THE UNIVERSITY OF MELBOURNE

SWEN30006
Software Modelling and Design

UML State Machine Diagrams and Modelling

Larman Chapter 29

No, no, you're not thinking, you're just being logical.

—Niels Bohr

Objectives

On completion of this topic you should be able to:

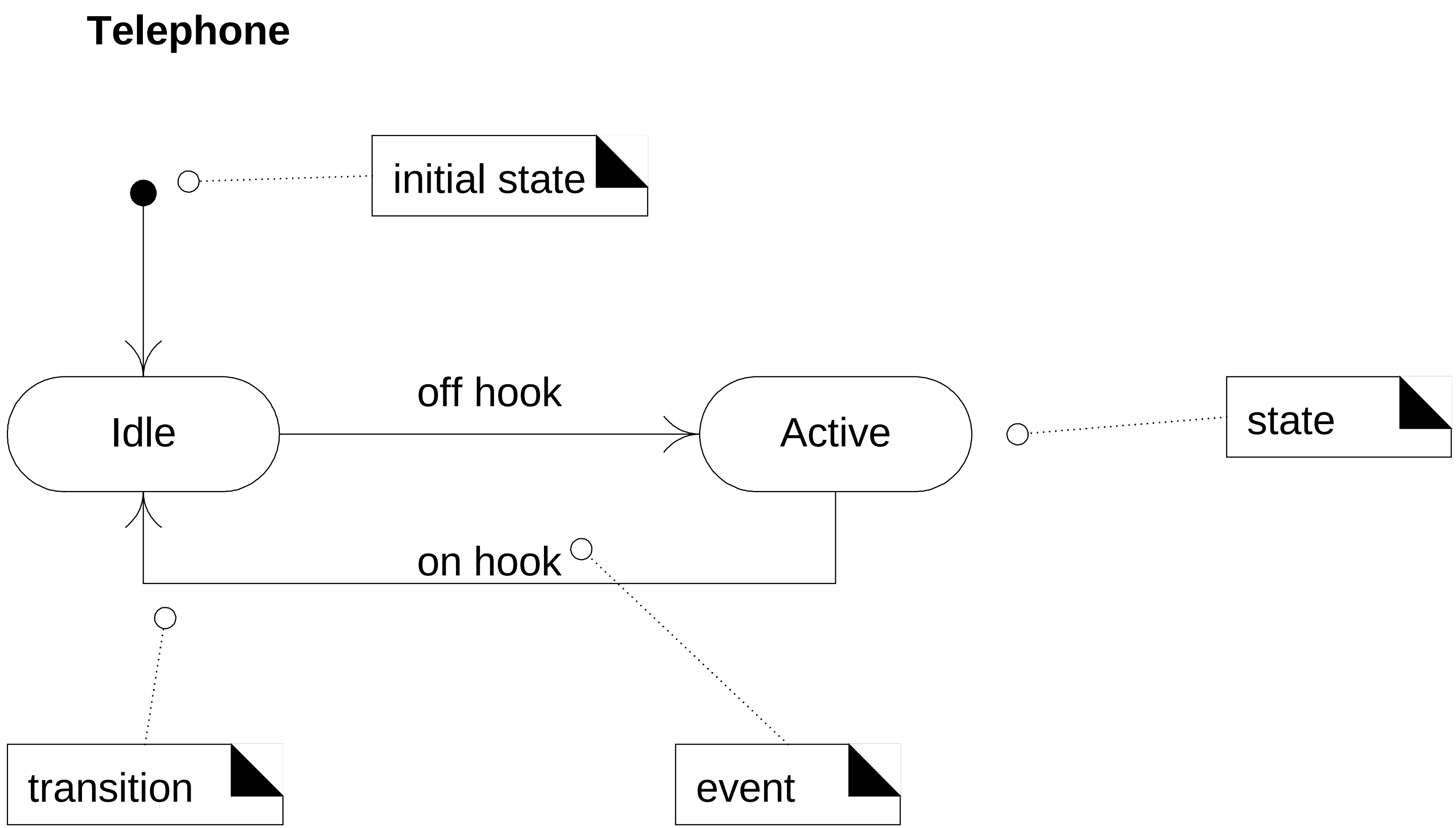
- Understand UML state machine diagram notation.
- Be aware of where state machine modelling is applicable.

State Machines

A *state machine* describes the behaviour of an object in terms of

- *events* that affect the object and
- the *state* of the object between events.

State Machine Diagram for a Telephone



State Machine Definitions (1)

Event:

- a significant or noteworthy occurrence.

State:

- condition of the object at a moment in time.

Transition:

- directed relationship between two states such that an event can cause the object to change from the prior state to the subsequent state.

State Machine Definitions (2)

state-independent w.r.t. an event:

- Always responds to event the same way

state-independent object:

- For all events of interest, always reacts to the event the same way

state-dependent object:

- Reacts differently to events depending on their state

How to Apply State Machine Diagrams? (1)

Guideline 1: consider state machines for state-dependent objects with complex behaviour.

