Cyber-Physical Systems (CSC.T431)

Timed Model (1)

Instructor: Takuo Watanabe (Department of Computer Science)

Agenda

- Timed Model (1)

Course Support & Material

- Slides: OCW-i
- Course Web: https://titech-cps.github.io
- Course Slack: titech-cps.slack.com

Models of Computation

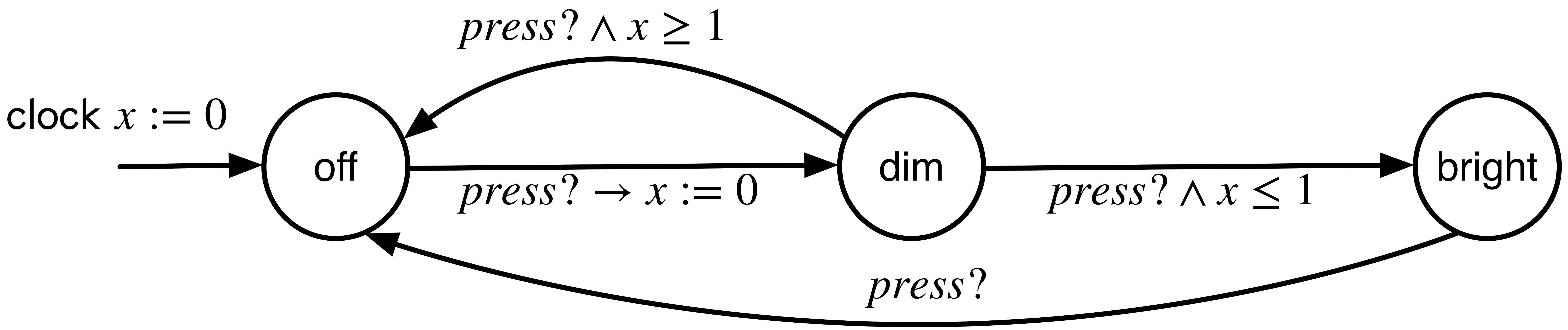
- Synchronous Model
  - Components execute in a sequence of discrete rounds in lock-step.
  - In each round, a component executes all tasks in a order consistent with constraints
- Asynchronous Model
  - The execution speeds of the different processes are independent.
  - In each step, a process executes a single enabled task
- Continuous-time Model
  - Components run synchronously under continuous time.
  - The execution of a component is described using a system of differential equations.

Timed Model

- Similar to asynchronous model.
- Timed Process
  - executes asynchronously, but rely on the global physical time.

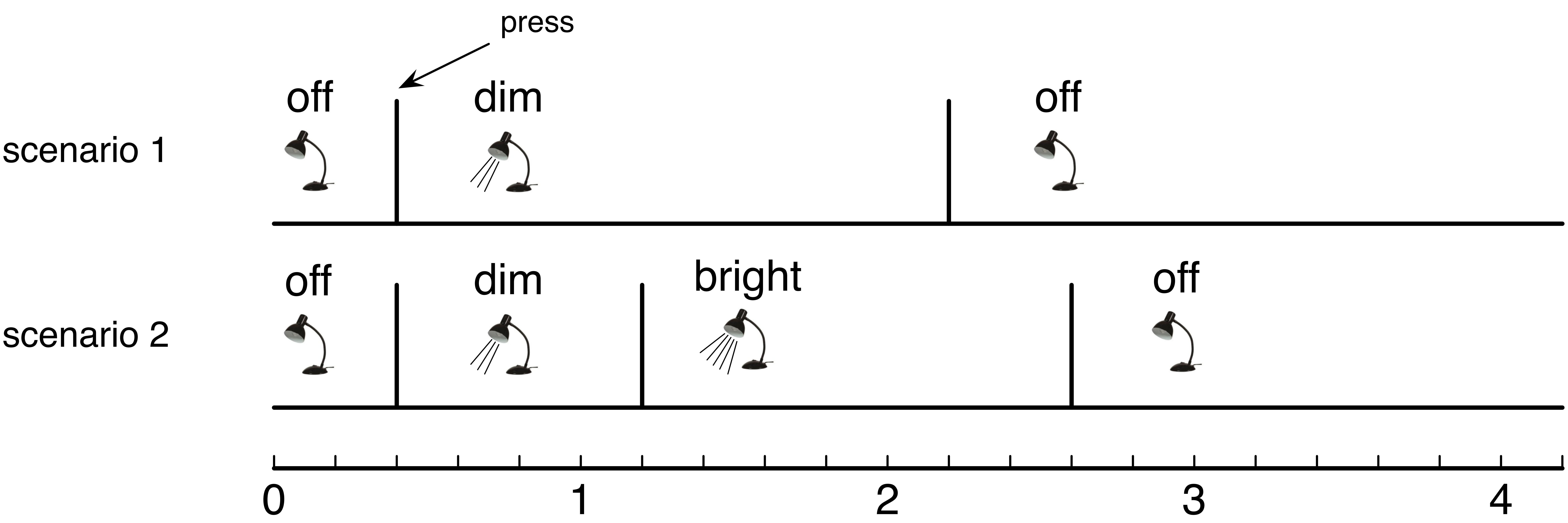
- For example, the timed model allows us to express:
  - "Execute the task corresponding to sensing of temperature every 5ms."
  - "The delay between the reception of an input value and the corresponding output response is between 2ms to 4ms."
  - "If an acknowledgment is not received within 4ms, resend."

Ex. Light Switch



- A light switch with a single push button
- If the button is pressed once, it turns the light on at a dim intensity.
- If the button is pressed twice quickly, it turns the light on at a bright intensity.
  - quickly? : the duration between the successive press $\leq$ 1 sec
  - If it takes more time ($>$ 1 sec), the second press turns the light off.

Ex. Light Switch



Timed Action

- As in an asynchronous process, a timed process may have input, output, and internal actions.
- A timed process also has *timed actions* that model the elapse of time.
- A timed process may have one or more <u>clock variables</u> that have nonnegative real values.
  - In a timed action of duration δ, each clock variable is incremented by δ.
  - During a timed action, state variables other than clock variables stay unchanged.
  - Clock variables do not change with input, output, and internal actions unless they are reset.
- Intuitively, clock variables keep increasing in the same rate while the process stays in a mode.

Possible Execusions

- scenario 1

$(\text{off},0) \xrightarrow{0.4} (\text{off},0.4) \xrightarrow{press?} (\text{dim},0) \xrightarrow{0.6} (\text{dim},0.6) \xrightarrow{0.8}$

$(\text{dim},1.4) \xrightarrow{0.4} (\text{dim},1.8) \xrightarrow{press?} (\text{off},1.8) \xrightarrow{0.5} (\text{off},2.3) \xrightarrow{1.7}$

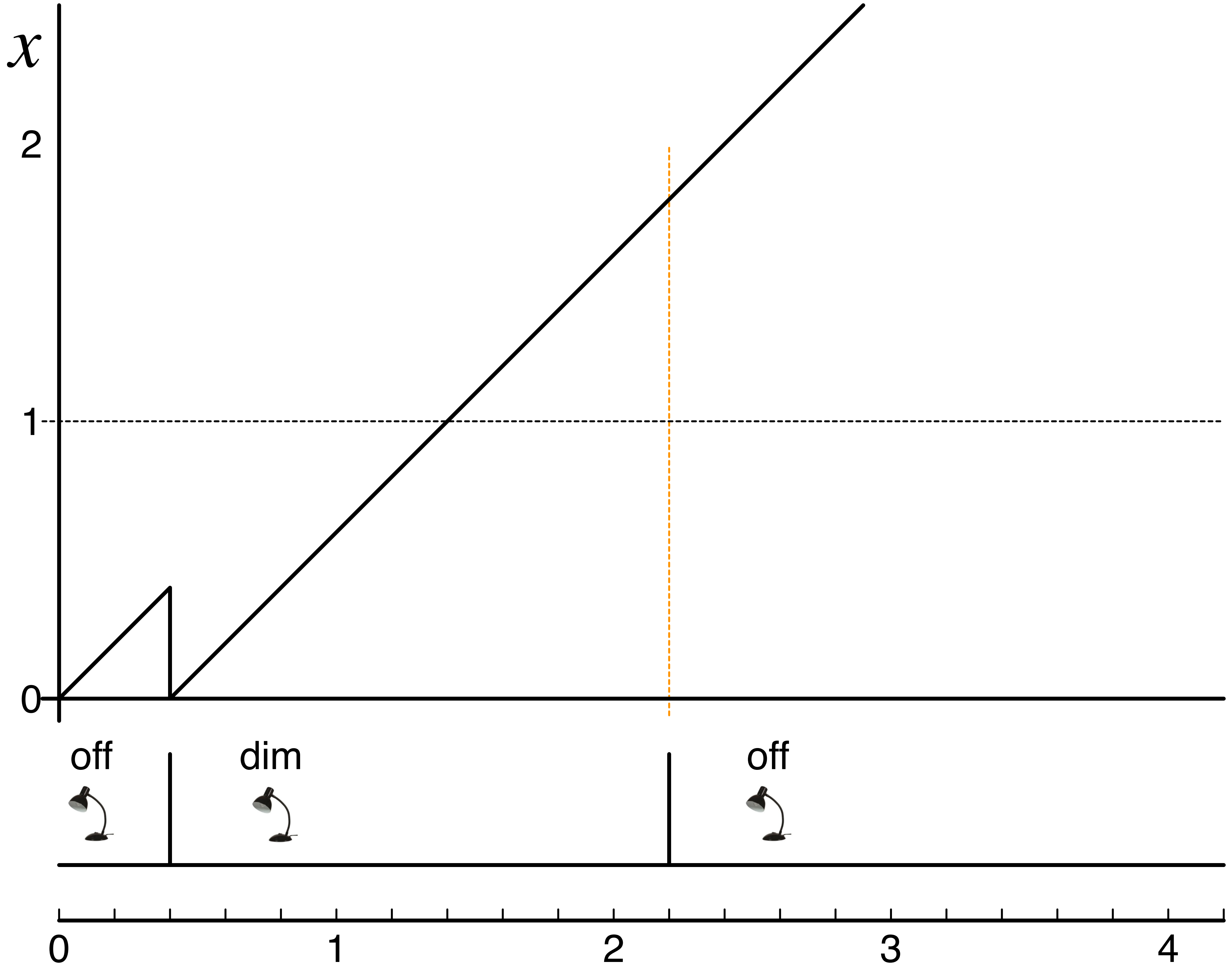
- scenario 2

$(\text{off},0) \xrightarrow{0.4} (\text{off},0.4) \xrightarrow{press?} (\text{dim},0) \xrightarrow{0.4} (\text{dim},0.4) \xrightarrow{0.4} (\text{dim},0.8) \xrightarrow{press?}$

