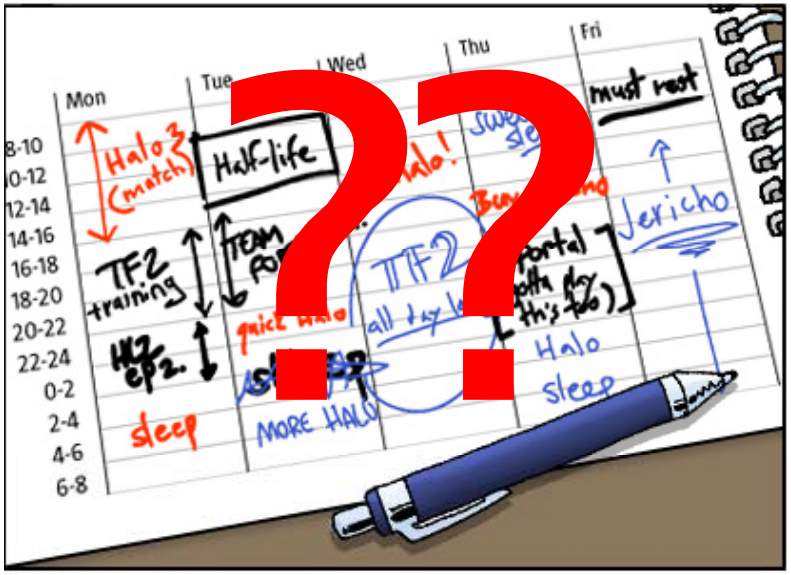
Schedulability & WCET Analysis



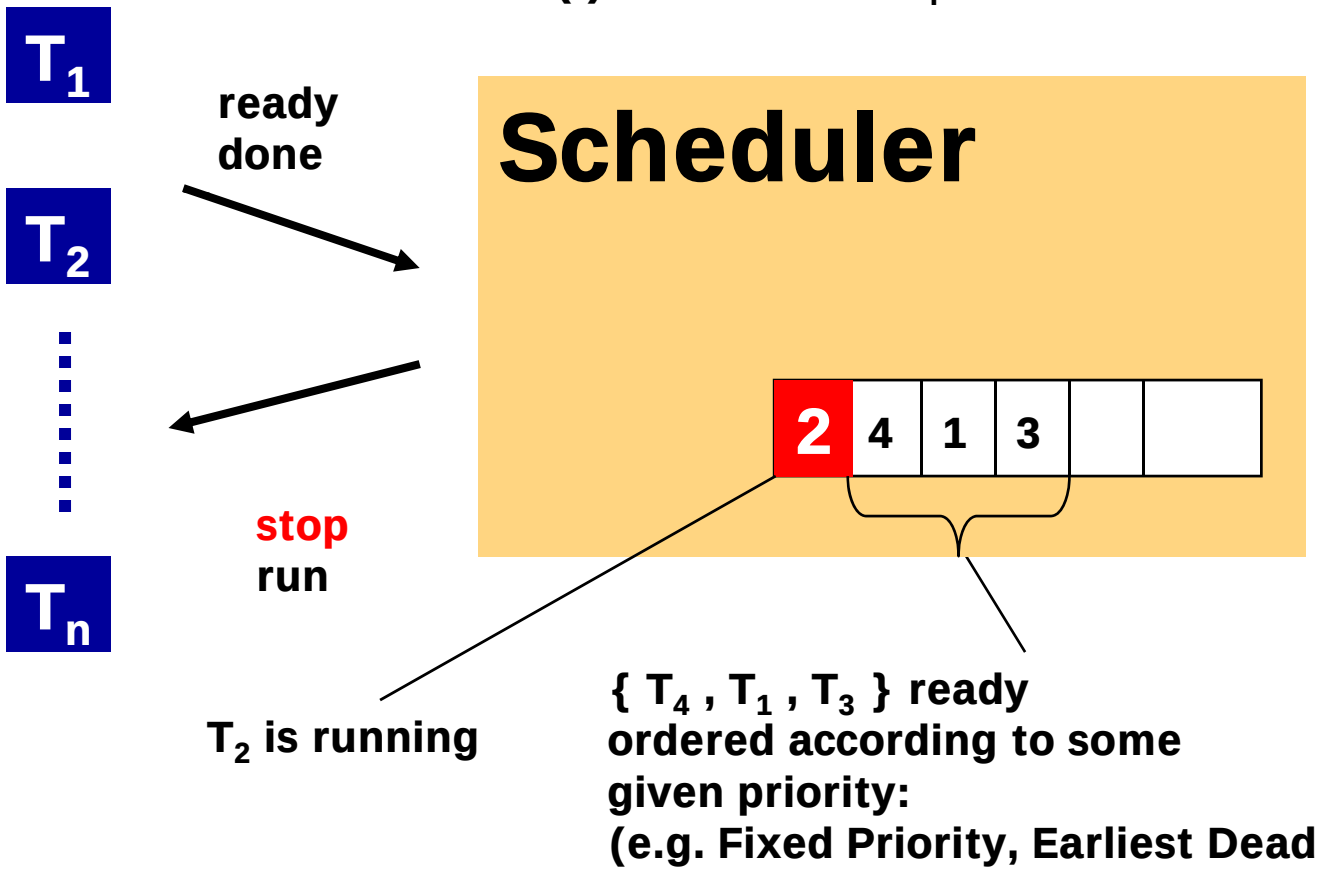
Task Scheduling

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



Classical Scheduling Theory

Utilisation-Based Analysis

- A simple sufficient but not necessary schedulability test exists

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\,(2^{1/N} - 1)$$

$$U \leq 0.69 \text{ as } N \to \infty$$

Where C is WCET and T is period

41

Response Time Equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of tasks with priority higher than task i

Solve by forming a recurrence relationship:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

The set of values $w_i^0, w_i^1, w_i^2, \ldots, w_i^n, \ldots$ is monotonically non decreasing
When $w_i^n = w_i^{n+1}$ the solution to the equation has been found, w_i^0 must not be greater that R_i (e.g. 0 or C_i)

42

Classical WCRT Analysis

Quasimodo

- "Classical" scheduling analysis technique
- *For all tasks i: WCRT$_i \leq$ Deadline$_i$*

$$R_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Blocking times for priority inheritance protocol (BSW):

$$Blocking(i) = \sum_{r=1}^{R} usage(r,i)\,WCET_{CriticalSection}(r)$$

Blocking times for priority ceiling protocol (ASW):

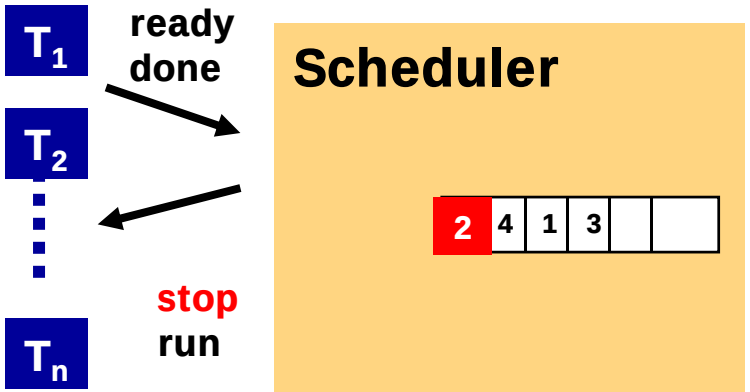
$$Blocking(i) = \max_{r=1}^{R} usage(r,i)\,WCET_{CriticalSection}(r)$$

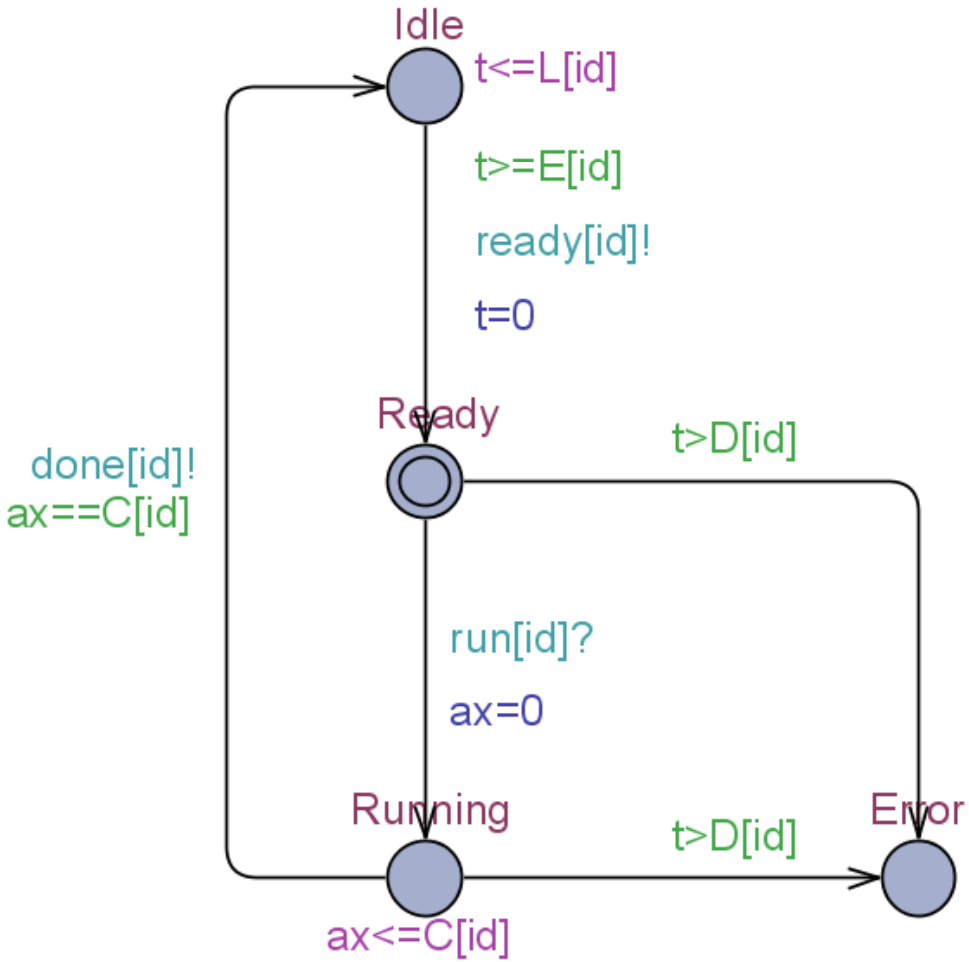
Quasimodo Workshop, Eindhoven, Nov 6, 2009 Page 21

✓ Simple to perform

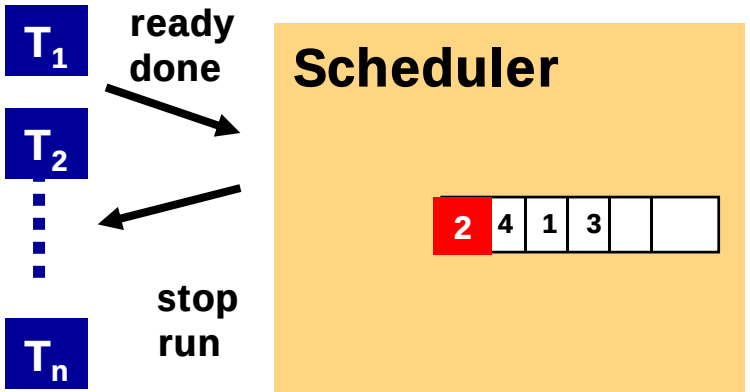
- Overly conservative
- Limited settings
- Single-processor

