CPS Research

Dionisio de Niz and Sagar Chaki

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

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.

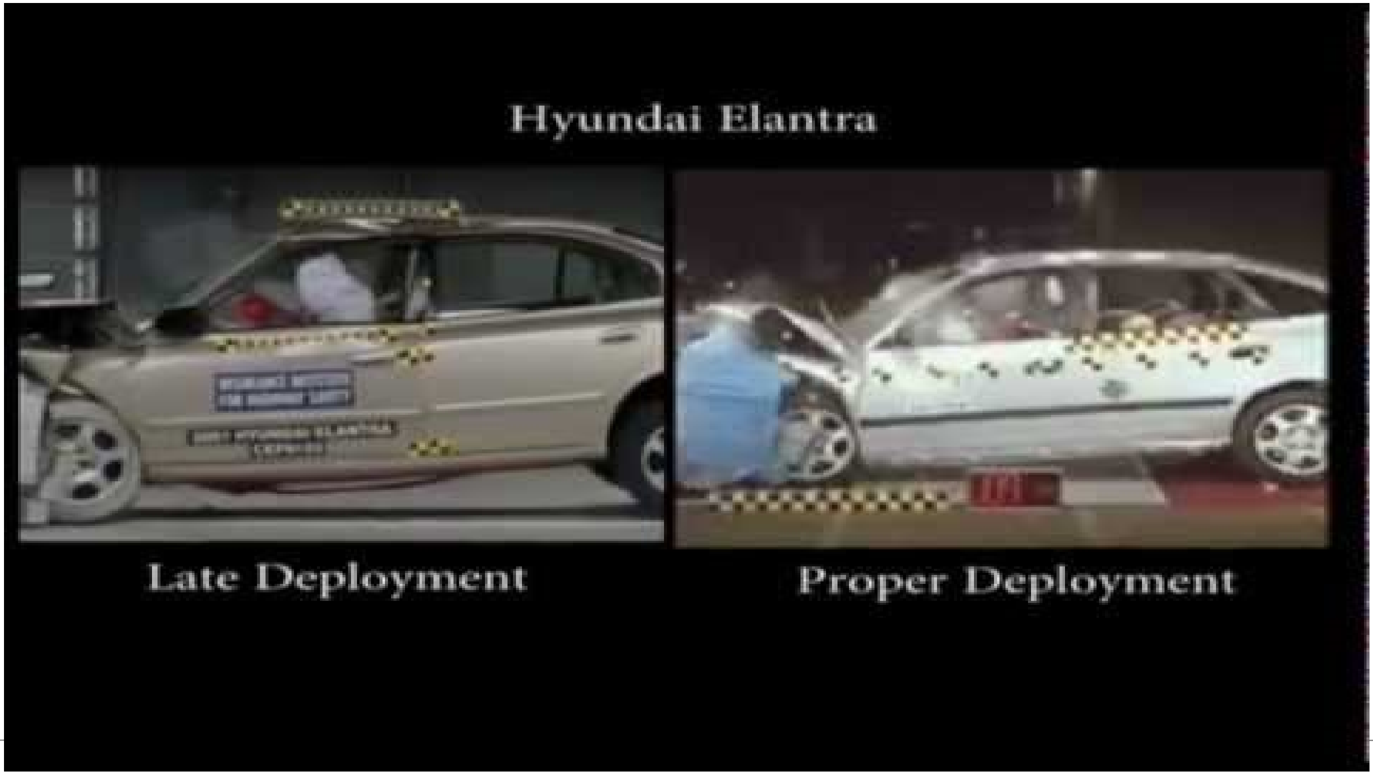
NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, 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.

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

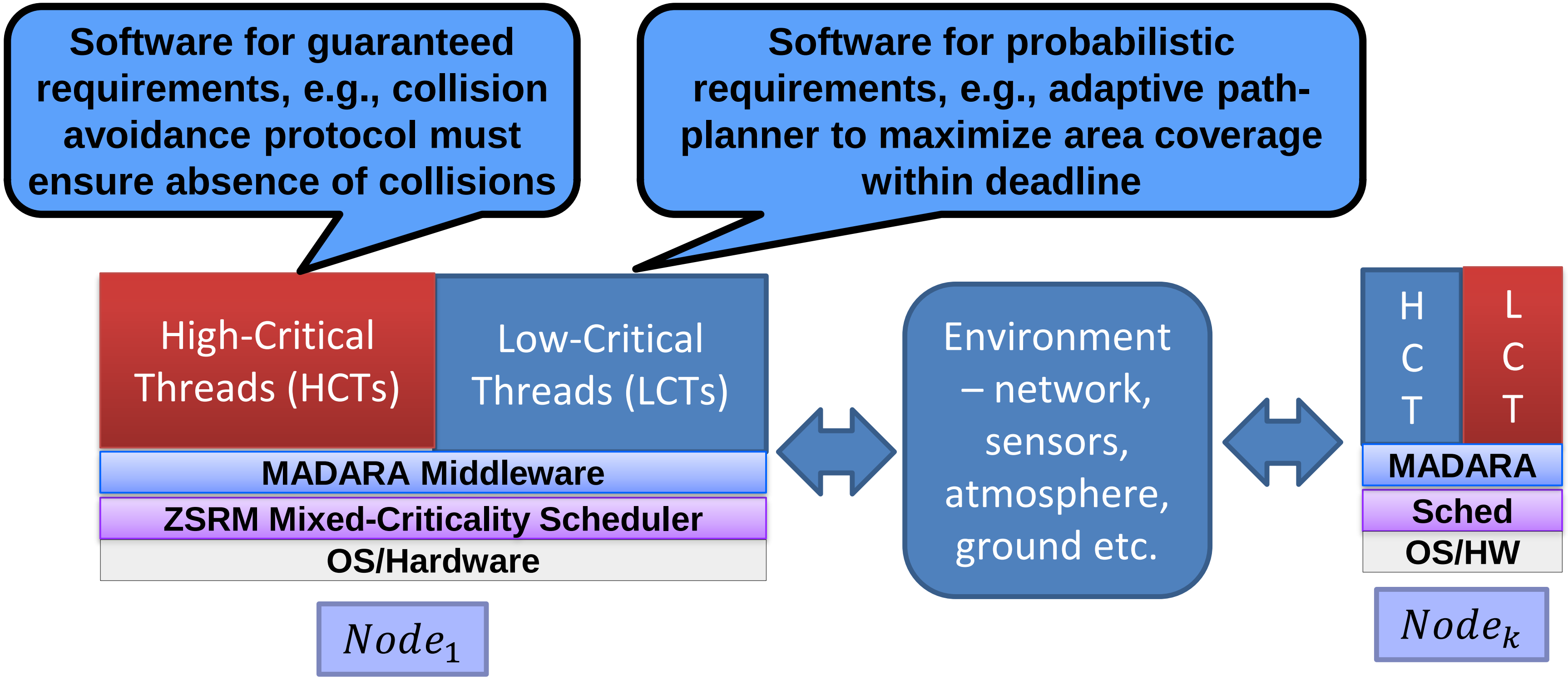
Scalable Combination

- 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

High-Critical Threads (HCTs)
Low-Critical Threads (LCTs)
MADARA Middleware
ZSRM Mixed-Criticality Scheduler
OS/Hardware
$Node_1$
Environment – network, sensors, atmosphere, ground etc.
HCT
LCT
MADARA
Sched
OS/HW
$Node_k$

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

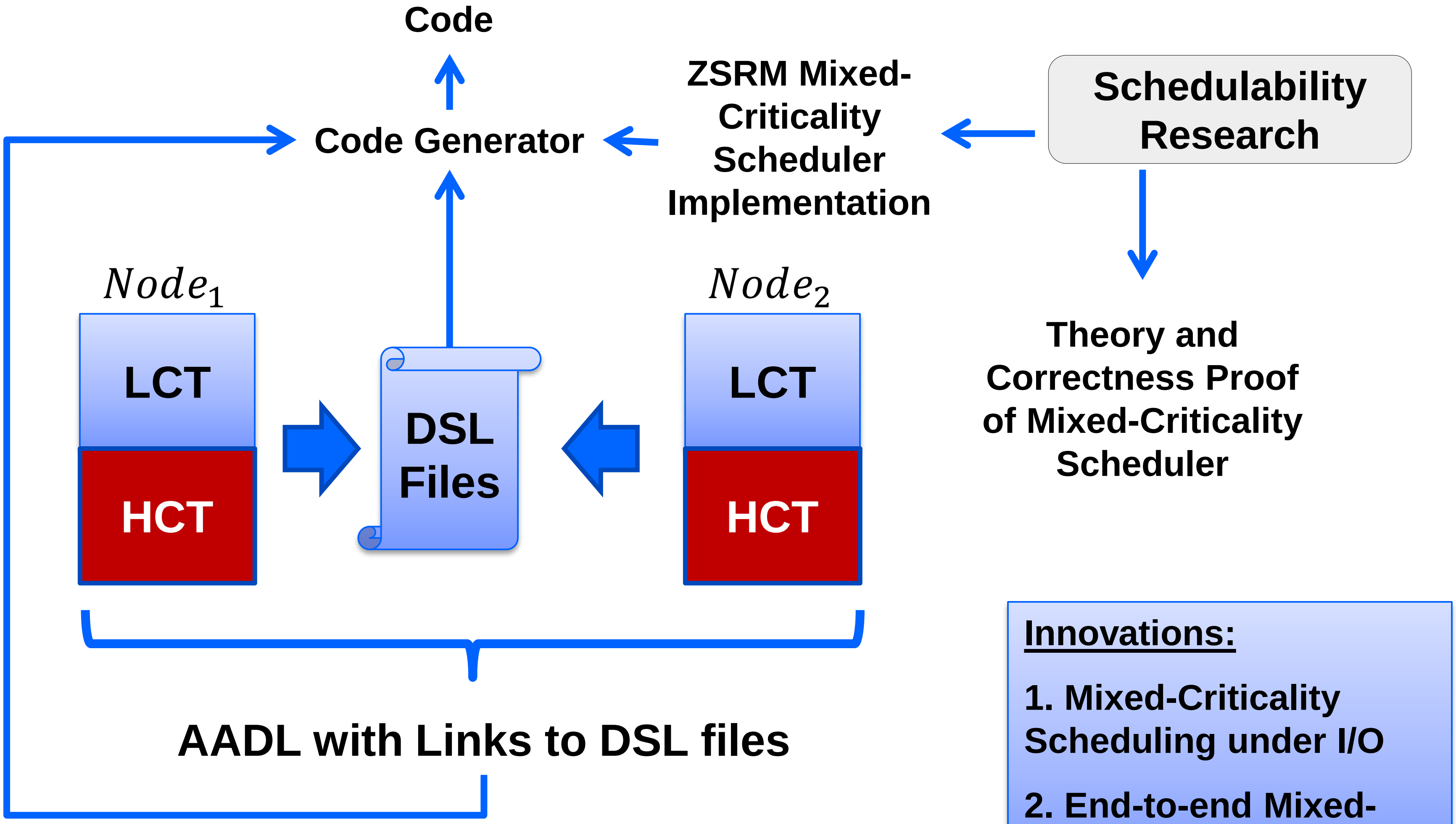
- Perform ZSRM schedulability via OSATE Plugin
- Generate appropriate DMPL annotations

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



Code

Code Generator

ZSRM Mixed-Criticality Scheduler Implementation

Schedulability Research

$Node_1$

LCT

HCT

DSL Files

$Node_2$

LCT

HCT

Theory and Correctness Proof of Mixed-Criticality Scheduler

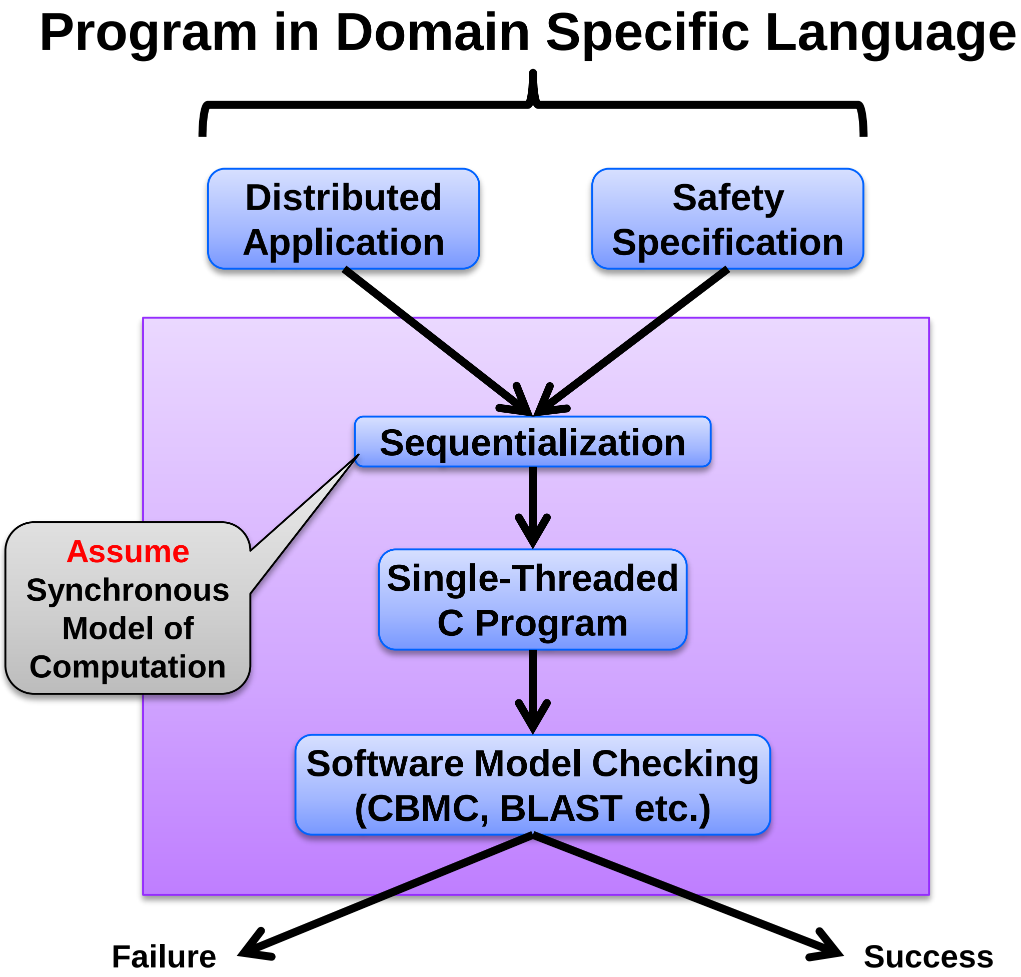
AADL with Links to DSL files

Innovations:

1. Mixed-Criticality Scheduling under I/O

2. End-to-end Mixed-Criticality Scheduling

Verification


Program in Domain Specific Language
Distributed Application
Safety Specification
Sequentialization
Single-Threaded C Program
Software Model Checking (CBMC, BLAST etc.)
Assume Synchronous Model of Computation
Failure
Success

Model Checking

Automatic verification technique for finite state concurrent systems.

