Having Fun Experimenting with Hardware Management and Mixed Criticality on Multicore

Jim Anderson

University of North Carolina at Chapel Hill

Outline

- Driving problem.
- Prior work: The MC^2 (mixed-criticality on multicore) framework.
- New work: MC^2 with shared hardware management.
- Future work.
- Running a large schedulability study.

This has led to the common practice of simply disabling all but one core in avionics systems if highly critical system components exist.

The FAA position paper "CAST 32" discusses problems associated with multicore platforms in this domain.

- C

"one out of m" mult problem, especially avionics:

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

– When using an m-core platform in a safety-critical domain, analysis pessimism can be so great, the capacity of the "additional" m – 1 cores 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.

What is Mixed-Criticality Analysis?

(Vestal [RTSS '07])

- Each task is assigned a criticality level.
- Each task has provisioned execution time (PET) specified at each criticality level.
  - » PETs at higher levels are (typically) larger.
- **Example:** Assuming criticality levels A (highest), B, C, etc., task τ_i might have PETs $C_i^A = 20$, $C_i^B = 12$, $C_i^C = 5$, …
- **Rationale:** Will use more pessimistic analysis at high levels, more optimistic at low levels.

What is Mixed-Criticality Analysis?

(Vestal [RTSS '07])

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

  » PETs at higher levels (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 PETs.

Outline

- Driving problem.
- Prior work: The MC^2 (mixed-criticality on multicore) framework.
- New work: MC^2 with shared hardware management.
- Future work.
- Running a large schedulability study.

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).

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

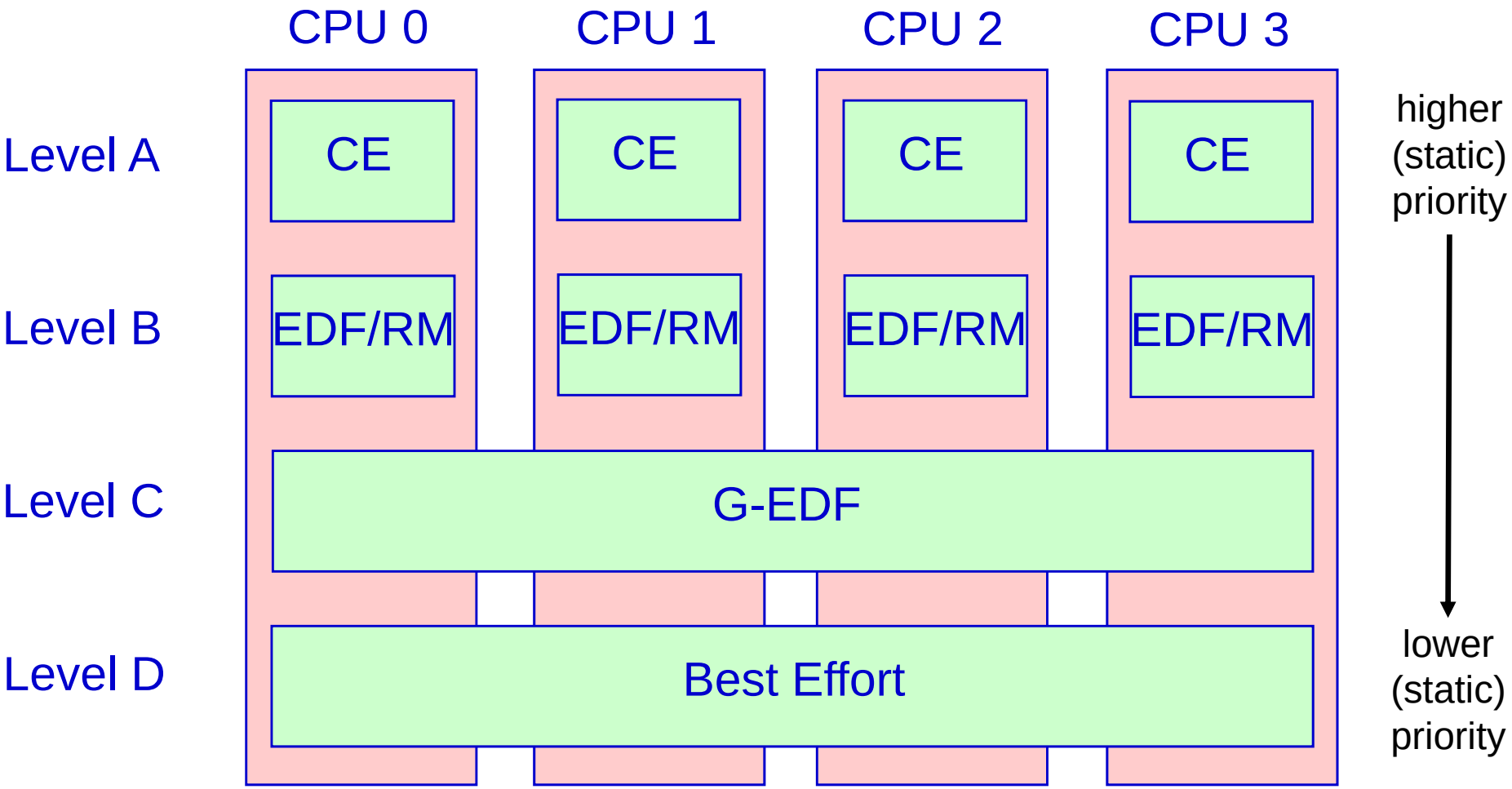
A Non-Goal: Developing a framework that could really be used in avionics today.

Basic MC^2 Design

- We assume four criticality levels, A-D.
  - » Originally, we assumed five, like in DO-178B/C.
  - » Levels A & B are hard real-time (HRT).
  - » Level C is soft real-time (SRT) and requires bounded deadline tardiness.
  - » Level D is non-RT.
  - » All tasks are periodic/sporadic.

