Behavioral Modeling Languages

Ákos Horváth and Dániel Varró
With Contributions from István Majzik,
Gergely Pintér, András Vörös,
Gábor Bergmann, Ábel Hegedüs

Model Driven Software Development

Lecture 5

Budapest University of Technology and Economics
Department of Measurement and Information Systems

An Overview of
Behavioral Modeling Languages

Dynamic Languages: An Overview

System

- State-based reactive
- Dataflow-based
- Event & Rule-based
- Agent-based
- Block diagrams
- Other

Property

- Requirements
- Scenarios

Analysis techniques:

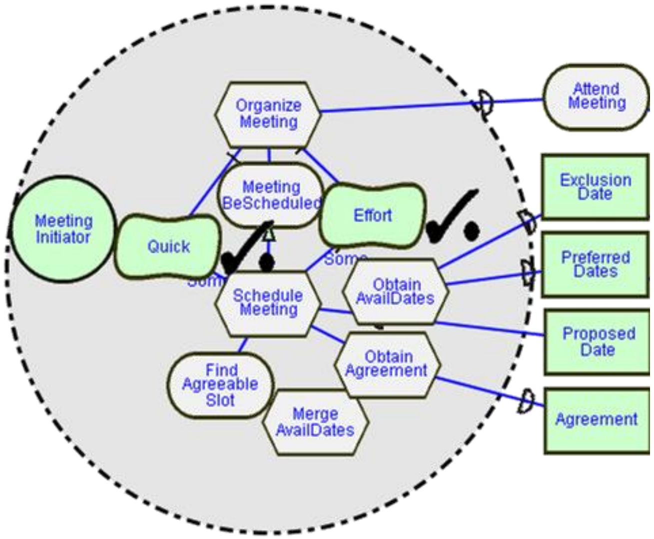
- Simulation, Static analysis, Model checking,
- Symbolic computation, ODE (Diff. Eq)

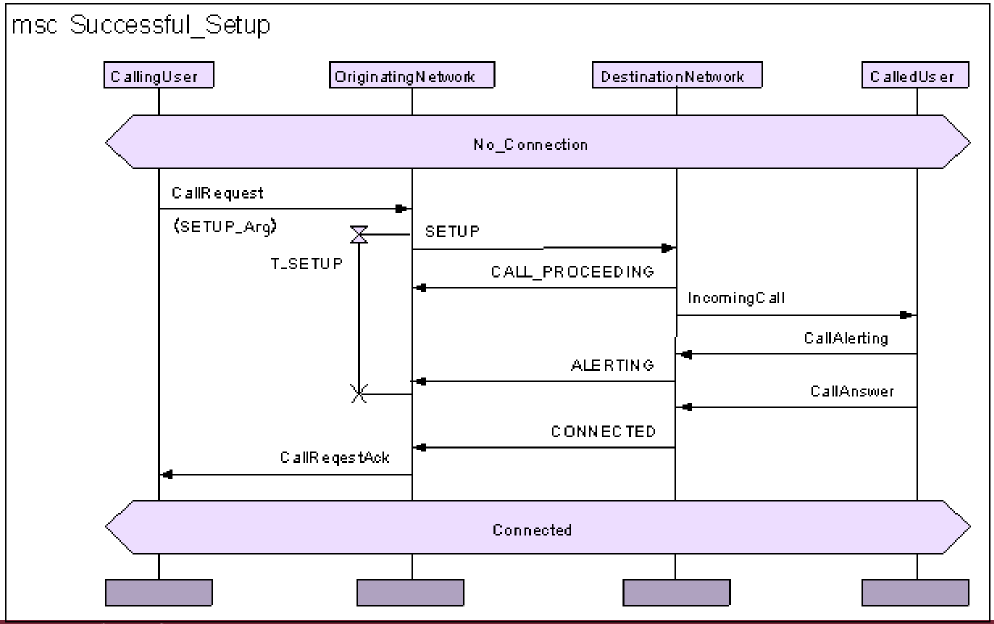
- Engineering languages:
  - Statecharts, Statemate, Business Process Models, Simulink Block Diagram, Message Sequence Charts, KAOS, Drools, CQL, Esterel, AnyLogic, Modelica, Ptolemy-II, …
- Formalisms:
  - Petri nets, Finite automata, Timed automata, Cellular autom. Bond graph, Process algebra, Queuing network, Kahn process network

Characteristics of Dynamic Languages

- Specification
  - Consistency
  - Completeness
  - Unambiguity
- Time
  - Untimed
  - Discrete
  - Continuous
- Communication
  - Synchronous
  - Asynchronous
- Determinism
  - Stochastic
  - Deterministic
- Causality
  - Causal
  - Non-causal
- Analysis
  - Exact vs. Approximative
  - Complete vs. Incomplete
- Other concepts
  - Conflict, priority
  - Dependency,

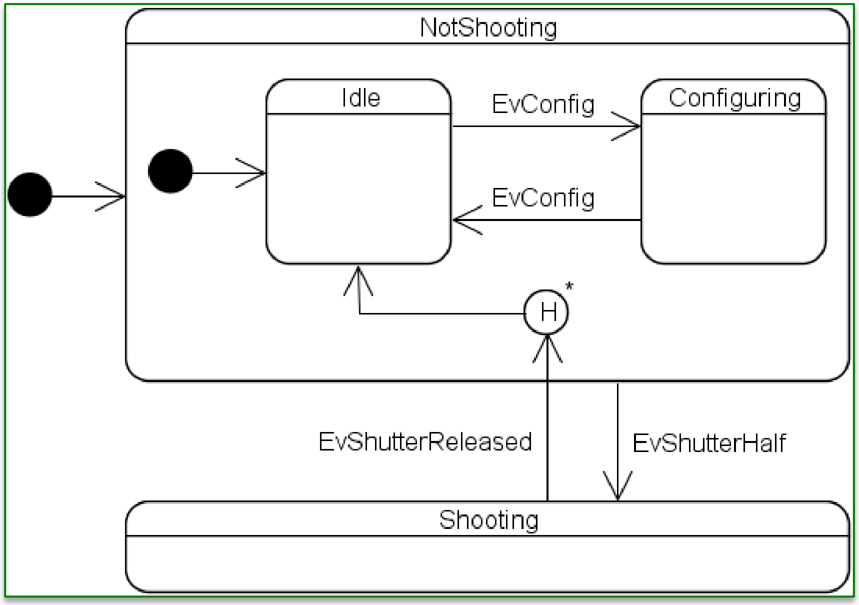
Property Specification Languages

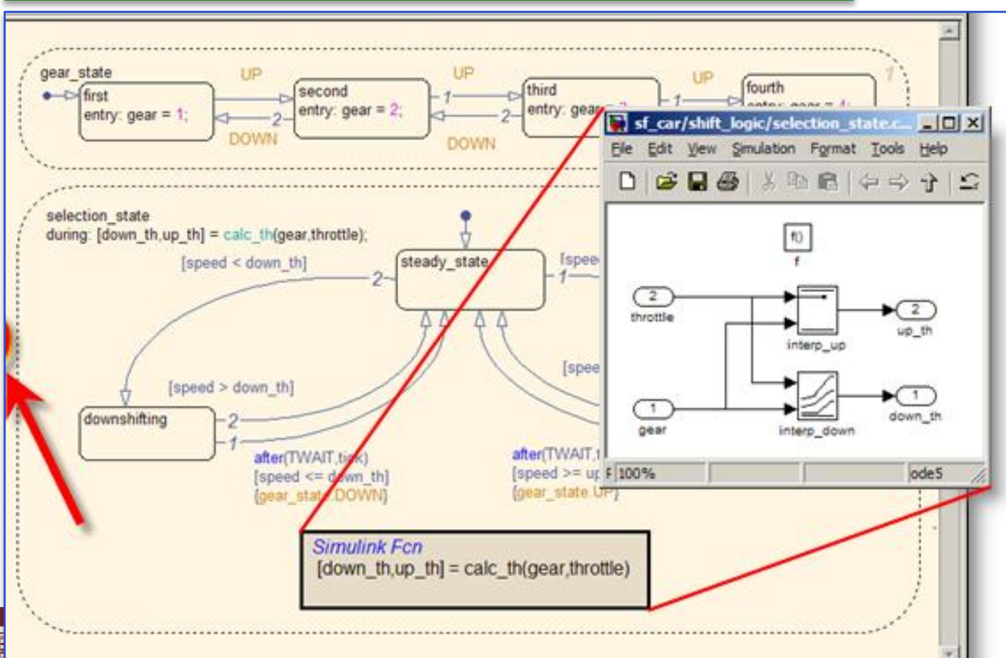




- Requirements
  - Human readable
  - Structured text (DOORS, SysML)
  - Requirements modeling notations (i*, KAOS)
- Scenarios
  - Specify permitted / forbidden execution paths
  - LTL, Temporal OCL
  - UML Sequence Diagrams
  - Message sequence charts

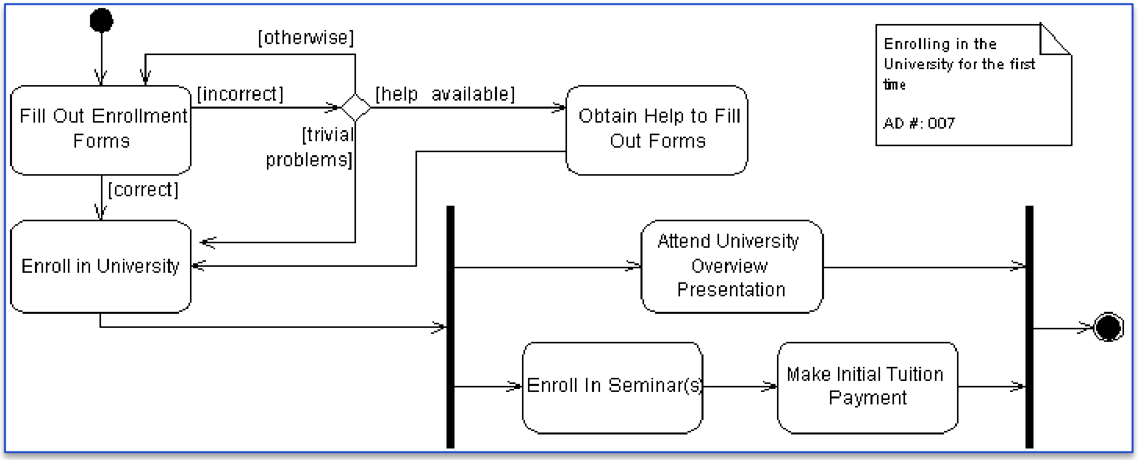
State-based languages

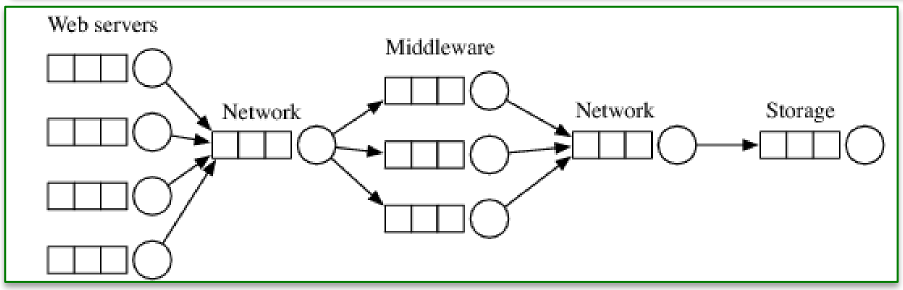


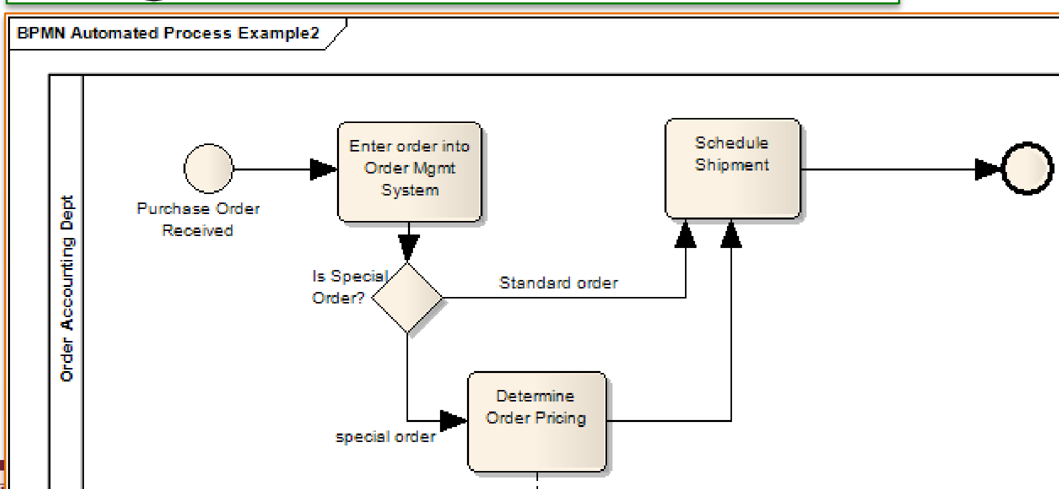


- Main concepts:
  - State , Transition
  - Event, Action
  - State hierarchy, history
- Examples:
  - Finite automata
  - Timed automata
  - Cellular automaton
  - Statemate (Harel)
  - UML Statecharts
  - Matlab Simulink Stateflow

