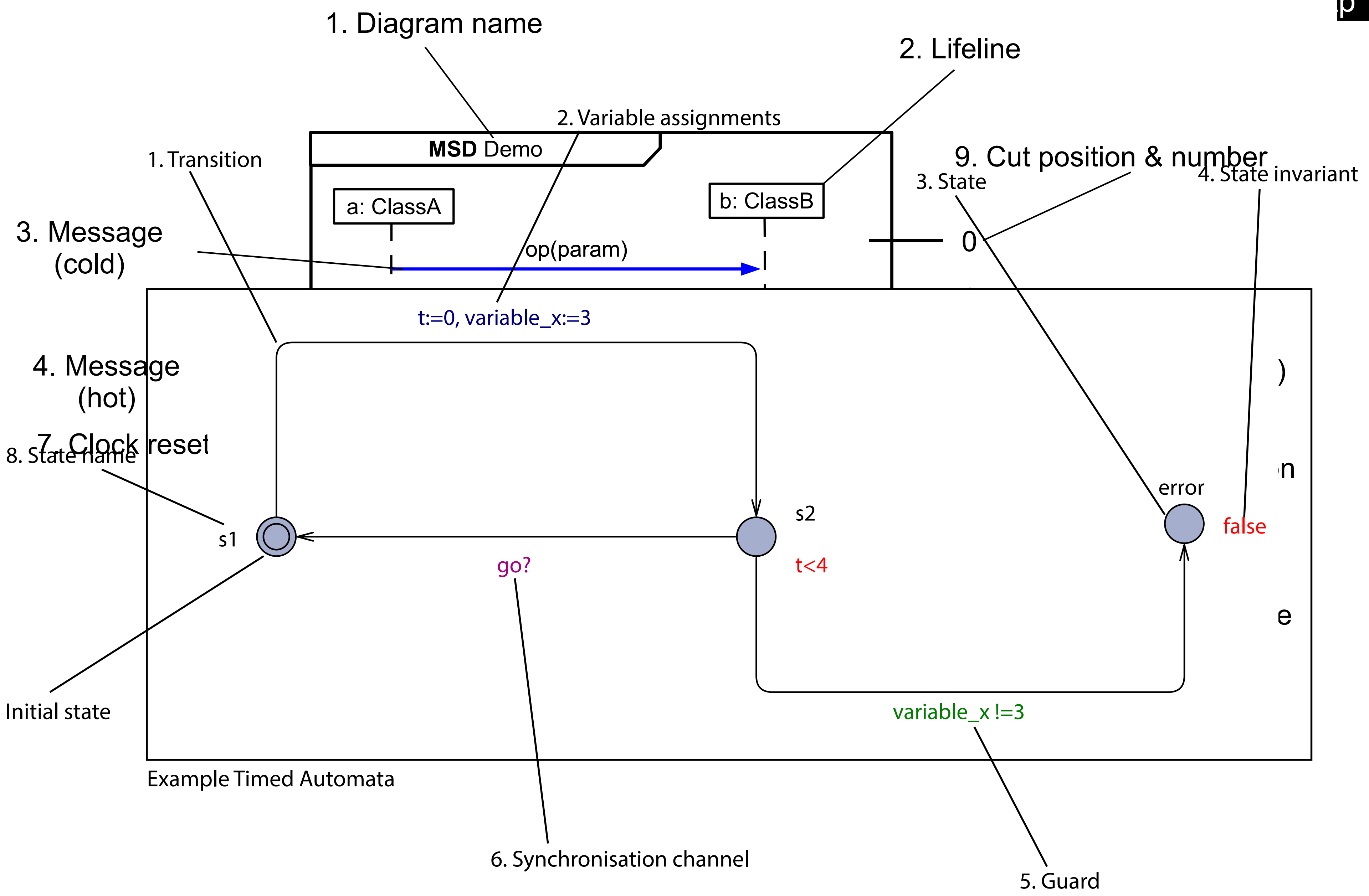
Thursday

- "Extended Modelling Notations, Experiment"

- Expressing scenarios that must/must not occur
- Analysing models (E.g. verification)
- Non-UML notations


1. Diagram name
2. Lifeline
2. Variable assignments
MSD Demo
1. Transition
a: ClassA
b: ClassB
9. Cut position & number
3. State
4. State invariant
3. Message (cold)
op(param)
0
t:=0, variable_x:=3
4. Message (hot)
7. Clock reset
8. State name
s1
s2
error
false
go?
t<4
Initial state
variable_x !=3
Example Timed Automata
6. Synchronisation channel
5. Guard

What about the Experiment?

- Second part of the lecture: Experiment
- Connected to my (Grischa) research: Are some notations harder/easier to understand than others?
- Participation is <u>**voluntary**</u>…but would really help me!
- And: Similar question style as voluntary exam III. So, it’s a good practice!

Object-oriented System Development

Lecture 9

State Machines

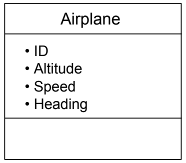
State machines

A group of similar things is abstracted as a class and their common lifecycle is abstracted as a **state machine**.

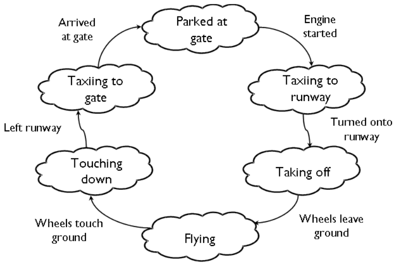
Like things

ABSTRACTED

Class

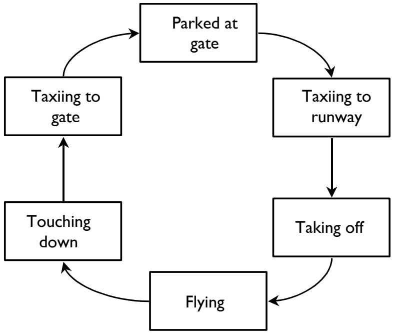


Common behavior pattern



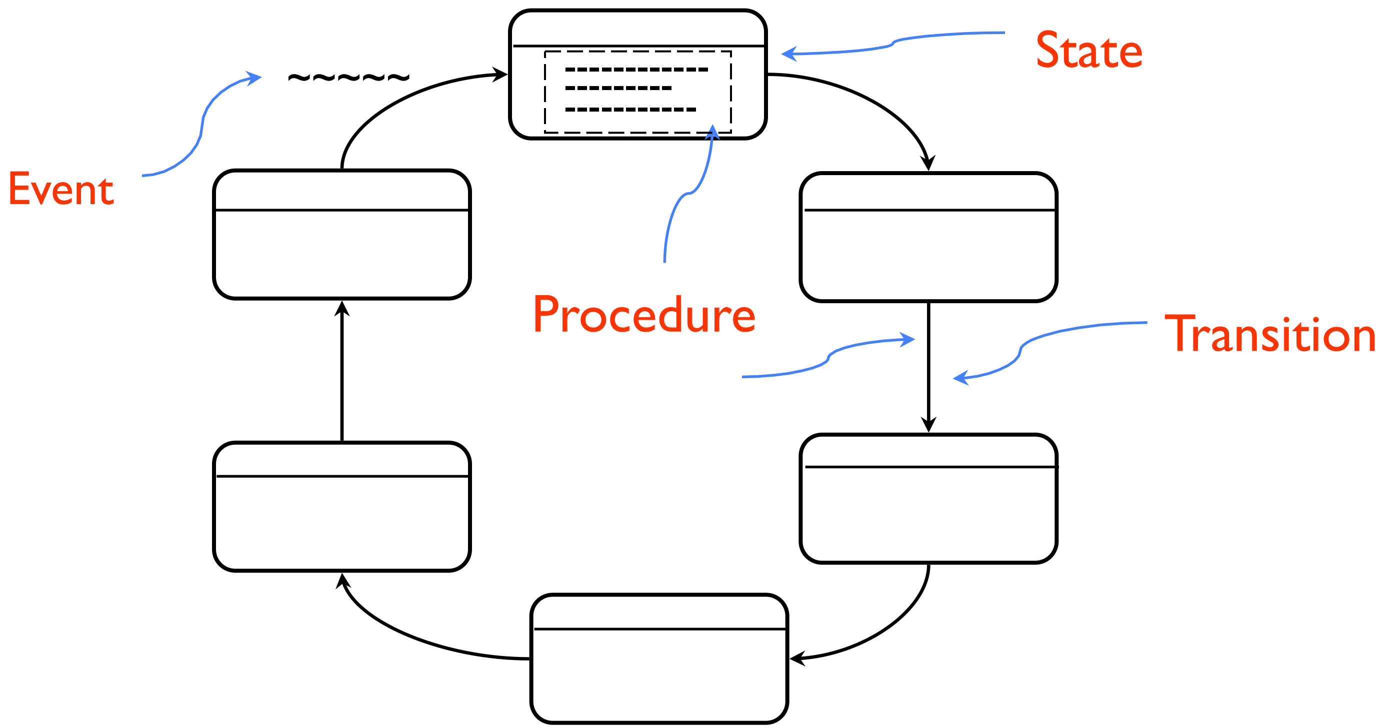
ABSTRACTED

State Machine



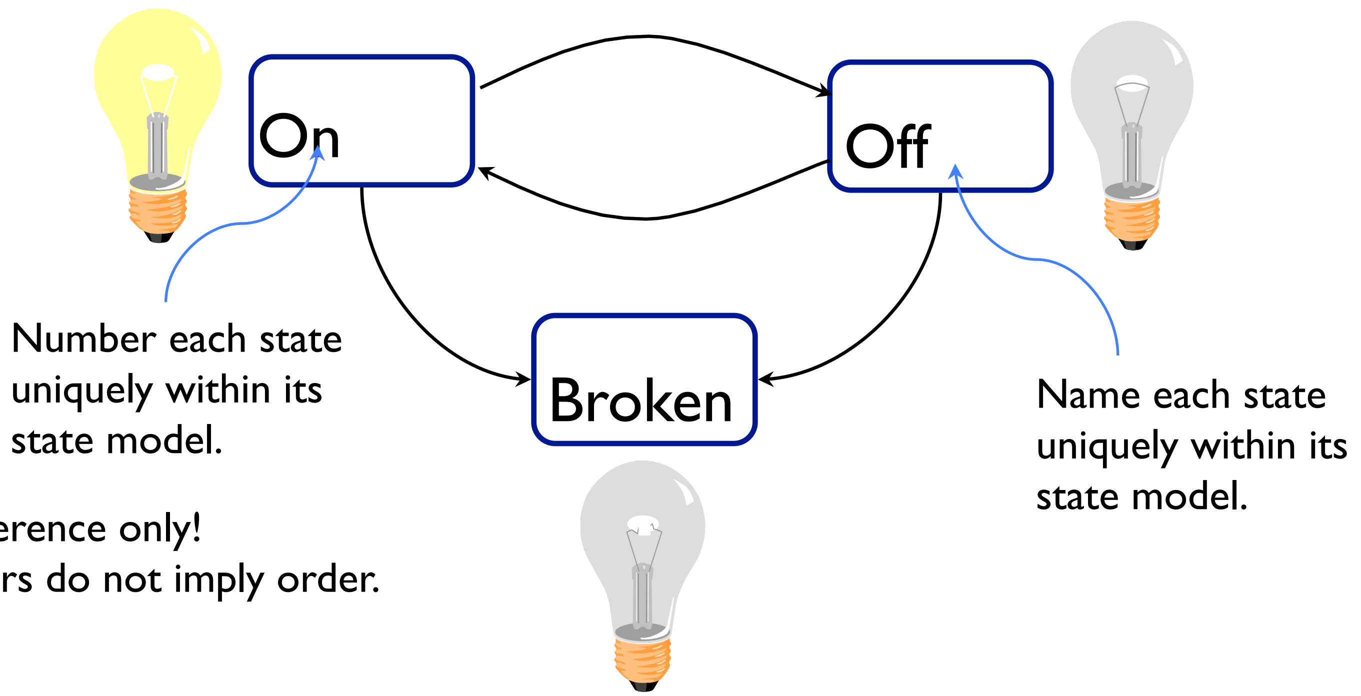
Statechart

A state machine formalizes a lifecycle in terms of states, events, transitions and procedures.



State

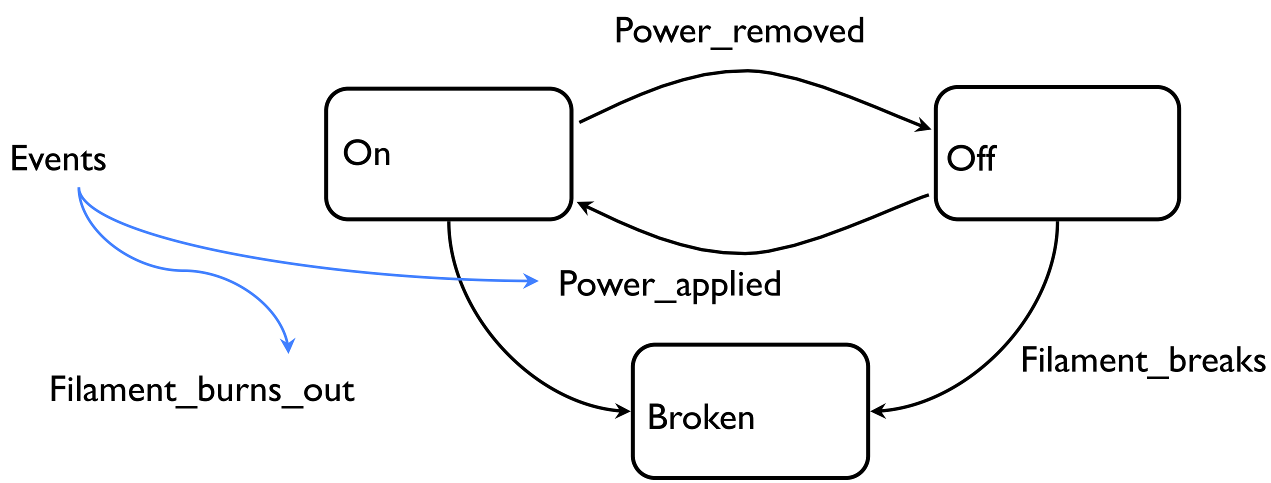
A state represents a condition of an object subject to a defined set of rules, policies, regulations and physical laws.



For reference only!
Numbers do not imply order.

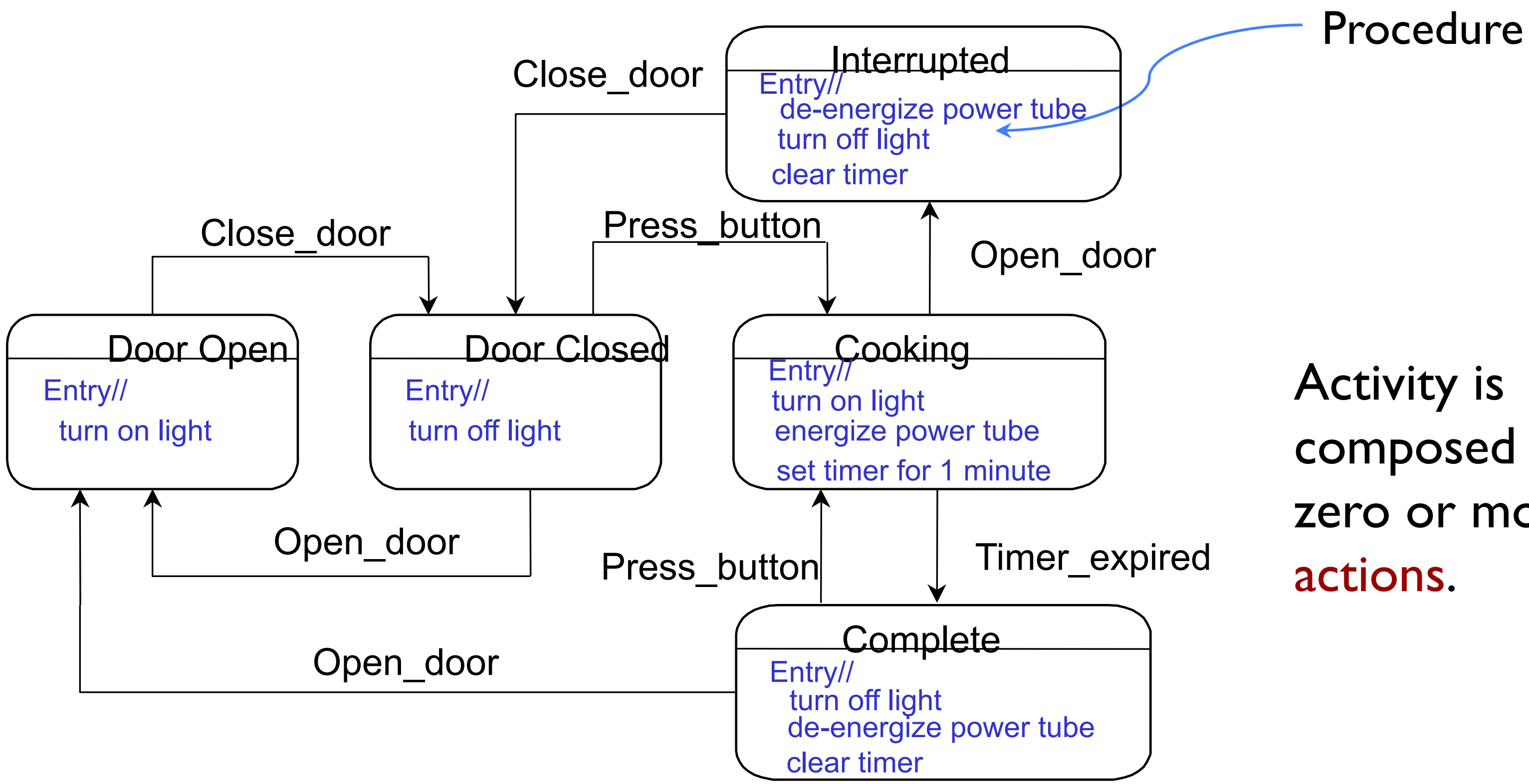
Event

An event represents something that has happened and that may trigger a transition.



Activity (Oven)

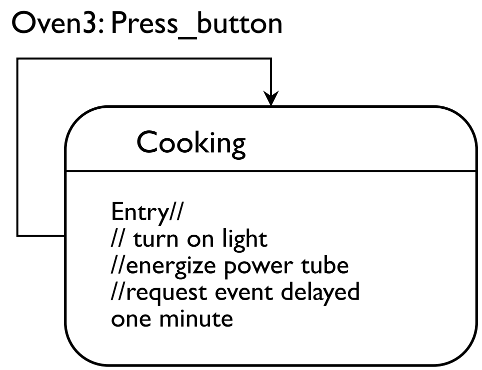
Activity is an operation executed by an instance when it enters a state.



Activity is composed of zero or more actions.

