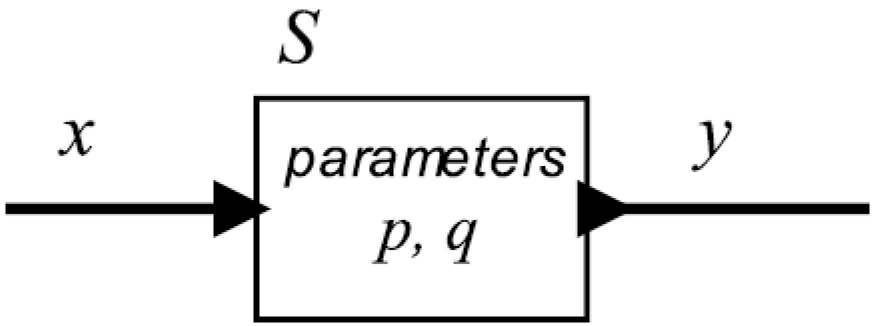
Embedded Systems

REVIEW: Actor models

- A *system* is a function that accepts an input *signal* and yields an output signal.



- The domain and range of the system function are sets of signals, which themselves are functions.

$$x\colon \mathbb{R} \to \mathbb{R}, \qquad y\colon \mathbb{R} \to \mathbb{R}$$

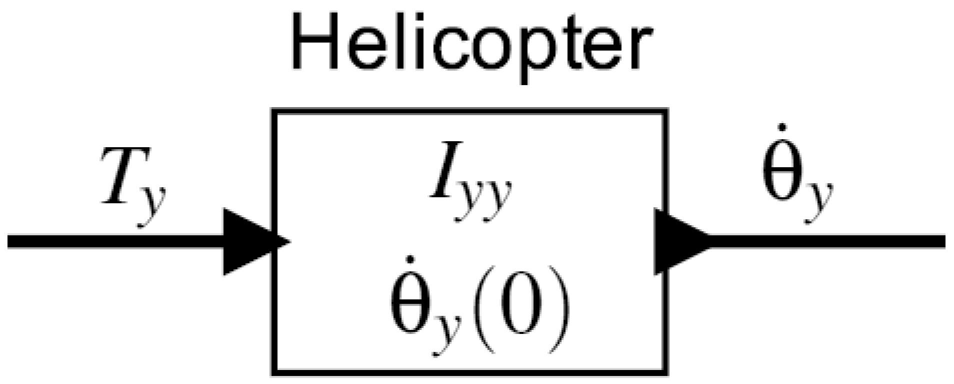
$$S\colon X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

- Parameters may affect the definition of the function *S*.

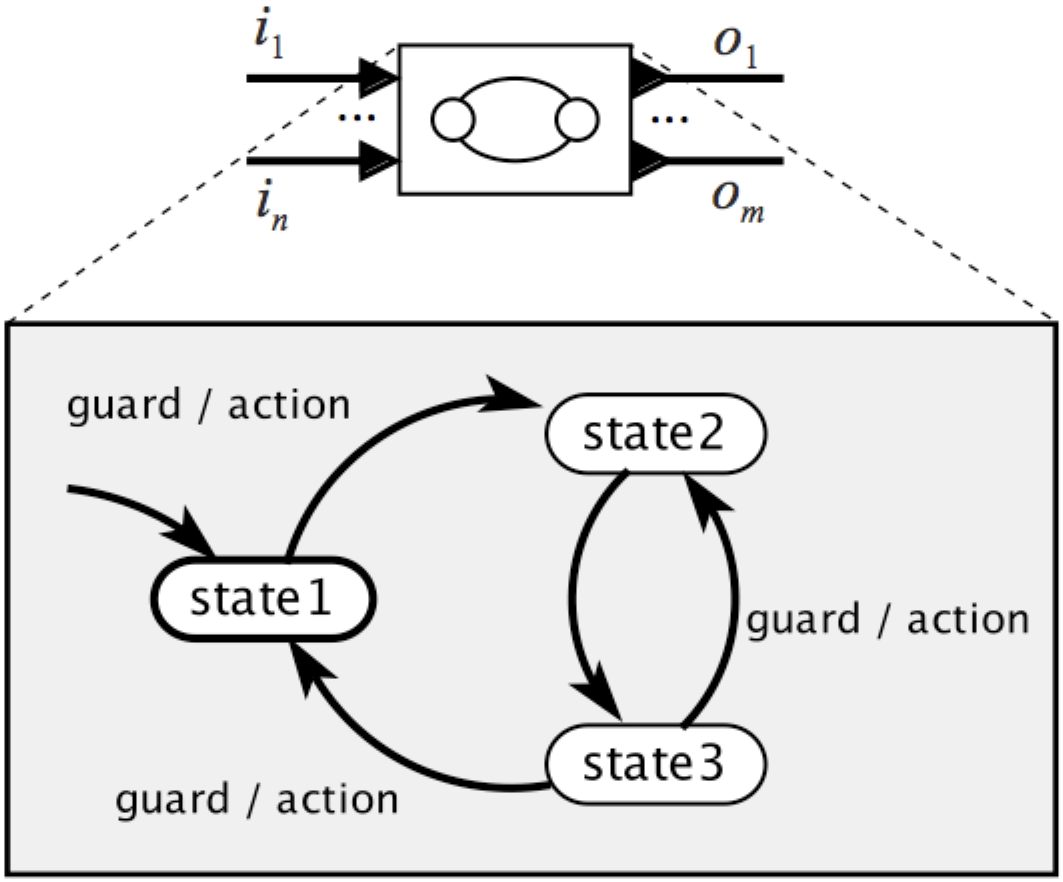
REVIEW: Actor models of continuous-time systems

Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the *y* axis.



$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

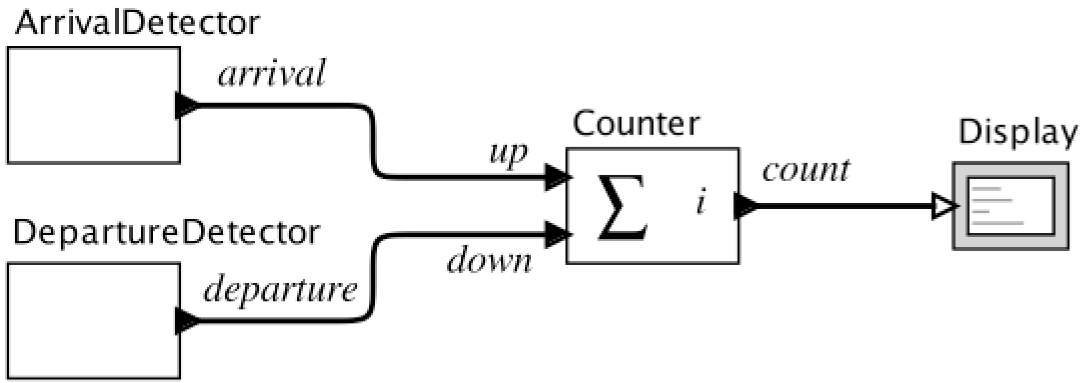
Actor Model of an FSM



REVIEW: Discrete Systems

Lee/Seshia, Chapter 3

- Example: count the number of cars that enter and leave a parking garage:



- Pure signal: $up: \mathbb{R} \to \{absent, present\}$
- Discrete actor:

$$Counter: (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$$
$$P = \{up, down\}$$

Discrete Signals

Let *e* be a signal $\mathbb{R} \rightarrow \{absent\} \cup X$
where *X* is any set of values.

Let $T = \{t \in \mathbb{R} \;:\; e(t) \neq absent\}$

Then *e* is **discrete** iff there exists a one-to-one function

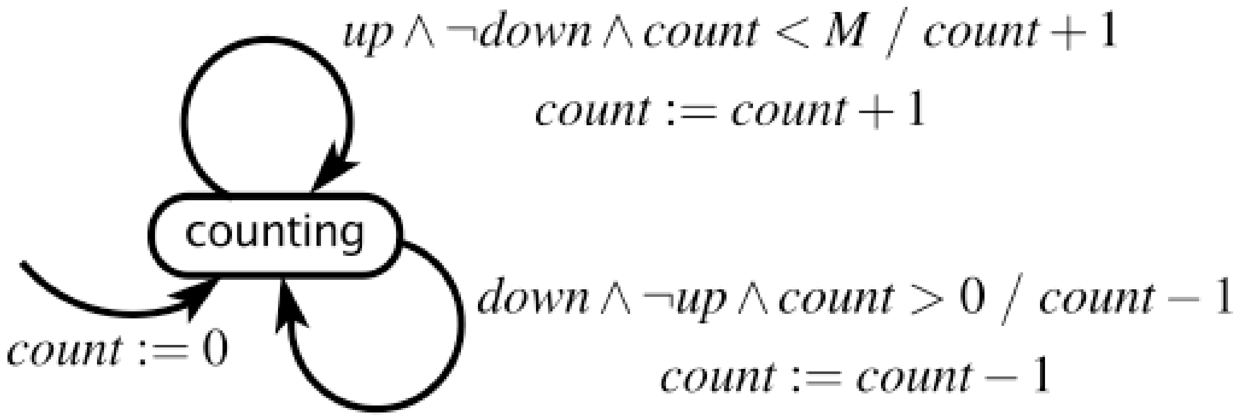
$$f : T \rightarrow \mathbb{N}$$

that is order-preserving, i.e., for all $t_1 \leq t_2$, $f(t_1) \leq f(t_2)$.

REVIEW: Extended State Machines

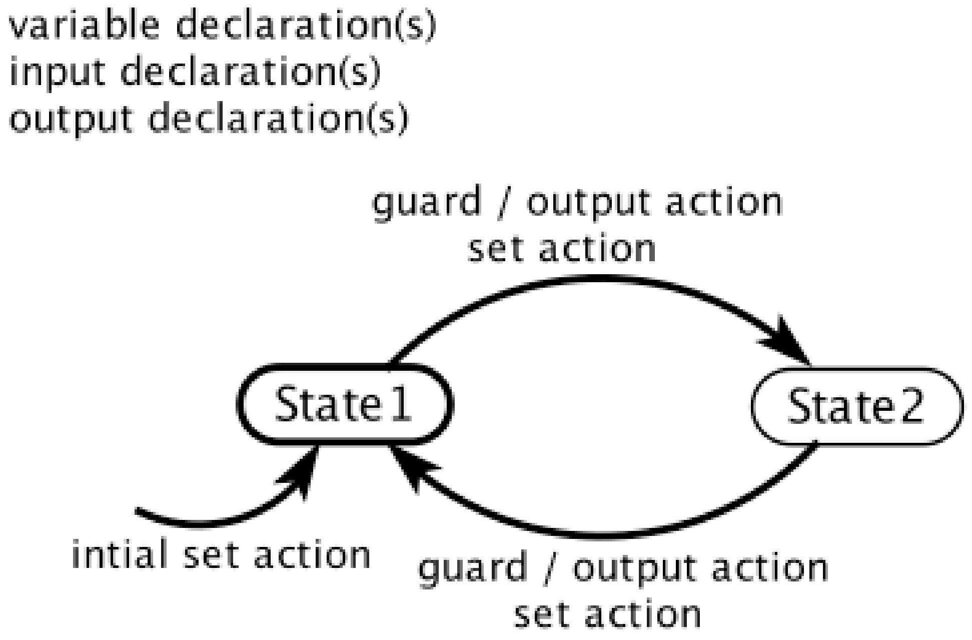
Extended state machines augment the FSM model with *variables* that may be read or written. E.g.:

variable: $count \in \{0, \cdots, M\}$
inputs: $up, down \in \{present, absent\}$
output $\in \{0, \cdots, M\}$

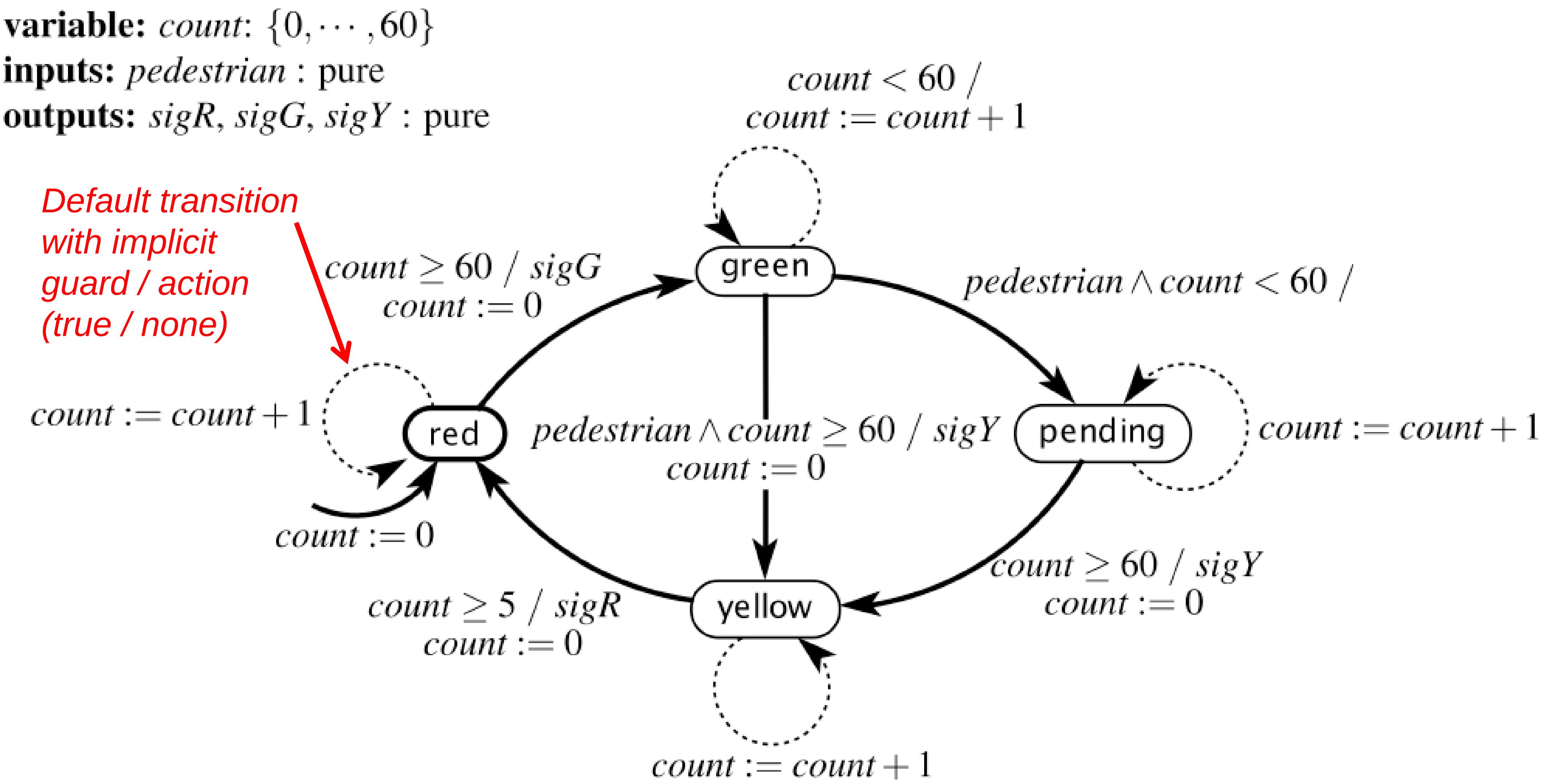

$up \land \neg down \land count < M \ / \ count + 1$
$count := count + 1$
counting
$count := 0$
$down \land \neg up \land count > 0 \ / \ count - 1$
$count := count - 1$

General Notation for Extended State Machines

We make explicit declarations of variables, inputs, and outputs to help distinguish the three.