Dataflow-based languages

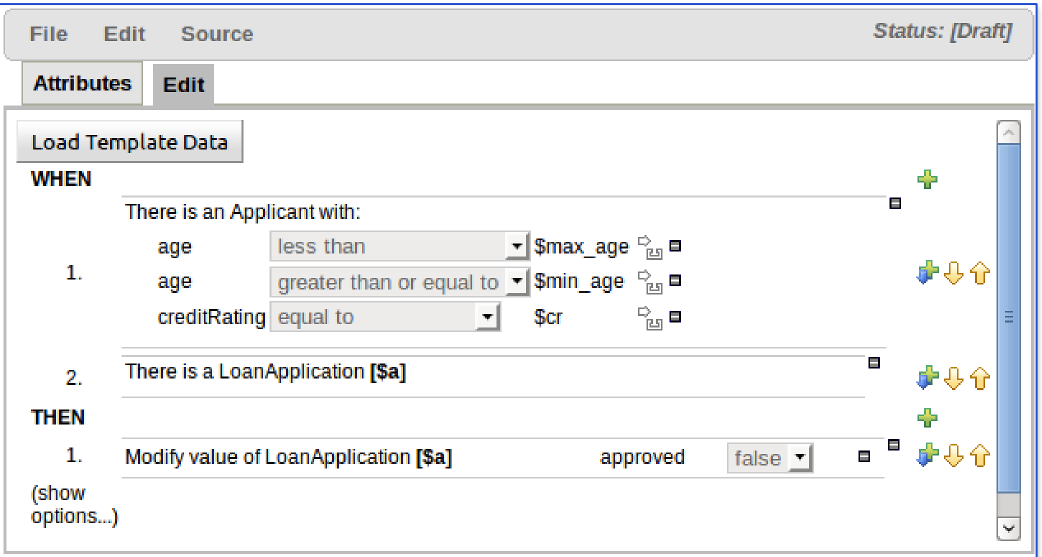






- Main concepts:
  - Process, activity channel, queue, token/message
- Examples:
  - Activity Diagrams
  - Business Process Models (also event-based)
  - Petri nets
  - Queuing networks
  - Kahn process networks
  - Esterel

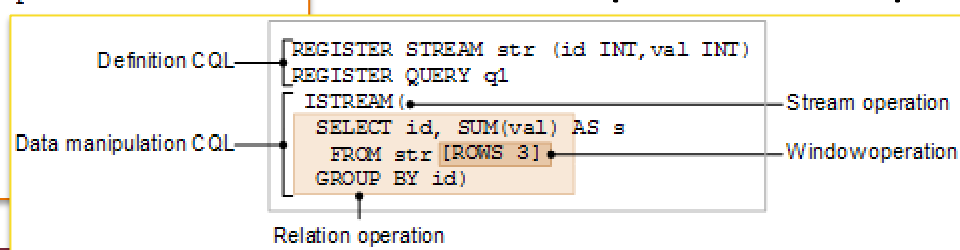
Event-based Rule languages



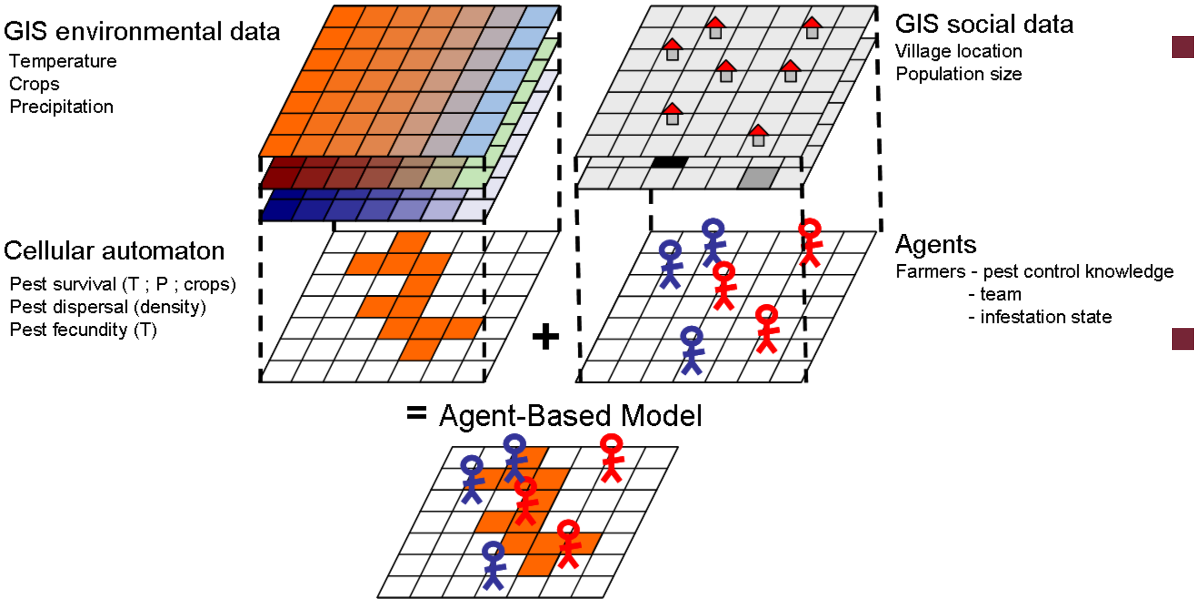

```
<processor>
    <name>stockProcessor</name>
    <rules>
        <query id="helloworldRule">
    <![CDATA[

    select T.StockName as shortName, T.LastPrice as price
    from stockInputChannel
    MATCH_RECOGNIZE (
        PARTITION BY shortName
        MEASURES A.shortName as StockName, A.price as LastPrice
        PATTERN ( A B B B A )
        DEFINE
            A as A.price > prev(A.price),
            B as B.price < prev(B.price)
    ) as T
```

- Main concepts:
  - Events (atomic, complex)
  - Event queue/stream
  - Timestamp, Time window
  - Rule(Precondition,Action)
- Examples:
  - Business rules (Drools)
  - Graph transformation
  - Stream processing (CQL)
  - Complex event processing



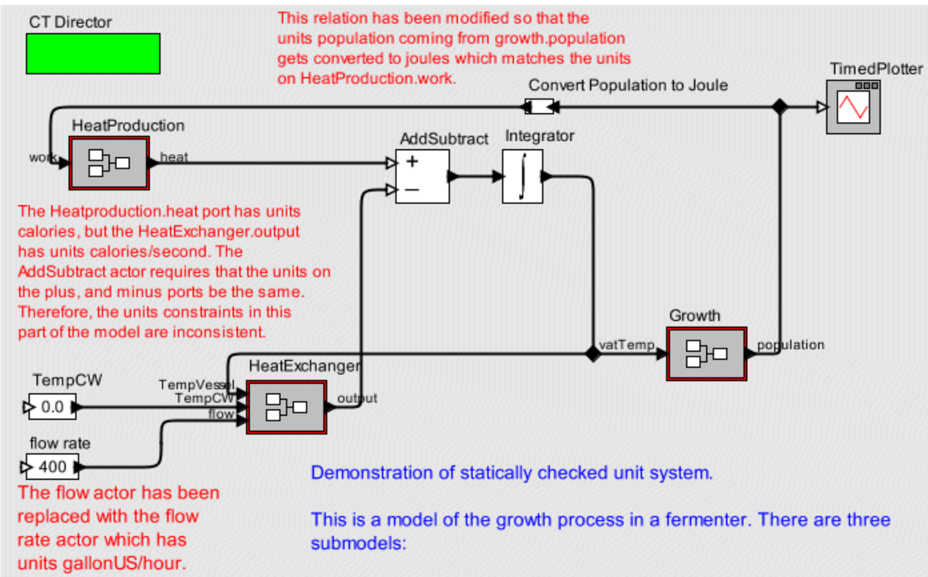
Agent-based languages

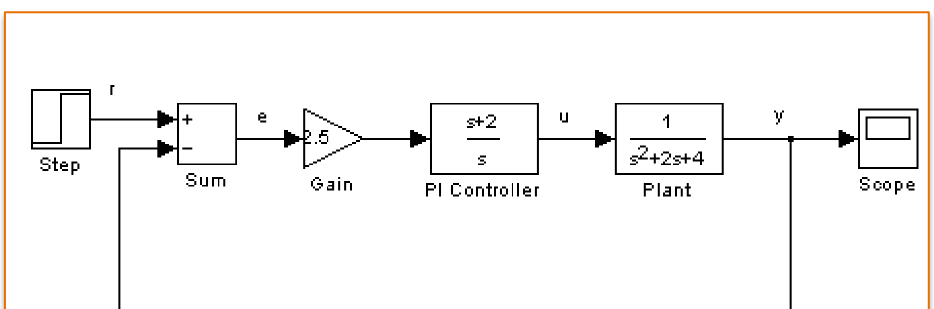


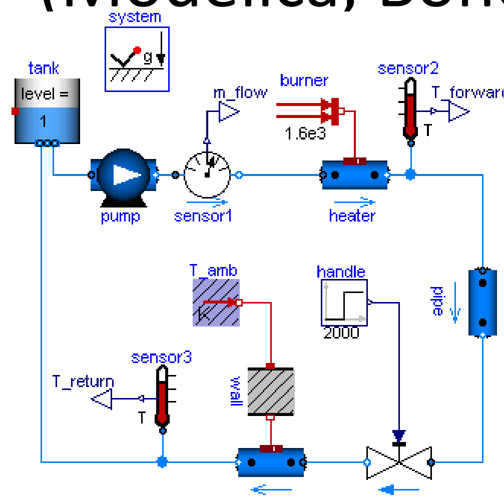
- Concepts
  - Agents + Connections
  - Behavior (create, destruct)
  - Space, Mobility,
  - Environment
- Characteristics
  - Decentralized
  - Individual-centric
- Examples:
  - AnyLogic
  - Social simulators

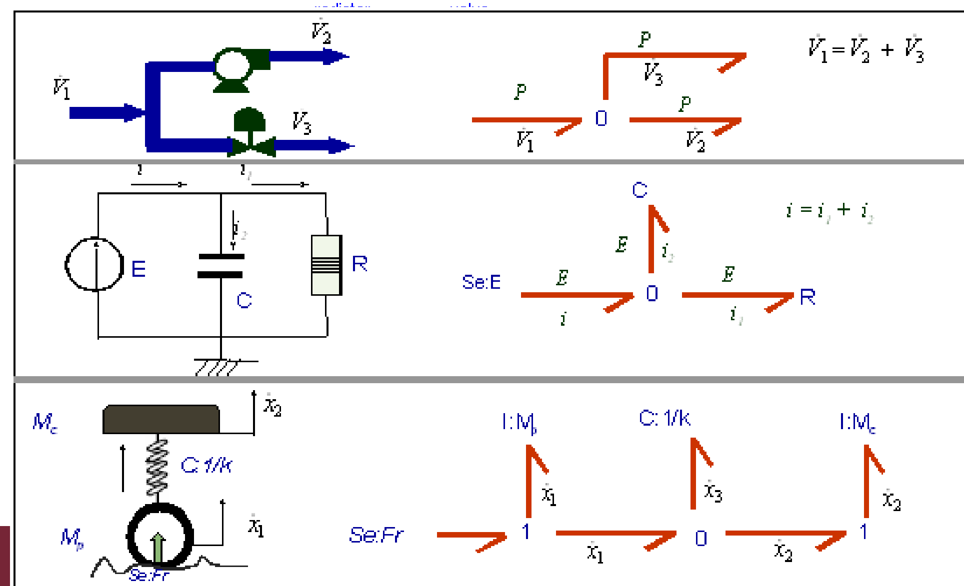
Continuous-time Languages

- Block diagrams (causal) (Simulink, Ptolemy)
- Multi-Physics (non-causal) (Modelica, Bond Graphs)



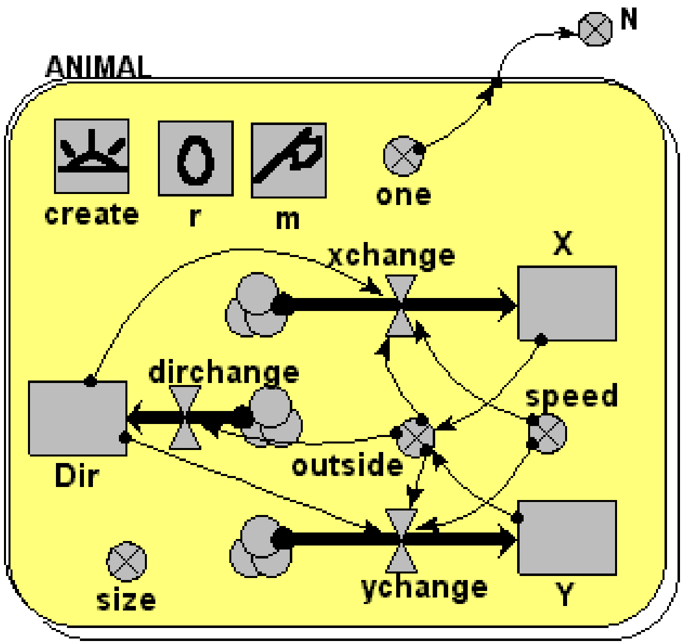




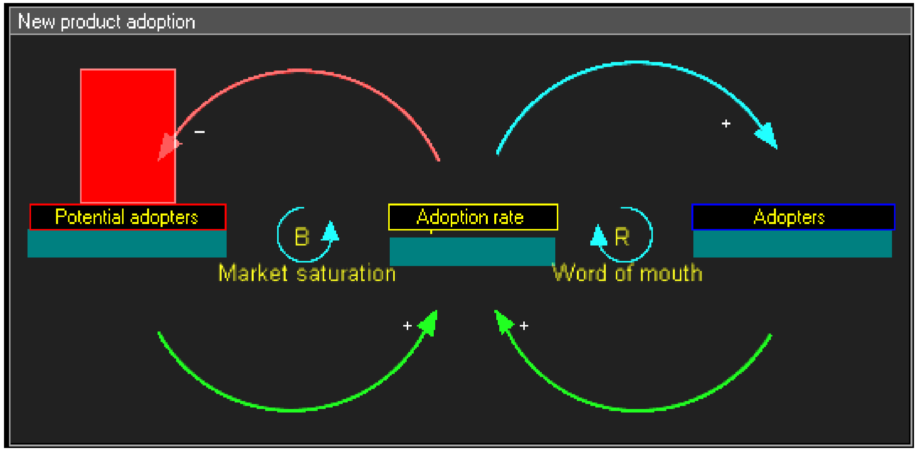


Other Dynamic Languages

- Population dynamics
  - N(t+1) = N(t) + B - D + I + E (birth, death, immigrants, emmigrants)
  - Calculation of rates

