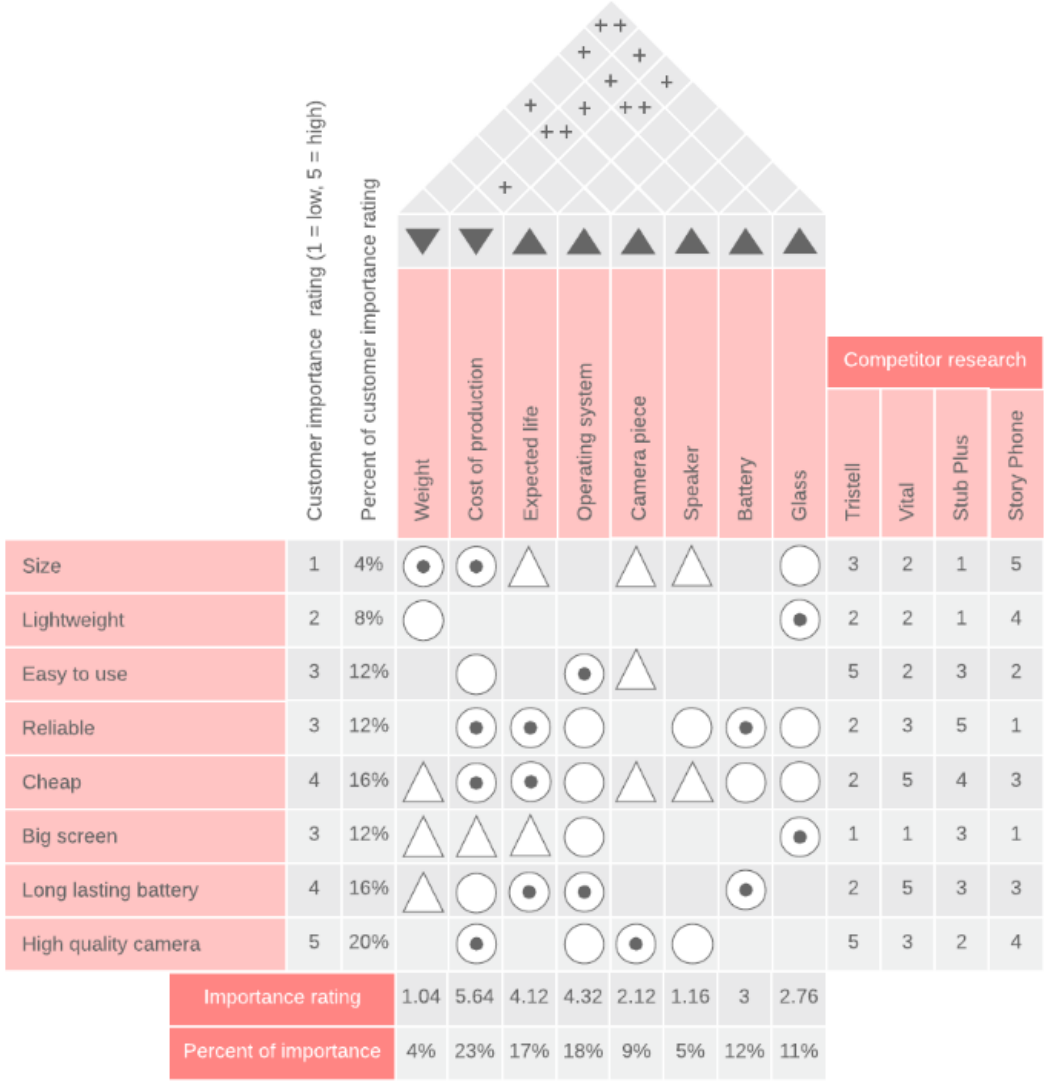
Lecture Recap

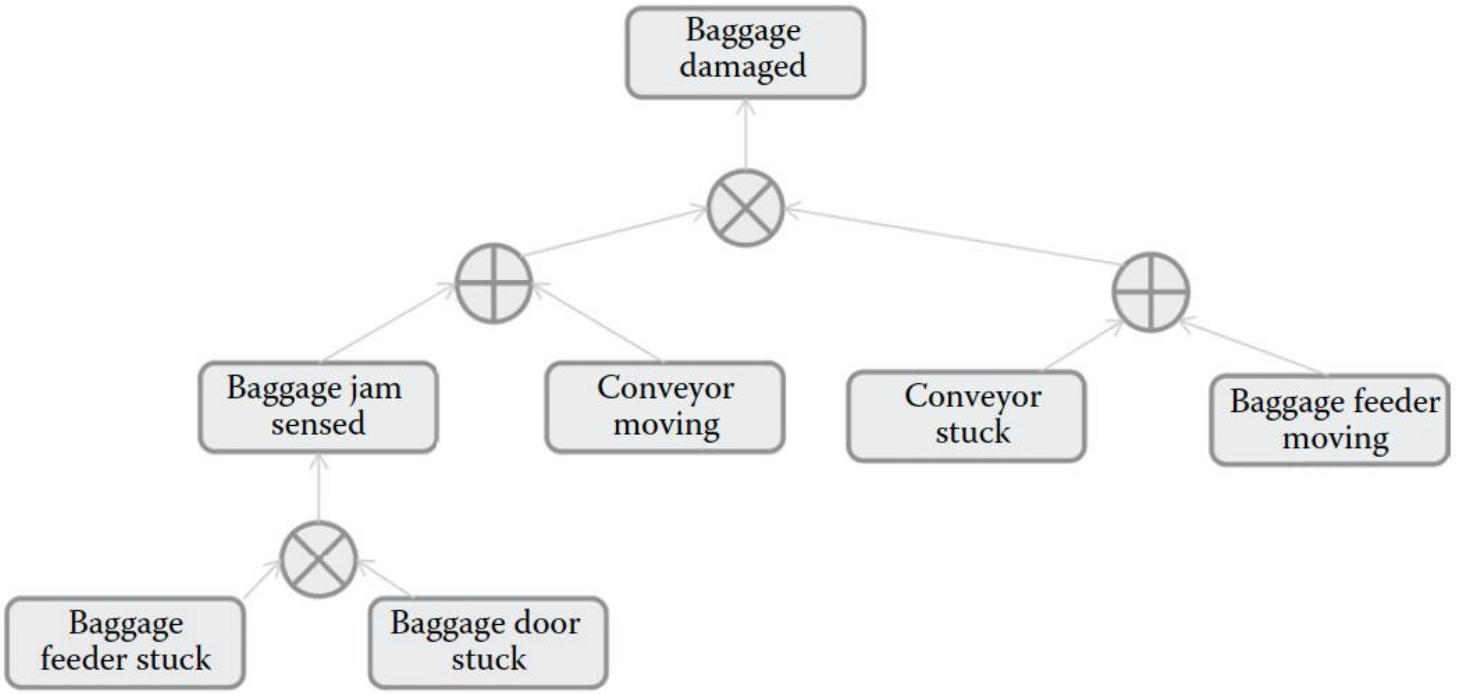


	Customer importance rating (1 = low, 5 = high)	Percent of customer importance rating	Weight	Cost of production	Expected life	Operating system	Camera piece	Speaker	Battery	Glass	Competitor research: Tristell	Vital	Stub Plus	Story Phone
Size	1	4%	●	●	△		△	△		○	3	2	1	5
Lightweight	2	8%	○							●	2	2	1	4
Easy to use	3	12%		○		●	△				5	2	3	2
Reliable	3	12%		●	●	○		○	●	○	2	3	5	1
Cheap	4	16%	△	●	●	○	△	△	○	○	2	5	4	3
Big screen	3	12%	△	△	△	○				●	1	1	3	1
Long lasting battery	4	16%	△	○	●	●			●		2	5	3	3
High quality camera	5	20%		●		○	●	○			5	3	2	4
Importance rating			1.04	5.64	4.12	4.32	2.12	1.16	3	2.76				
Percent of importance			4%	23%	17%	18%	9%	5%	12%	11%				

Relationship matrix		
●	Strong	9
○	Medium	3
△	Weak	1
	No assignment	0

Correlation matrix	
+ +	Strong positive
+	Positive
–	Negative
– –	Strong negative
	Not correlated

Source: https://www.lucidchart.com/blog/qfd-house-of-quality



- The fault tree leads us to write the following raw requirements:
  - If a baggage jam is sensed, then the conveyor shall not move.
  - If the baggage feeder is stuck, then the conveyor shall not move.
  - If the baggage door is stuck, then the conveyor shall not move.
  - If the conveyor is stuck, then the baggage feeder should not move.

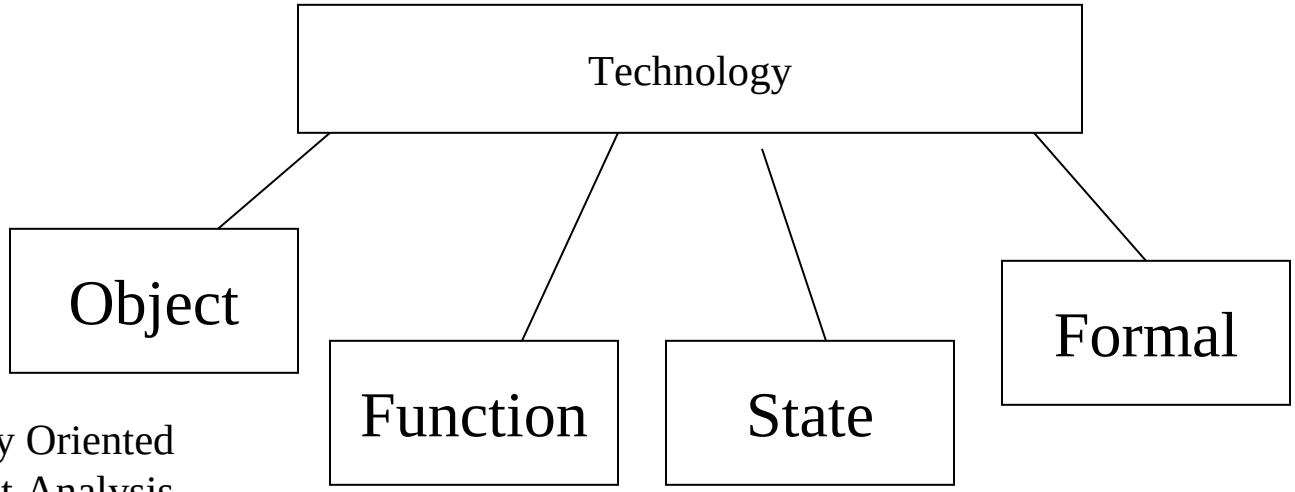
U O W

Software Requirements, Specifications and Formal Methods

Dr. Shixun Huang

UNIVERSITY OF WOLLONGONG AUSTRALIA

Methods for Requirements Engineering



- Ontology Oriented Requirement Analysis
- Concurrent Design Approach for Real-time System

- **Dataflow diagrams**
- **Sequence diagrams**
- Decision table
- Process specification
- Specification and Description Graphs
- Structure Analysis and Design Technique diagrams

- **Finite State Machine**
- **Petri nets**
- State chart

- **Petri nets**
- Constraint Satisfaction Problem
- Vienna Development Method
- **Z notation**

Requirement Modeling Strategies

- One approach, called *structured analysis,* considers data and the processes that transform the data as separate entities.
  - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- Another approach, called *object-oriented analysis,* focuses on
  - The definition of classes and
  - Relationship between them.

Functional (Structural) Analysis (Data-flow modeling)

Data-flow approaches use data-flow diagrams (DFD) to graphically represent the external entities, processes, data-flow, and data stores.

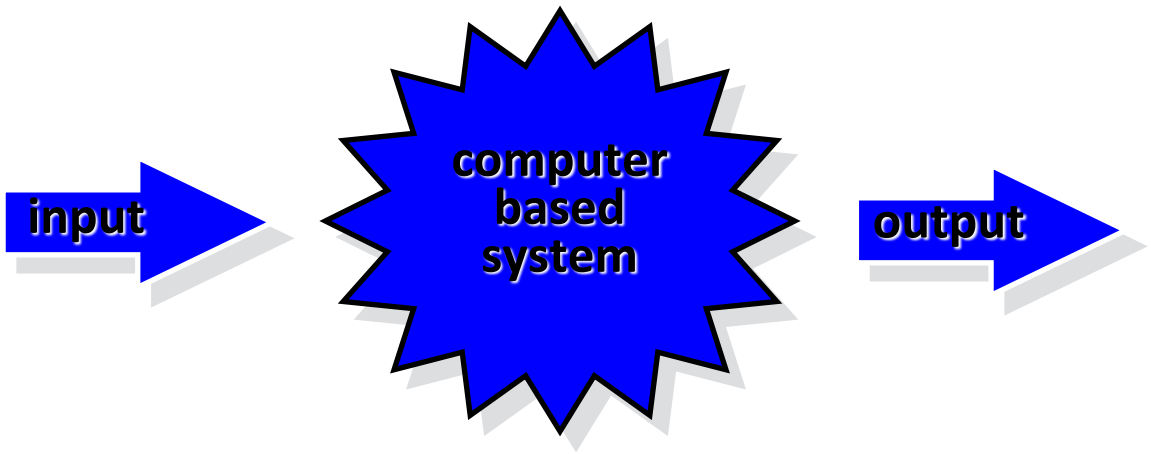
Data Flow Diagram

A diagram that specifies the processes (also referred as bubbles, transforms, transitions, activities, operations) and flow of data between them is called a Data Flow Diagram (DFD).

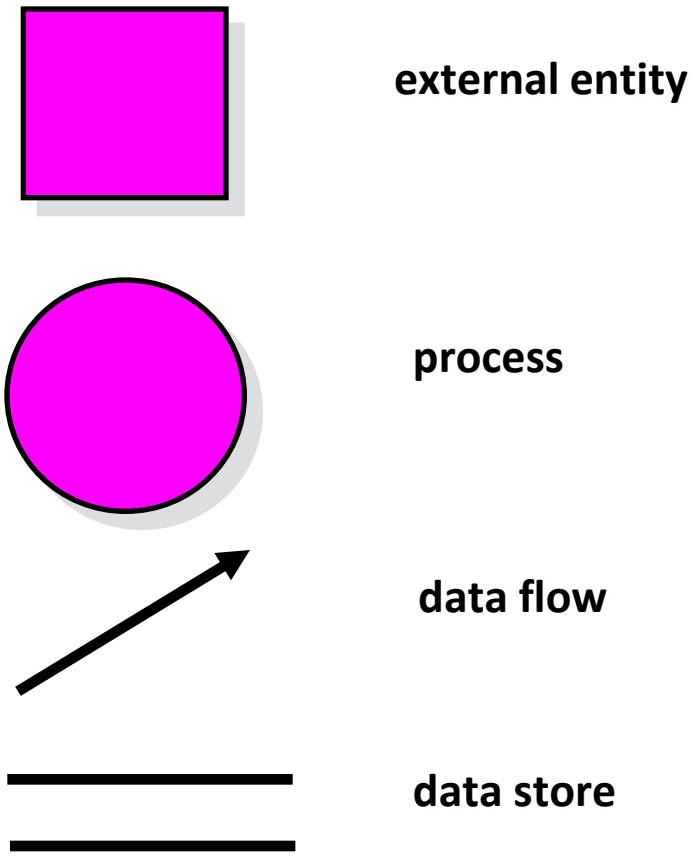
A DFD exhibits possible forms of information flow in a system, storage locations for data, and transformations of data as it flows through a system.

The Flow Model

Every computer-based system is an information transform ....



Flow Modeling Notation



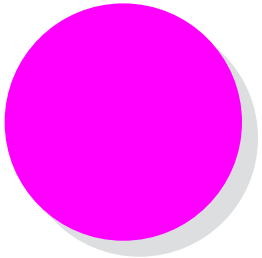
External Entity

A producer or consumer of data

Examples: a person, a device, a sensor

Data must always originate from somewhere and must always be sent to somewhere

Process

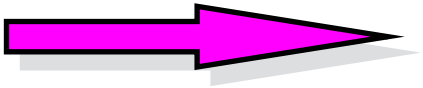


A data transformer (changes input to output)

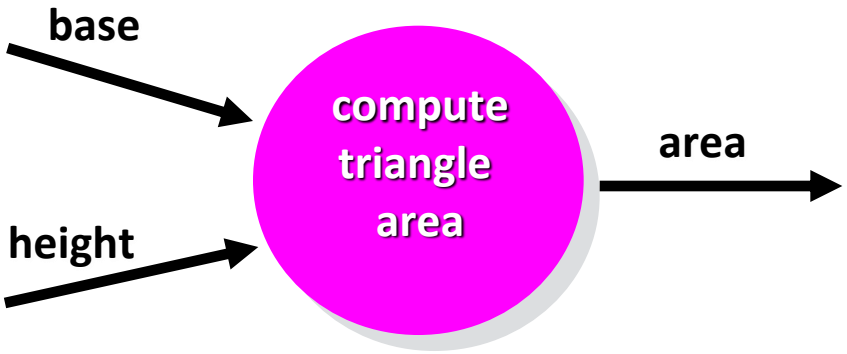
Examples: *compute taxes, determine area, format report, display graph*

Data must always be processed in some way to achieve system function

Data Flow

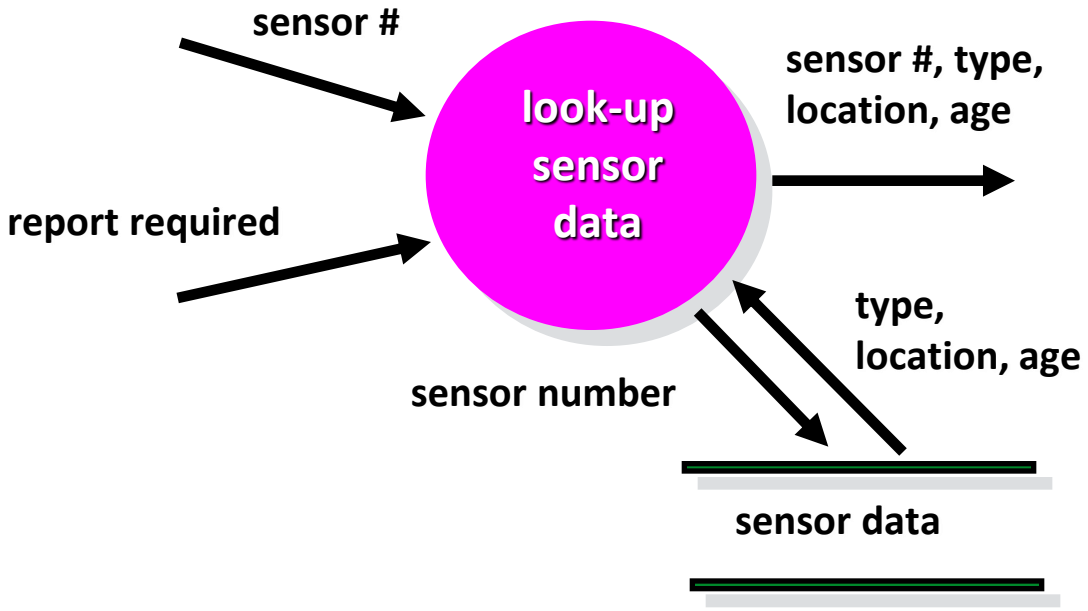


Data flows through a system, beginning as input and transformed into output.



