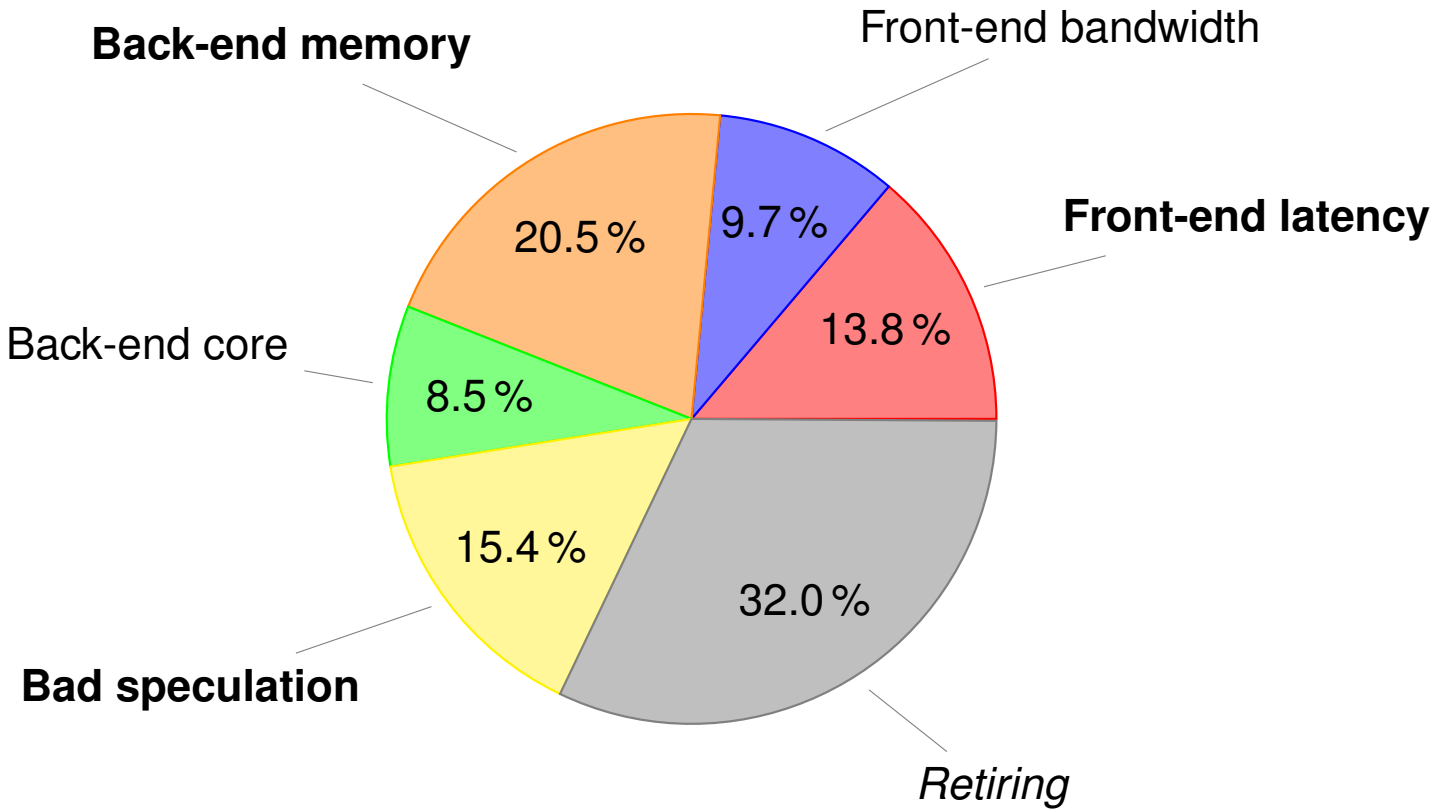
Accurate Predictions for High-Performance Processors

Alberto Ros

University of Murcia, Spain

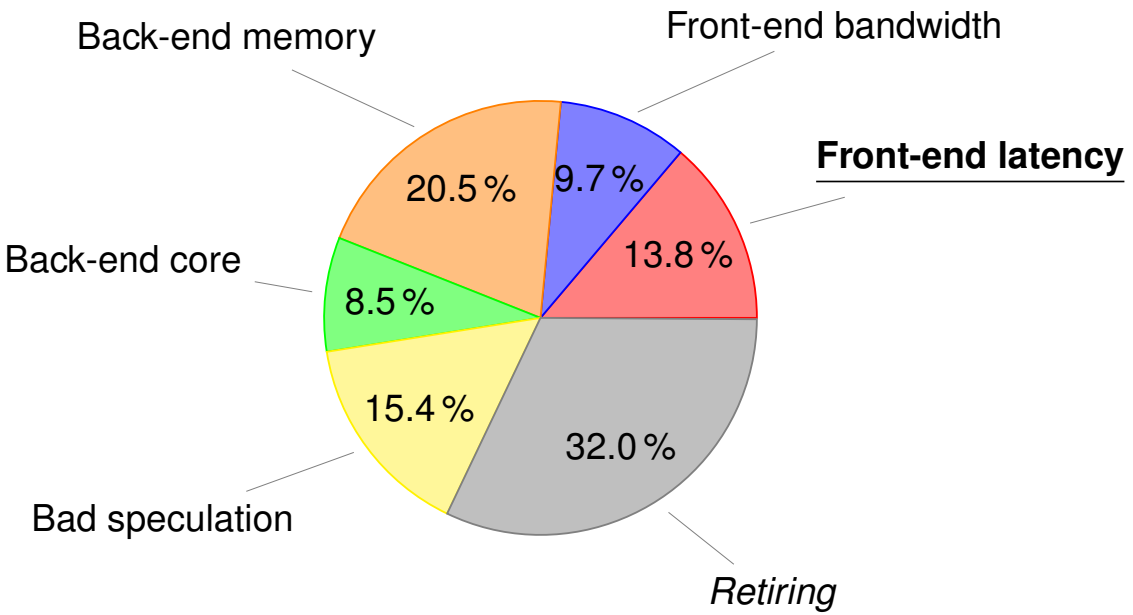
Oct 2, 2024

FACTS: SINGLE-THREAD PERFORMANCE MATTERS



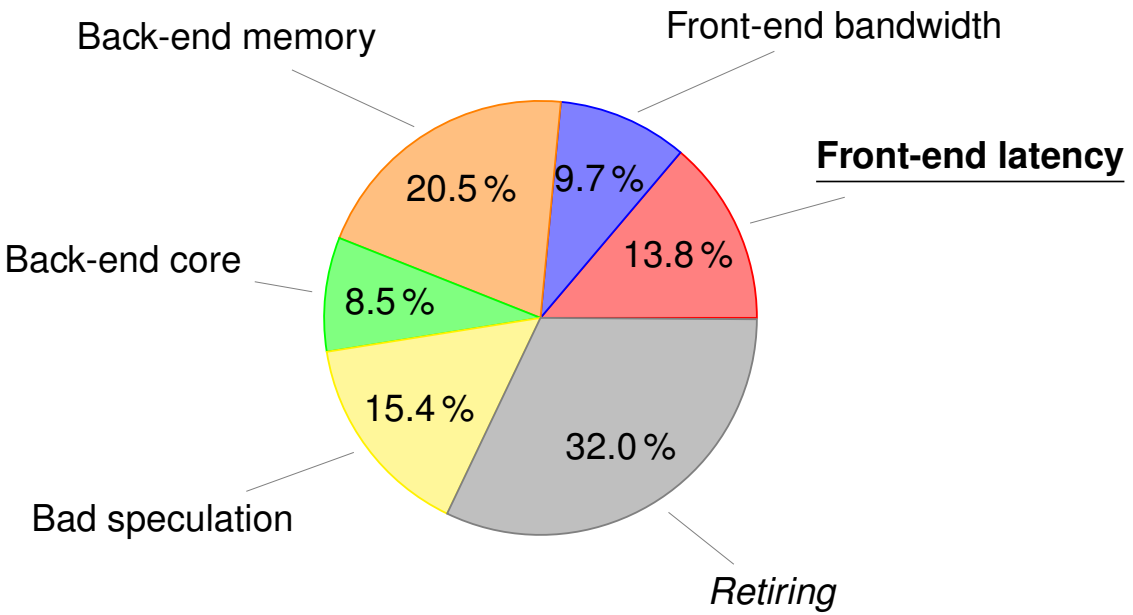
Data source: Ayers et al. *AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale Computers*, ISCA 2019.

Overview: Prediction is all you need



- Dominated by instruction cache (L1I) misses
  - Server and cloud apps getting larger, far from fitting in L1I
  - Hiting in the L2 or L3
- Critical! Processors need to keep the pipeline full

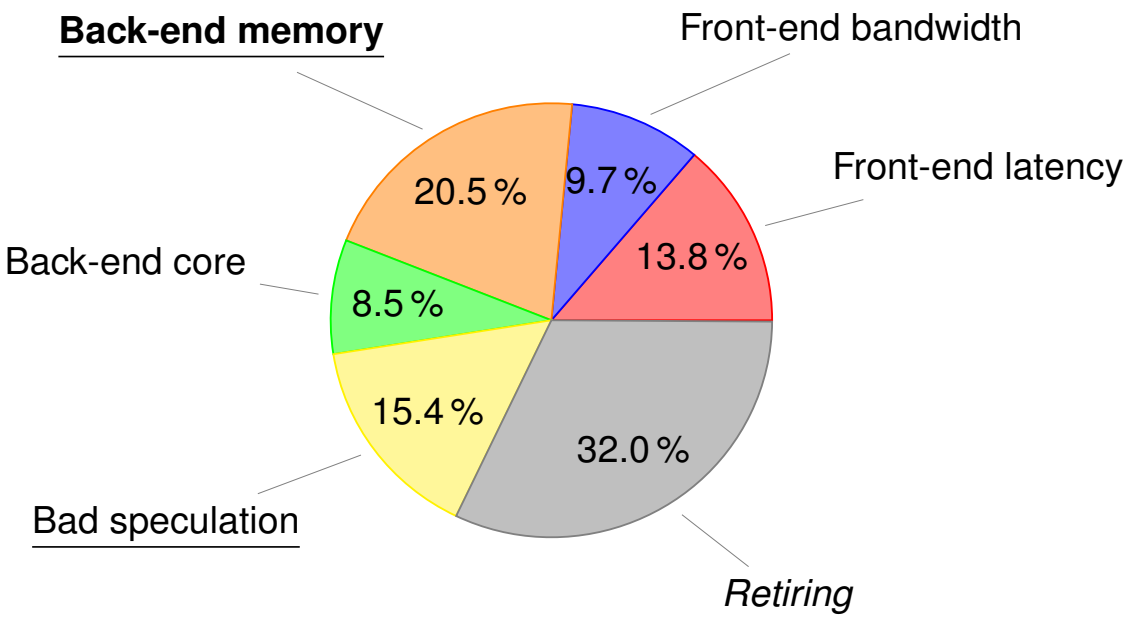
Overview: Prediction is all you need



- Dominated by instruction cache (L1I) misses
  - Server and cloud apps getting larger, far from fitting in L1I
  - Hiting in the L2 or L3
- Critical! Processors need to keep the pipeline full

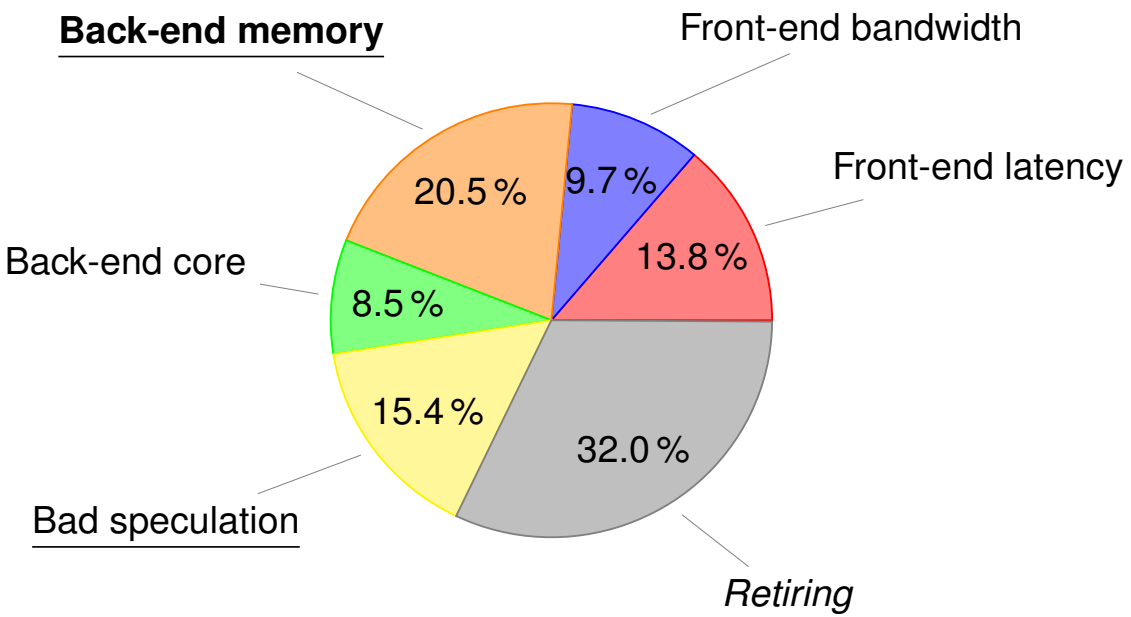
Contribution 1: Entangling instruction prefetcher [ISCA'21]

Overview: Prediction is all you need



- Due to data cache (L1D) misses
  - Many of them reaching main memory
- Cause stalls and late detection of bad speculation

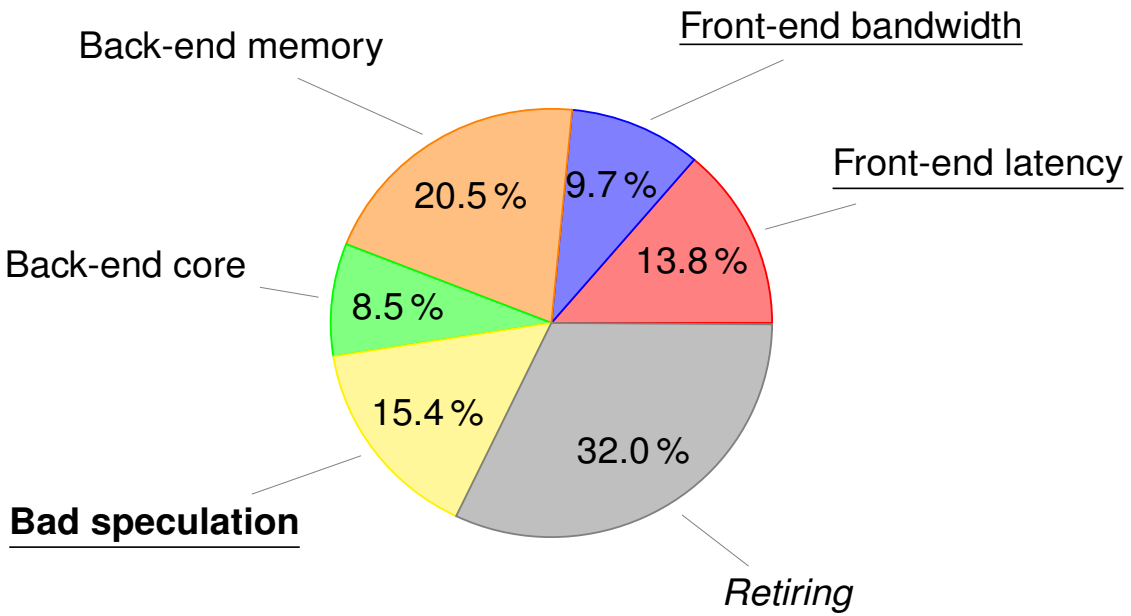
Overview: Prediction is all you need



- Due to data cache (L1D) misses
  - Many of them reaching main memory
- Cause stalls and late detection of bad speculation

Contribution 2: BERTI data prefetcher [MICRO'22]