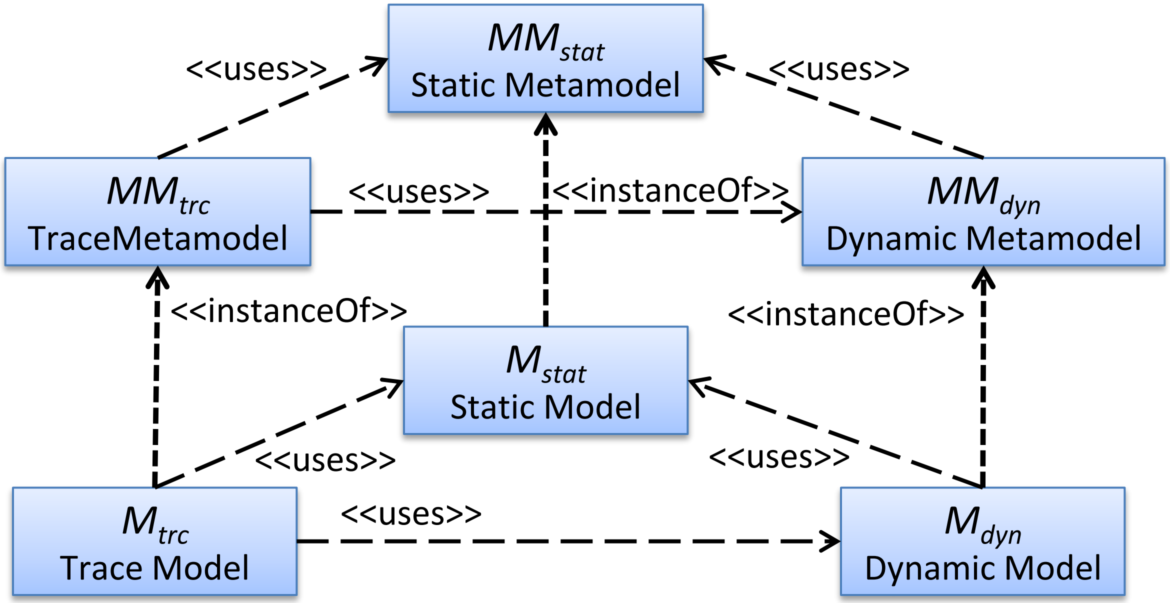


- Forrester System Dynamics
  - Stocks, Flows
  - Feedback, Time delays



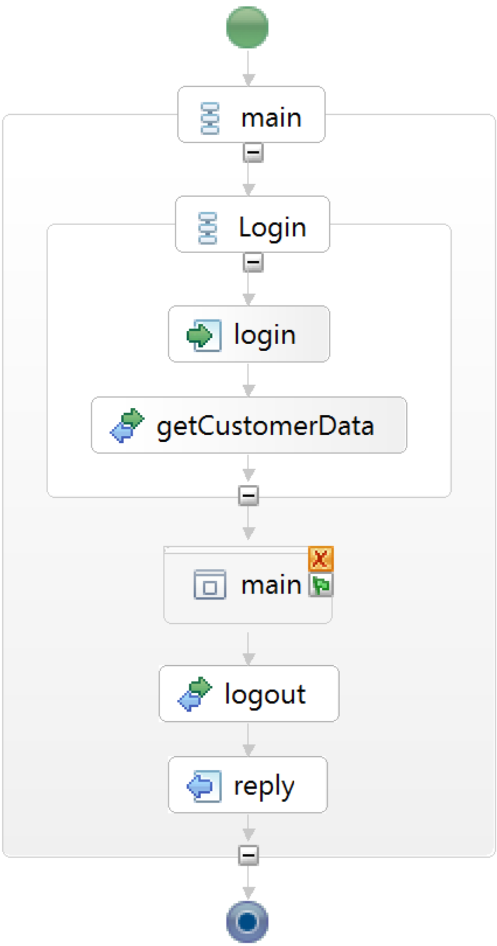
Dynamic Metamodeling in DSLs

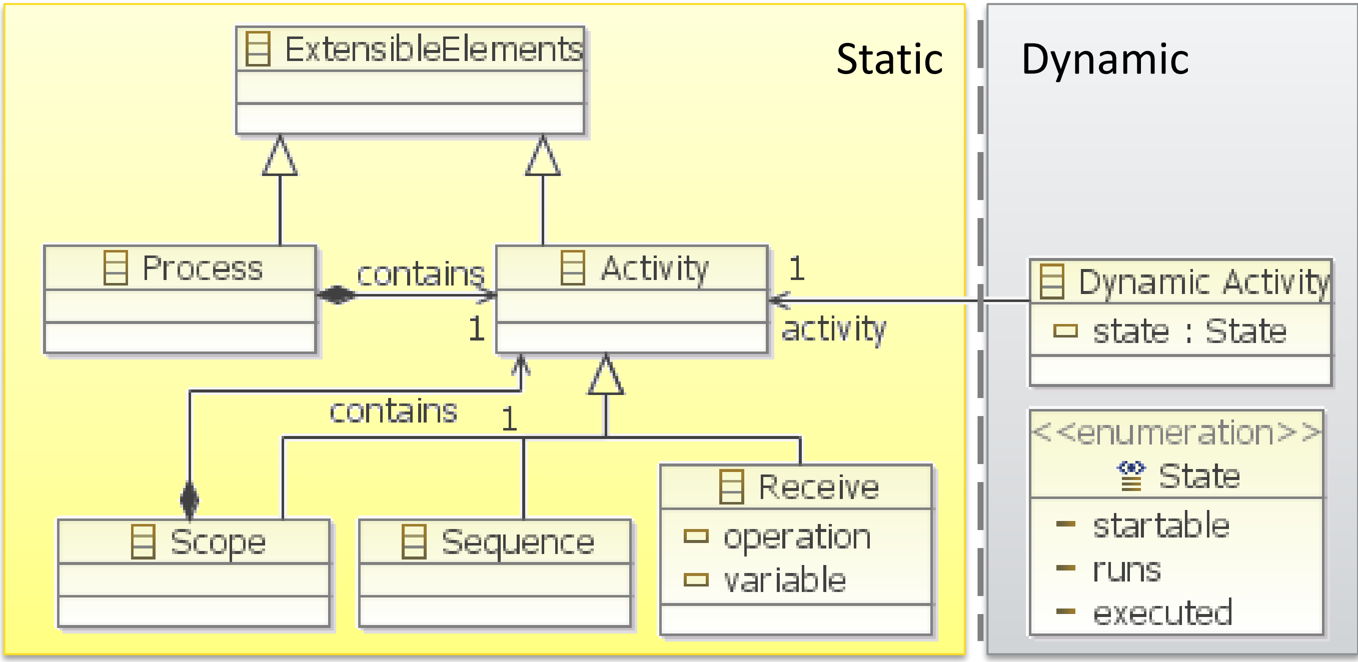
Dynamic Metamodeling in DSLs

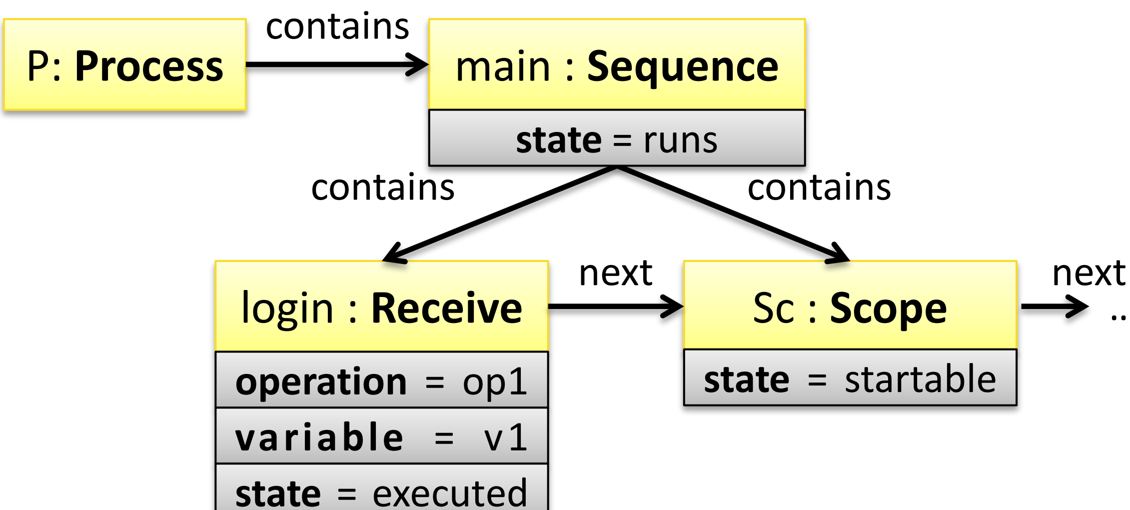


- Complement Static Metamodel with
  - Dynamic metamodel: currentState, configuration, etc.
  - Execution trace metamodel: previous state, replay

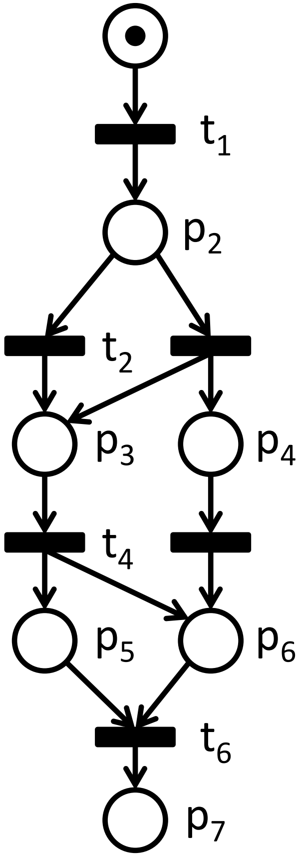
Example 1: Business Processes

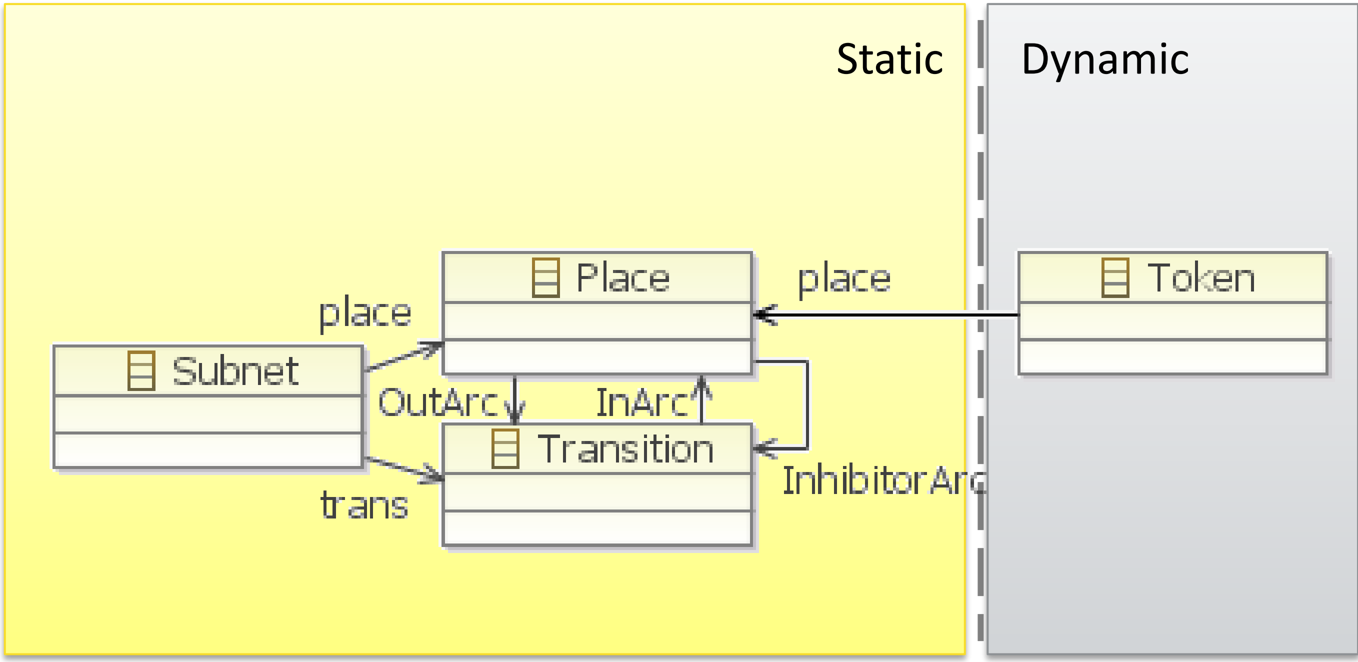


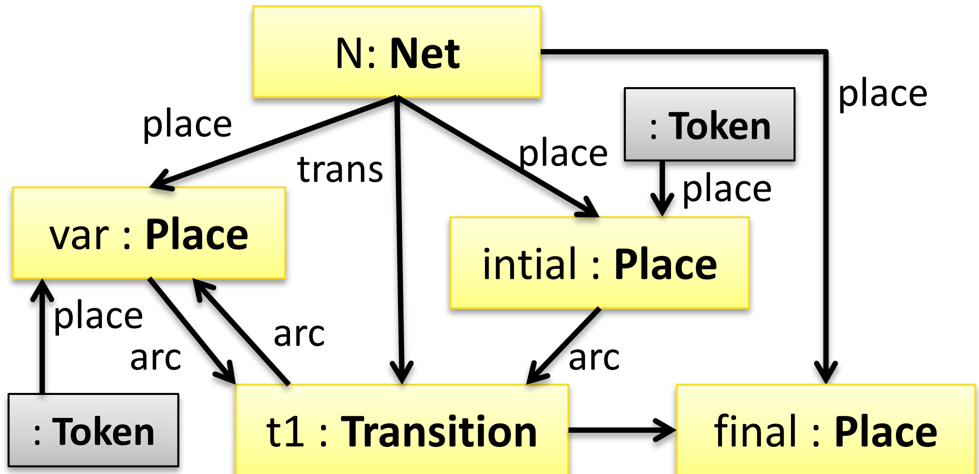




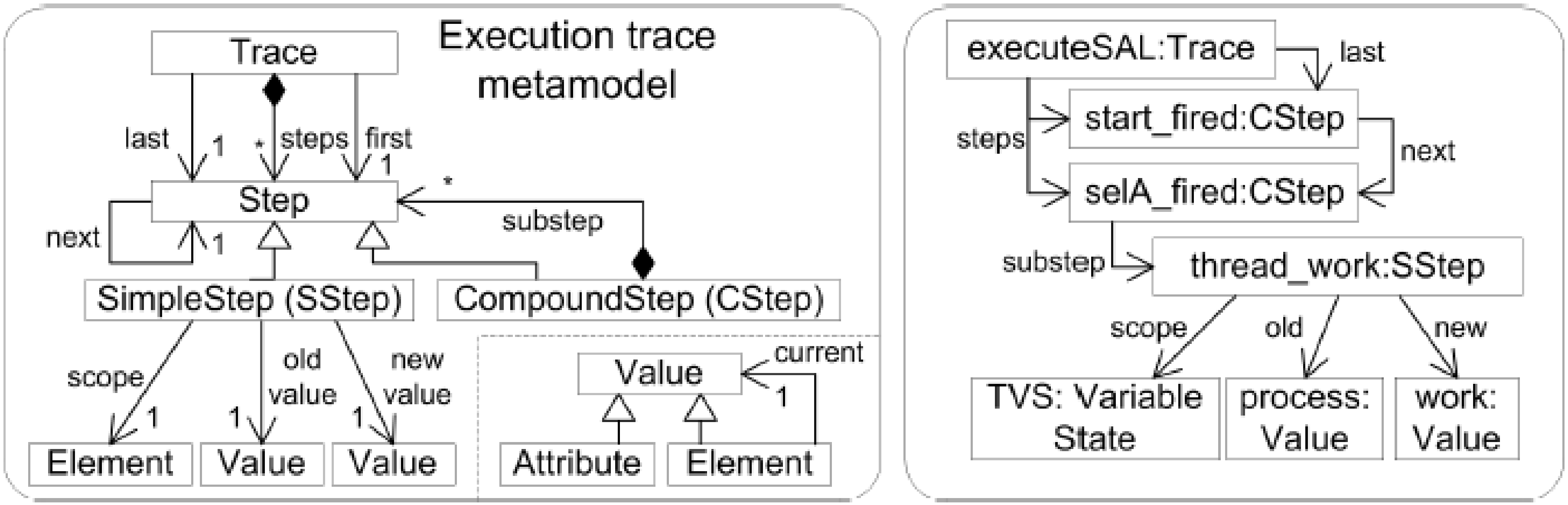
Example 2: Petri Nets







Metamodeling of Execution Traces



- Representation for
  - Hierarchy of steps (simple, compound)
  - Old value ➔ New value
  - Aim: Replayable

