PERFORMANCE AND ENERGY ASSESSMENT OF LAST-LEVEL CACHE REPLACEMENT POLICIES

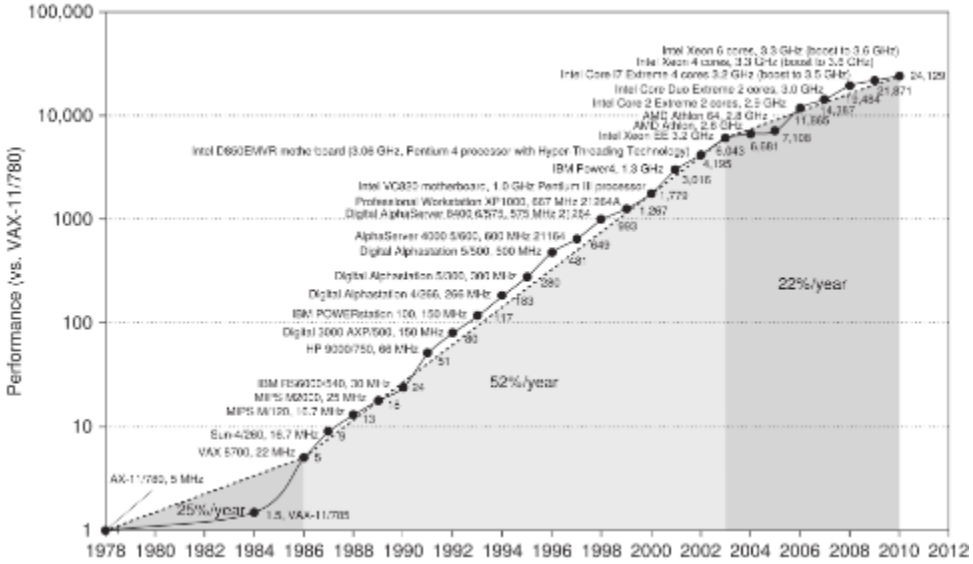
Pierre-Yves Péneau, David Novo, **Florent Bruguier**, Gilles Sassatelli, Abdoulaye Gamatié

