Testing Real-Time Embedded Systems Using UppAal-TRON
-Tool and Application

Kim G. Larsen, Marius Mikucionis,
Brian Nielsen, Arne Skou

Aalborg University, DK

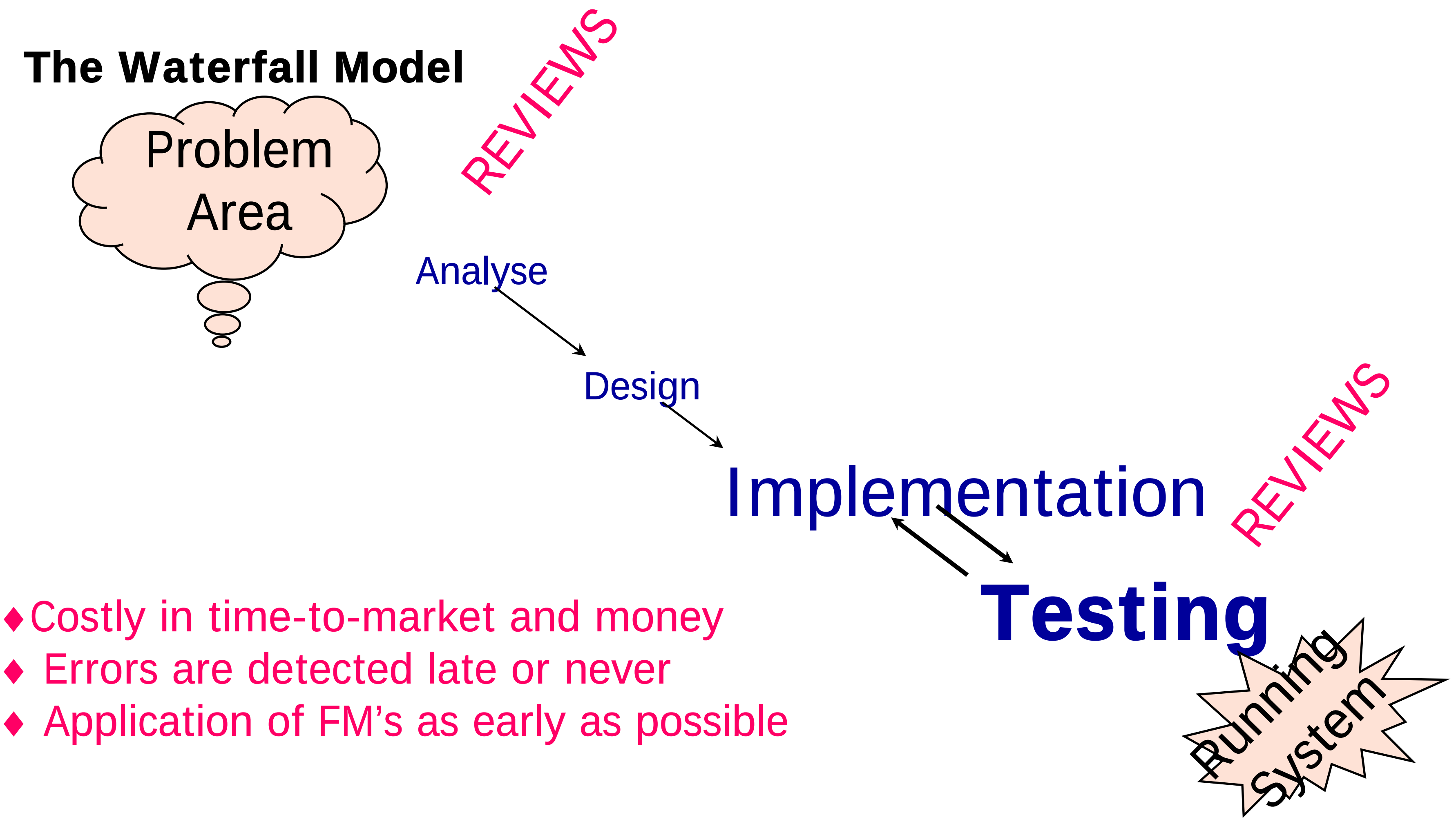
{kgl | marius | bnielsen | ask}@cs.aau.dk

AALBORG UNIVERSITY DENMARK

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Traditional Software Development

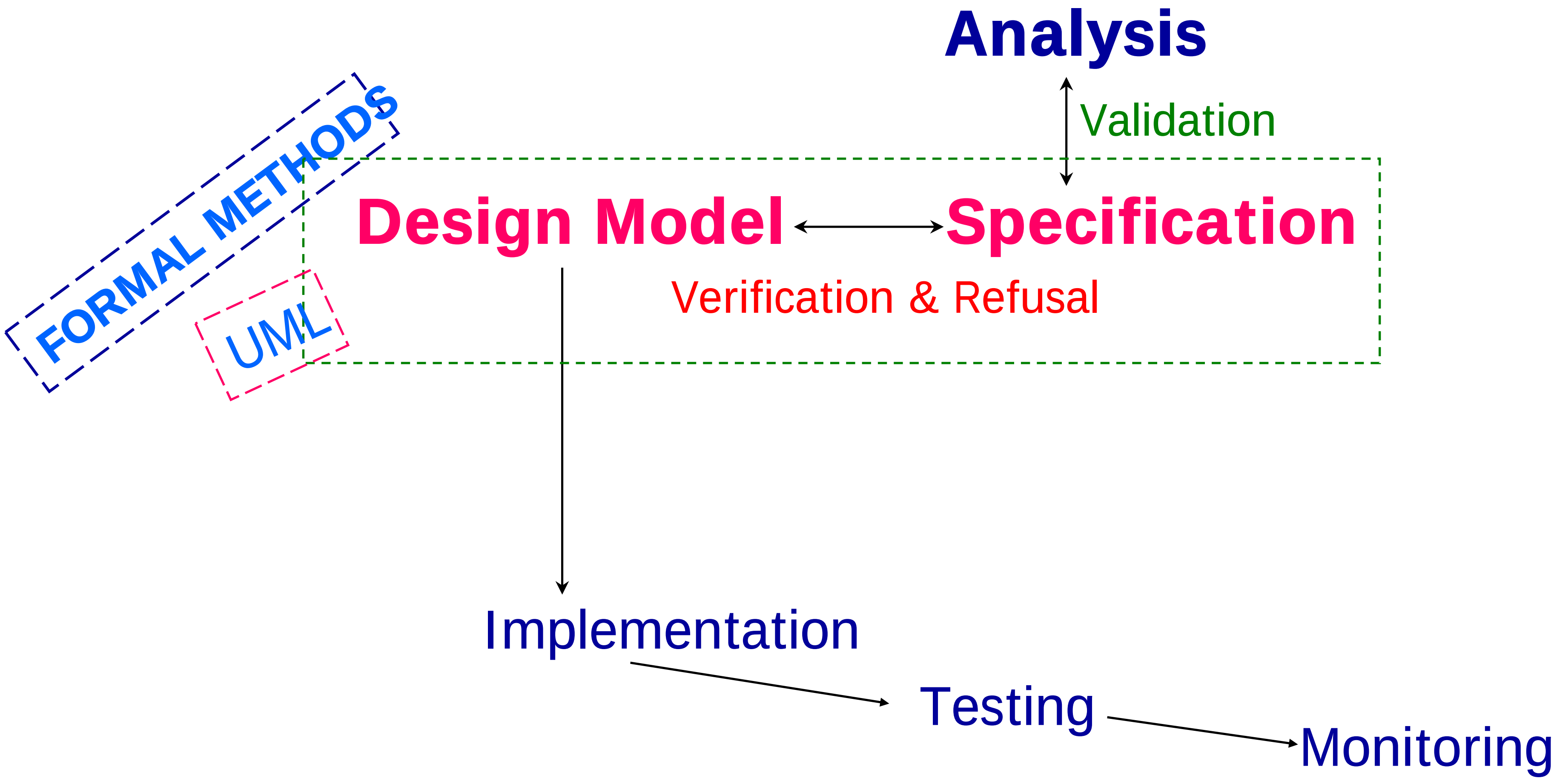


- Costly in time-to-market and money
- Errors are detected late or never
- Application of FM's as early as possible

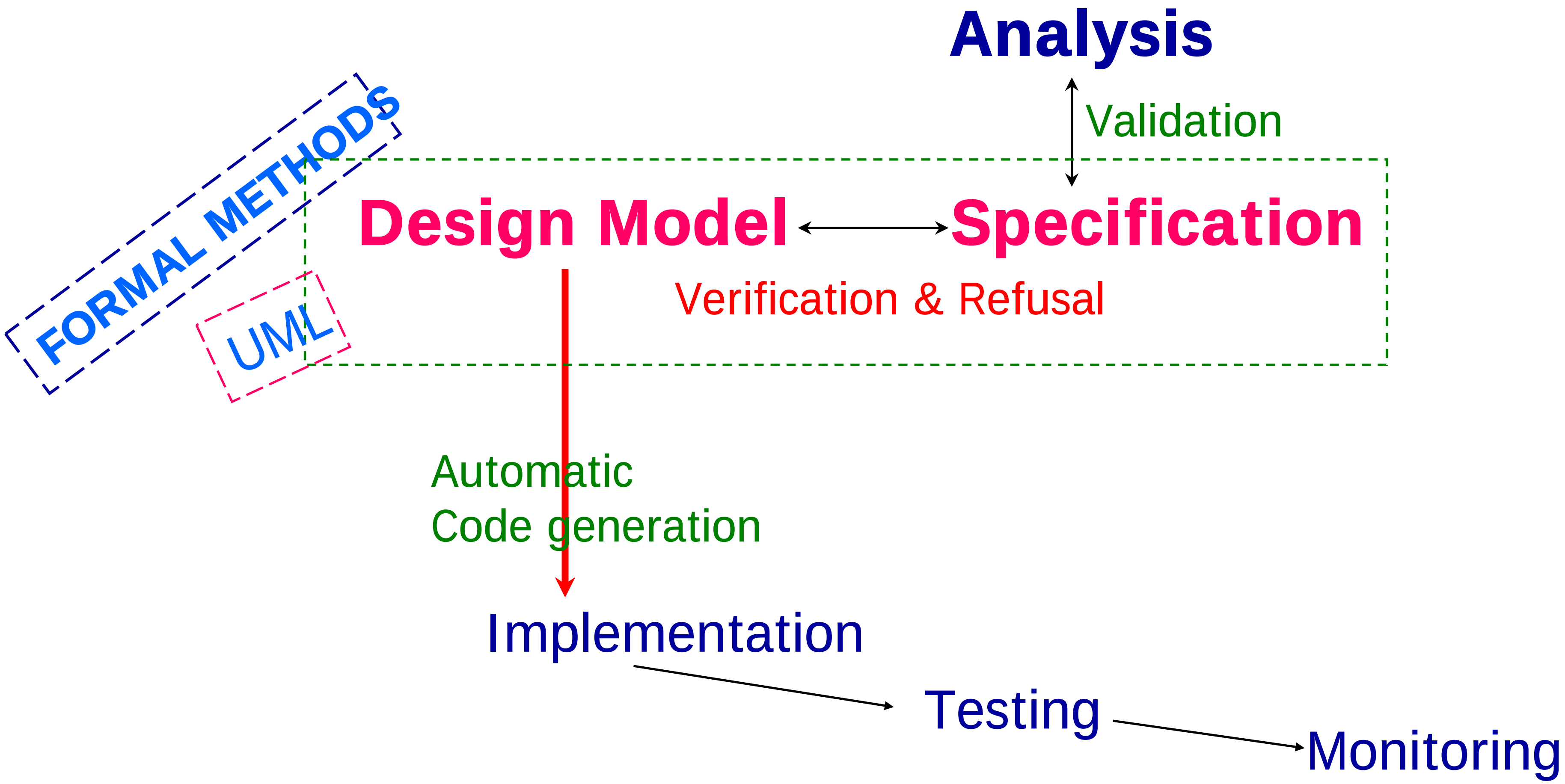
Models

- A model is a simplified representation of the real world.
- Engineers use models to gain confidence in the adequacy and validity of a proposed design.
- Focus on one or more aspects of interest:
  - Safety
  - Liveness
  - Peak time
  - Performance

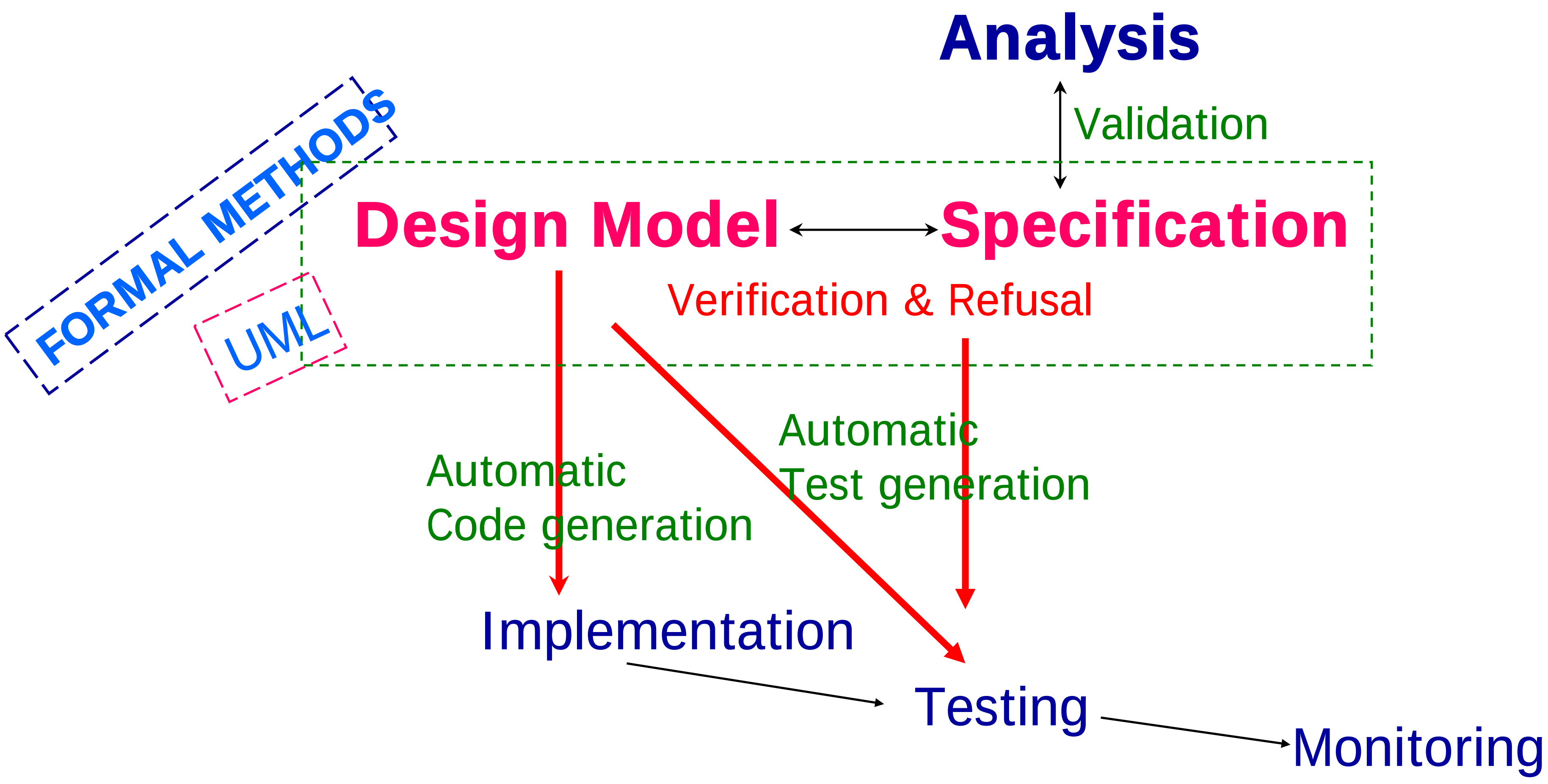
Model-based Validation



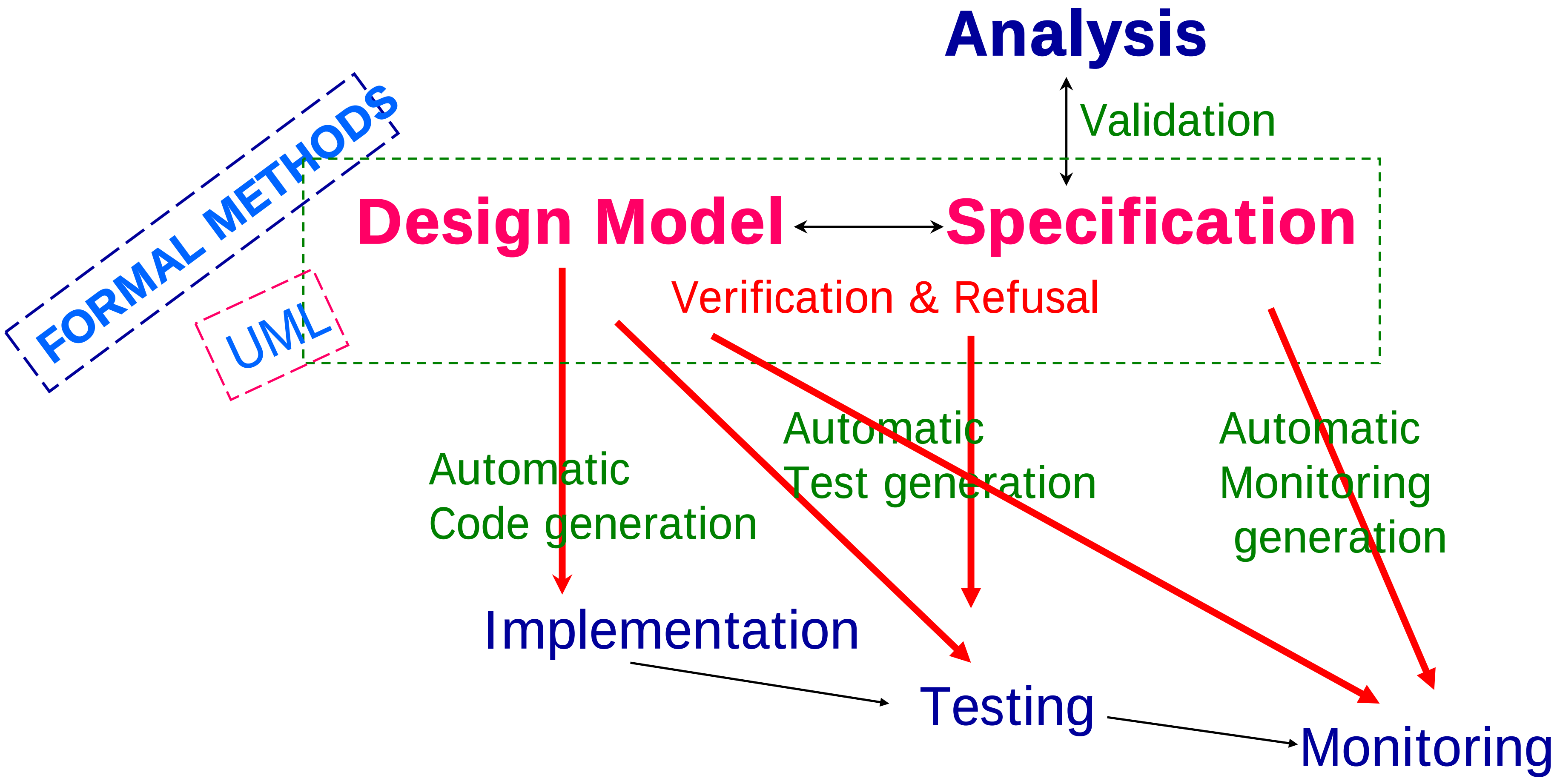
Model-based Validation



Model-based Validation



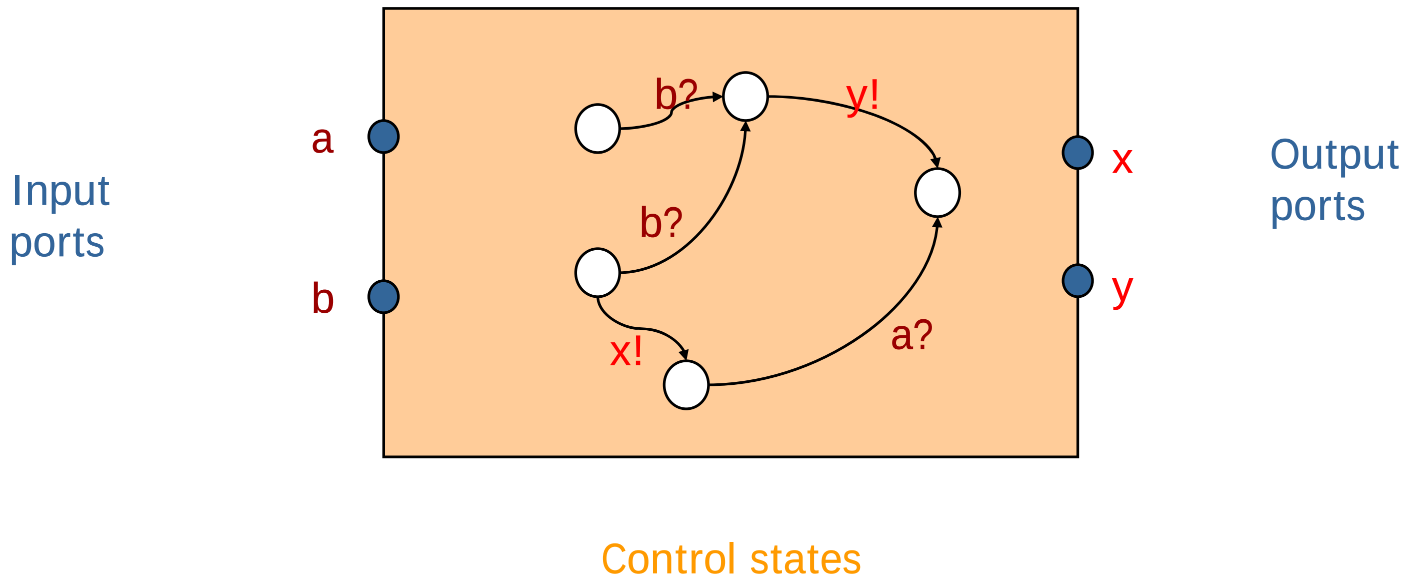
Model-based Validation



How?

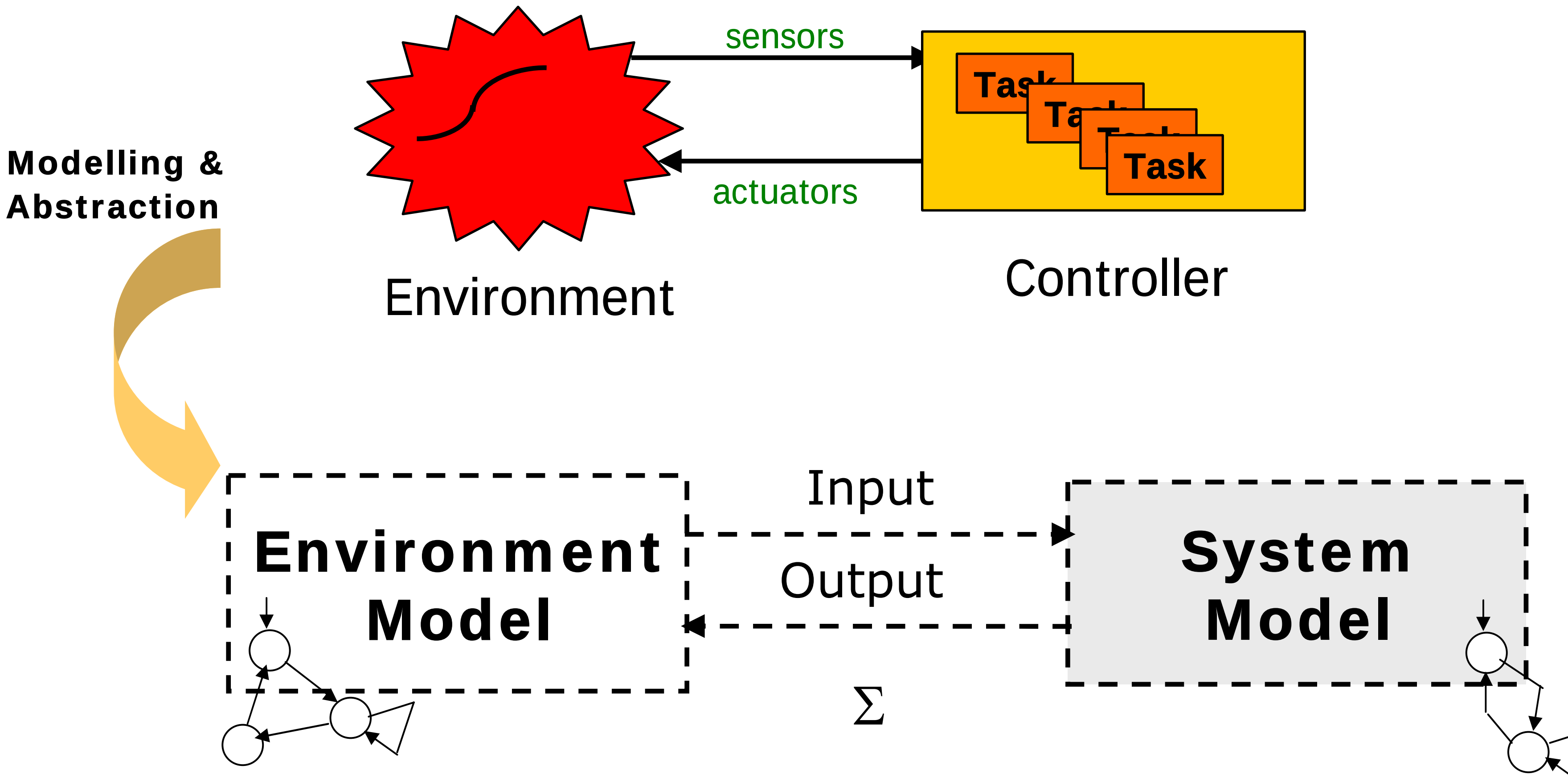
Unified Model = State Machine! +

Tools for analysis of state machines



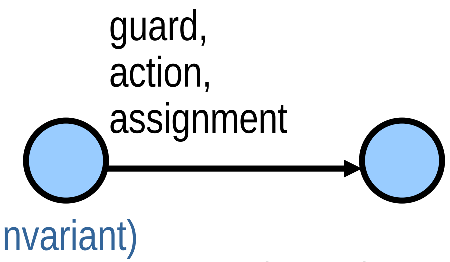
Real-Time Systems

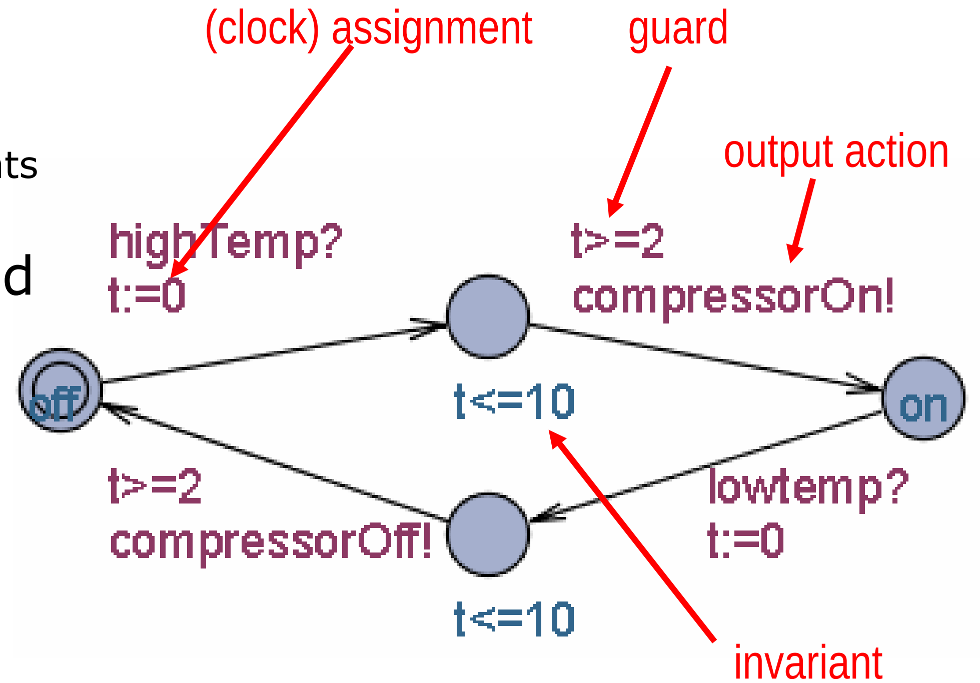
Real Time System
A system where correctness not only depends on the logical order of events but also on their timing



Timed Automata

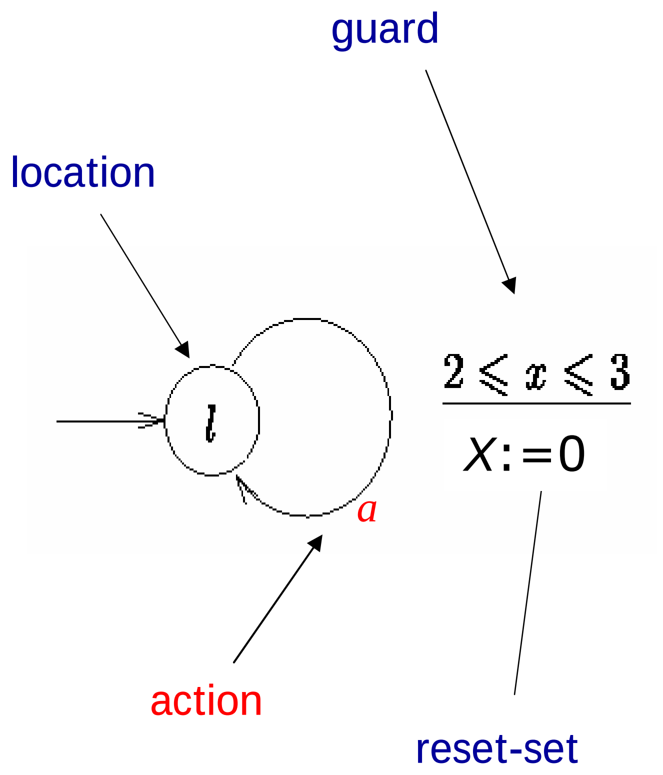
- I/O Timed Automata =FSM +
  - input? and output! actions
  - (discrete) data variables
  - (dense) **clocks**
  - guards and location invariants
  - assignments
- Clocks are special timed variables
- General form:



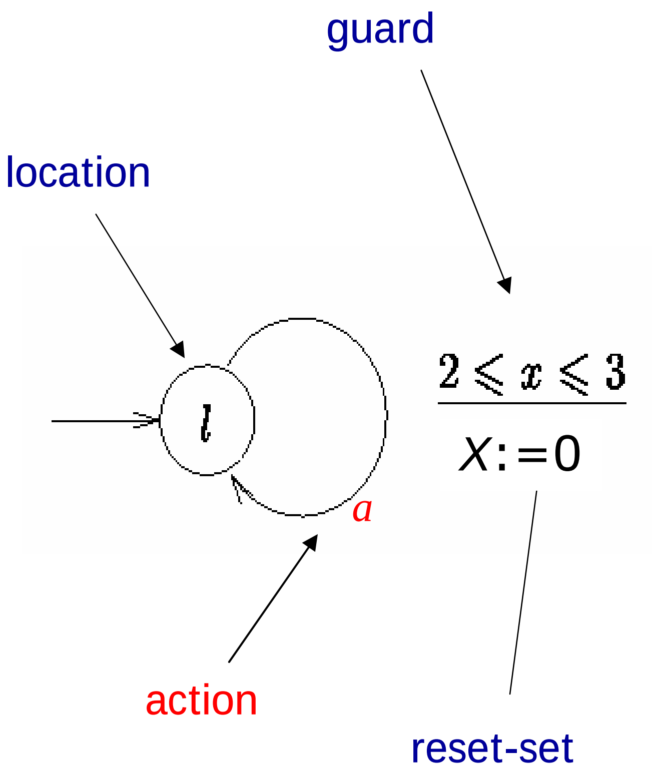


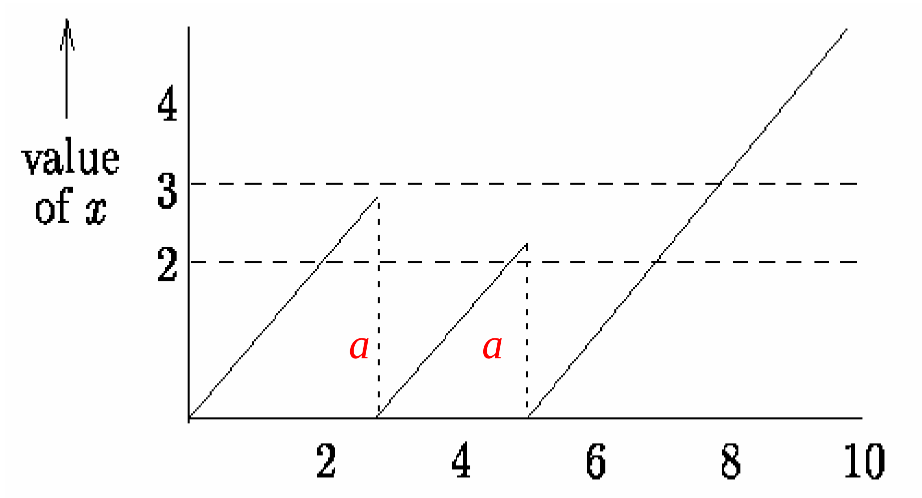
  - transition may fire if the guard is true, the action is ready, and then perform the assignment
  - location must be left before its invariant is violated
- Parallel Composition of TAs

Timed Automata: Example

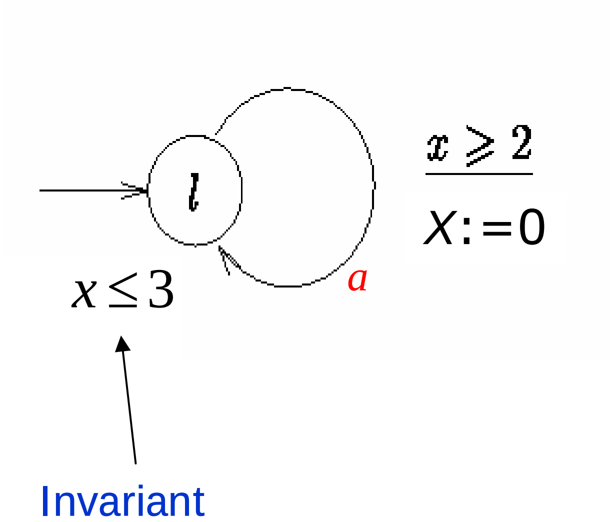


Timed Automata: Example

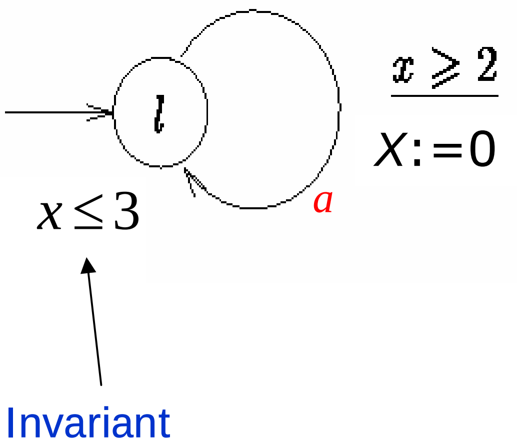


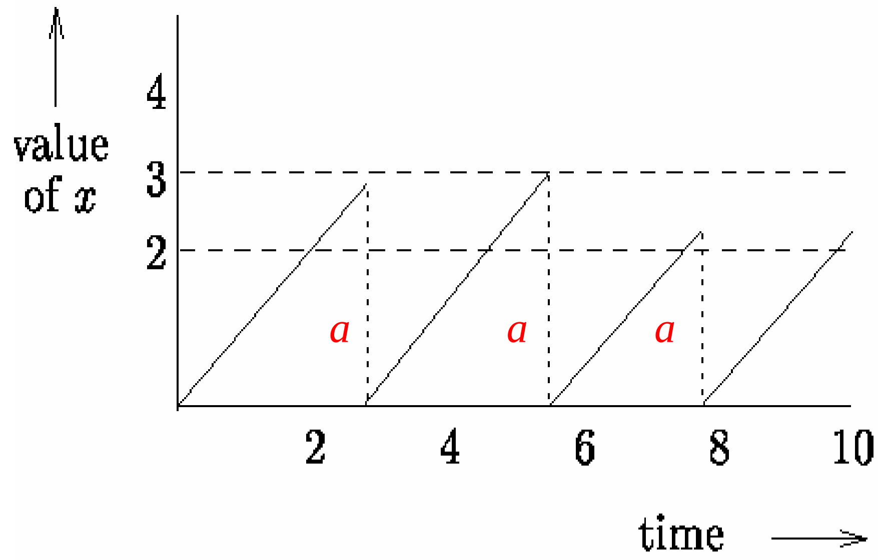


Timed Automata: Example

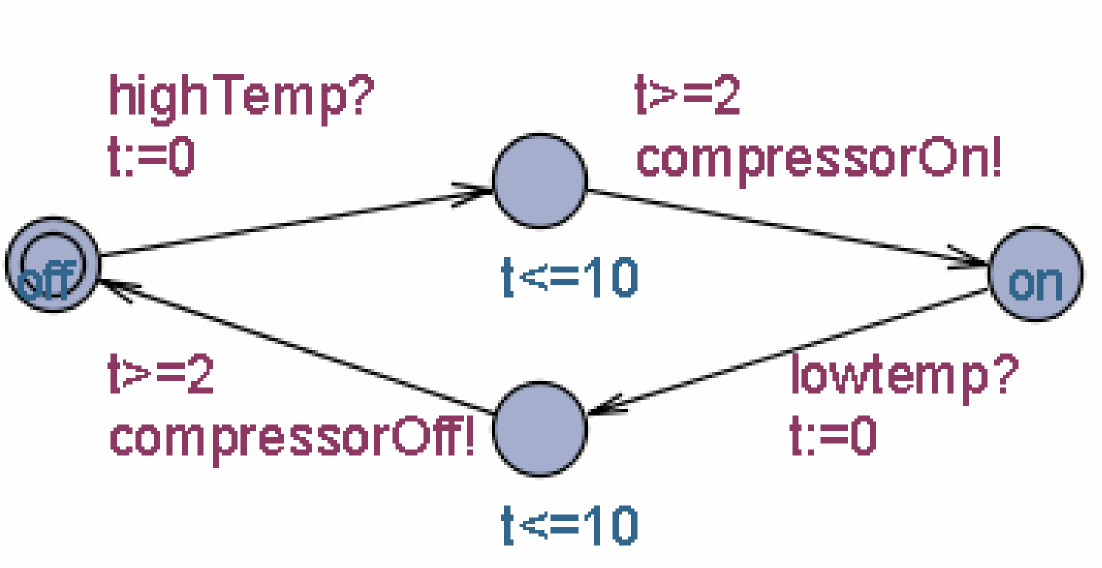


Timed Automata: Example





Sample Test Runs



highTemp!·3·compressorOn? ⇒ PASS

highTemp!·3·compressorOff? ⇒ FAIL

highTemp!·13·compressorOn? ⇒ FAIL

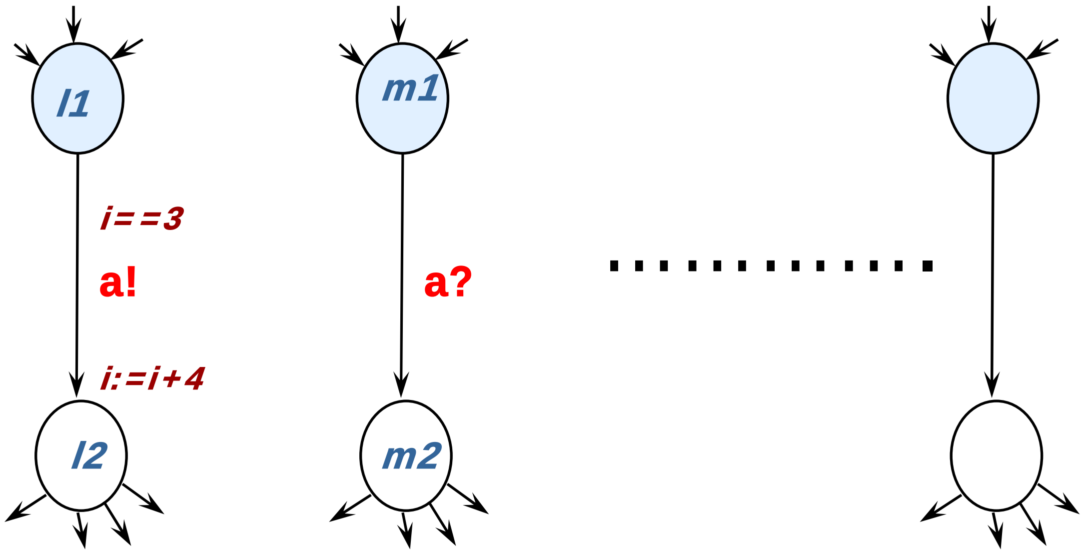
highTemp!·3·compressorOn?·123·lowTemp!·3·compressorOff? ⇒ PASS

highTemp!·3·compressorOn?·17·lowTemp!·3·compressorOff?·3.14·
highTemp!·5·compressorOn?·177·lowTemp!·3·compressorOff? ⇒ PASS

INFINITELY MANY SEQUENCES!!!!!!

The UPPAAL Model

= *Networks of Timed Automata + Integer Var + Array Var + ...*



Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

*(**l1**, **m1**,.........., x=2, y=3.5, i=3,.....)* $\xrightarrow{tau}$ *(**l2,m2**,........,x=0, y=3.5, i=7,.....)*

$\xrightarrow{0.2}$ ~~(**l1,m1**,.........,x=2.2, y=3.7, I=3,.....)~~

If **a** URGENT CHANNEL

Timed Automata in UPPAAL

- Timed (Safety) Automata
  + urgent action channels
  + broadcast action channels
  + urgent and committed locations
  + data-variables (with bounded domains)
  + arrays of data-variables + constants
  + guards and assignments over data-variables and
    arrays…
  + templates with local clocks, data-variables, and
    constants.

Timed Automata in UPPAAL

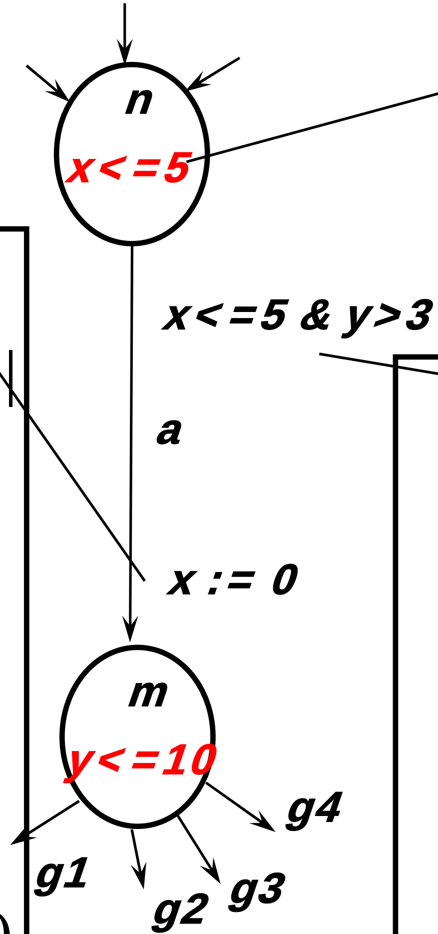
clock assignments

$$x := n$$

clock assignments

$$i := Expr$$

$$Expr ::= i \mid i[Expr] \mid n \mid -Expr \mid Expr + Expr \mid Expr - Expr \mid Expr * Expr \mid Expr / Expr \mid (g_d \,?\, Expr : Expr)$$



location invariants

$$inv ::= x < n \mid x \leq n \mid inv, inv$$

clock natural number and

$$g ::= g_c \mid g_d \mid g, g$$

$$g_c ::= x \prec n \mid x \prec y + n$$ **clock guards**

$$g_d ::= Expr \; op \; Expr$$ **data guards**

$$\prec \in \{<, <=, =, >=, >\}$$

$$op \in \{<, <=, =, >=, >, !=\}$$

Urgent Channels

```
urgent chan hurry;
```

Informal Semantics:

- There will be no delay if transition with urgent action can be taken.

Restrictions:

- No clock guard allowed on transitions with urgent actions.
- Invariants and data-variable guards are allowed.

Urgent Locations

Click "Urgent" in State Editor.

Informal Semantics:

- No delay in urgent location.

Note: the use of urgent locations **reduces** the number of clocks in a model, and thus the complexity of the analysis.

Committed Locations

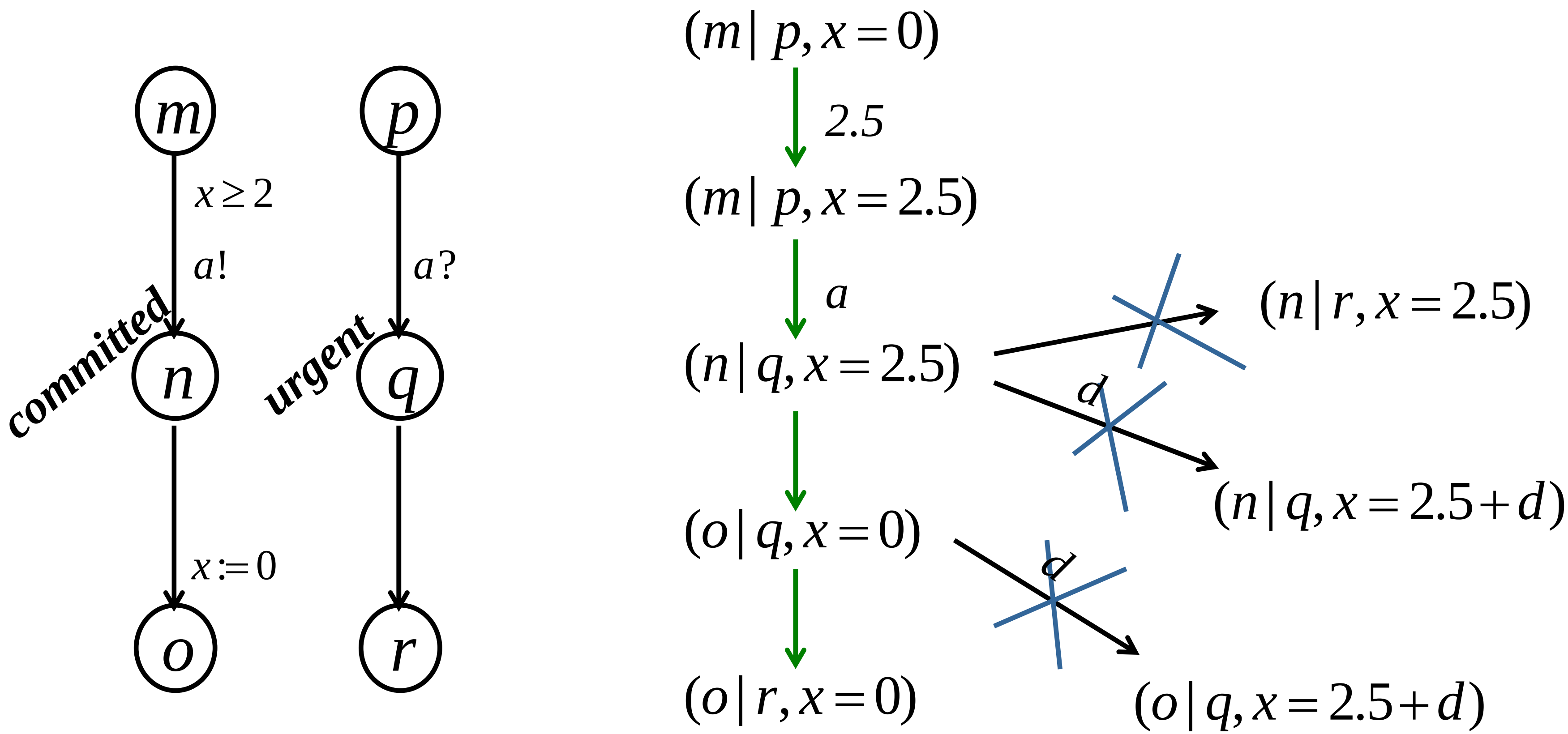
Click "Committed" in State Editor.

Informal Semantics:

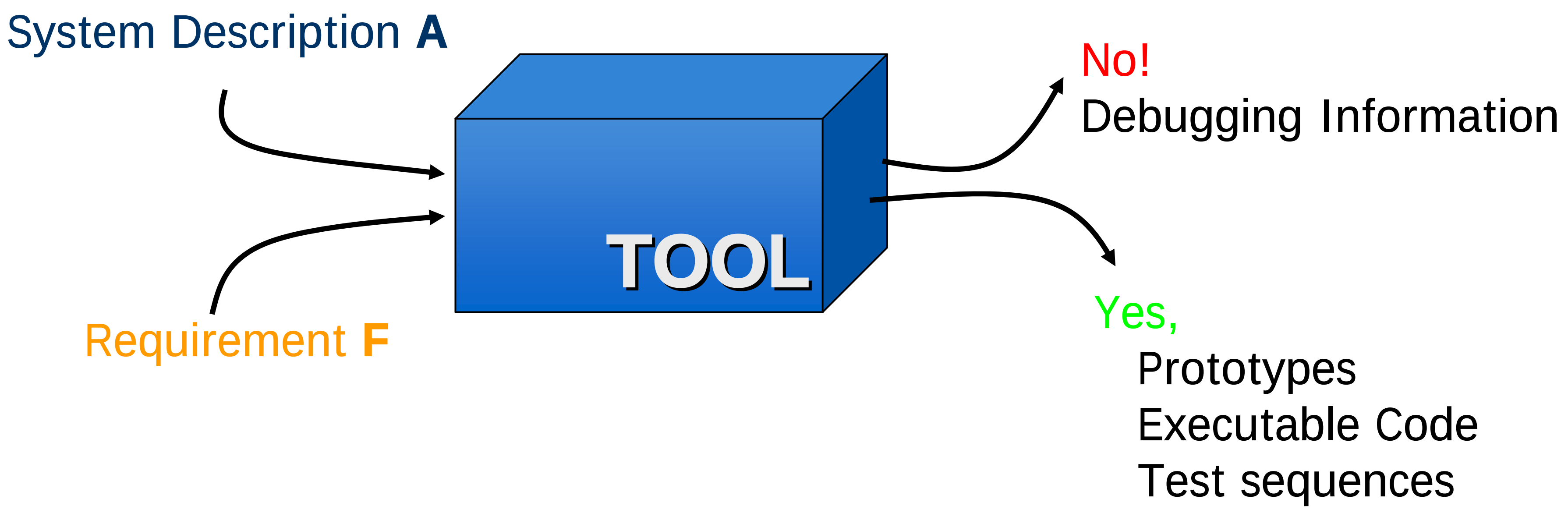
- No delay in committed location.
- Next transition must involve automata in committed location.

Note: the use of committed locations **reduces** the number of clocks in a model, and allows for more space and time efficient analysis.

Urgent and Committed Locations



Tool Support (model checking)



Tools: UPPAAL, visualSTATE, ESTEREL,
SPIN, Statemate, FormalCheck,
VeriSoft, Java Pathfinder,Telelogic...

UPPAAL Property Specification Language

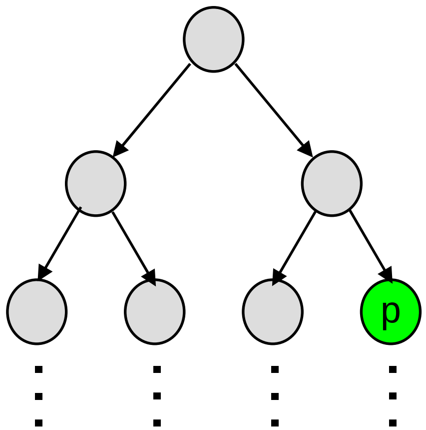
- **A[] p**
- **A<> p**
- **E<> p**
- **E[] p**
- **P --> q**

process location
data guards
clock guards

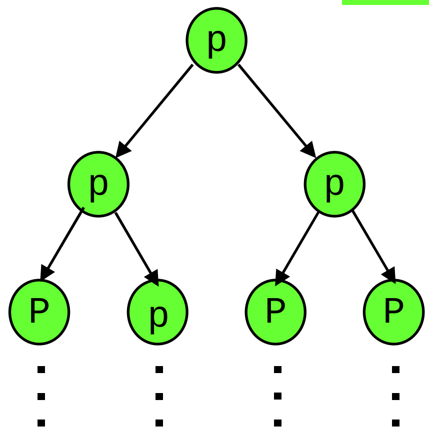
p::= a.l | g_d | g_c | p and p |
p or p | not p | p imply p |
(p) | deadlock (only for A[],E<>)

Uppaal "Computation Tree Logic"

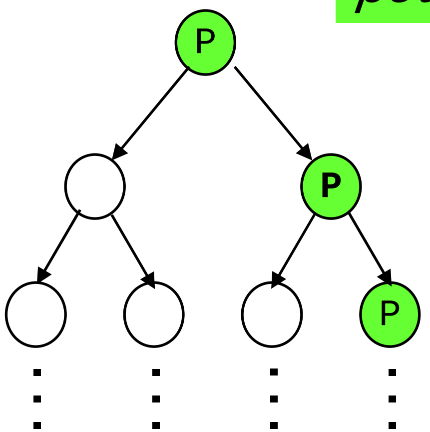
E<> p *Possible*



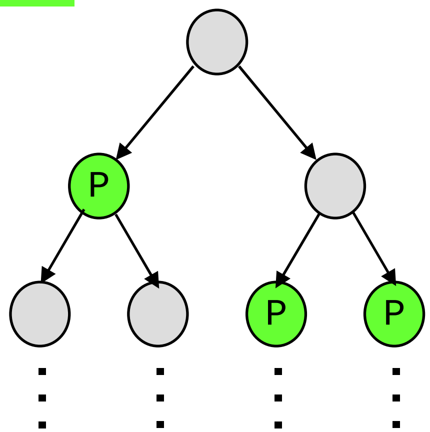
A[] p *always*



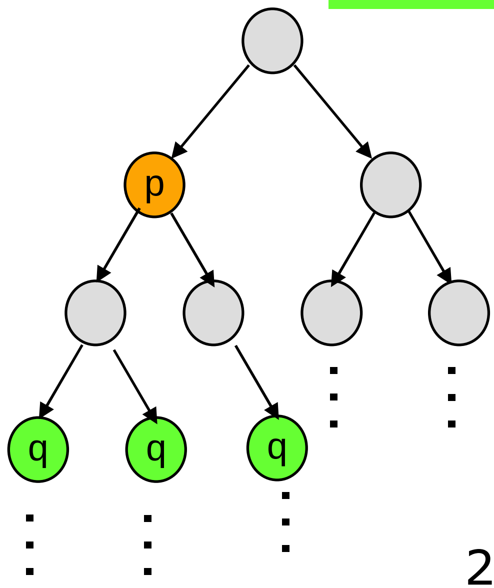
E[] p *potentially always*



A<> p *inevitable*

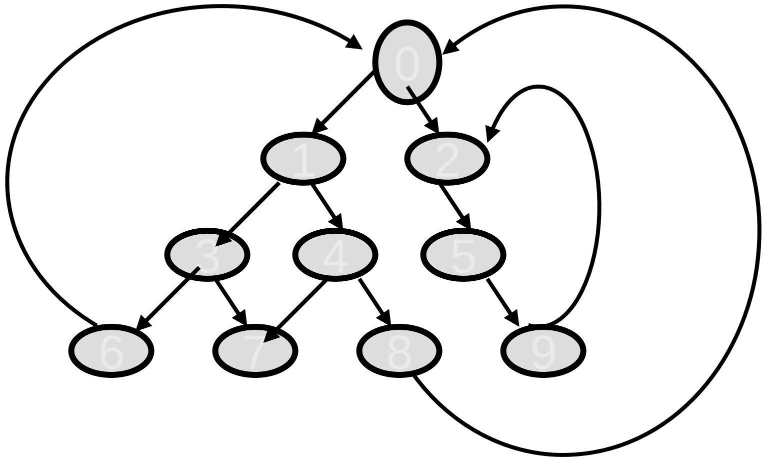


p --> q *leads-to*



Reachability Analysis

```
Passed:=Ø          //already seen states
Waiting:={S_0}     //states not examined yet
While(waiting!=Ø) {
  Waiting:=Waiting\{s_i}
  if s_i ∉ Passed
     whenever (s_j → s_j) then
        waiting:=waiting ∪ s_j
  }
```



Depth First: maintain waiting as a stack

Order: 0 1 3 6 7 4 8 2 5 9

Breadth First: maintain waiting as a queue (shortest counter example)

Order: 0 1 2 3 4 5 6 7 8 9

Home-Banking?

```
int accountA, accountB; //Shared global variables
//Two concurrent bank costumers
```

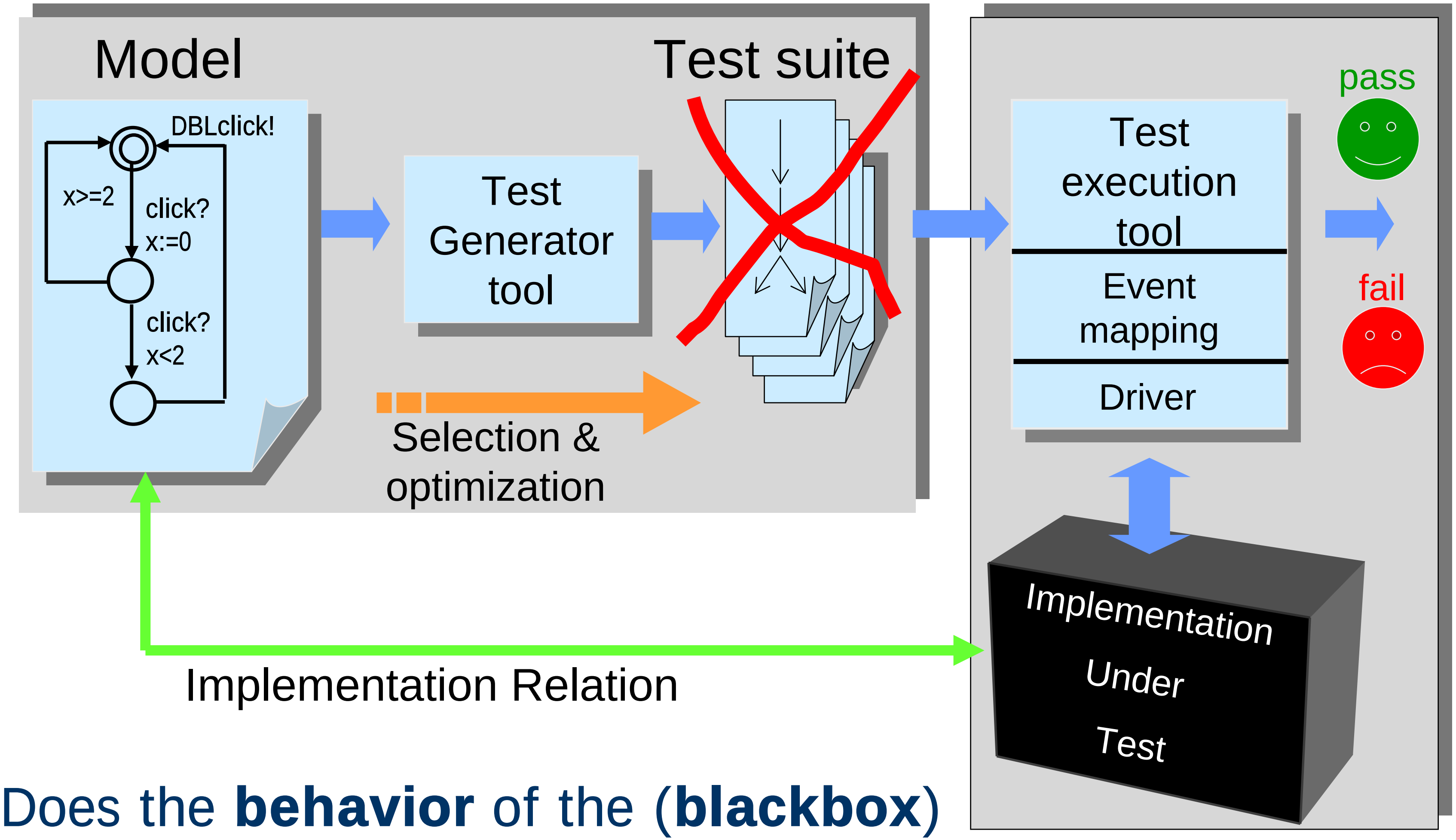
```
Thread costumer1 () {
  int a,b; //local tmp copy

  a=accountA;
  b=accountB;
  a=a-10;b=b+10;
  accountA=a;
  accountB=b;
}
```

```
Thread costumer2 () {
  int a,b;

  a=accountA;
  b=accountB;
  a=a-20; b=b+20;
  accountA=a;
  accountB=b;
}
```

- Are the accounts in balance after the transactions?

