Towards Certifiable Hardware Resource Sharing in Multicore Processors

Presenter: Dionisio de Niz

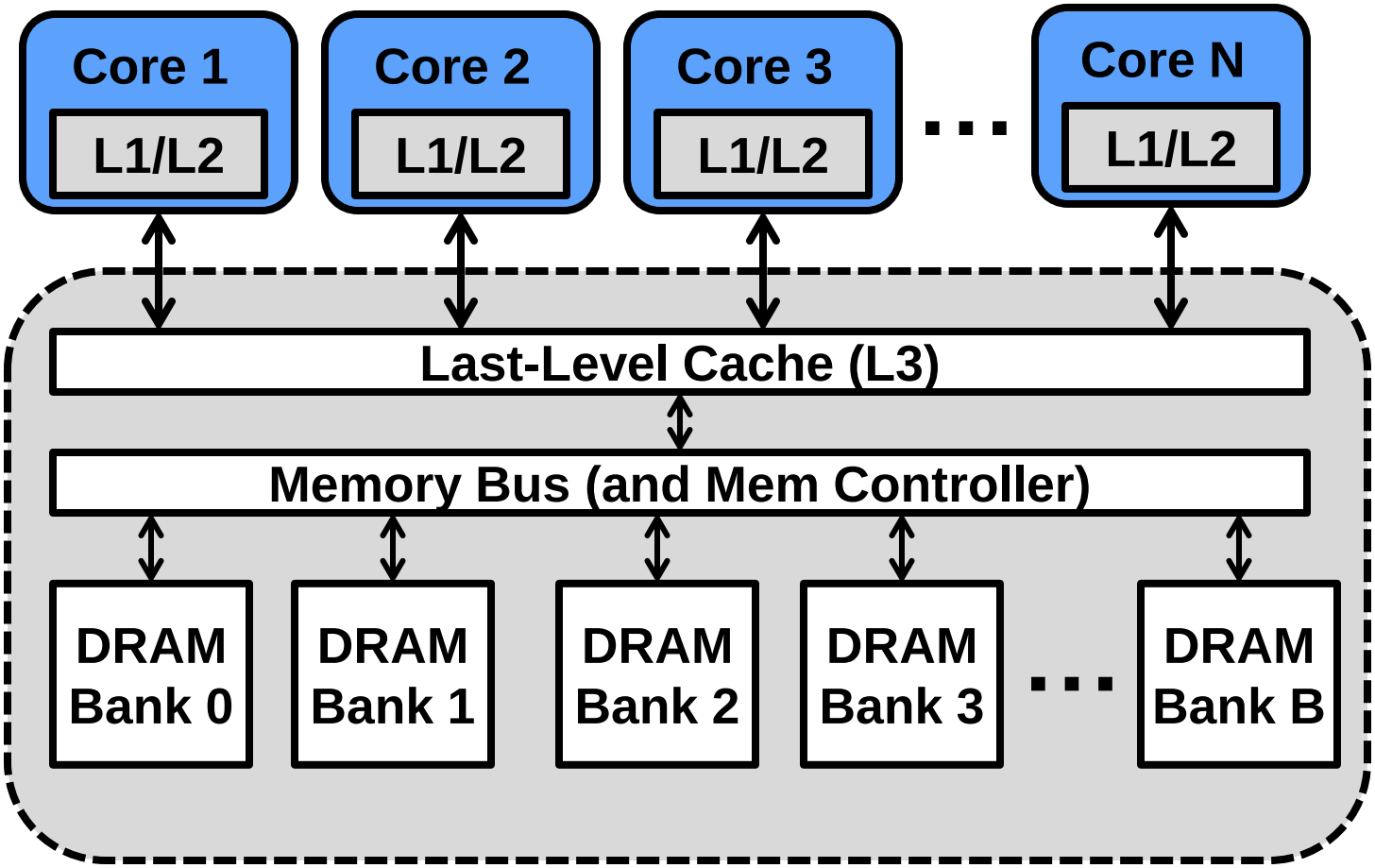
Hyoseung Kim, Bjorn Andersson, Mark Klein, and Raj Rajkumar.

CMAS 2015 April 13, 2015

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0002331

Sharing of Multiple Hardware Resources



Need of Coordinated Protection

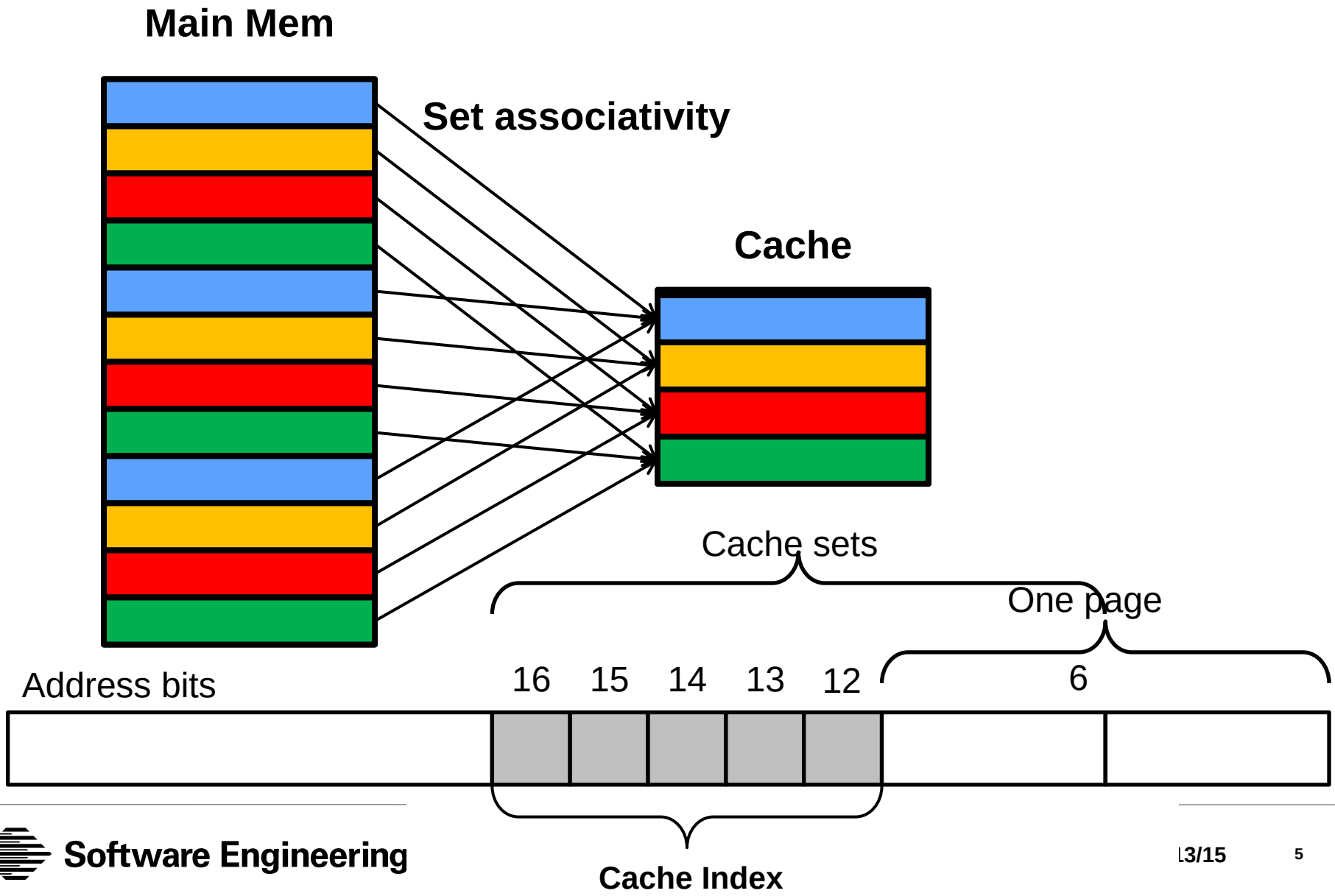
Need to constrain interference through each resource type

