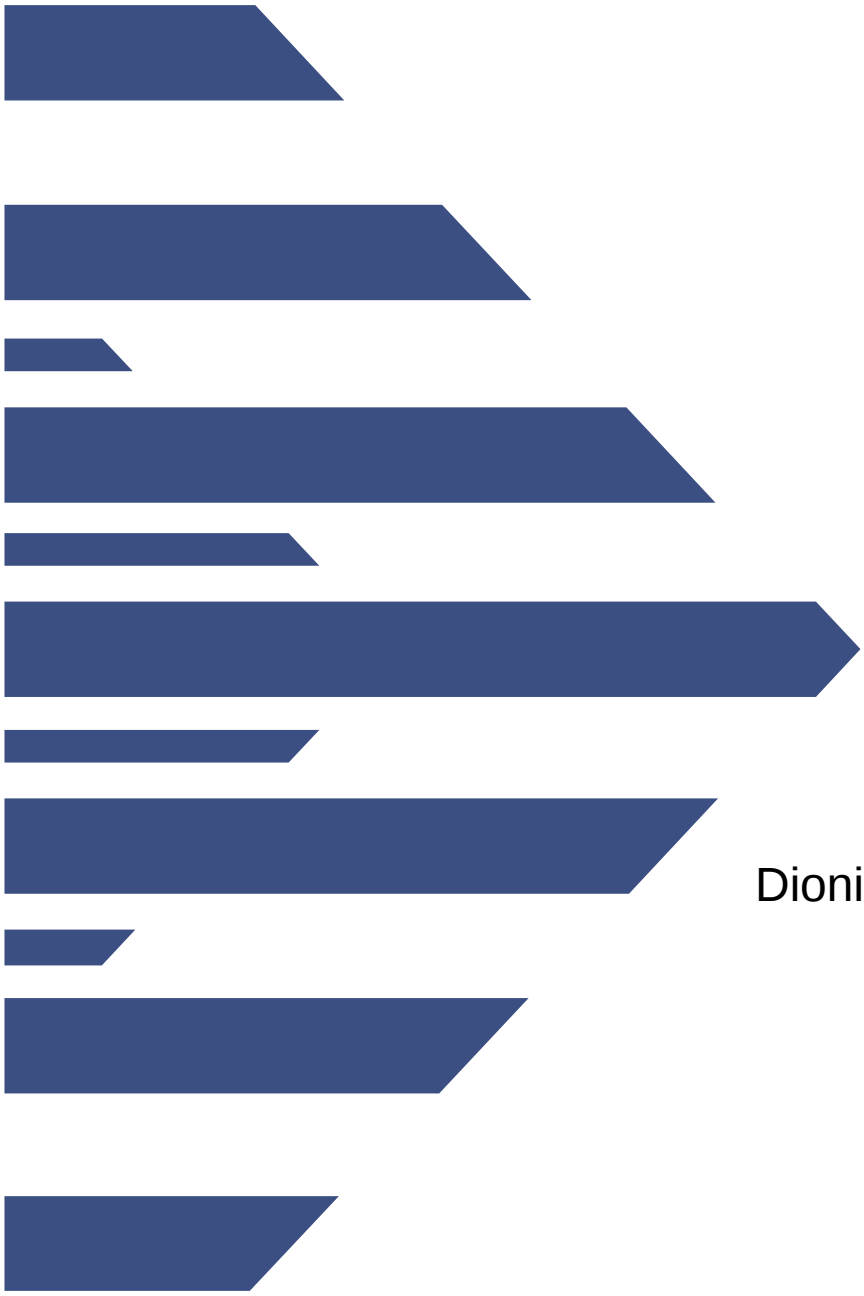




CPS Research

Dionisio de Niz and Sagar Chaki



Software Engineering Institute

[Distribution Statement A] Approved for public release and unlimited distribution

Carri

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use.

Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

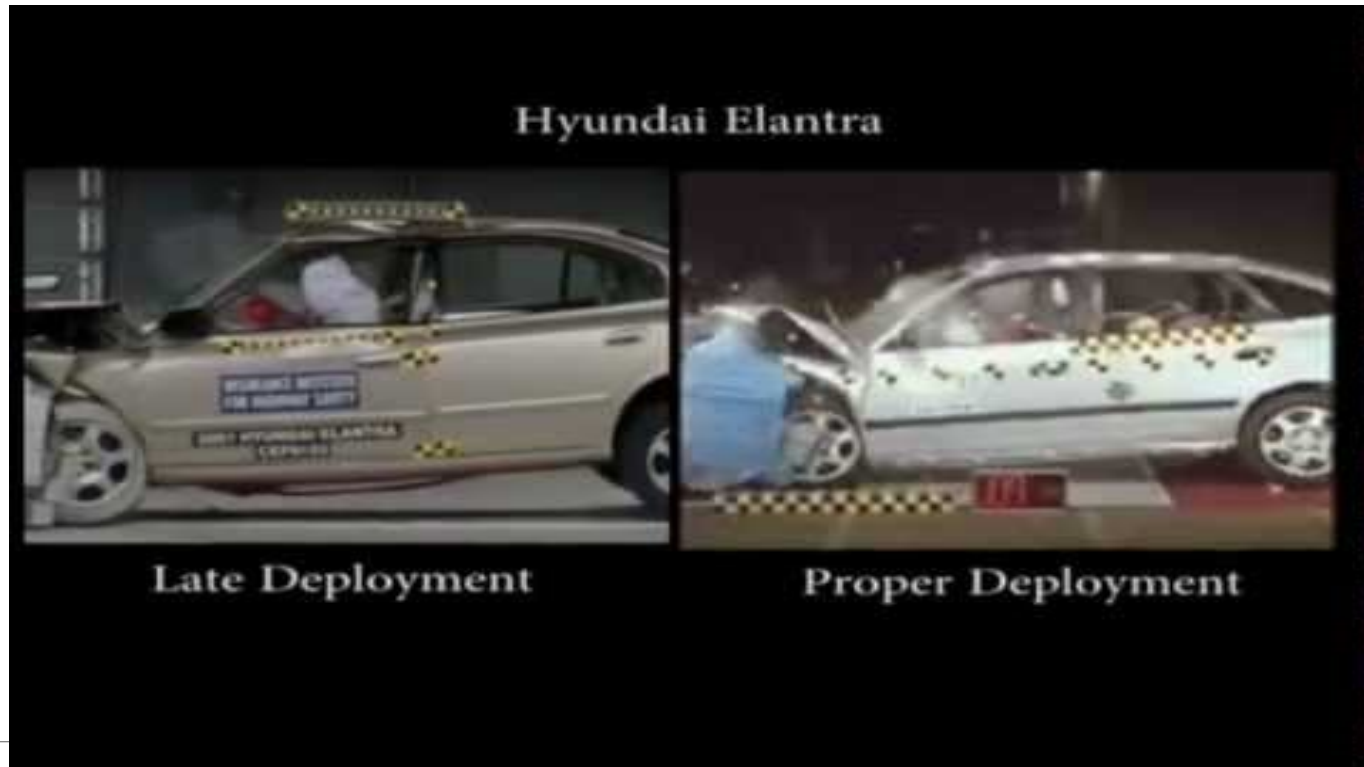
Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM17-0337



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

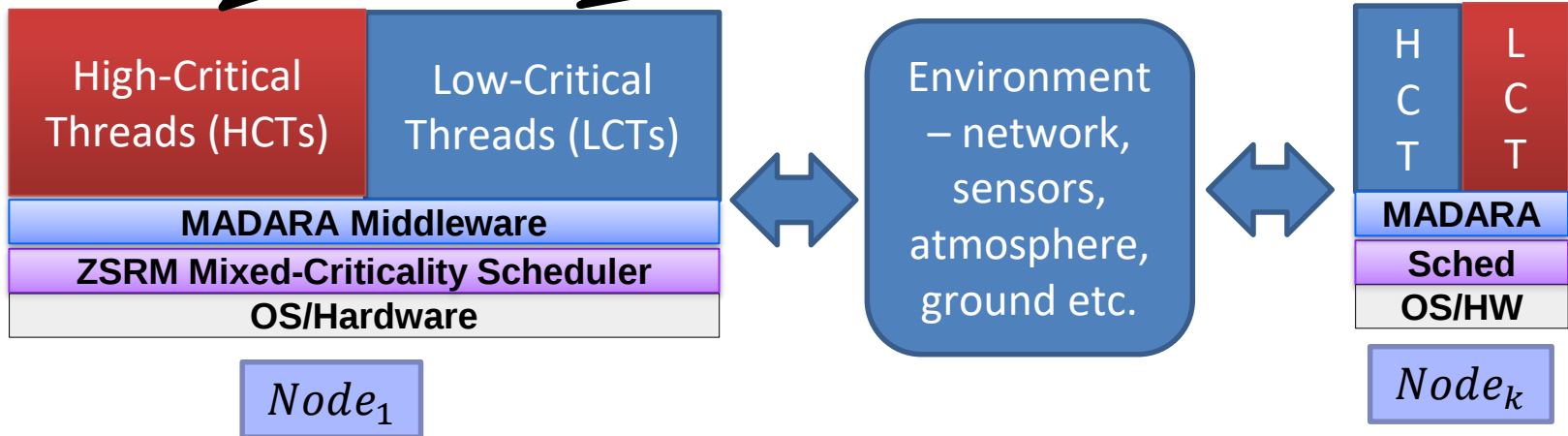
- Domain Specific Language: constrained executable
- Distributed Shared-Variables Middleware: synchronous computation
- Statistical Model Checking:
 - Montecarlo Simulations
 - Important Sampling / Semantic Important Sampling



Project 1: Distributed Adaptive Real-Time

Software for guaranteed requirements, e.g., collision avoidance protocol must ensure absence of collisions

Software for probabilistic requirements, e.g., adaptive path-planner to maximize area coverage within deadline



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

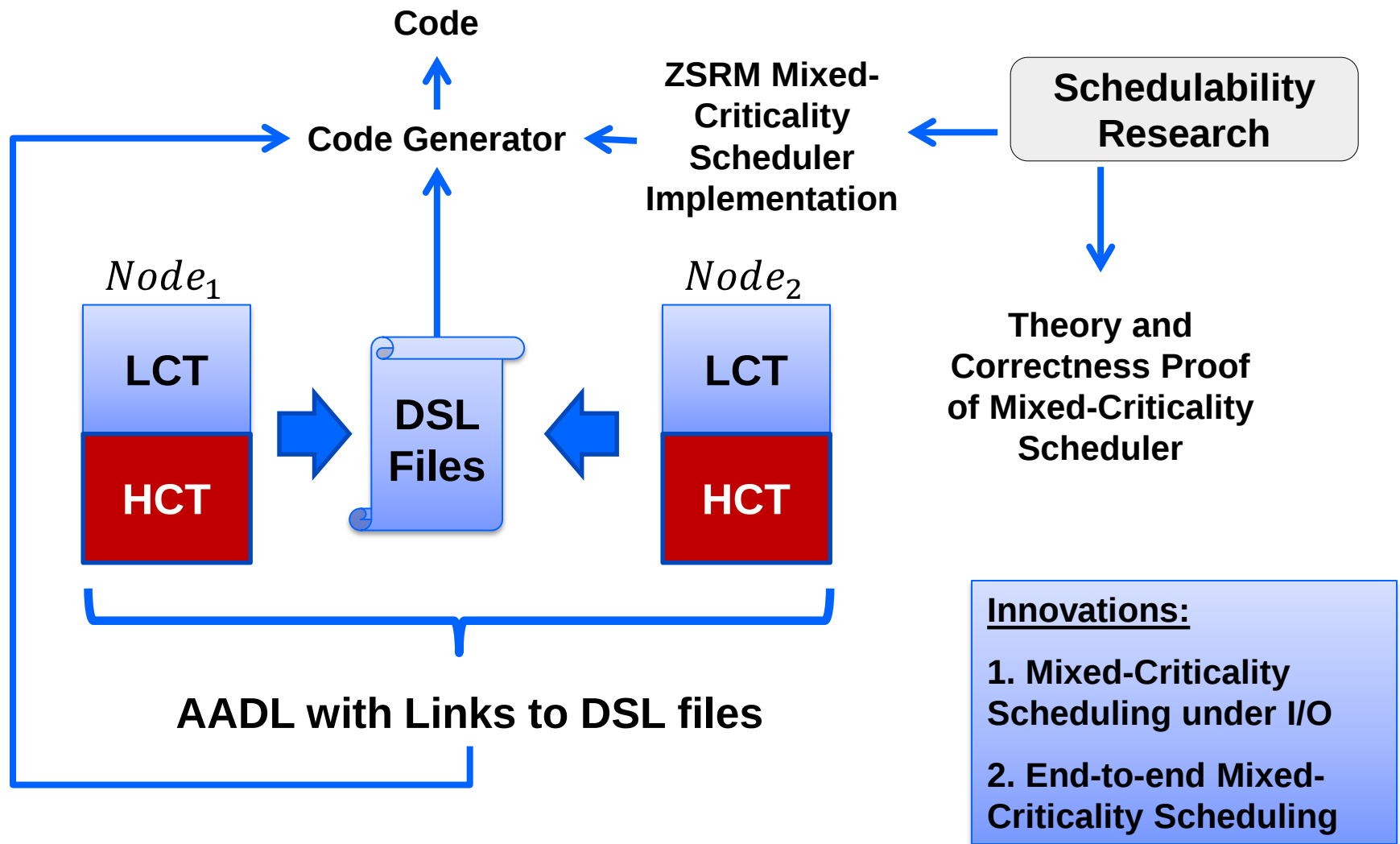
DMPL : Behavior

- 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

Implemented as a
DART Workbench.
Happy to share.

