

# **UML - Statecharts**

# Behavioral Modeling

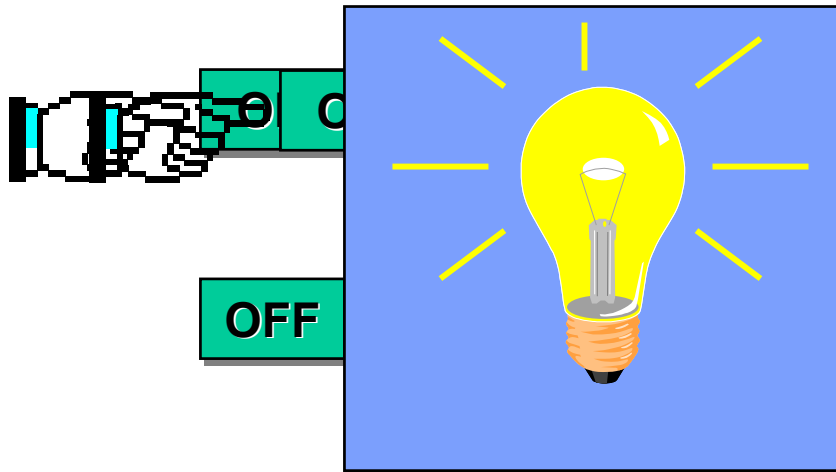
- Interactions and Collaborations
- Statecharts
- Activity Diagrams

# Overview

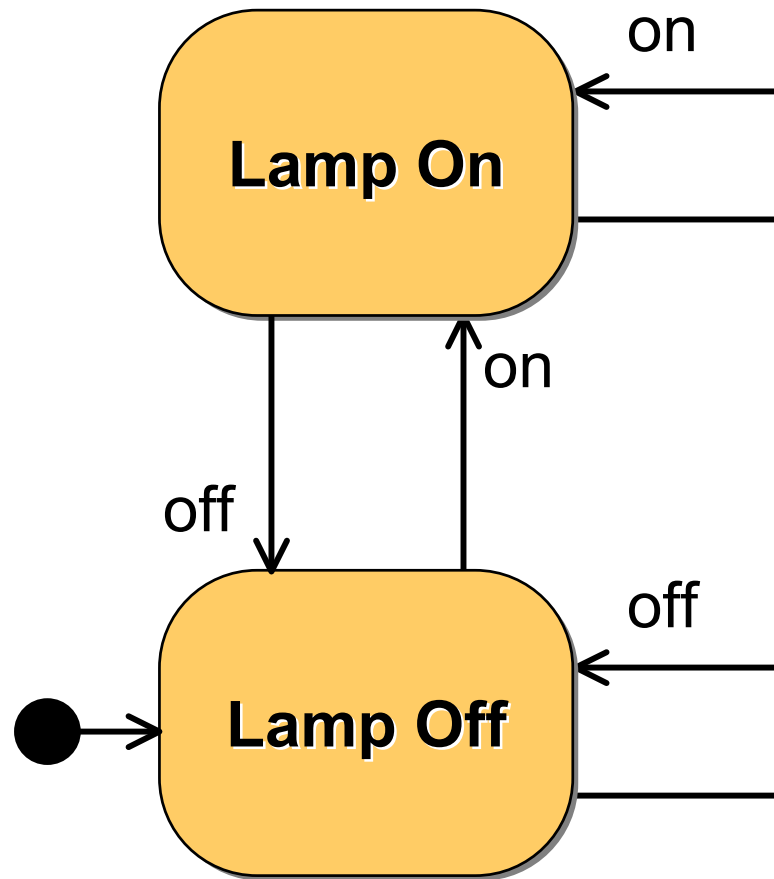
- **Basic State Machine Concepts**
- Statecharts and Objects
- Advanced Modeling Concepts
- Case Study

# Automata

- A machine whose output behavior is not only a direct consequence of the current input, but of some past history of its inputs
- Characterized by an internal **state** which represents this past experience

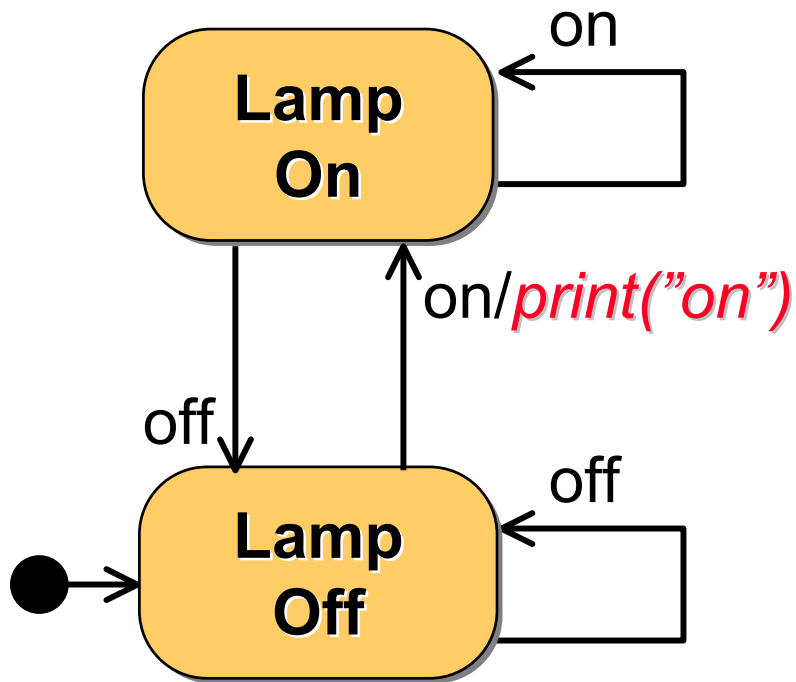


# State Machine (Automaton) Diagram

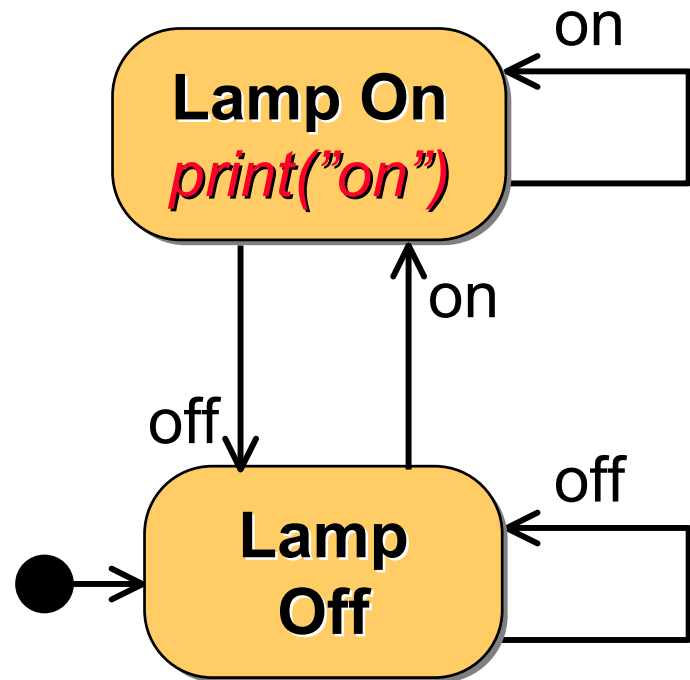


# Outputs and Actions

- As the automaton changes state it can generate outputs:



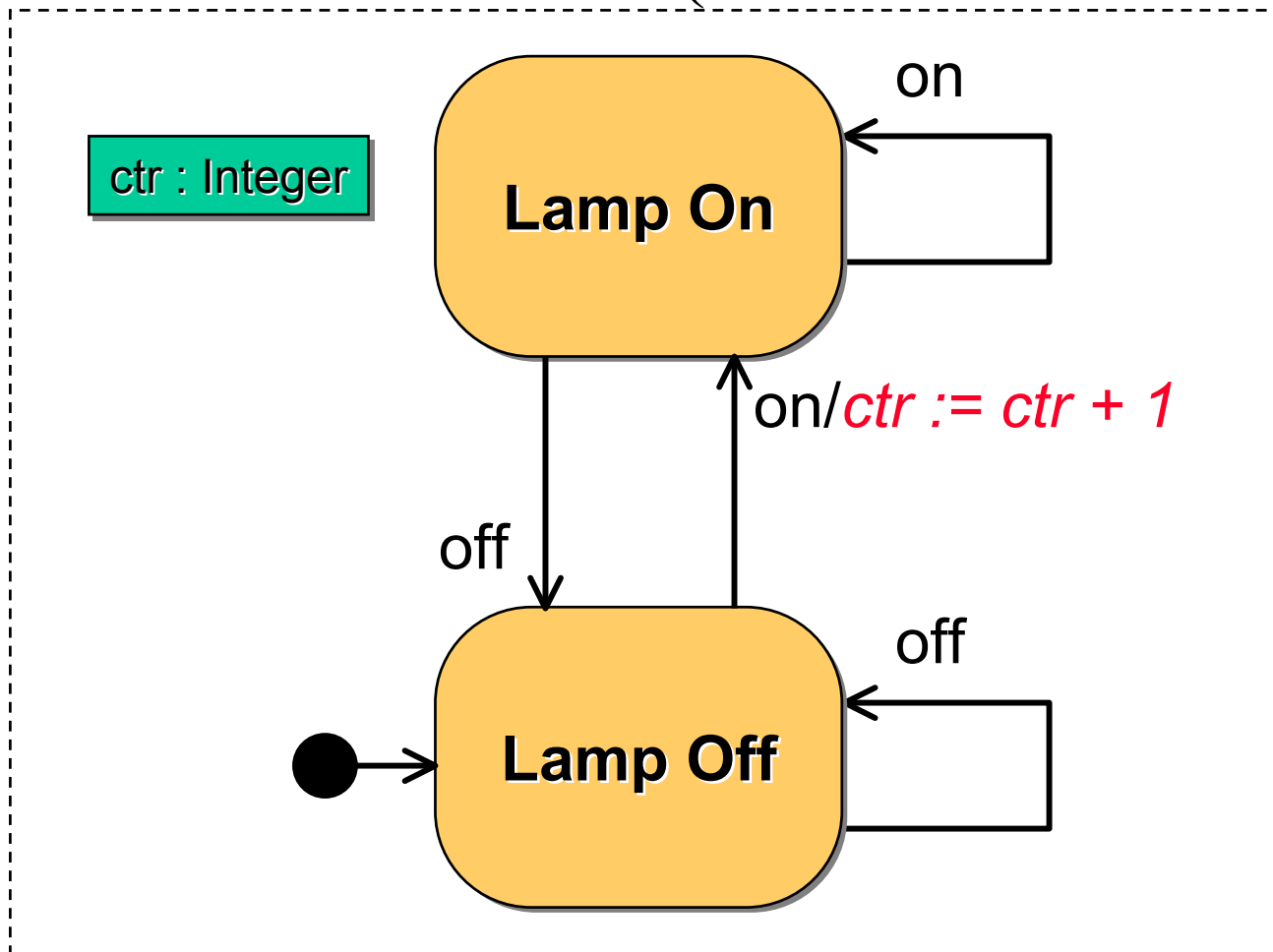
**Mealy** automaton



**Moore** automaton

# Extended State Machines

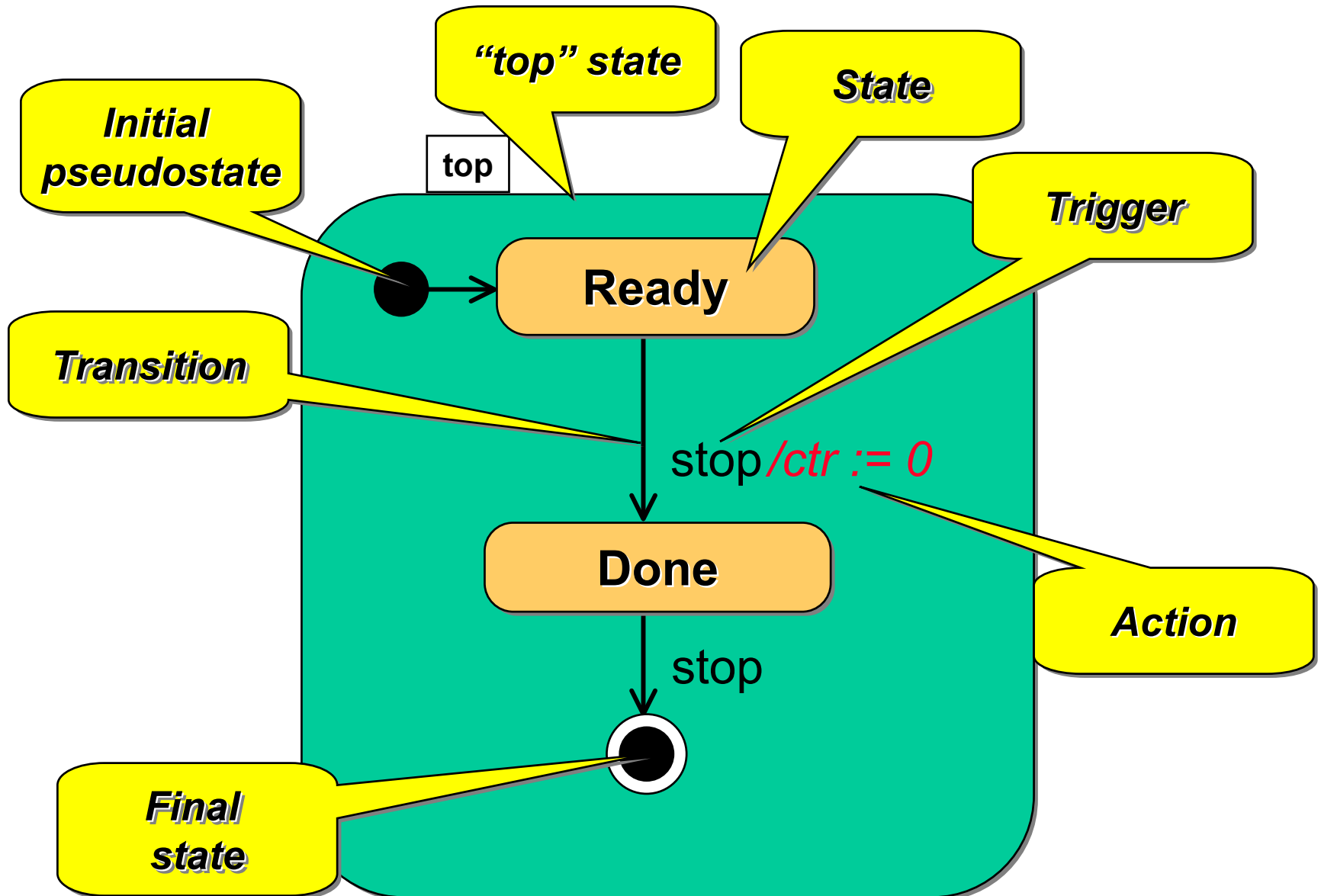
- Addition of variables (“**extended state**”)