Data Stores

Data is often stored for later use.



Data Flow Diagram: Guidelines

- all icons must be labeled with meaningful names
- the DFD evolves through a number of levels of detail
- always begin with a context level diagram (also called level 0)
- always show external entities at level 0
- always label data flow arrows
- do not represent procedural logic

Constructing a DFD—Step I

- review user scenarios and/or the data model to isolate data objects and use a grammatical parse to determine “operations”
- determine external entities (producers and consumers of data)
- create a level 0 DFD

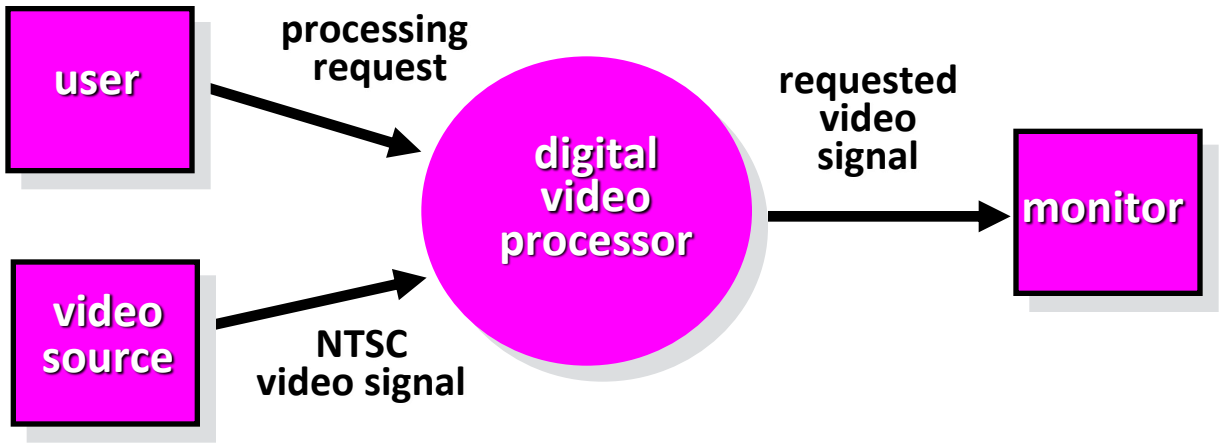
Level 0: Context level data-flow diagram

- The analyst represents the `target` system at its context level. It is also called parent-DFD.

- A context level DFD is a high-level diagram (omitting all but essential details) consisting of **a single bubble (process)**.

  The aim of the context level data-flow diagram is to view the system as an unexplored `black box', and to direct the focus of the analysis on the type of the data-flows that **enter the system from the source and those that travel from the system to their destination.**

Level 0 DFD Example



Constructing a DFD— Step II

- Write a narrative describing the transform
- Parse to determine next level transforms
- “Balance” the flow to maintain data flow continuity
- Develop a level 1 DFD
- Use a 1:5 (approx.) expansion ratio

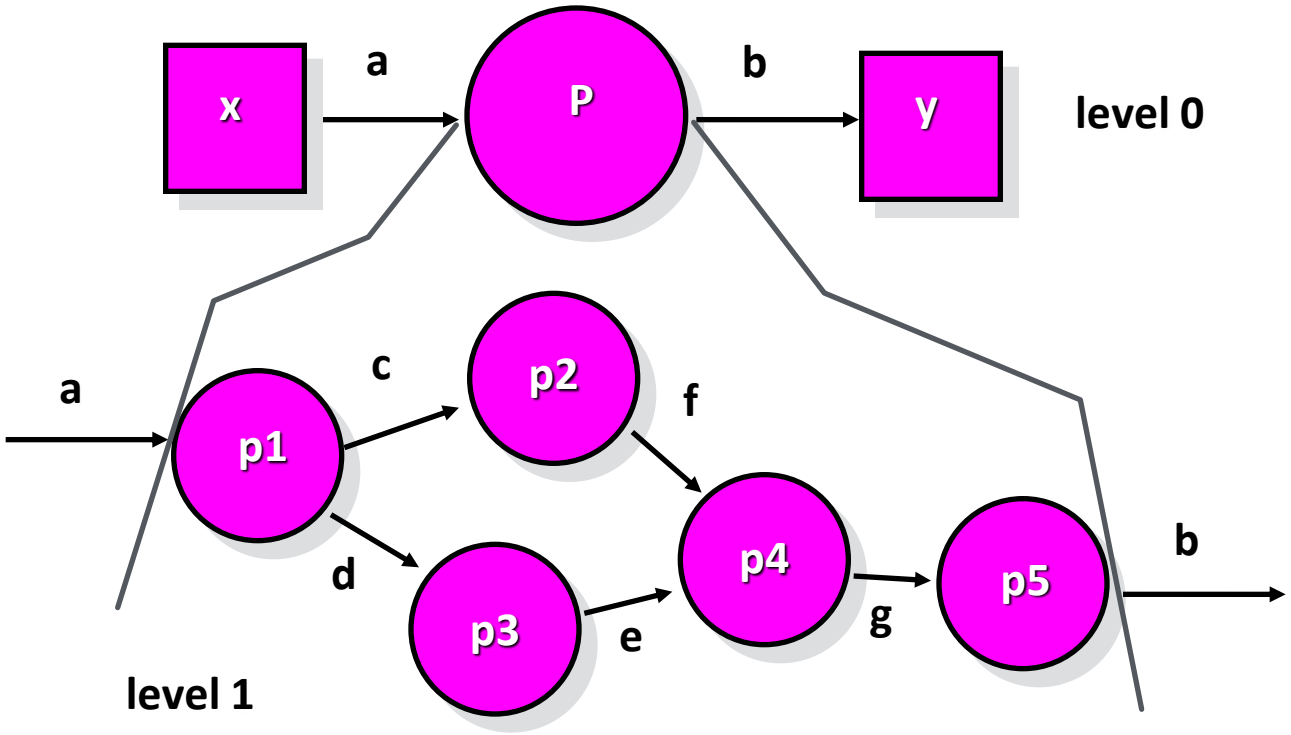
Level 1 of the data-flow diagram

Level 1 of the data-flow diagram is constructed by decomposing the top-level system bubble into **sub-systems**.

The level 1 is also called child-DFD. Each bubble represents a potential sub-system through further decomposition.

The decomposition can be repeated for bubbles inside the child-DFD until dataflow for a software process is understood.

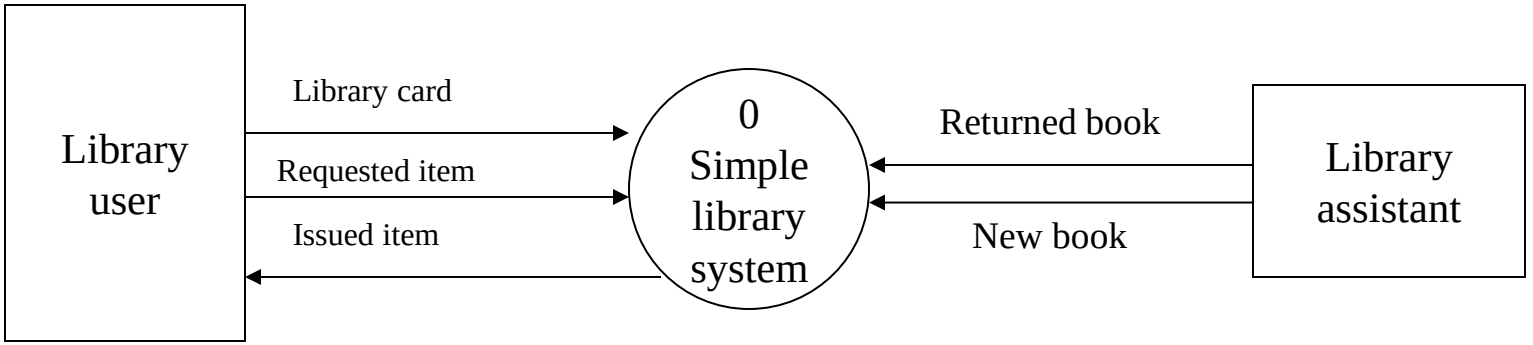
The Data Flow Hierarchy



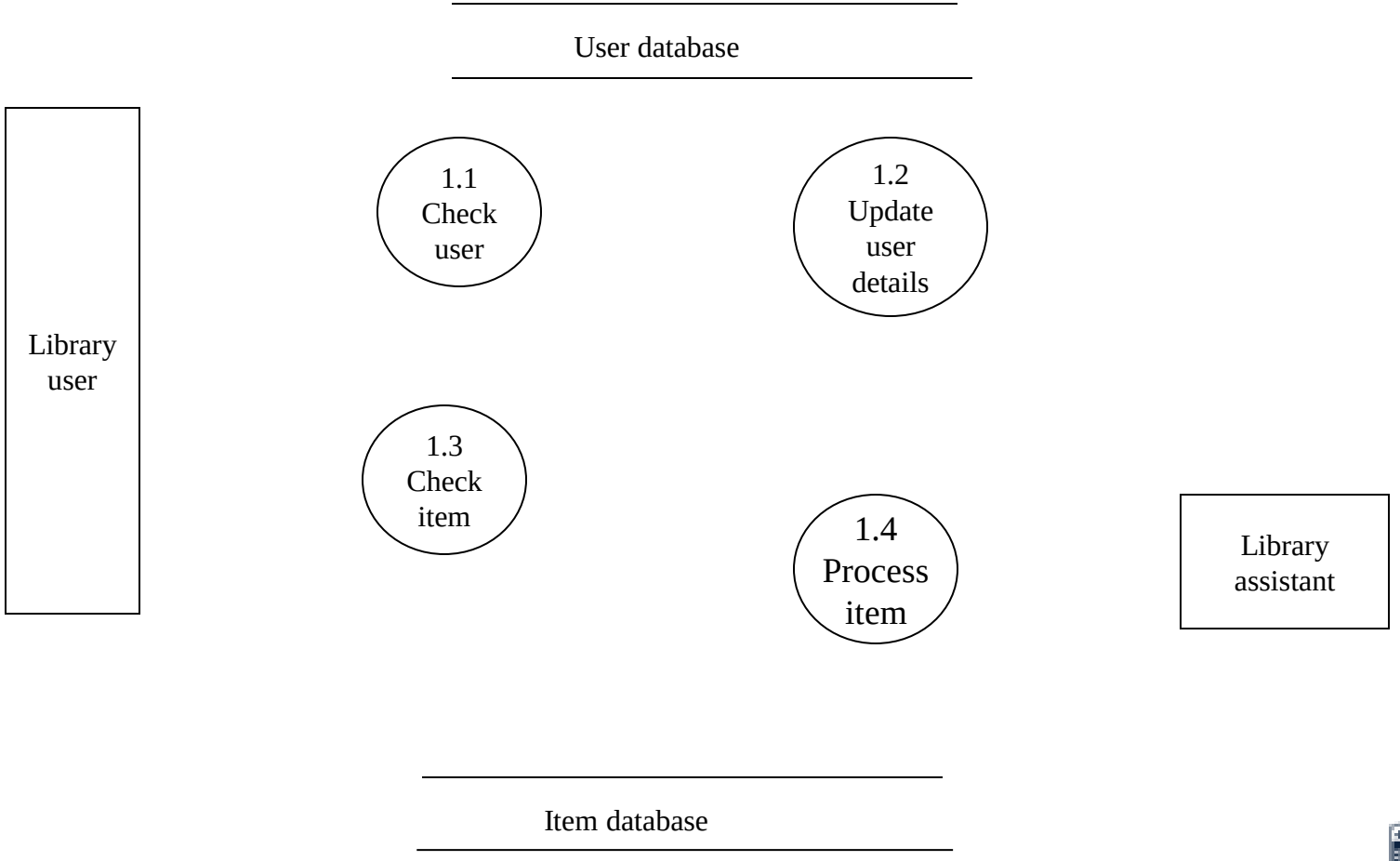
Flow Modelling Notes

- Each bubble is refined until it does just one thing
- The expansion ratio decreases as the number of levels increase
- Most systems require between 3 and 7 levels for an adequate flow model
- A single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)

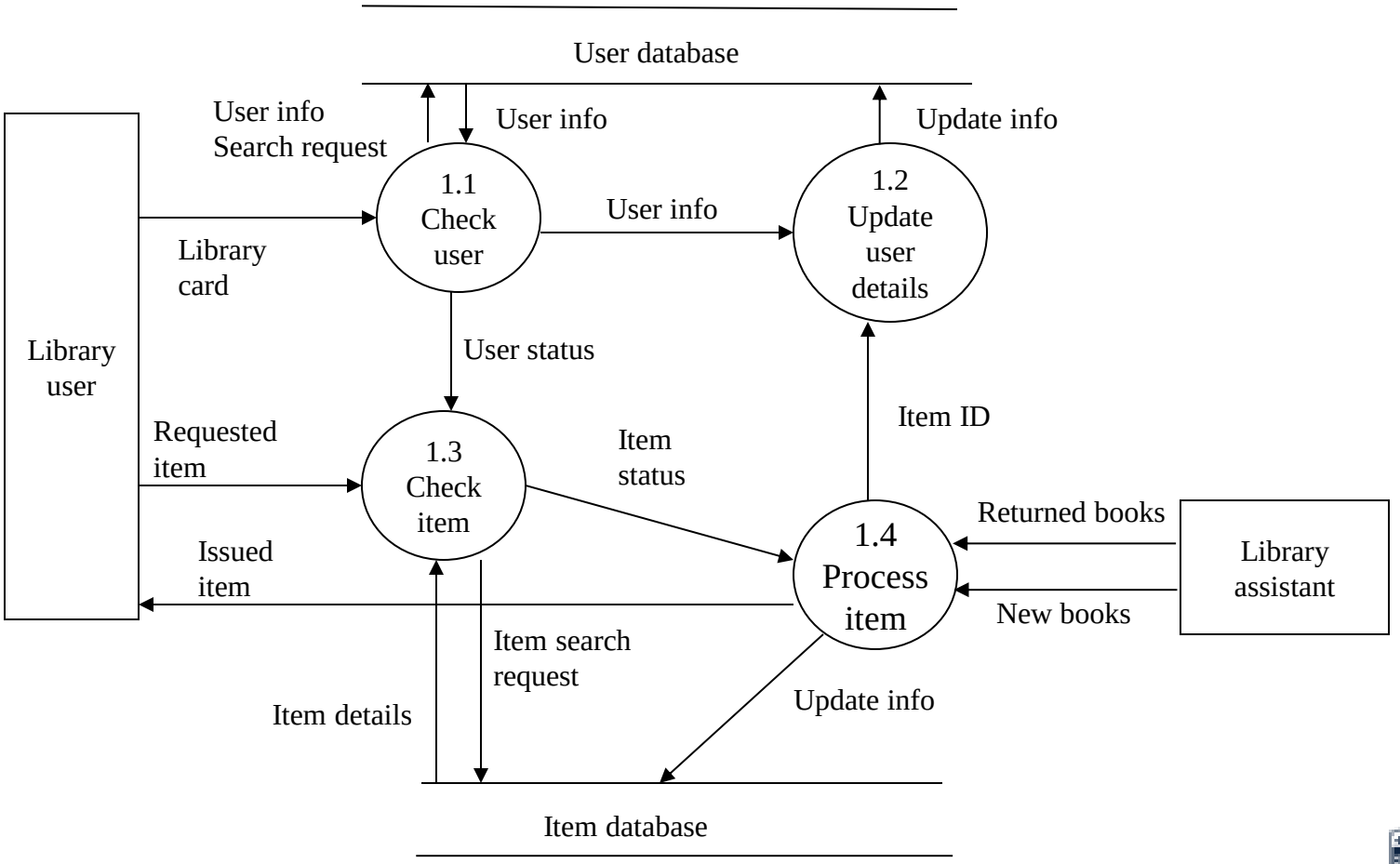
Example: the context-level (level 0) data-flow diagram for simple library system



Level 1 of the Data-flow diagram for the simple library system



Level 1 of the Data-flow diagram for the simple library system



Example: a lemonade system (Level 0)

Create a context level diagram identifying the sources and sinks (users).

