The UPPAAL modeling and verification environment

Dott. Luigi Di Guglielmo

Prof. Tiziano Villa

University of Verona

Dep. Computer Science

Italy

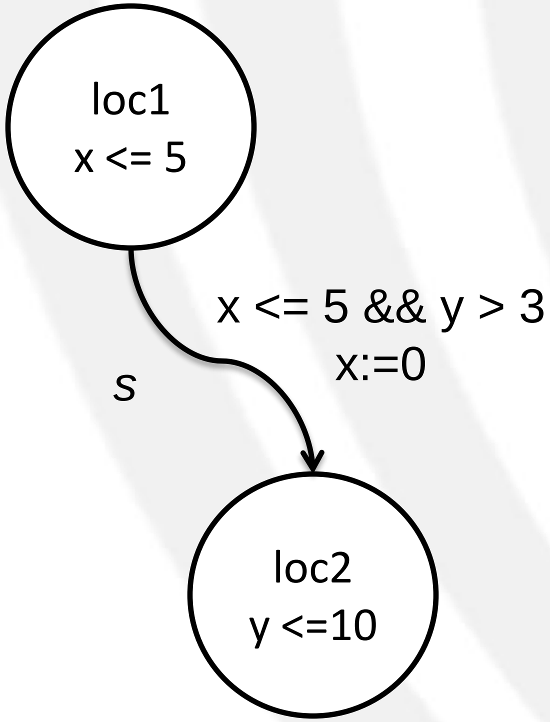
Outline

- Introduction
- Timed Automata in Uppaal
- Understanding Time
- Query Language in Uppaal
- Overview of the Uppaal toolkit
- Example
- Conclusion

Introduction

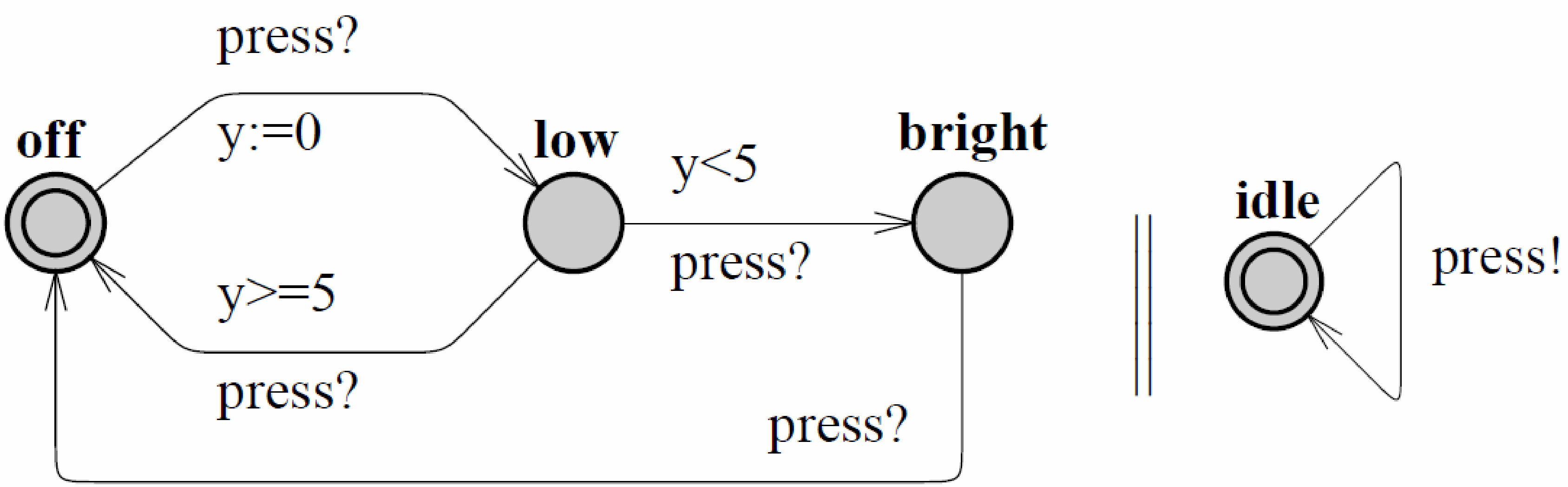
- *Uppaal* is a toolbox for modeling, simulating and verifying real-time systems
- It is jointly developed by **Upp**sala University (Sweden) and **Aal**borg University (Denmark)
- The tool is designed for real-time systems that can be modeled as networks of *timed automata*
- 3 components:
  - System editor: graphical user interface to build a system
  - Simulation: step by step movement through system
  - Verification: model checker evaluates questions in temporal logic

Timed Automata



- A timed automaton is a finite state machine extended with clock variables
  - Locations
    - Invariants
  - Edges
    - Guards
    - Resets
    - Synchronization labels
  - Clocks
    - Variables evaluate real numbers

Timed Automata in Uppaal (I)



a) Lamp

b) User

Timed Automata in Uppaal (II)

- Templates
  - Used to define automata
  - Characterized by a set of parameters
- Constants
  - Declared as **`const`** *`type name = value;`*
- Bounded integer variables
  - Declared as **`int [min,max]`** *`name;`*
- Boolean variables
  - Declared as **`bool`** *`name;`*
- Clock variables
  - Declared as **`clock`** *`name;`*

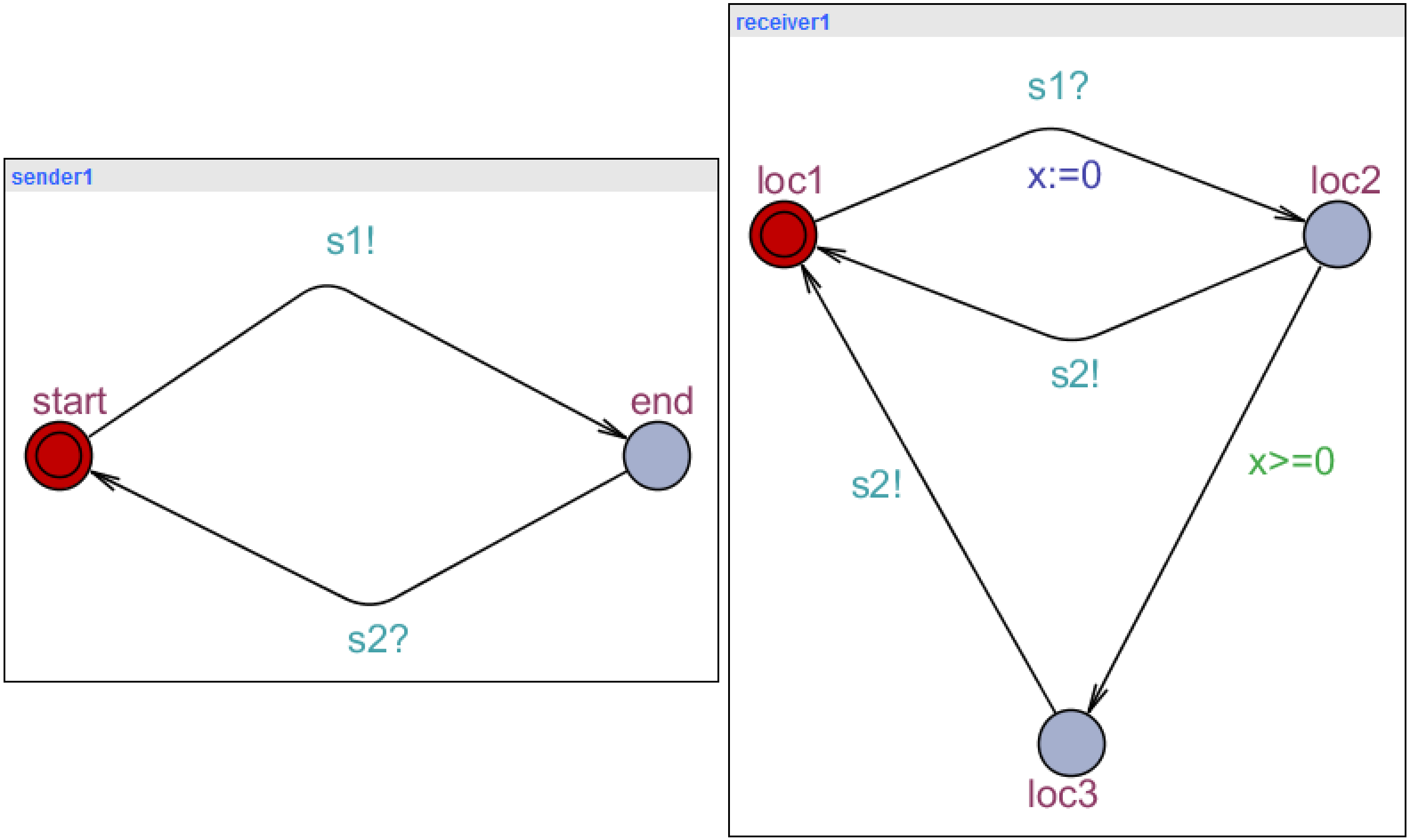
Timed Automata in Uppaal (III)

- Normal locations
  - Time can pass until the invariant is unsatisfied
    - non-determinism: invariants and guards on outgoing edges may not be disjoint
  - When the invariant is unsatisfied the location must be exited
- Urgent locations
  - Time cannot pass (must leave it immediately)
  - Semantically equivalent to adding an extra clock `x` that is reset on all incoming edges into the location and label the latter with the invariant `x <= 0`
- Committed locations
  - A committed location is an urgent location and one of its active edges must be taken as first (meaningful for a composition of automata)

