# A Fault-Aware Cache Management Policy for CMPs Operating at Ultra-Low Voltages

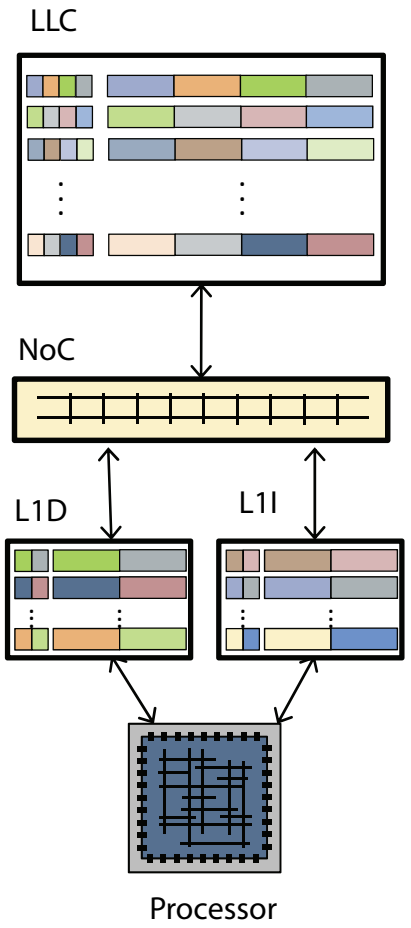
Alexandra Ferrerón-Labari, Jesús Alastruey-Benedé,
Darío Suárez Gracia, Teresa Monreal-Arnal,
Pablo Ibáñez-Marín, Víctor Viñals-Yúfera

Grupo de Arquitectura de Computadores
Departamento de Informática e Ingeniería de Sistemas
Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad de Zaragoza


Instituto Universitario de Investigación
de Ingeniería de Aragón
Universidad Zaragoza

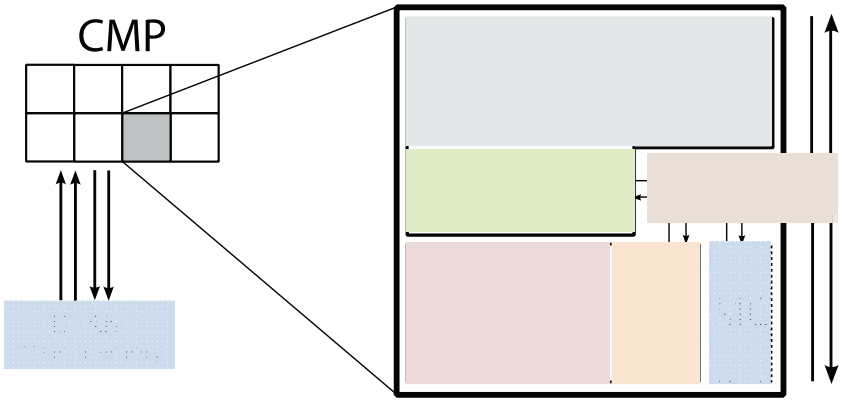
# How to Cope with Errors: Natural Redundancy

- Natural replication of blocks in inclusive cache hierarchies:
  - Exploit the memory hierarchy organization
  - → **BDOT-FA**
    a fault-aware cache mgt. policy



# System Overview

- 8-core tiled CMP




Ultrasparc III Plus, in-order, 1 instr/cycle, single-threaded, 1GHz at $V_{dd}$ 0.5 V

Private, 64 KB data and inst. caches, 4-way, 64 B block size, LRU, 2-cycle hit access time

Shared, 1 bank/tile, 1 MB/bank, 16-way, 64 B block size, Pseudo-LRU, Inclusive, 8-cycle hit access time

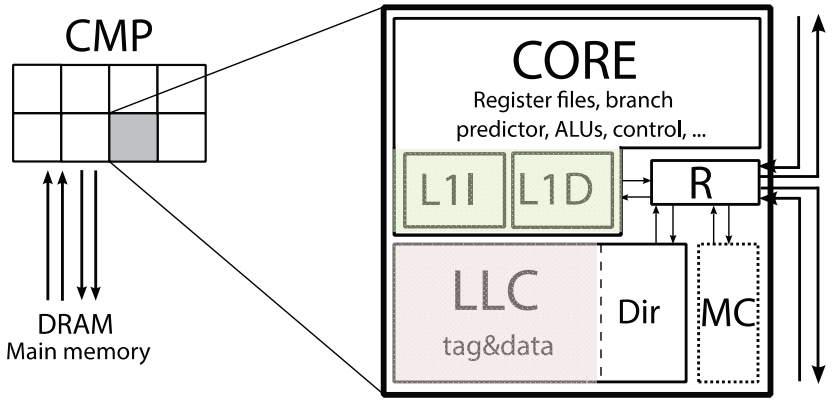
MESI, directory-based, full-map distributed among LLC banks

2 MC, DDR3 1333 MHz, 2 channels, 8 Gb/channel, 8 banks, 8 KB page size, open page policy

Mesh, 2 Virtual Networks (requests & replies), 16-byte flit size, 2-stage routers, 1-cycle latency hop

# System Overview

- 8-core tiled CMP




L1 caches built with robust cells (fault-free) [KH09];
e.g., 8T SRAM cells [CMN+08]

LLC cache built with regular 6T cells (prone to errors)


[CMN+08] L. Chang et al., "An 8T-SRAM for Variability Tolerance and Low-Voltage Operation in High-performance Caches." *IEEE Journal of Solid-State Circuits,* 2008

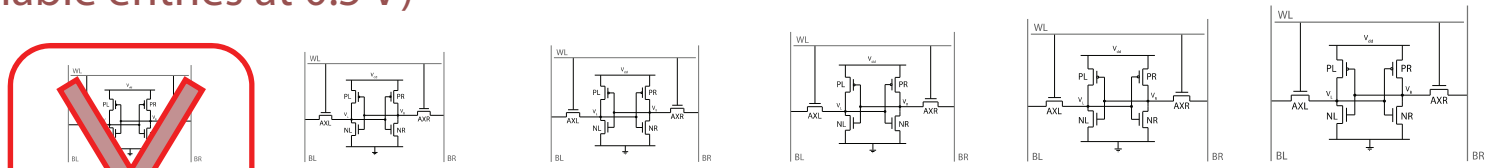
[KH09] R. Kumar and G. Hinton, "A Family of 45nm IA Processors." *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*, 2009

# SRAM Failure Model [ZKG+10]

- 6 6T SRAM cells, 32 nm, target $V_{nth} = 0.5$ V

++ Area
++ Power
++ Reliability

Not considered
(~ 0% available entries at 0.5 V)



| | C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|---|
| Relative Area | 1.00 | 1.12 | 1.23 | 1.35 | 1.46 | 1.58 |
| % non-faulty | 0.0 | 9.9 | 27.8 | 35.8 | 50.6 | 59.9 |

Percentage of 64 B non-faulty cache entries

## System model

- Random fault maps for LLC
- Monte Carlo simulations:
  ensure error < 5 % with confidence (1 - α) = 0.95



[ZKG+10] S. T. Zhou et al., "Minimizing Total Area of Low-voltage SRAM Arrays Through Joint Optimization of Cell Size, Redundancy, and ECC." *IEEE Int. Conf. on Computer Design*, 2010

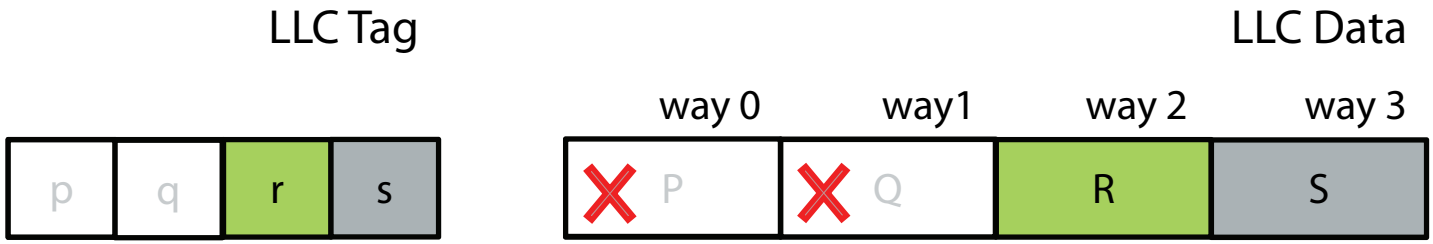
# Experimental Set-up

- Full-system simulation:
  Simics + GEMS + DRAMSim2 + McPAT

- Workloads
  - 20 multiprogrammed mixes:
    Random combinations of the 29 SPEC CPU 2006 applications
  - 4 parallel applications:
    PARSEC applications (sim-large, LLCMPKI >= 1.0)
    `canneal, ferret, streamcluster, vips`

# Coping with SRAM Errors

- Block Disabling (BD) [S89]:
  disable cache entries with faults



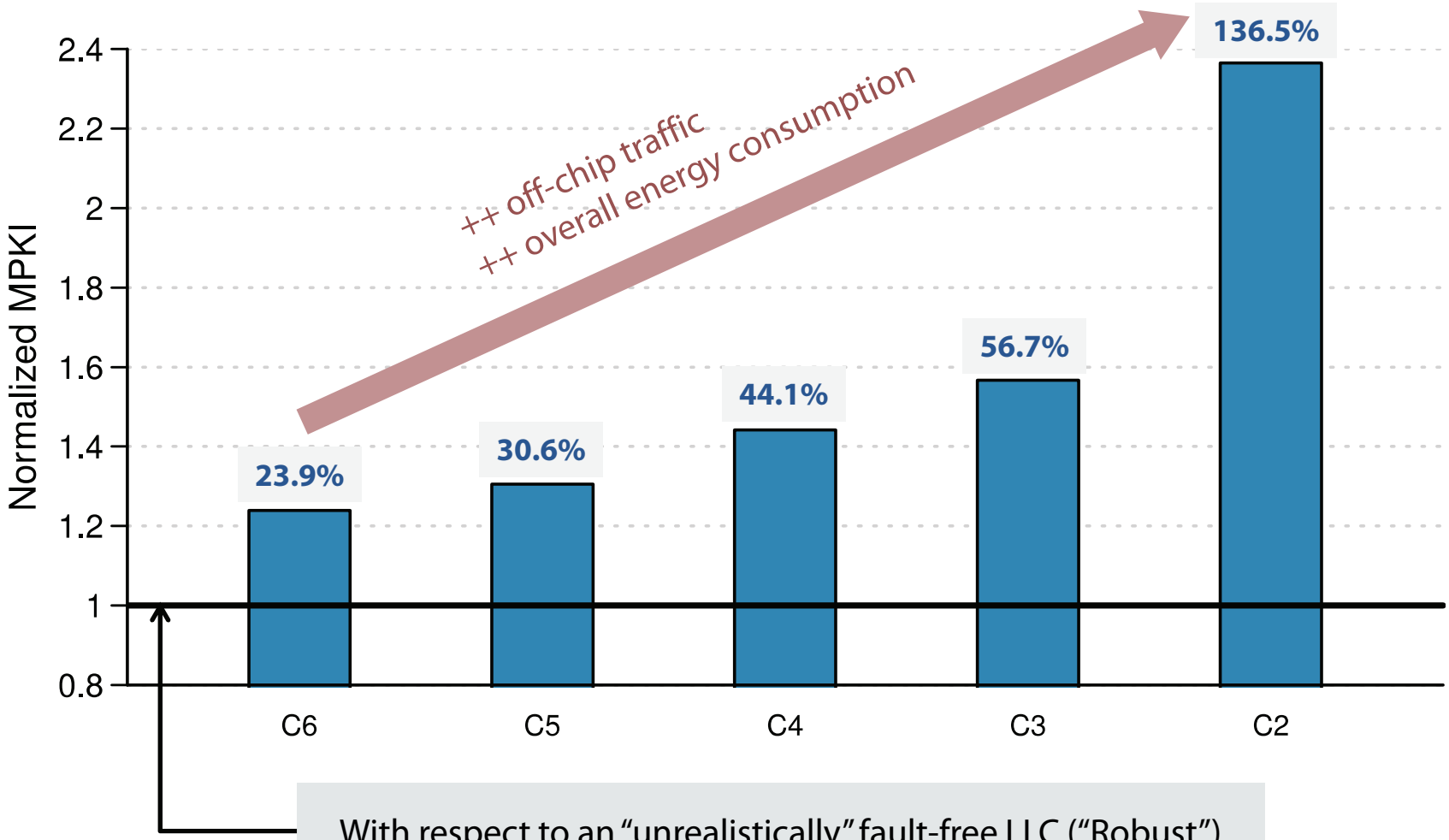

  - Simple and cheap (1 bit)
  - Performance degrades very fast as the number of faulty entries increases



