BEHAVIOUR MODELING WITH STATE MACHINE AND ACTIVITY DIAGRAMS

Objectives

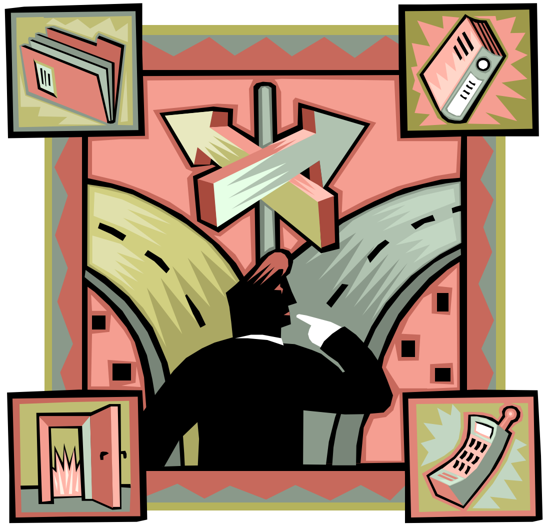
- Describe system behavior and show how to capture it in a model.
- Demonstrate how to read and interpret:
  - A state machine diagram
  - An activity diagram

Agenda

- How to describe behavior?
- Modelling with state machine diagrams
- Modelling with Activity diagrams

Where are we?

- How to describe behavior?
- Modelling with State machine diagrams
- Modelling with Activity diagrams



What Is System Behavior?

- System behavior is **how** a system acts and reacts.
  - It comprises the **actions** and **activities** of a system.

What is a Behavior Model?

- A view of a system that emphasizes the behavior of the system as a whole (as it appears to outside users)
- Uses:
  - Activity Diagrams
  - State machine Diagrams
  - *Sequence Diagrams*
  - *Interaction Overview Diagrams*

- The system/object is regarded as a black box and the functionalities are expressed from a user's perspective

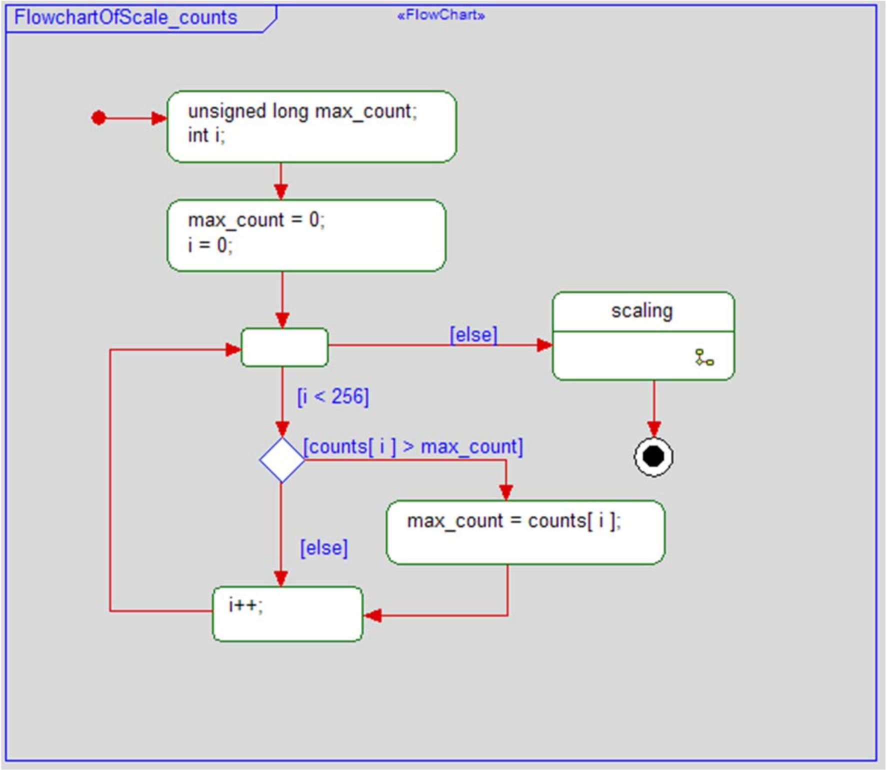
- Behavior modeling during development:
  - In the analysis phases:
    - The model must capture the requirements of the system, not the implementation solution
  - In the design & implementation phases:
    - The model must capture the implementation solution.

Types of behavior

- Behavior can be *simple:*
  - Simple behavior does not depend on the object's history.
- Behavior can be *continuous:*
  - Continuous behavior depends on the object's history but in a smooth, continuous fashion.
- Behavior can be *state-driven:*
  - State-driven behavior means that the object's behavior can be divided into disjoint sets.

Simple behavior

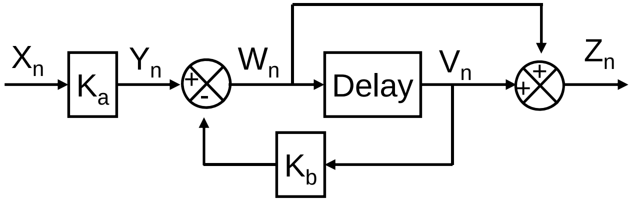
- Simple behavior is not affected by the object's history:
  - cos(x)
  - getTemperature( )
  - setVoltage(v)
  - Max(a,b)
  - $\int_a^b e^{-x^2} dx$



An activity diagram can be used to represent simple behaviour.

Continuous behavior

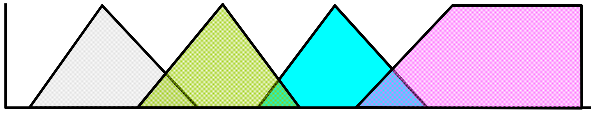
- Object’s behavior depends on history in a continuous way
  - PID control loop:



  - Digital filter:

$$f_j = \frac{d_j + d_{j-1} + d_{j-2} + d_{j-3}}{4}$$

  - Fuzzy logic:
    - Uses partial set membership to compute a smooth, continuous output



UML is not very good at describing continuous behavior.

State behavior

- Useful when an object
  - Exhibits discontinuous modes of behavior
  - Is reactive – waits for and responds to incoming events

- For example a Light can be :
  - Off

  - On

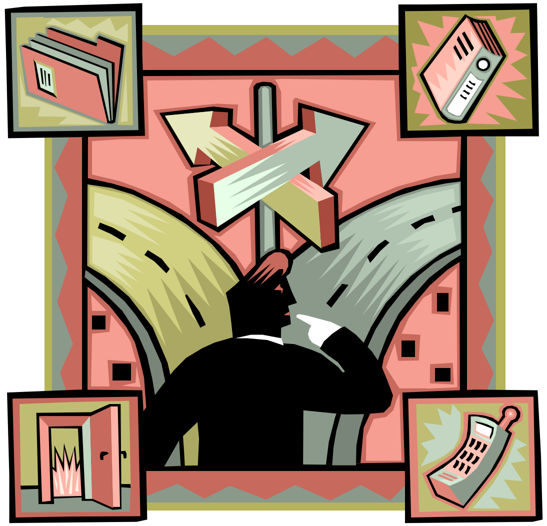
  - Flashing

- A light can be characterized as having 3 states – On, Off & Flashing.
  - The system can only reside in one of these states at a given time.

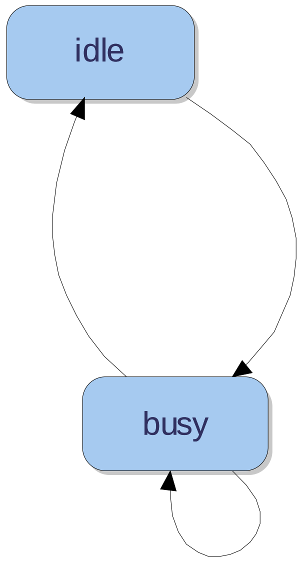
A state machine can be used to represent state behaviour.

Where are we?

- How to describe behavior?
- Introduction to State machine diagrams
- Introduction to Activity diagrams

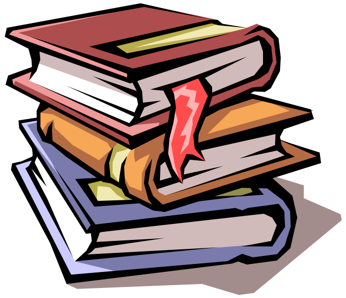


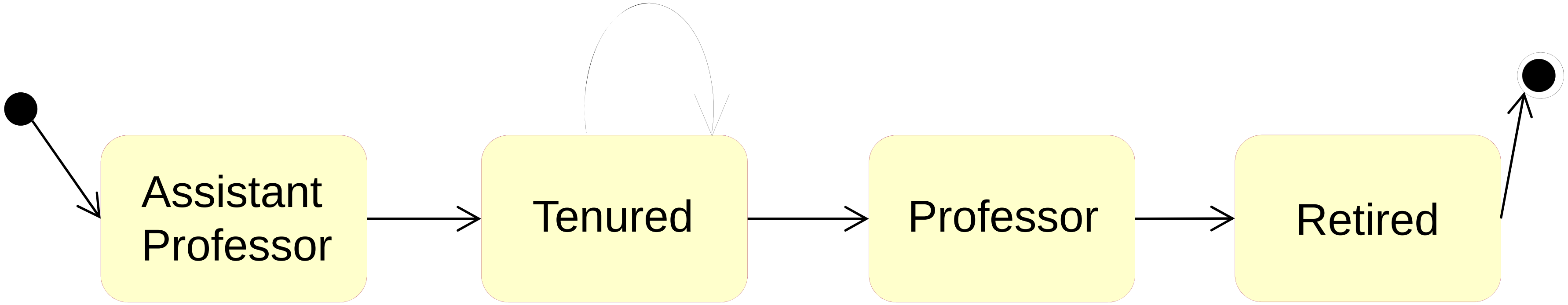
Quick picture of a state machine



Example: Professor

- There are a sequence of events before an instructor becomes a University professor.
  - Assistant professor (achieves tenure by producing a number of quality publications)
  - Tenure/Associate professor
  - Professor (based on seniority)





What is a State Machine?

- A state machine models **dynamic behavior** of a element (system, component, class, operation, etc.)
- State machines are ideal to describe **reactive, event-driven behavior.**
- It specifies
  - the sequence of **states** in which an object can exist in.
    - a finite number of states
  - The **events and conditions** that cause **transitions** between those states.
  - The **actions** that take place when those states are reached
- States can be
  - Composite
  - Orthogonal

Why use state machines?

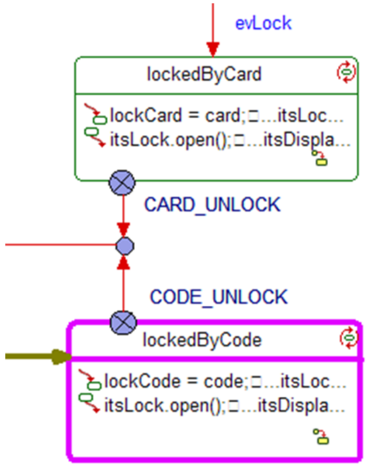
- State machines are **executable**
  - Because state machines are formally defined, they form executable models.

- State machines can be **visualized** graphically.

- State machines can be **animated** - their dynamic behavior shown graphically:
  - Standard debugging can be done, such as setting breakpoints, step through, step into, and so on.

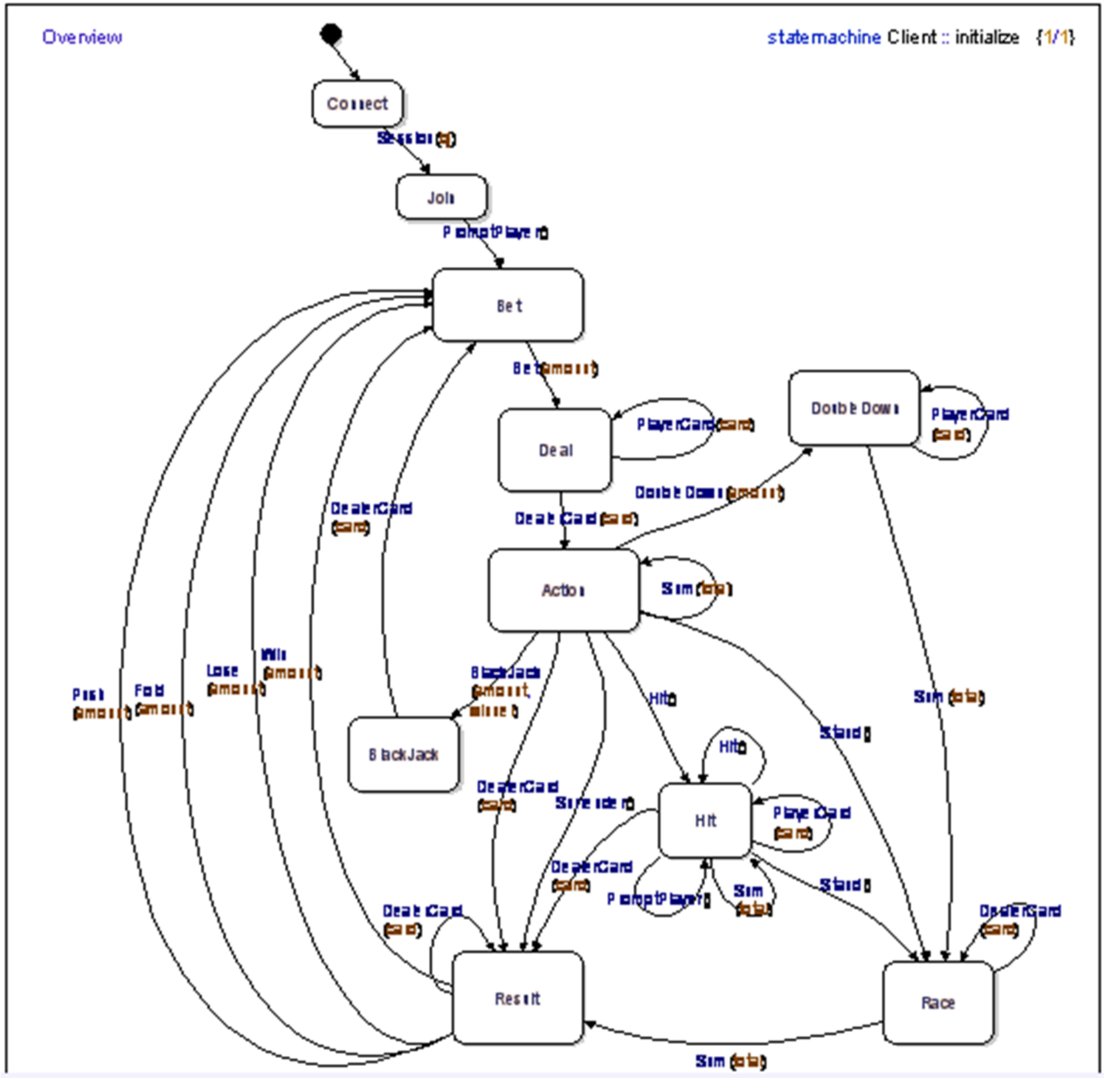
- You can focus on the abstract behavior:
  - Was the door locked with a card or a code?
  - Rather than, is some variable 0 or 1?

