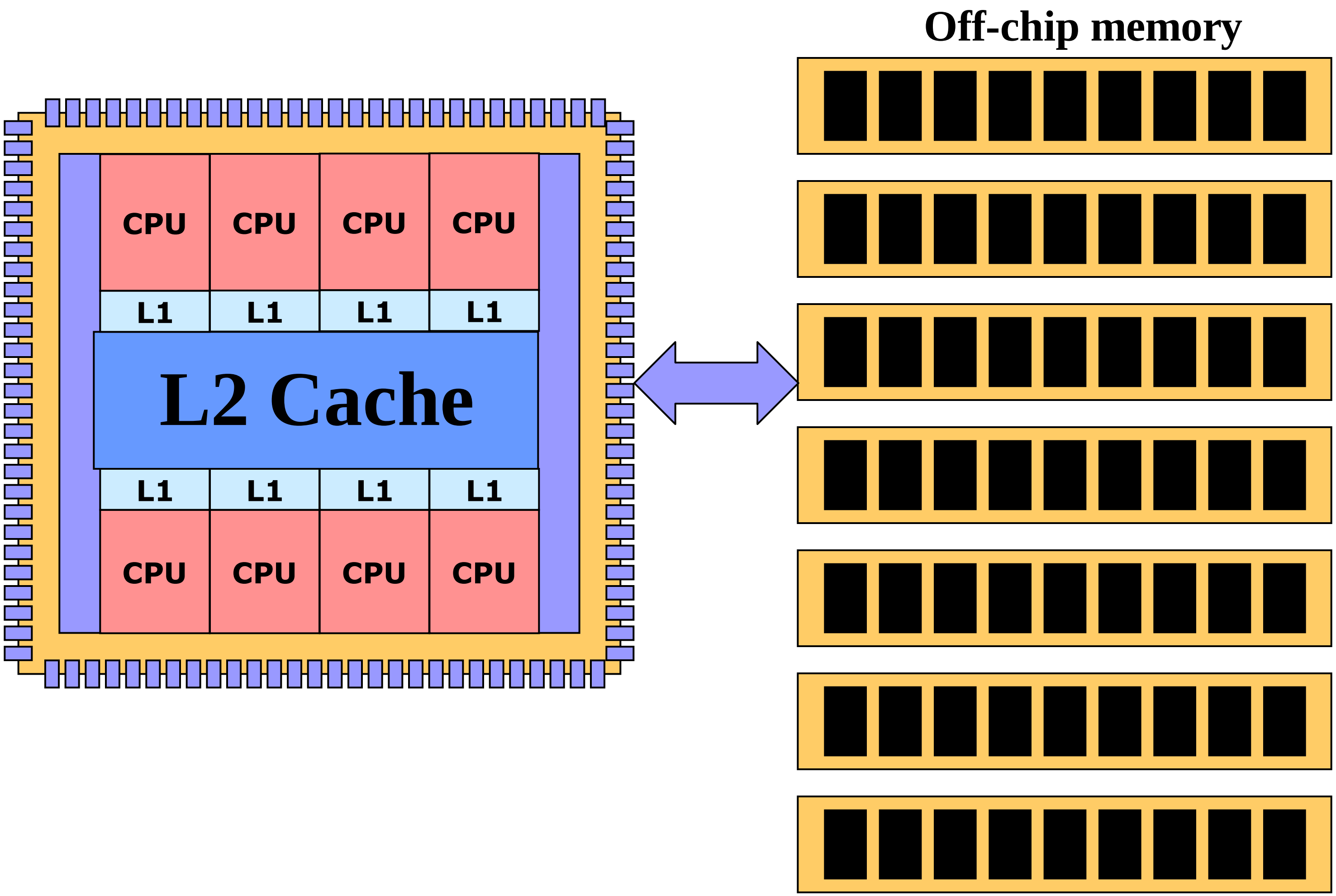
PART 2

Multicore Real-Time Systems

OUTLINE

- **Multicore Challenges (Real-Time Applications?)**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Cache Contention [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

What is multi-core, and why?



Multicore = Multiple hardware threads sharing the memory system

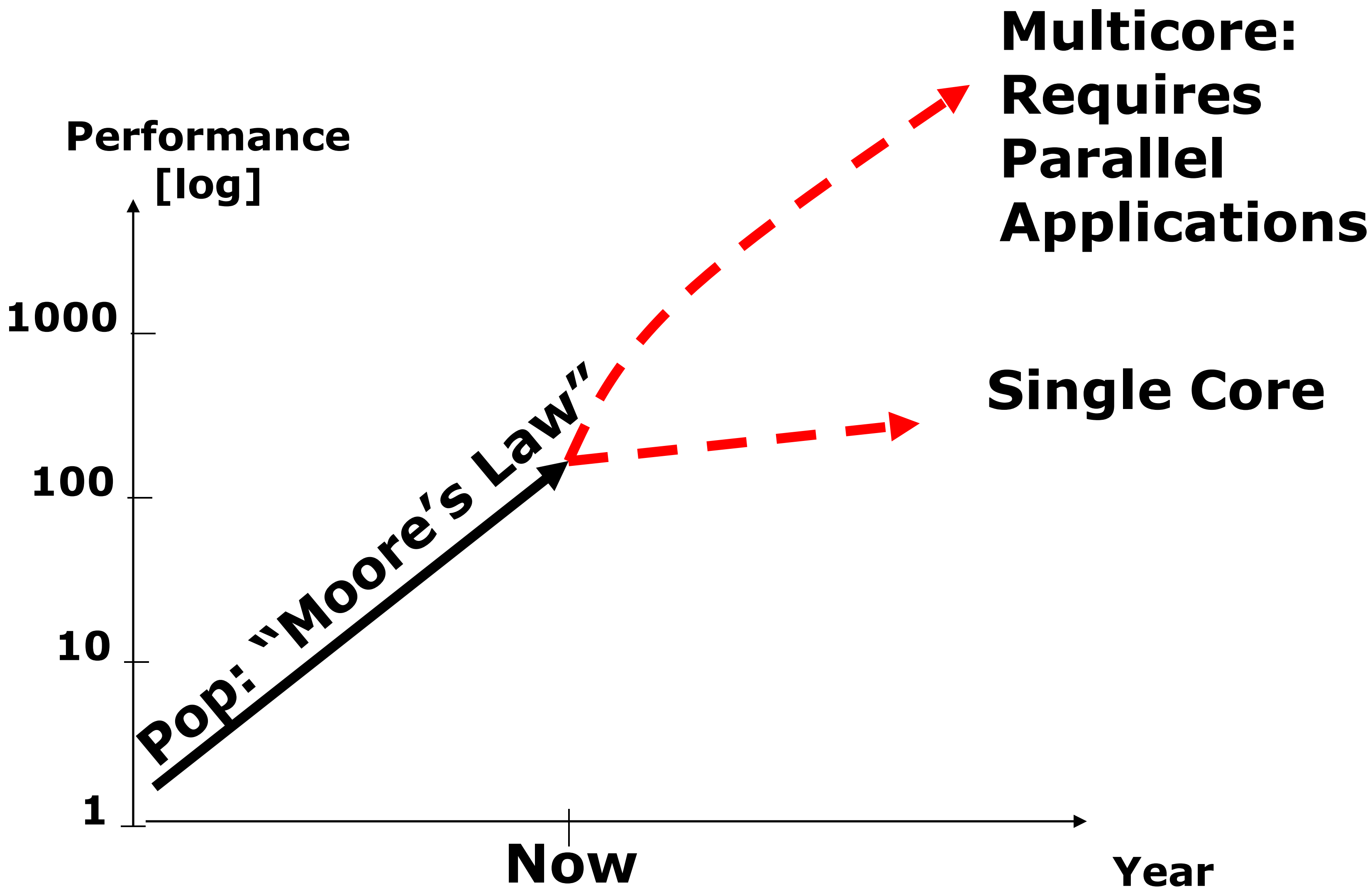
Year 2003-2007

The free lunch is over & Multicores are coming !

Erik Hagersten

Chief Architect at SUN (till 1999)
Professor of Computer Architecture, Uppsala

Free lunch is over, Erik Hagersten



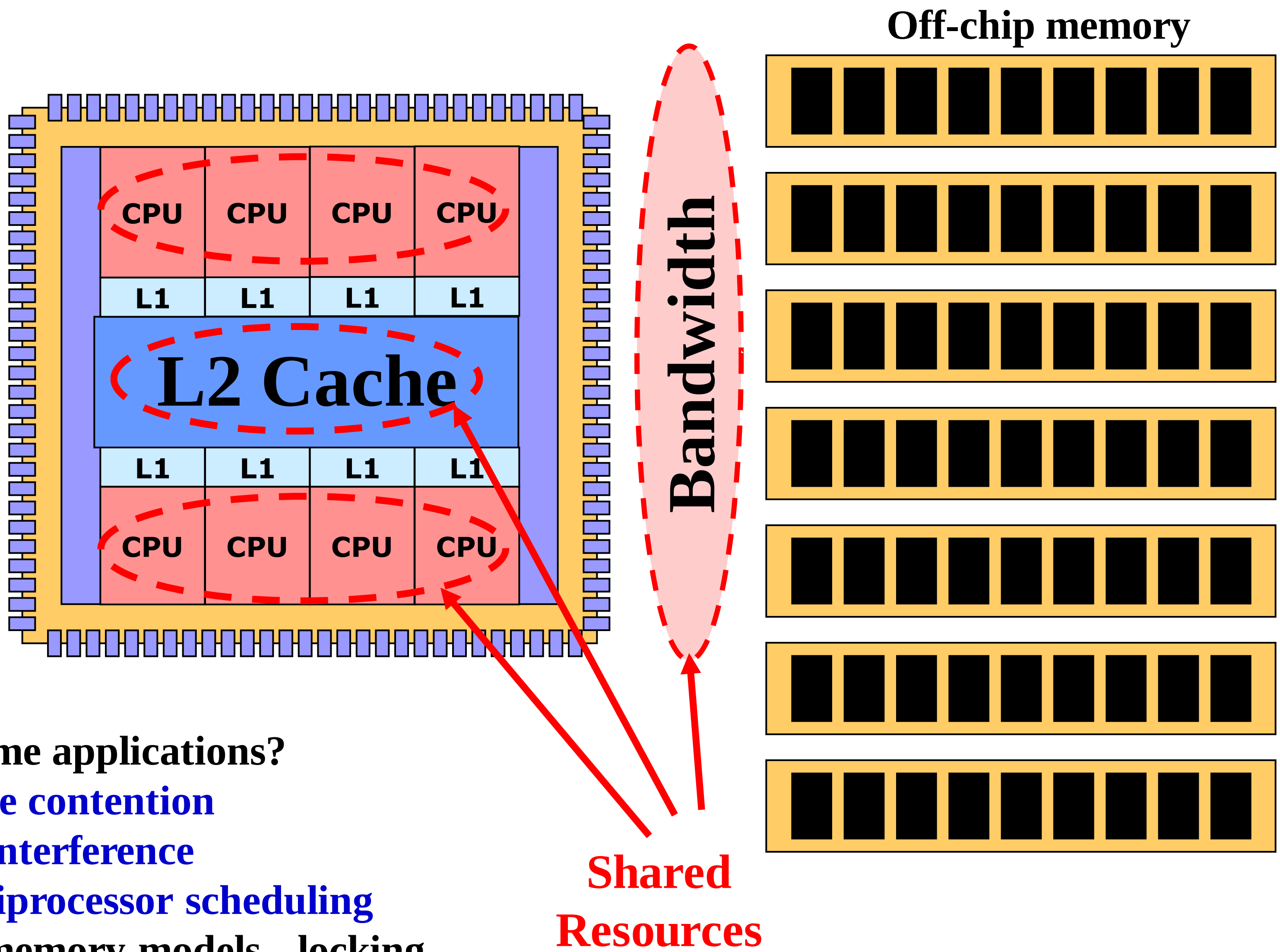
Theoretically you may get:

- Higher Performance
  - Increasing the cores -- unlimited computing power ∞ !
- Lower Power Consumption
  - Increasing the cores, decreasing the frequency
    - Performance (IPC) = Cores * F ➔ 2* Cores * F/2 ➔ Cores * F
    - Power = $C * V^2 * F$ ➔ $2 * C * (V/2)^2 * F/2$ ➔ $C * V^2 /4 * F$

➔ Keep the "same performance" using ¼ of the energy (by doubling the cores)

This sounds great for embedded & real-time applications!

Multicore Challenges



Real-time applications?

-- Cache contention

-- Bus interference

-- Multiprocessor scheduling

Weak memory models - locking

Cheap/expensive Synchronization

Year 2008 (June)

UPMARC:

Uppsala Programming Multicore Architecture Research Center

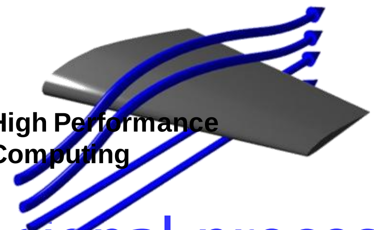
Awarded by the Swedish Research Council
10 millions US$: 2008 -- 2018

Similar centers: Stanford, UC Berkeley

UPMARC Research Areas

Applications & Algorithms

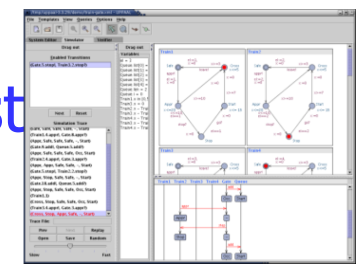
- Climate simulation
- PDE solvers
- Parallel algorithms for RT signal processing
- Parallelization of network protocols



Computer Networks

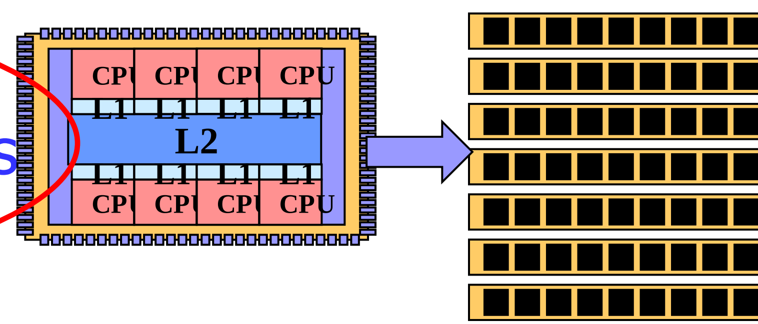
Verification & Language Technology

- Erlang, language constructs/libraries, run-time syst
- Static analysis, Model-checking , testing, UPPAAL



Resource Management

- Efficiency: performance opt.
- Predictability: real-time applications



Year 2008 (November)

CoDeR-MP:

Computationally Demanding Real-Time Applications on Multicore Platforms

Awarded by the Swedish Strategic Research Foundation
3 millions US$: 2009 -- 2014

Objective (CoDeR-MP)

New techniques for

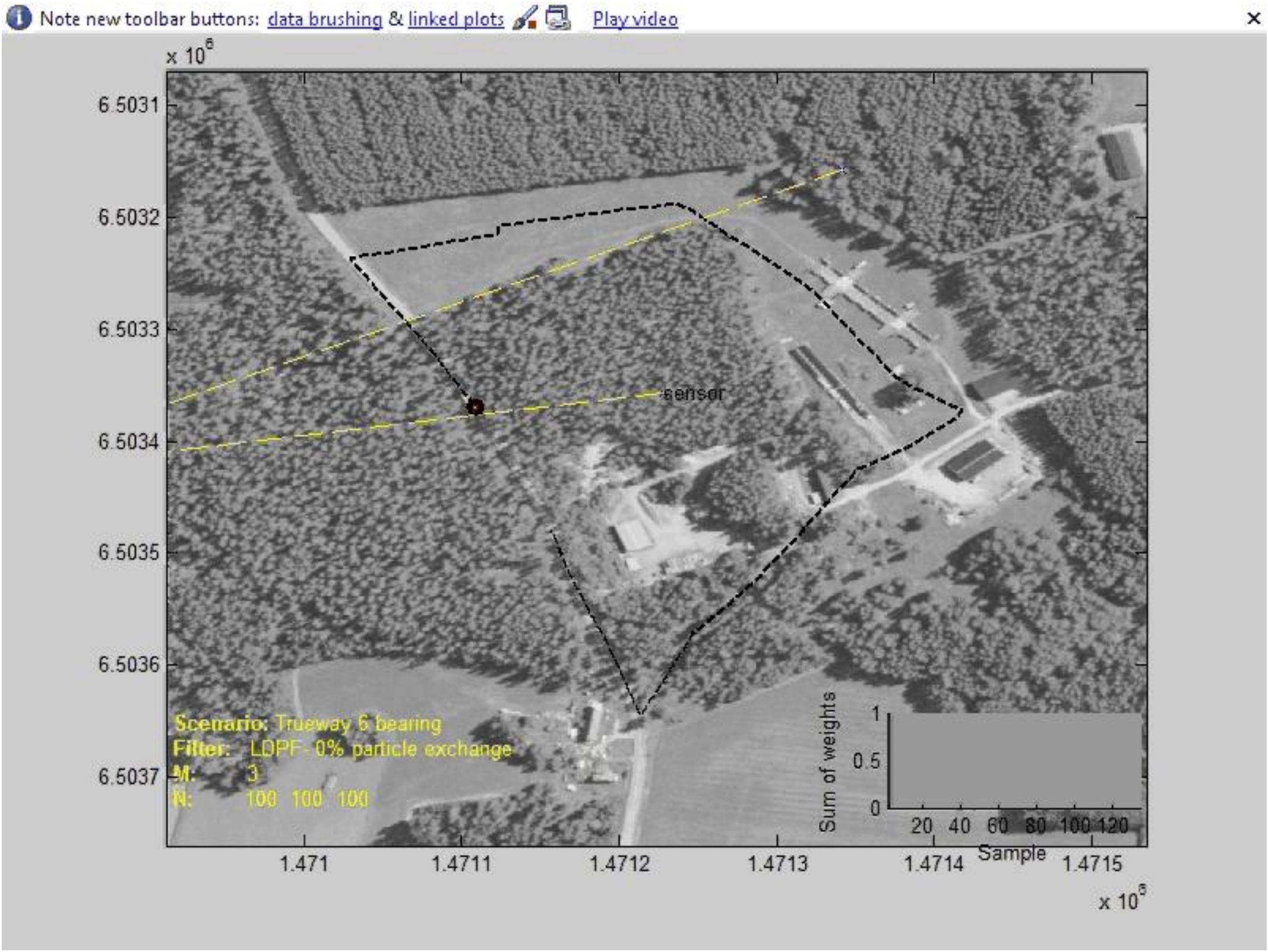
- High-performance software for soft RT applications &
- Predictable software for hard RT applications

on multicore

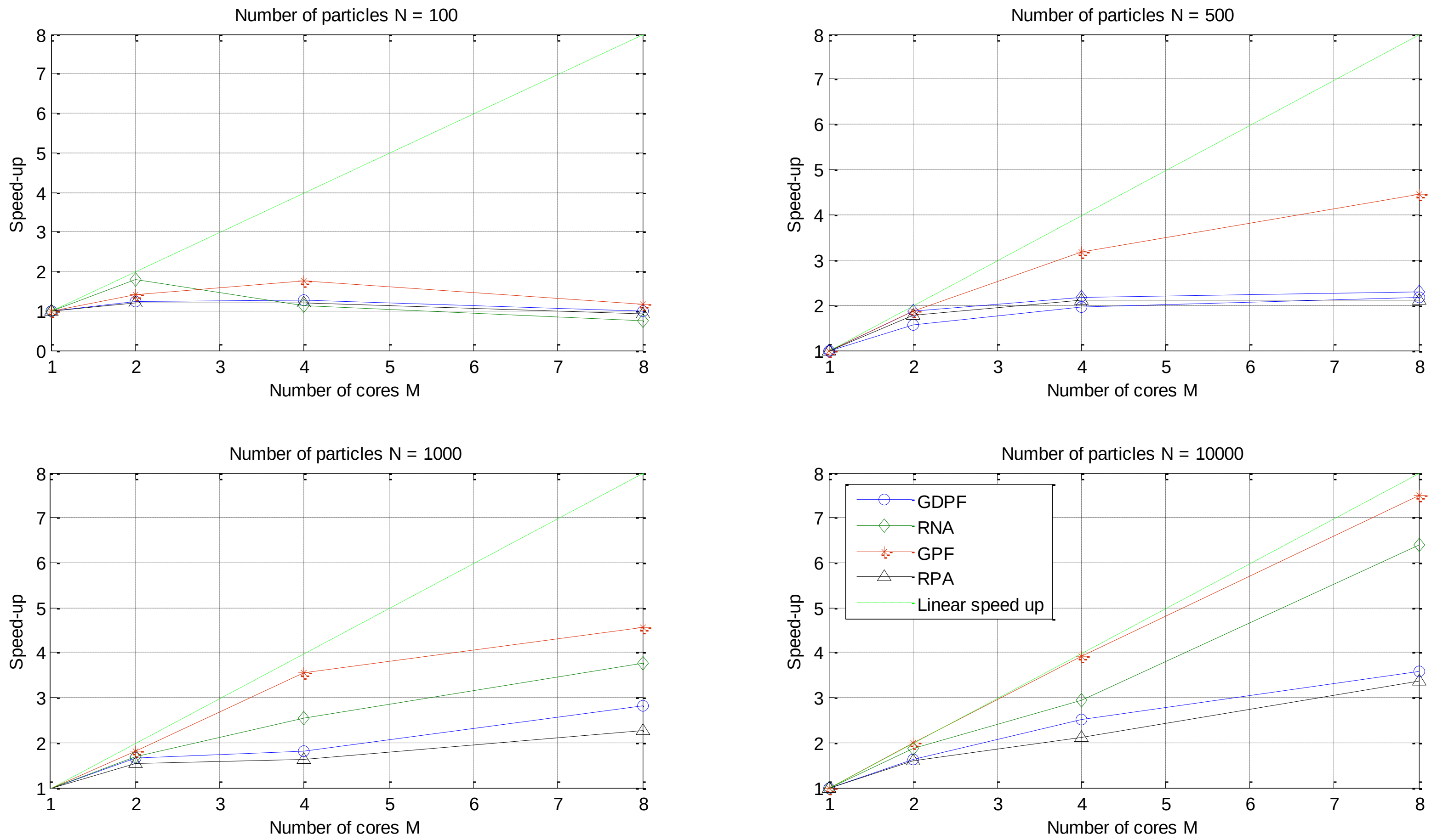
Industry participation

- Control Software for Industrial Robots – ABB robotics
- Tracking with parallel particle filter – SAAB

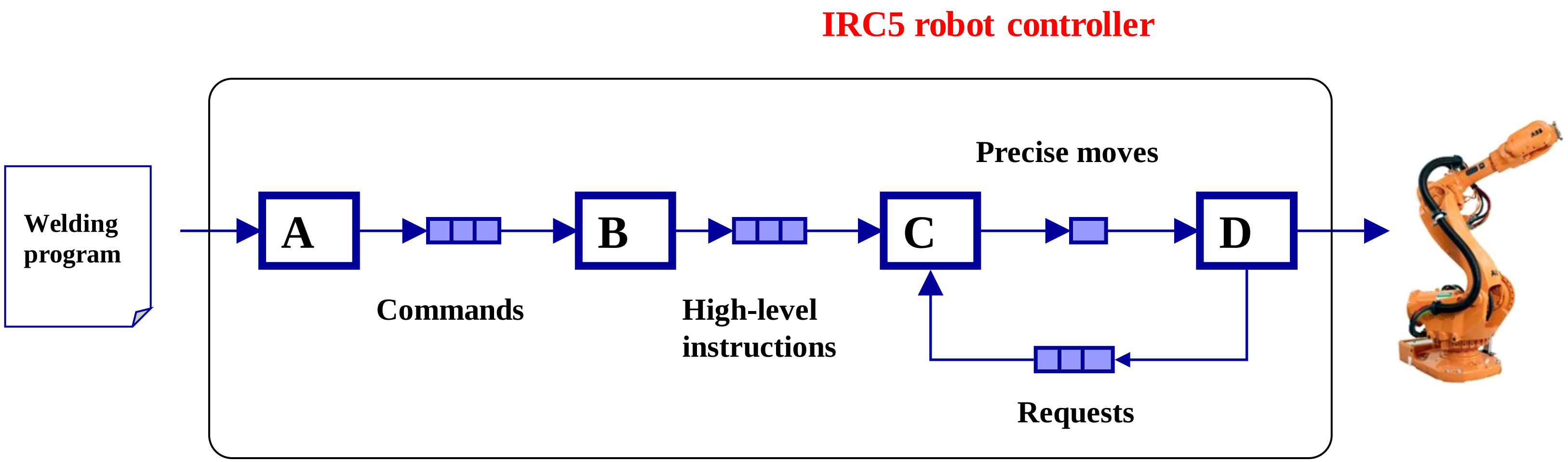
Real-Time Tracking with parallel particle filter – SAAB



Parallelization
(Speed-up for PF algorithms)



Real-Time Control – ABB Robotics



Mixed Hard and Soft Real-Time Tasks
20% hard real-time tasks

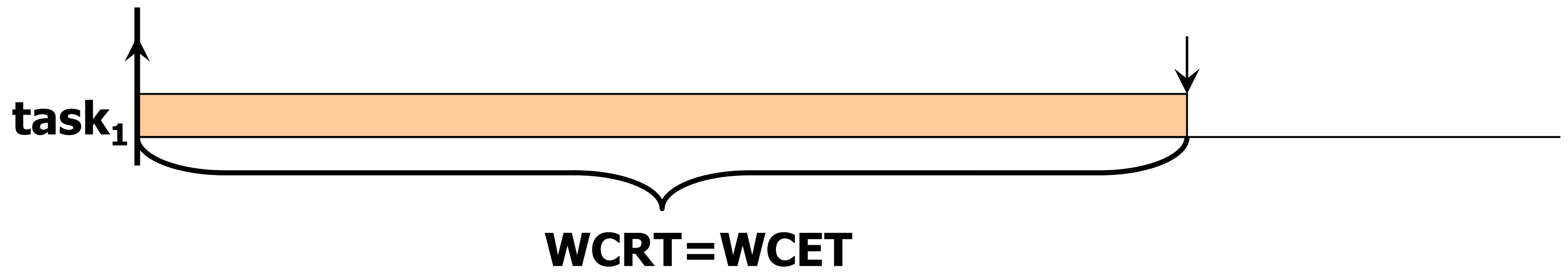
Main concerns:
Isolation between hard & soft tasks: “fire walls”
Real-time guarantee for the 20% “super” RT tasks
Migration to multicore?

OUTLINE

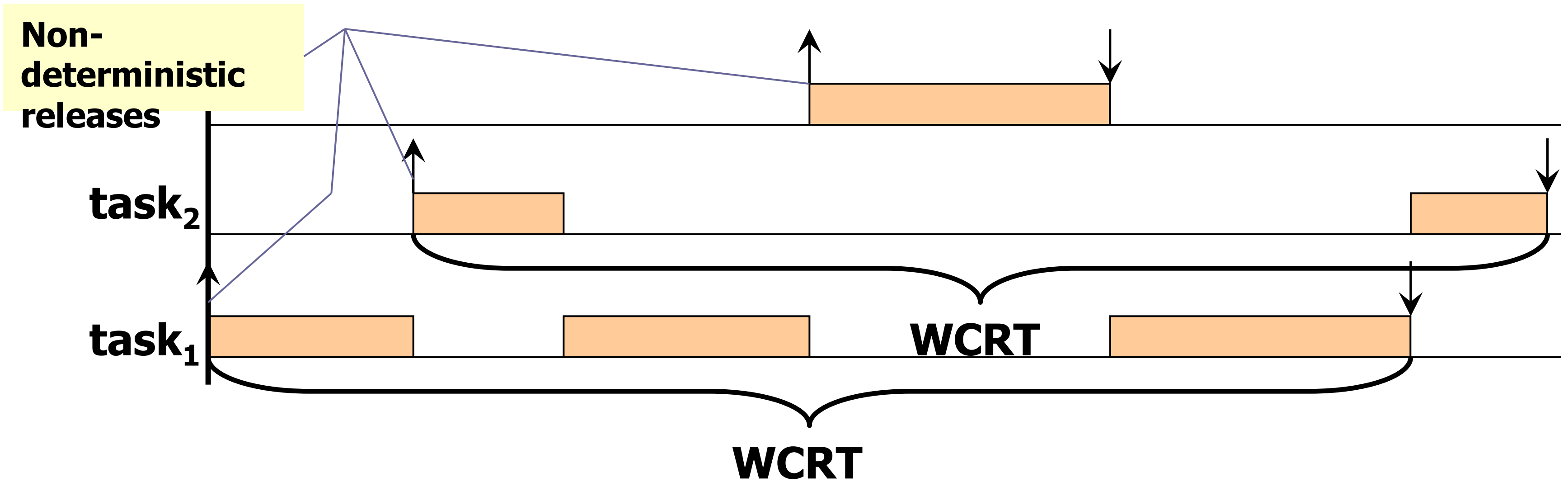
- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Cache Contention [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