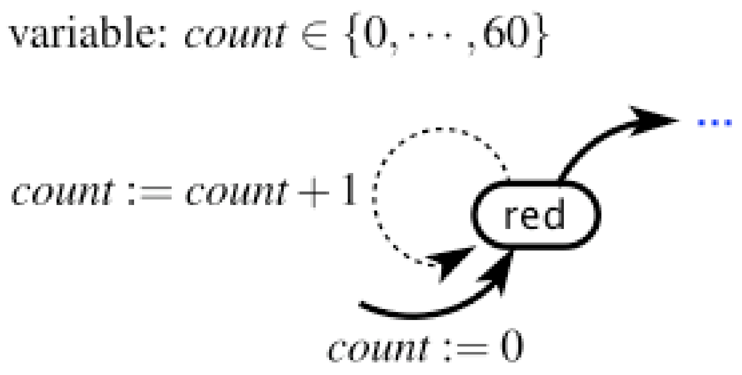
Extended state machine model of a traffic light controller at a pedestrian crossing:



variable: $count$: $\{0, \cdots, 60\}$
inputs: $pedestrian$: pure
outputs: $sigR$, $sigG$, $sigY$: pure
Default transition with implicit guard / action (true / none)
$count < 60$ / $count := count + 1$
$count \geq 60$ / $sigG$ $count := 0$
$pedestrian \wedge count < 60$ /
$count := count + 1$
$pedestrian \wedge count \geq 60$ / $sigY$ $count := 0$
$count := count + 1$
$count := 0$
$count \geq 60$ / $sigY$ $count := 0$
$count \geq 5$ / $sigR$ $count := 0$
$count := count + 1$
green
red
pending
yellow

- This model assumes one reaction per second
- (a *time-triggered* model)

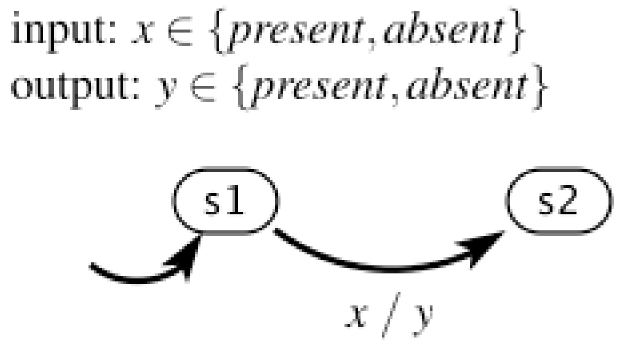
When does a reaction occur?



When a reaction occurs is not specified in the state machine itself. It is up to the environment.

This traffic light controller design assumes one reaction per second. This is a *time-triggered model*.

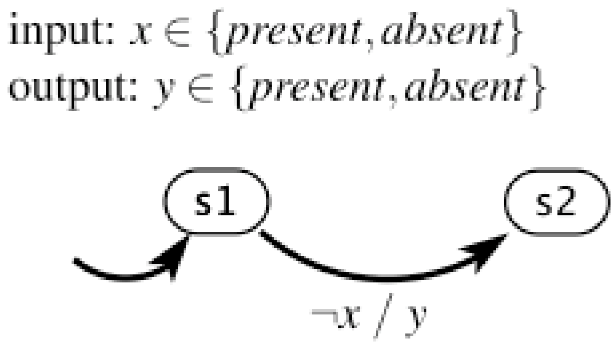
When does a reaction occur?


input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$
s1
s2
x / y

- Suppose all inputs are discrete and a reaction occurs when any input is present. Then the above transition will be taken whenever the current state is s1 and *x* is present.

- This is an *event-triggered model*.

When does a reaction occur?


input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$
s1
s2
$\neg x \;/\; y$

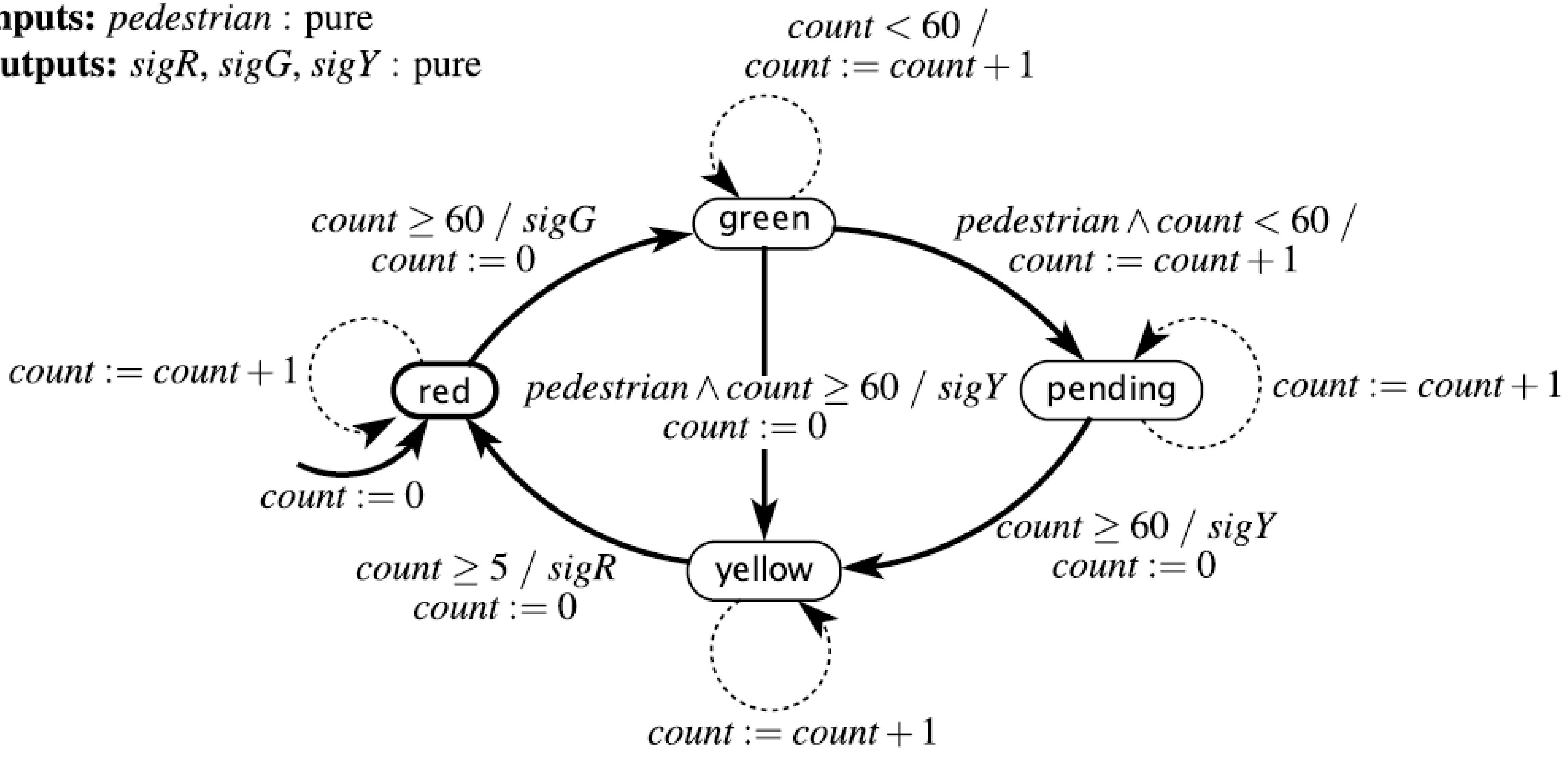
- Suppose *x* and *y* are discrete and pure signals.

When does the transition occur?

Answer: when the environment triggers a reaction and x is absent. If this is a (complete) event-triggered model, then the transition will never be taken because the reaction will only occur when x is present!

Traffic Light Controller

variable: $count$: $\{0, \cdots, 60\}$
inputs: $pedestrian$: pure
outputs: $sigR$, $sigG$, $sigY$: pure



$count < 60$ / $count := count + 1$
green
$count \geq 60$ / $sigG$ $count := 0$
$pedestrian \wedge count < 60$ / $count := count + 1$
$count := count + 1$
red
$pedestrian \wedge count \geq 60$ / $sigY$ $count := 0$
pending
$count := count + 1$
$count := 0$
$count \geq 60$ / $sigY$ $count := 0$
$count \geq 5$ / $sigR$ $count := 0$
yellow
$count := count + 1$

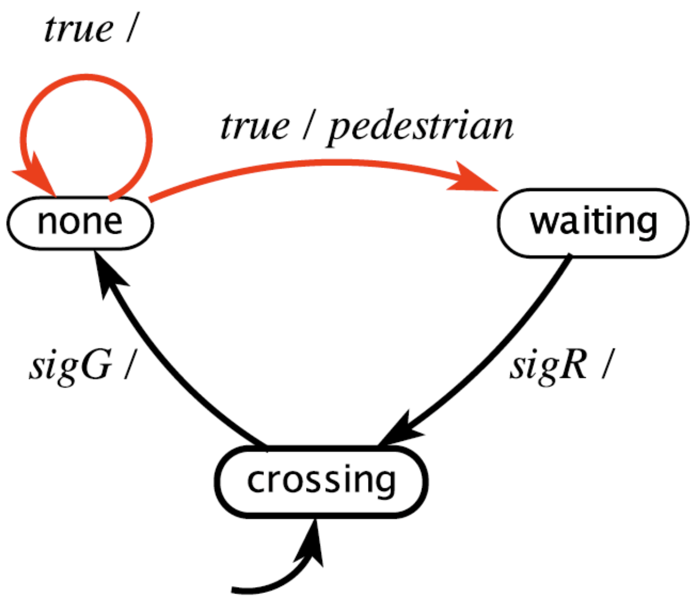
Definitions

- **Stuttering transition**: Implicit default transition that is enabled when inputs are absent and that produces absent outputs.
- **Receptiveness**: For any input values, some transition is enabled. Our structure together with the implicit default transition ensures that our FSMs are receptive.
- **Determinism**: In every state, for all input values, exactly one (possibly implicit) transition is enabled.

Example: Nondeterministic FSM

- Nondeterministic model of pedestrians arriving at a crosswalk:

inputs: $sigR, sigG, sigY$: pure
outputs: $pedestrian$: pure



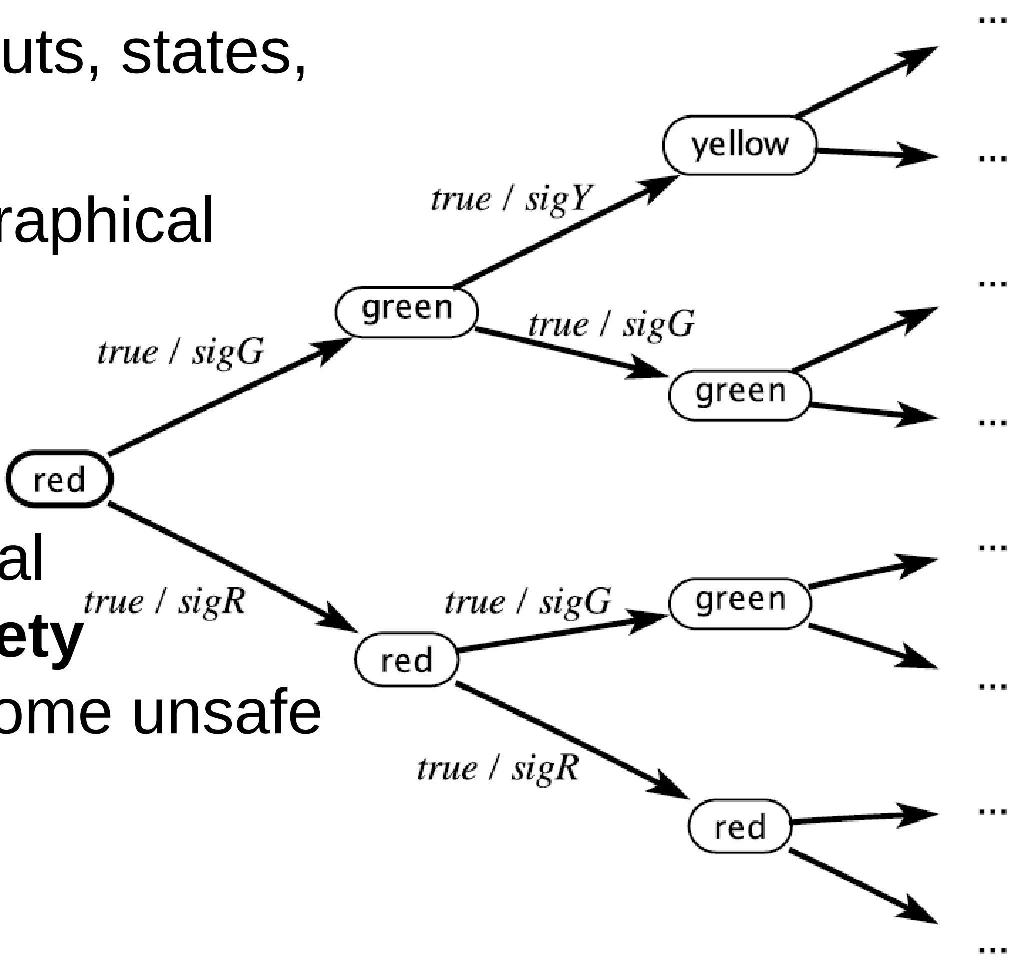
- Formally, the update function is replaced by a function

$$possibleUpdates : States \times Inputs \rightarrow 2^{States \times Outputs}$$

Behaviors and Traces

- FSM **behavior** is a sequence of (non-stuttering) steps.
- A **trace** is the record of inputs, states, and outputs in a behavior.
- A **computation tree** is a graphical representation of all possible traces.

- FSMs are suitable for formal analysis. For example, **safety** analysis might show that some unsafe state is not reachable.