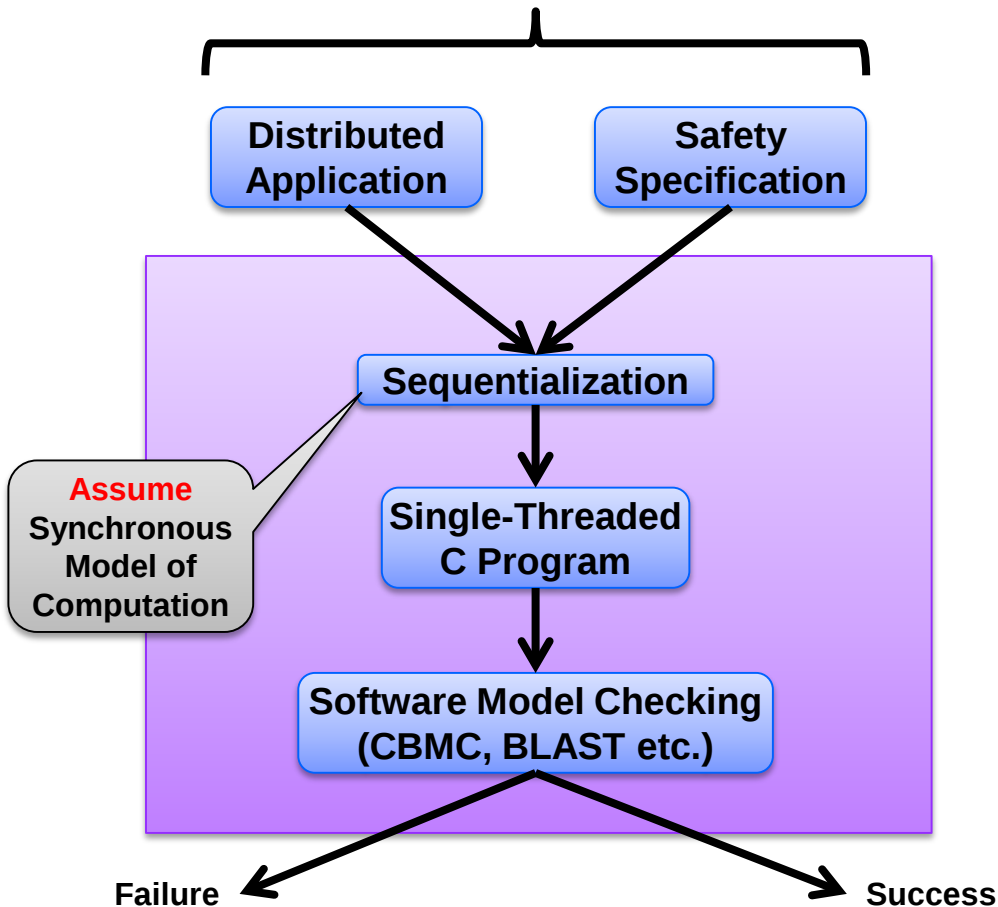


Real-Time Schedulability



Verification

Program in Domain Specific Language



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)

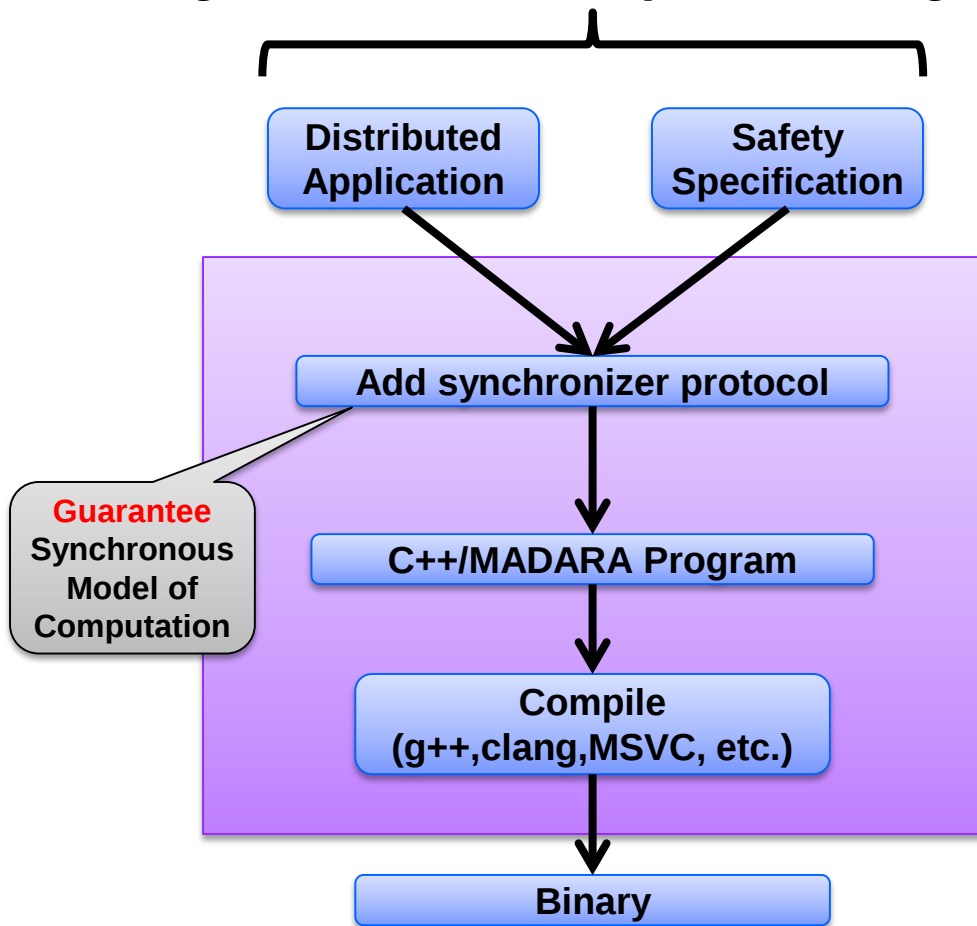
- Computation Tree Logic (CTL), Linear Temporal Logic (LTL), ...

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

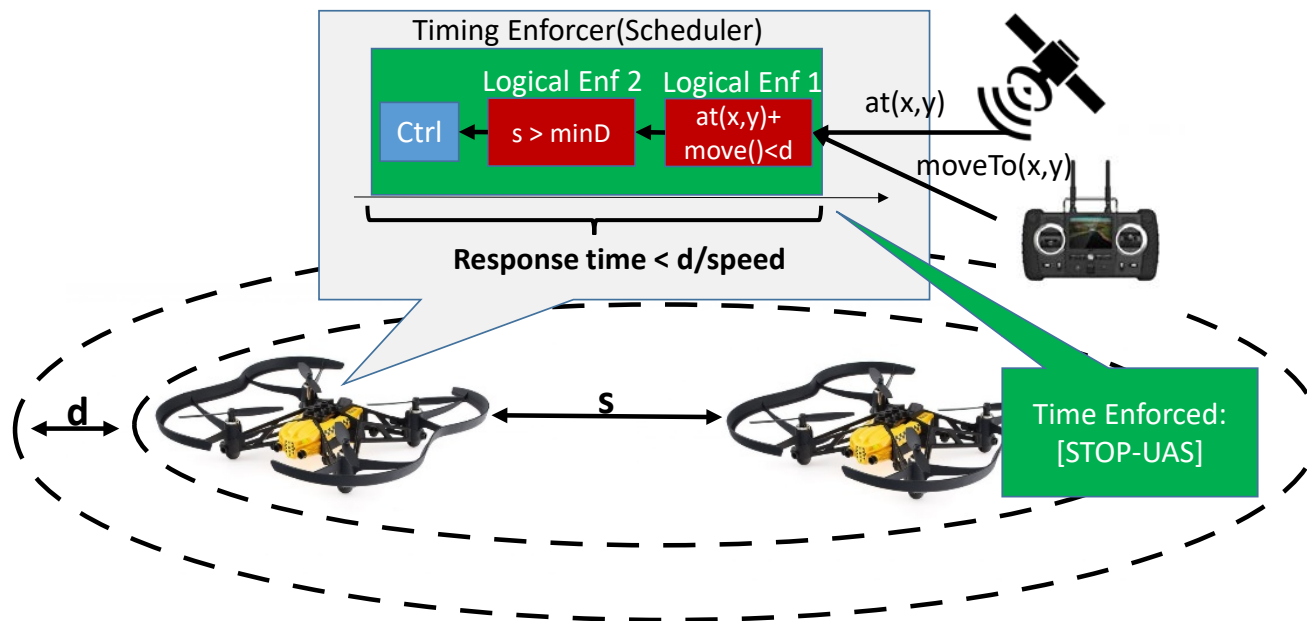
Implicit “receive” thread on each node

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

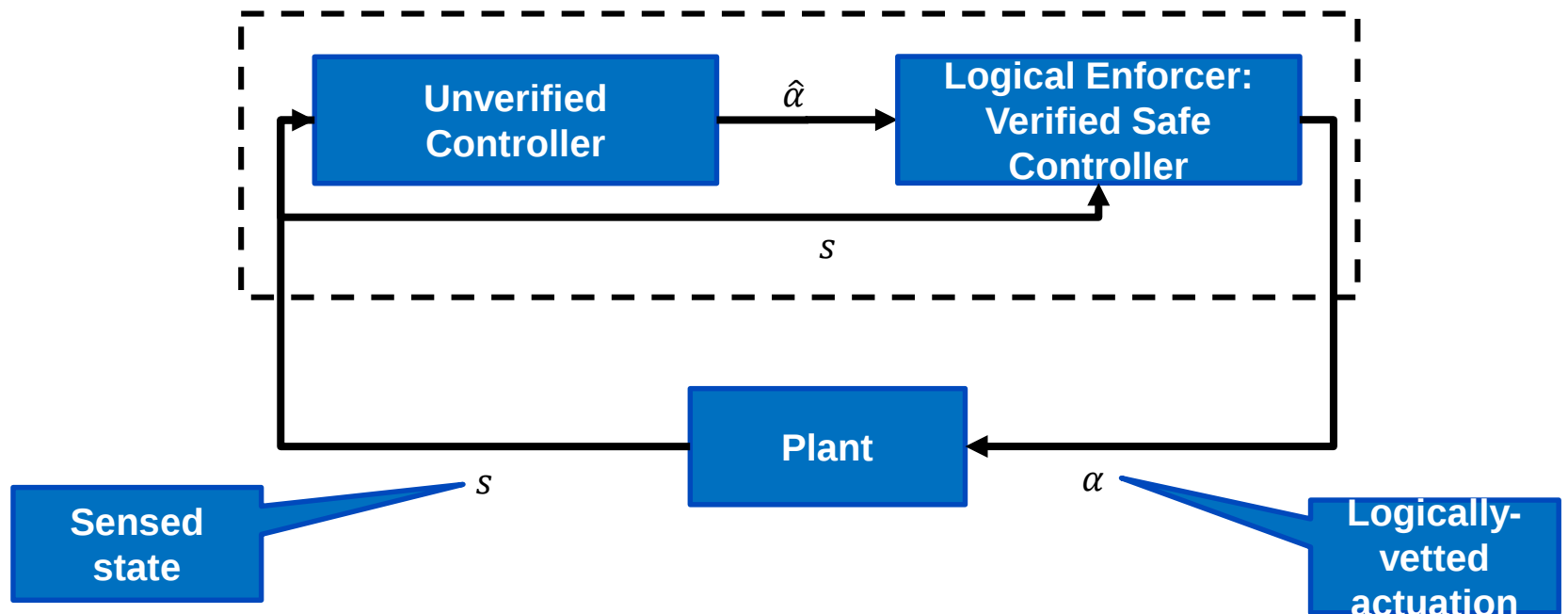
Portable to different OSes (Windows, Linux, Android etc.) and networking technology (TCP/IP, UDP, DDS etc.)



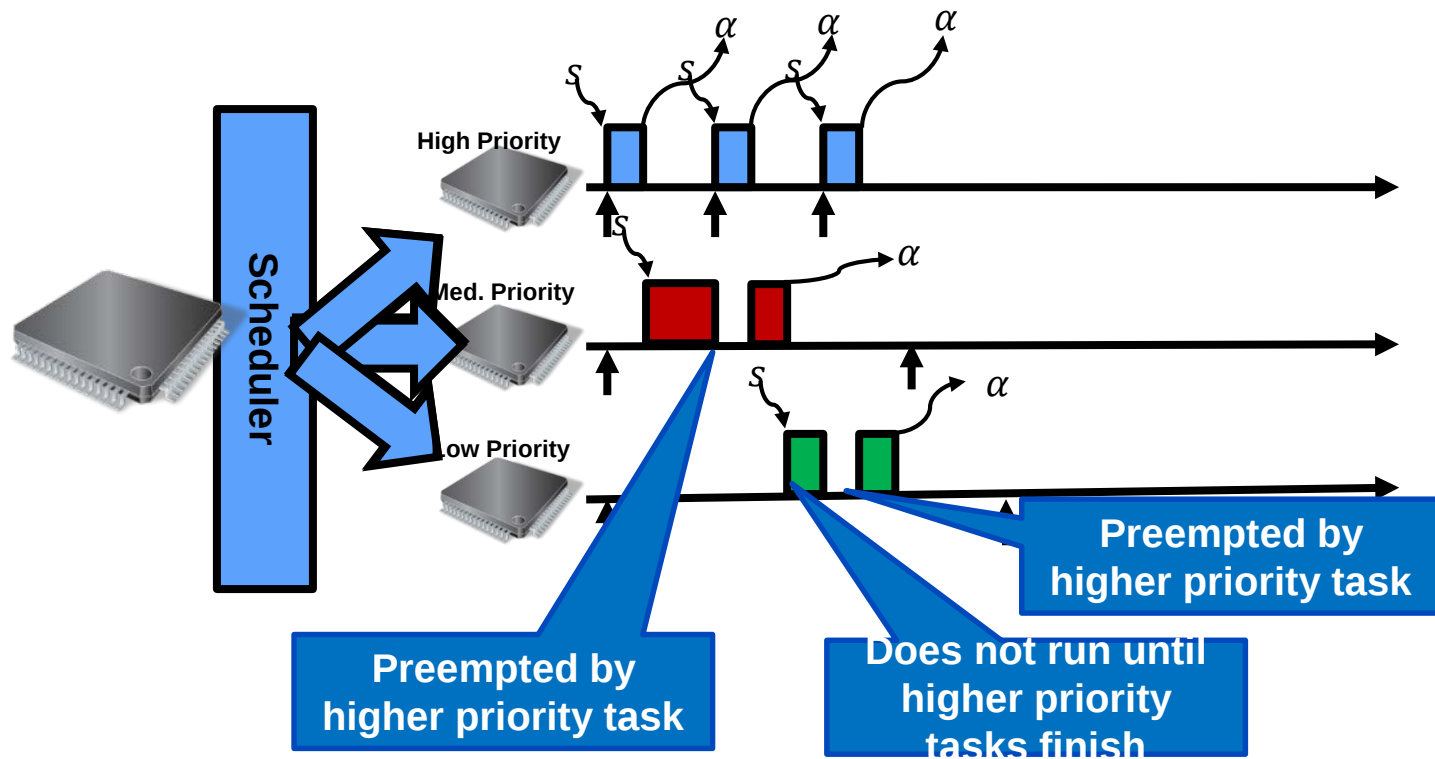
Project 2: Certifiable Distributed Runtime Assurance



Sense Actuation Loop + Logical Enforcer



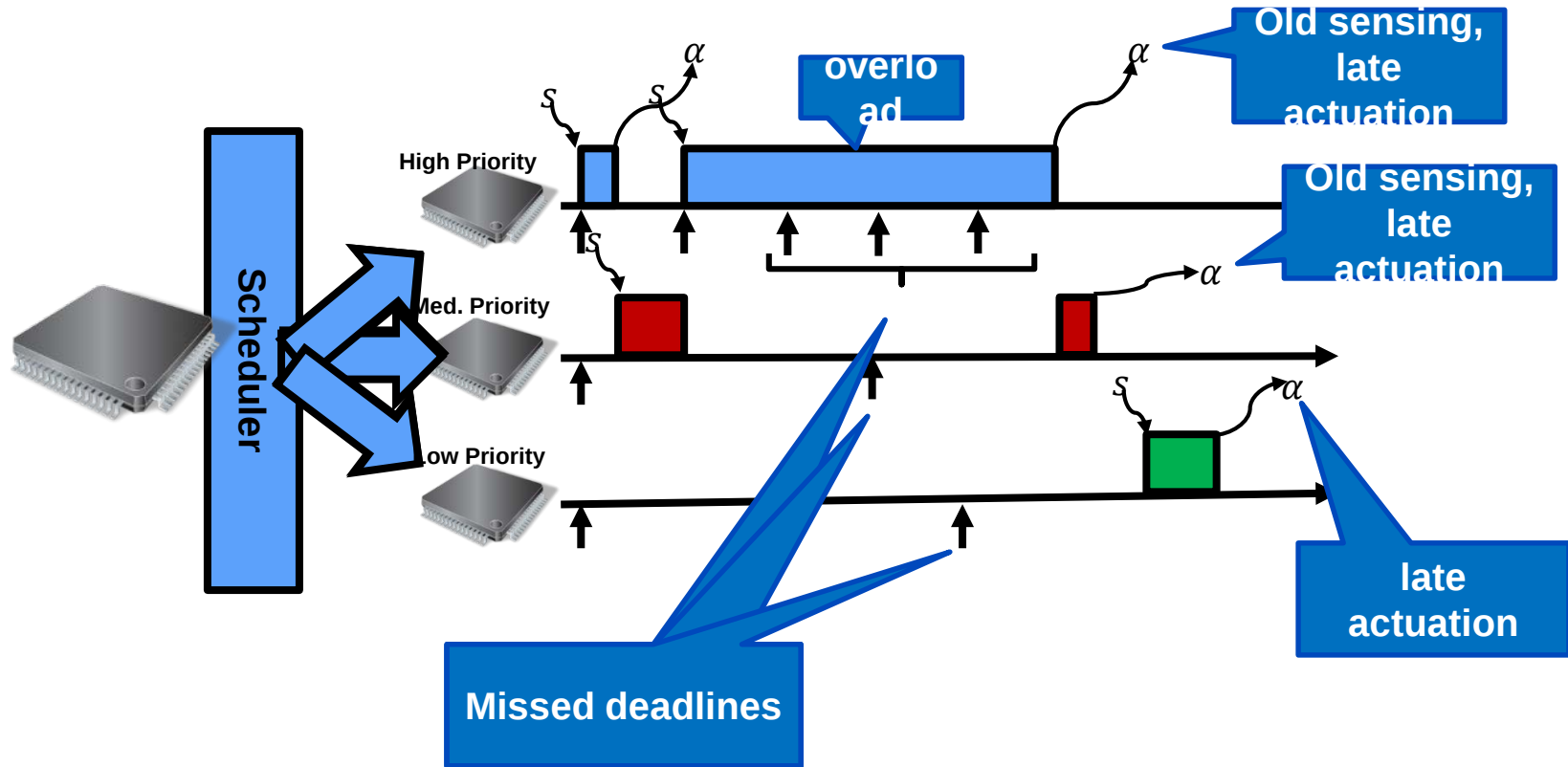
Fixed-Priority Scheduling + Rate Monotonic