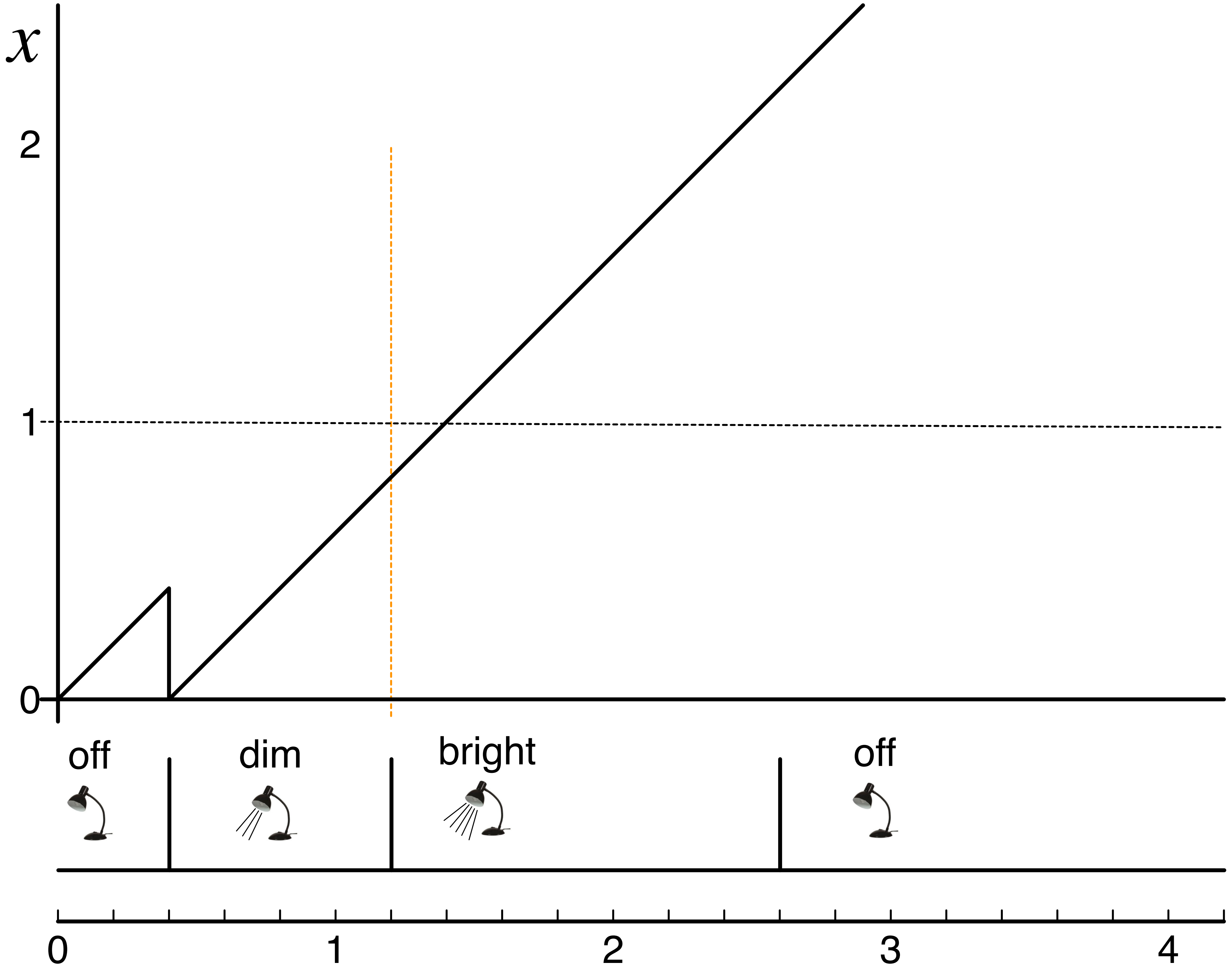
$(\text{bright},0.8) \xrightarrow{1.0} (\text{dim},1.8) \xrightarrow{0.4} (\text{dim},2.2) \xrightarrow{press?} (\text{off},2.2) \xrightarrow{0.6} (\text{off},2.8)$