Statecharts for Modeling Reactive Behavior

Statecharts

State-based behaviour modeling

- **State partition** (AKA **state space**)
  - A set of distinguished **system states**
  - Examples
    - {Mon, Tue, Wed, Thu, Fri, Sat, Sun}
    - States of microwave oven: {full power, defrost, off}
  - **DEF**: A state partition is a set, exactly one element of which characterizes the system at any time.
- **Current state**
  - E.g. today is Wed, the microwave is on defrost, etc.
  - **DEF**: At any given moment, the current state is the element of the partition which is currently valid.

Composite state modeling

- Modeling complex systems
  - Asynchronous components
  - Composite state space as **product of state spaces**
- Challenge: scalability
  - Exponential explosion of state space
    - 10 components of 6 local states each → 6^{10} states!
  - More concise notation required
- Solution: statechart languages
  - Hierarchical refinement with history
  - Concurrent regions

Statecharts = States + Transitions

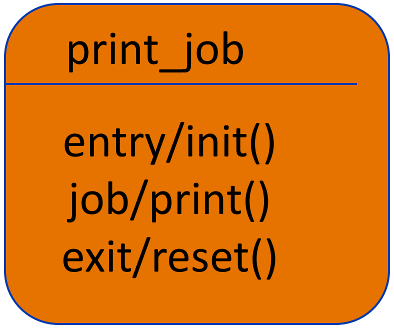
- Describes the states and state transitions of the system, of a subsystem, or of one specific object.
  - hierarchical and concurrent systems
- States
  - Concrete state:
    - Combination of possible values of attributes
    - Can have an infinite state space
  - Abstract states: (like in Statecharts)
    - Predicates over concrete states
    - One abstract state ← many concrete states
    - Hierarchical states:
      - Frequent in embedded apps (e.g. control of car brake)
- Transitions
  - Triggering Event
  - Guard
  - Action

Statechart - introduction

- For defining reactive behavior of objects
  - Responds to events:
    state transitions and actions
  - Traditional approach: state machine
- Statechart: extension to state machine
  - State hierarchy: refinement of states
  - Concurrent behavior: parallel threads
  - Memory: last active state configuration

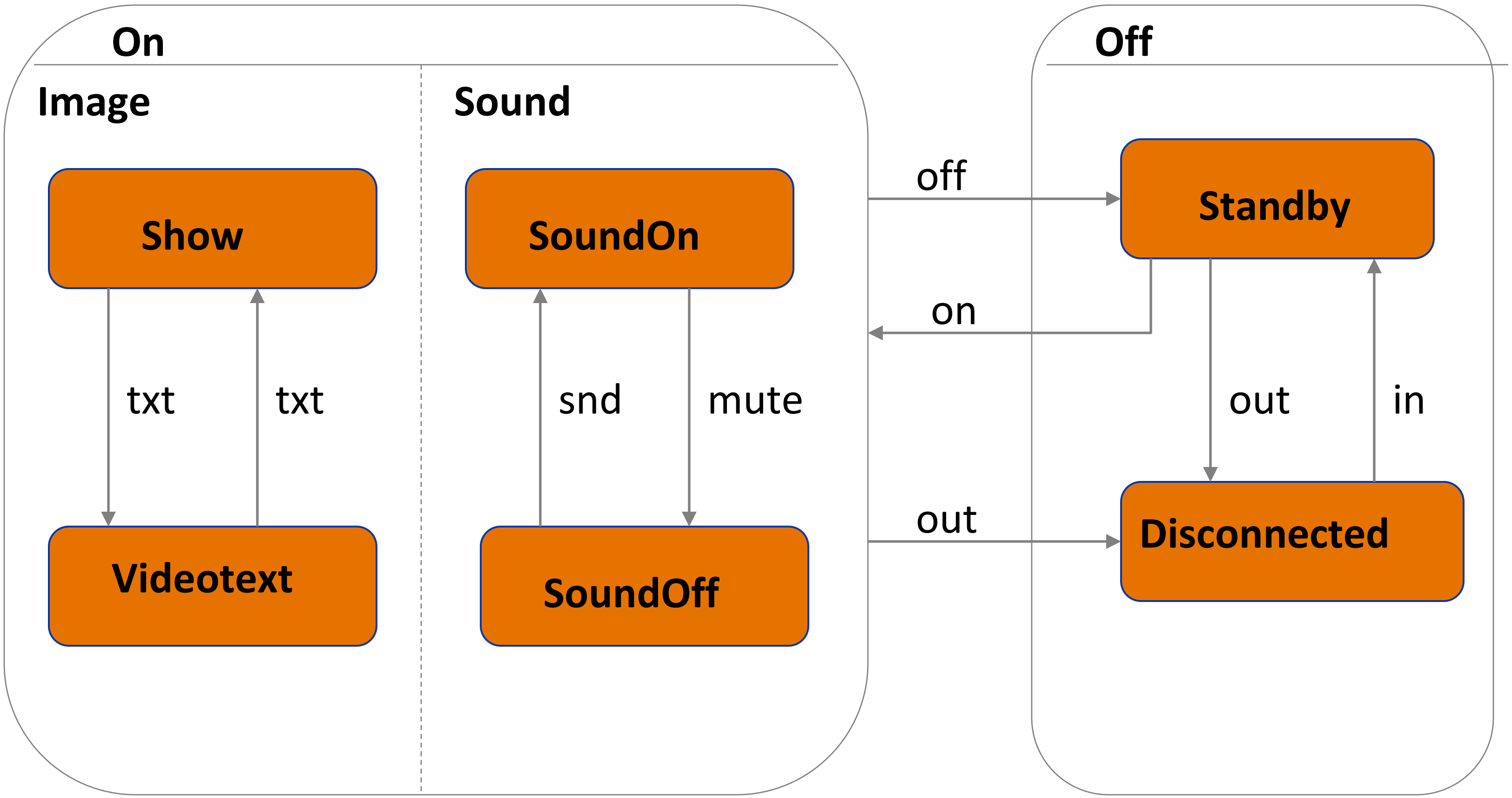
States I.

- Attributes:
  - entry action
  - exit action
  - static reaction



- State refinement
  - Simple state
  - OR refinement: auxillary state machine, only one active state
  - AND refinement: concurrent regions (state machines), all regions are active in parallel

Example for state refinement: TV

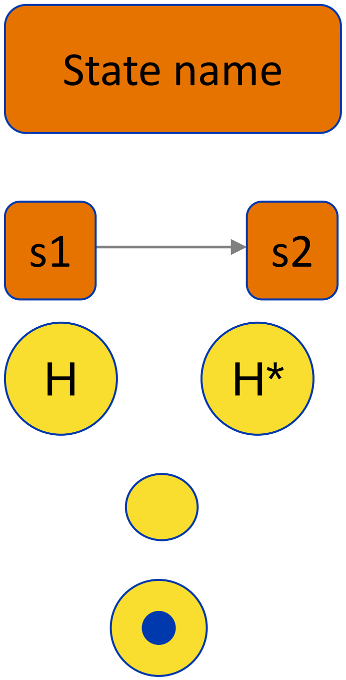


State II.

- History state
  - Stores the last active state configuration
  - Input transition: it sets the object to the saved state configuration
  - Output transition: defines the default state, if there were no active state since
- Inital state: becomes active when entered to the region
    - One in each OR refinement
    - One in each AND region
- Final state: state machine terminates

Statechart elements

- State
- (Transition)
- History state
- Initial State
- Final State



Transition I.

- Defining state changes
- Syntax:

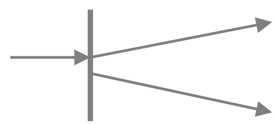
  trigger [guard] / action

  - trigger: event, triggered operation or time-out
  - guard: transition condition
    - Logic formula over the attributes of the objects and events
    - referring to a state: IS_IN(state) macro
    - Without trigger: if becomes true the transition is active
  - action: operations $\Rightarrow$ action semantics

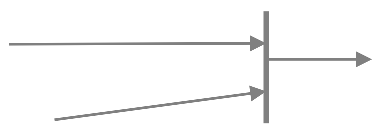
Transition II.

- Time-out trigger:
  - becomes active if the object stayes in he source state for the predefined interval

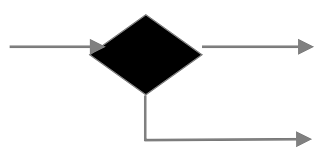
    e.g., tm(50), based on system time
- Complex transitions
  - Fork