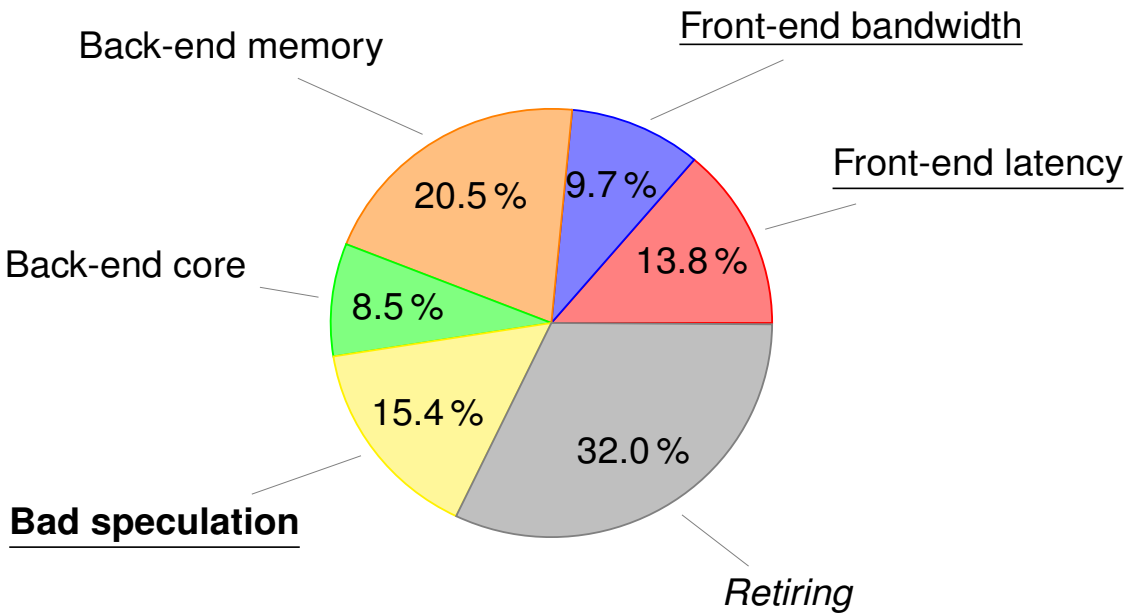
Overview: Prediction is all you need



- Memory dependence prediction

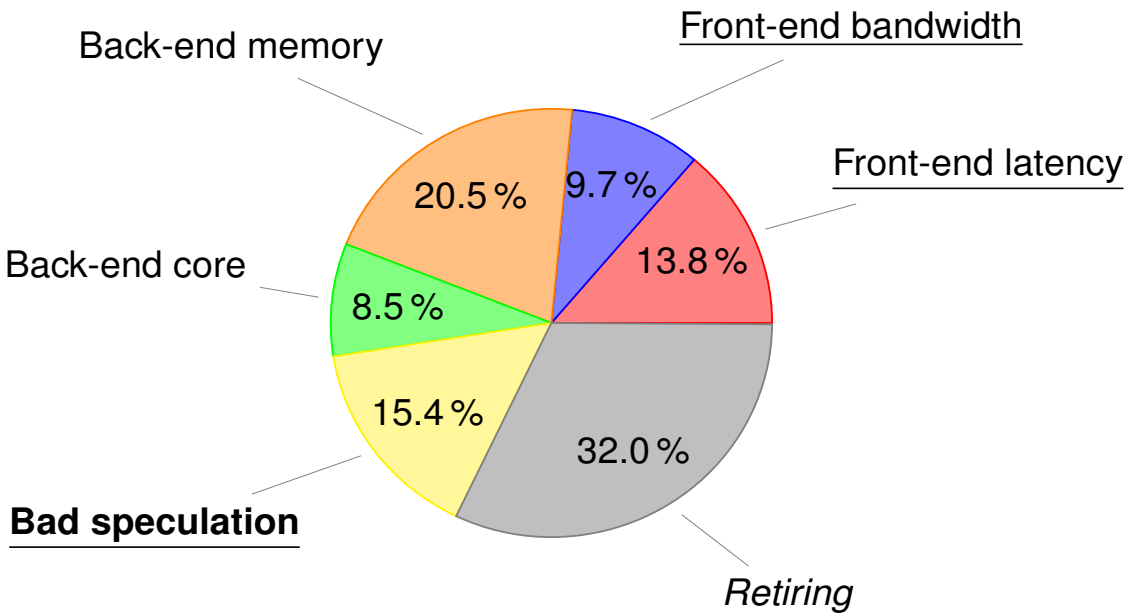
- Branch prediction

Overview: Prediction is all you need



- Memory dependence prediction
  - More critical as the pipeline depth increases
- Branch prediction

Overview: Prediction is all you need

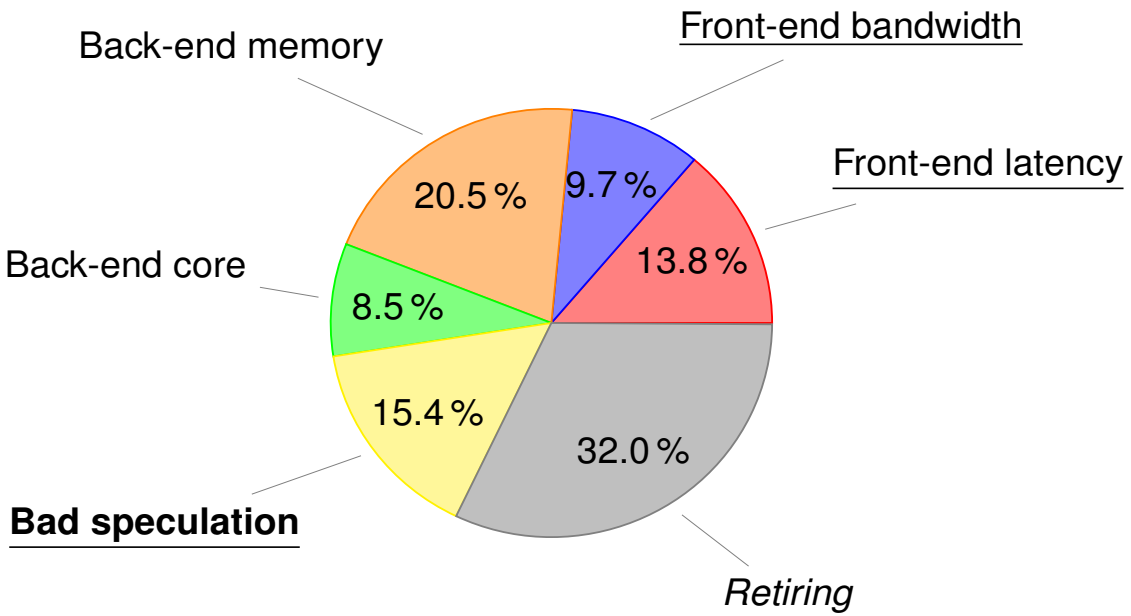


- Memory dependence prediction
  - More critical as the pipeline depth increases

Contribution 3: PHAST mem. dep. predictor [HPCA'24]

- Branch prediction

Overview: Prediction is all you need

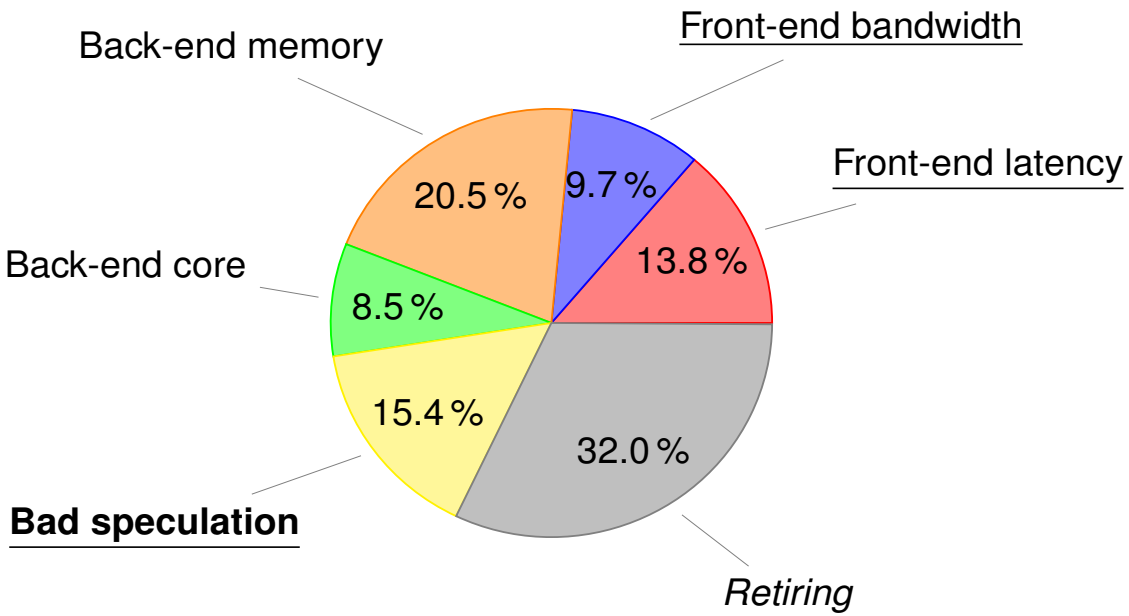


- Memory dependence prediction
  - More critical as the pipeline depth increases

Contribution 3: PHAST mem. dep. predictor [HPCA'24]

- Branch prediction
  - Hard to optimize!

Overview: Prediction is all you need



- Memory dependence prediction
  - More critical as the pipeline depth increases

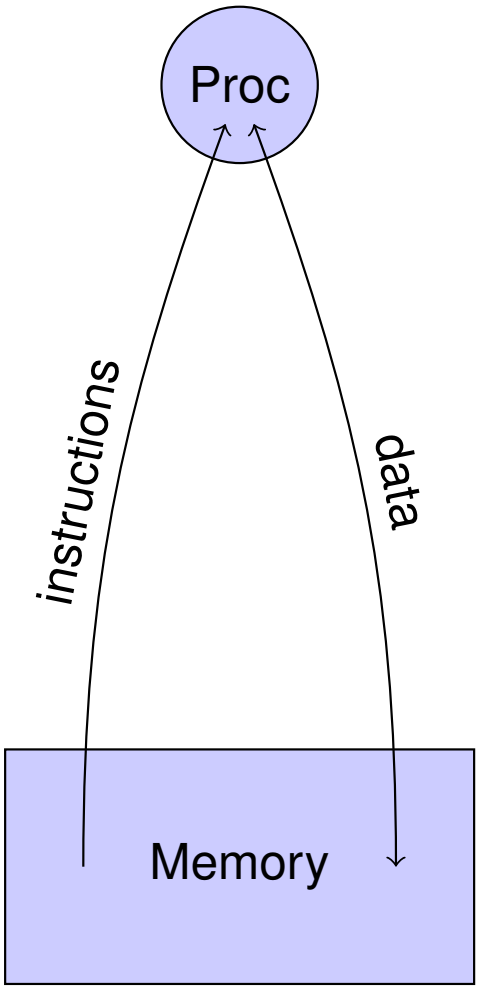
Contribution 3: PHAST mem. dep. predictor [HPCA'24]

- Branch prediction
  - Hard to optimize! $\Rightarrow$ Recover faster from mispredictions

Contribution 4: ALT-PATH UOP-CACHE prefetching [ISCA'24]

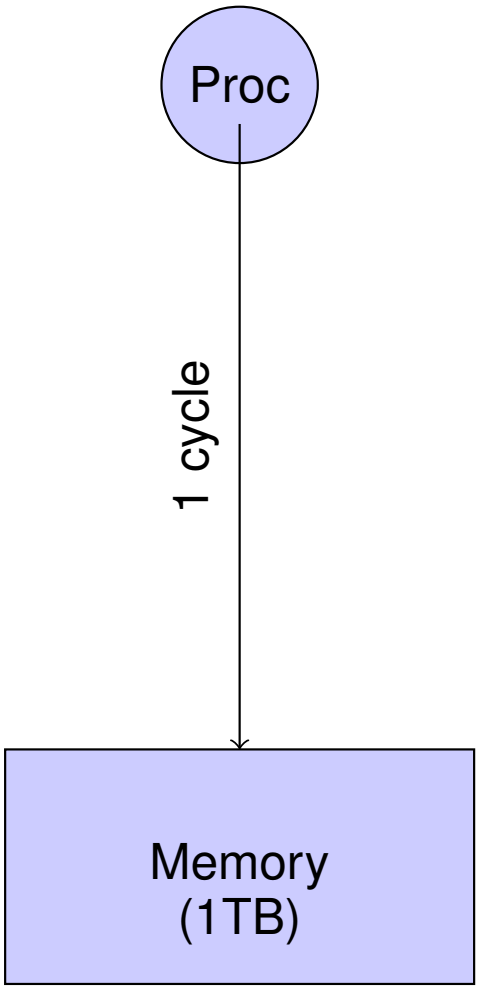
PREFETCHING

- Processors need to access instructions and data from memory



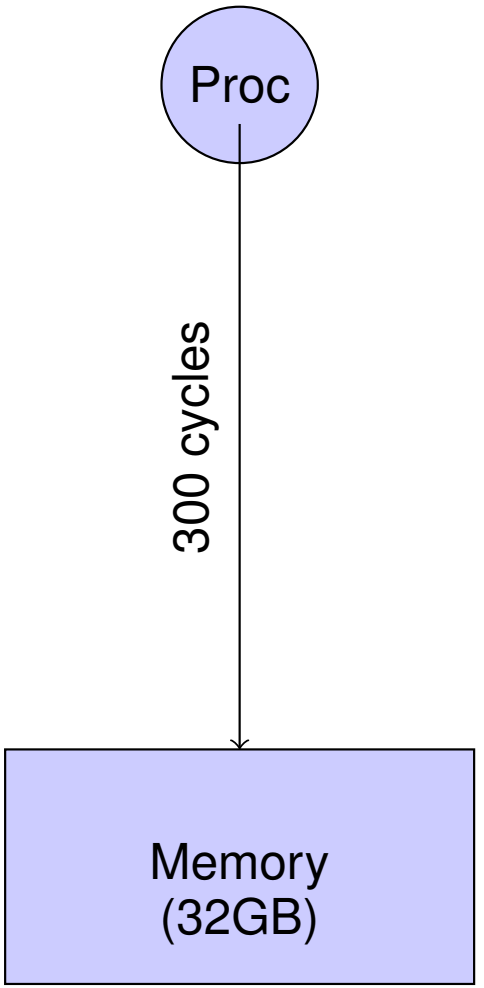
PREFETCHING

- Processors need to access instructions and data from memory
- Ideally processors would need a very large memory with a low access latency



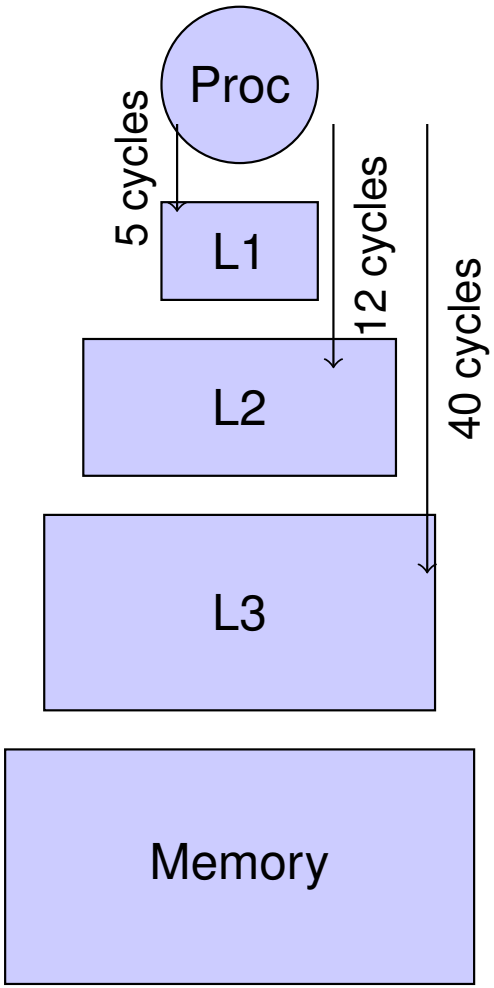
PREFETCHING

- Processors need to access instructions and data from memory
- Ideally processors would need a very large memory with a low access latency
  - → Not possible due to technology limitations



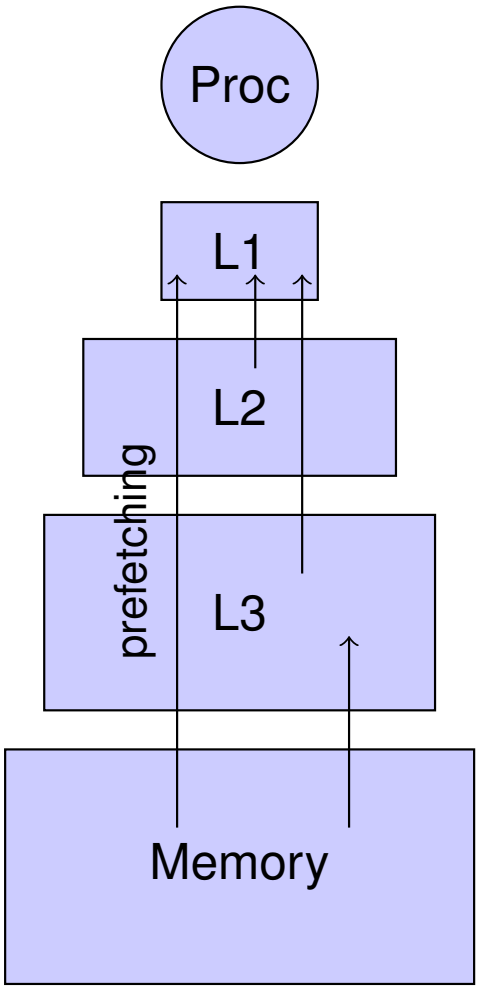
PREFETCHING

- Processors need to access instructions and data from memory
- Ideally processors would need a very large memory with a low access latency
  - → Not possible due to technology limitations
- Multi-level cache hierarchies approach this goal thanks to instr./data locality
  - → Still many long-latency accesses

