UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4351 Fall 2023

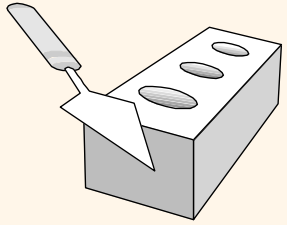
Software Engineering

M & W 4 to 5:30 PM

Prof. Victoria Hilford

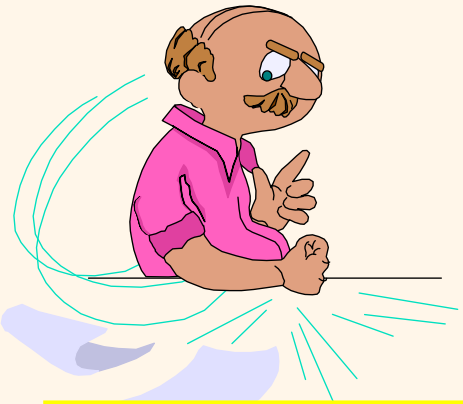
PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4351

4 to 5:30



PLEASE

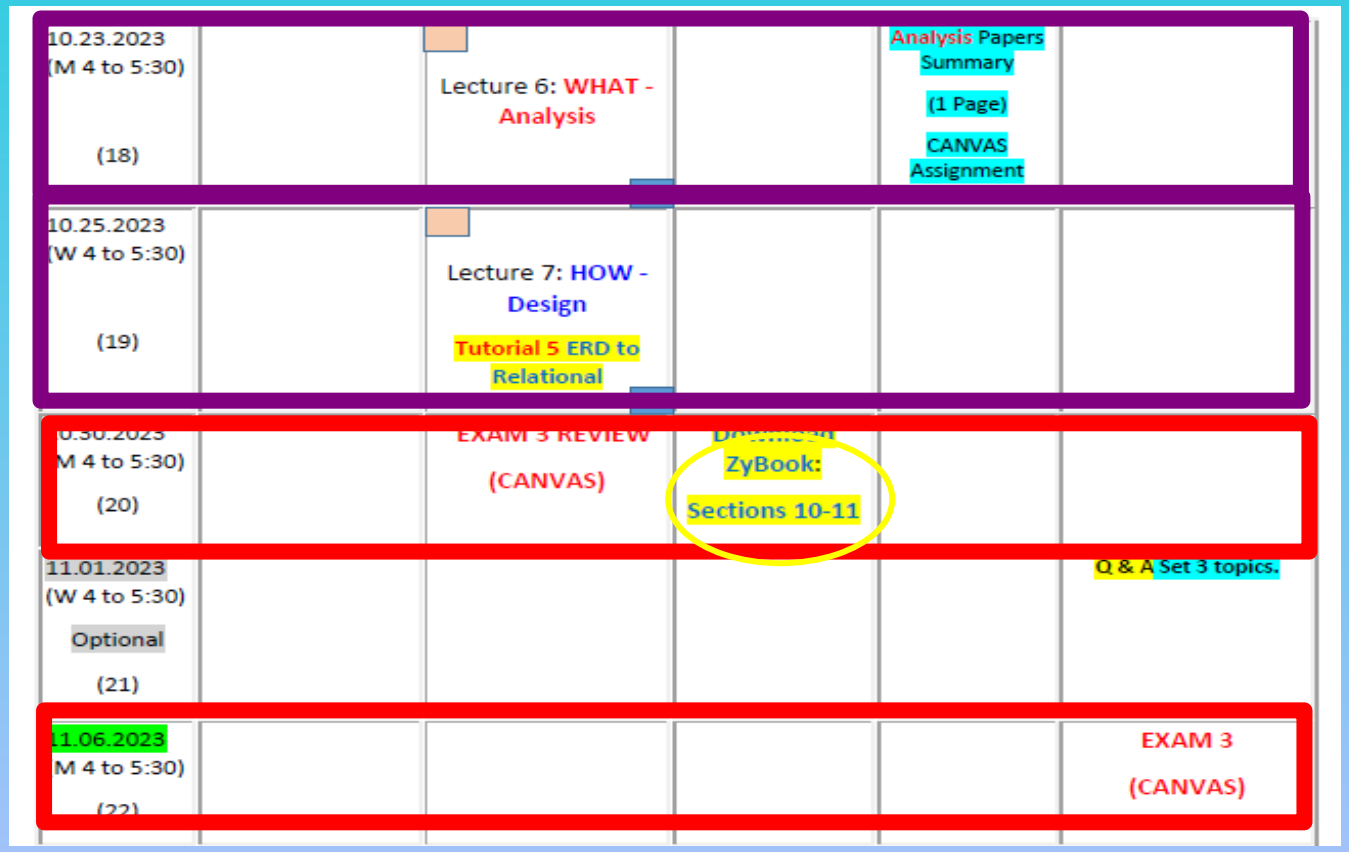
LOG IN

CANVAS

Youyi [A-L]

Kevin [M-Z]

Please close all other windows.

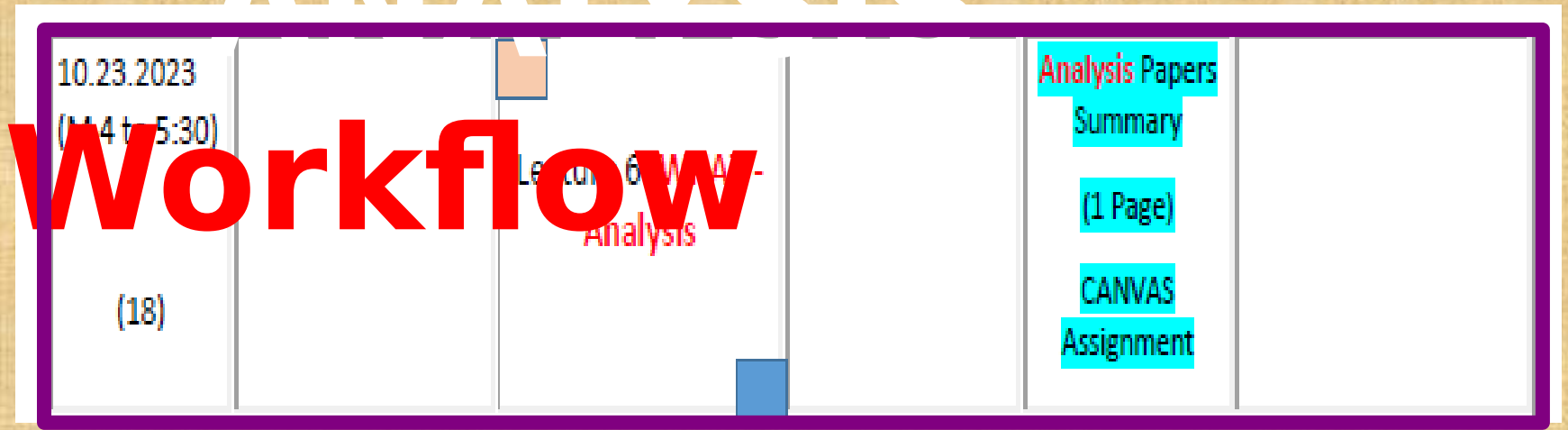

10.23.2023
(M 4 to 5:30)
(18)
Lecture 6: WHAT - Analysis
Analysis Papers Summary
(1 Page)
CANVAS Assignment
10.25.2023
(W 4 to 5:30)
(19)
Lecture 7: HOW - Design
Tutorial 5 ERD to Relational
M 4 to 5:30)
(20)
EXAM 3 REVIEW
(CANVAS)
ZyBook:
Sections 10-11
11.01.2023
(W 4 to 5:30)
Optional
(21)
Q & A Set 3 topics.
11.06.2023
M 4 to 5:30)
(22)
EXAM 3
(CANVAS)

Class 18

Lecture 6 on

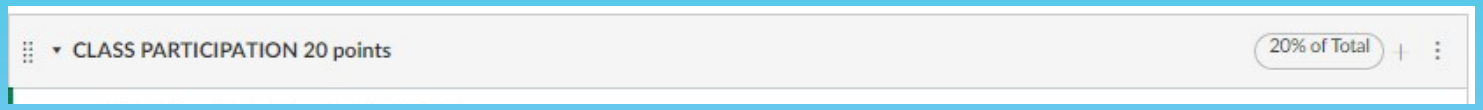
ANALYSIS

Workflow



From 4:00 to 4:10 – 10 minutes.

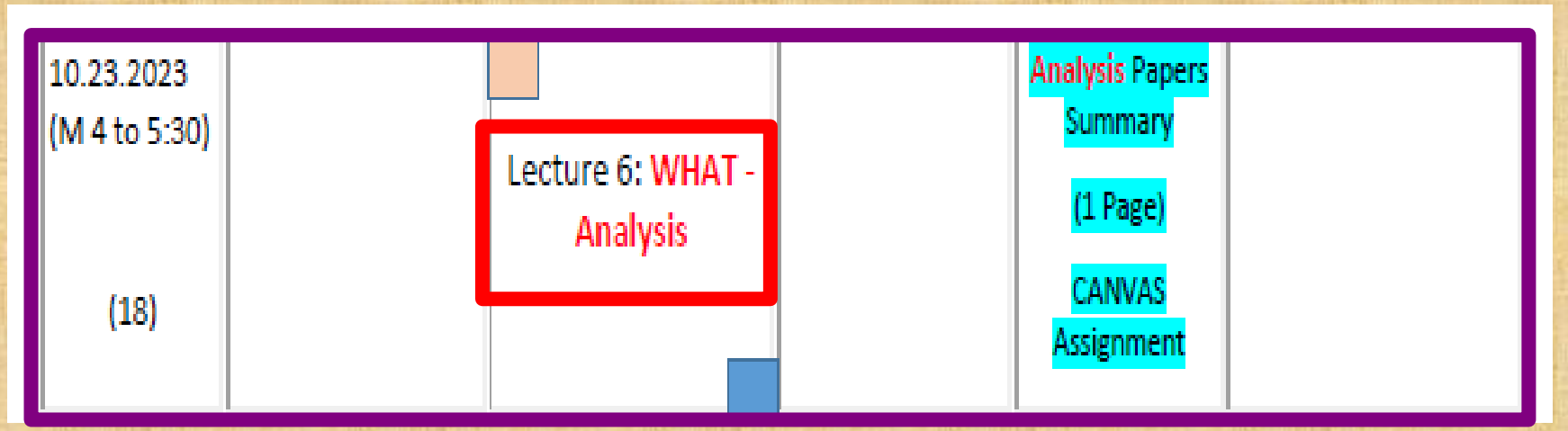
10.23.2023 (M 4 to 5:30) (18)		Lecture 6: WHAT - Analysis		Analysis Papers Summary (1 Page) CANVAS Assignment	



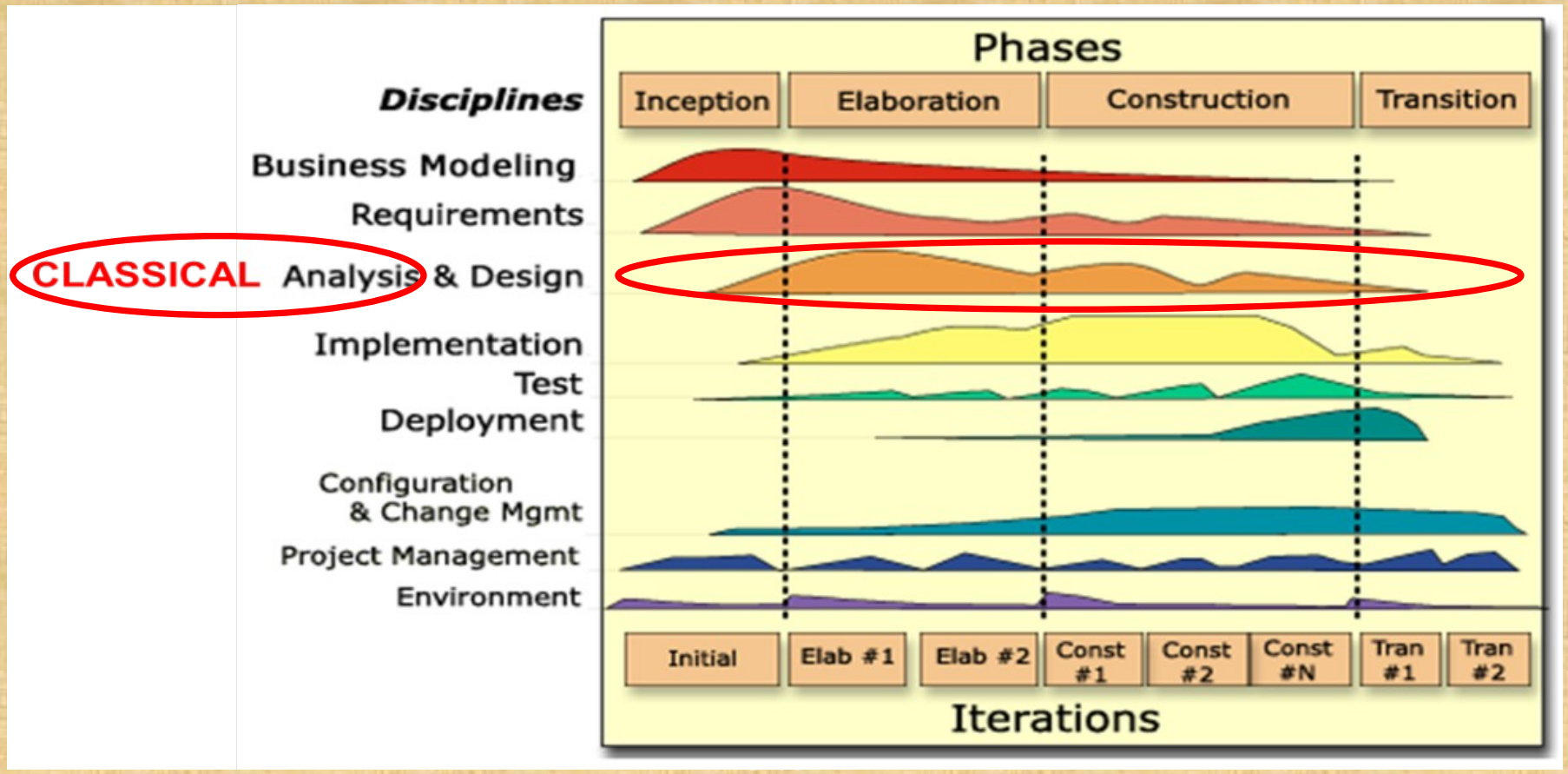
PASSWORD: I AM IN TEAMS

VH– Publish.

From 4:10 to 5:00 – 50 minutes.

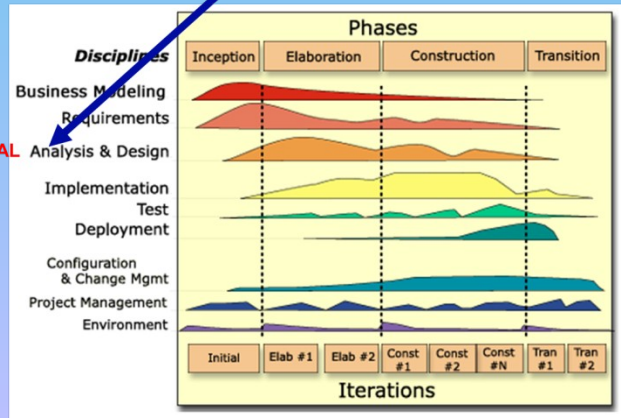


Analysis Workflow

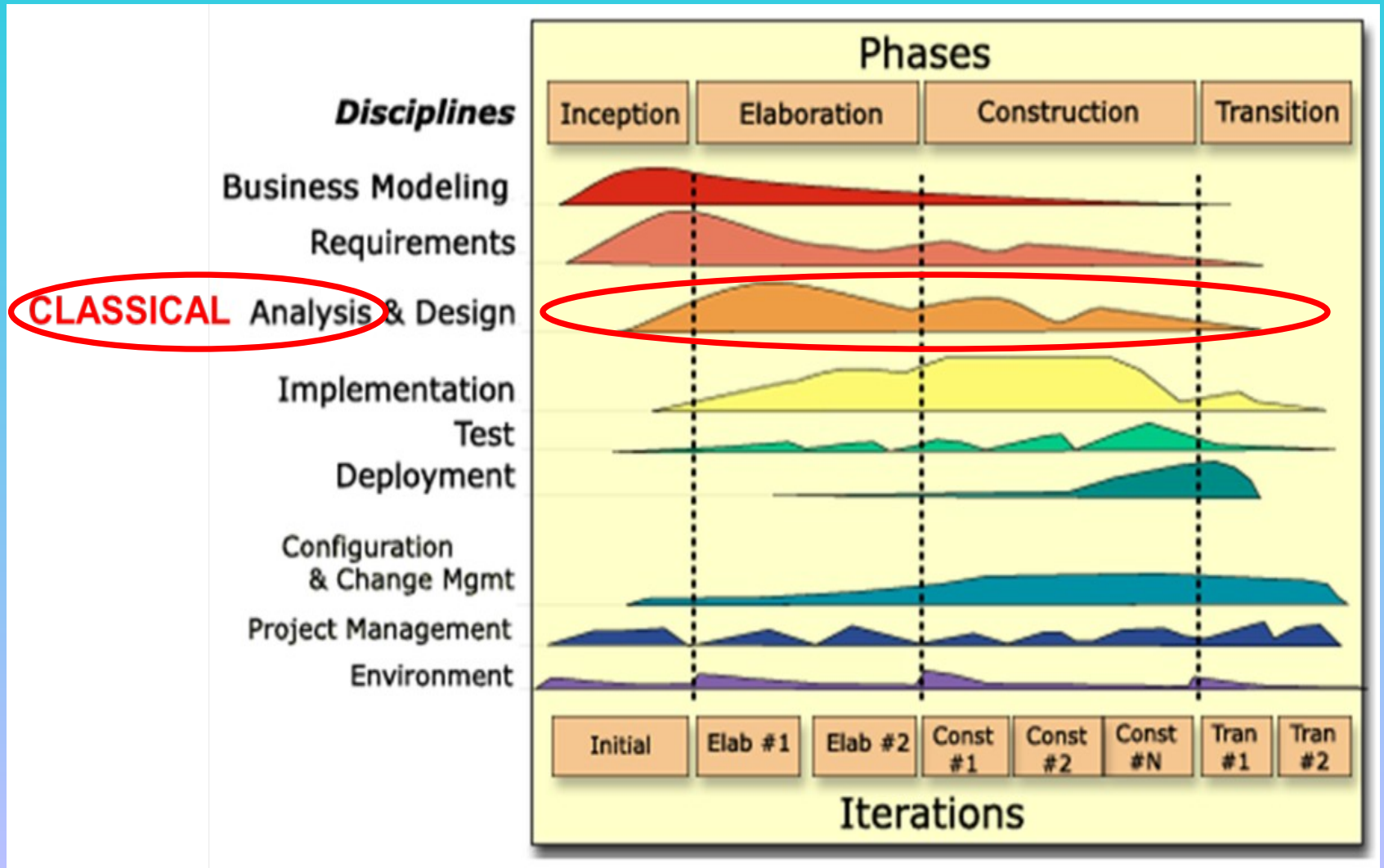


Writing the Software Specification (Analysis)

CLASSICAL ANALYSIS



CLASSICAL ANALYSIS - WHAT



Comparison of Classical Analysis (Specification)Techniques

Classical Analysis Method	Category	Strengths	Weaknesses

The **Specification** (**Analysis**) Document Must Be

Informal enough for the **Client**

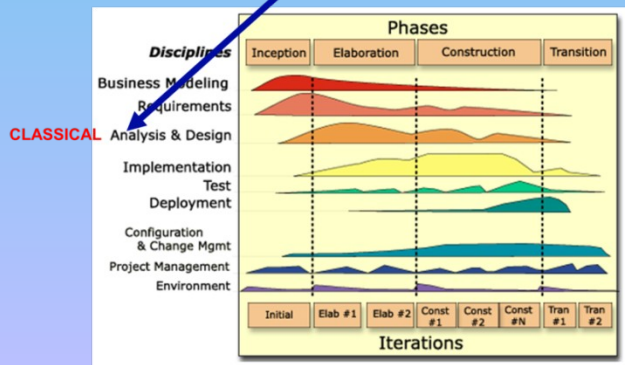
- The **Client** is generally **not a computer specialist**

Formal enough for the **Developers**

- It is the **sole source of information** for drawing up the **Analysis**

These **two** are ***mutually contradictory***

CLASSICAL **ANALYSIS**



Comparison of Classical Analysis (Specification)Techniques

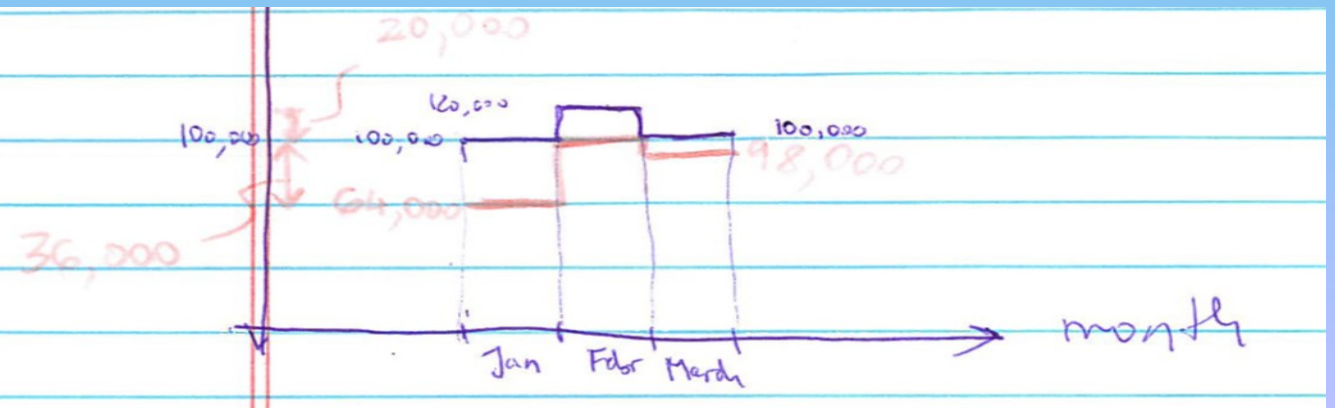
Informal Specifications (Analysis)

Informal **Specifications** are written in a **natural language**

- Examples: English, Mandarin, Kiswahili, Hindi

Example

"If the sales for the current month are below the target sales, then a report is to be printed, **unless** the difference between target sales and actual sales is less than half of the difference between target sales and actual sales in the previous month, **or** if the difference between target sales and actual sales for the current month is under 5%"



Ambiguous?