  - Join

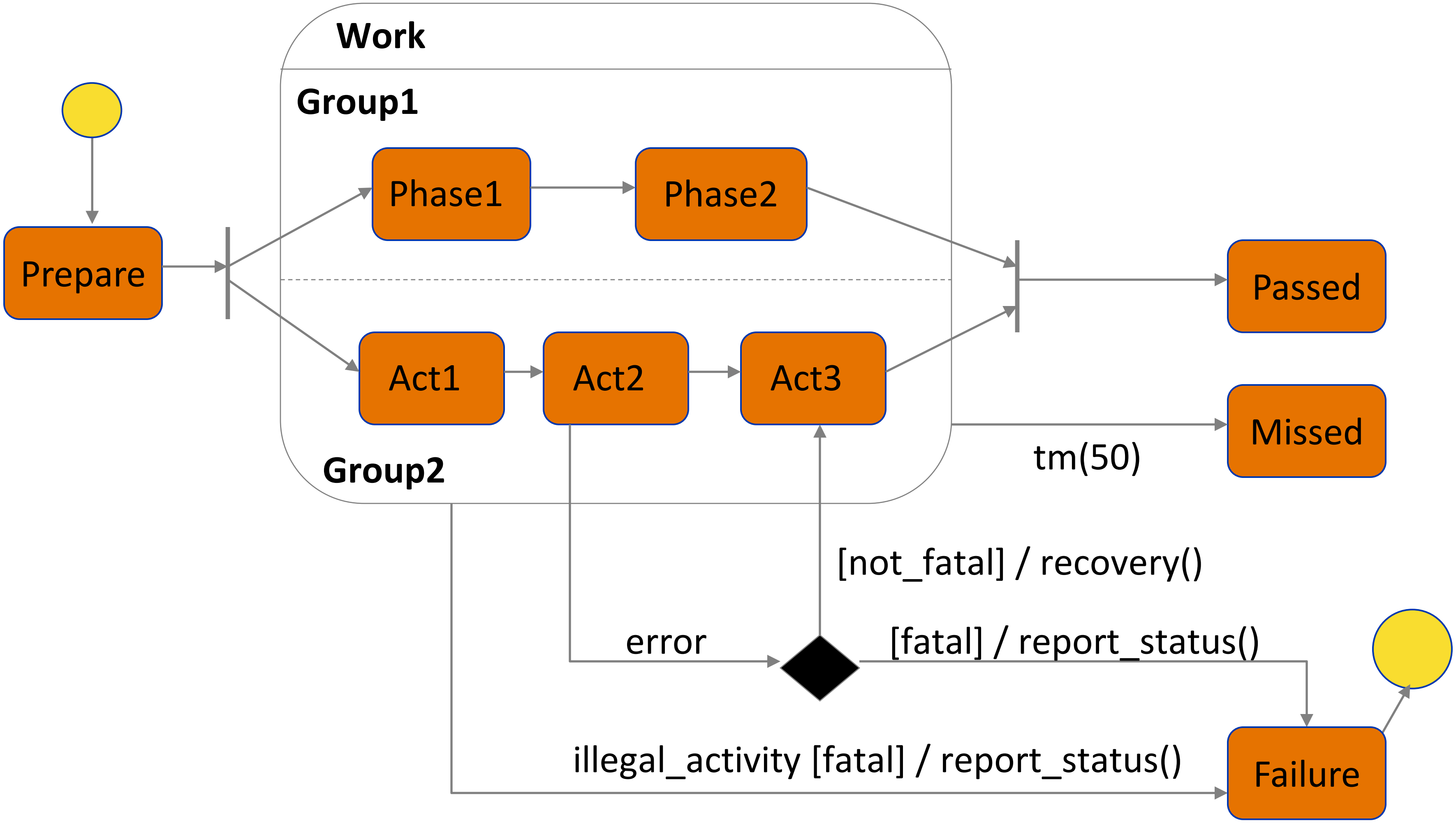


  - Condition



- Transitions between different hierarchy levels

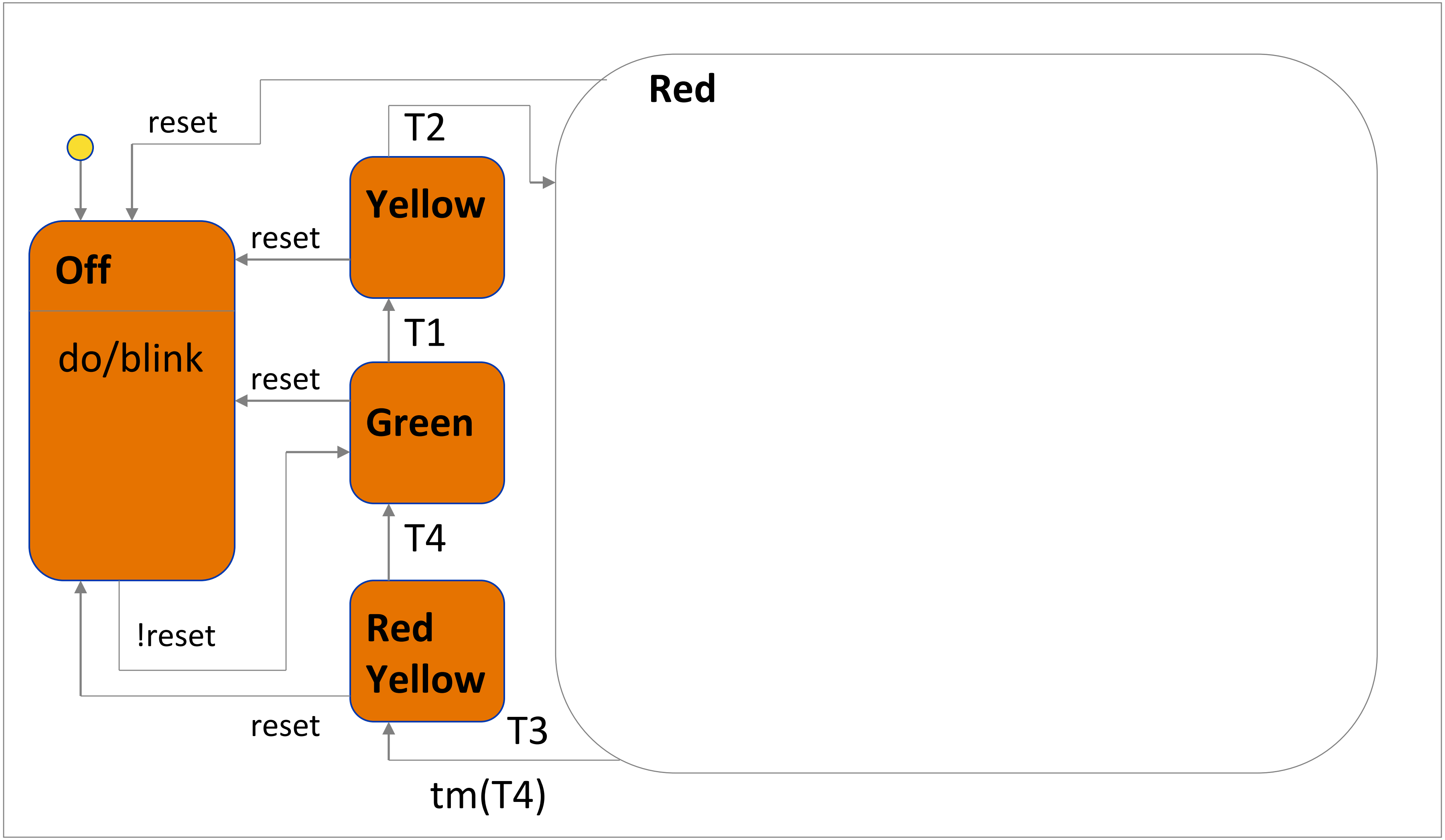
Transition example



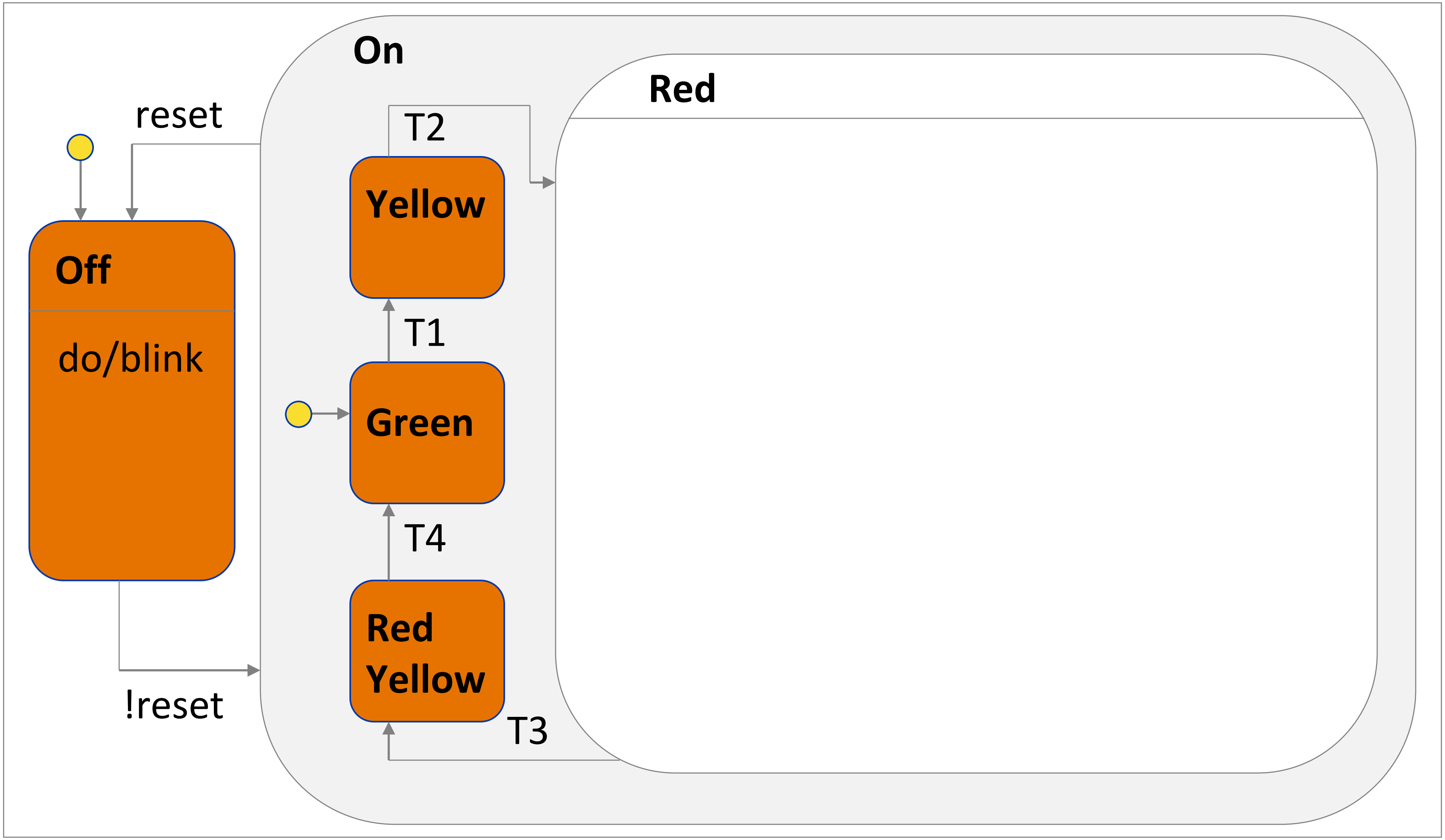
Complex Example

- Traffic light for an intersection with a prioritized road
  - Off: (blinking yellow)
  - On: green for the priority road
  - Green, yellow, red etc. Different timerange (timer)
  - 3 waiting vehicle on priority road: green light despite the timer’s ticks
  - Automatically take photos of vehicles crossing the piority road on red light. Manual on/off for this feature.

