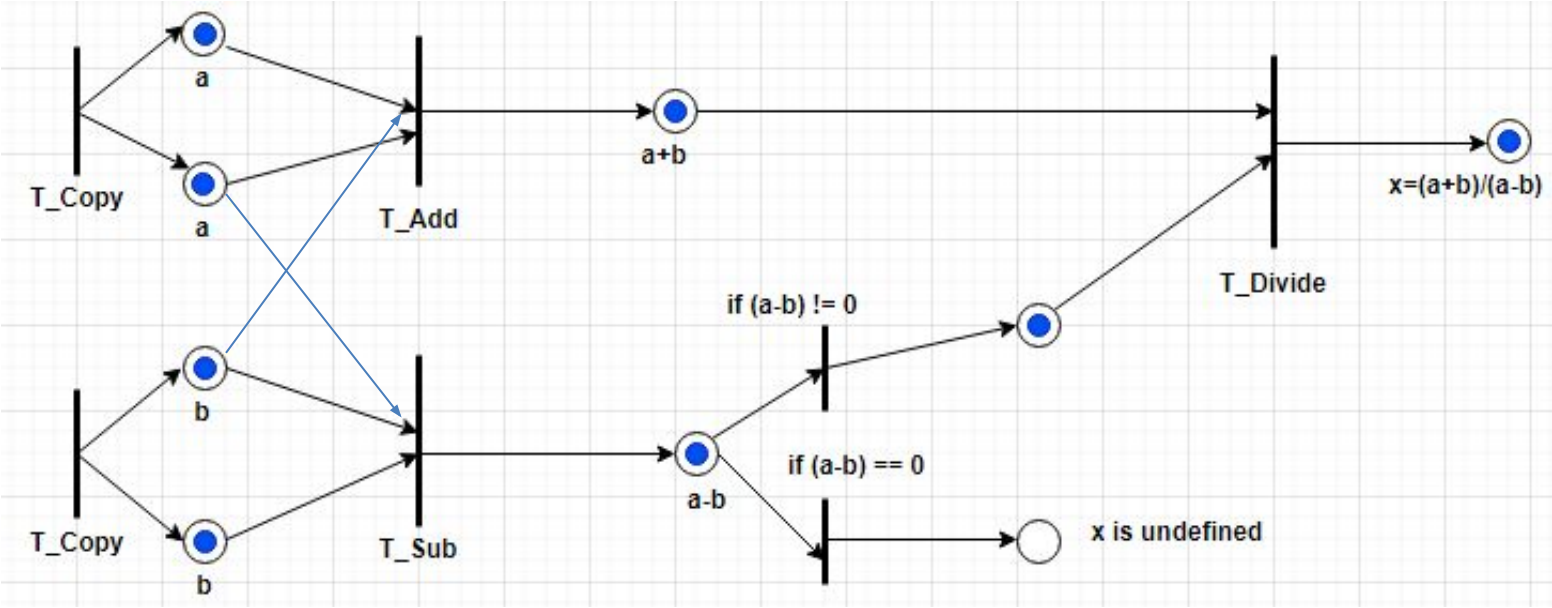


Fig :- Petri Net showing a data flow computation of $x = \frac{a+b}{a-b}$

Petri Nets.....

- **Example 4**

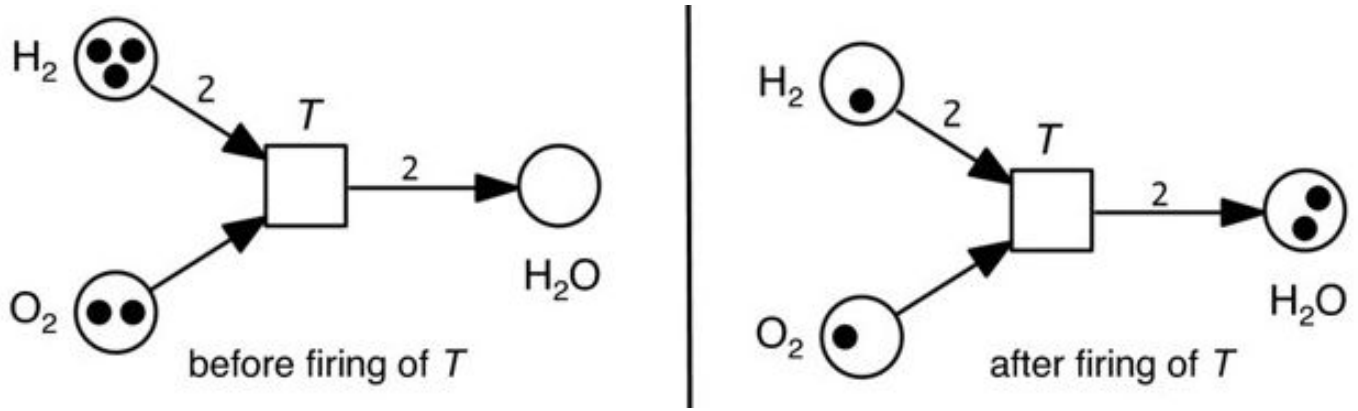


Fig :- Petri Net showing an equation $2H_2 + O_2 \rightarrow 2H_2O$

Petri Nets.....

- Formal Definition
  - PN = (P, T, I, O, M_{t0}) where
    - P → set of places = {P_1, P_2, …….,P_m}
    - T → set of transitions = {T_1, T_2, …….T_n}
    - I → is an input matrix on n × m
    - O → is an output matrix of n × m
    - M_{t0} → represents initial marking of machine

    Note:-

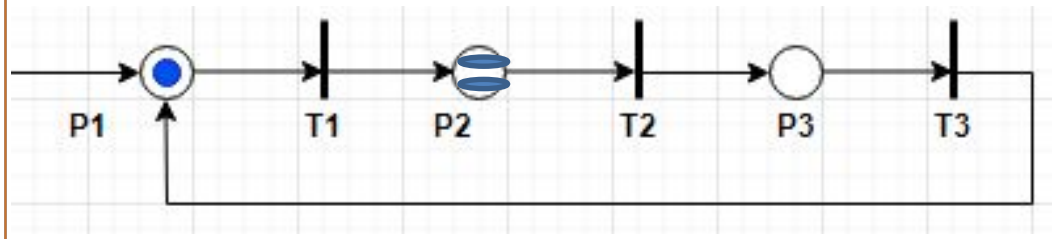
    A marking of a PN at time t,

    M(t) = {M_1(t), M_2(t),…….M_m(t)}

    where M_i(t) = no. of tokens in place i at time t)

Petri Nets.....

- **Formal Definition**

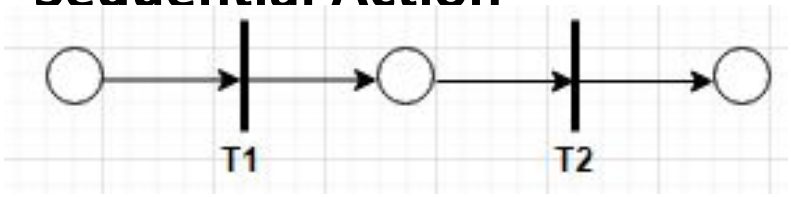


- P = {P1, P2, P3}
- T = {T1, T2, T3}
- $I = \begin{matrix} \\ T1 \\ T2 \\ T3 \end{matrix} \begin{matrix} \begin{matrix} P1 & P2 & P3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{matrix}$ $O = \begin{matrix} \\ T1 \\ T2 \\ T3 \end{matrix} \begin{matrix} \begin{matrix} P1 & P2 & P3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix}$
- $M_{t0} = (1 \quad 0 \quad 0)$ *no. of tokens*

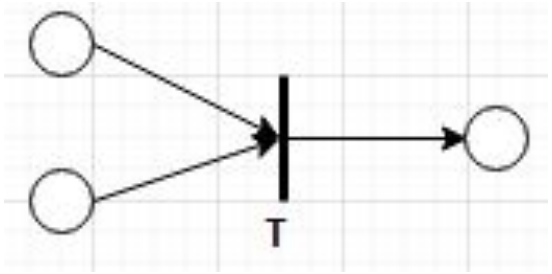
P1 P2 P3

Variations in PN

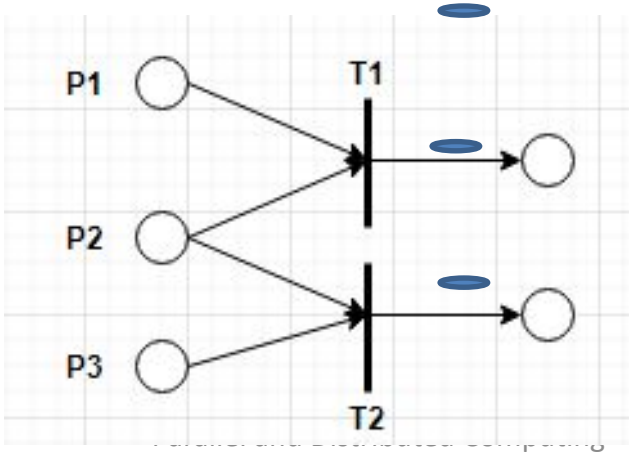
1. **Sequential Action**



2. **Dependency**



3. **Conflict**

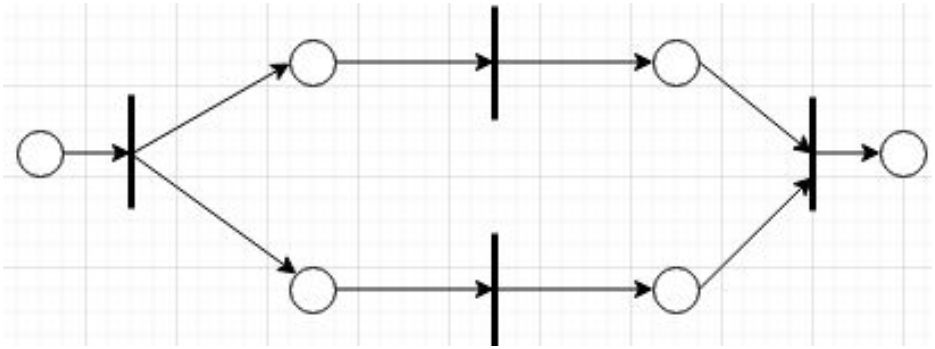


Variations in PN……

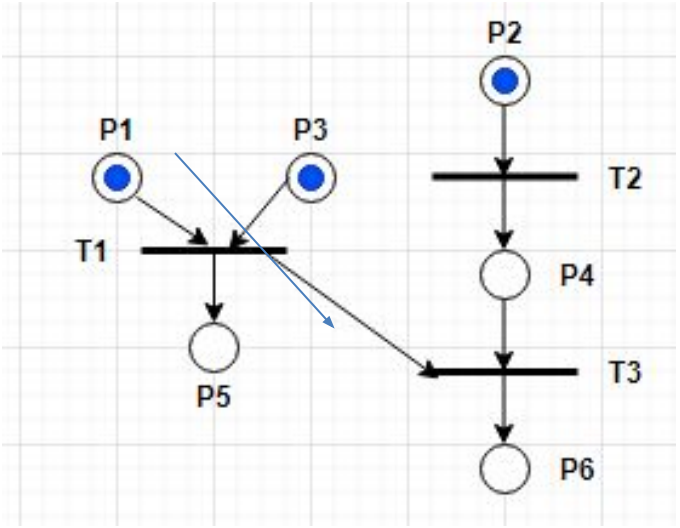
- **In Case of Conflict**
  - P1 and P2 have tokens but not P3 then T1 is fired
  - P2 and P3 have tokens but not P1 then T2 is fired
  - If all have tokens which one to fire then conflict (***Solution may be priority***)

Variations in PN……

- 4. **Concurrency**



- 5. **Confusion**
  - **(Conflict+ Concurrency)**



Variations in PN……

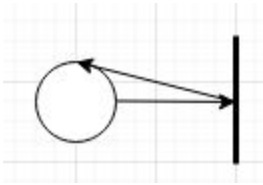
- **In case of confusion**
  - Two possible interpretation
  1. T1 occurred first without being in conflict with other events then T2 occurred
  2. T2 occurred first and then T1, T3 got in conflict
     - Conflict (T1, C1) = φ where C1 = {P1, P2, P3}
     - Conflict (T1, C2) = {T3} where C2 = {P1, P3, P4}

Pure and Simple Net

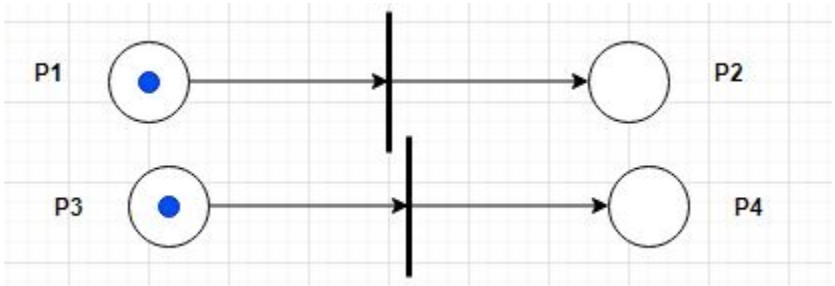
- Let PN = (P, T, F) p t l o m

1. PN is pure if $\forall x \in X : {\bullet x} \cap x^{\bullet} = \varphi$

   (***No Self LOOP***)



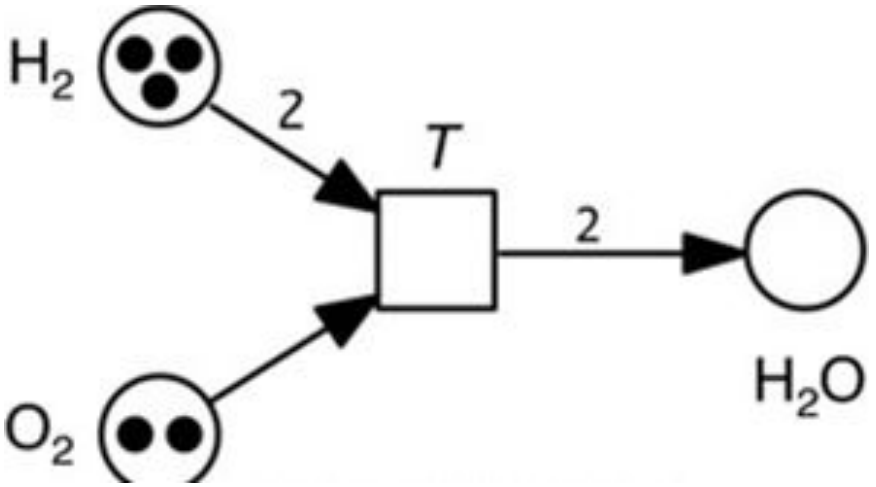
1. PN is simple if $\forall x, y \in X : [({\bullet x} = {\bullet y}$ and $x^{\bullet} = y^{\bullet}) \Rightarrow x = y]$



Place / Transition Net (P/T Net)

- Defined by 4 tuples (P, T, F, W) where (P, T, F) is a Petri Net and W is a weight function
- Example

4



Place / Transition Net (P/T Net)……

- Transition Rule
  - Let M = (P, T, F, W, M_0) be a place transition system

1. A transition t ∈ T is enabled at M

   M [t > if ∀p ∈ P, W(p, t) ≤ M(s)

   Eg:-

   M_0 [t_1 > M_1 m0= 2, 1, 0 t1=t m1=1, 0, 1

2. A transition t enabled at M may fire yielding a new marking M', M [t > M', such that ∀p ∈ P

   M'(p) = M(p) – W(p, t) + W(t, p)

Assignment

1. Design a Petri Net to simulate the behavior of the following finite state machine
   - A vender machine that accepts 5 and 10 cents.
   - Sell candy bars worth 15 or 20 cents
   - Maximum hold up coins = 20 cents

Properties of Petri Nets

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

Properties of Petri Nets……

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

    $M' [\sigma > M$

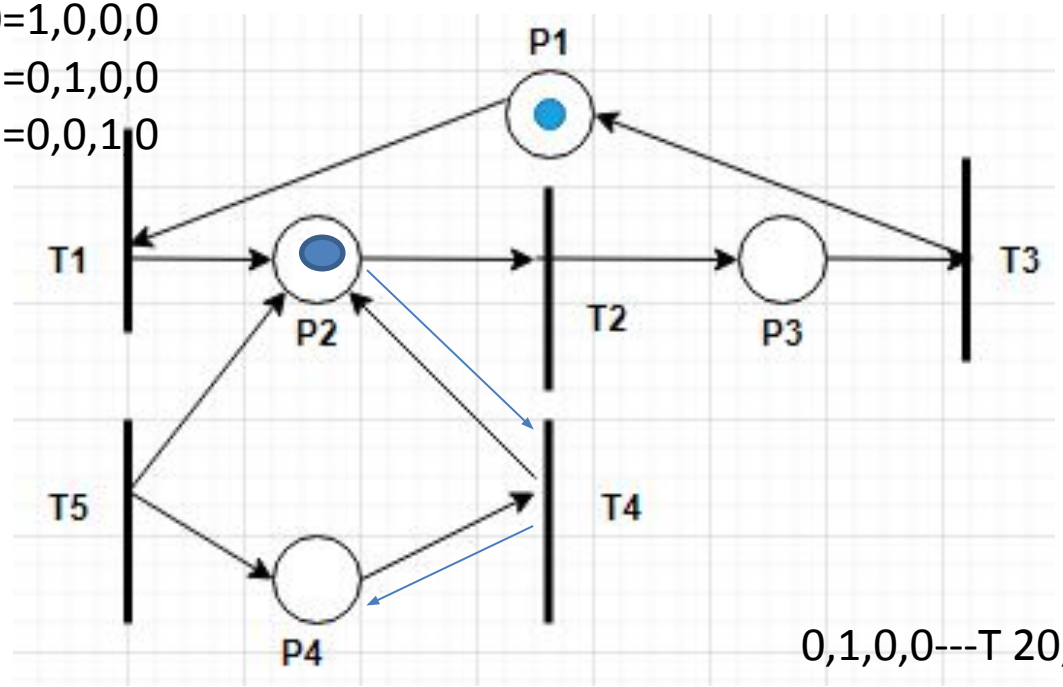
    i.e. $M' t_1 M'' t_2 M''' t_3 \ldots\ldots\ldots t_n M$, where $\sigma = t_1 t_2 \ldots t_n$
  - $R(M_0)$ → set of markings, reachable from the initial marking M_0

Properties of Petri Nets……

M1=(0,1,0,0)

- **Reachability (Example)**

M0=1,0,0,0
M1=0,1,0,0
M2=0,0,1,0



$$M_0 = \begin{pmatrix} P1 & P2 & P3 & P4 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

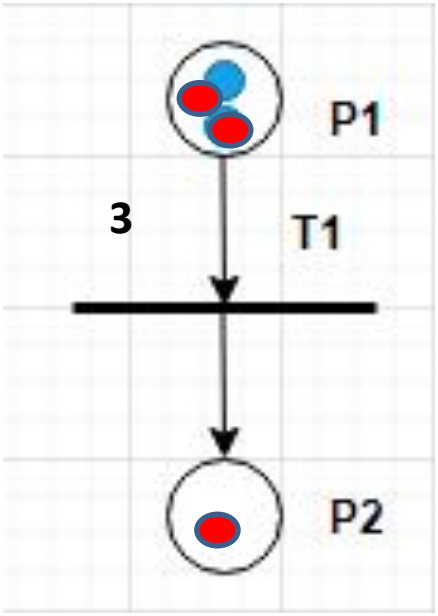
$R(M_0) = \{(0,1,0,0),$
$(0,0,1,0),$
$(0,0,0,1)\}$

Quiz:-
(1,0,1,0) → not Reachable

0,1,0,0---T 20,0,1,0---T3 1,0,0,0

Properties of Petri Nets……

- **Reachability (Example)**



M_0 =(2,0)
$R(M_0)$ = (1,1) (0,2)

(2,0) [T1 > (1,1)

Man
woman

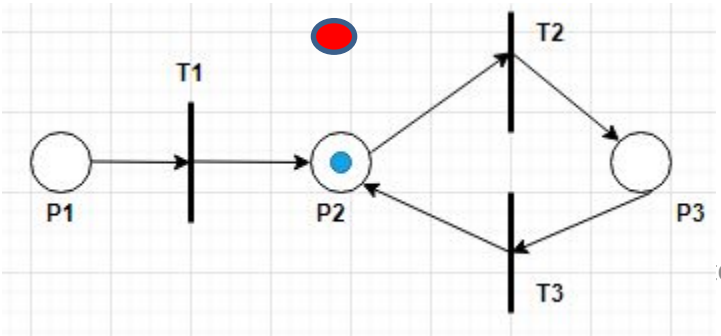
Properties of Petri Nets……

- **Boundness**
  - No. 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 $k > 0$

Properties of Petri Nets……

- **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 >**
  - A transition is deadlocked if it can never fire
  - A transition is live if it can never deadlock
  - A petri net is live if all its transitions are live
  - A transition is said quasi live if it can be fired at only once



T1 → Dead
T2 → Live
T3 → ?????