Single-Processor Timing Analysis

Sequential Case (WCET analysis)



Concurrent Case (Schedulability analysis)



On single processor:

WCET = #instructions + “cache miss penalty”

“Cache miss penalty” can be estimated “precisely”
by e.g abstract interpretation – based on the history of executions

On multicore processor:

WCET = #instructions + "cache miss penalty" + …

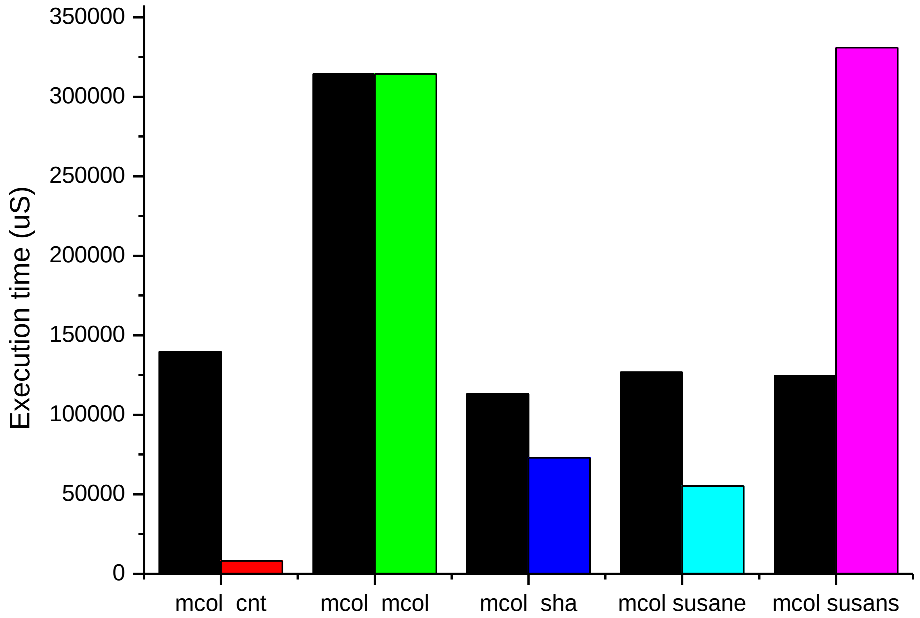
"Cache miss penalty" can be much larger due to cache contentions from the other cores … and also bus delays

WCET of a single task can not be estimated in isolation

An Experiment on a LINUX machine with 2 cores

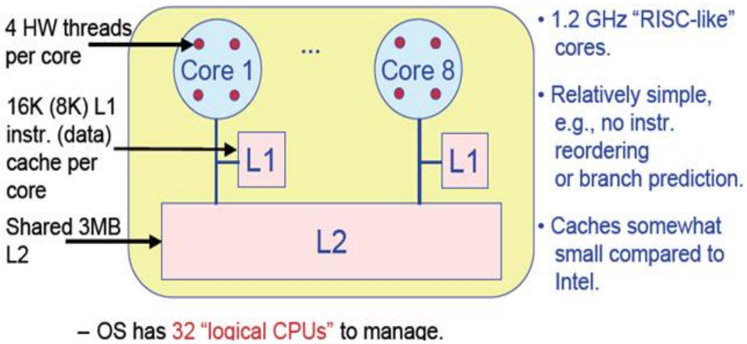
(Zhang Yi)

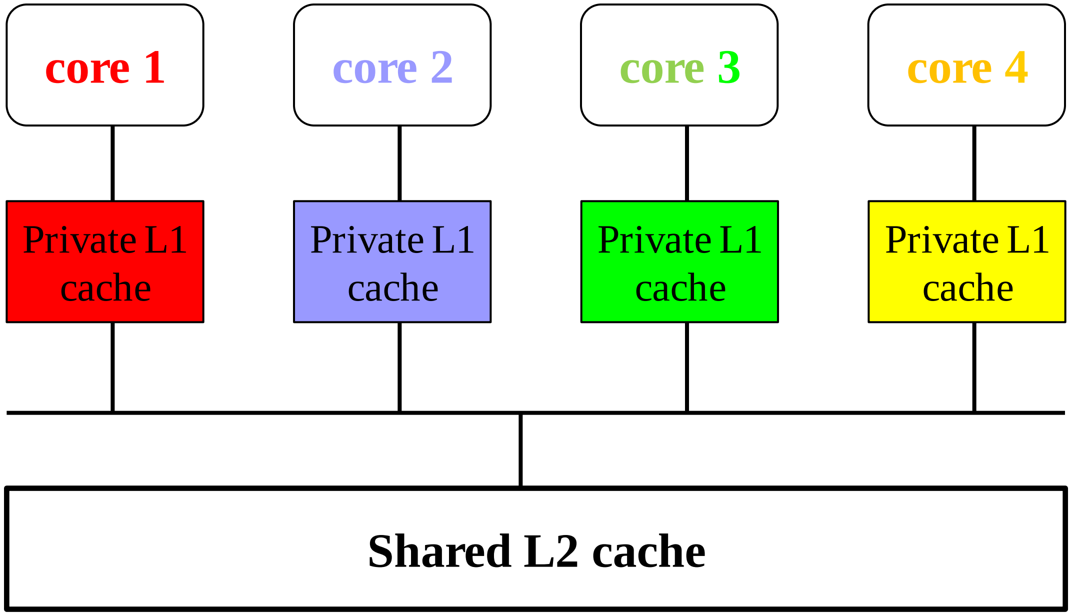
WCET (vary 10 – 50%)



