SWEN30006
Software Design and Modelling

UML State Machine Diagrams & Modelling

Textbook: Larman Chapter 29

“No, no, you’re not thinking, you’re just being logical."

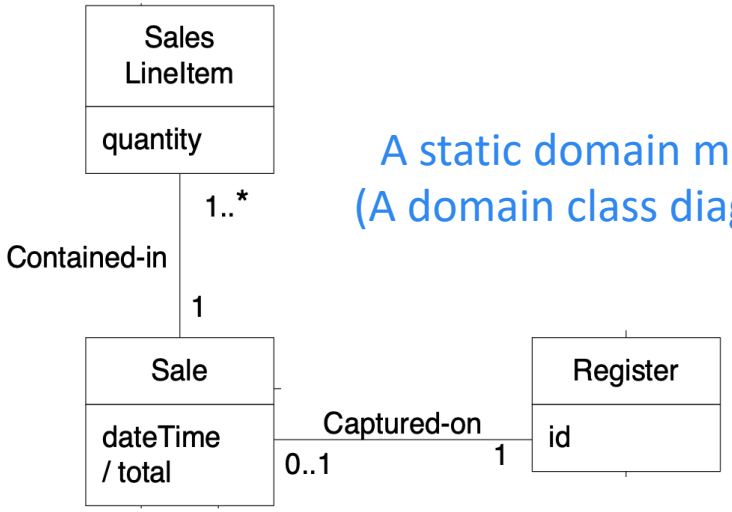
—Niels Bohr

Learning Objectives

On completion of this topic you should be able to:

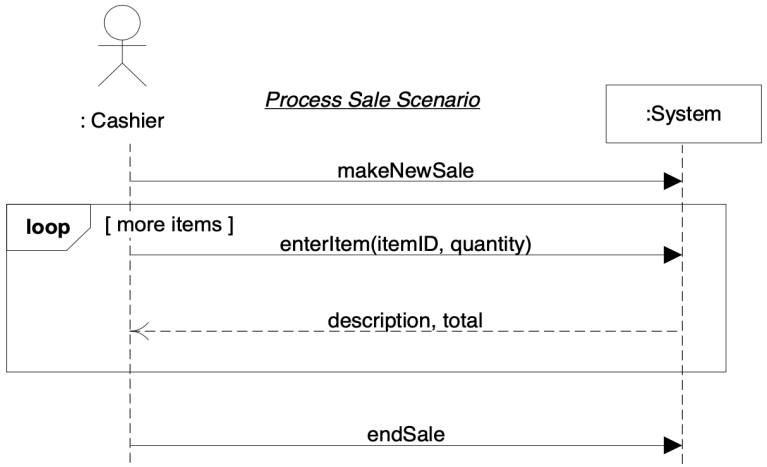
- Be aware of where state machine modelling is applicable.
- Understand UML state machine diagram notation.

(Revisited) Software Models

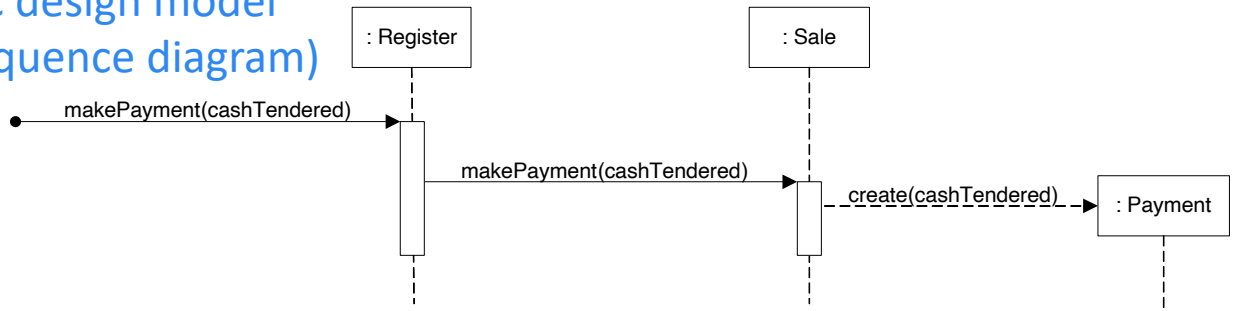


A static domain model
(A domain class diagram)

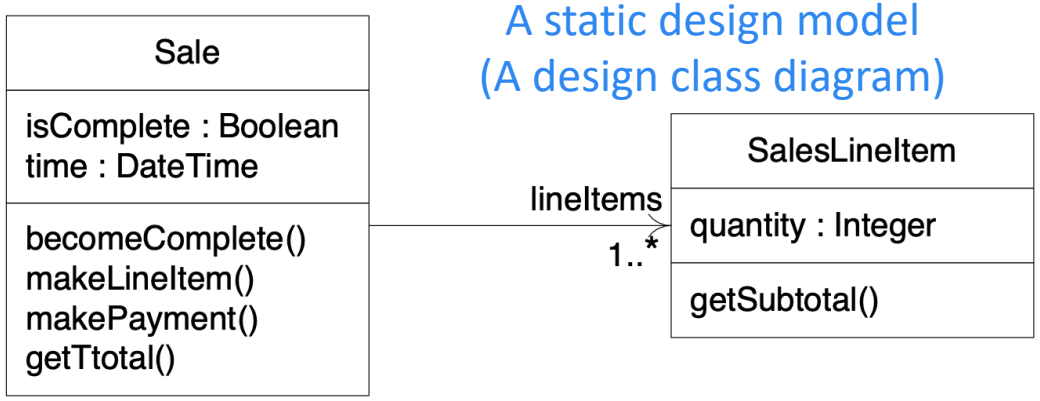
A dynamic domain model
(A system sequence diagram)



A dynamic design model
(A design sequence diagram)



A static design model
(A design class diagram)



If an object (or a system) can have different behavior based on its status or condition, how can we model it?