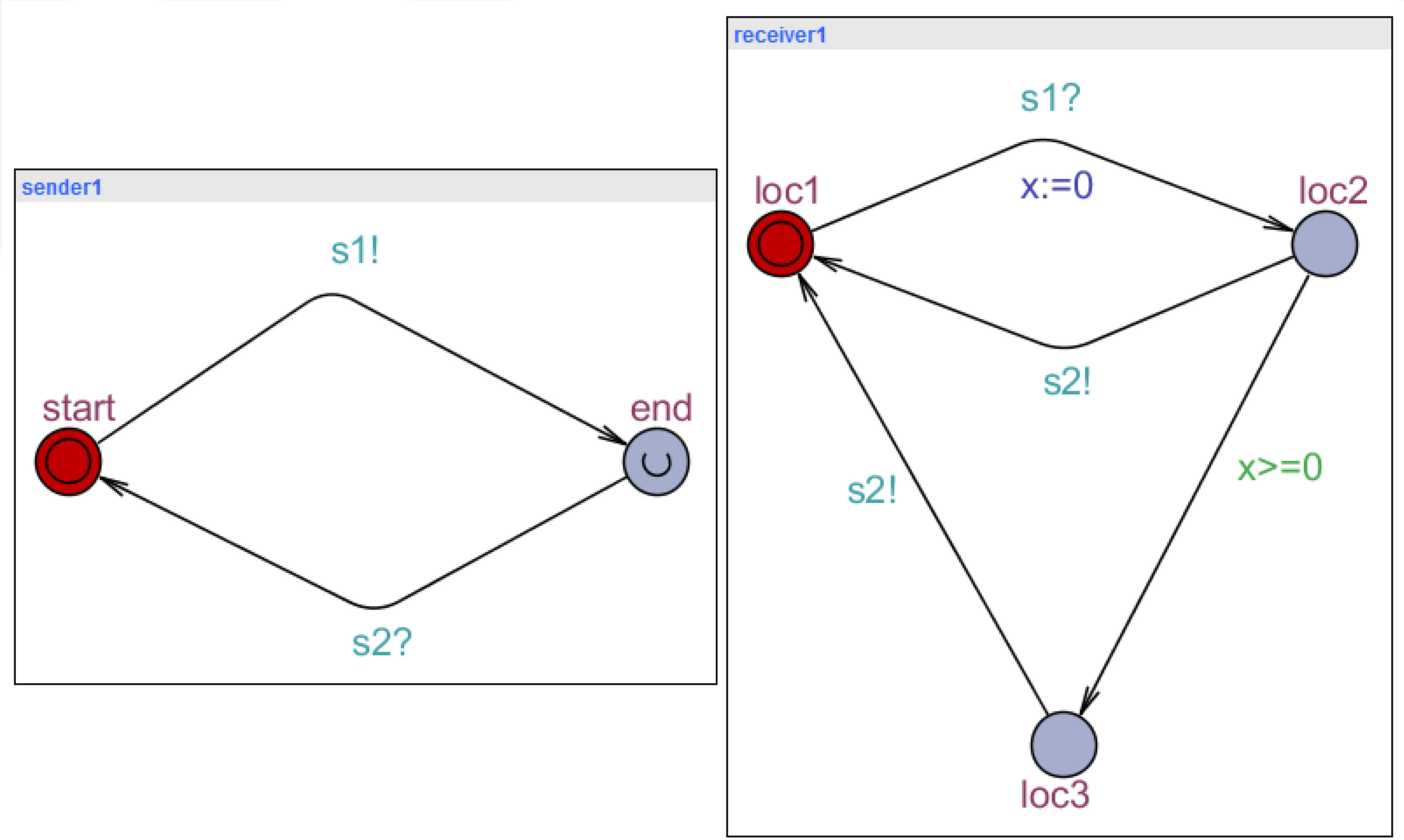
Timed Automata in Uppaal (IV)

- More on committed locations
  - A state is committed if any of the locations in the state is committed
  - A committed state cannot delay and the next transition must involve an outgoing edge of at least one of the committed locations