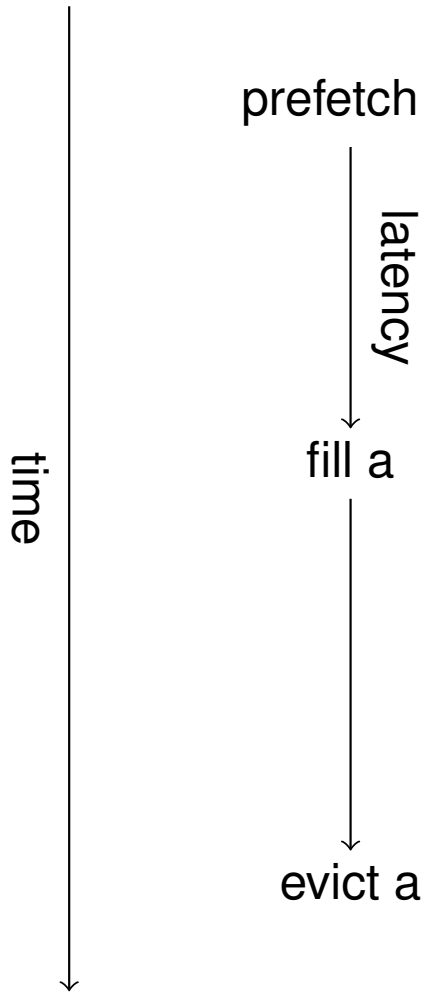


PREFETCHING

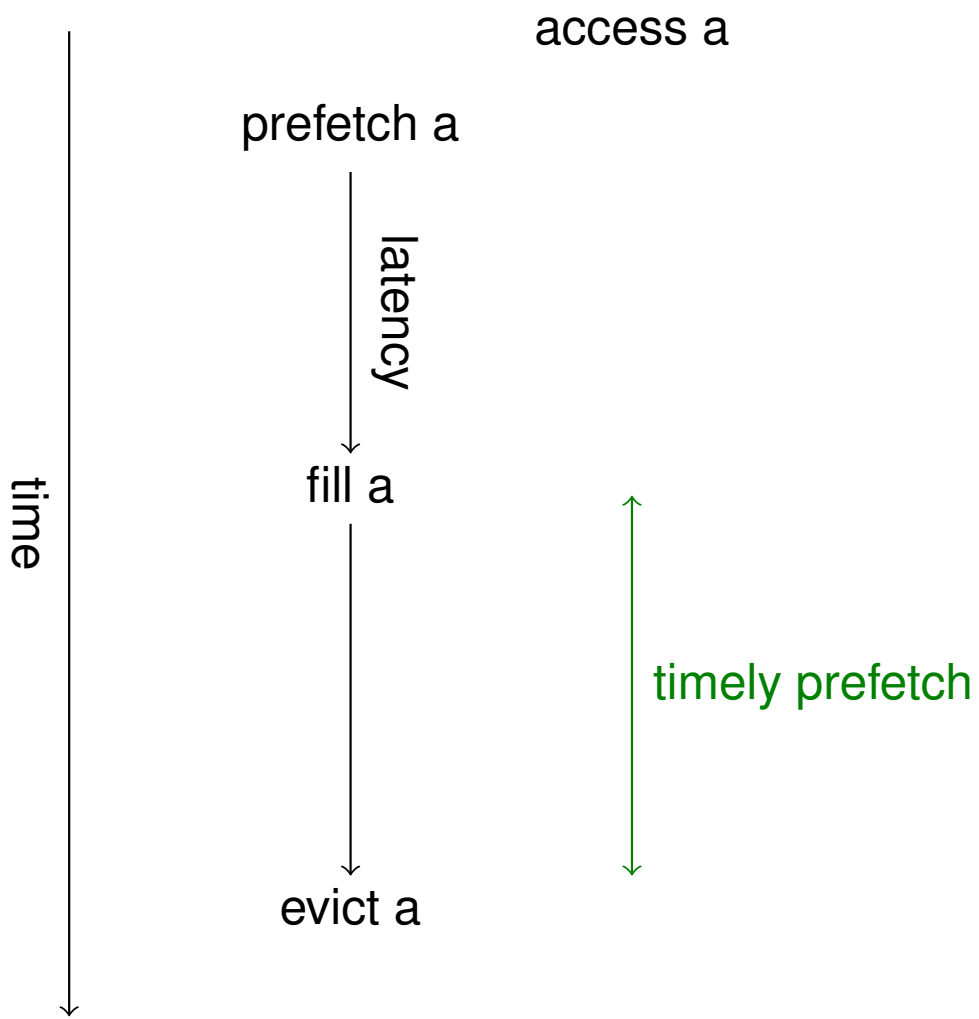
- Processors need to access instructions and data from memory
- Ideally processors would need a very large memory with a low access latency
  - → Not possible due to technology limitations
- Multi-level cache hierarchies approach this goal thanks to instr./data locality
  - → Still many long-latency accesses
- Computer architects came with a solution to this problem: prefetching
  - → *P*redict which memory addresses will be accessed by the processor and fetch them before the processor requests them