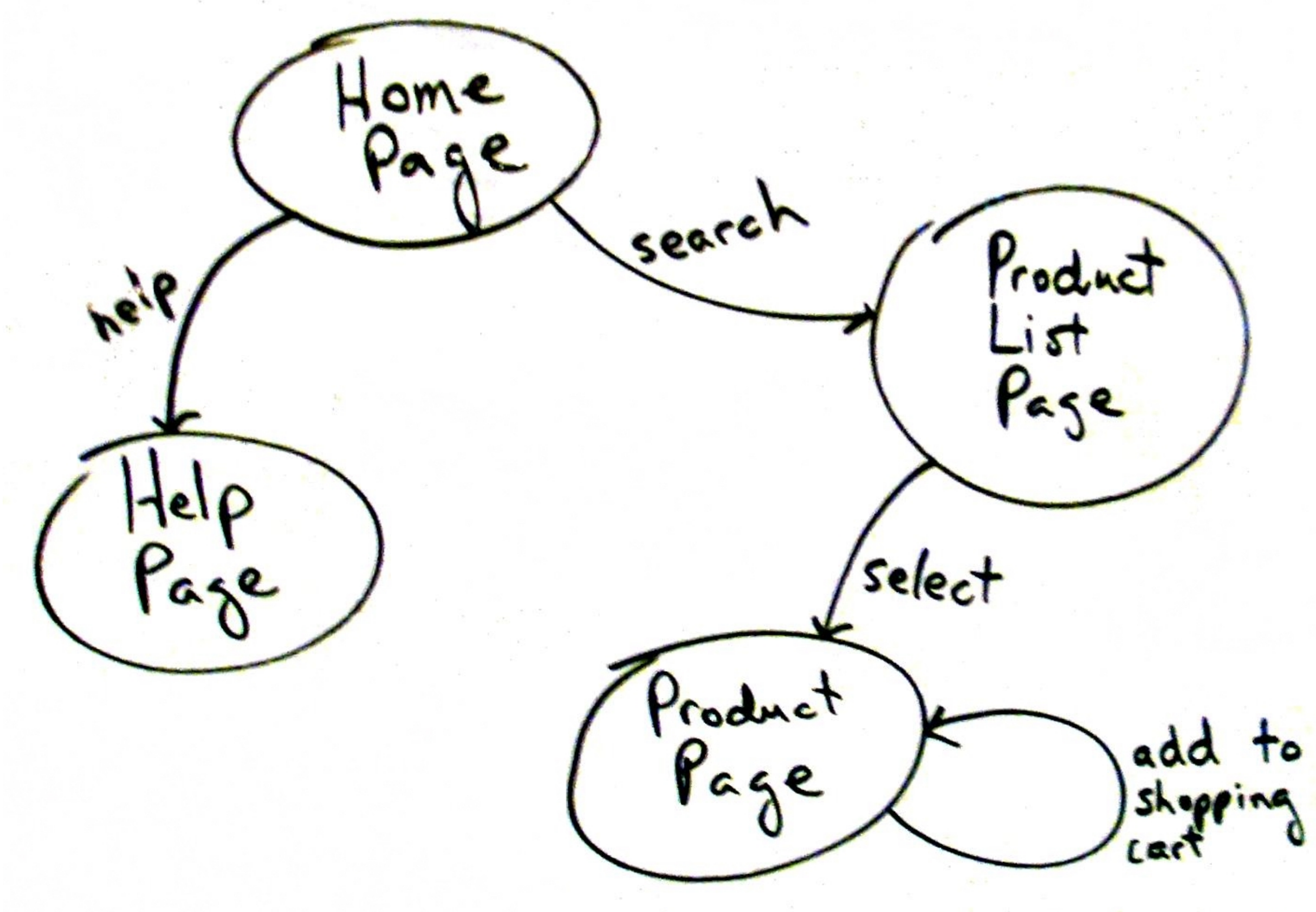
- model behaviour of complex reactive objects

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

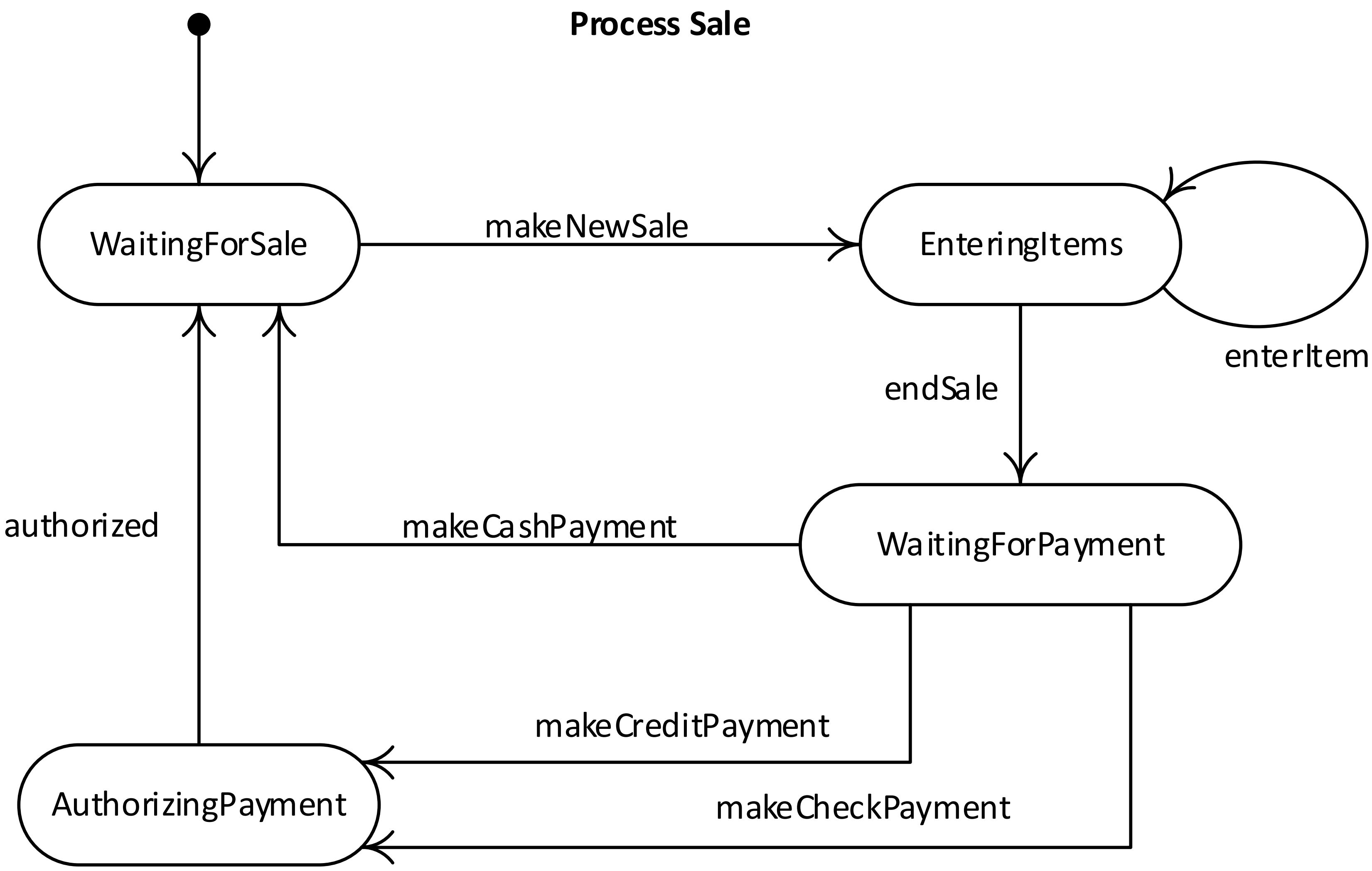
How to Apply State Machine Diagrams? (2)