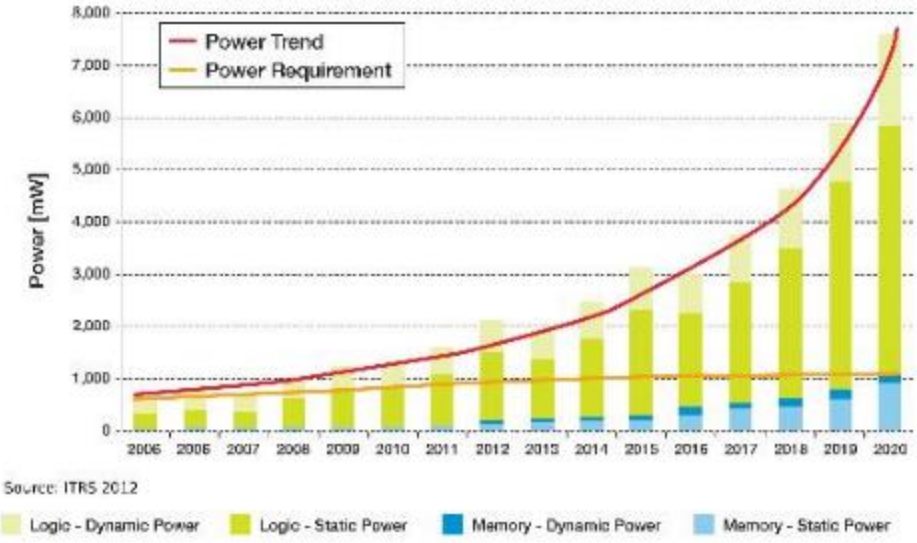
LIRMM, CNRS, University of Montpellier

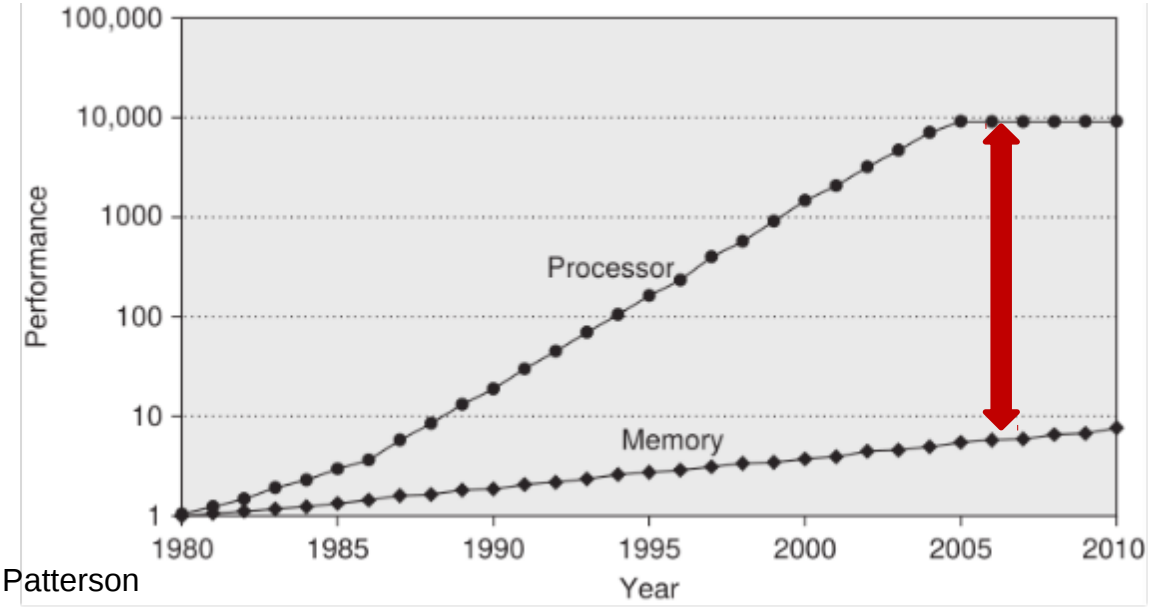
first.last@lirmm.fr

INTRODUCTION

Introduction





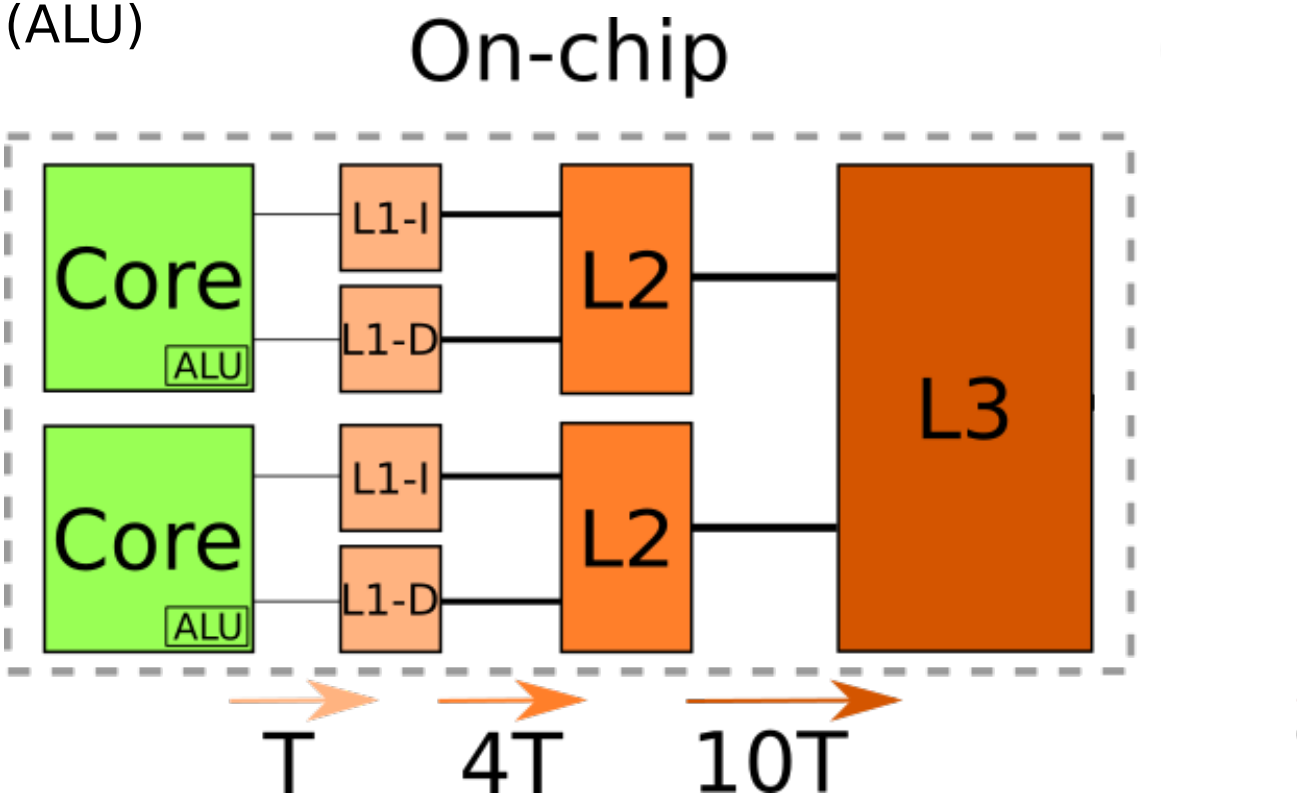


Memory-wall: computation time is no longer the issue

ITRS, Hennessy & Patterson

Memory hierarchy

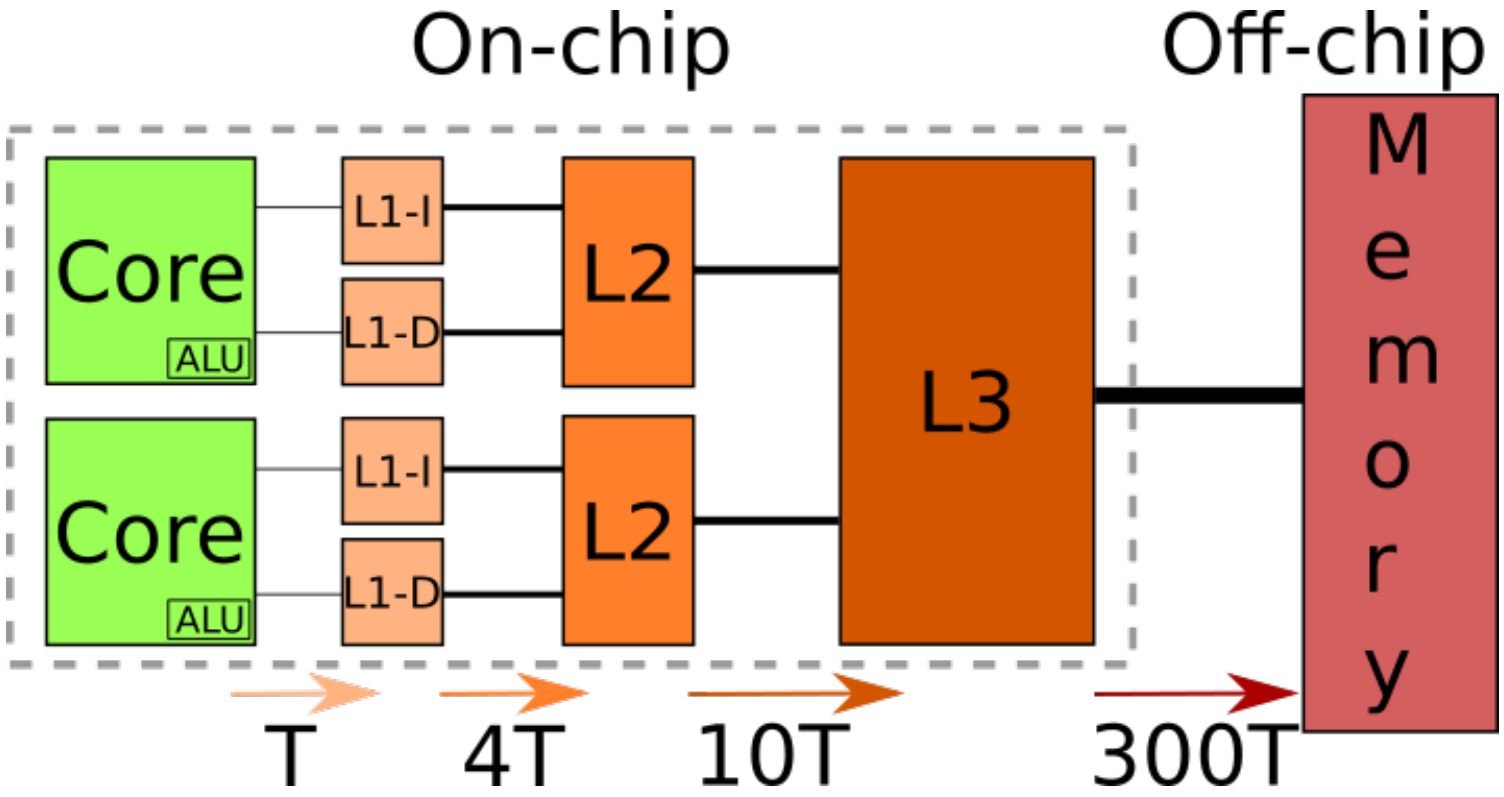
- Memory transactions cost is not linear
- Access to the main memory is a costly operation
  - x300 (time) compare to L1 access
  - x800 (energy) compare to the Arithmetic and Logic Unit (ALU)