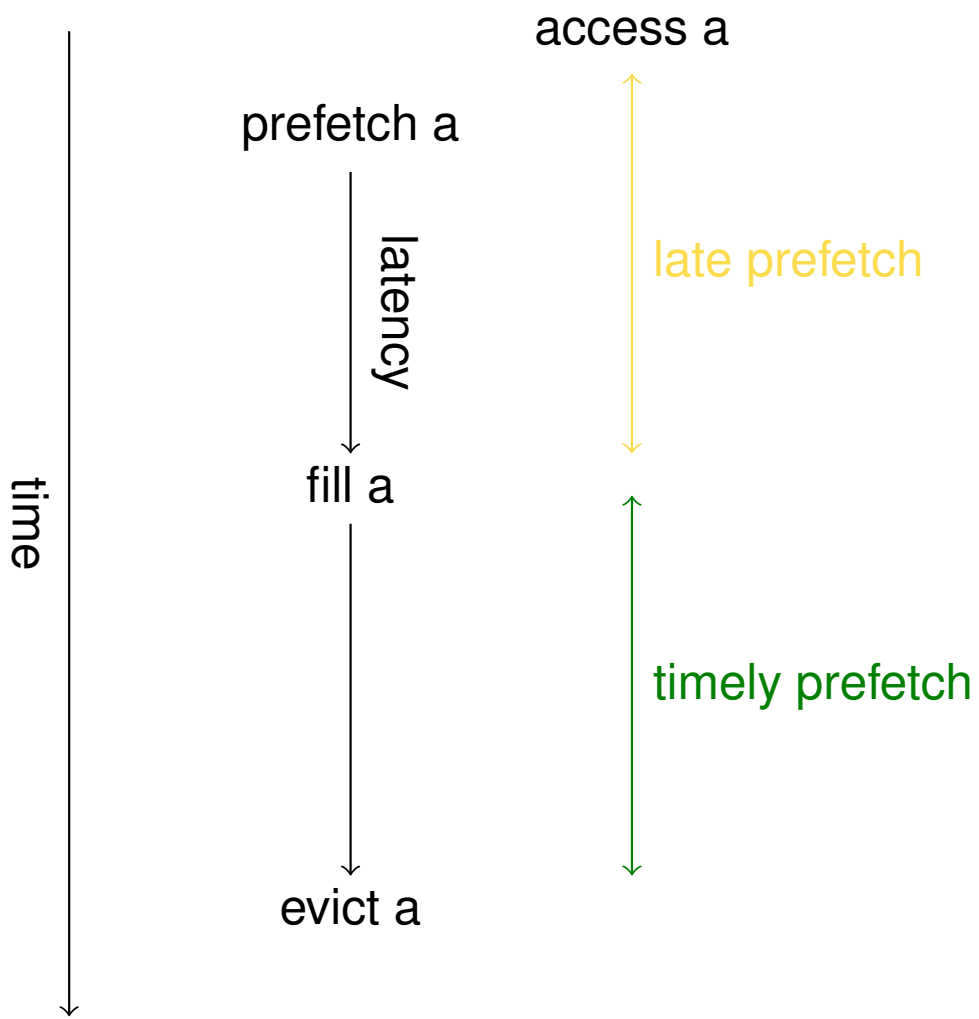
WHAT AND WHEN (TIMELINESS) TO PREFETCH



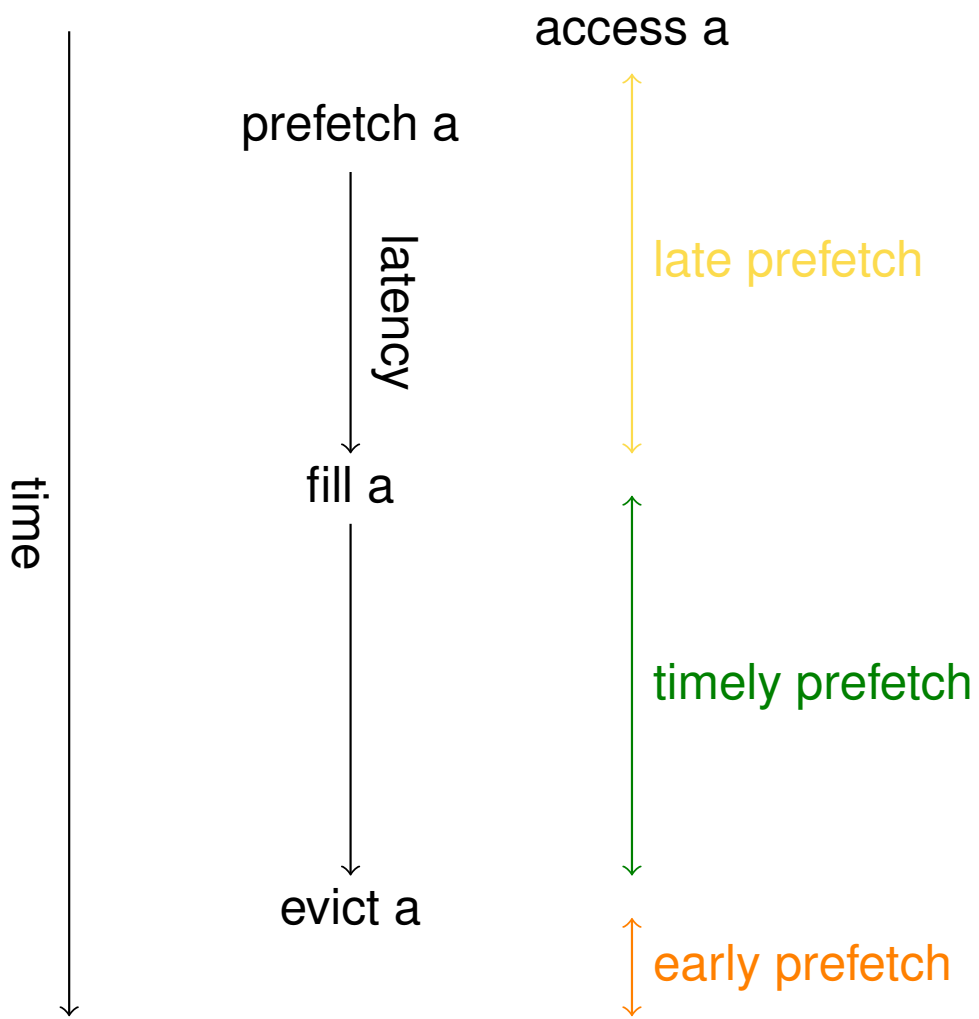
WHAT AND WHEN (TIMELINESS) TO PREFETCH



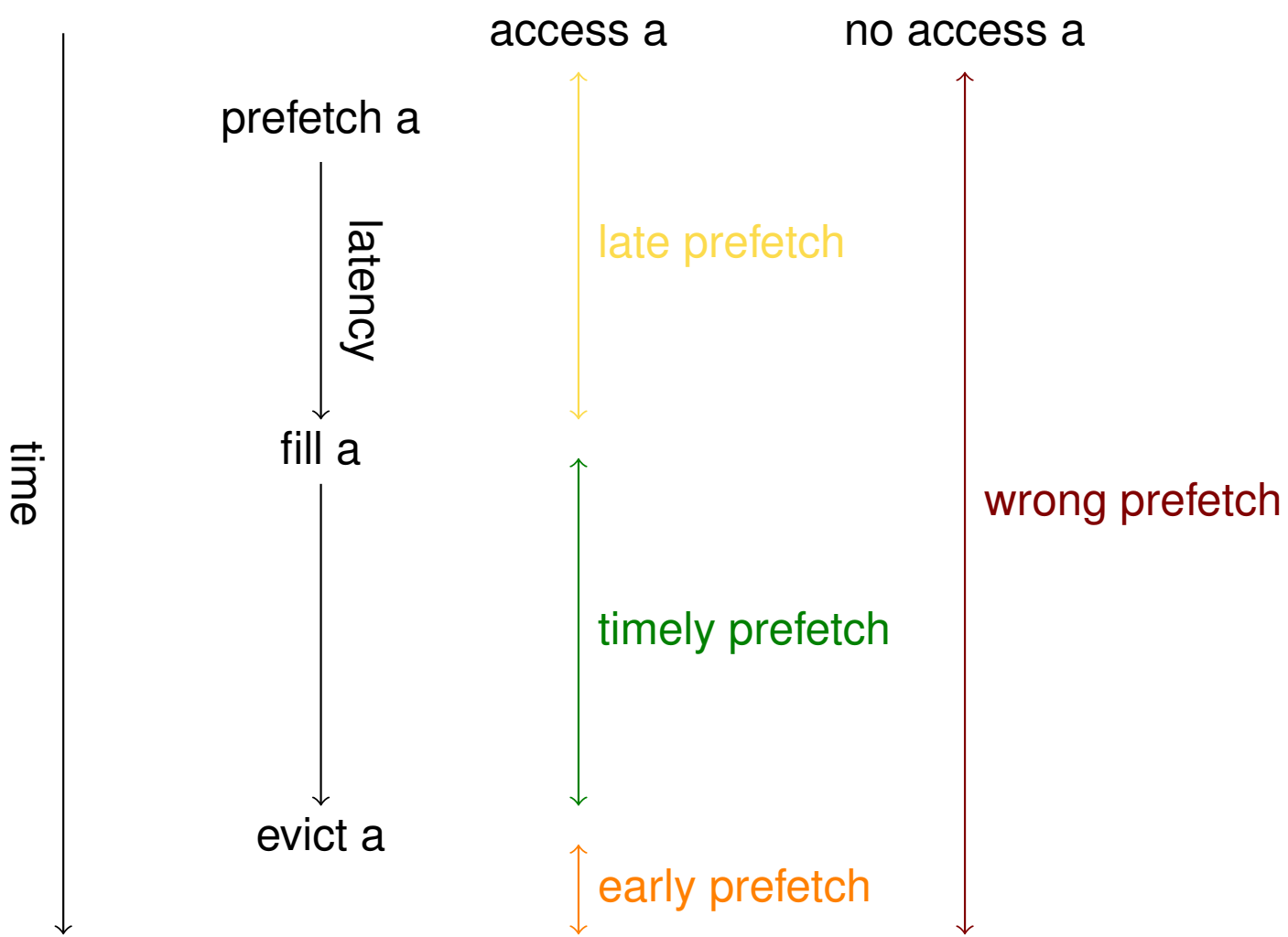
WHAT AND WHEN (TIMELINESS) TO PREFETCH



WHAT AND WHEN (TIMELINESS) TO PREFETCH

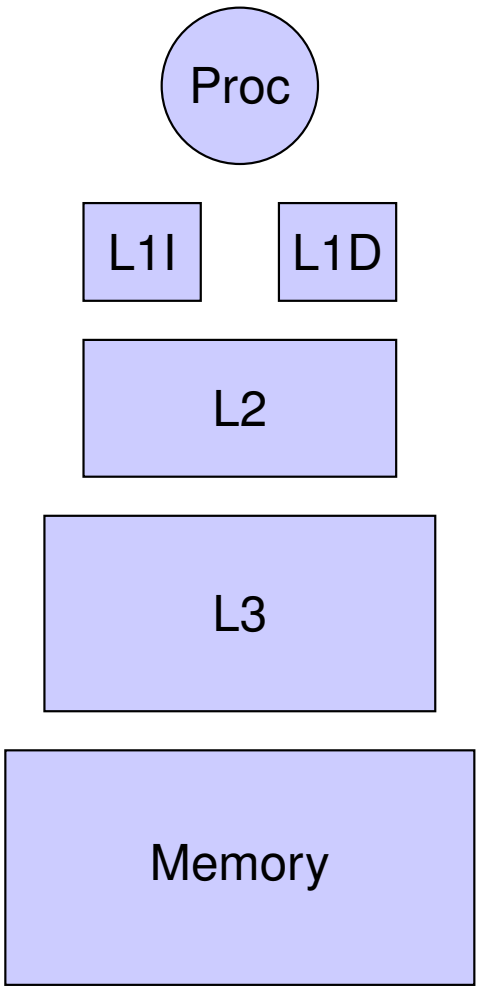


WHAT AND WHEN (TIMELINESS) TO PREFETCH



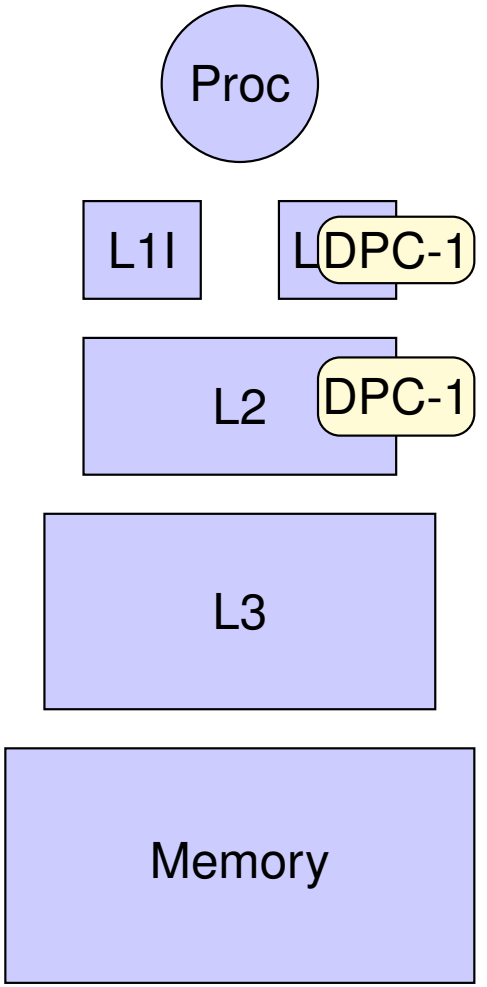
PREFETCHING CHAMPIONSHIPS

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria



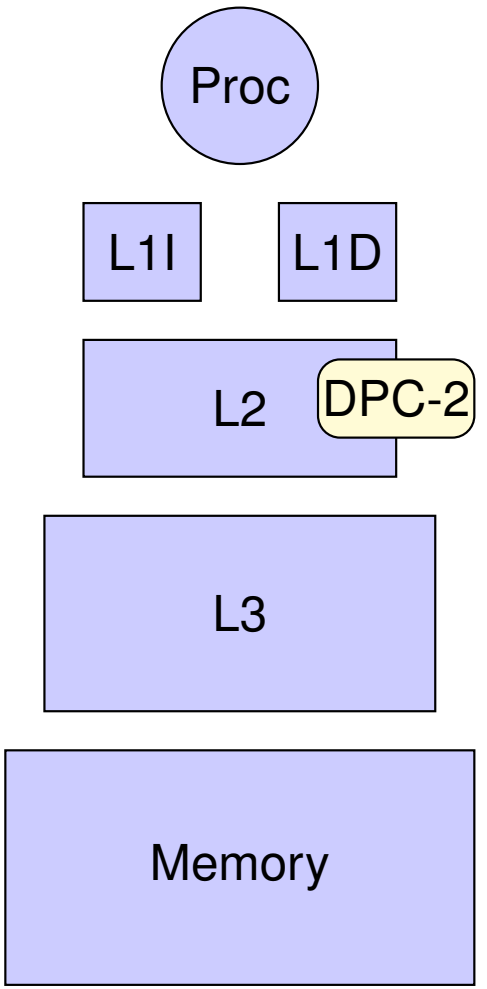
PREFETCHING CHAMPIONSHIPS

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- CHAMPIONSHIPS
  - 2009 – 1st Data Prefetching Championship (DPC-1)



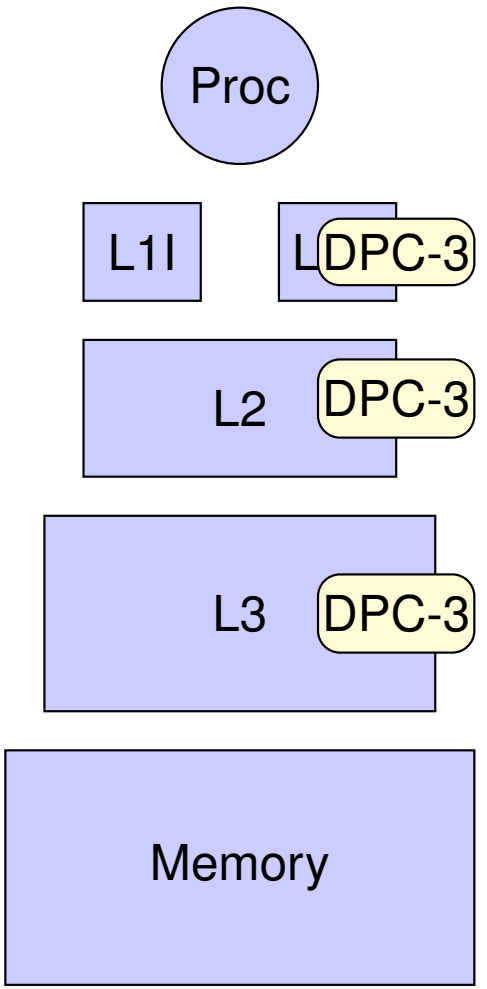
Prefetching Championships

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- Championships
  - 2009 – 1st Data Prefetching Championship (DPC-1)
  - 2015 – 2nd Data Prefetching Championship (DPC-2)



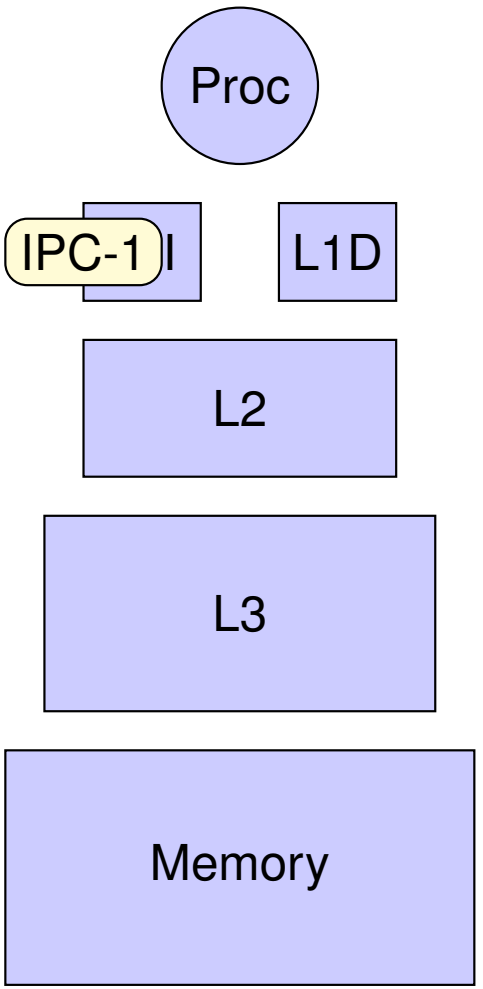
Prefetching Championships

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- Championships
  - 2009 – 1st Data Prefetching Championship (DPC-1)
  - 2015 – 2nd Data Prefetching Championship (DPC-2)
  - 2019 – 3rd Data Prefetching Championship (DPC-3)



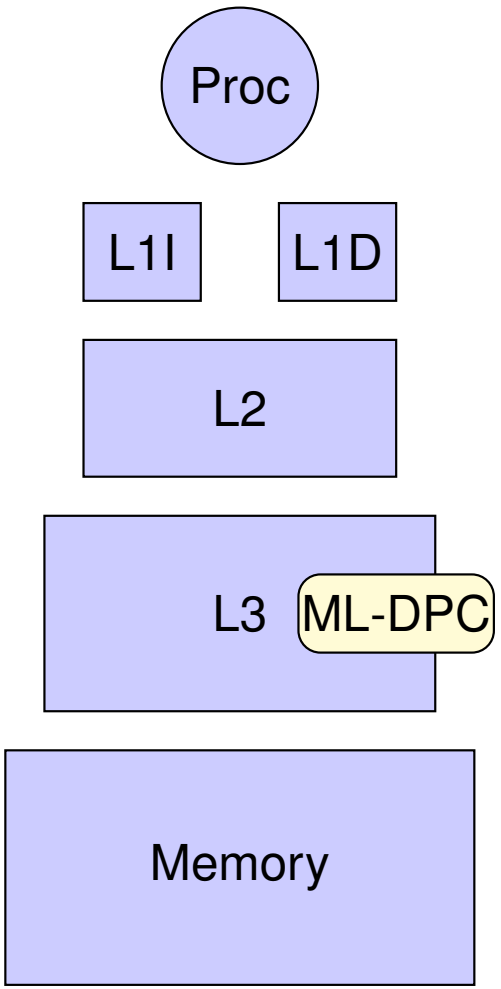
PREFETCHING CHAMPIONSHIPS

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- CHAMPIONSHIPS
  - 2009 – 1st Data Prefetching Championship (DPC-1)
  - 2015 – 2nd Data Prefetching Championship (DPC-2)
  - 2019 – 3rd Data Prefetching Championship (DPC-3)
  - 2020 – 1st Instruction Prefetching Championship (IPC-1)



PREFETCHING CHAMPIONSHIPS

- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- CHAMPIONSHIPS
  - 2009 – 1st Data Prefetching Championship (DPC-1)
  - 2015 – 2nd Data Prefetching Championship (DPC-2)
  - 2019 – 3rd Data Prefetching Championship (DPC-3)
  - 2020 – 1st Instruction Prefetching Championship (IPC-1)
  - 2021 – 1st ML-Data Prefetching Championship (ML-DPC)



The Entangling Instruction Prefetcher

(Addressing front-end latency)

Alberto Ros Alexandra Jimborean

University of Murcia, Spain

Overview

- Prefetching instructions is fundamental for performance
  - Even when a decoupled front-end is implemented

Overview

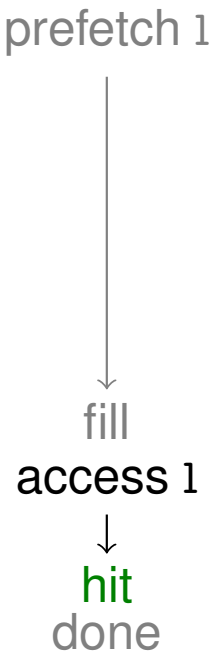
- Prefetching instructions is fundamental for performance
  - Even when a decoupled front-end is implemented
- Our contribution: An L1I Entangling prefetcher
  - Entangling: adaptive correlation based on latency
  - Winner of the 1st Instruction Prefetching Championship
  - A cost-effective prefetcher
  - Prefetcher code is available[1]

[1] https://github.com/alberto-ros/EntanglingInstructionPrefetcher

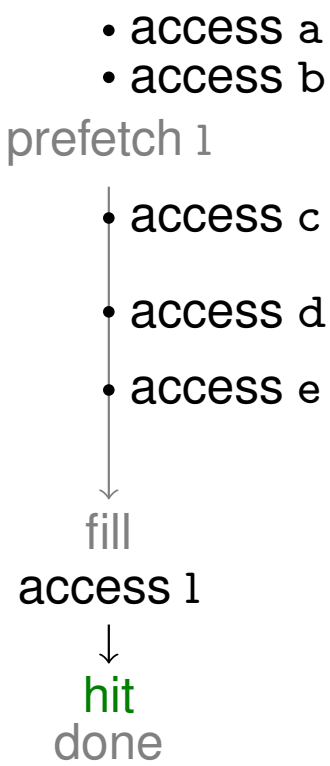
CONCEPT OF ENTANGLED ACCESSES

CONCEPT OF ENTANGLED ACCESSES

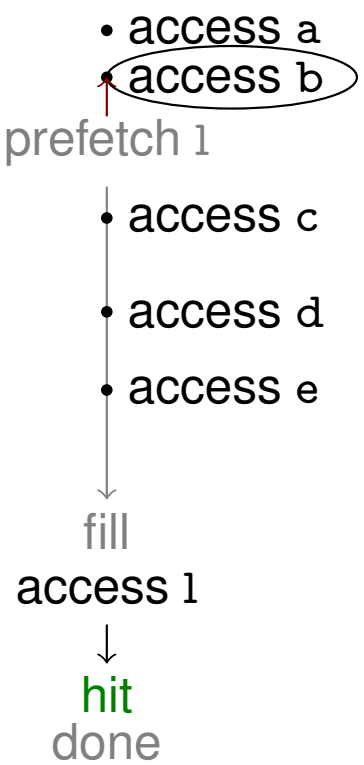
CONCEPT OF ENTANGLED ACCESSES



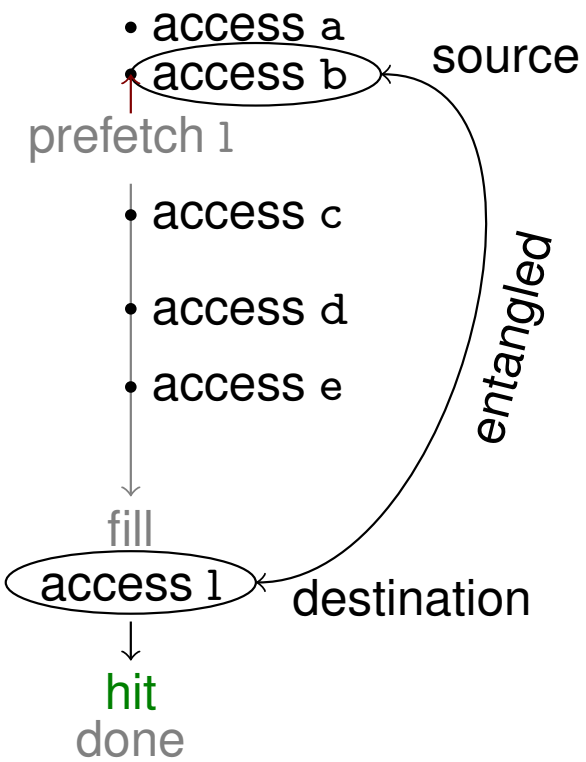
CONCEPT OF ENTANGLED ACCESSES



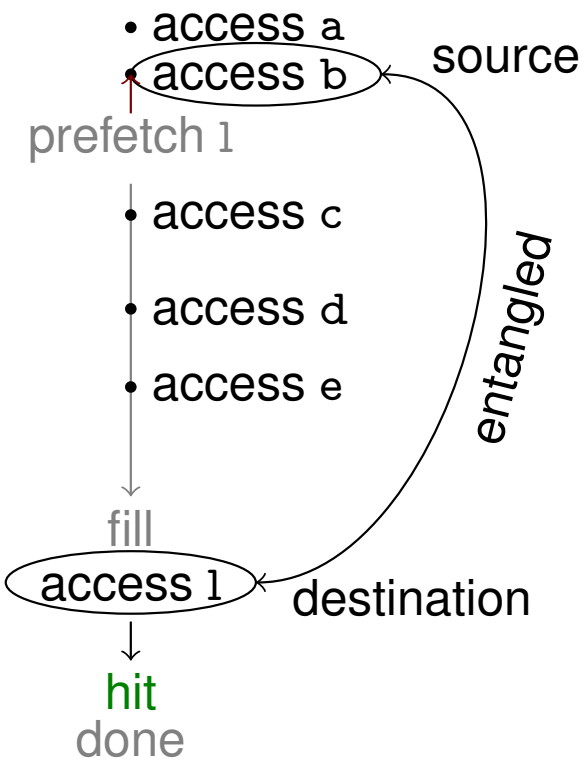
CONCEPT OF ENTANGLED ACCESSES

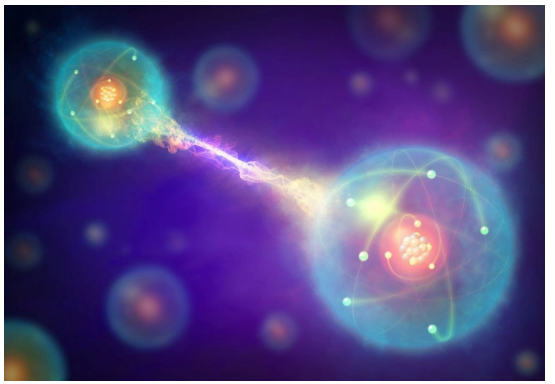


CONCEPT OF ENTANGLED ACCESSES



CONCEPT OF ENTANGLED ACCESSES

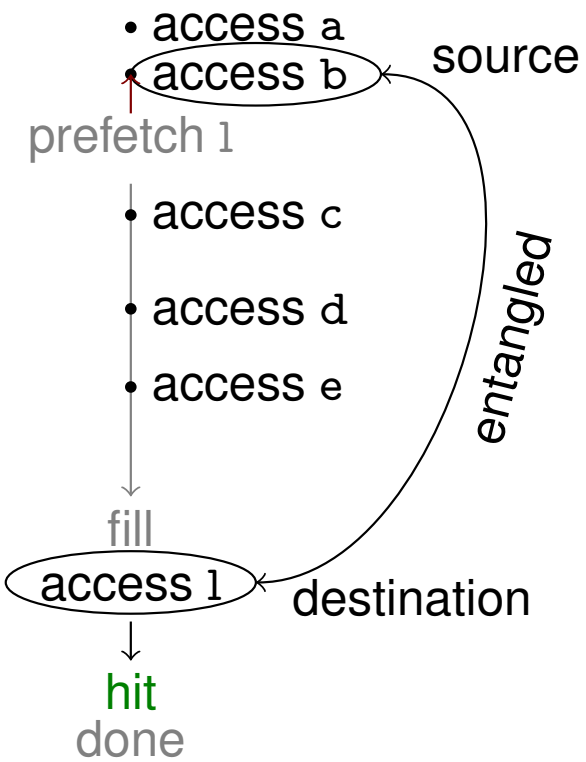


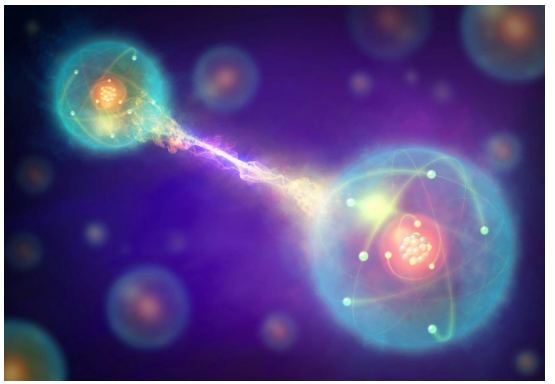


Quantum entanglement

(Image: © MARK GARLICK/SCIENCE PHOTO LIBRARY/Getty)