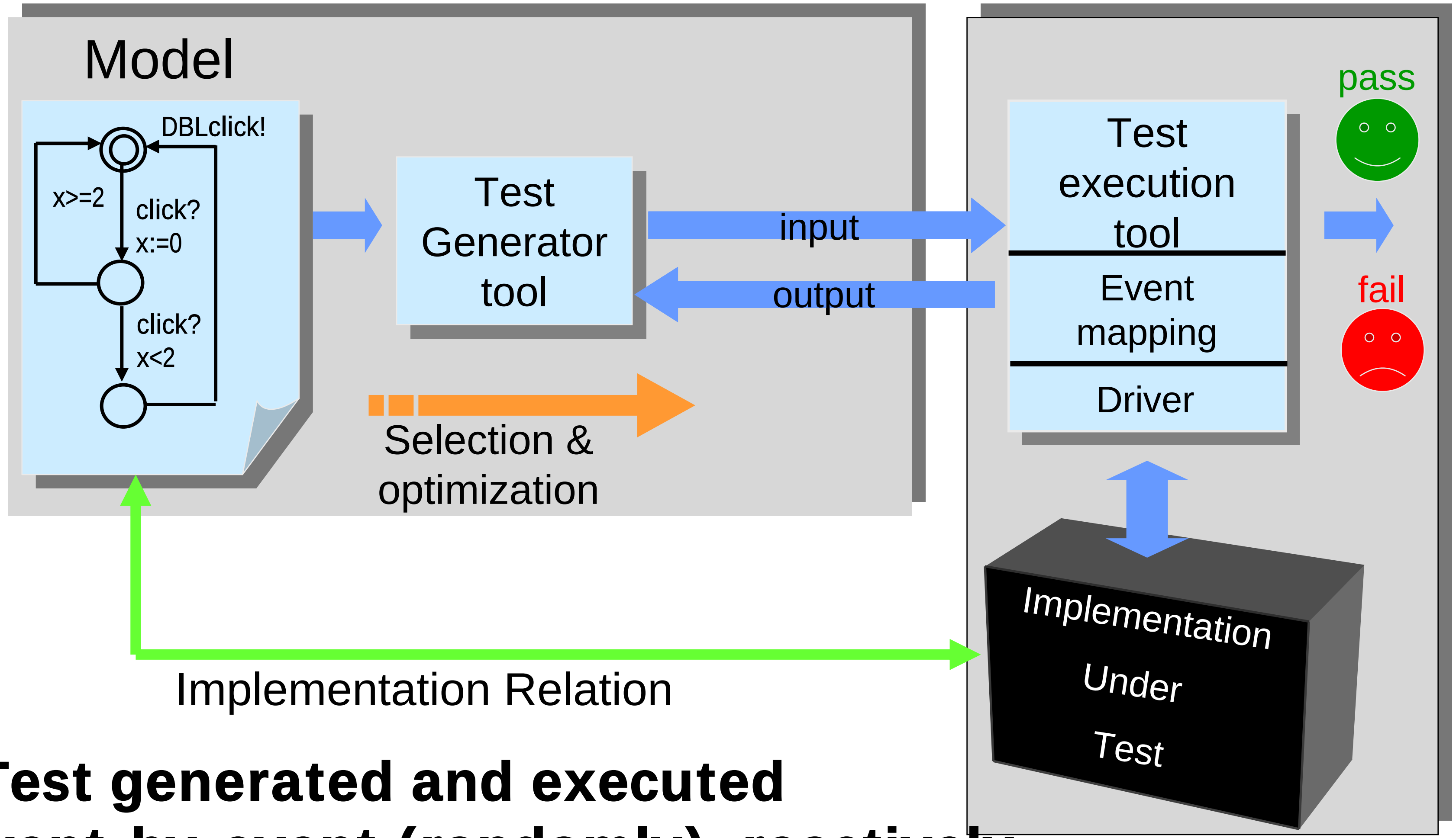
Uppaal Demo

Automated Model-Based Testing



Does the **behavior** of the (**blackbox**) implementation ***comply*** to that of the specification?

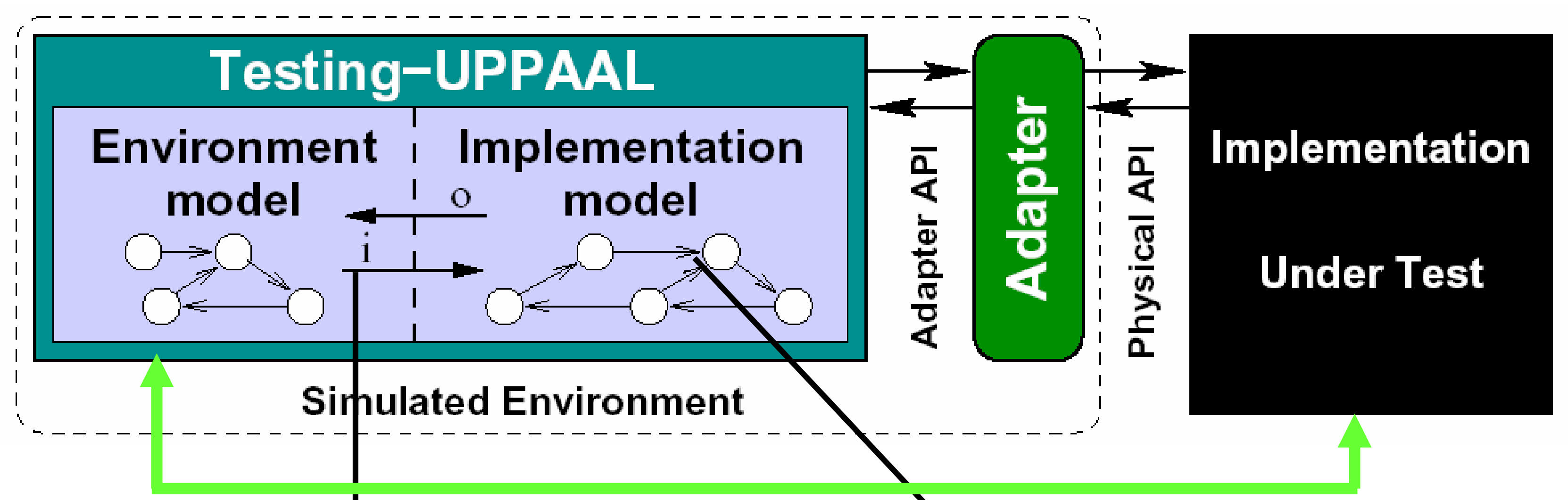
Online Testing



- **Test generated and executed event-by-event (randomly), reactively**
- **Long Running, deep testing, imaginative**

Our Framework

- ***UppAal Timed Automata*** *Network: Env || IUT*



"Formal Relativized i/o conformance" Relation

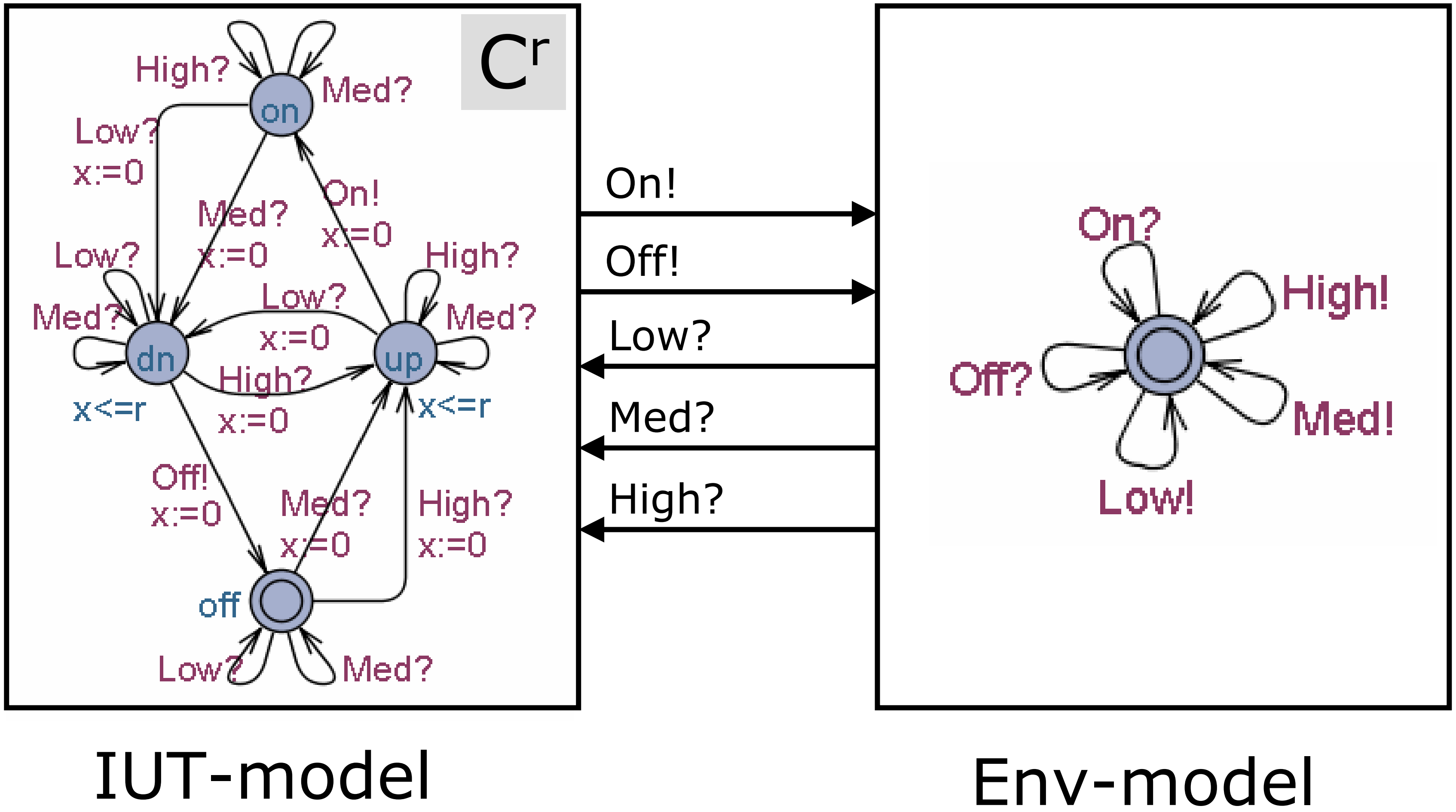
- Relevant input event sequences
- Load model

Correct system behavior

- Test Oracle
- Monitor

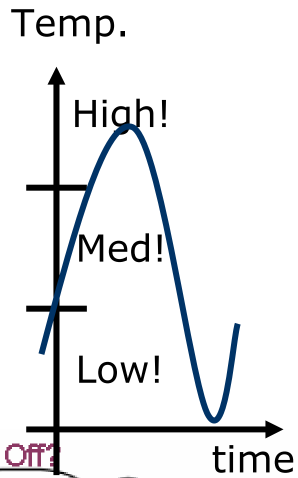
- *Complete and sound algorithm*
- Efficient symbolic reachability algorithms
- **UppAal-TRON:** Testing Real-Time Systems Online
- Release 1.3 **http://www.cs.aau.dk/~marius/tron/**

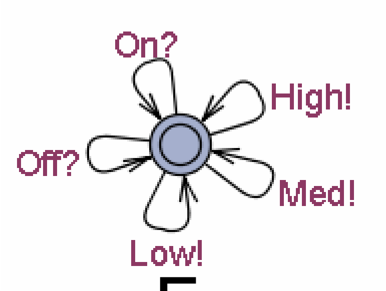
Sample Cooling Controller



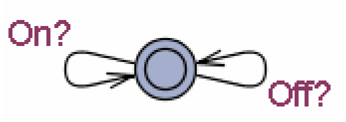
Env. Modeling

- Realism and Guiding
  - E_M Any action possible at any time
  - E_1 Only realistic temperature variations
  - E_2 Temperature never increases when cooling
  - E_L No inputs (completely passive)

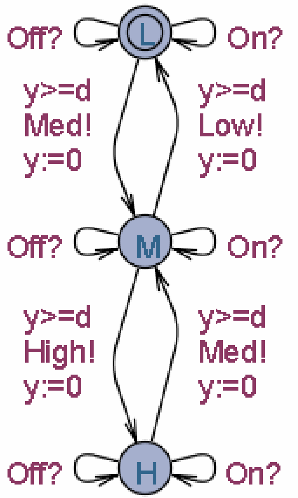




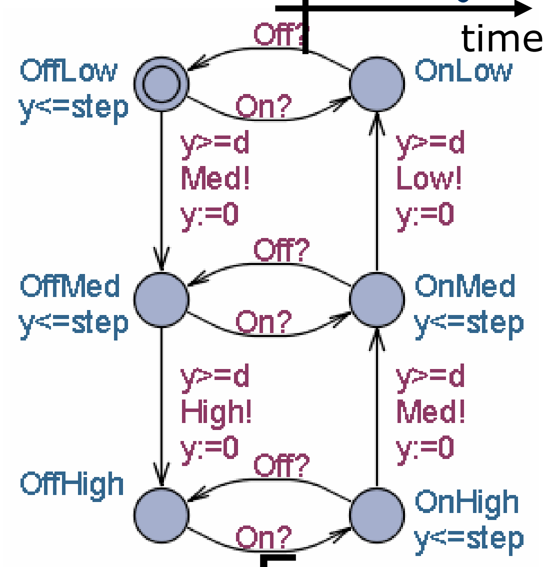
E_M



E_L



E_1

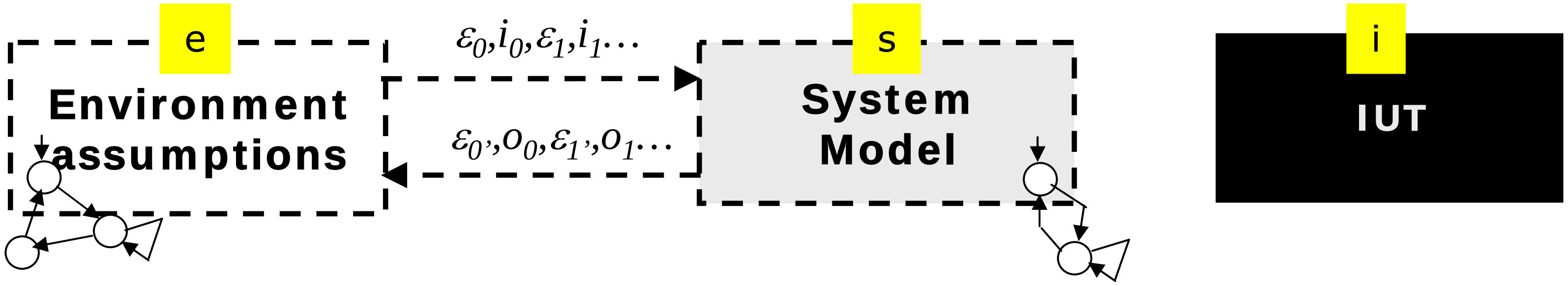


E_2

$$E_L \sqsubseteq E_2 \sqsubseteq E_1 \sqsubseteq E_M$$

Implementation relation

Relativized real-time io-conformance

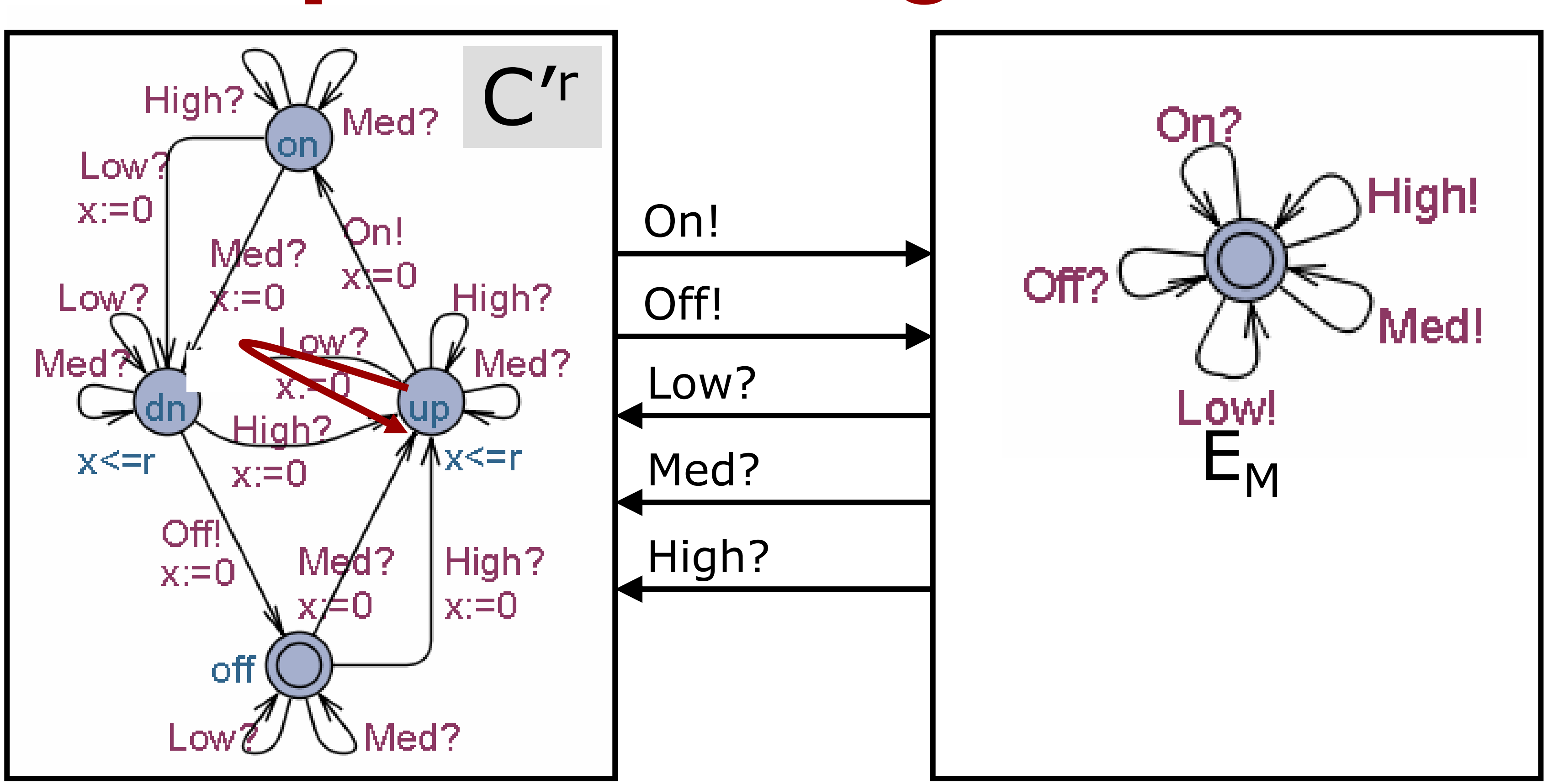


- **Let** *P* **be a set of states**
- **TTr**(*P*): the set of *timed traces* from states in *P*
- *P* **after** σ = the set of states reachable after timed trace σ
- **Out**(*P*) = possible outputs and delays in *P*

- i rt-ioco$_e$ s =def
  - $\forall \sigma \in TTr(e)$: Out((e,i) after σ) ⊆ Out((e,s) after σ)
- i rt-ioco$_e$ s iff TTr(i) ∩ TTr(e) ⊆ TTr(s) ∩ TTr(e)

- **Intuition, for all relevant environment behaviors**
  - **never produces illegal output, and**
  - **always produces required output in time**
- **~timed trace inclusion**

Sample Cooling Controller

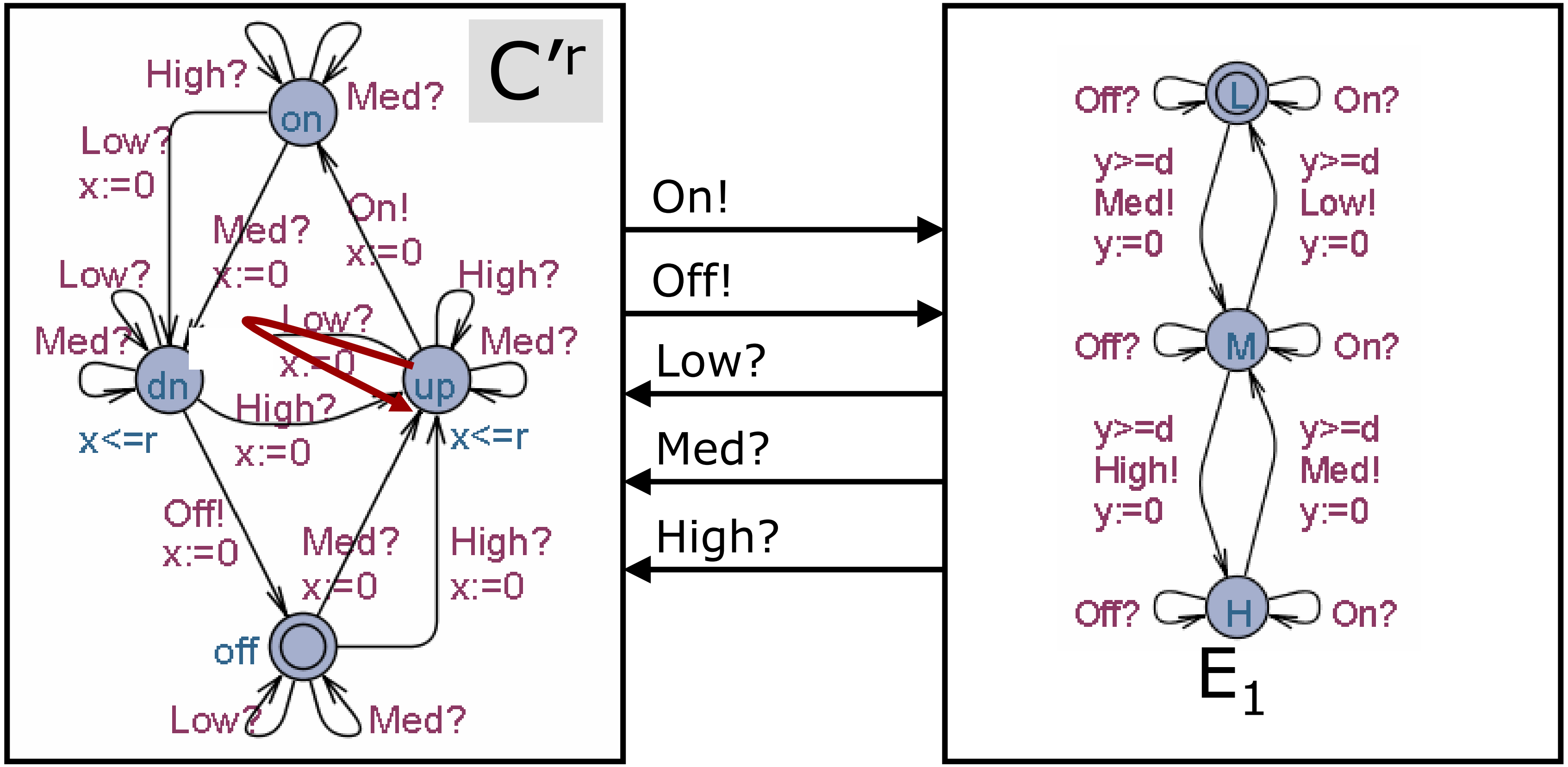


IUT

Env-model

C'^r rt-ioco $_{E_M}$ C^r

Sample Cooling Controller



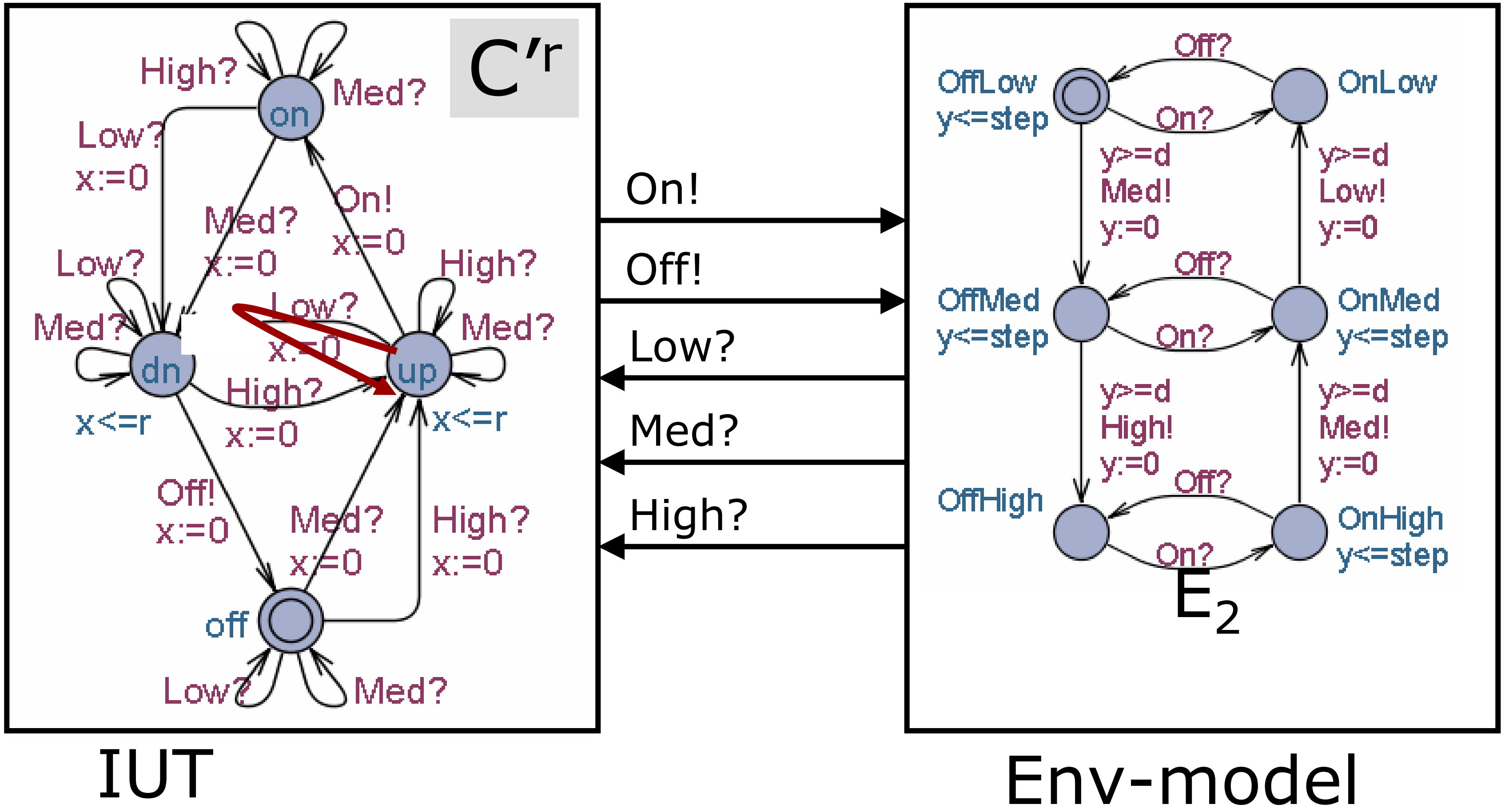
IUT

Env-model

C'^{r} ~~rt-ioco~~ $_{E_1}$ C^{r}, iff $3d<r$

d.Med?.d.High?.d.Med?.d.Low?.ε.On, $\varepsilon \le r$

Sample Cooling Controller



C'^r ~~rt~~-ioco $_{E_2}$ C^r

Randomized Online Algorithm

Algorithm *TestGenExec (TestSpec)* **returns** {**pass**, **fail**}

Z:={⟨l_0,0⟩},
While Z ≠∅ and #iterations≤T **do** choose randomly

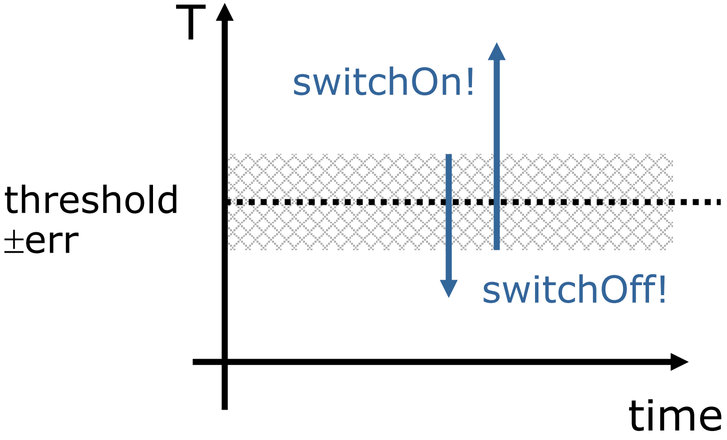
1. **if** *EnvOutput*(Z) ≠∅ // Offer an input
   choose randomly a ∈ *EnvOutput*(Z)
   send *i* to SUT
   Z:=Z ***after*** *a*
2. choose randomly δ ∈ *Delays*(Z) // Delay and wait for output
   Wait(δ)
   if *o* occurred after δ′ ≤ δ **then**
   Z:=Z ***after*** δ′
   if o ∉ *ImpOutput*(Z) **then return fail**
   Z:=Z ***after*** *o*
   else // no output within δ time
   Z:=Z ***after*** δ
3. reset IUT
   Z:={⟨l_0,0⟩}

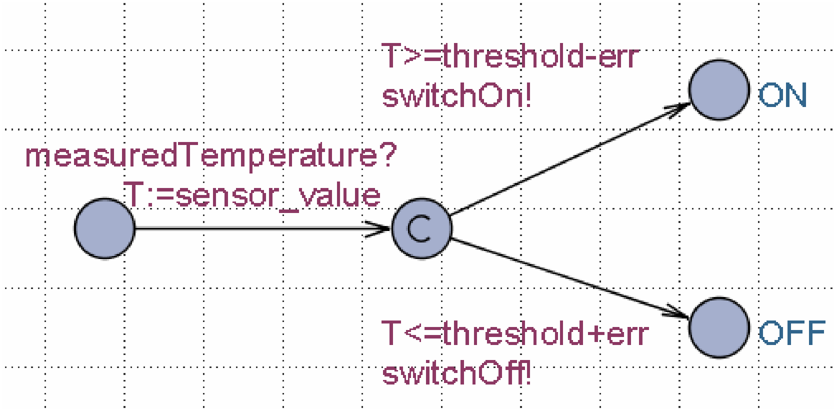
if Z=∅ **then return fail else return pass**

