The Reuse Cache

Downsizing the Shared Last-Level Cache

Jorge Albericio[1], Pablo Ibáñez[2], Víctor Viñals[2], and José M. Llabería[3]

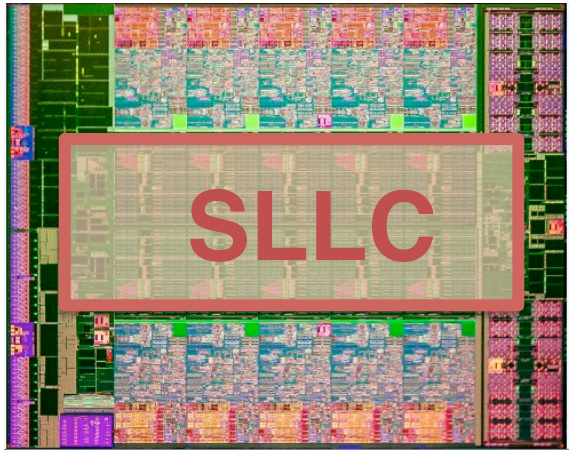
1 UNIVERSITY OF TORONTO

2 Universidad Zaragoza 1542

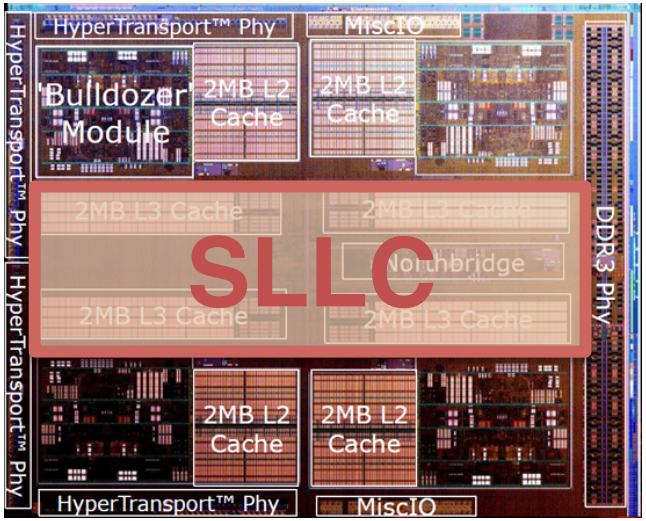
3 UPC

Modern CMPs

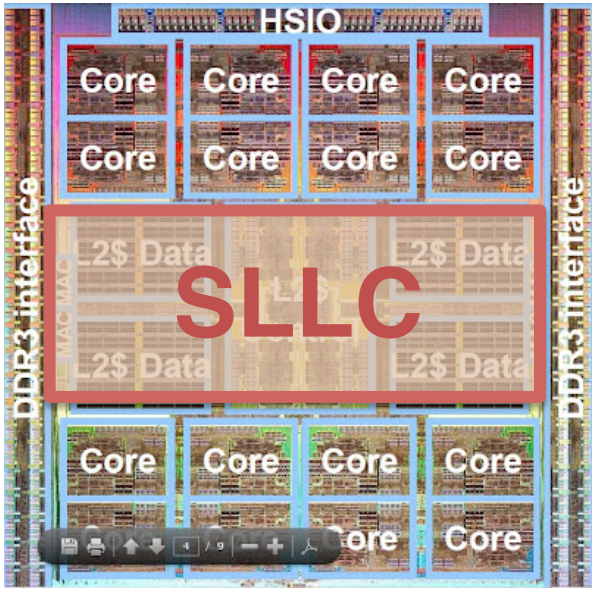
Intel e5 2600 (2013)



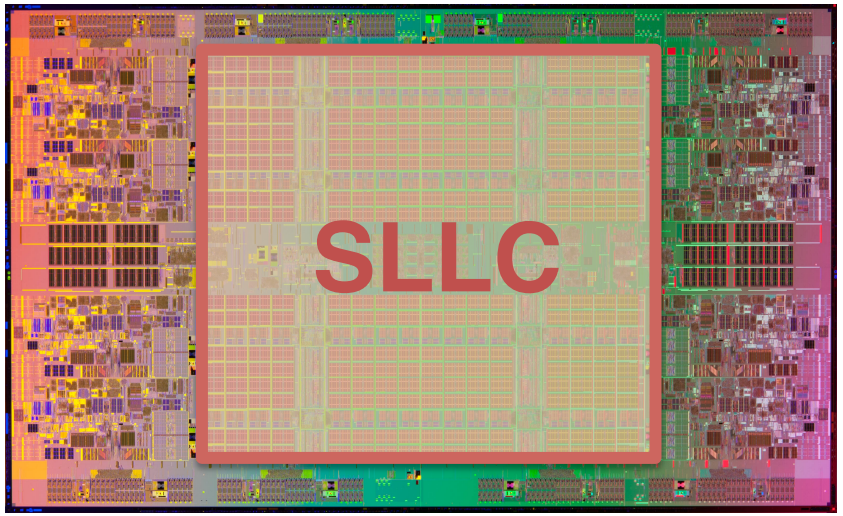
AMD Orochi (2012)



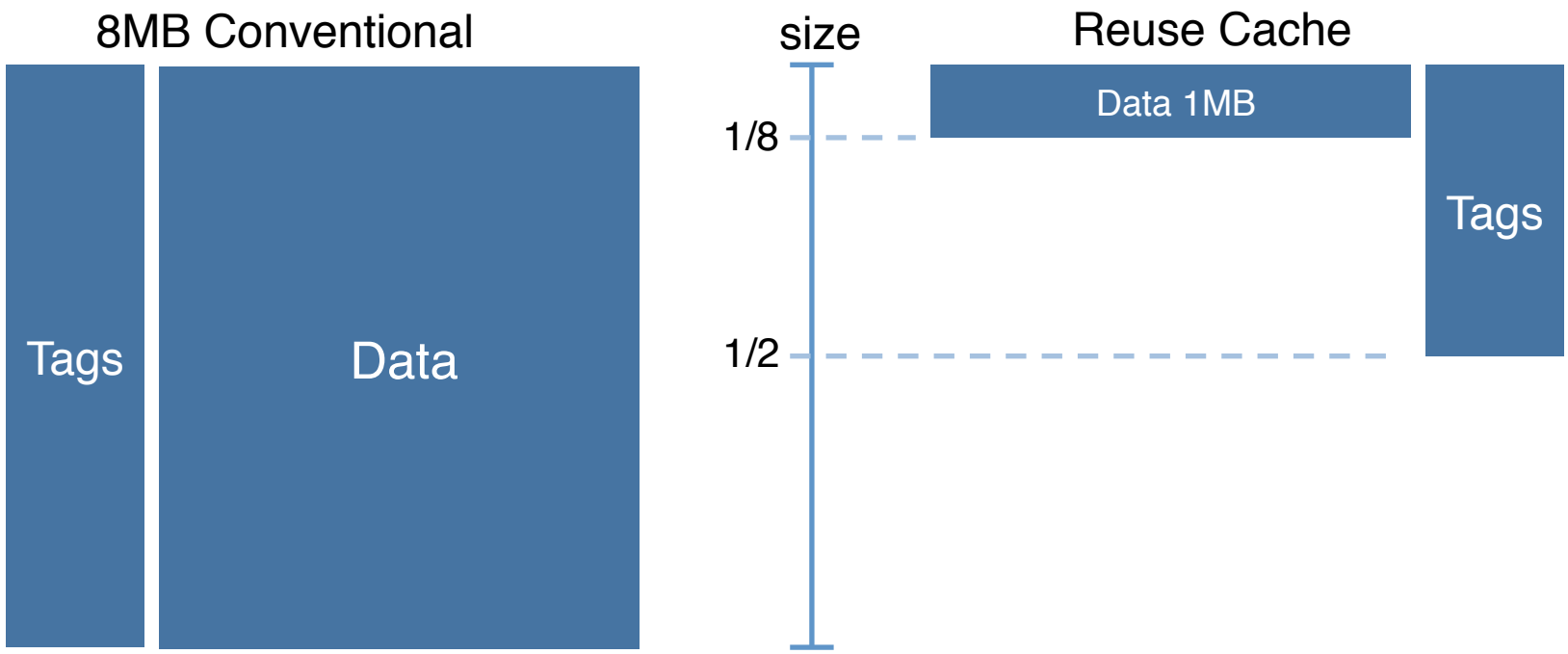
Fujitsu SPARC VIIIfx (2011)