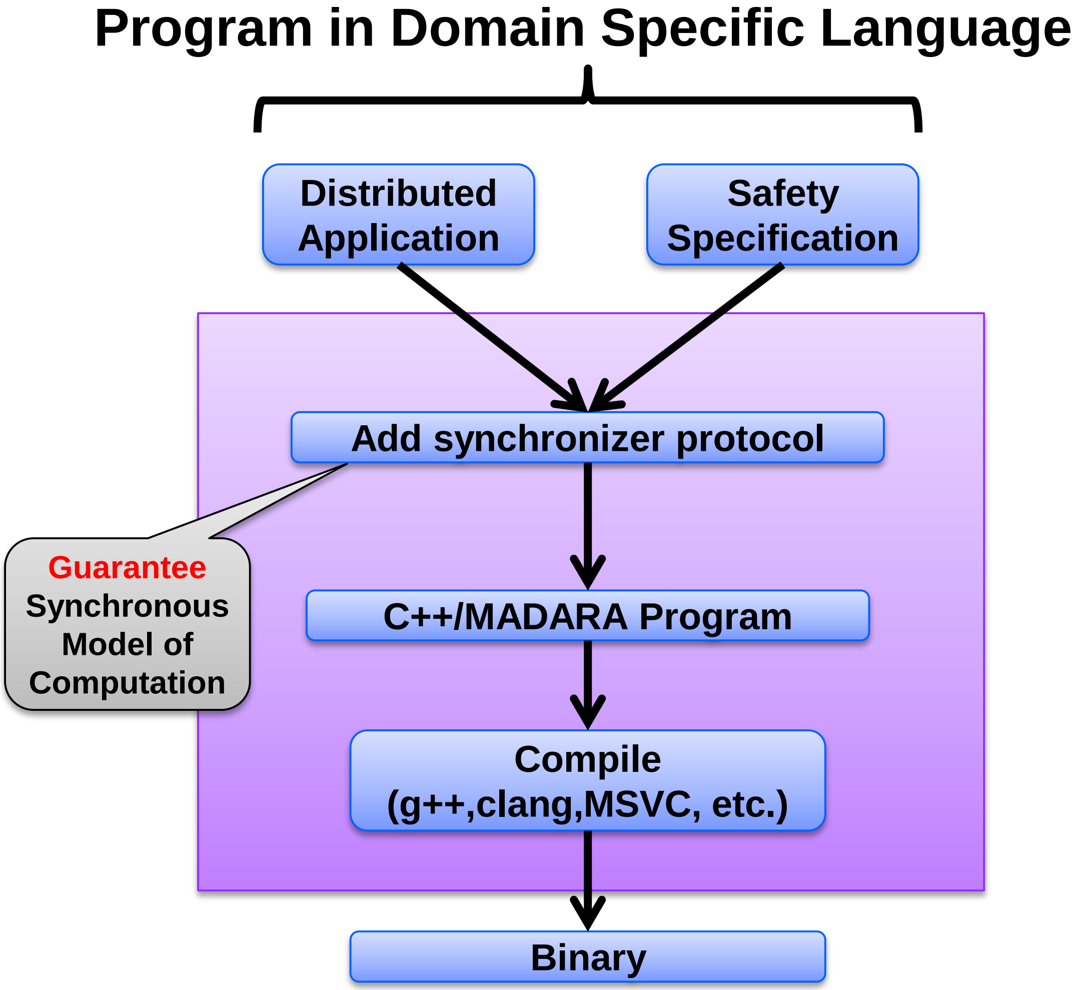
- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- ACM Turing Award 2007

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
Distributed Application
Safety Specification
Add synchronizer protocol
Guarantee Synchronous Model of Computation
C++/MADARA Program
Compile (g++,clang,MSVC, etc.)
Binary

MADARA Middleware

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

Node i has a local copy: DB_i

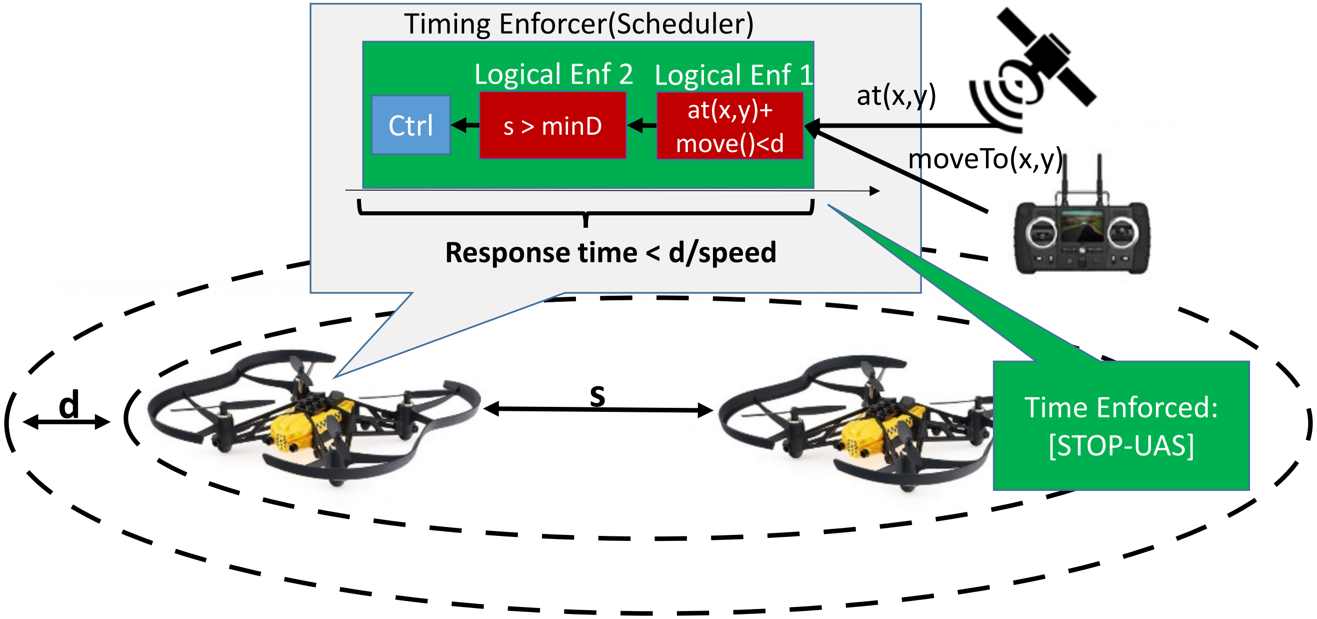
- update DB_i arbitrarily
- publish new variable mappings
  - Immediate or delayed
  - Multiple variable mappings transmitted atomically

Implicit “receive” thread on each node

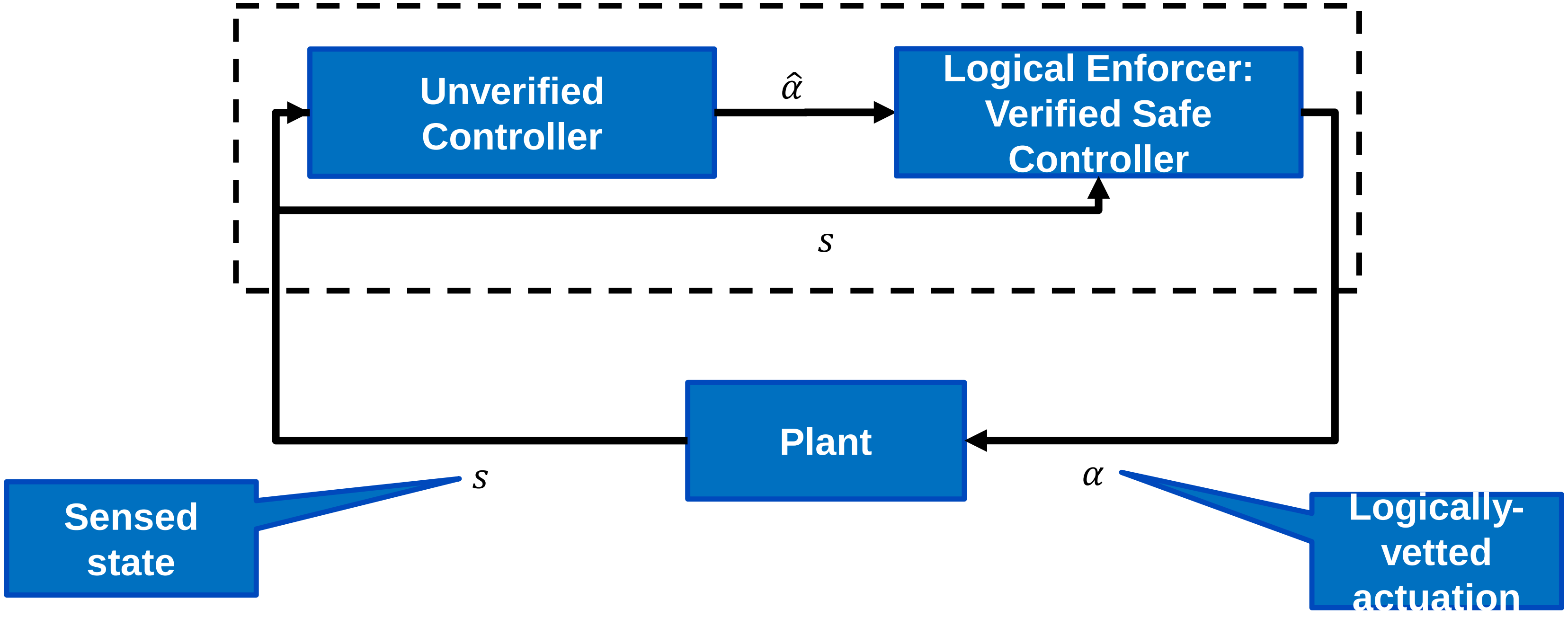
- Receives and processes variable updates from other nodes
- Updates ordered via Lamport clocks

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

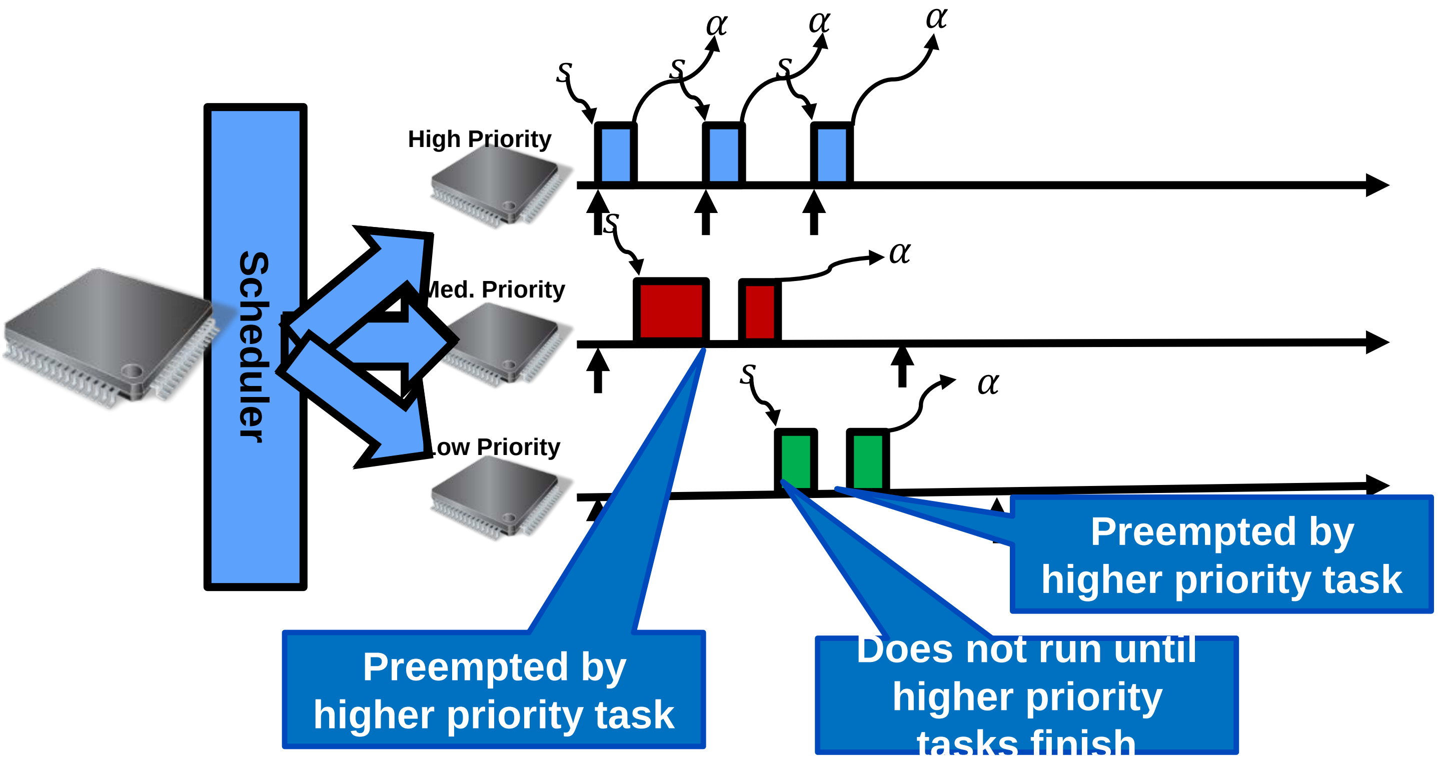
Project 2: Certifiable Distributed Runtime Assurance



Sense Actuation Loop + Logical Enforcer


Unverified Controller
$\hat{\alpha}$
Logical Enforcer: Verified Safe Controller
s
Plant
s
α
Sensed state
Logically-vetted actuation