Comparison of Classical Analysis Techniques

Classical Analysis Method	Category	Strengths	Weaknesses
Entity-relationship modeling (Section 11.6) PSL/PSA (Section 11.5) SADT (Section 11.5) SREM (Section 11.5) Structured systems analysis (Section 11.3)	Semiformal	Can be understood by the client More precise than informal techniques	Not as precise as formal techniques Generally cannot handle timing

Structured Systems Analysis

Three popular graphical **Specification Methods**

- DeMarco
- **Gane and Sarsen**
- Yourdon

All are equivalent

All are equally good

Classical Analysis Method	Category	Strengths	Weaknesses
Entity-relationship modeling (Section 11.6) PSL/PSA (Section 11.5) SADT (Section 11.5) SREM (Section 11.5) Structured systems analysis (Section 11.3)	Semiformal	Can be understood by the client More precise than informal techniques	Not as precise as formal techniques Generally cannot handle timing

Structured Systems Analysis

- Many U.S. corporations use them for **commercial products**
- Gane and Sarsen's **Method** is taught here because it is so widely used

Classical Analysis Method	Category	Strengths	Weaknesses
Entity-relationship modeling (Section 11.6) PSL/PSA (Section 11.5) SADT (Section 11.5) SREM (Section 11.5) Structured systems analysis (Section 11.3)	Semiformal	Can be understood by the client More precise than informal techniques	Not as precise as formal techniques Generally cannot handle timing

Sally's Software Shop Mini Case Study

- Sally's Software Shop buys software from various suppliers and sells it to the public. **Popular software** packages are kept in stock, but the rest must be ordered as required. Institutions and corporations are given credit facilities, as are some members of the public. **Sally's Software Shop** is doing well, with a monthly turnover of **300 packages** at an average retail cost of **$250** each.

- Despite her business success, Sally has been advised to computerize.

Should she?

Sally's Software Shop Mini Case Study

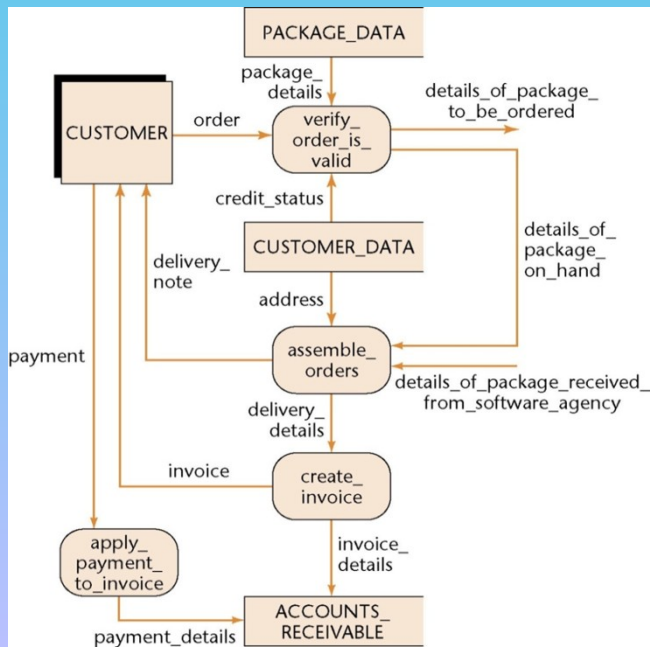
- We assume: Sally wishes to computerize "in order to make more **money**"
  - We need to perform **Cost Benefit Analysis** for each section of her business

Sally's Software Shop Mini Case Study

- The danger of many standard approaches
  - First **produce the solution, then find out what the problem is**!

- **Gane and Sarsen's Method for Structured Systems Analysis**
- Nine-step **Method**
  - **Stepwise refinement** is used in many steps
    1. Draw the **DFD**
    2. Decide what sections to computerize
    3. Determine the details of the Data Flow
    4. Define the logic of the **Processes**
    5. Define the **Data Stores**
    6. Define the physical resources
    7. Determine the Input-Output **Specifications**
    8. Perform the sizing
    9. Determine the **Hardware** requirements

DFD



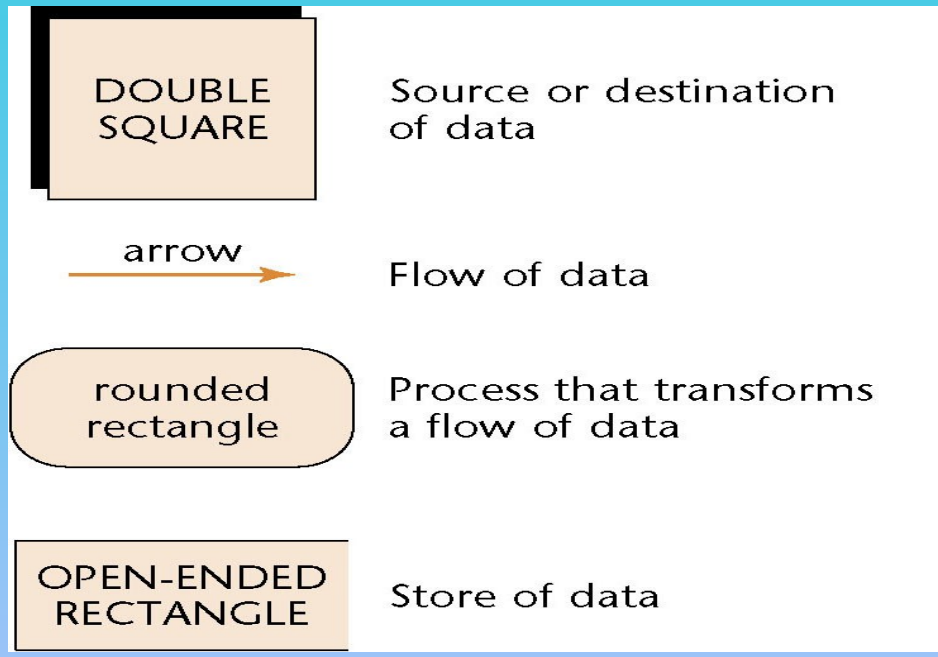
Sally's Software Shop Mini Case Study

The Data Flow Diagram (DFD) shows the logical Data Flow

(Pictorial representation of all aspects of the logical Data Flow)

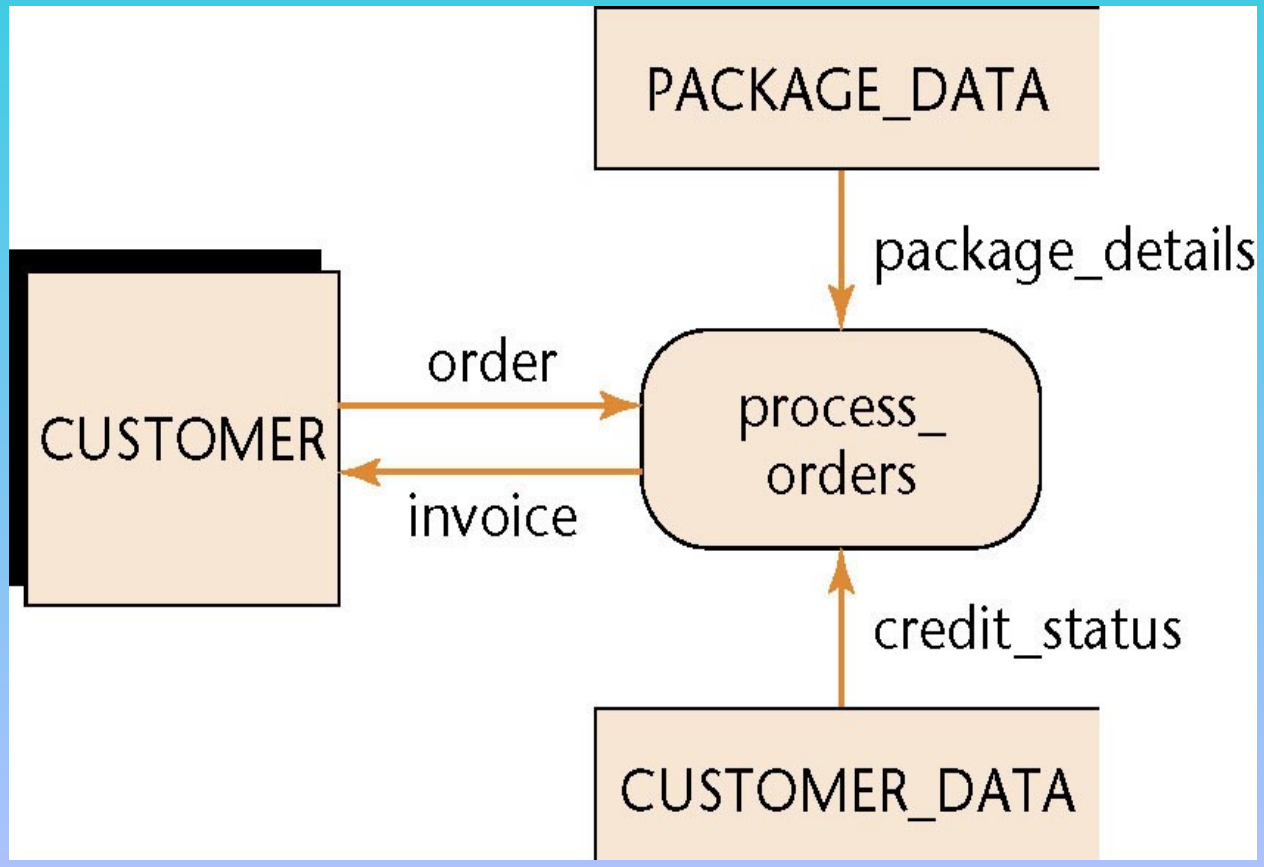
- "**WHAT** happens, **not HOW** it happens"

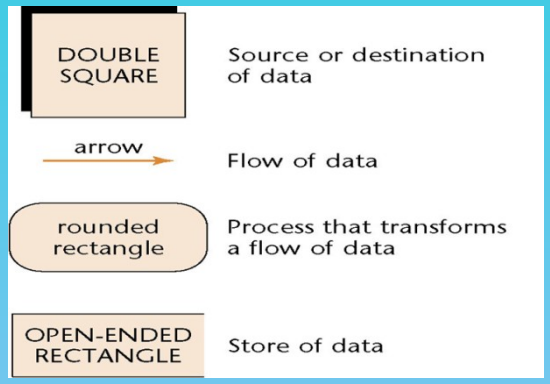
Symbols:



Step 1. Draw the DFD

- **First refinement**
  - Infinite number of possible interpretations





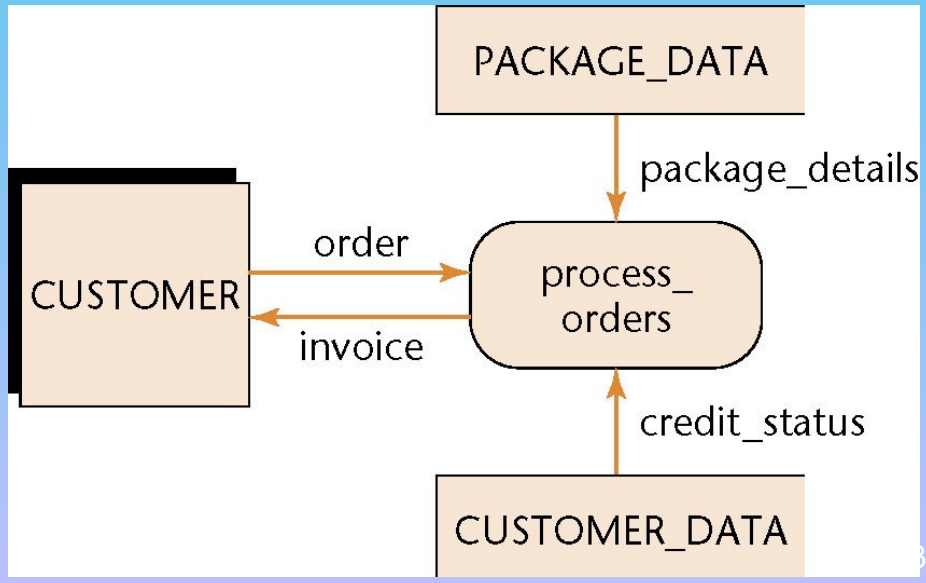
Gane and Sarsen's method for Structured

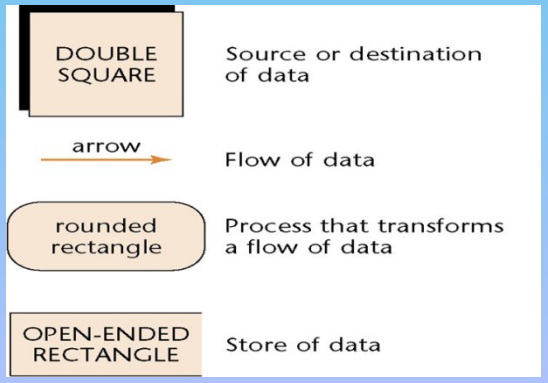
Nine-step method

- **Stepwise refinement** is used in many steps
  1. Draw the **DFD**
  2. Decide what sections to **COMPUTERIZE**
  3. Determine the details of the Data Flow
  4. Define the logic of the Processes
  5. Define the **Data Stores**
  6. Define the physical resources
  7. Determine the Input-Output **Specifications**
  8. Perform the sizing
  9. Determine the Hardware requirements

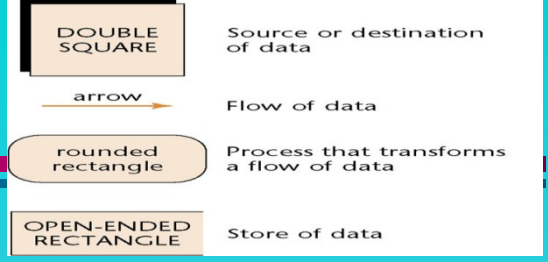
Step 1. Draw the **DFD**

- **PACKAGE_DATA** consists of 900 shrink-wrapped boxes containing diskettes or CDs displayed on shelves, as well as a number of catalogs in a desk drawer.
- Data store **CUSTOMER_DATA** is a collection of 5X7" cards held together by a rubber band, plus a list of customers whose payments are overdue. **Process** (action) **process_orders** is Sally looking for the appropriate package on the shelves, if necessary looking it up in a catalog, and then finding the correct 5X7 card and checking that the customer's name is not on the list of defaulters.
- **This implementation is totally manual and it corresponds to the way Sally currently conducts her business.**



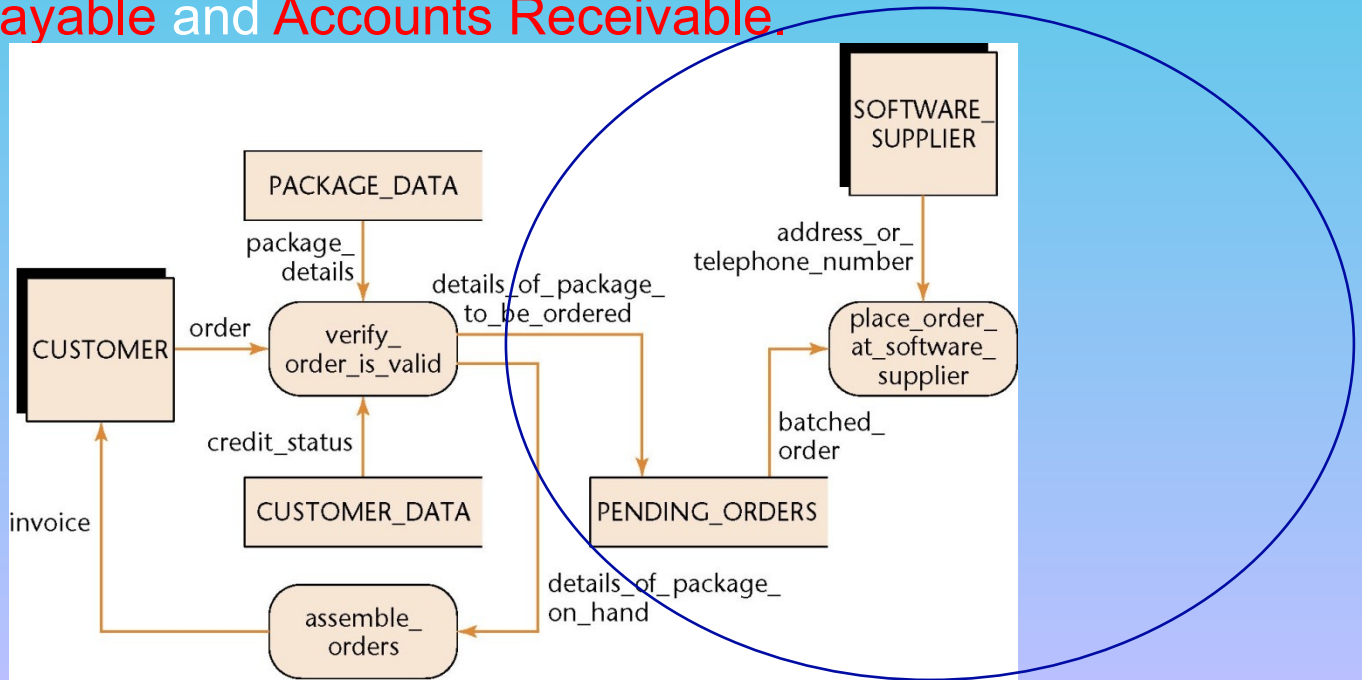


Step 1. Draw the DFD



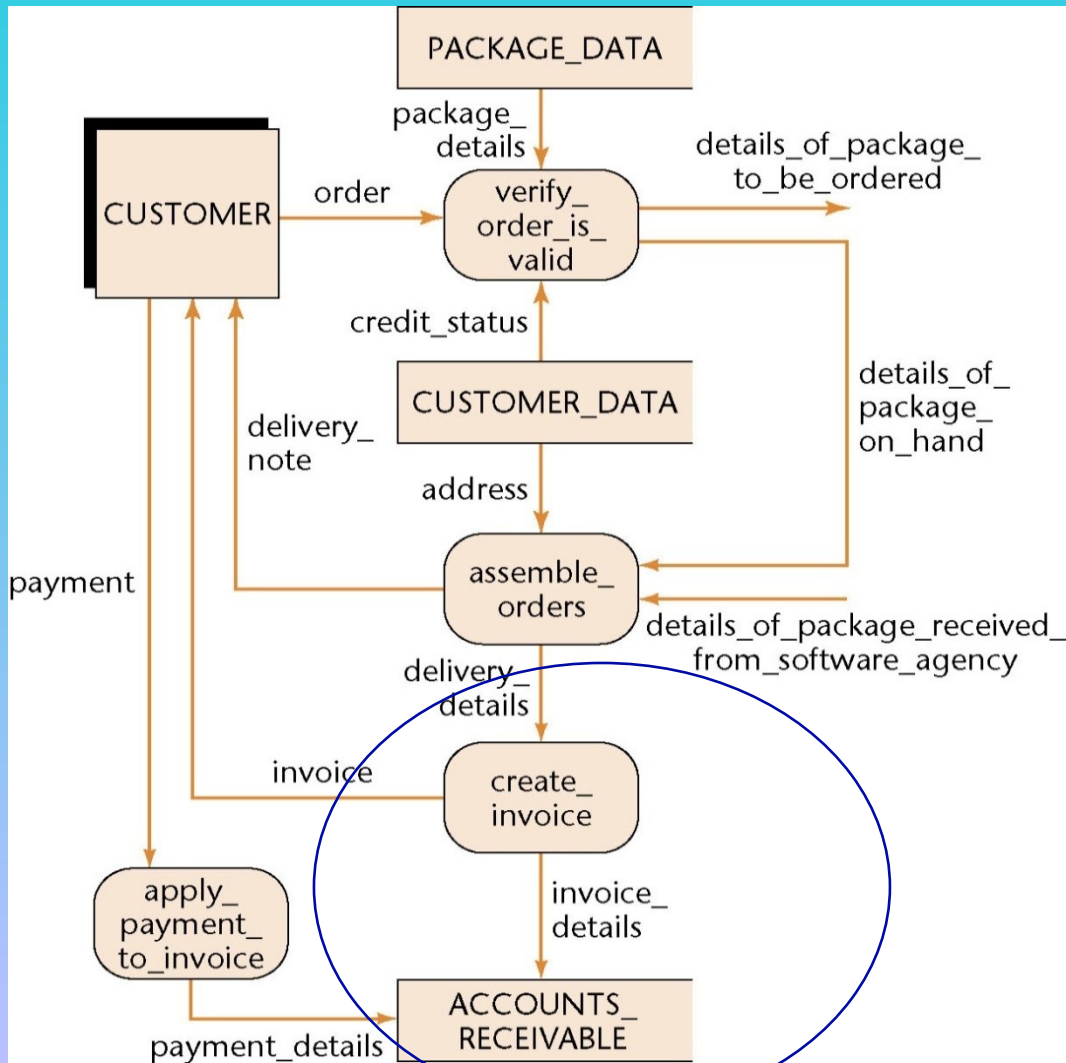
Second refinement

- The logical flow of Data representing **WHAT happens** when the customer requests a package Sally does not have on hand is added to the initial DFD. Specifically, details of that package are placed in the data source **PENDING_ORDERS**, which might be a computer file, but at this stage equally well could be a manila folder.
- This DFD does not show the logical flow of data when the software package **arrives** from the supplier nor does it show financial functions such as Accounts Payable and Accounts Receivable.



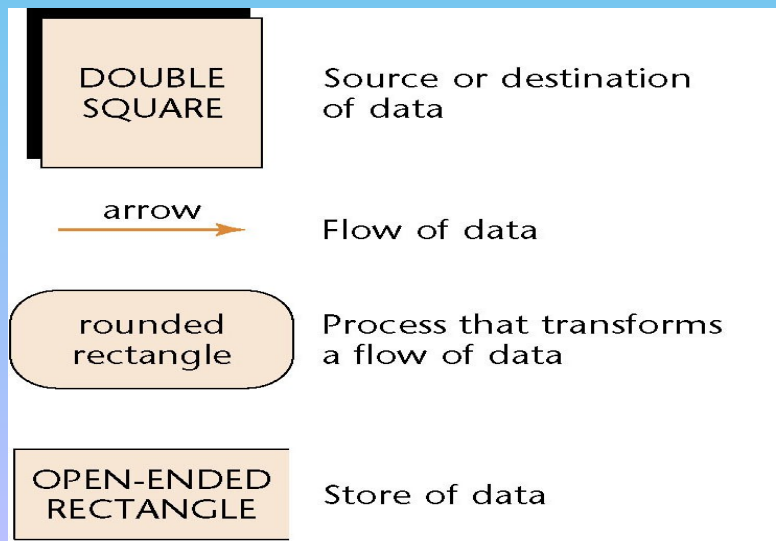
Step 1. Draw the **DFD**

Portion of third refinement: (the logical flow of Data relating to **ACCOUNTS_RECEIVABLE** is added to the DFD.)



