Adding Cache and Memory Management to the MC^2 (Mixed Criticality on Multicore) Framework

Jim Anderson

University of North Carolina at Chapel Hill

Outline

- Motivation for MC^2.
- Basic MC^2 design.
  - » Per-level scheduling and schedulability guarantees.
- MC^2 with shared hardware management.
  - » Cache and memory bank partitioning (and sharing).
  - » Isolating the OS.
  - » Complexities such as shared libraries.
- Future research directions.

Original Driving Problem

Joint Work with Northrop Grumman Corp.

- Goal of this work:
  - » To practically resolve the **"one out of m" multicore problem**, especially w.r.t. avionics:
    - When using an m-core platform in a safety-critical domain, analysis pessimism can be so great, the capacity of the "additional" m – 1 is entirely negated.
  - » We are attempting to combine two approaches:
    - Using mixed-criticality analysis that enables less critical components to be provisioned less pessimistically.
    - Managing hardware resources, as appropriate.

Image source: http://www.as.northropgrumman.com/products/nucasx47b/assets/lgm_UCAS_3_0911.jpg

Starting Assumptions

- Modest core count (e.g., 2-8).
  - » Quad-core in avionics would be a tremendous innovation.

Starting Assumptions

- Modest core count (e.g., 2-8).
- Modest number of criticality levels (e.g., 2-5).
  - » 2 may be too constraining
  - » ∞ isn't practically interesting.
  - » These levels may not necessarily match DO-178B/C.

Starting Assumptions

- Modest core count (e.g., 2-8).
- Modest number of criticality levels (e.g., 2-5).
- Statically prioritize criticality levels.
  - » Schemes that dynamically mix levels may be problematic in practice.
    - **Note:** This is done in much theoretical work on mixed criticality scheduling.
  - » Also, practitioners tend to favor simple resource sharing schemes.

Starting Assumptions

- Modest core count (e.g., 2-8).
- Modest number of criticality levels (e.g., 2-5).
- Statically prioritize criticality levels.

Main motivation: To develop a framework that allows interesting design tradeoffs to be investigated that is reasonably plausible from an avionics point of view.

A Non-Goal: Developing a framework that could really be used in avionics today.

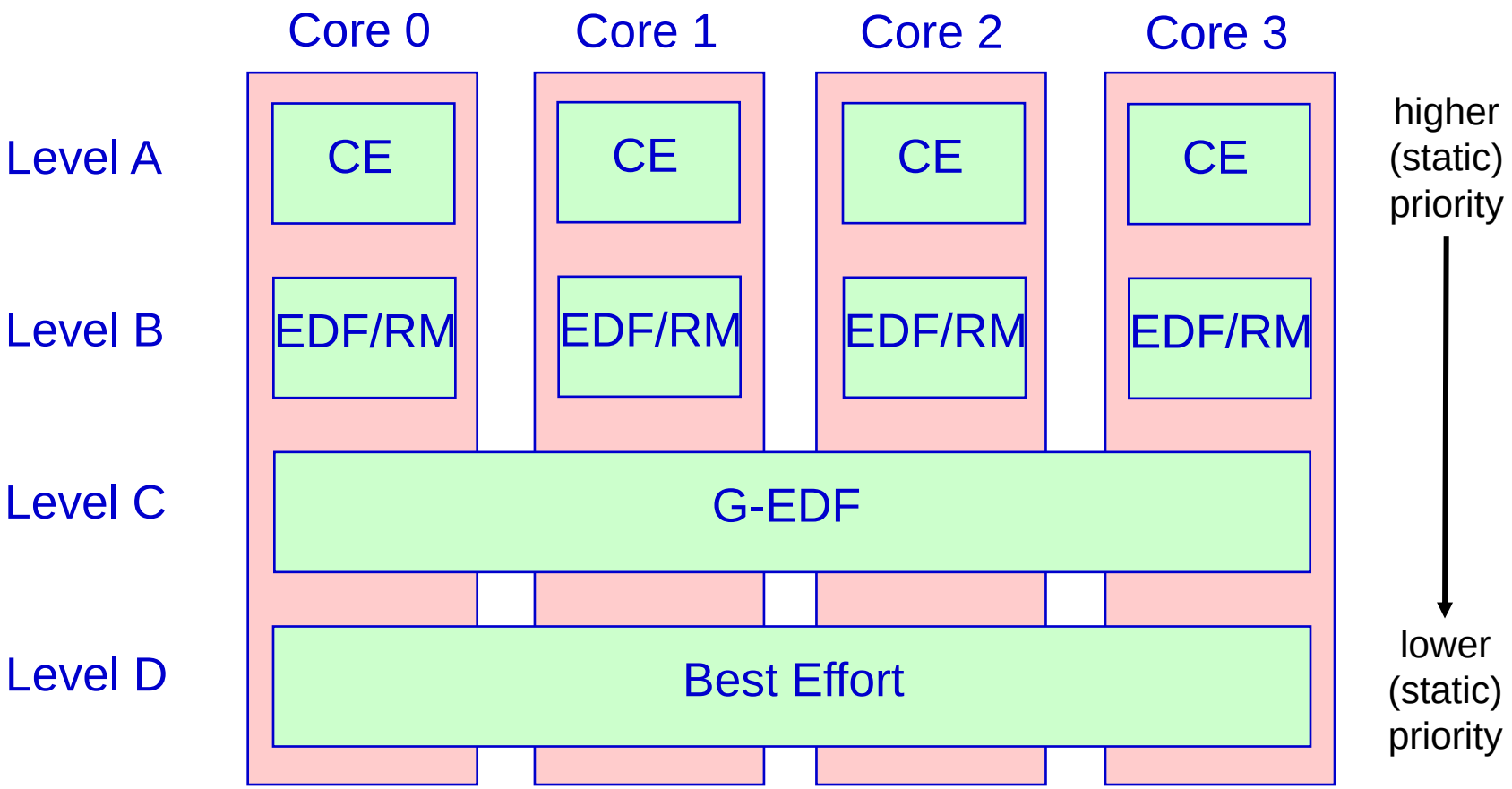
Outline

- Motivation for MC^2.
- Basic MC^2 design.
  - » Per-level scheduling and schedulability guarantees.
- MC^2 with shared hardware management.
  - » Cache and memory bank partitioning (and sharing).
  - » Isolating the OS.
  - » Complexities such as shared libraries.
- Future research directions.

