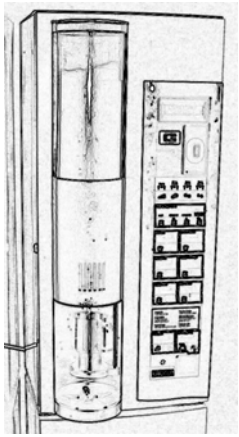
Quantitative Verification and Synthesis, of Embedded Systems

Kim G. Larsen
CISS – Aalborg University
DENMARK

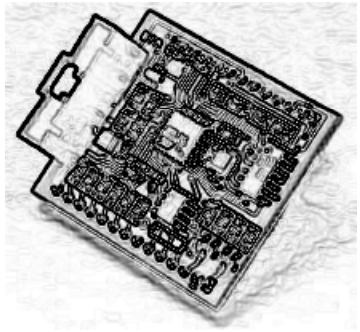
Embedded Systems

Plant
Continuous



sensors →

← actuators



Controller Program
Discrete

Eg.:

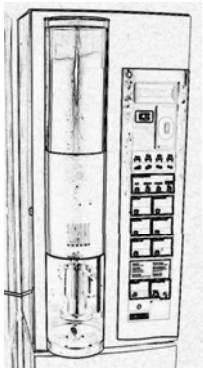
Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines

Quantities:

- timing
- energy
- memory
- bandwidth
- uncertainties

Verification

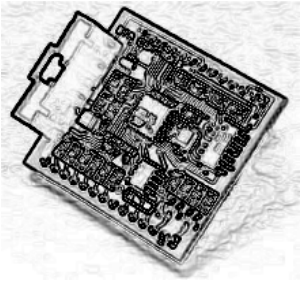
Plant
Continuous



sensors →

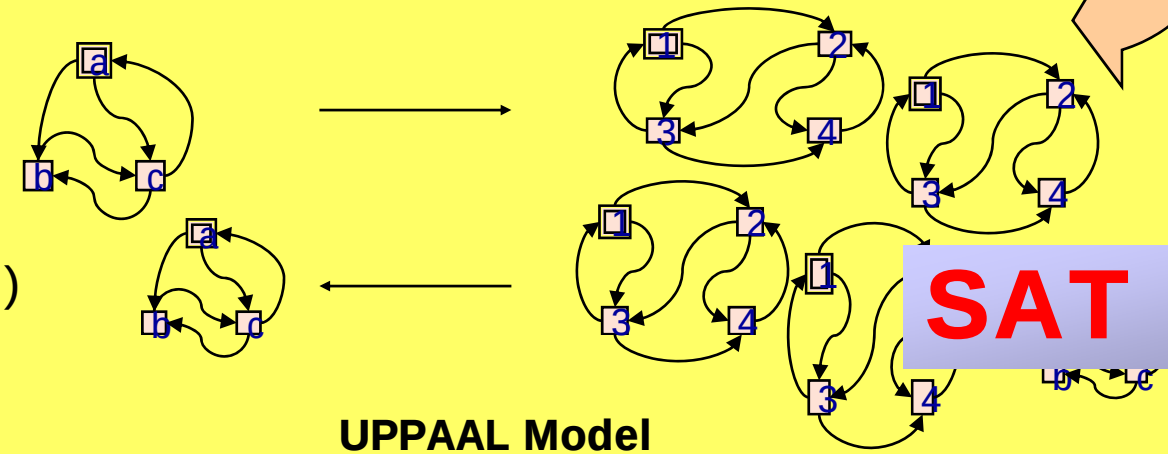
← actuators

Controller Program
Discrete



Model
of
tasks
(automatic?

Model
of
environment
(user-supplied /
non-determinism)

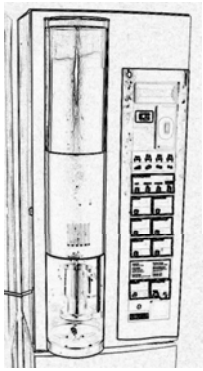


UPPAAL Model

SAT ϕ ??

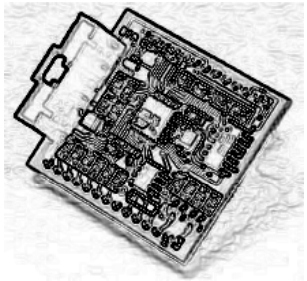
Synthesis

Plant
Continuous



sensors →

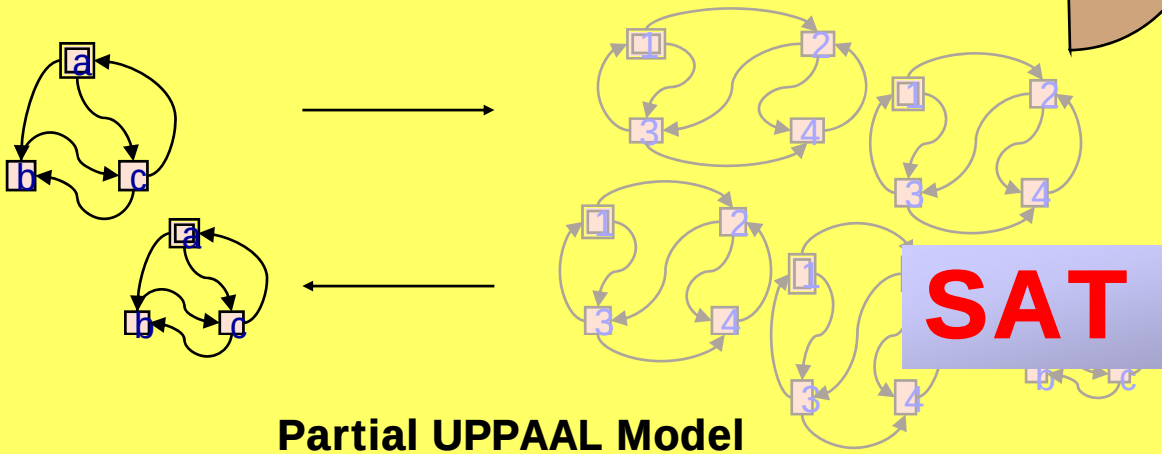
← actuators



Controller Program
Discrete

Synthesis
of
tasks
(automatic)

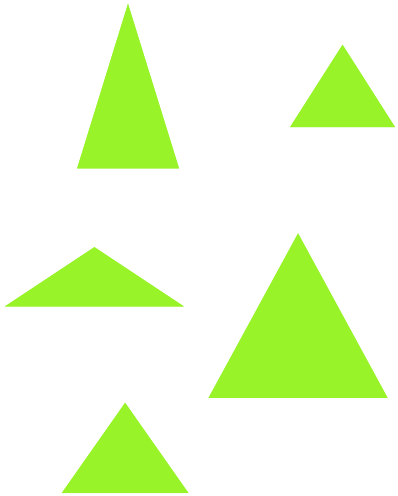
Mode
of
environment
(user-supplied)



SAT ϕ !!

Partial UPPAAL Model

Embedded Systems → Scheduling



Tasks:
Computation times
Deadlines
Dependencies
Arrival patterns
uncertainties

Scheduling Principles (OS)
EDF, FPS, RMS, DVS, ..

Resources
Execution platform
PE, Memory
Networks
Drivers
uncertainties

Approach – Timed Automata

Task

- Sch
  - Verify that given SP ensures deadlines.

- Performance Evaluation
  - Estimate resources (e.g. energy) required by given

CLASSIC

- Sc & Synthesis
  - (optimal) SP given objective.
  - *Scheduling*: SP controls

CORA

TIGA

TALK:

What can we do?

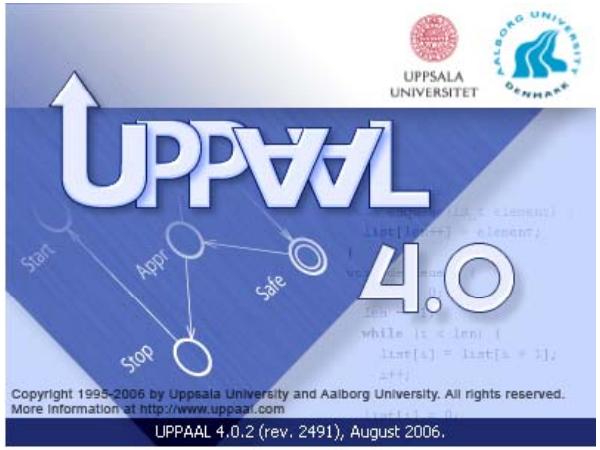
What can we do efficiently?

What would we like to do?

What can not be done?

Overview

- Scheduling
  - Timed Automata
- Optimal Scheduling
  - Priced Timed Automata
- Schedulability Analysis
  - Stop-watch Automata
  - Safety Criticial JAVA (FPGA)
- Synthesis
  - Timed Game Automata

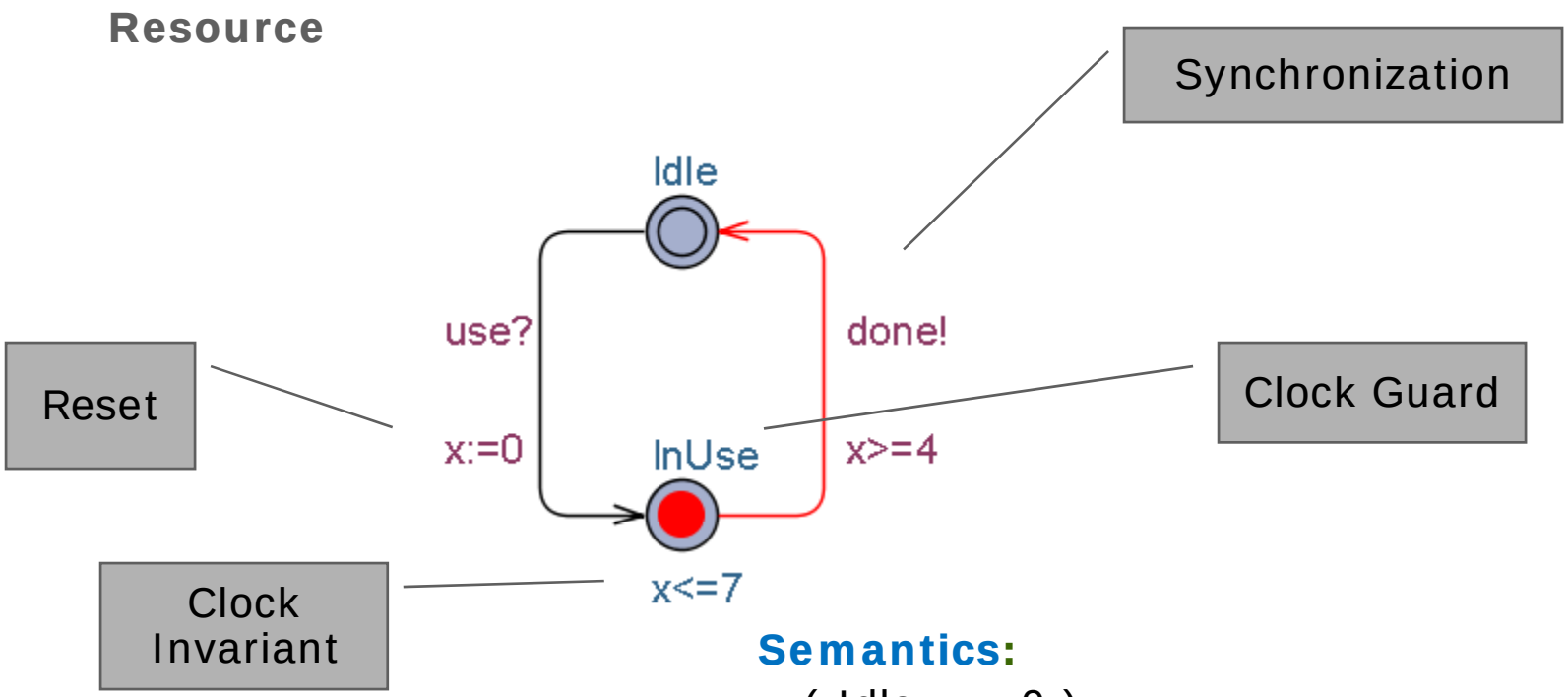


Timed Automata

UP


キャッツ株式会社 | 製品・サービス | UPPAAL - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.zipc.com/product/uppaal/index.html
Most Visited Getting Started Latest Headlines
Communication Art Technology Systems
HOME 会社地図 サイトマップ English
サイト内検索
製品・サービス サポート イベント・セミナ 企業情報 採用情報 お問い合わせ
すべての製品・サービス
ソフトウェア関連製品
ZIPC
ZIPC++
ZIPC SPLM
ZIPC Feature
ZIPC AUTOSAR
Perfect Pass
Drawrial
ASN1 Tool
XModelink
EPMツール
教育支援サービス
UPPAAL
代理店製品
取扱い代理店製品一覧
要件・構成管理ツール
HOME > 製品・サービス > UPPAAL
UPPAAL
リアルタイムシステムの安心・安全設計への処方箋!
UPPAAL日本国内で発売開始!
いまや、「リアルタイムシステムの安全性」は開発者だけでなく、高度に電子化されつつある現代社会のもっとも高い関心事のひとつです。しかし、リアルタイムでかつ複数のプロセスが同時に動作するようなシステムの安全性を確保するのは難しく、通常の方法(テスト)では非常にコストがかかります。
このツールは、「モデル検査」という新しい技術によって、そのその様な安心・安全なリアルタイム・システムの開発を支援するビジュアルな統合モデル検証ツールです。
Train(0)
Train(1)
Gate

Timed Automata [Alur & Dill'89]

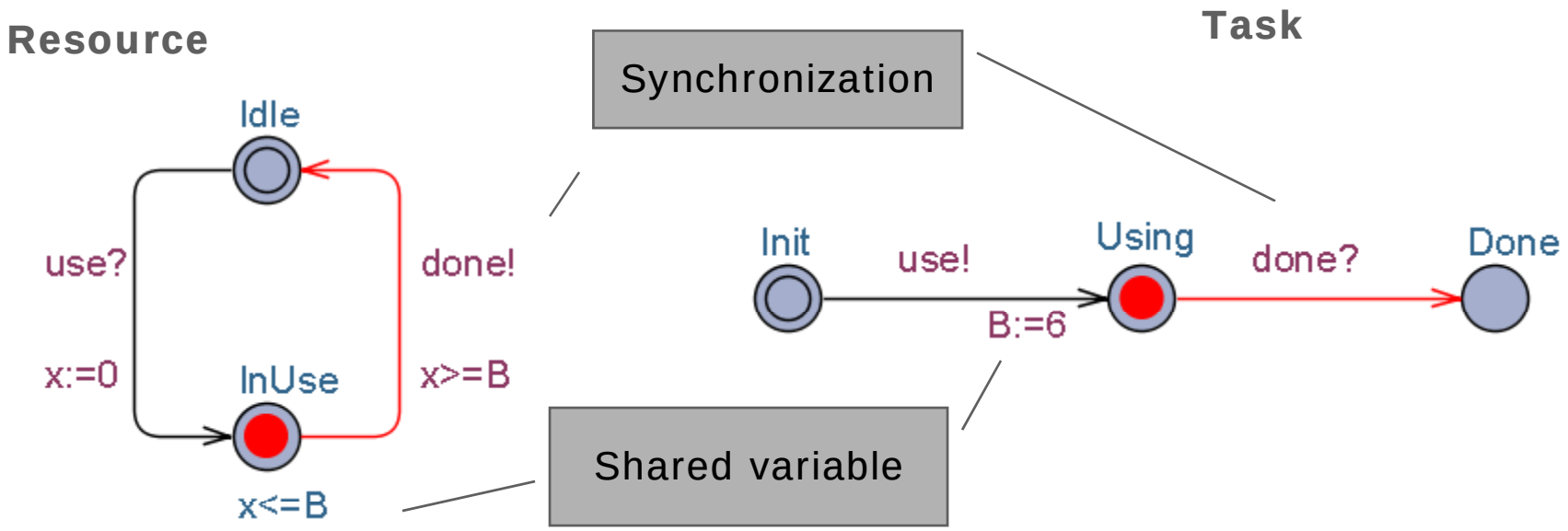


Semantics:

(Idle , x=0)

d(2.5)	→ (Idle , x=2.5)
use?	→ (InUse , x=0)
d(5)	→ (InUse , x=5)
done!	→ (Idle , x=5)

Composition



Semantics:

(Idle , Init , B=0, x=0)

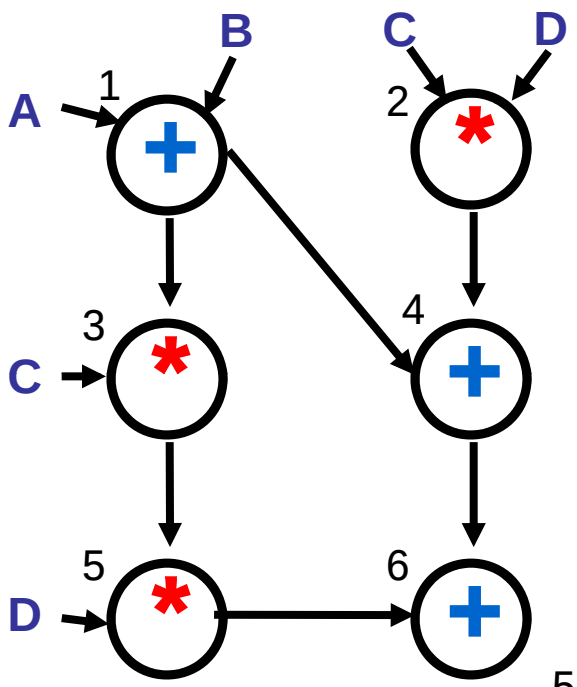
d(3.1415) → (Idle , Init , B=0 , x=3.1415)

use → (InUse , Using , B=6, x=0)

d(6) → (InUse , Using , B=6, x=6)

done → (Idle , Done , B=6 , x=6)

Task Graph Scheduling – Example



Compute :

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

4

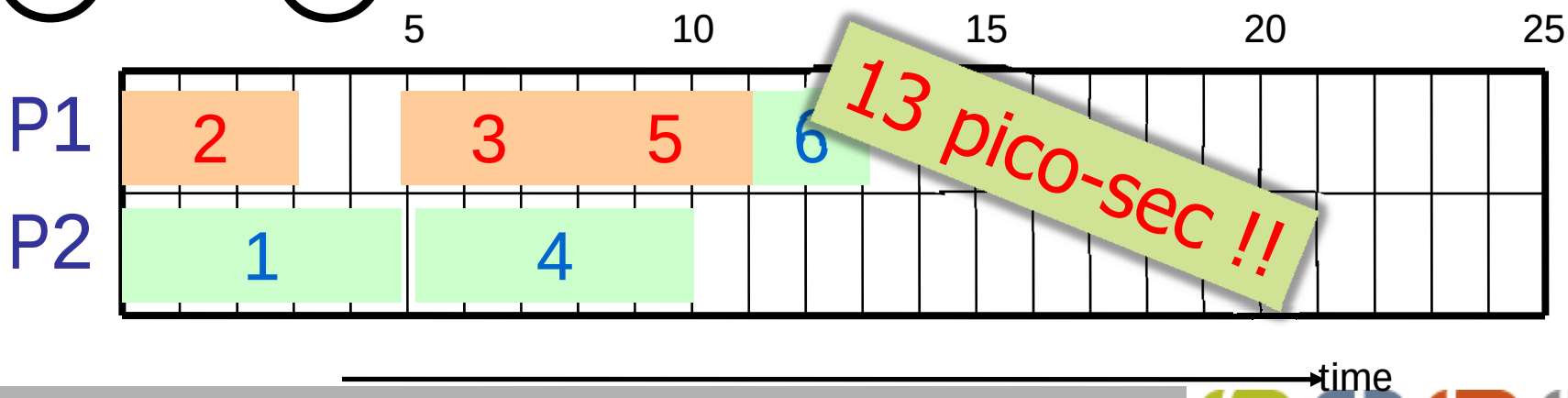
using 2 processors

P1 (fast)

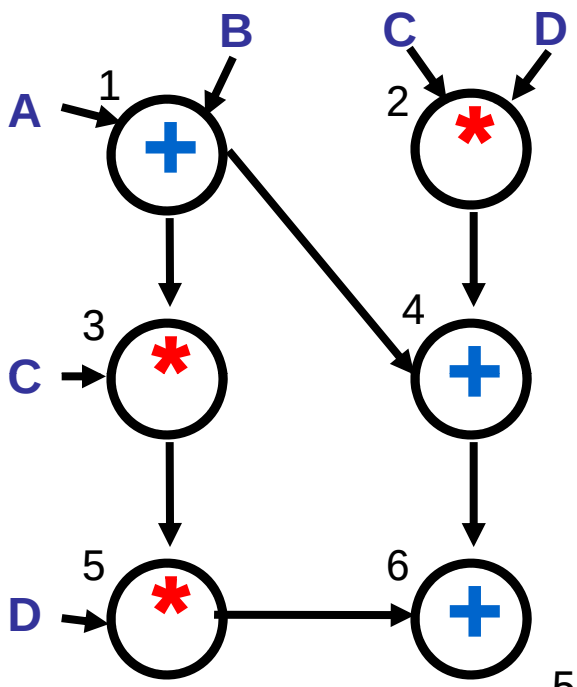
+	2ps
*	3ps

P2 (slow)

+	5ps
*	7ps



Task Graph Scheduling – Example



Compute :

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

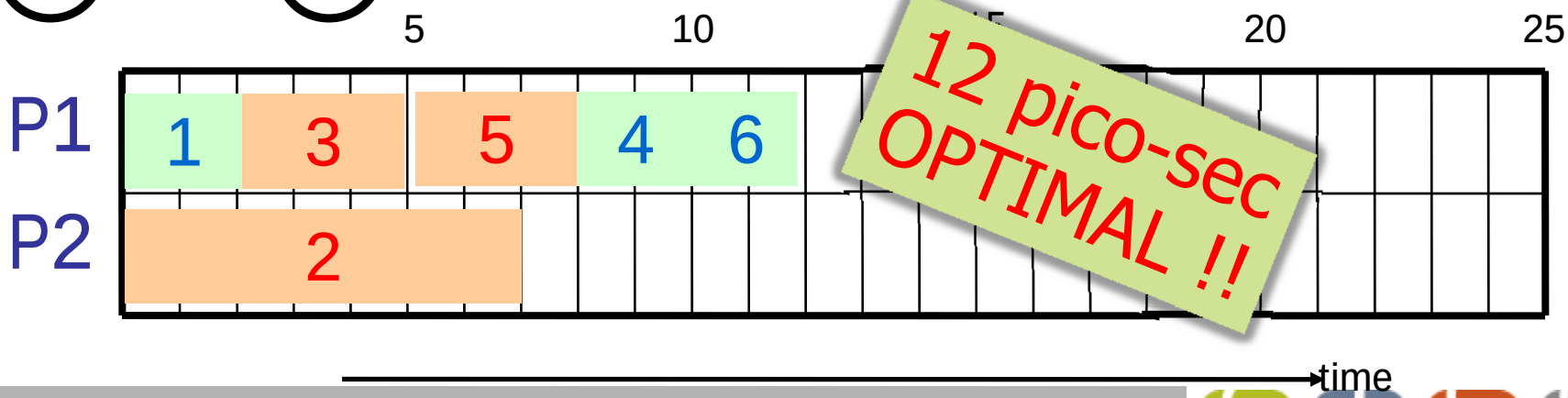
using 2 processors

P1 (fast)