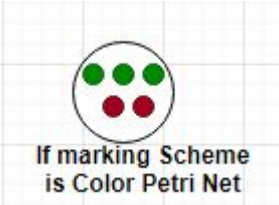
Properties of Petri Nets……

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

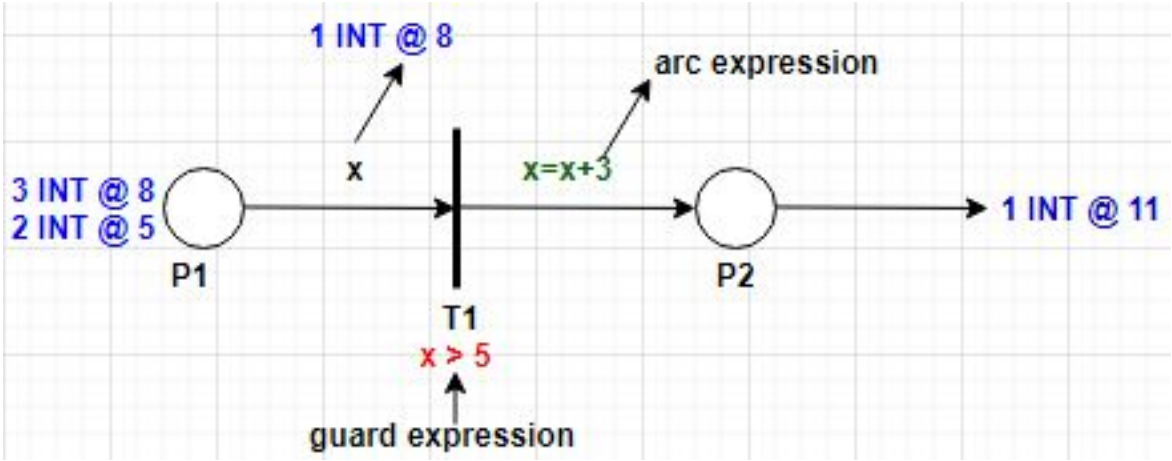
    i.e. $M_0 \in R(M), \forall M \in R(M_0)$

High Level Petri Net

- Consists of the following features
  1. Petri net
  2. Marking Scheme (Like ***Color Petrinet***)
  3. Transition Guard
  4. Arc Expression



Example



End of Session Unit 1.3

Unit 1.4 :- Complexity Theory

Complexity Theory

- Estimates the amount of computational resources (time and storage) need to solve a problem
- **Basic Terms**

1. **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
2. **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 resources prescribed by the writer of the algorithm
3. **Refinement of the measures of parallel computation**
   - How fast can we solve the problem, when a large number of

Turing Machine as a basis of consequences

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 for resources

Simplicity of abstraction

- Simple model of an abstract machine that consists of three components
  - Finite state machine
  - A read/write head
  - Infinite tape memory
- Mathematically Turing machine can be defined by

  TM = (Q, Σ, Γ, δ, q_0, B, F), where

  Q $\rightarrow$ set of states

  Σ $\rightarrow$ set of input alphabet

  Γ $\rightarrow$ set of tape symbols

  δ $\rightarrow$ transition function

  q_0 $\rightarrow$ starting state

Dual nature of TM

- Universal TM
- TM whose input consists of program and a data set for the program to process
- Specialty (an simulate any special purpose algorithm) and universality (can simulate any other TM)

Inclusion of non – determinism

- NDTM → alternative moves
- Branching tree called OR tree
- A computation halts, if there is a sequence that leads to an accepting state
- Rejects an input if and only if there is no possible sequence of moves

Realization of unbounded parallelism

- NDTM can be simulated using DTMs by allowing unbounded parallelism in computation
- Each time a choice is made, the DTM makes several copies of itself
- One copy is made for each possible choice
- Thus many copies of DTM are executing at the same time at different nodes in the tree
- The copy of DTM that reaches a solution early halts all other DTM, if a copy fails, it only

Provision of an easy abstract measure for resources

- **DTM** → by counting each moves as one unit time and each cell as scanned as one space time
- **NDTM** → complexity is the number of moves required for an accepting sequence of moves since there are accepting sequence

ATM (Alternating Turing Machine)

- Consists of an infinite memory tape and a finite state control as in TM
- States of ATM are partitioned into normal, existential and universal states

1. **Universal State** → it leads to an acceptance, if and only if all of its descendants lead to an accepting state i.e. AND tree
2. **Existential State** → a computation is accepting if and only if at least one of

Satisfiabilty Problem

- Let S = (x1 ∨ x2) ∧ (x3 ∨ x4 ∨ x5) ∧……
- Is there an assignment to each literal such that the sentence is true
- Example
- Let E be an CNF (Conjunctive Normal Form)

  E = (x ∨ y ∨ z) ∧ (x ∨ ¬y ∨ ¬w) ∧ (¬z ∨ w) ∧ (¬x)
- Satisfiable under x = z = w = F and y = T
- But **E = (x ∨ y) ∧ (¬x) ∧ (¬y)** **??????**

Parallel Complexity Models

- Synchronous parallel computation can be used to solve problems like matrix multiplication
- In a synchronous parallel system, it is easy to measure the time in terms of clock cycle
- It is because all identical processors work under the control of centralized clock

VLSI Computational Complexity

- Practical use of algorithm depends upon the design of VLSI
- Because the manner in which the signal propagates, influences the algorithm
- Two fundamental parameters deciding the efficiency of VLSI chip is

1. Area of circuit (Space complexity)
2. Time taken to provide an output for a given input (Time complexity)

End of Session
Unit 1.4

Unit 1.5 :- Distributed Computing

Distributed Computing

- Major obstacles in distributed computing is uncertainty due to differing processor speed, and occasional failure of components
- A system consists of n processors $P_1, P_2, \ldots\ldots, P_n$ and the network net
- Each processor P_i is modeled as state machine with state sets Q_i
- The state set Q_i, contains a distinguished initial state $q_{0,i}$
- Each processor P_i, contains $buff_i$, in which

Leader Election

- In distributed computing, leader election is the process of designating a single processor as the leader of some task distributed among several computer
- The existence of leader can simplify co-ordination among the processors and is helpful in achieving fault tolerance and saving resources
- Election Algorithm
  - Bully Algorithm

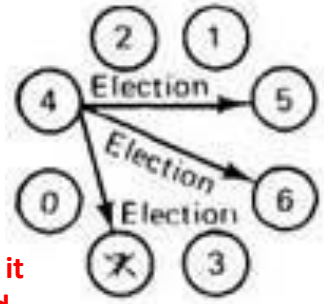
Bully Algorithm

- 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 IDs and address of every other processes
  - Communication is assumed reliable
  - Process initiates election if the coordinator failed

Bully Algorithm.....

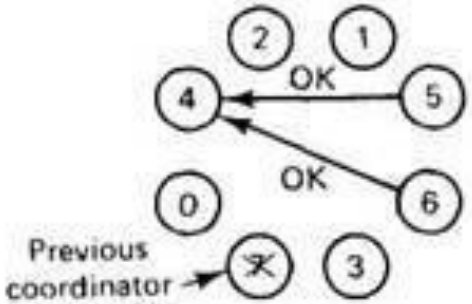
The job of 4 is done

....

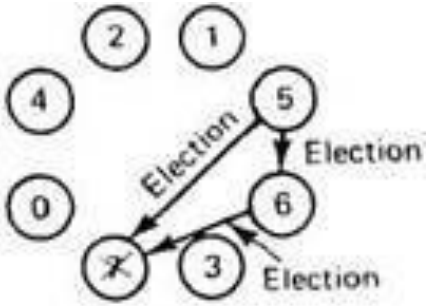


(a)

Since 7 is crashed it does not received The message

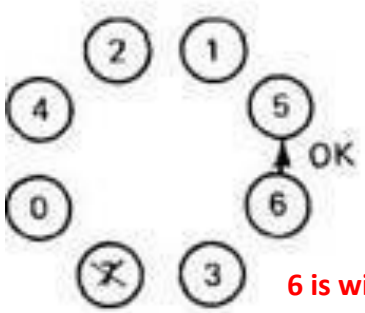


(b)



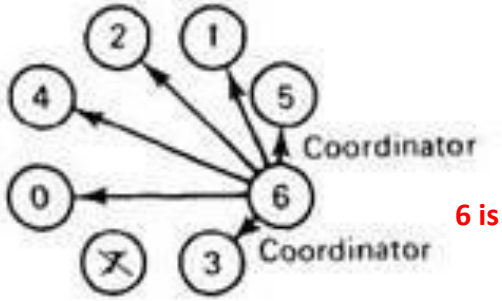
(c)

Both 5 and 6 hold the election



6 is winner

(d)



6 is announcing

(e)

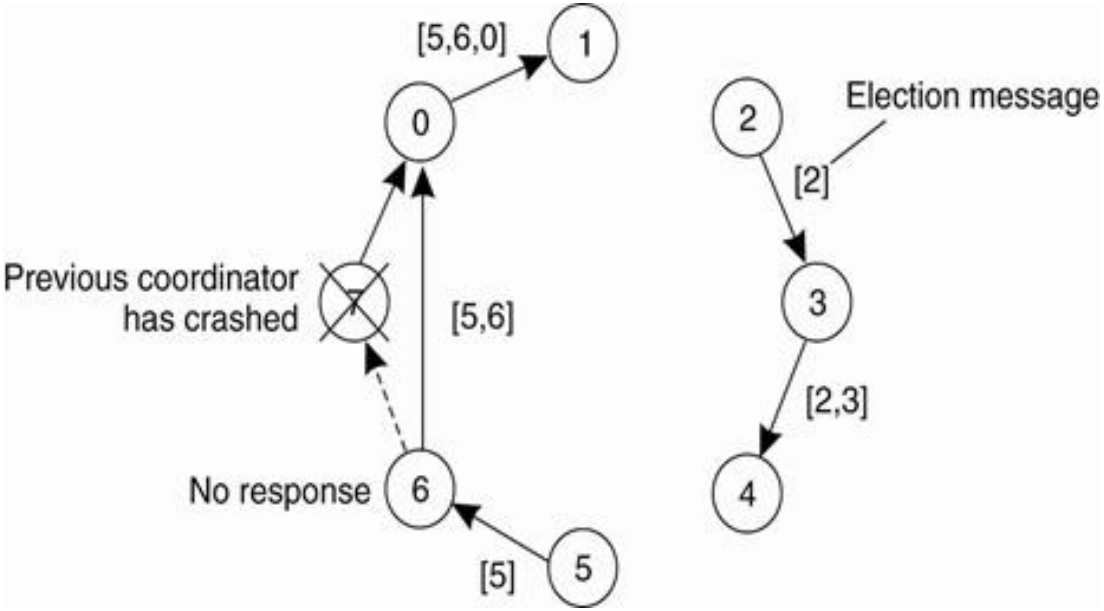
Ring Algorithm

- If any process 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 process adds its own identifier to the message
- When the election message reaches the originator, the election is complete

Ring Algorithm……..

- Procedure
  - If process P1 detects a coordinator failure, it creates a new active list which is empty initially. It sends election message to its neighbor on right side and adds number, to its active list
  - If process P2 receives message, in the same away it adds its number in active list and forwarded the message to its neighbor
  - If a process receives its own message, then it detects the process with highest number as a new leader

Ring Algorithm.....



Step 1:- 2 and 5 are initiating the election

Step 2 :- They forward the election message to its neighbor with its IDs and so on until the circuit builds

Step 3:-
2 → [2,3,4,5,*6*,0,1,2]
5 → [5, *6*,0,1,2,3,4,5]

Both elect 6 as 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 far 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 ordering among them
- If event ***a*** happened before event ***b***, then it can be represented as

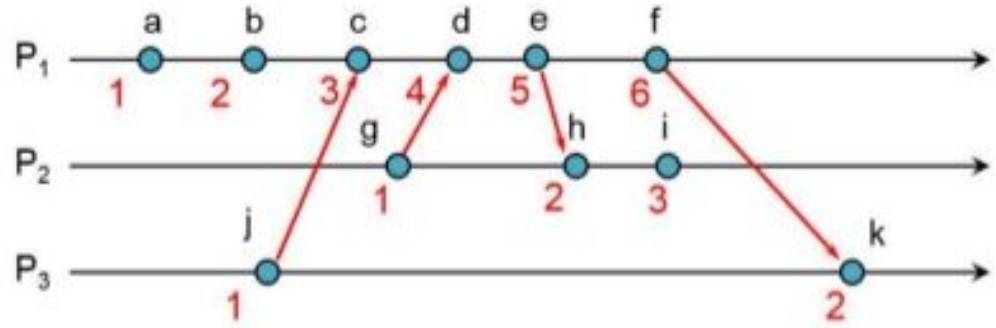
  a → b OR
- If and then
- If then T(a) < T(b) where T is timestamp
- But if T(a) < T(b) then can we guarantee

Logical clock and concurrency

- Assign a clock to each events, if $a \rightarrow b$, then ***clock(a) < clock(b)***
- If a and b occur on different processes, that do not exchange message, then neither $a \rightarrow b$ nor $b \rightarrow a$ are true, these events are called concurrent

Lamport's Logical Clock

- Let us assume the following example



Bad ordering
(e,h)5<2no
(f,k) 6<2 no
(j,c)(1,3)1<3

- Lamport's Algorithm
  - 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)

Vector Clock

- **Rules**

1. Vector initialization to 0 at each process

   $V_i[j] = 0$, for i, j = 1, 2, ……., N

1. Process increments its elements of the local vector before time stamping event

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

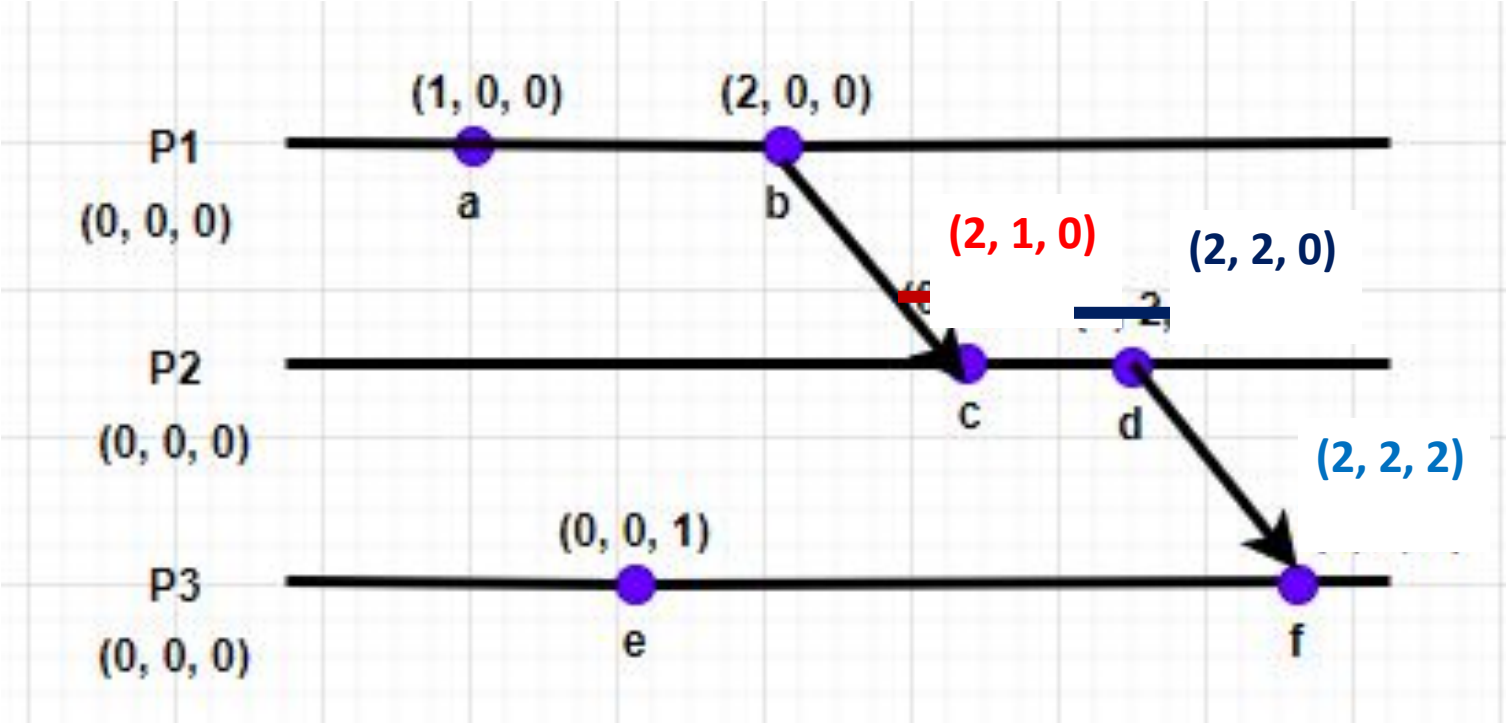
1. Message is sent from process P_i with V_i attached to it
2. When P_i receives message, compare vector elements, and set local vector to higher of

Vector Clock……..

- Comparing vector timestamps
  1. V = V’ iff V[i] = V’[i] for i = 1, 2, ……..,N
  2. V ≤ V’ iff V[i] ≤ V’[i] for i = 1, 2, ……..,N
- For any two events e and e’
  1. If e → e’ then V(e) < V(e’)
  2. If V(e) < V(e’) then e → e’
- Two events e and e’ are concurrent iff neither V(e) < V(e’) is true nor V(e’) < V(e) is true

Vector Clock……..

- Example



Vector Clock……..

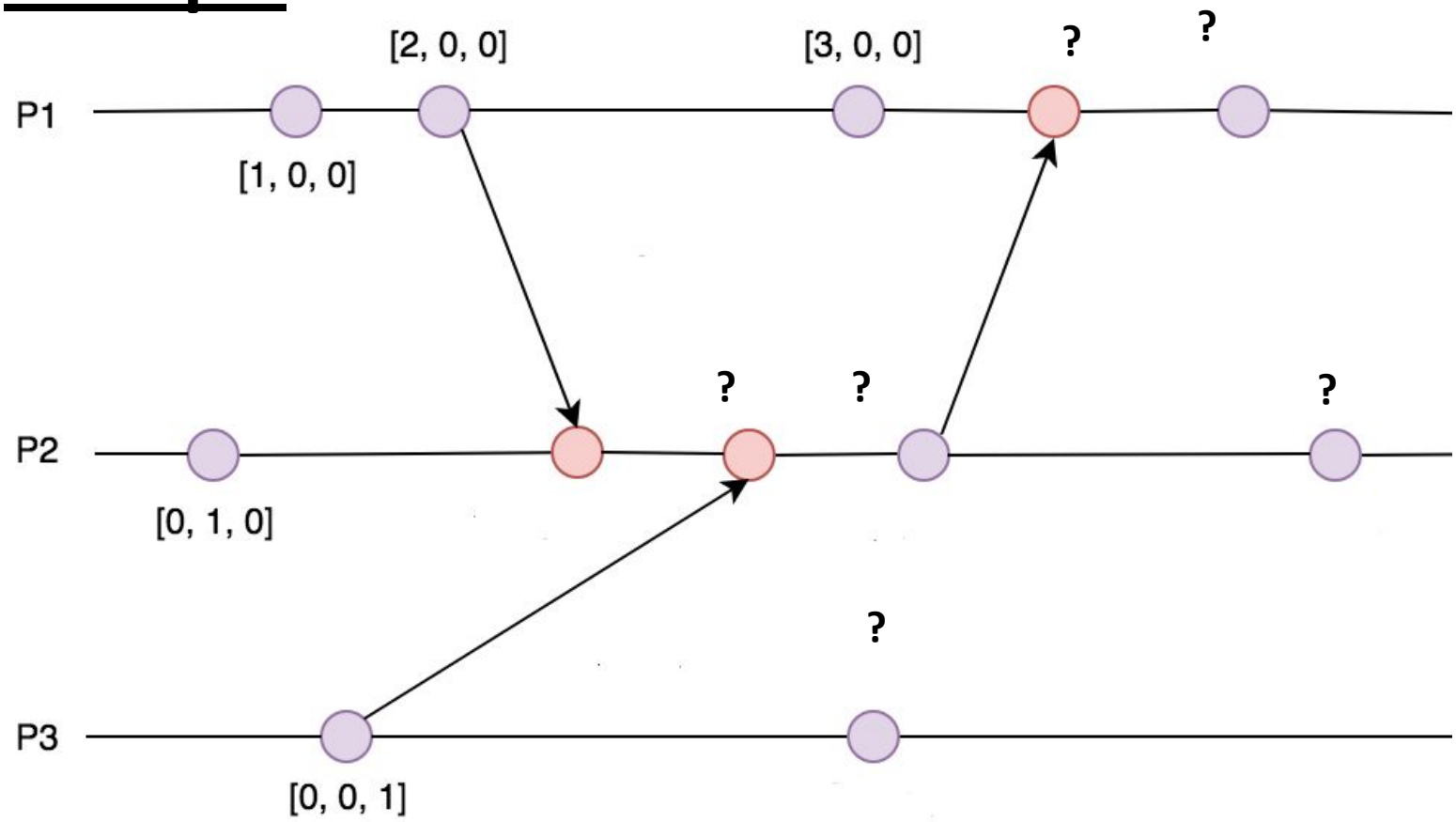
- **Events** **Timestamp**

a	(1, 0, 0)
b	(2, 0, 0)
c	(2, 1, 0)
d	(2, 2, 0)
e	(0, 0, 1)
f	(2, 2, 2)

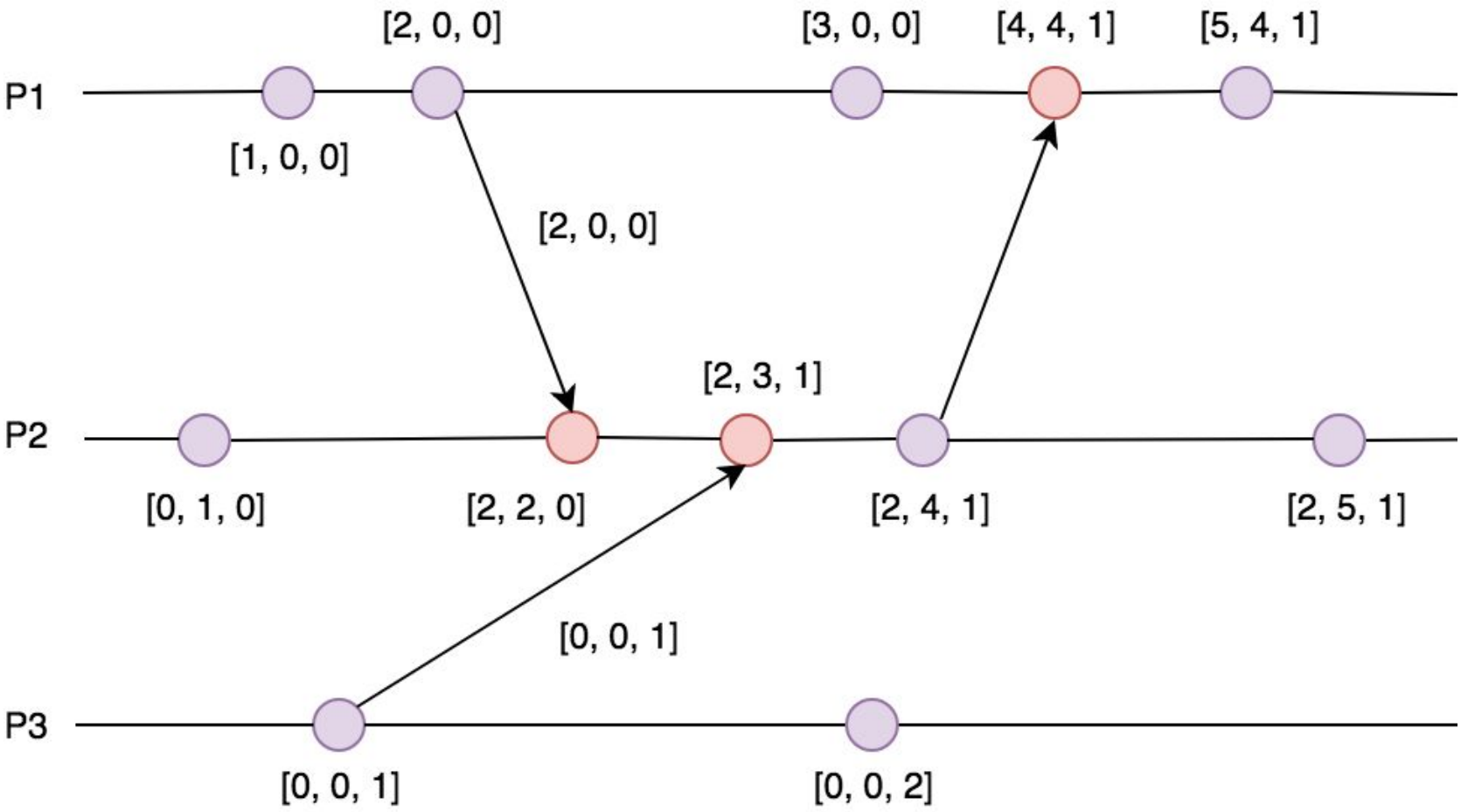
- **Which are concurrent events?**

Vector Clock……..

- **Example**



Vector Clock.....



Resource Allocation

- Problem arises in distributed systems when processors needs to share resources
- In such problem, the program is partitioned into 4 regions int a….. P{……a=a+1…..}

1. **Trying** → where the processor tries to acquire the resources
2. **Critical** → where the resources are being used
3. **Exit** → where some cleanup is performed
4. **Remainder** → the rest of the code

Tolerating processor failure in synchronous system

- In real systems, the various components may not operate correctly all the time
- It is assumed that communication links are reliable but processors are not reliable
- Two types of failure

1. Crash Failure
2. Byzantine Failure

Tolerating processor failure in synchronous system.....

- **Crash Failure**
  - When a processor halt during execution
  - A processor crashes in round ***r***, if it correctly operates in round ***1, 2, ..., r-1*** and it does not take a step in round ***r + 1***
  - Once the processor crashes, it is of no interest to the algorithm and no requirements are made on its decisions
- **Byzantine Failure**
  - Where failed processor (traitor) can behave arbitrarily

Assignment

- Discuss on
  - Sparse Network Covers
  - Tolerating Processor failure in Asynchronous system
  - Wait free implementation of shared object

Due date :- 2020 june 15

End of Session
Unit 1.5

Unit 2.1 :- PRAM Models

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

PRAM Models……..