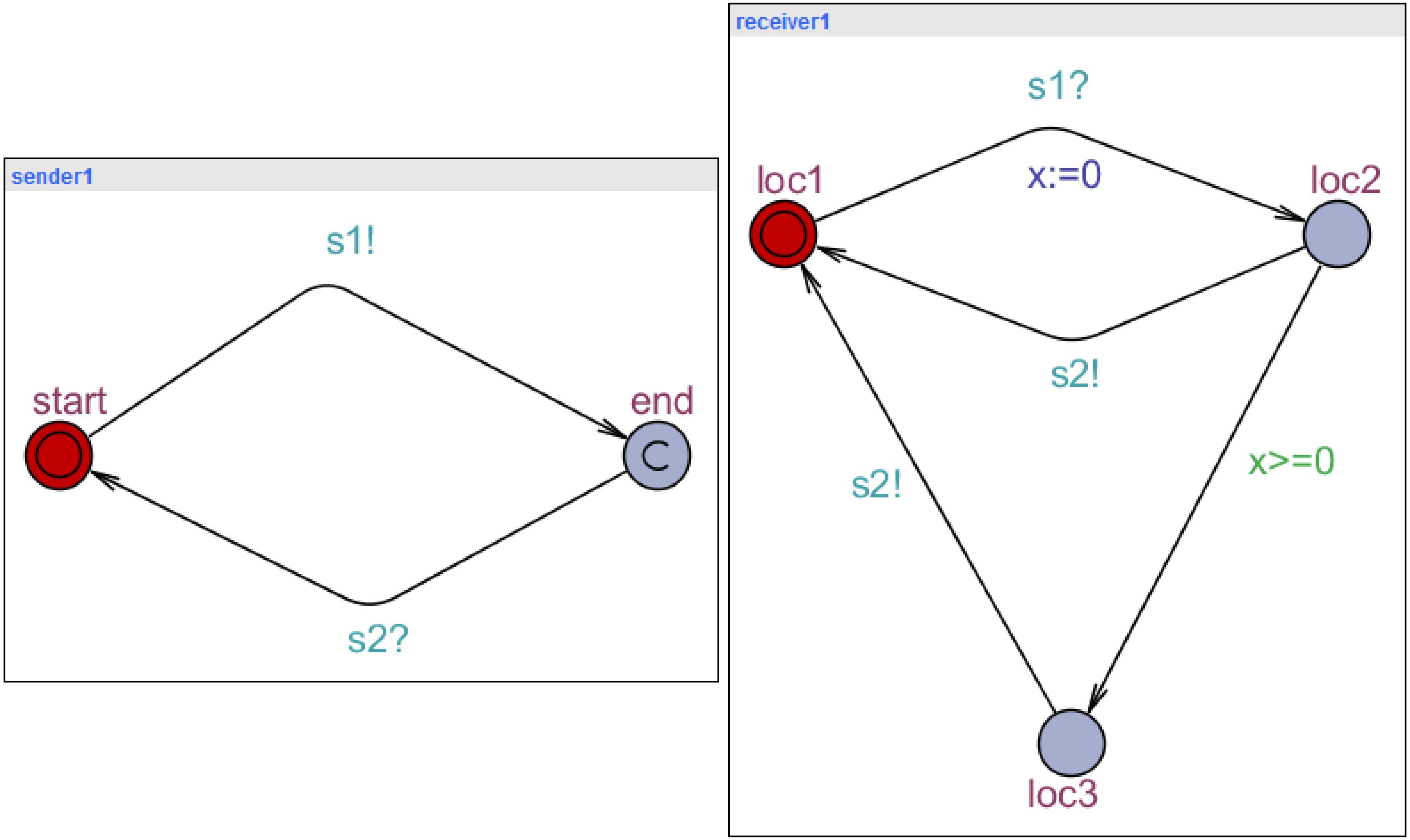
Normal Locations Example



Urgent Locations Example

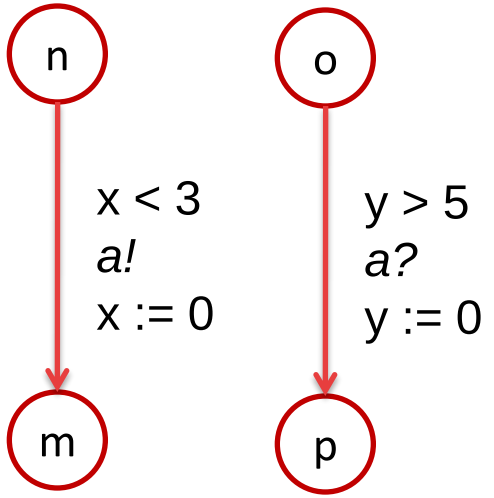


Committed Location Example



Synchronization Semantics in Timed Automata

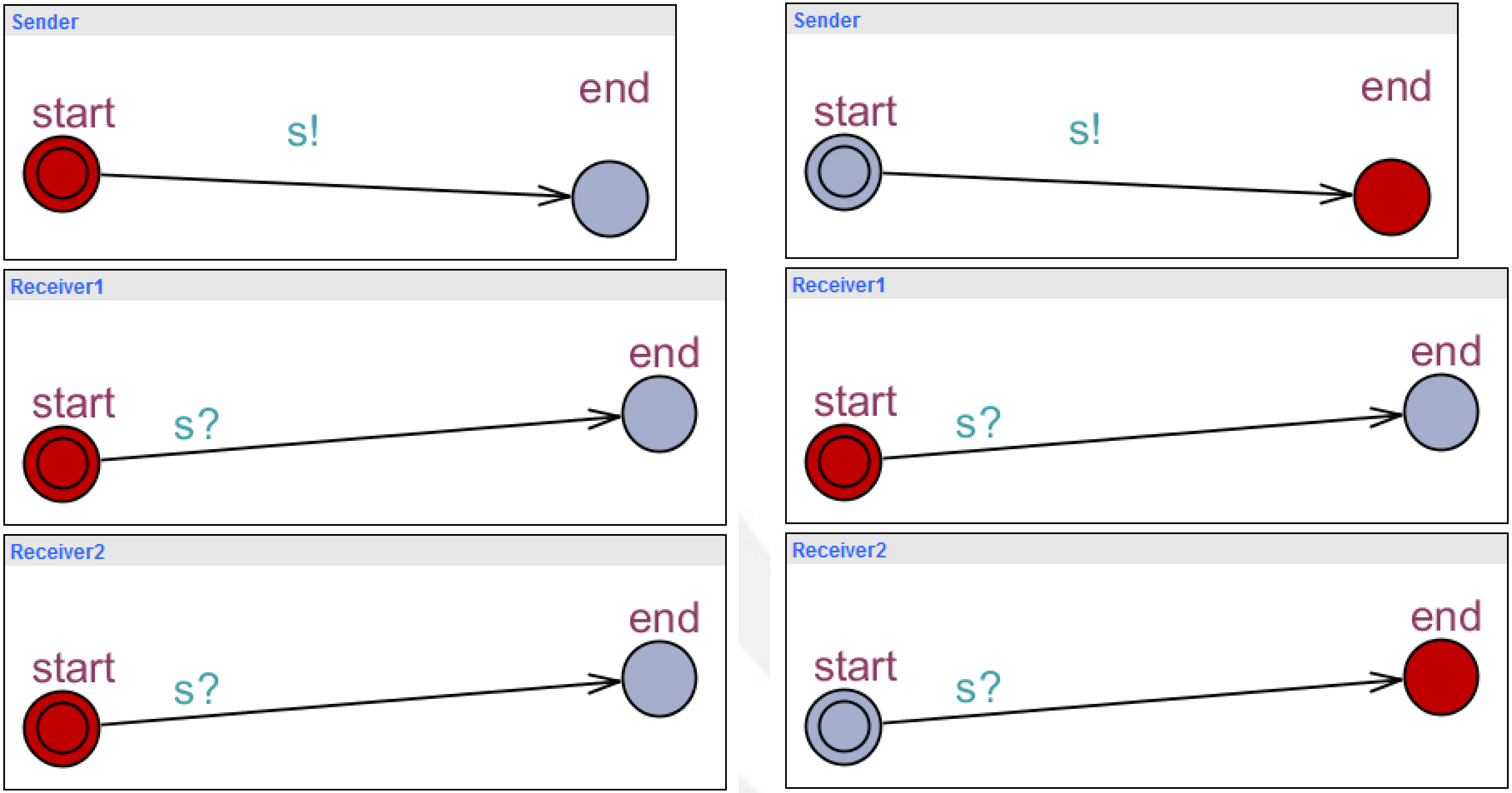
- Semantics:
  - Transitions with the same synchronization channel are activated simultaneously
    - Guards must be true
    - The event *a* is exchanged between automata
    - Resets are executed



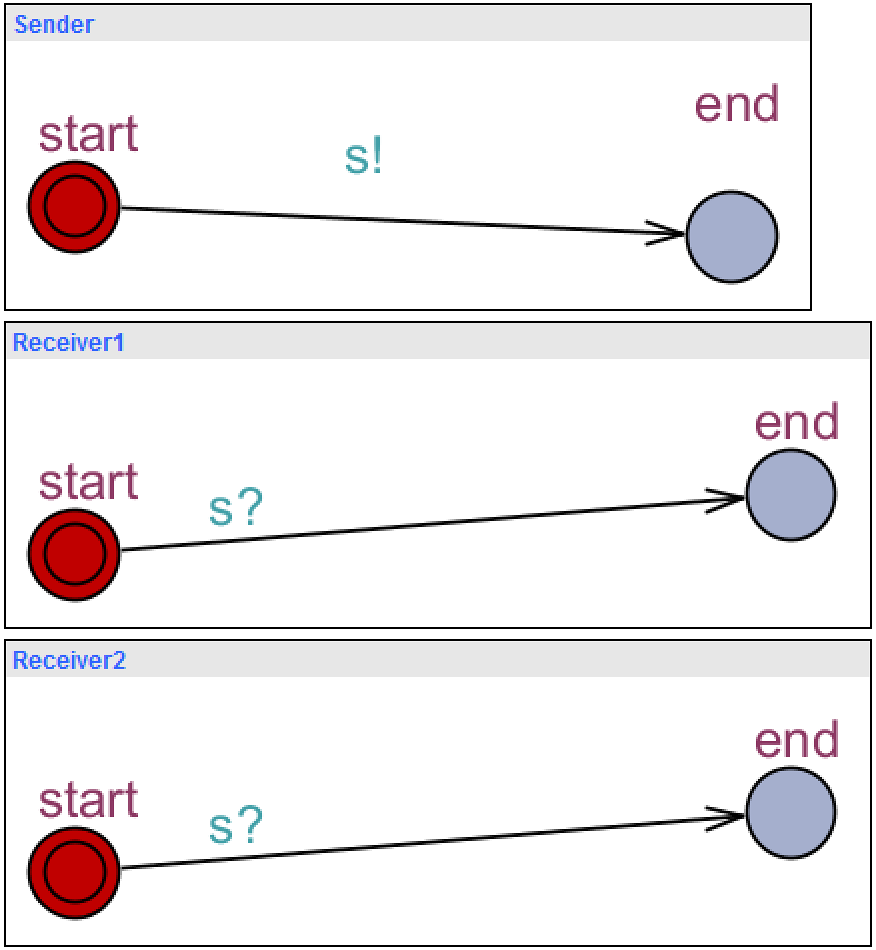
Synchronization in Uppaal (I)

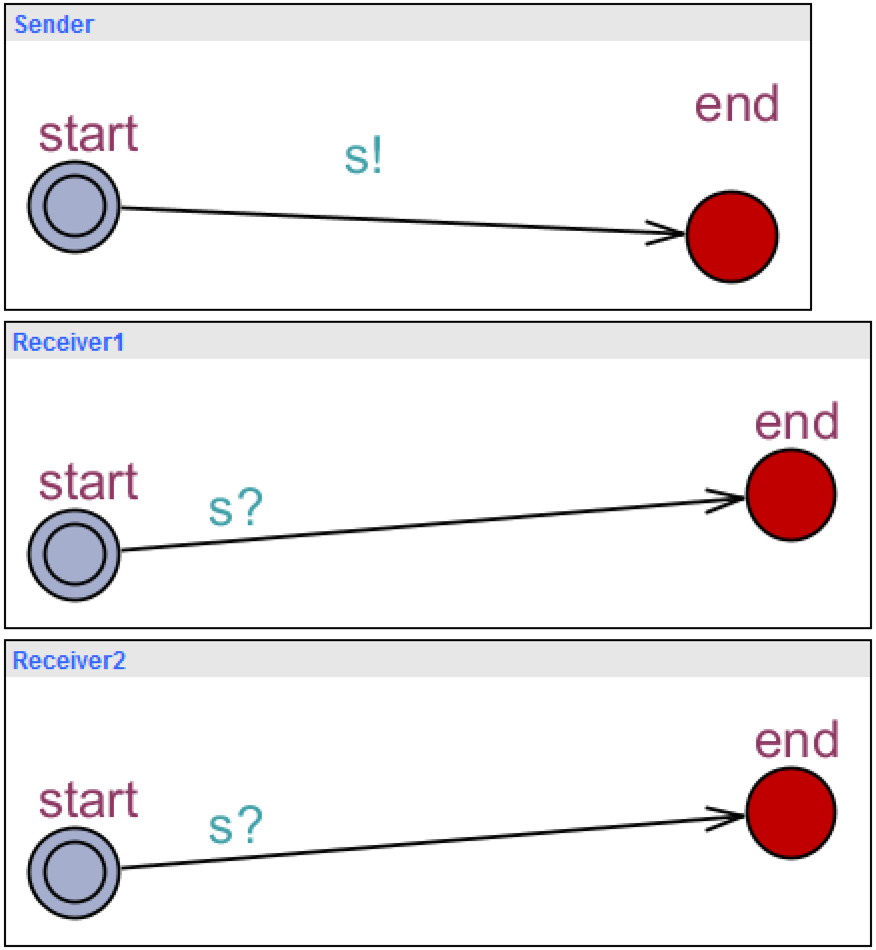
- Binary channels
  - Declared as **`chan c`**
  - An edge labeled with **`c!`** synchronizes with another labeled **`c?`**
- Broadcast channels
  - Declared as **`broadcast chan c`**
  - An edge labeled with **`c!`** synchronizes with an arbitrary number of receivers **`c?`**
- Urgent channels
  - Declared with the keyword **`urgent`**
  - Delays are not admitted in the current location if a synchronization on an urgent channel is enabled
  - Caveat
    - Leave location non-deterministically
    - Guards on the edges labeled with urgent channels are not allowed