Concept of entangled accesses



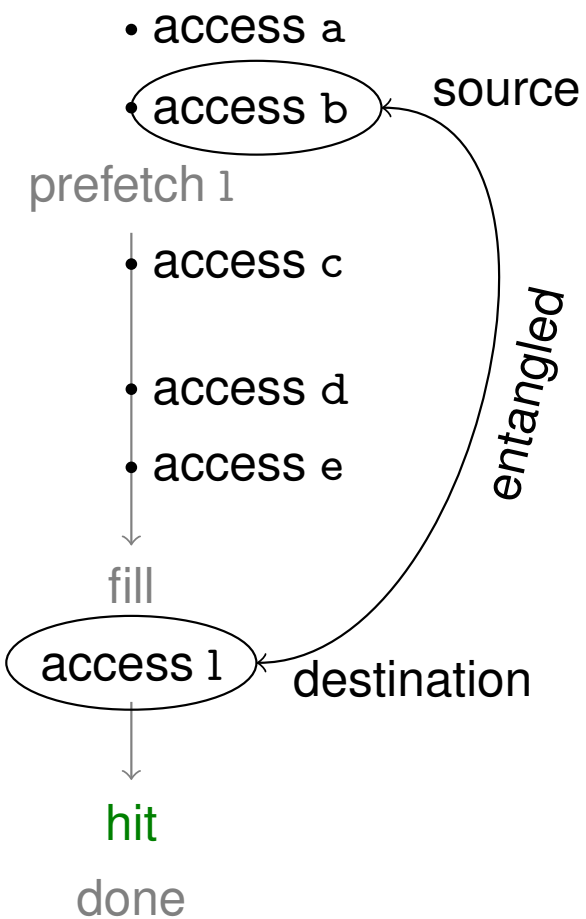


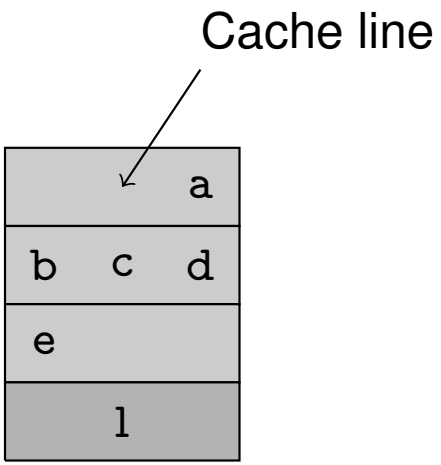
Quantum entanglement

(Image: © MARK GARLICK/SCIENCE PHOTO LIBRARY/Getty)

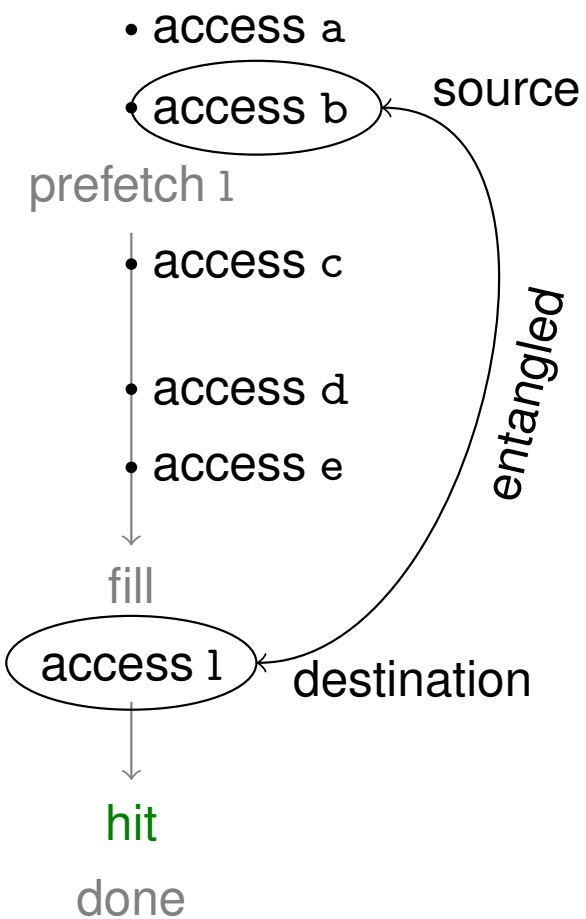
The Entangling Prefetcher for Instructions

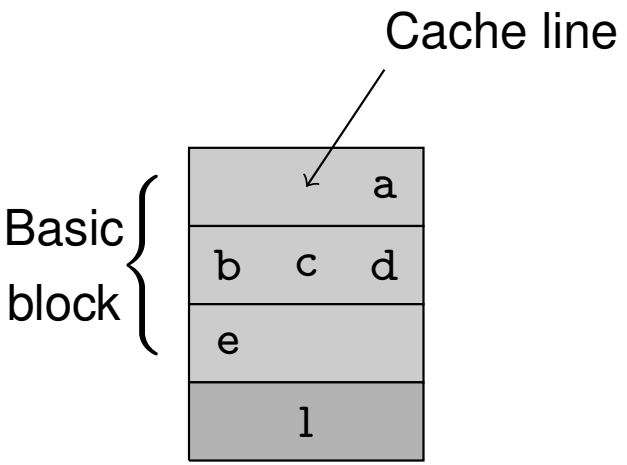
ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS



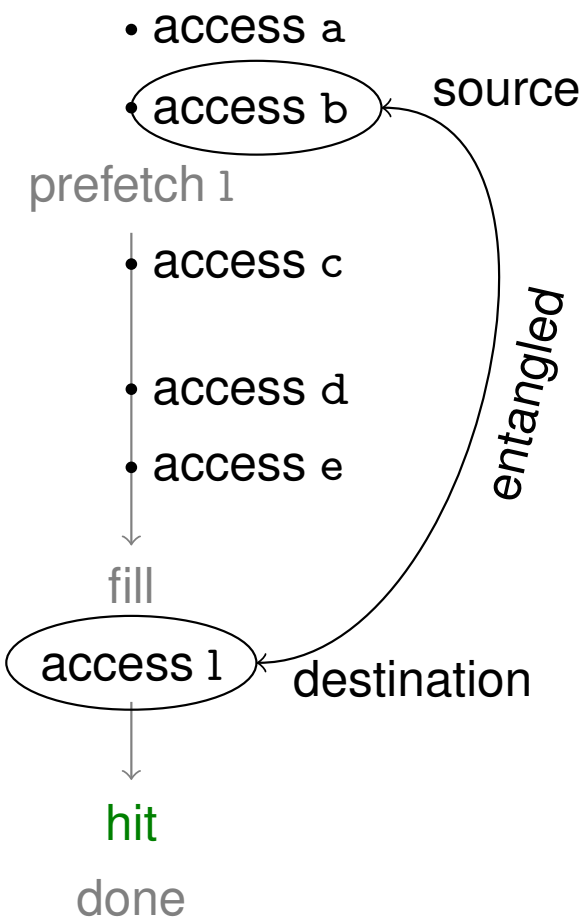


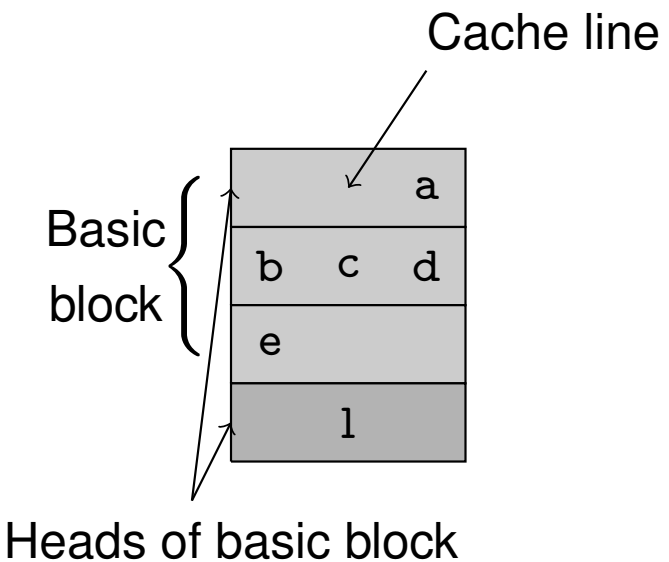
ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS



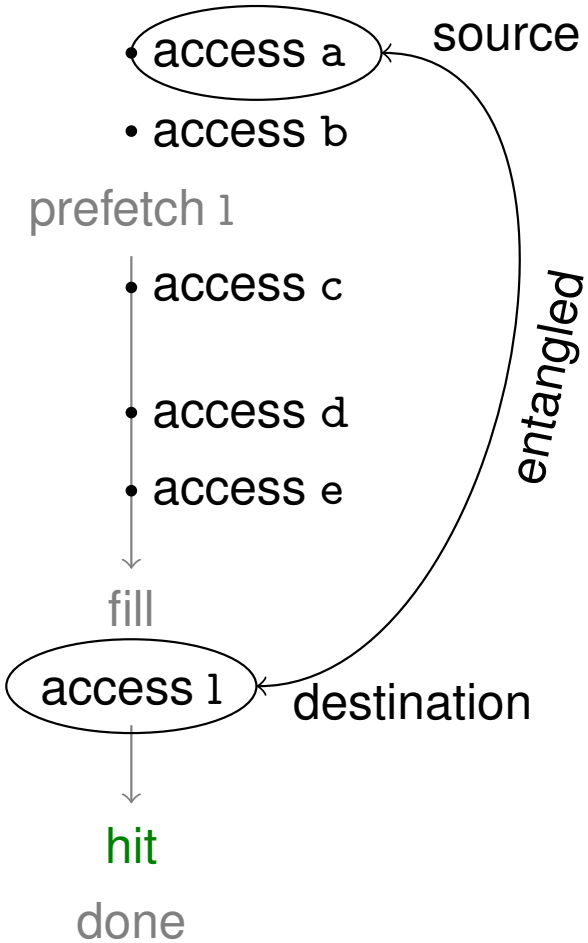


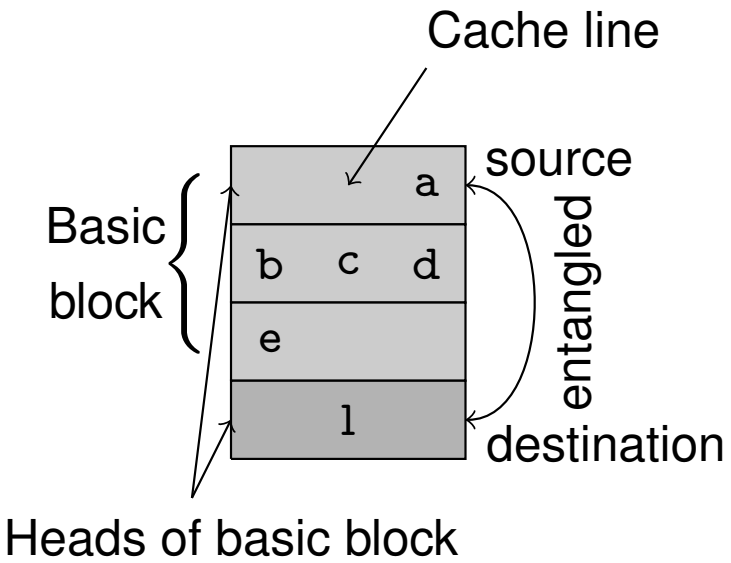
ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS



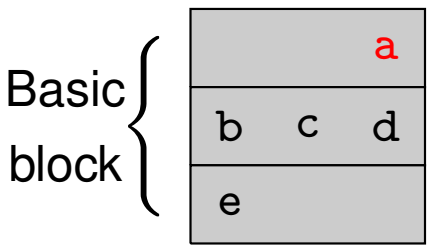


ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS

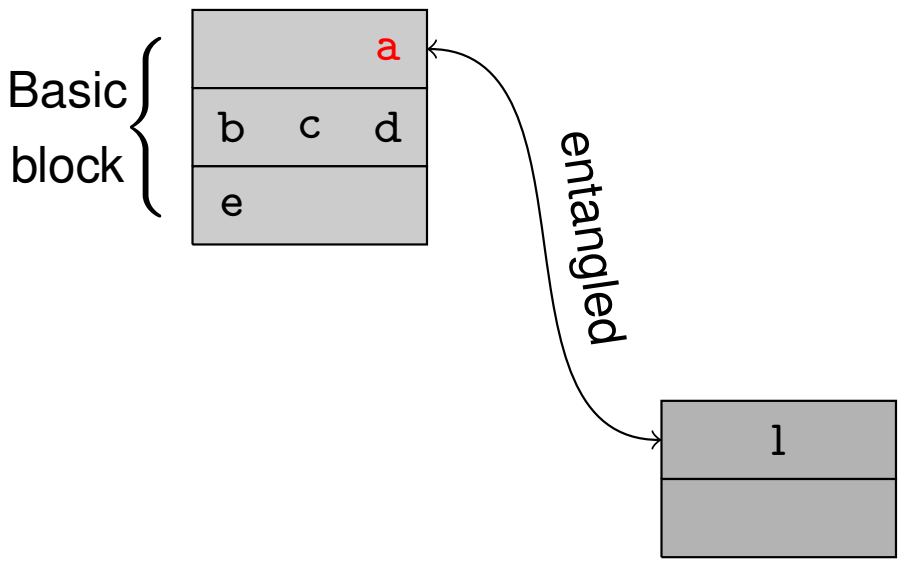




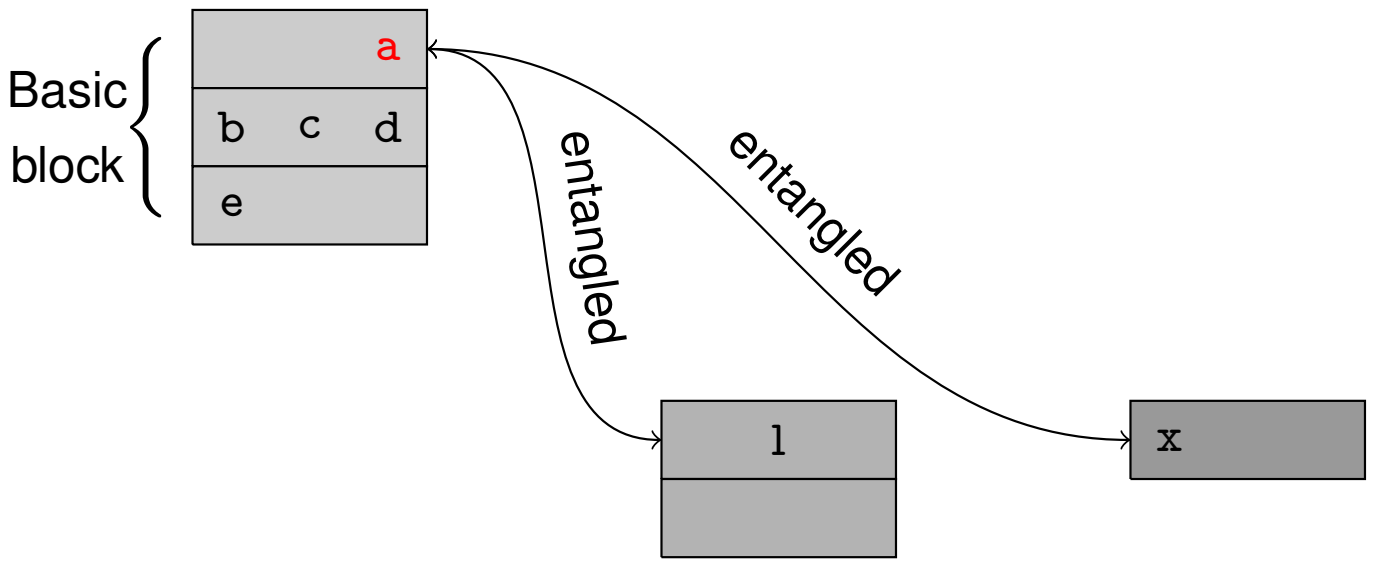
WHAT TO PREFETCH ON AN ACCESS TO a?



