CPS Research

Dionisio de Niz and Sagar Chaki

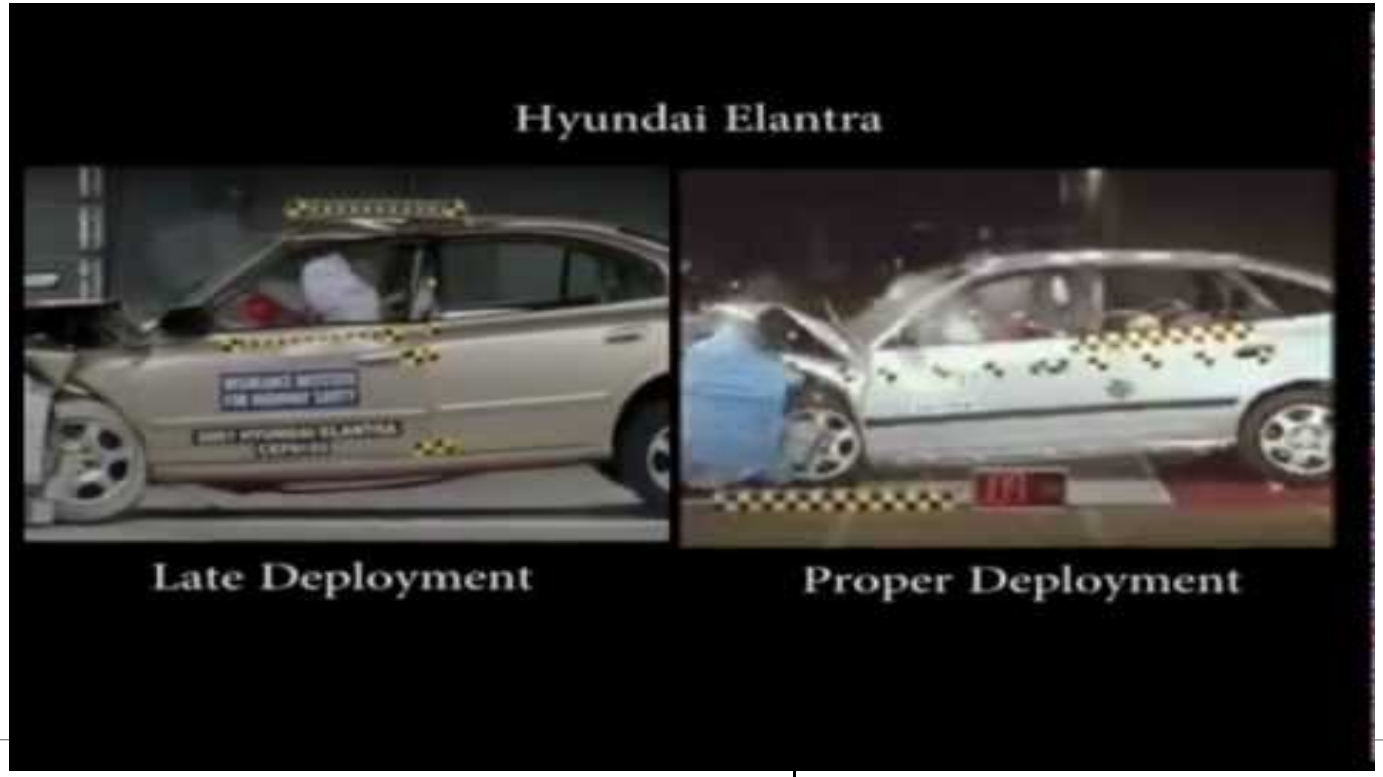
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0337

Verification of CPS

CPS Concerns

- Logic: correct value
- Timing: at the right time
- Scalability: for real-size systems



Approach

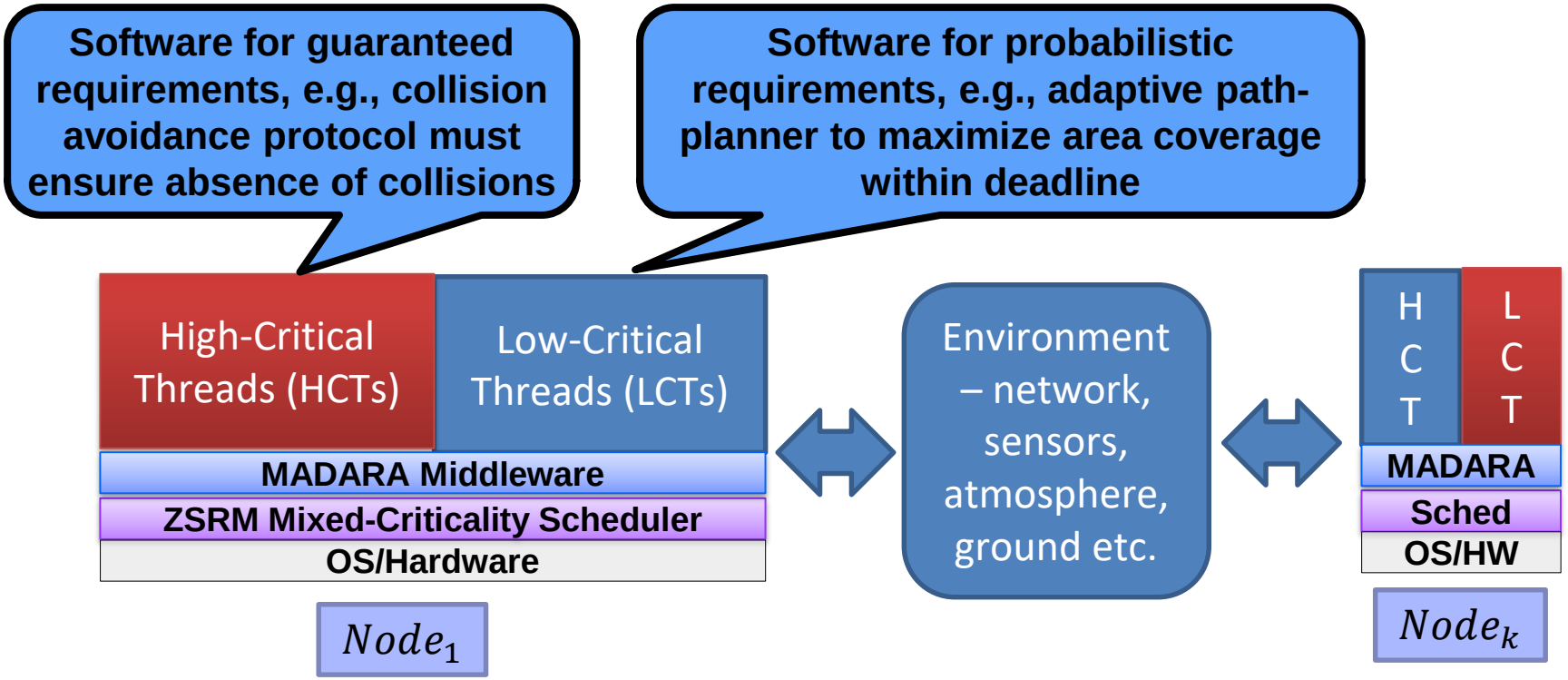
Logical Verification
- Model Checking
- Source-Code Logical Verification: CBMC, FRAMA-C

Timing Verification
- Real-Time Scheduling
- Variety of Applications: Mixed-Criticality, Distributed Pipelines
- Complex Hardware: Multicore Processors

Scalable Combination
- Reduced Interleavings: In Rate-Monotonic Ignore lower-priority threads
- Verified Timing Guarantees of Scheduler Code: Time as ghost variables

Improved Scalability
- Domain Specific Language: constrained executable
- Distributed Shared-Variables Middleware: synchronous computation
- Statistical Model Checking:
  - Montecarlo Simulations
  - Important Sampling / Semantic Important Sampling

Project 1: Distributed Adaptive Real-Time



Research Thrusts

- Proactive Self-Adaptation
- Statistical Model Checking
- Real-Time Schedulability
- Functional Verification

Validation Thrusts

- Model Problem
- Workbench

DART Programming : AADL + DMPL

AADL : Architecture Analysis and Description Language

DMPL : DART Modeling and Programming Language

AADL : High level architecture + threads + real-time attributes

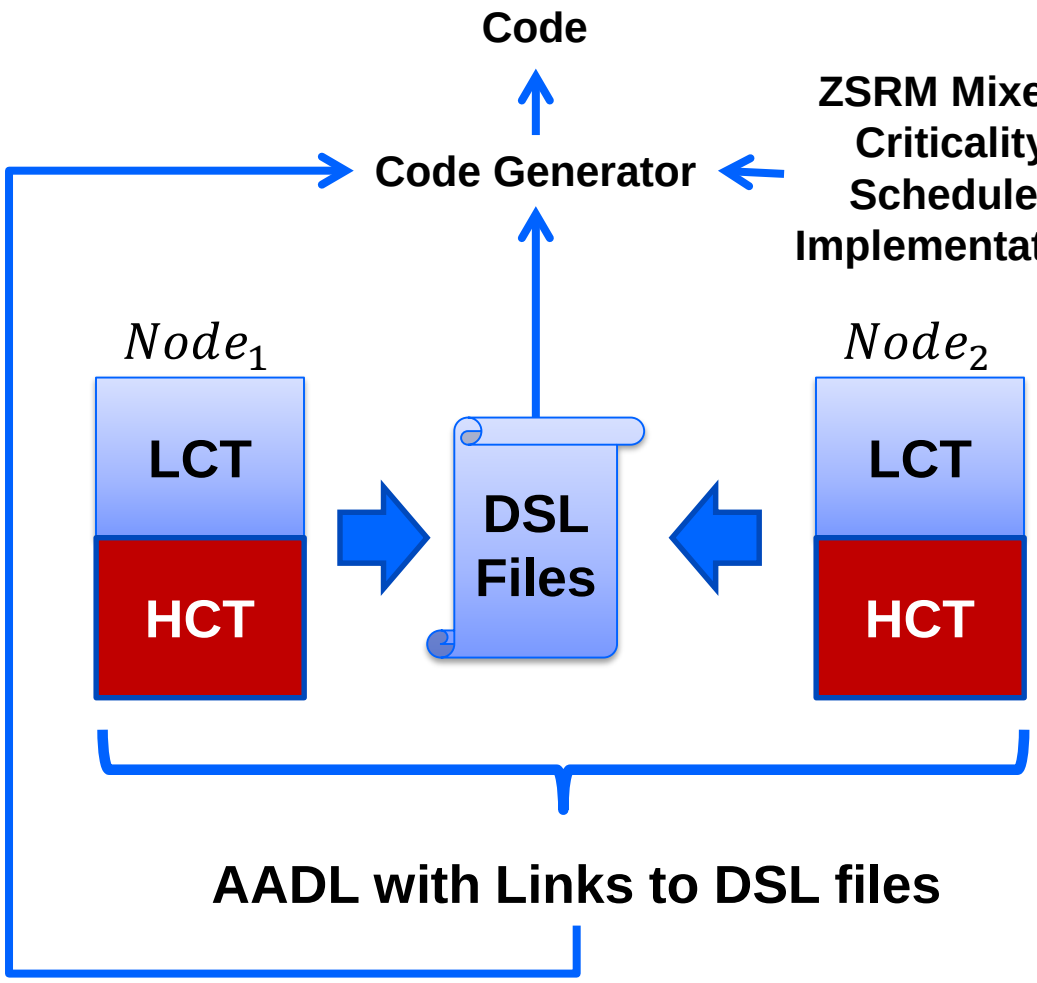
- Perform ZSRM schedulability via OSATE Plugin
- Generate appropriate DMPL annotations

Implemented as a DART Workbench. Happy to share.

DMPL : Behavior

- Roles : leader, protector
- Functions : mapped to real-time threads
  - Period, priority, criticality (generated from AADL)
  - Behavior : C-style syntax. Can call-out to arbitrary libraries.
- Functional properties (safety) : software model checking
- Probabilistic properties (expectation) : statistical model checking

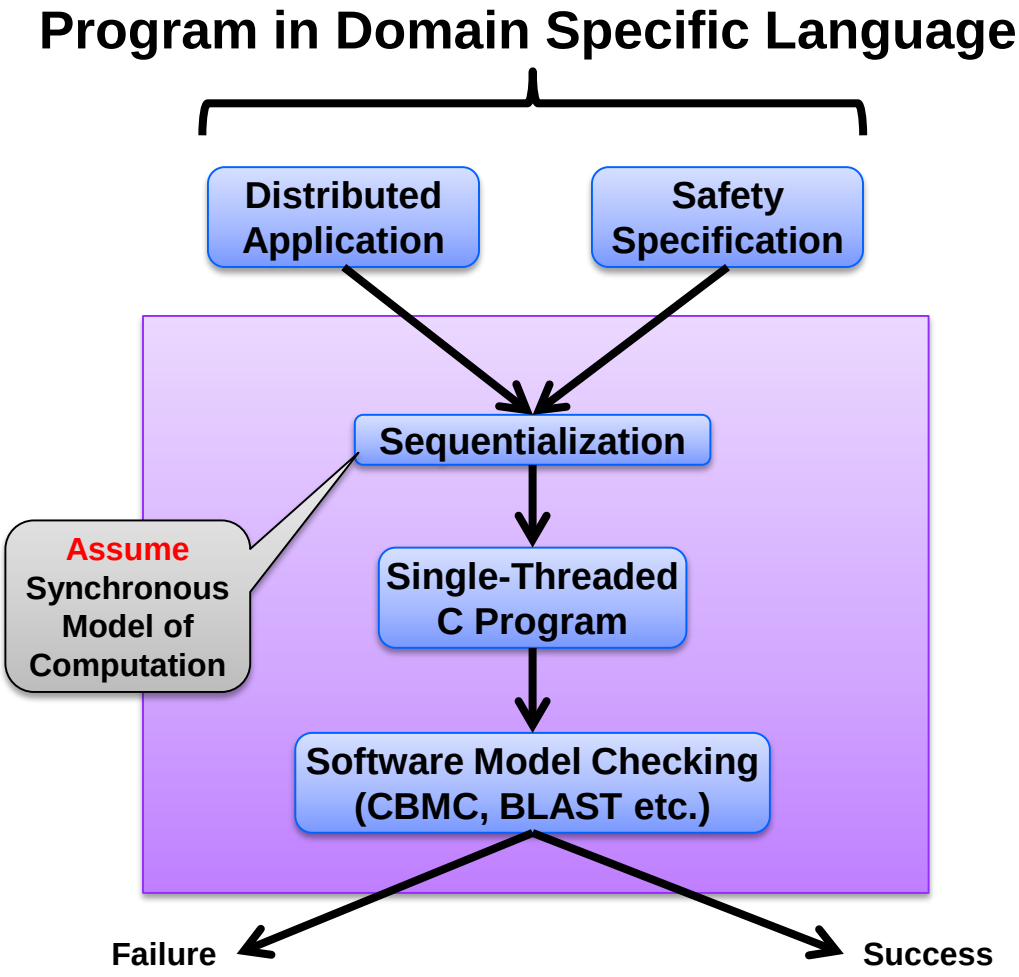
Real-Time Schedulability



Innovations:

1. Mixed-Criticality Scheduling under I/O
2. End-to-end Mixed-Criticality Scheduling

Verification



Model Checking

Automatic verification technique for finite state concurrent systems.

- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- ACM Turing Award 2007

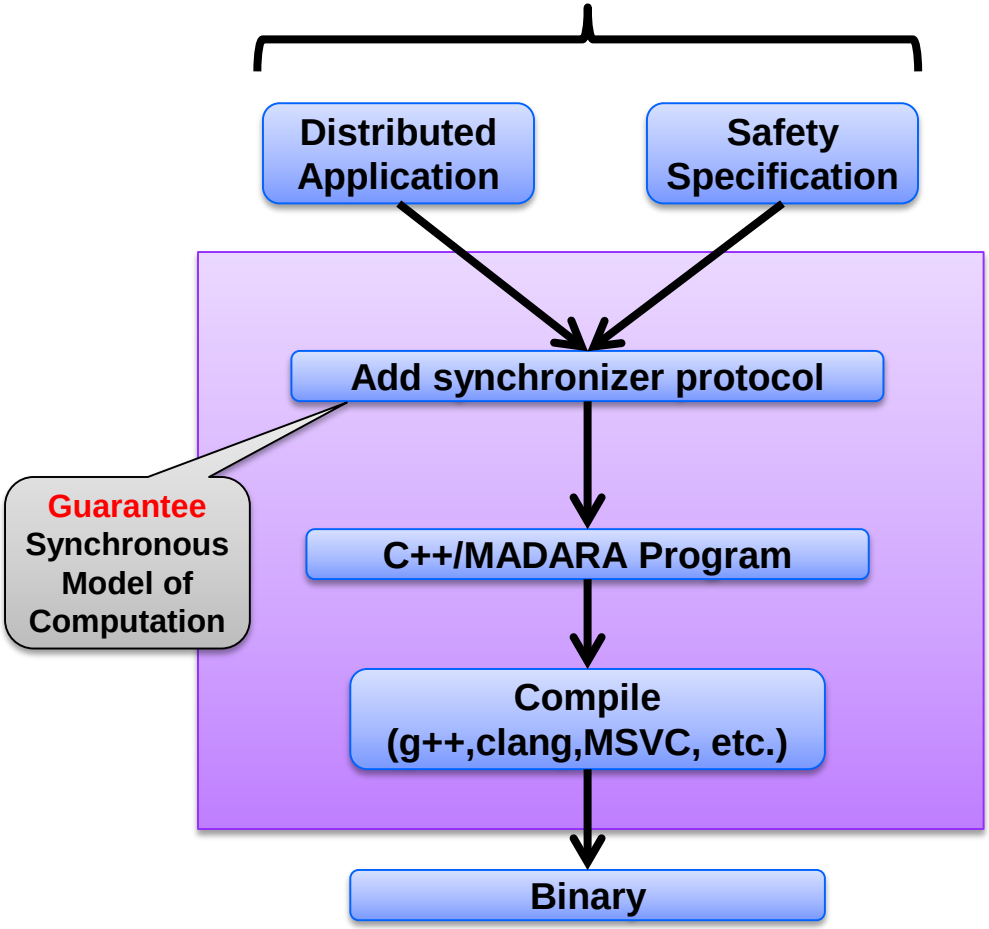
Specifications are written in propositional temporal logic. (Pnueli 77)

- Computation Tree Logic (CTL), Linear Temporal Logic (LTL), …

Verification procedure is an intelligent exhaustive search of the state space of the design

Code Generation

Program in Domain Specific Language



MADARA Middleware

A database of facts: $DB = Var \mapsto Value$

Node i has a local copy: DB_i

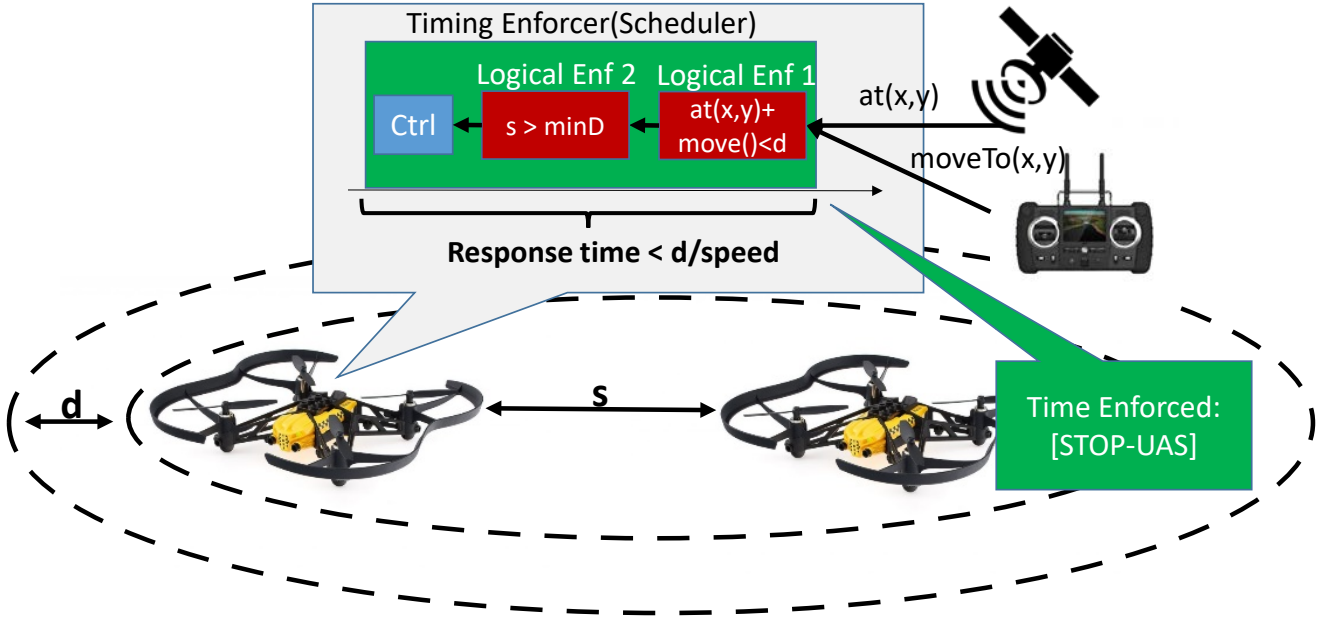
- update DB_i arbitrarily
- publish new variable mappings
  - Immediate or delayed
  - Multiple variable mappings transmitted atomically

Implicit "receive" thread on each node

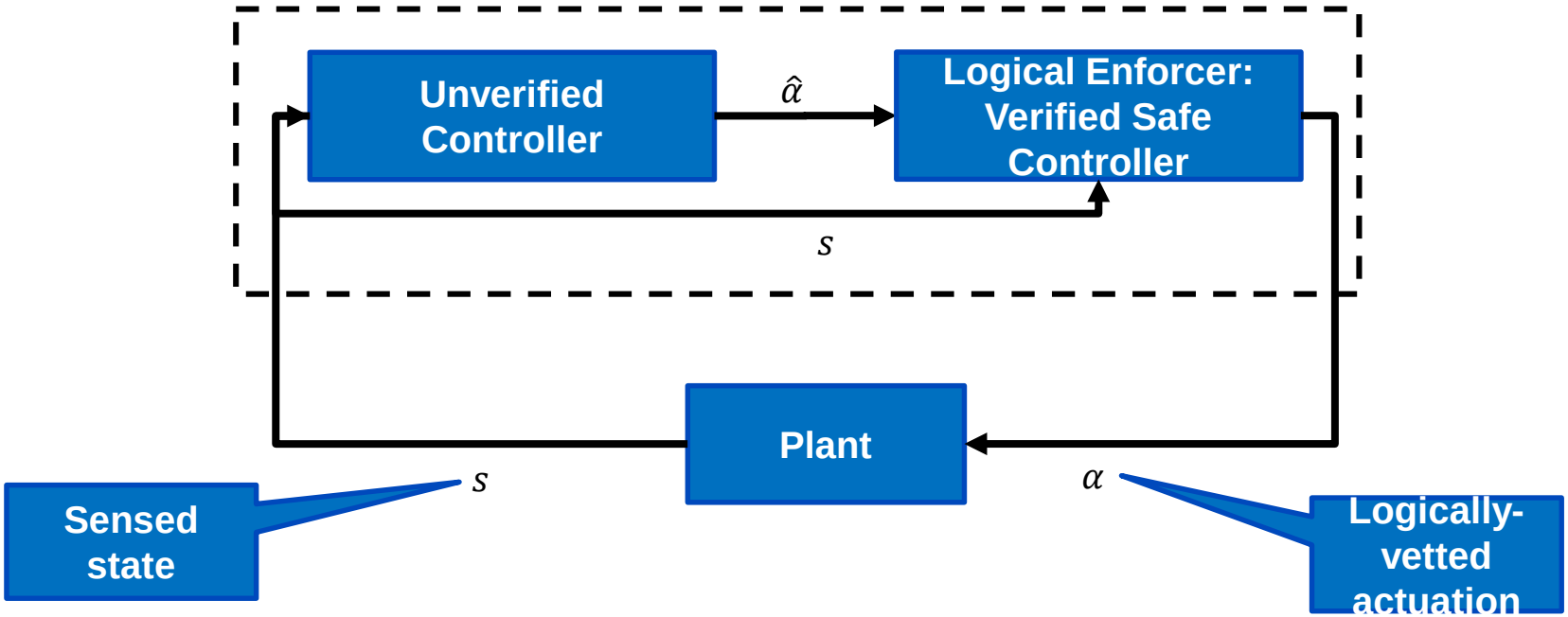
- Receives and processes variable updates from other nodes
- Updates ordered via Lamport clocks

Portable to different OSes (Windows, Linux, Android etc.) and networking technology (TCP/IP, UDP, DDS etc.)

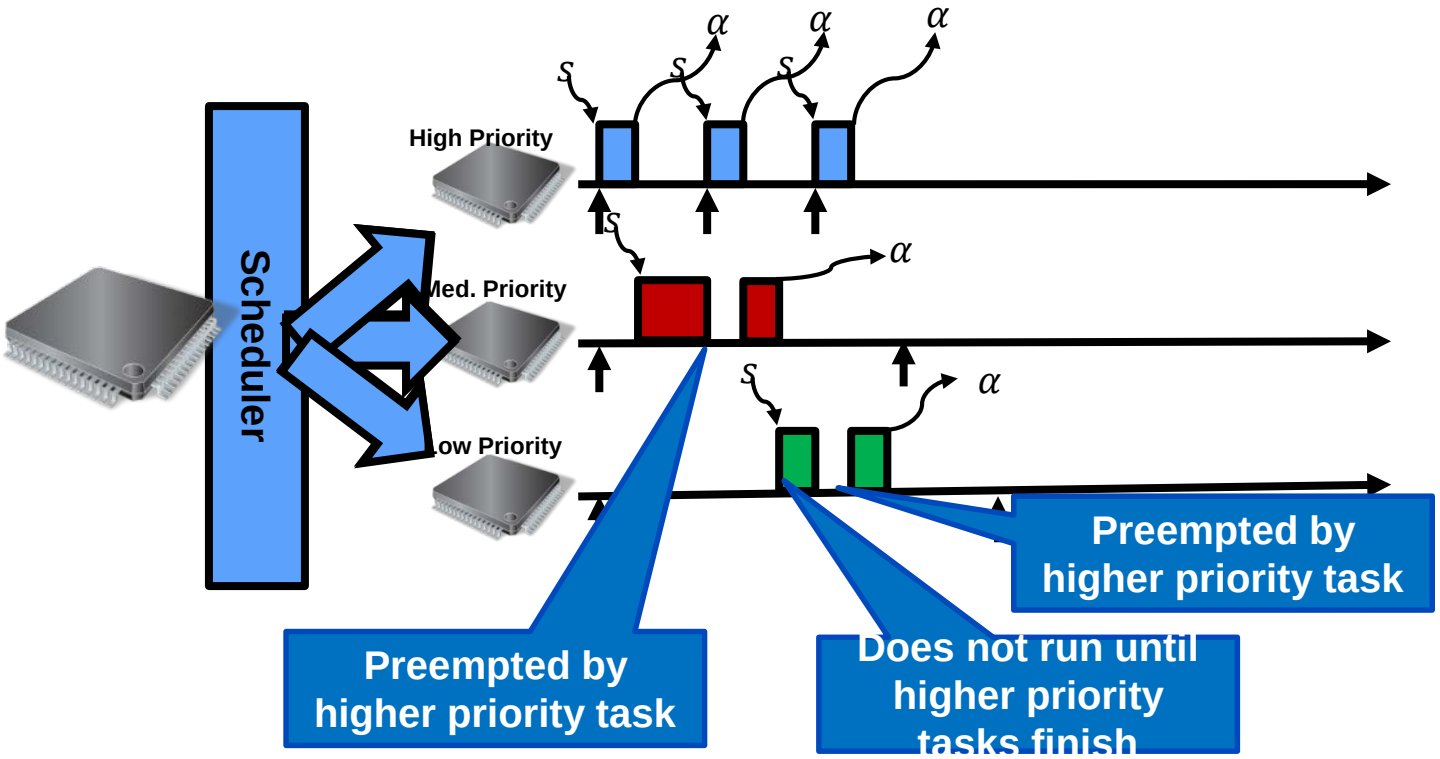
Project 2: Certifiable Distributed Runtime Assurance