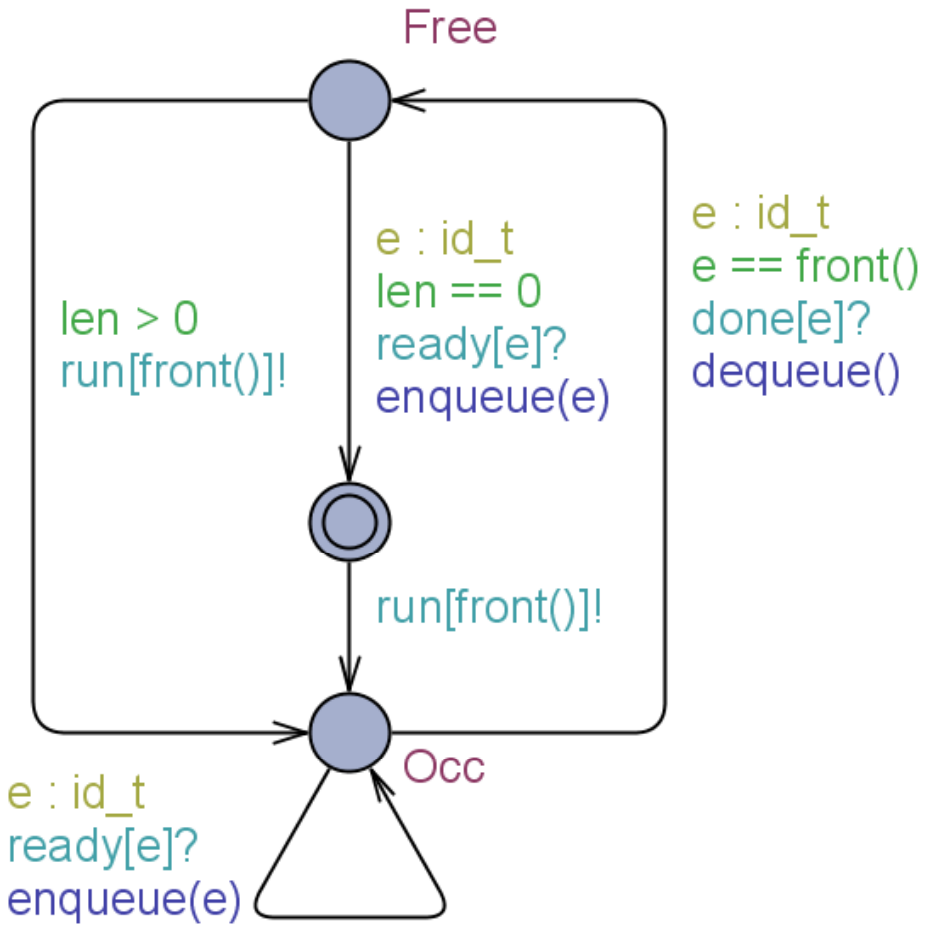
Modeling Task





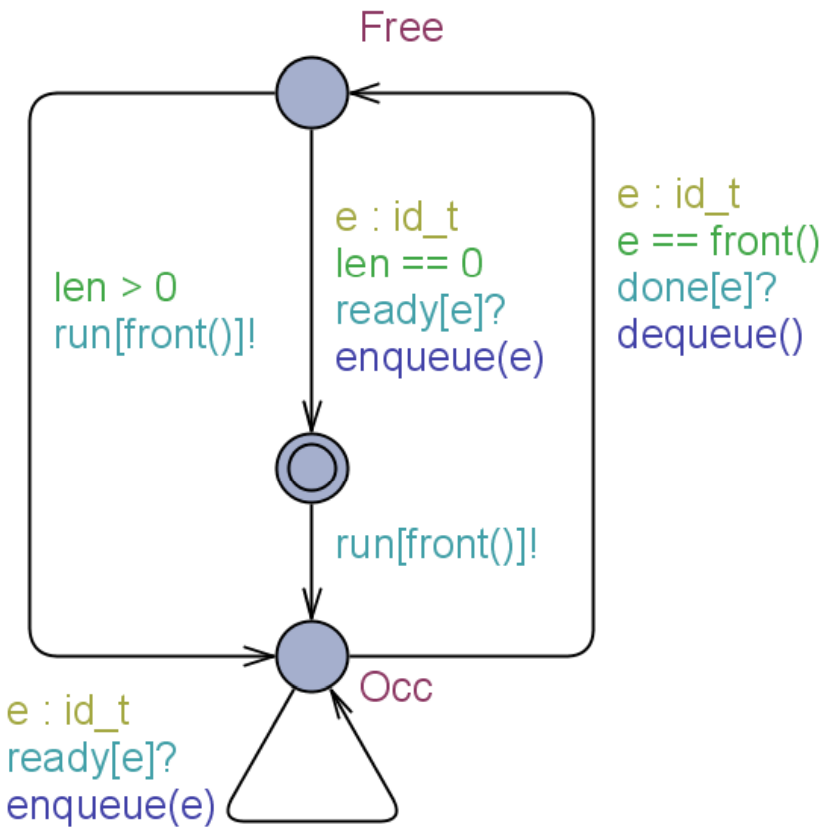
Modeling Scheduler





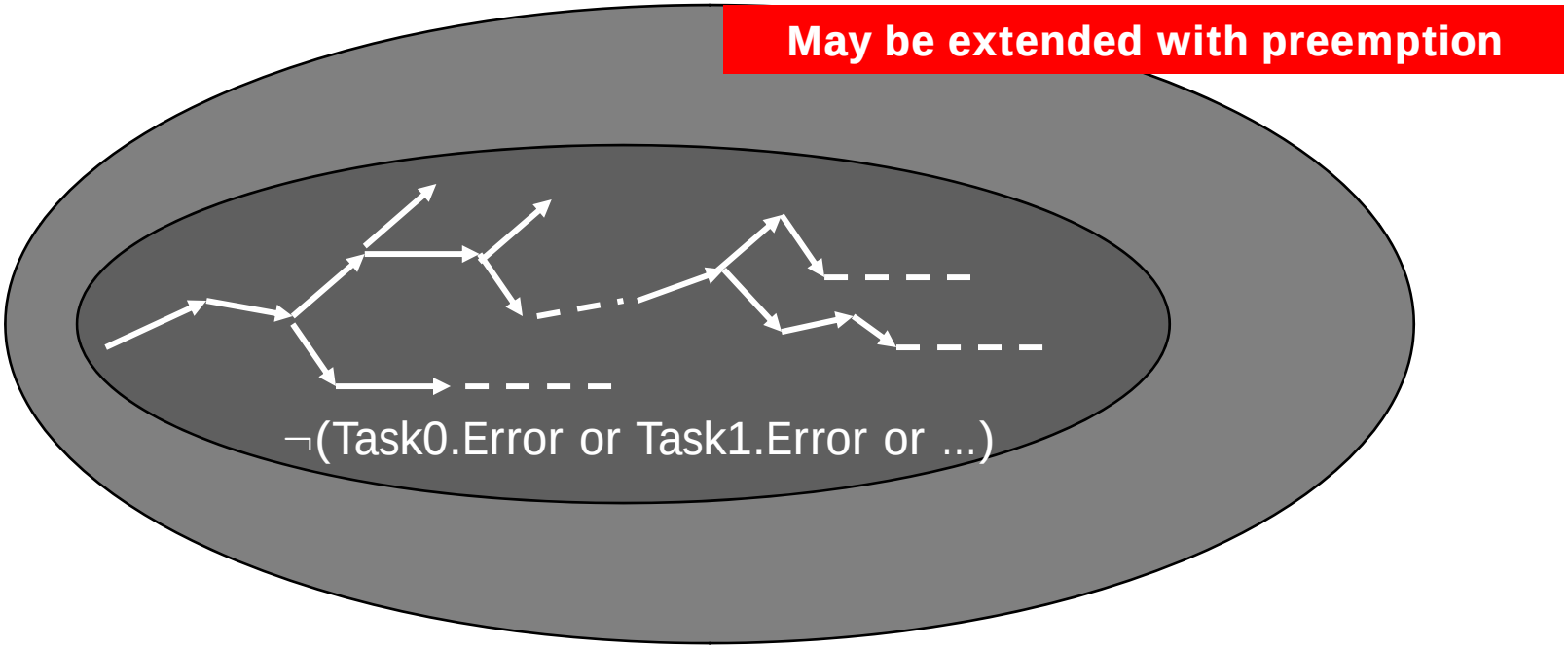
Modeling Queue

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()
{
  .....
```

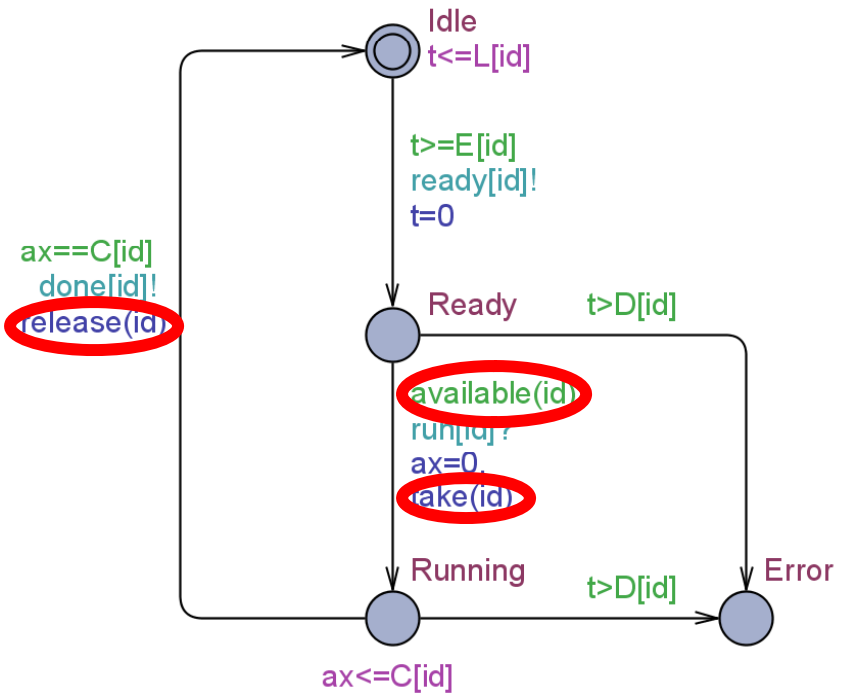
Schedulability = Safety Property



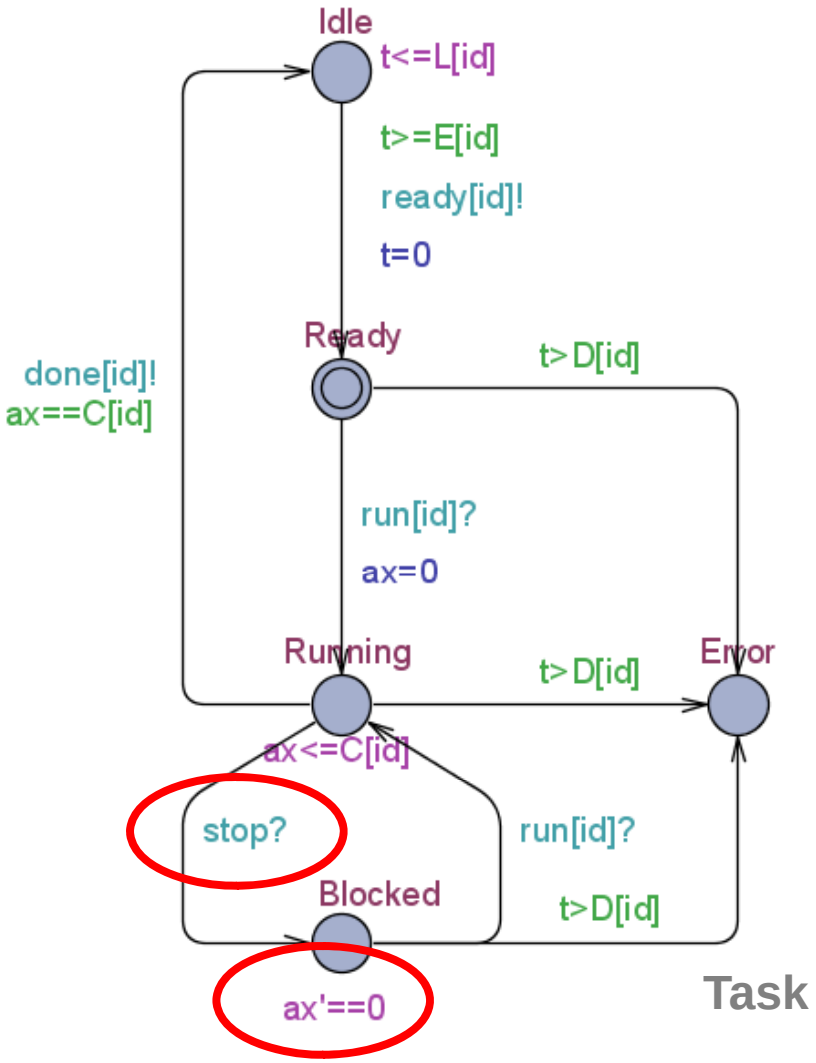
A□ ¬(Task0.Error or Task1.Error or ...)

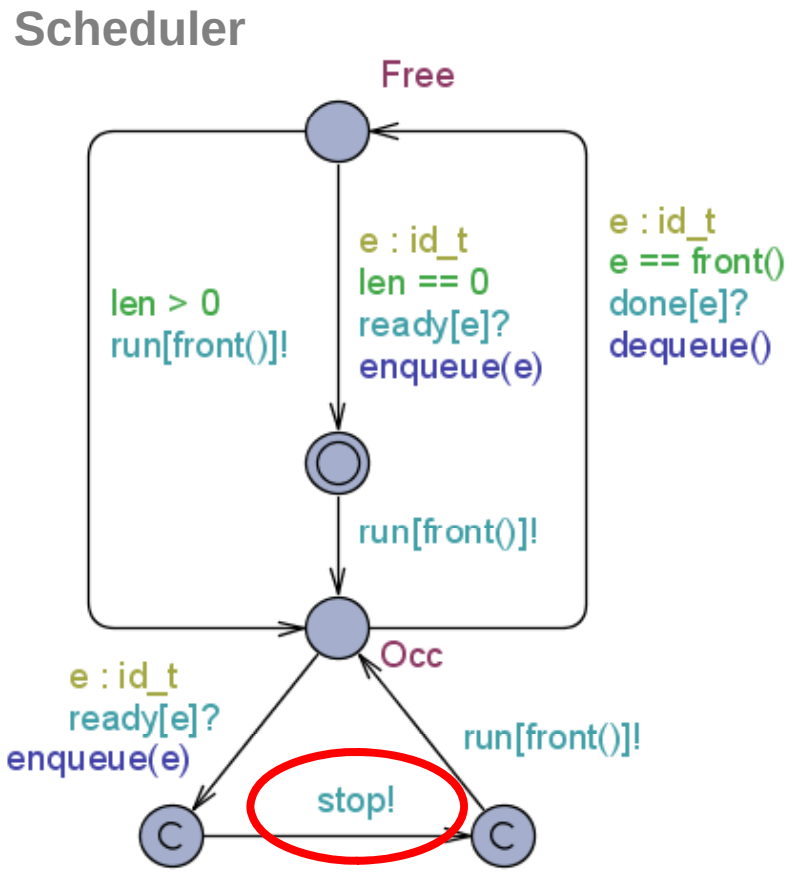
Dealing with Resources

```
bool resource[N];

bool available(id_t id)
{
    return !resource[need[id]];
}

void take(id_t id)
{
    assert(!resource[need[id]]);
    resource[need[id]] = true;
}

