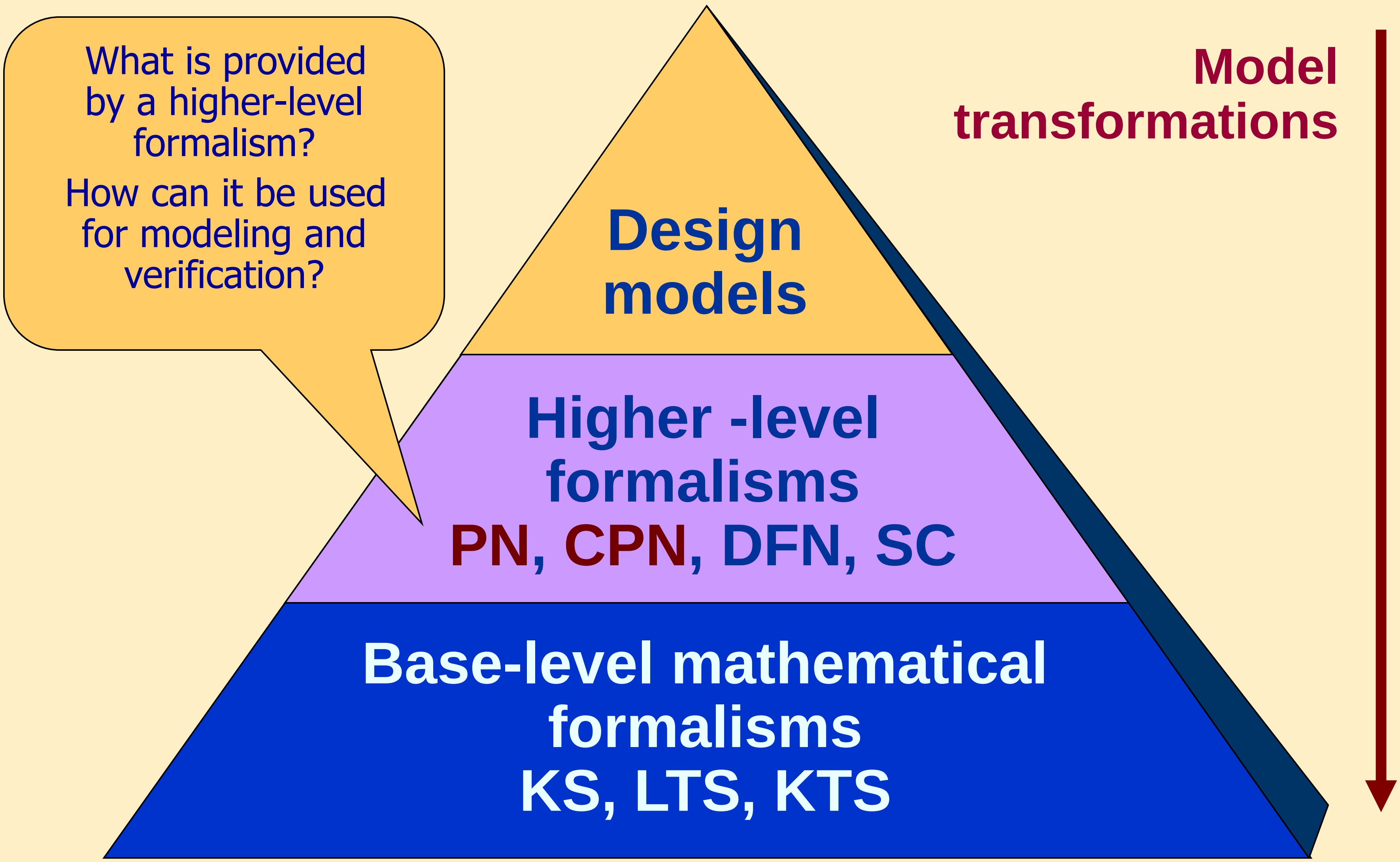
# Higher-level formalisms: Statecharts

dr. István Majzik

BME Department of Measurement and Information Systems

# Formal models for verification



# Outline

- Basic elements of statecharts
- Syntax of statecharts
  - UML 2 statechart diagram (state machine)
- Semantics of statecharts
  - UML 2 State Machine semantics
  - (Other semantics possible: e.g. Harel semantics)
- Using statecharts

# What is the goal of statecharts?

- Suitable for modeling the behavior of state-based, event-driven systems
  - Description of the behavior of a state machine (~automaton)
  - Reactive behavior:
    Describes change of state triggered by external events
    - E.g. incoming messages, signals, calls, …
  - Actions: operations assigned to transitions
    - E.g. assignment, outgoing message, …
- Common usage:
  - Embedded systems: processing incoming events (e.g. controlling a robot, security systems, …)
  - Protocols: processing messages

# Terminology

- State, active state
  - Certain conditions hold (e.g. operation can be executed)
  - State variables have certain values
- State transition
  - Change of state
  - Trigger event can make it happen
    - Transitions without trigger: “spontaneous” execution
  - Guard condition can be assigned to transitions
    - Transition can occur only if guard condition is true
  - Actions can be executed when transitions occur
    - Operation or behavior assigned to a transition
- Event
  - Asynchronous occurrence, can have parameters
  - Individual entity, the instance of an event class
    - Inheritance: to extend event parameters

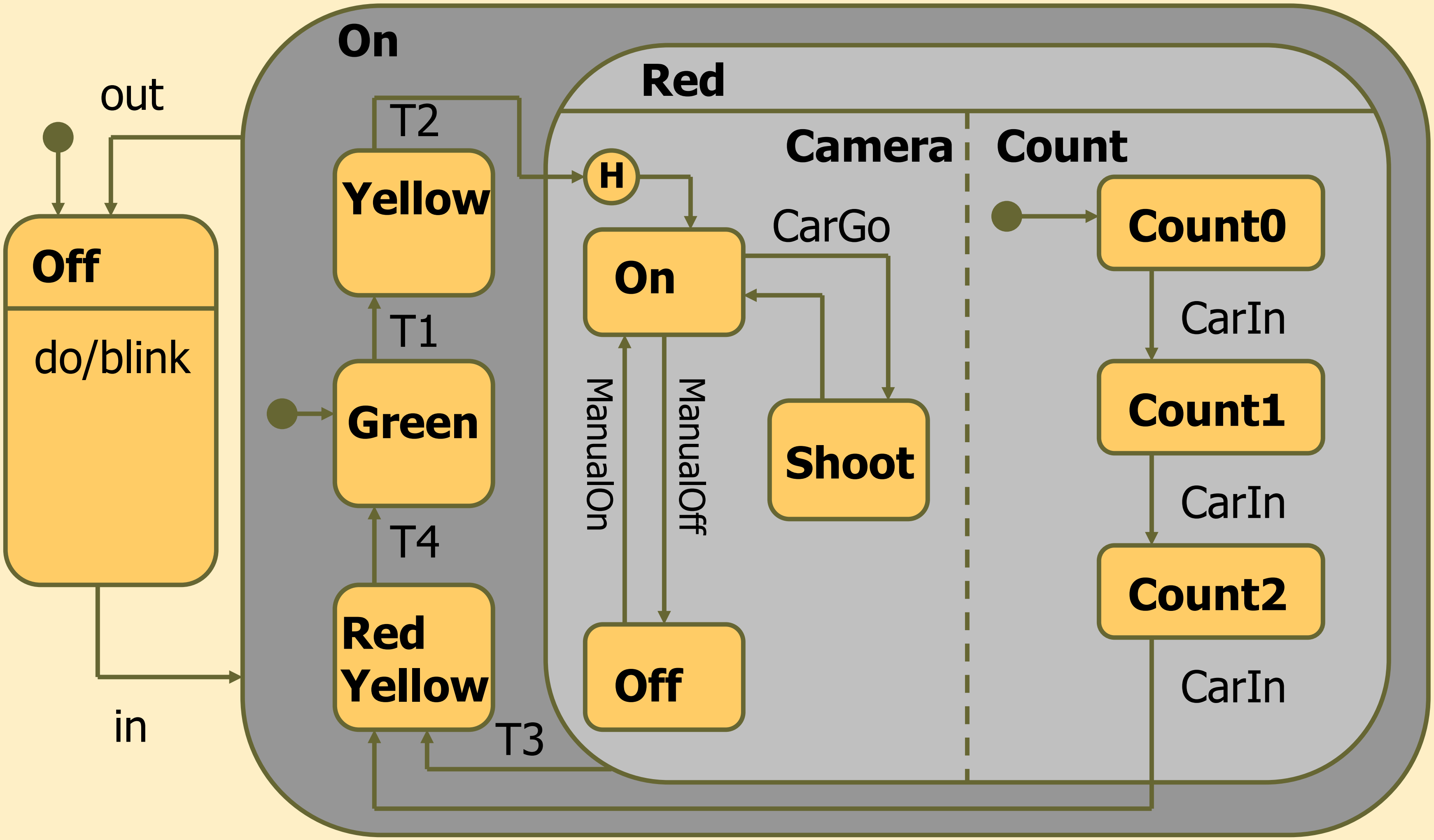
# Additional features for convenient modeling

- Refinement of states: State hierarchy
  - Superstate: for the common properties of substates
- Description of concurrent behavior
  - No ordering should be enforced (processing simultaneously or in an arbitrary order)
  - In case of multi-threaded/distributed/parallel execution
- Complex transitions
  - Fork, join, conditional branch
- Memory: Return to a previous state configuration
  - Return from the processing of an interrupting event
  - In a single level of hierarchy or even deeper