- CPU cycles
- Cache
- Memory Banks
- Memory Bus / inter-core network

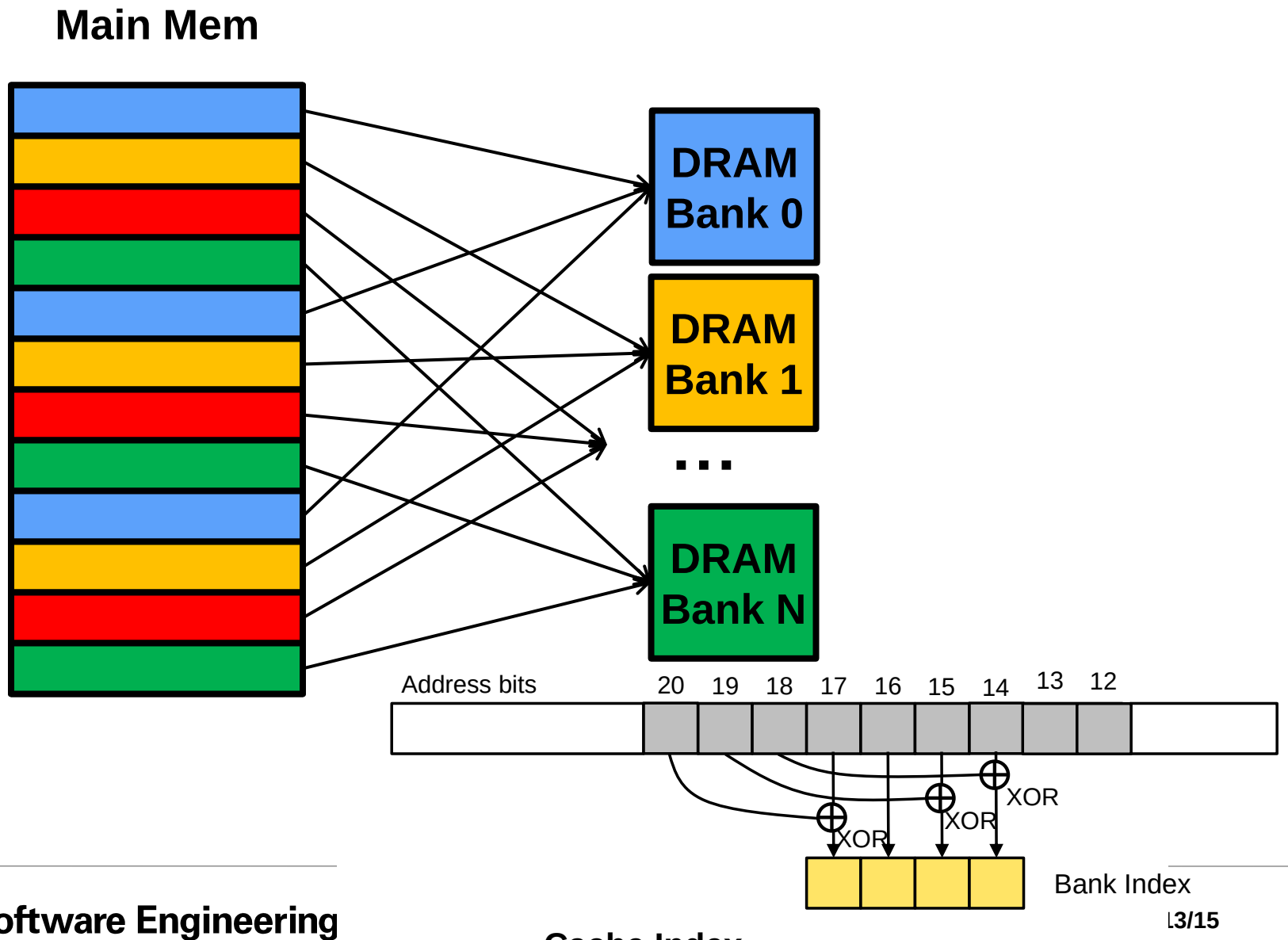
Ensure no inconsistent configuration

- Configuration for one resource does not invalidate configuration of another

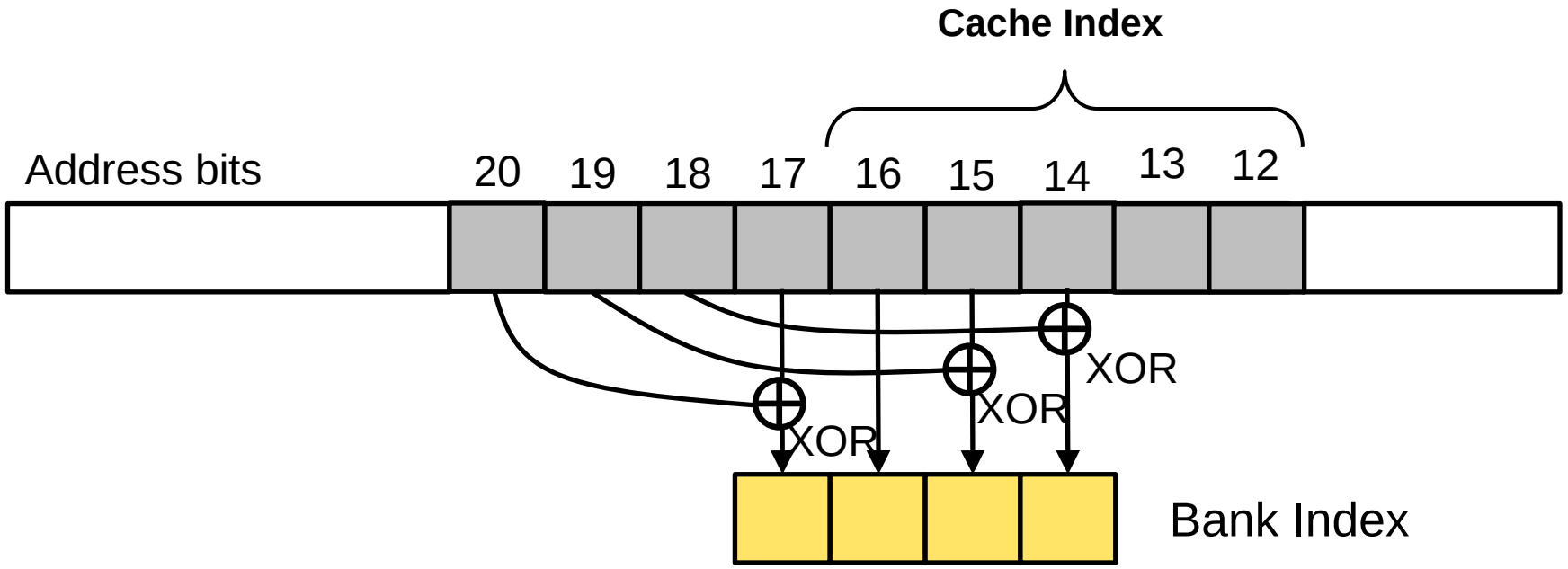
Cache Partitioning (Coloring)



Bank Partitioning (Coloring)