void release(id_t id)
{
    assert(resource[need[id]]);
    resource[need[id]] = false;
}
```



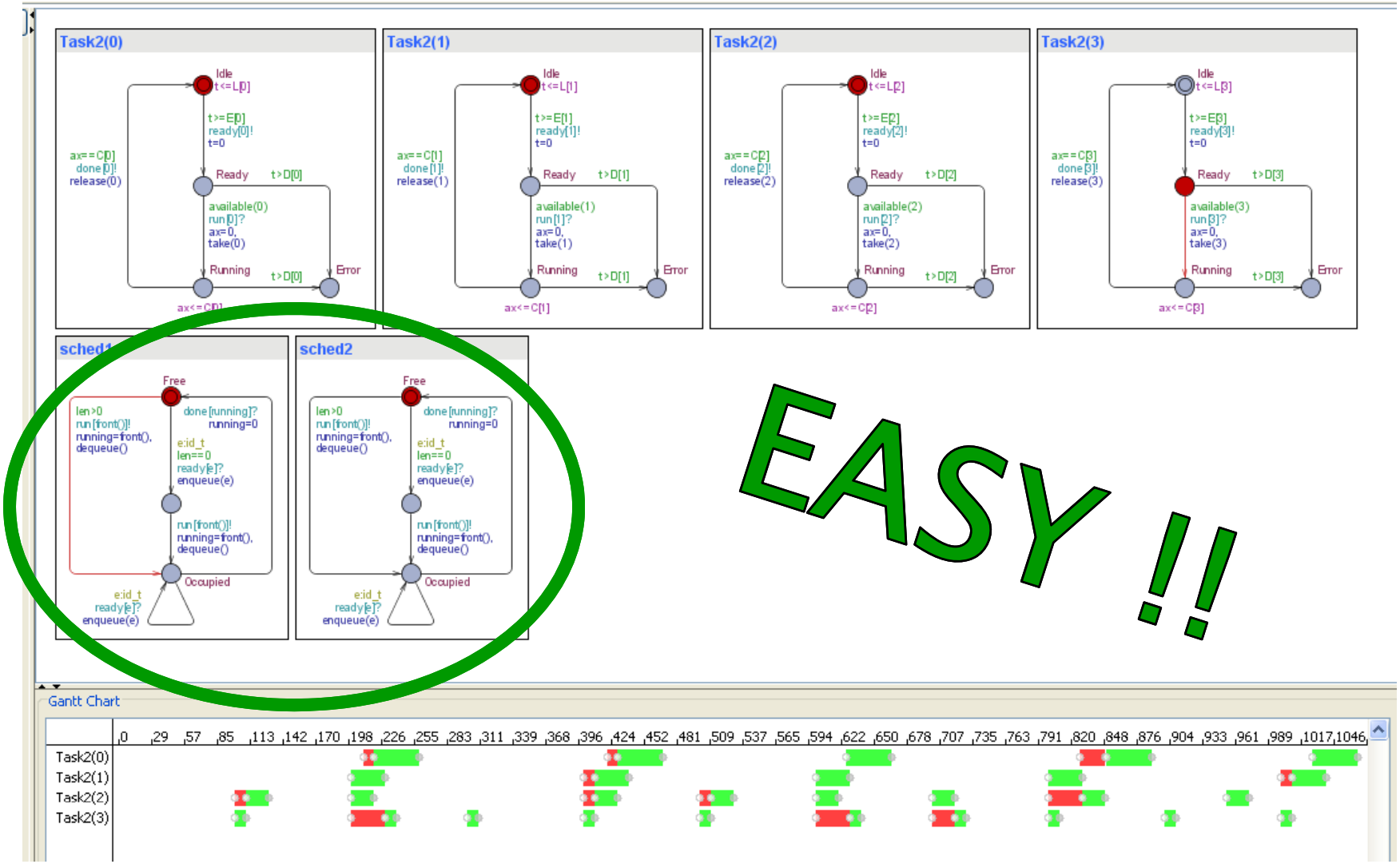
Preemption – Stopwatches!





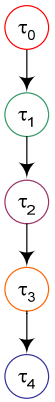
Defeating undecidability ☺

Multi-Processor

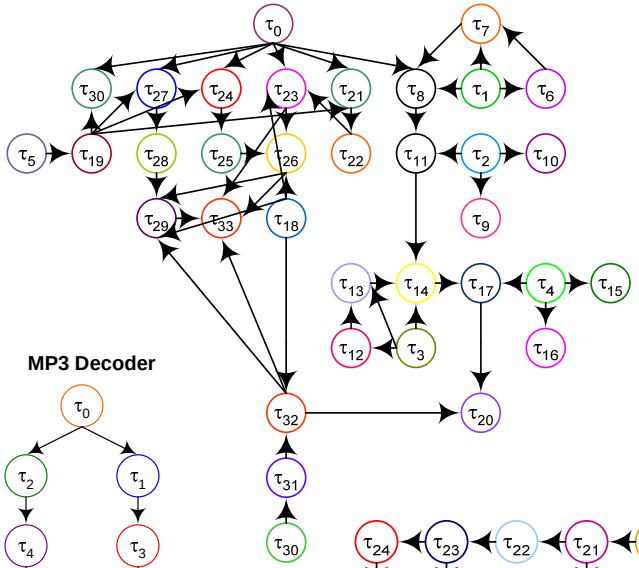


Handling realistic applications?

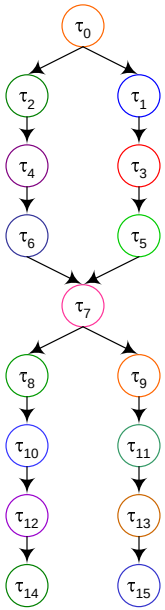
JPEG Encoder



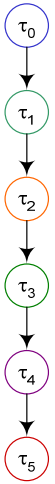
GSM Decoder



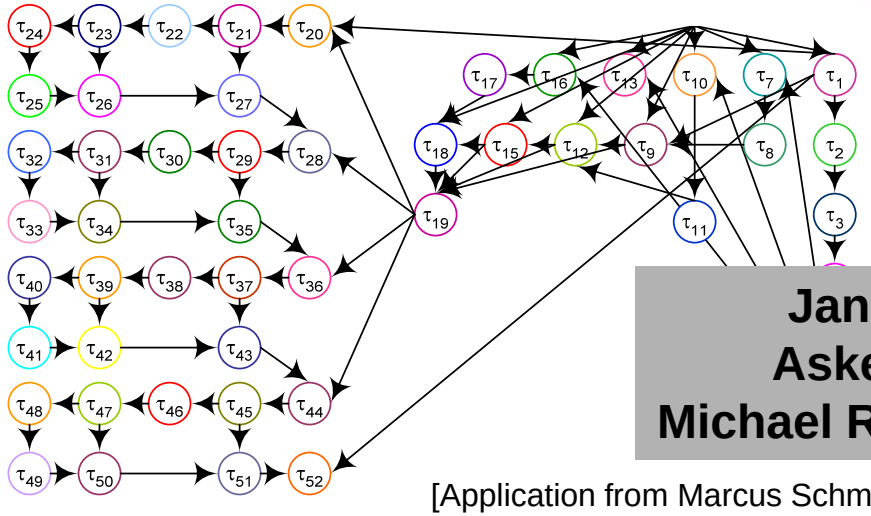
MP3 Decoder



JPEG Decoder



GSM Encoder

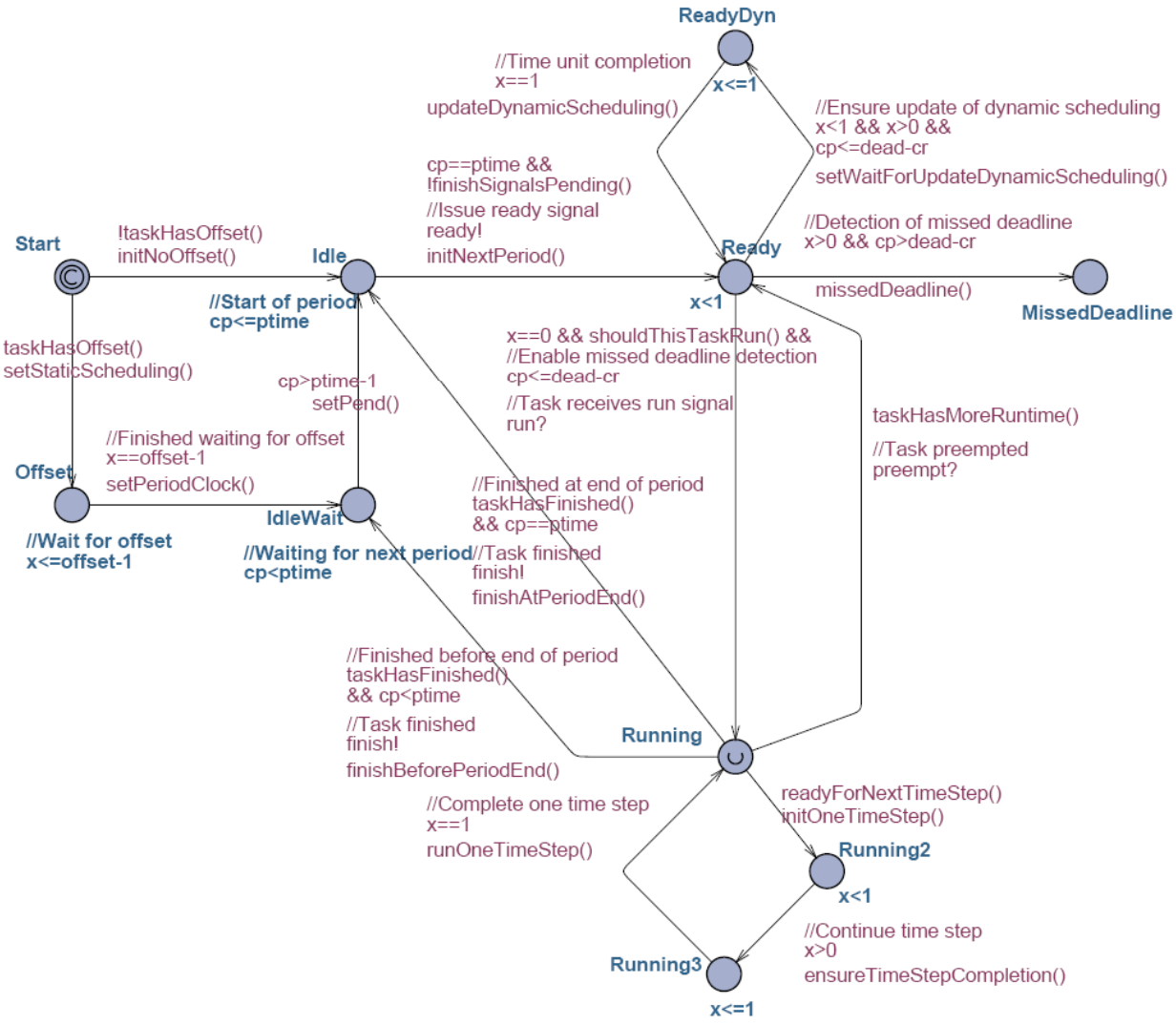


Smart phone:

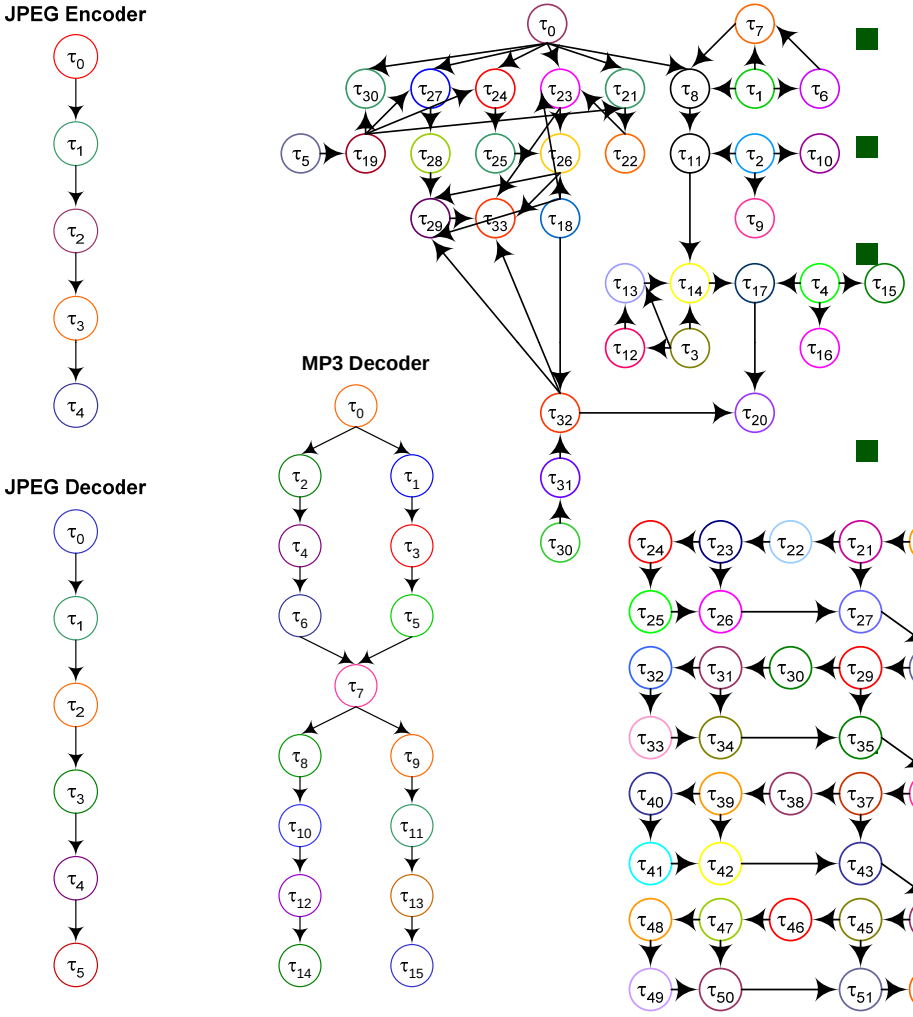
Jan Madsen
Aske Brekling
Michael R. Hansen/ DTU

[Application from Marcus Schmitz, TU Linkoping]

Timed Automata for a task



Smart phone



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

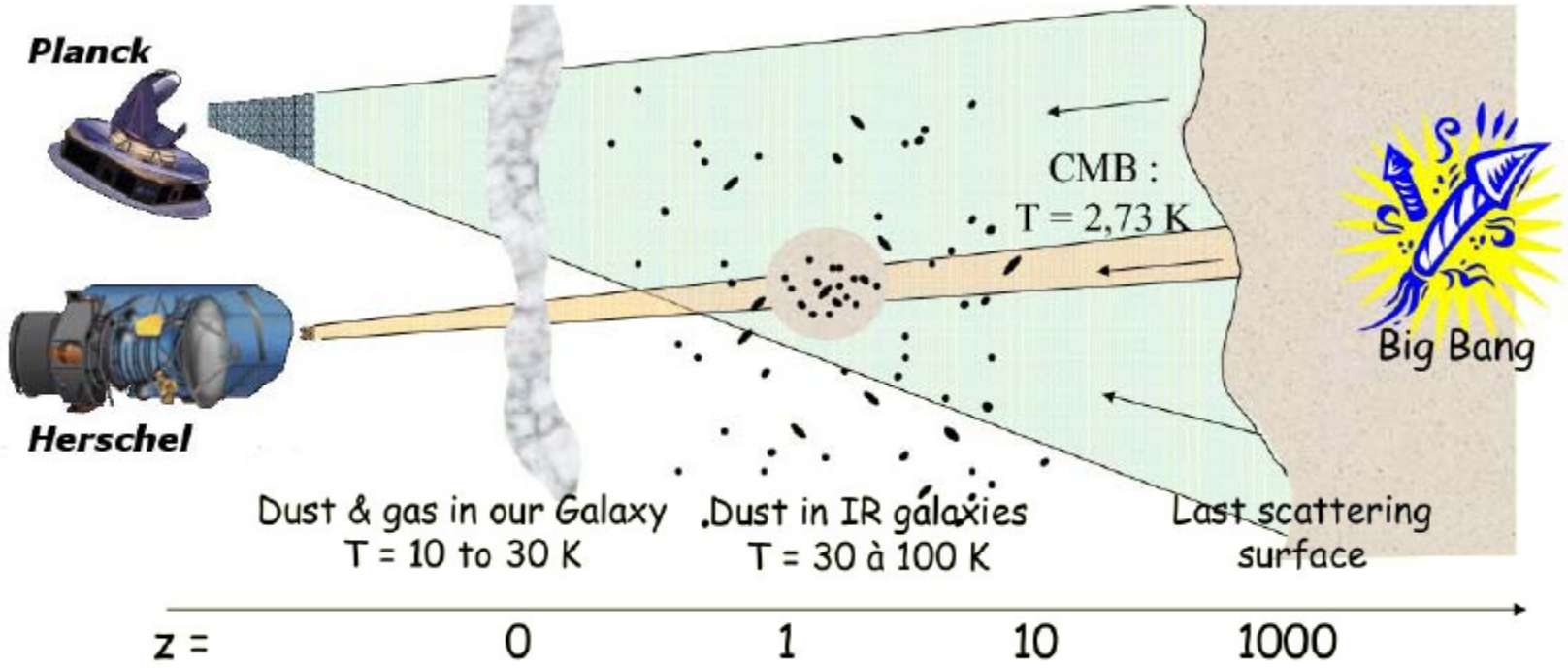
Verified in 1.5 hours!

ESA Missions

TERMA

Quasiomodo

- Solar System, cold dust clouds and cores, star and galaxy formations, cataloging galaxies, gravitational lensing, cosmic microwave background, topology of the universe...

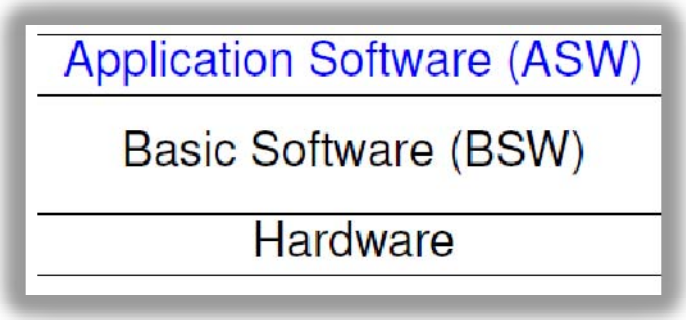


- Terma: Develop software for Attitude and Orbit Control System

Herschel & Planck Satelites

TERMA

- Application software (ASW)
  - built and tested by Terma:
  - does attitude and orbit control, tele-commanding, fault detection isolation and recovery.
- Basic software (BSW)
  - low level communication and scheduling periodic events.
- Real-time operating system (RTEMS)
  - Priority Ceiling for ASW,
  - Priority Inheritance for BSW
- Hardware
  - single processor, a few buses, sensors and act



Requirements:

Software tasks should be schedulable.
CPU utilization should not exceed 50% load

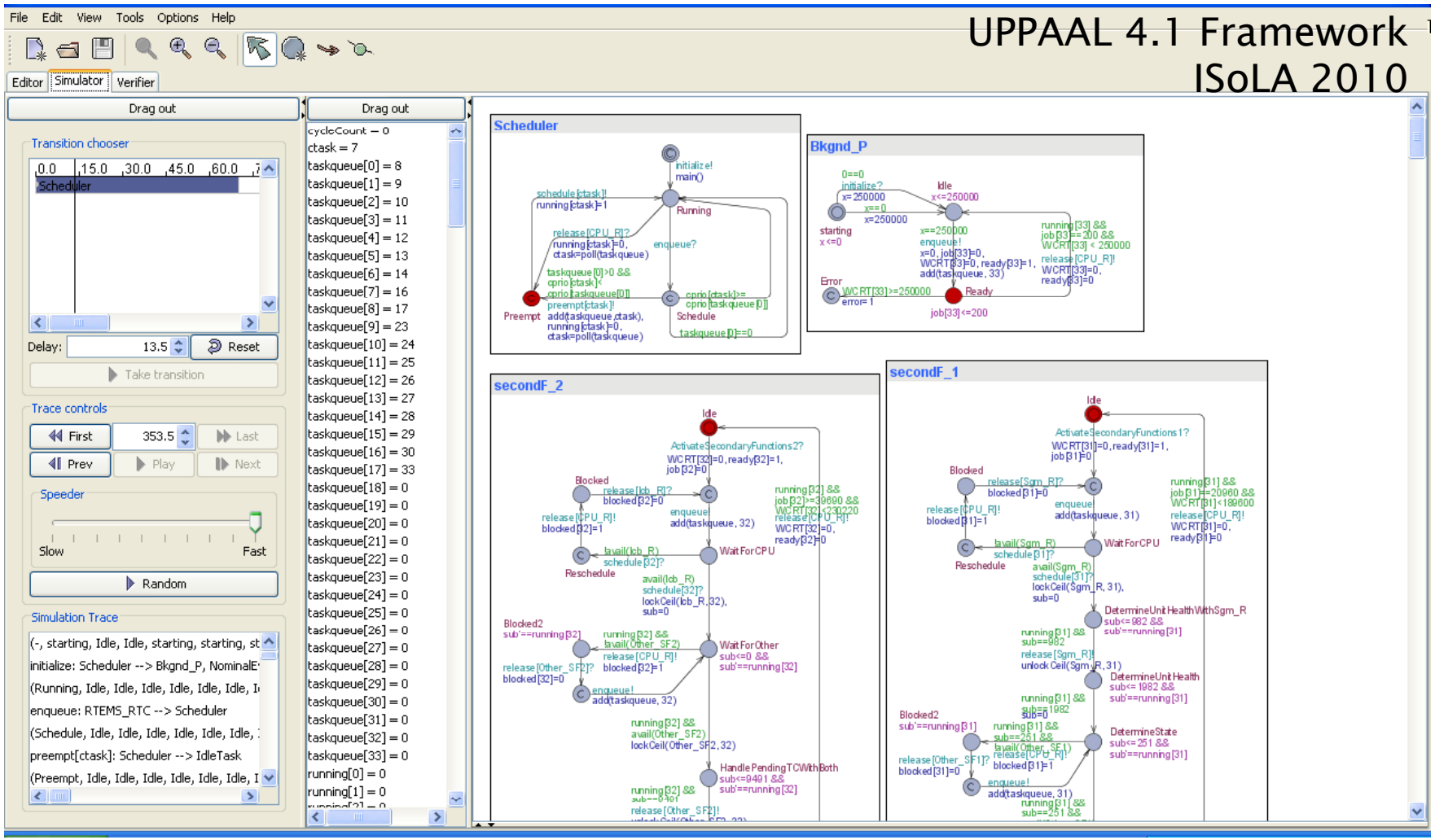
UPPAAL Model

TERMA

- One template for CPU scheduler:
  - maintains a queue of ready tasks
  - schedules tasks with highest priority in the queue
  - reschedule if higher priority task arrives to the queue
- One template per **each** ASW task.
- Two templates for BSW tasks: 1 plain, 1 using resource.
- One template for "idle" task to count CPU utilization.
- WCET is modeled by stopwatches and lower bound.
- WCRT is modeled by stopwatches.
- Deadline is enforced as guard on WCRT.
- System is schedulable if no deadline violated.
- CPU load is $\frac{\text{used time}}{\text{total time}}$.

Modeling in UPPAAL

TERMA



Gantt Chart 1. cycle

TERMA

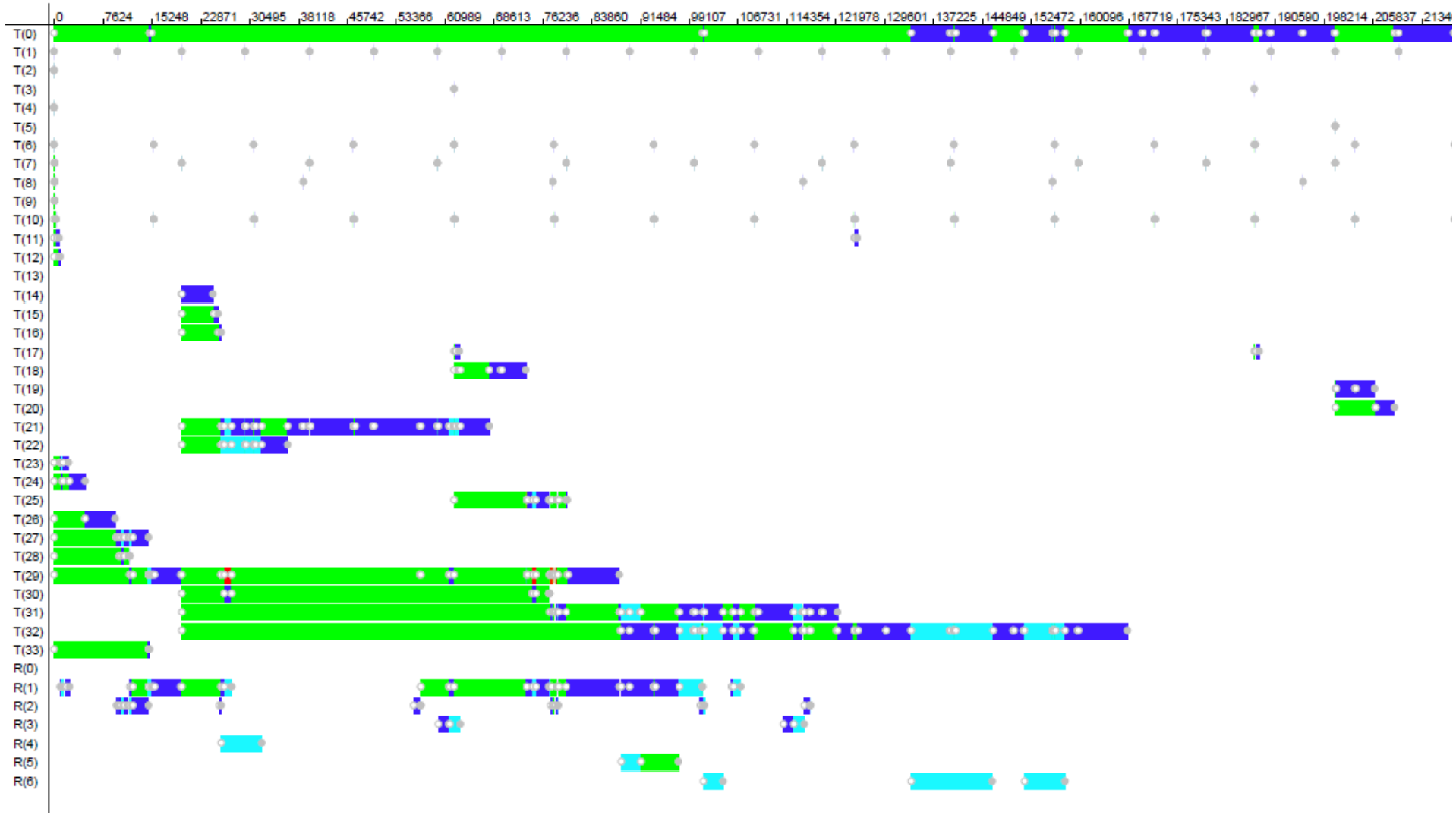