Basic MC^2 Design

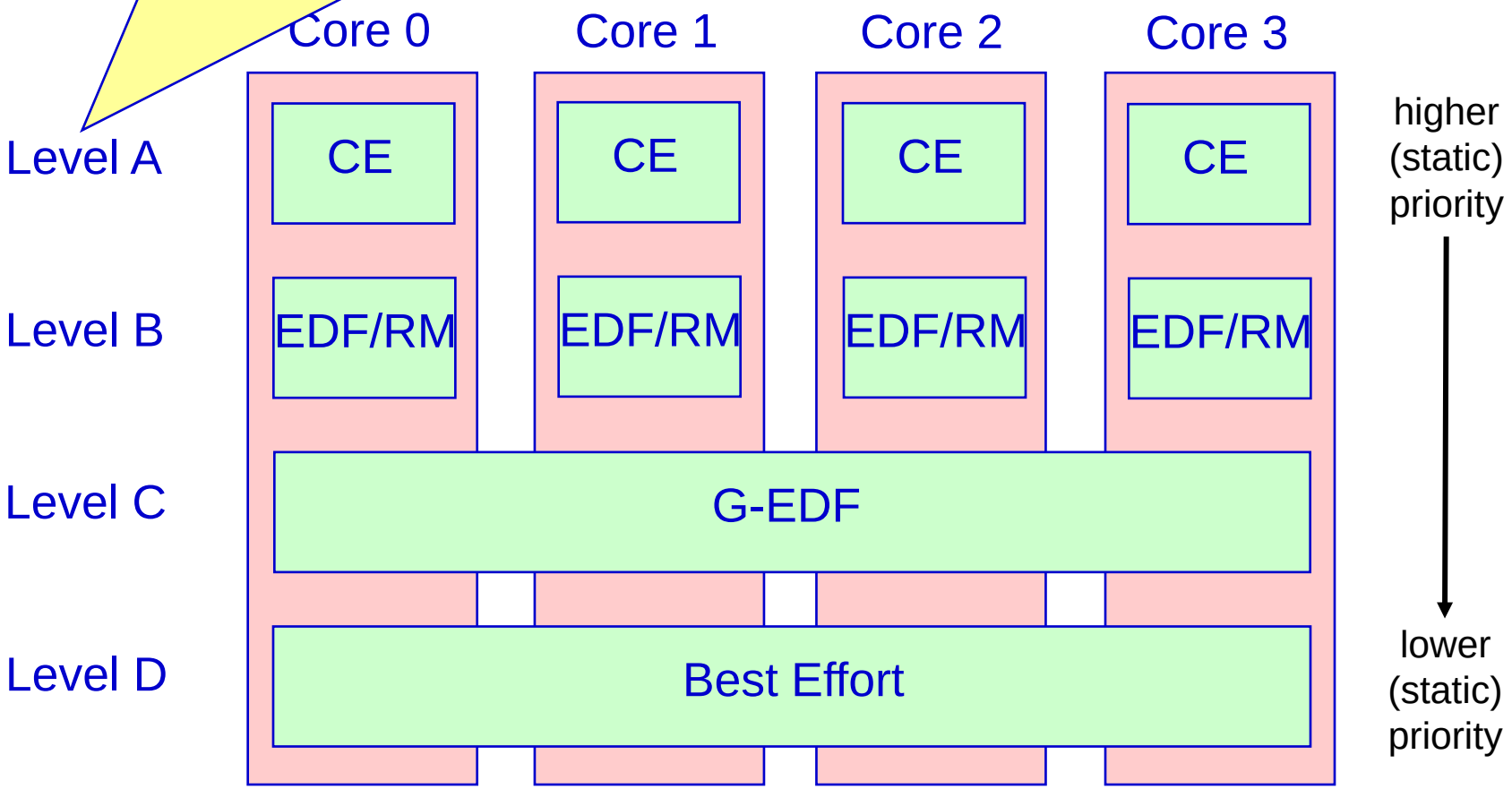
- We assume four criticality levels, A-D.
  - » Originally, we assumed five, like in DO-178B/C.
- We statically prioritize higher levels over lower ones.
- We assume:
  - » Levels A & B require HRT guarantees.
  - » Level C requires SRT guarantees in the form of bounded deadline tardiness.
  - » Level D is non-RT.
  - » All tasks are implicit-deadline periodic/sporadic.

MC^2 Architecture

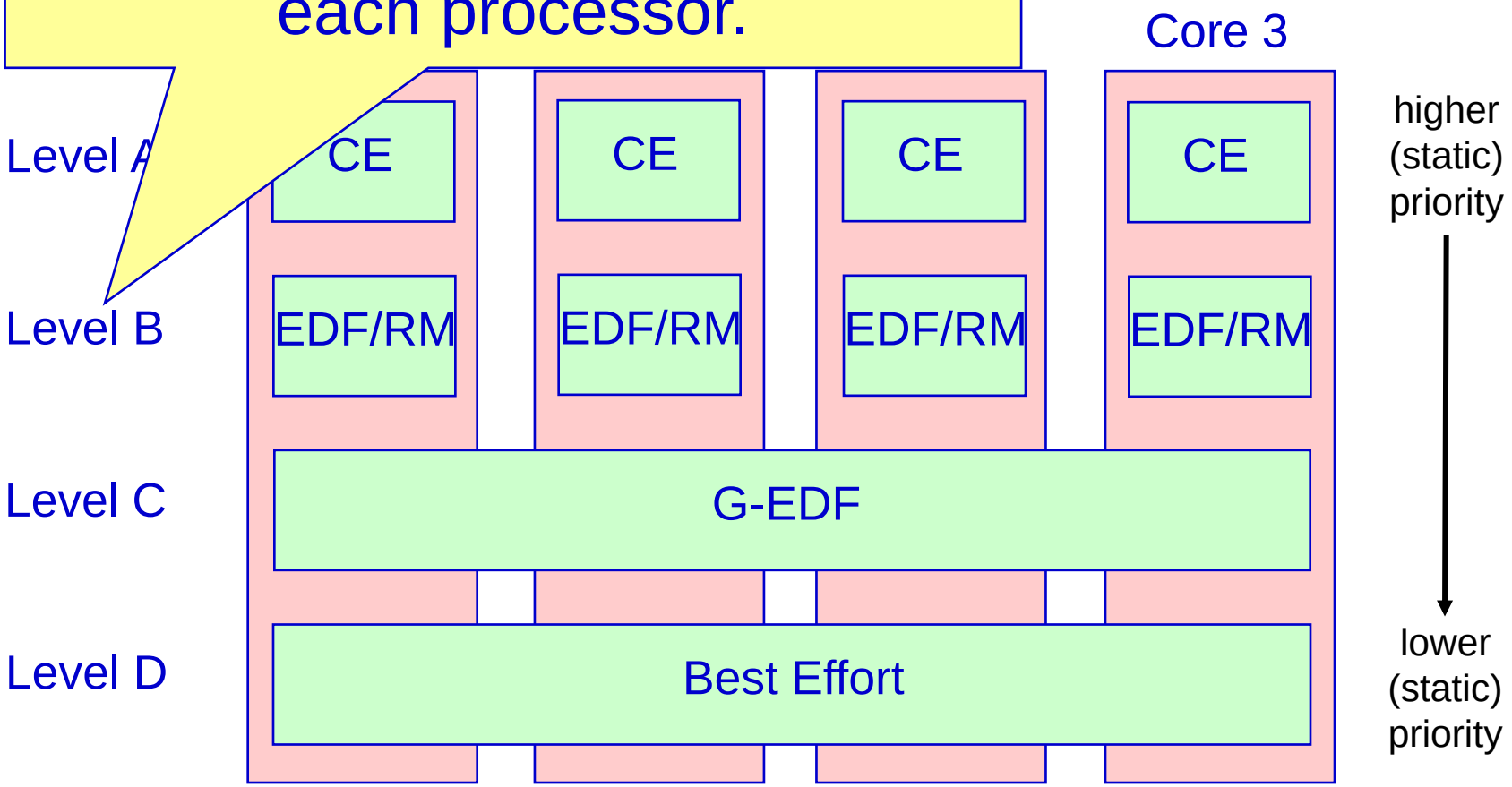


Level A: Partitioned scheduling. Time-triggered Cyclic Executive scheduler on each processor.