MC^2 Architecture

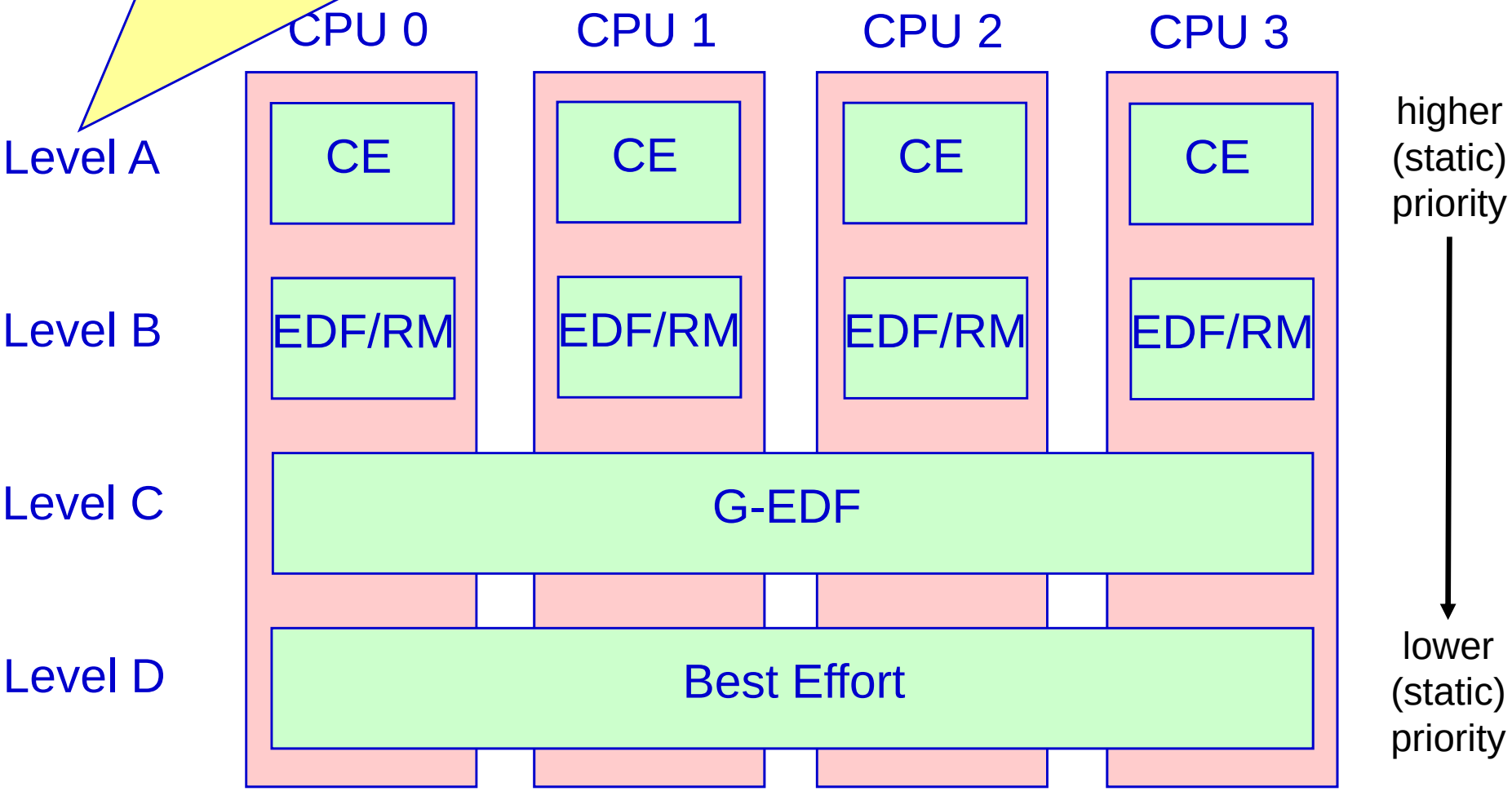
Implemented as a LITMUS^{RT} Plugin



re

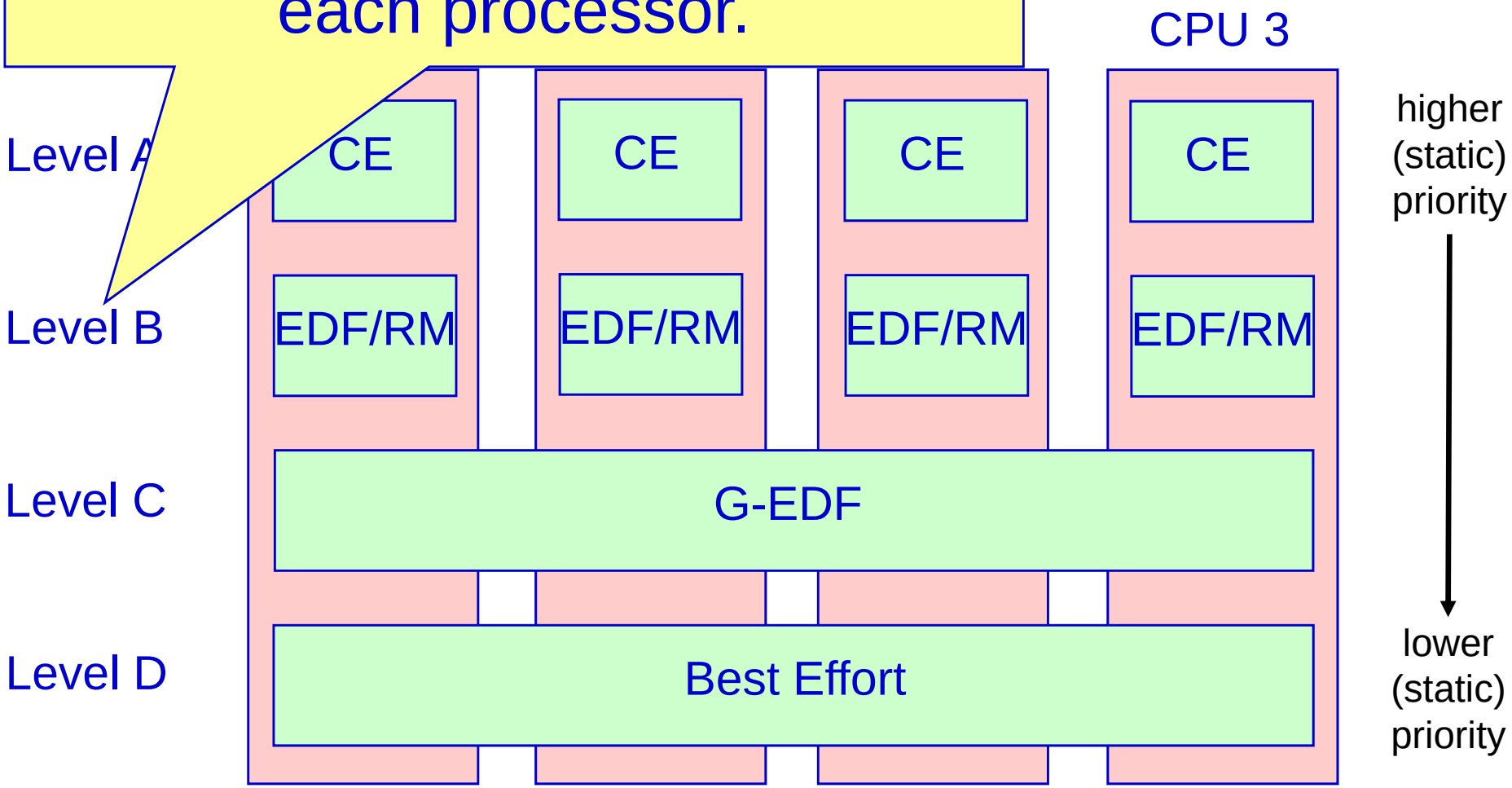
T Plugin

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

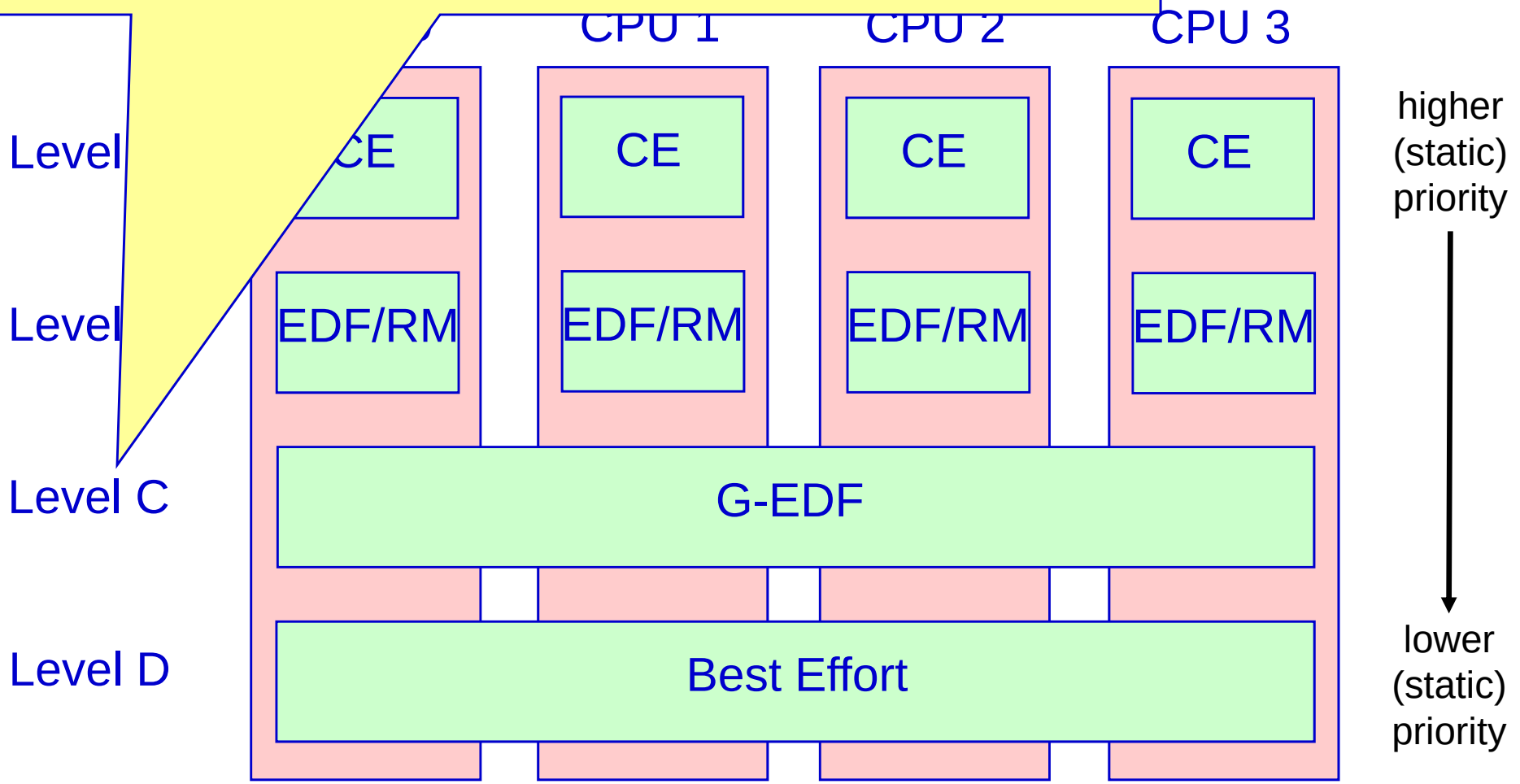