Babak Falsafi. *"Server Architecture for the Post-Moore Era"*. HotDC Keynote, 2017

Memory hierarchy

- Objective: limit the number of off-chip request
- Put the effort on the Last-Level Cache (LLC)
  - L3 is this example



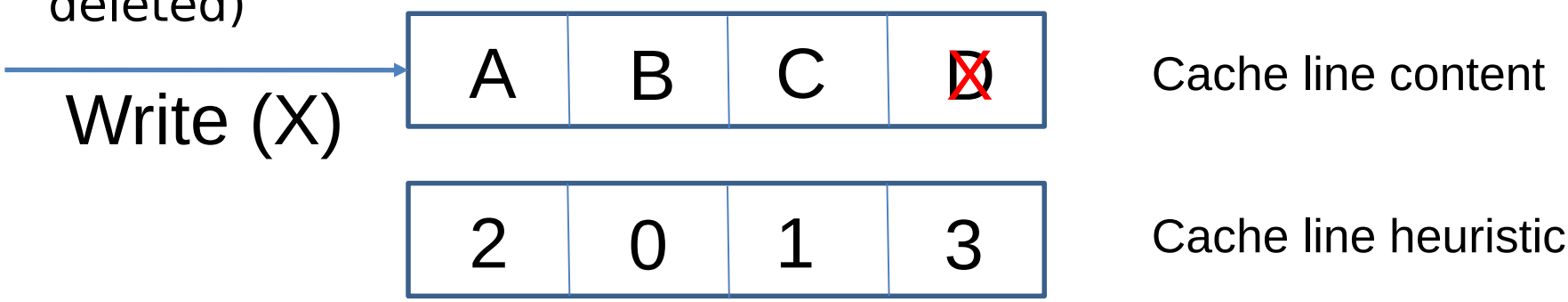
Babak Falsafi. *"Server Architecture for the Post-Moore Era"*. HotDC Keynote, 2017

BACKGROUND

Cache management

- A cache is defined by its policies
  - Replacement
  - Insertion

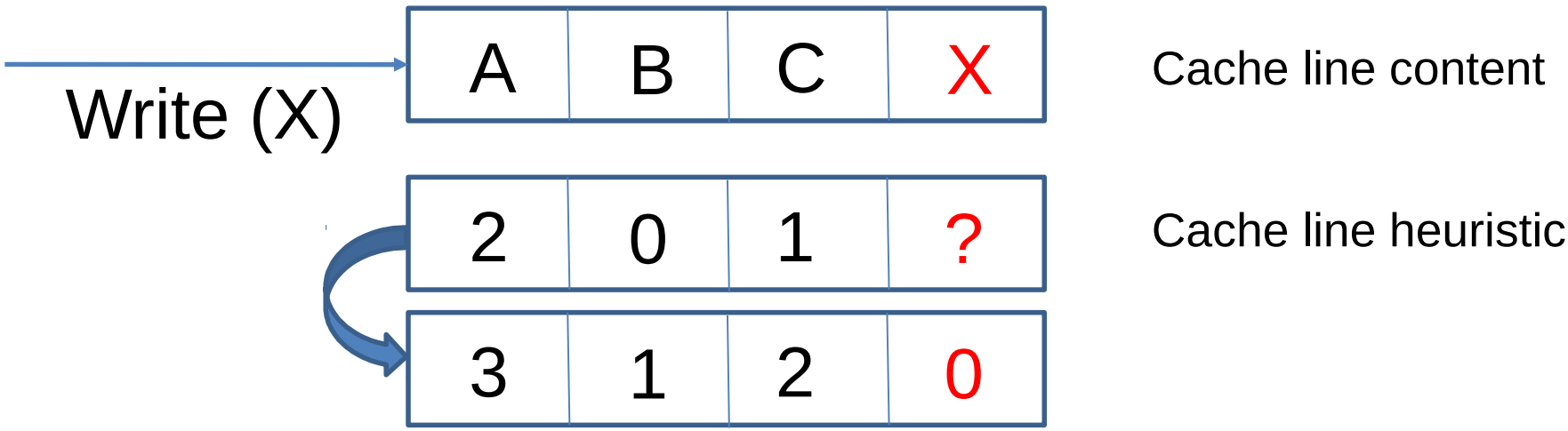
- Replacement: which block has to be deleted when a line is full ?
  - Use an heuristic (ex: the highest block's number is deleted)