- State Machines provide for easy testing.

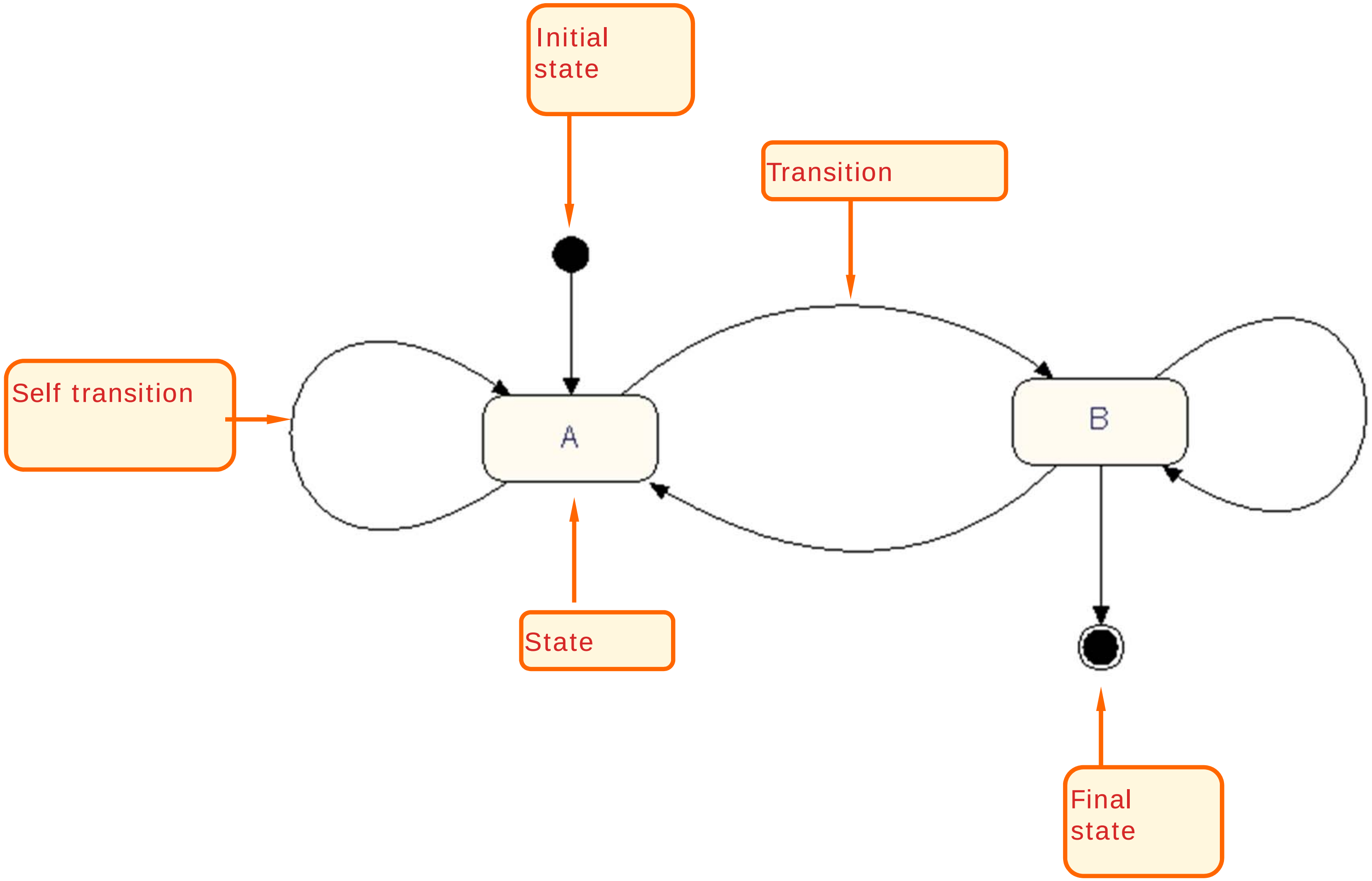

evLock
lockedByCard
lockCard = card;⊐...itsLoc...
itsLock.open();⊐...itsDispla...
CARD_UNLOCK
CODE_UNLOCK
lockedByCode
lockCode = code;⊐...itsLoc...
itsLock.open();⊐...itsDispla...

What is a State Machine Diagram?

- A state machine diagram **visualizes** a state machine
  - State symbol
  - Transition lines between the states



State Machine Diagram – Basic Syntax



States / transitions / events / actions

- What is a state?

A **state** is a distinguishable, disjoint, orthogonal condition of existence of an object that persists for a significant period of time.

- What is a transition?

A **transition** is a response to an event of interest moving the object from a state to a state.

- What is an event?

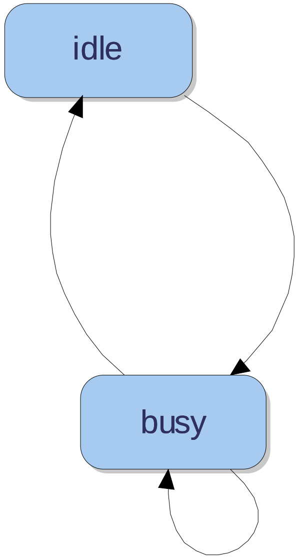
An **event** is an occurrence of a stimulus that can trigger a state transition.

- What is an action?

An **action** is a run-to-completion behavior. The object does not accept or process any new events until the actions associated with the current event are complete.

What is a state? (details)

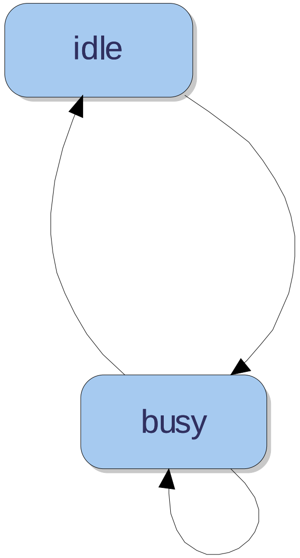
- Represents the mode of operation of an activity.

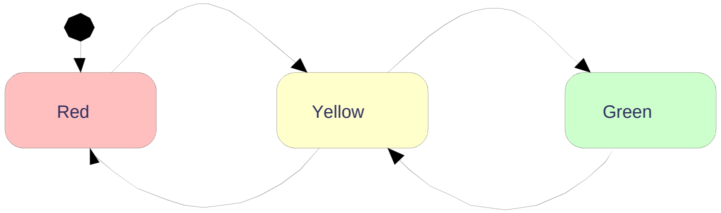


What is a state? (details)

- The state machine is either in a **stable state** or is **executing the actions** specified on a transition

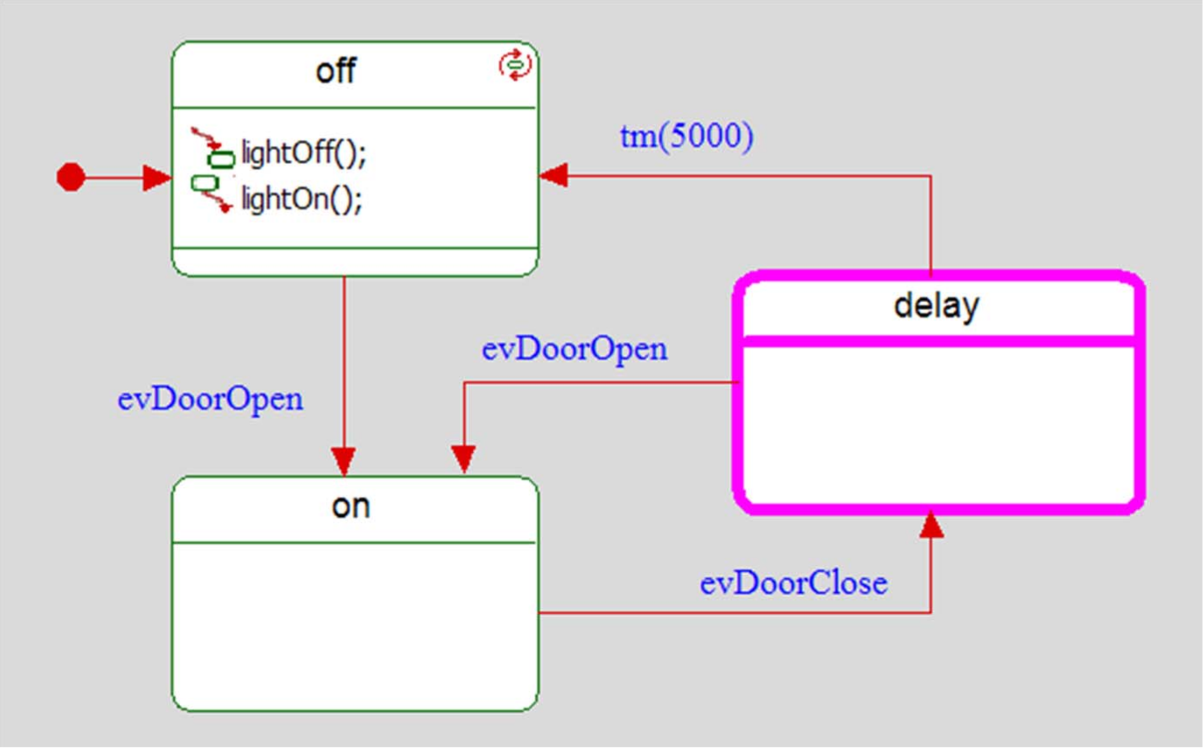
- A state is also a memory:
  - Use states to **memorize a sequence** of received input
  - **States** may be used **instead of variables / attributes** – if it improves the state machine structure, readability or maintainability





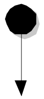
What is a state? (details)

- The system must always be in *exactly one* state at a given level of abstraction.
  - The object must be in either *off* or *on* or *delay* – it cannot be in more than one or none.

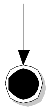


What is a state? (details)

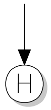
- Pseudostates
  - Initial

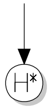


  - Final (return)



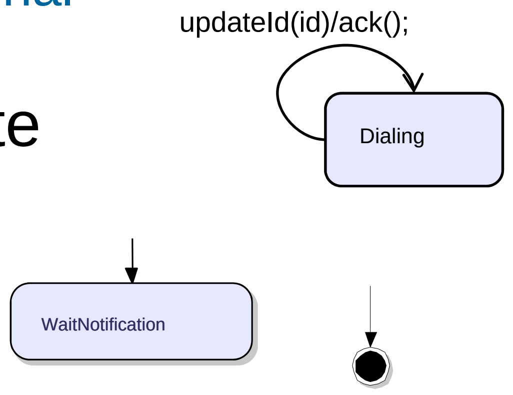
  - History state
    - Shallow
    - Deep





What is a transition? (details)