# State machines and statecharts

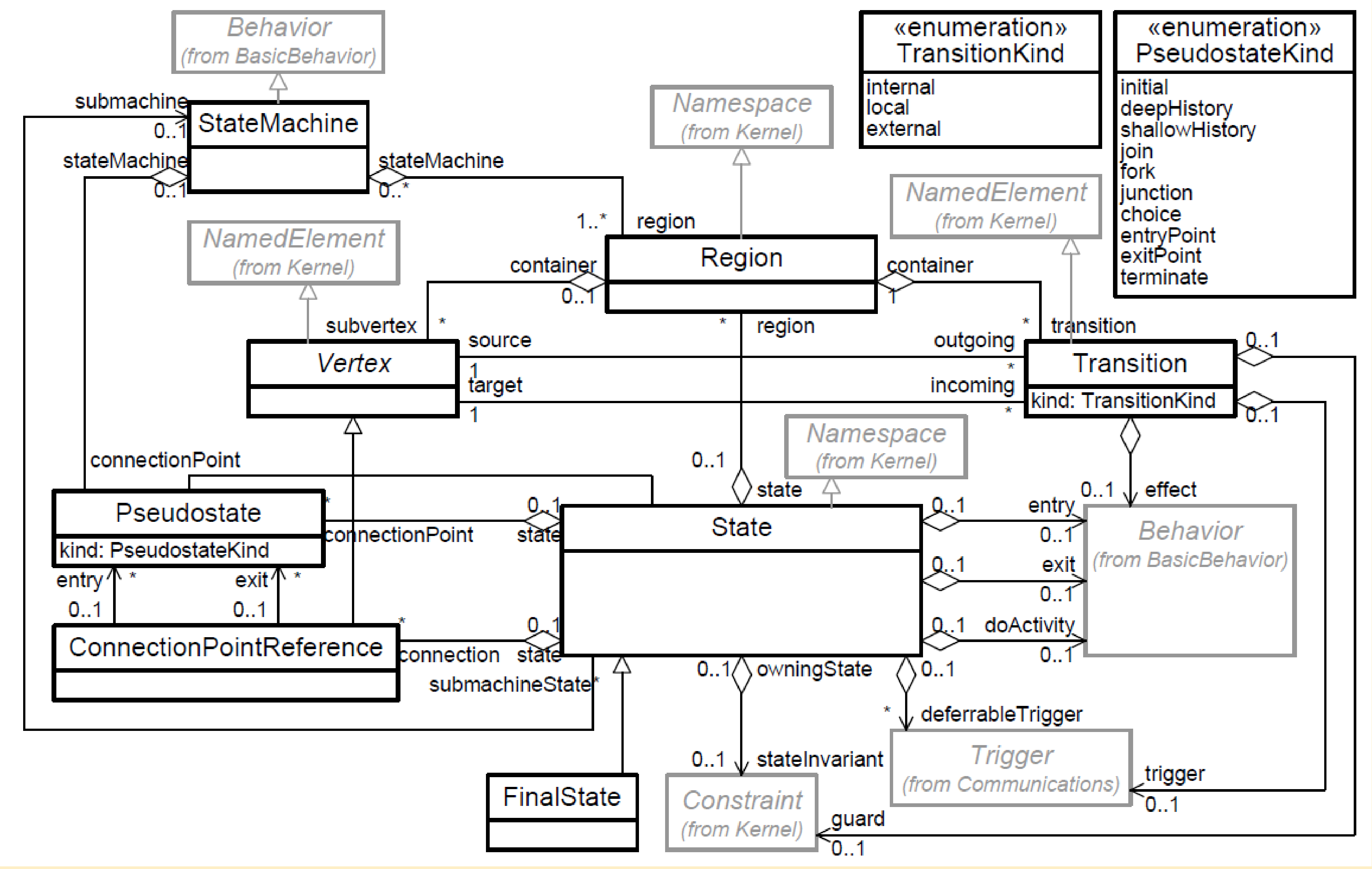
- State machine:
  - Flat, simple states and transitions
    - E.g. UPPAAL automata
- Statechart: extension of state machines
  - State hierarchy: state refinement
  - Concurrent regions: to describe concurrent behavior
  - Complex transitions: fork, join, branch
  - Memory: “Storing” the last active state configuration
  - Some syntactic sugar
  - Rarely used (unintuitive) extensions
    - Delayed event, synchronization state, …