e Plugin

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

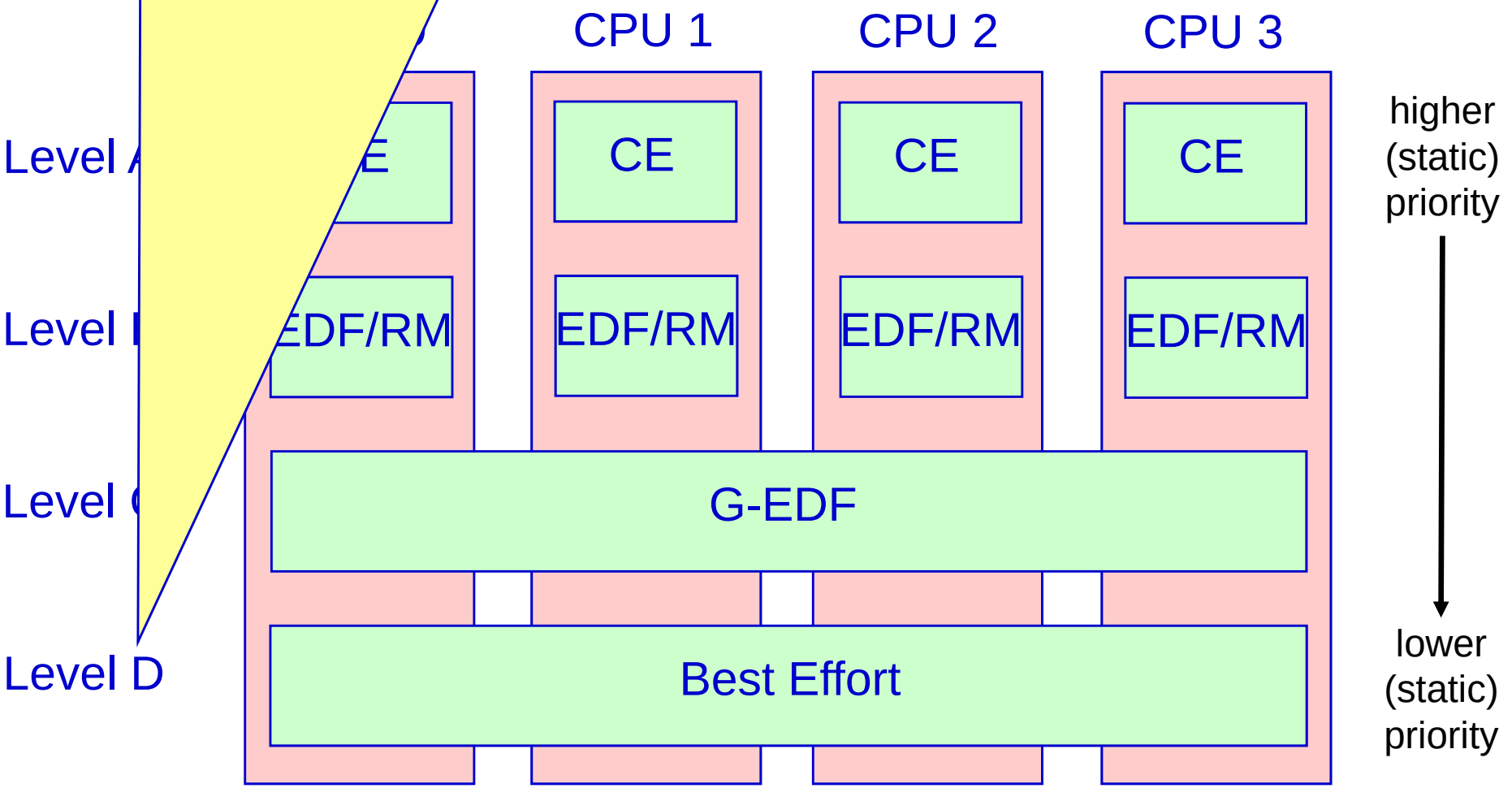


Level C: Global scheduling using either Earliest-Deadline-First or some other "EDF-like" scheduler.