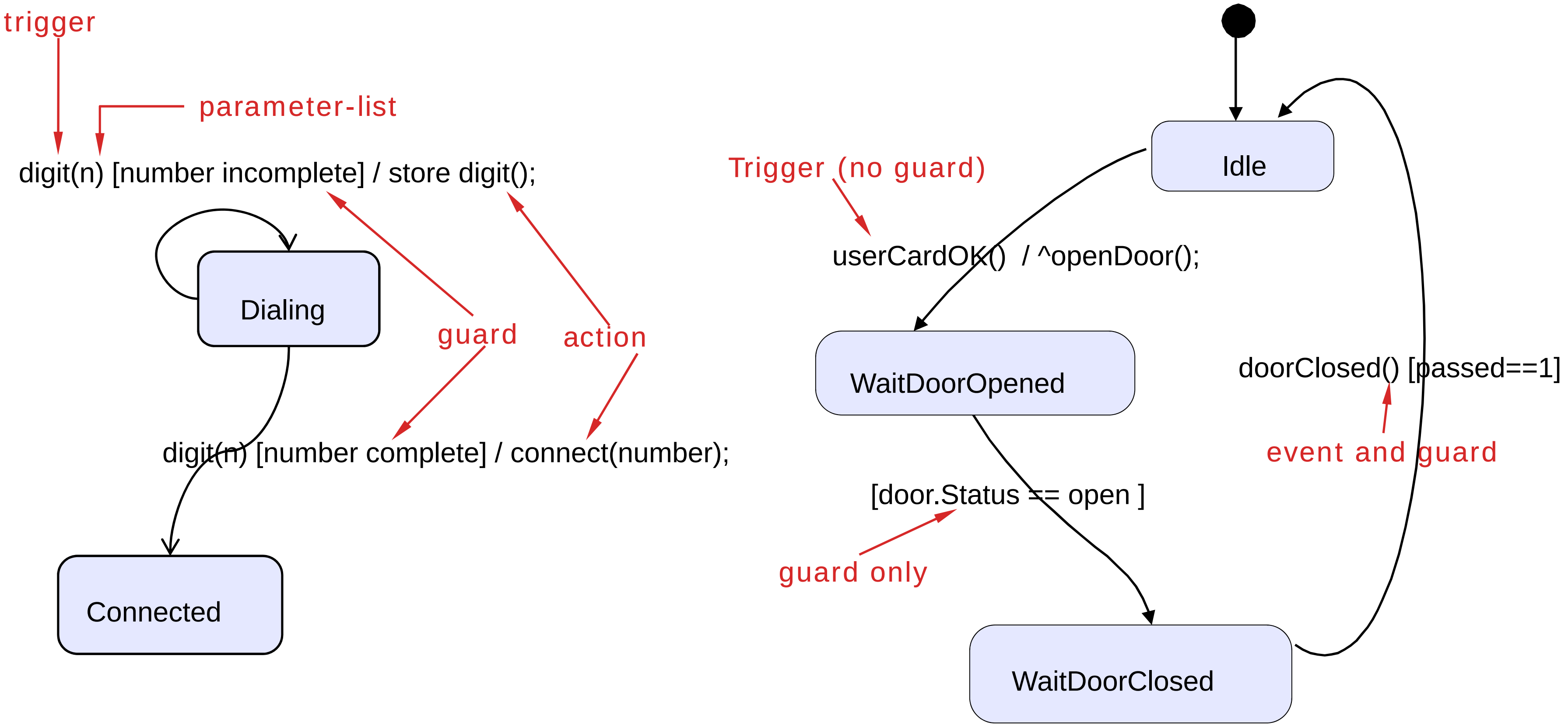
- A transition is
  - the journey from **one state to another**
  - the **placeholder** for behavior
  - **activated** when a **event that matches the trigger** is selected from a event pool (e.g. signal queue)
- Transitions may **end in the original** state
- A transition **ends in**
  - a state
  - Final state



- A Transition may be **split** using
  - a junction

What is a transition? (details)

- Transition syntax
  - trigger [guard] / action list

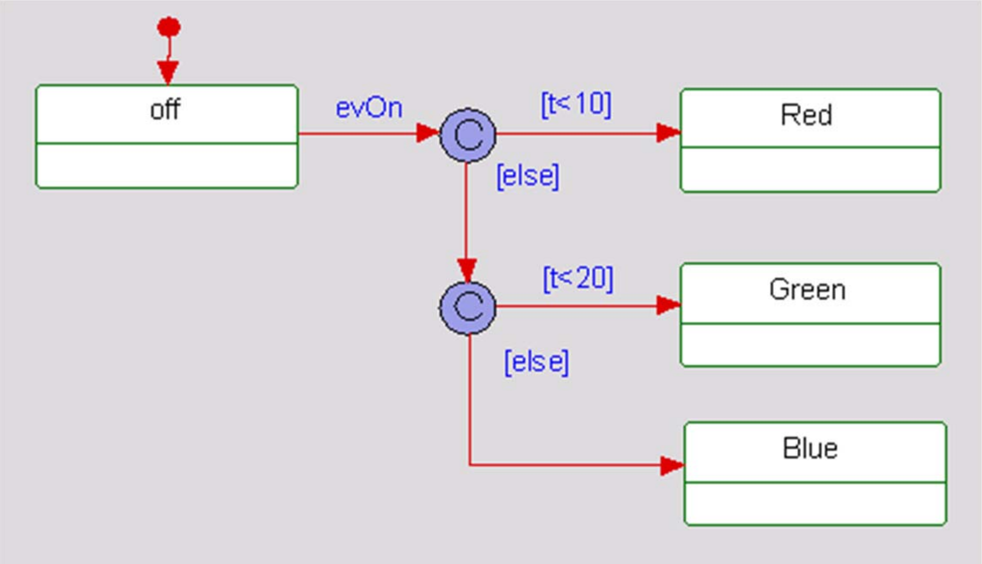


What is a transition? (details)

What are guards?

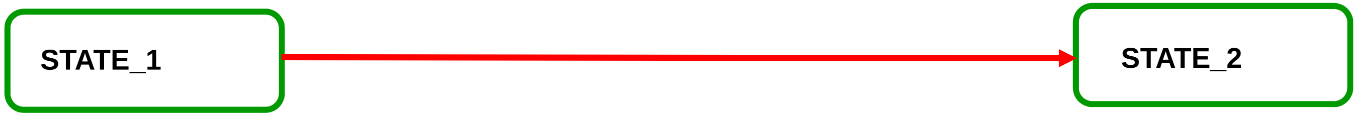
- A guard is some condition that must be met for the transition to be taken.
- Guards are evaluated *prior* to the execution of any action.
- Guards can be:
  - Variable range specification, for example: [cost<50]
  - Concurrent state machine is in some state [IS_IN(fault)]

If two guards are likely to be true at the same time, then it is a good idea to chain condition connectors, so that the order in which the guards are tested is known.

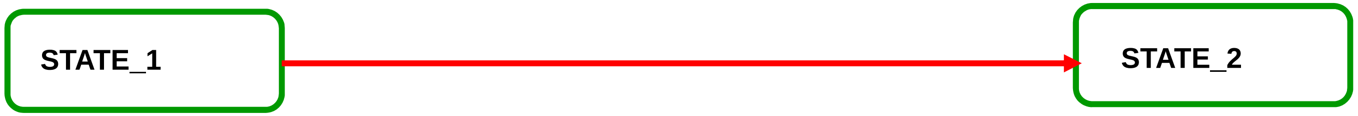


Problem 3

- Define the proper transition label based on the following criteria:
  1. The variable *failure_msg* is set to true and *x* is set to 5/y when the event abort occurs.



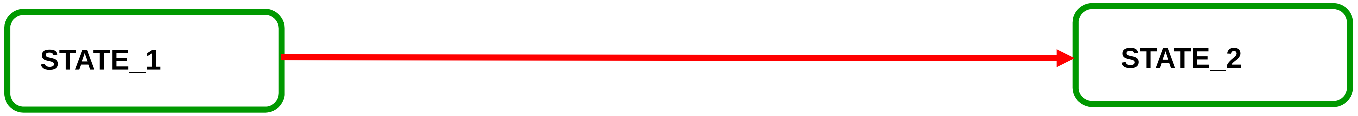
  2. Five time units after STATE_1 is entered, *system_status* is set to *level1*.



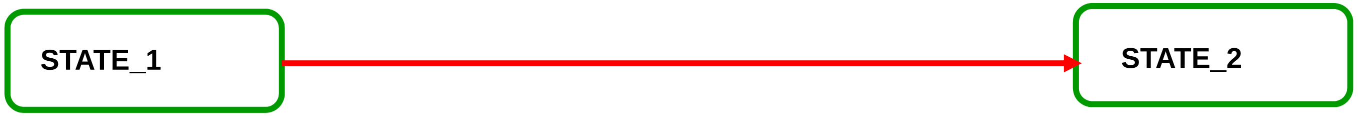
Problem 3 (continued)

- Define the proper transition label based on the following criteria:

3. If the event start_testing occurs and power is true, the variable light_cmd is set to true.

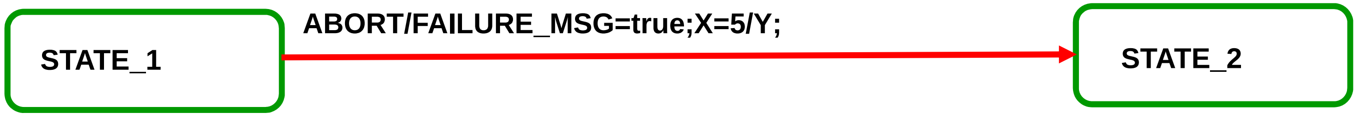


4. When the value of X changes, then set Z to the absolute value of X.

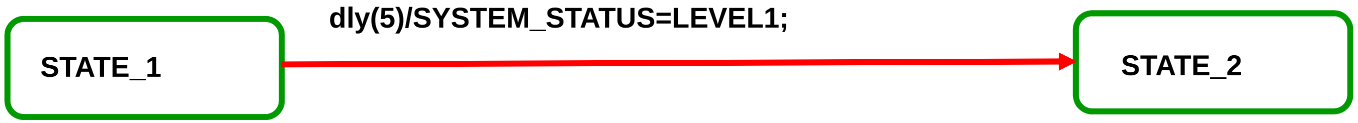


Problem 3 - Solution

- Define the proper transition label based upon the following criteria:
  1. The variable *failure_msg* is set to true and *x* is set to 5/y when the event abort occurs.



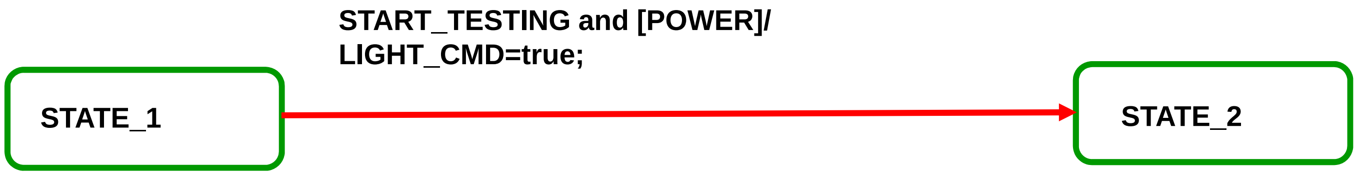
  2. Five time units after STATE_1 is entered, *system_status* is set to *level1*.



Problem 3 - Solution (continued)

- Define the proper transition label based on the following criteria:

3. If the event start_testing occurs and power is true, the variable light_cmd is set to true.

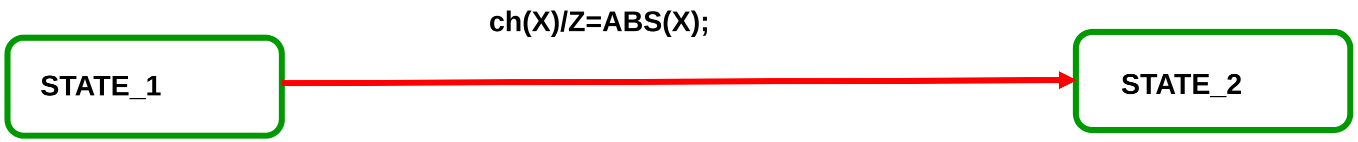


START_TESTING and [POWER]/
LIGHT_CMD=true;

STATE_1

STATE_2

4. When the value of X changes, then set Z to the absolute value of X.



ch(X)/Z=ABS(X);

STATE_1

STATE_2

What is a transition? (details)

How are transitions handled?

- If an object is in a state S that responds to an event evX, then it acts upon that event.
  - It transitions to the specified state, if the event triggers a transition and the guard (if any) on that transition evaluates to TRUE.
  - It executes any actions associated with that transition.
- Events are quietly discarded if:
  - A transition is triggered, but the transition's guard evaluates to FALSE.
  - A transition to a conditional pseudostate is triggered, but all exiting transition guards evaluate to FALSE.
  - The event does not explicitly trigger a transition or reaction.