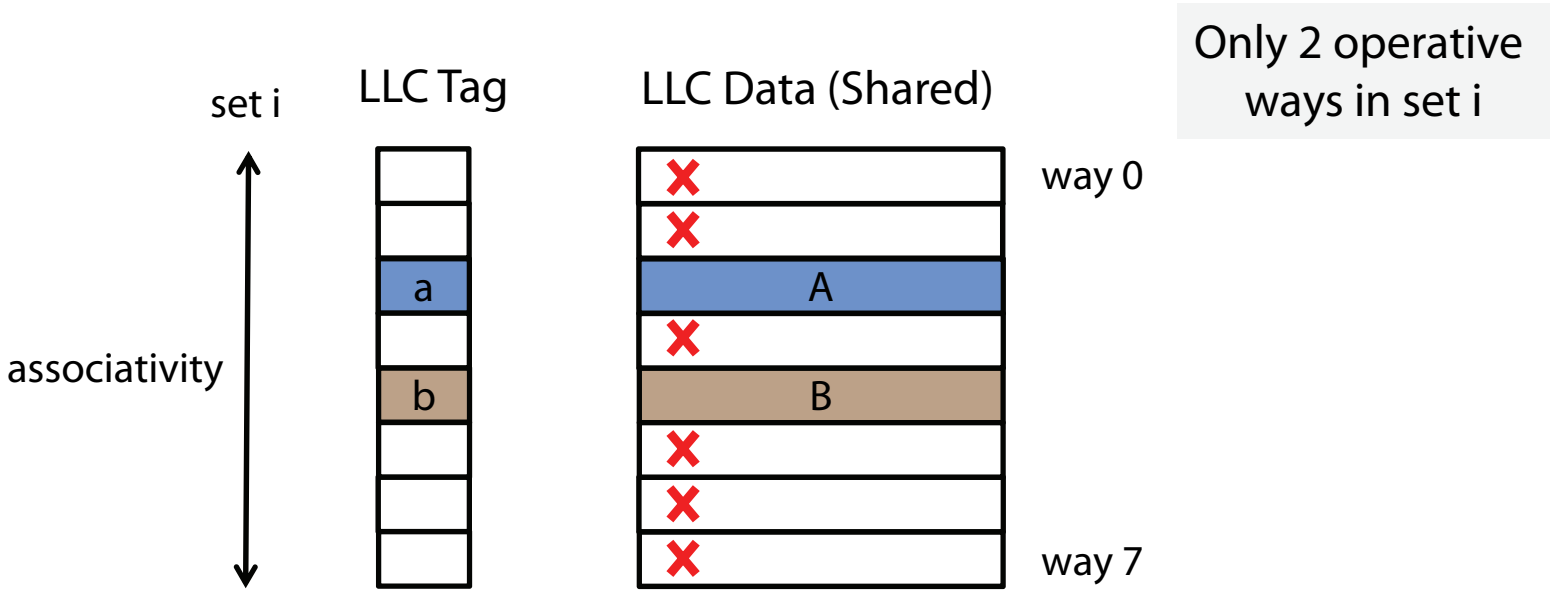
[S89] G. Sohi, "Cache Memory Organization to Enhance the Yield of High Performance VLSI Processors." *IEEE Trans. on Computers,* 1989