Binary Channel Example

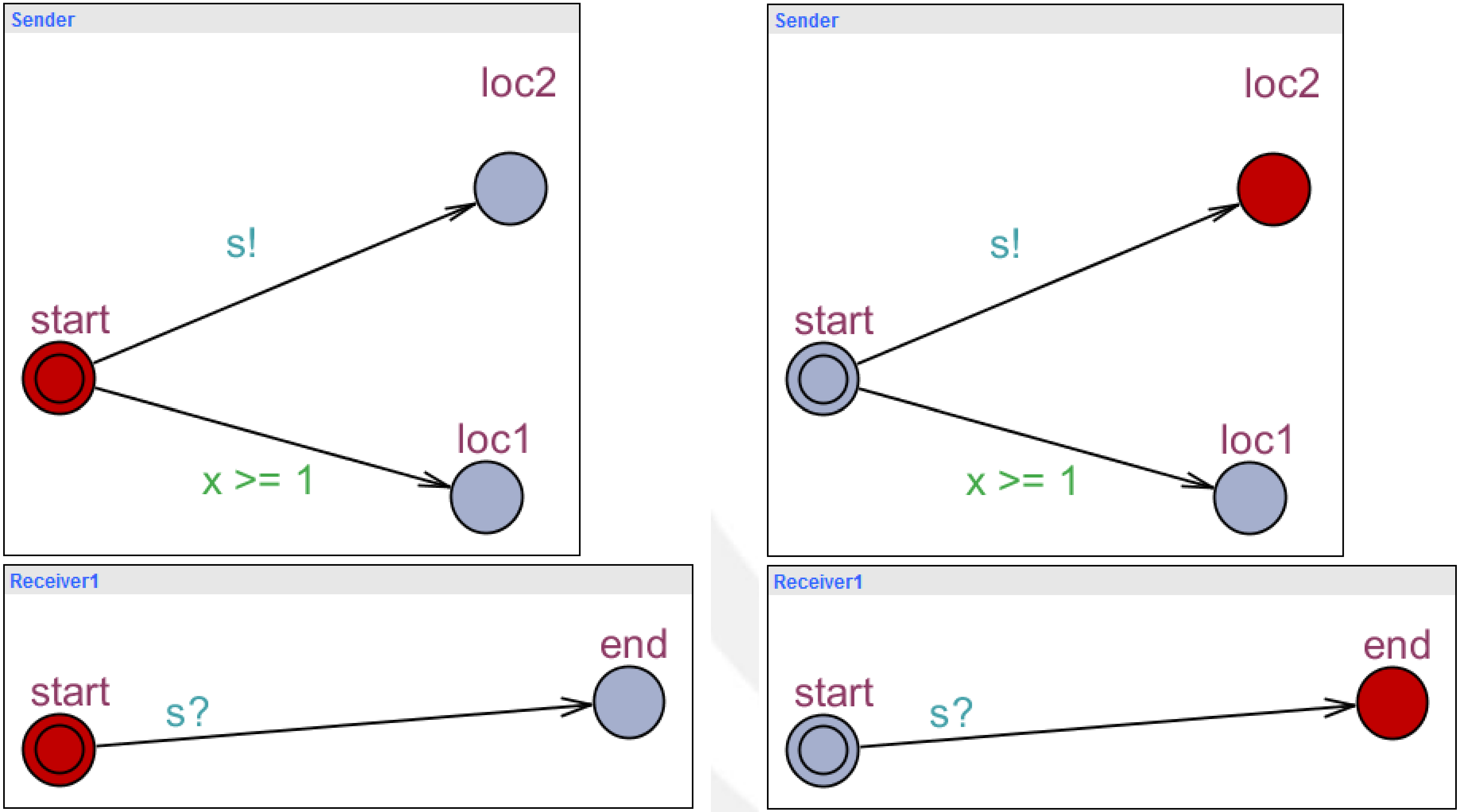


Broadcast Channel Example





Urgent Channel example

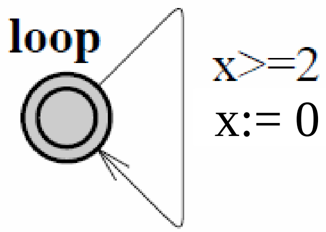


Expressions in Uppaal (I)

- Expressions range over clocks and integer variables
- Guard
  - Boolean expression
  - Clocks are compared only to integer expressions
- Synchronization
  - A synchronization label is either of the form ***expression*!** or ***expression*?**
  - In this case the expression must evaluate to a channel
- Assignment
  - Expressions on clocks and variables
  - Clocks are assigned with integer valuated expression
- Invariant
  - Expression on clocks and variables of the form **x < e** or **x<= e**

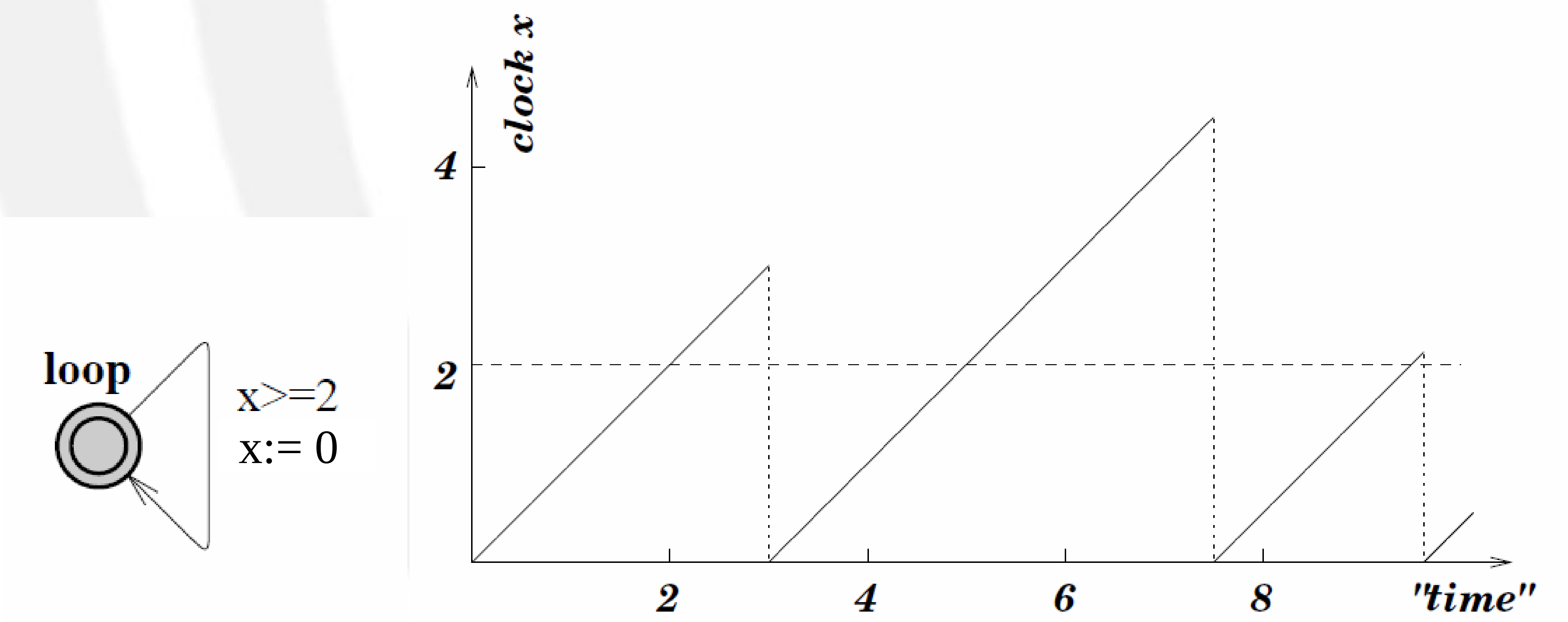
Understanding Time (I)

- Uppaal uses a continuous time model
- Let’s consider the following example



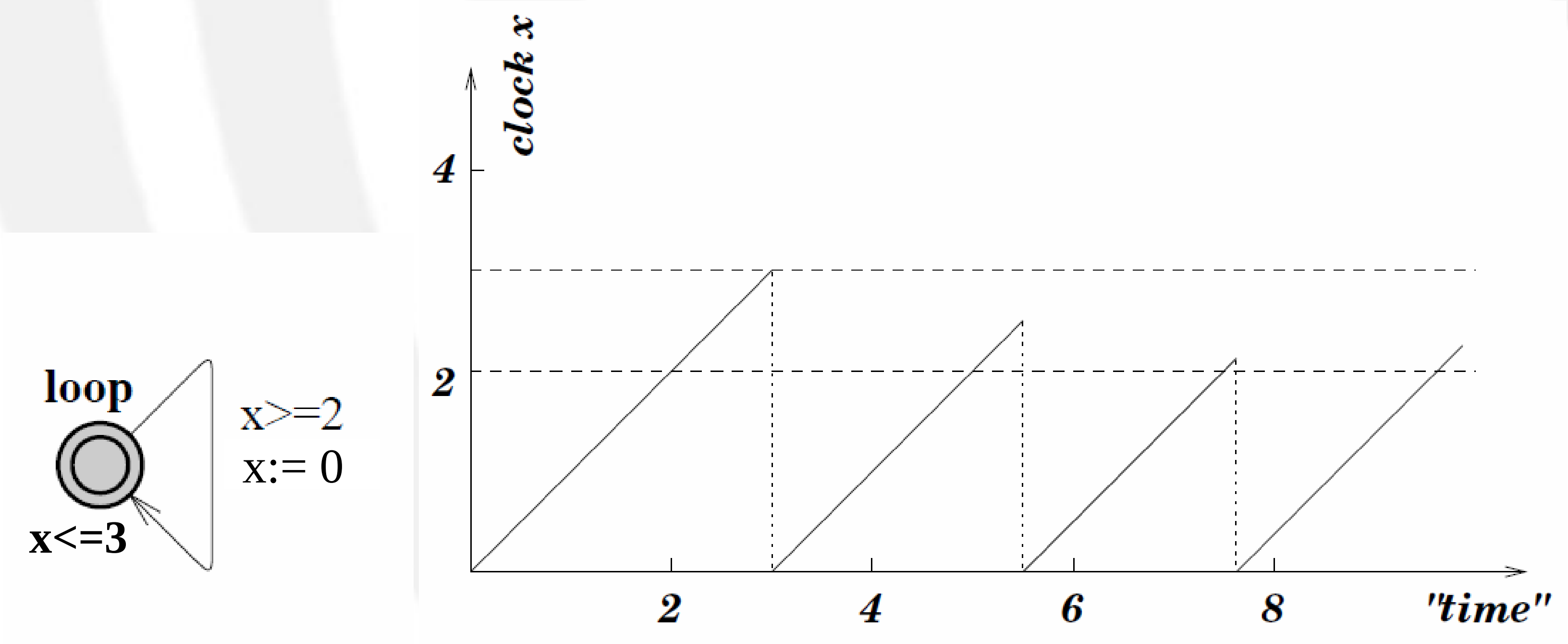
  - `x` is a clock
  - `x := 0` means “the clock is reset”