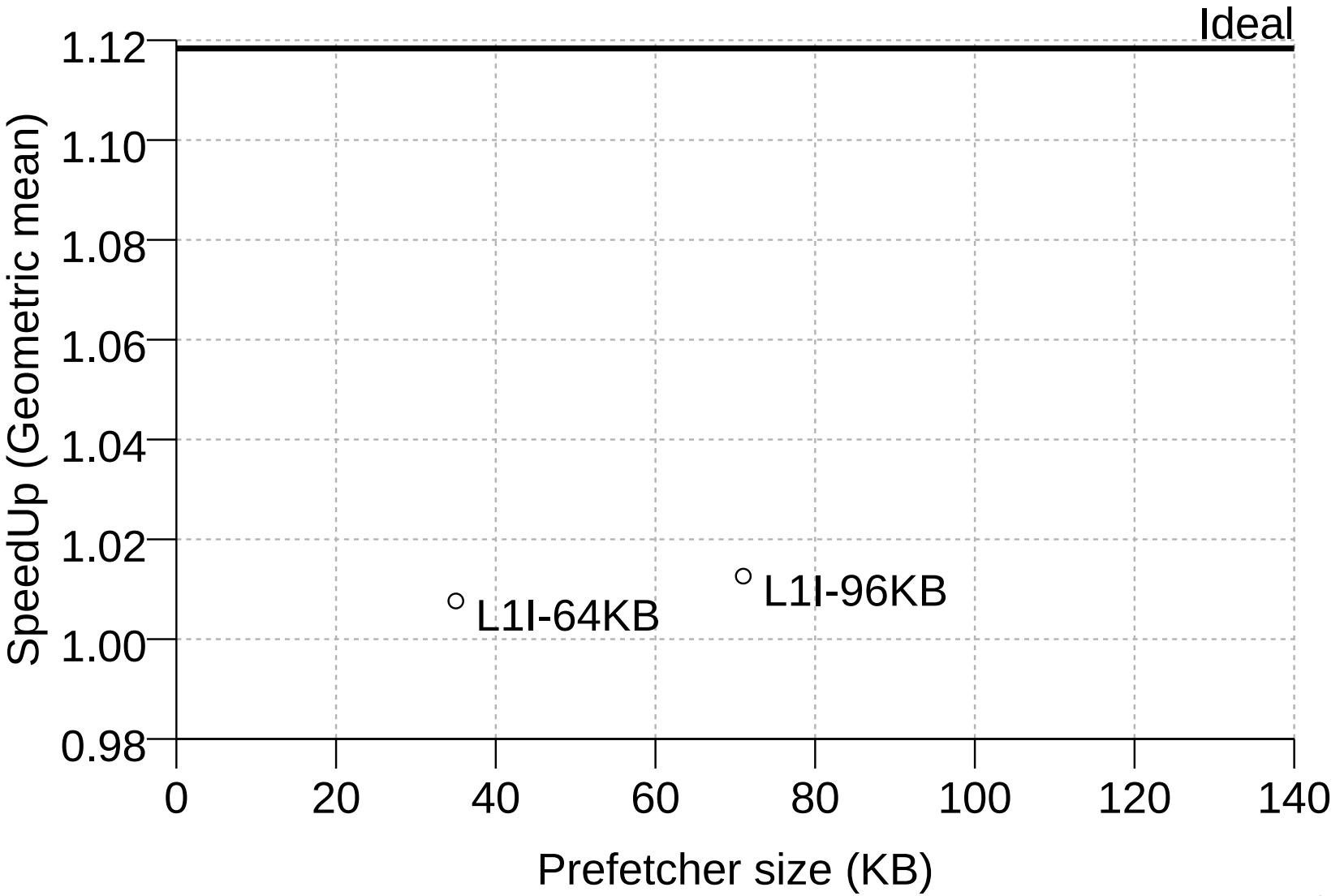
WHAT TO PREFETCH ON AN ACCESS TO a?



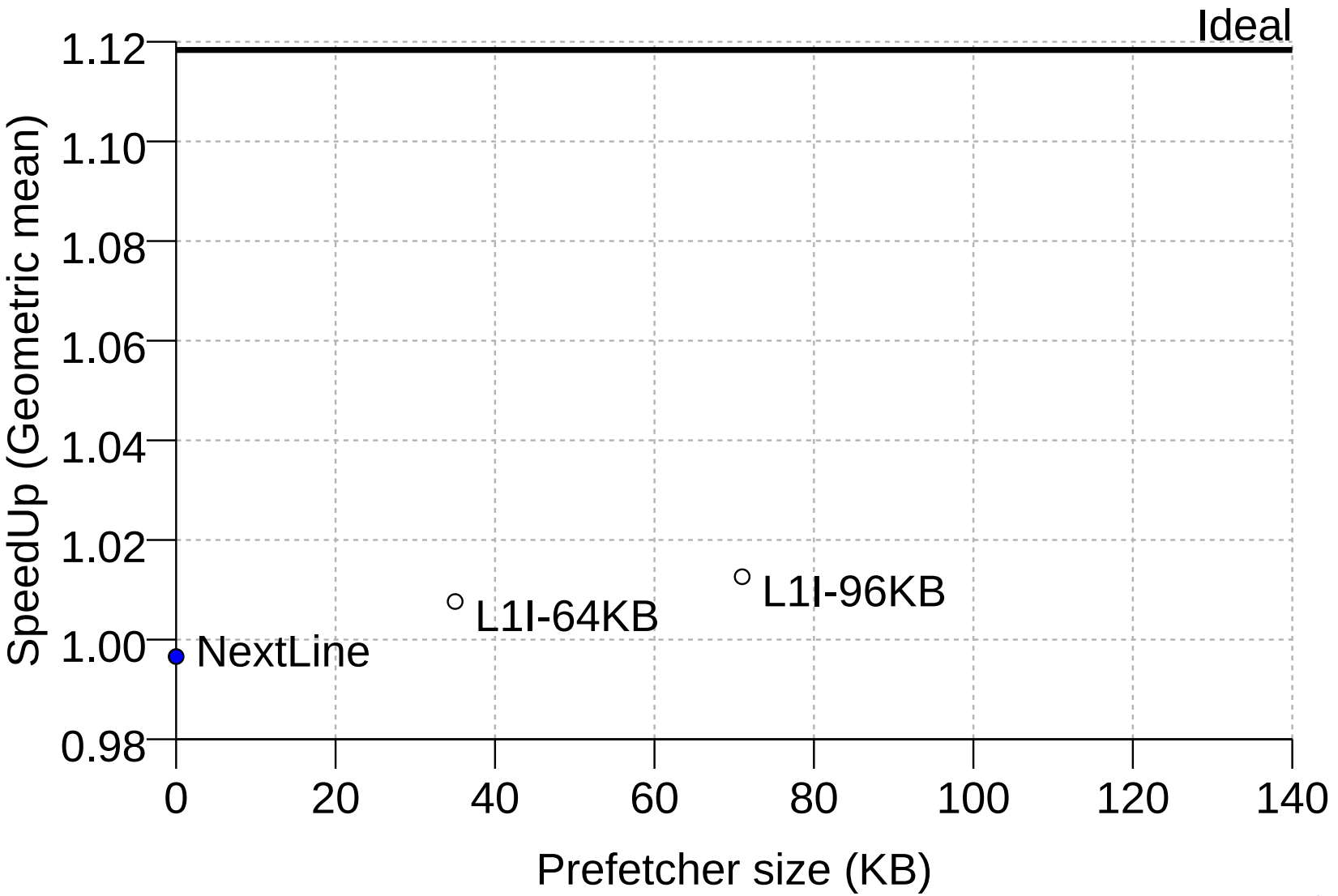
WHAT TO PREFETCH ON AN ACCESS TO a?