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



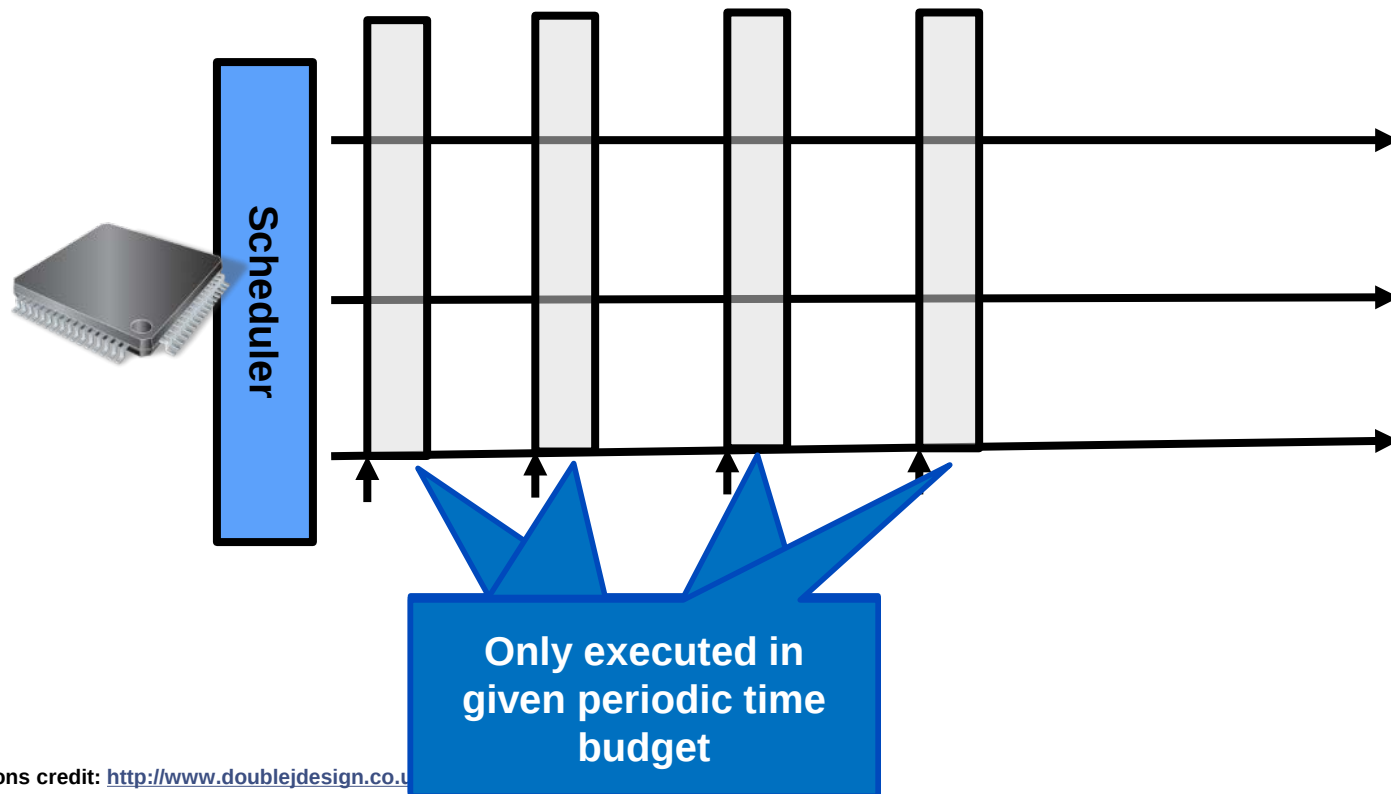
Overload => old sensed data + late actuation



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



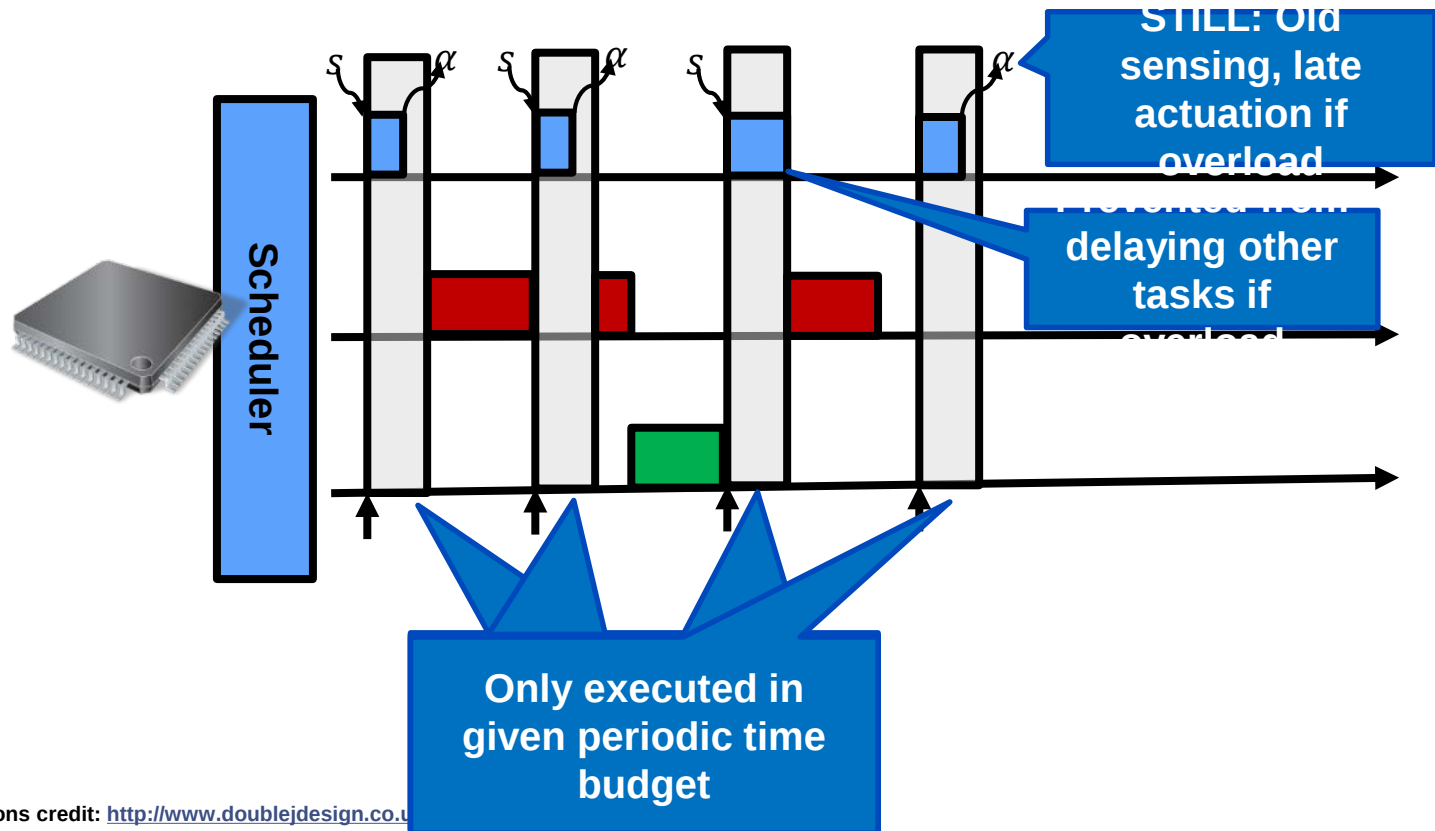
Solution: Enforce timing budgets (timing enforcement)



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



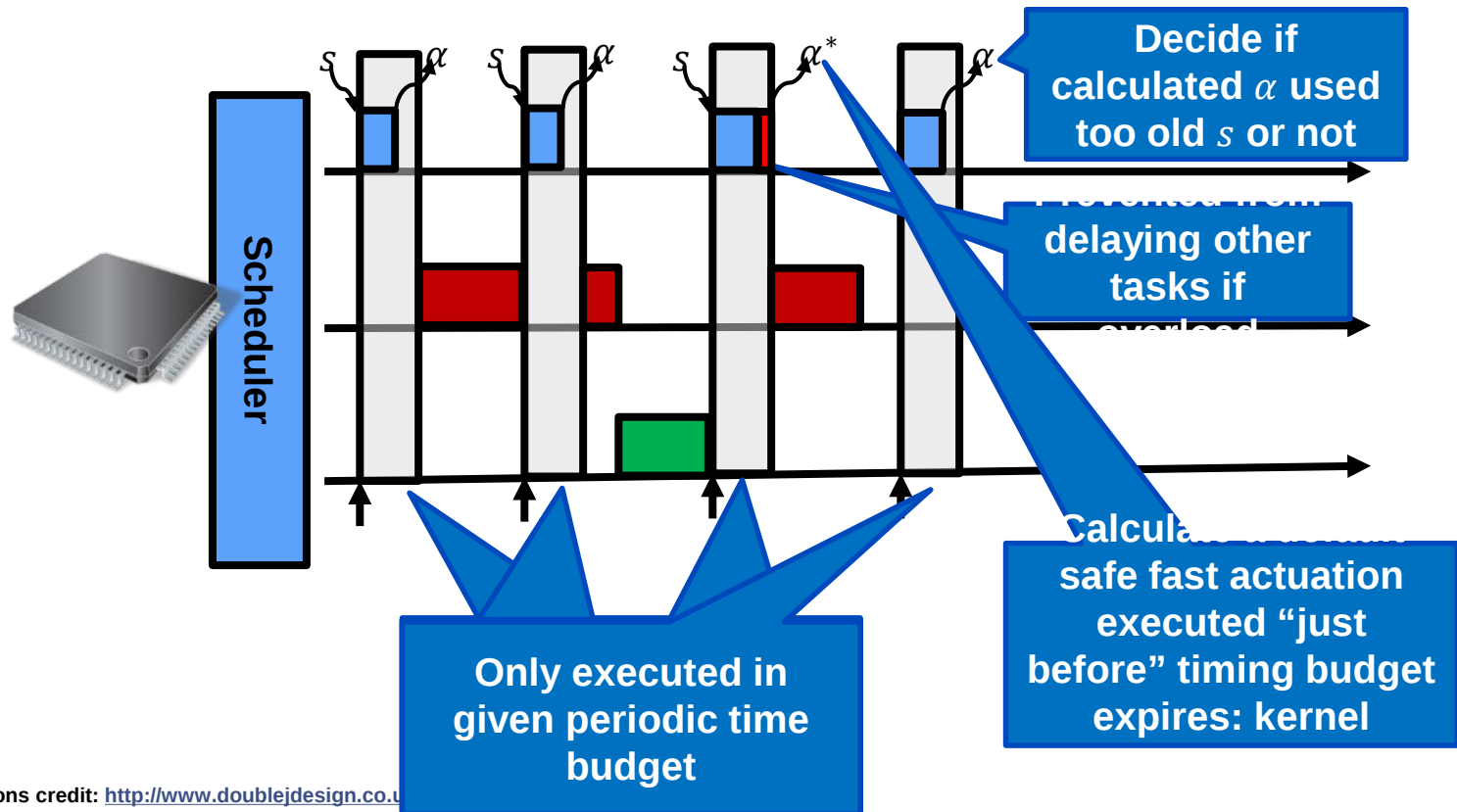
Solution step 1: enforce timing budgets (timing enforcement)



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

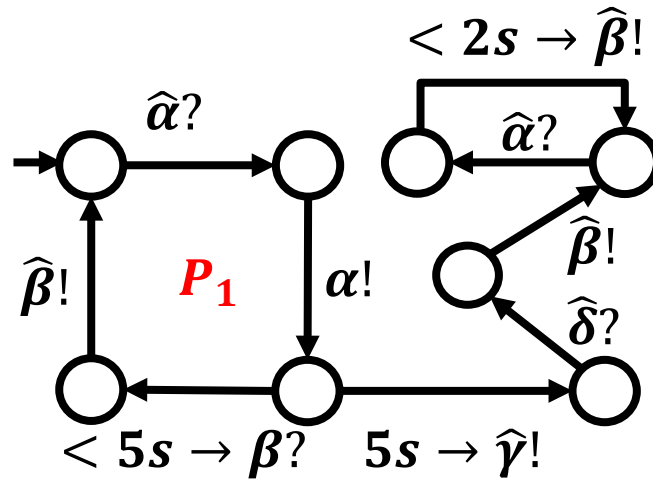


Solution step 2: fast actuation on timing enforcement



CDRA: Approach (1)