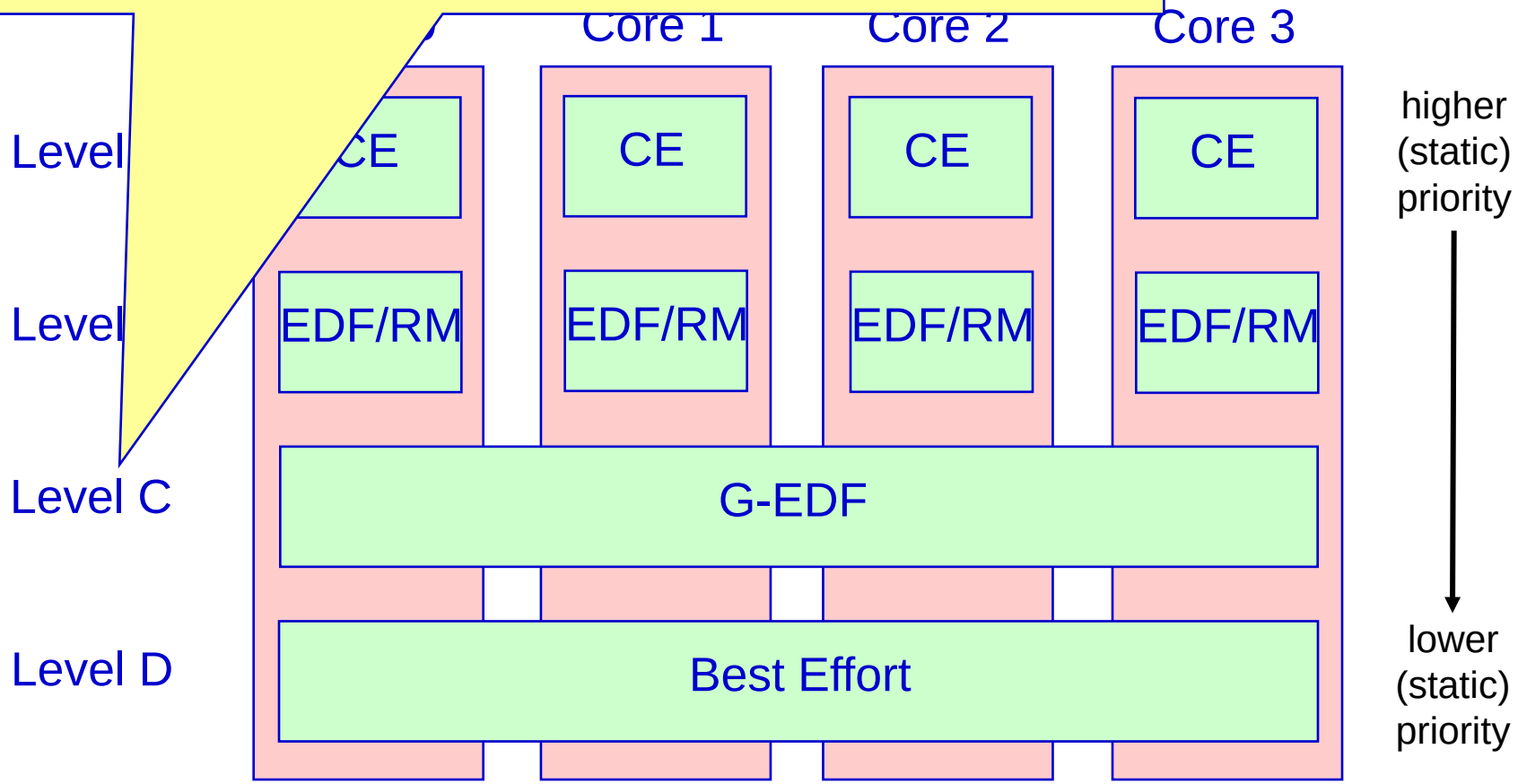
re



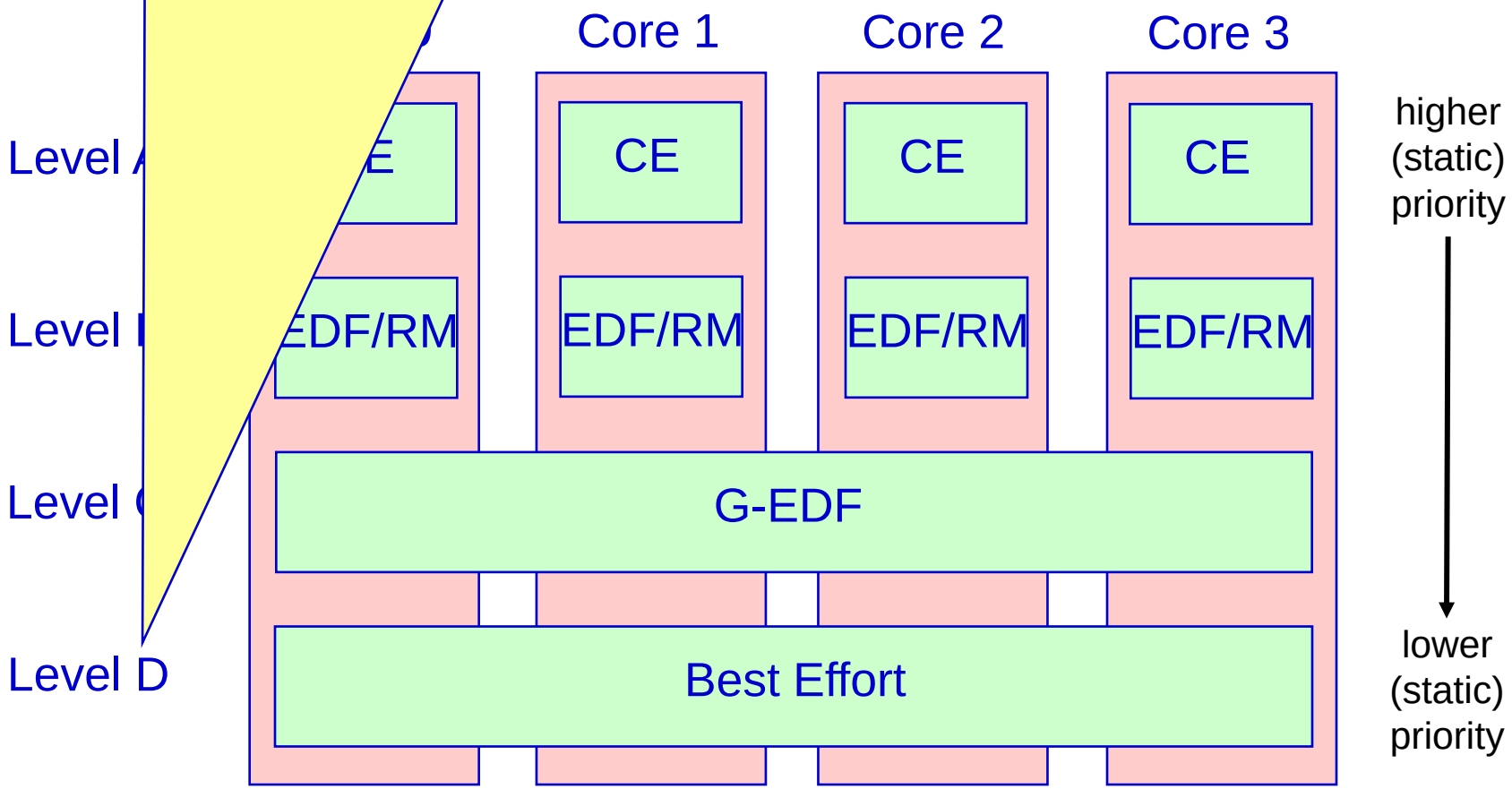
Level B: Partitioned scheduling. Either Earliest-Deadline-First or Rate-Monotonic scheduler on each processor.