1. **Complex Reactive Objects**
   - Physical device controllers
   - Transactions and related Business Objects
   - Role Mutators (objects that change role)
2. **Protocols and Legal Sequences**
   - Communications Protocols
   - UI Page/Window Flow, Navigation, or Session
   - Use Case Operation Sequencing

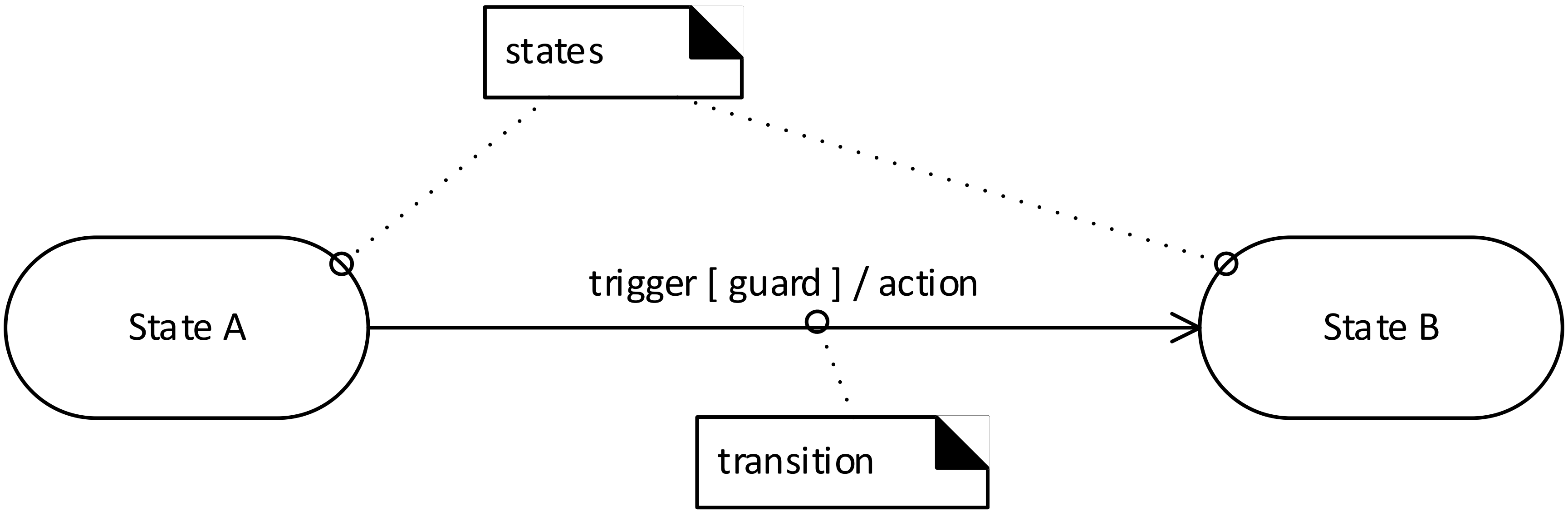
Web Page Navigation



Process Sale Operation Sequencing



Basic State Machine Notation



When <u>object</u> is in *State A*:

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

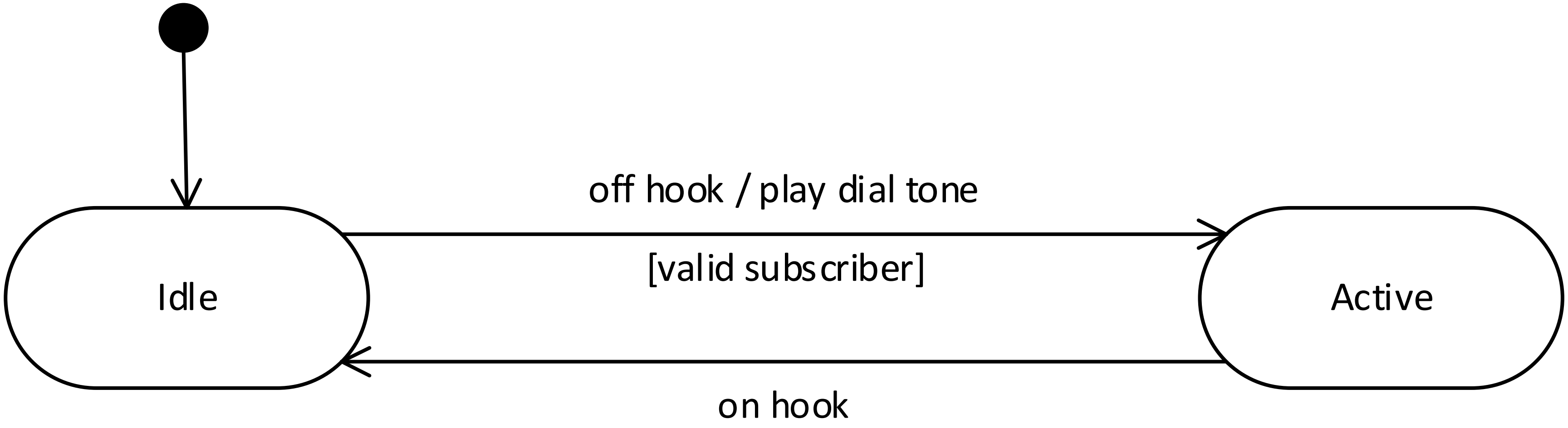
then

perform the behaviour *action* and

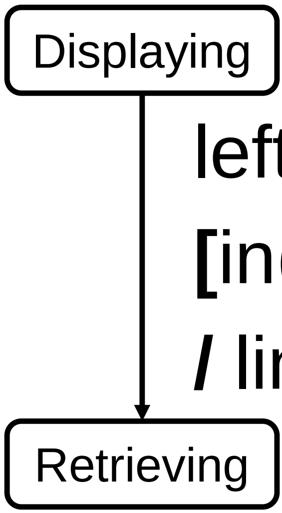
transition <u>object</u> to *State B*.

Transition Action and Guard Example

Telephone



Browser Transition Example



left-mouse-down(coordinates) trigger

[in(coordinates, active_window)**]** guard

/ link=select-link(coordinates); link.follow() action

When Browser is *Displaying*:

if *left-mouse-down(coordinates)* occurs and

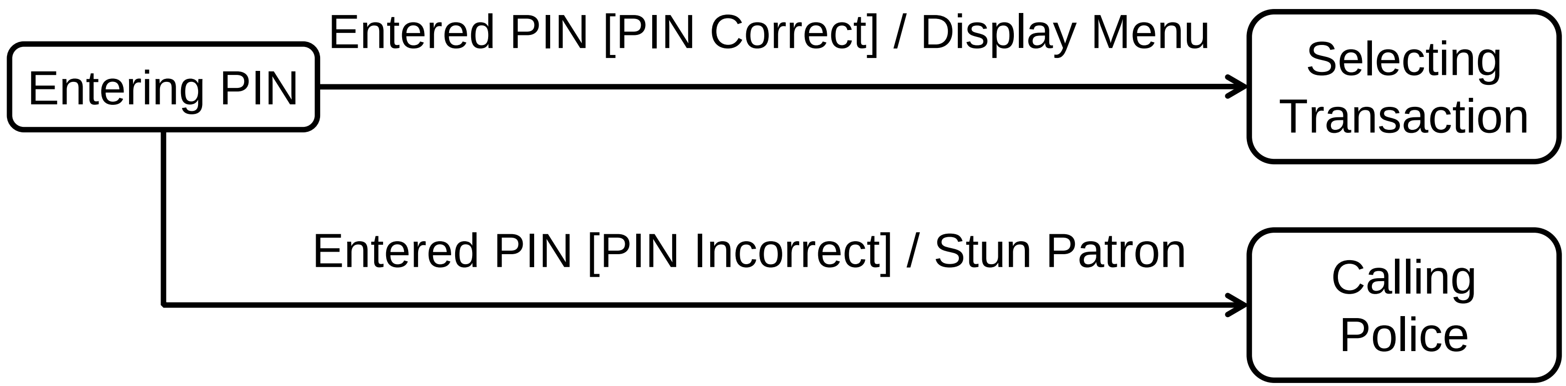
in(*coordinates, active_window)*

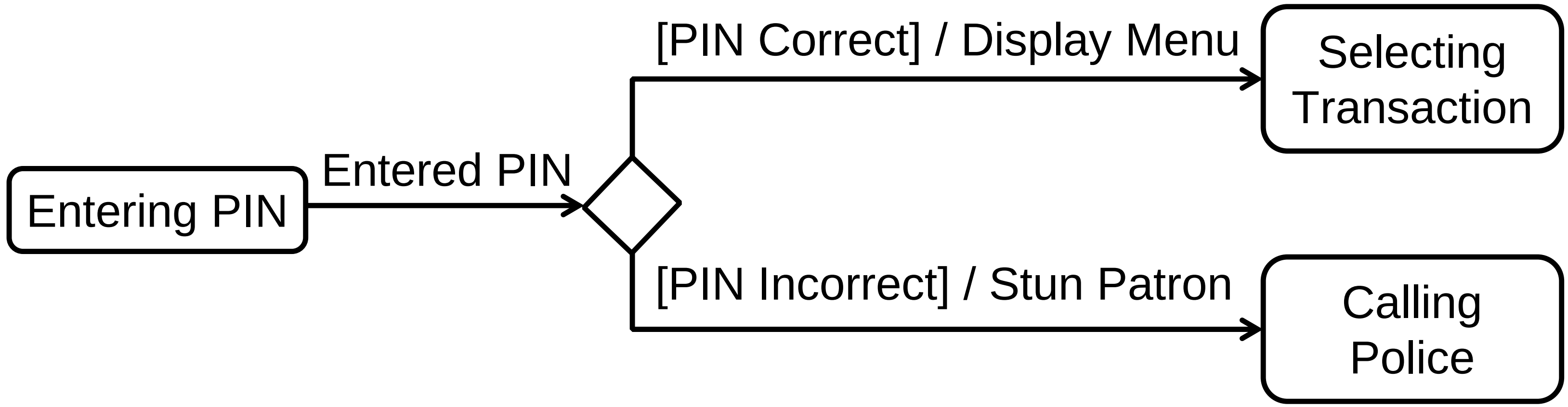
then

link=select-link(coordinates); link.follow();

