UNIVERSITY OF HULL

12: UML: Modelling S/W Behaviour

2019 - 2020

500084 - System Analysis, Design, and Process

Module Review Themes

+ve

- Labs – focus on work done – useful resources
- Supportive
- Lecture structure
- UML and Modeling
- Clarity in ACW

-ve

- Needs More time, Overlap of content
- More Examples
- Timetabling - early Starts
- Group forming

Module Review Themes

Ask for more support in the Lab (from me and Demonstrators)

Come and see me outside of labs for further support

Will being example of final documents / walk-throughs up into lecture.

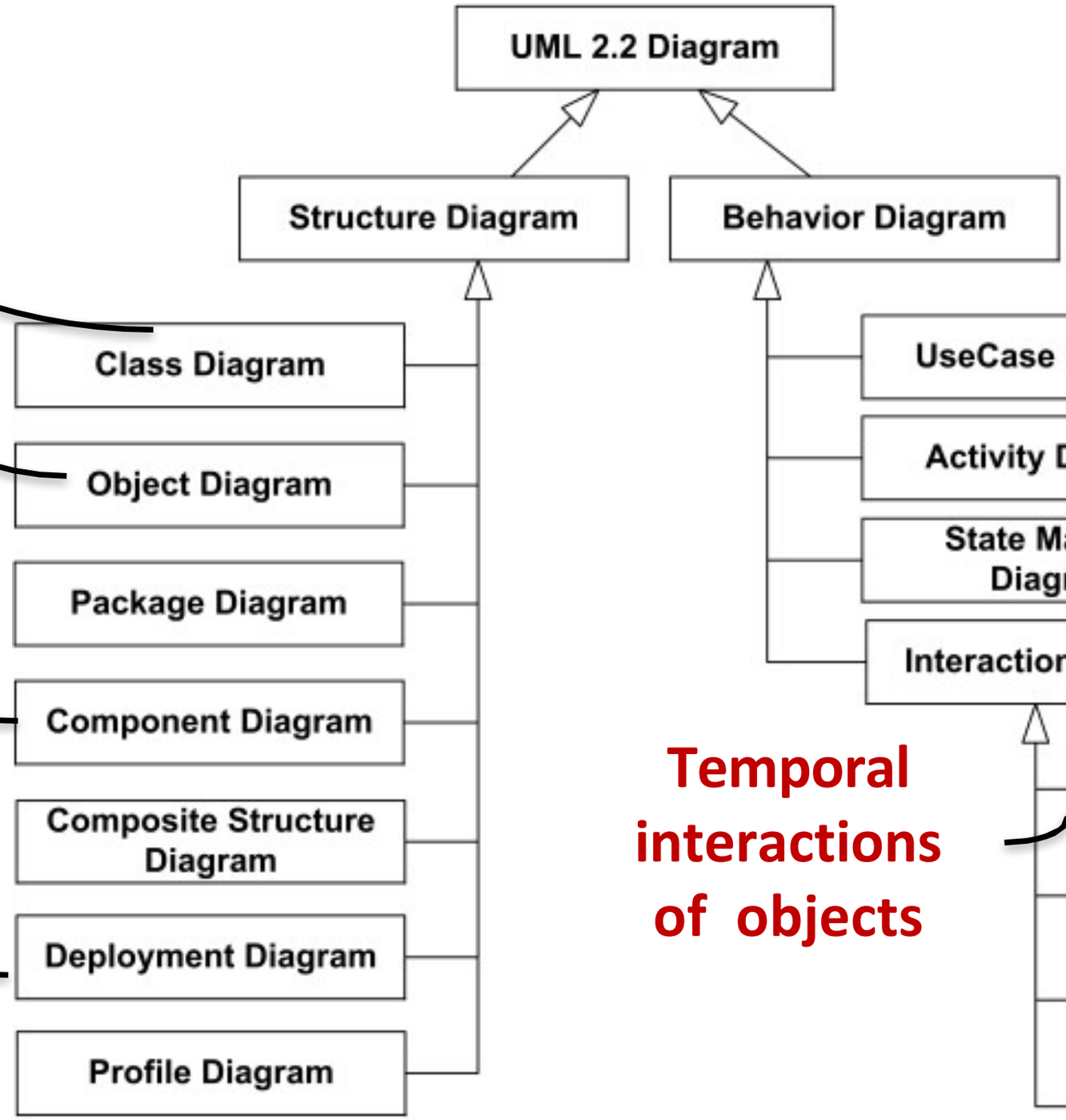
Exam Briefing next week.

Class Static Structure

Objects and their relationships

Physical components

Installation on Hardware.