Sense Actuation Loop + Logical Enforcer

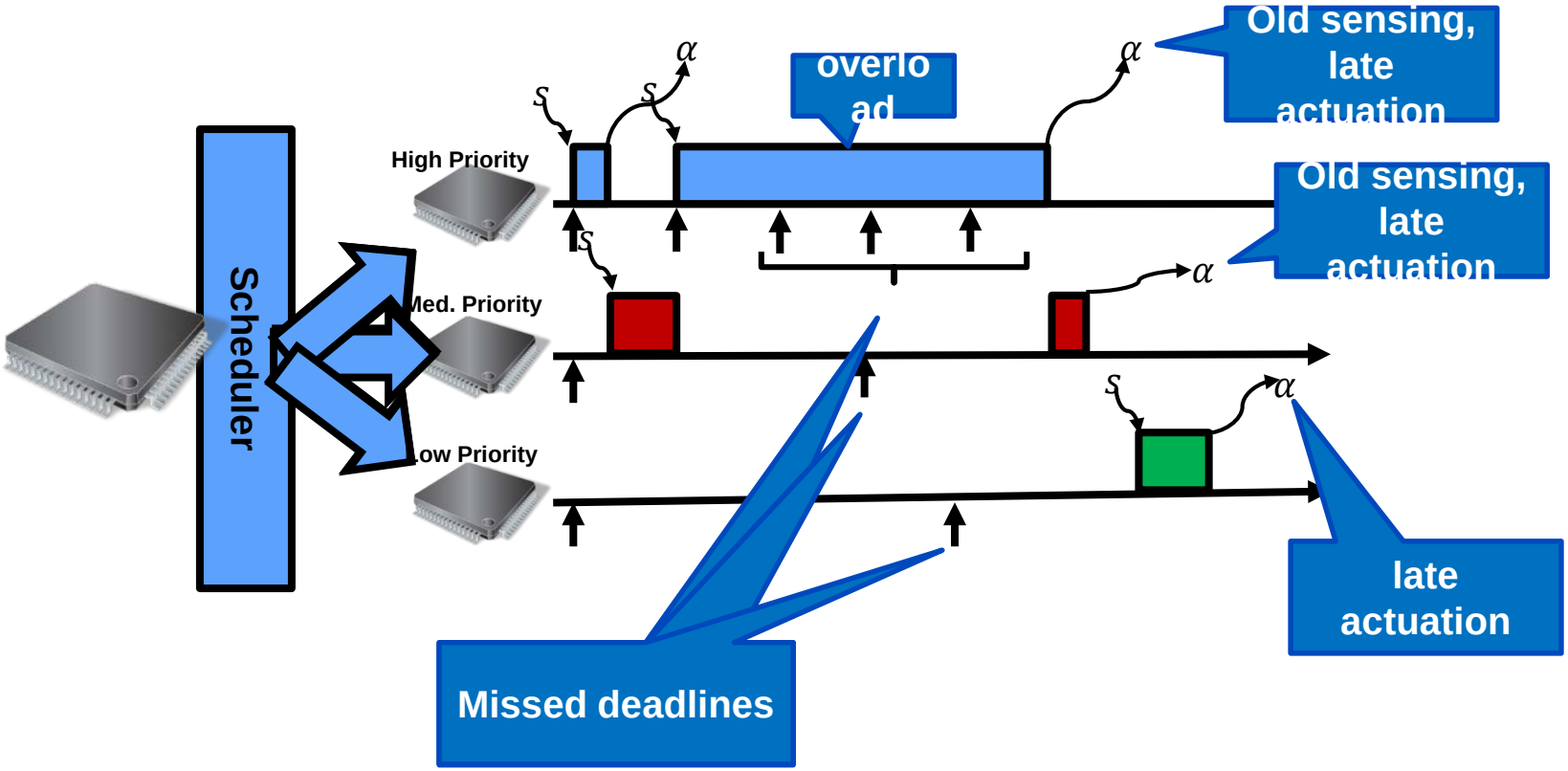


Fixed-Priority Scheduling + Rate Monotonic



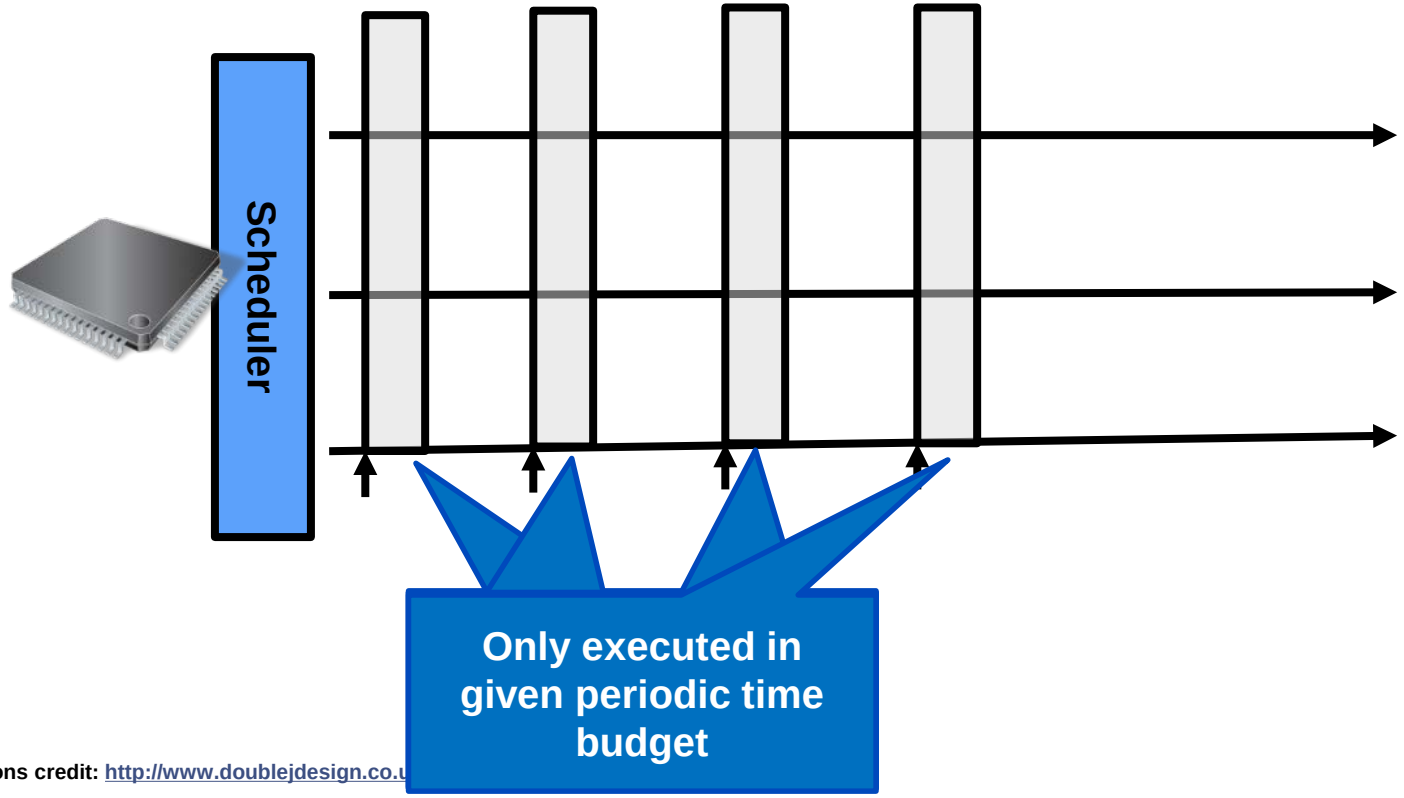
Icons credit: http://www.doublejdesign.co.uk

Overload => old sensed data + late actuation



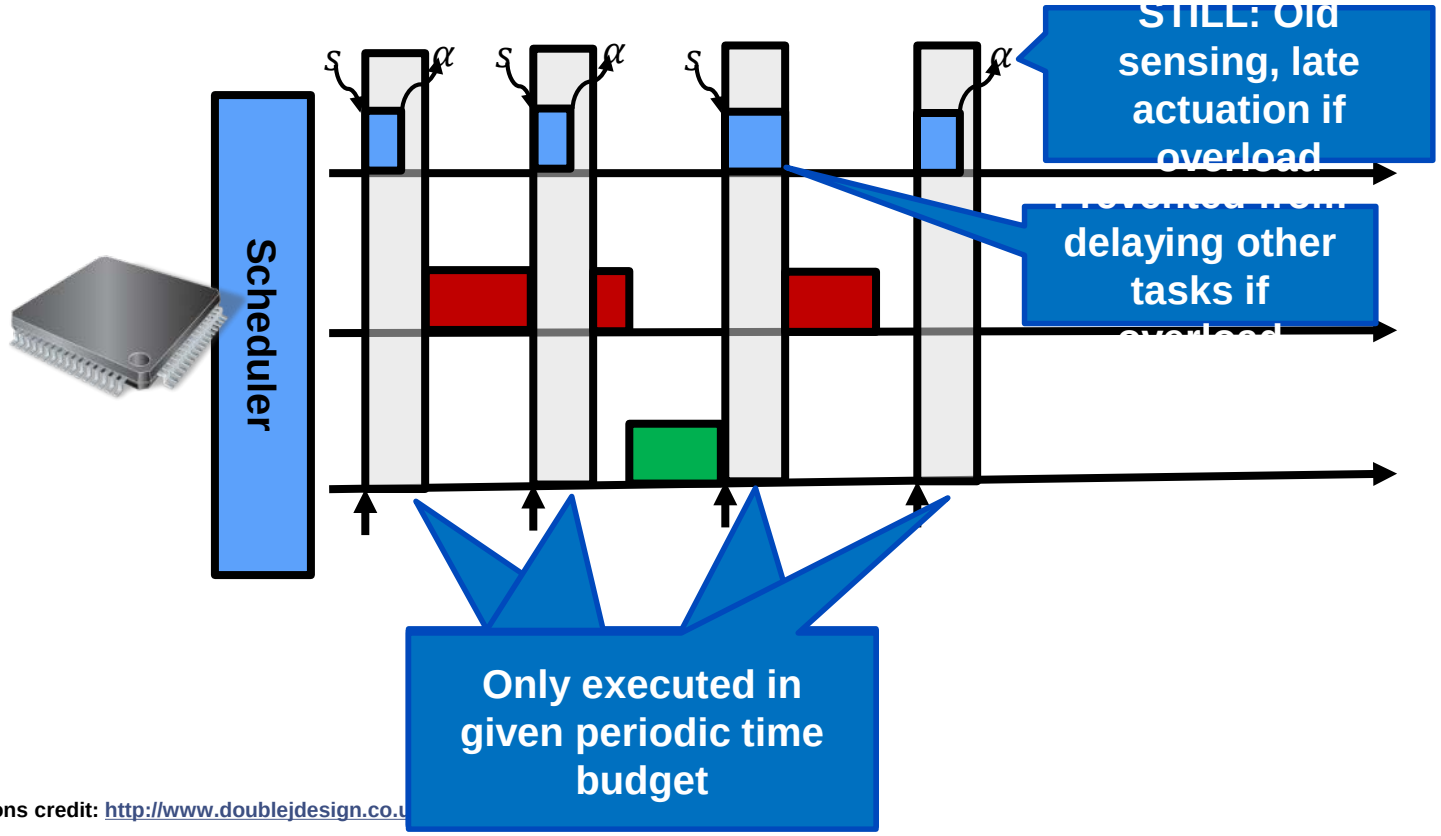
Icons credit: http://www.doublejdesign.co.uk

Solution: Enforce timing budgets (timing enforcement)



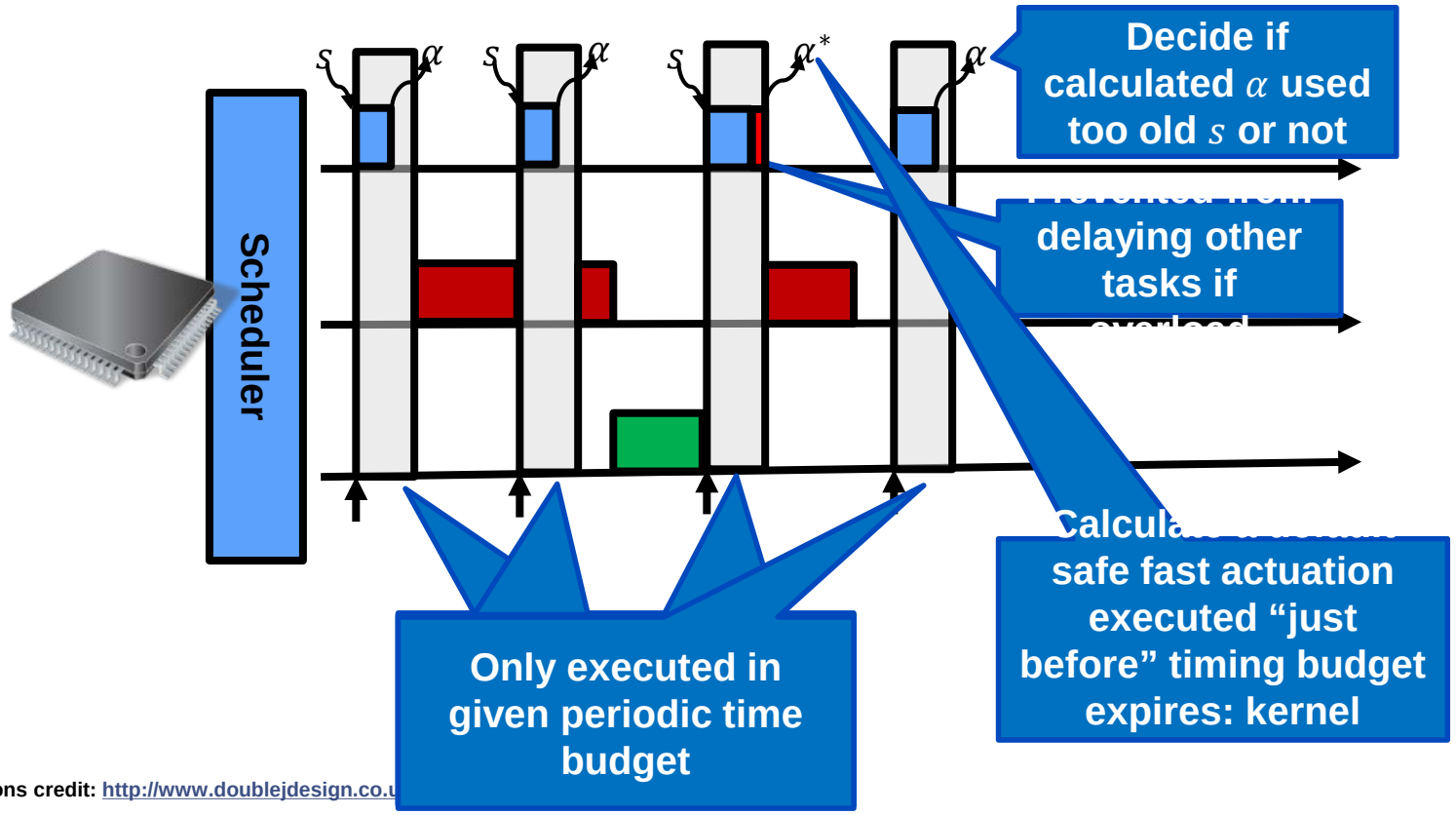
Icons credit: http://www.doublejdesign.co.u

Solution step 1: enforce timing budgets (timing enforcement)



Icons credit: http://www.doublejdesign.co.u

Solution step 2: fast actuation on timing enforcement

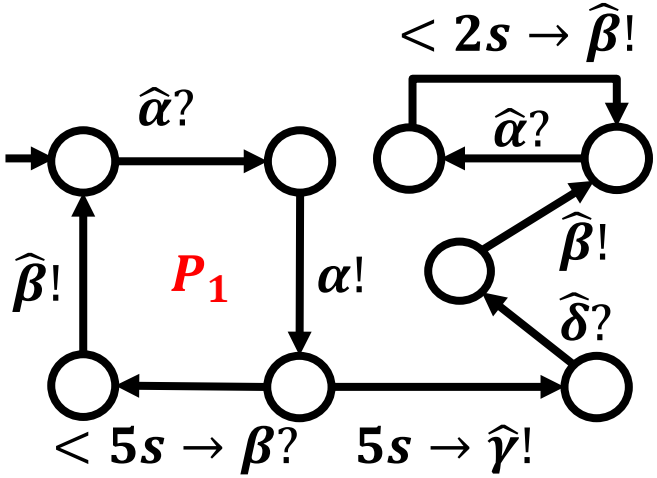


Icons credit: http://www.doublejdesign.co.u

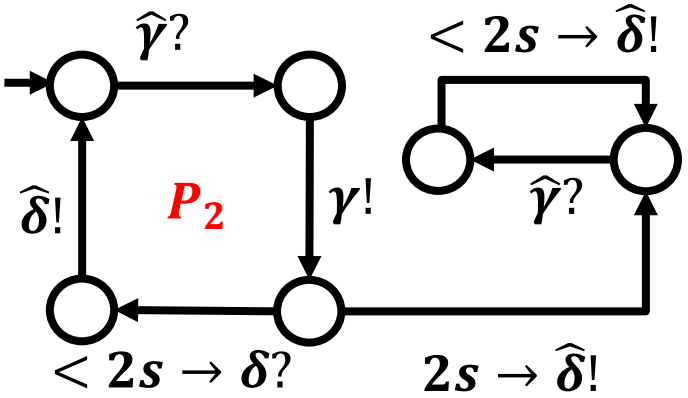
CDRA: Approach (1)

Prove $E_1 \preccurlyeq P_1$ and $E_2 \preccurlyeq P_2$ using software verification

Controller Policy



Logger Policy



Controller Enforcer Implementation (E_1)

Logger Enforcer Implementation (E_2)

Controller Component (C_1) $\{\alpha?, \beta!\}$ $\{\gamma?, \delta!\}$ **Logger Component (C_2)**

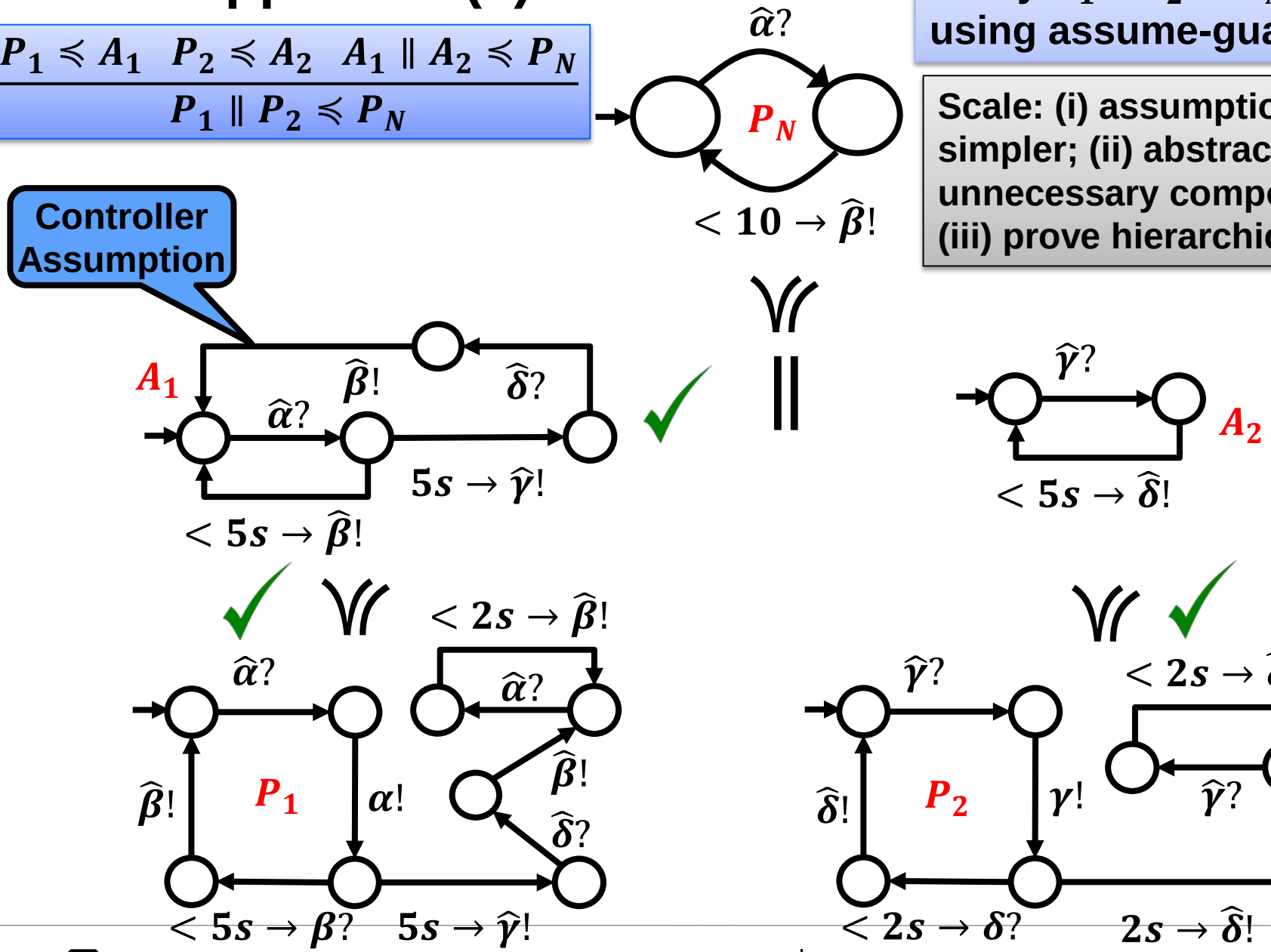
Node

CDRA: Approach (2)

$$\frac{P_1 \preccurlyeq A_1 \quad P_2 \preccurlyeq A_2 \quad A_1 \parallel A_2 \preccurlyeq P_N}{P_1 \parallel P_2 \preccurlyeq P_N}$$

Verify $P_1 \parallel P_2 \preccurlyeq P_N$ using assume-guarantee

Scale: (i) assumptions are simpler; (ii) abstract away unnecessary components; (iii) prove hierarchically.