# Performance of BD: LLC MPKI

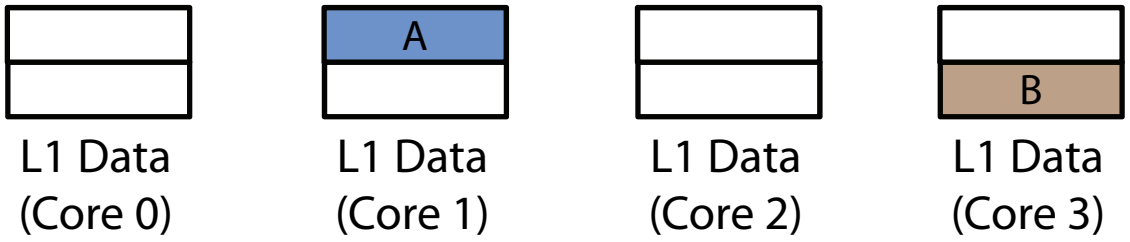
SPEC CPU 2006 multiprogrammed mixes



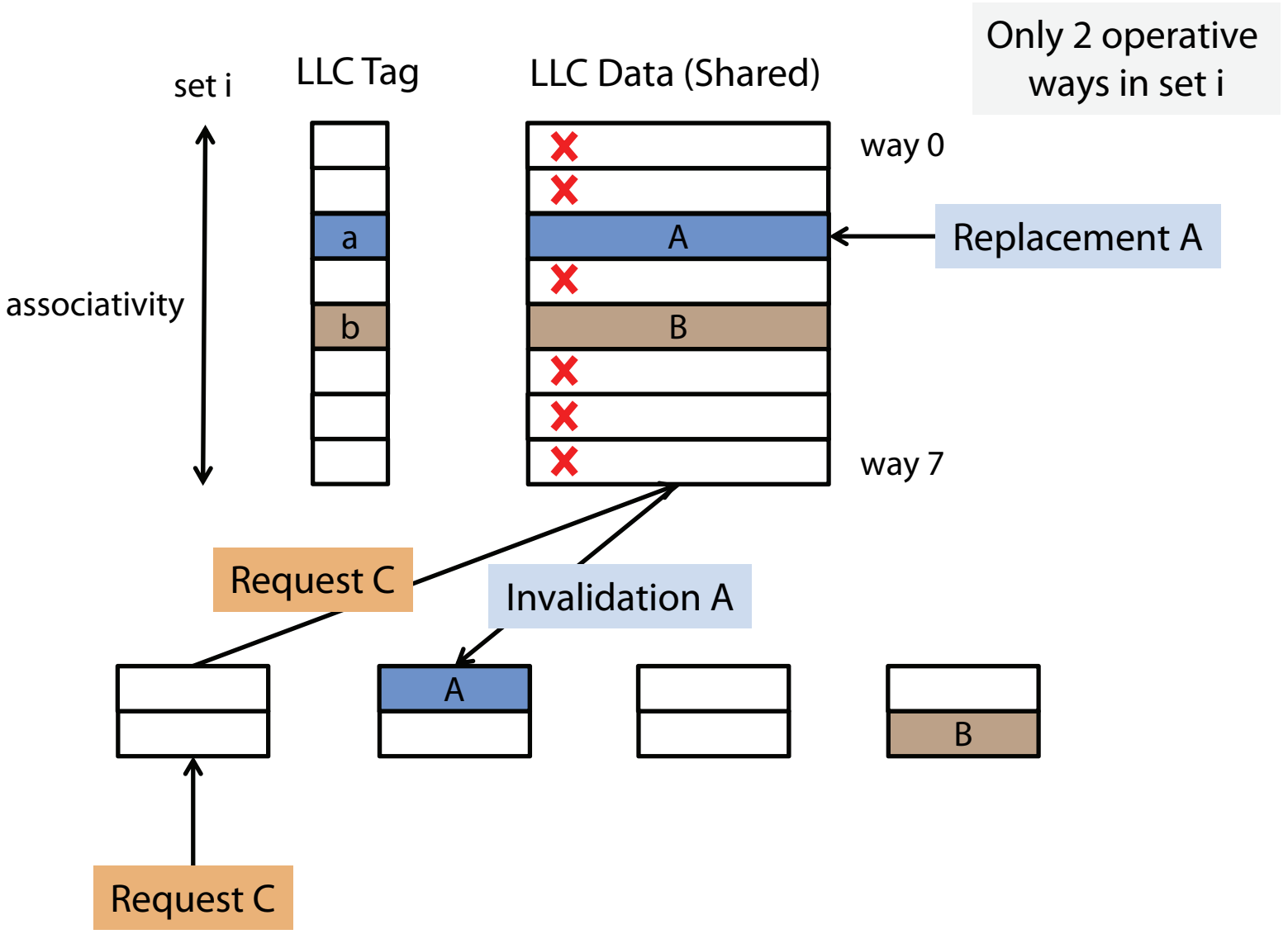
# BD and Inclusion Victims



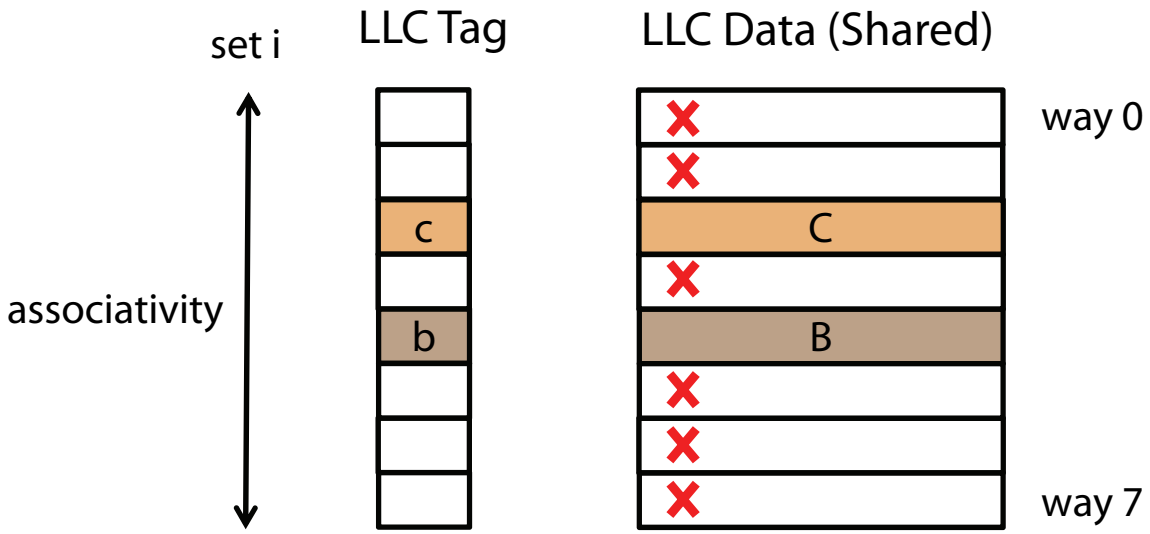

Inclusive hierarchy → all blocks in L1 must be mapped in LLC as well



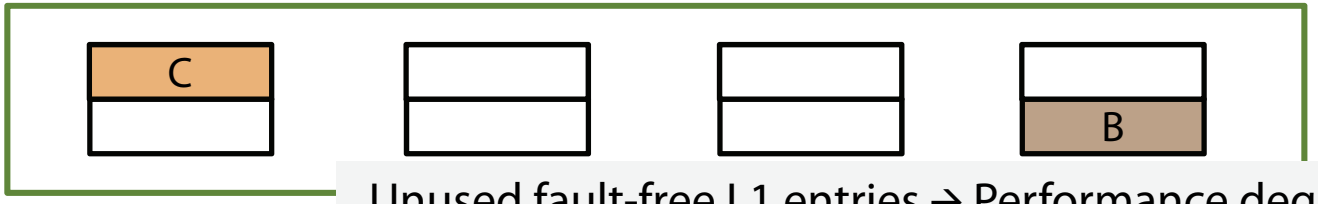
# BD and Inclusion Victims



# BD and Inclusion Victims




The associativity of LLC limits the associativity of L1



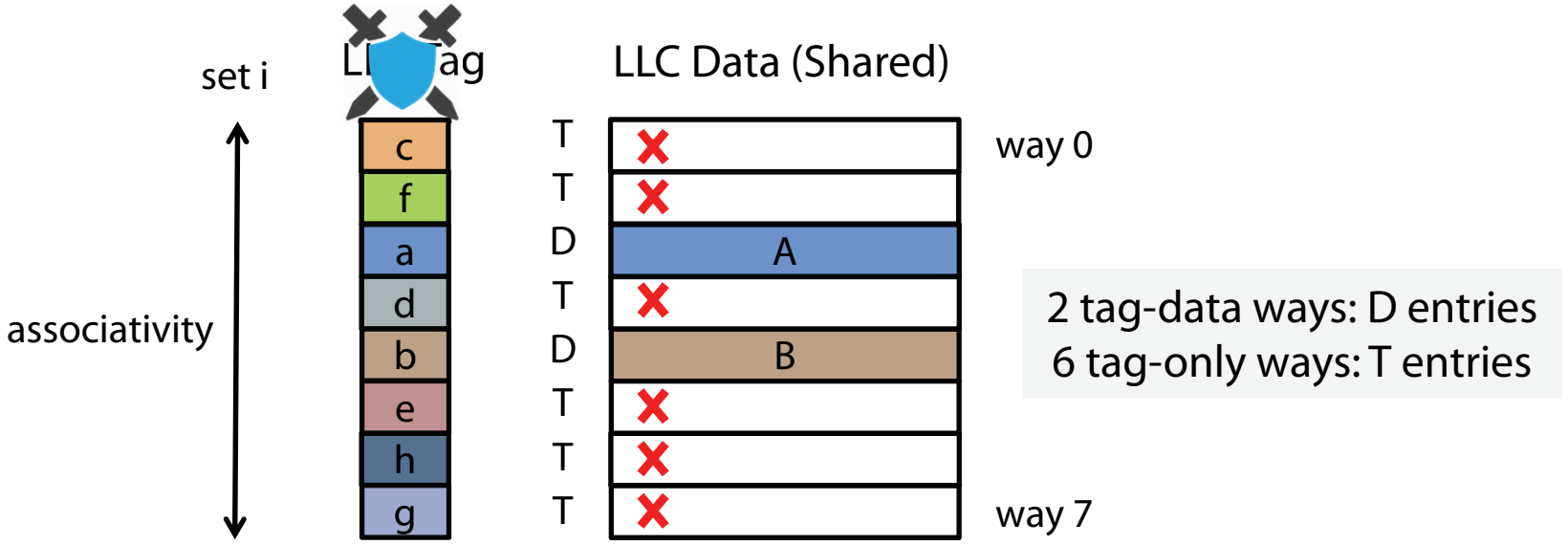

Unused fault-free L1 entries → Performance degrades!

# Avoid Inclusion Victims Increase?

- Observation:
  BD limits associativity of the private levels
  BUT, from the coherence perspective,
  only **directory inclusion** is needed

- Idea:
  Allow blocks to be mapped in L1 by keeping only the tag in the LLC (with its coherence information) via **resilient tag array**
  (e.g., 8T SRAM cells – 2% area overhead)

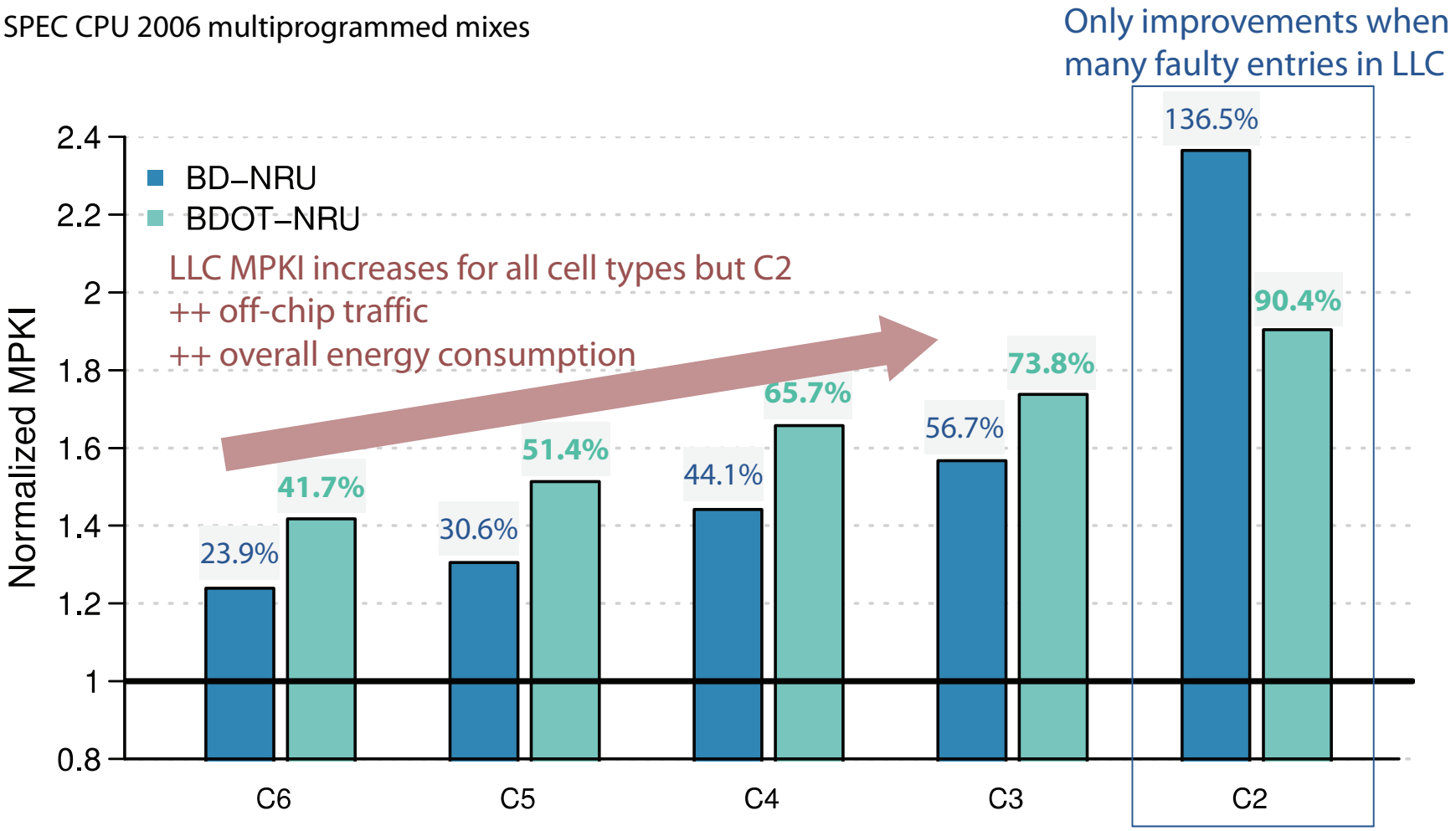
# BDOT: Block Disabling with Operational Tags




- Associativity seen by private levels is restored
- Requests to T entries:
  serviced by L1-to-L1 transfers or forwarded to main memory
- Replacement unchanged:
  no distinction between T and D entries

# Performance of BDOT: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes



# BDOT Limitations

- BDOT only improves performance when there are many faulty entries in the LLC

- Observation:
  A block used more than once allocated to a T entry **always** misses in LLC data array,
  while BD would allocate a data entry for it

- Idea:
  "Useful" blocks should be allocated to D entries
  → Maximize on-chip **hits** and on-chip **content**

# Cache Management Policy for BDOT

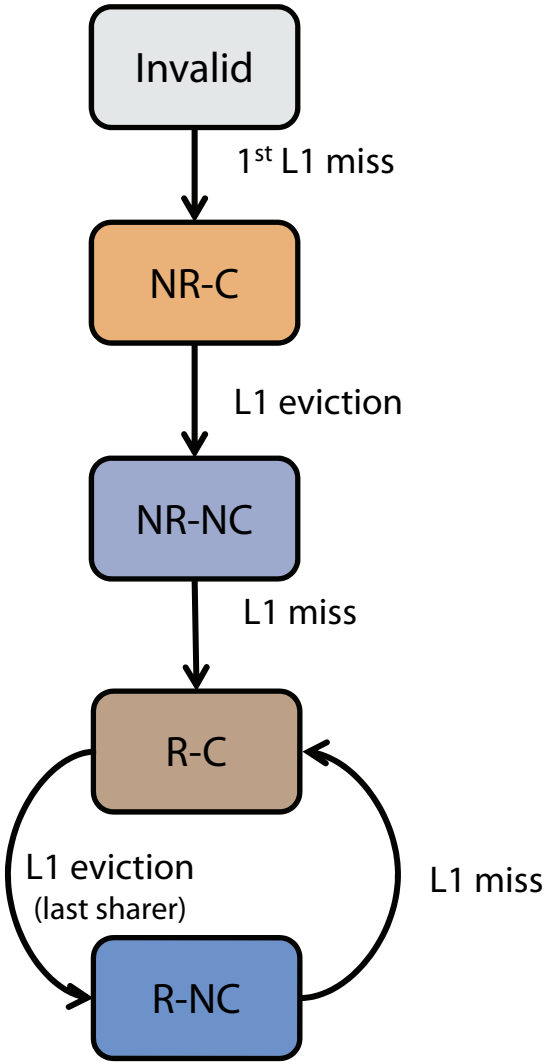
- Maximize on-chip hits:
  Allocate blocks more likely to be used in the near future to D entries → **Reuse** as future use detector:
  - Reuse locality in LLC [AIV+13]:
    Blocks accessed at least twice tend to be reused many times in the near future, and recently reused blocks are more useful than those reused earlier

- Maximize on-chip content:
  Give more priority (higher chances to be allocated to D entries) to blocks not present in L1
  - Blocks in L1 can be serviced through L1-to-L1 transfer



[AIV+13] J. Albericio et al., "Exploiting Reuse Locality on Inclusive Shared Last-level Caches."
*ACM Trans. on Architecture and Code Optimization,* 2013