Fig. 11. Gantt chart of a schedule from the first cycle: green means ready, blue means running, cyan means suspended, red means blocked. R stand for resources: CPU_R=0, Icb_R=1, Sgm_R=2, PmReq_R=3, Other_RCS=4, Other_SF1=5, Other_SF2=6.

Blocking & WCRT

TERMA

ID	Task	Specification			Blocking times			WCRT		
		Period	WCET	Deadline	Terma	UPPAAL	Diff	Terma	UPPAAL	Diff
1	RTEMS_RTC	10.000	0.013	1.000	0.035	0	0.035	0.050	0.013	0.037
2	AswSync_SyncPulseIsr	250.000	0.070	1.000	0.035	0	0.035	0.120	0.083	0.037
3	Hk_SamplerIsr	125.000	0.070	1.000	0.035	0	0.035	0.120	0.070	0.050
4	SwCyc_CycStartIsr	250.000	0.200	1.000	0.035	0	0.035	0.320	0.103	0.217
5	SwCyc_CycEndIsr	250.000	0.100	1.000	0.035	0	0.035	0.220	0.113	0.107
6	Rt1553_Isr	15.625	0.070	1.000	0.035	0	0.035	0.290	0.173	0.117
7	Bc1553_Isr	20.000	0.070	1.000	0.035	0	0.035	0.360	0.243	0.117
8	Spw_Isr	39.000	0.070	2.000	0.035	0	0.035	0.430	0.313	0.117
9	Obdh_Isr	250.000	0.070	2.000	0.035	0	0.035	0.500	0.383	0.117
10	RtSdb_P_1	15.625	0.150	15.625	3.650	0	3.650	4.330	0.533	3.797
11	RtSdb_P_2	125.000	0.400	15.625	3.650	0	3.650	4.870	0.933	3.937
12	RtSdb_P_3	250.000	0.170	15.625	3.650	0	3.650	5.110	1.103	4.007
14	**FdirEvents**	250.000	5.000	230.220	0.720	0	0.720	7.180	5.153	2.027
15	**NominalEvents_1**	250.000	0.720	230.220	0.720	0	0.720	7.900	5.873	2.027
16	**MainCycle**	250.000	0.400	230.220	0.720	0	0.720	8.370	6.273	2.097
17	HkSampler_P_2	125.000	0.500	62.500	3.650	0	3.650	11.960	5.380	6.580
18	HkSampler_P_1	250.000	6.000	62.500	3.650	0	3.650	18.460	11.615	6.845
19	Acb_P	250.000	6.000	50.000	3.650	0	3.650	24.680	6.473	18.207
20	IoCyc_P	250.000	3.000	50.000	3.650	0	3.650	27.820	9.473	18.347
21	**PrimaryF**	250.000	34.050	59.600	5.770	0.966	4.804	65.470	54.115	11.355
22	**RCSControlF**	250.000	4.070	239.600	12.120	0	12.120	76.040	53.994	22.046
23	Obt_P	1000.000	1.100	100.000	9.630	0	9.630	74.720	2.503	72.217
24	Hk_P	250.000	2.750	250.000	1.035	0	1.035	6.800	4.953	1.847
25	StsMon_P	250.000	3.300	125.000	16.070	0.822	15.248	85.050	17.863	67.187
26	TmGen_P	250.000	4.860	250.000	4.260	0	4.260	77.650	9.813	67.837
27	Sgm_P	250.000	4.020	250.000	1.040	0	1.040	18.680	14.796	3.884
28	TcRouter_P	250.000	0.500	250.000	1.035	0	1.035	19.310	11.896	7.414
29	Cmd_P	250.000	14.000	250.000	26.110	1.262	24.848	114.920	94.346	20.574
30	**NominalEvents_2**	250.000	1.780	230.220	12.480	0	12.480	102.760	65.177	37.583
31	**SecondaryF_1**	250.000	20.960	189.600	27.650	0	27.650	141.550	110.666	30.884
32	**SecondaryF_2**	250.000	39.690	230.220	48.450	0	48.450	204.050	154.556	49.494
33	Bkgnd_P	250.000	0.200	250.000	0.000	0	0.000	154.090	15.046	139.044

Marius Micusionis

Effort and Utilization

TERMA