Uses of nondeterminism

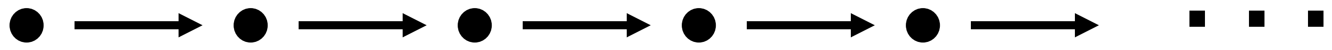
1. Modeling *unknown* aspects of the environment or system

2. Hiding detail in a *specification* of the system

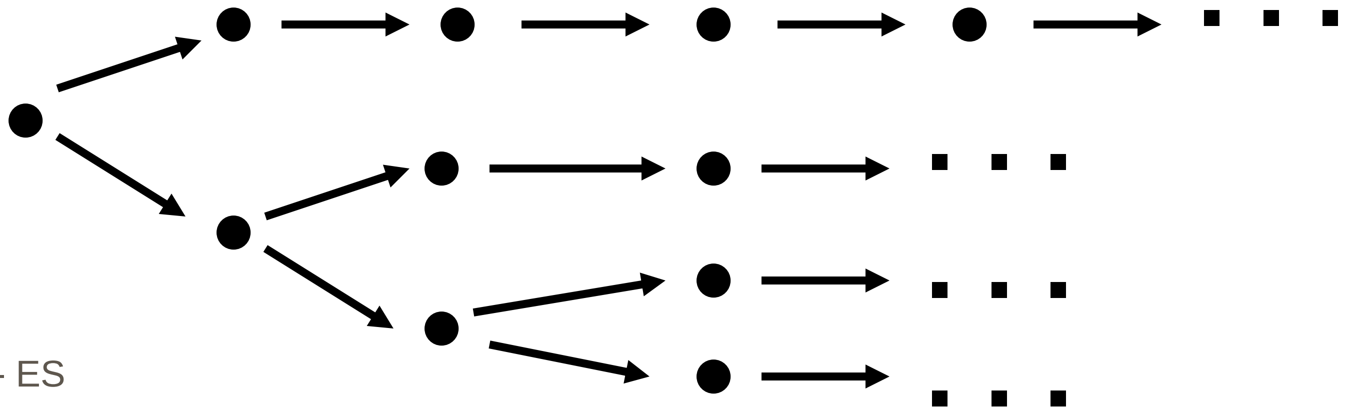
Non-deterministic Behavior: Tree of Computations

- For a fixed input sequence:
  - A deterministic system exhibits a single behavior
  - A non-deterministic system exhibits a **set of behaviors**

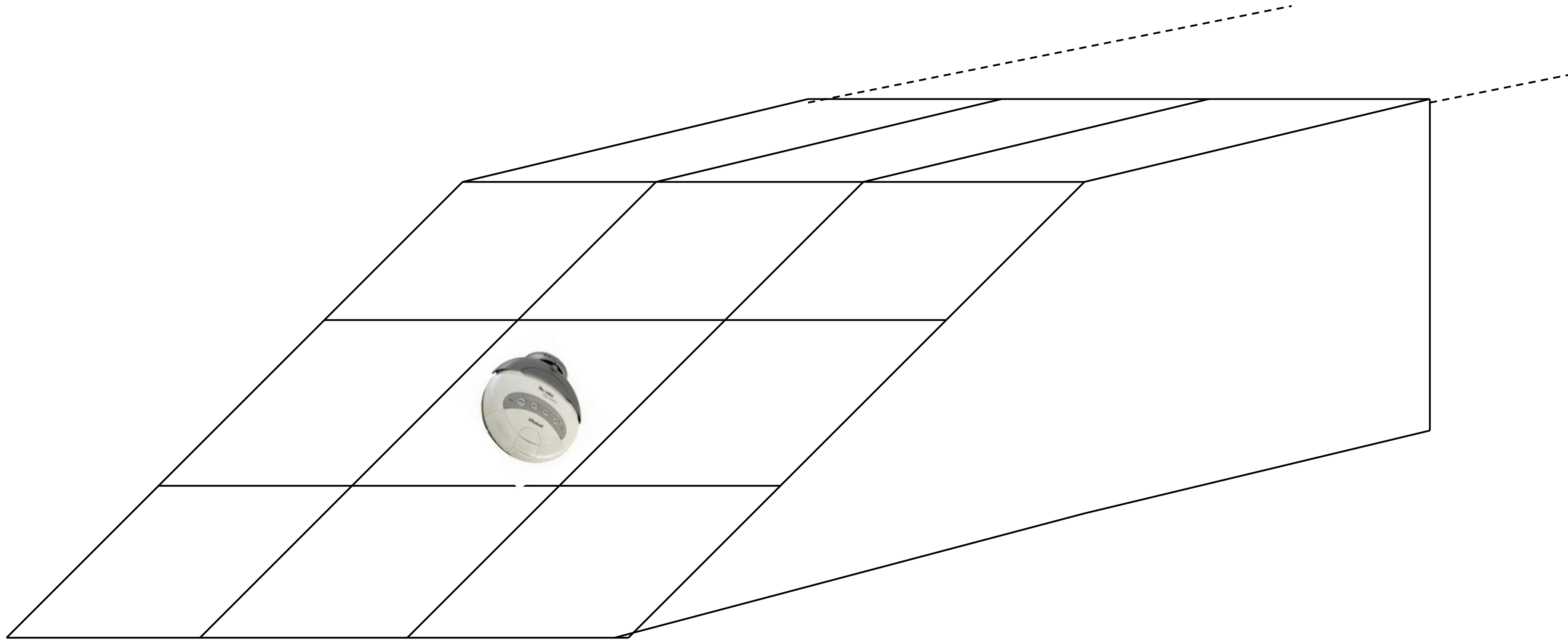
Deterministic FSM behavior for a particular input sequence:



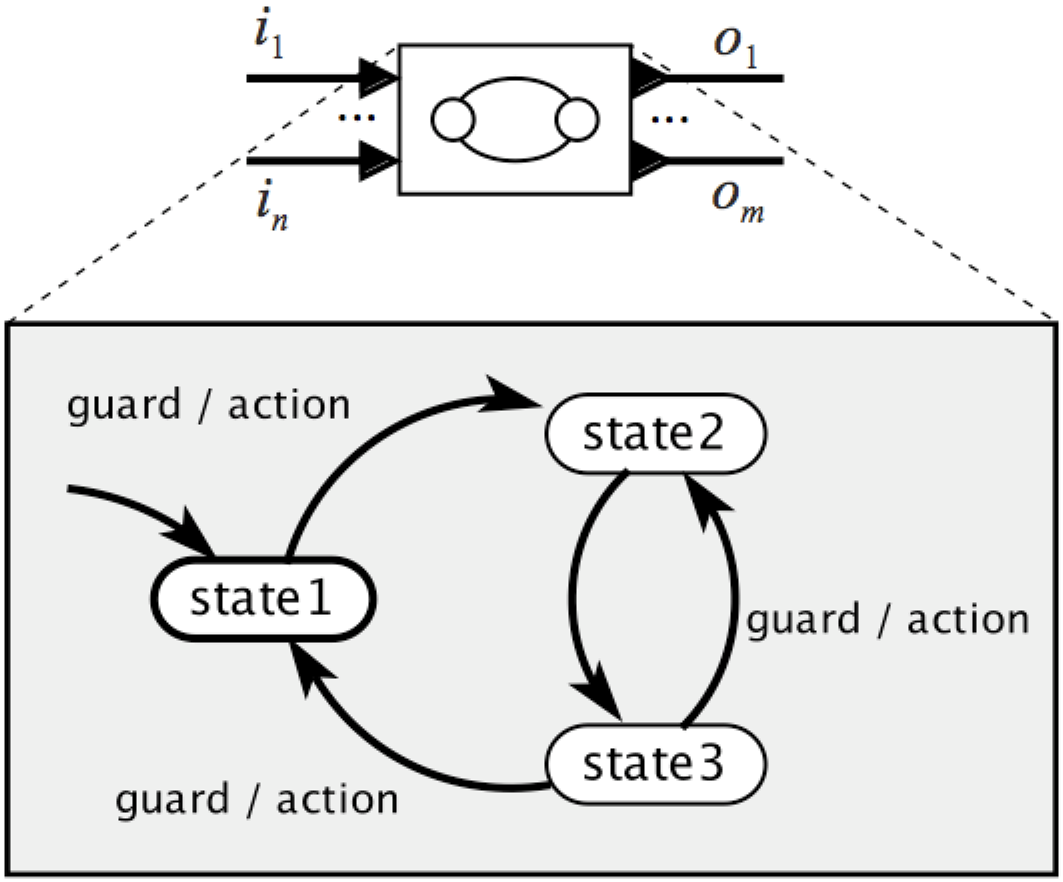
Non-deterministic FSM behavior for an input sequence:



It is sometimes useful to even model continuous systems as FSMs by discretizing their state space.



Actor Model of an FSM



*This model enables **composition** of state machines.*

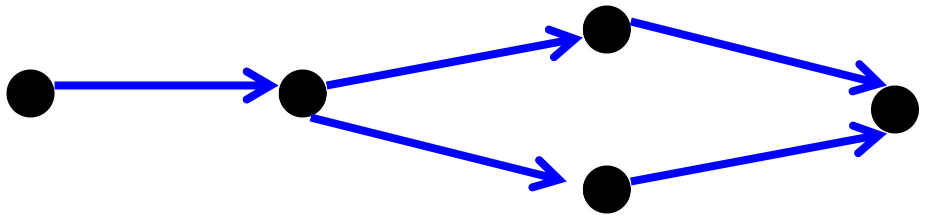
Applications of FSMs

FSMs provide:

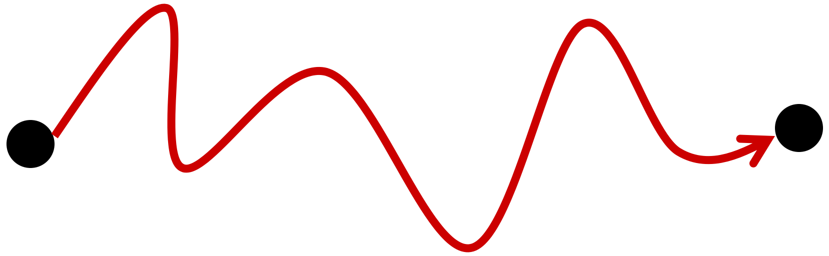
1. A way to represent the system for:
    - Mathematical analysis
    - So that a computer program can manipulate it
2. A way to model the environment of a system.
3. A way to represent what the system *must* do and *must not* do – its specification.
4. A way to check whether the system satisfies its specification in its operating environment.

Hybrid Automata

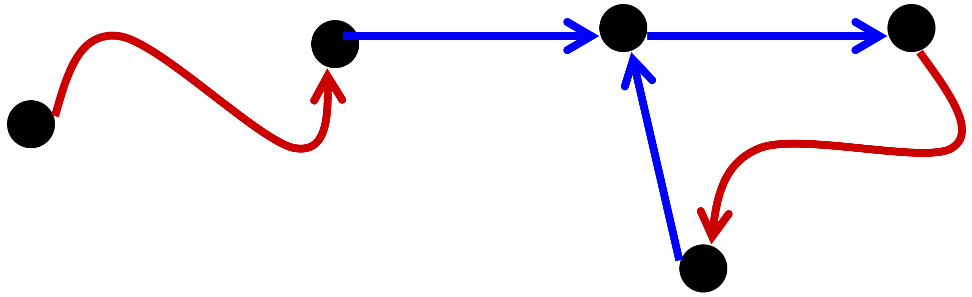
Discrete System (FSM)

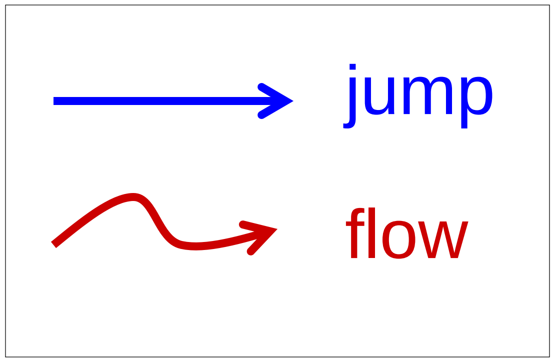


Continuous System



Hybrid System

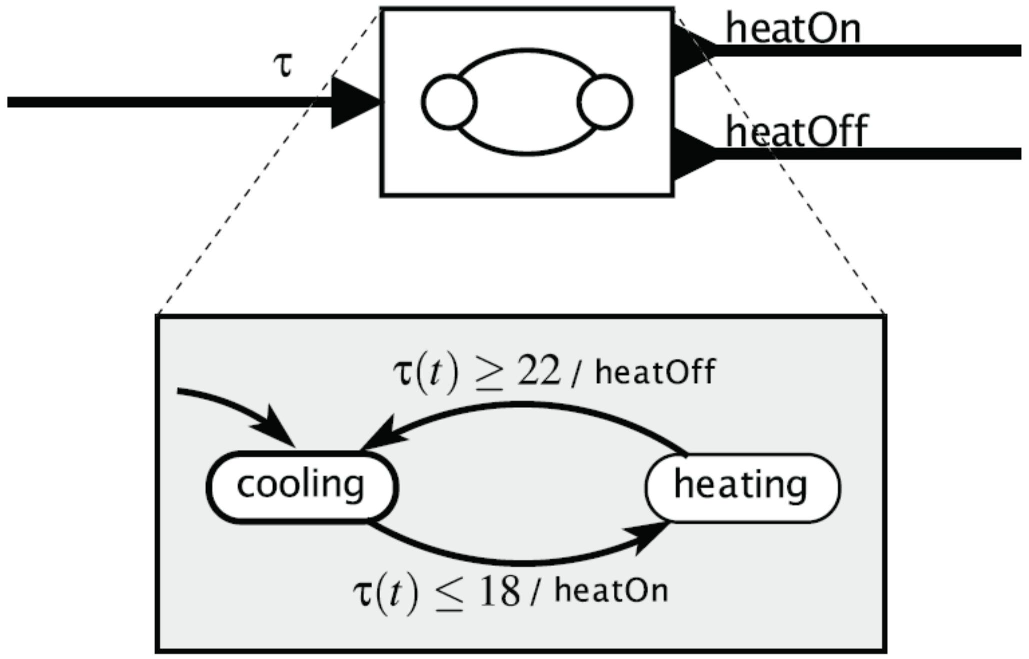




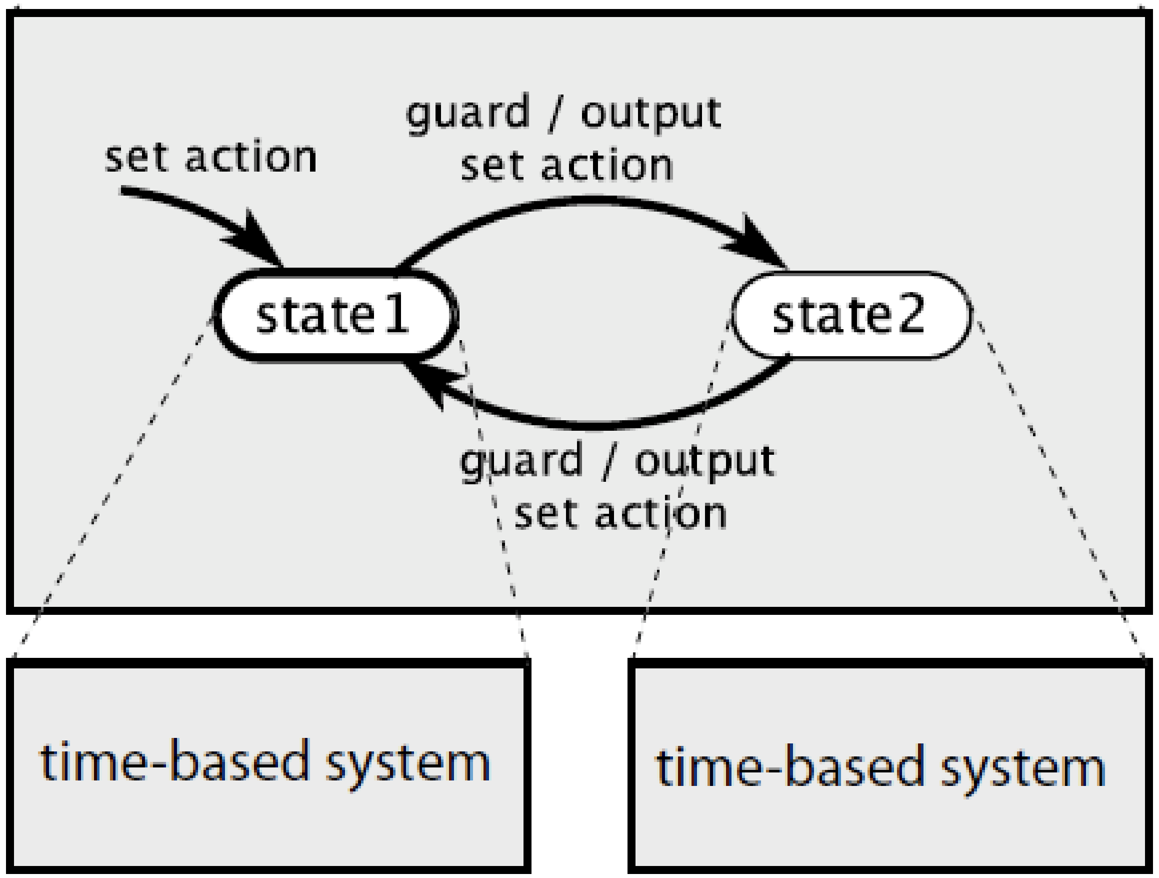
Where do Hybrid Systems arise?