Cache and Bank Address Bits



E.g. 2 bank bits
2 cache bits
1 shared bit

Cache \ Bank	00	01	10	11
00	X		X	
01	X		X	
10		X		X
11		X		X

Row-Bank Address Bit Xoring Improves Coverage

If two additional bits are xor with bank bits we can get all combinations

Bank Colors

	row	bank		row	bank		row	bank		row	bank	
	00	00		01	00		10	00		11	00	
	01	01	= 00	00	01	= 01	11	01	= 10	10	01	= 11
	10	10		11	10		00	10		01	10	
	11	11		10	11		01	11		00	11	
00	X			X			X			X		
01	X			X			X			X		
10	X			X			X			X		
11	X			X			X			X		

Cache Colors

Coordinated Cache and Bank Partitioning & Core Allocation

Avoid conflicting color assignments

Take advantage of different conflict behaviors

- Banks can be shared within same core but not across cores
- Cache cannot be shared within or across cores
- Coordinated core and bank color allocation

Take advantage of sensitivity of execution time to cache

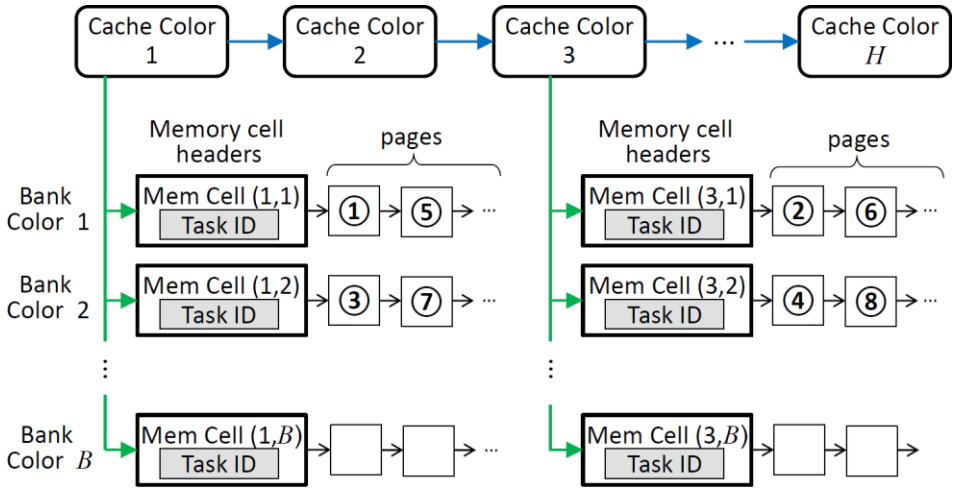
- Task with highest sensitivity to cache is assigned more cache
- Diminishing returns taken into account

Two algorithms explored

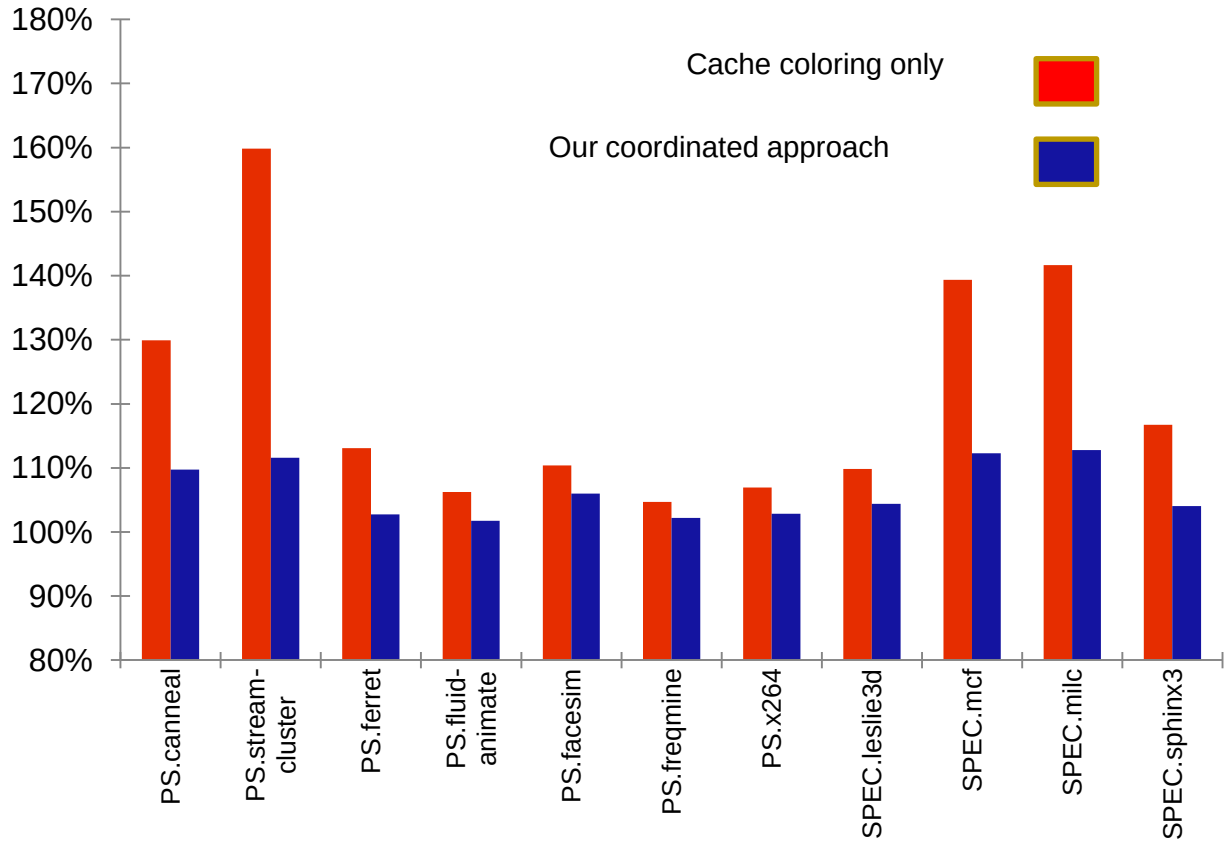
- Mixed-Integer Linear Programming
- Knapsack

Implementation of Cache+Bank Coloring

Linux / RK : Kernel Memory Manager
Memory reserves with set of bank and cache colors
Pages are classified in cache and bank colors
Added to resource sets that are attached to multiple processes/threads



Experimental Results



Limited Number of Private Partitions

Private partitions significantly reduces usable memory

- Number of bank/cache cells in memory
  - Number of cells = (B*H). Size of cell $C = \frac{M}{(B*H)}$
  - With: M = size of memory, B= # bank colors, H = # cache colors
  - E.g. Intel core i7 2600
    - M = 4GB, B = 16, H = 32 $C = \frac{4GB}{16*32} = 8MB$
- Private partitions ≡ one cell per cache color & one cell per bank color
  - Number of private partitions $PP = \min(B, H)$
  - E.g. Intel core i7 2600 : $PP = \min(16,32) = 16$
- Extreme (using all private partitions) total usable private partition memory
  - $PPM = PP * C$
  - Intel core i7 2600 : $PPM = 8MB * 16 = 128MB$
    - Memory utilization $= \frac{128MB}{4GB} = 3.13\%$

Allowing Sharing

In Partitioned Scheduling OK to share banks within core

- Number of banks are no longer a restriction: $PP = H$
- Partitions sharing banks in a core
  - # Sets of independent partitions $I = N$; N = number of cores
  - Memory utilization (uniform partitions) = $\frac{M}{I}$
- Intel Core i7 2600: $I = N = 4$
  - Memory utilization (uniform partitions) = $\frac{4GB}{4} = 1GB = 25\%$