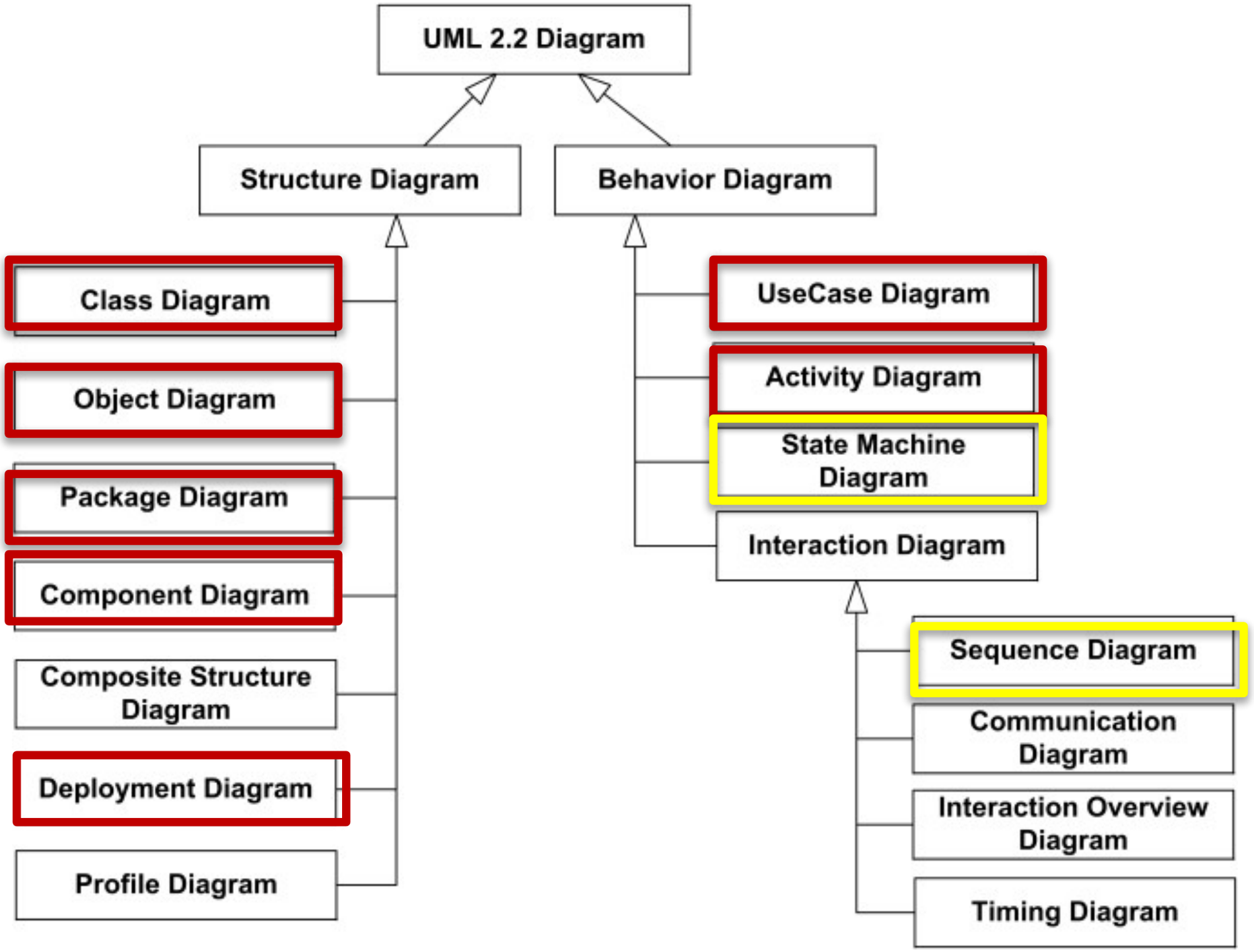


Systems functions from the user view.

Behaviour in terms of actions

Behaviour in terms of states

Temporal interactions of objects


UML 2.2 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Component Diagram
Composite Structure Diagram
Deployment Diagram
Profile Diagram
UseCase Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

Behavioural Modelling

- UML provides facilities for modelling the behaviour of a system

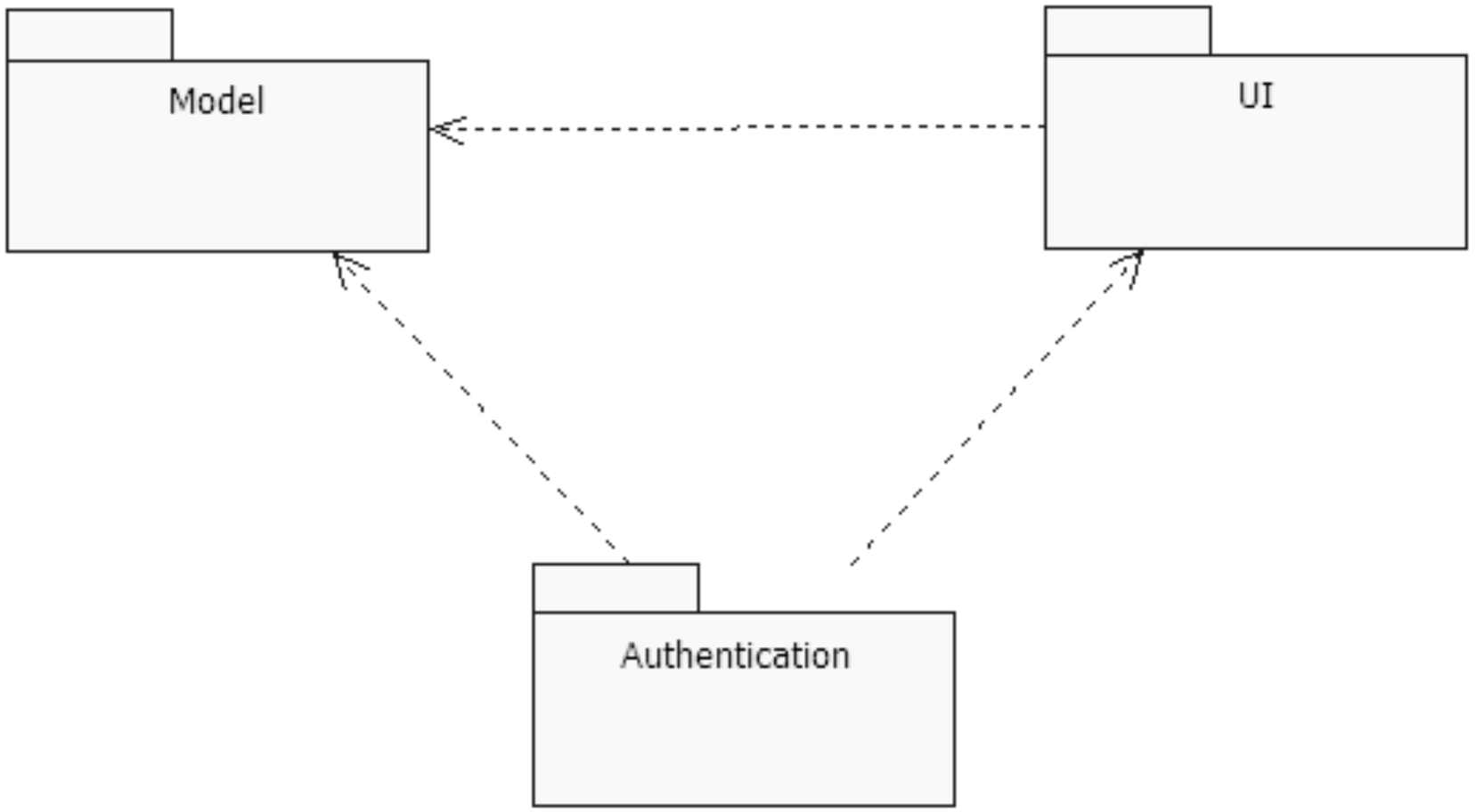
  –(Activity Diagrams) (Business and User)