mcol runs with different programs

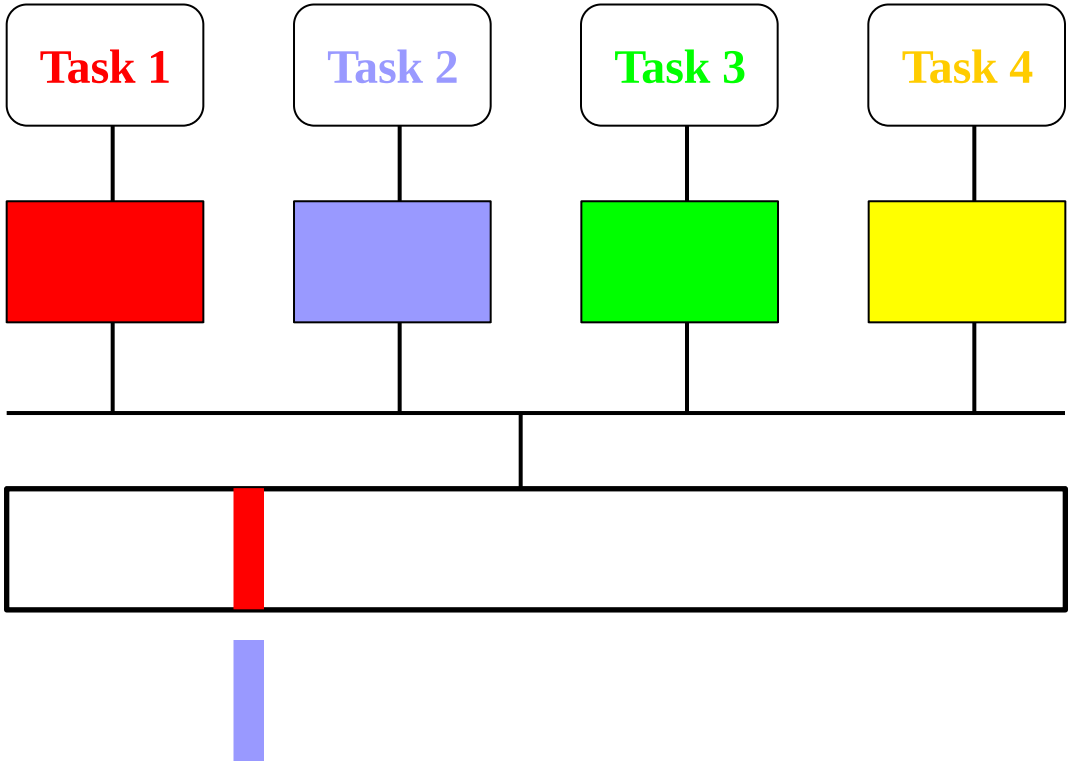
An Example Architecture





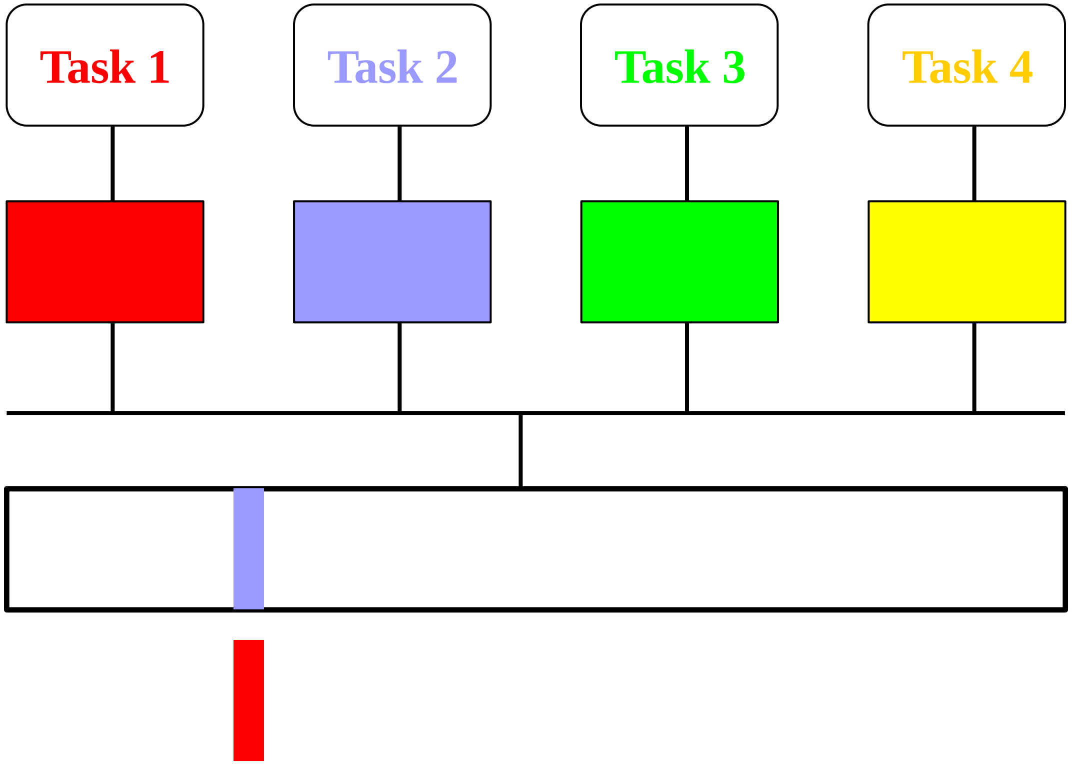
Cache analysis on multicore

- **L2 cache contents of task 1 may be over-written by task 2**

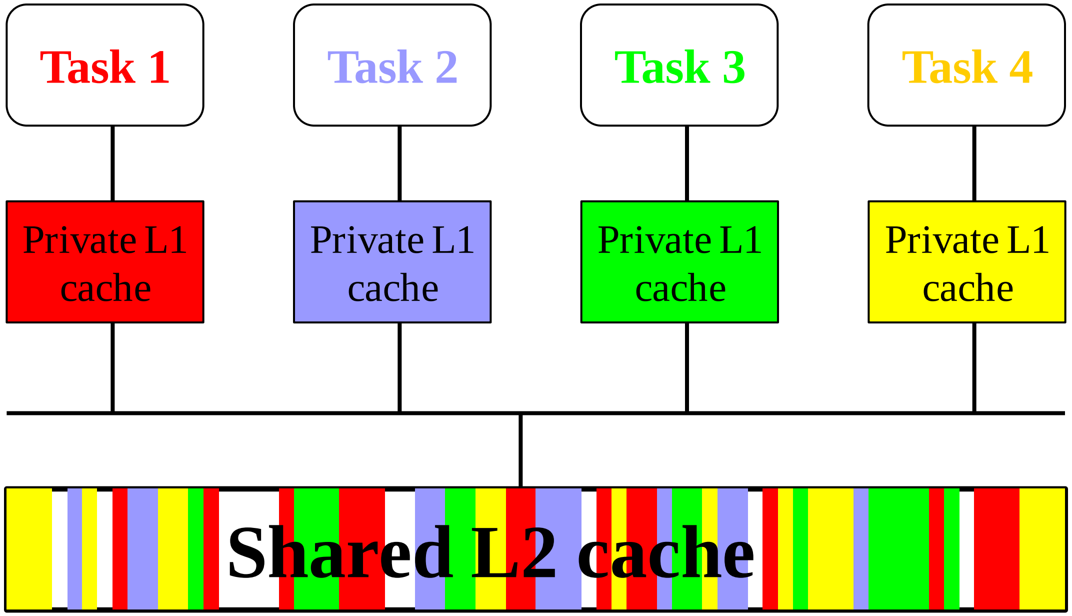


Cache analysis on multicore

- **L2 cache contents of task 1 may be over-written by task 2**



Cache analysis on multicore

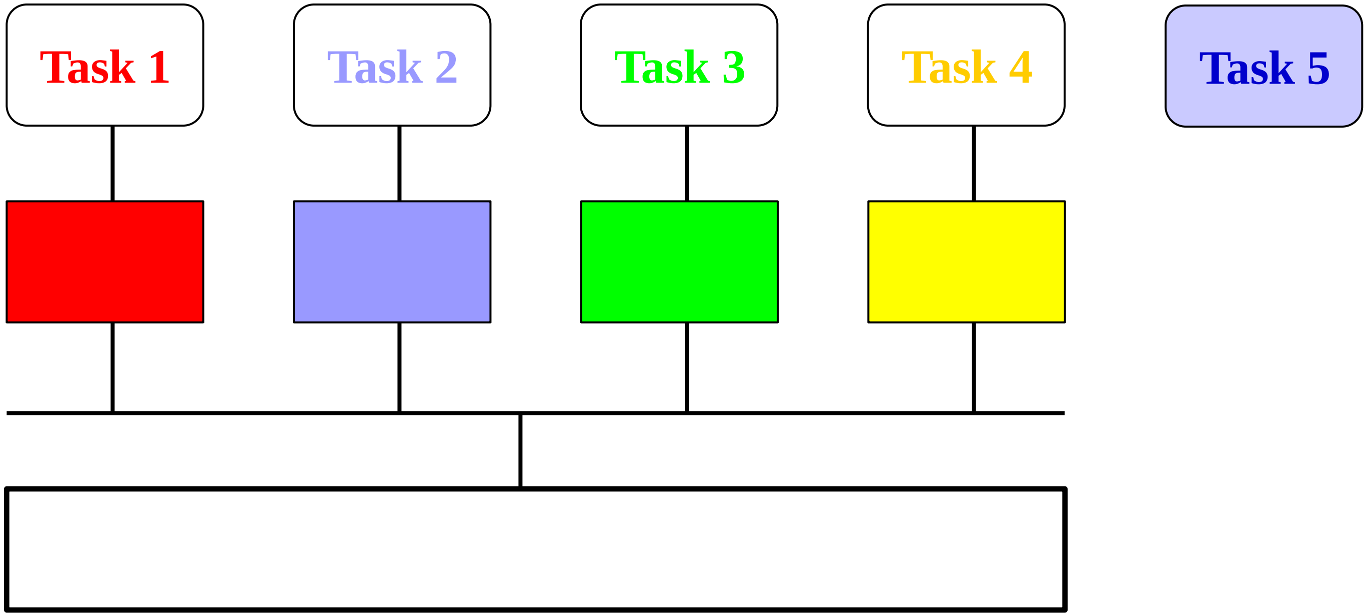


The multicore challenge: WCET analysis

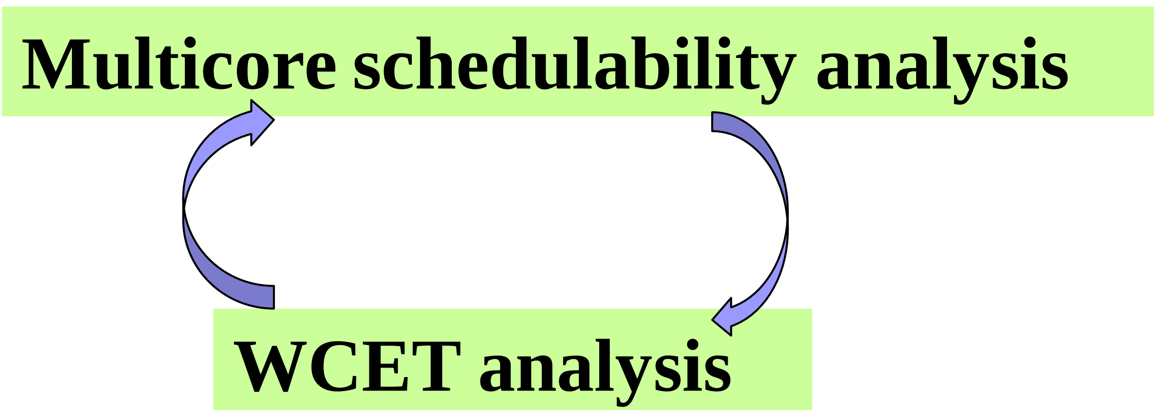
- Must explore all interleavings of “execution paths” on all cores
- Must represent “precise” timing information on each core (to keep track of the progress on each core and cache contents)

The multicore challenge: Schedulability analysis

- #cores < #tasks



Cyclic dependence



The “Impossible” Problem

1. We must “schedule” the shared cache lines
2. We must “schedule” the shared memory bus
   - when cache misses ocur
3. We must “schedule” the shared cores

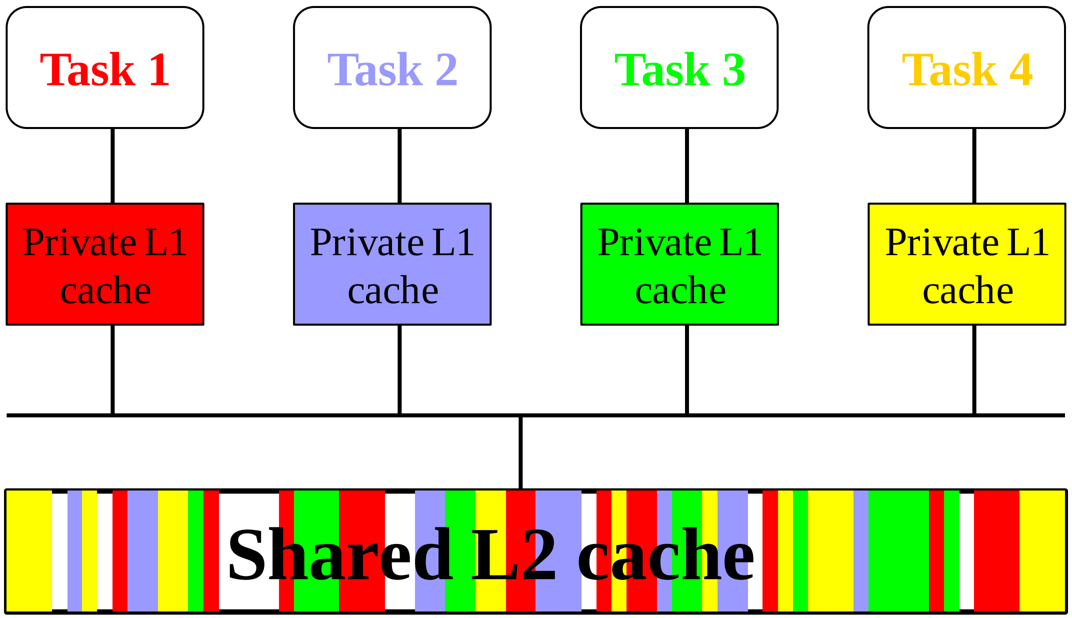
OUTLINE

- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

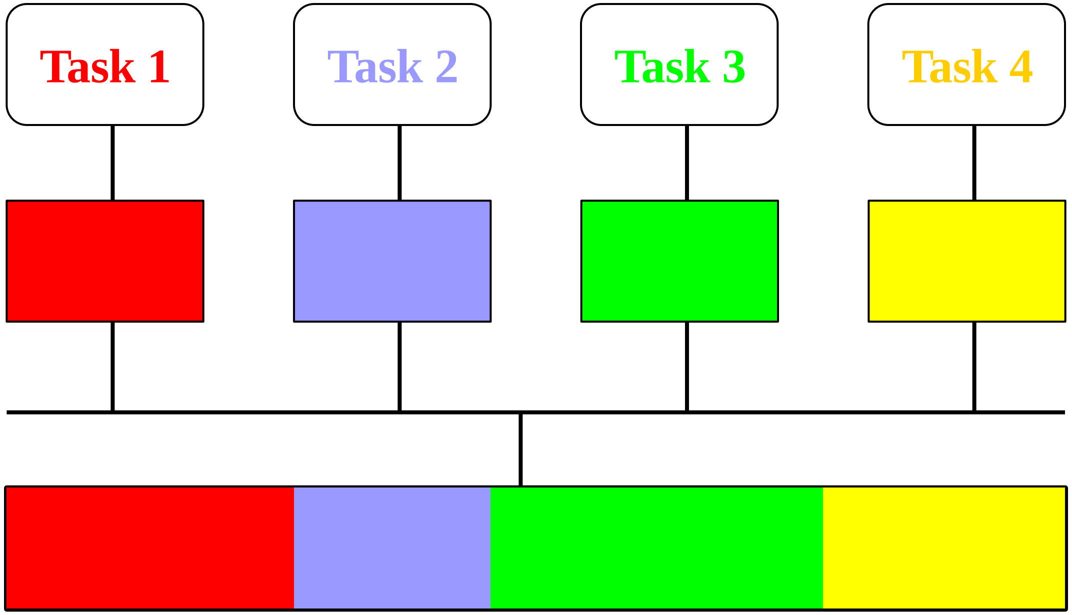
OUTLINE

- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

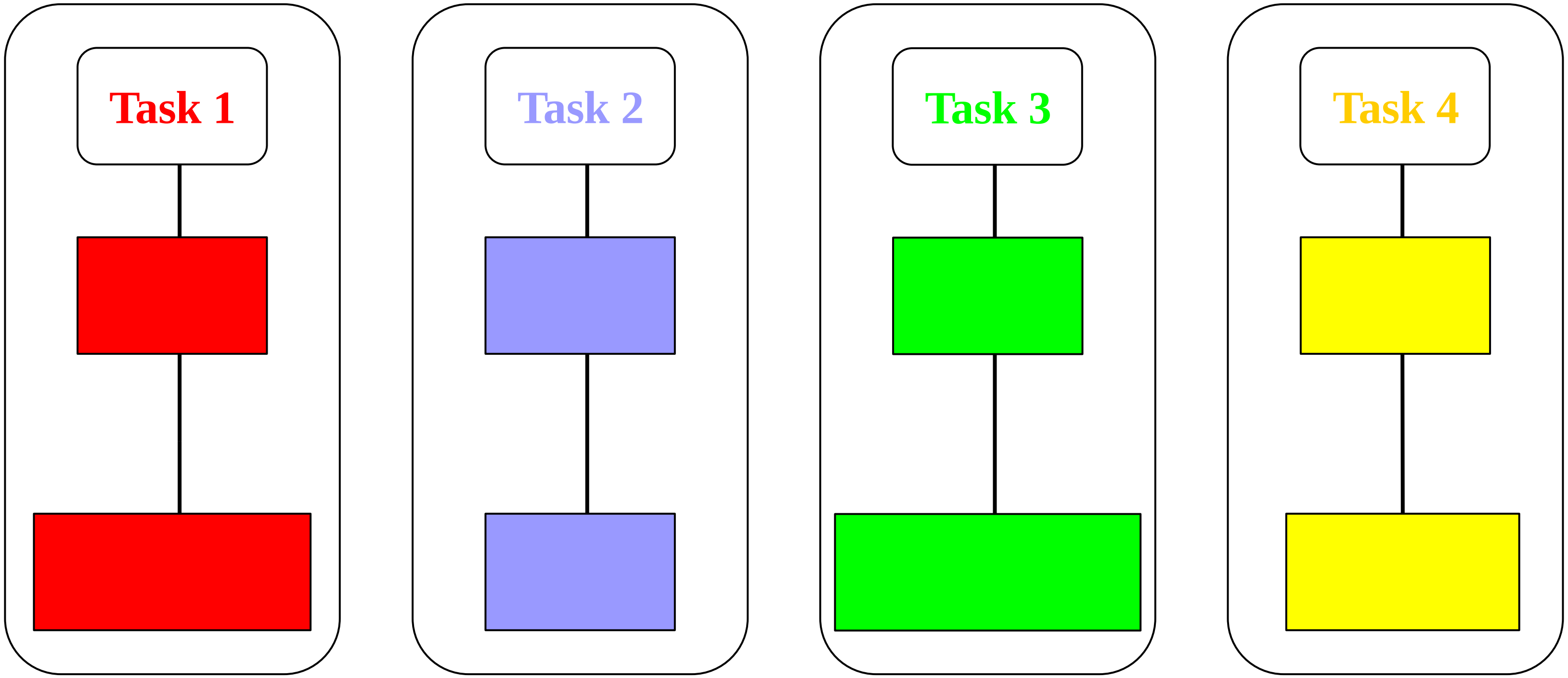
Cache analysis on multicore



Cache-Coloring: partitioning and isolation



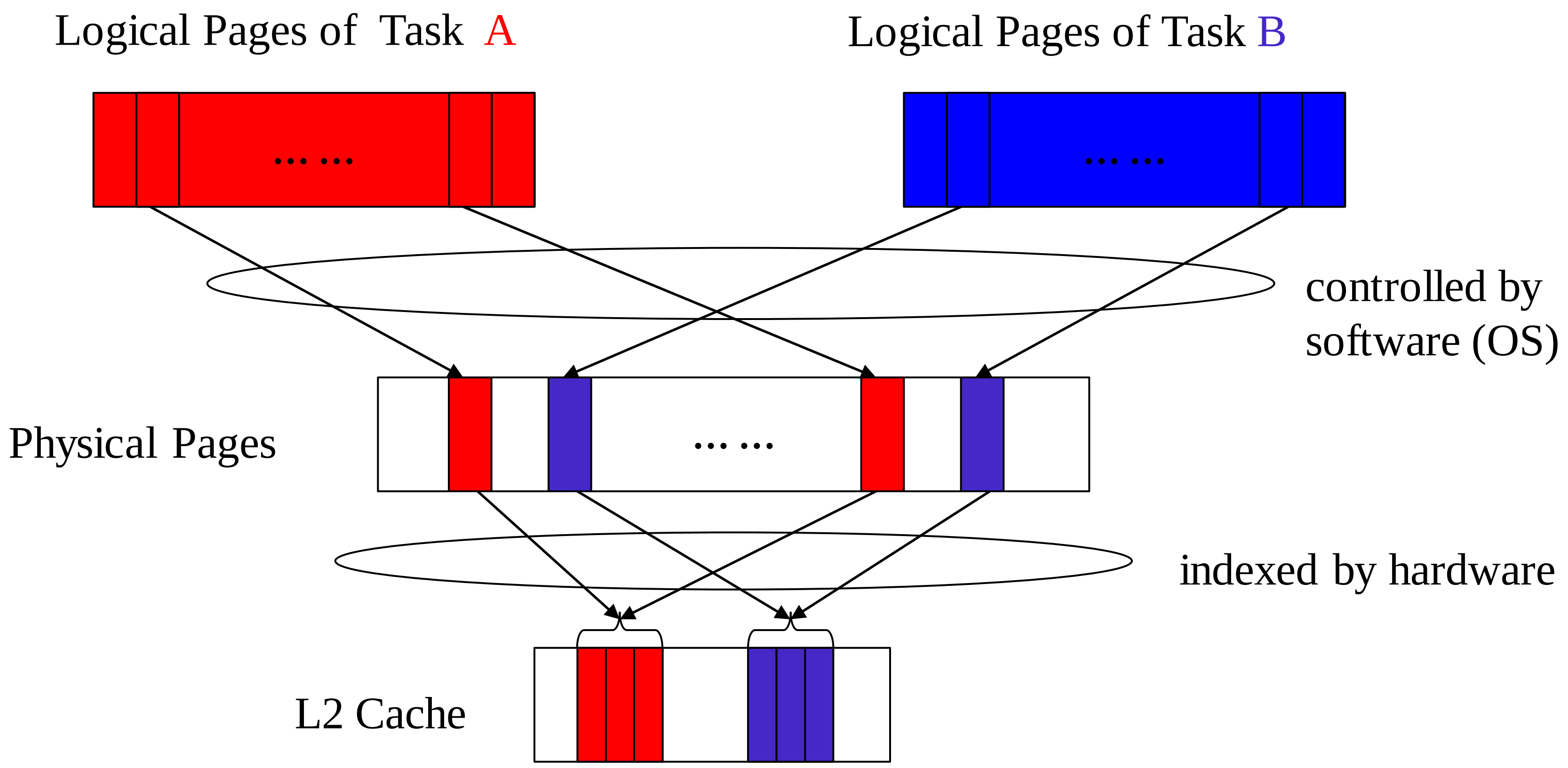
Cache-Coloring: partitioning and isolation