Need better utilization

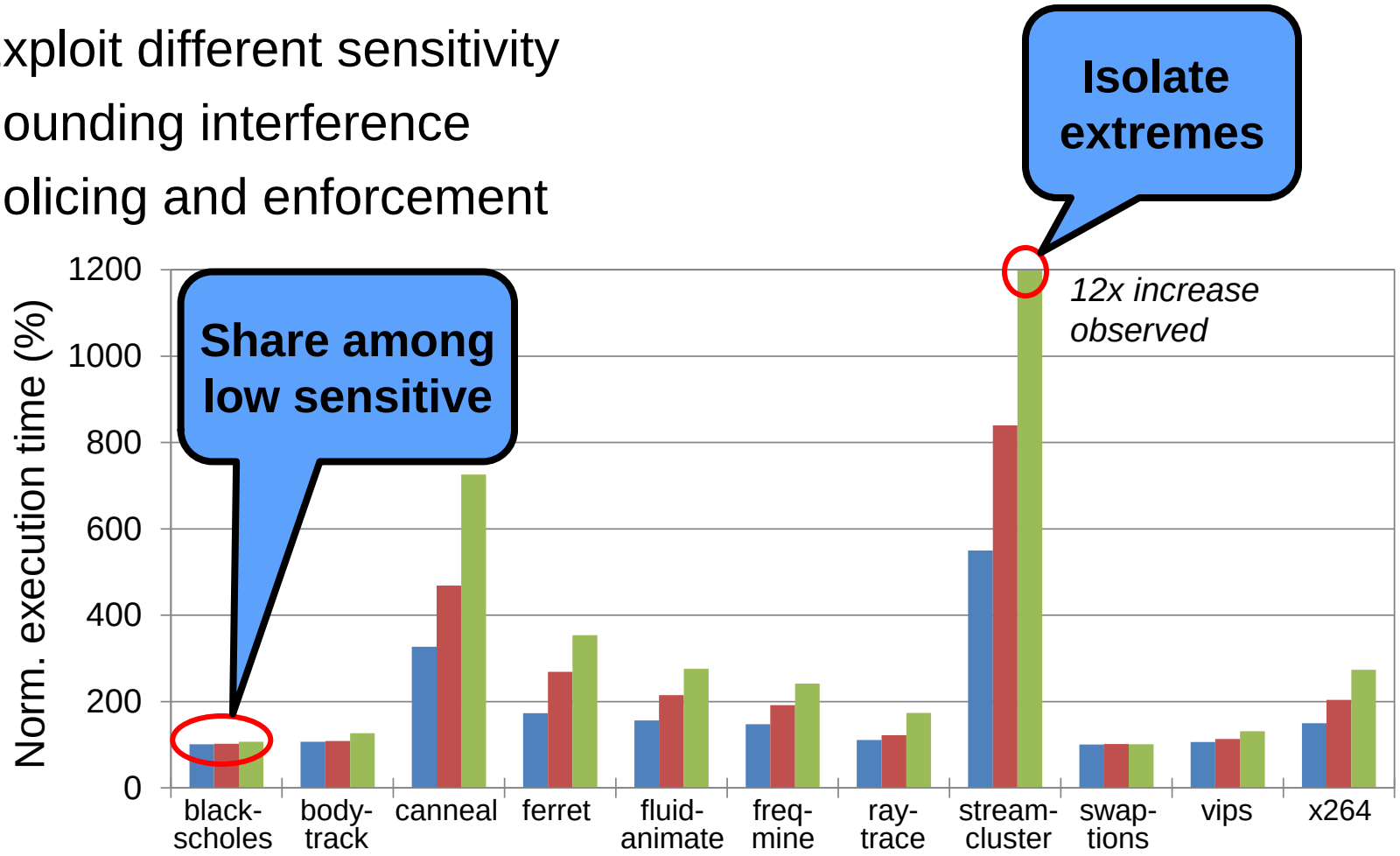
Partitions may not be enough for number of tasks

Predictable Sharing

Exploit different sensitivity

Bounding interference

Policing and enforcement



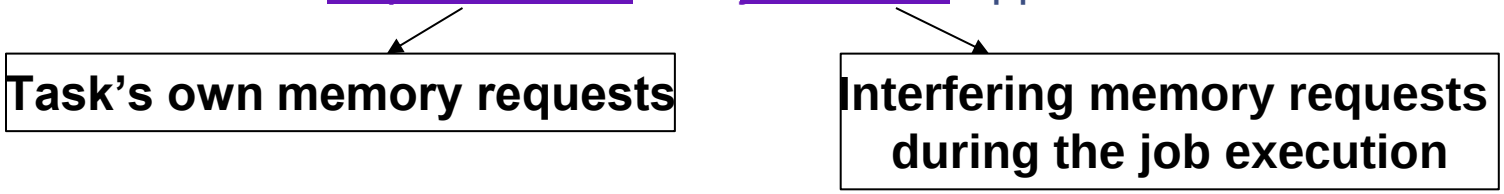
Bank Partitioning (Coloring) + Timing Analysis

Explicitly considers the timing characteristics of major DRAM resources

- Rank/bank/bus timing constraints (JEDEC standard)
- Request re-ordering effect

Bounding memory interference delay for a task

- Combines request-driven and job-driven approaches



Software DRAM bank partitioning awareness

- Analyzes the effect of dedicated and shared DRAM banks

Response-Time Test

- **Memory interference delay cannot exceed any results from the RD and JD approaches**
  - We take the smaller result from the two approaches

- **Extended response-time test**

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

Classical iterative response-time test

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

Request-Driven (RD) Approach

Job-Driven (JD) Approach

Memory-Interference Aware Task Allocation

Observations

- Memory interference due to tasks running in other cores
- Tasks running on same core do not interfere with each other
- Collocate memory-intensive tasks on same core

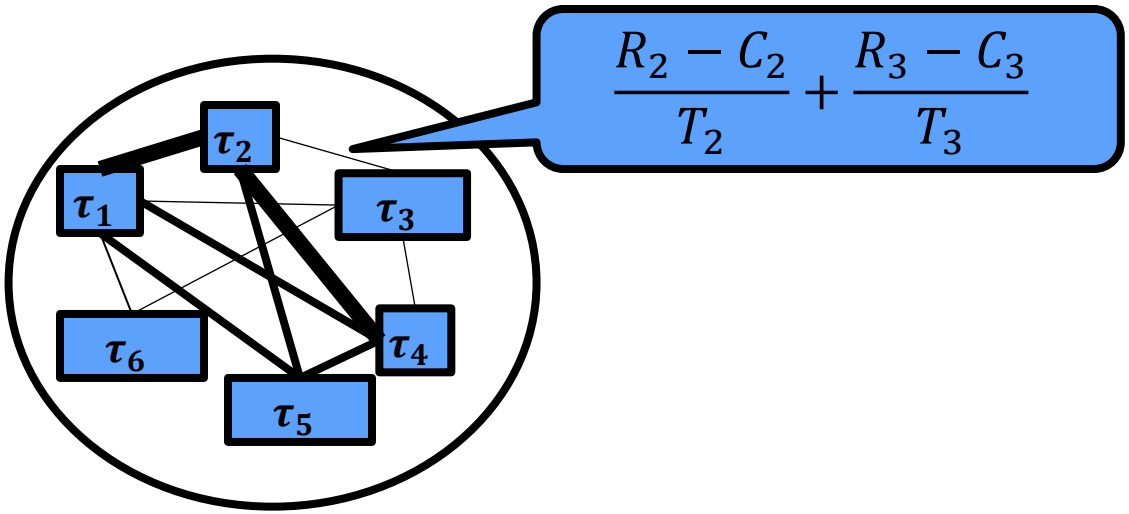
Graph $G = (V_i, E_{i,j}) : V_i = \tau_i, E_{i,j} = interference(\tau_i, \tau_j)$,