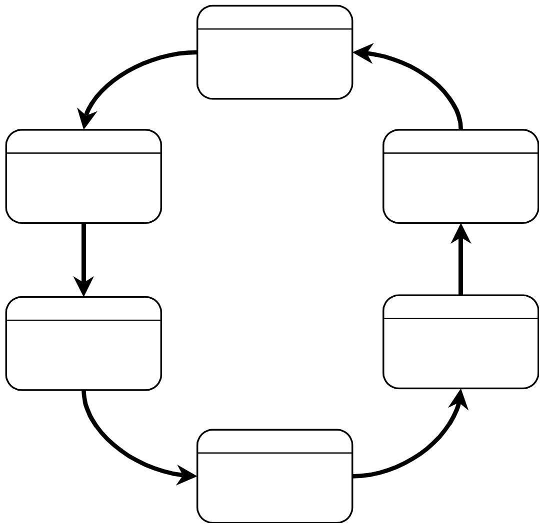
Reflexive transition

An event may invoke a reflexive transition from one state back into the same state.

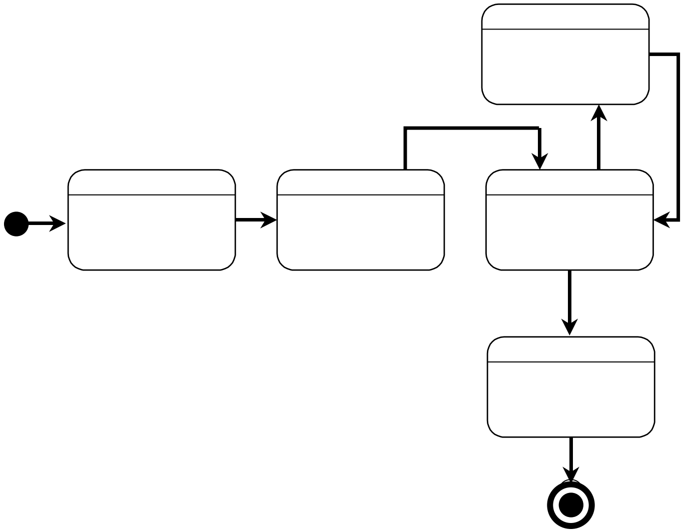


Be careful! The procedure is executed each time the state is entered.

Typical lifecycle patterns



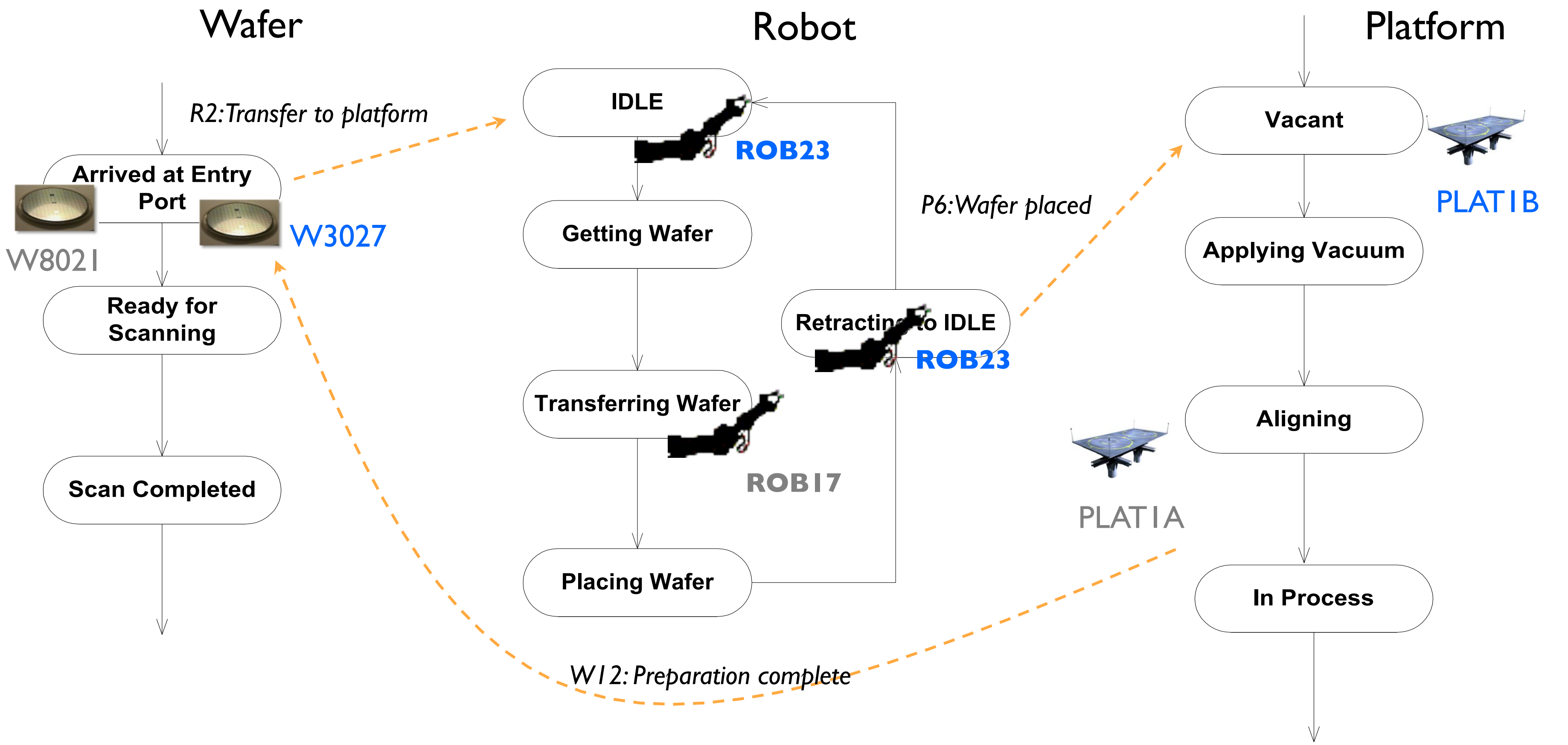
Cyclic



Born-and-die

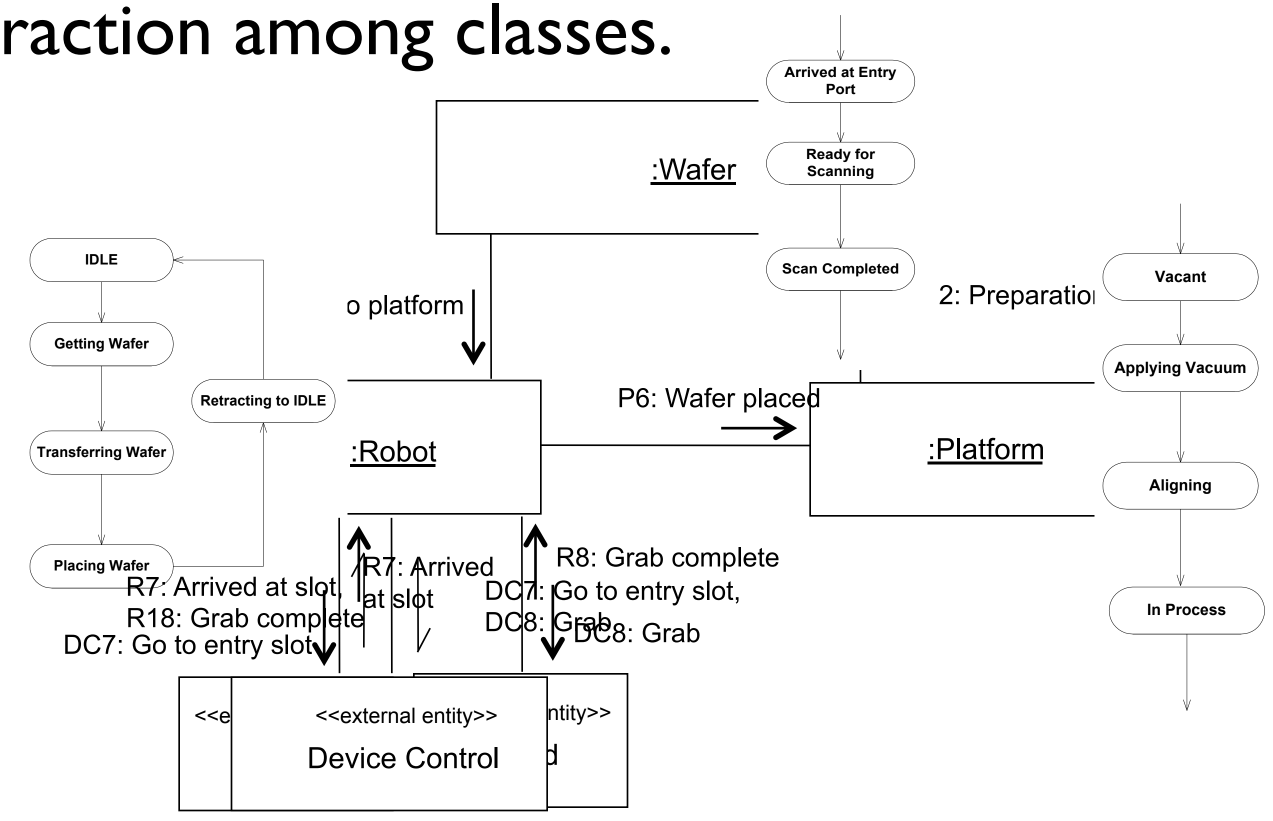
State machines communicate

Signals are exchanged among objects.

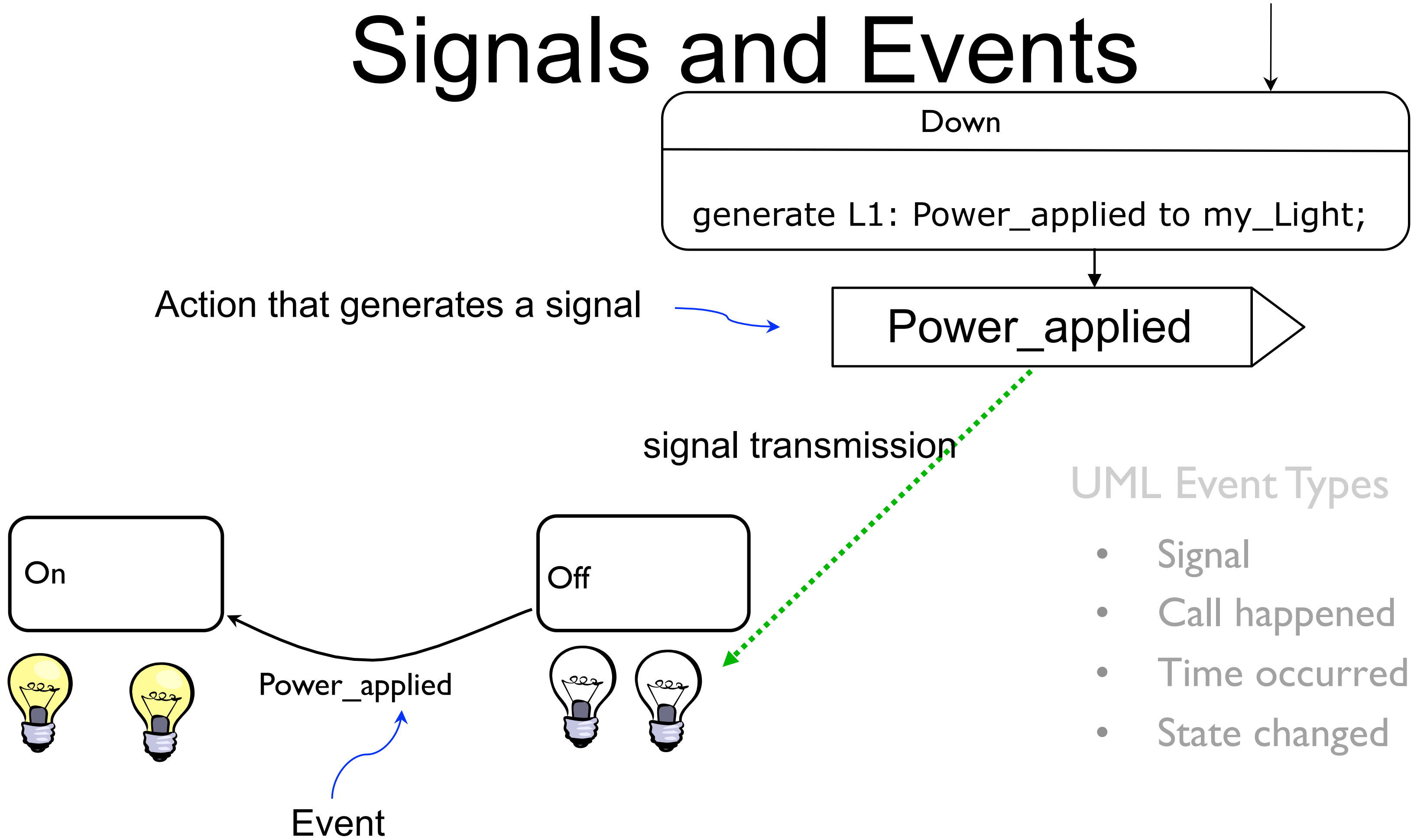


Collaboration diagram

Use a class collaboration diagram to illustrate interaction among classes.



Signals and Events

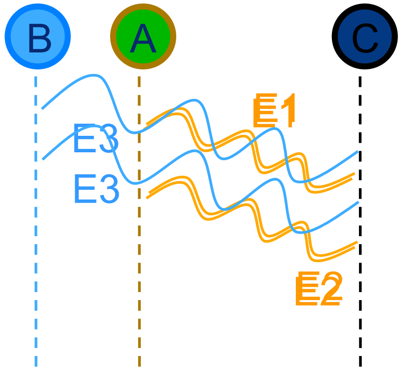


UML Event Types

- Signal
- Call happened
- Time occurred
- State changed

Order of arriving events

Each object has its own lifeline



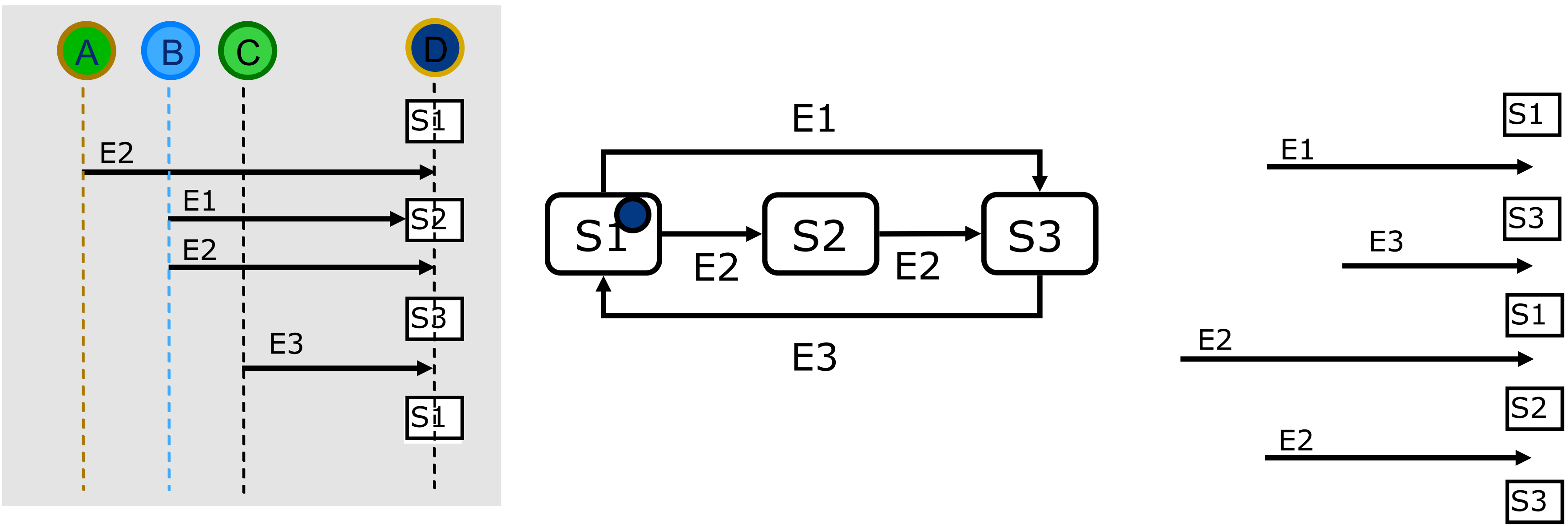
- Clocks cannot be synchronized.
- To observe the sequence of signals generated by a single object synchronization is not necessary.
- So the sequence of signals generated by the same object can be preserved and guaranteed by the architecture.

E1 from A occurs before E2 from A. But E3 can happen anywhere before, between or after E1 and E2.

Event sequencing example

Objects A, B and C trigger events in object D's statechart.

Two signals are sent in sequence from object B.

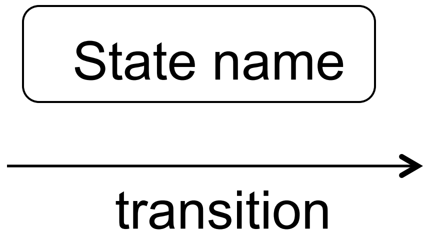


Our first test concludes in state S1.

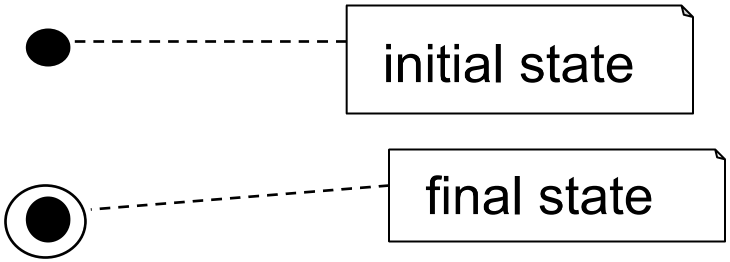
This time we end up in S3!