cycle limit	Uppaal resources CPU, s	Mem, KB	States, #	Herschel CPU utilization Idle, μs	Used, μs	Global, μs	Sum, μs	Used, %
1	465.2	60288	173456	91225	160015	250000	251240	0.640060
2	470.1	59536	174234	182380	318790	500000	501170	0.637580
3	461.0	58656	175228	273535	477705	750000	751240	0.636940
4	474.5	58792	176266	363590	636480	1000000	1000070	0.636480
6	474.6	58796	178432	545900	955270	1500000	1501170	0.636847
8	912.3	58856	352365	727110	1272960	2000000	2000070	0.636480
13	507.7	58796	186091	1181855	2069385	3250000	3251240	0.636734
16	1759.0	58728	704551	1454220	2545850	4000000	4000070	0.636463
26	541.9	58112	200364	2363640	4137530	6500000	6501170	0.636543
32	3484.0	75520	1408943	2908370	5091700	8000000	8000070	0.636463
39	583.5	74568	214657	3545425	6205745	9750000	9751170	0.636487
64	7030.0	91776	2817704	5816740	10183330	16000000	16000070	0.636458
78	652.2	74768	257582	7089680	12411420	19500000	19501100	0.636483
128	14149.4	141448	5635227	11633480	20366590	32000000	32000070	0.636456
156	789.4	91204	343402	14178260	24821740	39000000	39000000	0.636455
256	23219.4	224440	11270279	23266890	40733180	64000000	64000070	0.636456
312	1824.6	124892	686788	28356520	49643480	78000000	78000000	0.636455
512	49202.2	390428	22540388	46533780	81466290	128000000	128000070	0.636455
624	3734.7	207728	1373560	56713040	99286960	156000000	156000000	0.636455

Marius Micusionis

Safety Critical Java

SARTS: Schedulability Analysis

With

Bent Thomsen, Anders P. Ravn,
Thomas Bøgholm, Henrik Kragh-Hansen,
Petur Olsen, Rene R. Hansen,
Lone Leth Thomsen, Hans Søndergaard,

Java Object – Regional IKT Korridor

- Productivity of a programmer is increased with up to 700 % by changing from C/C++ to Java !
- Number of well-educated Java programmers increasing !

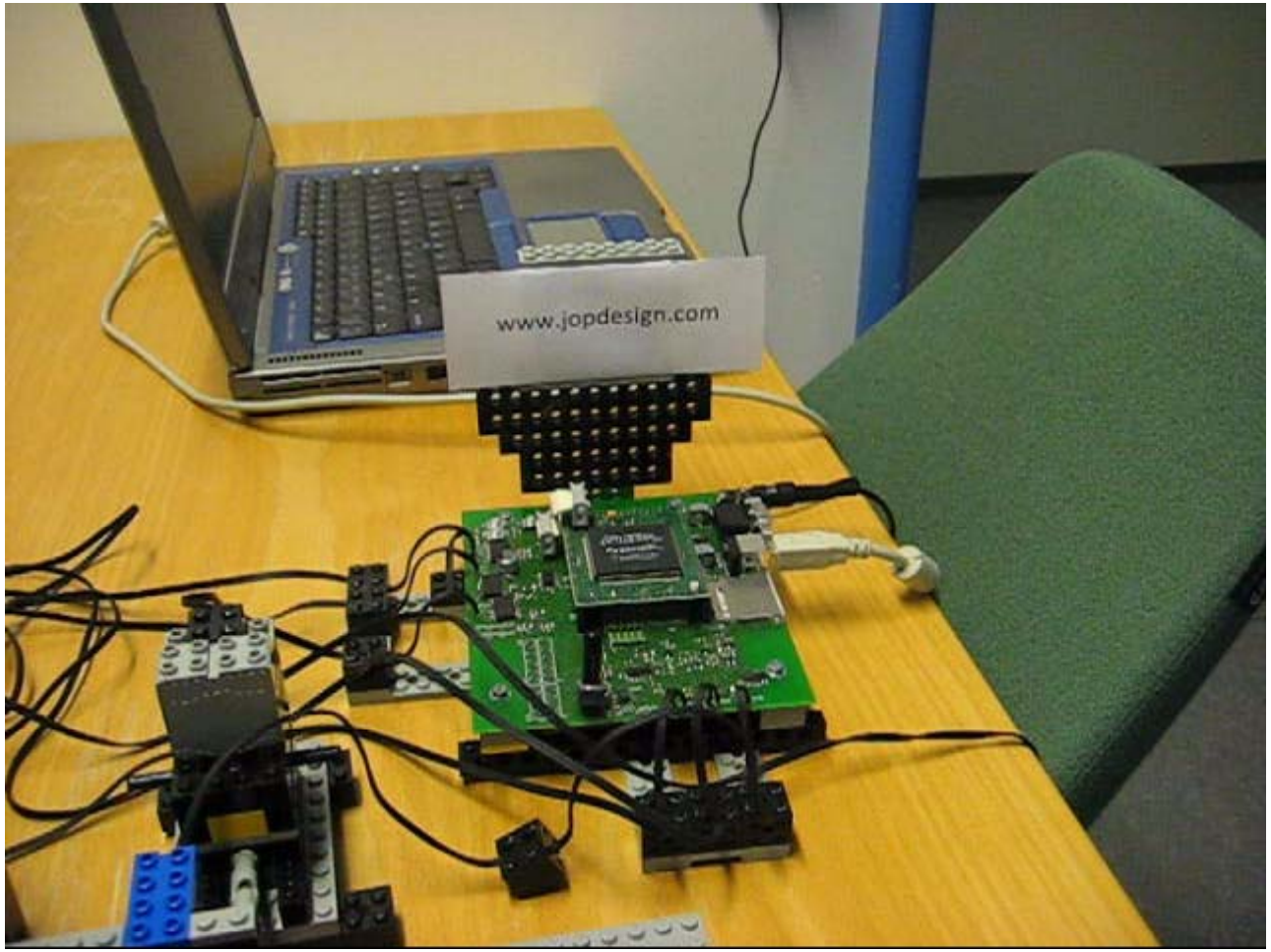
- Java for hard real-time systems ?
- Java and C/Assembler legacy code ?

- Emerging new profiles and hardware implementations !
- Eclipse framework !

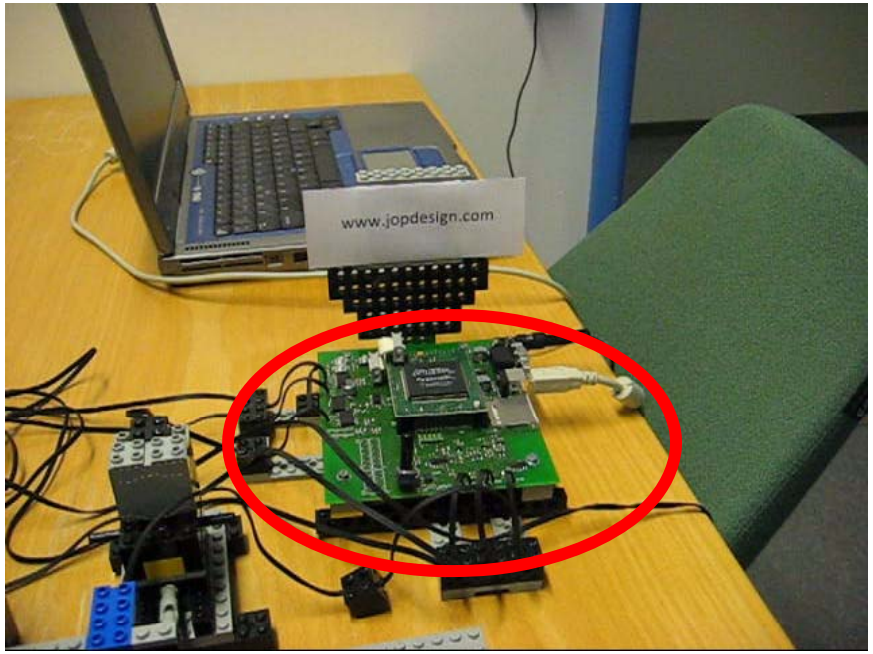
- Center for Embedded Software Systems
- Vitus Bering Denmark
- Polycom (Kirk Telecom A/S)
- Wirtek A/S
- Mechatronic Brick ApS
- Aalborg Industries A/S
- Prevas A/S
- Teknologisk Institut

- Tekkva Consult (project coordinator).

A Safety Critical System



Hardware

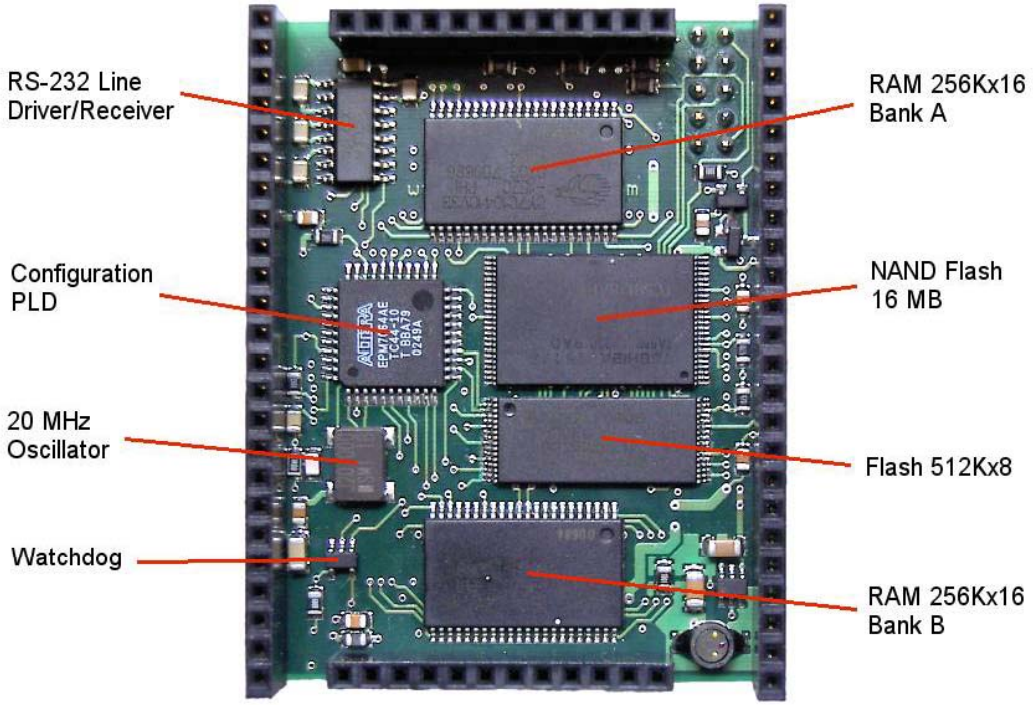


- JOP (Java Optimized Processor)
- Native execution of Java Bytecode
- Bytecode implemented in Microcode
- Avoid unpredictable data-cache
- Time predictable
- Developed new method and stack cache
- Implemented in FPGA

Java Optimizing Processor

Martin Schöberl
University of Tech., Vienna

FPGA



Safety Critical Java