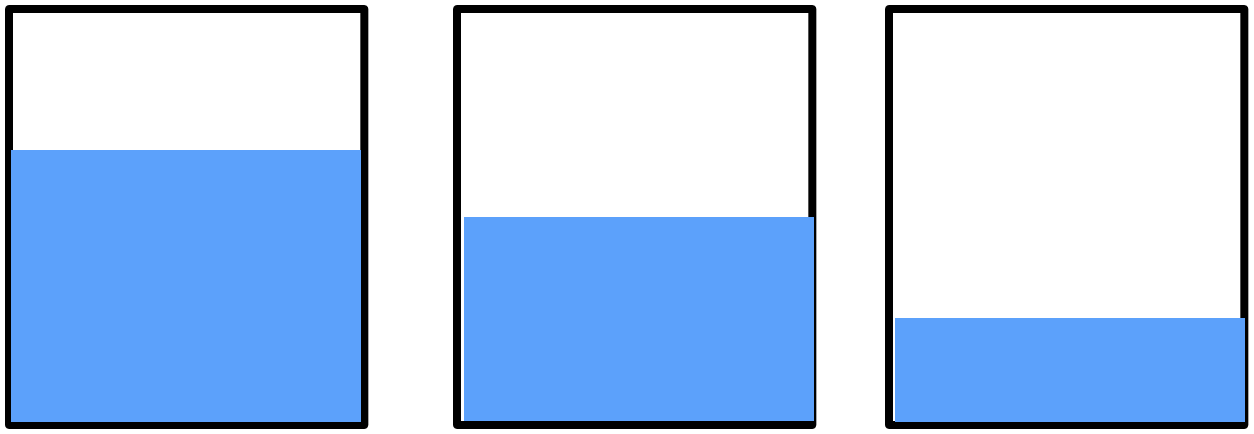
$$weight(E_{i,j}) = \frac{R_i - C_i}{T_i} + \frac{R_j - C_j}{T_j}$$

Following BFD:

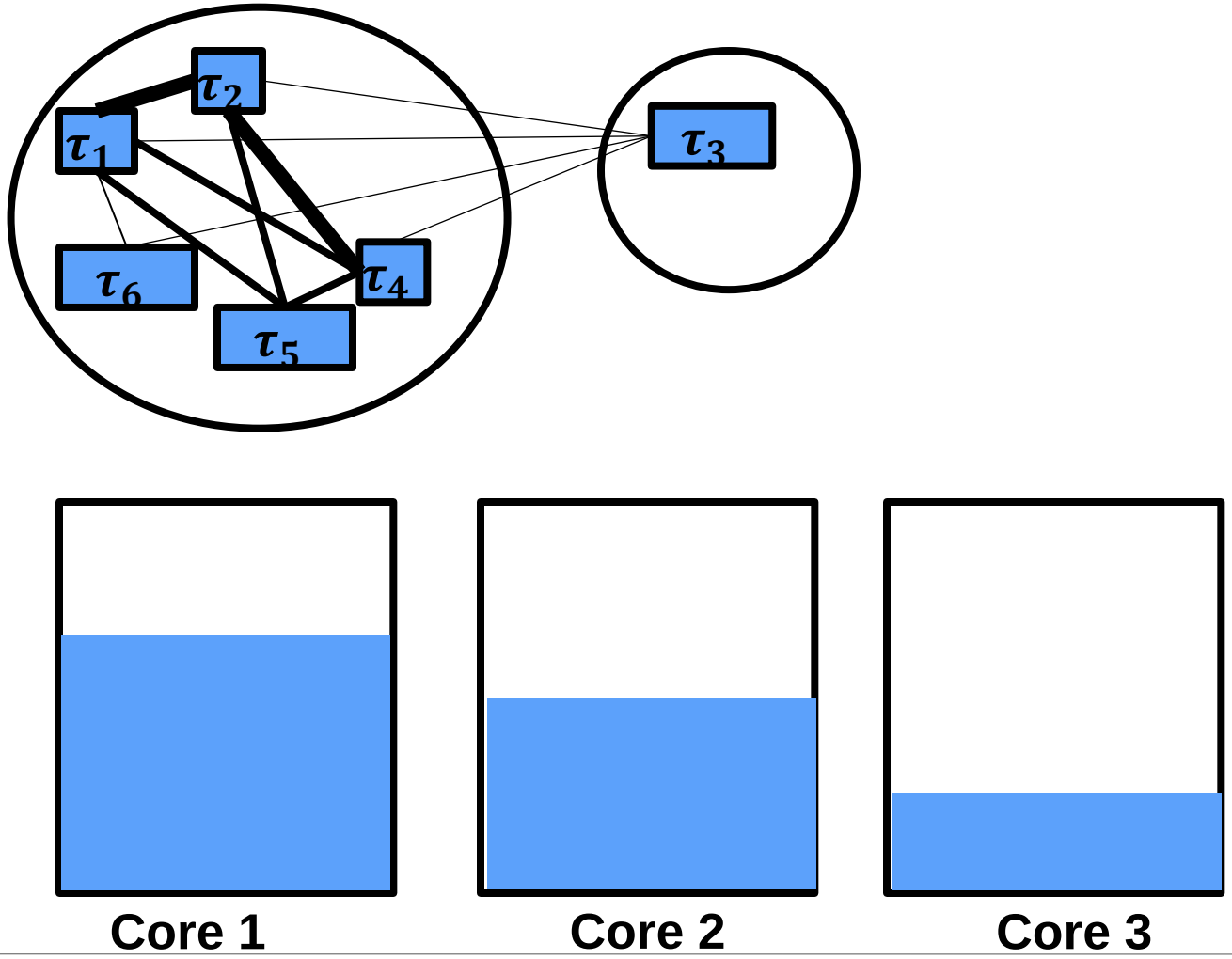
1. Try to deploy first un-deployed subgraph on bin (core)
2. If cannot
   - break graph with minimum cut (minimize edge weights)
     - One piece that fits largest gap + rest
3. Add to undeployed subgraphs
4. Goto 1