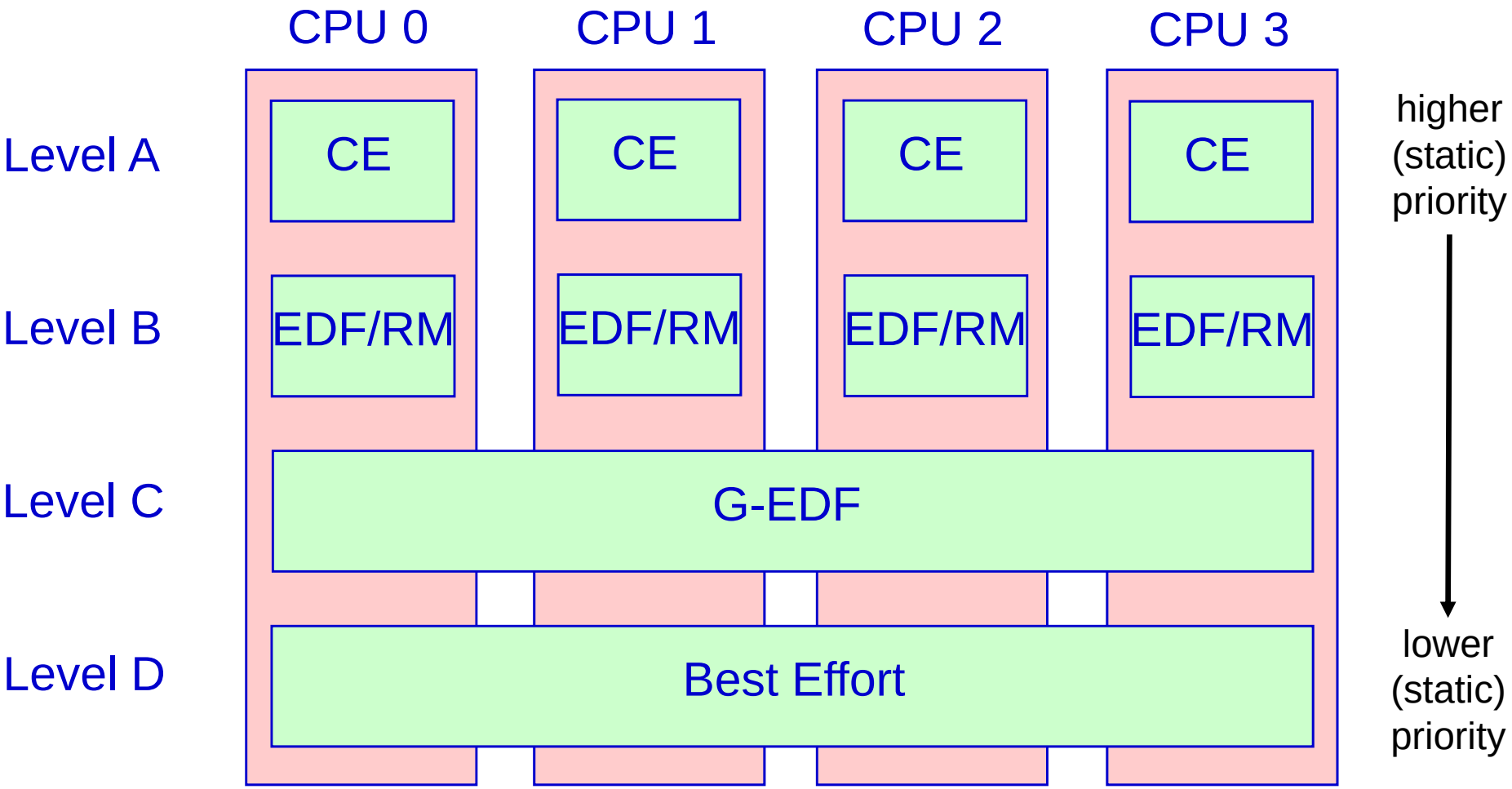
MC^2 Architecture

Level D: Global background scheduling.



MC^2 Architecture

Implemented as a LITMUS^{RT} Plugin



Rationale

- Experimental research at UNC has shown
  - » 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.

Outline

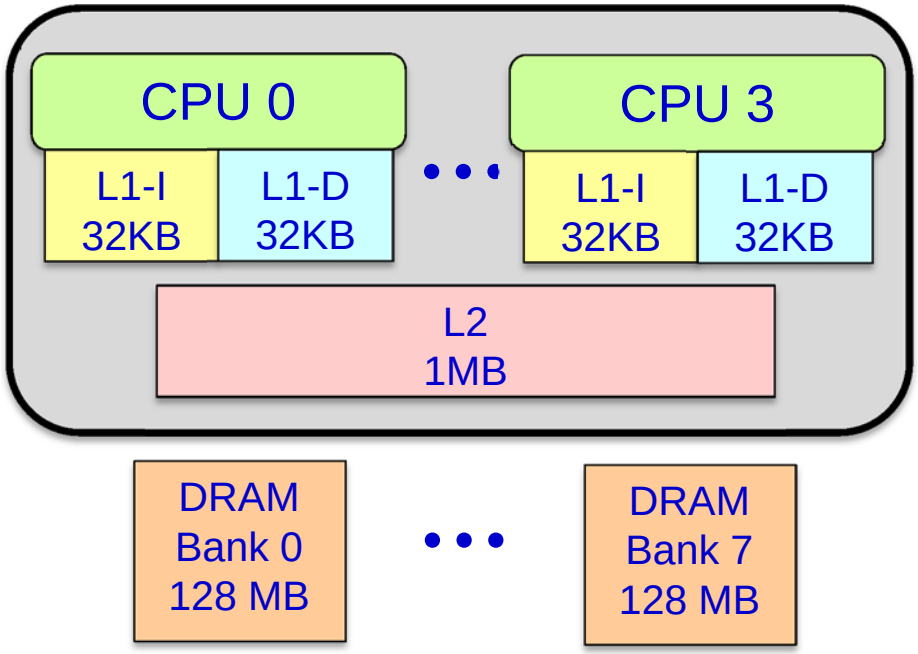
- Driving problem.
- Prior work: The MC^2 (mixed-criticality on multicore) framework.
- New work: MC^2 with shared hardware management.
- Future work.
- Running a large schedulability study.

Managing Shared Hardware in MC^2

- This year, we added support to MC^2 for managing shared caches and DRAM memory banks.
  - » **Goal:** Enable higher criticality tasks to be isolated from lower criticality ones w.r.t. these resources.
  - » **Why?**
    - – This lessens hardware interference and
    - – enables smaller and tighter task execution-cost estimates.

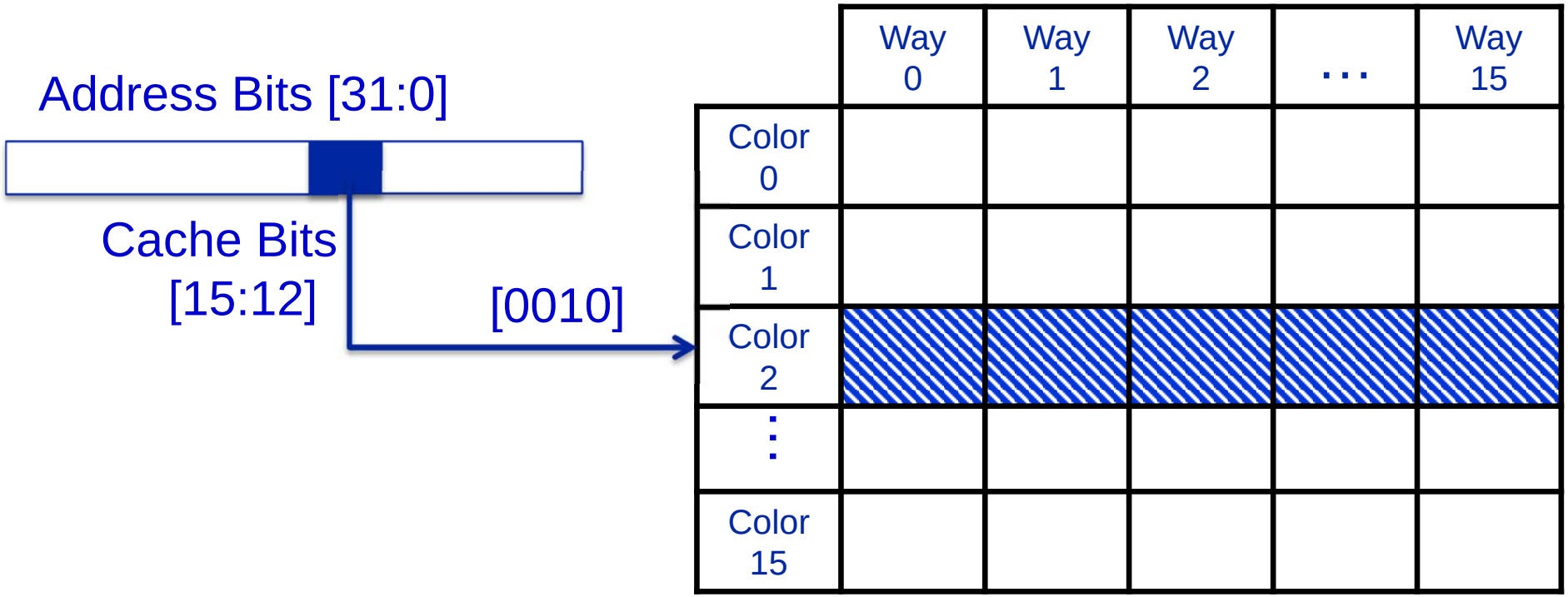
Hardware Platform

- Freescale i.MX 6Quad 800 MHz 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.