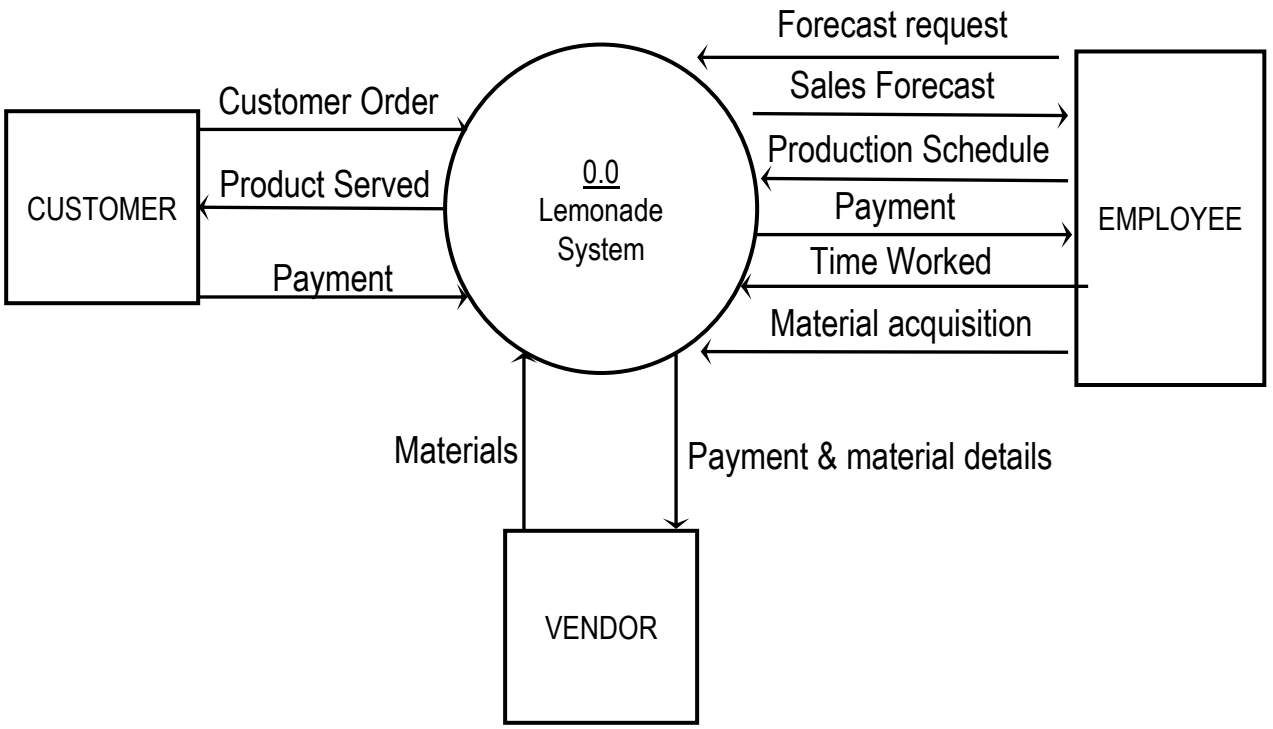
Customer Order
Collect Payment
Sales forecast

Produce Product
Store Product
Serve Product

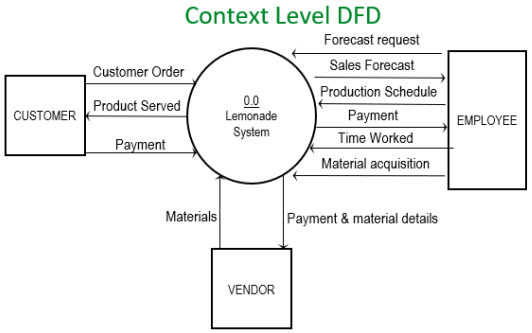
Order Raw Materials
Pay for Raw Materials

Pay for Labor

Context Level DFD



Example: a lemonade system (Level 1)



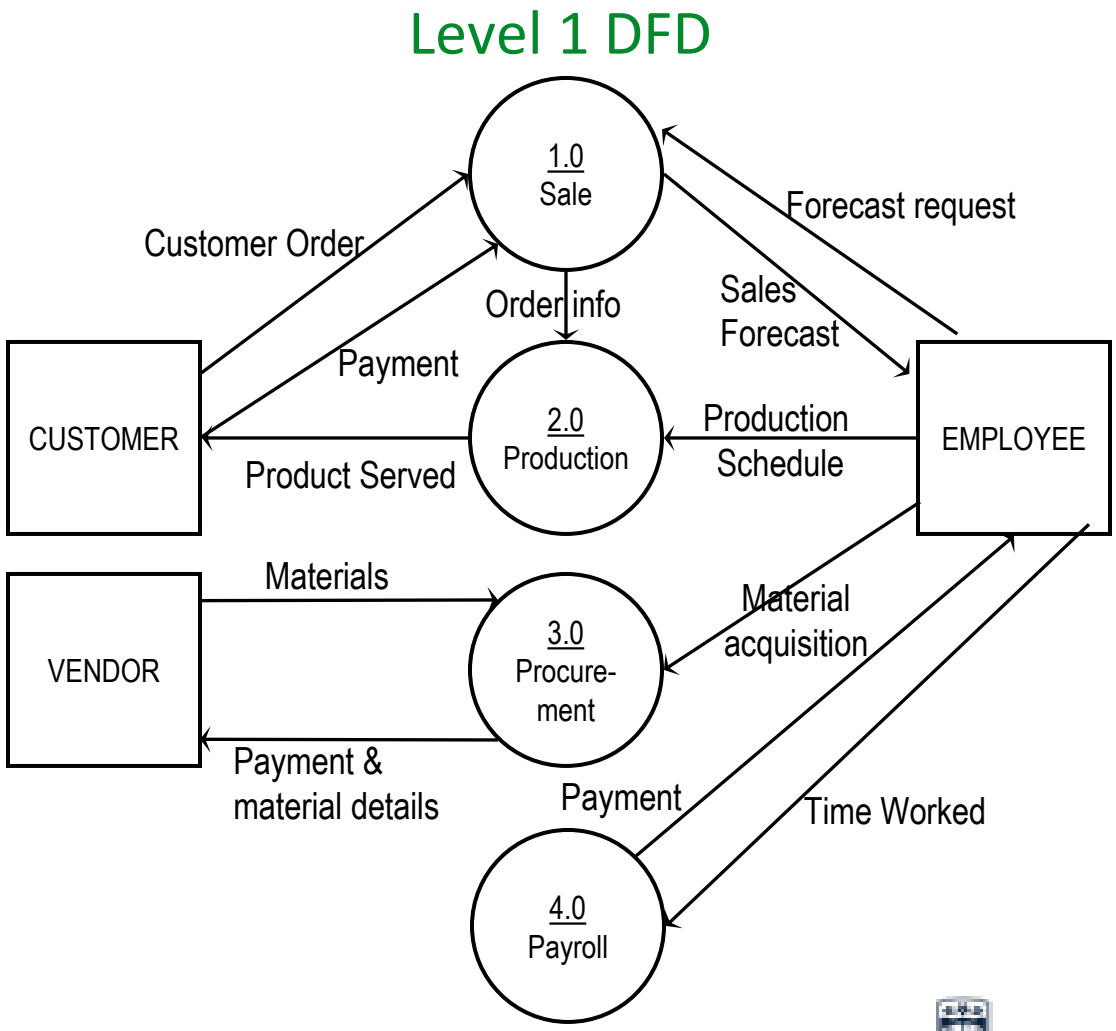
Create a level 1 diagram identifying the logical subsystems that may exist.

Customer Order
Collect Payment
Sales forecast

Produce Product
Store Product
Serve Product

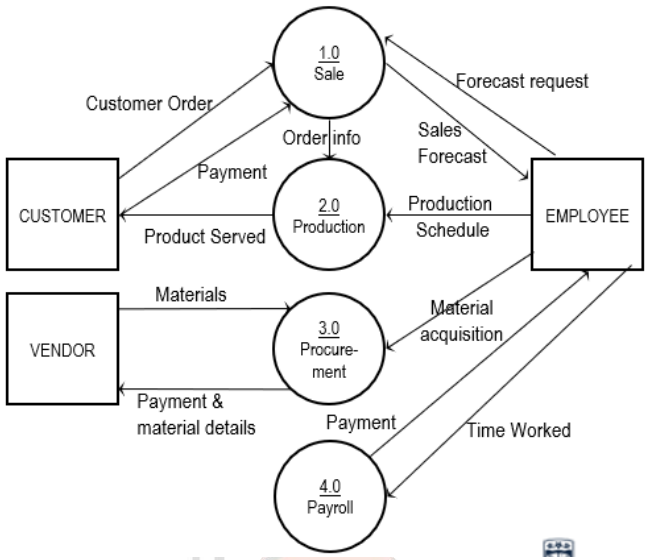
Order Raw Materials
Pay for Raw Materials

Pay for Labor



Example: a lemonade system (Level 2)

Level 1 DFD



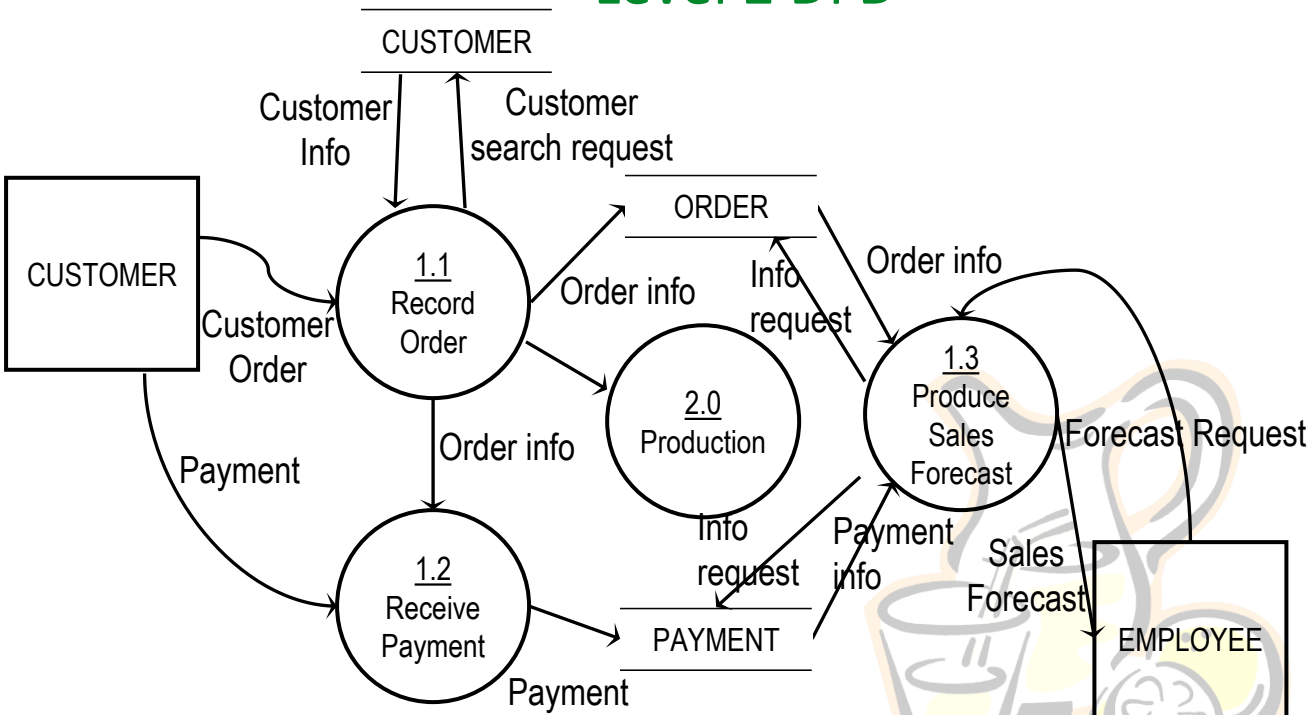
Customer Order
Collect Payment
Sales forecast

Produce Product
Store Product
Serve Product

Order Raw Materials
Pay for Raw Materials

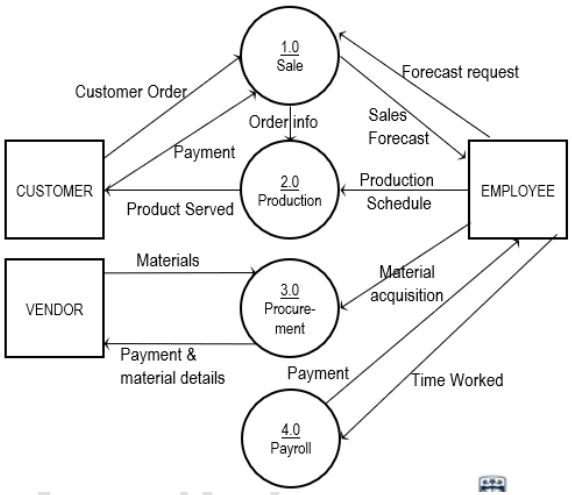
Pay for Labor

Level 2 DFD

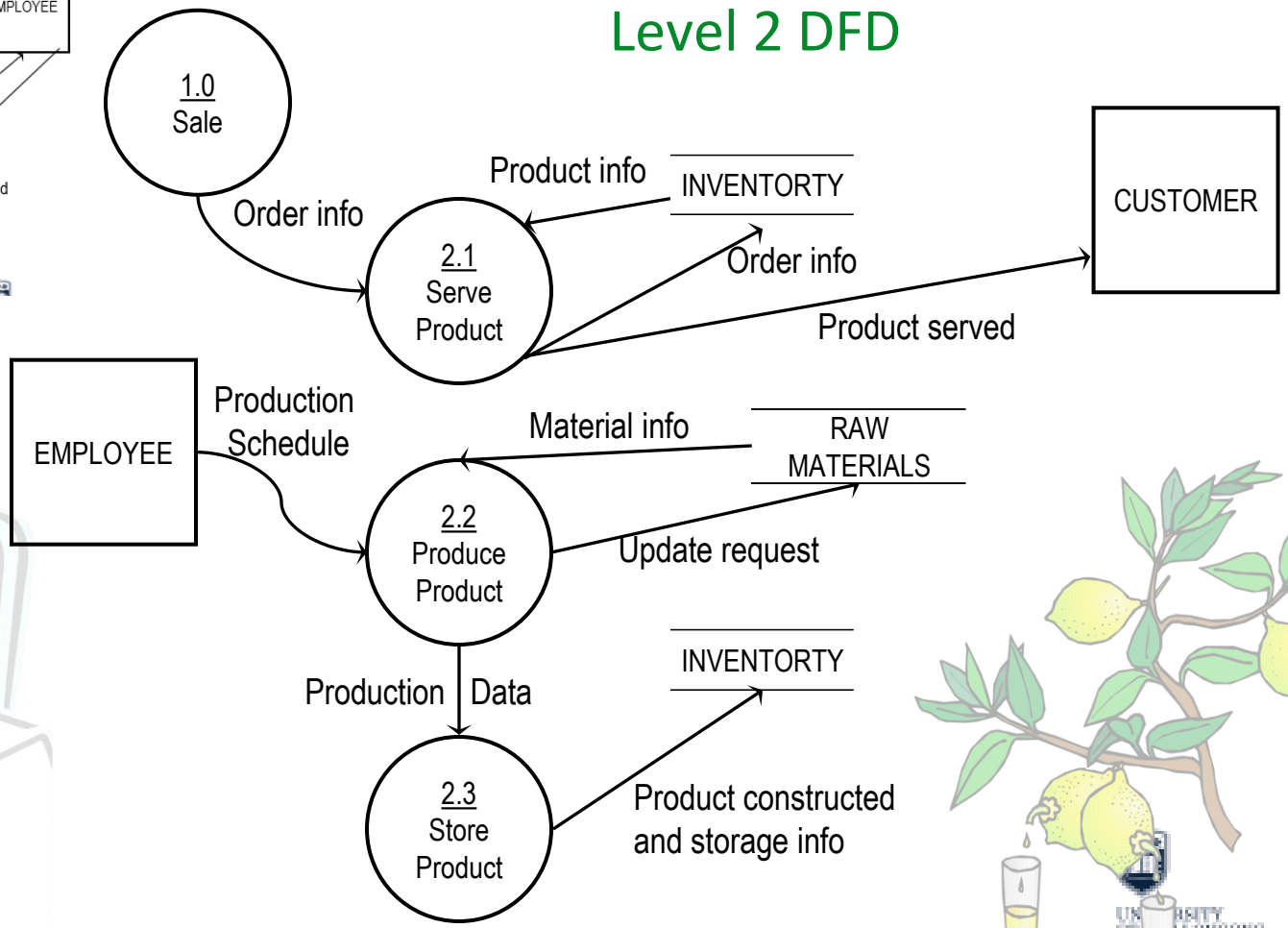


Example: a lemonade system (Level 2)

Level 1 DFD



Level 2 DFD



Customer Order
Collect Payment
Sales forecast

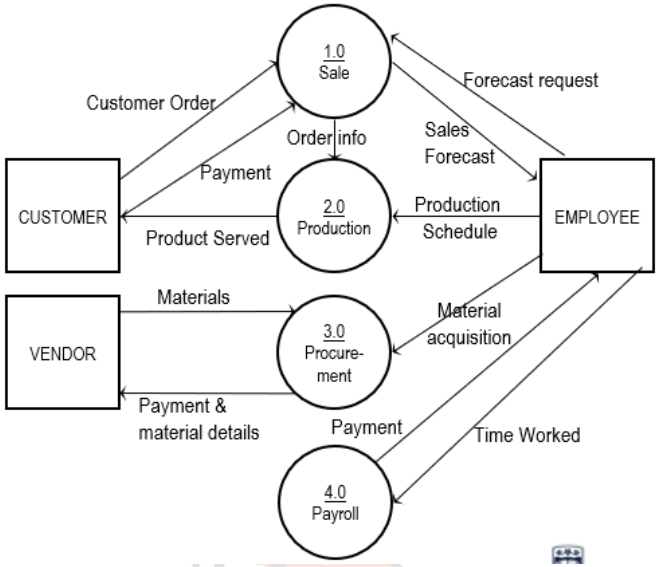
Produce Product
Store Product
Serve Product