Intel Itanium 9500 (2012)



Inclusive Shared Last-Level Cache (SLLC)



Same average performance* with 84% area savings

*100 multiprogrammed SPEC CPU 2006 workloads in an 8-core CMP

Outline

- Motivation
- The reuse cache
  - Idea
  - Organization
  - Coherence
  - Replacement
- Related work
- Evaluation
- Conclusions

Outline

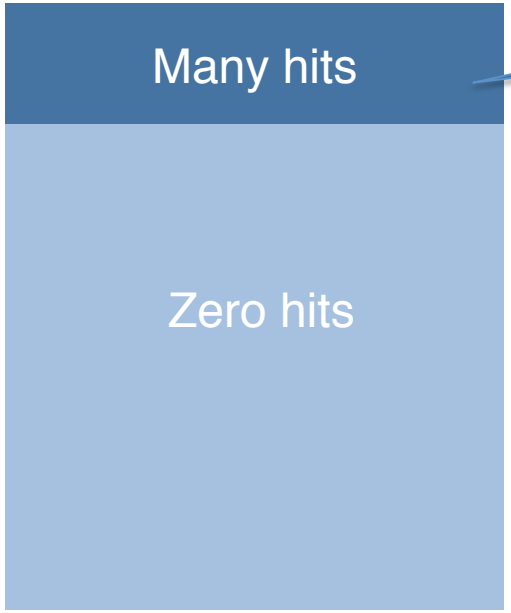
- **Motivation**
- The reuse cache
  - Idea
  - Organization
  - Coherence
  - Replacement
- Related work
- Evaluation
- Conclusions

Motivation

Reuse locality

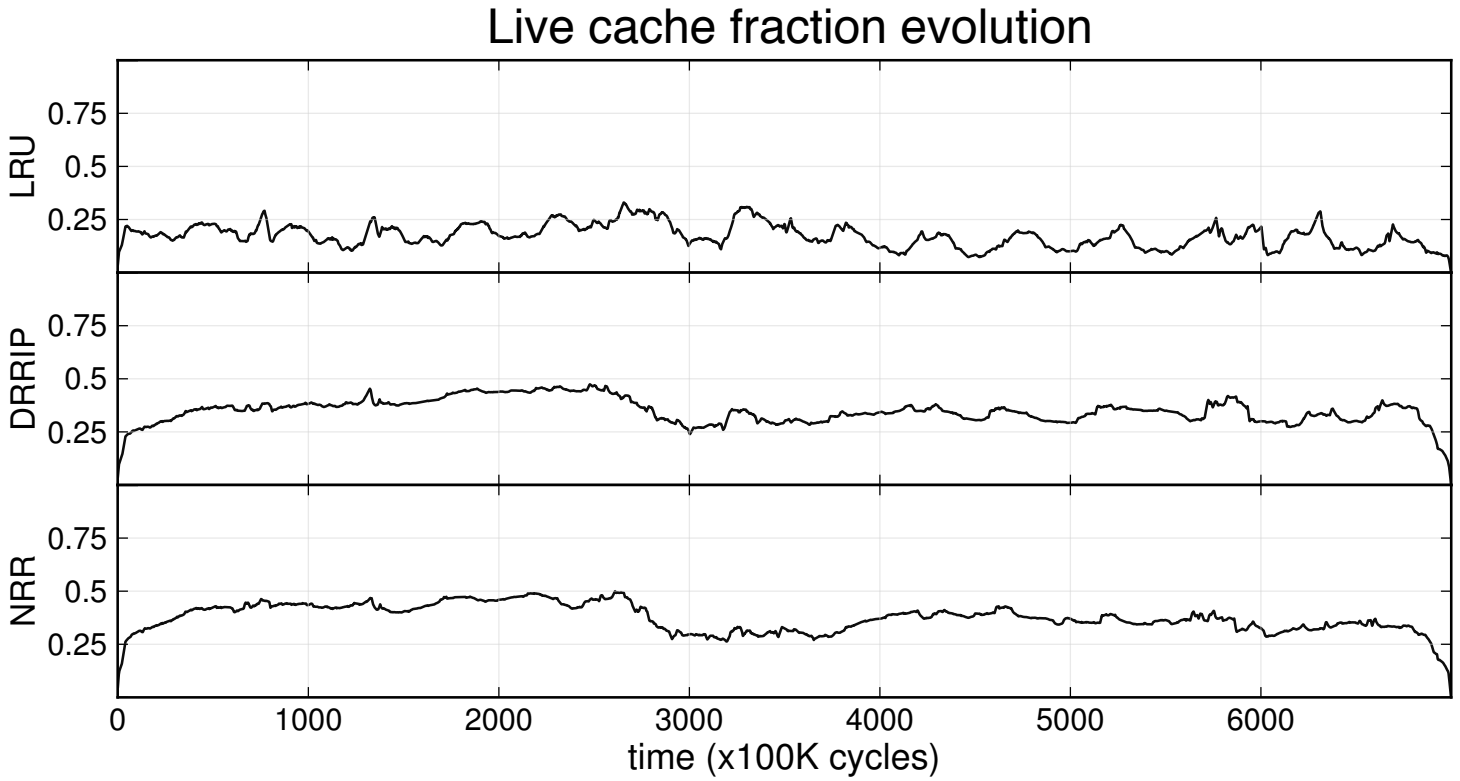
- Used more than one time
  likely to be used many times
- Recently reused lines
  more useful than lines reused before