Minimum-Cut Memory Interference Packing

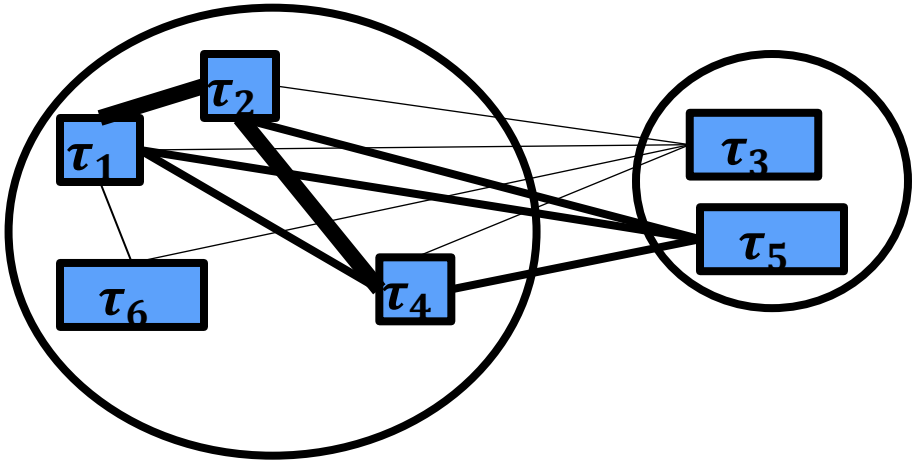


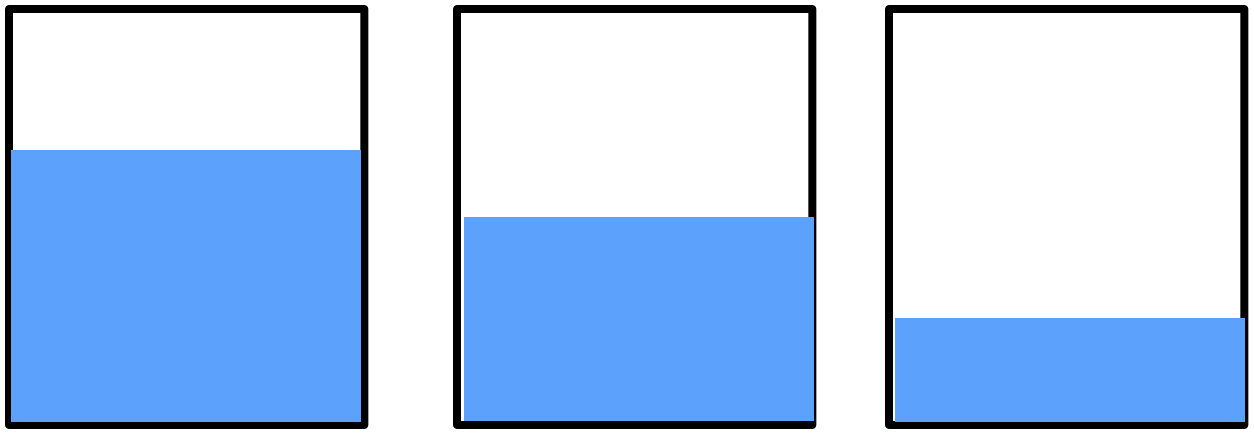


Minimum-Cut Memory Interference Packing

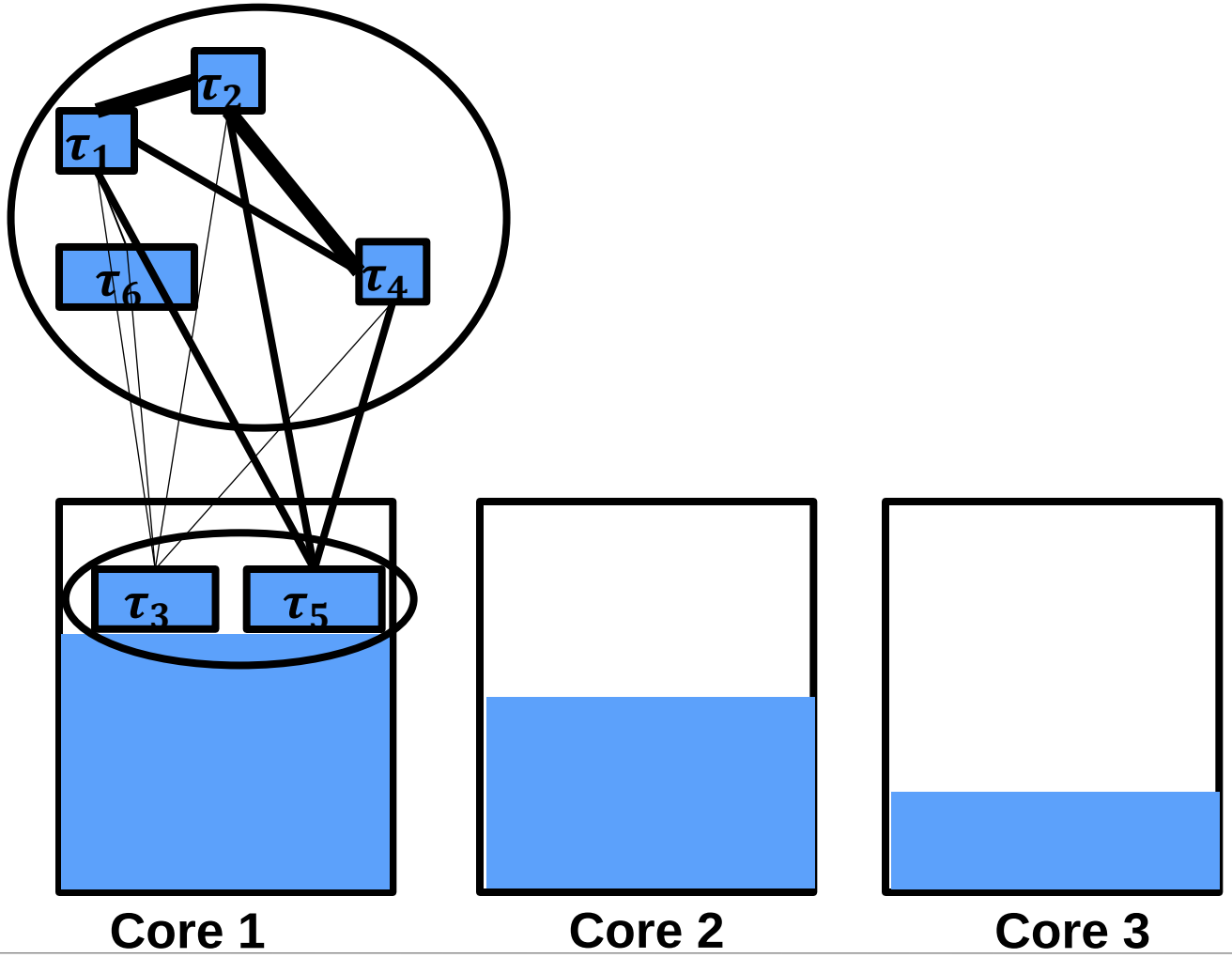


Minimum-Cut Memory Interference Packing

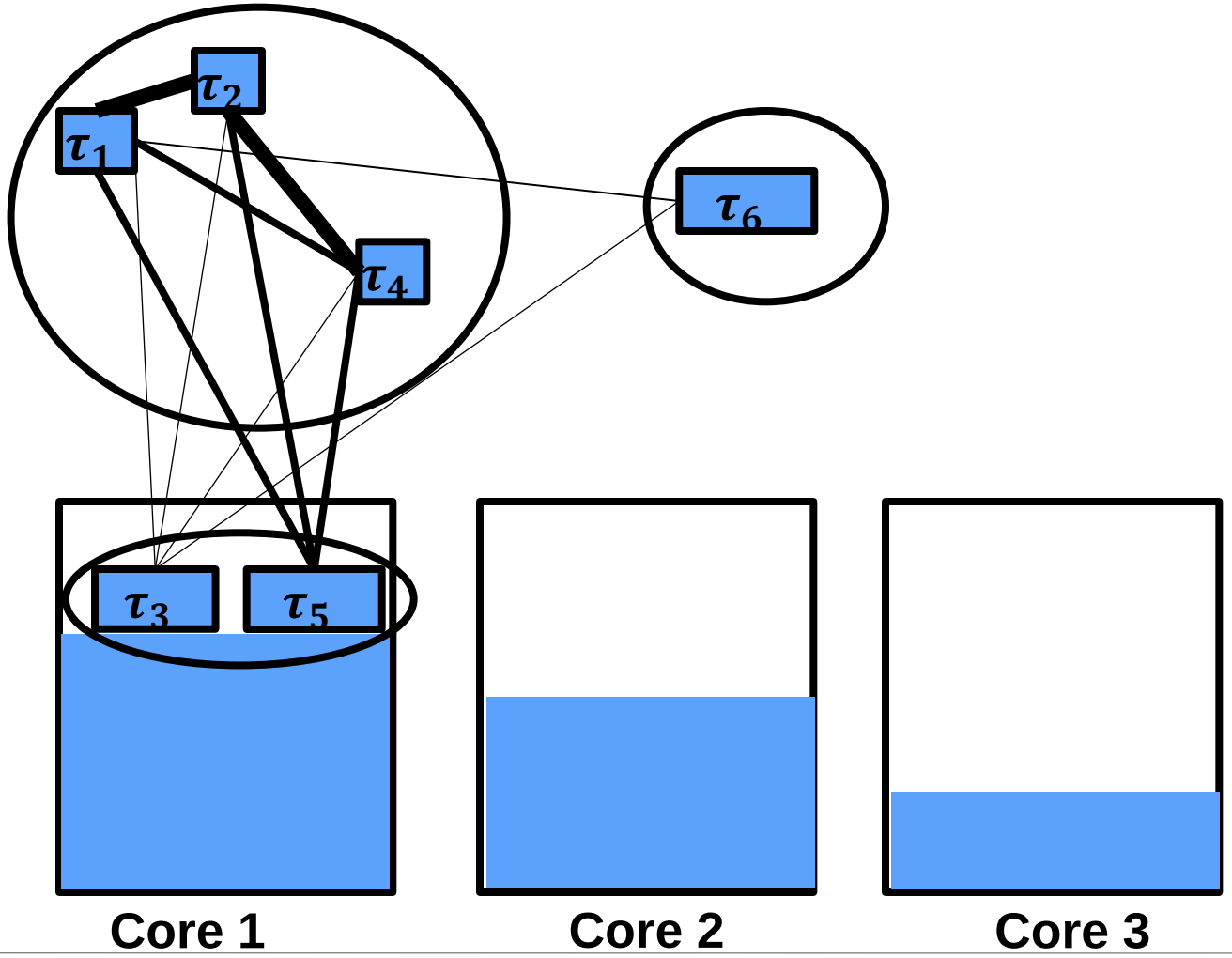




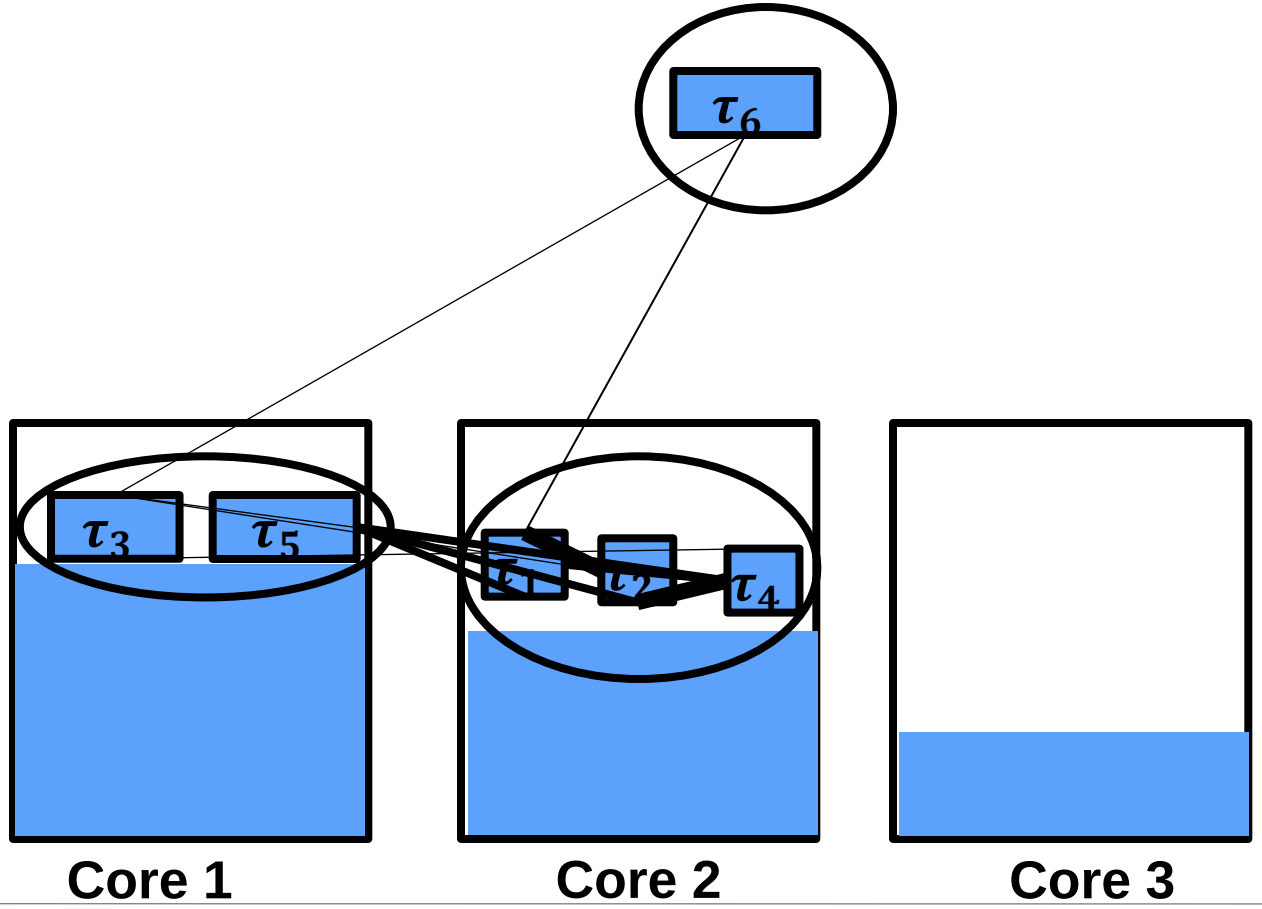
Minimum-Cut Memory Interference Packing



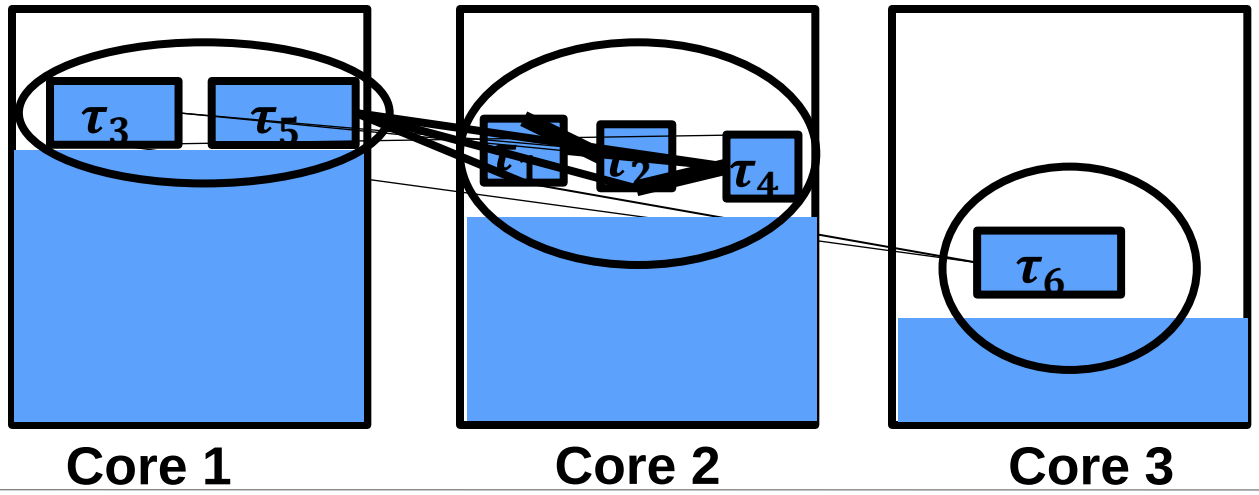
Minimum-Cut Memory Interference Packing



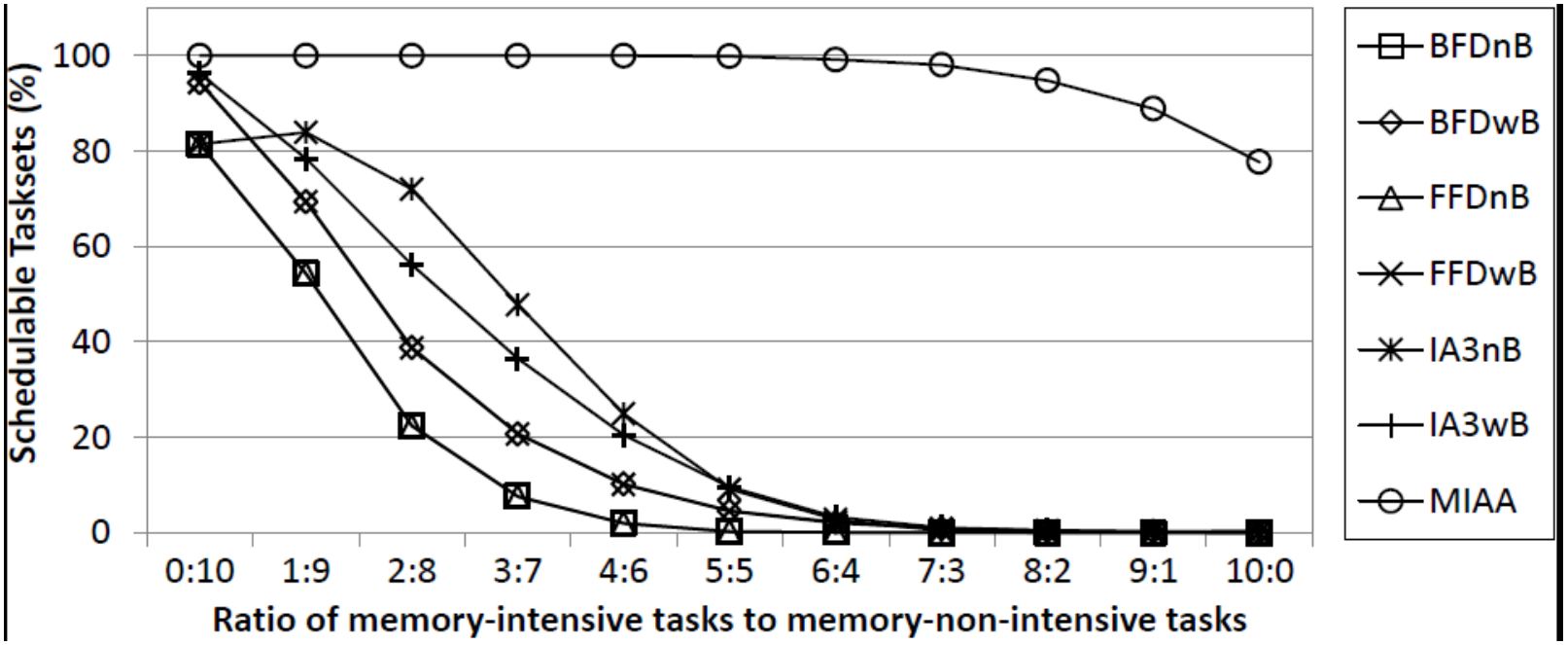
Minimum-Cut Memory Interference Packing



Minimum-Cut Memory Interference Packing

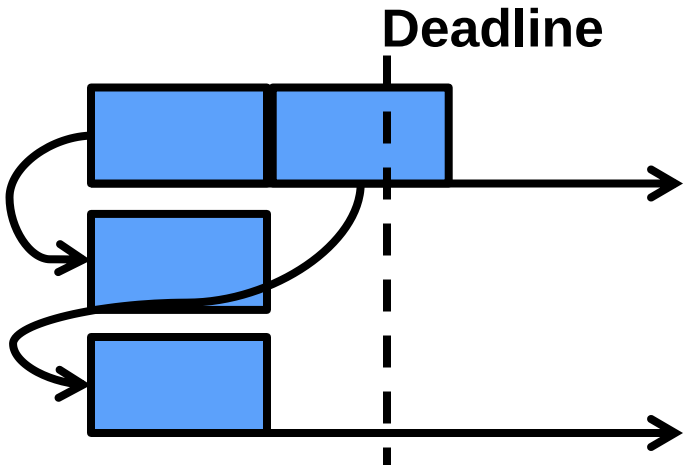


Memory-Interference Aware Task Allocation (MIAA)



IA^3 M. Paolieri, E. Qui~nones, F. Cazorla, R. Davis, and M. Valero. IA3: An interference aware allocation algorithm for multicore hard real-time systems. RTAS 2011.

Resource Conflicts for Parallelized Workloads



