Guaranteeing Real-Time Requirements on Multicores

Bjorn Andersson and Dionisio de Niz

Software Engineering Institute | Carnegie Mellon

Why Multi-Core Processors?

Processor development trend

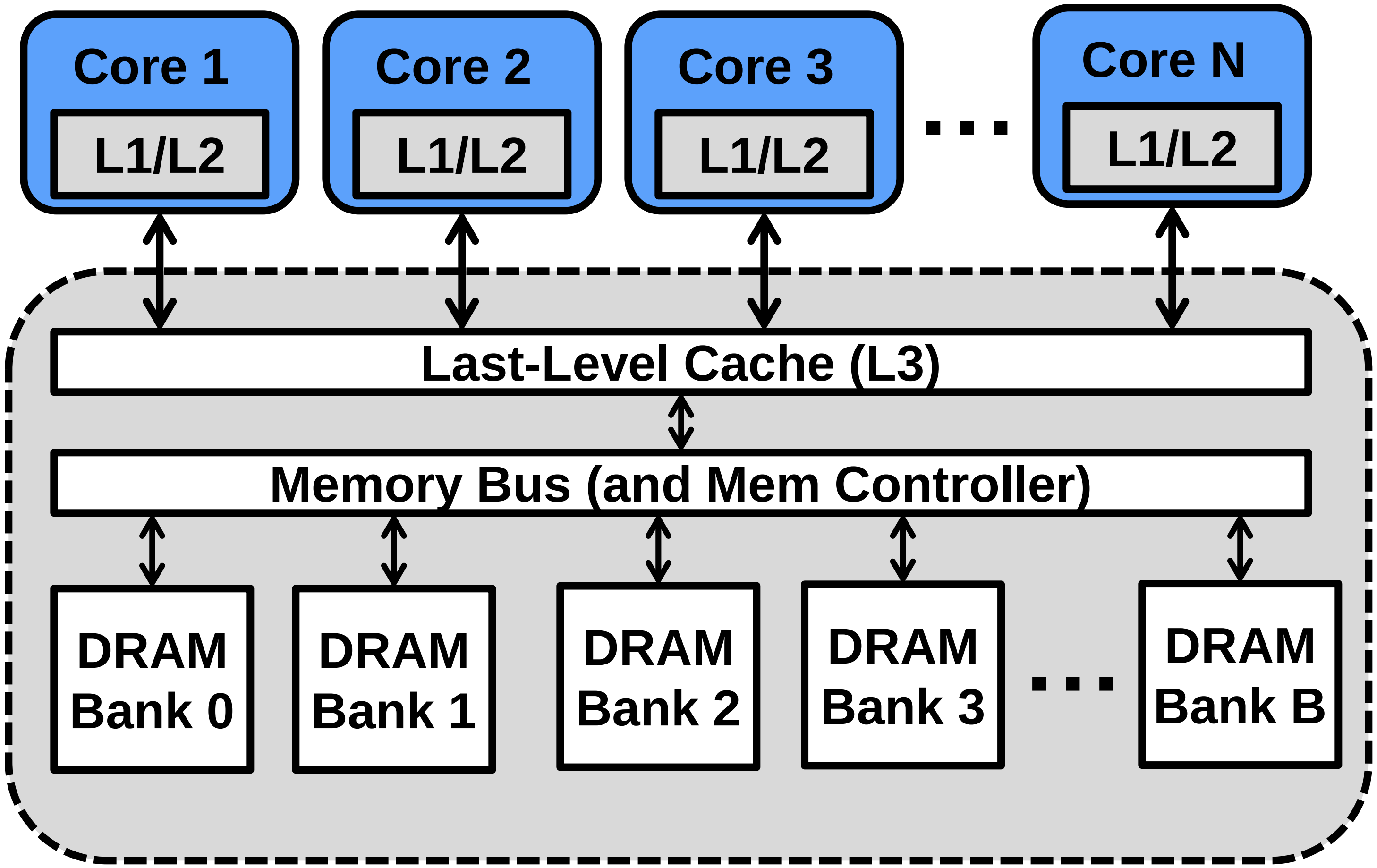
- Increasing overall performance by integrating multiple cores

Embedded systems: Actively adopting multi-core CPUs

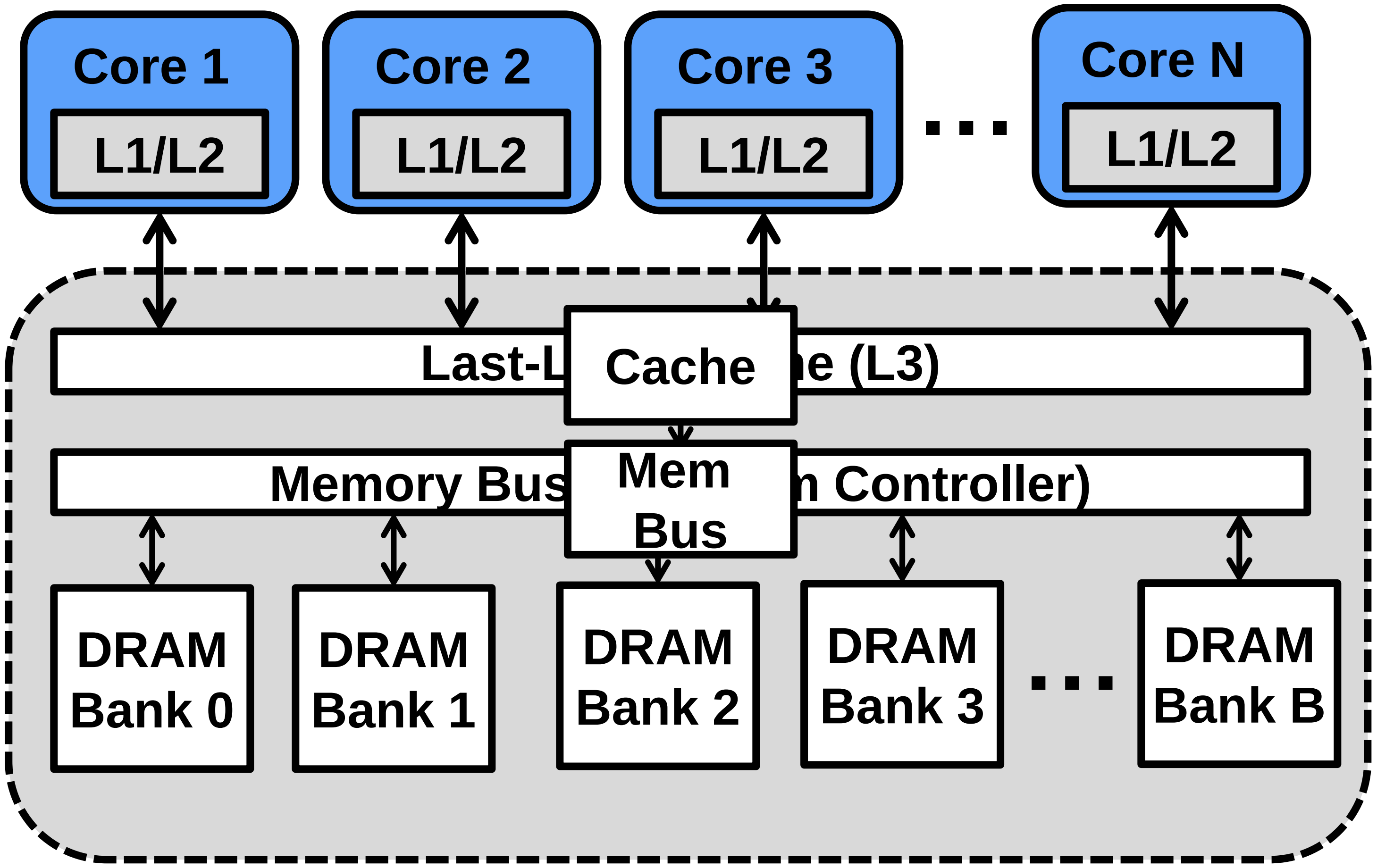
- **Automotive:**
  - Freescale i.MX6 4-core CPU
  - NVIDIA Tegra K1 platform

- **Avionics and defense:**
  - Rugged Intel i7 single board computers
  - Freescale P4080 8-core CPU

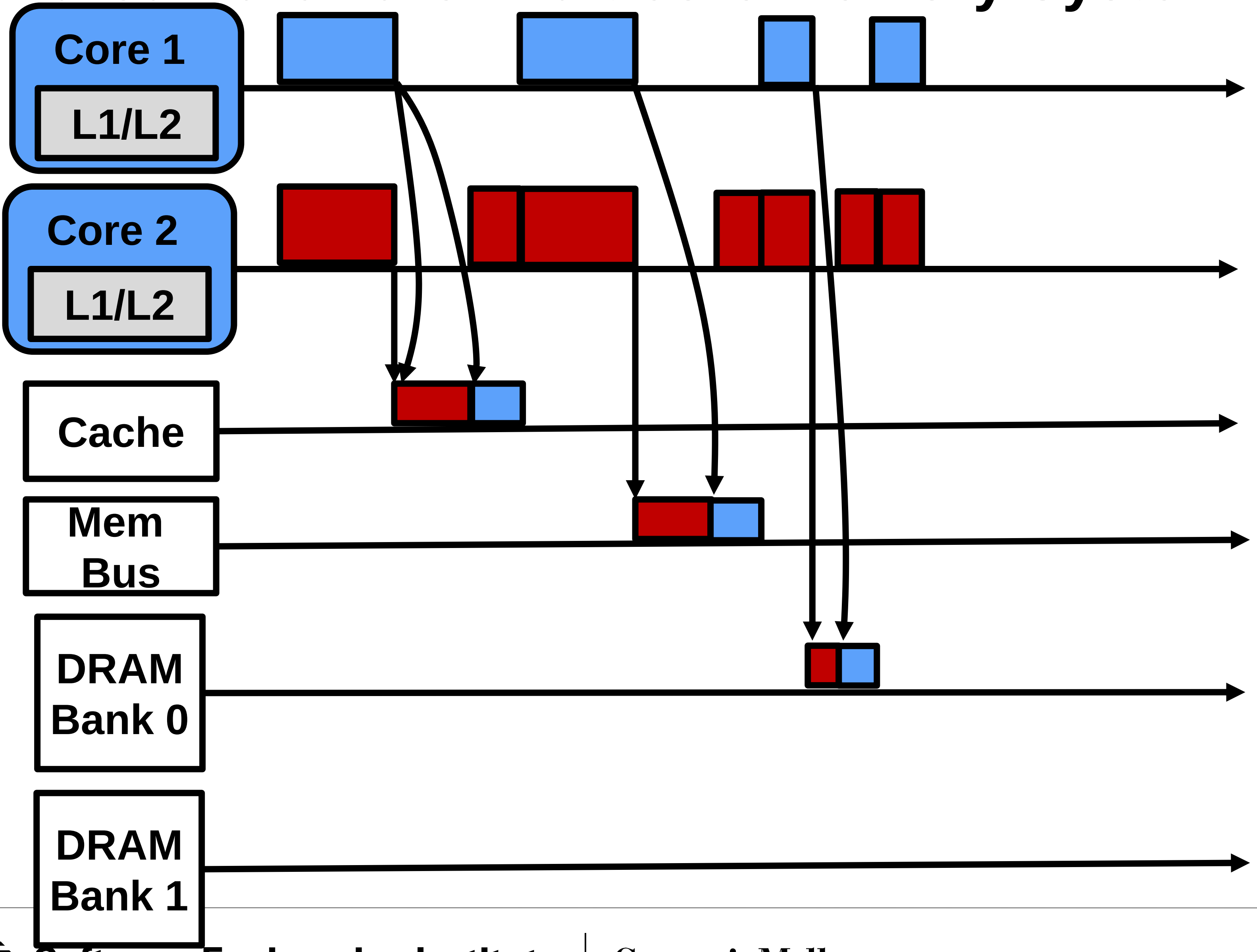
Shared Hardware: Multicore Memory System



