AMSS Lecture 7: UML State Diagrams

Traian-Florin Șerbănuță

2025

Agenda

Session 1: Foundations of State Diagrams **Session 2:** Advanced Modeling and Code Mapping

Session 1: Foundations of State Diagrams

What Are State Diagrams?

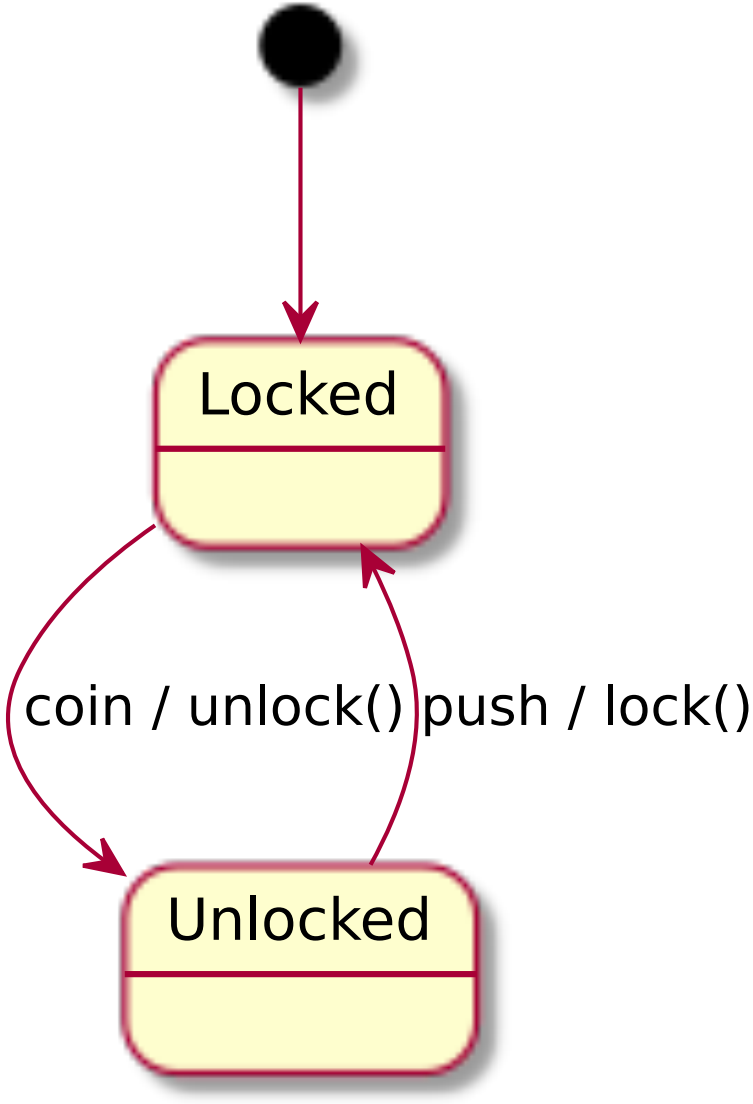
Definition: A UML State Machine Diagram models the *lifecycle* of an object — the states it goes through and how it transitions between them.

Use cases: - Modeling reactive systems. - Describing event-driven behavior. - Understanding object lifecycle and valid transitions.

Key Elements: - **States** (simple, composite) - **Transitions** (with optional triggers, guards, and actions) - **Initial/Final states**

Example: Turnstile State Diagram

A turnstile at a metro station:



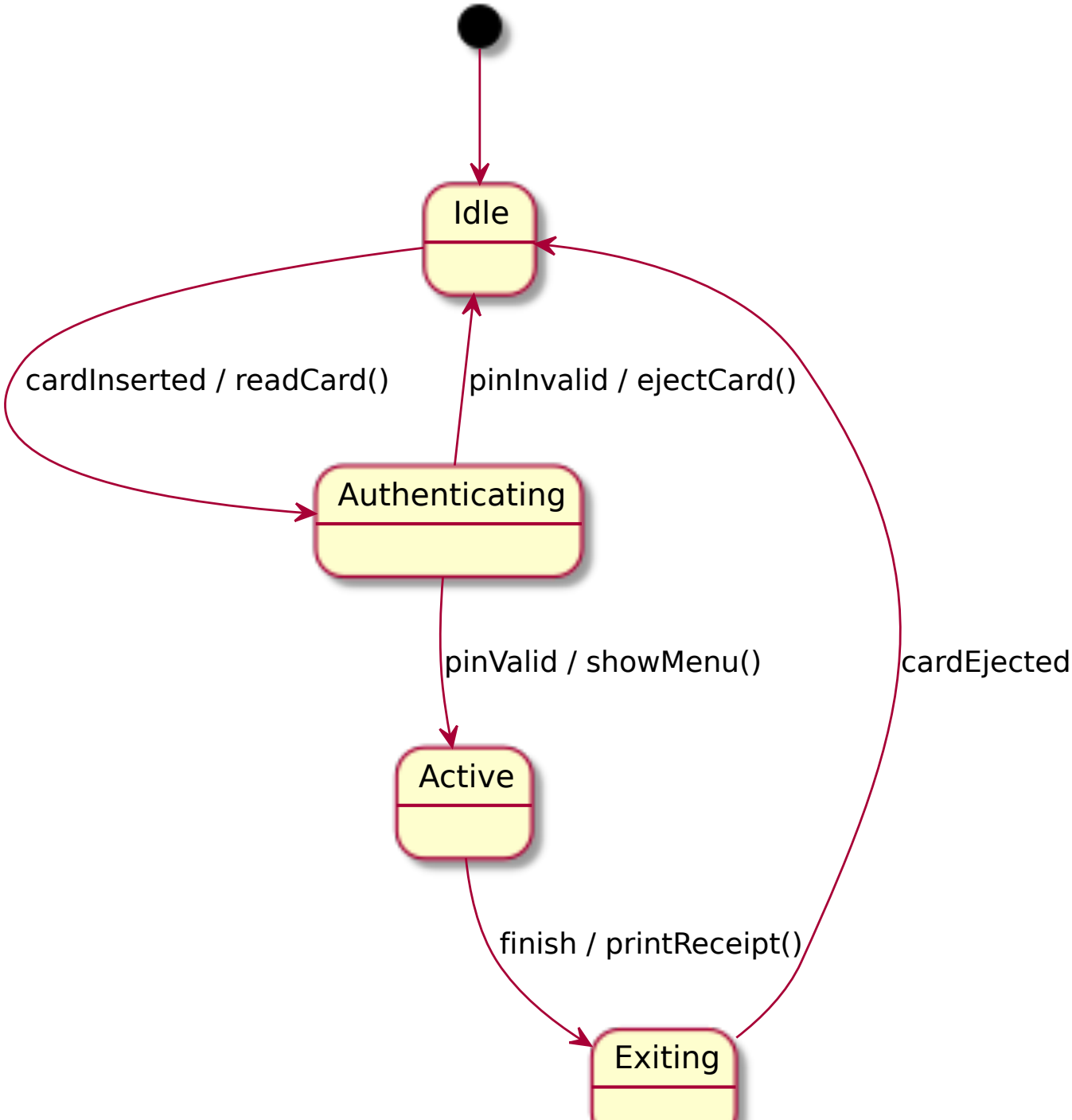
Components of Transitions

- **Trigger:** event that initiates the transition (e.g., `coin`, `push`)
- **Guard:** condition that must be true for transition to occur (e.g., `[balance > 0]`)
- **Action:** operation executed when the transition occurs (e.g., `unlock()`, `displayMessage()`)

Syntax:

```
trigger [guard] / action
```

Example: ATM Session Lifecycle



Interactive Exercise: Identify Missing Transitions

Scenario: ATM State Diagram

- What happens if the card reader fails?
- How do we represent timeout conditions?

