SOFTWARE ENGINEERING - UNIT III -18BIT41C

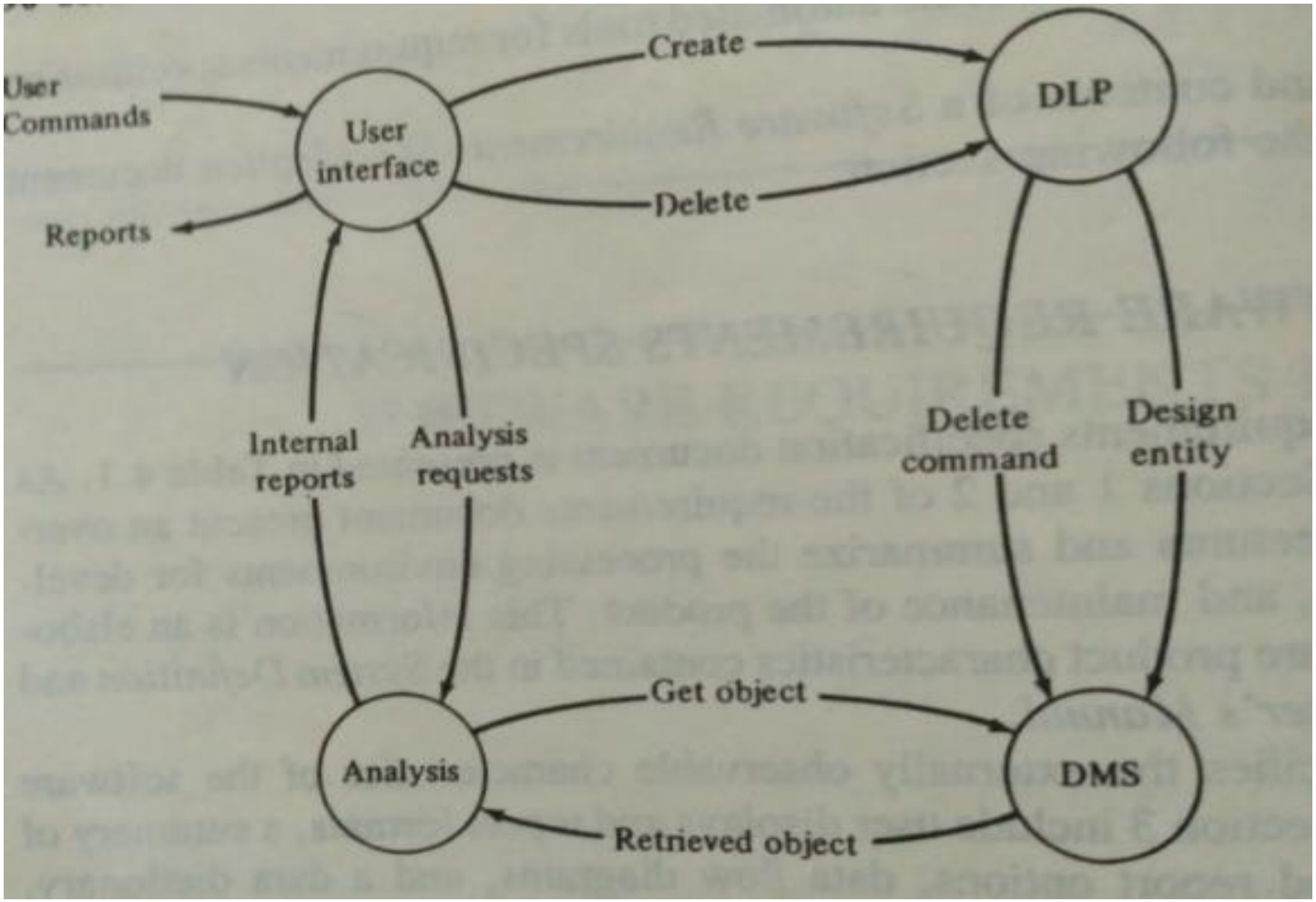
UNIT III: Software Requirements Definitions: The Software Requirements Specification – Formal Specification Techniques – Languages and Processors for Requirements Specification.