transition Browser to *Retrieving;*

Choice Pseudostate

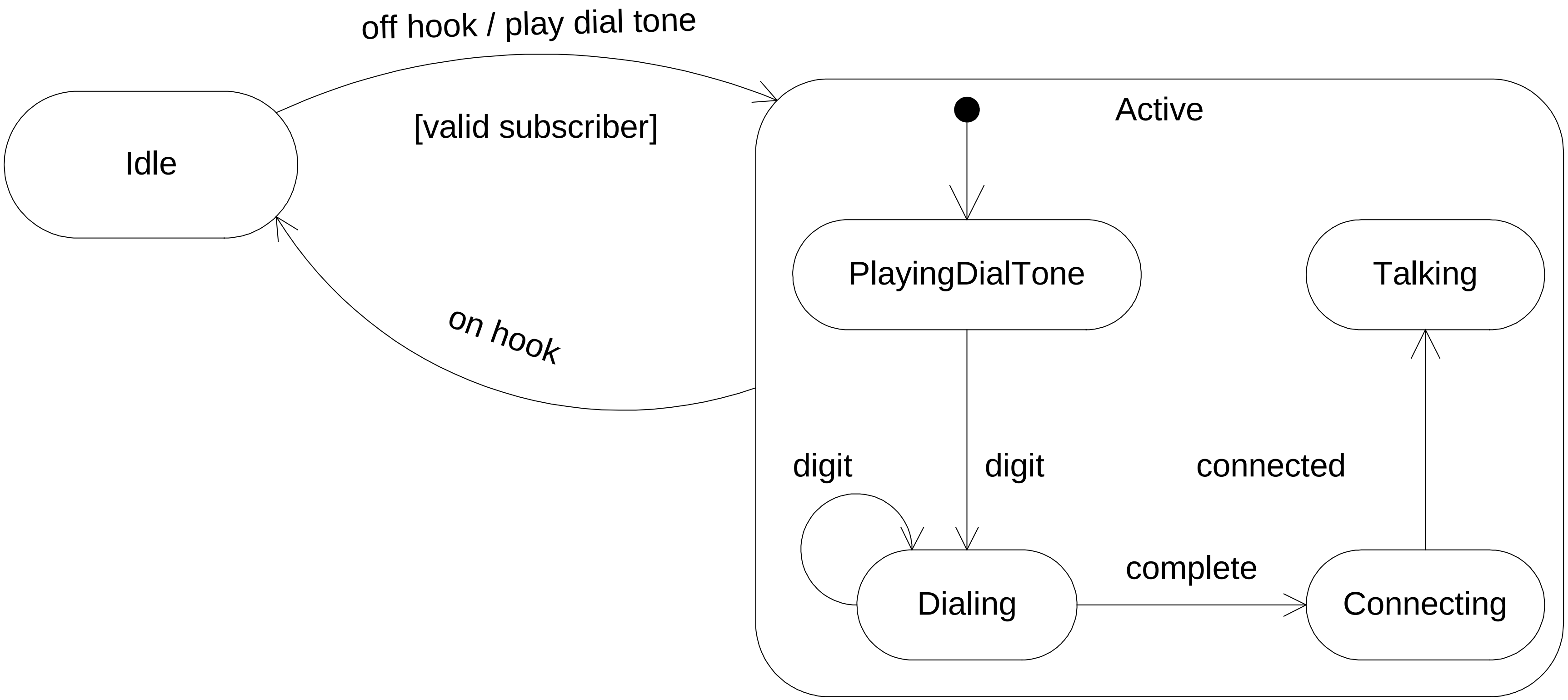




Choice Pseudostate (continued)