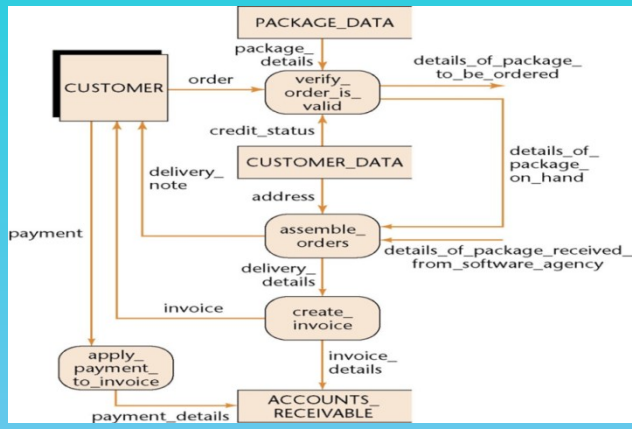
Step 1. Draw the DFD

- The **final** DFD is larger
  - But it is easily understood by the **Client**; Sally will sign it off (Contract)
- When dealing with larger DFDs
  - Set up a **hierarchy** of DFDs
  - A **box becomes a DFD** at a lower level



Step 2. Decide What Parts to Computerize and HOW

- It depends on how much $ Client is prepared to spend



- Large volumes, tight controls
  - Batch

- Small volumes, in-house microcomputer
  - Online

- **Cost Benefit Analysis**

Gane and Sarsen's method for Structured

Nine-step method

– **Stepwise refinement** is used in many steps

1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the **Data Flow**
4. Define the logic of the **Processes**
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the **Input-Output Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

Step 3. Determine the Details of the Data Flows

(arrows)

- **Determine the Data items for each Data Flow**

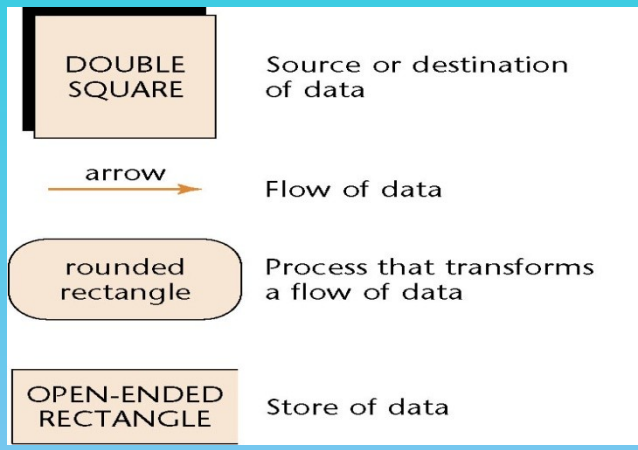
- **Refine each Flow stepwise**
  - Example:

    order:

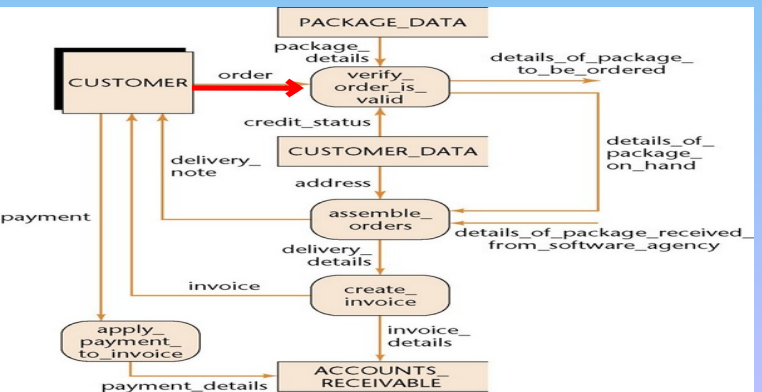
    order_identification

    customer_details

    package_details



- We need a **Data Dictionary** for larger products



Gane and Sarsen's method for Structured

Nine-step method

– **Stepwise refinement** is used in many steps

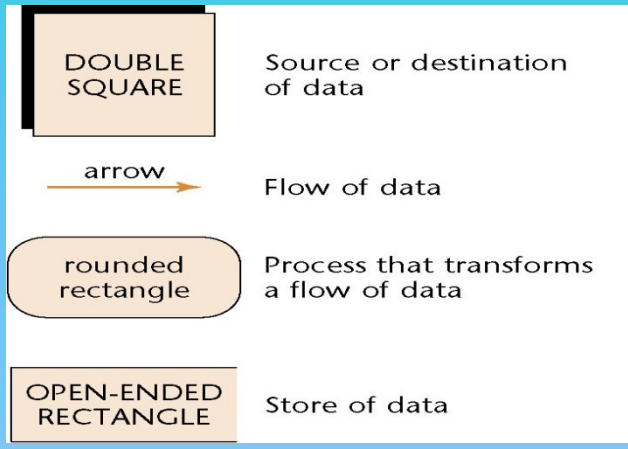
1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the **Data Flow**
4. Define the **logic of the Processes**
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the **Input-Output Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

Step 4. Define the Logic of the **Processes**

(rounded rectangles)

We have **Process** `give educational discount`

- Sally must explain the discount she gives to **educational institutions**
  - 10% on up to 4 packages
  - 15% on 5 or more



Gane and Sarsen's method for Structured

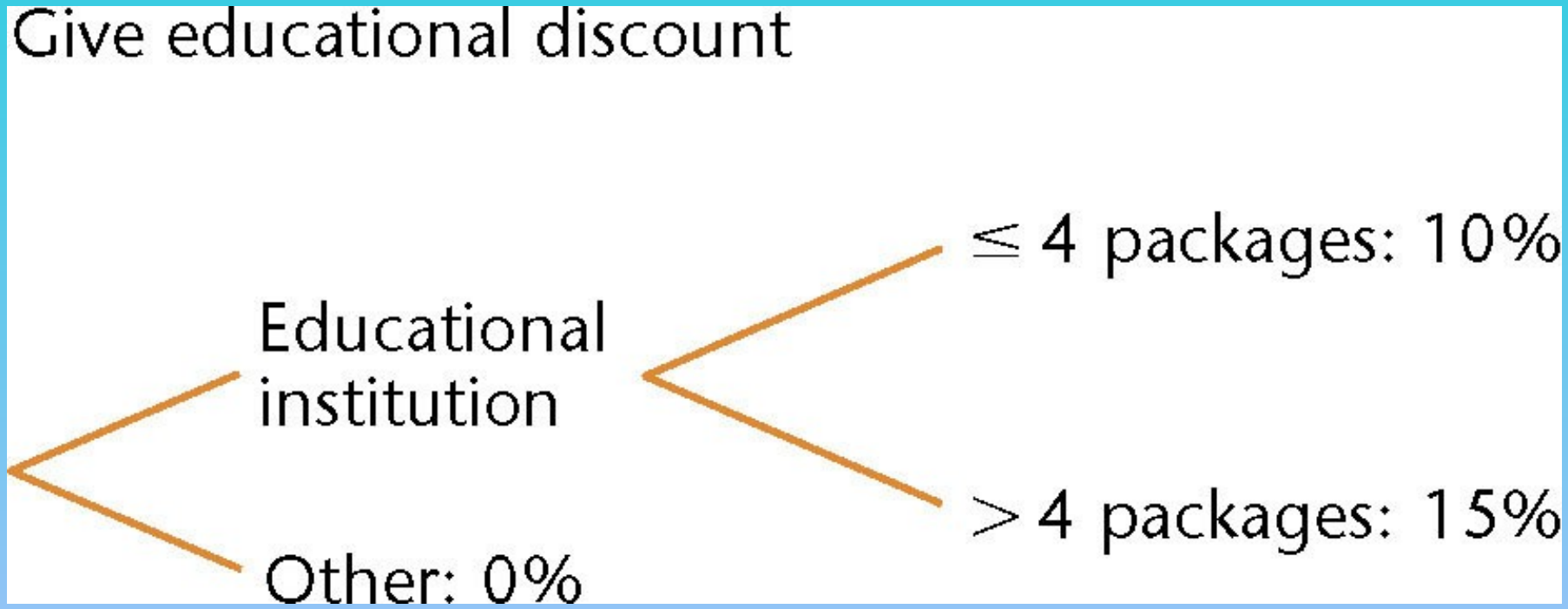
Nine-step method

- **Stepwise refinement** is used in many steps

1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the Data Flow
4. Define the logic of the Processes
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the Input-Output **Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

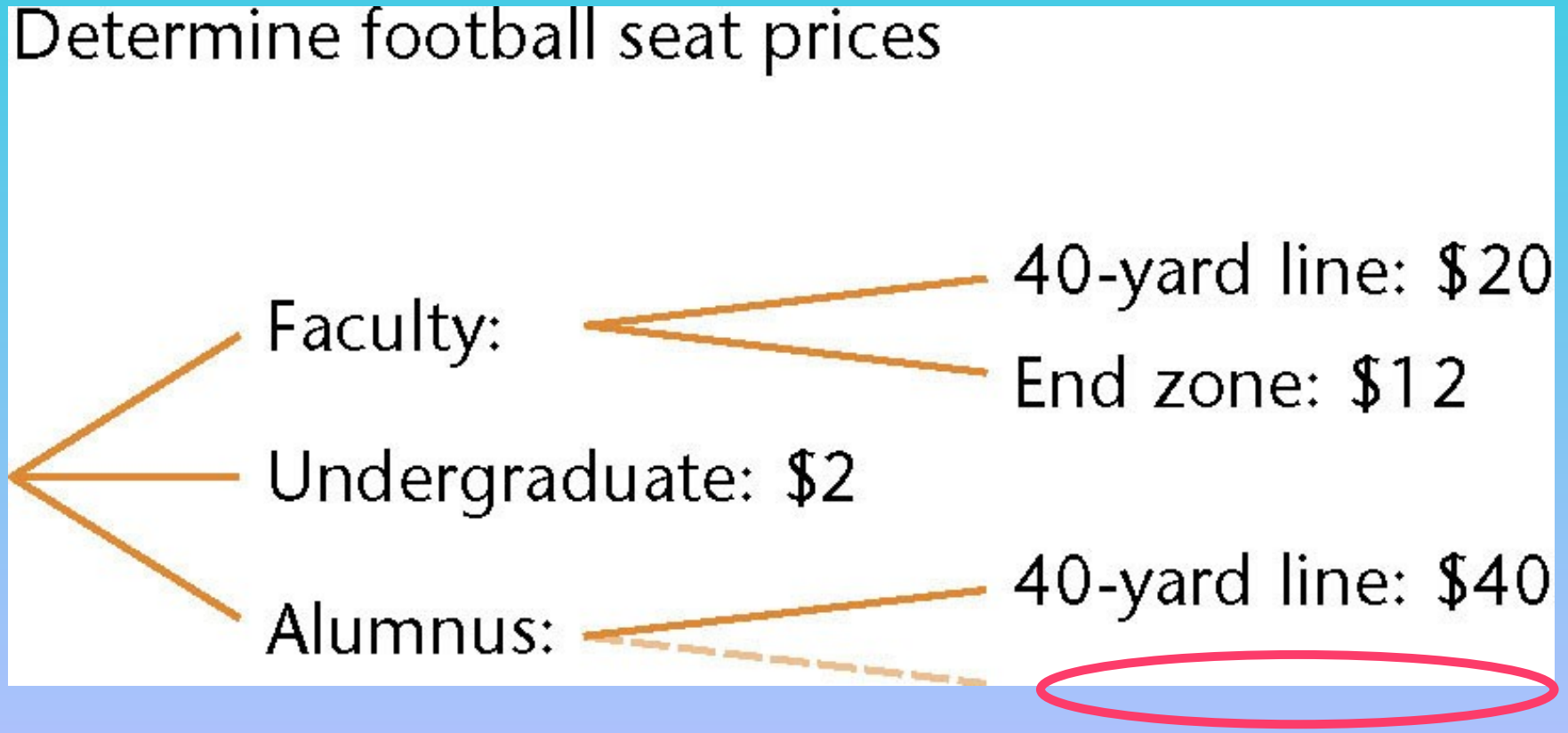
Step 4 . Define the Logic of the Processes

- Translate this into a **Decision Tree**



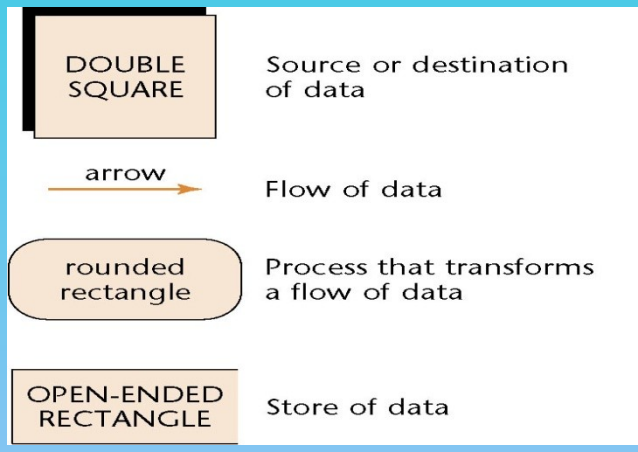
Step 4. Define the Logic of the Processes

- The advantage of a **Decision Tree**
  - **Missing items are quickly apparent**



Step 5. Define the Data Stores
(open rectangles)

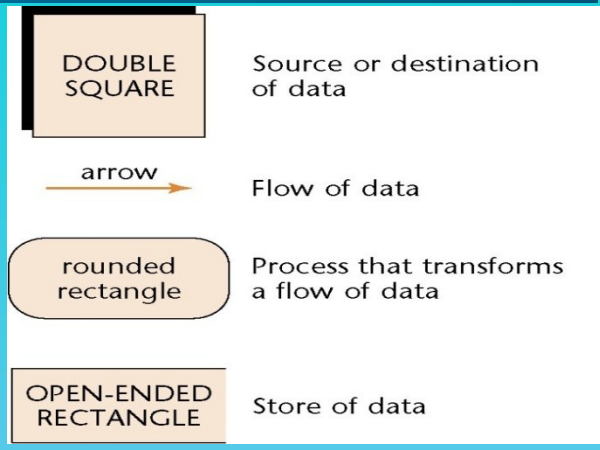
- Define the exact contents and representation (**format**)
  - COBOL: specify to `pic` level
  - Ada: specify `digits` or `delta`

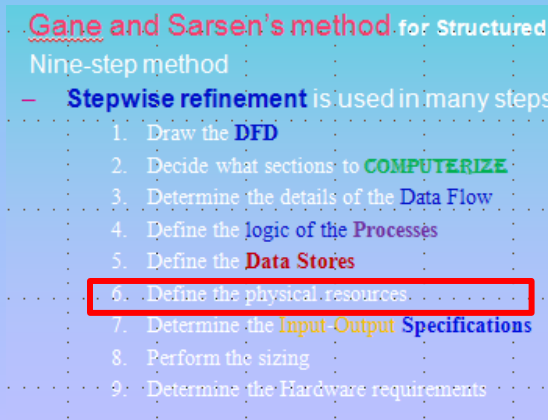


Gane and Sarsen's method for Structured
Nine-step method
– **Stepwise refinement** is used in many steps
1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the Data Flow
4. Define the logic of the Processes
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the Input-Output **Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

Step 6. Define the Physical Resources

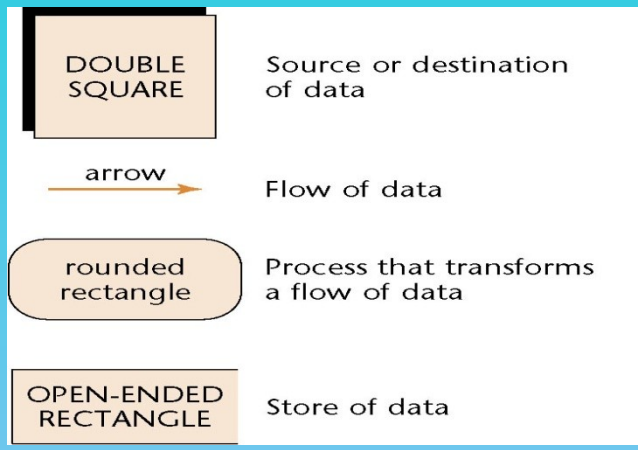
- For each File, specify
  - File name
  - Organization (Sequential, **Index**ed, etc.)
  - Storage medium
  - Blocking factor
  - **Records** (to Field level)
  - **TABLE/RELATION information, if a DBMS is to be used**



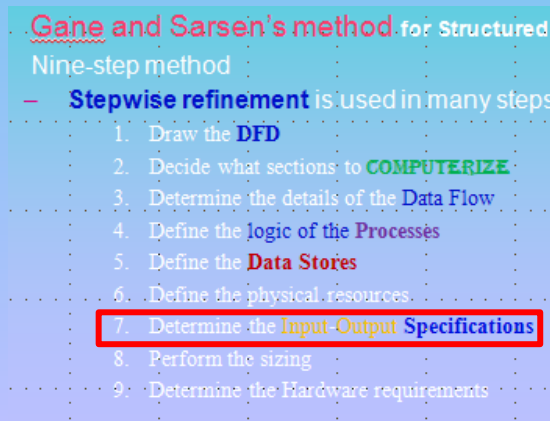


Step 7. Determine Input/Output Specifications

- Specify
  - Input Forms (V)
  - Input Screens (V)
  - Printed Output Reports (V)

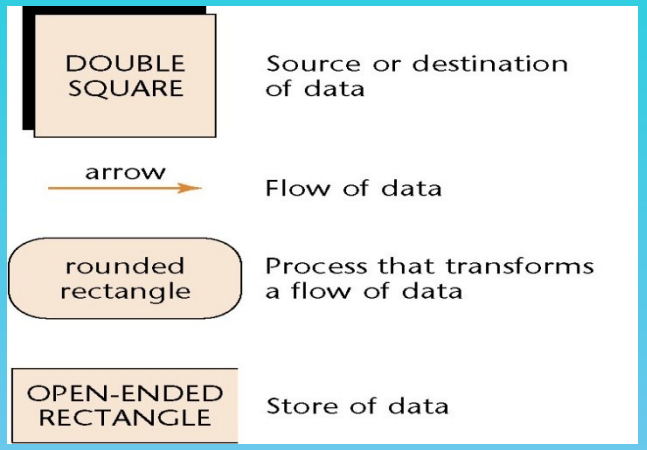


Input Forms and Output Reports Formats!



Step 8. Determine the **Sizing**

Obtain the numerical **data** needed in **Step 9** to determine the **Hardware** requirements



- Volume of **Input** (daily or hourly)
- Size, Frequency, Deadline of each printed **Report**
- Size, number of records passing between **CPU** and **mass storage**
- Size of each File

Gane and Sarsen's method for Structured

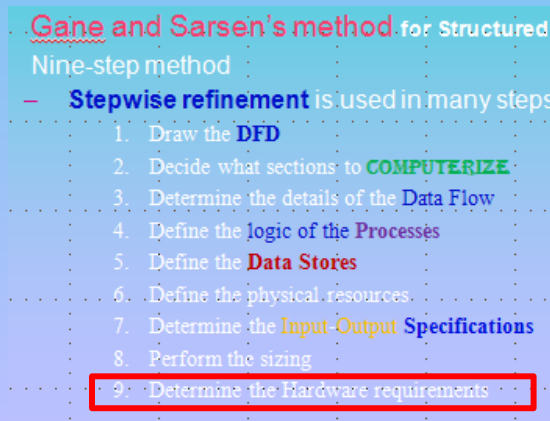
Nine-step method

- **Stepwise refinement** is used in many steps

1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the **Data Flow**
4. Define the **logic of the Processes**
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the **Input-Output Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

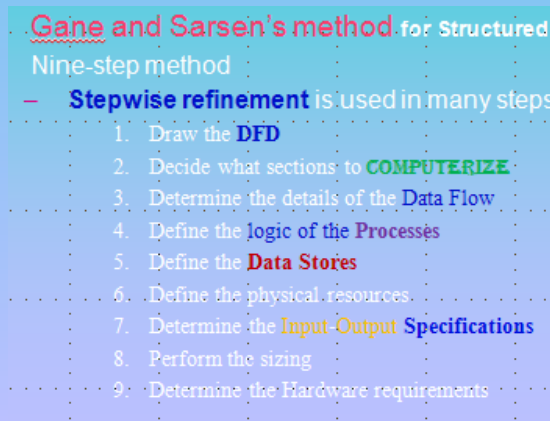
Step 9. Determine the **Hardware** Requirements

- Mass storage requirements
- Mass storage for back-up
- **Input** needs
- **Output** devices
- Is the existing **Hardware** adequate?
  - If not, recommend whether to buy or lease additional **Hardware**



However

- Response times **cannot** be determined
- The number of I/O channels **can only** be guessed
- CPU size and timing **can only** be guessed
- Nevertheless, **no other Method** provides these Data for **arbitrary products**



Overall

The method of **Gane and Sarsen/De Marco/ Yourdon** has resulted in major **improvements** in the **software Industry**

Gane and Sarsen's method for Structured

Nine-step method

- **Stepwise refinement** is used in many steps

1. Draw the **DFD**
2. Decide what sections to **COMPUTERIZE**
3. Determine the details of the Data Flow
4. Define the logic of the Processes
5. Define the **Data Stores**
6. Define the physical resources
7. Determine the Input-Output **Specifications**
8. Perform the sizing
9. Determine the Hardware requirements