Parallelization

- Computation time > Deadline
  - Must parallelized to meet deadline
  - Guarantee always finish before deadline

Resource interference within a task

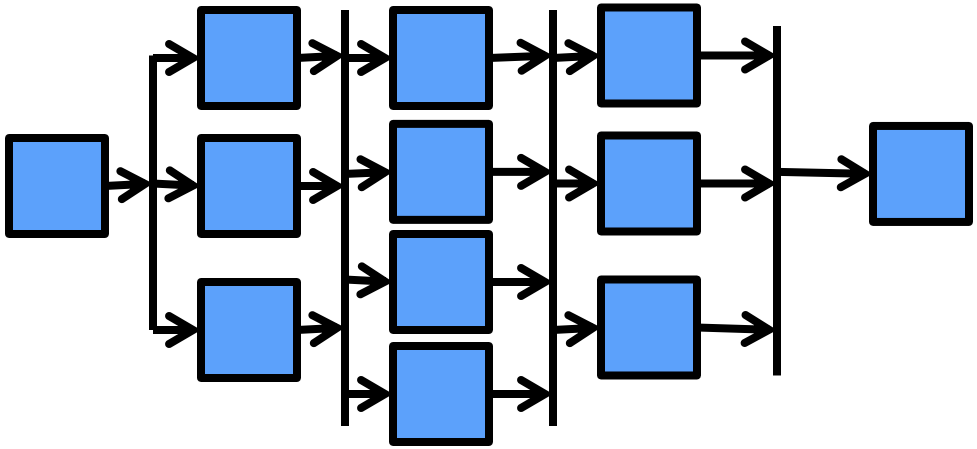
- Due to parallel subtasks
- Need to share memory to communicate

Predictable sharing

- Compatible with efficient parallelized task schedulers

Parallelized Task Scheduling

Developed a staged execution model



Scheduled under Global Earliest-Deadline First

- Most efficient scheduling for staged execution
  - If task schedulable under optimal scheduler our scheduler need at most twice the speed to schedule task

Challenges for Parallelized Task Resource Management

Intra-task partitions

- Threads with different sensitivities
- Assign different partitions to different parts of same tasks
  - Down to different colors for each page of a task

Inter-task shared partitions

- Shared partitions between parts of different tasks

Intra-task memory bus interference

Hardware and Software Profiling

Hardware

- Mapping of memory bits for cache and bank index
- Randomization strategies

Software

- Bound on number of memory accesses
- Temporal and spatial locality of accesses
- Techniques
  - Model checking (better term?)
    - Variable placement and access
    - Control-flow-based temporal and spatial locality
  - Profiling
    - Performance counters
    - Valgrind

Concluding Remarks

Multiple Resource Contention

- Need coordinated allocation/analysis

Resource Partitions

- Private partitions too restrictive
  - Too few – may not be enough for practical taskset sizes
  - Low utilization
- Shared partitions
  - Interference Sizing

Certifiable sharing

- Resource usage sizing : memory, cache, bus
- Sharing analysis models
- Allocation models