Understanding Time (I)



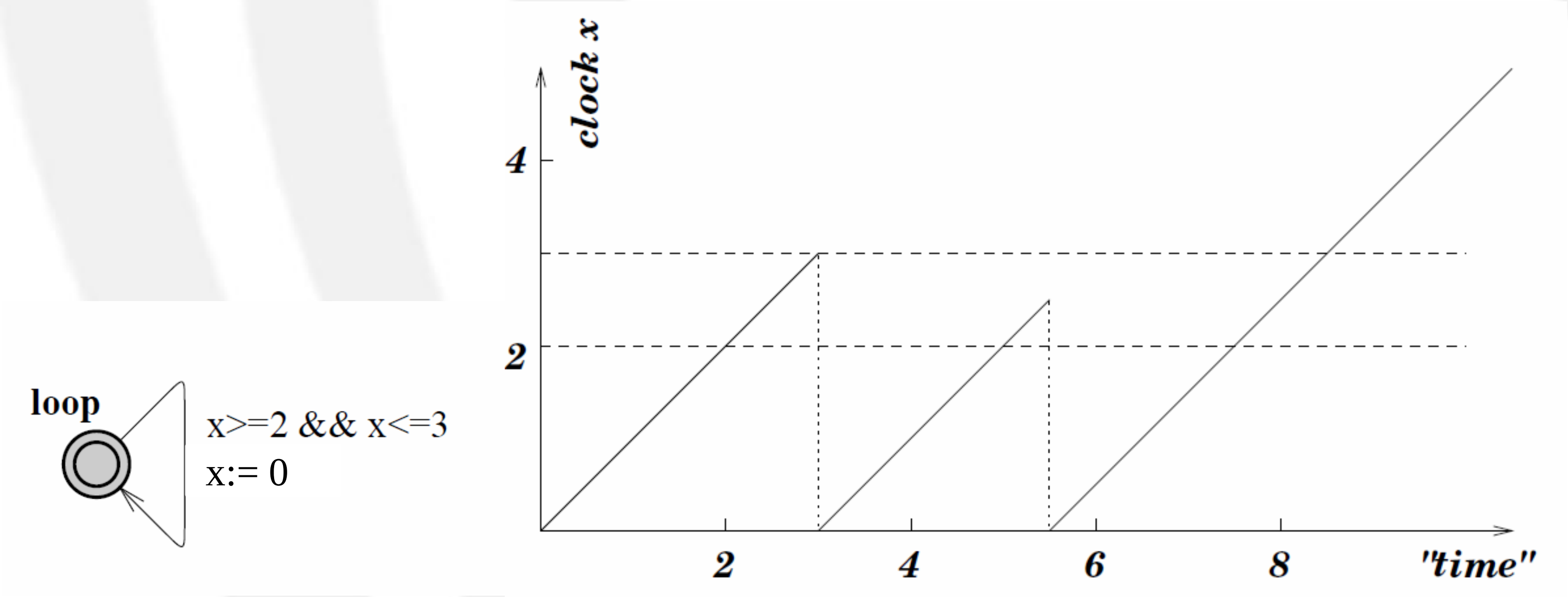
- The transition can be taken after 2 seconds

Understanding Time (II)



- The transition can be taken after 2 seconds
- The transition must be taken within 3 seconds

Understanding Time (III)

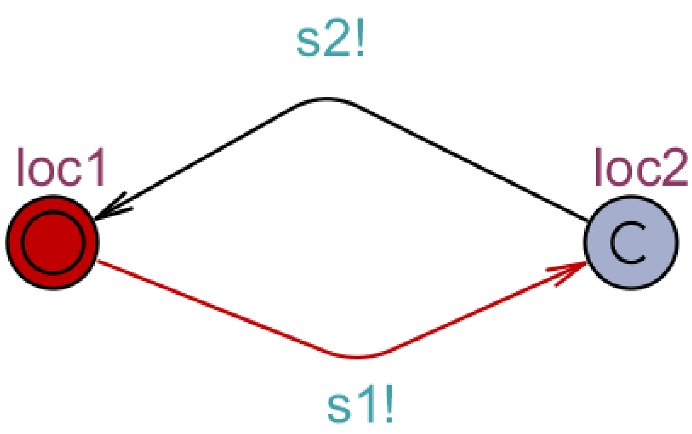


- The transition can be taken between 2 and 3 seconds
- When $x > 3$ let time pass, no transition can be taken

MODELING PATTERNS

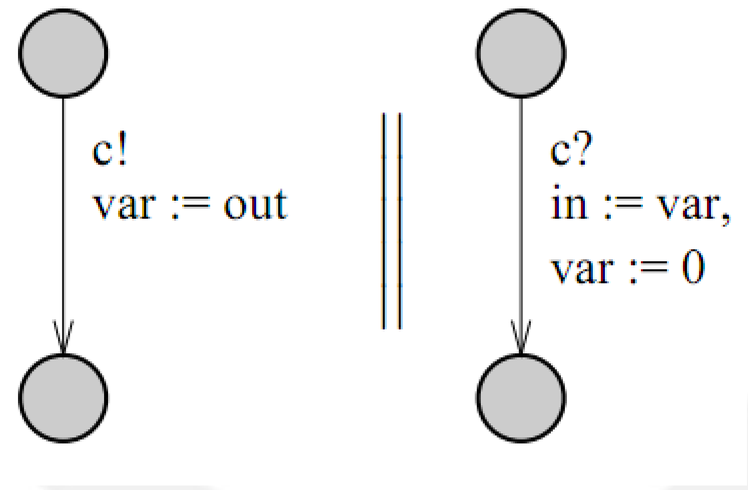
Atomicity (I)

- Some times it is necessary to model atomic behaviors
- How to model atomicity in Uppaal?
  - Committed locations
    - When a committed location is entered the execution flow must continue through such a location



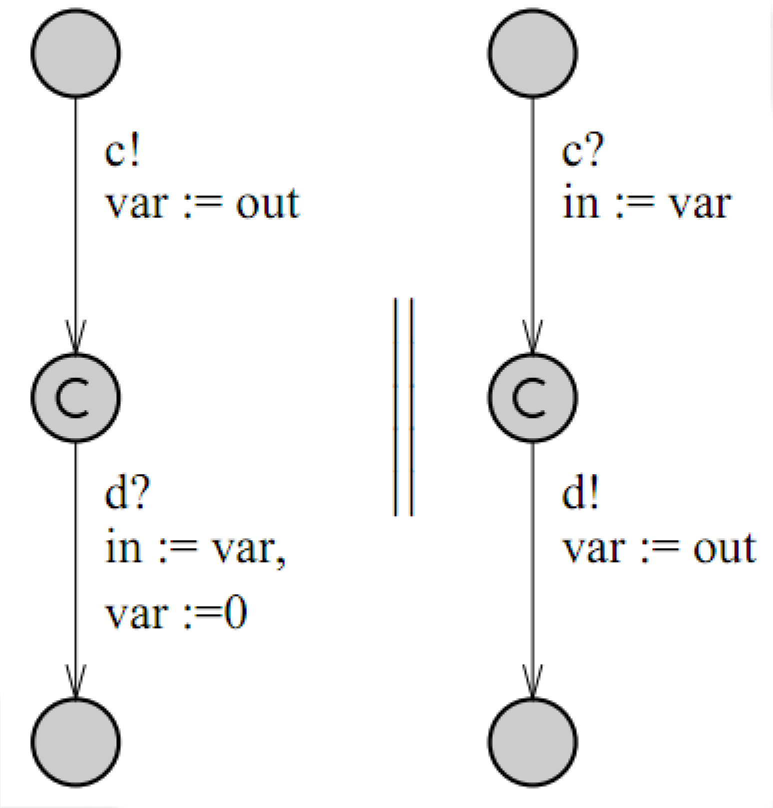
Synchronous Value Passing: one-way (II)

- `c` is a binary channel
- `var` is a shared variable (i.e., global)
- `in` and `out` are local variables
- Resets
  - The resets of the sender is executed before the resets of the receiver
  - Given a list of reset statements, these are executed sequentially



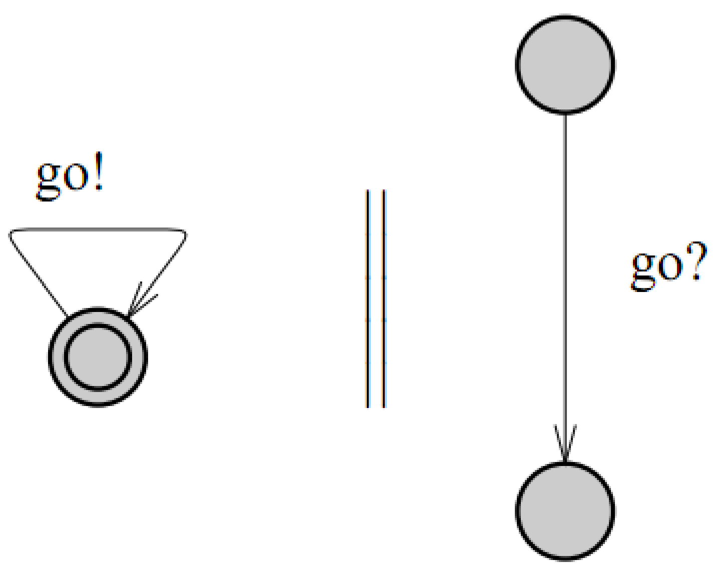
Synchronous Value Passing: two-way (III)