Controller Policy

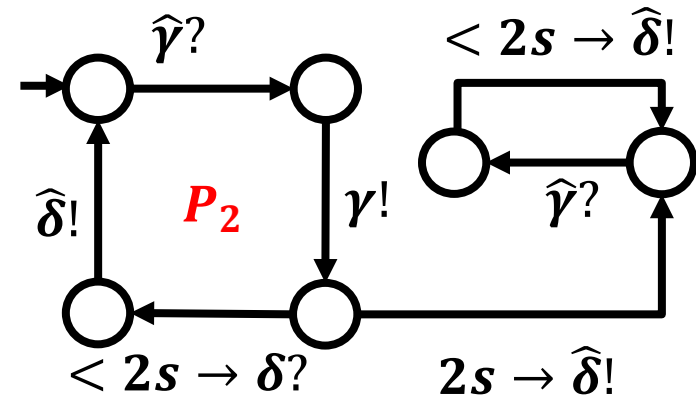


$\Vdash$

*Controller Enforcer
Implementation (E_1)*

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

Logger Policy



$\Vdash$

*Logger Enforcer
Implementation (E_2)*

*Controller
Component (C_1)*

$\{\alpha?, \beta!\}$

$\{\gamma?, \delta!\}$

*Logger
Component (C_2)*

Node



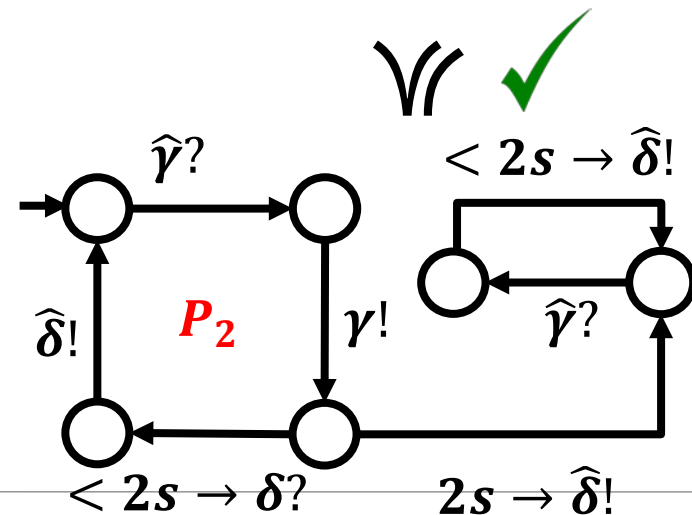
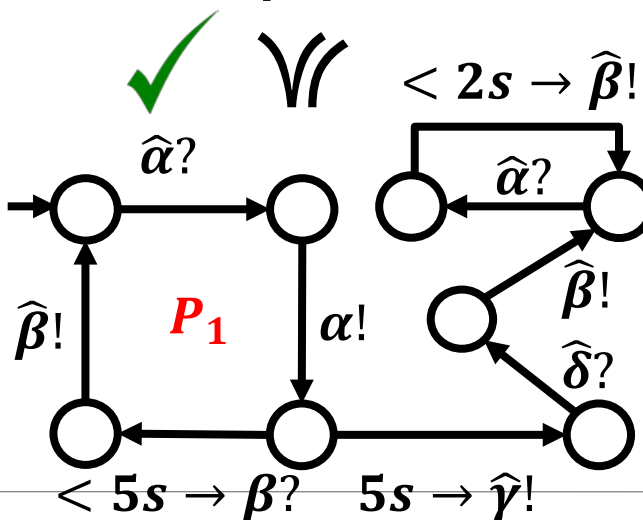
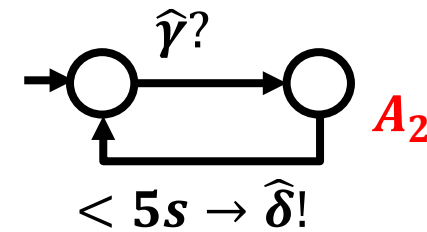
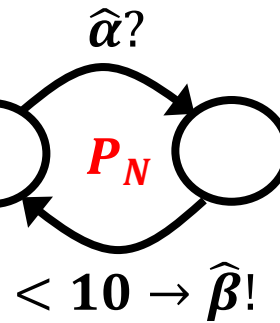
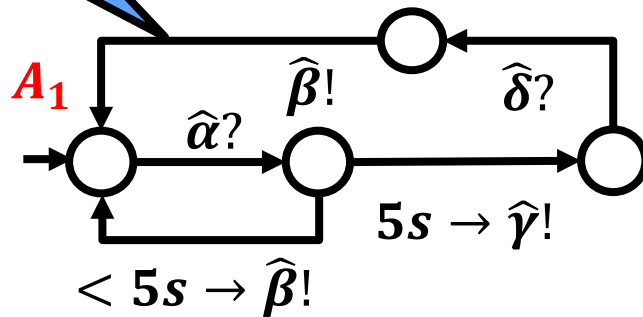
CDRA: Approach (2)

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

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

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

Controller Assumption



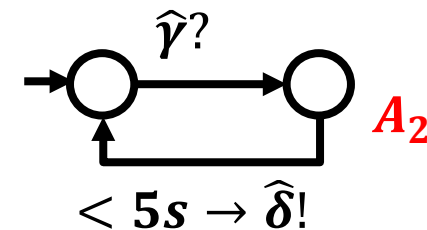
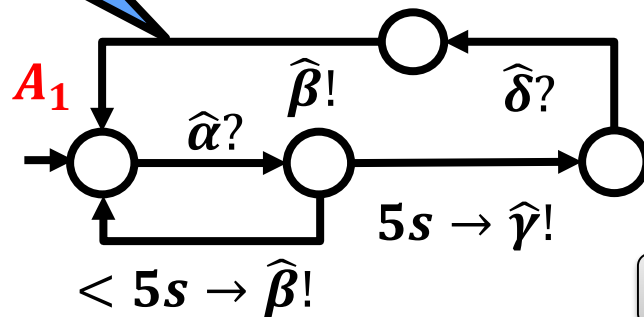
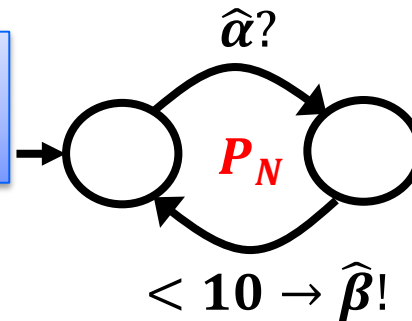
CDRA: Approach (3)

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

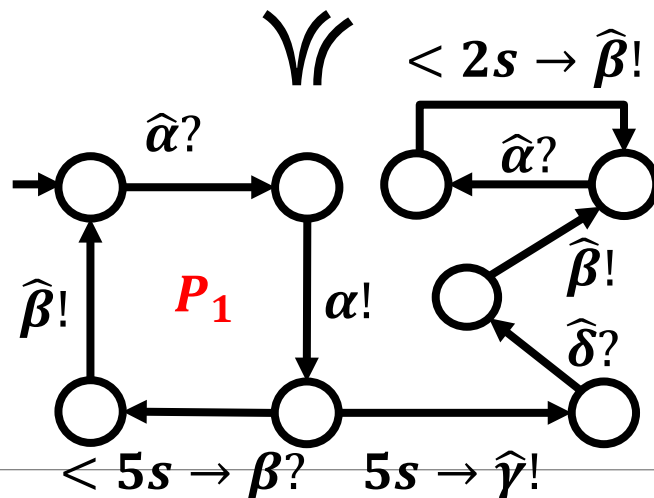
Scale: minimal system
re-verification needed
when a policy or
enforcer is modified

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

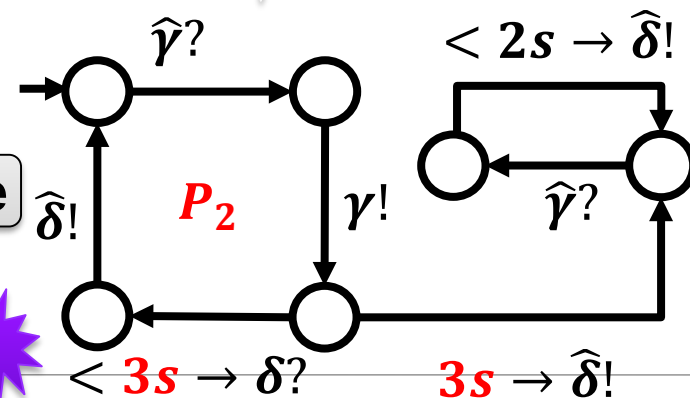
Controller
Assumption



Re-verification



Change

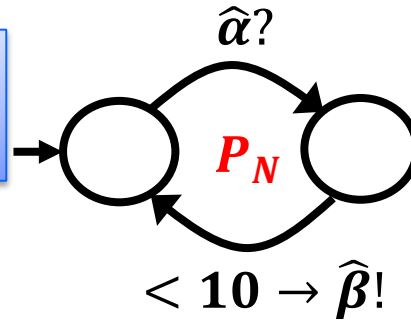


CDRA: Approach (4)

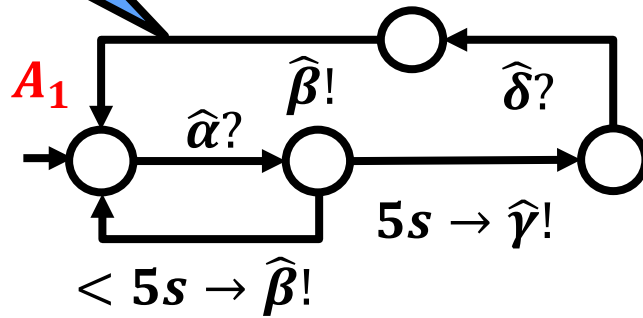
Verify $P_1 \parallel P_2 \leq P_N$
using assume-guarantee

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

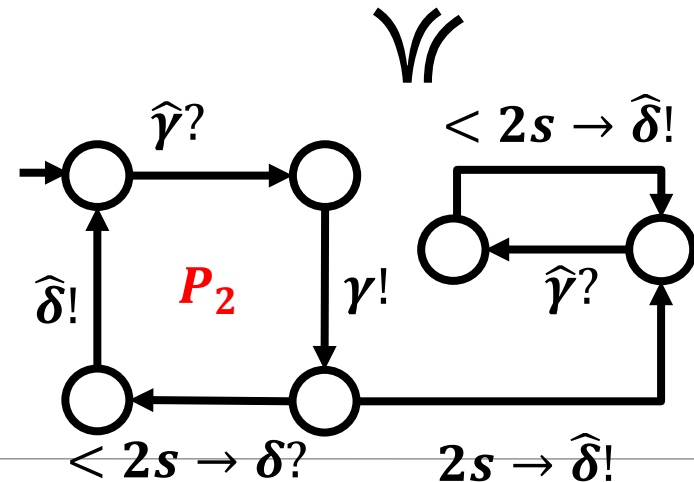
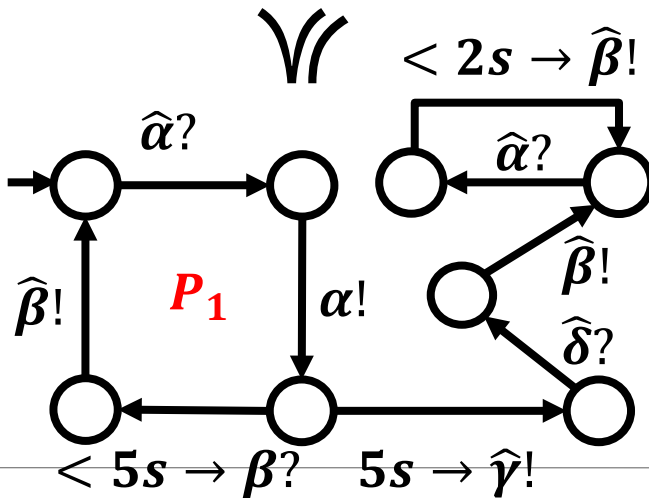
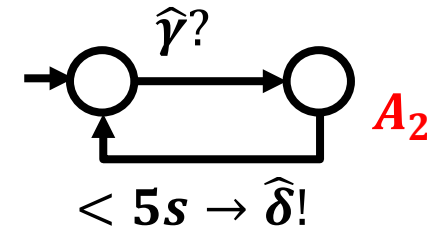
Controller
Assumption



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



$\Vdash$
 $\equiv$



CDRA: Approach (5)

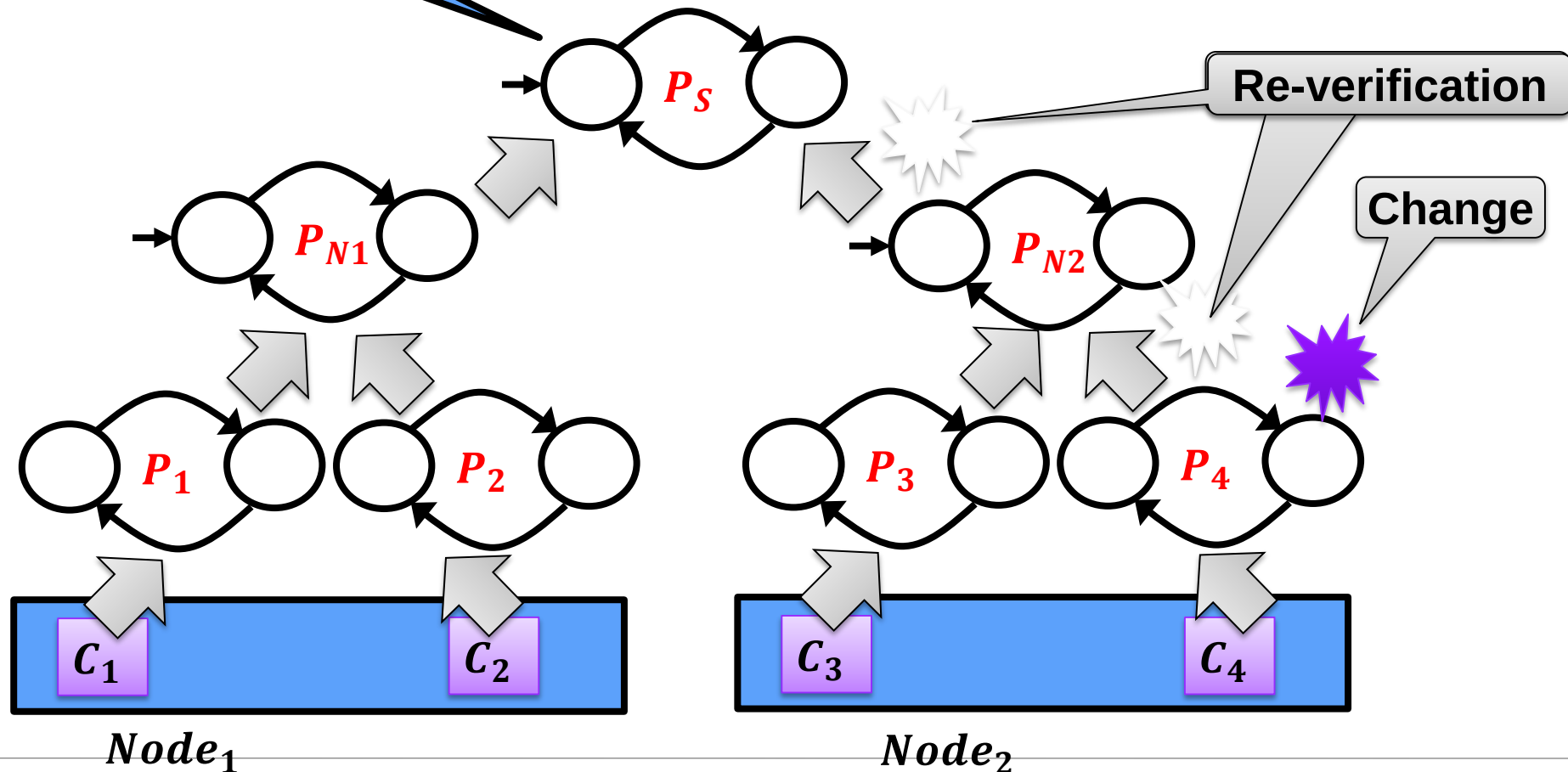
System-level
Policy

1: Verify $E_i \leq P_i$

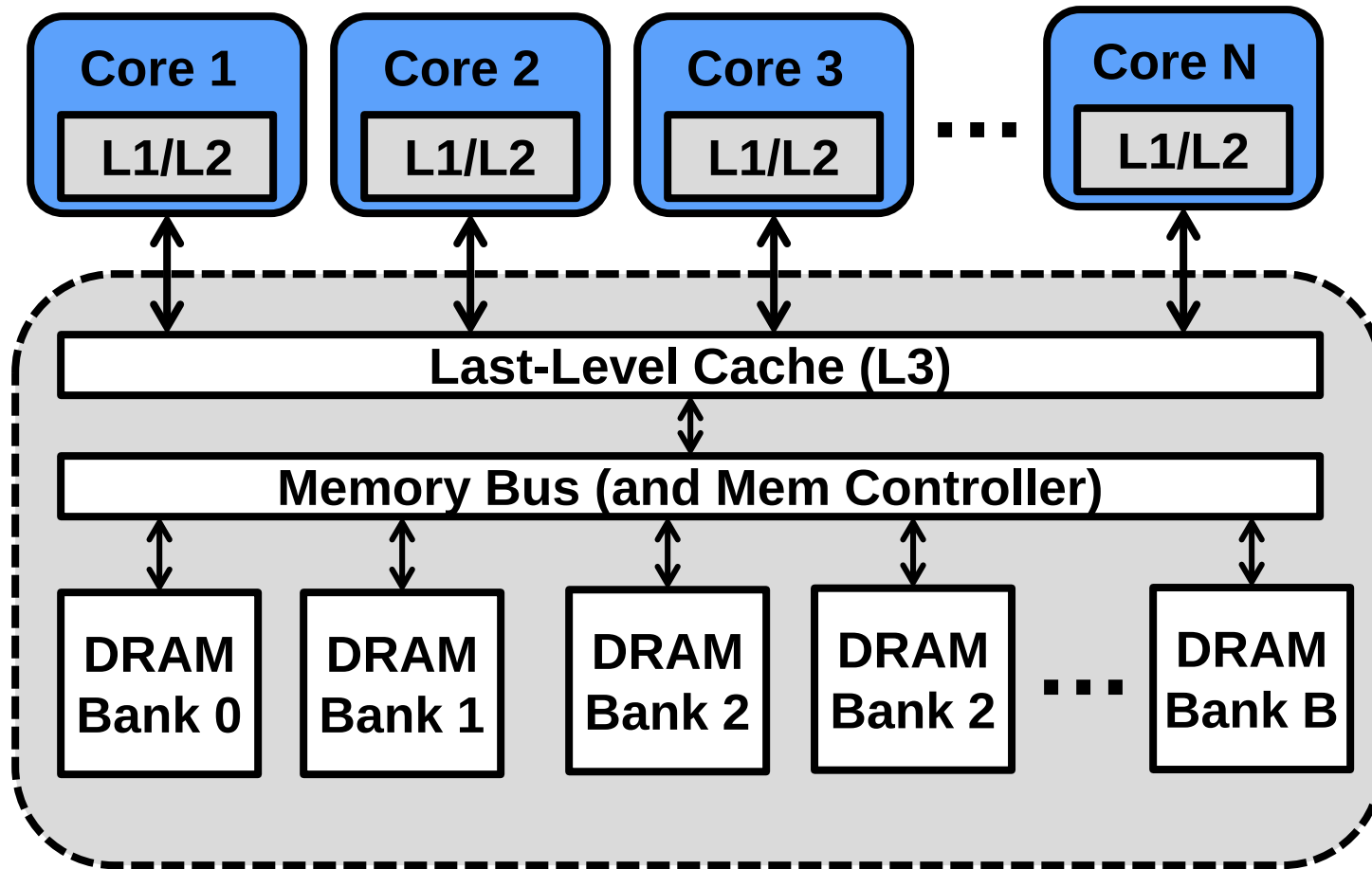
2: Verify $P_1 \parallel P_2 \leq P_{N1}$