Clock Variable

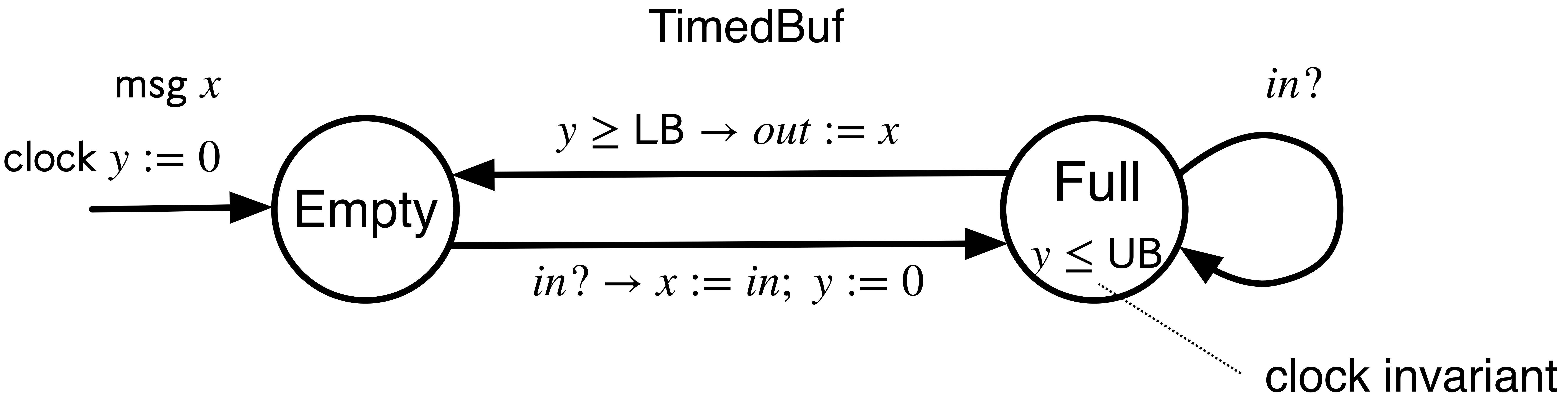
scenario 1



scenario 2

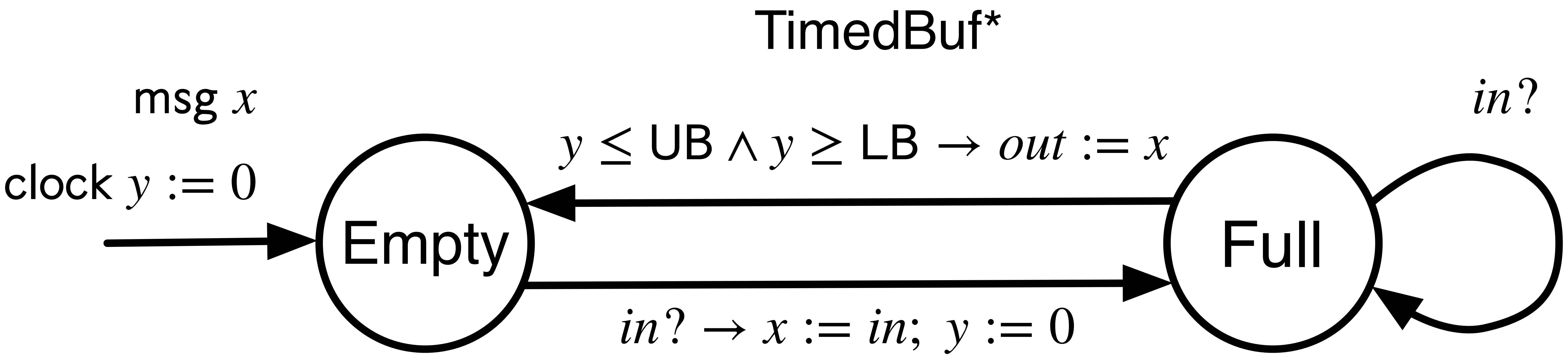


Ex. Buffer with a Bounded Delay



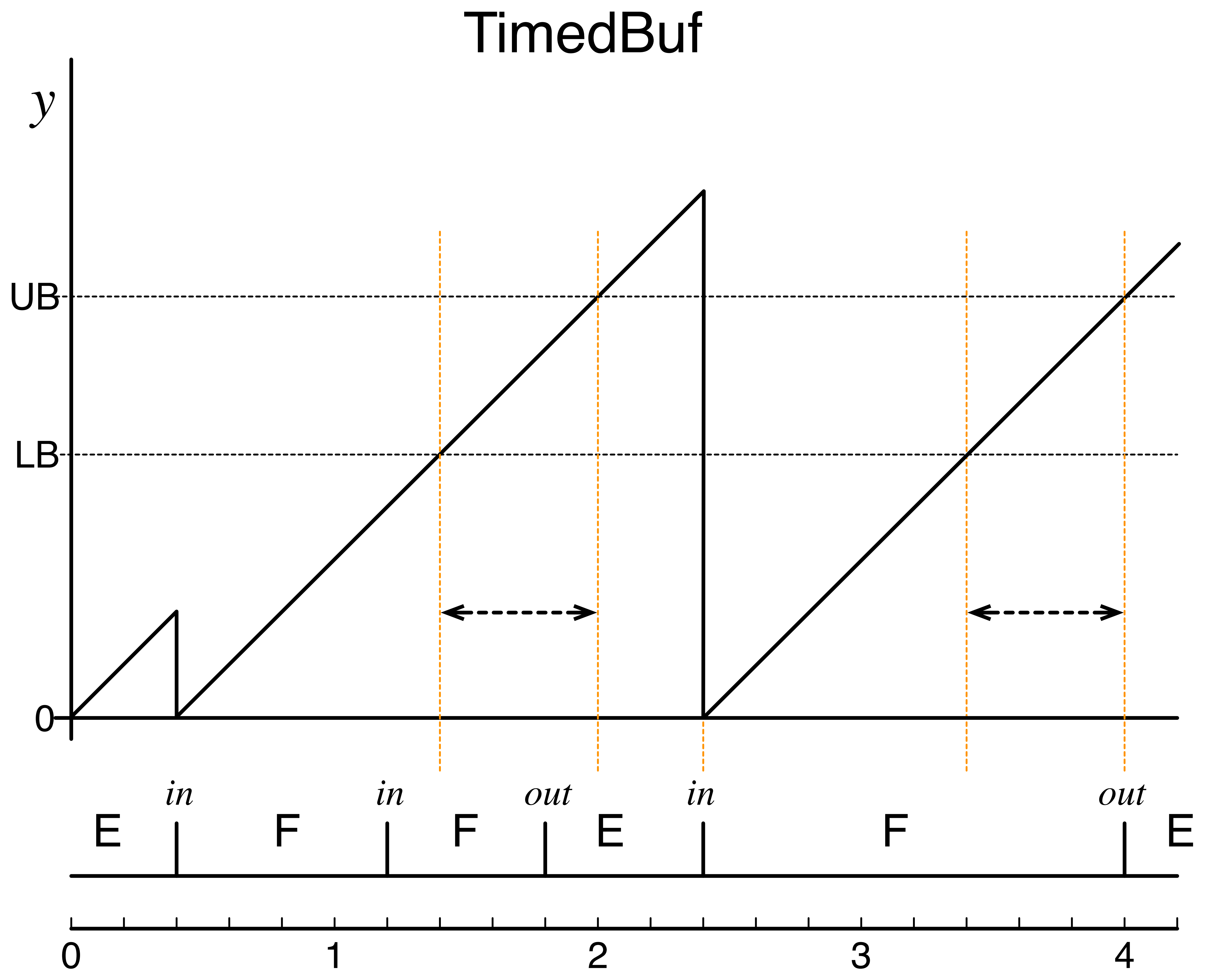
- Behavior of TimedBuf
  - It can receive an input value of type msg from in if it is empty.
  - While it stores the received value, it ignores the subsequent inputs.
  - The time it send the content to out after the reception is at least LB and at most UB.
- The condition associated with a mode is called a *clock invariant*.
  - It must hold while the process is in the mode.

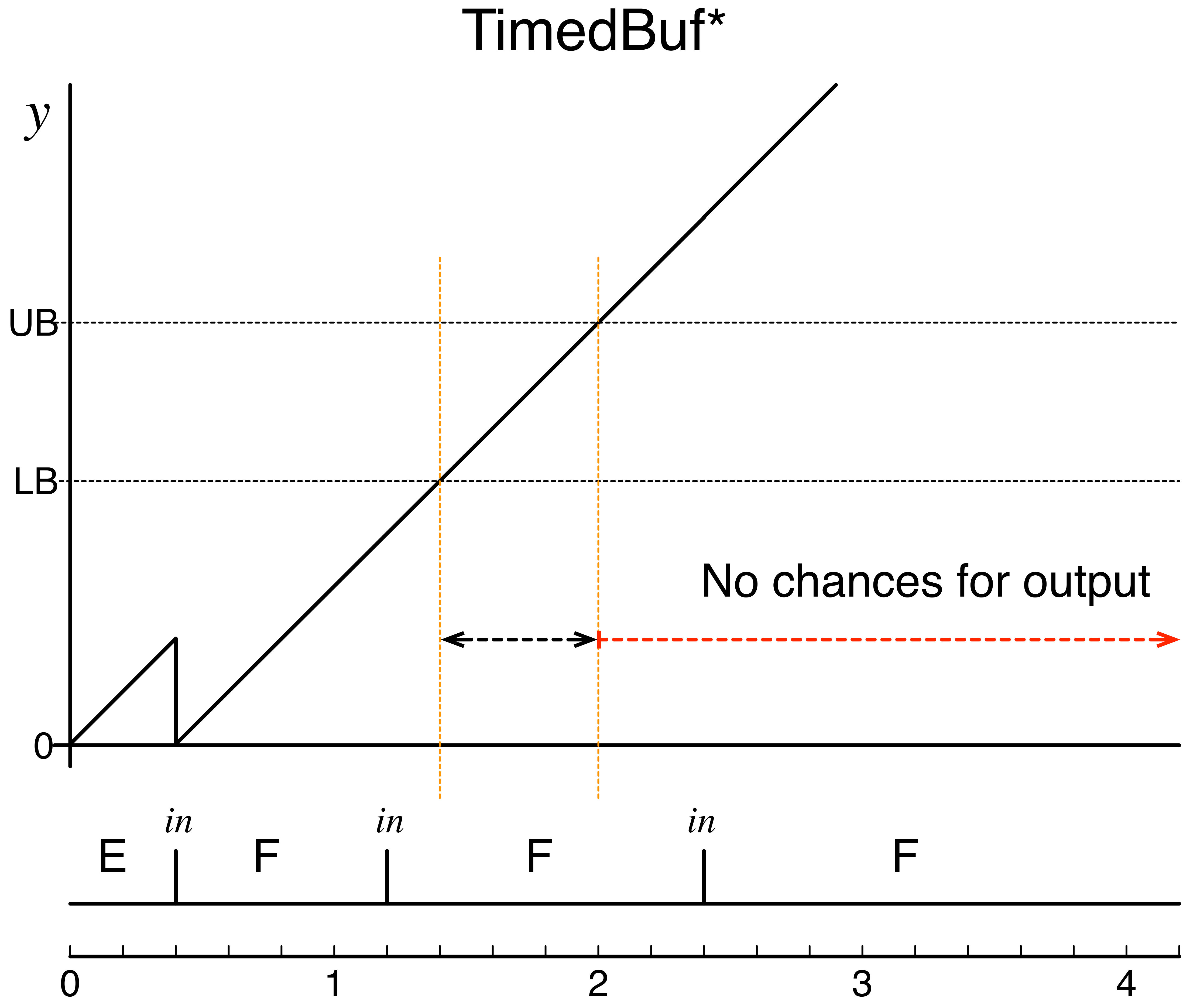
Ex. Buffer with a Bounded Delay



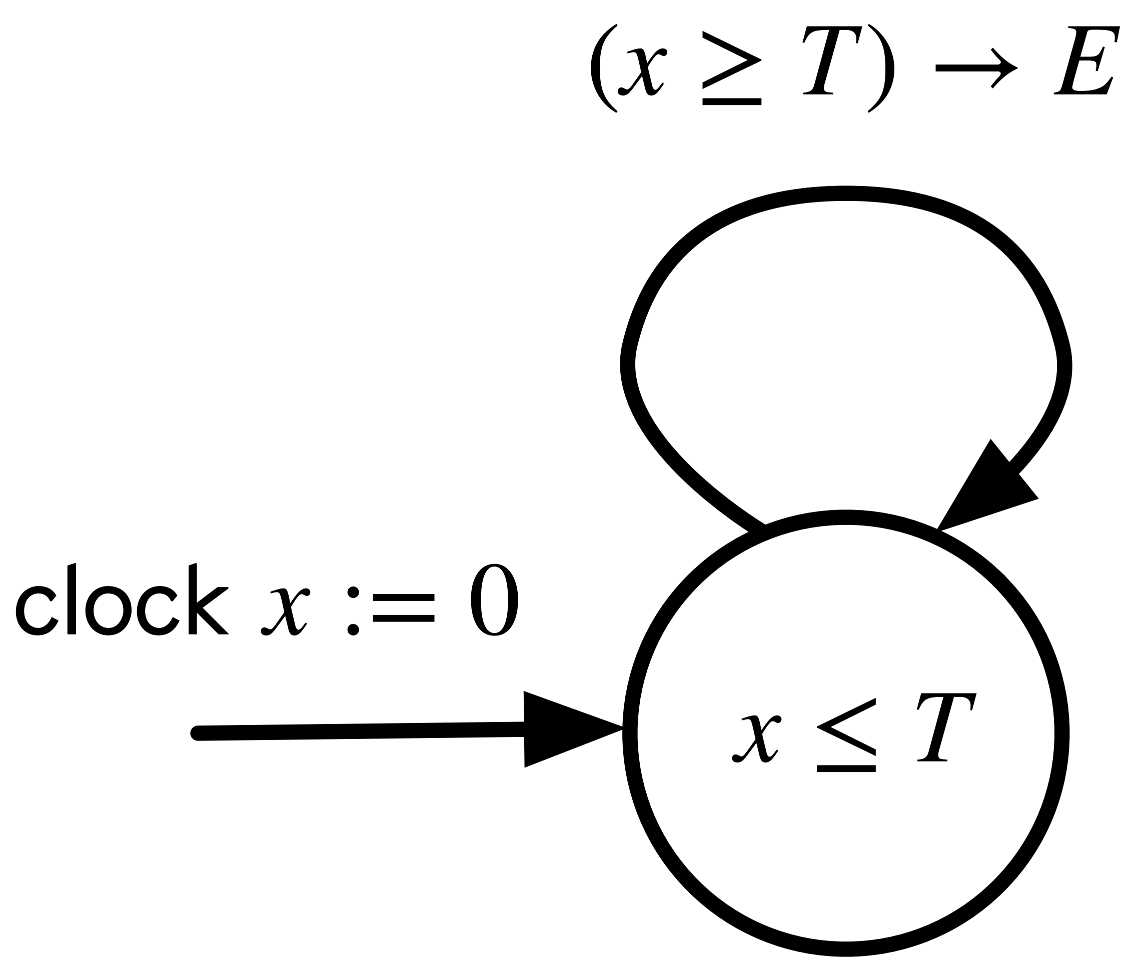
- Modify TimedBuf by moving the condition $y \leq$ UB to the guard condition.
- What is the difference between TimedBuf and TimedBuf*?
- Because Full has no clock invariant, a timed action with which y exceeds UB may happen.
- Once such an action happen, there is no way to send the content.