- **Example**
- Finding the smallest value in the set if ***N*** values
- **Algorithm 1**
  - All possible comparisons of the pairs of the elements from set of numbers and carried out simultaneously each processor executing one operation of comparison

PRAM Models……..

- **Algorithm 2**
  - Do in parallel (***Actual no. of processors required*** $(\boldsymbol{P}) = \frac{\boldsymbol{n(n-1)}}{\boldsymbol{2}}$)
    - P_i derives the pair (i_1, i_2) of indices that corresponds to a different I
    - P_i reads value $L(i_1)$ and $L(i_2)$
    - If $L(i_1) \geq L(i_2)$ then P_i sends negative outcome to Pi_1 else P_i sends negative outcome to Pi_2
    - As this stage the only active processors is P_j, $1 \leq j \leq n$, which did not receive a negative outcome
    - P_j reads the value of L(j) and write it into the output cell

PRAM Models……..

- **Tracing (Algorithm 2)**
- Example
  - L = {2, 6, 4, 8}, No. of elements (n) = 4
  - So, no. of processors $(P) = \frac{n(n-1)}{2} = \frac{4*3}{2} = 6$
  - $P_1 \rightarrow <1, 2> \rightarrow$ -ve P_2
  - $P_2 \rightarrow <1, 3> \rightarrow$ -ve P_3
  - $P_3 \rightarrow <2, 3> \rightarrow$ -ve P_2
  - $P_4 \rightarrow <1, 4> \rightarrow$ -ve P_4
  - $P_5 \rightarrow <2, 4> \rightarrow$ -ve P_4
  - $P_6 \rightarrow <3, 4> \rightarrow$ -ve P_4

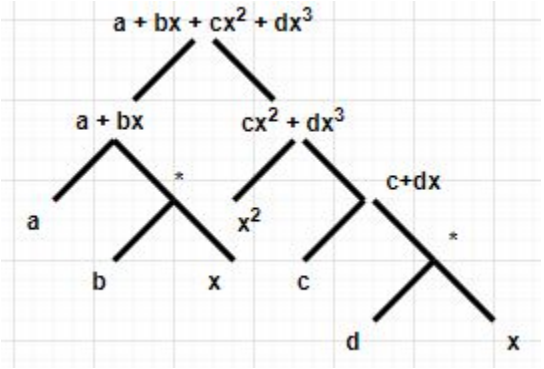
Here, for $1 \leq j \leq n(4)$, among P_1, P_2, P_3 and P_4, P_1 didn't get –ve outcome, so j=1 and L(1) = 2 is the smallest element

PRAM Models……..

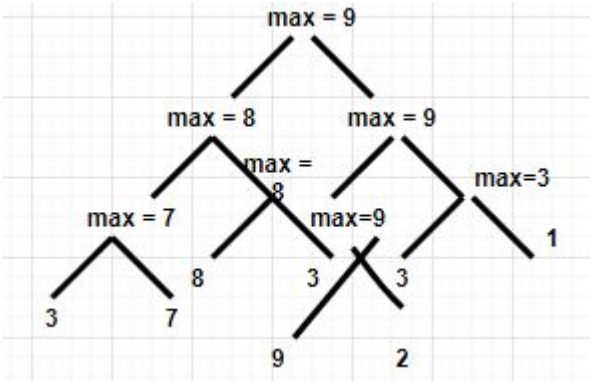
- **Tracing (Algorithm 2)**
  - Here processor P_i is chose as $i = \frac{(i_2-1)(i_2-2)}{2} + i_1$
  - Example $\rightarrow$ {2, 6, 4, 8}
  - (1, 2) $\rightarrow \frac{(2-1)(2-2)}{2} + 1 = 1 \rightarrow P_1$
  - (1, 3) $\rightarrow \frac{(3-1)(3-2)}{2} + 1 = 2 \rightarrow P_2$
  - (1, 4) $\rightarrow \frac{(4-1)(4-2)}{2} + 1 = 4 \rightarrow P_4$
  - (2, 3) $\rightarrow \frac{(3-1)(3-2)}{2} + 2 = 3 \rightarrow P_3$
  - (2, 4) $\rightarrow \frac{(4-1)(4-2)}{2} + 2 = 5 \rightarrow P_5$
  - (3, 4) $\rightarrow \frac{(4-1)(4-2)}{2} + 3 = 6 \rightarrow P_6$

Techniques for the design of parallel algorithm

1. Divide and conquer techniques



2. Balanced binary tree method



[3,7,8,3,9,2,3,1] → [7,8,9,3] → [8,9] → 9

PRAM Model

1. **EREW** (Exclusive Read Exclusive Write) → every memory cell can be read or written to by only one processor at a time
2. **CREW** (Concurrent Read Exclusive Write) → multiple processors can read a memory cell but only one can write at a time
3. **ERCW** (Excusive Read Concurrent Write) → never considered
4. **CRCW** (Concurrent Read Concurrent Write) → multiple processors can read and write

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
- Work of a parallel algorithm A is,

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

- $Speedup = \frac{t(N)}{T(N)}$ where t(N) $\rightarrow$ sequential time
- $Efficiency = \frac{Sppedup}{P(N)}$

Basic PRAM algorithms

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

Computing Prefix sums

- Given, A = $\{a_0, a_1, a_2, \ldots\ldots\ldots\ldots\ldots, a_{n-1}\}$
- We have to compute,

  $\{a_0, (a_0 + a_1), (a_0 + a_1 + a_2), \ldots\ldots\ldots, (a_0 + a_1 + \ldots\ldots\ldots\ldots + a_{n-1})\}$
- Eg
- A = {5, 3, -6, 2, 7, 10, -2, 8}
- The output is

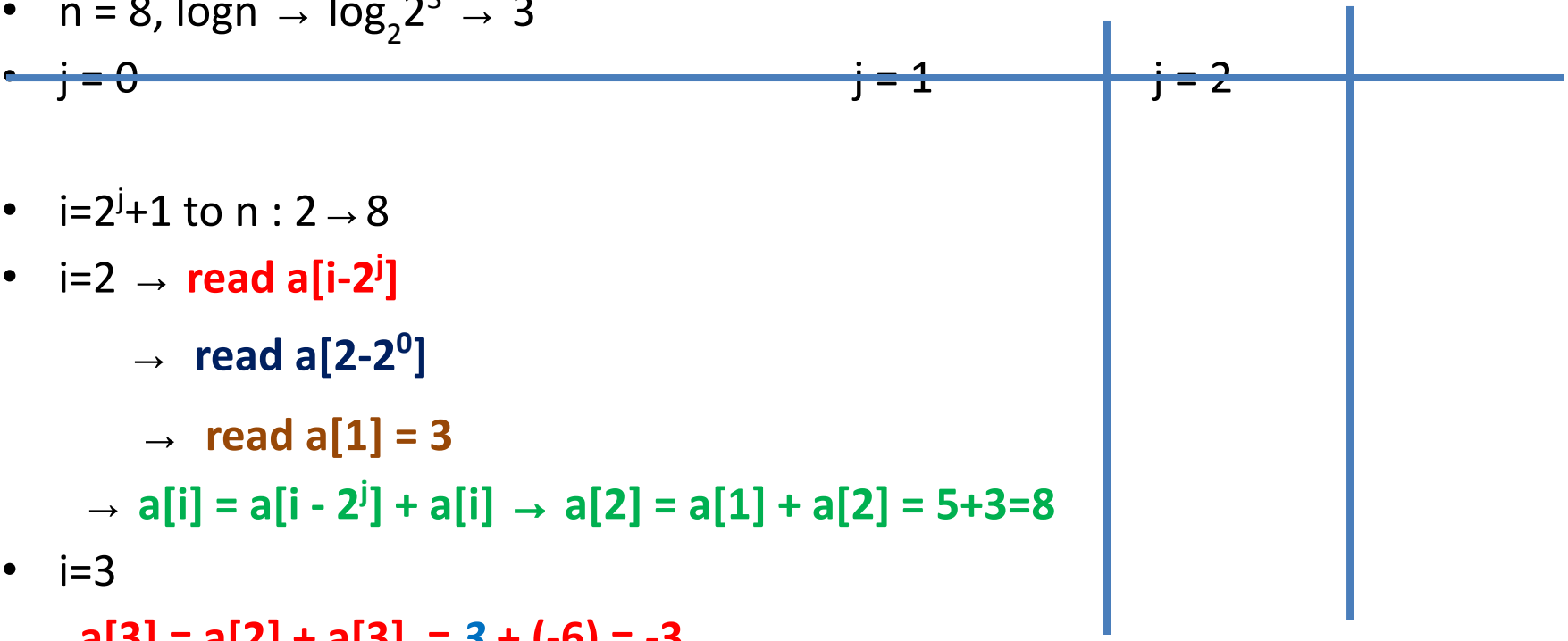
  {5, 8, 2, 4, 11, 21, 19, 27}

Computing Prefix sums……

- Algorithm
  - Input an array [a_1, a_2, ……, a_n]
  - If (n == 1) then $S_1 \leftarrow a[1]$
  - Else
    - For j = ***0*** to ***logn – 1*** do
      - For i = $\boldsymbol{2^j + 1}$ to ***n*** do in parallel
        - » Processor P_i
        1. Obtains $a[i - 2^j]$ from through shared memory
        2. $a[i] = a[i - 2^j] + a[i]$

Computing Prefix sums……

- Tracing :- A = {5, 3, -6, 2, 7, 10, -2, 8}
- n = 8, logn → $\log_2 2^3$ → 3
- j = 0 j = 1 j = 2



- i=2^j+1 to n : 2 → 8
- i=2 → **read a[i-2^j]**

  → **read a[2-2^0]**

  → **read a[1] = 3**

  → **a[i] = a[i - 2^j] + a[i] → a[2] = a[1] + a[2] = 5+3=8**
- i=3

  a[3] = a[2] + a[3] = *3* + (-6) = -3
- i=4
- i=5
- i=6

List Ranking

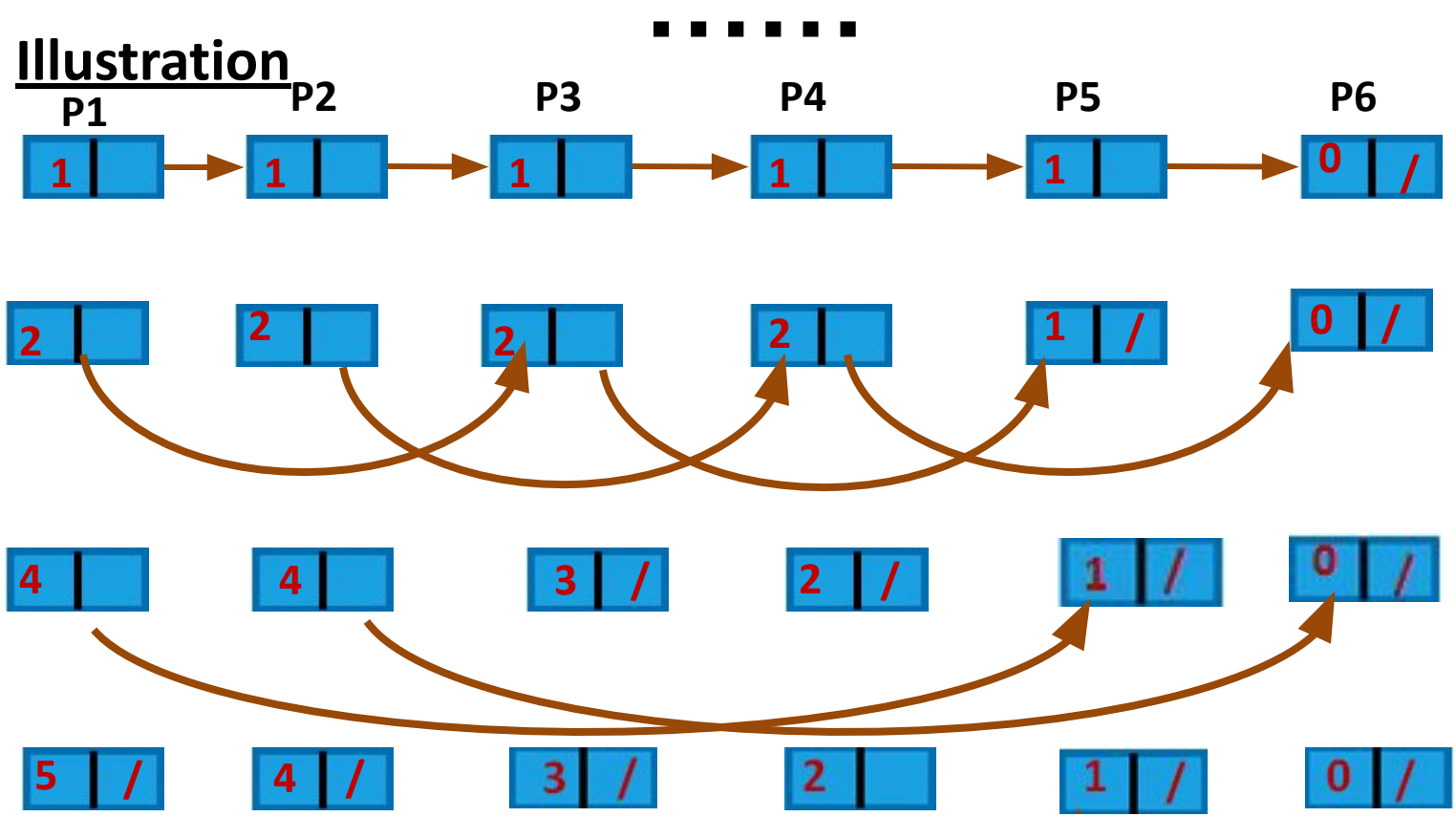
- The problem is that, given a singly linked list ***L***, with ***N*** objects, for each node compute the distance to the end of the list
- If d denotes the distance

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

- **Parallel algorithm**
  - Assign one processor for each node
  - For each node i, do in parallel
    - i.d = i.d + i.next.d
    - i.next = i.next.next

List Ranking.....

- **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 $<a_0, a_1, \ldots\ldots, a_{n-1}>$, there exists $<a_0, a_1, \ldots., a_i>$ is increasing and $<a_{i+1}, a_{i+2}, \ldots\ldots., a_{n-1}>$ is decreasing
- If an/2 is the beginning of the decreasing sequence S, then

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

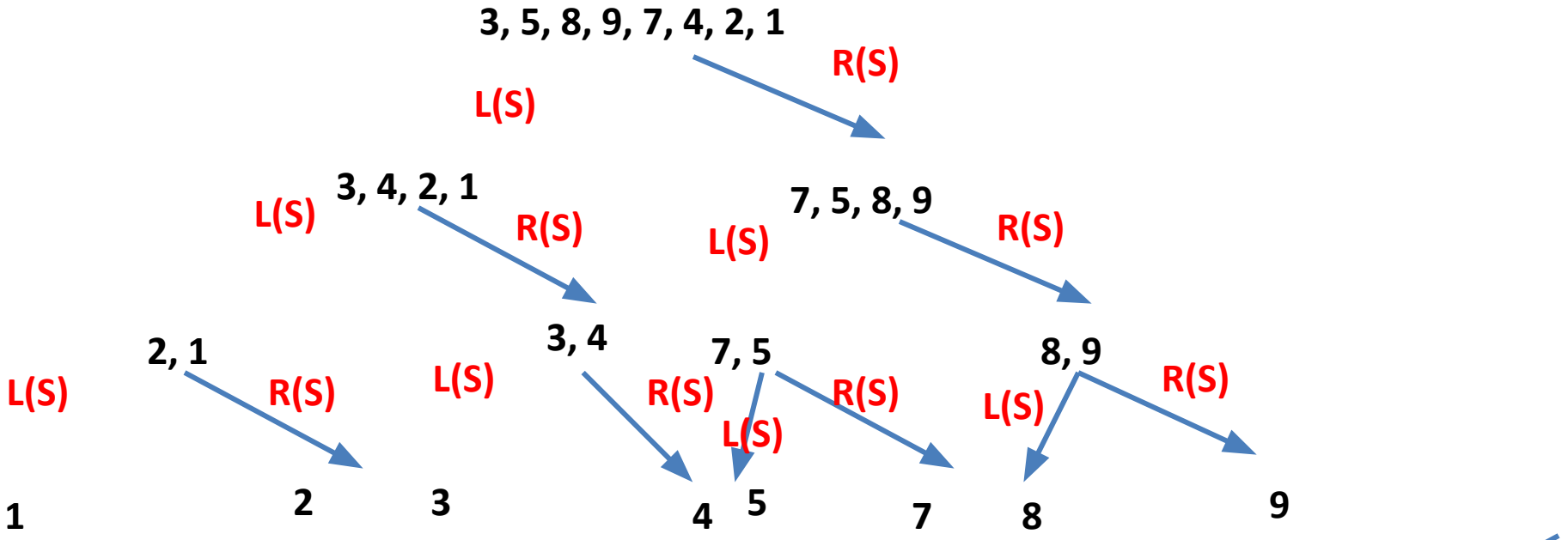
  $R(S) = \{\max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots.., \max(a_{n/2-1}, a_{n-1}\}$

Parallel Sorting Algorithm……

- 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)

Parallel Sorting Algorithm……

- **Illustration**



Randomized Algorithm

- Is the algorithm where execution is controlled at one or more points by randomly made choices
- Primality Testing
  - Fermat's Little Theorem

    $$b^p \equiv b(mod\ p)$$

  - Where b $\in Z_p$

Deficiencies of PRAM algorithm

- No mechanisms for representing communication between the processors
- Storage management and communication issues are hidden from the algorithm designer

End of Session
Unit 2.1

Unit 2.2 :- BSR (Broadcasting with Selective Reduction)

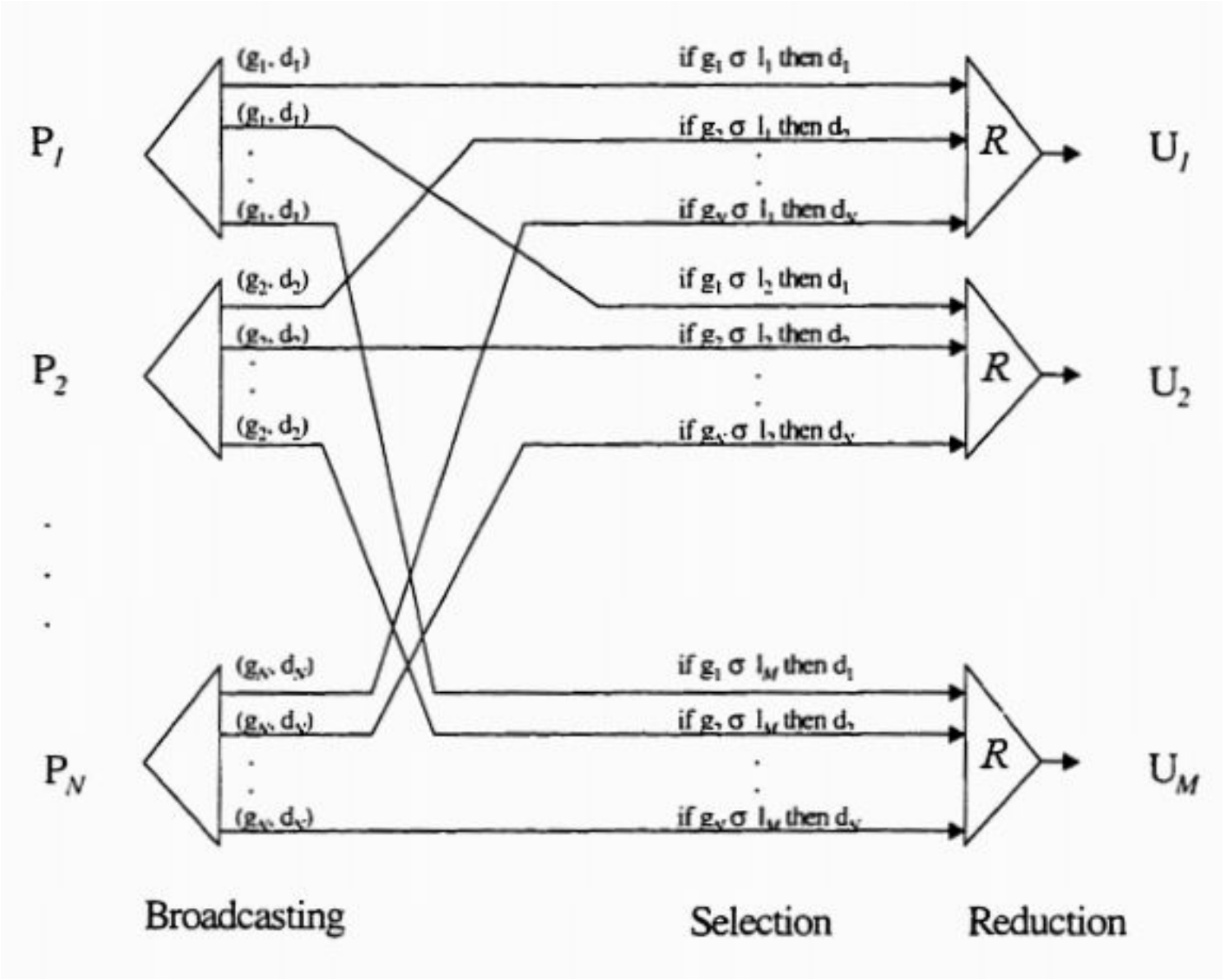
BSR

- Broadcast instruction consists of three phases
  1. Broadcast Phase
  2. Selection Phase
  3. Reduction Phase
- **Broadcast Phase**
  - Allows all of the ***N*** processors to write concurrently to all of the ***M*** memory locations
  - Each processor P_i, $1 \leq i \leq N$, produced a record containing two fields, a tag t_i and a datum d_i, to be stored

BSR.....

- **Selection Phase**
  - After the data are received at each memory locations Uj, $1 \leq j \leq M$, a switch Sj will select the receiving data di by comparing tag value ti by using a selection rule, σ [<, ≤, =, >, ≥, ≠]
- **Reduction Phase**
  - Selected data are reduces to a single value using reduction rule, R {Σ, π, $\wedge$, $\vee$, $\oplus$,}

BSR.....


P_1
P_2
P_N
(g_1, d_1)
(g_1, d_1)
(g_1, d_1)
(g_2, d_2)
(g_2, d_2)
(g_N, d_N)
(g_N, d_N)
(g_N, d_N)
if $g_1 \sigma l_1$ then d_1
if $g_2 \sigma l_1$ then d_2
if $g_N \sigma l_1$ then d_N
if $g_1 \sigma l_2$ then d_1
if $g_2 \sigma l_2$ then d_2
if $g_N \sigma l_2$ then d_N
if $g_1 \sigma l_M$ then d_1
if $g_2 \sigma l_M$ then d_2
if $g_N \sigma l_M$ then d_N
R
R
R
U_1
U_2
U_M
Broadcasting
Selection
Reduction

BSR.....

- **A generalized BSR model**
  - Mathematically, it can be expressed as

$$\bigwedge_{1\le j\le M} \; t(i, h) \; \sigma_h \; \bigwedge_{1\le i\le N} \; l(j, h) \; \bigwedge_{1\le h\le k} \; \bigwedge_{1\le j\le M}$$

  - Where
    - N → no. of processors
    - d_i → datum broadcast by the processor P_i
    - σ_h → selection operation, 1≤h≤k, where ***k*** is the number of criteria
    - t(i, h) → tag broadcast by processor i for criteria h
    - l(i, h) → limit value j for criteria σ_h
    - R → reduction operation, $R \in \{\Sigma, \pi, \wedge, \vee, \oplus, \cap, \cup\}$

BSR.....