- Digital controller of physical “plant”
  - thermostat
  - intelligent cruise control in cars
  - aircraft auto pilot
- Phased operation of natural phenomena
  - bouncing ball
  - biological cell growth
- Multi-agent systems
  - ground and air transportation systems
  - interacting robots

FSM with continous-time input



An alternative to FSMs that is explicit about the passage of time: *Timed automata*, a special case of hybrid systems.



Timed automata

- A clock is a continous-time signal *s* with constant rate

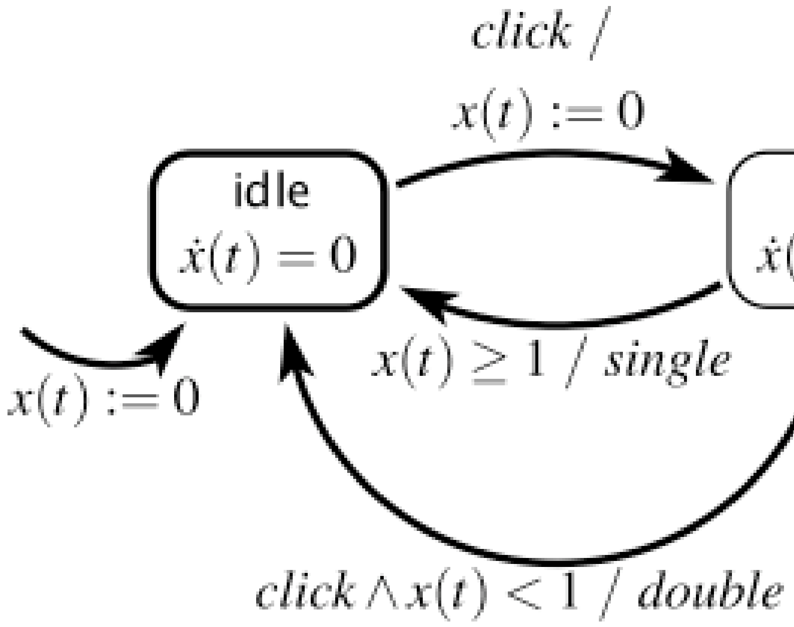
$$\forall\, t \in T_m, \quad \dot{s}(t) = a$$

  while the system is in some mode *m*

- Timed automata are FSMs extended with clocks.

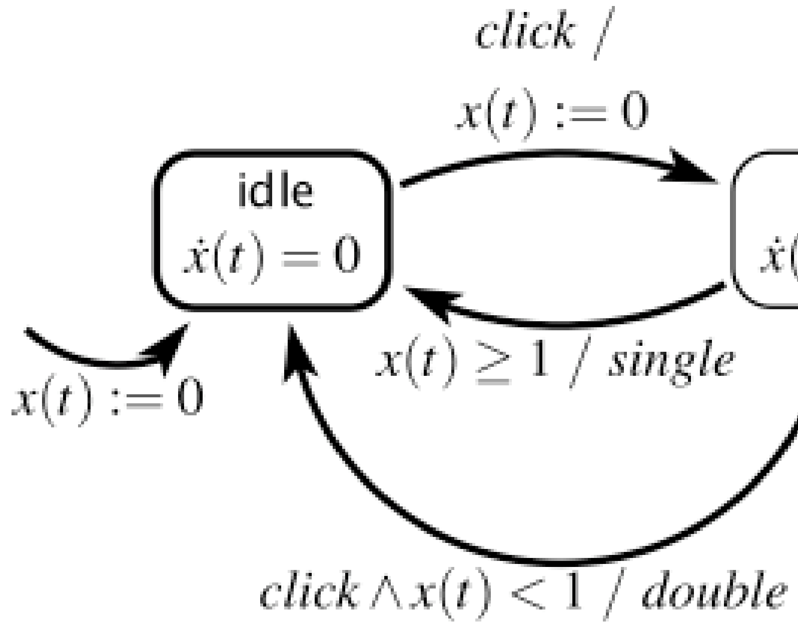
Example: Mouse Double Click Detector

continuous variable: $x(t) \in \mathbb{R}$
inputs: $click \in \{present, absent\}$
outputs: $single, double \in \{present, absent\}$



Example: Mouse Double Click Detector

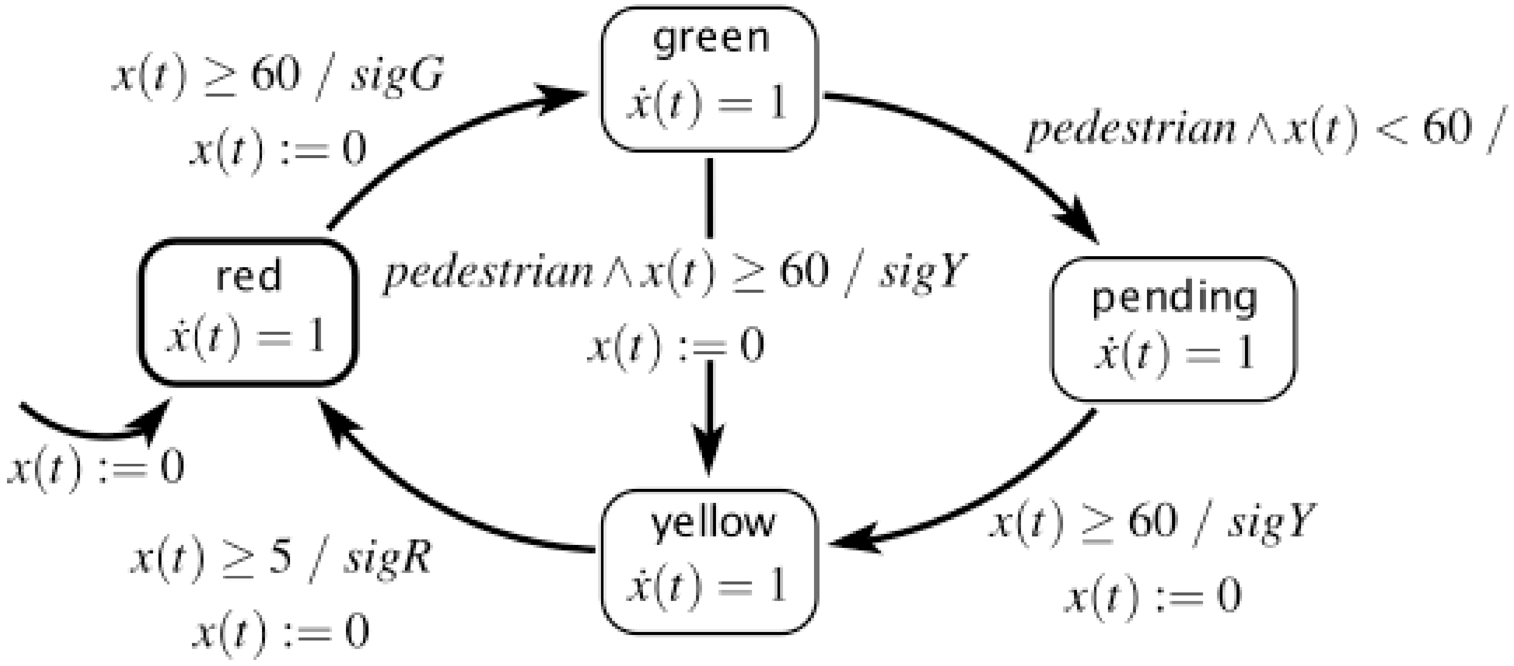
continuous variable: $x(t) \in \mathbb{R}$
inputs: $click \in \{present, absent\}$
outputs: $single, double \in \{present, absent\}$



How many states does this automaton have?

Timed automaton model of a traffic light controller

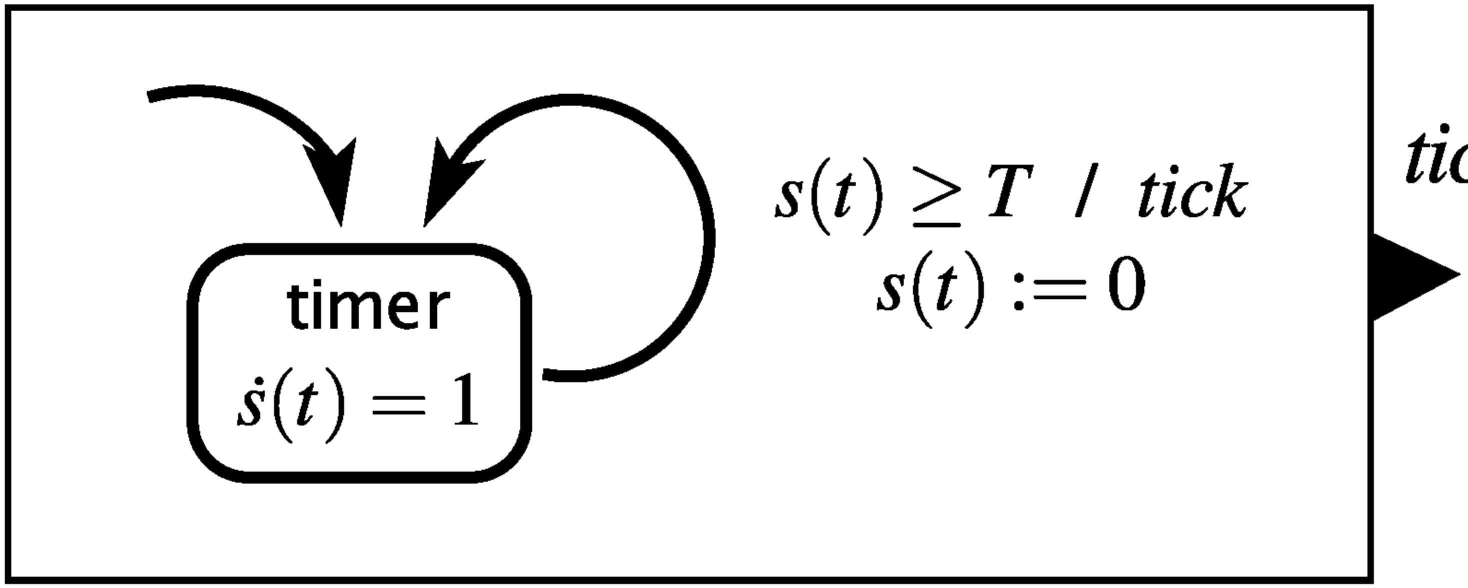
continuous variable: $x(t) : \mathbb{R}$
inputs: $pedestrian$: pure
outputs: $sigR, sigG, sigY$: pure



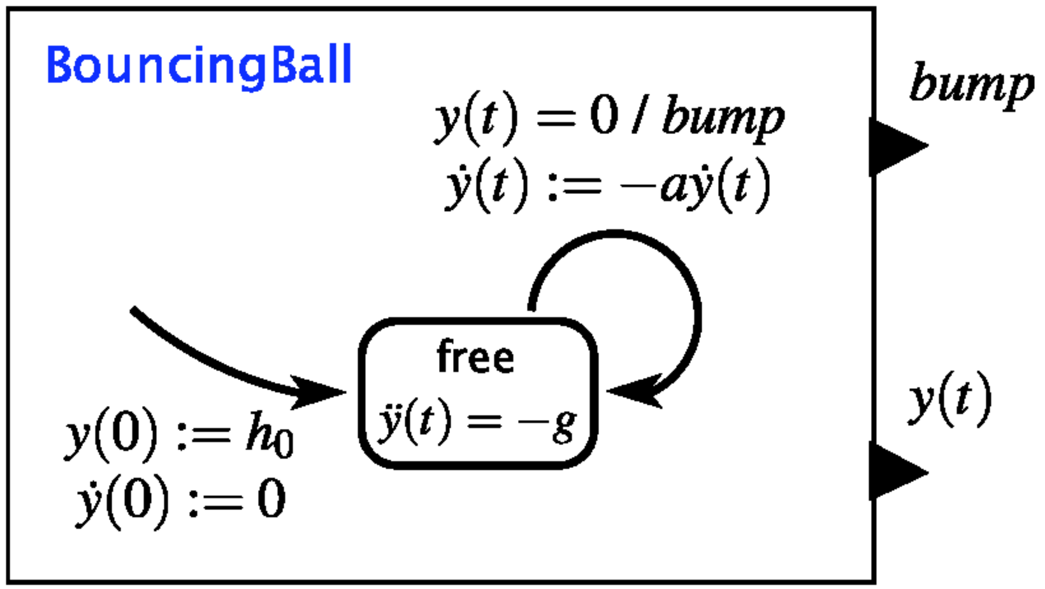
This light remains green at least 60 seconds, and then turns yellow if a pedestrian has requested a crossing. It then remains red for 60 seconds.

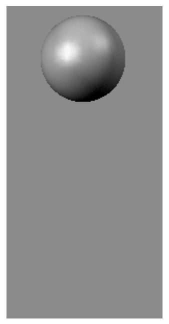
Example: “Tick” Generator (Timer)

How would you model a timer that generates a ‘tick’ each time T time units elapses?



Hybrid Automaton for Bouncing Ball



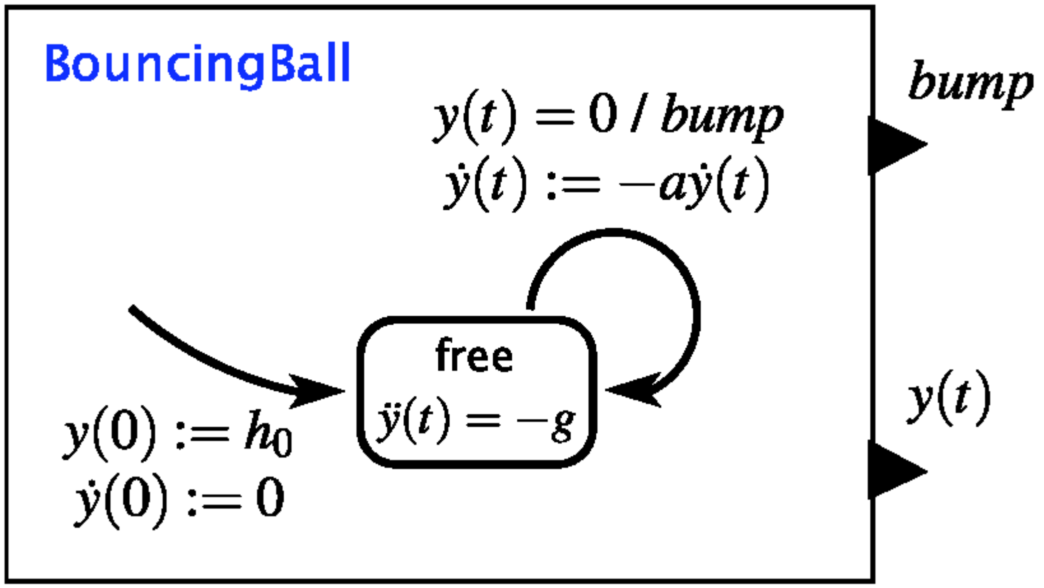


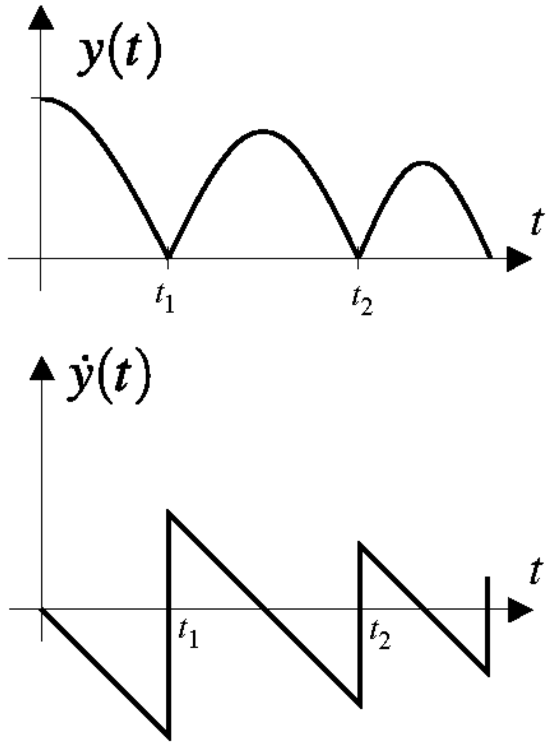
y – vertical distance from ground (position)

a – coefficient of restitution, $0 \cdot a \cdot 1$

If you plotted y(t), what would it look like?