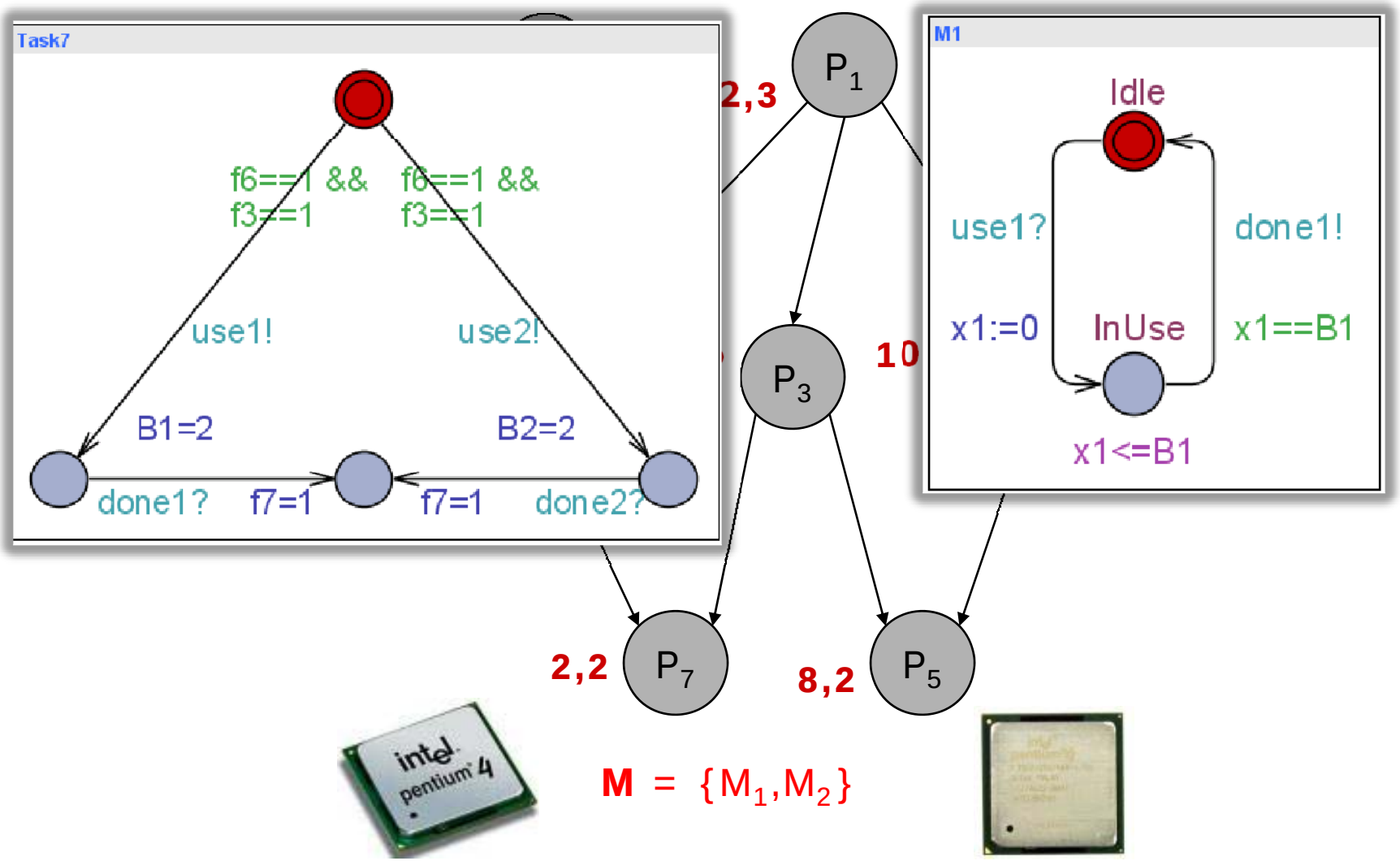
+	2ps
*	3ps

P2 (slow)

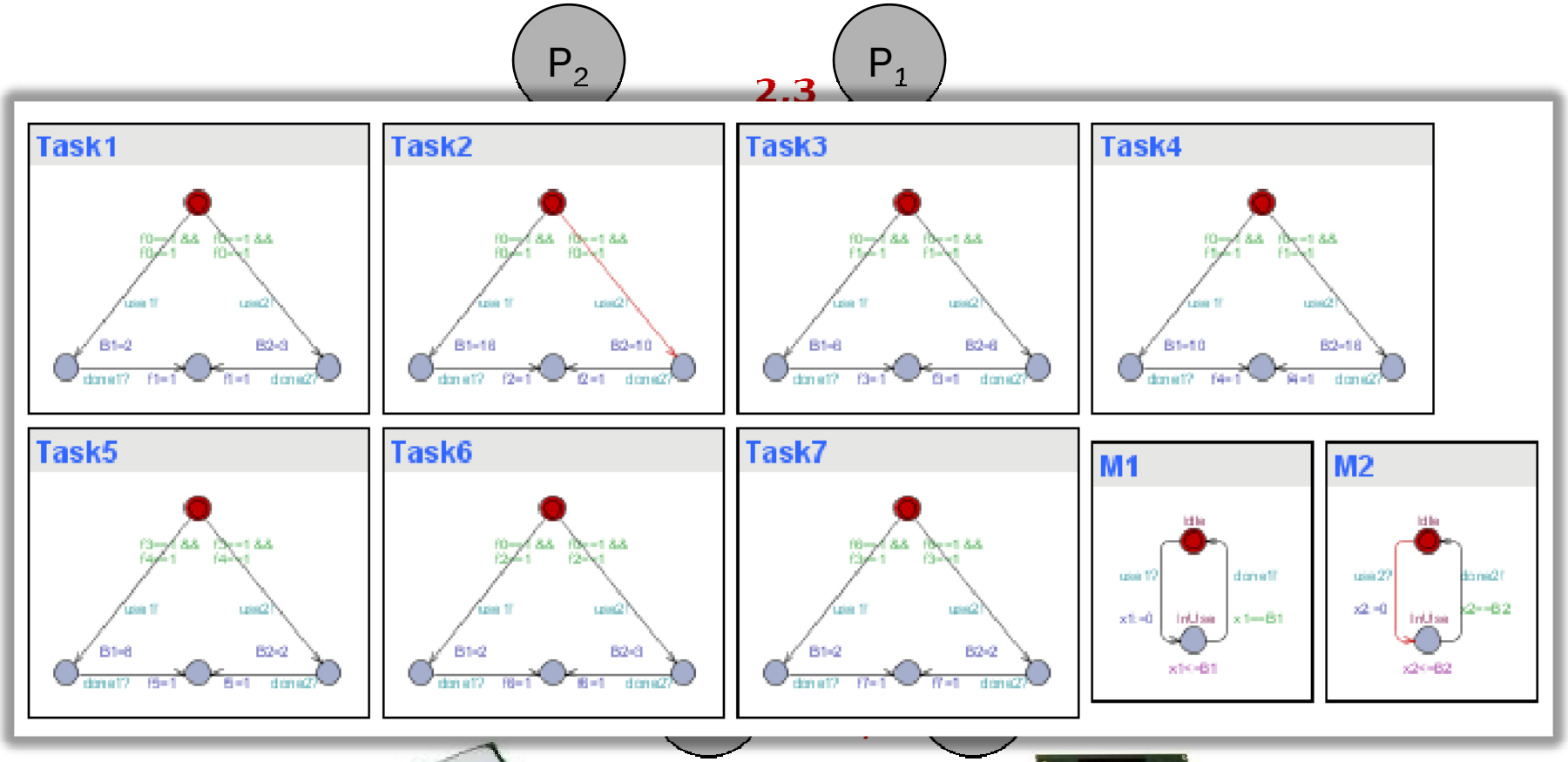
+	5ps
*	7ps



Task Graph Scheduling



Task Graph Scheduling



E<> (Task1.End and … and Task7.End)

Experimental Results

name	#tasks	#chains	# machines	optimal	TA
001	437	125	4	1178	1182
000	452	43	20	537	537
018	730	175	10	700	704
074	1007	66	12	891	894
021	1145	88	20	605	612
228	1187	293	8	1570	1574
071	1193	124	20	629	634
271	1348	127	12	1163	1164
237	1566	152	12	1340	1342
231	1664	101	16	t.o.	1137
235	1782	218	16	t.o.	1150
233	1980	207	19	1118	1121
294	2014	141	17	1257	1261
295	2168	965	18	1318	1322
292	2333	318	3	8009	8009
298	2399	303	10	2471	2473

AMETIST
advanced methods for timed systems

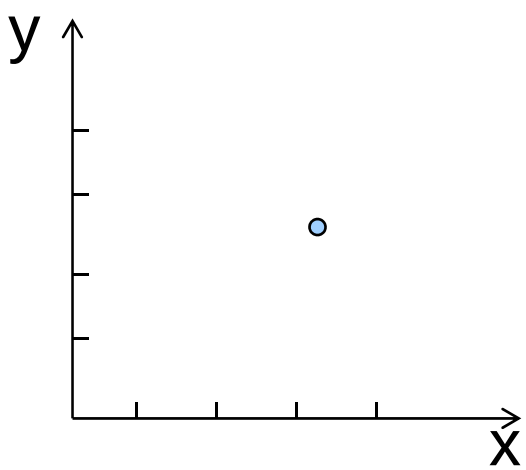
Symbolic A*
Branch-&-Bound
60 sec

Abdeddaïm, Kerbaa, Maler

Zones – From infinite to finite

State

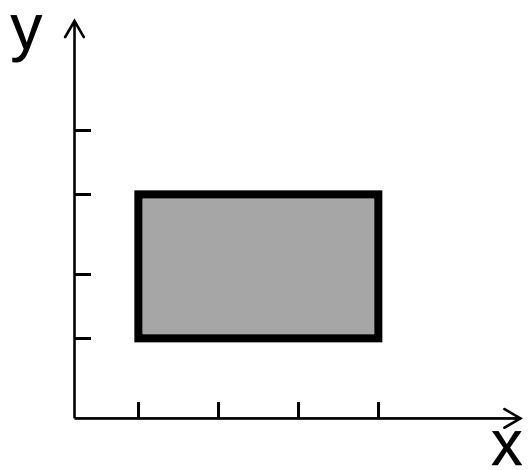
(n, x=3.2, y=2.5)



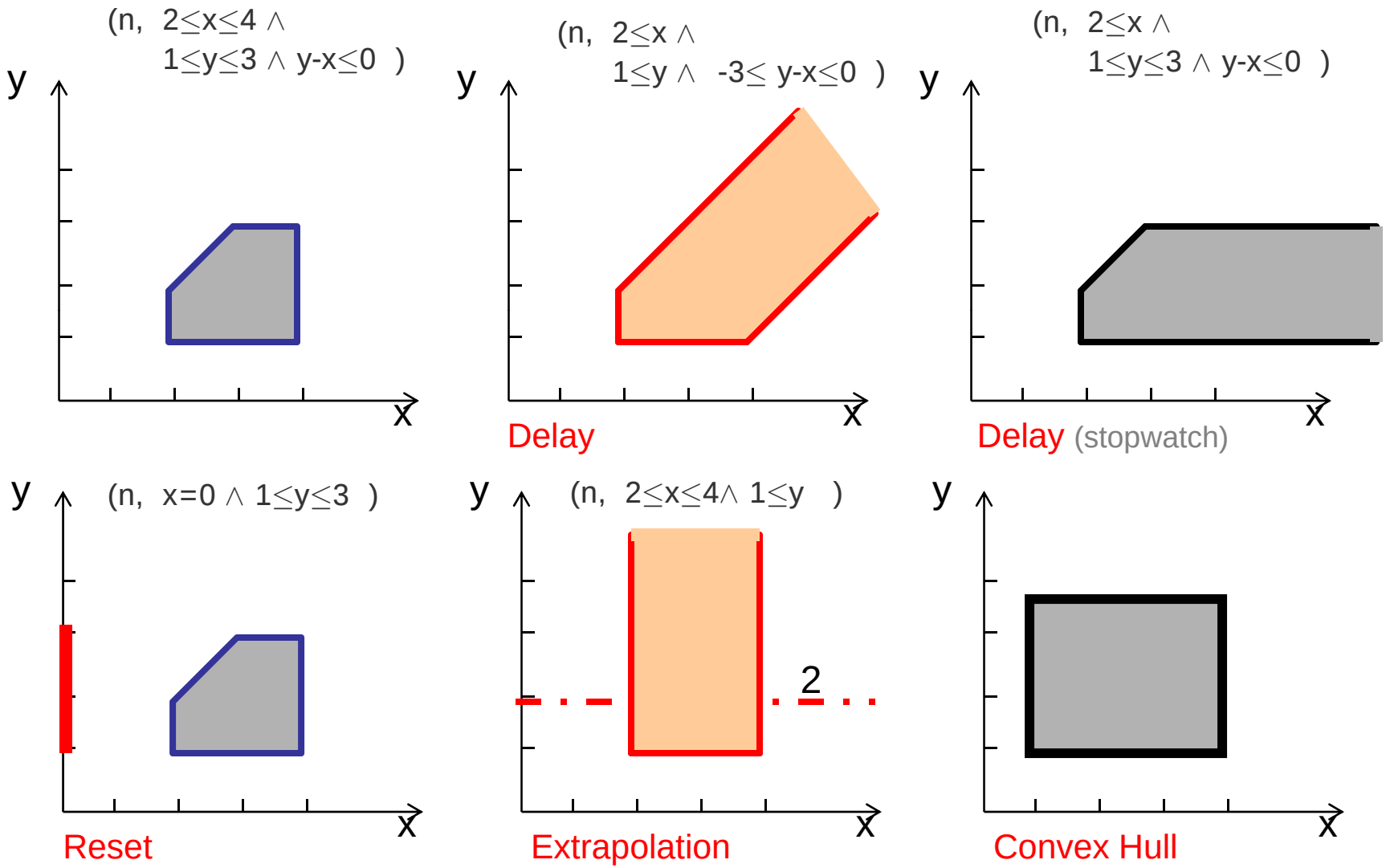
Symbolic state (set)

(n, $1 \leq x \leq 4$, $1 \leq y \leq 3$)

Zone:
conjunction of
x-y<=n, x<=>n

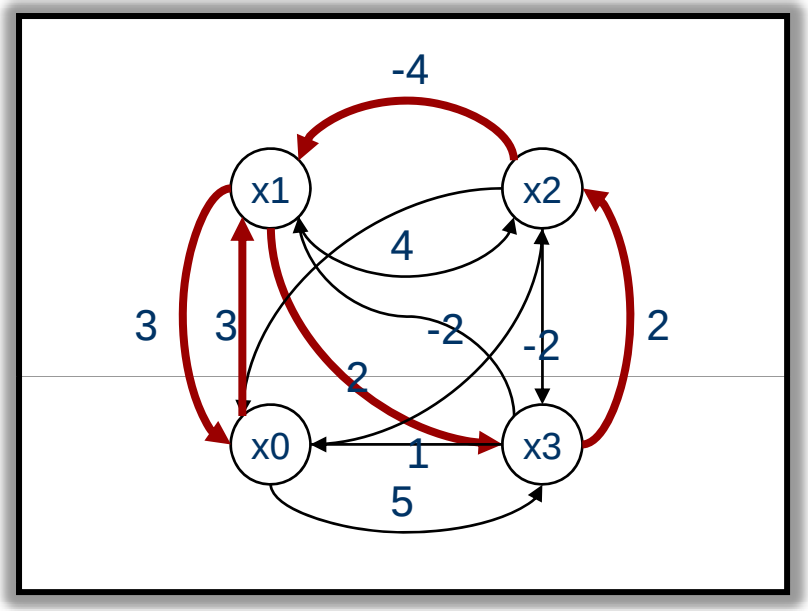


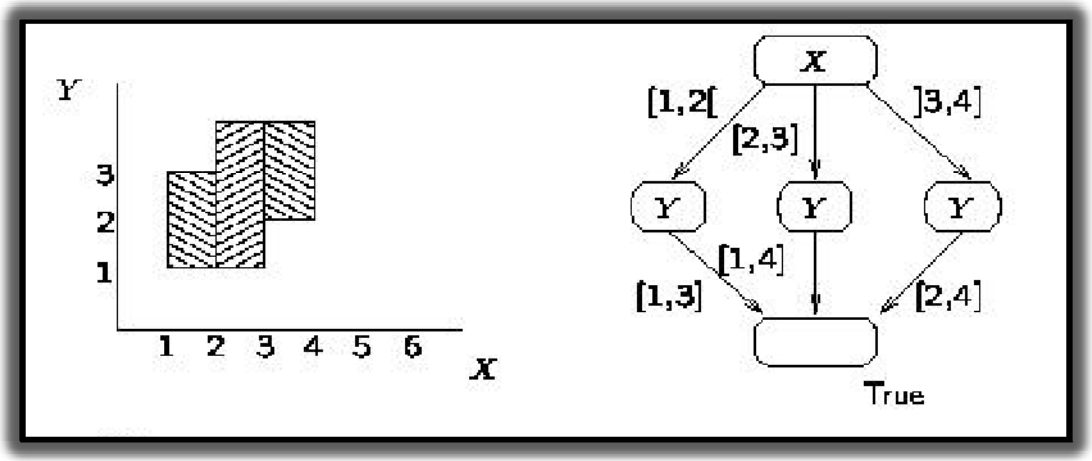
Zones – Operations



Datastructures for Zones

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]
- PW List
  [SPIN03]

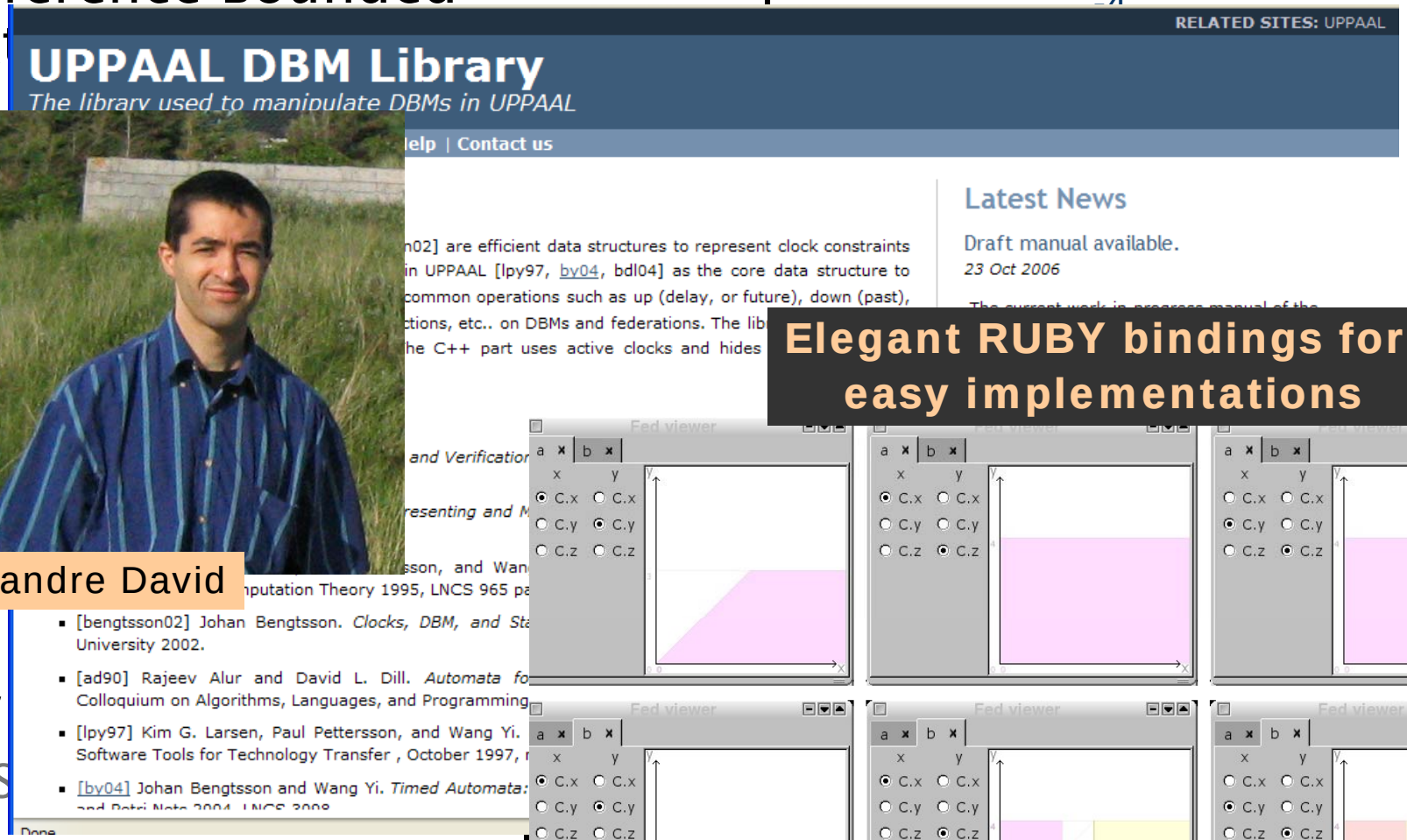




Datastructures for Zones

- Difference Bounded Mat
- M F
- C D
- PW [S

Alexandre David



Elegant RUBY bindings for easy implementations

Kim Larsen [20]

Case Studies: Controllers

- Gearbox Controller [TACAS'98]
- Bang & Olufsen Power Controller [RTPS'99,FTRTFT'2k]
- SIDMAR Steel Production Plant [RTCSA'99, DSVV'2k]
- Real-Time RCX Control-Programs [ECRTS'2k]
- Terma, Verification of Memory Management for Radar (2001)
- Scheduling Lacquer Production (2005)
- Memory Arbiter Synthesis and Verification for a Radar Memory Interface Card [NJC'05]

- Adapting the UPPAAL Model of a Distributed Lift System, 2007
- Analyzing a χ model of a turntable system using Spin, CADP and Uppaal, 2006
- Designing, Modelling and Verifying a Container Terminal System Using UPPAAL, 2008
- Model-based system analysis using Chi and Uppaal: An industrial case study, 2008

Case Studies: Protocols

- Philips Audio Protocol [HS'95, CAV'95, RTSS'95, CAV'96]
- Bounded Retransmission Protocol [TACAS'97]
- Bang & Olufsen Audio/Video Protocol [RTSS'97]
- TDMA Protocol [PRFTS'97]
- Lip-Synchronization Protocol [FMICS'97]
- ATM ABR Protocol [CAV'99]
- ABB Fieldbus Protocol [ECRTS'2k]
- IEEE 1394 Firewire Root Contention (2000)
- Distributed Agreement Protocol [Formats05]
- Leader Election for Mobile Ad Hoc Networks [Charme05]

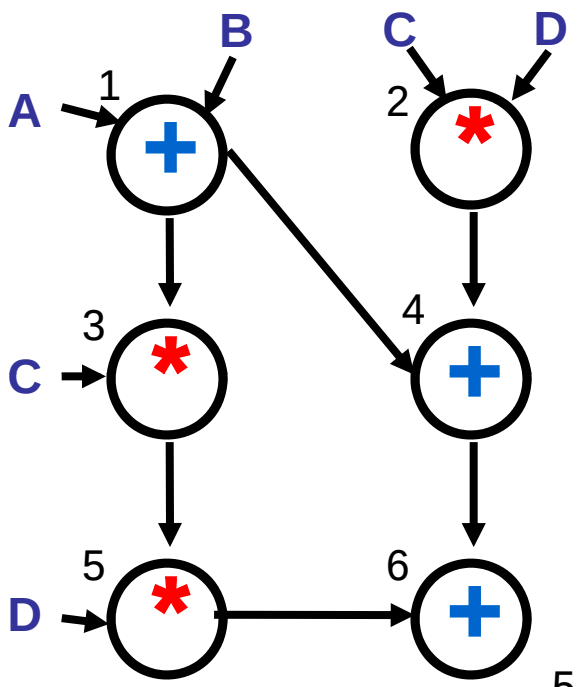
- Analysis of a protocol for dynamic configuration of IPv4 link local addresses using Uppaal, 2006
- Formalizing SHIM6, a Proposed Internet Standard in UPPAAL, 2007
- Verifying the distributed real-time network protocol RTnet using Uppaal, 2007
- Analysis of the Zeroconf protocol using UPPAAL, 2009
- Analysis of a Clock Synchronization Protocol for Wireless Sensor Networks, 2009

Using UPPAAL as Back-end

- Vooduu: verification of object-oriented designs using Uppaal, 2004
- Moby/RT: A Tool for Specification and Verification of Real-Time Systems, 2000
- Formalising the ARTS MPSOC Model in UPPAAL, 2007
- Timed automata translator for Uppaal to PVS
- Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT, 2008
- Verification of COMDES-II Systems Using UPPAAL with Model Transformation, 2008

Priced Timed Automata

Task Graph Scheduling – Revisited



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

using 2 processors

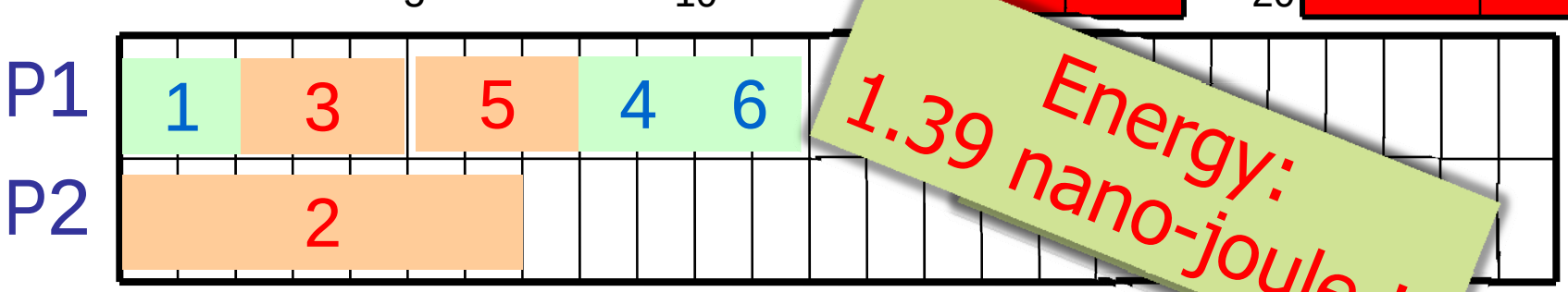
P1 (fast)

+	2ps
*	3ps
Idle	10W
In use	90W

P2 (slow)

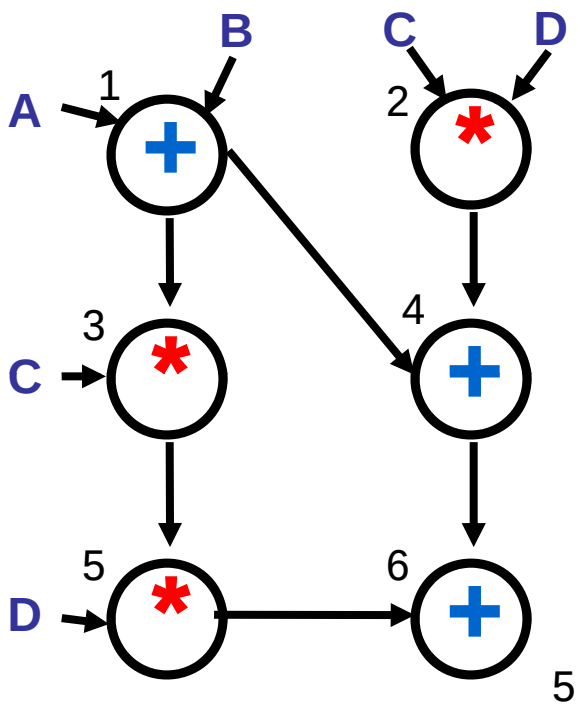
+	5ps
*	7ps
Idle	20W
In use	30W

ENERGY:



Energy: 1.39 nano-joule !!