  –Sequence Diagrams

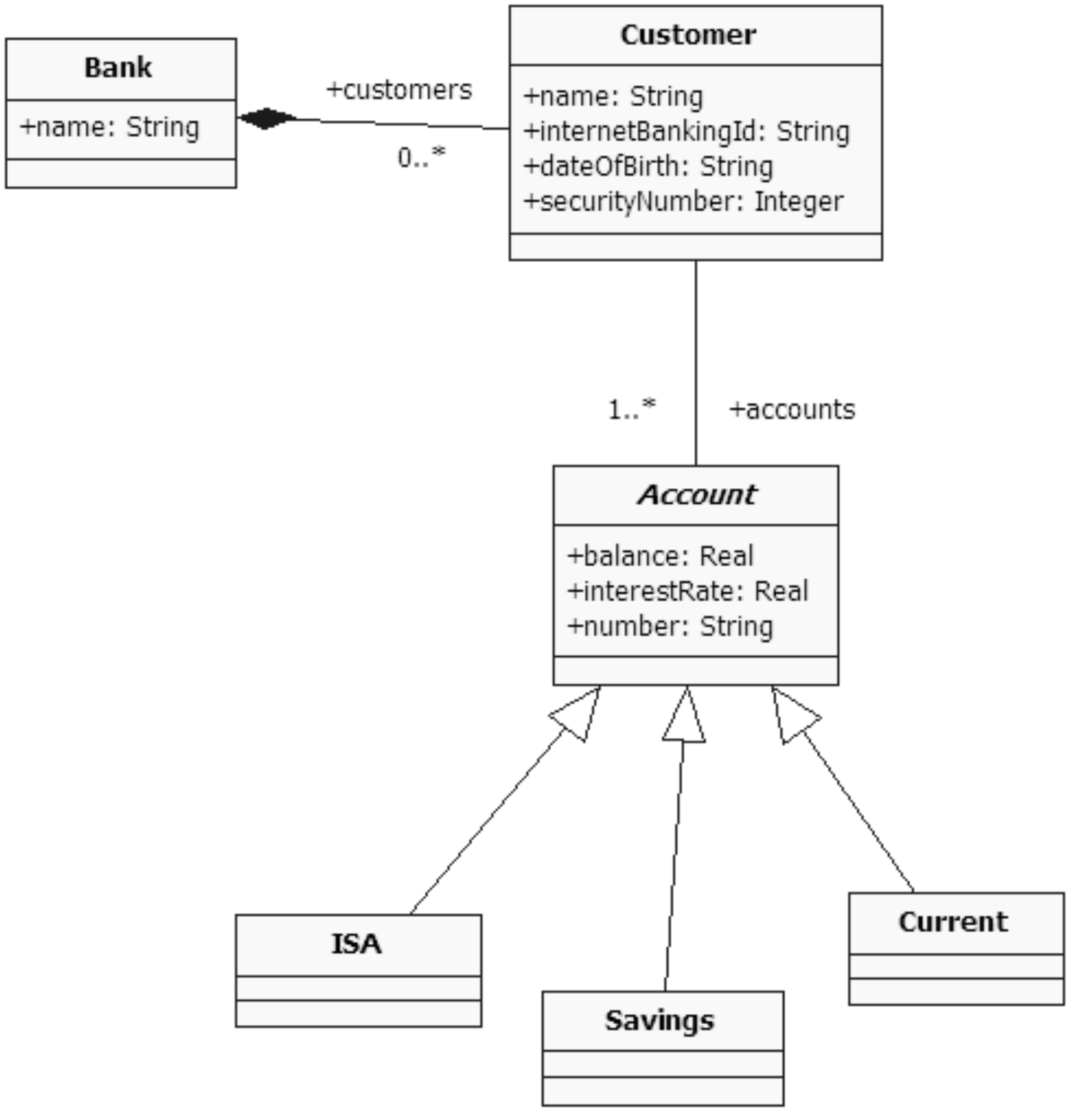
  –State Diagrams

Reality check: UML is NOT a programming language

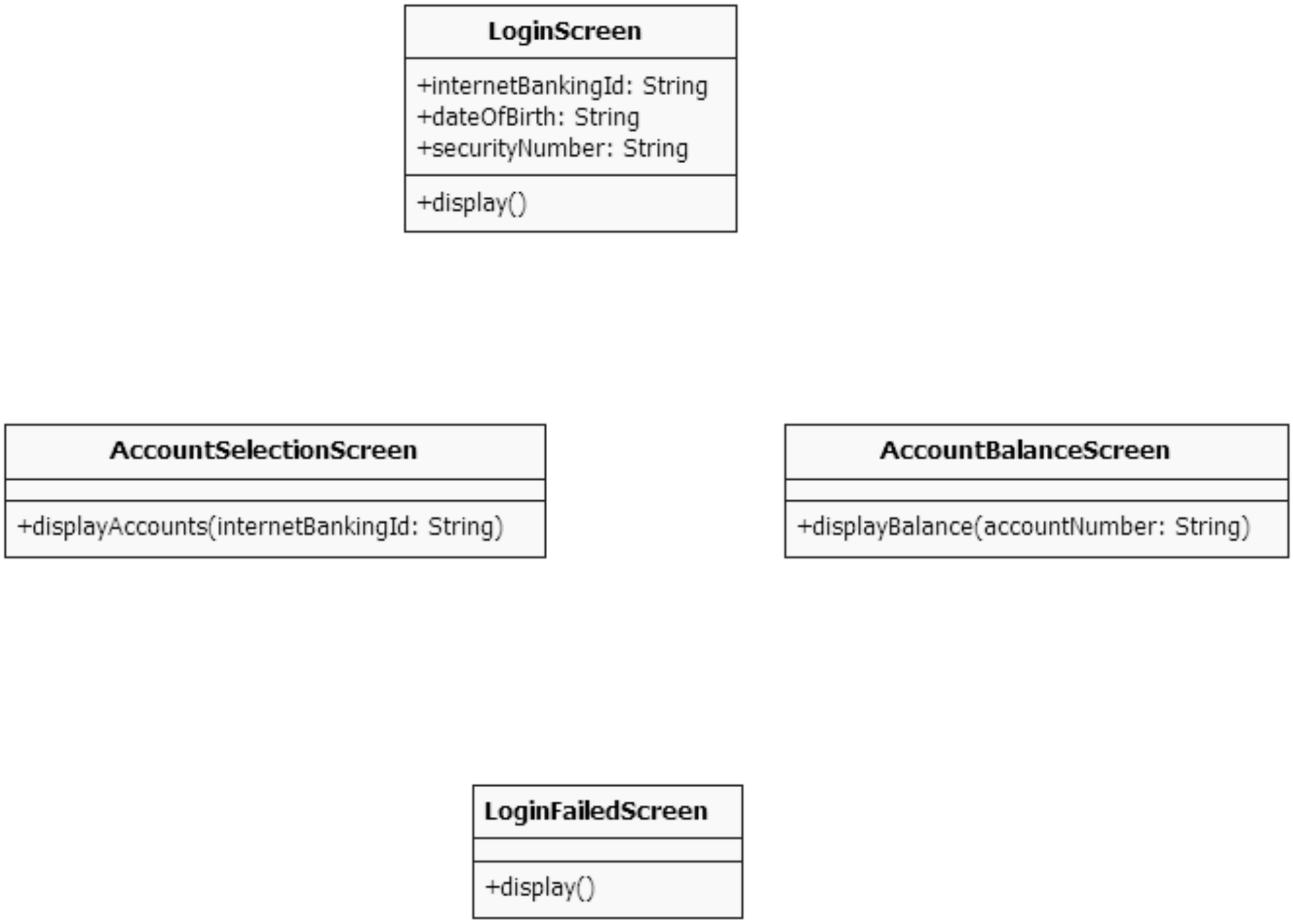
Reference System: Bank



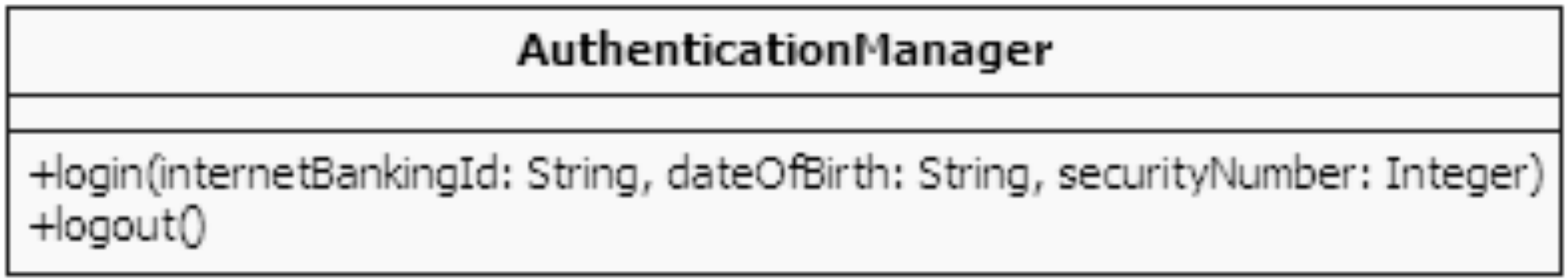
Reference System: Bank Model



Reference System: Bank UI



Reference System: Authentication



Sequence Diagrams

Object View of the System