WCET can be estimated using static techniques for single processor platforms (for the given portion L2 cache)

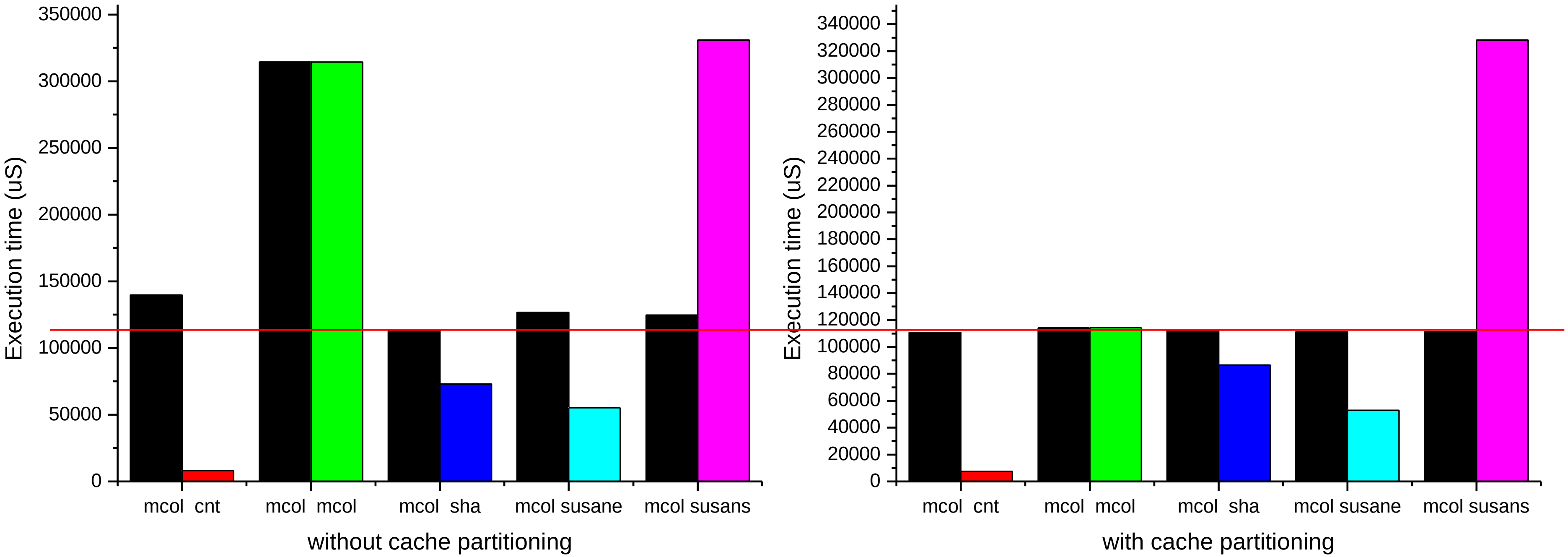
Cache-Coloring: partitioning and isolation

- E.g. LINUX – Power5 (16 colors)



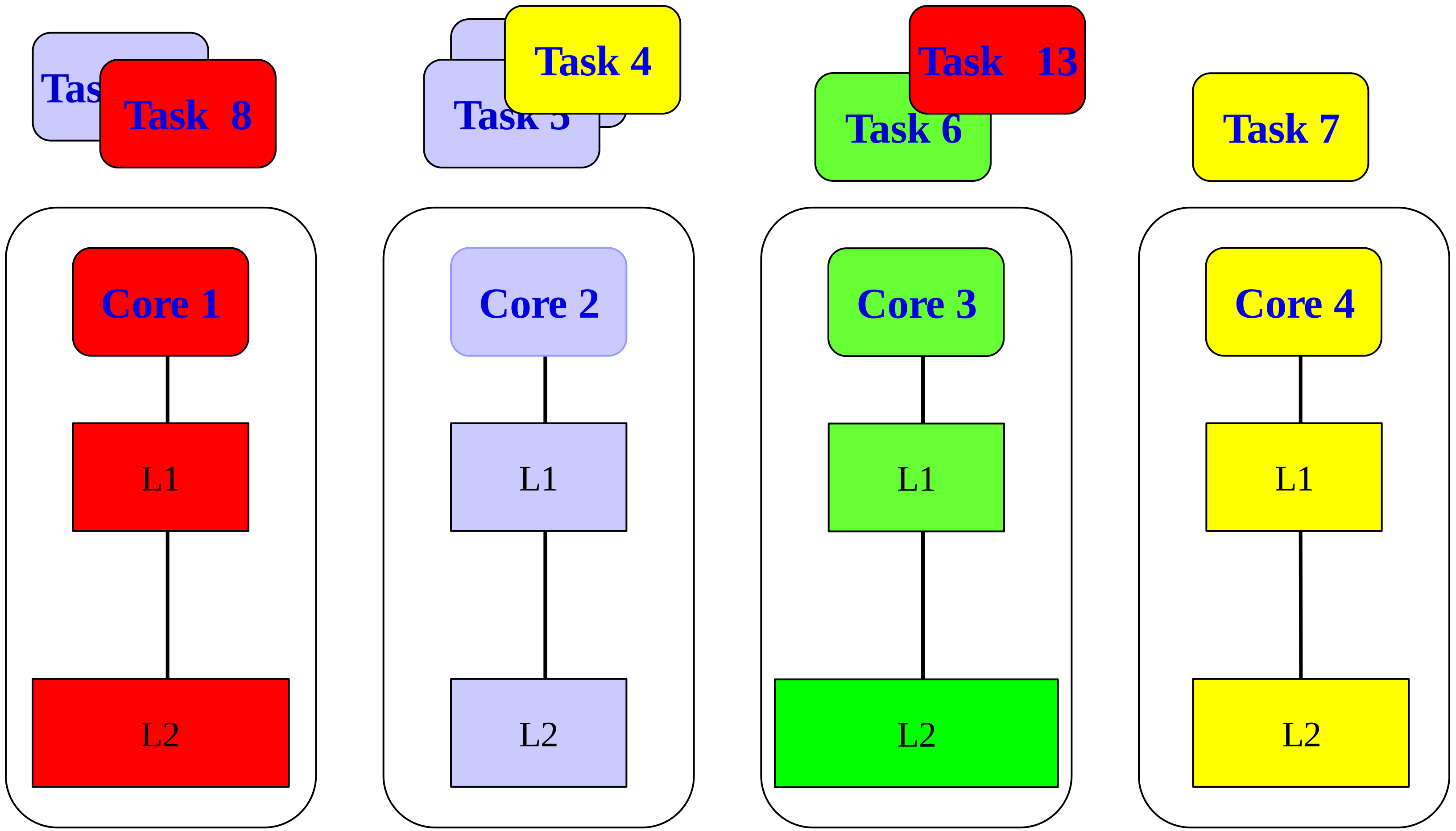
An Experiment on a LINUX machine with 2 cores

with Cache Coloring/Partitioning [ZhangYi et al]



What to do when #tasks > #cores ?

Task partitioning



What to do when #tasks > #cores ?

Cache-Aware Scheduling and Analysis for Multicores [EMSOFT 2009]

Main message:

- **“Isolation”: tasks of “same color” should not run at the same time**
- **The schedulability problem can be solved as an LP problem**

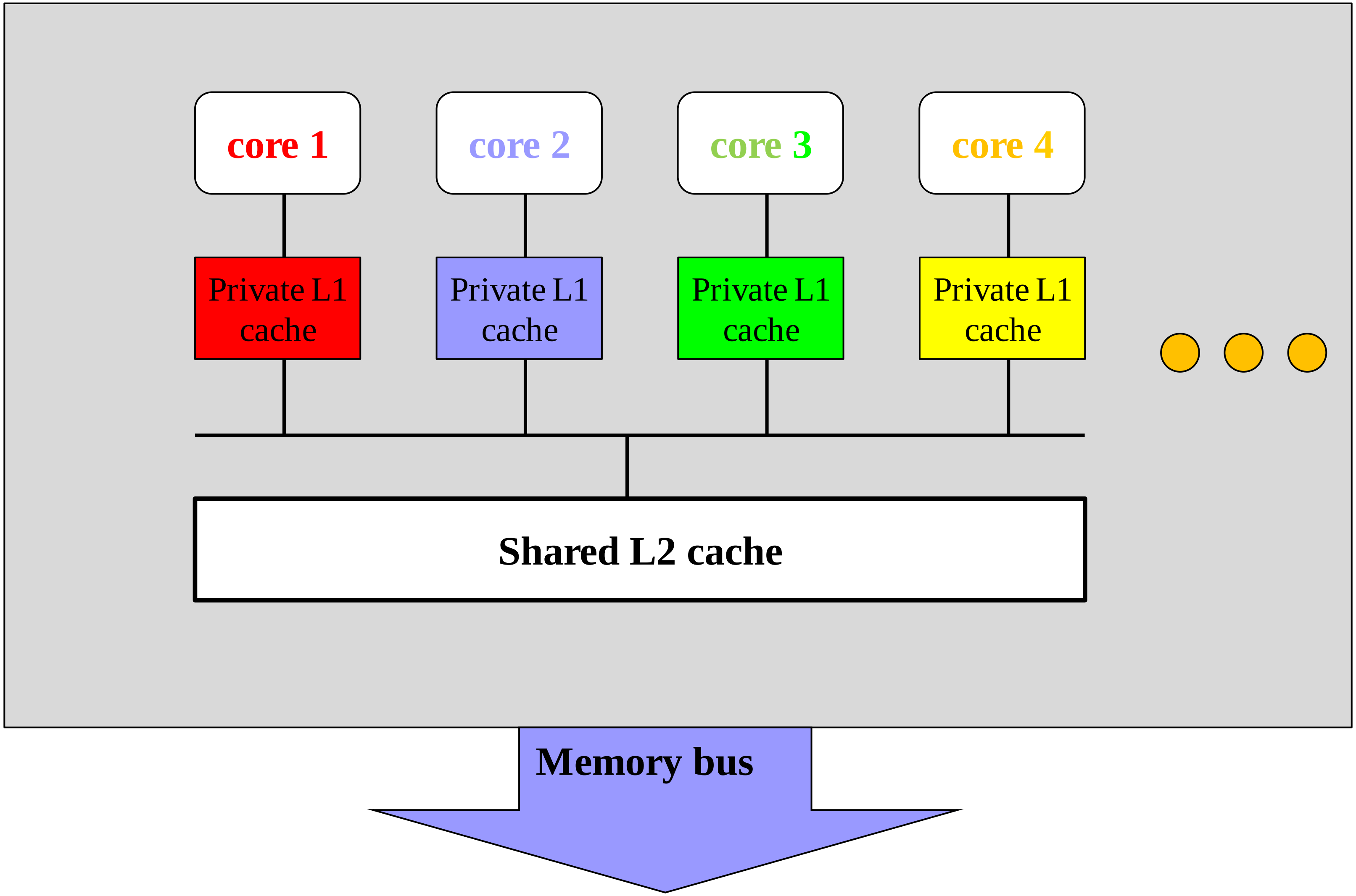
Task Partitioning & Scheduling

- Color assignment: assign cores with "cache colors"
  - Equally or according to some policy e.g. cores devoted to critical tasks get more colors
  - WCET analysis for tasks on different cores and colors

- Task assignment: partition tasks onto cores
  - Partition-based multiprocessor scheduling
  - Challenge: tasks may have different WECTs on different cores

- Global scheduling: need dynamic coloring (expensive without hardware support)

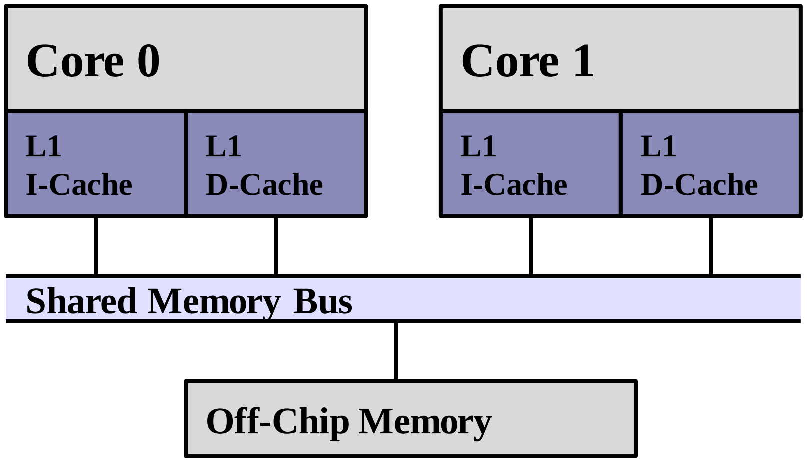
What happens when L2 cache miss?
-- extra delays due to bus contention



OUTLINE

- **Multicore Challenges**
  - Why and what are multicores?
  - What we are doing in Uppsala: CoDeR-MP
  - The timing analysis problem
- **Possible Solutions – Partition/Isolation**
  - Dealing with Shared Caches [EMSOFT 2009]
  - Dealing with Bus Interference [RTSS 2010]
  - Dealing with Core Sharing [RTAS 2010]

Bus Intererence Estimation & WCET Analysis



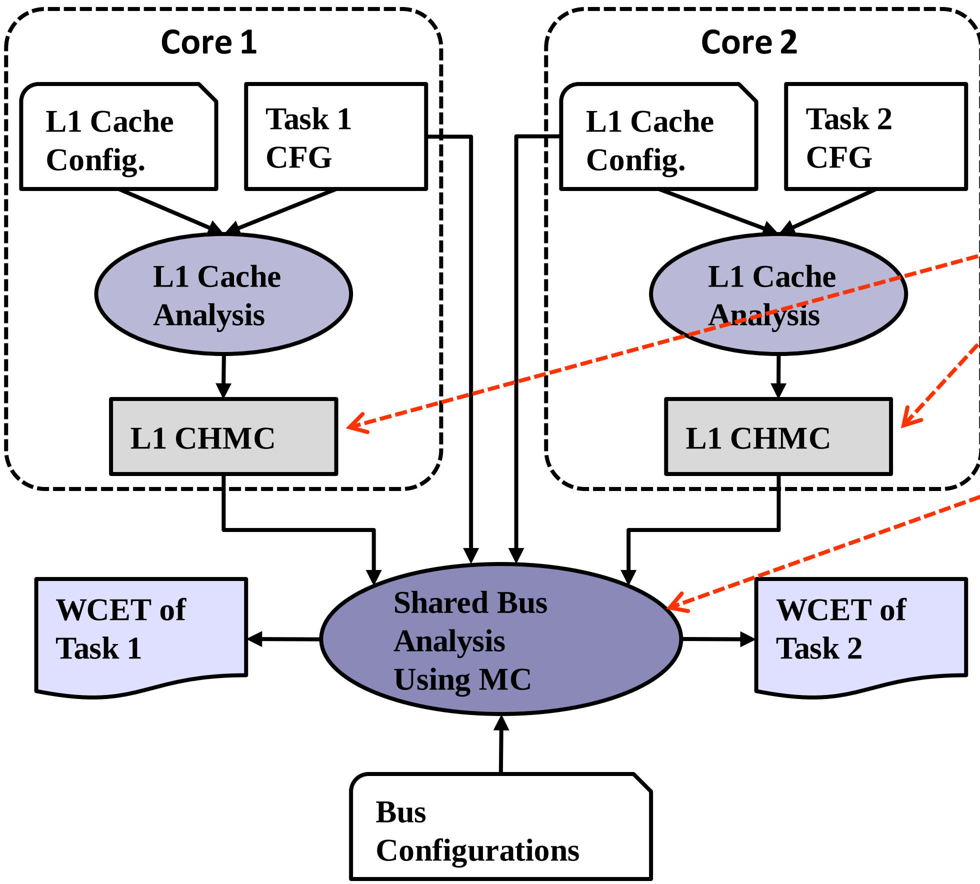
Duo-core processor with private L1 cache and shared memory bus