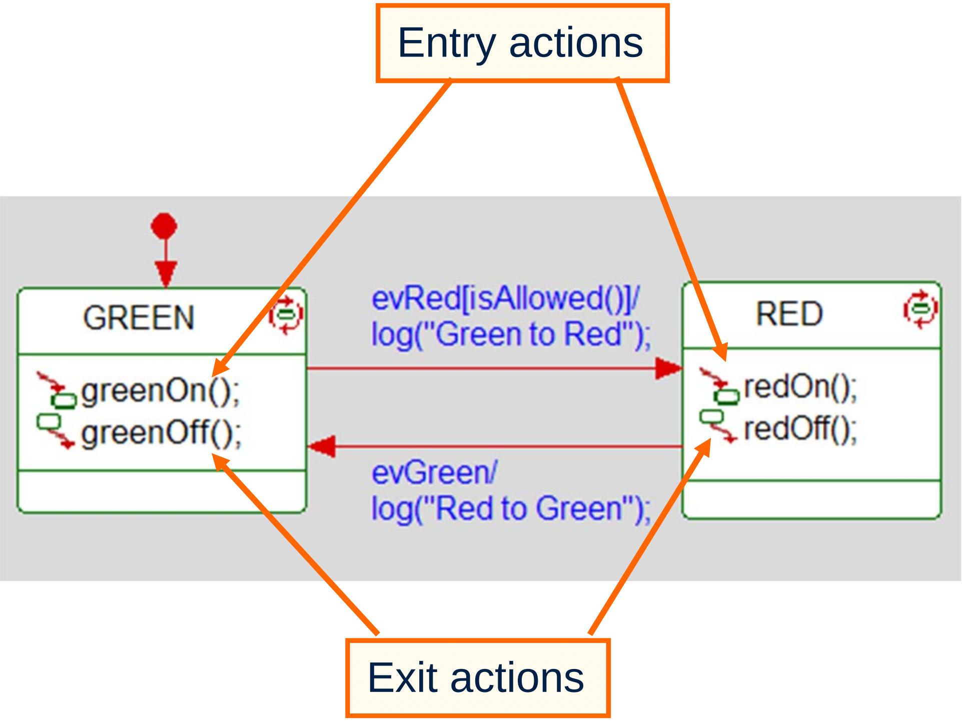
What is an action? (details)

- Actions are run to completion:
  - Normally actions take an insignificant amount of time to perform

- They may occur when:
  - A transition is taken
  - A *reaction* occurs
  - A state is entered
  - A state is exited

What is an action? (details)

Entry / exit actions



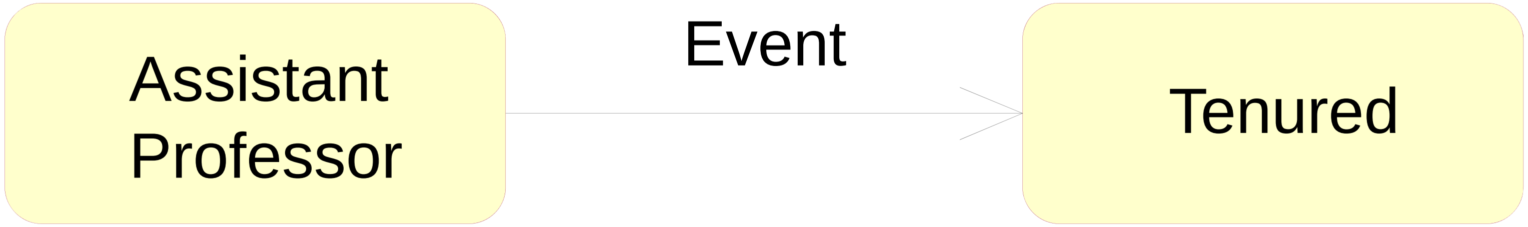
Entry actions

Exit actions

Note the order of execution of the actions and that the guard gets checked before any actions are taken.

What is an Event? (details)

- An event is an occurrence of a stimulus that can trigger a state transition.
  - Example: Successful publication of numerous papers

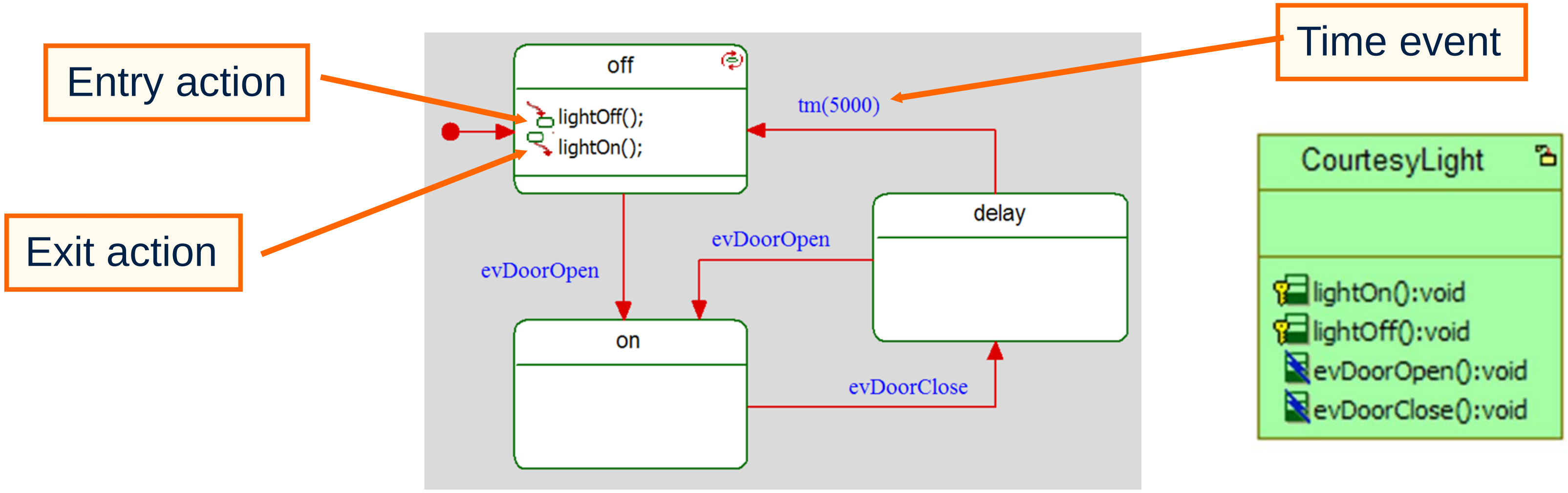


What is an Event? (details)

- UML defines 4 kinds of events:
  - Signal Event
    - Asynchronous signal received for example, evOn, evOff
  - Call Event
    - Operation call received, for example, opCall(a,b,c)
    - This is known as a *Triggered Operation* in Rational Rhapsody
  - Change Event
    - Change in value occurred
  - Time Event
    - Absolute time arrived
    - Relative time elapsed, for example, tm(PulseWidthTime)

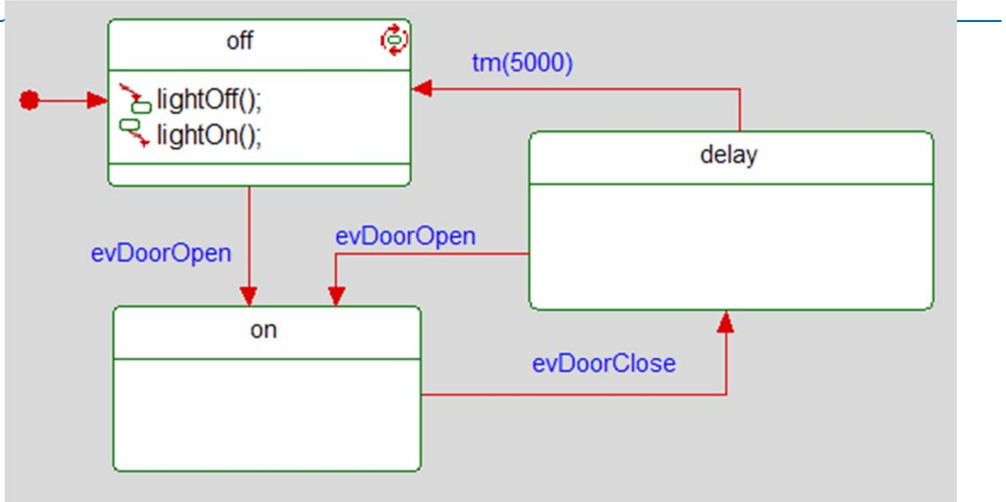
Time event

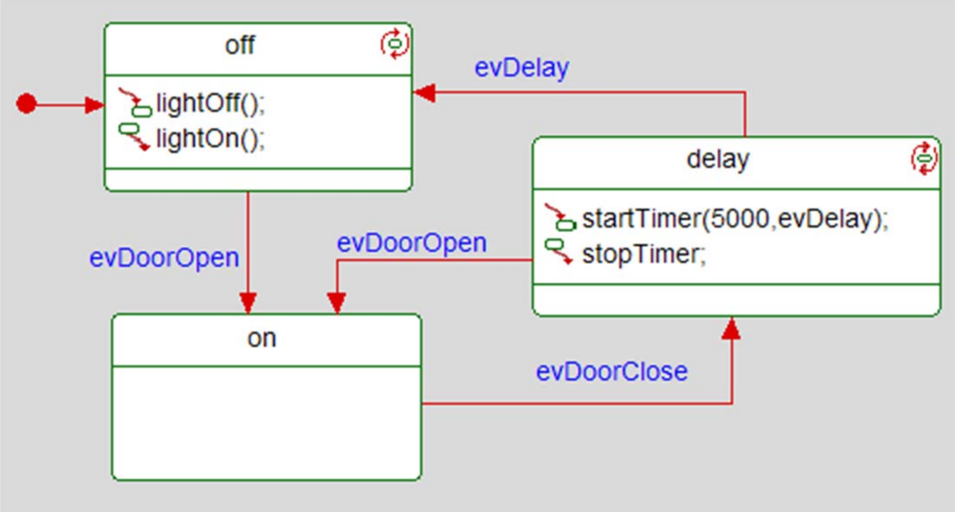
- When an object enters a state, any Timeout from that state is started. When the Timeout expires, the state machine receives the expiration as an event.
- When an object leaves a state, any timeout that was started on entry to that state is cancelled.
- Only one timeout can be used per state; nested states can be used if several timeouts are needed.



tm(delayTime)

- tm(delayTime) is specific to Rational Rhapsody and code is automatically generated to start and stop the timeout.
- This is equivalent to the second state-chart where a timer is started on entering the state and stopped on exiting the state. If the timer expires, then it would send the requested event, for example, evDelay.

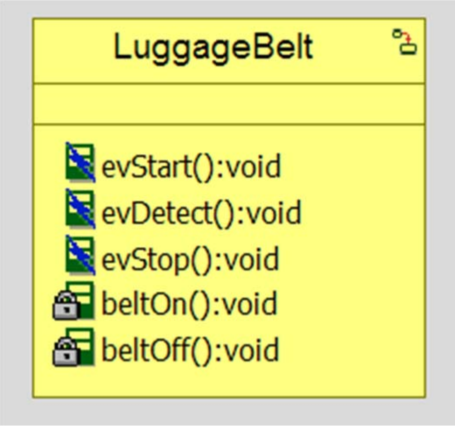




UML actually defines the keyword after(Delay) instead of tm(Delay).

Exercise: luggage belt system

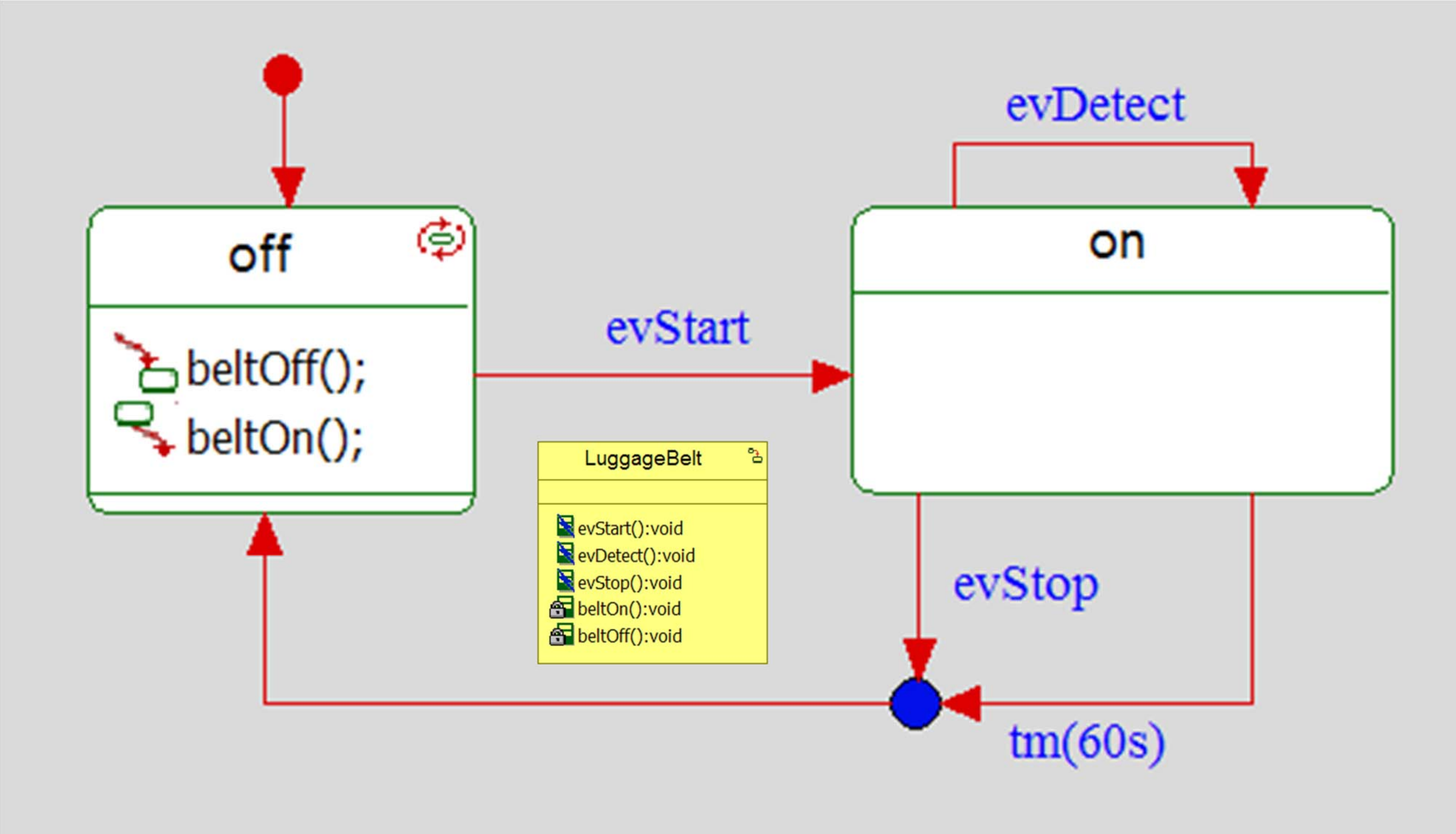
- Draw the state machine for a luggage belt system. The belt is started when the start button is pressed and runs until either the stop button is pressed or until there is no luggage on the belt. This condition is when no luggage has been detected in the previous 60 seconds.