Task Graph Scheduling – Revisited



Compute :

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

using 2 processors

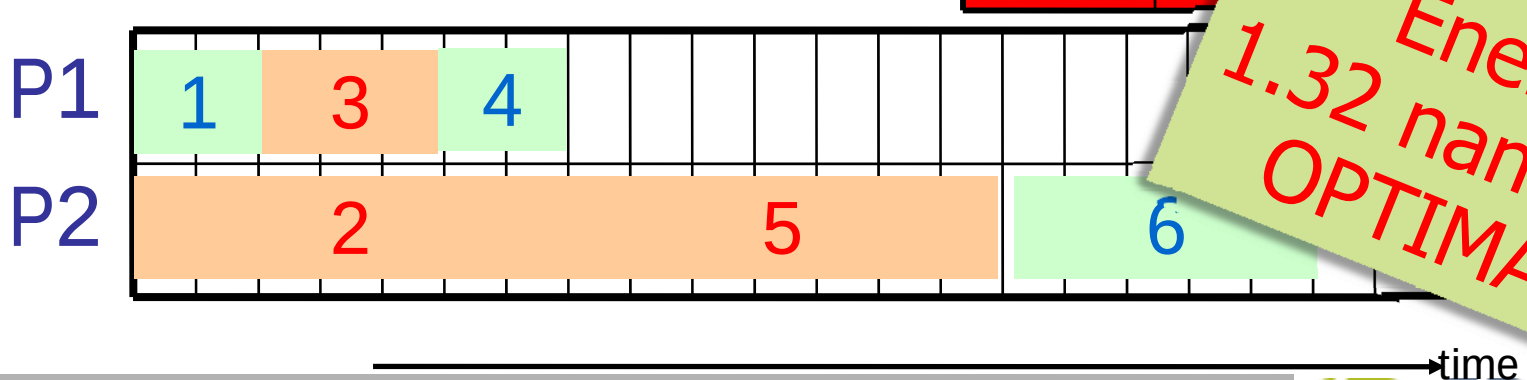
P1 (fast)

+	2ps
*	3ps
Idle	10W
In use	90W

P2 (slow)

+	5ps
*	7ps
Idle	20W
In use	30W

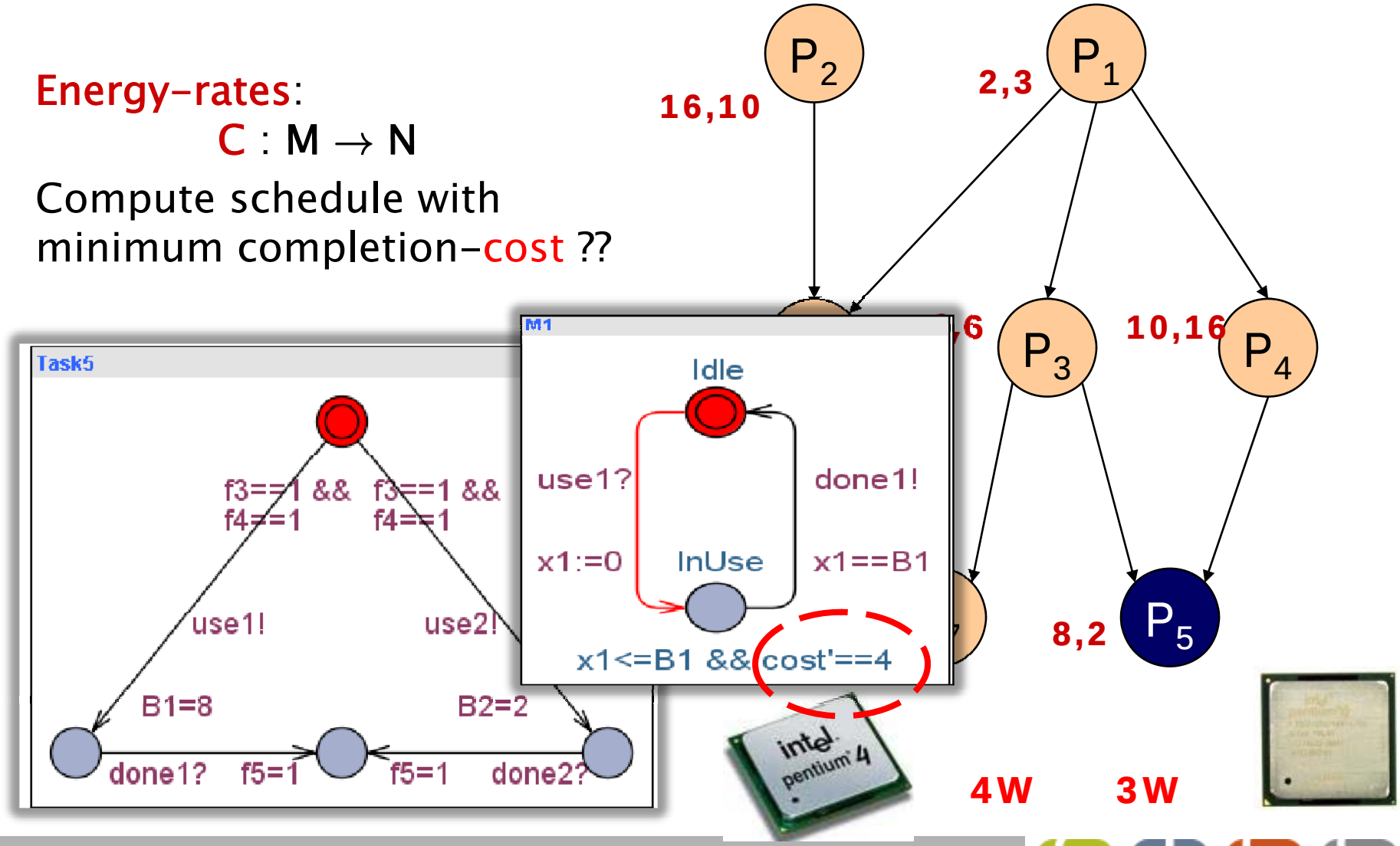
ENERGY:



Energy:
1.32 nano-joule
OPTIMAL !!

Optimal Task Graph Scheduling

- Energy-rates:
  $C : M \rightarrow N$
- Compute schedule with minimum completion-cost ??

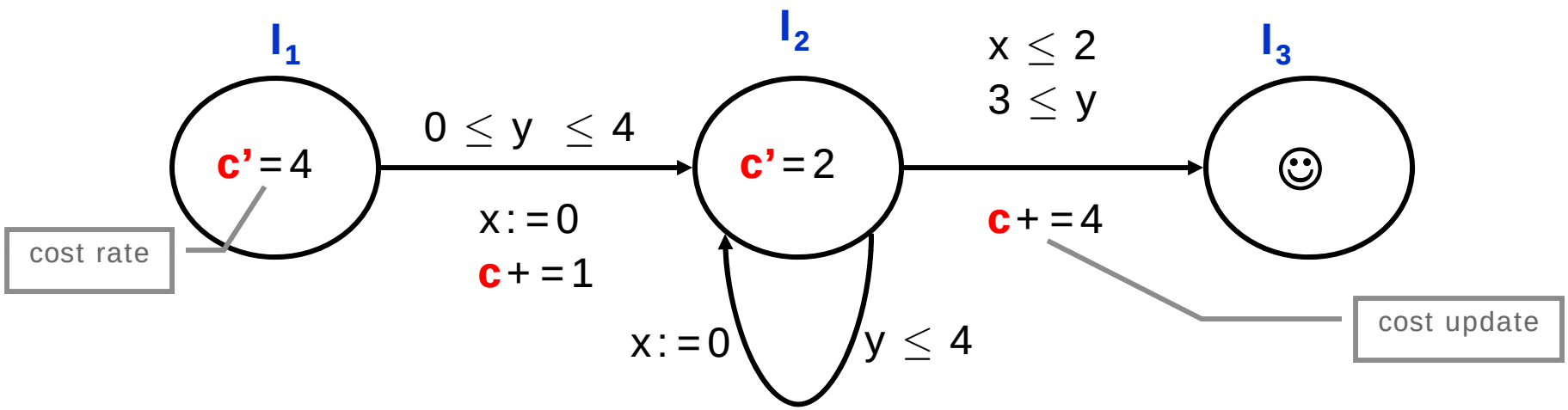


Priced Timed Automata

Behrmann, Fehnker, et all (HSCC'01)

Alur, Torre, Pappas (HSCC'01)

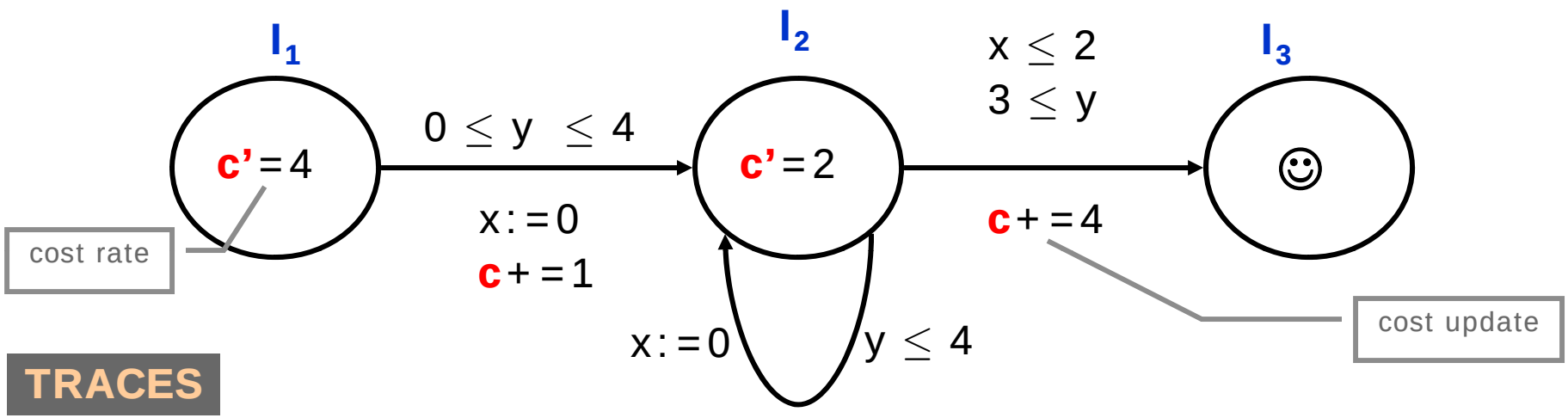
Timed Automata + **COST** variable



Priced Timed Automata

Behrmann, Fehnker, et all (HSCC'01)
Alur, Torre, Pappas (HSCC'01)

Timed Automata + **COST** variable



TRACES

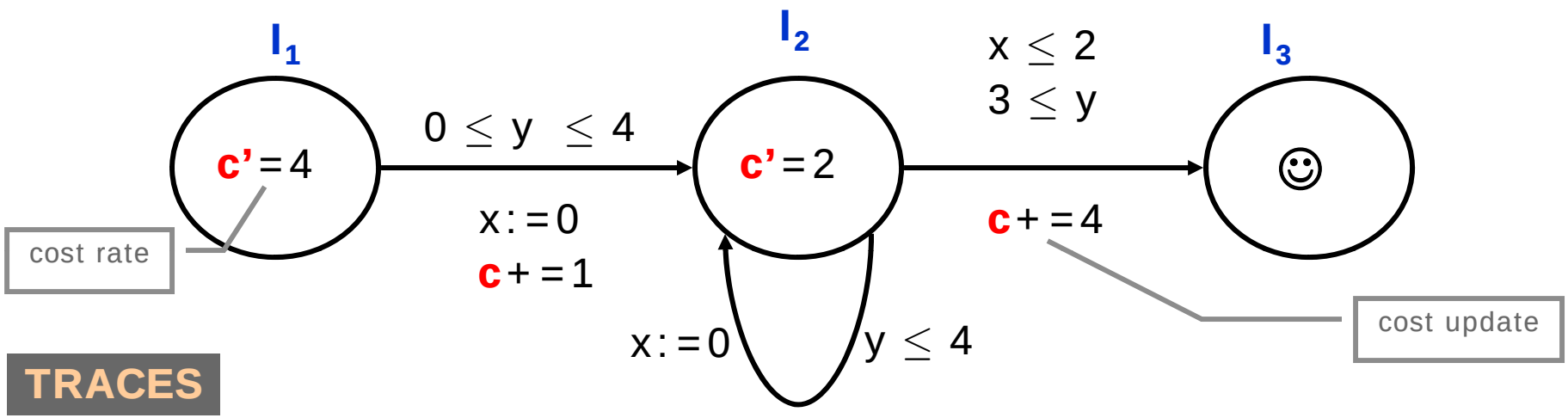
$$(l_1, x=y=0) \xrightarrow[12]{\varepsilon(3)} (l_1, x=y=3) \xrightarrow[1]{} (l_2, x=0, y=3) \xrightarrow[4]{} (l_3, \_, \_)$$

$\sum c = 17$

Priced Timed Automata

Behrmann, Fehnker, et all (HSCC'01)
Alur, Torre, Pappas (HSCC'01)

Timed Automata + **COST** variable



TRACES

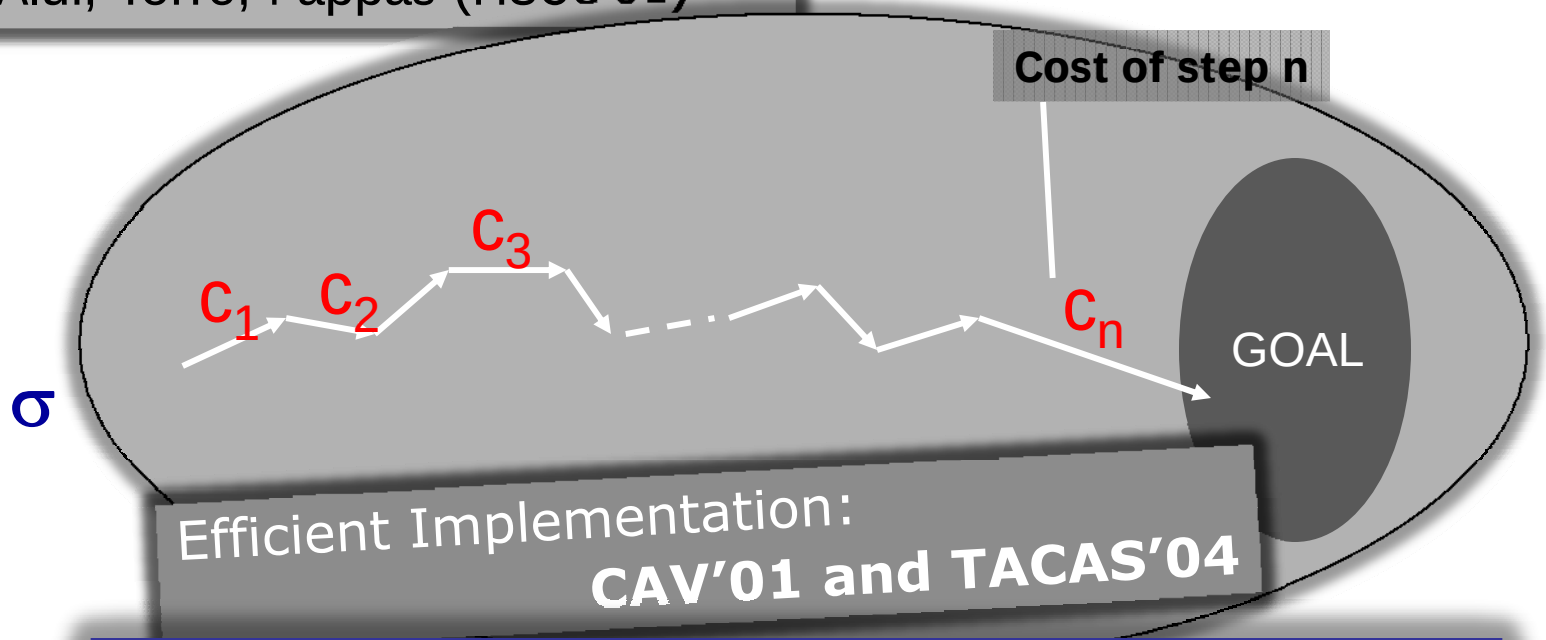
$(l_1, x=y=0) \xrightarrow[12]{\varepsilon(3)} (l_1, x=y=3) \xrightarrow[1]{} (l_2, x=0, y=3) \xrightarrow[4]{} (l_3, \_, \_)$ $\quad \sum c = 17$

$(l_1, x=y=0) \xrightarrow[10]{\varepsilon(2.5)} (l_1, x=y=2.5) \xrightarrow[1]{} (l_2, x=0, y=2.5) \xrightarrow[1]{\varepsilon(.5)} (l_2, x=0.5, y=3) \xrightarrow[4]{} (l_3, \_, \_)$ $\quad \sum c = 16$

$(l_1, x=y=0) \xrightarrow[1]{} (l_2, x=0, y=0) \xrightarrow[6]{\varepsilon(3)} (l_2, x=3, y=3) \xrightarrow[0]{} (l_2, x=0, y=3) \xrightarrow[4]{} (l_3, \_, \_)$ $\quad \sum c = 11$

Cost-Optimality Reachability

Behrmann, Fehnker, et all (HSCC'01)
Alur, Torre, Pappas (HSCC'01)



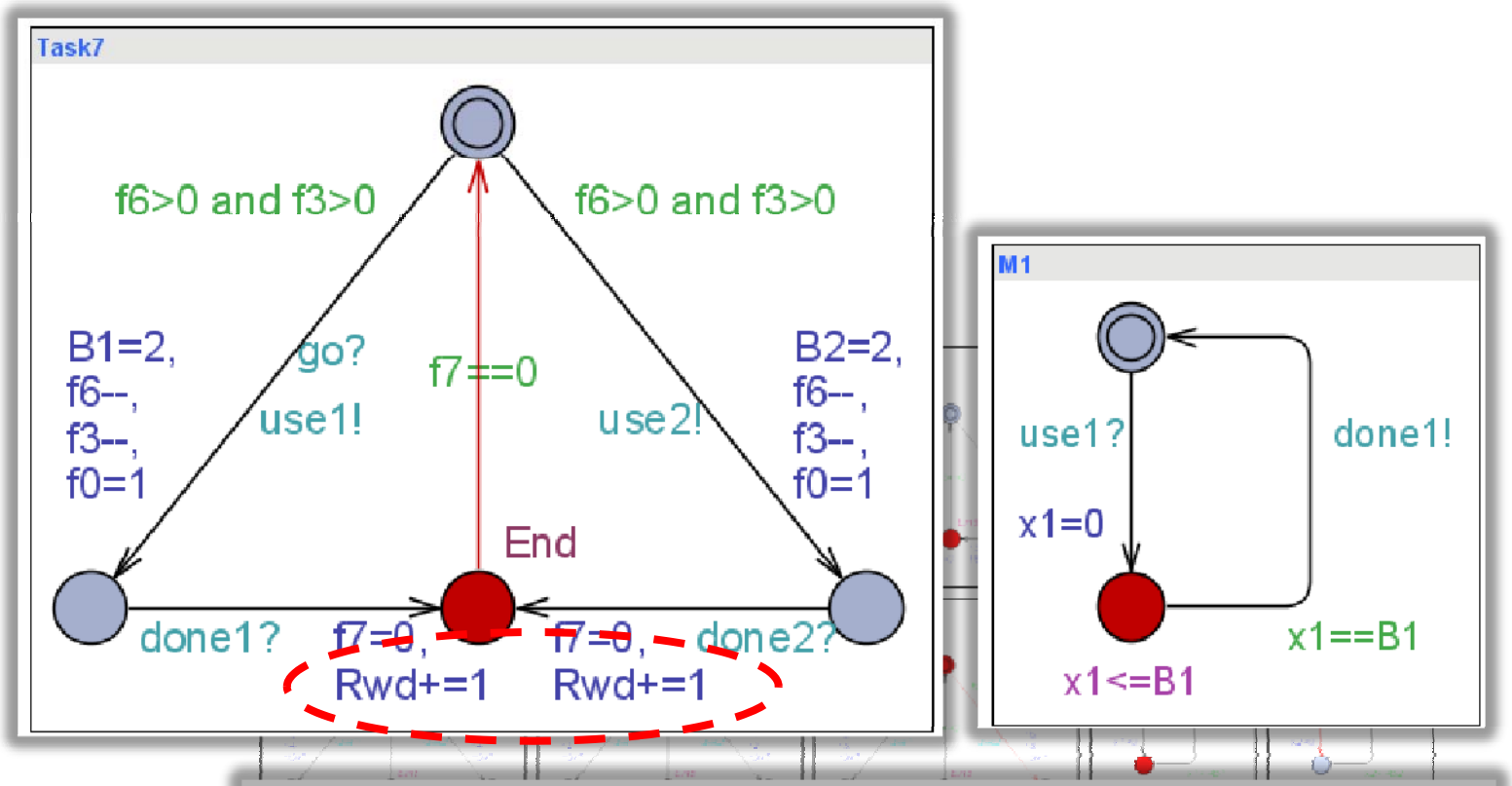
Efficient Implementation:
CAV'01 and TACAS'04

Competitive with and Complementary to MILP

Va

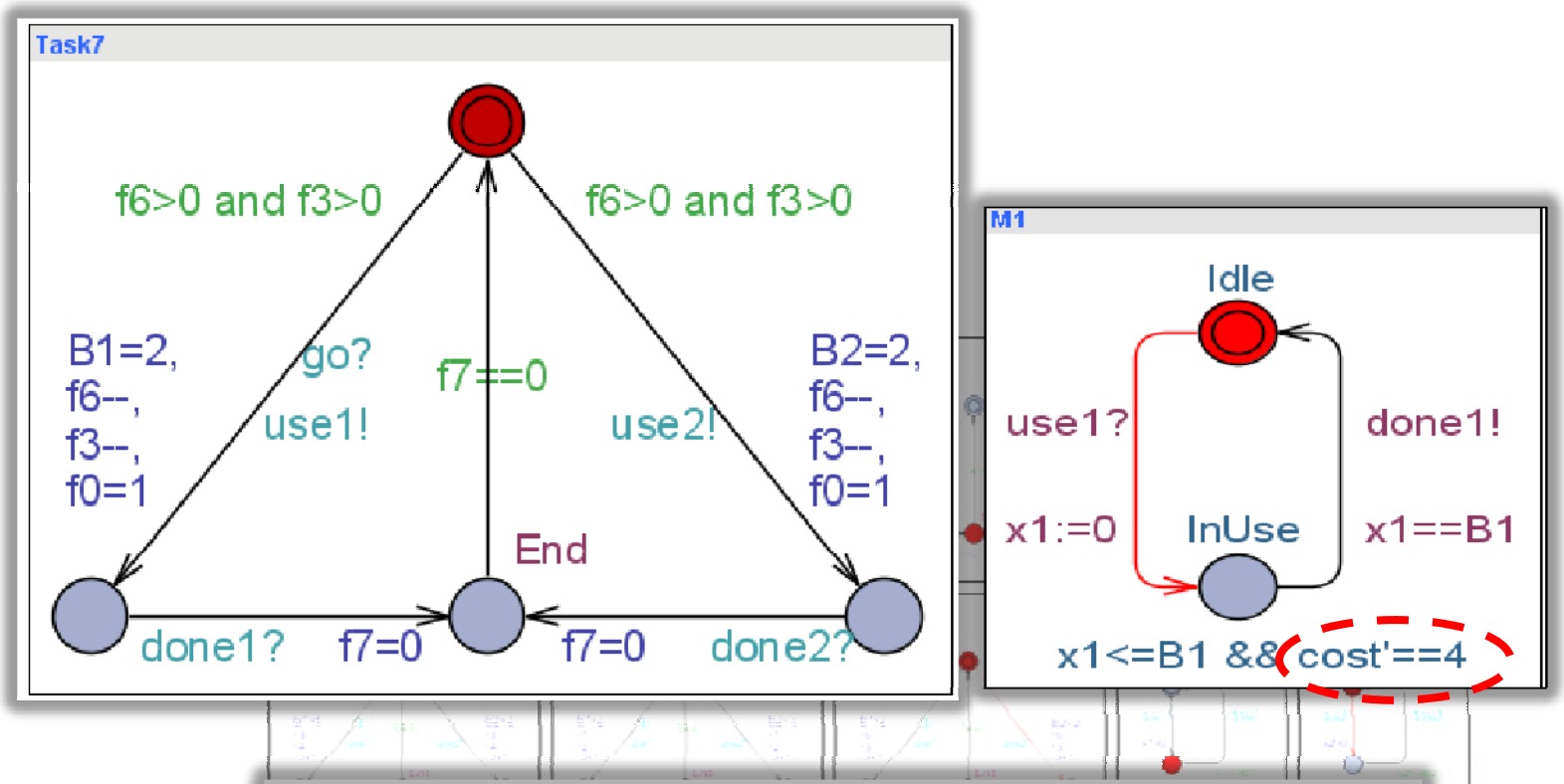
Optimal Schedule σ^* : $val(\sigma^*) = \inf_{\sigma} val(\sigma)$