State Charts

Syntax

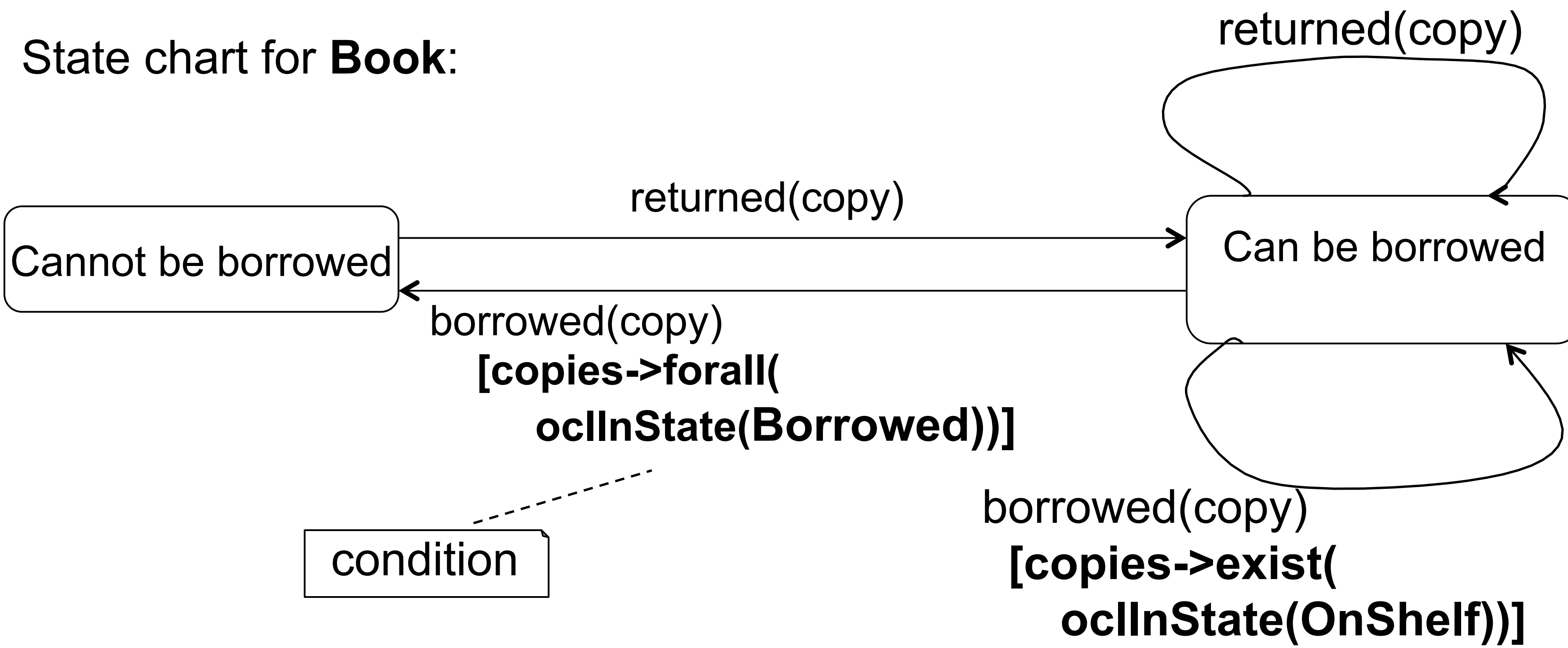


<event> ::= <event> [‘**,**’event][‘**[**‘guarded-constraint‘**]**’] [‘**/**’action]

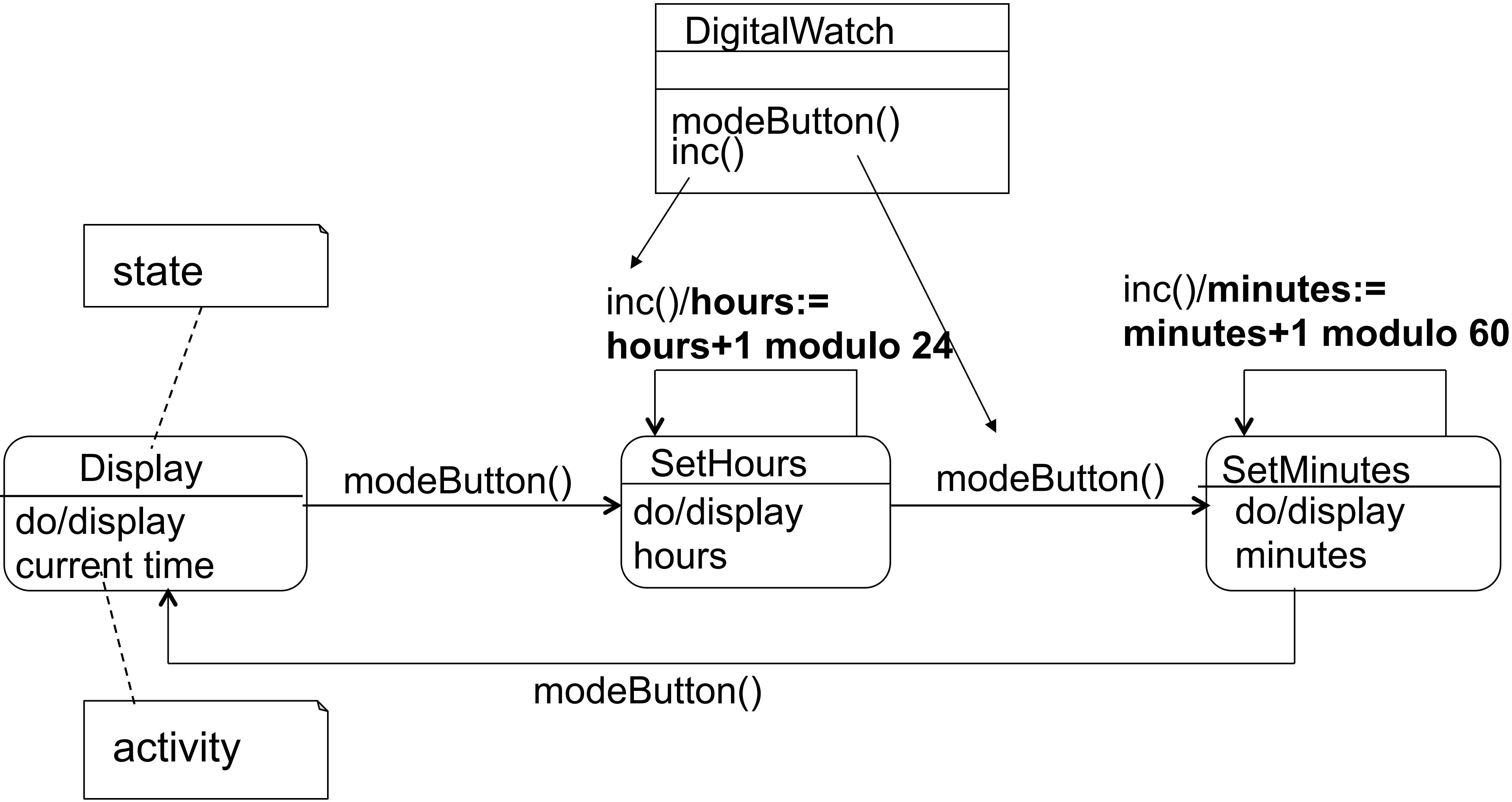


Conditions

State chart for **Book**:

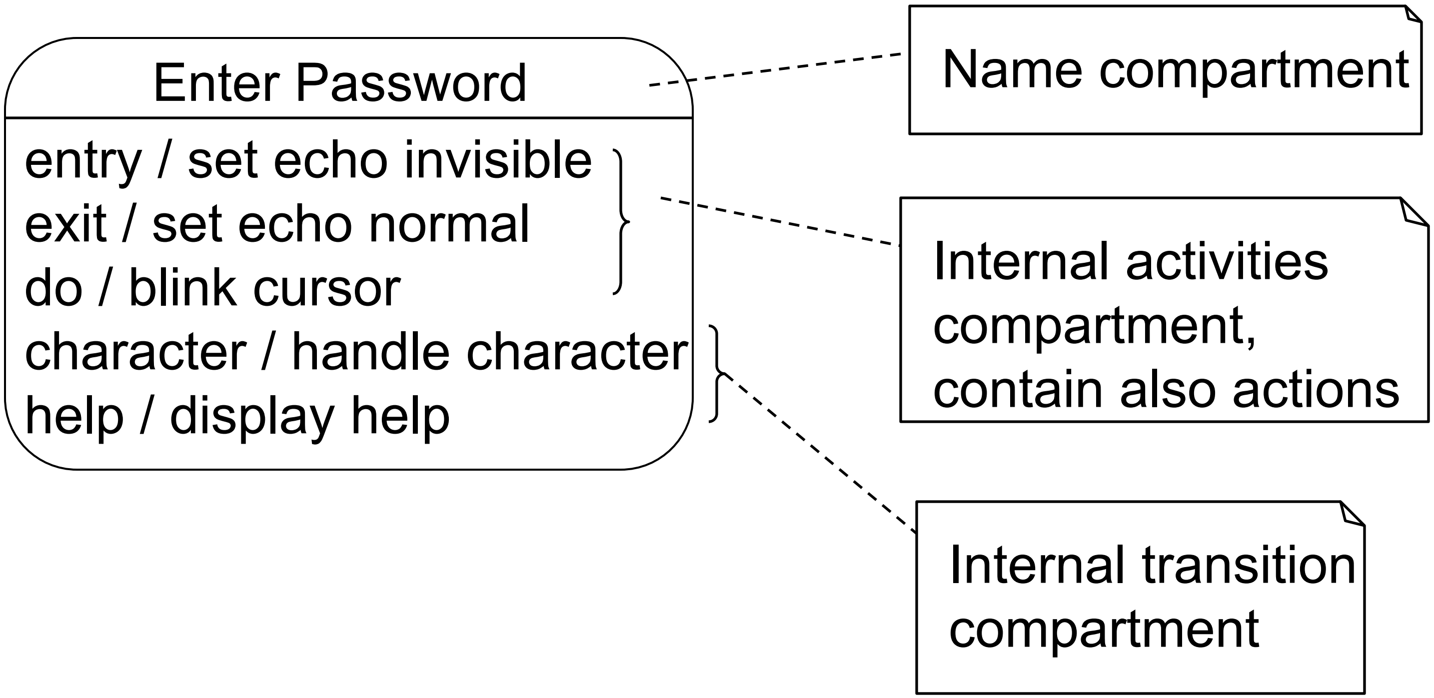


Action

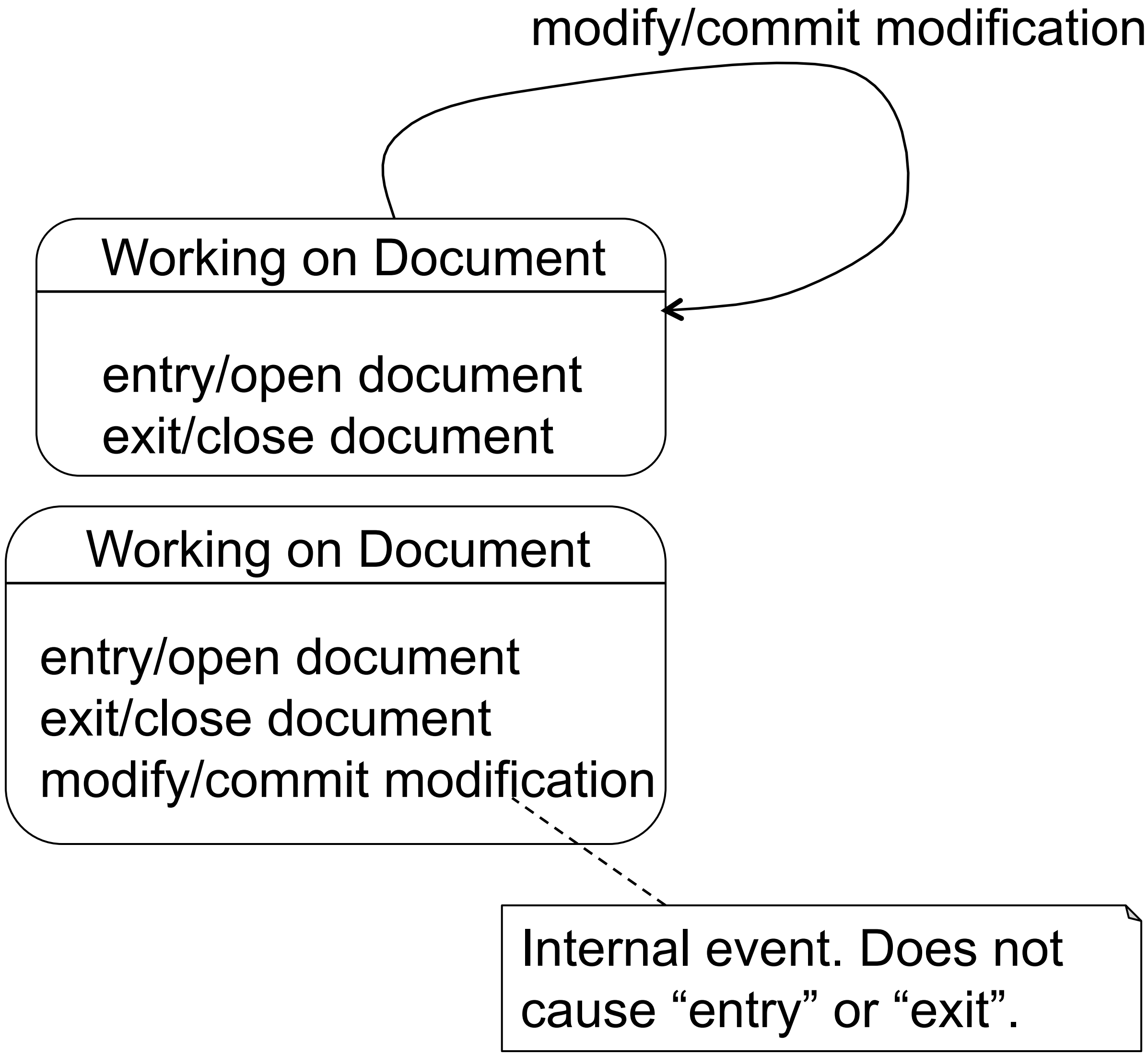

DigitalWatch
modeButton()
inc()
state
inc()/**hours:=**
hours+1 modulo 24
inc()/**minutes:=**
minutes+1 modulo 60
Display
do/display
current time
modeButton()
SetHours
do/display
hours
modeButton()
SetMinutes
do/display
minutes
modeButton()
activity

Compartments

- Simple state chart describing how a password entry widget works:



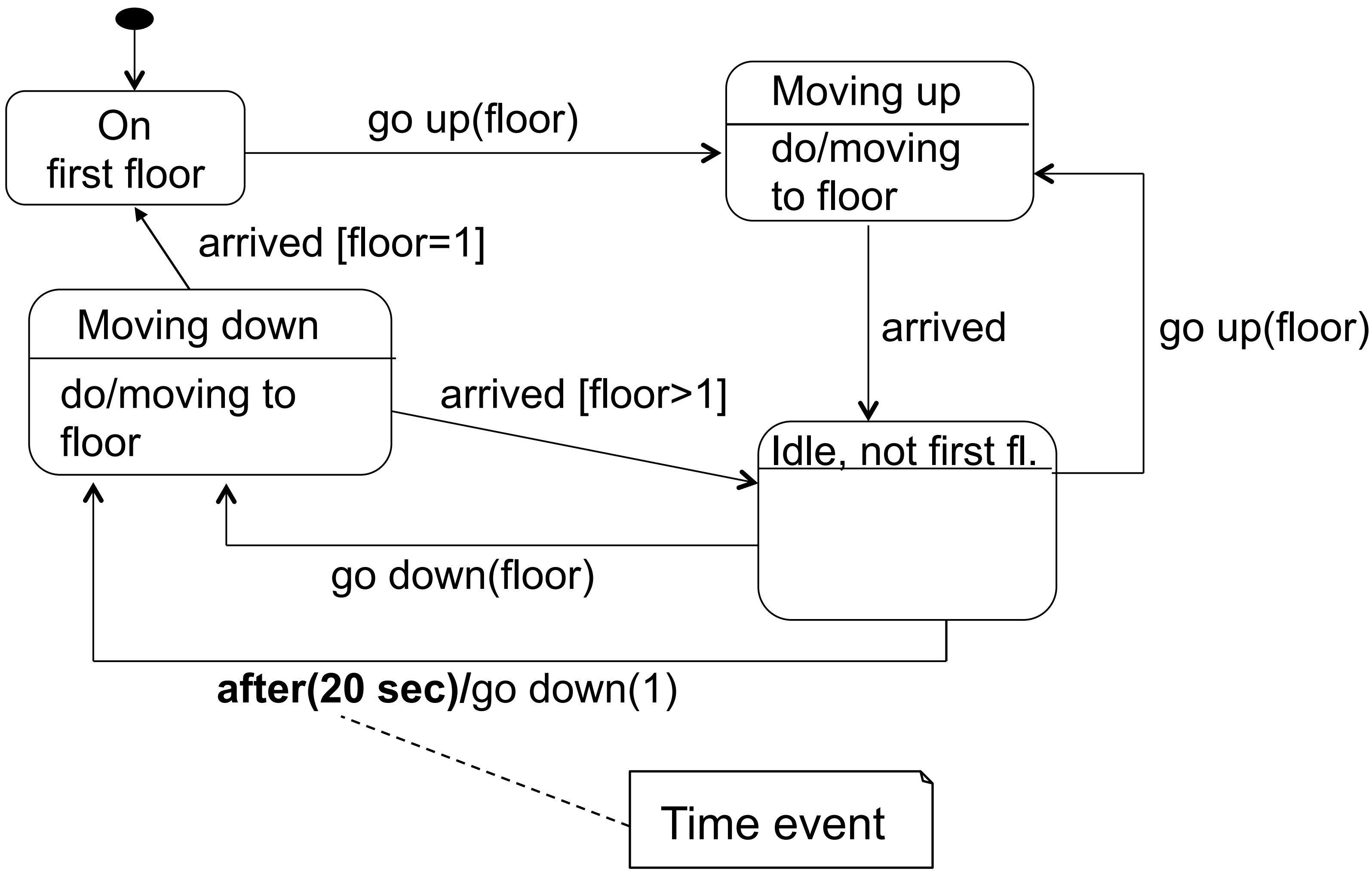
Internal Events



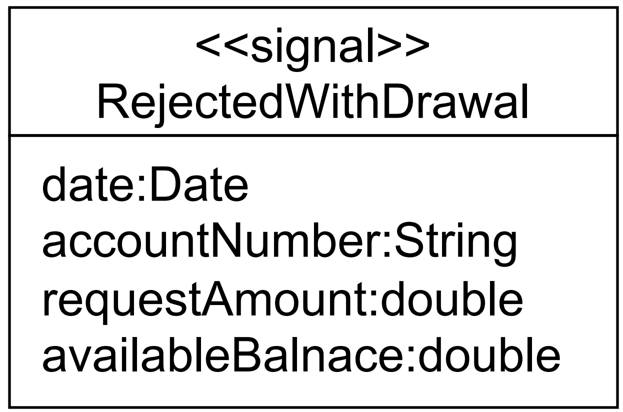
Events

- Call events
- Time events
- Change events
- Signal events

Time Event

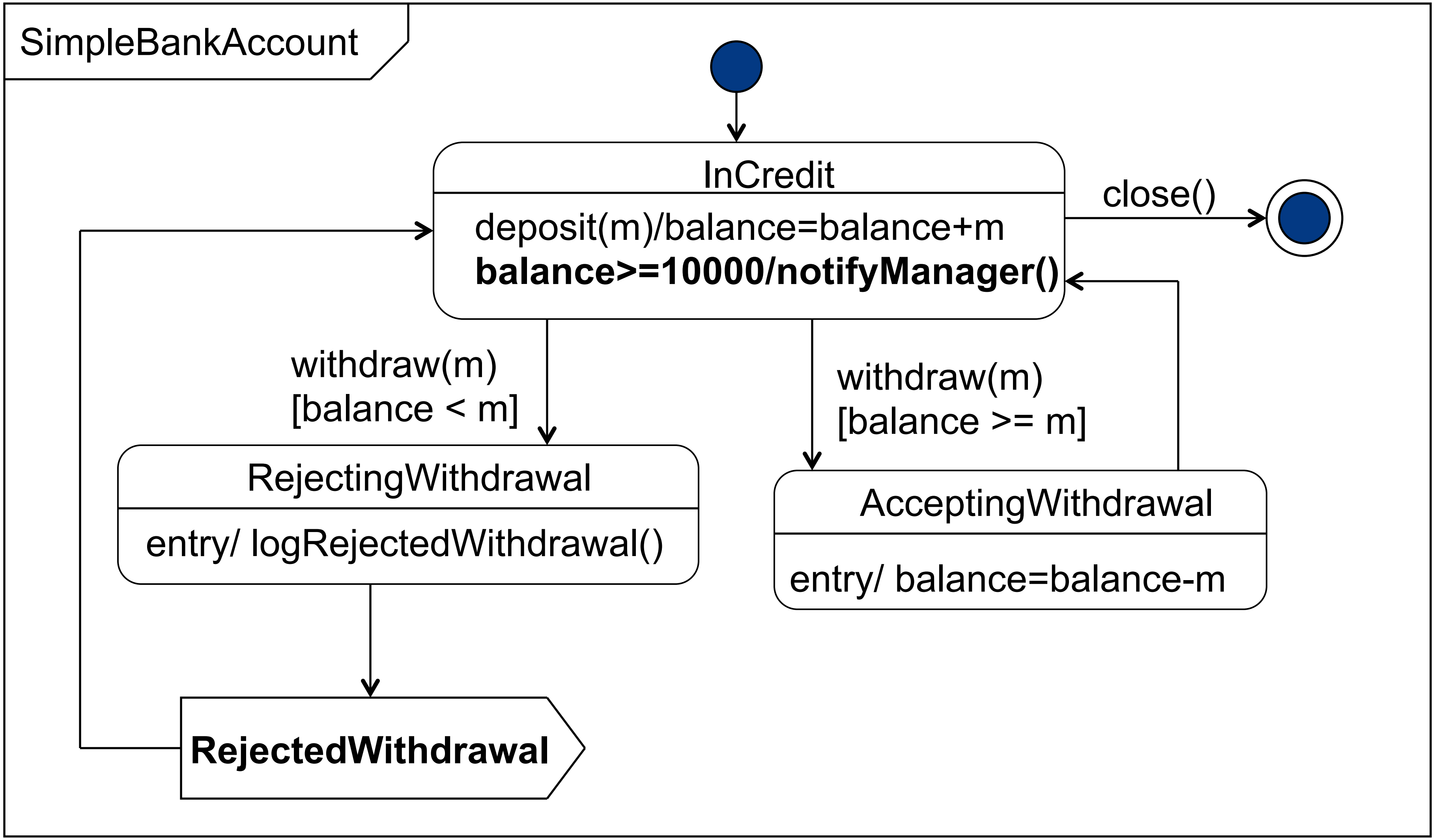

On
first floor
go up(floor)
Moving up
do/moving
to floor
arrived [floor=1]
Moving down
do/moving to
floor
arrived
go up(floor)
arrived [floor>1]
Idle, not first fl.
go down(floor)
after(20 sec)/go down(1)
Time event