D is selected D is the "victim"

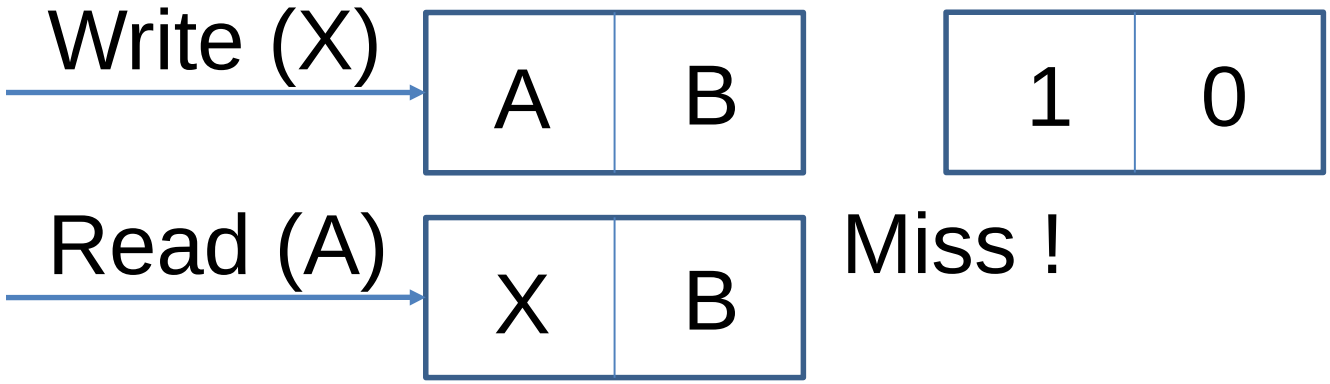
Cache management

- A cache is defined by its policies
  - Replacement
  - Insertion
- Insertion: in which position inserting the new block according to the replacement heuristic ?
  - First (0), Last (3), Intermediate ?



Replacement policy impact

- Directly affects the cache performance
- Responsible of the number of **cache misses**
- A cache miss induces a request to a lower level of the memory
  - The lower level from L3 is the main memory (costly!)

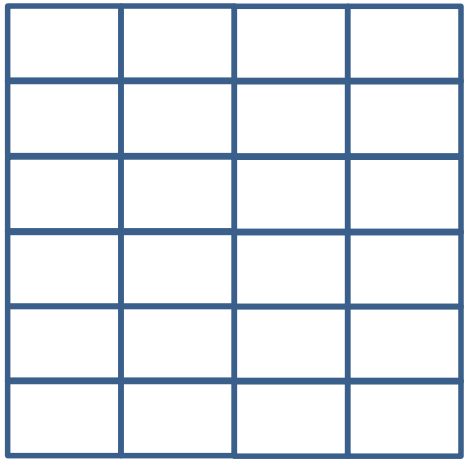


- B should have been evicted to avoid a costly transaction to the main memory

Cache access pattern

- Defines how the cache is accessed by a workload
- Use to guide the heuristic

- 4 types of access
  - Recency pattern (short re-use)
  - Streaming (no re-use)
  - Scan (no locality)
  - Mixed

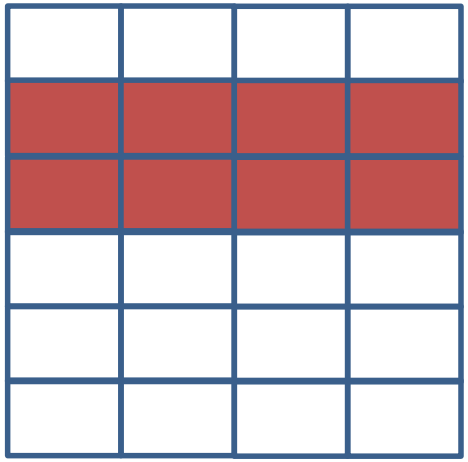


Jaleel et al. *"High performance cache replacement using re-reference interval prediction (RRIP)."* ACM SIGARCH Computer Architecture News. Vol. 38. No. 3. ACM, 2010.`

Cache access pattern

- Defines how the cache is accessed by a workload
- Use to guide the heuristic

- 4 types of access
  - Recency pattern (short re-use)
  - Streaming (no re-use)
  - Scan (no locality)
  - Mixed



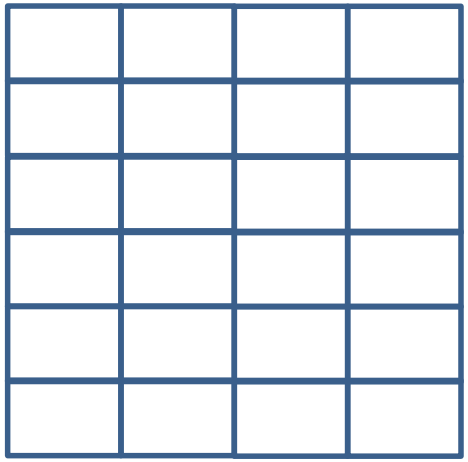
- The heuristic is trained according to block’ s re-use

Jaleel et al. *"High performance cache replacement using re-reference interval prediction (RRIP)."* ACM SIGARCH Computer Architecture News. Vol. 38. No. 3. ACM, 2010.`

Cache access pattern

- Defines how the cache is accessed by a workload
- Use to guide the heuristic

- 4 types of access
  - Recency pattern (short re-use)
  - Streaming (no re-use)
  - Scan (no locality)
  - Mixed