Optimal Infinite Scheduling



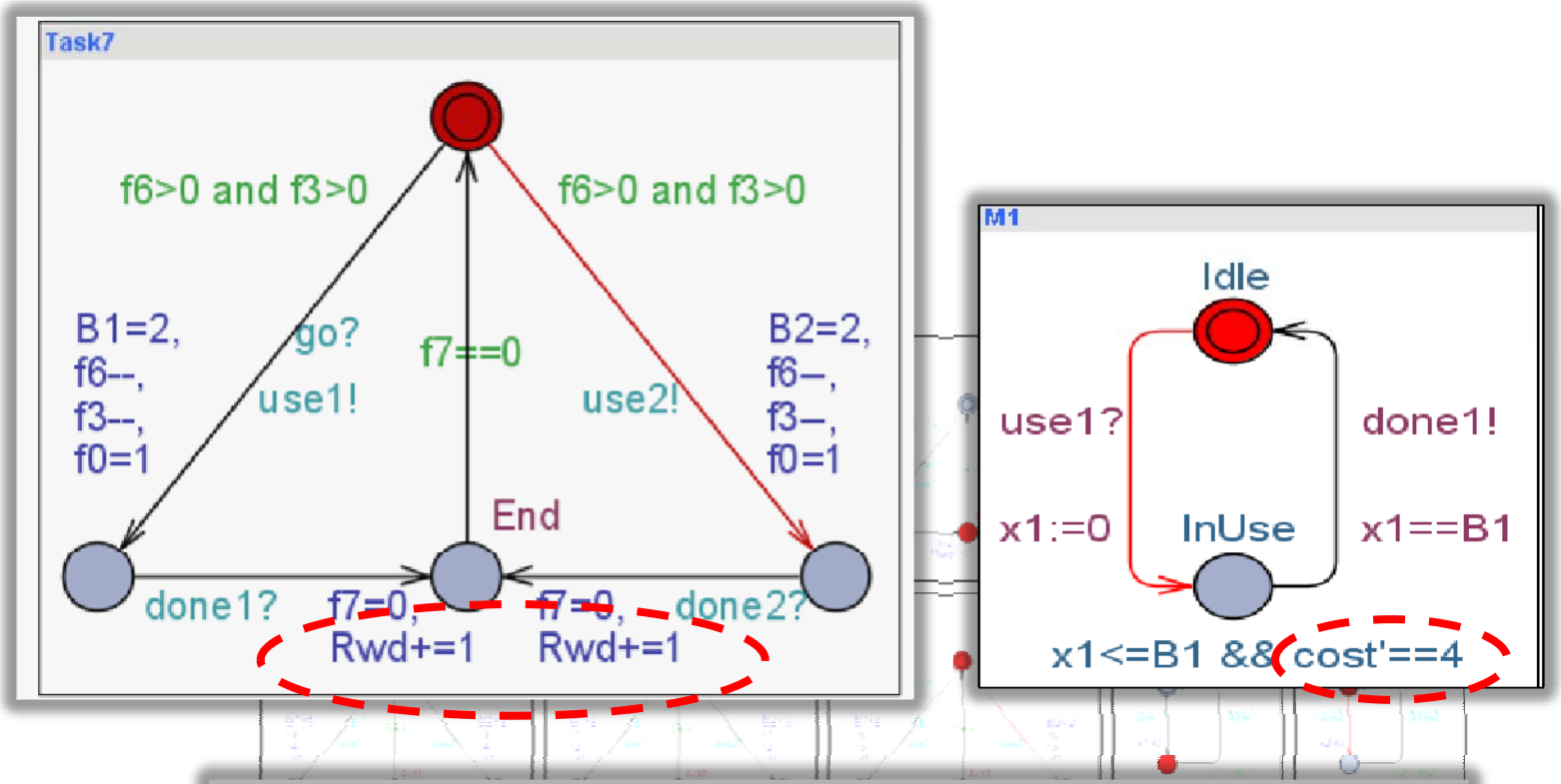
Maximize throughput:
i.e. maximize Reward / Time in the long run!

Optimal Infinite Scheduling



Minimize Energy Consumption:
i.e. minimize Cost / Time in the long run

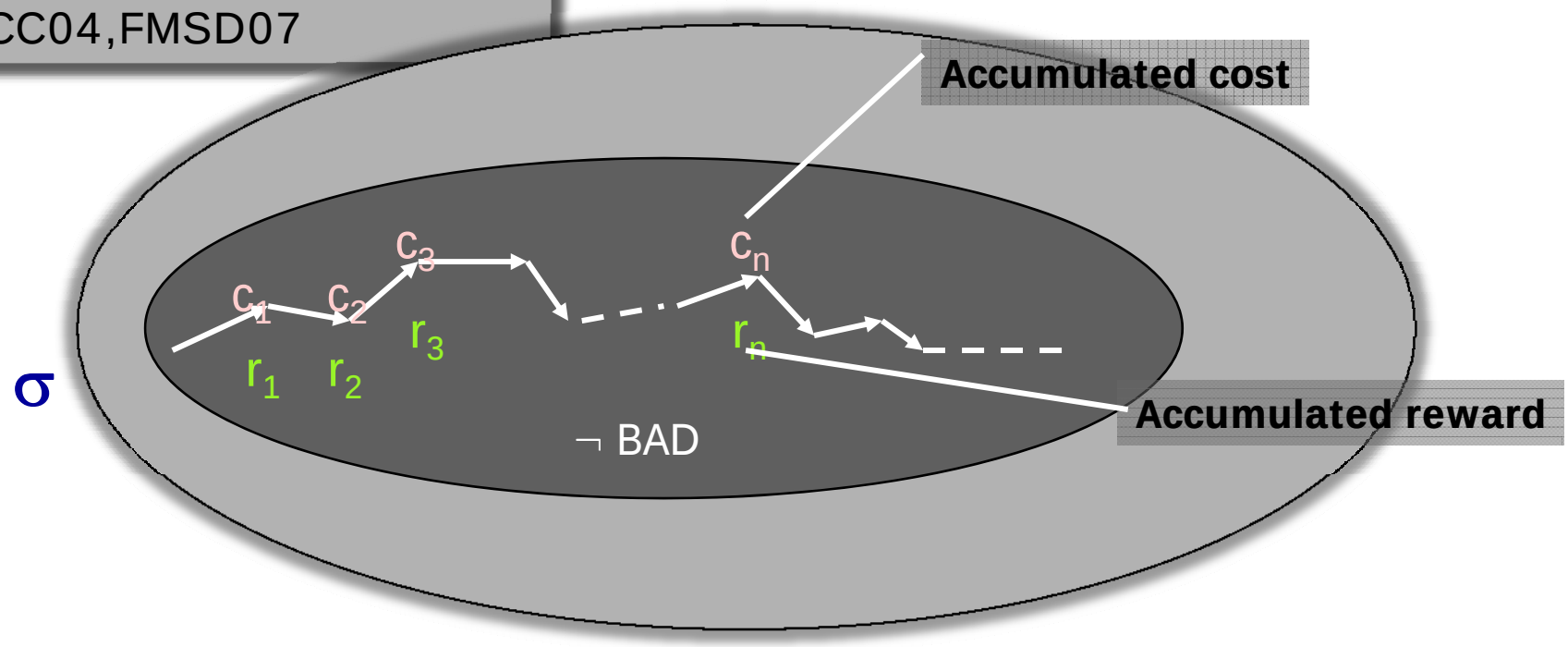
Optimal Infinite Scheduling



Maximize throughput:
i.e. maximize Reward / Cost in the long run

Mean Pay-Off Optimality

Bouyer, Brinksma, Larsen:
HSCC04,FMSD07



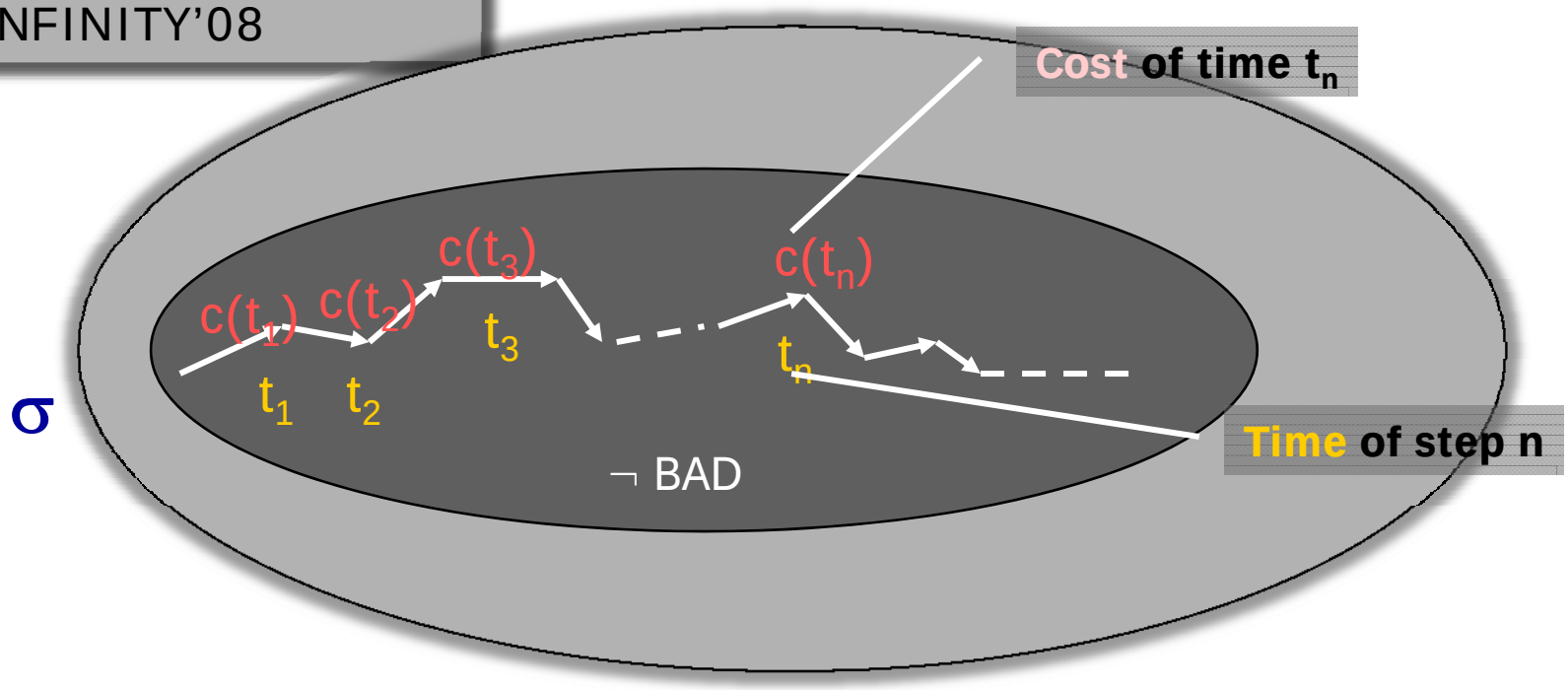
Value of path σ: $\mathrm{val}(\sigma) = \lim_{n\to\infty} c_n/r_n$

Optimal Schedule σ^*: $\mathrm{val}(\sigma^*) = \inf_\sigma \mathrm{val}(\sigma)$

Discount Optimality

$\lambda < 1$: discounting factor

Larsen, Fahrenberg: INFINITY'08

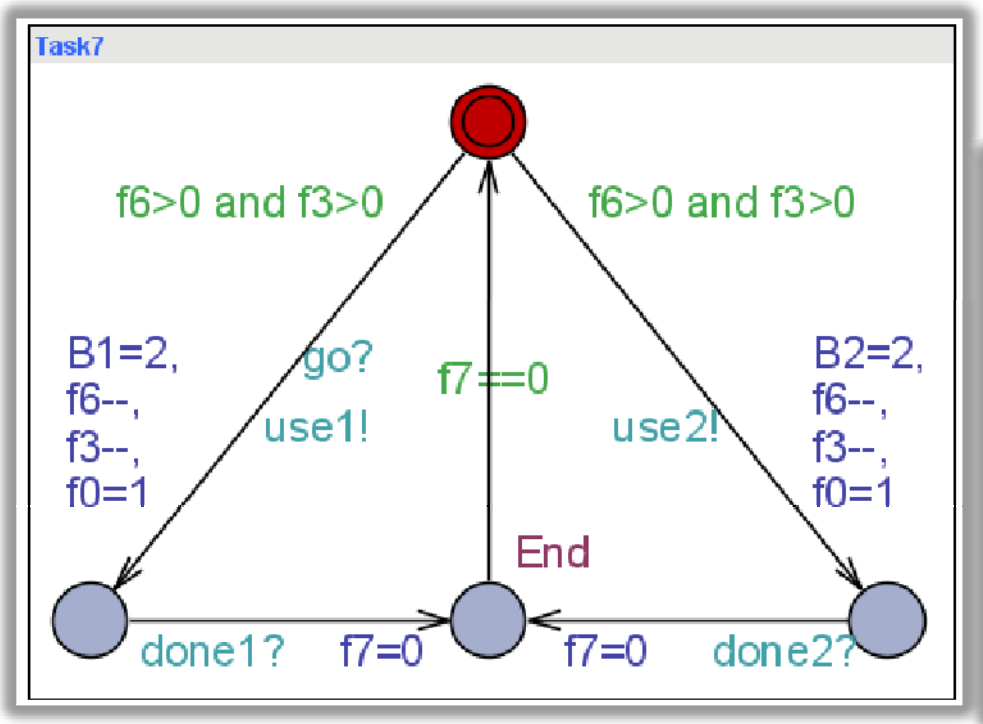


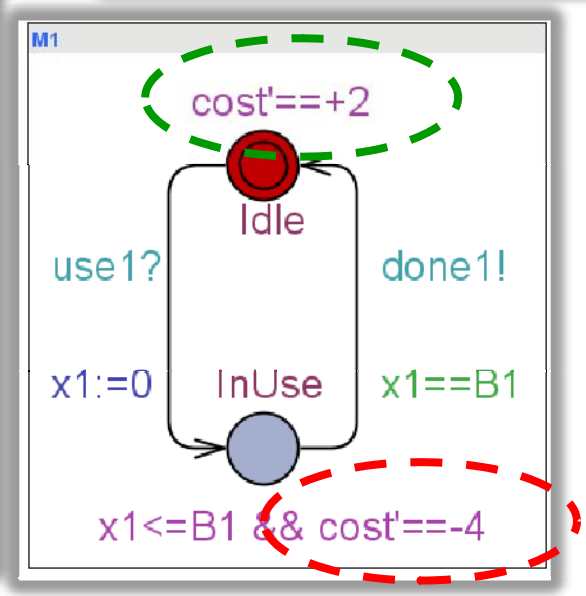
Value of path σ: $\mathrm{val}(\sigma) = \int_{t=0}^{t=\infty} c(t)\lambda^t dt$

Optimal Schedule σ^*: $\mathrm{val}(\sigma^*) = \inf_\sigma \mathrm{val}(\sigma)$

Consuming & Harvesting Energy

Bouyer, Fahrenberg, Larsen, Markey, Srba: **FORMATS 2008**



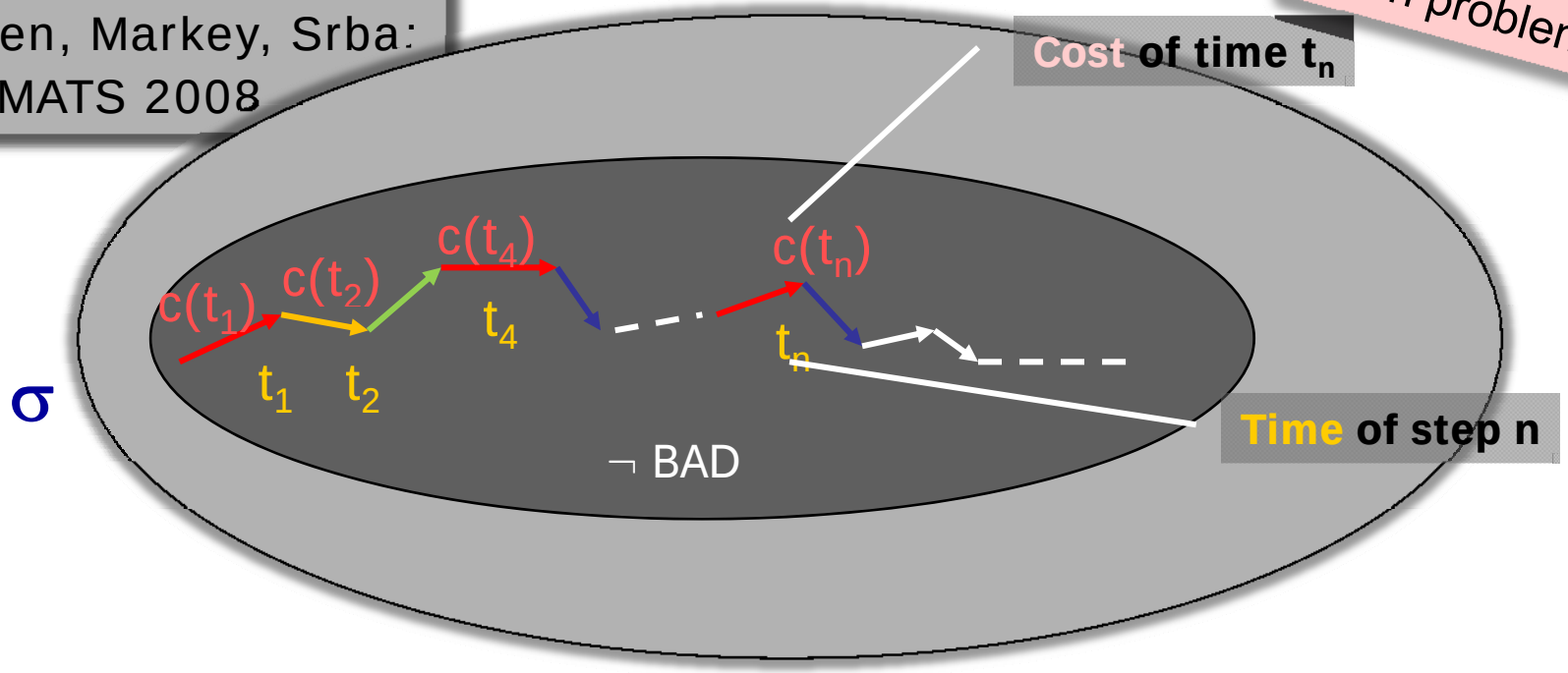


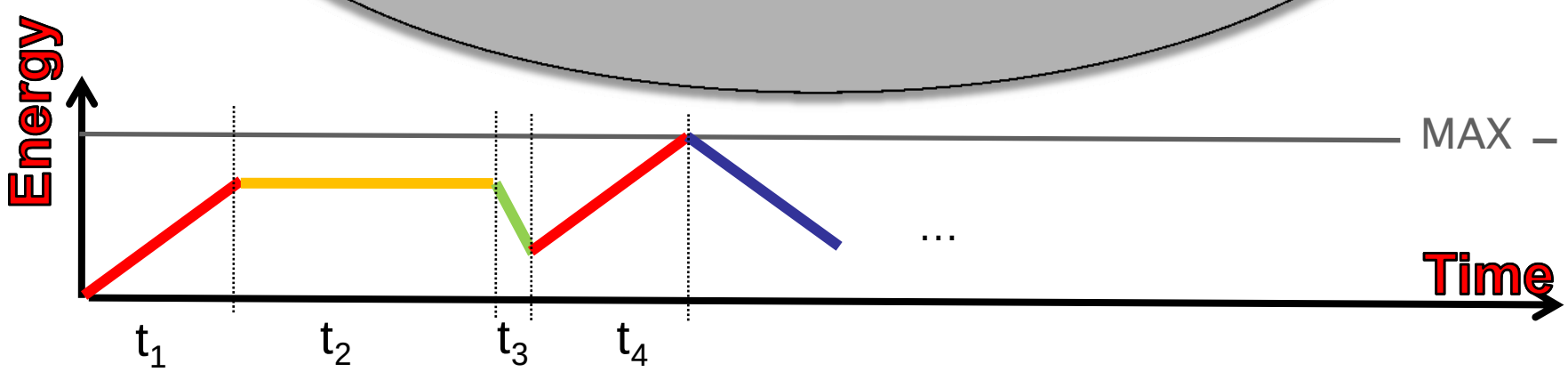
Maximize throughput
while respecting: $0 \leq \mathbf{E} \leq \mathrm{MAX}$

Energy-Bounded Infinite Runs

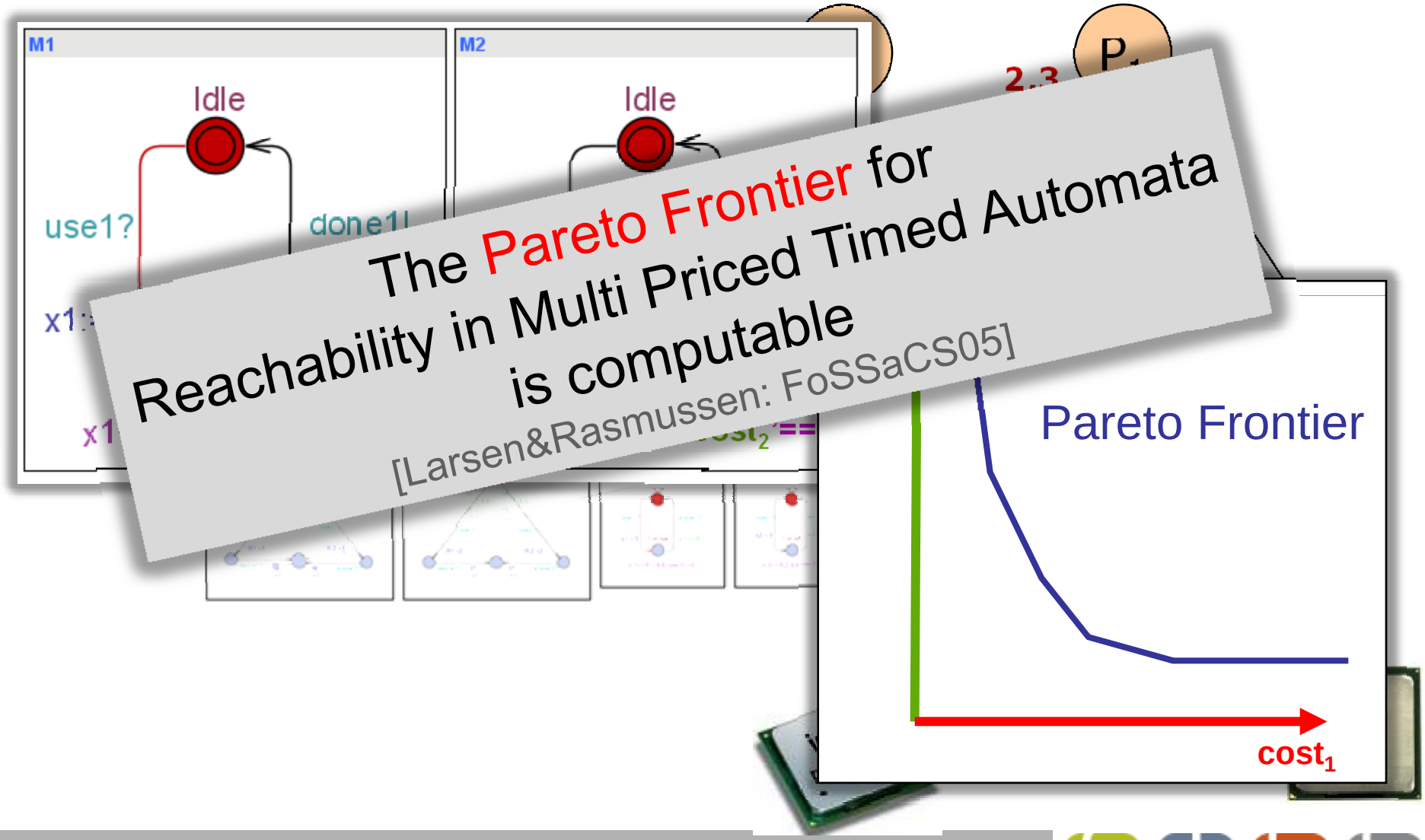
Bouyer, Fahrenberg, Larsen, Markey, Srba: FORMATS 2008

Several open problems





Multiple Objective Scheduling



The Pareto Frontier for Reachability in Multi Priced Timed Automata is computable