Signal

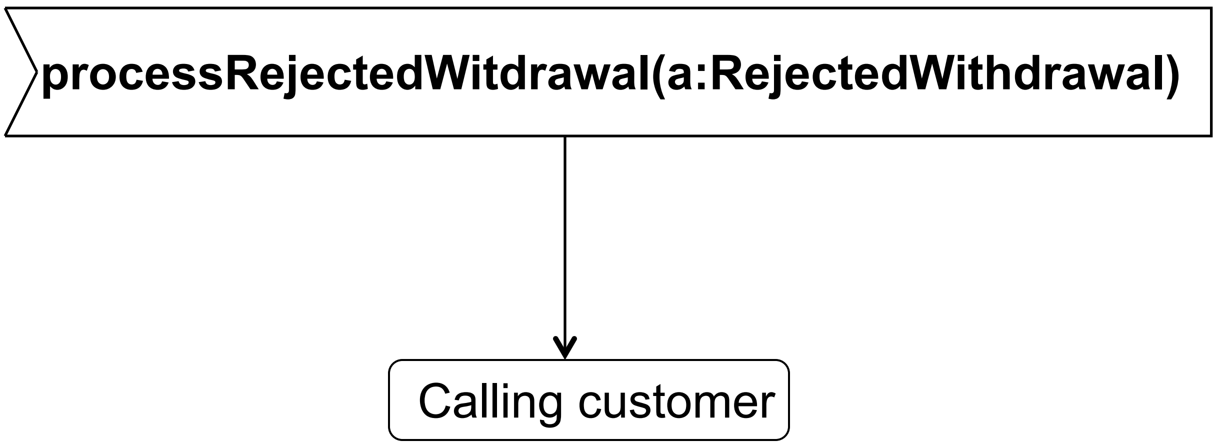


Some of the following examples are taken from:
UML 2 and The Unified Process
Arlow and Neustadt

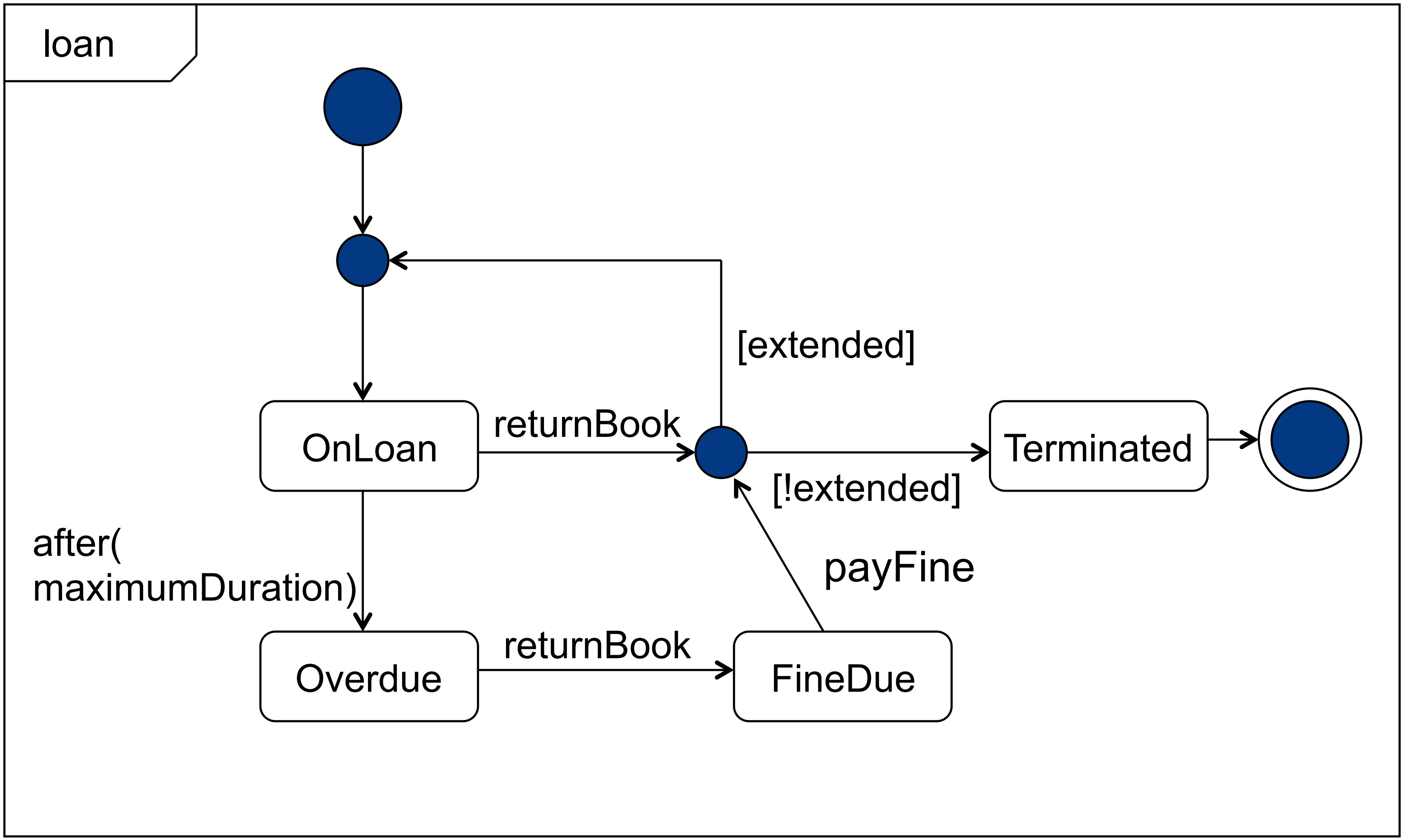
Signal events(1)



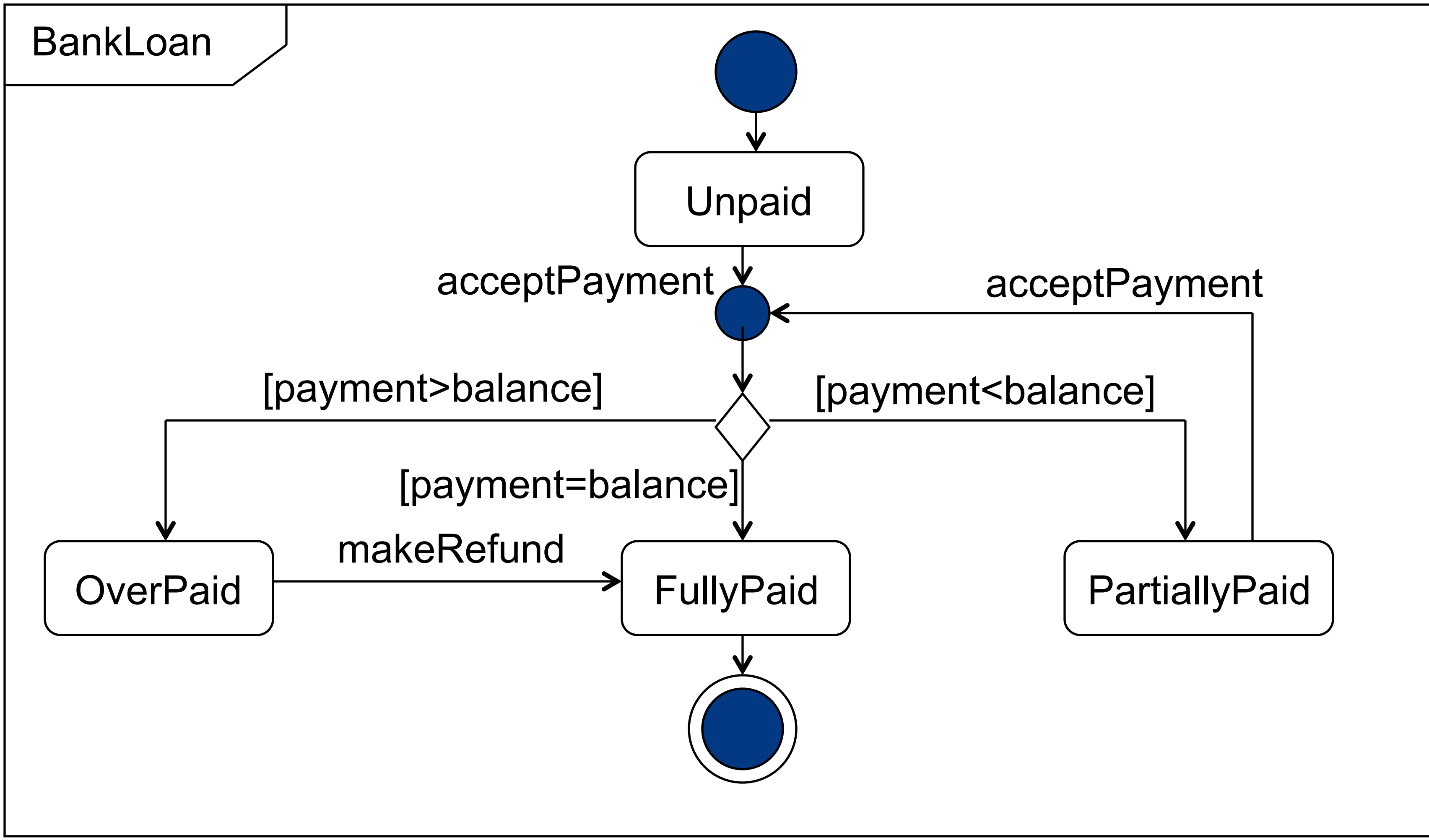
Signal events(2)