Hybrid Automaton for Bouncing Ball

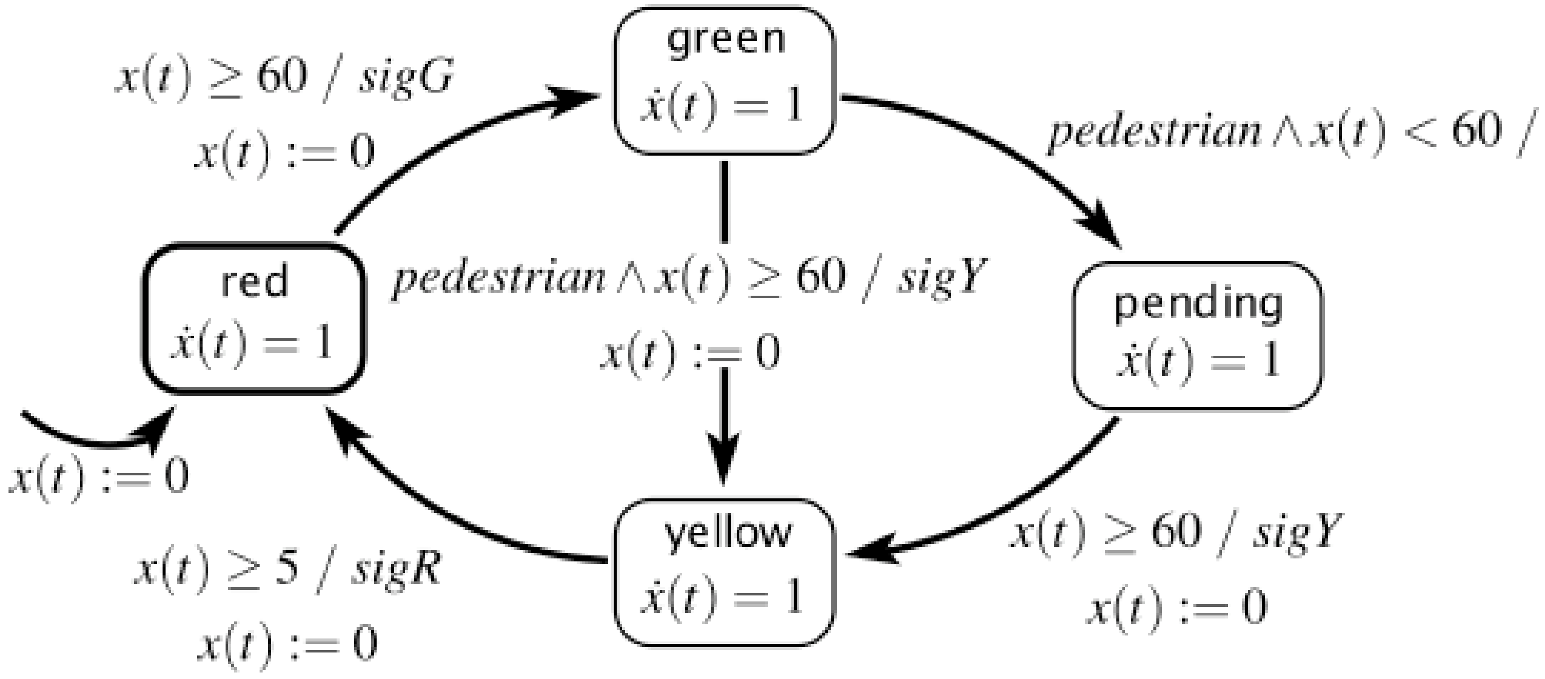




y – vertical distance from ground (position)
a – coefficient of restitution, $0 \cdot a \cdot 1$

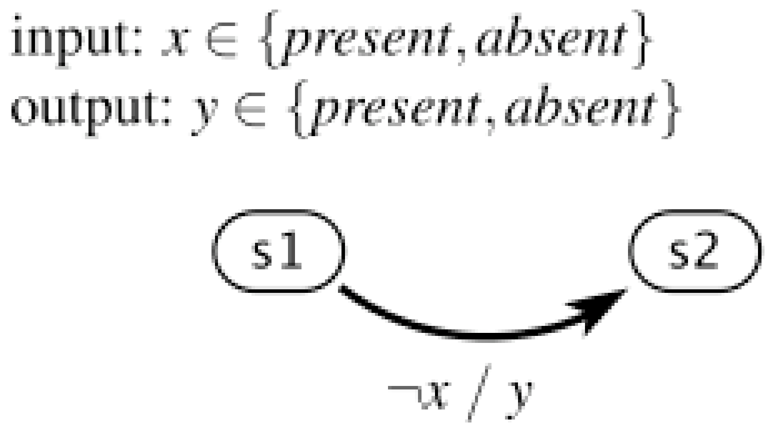
When do reactions occur in a hybrid automaton?

continuous variable: $x(t)\colon \mathbb{R}$
inputs: $pedestrian$: pure
outputs: $sigR$, $sigG$, $sigY$: pure



- Reactions are occurring continually, with the continuous state variable *x* being continually updated.

When do reactions occur in a hybrid automaton?


input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$

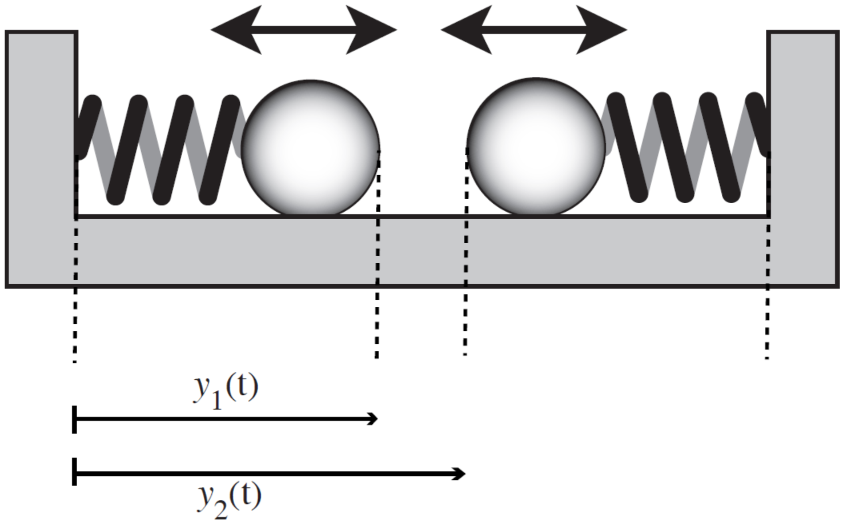
- Suppose *x* and *y* are discrete and pure signals.

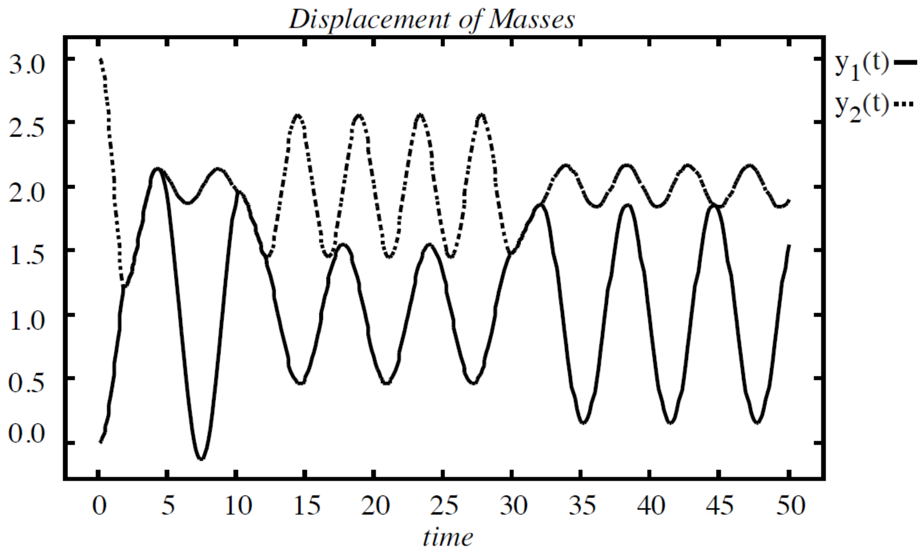
When does the transition occur?

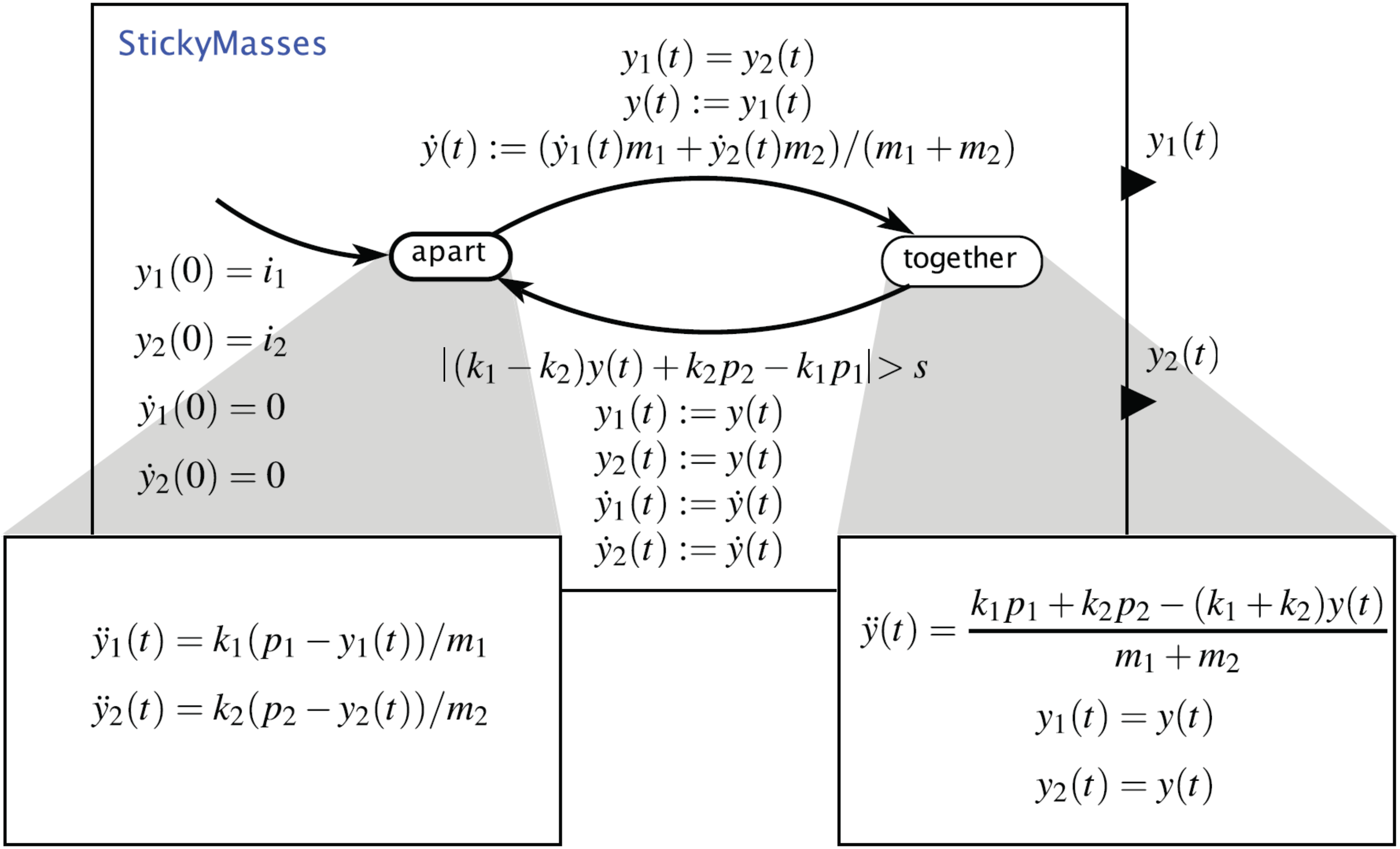
Answer: at the earliest time t when x is absent after entering s1. This will always be the same time when s1 is entered. Why?

If x is absent when s1 is entered, then the transition is taken then. If x is present when s1 is entered, then it will be absent at a time infinitesimally larger. How to model this rigorously?

Sticky Masses






StickyMasses
apart
together
$y_1(t) = y_2(t)$
$y(t) := y_1(t)$
$\dot{y}(t) := (\dot{y}_1(t)m_1 + \dot{y}_2(t)m_2)/(m_1+m_2)$
$y_1(0) = i_1$
$y_2(0) = i_2$
$\dot{y}_1(0) = 0$
$\dot{y}_2(0) = 0$
$|(k_1-k_2)y(t) + k_2p_2 - k_1p_1| > s$
$y_1(t) := y(t)$
$y_2(t) := y(t)$
$\dot{y}_1(t) := \dot{y}(t)$
$\dot{y}_2(t) := \dot{y}(t)$
$y_1(t)$
$y_2(t)$
$\ddot{y}_1(t) = k_1(p_1 - y_1(t))/m_1$
$\ddot{y}_2(t) = k_2(p_2 - y_2(t))/m_2$
$\ddot{y}(t) = \frac{k_1p_1 + k_2p_2 - (k_1+k_2)y(t)}{m_1+m_2}$
$y_1(t) = y(t)$
$y_2(t) = y(t)$

Example: Newton’s Cradle

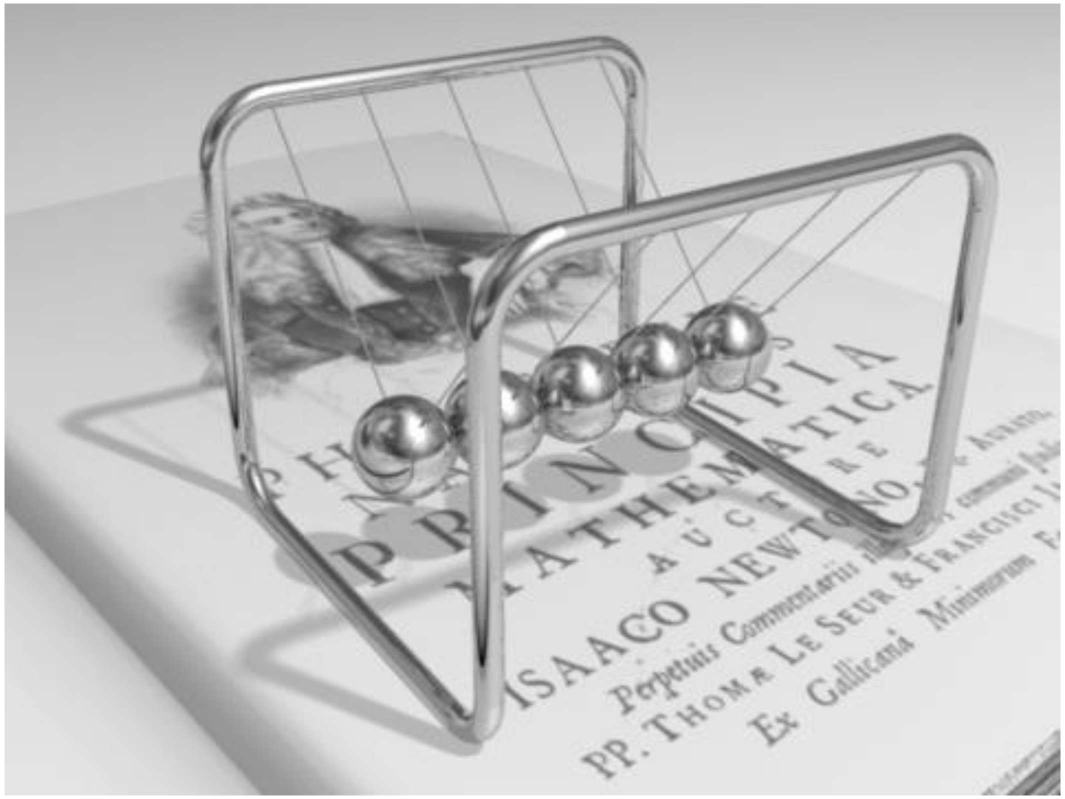

PHILOSOPHIÆ NATURALIS
PRINCIPIA
MATHEMATICA
AUCTORE
ISAACO NEWTONO
Perpetuis Commentariis illustrata
PP. THOMÆ LE SEUR & FRANCISCI JACQUIER
Ex Gallicanâ Minimorum Familiâ

Image src: Wikipedia Commons

A middle ball does not move, so its momentum must be 0. But the momentum of the first ball is transferred somehow to the fifth. So there is an instant at which it is non-zero!