- A collection of objects work together to achieve the goal of the system.

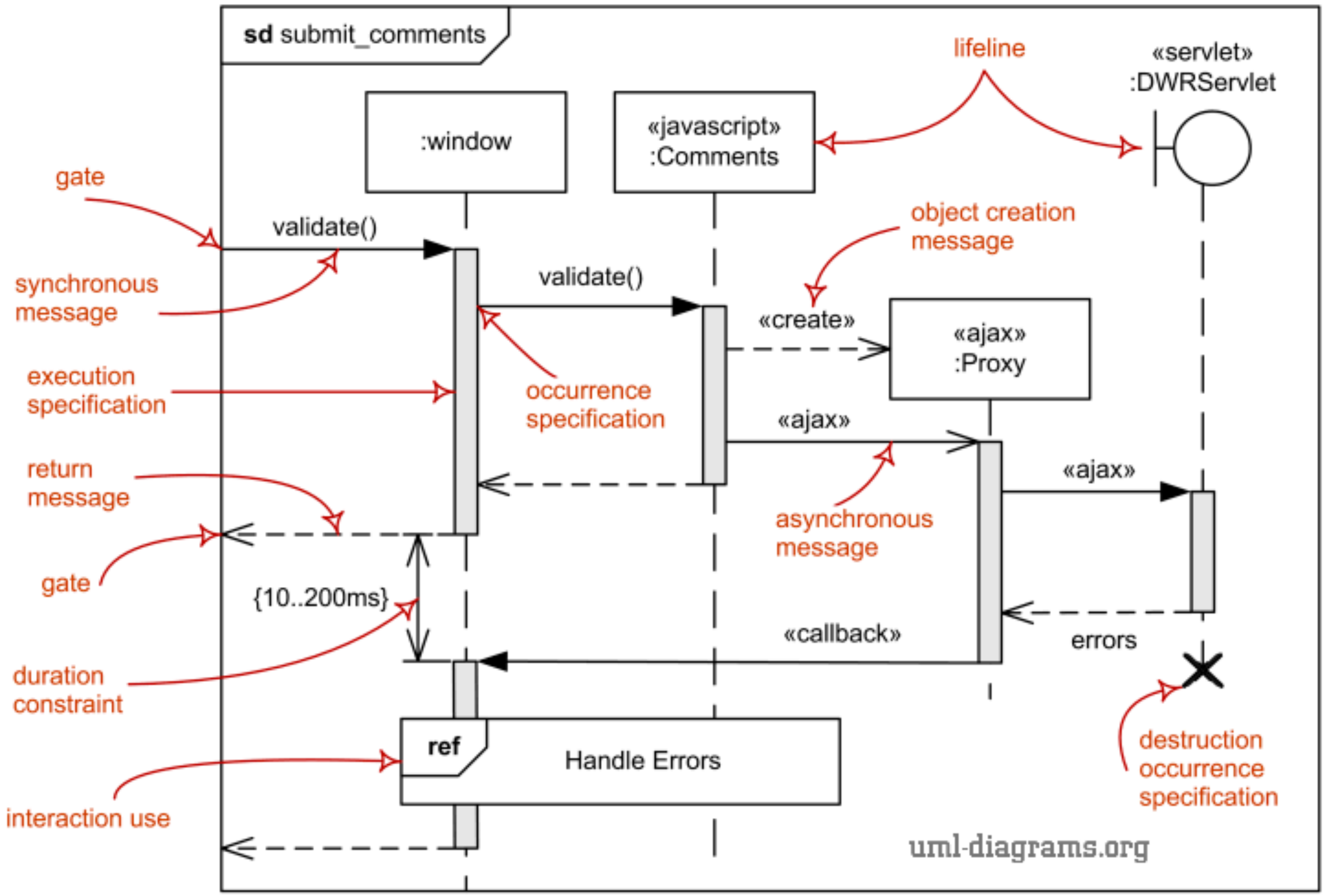
- How objects collaborate?

  - Communicate via sending messages, by invoking operations of objects

- Sending the message getBalence() to an Acount object, might use the following syntax.

currentBalance= a.getBalance();

Notation [Source: uml-diagrams.com]