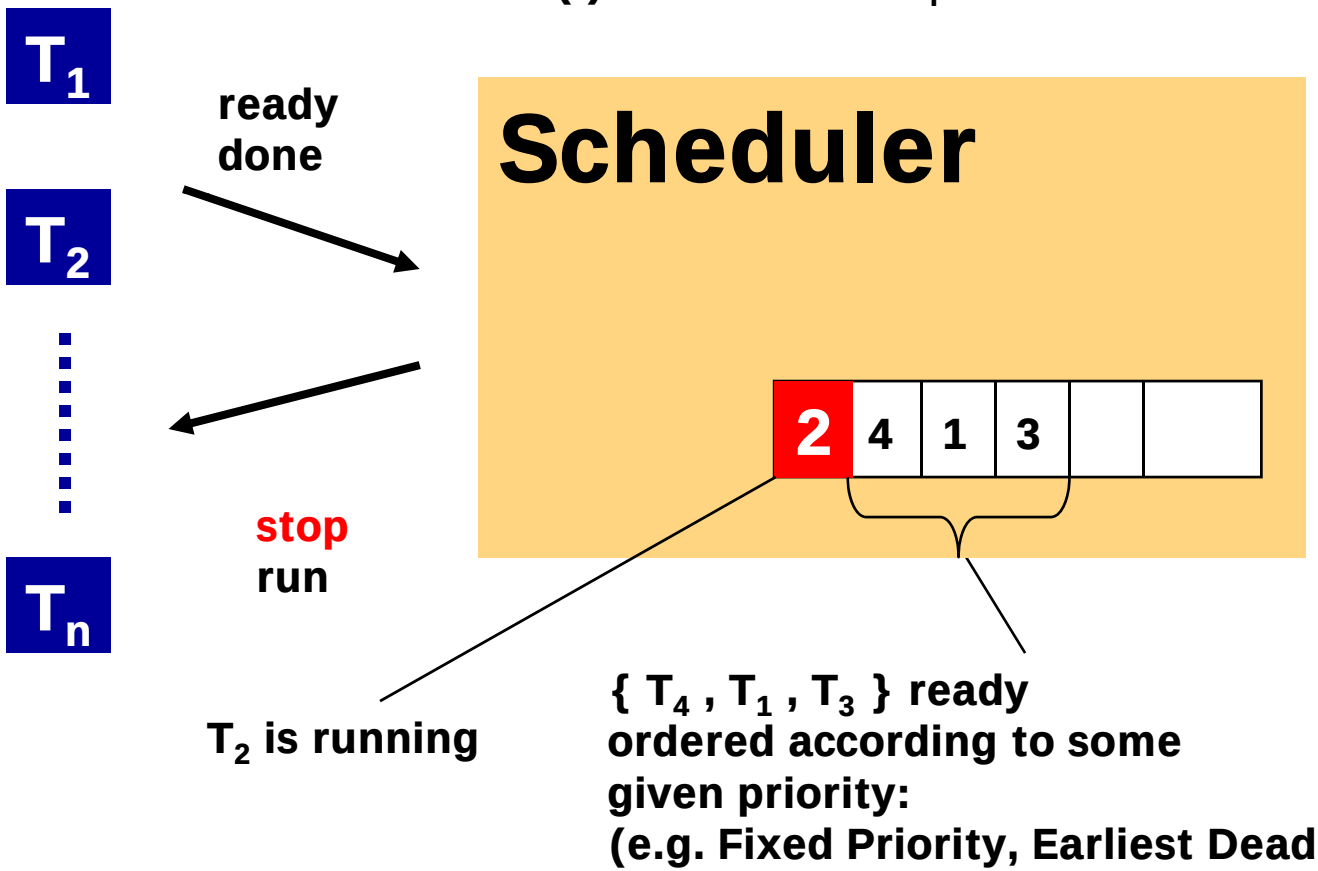
[Larsen&Rasmussen: FoSSaCS05]

Schedulability Analysis

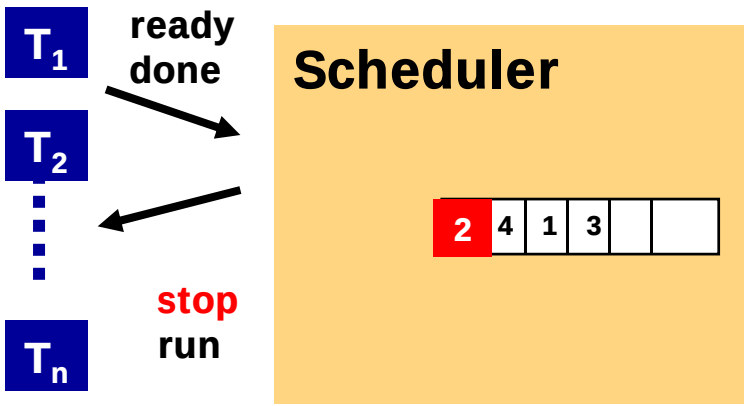
Task Scheduling

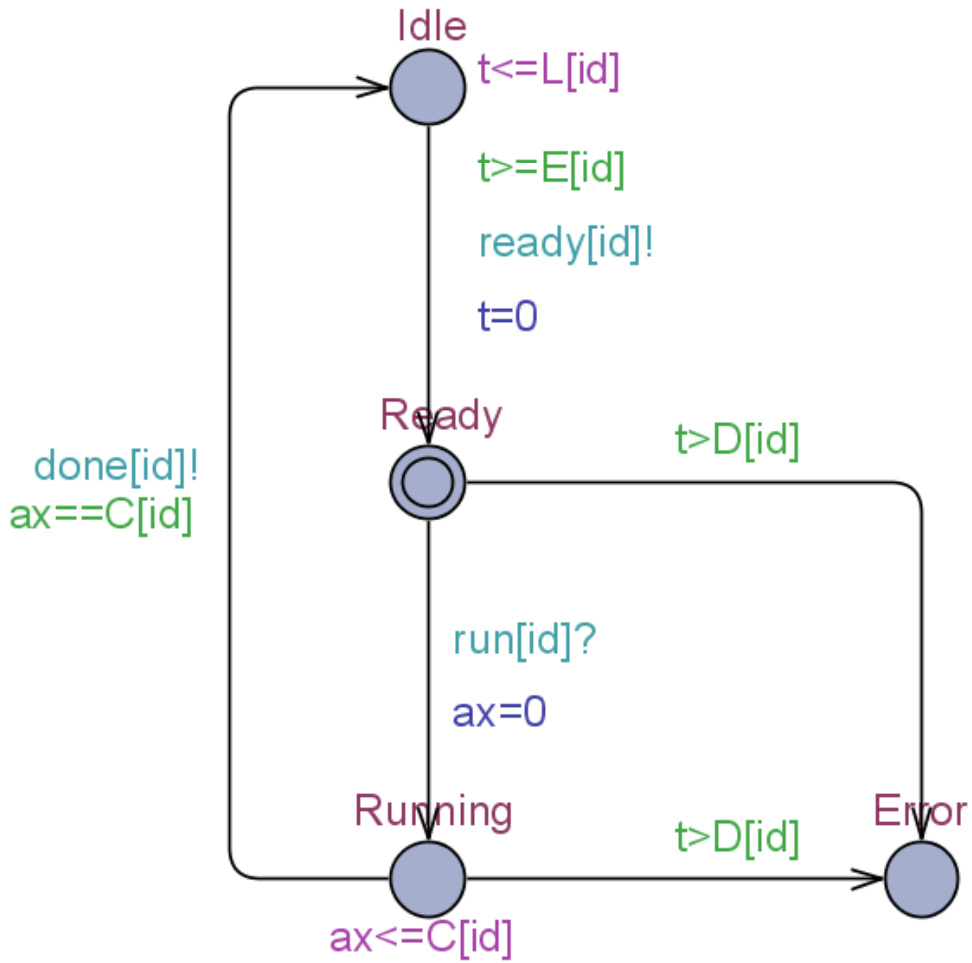
utilization of CPU

P(i), [E(i), L(i)], .. : period or
earliest / latest arrival or .. for T_i
C(i): execution time for T_i
D(i): deadline for T_i

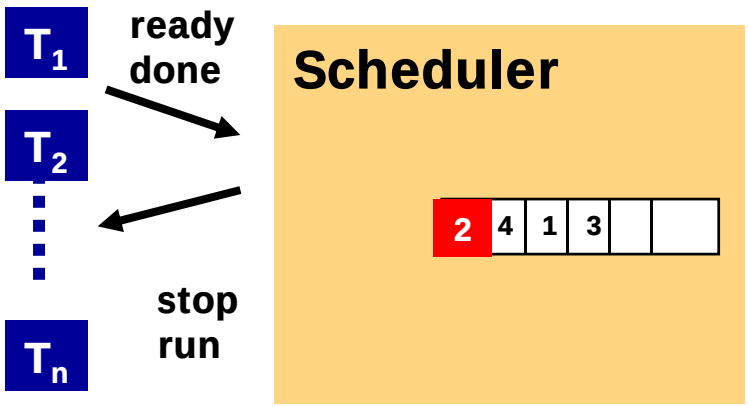


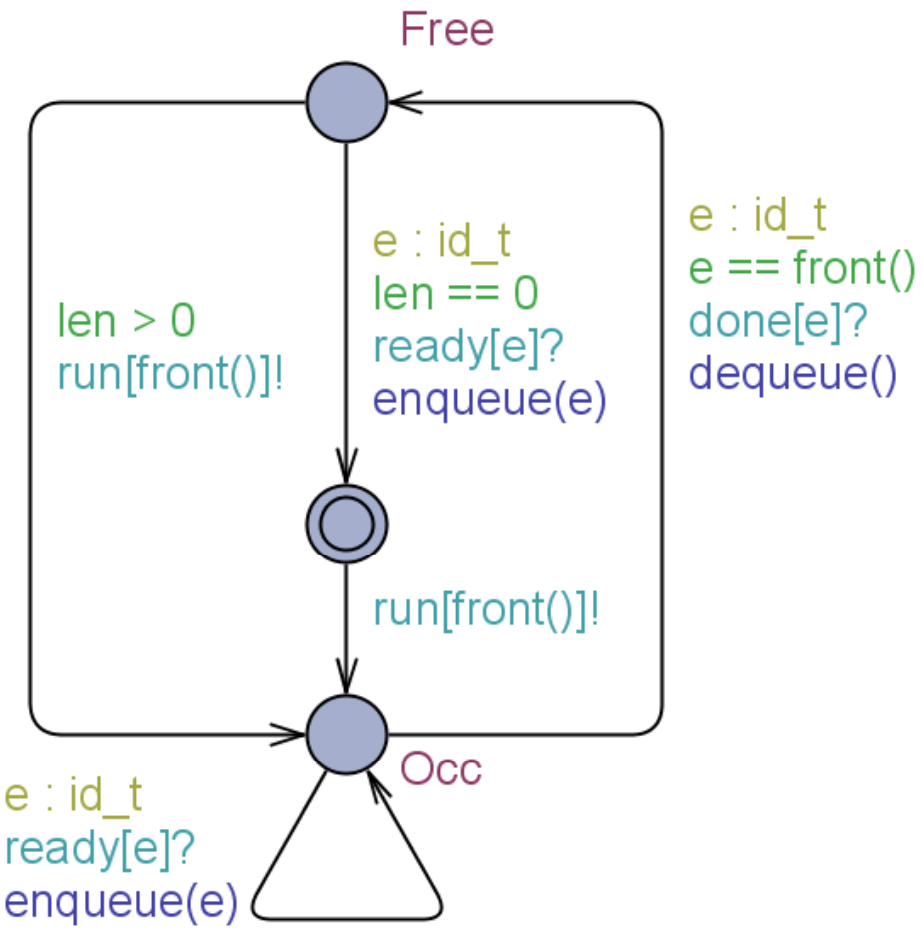
Modeling Task





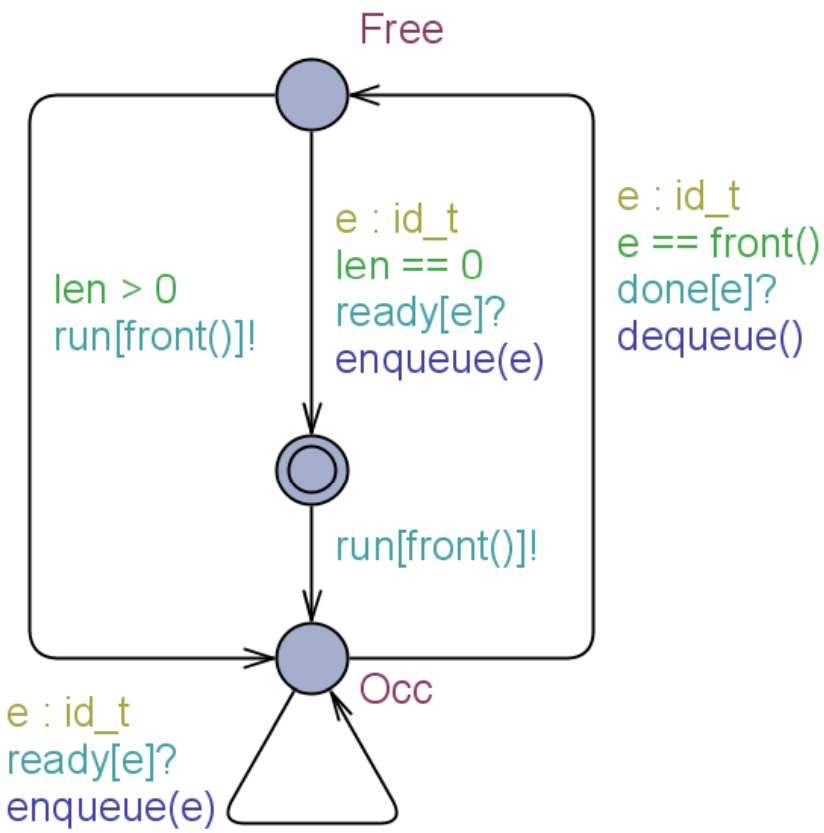
Modeling Scheduler





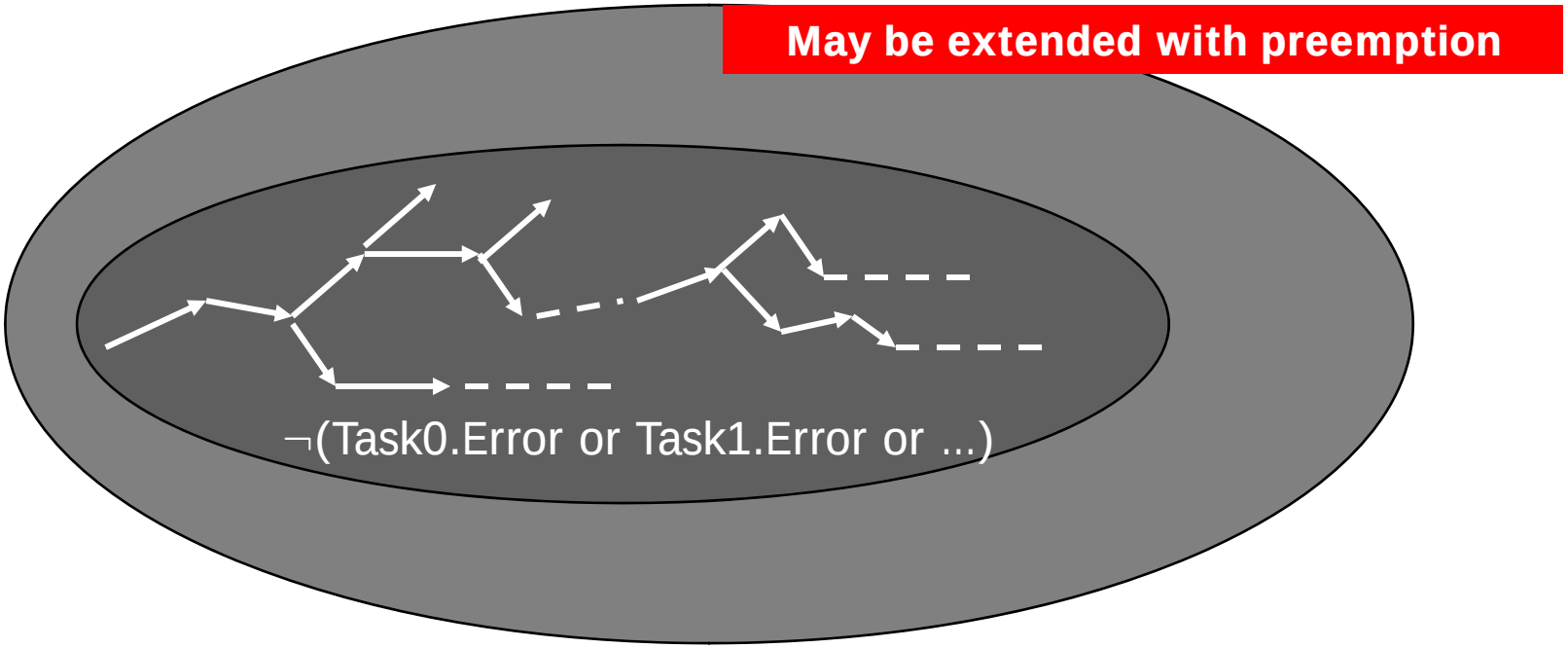
Modeling Queue

In UPPAAL 4.0
User Defined Function



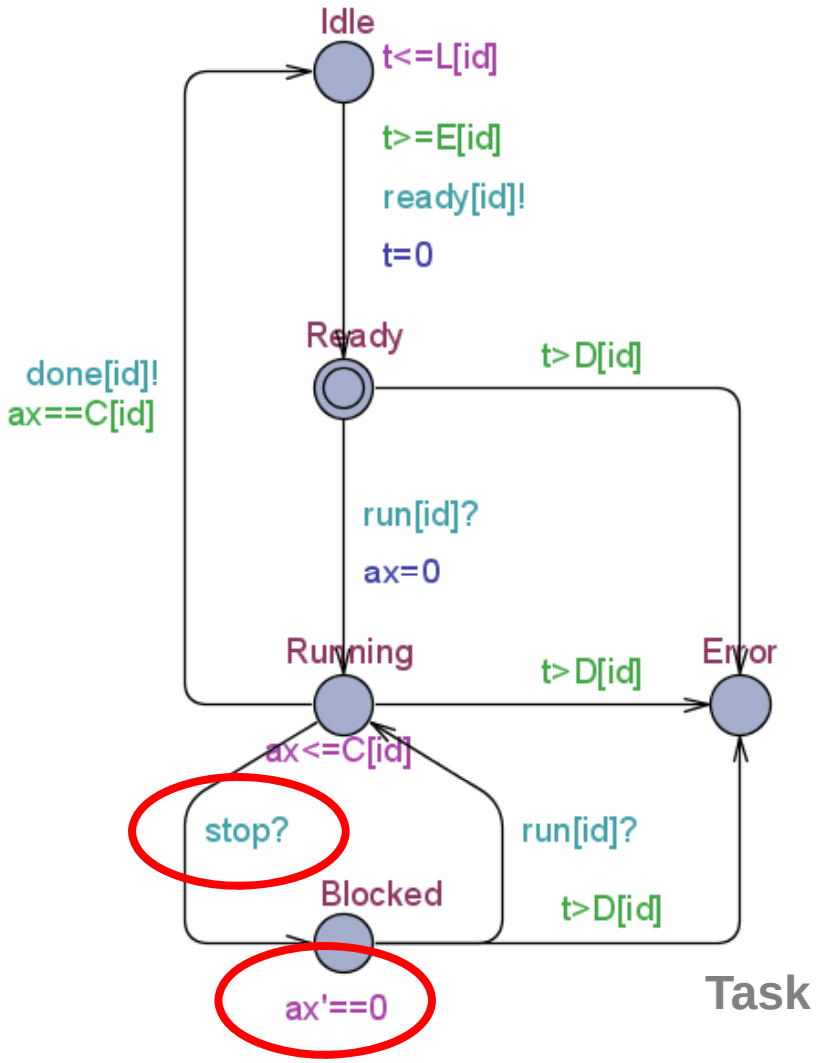

```
// Put an element at the end of the queue
void enqueue(id_t element)
{
int tmp=0;
list[len++] = element;
if (len>0)
{
        int i=len-1;
        while (i>1 && P[list[i]]>P[list[i-1]])
        {
                tmp = list[i-1];
                list[i-1] = list[i];
                list[i] = tmp;
                i--;
        }
}
}

// Remove the front element of the queue
void dequeue()
{ ......
```

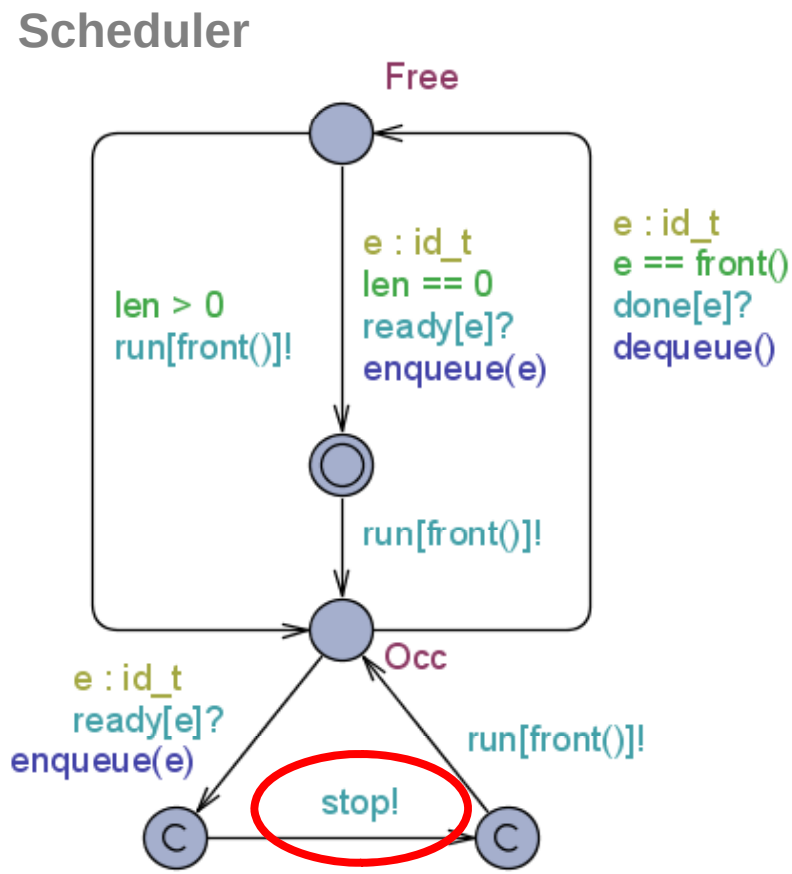
Schedulability = Safety Property



A□ ¬(Task0.Error or Task1.Error or ...)

Preemption – Stopwatches!



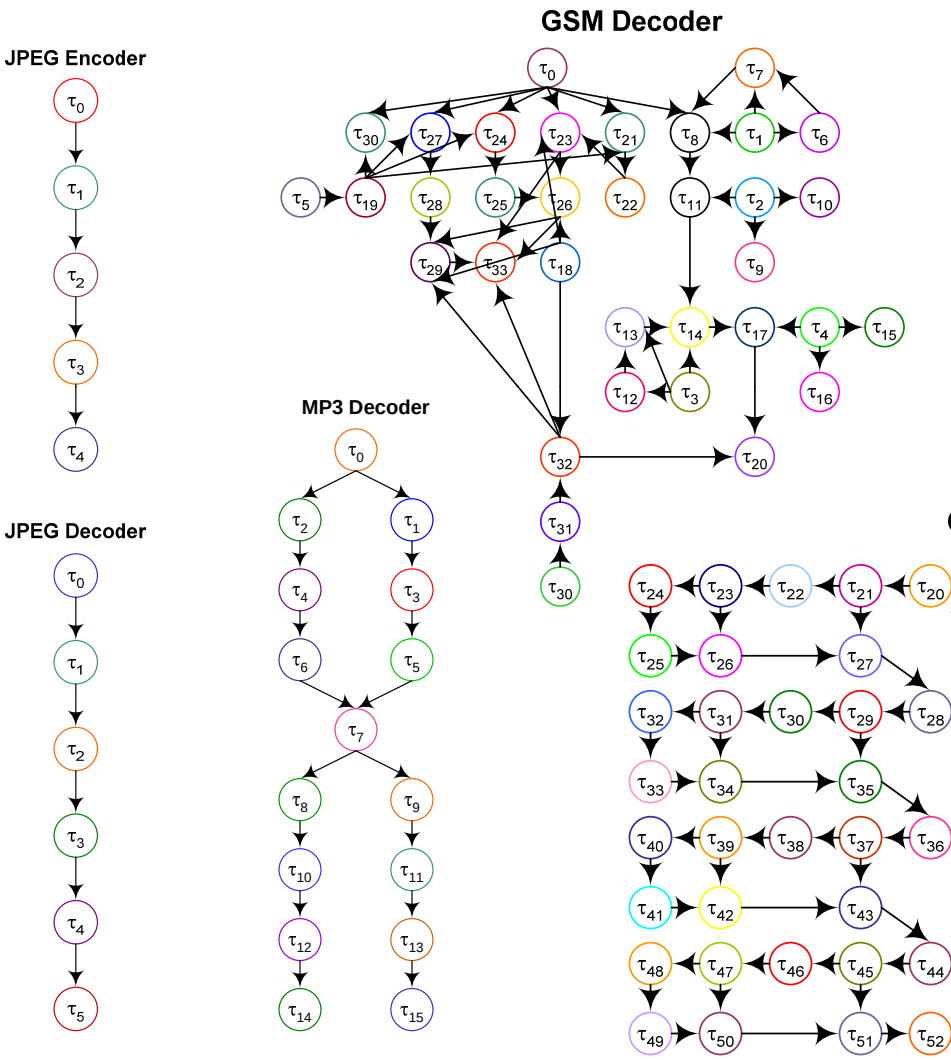


Defeating undecidability ☺

Handling realistic applications?