SLLC data



Opportunity for a **smaller SLLC** which stores only reused data

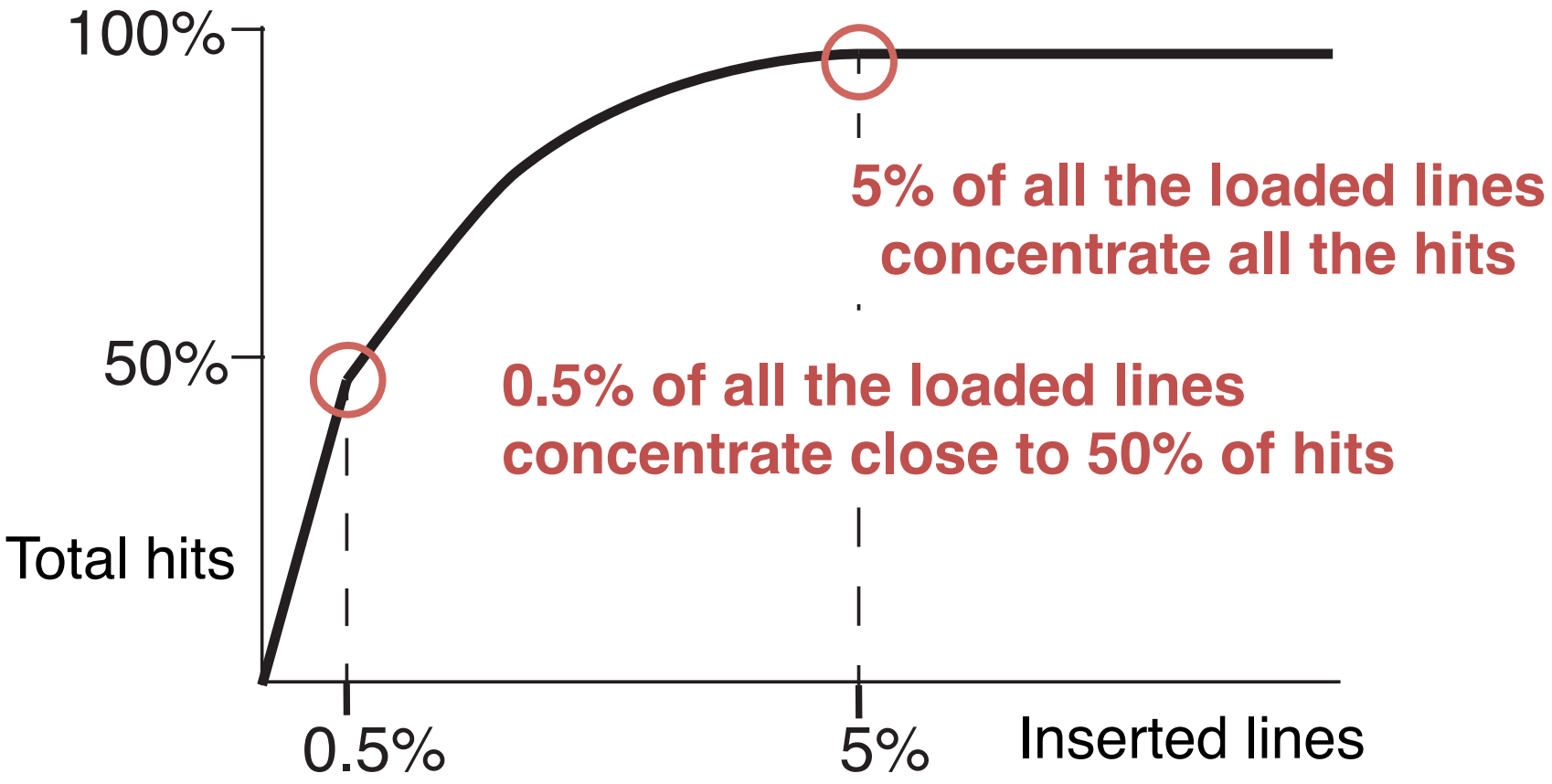
Motivation



- The fraction of live SLLC lines is very small (10-30% LRU)
  - On average 78% of lines won't receive any additional access
- State-of-art replacement policies better

8MB Shared LLC, 16-way, LRU, 8-core CMP, mix #14

Motivation



Most of the inserted lines will not experience any hit because hits concentrate in a few lines

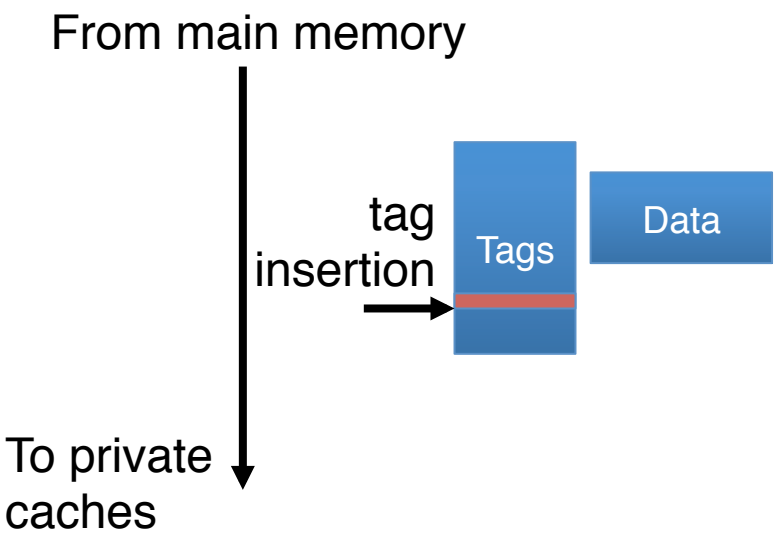
8MB Shared LLC, 16-way, LRU, 8-core CMP, mix #14

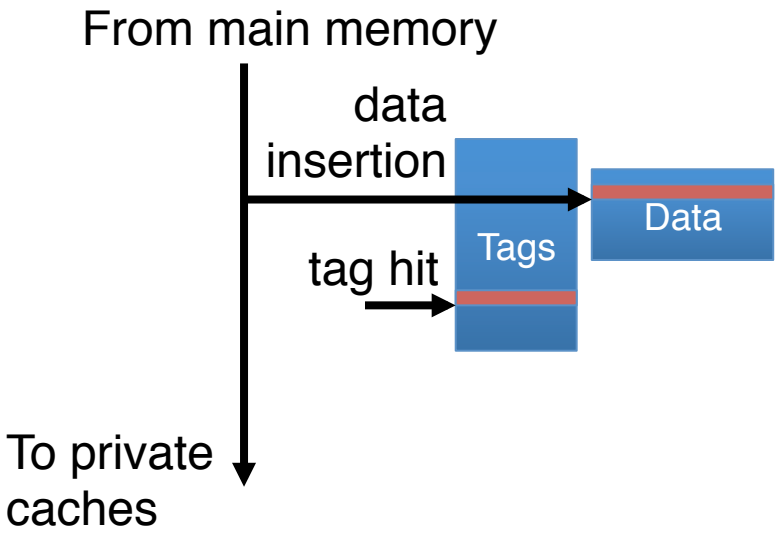
Outline

- Motivation
- **The reuse cache**
  - Idea
  - Organization
  - Coherence protocol
  - Replacement
- Related work
- Evaluation
- Conclusions