Fixed-Priority Scheduling + Rate Monotonic



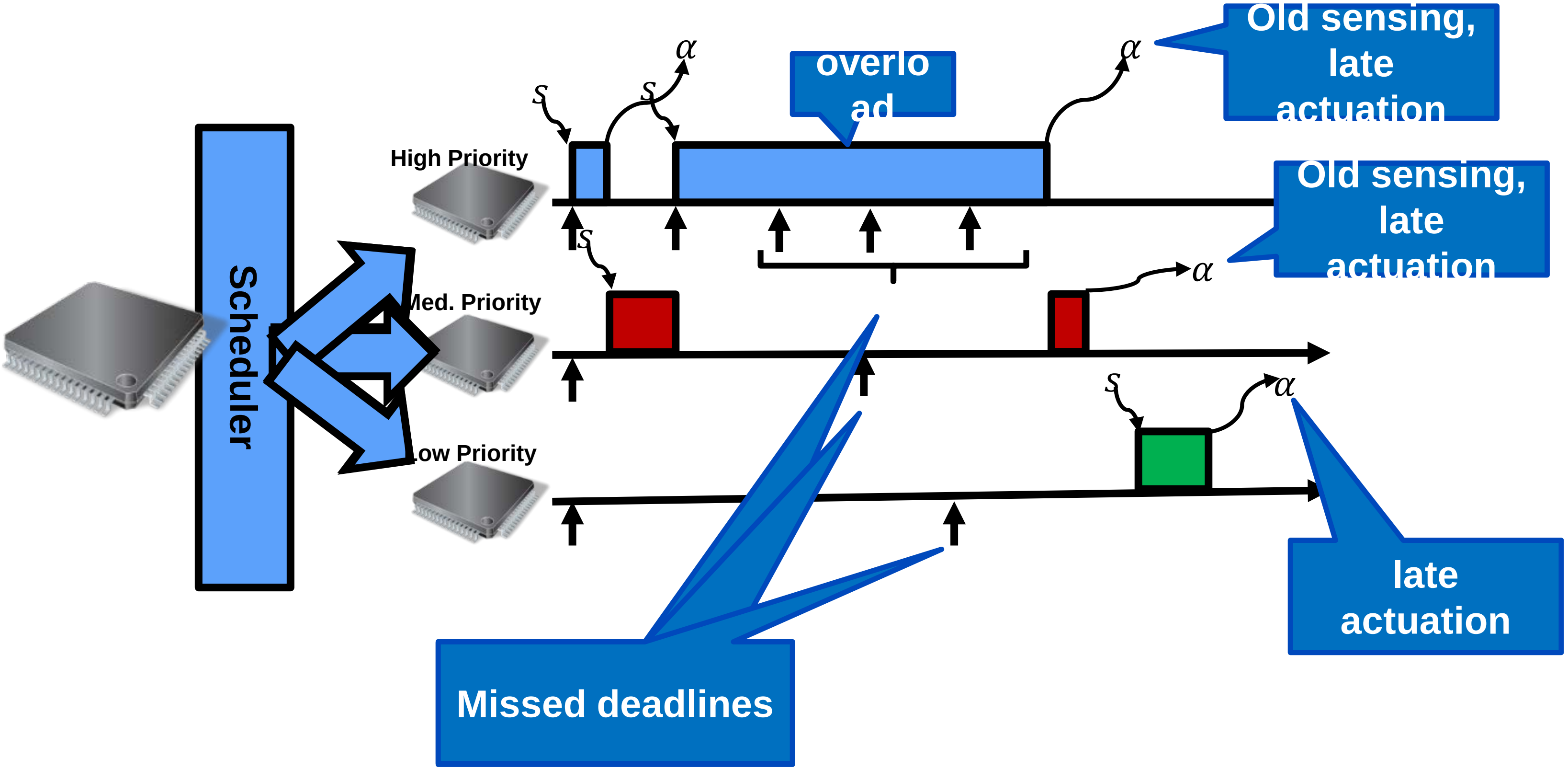
Preempted by higher priority task

Does not run until higher priority tasks finish

Preempted by higher priority task

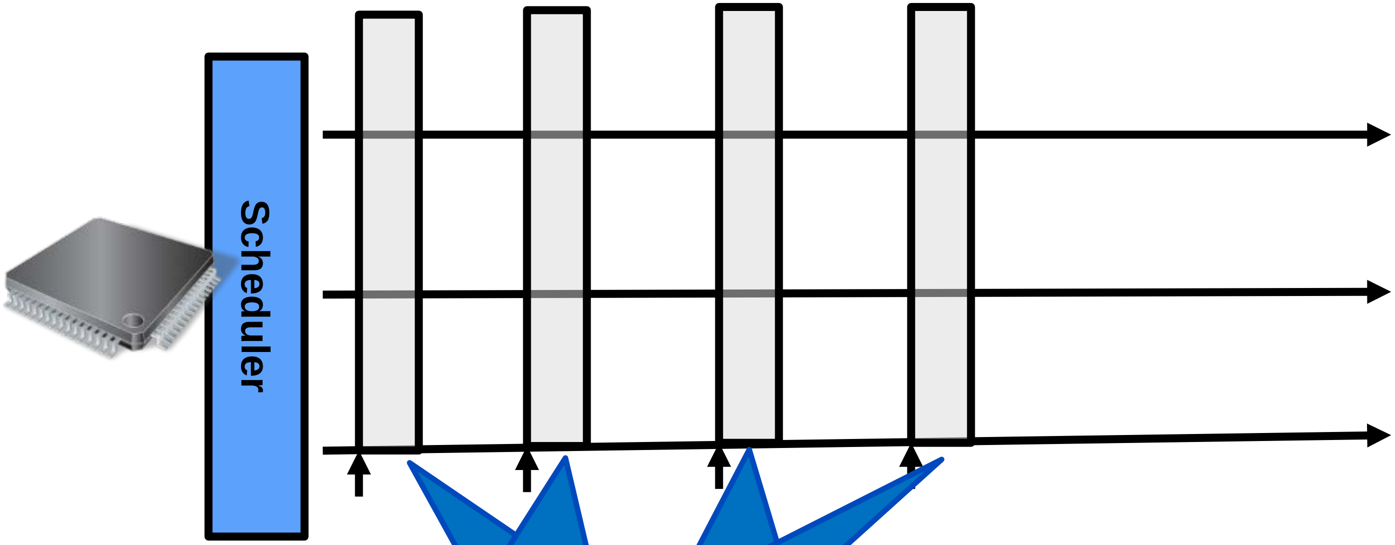
Icons credit: http://www.doublejdesign.co.uk

Overload => old sensed data + late actuation



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

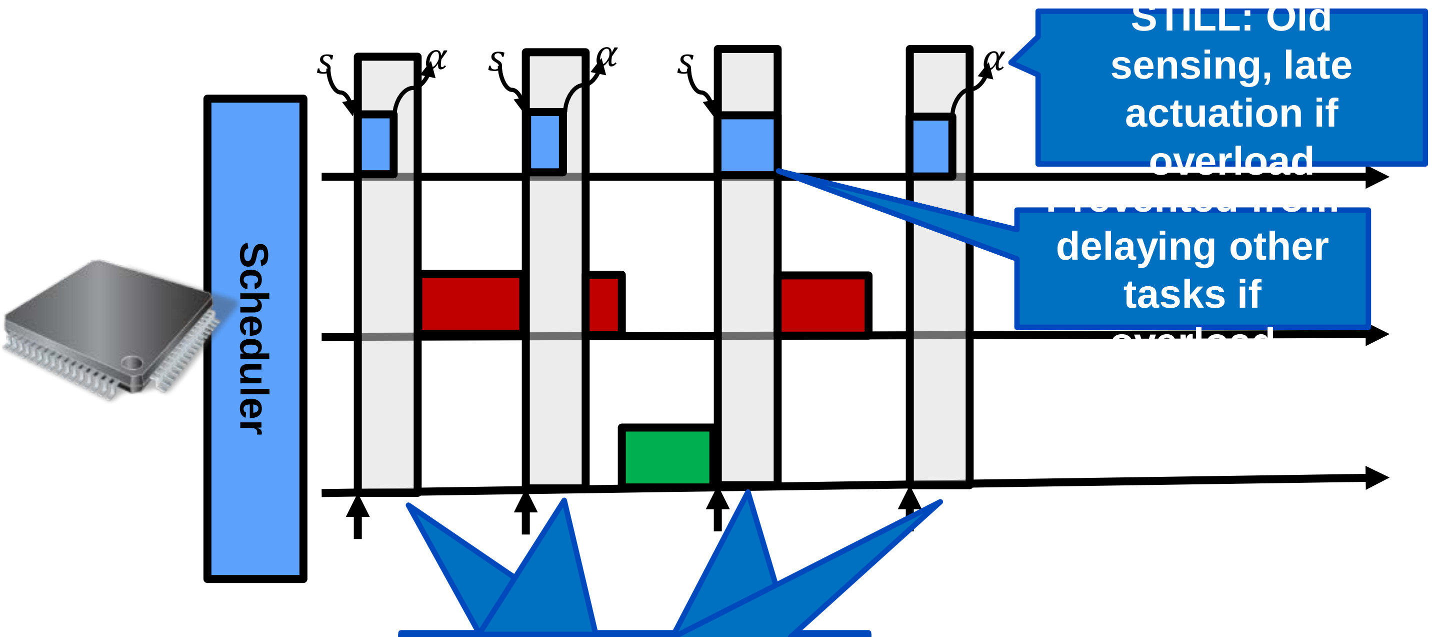
Solution: Enforce timing budgets (timing enforcement)



Only executed in given periodic time budget

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

Solution step 1: enforce timing budgets (timing enforcement)



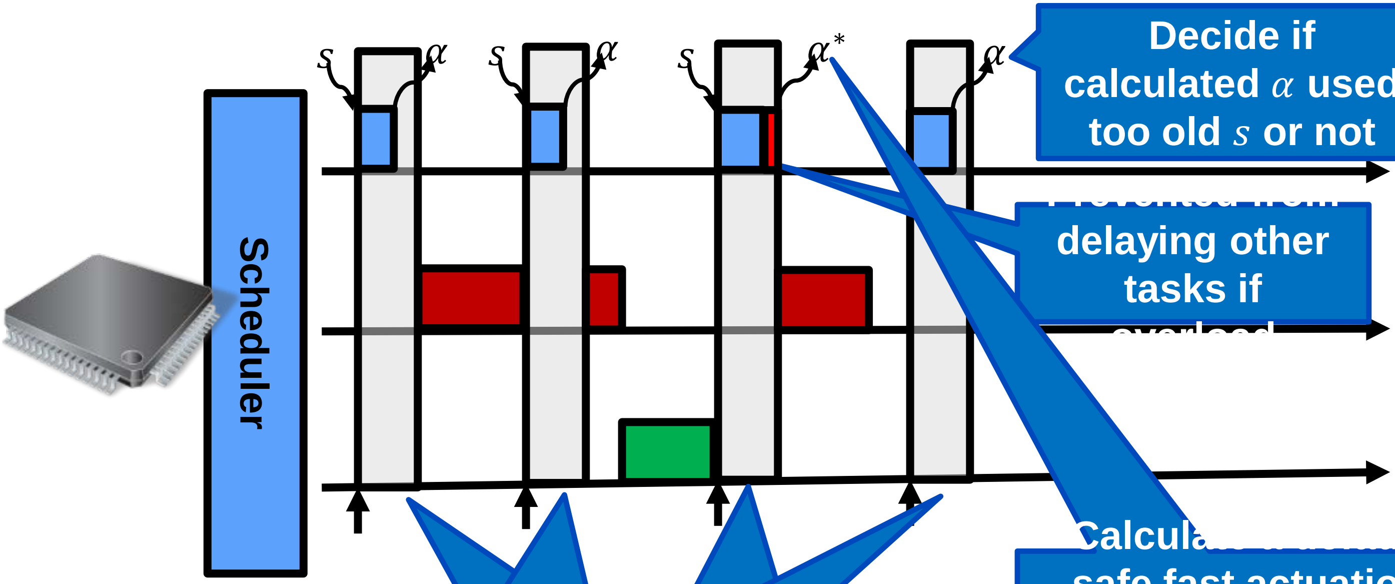
STILL: Old sensing, late actuation if overload

Prevented from delaying other tasks if overload

Only executed in given periodic time budget

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

Solution step 2: fast actuation on timing enforcement



Decide if calculated α used too old s or not

Prevented from delaying other tasks if overload

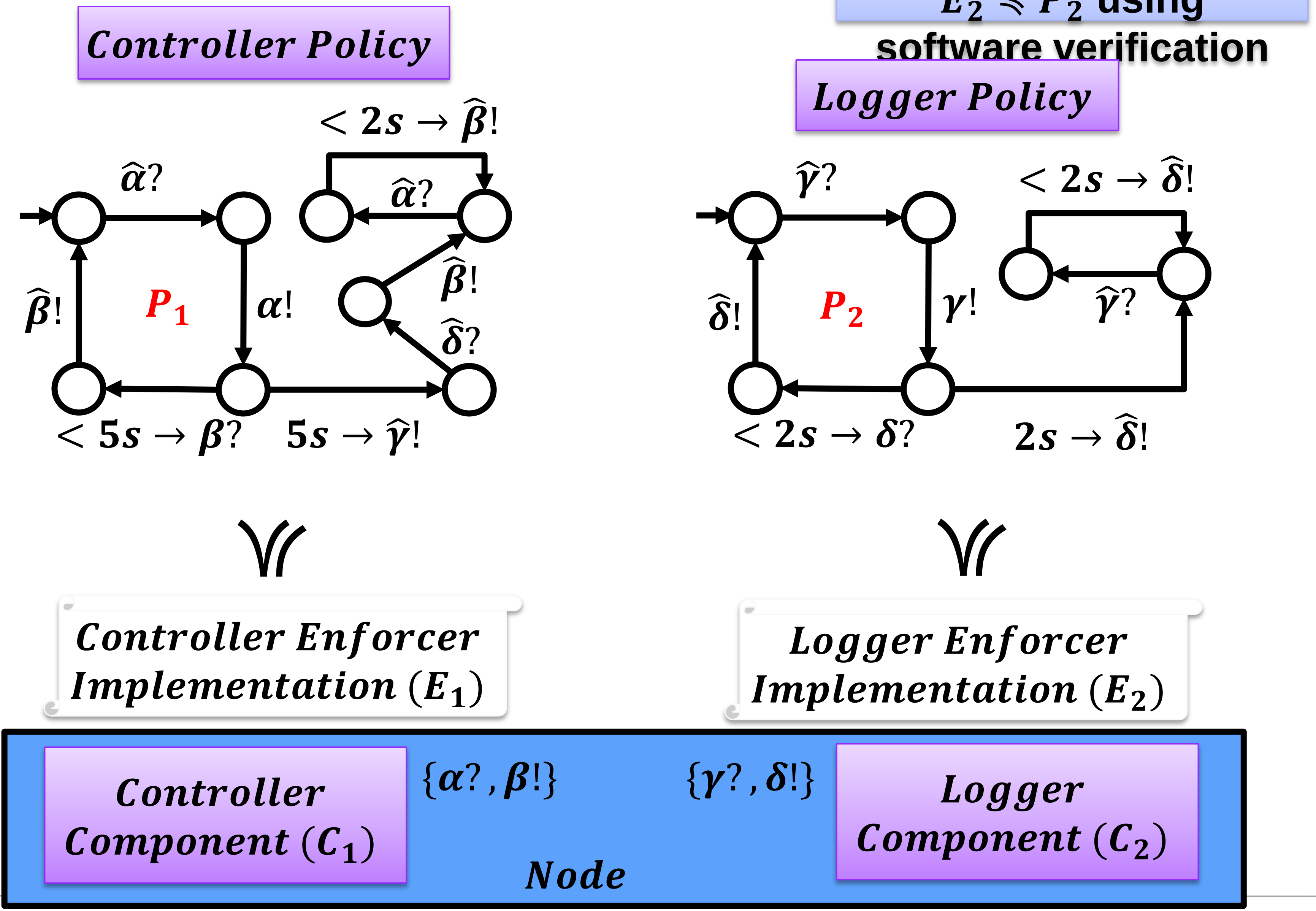
Calculate a default safe fast actuation executed “just before” timing budget expires: kernel

Only executed in given periodic time budget

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



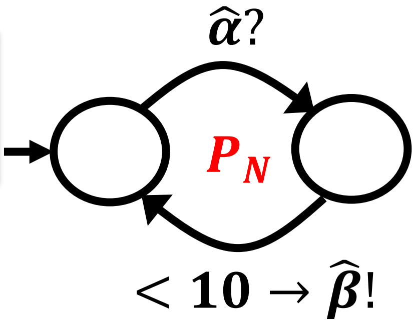
Controller Policy
$< 2s \rightarrow \hat{\beta}!$
$\hat{\alpha}?$
$\hat{\alpha}?$
$\hat{\beta}!$
P_1
$\alpha!$
$\hat{\beta}!$
$\hat{\delta}?$
$< 5s \rightarrow \beta?$
$5s \rightarrow \hat{\gamma}!$
Logger Policy
$\hat{\gamma}?$
$< 2s \rightarrow \hat{\delta}!$
$\hat{\delta}!$
P_2
$\gamma!$
$\hat{\gamma}?$
$< 2s \rightarrow \delta?$
$2s \rightarrow \hat{\delta}!$
$\succcurlyeq$
$\succcurlyeq$
Controller Enforcer Implementation (E_1)
Logger Enforcer Implementation (E_2)
Controller Component (C_1)
$\{\alpha?, \beta!\}$
$\{\gamma?, \delta!\}$
Logger Component (C_2)
Node

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

CDRA: Approach (2)

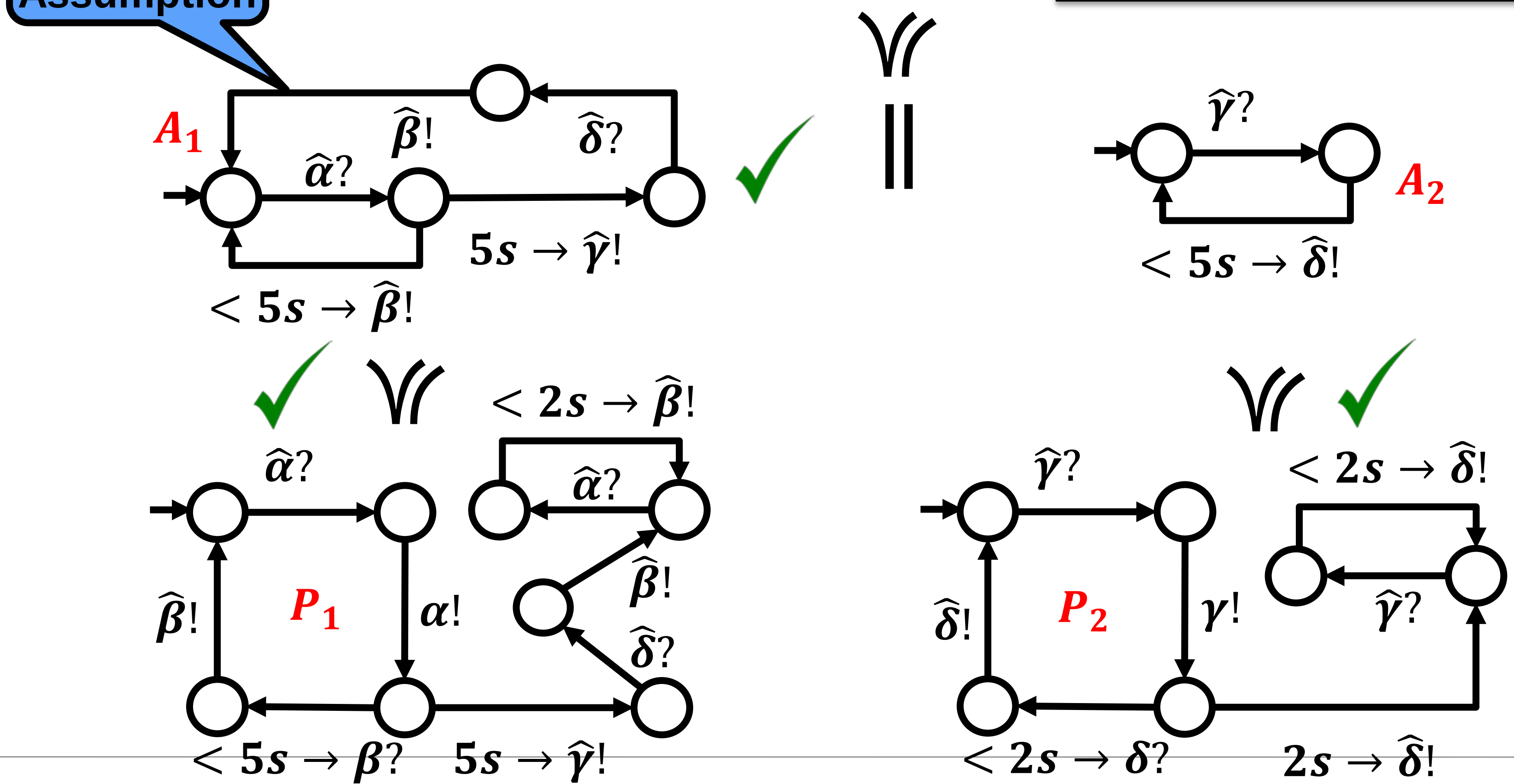
$$\frac{P_1 \preccurlyeq A_1 \quad P_2 \preccurlyeq A_2 \quad A_1 \parallel A_2 \preccurlyeq P_N}{P_1 \parallel P_2 \preccurlyeq P_N}$$

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.