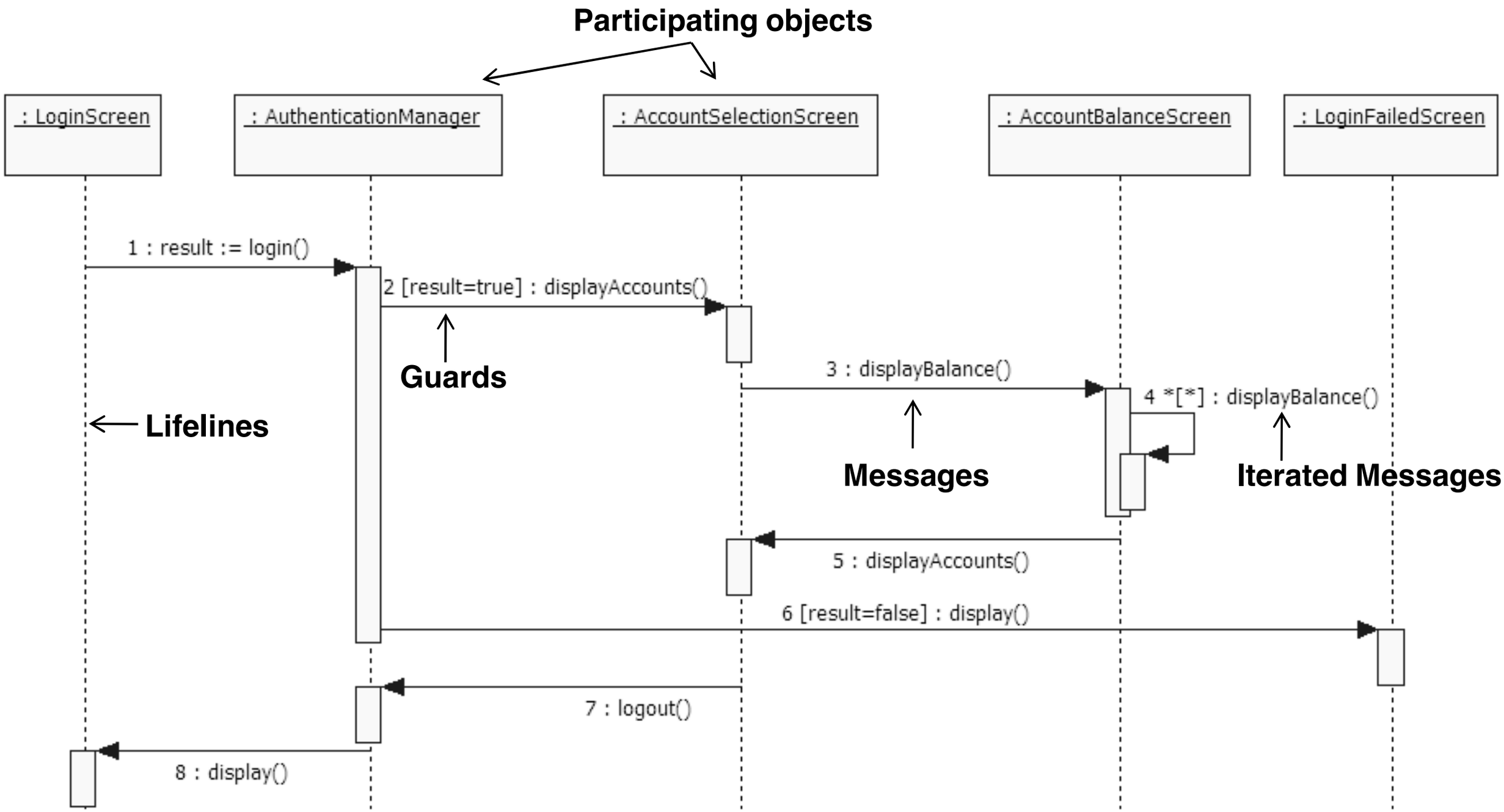
Scenario

- A user provides his details in the login screen. If the details are valid, he is taken to a screen which displays all his accounts. He can then select an account and view its balance in a new screen. If the details are not valid, an error screen is displayed.

- The user can refresh the account balance screen or go back to the list of accounts screen.

- From the list of accounts screen, the user can log out of the system.

Sequence Diagram



Message Notation

- Numbers (e.g. 1. 2. 3. etc.) represent message order

- result := login()
    - Executes login() and stores the result in the result variable

- [result = true] : displayAccounts()
    - If result = true (guard) then execute displayAccounts()
    - result is the variable we created in the previous message

Message Notation

- * [*] : displayBalance()
  - Repeat (first *) an unlimited number of times (second *) displayBalance()
  - * [2] : displayBalance() would mean repeat exactly twice
- Don’t worry too much if you cannot express complex algorithms with this notation:
  - UML is NOT a programming language
  - Capture additional details using notes

Sequence Diagram Advice

- First, determine what the system should do, at a high level
  - System services and scenarios of use
- Then identify the actors and objects that collaborate to achieve the required behaviour
  - Either via class identification, or as a step towards identifying classes

Sequence Diagram Advice

- Build system sequence diagrams by identifying external events and system responses
- Build diagrams of internal behaviour by identifying how objects need to communicate
- Focus on capturing the intent rather than the physical effect
  - Do not try to use the notation to draw a flowchart
- Remember: UML is NOT a programming language

State Diagrams

State Diagrams

- Class Diagrams describe static structure
- Sequence Diagrams describe object interaction
- State Diagrams describe internal object behaviour
  - The states that an object can get to
  - How the object responds to events