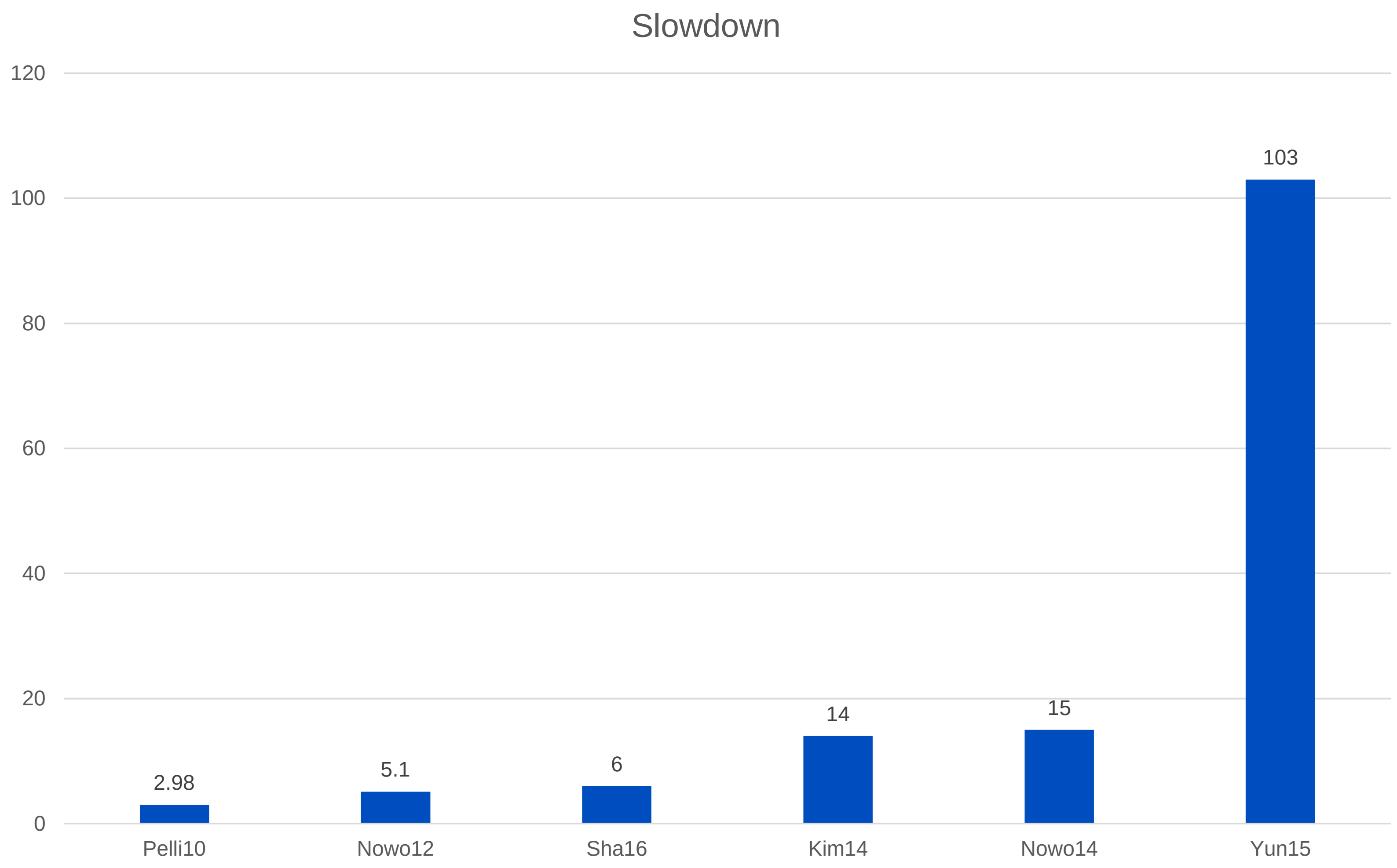
Shared Hardware: Multicore Memory System



Shared Hardware: Multicore Memory System



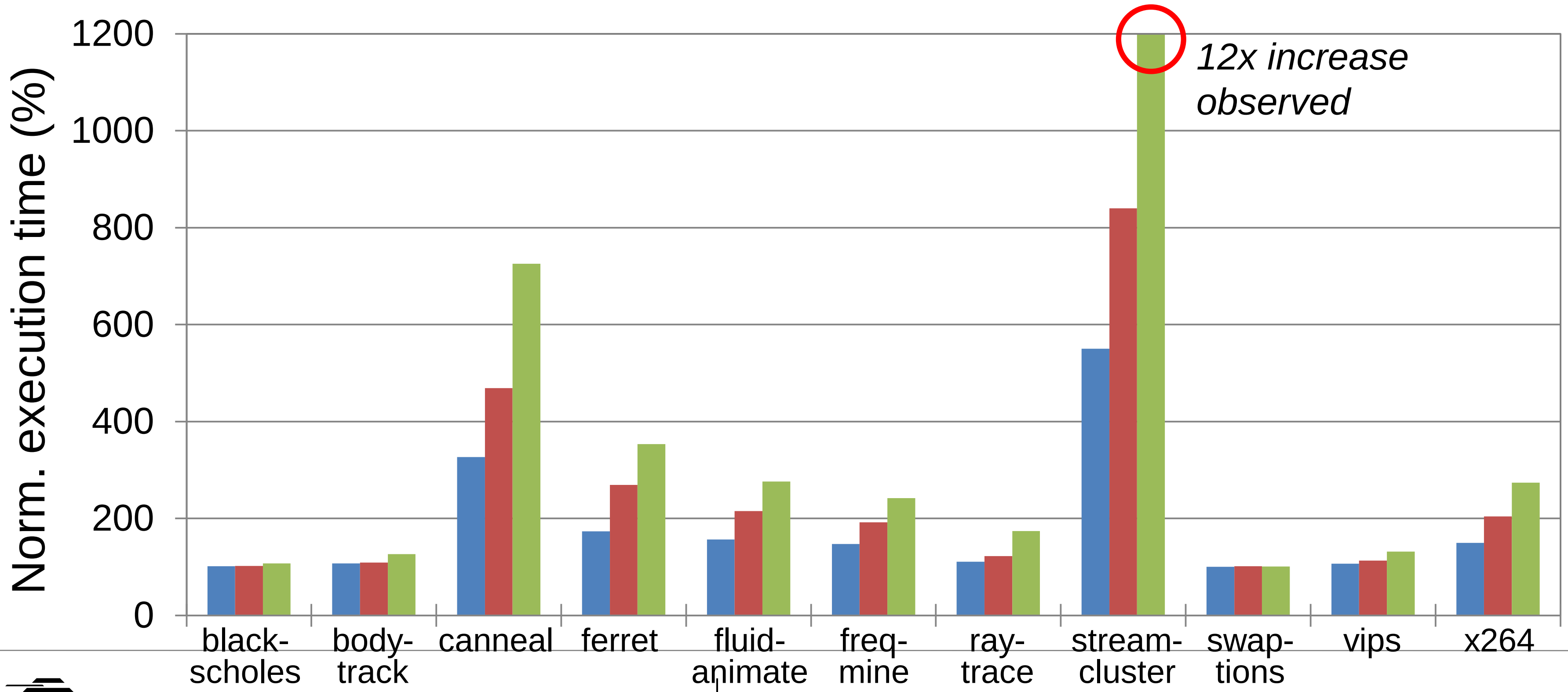
How Bad?