1. Basic state machines

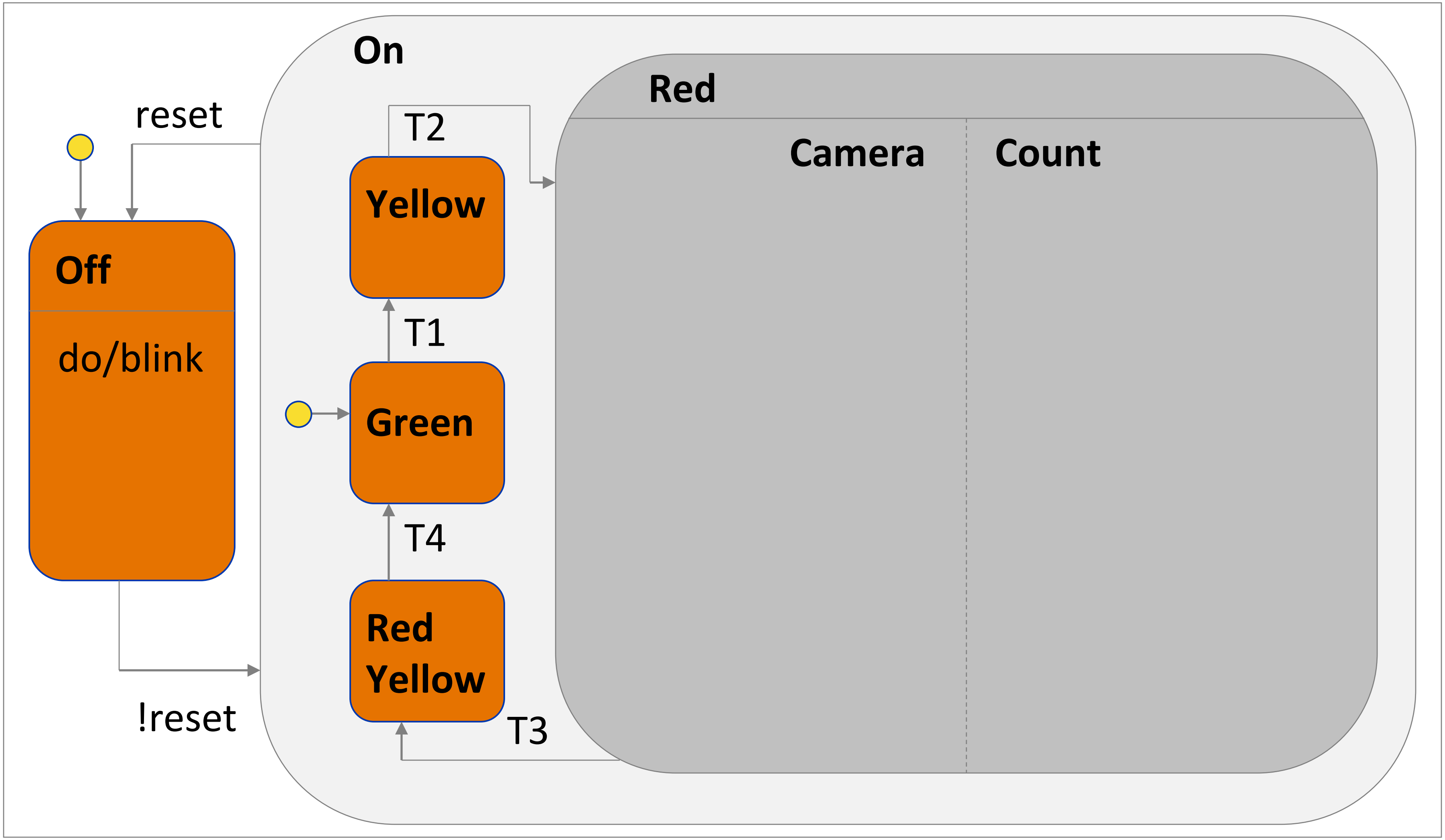


2. Hierarchy

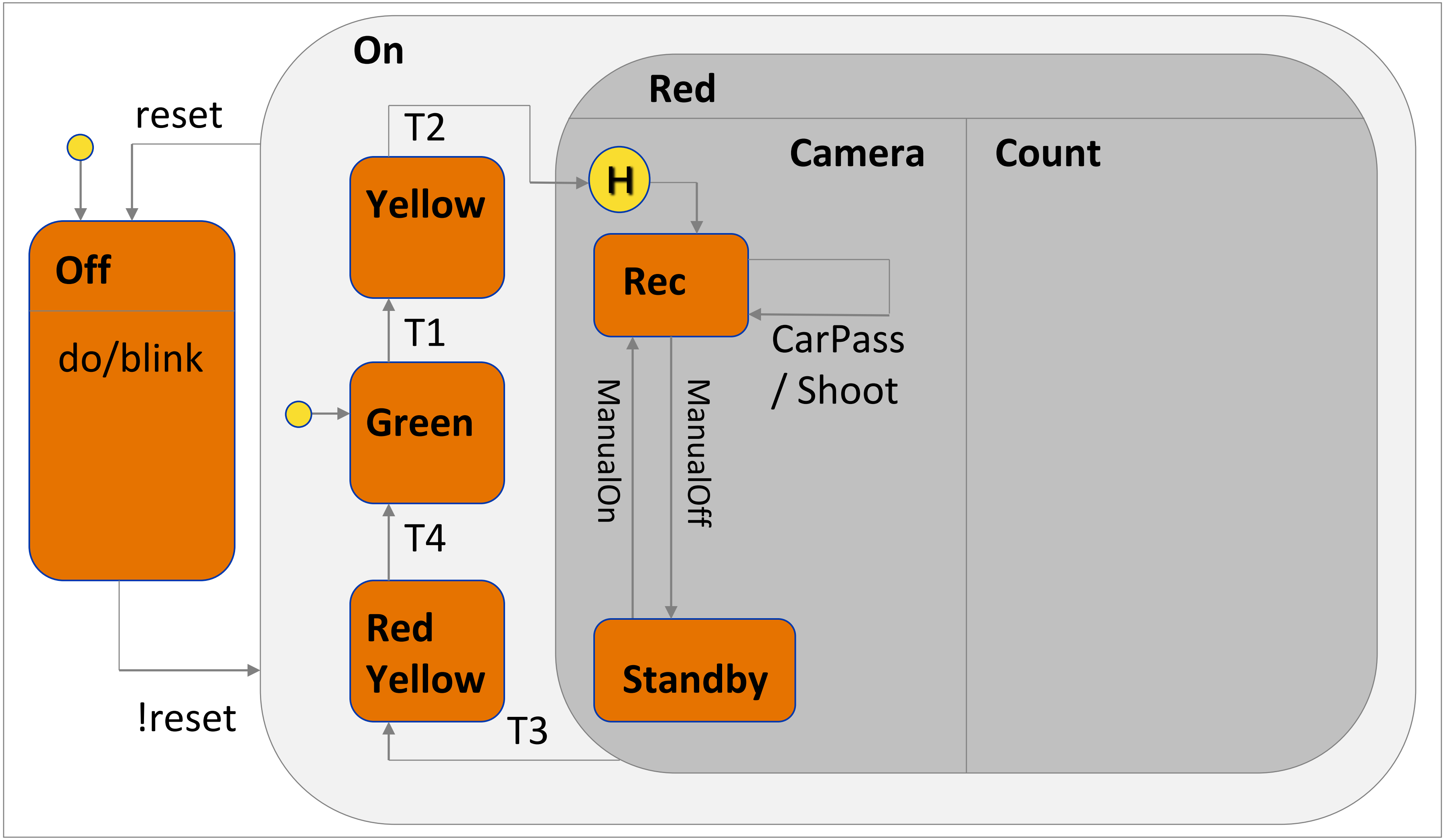


On
Red
reset
T2
Yellow
Off
do/blink
T1
Green
T4
Red
Yellow
!reset
T3

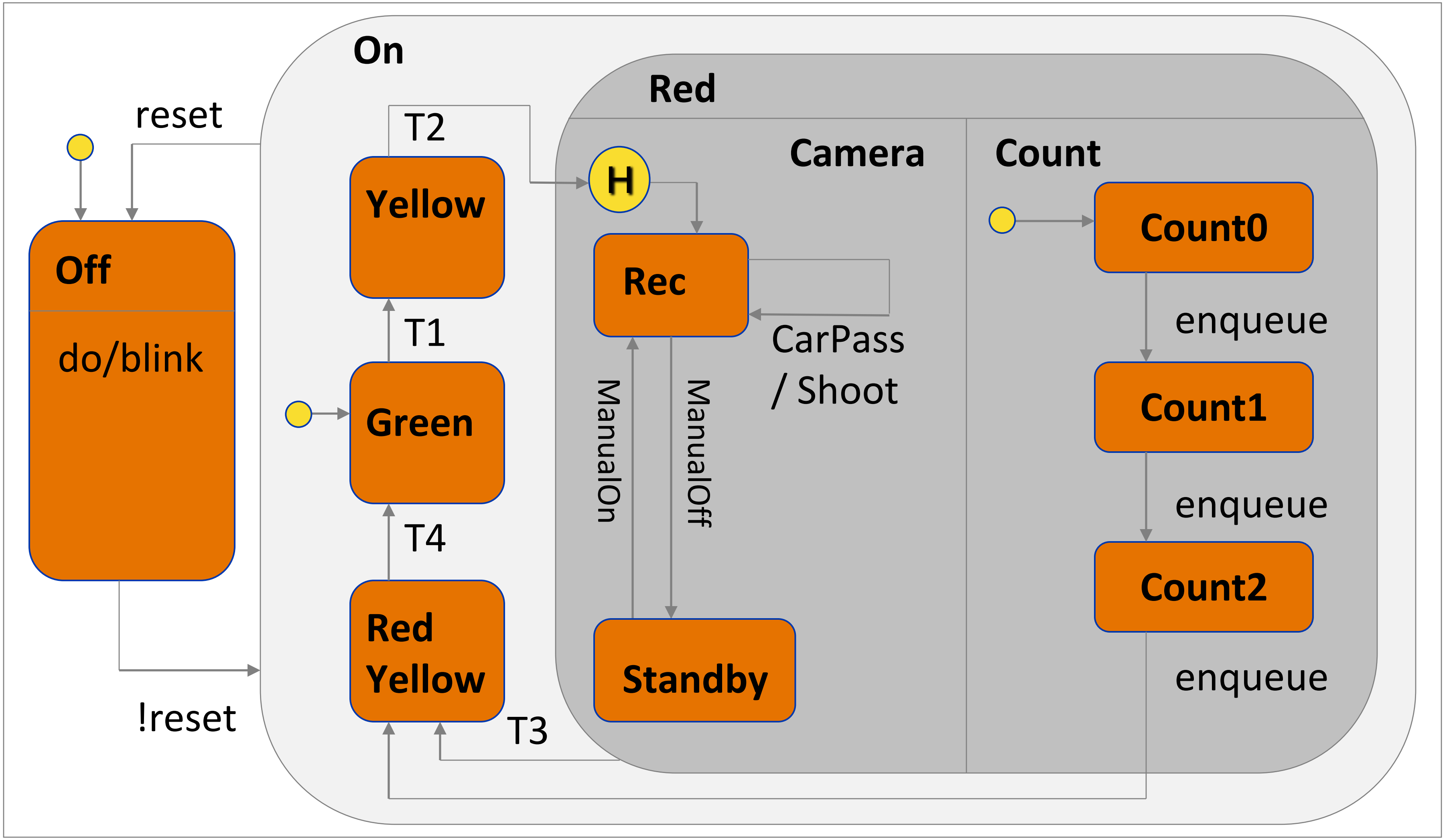
3. Concurrent states



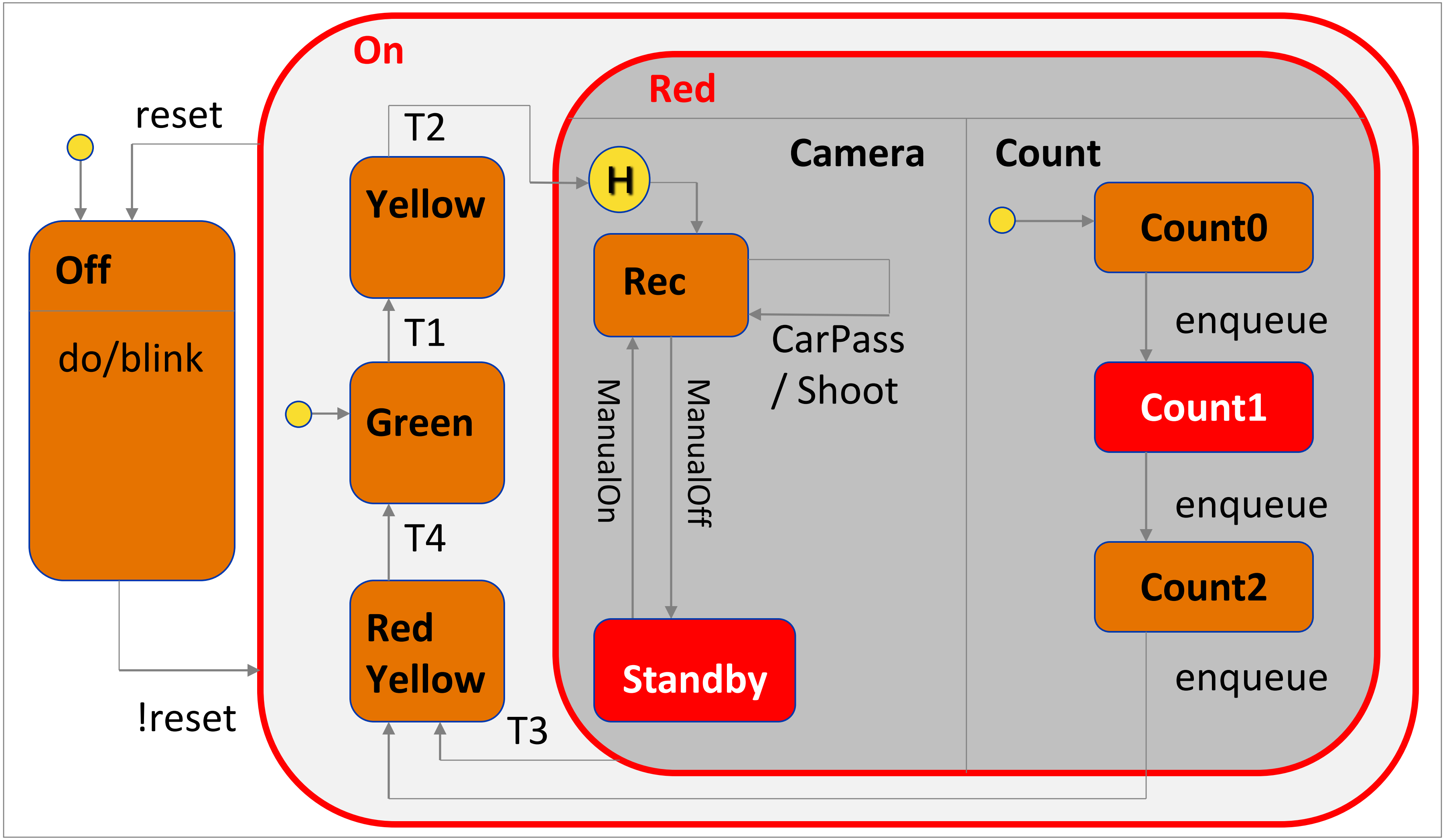
4. History States



Complete System



Example Concrete State



Active states:
{Standby, Count1, Red, On}

Inactive states:
{Off, Yellow, Green, RedYellow, Rec, Count0, Coun2}

Semantics: How does it work?

- Basics:
  - Hierarchical state machine (state chart)
  - Event queue + scheduler
- Semantics defines:
  Behavior in case an event occurs
  $\rightarrow$ one step of the state chart
  - (concurrent) transitions fire
  - State configuration changes
    in all region in the active state and also one substate in the OR refinement (recursively)

Semantics of State Transitions

- Separately processed events:
  - Scheduler only triggers the next event if the previous one is completely processed
    stable configuration: there is no state change without an event
- Complete processing of events:
  - The largest set of possible fireable transitions
    (all enabled transitions fire, if they are not in conflict)
  - How does it work?:
    - →Steps of the event processing

Steps of event processing I.

- Scheduler triggers an event for the statechart in a stable state configuration
- Enabled transitions:
  - Source state is active
  - The event is their trigger
  - Guards are evaluated to true

Based on the number of fireable transitions

- Only one: fire!
- None: do nothing
- More than one: select transitions to fire?

Steps of event processing II.

- Selection of fireable transitions:
  - Fireable = Enabled + Max priority
  - Conflict: Has the same source state
    - Formally: the intersection of their left (exit) states is not empty

→Conflict resolution → priority:

- Defined between two transitions (t_1 and t_2)
- $t_1 > t_2$, if and only if the source state of t_1 is a substate within the state hierarchy of t_2 („lower level")

→Priority insufficient to resolve conflict if

- Same source state (or parallel subregions)

Steps of event processing III.

- Selection of transitions to fire:
  - Parallel execution of concurrent transitions
    - Maximal set of fireable transitions
      (= cannot be extended any further)
    - There is no conflict between any two transitions
  - Selection of this set:
    - <u>Nondeterministic!</u>

Steps of event processing IV.

- Selected transitions fire:

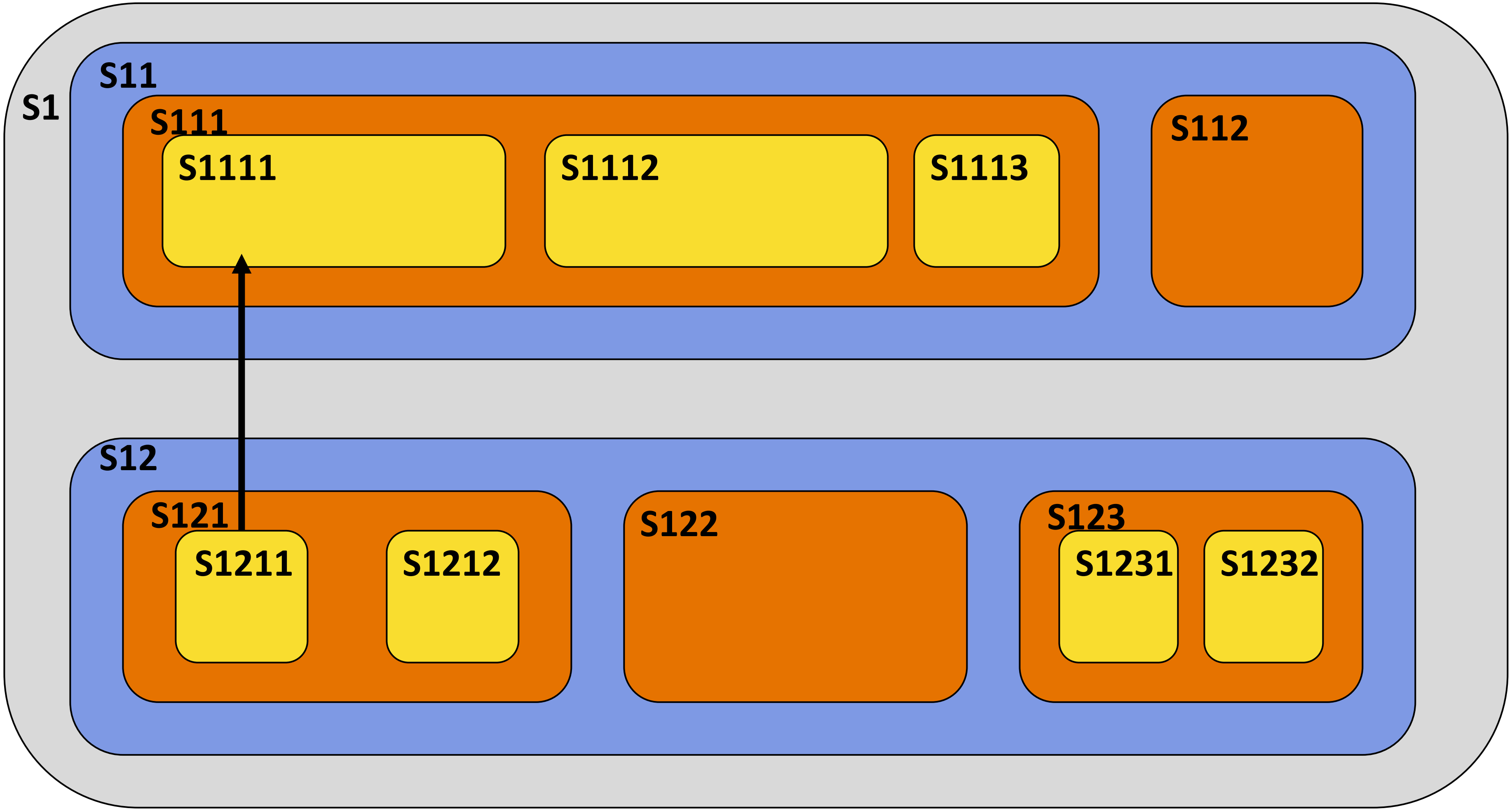
  in nondeterministic order

- Firing one transition:
  - Leaving the source states from the bottom to top and execute all their exit operations
  - Execute the action of the transition
  - Entering the target states from top to bottom and execute the entry actions $\rightarrow$ new state configuration