# A Bit of Theory

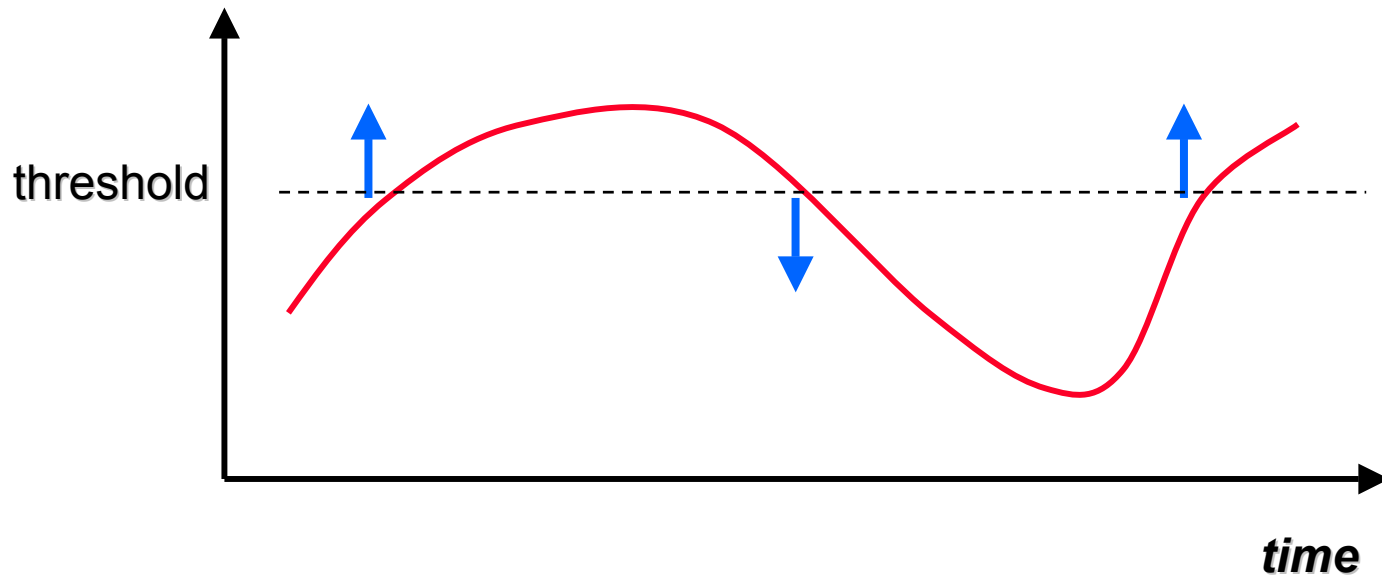
- An extended (Mealy) state machine is defined by:
  - a set of input signals (input alphabet)
  - a set of output signals (output alphabet)
  - a set of states
  - a set of transitions
    - triggering signal
    - action
  - a set of extended state variables
  - an initial state designation
  - a set of final states (if terminating automaton)

# Basic UML Statechart Diagram



# What Kind of Behavior?

- In general, state machines are suitable for describing event-driven, discrete behavior
- inappropriate for modeling continuous behavior



# Event-Driven Behavior

- **Event** = a *type* of observable occurrence
  - interactions:
    - synchronous object operation invocation (**call event**)
    - asynchronous signal reception (**signal event**)
  - occurrence of time instants (**time event**)
    - interval expiry: “after(t)”
    - calendar/clock time: “at(t)”
  - change in value of some entity (**change event**): “when(e)”
- **Event Instance** = an instance of an event (type)
  - occurs at a particular time instant and has *no duration*

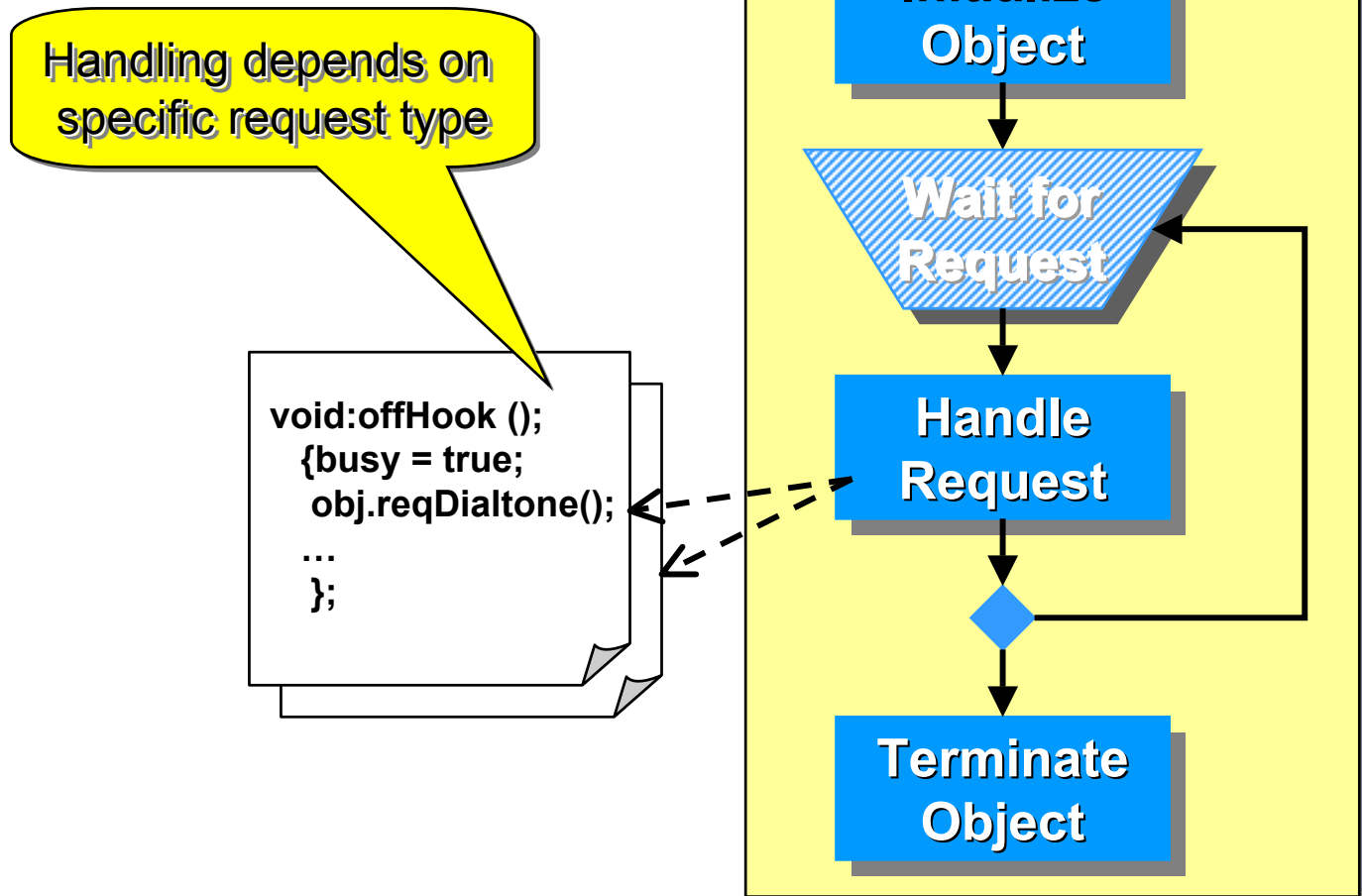
# The Behavior of What?

- In principle, anything that manifests event-driven behavior
- NB: there is no support currently in UML for modeling continuous behavior
- In practice:
  - the behavior of individual objects
  - typically associated with classifiers (e.g. Class, Package,...)

- Basic State Machine Concepts
- Statecharts and Objects
- Advanced Modeling Concepts
- Case Study

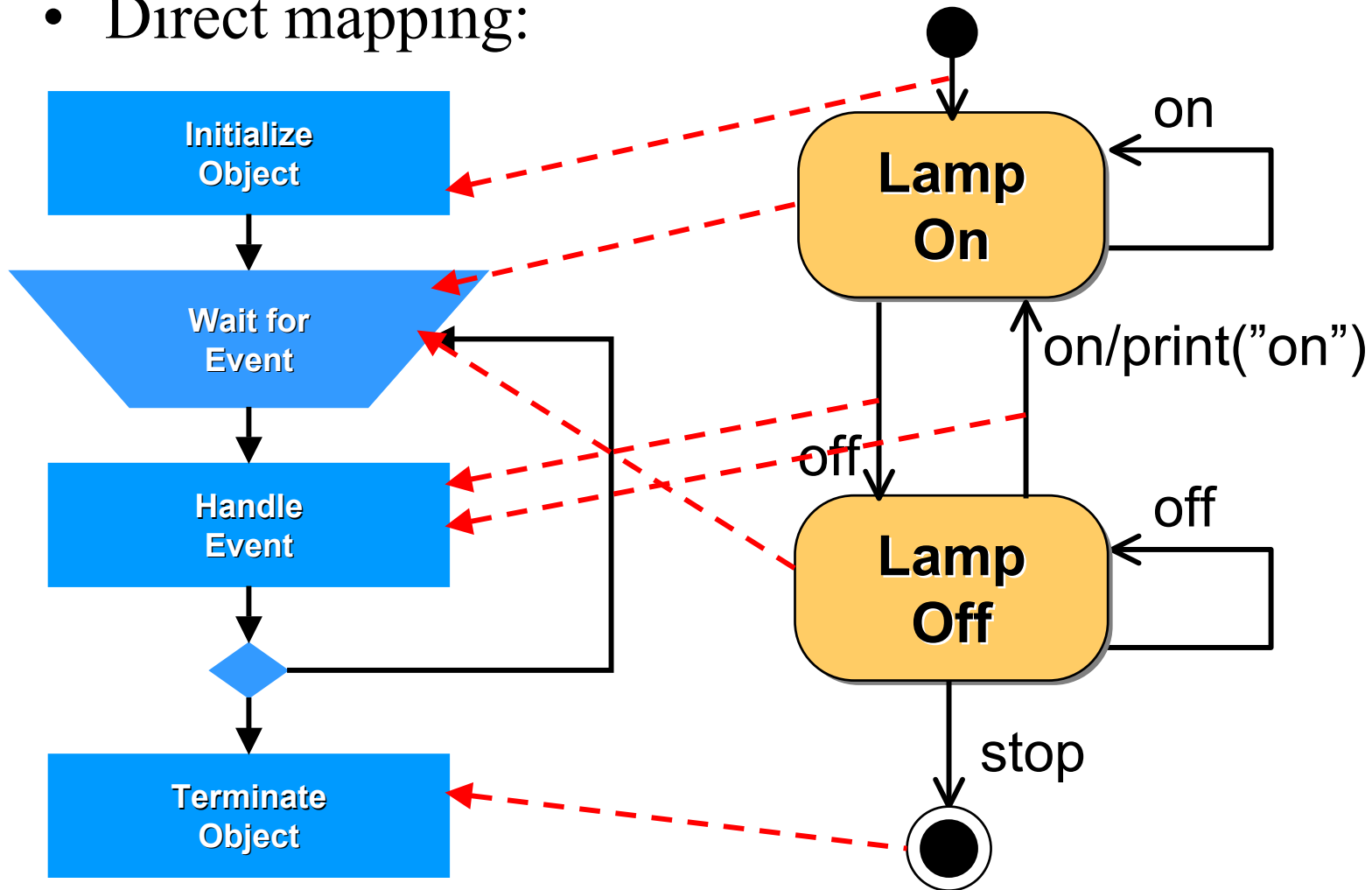
# Object Behavior - General Model

- Simple **server** model:



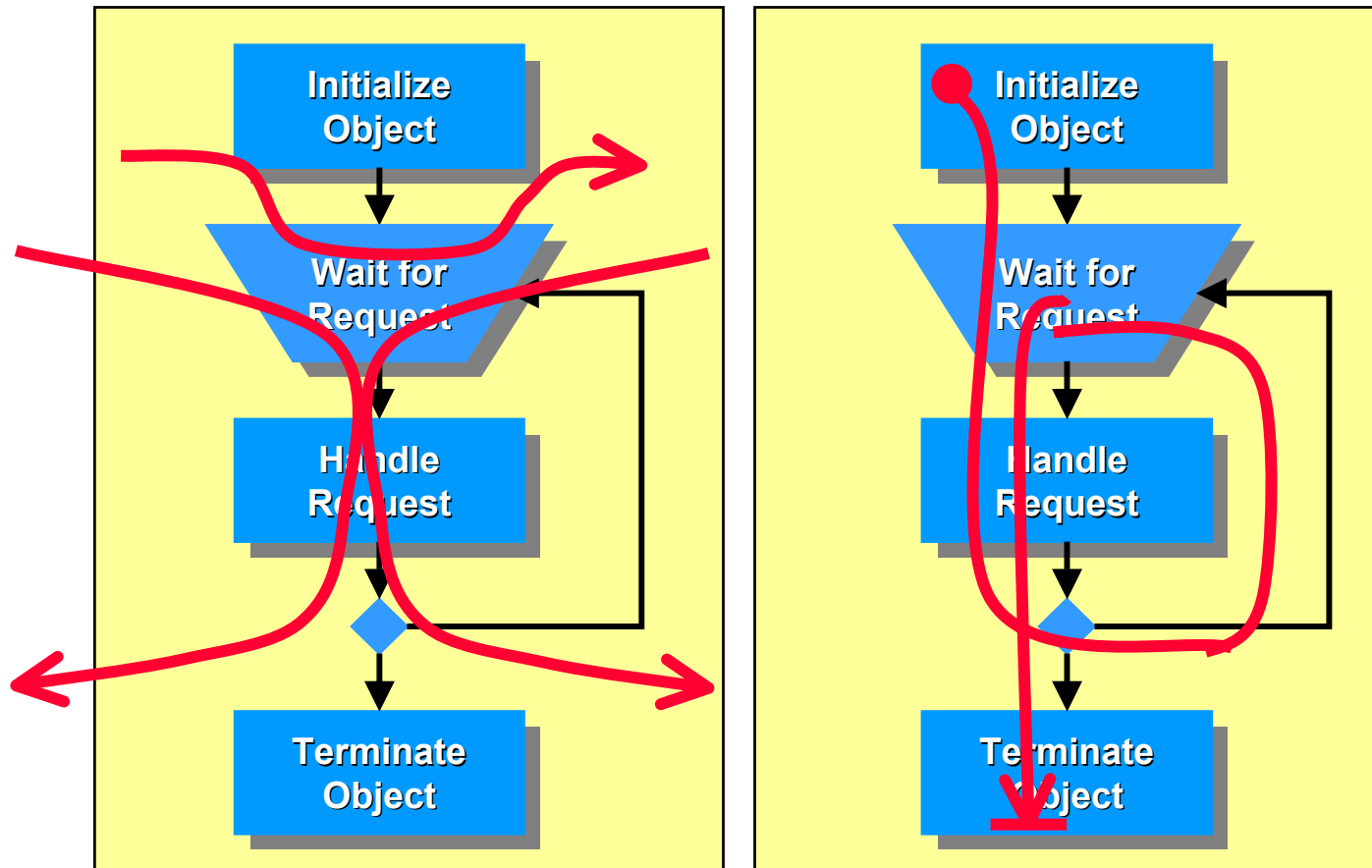
# Object Behavior and State Machines

- Direct mapping:

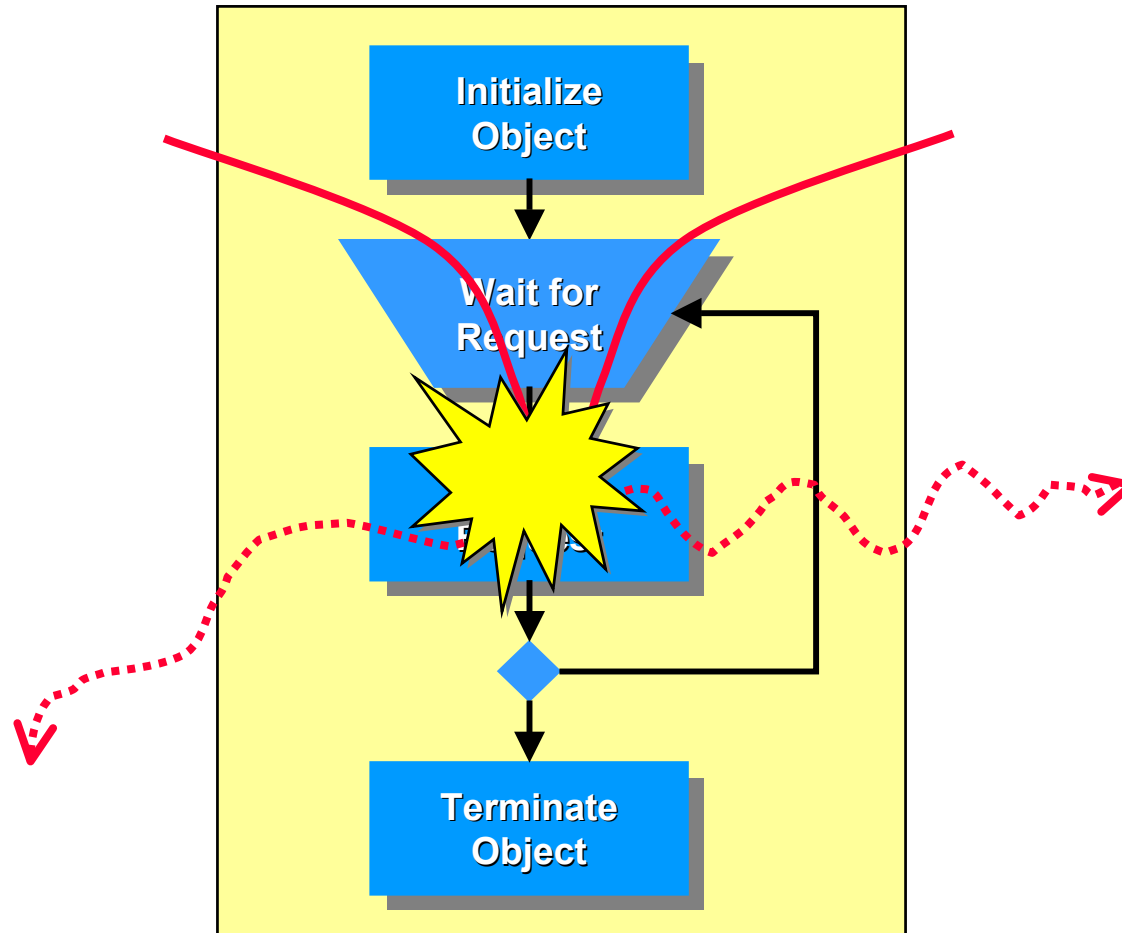


# Object and Threads

- **Passive objects:** external thread of execution
- **Active objects:** own thread of execution



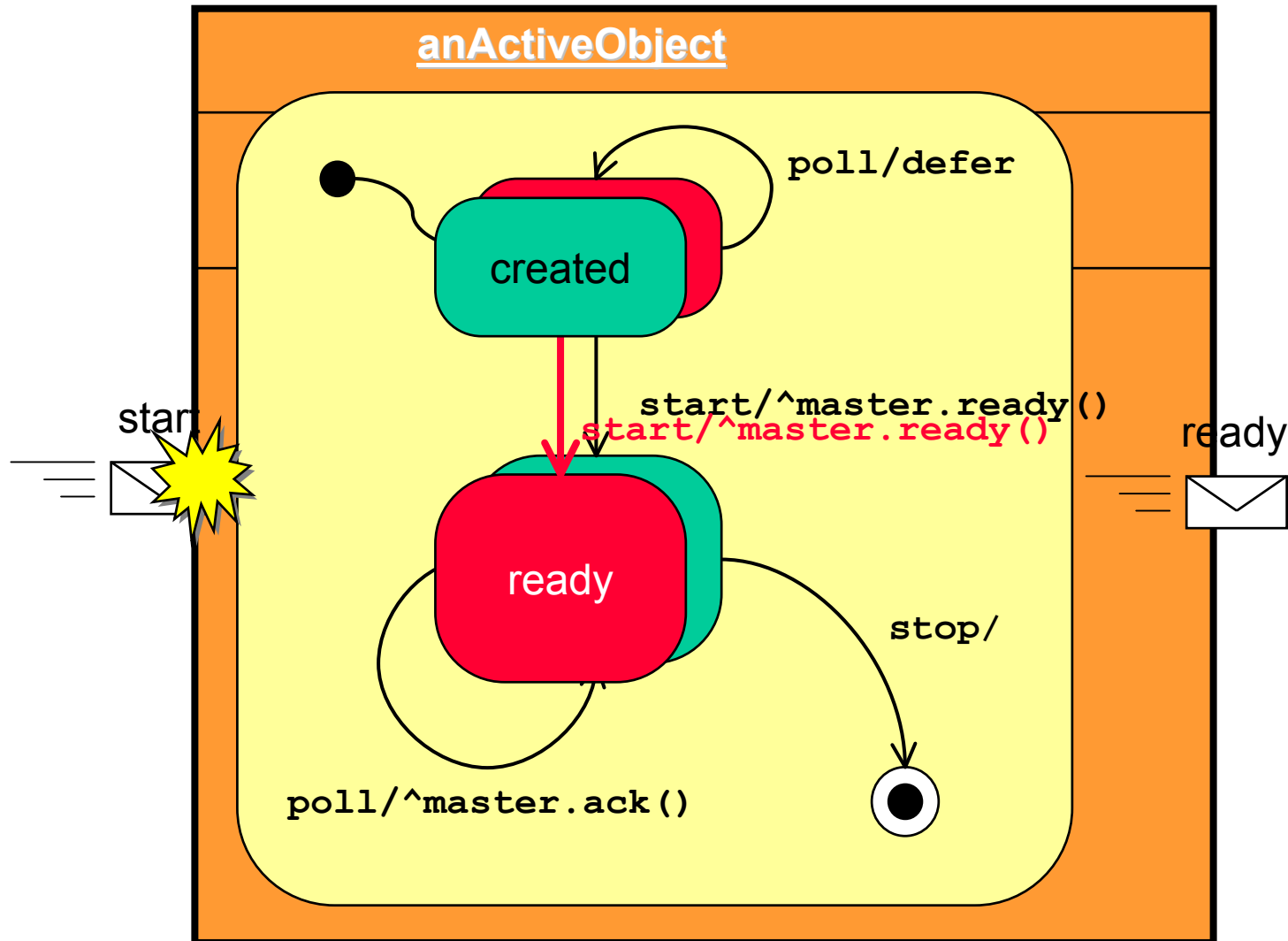
# Passive Objects: Dynamic Semantics



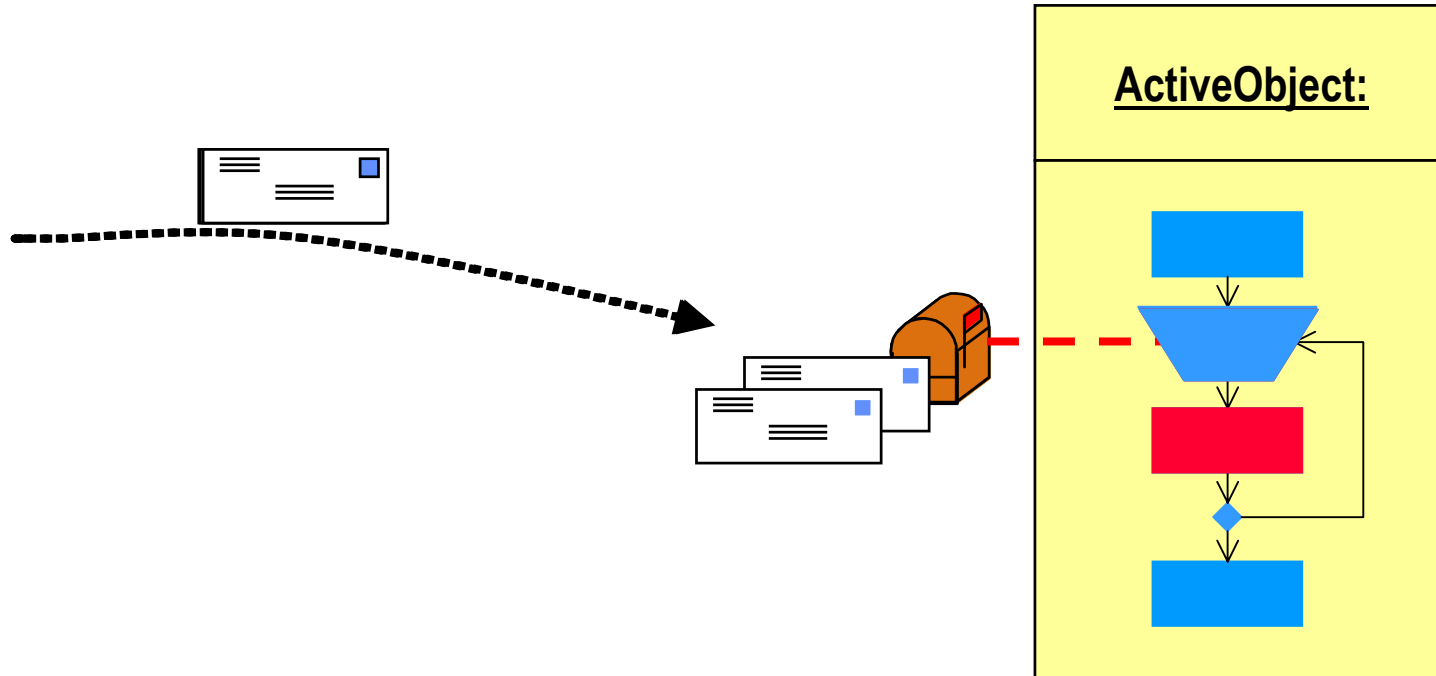
- Encapsulation does not protect the object from concurrency conflicts!
  - Explicit synchronization is still required

# Active Objects and State Machines

- Objects that encapsulate own thread of execution



# Active Objects: Dynamic Semantics

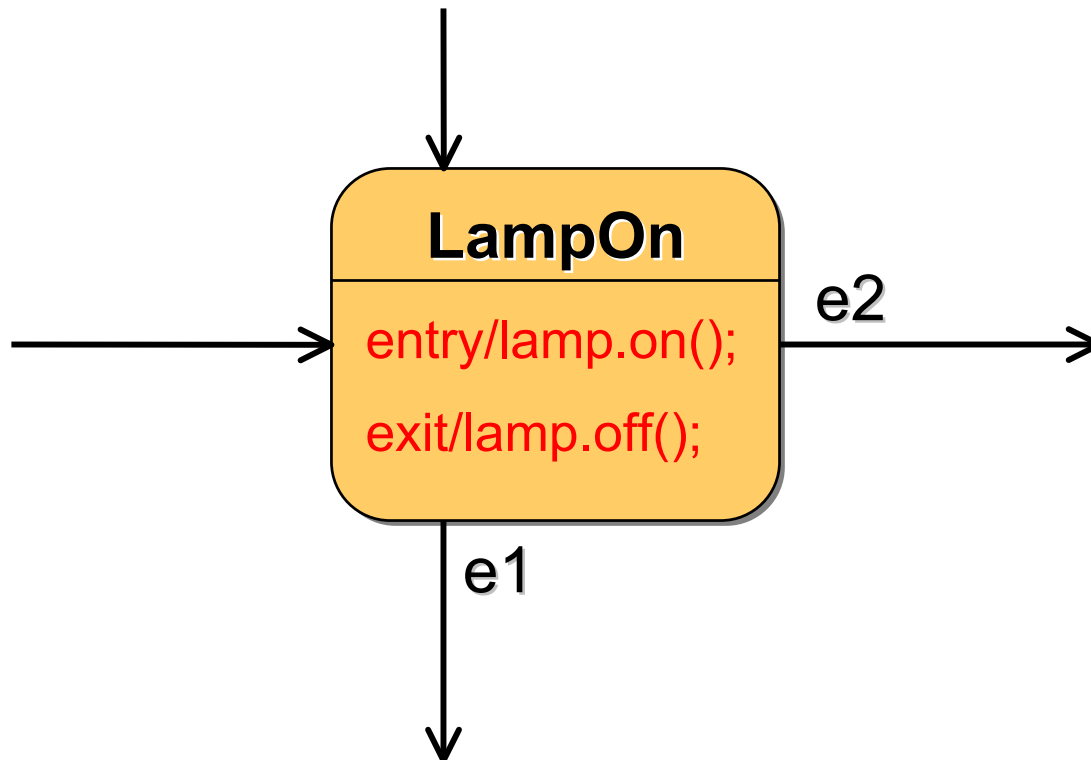


- **Run-to-completion model:**
  - serialized event handling
  - next event is processed after previous completed

- Basic State Machine Concepts
- Statecharts and Objects
- **Advanced Modeling Concepts**
- Case Study

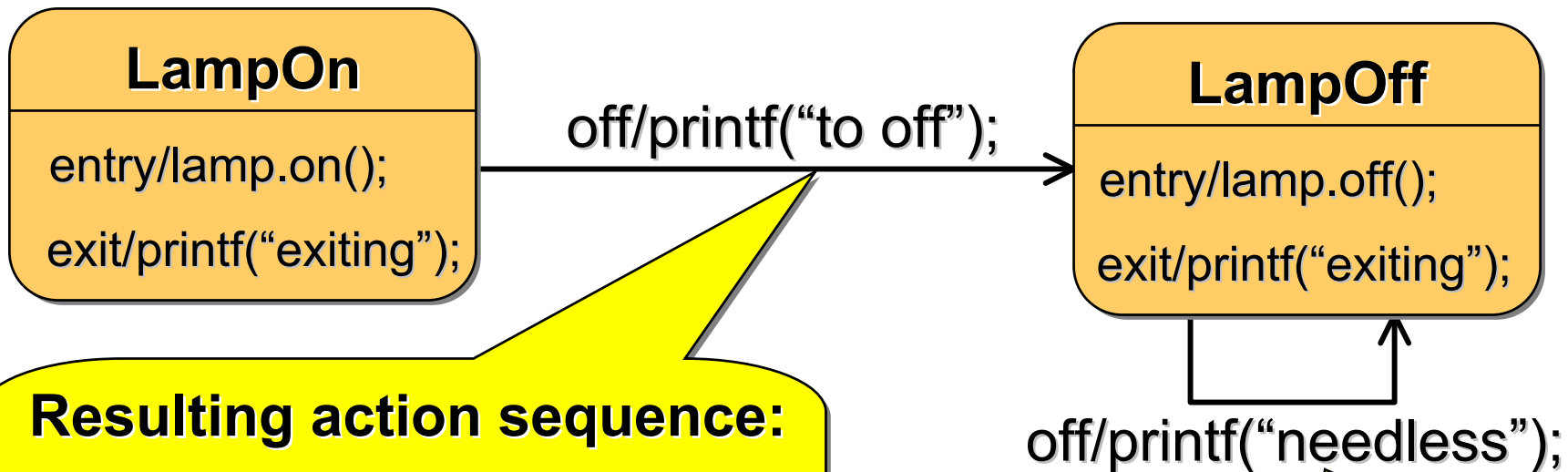
# State Entry and Exit Actions

- A dynamic assertion mechanism



# Order of Actions: Simple Case

- Exit actions prefix transition actions
- Entry action postfix transition actions



**Resulting action sequence:**

```
        ("to off") ;  
lamp.off() ;
```

```
printf("exiting") ;  
printf("needless") ;  
lamp.off() ;
```