Controller Assumption

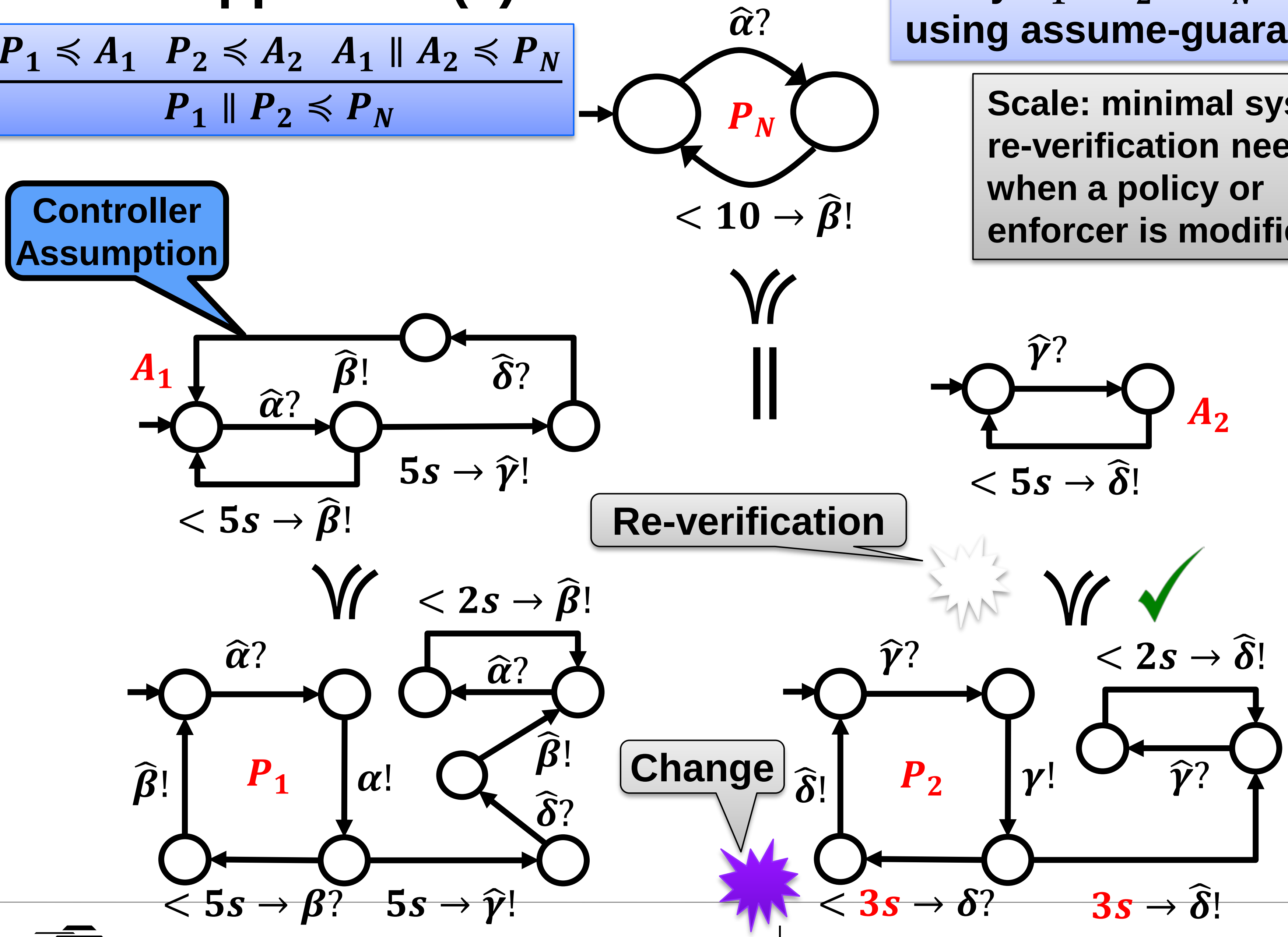


CDRA: Approach (3)

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

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

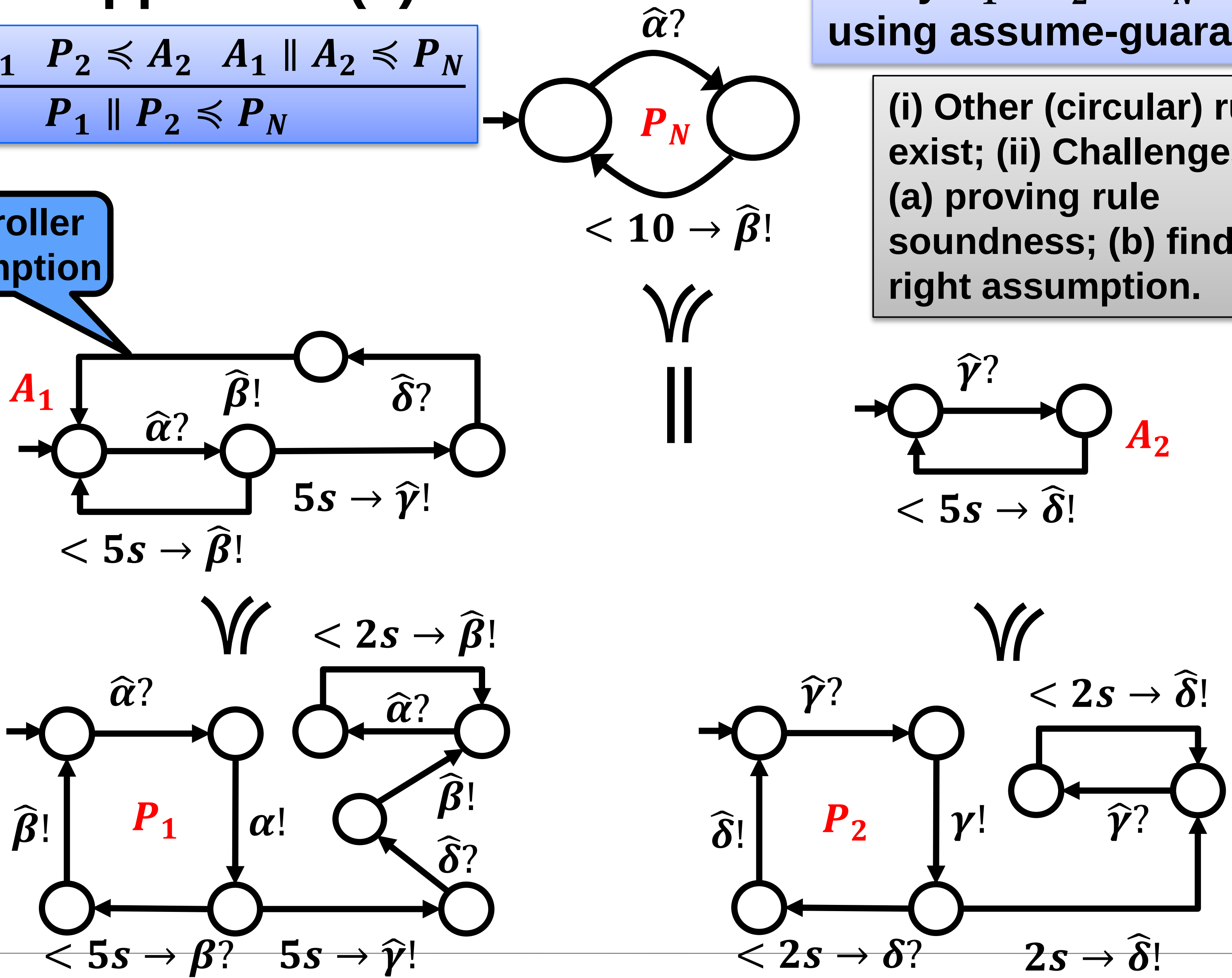


CDRA: Approach (4)

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

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.


$\hat{\alpha}?$
P_N
$< 10 \rightarrow \hat{\beta}!$
$\succcurlyeq$
$\parallel$
A_1
$\hat{\beta}!$
$\hat{\delta}?$
$\hat{\alpha}?$
$5s \rightarrow \hat{\gamma}!$
$< 5s \rightarrow \hat{\beta}!$
$\hat{\gamma}?$
A_2
$< 5s \rightarrow \hat{\delta}!$
$\succcurlyeq$
$< 2s \rightarrow \hat{\beta}!$
$\hat{\alpha}?$
$\hat{\alpha}?$
$\hat{\beta}!$
P_1
$\alpha!$
$\hat{\beta}!$
$\hat{\delta}?$
$< 5s \rightarrow \beta?$
$5s \rightarrow \hat{\gamma}!$
$\succcurlyeq$
$\hat{\gamma}?$
$< 2s \rightarrow \hat{\delta}!$
$\hat{\delta}!$
P_2
$\gamma!$
$\hat{\gamma}?$
$< 2s \rightarrow \delta?$
$2s \rightarrow \hat{\delta}!$

Controller Assumption

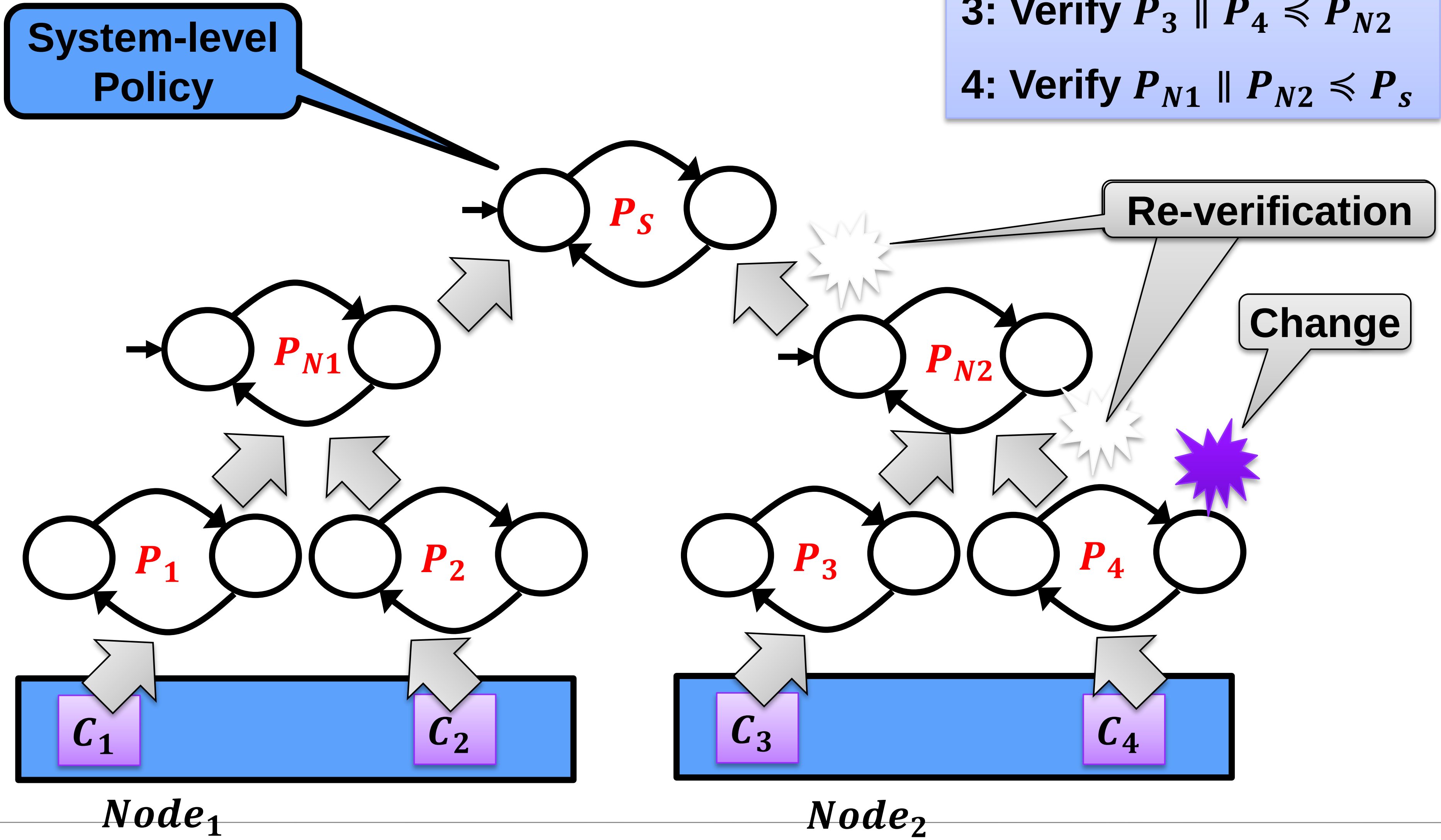
CDRA: Approach (5)