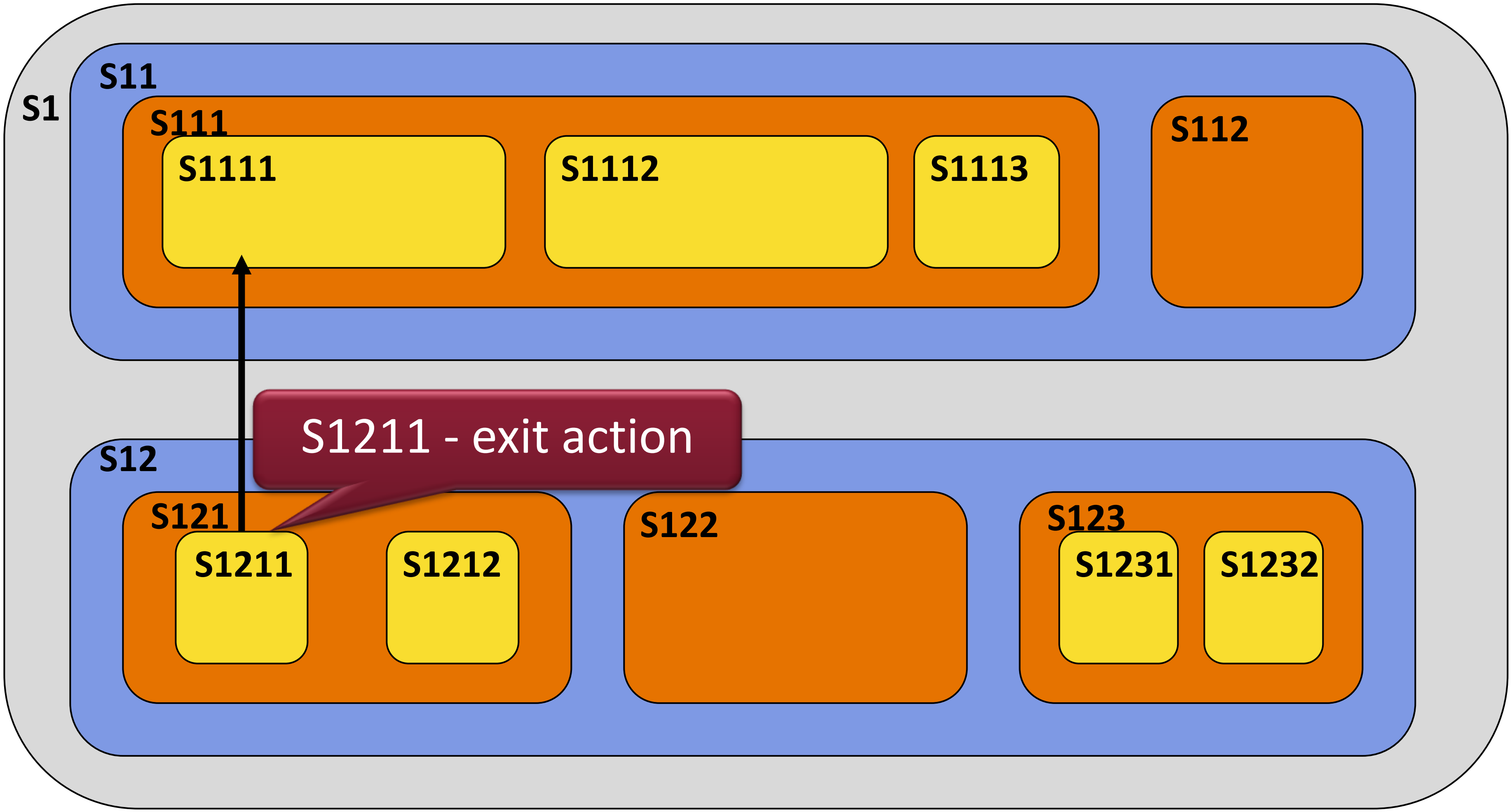
Steps of event processing V.

- Entering a new state configuration:
  - Simple target state: part of the state configuration
  - Non-concurrent superstate: direct target of one of its substate or its initial state
  - Concurrent target state: all of its regions have to have an active state either as direct target state (maybe via fork) or as initial state
  - History state : the last active state configuration
    if there is none: the target state of the history state