Super-Dense Time

- We will solve this problem by modeling signals as functions of super-dense time:

$$x : \mathbb{R} \times \mathbb{N} \to \{present, absent\}$$

- In this way, *x* can be present and absent at the same time, with a well-defined order between its presence and absence.

Why science teachers should not be given playground duty.

Marwedel, Sec. 2.4.2

StateCharts – Additional features compared to FSMs

- Hierarchy
- Concurrency

StateCharts

- Statecharts introduced in
  Harel: “StateCharts: A visual formalism for complex systems”. Science of Computer Programming, 1987.
- More detailed in
  Drusinsky and Harel: “Using statecharts for hardware desription and synthesis”, IEEE Trans. On Computer Design, 1989.
- Formal semantics in
  Harel, Naamad: “The statemate semantics of statecharts”, ACM Trans. Soft. Eng. Methods, 1996.
- *many variations of the semantics implemented in tools*

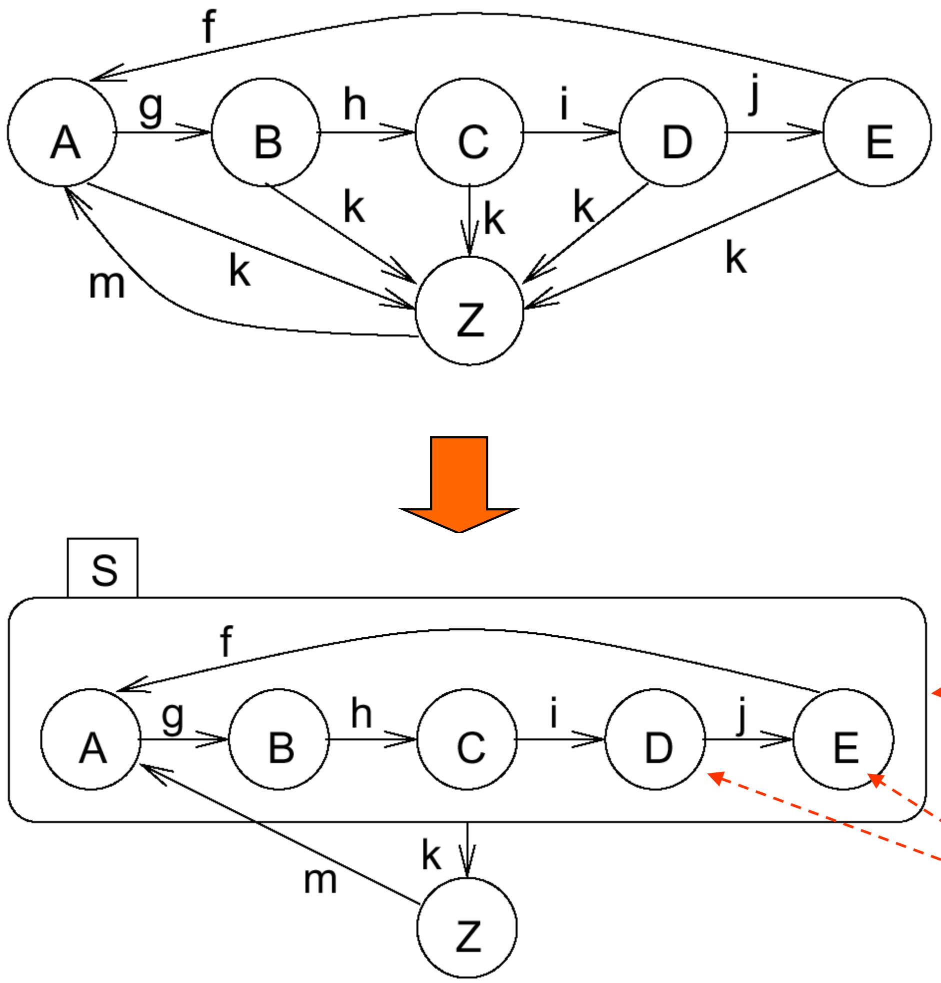
Non-deterministic transitions

Edge label (simple version):

$$\begin{tikzcd} A \arrow[r, "f/g"] & B \end{tikzcd}$$

- Transition from A to B iff event f is present.
- Effect of transition from A to B: Event g is produced.
- Events may be
  - External events (provided by the *environment*)
  - Internal events (produced by internal transitions)
- Produced events exist only for one step.

Introducing hierarchy



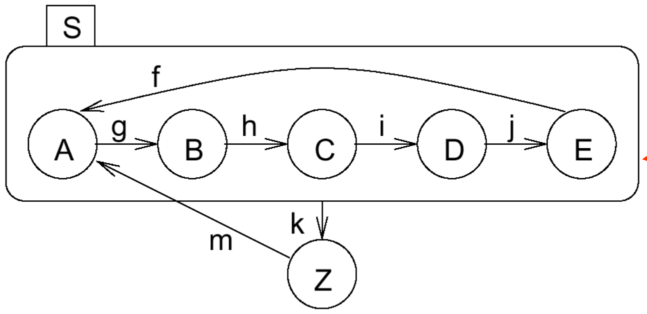
FSM will be **in** exactly one of the substates of S if S is **active** (either in A or in B or ..)

superstate

substates

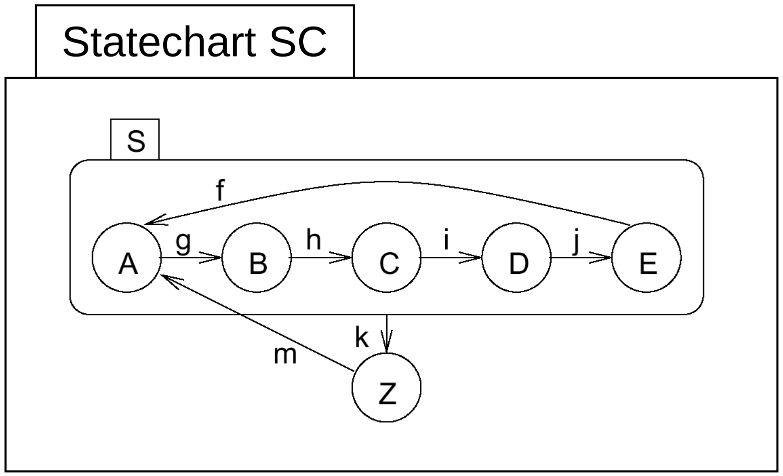
Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**
- States containing other states are called **super-states**.
- For each basic state s, the super-states containing s are called **ancestor states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.

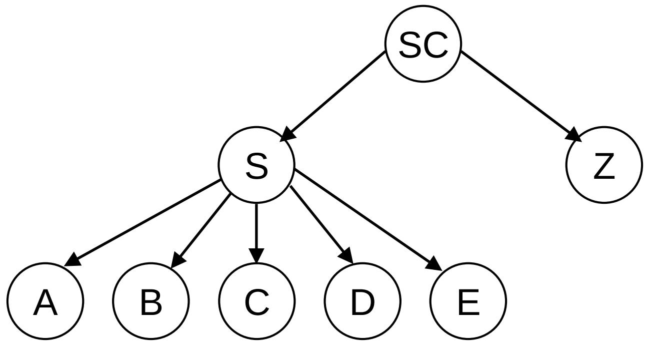


ancestor state of E

Hierarchy

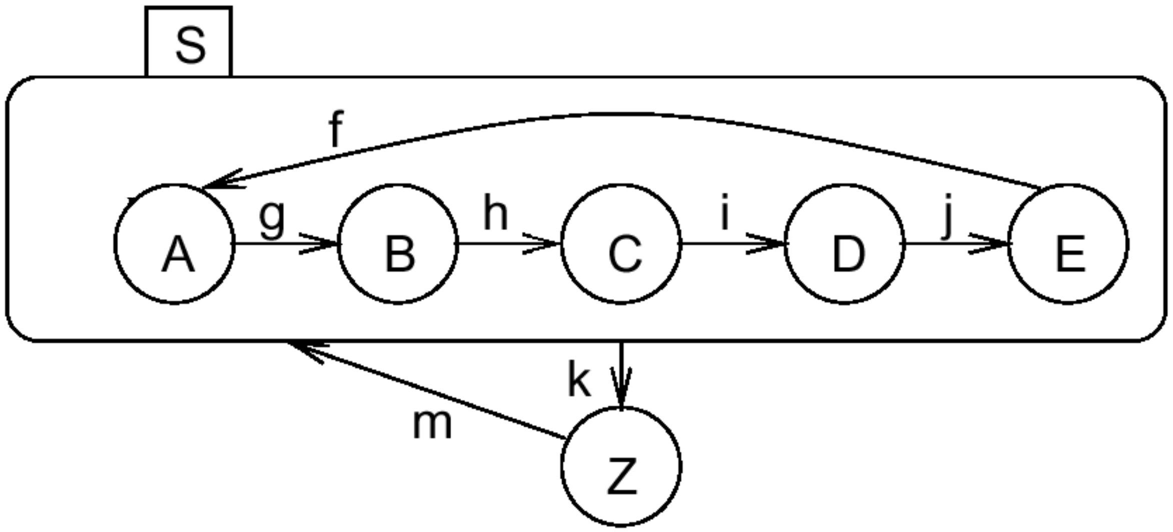


- Hierachy information may be represented by a hierarchy tree with basic states as leaves.



- Transitions between all levels of hierarchy possible!
- When a basic state is active, then all its ancestor states are active, too.

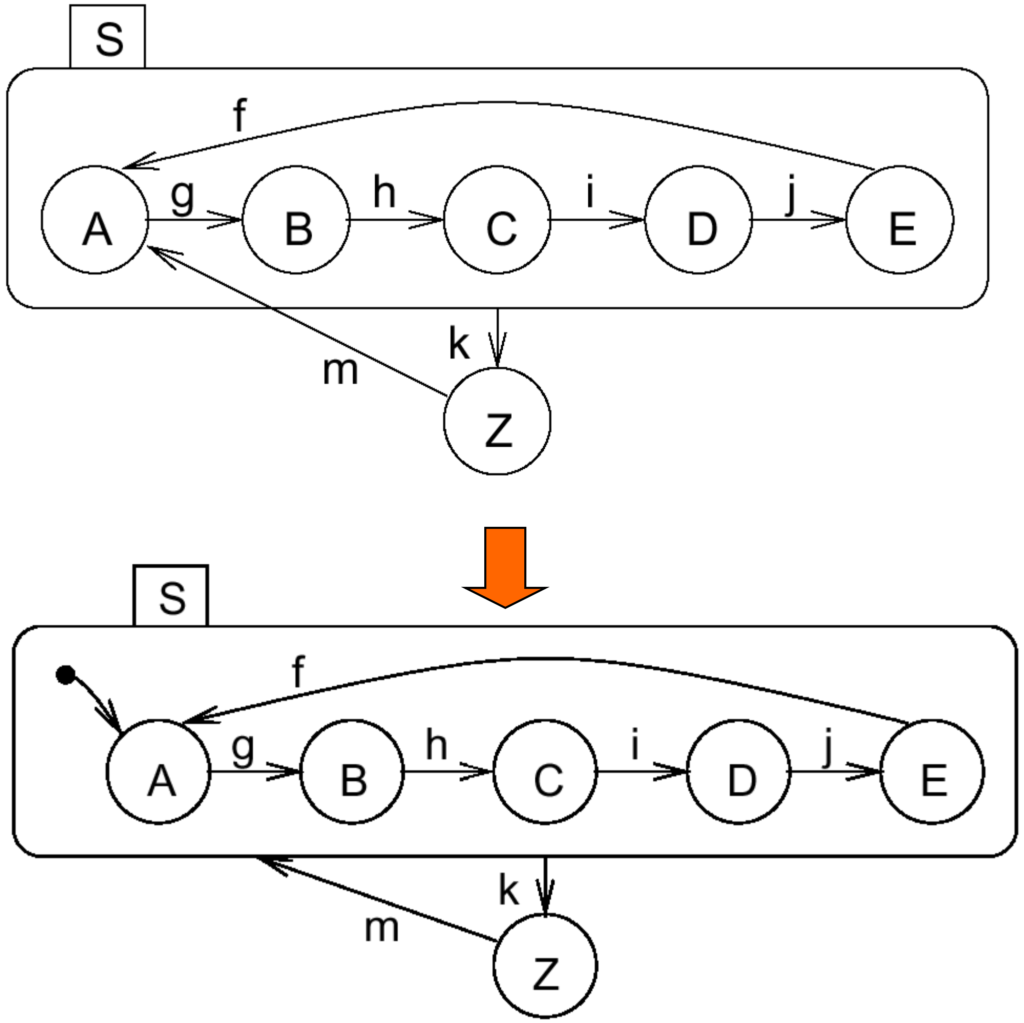
Hierarchy - transitions to super-states



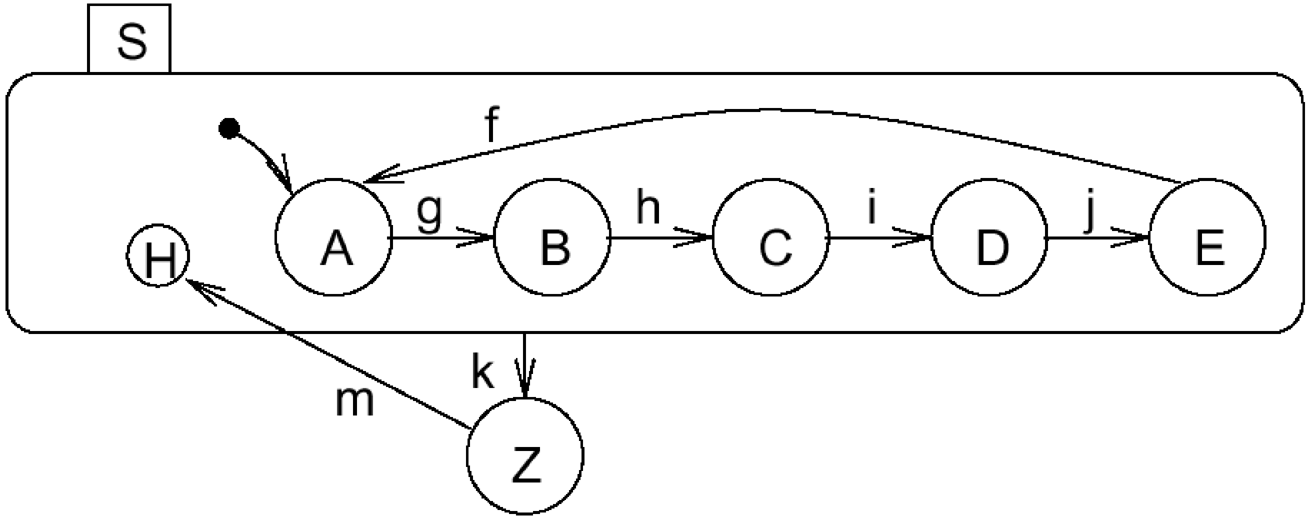
- What is the meaning of transitions to superstates, i.e., what basic state is entered when a superstate is entered?
  - default state mechanism
  - history mechanism