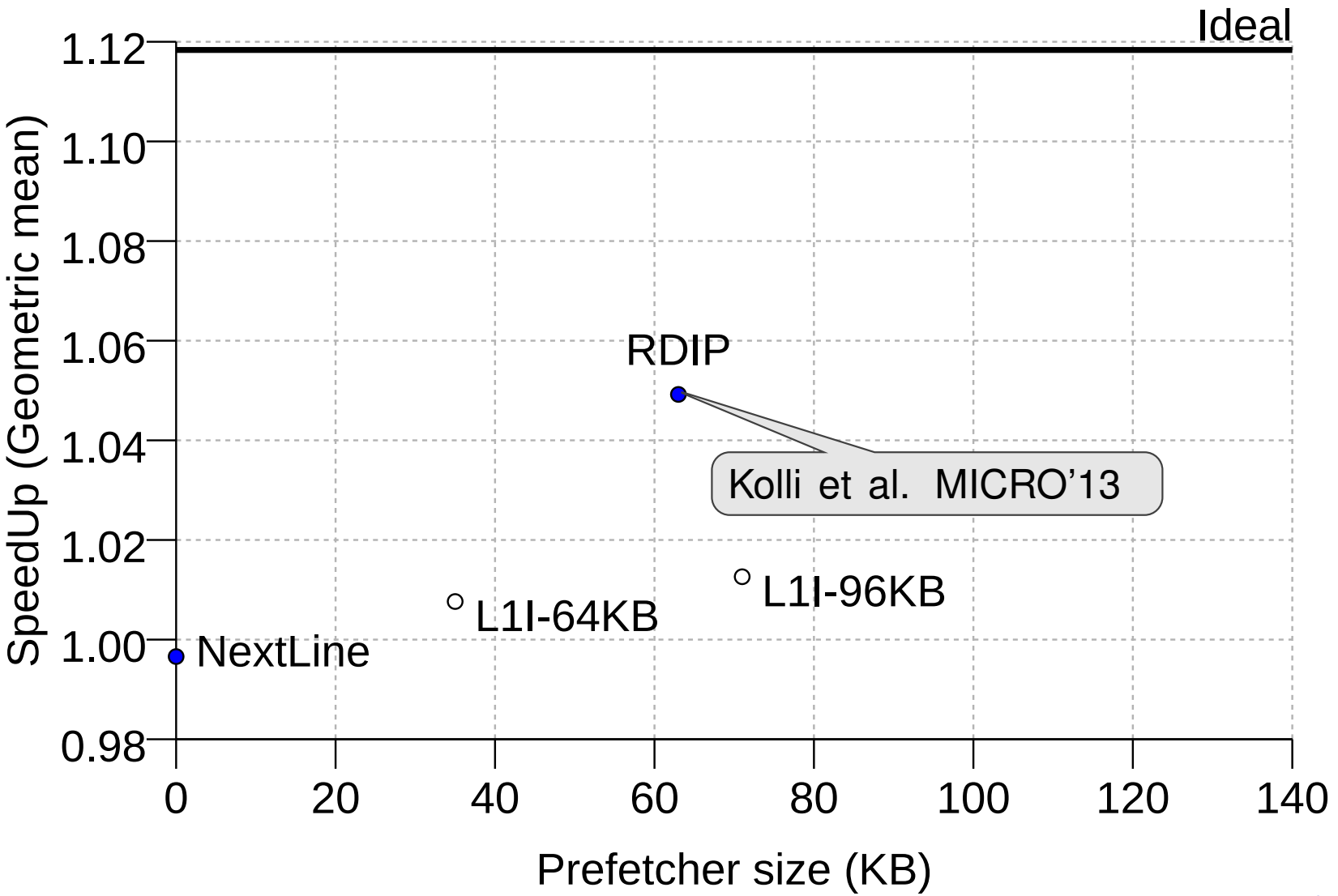
RESULTS: IPC VS MEMORY OVERHEAD



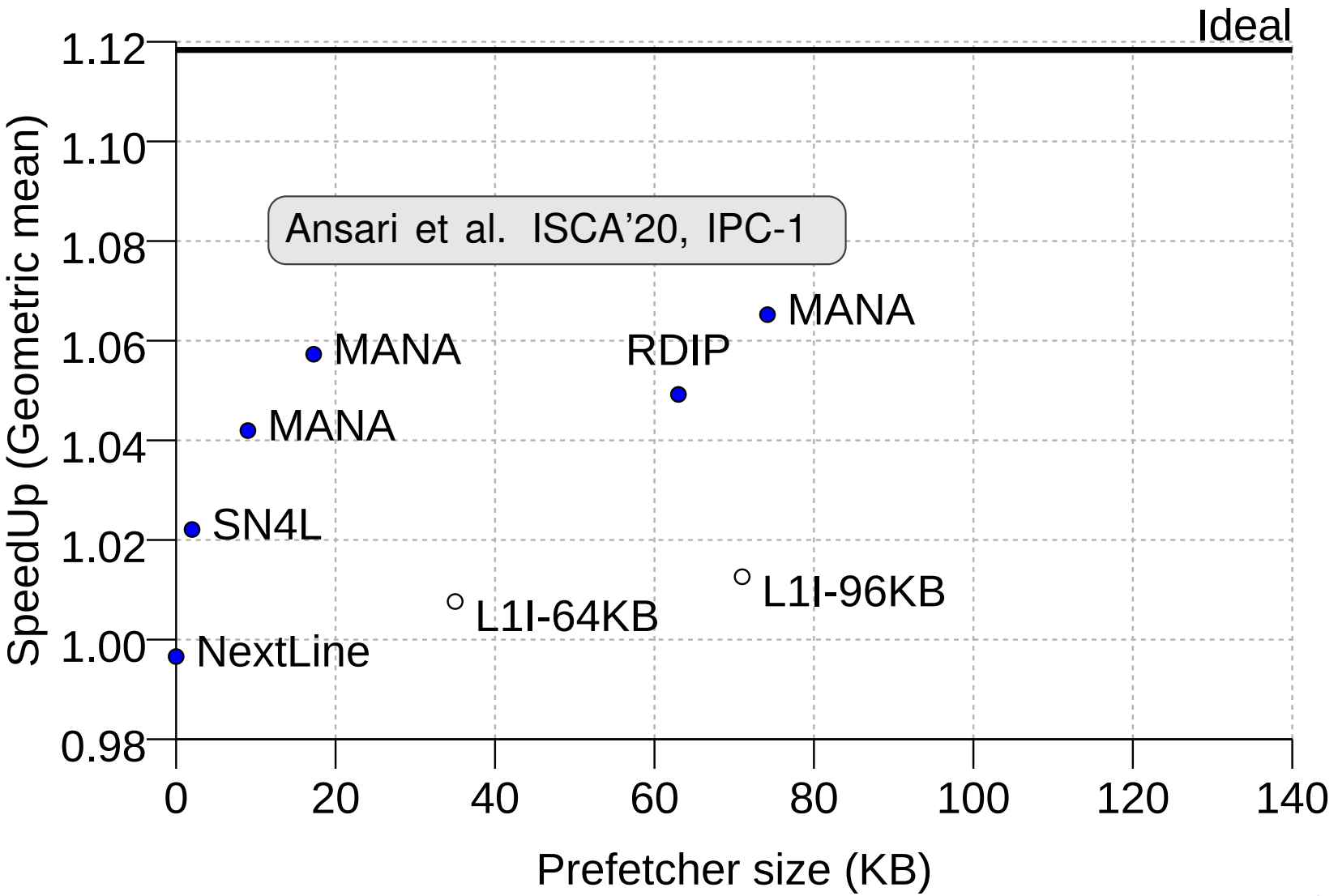
RESULTS: IPC VS MEMORY OVERHEAD



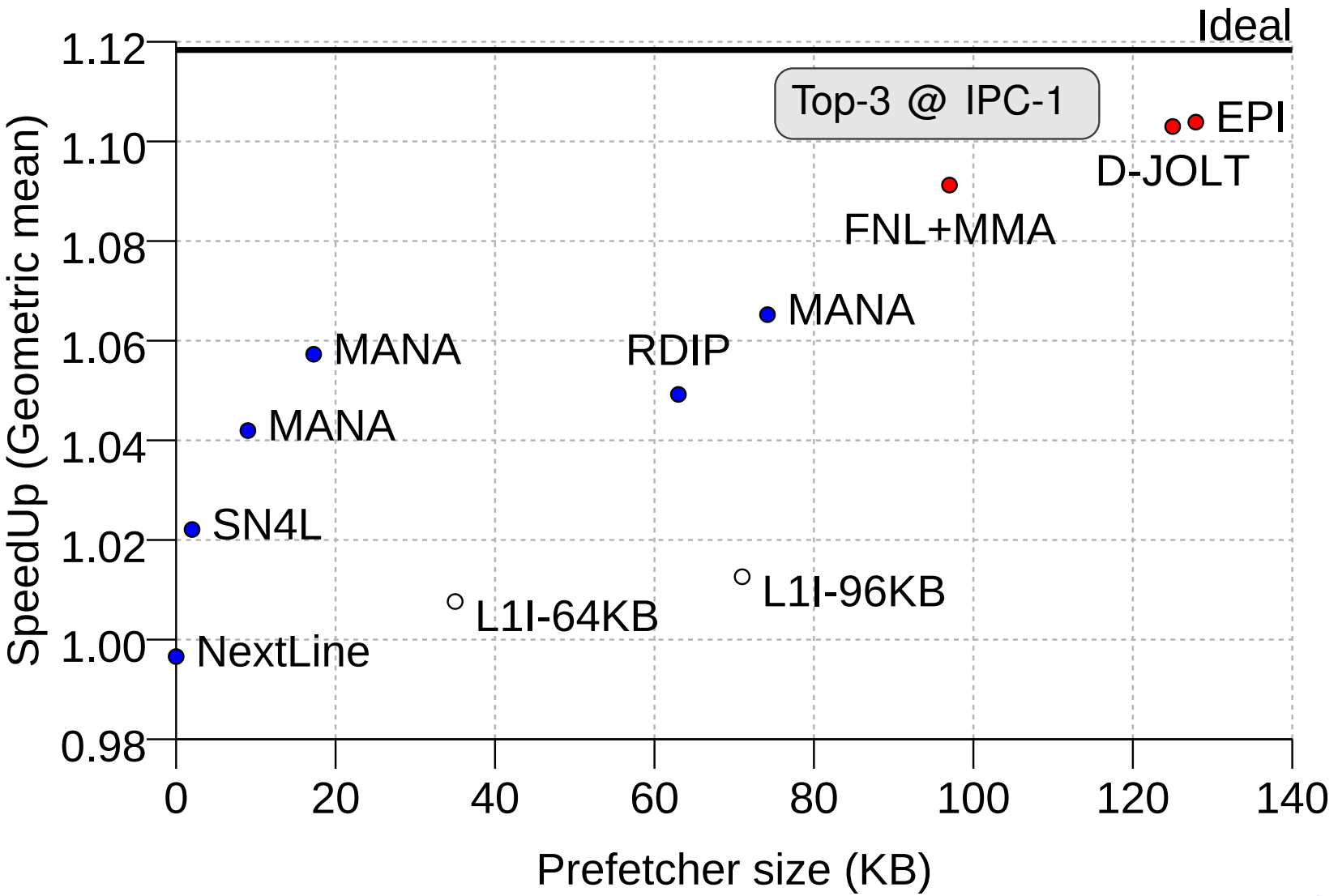
RESULTS: IPC VS MEMORY OVERHEAD



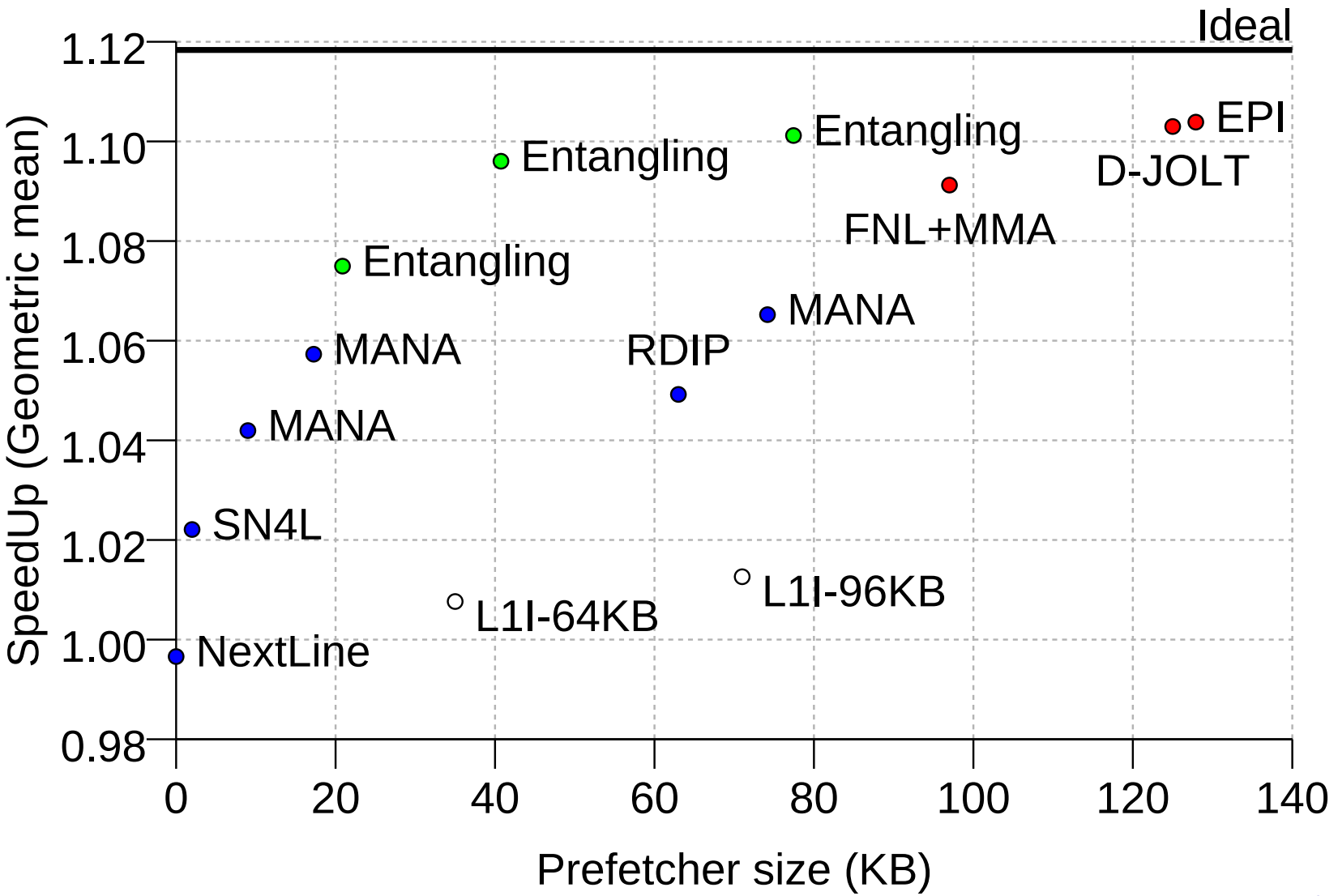
RESULTS: IPC VS MEMORY OVERHEAD



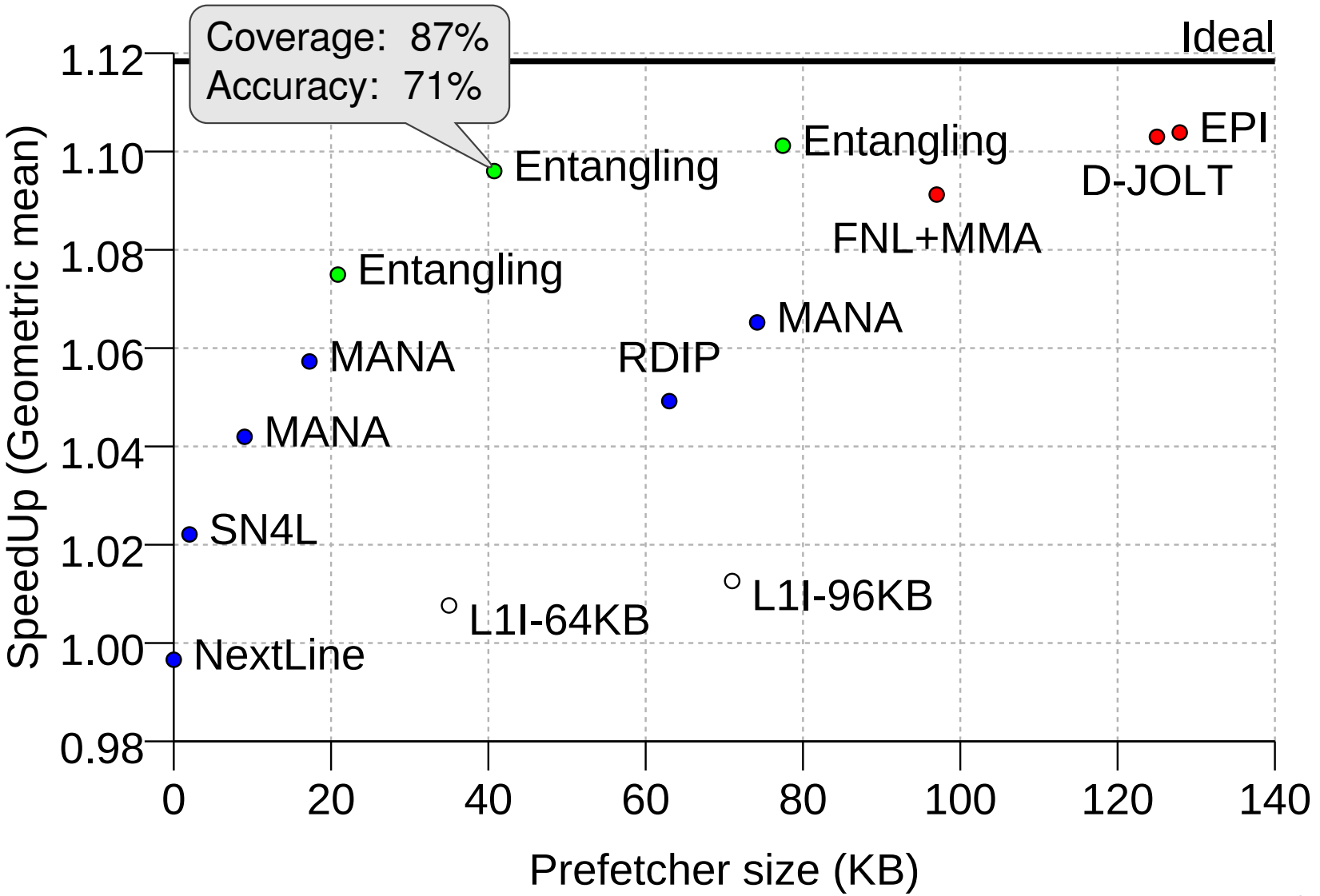
RESULTS: IPC VS MEMORY OVERHEAD



RESULTS: IPC VS MEMORY OVERHEAD



RESULTS: IPC VS MEMORY OVERHEAD



BERTI: AN ACCURATE LOCAL-DELTA DATA PREFETCHER

(ADDRESSING BACK-END MEMORY)

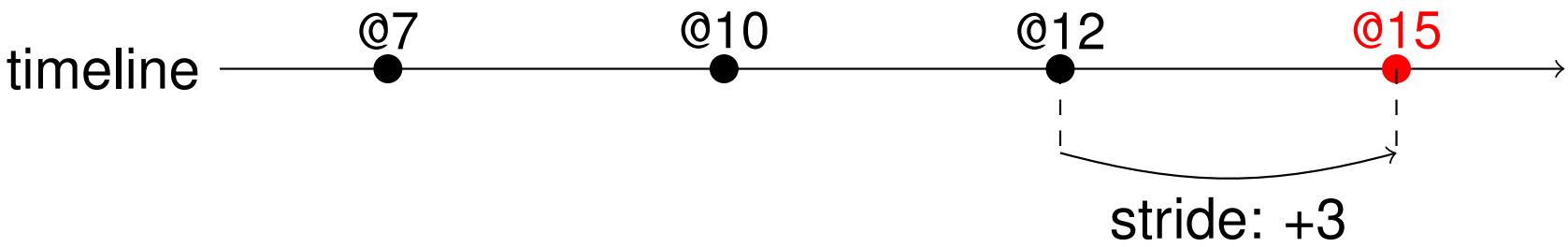
Agustín Navarro-Torres[1] Biswabandan Panda[2]
Jesús Alastruey-Benedé[3] Pablo Ibañez[3]
Víctor Viñals-Yúfera[3] Alberto Ros[1]

[1]University of Murcia, Spain
[2]Indian Institute of Technology Bombay, India
[3]University of Zaragoza, Spain

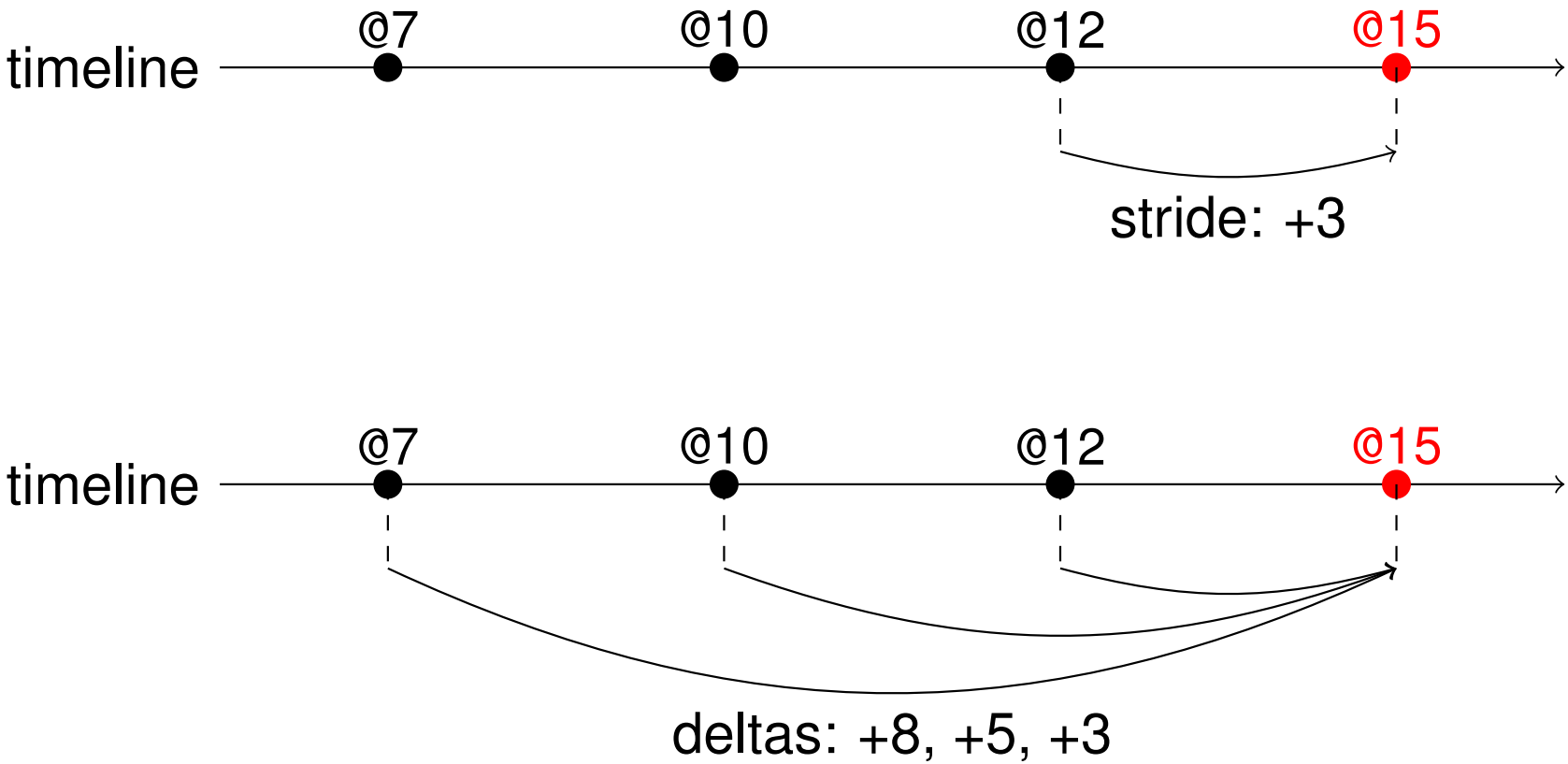
BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

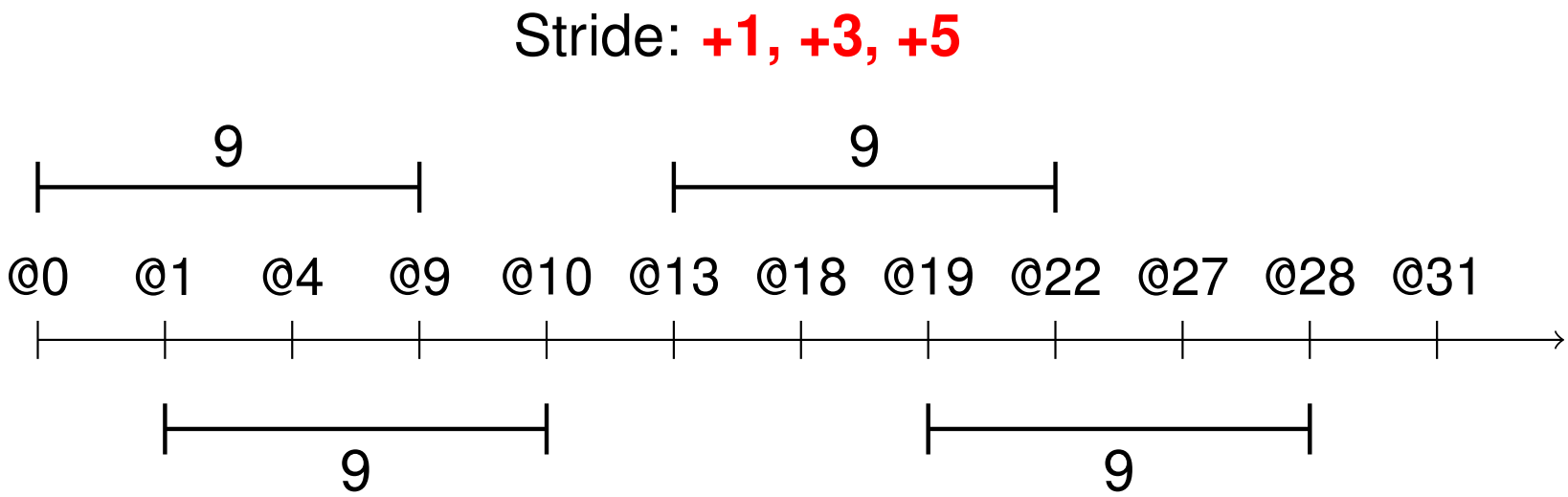
- An L1D prefetcher that orchestrates data prefetching from all cache levels
- Advantages of L1D prefetching
  - Training with all processor references
  - Using the Instruction Pointer (IP) information
  - Operating with virtual addresses: cross-page prefetching