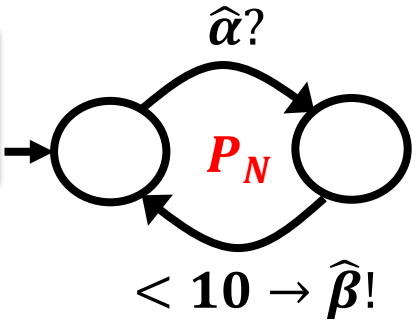


CDRA: Approach (3)

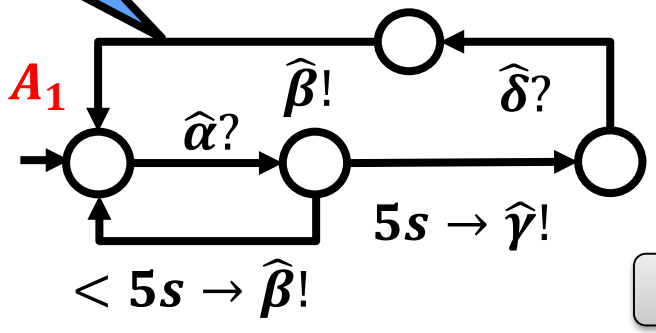
$$\frac{P_1 \preccurlyeq A_1 \quad P_2 \preccurlyeq A_2 \quad A_1 \parallel A_2 \preccurlyeq P_N}{P_1 \parallel P_2 \preccurlyeq P_N}$$

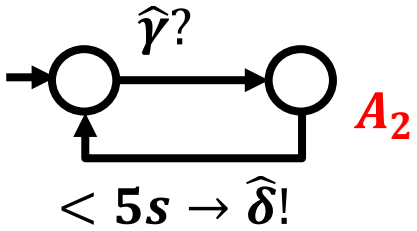
Verify $P_1 \parallel P_2 \preccurlyeq P_N$ using assume-guarantee

Scale: minimal system re-verification needed when a policy or enforcer is modified

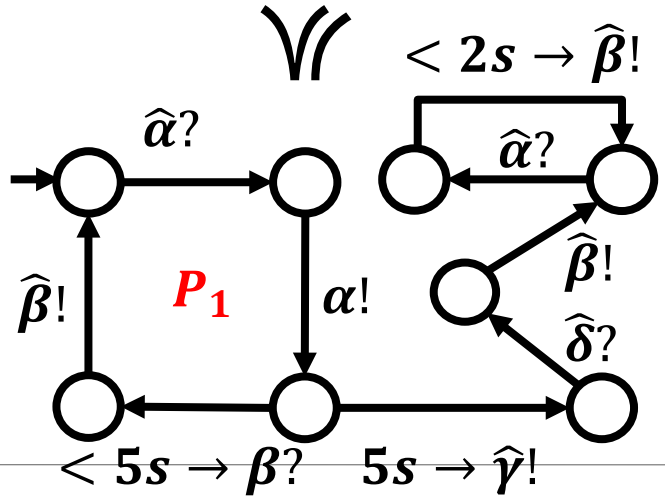


Controller Assumption

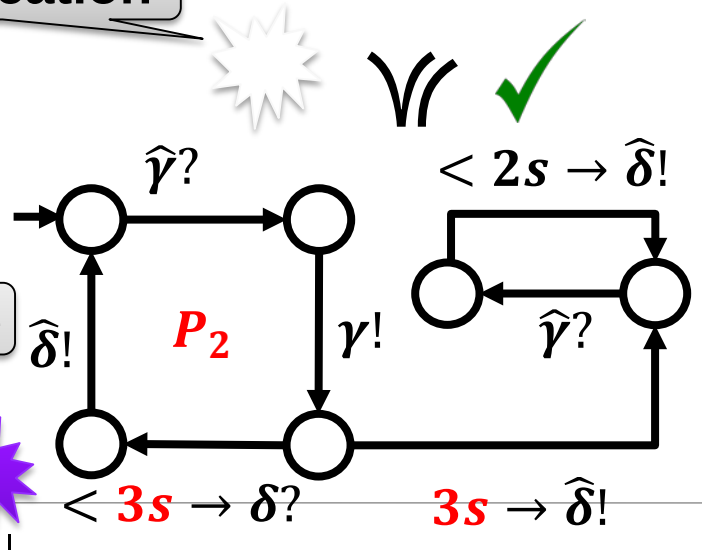




Re-verification



Change

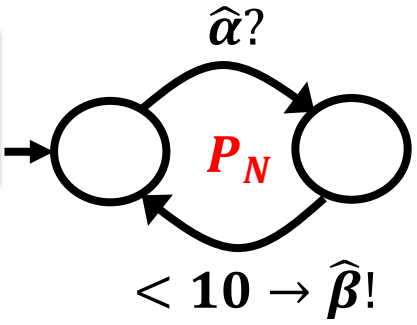


CDRA: Approach (4)

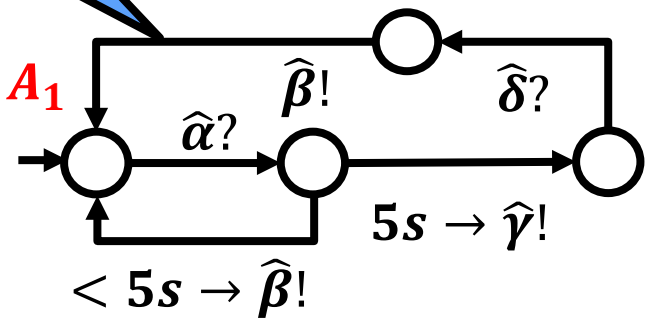
$$\frac{P_1 \preccurlyeq A_1 \quad P_2 \preccurlyeq A_2 \quad A_1 \parallel A_2 \preccurlyeq P_N}{P_1 \parallel P_2 \preccurlyeq P_N}$$

Verify $P_1 \parallel P_2 \preccurlyeq P_N$ using assume-guarantee

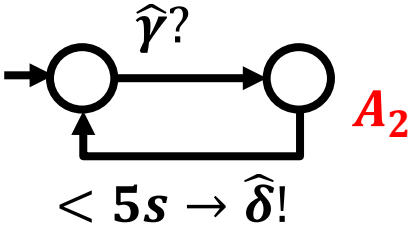
(i) Other (circular) rules exist; (ii) Challenges – (a) proving rule soundness; (b) finding right assumption.



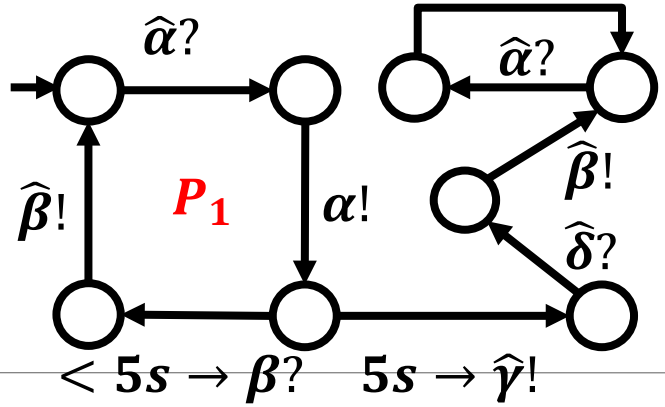
Controller Assumption



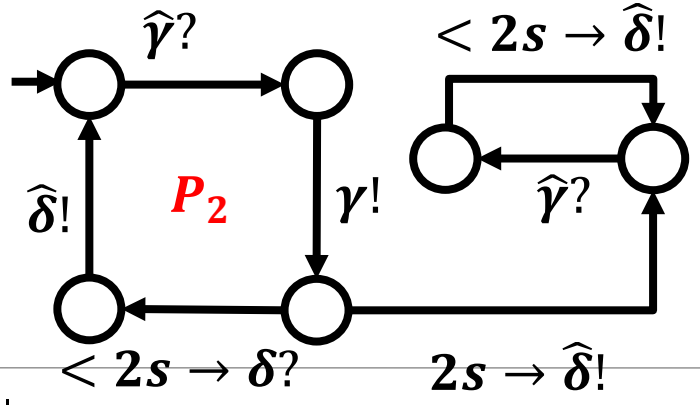
$\parallel$



$\vee$ $< 2s \rightarrow \hat{\beta}!$



$\vee$ $< 2s \rightarrow \hat{\delta}!$



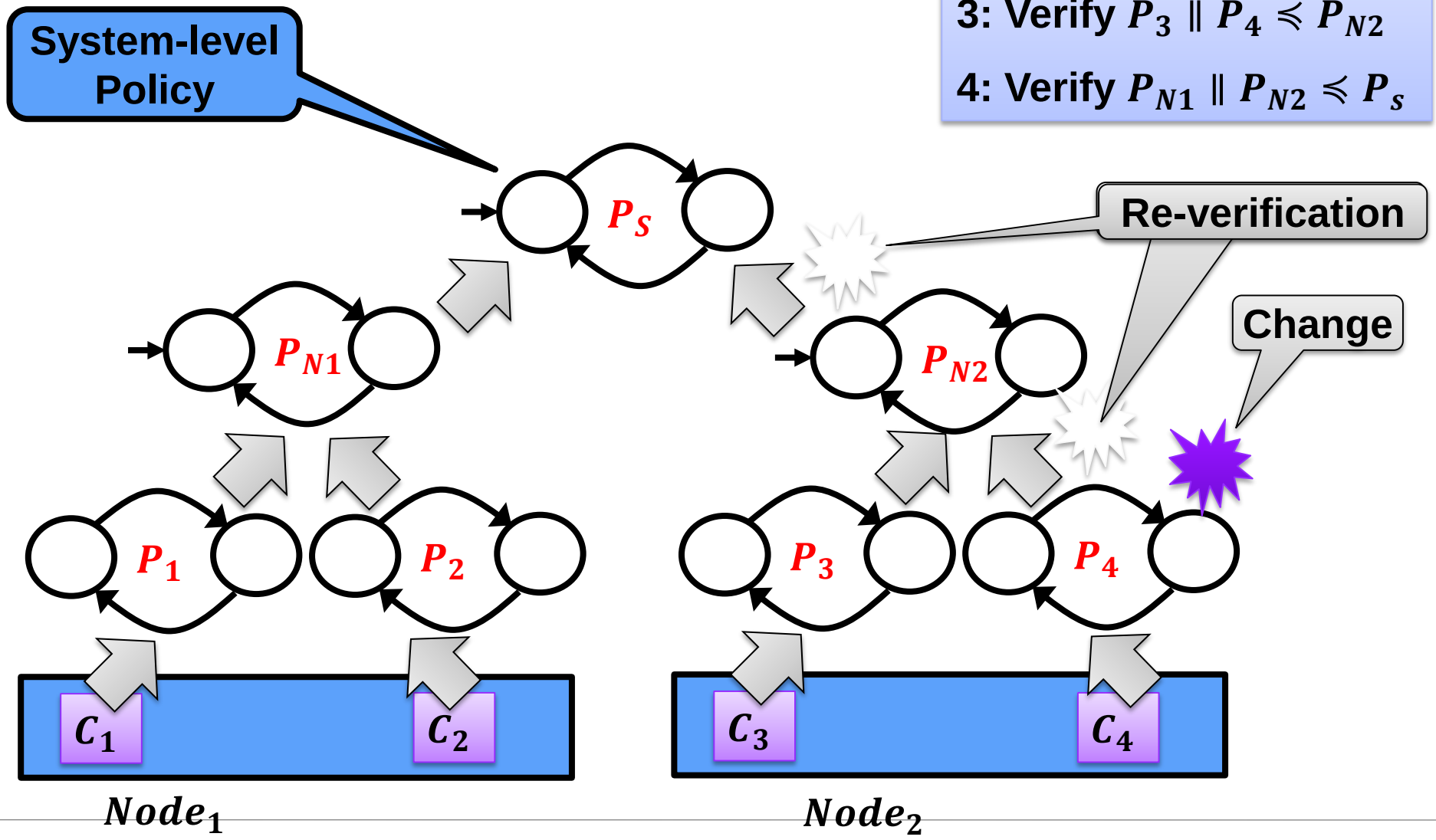
CDRA: Approach (5)