```
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);
```

Min interarrival

Deadline

IMPLEMENTATIONS of SC Java

On JOP and Ajile aJ-100

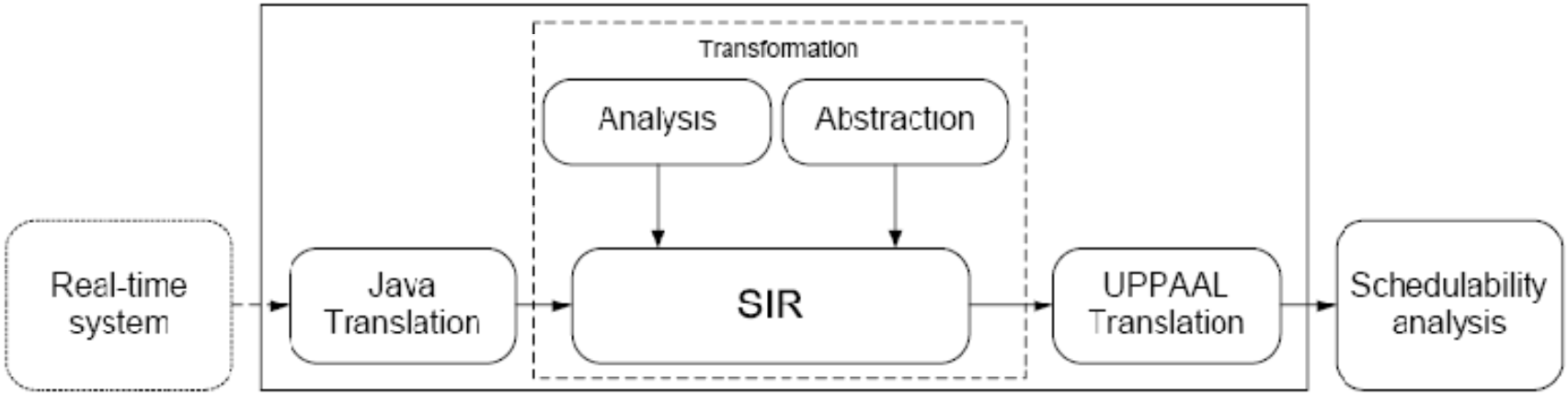
Use existing schedulers and threads

On Mechatronic Brick and Polycom (Kirk)

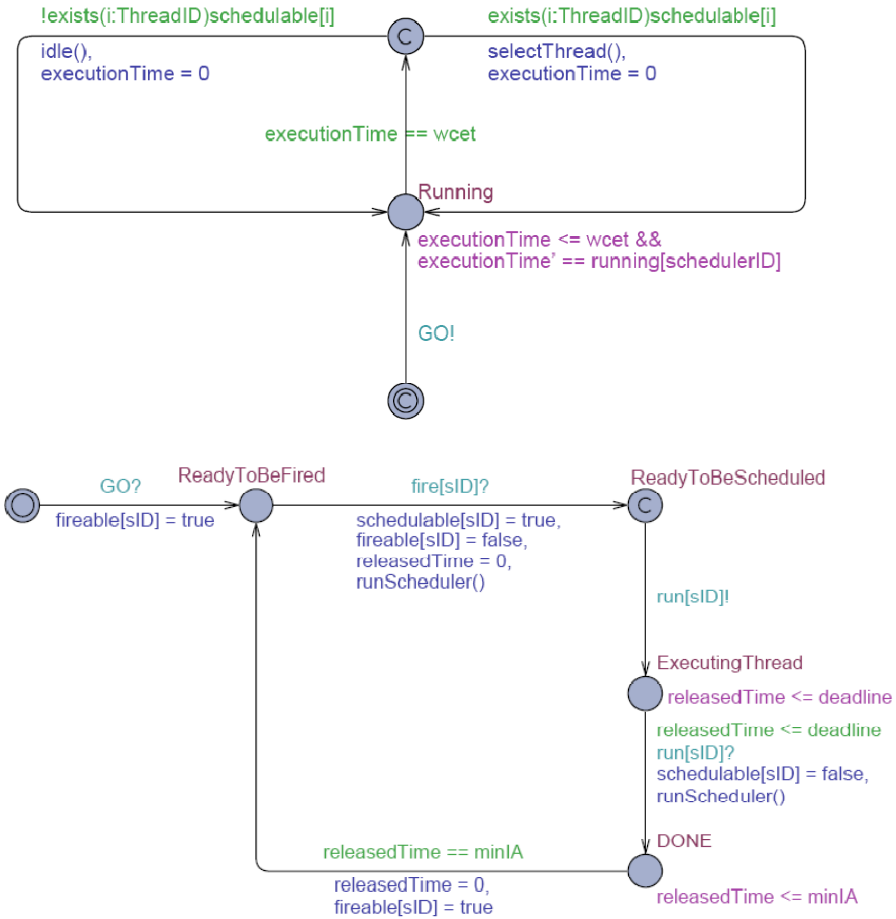
Currently experimenting with JamVM

- Periodic Threads
- Sporadic Threads
- RunTimeSystem
- Relative Time
- Immortal and Raw Memory
- Preemptive FP Scheduling
- Priority Ceiling

SARTS – Overview

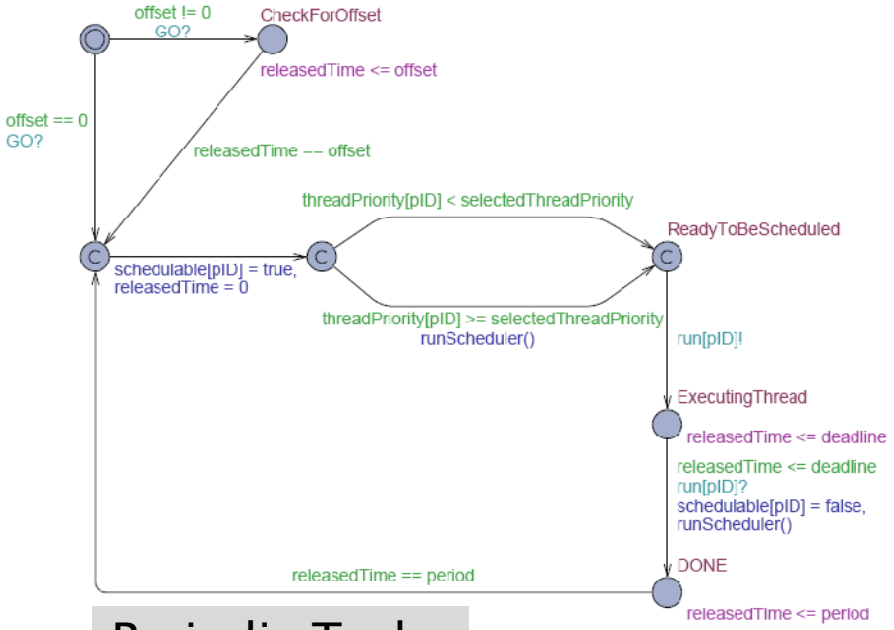


SARTS – TA Templates



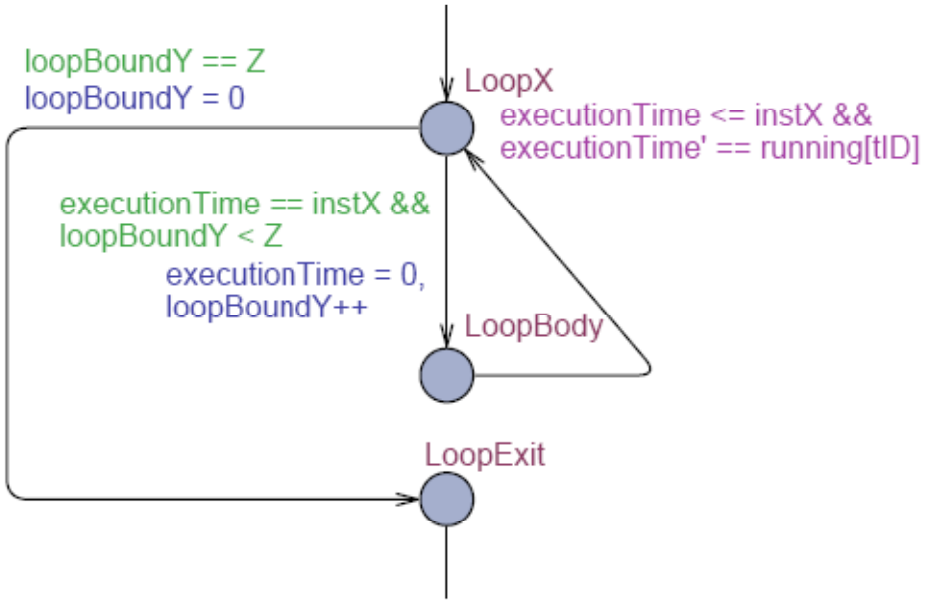
Sporadic Tasks

TA for RM Scheduler



Periodic Tasks

SARTS – TA Templates



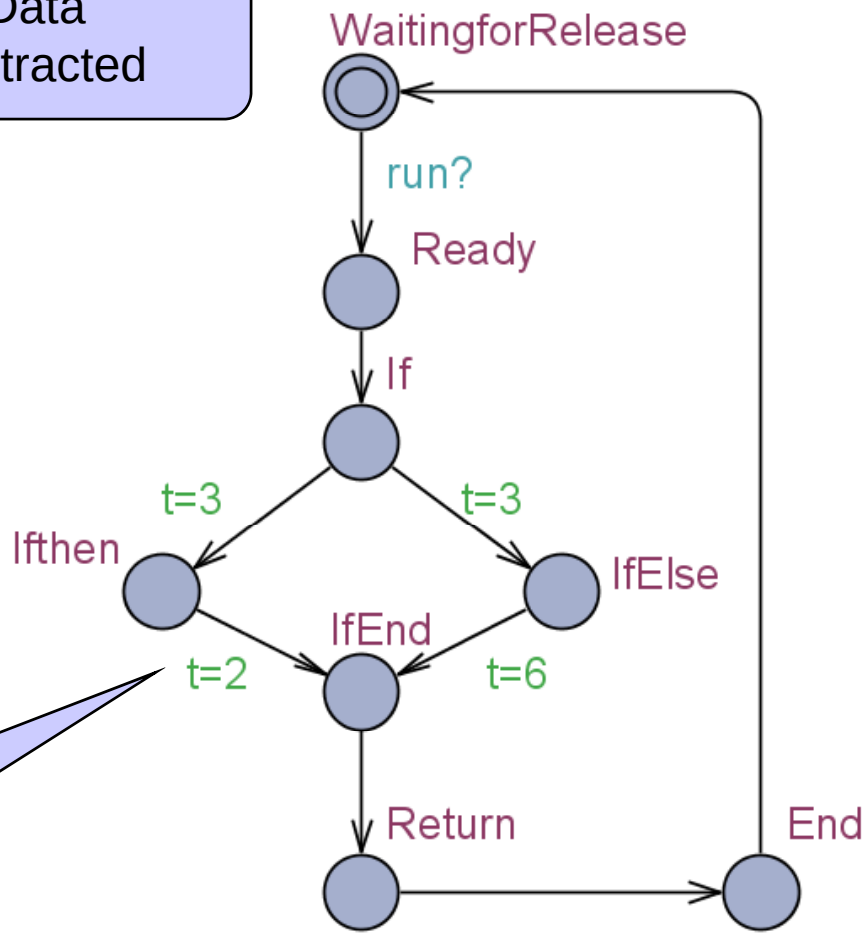
- Translation of Basic Blocks into states and transitions
- Patterns for:
  - Loops
  - Monitor statements
  - If statements
  - Method invoke
  - Sporadic task release

Byte code – Timed Automata

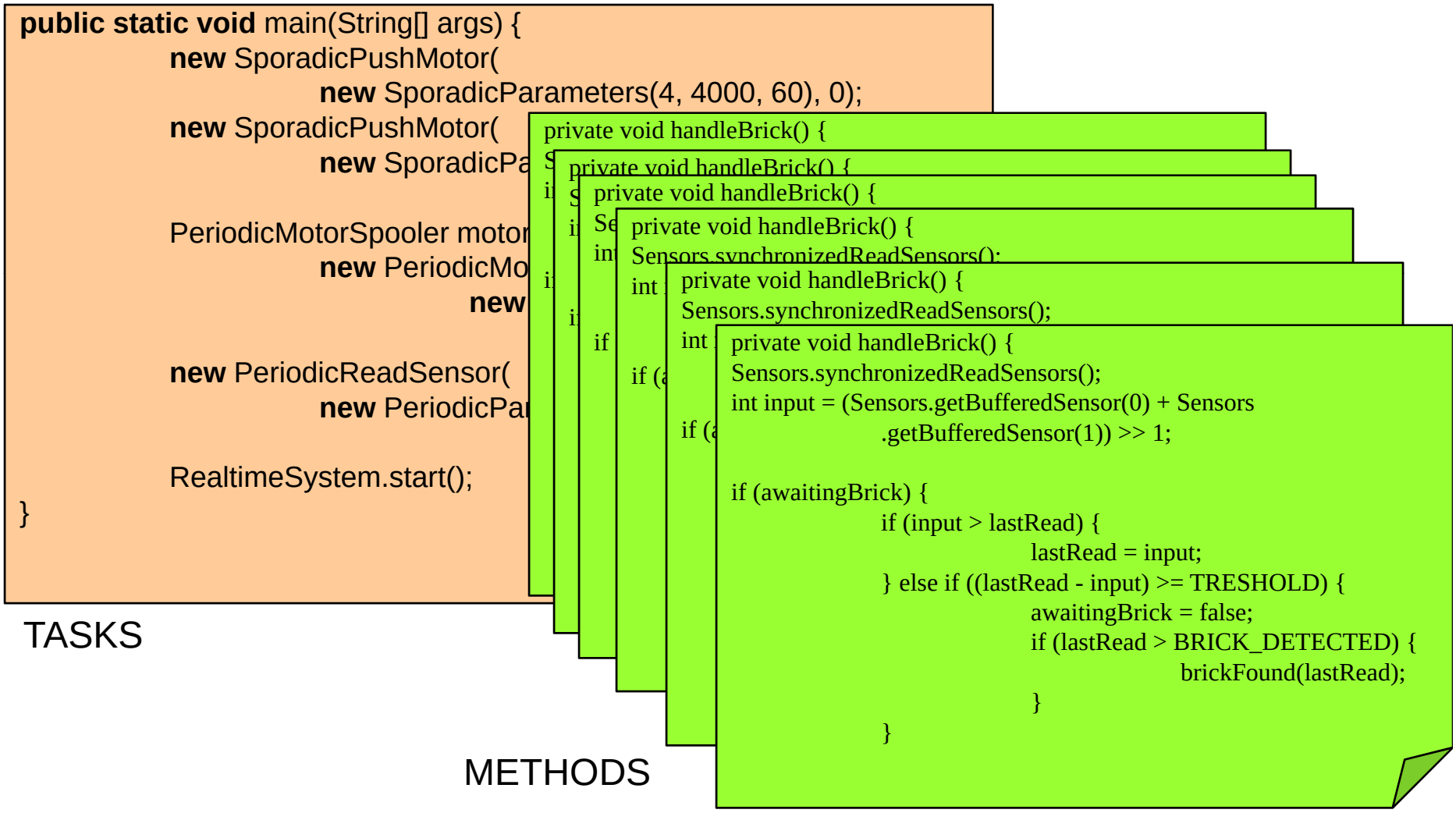
```
protected boolean run()
    if i<5 {
        i = i + 4;
    } else {
        i = i * 4;
    }
    return true;
}
```

Data abstracted

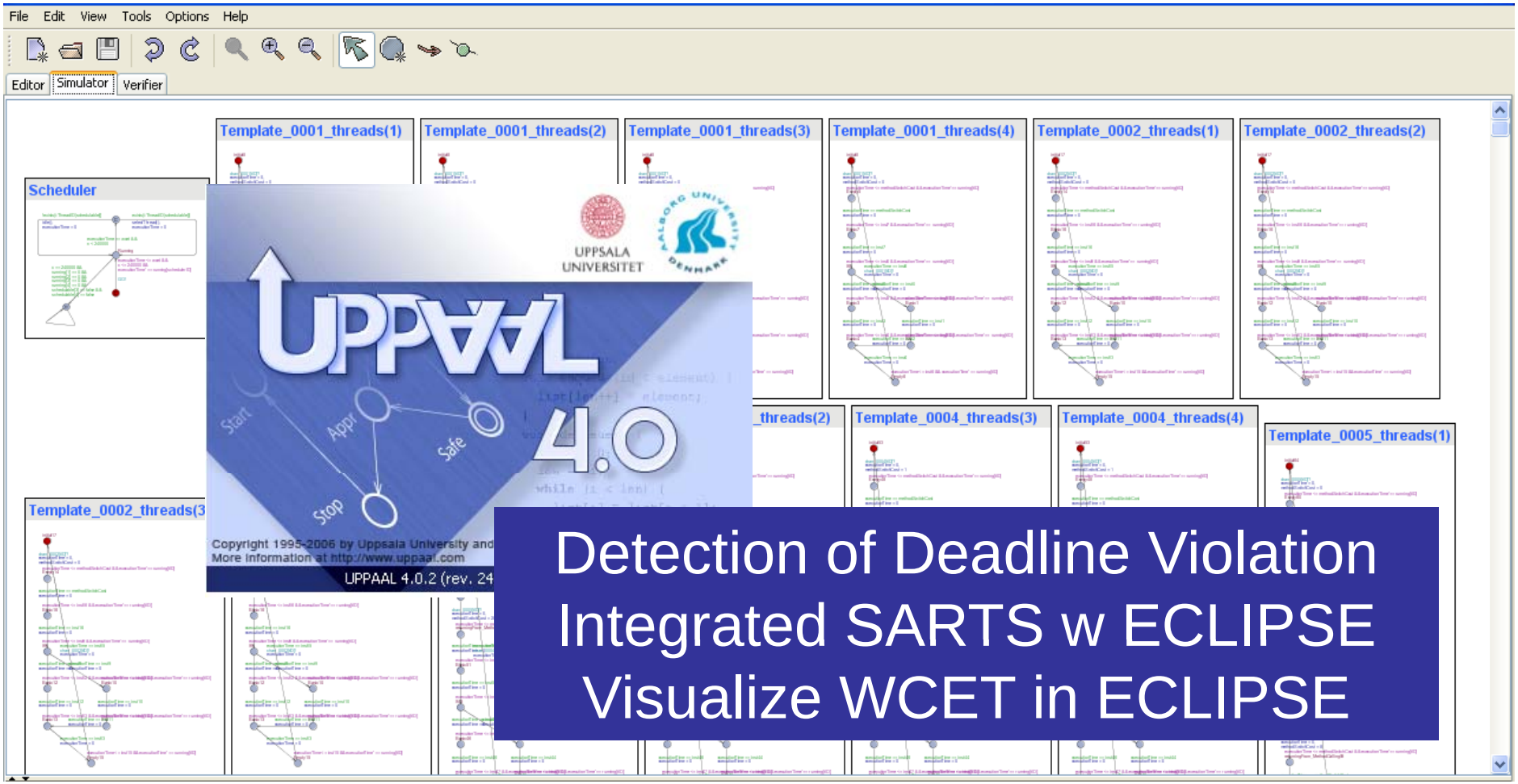
Timing = WCET from microcode



SARTS – from Safety Critical Java



SARTS – to Timed Automata



18 methods + 4 tasks = 76 components

SARTS – Experiments

Compiler	Verification time	Result
Javac	14m 29s	Satisfied
Eclipse	1m 55s	Satisfied

Breadth First Search

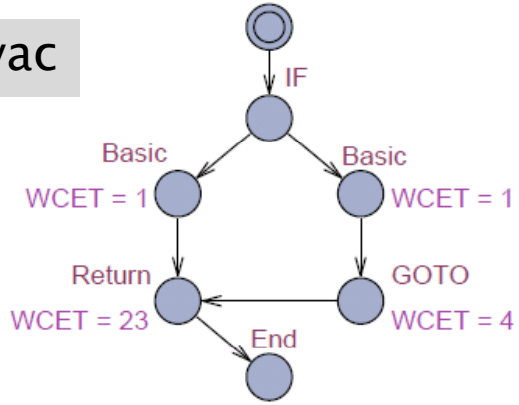
Compiler	Verification time	Result
Javac	4m 23s	Satisfied
Eclipse	51s	Satisfied

Depth First Search + Agressive SS Reduction

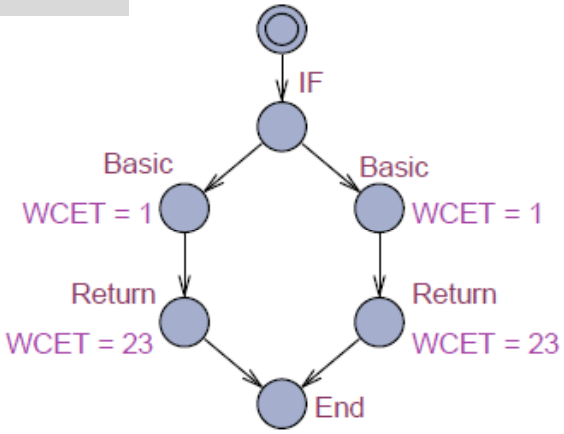
Compiler	Verification time	Result
Javac	16s	Satisfied
Eclipse	9s	Satisfied

Convex Hull Approximation

Javac



Eclipse

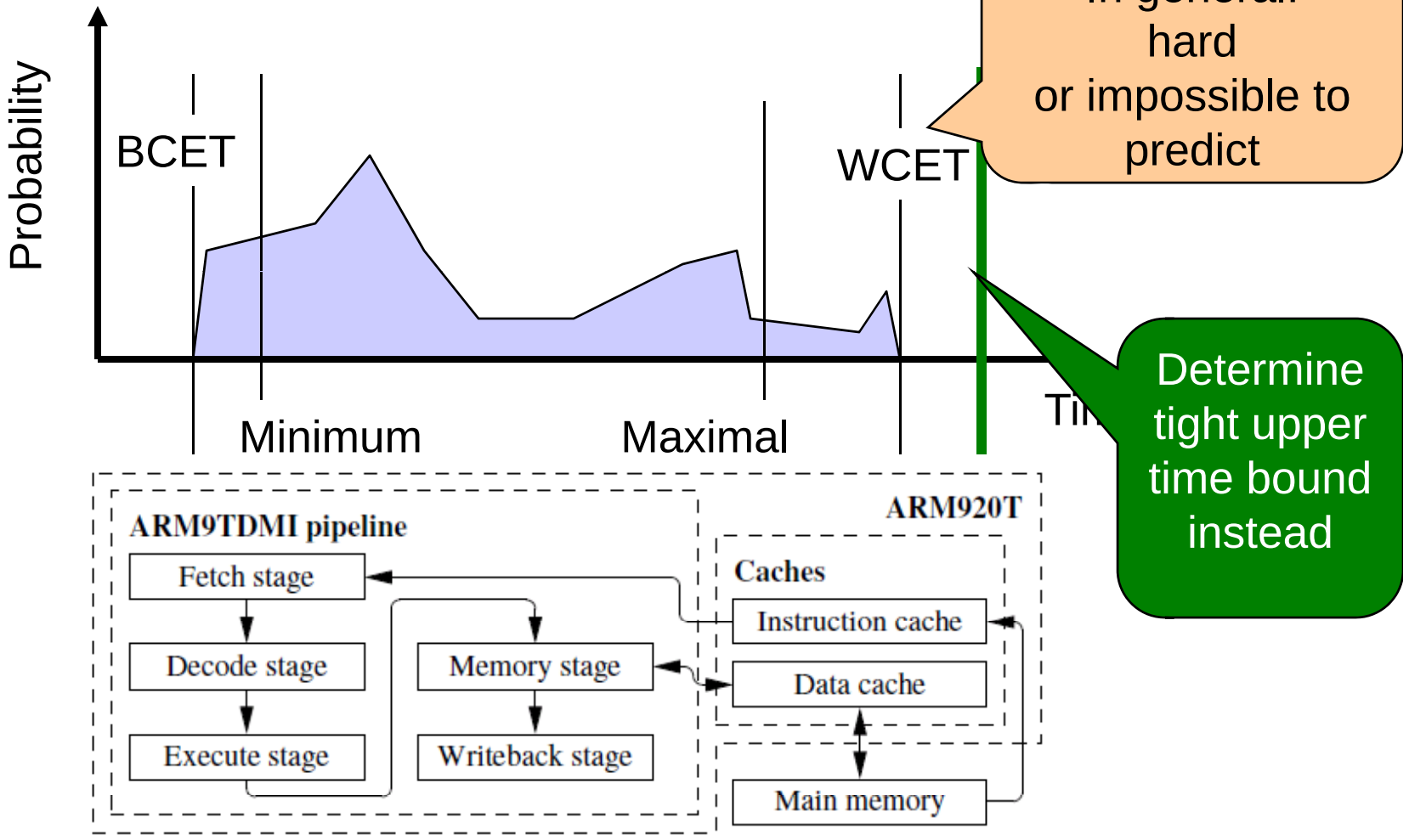