LuggageBelt

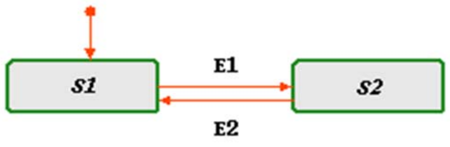
evStart():void
evDetect():void
evStop():void
beltOn():void
beltOff():void

Exercise: Luggage belt system (Solution)

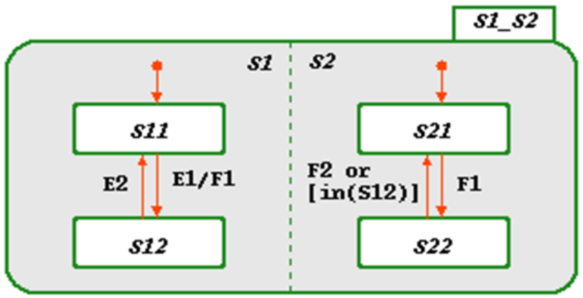


Hierarchy & concurrency in state machines

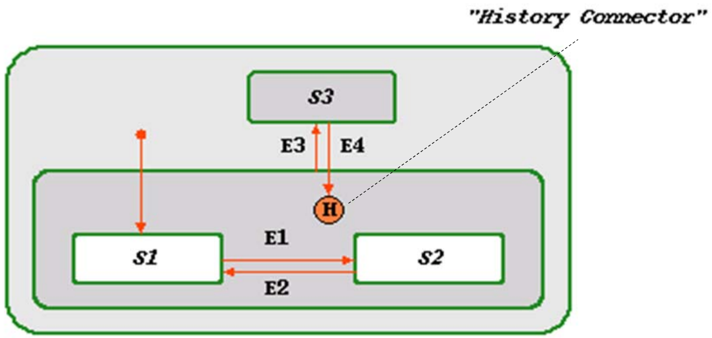
- **Simple State Machine**



- **Concurrent state machine**

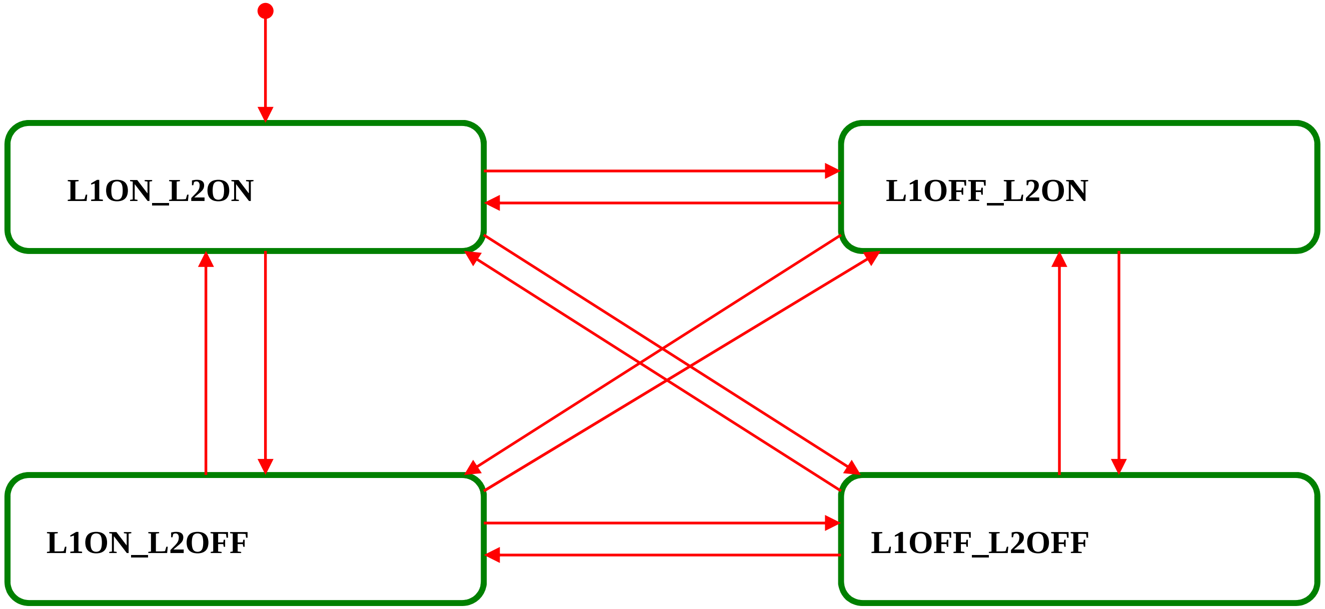


- **Hierarchical state machine**



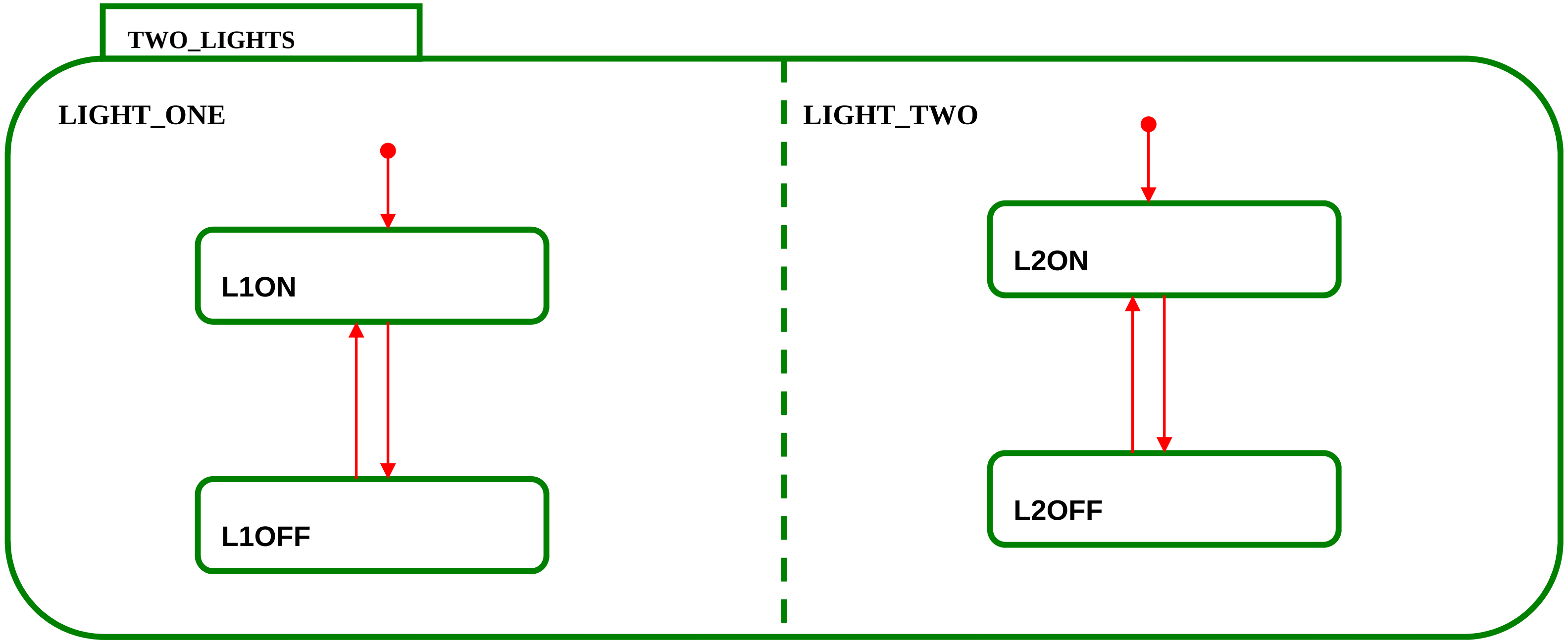
Concurrency

- What happens when you want to look at the behavior of two lights simultaneously?



Concurrency

- Allows the behavior of two lights to be viewed simultaneously.



- When using concurrent states it is recommended that the states do not interact or broadcast data.

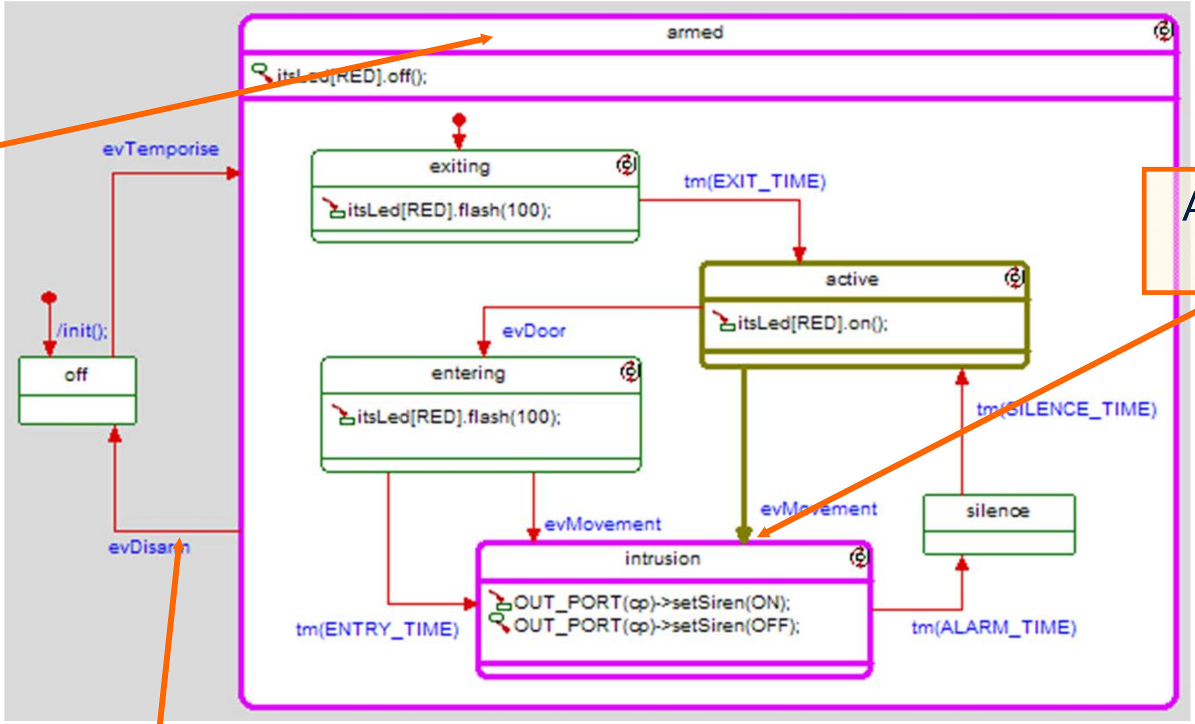
concurrent state communication

- concurrent states may communicate via:
  - Broadcast events
    - All active concurrent states receive their own copy of each received event and are free to act on it or discard it.
  - Propagated events
    - A transition in one concurrent state can send an event that affects another.
  - Guards
    - [IS_IN(state)] uses the substate of a concurrent state in a guard.
  - Attributes
    - Since the concurrent states are of the same object, they “see” all the attributes of the object.

IS_IN is a Rational Rhapsody C macro that can be used to test to see if an object is in a particular state.

Composition

A **composite** state is a state which is composed of other states.
The states contained within a composite state are called **substates**.