1: Verify $E_i \preccurlyeq P_i$

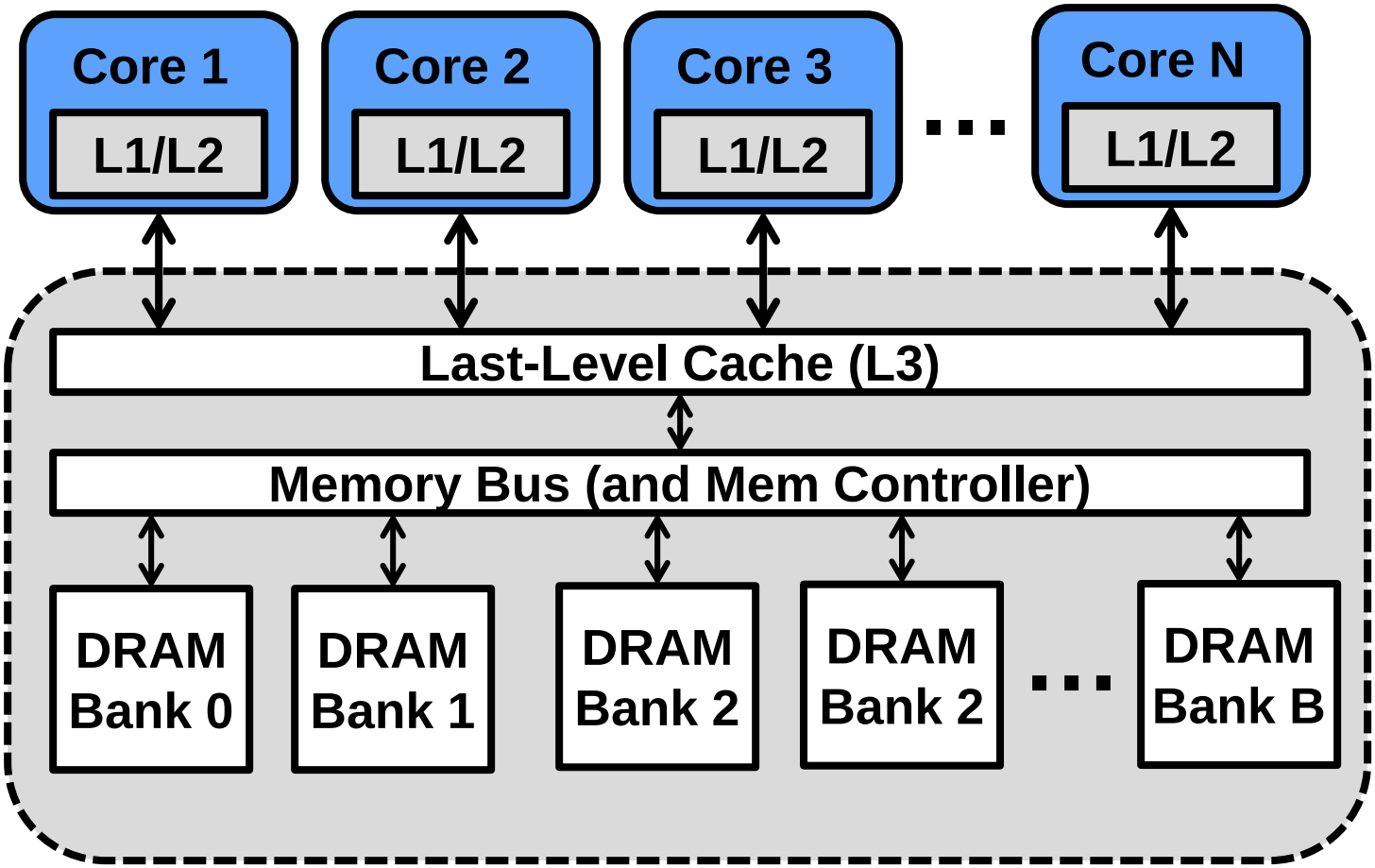
2: Verify $P_1 \parallel P_2 \preccurlyeq P_{N1}$

3: Verify $P_3 \parallel P_4 \preccurlyeq P_{N2}$

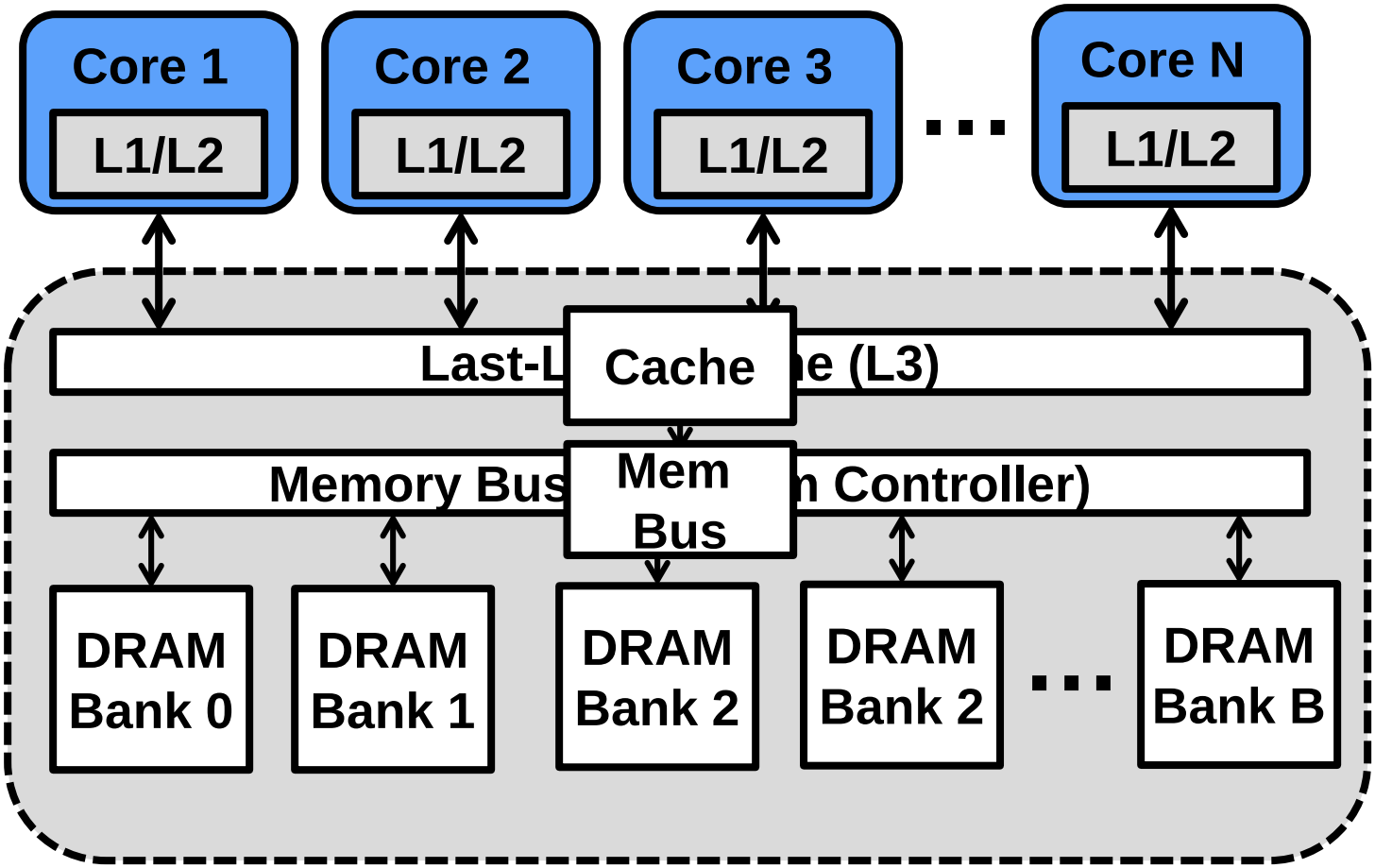
4: Verify $P_{N1} \parallel P_{N2} \preccurlyeq P_s$



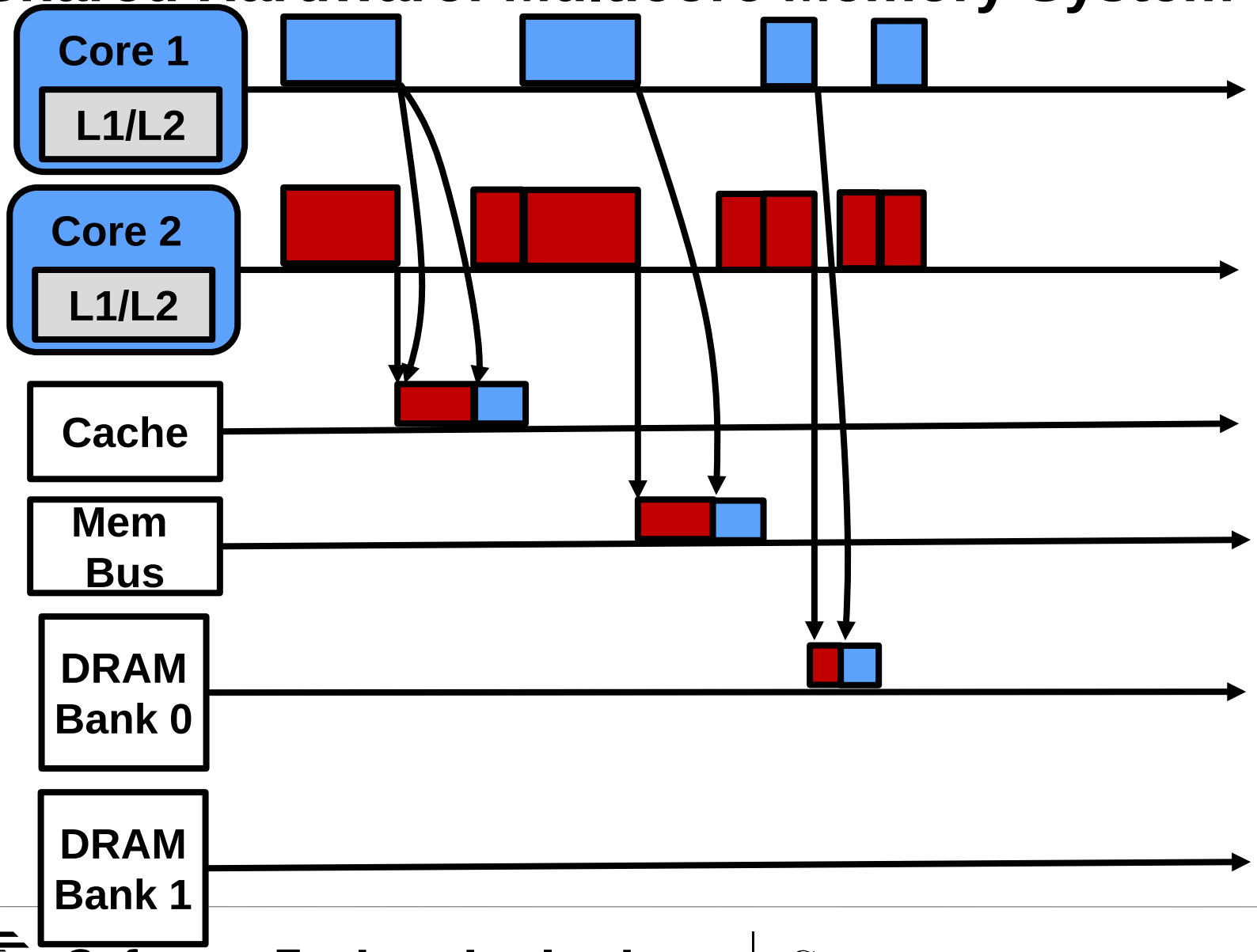
Project 3: Real-Time Scheduling for Multicore



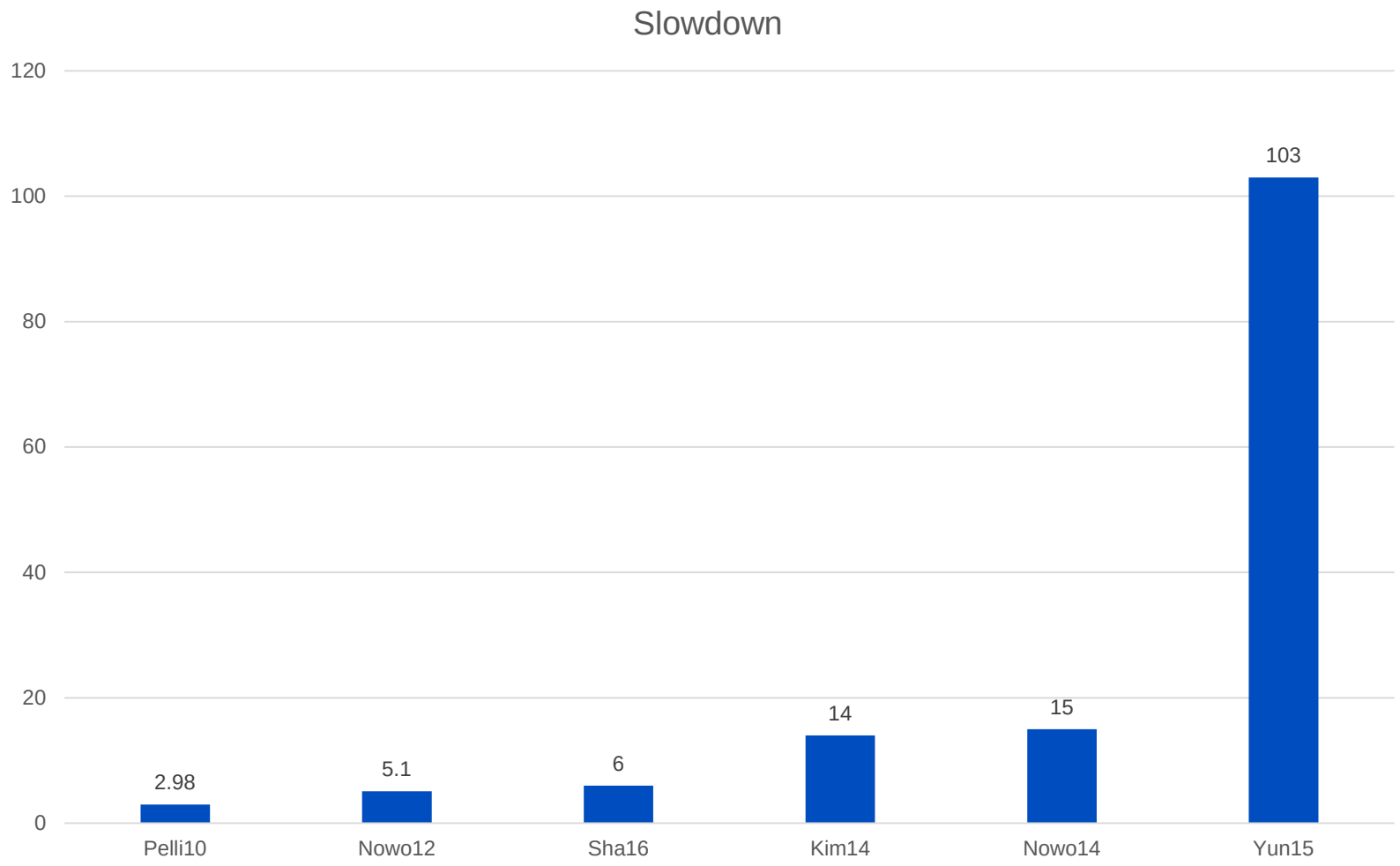
Shared Hardware: Multicore Memory System



Shared Hardware: Multicore Memory System



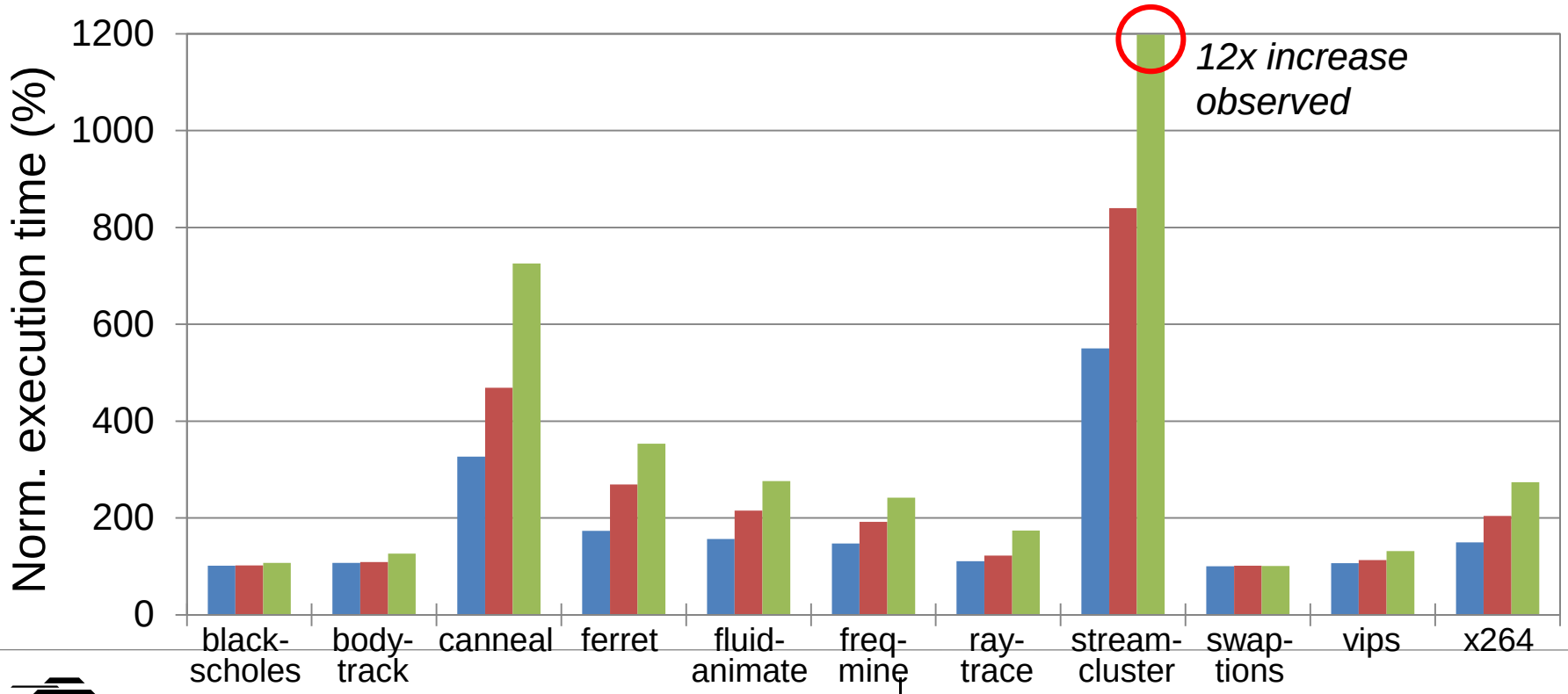
How Bad?