- `c` is a binary channel
- `var` is a shared variable (i.e., global)
- `in` and `out` are local variables

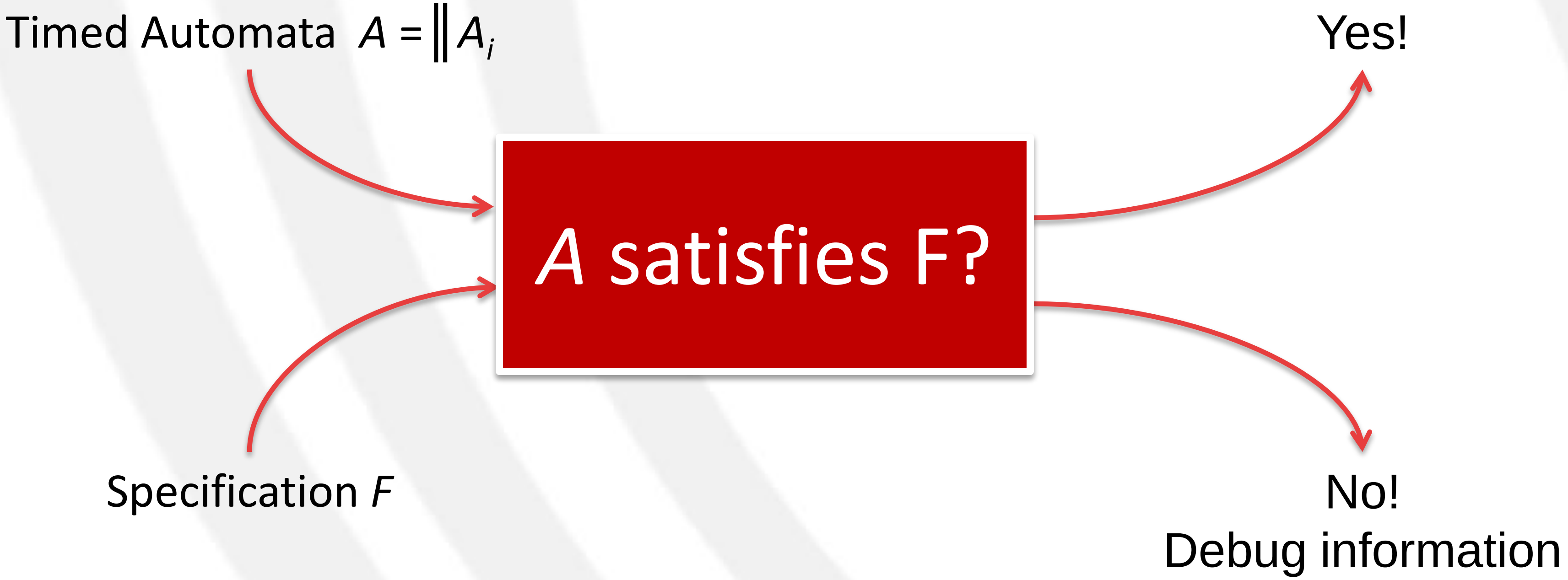


Urgent Edges (IV)

- Uppaal provides
  - Urgent locations
  - Urgent channels
- How to model urgent edges?
  - go is declared as an urgent channel
  - Adding an automaton with one location with a self-loop labeled with the urgent channel go



Model Checking



The Query Language

- The main purpose of a model checker is verify the model w.r.t. a requirement specification
- Uppaal uses a simplified version of CTL for defining the specifications
- The query language consists of
  - State formulae
    - Describe individual states
  - Path formulae
    - Quantify over paths of the model
      - Reachability
      - Safety
      - Liveness

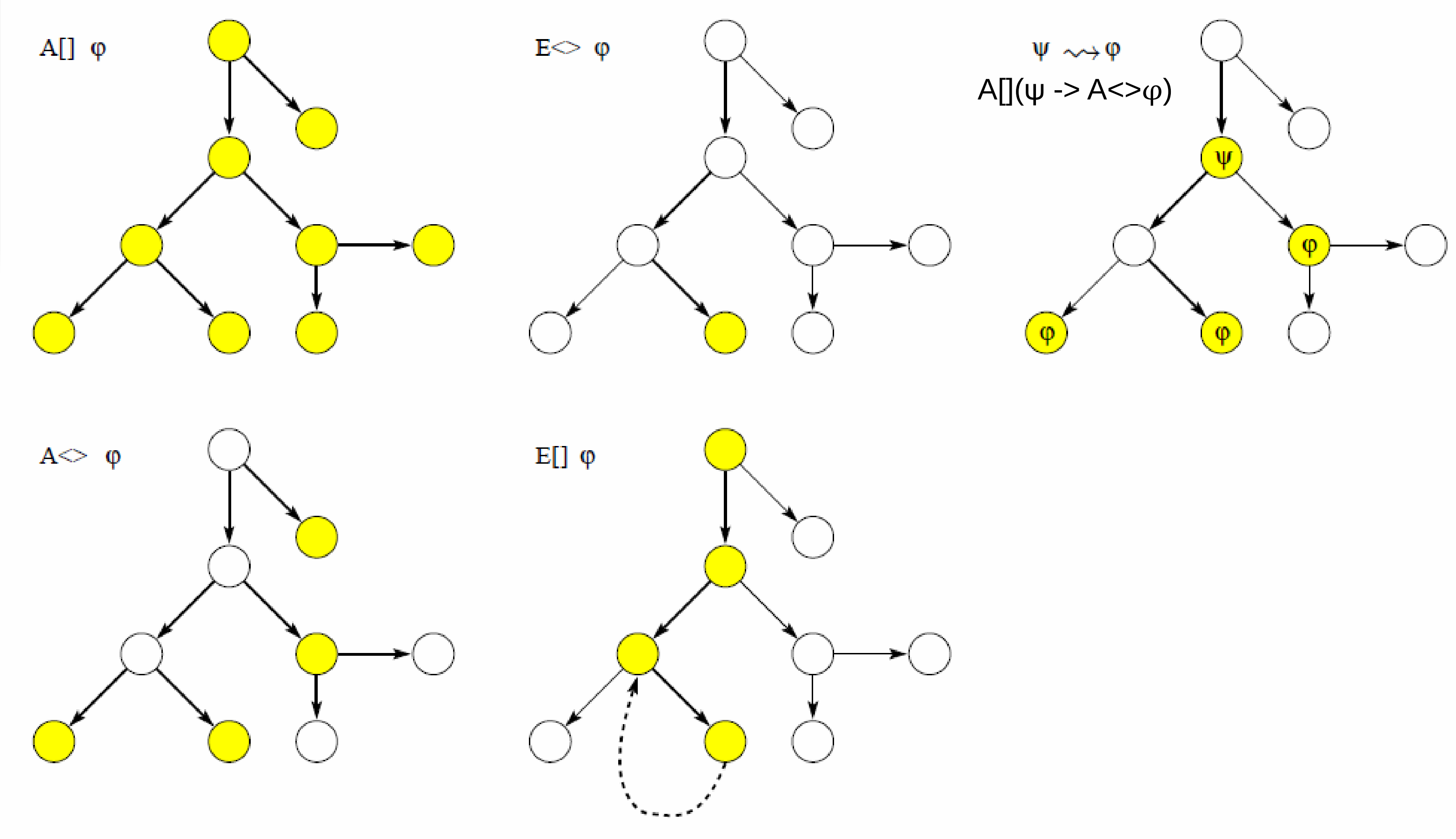
State Formulae

- A *state formula* is an expression that can be evaluated for a single state

SF ::= `Proc.loc` | `deadlock` |
`x == n` | `x<=n` | `x < n` | `x > n` | `x >= n` |
SF `and` *SF* | *SF* `or` *SF* |
SF `imply` *SF* | `not` *SF*

- where
  - `x` is a clock or a discrete variable
  - `n` is an integer

Path Formulae (I)



Path Formulae (II)

- Reachability properties
  - Reachability properties ask whether a given state formula φ *possibly* can be satisfied by any reachable state
  - E.g.:
    E<> φ
- Safety properties
  - Safety properties are on the form: “something bad will never happen”
  - E.g.:
    A[] φ , E[] φ
- Liveness properties
  - Liveness properties are on the form: “something will eventually happen”
  - E.g.:
    A<> φ , φ --> ψ