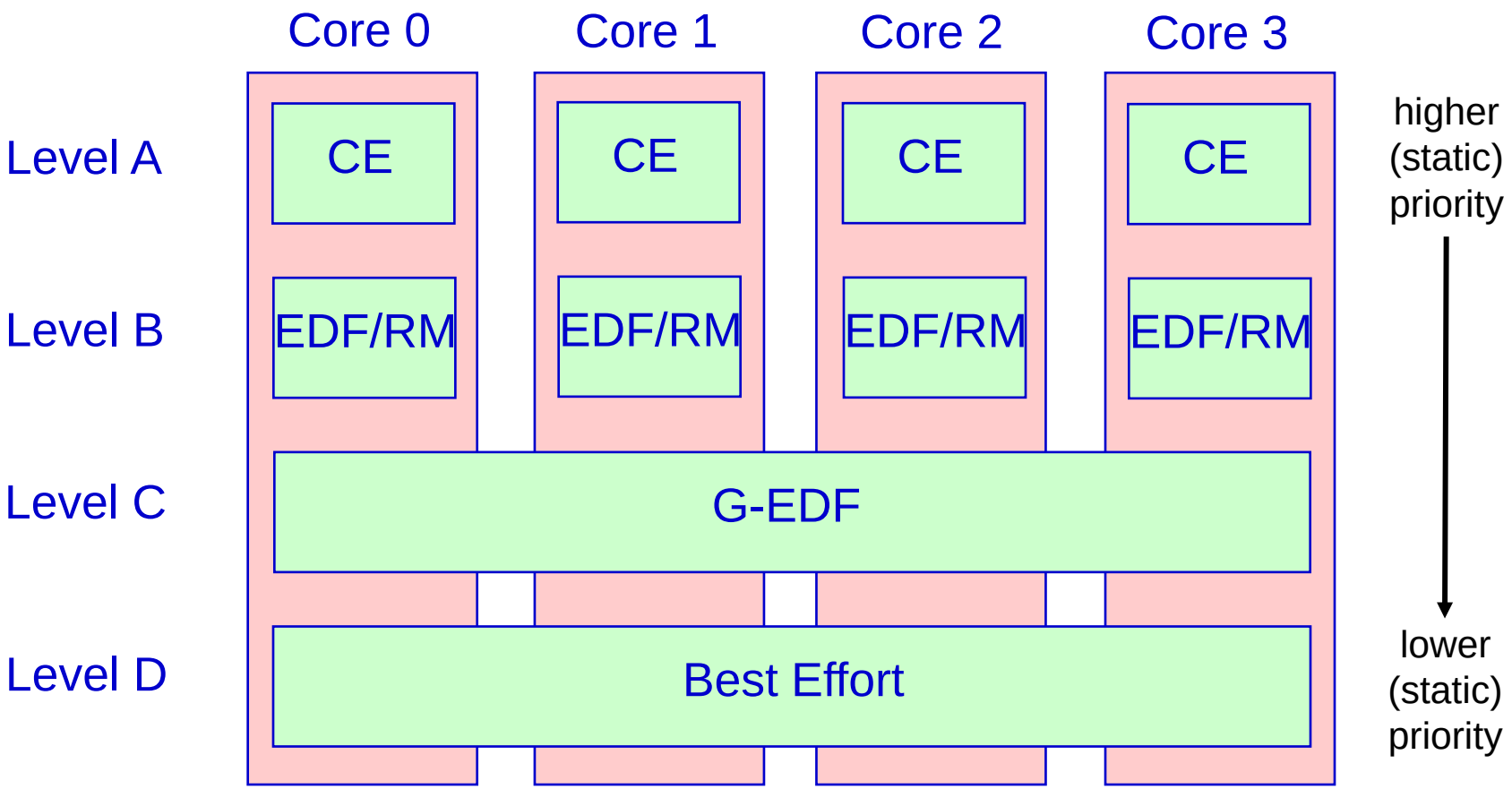
Level C: Global scheduling using either Earliest-Deadline-First or some other "EDF-like" scheduler.



Level D: Global background scheduling.



MC^2 Architecture



Rationale

- Experimental research at UNC has shown that partitioned schedulers are best for HRT and global schedulers are best for SRT.
- This design enables many interesting tradeoffs to be explored in a setting with several criticality levels (not just two):
  - » Table-driven vs. priority scheduling.
  - » Partitioned vs. global scheduling.
  - » HRT vs. SRT.

Nuances

- Can either enforce execution budgets at each level or not.
- Slack shifting can be used to reallocate unused processing time.
- Level C can be provisioned either on a worst- or average-case basis.
  - » So, response times can be bounded either in the worst case or in expectation.
- Overload can be dealt with at Level C by scheduling in a virtual time domain.

Main Limitations

As of Now

- Haven’t yet considered support for
  - » synchronization (either critical sections or precedence constraints), or
  - » dynamic workload changes.

Checking Schedulability

Back to the "One of out m" Problem…

- We use mixed-criticality schedulability analysis as proposed by Vestal [RTSS '07].
- Each task has an execution cost specified at each criticality level (A-D).
  - » Costs at higher levels are (typically) larger.
- Example:

  $T.e_A = 20$, $T.e_B = 12$, $T.e_C = 5$, …
- Rationale: Will use more pessimistic analysis at high levels, more optimistic at low levels.

Checking Schedulability

Back to the "One of out m" Problem…

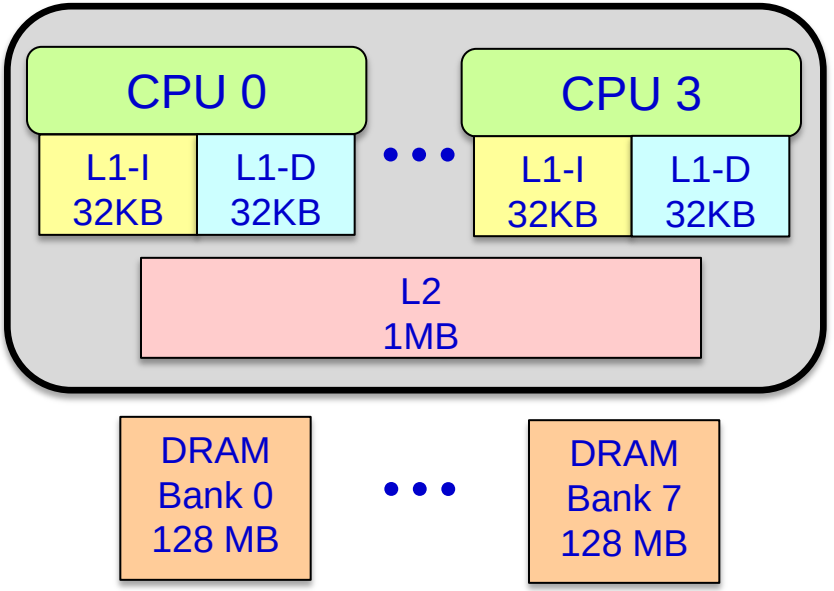
- We use mixed-criticality schedulability
- Each task has multiple execution costs (one per criticality level).
  - » Costs at higher Levels are (typically) larger.
- The task system is *correct at Level X* iff all Level-X tasks meet their timing requirements assuming all tasks have Level-X execution costs.

Some "weirdness" here: Not just one system anymore, but four: the Level-A system, Level-B,…

Outline

- Motivation for MC^2.
- Basic MC^2 design.
  - » Per-level scheduling and schedulability guarantees.
- MC^2 with shared hardware management.
  - » Cache and memory bank partitioning (and sharing).
  - » Isolating the OS.
  - » Complexities such as shared libraries.
- Future research directions.