TimedBuf vs. TimedBuf*

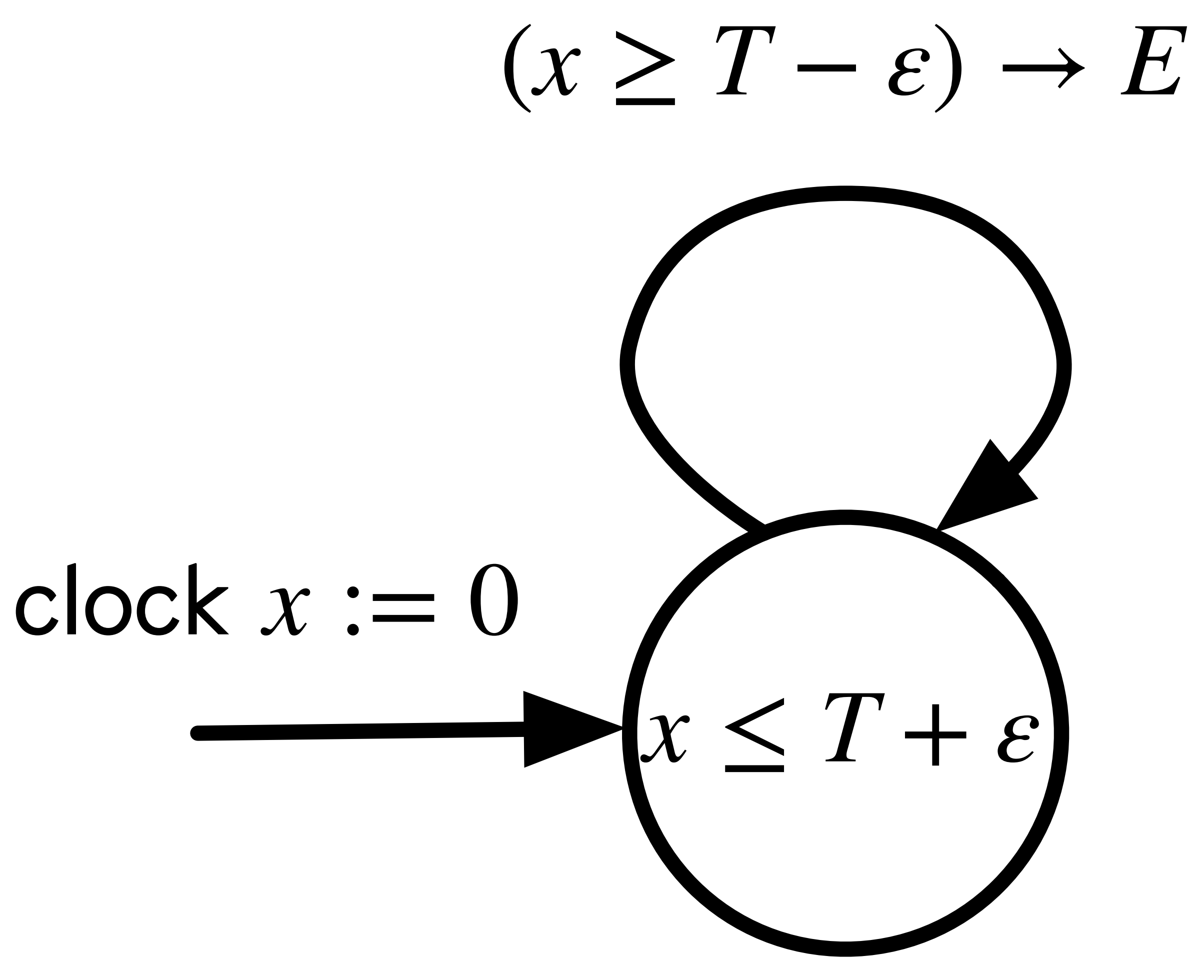




Periodic Tasks

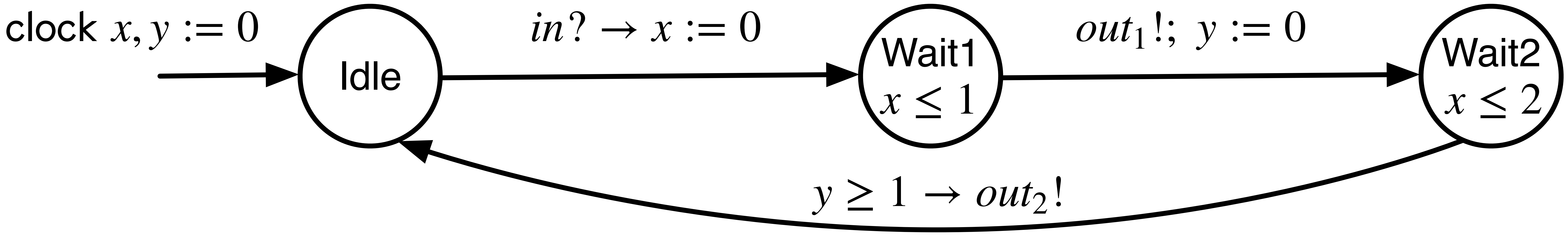


A periodic task with period T



A periodic task with period T
(with jitters)