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

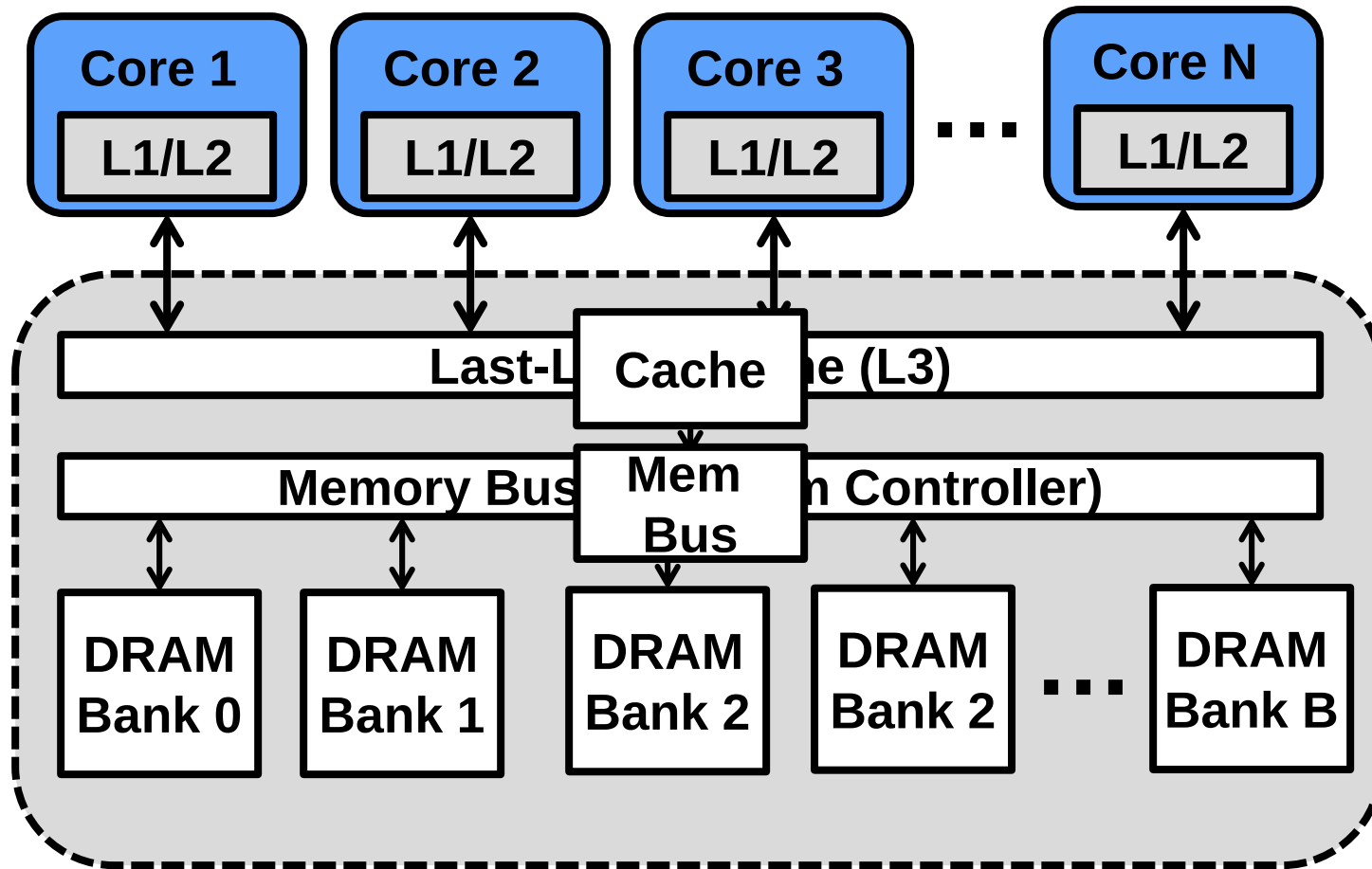
4: Verify $P_{N1} \parallel P_{N2} \leq P_s$



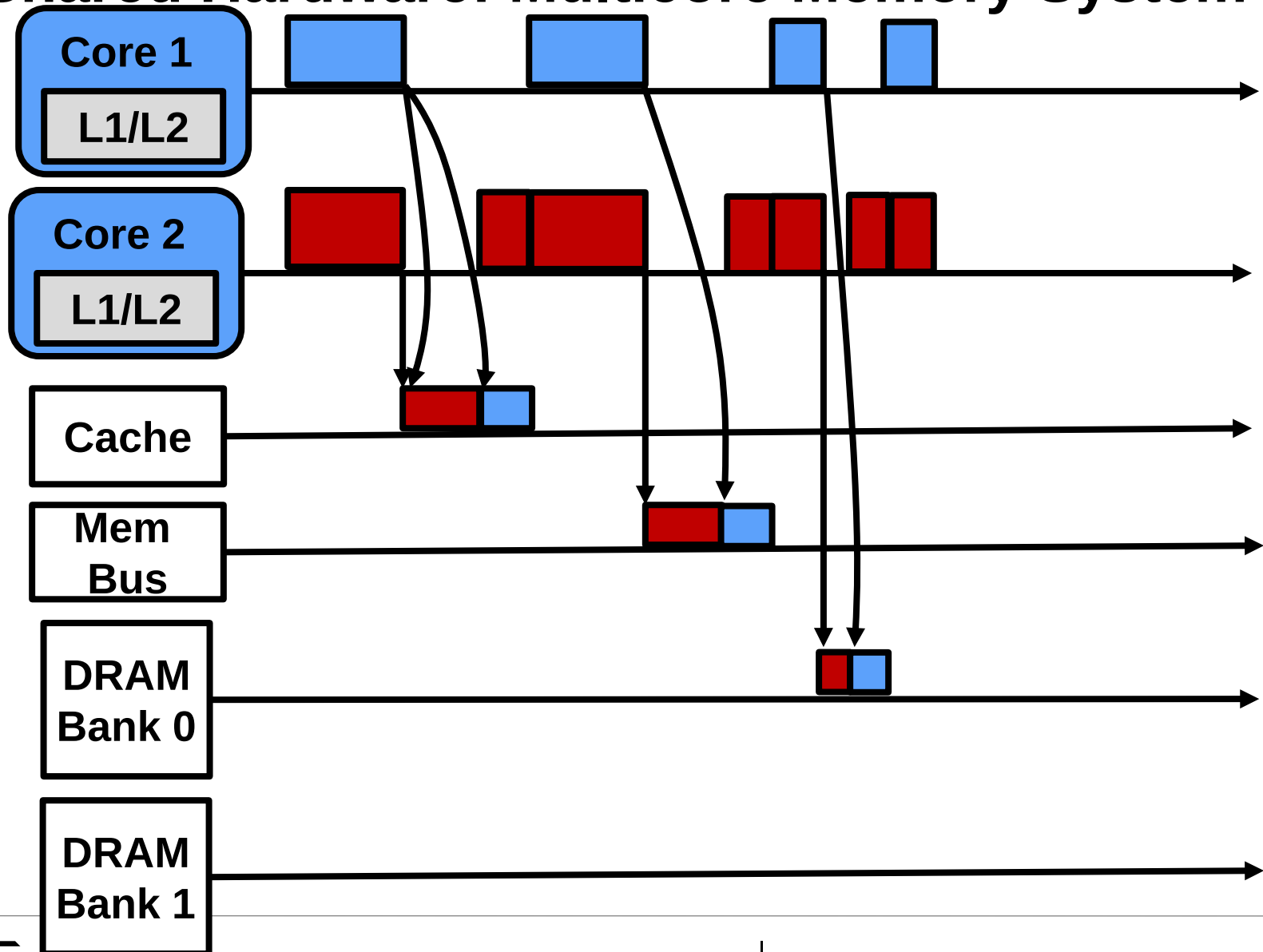
Project 3: Real-Time Scheduling for Multicore



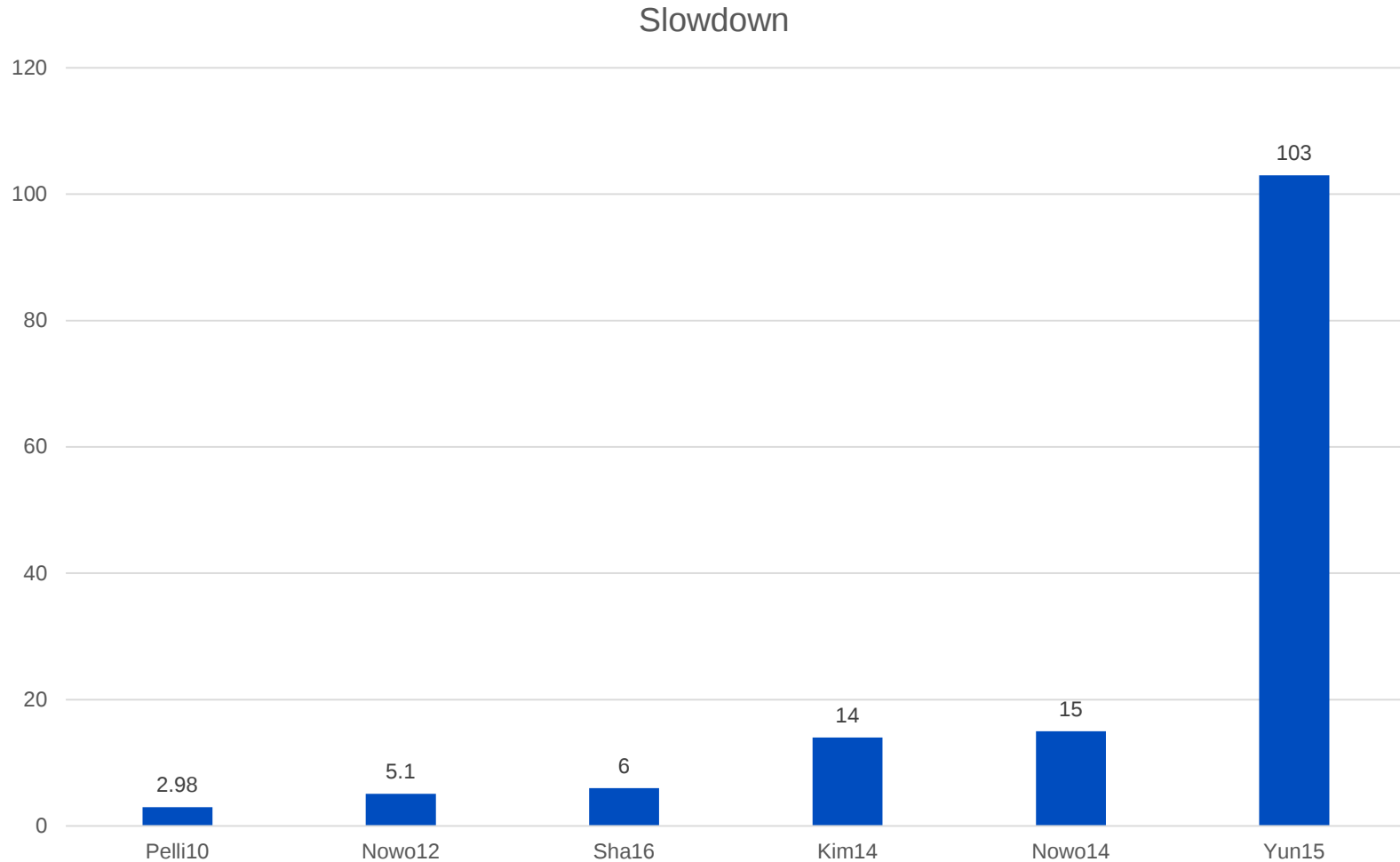
Shared Hardware: Multicore Memory System



Shared Hardware: Multicore Memory System



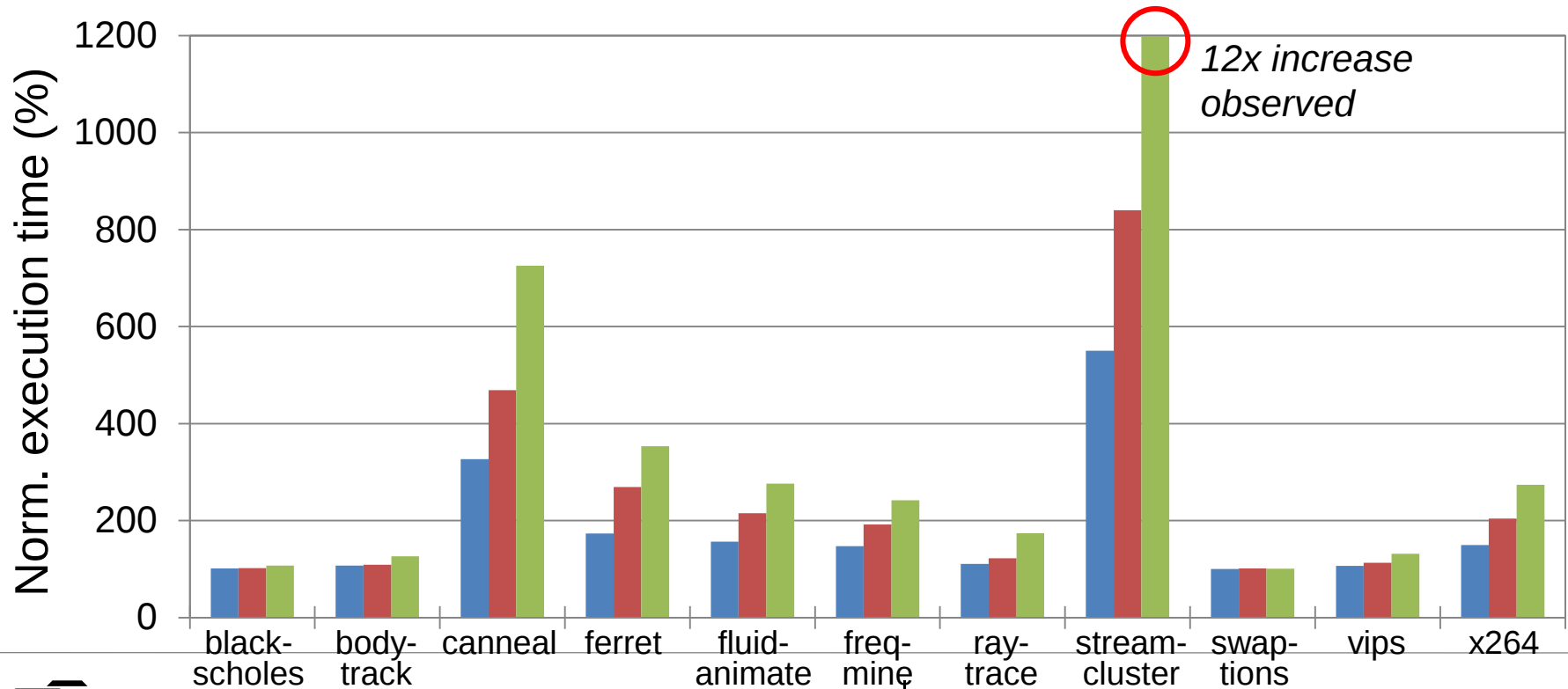
How Bad?