1: Verify $E_i \preccurlyeq P_i$

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

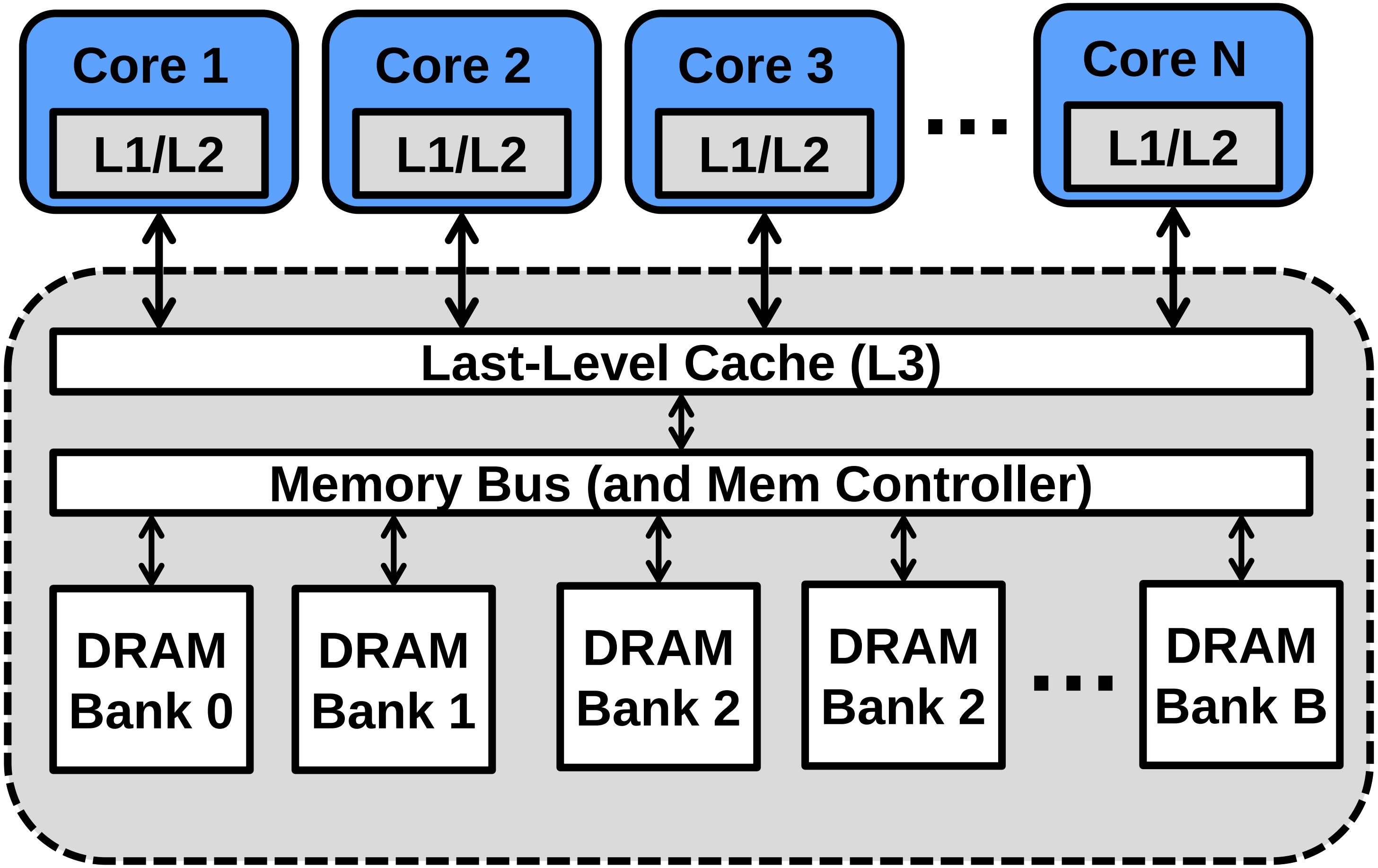
3: Verify $P_3 \parallel P_4 \preccurlyeq P_{N2}$

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

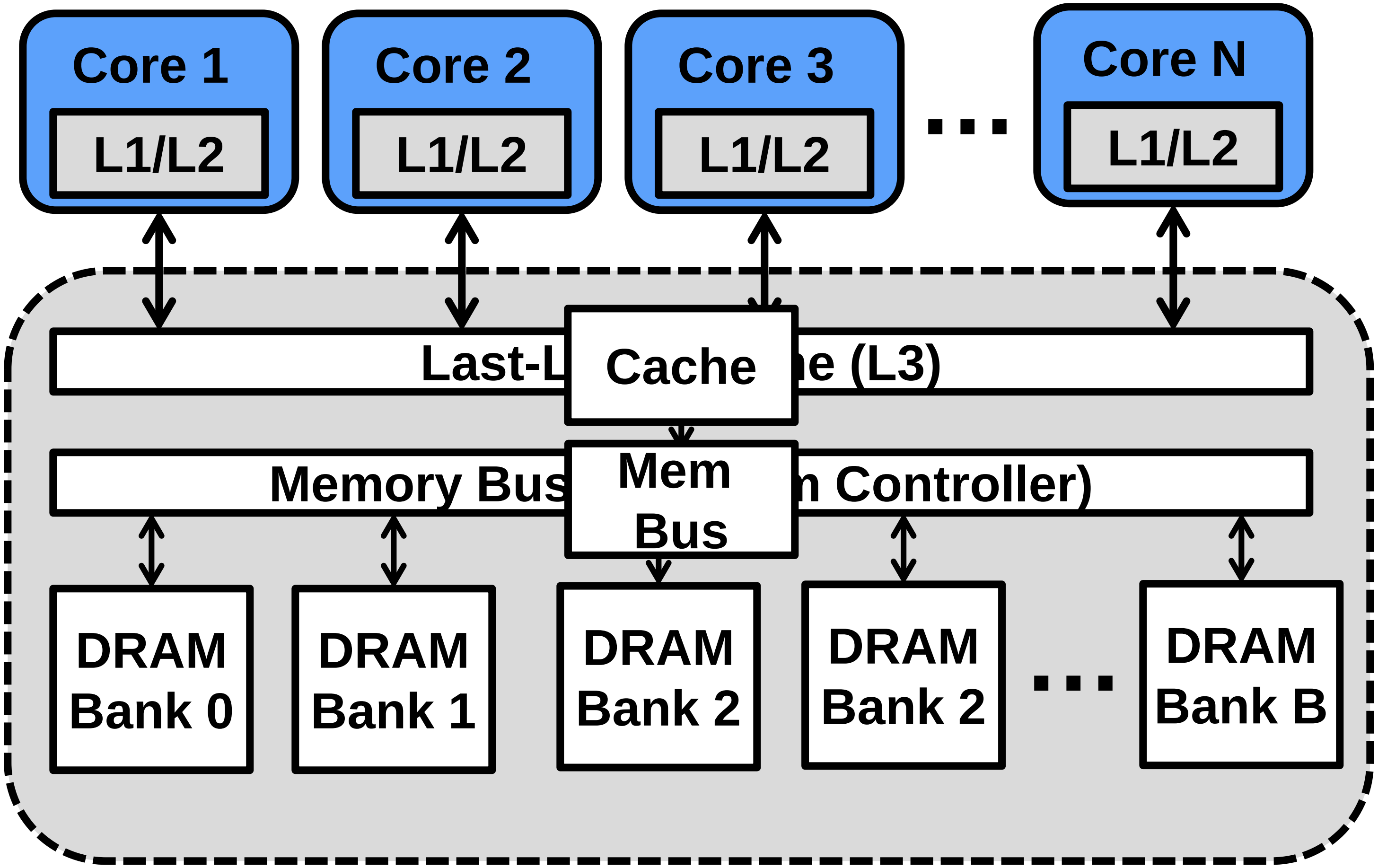


[Distribution Statement A] Approved for public release and unlimited distribution

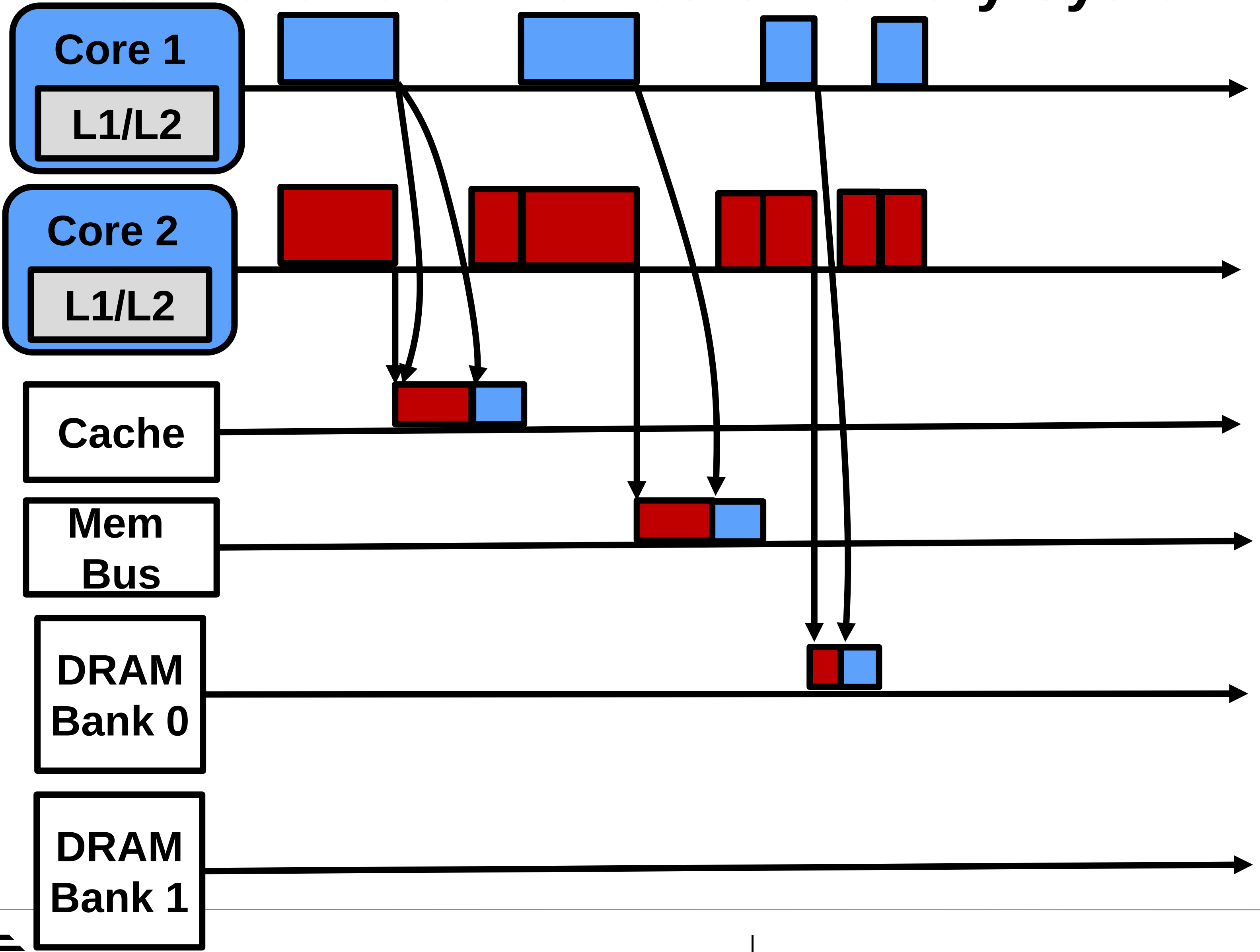
Project 3: Real-Time Scheduling for Multicore



Shared Hardware: Multicore Memory System

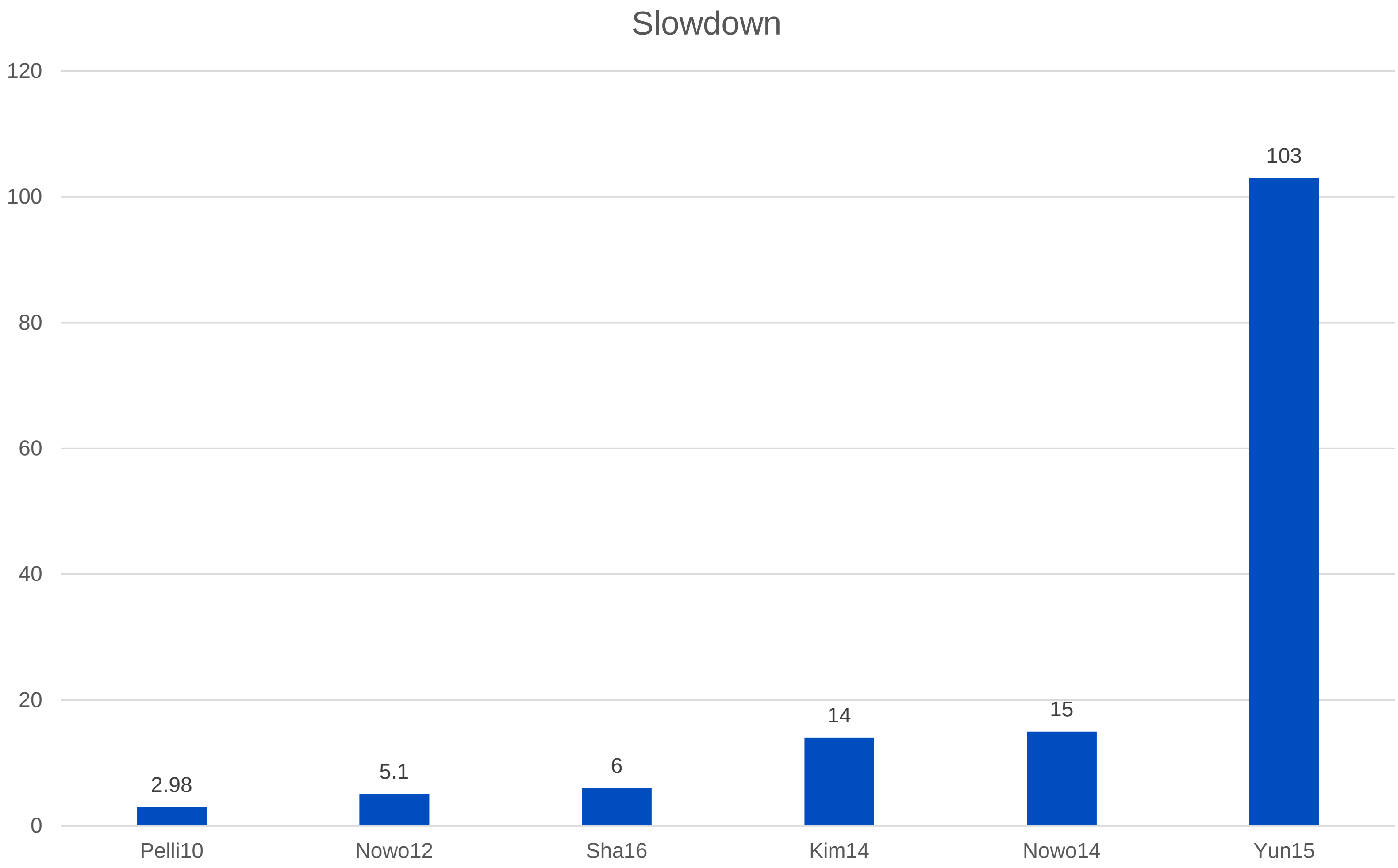


Shared Hardware: Multicore Memory System



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

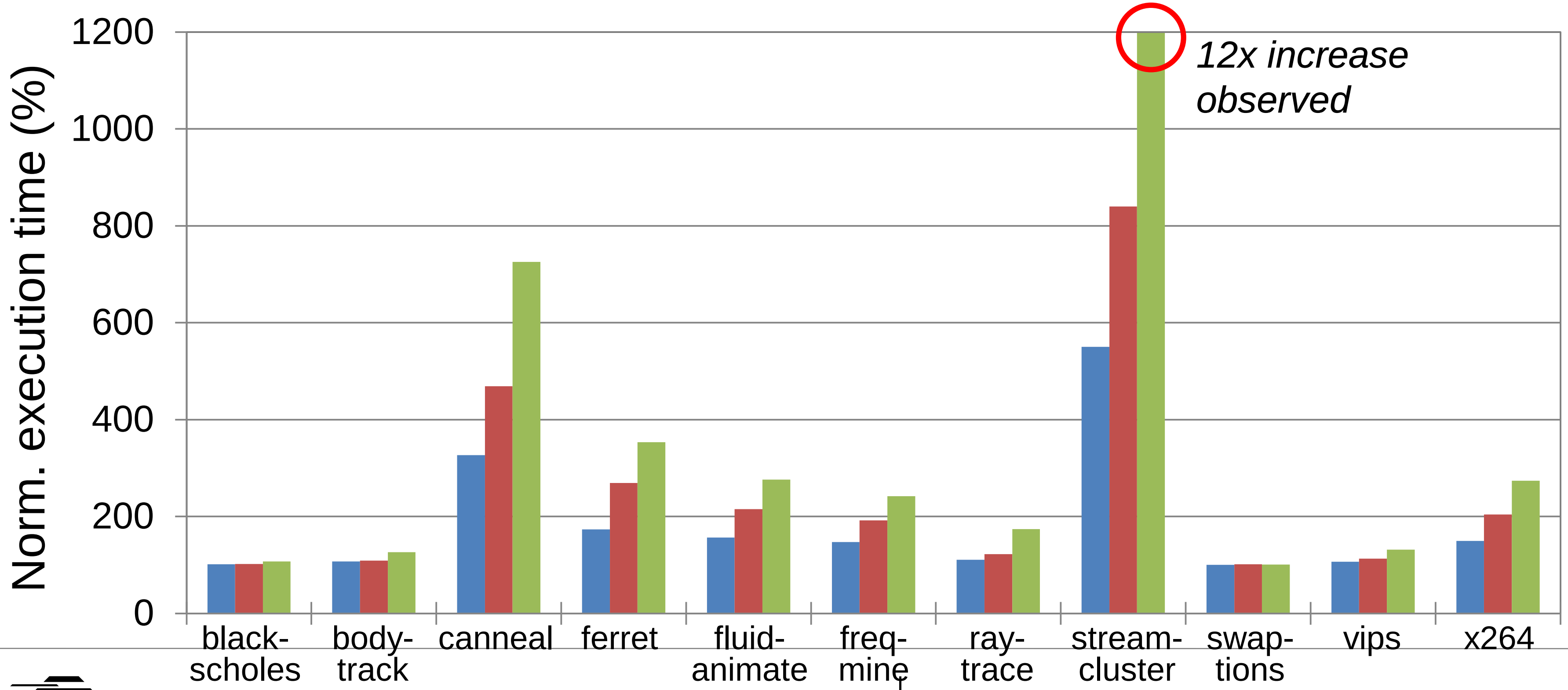
How Bad?