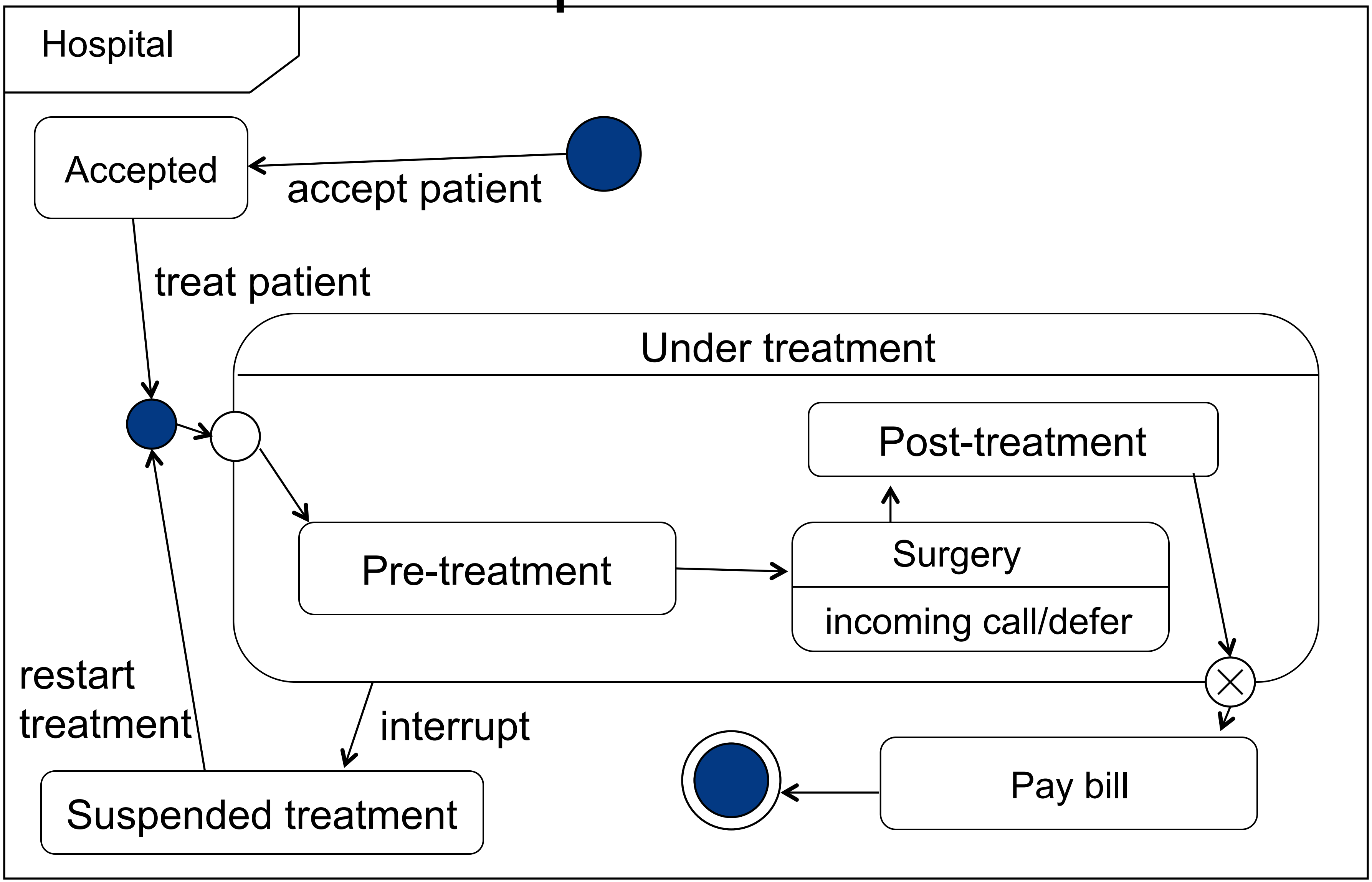
Connection transition



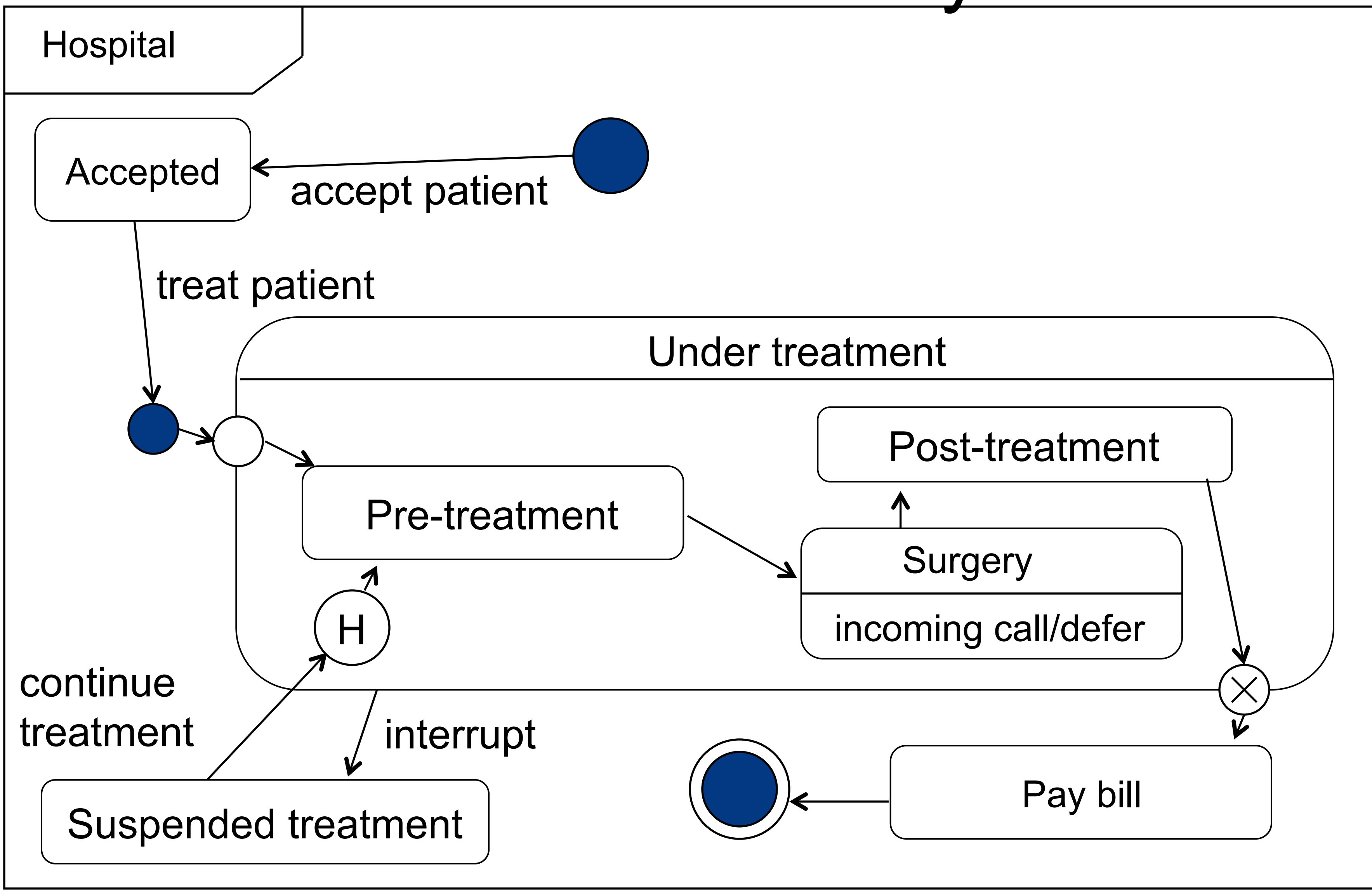
Branching Transitions



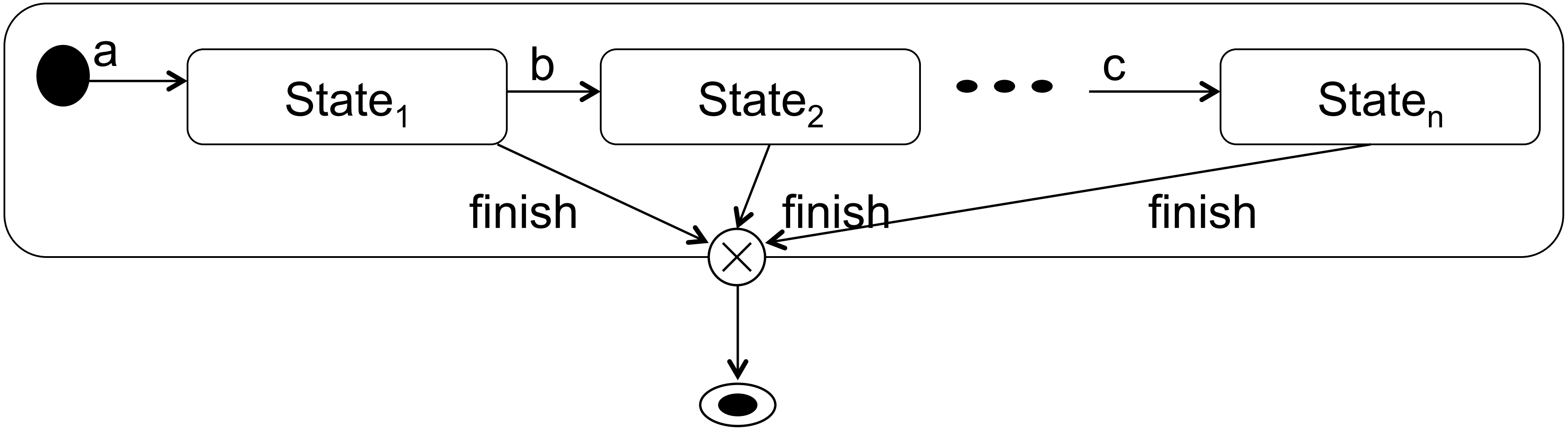
Composite states



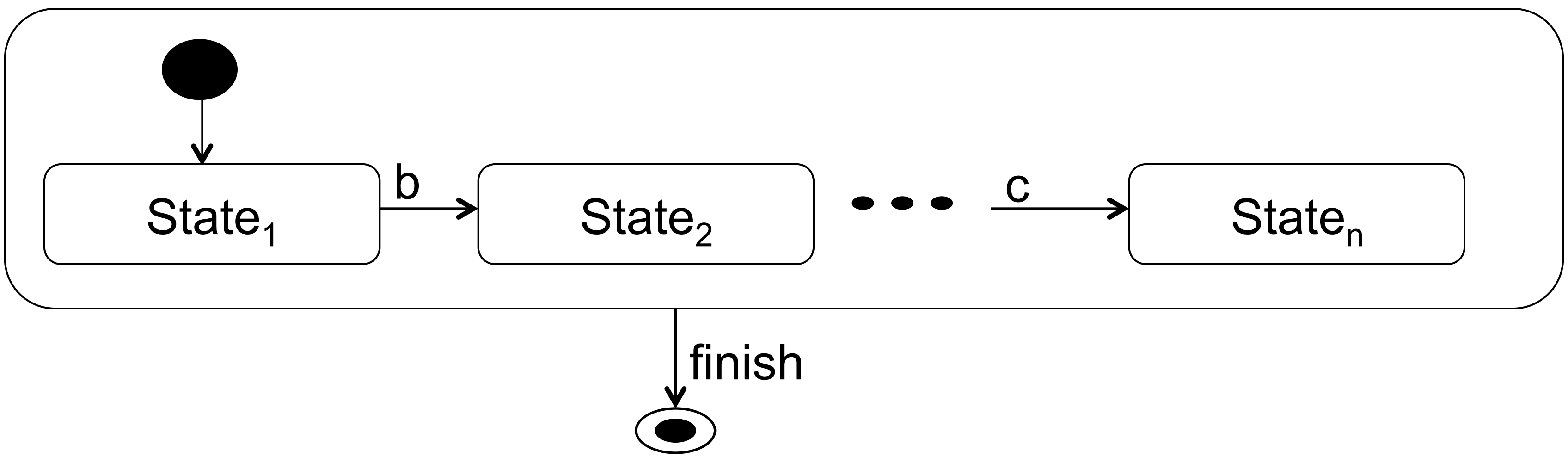
Shallow history



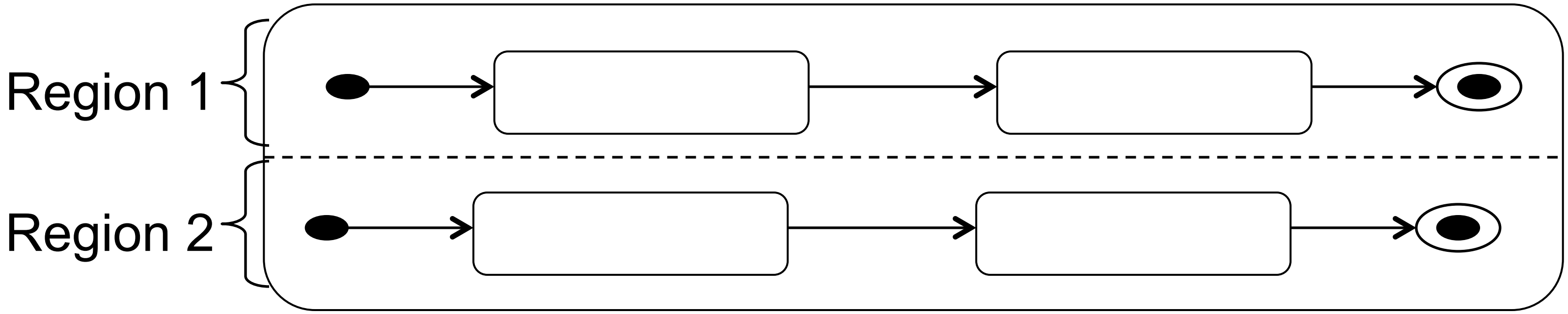
States with Substates



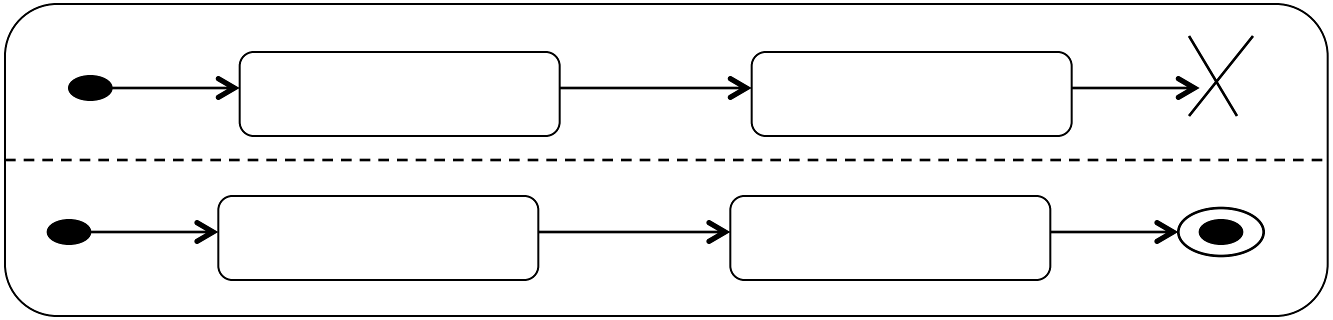
Can be written as:



Composite State



- If region one finishes, then that region will terminate, but region two will continue to execute.



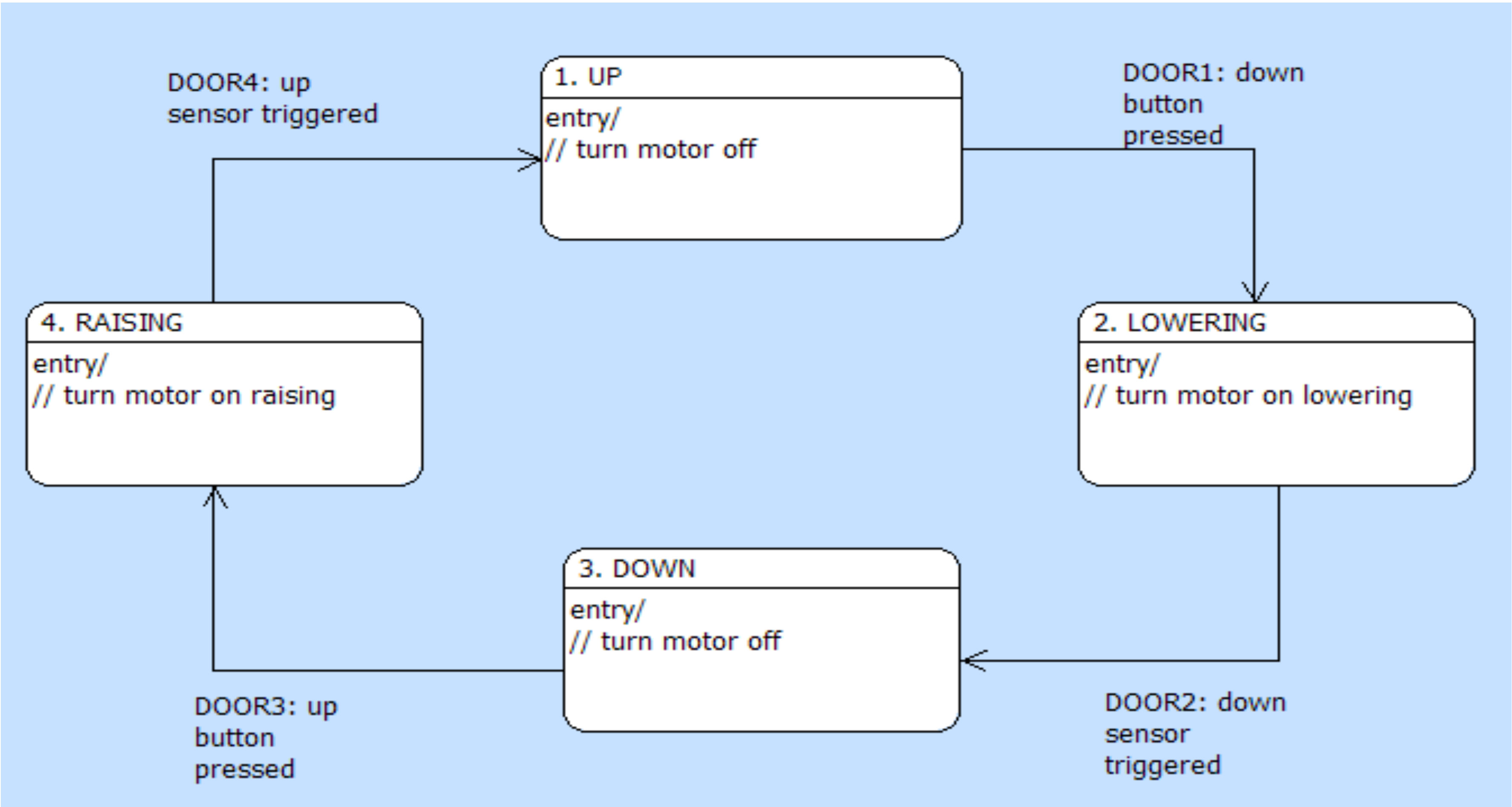
- In this case, if region one terminates first, the whole composite state will stop executing.

Constructing State Machines

- Draw and name the states you know.
- Write a comment: what does this state mean?
- Draw the transitions you know, into or out of each state.
- Do incomplete transitions suggest missing states?
- Define and name the known events.
- Assign an event to each transition; any missing events?
- Do events need to carry event data?
- Check for completeness; add discovered states/transitions.

Checking for completeness

An automatic garage door: two buttons – up & down – and position sensors



Filling the State Transition Table

Events

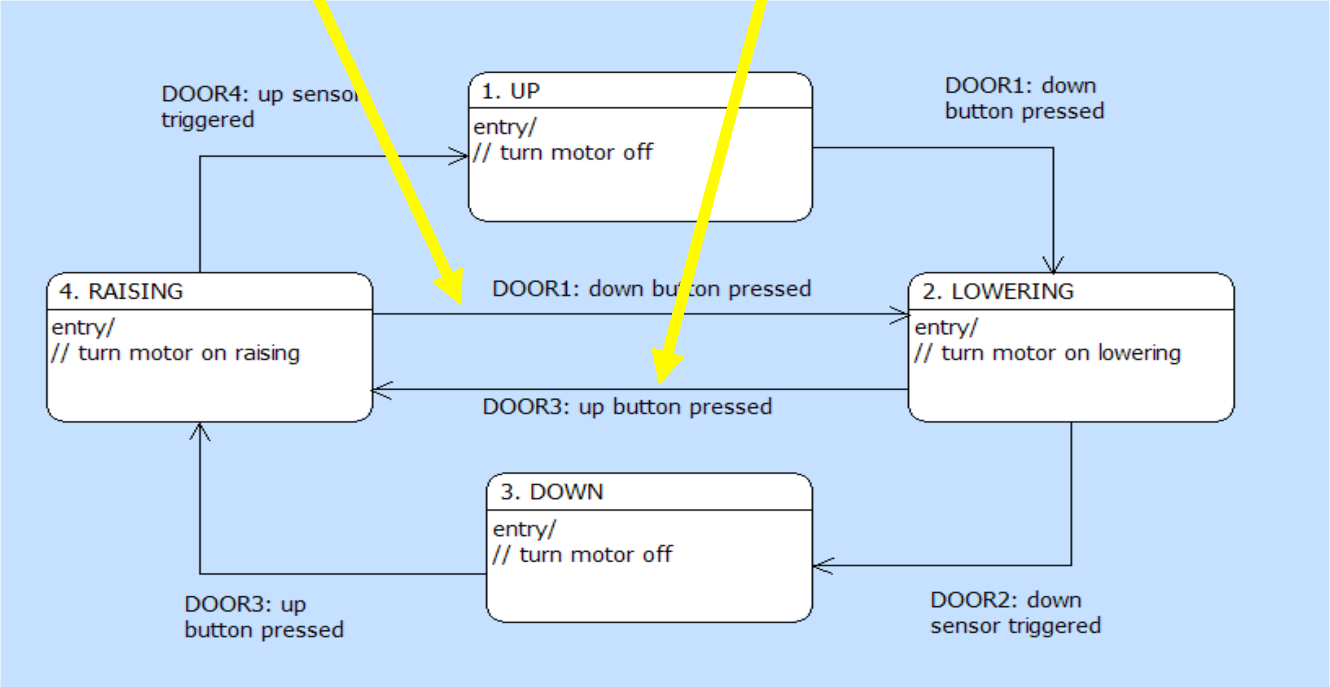
Entry Action

	Down button	Down sensed	Up button	Up sensed	Action
UP	LOWERING				Stop motor
LOWERING		DOWN			Motor down
DOWN			RAISING		Stop motor
RAISING				UP	Motor up

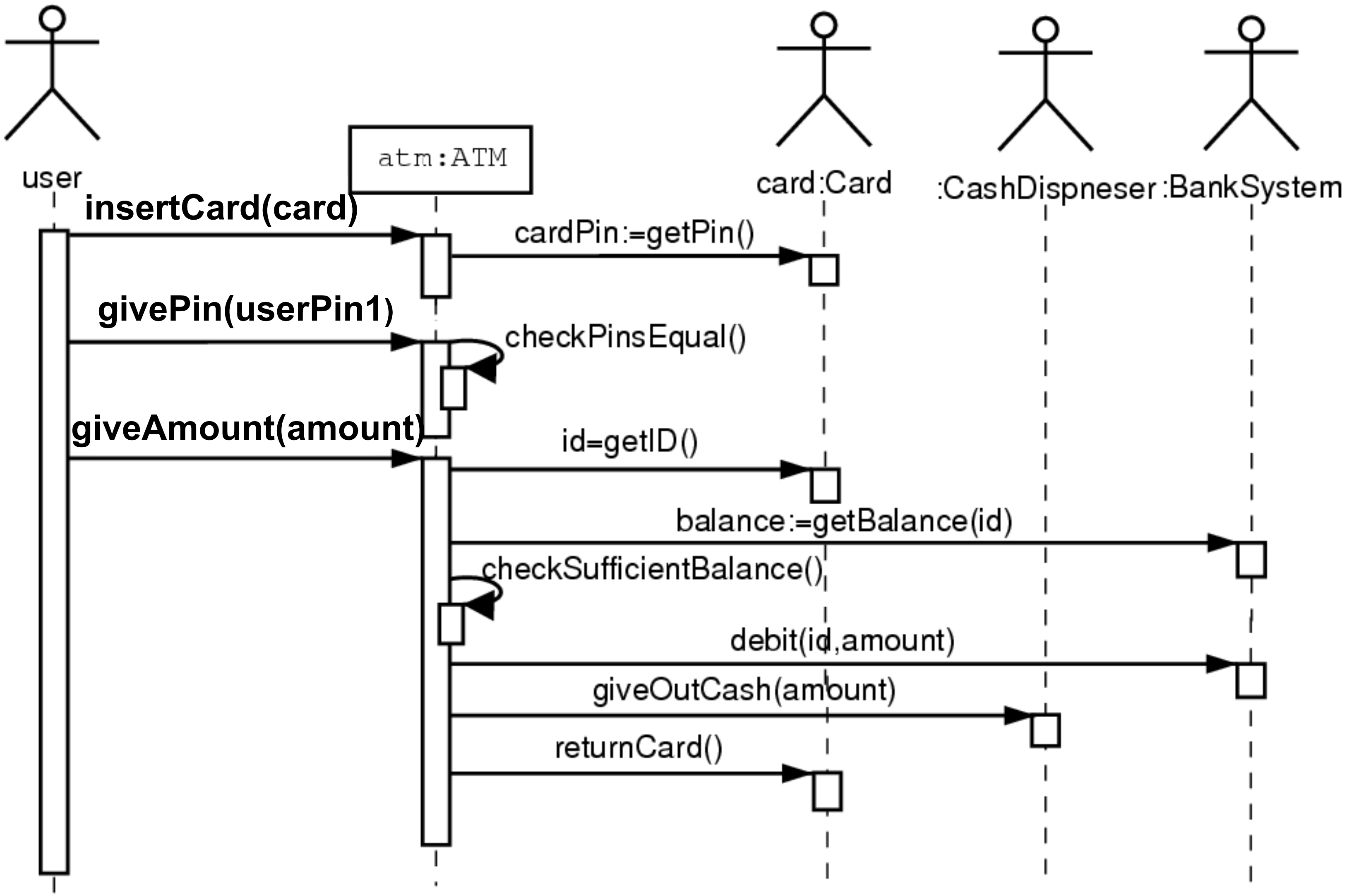
States

What do the empty cells mean?