Idea

Data is stored only when **reuse** is detected



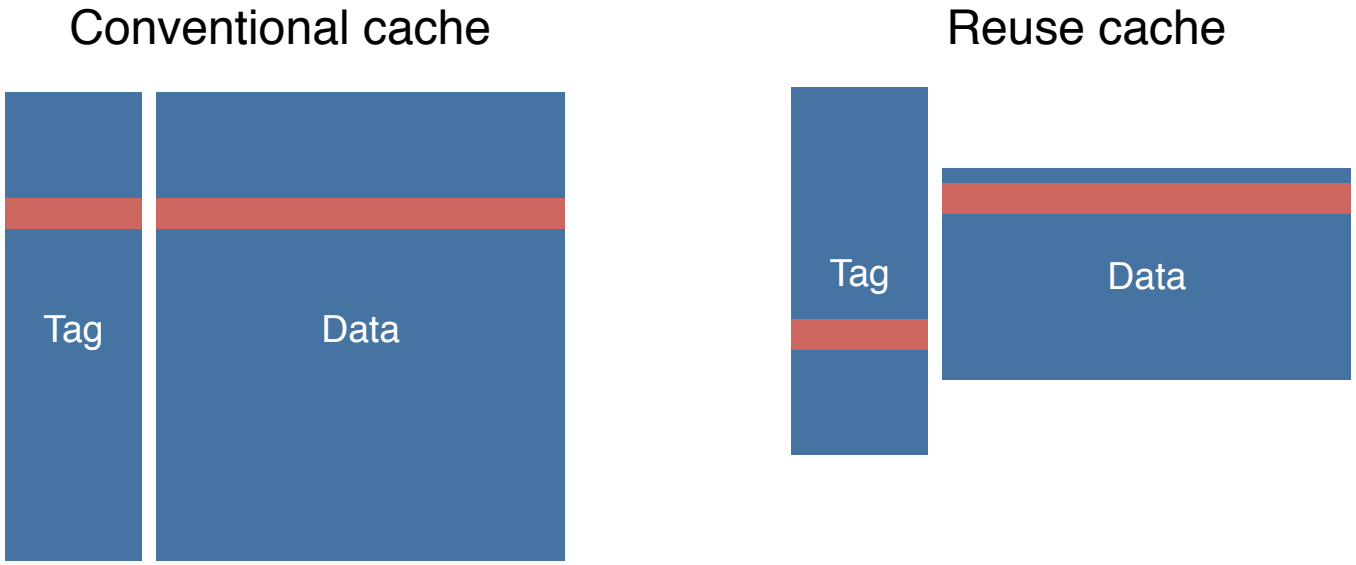


First access:
Miss → only tag insertion

Second access:
Tag hit → data insertion
(reuse detected)

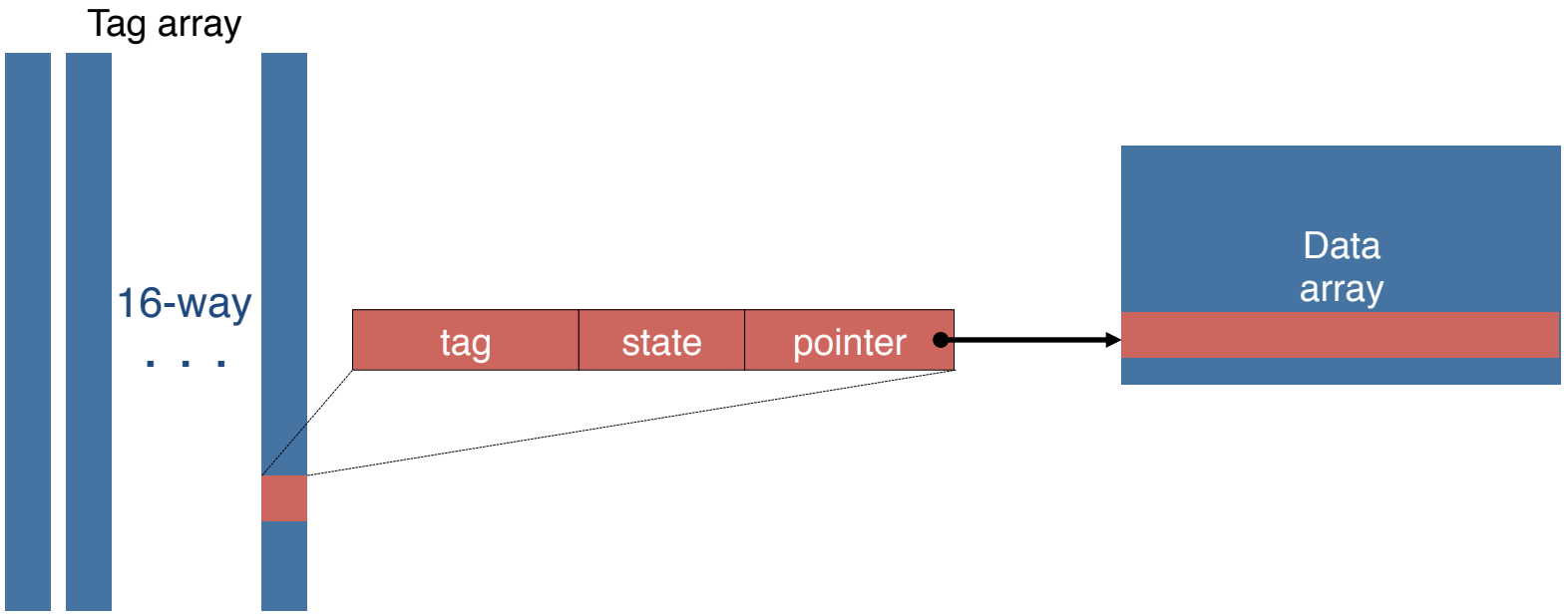
Organization

- Decoupled tag and data arrays
  - More tag than data entries



Organization

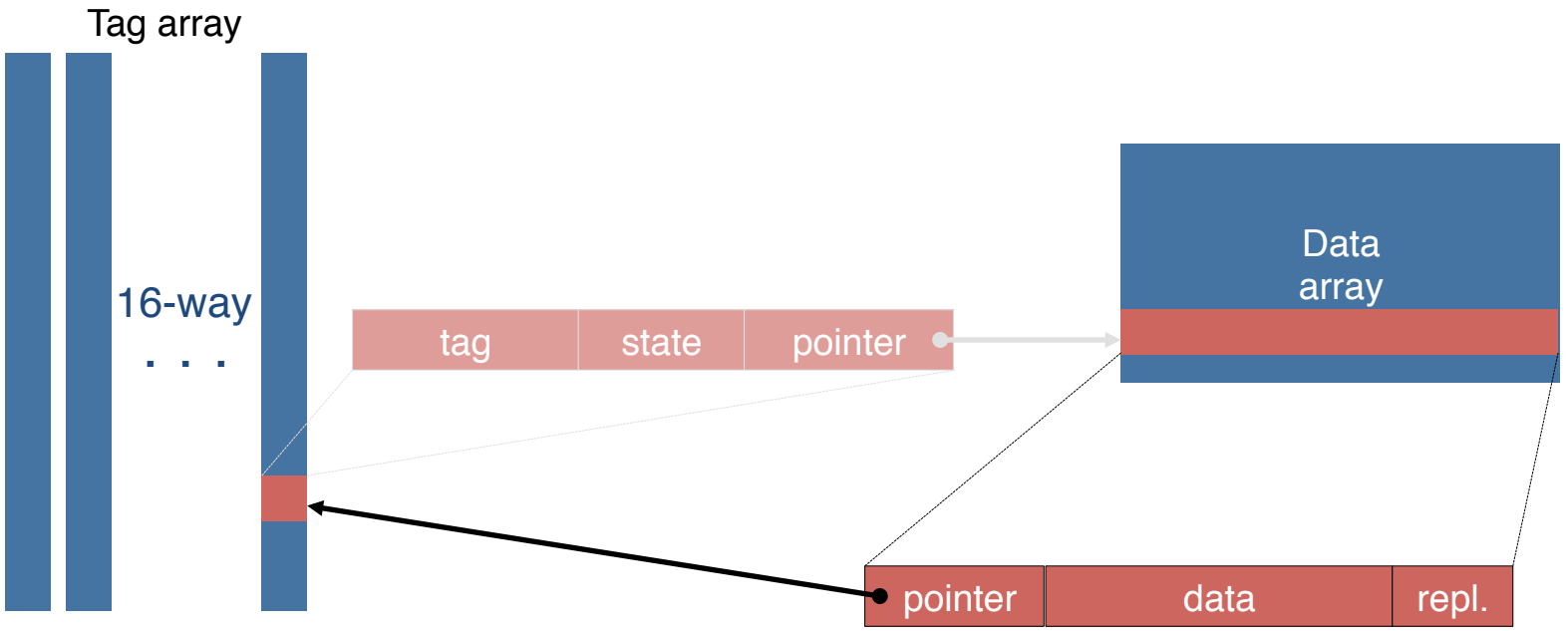
- Tag array
  - NORMAL
    - Maintains coherence/inclusion
    - Set-associative
  - NEW
    - Each entry has associated data or not
    - Forward pointers: to indicate corresponding data array entry