# A statechart

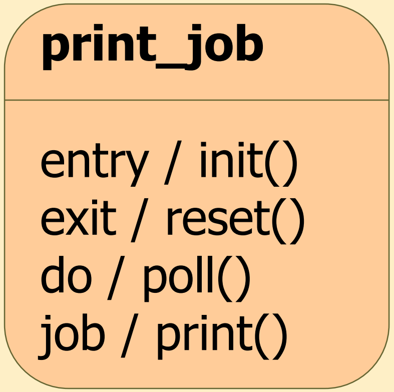


# Syntax of statecharts (conforming to UML)

# UML State Machine metamodel

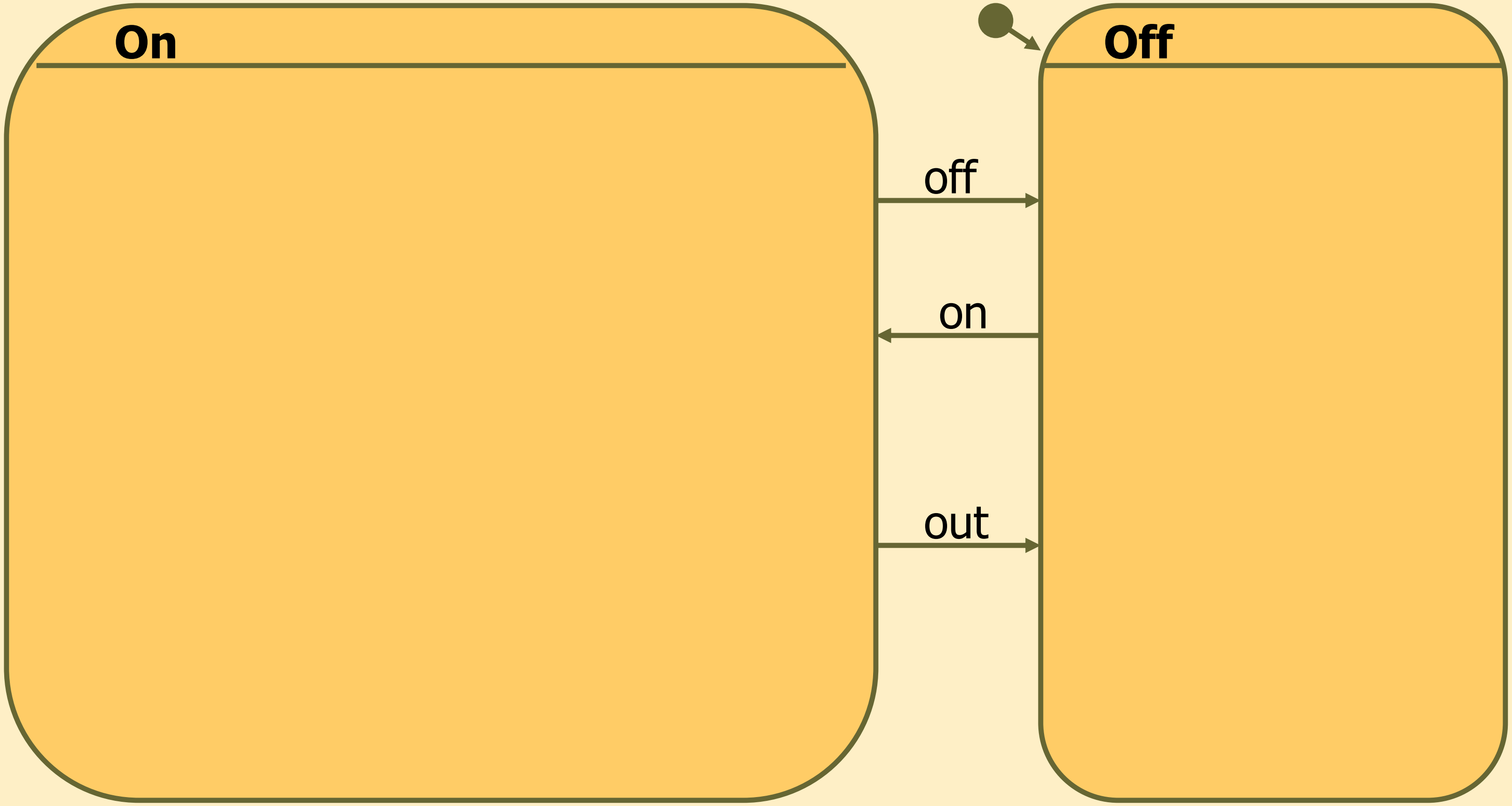


# States: Actions and state refinement

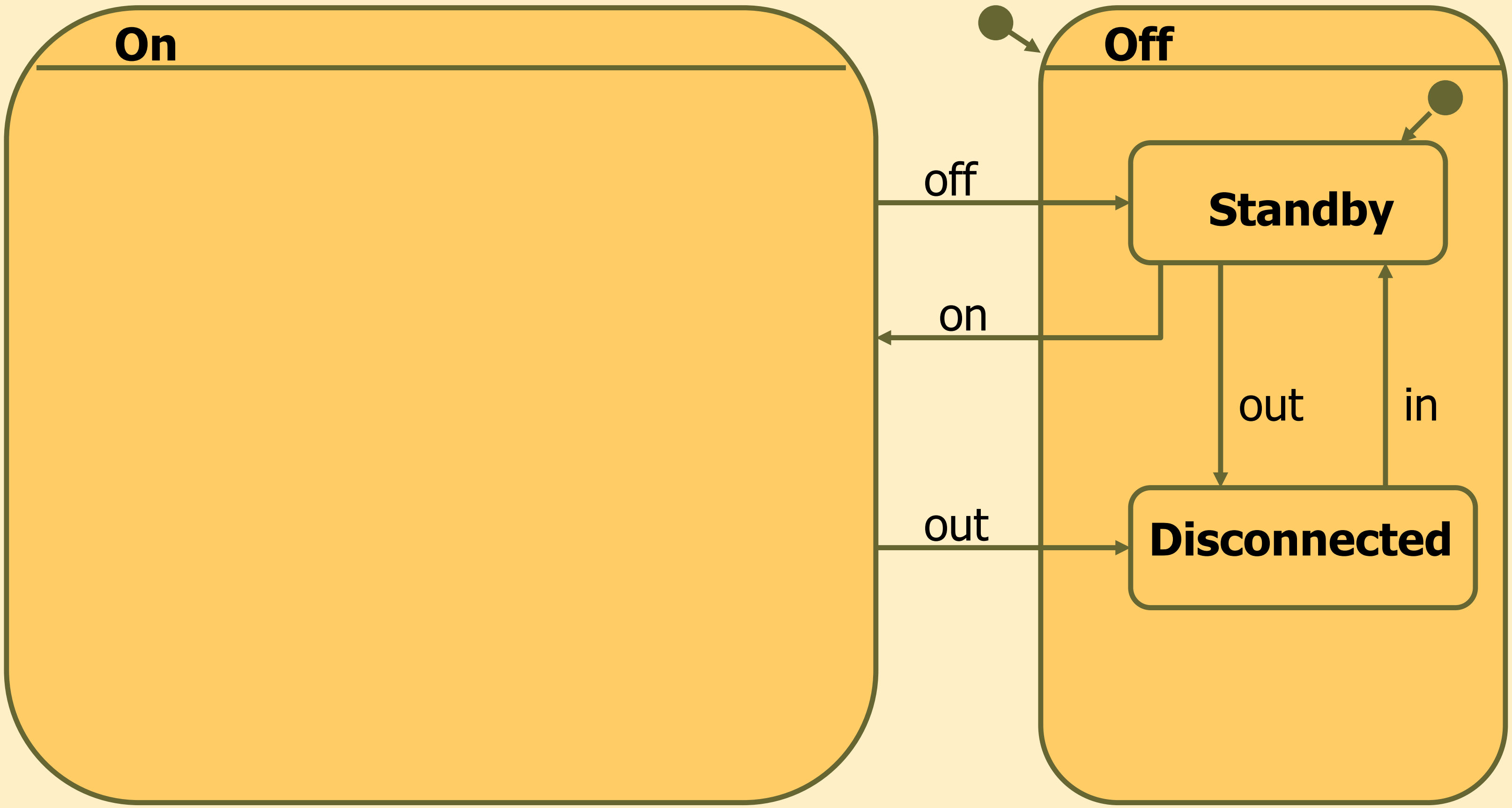



- States: Basic modeling element
- Actions assigned to states:
  - Entry action (entry / ...)
  - Exit action (exit / ...)
  - Internal actions (do / ..., <event> / ...)
- State refinement
  - Simple state: no refinement
  - OR-refinement: substates of a superstate
    - Exactly one substate is active when superstate is active
  - AND-refinement: cocurrent regions
    - One substate in every region is active when superstate is active!