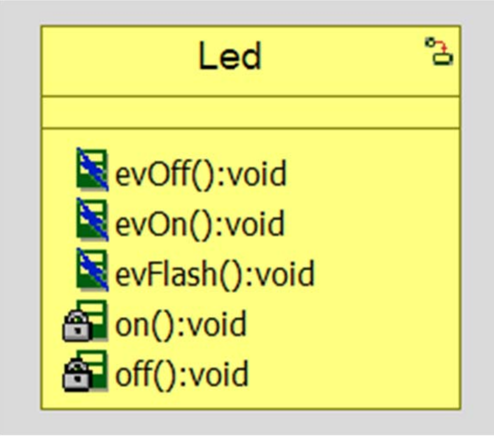
A composite state

A substate of armed

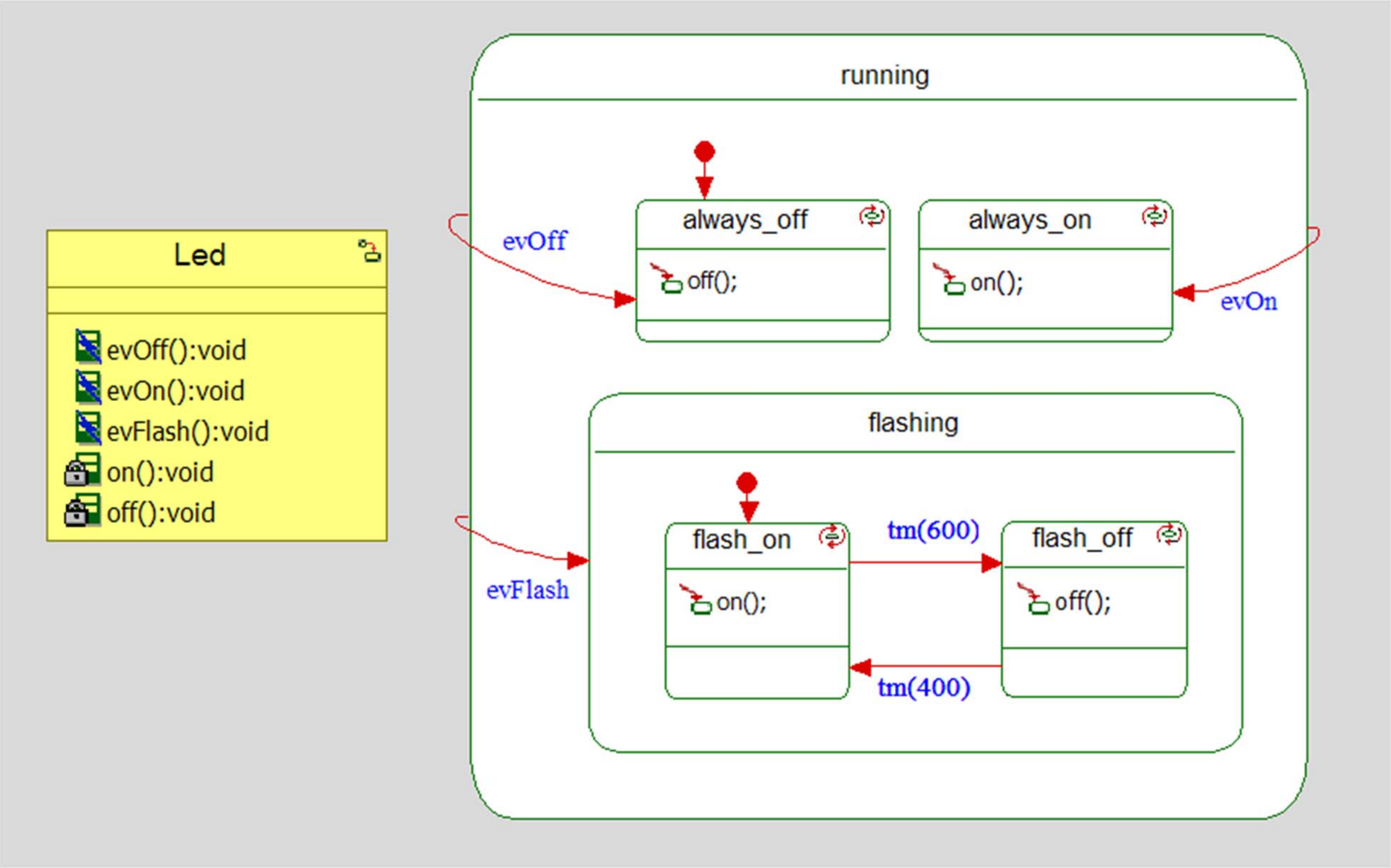
If the event *evDisarm* is received when the object is in state armed, then irrespective of which nested state is active, the transition will be taken and the object will go into the off state.

Exercise: LED

- Draw the state machine for an LED class that can be in one of three modes: on, off and flashing at 1Hz.



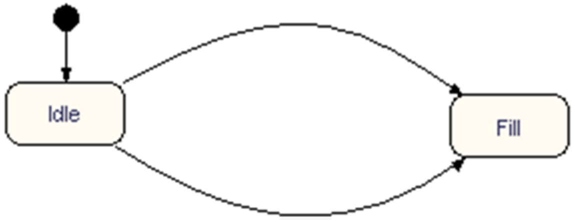
Exercise: LED (Solution)

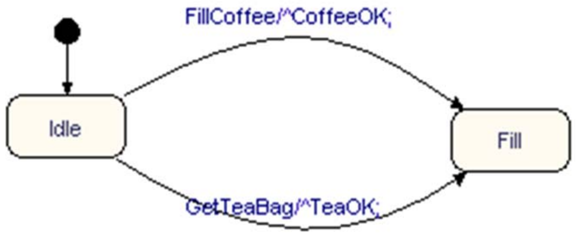


State machine design guidelines

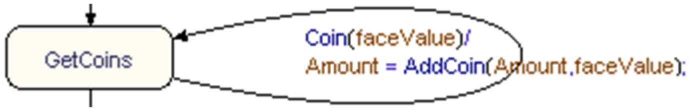
- Identify and define states of the System

- Identify and define transitions





- Identify and define events and actions

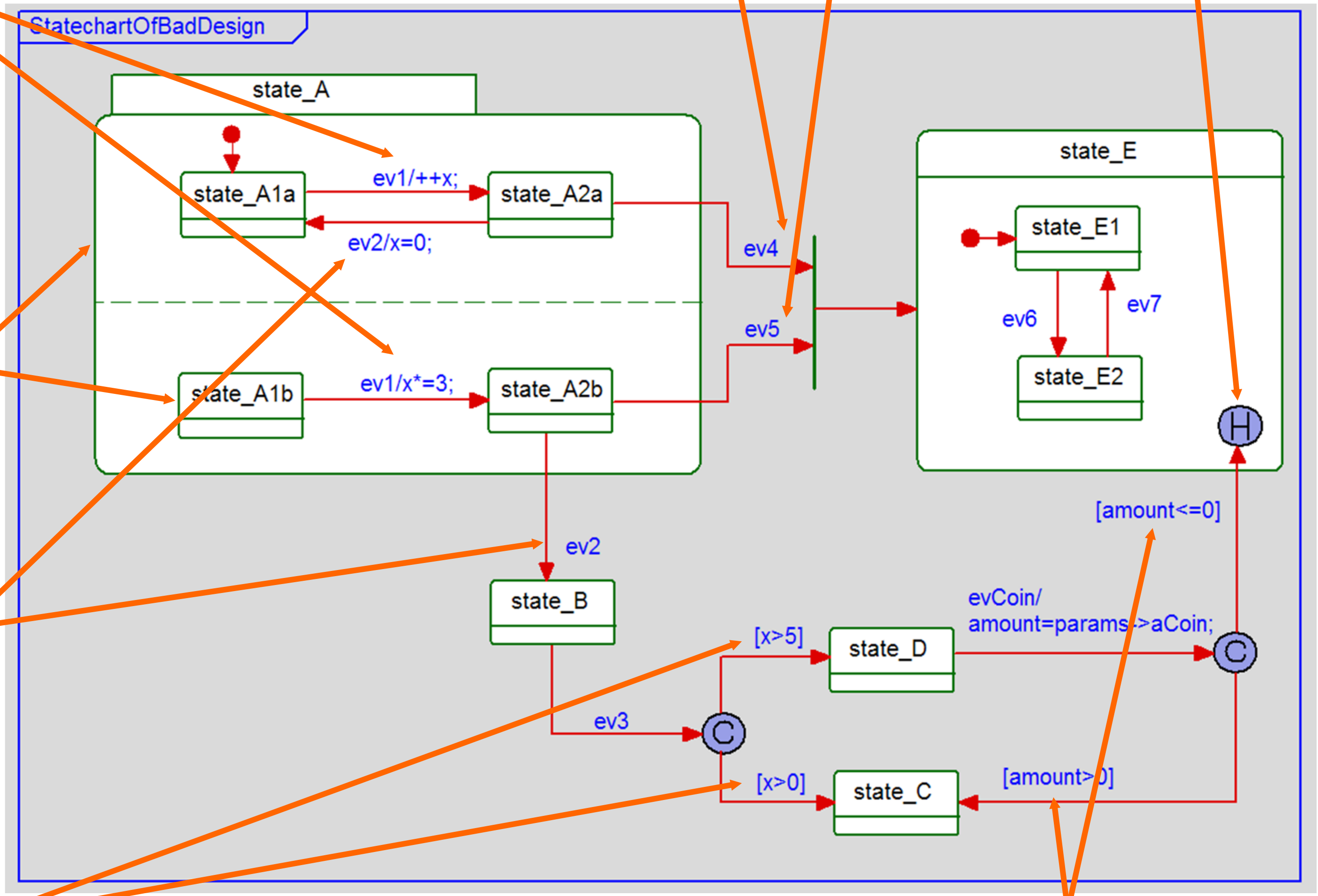


Poorly formed state machine

Race condition

Must be same event

History not initialized



No default state

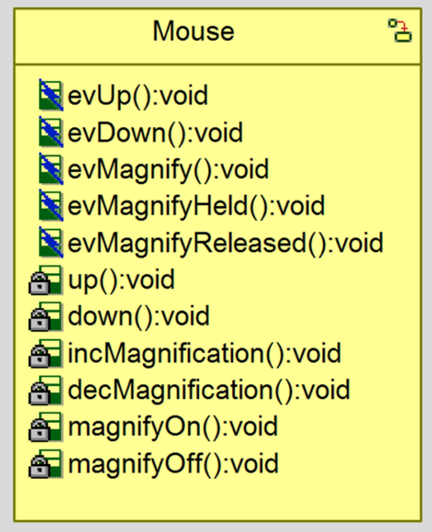
Conflicting transitions

Overlapping guards

Use before initialization

Exercise: mouse

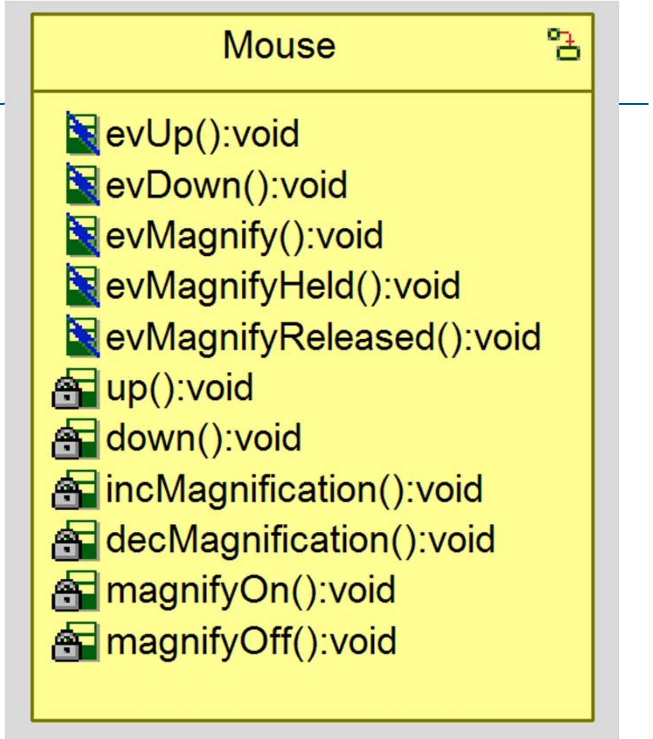
- Draw the state machine for the following mouse that has three extra buttons:
  - One of these buttons allows the Mouse to magnify the area around the mouse. This *magnify* mode is invoked and exited by pressing the magnify button (*evMagnify*).
  - When in the *magnify* mode, if the magnify button is held (*evMagnifyHeld*), then the up (*evUp*) and down (*evDown*) buttons control the magnification, invoking operations *incMagnification()* and *decMagnification(). It* remains in this mode until the magnify button is released (*evMagnifyReleased*).
  - When the magnify button is not held, the up (*evUp*) and down (*evDown*) buttons invoke operations *up()* and *down().*



Mouse

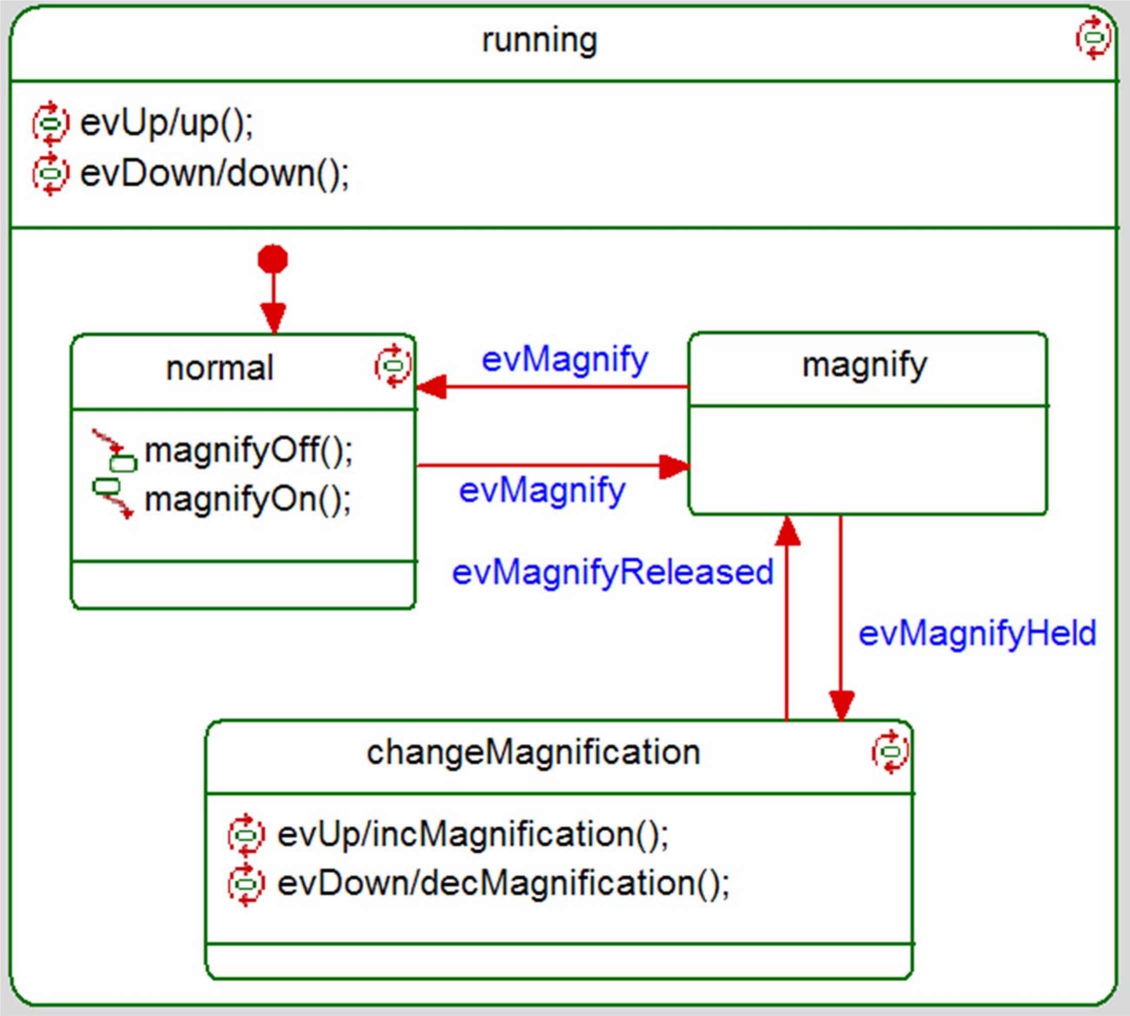
evUp():void
evDown():void
evMagnify():void
evMagnifyHeld():void
evMagnifyReleased():void
up():void
down():void
incMagnification():void
decMagnification():void
magnifyOn():void
magnifyOff():void