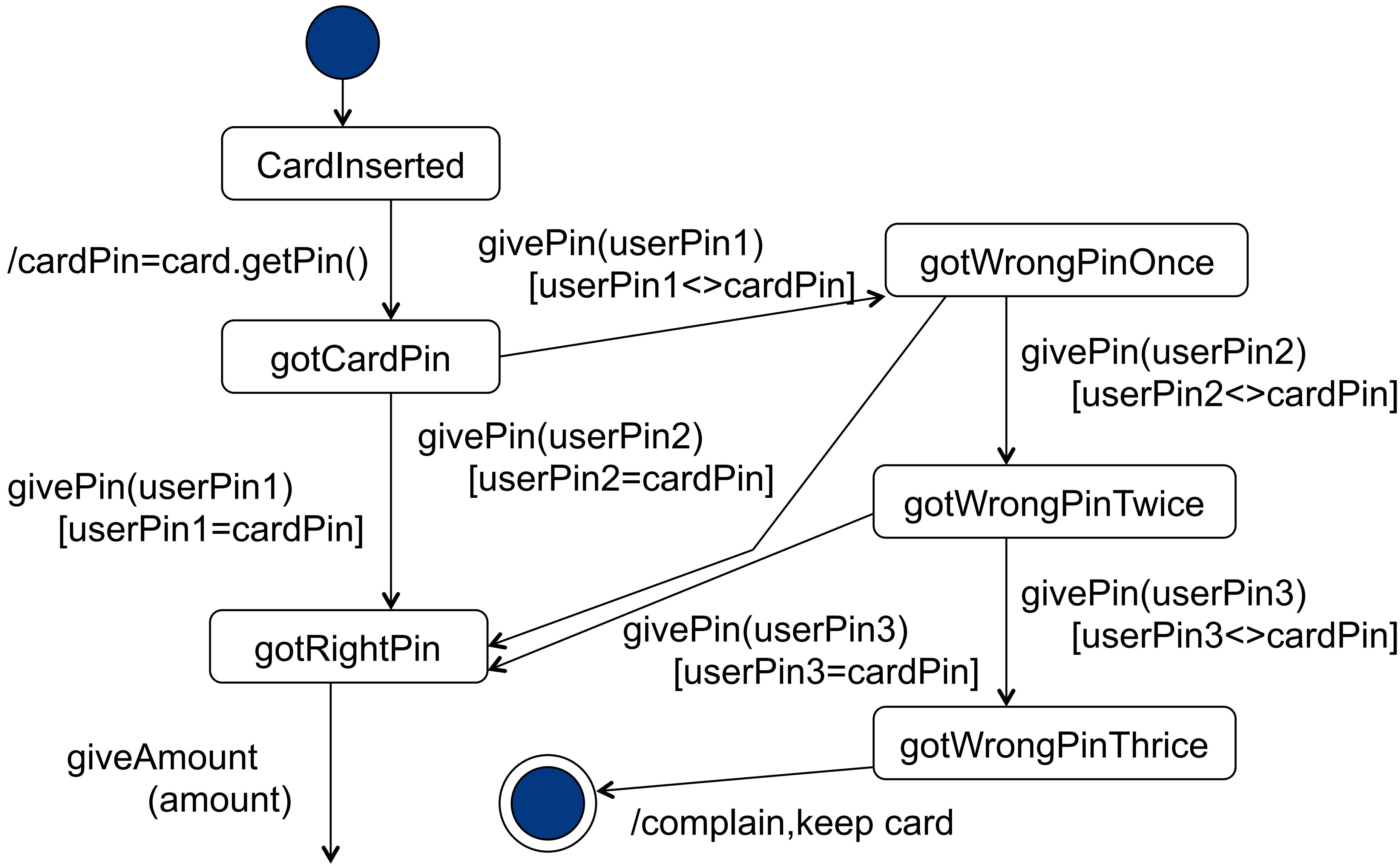
	Down button	Down sensed	Up button	Up sensed	Action
UP	LOWERING	Can't happen	Event Ignored	Can't happen	Stop motor
LOWERING	Event Ignored	DOWN	RAISING	Can't happen	Motor down
DOWN	Event Ignored	Can't happen	RAISING	Can't happen	Stop motor
RAISING	LOWERING	Can't happen	Event Ignored	UP	Motor up



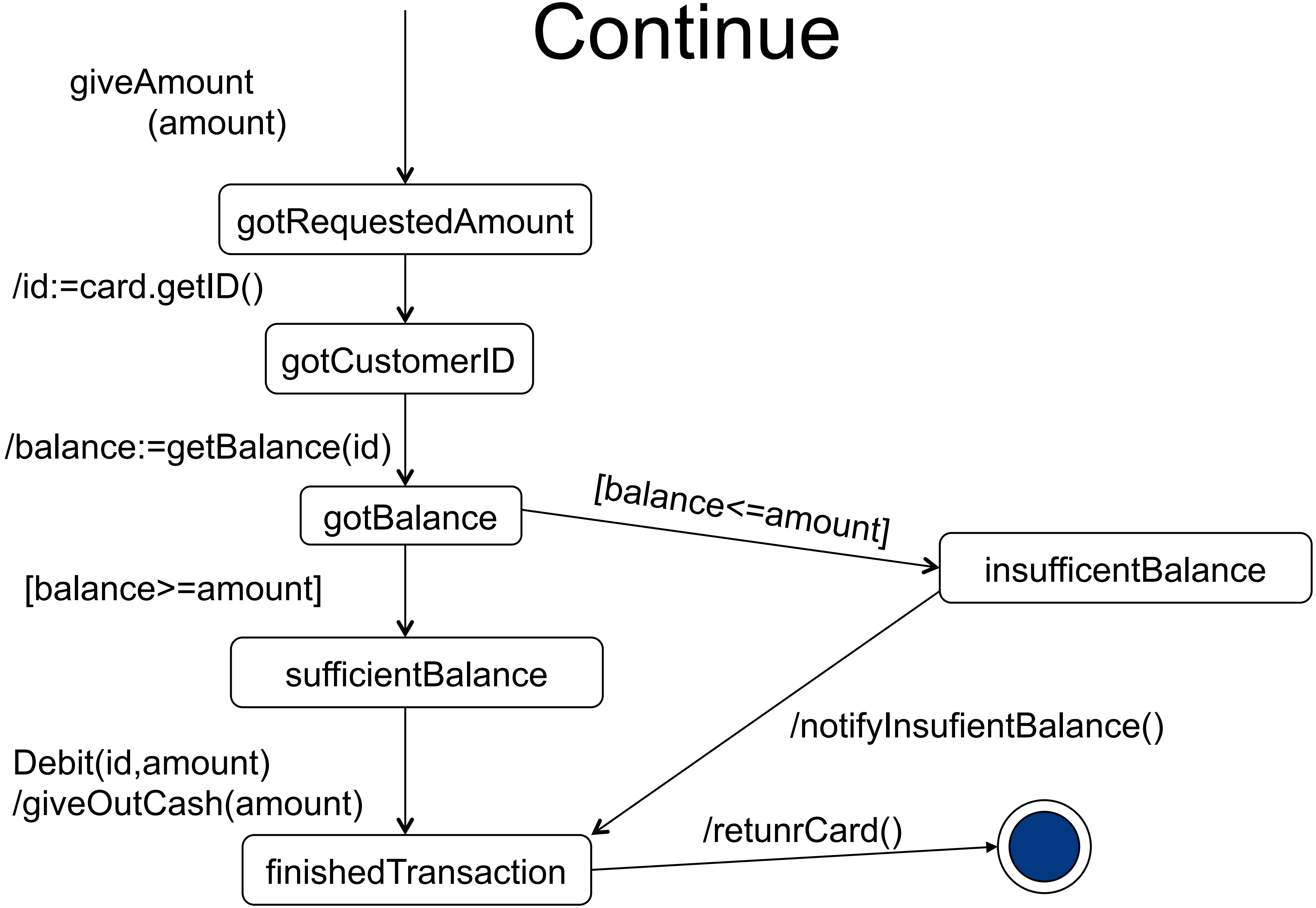
System Sequence Diagram
Withdraw Money



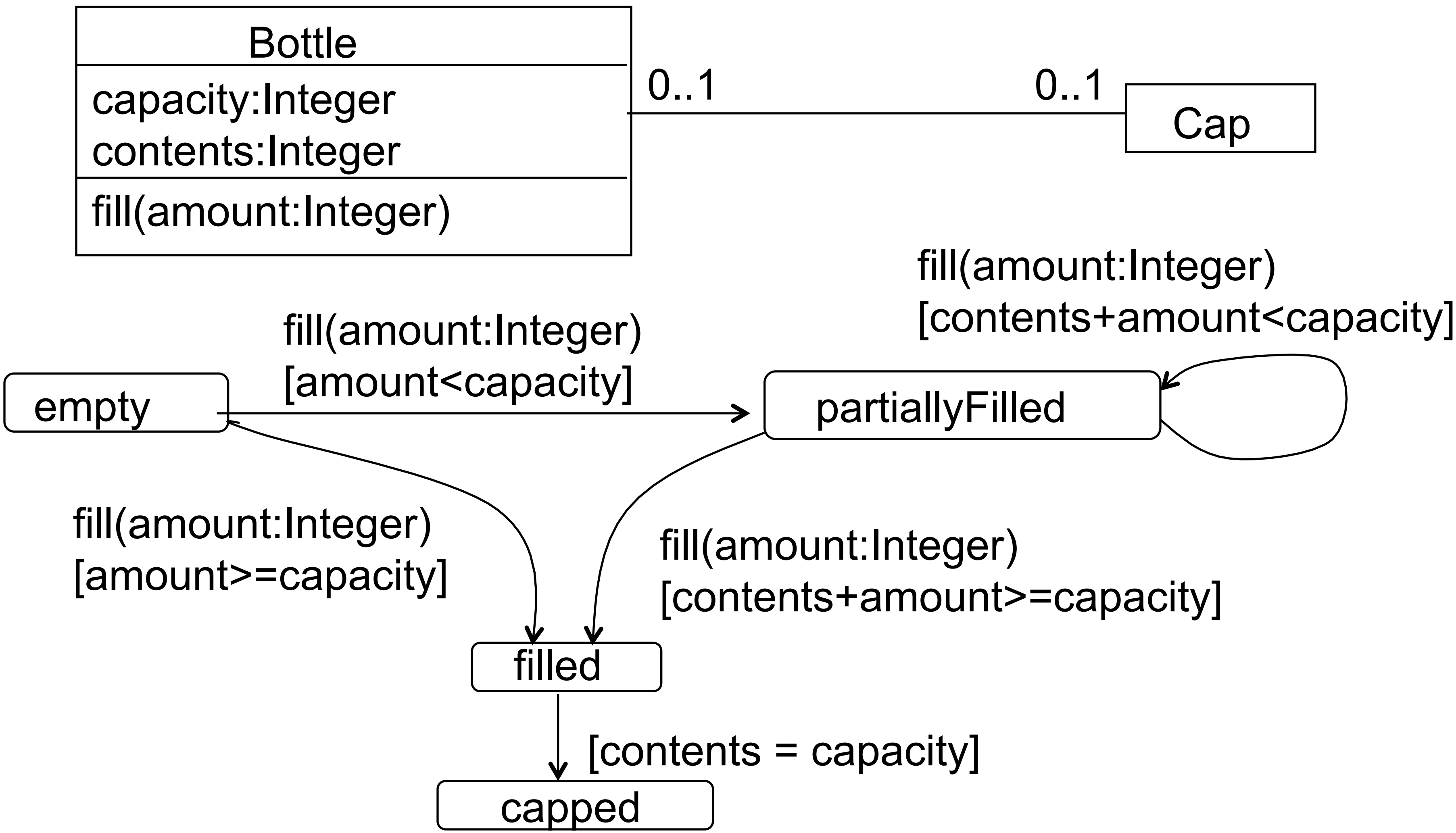
State Chart for Withdraw



Continue

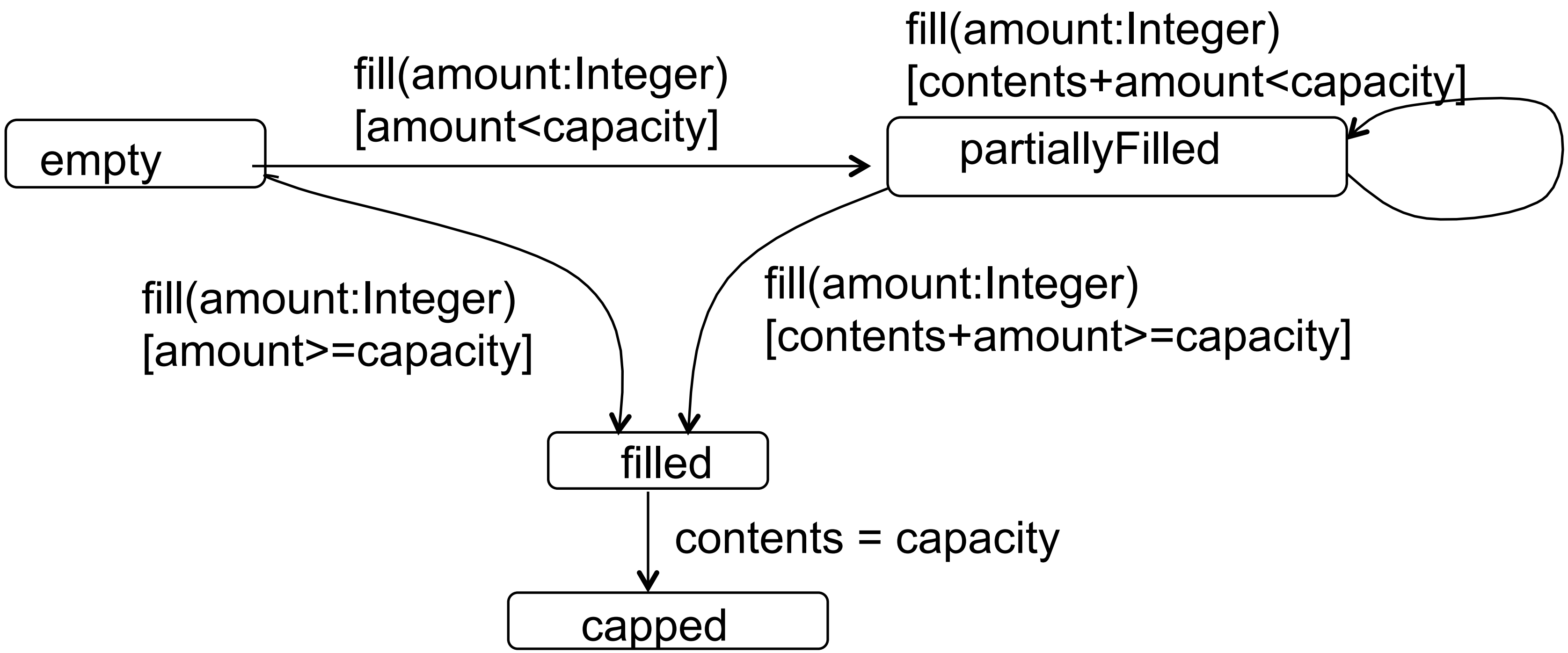

giveAmount
(amount)
gotRequestedAmount
/id:=card.getID()
gotCustomerID
/balance:=getBalance(id)
gotBalance
[balance<=amount]
insufficentBalance
[balance>=amount]
sufficientBalance
/notifyInsufientBalance()
Debit(id,amount)
/giveOutCash(amount)
/retunrCard()
finishedTransaction

State Charts

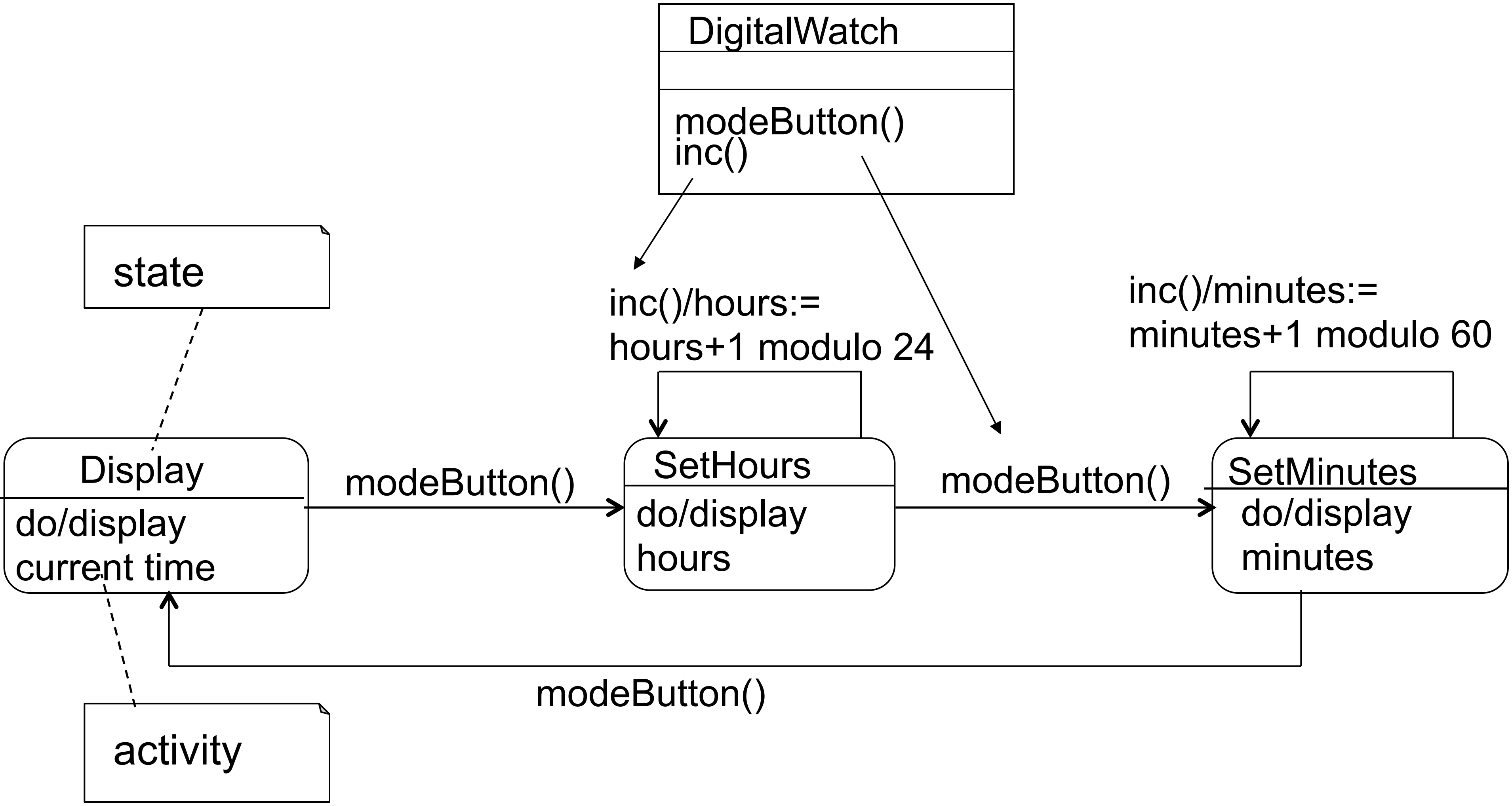


Making Contract

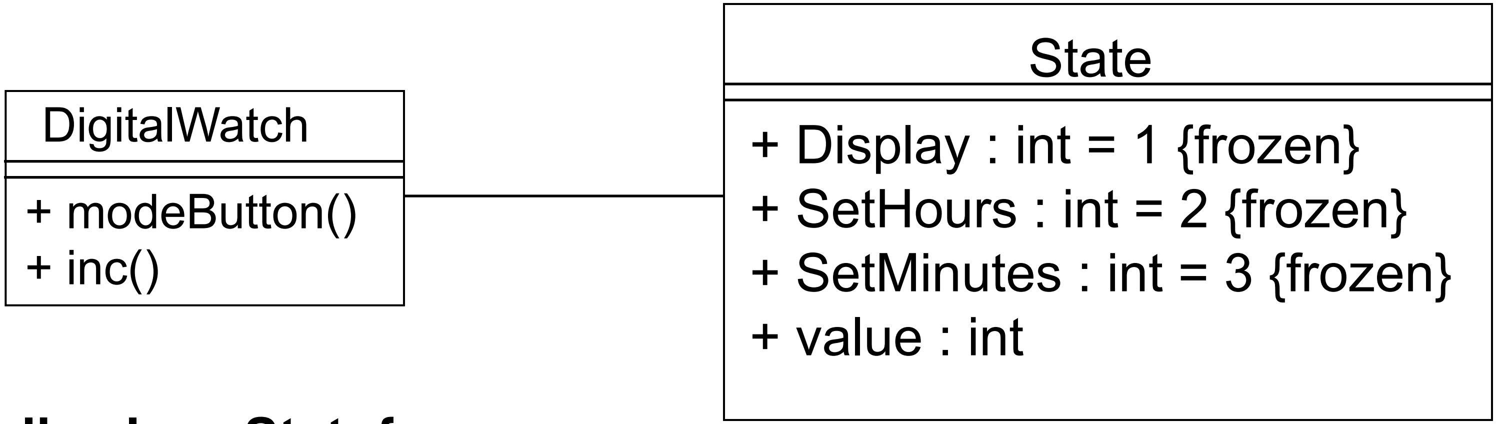
context Bottle::fill(amount:Integer)
pre: not filled and not capped
post: (partiallyFilled and
content@pre + amount < capacity)
or
(filled and contents@pre + amount >= capacity)