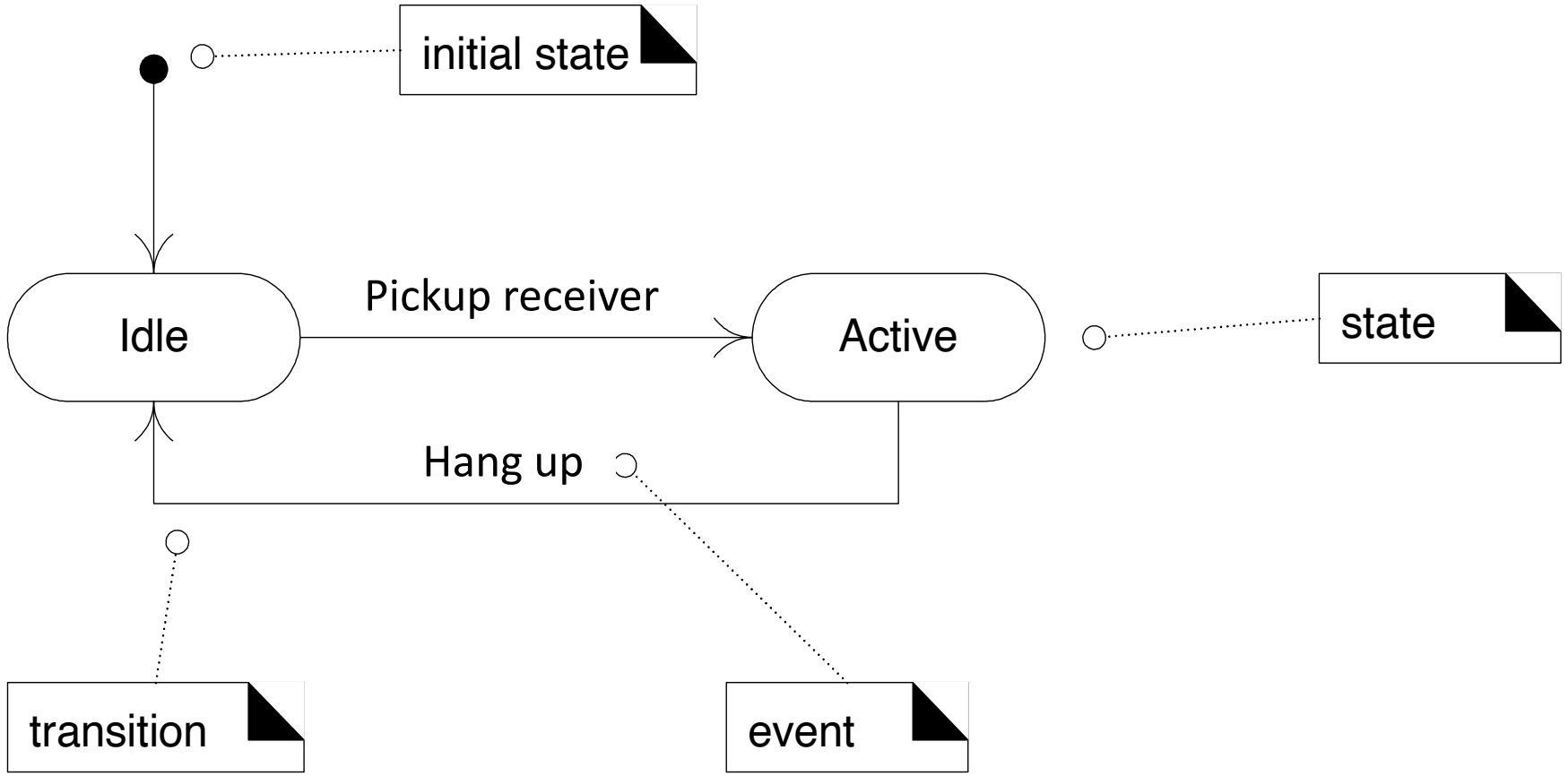
State Machines

Definition: A state machine is a behavior model that captures the dynamic behavior of an object in terms of states, events, and state transitions.

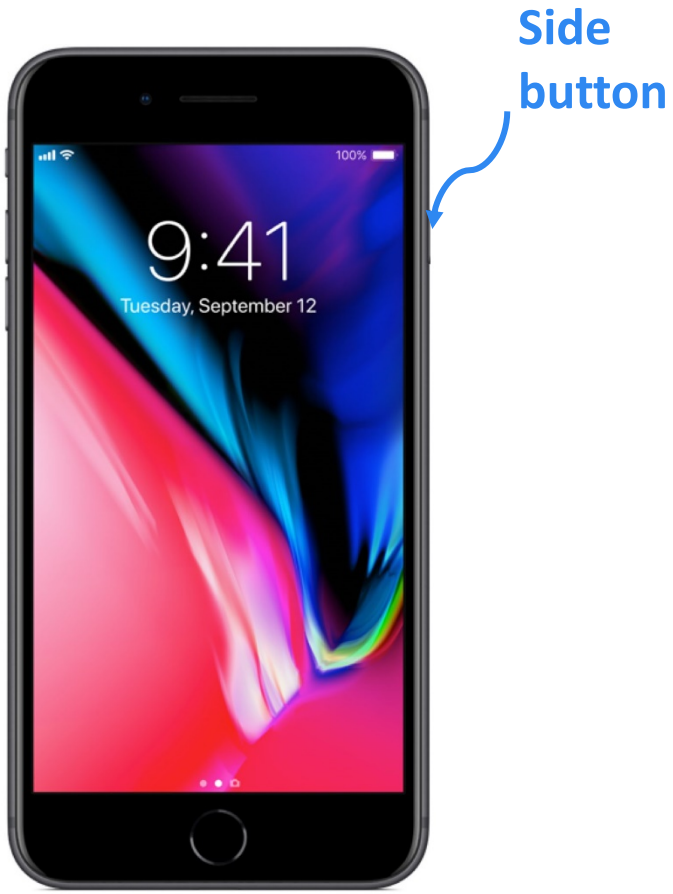
- A *state* is the condition of an object at a moment in time
- An *event* is a significant or noteworthy occurrence that affects the object to change a state
- A transition is a directed relationship between two states such that an event can cause the object to change from the prior state to the subsequent state