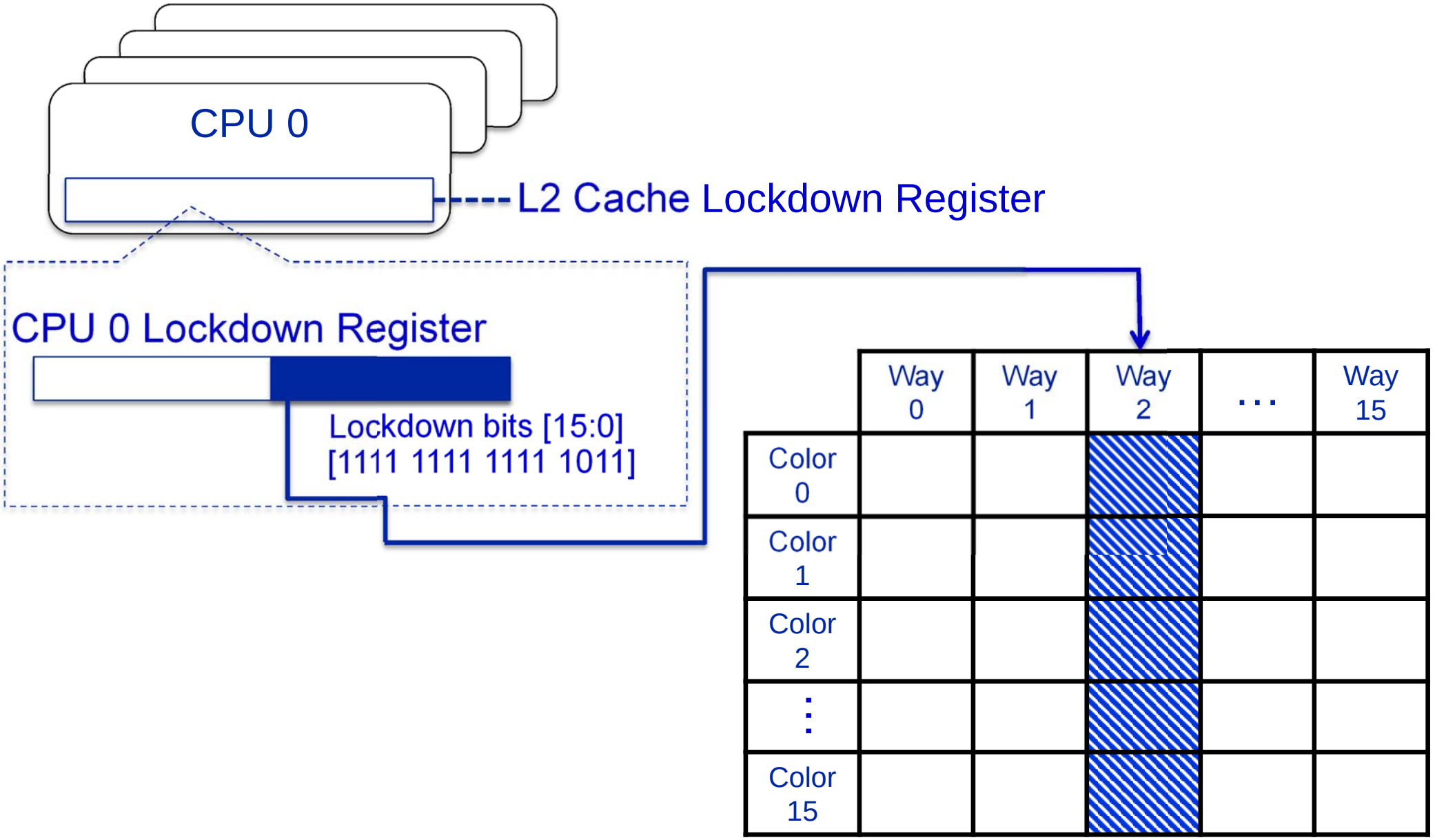
Cache Partitioning (of the Shared L2)