- **Types**
  1. One criterion BSR algorithm
  2. Two criterion BSR algorithm
  3. Three criterion BSR algorithm
  4. Multiple criterion BSR algorithm

One Criterion BSR

- Each broadcast datum has to pass a single test being allowed to participate in the reduction process
- **Example**
  - Sorting
  - Parenthesis Matching
  - Optimal Sum Subsegment

One Criterion BSR.....

- **Sorting**
  - Number of processors needed = N (number of elements in array is N)
  - Rank of element xj can be expressed as

    $r_j = \Sigma 1 \mid x_i \leq x_j$
  - When all the elements in the array are distinct, every datum x_j in its position r_j are in the sorted array

One Criterion

BSR……

- **Sorting**
  - Illustration

A = {3, 2, 6, 5}

P1 P2 P3 P4

for 3 → 3 ≤ 3, 2 ≤ 3, ∴rank = 2

for 2 → 2 ≤ 2, ∴ rank =1

for 6 → 3 ≤ 6, 2 ≤ 6, 6 ≤ 6, 5 ≤ 6, ∴rank = 4

for 5 → 3 ≤ 5, 2 ≤ 5, 5 ≤ 5, ∴rank = 3

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

One Criterion BSR……

- **Sorting**
  - What happens if some of the elements are duplicate
  - A = {3, 4, 3}

One Criterion BSR……

- **Sorting (in case of duplicate)**
  - $S_j = \Sigma 1 \mid t_i \leq l_j$ where $t_i = r_i - \frac{1}{i}, l_j = r_j - \frac{1}{j}$
  - Illustration $\rightarrow$ A = {3, 4, 3}

Parenthesis Matching

- One criterion BSR algorithm
- The problem is to find the pairs of matching parenthesis in a given legal sequences l_1, l_2, ……., l_n of parenthesis
- By legal means that every parenthesis has its
-

(	)	(	(	(	)	)	)	(	(	)	)
1	2	3	4	5	6	7	8	9	10	11	12
2	1	8	7	6	5	4	3	12	11	10	9

Output for the input sequence,

( ) ((( ))) (( )) is,

Parenthesis Matching……

- **Algorithm**
  - For each processor j do in parallel
  - If l_j == '(' then $b_j = 1$ else $b_j = -1$
  - $P_j = \Sigma b_j \mid i \leq j$
  - If b_j == -1 then $P_j = P_j + 1$
  - $P'_j = P_j - \frac{1}{j}$
  - $q_j = -1$, $t_j = 0$, $r_j = 0$
  - $q_j = \cap P'_i | P'_i < P'_j$
  - $t_j = \cap i \mid P'_i = q_j$
  - $r_j = \cup i | t_i = j$
  - If l_j = ')' then $m_j = t_j$ else $m_j = r_j$

Parenthesis Matching

- One criterion BSR algorithm
- The problem is to find the pairs of matching parenthesis in a given legal sequences l_1, l_2, ……., l_n of parenthesis
- By legal means that every parenthesis has its
-

(	)	(	(	(	)	)	)	(	(	)	)
1	2	3	4	5	6	7	8	9	10	11	12
2	1	8	7	6	5	4	3	12	11	10	9

Output for the input sequence,

( ) ((( ))) (( )) is,

Parenthesis Matching……

- **Algorithm**
  - For each processor j do in parallel
  - If l_j == '(' then $b_j = 1$ else $b_j = -1$
  - $P_j = \Sigma b_j \mid i \leq j$
  - If b_j == -1 then $P_j = P_j + 1$
  - $P'_j = P_j - \frac{1}{j}$
  - $q_j = -1$, $t_j = 0$, $r_j = 0$
  - $q_j = \cap P'_i | P'_i < P'_j$
  - $t_j = \cap i \mid P'_i = q_j$
  - $r_j = \cup i | t_i = j$
  - If l_j = ')' then $m_j = t_j$ else $m_j = r_j$

Parenthesis Matching

j=1, P'_1=0,there is no less P' value less than 0

j=2, P'_2=0.5,less than 0.5=0,and max = 0

j=4, P'_4=1.75, less than 1.75 = 0,0.5,0.66,0.87,0.89,0.92 and max = 0.92

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
Sequence	(	)	(	(	(	)	)	)	(	(	)	)
Index(j)	1	2	3	4	5	6	7	8	9	10	11	12
b_j	1	-1	1	1	1	-1	-1	-1	1	1	-1	-1
$\sum b_i \mid i \le j$	1	0	1	2	3	2	1	0	1	2	1	0
P_j	1	1	1	2	3	3	2	1	1	2	2	1
P'_j	0	0.5	0.66	1.75	2.8	2.83	1.86	0.87	0.89	1.9	1.91	0.92
q_j	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
q_j	-1	0	0.5	0.92	1.91	2.8	1.75	0.66	0.87	1.86	1.9	0.89
t_j	0	1	2	12	11	5	4	3	8	7	10	9
r_j	2	3	8	7	6	0	10	9	12	11	5	4

J=1

J=1, (

J=7,)

Maximal Sum Sub Segment

- Given an array of numbers $d_1, d_2, \ldots\ldots, d_m$, it is required to find a contiguous sub array of maximal sum
- Example
  - A = {-2, -3, 4, -1, -2, 1, 5, -3}
  - Maximal Sum = 4 + (-1) + (-2) + 1 + 5 = 7

Maximal Sum Sub Segment……

- **Algorithm**

Algorithm

- $S_j = \sum d_i | i \leq j$
- $m_j = \cap S_i | i \geq j$
- $e_j = \cap i | S_i = m_j$
- $m_j = m_j - S_j + d_j$
- $t = \cap m_i$
- $x = \cap i | m_i = t \rightarrow starting\ point$
- $y = e_x \rightarrow ending\ point$

Maximal Sum Sub Segment……

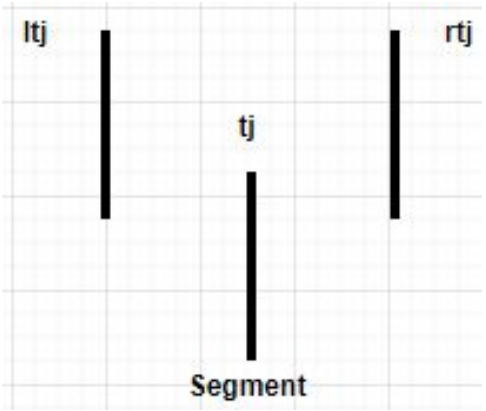
- **Tracing**
  - A = {31, -41, 59, 26, -53, 58, 97, -93, -23, 84}

Two criterion BSR algorithm

- BSR algorithm where broadcast data are tested for their satisfaction of two criteria before being allowed to take part in reduction process
- **Example**
  - 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
  - Eg
  - No. of inversions in {1, 6, 2, 9, 5, 3} = 3
  - They are {6, 2}, {6, 5}, {9, 5}
  - Rules
    - $y_j = \sum 1 \,|\{i < j\} \wedge \{\pi(i) > \pi(j)\}$

Three criterion BSR algorithm

- Have to pass three criterion to go to reduction process
- Eg:- vertical segment visibility



- $lt_j = \cap\, i | x_i < x_j \wedge t_i \geq t_j \wedge b_i \leq t_j (left\ top)$
- $rt_j = \cap\, i | x_i > x_j \wedge t_i \geq t_j \wedge b_i \leq t_j (right\ top)$

End of Session
Unit 2.2

Unit 2.3 :- Data Flow Models

Data Flow Models

- Flow of information between data processing entities
- Control flow and data flow
- Control flow assumes that a program in a series of instructions, each of which specifies
  - Either an operation with memory location
  - OR specifies transfer of control to another instruction
- Example
  - **C = if (n == 0) then a + b else a - b**

Basic primitives of data flow

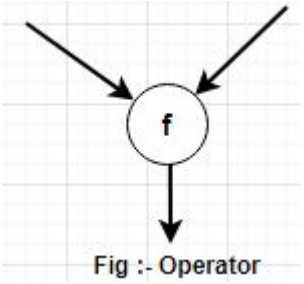


Fig :- Operator

a data value is produced by an operator as a result of some operation f

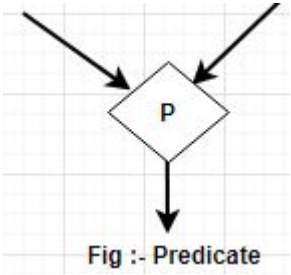


Fig :- Predicate

a TRUE or FALSE control value is generated by a decider (predicate) depending on its input tokens

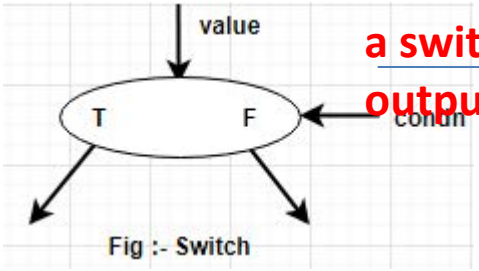


Fig :- Switch

a switch actor directs an input data token to one of its output depending upon the control input

Basic primitives of data flow……

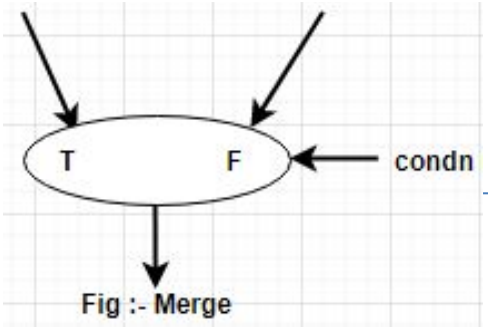


Fig :- Merge

A merge actor passes one of its input tokens to the output depending on the input control token

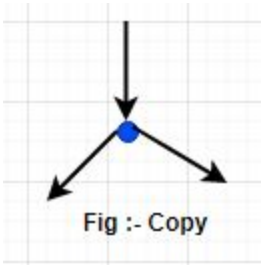
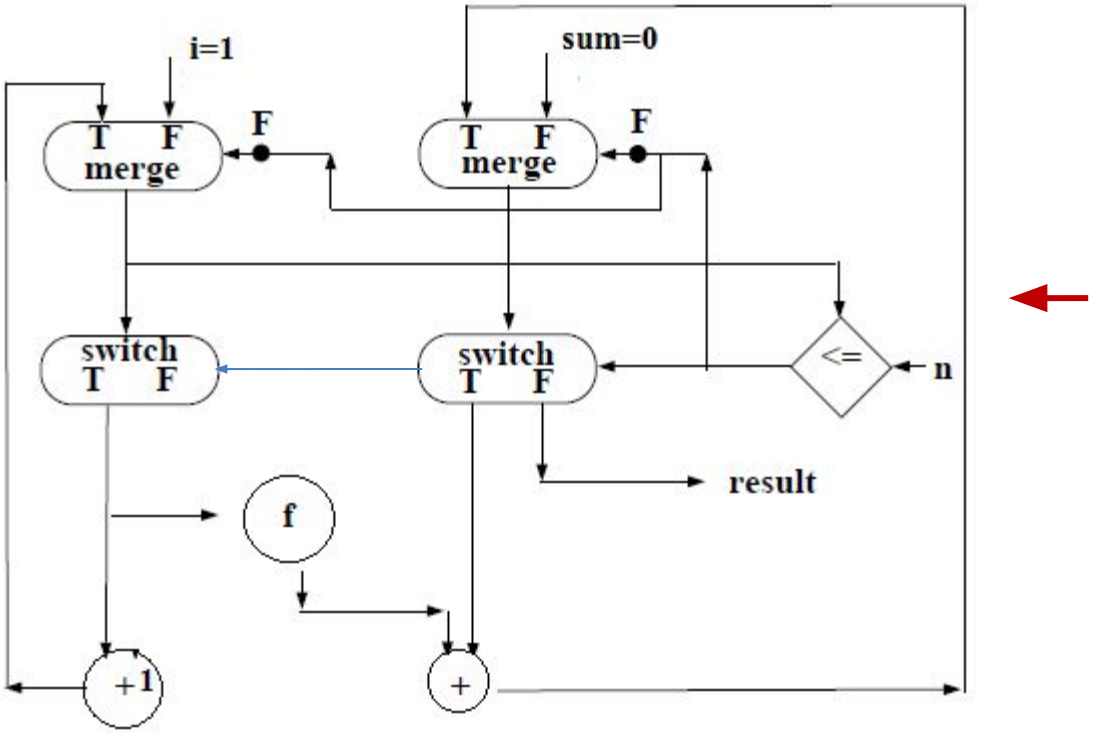


Fig :- Copy

A copy is an identity operator which duplicates input tokens

Basic primitives of data flow……

- **Example**



Basic primitives of data flow……

- **Description**
  - Control units to the merge actors are set to ***FALSE*** for initialization
  - The input values ***i*** and ***sum*** are admitted as the initial values of the iteration
  - The predicate ***i≤n*** is then tested
  - If it is true, the values of i and sum are routed to the ***TRUE*** sides of switch
  - This initiates the firing at the function ***f*** as well as increment of ***i***
  - Once the execution of the body of the loop

Basic primitives of data flow……

- **Example**

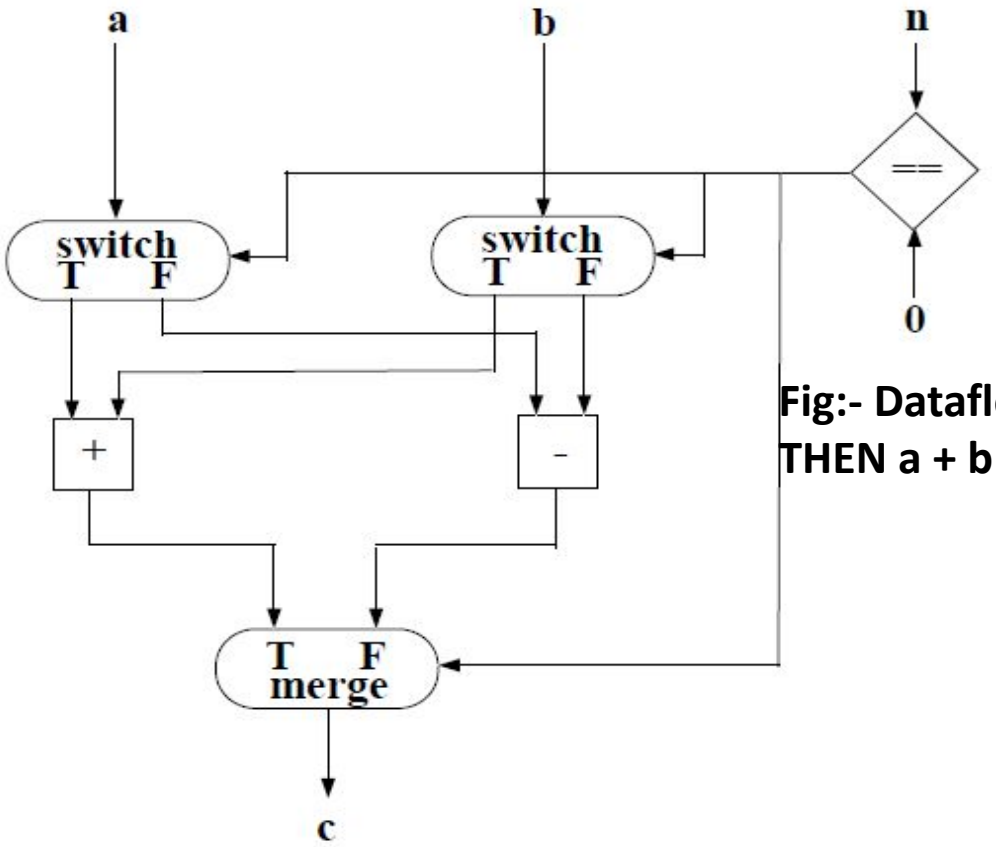


Fig:- Dataflow graph for C = IF n == 0 THEN a + b ELSE a - b

Dataflow graph for `c = if n == 0 then a+b else a-b`

Demand Driven Data Flow Computing Model

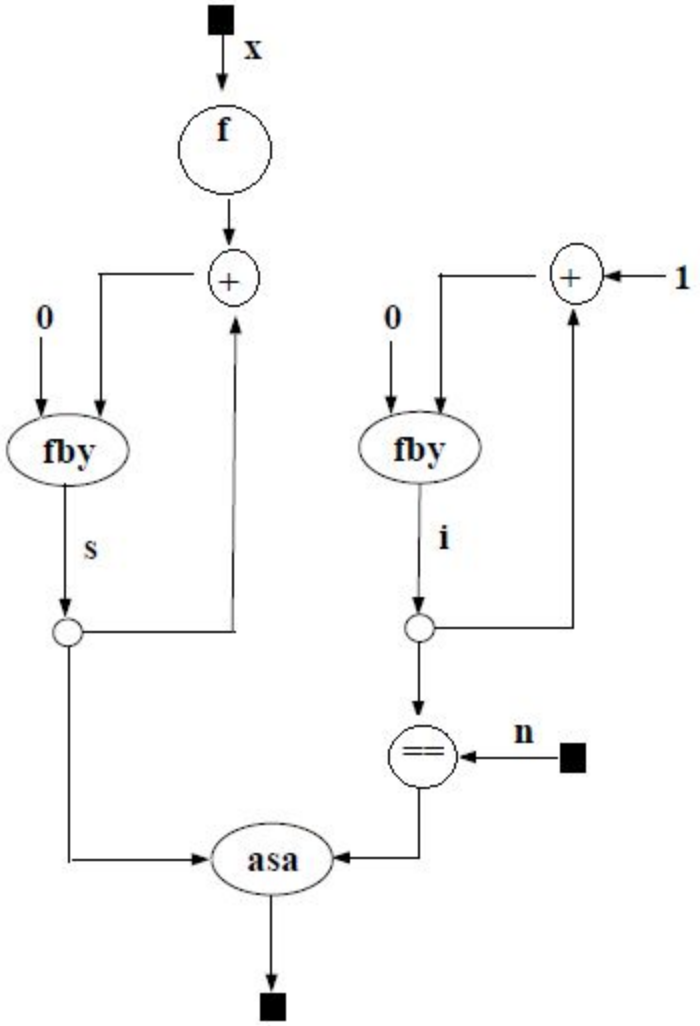
- An operation of a node will be performed only if there are tokens on all the input edges and there is a demand for the result of applying the operation
- **Operator net**
  - Programming model for demand driven
  - Consists of set of nodes, a set of arcs and set of equation that relate the output arcs of nodes to functions / operators applied to the input arcs

Demand Driven Data Flow Computing Model.....

- **Example (Operator net)** .....

fby → followed by
asa → as soon as

$$S = \sum_{i=1}^{n} f(x)$$



Assignment

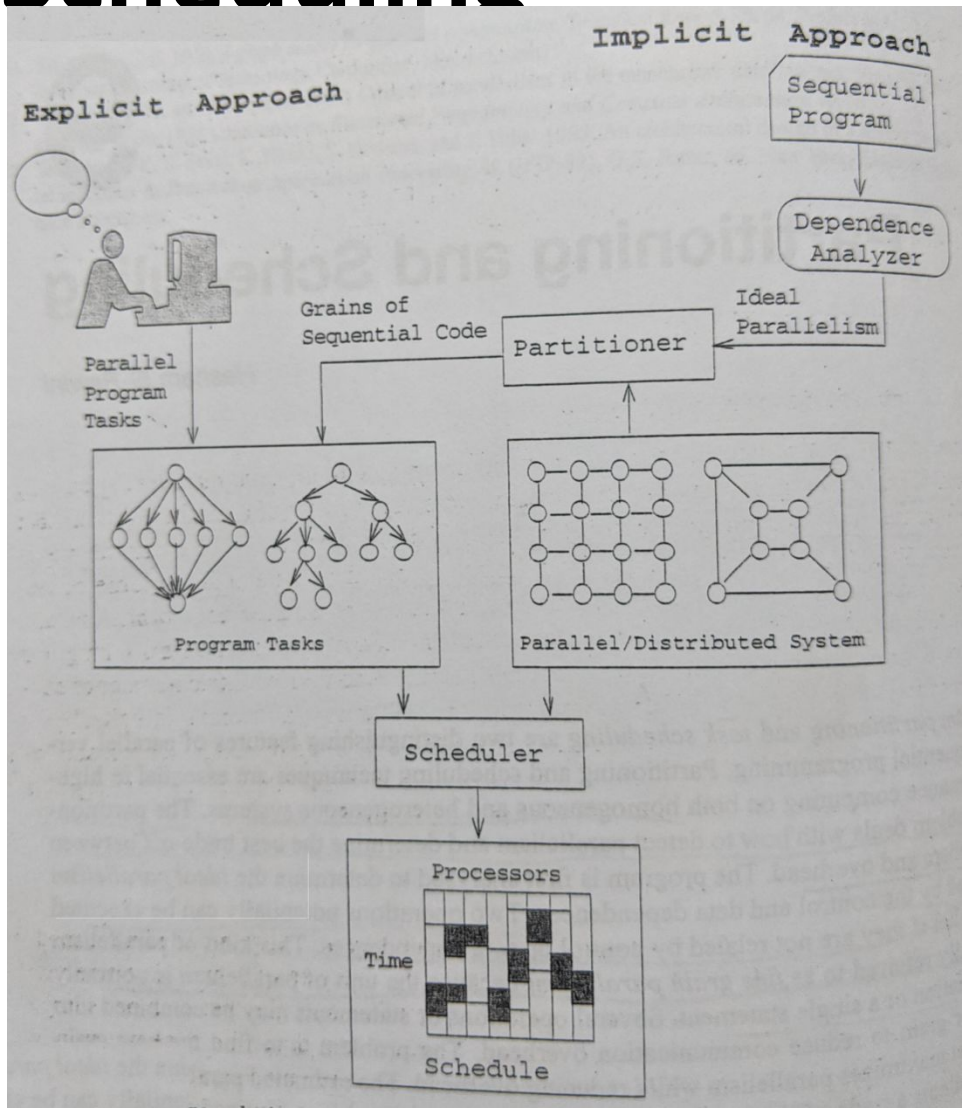
- Data flow architecture
  - Static Data flow model
  - Dynamic Data flow model

End of Session
Unit 2.3

Unit 2.4 :- Partitioning and Scheduling

Partitioning and Scheduling

- Partitioning deals with parallelism



Partitioning and Scheduling……

- Program Partitioning
  1. Data Partitioning
     - Same computation on different set of data concurrently
     - Eg:- matrix multiplication
  2. Function Partitioning
     - Different computation on same set of data concurrently
     - Eg:- flight simulation

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:-

Scheduling System Models

- **Consists of**
  1. Parallel program tasks
  2. Target machine
  3. Schedule
  4. Performance criteria

Parallel program tasks

- Can be defined as the system (T, <, D_{ij}, A_i), where
  1. T = {t_1, t_2, ……,t_n} is a set of tasks to be executed
  2. **<** 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
  3. 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
  4. A_i is n vector of the amount of computations i.e.

Parallel program tasks.....