Order Raw Materials
Pay for Raw Materials

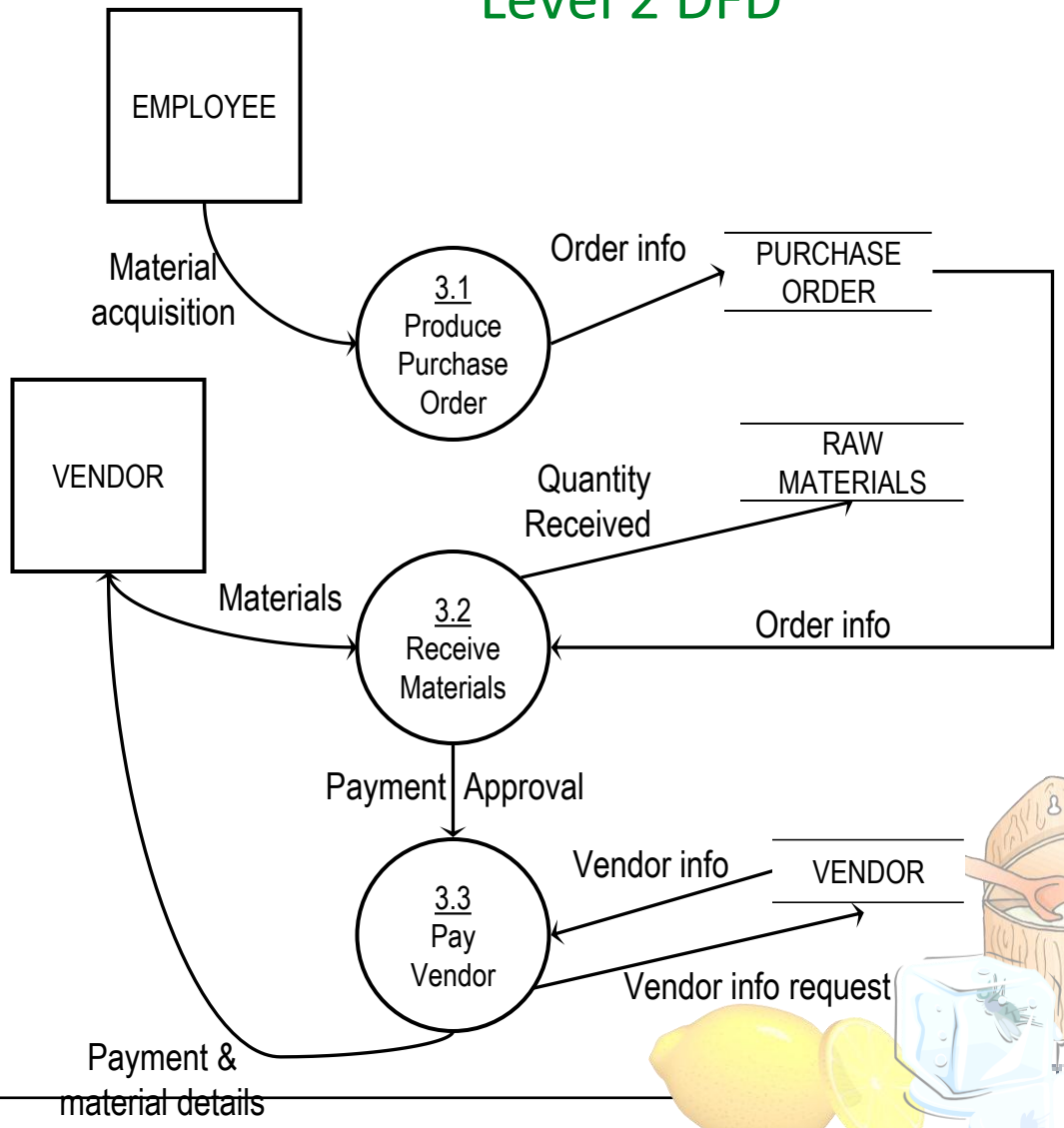
Pay for Labor

Creating Data Flow Diagrams

Level 1 DFD



Level 2 DFD



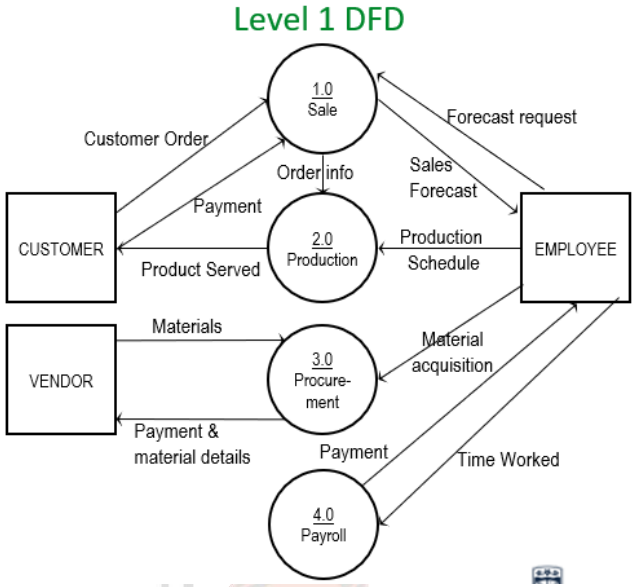
Customer Order
Collect Payment
Sales forecast

Produce Product
Store Product
Serve Product

Order Raw Materials
Pay for Raw Materials

Pay for Labor

Creating Data Flow Diagrams



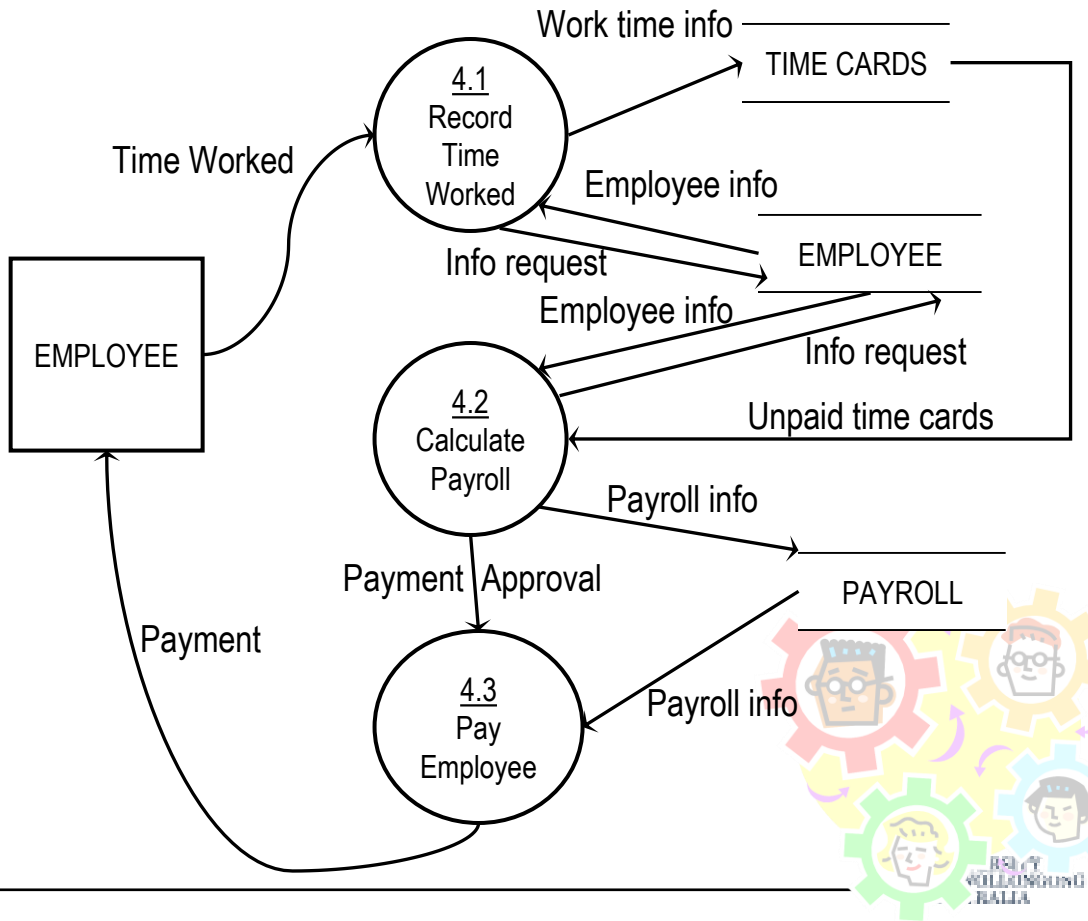
Customer Order
Collect Payment
Sales forecast

Produce Product
Store Product
Serve Product

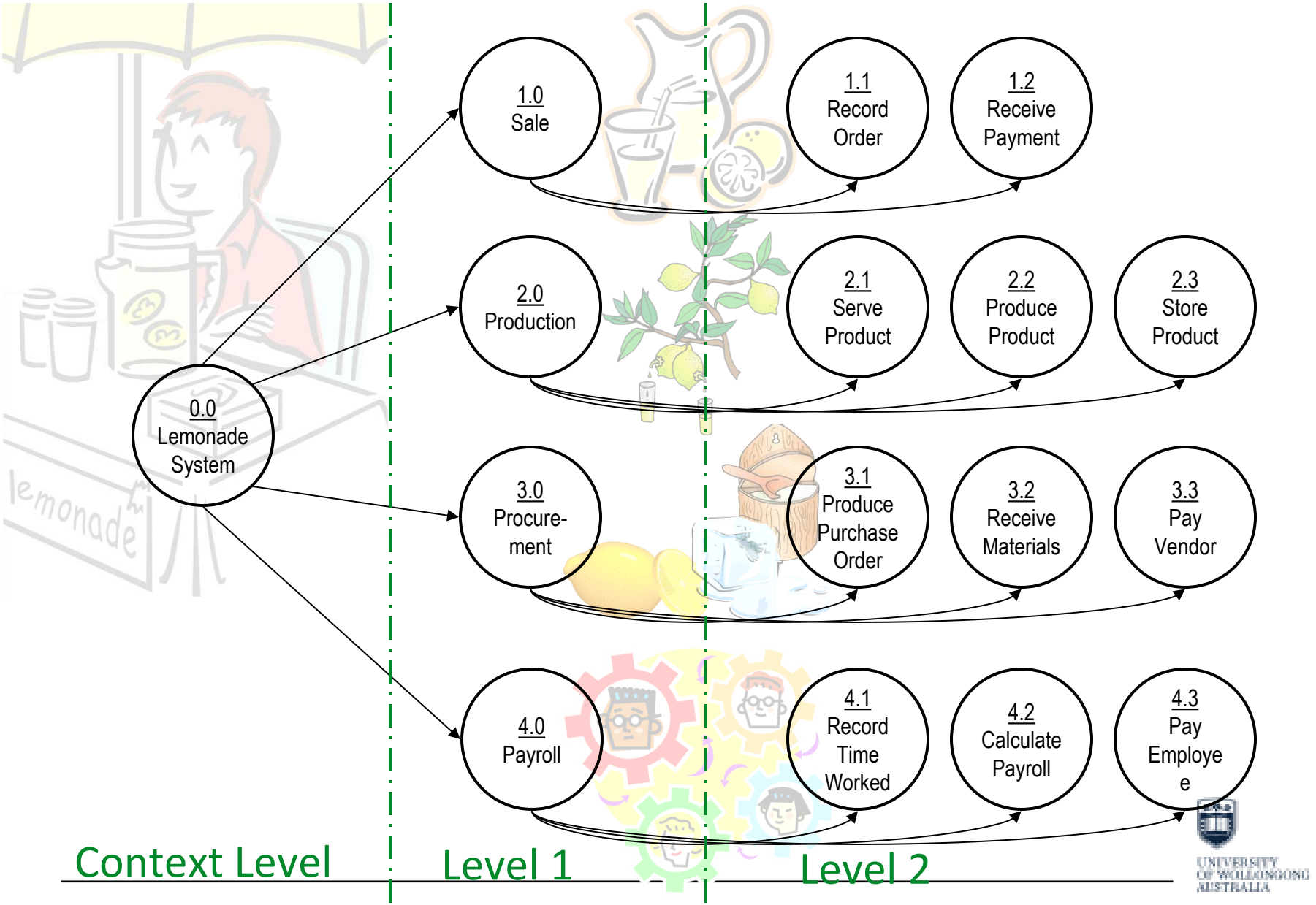
Order Raw Materials
Pay for Raw Materials

Pay for Labor

Level 2 DFD



Process Decomposition



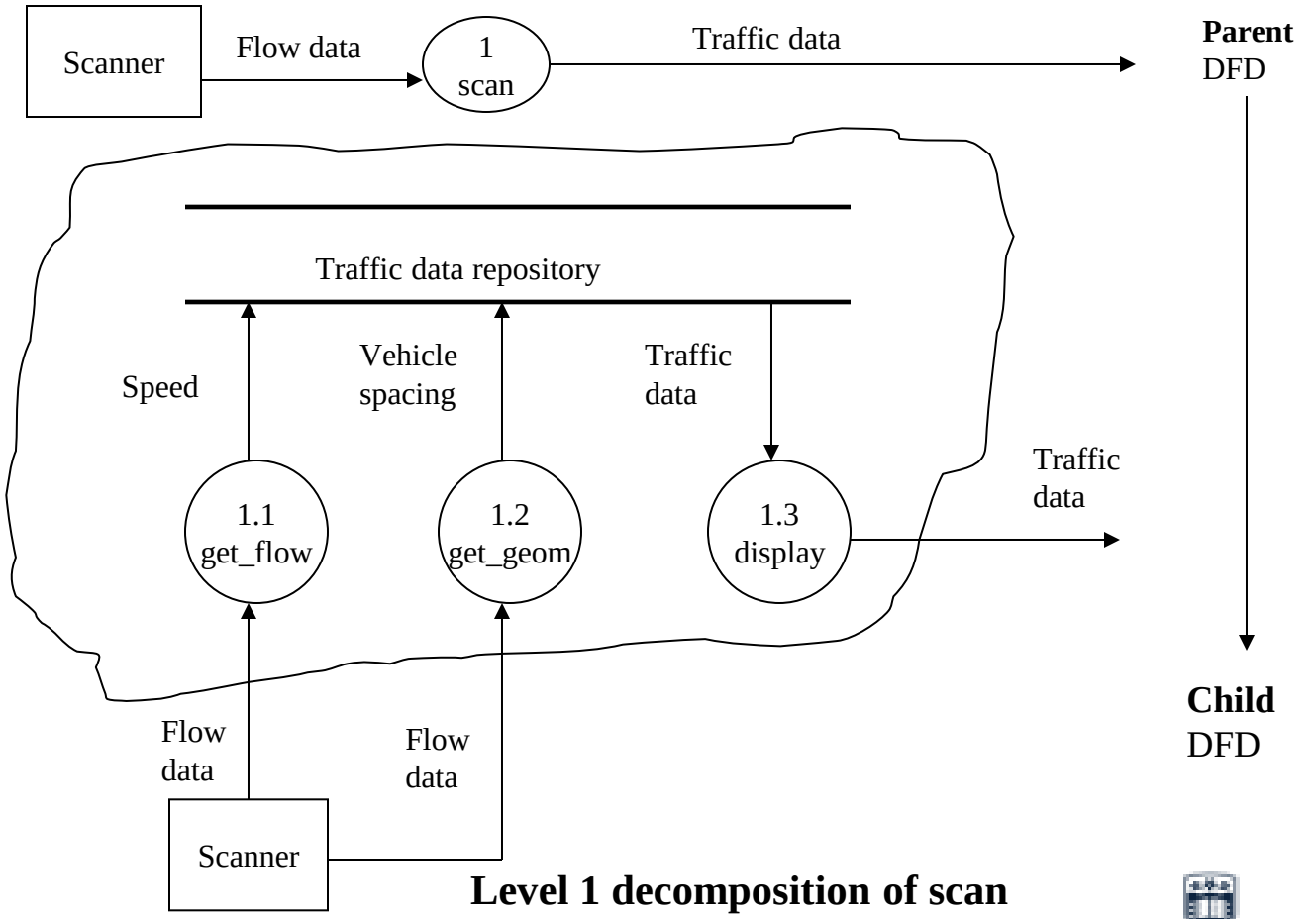
Data Dictionary

- A data dictionary stores information about data items found in a DFD.
- A data dictionary supplies information such as a data typing, required accuracy of data useful to designers and implementers.
- A data dictionary can be used to check the completeness and consistency of DFDs.