Exercise: Mouse (Solution)



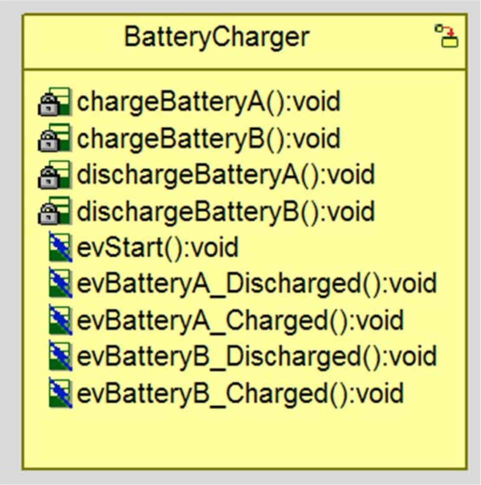
Mouse

evUp():void
evDown():void
evMagnify():void
evMagnifyHeld():void
evMagnifyReleased():void
up():void
down():void
incMagnification():void
decMagnification():void
magnifyOn():void
magnifyOff():void



Exercise: Battery charger

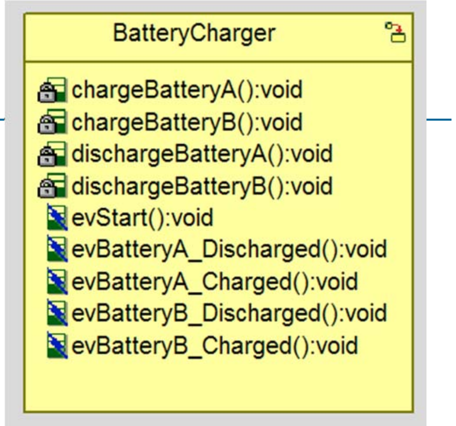
- Draw the state machine for a simple Battery Charger that can charge two batteries in parallel. The charger has three modes: idle, discharging, and charging.
- A button can be pressed (evStart) to start charging the batteries. However, before each battery can be charged, it must be discharged.
- When each battery is discharged, it sends an event (evBatteryA_Discharged or evBatteryB_Discharged) to the Battery Charger.
- When each battery is charged, it sends an event (evBatteryA_Full or evBatteryB_Full) to the Battery Charger.
- When both batteries are charged, the Battery Charger returns to the idle mode.



BatteryCharger

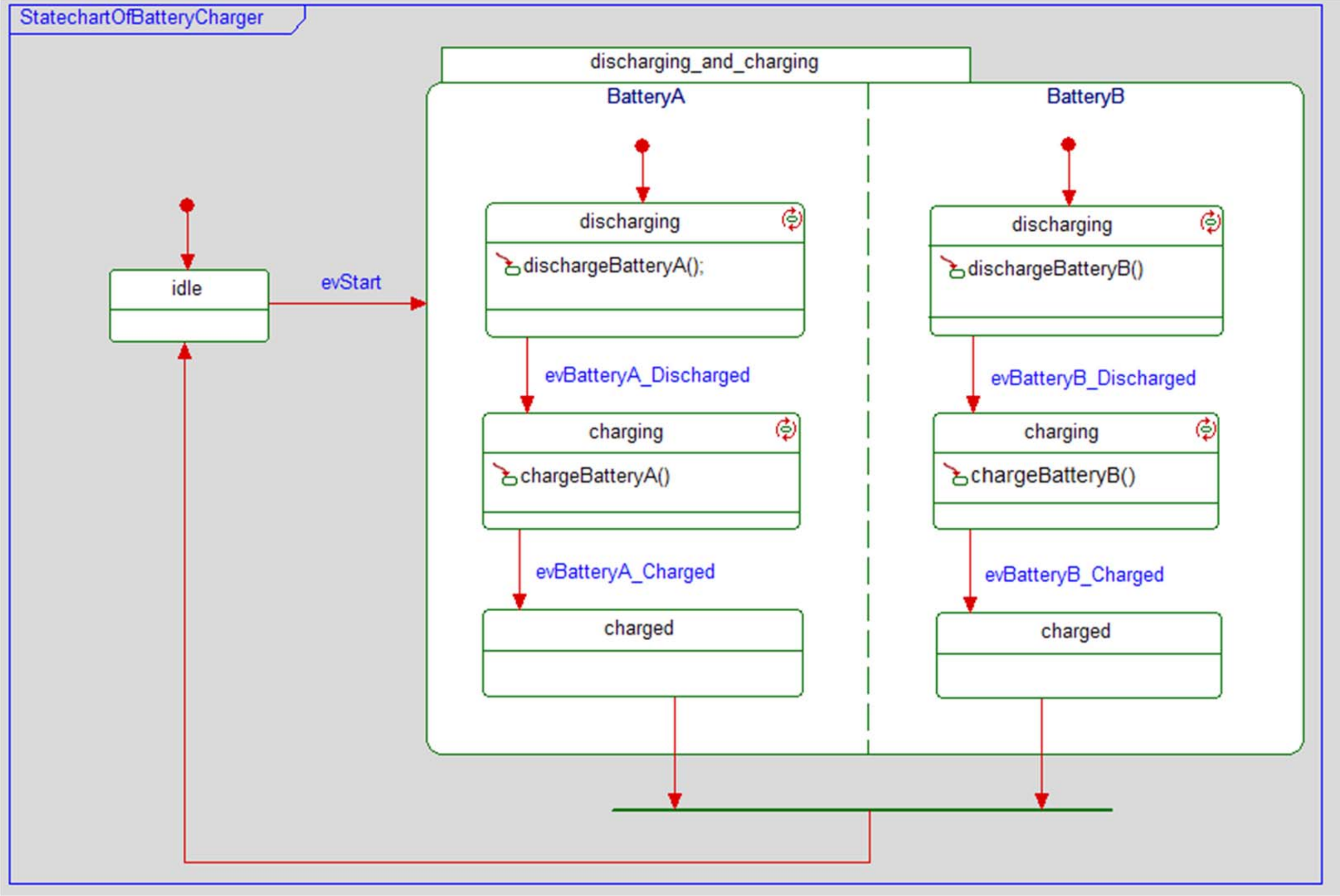
chargeBatteryA():void
chargeBatteryB():void
dischargeBatteryA():void
dischargeBatteryB():void
evStart():void
evBatteryA_Discharged():void
evBatteryA_Charged():void
evBatteryB_Discharged():void
evBatteryB_Charged():void

Exercise: Battery charger (Solution)



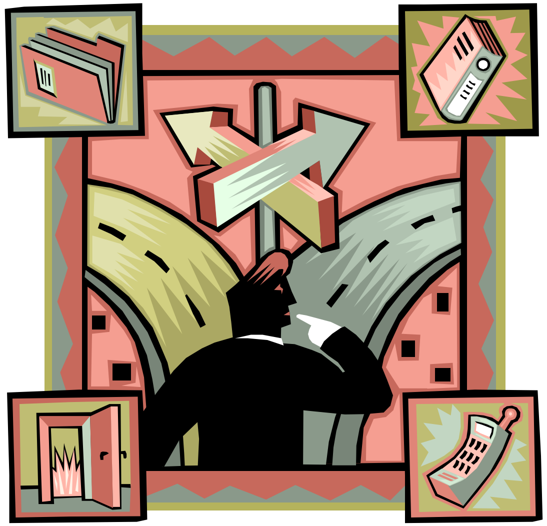
BatteryCharger

- chargeBatteryA():void
- chargeBatteryB():void
- dischargeBatteryA():void
- dischargeBatteryB():void
- evStart():void
- evBatteryA_Discharged():void
- evBatteryA_Charged():void
- evBatteryB_Discharged():void
- evBatteryB_Charged():void



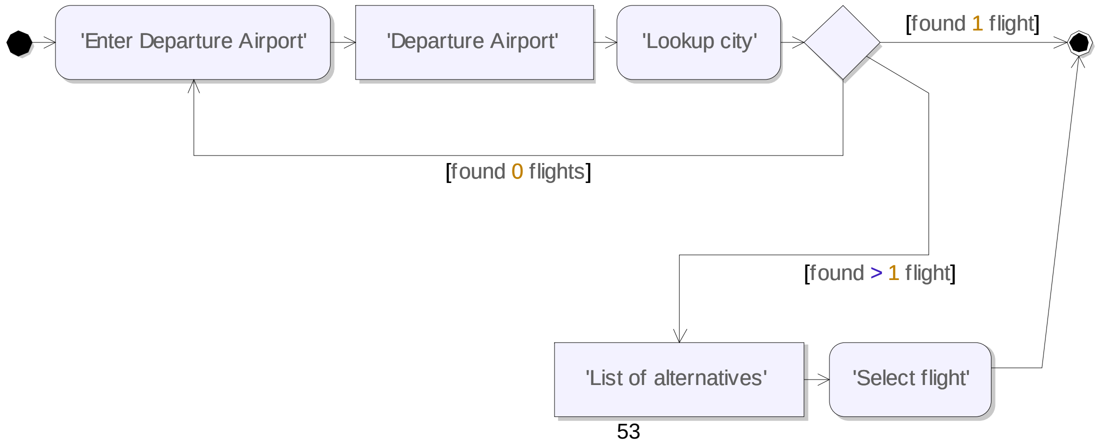
Where are we?

- How to describe behavior?
- Modelling with State machine diagrams
- Modelling with Activity diagrams

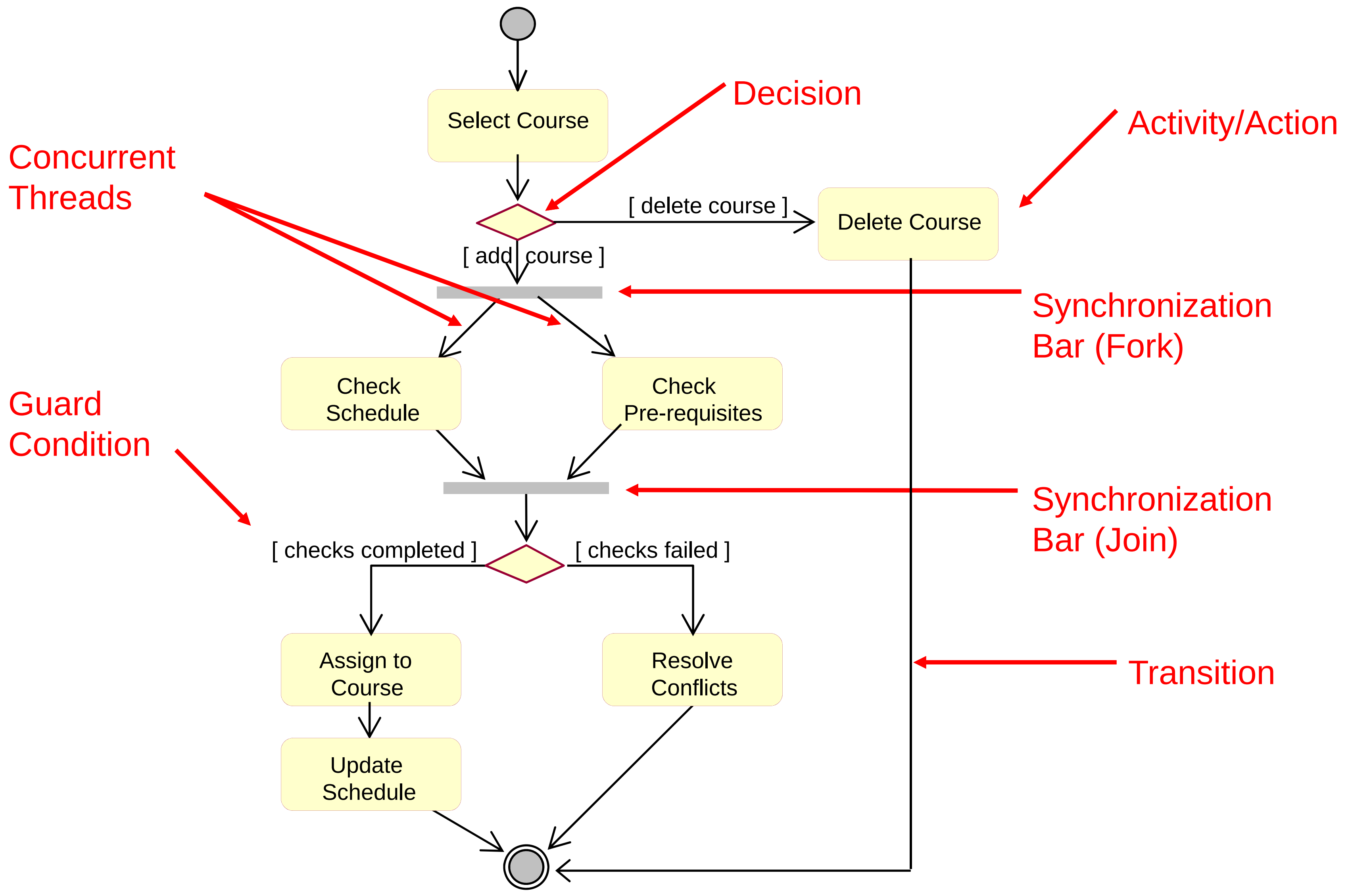


What Is an Activity Diagram?