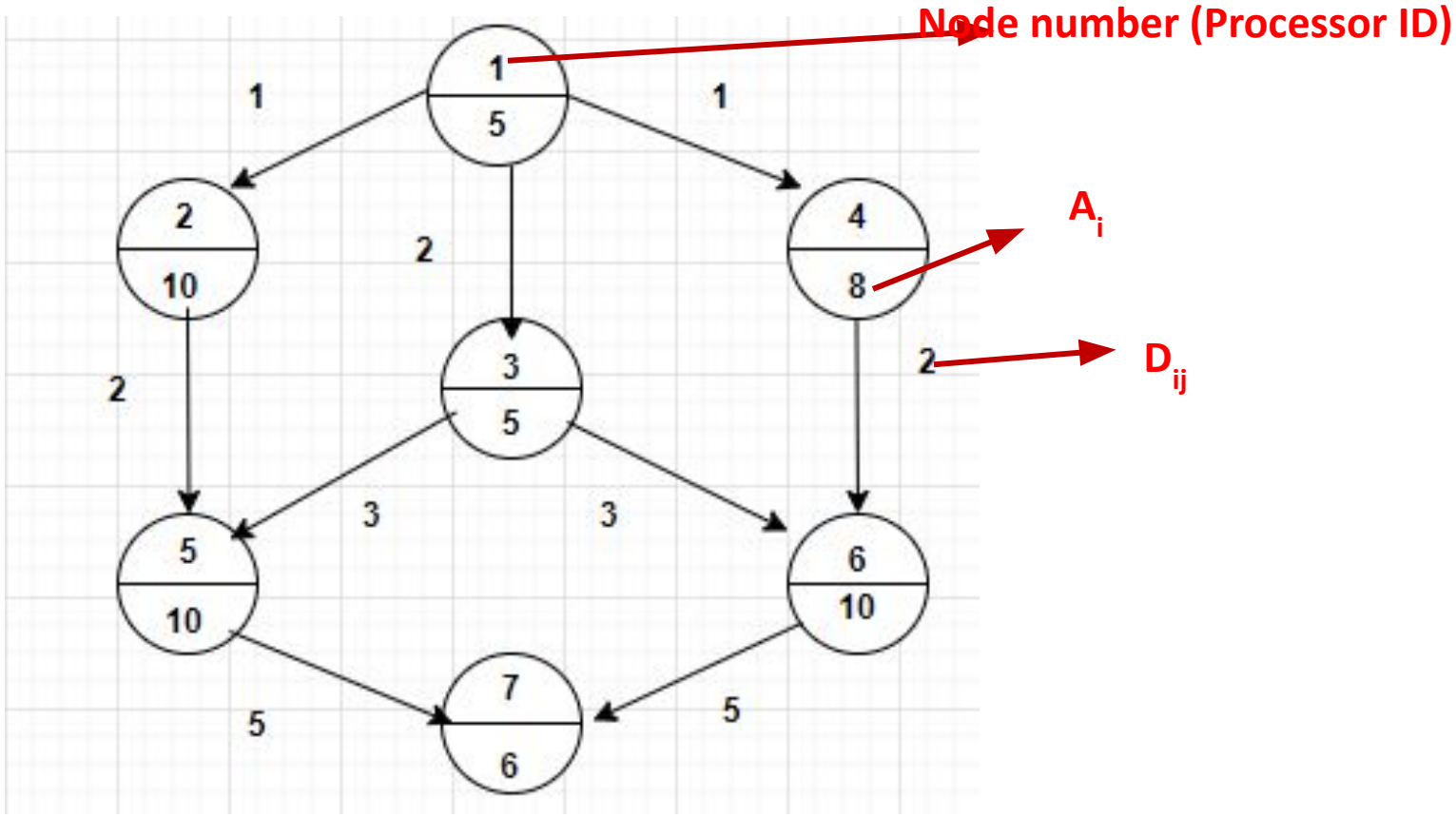


Figure :- Task Graph (n = 7)

Target Machine

- Assumed to be made up of "***m***" heterogeneous processing elements connected using an arbitrary interconnection network
- Formally target machine can be defined as (P, P_{ij}, S_i, T_i, B_i, R_{ij}), where
  1. ***P = {P_1, P_2,……., P_m}*** is a set of m processors forming the parallel architecture
  2. ***P_{ij}*** is an ***m×m*** interconnection topology matrix
  3. ***S_i*** specifies the speed of processor ***P_i***

$$T_{ij} = \frac{A_i}{S_j} + B_j$$

Schedule

- Formally defined as

  $f : v \rightarrow \{1, 2, \ldots\ldots\ldots, m\} \times \{0, 1, \ldots\ldots\ldots, \infty\}$
- i.e. f(v) = (i, t) for some ***v*** means task ***v*** is scheduled to be processed by processor P_i starting at time ***t***
- Task ***1*** starts on processor P_2 at time ***0*** and finishes at time ***10***
- Task ***5*** starts on processor P_3 at time ***0*** and finishes at time ***30***
- Shaded area is communication delay

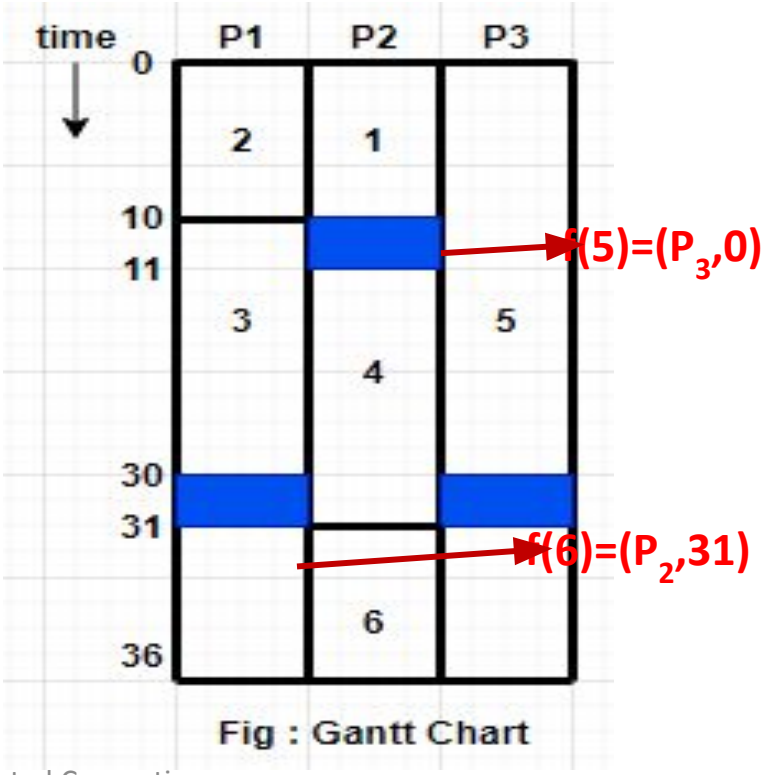


Fig : Gantt Chart

Performance Criteria

- Scheduling length or max finishing time
- Formally

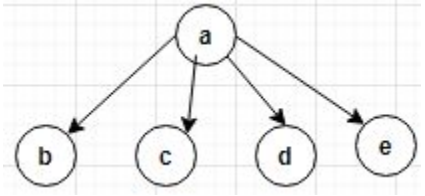
  length(f) = maximum $\{t + T_{ij}\}$
- Where $f(i) = (j, t)\ \forall i \in v$

Communication Model

- Two key component that contribute to the total completion cost is
  1. Execution Time
  2. Communication Delay
- **Model A**
  - Total cost = communication cost + execution cost
  - Where
    - Execution cost = schedule length
    - Communication cost = no. of message × cost per message
    - No. of message = the no. of node pairs (u, v) such that

Communication Model……

- **Example**



- Suppose that each of given task takes one unit of execution time in either P_1 or P_2
- Task a sends message to b, c, d, e
- Suppose a, b, d → allocated to P_1

  c, e → allocated to P_2

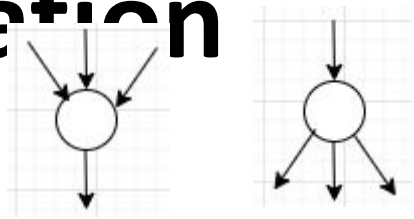
Optimal Scheduling Algorithm

- Deal in 3 cases
  1. When the task graph is a tree
  2. When the task graph is an interval order
  3. When there are only 2 processors available

Scheduling tree structure task graphs

A. Scheduling in – forests / out – forests without communication
B. Scheduling in – forests / out – forests with communication

Scheduling in – forests / out – forests without communication



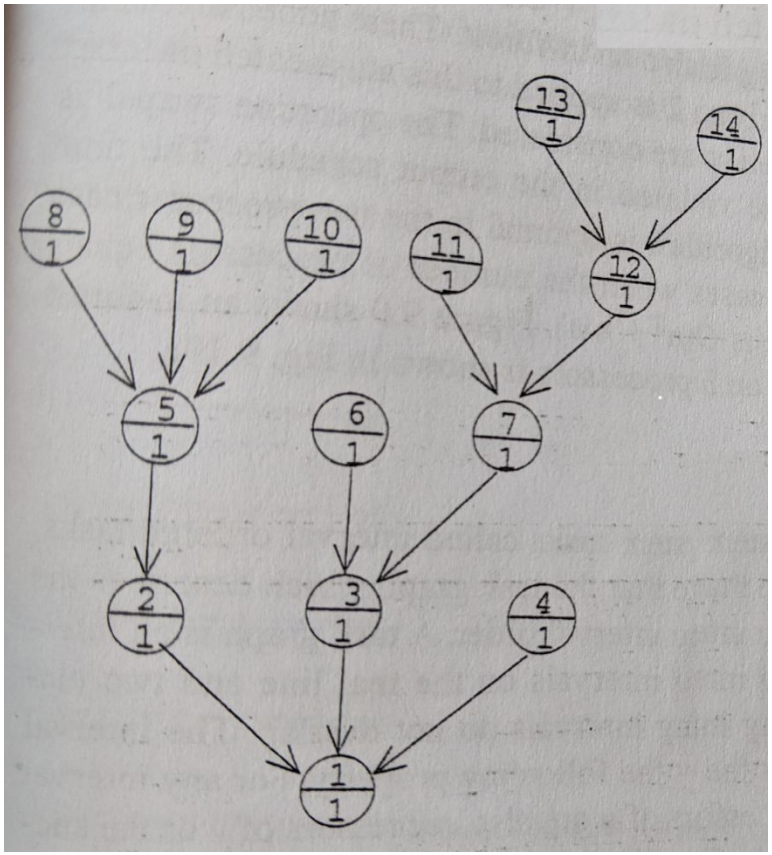
- Task graph is either
  - In-forest → each node has at most one immediate successors OR
  - Out-forest → each node has at most one immediate predecessors
- General strategy → used the highest level first
- **Algorithm**
  1. The level of each node in the task graph is calculated and used as each node priority
  2. Whenever a processor becomes available, assign the unexecuted ready task with the highest

forests without communication.....

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

.....

∴Example



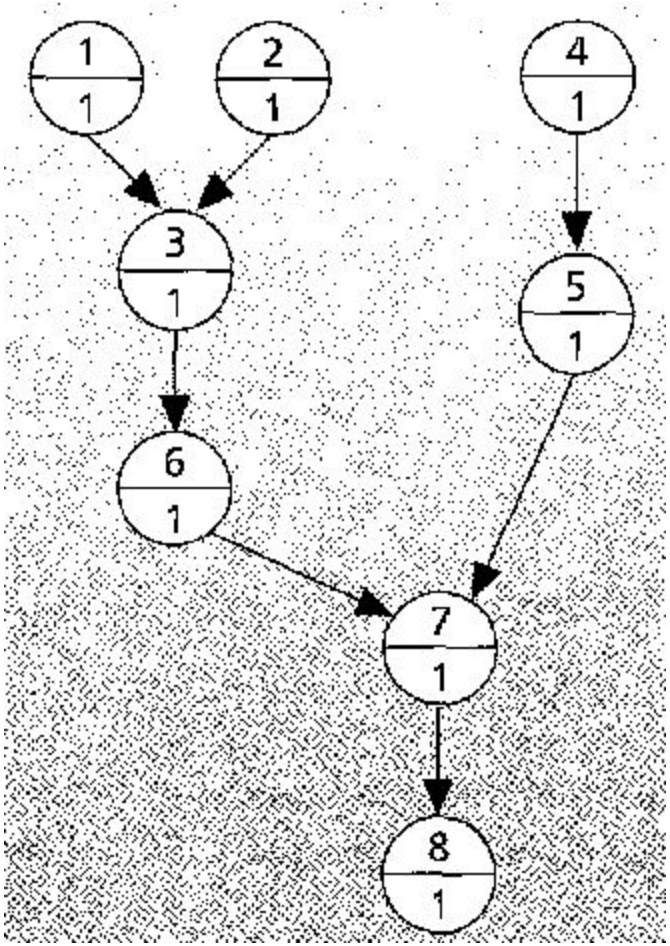
Task Priority →

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		

forests without communication……

- **Example**



Communication Model

- Two key component that contribute to the total completion cost is
  1. Execution Time
  2. Communication Delay
- **Model A**
  - Total cost = communication cost + execution cost
  - Where
    - Execution cost = schedule length
    - Communication cost = no. of message × cost per message
    - No. of message = the no. of node pairs (u, v) such that

Communication Model……

- **Example**

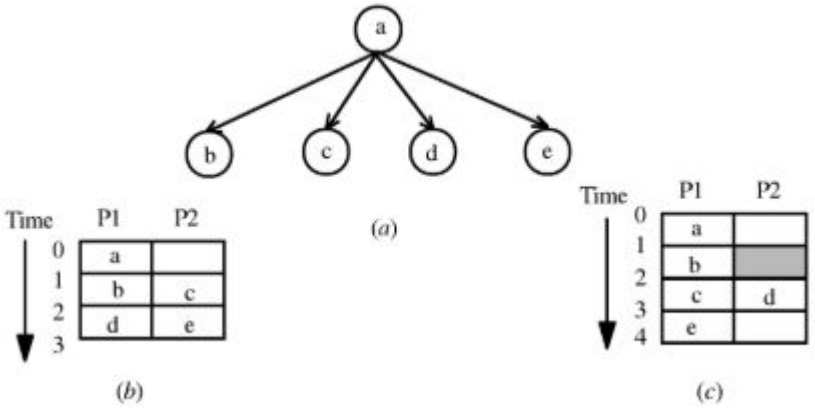


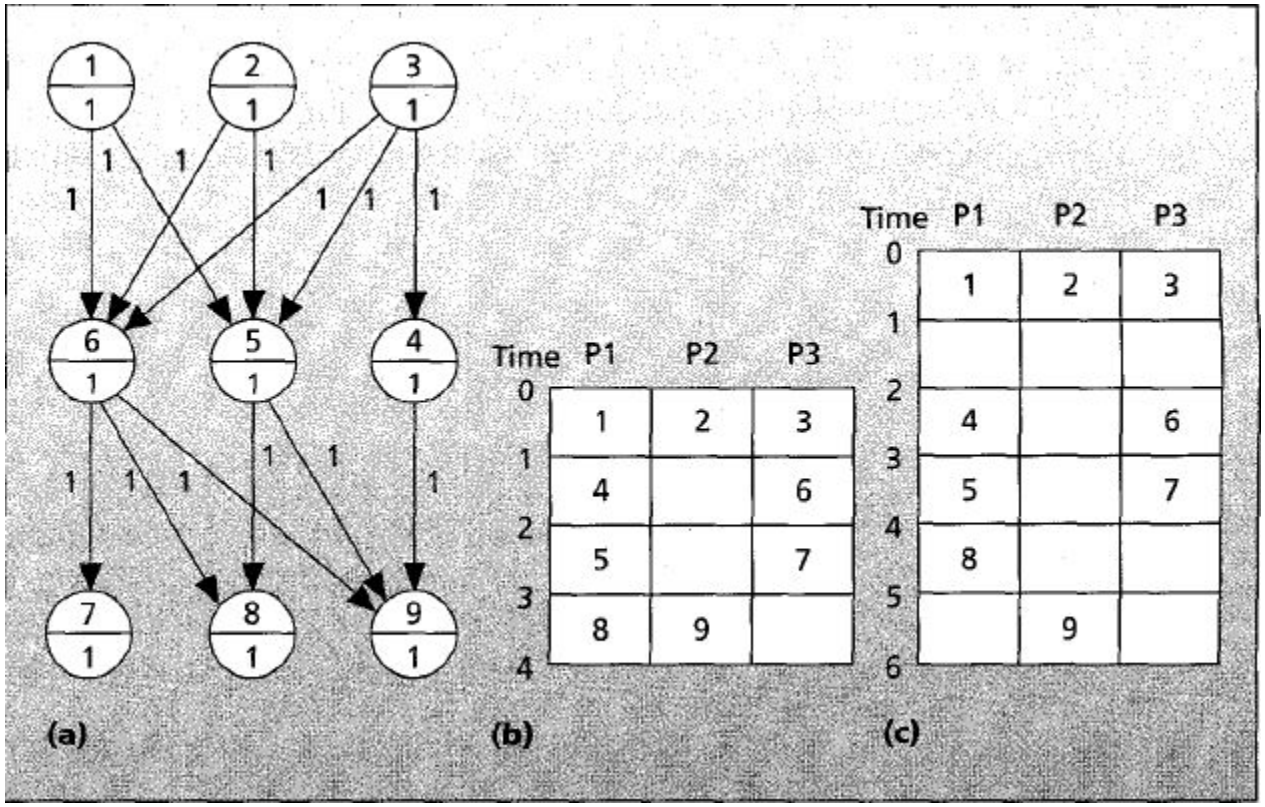
Fig a :- Task Graph
Fig b :- Task Schedule (***Model A and B***)
Fig c :- Task Schedule (***Model C***)

Total cost = communication cost + execution cost
Communication cost = no. of message × cost per message

- **Model A**
  - No. of message = |(a, c), (a, e)| = 2
  - Total cost = 2*1 + 3 = 5

Communication Model……

- **Exercise**



a. Task Graph b. Task Schedule (A & B) c. Task Schedule ©

Calculate total cost using Model A, Model B and Model C

Communication Model……

Model A

Execution cost = schedule length = 4 units of time

Number of messages = | (1, 6), (2, 5), (2, 6), (3, 4), (3, 5), (4, 9), (5, 9), (6, 8), (6, 9) | = 9

Communication cost = 9 * 1 units of time

Total cost = 4 + 9 = 13 units of time

Model B

Execution cost = schedule length = 4 units of time

Number of messages = | (P3, 1), (P1, 2), (P3, 2), (P1, 3), (P2, 4), (P2, 5), (P1, 6), (P2, 6) | = 8

Total cost = 8*1 + 4 = 12

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 (H. El-Rewini, T. Lewis, and H. Ali, *Task Scheduling in Parallel and DistributedSystems, Prentice Hall, Englewood Cliffs, N.J., 1994.*)
- **Node – depth**

with communication……

∴Algorithm

1. Given an in-forest ***G = (V, E)***, identify the set of siblings $S_1, S_2, \ldots\ldots\ldots\ldots, S_k$ 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 ∈ Si with the maximum depth***
   - $A_1 \leftarrow A_1 - (v, child(S_i))\ \forall v \in S_i$ ***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 $f = (V, A_1)$

with communication.....

: Example

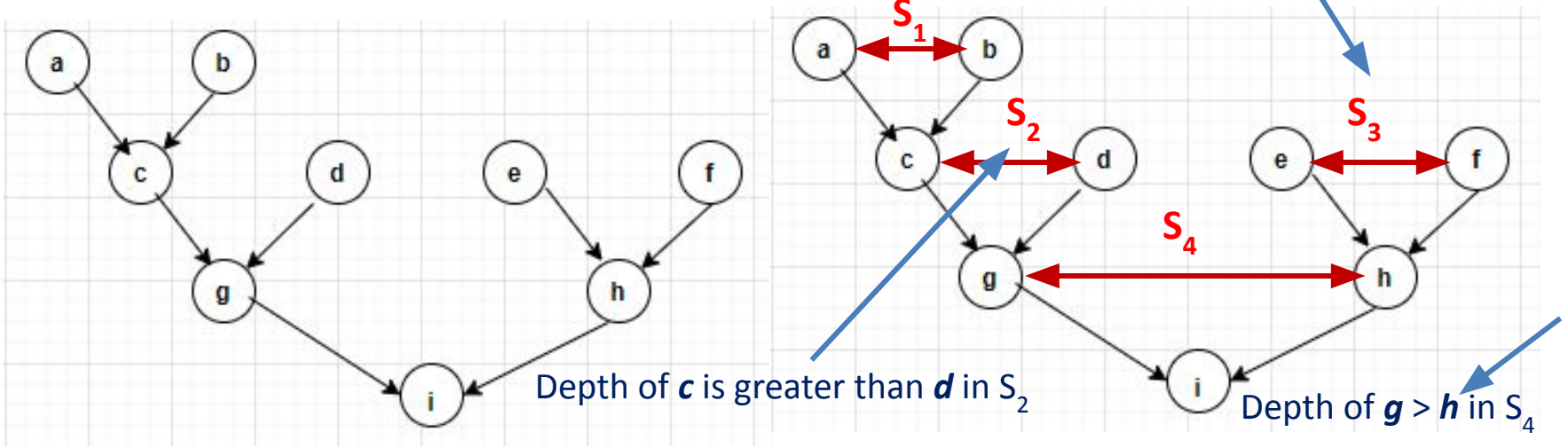


Fig a :- original task graph

Fig b :- formation of S_i

with communication……

- **Example…….**
- S_1 = {a, b} → b
- S_2 = {c, d} → c
- S_3 = {e, f} → f
- S_4 = {g, h} → g
- Case
- S_1 = {a, b} → b

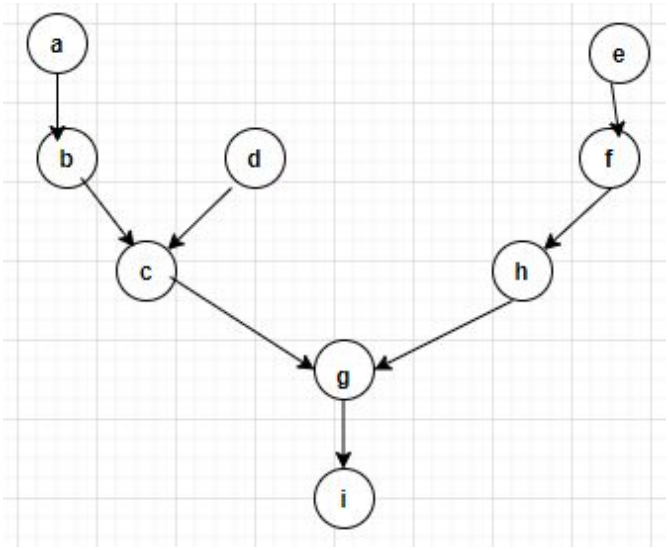
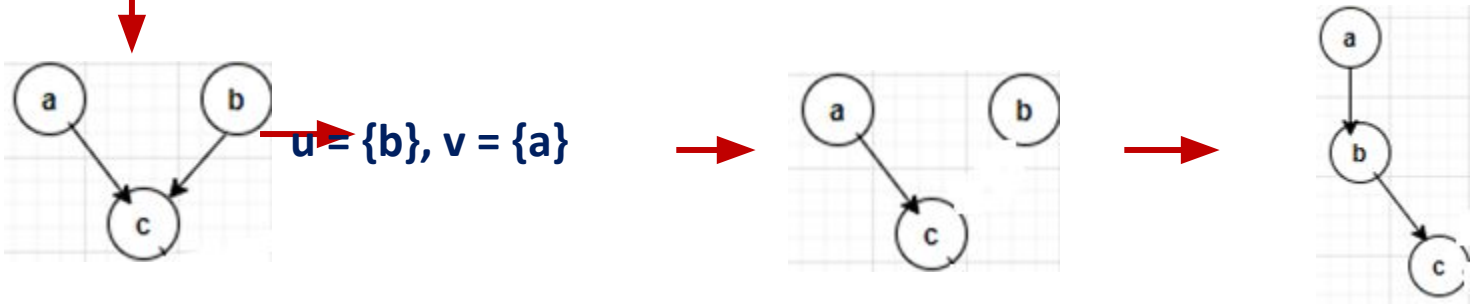