THE SOFTWARE REQUIREMENT SPECIFICATION

Format of a software requirement specification

- Product overview and summary
- Development, operating and maintenance environments
- External interfaces and dataflow
- Functional requirements
- Performance requirements
- Exception handling
- Early subsets and implementation priorities
- Forseable modifications and enhancements
- Acceptance criteria
- Design hints and guidelines
- Cross reference index
- Glossary of terms

AN INFORMAL DFD



Sec 1 and 2 elaboration of the software product characteristics contained in the system definition and the preliminary user manual.

Sec 3 includes user displays, report formats, user commands, DFD , data dictionary.

DFD specify data sources, data sinks, data stores, transformations to be performed on the data and the flow of data between sources, sinks, transformations and stores.

A formal DFD

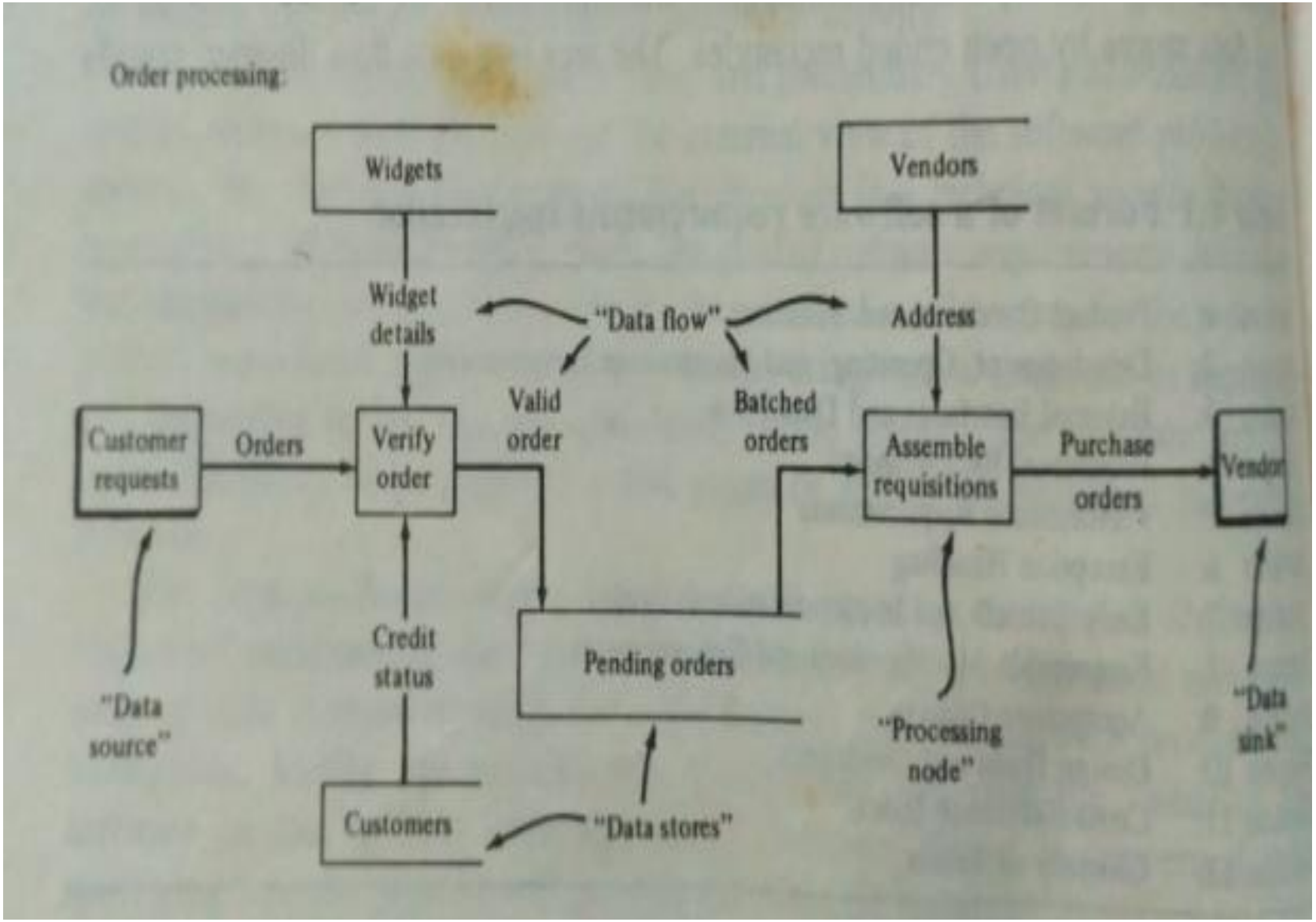
Sources and data sinks ⟶ shaded rectangles