Combining Abstract Interpretation and Model Checking for Multicore WCET Analysis [RTSS 2010]

Basic Idea:

Construct a timed model -- describing all possible timed traces of bus requests, that are possible from each core

Combining Static Analysis & Model-Checking



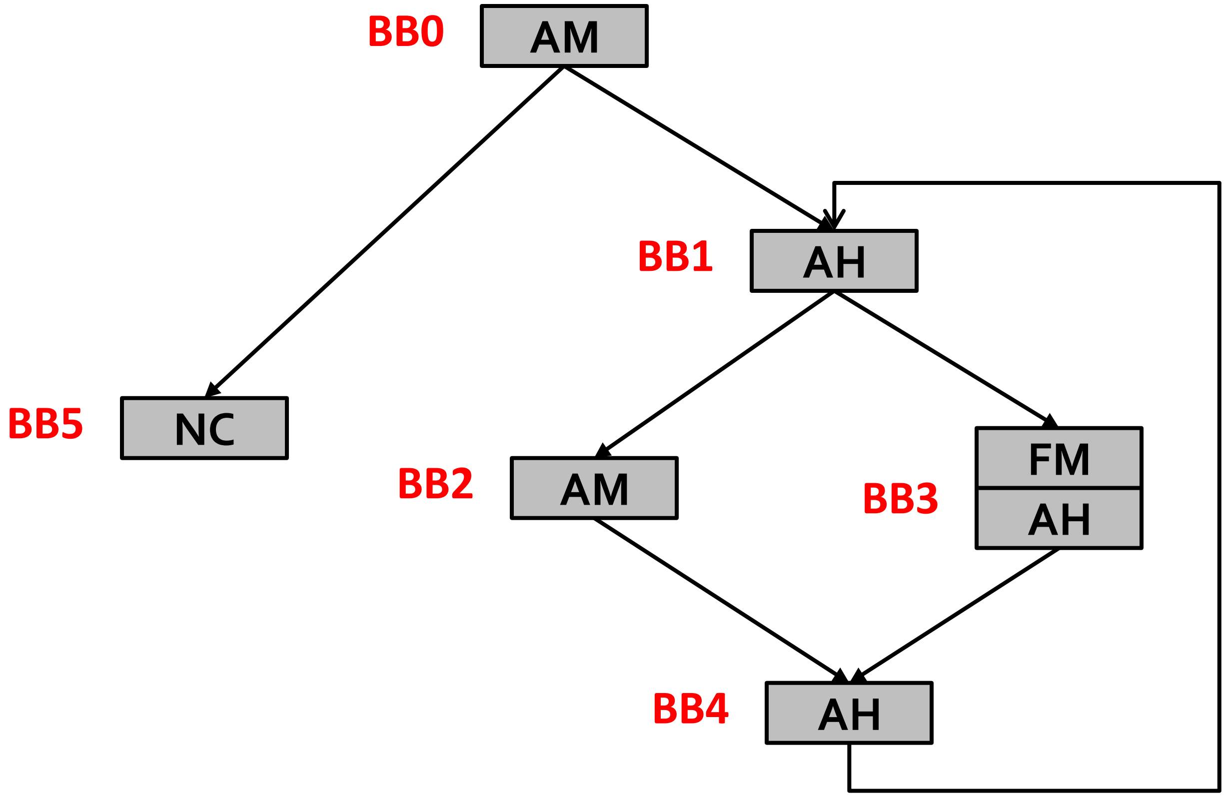
(1) Local cache analysis by abstract interpretation

(2) Construct a timed automaton for each program to model the precise timing information on when to access the shared bus

(3) Construct a timed automaton modeling the bus arbitration

(4) Explore the TA models using UPPAAL to get the WCETs

Example (CFG with CHMC info from AI analysis)

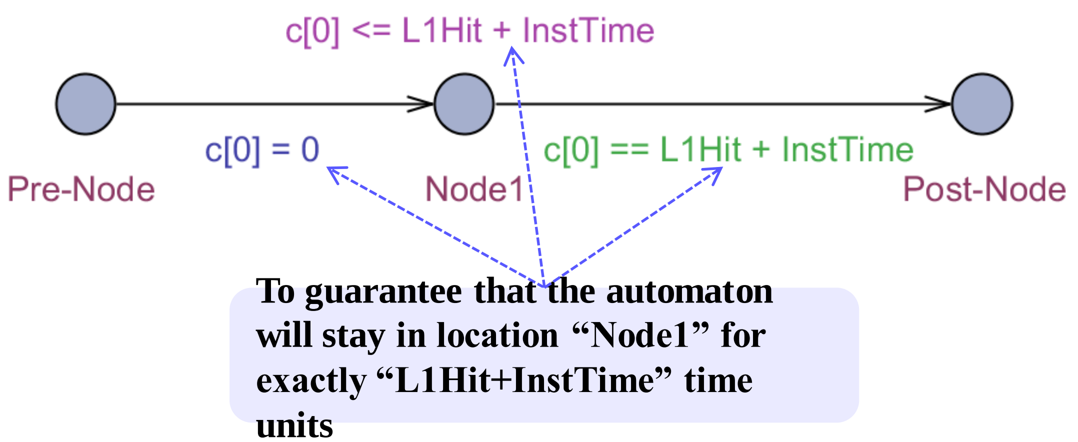


Private Cache Analysis by AI

- MUST analysis, classify instructions that are predicted as AH
- MAY analysis, classify instructions that are predicted as AM
- PERSISTENCE analysis, classify instructions that are predicted as FM
- Everything else as Not "Classified (NC)"

From CFG with CHMC to Timed Automata

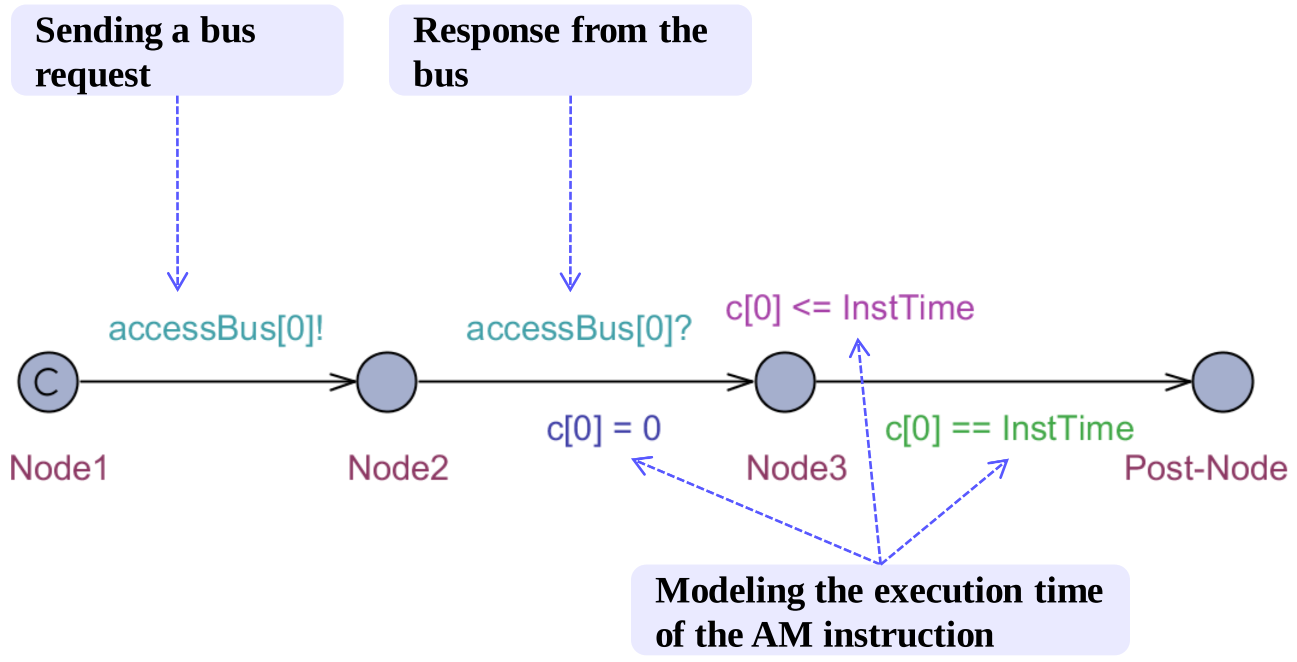
- Modeling AH instructions
  - If an instruction is AH, it never access the bus, so we only model the L1 Cache access time and the instruction execution time



c[0]: a clock variable used for core-0 to model the elapse of time
L1Hit: the delay of a L1 cache hit
InstTime: the execution time of an instruction

From CFG with CHMC to Timed Automata

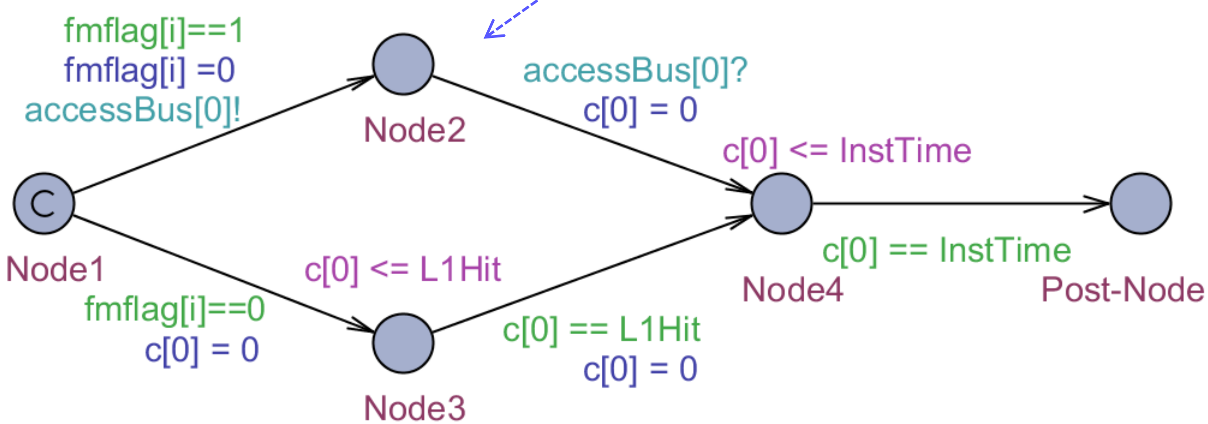
- Modeling AM instructions
  - An AM instruction is guaranteed to access the shared bus, so we model bus access behavior and instruction execution



From CFG with CHMC to Timed Automata

- Modeling FM instructions
  - For an FM instruction, one should distinguish between the first reference and the other references

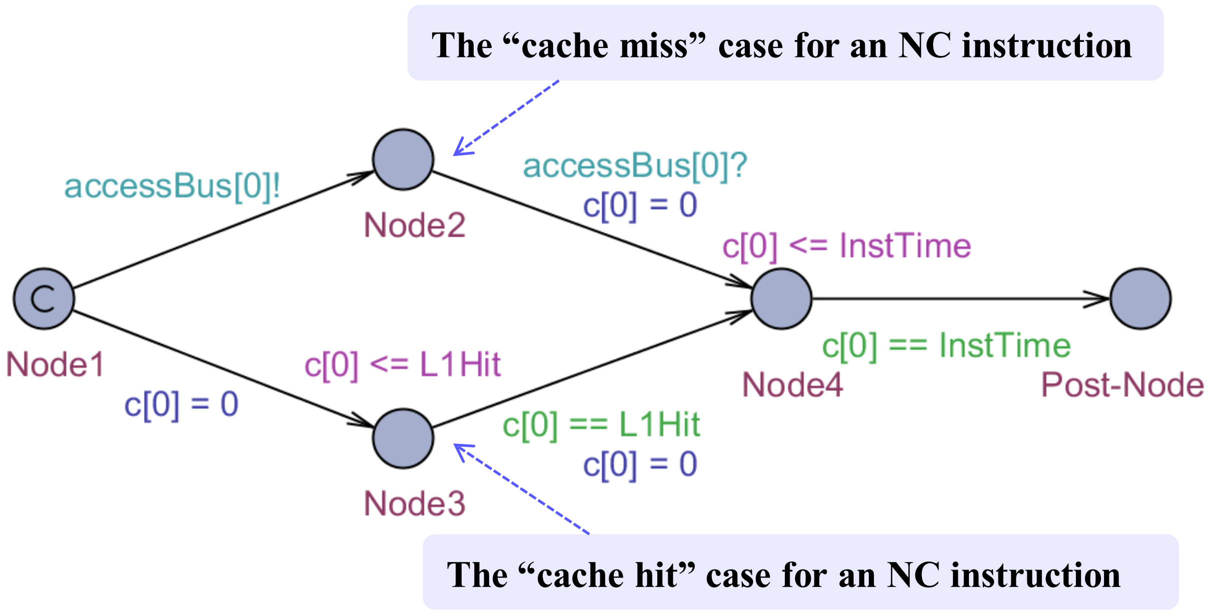
The upper path models the first reference to the instruction, which is a cache miss (access bus)



The lower path models the other references to the instruction, which are cache hits (do not access bus)