Code: DigitalWatch



State




```
public class State{
   public final int Display = 1;
   public final int SetHours = 2;
   public final int SetMinutes = 3;
   public int value;
}
```

DigitalWatch

```
public class DigitalWatch{
    private State state = new State();
    private DigitalDisplay LCD = new DigitalDisplay();

   public DigitalWatch(){
      state.value = state.Display;
      LCD.displayTime();
    }

    public void modeButton() { … }

    public void inc() { … }
}
```

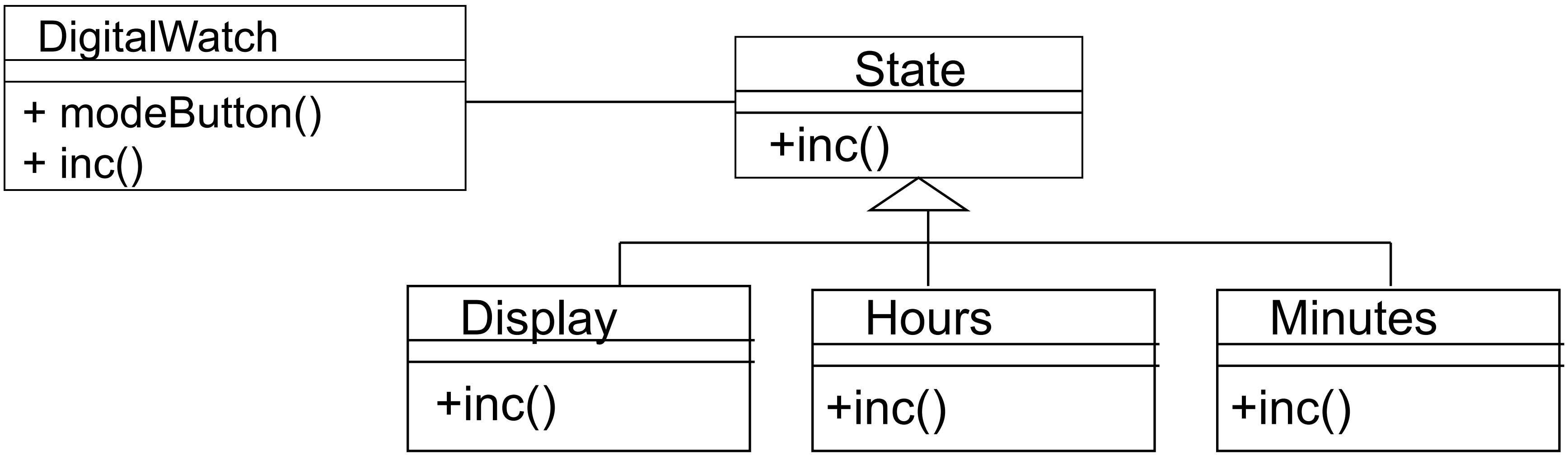
ModeButton

```
public void modeButton() {
      switch (state.value){
          case state.Display :
                        LCD.displayTime();
                        state.value =  state.SetHours;
                        break;
           case state.SetHours:
                        LCD.displayHours();
                        state.value = state.SetMinutes;
                        break;
            case state.SetMinutes:
                       LCD.displayTime();
                       state.value = state.Display;
                       break;
        }
}
```

Inc

```
public void inc() {
        switch (state.value){
            case state.Display : break;
            case state.SetHours: LCD.incHours();
                                     break;
            case state.SetMinutes: LCD.incMinutes();
                                       break;
         }
}
```

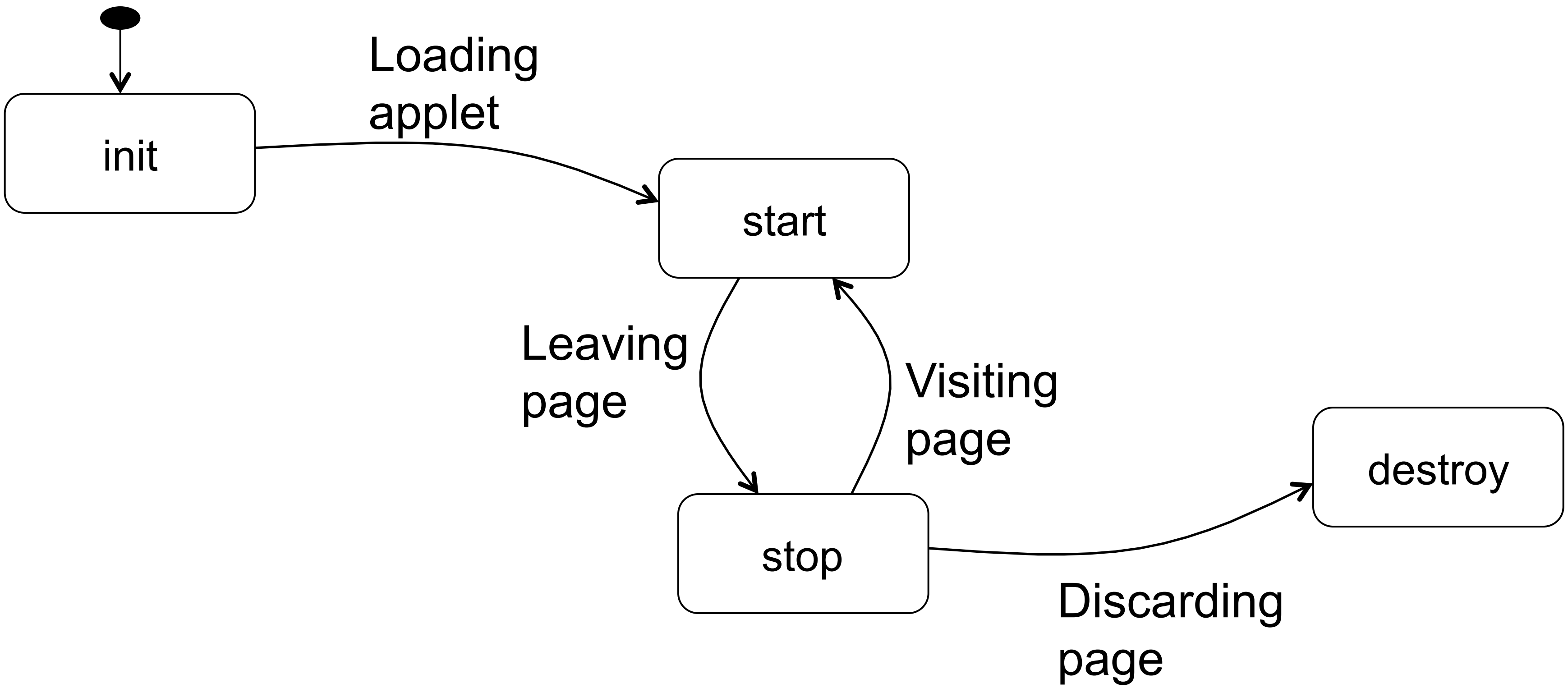
Design Pattern: State



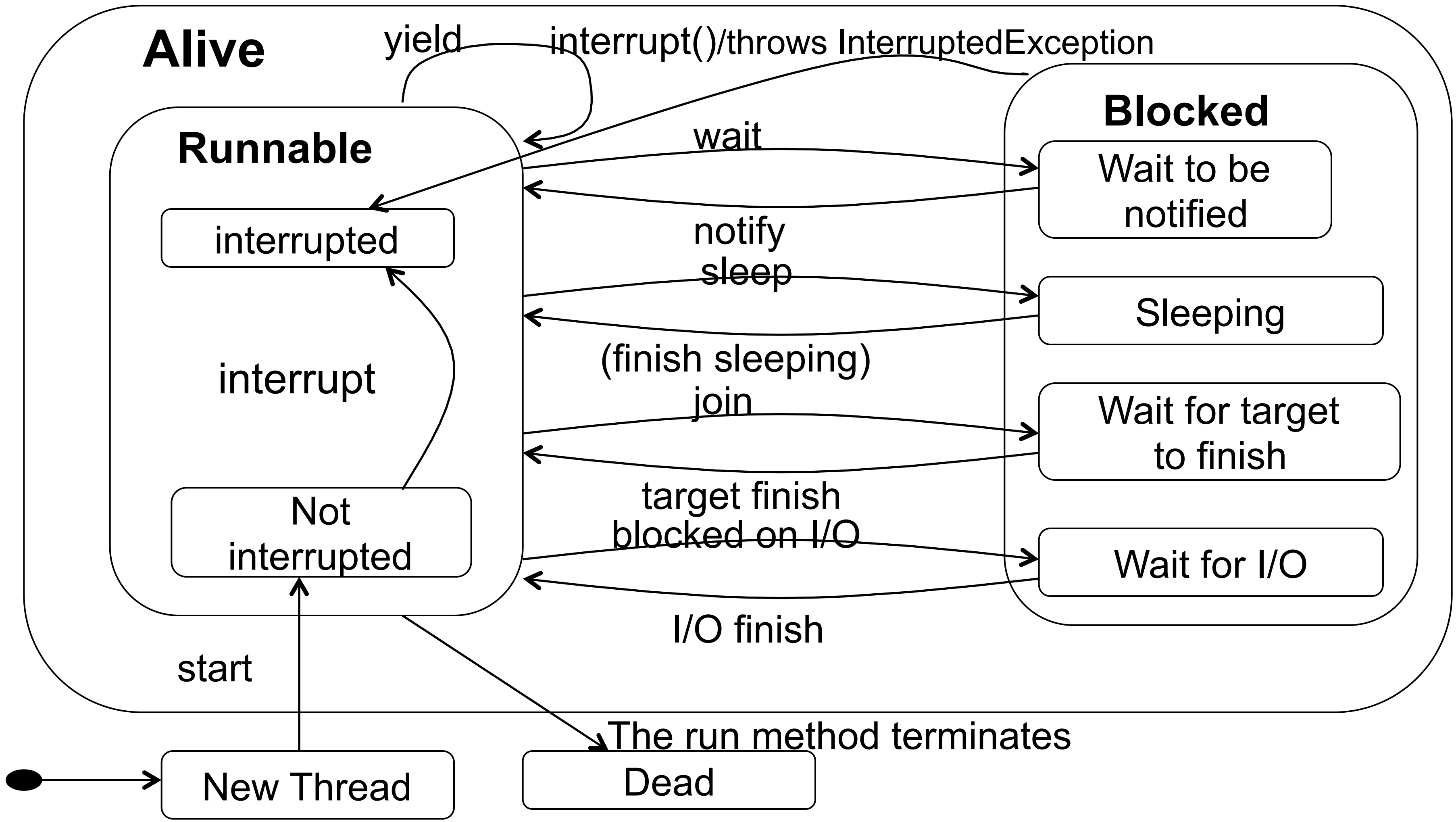
Comment:
This is more object oriented!

Two examples to show the power of state charts

The life cycle of an applet



Thread states



public final boolean isAlive()

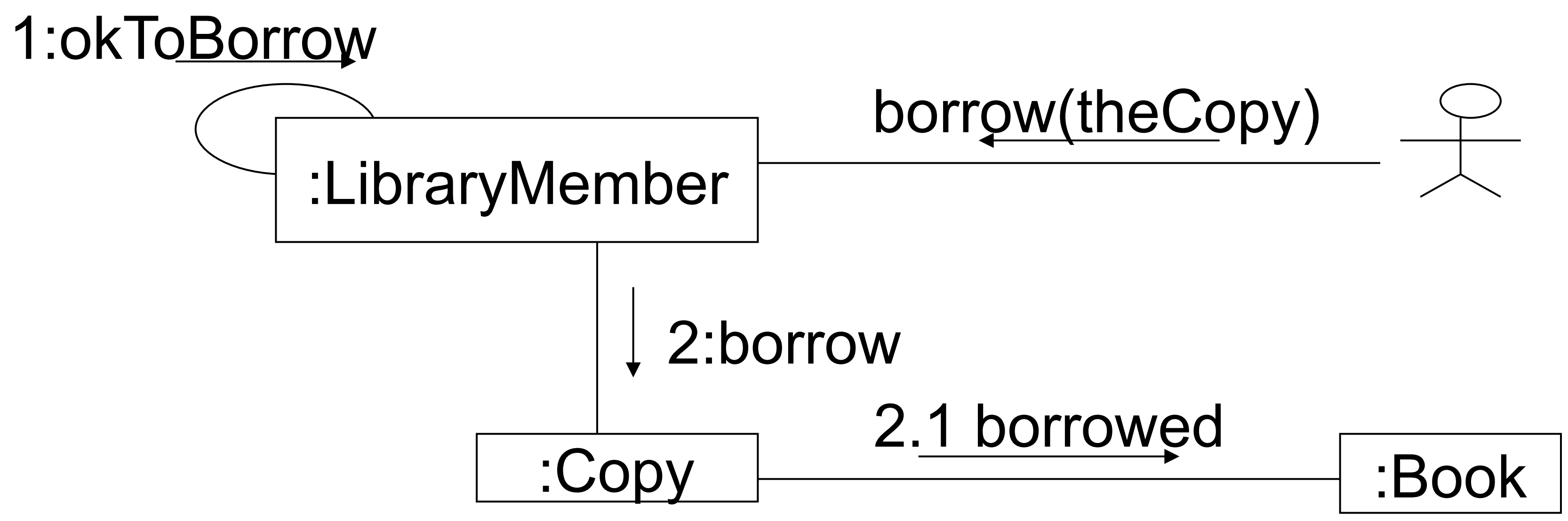
A thread is alive if it is in the state ”Runnable” or ”Blocked” .

Appendix

Interaction Diagrams in UML2

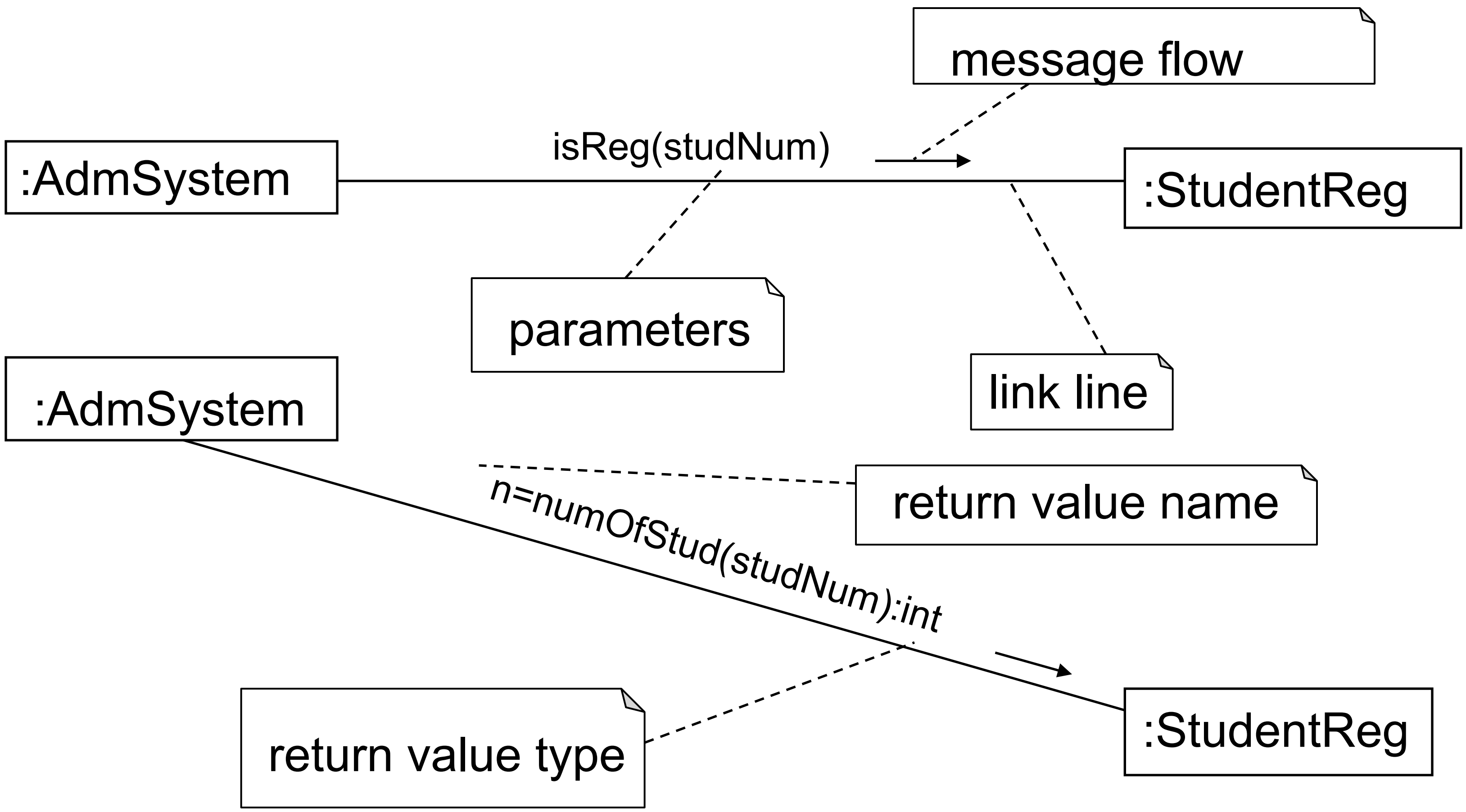
- There are four different kinds of interaction diagrams:
  - Sequence diagrams
  - Communication diagrams (formerly known as collaboration diagrams)
  - Interaction overview diagrams (combination of activity and sequence diagrams)
  - Timing diagrams (not treated in this course)