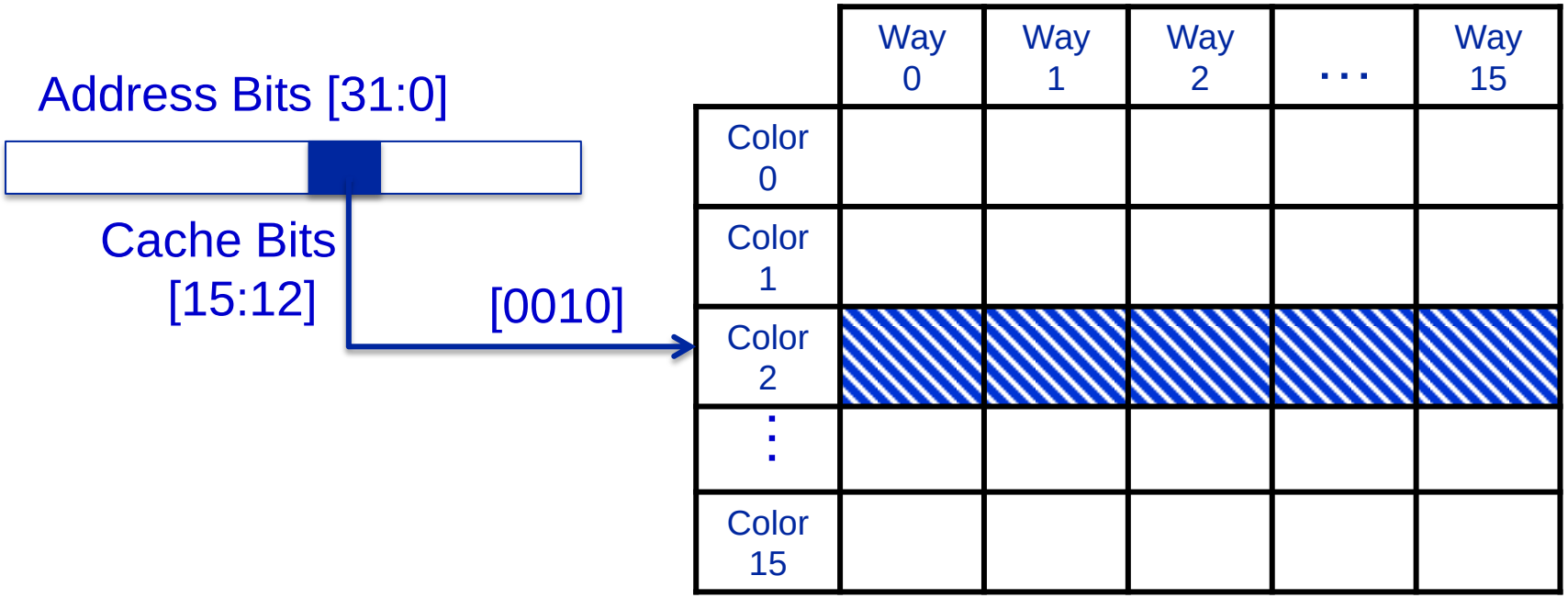
Hardware Platform



- Freescale i.MX 6Quad 1 GHz ARM®Cortex™-A9 processor.
- Caches:
  - » 32 KB L1 I-cache per core.
  - » 32 KB L1 D-cache per core.
  - » 1 MB shared L2 cache.
    - – Cache line size =32 B, 2048 Sets, 16 Ways.
- 1 GB DDR3 SDRAM up to 533 MHz memory.
  - » 8 Banks, each 128 MB.

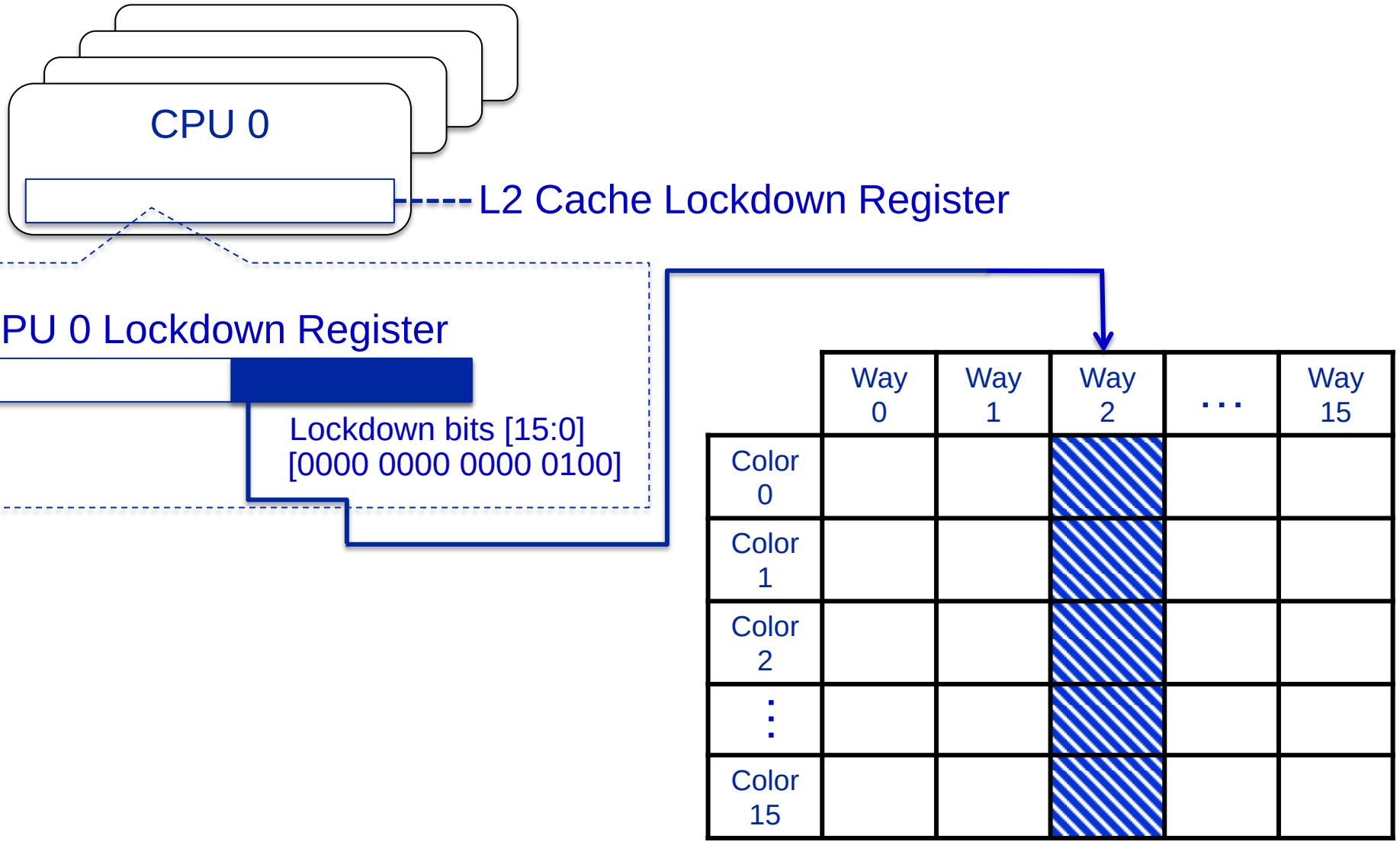
Cache Partitioning (of the Shared L2)