- **Sound**
- **Complete as T→∞**

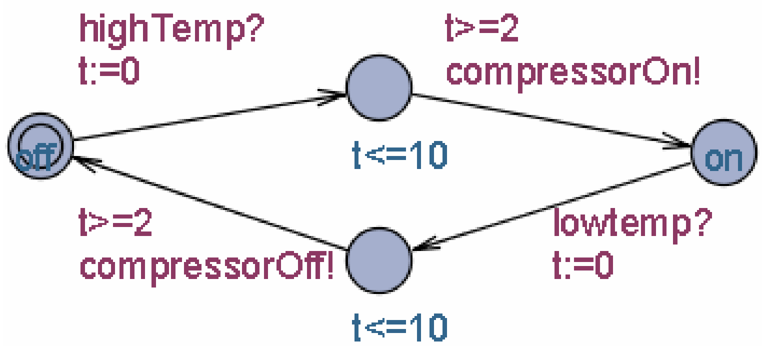
Non-Determinism

- **Modeling Action uncertainty**
- A controller switches a relay when a control variable crosses 'around' threshold value





- **Modeling Timing uncertainty**
- A controller switches a relay between 2 and 10 time units



State-set computation

- Compute all potential states the model can occupy after the timed trace $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$

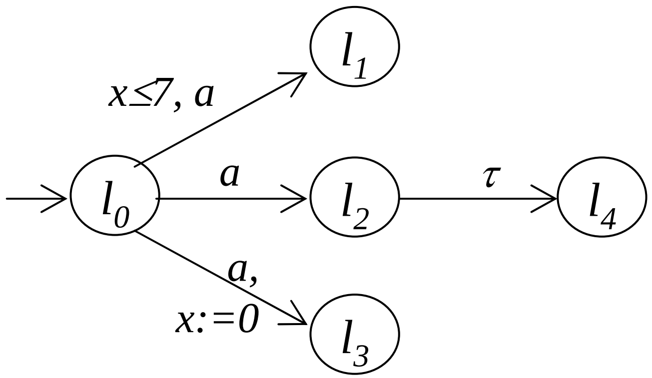
- *Let Z be a set of states*
  - ***Z after a***: *possible states after executing a (and t*)*
  - ***Z after*** ε :*possible states after t* and* ε_i *, totaling a delay of* ε

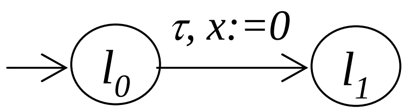
  - ***o is a legal output from SUT iff O in ImpOutput(Z)***
  - ***a is a relevant input in Env iff I in EnvOutput(Z)***

  - ε ***is a permitted delay iff Z after*** $\varepsilon \neq \emptyset$
  - ε ***is a relevant delay iff Delays (Z)***

State-set Computation

- Compute all potential states the model can occupy after the timed trace $\varepsilon_0, i_0, \varepsilon_1, o_1, \varepsilon_2, i_2, o_2, \ldots$
- *Let Z be a set of states*
  - **Z after a**: *possible states after executing a (and τ^*)*
  - **Z after ε** :*possible states after τ^* and ε_i , totaling a delay of ε*





$\{ \langle l_0, x=3 \rangle \}$ **after** *a* = $\{ \langle l_2, x=3 \rangle, \langle l_4, x=3 \rangle, \langle l_3, x=0 \rangle \}$

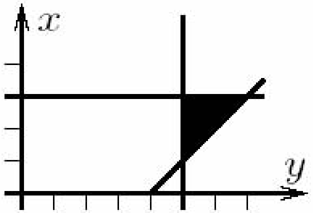
$\{ \langle l_0, x=0 \rangle \}$ **after** 4 = $\{ \langle l_0, x=4 \rangle, \langle l_1, 0 \le x \le 4 \rangle \}$

- Represent state sets as sets of symbolic states
- Use symbolic reachability
- (similar to model checkers like UppAal)

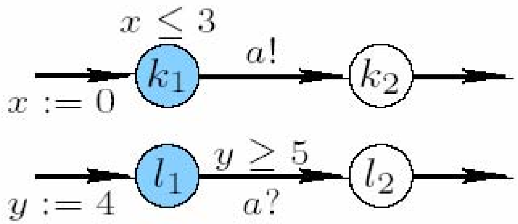
Symbolic Reachability

- *Zone* is a conjunction of clock constraints of the form: $\{x_i - x_j \prec c_{ij}\} \cup \{a_i \prec x_i\} \cup \{x_j \prec b_j\}$ where $\prec \in \{\leq, \leq\}$
- *Difference bound matrix* - compact representation.
- Symbolic state set $Z = \{\langle \bar{l}_1, z_1 \rangle, \ldots, \langle \bar{l}_n, z_n \rangle\}$
- *Action* transition: $\langle \bar{l}, z \rangle \xrightarrow{a} \langle \bar{l}', (z \wedge g)_r \wedge I(\bar{l}') \rangle$: $l \xrightarrow{g,a,r} l'$ is a-action transition and $z \wedge g \neq \emptyset$,$(z \wedge g)_r \wedge I(\bar{l}') \neq \emptyset$.
- *Delay* transition: $\langle \bar{l}, z \rangle \xrightarrow{\delta} \langle \bar{l}, z^{+\delta} \wedge I(\bar{l}) \rangle$ iff $z^{+\delta} \wedge I(\bar{l}) \neq \emptyset$.

$$z = [(y - x \leq 4) \wedge (y \geq 5) \wedge (x \leq 3)]$$



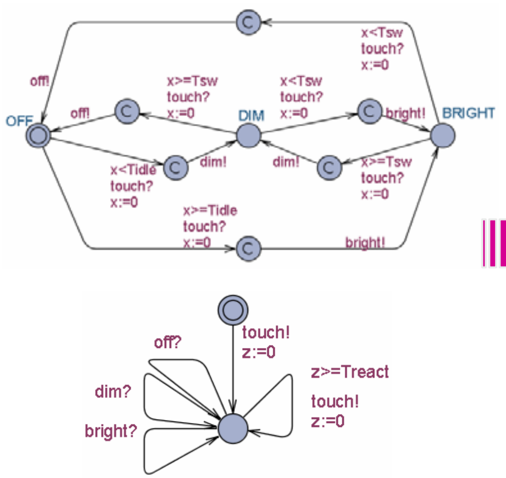
	0	y	x
0	–	−5	0
y	∞	–	4
x	3	∞	–



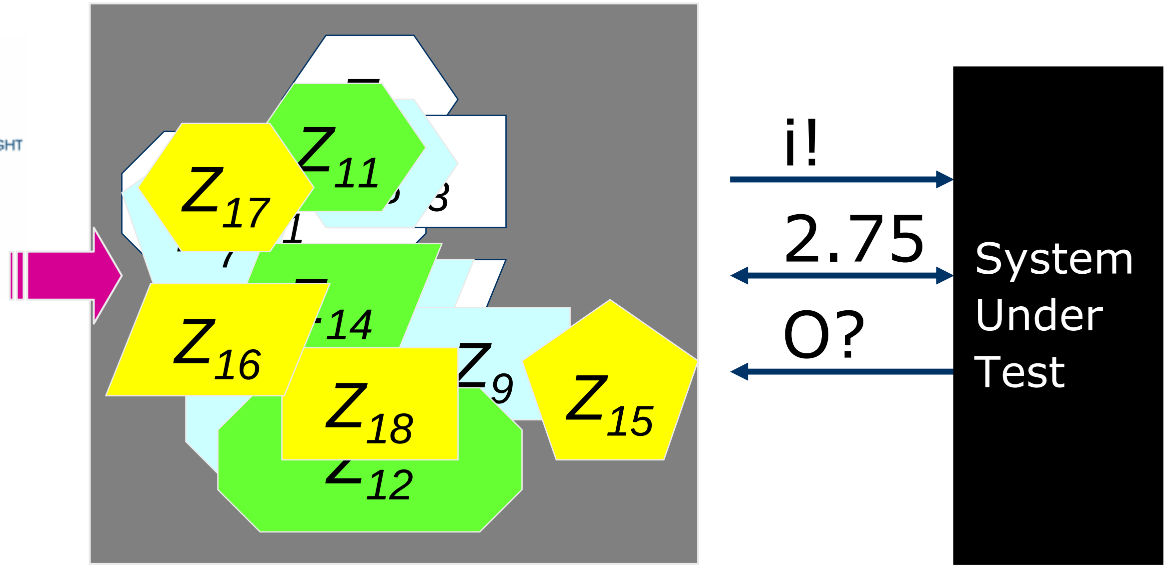
Real-time Online

- Compute *all* states reachable **after** timed trace
- Maintain a *set* of *symbolic* states in real time!

Specification TA-network



Online Tester:



[Tripakis'02, Krichen'04]

Danfoss EKC Case

Electronic Cooling Controller

Sensor Input

- air temperature sensor
- defrost temperature sensor
- (door open sensor)

Keypad Input

- 2 buttons (~40 user settable parameters)

Output Relays

- compressor relay
- defrost relay
- alarm relay
- (fan relay)

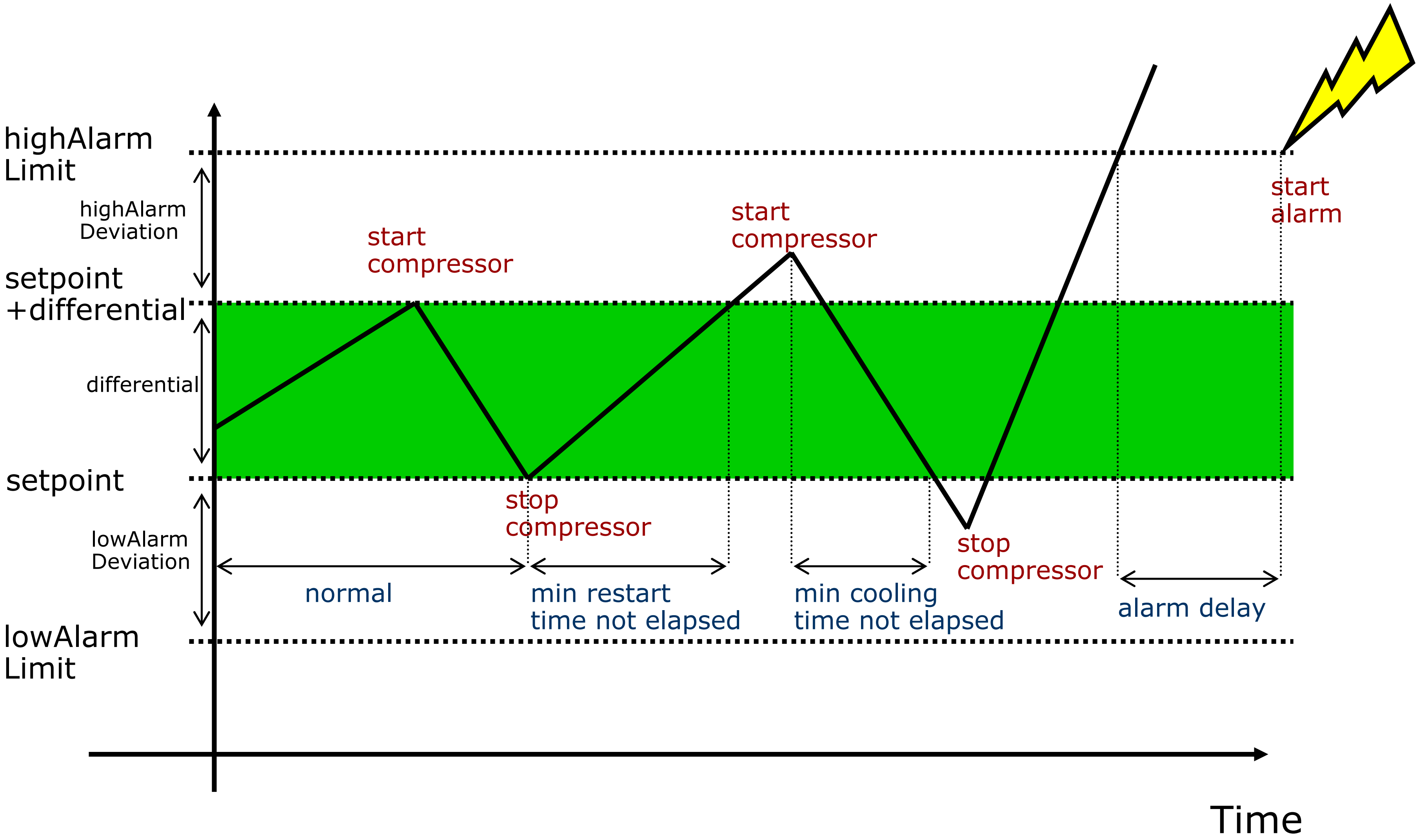
Display Output

- alarm / error indication
- mode indication
- current calculated temperature

- Optional real-time clock or LON network module

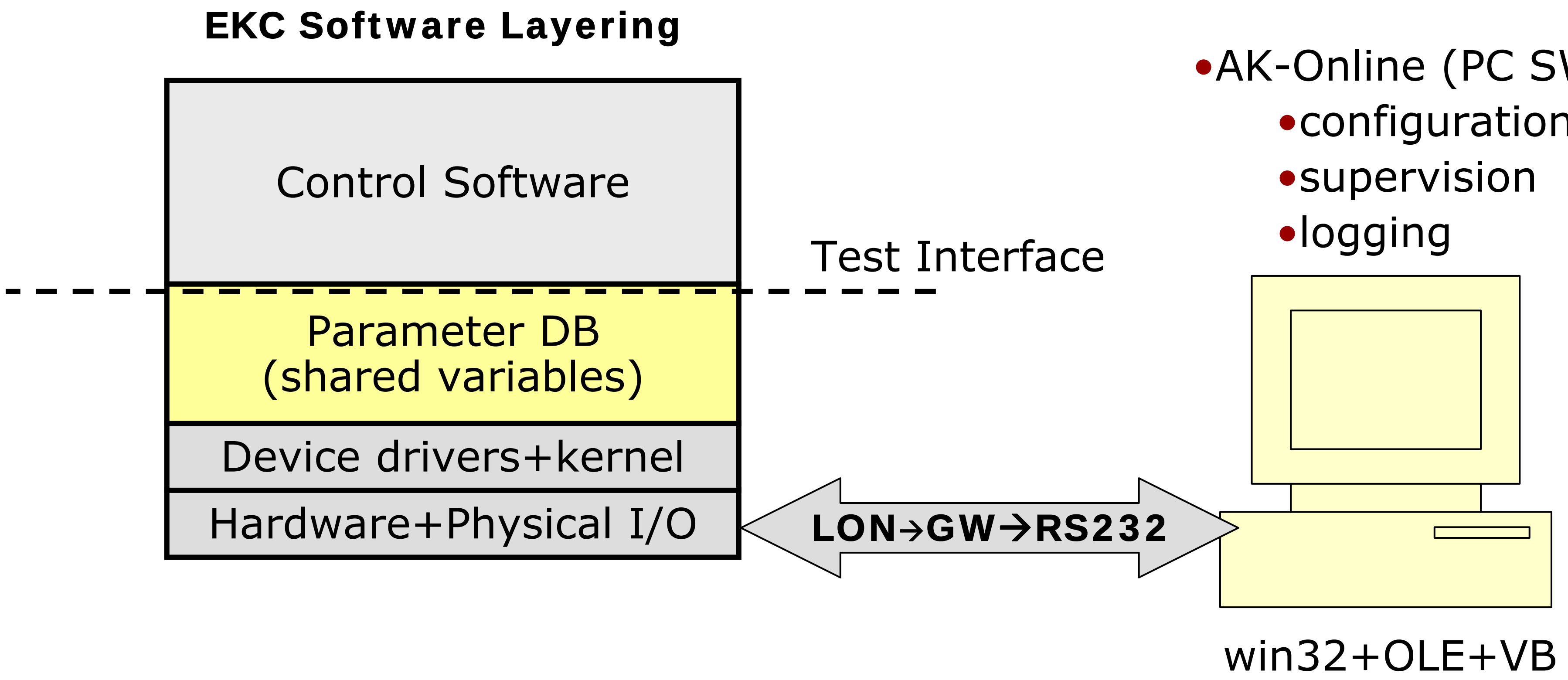
Industrial Cooling Plants

Basic Refrigeration Control



EKC Adaptation 1

- **Read and write parameter "database"**
- **47 parameters**



- AK-Online (PC SW)
  - configuration
  - supervision
  - logging

EKC Adaptation 2

TRON Engine

compressorOn ***setTemp(20)***

Adaptor

16.7	0	0	old copy
22.1	0	1	new copy

"continous" readout

2 readouts/s

22.3	0	1

"par#4=20.0"

Need better test interface!

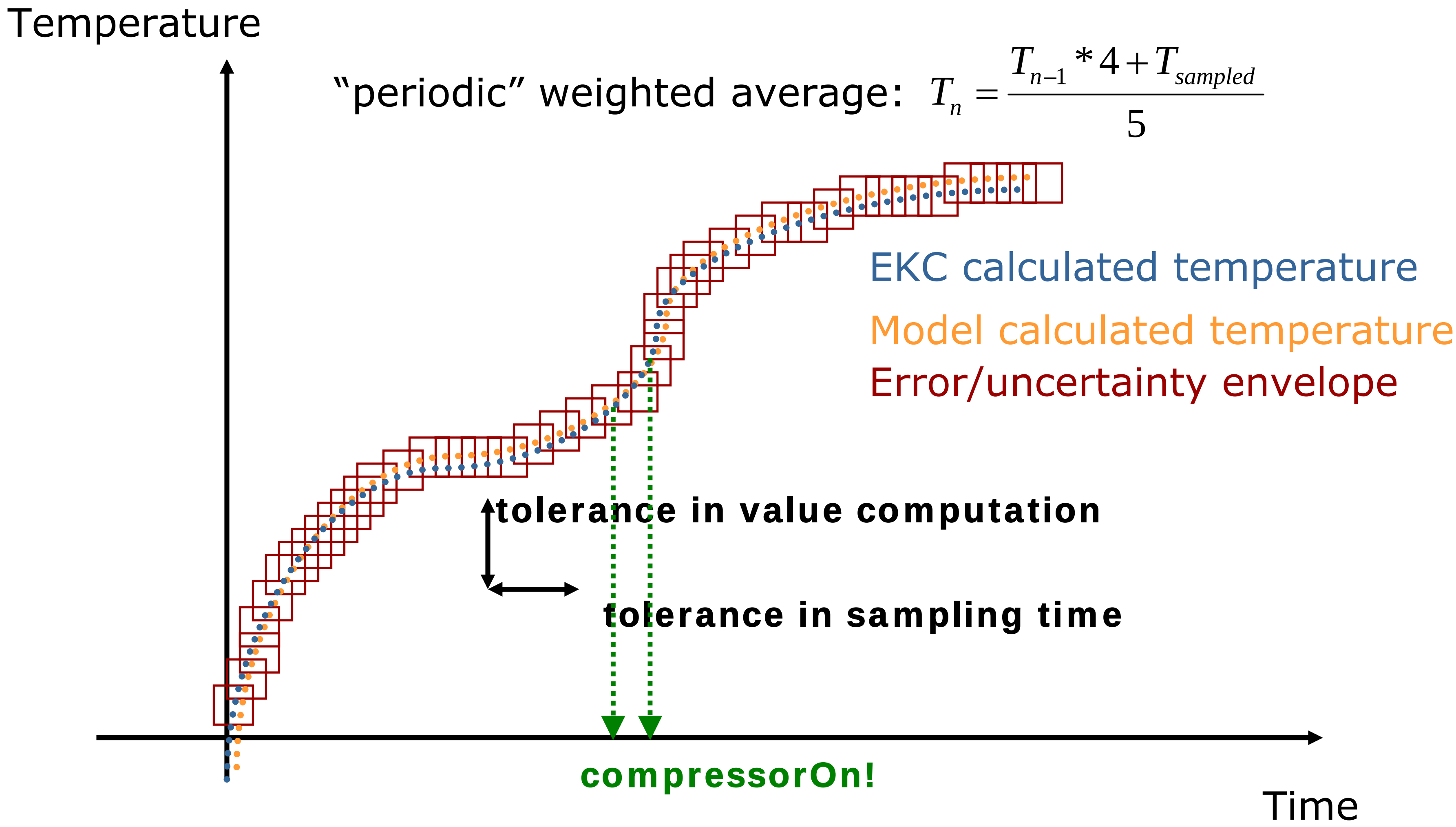
- Read-only parameters
- Delay and synchronization

win32+OLE+VB

Solaris/Linux (C++)

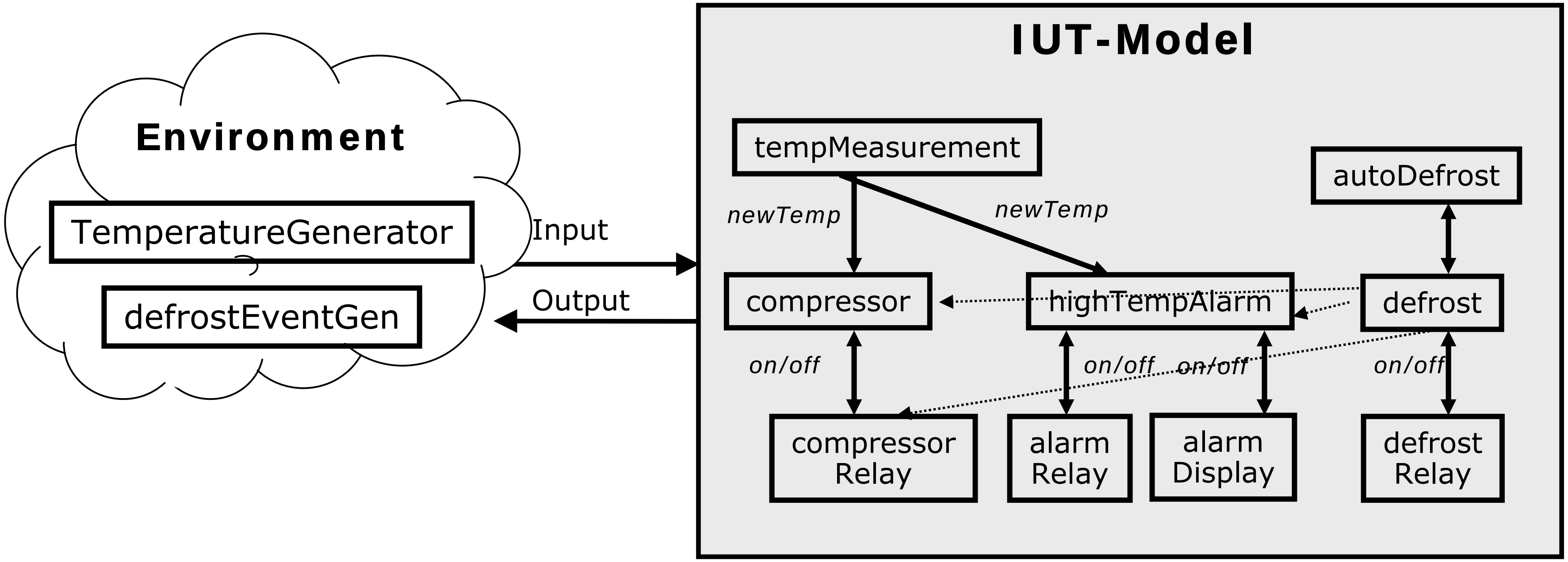
Temperature Tracking

"periodic" weighted average: $T_n = \frac{T_{n-1} * 4 + T_{sampled}}{5}$



Main Model Components

- 18 concurrent timed automata
- 14 clocks, 14 integers



Reverse Engineering

- Unclear and incomplete specifications
- Method of Working
  1. Formulate hypothesis model
  2. Test
  3. **FAIL**-verdict ⇒ Refine model
  4. **(PASS)** ⇒ Confirm with Danfoss
- Detects differences between actual and modeled behavior
- ***Indicates promising error-detection capability***
- 4 examples

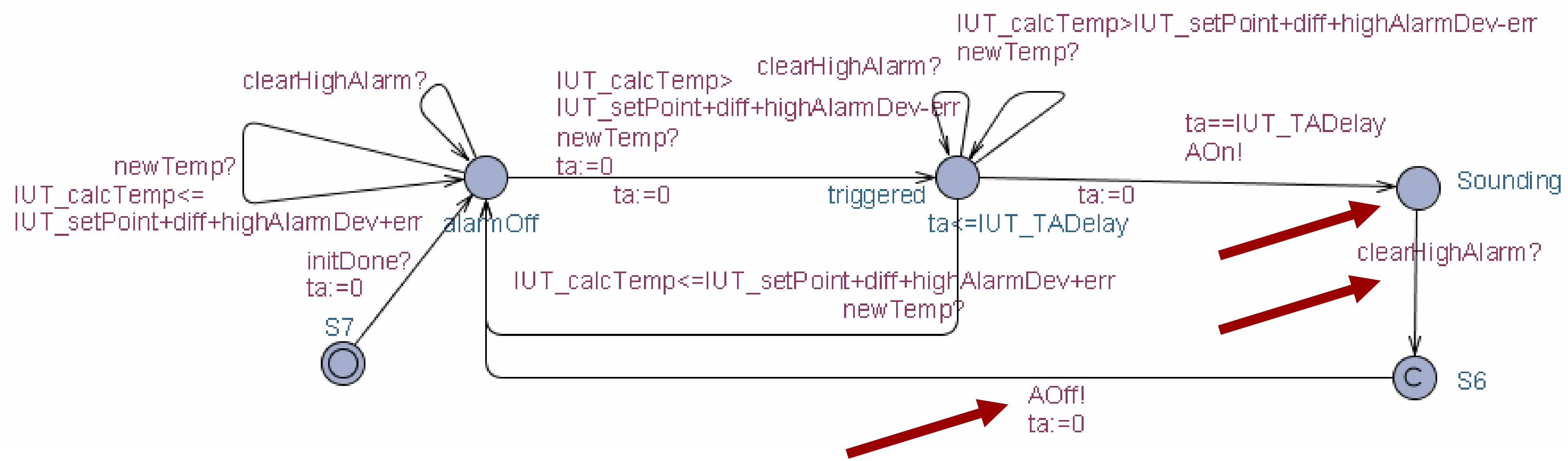
Ex1: Control Period

- Control actions issued when ”calculatedTemp” crosses thresholds

“periodic” weighted average: $$T_n = \frac{T_{n-1} * 4 + T_{sampled}}{5}$$

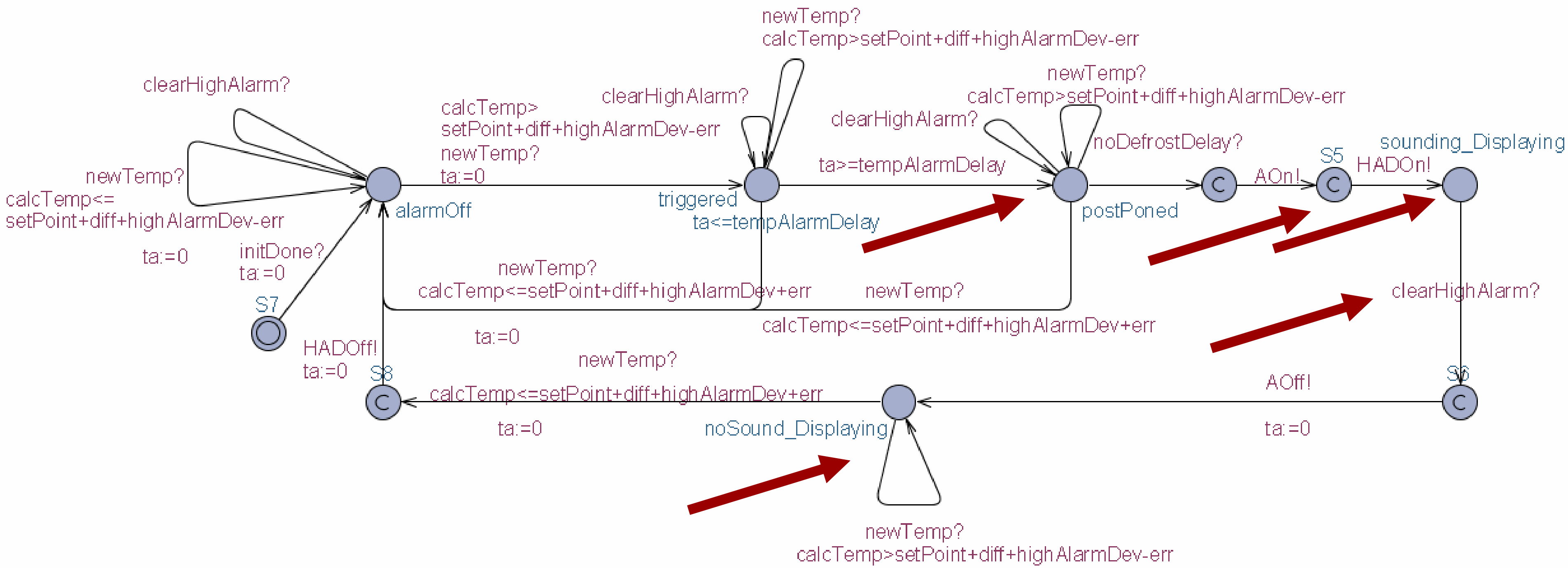
- No requirements on period given
- Tested to be 1.2 seconds

Ex2: High Alarm Monitor v1



Clearing the alarm do not switch off alarm state, only alarm relay

Ex2: High Alarm Monitor v2



- Add HighAlarmDisplay action
- Add location for “noSound, but alarmDisplaying”
- (Postpone alarms after defrosting)

Ex3: Defrosting and Alarms

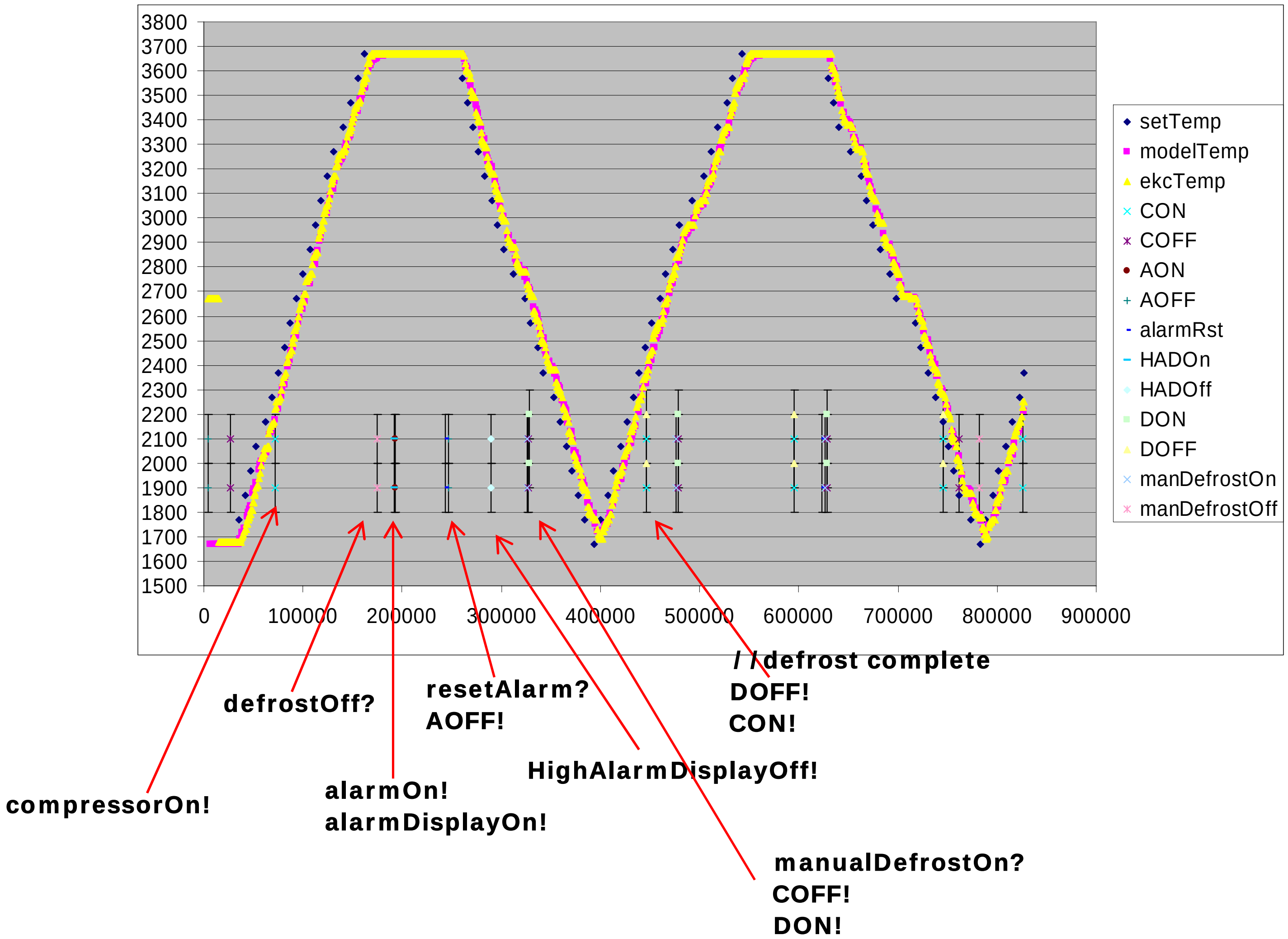
- When defrosting the temperature rises
- Postpone high temperature alarms during defrost
- System parameter alarmDelayAfterDefrost
- Several Interpretations
  1. Postpone alarmDelayAfterDefrost+alarmDelay after defrost?
  2. Postpone alarmDelayAfterDefrost+alarmDelay after highTemp detected?
  3. Postpone alarmdelayAfterDefrost until temperature becomes low; then use alarmDelay
- Option 3 applies!