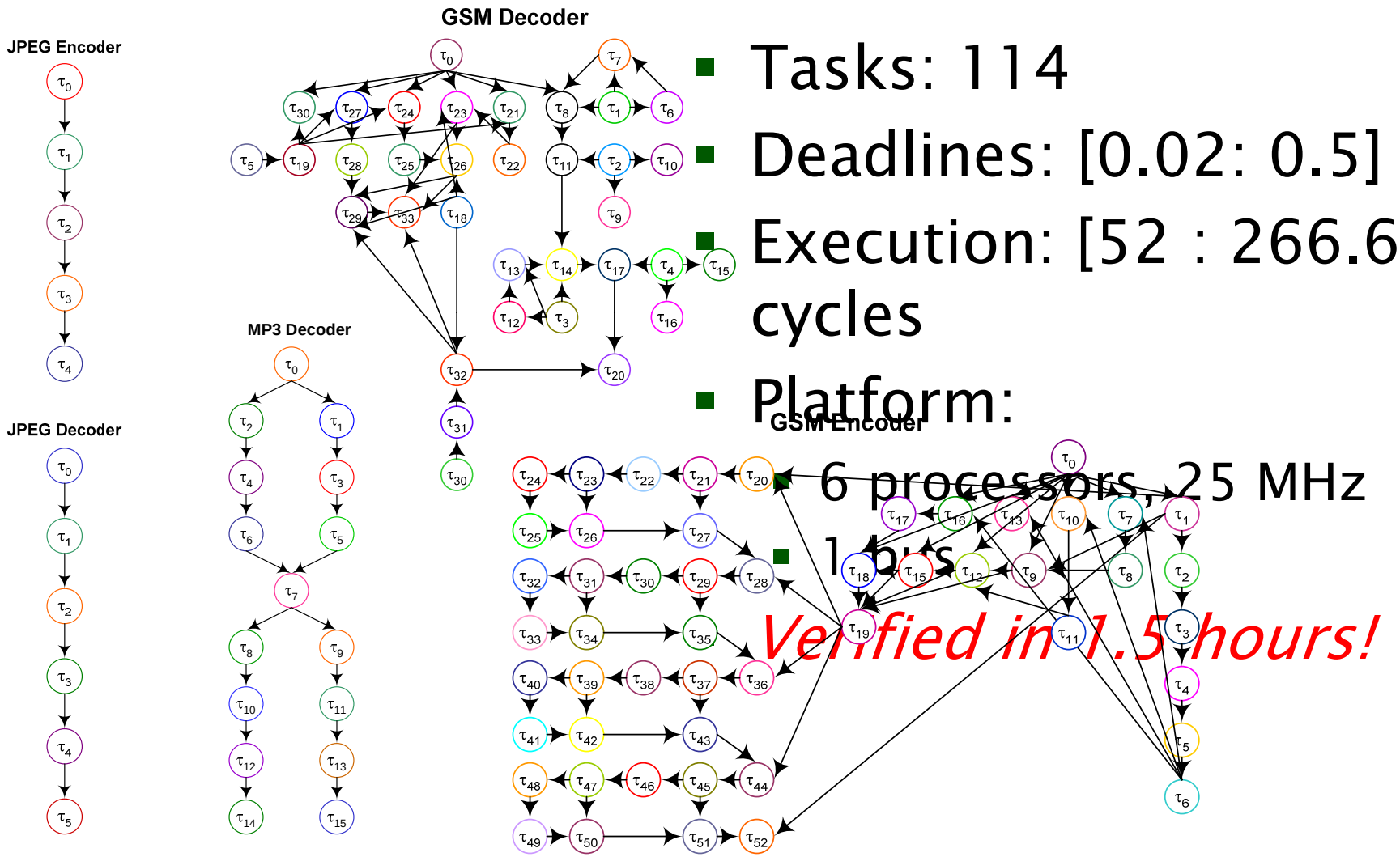
Jan Madsen / DTU

Smart phone:



[Application from Marcus Schmitz, TU Linkoping]

Smart phone



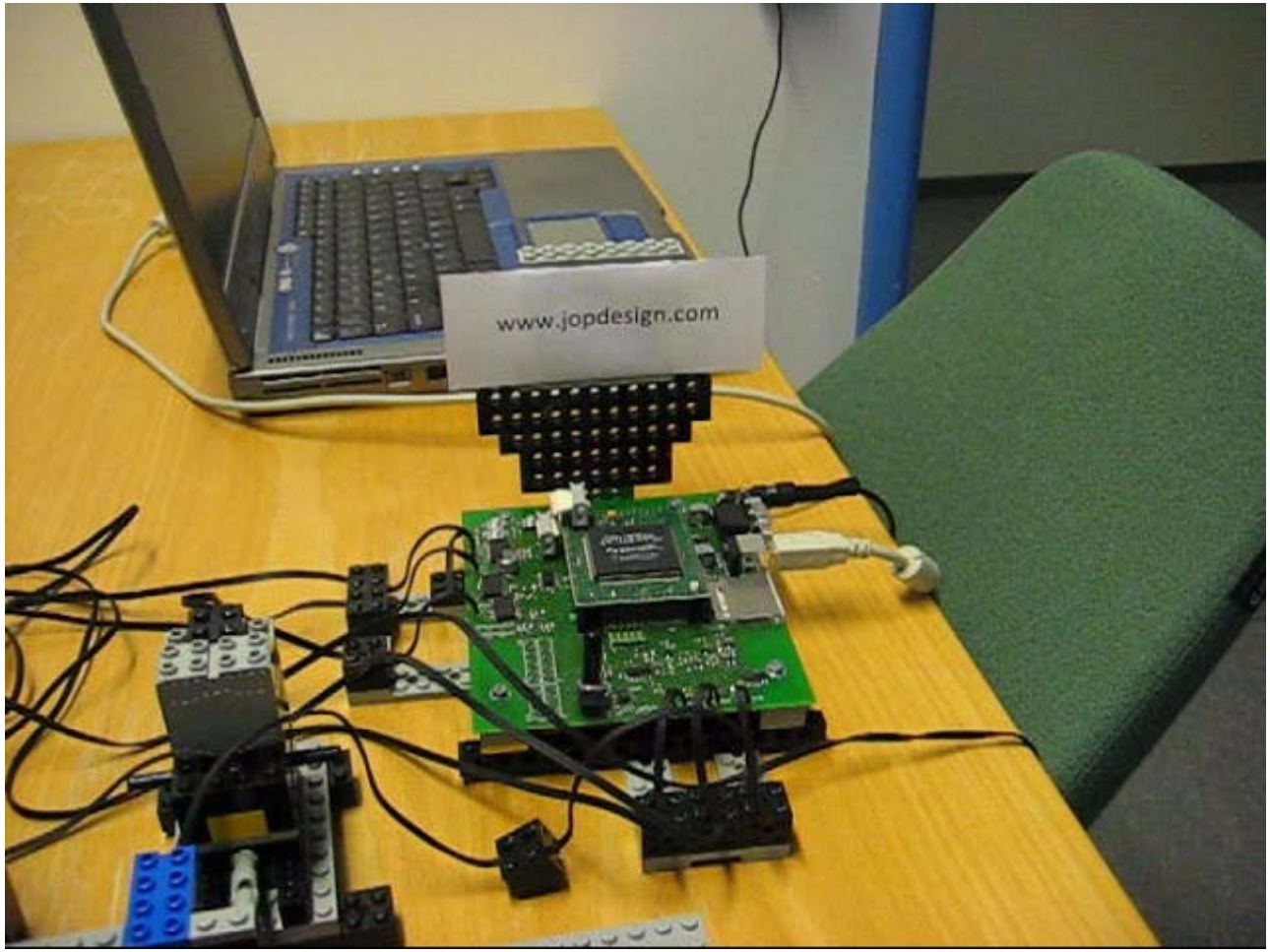
- Tasks: 114
- Deadlines: [0.02: 0.5] sec
- Execution: [52 : 266.687] cycles
- Platform:
  - 6 processors, 25 MHz
  - 1 bus

Verified in 1.5 hours!

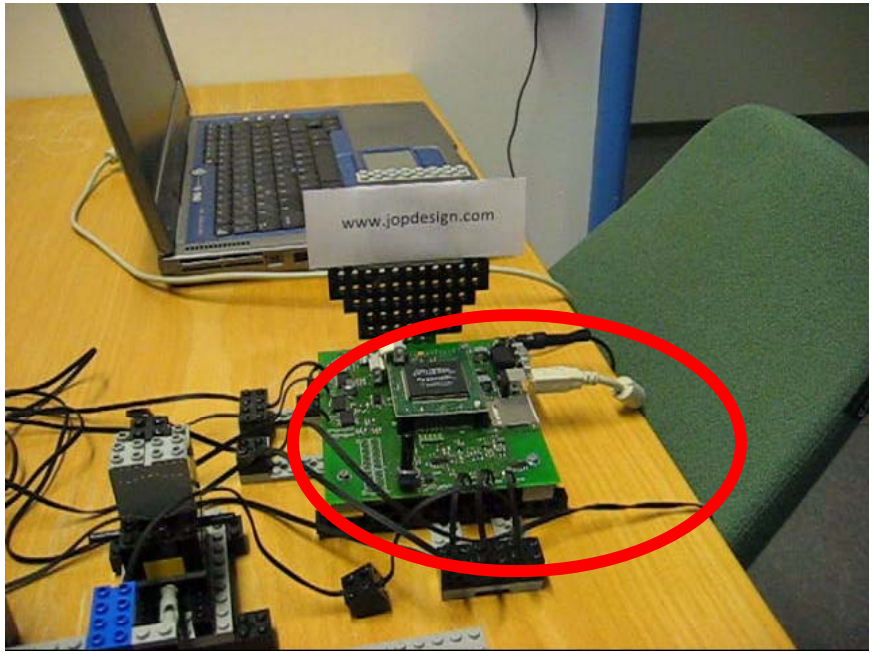
Safety Critical Java Schedulability Analysis

With Bent Thomsen
Petur Olesen
Thomas Bøghholm

A Safety Critical System



Hardware

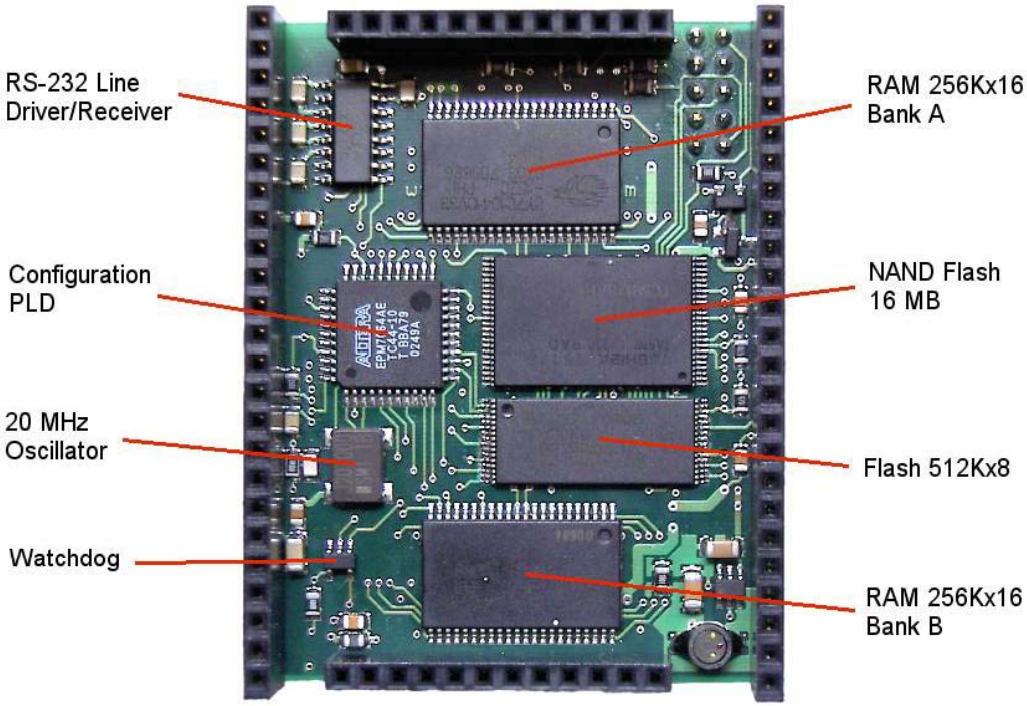


- JOP (Java Optimized Processor)
- Native execution of Java Bytecode
- Bytecode implemented in Microcode
- Avoid unpredictable data-cache
- Time predictable
- Developed new method and stack cache
- Implemented in FPGA

Java Optimizing Processor

Martin Schöberl
University of Tech., Vienna

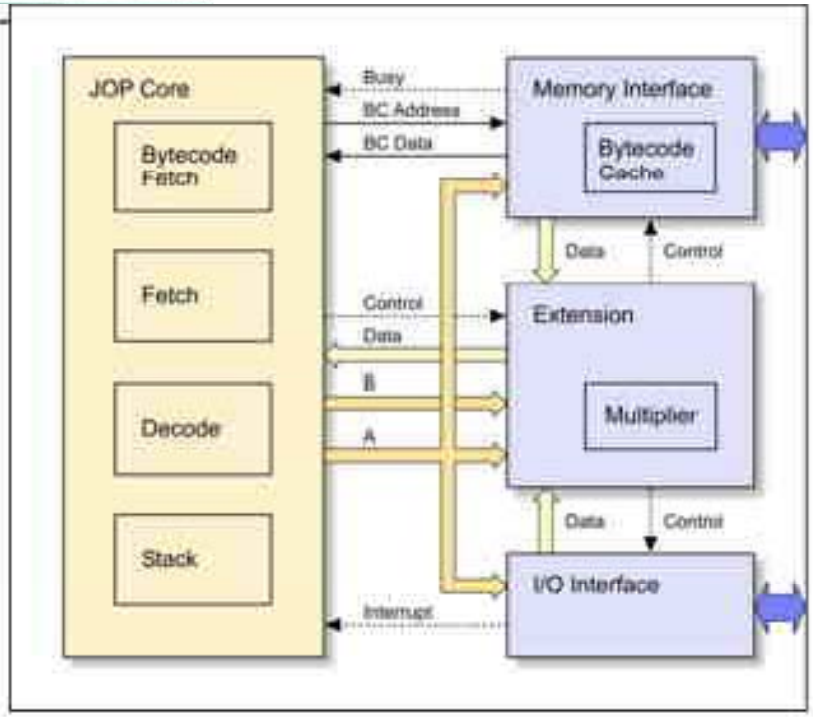
FPGA

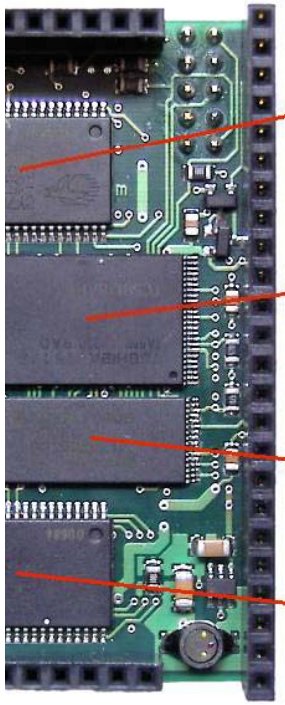


JOP Block Diagram

FPGA

3-4 different FPGAs
6 different boards





Safety Critical Java

Min interarrival

Tasks

Deadline

```
public static void main(String[] args) {
   new SporadicPushMotor(
        new SporadicParameters(4, 4000, 60), 0);
   new SporadicPushMotor(
        new SporadicParameters(2, 4000, 60), 1);

   PeriodicMotorSpooler motorSpooler =
        new PeriodicMotorSpooler(
                new PeriodicParameters(4000));

   new PeriodicReadSensor(
        new PeriodicParameters(2000), motorSpooler);

   RealtimeSystem.start();
}
```

Byte code – Micro code

```
protected boolean run()
    if i<5 {
        i = i + 4;
    } else {
        i = i * 4;
    }
    return true;
}
```

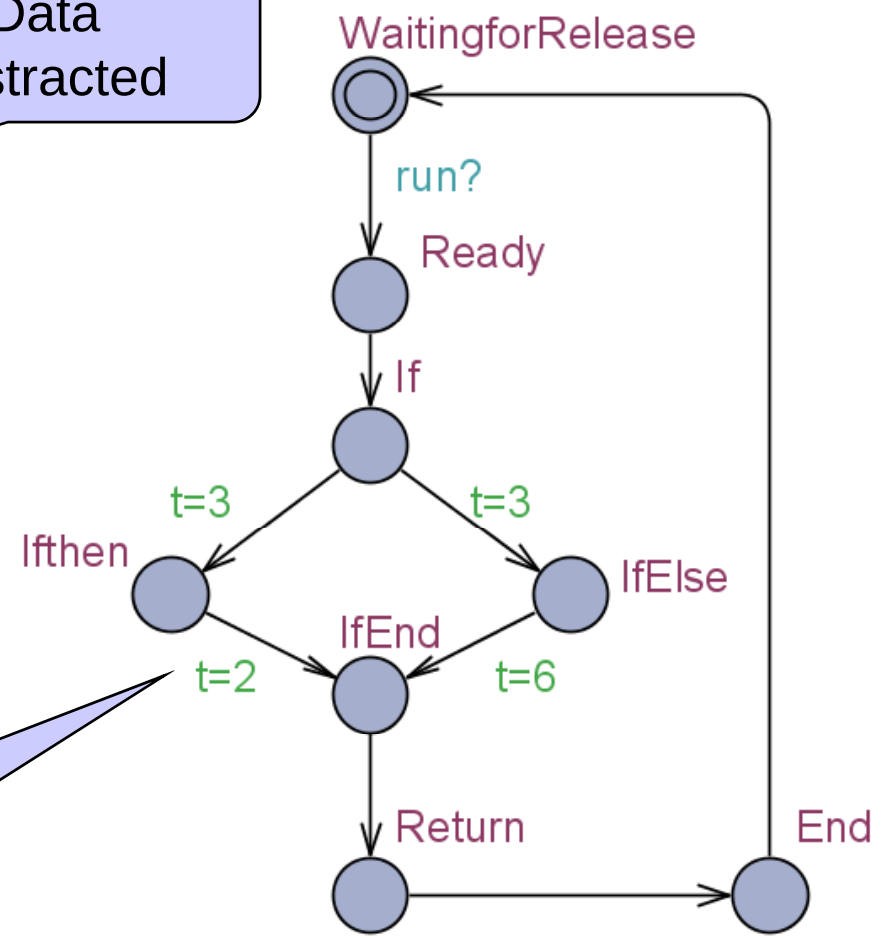
```
Method: run ()Z
 0: aload_0
 1: getfield
 4: ifeq -> 20
 7: aload_0
 8: dup
 9: getfield
 12: iconst_1
 13: iadd
 14: putfield
 17: goto -> 30
 20: aload_0
 21: dup
 22: getfield
 25: iconst_1
 26: isub
 27: putfield
 30: iconst_1
 31: ireturn
```

Byte code – Timed Automata

```
protected boolean run()
    if i<5 {
        i = i + 4;
    } else {
        i = i * 4;
    }
    return true;
}
```

Data abstracted

Timing = WCET from microcode



SARTS – from Safety Critical Java

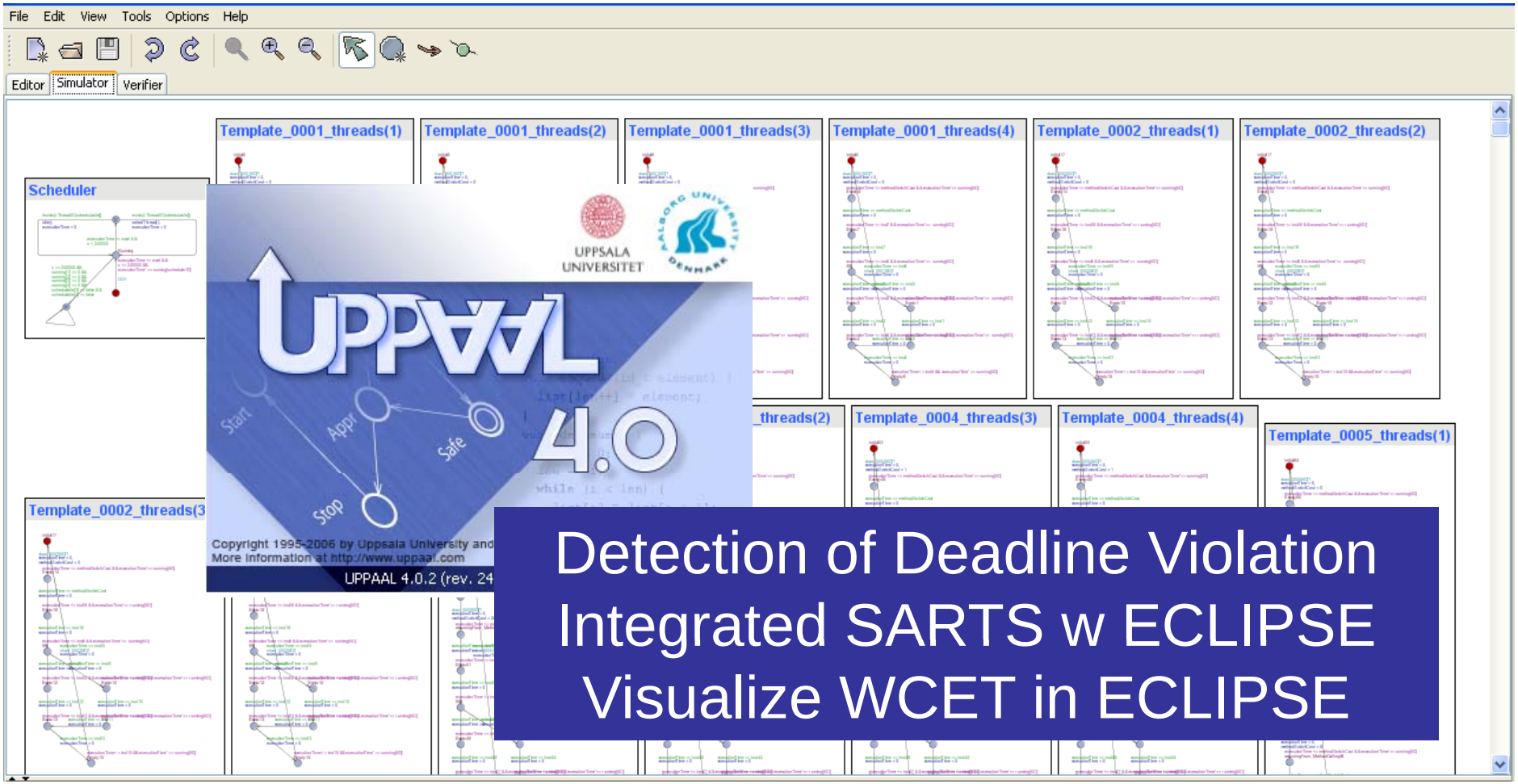
```
public static void main(String[] args) {
        new SporadicPushMotor(
                new SporadicParameters(4, 4000, 60), 0);
        new SporadicPushMotor(
                new SporadicPa

        PeriodicMotorSpooler motor
                new PeriodicMo
                        new

        new PeriodicReadSensor(
                new PeriodicPa

        RealtimeSystem.start();
}
```

TASKS

```
private void handleBrick() {
private void handleBrick() {
private void handleBrick() {
private void handleBrick() {
Sensors.synchronizedReadSensors();
private void handleBrick() {
Sensors.synchronizedReadSensors();
private void handleBrick() {
Sensors.synchronizedReadSensors();
int input = (Sensors.getBufferedSensor(0) + Sensors
                .getBufferedSensor(1)) >> 1;

if (awaitingBrick) {
                if (input > lastRead) {
                        lastRead = input;
                } else if ((lastRead - input) >= TRESHOLD) {
                        awaitingBrick = false;
                        if (lastRead > BRICK_DETECTED) {
                                brickFound(lastRead);
                        }
                }
```

METHODS

SARTS – to Timed Automata



Detection of Deadline Violation
Integrated SARTS w ECLIPSE
Visualize WCET in ECLIPSE