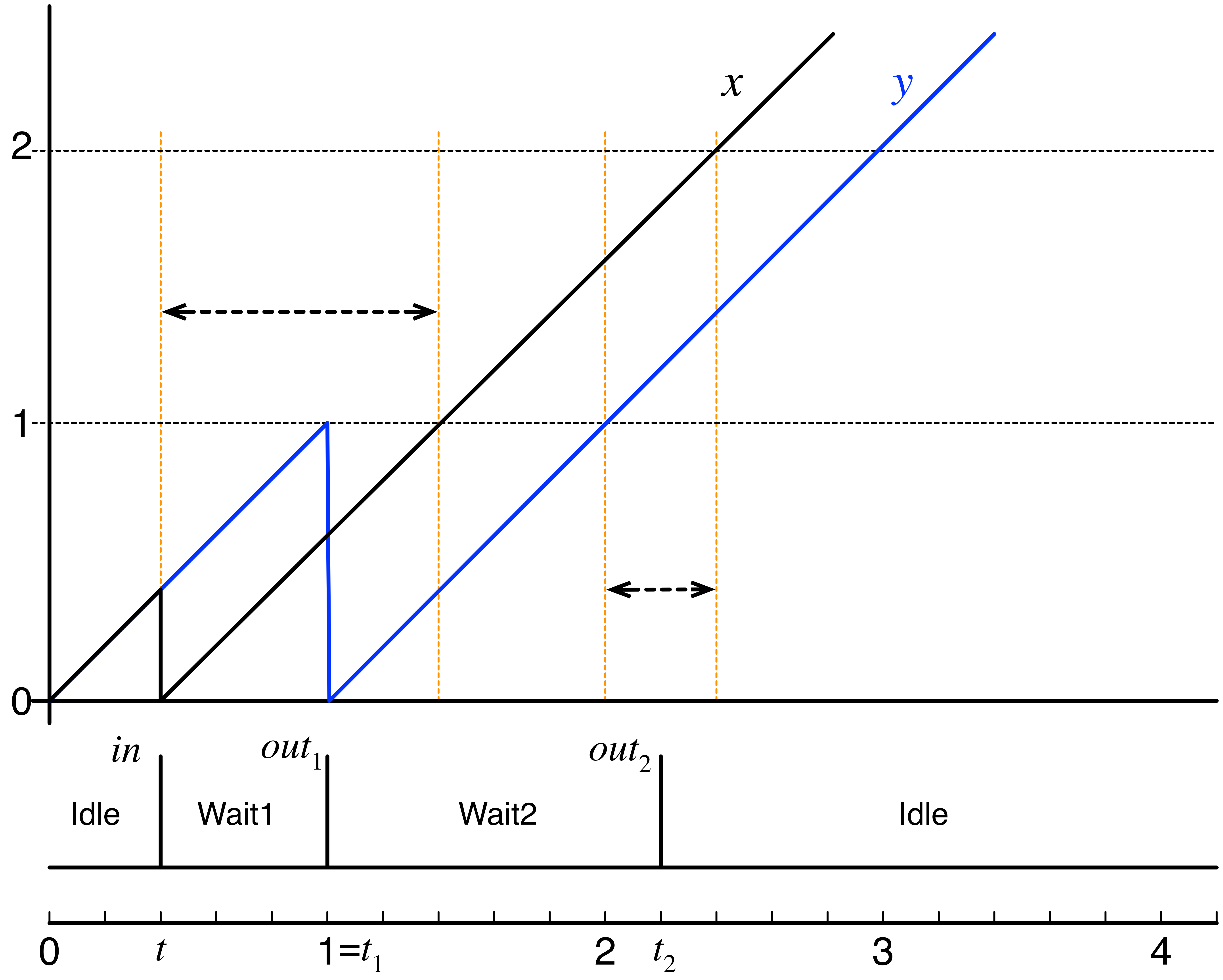
Ex. Multiple Clocks



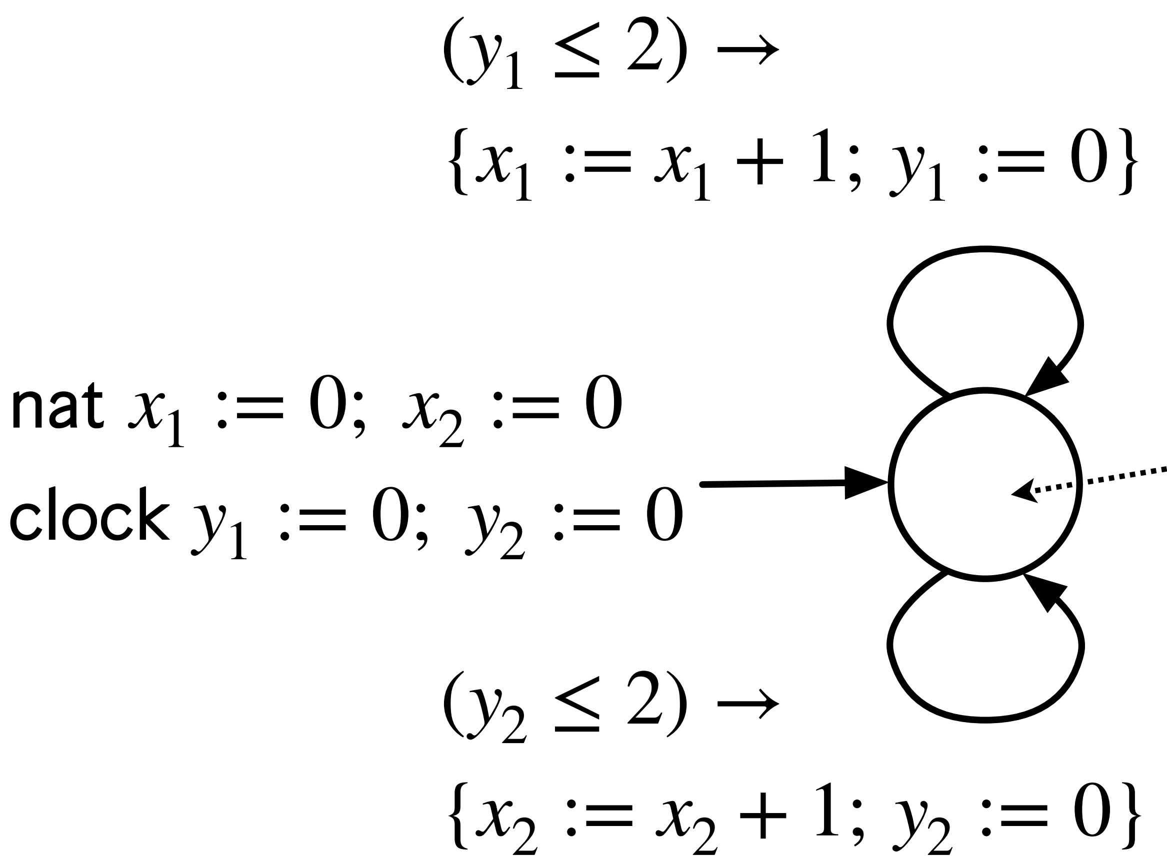
- Suppose that an input event on in happens at time t. The process responds by producing an output event on out_1 at time t_1, followed by an output event on out_2 at time t_2.
- Then (1) $t_1 - t \leq 1$, (2) $t_2 - t \leq 2$, and (3)$t_2 - t_1 \geq 1$ hold.
- All clock variables increase at the same rate.
    - $(m, c_1, c_2) \xrightarrow{\delta} (m, c_1 + \delta, c_2 + \delta)$

Ex. Multiple Clocks

$$
\begin{aligned}
& && (\mathrm{Idle}, 0, 0) \\
& && \xrightarrow{0.4} (\mathrm{Idle}, 0.4, 0.4) \\
t &= 0.4 && \xrightarrow{in?} (\mathrm{Wait1}, 0, 0.4) \\
& && \xrightarrow{0.6} (\mathrm{Wait1}, 0.6, 1) \\
t_1 &= 1.0 && \xrightarrow{out_1!} (\mathrm{Wait2}, 0.6, 0) \\
& && \xrightarrow{1.2} (\mathrm{Wait2}, 1.8, 1.2) \\
t_2 &= 2.2 && \xrightarrow{out_2!} (\mathrm{Idle}, 1.8, 1.2)
\end{aligned}
$$



Ex. TimedInc



nat $x_1 := 0;\ x_2 := 0$
clock $y_1 := 0;\ y_2 := 0$

$CI :\ y_1 \le 2 \land y_2 \le 2$
$A_1 :\ y_1 \ge 1 \to \{x_1 := x_1 + 1;\ y_1 := 0\}$
$A_2 :\ y_2 \ge 1 \to \{x_2 := x_2 + 1;\ y_2 := 0\}$

AsyncInc

nat $x_1 := 0; x_2 := 0$

$A_1 :\ x_1 := x_1 + 1$
$A_2 :\ x_2 := x_2 + 1$

- AsyncInc with timing constraints
- Possible scenario:
  - $(0,0,0,0) \xrightarrow{\delta_1} (0,\delta_1,0,\delta_1) \xrightarrow{A_1} (1,0,0,\delta_1) \xrightarrow{\delta_2} (1,\delta_2,0,\delta_1+\delta_2) \xrightarrow{A_1} (2,0,0,\delta_1+\delta_2)$
  - The only possible action after this is A_2 because $\delta_1 \ge 1 \land \delta_2 \ge 1 \land \delta_1 + \delta_2 \le 2$.

Asynchronous Process

Formal Definition

- An asynchronous process $P = (I, O, S, Init, \mathscr{A}_I, \mathscr{A}_O, \mathbf{A})$ consists of:
  - I : a finite set of typed *input channels*
  - O : a finite set of typed *output channels*
  - S : a finite set of typed state variables
  - $Init$: a description of the *initialization* defining the set $[\![Init]\!] \subseteq Q_S$ of initial states
  - $\mathscr{A}_I = \{\mathbf{A}_x \mid x \in I\}$ where $\mathbf{A}_x$ is the set of *input tasks* for input channel x
  - $\mathscr{A}_O = \{\mathbf{A}_y \mid y \in O\}$ where $\mathbf{A}_y$ is the set of *output tasks* for output channel y
  - $\mathbf{A}$: set of *internal tasks*

Timed Process

Formal Definition

- A *timed process* TP consists of
  - an asynchronous process P, where some of its state variables can be of type clock, and
  - a clock invariant CI, which is a Boolean expression over the state variables S.
- Inputs, outputs, states, initial states, internal actions, input actions, and output actions of TP are the same as that of asynchronous process P.
- Given a state s and a real-valued time $\delta > 0$, $s \xrightarrow{\delta} s + \delta$ is a timed action of TP if the state $s + t$ satisfies the expression CI for all values $0 \leq t \leq \delta$.
  - Note: For a state s and a real value δ, $s + \delta$ denotes a state s' such that $s'(x) = s(x) + \delta$ for each clock variable x and $s'(y) = s(y)$ for each discrete (= non-clock) variable y.