Transformations ⟶ ordinary rectangles

Data stores ⟶ open ended rectangles

Arcs ⟶ data flow

ORDER PROCESSING



- The entries in a data dictionary include the name of the data item and attributes such as the DFD, where it is used its purpose, where it is derived from its sub items and any notes that may be appropriate.

Ex: data dictionary entry

Name	: create
Where used	:SDLP
Purpose	: create passes- user created and design
Derived from	: user interface processor
Subitems	:name
	uses
	procedures
	references
Notes	: create contains one complete user-created design entity.

Sec 4: software requirement specification specifies the functional requirements it is expressed in relational and state oriented notations.

Sec 5: response time activities, processing time for various processes, throughput, pricing, secondary memory constraints.

Sec 6: Exception handling : A table of exception conditions and exception responses should be prepared. Ex: table overflow, array indices out of range, runtime stack overflow etc.,

Sec7: Software developed as a series of successive versions. The initial version may be skeletal prototype each successive user functions and provides framework for evolution of the product.

Sec 8: foreseeable modifications and enhancement that may be incorporated in the product following initial product release.

Sec 9: software product acceptance criteria are specified i.e functional and performance test that must be performed, and the standards to be applied to the source code.

Sec 10: contain design hints and guidelines

Sec 11: relates product requirements to the sources of information used in

Deriving the requirements verification and reexamination of requirements, constraints and assumptions.

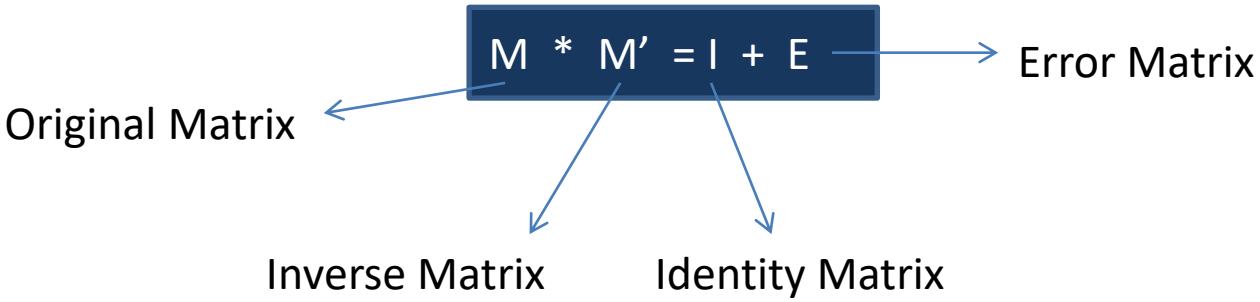
Sec 12: provides definition of terms that may be unfamiliar to the customer and the product developers.

FORMAL SPECIFICATION TECHNIQUES

1. Relational Notations:

a. Implicit equations: State the properties of a solution without stating a solution method.

ex: Matrix inversion as



E- specifies allowable computable errors

Ex: Square root function, SQRT

(0<=X<=Y)[ABS(SQRT(X)**2-X)<E]

for all real values of X in the closed range 0 to Y, computing the square root of X, squaring it, subtract X, result in error value.

b. Recurrence Relations: The recurrence relation consists of an initial part called the basis and one or more recursive parts.

Fibonacci numbers can be defined as:

FI(0)=0

FI(1)=1

FI(N)=FI(N-1)+FI(N-2) for all N>1

c. Algebraic axioms: Mathematical systems are defined by axioms. The axioms specify the fundamental properties of a system and provide a basis for deriving additional properties that are implied by the axioms. These additional properties are called theorems.

Ex: LIFO property of stack object operations

NEW	Creates a new stack
PUSH	Adds a new item to the top of a stack
TOP	Returns a copy of the top item
POP	Removes a top item
EMPTY	Tests for an empty stack

Operation NEW yields a newly created stack PUSH requires two arguments.