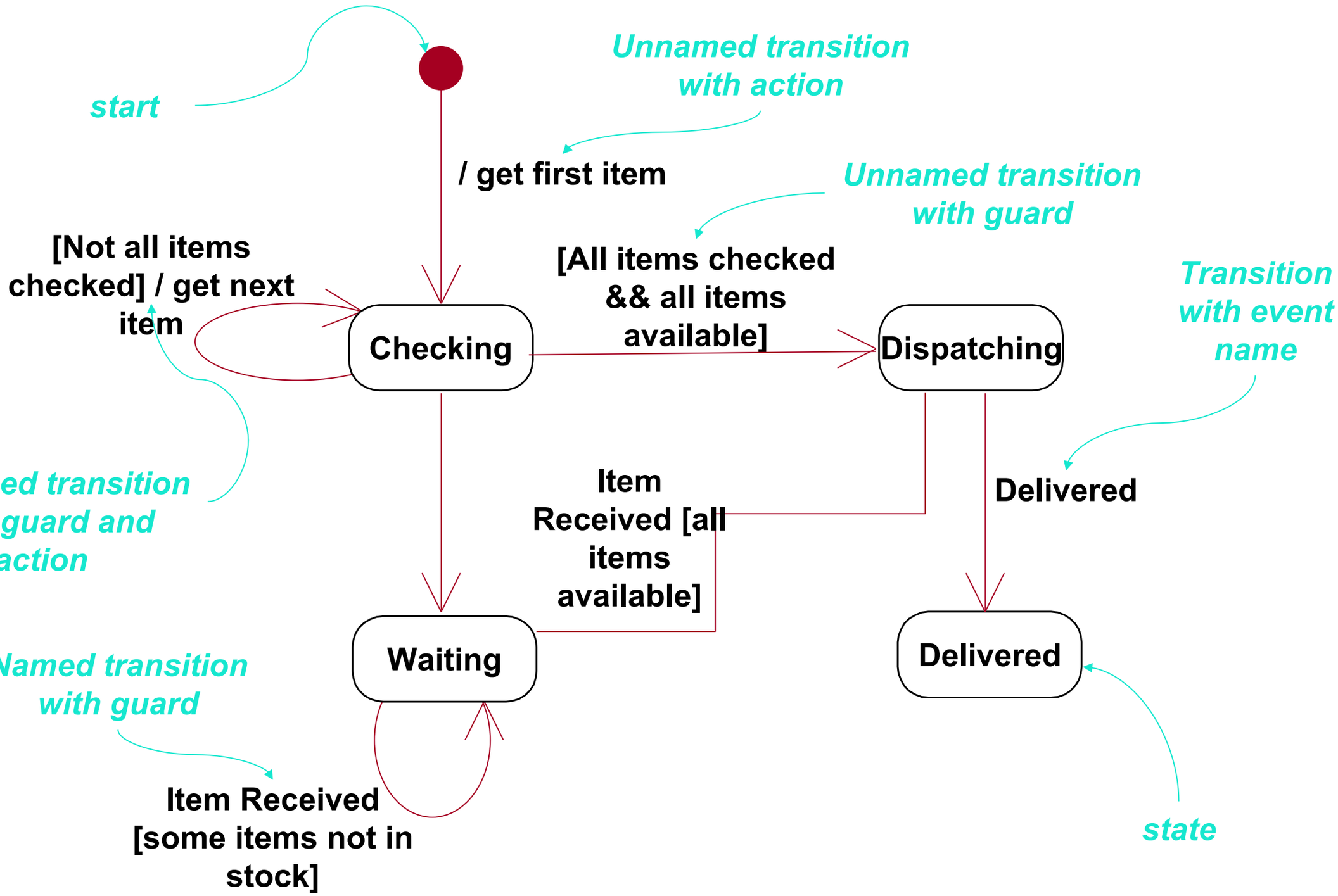
State Diagrams: Basic concepts

- In its simplest form a state diagram contains
  - States
    - The states of the object (including initial and final states)
  - A state models a situation during which some invariant condition holds.
  - Three types of states: simple, composite, submachine

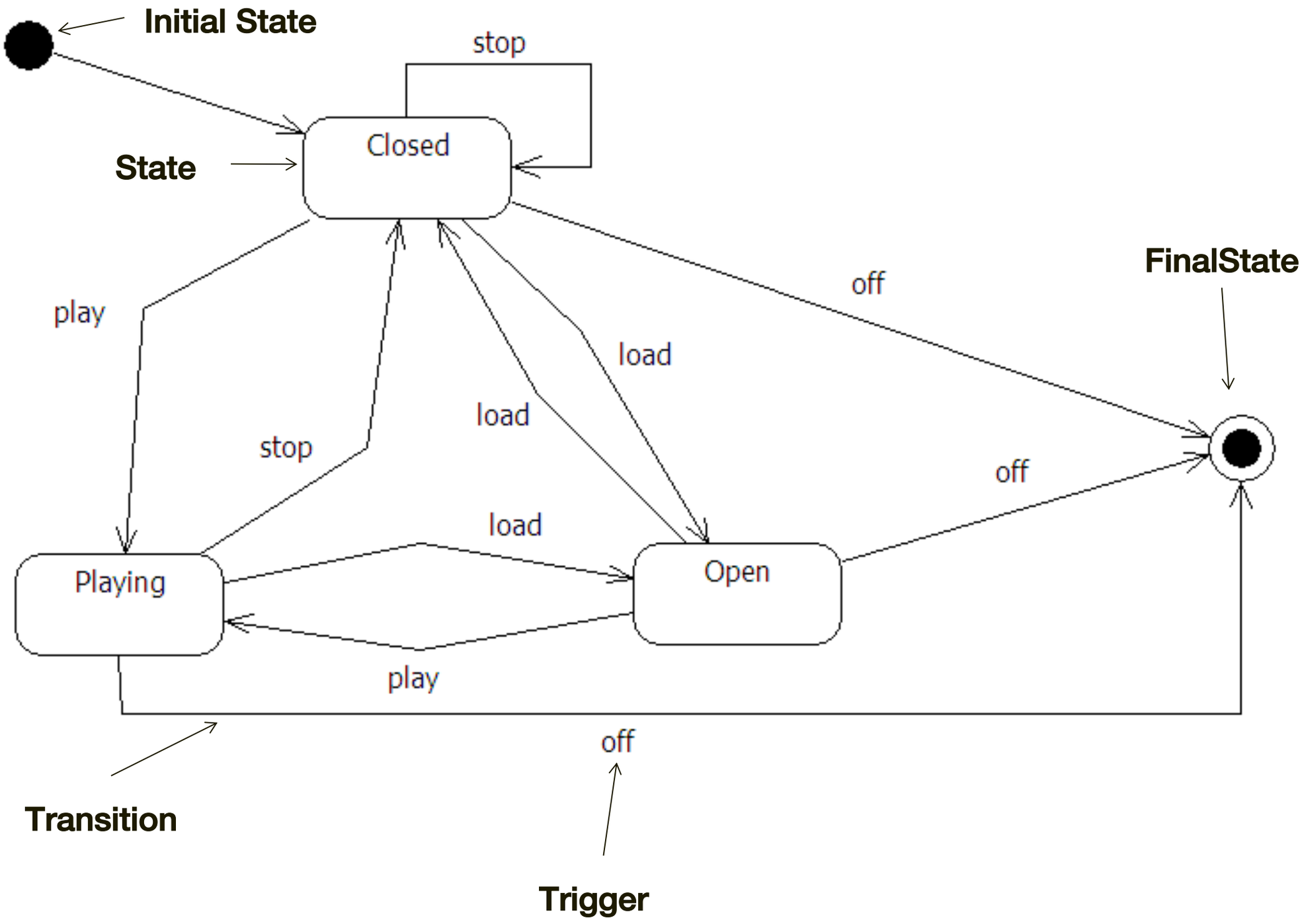
State Diagrams: Basic concepts

- Transitions
  - Allowed transitions between states
- Triggers
  - Events that trigger transitions between states

Notation


start
Unnamed transition with action
/ get first item
Unnamed transition with guard
[All items checked && all items available]
[Not all items checked] / get next item
Transition with event name
Checking
Dispatching
Unnamed transition with guard and action
Item Received [all items available]
Delivered
Waiting
Delivered
Named transition with guard
Item Received [some items not in stock]
state