Slowdown
120
100
80
60
40
20
0
2.98
5.1
6
14
15
103
Pelli10
Nowo12
Sha16
Kim14
Nowo14
Yun15

Different for Applications (PARSEC Benchmark)

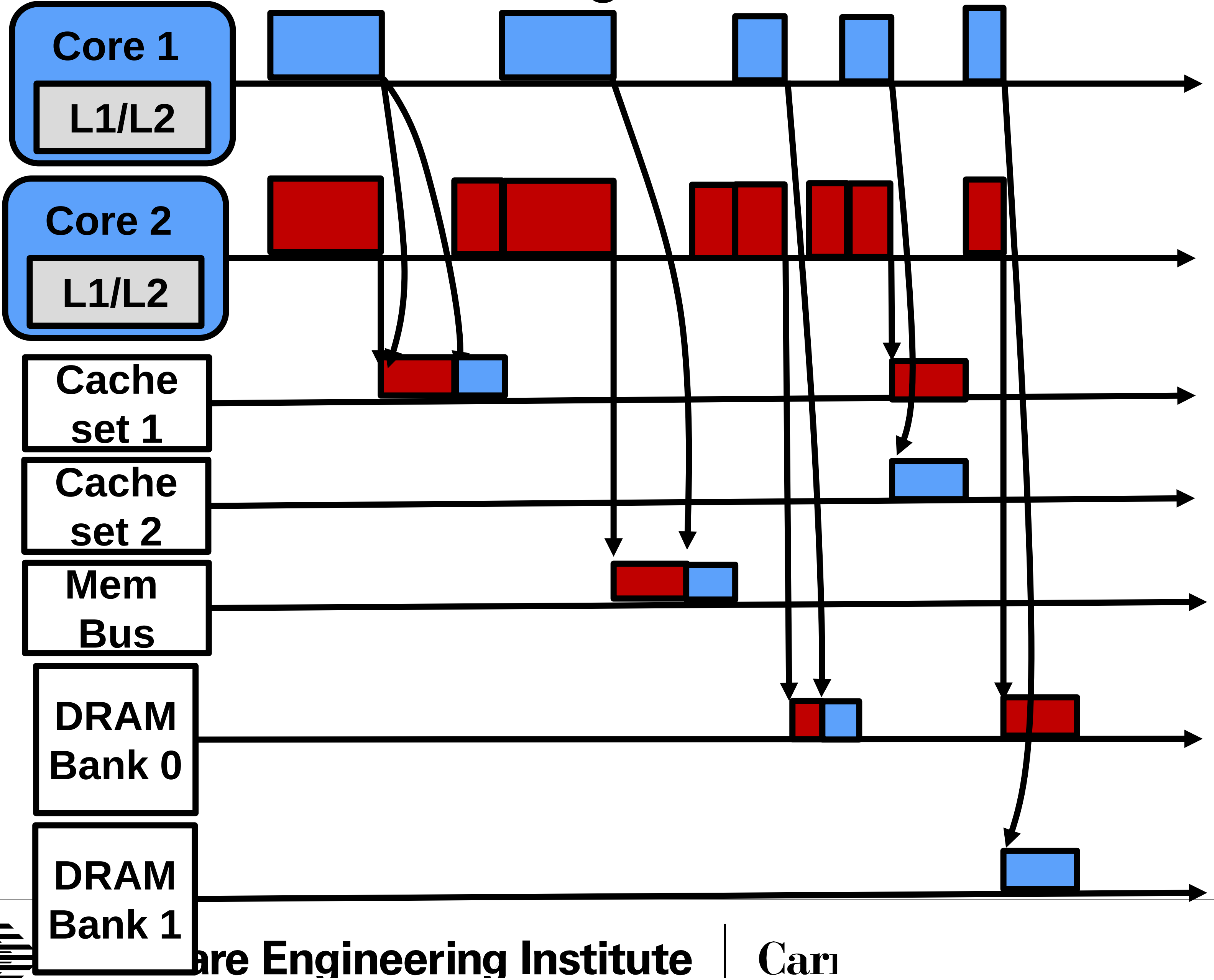
- 1 attacker → Max 5.5x increase
- 2 attackers → Max 8.4x increase
- 3 attackers → Max 12x increase

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



12x increase observed

Solution 1: Partitioning

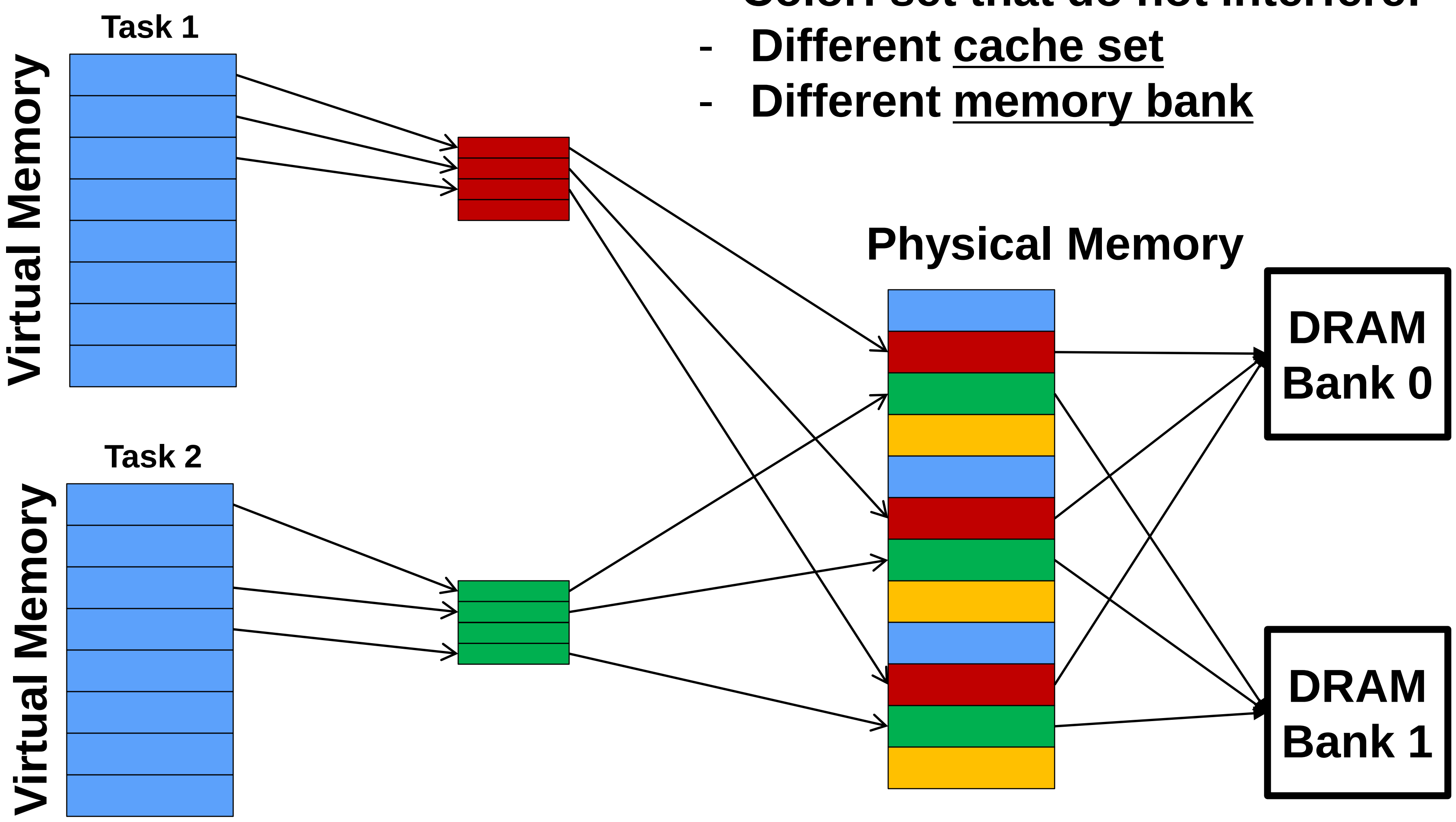


[Distribution Statement A] Approved for public release and unlimited distribution

Solutions 1: Virtual Memory “Coloring”

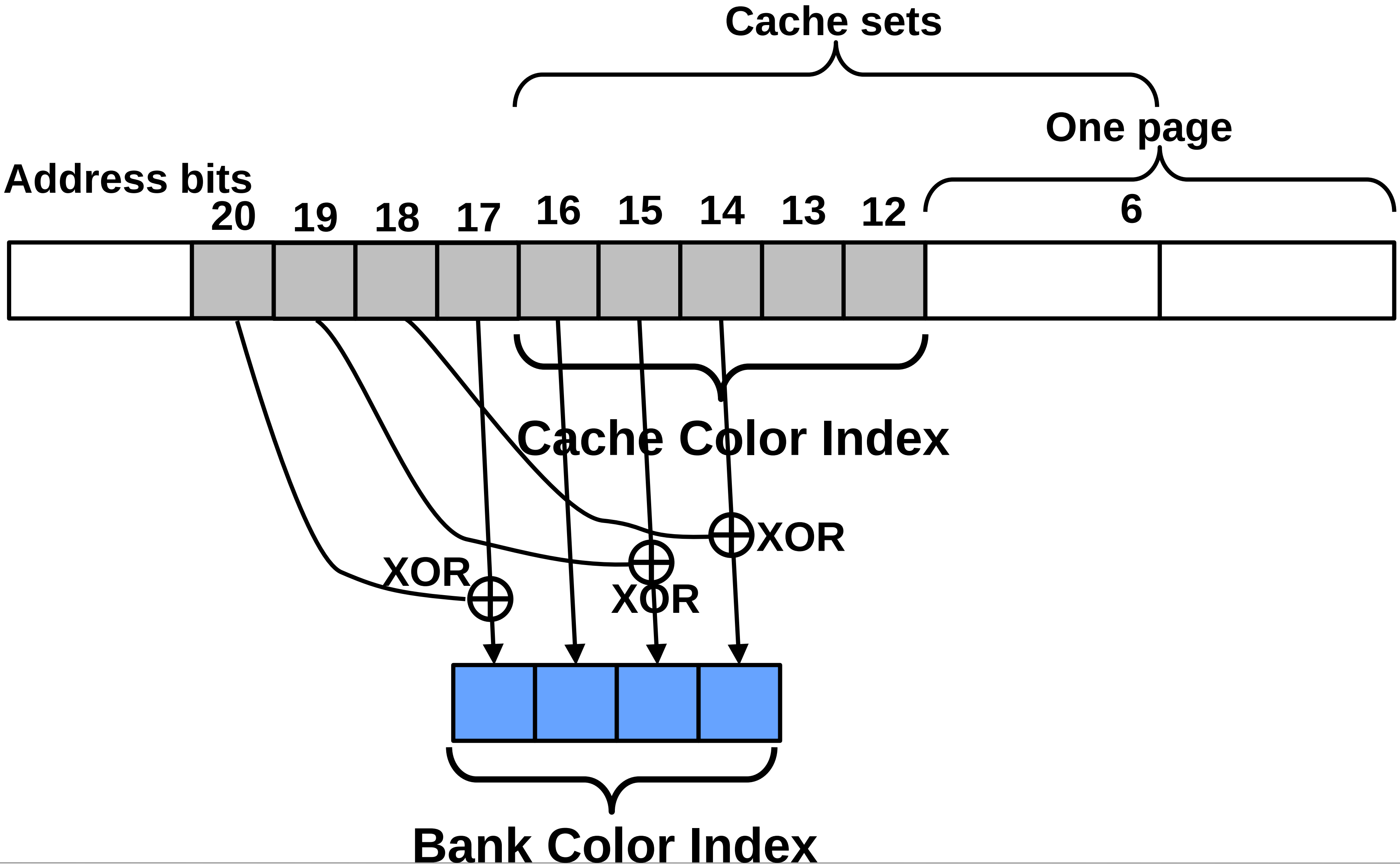
Color: set that do not interfere:

- Different cache set
- Different memory bank

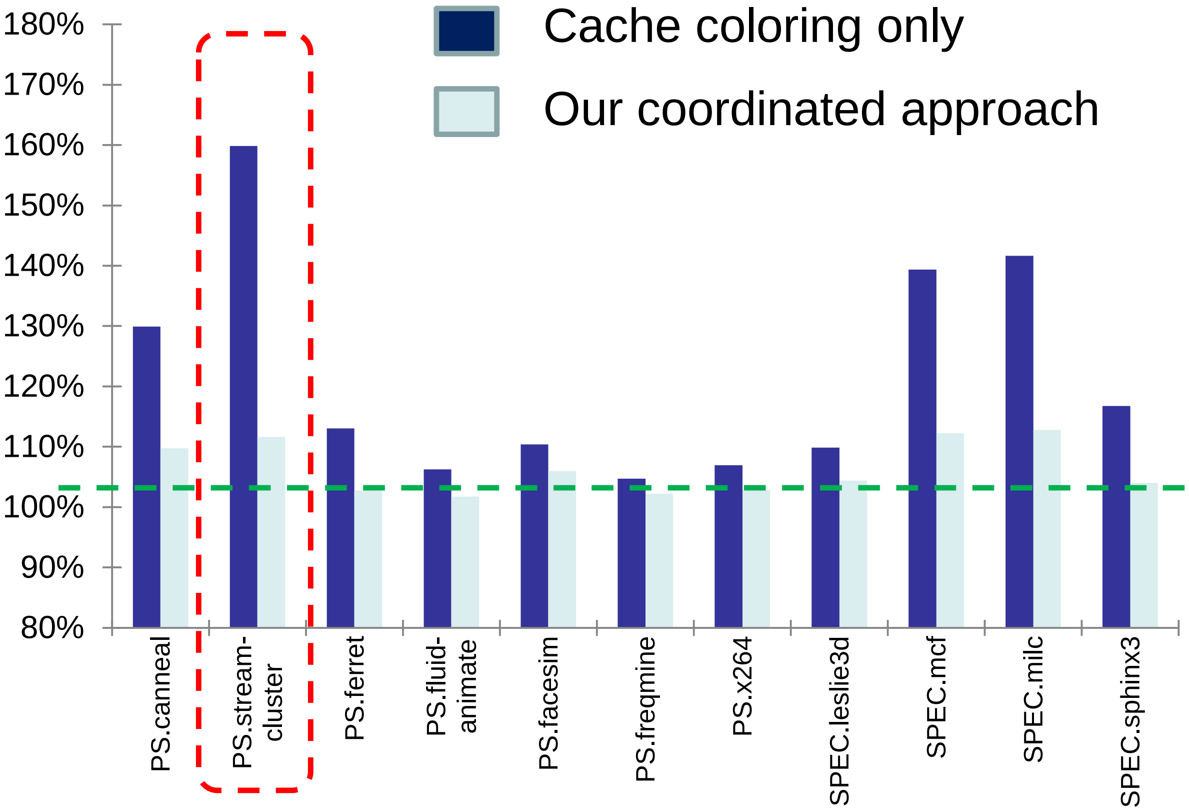


[Distribution Statement A] Approved for public release and unlimited distribution