Syntax:

OPERATION DOMAIN RANGE

NEW () → STACK

PUSH (STACK, ITEM) → STACK

POP (STACK) → STACK

TOP (STACK) → ITEM

EMPTY (STACK) → BOOLEAN

AXIOMS

(1)EMPTY(NEW)= true

(2)EMPTY(PUSH(stack, item))=false

(3)POP(NEW)=error

(4)TOP(NEW)=error

(5)POP(PUSH(stack, item)=stack

(6)TOP(PUSH(stack, item))=item

The axioms can be stated in English:

1. The new stack is empty
2. A stack is not empty immediately pushing an item onto it.
3. Attempting to pop a new stack results in error
4. There is no top item on a new stack
5. Pushing an item onto a stack and immediately popping it off leaves the stack unchanged.
6. Pushing an item onto a stack and immediately requesting the top item returns the item just pushed onto the stack.

d. Regular Expressions is used to specify the syntactic structure of symbol strings. So, regular expressions can thus be viewed as language generators.

Rules for forming Regular Expressions

1. Atoms: The basic symbols in the alphabet of interest form regular expressions.
2. Alternation: if R1 and R2 are regular expression then (R1|R2) is a regular expression.
3. Composition: if R1 and R2 are regular expressions then (R1.R2) is a regular expression.
4. Closure : if R1 is a regular expression, then (R1)* is a regular expression.
5. Completeness: nothing else is a regular expression.

Example

1. Given atoms a and b then a denotes the set {a} and b denotes the set {b}.
2. Given atoms a and b, then (a/b) denotes the set {a,b}
3. Given atoms a, b and c then ((a/b)/c) denotes the set {{a,b},c}
4. Given atoms a and b then (ab) denotes the set {ab} containing one element ab.

5. Given atoms a, b and c then ((ab)c) denotes the set {abc} containing one element abc.

6. Given atom a, then (a)* denotes the set {e,a,aa,aaa….} e is the empty set.

Ex:

(a(b/c)) ⟹ {ab,ac}

(a/b)* ⟹ {e,a,b,aa,bb,ab,ba,aab…..}

((a(b/c)))* ⟹ {e,ab,ac,abab,acac,abac,acab……}

* ⟶ Kleene star

+ ⟶ Kleene plus notations

2. State-oriented notations

a. Design Tables: provides a mechanism for recording complex decision logic. It is segmented into 4 quadrants.

- Condition stub
- Condition entry
- Action stub
- Action entries.

Basic elements of a decision table

	Decision Rules			
	Rule1	Rule2	Rule3	Rule4
(Condition stub)	(Condition entries)			
(Action stub)	(Action entries)			

Limited-entry decision table

	1	2	3	4
Credit limit satisfactory	Y	N	N	N
Pay experience is favourable	-	Y	N	N
Special clearance is obtained	-	-	Y	N
Perform approve order	X	X	X	
Go to reject order				X

Y-Yes, N-No, - = don't care , X - perform action.

An Ambiguous Decision Table

	Decision rule			
	Rule1	Rule2	Rule3	Rule3
C1	Y	Y	Y	Y
C2	Y	N	N	N
C3	N	N	N	N
A1	X			
A2		X		
A3			X	X

An incomplete Decision table

	Decision rule		
	Rule1	Rule2	Rule3
C1	Y		Y
C2		Y	N
C3			Y
A1		X	
A2	X		
A3			X

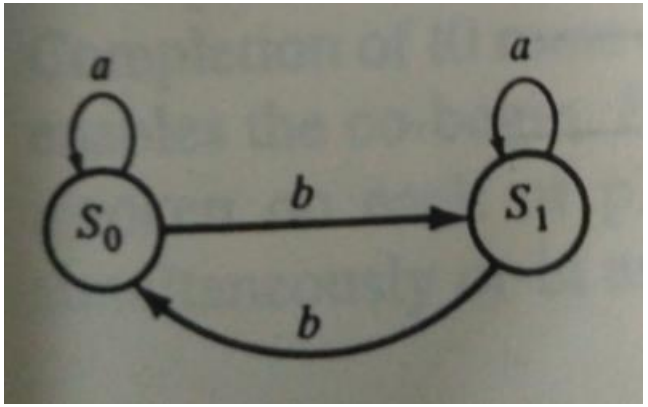
R3 and R4 are redundant rules.
R2 and R3, and R2 and R4 are contradictory rules

A decision table is complete if every possible set of conditions has a corresponding action prescribed failure to specify an action for any one of the combinations results in an incomplete decision tables.

b. Transition tables

Transition tables are used to specify changes in the state of a system as a function of driving forces.