# Reuse and L1 Presence Tracking

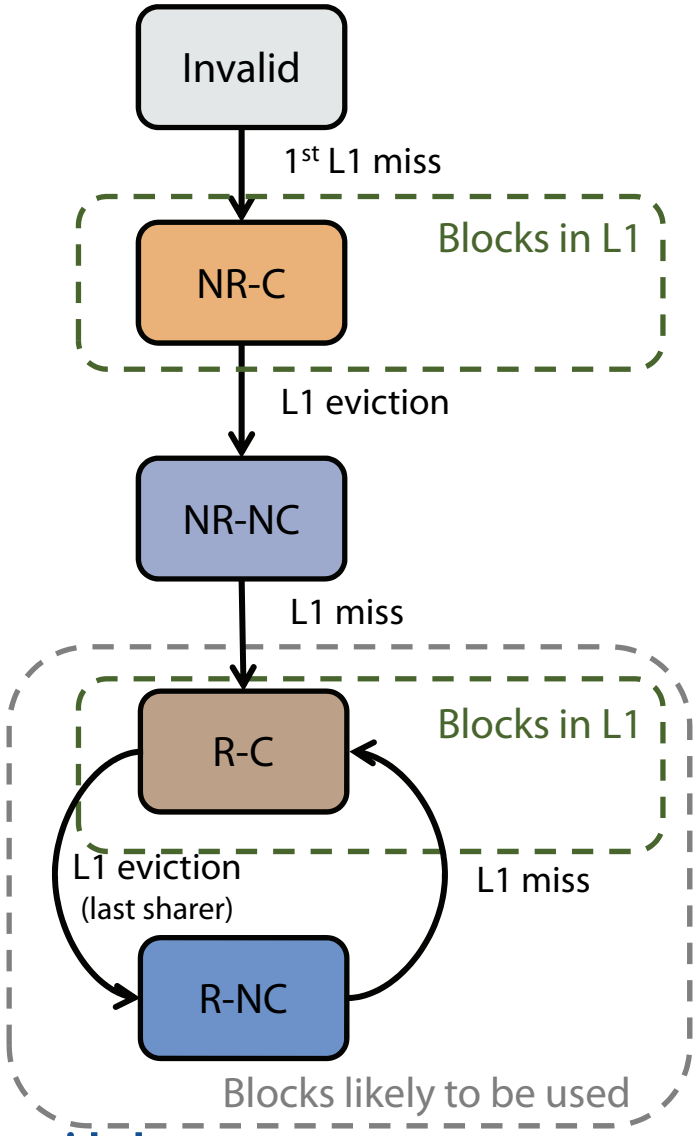



- NRR (Not-Recently Reused)[AIV+13]
  - Track Reuse: NR/R (reuse bit)
  - Track L1 presence: NC/C (L1 presence vector)
- 4 states:
  - NR-C: NonReused-Cached
  - NR-NC: NonReused-NonCached
  - R-C: Reused-Cached
  - R-NC: Reused-Cached

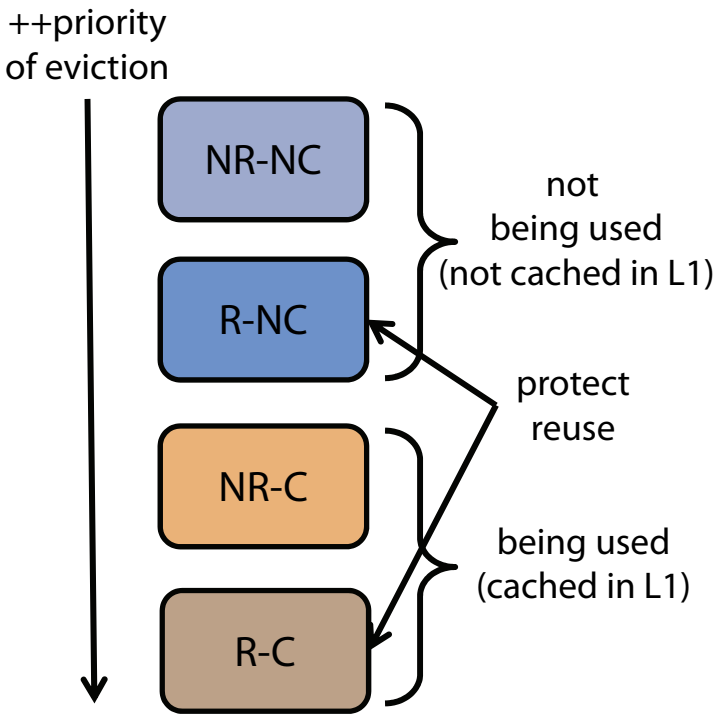


[AIV+13] J. Albericio et al., "Exploiting Reuse Locality on Inclusive Shared Last-level Caches." *ACM Trans. on Architecture and Code Optimization,* 2013