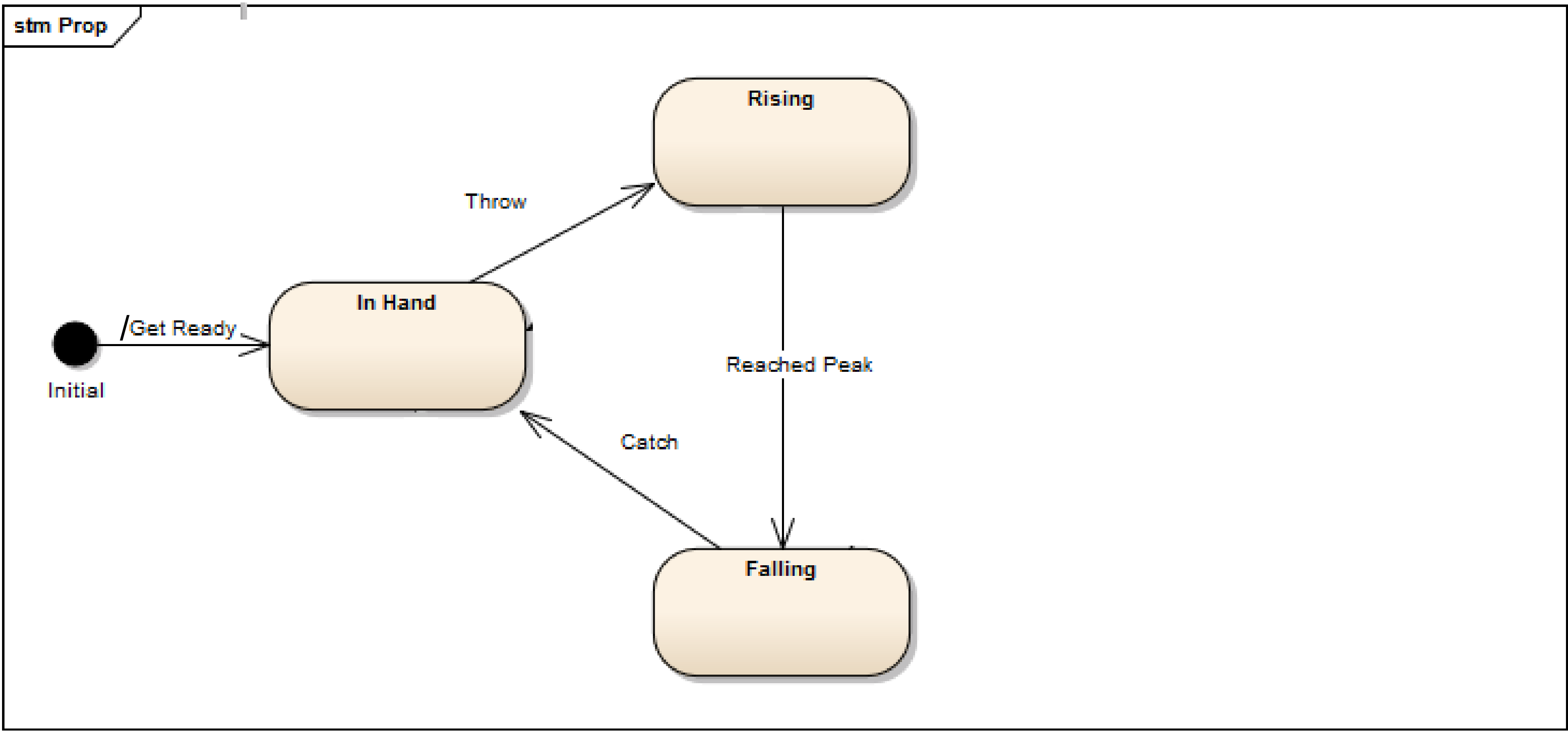
- Can have two or more outgoing transitions
  - E.g. [bal < 0] [0 <= bal < min] [min <= bal]
- Can also use predefined [else] guard
  - [else] outgoing transition chosen if no other guards are true
  - E.g. [day = Sat] [day = Sun] [else]

Nested States



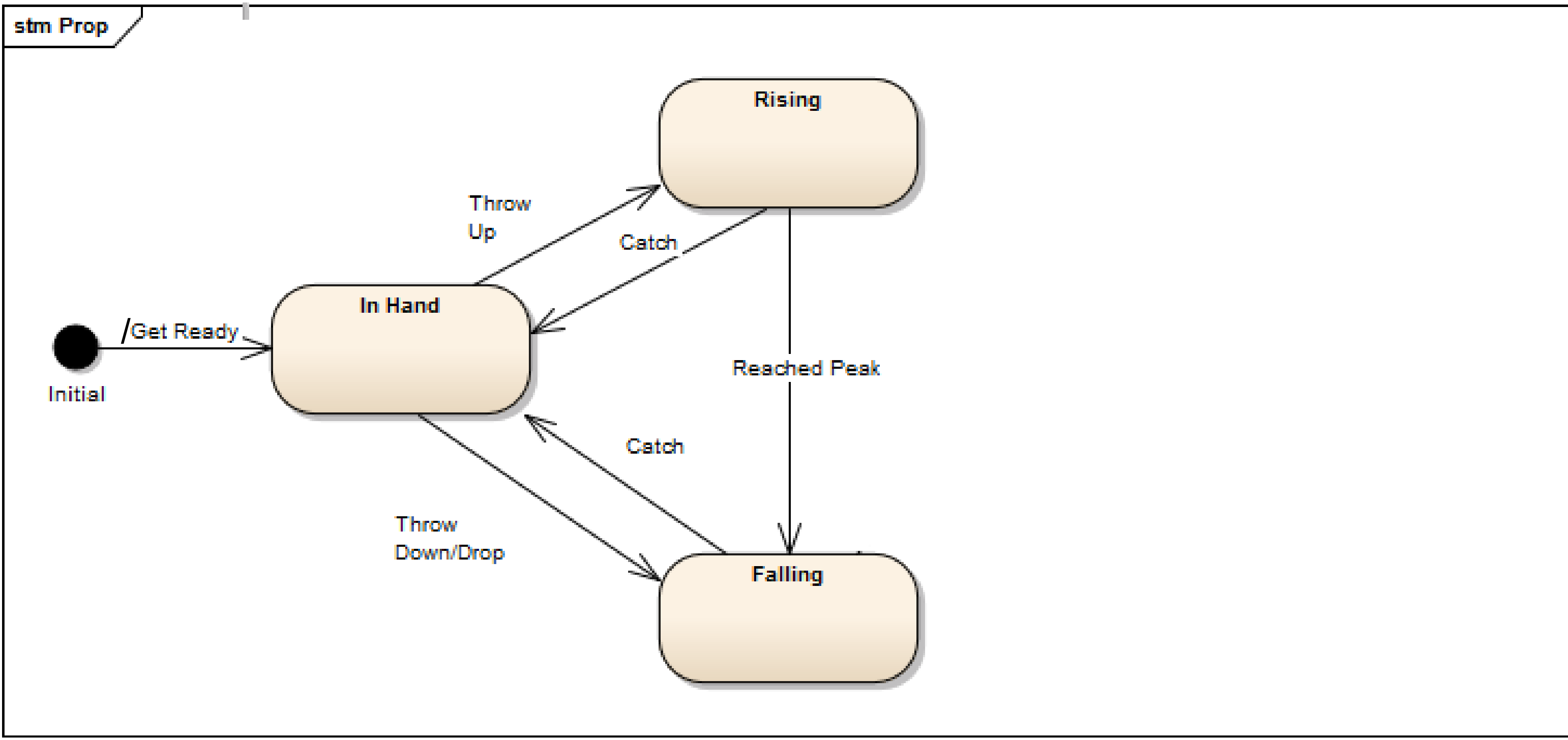
- Transition into *Active* (via *off hook*) transitions into substate *PlayingDialTone*
- All substates of *Active* inherit the *onhook* transition.