Transition Diagram



Current state	Current input	
	a	b
S0	S0	S1
S1	S1	S0

Transition Table

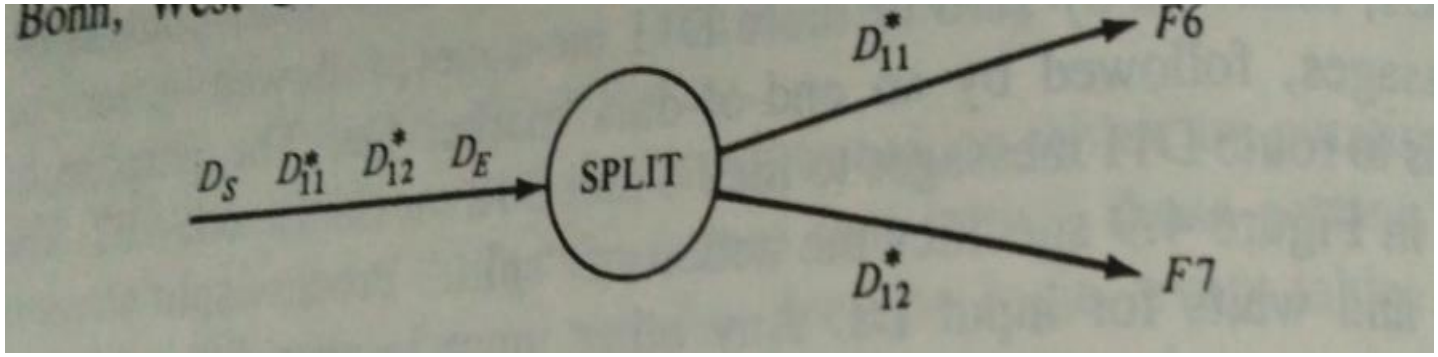
An augmented Transition Table

Present state	Input	Action	Output	Next state
S0	a b			S0 S1
S1	a b			S0 S1

- ✓ Transition table that is augmented to indicate actions to be performed and outputs to be generated in the transition to the next state.
- ✓ Transition diagrams are alternative representatives for transition tables.
- ✓ It is represented as arcs between nodes.
- ✓ Arcs are labelled with conditions that cause transitions.
- ✓ The transition diagrams and transition tables are representations for finite state automata.

c. Finite state mechanisms

Data flow diagrams, regular expressions, transition tables can be combined to provide a powerful finite mechanism for functional specification of software systems.



D_s ⟶ start marker
D_{11} ⟶ *zero or more* D_{11} *messages*
D_{12} ⟶ zero or more D_{12} messages
D_E ⟶ end-of-data marker

The system for which the incoming data stream consists of a start marker D_S , followed by zero or more D_{11} messages, followed by zero or more D_{12} messages followed by an end-of-data marker D_E . The purpose is to split D_{11} messages to file F_6 and D_{12} messages to file F_7 .

Specification of the “Split” process.

Present state	Input	Actions	Outputs	Next state
$S0$	D_S	Open $F6$ Open $F7$		$S1$
$S1$	D_{11}	Write $F6$	D_{11} :$F6$	$S1$
	D_{12}	Close $F6$ Write $F7$	D_{12} :$F7$	$S2$
	D_E	Close $F6$ Close $F7$		$S0$
$S2$	D_{12}	Write $F7$	D_{12} :$F7$	$S2$
	D_E	Close $F7$		$S0$

d. Petri Nets

- Petri nets invented by carl petri at the university of Bonn, west germany.
- Petri nets provide a graphical representation technique.
- Petri nets overcome the limitations of finite state mechanisms.
- Petri nets is represented as a bipartite directed graph. Two types of nodes are called places(tokens) and transitions.
- Petri nets are characterized by an initial marking of places and a firing rule.
- An enabled transition can fire , when a transition fires, each input place of that transition loses one token and each output place of that transition gains one token.
- Petri net defined as a quadruple, consisting of