Organization

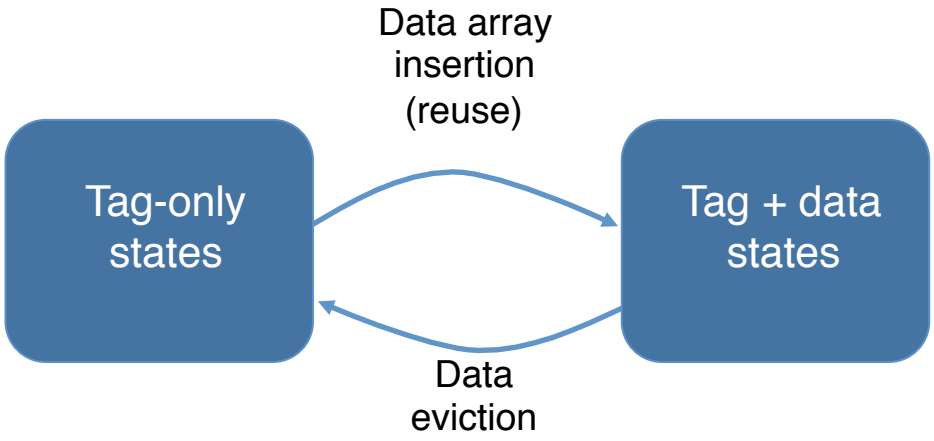
- Tag array
  - Maintains coherence/inclusion
  - Set-associative
  - Each entry has associated data or not
  - Forward pointers: to indicate corresponding data array entry

- Data array
  - Only stores reused lines
  - Queue or set-associative
  - Reverse pointers: update tag array entry



Coherence

- Conventional coherence protocols can be used with small changes
  - Two types of states have to be considered
    - ✓ Tag-only states
    - ✓ Tag + data states
  - Transitions between both types of states are triggered by data insertions or evictions



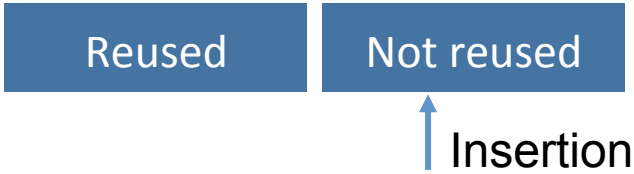
Replacement

- Tag and data arrays have independent replacement policies
  - Tag array
    - ✓ Not Recently-Reused (NRR) [1] [Albericio *et al.* TACO'13]
      - reused and private cache contents are unlikely to be evicted
  - Data array
    - ✓ Clock, for queue organization [Corbató MIT'68]
    - ✓ Not Recently-Used (NRU), for set associativity

NRU vs NRR

- NRU
  - Exploiting temporal locality (approx. to LRU)
  - 1 bit per line
  - Used on intel i7 and SPARC T2

- NRR[1]
  - Exploiting reuse locality
  - 1 bit per line
  - Private cache contents are protected



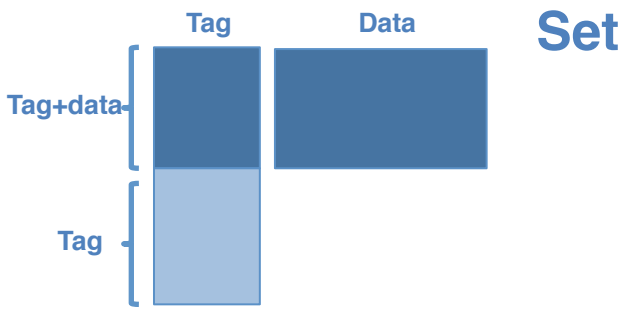
Outline

- Motivation
- The reuse cache
  - Idea
  - Organization
  - Coherence protocol
  - Replacement
- **Related work**
- Evaluation
- Conclusions

Related work

- Many proposals decouple tag/data arrays
  - Decoupled sectored caches, Seznec, ISCA 1994
  - The pool of subsectors, Rothman and A. Smith, ICS 1999
  - NuRAPID, Chishti *et al.*, MICRO 2003
  - V-way, Qureshi *et al.*, ISCA 2005

- Non-inclusive cache, inclusive directory (NCID), Zhao *et al.*, Computer Frontiers 2010
  - Additional tags to maintain inclusion
  - Selective allocation policy based on *set dueling*



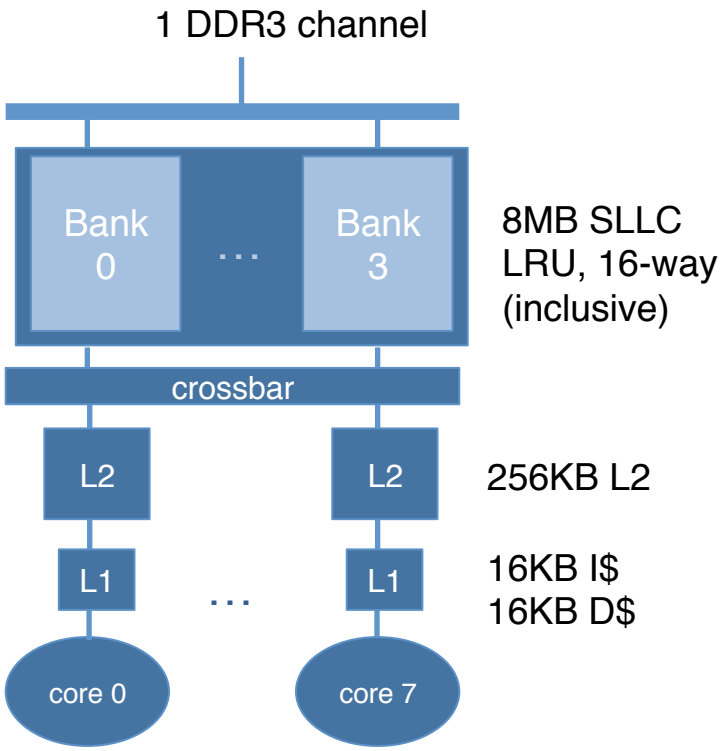
Outline

- Motivation
- The reuse cache
  - Idea
  - Organization
  - Coherence protocol
  - Replacement
- Related work
- **Evaluation**
- Conclusions