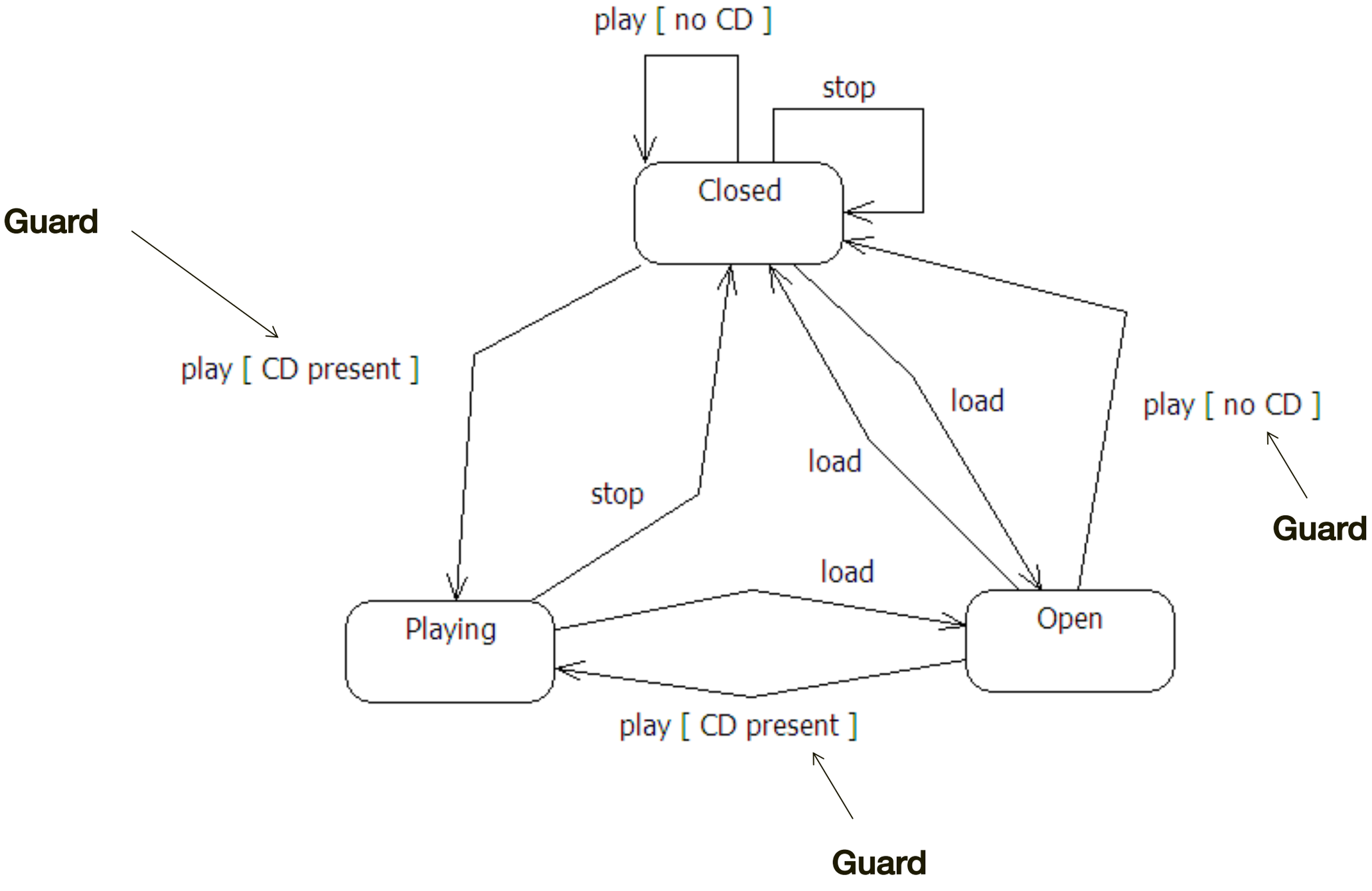
Basic state diagram for a disk player



State Diagrams: Guards

- Except for triggers, transitions can also specify guards
  - Transition happens if trigger is generated AND guards are satisfied
  - Guards appear next to triggers enclosed in [ ]