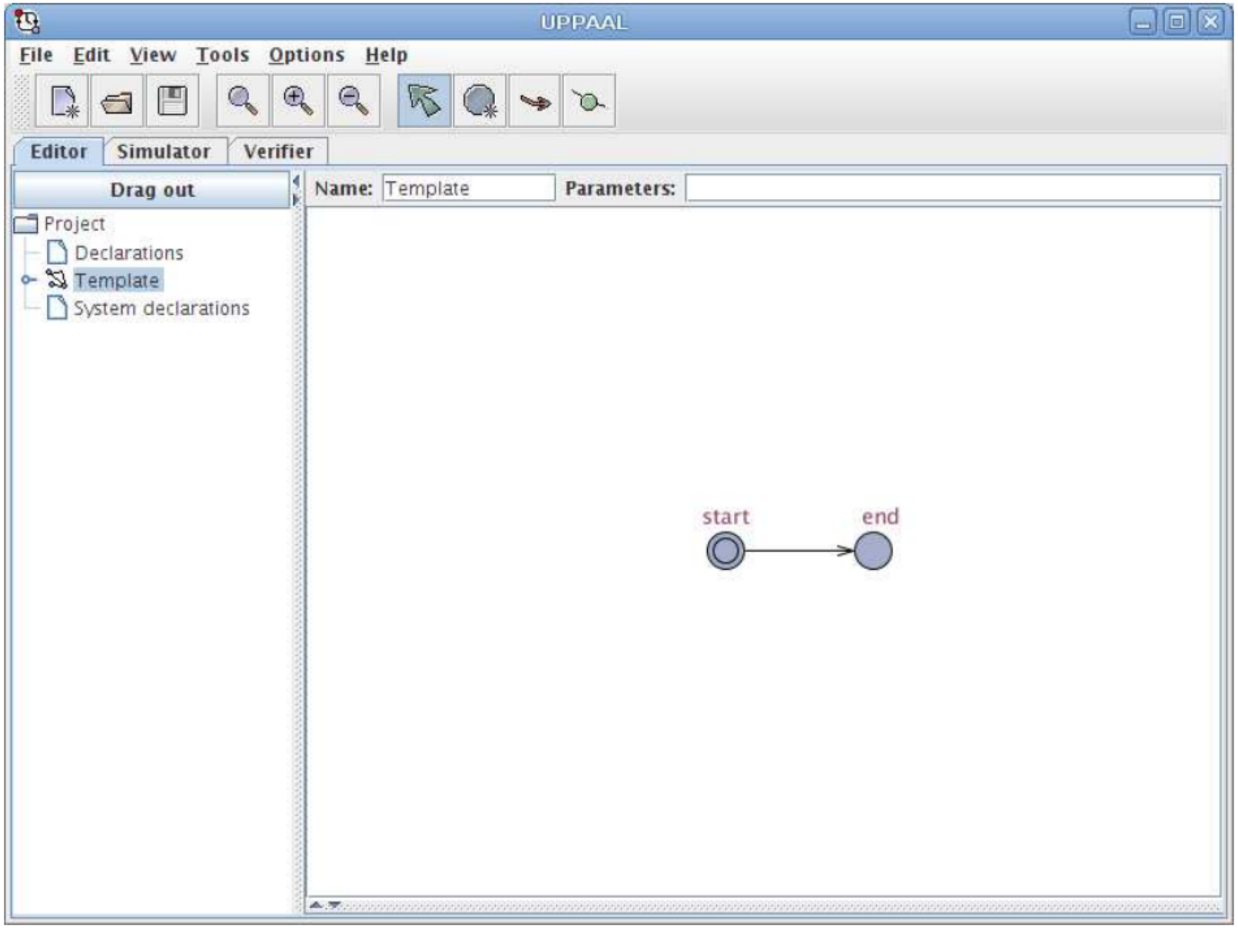
Queries Examples (I)

- A deadlock never occurs
  - A[] not deadlock
- An automaton *A1* remains into a state *q* for at least 10 seconds
  - E<> A1.q and x > 10
- An automaton *A2* may never enter a state *q*
  - E[] not A2.q

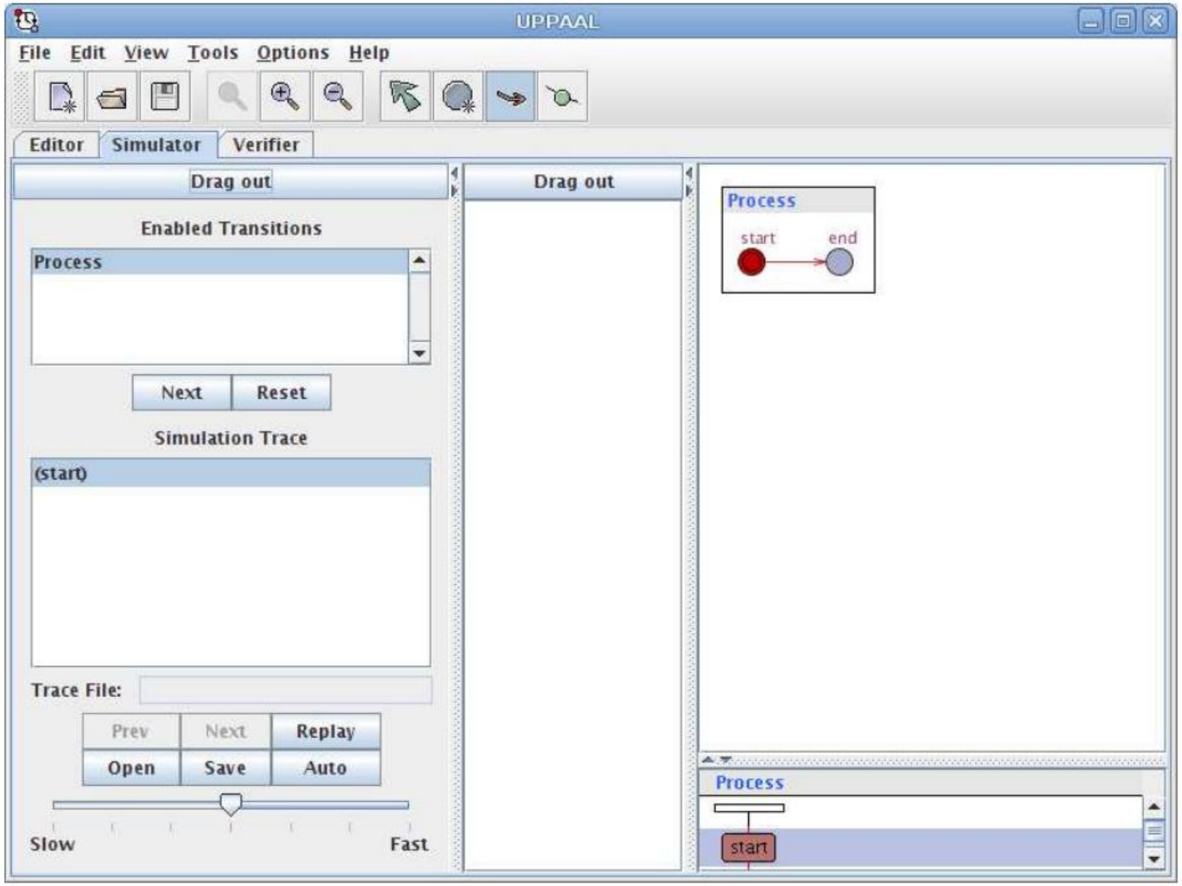
Queries Examples (II)

- Nothing bad can happen
  - A[] φ
- Infinitely often φ (i.e., it is repeatedly satisfied)
  - A[]A<> φ
- Always φ is possible
  - A[]E<> φ
- There exists a state from which φ always holds
  - E<>A[] φ

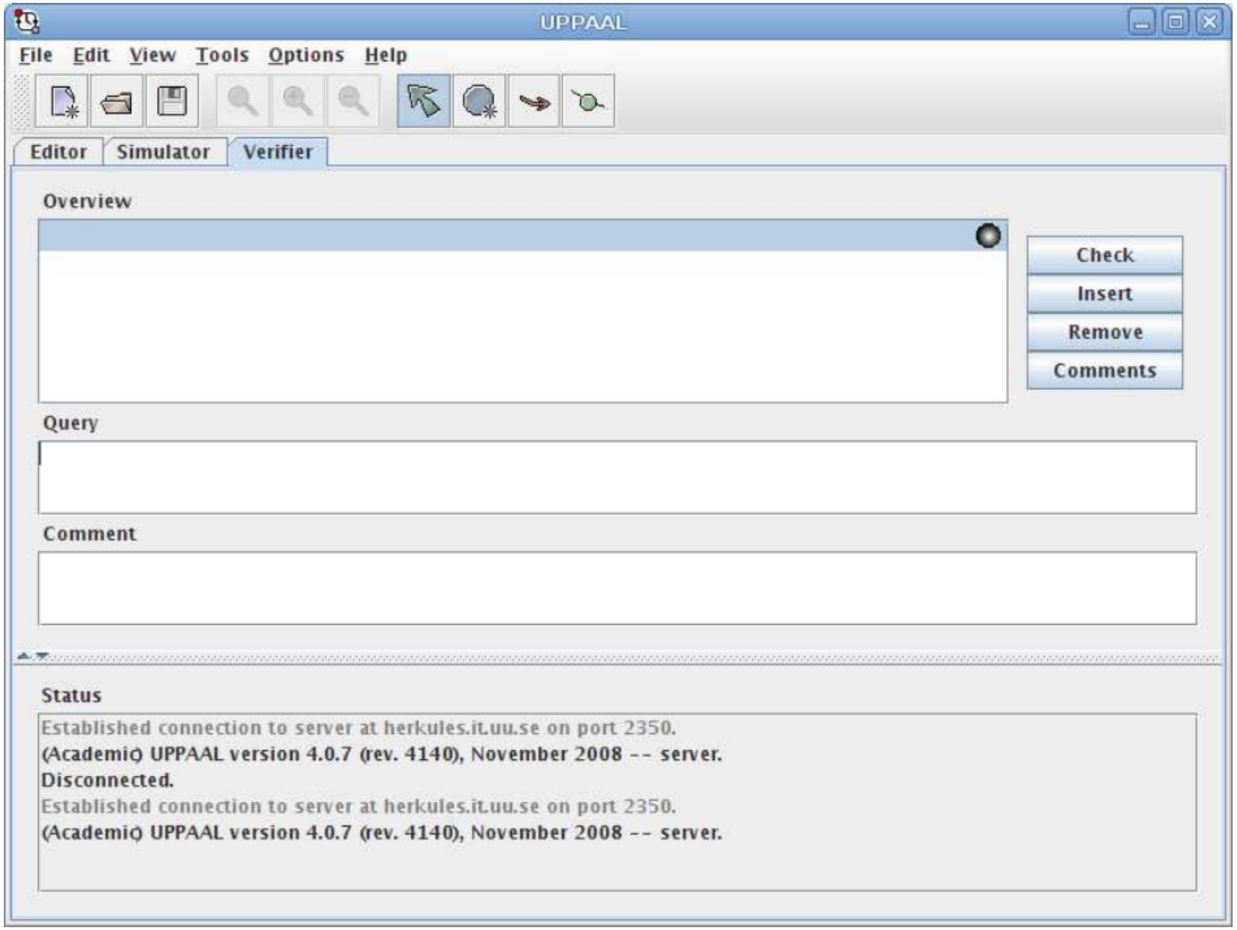
Overview of the Uppaal toolkit (I)