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

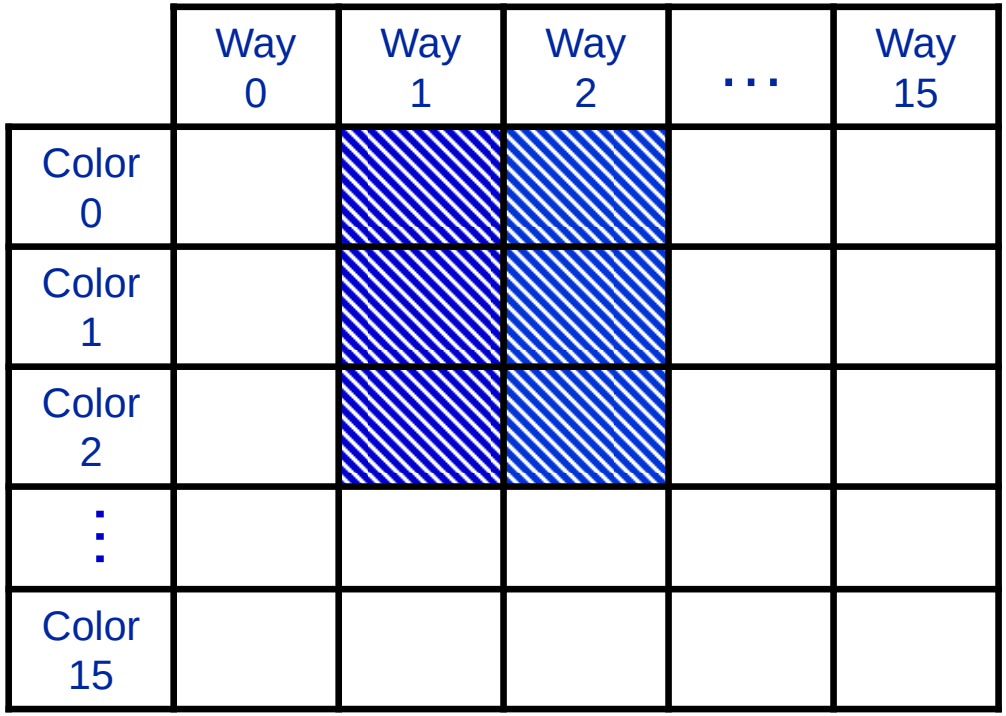


Cache Partitioning

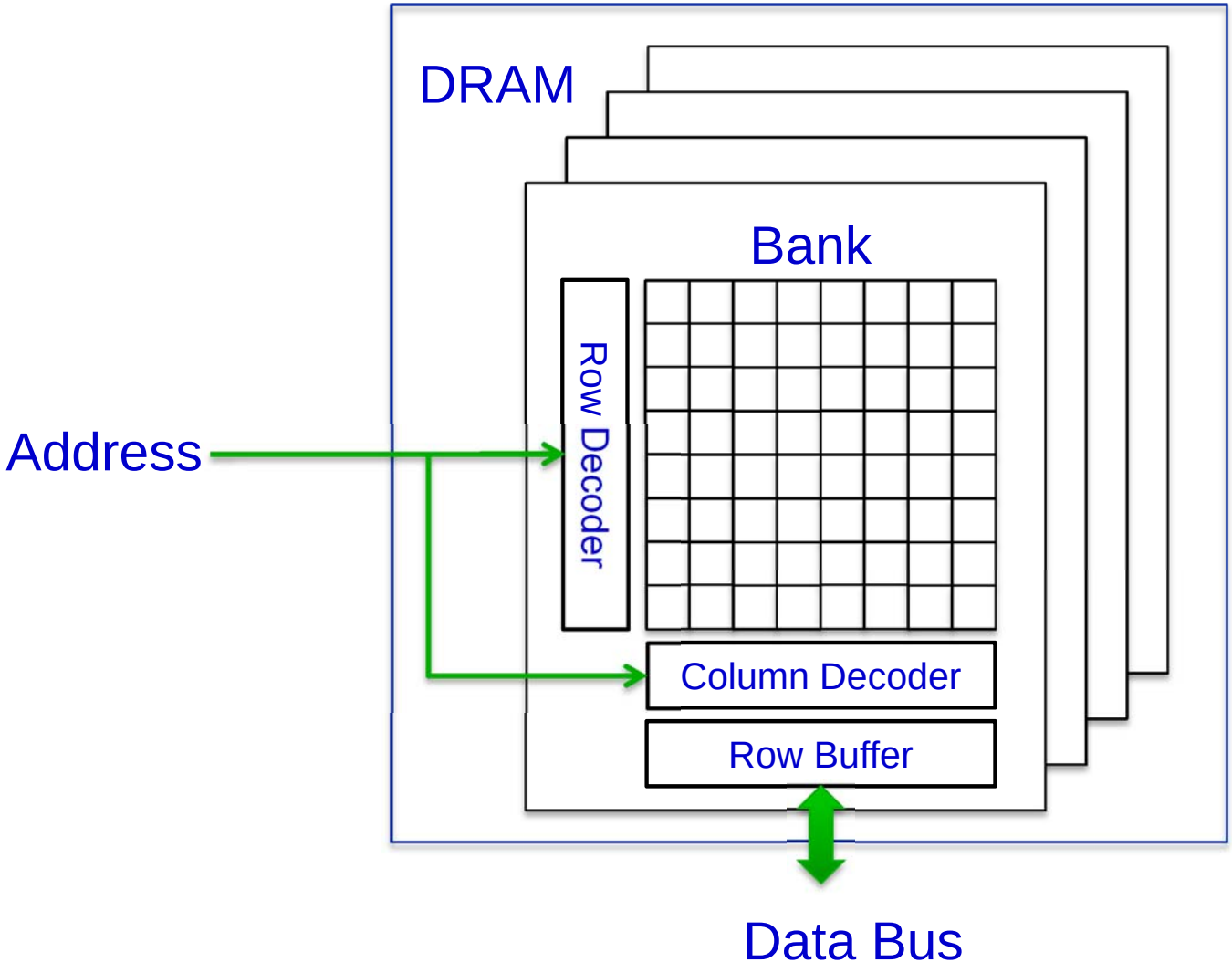
Option 2: Way Partitioning



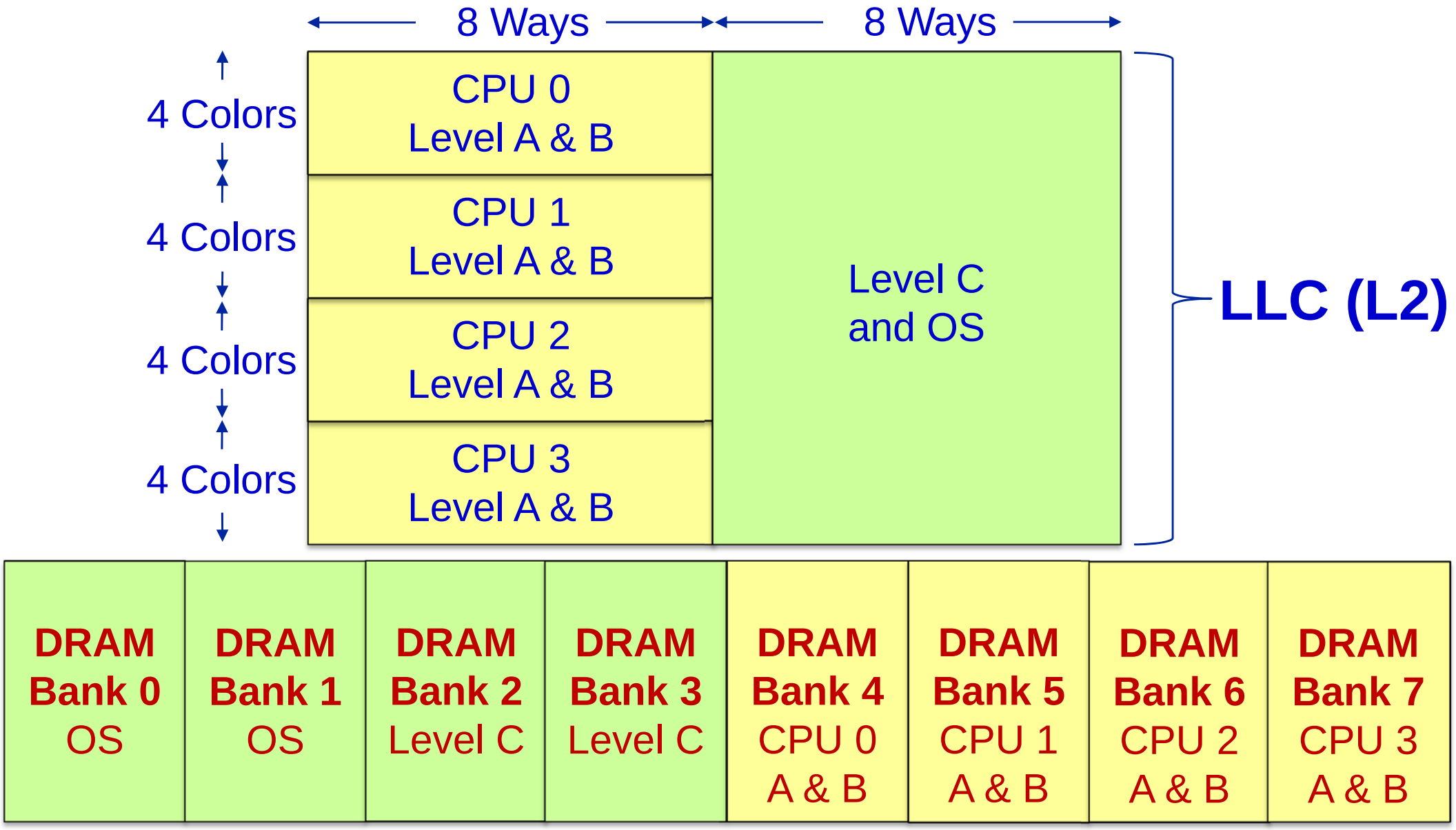
Can Combine these Approaches



DRAM Banks


DRAM
Bank
Row Decoder
Address
Column Decoder
Row Buffer
Data Bus

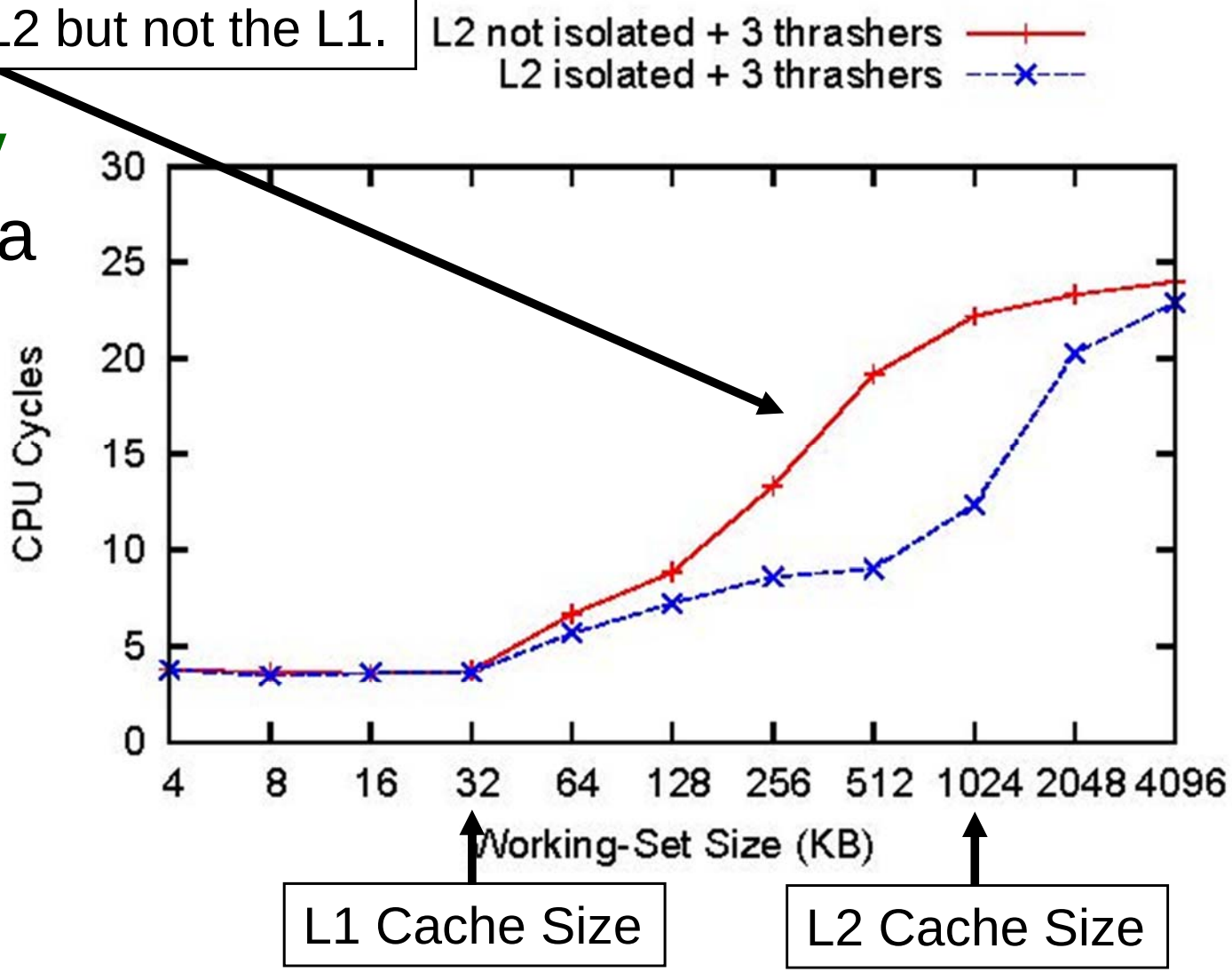
Currently Implemented Allocation Strategy



Importance of Controlling L2 Interference

Up to 2X reduction when working set mostly fits within the L2 but not the L1.