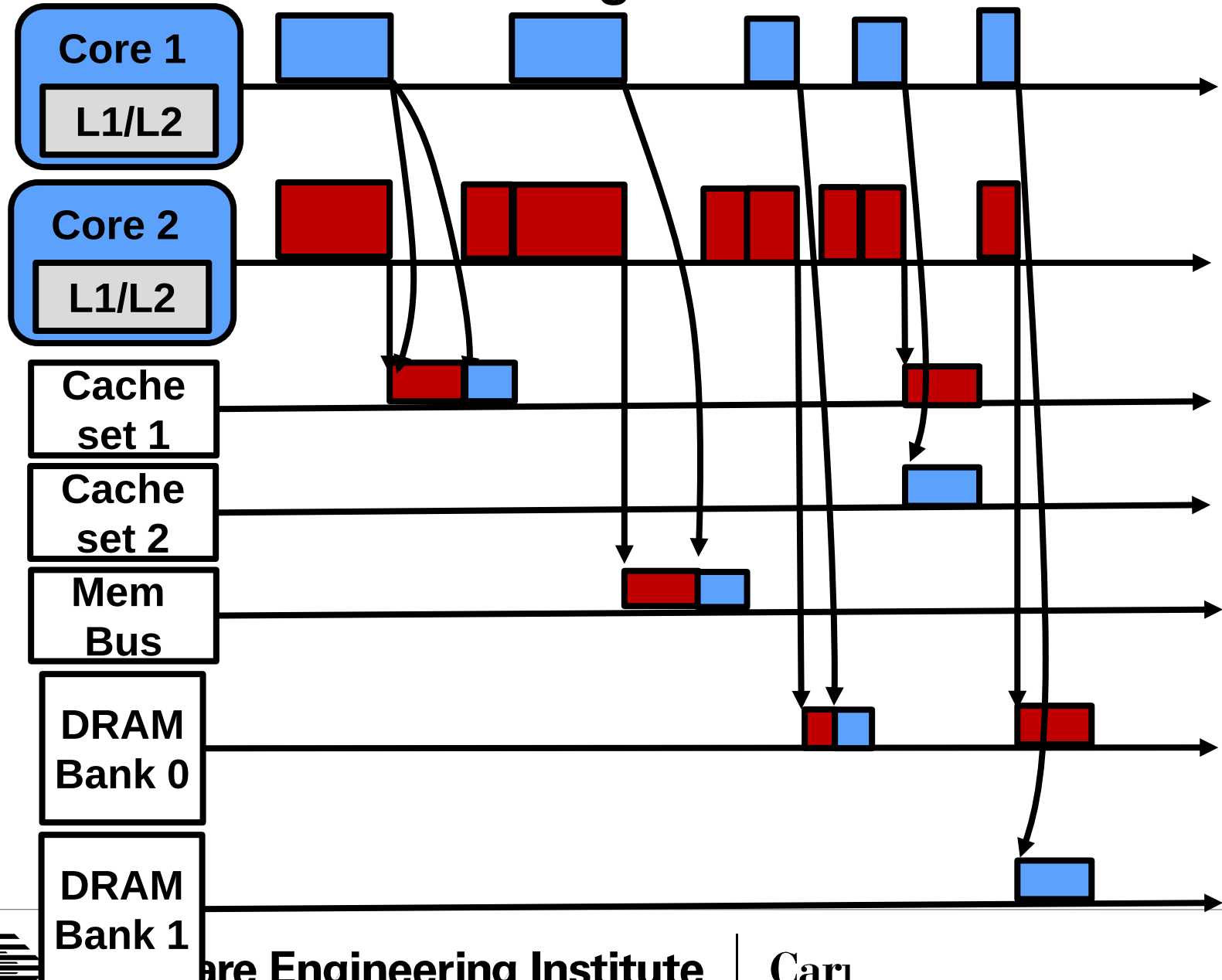
Different for Applications (PARSEC Benchmark)

- 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!



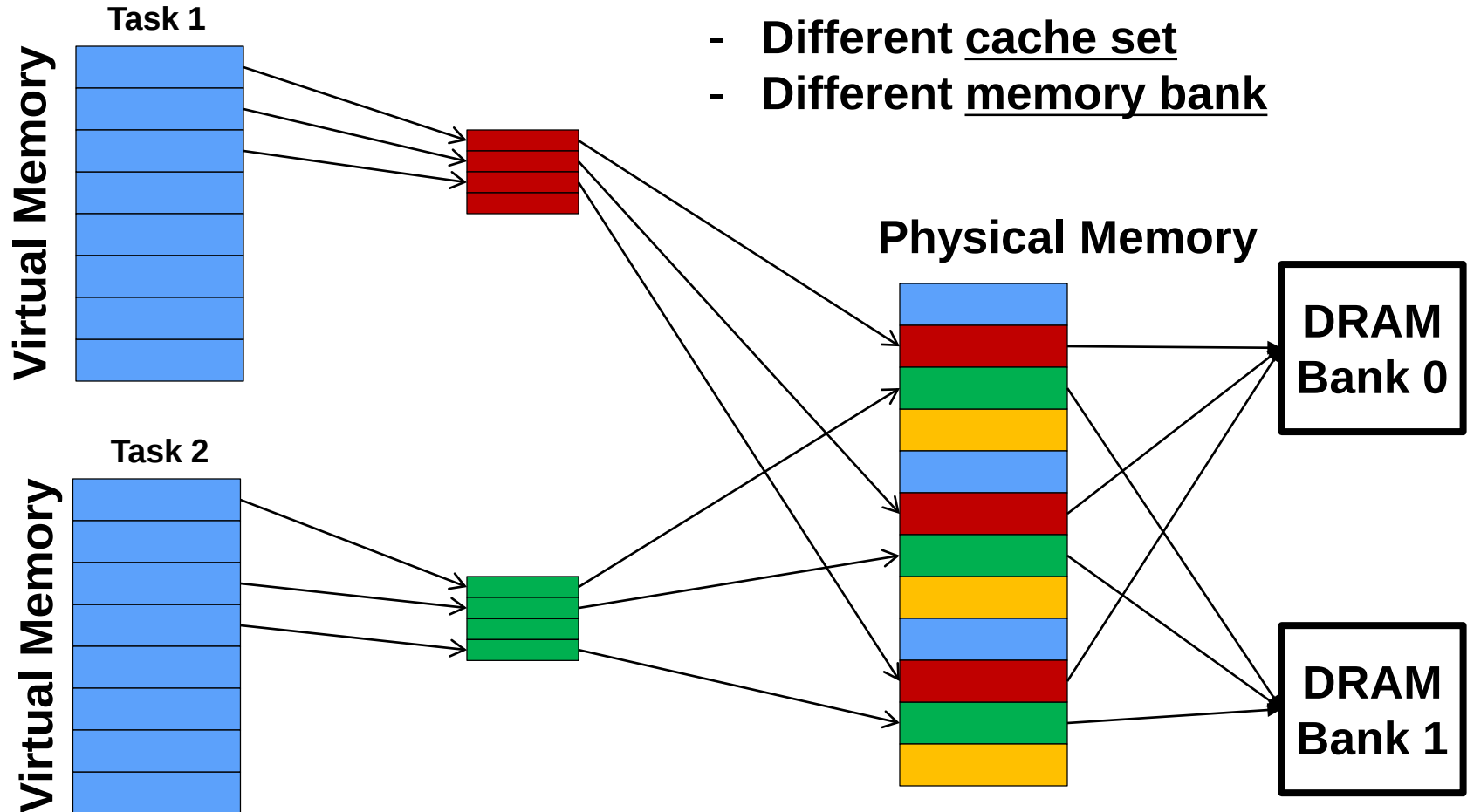
Solution 1: Partitioning



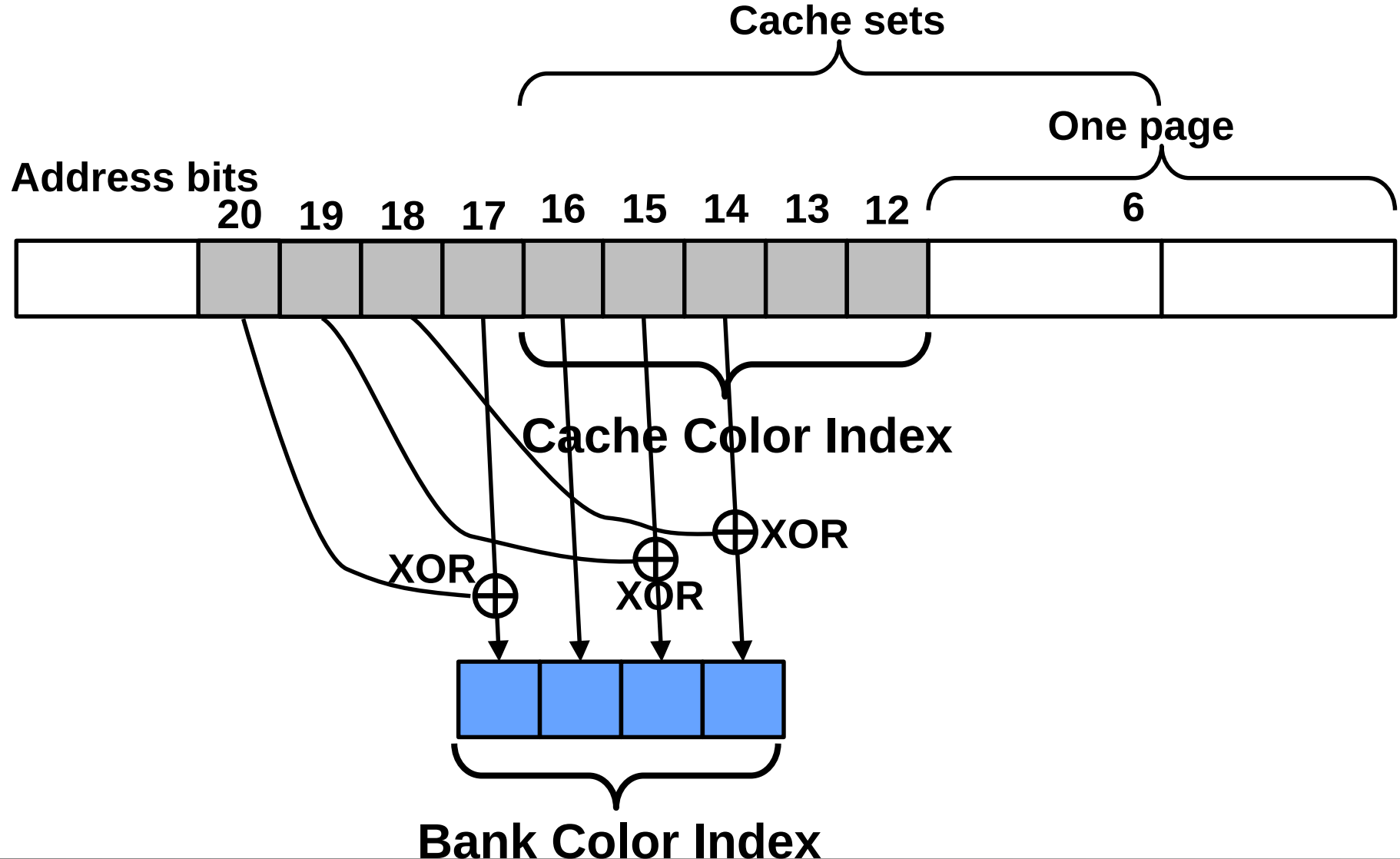
Solutions 1: Virtual Memory “Coloring”

Color: set that do not interfere:

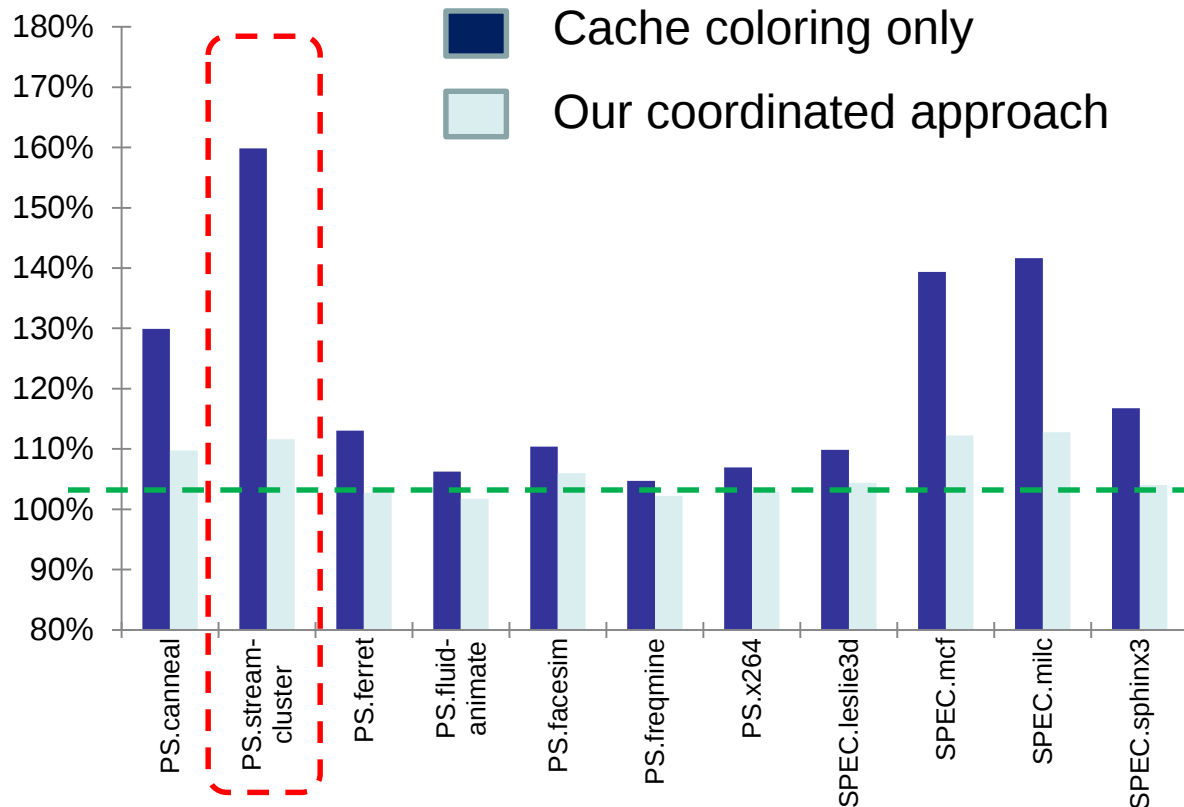
- Different cache set
- Different memory bank