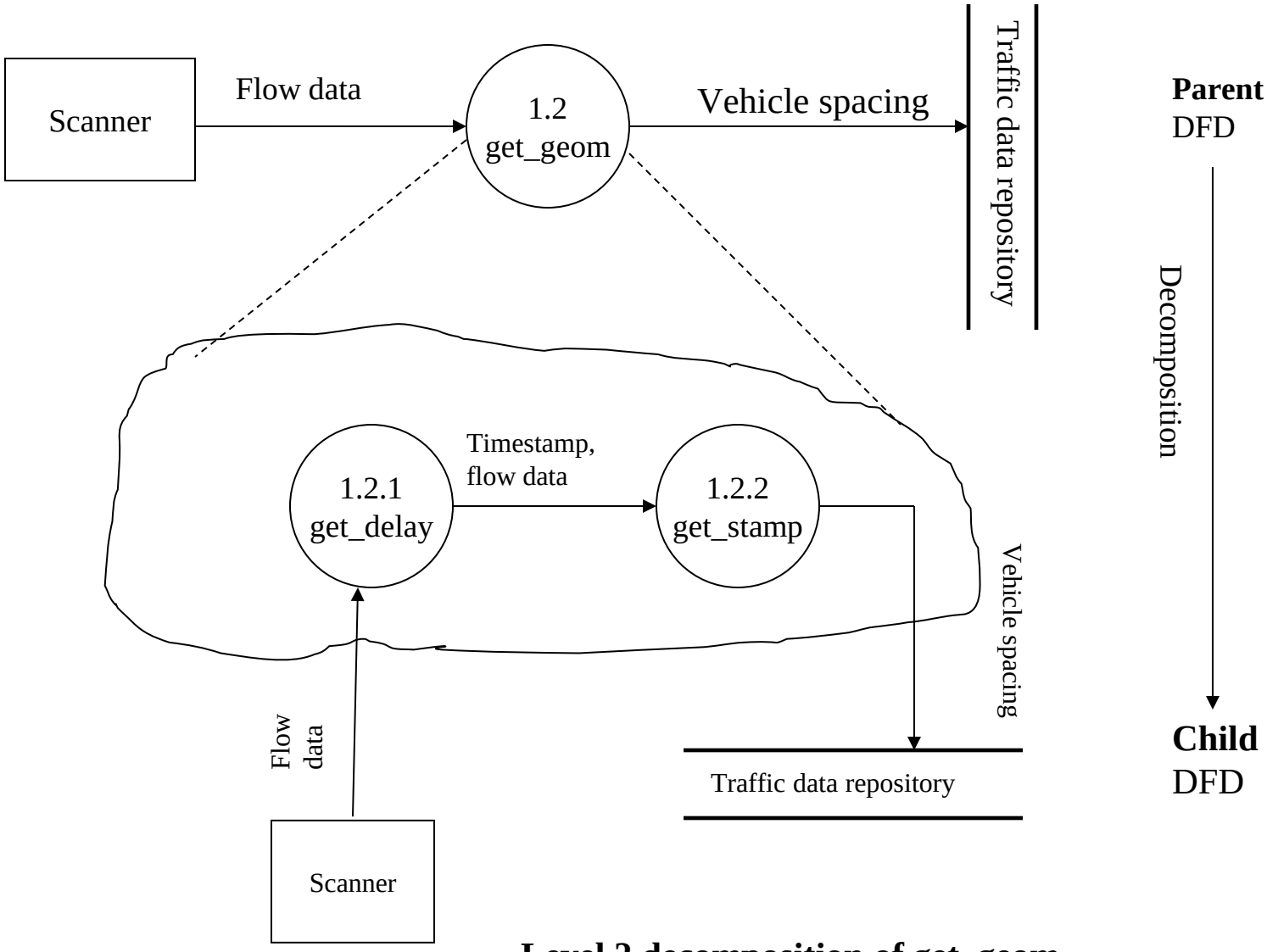
Data Dictionary Features

Information Stored	Explanation
Name	Identifies data item
Alias	Identifies other names, abbreviations used to identify a data item
Data structure (type)	Type of data (e.g. integer, queue)
Description	data description with more details
Duration (begins)	Life span of data (when created)
Accuracy	High, medium, low accuracy
Range of values	Allowable values of data item
Data flows	Identifies process that generate/receive data

Example: Three levels of data-flow diagrams for Scanner in a navigation system



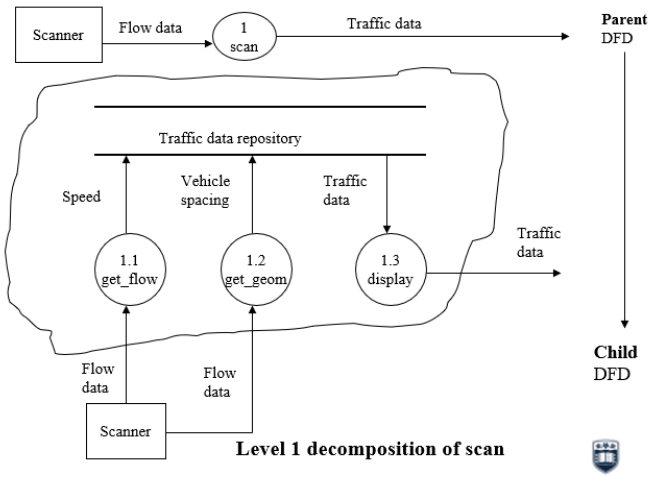
Level 1 decomposition of scan



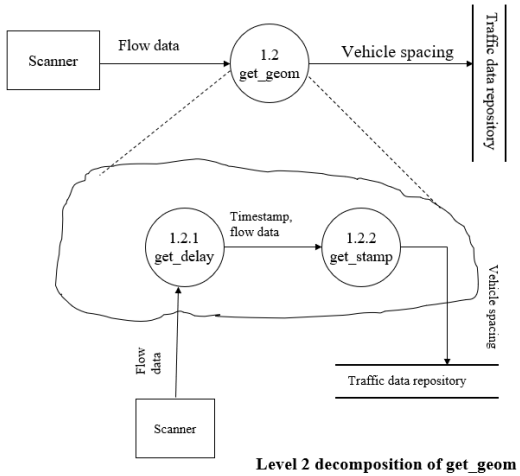
Level 2 decomposition of get_geom

Sample Data Dictionary

Name	Type	About	Range	Accuracy	Data Flows
Speed	Real	Average Vehicle speed	0 ≤ speed ≤ 60 mph	± 0.01	get_flow
Vehicle spacing	Real	Average vehicle spacing	0 ≤ speed ≤ 1000 ft	± 0.01	get_geom
Flow data location	String	Traffic location	10 ≤ speed ≤ 30	Not specified	get_geom, get_flow
Flow data volume	Integer	Current traffic volume	0 ≤ volume ≤ 4000/hr	Not specified	get_geom, get_flow
Time	String	Date and time	20 characters	Not specified	get_timestamp



Level 1 decomposition of scan



Level 2 decomposition of get_geom

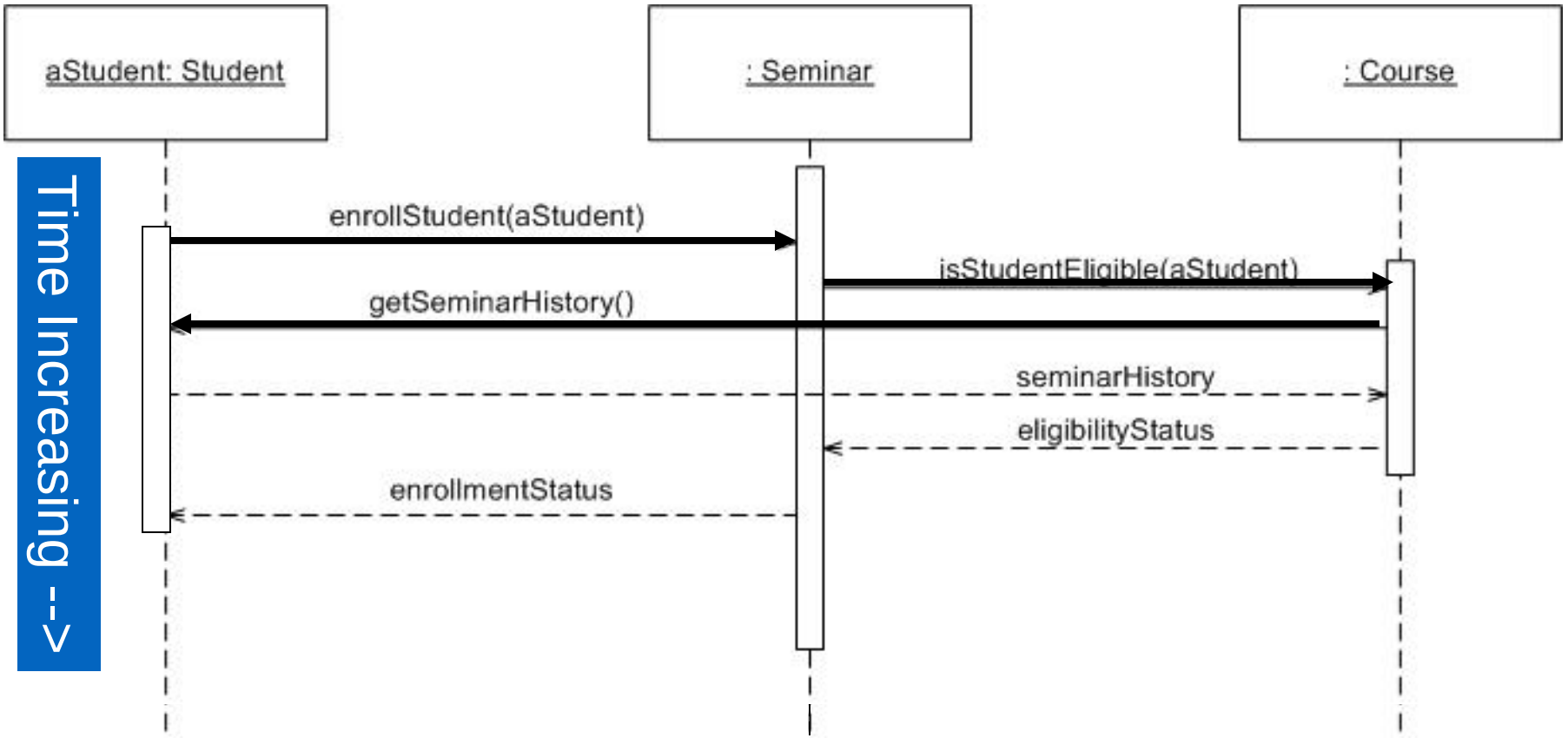
UNIVERSITY OF WOLLONGONG AUSTRALIA

Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

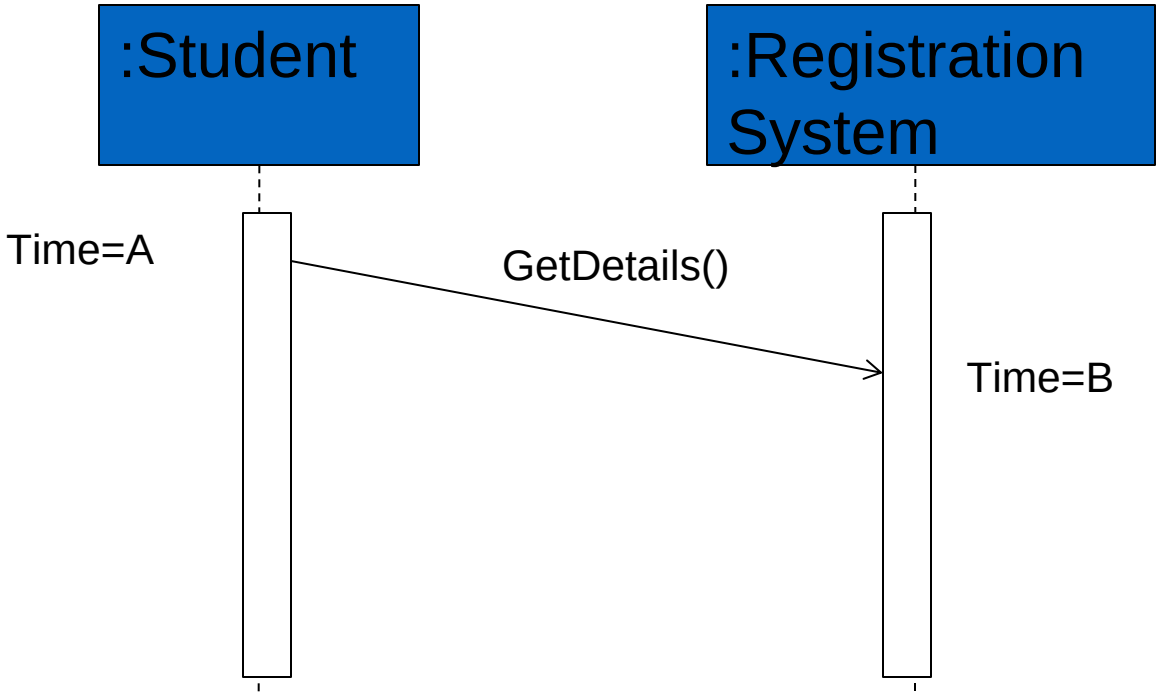
Sequence Diagram



horizontal lines indicate instantaneous actions. Additionally if ActivityA happens before ActivityB, ActivityA must be above activity A

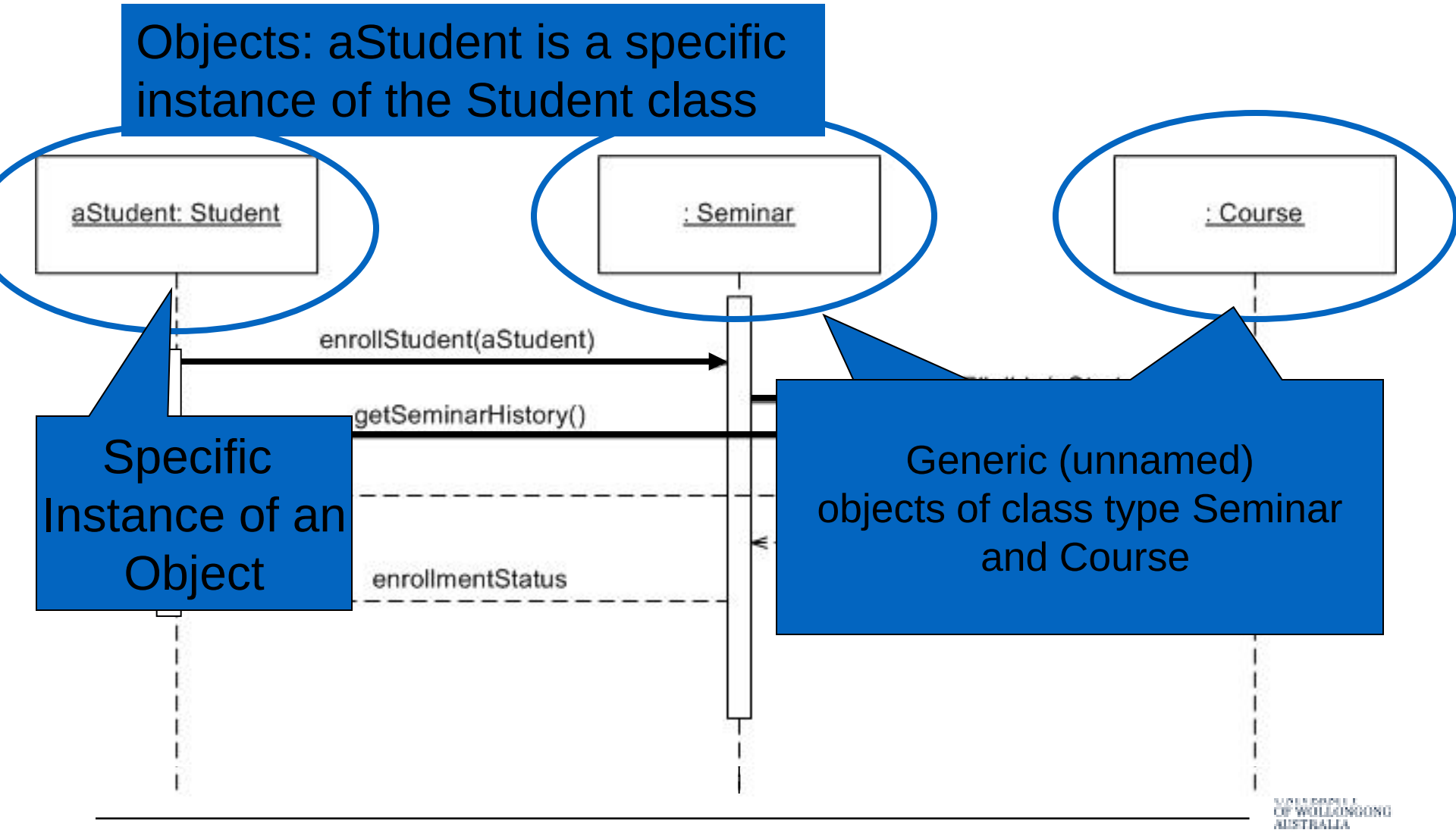
Lower = Later!