Option 1: Set Partitioning, i.e., Page Coloring

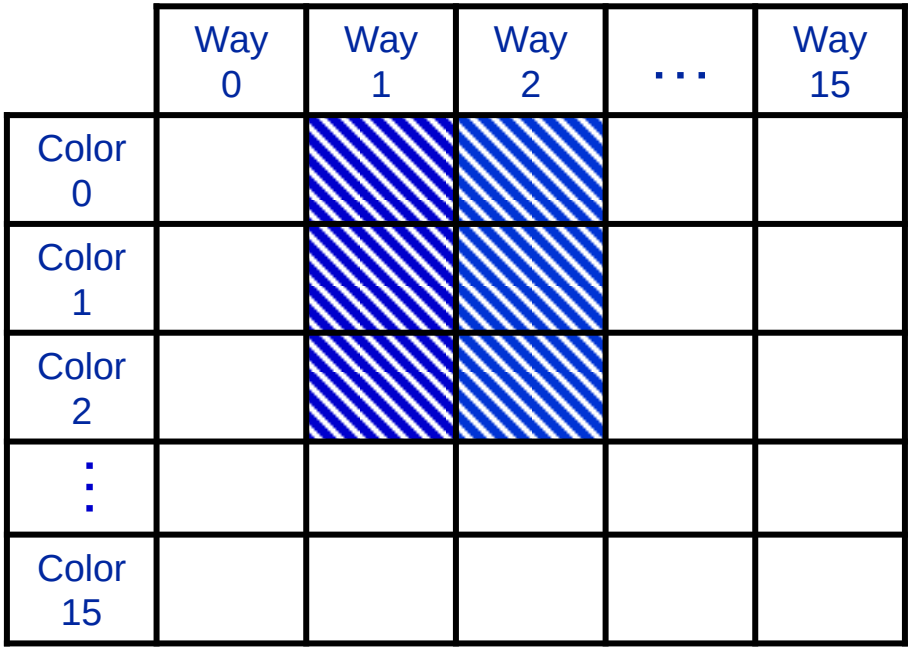


Cache Partitioning

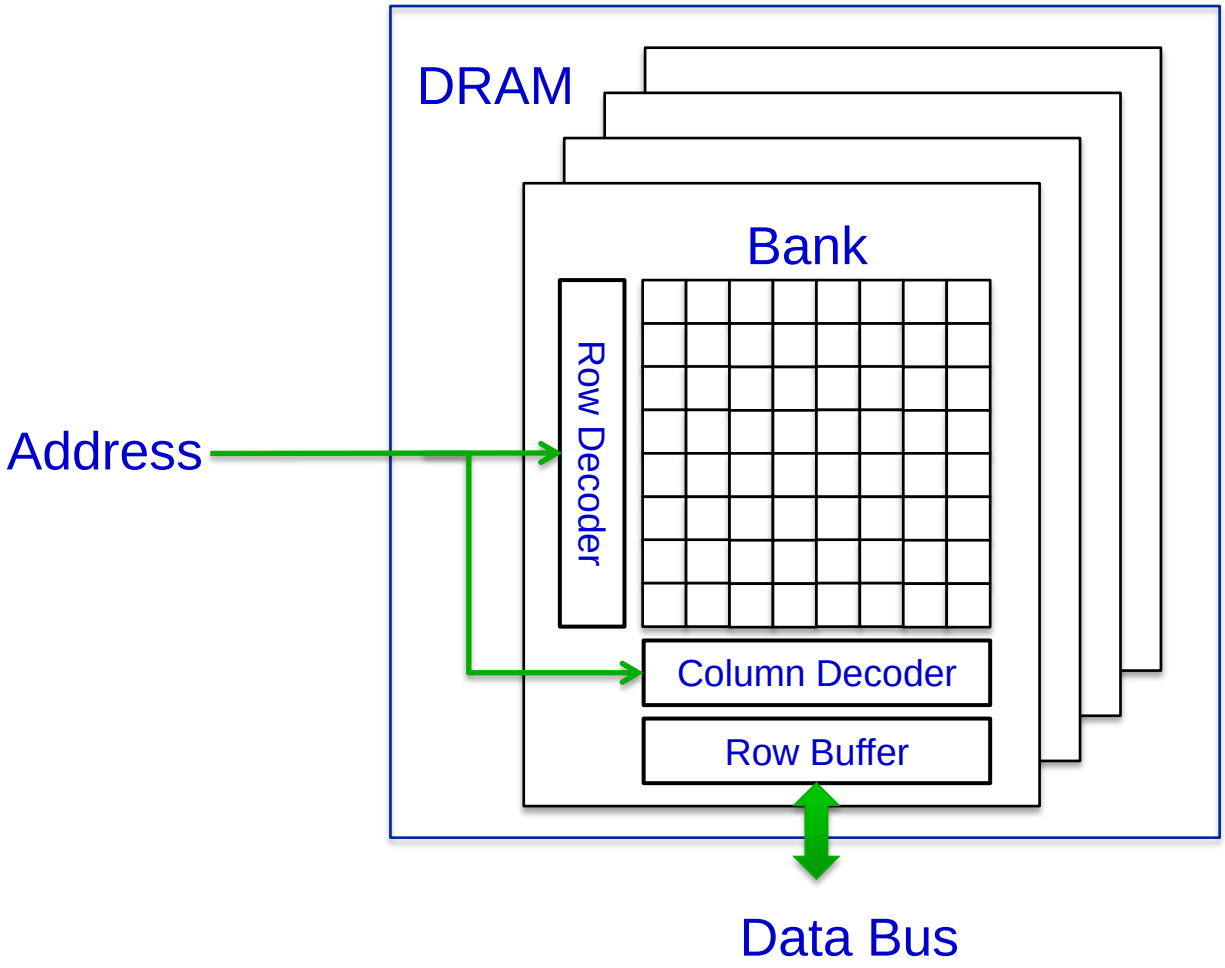
Option 2: Way Partitioning



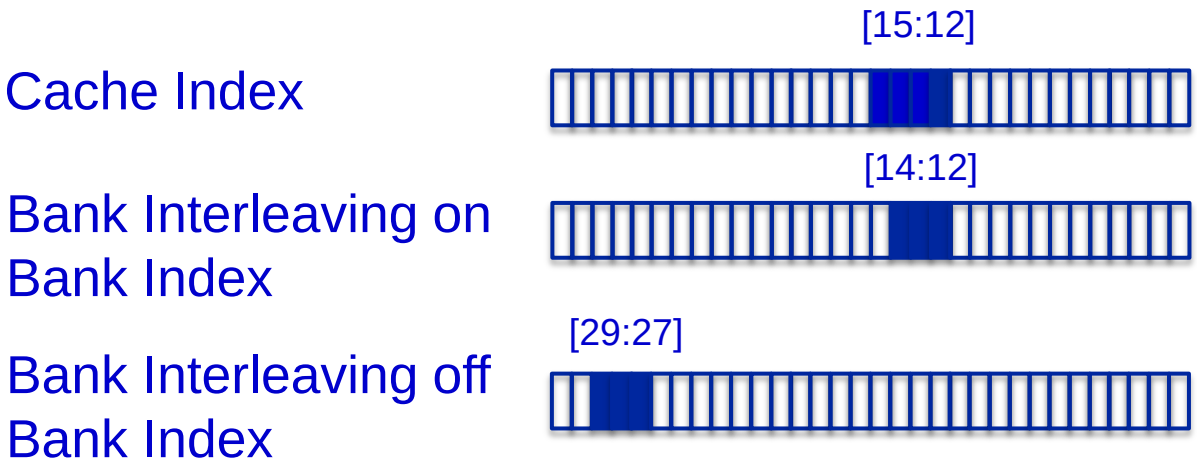
Can Combine these Approaches



DRAM Banks

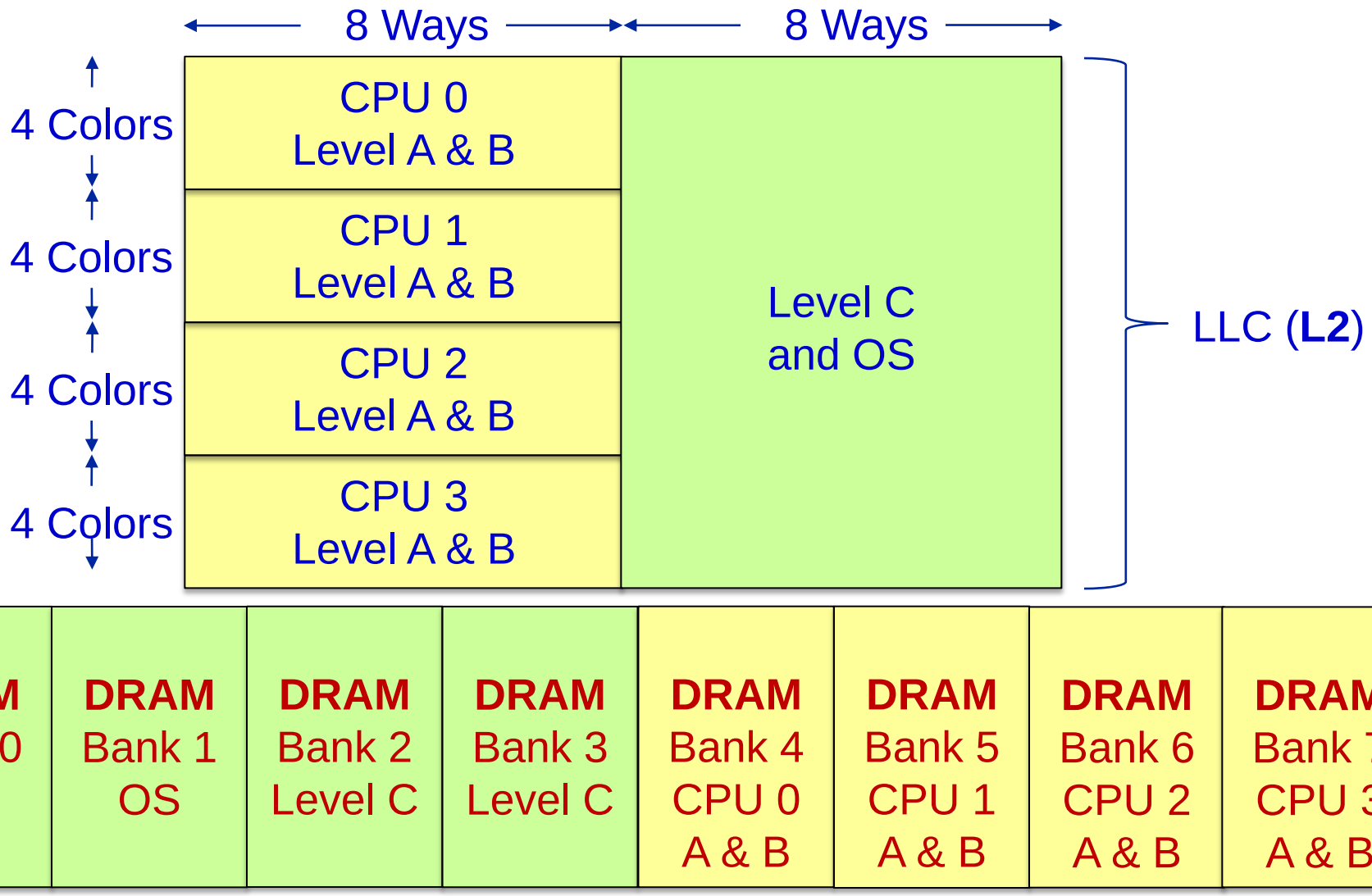

DRAM
Bank
Row Decoder
Address
Column Decoder
Row Buffer
Data Bus

Address Decoding



Current Prototype: Turn interleaving off. Give dedicated banks to Levels A and B. Let Level C share banks with statically allocated OS code and data.

Overall Hardware Allocation Strategy



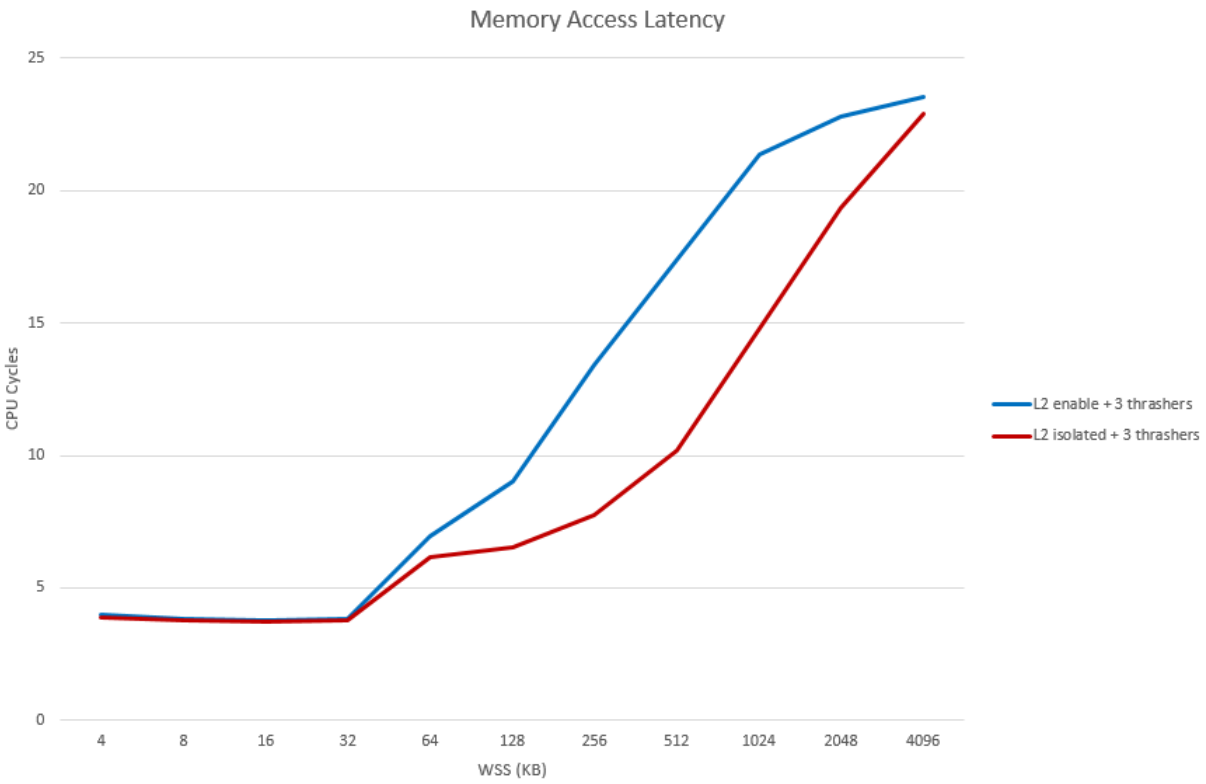
Current Implementation Status

MC^2 is Implemented as a $LITMUS^{RT}$ Plugin