Fig c :- Augmented task graph



with communication……

- **Example…….**

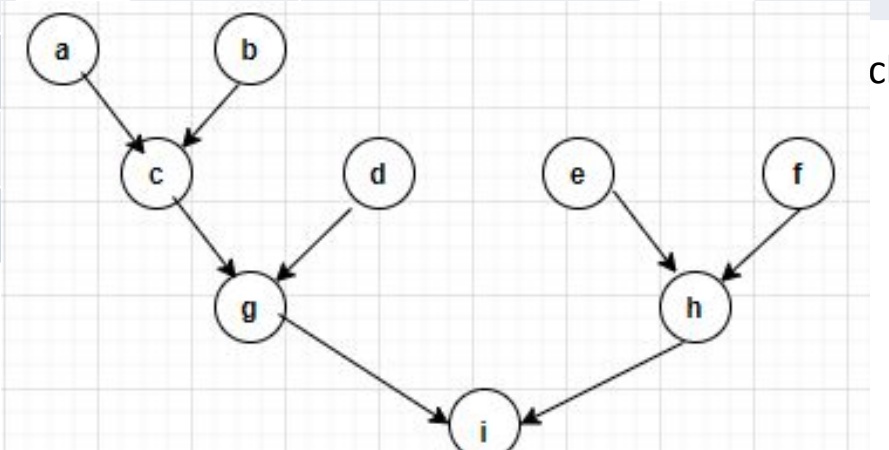
Node	Level
a	5
b	4
c	3
d	4
e	5
f	4
g	2
h	3
i	1

Fig d :- Task Priority

	P1	P2
0	a	e
1	b	d
2	f	c
3	h	
4	g	
5	i	

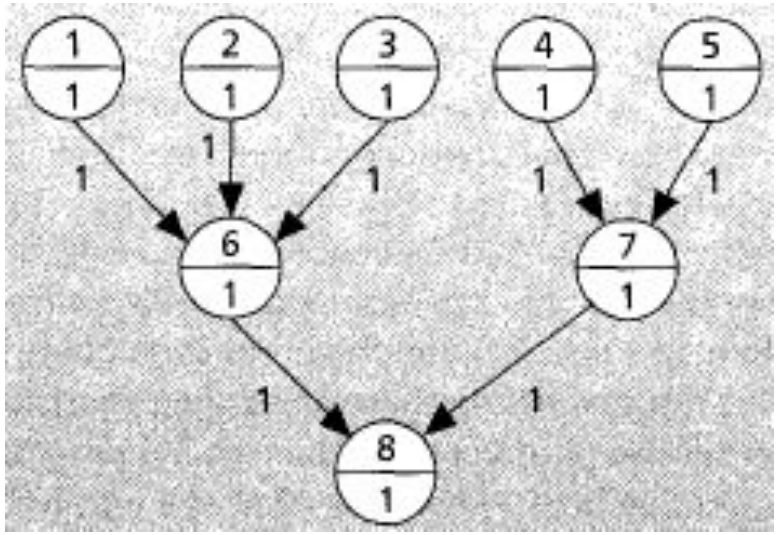
	P1	P2
0	a	e
1	b	d
2	c	f
3		h
4		g
5		i

cheduling after swap_all



Scheduling in forest / out forest with communication……

∴ **Exercise**



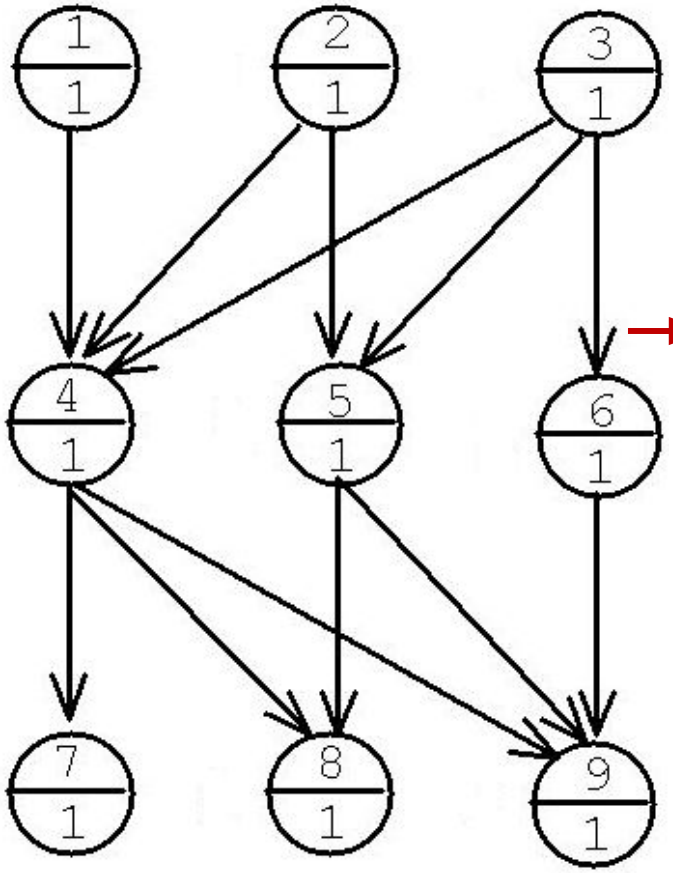
Scheduling Interval Ordered Task

- Used to indicate that the task graph which describes the precedence relation among the tasks in an interval order
  1. Without Communication
  2. With Communication
- **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

Scheduling Interval Ordered Task

...

- **Without communication**



Task Graph

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

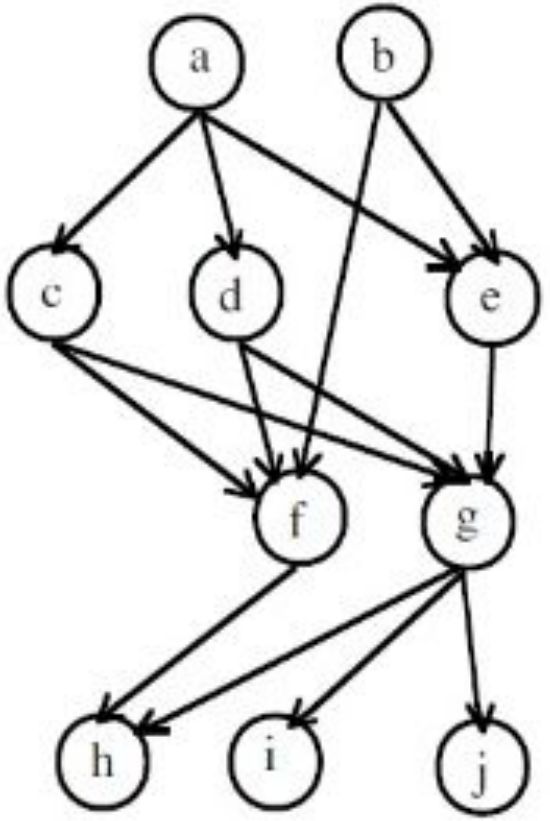
Task Priority

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

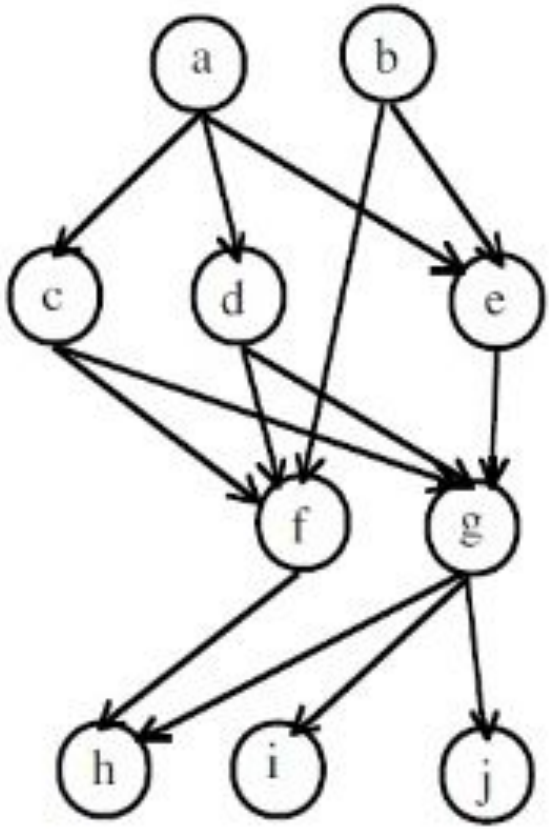
Task Scheduling

Scheduling Interval Ordered Task ...

- **Without communication (Ex)**



Scheduling Interval Ordered Task ...



Task Graph

Node	Number of Sucessors
a	8
b	6
c	5
d	5
e	4
f	1
g	3
h	0
i	0
j	0

Task Priority

Time	P1	P2	P3
0	a	b	
1	c	d	e
2	g	f	
3	h	i	j
4			

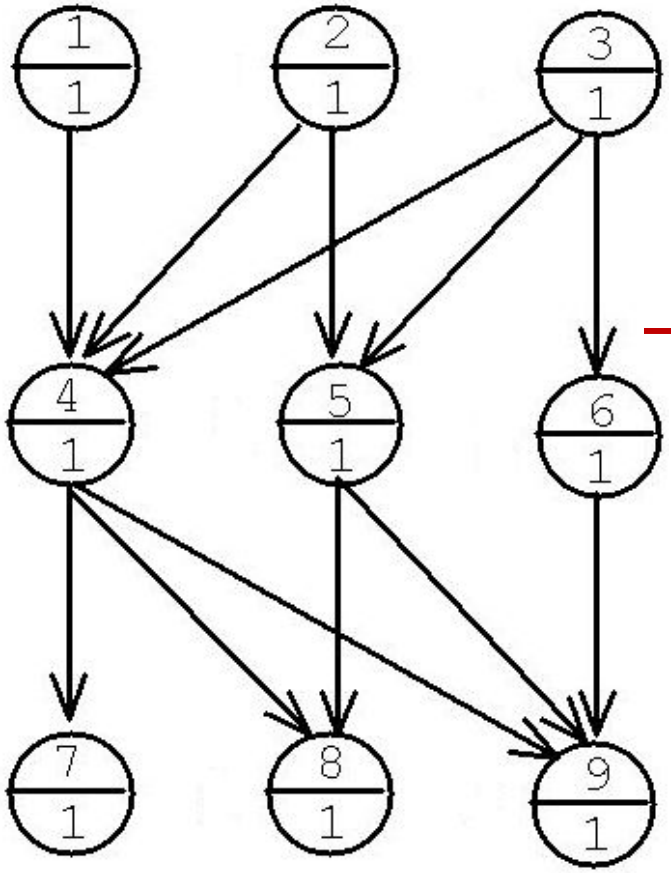
Gantt Chart

Scheduling Interval Ordered Task ...

- **With Communication**
- ***start_time (v, i, f)*** → earliest time at which task ***v*** can start execution on processor P_i, in schedule ***f***
- ***task(i, t, f)***→task schedule on processor 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

Scheduling Interval Ordered Task ...

- **Example**



Task Graph

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

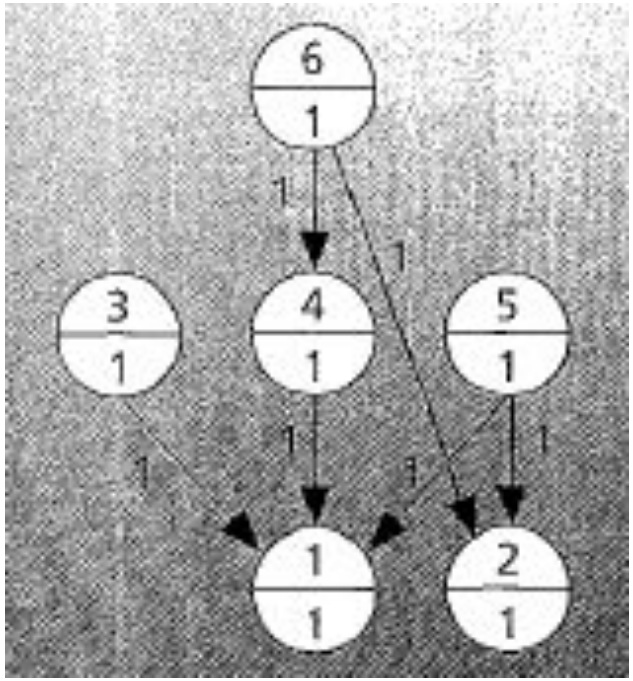
Task Priority

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

Task Scheduling

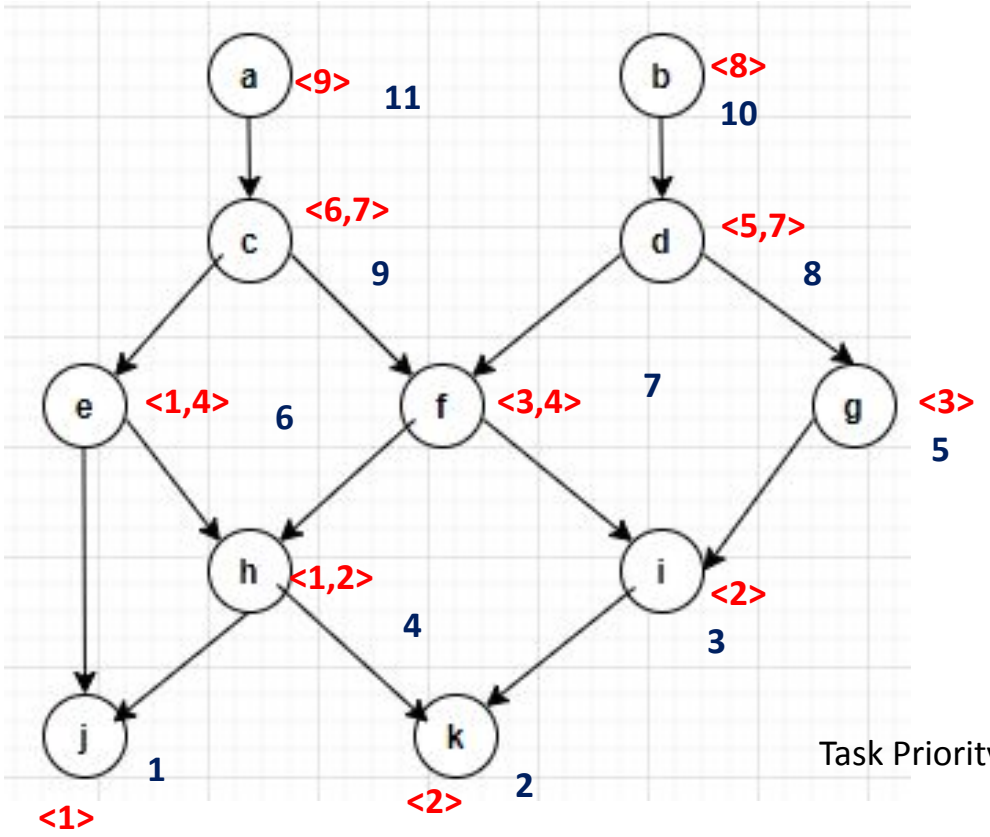
Scheduling Interval Ordered Task ...

- Exercise



Two Processor Scheduling

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



Task Graph

Node	Label
a	11
b	10
c	9
d	8
e	6
f	7
g	5
h	4
i	3
j	1
k	2

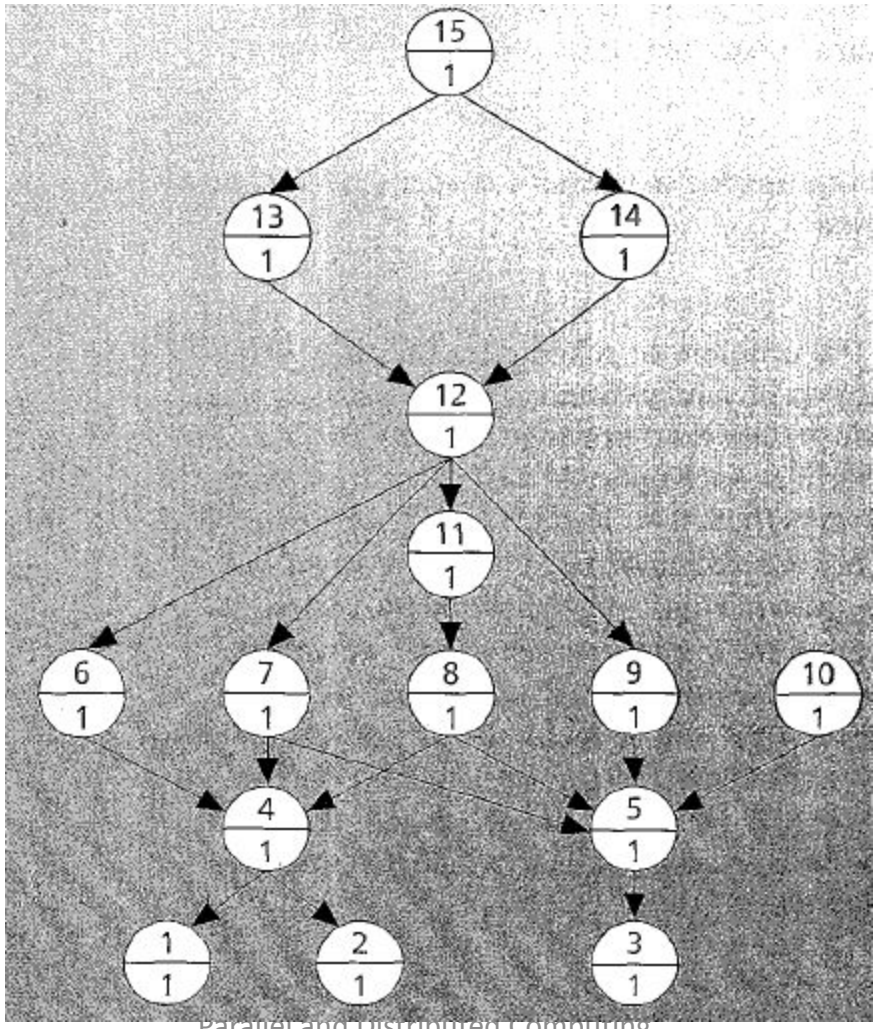
Task Priority

	P1	P2
0	a	b
1	c	d
2	f	e
3	g	h
4	i	j
5	k	

Task Scheduling

Two Processor Scheduling

- **Ex**

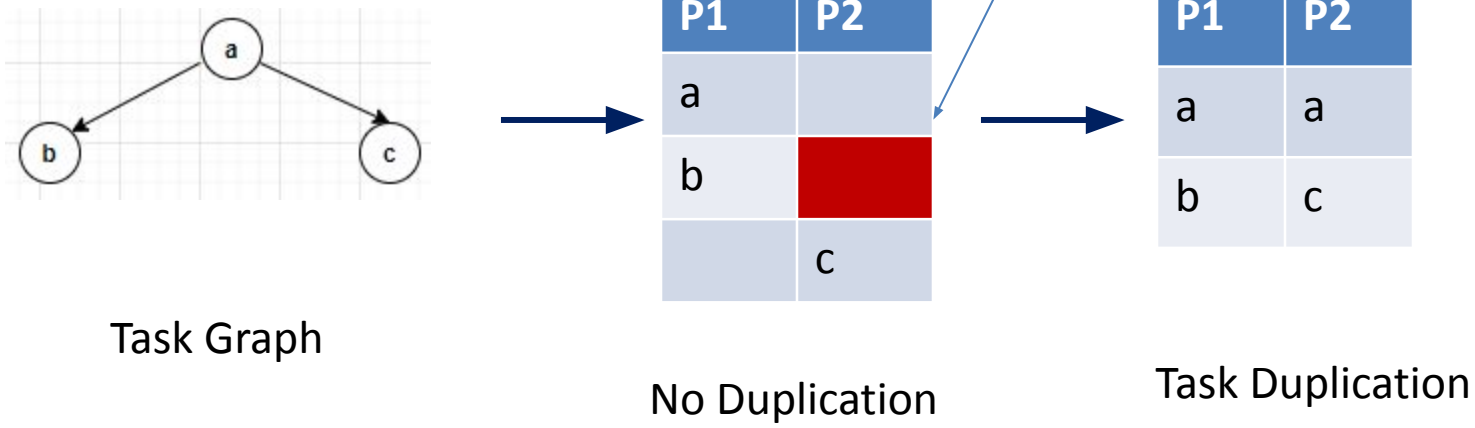


Scheduling Heuristic Algorithm

- **Priority based scheduling**
  1. Each node is assigned priority
  2. As long as priority queue is not empty
     - A task is obtained from front of queue
     - An idle processor is selected
     - When all the immediate predecessors of a particular task are executed, the successor is now ready and inserted to the queue
- **Scheduling using clustering heuristics**
  1. Cluster merging → if no. if clusters is greater than no. of processors available

Scheduling Heuristic Algorithm…..