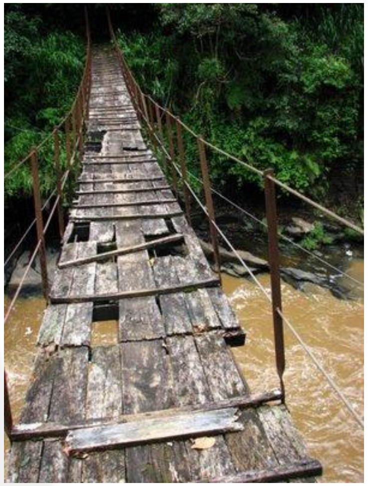
Overview of the Uppaal toolkit (II)



Overview of the Uppaal toolkit (III)

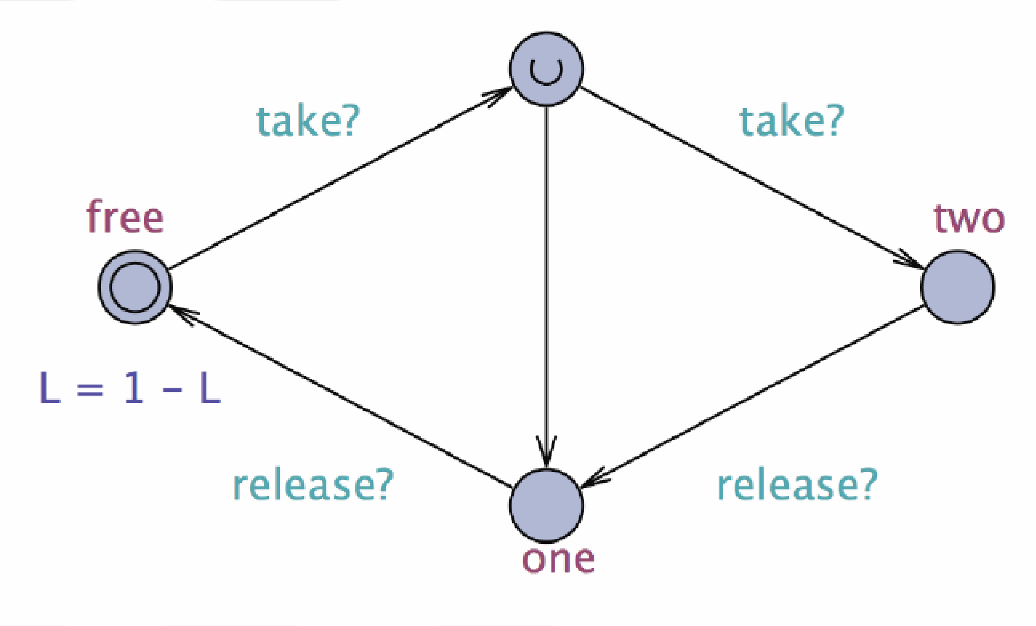


The Vikings Example (I)



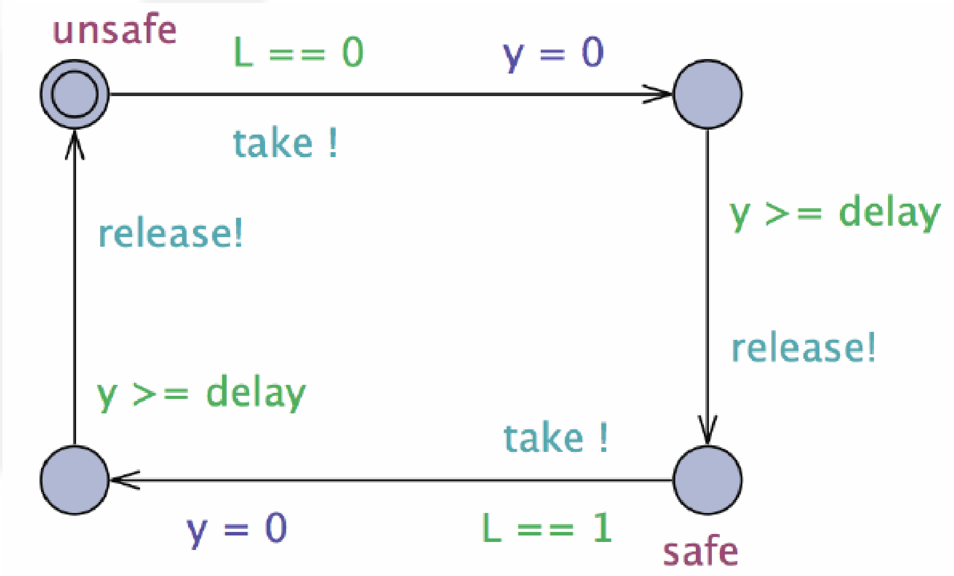
- 4 Vikings cross the bridge in the middle of the night
  - Every Viking takes a different time to cross the bridge (i.e., one Viking can be faster than another)
- The bridge can carry only 2 Vikings at the same time
- Vikings need a torch to cross and they have only one
- Can the Vikings get safe within 60 minutes?

The Vikings Example: the Torch model



- L represents the side the torch is on:
  - If L == 0 then the torch is on this side of the bridge
  - If L == 1 then the torch is beyond the bridge

The Vikings Example: the Viking model



- **cost int** `delay;` represents the time required by the Viking to pass the bridge
- **clock** `y;` is an internal clock of the automaton

The Vikings Example: the system model

- Global variables:
  - *Declarations*

```
chan take, release;      // Take and release torch
int[0,1] L;              // The side the torch is on
clock time;              // Global time
```

- System variables:
  - *System declarations*

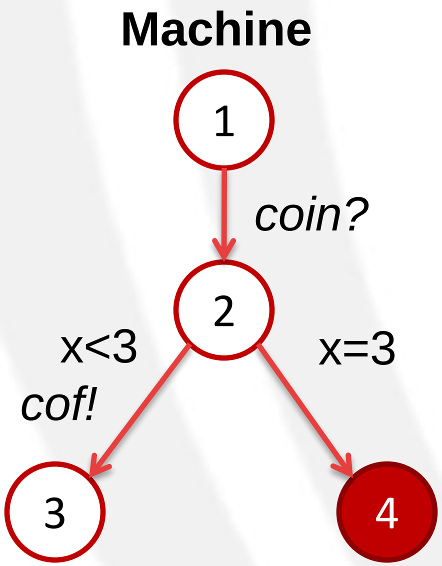
```
Viking1 = Soldier(fastest);
Viking2 = Soldier(fast);
Viking3 = Soldier(slow);
Viking4 = Soldier(slowest);

system Viking1, Viking2, Viking3, Viking4,
Torch;
```

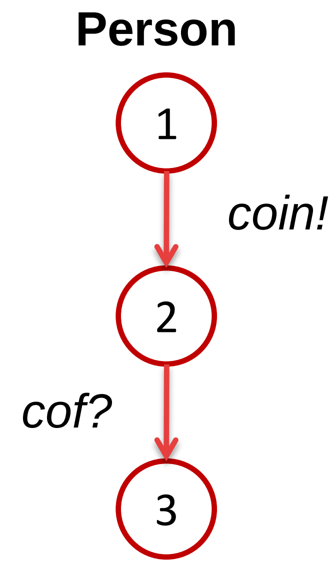
The Vikings Example: Exercises

- Which is the minimal time required to let every Viking cross the bridge?
  - Use the verification functionalities of Uppaal
- Change the example:
  - Adding new Vikings
  - Adding a torch
  - Allowing 3 Vikings to bring the torch

EXERCISES

Example: the Vending Machine





- *coin* and *cof* are synchronization channels
- Person
  - Puts a coin into the machine (*coin!*) and waits for the coffee (*cof?*)
- Machine
  - Accepts the coin (*coin?*) and, within 3 seconds, prepares the coffee (*cof!*), otherwise enters an error state

The Vending Machine (I)

- Modeling a vending machine
  - A bottle of coke costs 5 coins
  - The user inserts coins (CoinIn) and then presses the "RequestCan" button or "Cancel"
  - If "Cancel" is pushed the machine returns the inserted coins (CoinOut)
  - If "RequestCan" is pushed and the credit is correct, the machine returns the bottle and the change (if necessary)
  - The machine requires between 3 to 5 seconds for issuing a bottle
  - The user cannot insert more than 10 coins without pushing a button

The Vending Machine (II)

- Define the automata “Machine” and “User” according to the previous specifications
- Check that:
  - The system does not allow deadlocks
  - If the credit is correct, the machine releases a bottle of coke within 5 seconds after the user pushes the “RequestCan” button
  - If the User pushes “Cancel”, the machine returns the coins (if any)
- Hint
  - Start with a simple model in which a bottle costs 1 coin. Then continue with the complex model