# Example: State refinement

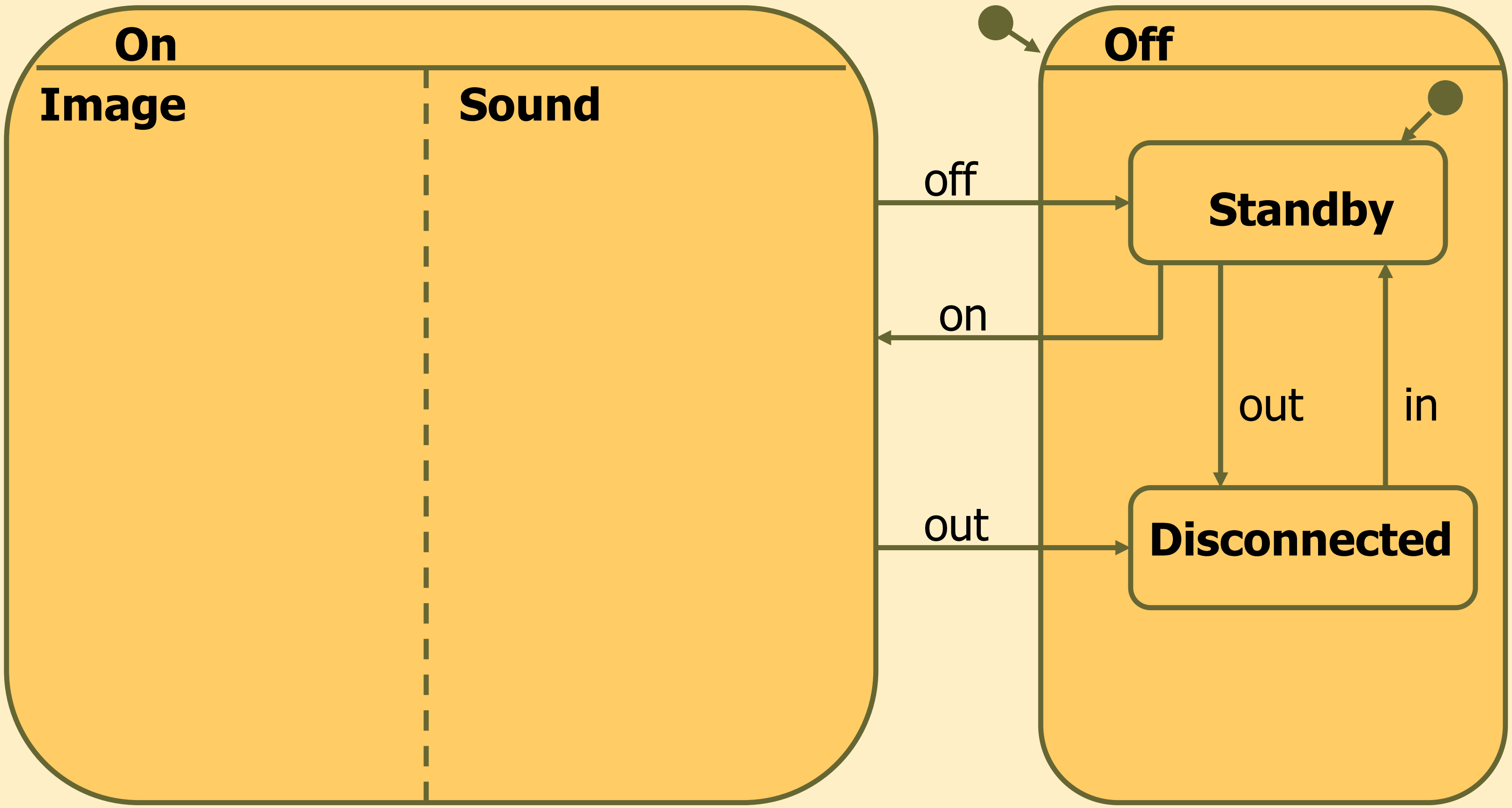


# Example: State refinement




OR-refinement

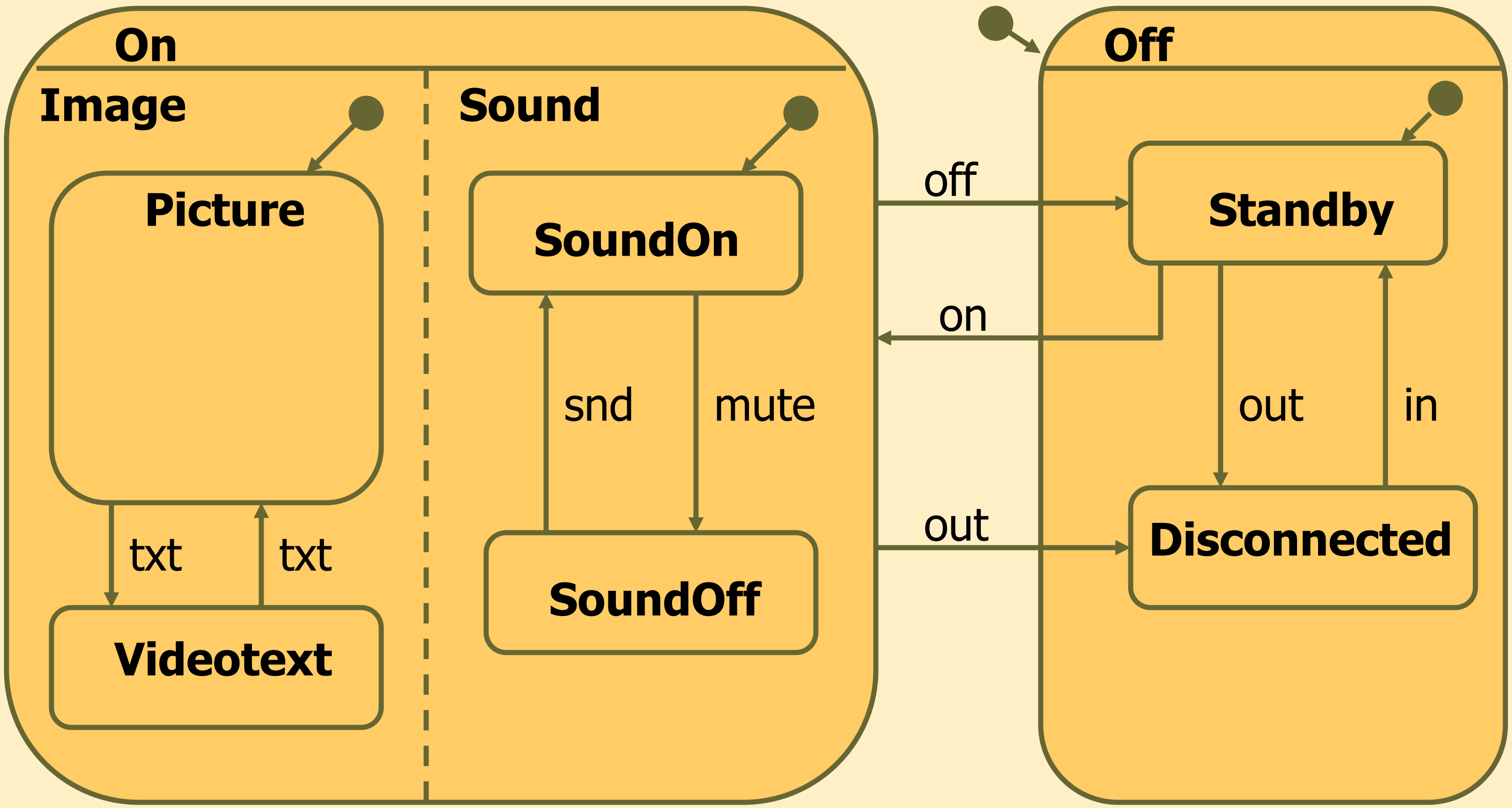
# Example: State refinement




AND-refinement

OR-refinement

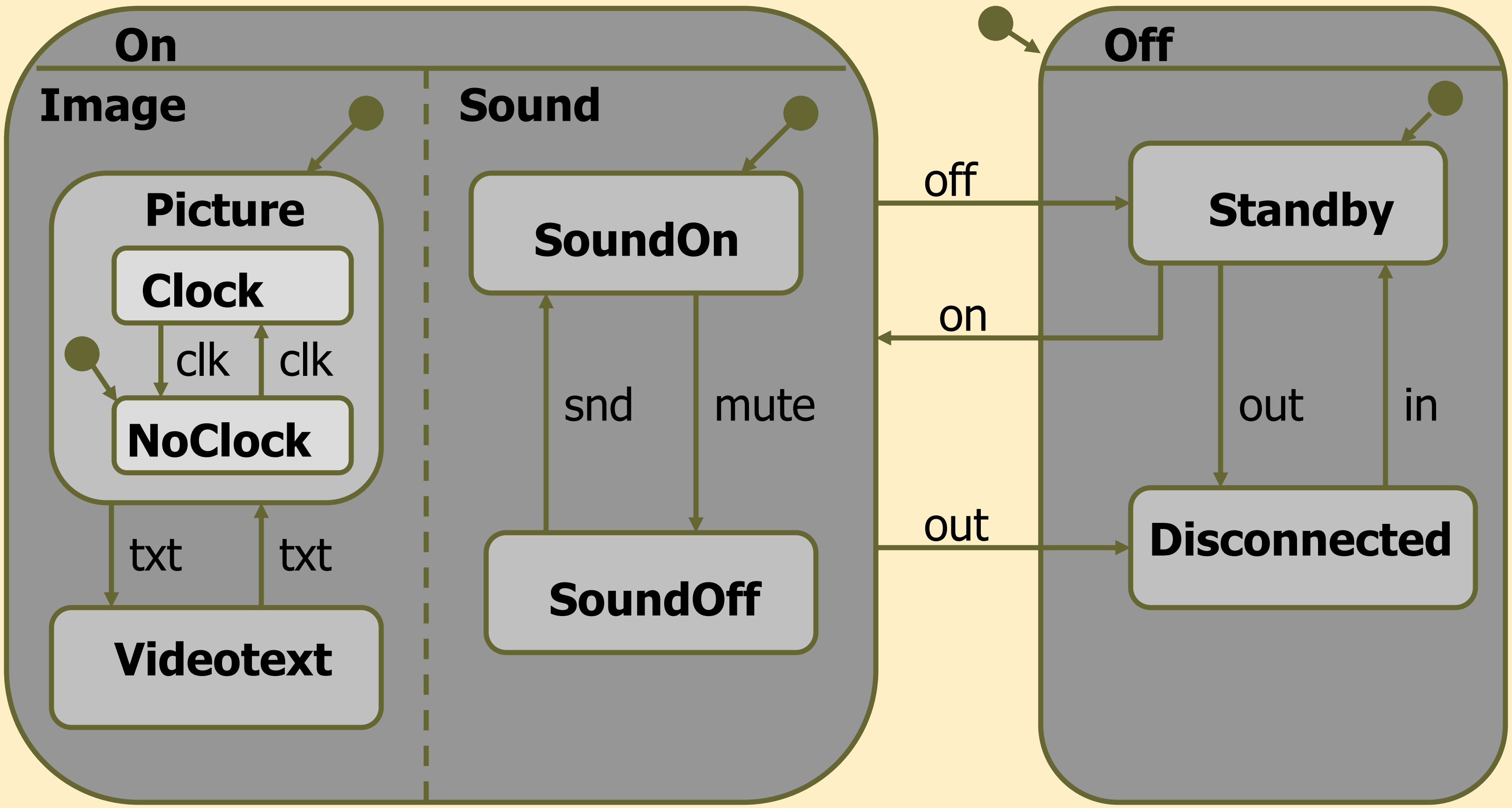
# Example: State refinement




AND+OR-refinement

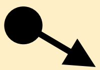
OR-refinement

# Example: State refinement



# Pseudostates

- Initial state: activates when superstate is activated
  - Should be one in every OR-refinement
  - Should be one in every region of an AND-refinement



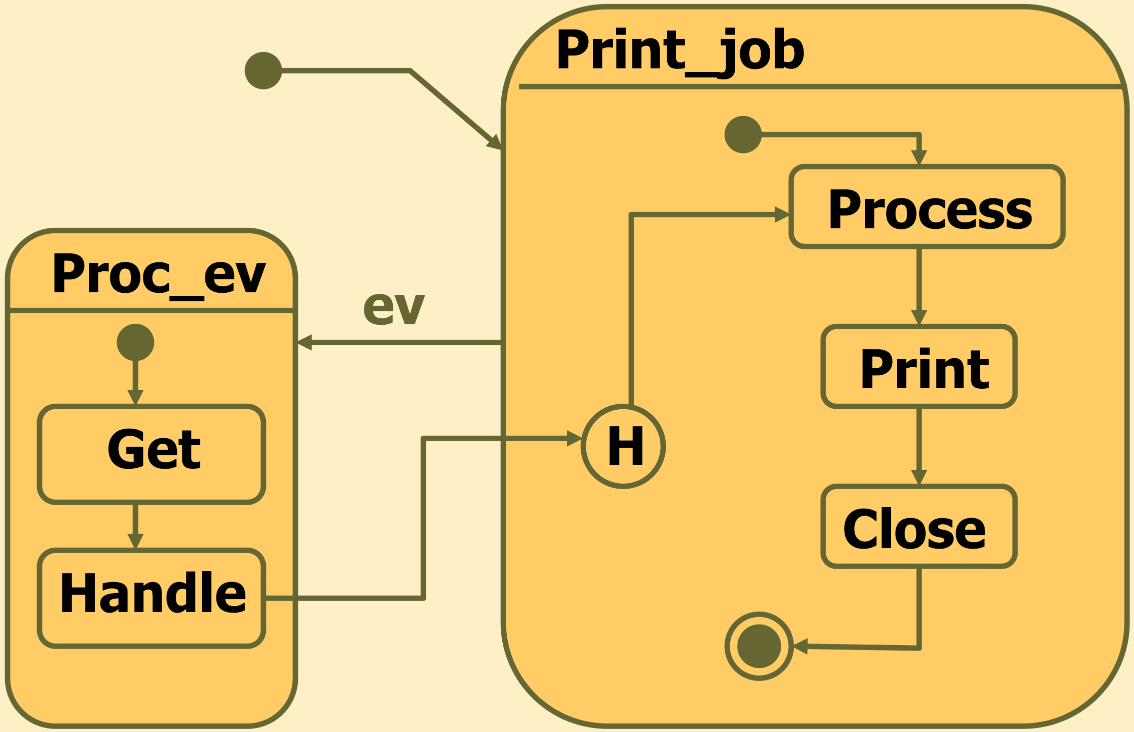
- Final state: statechart terminates

- History states:
  - “Stores” last active state configuration
    - Simple history state: only on given hierarchy level
    - Deep history state: remembers lower levels as well
    - In a region of an AND-refinement: Only for the region
  - What is the meaning of a transition entering the history state?
    - When executed, the state configuration of the region is restored
    - The history state is a “reference” to the stored state configuration
  - What is the meaning of a transition leaving the history state?
    - Selects the default state in case the region has not been activated before

H

H*

# Example: History state



# (State) transitions

- Specification of transitions (in addition to arrows)
- Syntax:

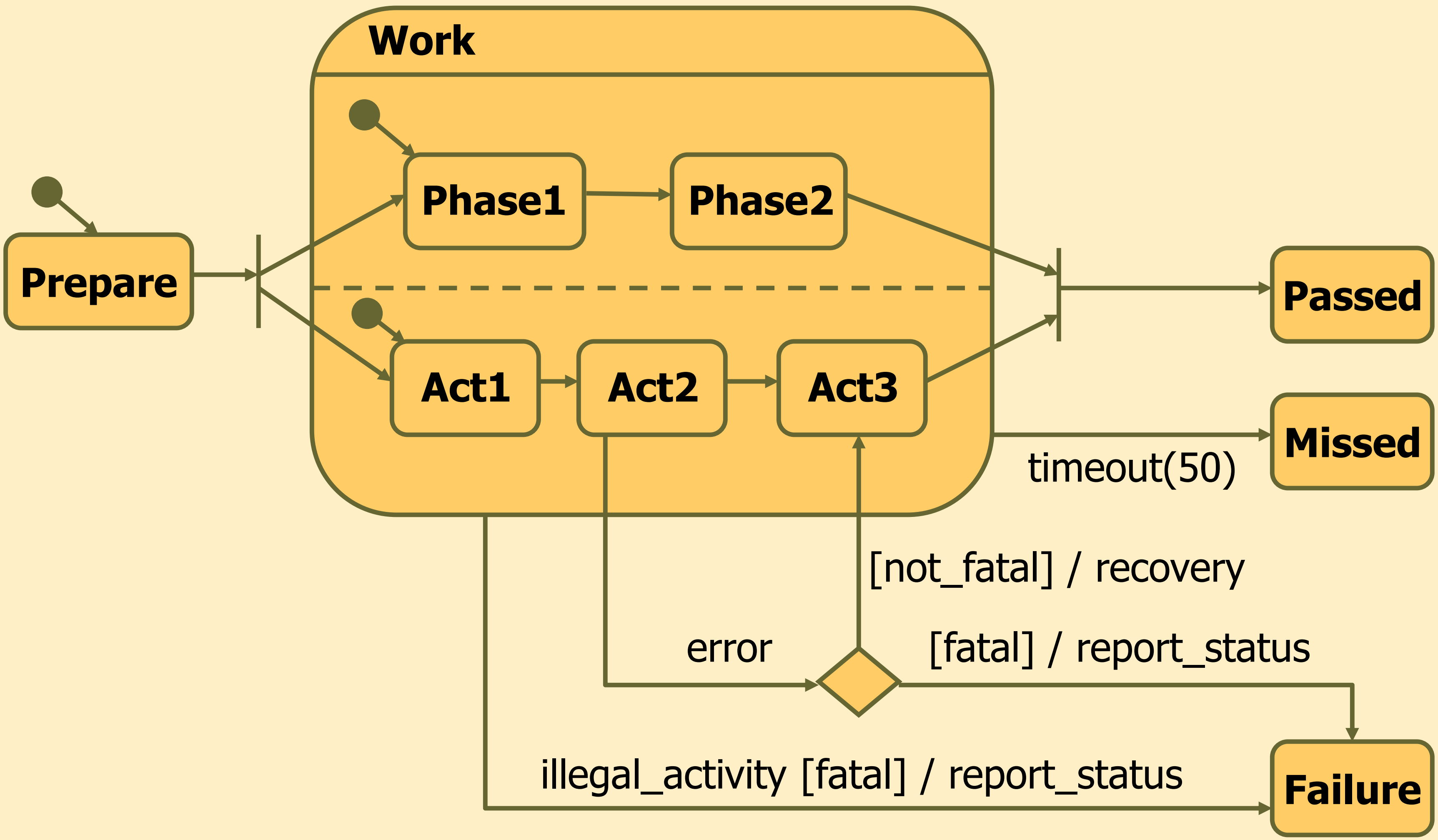
```
trigger [guard] / action
```

  - **`trigger`**: triggering event
  - **`guard`**: guard condition of the state
    - Predicate over state variables and parameters of the event
    - Can also refer to states: is_in(state)
  - **`action`**: operation
    - Action semantics: atomic operation

# Special transitions

- Complex transitions
  - Fork: to enter multiple states, each in a concurrent region
  - Join: leave states in concurrent regions simultaneously
  - Branching (condition): combined notation for multiple transitions differing in guard conditions and actions (segments)
- Transitions crossing hiearchy levels
  - Permitted (although not elegant)
- Time-out as a trigger
  - Occurs when the source state has been active for the specified time

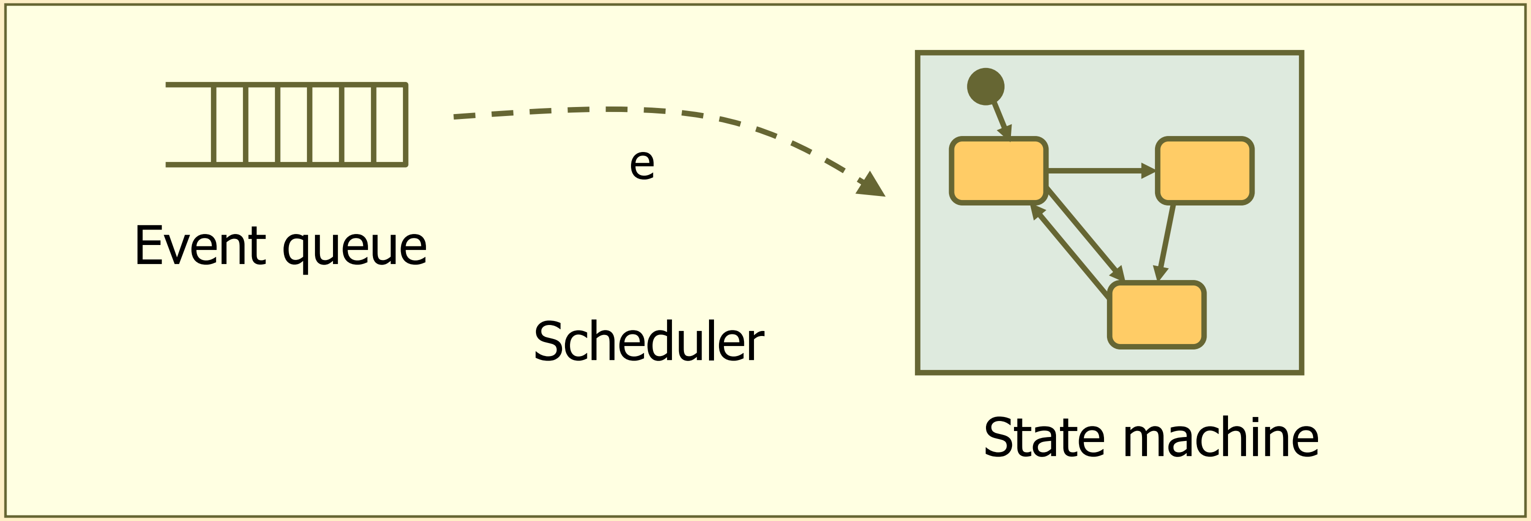
# Example: State transitions



# Formal semantics of statecharts (conforming to UML)

# Semantics: How does it work?

- Basic elements:
  - State machine: The statechart describes its behavior
  - Event queue + Scheduler: „runtime environment” (external elements)



# What is specified by the semantics?

Behavior of the state machine when processing an event $\rightarrow$ a step of the state machine

- Transitions "fire"
  - What's new: a single event may trigger multiple concurrent transitions (in active regions)
- Change of state configuration
  - There may be multiple active states
    - One active substate in every region of an active superstate
    - One active substate in an OR-refined superstate
  - Still, at most one active state in an OR-refinement or region
  - Applied recursively

# Basic properties of the semantics

- Events are processed one by one
  - The scheduler passes the new event only if the previous event has been completely processed
    - Stable state configuration: no enabled spontaneous transitions
- Complete processing of events (run to completion)
  - Maximal set of transitions fire
    - Every enabled transition will fire unless prevented by a conflict
  - After firing all of these, the next event is passed
- The main point of the semantics is event processing
  - Based on this, the statechart can be implemented by software (source code generation)

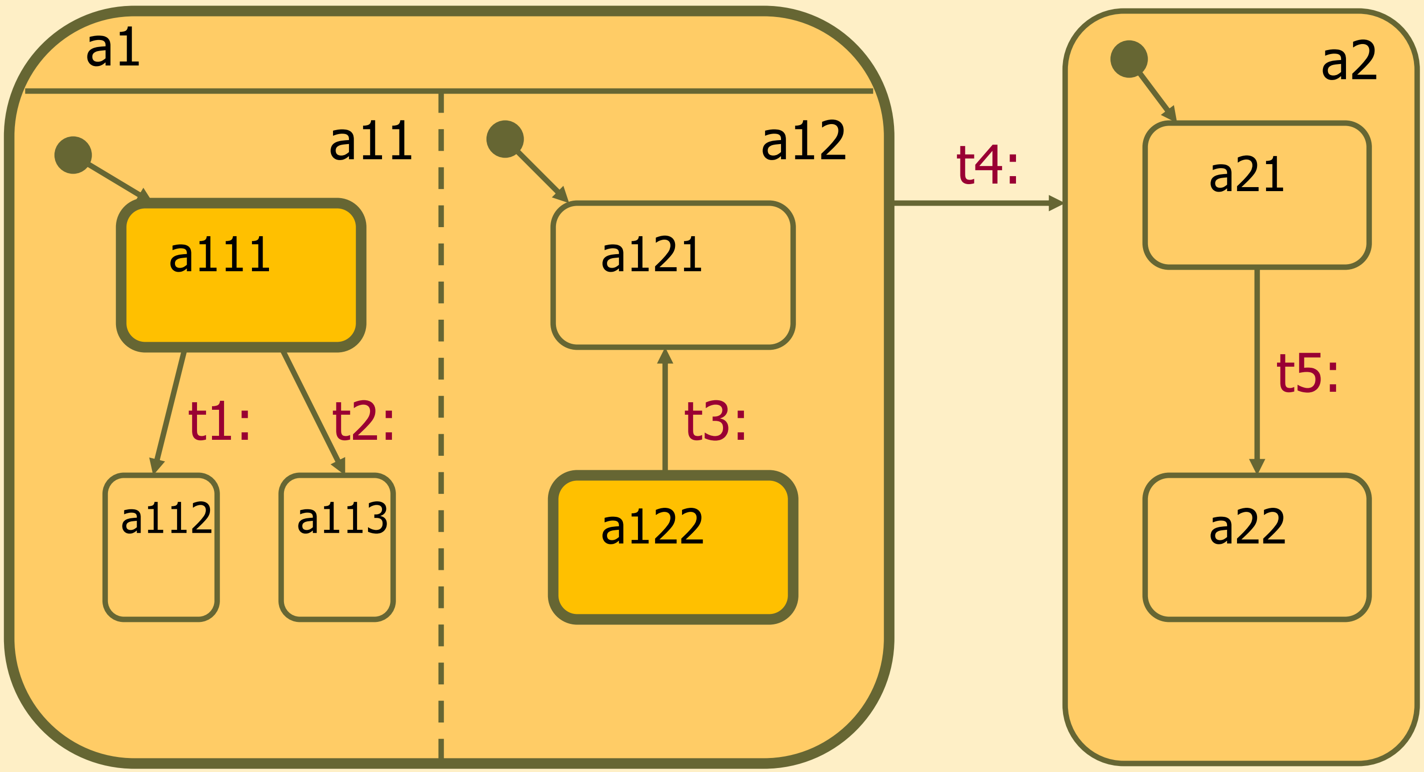
# Steps of event processing 1/4

- External condition: The scheduler passes an event to the stable state machine
- Enabled transitions:
  - Source state is active
  - Selected event triggers transition
  - Guard condition is true

Based on the number of enabled transitions:
  - If only one: Fire!
  - If none: Is the event delayed?
    - Yes: store it, ask a new event
    - No: event may be discarded (without any actions)
  - If multiple: Need to select transitions to fire
    - Based on: conflicts

# Example: Conflict

In this example, transitions t1, …, t5 are triggered by the same event e.
Active states are denoted by thicker borders.