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

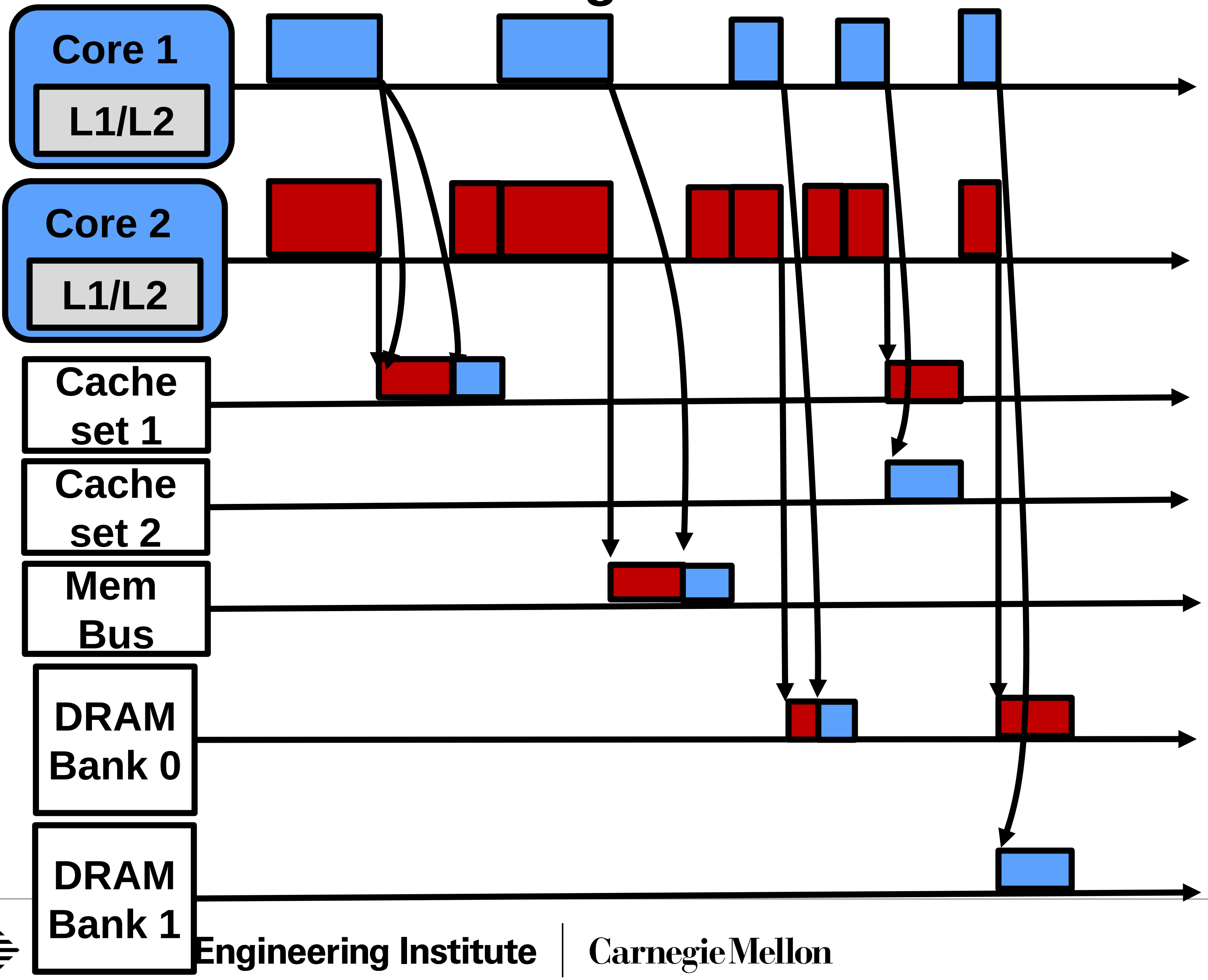
Different for Applications (PARSEC Benchmark)

- 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!



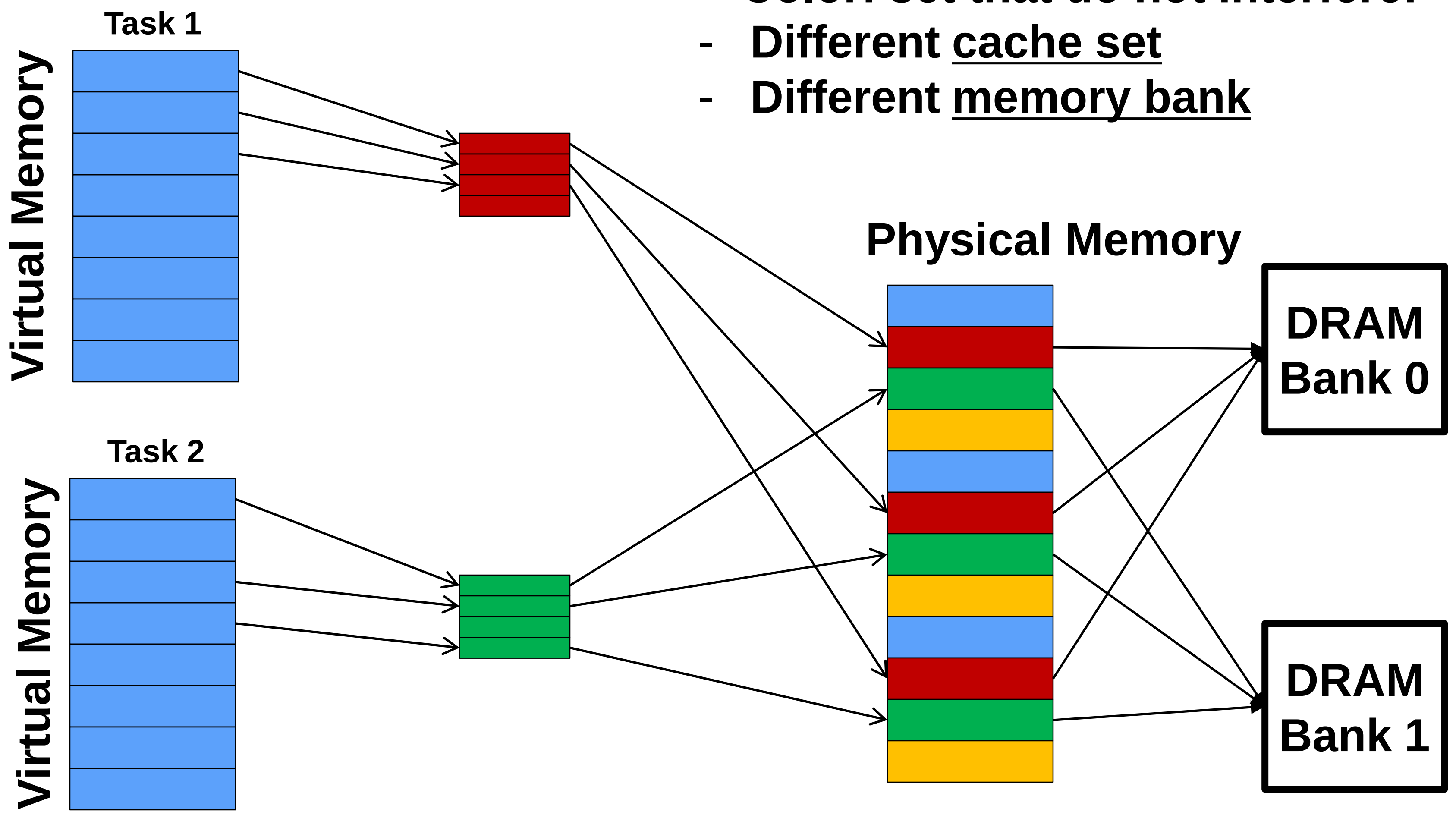
Solution 1: Partitioning



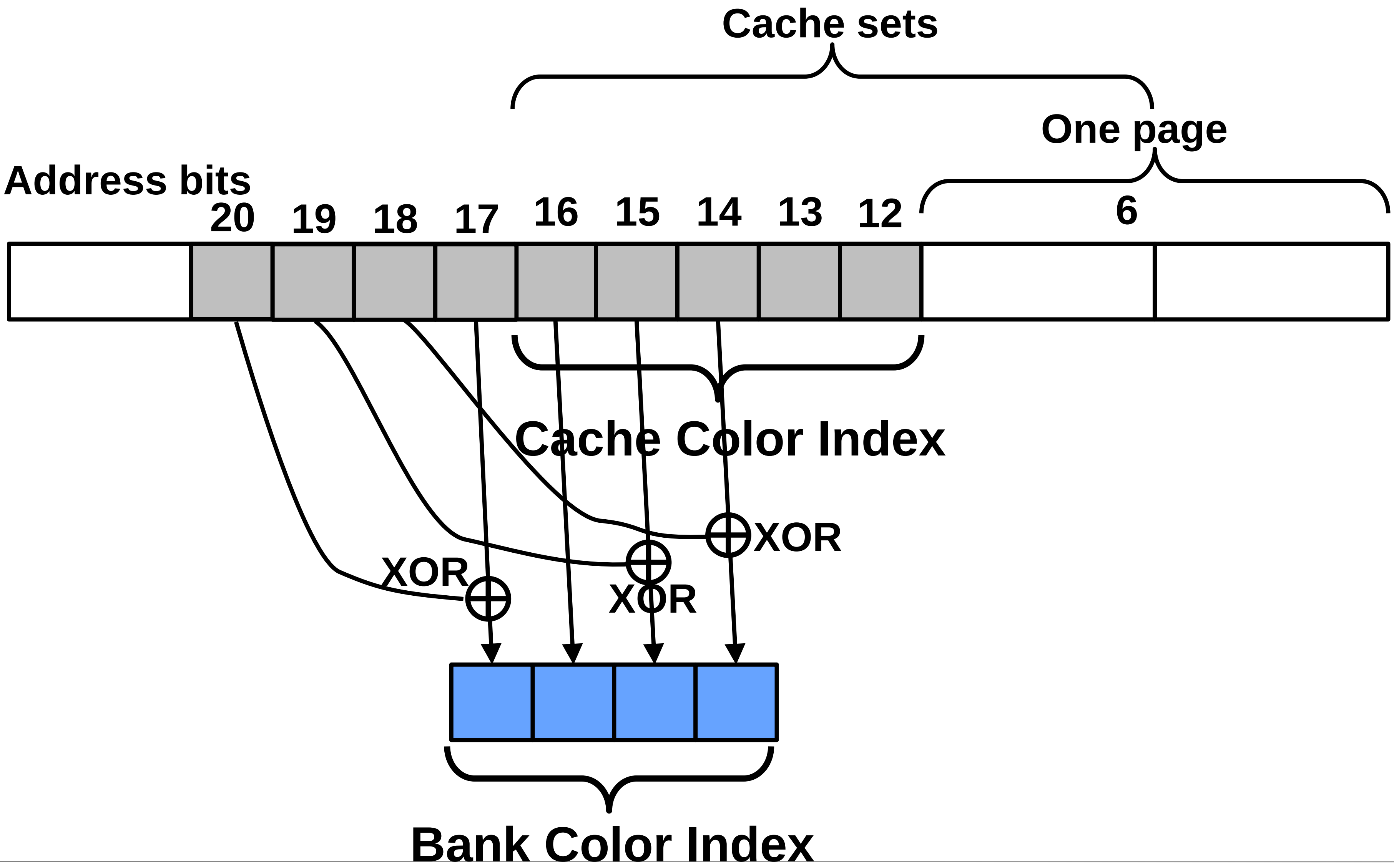
Solutions 1: Virtual Memory “Coloring”

Color: set that do not interfere:

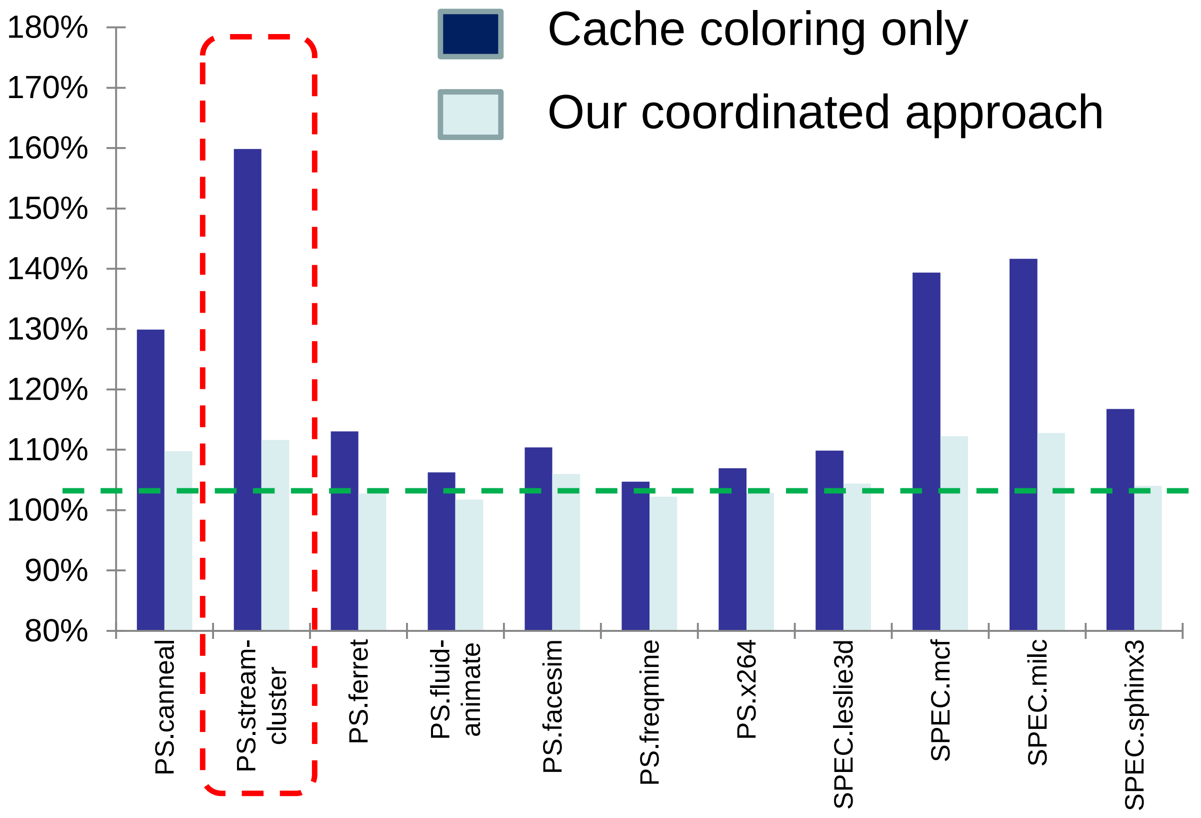
- Different cache set
- Different memory bank



Solution 1: Challenge – Conflicting Partitions



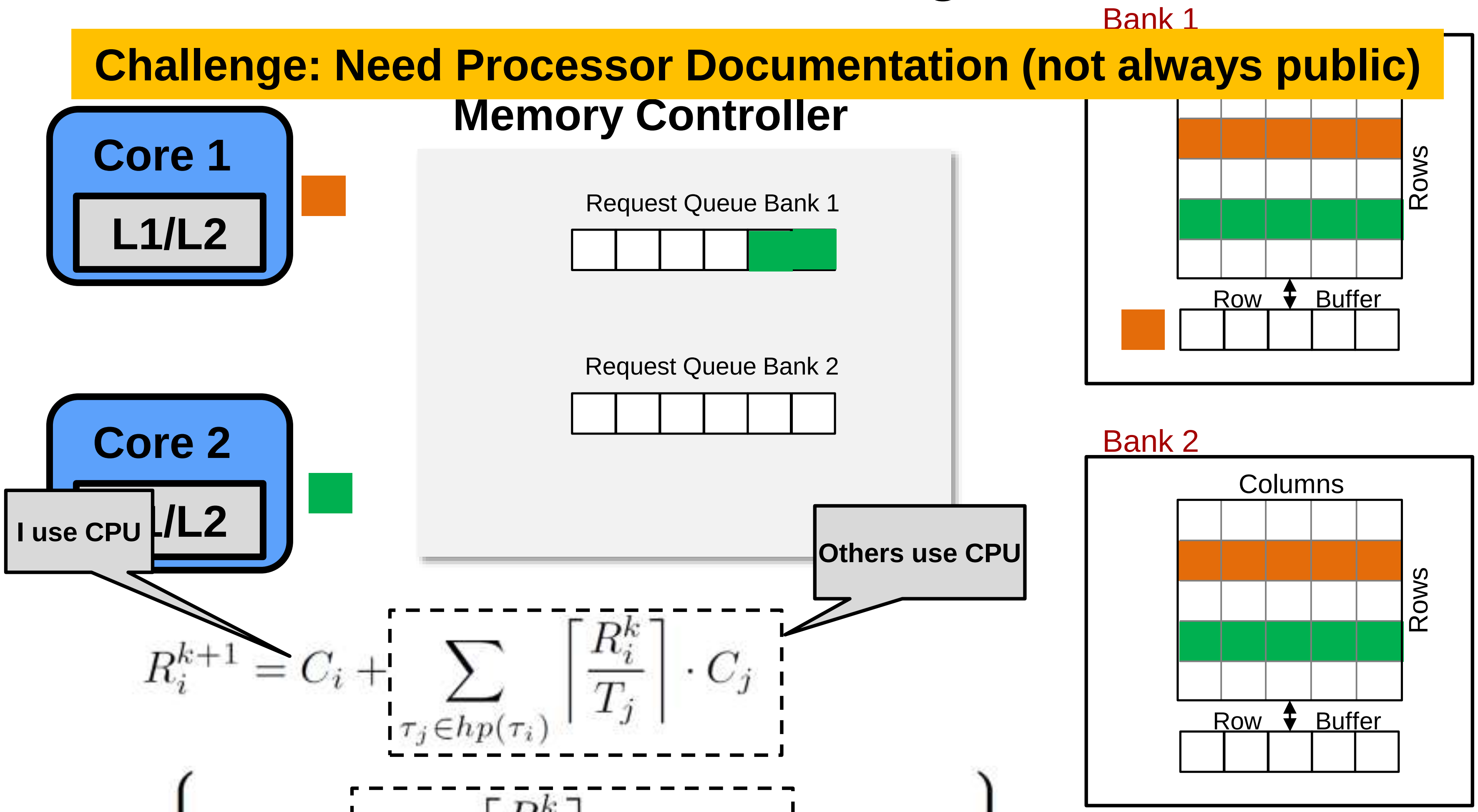
Solution 2: Coordinated Approaches



Challenge: Small Number of Partitions

Solution 3: Predictable Sharing of Partitions

Challenge: Need Processor Documentation (not always public)



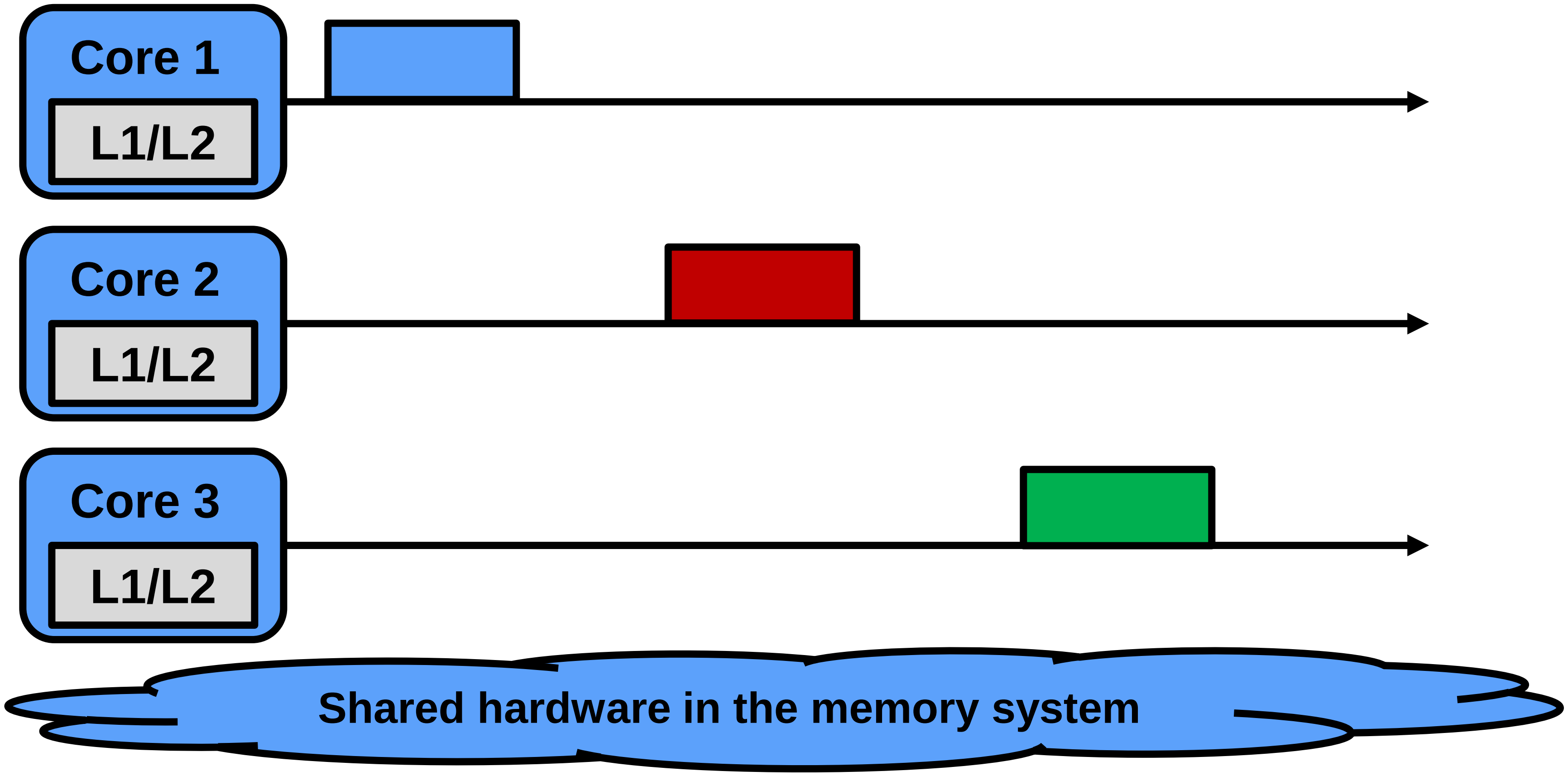
$$R_i^{k+1} = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot C_j$$