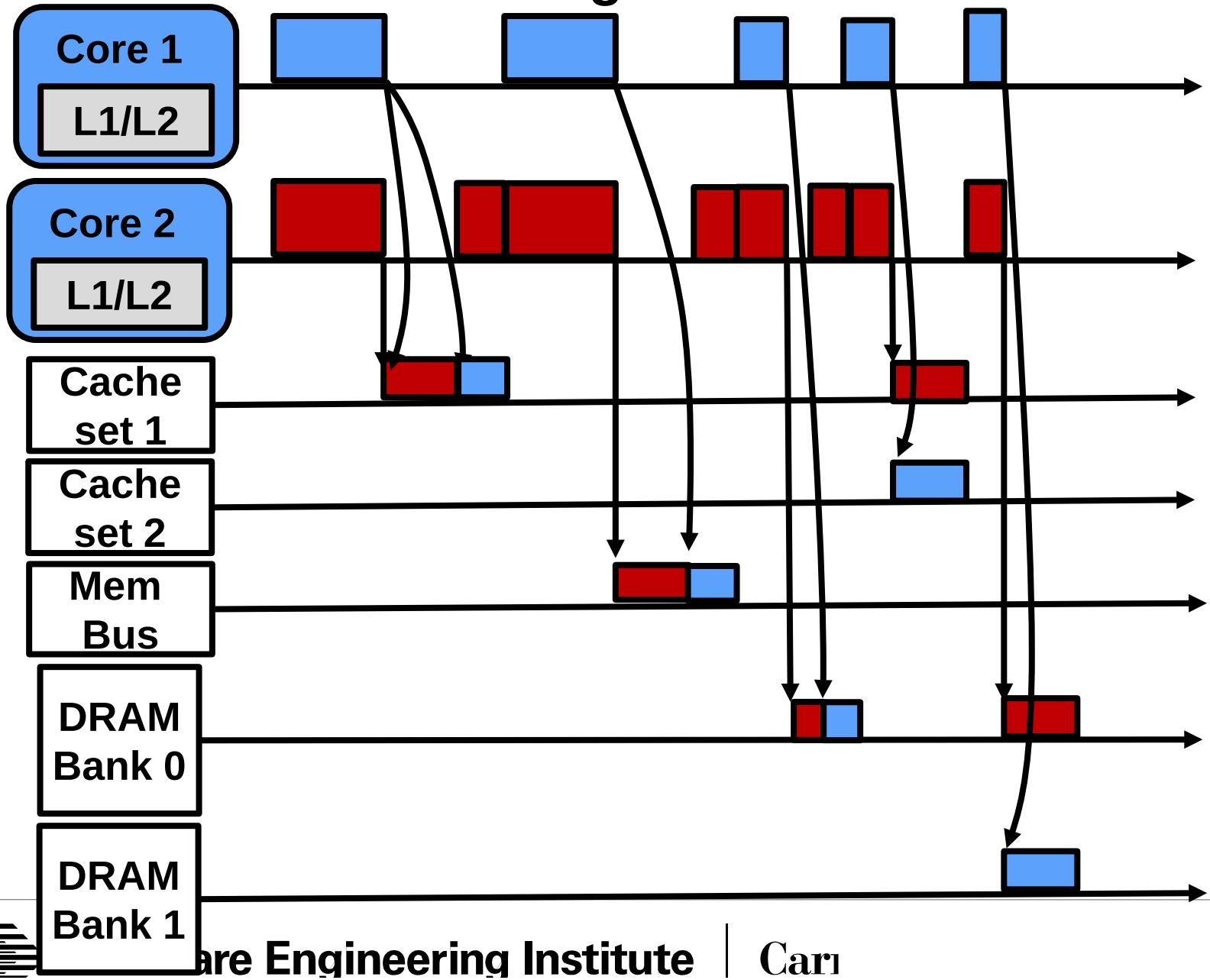
Different for Applications (PARSEC Benchmark)

- 1 attacker → Max 5.5x increase
- 2 attackers → Max 8.4x increase
- 3 attackers → Max 12x increase

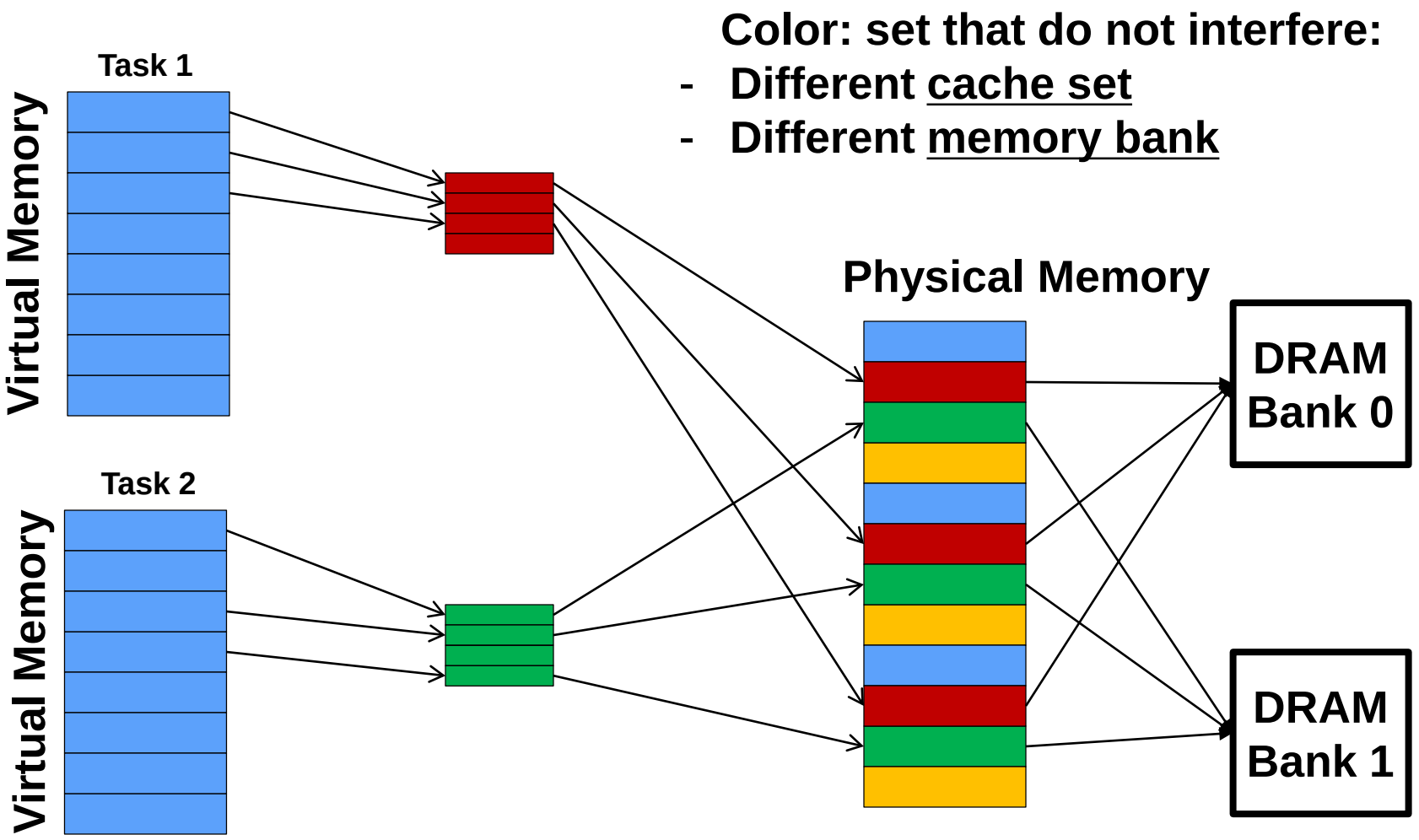
We should predict, bound and reduce the memory interference delay!



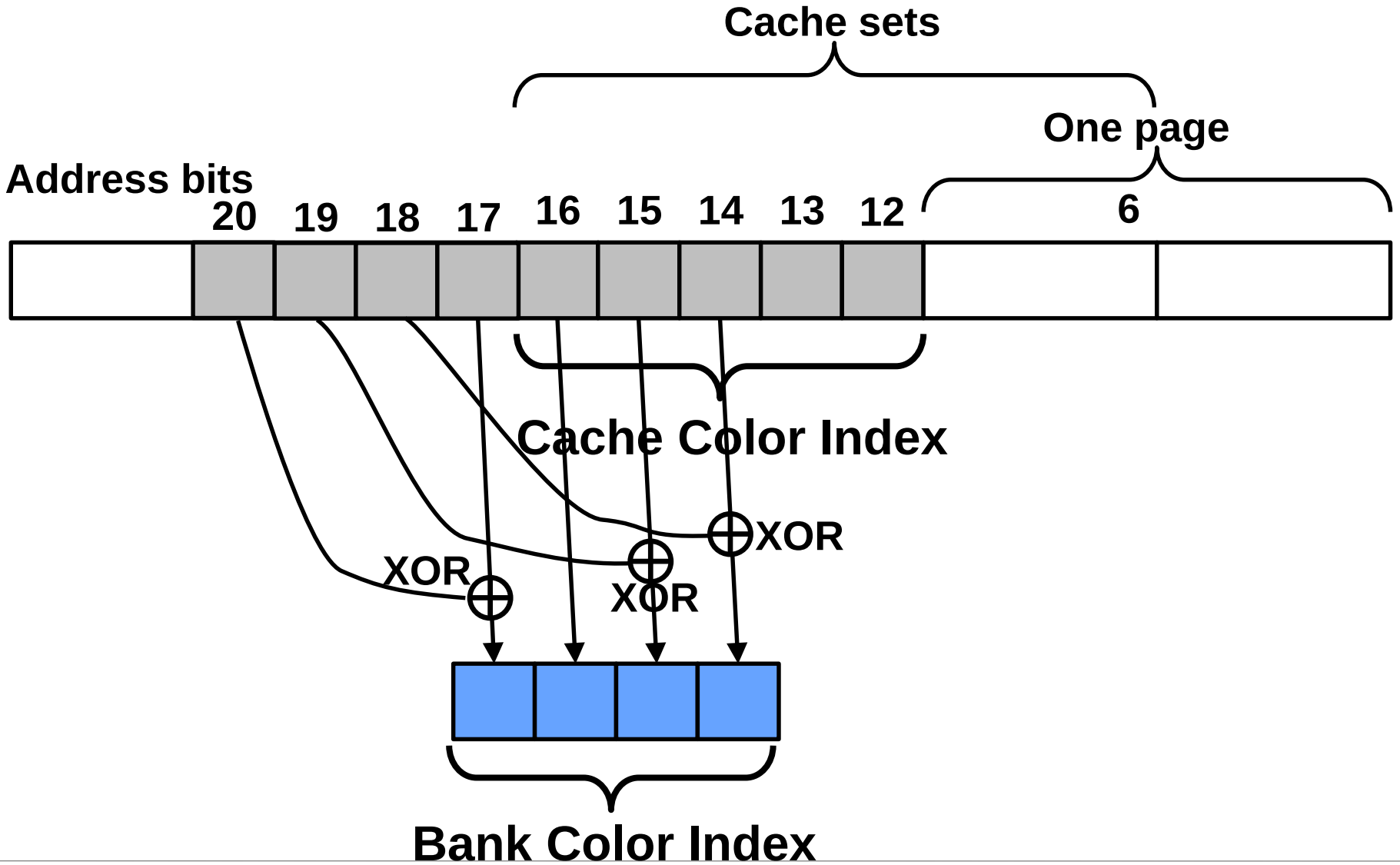
Solution 1: Partitioning



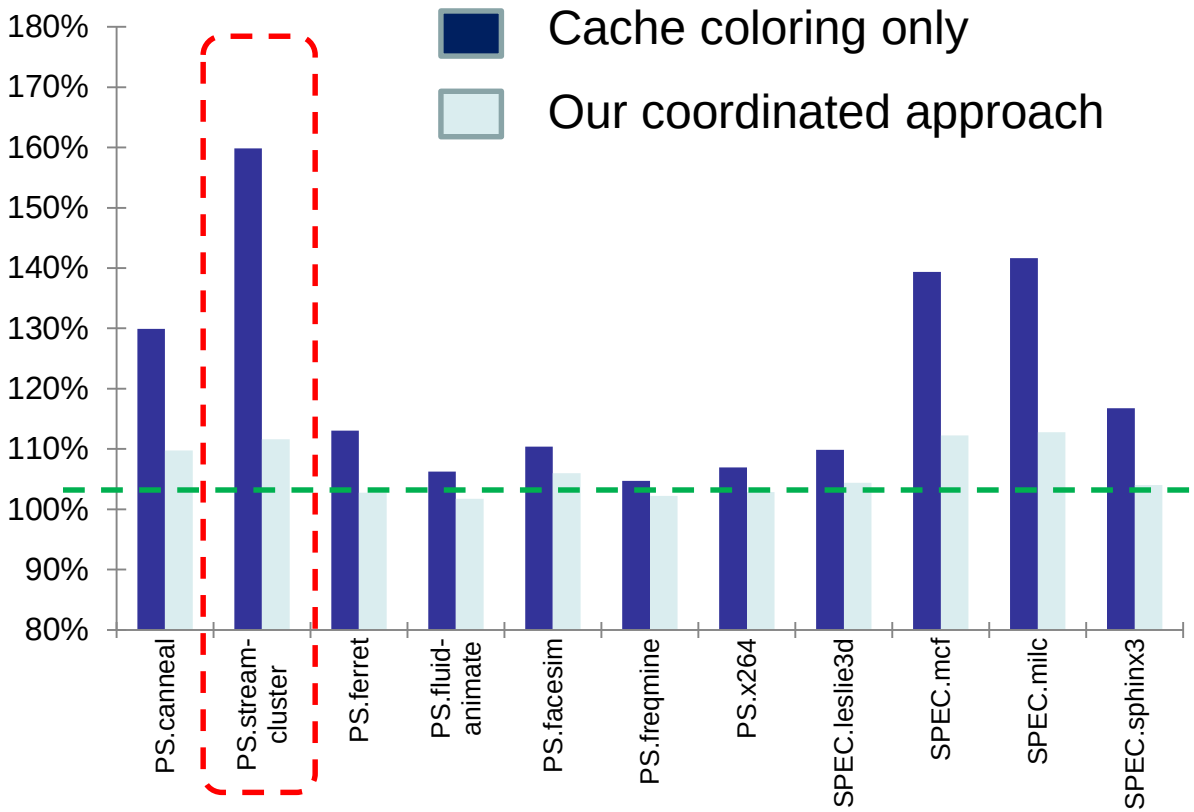
Solutions 1: Virtual Memory “Coloring”