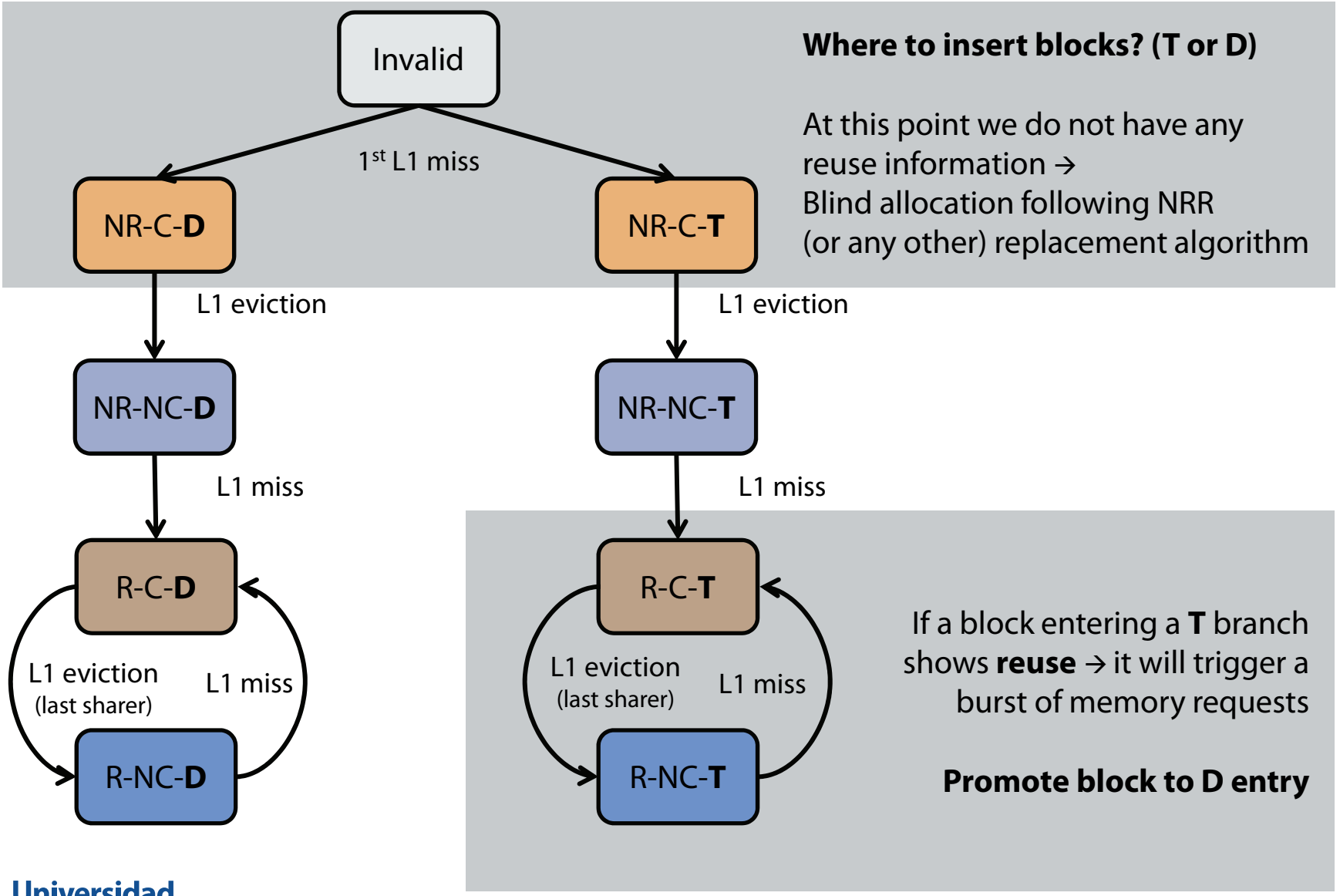
# Reuse and L1 Presence Tracking



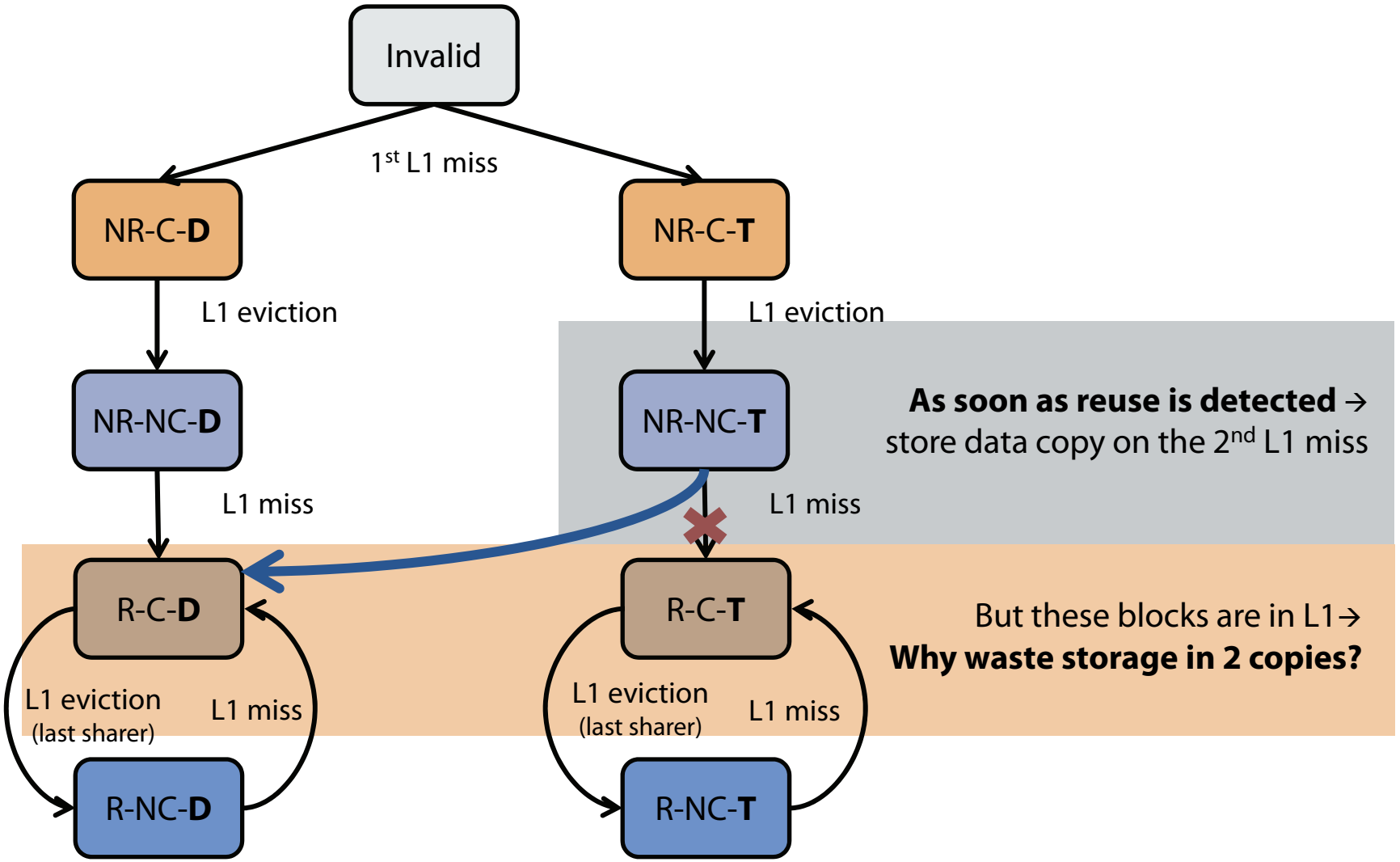

LLC Victim Selection



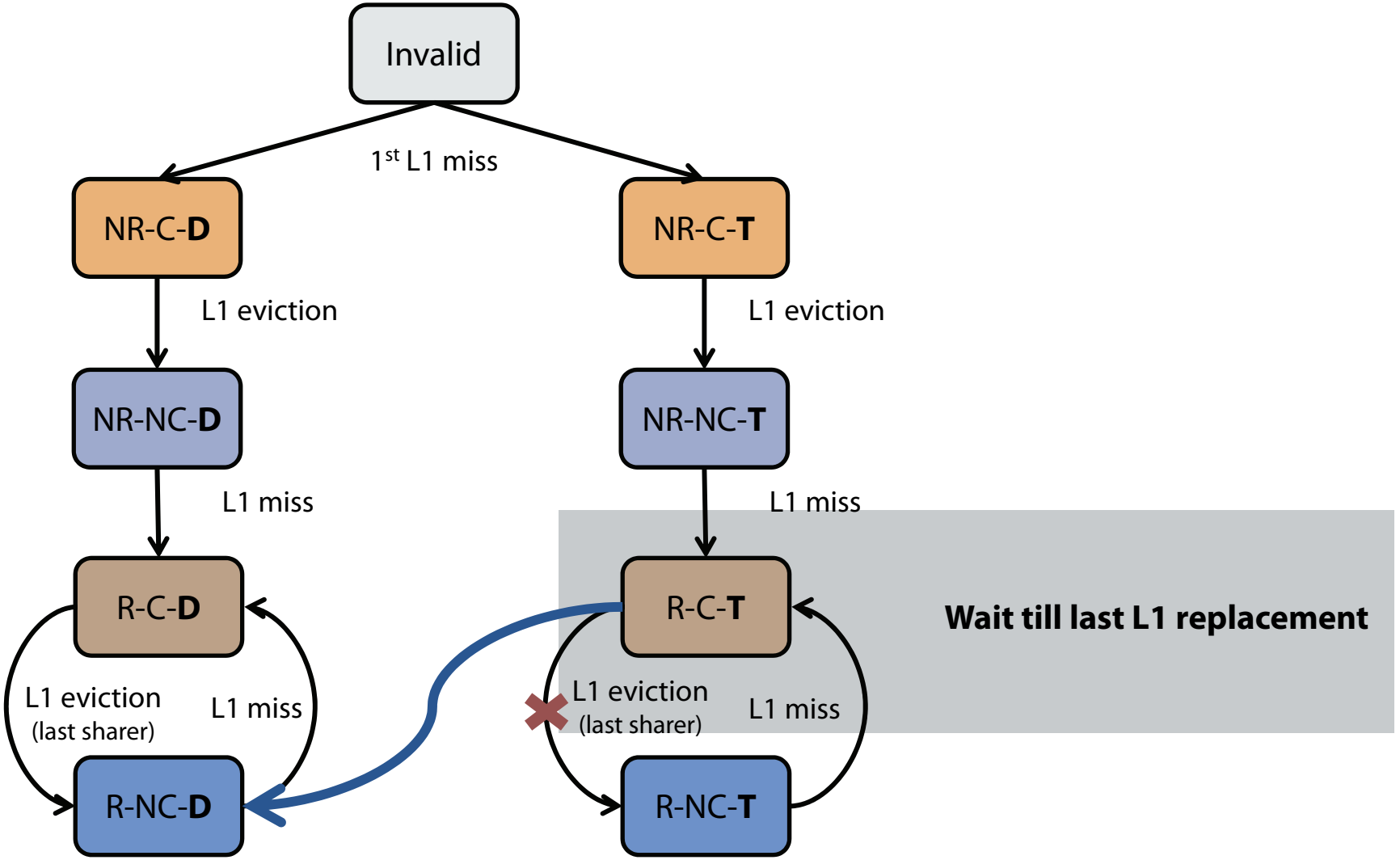
# Fault-aware Cache Management



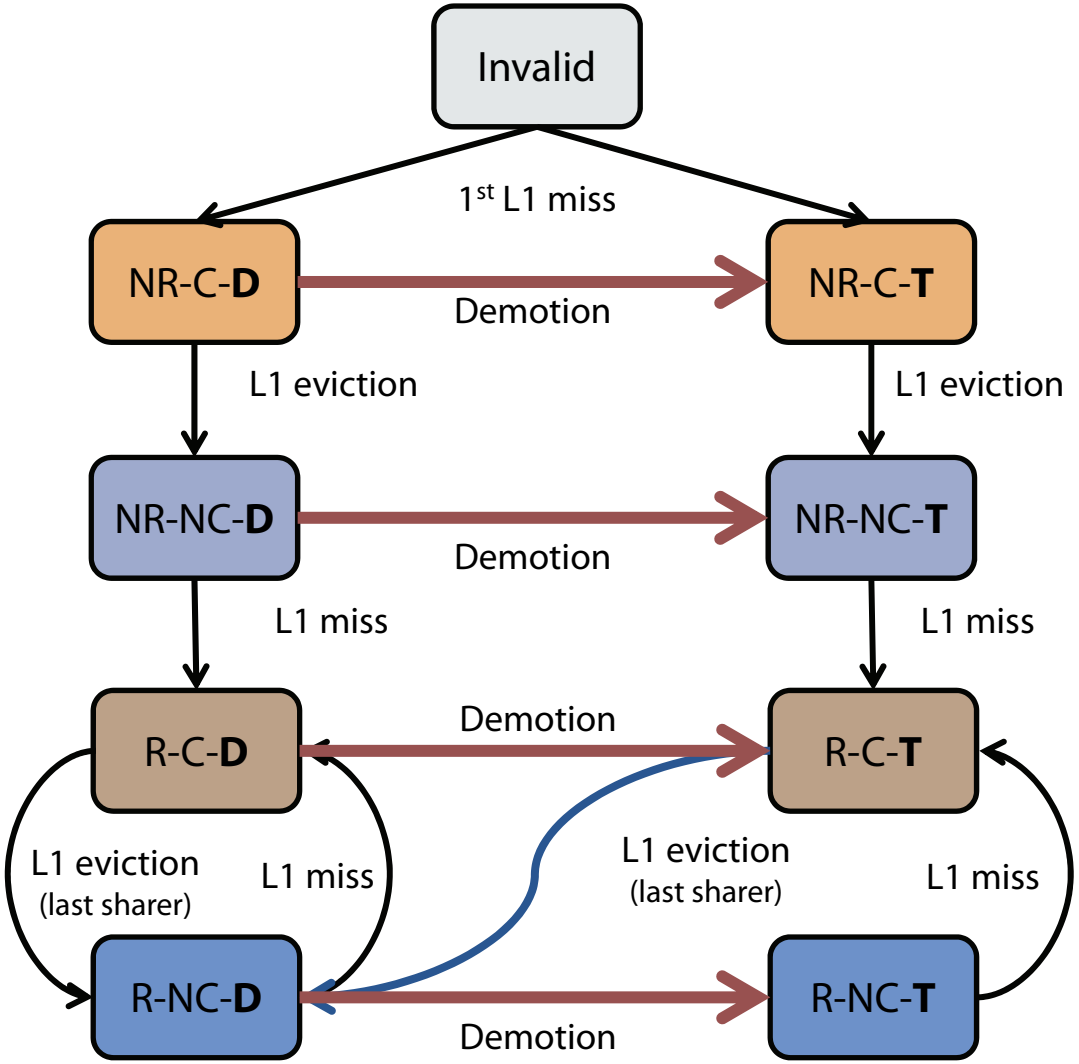
# When to Promote Blocks?