Diagonal Lines

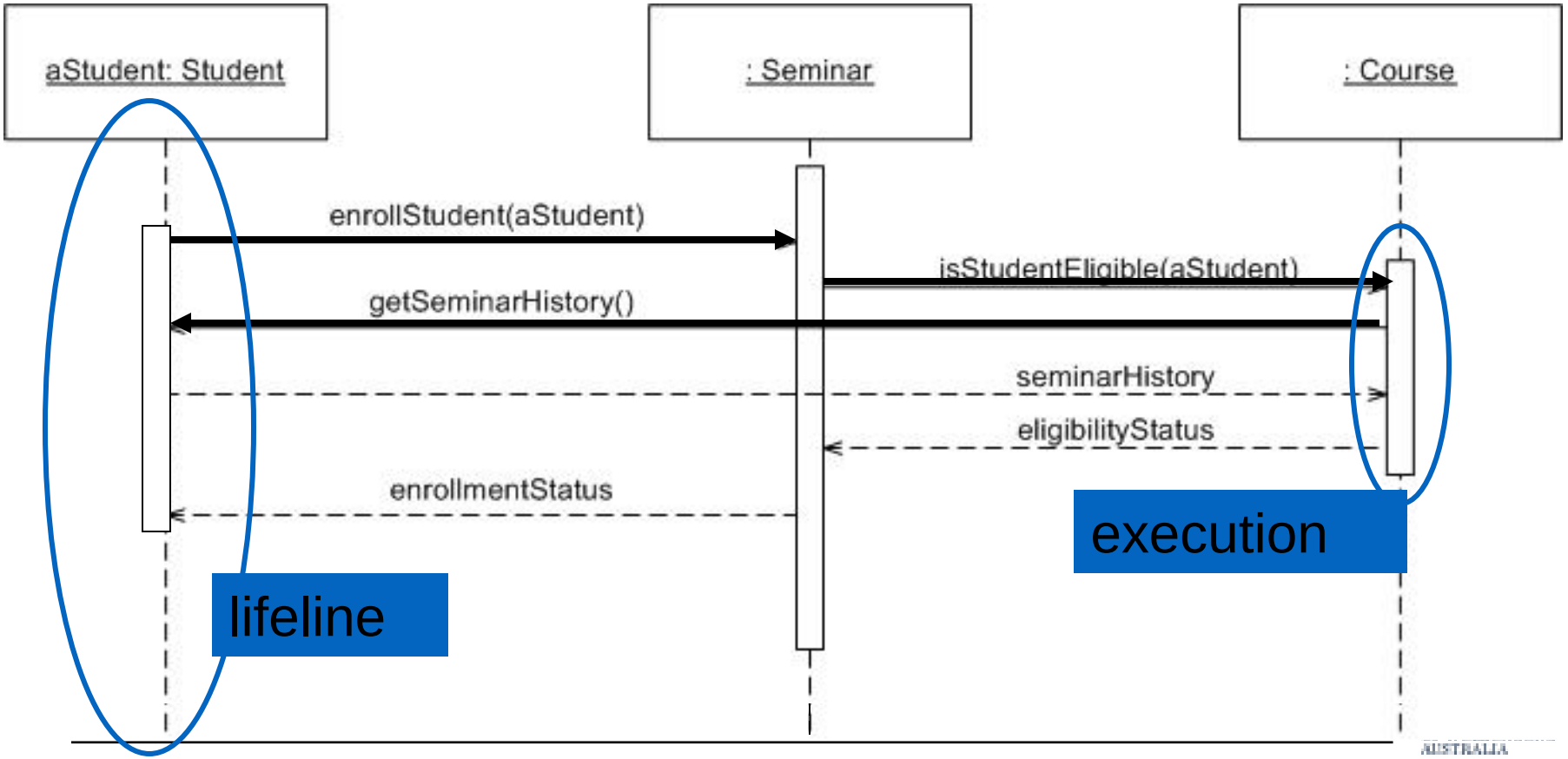


- What does this mean?

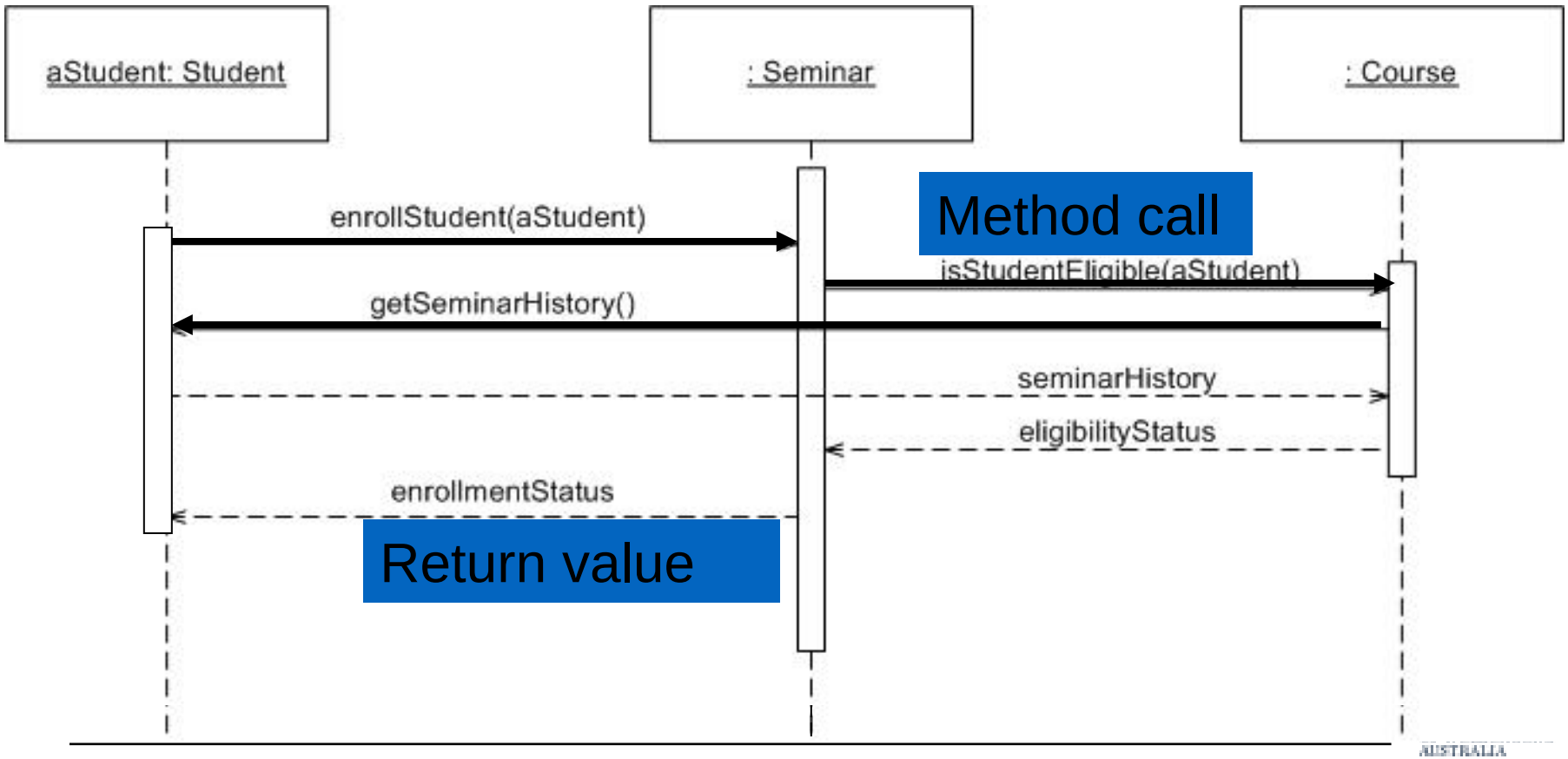
Components


Objects: aStudent is a specific instance of the Student class
aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
getSeminarHistory()
enrollmentStatus
Specific Instance of an Object
Generic (unnamed) objects of class type Seminar and Course

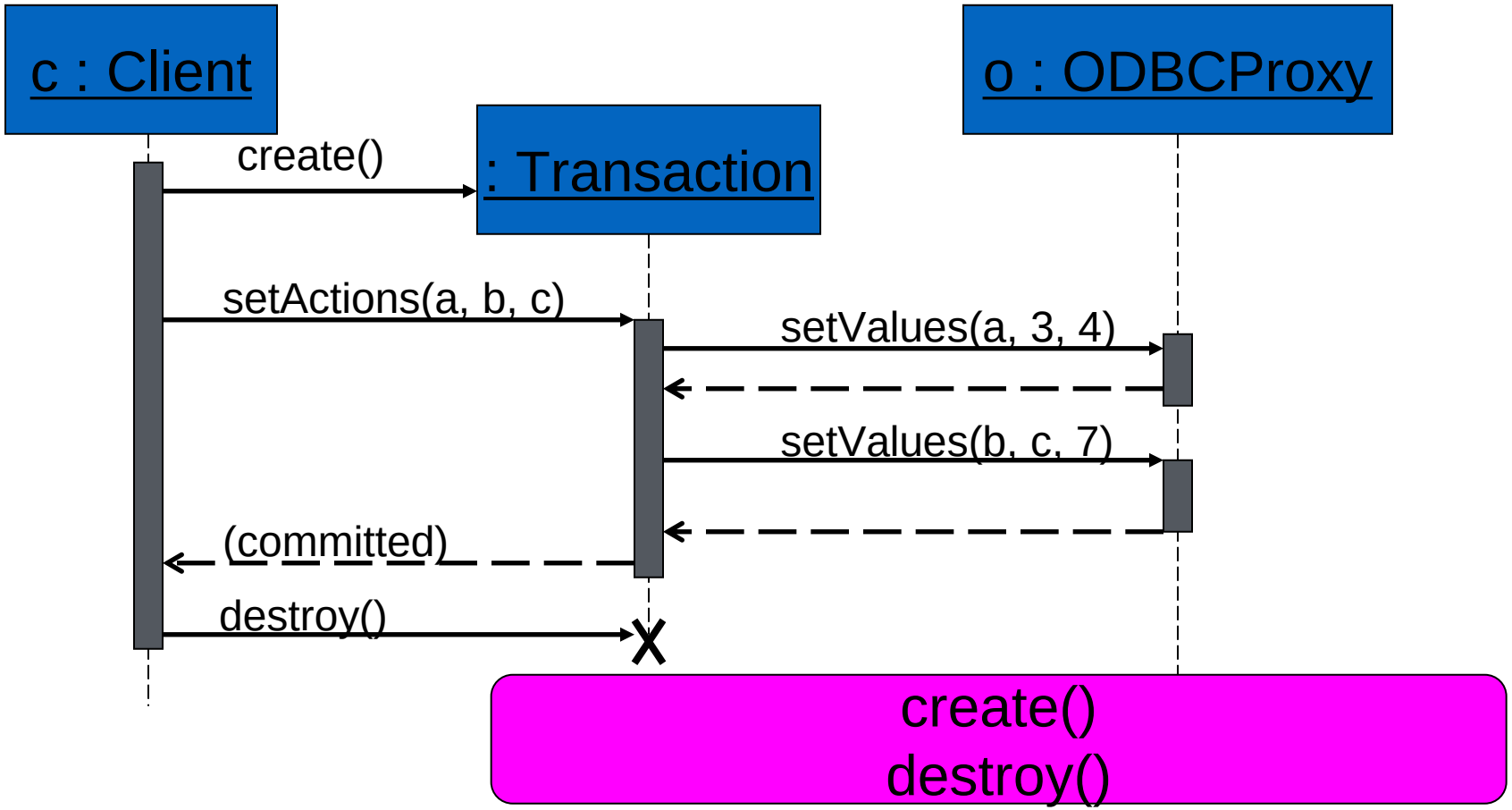
Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
lifeline
execution

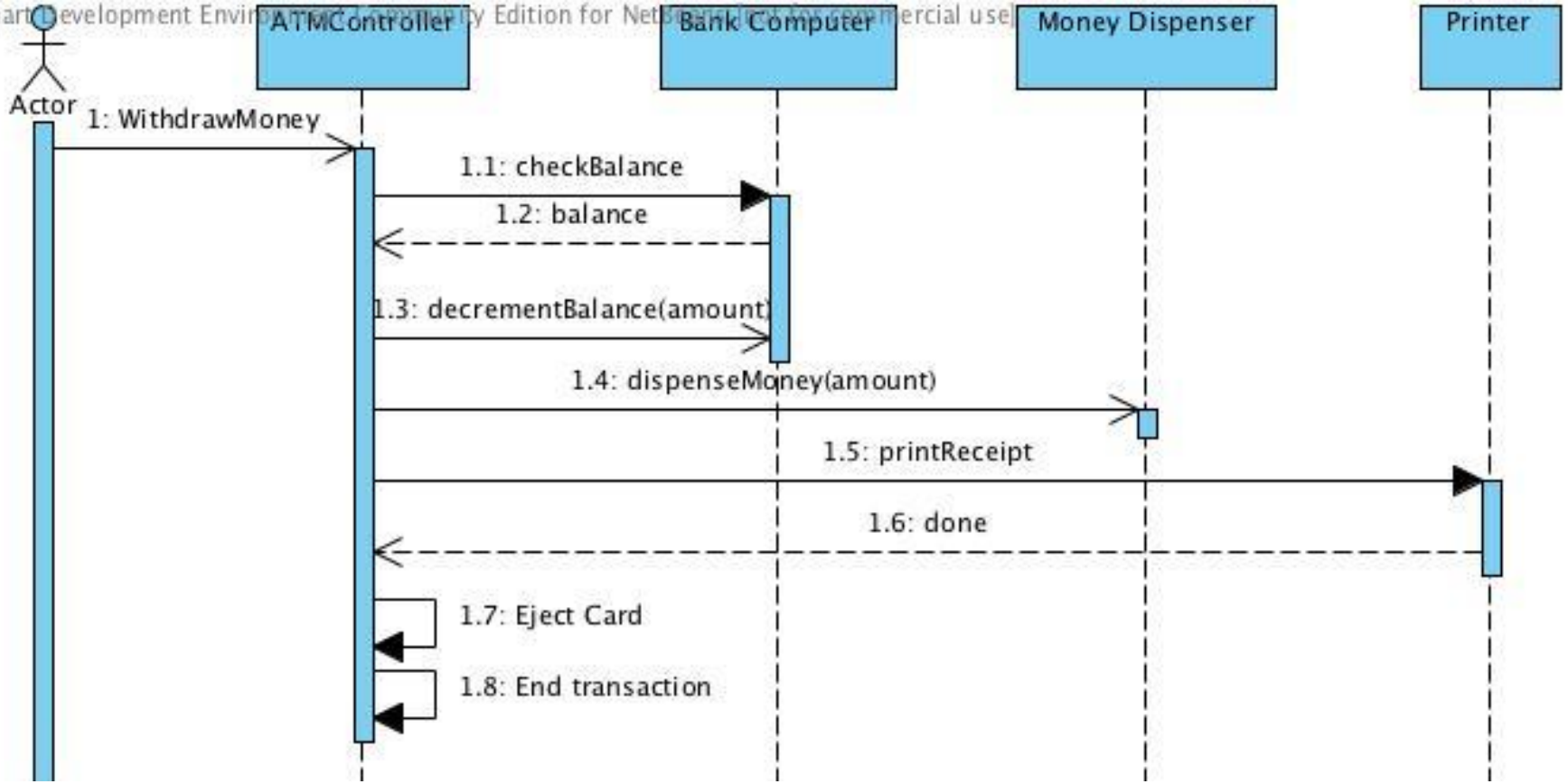
Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
Method call
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
Return value

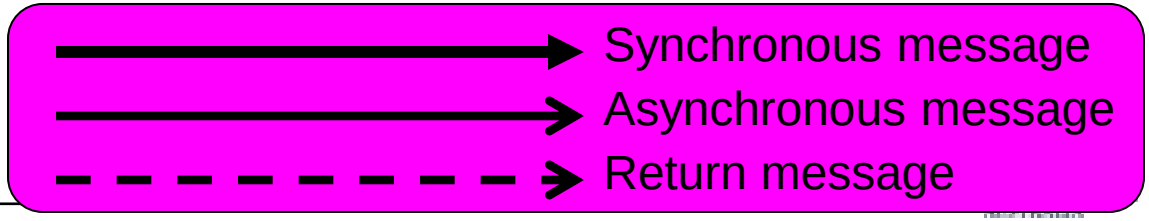
Components


c : Client
o : ODBCProxy
: Transaction
create()
setActions(a, b, c)
setValues(a, 3, 4)
setValues(b, c, 7)
(committed)
destroy()
create()
destroy()