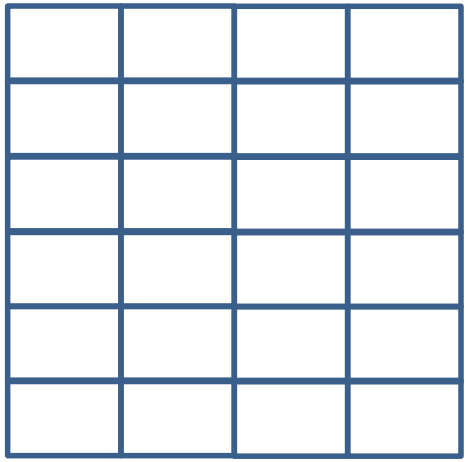
- No block re-use, no training

Jaleel et al. *"High performance cache replacement using re-reference interval prediction (RRIP)."* ACM SIGARCH Computer Architecture News. Vol. 38. No. 3. ACM, 2010.`

Cache access pattern

- Defines how the cache is accessed by a workload
- Use to guide the heuristic

- 4 types of access
  - Recency pattern (short re-use)
  - Streaming (no re-use)
  - Scan (no locality)
  - Mixed



- No locality, training is hard

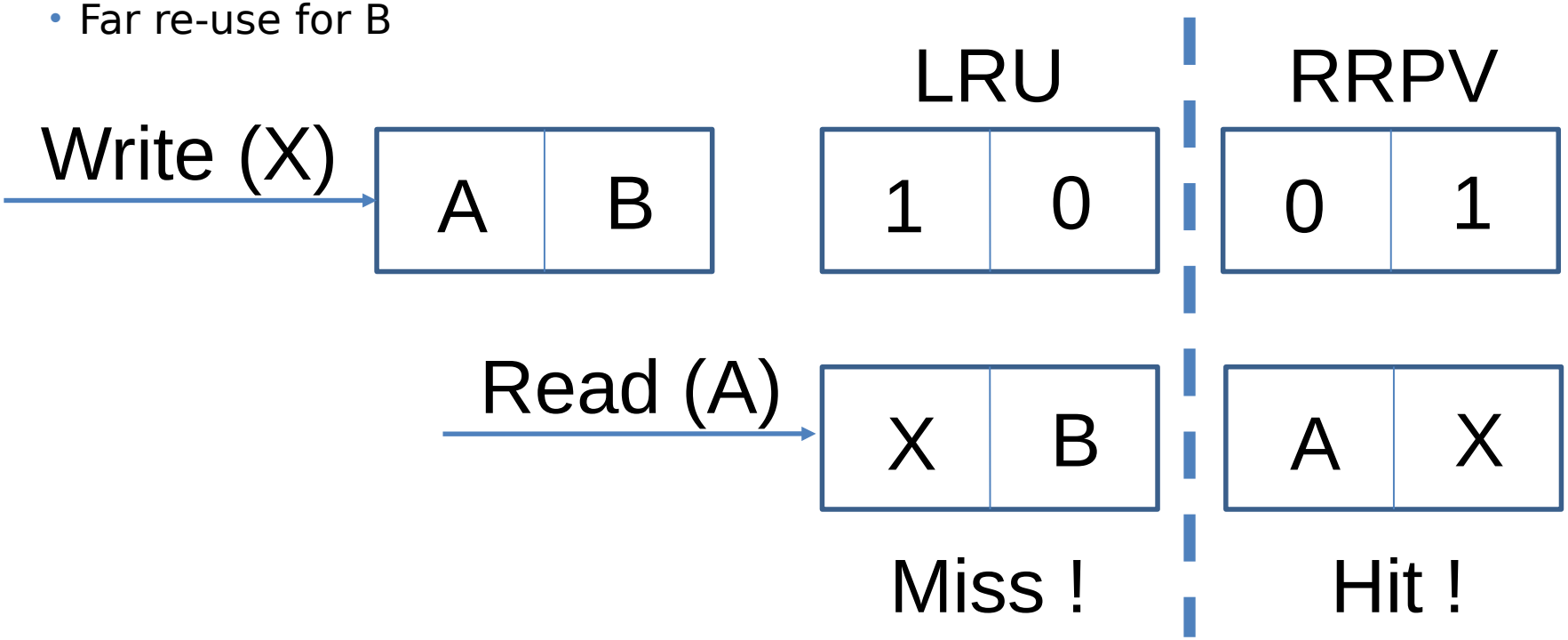
Jaleel et al. *"High performance cache replacement using re-reference interval prediction (RRIP)."* ACM SIGARCH Computer Architecture News. Vol. 38. No. 3. ACM, 2010.`

Literature overview

- Default replacement policy: Least-Recently Used (LRU)
  - Most common policy in our machines today
  - Works only with recency pattern
- Recent proposal: SRRIP, DRRIP, SHiP, Hawkeye
- New heuristic that replace LRU:
  - Re-Reference Prediction interVal (RRPV)
- Mimics the notion of “distance” between two uses
  - A recently used block does not necessary means a near-future re-use

Re-Reference Prediction interVal (RRPV)

- Mimics the notion of “distance” between two uses
  - A recently used block does not necessary means a near-future re-use
- Previous cache accesses: A-A-B-B
  - B has just been used, LRU position is 0
- Future cache accesses: A-A-X-X-A-A-B-B
  - Far re-use for B



EXPERIMENTAL RESULTS

Questions

- What is the performance impact of such policies ?
- What is the energy impact ?
  - Faster execution means more energy consumption ?
  - Vice-versa
- What is the replacement policy behavior when prefetching is activated ?
  - Both mechanism are in competition to reach the same objective: decrease the number of misses