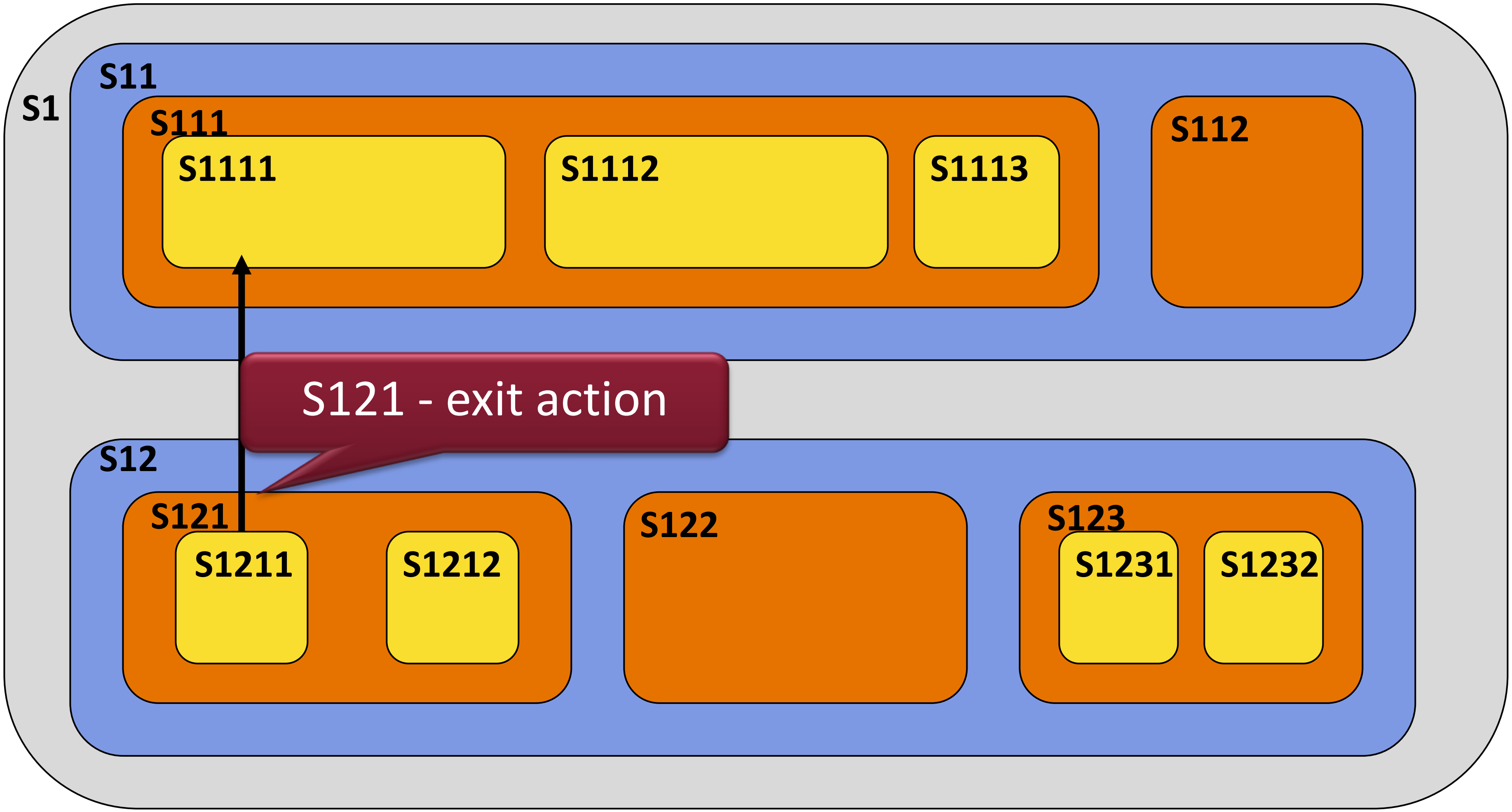
State transition example



State transition example

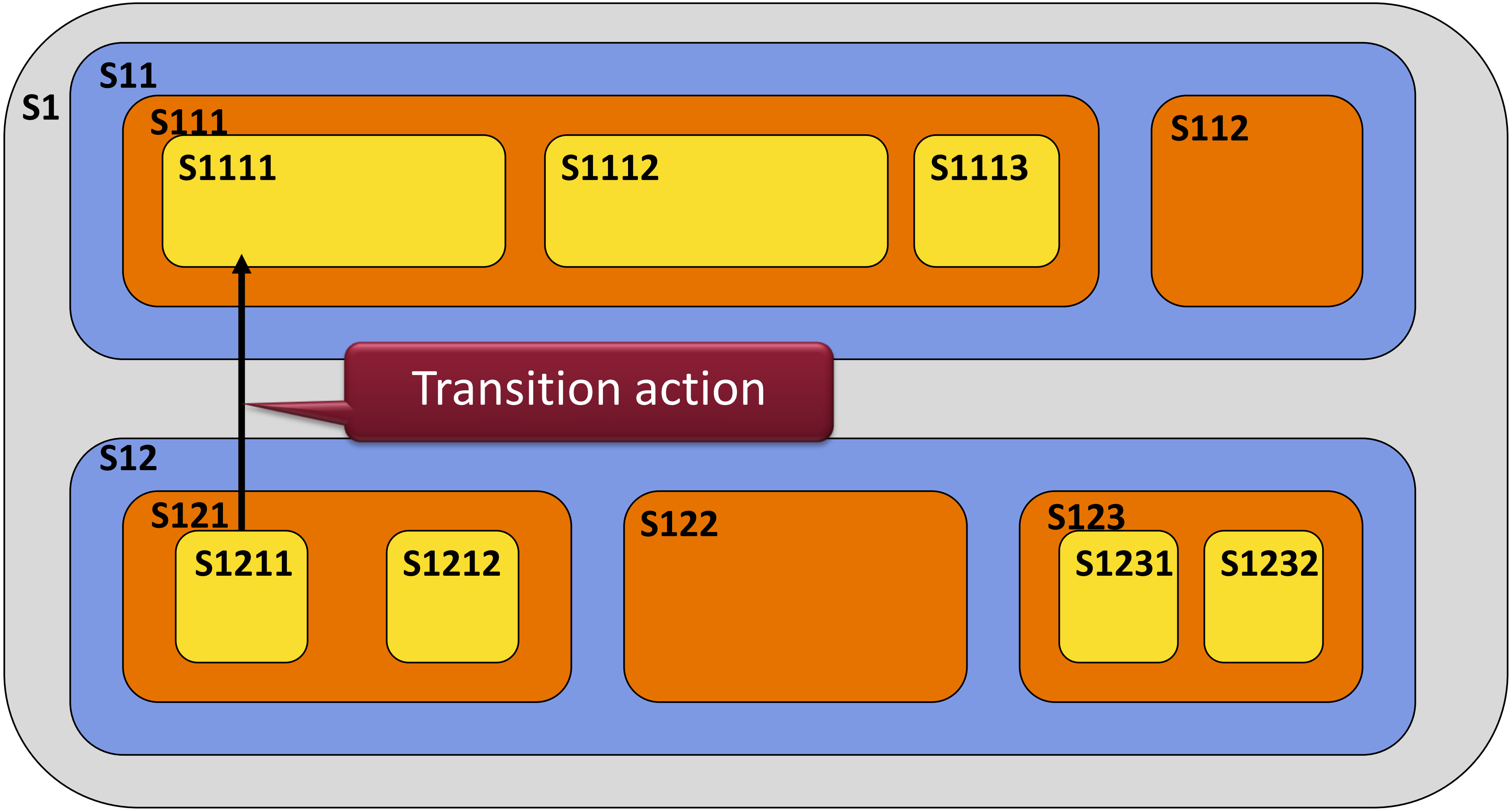


State transition example

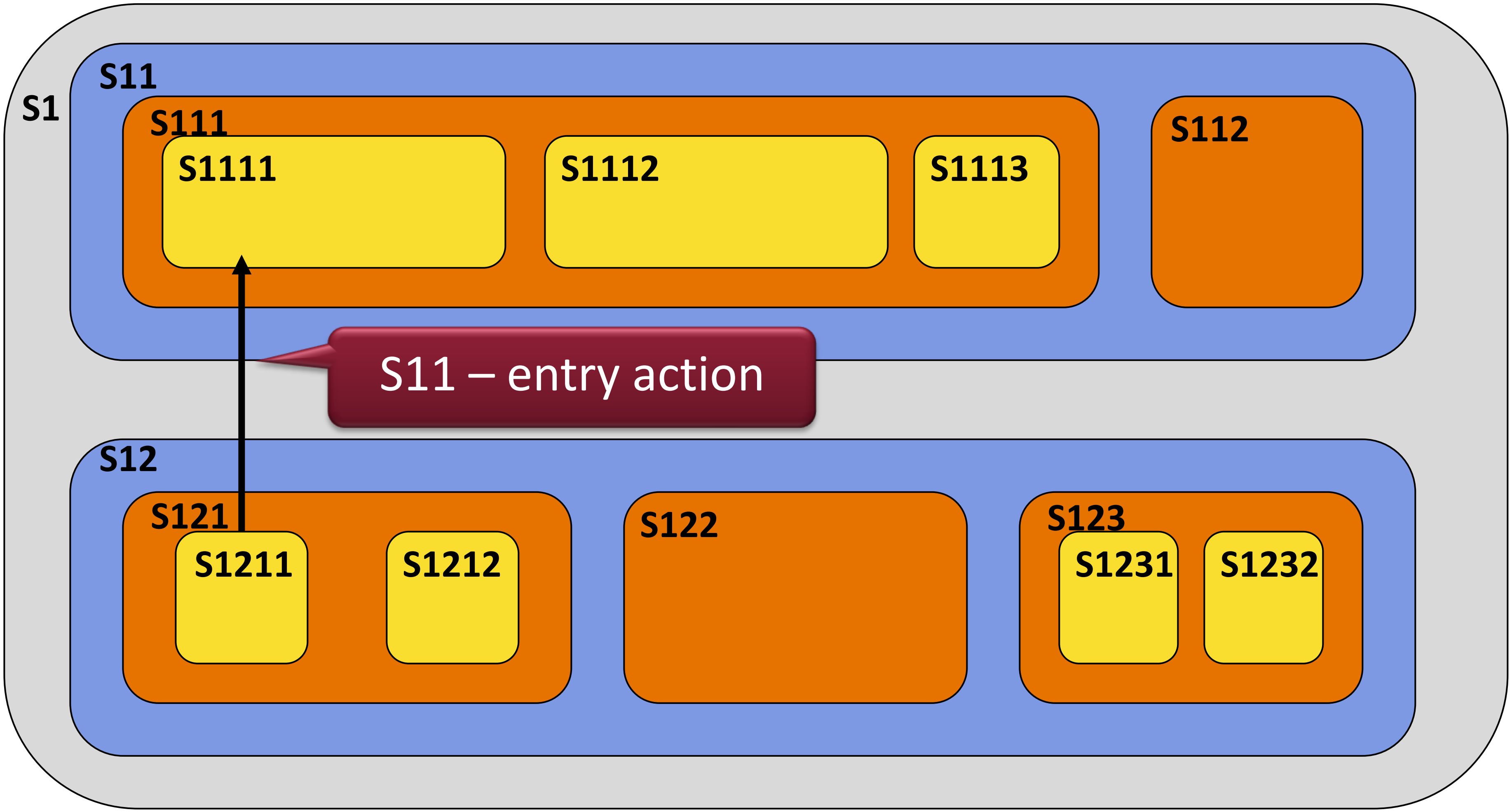


State transition example

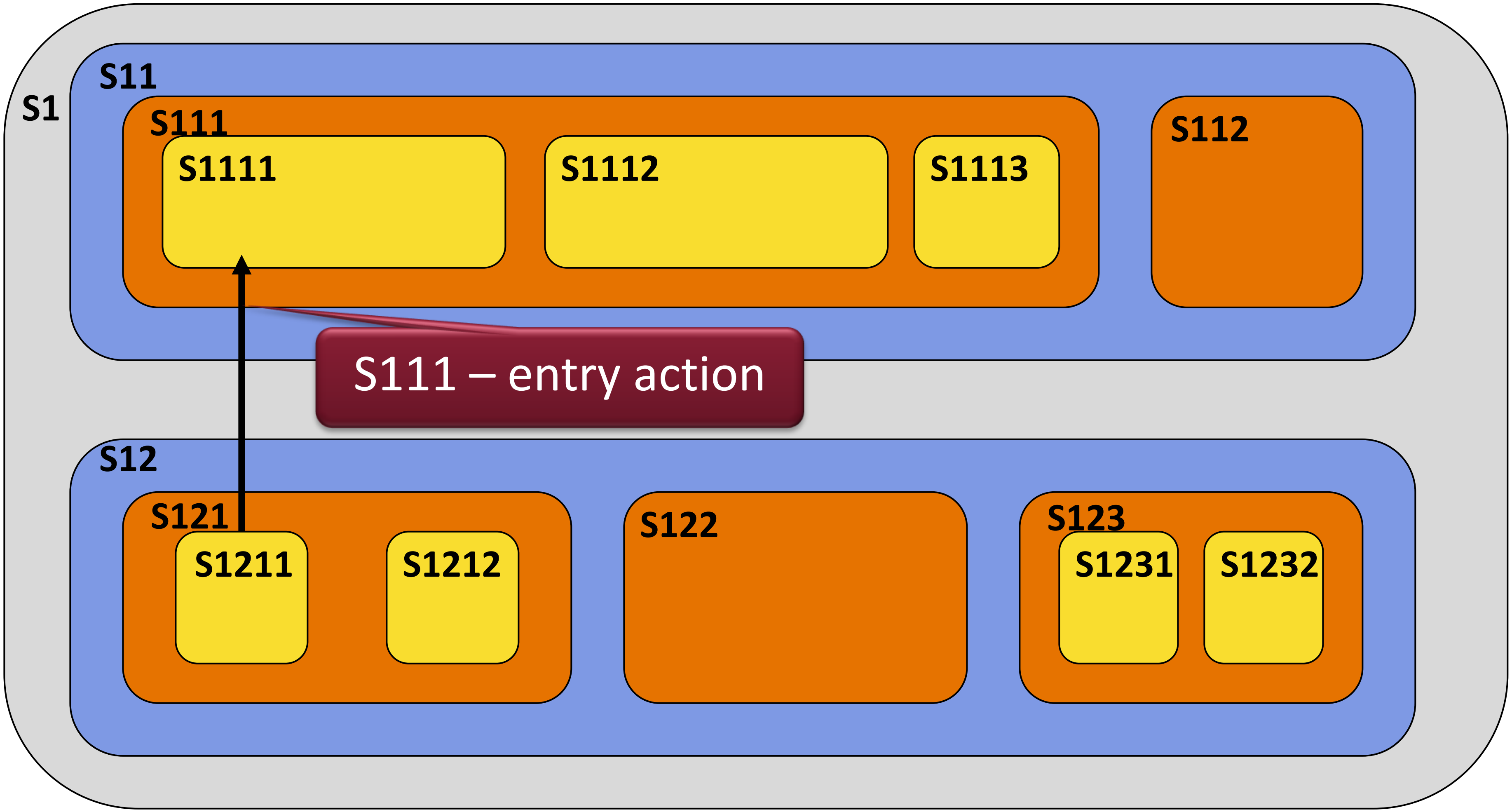
State transition example



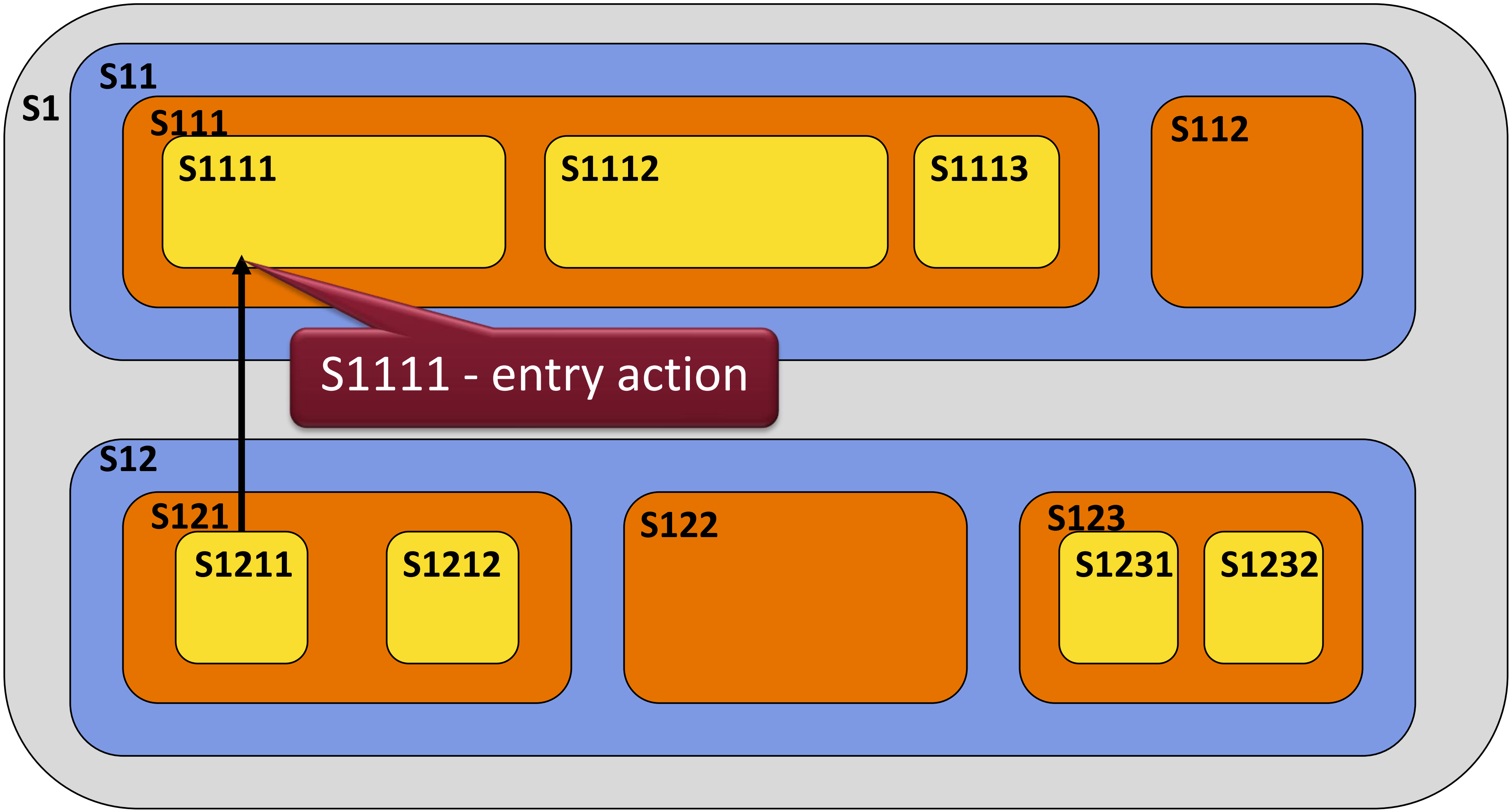
State transition example



State transition example

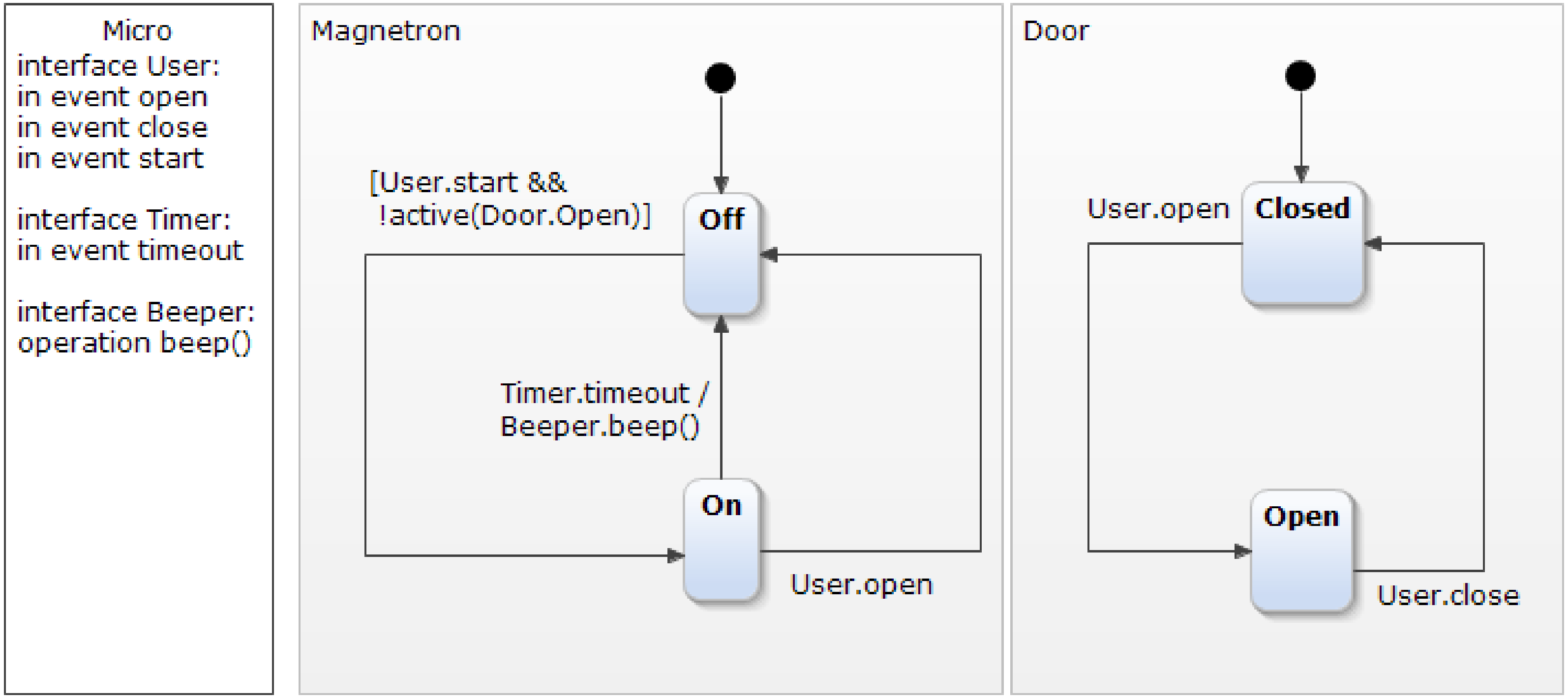


State transition example



Yakindu Statechart Tools

- Example tool support: Yakindu
  - Hierarchical state chart language



Yakindu Statechart Tools

- Java/C++ code generation from statechart
  - *Magnetron* switches to state *On* (simplified)

```
/* The reactions of state On. */
private void reactMagnetron_On() {
    if (sCITimer.timeout) {
        sCIBeeper.operationCallback.beep();
        stateVector[0] = State.magnetron_Off;
    } else {
        if (sCIUser.open) {
            stateVector[0] = State.magnetron_Off;
        }
    }
}
```

Summary

- Effective technique to model certain dynamic systems
- Hierarchic refinement allows iterative development
- Already used in many application domains
  - Avionics, automotive, …