Solution 1: Challenge – Conflicting Partitions



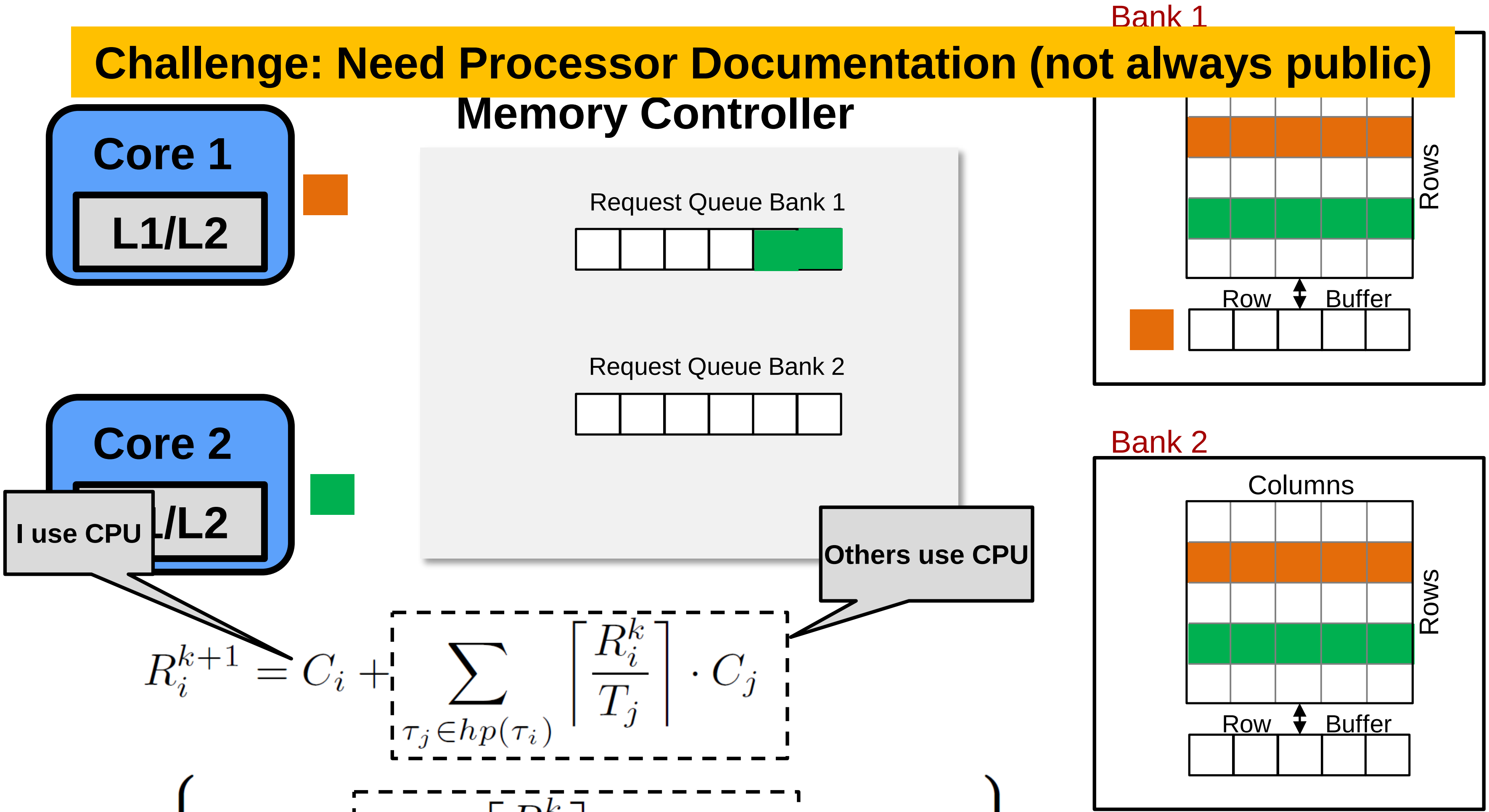
Solution 2: Coordinated Approaches



Challenge: Small Number of Partitions

Solution 3: Predictable Sharing of Partitions

Challenge: Need Processor Documentation (not always public)



$$R_i^{k+1} = C_i + \boxed{\sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot C_j}$$

$$+ \min\left\{ H_i \cdot RD_p + \boxed{\sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot H_j \cdot RD_p,} \; JD_p(R_i^k) \right\}$$

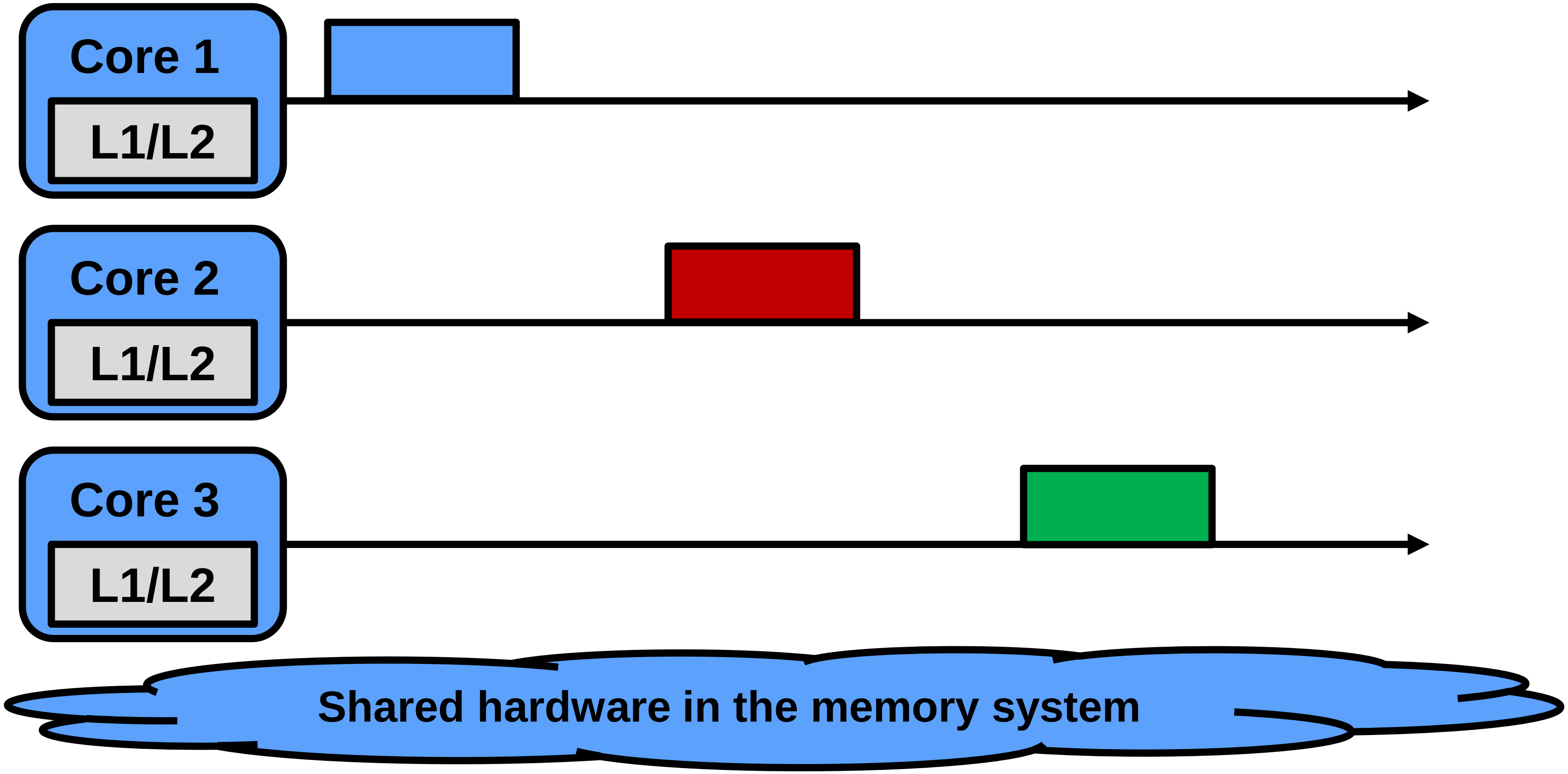
I use CPU

Others use CPU

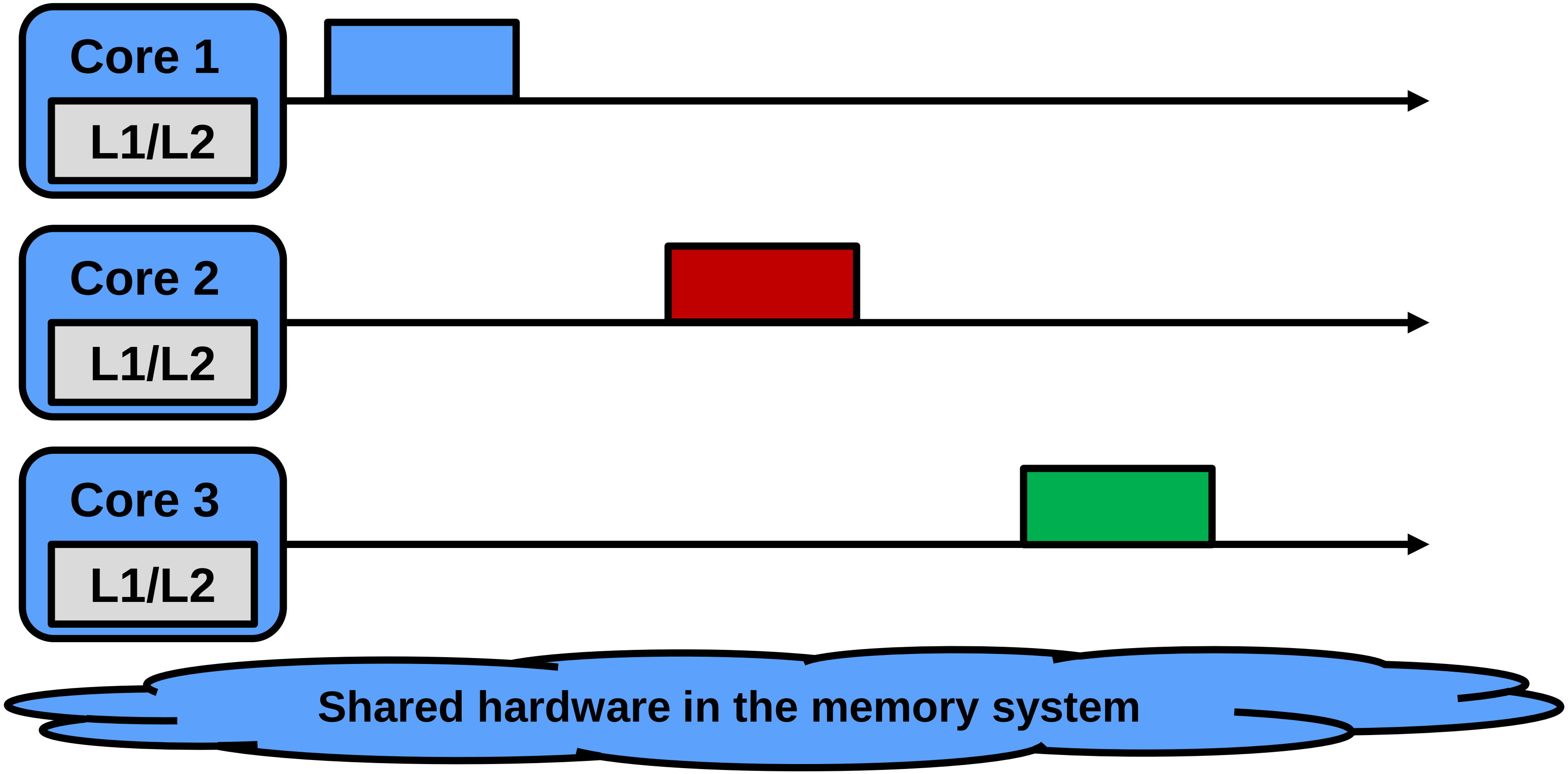
My Mem Reqs.

Others Mem Reqs.