- **Task allocation using duplication**



End of Session
Unit 2.4

Unit 2.5 :- Checkpoint in Parallel and Distributed System

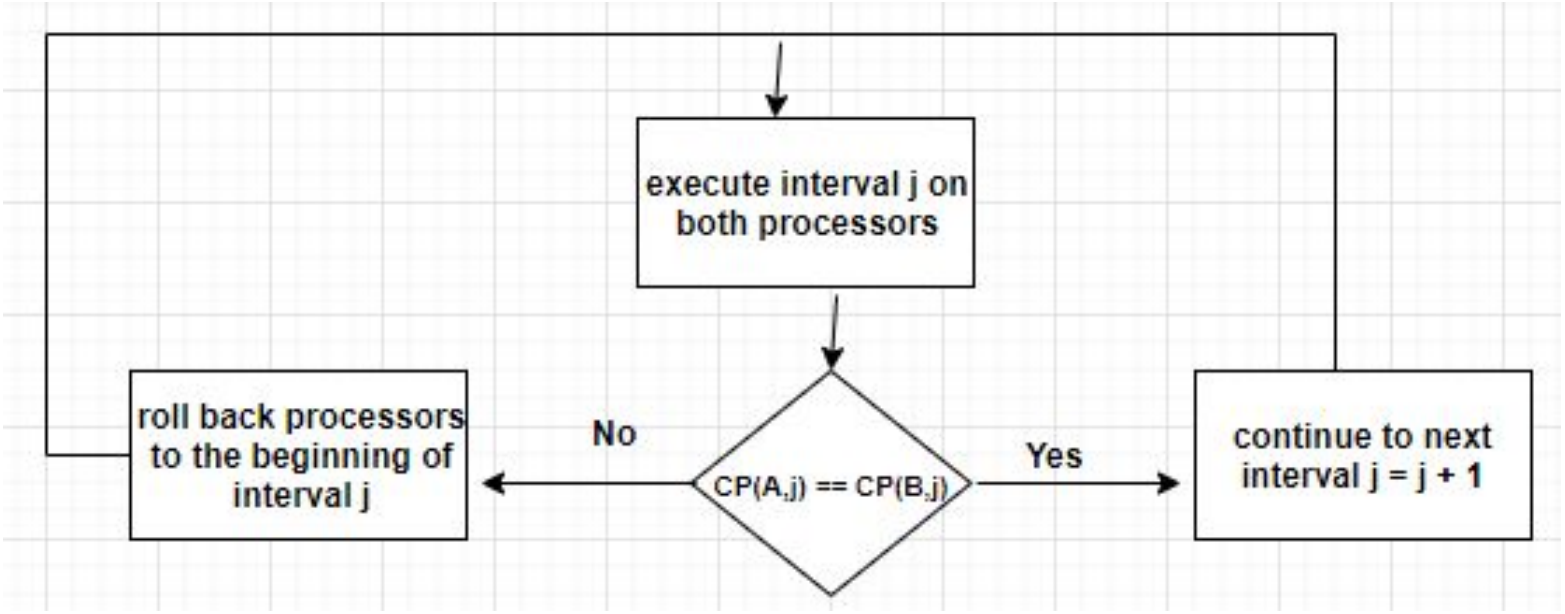
Checkpointing in Parallel and Distributed System

- **Fault Detection**
  - Process of recognizing that an error has occurred in the system
- **Fault Location**
  - After the error in the system is detected, the location of the part in the system that cause the error is identified
- **Fault Containment**
  - After the fault is located, the faulty part is isolated to prevent the possible propagation
- **Fault Recovery**

Checkpointing using task duplication

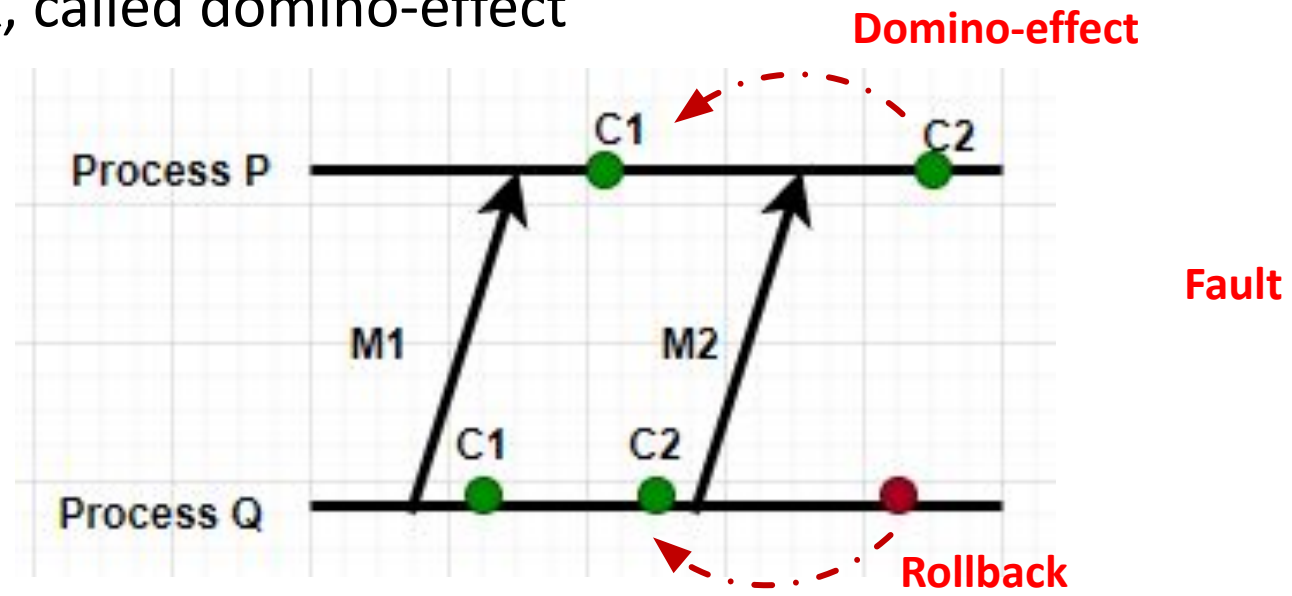
- Task duplication is analogous to shadowing techniques for recovery in database transaction system
- Compares the state of same task that are executed in parallel on different processors
- Here fault detection is achieved by duplicating the task into two or more processors and comparing the states of the processors at the checkpointing
- Matching states is an indication of a correct

Checkpointing with simple rollback



Techniques for consistent Checkpointing

- When a fault is detected in one of the processors, the process is rolled back to its last saved check point
- In order to maintain consistent checkpoint, some other processes that communicates with this task need to be rolled back, called domino-effect



End of Session
Unit 2.5

Unit 2.6 :- Open Distributed System

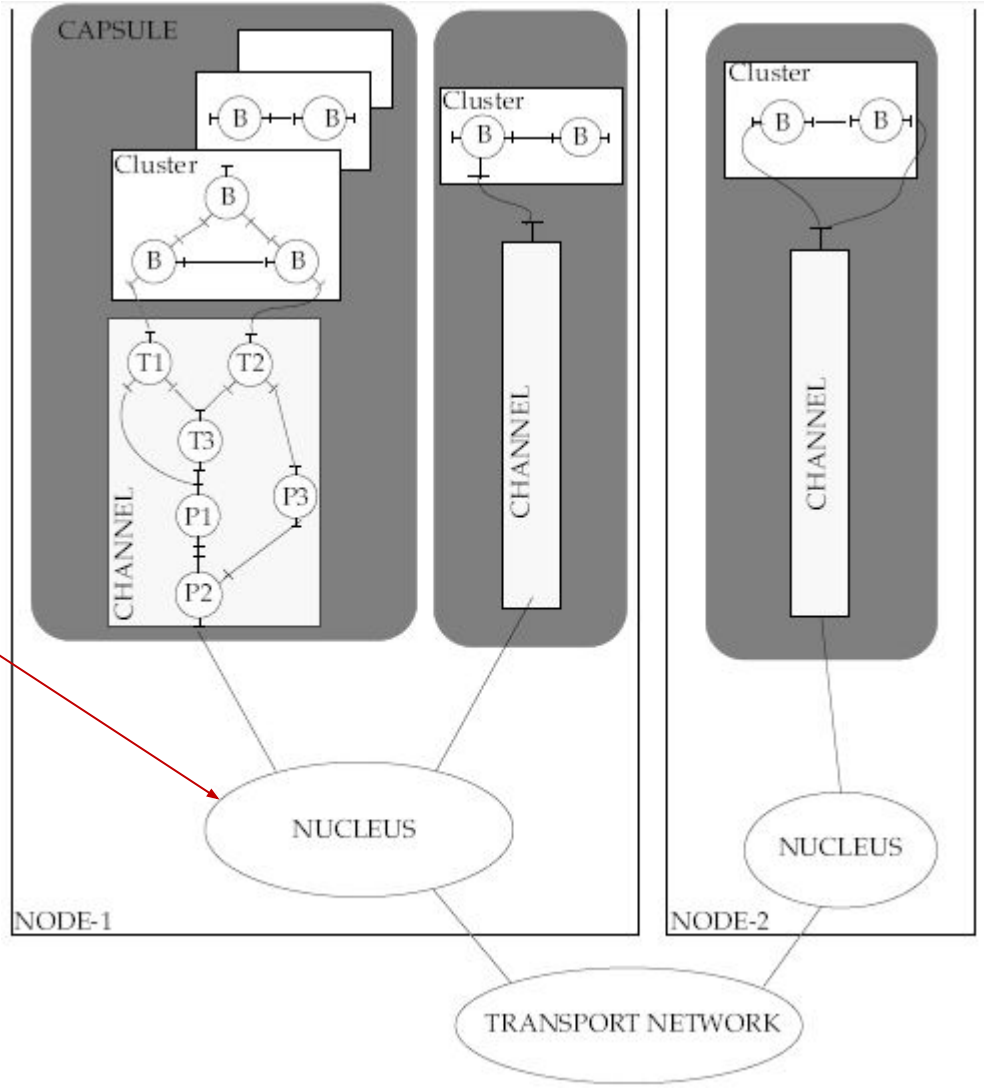
Open Distributed System

- Distributed systems are complex structure composed of many types of hardware and software components
- Eg Telecommunication system
- Heterogeneous environment
- **Components of open distributed system architecture**
  1. **End user viewpoint** → information content by view of user
  2. **Enterprise viewpoint** → describing the needs of

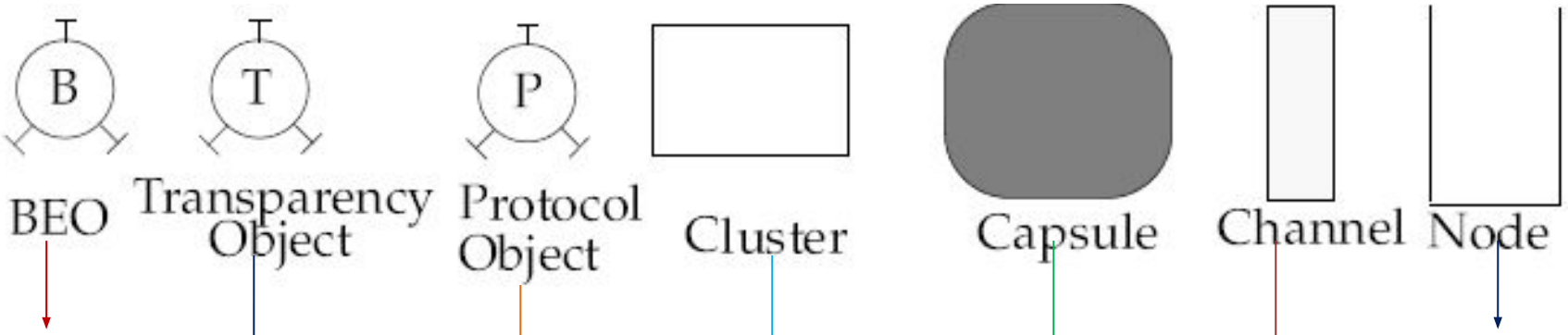
Open Distributed System……

- **Engineering Model**

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



Open Distributed System……



c Engineering Objects are the run time representations of computational object

Single computer system

Special purpose model

Collection of BEO, TO, PO in a node

Responsible to make a bridge to communicate objects

Runtime binding between objects

Concepts of related modules

End of Session
Unit 2.6

Unit 3.1 :- Fundamentals of Parallel Algorithm

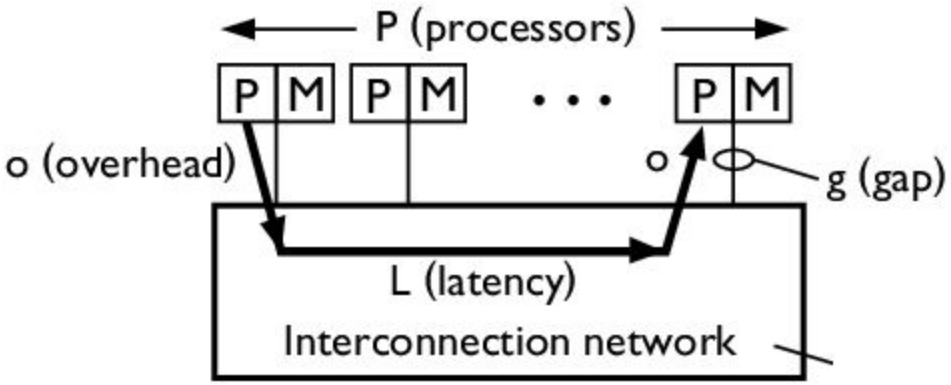
Fundamentals of Parallel Algorithm

- **Models of parallel computation**

1. PRAM Model
   - Complexity measure used for evaluating the performance of a PRAM algorithm is $T_p(n) = O(\frac{T(n)}{p})$
     - Where T(n) → best sequential time
     - p → no. of processors
2. Shared Memory Model
   - Retains single address space of PRAM but assumes that the processors execute their operations asynchronously (like multiprogramming)
3. Network Model
   - Topology of the inter-connection network as the basis

Fundamentals of Parallel Algorithm…

- **LogP Model**



- L → upper bound of the latency occurred in communicating the message
- o → overhead, length of a time that a processor is engaged in submission of a message
- g → gap, minimum time interval between consecutive node message transmission and submission
- P → no. of processors

Balanced Trees

- A common strategy for designing parallel algorithms is to build a balanced tree on the input elements and traverse the tree, storing some information on internal node

$input \rightarrow n = 2^h$ $output \rightarrow sum = \sum_{i=0}^{n-1} A[i]$

- **Eg: Computing the sum of elements in the array** $0 \leq i \leq \frac{n}{2^h - 1}$
- Algorithm(Data Parallel Sum)

– Begin

Divide and Conquer

- Partitioning the input into several partitions of almost equal size
- Solve each in parallel
- Merge the solution
- Eg: Quick Sort

End of Session
Unit 3.1

Unit 3.2 :- Graph Theoretic Concepts and Notations

Graph Theoretic Concepts and Notations

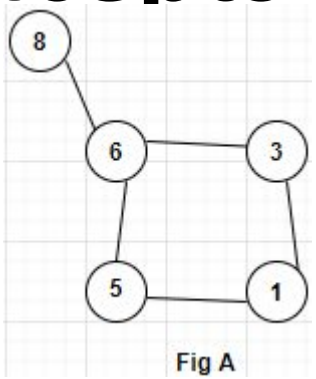


Fig A

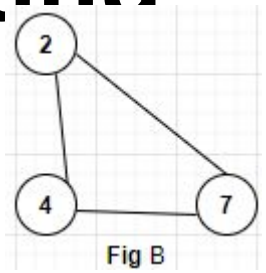


Fig B

- Undirected graph
- Sub-graph
- Connected graph
  - If there is a path between every pairs of vertices
- Bi-connected graph
  - Removing an arbitrary vertex along with all edges incident to it is still connected
- Cut-vertex
  - A vertex ***v*** of G is cut vertex if the graph induced by ***v*** is not connected

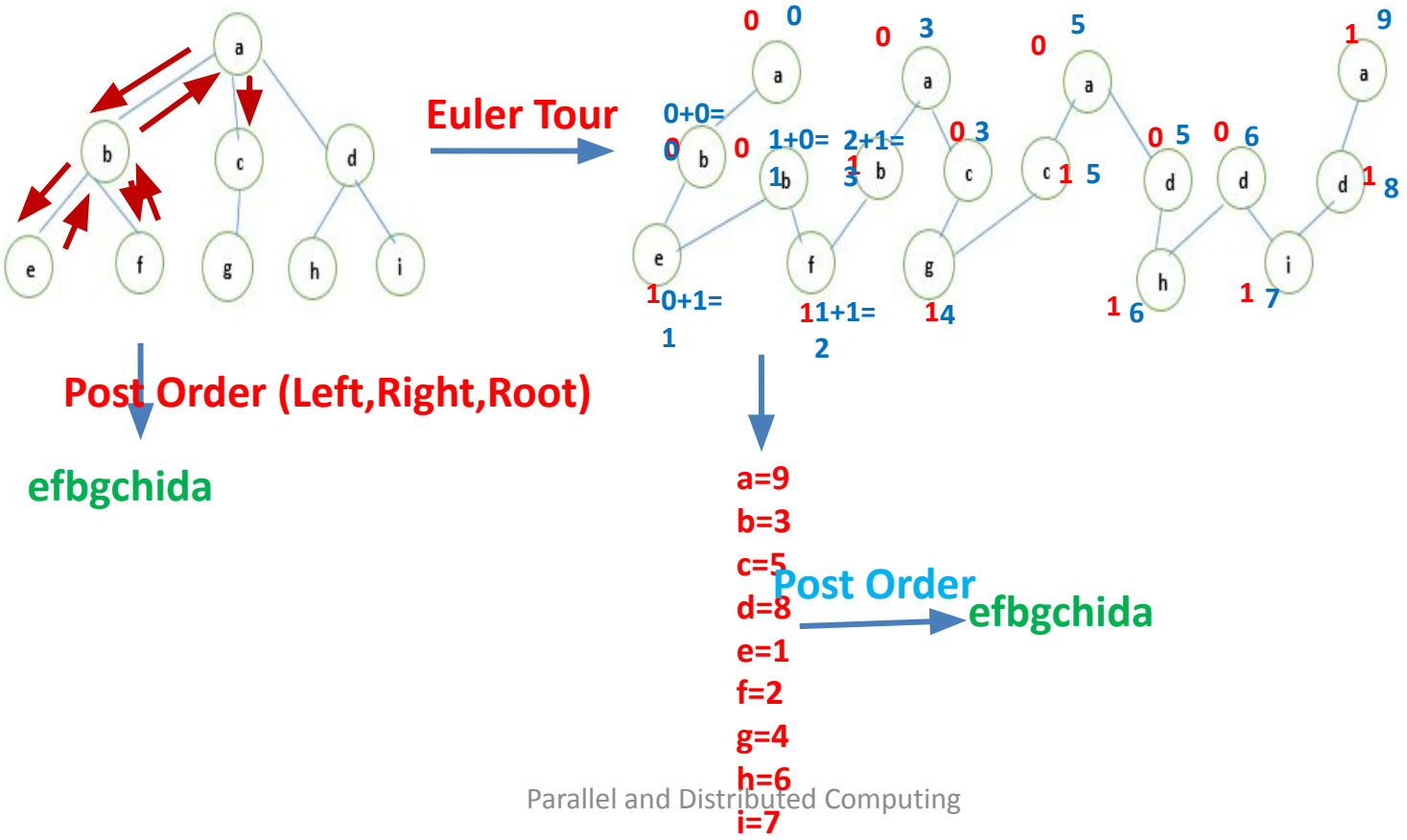
Tree Graph

- **Computing a post order numbering**
  - Left, Right, Root
  - Given a rooted tree ***T*** along its Euler Tour, the task of computing a post order numbering of the nodes ***T*** can be performed as follows

1. For every node ***v*** of ***T***, assign a weight of ***0*** to v^1, v^2,……..,$v^{d(v)}$ and weight of ***1*** to $v^{d(v)+1}$
2. Compute the weighted rank of every node in the Euler Tour
3. For every node $v^{d(v)+1}$, the corresponding weighted rank is the post order of ***v***

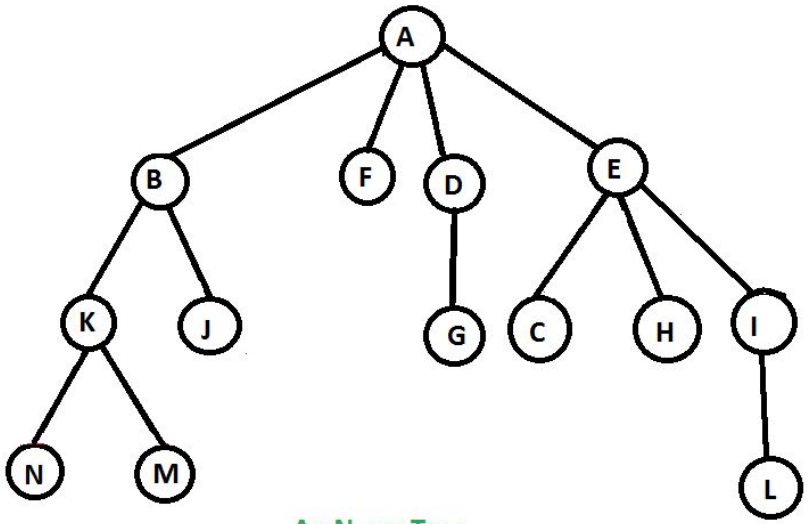
Computing a post order numbering…..

- **Example**



Computing a post order numbering…..

- **Exercise (Compute post order of following tree)**



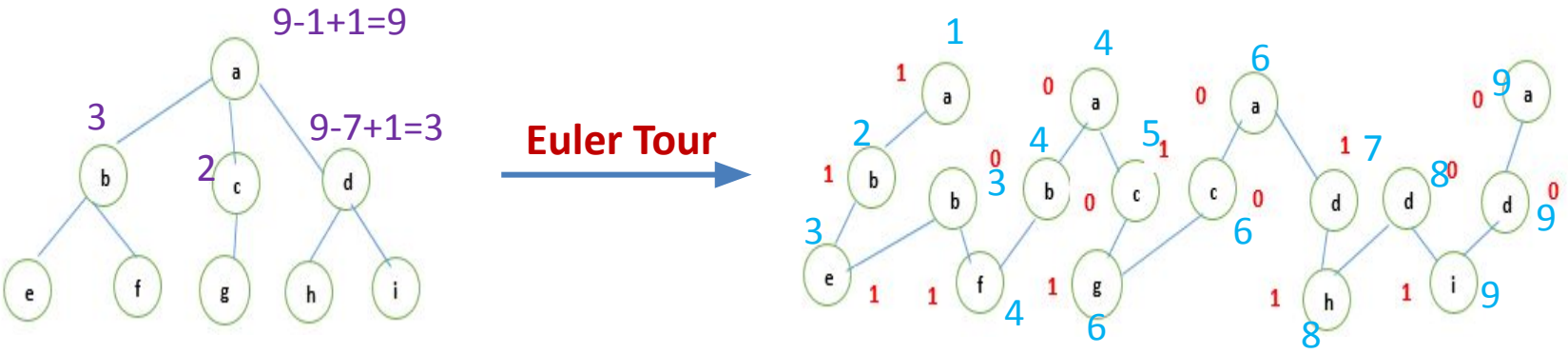
An N-ary Tree.

Computing the number of descendants

- Given the rooted tree T and Euler tour, assign a weight of 1 to u^1 and weight of 0, to $u^2, u^3, \ldots\ldots\ldots, u^{d(u)+1}$
- Let $r(u^1)$ and $r(u^{d(u)+1})$ be the corresponding values of prefix sum at u^1 and $u^{d(u)+1}$ respectively
- The number of descendants of u is exactly, $r(u^{d(u)+1}) - r(u^1) + 1$

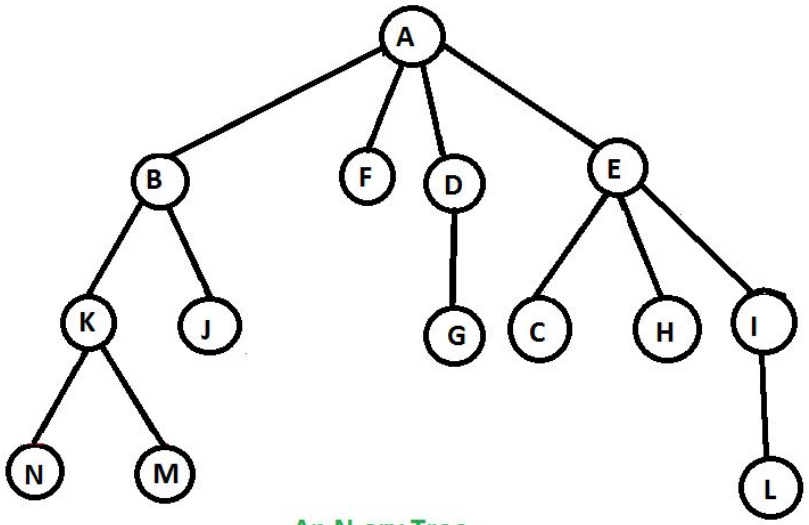
Computing the number of descendants…

- **Example**



Computing the number of descendants…

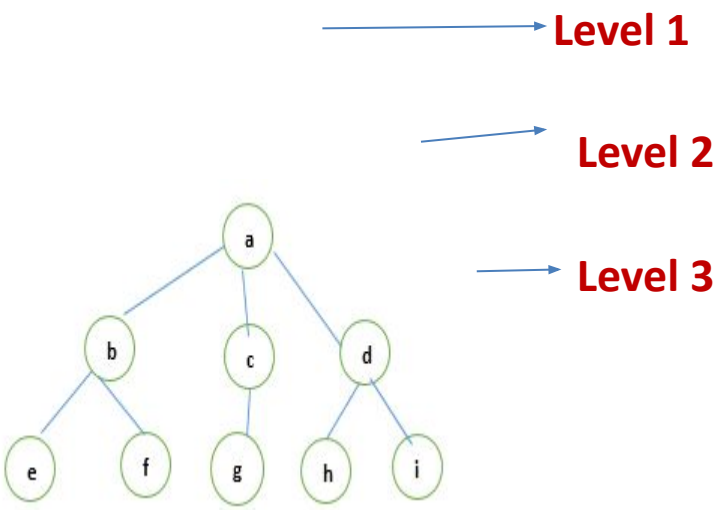
- **Exercise (Compute the number of descendants of every node in following tree)**



An N-ary Tree.