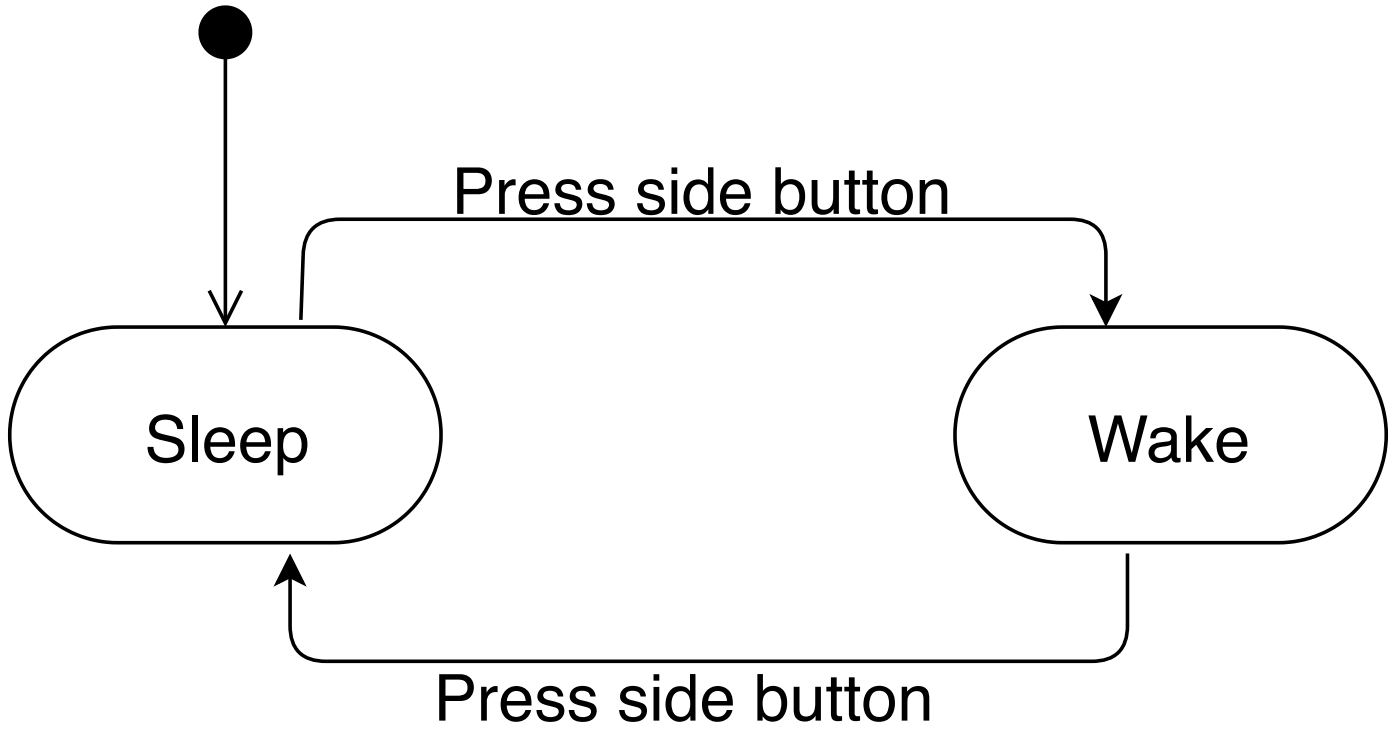
A visual model: UML State Machine Diagram

UML State Machine Diagram



Example: Partial State Machine Diagram for an iPhone





How to Apply State Machine Diagrams? (1)

Determines the behavior of an object

- **State-dependent object**:
  - Reacts differently to events depending on the object's state
- **State-independent object**:
  - For all events of interest, an object always reacts to the event the same way
- **State-independent w.r.t. an event**:
  - Always responds to event the same way

How to Apply State Machine Diagrams? (2)

Guideline 1: Consider state machines for *state-dependent objects* with complex behavior.

- Model behavior of complex reactive objects

Guideline 2: Complex state-dependent classes are *less* common for business information systems, and *more* common in communications and control applications.