Ex4: Defrost TimeTolerance

- Defrost relays engaged earlier and disengaged later than expected
- Assumed 2 seconds tolerance
- Defrosting takes long time
- Implementation uses a low resolution timer (10 seconds)

Example Test Run

(log visualization)



Testing=Environment emulation+monitoring

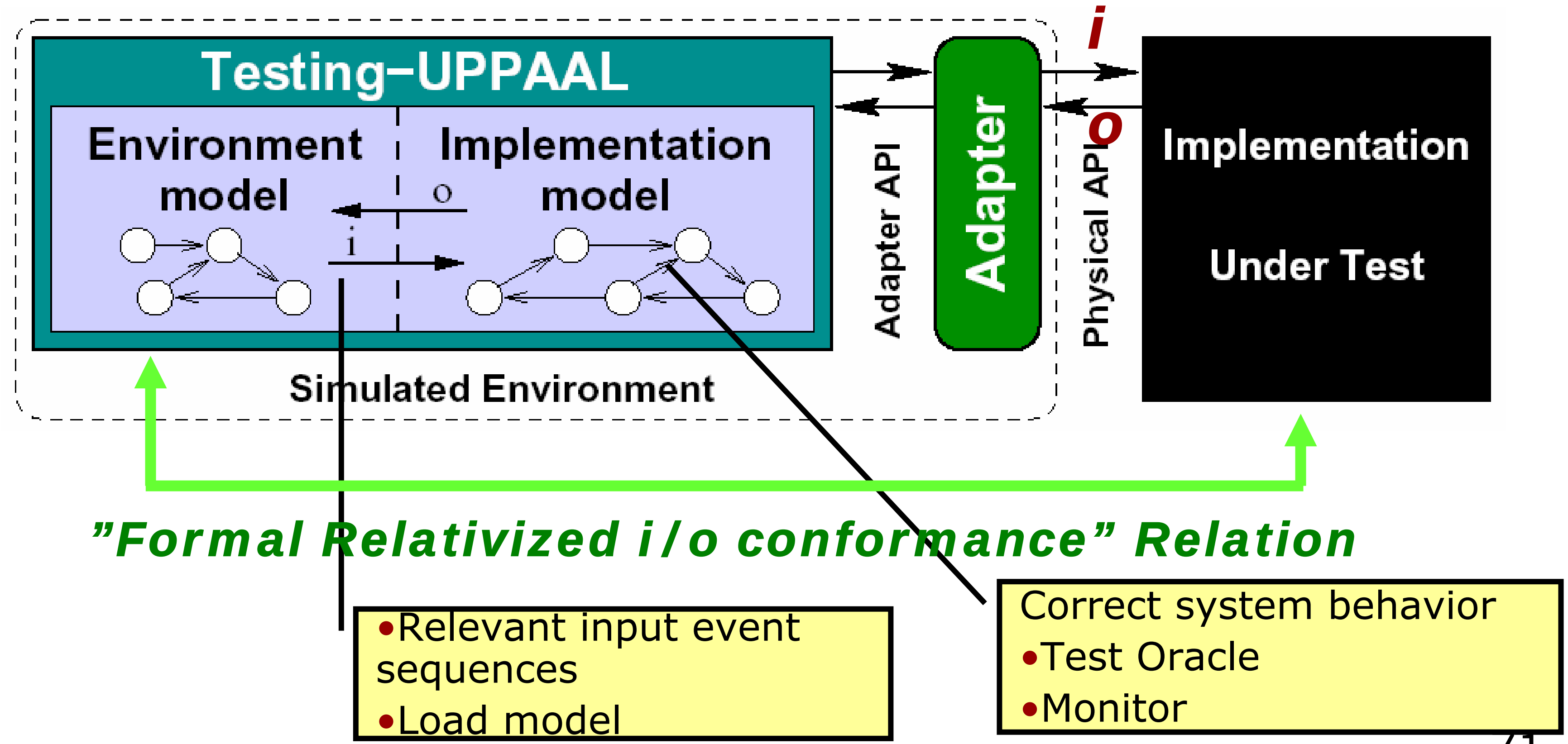
Aalborg University Denmark

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

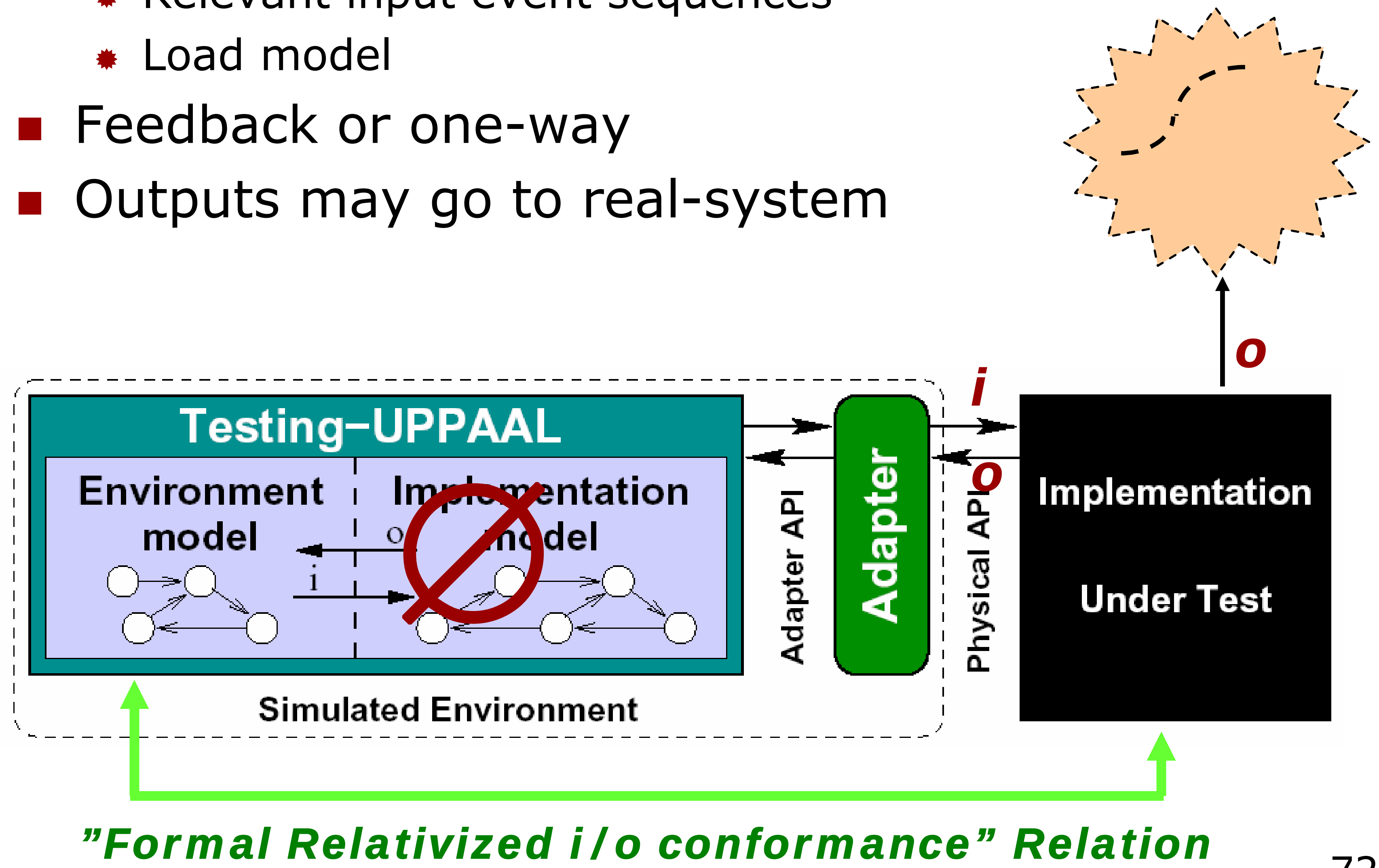
Testing

- Replace Systems Real Environment by Tester
- Tester provides inputs
- Tester observes outputs



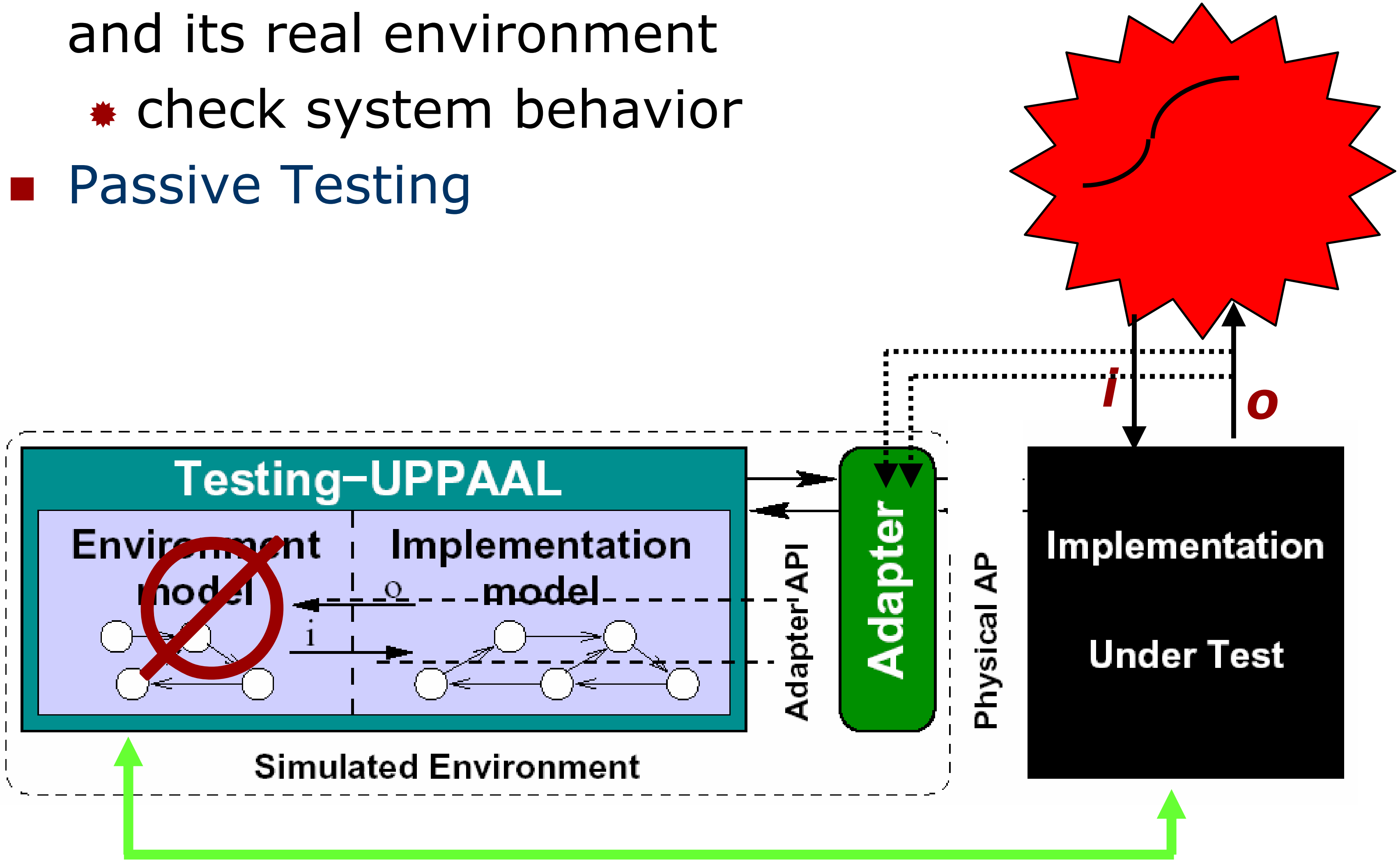
Environment Emulation

- Compute inputs from environment model
  - Relevant input event sequences
  - Load model
- Feedback or one-way
- Outputs may go to real-system



Monitoring

- Passively check communication between system and its real environment
  - check system behavior
- Passive Testing

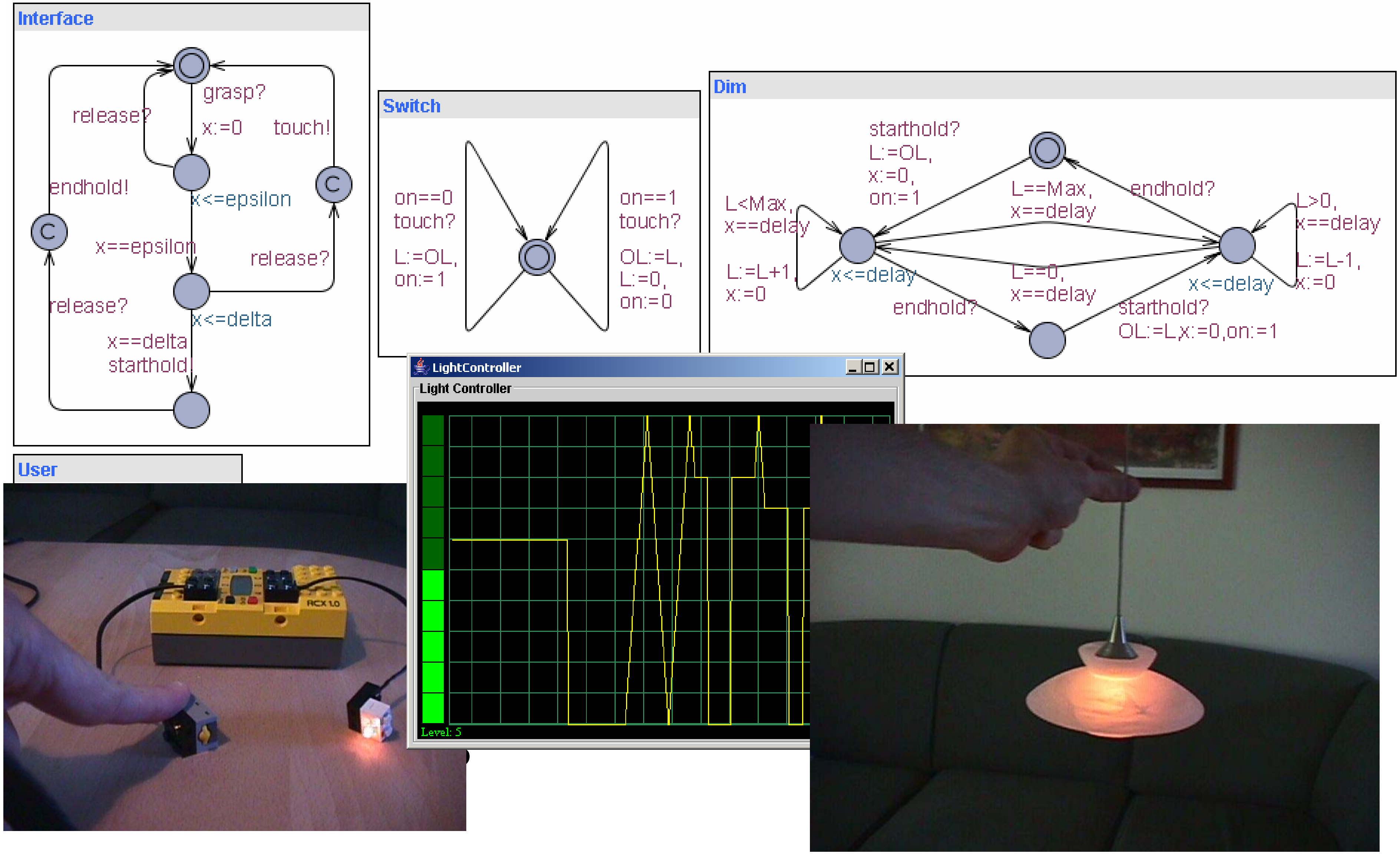


"Formal Relativized i/o conformance" Relation

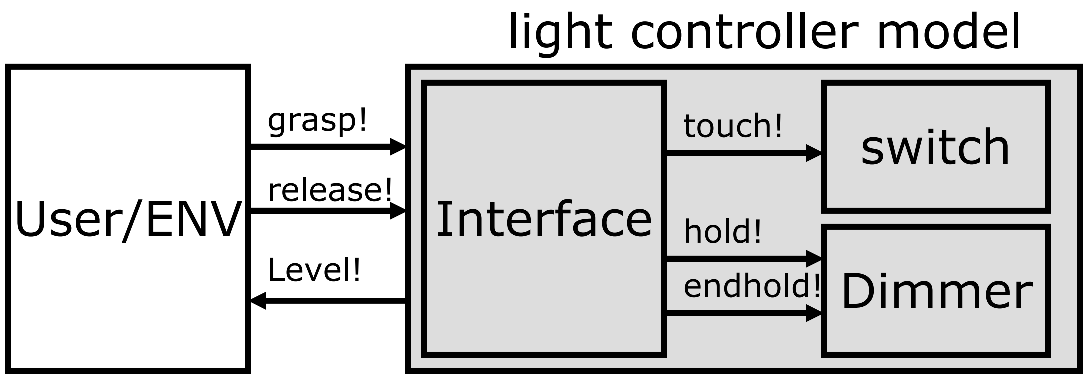
Conclusions

- Can accurately model EKC-like devices
- Can create models suitable for online testing
- Complete and detailed model not required
  - Select aspects
  - Abstraction
- MBT feasible even if specification is incomplete/unclear
- Promising error-detection capabilities
  - Differences between actual and specified behavior in industrial case
  - Academic mutation studies
- Excellent performance
- Very non-deterministic models causes very large state-sets which can become a computational bottleneck
- Real-time synchronization of IUT and tester is problematic

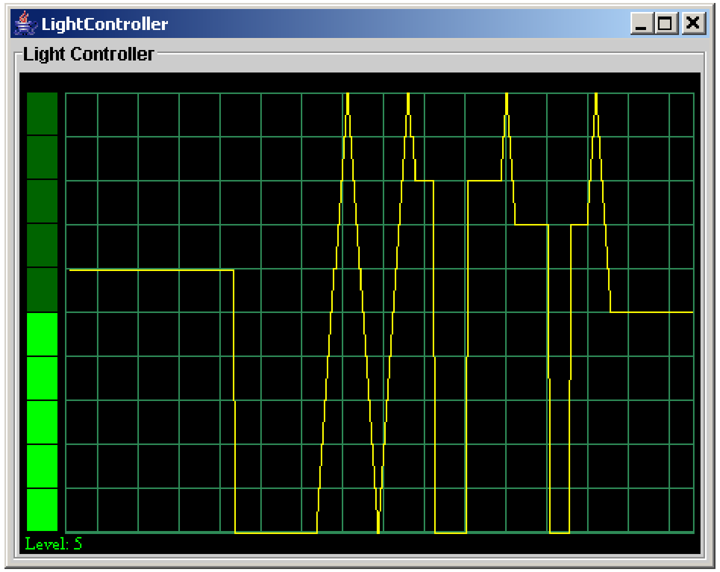
Touch-Sensitive Light-Controller

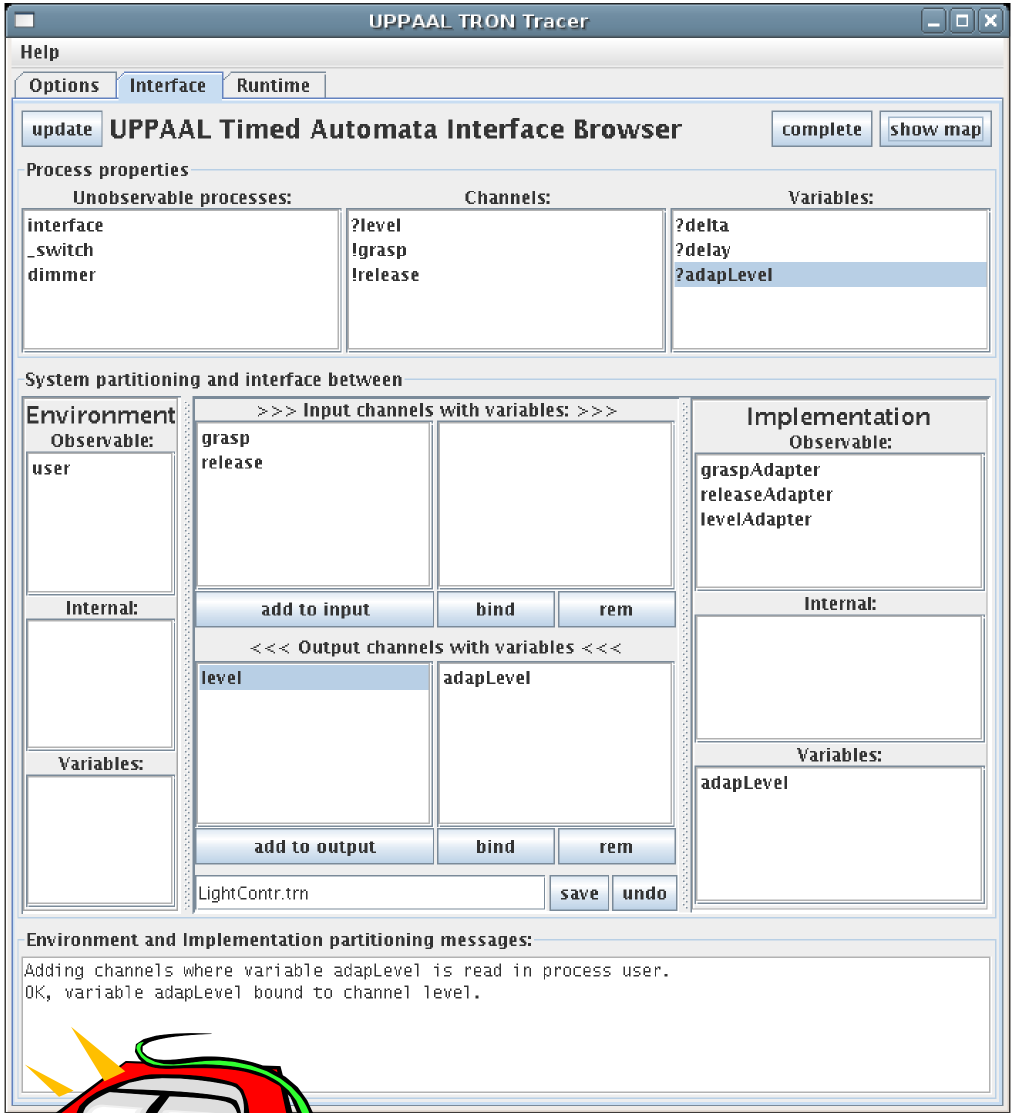


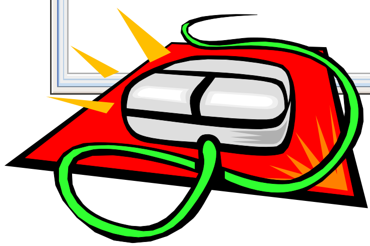
Touch-sensitive Light-Controller Model



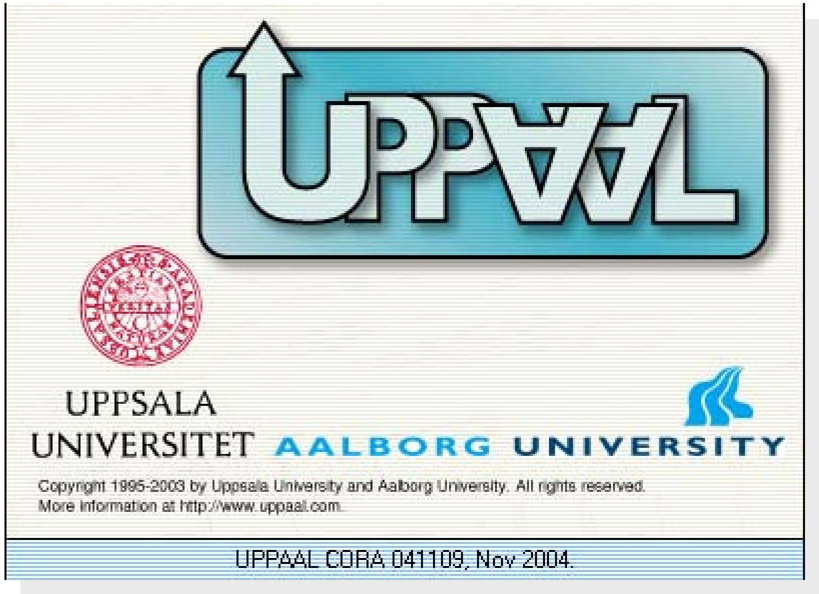
Demo


LightController
Light Controller
Level: 5


UPPAAL TRON Tracer
Help
Options
Interface
Runtime
update
UPPAAL Timed Automata Interface Browser
complete
show map
Process properties
Unobservable processes:
interface
_switch
dimmer
Channels:
?level
!grasp
!release
Variables:
?delta
?delay
?adapLevel
System partitioning and interface between
Environment
Observable:
user
Internal:
Variables:
>>> Input channels with variables: >>>
grasp
release
add to input
bind
rem
<<< Output channels with variables <<<
level
adapLevel
add to output
bind
rem
LightContr.trn
save
undo
Implementation
Observable:
graspAdapter
releaseAdapter
levelAdapter
Internal:
Variables:
adapLevel
Environment and Implementation partitioning messages:
Adding channels where variable adapLevel is read in process user.
OK, variable adapLevel bound to channel level.