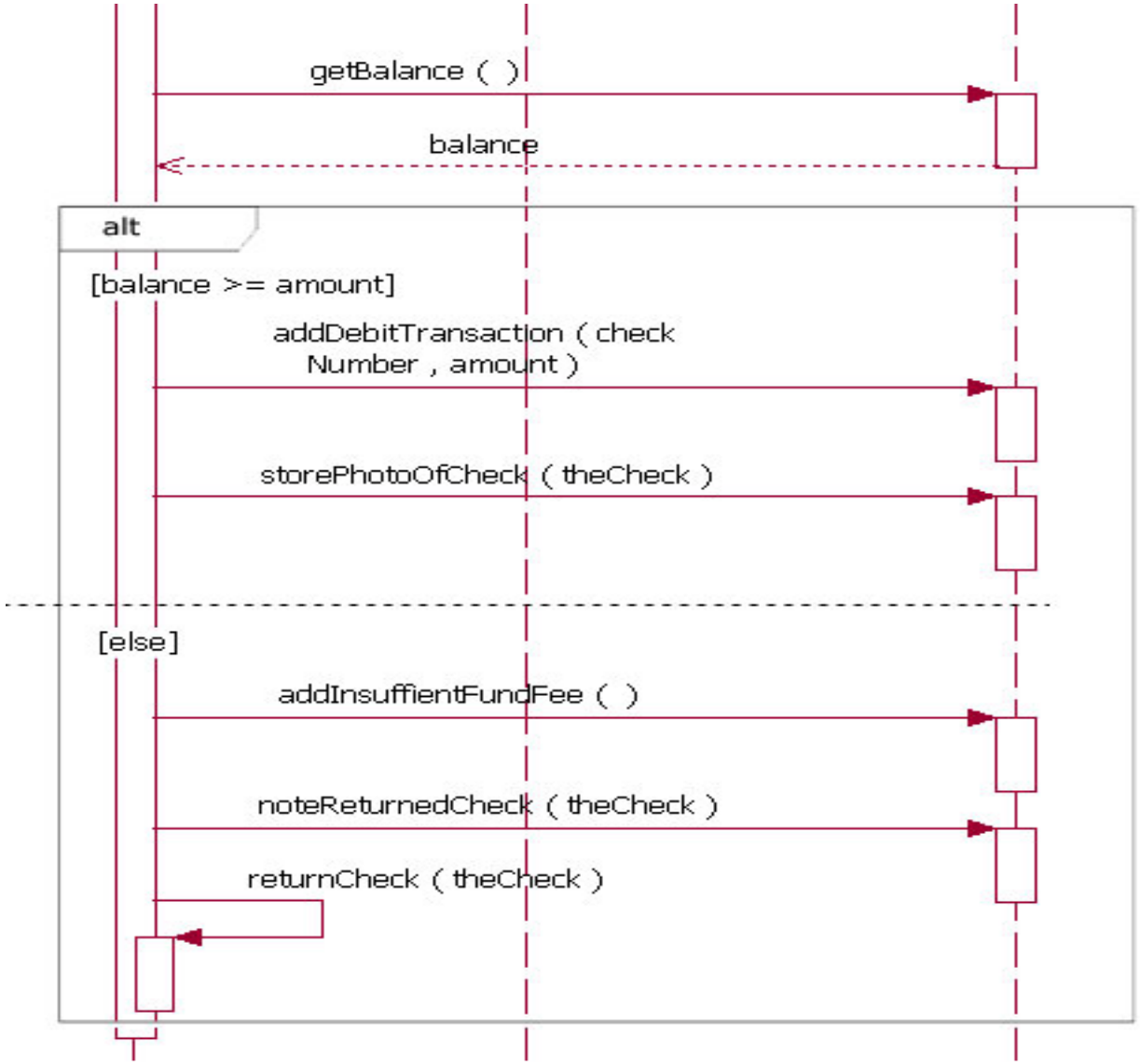
Async Message Example



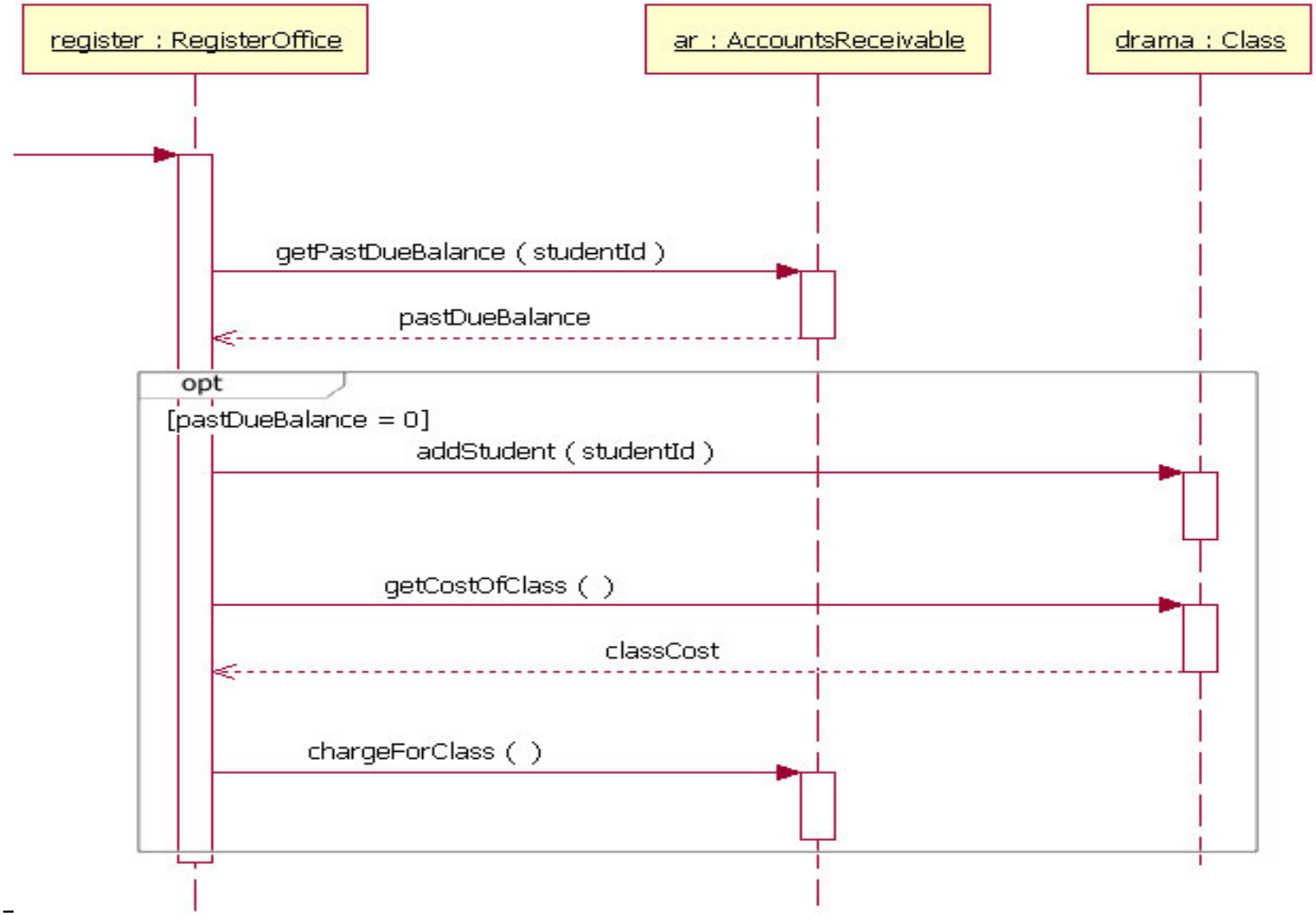
There are problems here… what are they?



Components: alt/else

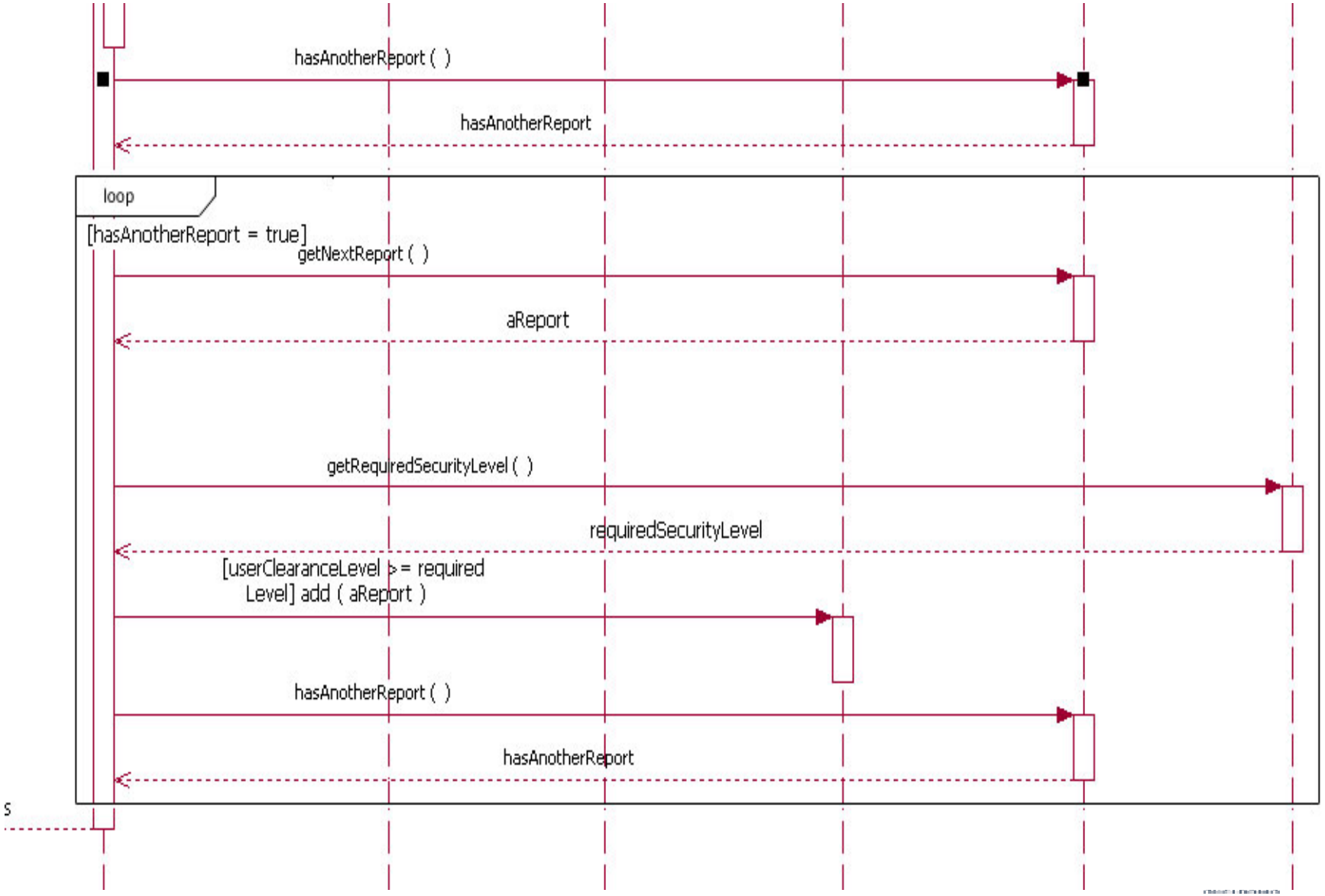


Components: option



Coming up: Components: loop

Components: loop



Finite State Machines (FSM)

At its lowest level, observable behavior of a system can be described with Finite State Machines (FSMs), which are special cases of automata (abstract machine).

The state of a system can be determined by checking the value of one or more state variables.

Suppose that a system might have state variable q1, q2, q3, q4 with values from the set {waiting, reading, writing, searching}. The possible state sequence for the system might be:

waiting →	reading →	searching →	writing →	waiting
(state q1)	(q2)	(q3)	(q4)	(q1)

Moore and Mealy FSMs

Two types of finite state machines differ in the output logic:

- **Moore FSM:**
  - outputs depend only on the current state
- **Mealy FSM:**
  - outputs depend on the current state and the inputs

Mealy is more general

Definition of a finite-state machine

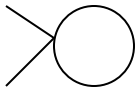
A finite-state machine M is defined by a five-tuple (Q, F, q0, S, δ) where

- Q is a finite set of states {q0, q1, q2, …, qn}
- F is a subset of final (accepting) state of Q
- q0 is a single start state in Q
- S is an input/output alphabet
- δ: Q x S $\rightarrow$Q maps the current input and current state into the next state

Notation FSM diagrams

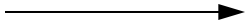
State

Start state

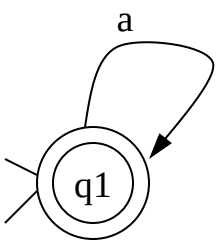


Final state

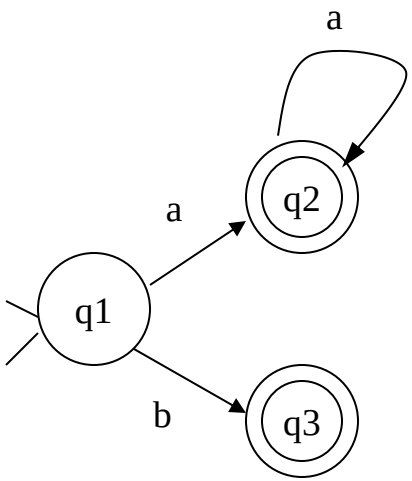
Transition



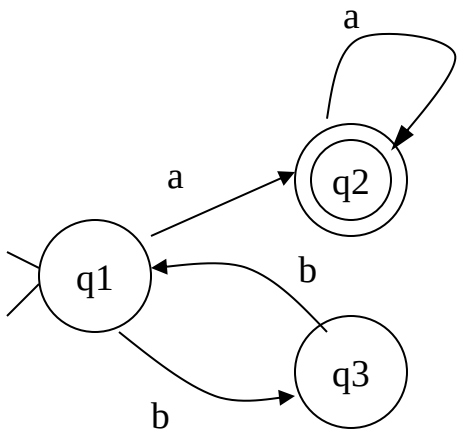
Sample finite-state machines



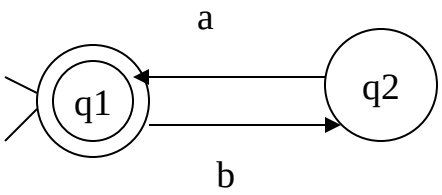
(a) accepts
a, aa, aaa, ...



(b) accepts b or
a, aa, aaa, aaaa, ...

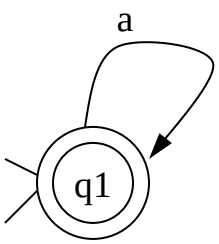


(c) accepts
bba, bbaa, …, or
a, aa, aaa, aaaa, ...

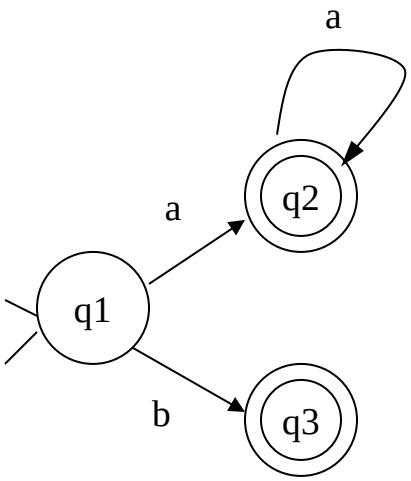


(d) accepts
ba, baba, ...

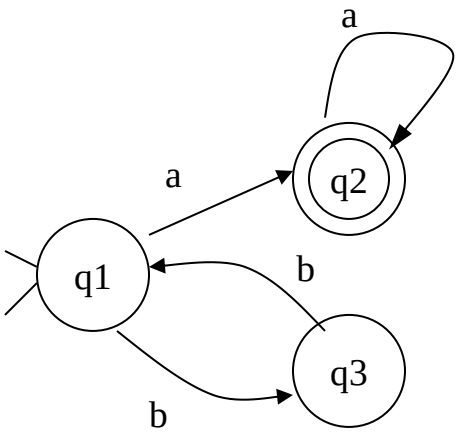
Sample finite-state machines



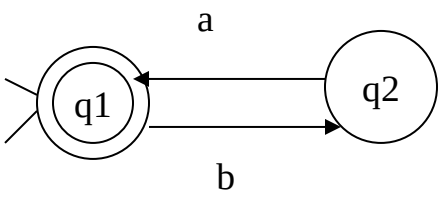
(a) accepts
a, aa, aaa, ...



(b) accepts b or
a, aa, aaa, aaaa, ...



(c) accepts
bba, bbaa, …, or
a, aa, aaa, aaaa, ...



(d) accepts
ba, baba, ...

A finite-state machine M is defined by a five-tuple (Q, F, q0, S, δ) where

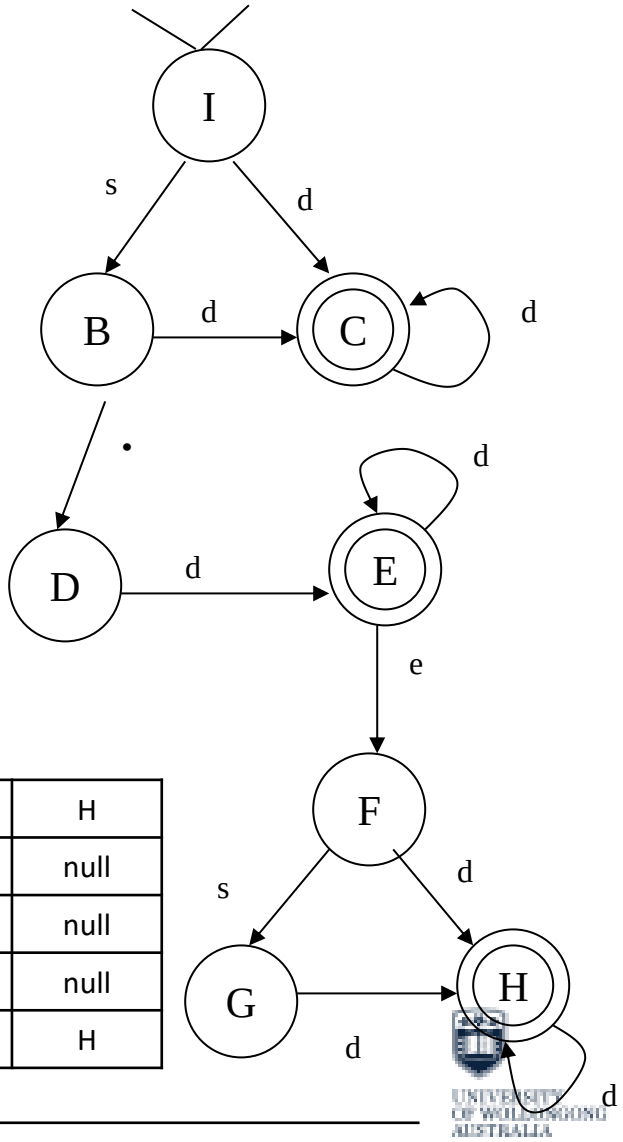
- Q is a finite set of states {q0, q1, q2, …, qn}
- F is a subset of final (accepting) state of Q
- q0 is a single start state in Q
- S is an input/output alphabet
- δ: Q x S $\rightarrow$Q maps the current input and current state into the next state

State table

The behaviors of FSMs can be represented in a table form. For example, a complete representation of the behavior of machine (b) is given in the following table.