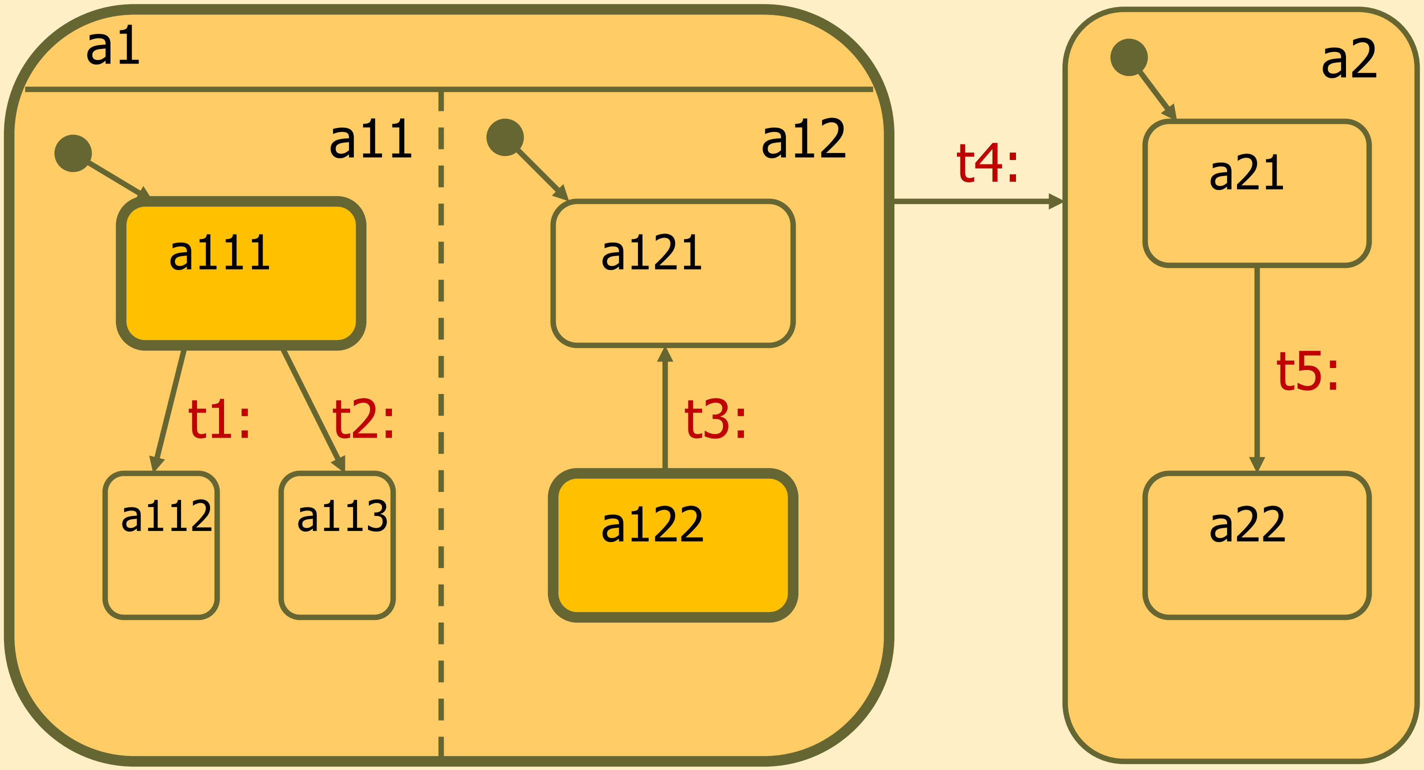

- Not enabled: t5 (source state inactive)
- Cannot fire together: (t1,t2);   (t4,t1); (t4,t2); (t4,t3)
- May fire together: (t1,t3); (t2,t3);

# Steps of event processing 2/4

- Fireable transitions are selected:
  - Maximal number of transitions without conflict
    - Simultaneous firing of concurrent transitions
- Conflict between transitions:
  - They leave the same state, that is, the intersection of the sets of states inactivated is non-empty
- Resolving conflicts:
  - Based on priority: the priority of a transition is higher if its source state is lower in the refinement hierarchy
    - OO concept: refinement “overrides”
  - Nondeterministic choice in case of same priority

# Example: Conflict resolution

Transitions t1, …, t5 are triggered by the same event e.
Which may fire together?

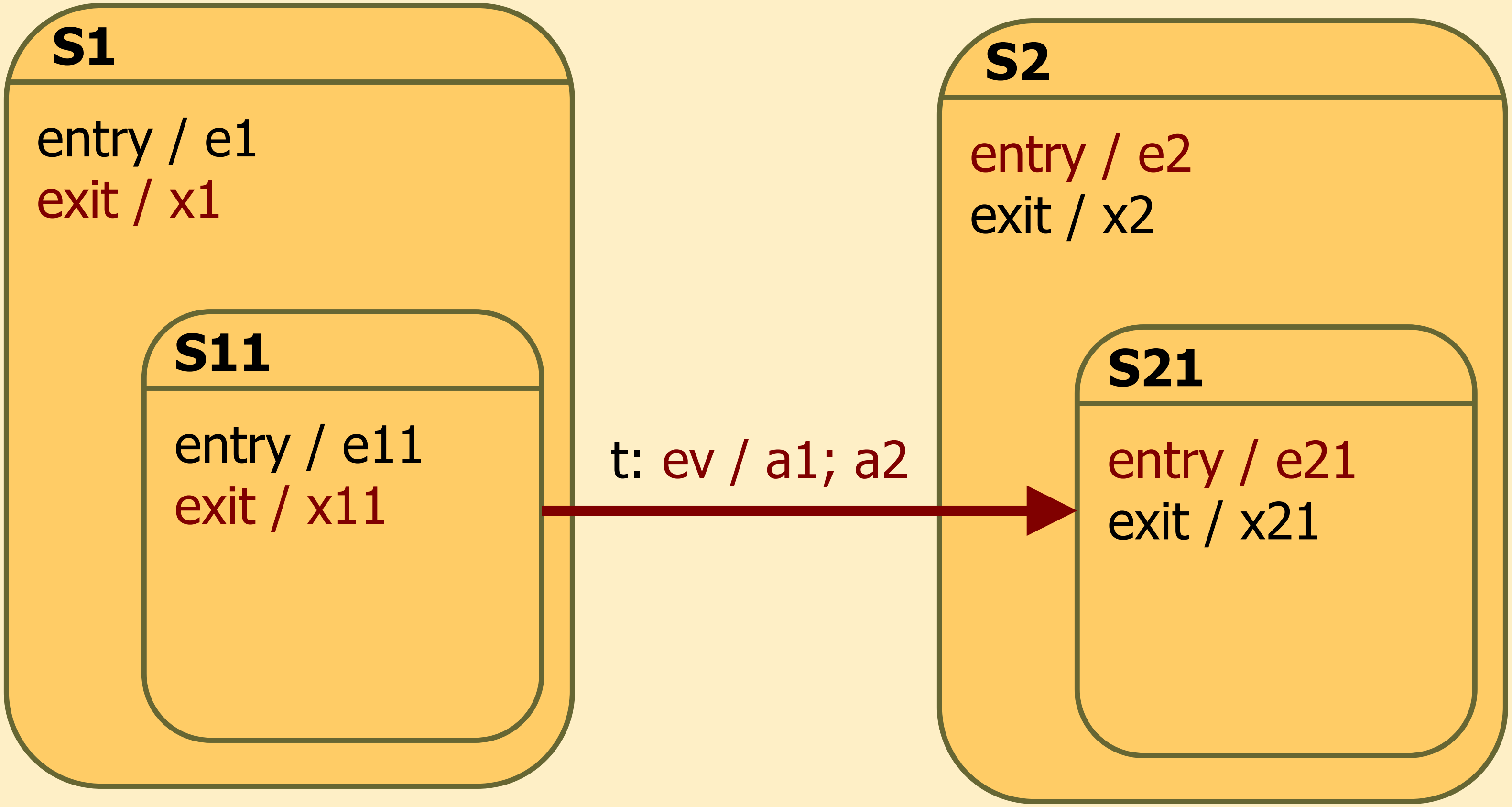



- Larger priority than t4: t1, t2 and t3
- Cannot fire together: (t1,t2);   (t4,t1); (t4,t2); (t4,t3)
- Fireable: (t1,t3) or (t2,t3)

# Steps of event processing 3/4

- Selected transitions fire:
  - In a nondeterministic order (no conflict)
  - Therefore the order of actions is also nondeterministic
- Firing a single transition:
  1. Source states are exited
     - On lower hierarchies first
     - Execution of exit actions
  2. Actions of transitions are executed
  3. Target states are entered $\rightarrow$ new configuration
     - On higher hierarchies first
     - Execution of entry actions