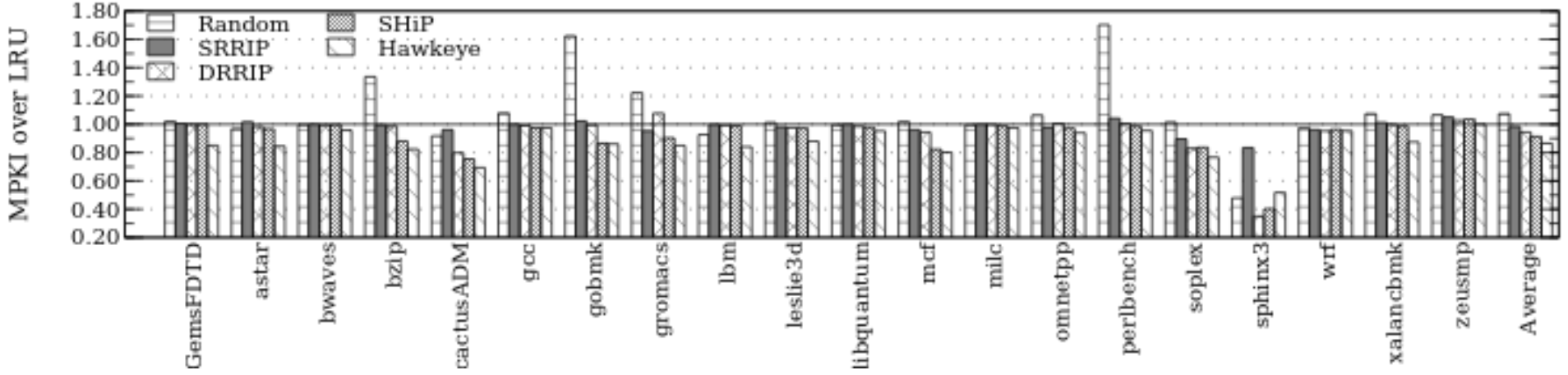
Experimental setup

- 20 applications from SPEC CPU2006 benchmark suite
- ChampSim simulator used at the ISCA's Cache Replacement Championship (CRC)
- Energy model based on CACTI
- Formula used for computation:
  - $E = [(N_{read} + N_{write}) * E_{access}] + [Time * P_{leak}]$
- Reference: 2MB 16-way LLC with LRU

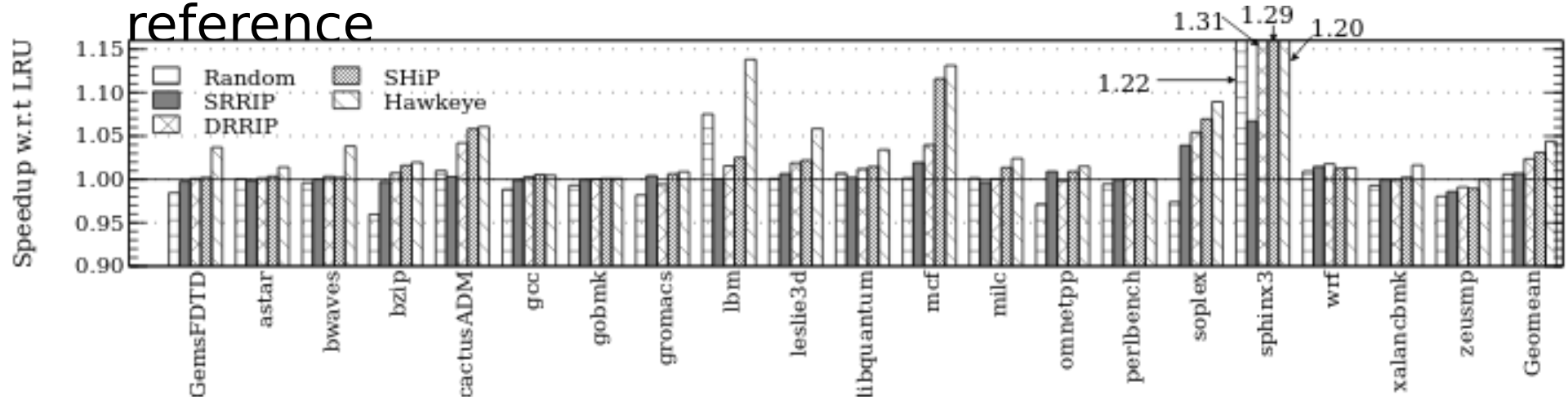
The 2nd Cache Replacement Championship (CRC2, 2017) - http://crc2.ece.tamu.edu
Muralimanohar et al. *"CACTI 6.0: A tool to model large caches."* HP Laboratories (2009)