Ex. TimedBuf

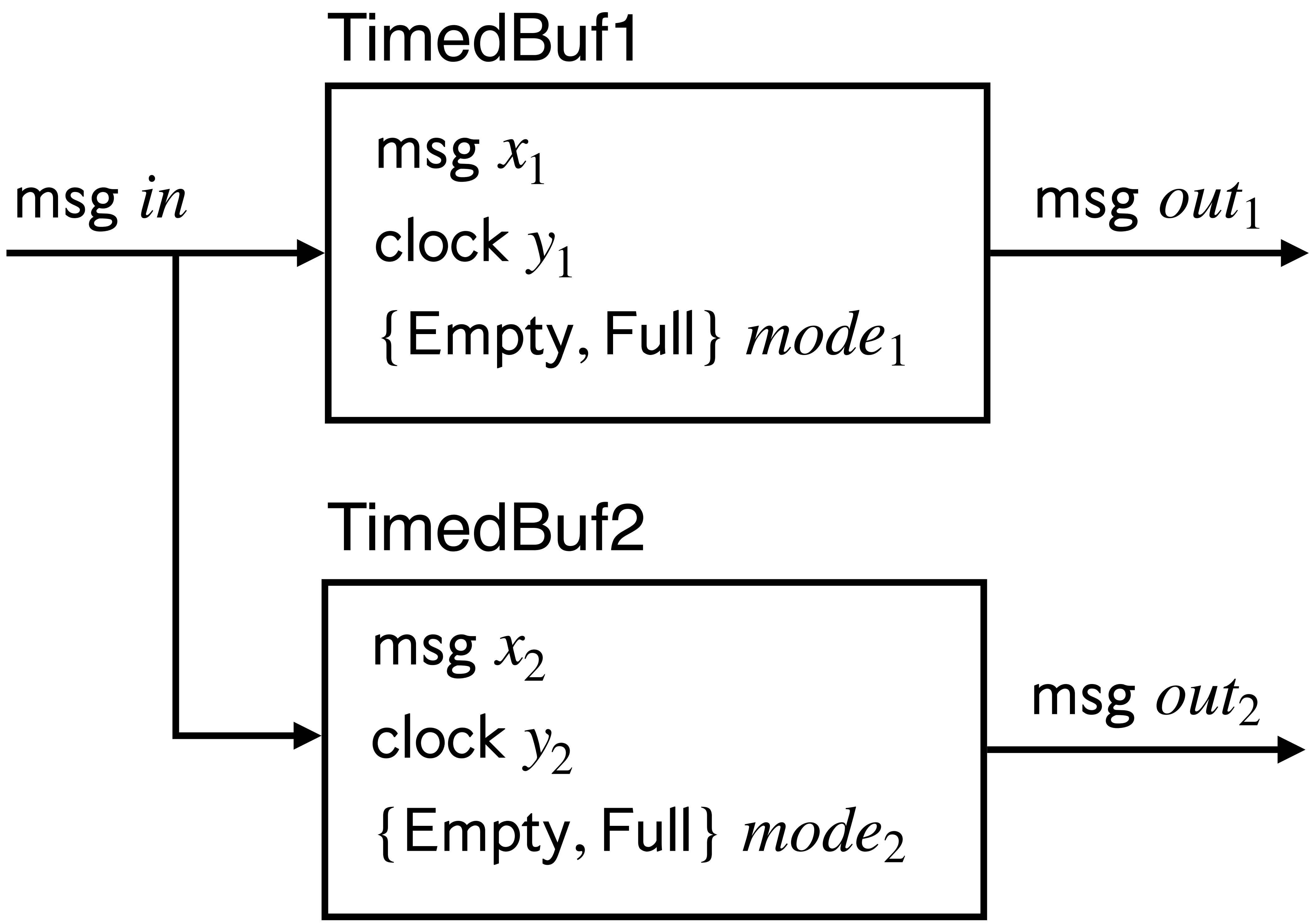
Formal Description

- Input channel: in of type msg
- Output channel: out of type msg
- State variables: $mode$ of type {Empty, Full}, x of type msg, y of type clock
- Initial values: $mode := \text{Empty}, y := 0$
- Input task: $1 \rightarrow \text{if } (mode = \text{Empty}) \text{ then } \{mode := \text{Full}; x := in\}$
- Output task: $(mode = \text{Full}) \wedge y \geq \text{LB} \rightarrow out := x$
- Internal tasks: none
- Clock invariant: $(mode = \text{Full}) \rightarrow (y \leq \text{LB})$

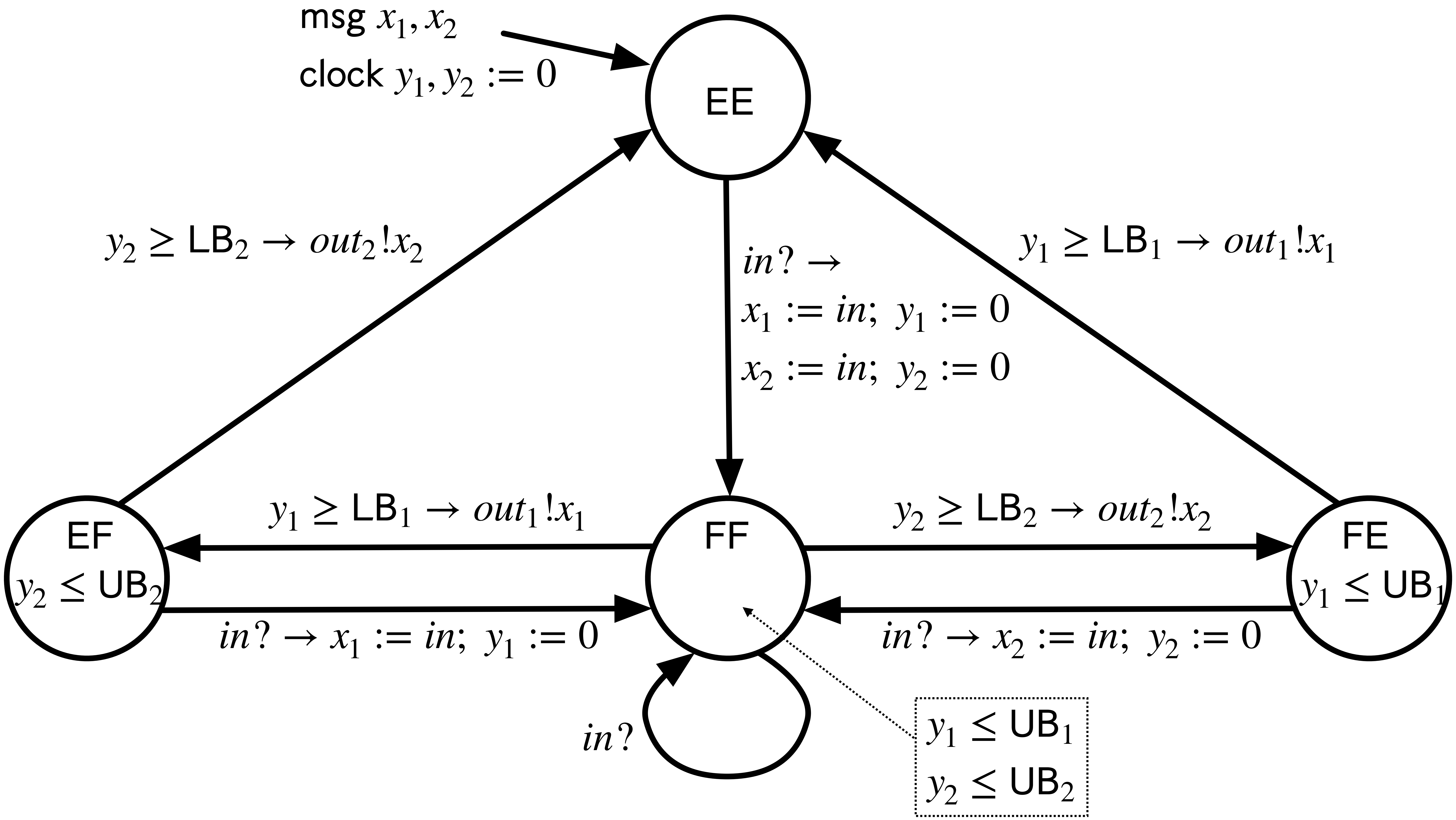
Timed Process Composition

- For two timed processes $TP_1 = (P_1, CI_1)$ and $TP_2 = (P_2, CI_2)$, such that the ooutput channels of them are disjoint, the *parallel composition* $TP_1 \,|\, TP_2$ is the timed process $(P_1 \,|\, P_2, CI_1 \wedge CI_2)$.
    - For states s_1 of TP_1, s_2 of TP_2, and a duration $\delta > 0$, $(s_1, s_2) \xrightarrow{\delta} (s_1 + \delta, s_2 + \delta)$ is a timed action of $TP_1 \,|\, TP_2$ exactly when $s_1 \xrightarrow{\delta} s_1 + \delta$ is a timed action of TP_1 and $s_2 \xrightarrow{\delta} s_2 + \delta$ is a timed action of TP_2.