# Example: Ordering of actions

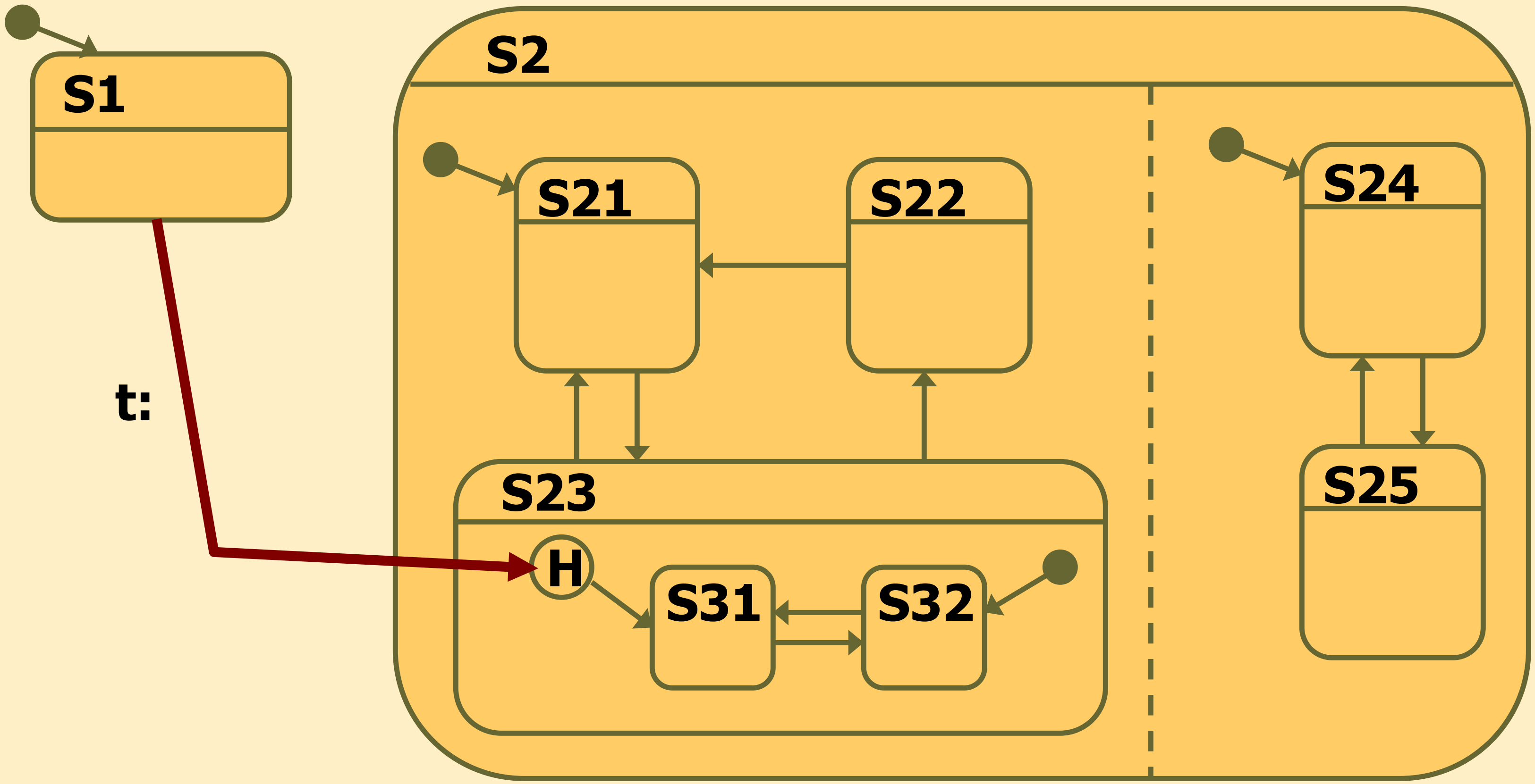



Order of actions: x11; x1; a1; a2; e2; e21

# Steps of event processing 4/4

- Entering new configurations in the case of different target states:
  - If target state is simple (not refined):
    - Will be part of the new configuration
    - Its superstates (in which it is a substate) also activate
    - Activated superstates will activate a substate in each of their regions (determined by initial state)
  - If target state has OR-refinement:
    - A substate is activated as an initial state
  - If target state has AND-refinement:
    - A substate is activated in every region as an initial state
  - If history state:
    - The most recent state configuration is reactivated
    - If this is the first activation: default state
  - If state is not stable: proceed immediately

# Example: Entering a concurrent state

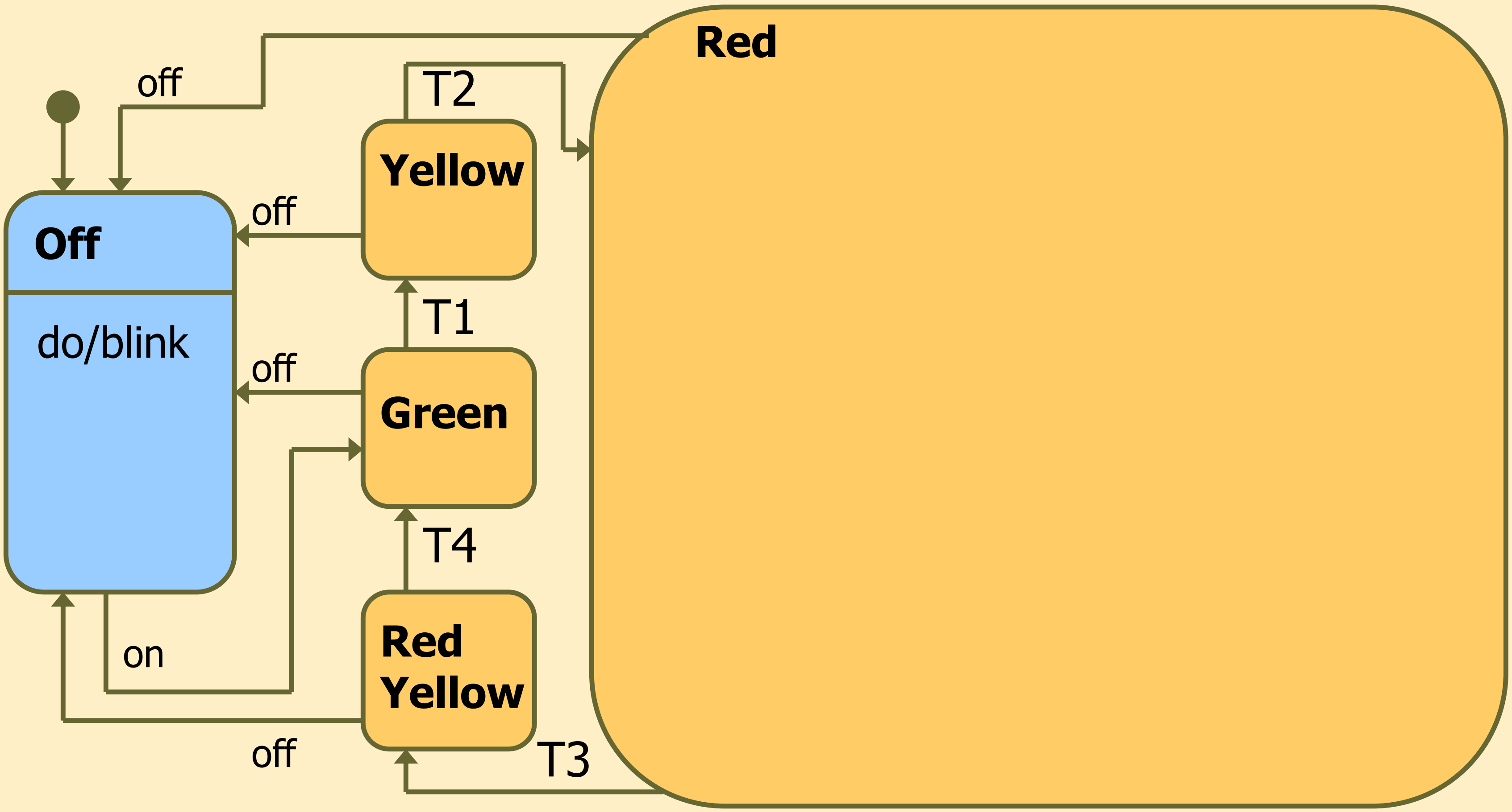



What will be the new state configuration after firing transition **t**?

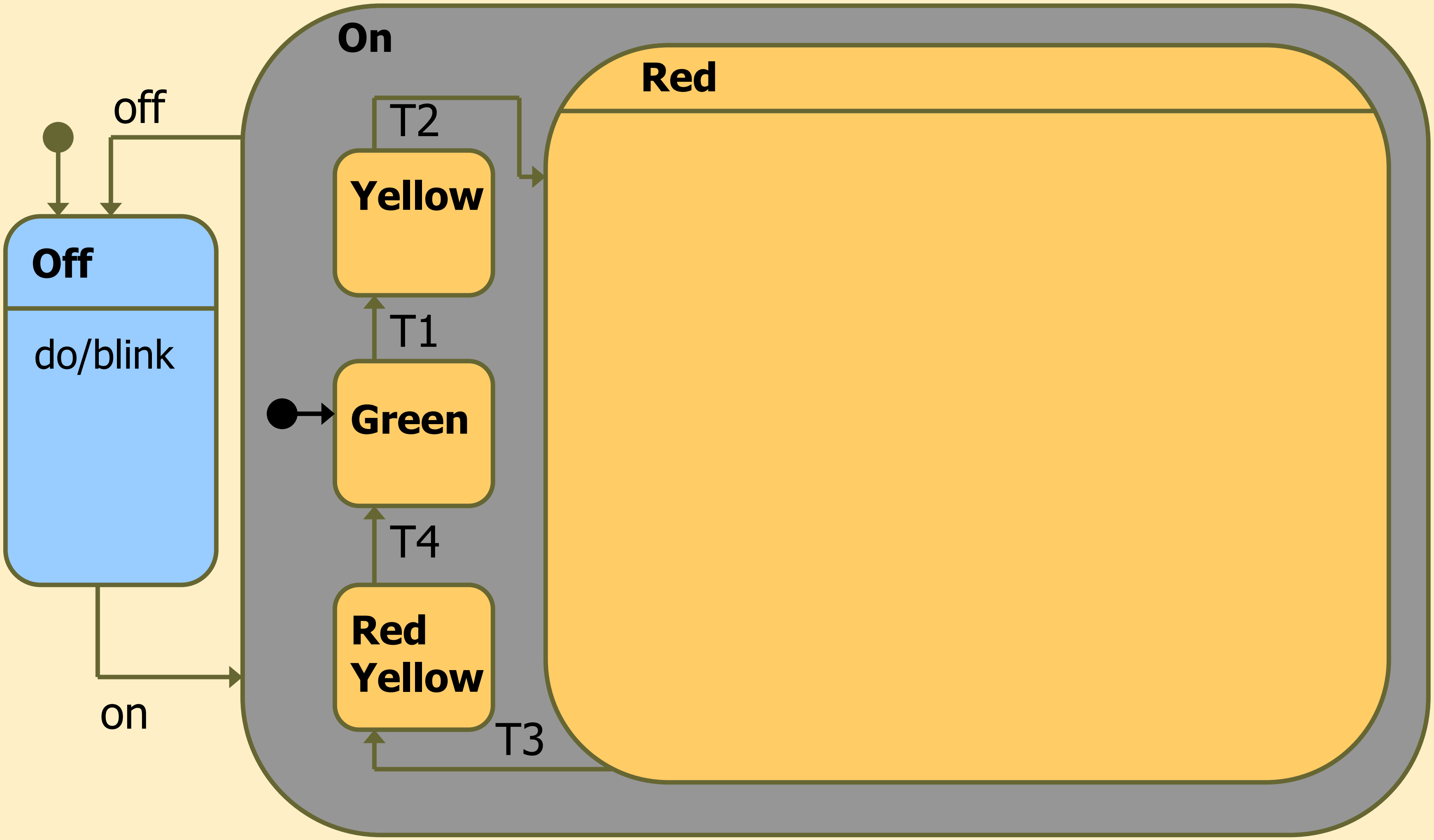
# Modeling example

- Traffic light controller in the intersection with a prioritized road
  - Off: blinking yellow
  - When turned on: green for prioritized road
  - Green, yellow, red cycle: with timer events
  - If at least 3 cars waiting on prioritized road: switch to green regardless of timers
  - Automatically take photos of vehicles crossing the priority road on red light
    - Manual on/off for this feature