State \ Input	a	b
q1	q2	q3
q2	q2	Φ
q3	Φ	Φ

Example: Use FSM to give the specification of numeric constants as accepted by the Pascal programming language



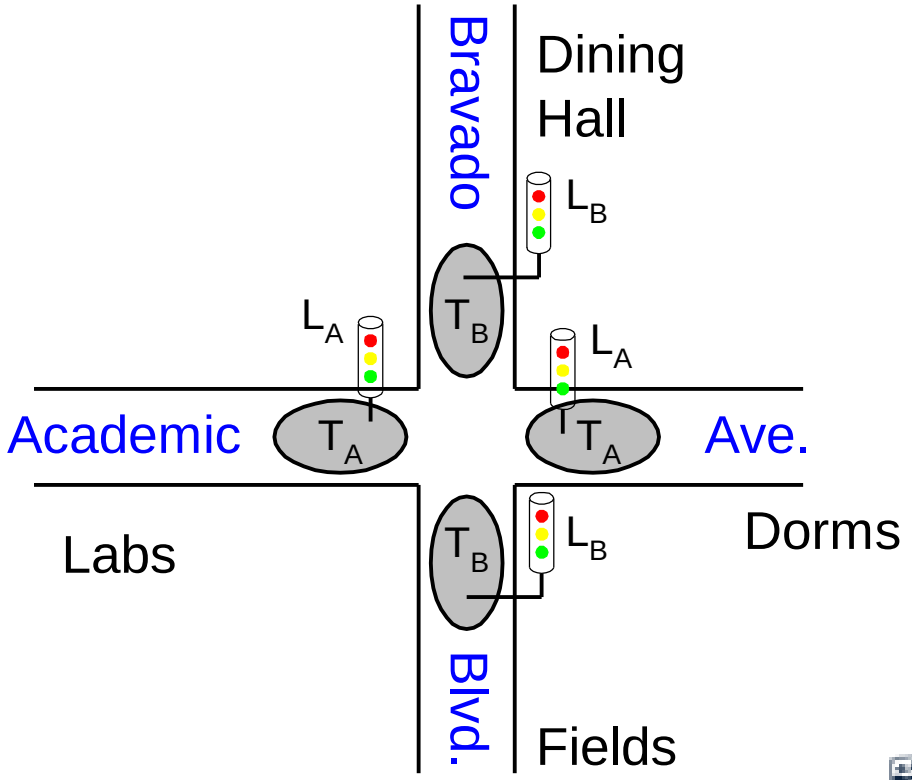
Inputs: S = { s, ., e, d}
States: Q={I, B, C, D, E, F, G, H}
Final states: F={C, E, H}
Initial state: q0=I

State Table

	I	B	C	D	E	F	G	H
s	B	null	null	null	null	G	null	null
.	null	D	null	null	null	null	null	null
e	null	null	null	null	F	null	null	null
d	C	C	C	E	E	H	H	H

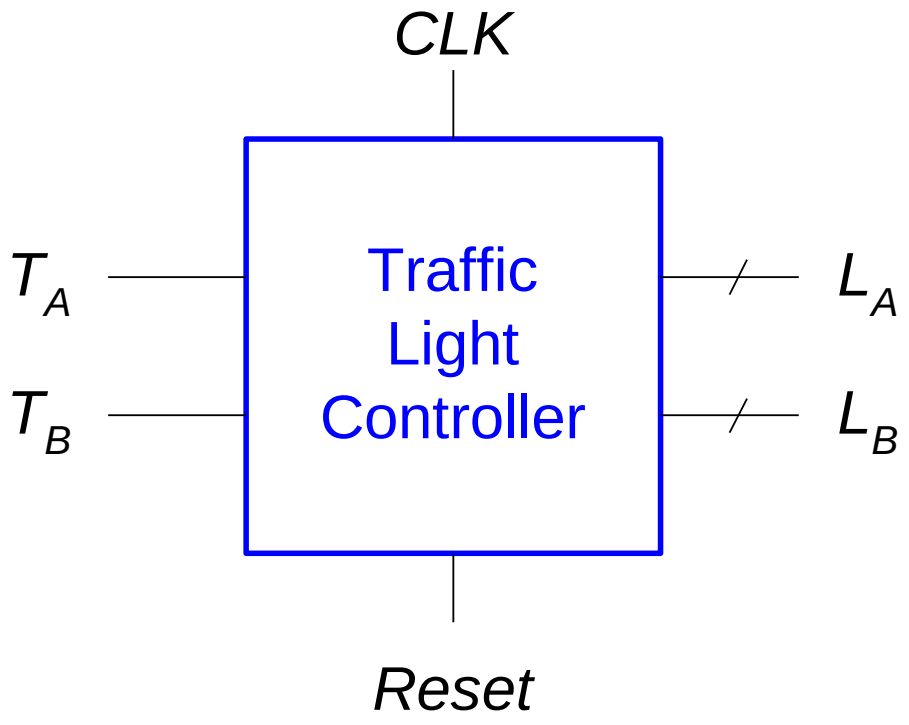
FSM Example

- Traffic light controller
  - Traffic sensors: T_A, T_B (TRUE when there's traffic)
  - Lights: L_A, L_B



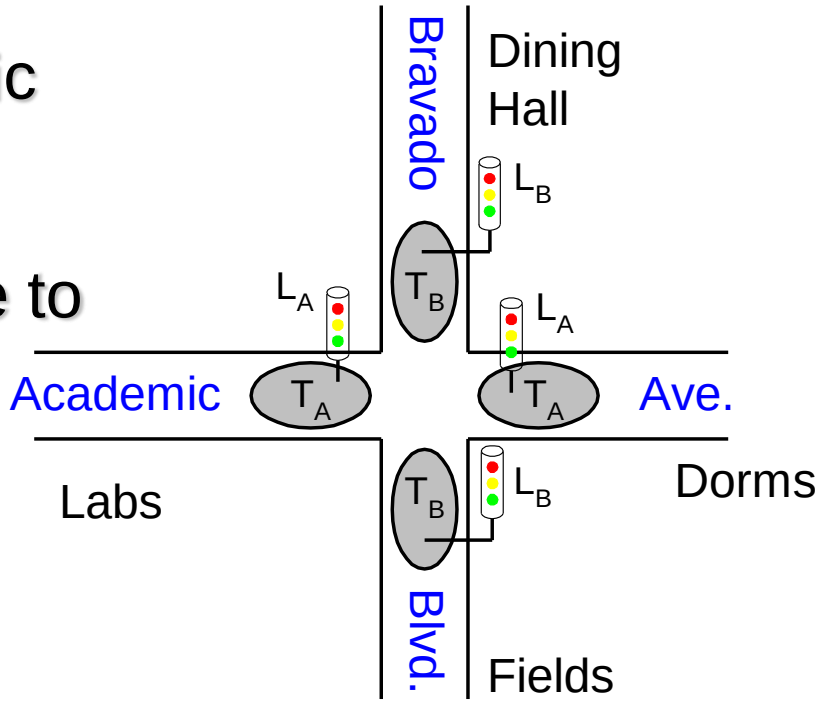
FSM Black Box

- Inputs:
  - CLK, Reset, T_A, T_B
- Outputs: L_A, L_B



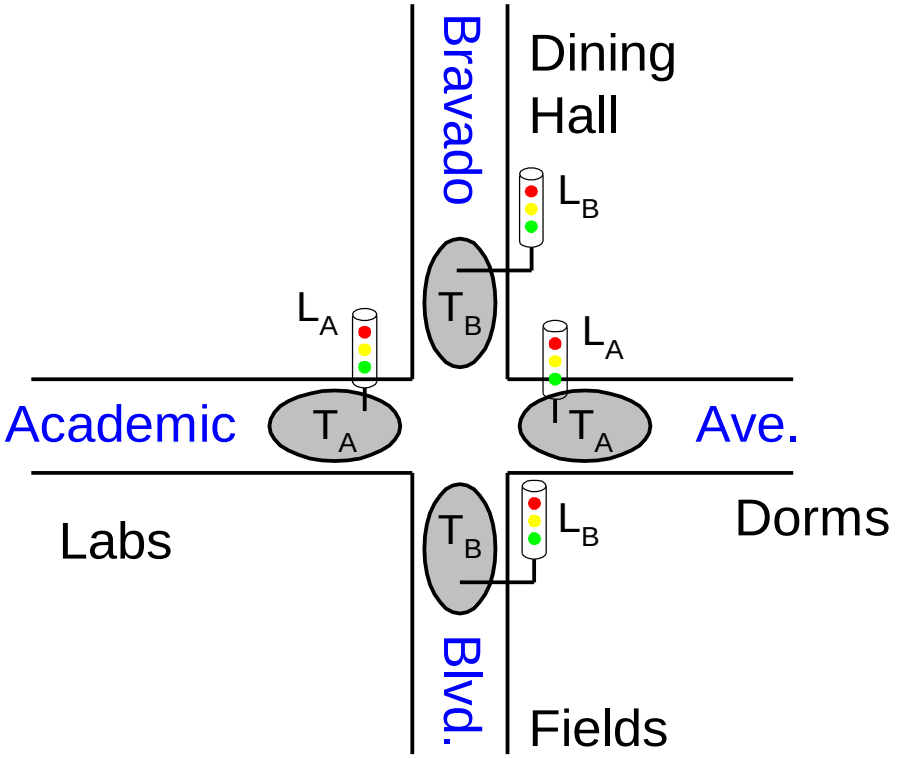
Design Simple FSM

- When *reset*, L_A is green and L_B is red
- As long as traffic on Academic (T_A high), keep L_A green
- When T_A goes low, sequence to traffic on Bravado
- Follow same algorithm for Bravado
- Let's say clock period is 5 sec (time for yellow light)



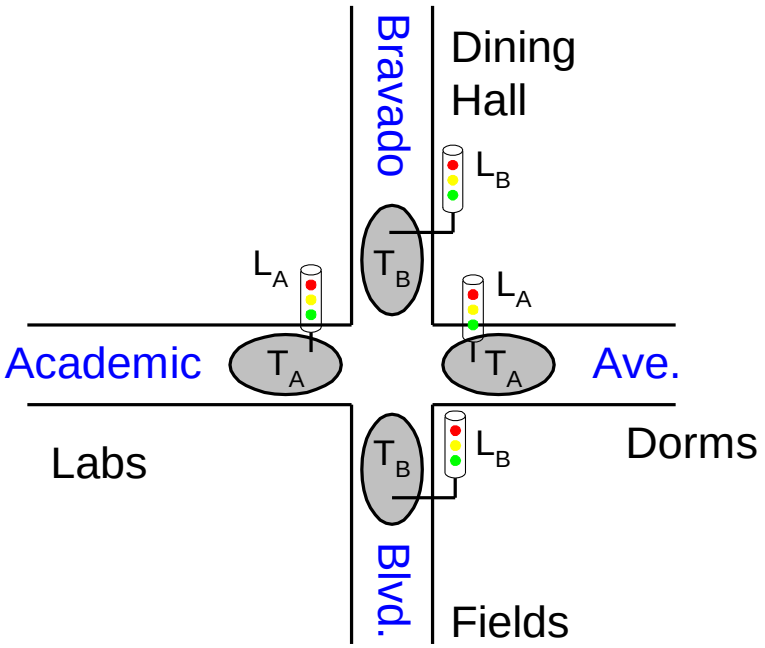
States

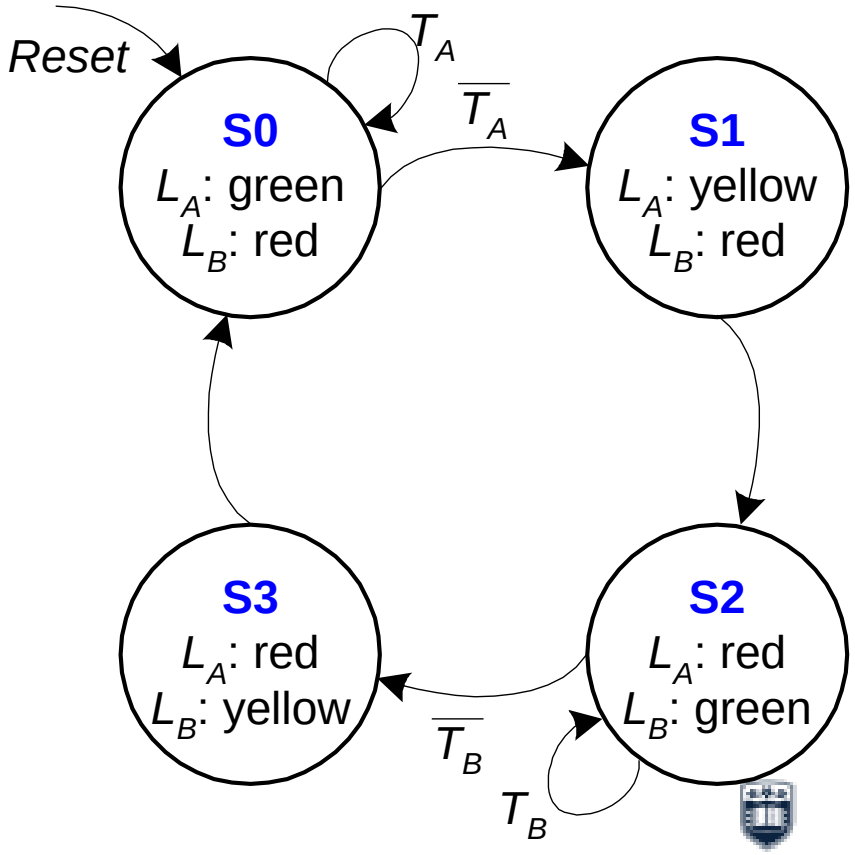
- What sequence do the traffic lights follow?
  - Reset ➔ State 0, L_A is green and L_B is red
  - Next (on board)?



State Transition Diagram

- **Moore FSM: outputs labeled in each state**
- **States: Circles**
- **Transitions: Arcs**





State Transition Table

Current State S	Inputs T_A	T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

FSM Encoded State Transition Table

State	Encoding
S0	00
S1	01
S2	10
S3	11

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

$$S'_1 = S_1 (XOR) S_0$$

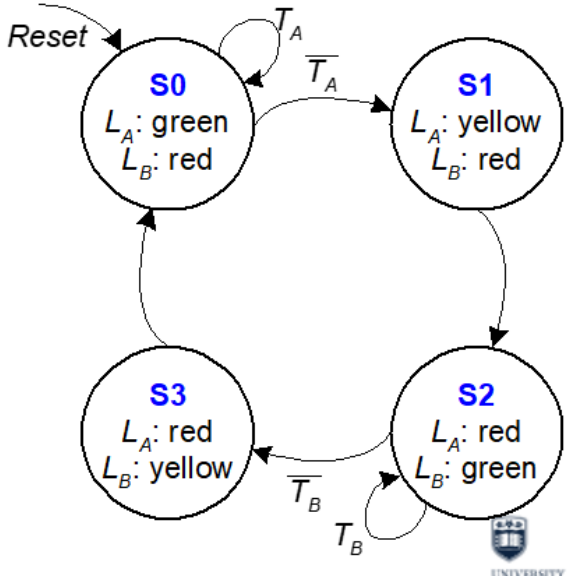
$$S'_0 = \overline{S_0}\,\overline{S_1}\,\overline{T_A} (\text{OR})\, \overline{S_0} S_1 \overline{T_B}$$

FSM Output Table

Output	Encoding
green	00
yellow	01
red	10

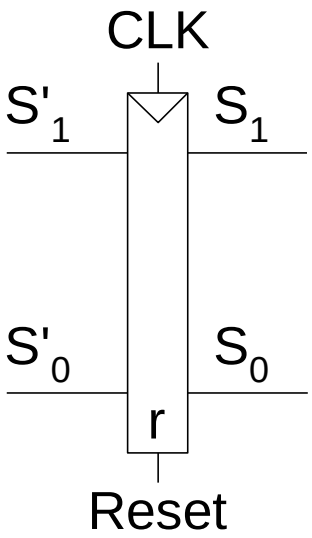
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

State	Encoding
S0	00
S1	01
S2	10
S3	11



$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$

FSM Schematic: State Register



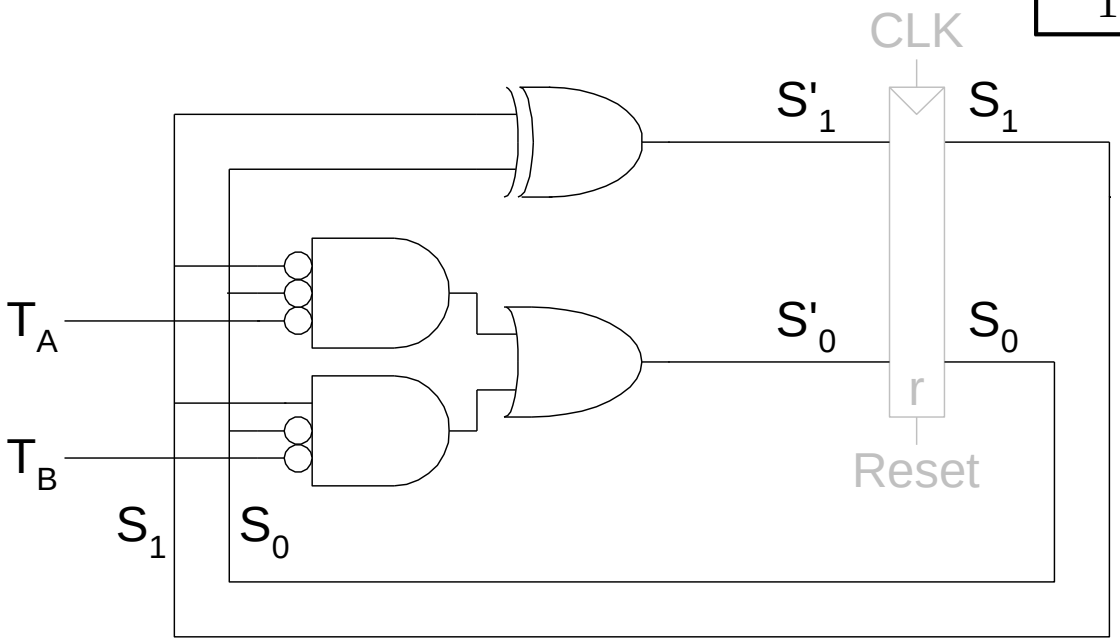
state register

Next State Logic

$$S_1' = S_1 (XOR) S_0$$

$$S_0' = \overline{S_0}\overline{S_1}\ \overline{T_A} (\text{OR})\ \overline{S_0} S_1 \overline{T_B}$$

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0



Output Logic

$$L_{A1} = S_1$$

$$L_{A0} = \overline{S_1} S_0$$

$$L_{B1} = \overline{S_1}$$

$$L_{B0} = S_1 S_0$$