Solution 1: Challenge – Conflicting Partitions



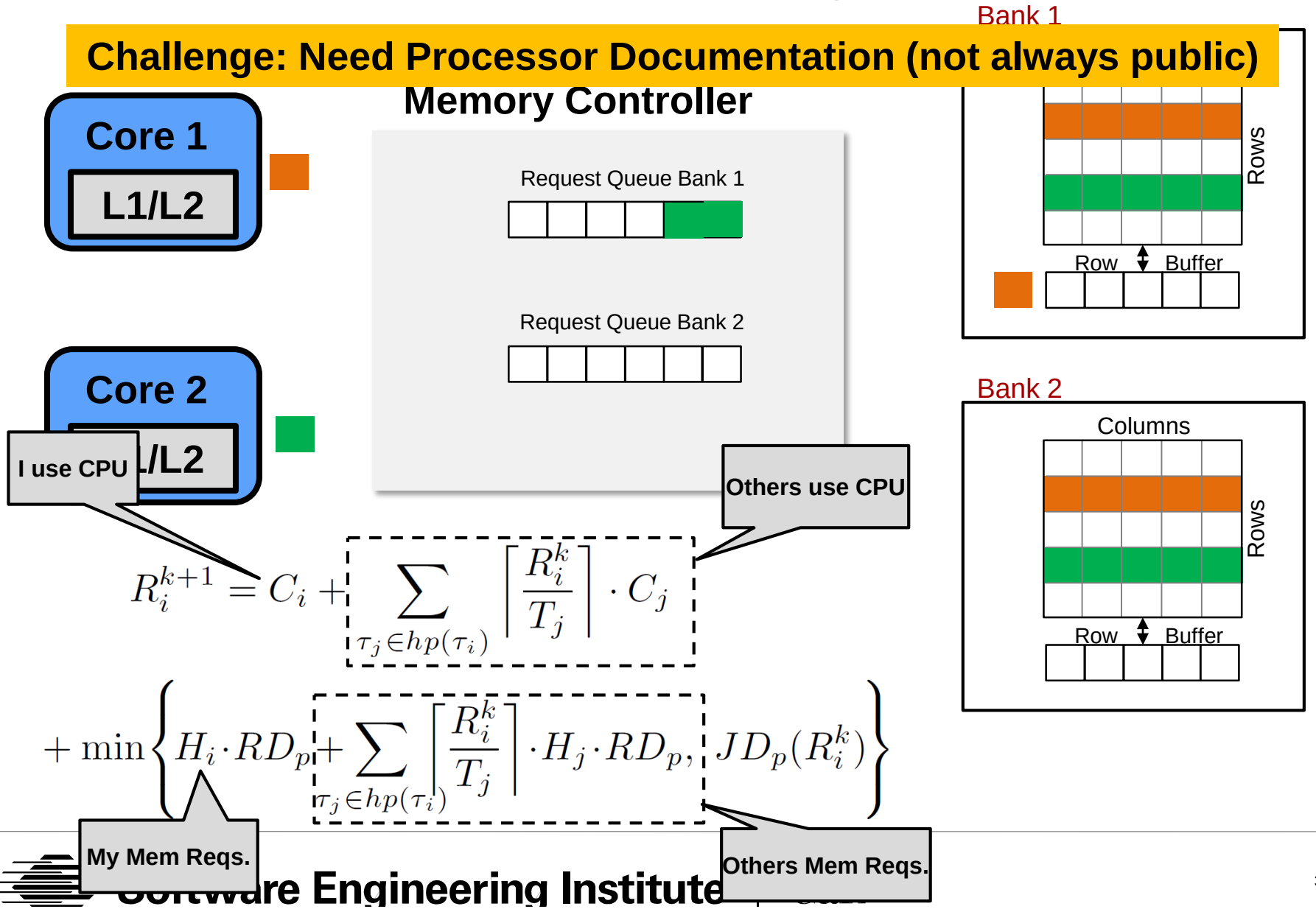
Solution 2: Coordinated Approaches



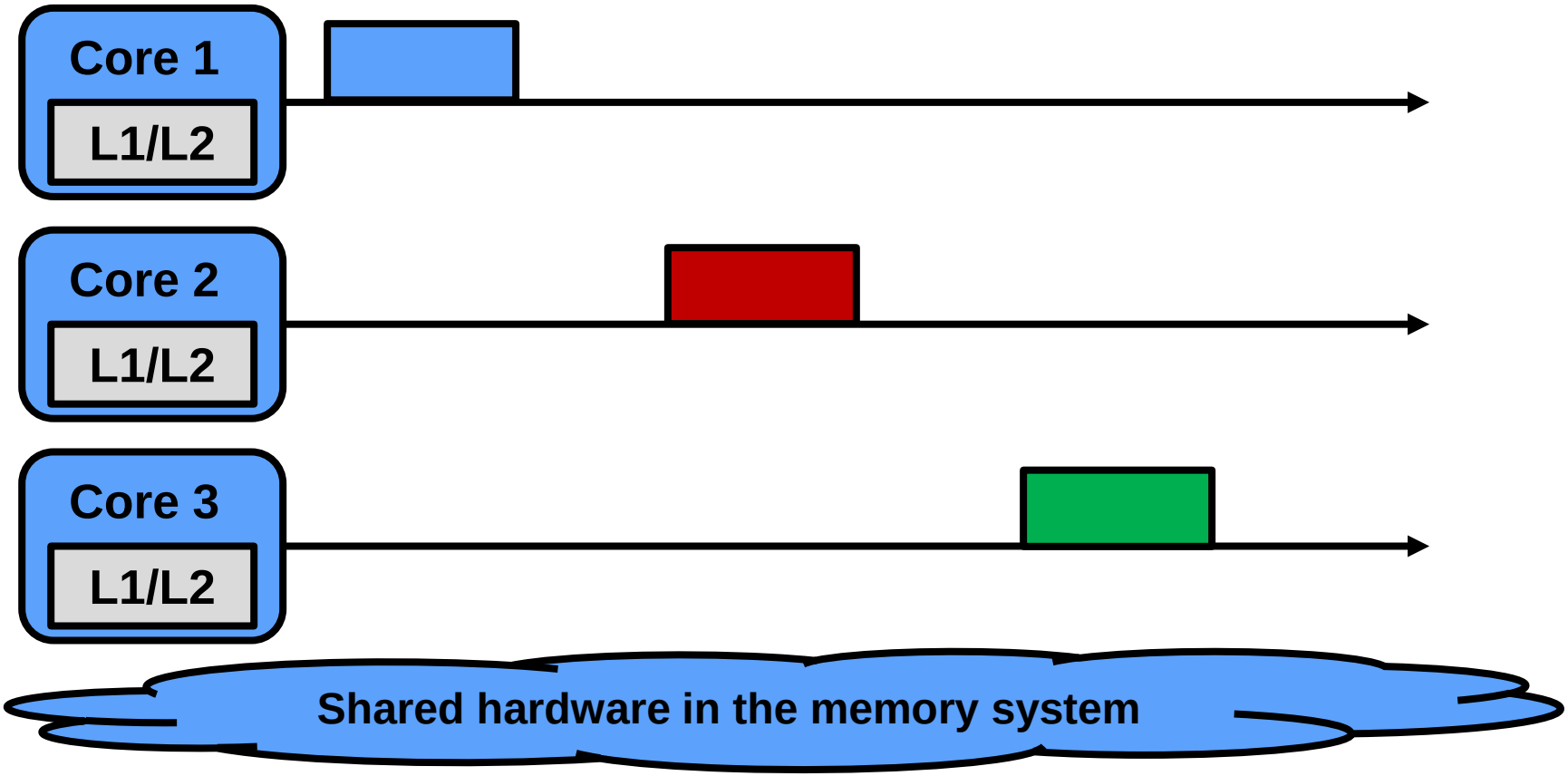
Challenge: Small Number of Partitions

Solution 3: Predictable Sharing of Partitions

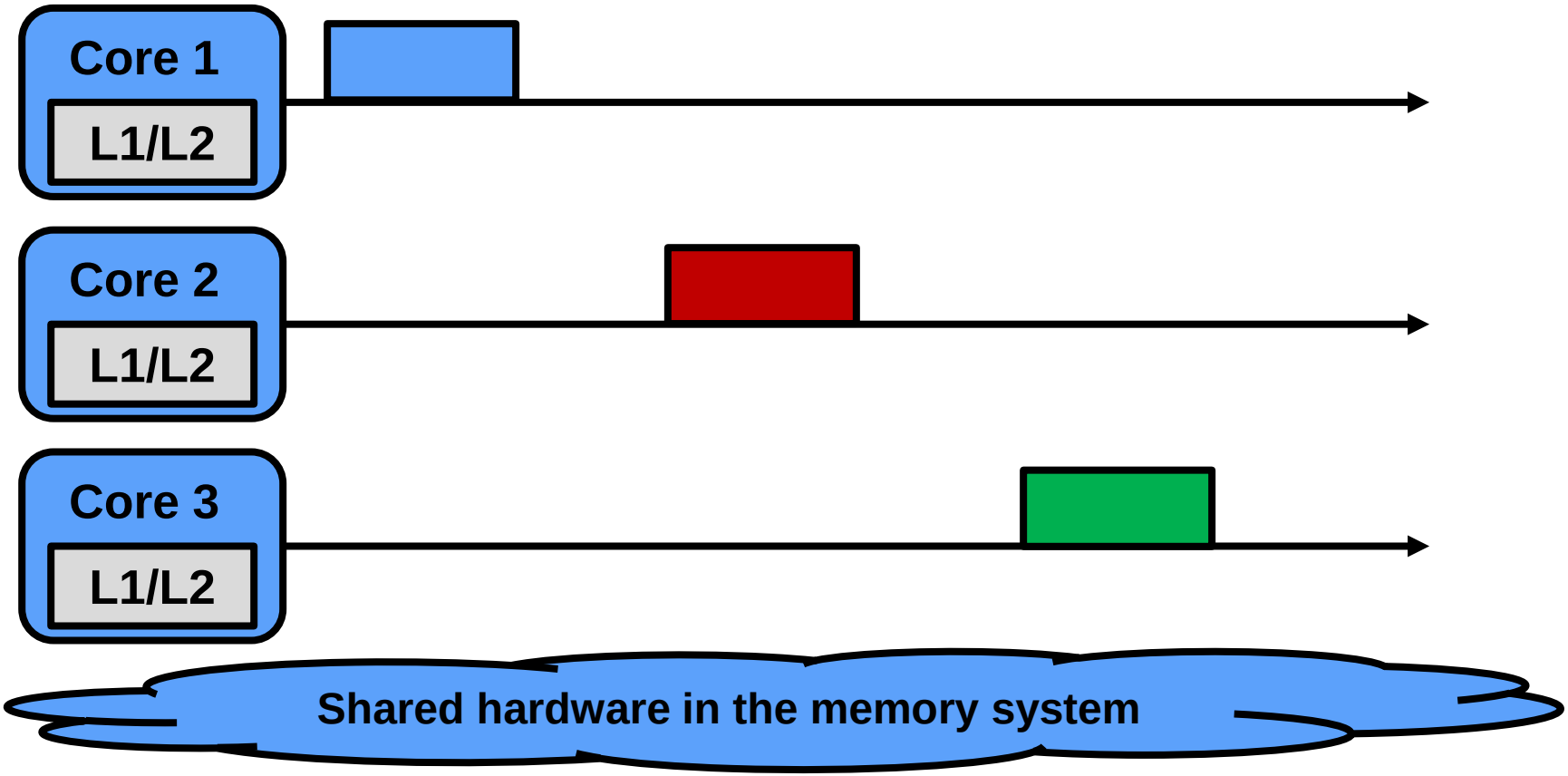
Challenge: Need Processor Documentation (not always public)