# Internal Transitions

- Self-transitions that bypass entry and exit actions

Internal transition  
triggered by  
an “off” event

## LampOff

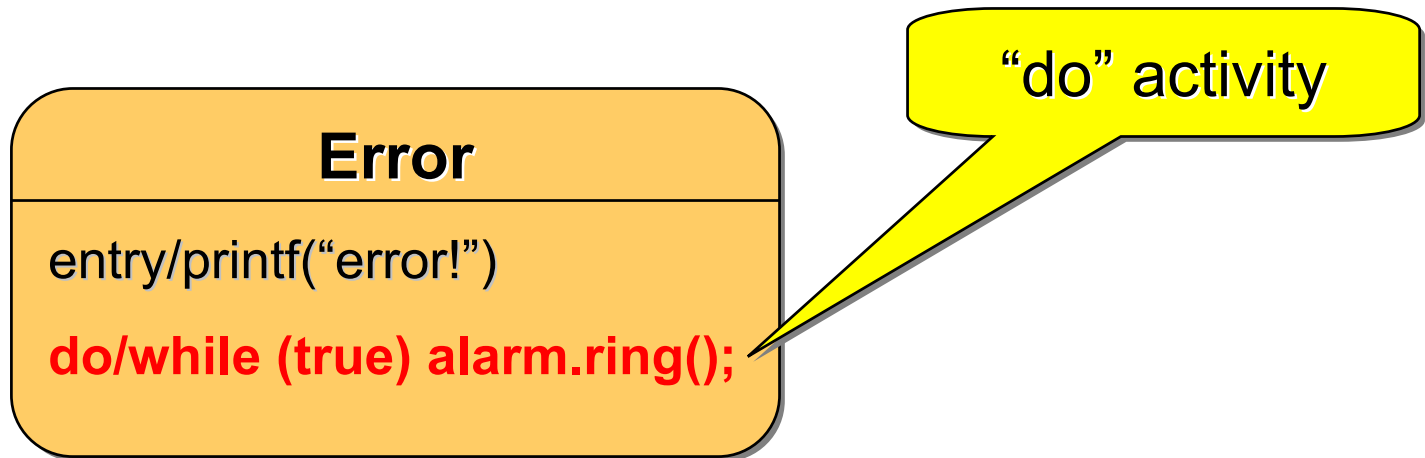
entry/lamp.off();

exit/printf(“exiting”);

**off/null;**

# State (“Do”) Activities

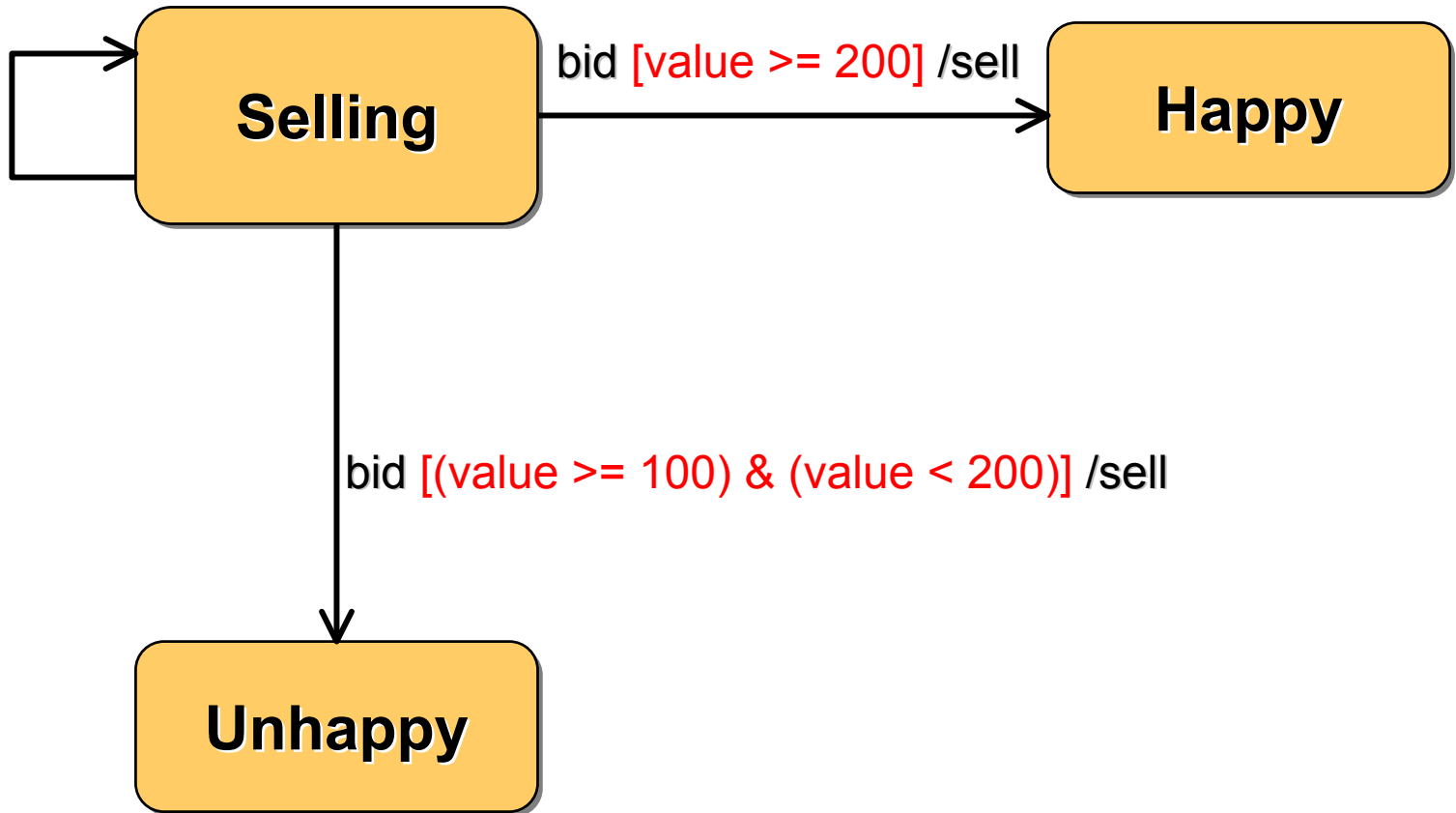
- Forks a concurrent thread that executes until:
  - the action completes or
  - the state is exited through an outgoing transition



# Guards

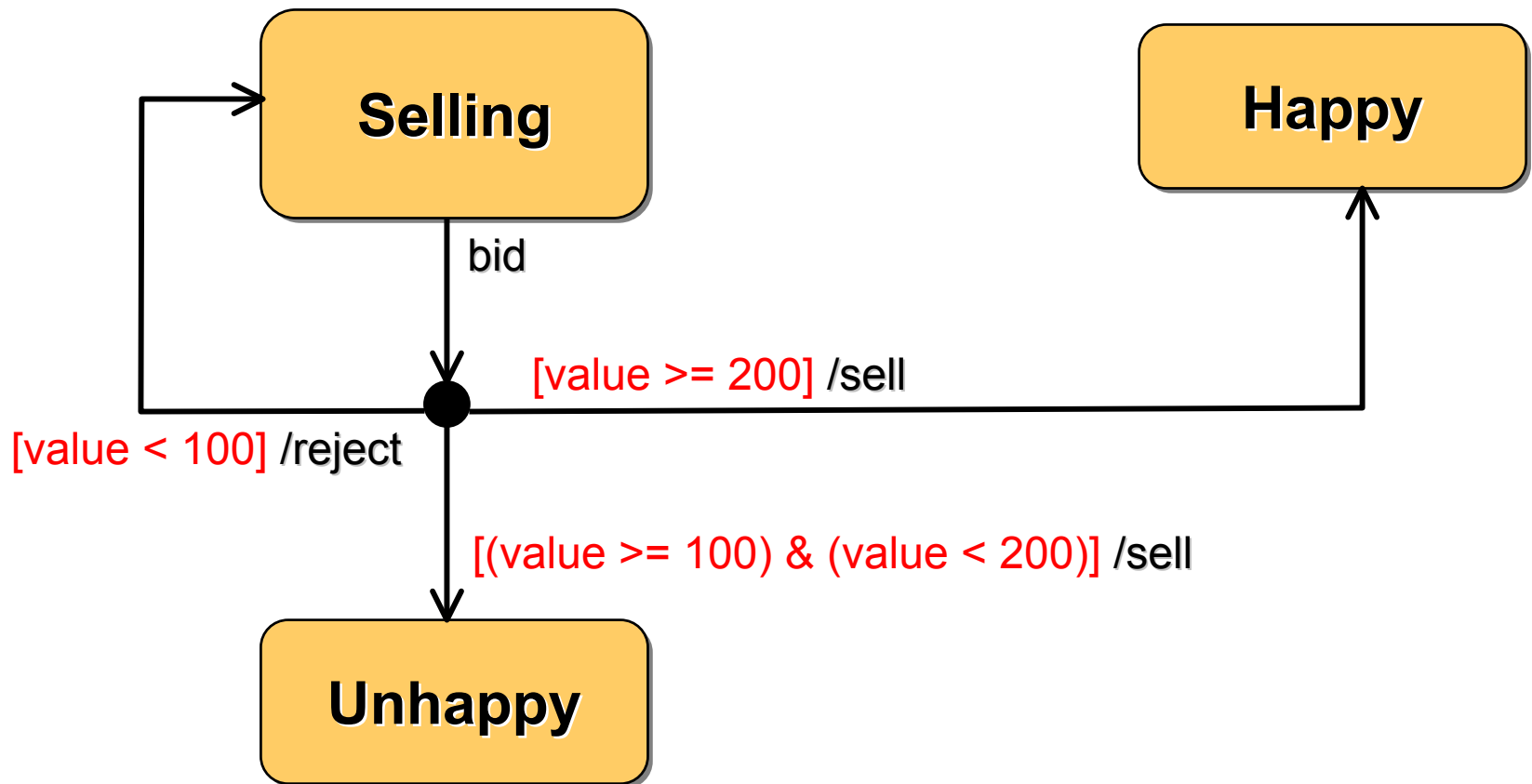
- Conditional execution of transitions
  - guards (Boolean predicates) must be side-effect free

bid [value < 100] /reject



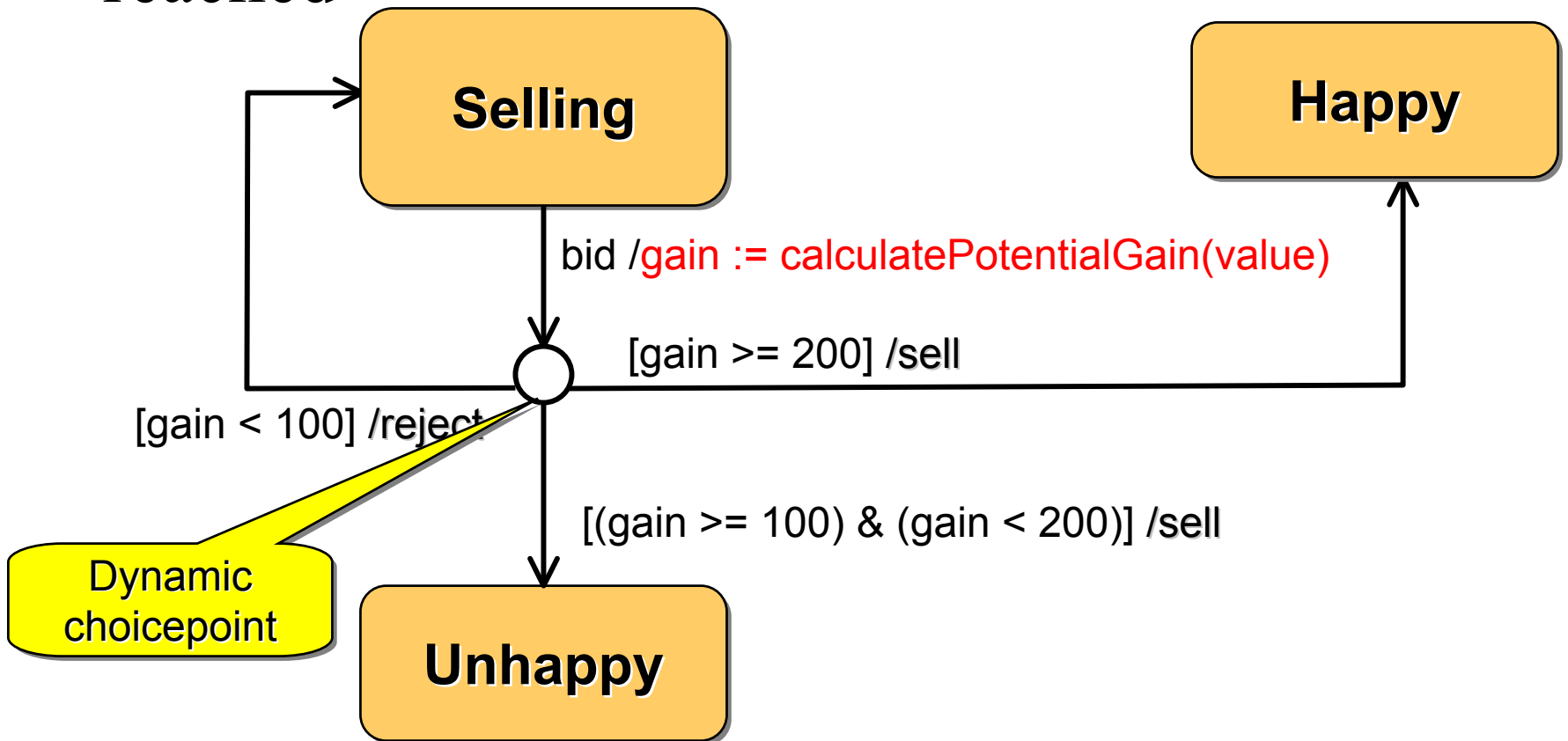
# Static Conditional Branching

- Merely a graphical shortcut for convenient rendering of decision trees



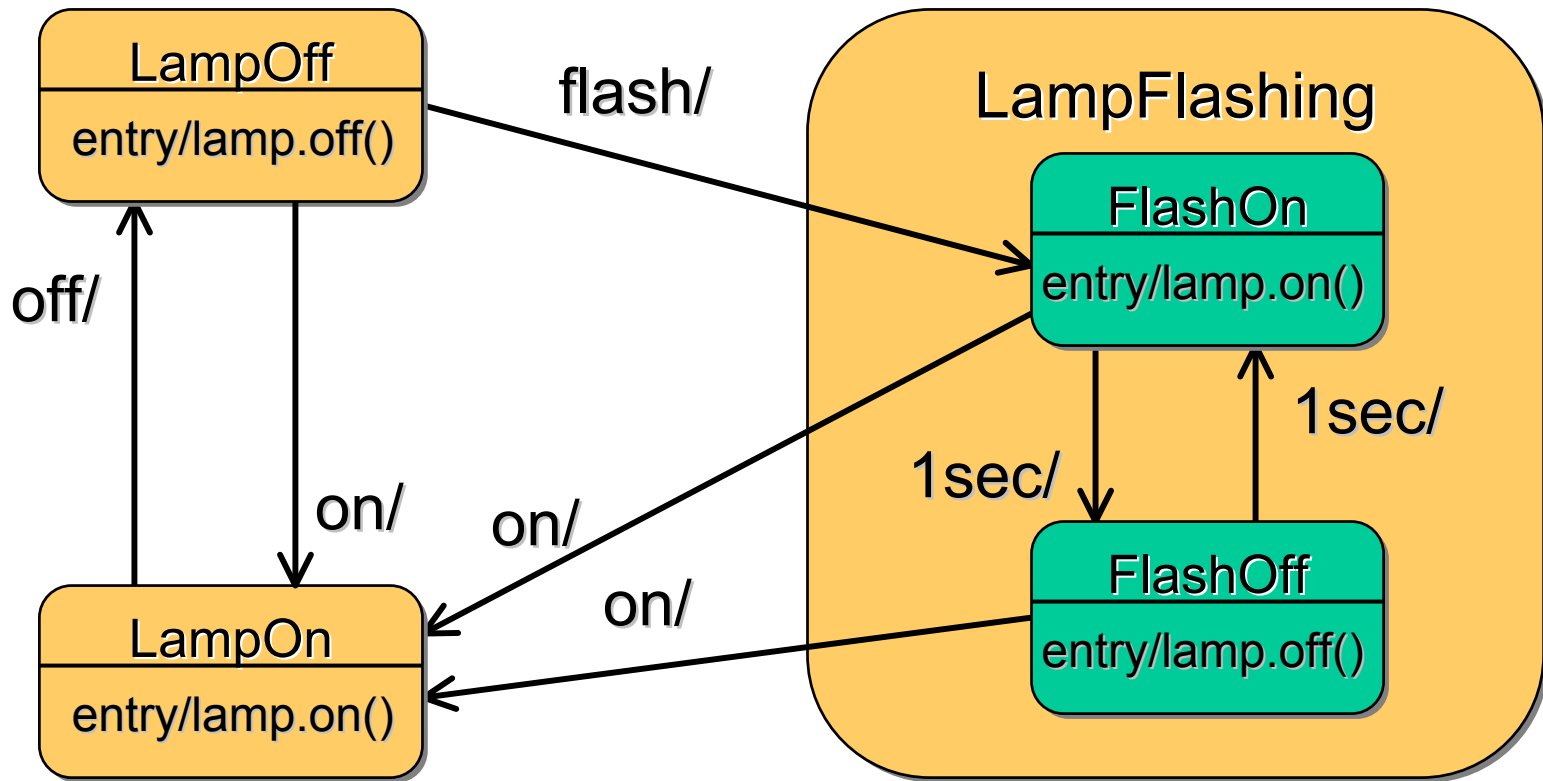
# Dynamic Conditional Branching

- **Choice** pseudostate: guards are evaluated at the instant when the decision point is reached