18 methods + 4 tasks = 76 components

METAMOC

Modular Execution Time Analysis using Model Chekcing

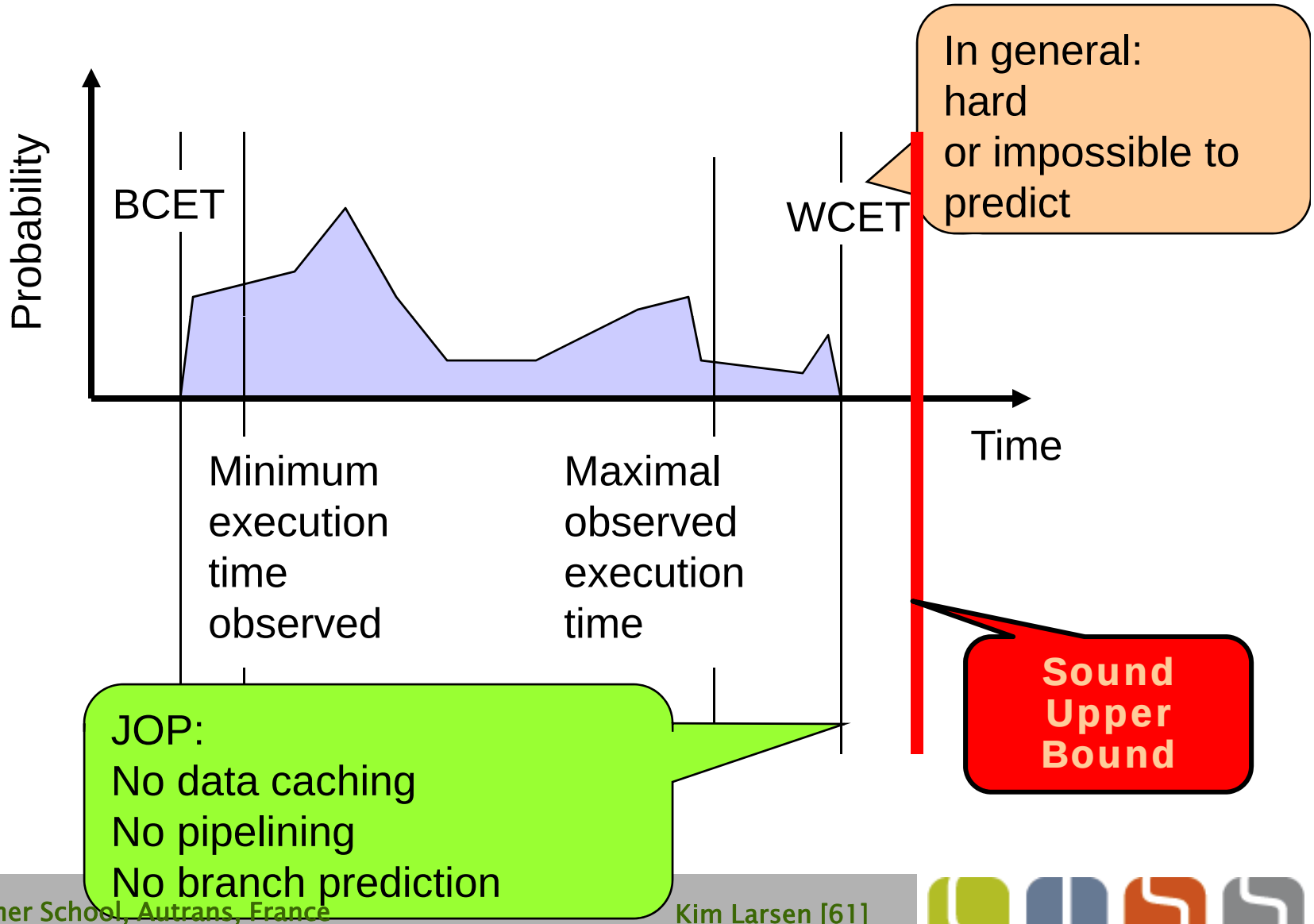
With Rene R Hansen
Andreas Dalsgaard
Mads Olesen
Martin Toft

WCET: Worst Case Execution Time

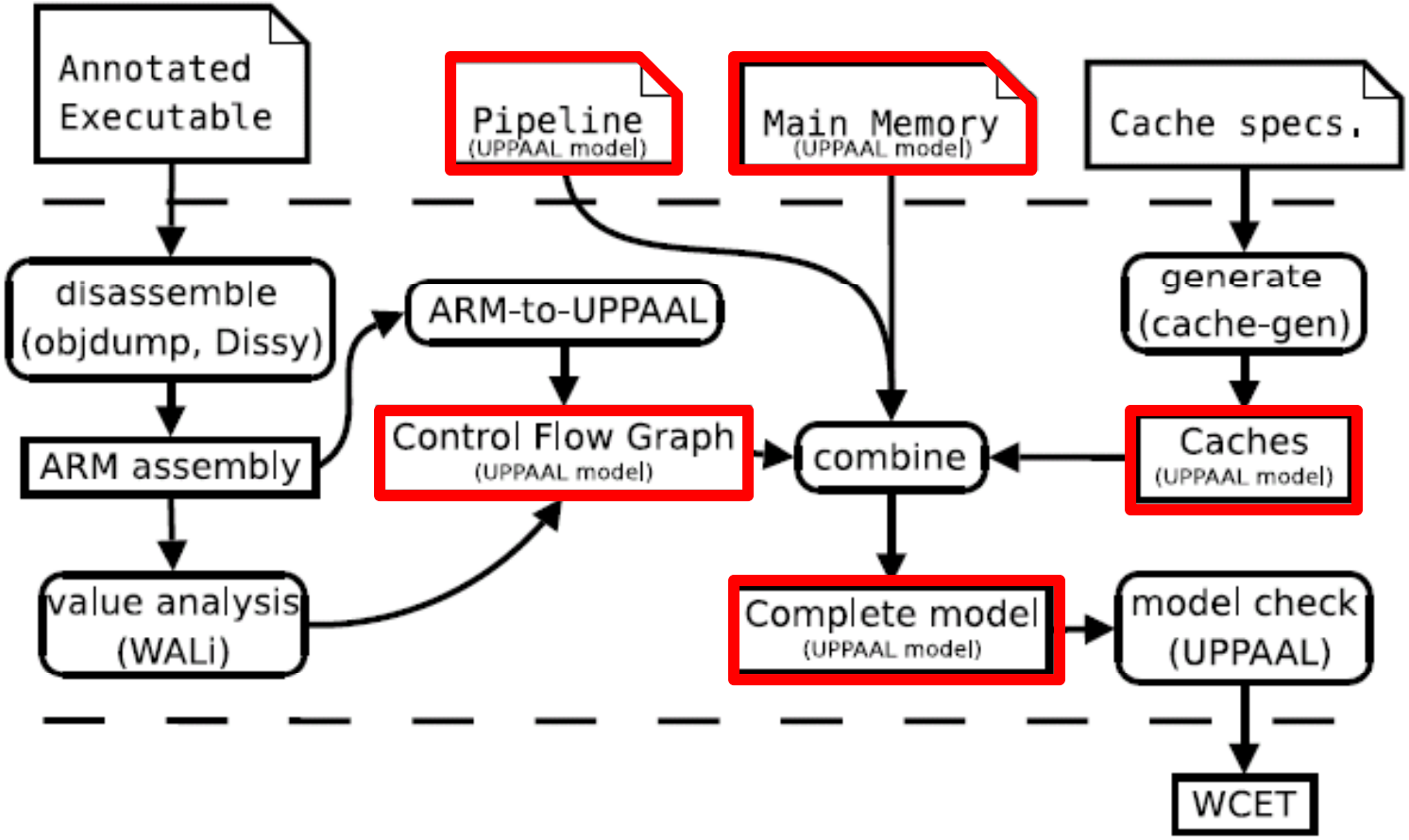
- Isolated, non-blocking code
- Annotation with loop counts.
- Cache used for speeding up access of memory blocks.
- Full associatively.
- LRU replacement strategy.

```
var A;
var B;
var C;
var D;
var E;

while (E < 11) 10 {
   output B;
   output C;
   output A;
   output D;
   output C
}
```

WCET: Worst Case Execution Time



Tool Chain


Annotated Executable
Pipeline (UPPAAL model)
Main Memory (UPPAAL model)
Cache specs.
disassemble (objdump, Dissy)
ARM-to-UPPAAL
generate (cache-gen)
ARM assembly
Control Flow Graph (UPPAAL model)
combine
Caches (UPPAAL model)
value analysis (WALi)
Complete model (UPPAAL model)
model check (UPPAAL)
WCET

Disassembler – Dissy

Dissy - /home/mchro/DAT6/svn/wcet_bench/web/fibcall/fibcall.o

File Navigation Options Help

Lookup Highlight

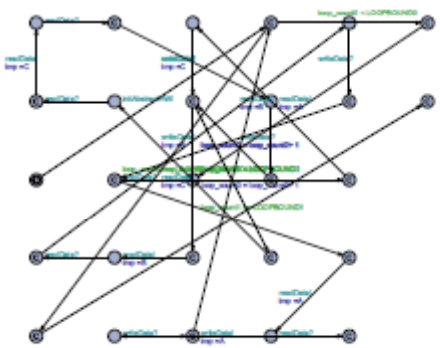
Address	Size	Label
0x0000832c	84	fib
0x00008380	32	main
0x000083a0	4	__libc_csu_fini
0x000083a4	116	__libc_csu_init
0x0001043c	0	__frame_dummy_init_array_entry
0x00010440	0	__do_global_dtors_aux_fini_array_entry
0x00010440	4	__init_array_end

Address	b0	b1	b2	Instruction
				int fib(int n)
				{
0x00008330				push {lr}
0x00008334				mov lr, r0
				int i, Fnew, Fold, temp,ans;
				Fnew = 1; Fold = 0;
				for (i = 2;
				i <= 30 && i <= n; /* apsim_loop 1 0
0x00008338				movle r0, #1
0x0000833c				ble 8378
0x00008340				mov r2, #2
0x00008344				mov r1, #1
0x00008348				mov ip, #0
				i++)
0x0000834c				add r2, r2, #1
				{
				int i, Fnew, Fold, temp,ans;

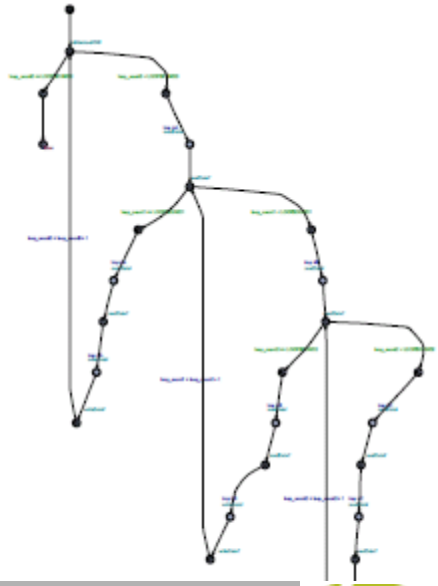
Instruction Information

Model Creation – pyuppaal

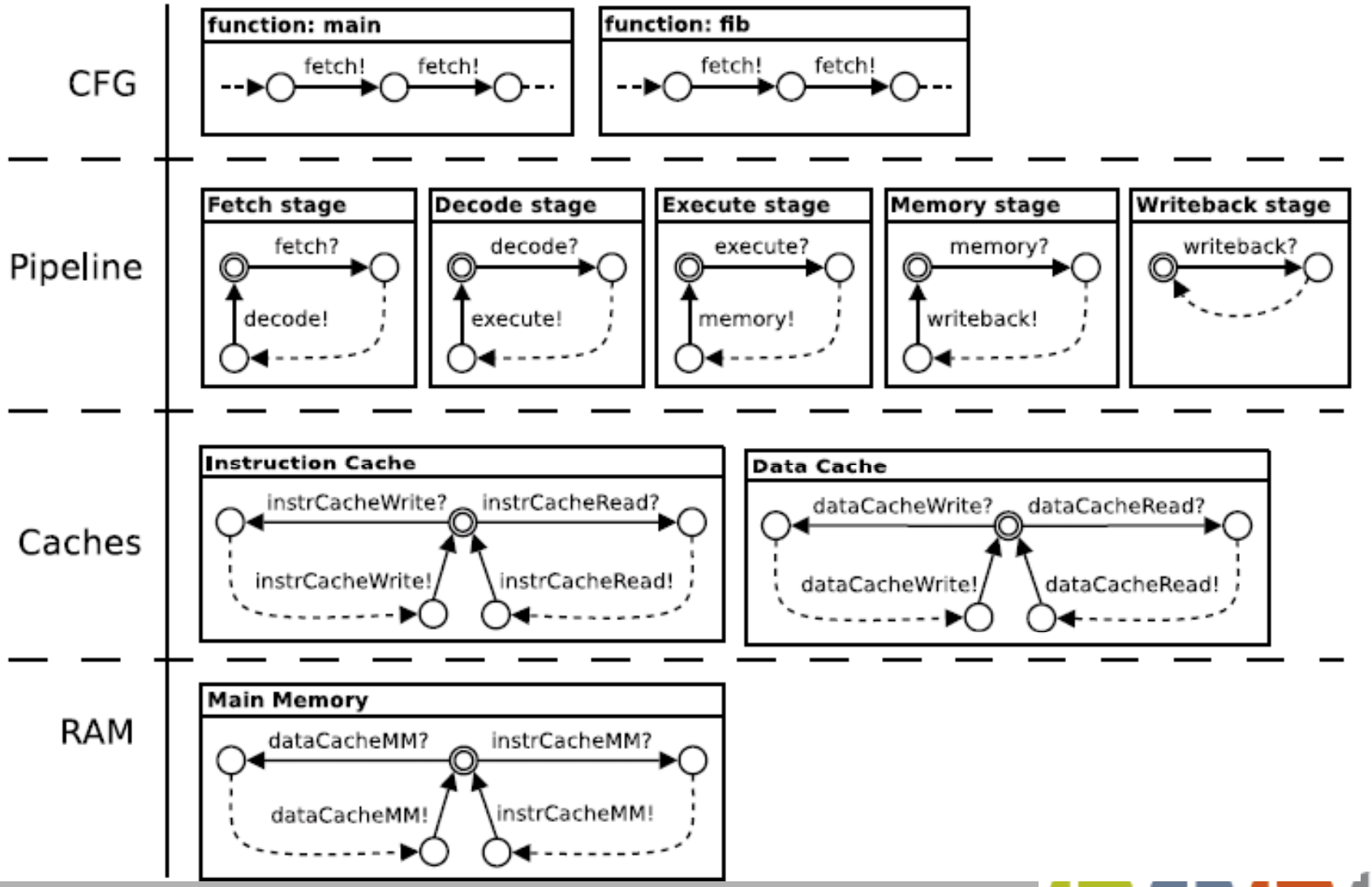
- Library written in Python for manipulating UPPAAL models
- Import/Export UPPAAL models
- Automatic layout of models
- Used to generate most of our models
- Open Source



$\Rightarrow$



Prototype Implementation



Experiments

- Conducted on the concrete implementation for the ARM920T processor
- Examine three qualities:
  - Size and complexity of processes
  - How much sharper WCETs are found by taking caching into account
  - Resource usage (time and memory)
- No evaluation of the pipeline
- No reference WCETs available
- Benchmark programs from the WCET Analysis Project by Mälardalen Real-Time Research Center
  - Wide selection of computation tasks
  - Used to benchmark WCET analysis methods

Experiments

- The most interesting findings:
  - Taking the **instruction cache** into account yields WCETs that are up to 97% sharper (78% on average at -O2)
  - Taking the **data cache** into account yields WCETs that are up to 68% sharper (31% on average at -O2)
  - Almost all results are obtained within five minutes
- Some programs fail due to
  - State space explosion (9)
  - Write to program counter (2)
  - Floating point operations
  - Value analysis problems

Future Work

- Still work in progress
- UPPAAL provides flexible framework for modularization
- Analysis times acceptable.

- Integration of abstract caches
- Better value analysis
- Integration of WCET and Schedulability Analysis

More Information

http://sarts.boegholm.dk
http://metamoc.martintoft.dk

Timed Games

STRATEGY

When you are in

(Task1.End Task2.End Task3._id2 Task4.End Task5._id0 Task6._id0 Task7._id0 M1._id4 M2._id7)
f1=1 f2=1 f3=0 f4=1 f5=0 f6=0 f7=0 f0=1 B1=0 B2=6
(4<=x2 && time==18 && x2<=8),

Take transition

Task6._id0->Task6._id1 { a == 1 && b == 1, use1!, B1 := D1 }
M1._id4->M1._id5 { 1, use1?, x1 := 0 }

When you are in

(Task1.End Task2.End Task3.End Task4.End Task5._id0 Task6._id1 Task7._id0 M1._id5 M2._id6)
f1=1 f2=1 f3=1 f4=1 f5=0 f6=0 f7=0 f0=1 B1=3 B2=10
(18<=time && x1<=6 && time<=22 && time-x1<=18),

Take transition

Task5._id0->Task5._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

(Task1.End Task2.End Task3.End Task4._id0 Task5._id0 Task6._id1 Task7._id0 M1._id5 M2._id6)
f1=1 f2=1 f3=1 f4=0 f5=0 f6=0 f7=0 f0=1 B1=3 B2=2
(x1-time==-10 && time==10),

Take transition

Task4._id0->Task4._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

(Task1.End Task2.End Task3.End Task4.End Task5._id1 Task6._id0 Task7._id0 M1._id5 M2._id6)
f1=1 f2=1 f3=1 f4=1 f5=0 f6=0 f7=0 f0=1 B1=8 B2=8
(x1<=3 && x1-time==-18) || (20<=time && x1-time<=-12 && time<=21 && time-x1<18),

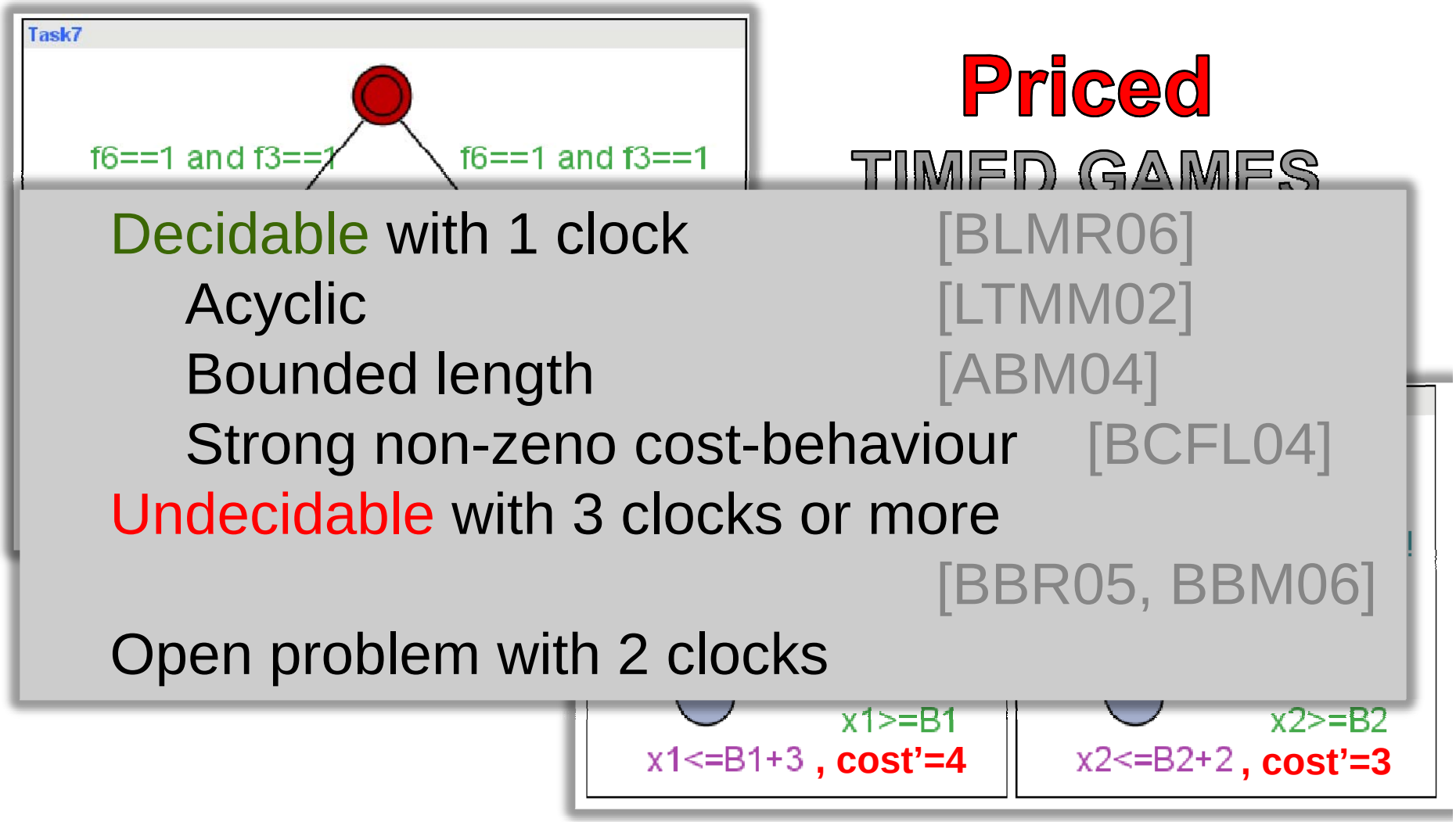
Take transition

Task6._id0->Task6._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

...

Optimal Scheduling under Uncertainty



Priced

TIMED GAMES

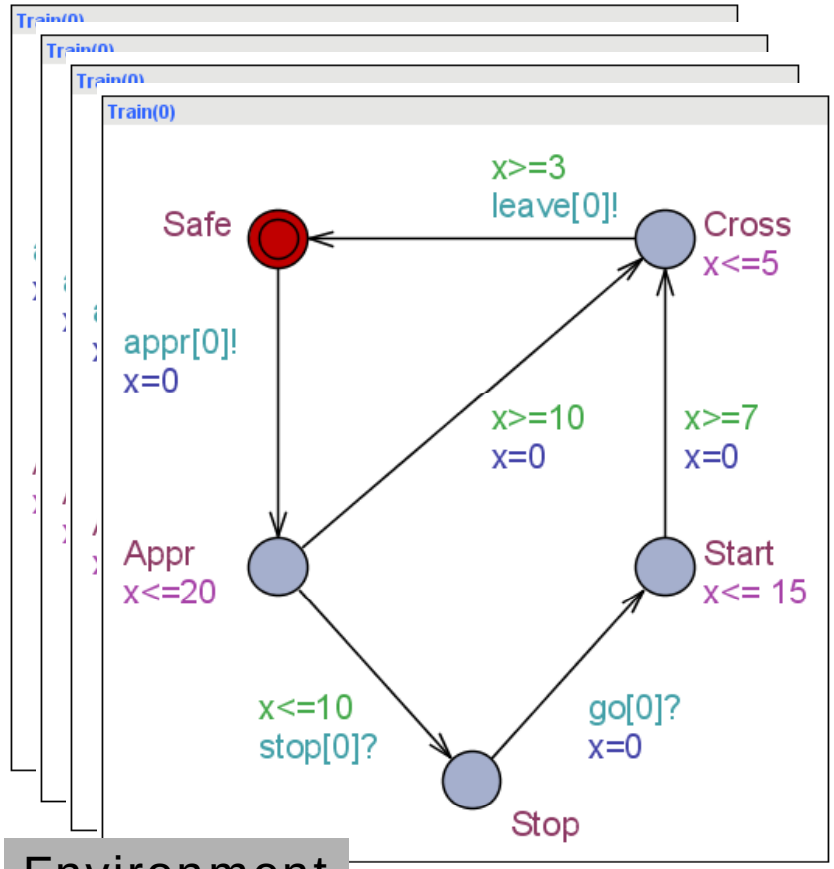
Decidable with 1 clock [BLMR06]

- Acyclic [LTMM02]
- Bounded length [ABM04]
- Strong non-zeno cost-behaviour [BCFL04]

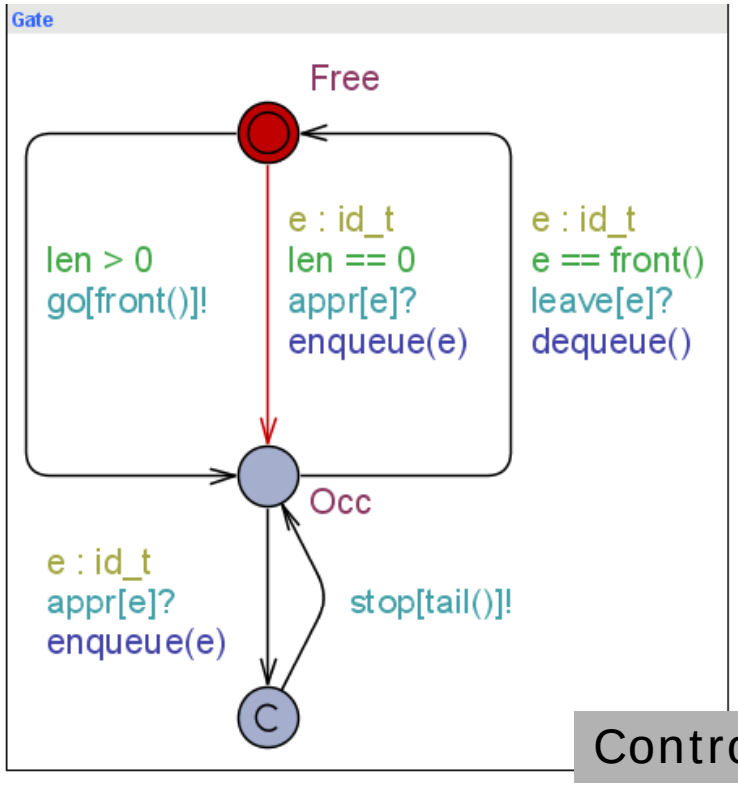
Undecidable with 3 clocks or more [BBR05, BBM06]

Open problem with 2 clocks

Model Checking (ex Train Gate)