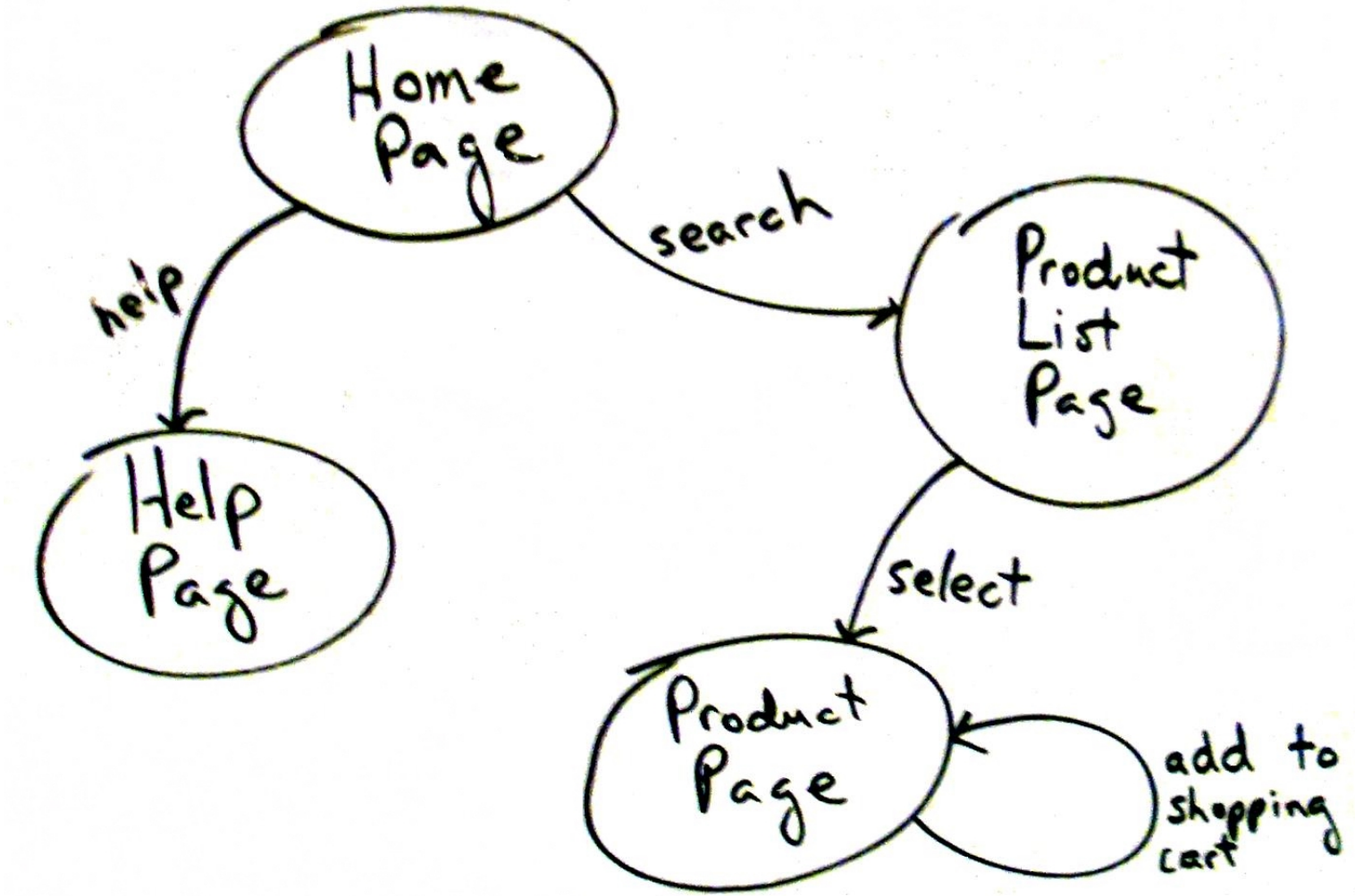
Example: Complex Reactive Objects

- Physical device controllers
- Transactions and related Business Objects
- Role Mutators (objects that change role)

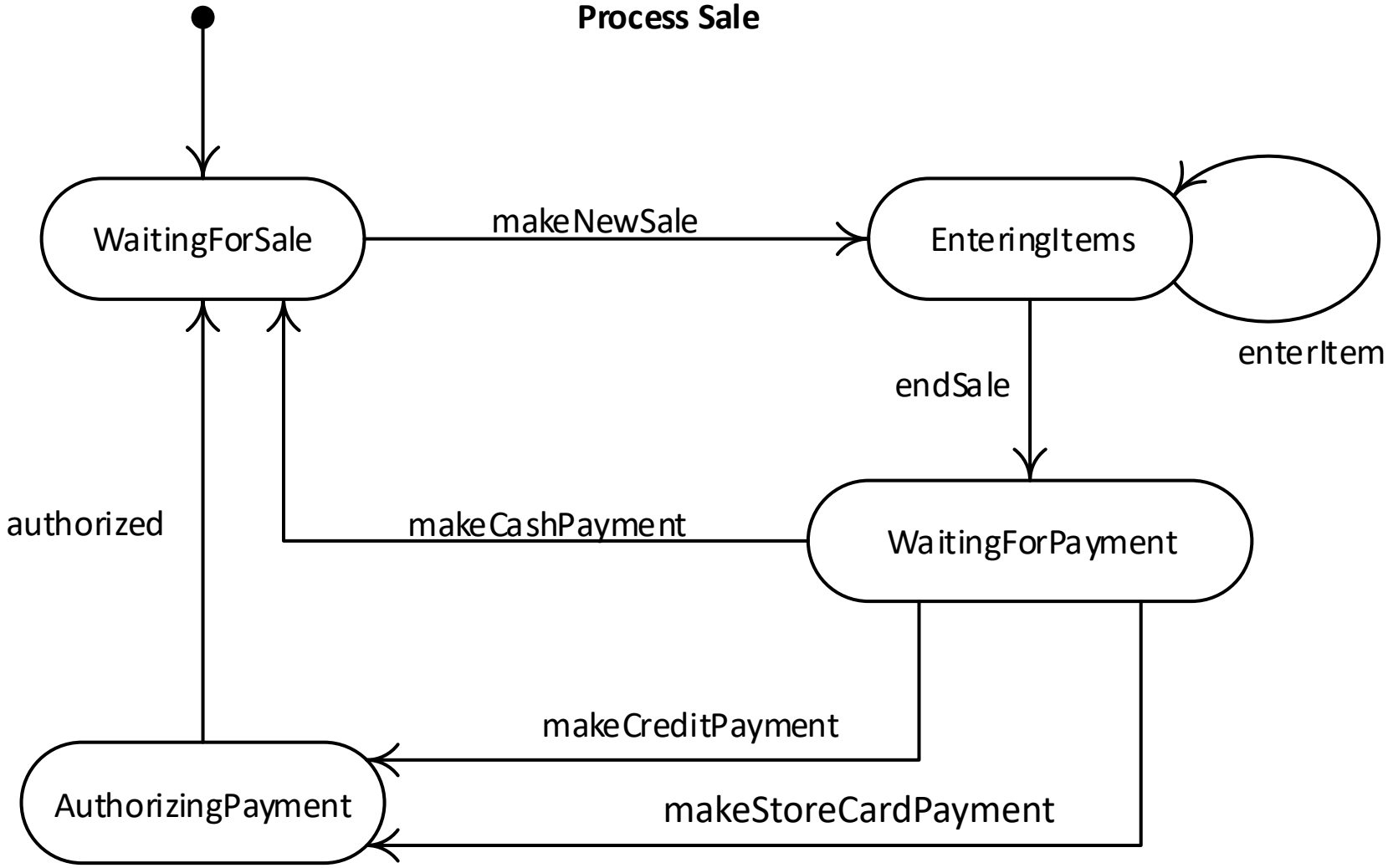
Example: Protocols and Legal Sequences

- Communications Protocols
- UI Page/Window Flow, Navigation, or Session
- Use Case Operation Sequencing