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

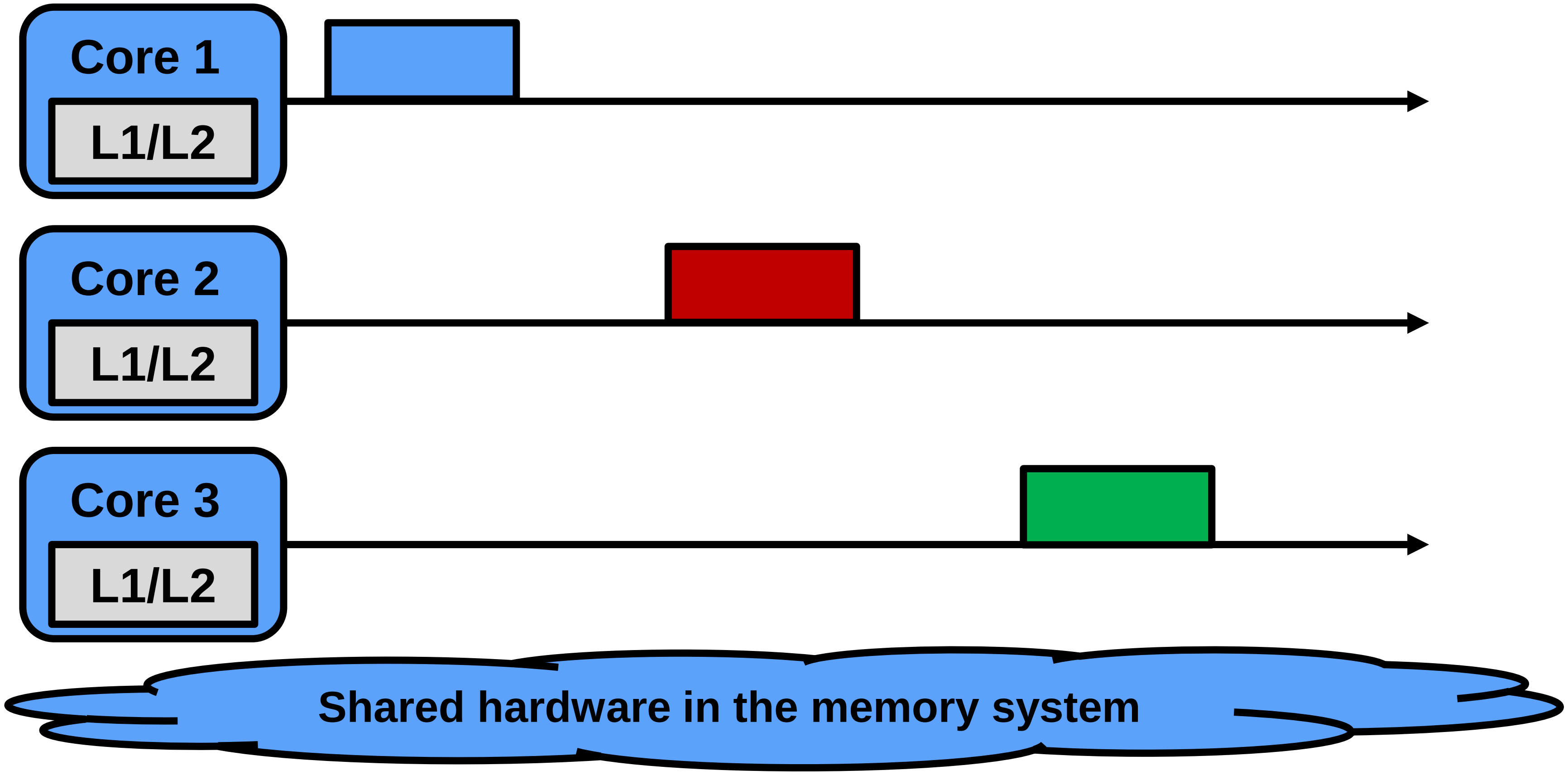
My Mem Reqs.

Others Mem Reqs.

Solution 4: Black Box Analysis

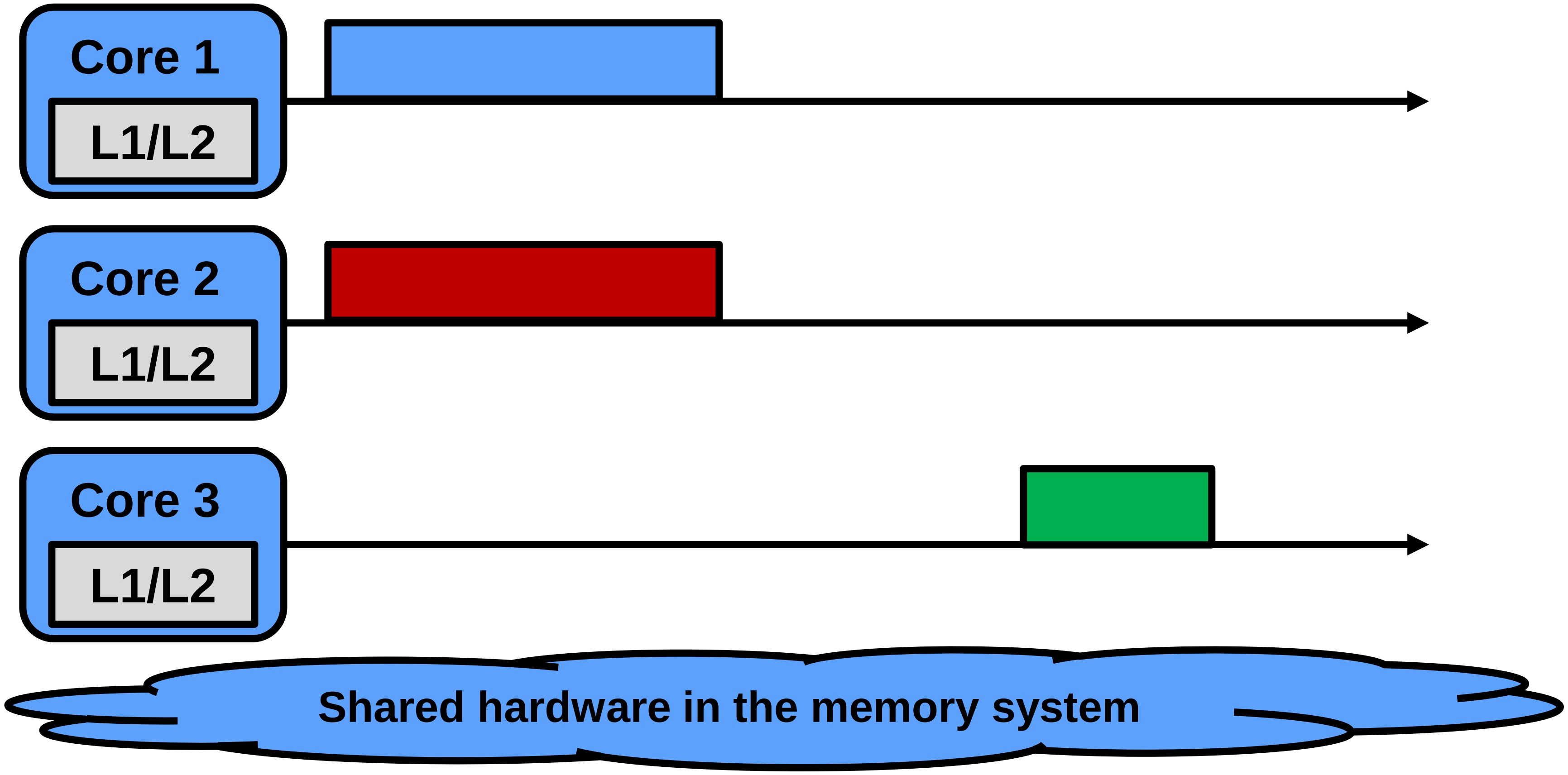


Solution 4: Black Box Analysis



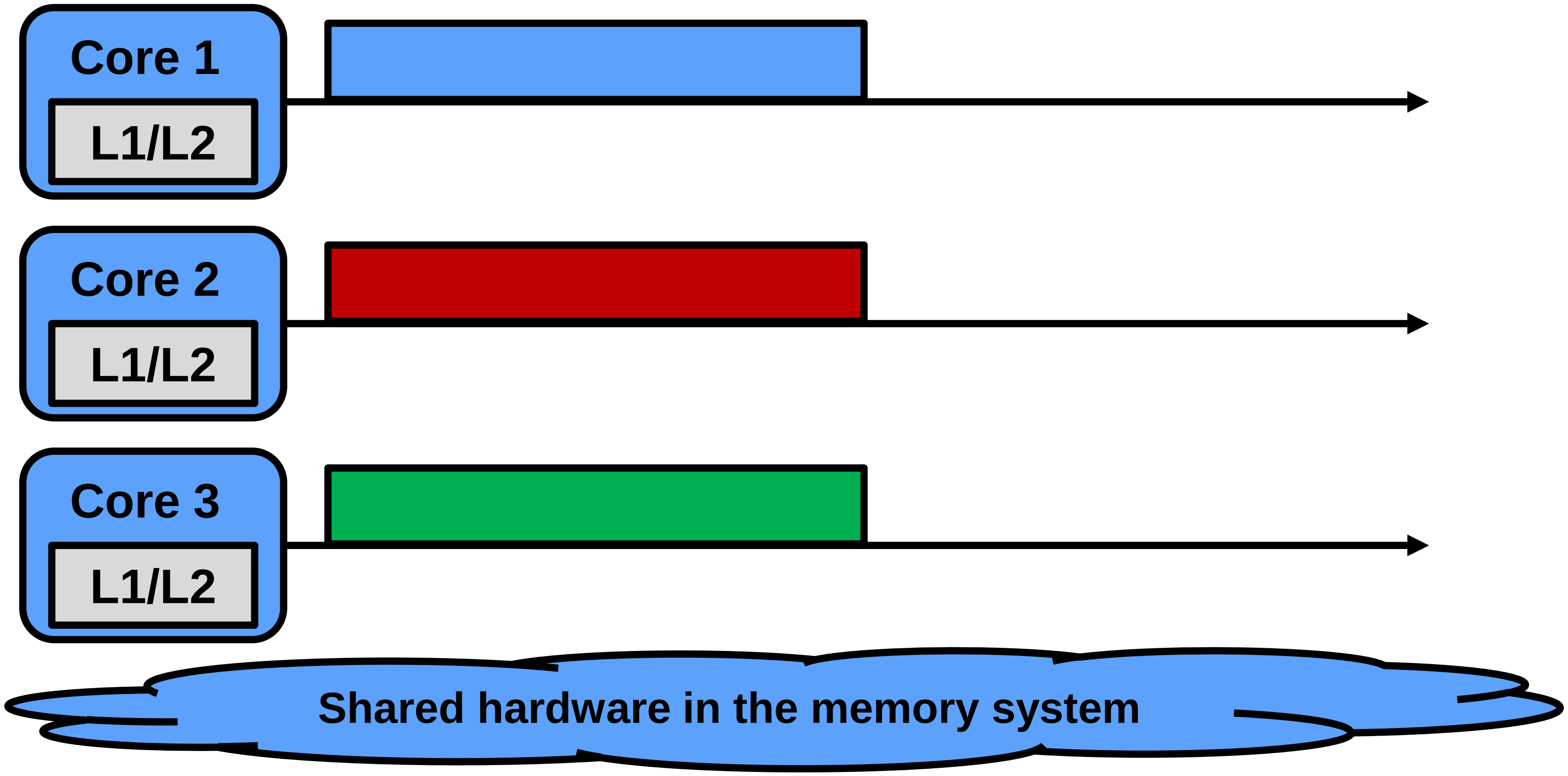
The blue, red, and green tasks execute at different times ⇒ no slowdown