METAMOC

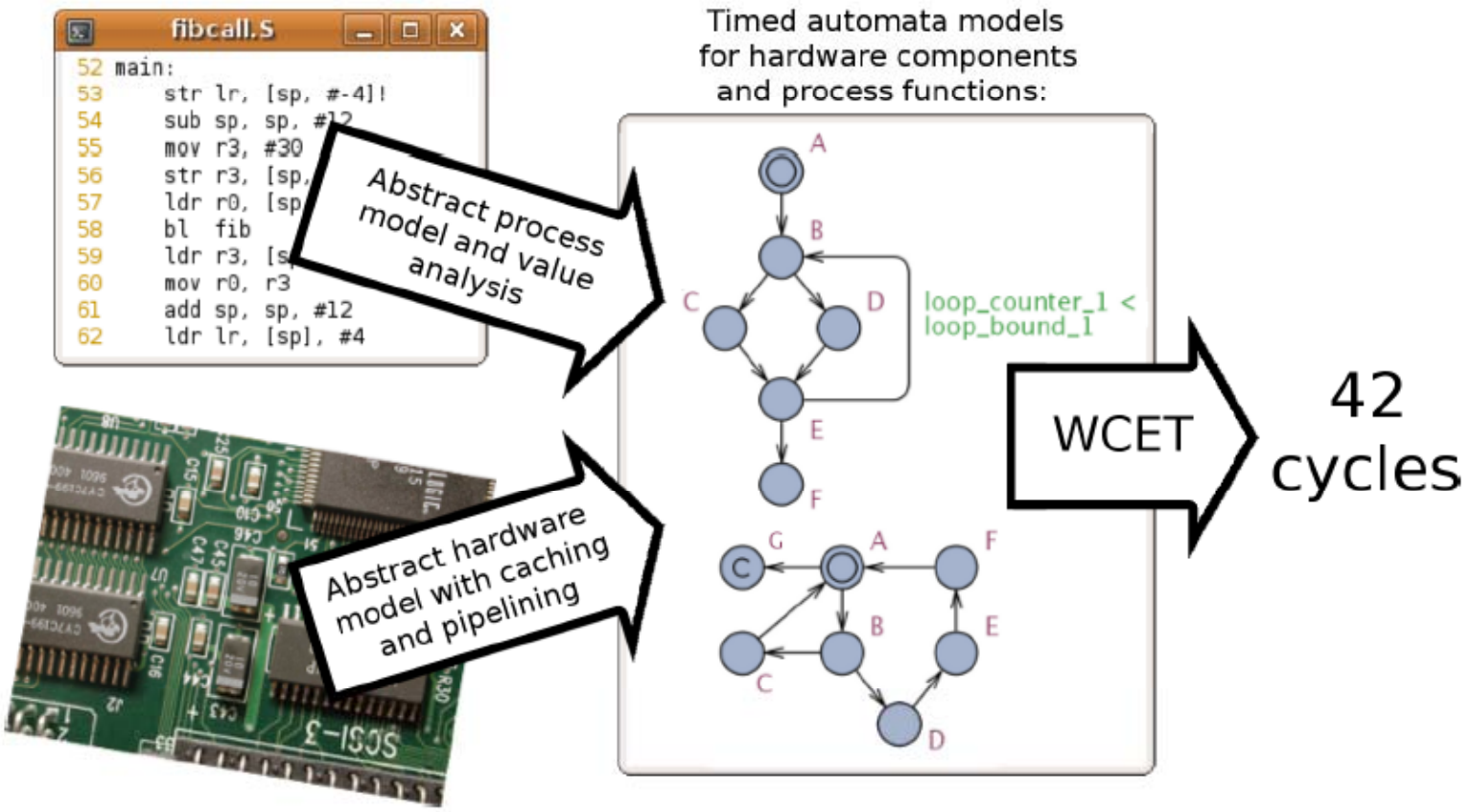
Modular Execution Time Analysis using MOdel Checking

with
Andreas Dalsgaard
Mads Christian Olesen
Martin Toft
René Rydhof Hansen

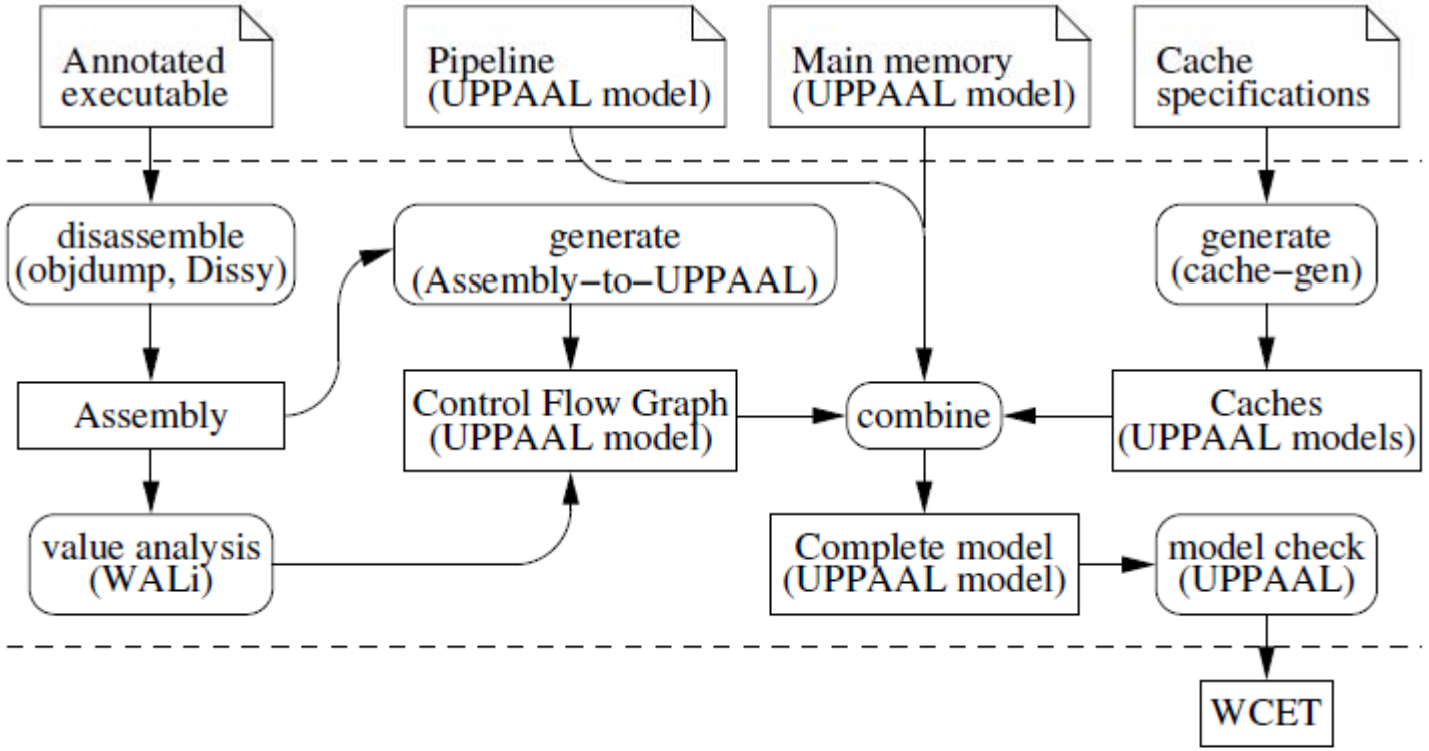
WCET: Worst Case Execution Time



METAMOC


fibcall.S
52 main:
53 str lr, [sp, #-4]!
54 sub sp, sp, #12
55 mov r3, #30
56 str r3, [sp,
57 ldr r0, [sp
58 bl fib
59 ldr r3, [s
60 mov r0, r3
61 add sp, sp, #12
62 ldr lr, [sp], #4
Abstract process model and value analysis
Abstract hardware model with caching and pipelining
Timed automata models for hardware components and process functions:
A
B
C
D
E
F
loop_counter_1 < loop_bound_1
G
WCET
42 cycles

Overview of METAMOC

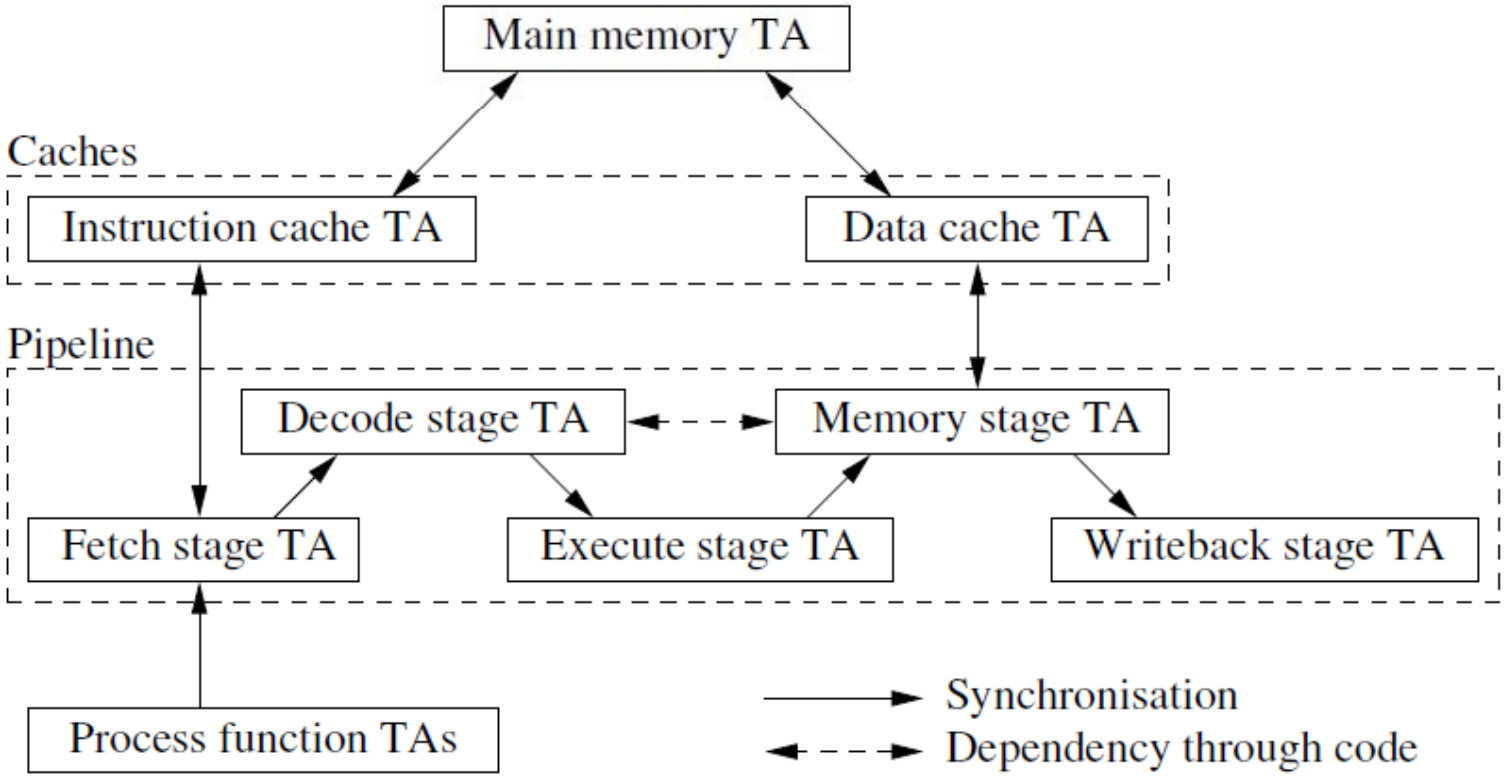


Value analysis in METAMOC

- Memory addresses needed for cache hit/miss predictions
- Registers used as base and offset for memory accesses
- Overapproximate possible register values
- METAMOC uses Weighted Push-Down Systems (WPDSs) for an inter-procedural, control-flow sensitive value analysis[1]
- Weighted Automata Library (WALi) utilised

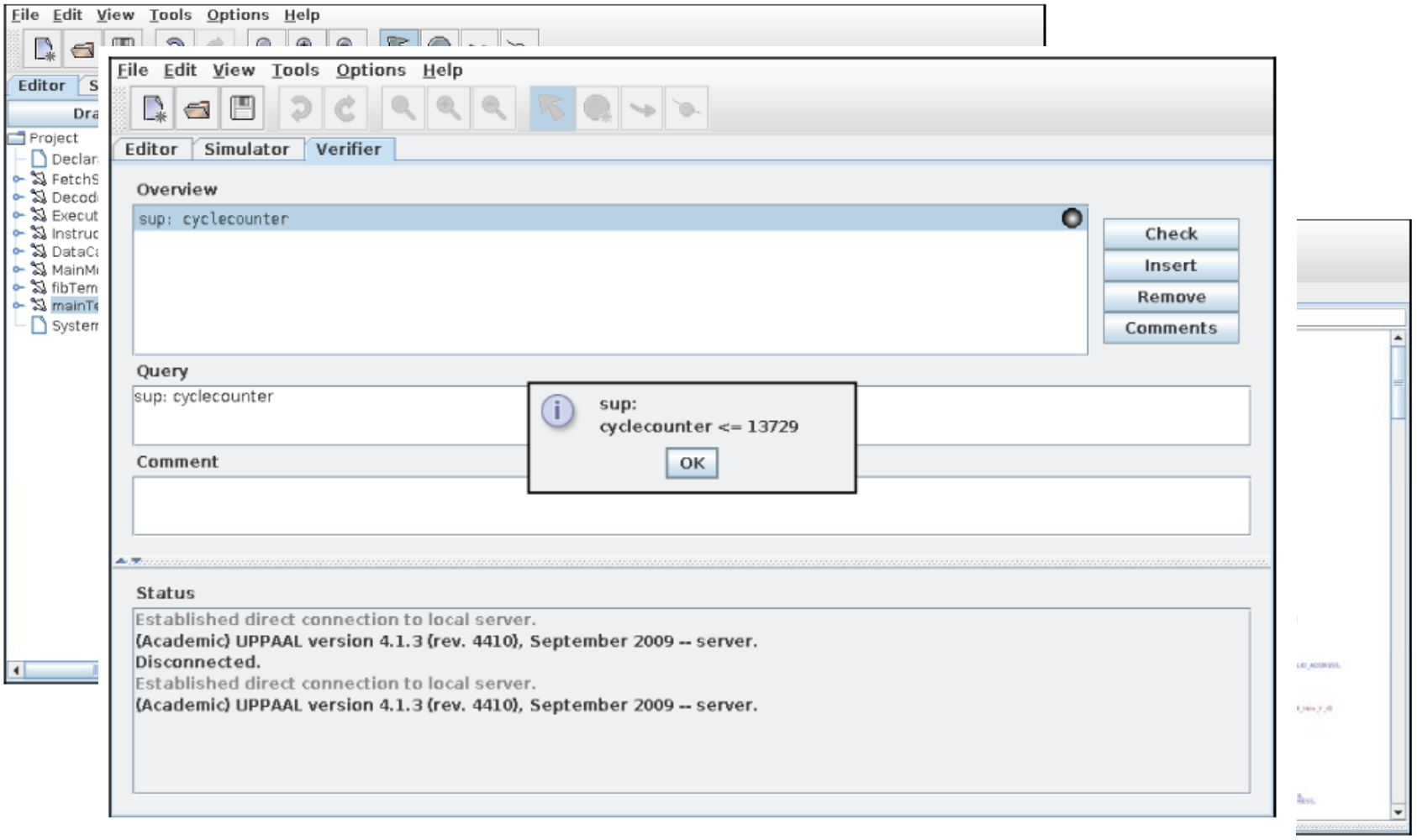
[1] T. Reps, A. Lal, N. Kidd. *Program Analysis using Weighted Push-Down Systems*. In *FSTTCS 2007*, vol. 4855 of *LNCS*, pp. 23–51.

Modeling in METAMOC

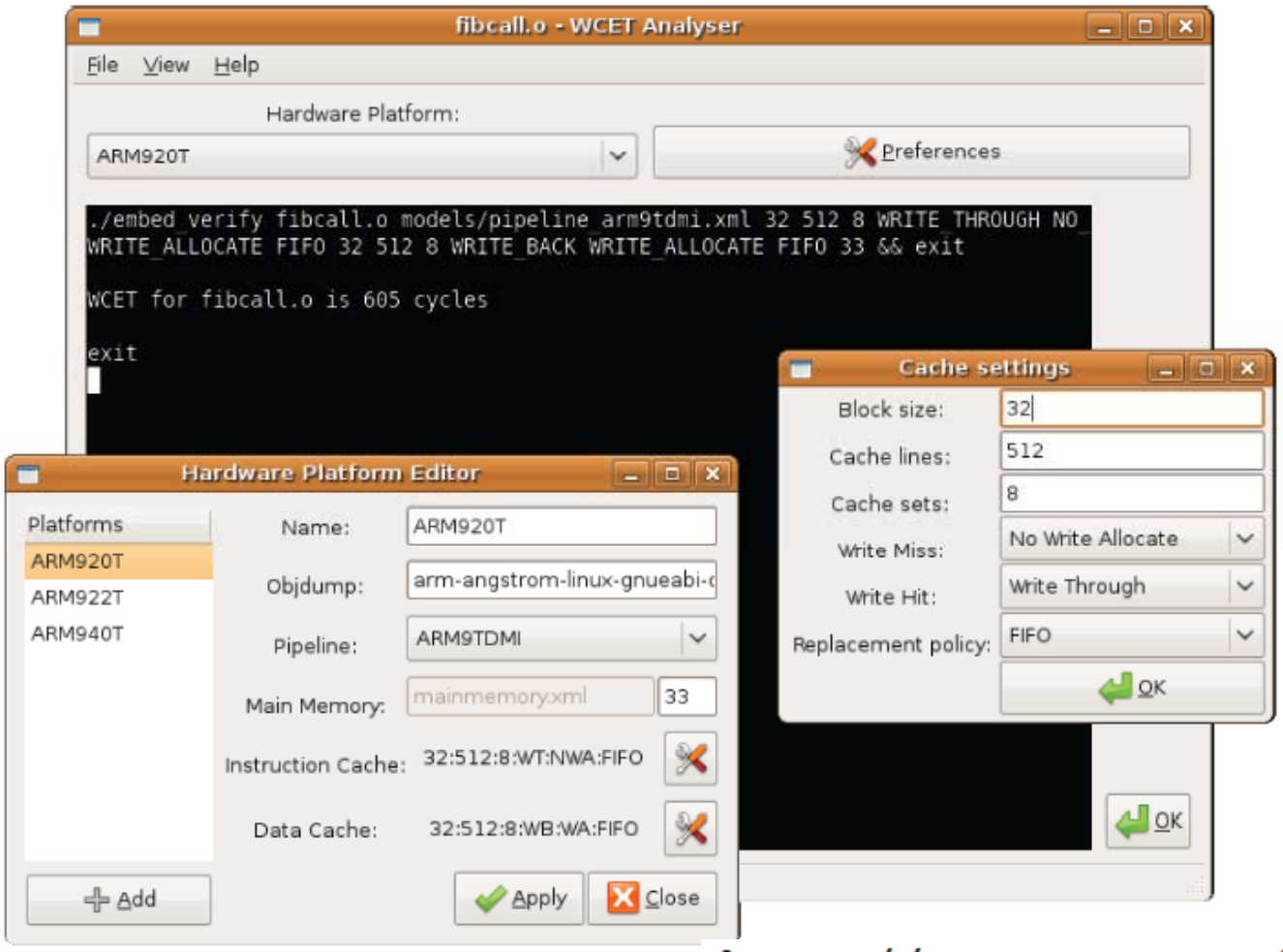


Overview of the ARM9 automata

Modeling in UPPAAL



GUI for METAMOC



http://metamoc.martintoft.dk

Status

- Started out with ARM9 support
  - Five stage pipeline, instruction cache, data cache, simple main memory
- Demonstrated the method convincingly—we thought
- The WCET community: "It's a case study. You haven't demonstrated the method's modularity."
- Now:
  - Support for ARM7, ARM9 and ATMEL AVR 8-bit
  - . . . with modest effort