Example: Web Page Navigation



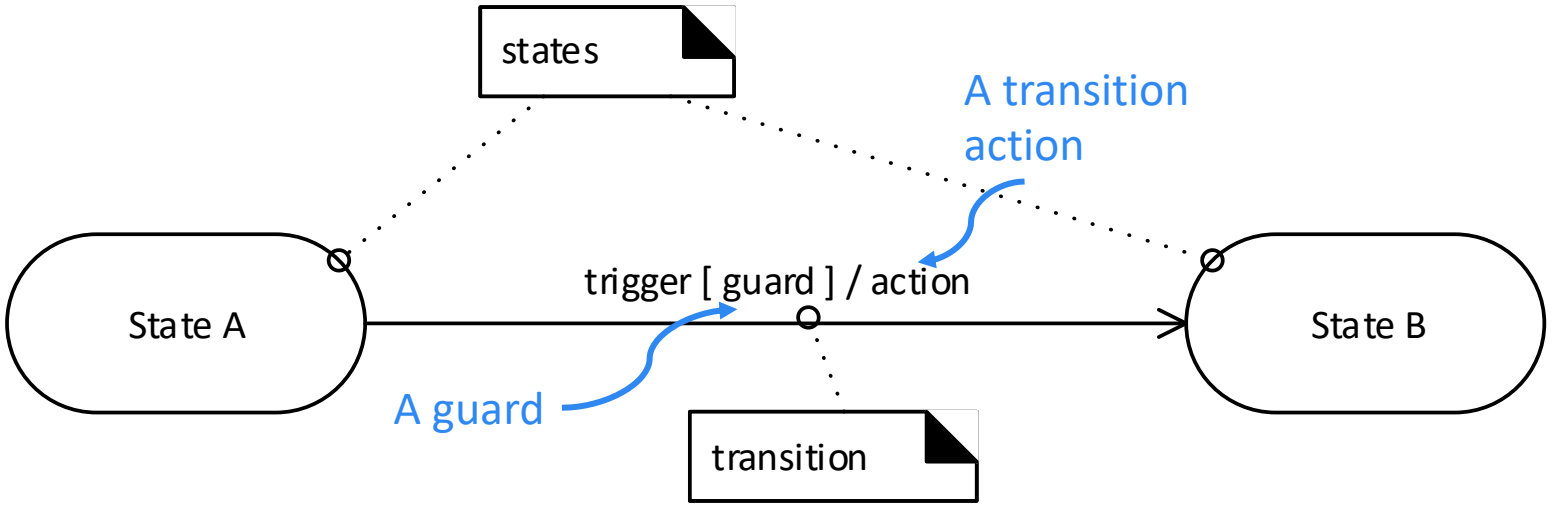
Example: Process Sale Operation Sequencing



Which of the following object(s) is worth to have a state machine model?

Transition Actions and Guards

- **A transition action** is an action (an object does something) when a transition happens
  - In a software implementation, this may represent the invocation of a method of the class of the state machine diagram.
- **A guard is** a *pre-condition* to a transition, i.e., a transition doesn't happen if a guard condition is not true.



When object is in *State A*:

if ***trigger*** event occurs and ***guard*** is true

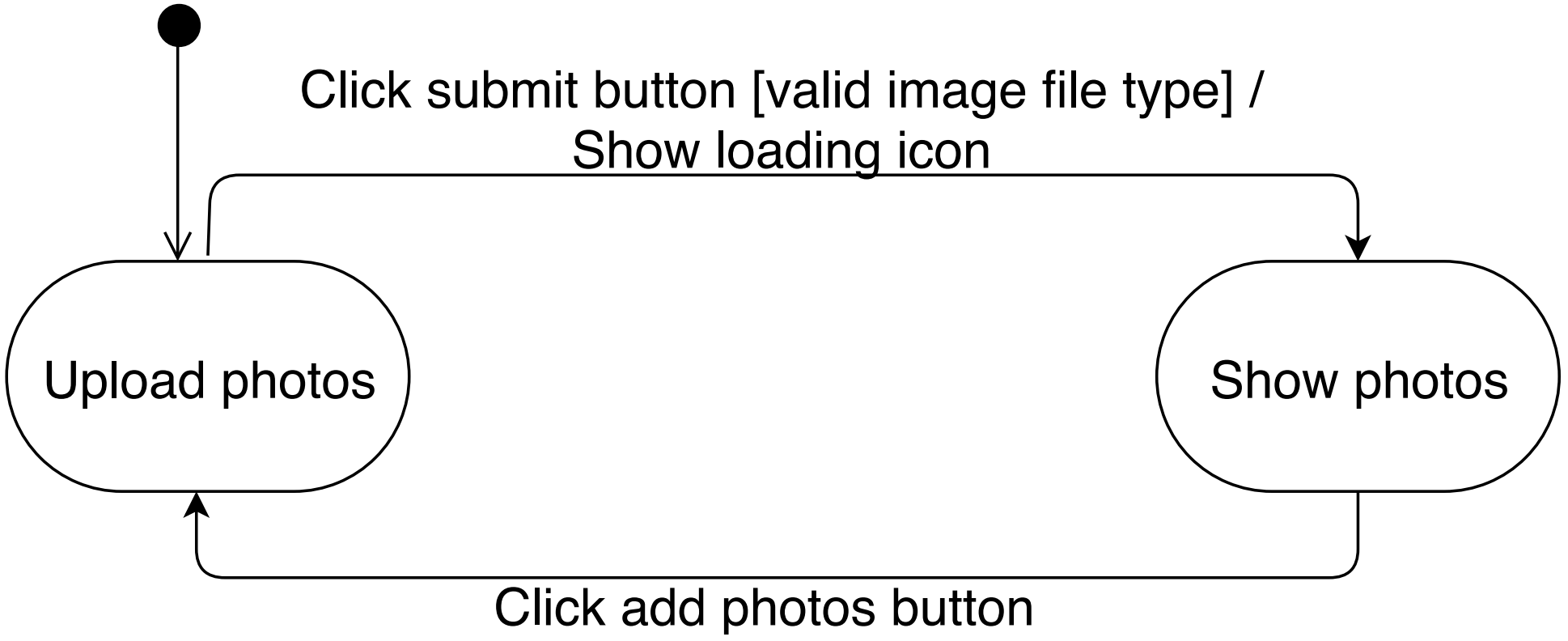
then

perform the behaviour ***action*** and

transition object to *State B*.