Methodology

Baseline system configuration

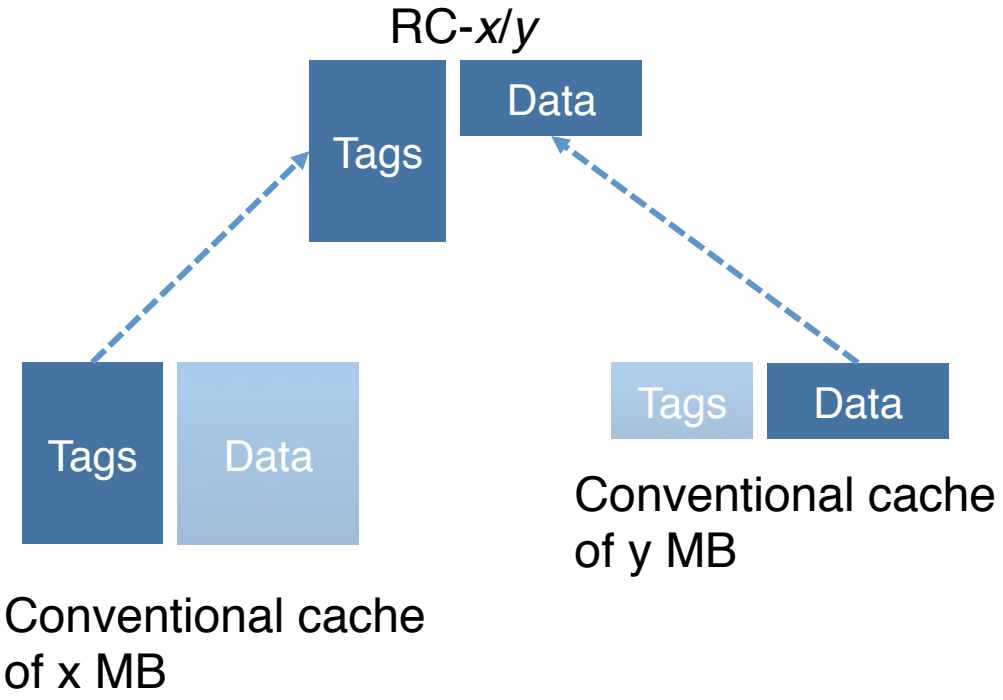
- 8 in-order core CMP system



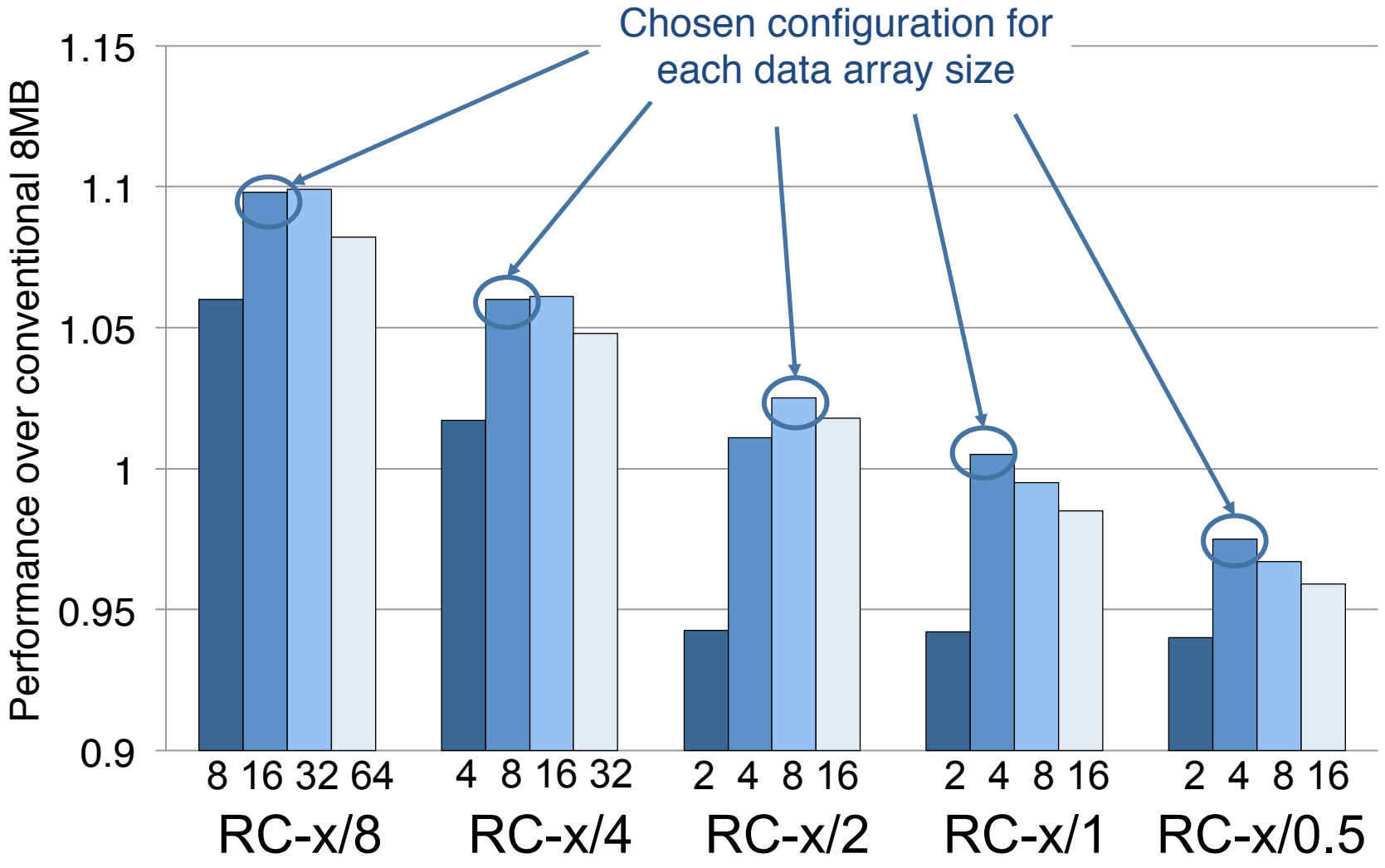
- Simulation
  - Simics full-system simulator
  - Ruby from GEMS Multifacet
- Area and latency
  - CACTI 6.5
- Workloads
  - Multi-programmed: SPEC CPU 2006 (100 random mixes from all the 29 applications)
  - Parallel: from PARSEC and SPLASH2

Terminology

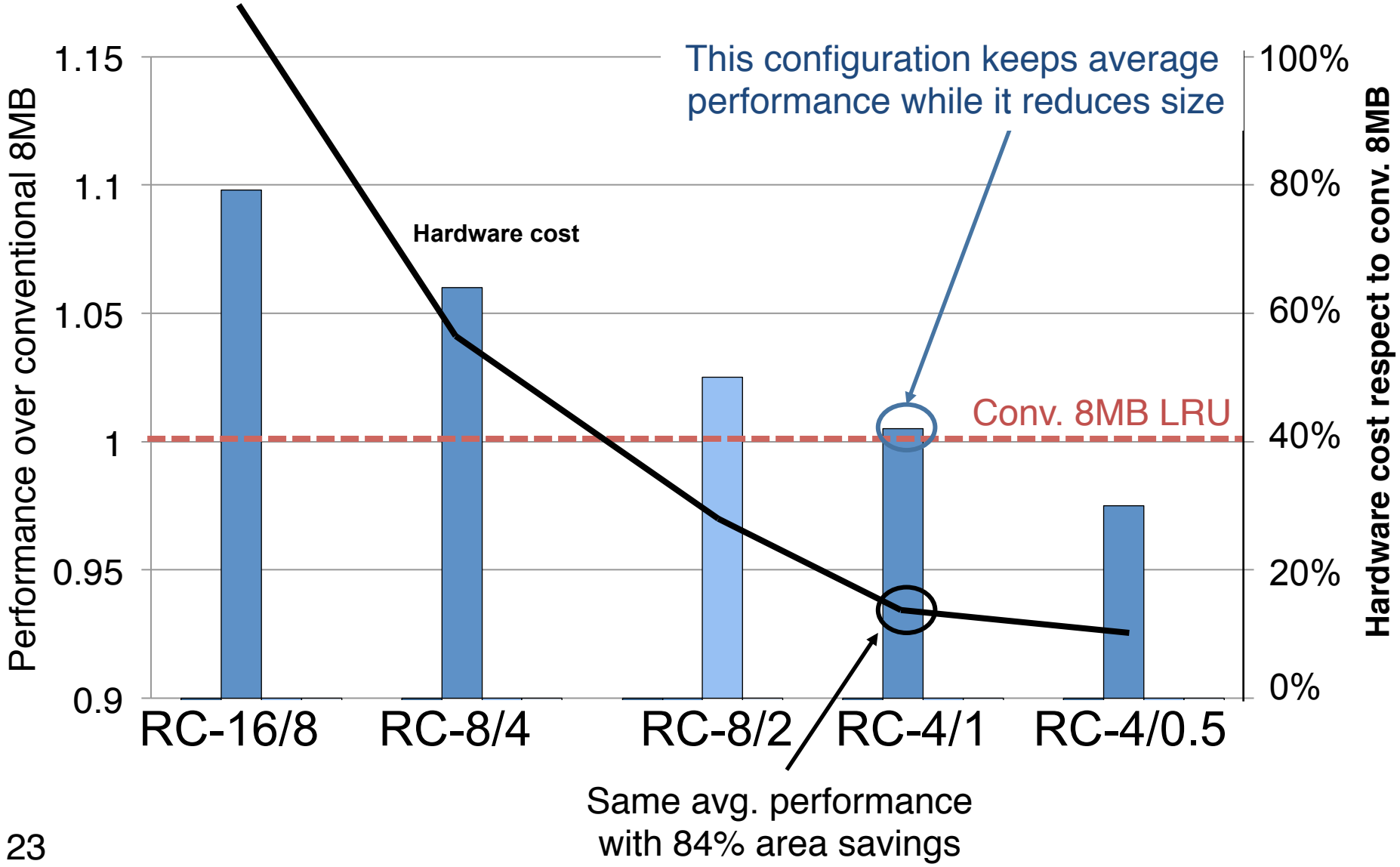
- RC-*x*/*y* = Reuse cache with tags equivalent to *x* MB, *y* MB of data