- Accepted paper for WCET 2010, the 10th Int'l Workshop on Worst-Case Execution-Time Analysis
  - A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen and K. G. Larsen. *METAMOC: Modular Execution Time Analysis using Model Checking.*

Experiments

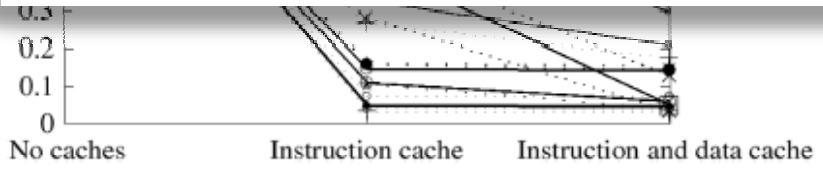
- Evaluation using WCET benchmark programs from Mälardalen Real-Time Research Centre[2]
  - Applicability
  - Performance
- Discarded a number of programs
  - Floating point operations handled by software routines
  - Dynamic jumps
  - Some programs do not compile
- 21 programs for ARM and 19 programs for AVR
- Manually annotated loop bounds

Experiments / Future

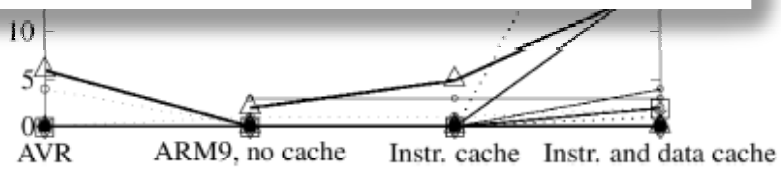
ARM9, 21 benchmarks	
Analysable without caches	21
Analysable with instruction cache	20
Unanalysable, state space explosion	1
Analysable with data and instruction cache	20
Unanalysable, state space explosion	1

ATMEL AVR 8-bit, 19 benchmarks	
Analysable	16
Unanalysable, state space explosion	3

- Looking for WCET benchmark programs with reference WCETs
- A general, cycle-accurate hardware emulator, turning a hardware description and a program into a WCET



Relative improvement in WCET for ARM9.



Analysis times in minutes for AVR and ARM9.

Conclusion & Future

CONCLUSION

- Reduction of Verification Gap
- Simplicity
  - Programming languages
  - Processor architecture
  - Scheduling principles
- RT Model checking
  - Less pessimistic
  - Enables more complex settings
  - Ease of use

FUTURE

- Support for more platforms
- Combined Schedulability & WCET Analysis
- Loop bounds ?
- More applications
- Improved Model Checker
  - 64 bit version
  - Distributed MC
  - Abstract Caches using Lattice Types

- sarts.boegholm.dk
- metamoc.martintoft.dk
- www.uppaal.com