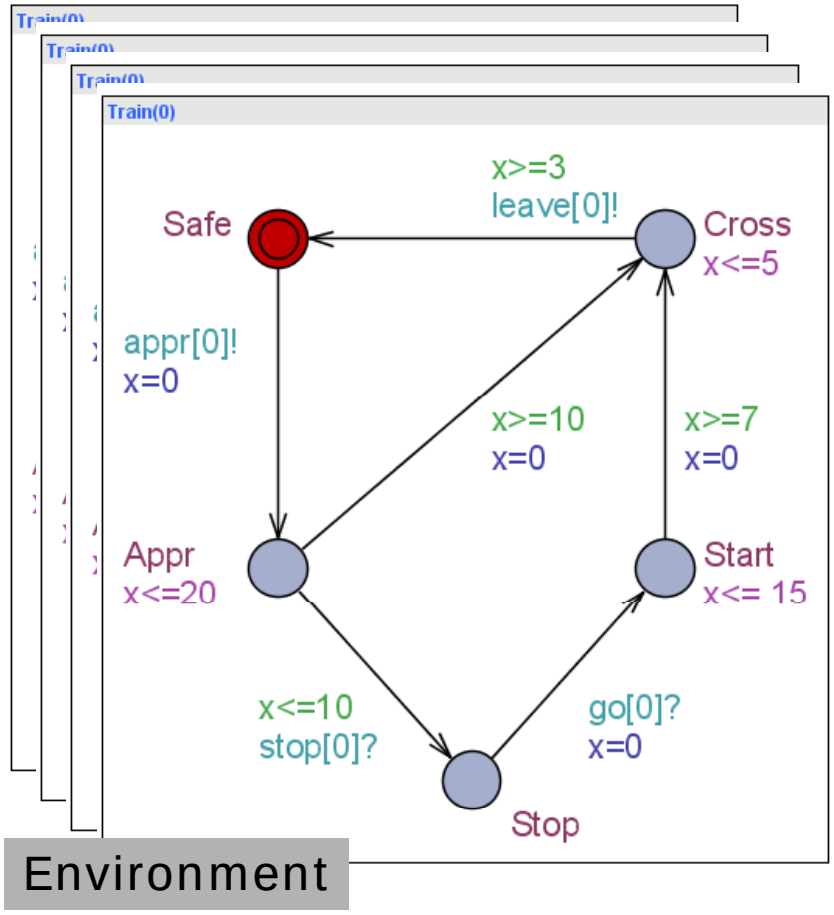
Environment

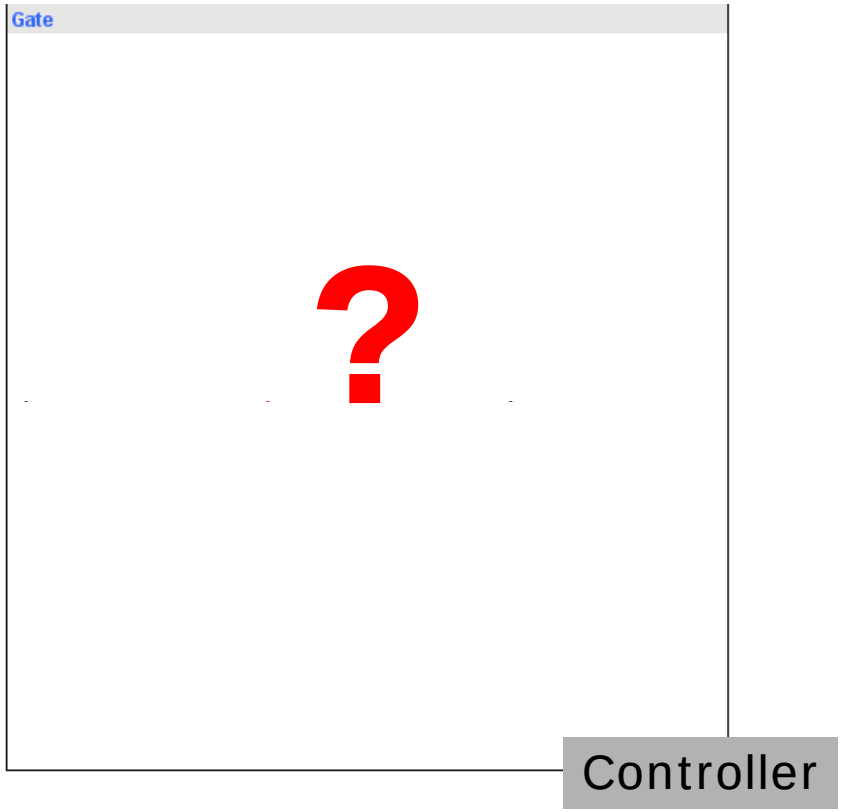


Controller

ϕ: Never two trains at the crossing at the same time

Synthesis (ex Train Gate)



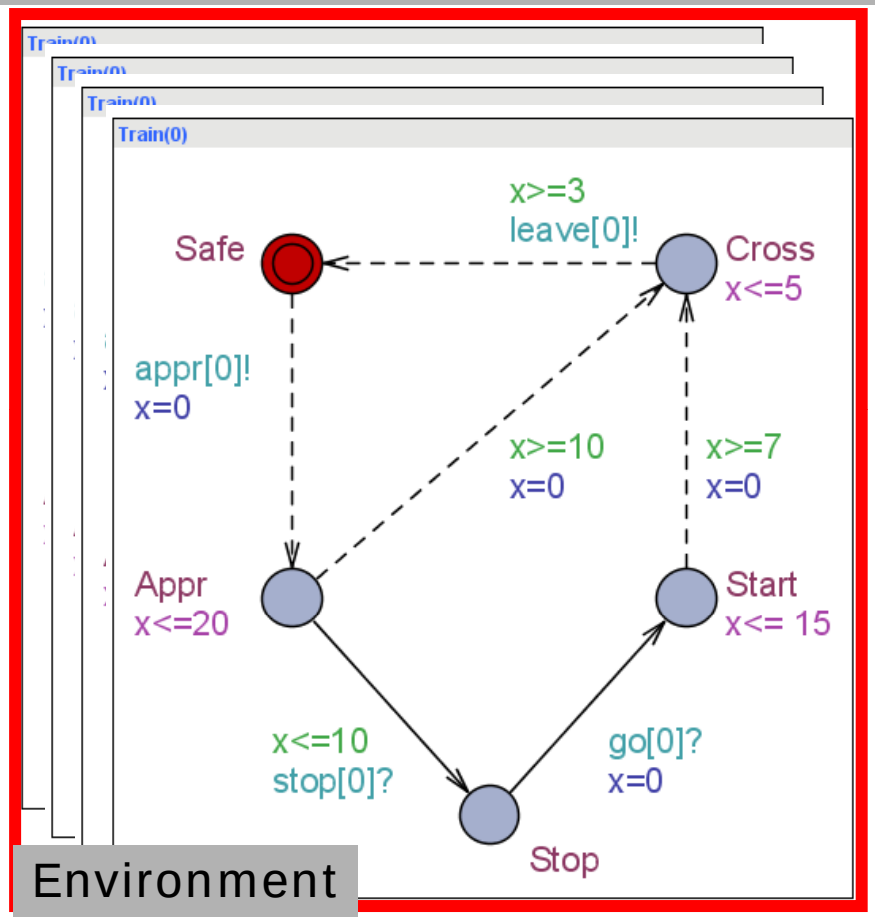


ϕ: Never two trains at the crossing at the same time

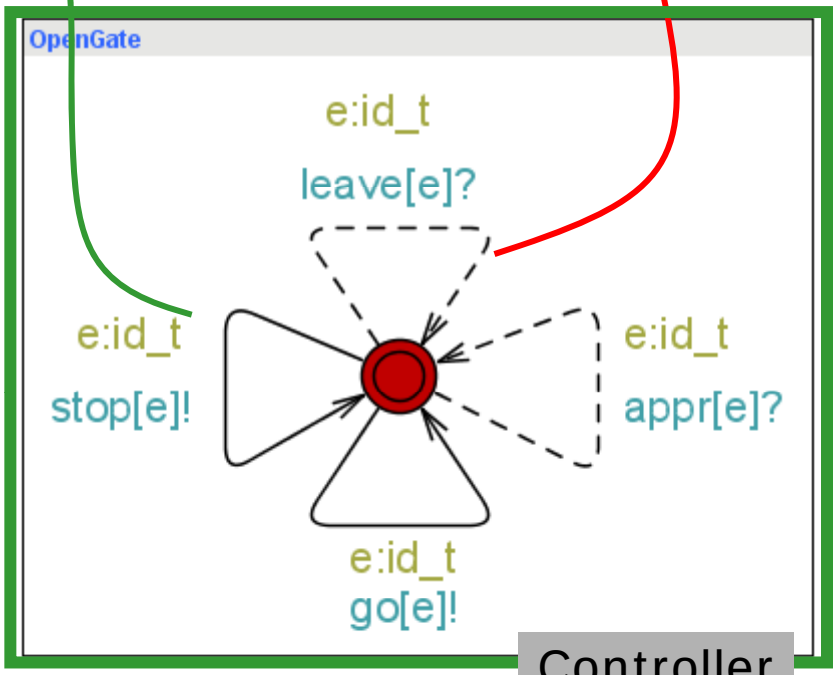
Synthesis

Two Player Game

Controllable Uncontrollable



Environment



Controller

Find strategy for controllable actions st behaviour satisfies ϕ

ϕ: Never two trains at the crossing at the same time

Timed Games

Reachability

Memoryless strategy:

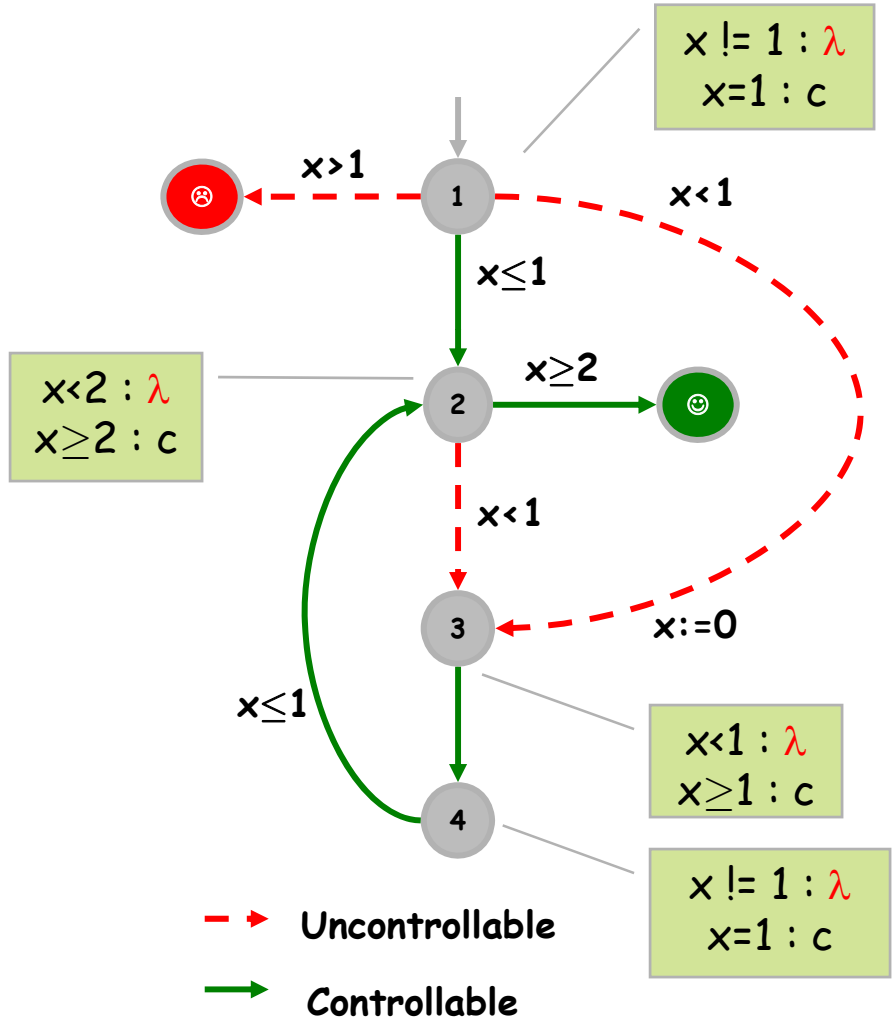
$F : Q \rightarrow E_c \cup \lambda$

Winning Run:

$States(\rho) \cap G \neq \emptyset$

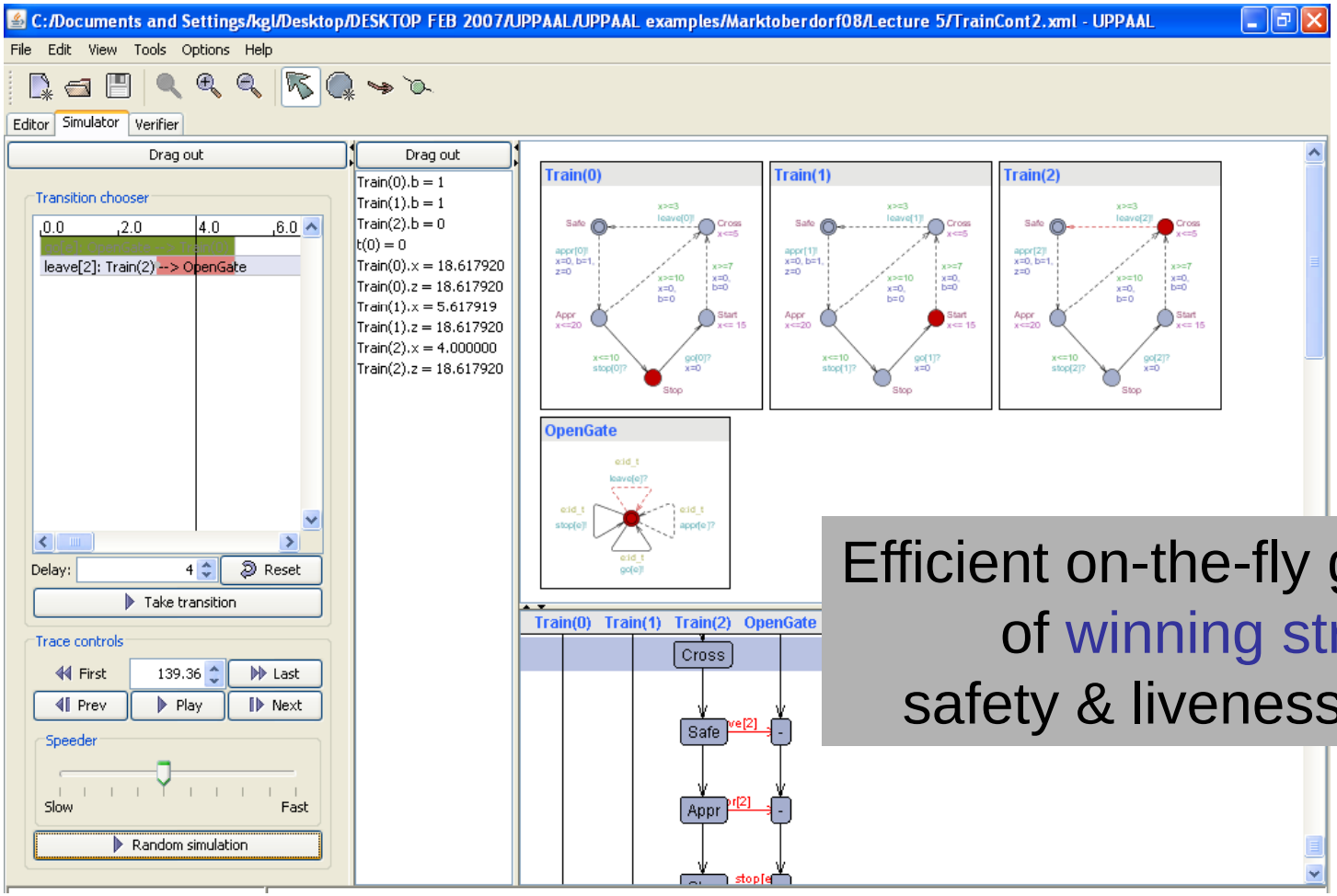
Winning Strategy:

$Runs(F) \subseteq WinRuns$



UPPAAL Tiga

Synthesis of winning strategies for TIMED GAMES

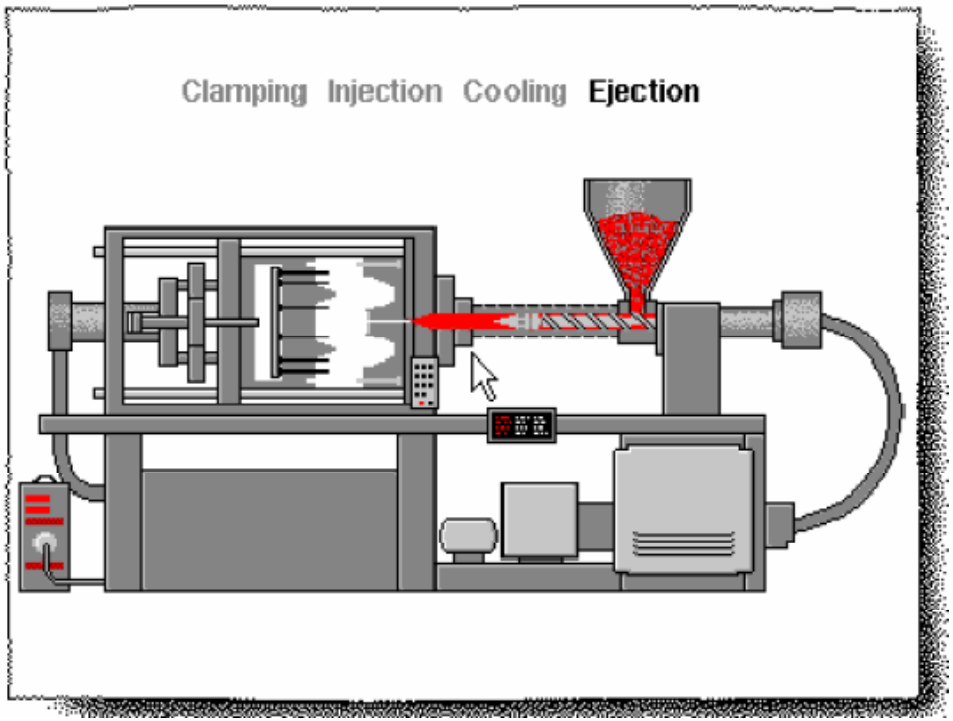


CONCUR05, CAV07, FORMATS07

Efficient on-the-fly generation of winning strategies for safety & liveness objectives

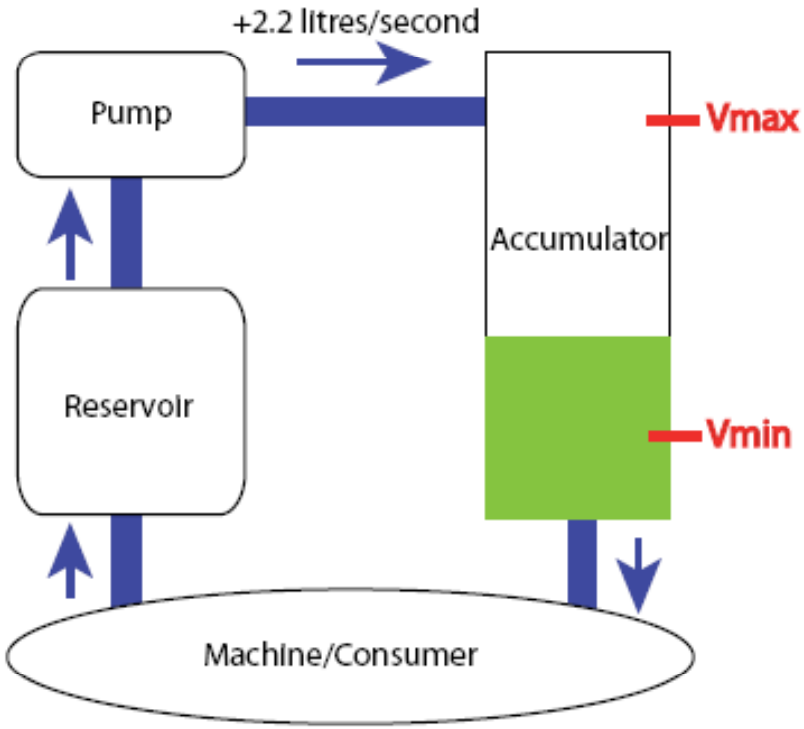
Synthesis of Hydralic Controller

Quasimodo
(Cassez, Jessen Larsen, Rainier, Raskin - HSCC09)



- Tool Chain
  - Synthesis: UPPAAL TIGA
  - Verification: PHAVer
  - Performance: SIMULINK
- 40% improvement of existing solutions..

Oil Pump Control Problem

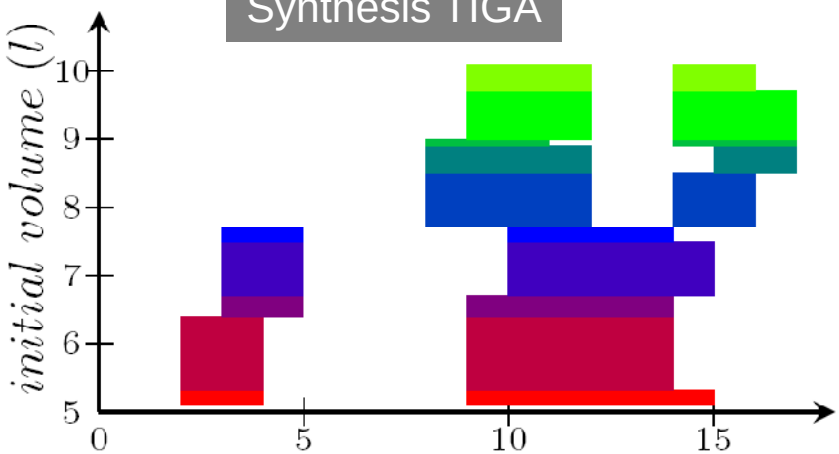


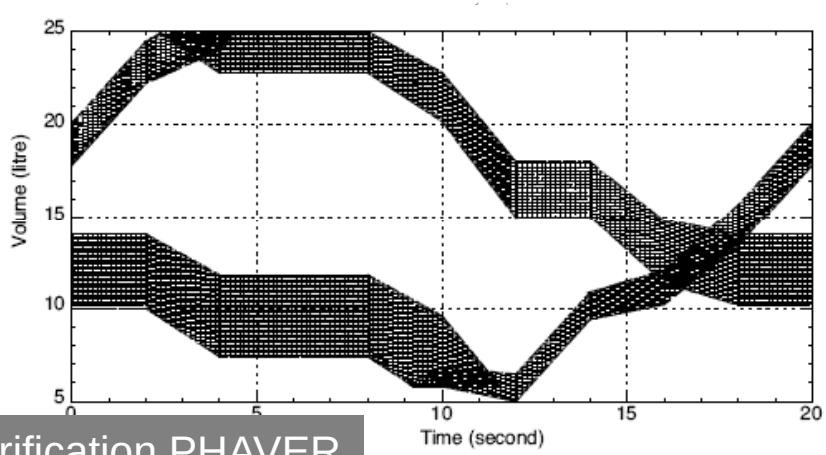
- R1: stay within safe interval [5 , 25]
- R2: minimize average/overall oil volume

$$\int_{t=0}^{t=T} v(t)dt \,/\, T$$

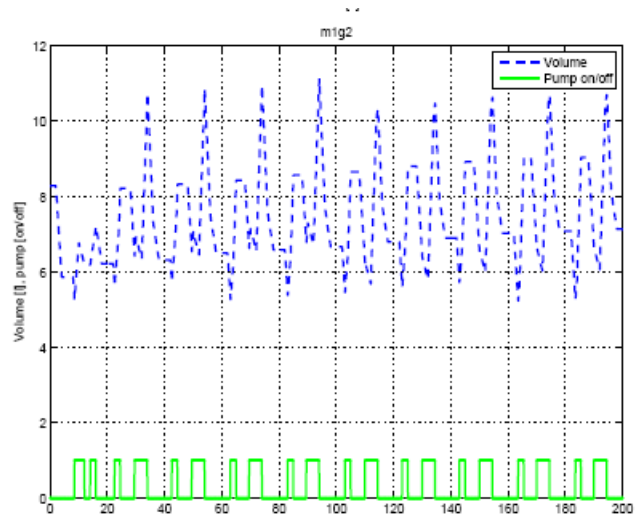
Tool Chain

Synthesis TIGA





Verification PHAVER



Performance Evaluation
SIMULINK

Guaranteed

Correctness

Robustness

with

40% Improvement

Conclusion & Open Problems

Decidability

- Priced Timed Games
  - Reachability & Model Checking
  - Energy-Bounded Prb.
  - Safety
- Probabilistic Priced Timed Automata

Efficiency

- Timed Automata
  - Fully Symbolic (CDD)
  - Static Analy ing of C-code
- Timed Games
  - Partial Obser ty
  - Alternating
  - CEGAR

- Priced Timed Automata
  - Optimal Infinite
- m TA models of controllers
  - From stategies

Usability

- APPLICATIONS
- Live Sequence Charts
- Gantt Chart
- Probabilistic Timed Automata

Thanks for your attention!

www.uppaal.com