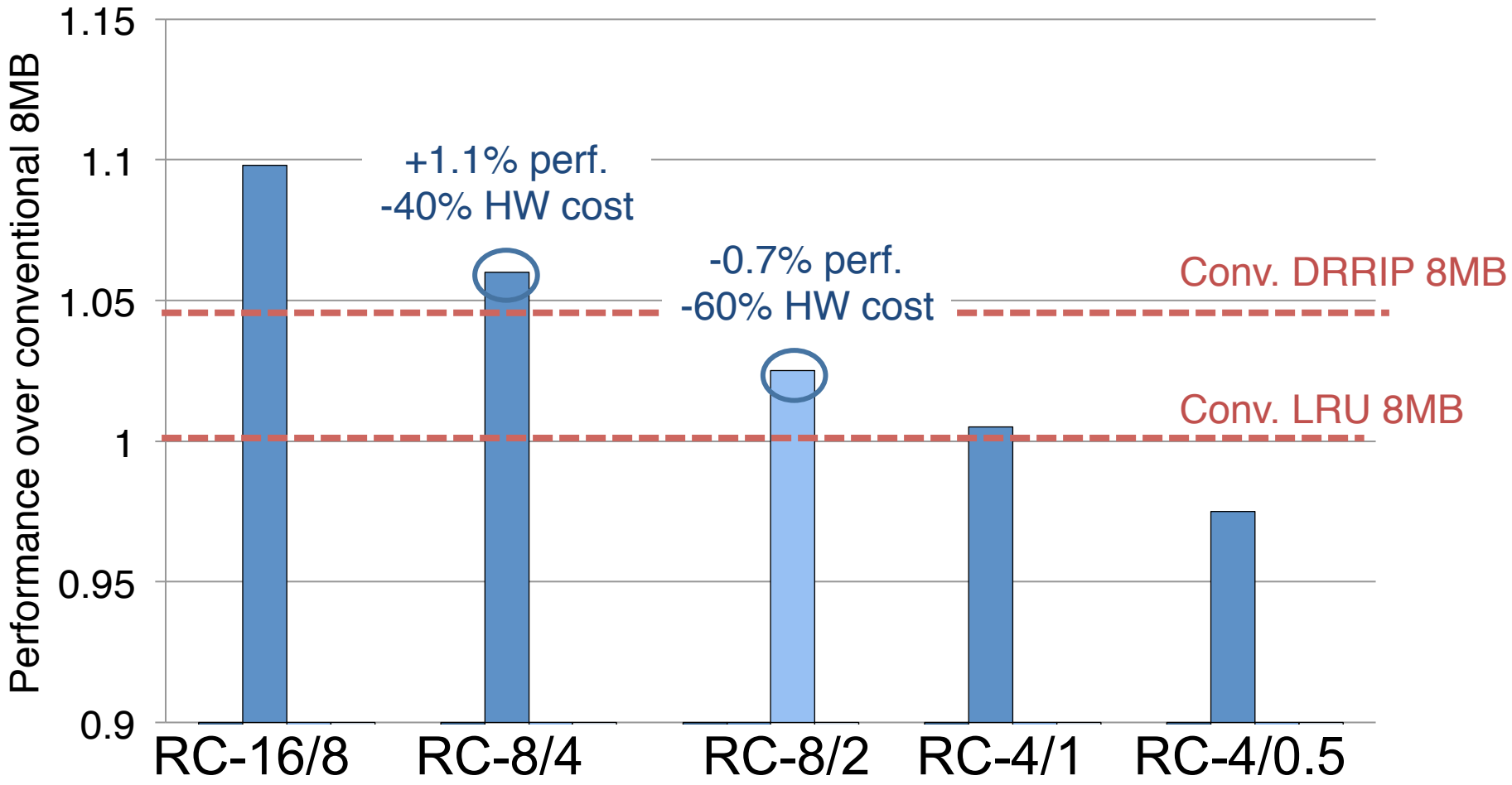
Size exploration



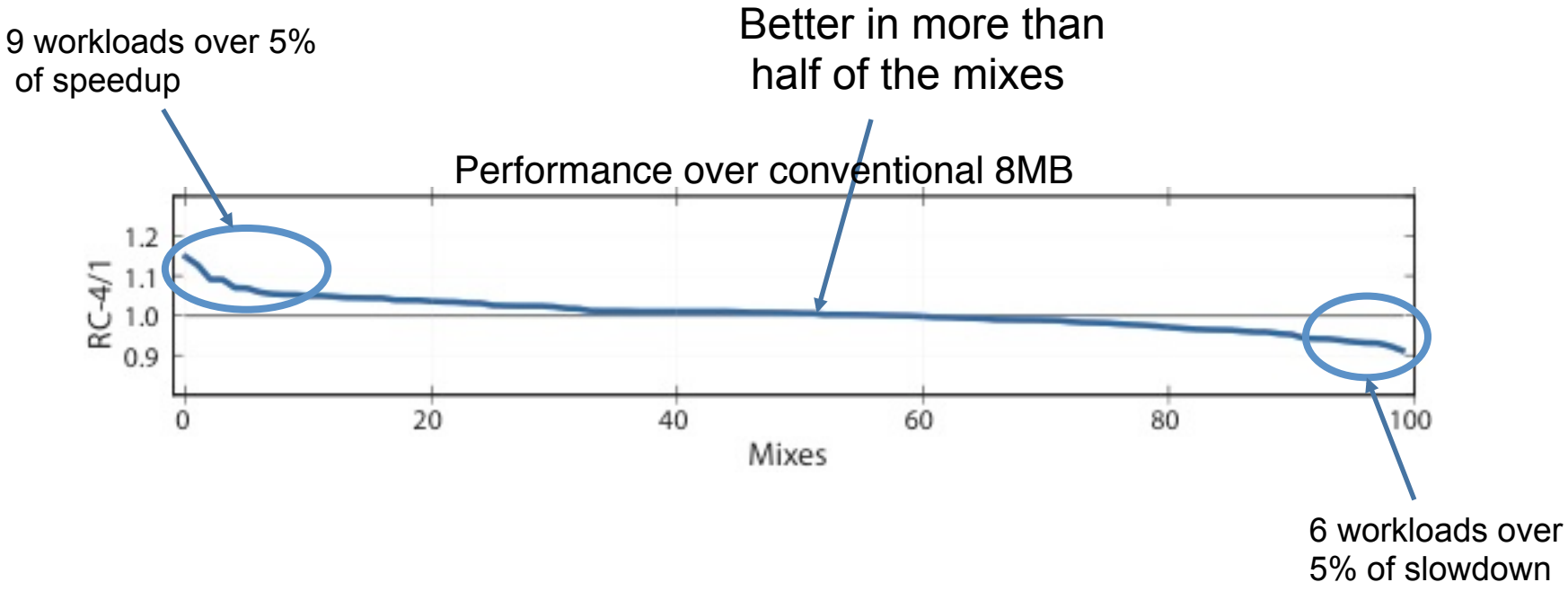
Size exploration



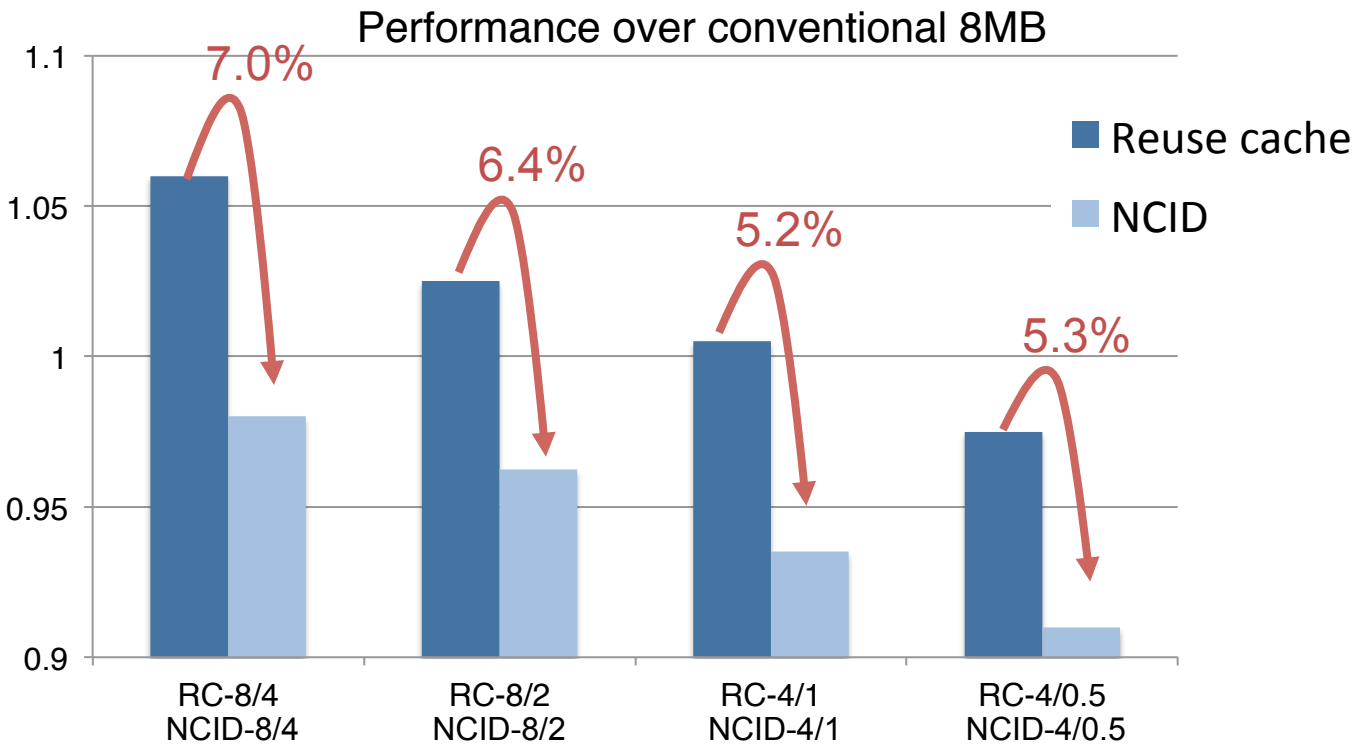
Better baseline



Multiprogrammed



NCID comparison

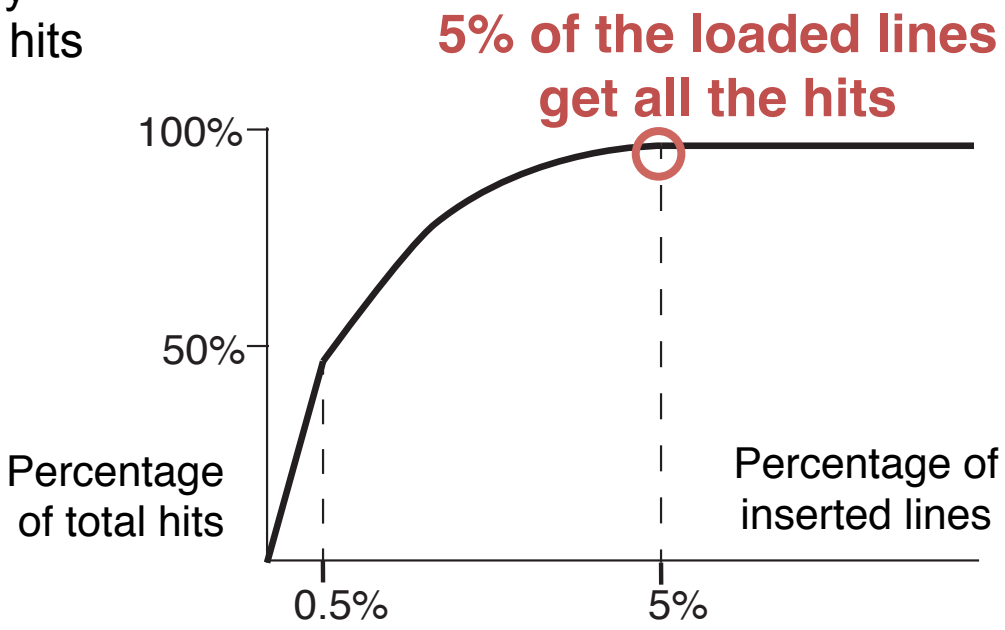


Behavior insight

Percentage of data stored into the cache with respect to the number of tags

- Conventional cache: 100%
- RC-4/1: 5%

Our selection of contents policy is selecting lines that will receive hits



Additional results

- Performance analysis for more sizes
- Set-associative data array
- Per-application study
- Parallel workloads

Outline

- Motivation
- The reuse cache
  - Idea
  - Organization
  - Coherence protocol
  - Replacement
- Related work
- Evaluation
- **Conclusions**

Conclusions

- A big fraction of the SLLC contents is dead
  - A selective allocation policy is needed
- A SLLC organization is proposed: the reuse cache
  - Very selective line allocation policy, based on reuse
  - Same avg. performance with 84% storage savings
  - Alternative applications:
    - ✓ Further reduction of energy
    - ✓ More computing elements with same area

The Reuse Cache

Downsizing the Shared Last-Level Cache

Jorge Albericio[1], Pablo Ibáñez[2], Víctor Viñals[2], and José M. Llabería[3]

1 UNIVERSITY OF TORONTO

2 Universidad Zaragoza 1542

3 UPC