Solution 4: Black Box Analysis

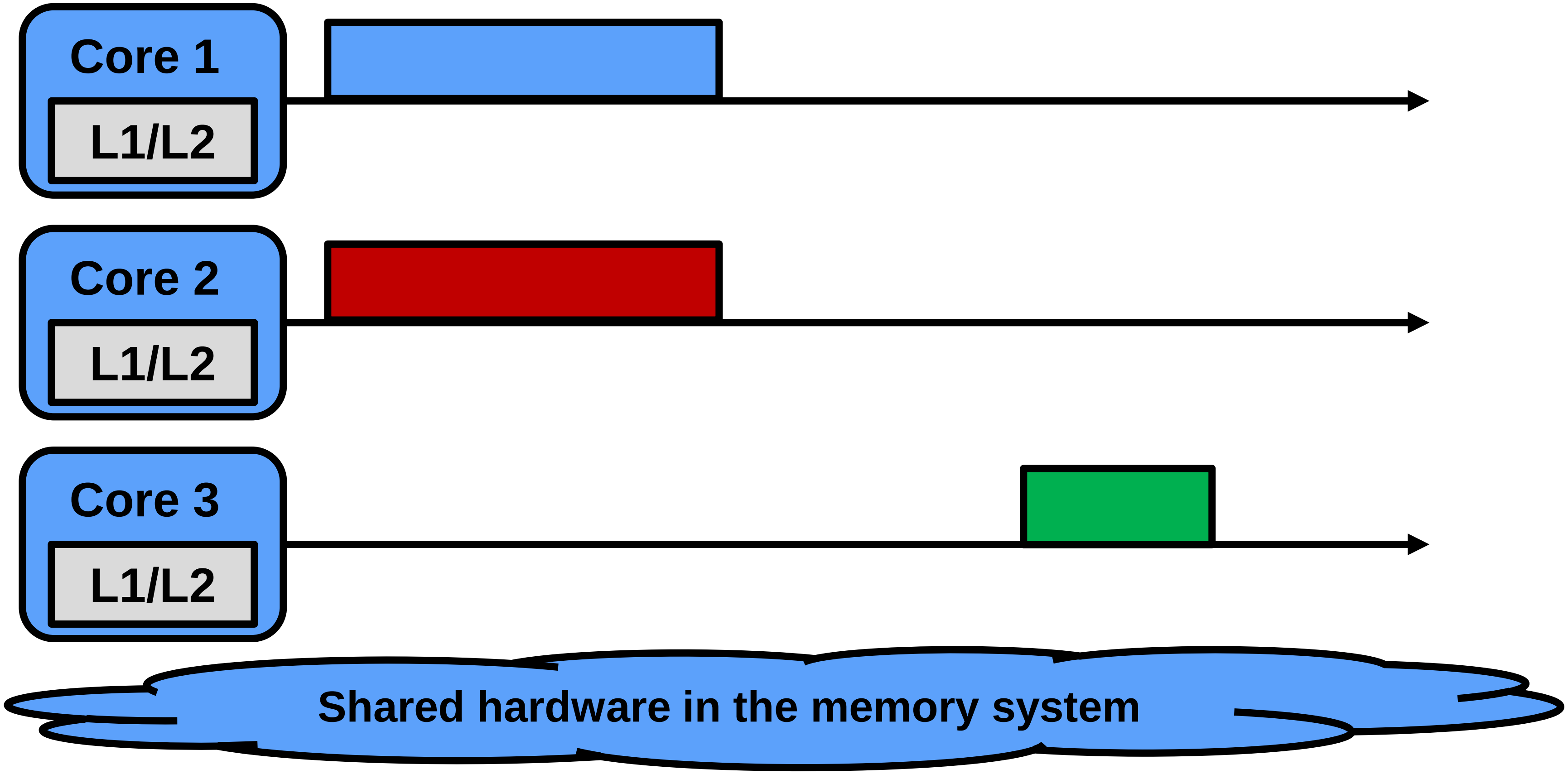


Solution 4: Black Box Analysis



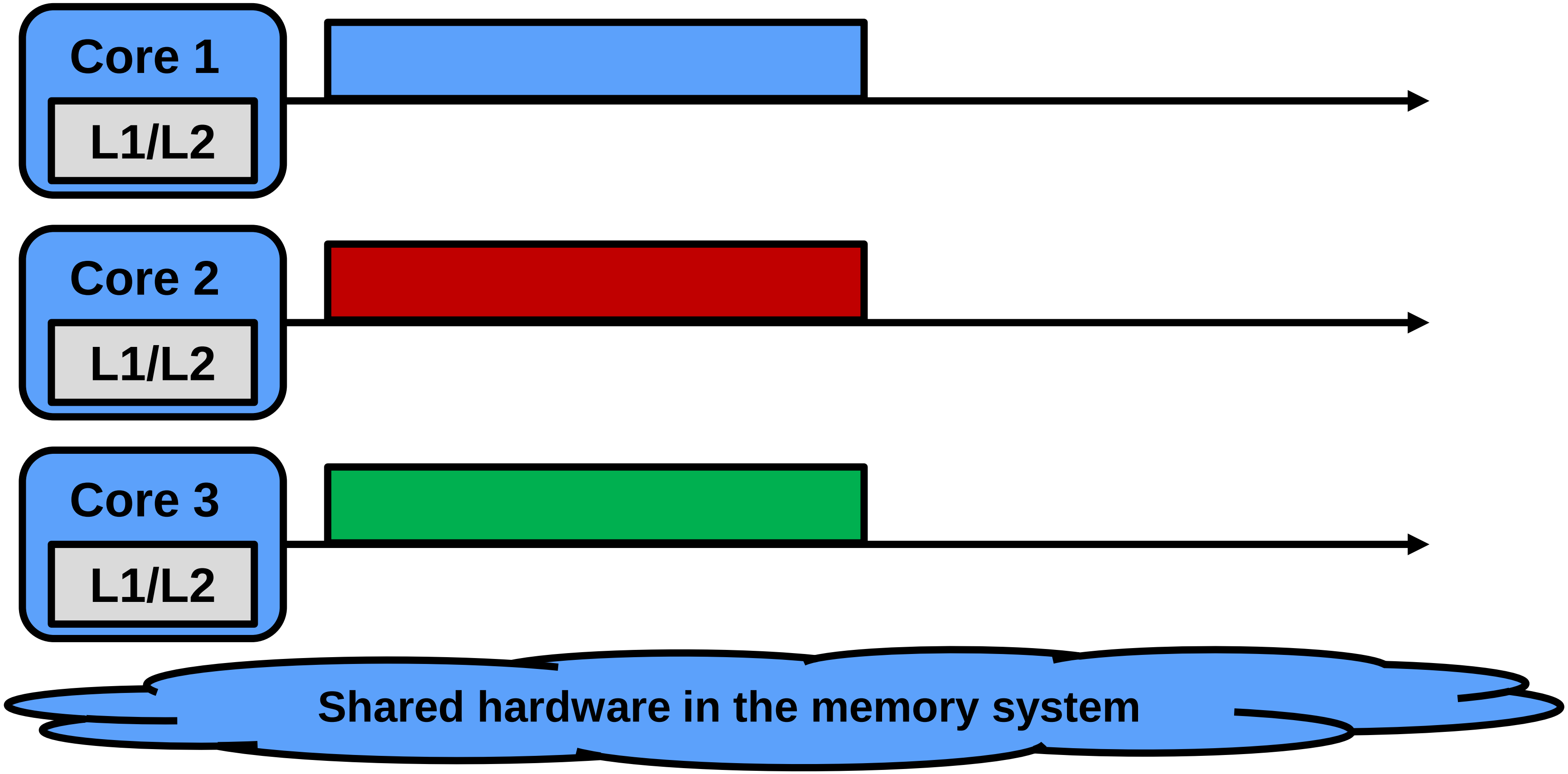
The blue, red, and green tasks execute at different times ⇒ no slowdown

Solution 4: Black Box Analysis



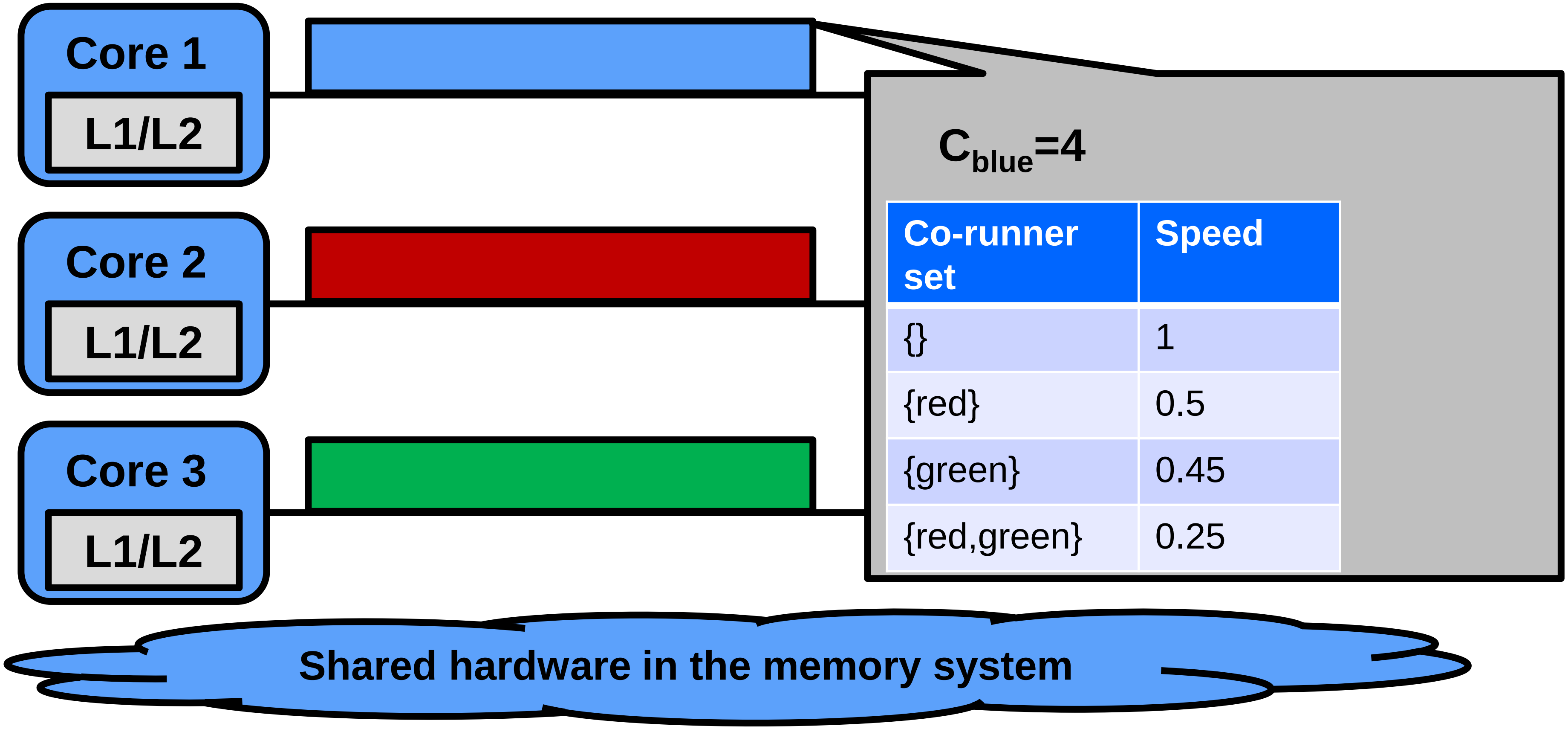
The blue and red tasks execute at the same time ⇒ slowdown ⇒ increased execution time of blue and red.

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

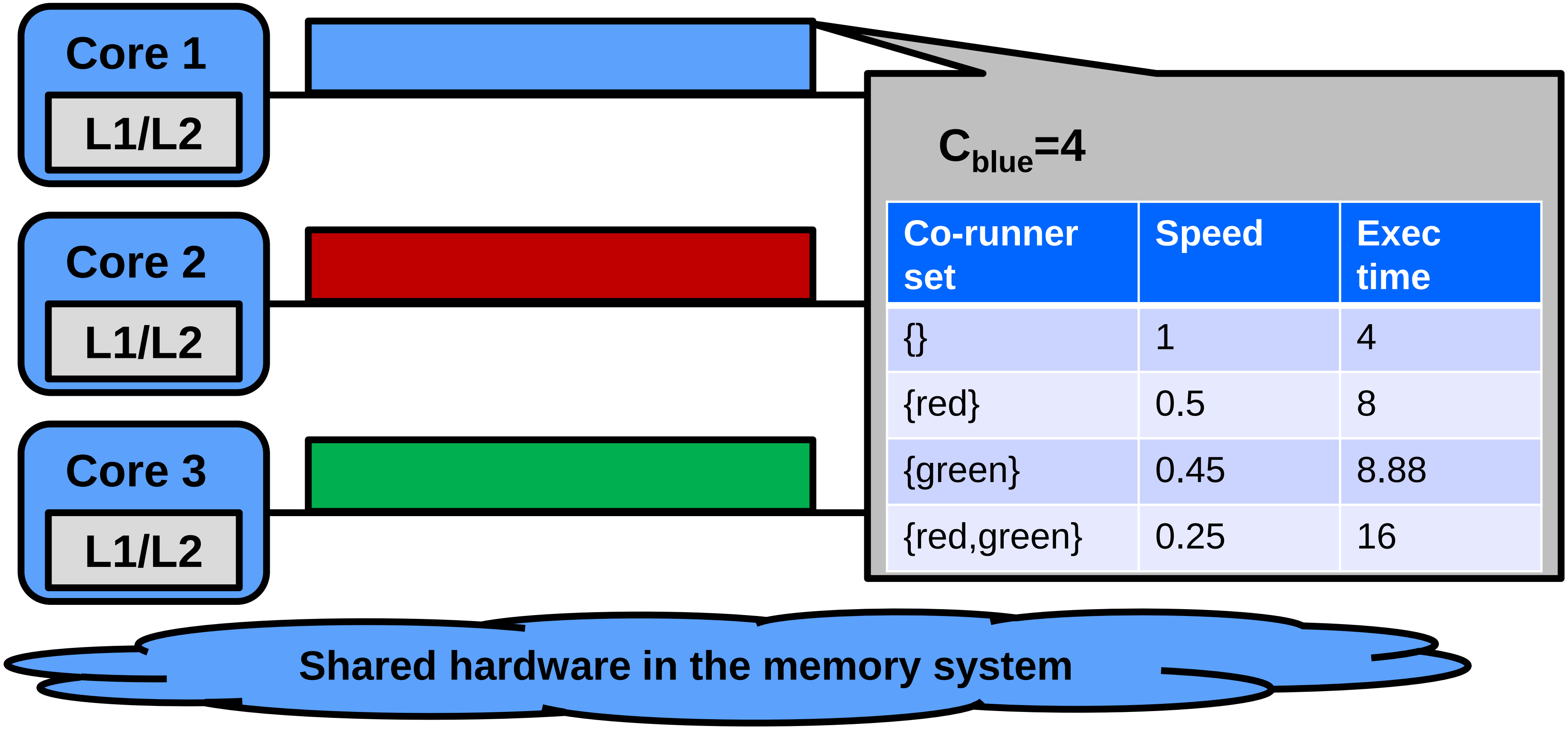


C_{blue}=4

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

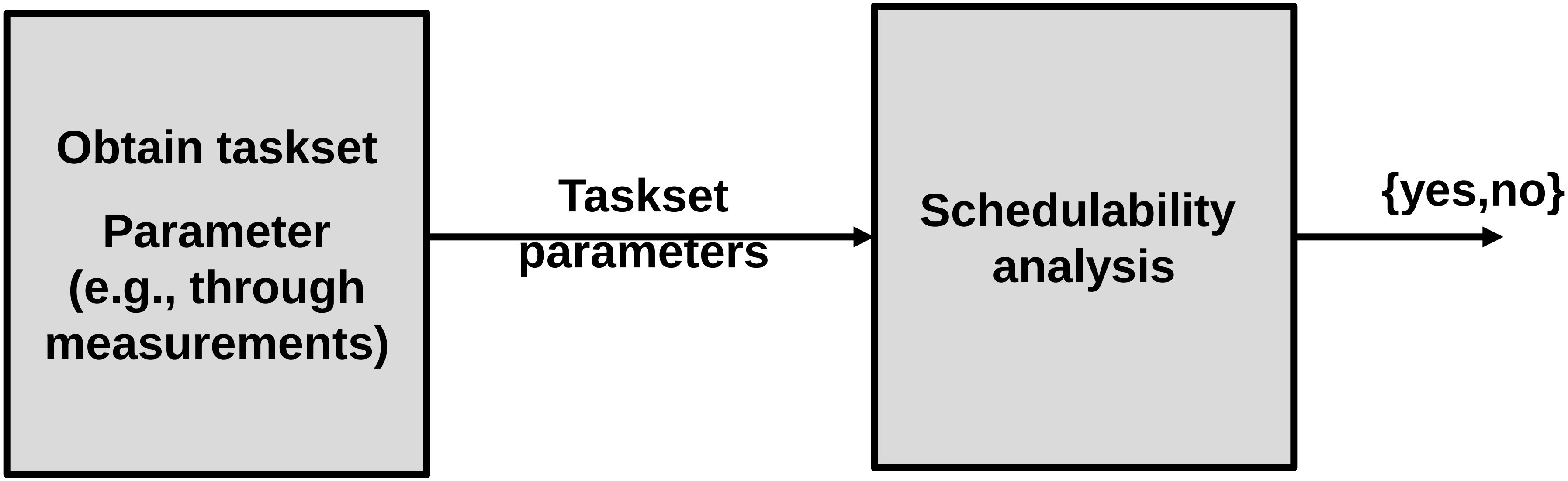


$C_{blue}=4$

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

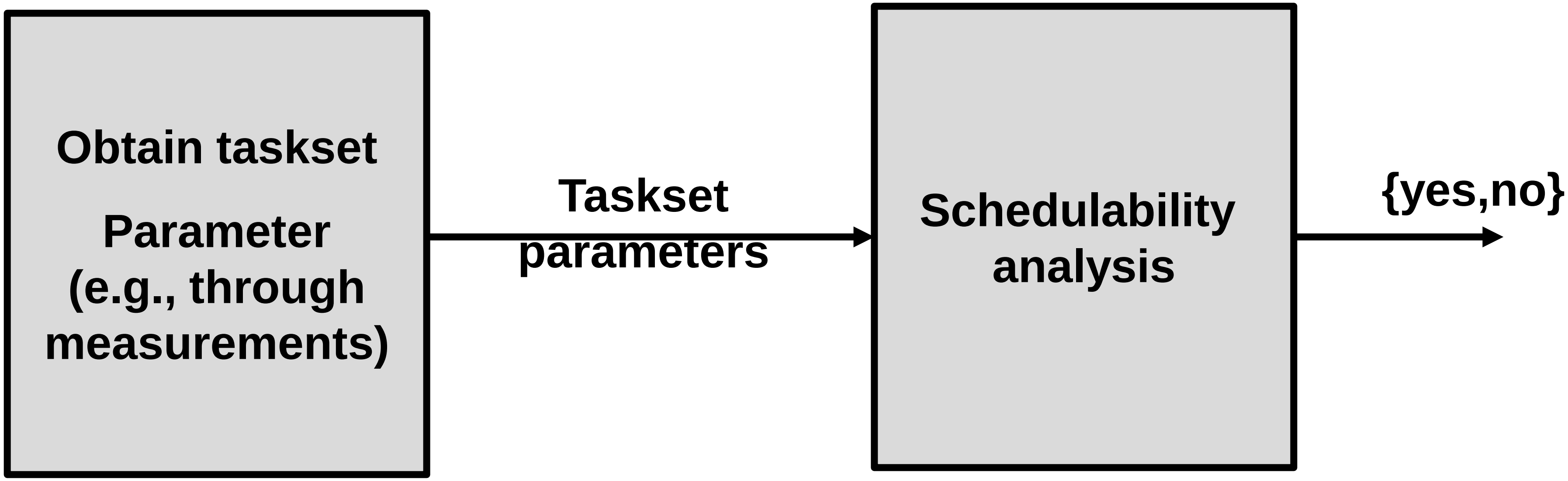
The blue, red, and green tasks execute at the same time ⇒ slowdown ⇒ increased execution time of all tasks.

Solution 4: Black Box Analysis



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

Solution 4: Black Box Analysis



Obtain taskset

Parameter (e.g., through measurements)

Taskset parameters

Schedulability analysis

{yes,no}

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