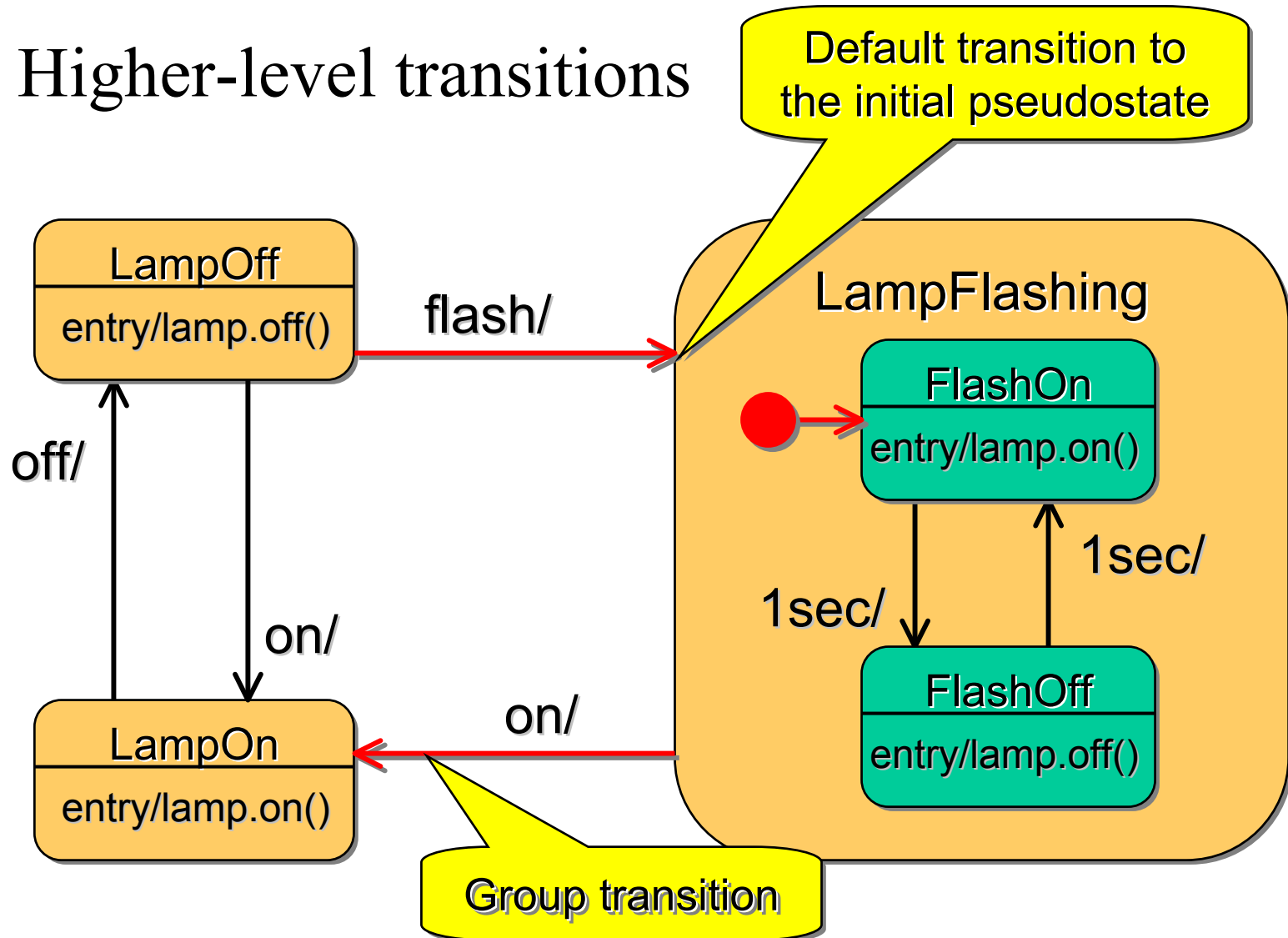
# Hierarchical State Machines

- Graduated attack on complexity
  - states decomposed into state machines



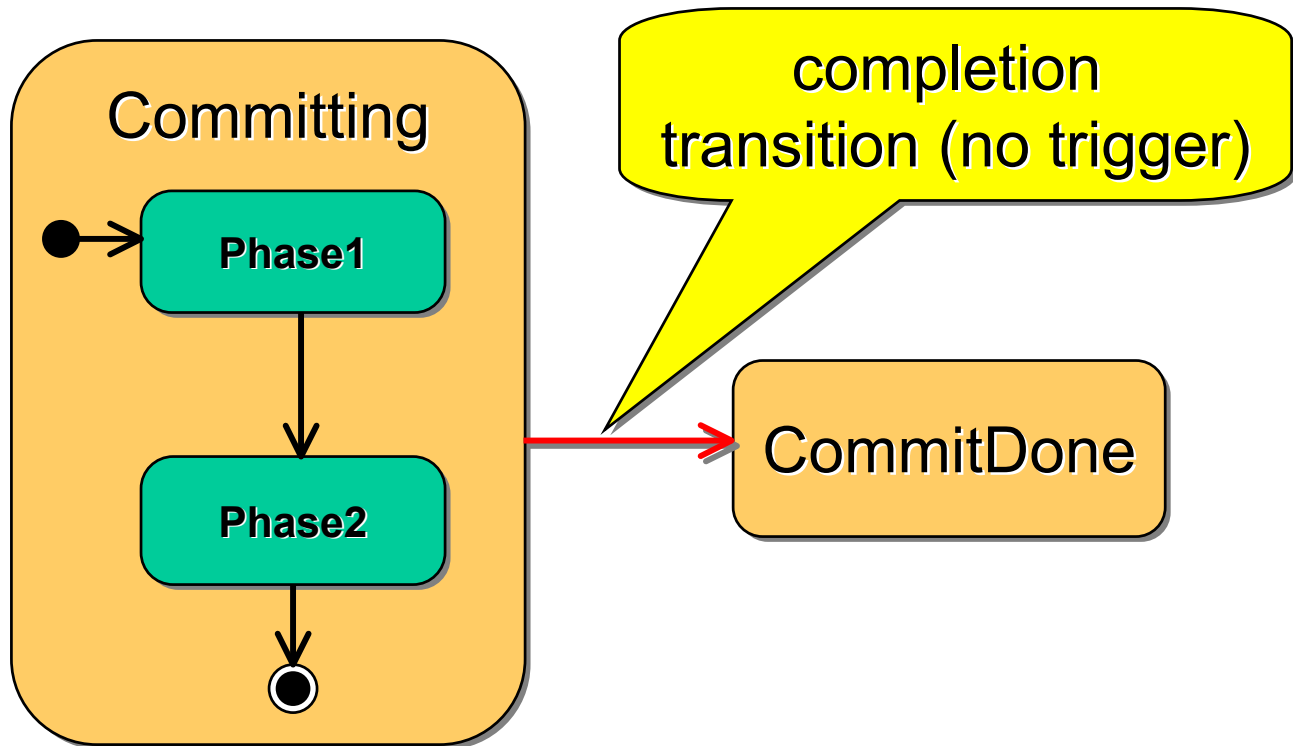
# Group Transitions

- Higher-level transitions



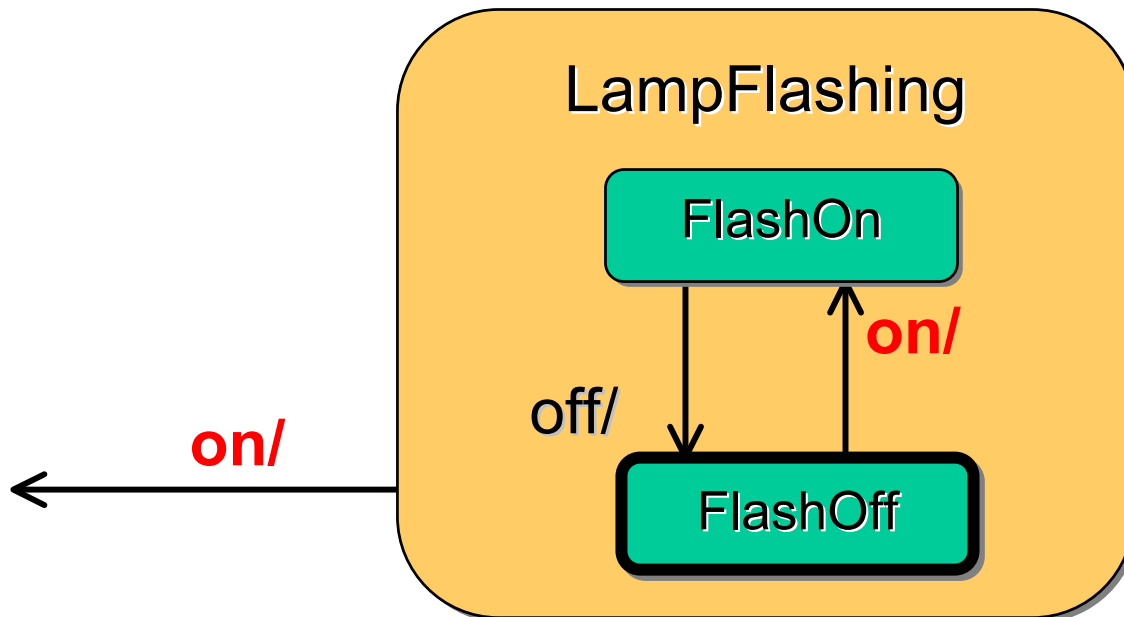
# Completion Transitions

- Triggered by a **completion event**
  - generated automatically when an immediately nested state machine terminates



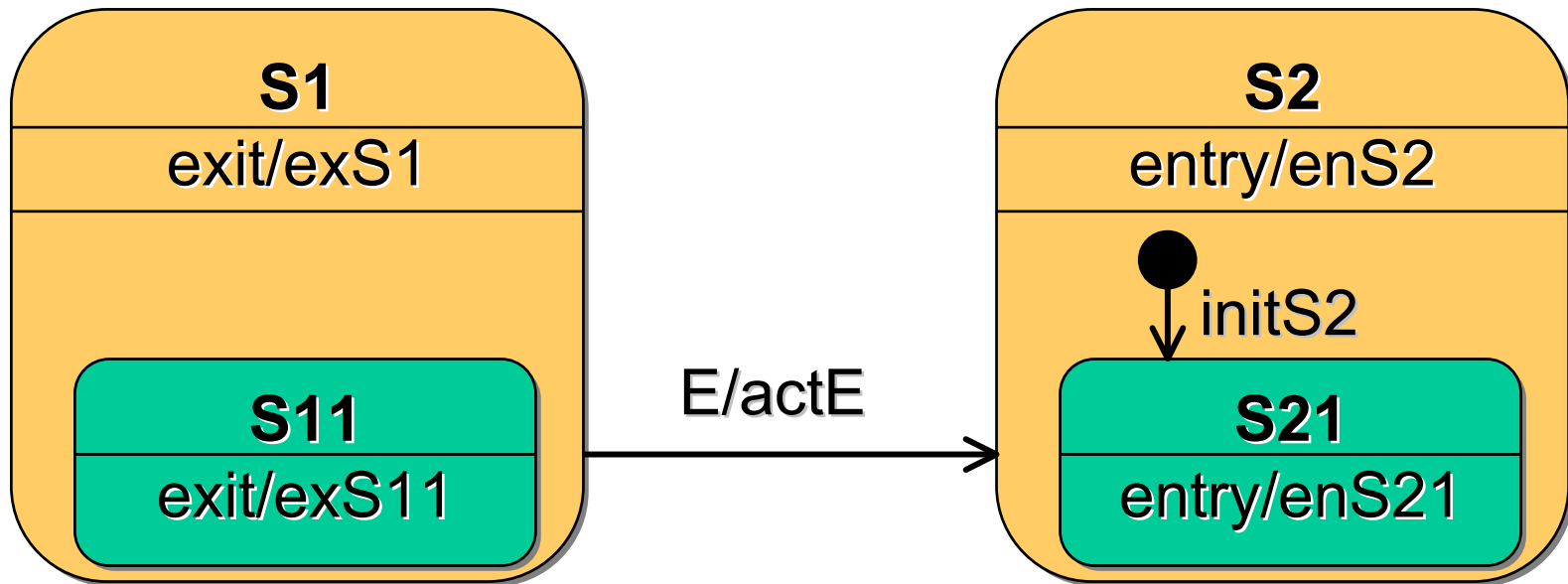
# Selection Rules

- Two or more transitions may have the same event trigger
  - innermost transition takes precedence
  - if no transition is triggered, event is discarded