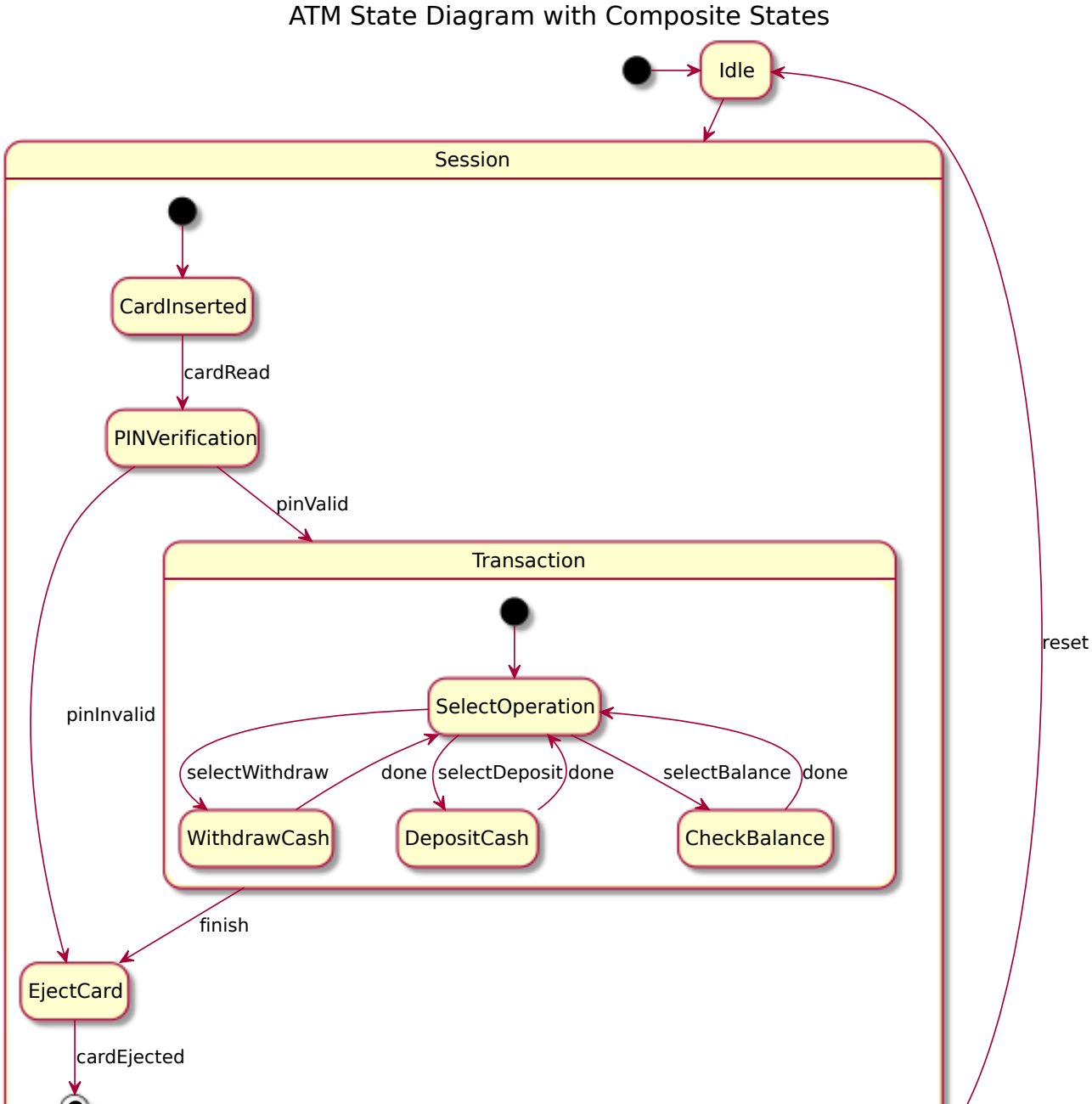
Task: Add error or timeout transitions to the diagram.

Recap

- **States** capture modes of behavior.
- **Transitions** define possible responses to events.
- **Actions** are side effects of transitions.
- State diagrams complement sequence/activity diagrams by focusing on *object lifecycles*.