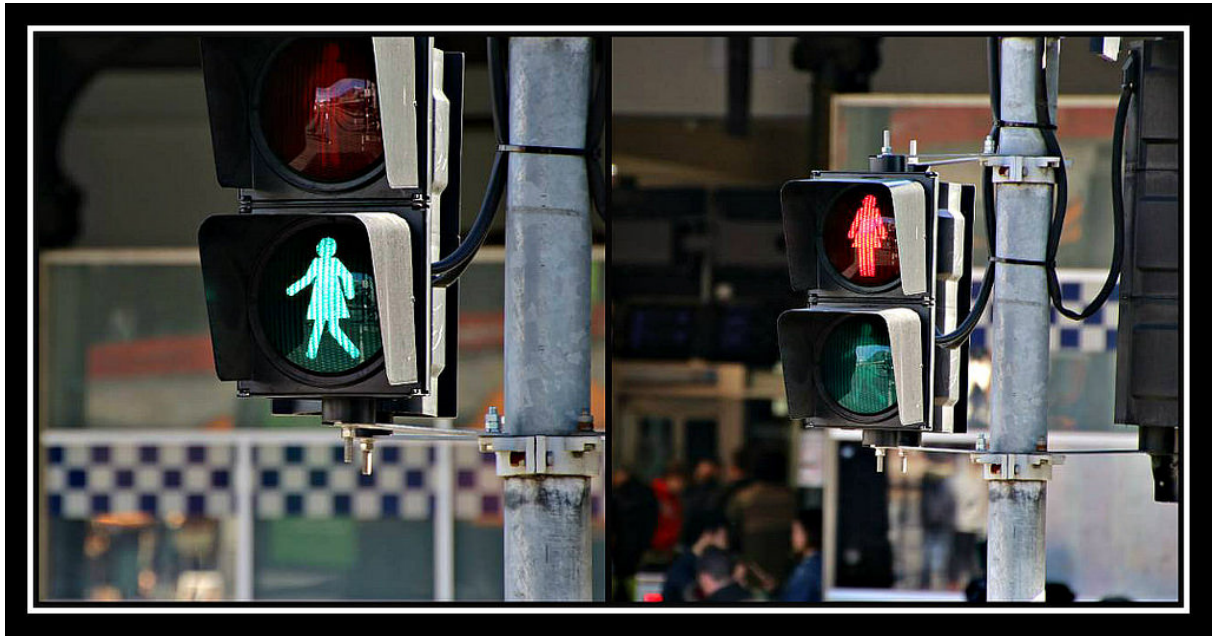
Example: Transition Action and Guard

Photography website



Exercise: Pedestrian Crossing Light

- The **crossing** light starts with the “Red Standing” light
- If the crossing button is pressed when the **traffic** light is “Red”, the **crossing** light becomes “Green Walking” and a timer starts
- When a timer reaches 30 seconds, the **crossing** light becomes “Flashing Red Standing”
- When a timer reaches 60 seconds, the crossing light becomes “Red Standing”

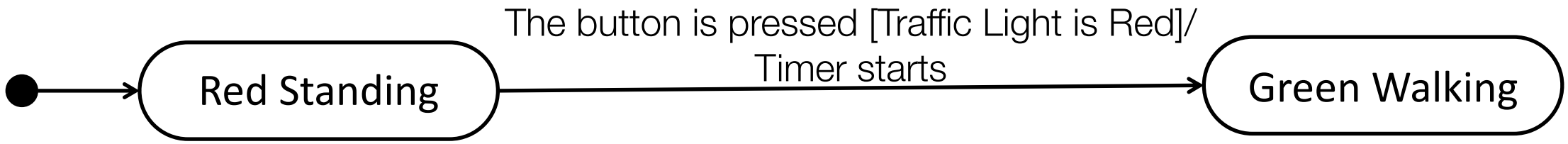


Example: Pedestrian Crossing Light

The **crossing** light starts with the “Red Standing” light

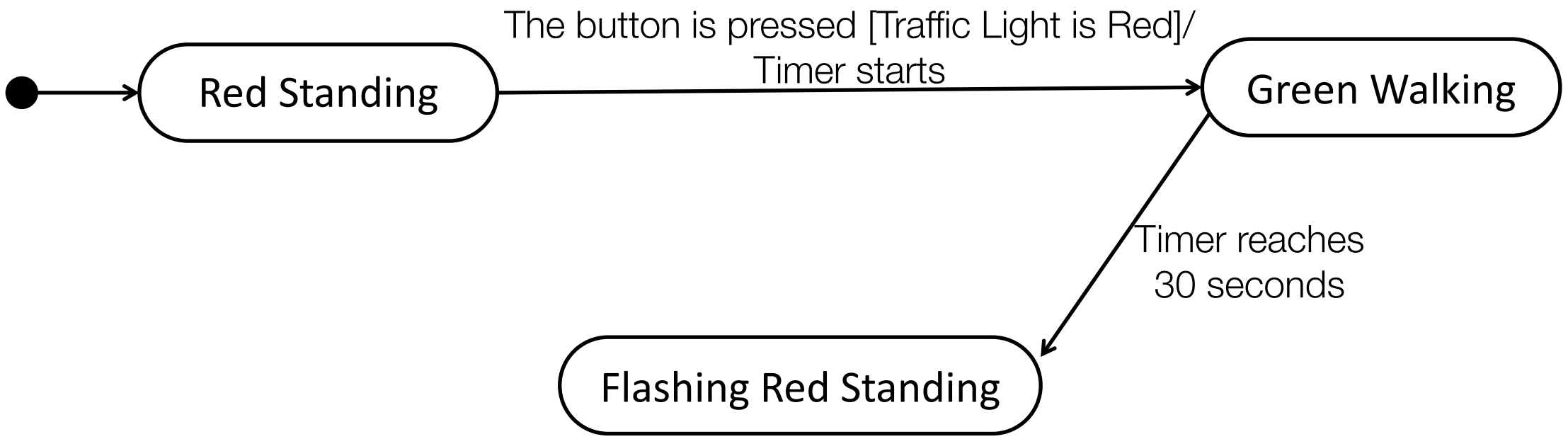
Example: Pedestrian Crossing Light

If the crossing button is pressed when the **traffic** light is “Red”, the **crossing** light becomes “Green Walking” and a timer starts