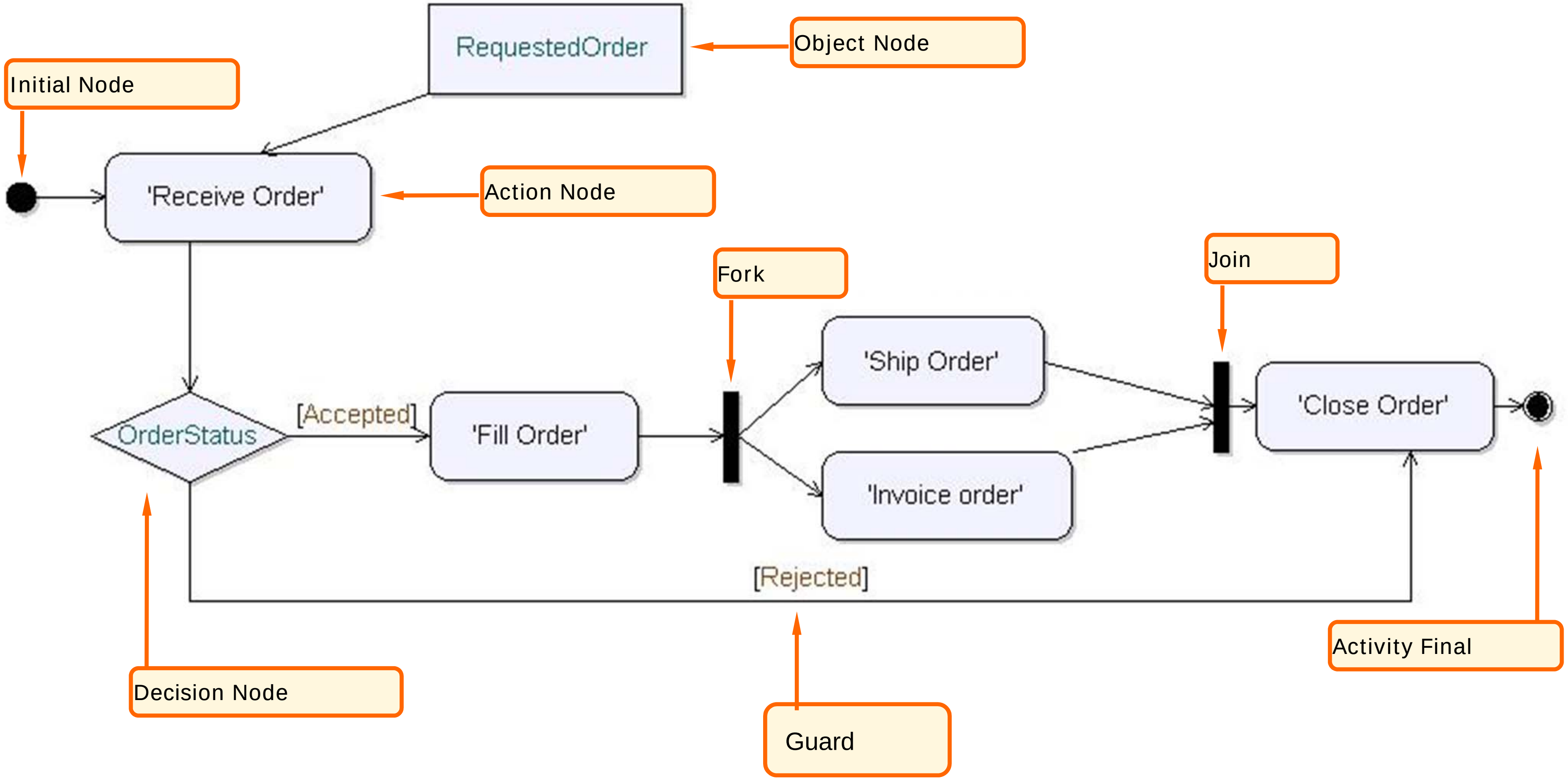
- Describe the **workflow behavior** of a system
  - An activity diagram captures the **activities** and **actions** performed.
- It is essentially a **flow chart**, showing flow of control from one activity or action to another.
- Show activities that are **conditional** or **parallel**.
- Useful for:
  - analyzing a use case by describing **what actions** need to take place and when they should occur
  - describing a complicated sequential **algorithm**
  - modeling applications with **parallel processes**
  - modeling **bussiness workflow**



Example: Student course selection

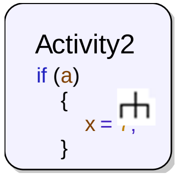


Activity Diagram – Basic Syntax



Activity Diagram Symbols - 1

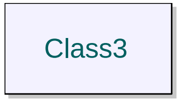
- Initial node
  - Starting point for invoking other activities. An activity may have several starting points.



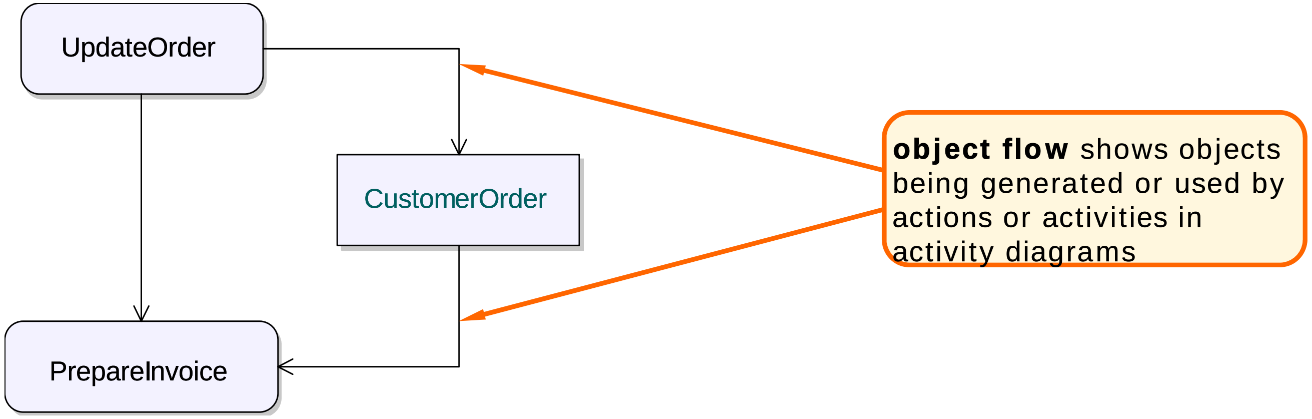
```
if (a)
   {
      x = 7;
   }
```

- Action/Activity
  - An action is an executable unit. Can also refer to a new activity diagram - sub-activity

Activity Diagram Symbols - 2

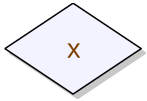


- Object node
  - Used to show input to or output from an action.

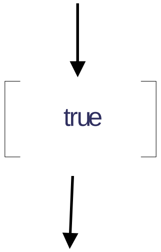


object flow shows objects being generated or used by actions or activities in activity diagrams

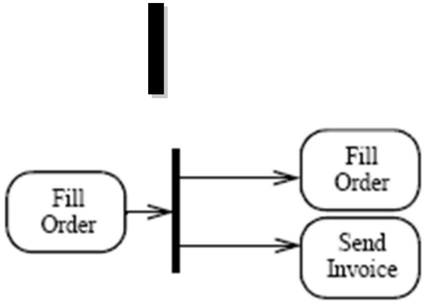
Activity Diagram Symbols - 3



- Decision node
  - A decision node is a control node that chooses between outgoing flows.
  - Each branch has its own guard condition
  - “Else” may be defined for at most one outgoing transition

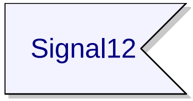


- Guard
  - Alternative from a decision node that is mutually exclusive from the other alternatives

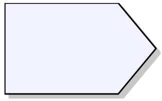


- Fork/join symbol
  - Divides a flow into multiple concurrent flows. Flows can be split and synchronized again.

Activity Diagram Symbols - 4



- Accept event symbol
  - Represents an input action.

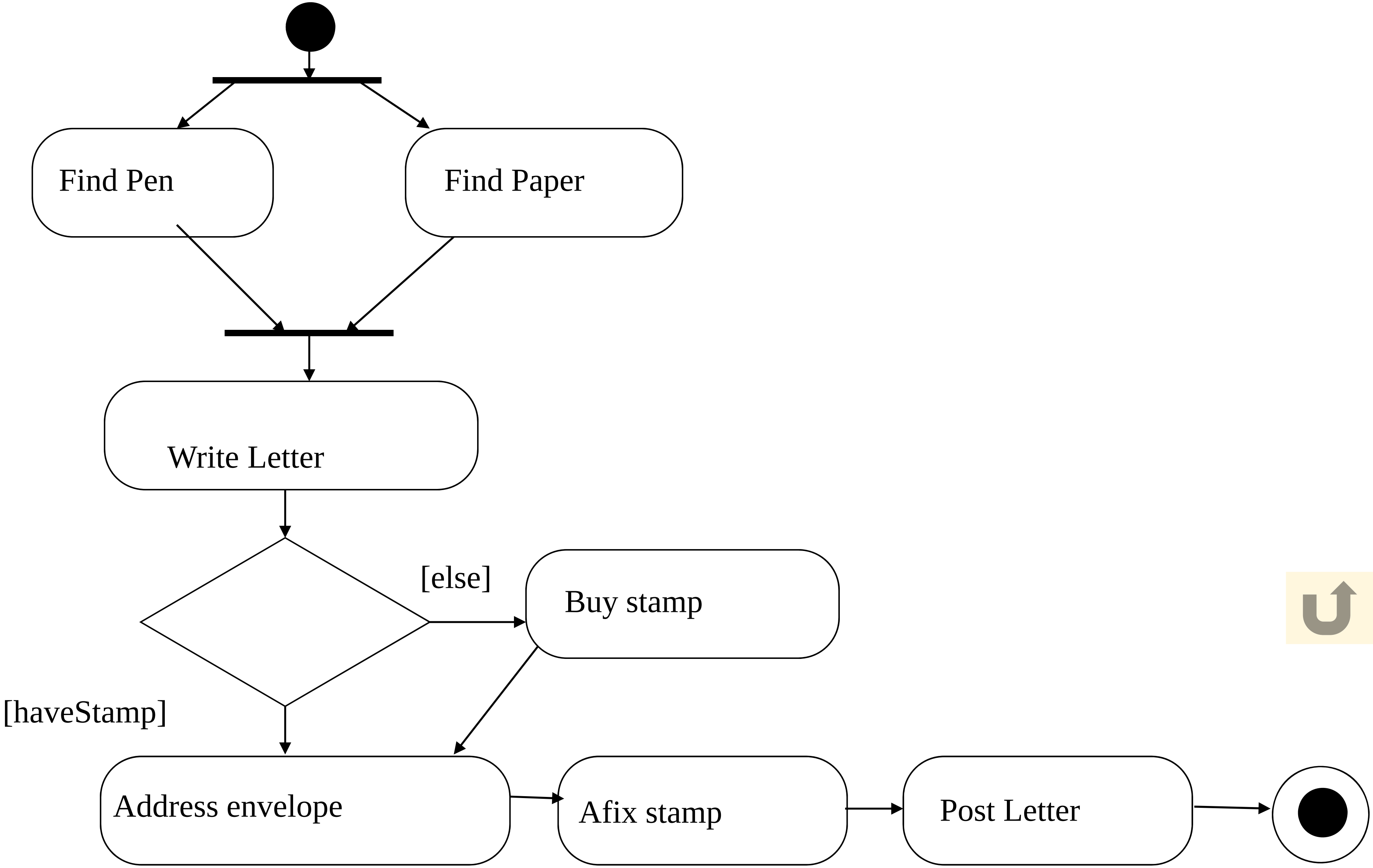


- Send signal symbol
  - Represents an *output* action.

Exercise

- Write an activity diagram to describe the process of writing and posting a letter. You might or might not have a stamp.

Exercise - Solution



Review

- What is system behavior?
- What is a state machine diagram? Describe the different parts of the diagram.
- Define state.
- What is an activity diagram?
- What kind of behavior is best suited to be modeled with state machine Diagrams? What kind of behavior is best suited to be modeled with Activity Diagrams?