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

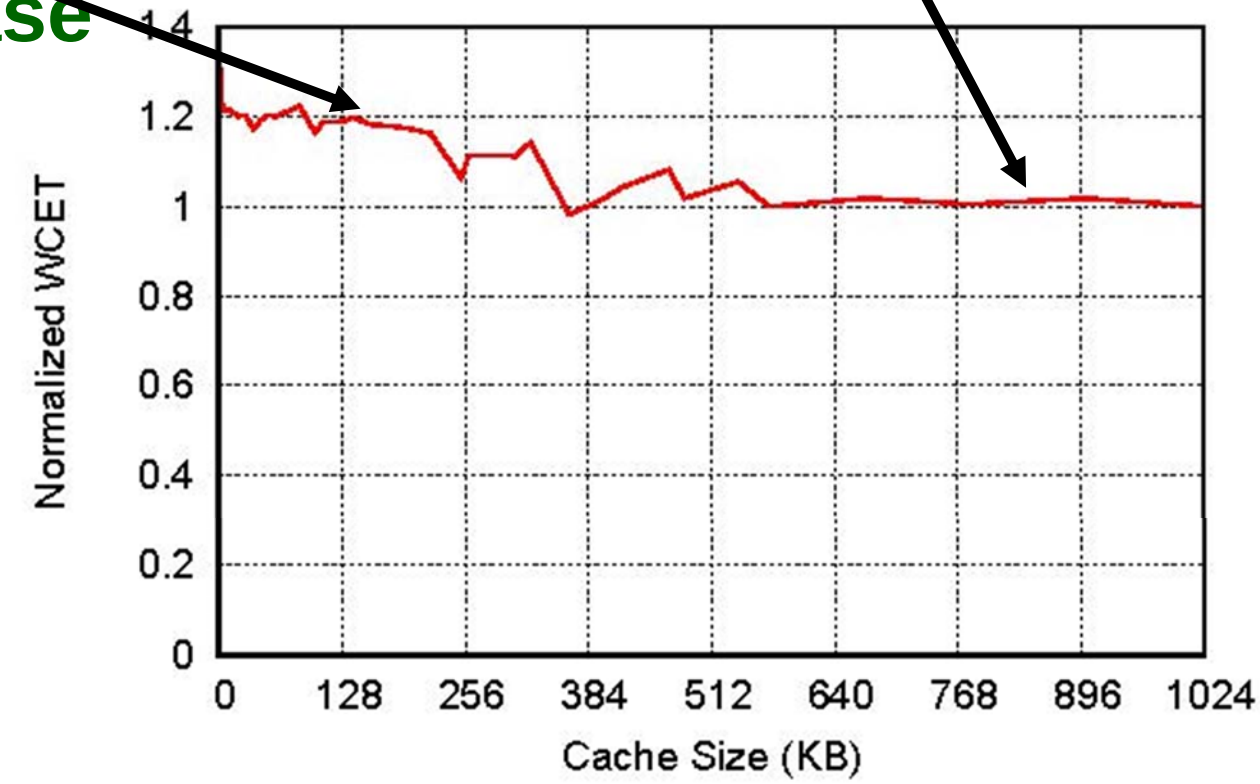


Importance of Controlling DRAM Bank Interferen

~20% reduction from bank isolation here.

No benefit from bank isolation when allocated L2 area is “large.”

Normalized worst-case execution time of a synthetic task with a 256KB working set size, as a function of allocated L2 area.

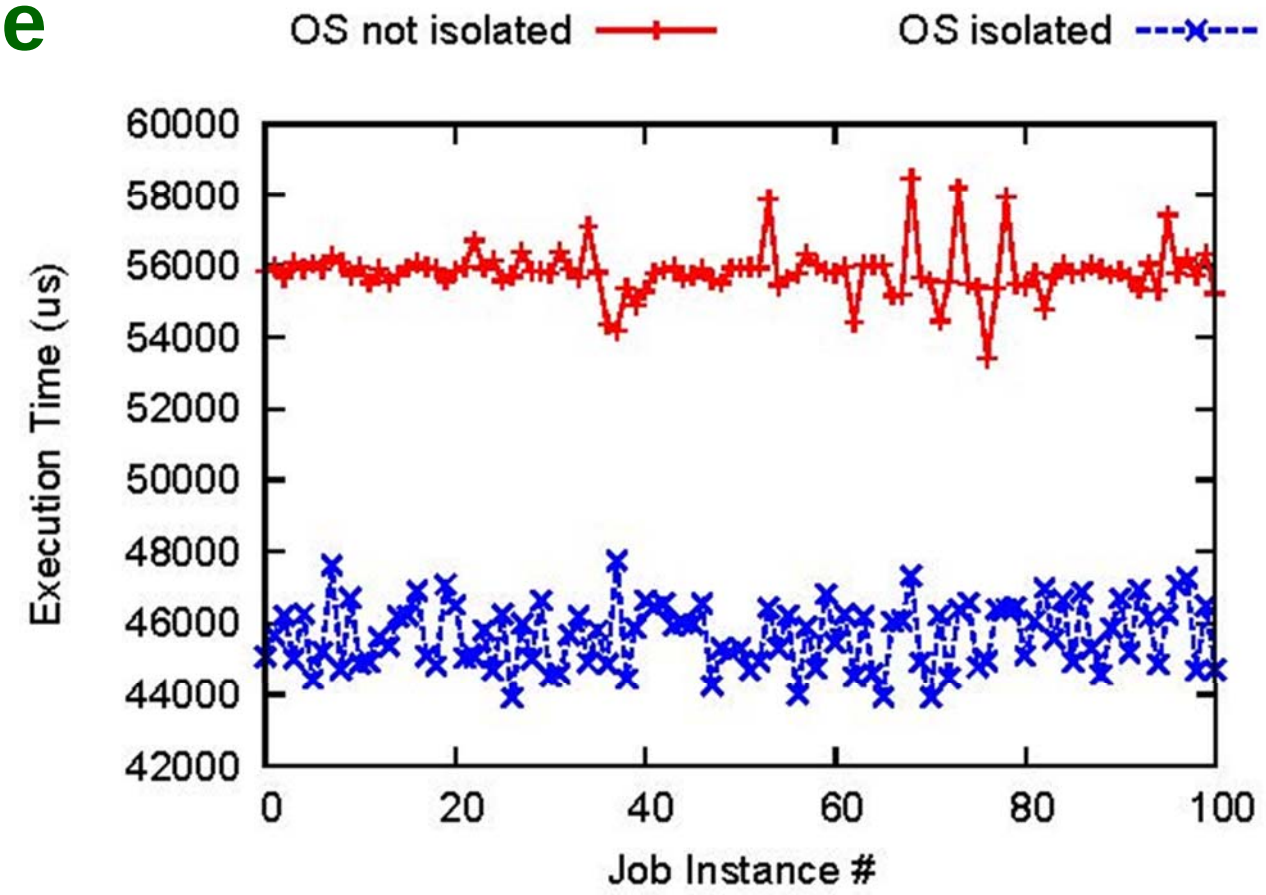


Normalized WCET =

$$\frac{\text{(WCET w/o bank isolation)}}{\text{(WCET w/ bank isolation)}}$$

Importance of Controlling OS Interference

Measured worst-case execution times of a synthetic task that repeatedly invokes a dummy system call, with **(BLUE)** and without **(RED)** OS isolation.