Solution 4: Black Box Analysis

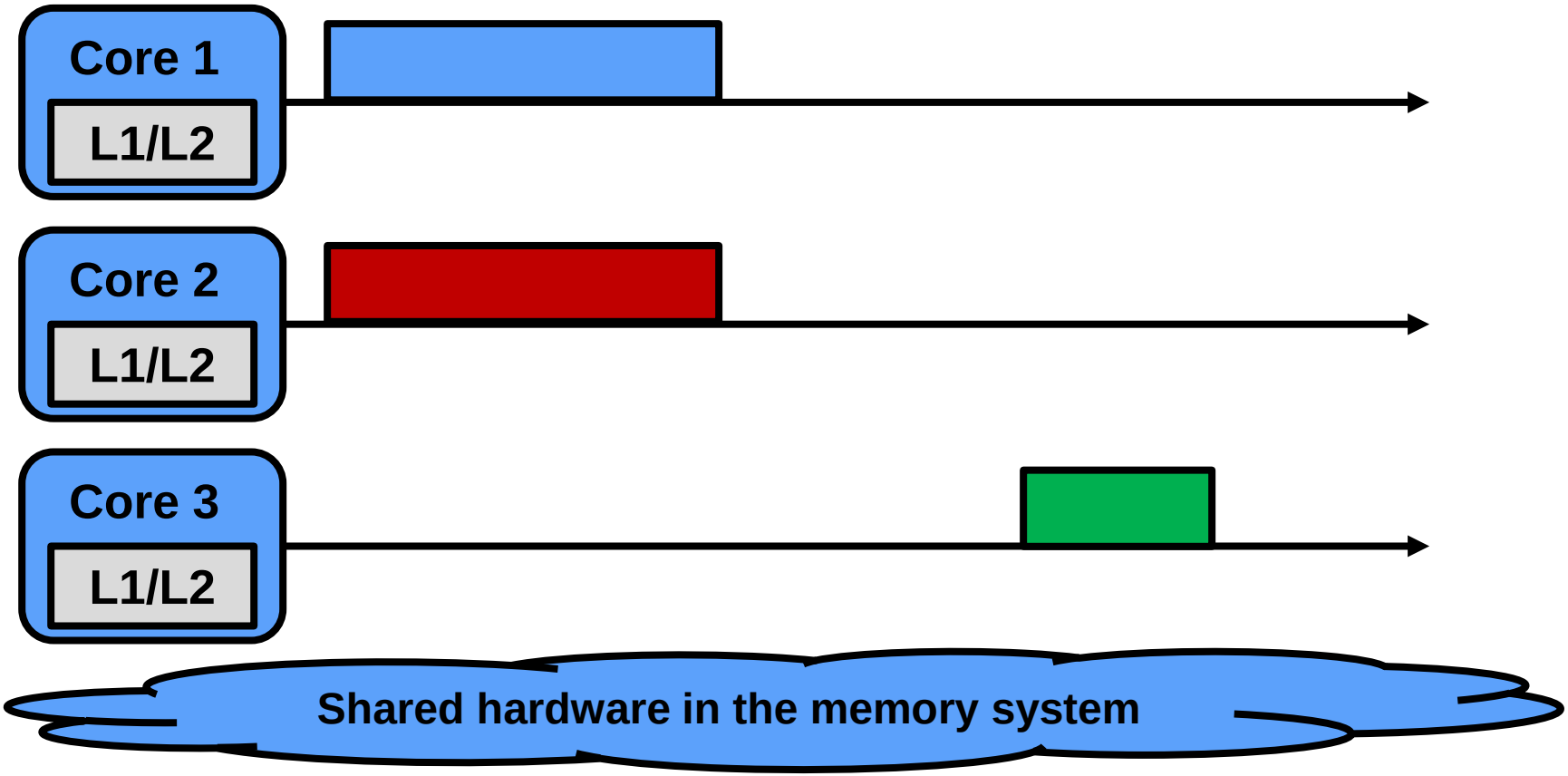


Solution 4: Black Box Analysis



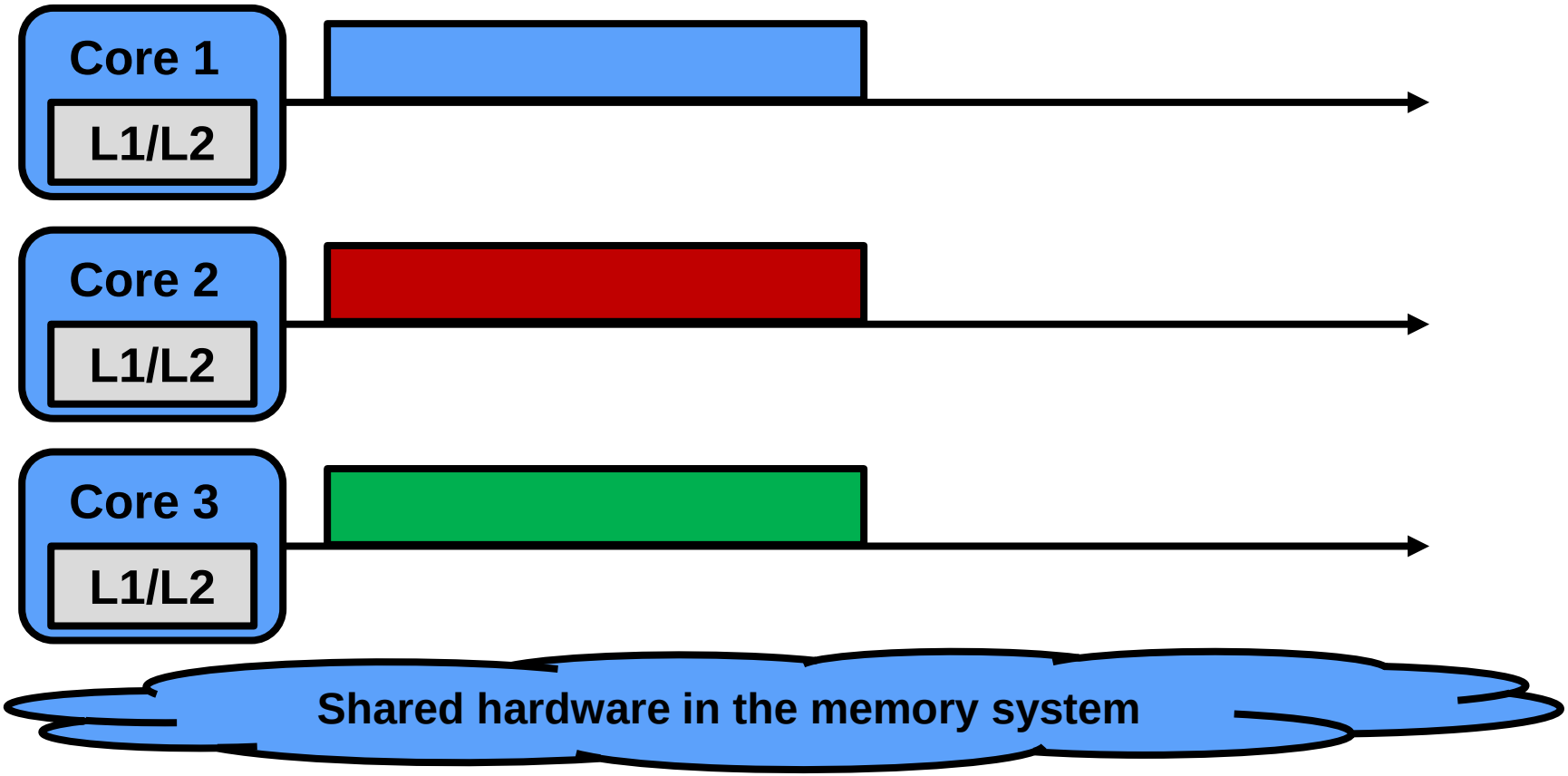
The blue, red, and green tasks execute at different times ⇒ no slowdown

Solution 4: Black Box Analysis



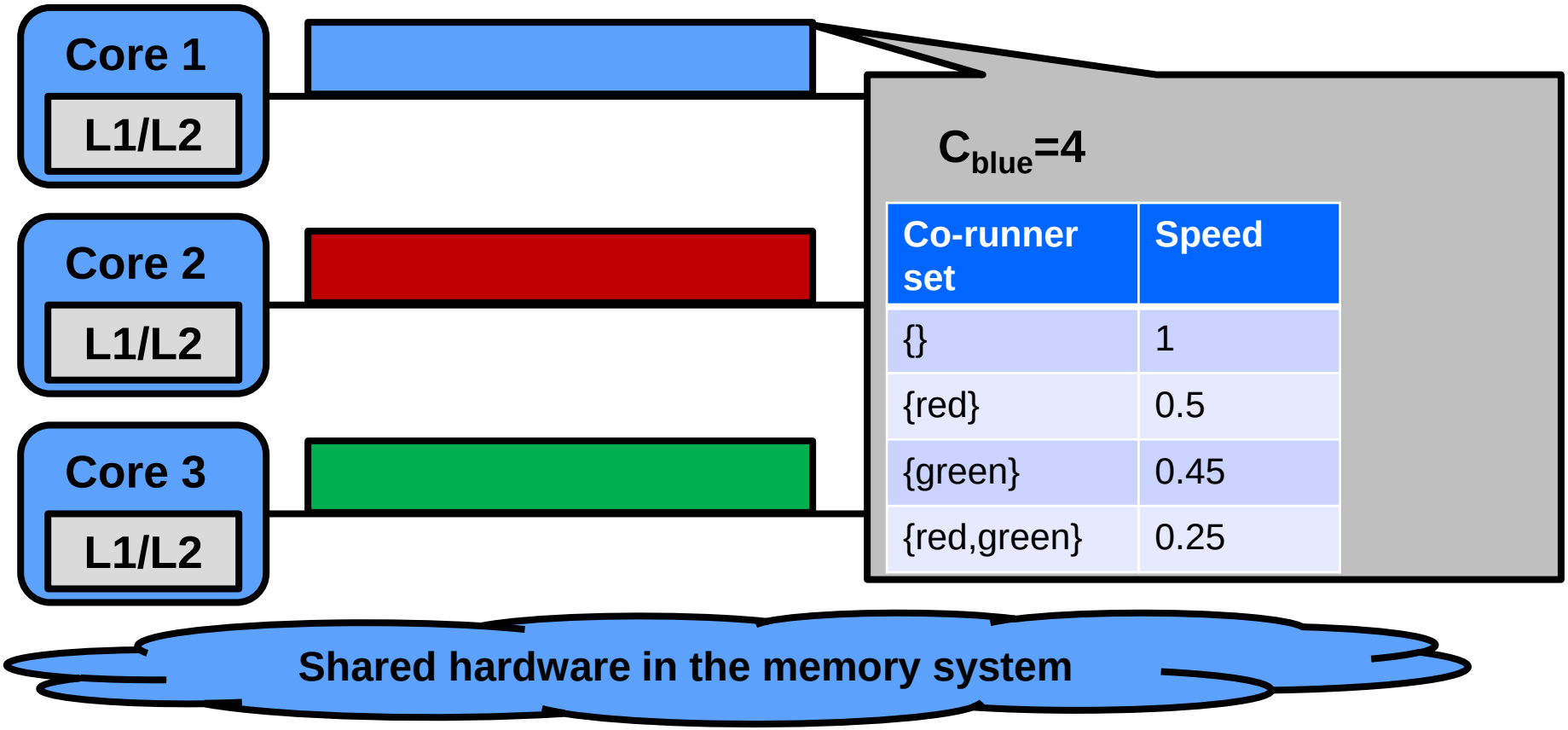
The blue and red tasks execute at the same time ⇒ slowdown ⇒ increased execution time of blue and red.

Solution 4: Black Box Analysis



The blue, red, and green tasks execute at the same time ⇒ slowdown ⇒ increased execution time of all tasks.

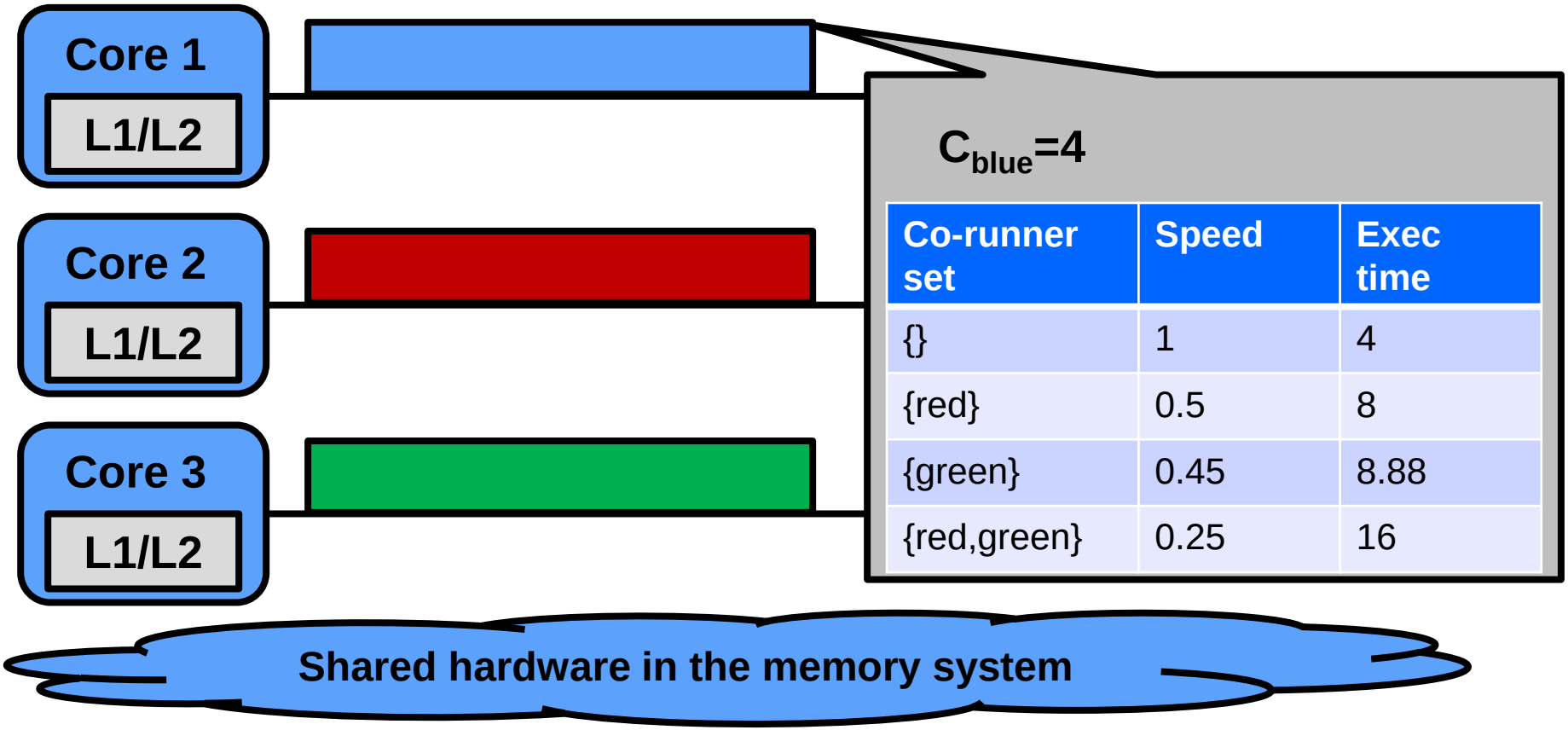
Solution 4: Black Box Analysis



Co-runner set	Speed
{}	1
{red}	0.5
{green}	0.45
{red,green}	0.25

The blue, red, and green tasks execute at the same time ⇒ slowdown ⇒ increased execution time of all tasks.

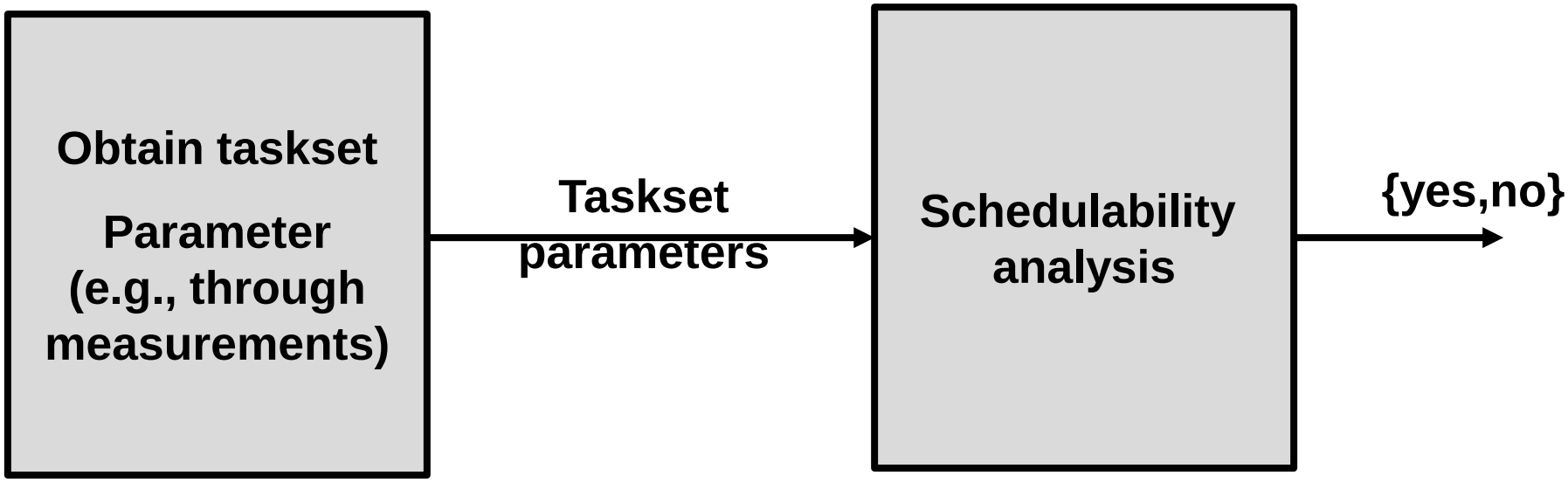
Solution 4: Black Box Analysis



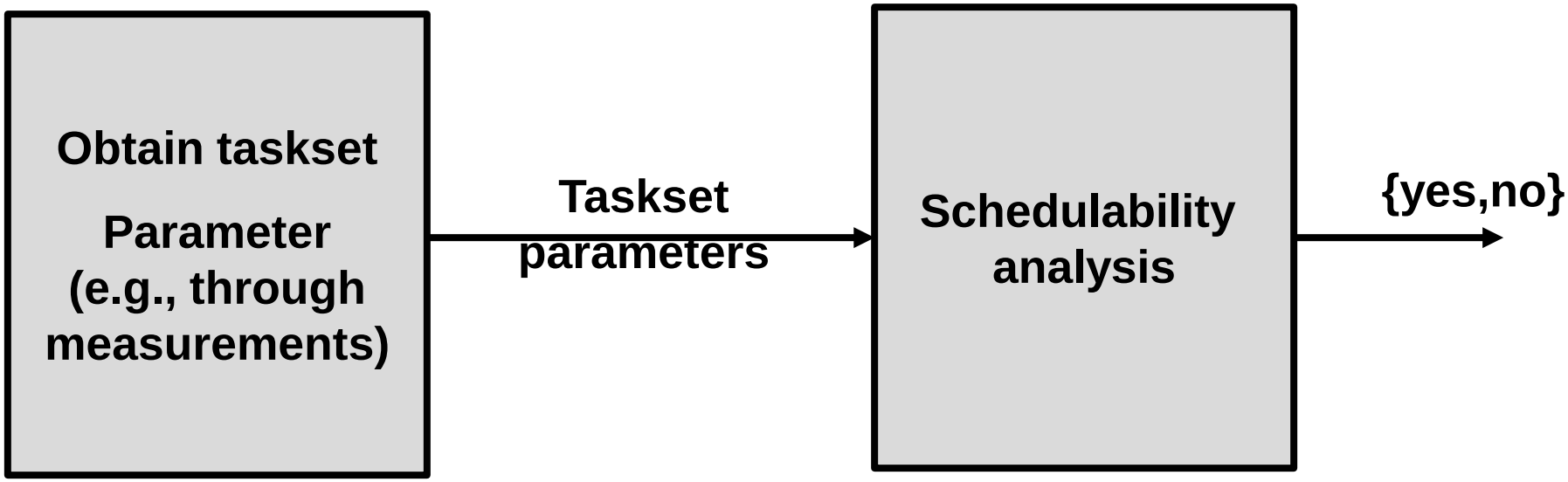
Co-runner set	Speed	Exec time
{}	1	4
{red}	0.5	8
{green}	0.45	8.88
{red,green}	0.25	16

The blue, red, and green tasks execute at the same time ⇒ slowdown ⇒ increased execution time of all tasks.

Solution 4: Black Box Analysis



Solution 4: Black Box Analysis



Able to offer real-time guarantee even for h/w that is not documented (assuming that task parameters are OK)

Summary

CPS Verification Involves Multiple Domains

- Logic
- Timing

Addressing Scalability

- Restrict Behavior
  - Domain Specific Language + Restricted Communication (middleware)
  - Enforcers
- Scalable Verification
  - Statistical Model Checking: Semantic Important Sampling

Evolving Hardware

- Multicore Scheduling