  a set of places P

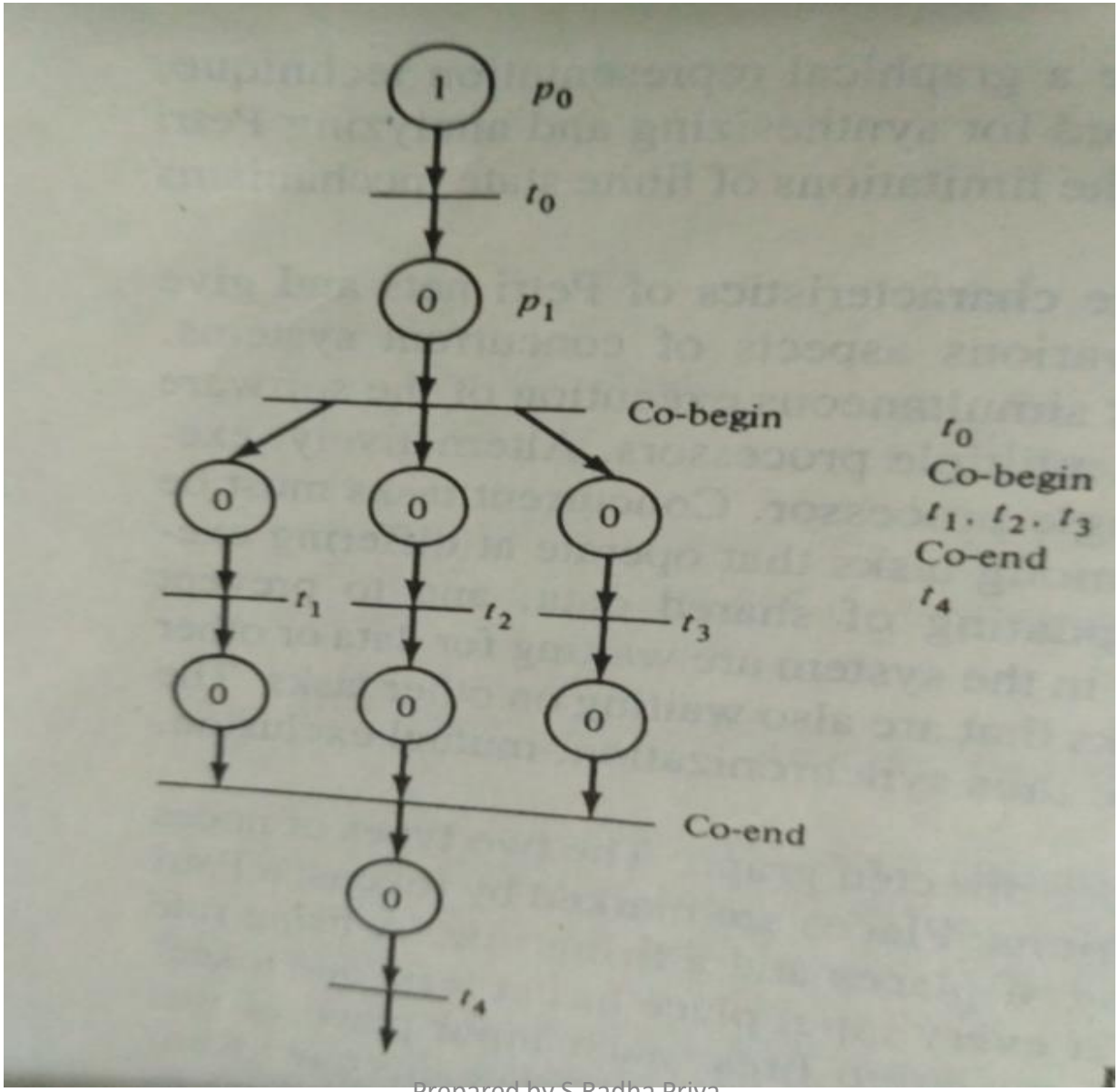
  a set of Transitions T

  a set of arcs A

  a marking M

  C=(P, T, A, M)