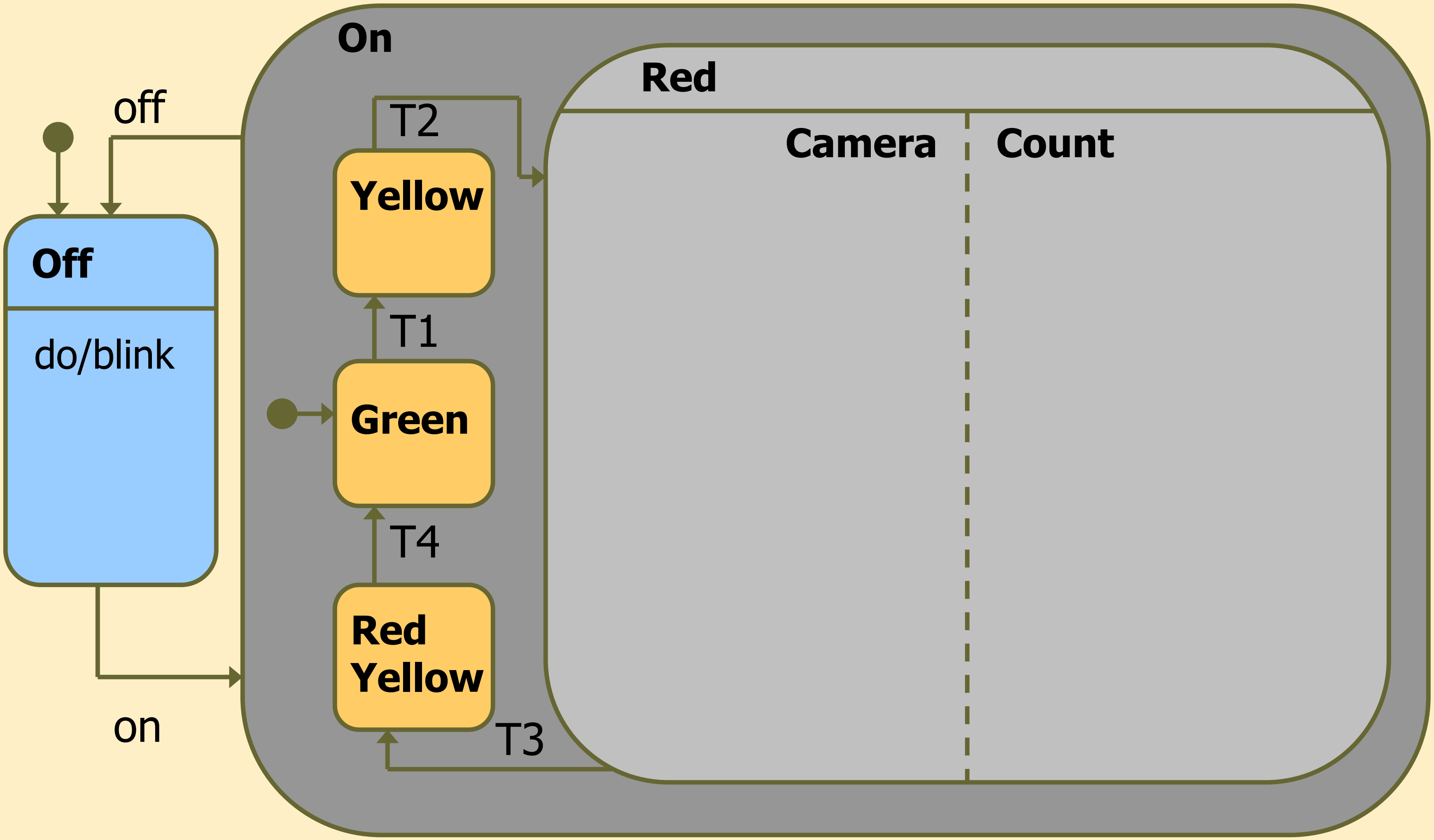
# 1. Main cycle (for prioritized road)



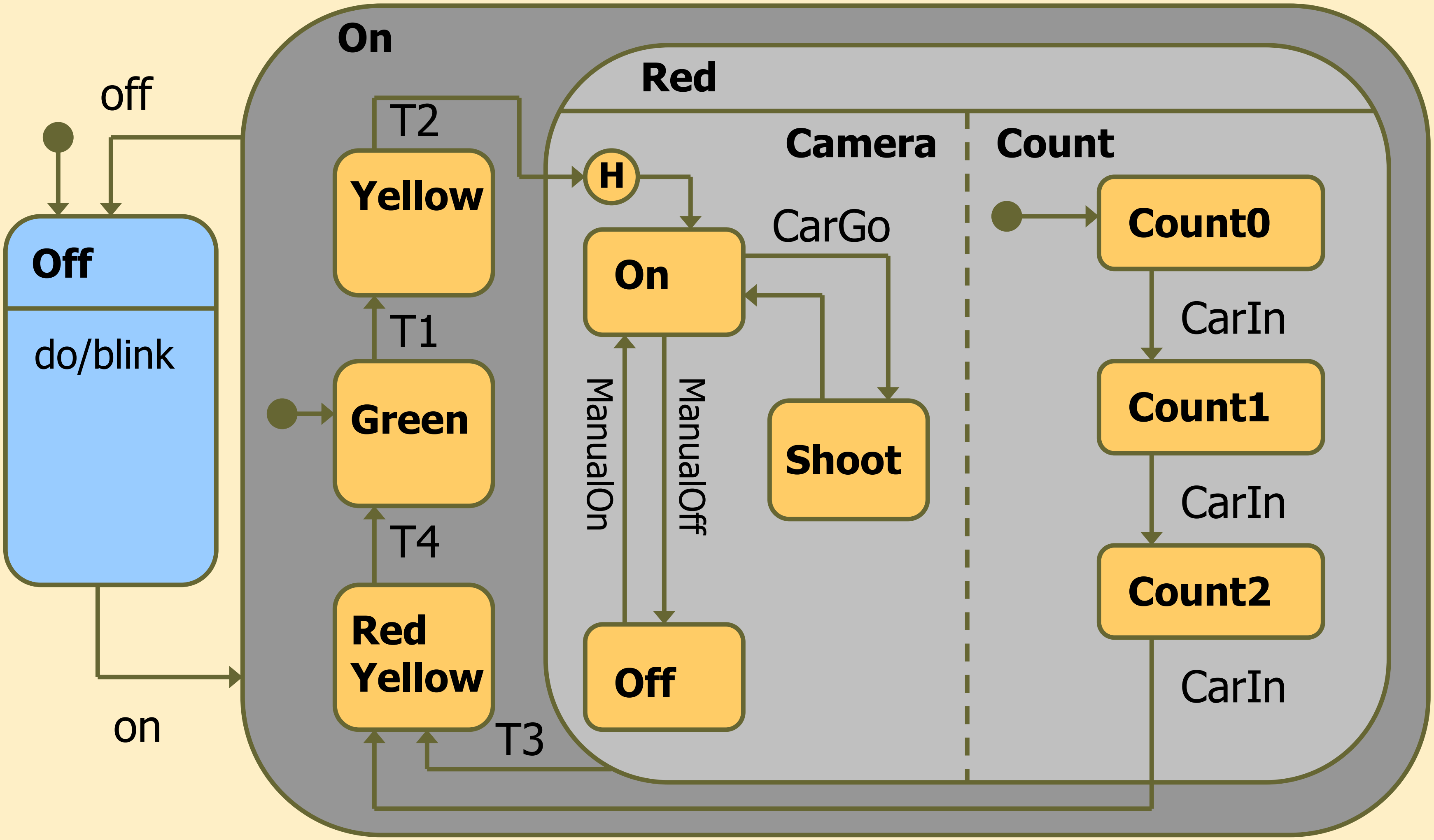
# 2. With hierarchy



# 3. Concurrent regions



# 4. Complete controller



# Role of statecharts in UML 2

- Description of state-based, event-driven behavior
  - To model the behavior of an active object
- To formalize actions: UML 2 Action Semantics
  - Method call
  - Read/write attributes
  - … (many possible operations)
  - Ideas similar to Colored Petri nets (see later)
- To describe actions: There are other alternatives (e.g. Alf)

# Basics of statecharts (summary)

- Extensions
- Statechart syntax
  - State hierarchy, concurrent regions, history states
  - Complex transitions
- Statechart (informal) semantics
  - Enabledness of transitions
  - Selection of fireable transitions
  - Firing transitions
  - Forming a new state configuration
- Statechart tools
  - UML 2 toolsets
  - Yakindu Statechart Tools (statecharts.org)
  - Quantum Programming (state-machine.com)

# What can we do with statecharts?

- Generate source code
  - Multiple templates
- Model checking
  - PLTL algorithm can be “customized” for statecharts
  - May verify by transforming into low-level model
- Generate tests
  - Can be realized with a model checker
- Runtime verification: Generate monitor code
  - Statechart as a reference
    (specify valid behavior to compare to implementation)