# Order of Actions: Complex Case

- Same approach as for the simple case

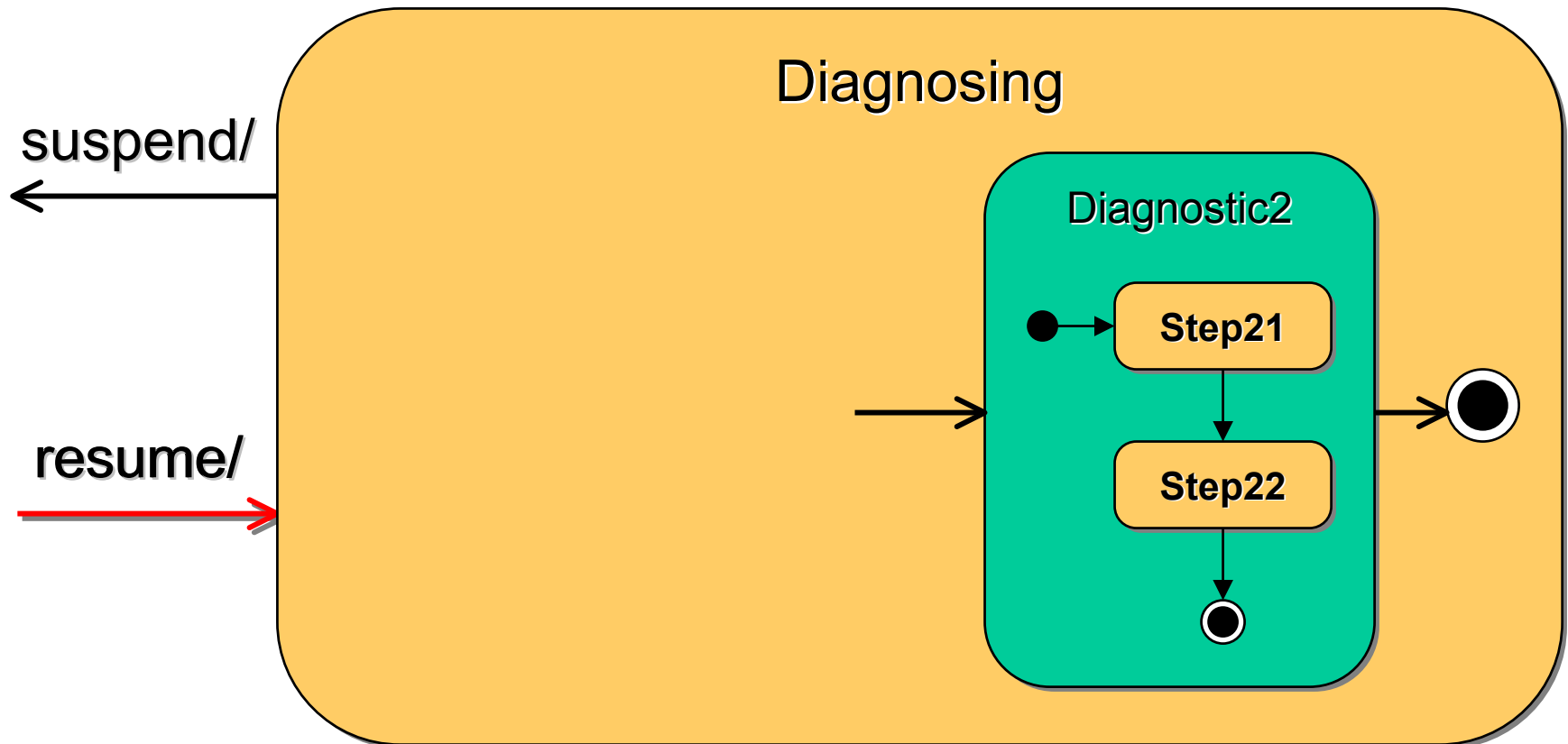


**Actions execution sequence:**

exS11  $\Rightarrow$  exS1  $\Rightarrow$  actE  $\Rightarrow$  enS2  $\Rightarrow$  initS2  $\Rightarrow$  enS21

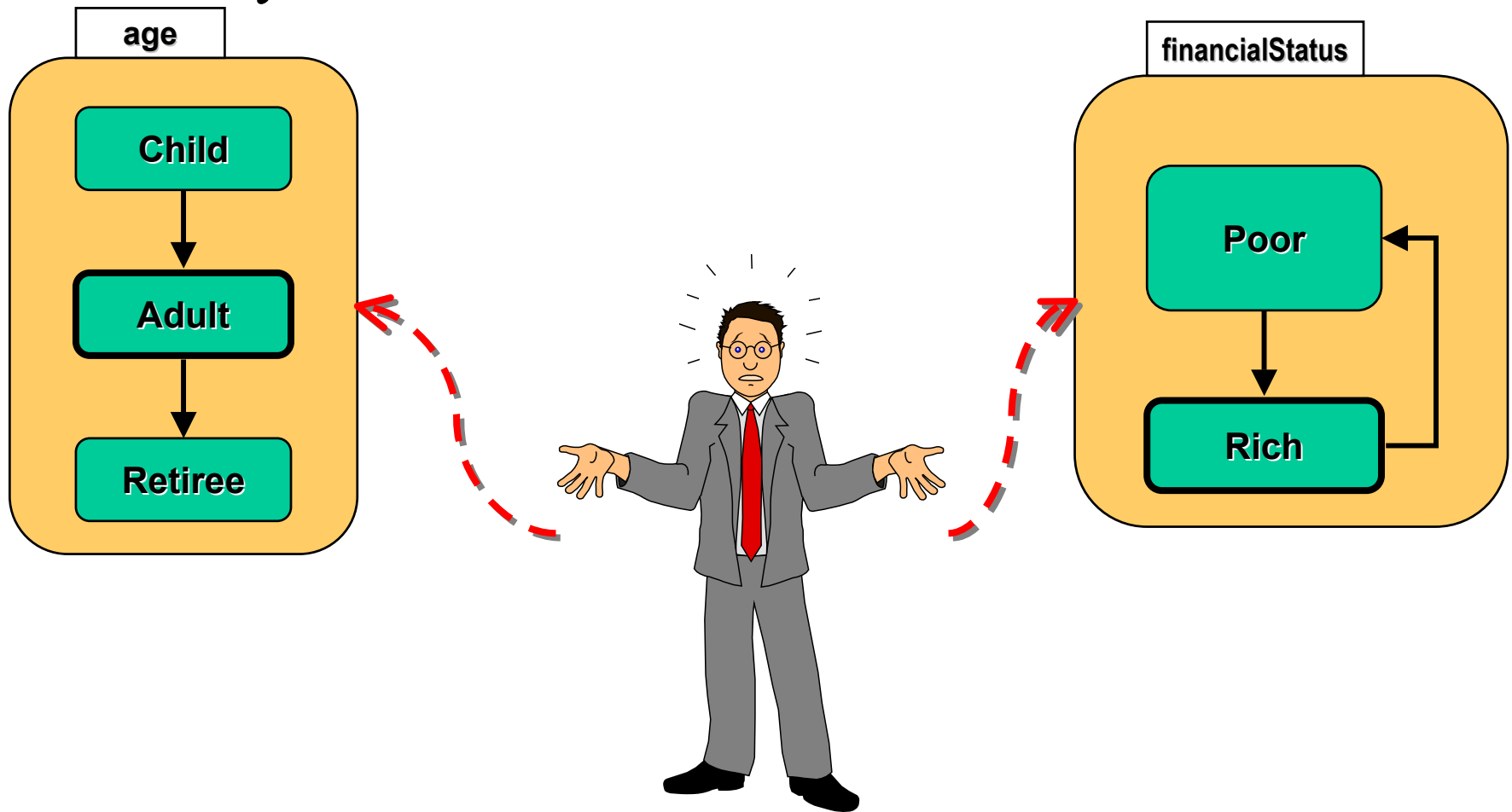
# History

- Return to a previously visited hierarchical state
  - deep and shallow history options



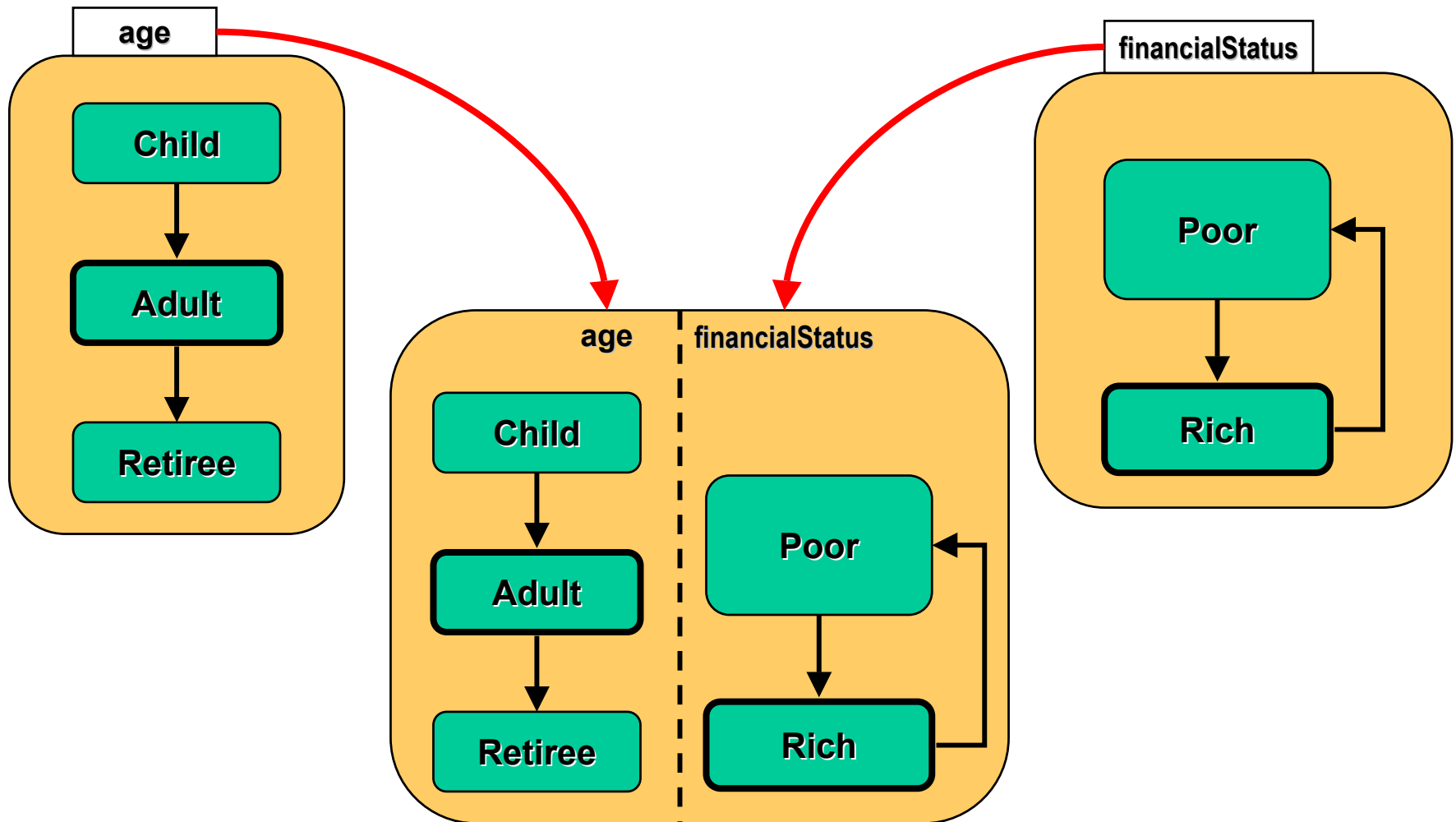
# Orthogonality

- Multiple simultaneous perspectives on the same entity



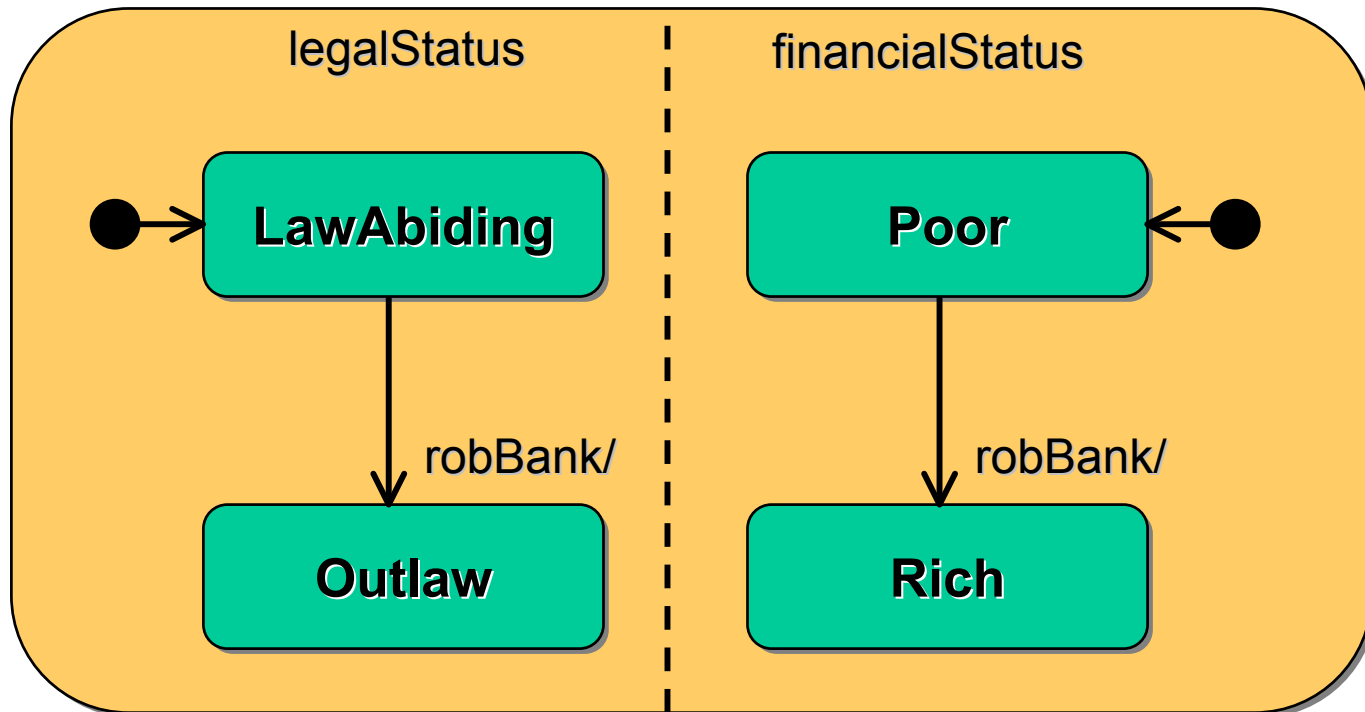
# Orthogonal Regions

- Combine multiple simultaneous descriptions



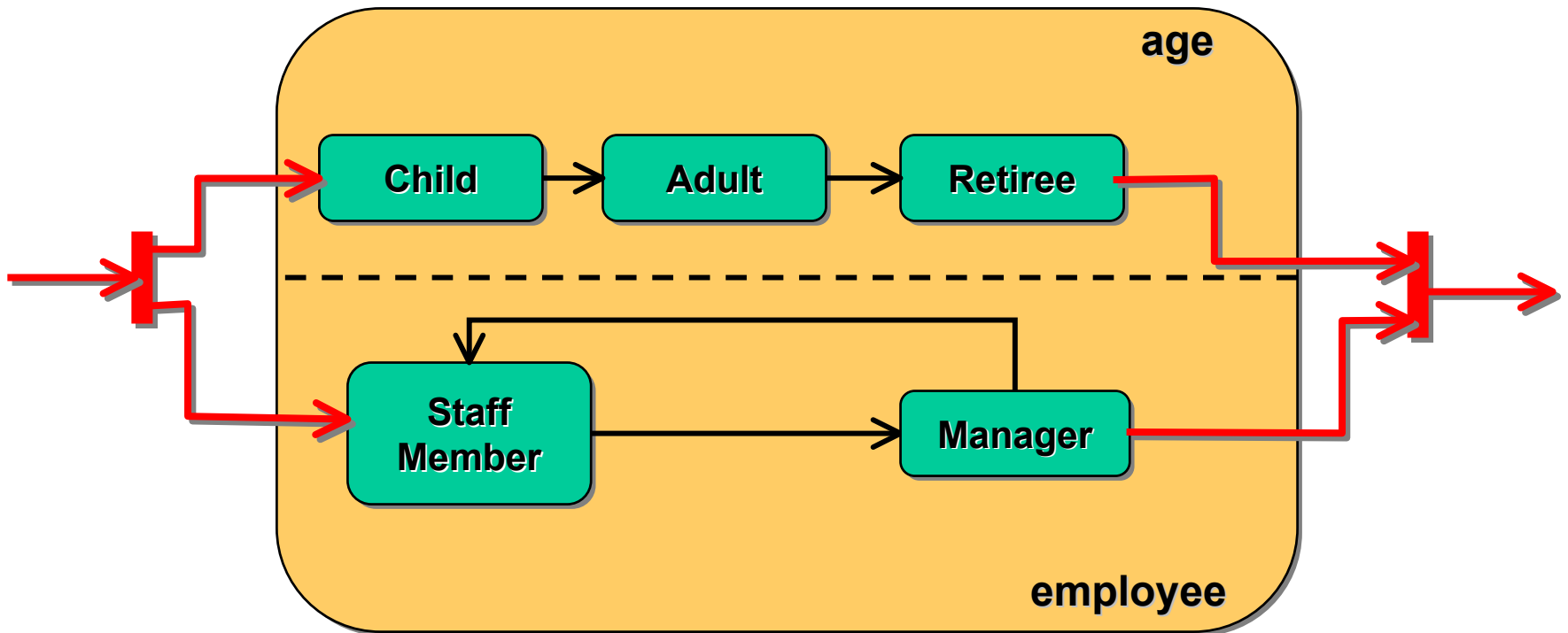
# Orthogonal Regions - Semantics

- All mutually orthogonal regions detect the same events and respond to them “simultaneously”
  - usually reduces to interleaving of some kind