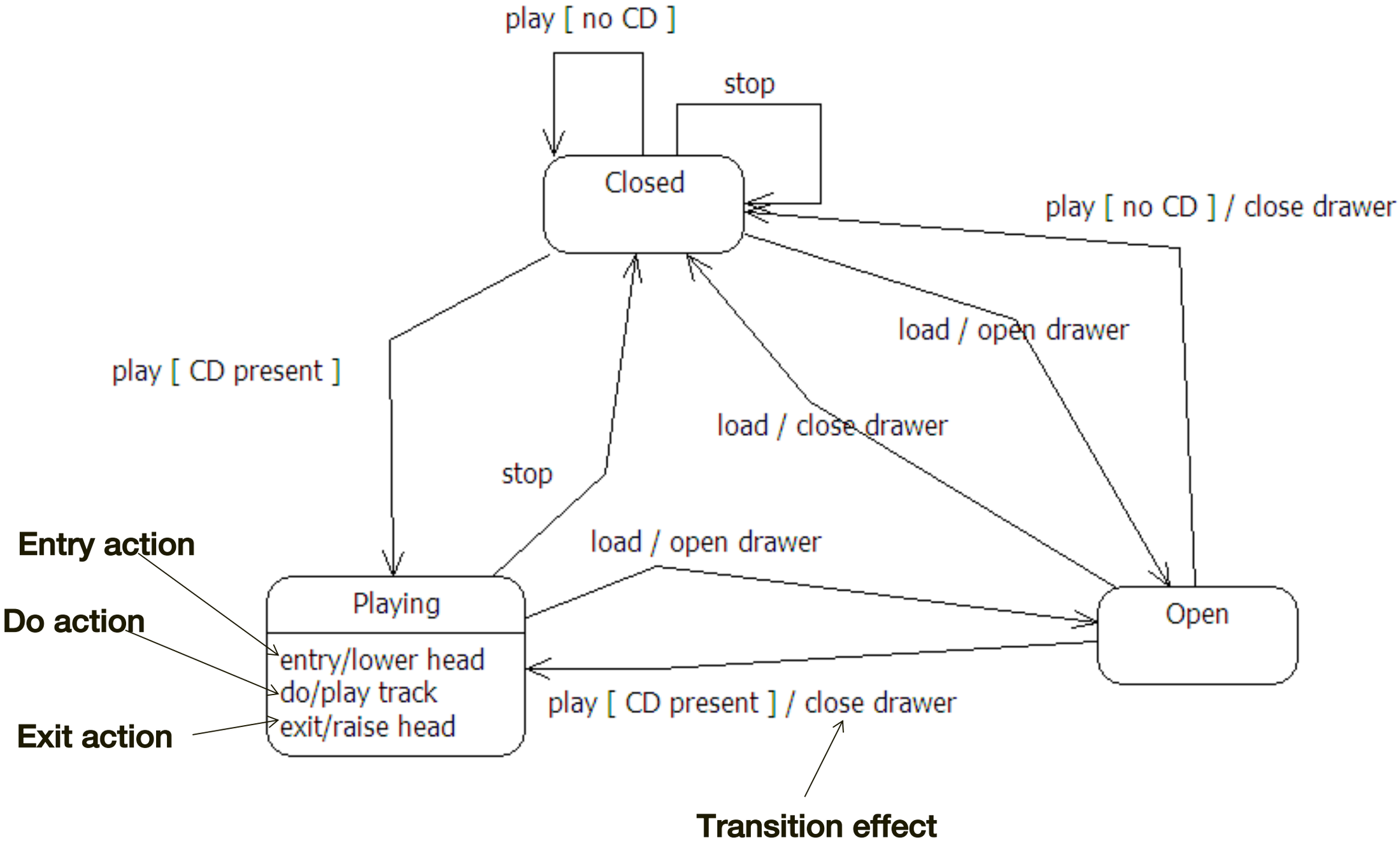
State Diagrams: Guards



State Diagrams: Effects and Activities

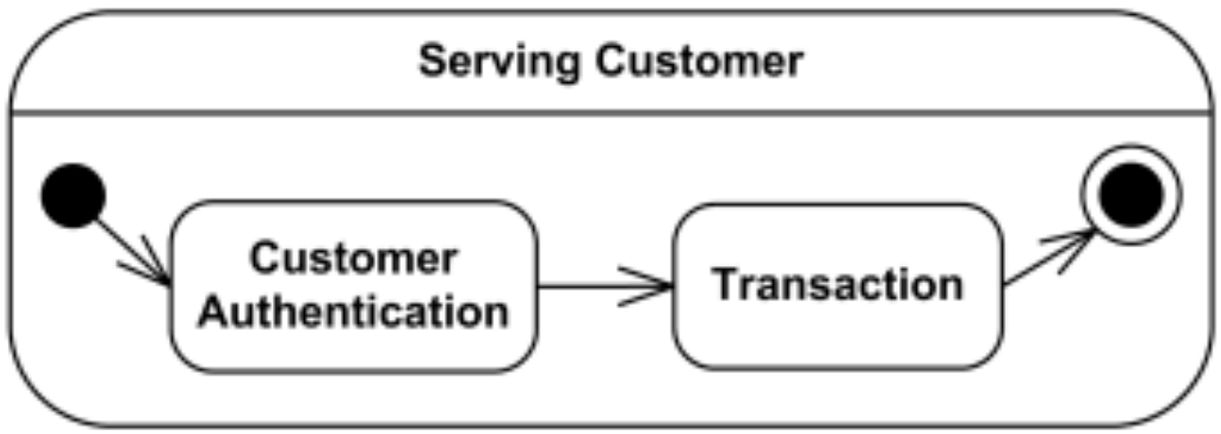
- Transitions can have effects
  - Specify what happens during the transition
- States can specify three types of actions
  - Entry (when object enters state)
  - Do (while object in state)
  - Exit (just before object exists state)

State Diagrams: Effects and Activities



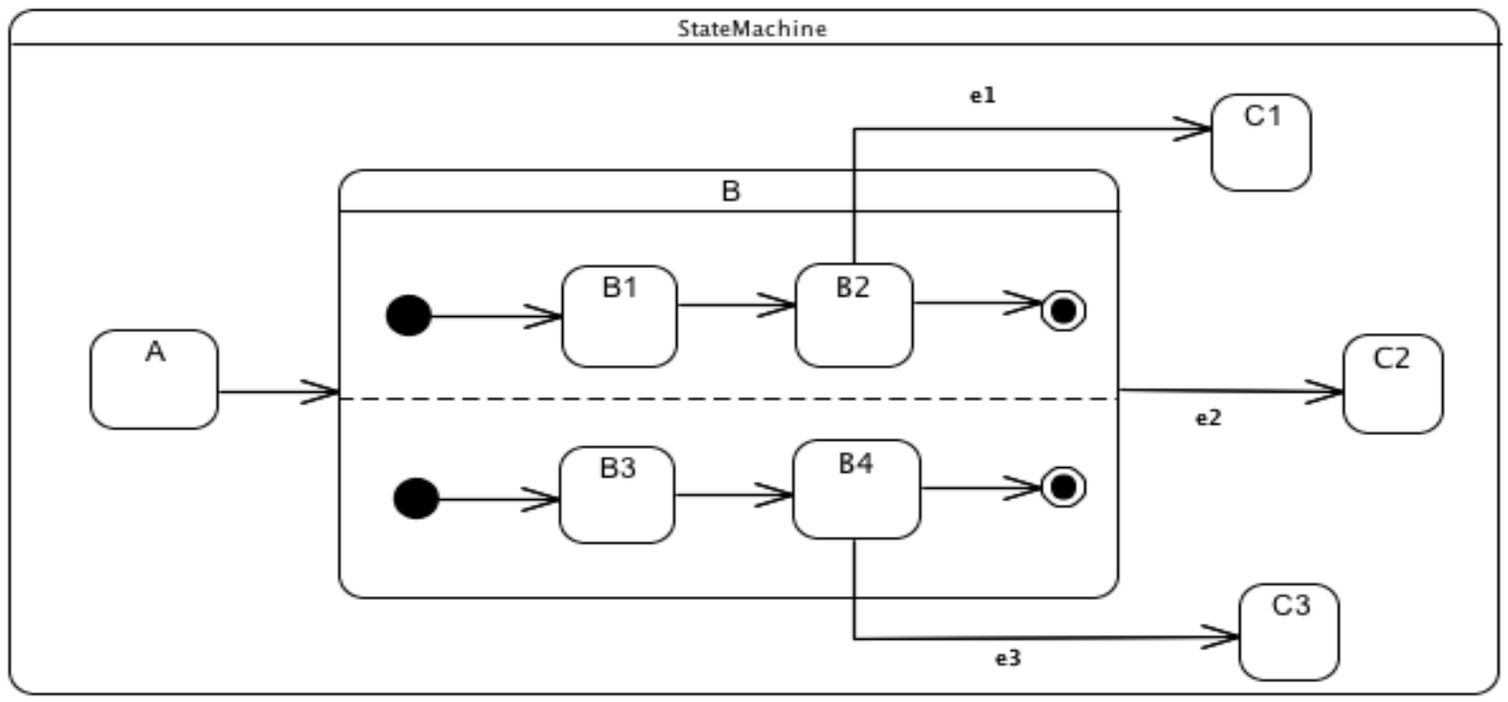
Composite State

- A composite state is a state with one or more regions.
- A region is simply a container for substates.
- Decomposition compartment.



Composite states / regions

- Each region within a composite state executes in parallel.
- A transition to the final state of a region indicates completing the activity for that region. •
- Once all the regions have completed, the composite state triggers a completion event and a completion transition (if one exists) triggers.



When to Use State Diagrams

- State diagrams give the per-object view of system behaviour
  - The cross-view, through all the scenarios of use concerning an object
  - A control view for the objects of a class
- State diagrams are not good at describing behaviour that involves a number of collaborating objects
  - Use a sequence diagram for this
- Often, many classes have only simple object behaviour
  - Not worth drawing a state diagram for every class

Reading

- UML Distilled: Applying the Standard Object Modeling Language – **Chapter 4, 10,12**
- Systems analysis and design with UML: an object-oriented approach – **Chapter 7**

- **https://www.uml-diagrams.org/**