- We support both set- and way-based partitioning and DRAM partitioning.
  - » For set-based, we re-color (almost) all task pages before task execution.
  - » Current limitations:
    - – Each task has a signal handling page (which should rarely be accessed) that is not colored.
    - – We don't color shared pages (but can handle shared libraries through static linking).
    - – OS is isolated w.r.t. DRAM except for dynamically allocated pages.
    - – We are ignoring Level D for now.

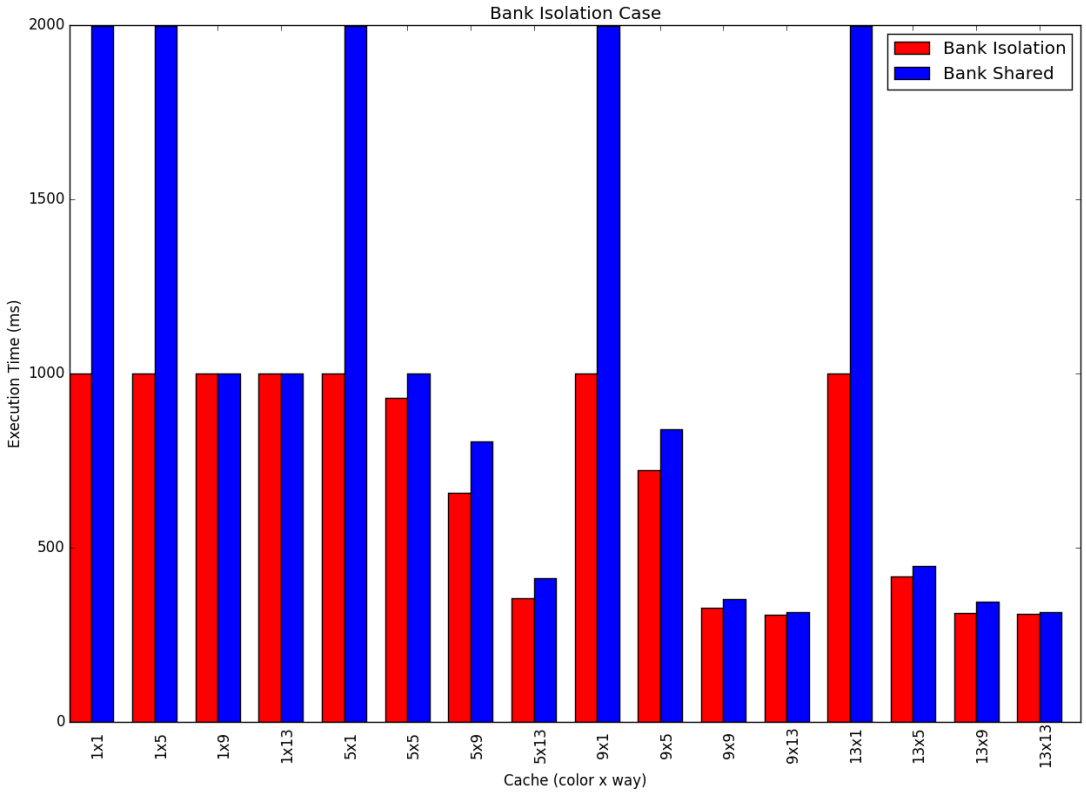
Importance of Controlling L2 Interference

Measured memory access latency of a synthetic task on a loaded system, with **(RED)** and without **(BLUE)** L2 isolation, as a function of working set size.