Solution 4: Black Box Analysis



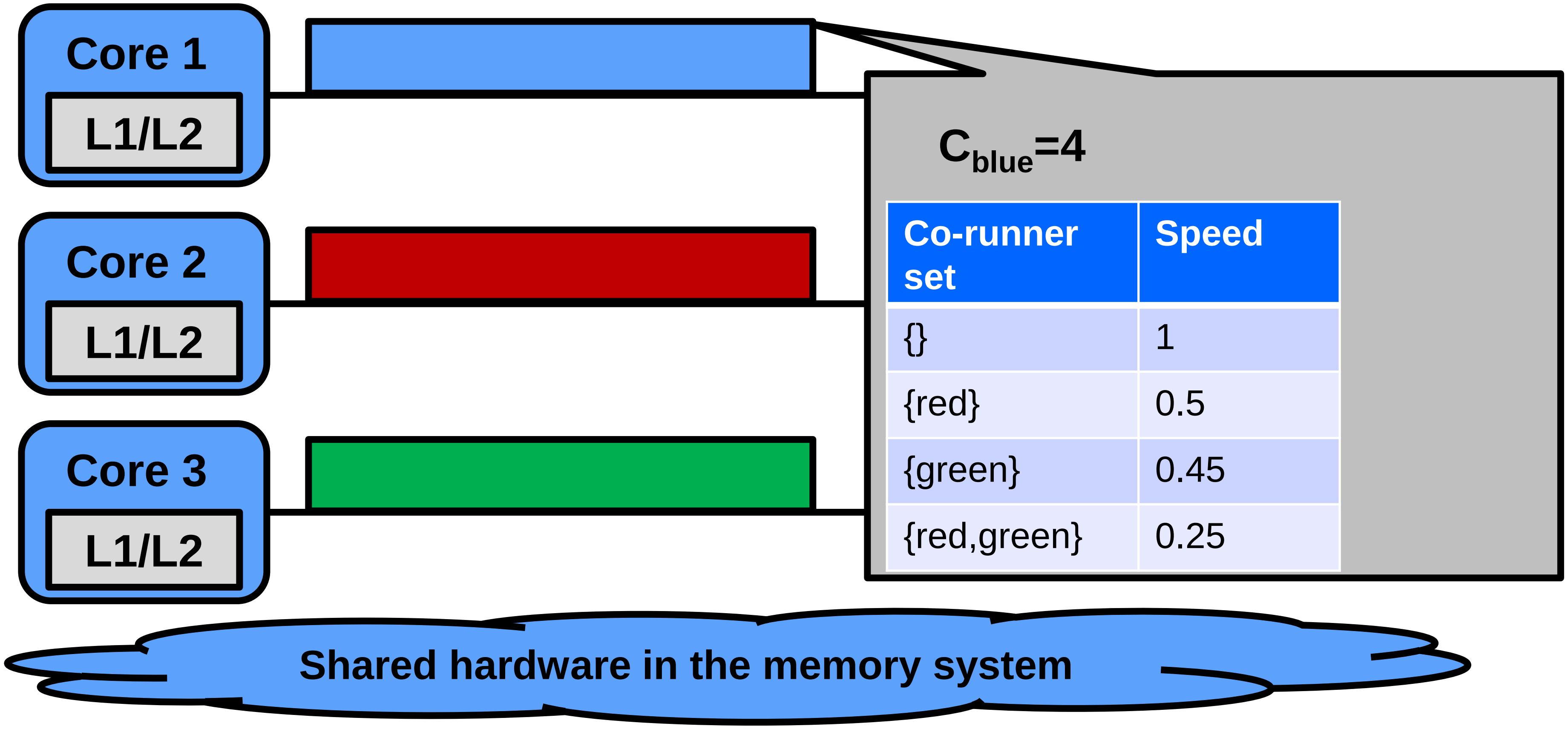
The blue and red tasks execute at the same time ⇒ slowdown ⇒ increased execution time of blue and red.