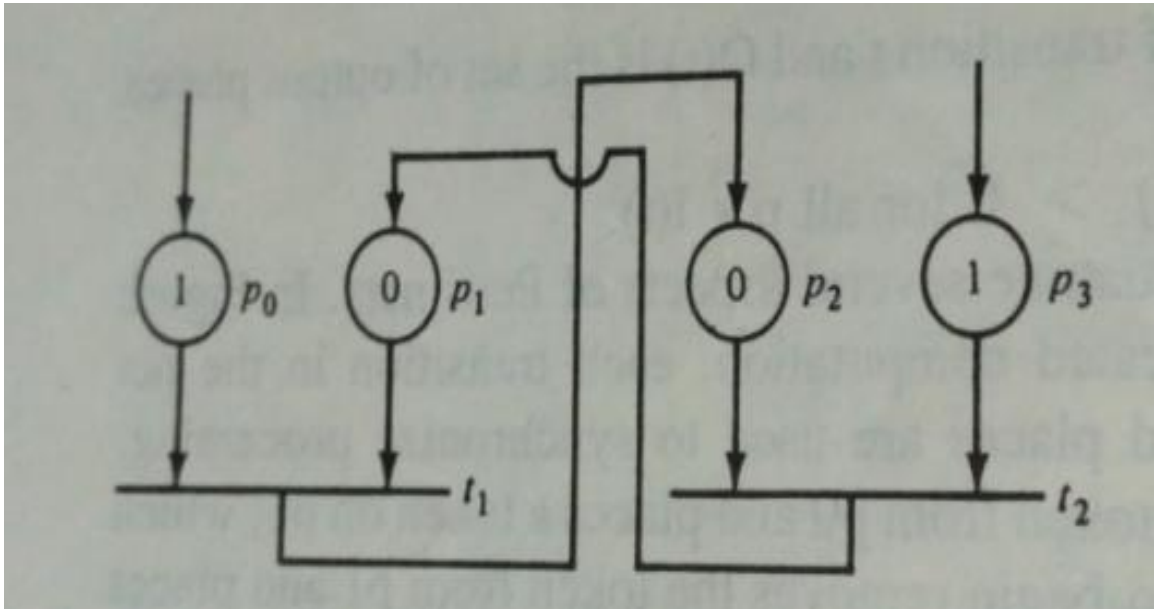
Petrinet model of concurrent processes t_1, t_2, t_3



Completion of p_1 which enables co-begin. Firing of the co-begin removes the token from p_1 and places in $p_{2,}$ p_3 , p_4. This enables t_1 , t_2 ,t_3. They can fire simultaneously in any order. When each tasks t_1 , t_2 ,t_3 computes, it is placed in p_5 , p_6 ,p_7.

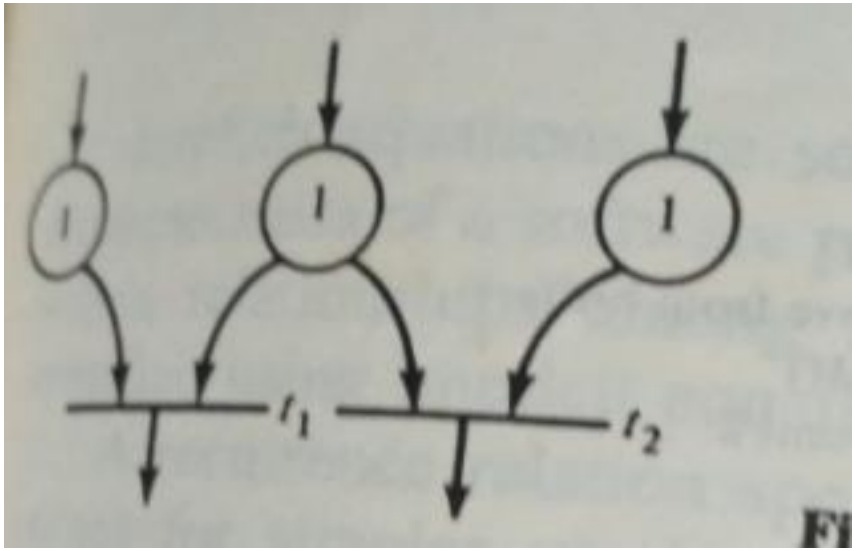
A deadlock petri net

Both t_1 and t_2 are waiting for the other to fire and neither can proceed.