Solution 1: Challenge – Conflicting Partitions



Solution 2: Coordinated Approaches



Challenge: Small Number of Partitions

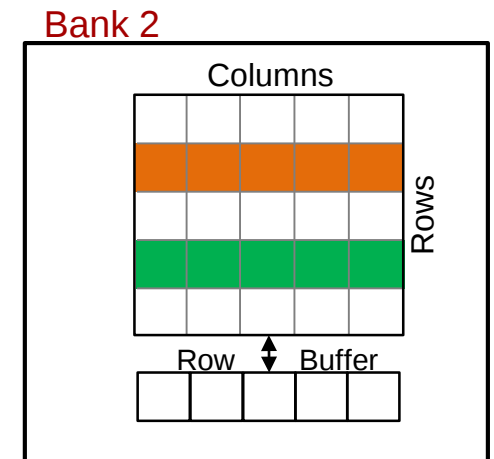
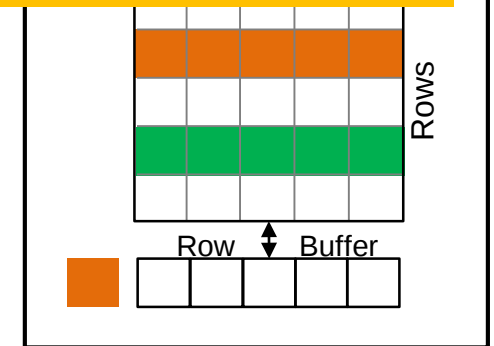
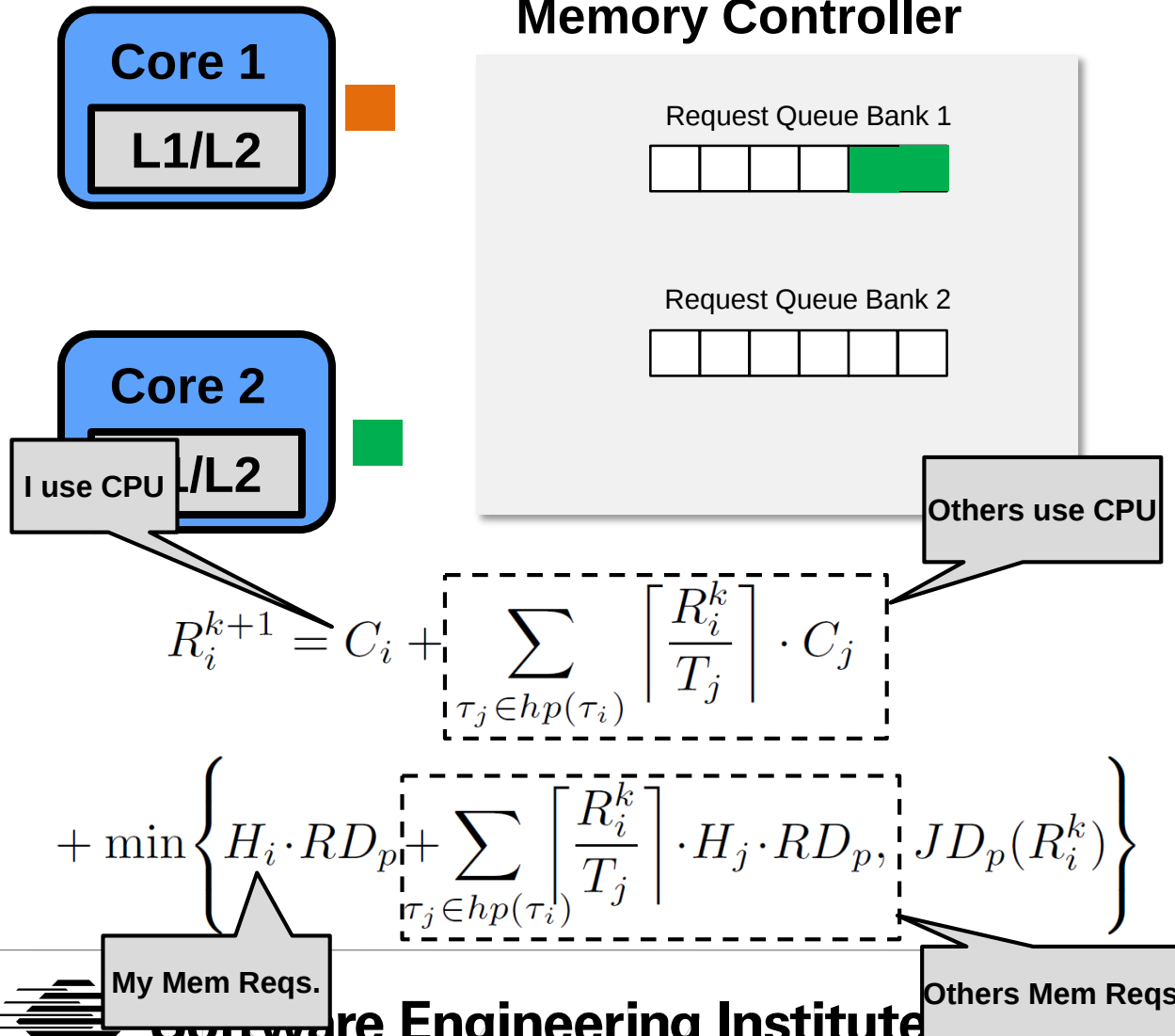


Solution 3: Predictable Sharing of Partitions

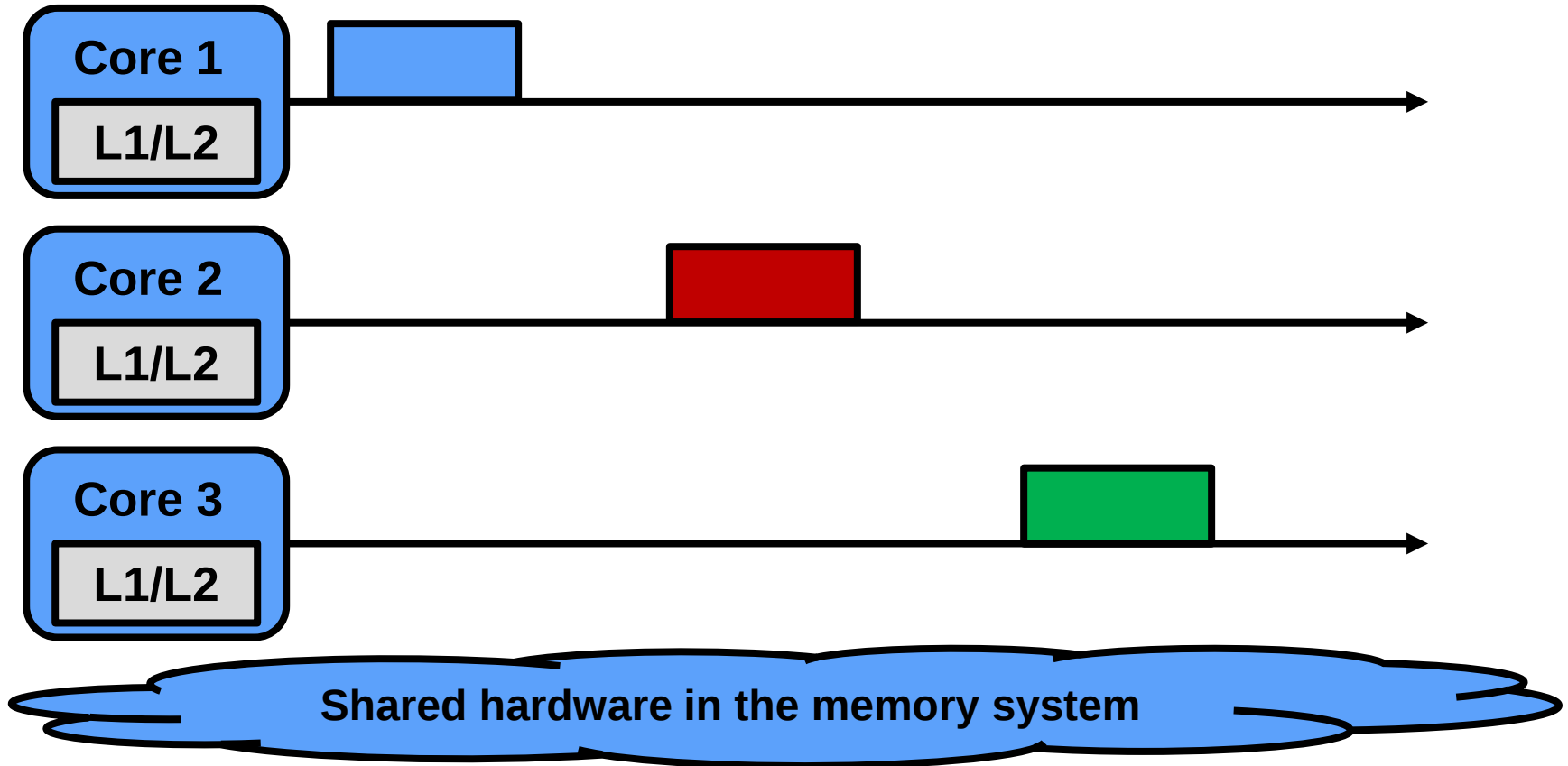
Bank 1

Challenge: Need Processor Documentation (not always public)

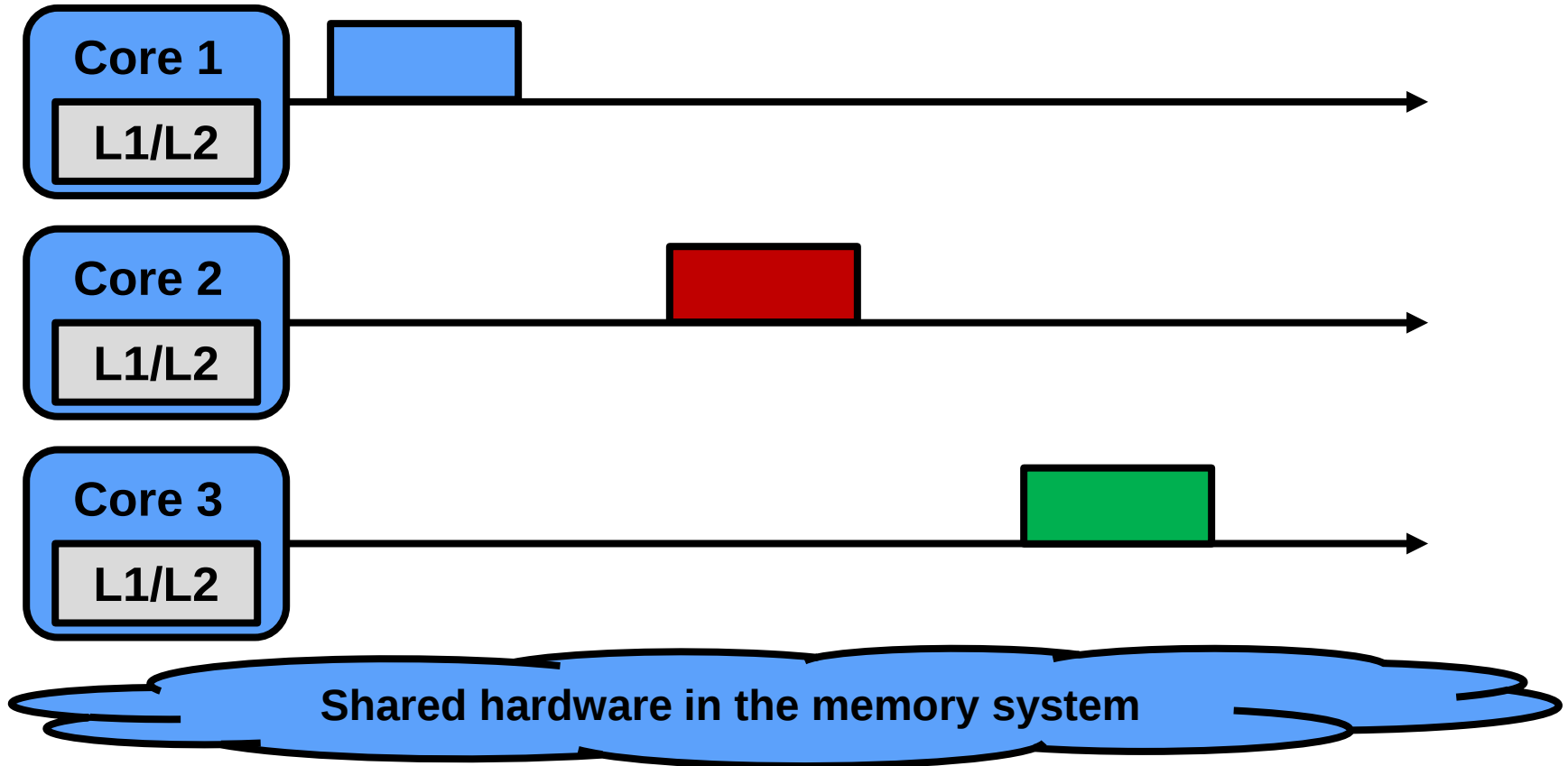
Memory Controller



Solution 4: Black Box Analysis

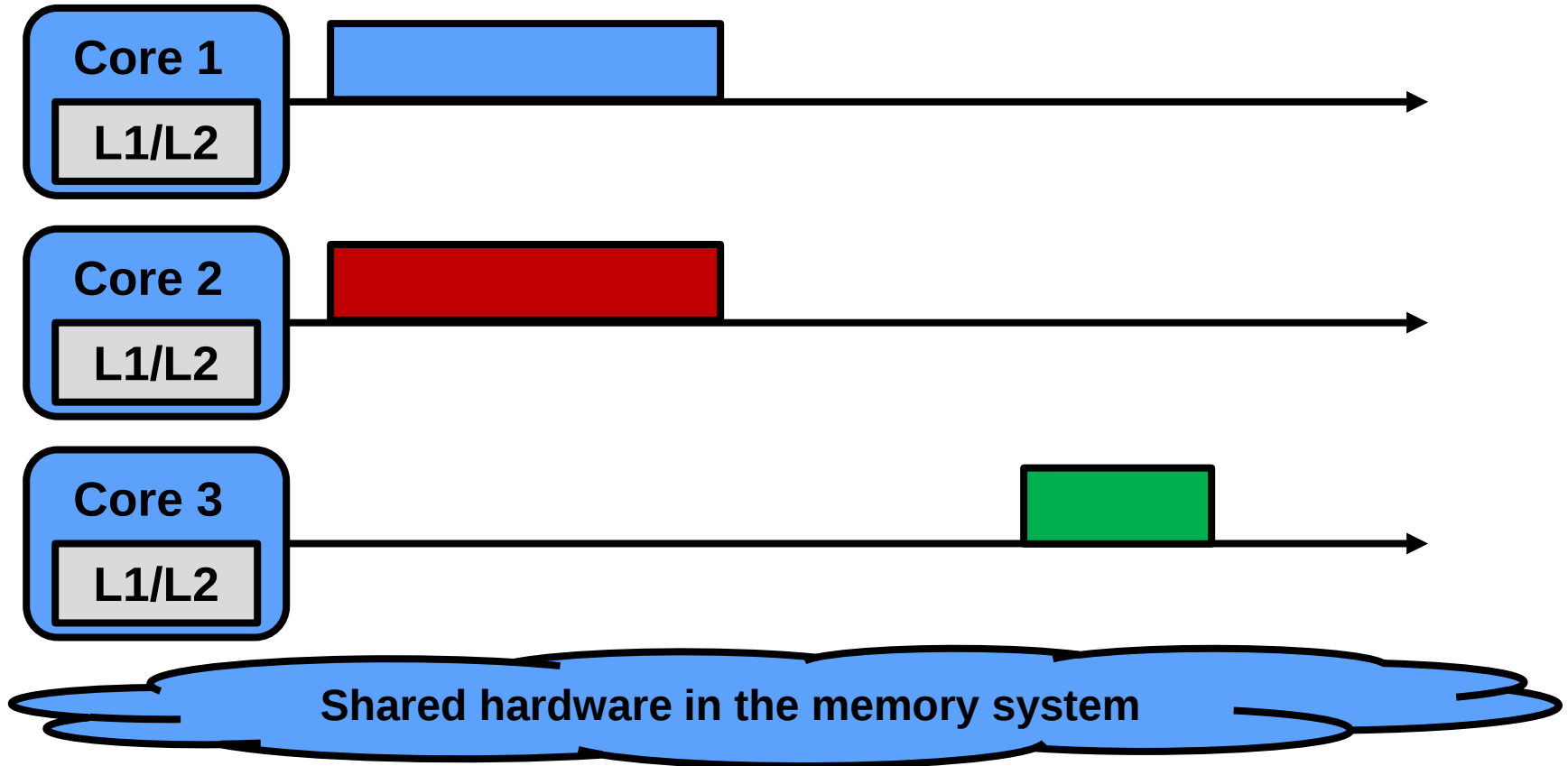


Solution 4: Black Box Analysis



The blue, red, and green tasks execute at different times $\Rightarrow$ no slowdown