Tasks have “large” working set sizes, which stresses the L2 cache.

Tasks have “small” working set sizes, so the L2 cache is stressed less.

To illustrate the importance of applying both MC analysis and HW management, we constructed two quad-core task systems, LLC-Heavy and LLC-Li

Only these result in a schedulable system.

We obtained these total utilization measurements:

Task System	Average Total Utilization			
	no HW mgt. no MC anal.	HW mgt. no MC anal.	no HW mgt. MC anal.	HW mgt. MC anal.
LLC-Heavy	8.688	5.466	4.768	3.592
LLC-Light	4.395	4.229	3.721	3.661

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

- Driving problem.
- Prior work: The MC^2 (mixed-criticality on multicore) framework.
- New work: MC^2 with shared hardware management.
- Future work.
- Running a large schedulability study.

Future Work

- Our future plans include:
  - » Devising (near) optimal algorithms for allocating L2 areas and DRAM banks.
  - » Extending page coloring to fully deal with dynamically allocated pages and shared pages.
  - » Enabling dynamic task-system adaptations and synchronization.
  - » Conducting a major schedulability study to fully understand relevant resource-allocation tradeoffs.

Our Methodology for Schedulability Exps.

Developed Jointly with Björn Brandenburg

We use worst-case (average-case) overheads for HRT (SRT).

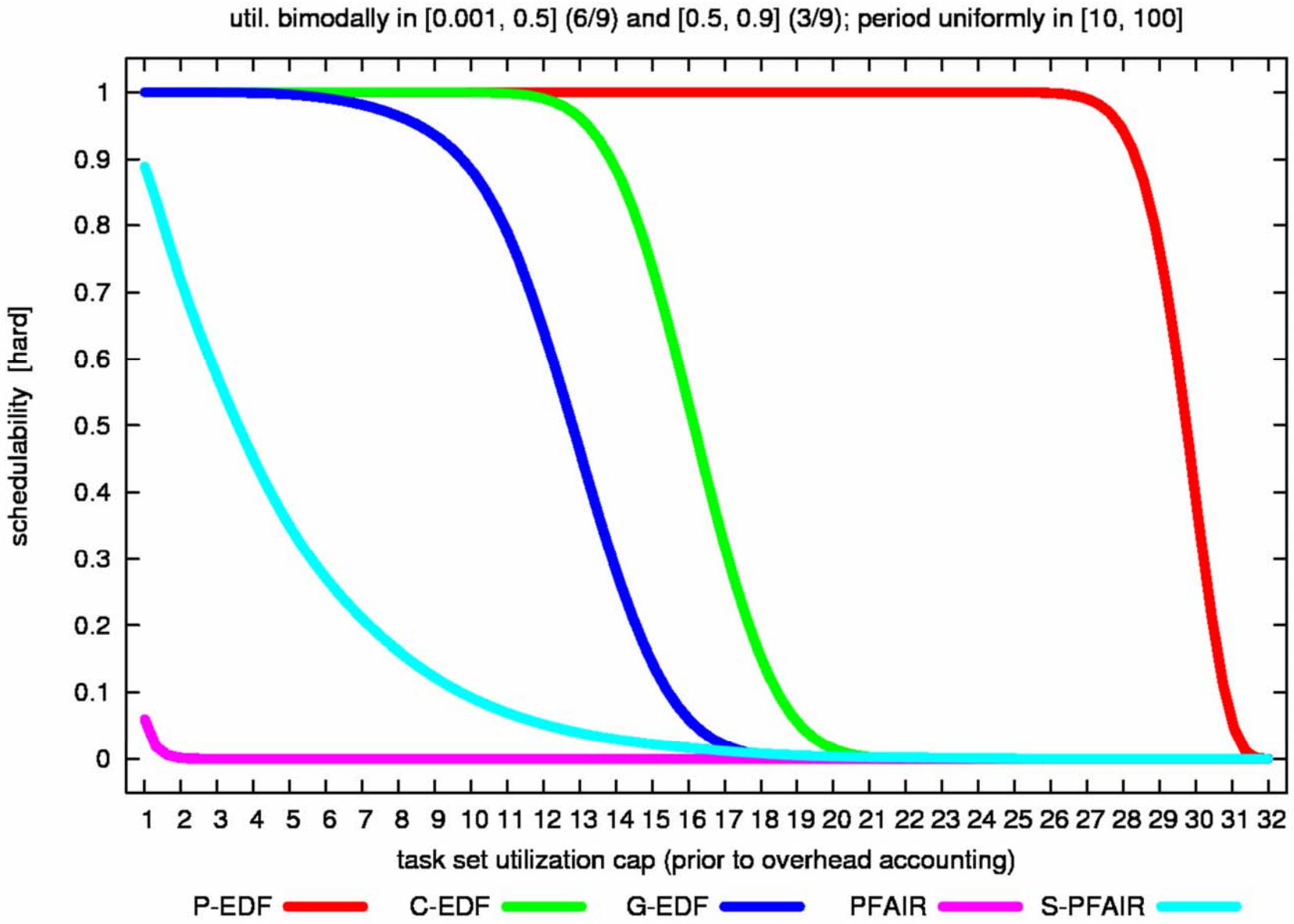
Use monotor
piecewise linear inte
to compute overhead
expressions as a fun
(the task coun

Generate several million random task sets and check schedulability with overheads considered. Done on a 500+ node research cluster. Can take a day or more.

Involves tracing the behavior of 1000s o synthetic tasks in LITMUS^{RT} on test platform. Usually takes 8-12 hours. Yields many gigabytes of trace data.

ability experiments.

The End Result is Lots and Lots of Schedulability Graphs that Look Like This



Running Larger Scale Experiments

- The process just described pertains to evaluating ordinary multiprocessor schedulers.
- In recent work by Glenn Elliott involving GPUs, the sheer scale of this process started becoming an issue.
  - » Large scale means experiments take a long time to complete…
    - 250,000 CPU hours for Glenn's dissertation!
  - » … and huge amounts of data must be understood.
    - Glenn used special query-processing tools to address this. Also, weighted schedulability graphs are useful here.

Controlling Scale

- When randomly generating task systems, we have two options:
  - » If a parameter can be reasonably constrained based upon domain knowledge or measurements, then it make sense to constrain it.
    - – For example, task periods are commonly in the range of 10s of ms to 100s of ms.
    - – OS overheads (obtained via measurement) often have small ranges.
  - » Otherwise, a large range that encompasses all reasonable values should really be assumed.
    - – But this causes the scale to blow up!

Questions

- In generating random task systems, what should we assume about:
  - » The distribution of tasks across criticality levels?
  - » The assignment of task to processors?
  - » The allocation of hardware resources to tasks?
  - » Per-criticality-level execution times?
  - » Per-criticality-level overhead values?
- And down the road…
  - » Critical sections and precedence constraints?
  - » Dynamic task behaviors?

Questions (Cont'd)

- With respect to the measurement process:
  - » Should we use synthetic tasks?
    - – **Advantage:** Their properties can be systemically controlled.
    - – **Disadvantage:** May not reflect "practical workloads."
  - » Benchmark tasks?
    - – **Advantage:** May exhibit more "real world behaviors."
    - – **Disadvantage:** Their properties may be hard to discern.
    - – **Disadvantage:** ***Just because a program is labelled as a "benchmark" in one domain doesn't mean it has any relevance in another.***
  - » If so, which benchmarks should we use?

Help!

- I look forward to hearing your thoughts on these and related questions as the workshop progresses today…

MC^2 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.

MC^2 Papers

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

- N. Kim, B. Ward, M. Chisholm, C.-Y. Fu, J. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” manuscript.
  - » Is the basis for **today’s presentation**.
- M. Chisholm, B. Ward, N. Kim, and J. Anderson, “Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems,” manuscript.
  - » Presents linear-programming-based techniques for **optimizing LLC area allocations**.

Thanks!

- Questions?