Example



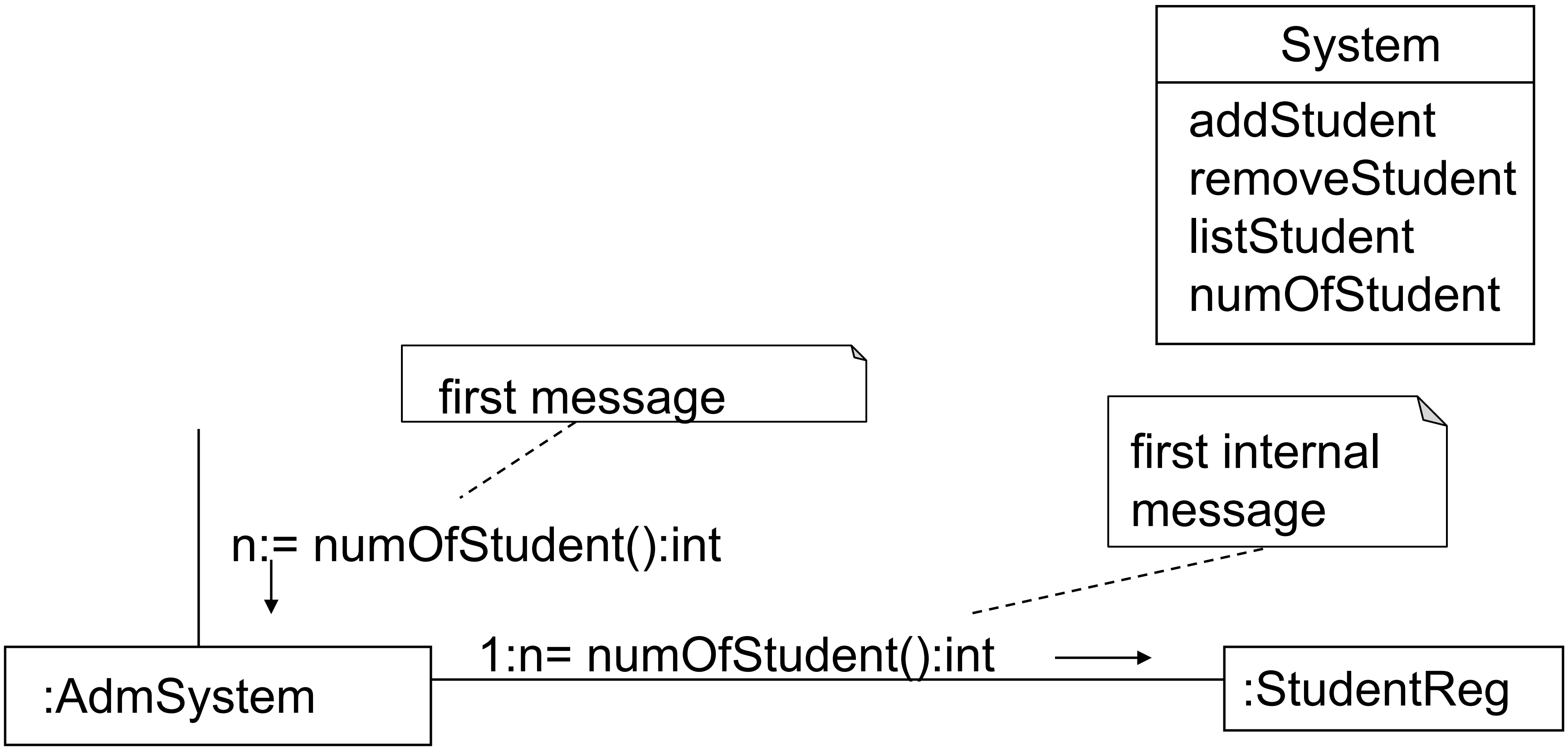
- Communication diagrams are usually more concise than sequence diagrams
- But: They are often considered harder to read
- In UML2, communication diagrams are far less powerful than sequence diagrams

Message

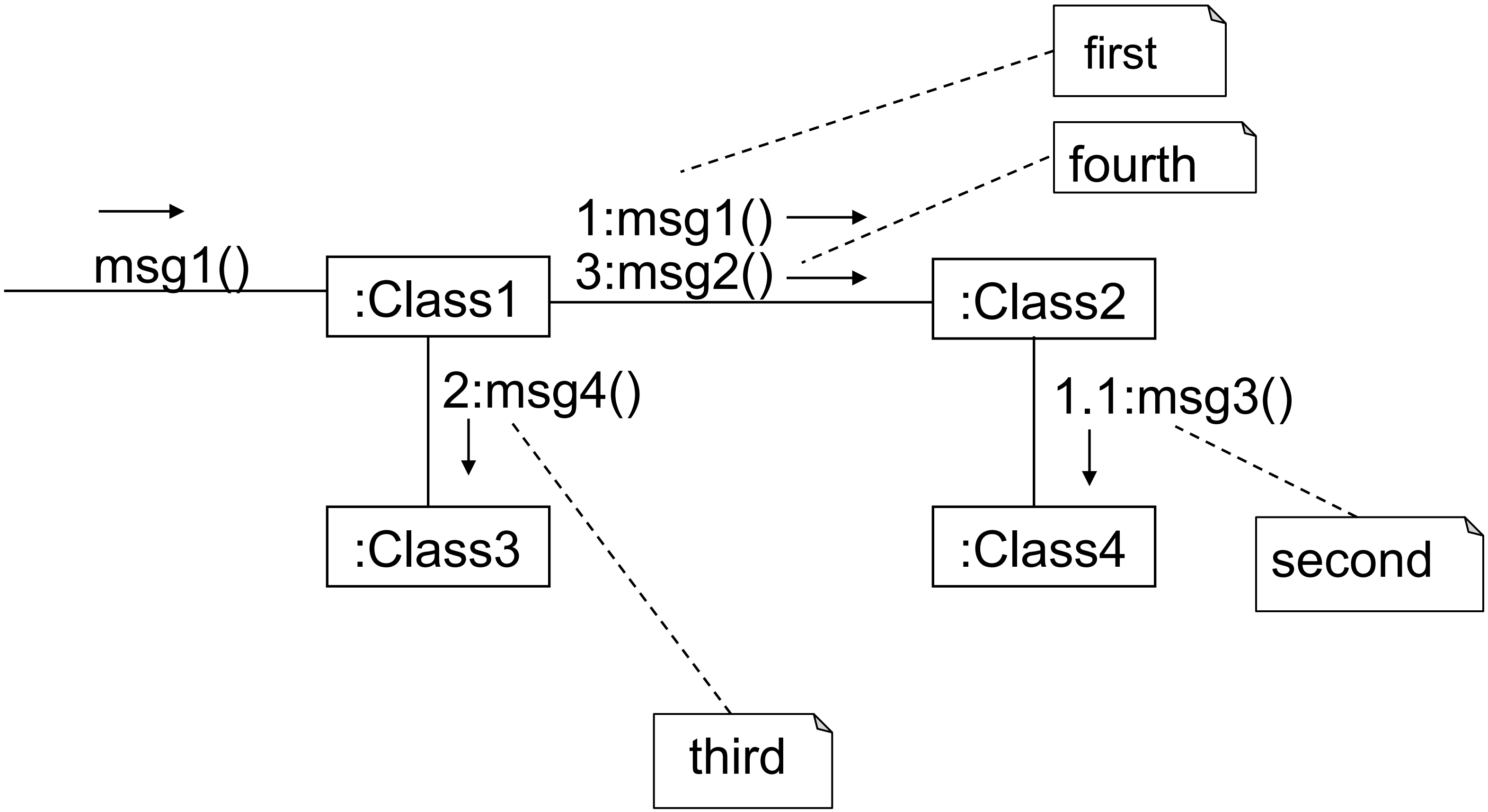

message flow
:AdmSystem
isReg(studNum)
:StudentReg
parameters
link line
:AdmSystem
return value name
n=numOfStud(studNum):int
return value type
:StudentReg

Start-message

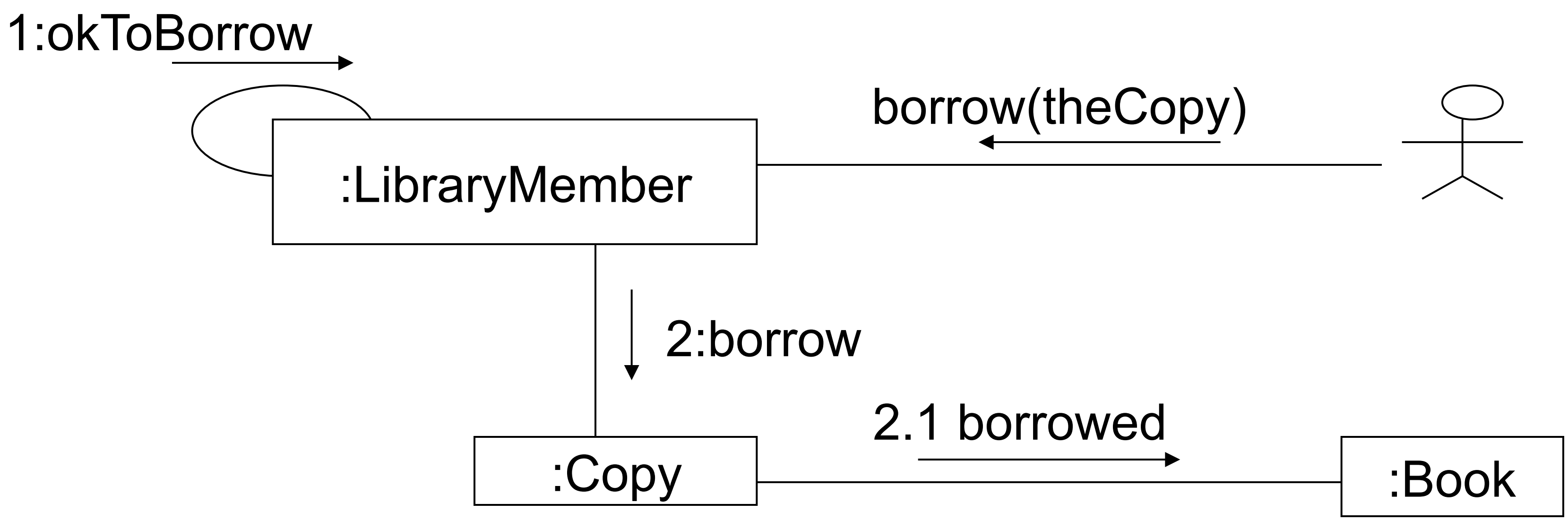
- Can start with a system call. The system operation can be found in the system class.



Sequence numbering

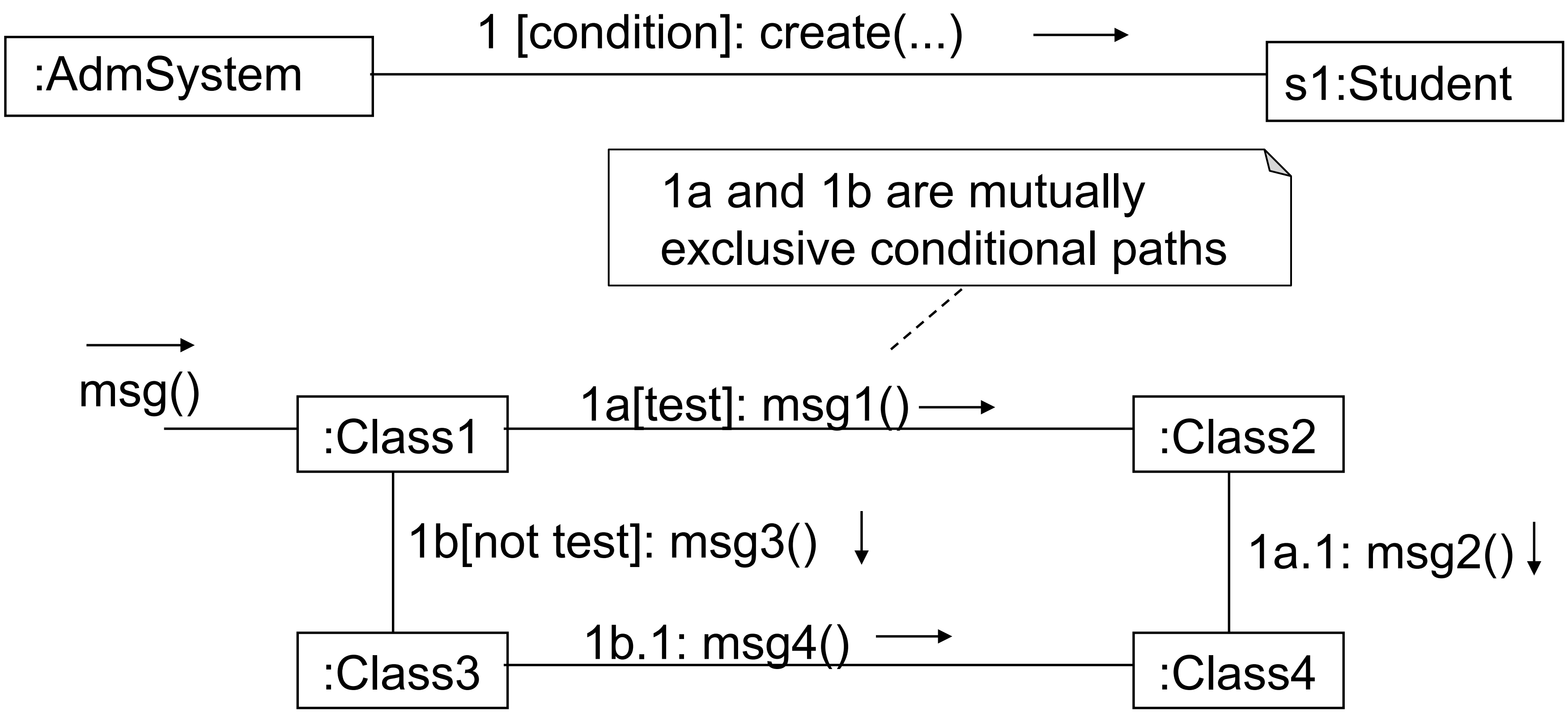


Example

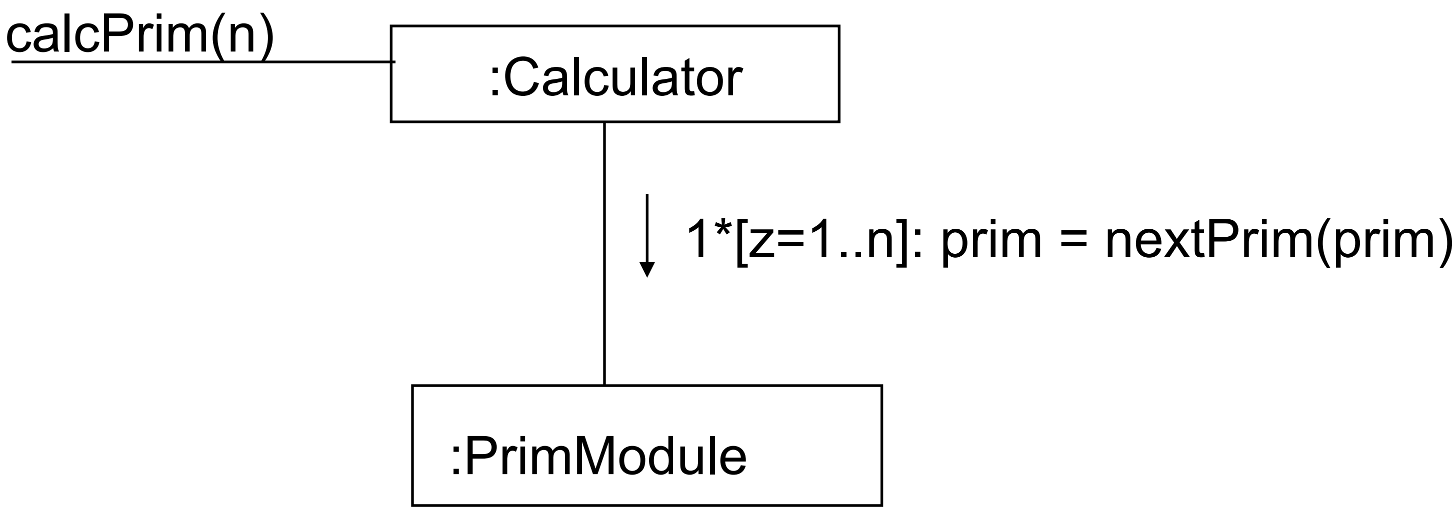

1:okToBorrow
borrow(theCopy)
:LibraryMember
2:borrow
2.1 borrowed
:Copy
:Book

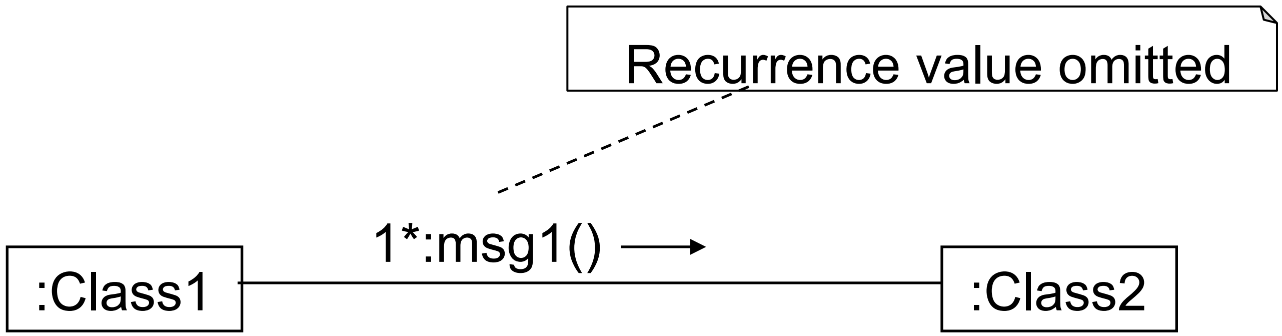
Create

Conditions


1 [condition]: create(...)
:AdmSystem
s1:Student
1a and 1b are mutually exclusive conditional paths
msg()
:Class1
1a[test]: msg1()
:Class2
1b[not test]: msg3()
1a.1: msg2()
:Class3
1b.1: msg4()
:Class4

Iterations


calcPrim(n)
:Calculator
1*[z=1..n]: prim = nextPrim(prim)
:PrimModule


Recurrence value omitted
1*:msg1()
:Class1
:Class2

Class methods