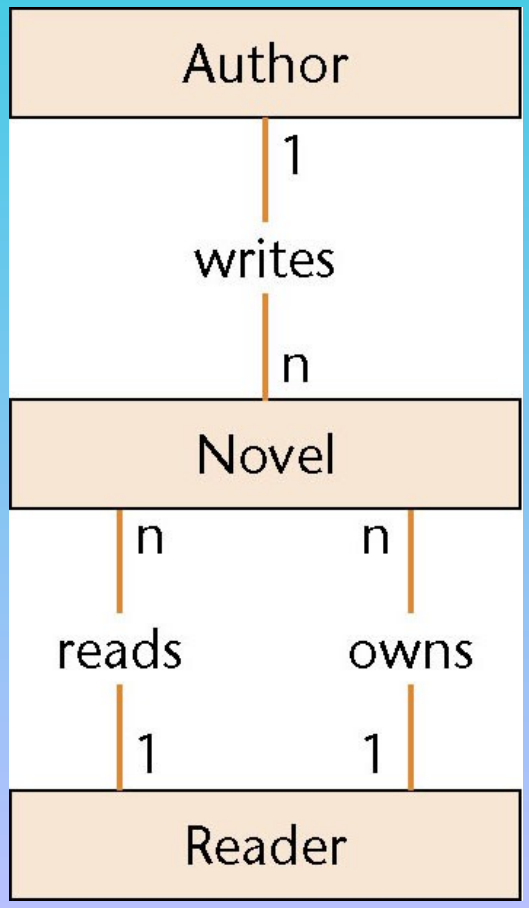
Other Semiformal Techniques

- **Semiformal (Graphical) techniques** for Classical **Analysis** include
  - PSL/PSA – Problem Statement Language/Problem Statement Analyzer
  - SADT – Structured Analysis and Design Techniques
  - SREM – Software Requirements Engineering Method

Entity Relationship Modeling

Semi Formal technique (geared towards **data Modeling**)

- **Widely used for Specifying Databases**
- Example **Relationships** between **Authors**, **Novels**, and **Readers**:



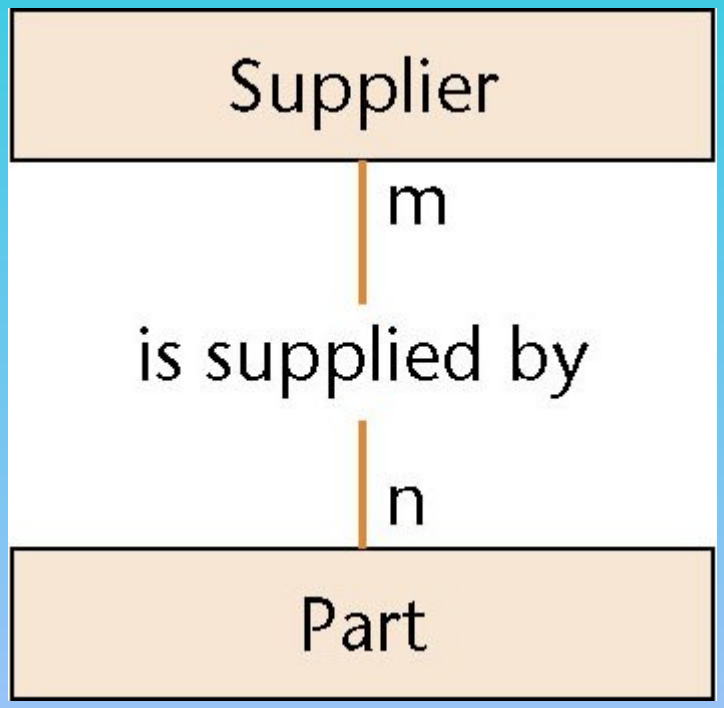
How many Es? **3**

How many Rs? **3**

Classical Analysis Method	Category	Strengths	Weaknesses
Entity-relationship modeling (Section 11.6)	Semiformal	Can be understood by the client	Not as precise as formal techniques
PSL/PSA (Section 11.5)		More precise than informal techniques	Generally cannot handle timing
SADT (Section 11.5)			
SREM (Section 11.5)			
Structured systems analysis (Section 11.3)			

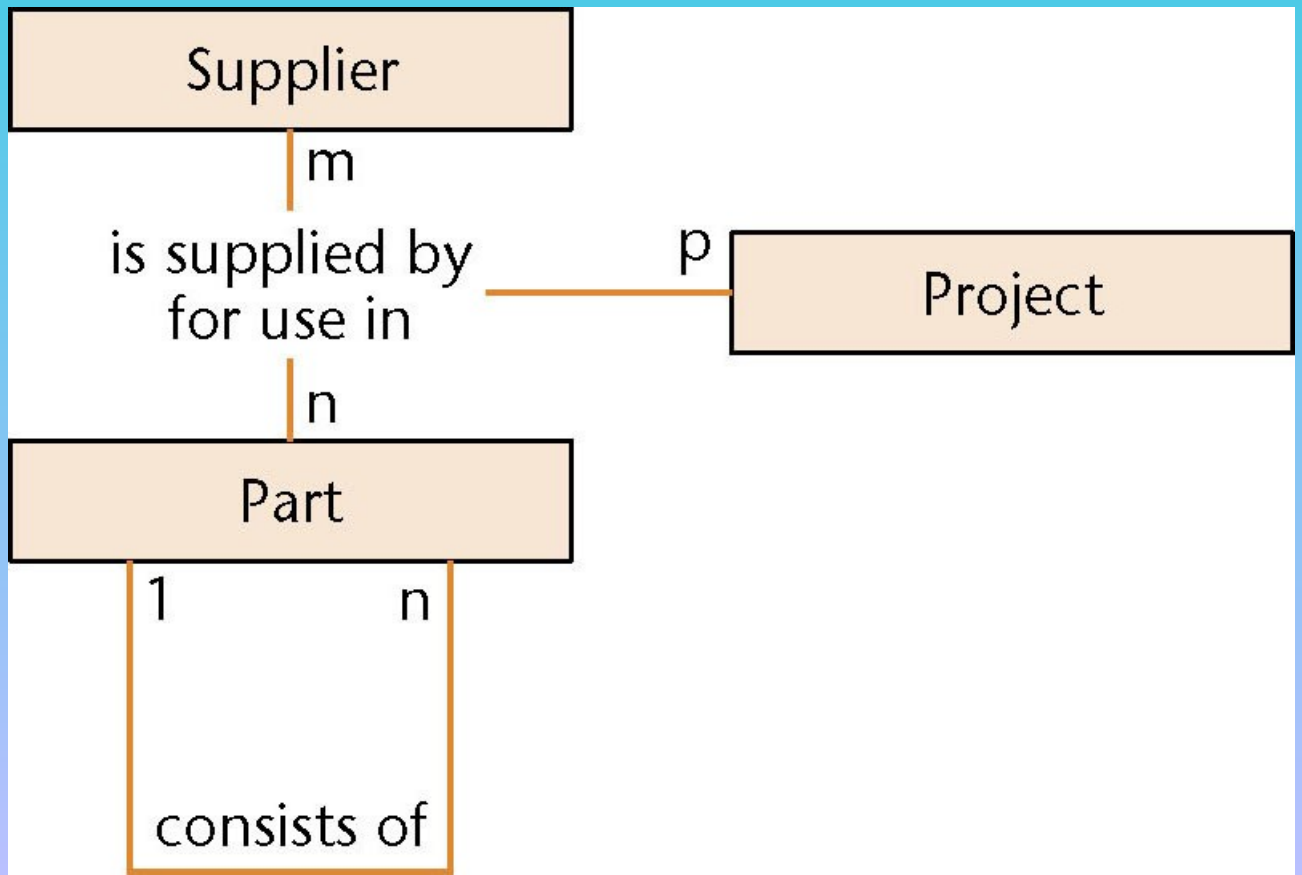
Entity Relationship Diagrams

- Many-to-many **Relationship**



Entity Relationship Diagrams

More complex Entity Relationship Diagrams (a Part can consist of a number of component Parts; A particular Part may be supplied by several Suppliers, depending on the Project;The various Parts supplied for a specific Project may come from different Suppliers)



Comparison of Classical **Analysis** Techniques

Classical Analysis Method	Category	Strengths	Weaknesses
Anna (Section 11.10) CSP (Section 11.10) Extended finite state machines (Section 11.7) Gist (Section 11.10) Petri nets (Section 11.8) VDM (Section 11.10) Z (Section 11.9)	Formal	Extremely precise Can reduce analysis faults Can reduce development cost and effort Can support correctness proving	Hard for the development team to learn Hard to use Almost impossible for most clients to understand

Formal Techniques

- **Formal Techniques** for Classical **Analysis** include
  - Finite State Machines
  - Petri Nets
  - Z

Anna (Section 11.10)	Formal	Extremely precise	Hard for the development team to learn
CSP (Section 11.10)		Can reduce analysis faults	Hard to use
Extended finite state machines (Section 11.7)		Can reduce development cost and effort	Almost impossible for most clients to understand
Gist (Section 11.10)		Can support correctness proving	
Petri nets (Section 11.8)			
VDM (Section 11.10)			
Z (Section 11.9)			

Finite State Machines

Anna (Section 11.10)	Formal	Extremely precise	Hard for the development team to learn
CSP (Section 11.10)		Can reduce analysis faults	Hard to use
Extended finite state machines (Section 11.7)		Can reduce development cost and effort	Almost impossible for most clients to understand
Gist (Section 11.10)		Can support correctness proving	
Petri nets (Section 11.8)			
VDM (Section 11.10)			
Z (Section 11.9)			

Finite State Machines

Case study

A Safe has a combination lock that can be in one of three positions, labeled 1, 2, and 3. The dial can be turned left or right (L or R). Thus there are **six possible dial movements**, namely 1L, 1R, 2L, 2R, 3L, and 3R.

The **combination** to the Safe is **1L**, **3R**, **2L**; any other dial movement will cause the **alarm** to go off

STD – State Transition Diagram

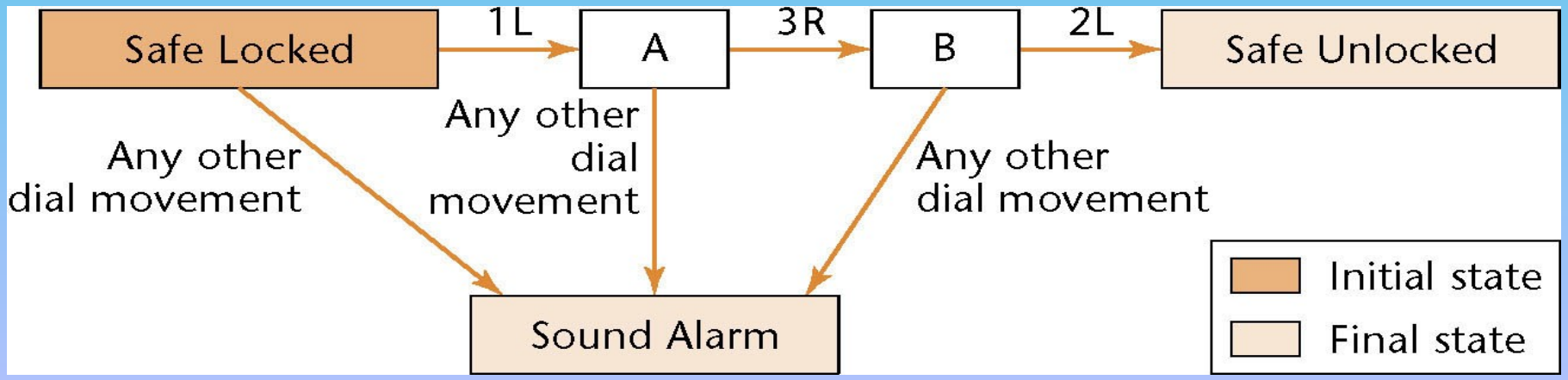


Figure 11.13

Finite State Machines (5 – tuple)

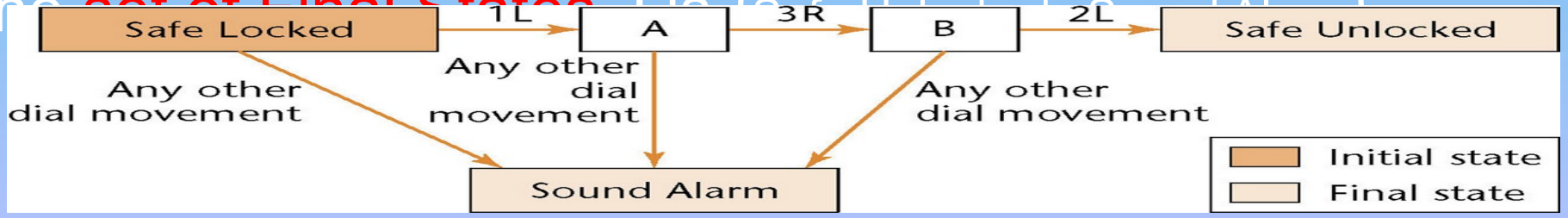
The set of States J is {Safe Locked, A, B, Safe Unlocked, Sound Alarm}

The set of **Inputs** K is {1L, 1R, 2L, 2R, 3L, 3R}

The **Transition Function** T is on the next slide

The Initial State J is {Safe Locked}

The set of Final States J is {Safe Unlocked}

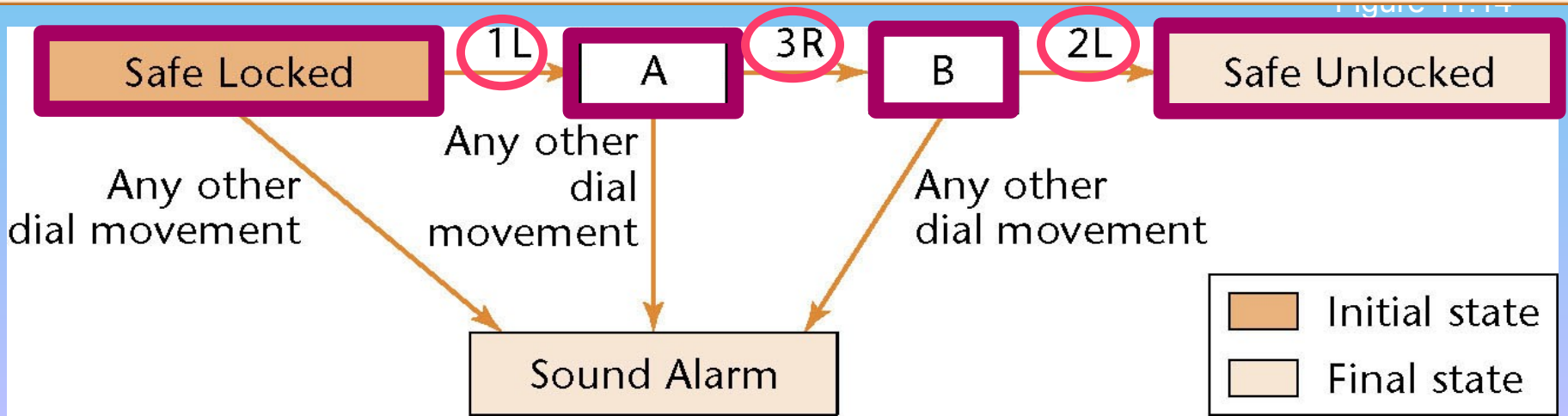


Finite State Machines

- Transition Table

Table of Next States

Dial Movement \ Current State	Safe Locked	A	B
1L	A	Sound alarm	Sound alarm
1R	Sound alarm	Sound alarm	Sound alarm
2L	Sound alarm	Sound alarm	Safe unlocked
2R	Sound alarm	Sound alarm	Sound alarm
3L	Sound alarm	Sound alarm	Sound alarm
3R	Sound alarm	B	Sound alarm



Extended Finite State Machine: Elevator Problem Case Study

A product is to be installed to control n **elevators** in a building with m floors. The problem concerns the logic required to move **elevators** between **floors** according to the following constraints:

1. Each **elevator** has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the **elevator**

2. Each **floor**, except the first and the top floor, has two buttons, one to request an up-**elevator**, one to request a down-**elevator**. These buttons illuminate when pressed. The illumination is canceled when an **elevator** visits the floor, then moves in the desired direction

3. If an **elevator** has no requests, it remains at its current floor with its doors closed

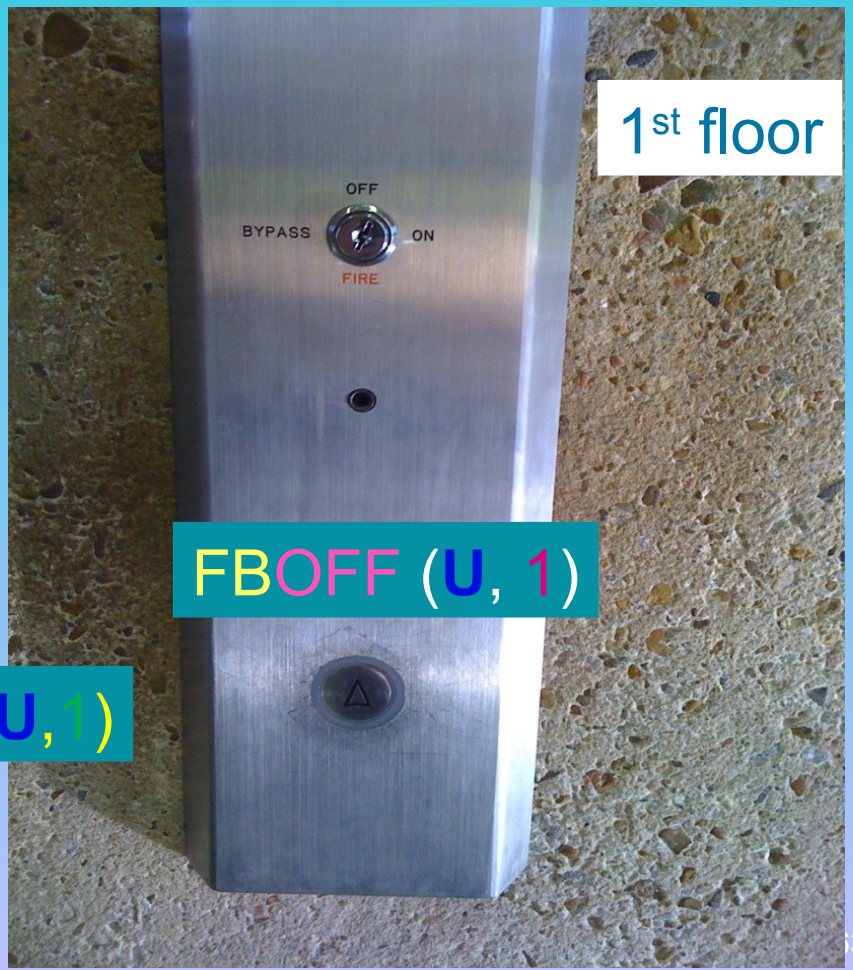
PGH (one side) Building Floors

6 floors f (numbered 1,2,3,4,5,6)

Floor Button FB

Floor f=1 only UP

Floor f=2,3,4,5 DOWN & UP



1st floor

FBOFF (U, 1)

FB (U,1)

2nd floor

FBOFF (U, 2)

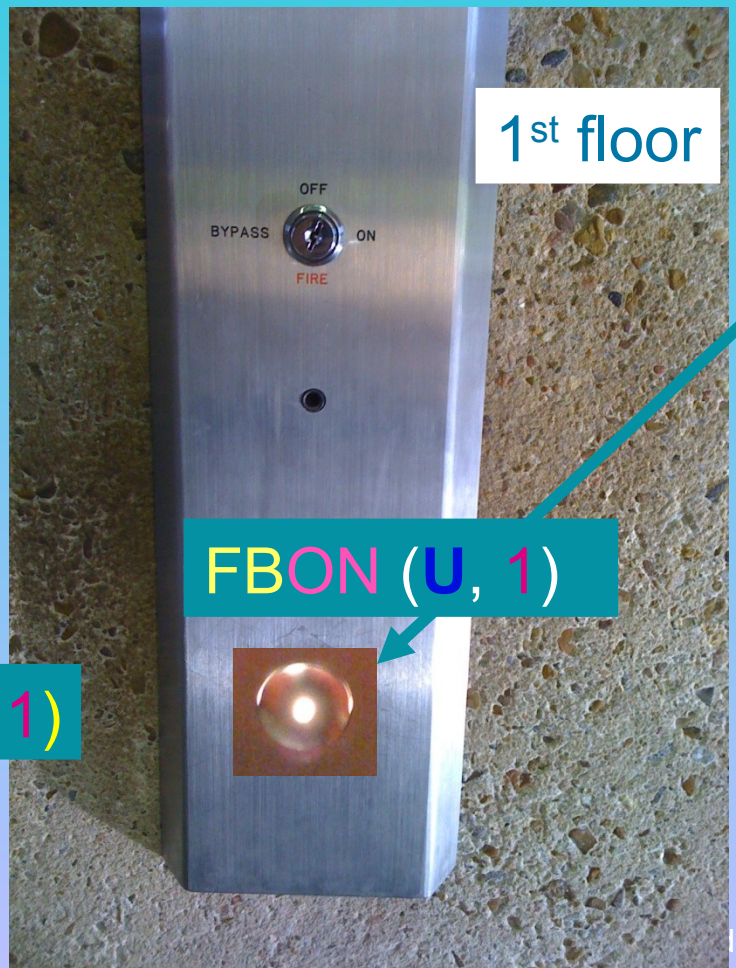
FB (U,2)

FB (D,2)

FBOFF (D, 2)

PGH (one side) Building Floors

6 floors f (numbered 1,2,3,4,5,6)



1st floor

Action:
Press Floor Button FBP

FBON (U, 1)

FB (U,1)

PGH (one side) Building Elevators

3 elevators **e** (numbered 1,2,3)

Elevator Button EB (**e**, f)

6 floors **f** (numbered 1,2,3,4,5,6)

Action:

Elevator Has Arrived At Floor

EHAF (**1**,**2**)

Extended Finite State Machines: Elevator Problem Case Study

- The **product** is specified now using the **Extended Finite State Machine**.

current state and **event** and **predicate** ⇒ **new state**

Extended Finite State Machines: Elevator Problem Case Study

There are two sets of Buttons

Elevator Buttons (n elevators X m buttons each)

- Inside each Elevator (n Elevators), one for each floor (m Floors)

Floor Buttons

- Two on each Floor, one for up-Elevator, one for down-Elevator

Elevator Buttons

EB (e, f):

- Elevator Button in Elevator e pressed to request Floor f
- EB(e,f) can be in 2 **States**, with Button ON (illuminated) or OFF

EBON (e, f): Elevator Button (e,f) **ON**

EBOFF (e, f): Elevator Button (e,f) **OFF**

- **2 events** are involved:

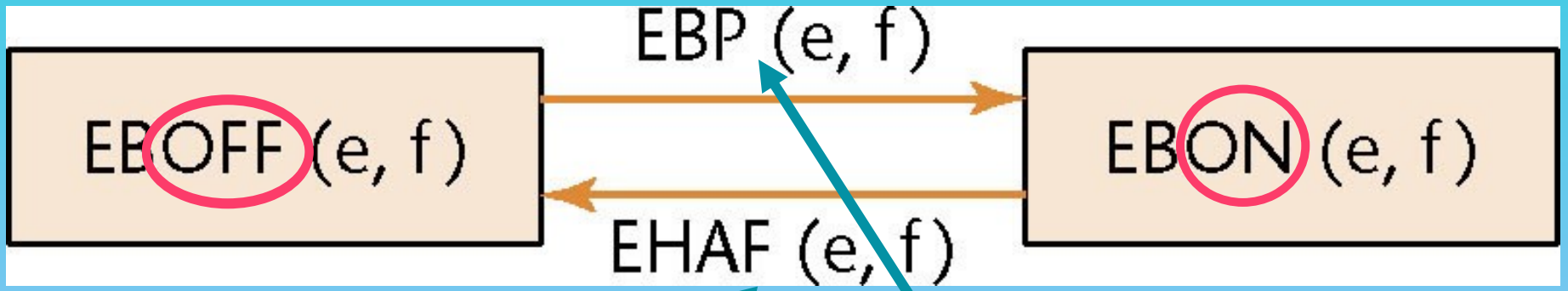
EBP (e, f): Elevator Button (e,f) Pressed

EHAF (e,f): Elevator e Has Arrived at Floor f

Elevator Buttons

- **2 States**

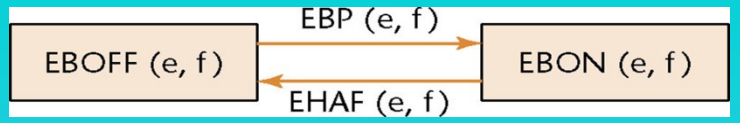
EBON (e, f): Elevator Button (e,f) ON
EBOFF (e,f): Elevator Button (e,f) OFF



- If an Elevator Button is ON and the Elevator Has Arrived at Floor f, then the Elevator Button is turned OFF

- If the Elevator Button is OFF and the Elevator Button is Pressed, then the Elevator Button comes ON

Elevator Buttons



Global predicate

V (**e**, **f**): Elevator **e** is **V**isiting (stopped at) floor **f**

Transition Rules

If Elevator Button (e, **f**) is **OFF** (current state) and Elevator Button (e, **f**) is Pressed (event) and Elevator e is **not** Visiting Floor **f** (predicate), then the Button is turned **ON**

EB**OFF** (e, **f**) and EBP (e, **f**) and **not** **V** (e, **f**) ⇒ EB**ON** (e, **f**)

If the Elevator Button is **ON** and the Elevator Has Arrived at Floor **f**, then it is turned **OFF**

EB**ON** (e, **f**) and EHAF (e, **f**) ⇒ EB**OFF** (e, **f**)

current state and **event** and **predicate** ⇒ **new state**

Floor Buttons

- FB (**d**, **f**):
  - Floor Button on Floor f that requests Elevator traveling in Direction d

- Events

  FBP (d, **f**): Floor Button (d, f) Pressed

  EHAF (1..n, **f**): Elevator 1 or … or n Has Arrived at Floor **f**

Floor Buttons

2 States

FBON (d, **f**): Floor Button (d, **f**) ON
FBOFF (d, **f**): Floor Button (d, **f**) OFF

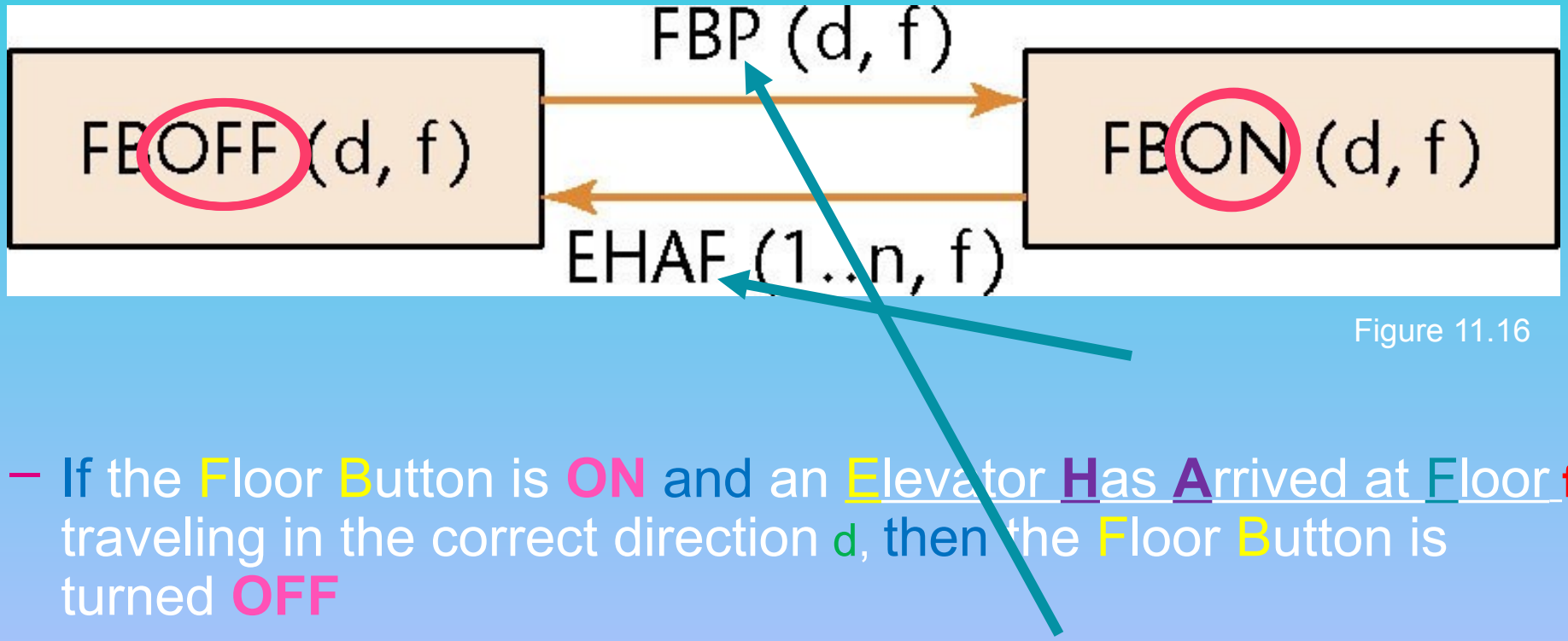
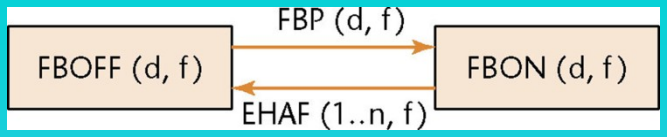


Figure 11.16

- If the Floor Button is **ON** and an Elevator **H**as **A**rrived at Floor **f**, traveling in the correct direction d, then the Floor Button is turned **OFF**

- If the Floor Button is **OFF** and the Floor Button is **P**ressed, then

Floor Buttons



Global predicate

S (d, e, **f**): Elevator e is Visiting Floor **f** direction of motion is up (d = U), down (d = D), or no requests are pending (d = N

Transition rules

If the Floor Button at Floor **f** for motion in the direction d is OFF and the Button is Pushed and none of the Elevators currently is Visiting Floor **f** about to move in direction d, then the Floor Button is turned ON

FBOFF (d, **f**) and FBP (d, **f**) and **not** **S** (d, 1..n, **f**) ⇒ FBON (d, **f**)

If the Floor Button is ON and at least one Elevator Has Arrived at Floor **f** and the Elevator is about to move in Direction d, then the Button is turned OFF

current state and **event** and **predicate** ⇒ **new state**

S (d, 1..n, **f**) ⇒ FBOFF (d, **f**),

Extended Finite State Machines

The power of an **EFSM** to **Specify**(Analysis) complex systems

There is no need for complex preconditions and postconditions

Specifications(**Analysis**) take the simple form

current state and **event** and **predicate** ⇒ **next state**

Extended Finite State Machines

- Using an **EFSM**, a **Specification**(**Analysis**) is
  - Easy to write down
  - Easy to validate
  - Easy to convert into **Design**
  - Easy to convert into **Code automatically**
  - More precise than Graphical Methods
  - Almost as easy to understand
  - Easy to **Maintain**

- However
  - Timing considerations **are not handled**

Petri Nets

Anna (Section 11.10)	Formal	Extremely precise	Hard for the development team to learn
CSP (Section 11.10)		Can reduce analysis faults	Hard to use
Extended finite state machines (Section 11.7)		Can reduce development cost and effort	Almost impossible for most clients to understand
Gist (Section 11.10)			
Petri nets (Section 11.8)		Can support correctness proving	
VDM (Section 11.10)			
Z (Section 11.9)			

Petri Nets

- A difficulty with **Specifying** (**Analysis**) **real-time systems** is timing
  - Synchronization problems
  - Race conditions
  - Deadlock

- Consequence of **Specifications** (**Analysis**) not properly drawn up, thus poor **Design** or Faulty **Implementation**

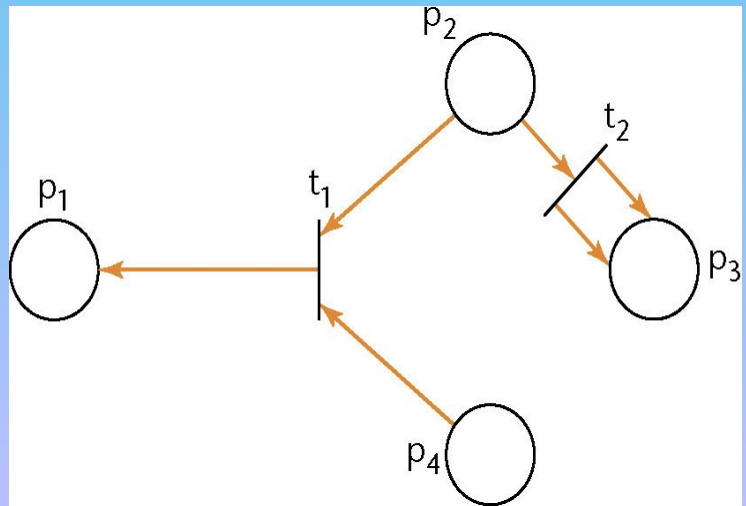
Petri Nets

Petri Nets

- A powerful technique for **Specifying** (**Analysis**) **systems** that have potential problems with interrelations
- Wide applicability in computer science such as Performance Evaluation, Operating Systems, and Software Engineering

A Petri Net consists of four parts:

- A **set** of **Places** P
- A **set** of **Transitions** T
- **Input** function I
- **Output** function O



Petri Nets

- **Set** of **Places** P is $\{p_1, p_2, p_3, p_4\}$
- **Set** of **Transitions** T is $\{t_1, t_2\}$

Places? # $P = \{p_1, p_2, \ldots, p_n\}$
Transitions? # $T = \{t_1, t_2, \ldots, t_m\}$
Input Functions? # $I : P \rightarrow T^{\infty}$
Output Functions? # $O : T \rightarrow P^{\infty}$

- **Input** Functions:

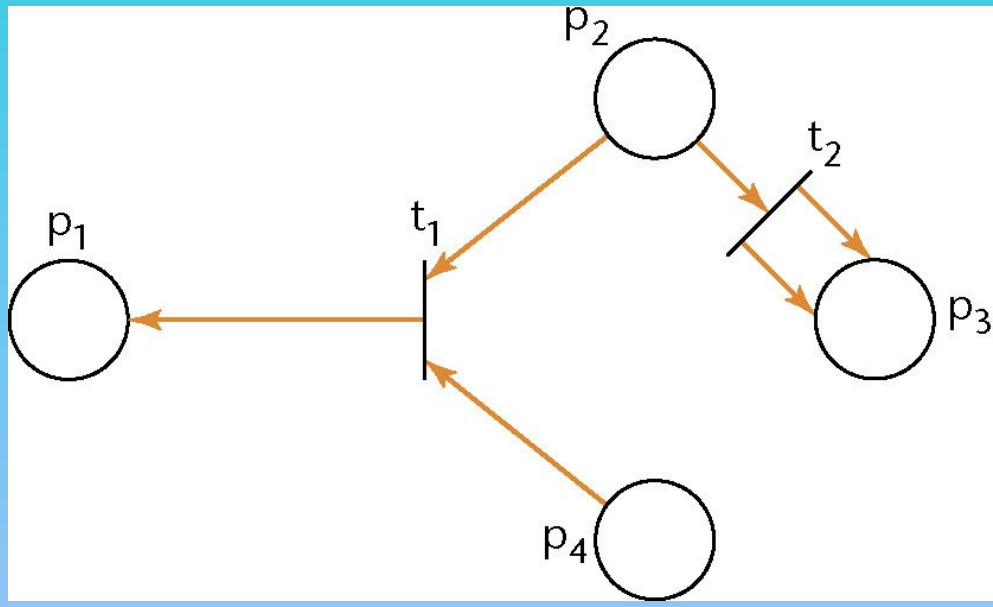
  $I(t_1) = \{p_2, p_4\}$

  $I(t_2) = \{p_2\}$

- **Output** Functions:

  $O(t_1) = \{p_1\}$

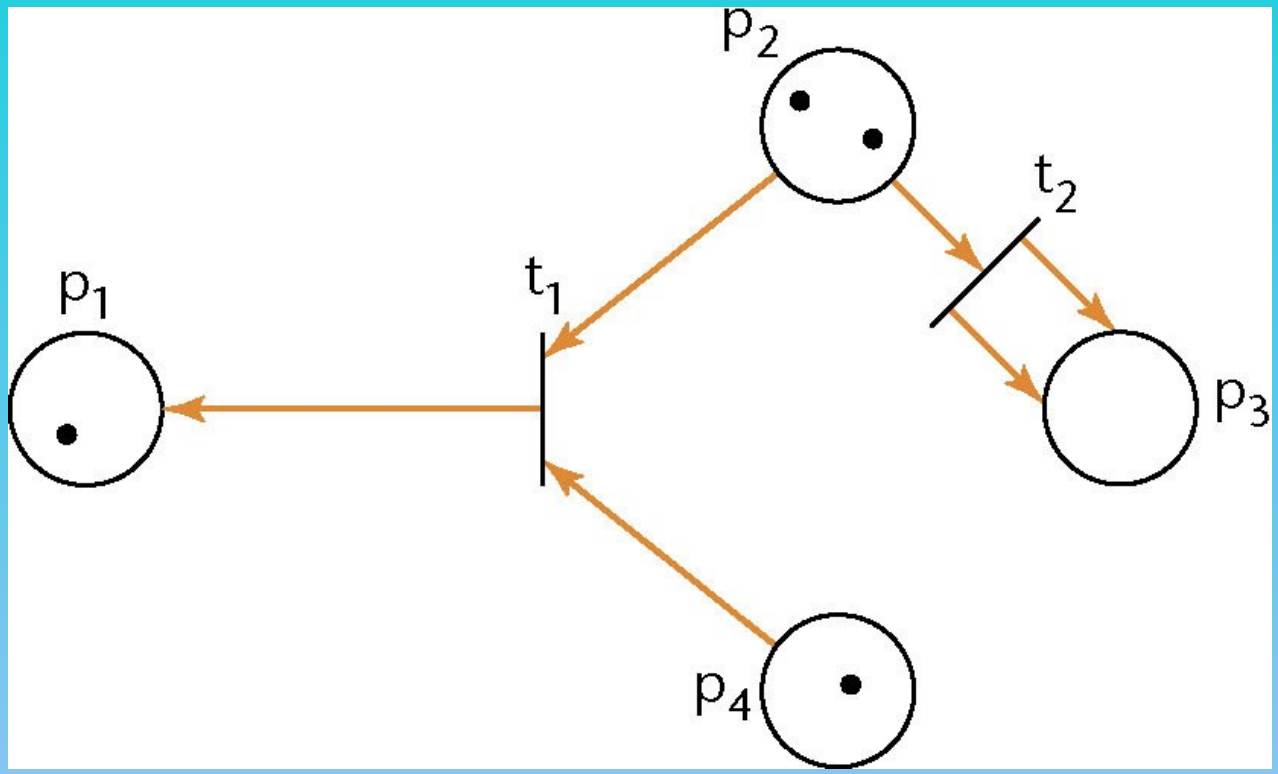
  $O(t_2) = \{p_3, p_3\}$



EXAM 3 REVIEW

Petri Nets

How many tokens?

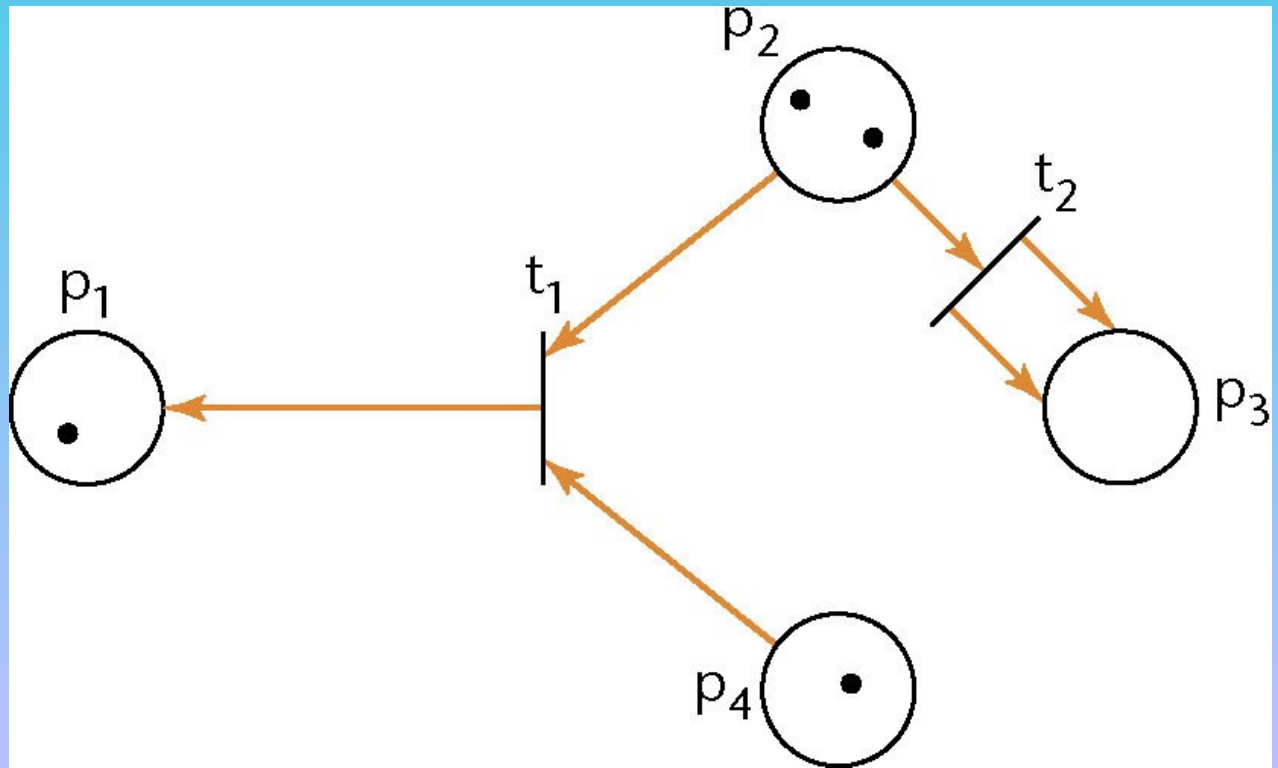


- Four tokens: 1 in p_1, 2 in p_2, 0 in p_3, and 1 in p_4
  - Represented by the M vector (1,2,0,1)

Petri Nets

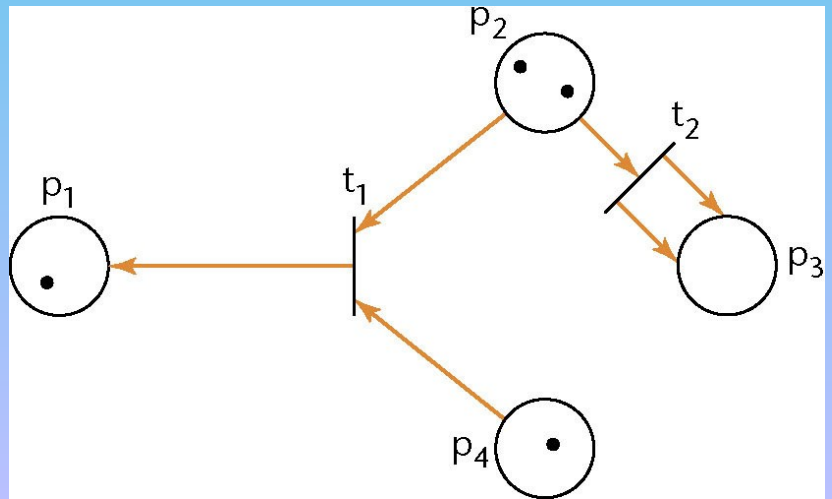
- A **Transition** t_i is enabled if each of its **input Places** has as many tokens in it as there are **arcs** from that Place to that **Transition**