Performance without prefetching

- Normalized Miss Per Kilo Instructions (IPC) over reference

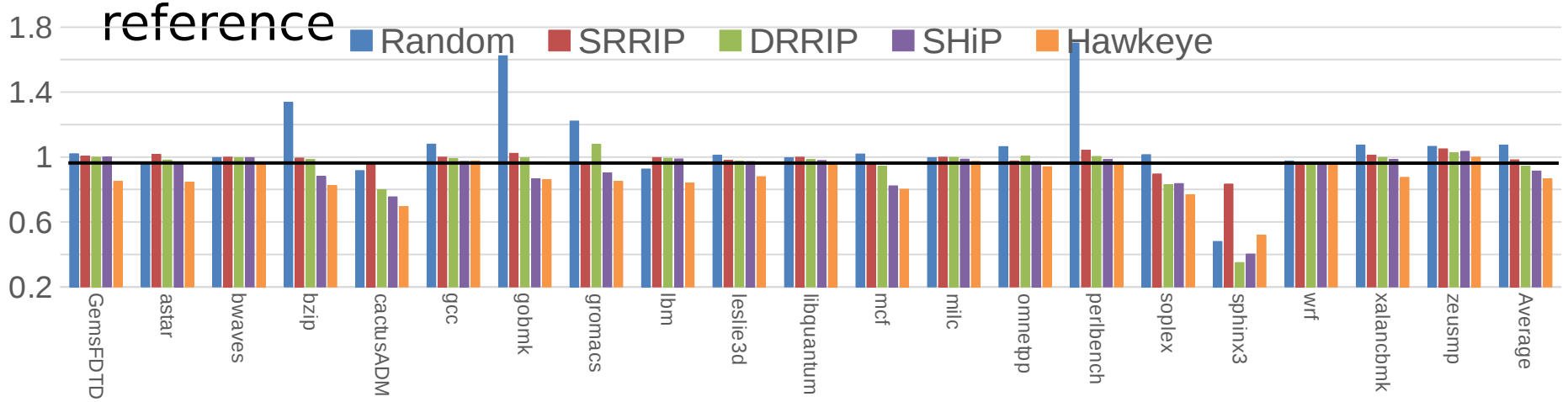


- Normalized Instruction Per Cycle (IPC) over reference

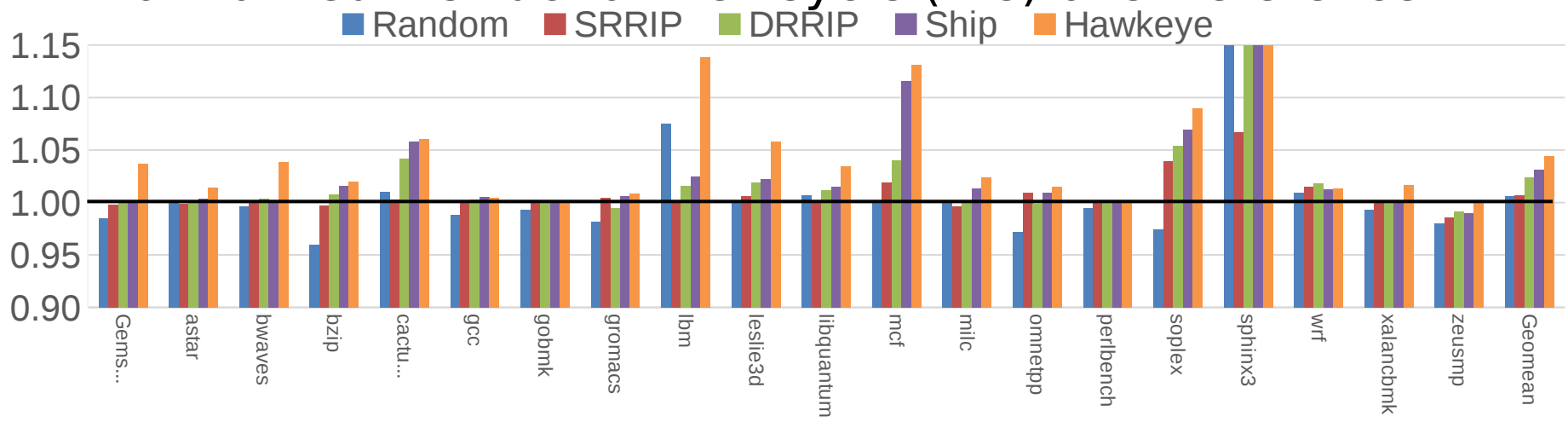


Performance without prefetching

- Normalized Miss Per Kilo Instructions (IPC) over reference



- Normalized Instruction Per Cycle (IPC) over reference



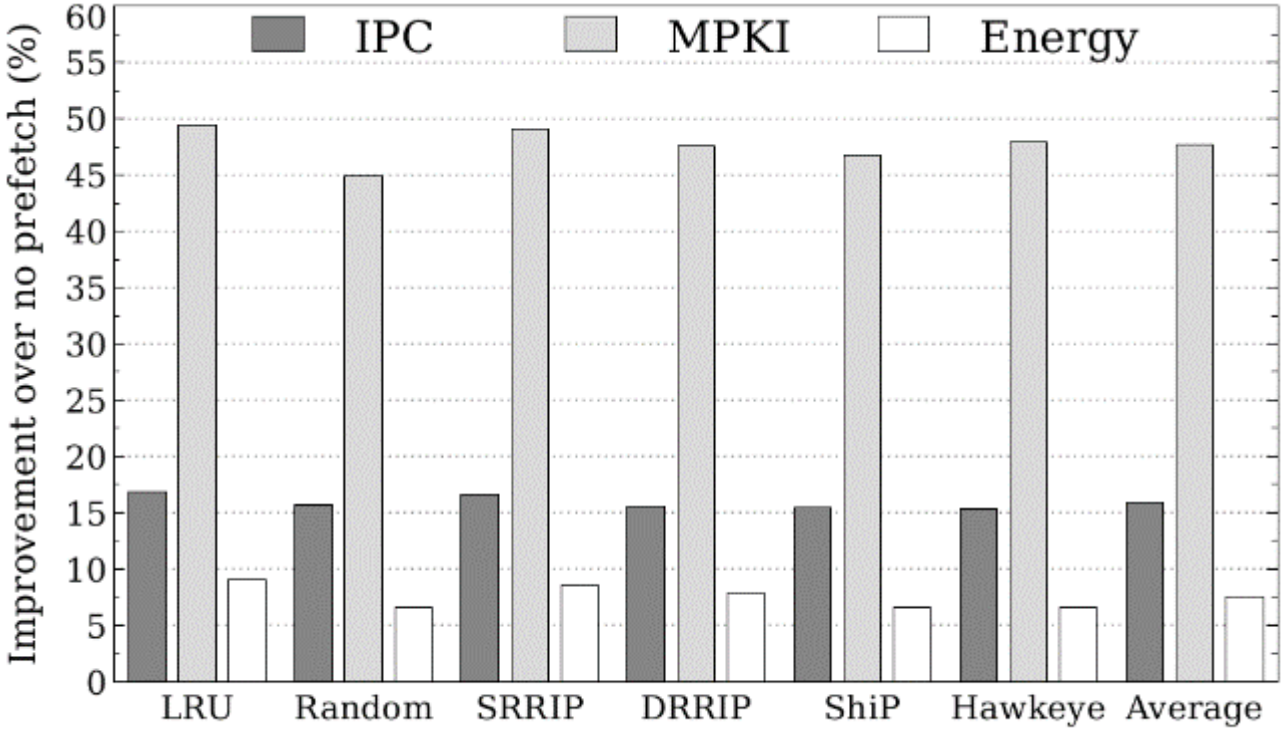
Prefetching effect

- Advanced replacement policies (SHiP, Hawkeye) base their predictions on Program Counter (PC)
  - Each instruction that access a block has a PC
- Prefetched blocks are automatically accessed
  - No related instructions no PC