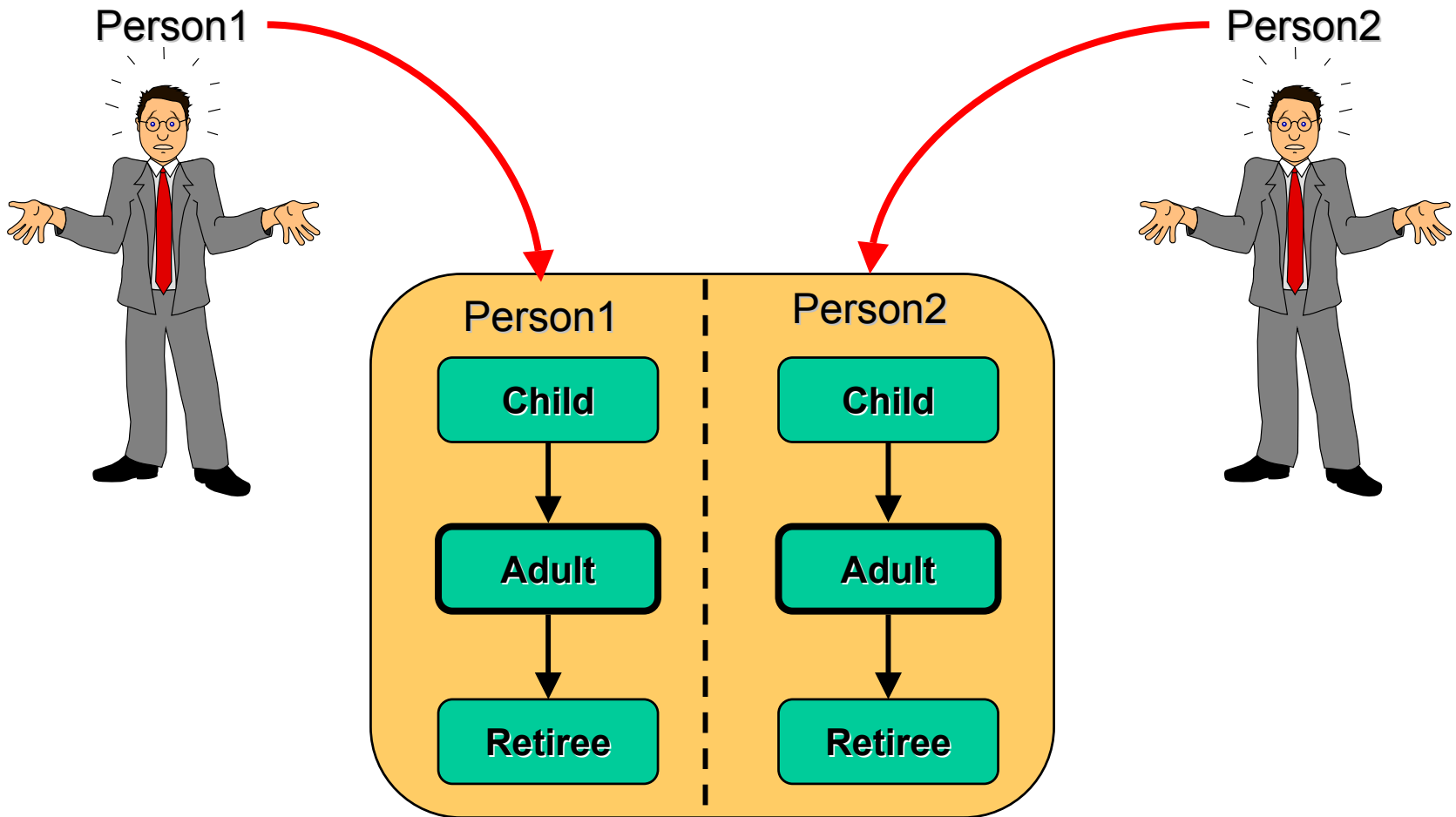
# Transition Forks and Joins

- For transitions into/out of orthogonal regions:



# Common Misuse of Orthogonality

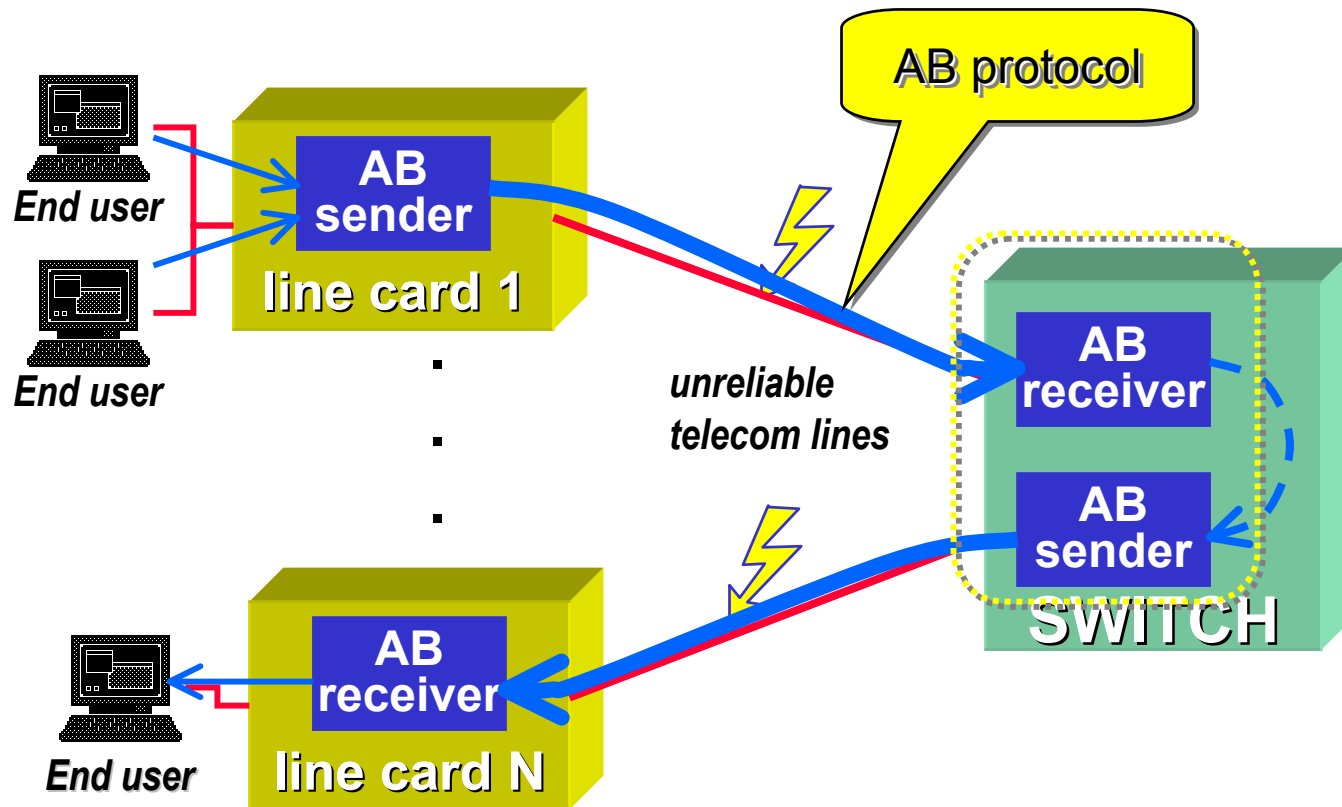
- Using regions to model independent objects



- Basic State Machine Concepts
- Statecharts and Objects
- Advanced Modeling Concepts
- Case Study

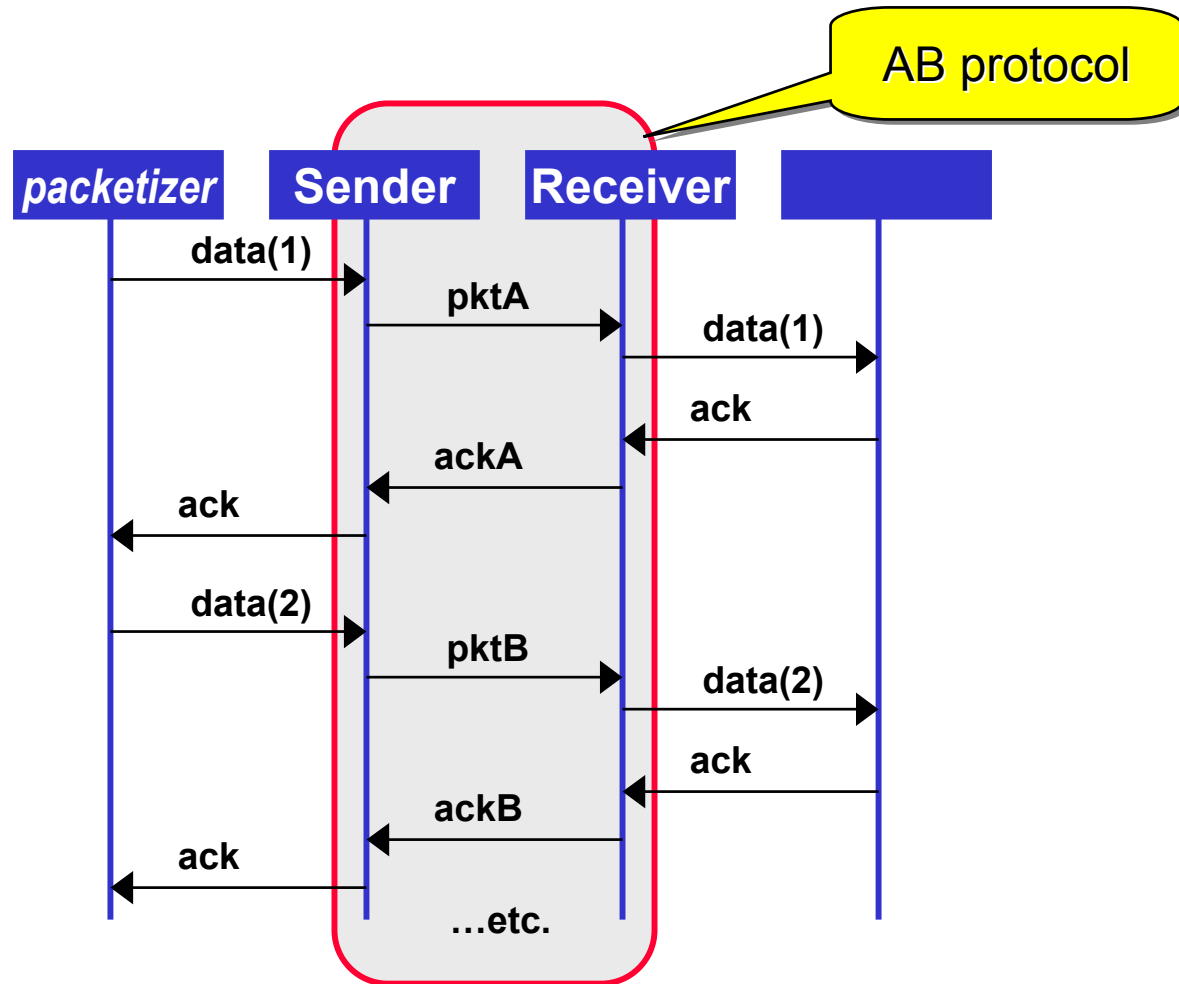
# Case Study: Protocol Handler

- A multi-line packet switch that uses the alternating-bit protocol as its link protocol



# Alternating Bit Protocol

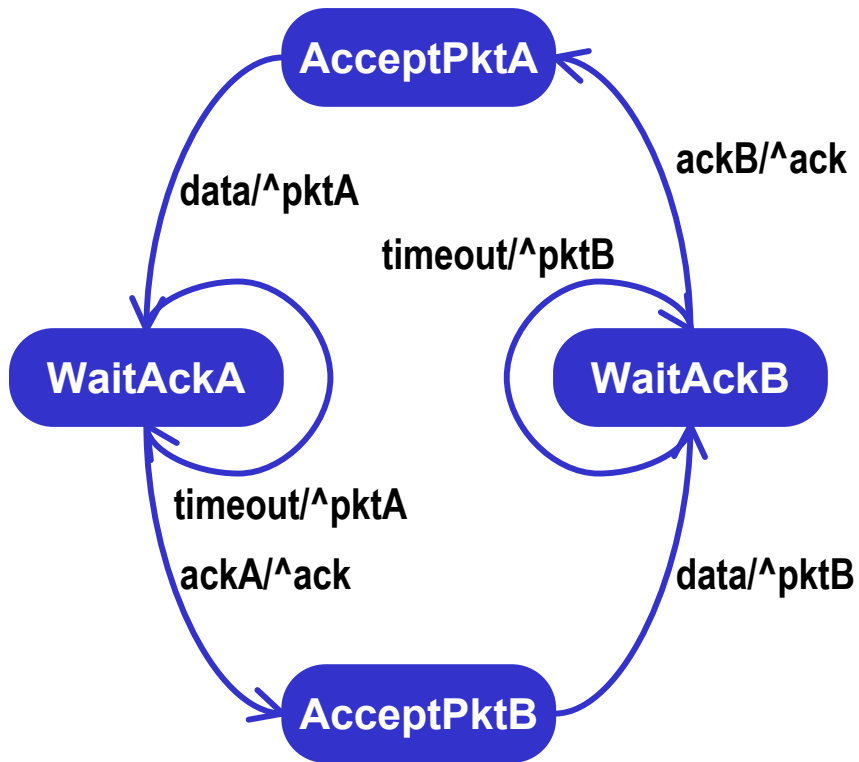
- A simple one-way point-to-point packet protocol