What **Transitions** t_i are enabled? t_1 & t_2



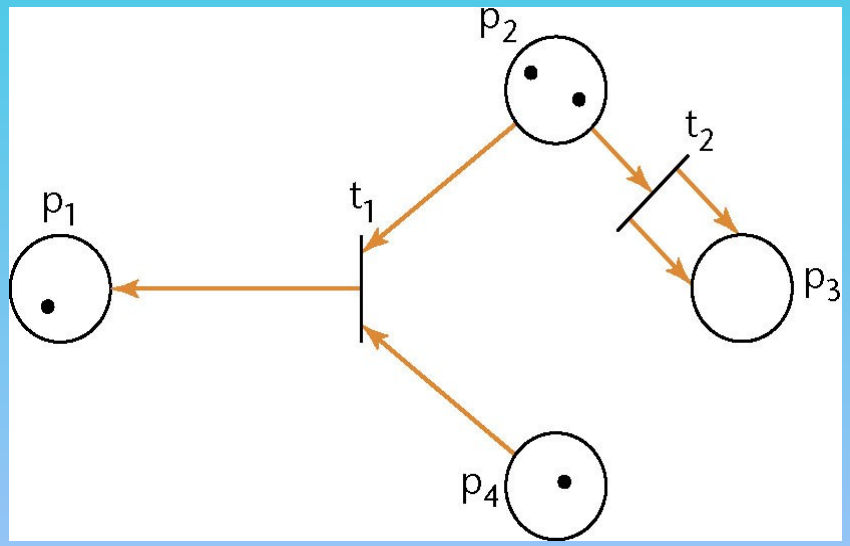
Petri Nets

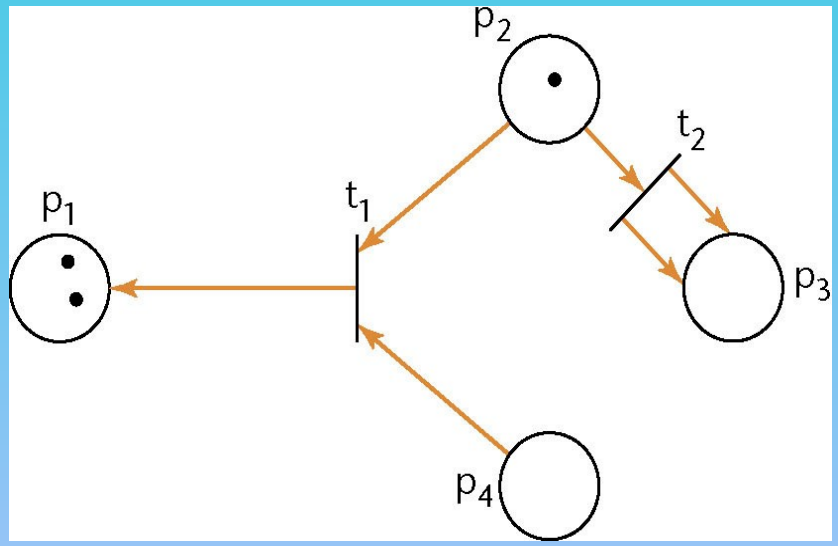
- **Transition** t_1 is enabled (ready to fire)
  - If t_1 fires, 1 token is removed from p_2 and 1 from p_4, and 1 new token is placed in p_1
- **Transition** t_2 is also enabled
- Important:
  - The number of tokens is not conserved



Petri Nets

- Petri Nets are nondeterministic; that is, if more than one **Transition** can fire, then any one of them can be fired.
  - Suppose t_1 fires





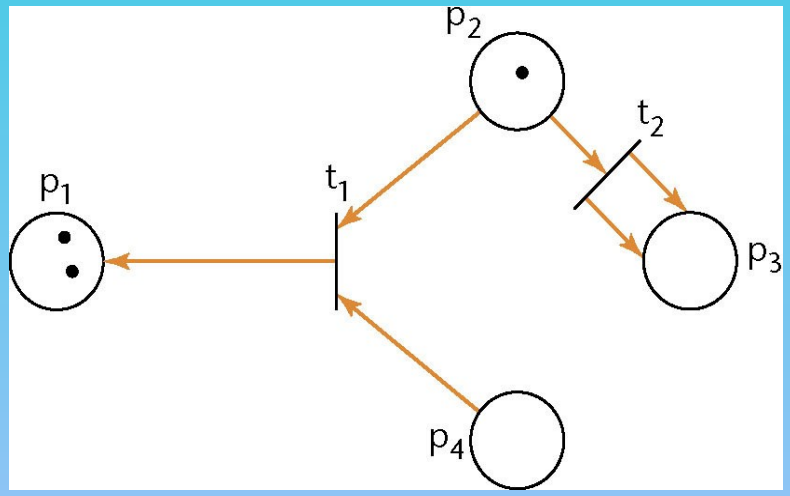
The Marking is (1,2,0,1)

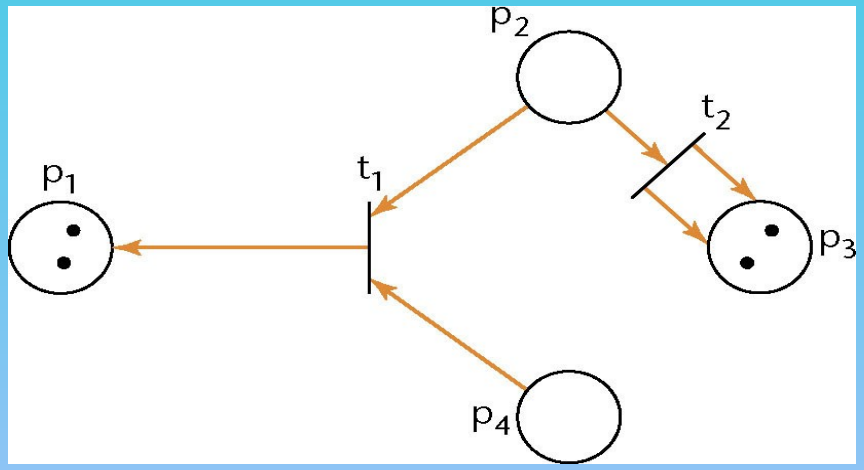
The Marking?

The Marking is (2,1,0,0)

Petri Nets

- Now only t_2 is enabled
  - It fires





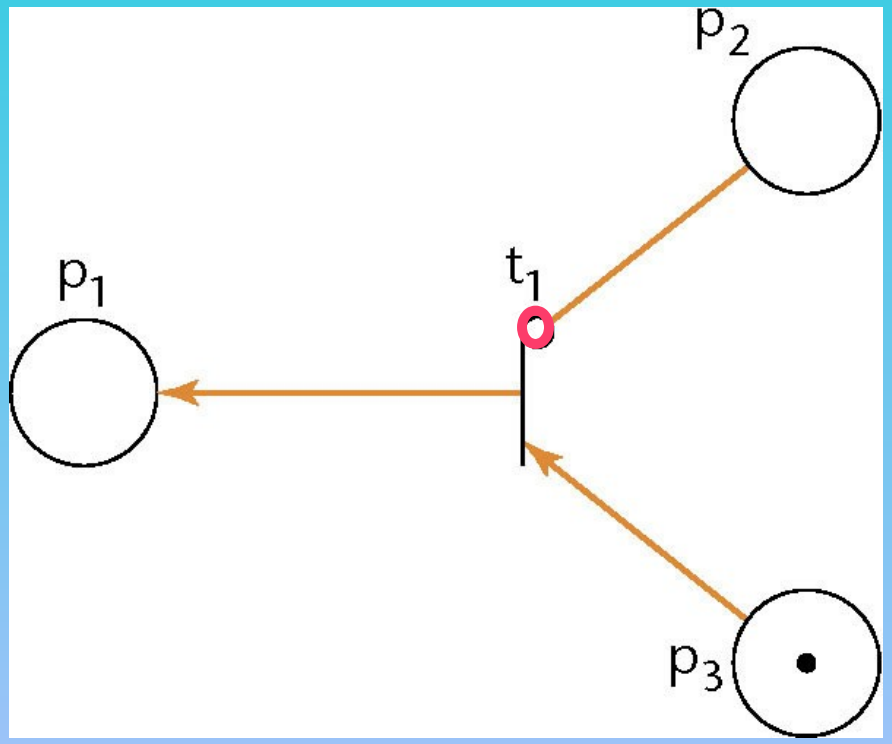
The Marking is (2,1,0,0)

The Marking?

The Marking is now (2,0,2,0)

Petri Nets

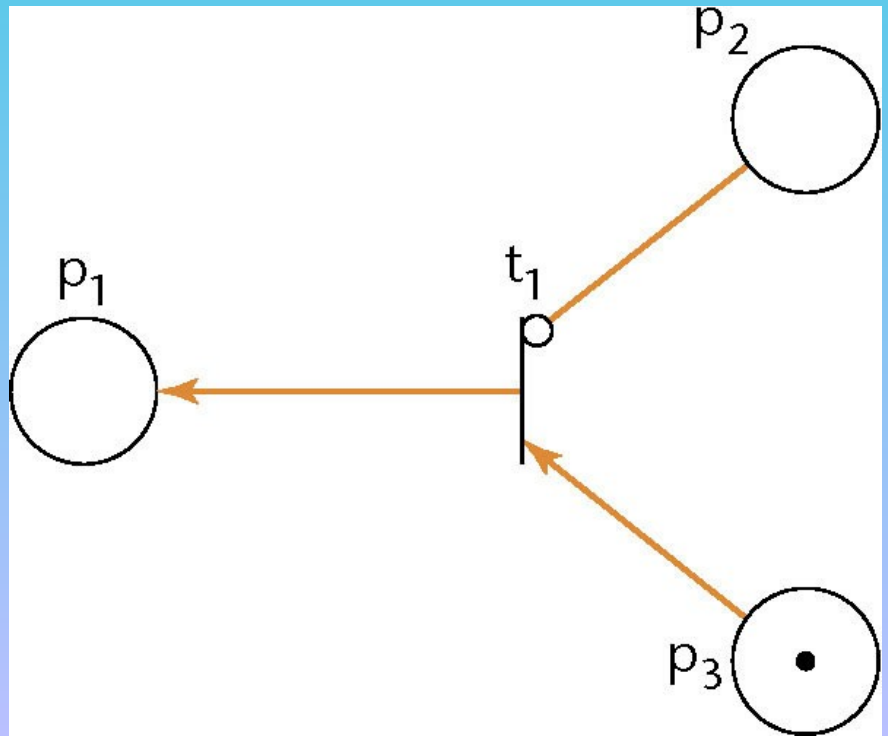
- Inhibitor arcs
  - An inhibitor arc is marked by a small circle, not an arrowhead



- **Transition** t_1 is enabled

Petri Nets

- In general, a **Transition** is enabled if there is at least one token on each (normal) input arc, and no tokens on any inhibitor input arcs



PGH (one side) Building Elevators: Elevator Buttons

Elevator Button EB is a Place EB_f

6 floors f (numbered 1,2,3,4,5,6)

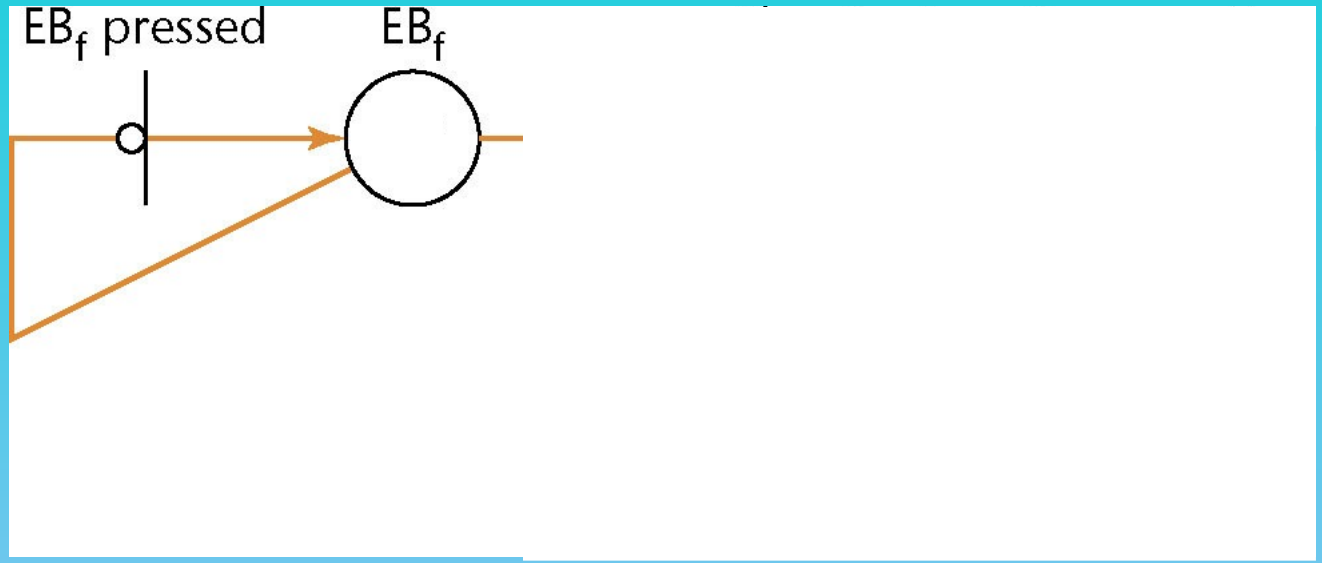
A **token** in

EB_2

means that the **Elevator Button** is

Illuminated

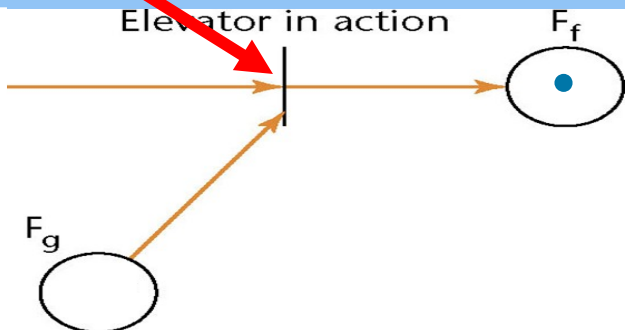
Petri Nets: The Elevator Problem Case Study
Elevator Button



- If Button EB$_f$ is not illuminated, no token is in Place and transition EB$_f$ pressed is enabled
  - The transition fires, a new token is placed in EB$_f$
- Now, no matter how many times the Button is Pressed transition EB$_f$ pressed cannot be enabled

Petri Nets: The **Elevator Problem** Case Study

- When the Elevator reaches Floor g
  - A token is in Place F_g
  - **Transition** Elevator in action is enabled, and then fires
- The tokens in EB_f and F_g are removed
  - This turns off the light in Button EB_f
- A new token appears in F_f
  - This brings the Elevator from Floor g to Floor f



Petri Nets

Petri Nets possess the expressive power necessary for **Specifying** synchronization aspects of real-time systems

Petri Nets can **also be used for Design**

Z

Z (pronounced “zed”) is a formal **Specification**(**Analysis**) Language

Anna (Section 11.10)	Formal	Extremely precise	Hard for the development team to learn
CSP (Section 11.10)		Can reduce analysis faults	Hard to use
Extended finite state machines (Section 11.7)		Can reduce development cost and effort	Almost impossible for most clients to understand
Gist (Section 11.10)		Can support correctness proving	
Petri nets (Section 11.8)			
VDM (Section 11.10)			
Z (Section 11.9)			

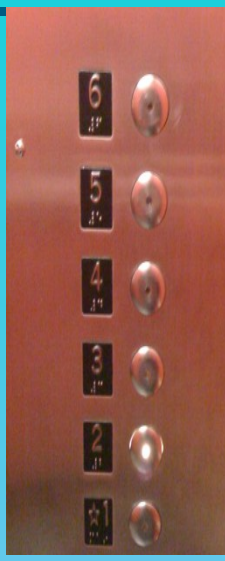
Z: The Elevator Problem Case Study

- A Z **Specification** consists of four sections:
  - 1. Given sets, data types, and constants
  - 2. **State** definition
  - 3. **Initial State**
  - 4. **Operations**

1. Given sets **Elevator Problem**

In this **problem** there are four subsets of Button

- The floor_buttons
- The elevator_buttons
- buttons (the set of **all** buttons in the **Elevator Problem**)
- pushed (the set of buttons that have been Pushed)



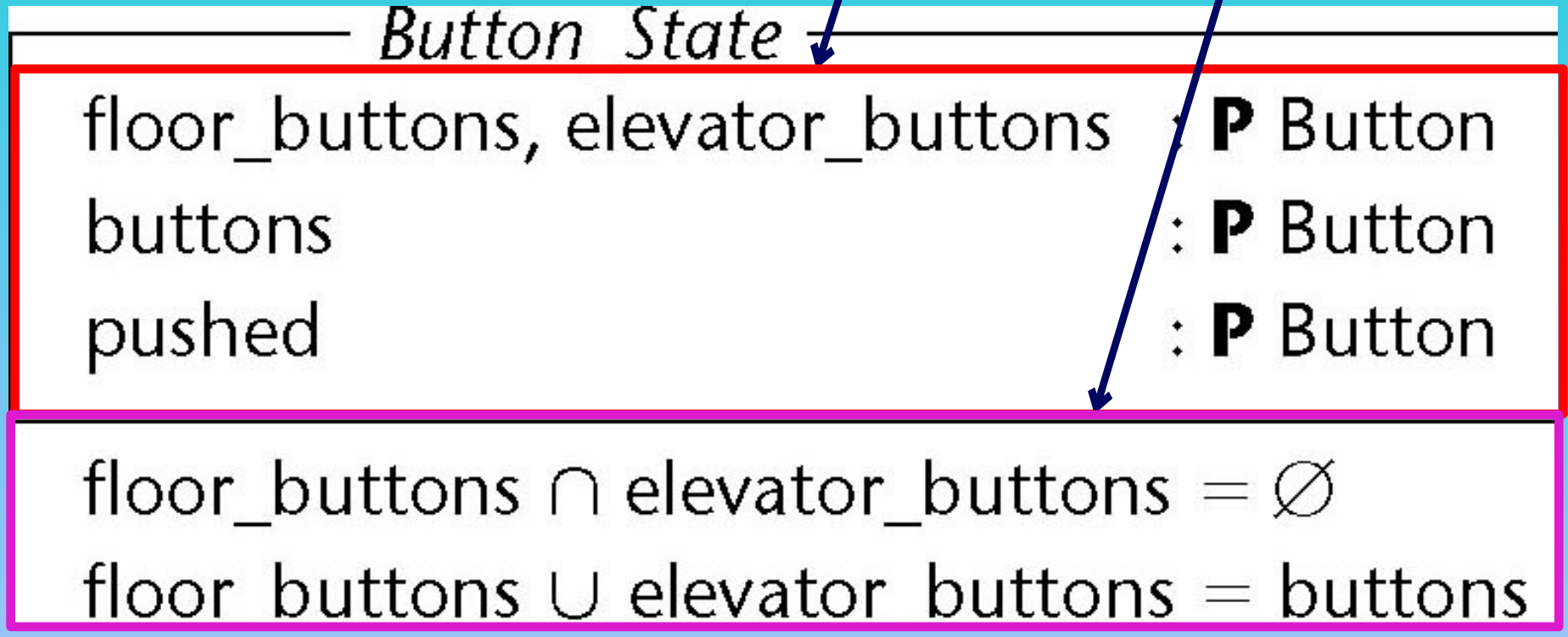
$$\begin{aligned} &floor\_buttons, elevator\_buttons : \mathbb{P}\ Button \\ &buttons : \mathbb{P}\ Button \\ &pushed : \mathbb{P}\ Button \end{aligned}$$

A Z **Specification** consists of four sections:

- 1. Given sets, data types, and constants
- 2. **State** definition
- 3. **Initial state**
- 4. **Operations**

2. State Definition **Elevator Problem**

- This is a **Button_State** (**variables** and predicates)



A Z **Specification** consists of four sections:

1. Given sets, data types, and constants
2. **State** definition
3. **Initial state**
4. **Operations**

3. Initial State **Elevator Problem**

- The State when the **system** is first turned on

$$Button\_Init = [Button\_State' \mid \mathbf{pushed'} = \varnothing]$$

This schema asserts that, when the **elevator system** first is turned on, the set **pushed** initially is empty; that is, all the buttons are off.

A Z **Specification** consists of four sections:

- 1. Given sets, data types, and constants
- 2. **State** definition
- 3. **Initial state**
- 4. **Operations**

4. Operations Elevator Problem

Δ means this operation changes the State of Button_State

The operation has one input variable, button?

Push_Button

$\Delta Button\_State$

button?: Button

$$(button? \in buttons) \land$$
$$(((button? \notin pushed) \land (pushed' = pushed \cup \{button?\})) \lor$$
$$((button? \in pushed) \land (pushed' = pushed)))$$

- A button? pushed for the first time is turned on, and added to set *pushed'*
- Without the third precondit unspecified

A Z **Specification** consists of four sections:

- 1. Given sets, data types, and constants
- 2. **State** definition
- 3. **Initial state**
- 4. **Operations**

Analysis of Z

Z is the most widely used formal **Specification** Language

It has been used to **Specify**(**Analysis**)

- **CICS** (part), the **IBM's transaction processing system**
- An **oscilloscope**
- A **CASE tool**
- Many **large-scale projects** (especially in **Europe**)

A Z **Specification** consists of four sections:

1. Given sets, data types, and constants
2. **State** definition
3. **Initial state**
4. **Operations**

Analysis of **Z**

- Reasons for the great success of **Z**
  - It is easy to find faults in **Z Specifications** (**Analysis**)
  - The specifier must be extremely precise
  - We can prove correctness (we do not have to)
  - Only high-school math needed to *read* **Z**
  - **Z** decreases **Development** time

  - A "translation" of a **Z Specification** (**Analysis**) into English (or another natural language) is **clearer** than an **informal Specification** (**Analysis**)

A Z **Specification** consists of four sections:

1. Given sets, data types, and constants
2. **State** definition
3. **Initial state**
4. **Operations**

Other **Formal** Techniques

Anna

- For Ada

Gist, Refine

- Knowledge-based

Anna (Section 11.10)	Formal	Extremely precise	Hard for the development team to learn
CSP (Section 11.10)		Can reduce analysis faults	Hard to use
Extended finite state machines (Section 11.7)		Can reduce development cost and effort	Almost impossible for most clients to understand
Gist (Section 11.10)			
Petri nets (Section 11.8)			
VDM (Section 11.10)		Can support correctness proving	
Z (Section 11.9)			

VDM – Vienna Definition **Method**

- Uses denotational semantics
- It can also be used to **Design** and **Implementation**

CSP – Communicating Sequential Processes

- CSP **Specifications** (**Analysis**) are executable
- Provides the framework to go from **Specifications** (**Analysis**) to **Design** to **Implementation** preserving **validity**

Newer Methods

- Many are untested in practice
- There are risks involved
  - **Training Costs**
  - Adjustment from the classroom to an **actual project**
  - **CASE tools may not work properly**

» However, possible **gains *may* be huge**

Which **Analysis** Technique Should Be Used?