# When to Promote Blocks?

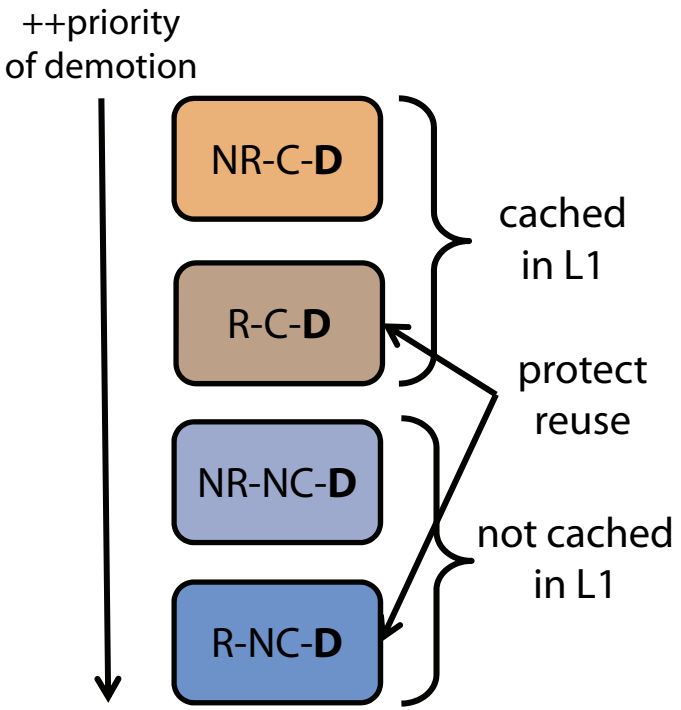


# Promotions Necessarily Trigger Demotions

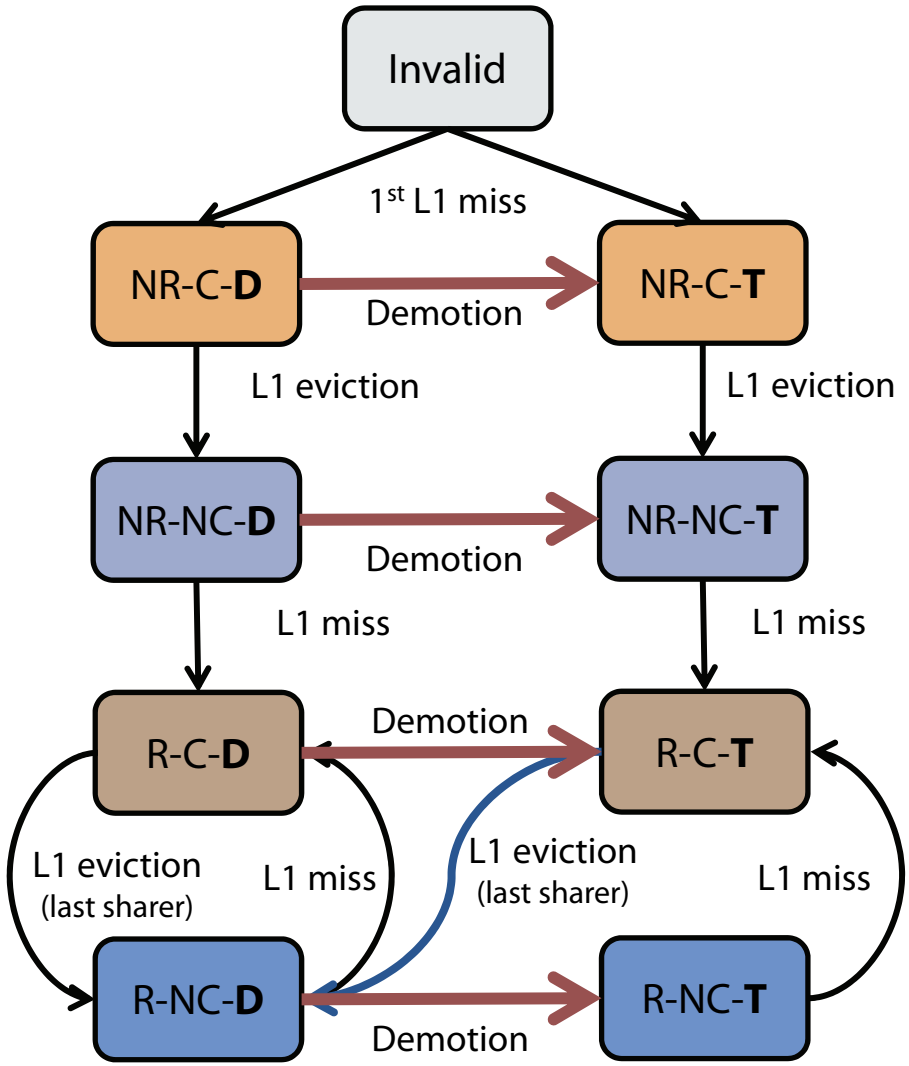


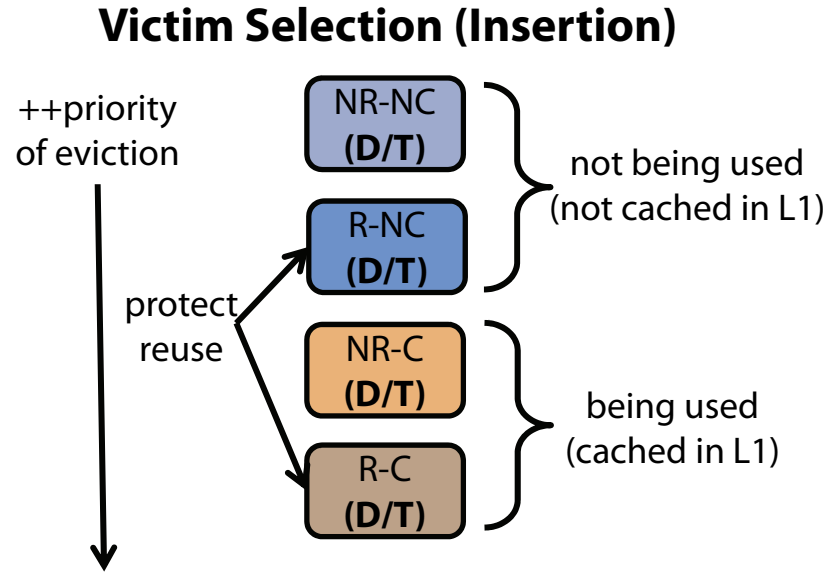

Demotion: D → T

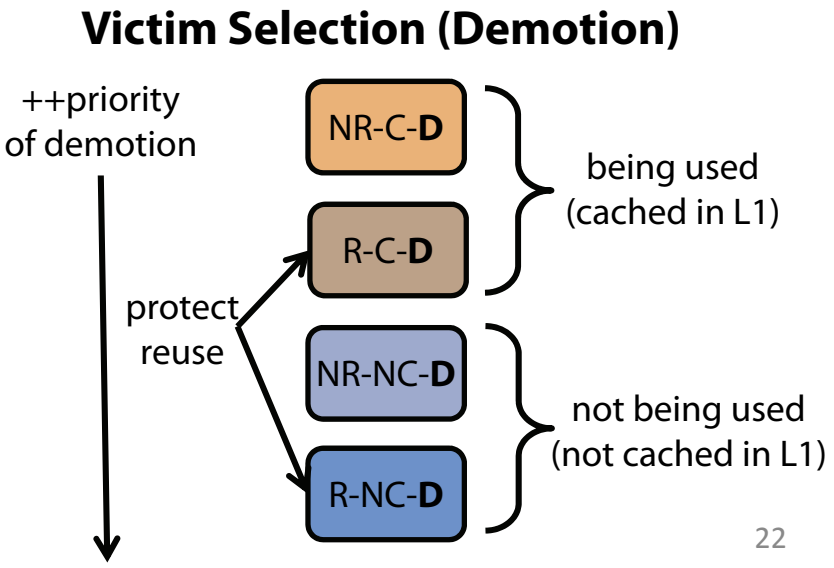
**Which block to demote?**



# BDOT-FA Cache Management Policy



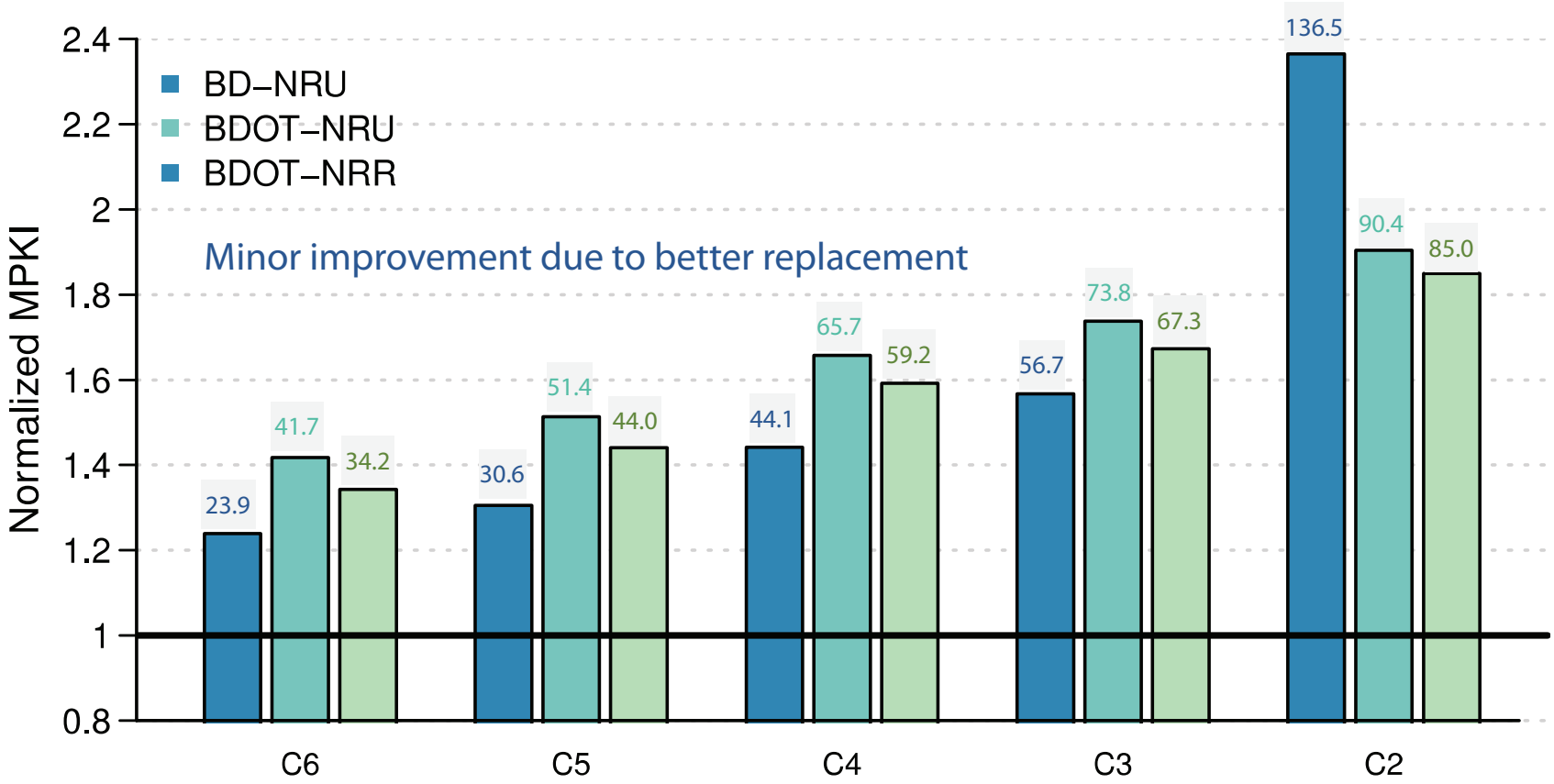




Universidad Zaragoza

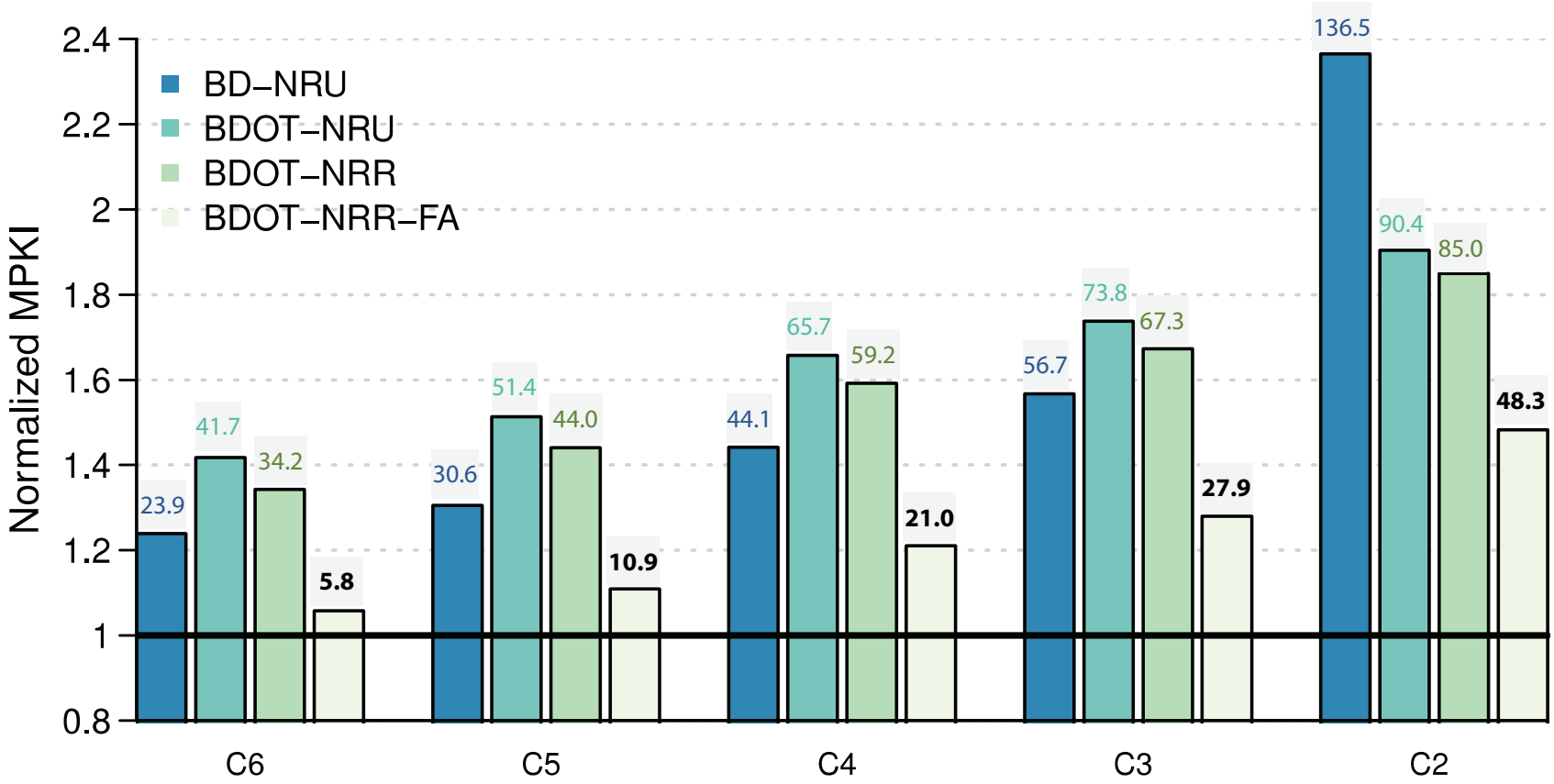
# Performance of BDOT with NRR: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes



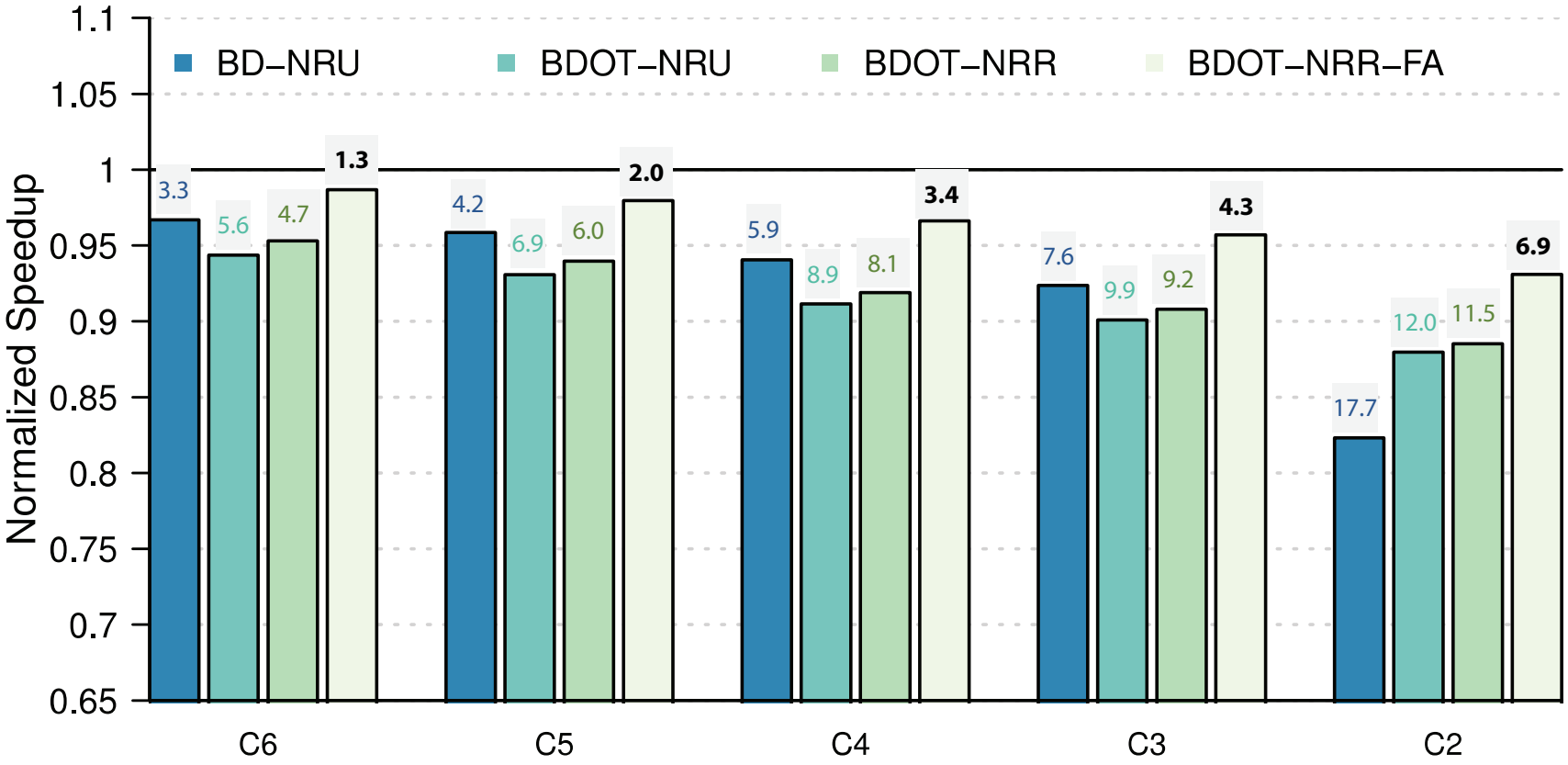
# Performance of BDOT-FA: LLC MPKI

SPEC CPU 2006 multiprogrammed mixes



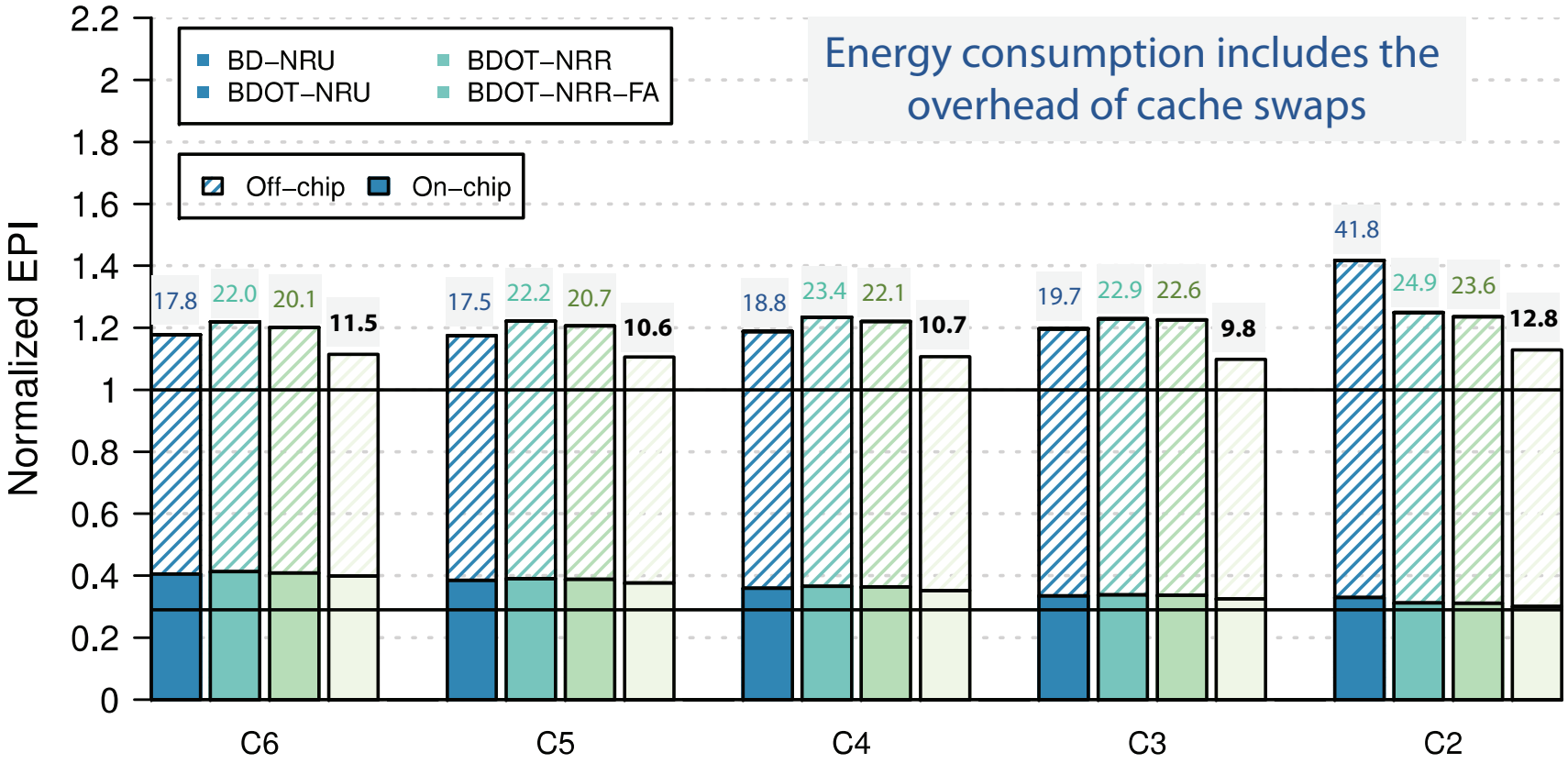
# Performance of BDOT-FA: System Speedup

SPEC CPU 2006 multiprogrammed mixes



# Energy Consumption

SPEC CPU 2006 multiprogrammed mixes




Assuming "Robust" has the same area/power as C2

# BDOT-FA: Summary

- BD limits capacity and associativity of shared cache
  - Limits the associativity of the private caches as well!
  - Inclusion victims increase
- BDOT: Use the tags of the faulty entries → restore LLC associativity
  - Only improves performance when many faulty entries
- BDOT-FA: Fault-aware cache mgt. policy for BDOT
  - Smart allocation of blocks to entries T/D based on reuse and L1 presence
  - Performance speedup with respect to BD:
    - 2.1 – 13.1% (multiprogrammed)
  - Low overhead:
    - Based on BD, does not add additional storage overhead
    - Logic for cache swaps

# A Fault-Aware Cache Management Policy for CMPs Operating at Ultra-Low Voltages

Alexandra Ferrerón-Labari, Jesús Alastruey-Benedé,
Darío Suárez Gracia, Teresa Monreal-Arnal,
Pablo Ibáñez-Marín, Víctor Viñals-Yúfera

Grupo de Arquitectura de Computadores
Departamento de Informática e Ingeniería de Sistemas
Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad de Zaragoza


Instituto Universitario de Investigación
de Ingeniería de Aragón
Universidad Zaragoza

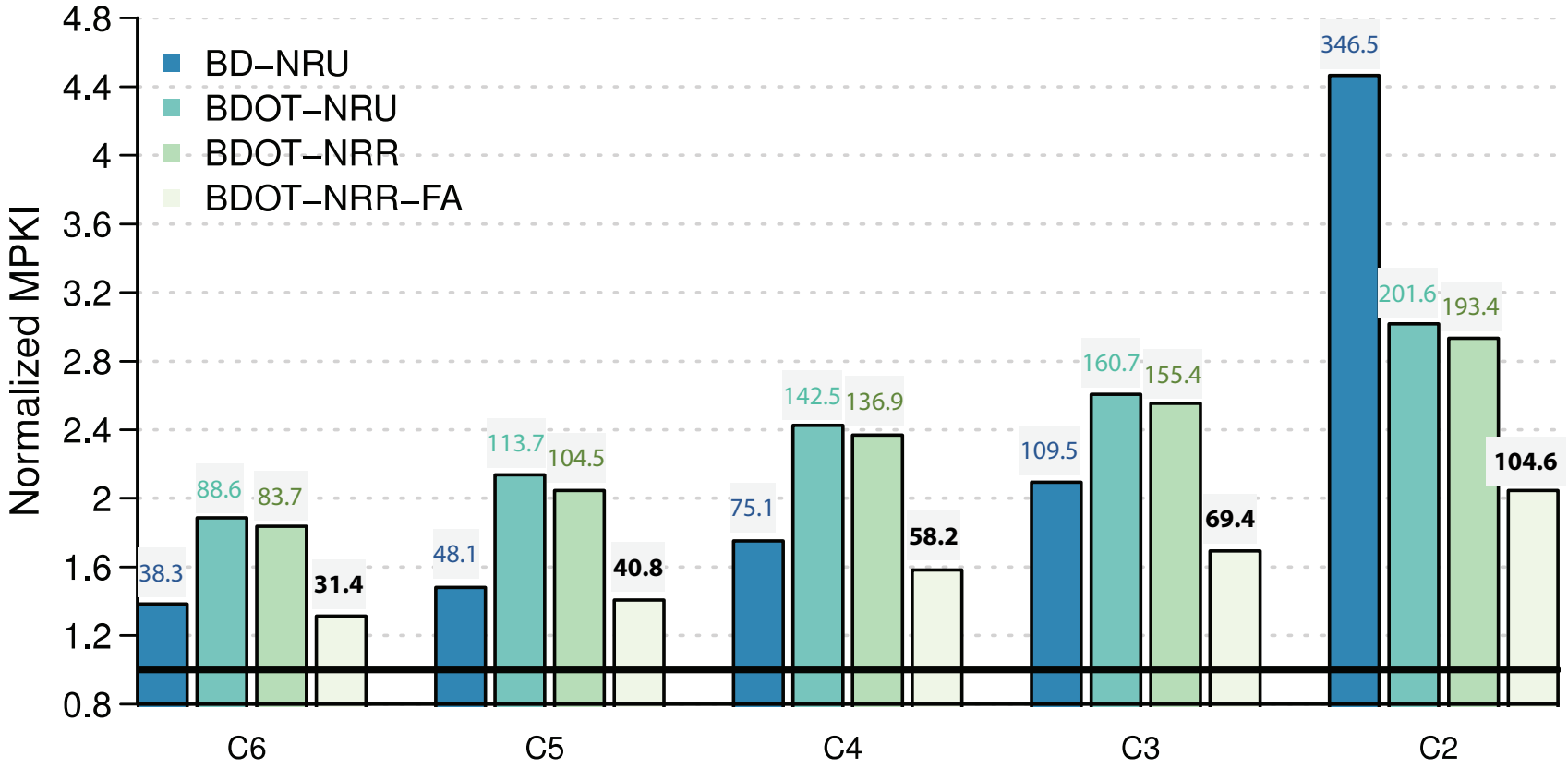
# Back-up
# BDOT-FA

# Putting All Together: BDOT-FA

- Fault-aware cache mgt. policy on top of BDOT
- Insertion:
  - Blind: Rely on replacement algorithm
- Block reuse detection on T entry:
  - Trigger block promotion to D entry (cache swap)
  - Promotion implies demotion of a block from D to T entry: first blocks in L1, protect reuse
- No storage overhead:
  - Reuse bit from replacement policy
  - Presence vector from coherence protocol (no silent L1 evictions)
  - Cache swap logic