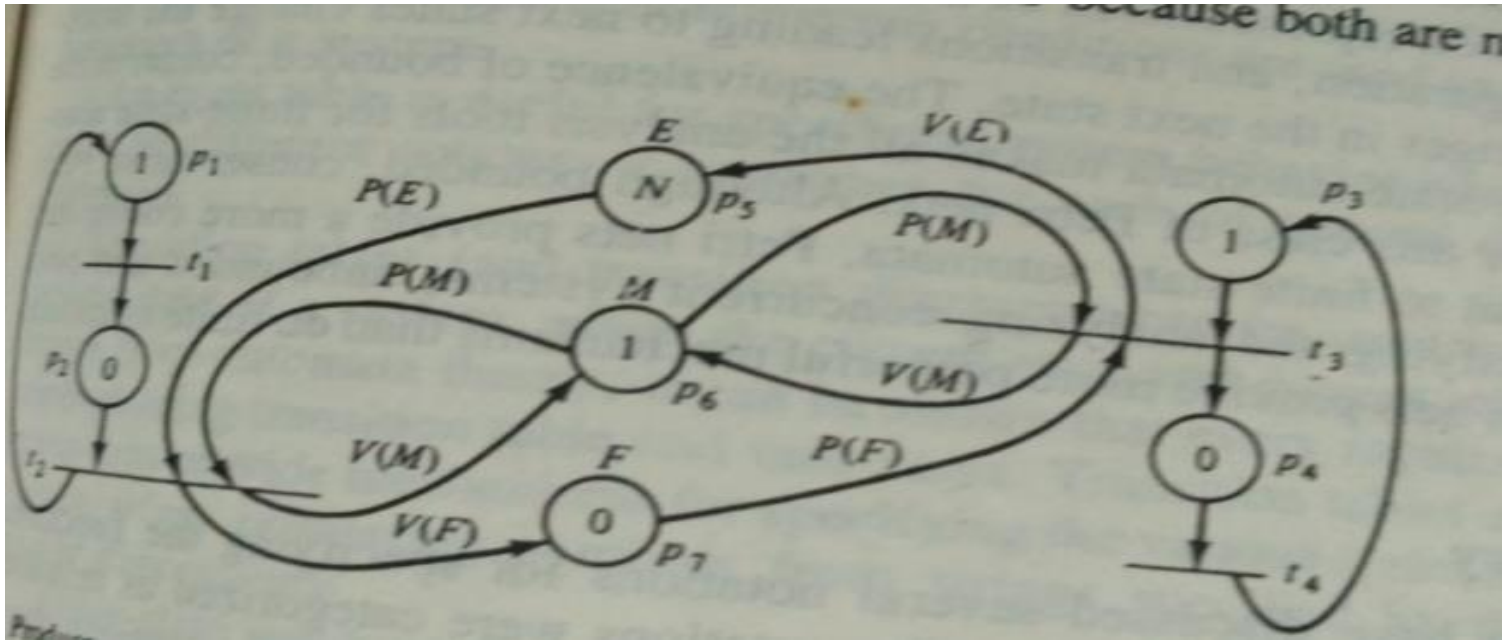
Mutual exclusion conflict situation

Both t_1 and t_2 are enabled , but only one can fire. Firing one will disable the other .



Initial Marking for the producer/consumer Petri Nets

t_1 produce
t_2 Place in buffer
t_3 Remove from buffer
t_4 Consume



- t_1 is the producing process, t_2 places a produced item in the buffer
- t_3 removes an item from the buffer, t_4 consumes it
- E current number of empty buffer positions
- F indicates the current number of filled buffer positions

- M prevents simultaneous insertion and removal of items from buffer
- Initially M, P_1, P_3 has one token P_2, P_7, p_4, F has zero token E has N tokens.
- E prevents buffer overflow
- F prevents buffer underflow
- M prevents simultaneous reading and writing of the buffer
- An invariant is a set of places whose total number of tokens remains constant and which has no invariant subsets.
- ex:$\{P_{1,}P_2\}\,\{P_3, p_4\}\{p_5, p_7\}\{p_6\}$

The petri net exhibits mutual exclusion between t_2 and t_3 because both are marked by P_6 and $\{P_6\}$ is an invariant, having one token. Thus t_2 and t_3 are mutually exclusive.