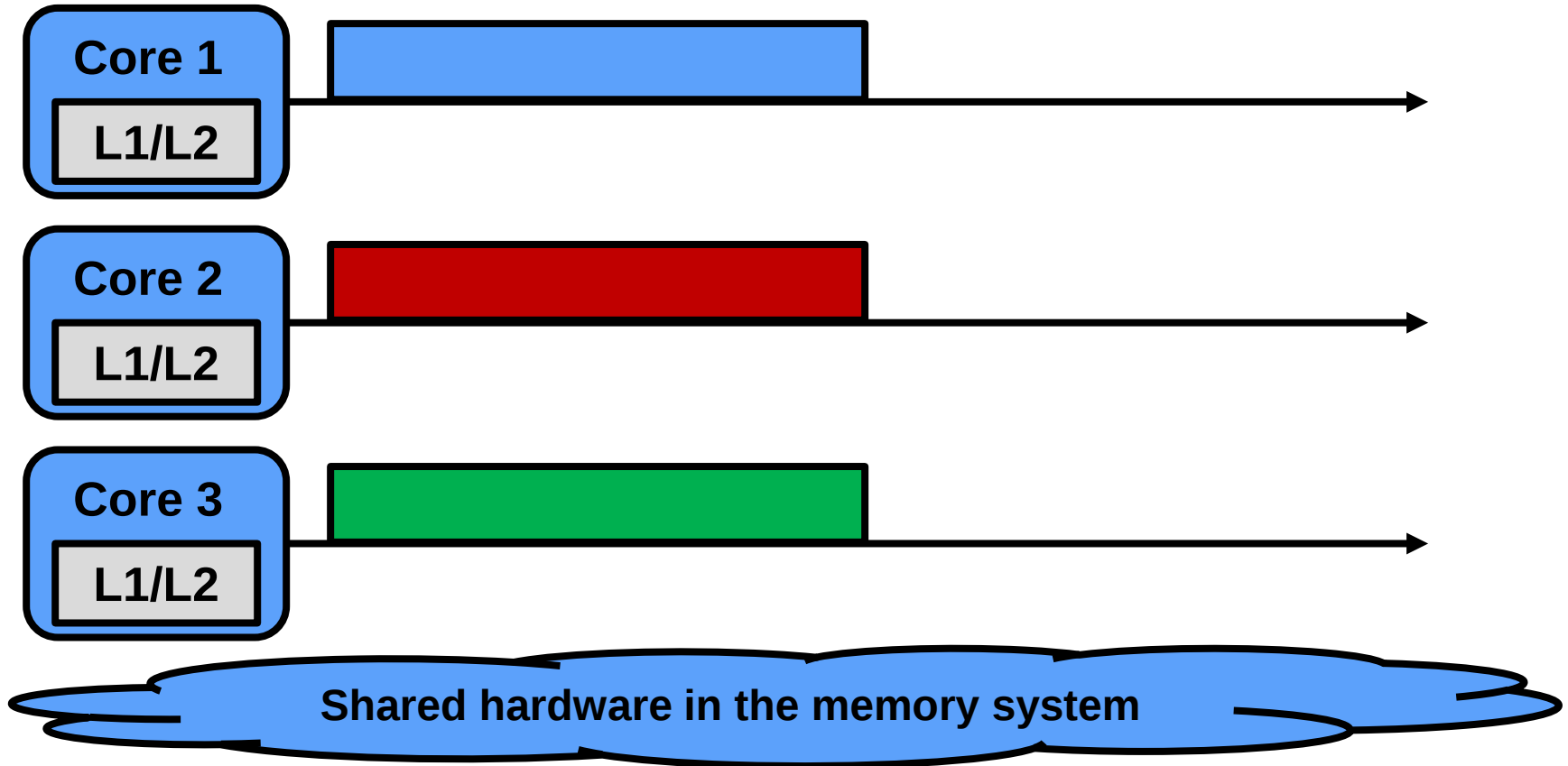
Solution 4: Black Box Analysis



The blue and red tasks execute at the same time $\Rightarrow$ slowdown $\Rightarrow$ increased execution time of blue and red.

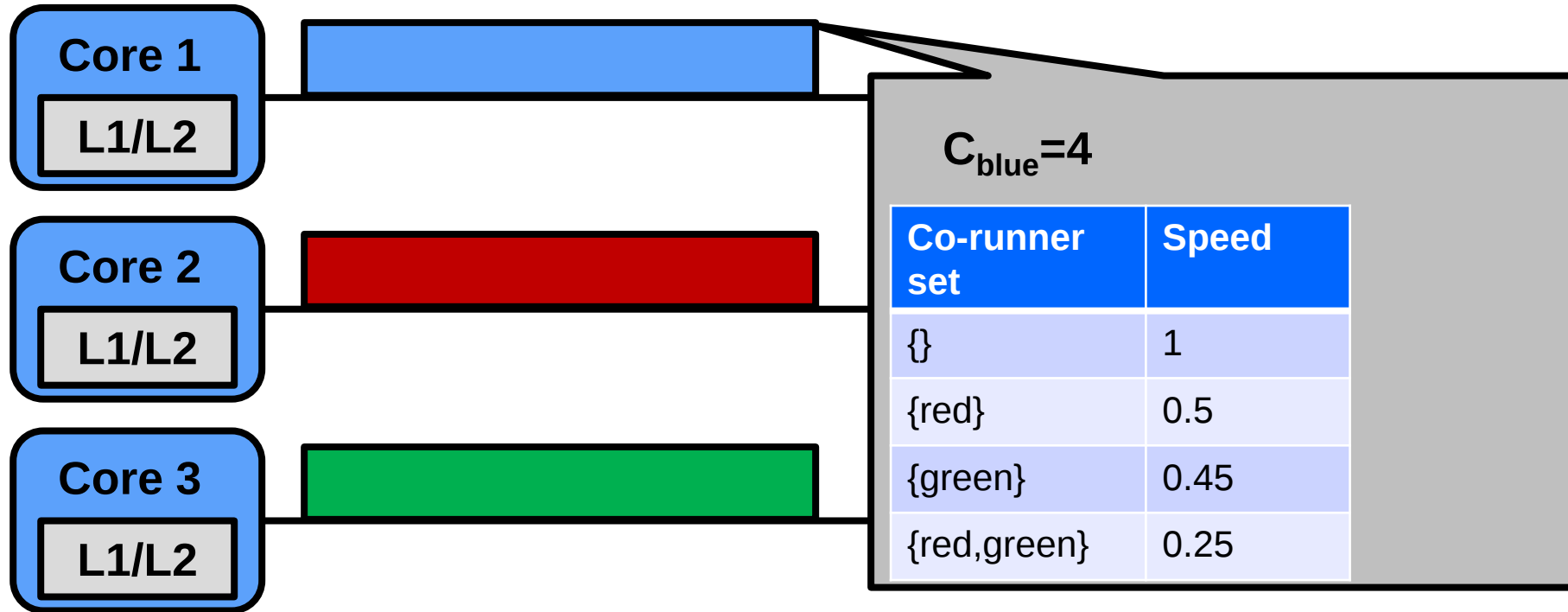


Solution 4: Black Box Analysis



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

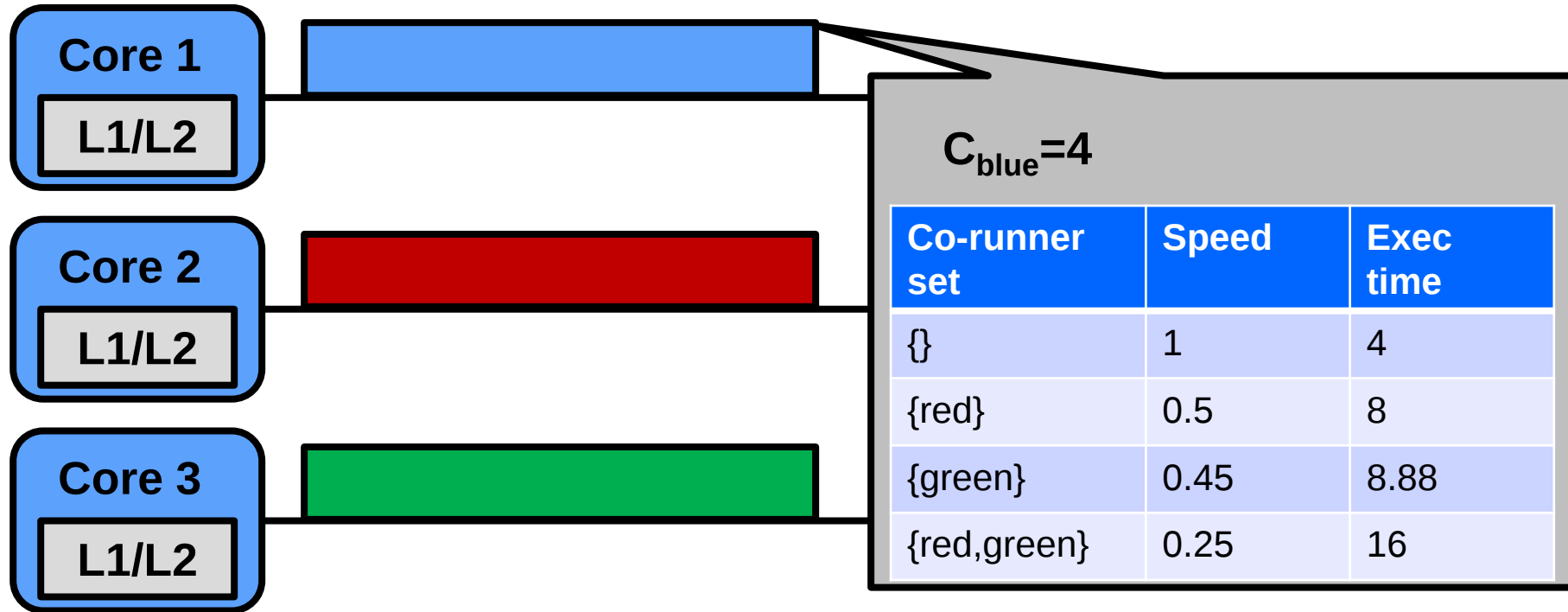
Solution 4: Black Box Analysis



Shared hardware in the memory system

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

Solution 4: Black Box Analysis

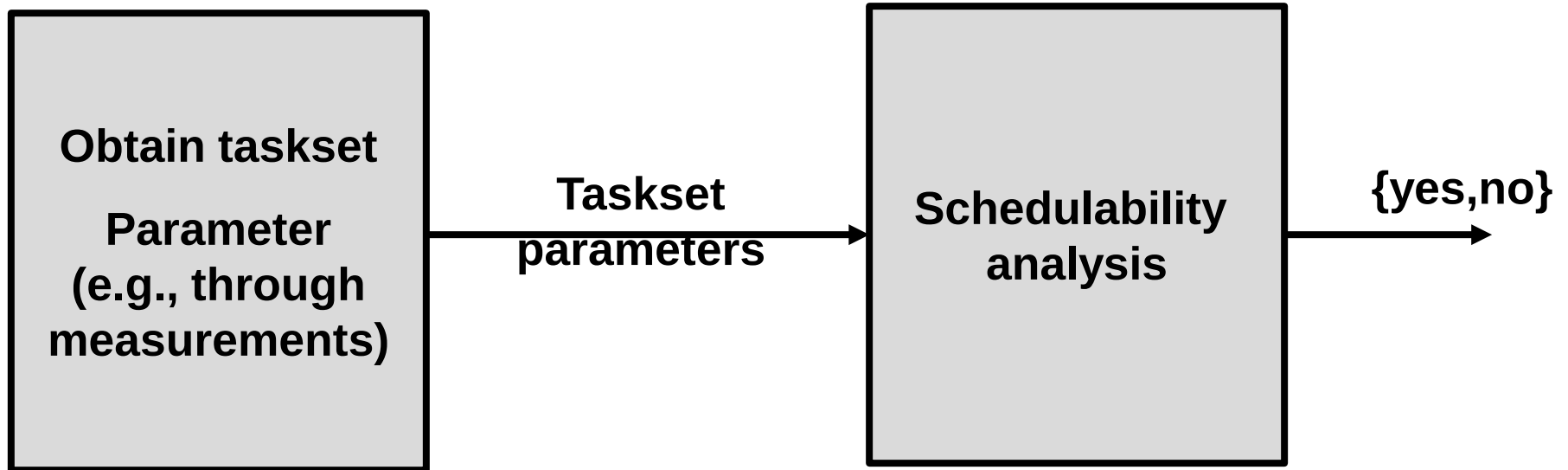


Shared hardware in the memory system

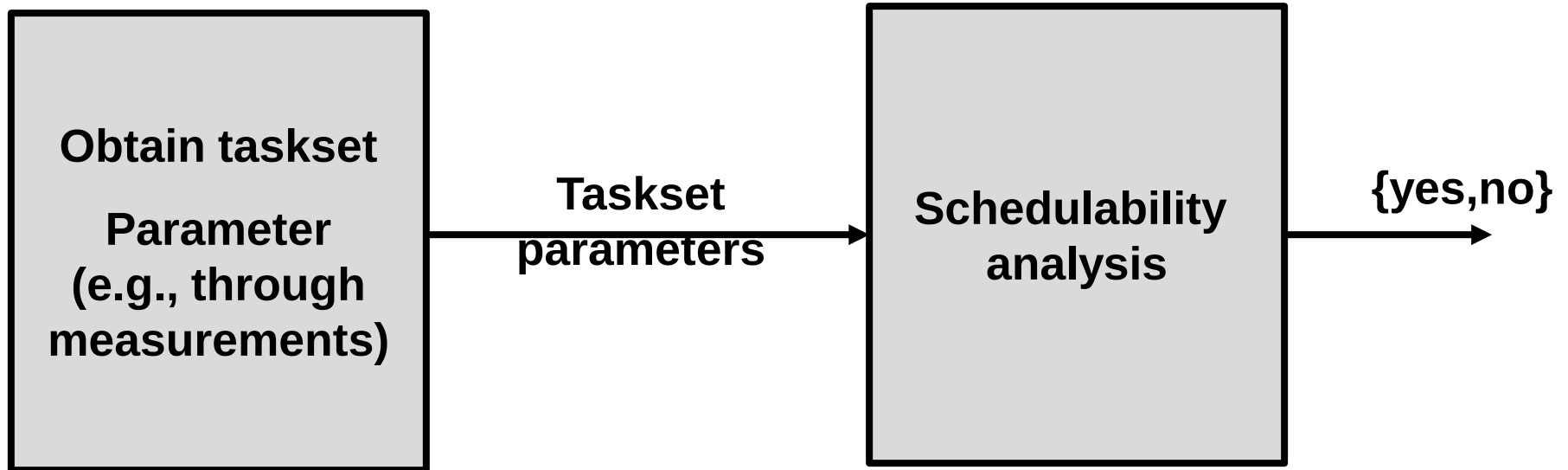
The blue, red, and green tasks execute at the same time $\Rightarrow$ slowdown $\Rightarrow$ increased execution time of all tasks.



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