From CFG with CHMC to Timed Automata

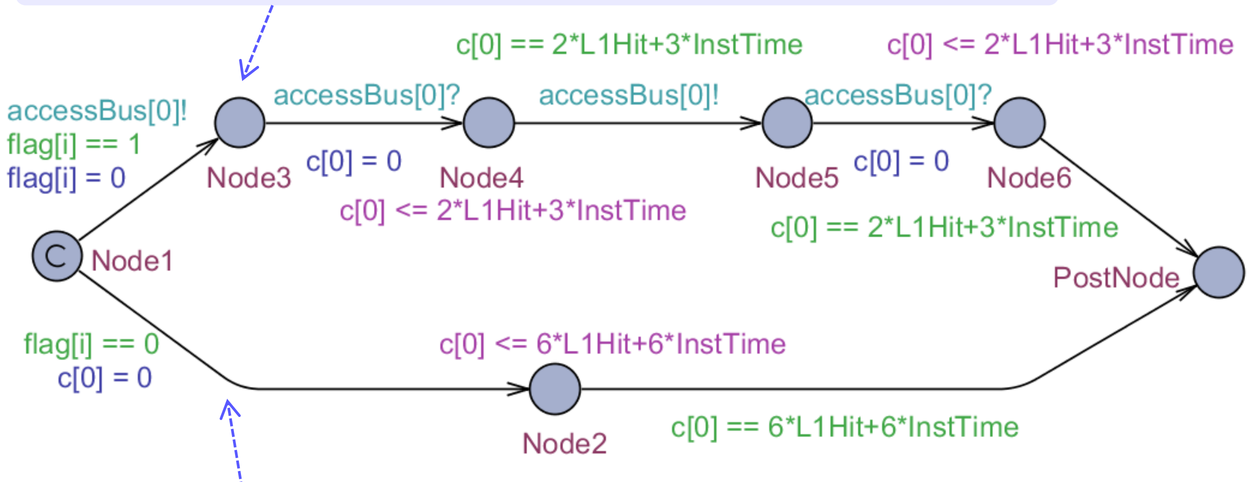
- Modeling NC instructions
  - So for NC instructions, we have to model both possibilities of cache misses and cache hits, and let the model checker to explore them



From CFG with CHMC to Timed Automata

- Optimization by grouping
  - To reduce state space by reducing the number of locations and edges, we grouping consecutive FM or AH instructions
  - Given a sequence <*FM, AH, AH, FM, AH, AH*>

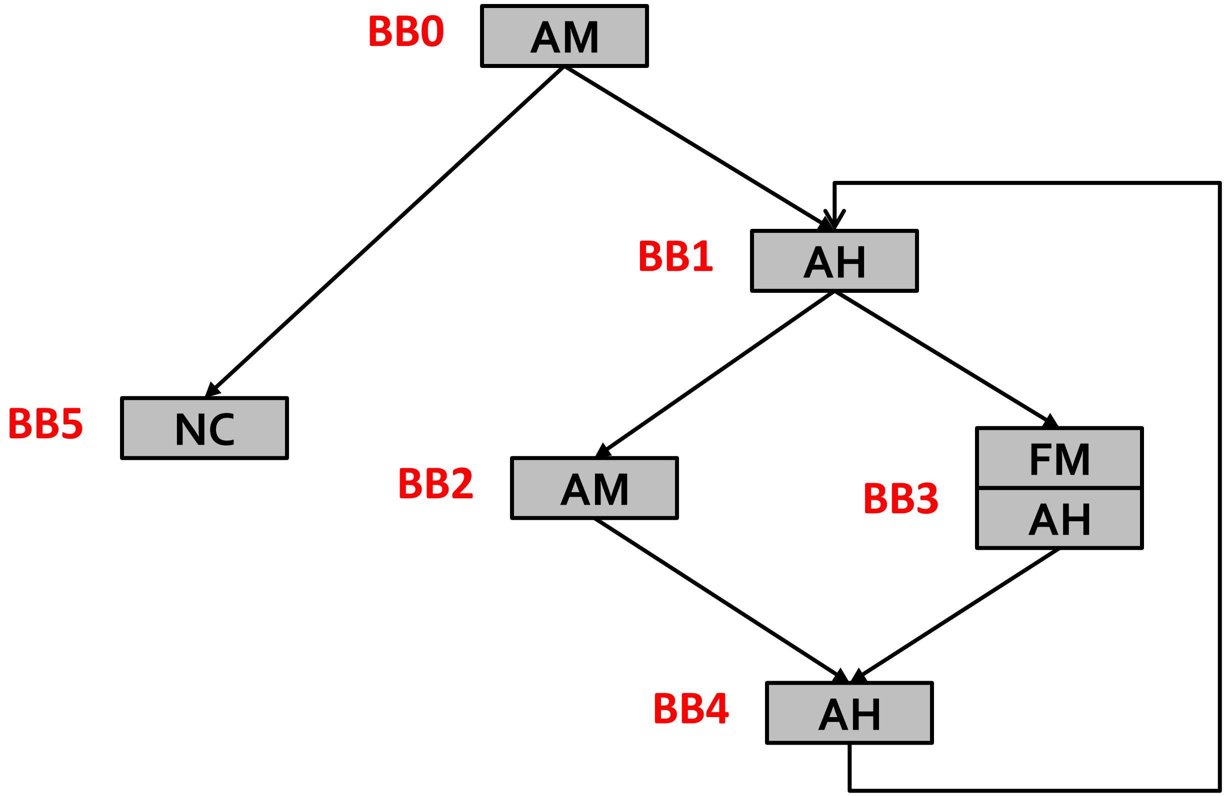
The upper path models the first time the sequence is executed



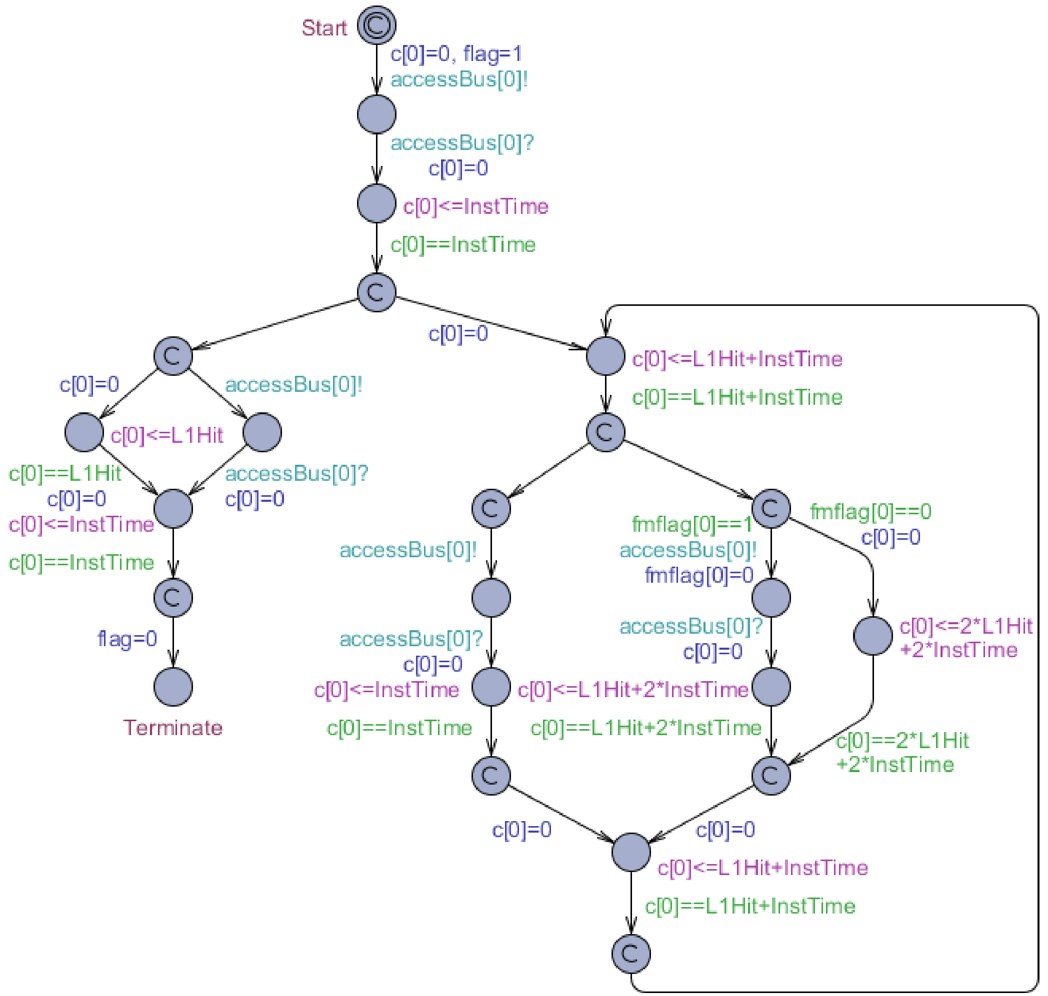
The lower path models all but the first time the sequence is executed

Without grouping:
12 locations
With grouping:
6 locations
(PostNode not included)

Example (CFG with CHMC info from AI analysis)

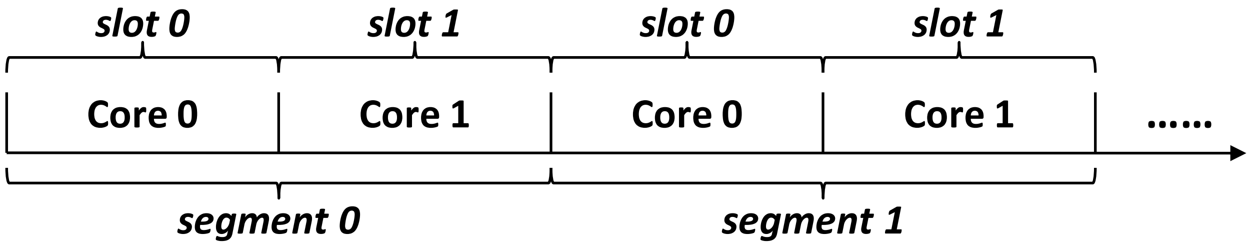


The Timed Automaton Describing "Bus Interference"



Modeling the Shared Bus

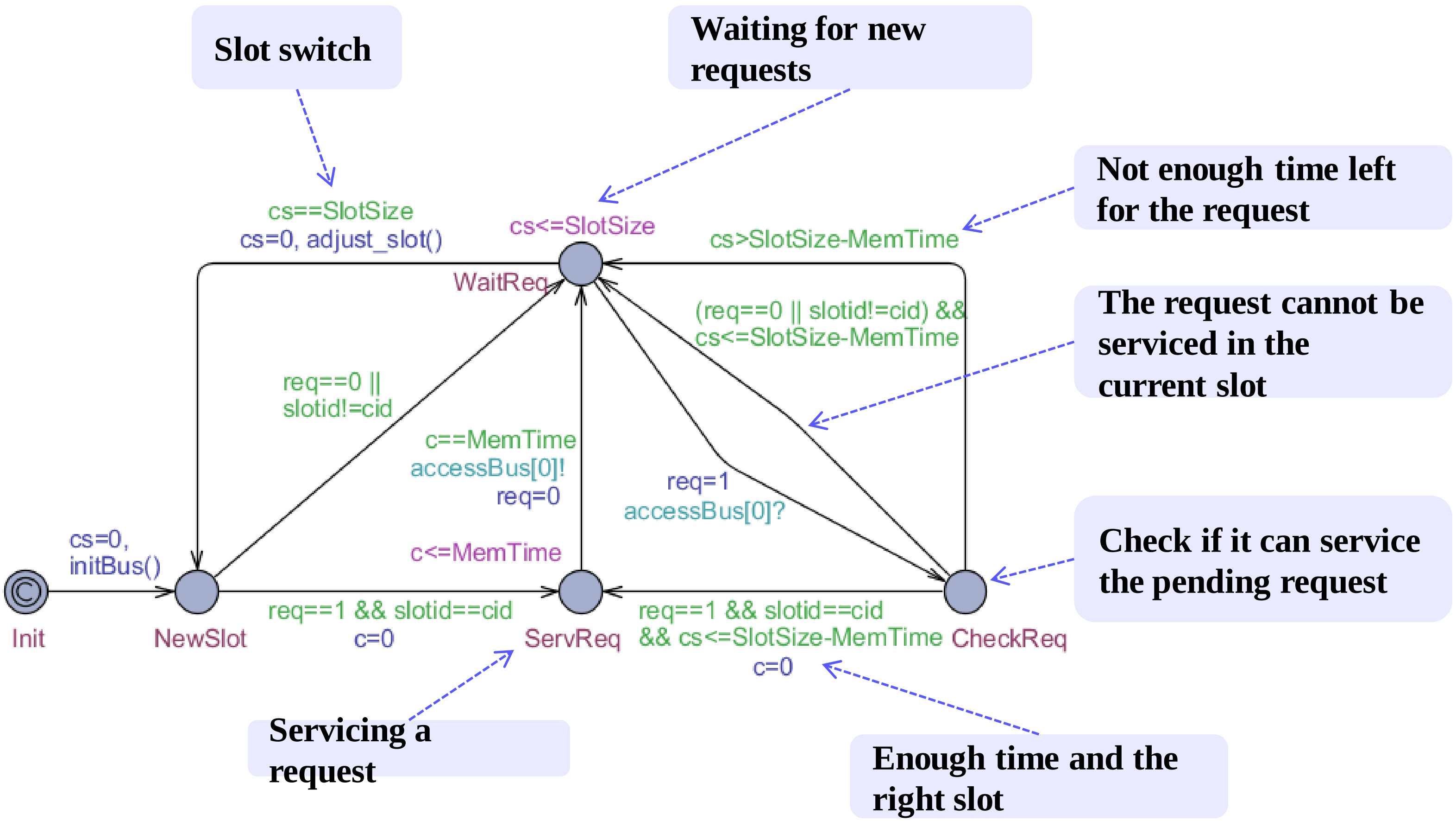
- Example: TDMA bus schedule



  - The bus schedule is composed of consecutive *segments*
  - *Segments* are divided into *slots*, where each *slot* is assigned to one core