END



CORA

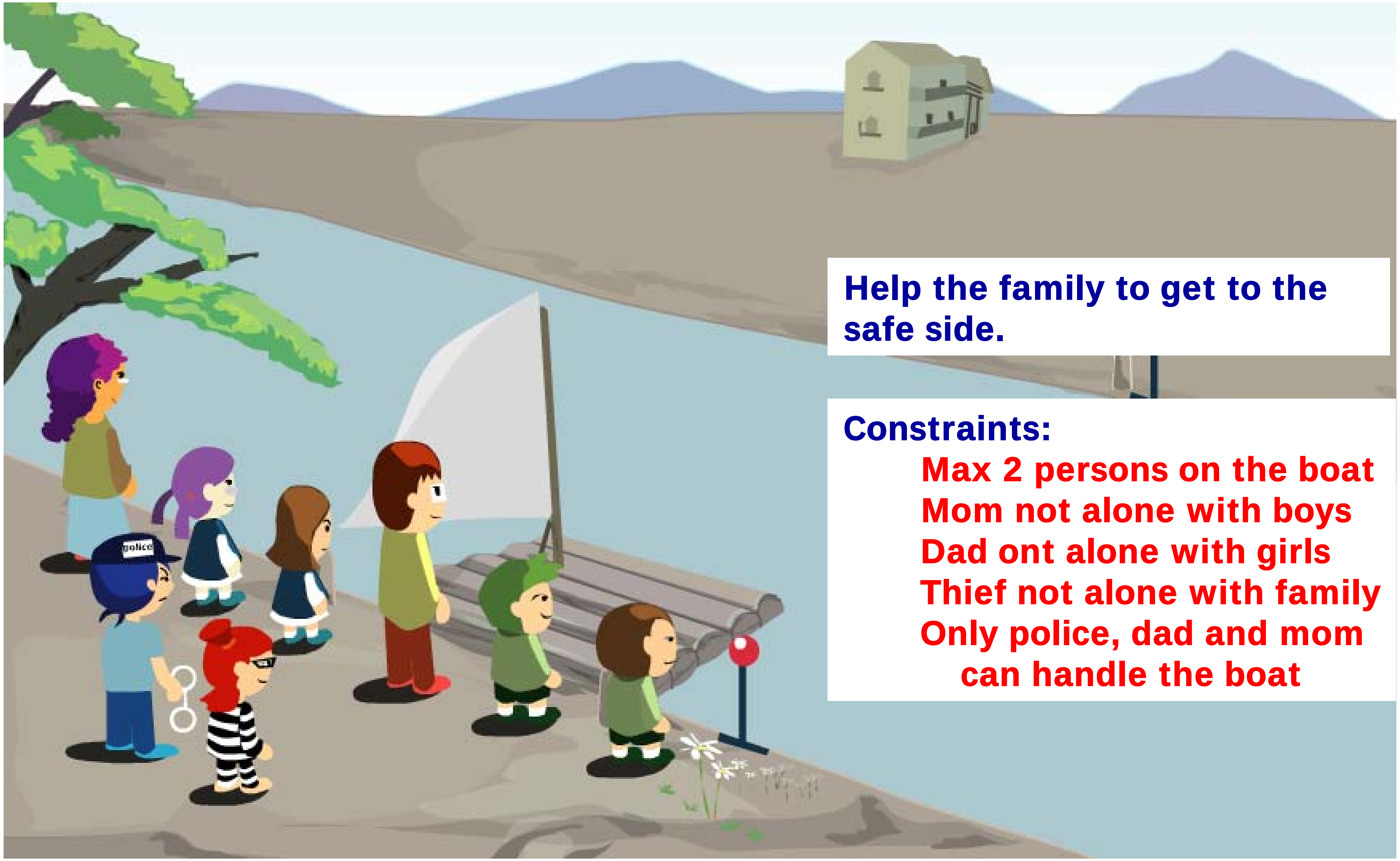
A tool for scheduling, optimization, and synthesis of real-time & embedded control programs

AALBORG UNIVERSITY DENMARK

BRICS Basic Research in Computer Science

CISS CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

Scheduling & Planning



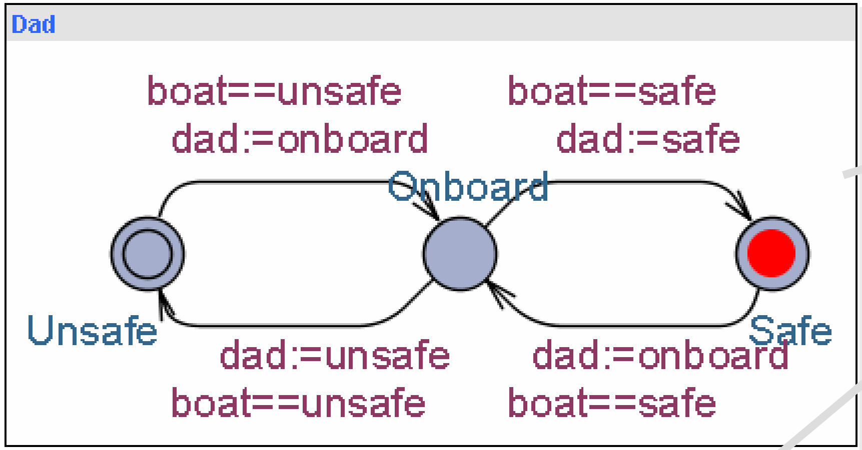
Help the family to get to the safe side.

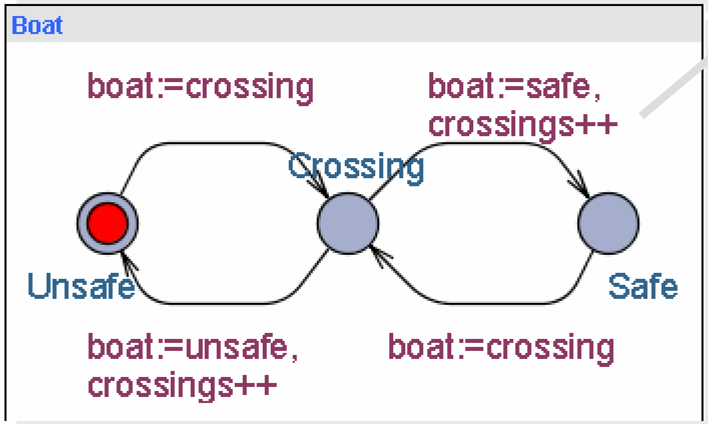
Constraints:

- **Max 2 persons on the boat**
- **Mom not alone with boys**
- **Dad ont alone with girls**
- **Thief not alone with family**
- **Only police, dad and mom can handle the boat**

Scheduling & Planning

Solve Scheduling Problem using UPPAAL

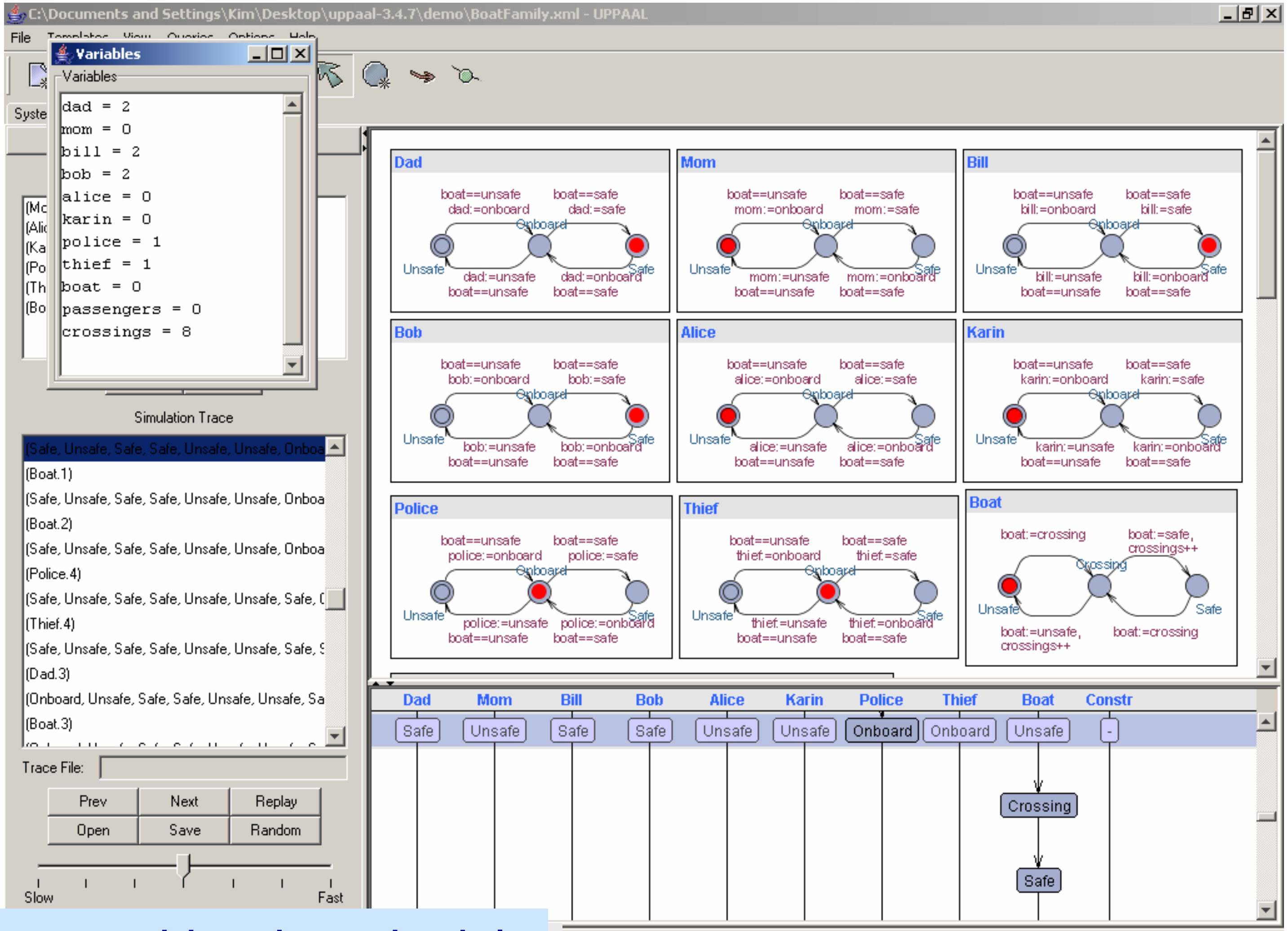




Constraints

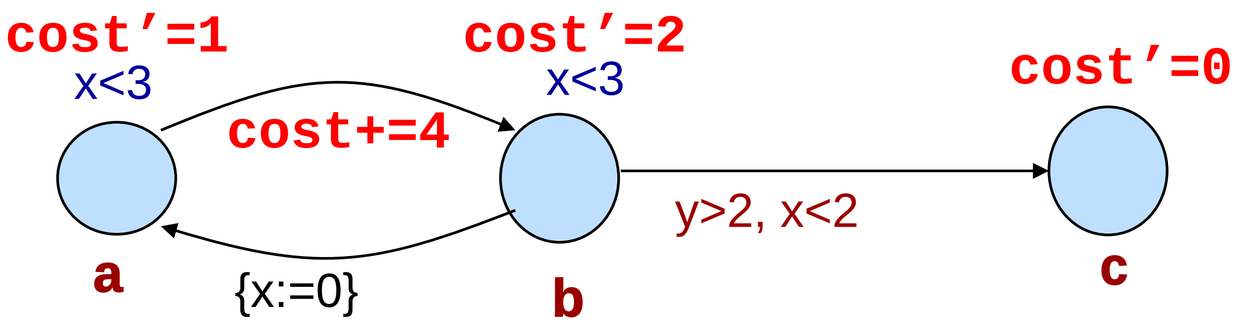
dad==alice imply mom==dad,
dad==karin imply mom==dad,
mom==bill imply dad==mom,
mom==bob imply dad==mom,
boat==crossing imply (dad==onboard or police==onboard or mom==onboard),
((dad==onboard)+(bill==onboard)+ (bob==onboard) +
(mom==onboard) + (alice==onboard) + (karin==onboard) +
(police==onboard) + (thief==onboard))<=2,
thief==dad imply police==thief,
thief==mom imply police==thief,
thief==bill imply police==thief,
thief==bob imply police==thief,
thief==alice imply police==thief,
thief==karin imply police==thief

Scheduling & Planning



UPPAAL provides the Schedule

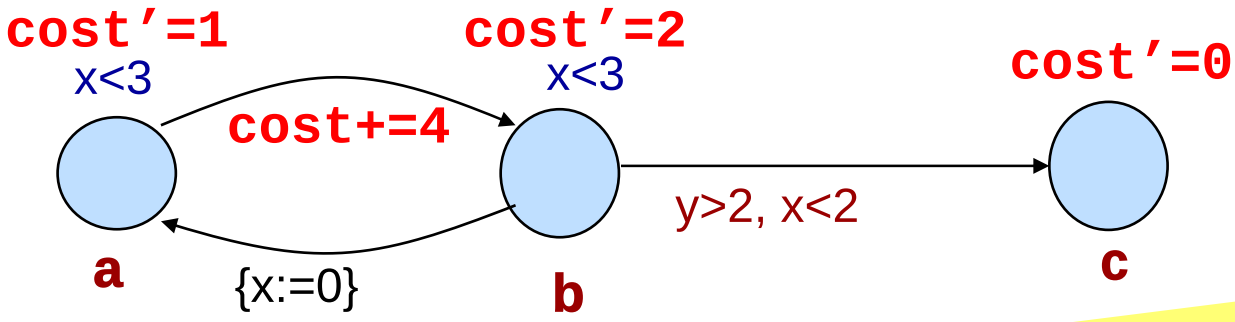
Linearly Priced Timed Automata:



- **Timed Automata + Costs on transitions and locations.**
  - Cost of performing transition: **Transition cost**.
  - Cost of performing delay **d**: (**d** x **Location cost**).

- Trace: (**a**,x=y=0) $\xrightarrow[4]{}$ (**b**,x=y=0) $\xrightarrow[2.5 \times 2]{\varepsilon(2.5)}$ (**b**,x=y=2.5) $\xrightarrow[0]{}$ (**a**,x=0,y=2.5)
- Cost of Execution Trace: Sum of costs: **4 + 5 + 0 = 9**

Linearly Priced Timed Automata:



- **Timed Automata**

Problem :

Find the minimum cost of reaching location **c**

-

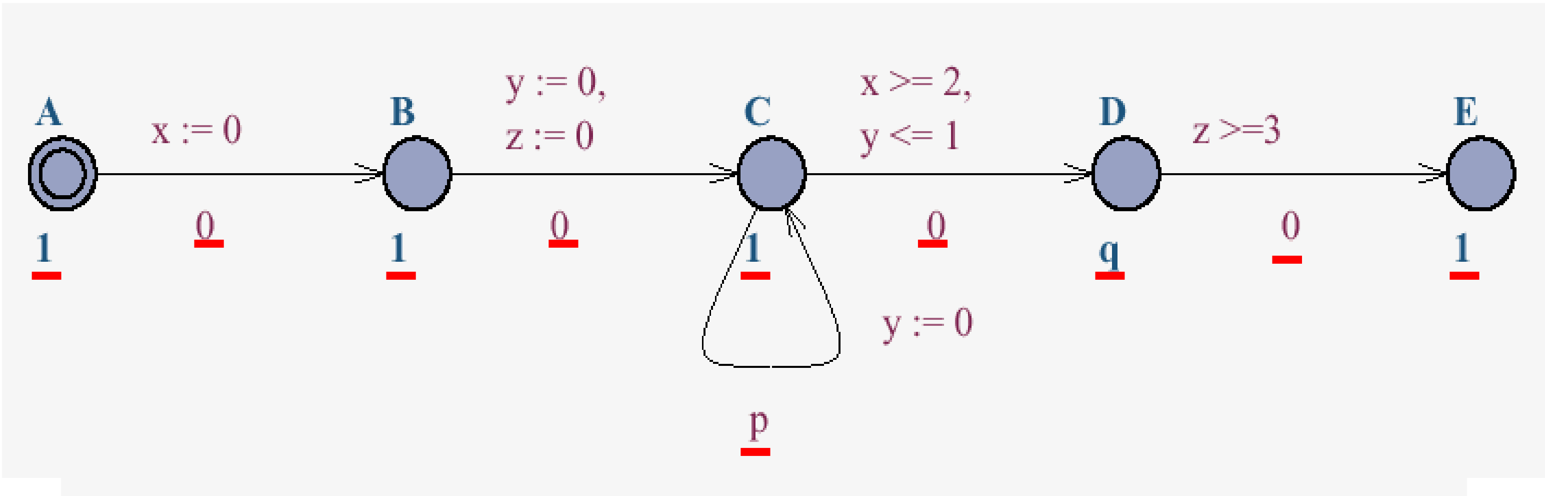
Trac(... → (**b**,x=y=2.5) → (**a**,x=0,y=2.5)

2.5 x **2** ... **0**

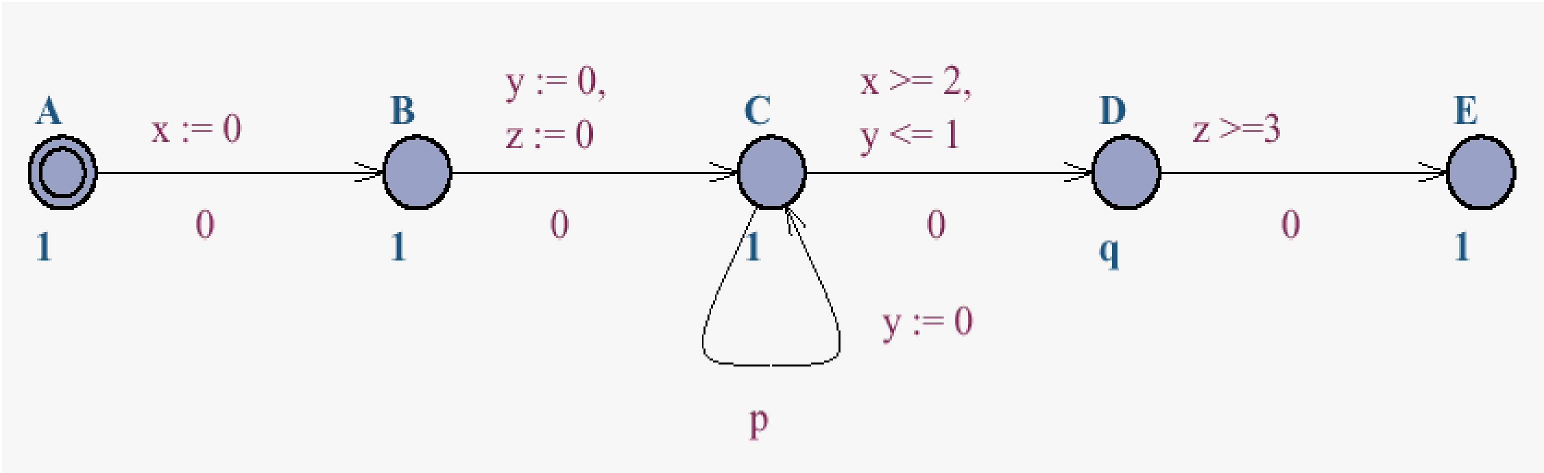
- Cost of Execution Trace: Sum of costs: **4 + 5 + 0 = 9**

Example

Prices

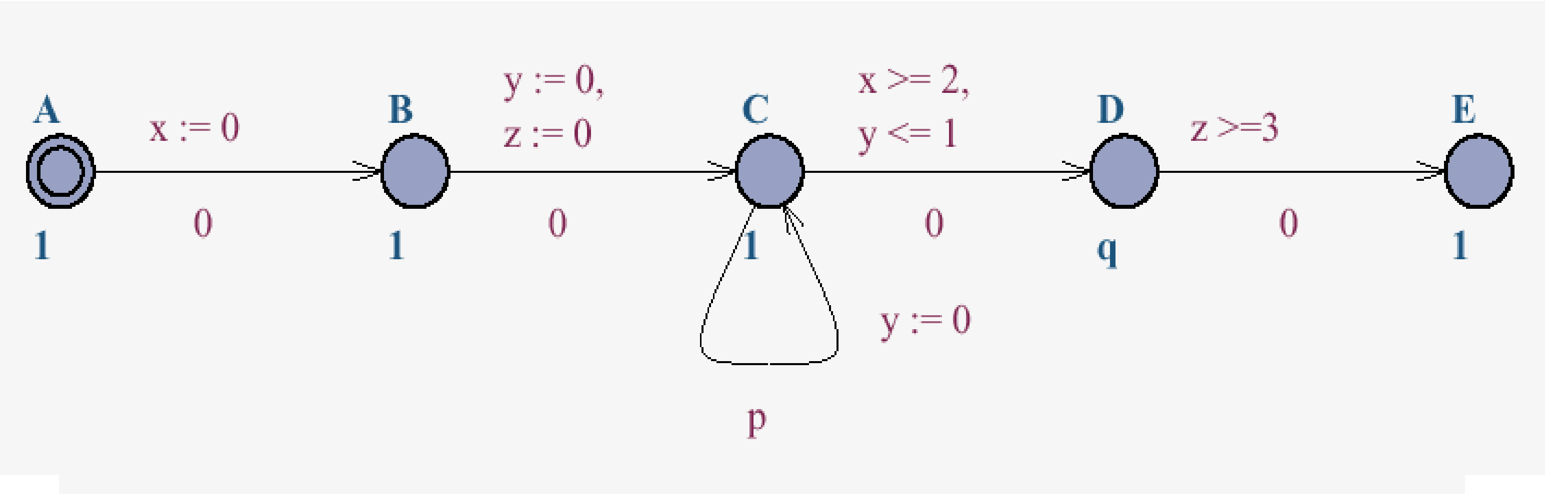


Example (execution)



$$(A,0,0,0) \xrightarrow{\tau,0} \xrightarrow{\tau,0} \xrightarrow{\epsilon(2),2} (C,2,2,2) \xrightarrow{\tau,p} (C,2,0,2) \xrightarrow{\tau,0} \xrightarrow{\epsilon(1),q} (D,3,1,3) \xrightarrow{\tau,0} (E,3,1,3)$$

Example (min-cost)



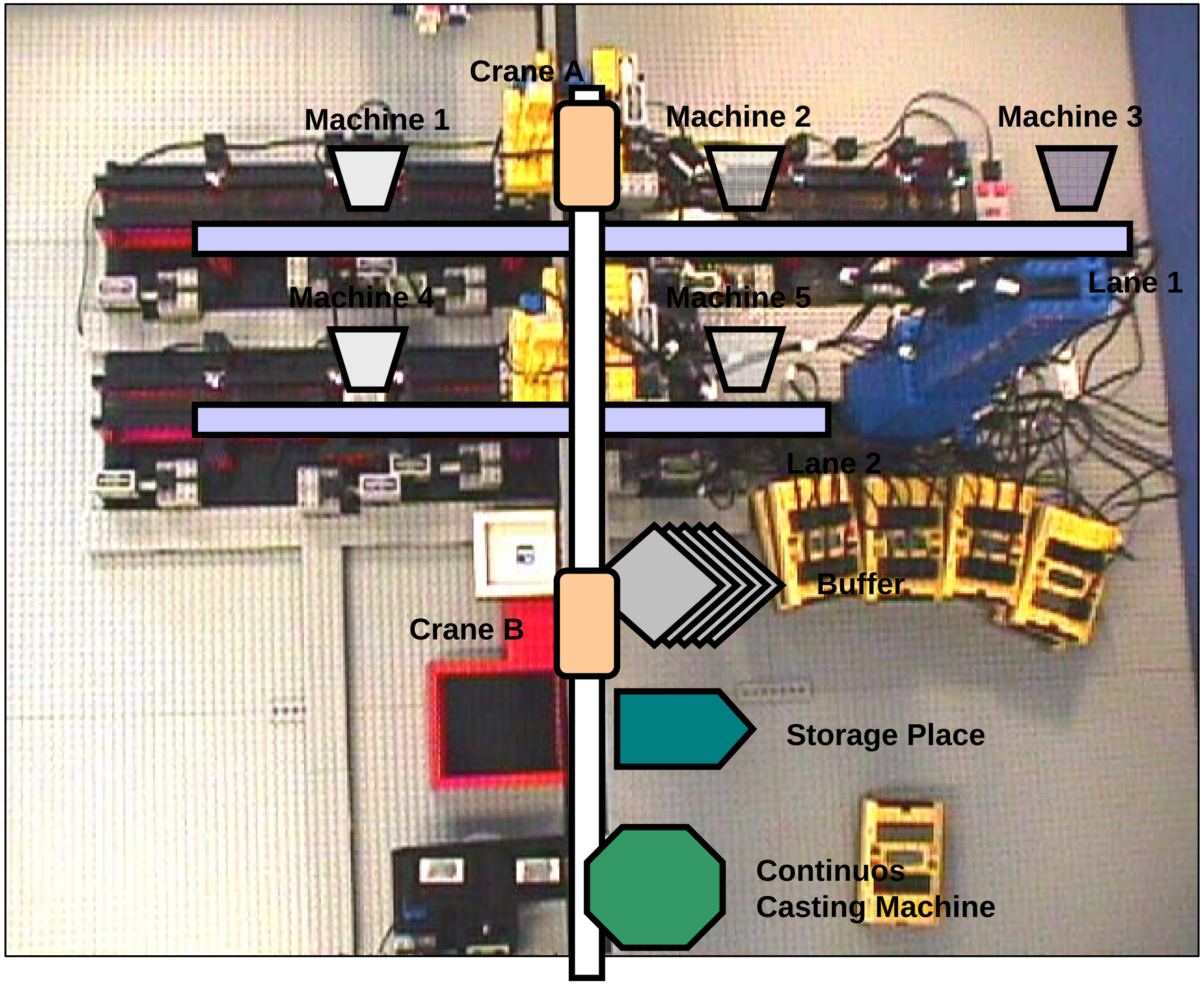
Optimal cost of reaching E depends on values p and q:

$$\min(3+p, 2+p+q, 2+2q)$$

SIDMAR Steel Production Plant

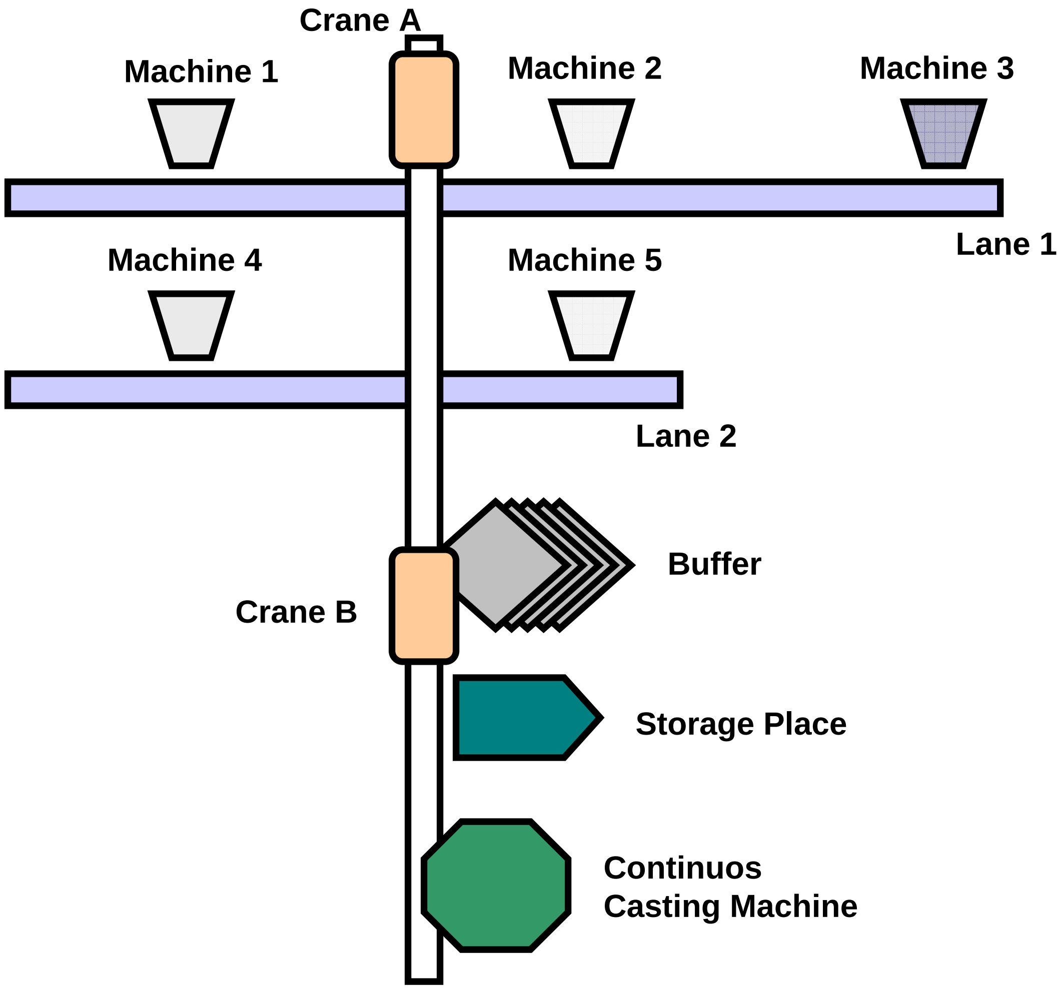
Gent, Belgium

SIDMAR Steel Production Plant



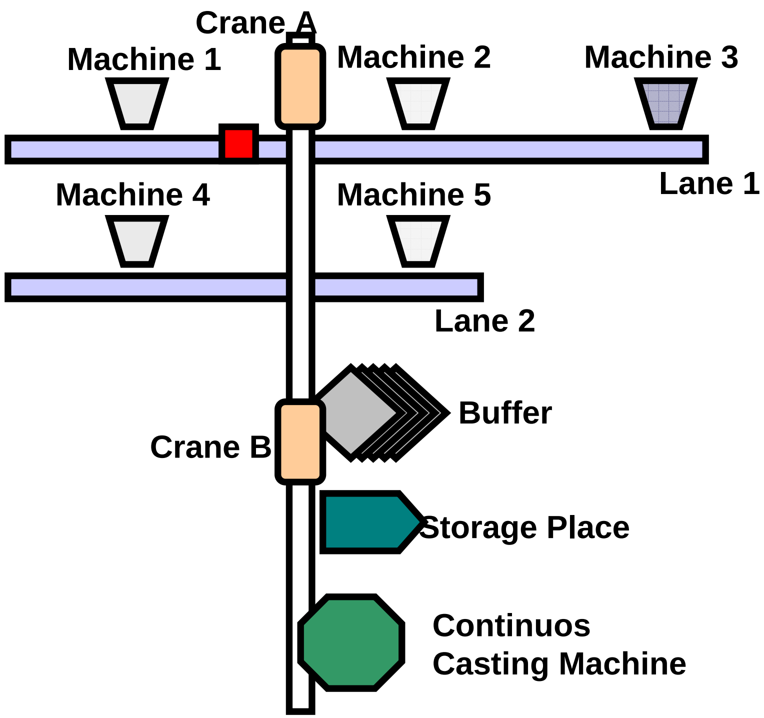
In LEGO MindStorm

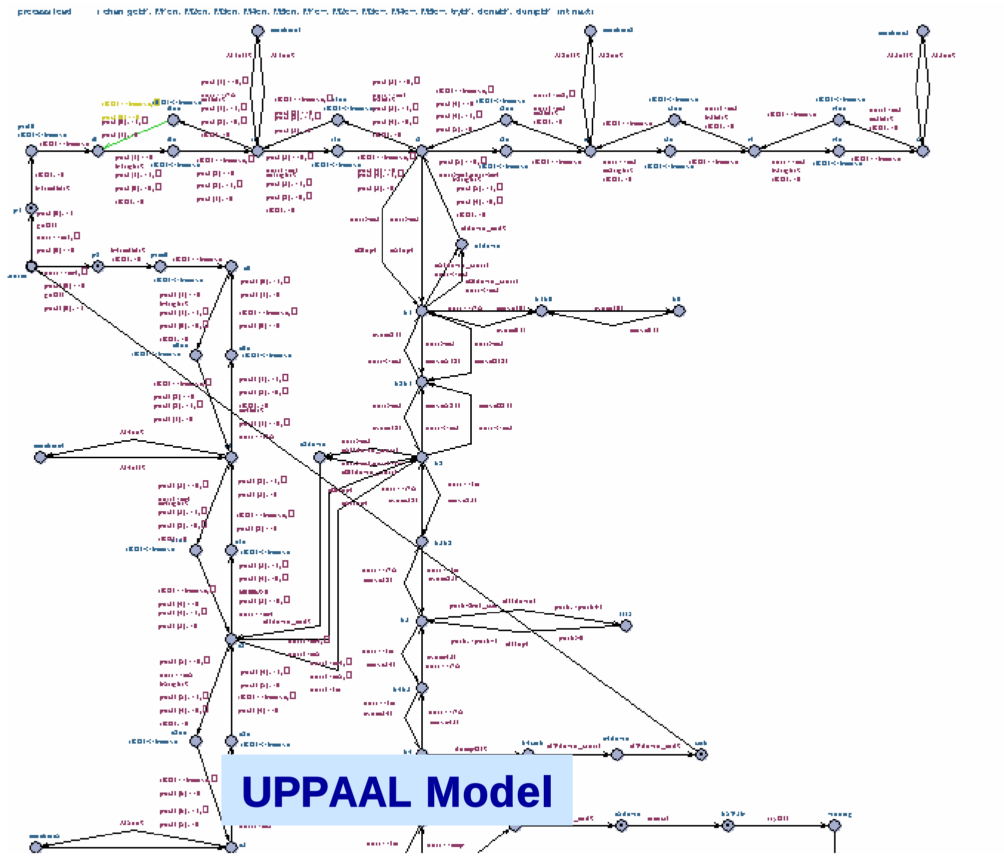
SIDMAR Overview



SIDMAR Modelling

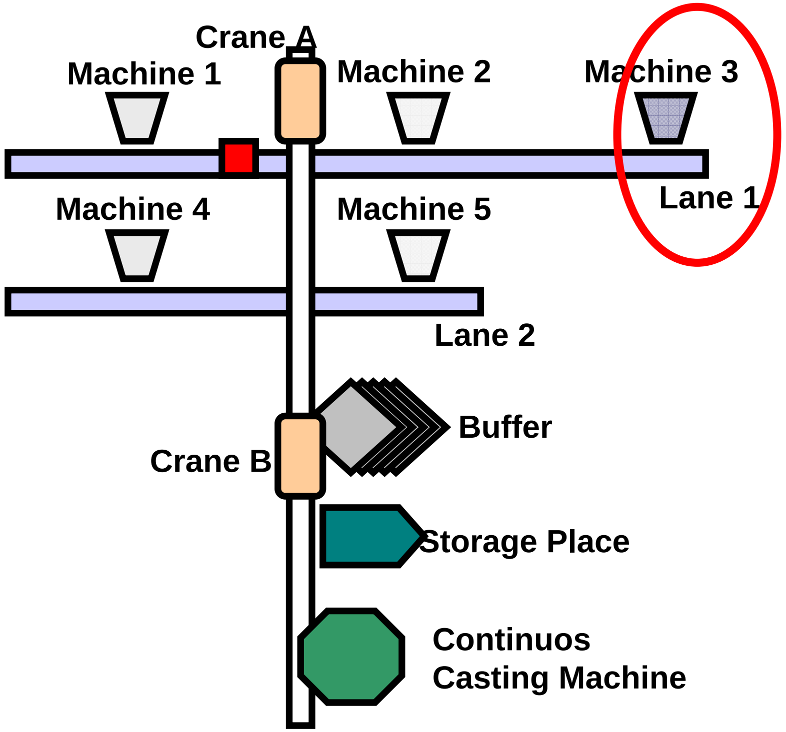
A Single Load

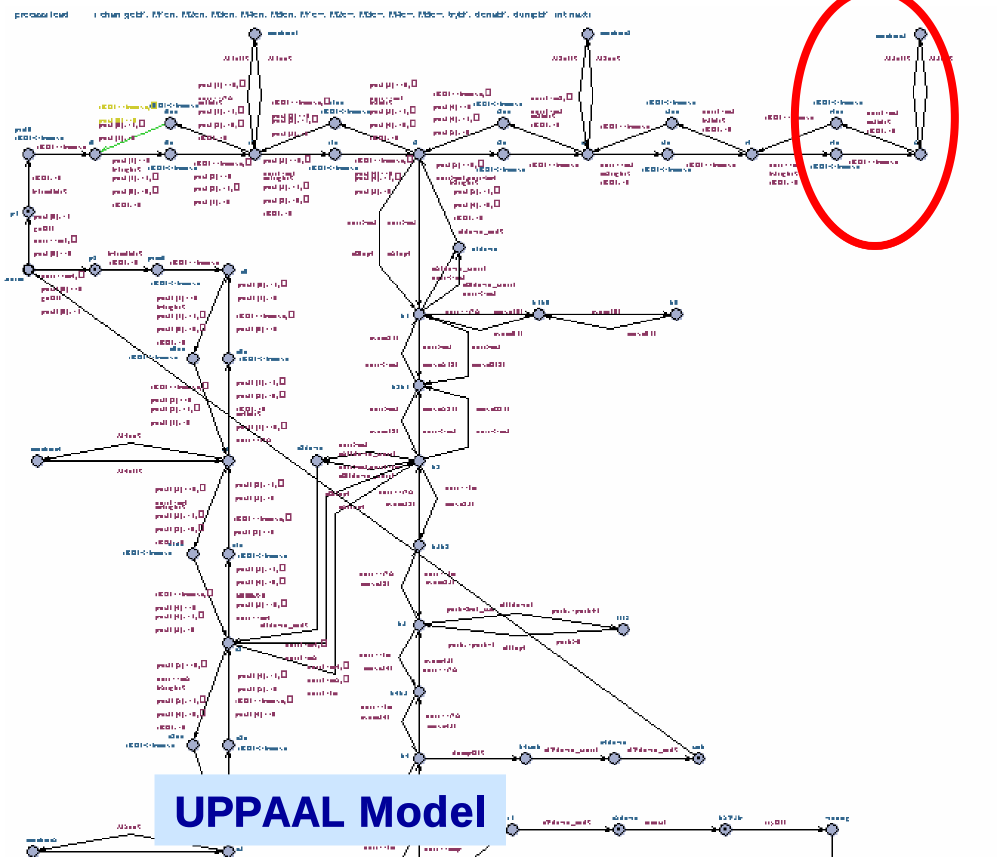




SIDMAR Modelling

A Single Load

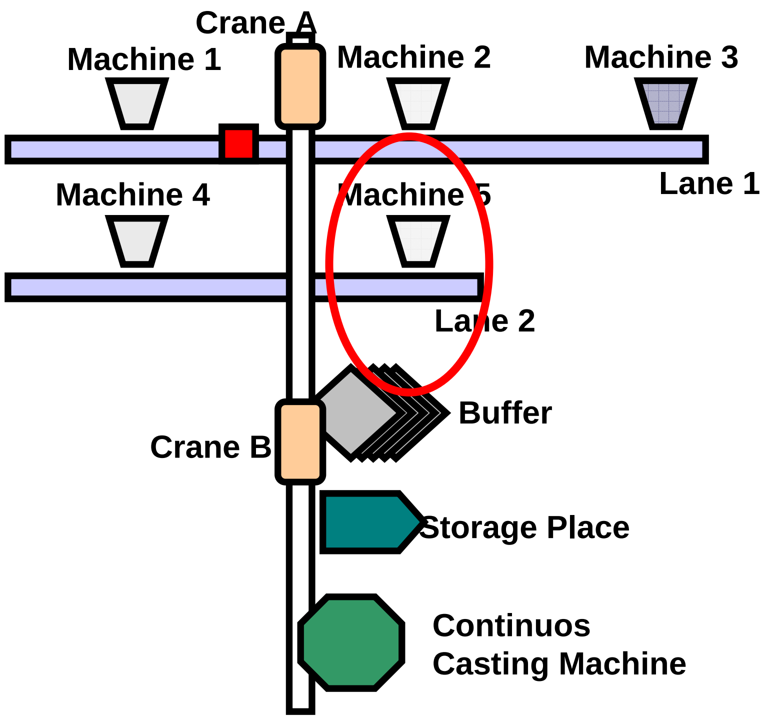


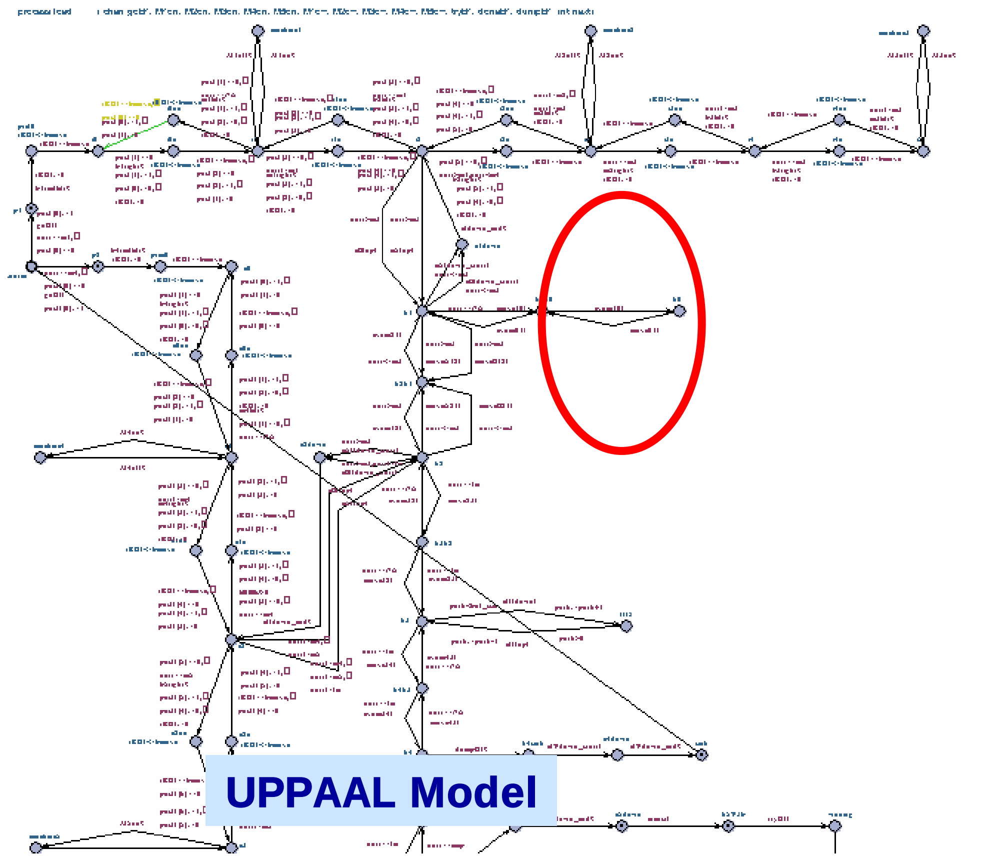


UPPAAL Model

SIDMAR Modelling

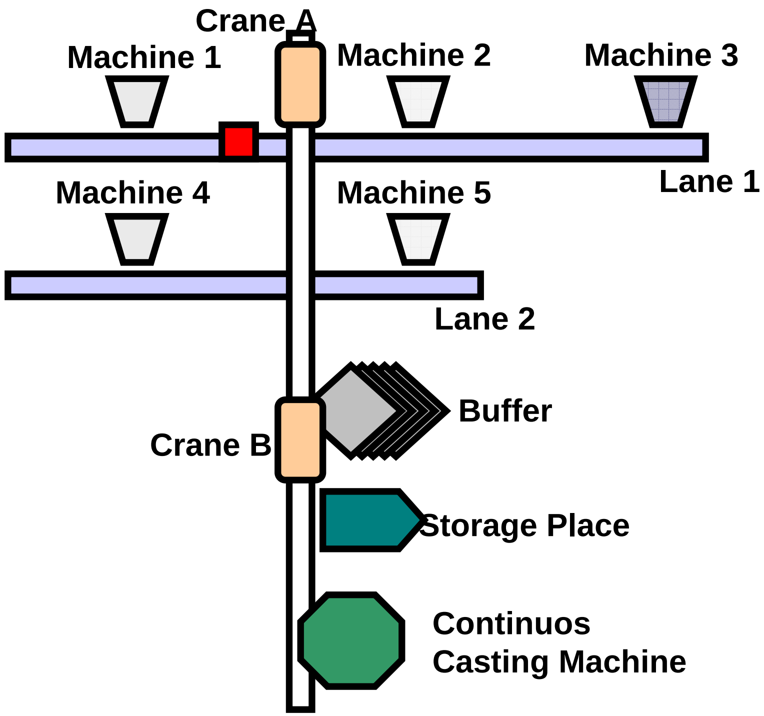
A Single Load

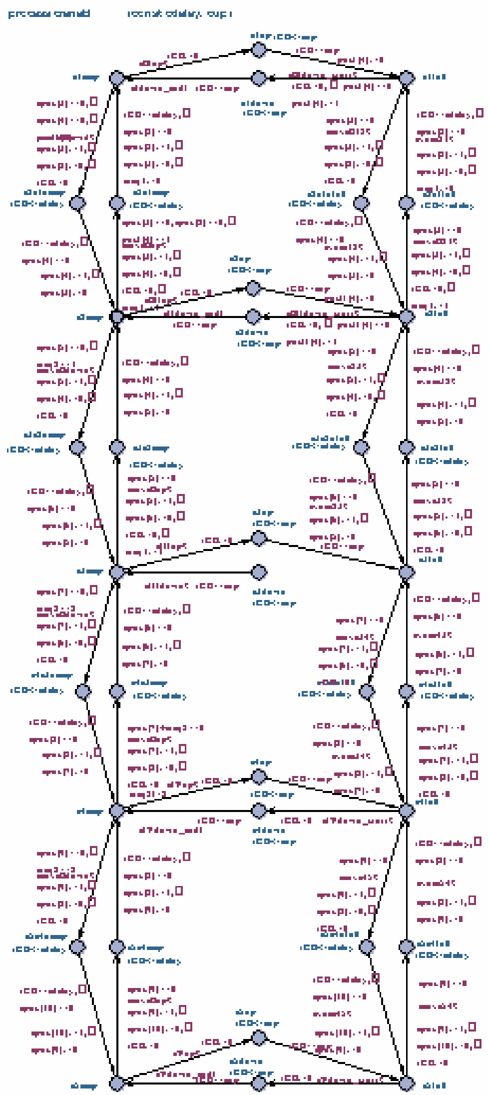




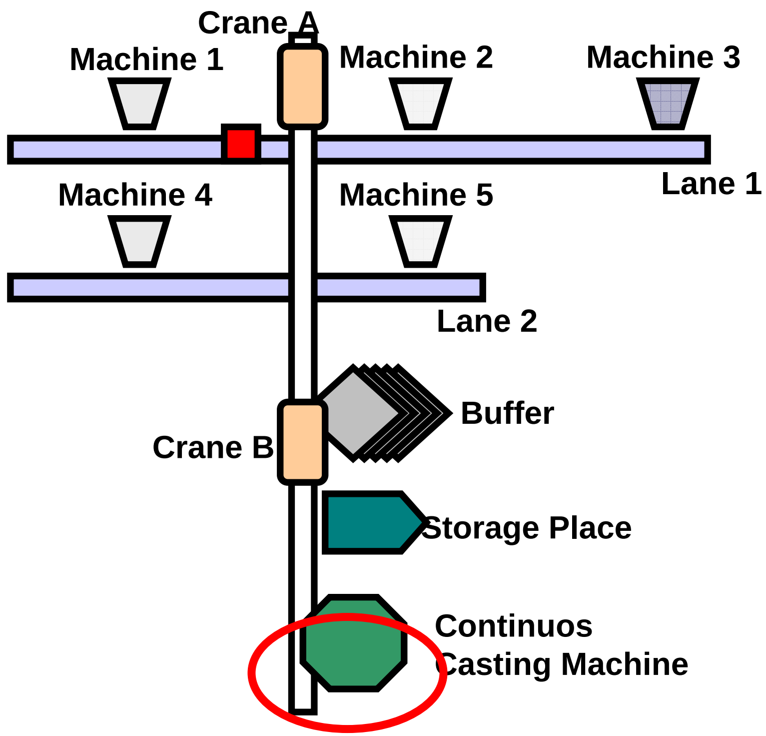
SIDMAR Modelling

Crane B

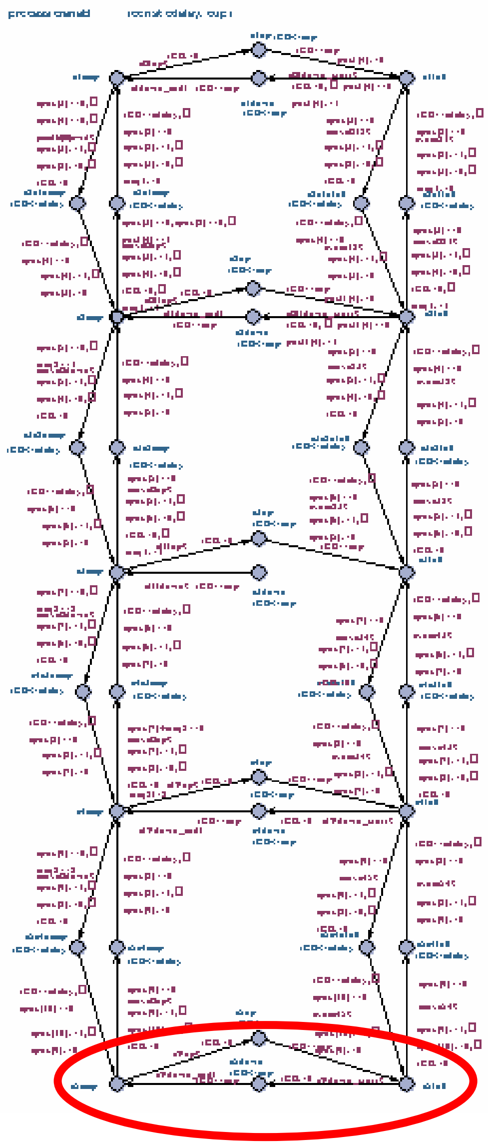




SIDMAR Modelling

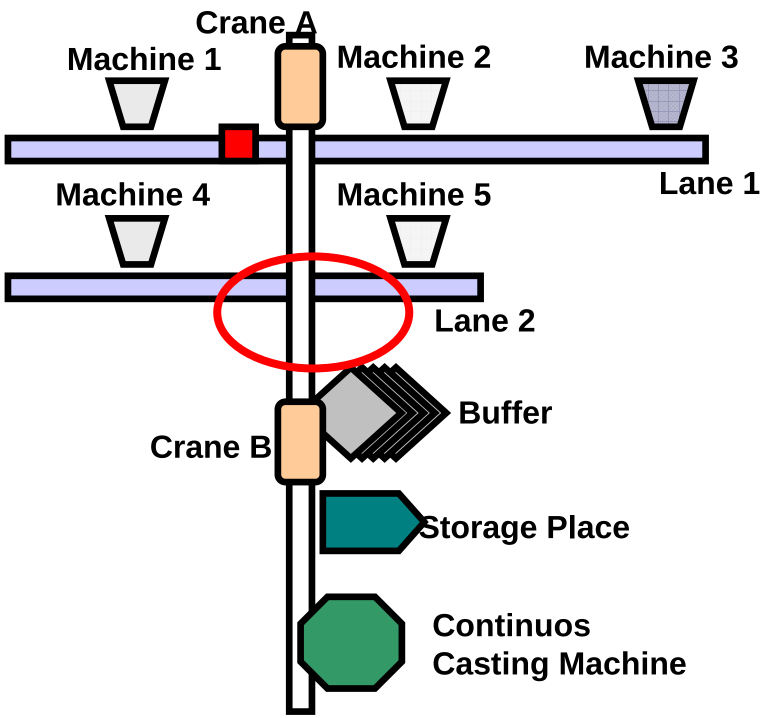


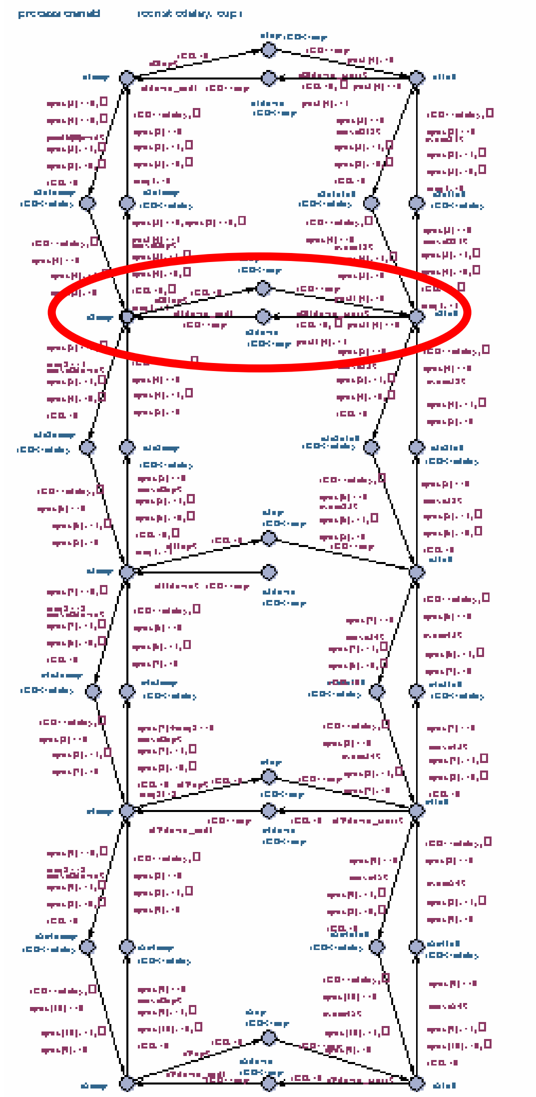
Crane B



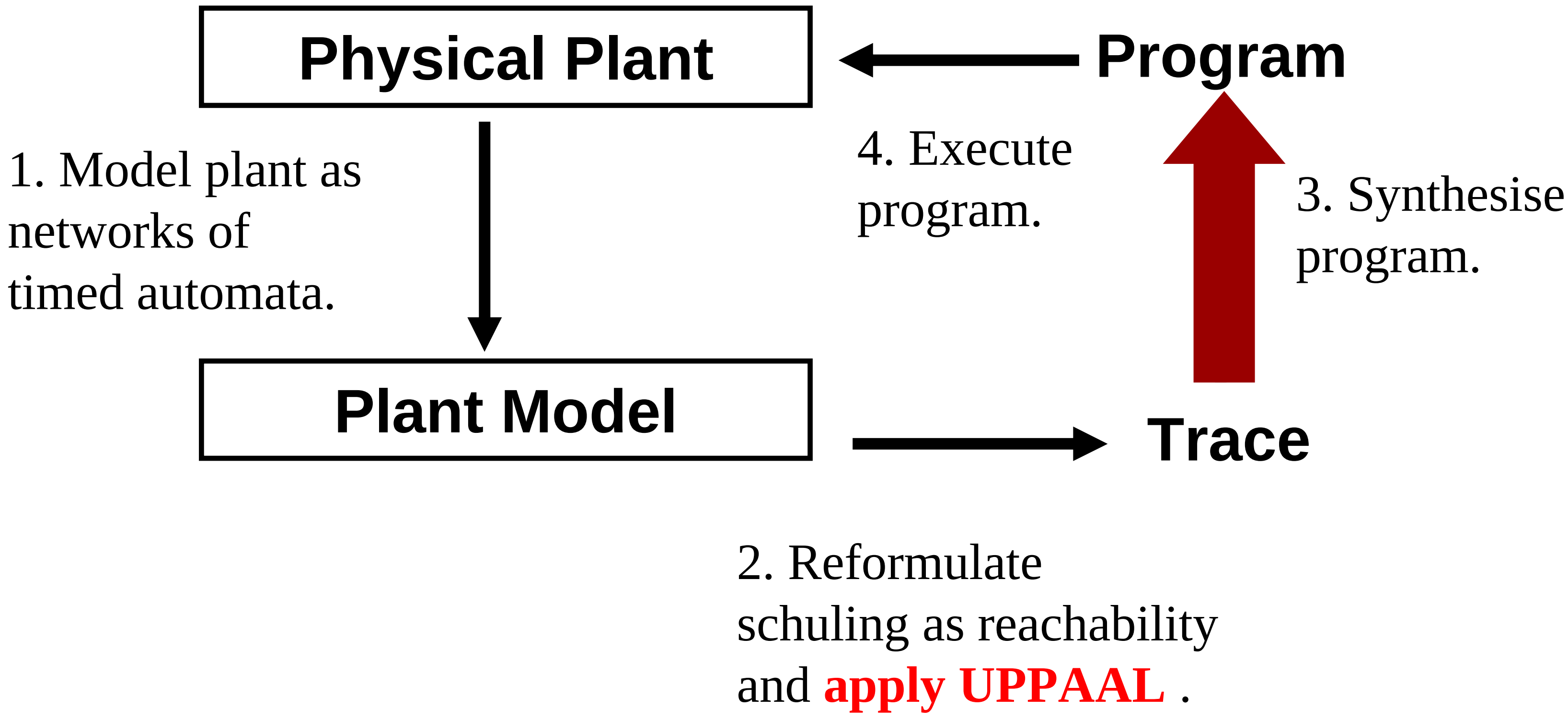
SIDMAR Modelling

Crane B





Modus Operandi



Extracting Programs

1971 lines of RCX code (n=5),
24860 - " - (n=60).

Trace

...
(loadB1.p1 recipeB1.gotoT1
loadB2...
{ loadB1.x=5 recipeB1.tot=5
recipeB1...
Sync: b1right
(loadB1.pre recipeB1.gotoT1
loadB2...
{ loadB1.x=5 recipeB1.tot=5
recipeB1...
delay(5)
(loadB1.pre recipeB1.gotoT1
loadB2...
{ loadB1.x=10 recipeB1.tot=10
recipe...
Sync: B1M1on
(loadB1.onM1 recipeB1.onT1
loadB2...
{ loadB1.x=0 recipeB1.tot=10
recipe...
delay(10)
(loadB1.onM1 recipeB1.onT1
loadB2...

Schedule

...

belt1 right

delay 5

load B1 on Machine 1

delay 10

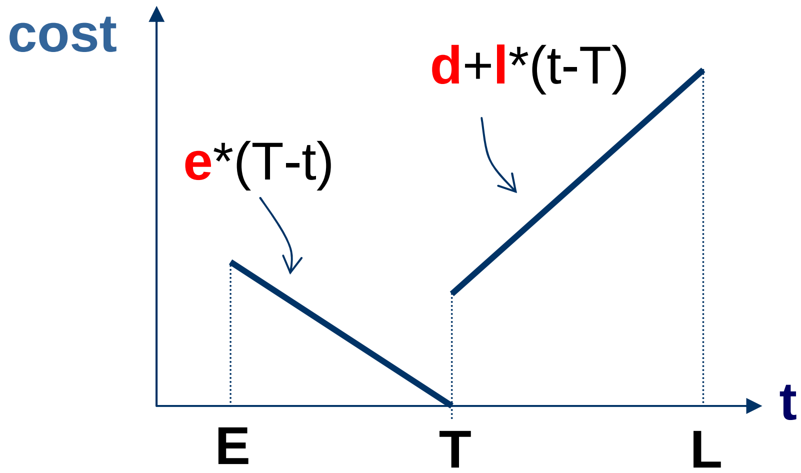
load B1 off Machine 1
...