- It depends on the
  - Project
  - **Development Team**
  - **Management Team**
  - Myriad other factors
- It is unwise to ignore the latest developments

Testing during Classical Analysis

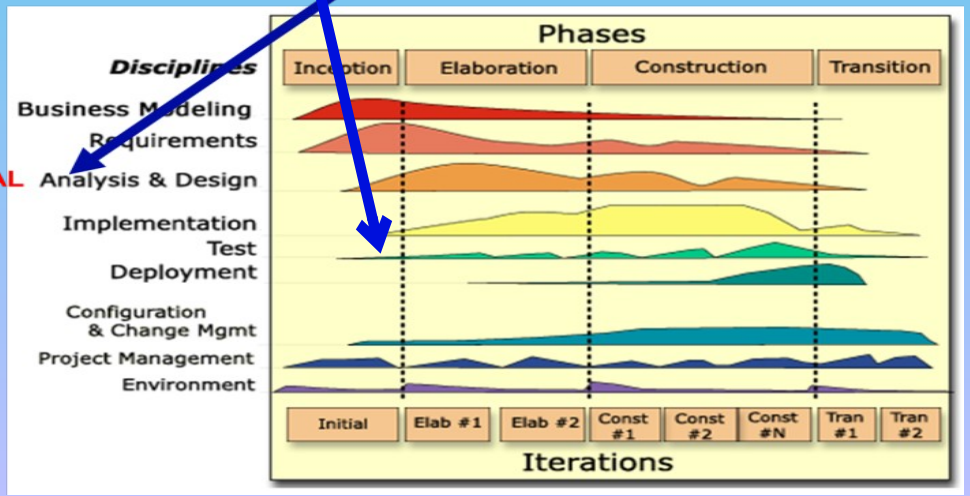
Specification (**Analysis**) **Inspection**

- Aided by **Fault Checklist**

Results of Doolan [1992]

- 2 million lines of FORTRAN
- 1 hour of **Inspecting** saved 30 hours of execution-based **Testing**

CLASSICAL **ANALYSIS**

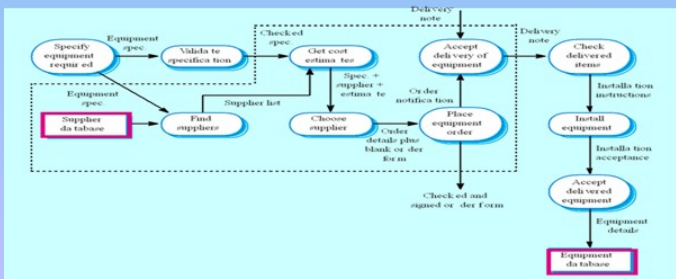


Metrics for CASE Tools

- **Five** fundamental **Metrics**: **size**, **cost**, **duration**, **effort**, and **Quality**

- **Metrics** for "predicting" the **size** of a target product
  - Total **number of items** in the **Data Dictionary**
  - The **number of items** of each **type**
  - **processes** and **modules**
- **Quality**
  - **Fault statistics**
  - The **Number, Type of each Fault**
  - The **Rate of Fault Detection**

DFD Model



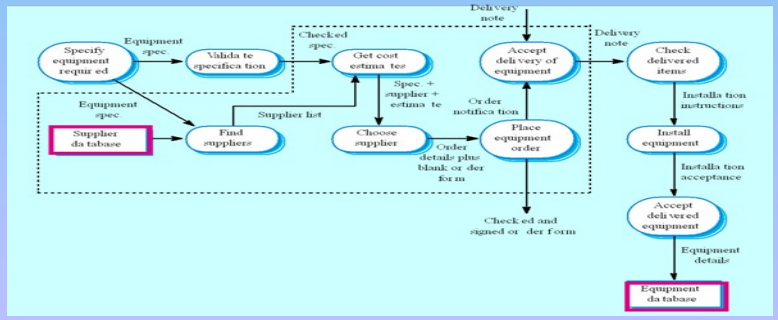
Challenges of Classical Analysis

- A Specification (**Analysis**) Document must be
  - Informal enough for the **Client**; but
  - Formal enough for the **Development Team**

- **Analysis** ("**WHAT**") should not cross the boundary into **Design** ("**HOW**")

- Do not try to assign **modules** to **process** boxes of the **DFD**s until the Classical ***Design*** Phase

DFD Model

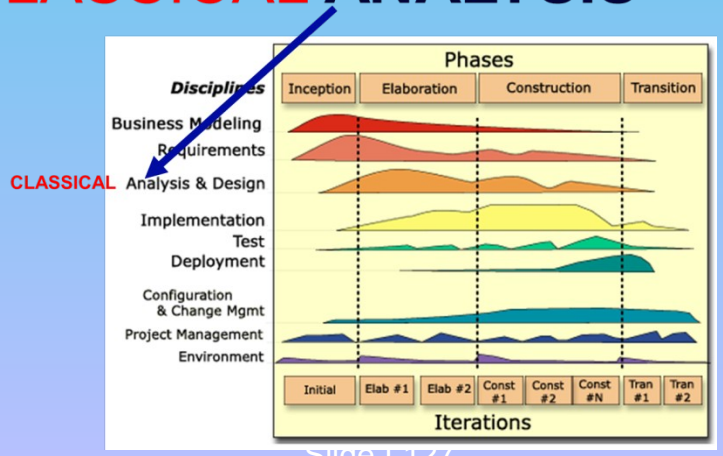


CLASSICAL ANALYSIS

END

CLASSICAL ANALYSIS - WHAT

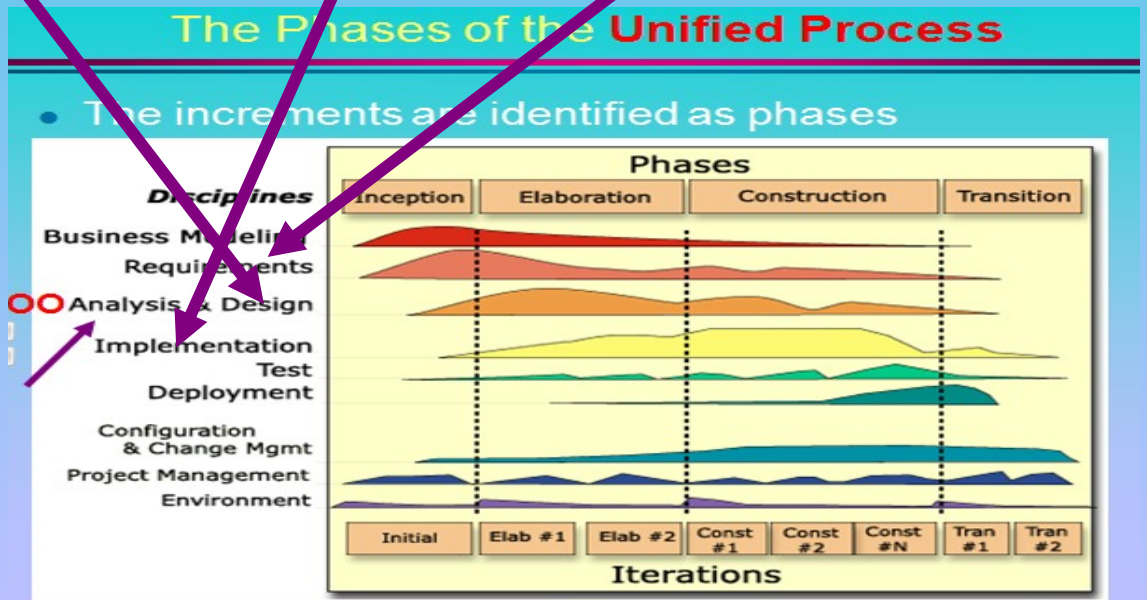
CLASSICAL ANALYSIS



The Object Oriented Analysis Workflow

The **Object Oriented Analysis Workflow** has two aims:

- Obtain a *deeper understanding of the* ***Requirements***
- Describe ***Requirements*** in a way that will result in a **Maintainable OO Design** and **OO Implementation Workflows**



The OO Analysis Workflow (UML)

Perform the following steps

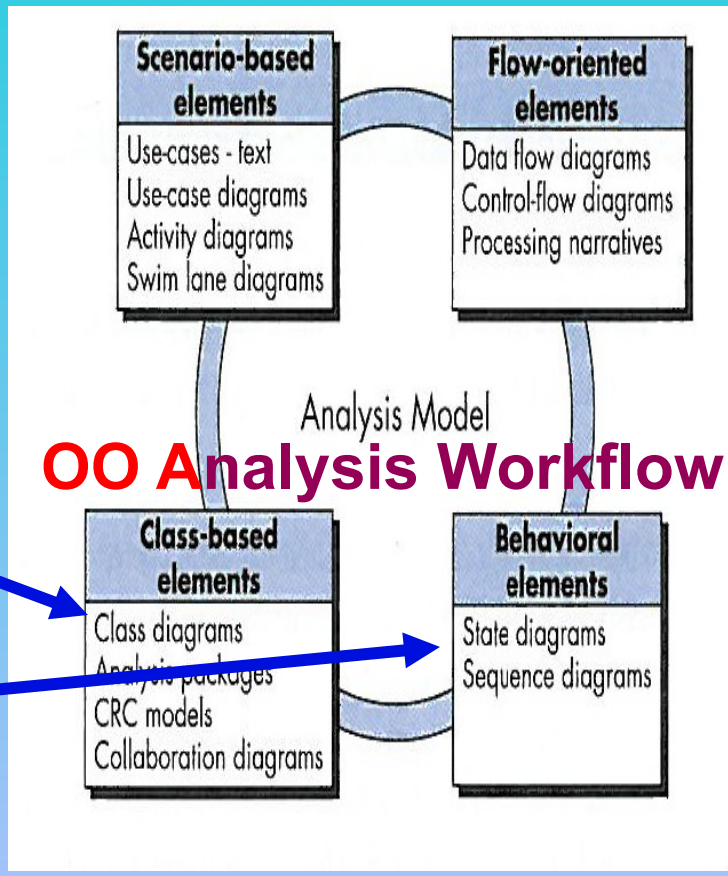
Incrementally and Iteratively (Agile)

Class Modeling

- Determine the **Entity Classes** and their **attributes**
- Determine the **interrelationships and interactions between the Entity Classes (methods)**
- Present this information in the form of a **UML MVC OOA *Class Diagram***

Dynamic Modeling

- Determine the **operations performed** by or to each **Entity Class**
- Present this information in form of a **UML**

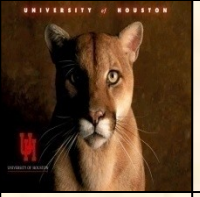


OO Analysis Workflow

Object Oriented Analysis: The Elevator Problem Case Study

A product is to be installed to control n elevators in a building with m floors. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the elevator
2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction
3. If an elevator has no requests, it remains at its current floor with its doors closed



UML Class Diagram Modeling Steps

OO Analysis Workflow

Step 1: Identify Classes

Class

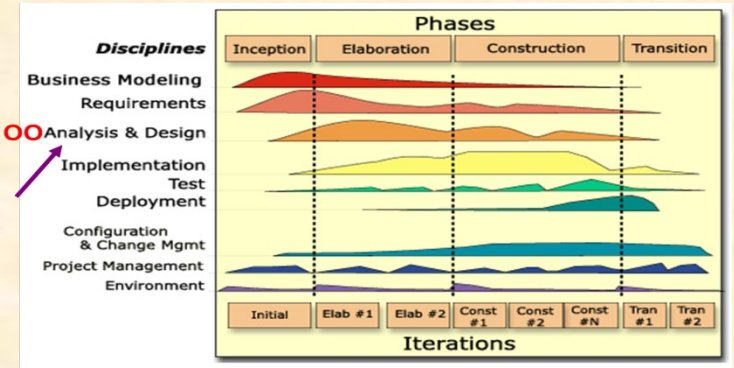
Book

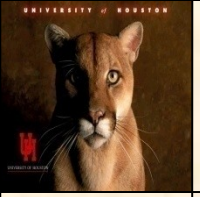
Step 2: Identify Attributes

Class: # - Attribute

Book: 1 - id

Step 3: Draw UML Classes Diagram

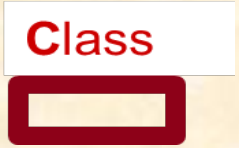


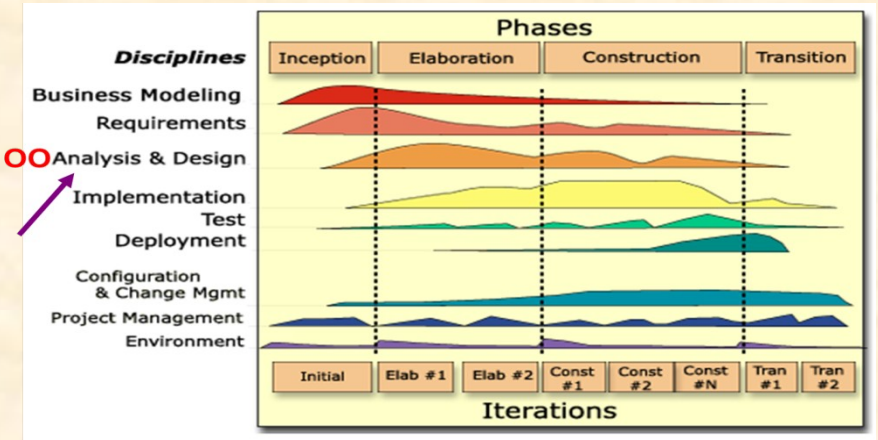


UML Class Diagram Modeling Steps

OO Analysis Workflow

Step 1: Identify Classes





UML Class Diagram: The Elevator Problem Case Study

Description of the SOFTWARE PRODUCT in single paragraph:

Button Elevator Button Floor Button

Buttons in elevators and on the floors control the movement of

Elevator

n elevators in a building with m floors. Buttons illuminate

when pressed to request the elevator to stop at a specific floor;

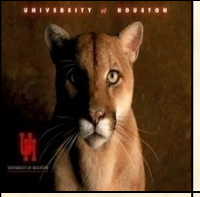
the illumination is canceled when the request has been satisfied.

When an elevator has no requests, it remains at its current floor

with its doors closed

UML Entity Class Modeling

Can you determine the **Entity Classes** and draw the class hierarchy. Fill in as many **attributes** for each **Class**.

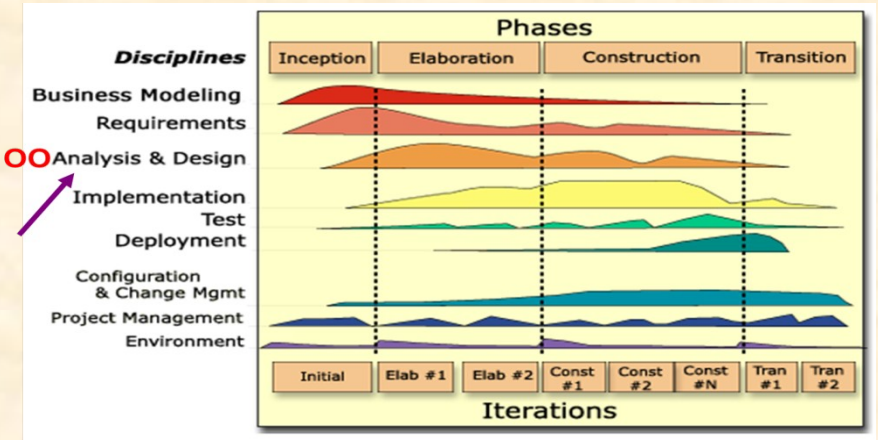


UML Class Diagram Modeling Steps

OO Analysis Workflow

Step 2: Identify Attributes

Class: # - Attribute



UML Class Diagram: The Elevator Problem Case Study

Description of the SOFTWARE PRODUCT in single paragraph:

Button Elevator Button Floor Button

Buttons in elevators and on the floors control the movement of

Elevator Button: 1 - illuminated

n elevators in a building with m floors. Buttons illuminate

when pressed to request the elevator to stop at a specific floor;

the illumination is canceled when the request has been satisfied.

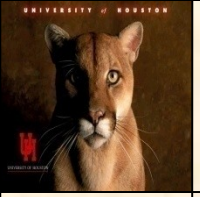
When an elevator has no requests, it remains at its current floor

with its doors closed

Elevator: 1 – doorsOpen

UML Entity Class Modeling

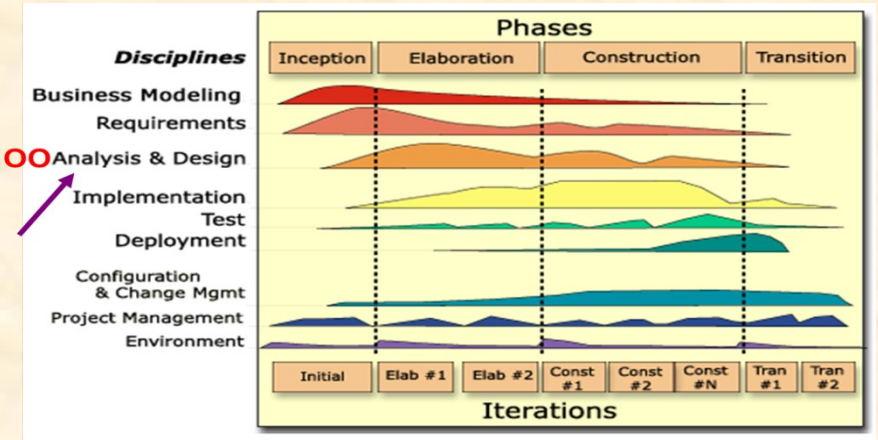
Can you determine the **Entity Classes** and draw the class hierarchy. Fill in as many **attributes** for each **Class**.



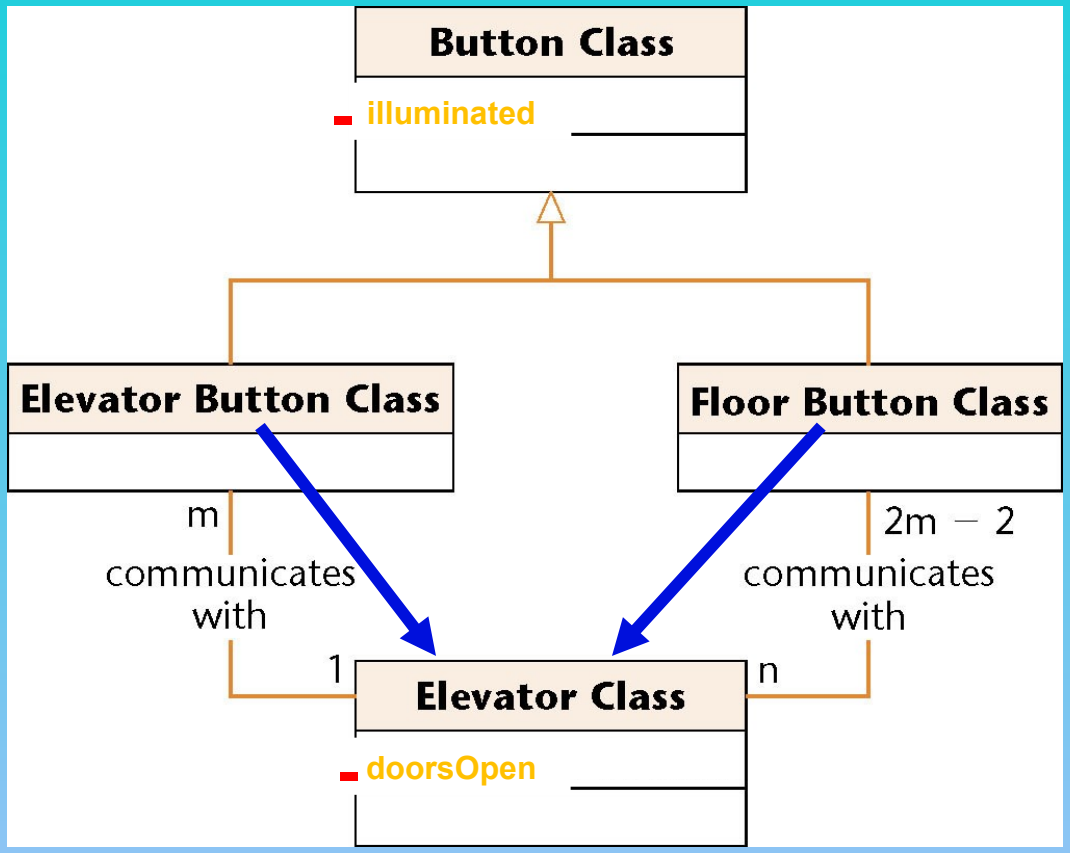
UML Class Diagram Modeling Steps

OO Analysis Workflow

Step 3: Draw UML Classes Diagram



First Iteration of UML Class Diagram



Model

- Problem
  - **Buttons** do not communicate directly with **Elevators**
  - We need an additional **Class**: **Elevator Controller Class**