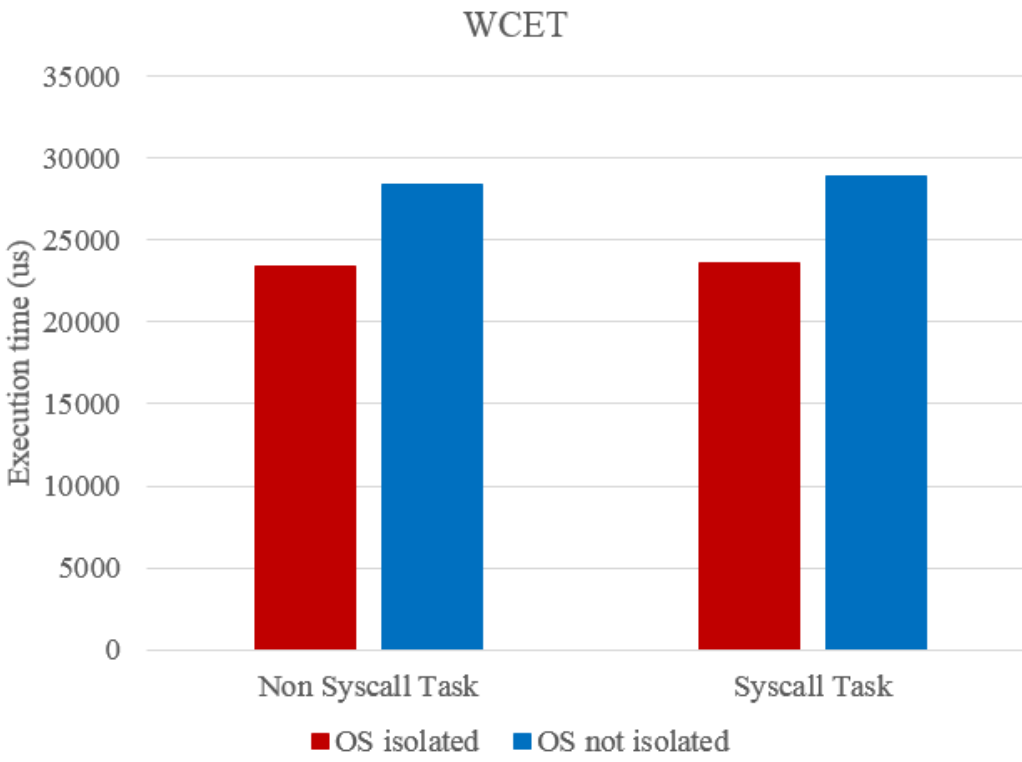
Importance of Controlling DRAM Bank Interference

Measured worst-case execution time of a synthetic task on a loaded system, with **(RED)** and without **(BLUE)** bank isolation, as a function of the size (number of ways and colors) of the allocated L2 area. Bank isolation really matters if working set doesn't fit within the L2.



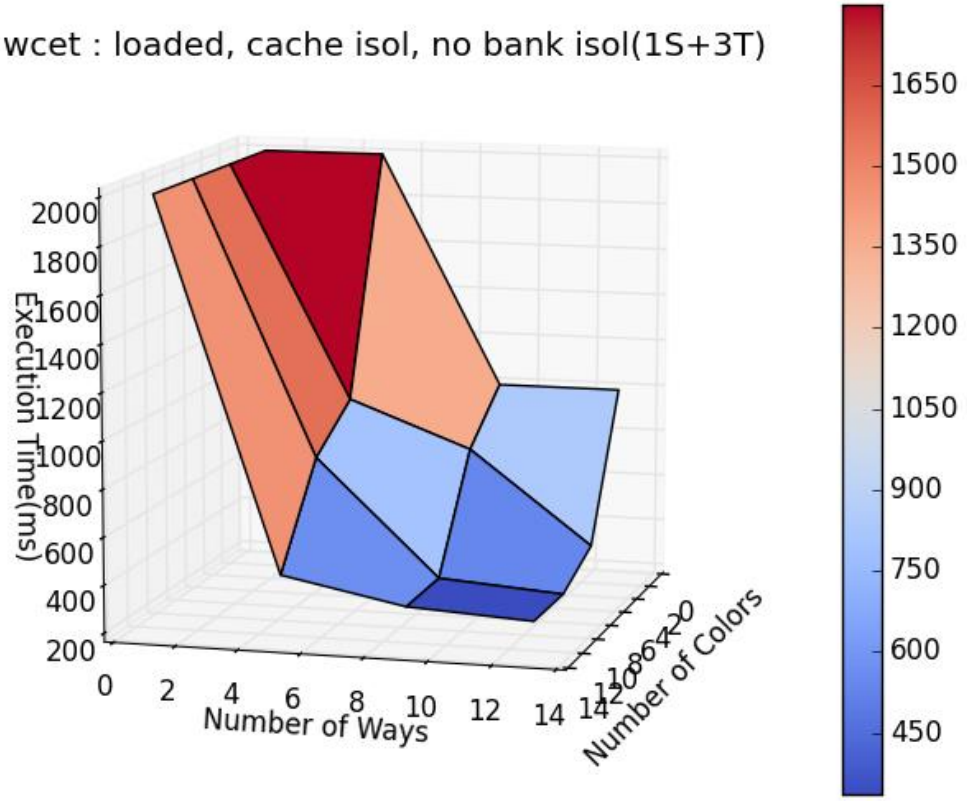
Importance of Controlling OS Interference

Measured worst-case execution times of two synthetic tasks, with **(RED)** and without **(BLUE)** OS isolation, where one task performs repeated system calls and the other doesn't.



Interesting Tradeoffs Exist w.r.t. Allocating L2 Areas

Measured worst-case execution time of a synthetic task on a loaded system, with L2 isolation, as a function of the size (number of ways and colors) of the allocated L2 area.



Major Principles

- Solving the "one out of m" problem requires:
  - » Provisioning less pessimistically where appropriate.
  - » Enabling hardware isolation, but only where needed and where possible.
    - – Lower criticality tasks might actually benefit from sharing.
    - – It's OK if some hardware resources are not managed, as long as interferences due to such resources are accounted for in analysis.

Outline

- Motivation for MC^2.
- Basic MC^2 design.
  - » Per-level scheduling and schedulability guarantees.
- MC^2 with shared hardware management.
  - » Cache and memory bank partitioning (and sharing).
  - » Isolating the OS.
  - » Complexities such as shared libraries.
- Future research directions.

Future Work

- Our future plans include:
  - » Devising (near) optimal algorithms for allocating L2 areas.
    - Such algorithms must account for different requirements at different criticality levels.
  - » Determining whether we can fully isolate the OS (even w.r.t. dynamically allocated DRAM data).
    - May potentially integrate with PALLOC [Yun et al., RTAS '14].
  - » Enabling dynamic adaptations and synchronization.
  - » Extending page coloring to fully deal with shared pages.

MC^2 Papers

(All papers available at http://www.cs.unc.edu/~anderson/papers.html)

- J. Anderson, S. Baruah, and B. Brandenburg, "Multicore Operating-System Support for Mixed Criticality," *Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.
  - » A **"precursor" paper** that discusses some of the design decisions underlying MC^2.
- M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos, "Mixed Criticality Real-Time Scheduling for Multicore Systems," *Proc. of the 7th IEEE International Conf. on Embedded Software and Systems*, 2010.
  - » Focus is on **schedulability**, i.e., how to check timing constraints at each level and "shift" slack.
- J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson, "RTOS Support for Multicore Mixed-Criticality Systems," *Proc. of the 18th RTAS,* 2012.
  - » Focus is on **RTOS design**, i.e., how to reduce the impact of RTOS-related overheads on high-criticality tasks due to low-criticality tasks.
- B. Ward, J. Herman, C. Kenna, and J. Anderson, "Making Shared Caches More Predictable on Multicore Platforms," *Proc. of the 25th ECRTS*, 2013.
  - » Adds **shared cache management** to a two-level variant of MC^2. The approach in today's talk is different.
- J. Erickson, N. Kim, and J. Anderson, "Recovering from Overload in Multicore Mixed-Criticality Systems," *Proc. of the 29th IPDPS*, 2015.
  - » Adds **virtual-time-based scheduling** to Level C.

Thanks!

- Questions?