Default state mechanism

- Filled circle indicates sub-state entered whenever super-state is entered.
- Not a state by itself!
- Allows internal structure to be hidden for outside world



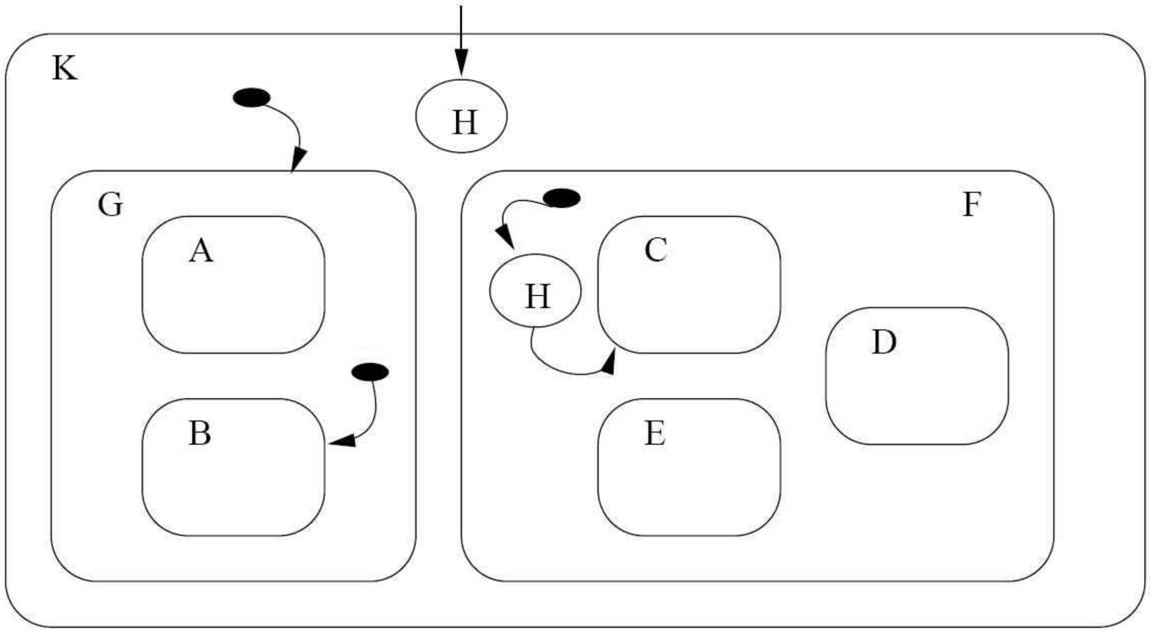
History mechanism



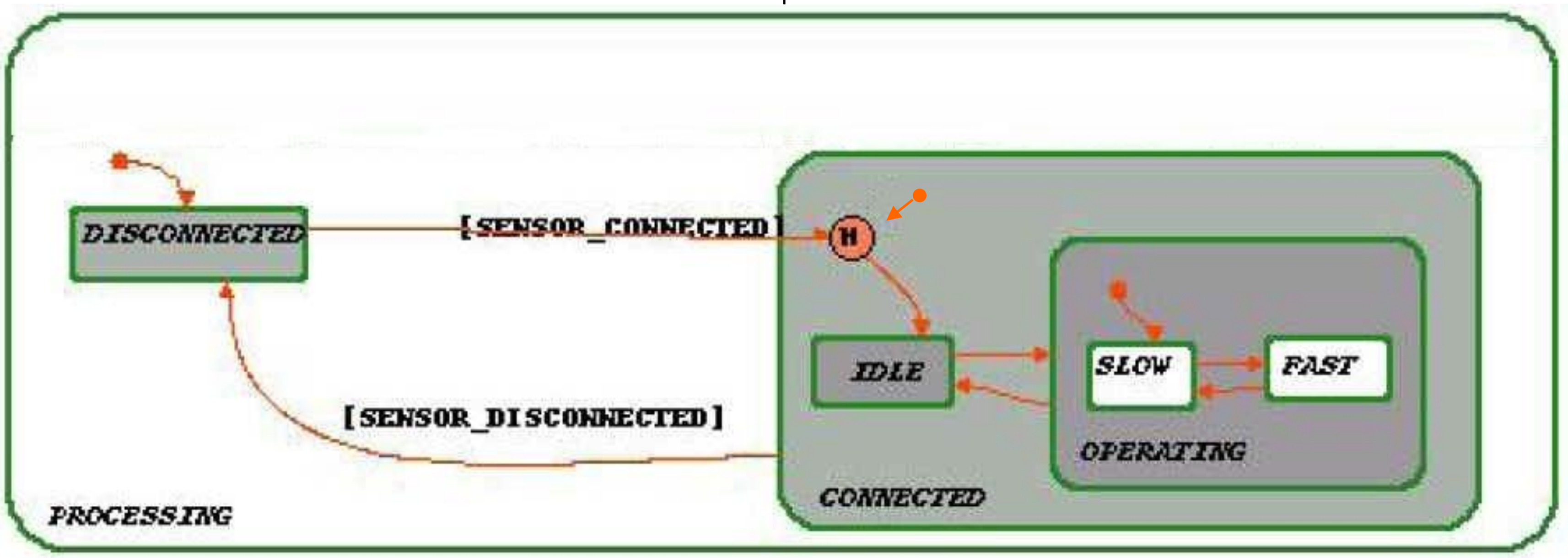
- For event m, S enters the state it was in before S was left (can be A, B, C, D, or E). If S is entered for the very first time, the default mechanism applies.

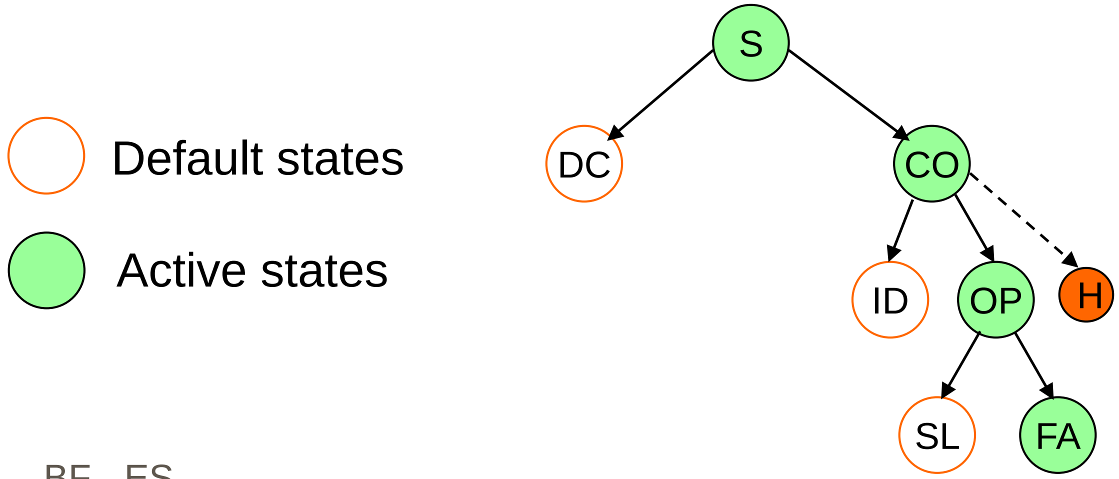
History and default state mechanism

- History and default mechanisms may be used at different levels of hierarchy.



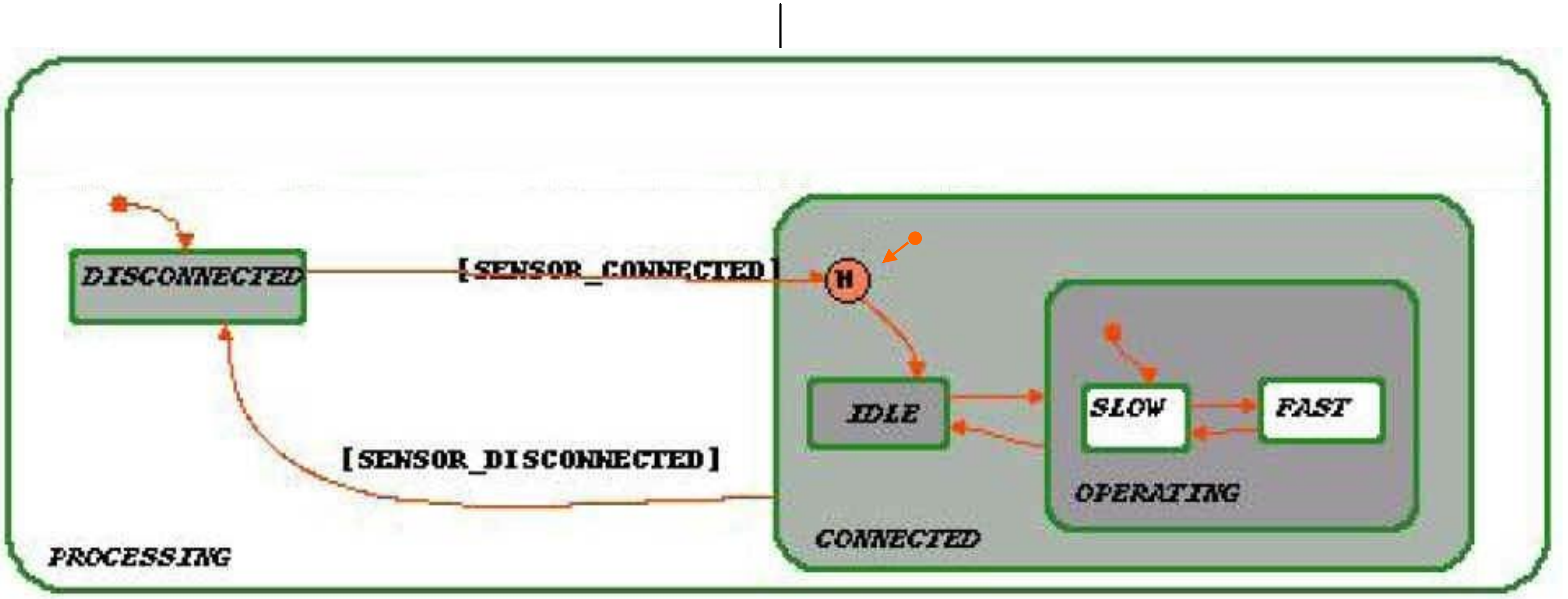
History and deep history

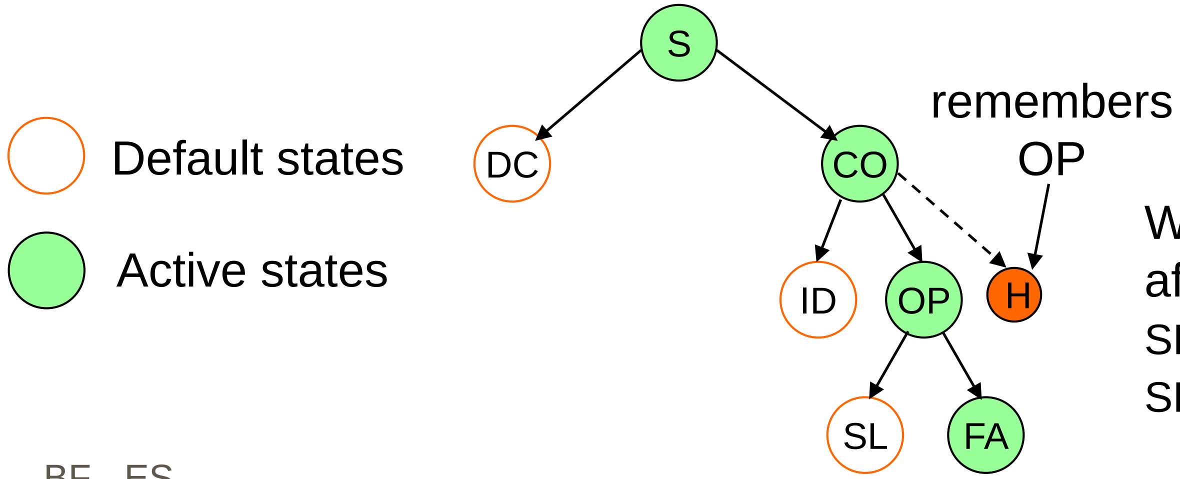




History connectors remember states **at the same level** as the history connector!