Juggling Prop (1)



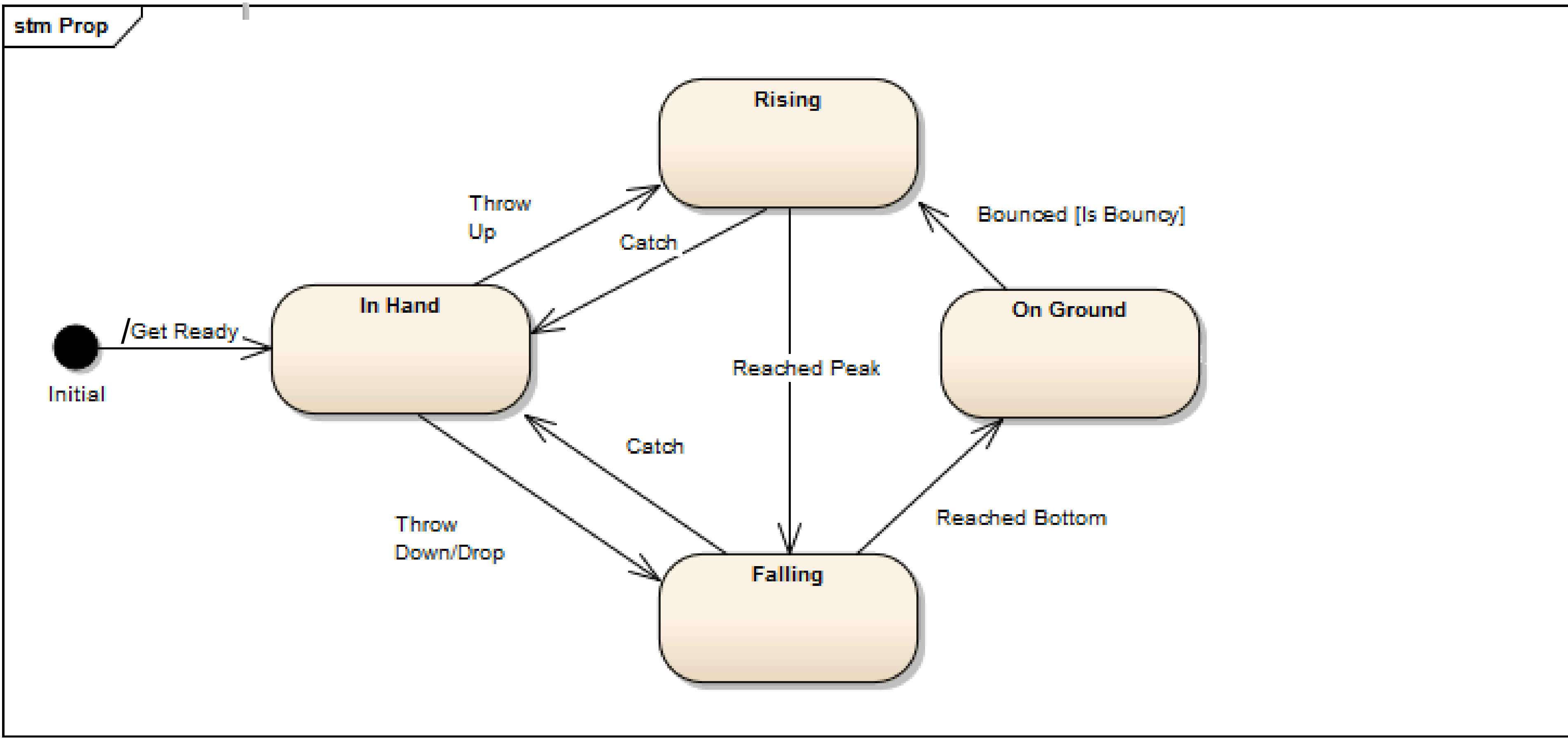
Juggling Pattern: Cascade (symmetric)

Juggling Prop (2)