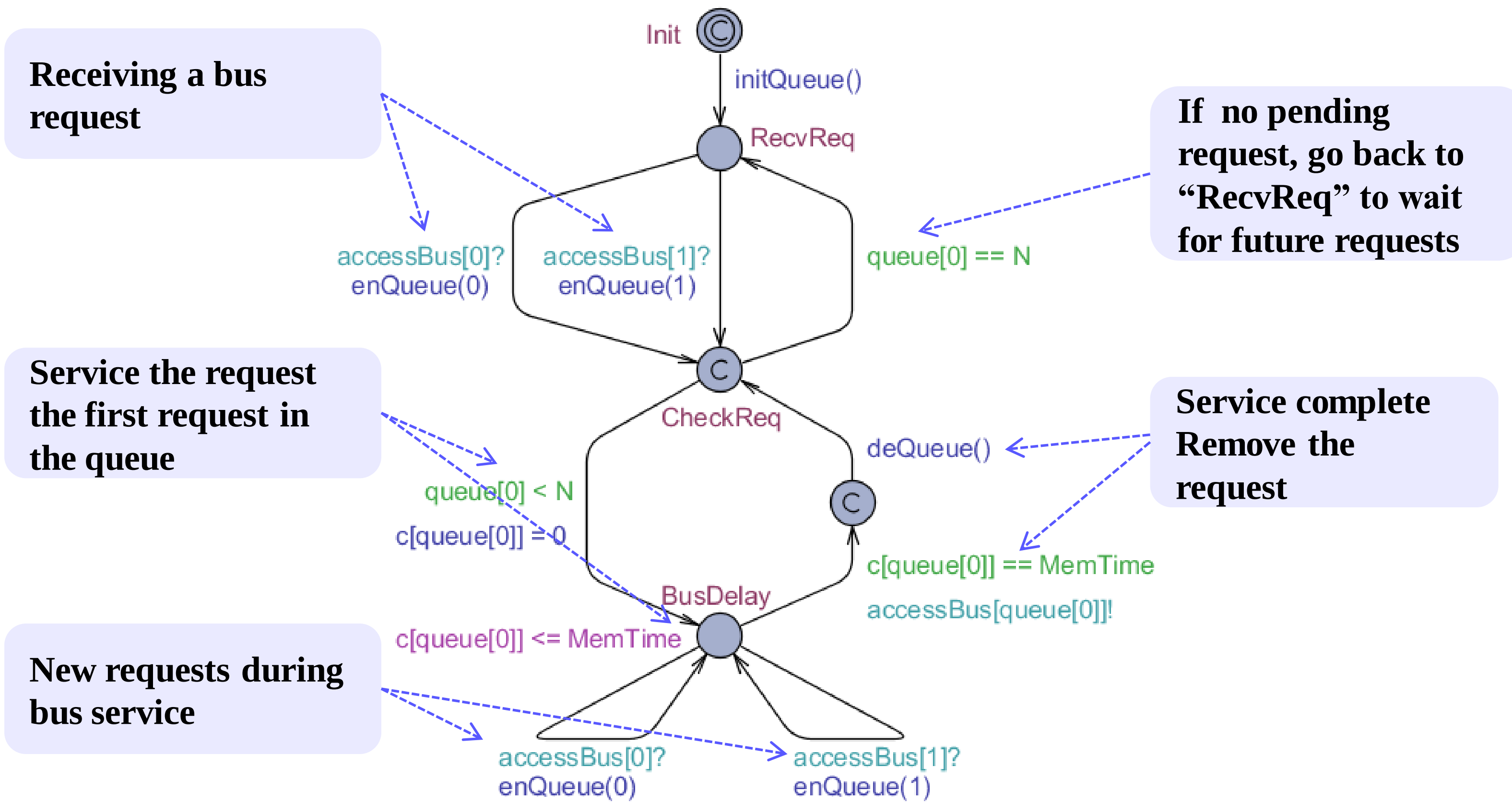
Modeling the TDMA Bus

- Timed automaton for the TDMA bus



Modeling the FCFS Bus

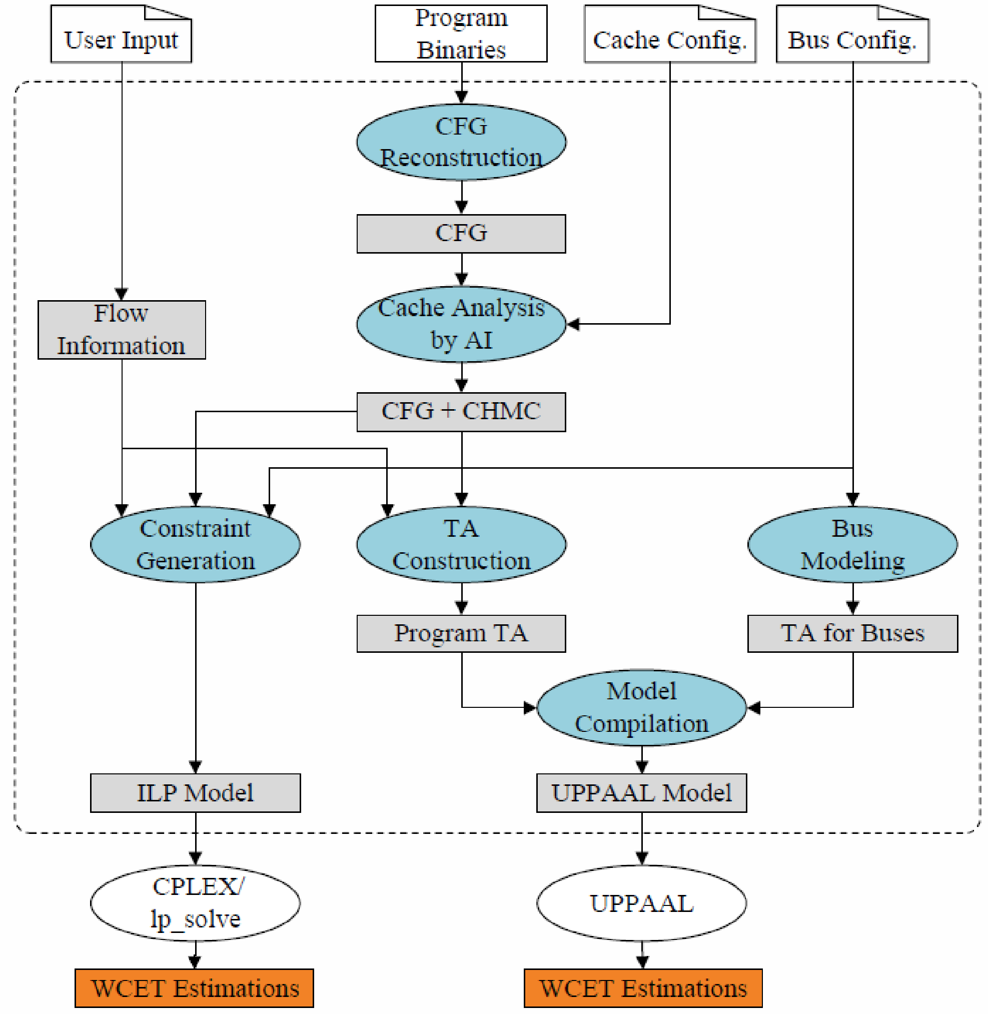
- A work-conserving non-preemptive FCFS bus



Putting All Together

- Now, we have
  - TA models for the programs running on all cores, describing all bus requests annotated with timing info, that are possible from the cores
  - TA model for a given bus arbitration protocol e.g TDMA, FCFS, RR …
- WCET estimation
  - Let the UPPAAL model checker explore the network of TA models
  - The WCETs are extracted from the clock constraints within the UPPAAL model checker
- Scalability: for TDMA, it scales very well: the analysis can be done separately for each program and the bus schedule.

A Tool for Multicore WCET Analysis



Experiments and Evaluation

- WCET Benchmark programs (Maladalen)

Name	Description	# instructions
bs	Binary search algorithm for an array	78
edn	Finite Impulse Response (FIR) filter calculations	896
fdct	Fast Discrete Cosine Transform	647
insertsort	Insertion sort on a reversed array	106
jfdctint	Discrete Cosine Transformation on a pixel block	691
matmult	Matrix multiplication	287

Results for the TDMA Bus

- System configurations
  - Duo-core or 4-core systems
  - L1 Cache size = 2KB,
  - Cache associativity = 4
  - Cache line size = 8B
  - L1 hit latency = 1 cycle
  - Instruction execution = 1 cycle
  - Bus service time = 40 cycles
  - Two different slot sizes: 100 cycles, 200 cycles

Results for the TDMA Bus

- The WCET of each program can be calculated independently for the TDMA bus
- The worst-case bus delay scenario
  - A bus request arrives in the slot assigned to it, but finds that there are only 39 cycles left, which is just not enough to serve the request
  - For slot size 100, worst-case delay = 39 + 100 + 40 = 179
  - For slot size 200, worst-case delay = 39 + 200 + 40 = 279
- Improvement
  - $(WCET_{AI+WC} / WCET_{AI+MC} - 1)$
  - Describes how much our approach can tighten compared to assuming worst-case bus delay

Results for the TDMA Bus

- Results for a duo-core system with slot size 100

Programs	WCET		Improvement
	AI + MC	AI + Worst-Case	
bs	8,282	14,644	77%
edn	9,219,082	16,565,100	80%
fdct	268,882	479,946	78%
insertsort	21,041	29,702	41%
jfdctint	315,882	563,936	79%
matmult	151,241	174,390	15%
Average			62%

Results for the TDMA Bus

- Results for a duo-core system with slot size 200

Programs	WCET		Improvement
	AI + MC	AI + Worst-Case	
bs	8,484	22,444	165%
edn	9,207,282	25,756,000	180%
fdct	267,282	742,646	178%
insertsort	21,282	40,302	89%
jfdctint	314,564	873,336	178%
matmult	150,841	203,090	35%
Average			138%

Results for the TDMA Bus

- Results for a 4-core system with slot size 100

Programs	WCET		Improvement
	AI + MC	AI + Worst-Case	
bs	16,082	30,244	88%
edn	18,428,441	34,946,900	90%
fdct	529,682	1,005,350	90%
insertsort	31,641	50,902	61%
jfdctint	624,482	1,182,740	89%
matmult	179,241	231,790	29%
Average			75%

Results for the TDMA Bus

- Results for a 4-core system with slot size 200

Programs	WCET		Improvement
	AI + MC	AI + Worst-Case	
bs	16082	53644	234%
edn	18404164	62519600	240%
fdct	529682	1793450	239%
insertsort	32082	82702	158%
jfdctint	628164	2110940	236%
matmult	179241	317890	77%
Average			197%

Results for the FCFS Bus

- System configurations
  - Duo-core system
  - L1 Cache size = 8KB
  - Cache line size = 8B
  - Cache associativity = 4
  - L1 cache hit latency = 1 cycle
  - Instruction execution time = 1 cycle
  - Bus service time = 40 cycles

Results for the FCFS Bus

- Evaluation method
  - Grouping the six benchmark programs into two task sets
  - {bs, edn, fdct} and {insertsort, jfdctint, matmult}
  - Each task set is allocated on one core
  - The tasks within the same task set are statically scheduled

Schedules	Core-0	Core-1
S1	edn, bs, fdct	matmult, insertsort, jfdctint
S2	bs, fdct, edn	matmult, insertsort, jfdctint
S3	fdct, edn, bs	matmult, insertsort, jfdctint
S4	edn, bs, fdct	insertsort, jfdctint, matmult
S5	fdct, bs, edn	Jfdctint, matmult, insertsort
S6	fdct, bs, edn	matmult, insertsort, jfdctint
S7	edn, bs, fdct	jfdctint, insertsort, matmult
S8	fdct, edn, bs	Jfdctint, matmult, insertsort

Results for the FCFS Bus

- The worst-case bus delay scenario
  - A request req_i arrives when the bus is servicing a request from the other core which is issued immediately before req_i
  - Given the above system configurations, the worst-case bud delay for the FCFS bus is 80 cycles (two times the bus service time)

Results for the FCFS Bus

Programs	WCET (AI + MC)		WCET AI+Worst-Case	Maximal Impr.	Average Impr.
	Minimal	Average			
bs	3,802	4,319	6,922	82%	67%
edn	240,267	246,970	276,068	15%	12%
fdct	37,573	44,620	63,453	69%	46%
insertsort	14,968	15,763	19,208	28%	23%
jfdctint	40,153	48,056	67,793	69%	45%
matmult	138,406	140,117	145,977	5%	4%
Average improvement for all programs					33%

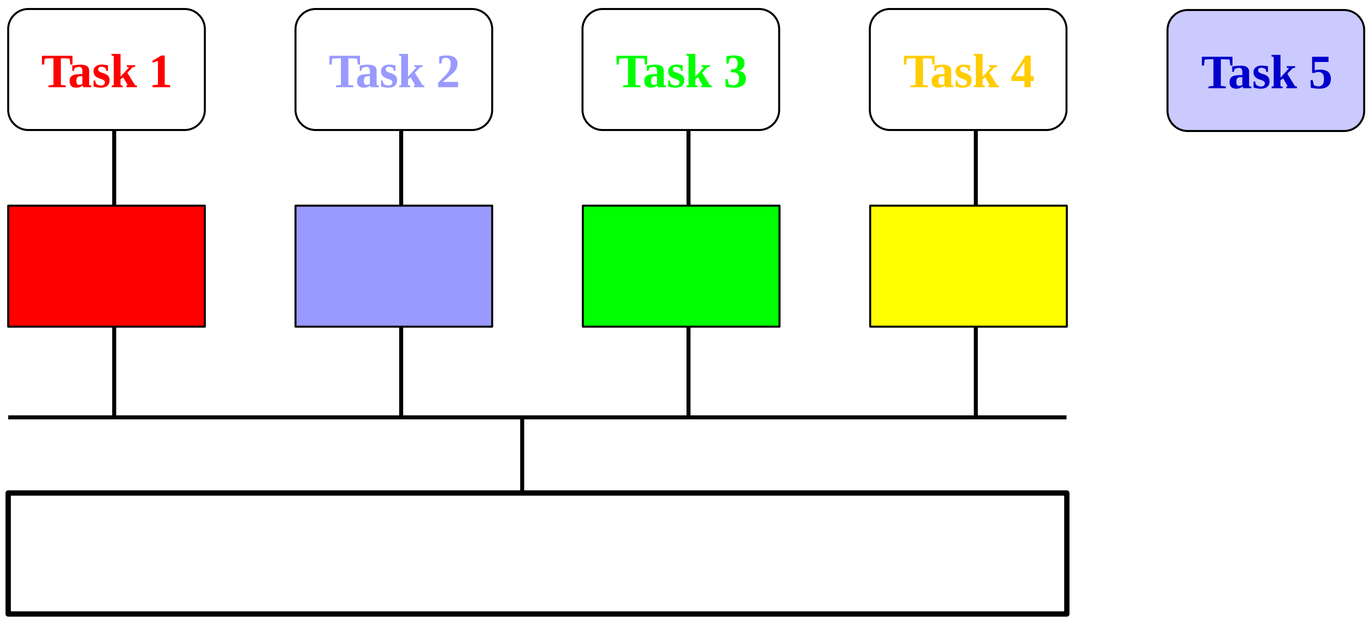
Now, assume that we have a "safe WCET bound" for each task

Remember, we need to:

- "partition" the shared caches
- "partition" the shared memory bus

The multicore challenge: Scheduling & schedulability analysis

- #cores < #tasks



Dealing with Shared Cores

Multiprocessor Scheduling