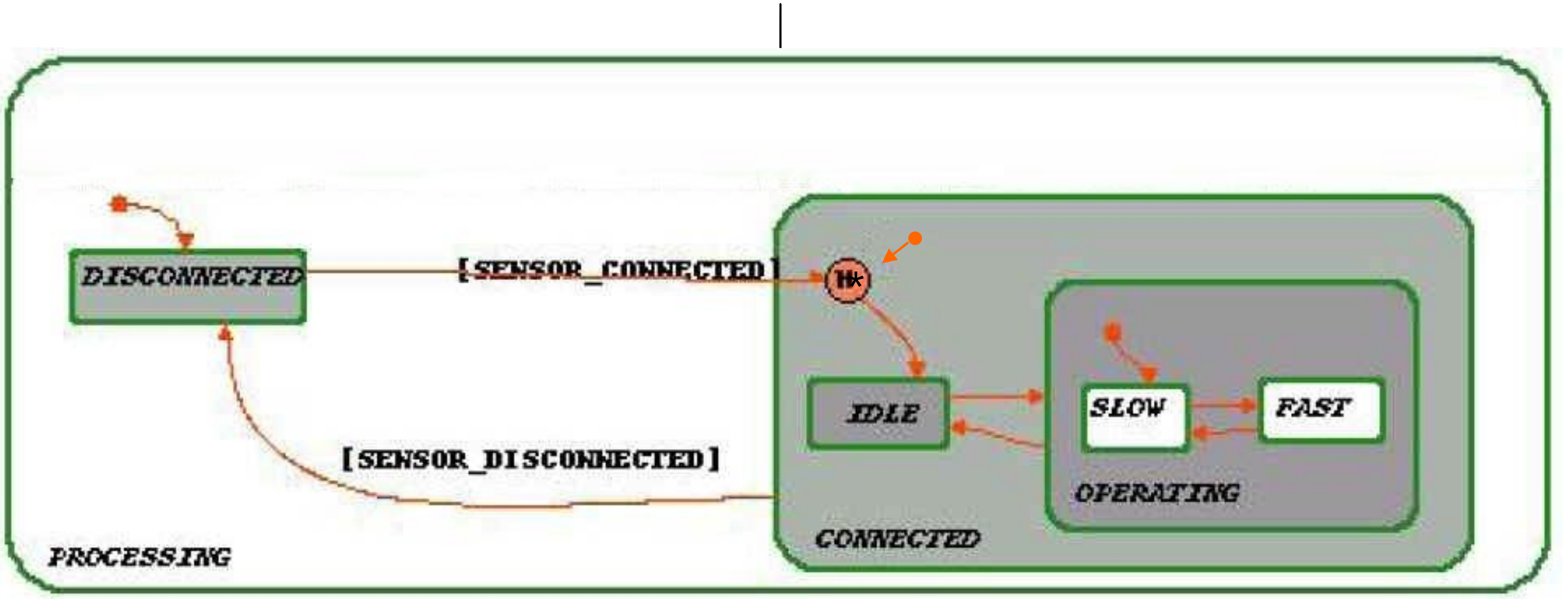
History and deep history

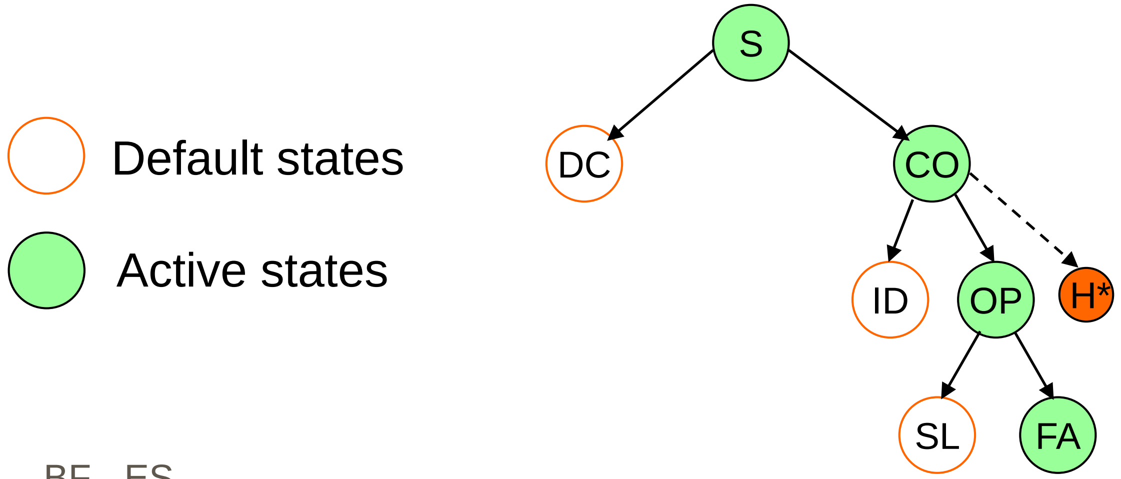




What state is entered after sequence SENSOR_DISCONNECTED, SENSOR_CONNECTED?

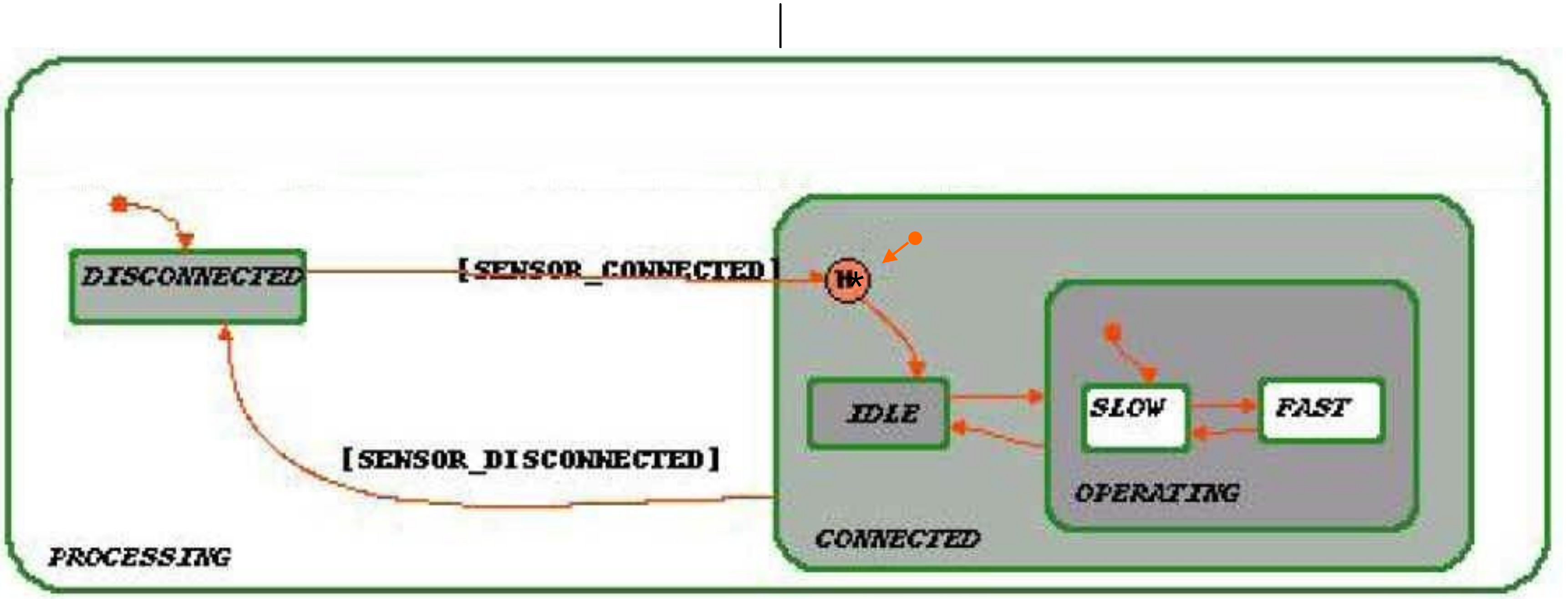
History and deep history

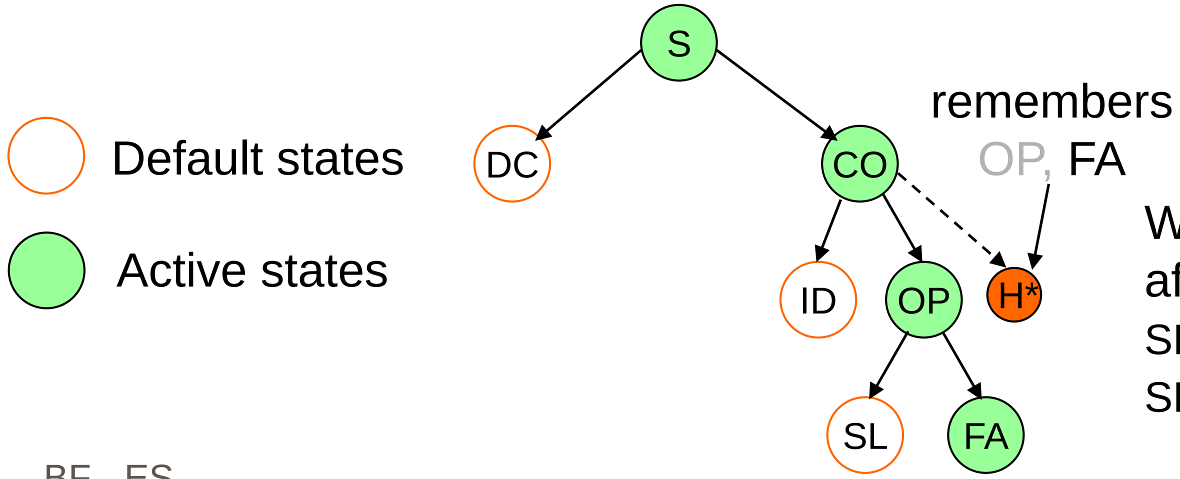




Deep history connectors H* remember **basic states**!

History and deep history





What state is entered after sequence SENSOR_DISCONNECTED, SENSOR_CONNECTED?

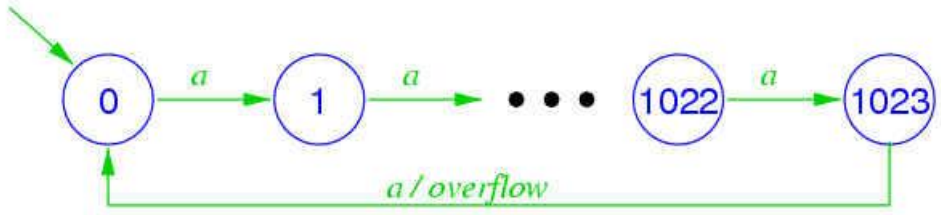
Variables with complex data types

Similar to extended FSMs:

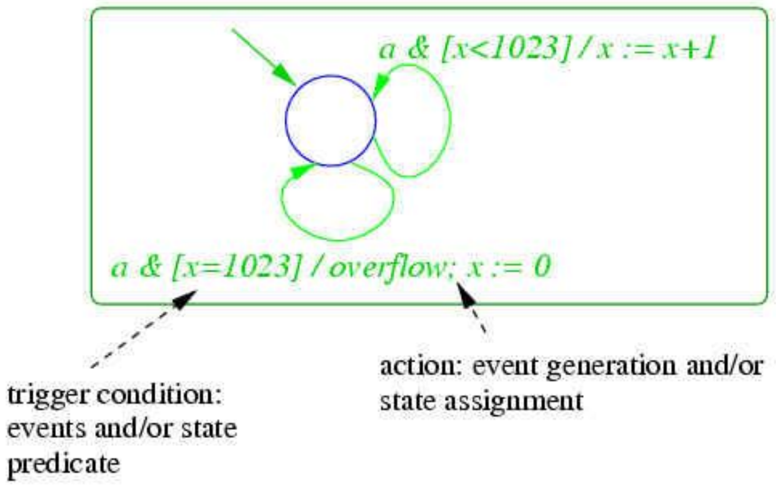
- Include typed variables (e.g. integers, reals, strings, records) to represent data
- Both „graphical states“ and variables contribute to the state of the statechart.
- Notation:
  - „graphical states“ = states
  - „graphical states“ + variables = **status**

A 10-Bit counter, counting on event a and issuing $overflow$ after 1024 occurrences:

As FSM:



As Statechart:


a & [x<1023] / x := x+1
a & [x=1023] / overflow; x := 0
trigger condition: events and/or state predicate
action: event generation and/or state assignment

Events and variables

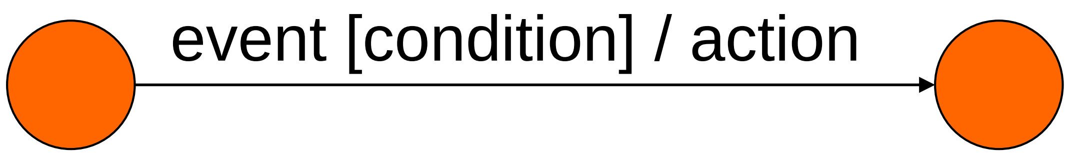
Events:

- Exist only until the next evaluation of the model
- Can be either internally or externally generated

Variables:

- Values of variables keep their value until **they are reassigned.**

General form of edge labels



Meaning:

- Transition may be taken, if event occurred in last step and condition is true
- If transition is taken, then reaction is carried out.

Conditions:

- Refer to values of variables

Actions:

- Can either be assignments for variables or creation of events

Example:

- a & [x = 1023] / overflow; x:=0

Events, conditions, actions

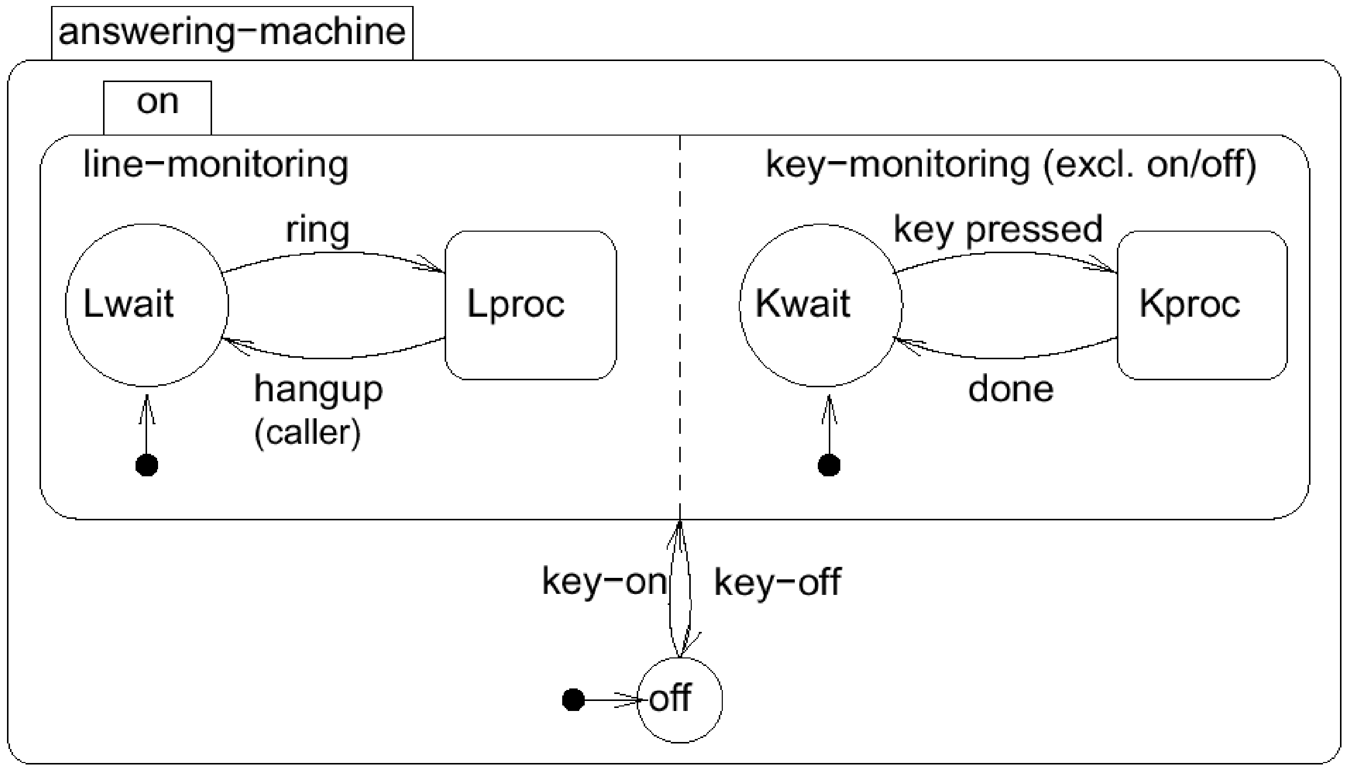
- Possible events (incomplete list):
  - Atomic events
    - Basic events: A, B, BUTTON_PRESSED
    - Entering, exiting a state: en(S), ex(S)
    - Condition test: [cond], e.g. [X>5]
    - Timeout events: tm (e,d): event tm(e,d) is emitted d time units after event e has occurred
  - Compound events: logical connectives and, or, not
- Possible conditions (incomplete list):
  - Atomic conditions
    - Constants: true, false
    - Condition variables (i.e. variables of type boolean)
    - Relations between values: X > 1023, X · Y
    - Residing in a state: in(S)
  - Compound events: logical connectives and, or, not

Events, conditions, actions

- Possible actions (incomplete list):
  - Atomic actions
    - Emitting events: E (E is event variable)
    - Assignments: X := expression
    - Scheduled actions: sc!(A, N) (means perform action after N time units)
  - Compound actions
    - List of actions: A1; A2; A3
    - Conditional action: if cond then A1 else A2

Concurrency

- **AND-super-states**: FSM is in **all** (immediate) sub-states of a AND-super-state; Example:


answering−machine
on
line−monitoring
key−monitoring (excl. on/off)
ring
Lwait
Lproc
hangup (caller)
key pressed
Kwait
Kproc
done
key−on
key−off
off