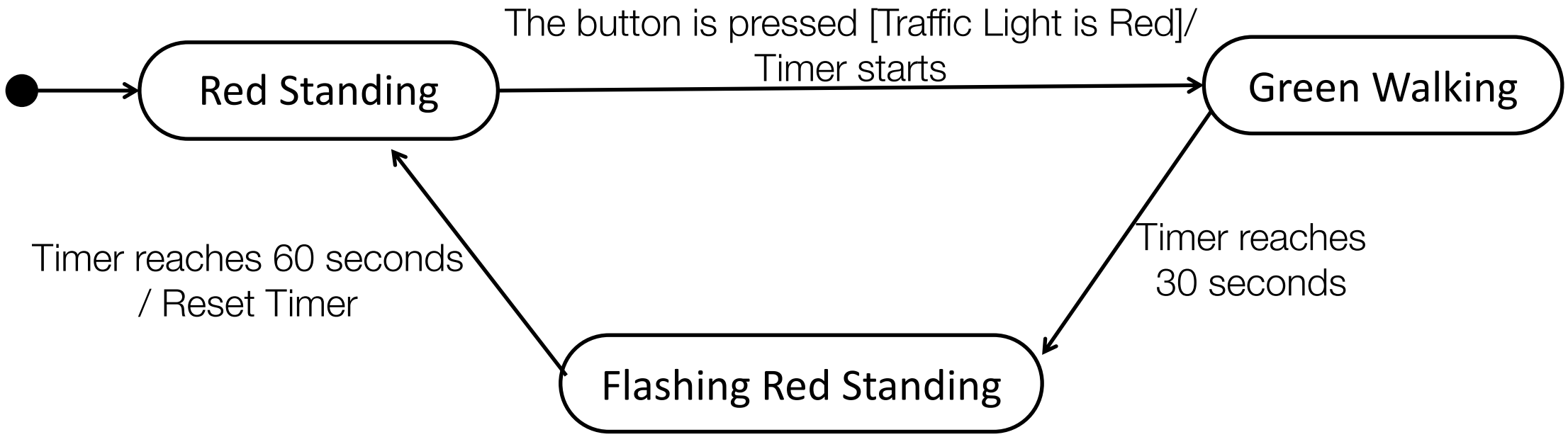
Example: Pedestrian Crossing Light

When a timer reaches 30 seconds, the **crossing** light becomes “Flashing Red Standing”



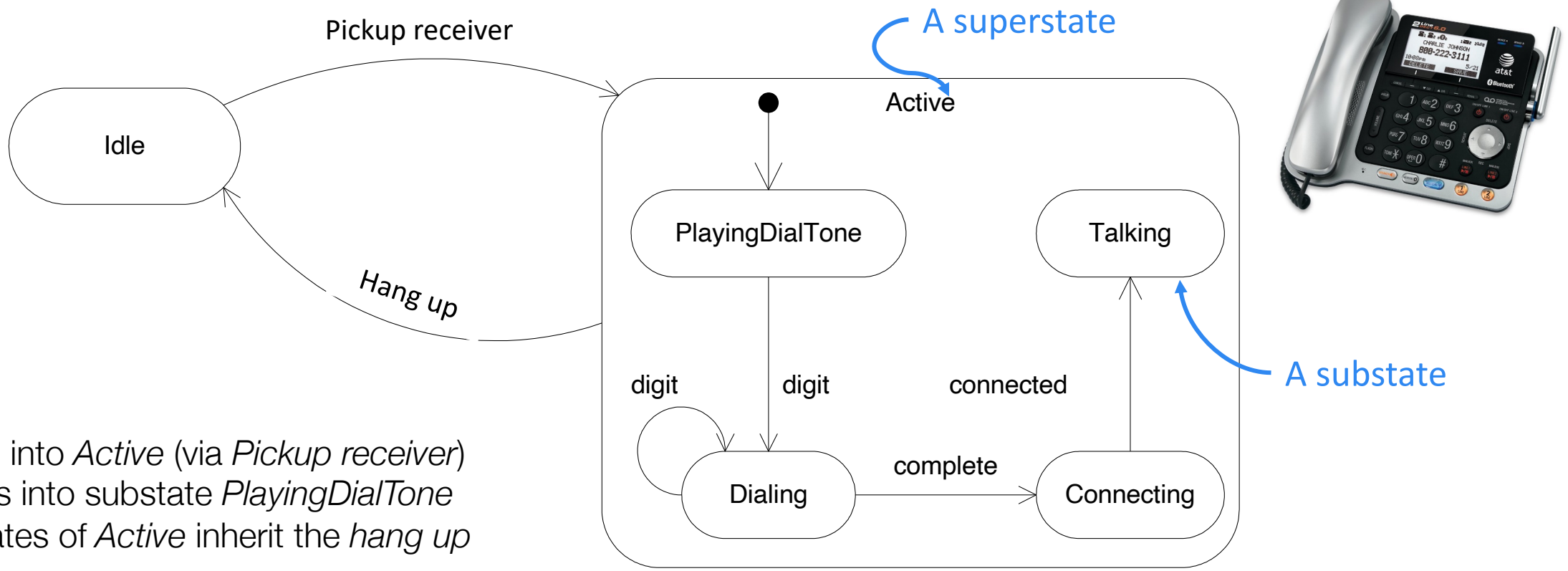
Example: Pedestrian Crossing Light

When a timer reaches 60 seconds, the crossing light becomes “Red Standing”