Ex. Composition of two Timed Buffers

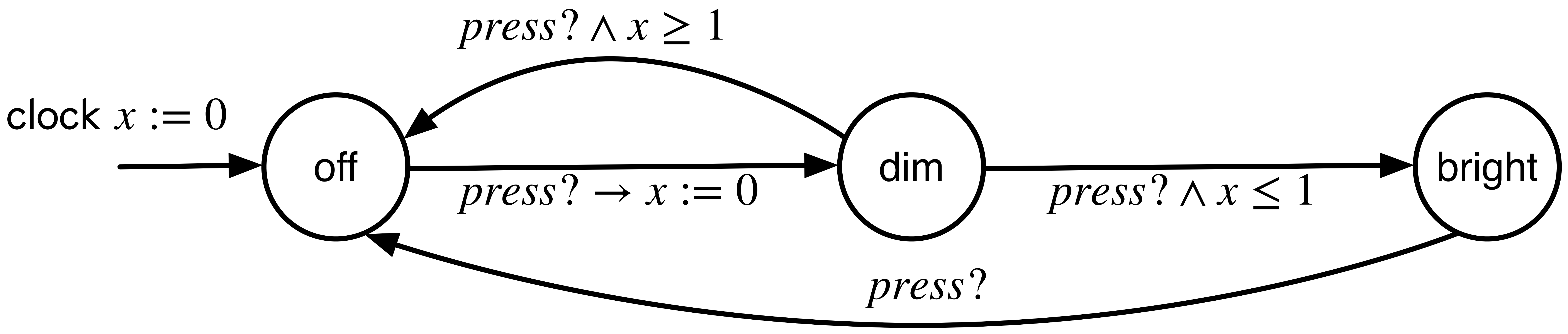


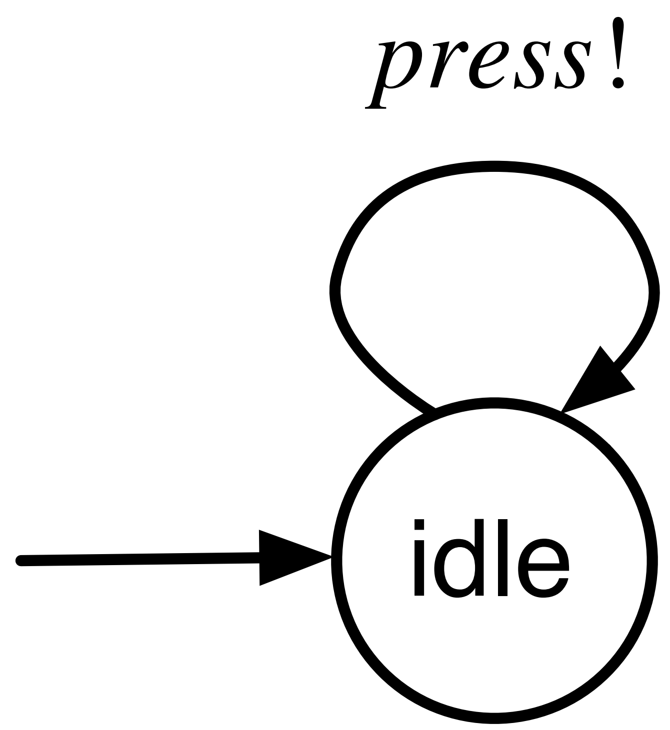
Ex. Composition of two Timed Buffers



Users of the Light

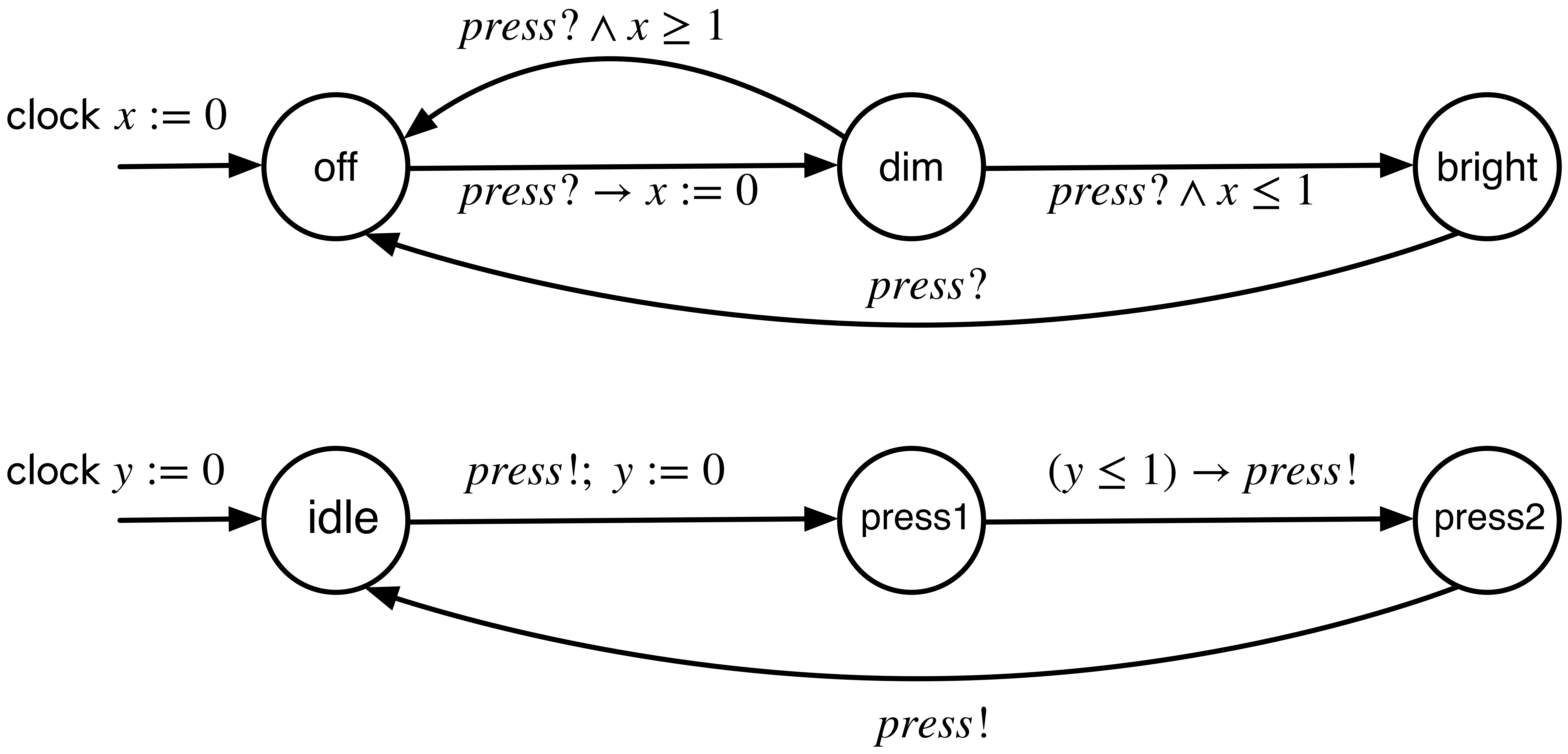
1. Random User





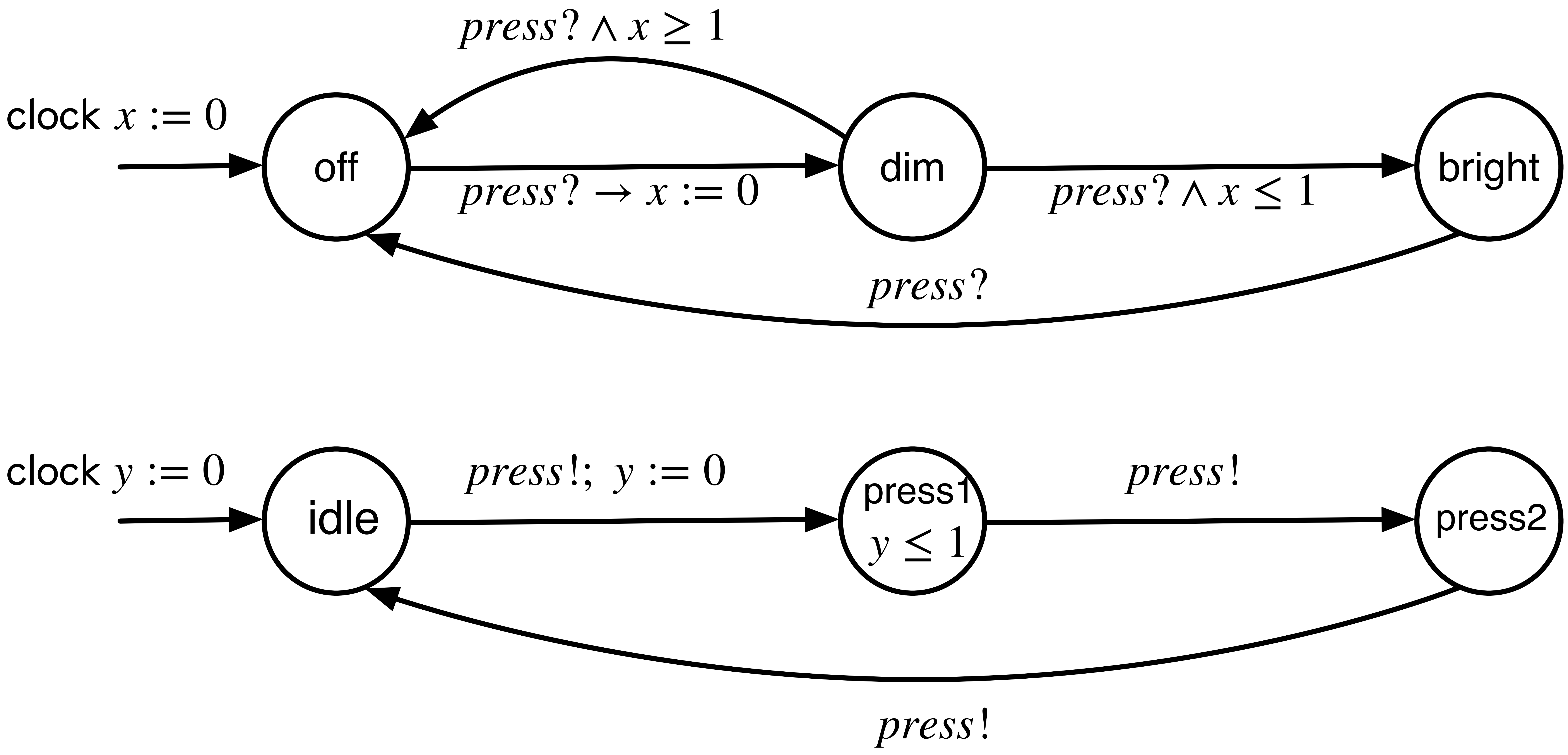
Users of the Light

2. A User who tries to make the light bright (1)



Users of the Light

3. A User who tries to make the light bright (2)



Mutual Exclusion Problem

```
// shared (atomic) variable
int x := 0
```

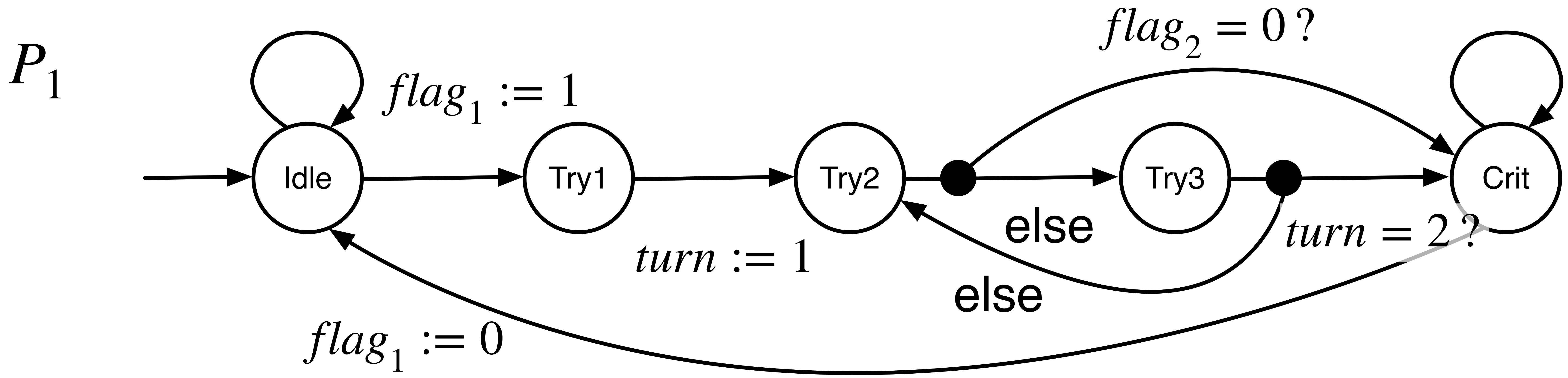
```
// P1
while (true) {
  NC
  EnterCS;
  y1 := x;
  x := y1 + 1;
  ExitCS
}
```