Session 2: Advanced Modeling and Code Mapping

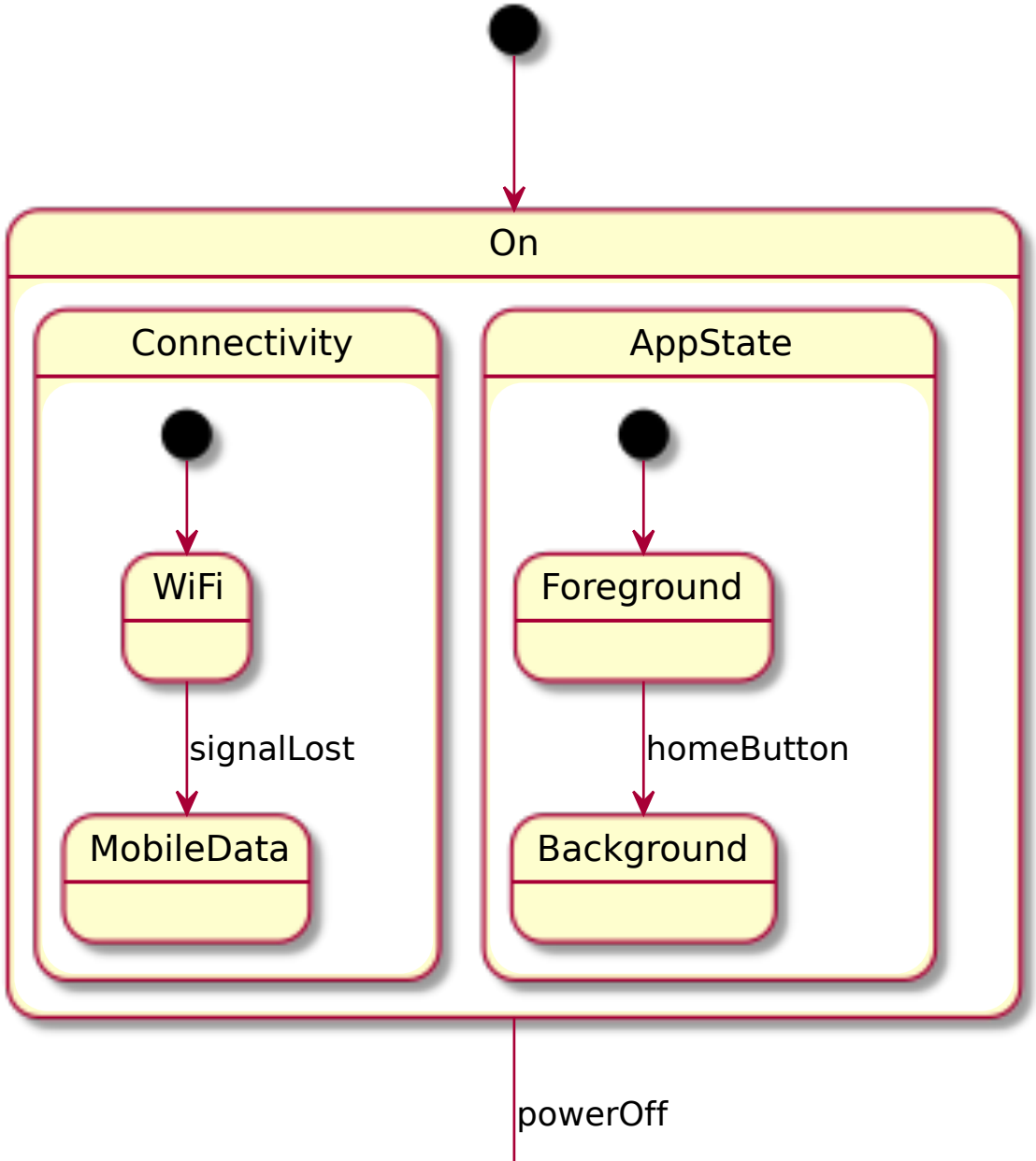
Composite States and Substates



Concurrent (Orthogonal) States

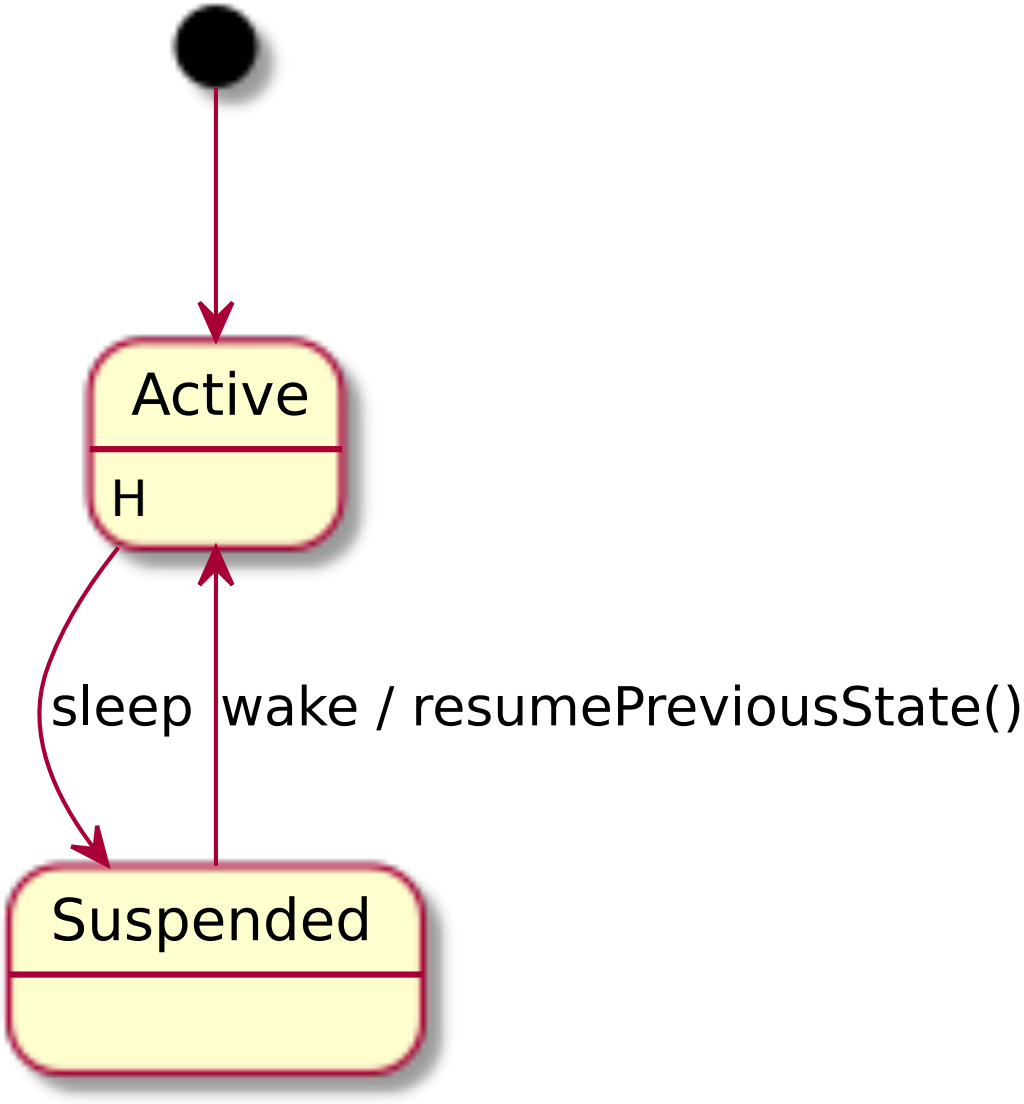
Some systems have *independent* aspects of behavior.

Example: Smartphone with concurrent regions:



History States

Used to remember the **last active substate** when re-entering a composite state.



From UML to Code (Java Example)

Simple **State Pattern** implementation mapping UML concepts to code:

```
interface State {
    void handle(Context ctx, String event);
}

class Locked implements State {
    public void handle(Context ctx, String event) {
        if (event.equals("coin")) {
            System.out.println("Unlocked");
            ctx.setState(new Unlocked());
        }
    }
}

class Unlocked implements State {
    public void handle(Context ctx, String event) {
        if (event.equals("push")) {
            System.out.println("Locked");
```

Interactive Task

Task: Implement a simple state machine for a *Traffic Light* using the State pattern. States: `Red`, `Green`, `Yellow`. Transitions triggered by `next()`.

Wrap-Up

Concept	Description	Example
Simple state	Mode of behavior	Locked/Unlocked
Composite state	Grouped states	Playing (Buffering, St
Concurrent state	Independent regions	Connectivity, AppStat
History	Remembers previous substate	Resume after pause

Takeaway: UML state diagrams help capture dynamic, event-driven aspects of systems and bridge toward implementable designs.

Next Lecture: Structural Design Patterns – Visitor, Mediator, Bridge.