Program

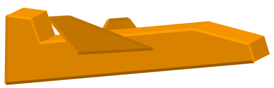
...

```
// Belt Unit 1 move RIGHT
PB.SendPBMessage 2, 20

// Delay 5
PB.Wait 2, 500

// Machine 1 START
PB.SendPBMessage 2, 23

// Delay 10
PB.Wait 2, 100

// Machine 2 STOP
PB.SendPBMessage 2,24
...
```

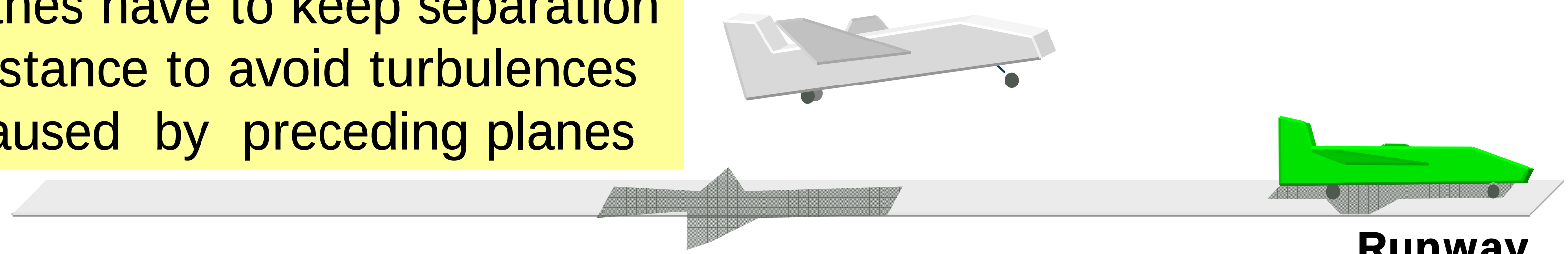
Example: Aircraft Landing



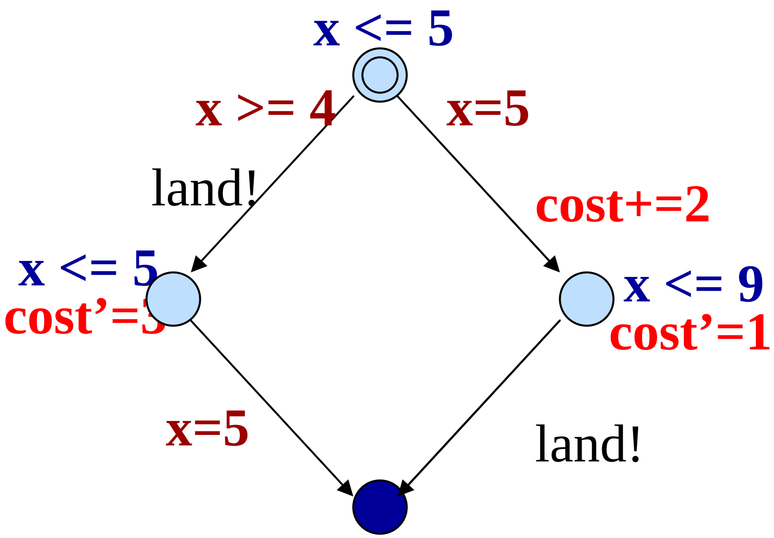
E earliest landing time
T target time
L latest time
e cost rate for being early
l cost rate for being late
d fixed cost for being late



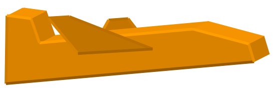
Planes have to keep separation distance to avoid turbulences caused by preceding planes



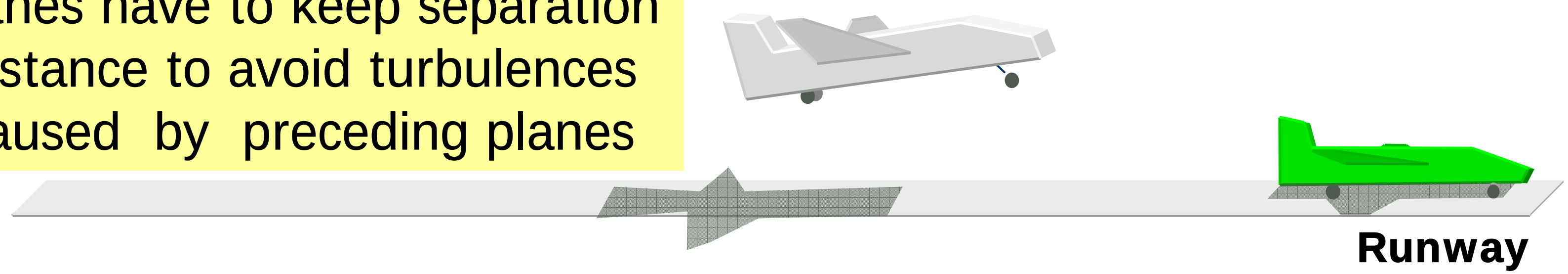
Example: Aircraft Landing



4 earliest landing time
5 target time
9 latest time
3 cost rate for being early
1 cost rate for being late
2 fixed cost for being late



Planes have to keep separation distance to avoid turbulences caused by preceding planes



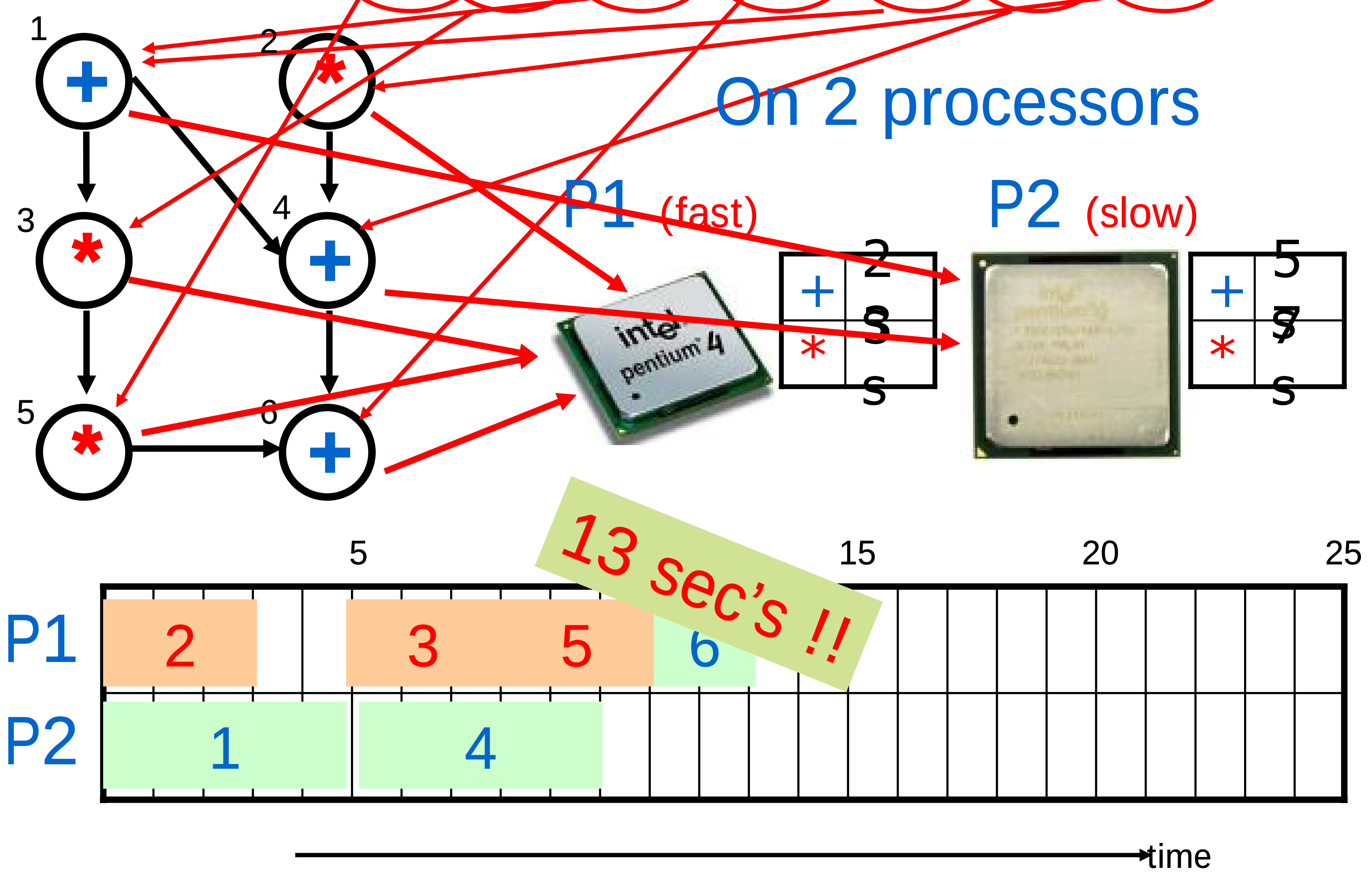
Aircraft Landing

Source of examples:
Baesley et al'2000

	problem instance	1	2	3	4	5	6	7
	number of planes	10	15	20	20	20	30	44
	number of types	2	2	2	2	2	4	2
1	optimal value	700	1480	820	2520	3100	24442	1550
	explored states	481	2149	920	5693	15069	122	662
	cputime (secs)	4.19	25.30	11.05	87.67	220.22	0.60	4.27
2	optimal value	90	210	60	640	650	554	0
	explored states	1218	1797	669	28821	47993	9035	92
	cputime (secs)	17.87	39.92	11.02	755.84	1085.08	123.72	1.06
3	optimal value	0	0	0	130	170	0	N/A
	explored states	24	46	84	207715	189602	62	
	cputime (secs)	0.36	0.70	1.71	14786.19	12461.47	0.68	
4	optimal value	N/A	N/A	N/A	0	0	N/A	N/A
	explored states				65	64		
	cputime (secs)				1.97	1.53		

Ressource Optimal Scheduling

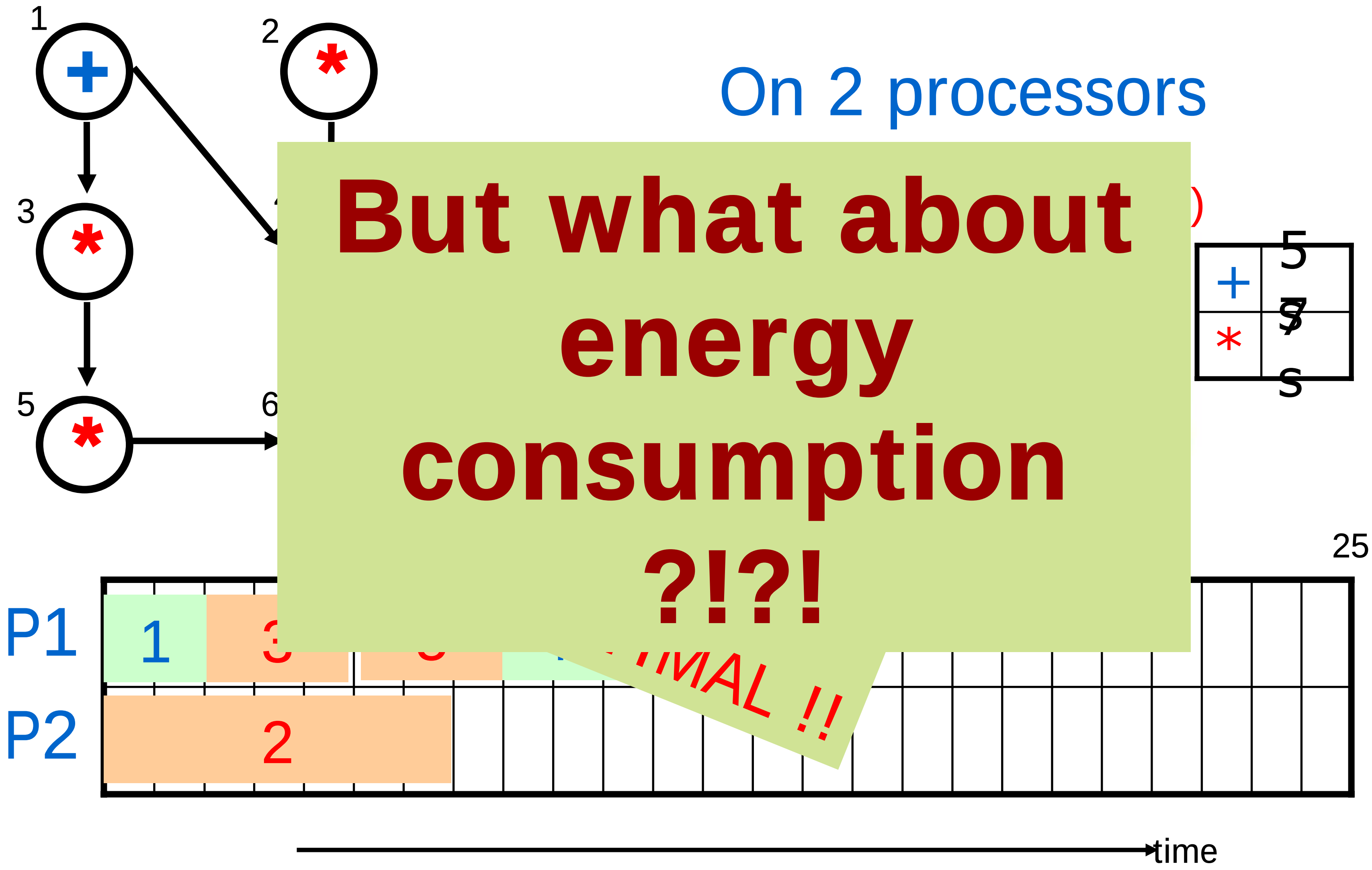
Compute : (D * ((C * (A + B)) + ((A + B) + (C * D))



Ressource Optimal Scheduling

Compute : (D * (C * (A + B)) + ((A + B) + (C * D))

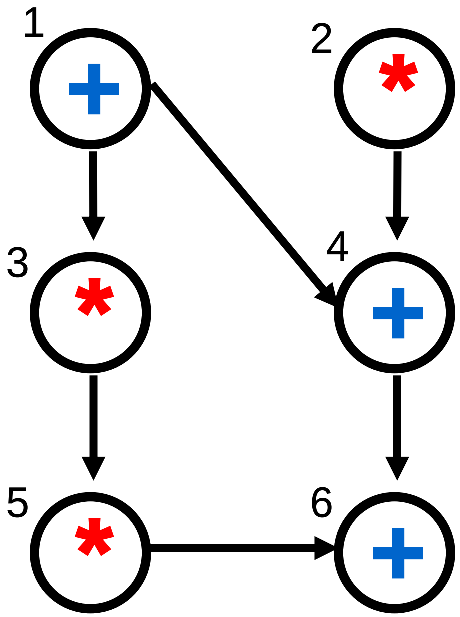
On 2 processors



But what about energy consumption ?!?!

Ressource Optimal Scheduling

Compute : (D * (C * (A + B)) + ((A + B) + (C * D))



On 2 processors

P1 (fast)

+	2 s
*	3 s

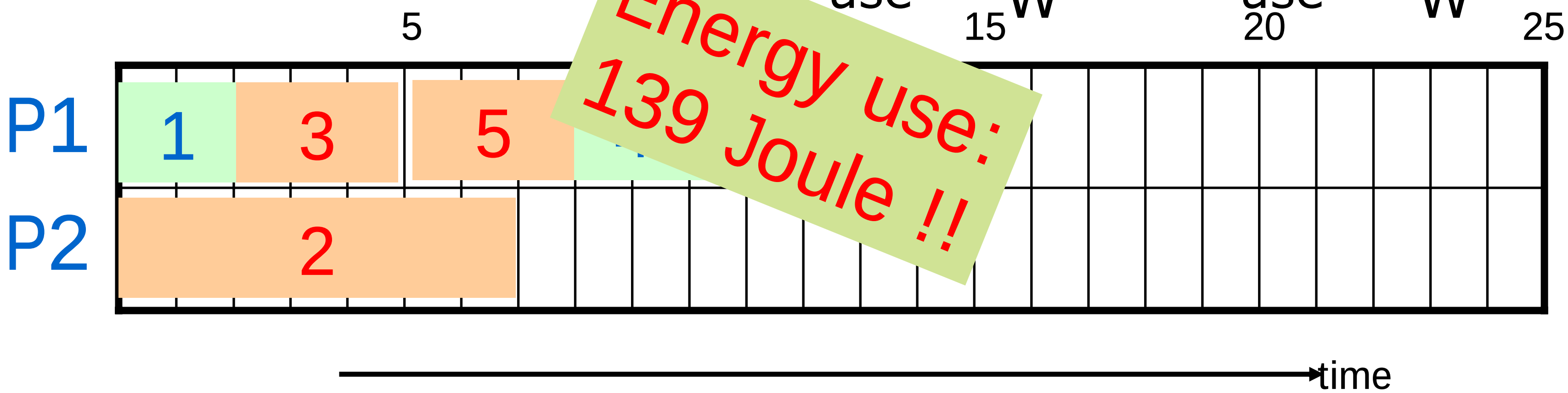
Idle	1 W
In use	9 W

P2 (slow)

+	5 s
*	7 s

Idle	2 W
In use	5 W

Energy :

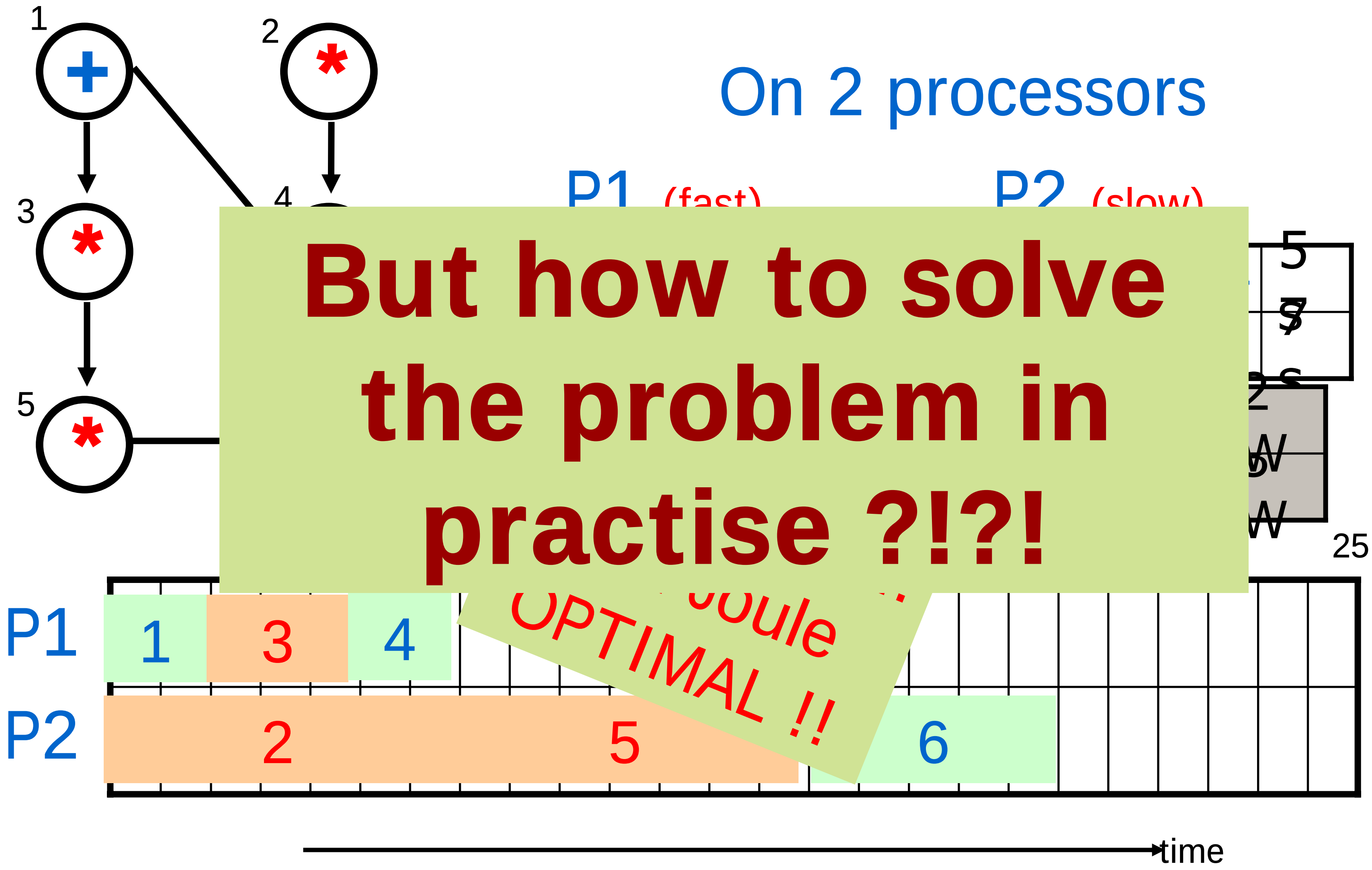


Energy use: 139 Joule !!

Ressource Optimal Scheduling

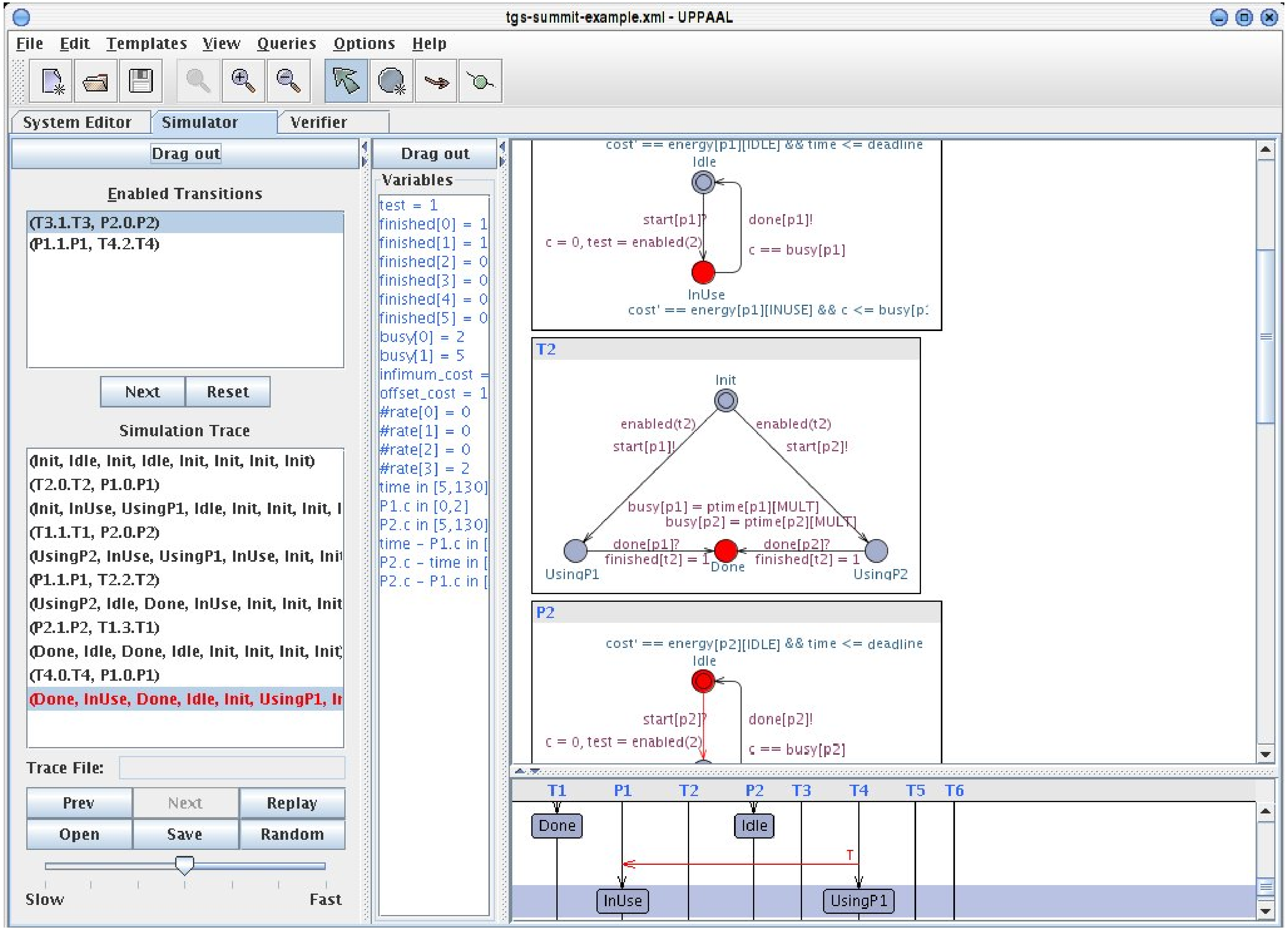
Compute : (D * (C * (A + B)) + ((A + B) + (C * D))

On 2 processors

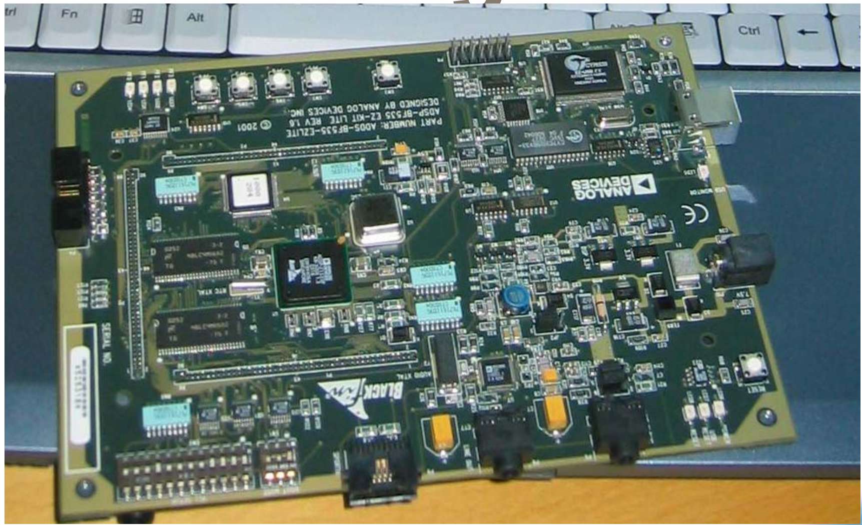


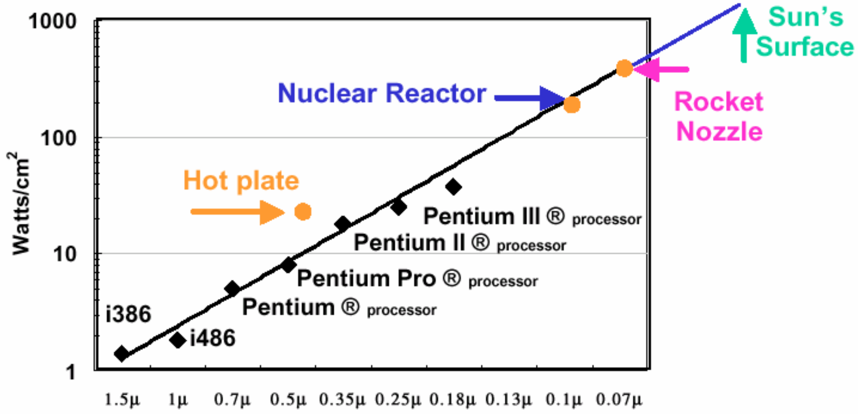
But how to solve the problem in practise ?!?!

Use UPPAAL Cora



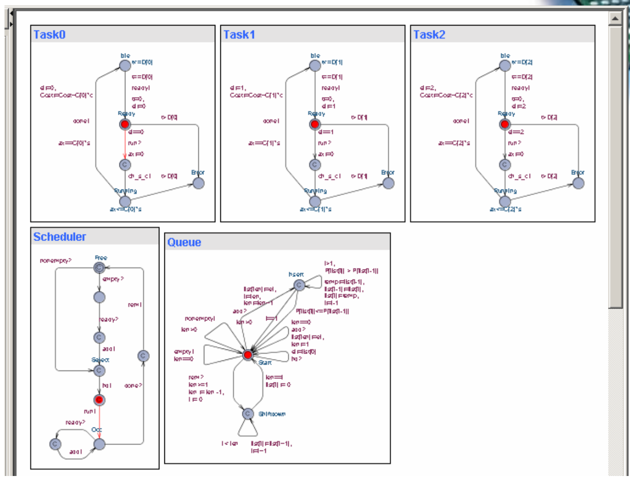
Application: *Dynamic Voltage Scaling*

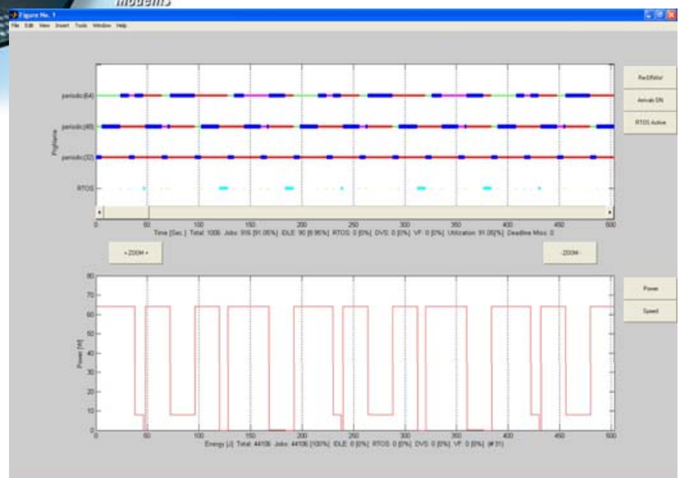


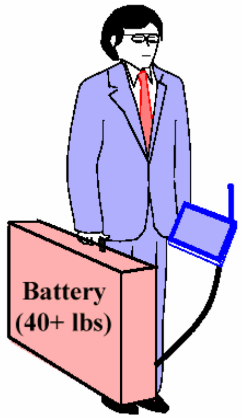


UPPAAL Cora

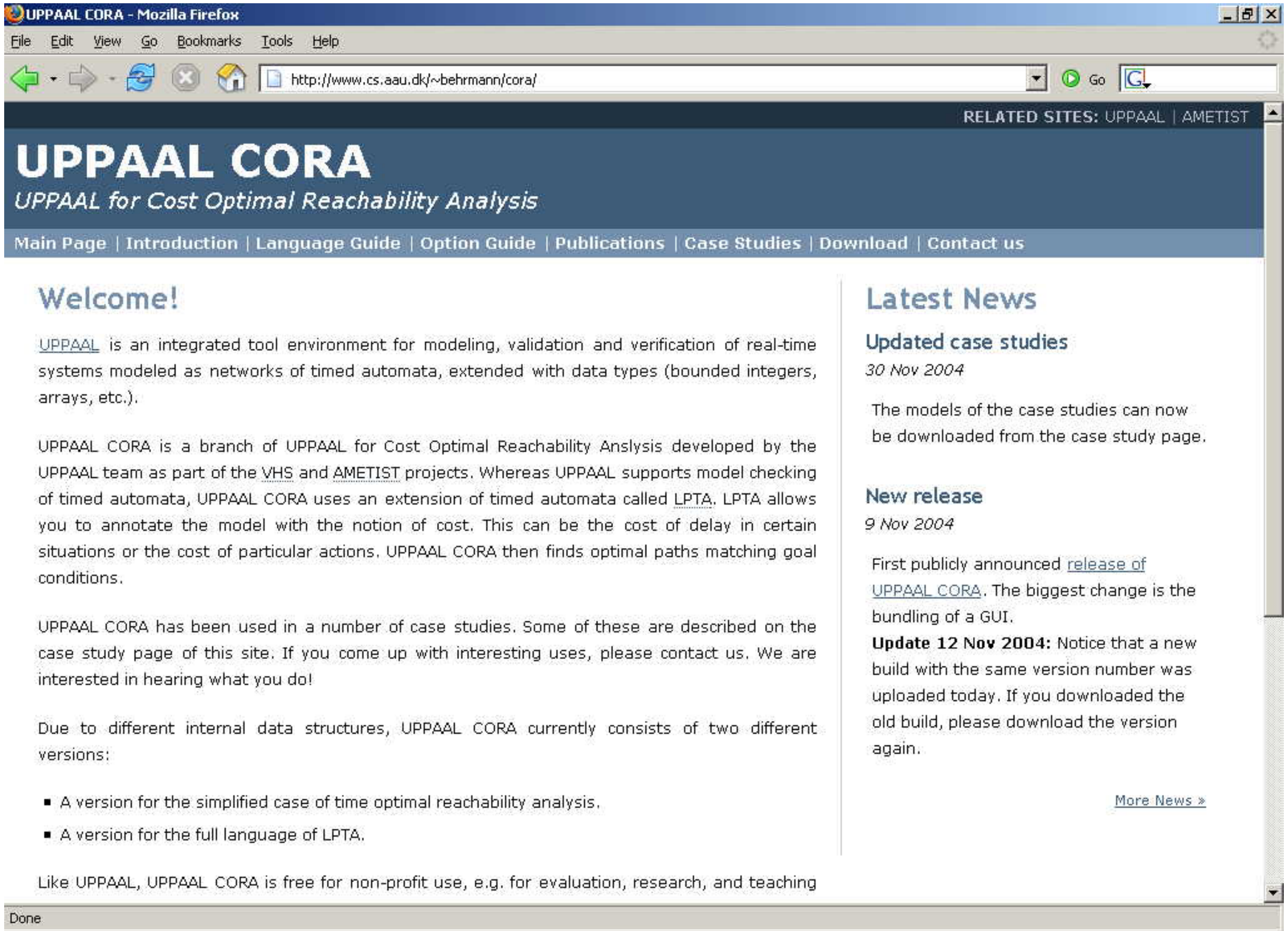
Energy Optimal Schedule







More Information



www.cs.aau.dk/~behrmann/cora