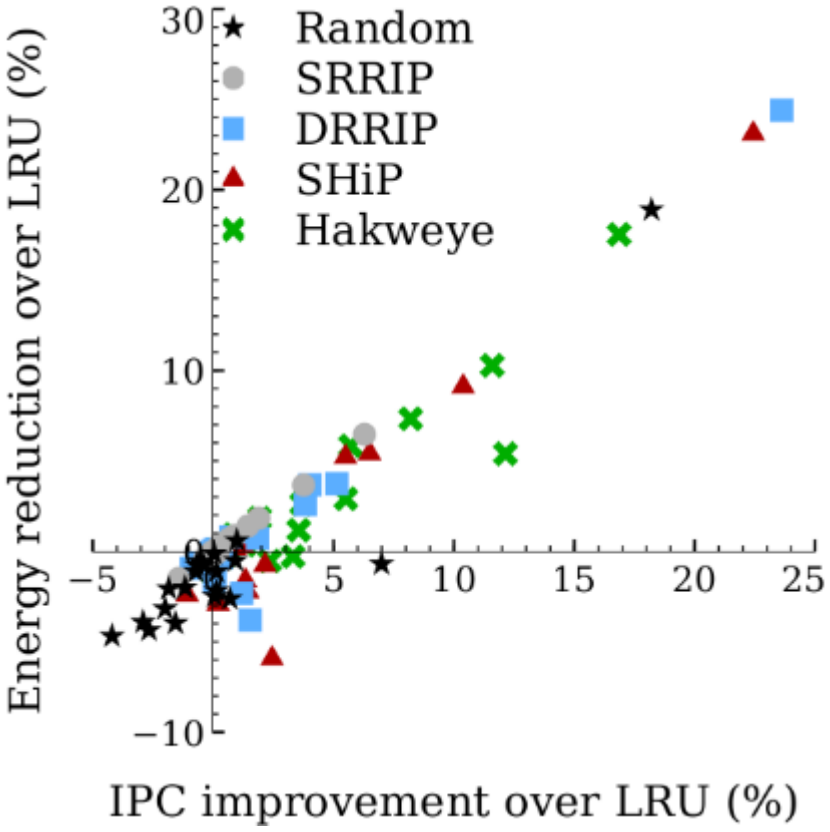
- Impossible to make prediction based on prefetched blocks. Two solutions:
  - Ignore these blocks (SRRIP, DRRIP, SHiP)
  - Implement a dedicated mechanism (Hawkeye)

Prefetching effect

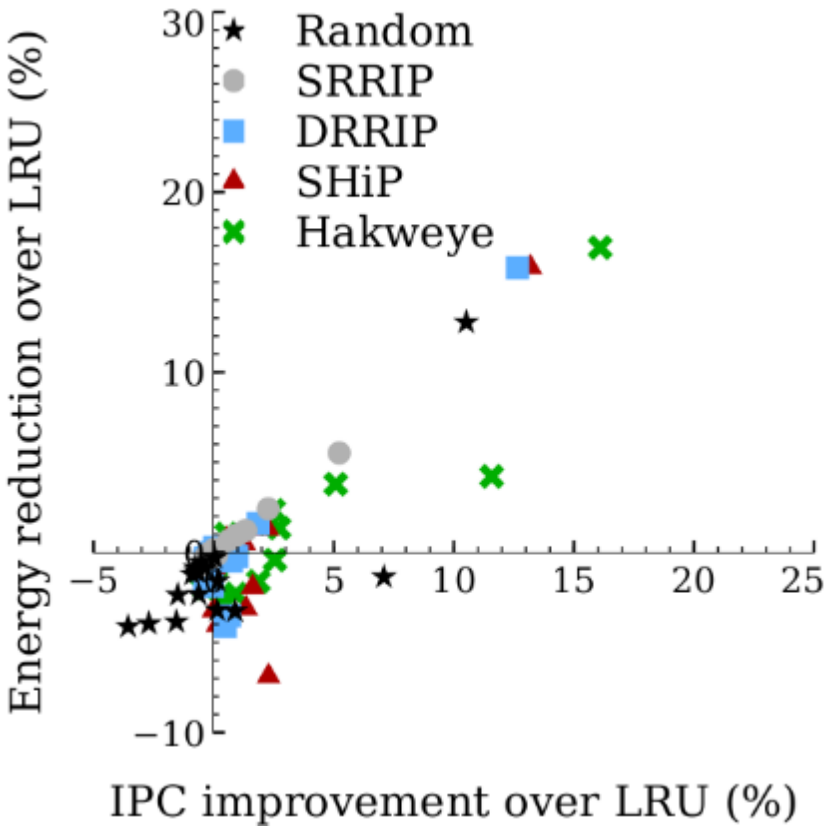
- Positive effect on MPKI, IPC and energy globally
- The Hawkeye policy does not benefit as much as the other in terms of IPC despite advanced mechanism to manage concurrent decisions



Energy and performance impact over LRU



- No prefetching
  - Linear progression



- With prefetching
  - No linearity
  - Lower impact for both IPC and energy

CONCLUSION

Conclusion and perspectives

- All policies outperform LRU on average by 2.2%
  - Up to 4.2% for Hawkeye
- Gains in execution time allow an energy improvement by 1.45%
  - Up to 4% for Hawkeye
- Prefetching impact: results show that performance improvement with prefetching is less important than without

Conclusion and perspectives

- LRU is not the optimal policy
  - Works well on average
  - Cheap to implement
- Advanced replacement policies require additional hardware and complexity in terms of design
  - No real implementation for now
- Prefetching is in competition with replacement policy
  - Wrong eviction decision could exist due to a lack of information from the prefetcher
  - Future work should focus on this particular aspect