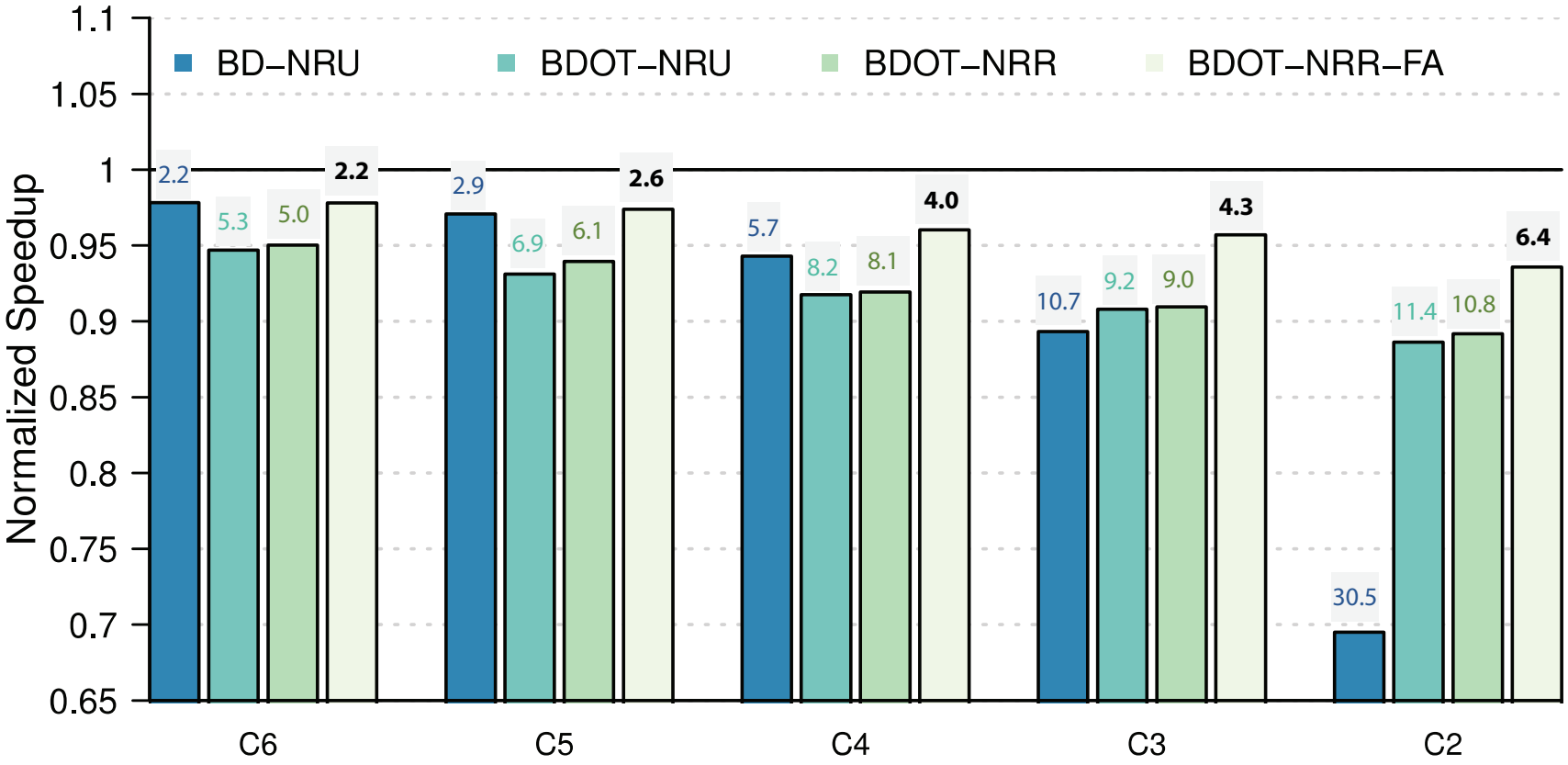
# BDOT-FA: Parallel Workloads Performance

PARSEC applications



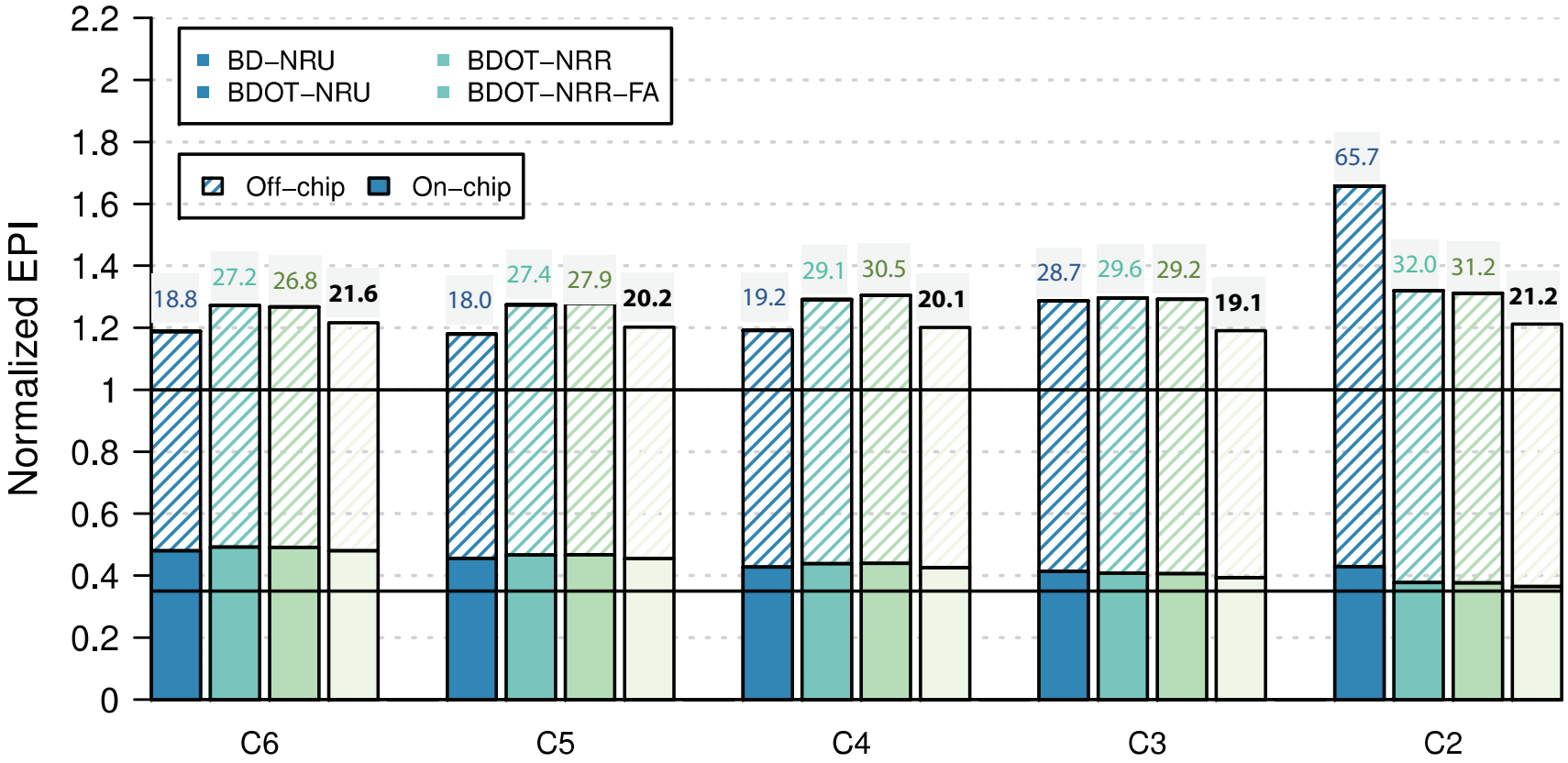
# BDOT-FA: Parallel Workloads Performance

PARSEC applications

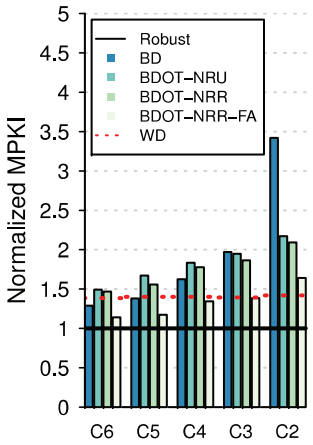


# BDOT-FA: Energy Consumption

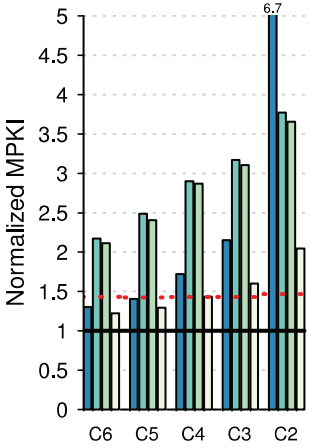
PARSEC applications



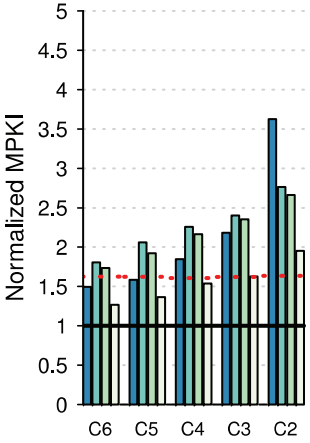
# BDOT-FA: Parallel Workloads Performance



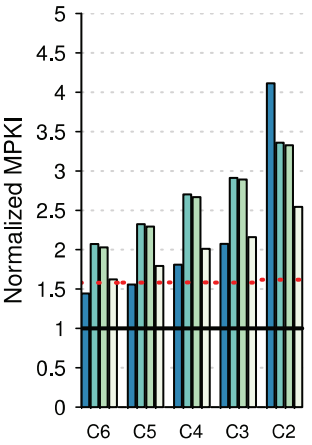

canneal




ferret




streamcluster




vips