Nested States

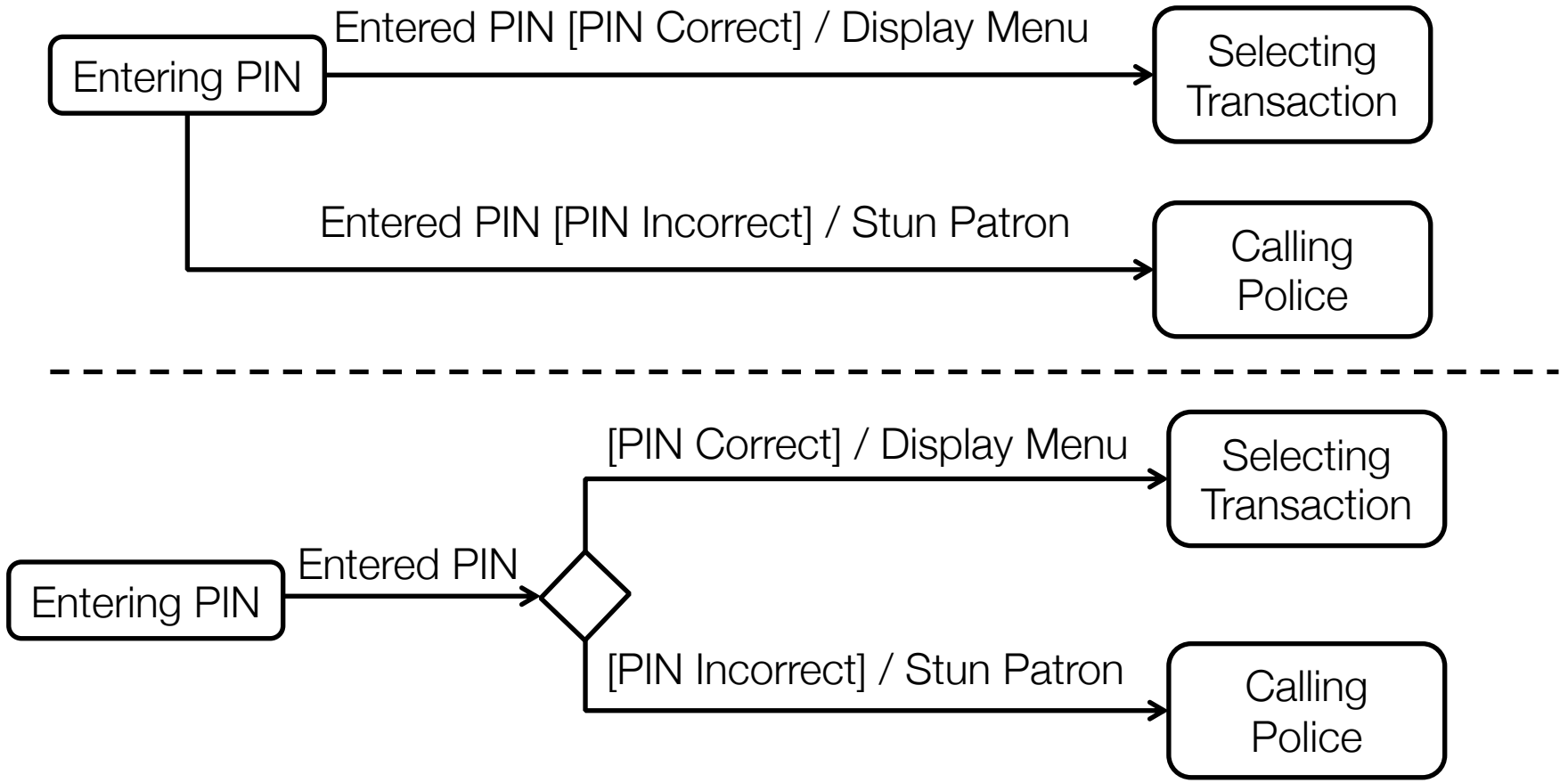
A state allows nesting to contain *substates;* a substate inherits the transitions of its superstate (the enclosing state)



- Transition into *Active* (via *Pickup receiver*) transitions into substate *PlayingDialTone*
- All substates of *Active* inherit the *hang up* transition.

Choice Pseudostate

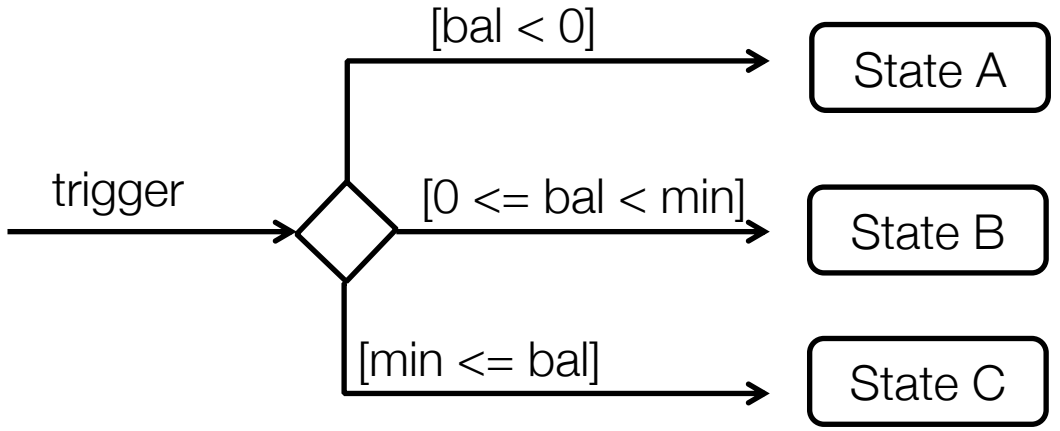
Choice pseudostate realizes a dynamic conditional branch. It evaluates the guards of the triggers of its outgoing transitions to select only one outgoing transition.



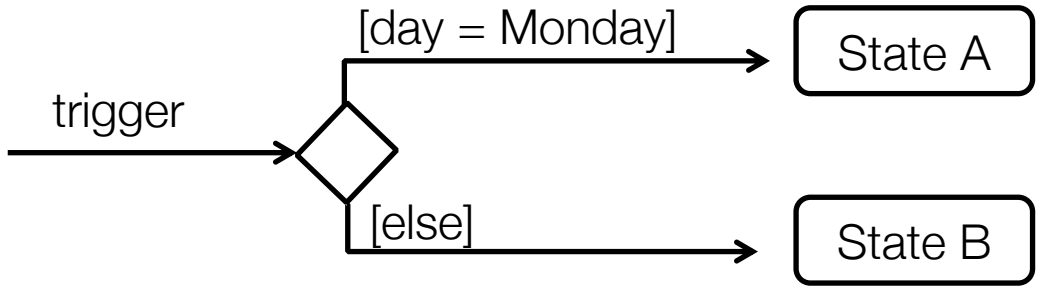
Choice Pseudostate (Continue)

Choice Pseudostate can

- Have two or more outgoing transitions



- Use predefined [else] guard
  - [else] outgoing transition chosen if no other guards are true



Summary & Remarks

- A state machine is dynamic model that illustrates the behavior of an object in terms of states, events, and state transitions
- A state machine should be used if an object has complex behavior that depends on states or conditions
- In the workshop 6 supplementary materials, you will learn how to implement state machines

Week 7 – 9 Lecture :
GoF Design Patterns
by Dr Peter Eze

Lecture Identification

Lecturer: Patanamon Thongtanunam

Semester 2, 2020

These slides include materials from:

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, by Craig Larman, Pearson Education Inc., 2005.