Solution 4: Black Box Analysis



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

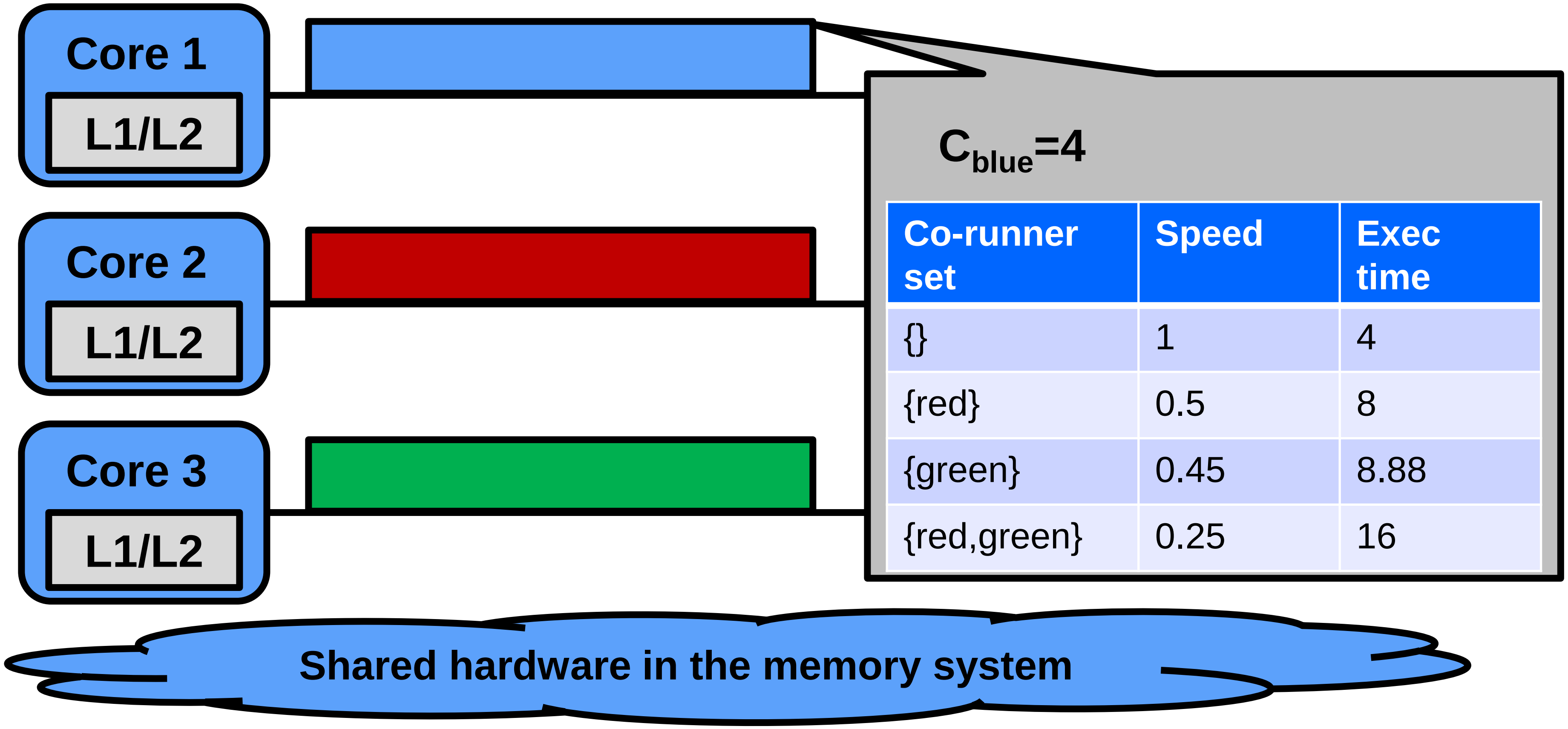
Solution 4: Black Box Analysis



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

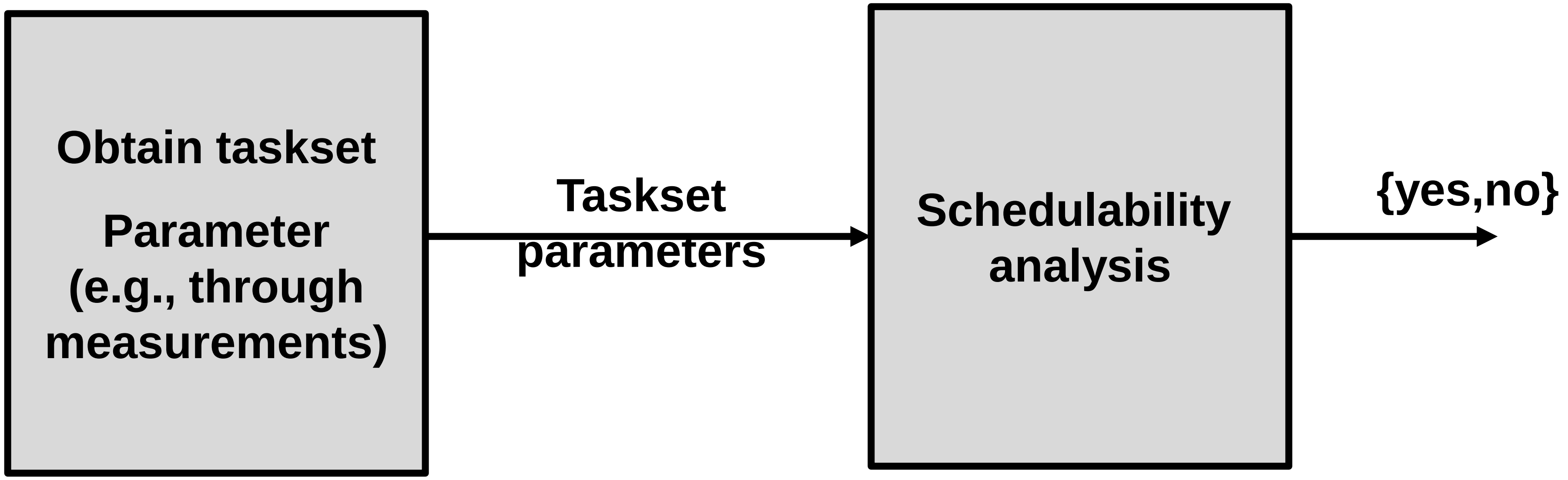


$C_{blue}=4$

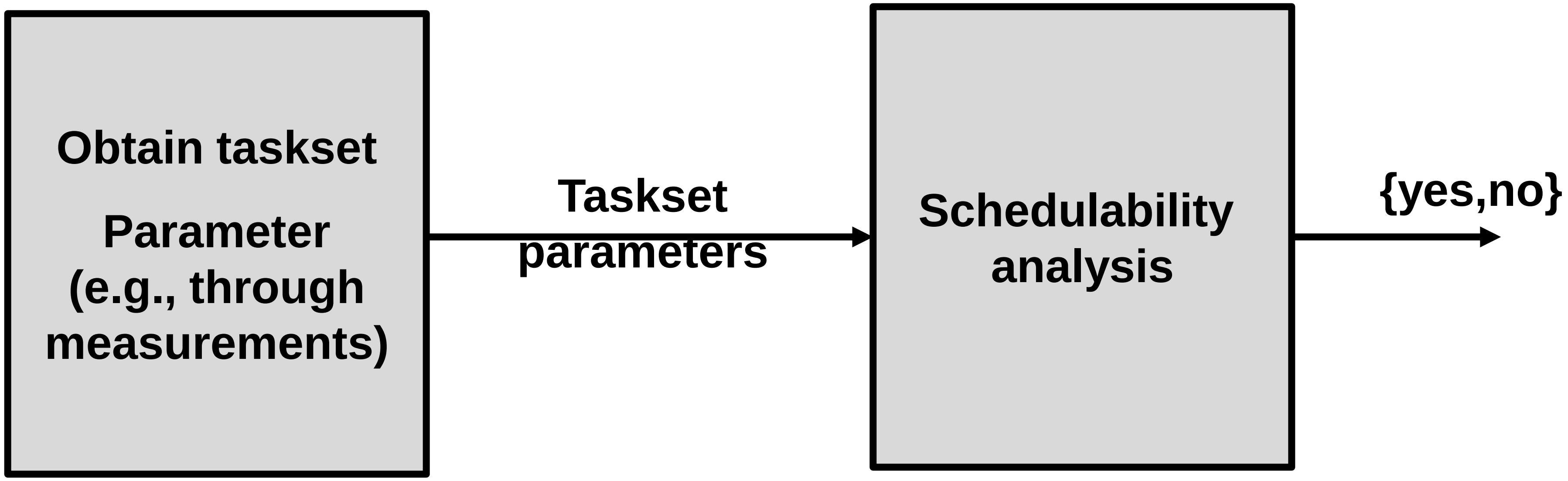
Co-runner set	Speed	Exec time
{}	1	4
{red}	0.5	8
{green}	0.45	8.88
{red,green}	0.25	16

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

Solution 4: Black Box Analysis



Solution 4: Black Box Analysis



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

Limitation: Scalability

MemGuard

What is the problem?

Tasks on different processors may access the memory bus simultaneously; then one has to wait.

How does MemGuard work?

For each task, assign a budget and a period associated with that

budget. At run-time, the number of memory accesses that a task is allowed to perform in a time interval equal to the period is at most the budget; if exceeded, then the task is suspended.

Pro

Provides some temporal isolation wrt to memory bus

Con

Overly pessimistic; Designed only for soft real-time

PREM

What is the problem?

Tasks on different processors may evict each other’s cache blocks and then use other resources in the memory system.

How does PREM work?

Structure a task into three phases. 1st phase: fetch data; 2nd phase: perform computation; 3rd phase: Write back. Memory accesses that result in cache misses are not allowed in the second phase.

Pro

Provides temporal isolation wrt to all resources; works for hard real-time

Con

Assume working set of a task fits in local memory; typically requires specialized hardware (scratchpad memory)

PRET

What is the problem?

Today’s processors are designed for high average-case performance rather than time-predictability.

How does PRET work?

Don’t use caches. Use multithreading to hide memory latency.

Pro

Provides temporal isolation wrt to all resources; works also for hard real-time

Con

Requires specialized hardware; does not work for COTS processors

Summary

Preliminary Solutions

- Partitions
- Coordinated Partitioning
- Shared Partitions
- White / Black Box approaches

Limitations

- Small number of partitions
- Processor documentation not always available

Work Ahead

- Intra-task partitions: shared partitions for lightly used regions
- Increase scalability of black-box approaches
- Unmanaged features / resources:
  – Speculative execution
  – Memory bus
  – I/O
- Parallel tasks
- Tile Processors

Thanks!

References

[Kirk89] D. Kirk, “SMART (Strategic Memory Allocation for Real-Time) Cache Design,” RTSS, 1989.
Main idea: The hardware is designed so that a cache is composed of M partitions and one shared pool. There is also a hardware unit called mapping function which translates each memory access (based on memory address and user id and other info) to a decision on whether the memory access should operate on the shared pool or one of the partitions (and if so, which partition). A task can be assigned more than one partition. The decision on how to allocate partitions to tasks is performed with the idea of maximizing the marginal reduction in the utilization of the taskset.

[Wolfe94] A. Wolfe, “Software-Based Cache Partitioning for Real-Time Applications,” International Workshop on Responsive Computer Systems, 1997.
Main idea: Use the virtual-to-physical address translation to make sure that for different processes, the physical addresses generated map to different cache sets (and hence avoid cache eviction).

[Mueller95] F. Mueller, “Compiler Support for Software-Based Cache Partitioning,” ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems, 1995.
Main idea: Use the idea in [Wolfe94] but let the compiler do the cache coloring.

[Liedtke97] J. Liedtke, H. Hartig, and M. Hohmuth, “OS-controlled cache predictability for real-time systems,” RTAS, 1997.
Main idea: Similar to [Wolfe94] but with OS perspective.

[Bellosa97] F. Bellosa, “Process Cruise Control: Throttling Memory Access in a Soft Real-Time Environment,” Technical Report, University of Erlangen-Nürnberg, 1997.
Main idea: If a given process performs more accesses to the memory bus than it is allowed, then the process is slowed down (by having the TLB miss handler executing NOP instructions).

[Schönberg03] S. Schönberg, “Impact of PCI-bus load on applications in a PC architecture,” RTSS, 2003.
Main idea: Compute the slowdown (from DMA accesses causing memory bus accesses which contend with the program’s accesses on the memory bus) of the execution of a program

[Edwards07] S. Edwards and E. Lee, “The Case for Precision Timed (PRET) Machine,” DAC, 2007.
Main idea: Hw and sw abstractions need to change to be time predictable; e.g., cache should be replaced with scratchpad.

[Rosén07] J. Rosén, A. Andrei, P. Eles, and Z. Peng, “Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-On-Chip,” RTSS’07.
Main idea: Create a TDMA bus schedule according to the needs of a program (both message passing and cache misses).

[Pellizzoni07] R. Pellizzoni and M. Caccamo, “Toward the Predictable Integration of Real-Time COTS based Systems,” RTSS’07.
Main idea: Find a bound on the number of cache misses of a program and a bound on the number of front-side bus accesses from I/O device and compute additional execution time of program. Round-robin bus. Also, perform policing of I/O device.

References

[Steffens08] L. Steffens, M. Agarwal, and P. Wolf, “Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach,” ECRTS, 2008.
Main idea: Analyze cumulative delays of cache misses (low latency streams) using network calculus and also consider message passing. Configure enforcement. Simulation-based approach to obtain cumulative delays of cache misses.

[Schliecker08] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, R. Ernst, “Reliable Performance Analysis of a Multicore Multithreaded System-on-Chip,” CODES+ISSS, 2008.
Main idea: Compute cumulative delay of memory accesses considering contention on the memory bus. Assume work-conserving memory bus but except from that, make no assumption on arbitration.

[Pellizzoni08] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha, “Coscheduling of CPU and I/O Transactions in COTS-based Embedded Systems,” RTSS, 2008.
Main idea: Extension of [Pellizzoni07]. Intel Core2. Implementing the policer.

[Bui08] B. Bui, M. Caccamo, L. Sha, and J. Martinez, “Impact of Cache Partitioning on Multi-Tasking Real-Time Embedded Systems,” RTCSA, 2008.
Main idea: Use genetic programming to decide how many cache colors a task should have.

[Bourgade 08] R. Bourgade, C. Ballabriga, H. Cassè, C. Rochange, and P. Sainrat, “Accurate analysis of memory latencies for WCET estimation,” RTNS, 2008.
Main idea: DRAM memories are organized as banks with one row buffer for each bank. If a memory access has a memory address such that for the bank that holds that data, its row contains the data to be accessed, then the memory latency is small; otherwise it is large. This paper considers this effect in WCET analysis.

References

[Andersson09] B. Andersson, A. Easwaran, and J. Lee, “Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems,” RTSS-WIP, 2009.
Main idea: Model the memory bus of COTS multicore as work-conserving (cache misses). Obtain model from traces.

[Paolieri09] M. Paolieri, E. Quiones, F. Cazorla, G. Bernat, and M. Valero, “Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems,” ISCA, 2009.
Main idea: Create hardware that makes timing predictable. Use TDMA bus and h/w cache partitioning. Implement a WCET computation mode (which ensures that that time a memory operation takes is equal to its maximum).

[Kinnan09] L. Kinnan, “Use of multicore processors in avionics systems and its potential impact on implementation and certification,” DASC, 2009.
Main idea: General discussion on the topic. Mentions the importance of service history. Mentions that cache coherency protocols can operate much faster in multicores than in multiprocessors on separate chips. Mentions that contention/eviction on a shared L2 cache is particularly severe if two tasks on different processor cores run the same software synchronized (this might be an issue if a multicore is used to achieve fault-tolerance). Also points out that certification requires transparency of hardware but chip makers typically do not want to disclose details. Points out that processor cores within a multicore share clock signals and power signals and hence are less fault tolerant than multiprocessors implemented with multiple chips.

References

[Pellizzoni10] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, “Worst Case Delay Analysis of Memory Interference in Multicore Systems,” DATE, 2010.
Main idea: Compute upper bounds on extra execution of a task due to bus contention. Assume TDMA scheduling of tasks. Assume different types of bus arbitration (RR,FCS,priority).