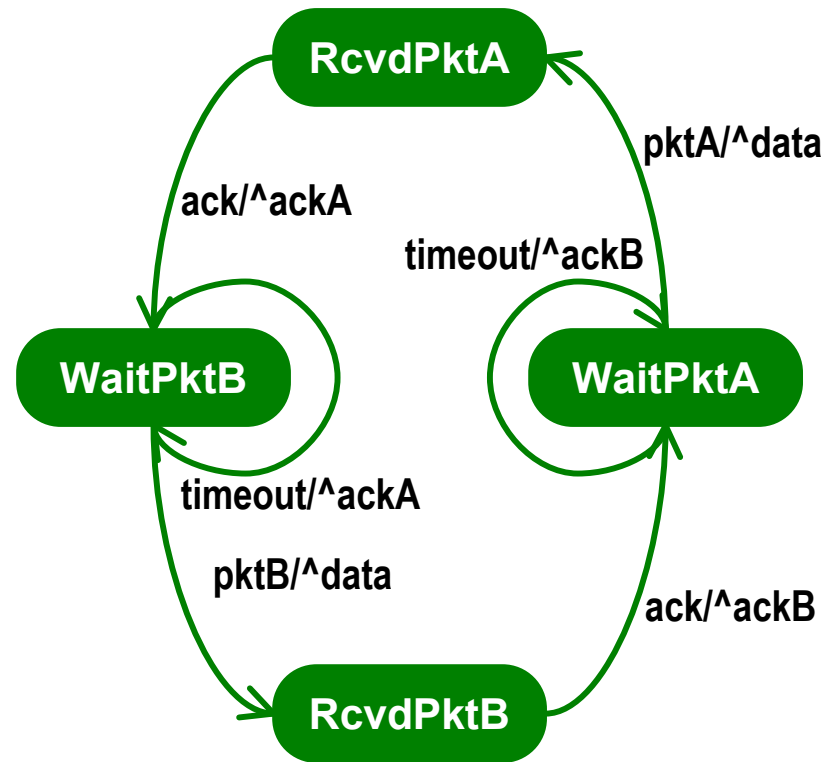
# Alternating Bit Protocol (2)

- State machine specification

Sender SM

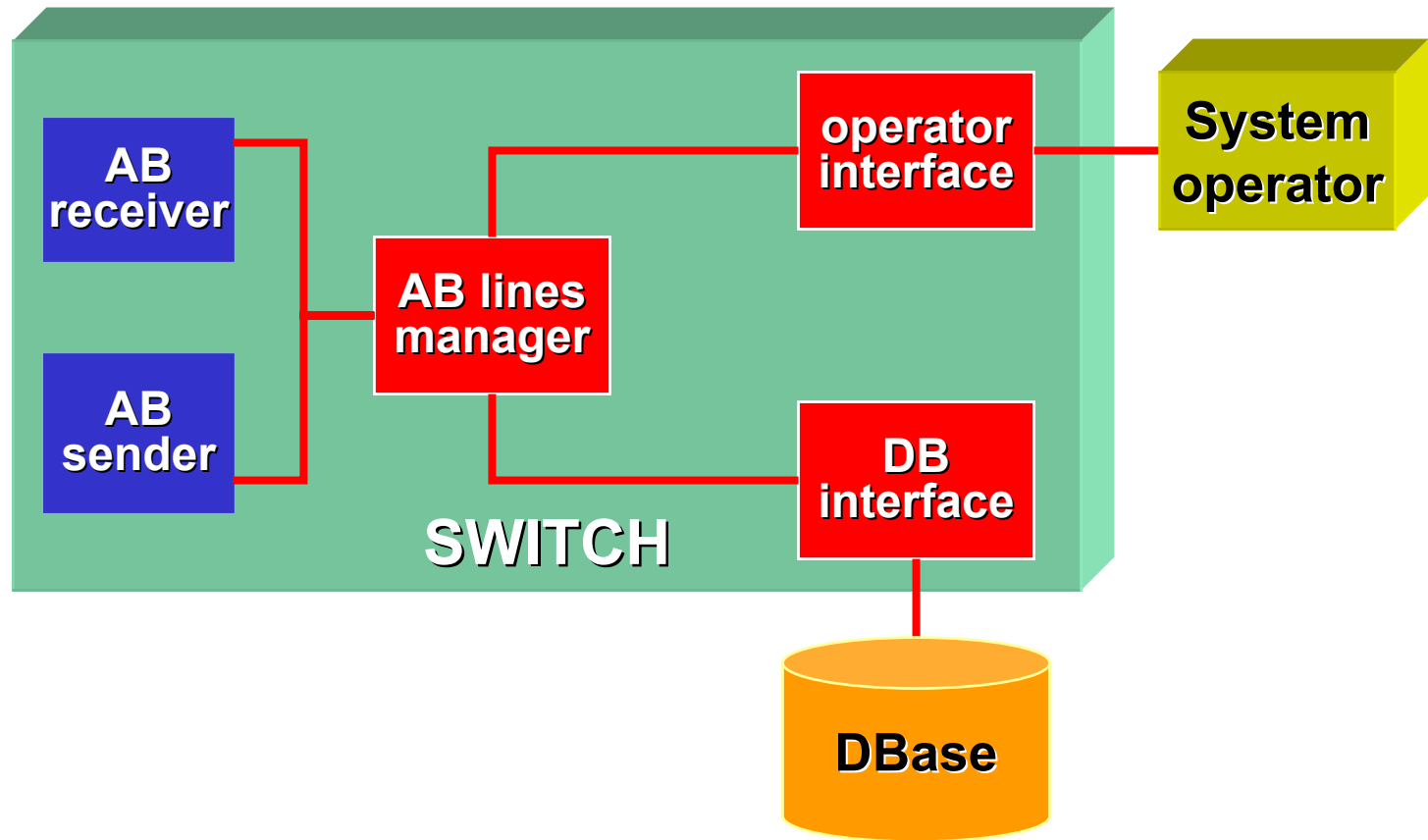


Receiver SM



# Additional Considerations

- Support (control) infrastructure

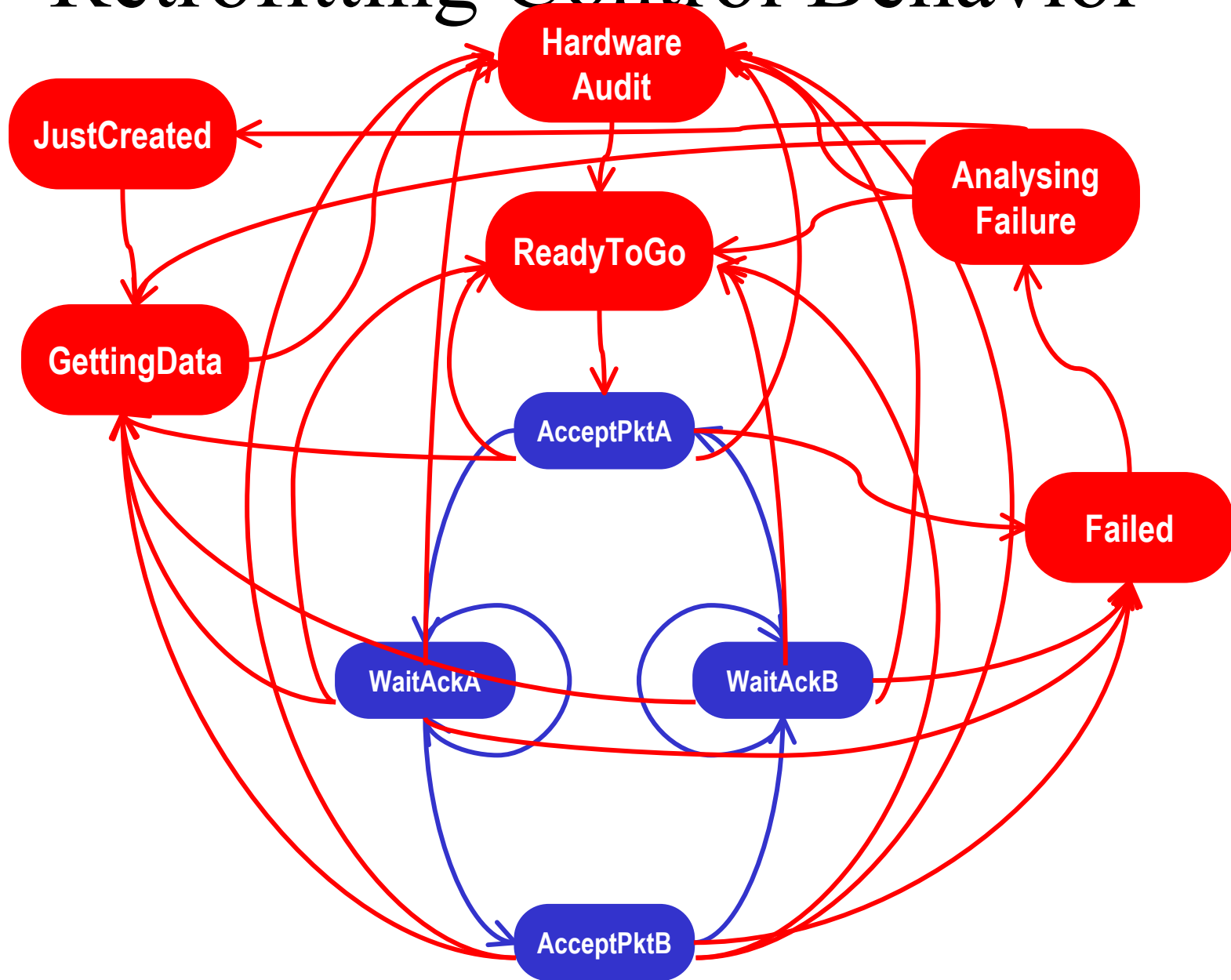


# Control

*The set of (additional) mechanisms and actions required to bring a system into the desired operational state and to maintain it in that state in the face of various planned and unplanned disruptions*

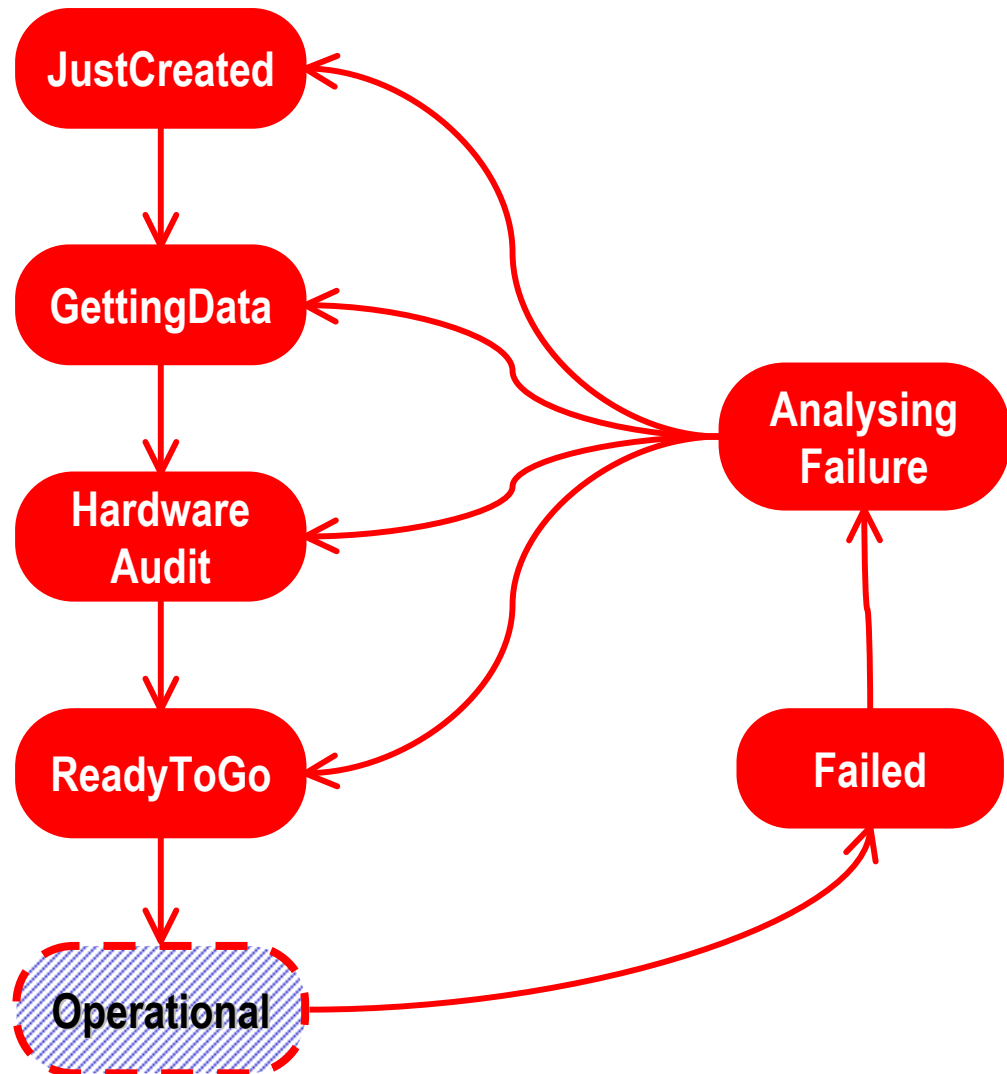
- For software systems this includes:
  - system/component start-up and shut-down
  - failure detection/reporting/recovery
  - system administration, maintenance, and provisioning
  - (on-line) software upgrade

# Retrofitting Control Behavior



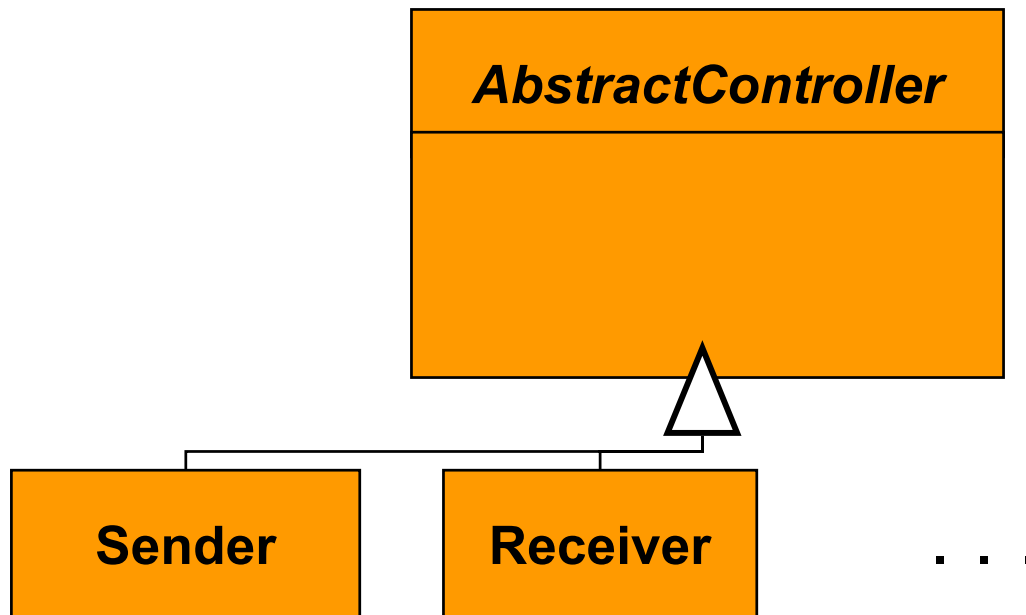
# The Control Automaton

- In isolation, the same control behavior appears much simpler



# Exploiting Inheritance

- Abstract control classes can capture the common control behavior



# Exploiting Hierarchical States