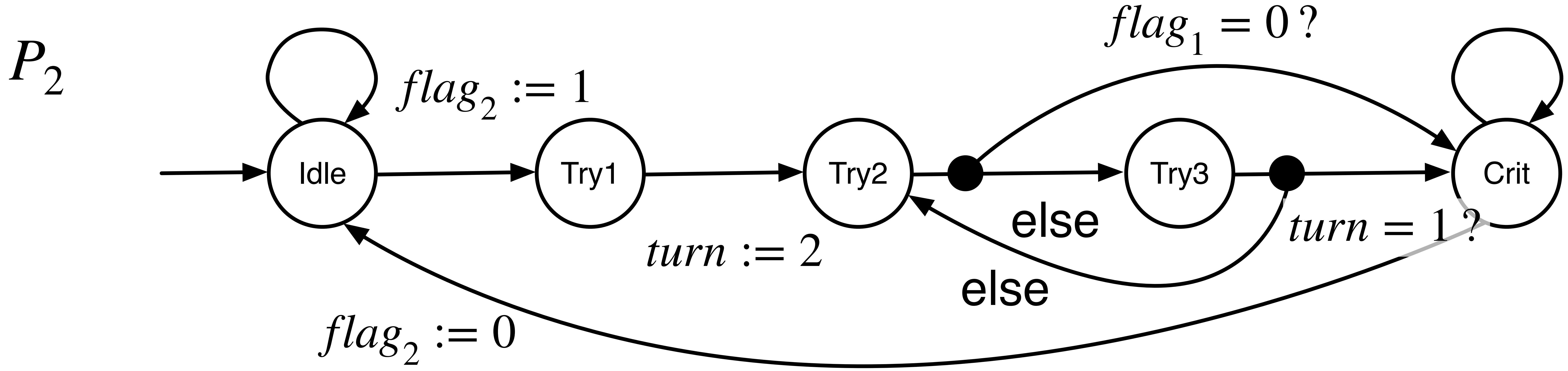
```
// P2
while (true) {
  NC
  EnterCS;
  y2 := x; ;
  x := y2 + 1;
  ExitCS
}
```

- Critical Section
  - A part of a program that have accesses to resources shared by two or more processes
- Mutual Exclusion
  - Safety: No two processes can enter the critical section at the same time.
  - Liveness: Once a process wants to enter the critical section, it should eventually be able to enter (aka freedom from deadlocks).

Peterson's Mutual Exclusion Protocol

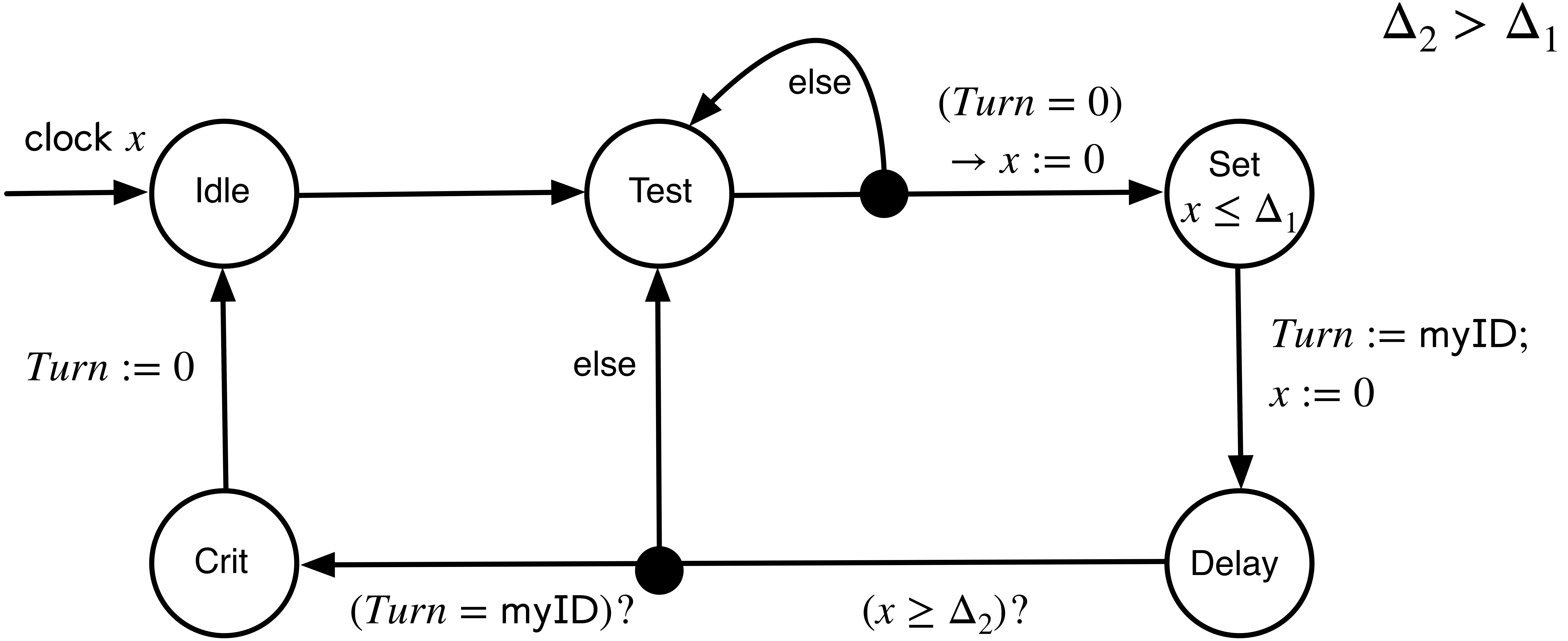
AtomicReg[$bool$] $flag_1 := 0$; $flag_2 := 0$; AtomicReg[$\{1,2\}$] $turn$





Fischer's Protocol

Timing-Based Mutual Exclusion



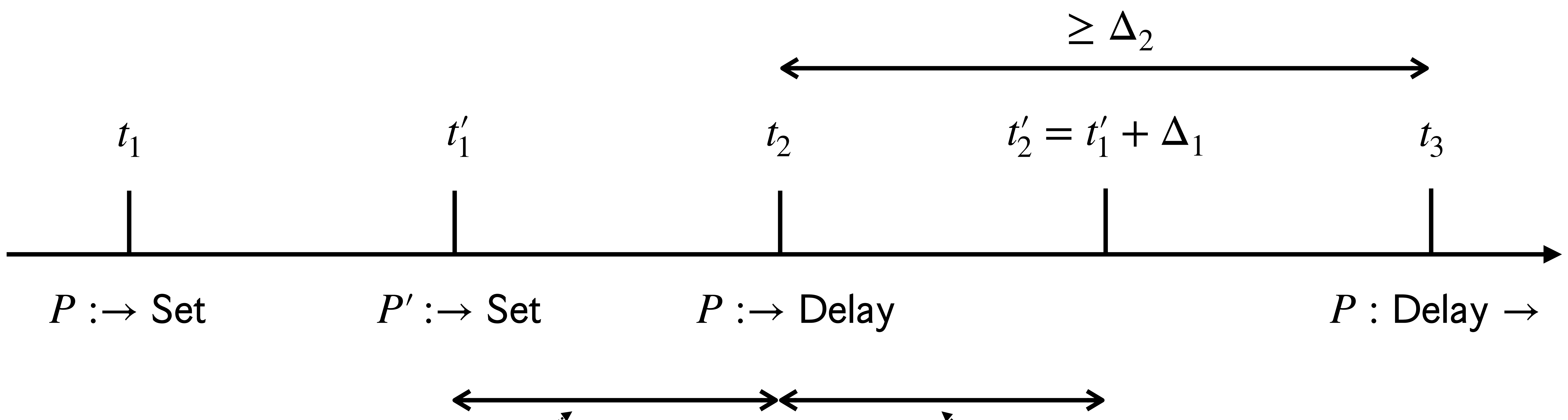
Fischer's Protocol

```
// Shared Variable
{0, ID1, ID2} Turn := 0
```

```
// P1
while (true) {
  NC
  do {
    while (Turn != 0);
    delay≤(Δ1);
    Turn = ID1;
    delay≥(Δ2);
  } while (Turn != ID1);
  CS
  Turn = 0;
}
```

```
// P2
while (true) {
  NC
  do {
    while (Turn != 0);
    delay≤(Δ1);
    Turn = ID2;
    delay≥(Δ2);
  } while (Turn != ID2);
  CS
  Turn = 0;
}
```

Fischer's Protocol



If P' writes $Turn$ in this interval, P will overwrite it at t_2.

If P' writes $Turn$ in this interval, P will find it at t_3 and goes back to Test again.

Fischer's Protocol

- For $\Delta_2 > \Delta_1$, it follows that the protocol satisfies both mutual exclusion and deadlock freedom.
- Mutual Exclusion: $\neg(P.mode = \text{Crit} \wedge P'.mode = \text{Crit})$ is an invariant for every pair of processes P and P'.
- Deadlock Freedom: If $P.mode = \text{Test}$ for a process P, then eventually $P'.mode = \text{Crit}$ for some process P'.

Summary

- Timed Model (1)
  - Clock Variables, Timed Actions
  - Composition of Timed Processes
  - Timing-Based Mutual Exclusion: Fischer's Protocol