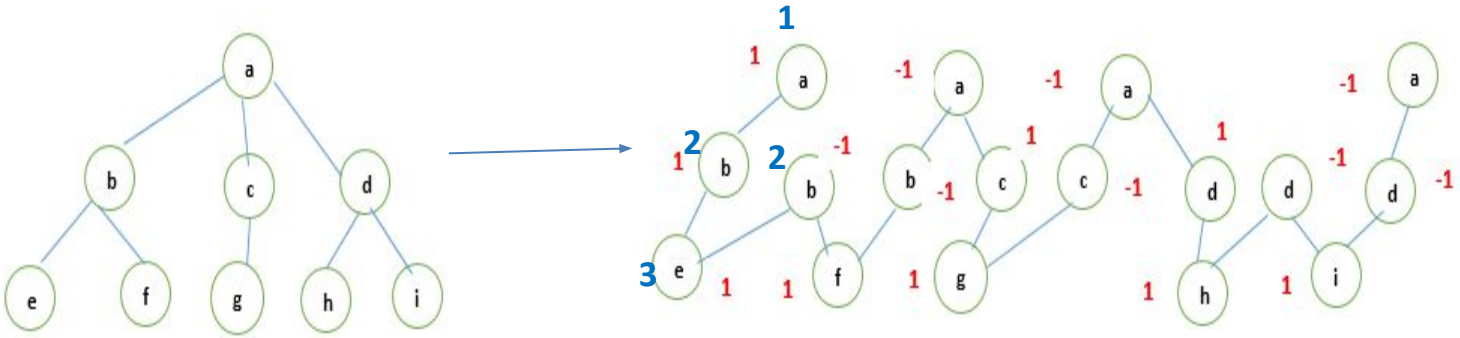
Level Computation

- The level of a node ***v*** of ***T*** is defined as the number of edges on the path joining ***v*** to the root
- Consider a tree ***T*** with Euler tour
- Assign a weight of 1 to v^1 and weight of -1 to every v^j where $j \neq i$
- Example



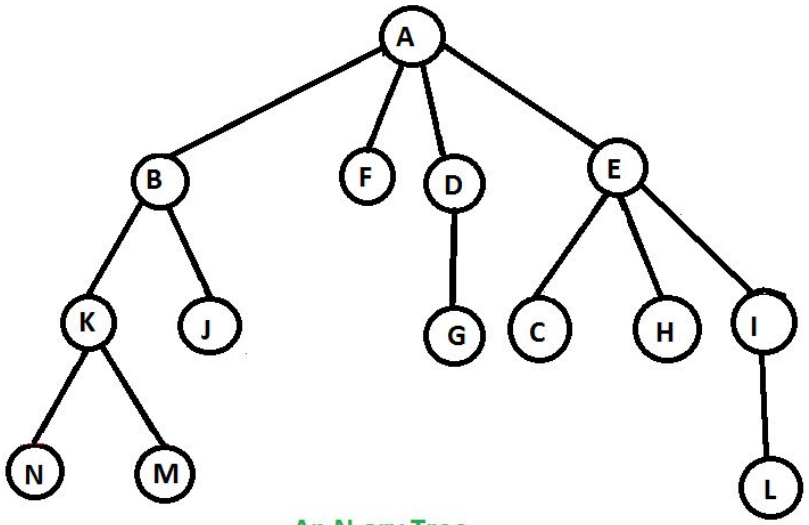
Level Computation……

- Example



Level computation……

- **Exercise (Compute the level f every node in following tree)**

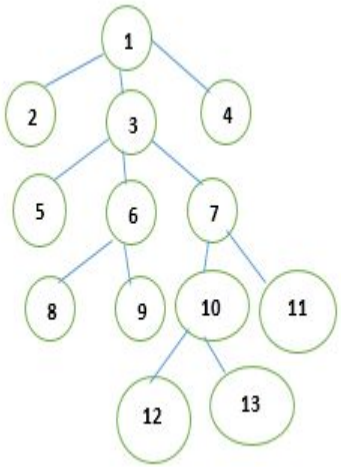


An N-ary Tree.

LCA (Lowest Common Ancestor) Computation

- Let ***T*** be a rooted tree with ***n*** nodes
- The lowest common ancestor between two nodes ***v*** and ***w*** is defined as the lowest node in ***T***, that has both ***v*** and ***w*** as descendants
- **Example**

LCA(9,12) = 3

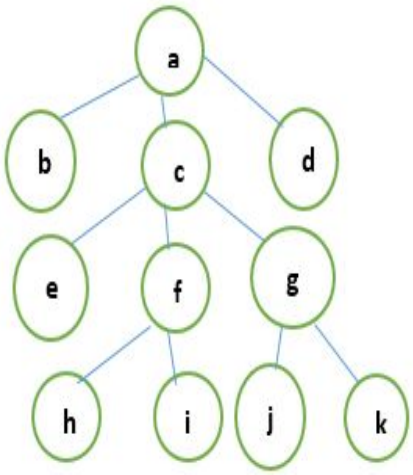


LCA (Lowest Common Ancestor) Computation…

- **Algorithm**
  - Compute pre-ordering numbering of the nodes in T
  - For all nodes u of T, do in parallel
    - Computer l(u), r(u)
  - For all $v \in [l(u), r(u)]$, do in parallel
    - A[u, v] = u
  - For all $x \in [l(w_i), r(w_i)]$ and $y \in [l(w_j), r(w_j)]$, $1 \leq i \neq j \leq d(u)$ do in parallel
    - A[x, y] = u
  - Return A

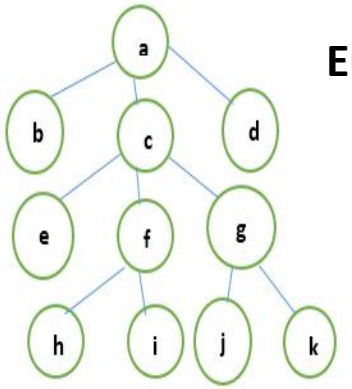
LCA (Lowest Common Ancestor) Computation…

- **Example**

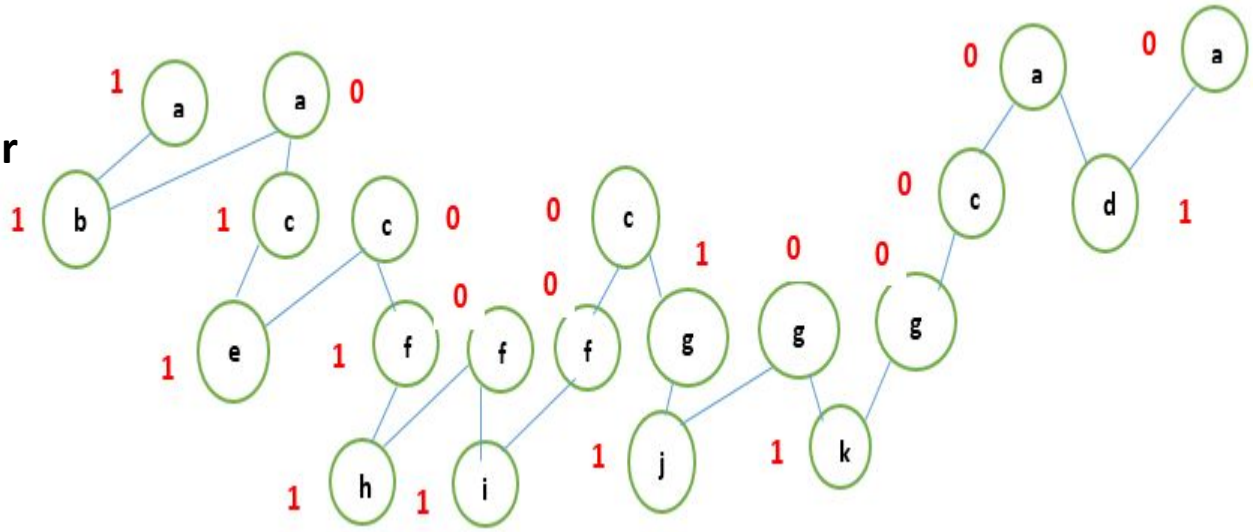


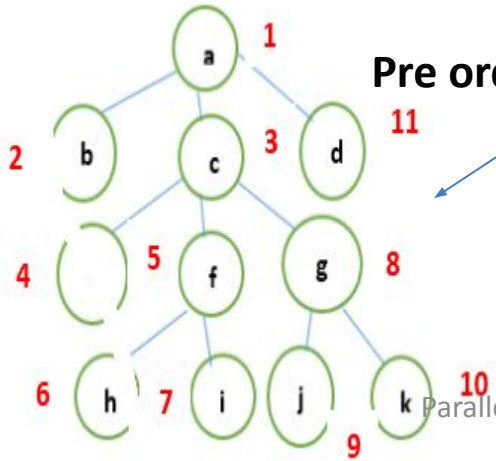
LCA (Lowest Common Ancestor) Computation…

- **Example**



Euler Tour

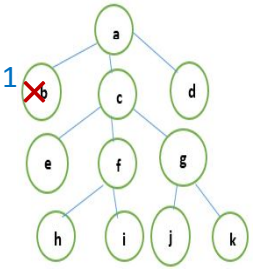




Pre ordering Number

l(a)= 1 , r(a) = 11; l(b)= 2 , r(b) = 2
l(c)= 3 , r(c) = 10 ; l(d)= 11 , r(d) = 11
l(e)= 4 , r(e) = 4 ; l(f)= 5 , r(f) = 7
l(g)= 8 , r(g) = 10 ; l(h)= 6 , r(h) = 6
l(i)= 7 , r(i) = 7 ; l(j)= 9 , r(j) = 9
l(k)= 10 , r(k) = 10

LCA (Lowest Common Ancestor) Computation…



(2,3) (2,4) (2,5) (2,6) (2,7) (2,8) (2,9) (2,1

l(a)=1,r(a)=11 For "a"

(2,2) (3,10) (11,11)

bc=(2,2)->(3,10)
bd=(2,2)->(11,11)
cd=(3,10)->(11,11)

(2,11)
(3,11) (4,11) (5,11) (6,11) (7,11) (8,11) (9,11) (10,11)

	1	2	3	4	5	6	7	8	9	10	11
1	A	A	A	A	A	A	A	A	A	A	A
2	A	B									A
3	A		C	C	C	C	C	C	C	C	
4	A		C	E							
5	A		C		F	F	F				
6	A		C		F	H					
7	A		C		F		I				
8	A		C					G	G	G	
9	A		C					G	J		
10	A	A	C					G		K	
11	A										D

LCA (Lowest Common Ancestor) Computation…

	1	2	3	4	5	6	7	8	9	10	11
1	A	A	A	A	A	A	A	A	A	A	A
2	A	B	A	A	A	A	A	A	A	A	A
3	A	A	C	C	C	C	C	C	C	C	A
4	A	A	C	E	C	C	C	C	C	C	A
5	A	A	C	C	F	F	F	C	C	C	A
6	A	A	C	C	F	H	F	C	C	C	A
7	A	A	C	C	F	F	I	C	C	C	A
8	A	A	C	C	C	C	C	G	G	G	A
9	A	A	C	C	C	C	C	G	J	G	A
10	A	A	C	C	C	C	C	G	G	K	A
11	A	A	A	A	A	A	A	A	A	A	D

End of Session
Unit 3.2

Unit 3.3 :- Data Parallel Algorithms

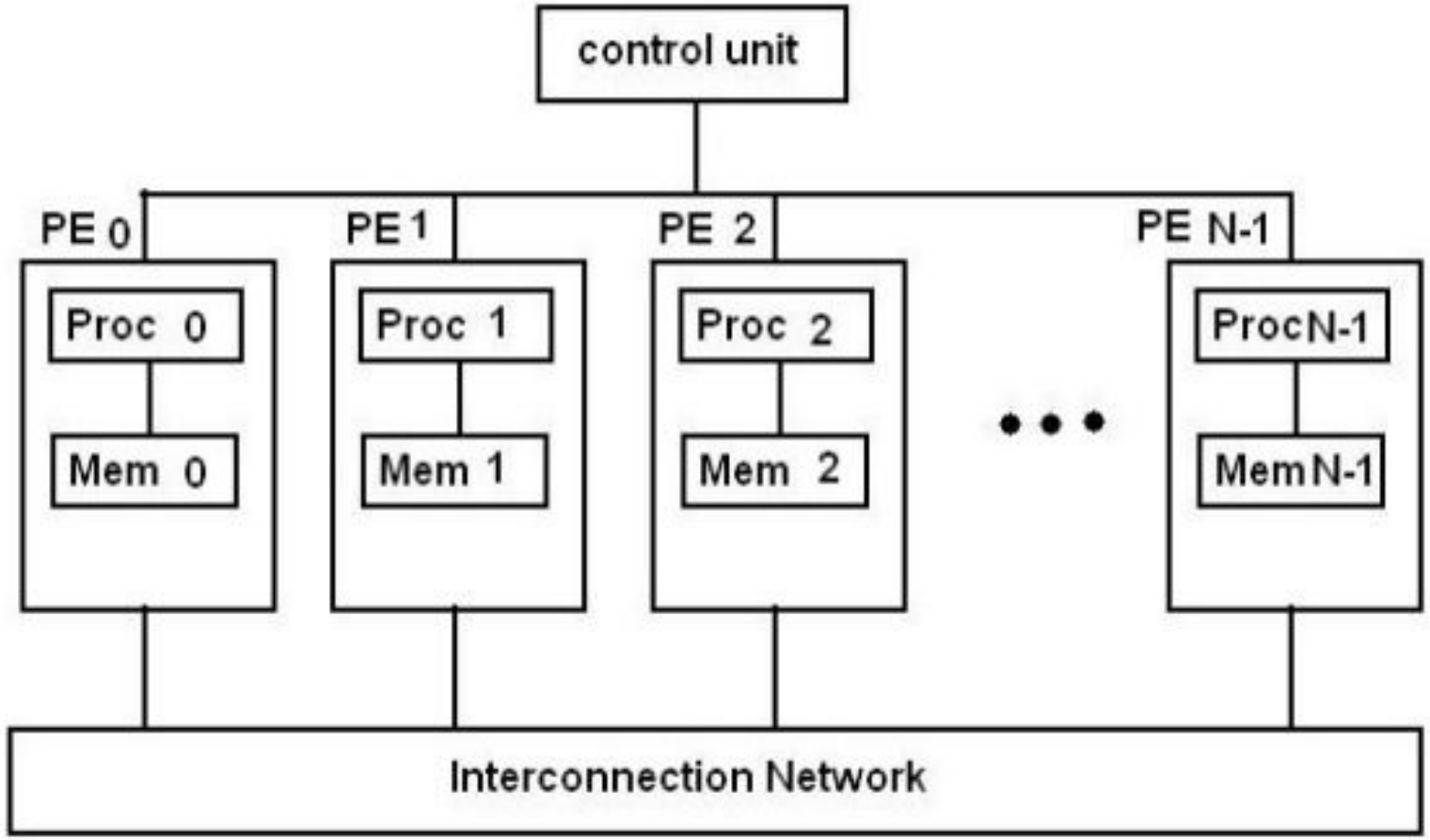
Data Parallel Algorithms

- Is a model of parallel computing in which the same set of instructions is applied to all the elements in a data set

Machine Model

- Based on SIMD machine model
- A single sequence of instruction is executed simultaneously in an arbitrary set of processors with each processor operating on its own data
- Consists of N processors, N memory unit, and a control unit and an interconnected network

Machine Model……

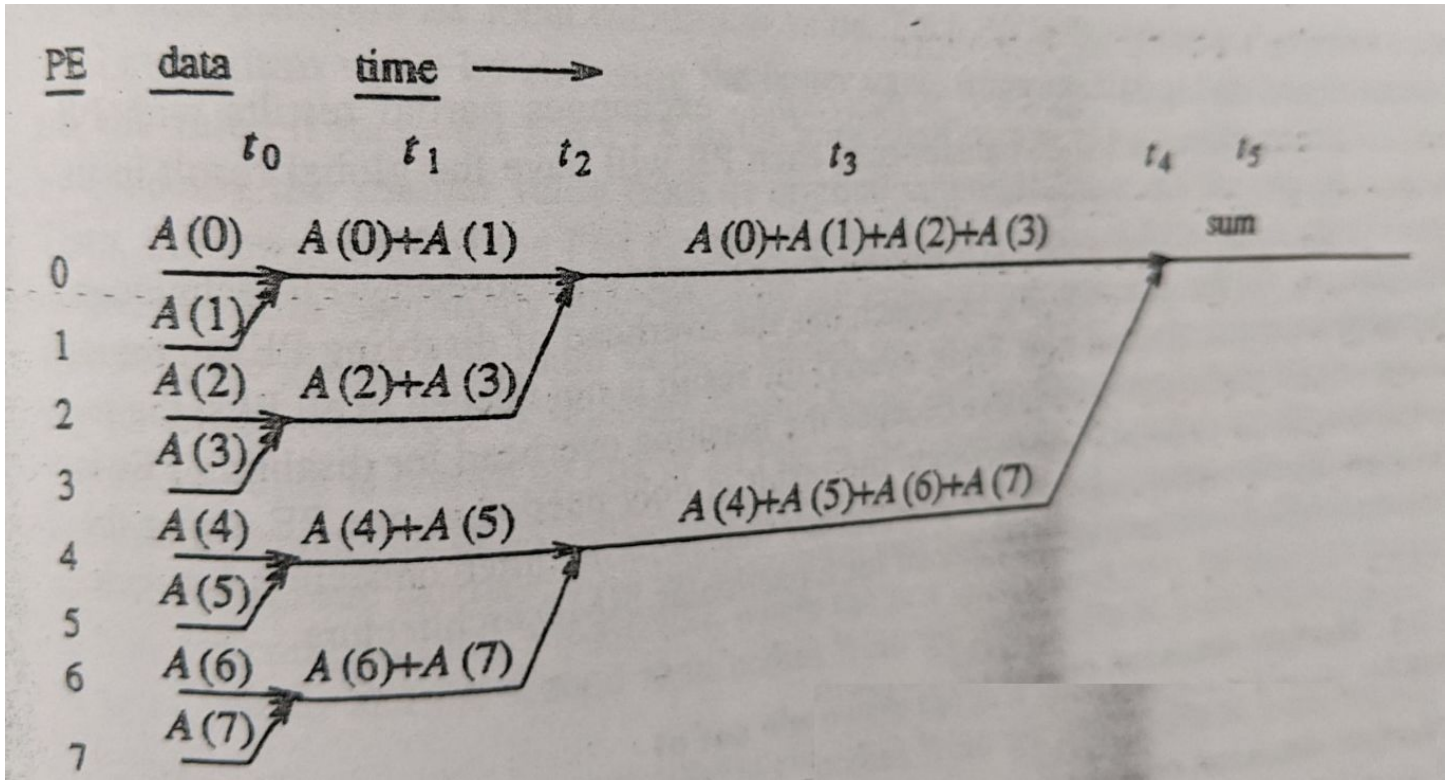


CU / PE Overlap

- CU initiates parallel computations by sending blocks of SIMD code to the CU instruction broadcast queue
- Once in the queue, each SIMD instruction is broadcast to all the PE's, while CU can be performing its own computation
- This property is called CU / PE overlap

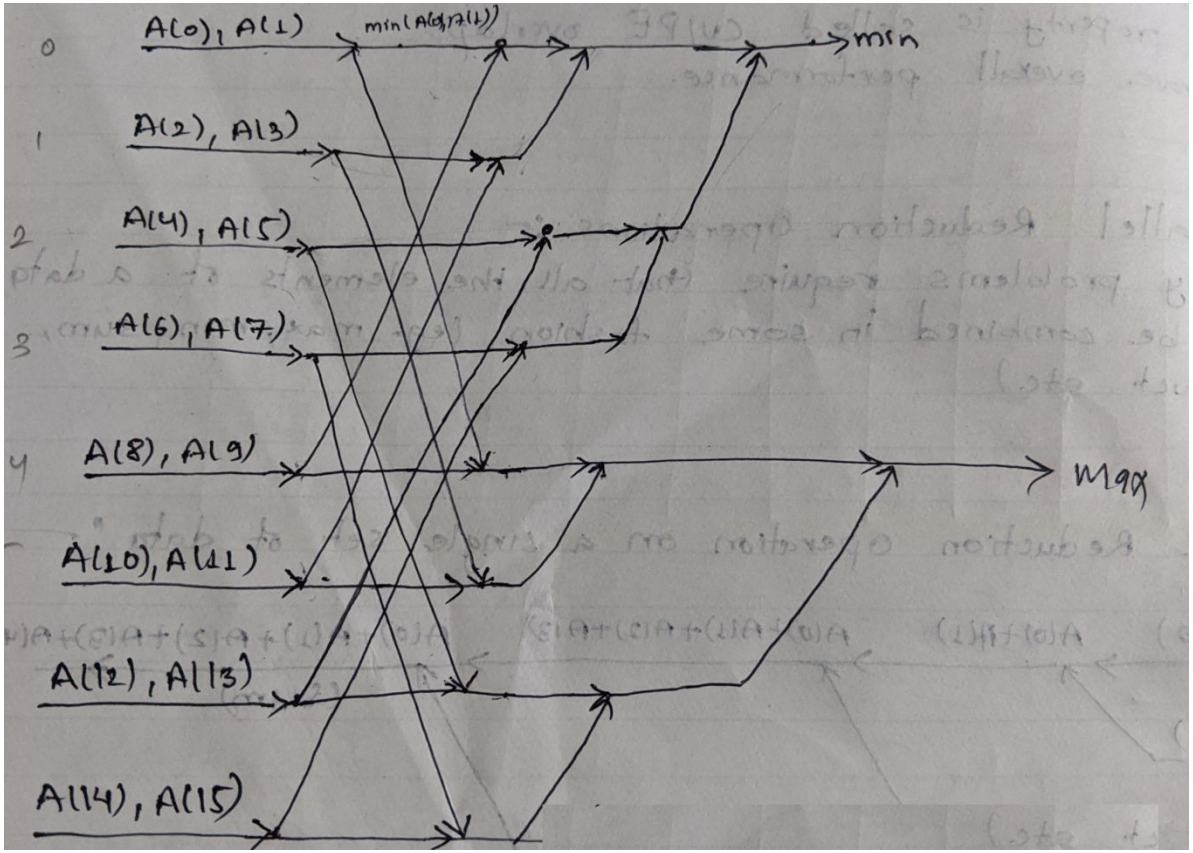
Parallel Reduction Operations

- **Single Reduction Operation on a single set of data**



Parallel Reduction Operations……

- **Multiple Reduction in a single set of data**



Mapping algorithm onto partition able machine

- Are parallel processing systems that can be divided into independent sub systems, each having the same characteristics as the original machine
- Ability to form multiple independent sub systems to execute multiple tasks in parallel
- Eg MIMD can be partitioned under a system control

Matrix and Vector Operations

- Matrix transposition
- Matrix multiplication

Achieving scalability using set of algorithms

- A parallel algorithm is scalable if it is capable of delivering an increase of the number of processors utilized
- i.e $performance_{PA} \propto processors_{utilized}$

End of Session
Unit 3.3