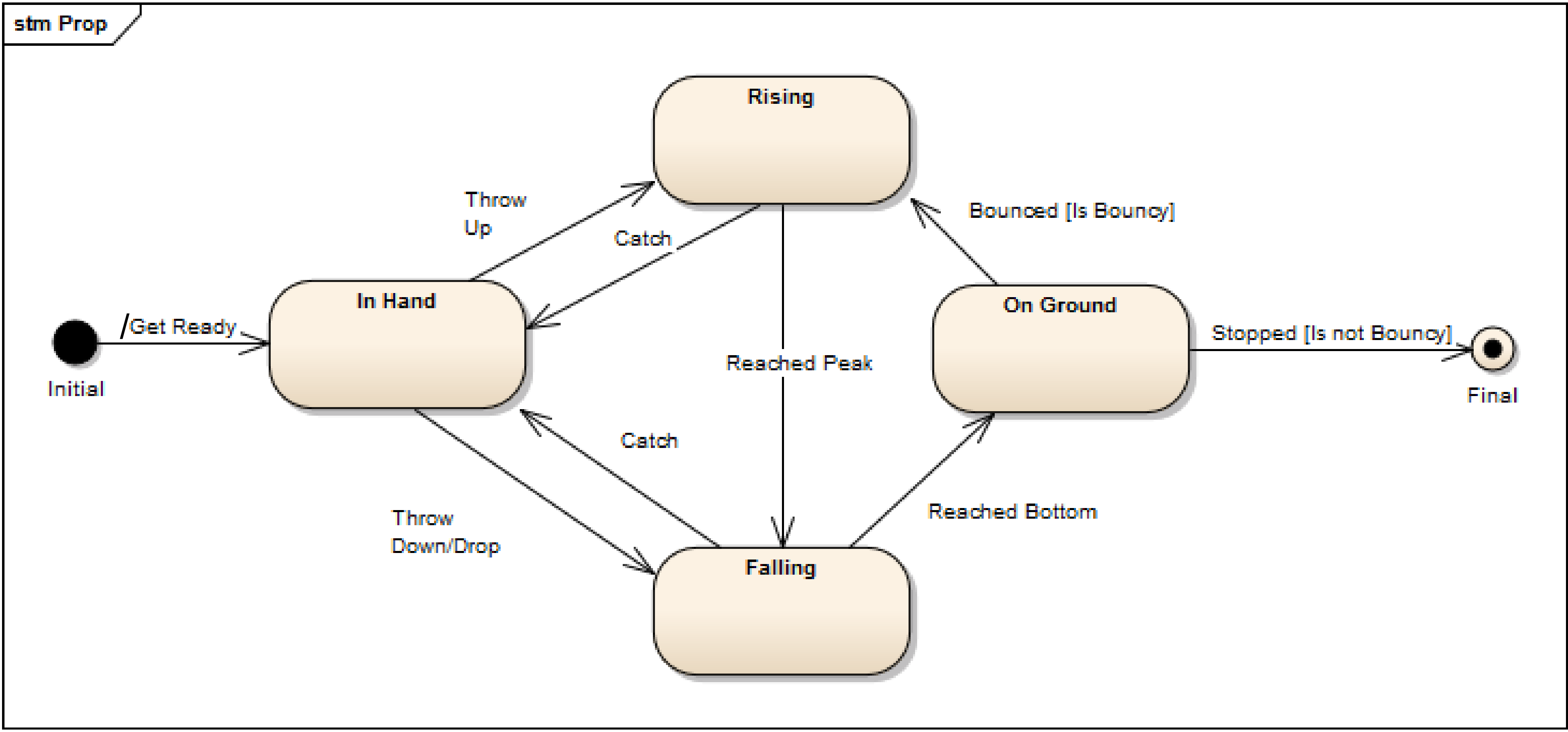
Juggling Pattern: Cascade; Shower; …

Juggling Prop (3)



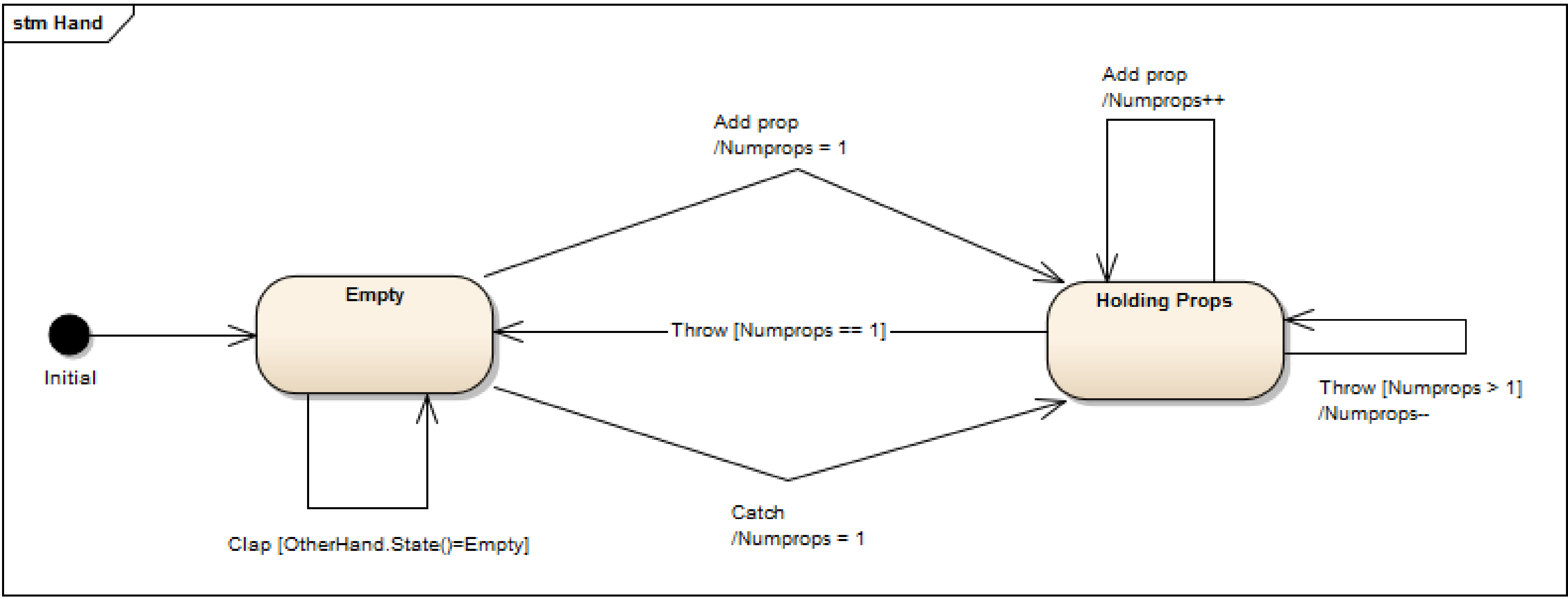
+ drops and bounces

Juggling Prop (4)



Explicit final state for drops

Juggling Hand



Summary

- State machines are widely used and valued for modelling complex behaviour
- UML state machines provide a rich notation set for expressing these models
- Different state-based views of one system or object are possible
  - Decision must be made as to what behaviour and at what level of abstraction we are trying to model