Controller

Second Iteration of UML Class Diagram

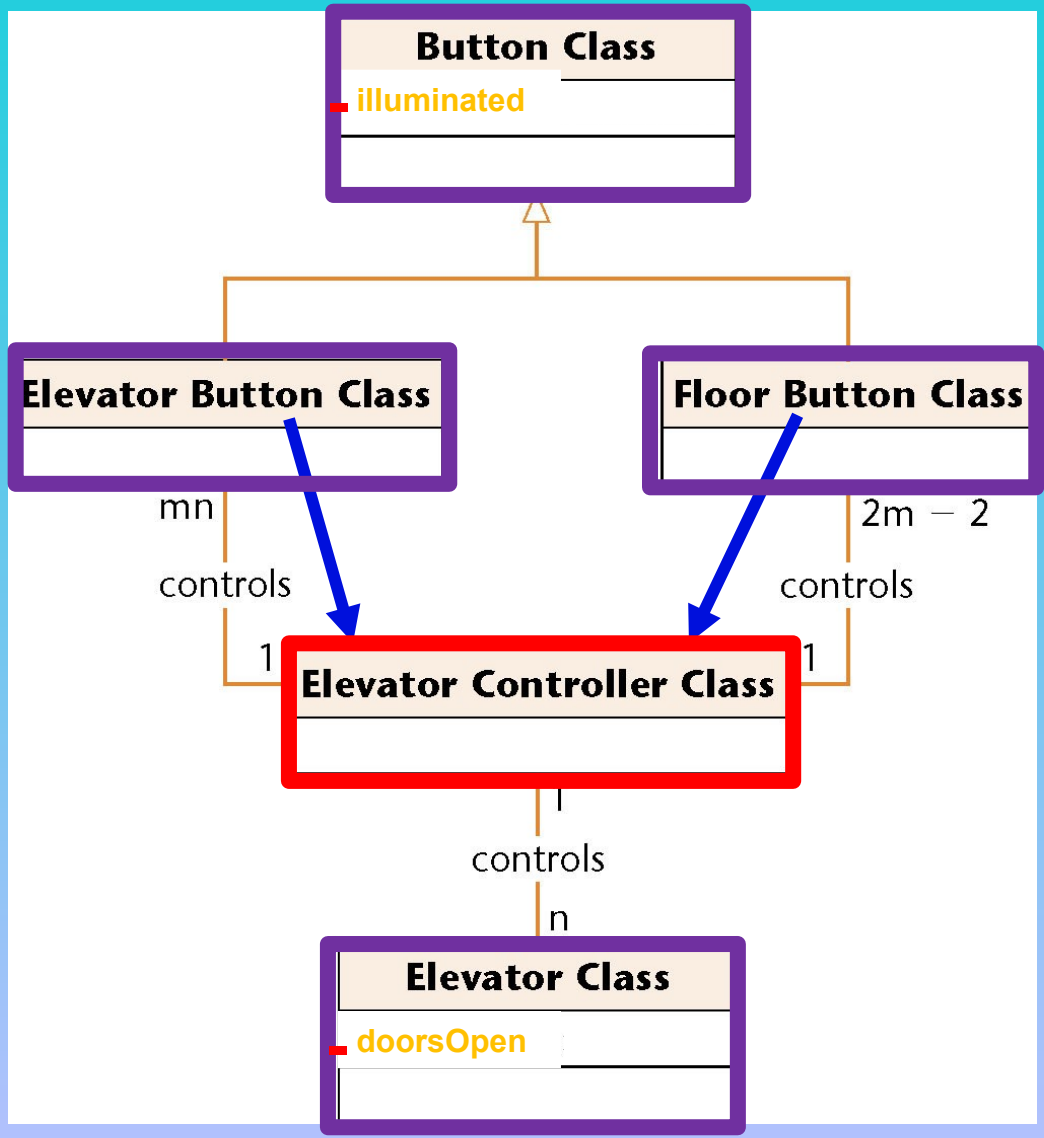


Figure 12.6

Dynamic Modeling: The Elevator Problem Case Study

State, event, and predicate are distributed over the **UML Statechart**

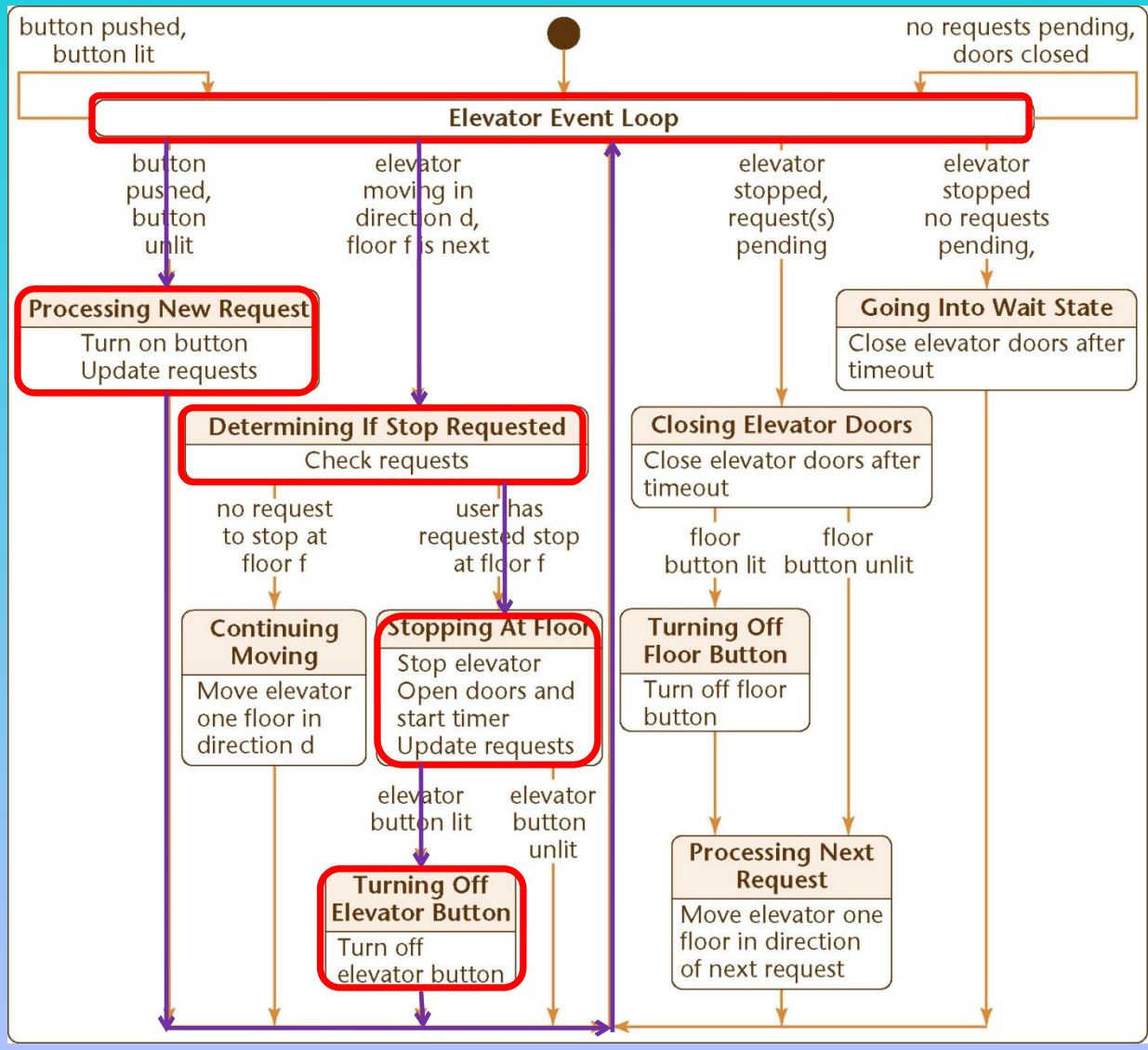
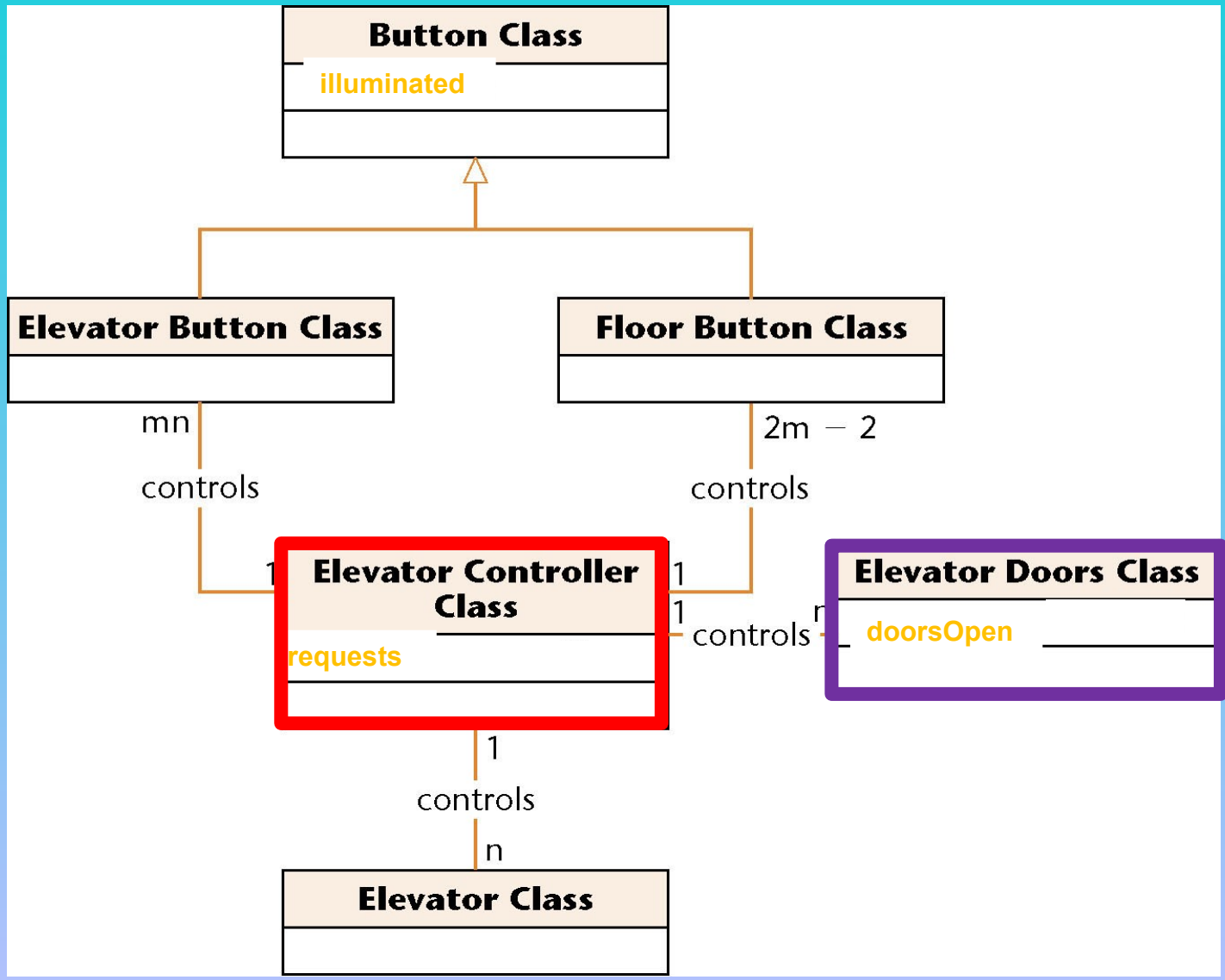


Figure 12.7

Third Iteration of UML Class Diagram



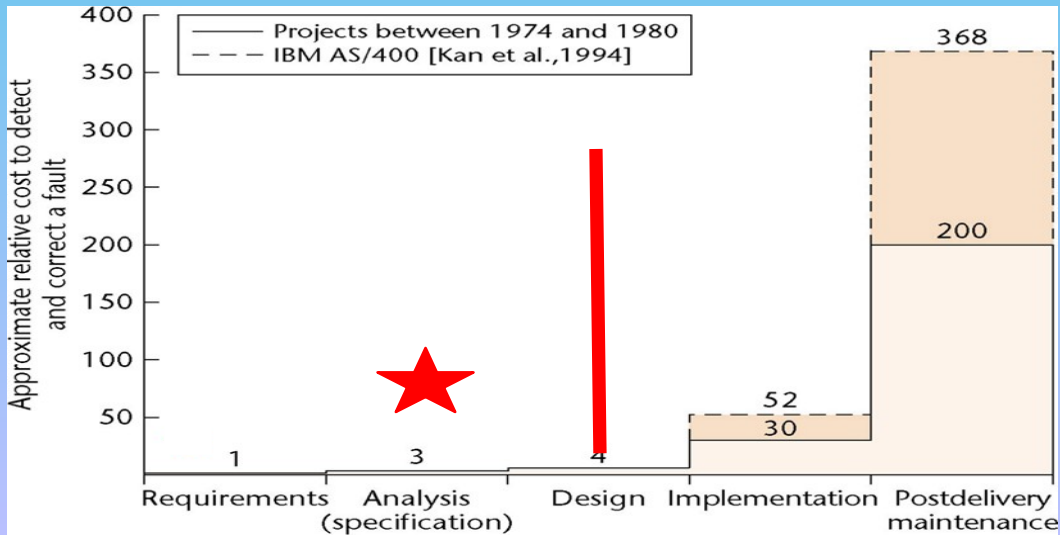
OOA: Elevator Problem

The **Object Oriented Analysis** is now fine

We should rather say:

- The **Object Oriented Analysis** is fine ***for now***

We may need to return to the **Object Oriented Analysis Workflow** during the **Object Oriented Design Workflow**



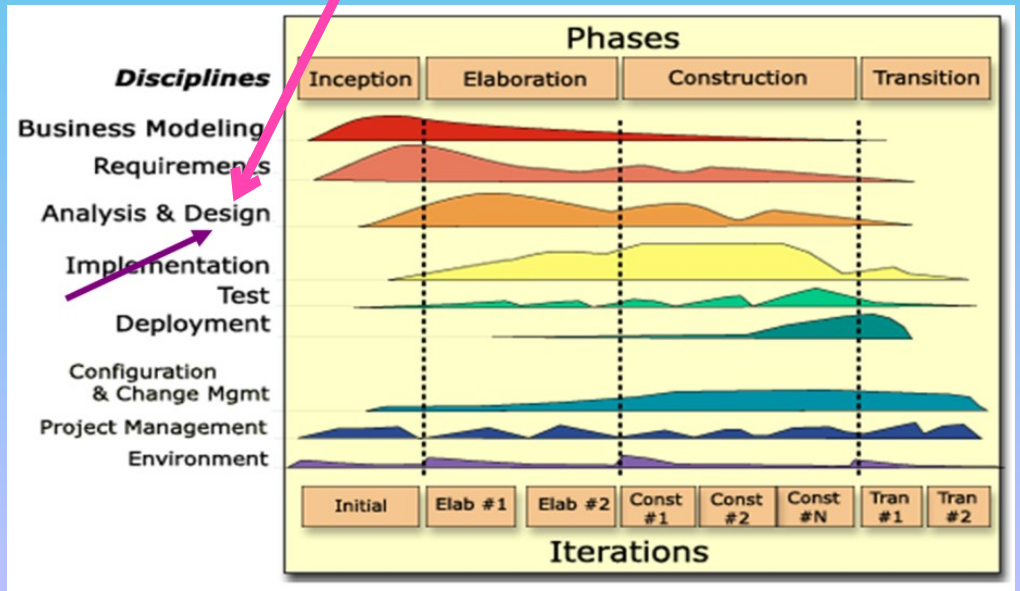
Challenges of the Object Oriented Analysis Workflow

Do **not** cross the boundary into **Object Oriented Design**

(OO Design Workflow)

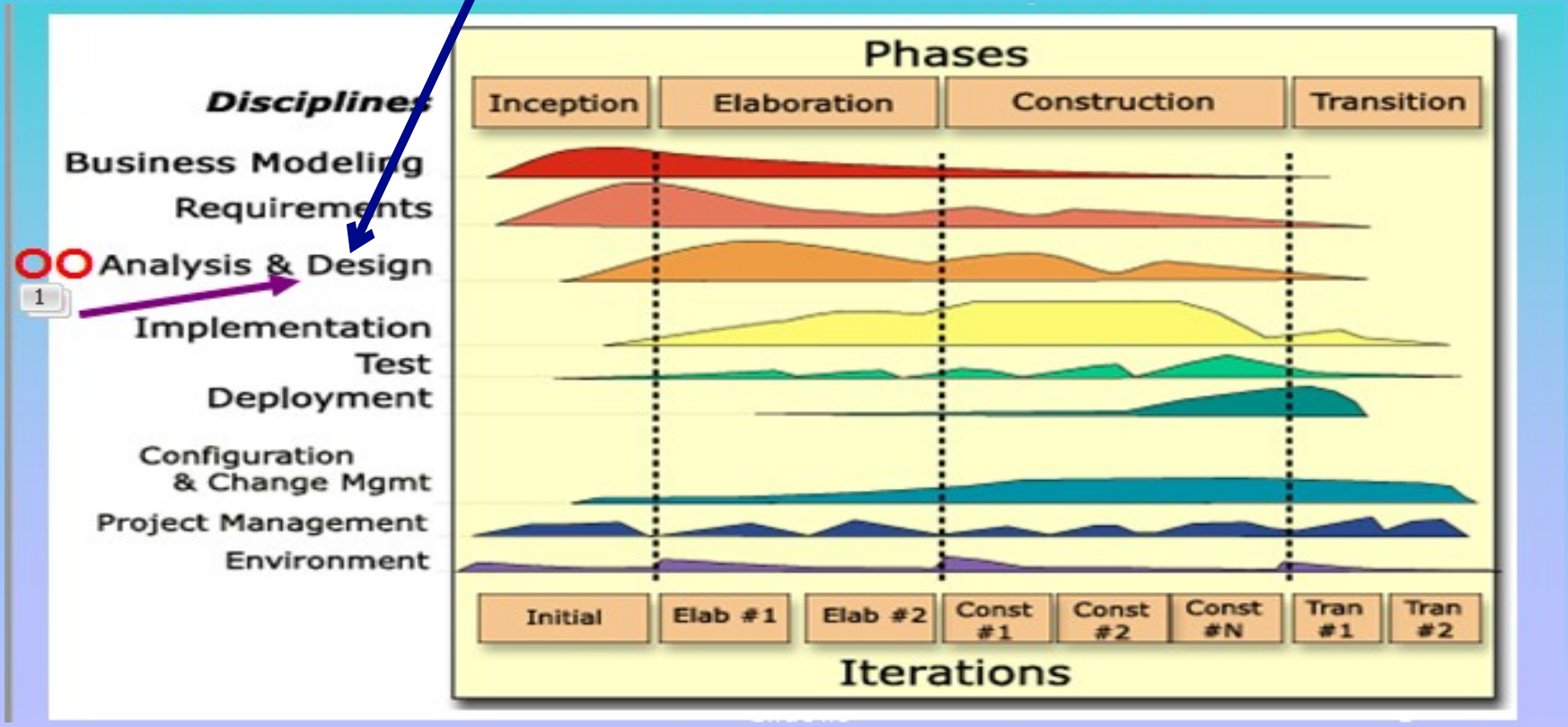
Do **not allocate methods** to **Classes** ***yet!***

- Reallocating **methods** to **Classes** during stepwise refinement **is wasted effort**



End **OOA – WHAT** Workflow

- Next **OO Design – HOW Workflow**

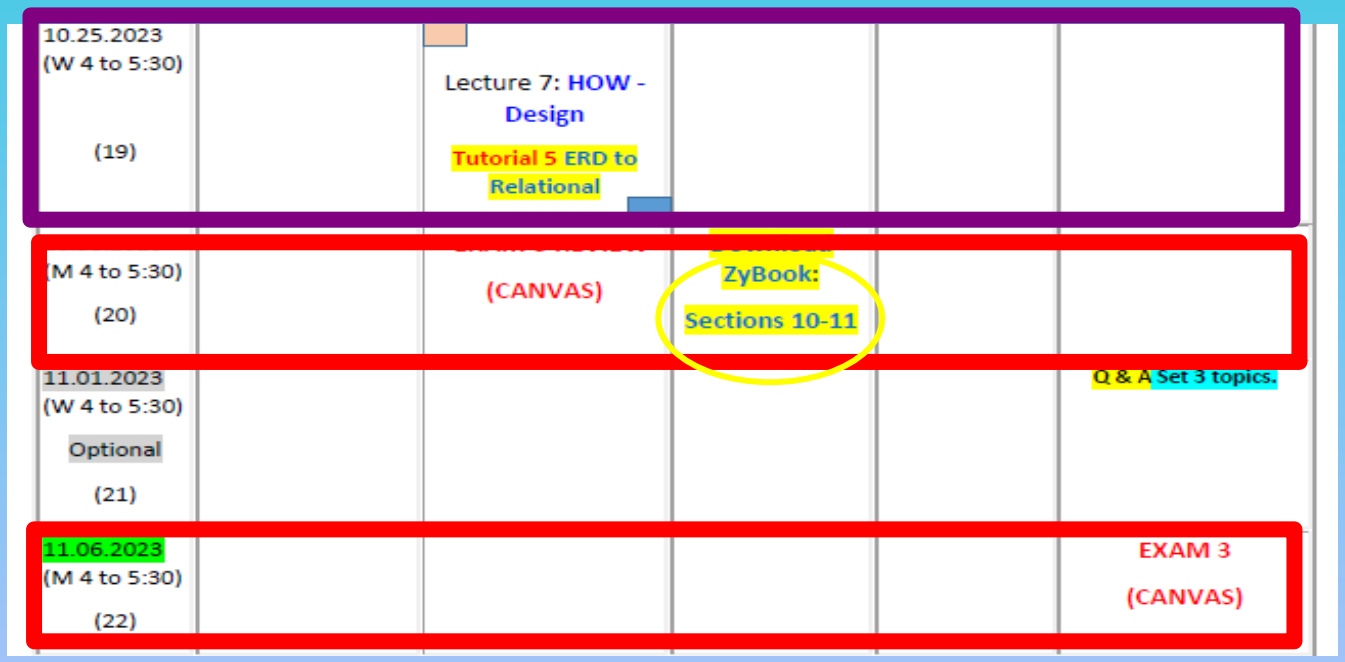


From 4:10 to 5:00 – 50 minutes.

END

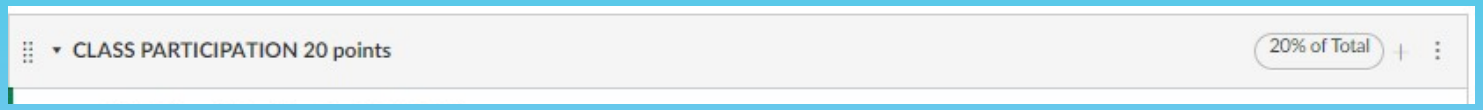
Analysis Workflow

NEXT

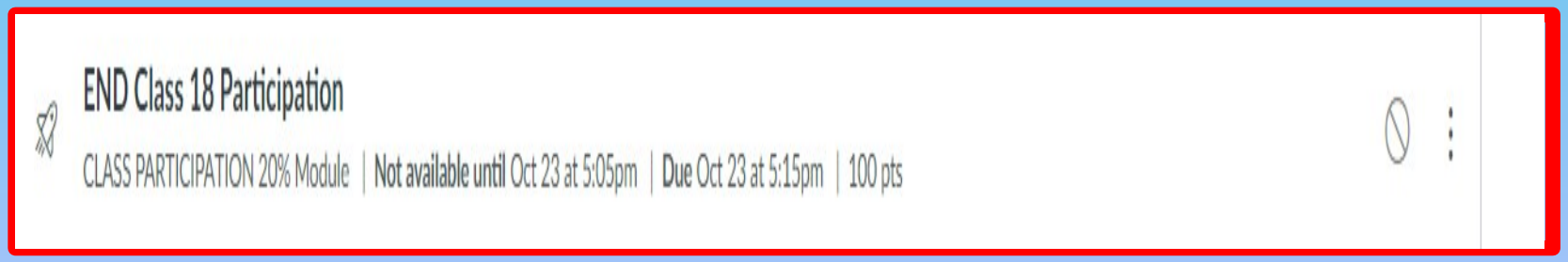

10.25.2023
(W 4 to 5:30)
(19)
Lecture 7: HOW - Design
Tutorial 5 ERD to Relational
(M 4 to 5:30)
(20)
(CANVAS)
ZyBook:
Sections 10-11
11.01.2023
(W 4 to 5:30)
Optional
(21)
Q & A Set 3 topics.
11.06.2023
(M 4 to 5:30)
(22)
EXAM 3
(CANVAS)

From 5:05 to 5:15 – 10 minutes.

10.23.2023 (M 4 to 5:30) (18)		Lecture 6: WHAT - Analysis		Analysis Papers Summary (1 Page) CANVAS Assignment	



PASSWORD: I AM IN TEAMS



VH– Publish.

At 5:15.

End Class 18

VH, Download Attendance Report
Rename it:
10.23.2023 Attendance Report FINAL

VH, upload Class 18 to CANVAS