[Schranzhofer10] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, “Worst-Case Response Time Analysis of Resource Access Models in Multi-Core Systems,” DAC, 2010.
Main idea: Extend [Pellizzoni10] with new models for accessing shared hardware resources; one of them is “dedicated phases” which only allows implicit-communication in the beginning and end of a superblock. Use TDMA bus.

[Pellizzoni10] R. Pellizzoni and M. Caccamo, “Impact of Peripheral-Processor Interference on WCET Analysis of Real-Time Embedded Systems,” IEEE Transactions on Computers, 2010.
Main idea: Extend [Pellizzoni07] to a journal article.

[Chattopadhyay10] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, “Modeling Shared Cache and Bus in Multi-cores for Timing Analysis,” SCOPES, 2010.
Main idea: Analyze shared cache and memory bus jointly. Assume TDMA bus and use abstract interpretation in cache analysis. Consider an application comprises multiple tasks with potentially precedence constraints between these tasks. Non-preemptive partitioned scheduling.

[Fuchsen10] R. Fuchsen and R. Winterheim, “How to address certification for multi-core based IMA platforms: current status and potential solutions,” DASC, 2010.
Main idea: Measure slowdown of execution because of sharing resources in the memory system.

[Lv10] M. Lv, W. Yi, N. Guan, and G. Yu, “Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software,” RTSS, 2010.
Main idea: Describe a program with a control flow graph (CFG) and use abstract interpretation to classify memory accesses in each basic block and then formulate a timed automaton for each task with each basic block being a sequence of locations and then analysis bus contention delay with a Timed-Automata model checker (Uppaal).

References

[Dasari11] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and Jinkyu Lee, "Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus," TrustCom, 2011.
Main idea: Compute worst-case response times of tasks making no assumption on the arbitration policy for the memory bus except assuming that the memory bus is work-conserving.

[Rosén11] J. Rosén, C. F. Neikter, P. Eles, Z. Peng, P. Burgio, and L. Benini, “Bus Access Design for Combined Worst and Average Case Execution Time Optimization of Predictable Real-Time Applications on Multiprocessor Systems-On-Chip,” RTAS’11.
Main idea: Similar to [Rosén07].

[Yoon11] M.-K. Yoon, J.-E. Kim, L. Sha, “Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems,” RTSS, 2011.
Main idea: Use special hardware that provides predictable WCET. Consider a TDMA memory bus so that the total utilization of the taskset is minimized (e.g., a task with small period and/or many memory accesses should receive more slots in the TDMA schedule).

[Chattopadhyay11] S. Chattopadhyay and A. Roychoudhury “Scalable and Precis Refinement of Cache Timing Analysis via Model Checking,” RTSS 2011.
Main idea: Extend WCET and CRPD analysis to use model checking for better precision.

[Radojkovic11] P. Radojkovic, S. Girbal, A. Grasset, E. Quinones, S. Yehia, and F. J. Cazorla, “On the evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-Critical Environments,” ACM Transactions on Architecture and Code Optimization, 2011.
Main idea: Create stressing-benchmarks that stress different types of shared resources (e.g. instruction fetch stage in pipeline, later stages in pipeline, L1 cache, L2 cache, memory bandwidth) and find experimentally how much the execution of a pair of stressing-benchmarks is slowed down when executing in parallel. Also, experimentally find slowdown when an application executes in parallel with one of the stressing benchmarks.

[Herter11] J. Herter, P. Backes, F. Haupenthal, and J. Reineke, “CAMA: A Predictable Cache-Aware Memory Allocator,” ECRTS, 2011.
Main idea: Memory allocator where a task specifies not only the size of requested memory block but also the cache color it the requested memory block. This
provides more information to WCET analysis. The allocator is implemented by having one list of free memory blocks per cache color.

References

[Nowotsch12] J. Nowotsch and M. Paulitsch, “Leveraging multi-core computing architectures in avionics,” EDCC, 2012.
Main idea: Provide a test approach that models the worst-case behavior for the case of concurrent network and memory usage by multiple applications.

[Mancuso13] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, “Real-Time Cache Management Framework for Multicore Architectures,” RTAS, 2013.
Main idea: Use profiling of memory accesses of programs and use it to guide cache allocation. Also, combine page coloring with cache locking (use page coloring to map frequently accessed pages to certain cache sets and then lock cache blocks of those cache sets).

[Ward13] B. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, “Making Shared Caches More Predictable on Multicore Platforms,” ECRTS, 2013.
Main idea: Use cache coloring and treat cache sets as a shared resource; that is, a task must clock cache sets before starting to execute; then it can release.

[Wu13] Z. Wu, Y. Krish, and R. Pellizzoni, “Worst-Case Analysis of DRAM Latency in Multi-Requestor Systems,” RTSS, 2013.
Main idea: Model the time it takes for a memory operation to be performed considering DRAM timing parameters. Then use this to compute upper bounds on cumulative delay that a program can experience.

[Suzuki13] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, “Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses,” ICESS, 2013.
Main idea: Setup the virtual-to-physical translation so that different tasks access different cache sets and different memory banks. This provides cache and memory bank isolation.

References

[Kim14] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, R. Rajkumar, “Bounding Memory Interference Delay in COTS-based Multi-Core Systems,” RTAS, 2014.

Main idea: Model the time it takes for a memory operation to be performed considering DRAM timing parameters. Then use this to compute upper bounds on response times. Assume that a task τ_i performs at most H_i memory accesses.This work differs from [Wu13] in that (i) schedulability analysis is performed (not just compute cumulative latency) and (ii) memory bank sharing is allowed.

[Lampka14] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, N. Stoimenov, “A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets,” Real-Time Systems, 2014.

Main idea: Use PREM (that is a program is divided into three parts, fetch data, compute, and write-back result) and partitioned non-preemptive scheduling. Consider the software as consisting of superblocks; a superblock has upper and lower bound on execution time and memory accesses. For each processor core, find a function that is an upper bound on the number of memory accesses in a time interval of duration t. For a processor core under analysis (denoted p), describe the upper bound of the number of memory accesses from other processor cores and let us timed automaton represent the events that memory accesses are generated; this timed automaton must respect the upper bound as mentioned. Then model the bus arbitrator as a timed automaton. And model a superblock as a timed automaton as well. Then state the query that for all possible execution, the response time is at most certain bound. Do binary search on this upper bound. This gives us upper bound on the response time. The paper shows that almost tight bounds can be computed.

[Ye14] Y. Ye, R. West, Z Cheng, and Y. Li, “COLORIS: A Dynamic Cache Partitioning System Using Page Color,” PACT, 2014.

Main idea: Use cache partitioning implemented in software (using the virtual-to-physical translation mechanism) and change the partitioning at run-time (in order to support more tasks and so support changes in the memory footprint).

[Nowotsch14] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, “Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement,” ECRTS, 2014.

Main idea: Use static scheduling (TDMA) to schedule tasks. Assume a round-robin bus. Compute the execution times of tasks.

References

[Yun15a] Heechul Yun,, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha, "Memory Bandwidth Management for Efficient Performance Isolation in Multi-core Platforms," IEEE Transactions on Computers, 2015.
Main idea: Perform policing on the memory bus. The available bandwidth is time-varying because some memory operations are fast (e.g., row hit) and others are slow (e.g., row miss). For soft real-time: reclaim unused memory bandwidth; for hard real-time: disable the reclamation. The sum of bandwidth should be kept below a certain threshold (e.g., 1.2GBps); this is typically much smaller than peak bandwidth (6.4GBps in the system considered in the article).

[Graciolo15] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Frölich, and R. Pellizzoni, “A Survey on Cache Management Mechanisms for Real-Time Embedded Systems,” ACM Computing Surveys, 2015.

[Yun15b] H. Yun, R. Pellizzoni, and P. K. Valsan, “Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems,” ECRTS, 2015.
Main idea: Modify [Kim14] so that the model the analysis is based on allows read-prioritization and multiple outstanding memory requests.

[Yun15c] H. Yun and P. K. Valsan, “Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Processors,” OSPERT, 2015.
Main idea: Evaluate the impact of co-runners on execution times. Do this evaluation on three platforms: ARM7, ARM15, and Intel Nehalem. Find that in some cases the execution time can increase 103 times. Even with cache partitioning, the execution time can increase 14times; this is because of the Miss Status Holding Register (MSHR).

[Panchamukhi15] S.A. Panchamukhi and F. Mueller, “Providing Task Isolation via TLB Coloring,” RTAS, 2015.
Main idea: Use the compiler/linker to allocate code and data of each task so that when the tasks run, TLB entries of one task does not evict TLB entries of another task.

References

[Li16] Y. Li, B. Akesson, K. Lampka, and K. Goossens, “Modeling and Verification of Dynamic Command Scheduling for Real-Time Memory Controllers,” RTAS, 2016.
Main idea: Model many of the details of the memory controller (timing specifications by JEDEC) as a timed automaton. Then describe a network of timed automata and compute the worst-case response time of a task.

[Sha16] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. B. Kegley, D. Perlman, G. Arundale, and R. Bradford, “Real-Time Computing on Multicore Processors,” Computer, 2016.
Main idea: A framework single-core equivalence (SCE) involving (i) cache locking, (ii) bank coloring, and (iii) memory guard (policing the memory accesses). The memory guard makes the execution time of one task independent of the memory bus contention of other task but it comes at the cost of low memory bandwidth (1Gbps). SCE uses an I/O partition. SCE assumes that the h/w supports cache locking and performance monitoring counters. With SCE, the execution time of a task can increase by approximately 50% (see Figure 5) for 8 cores.

[Kim16] N. Kim, B. C. Ward, M. Chisholm, C.-Y. Fu, J.H. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” RTAS, 2016.
Main idea: Use isolation mechanisms for high-criticality tasks and let low-criticality tasks share resources.

[Kim16] N. Kim, B. C. Ward, M. Chisholm, C.-Y. Fu, J.H. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” RTAS, 2016.
Main idea: Use isolation mechanisms for high-criticality tasks and let low-criticality tasks share resources.

[CAST32A] Certification Authorities Software Team (CAST), Position Paper, CAST-32A, Multi-core Processors, COMPLETED November 2016 (Rev 0), Available at https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/

[Sha16b] L. Sha, M. Caccamo, G. Shelton, M. Nuessen, J. P. Smith, D. Miller, R. Bradford, R. Kegley, D. Perlman, J. Preston, J. W. Wlad, M. Storr, D. DeNiz, S. Chaki, M. Klein, B. Andersson, I. Bate, A. Burns, S. Palin, S. Bak, D. Kingston, M. Clark, T. Kim, and E. Pak, “Position Paper on Minimal Multicore Avionics Certification Guidance,” August 4, 2016.