Definition of delta



Definition of delta

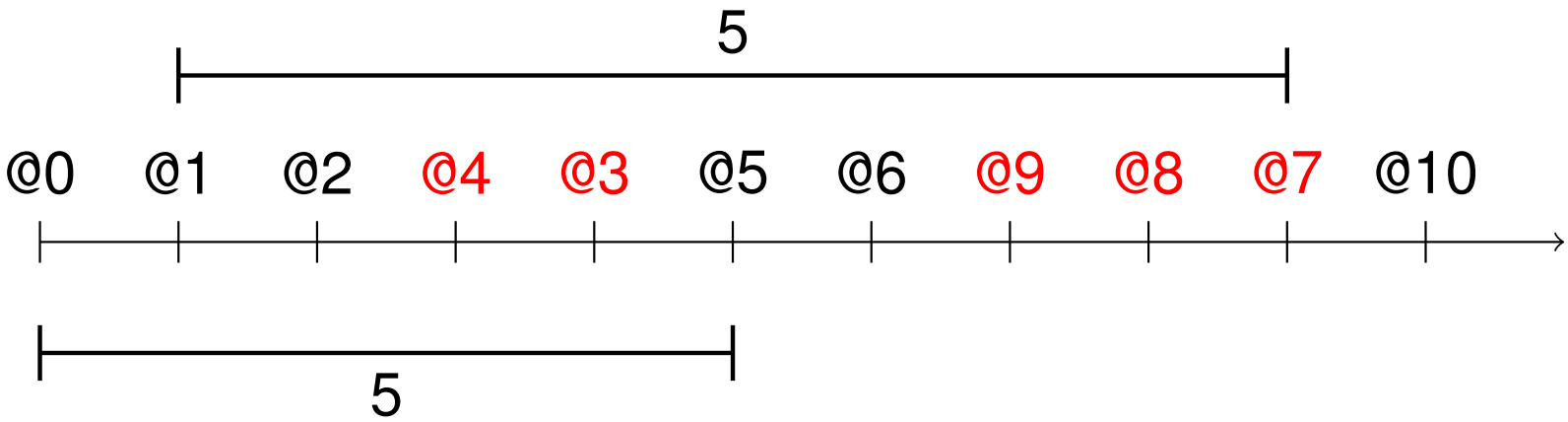




With which delta should I prefetch?
delta = 1 + 3 + 5 = 9 → **always hit**

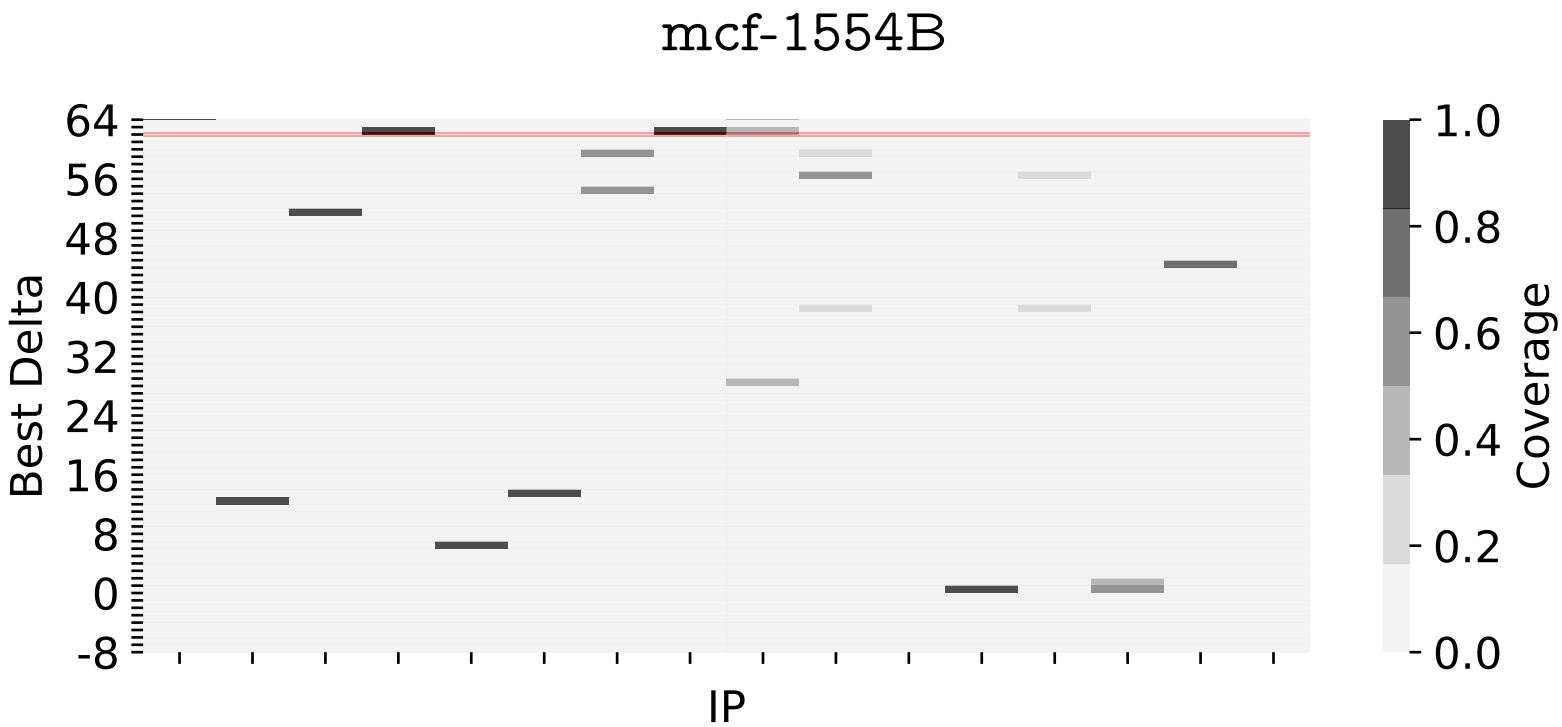
BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

Addresses reordered by the out-of-order processor



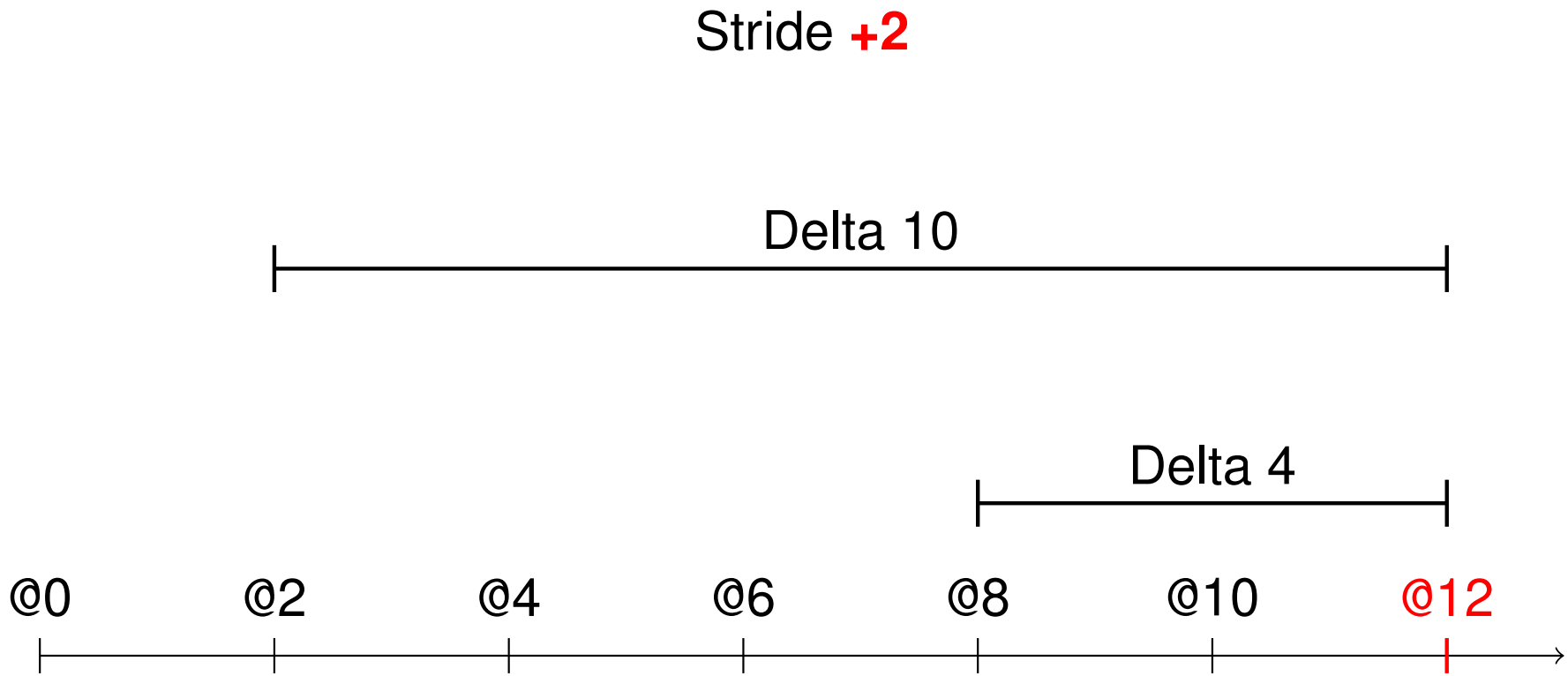
Stride prefetch requires specific order
Berti can prefetch with delta = 5, for example

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

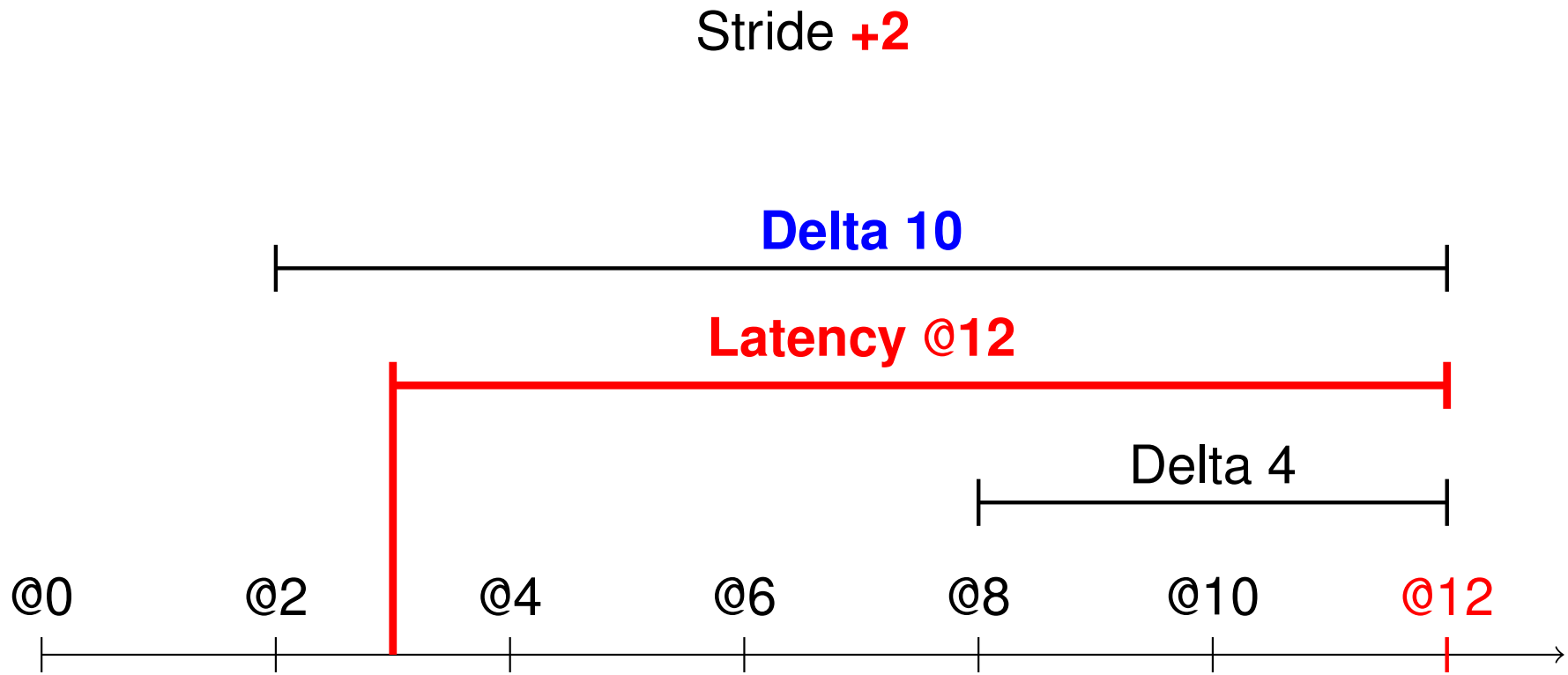


- **Red line**: best delta by BOP[1], coverage: 2%
- **Black lines**: per-IP local deltas, coverage: 10%

[1] Winner of 2nd Data Prefetching Championship (DPC-2)



How far in advance should I prefetch address 12?



How far in advance should I prefetch address 12?
Depends on its latency

TRAINING

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

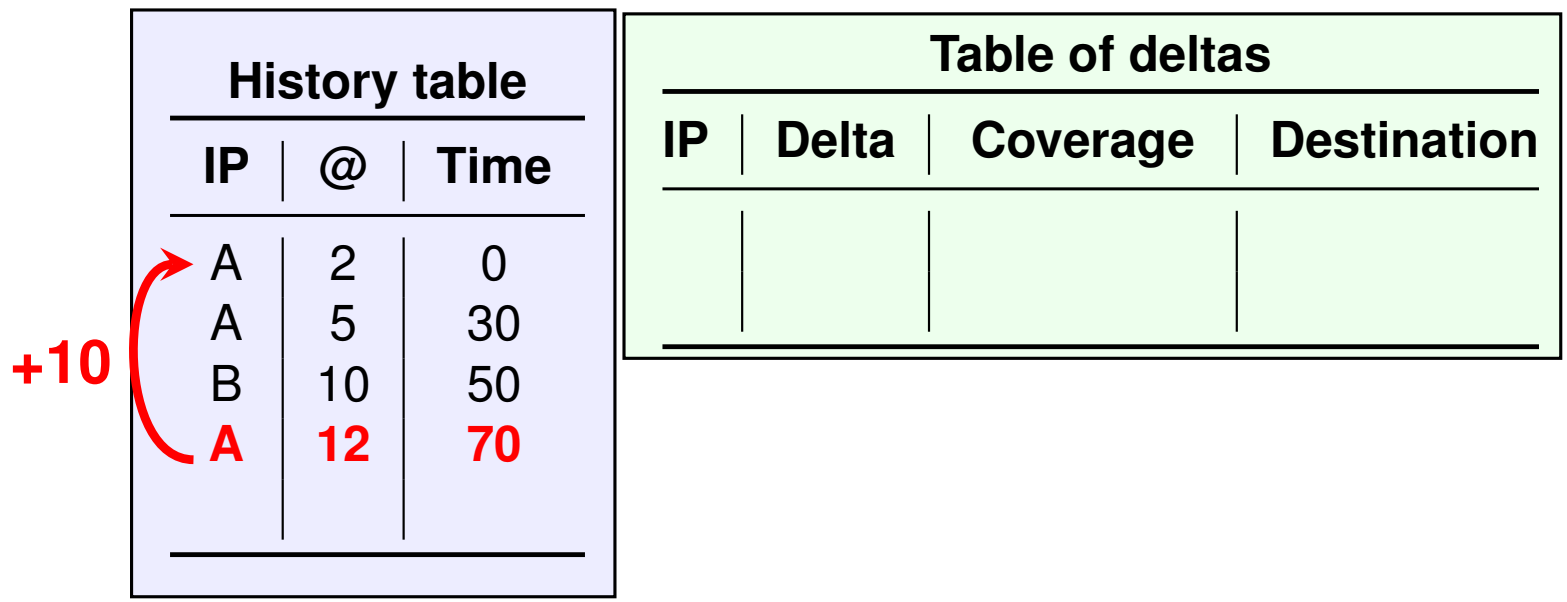
IP	@	Time
A	2	0
A	5	30
B	10	50

Table of deltas

IP	Delta	Coverage	Destination

TRAINING

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas



TRAINING

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

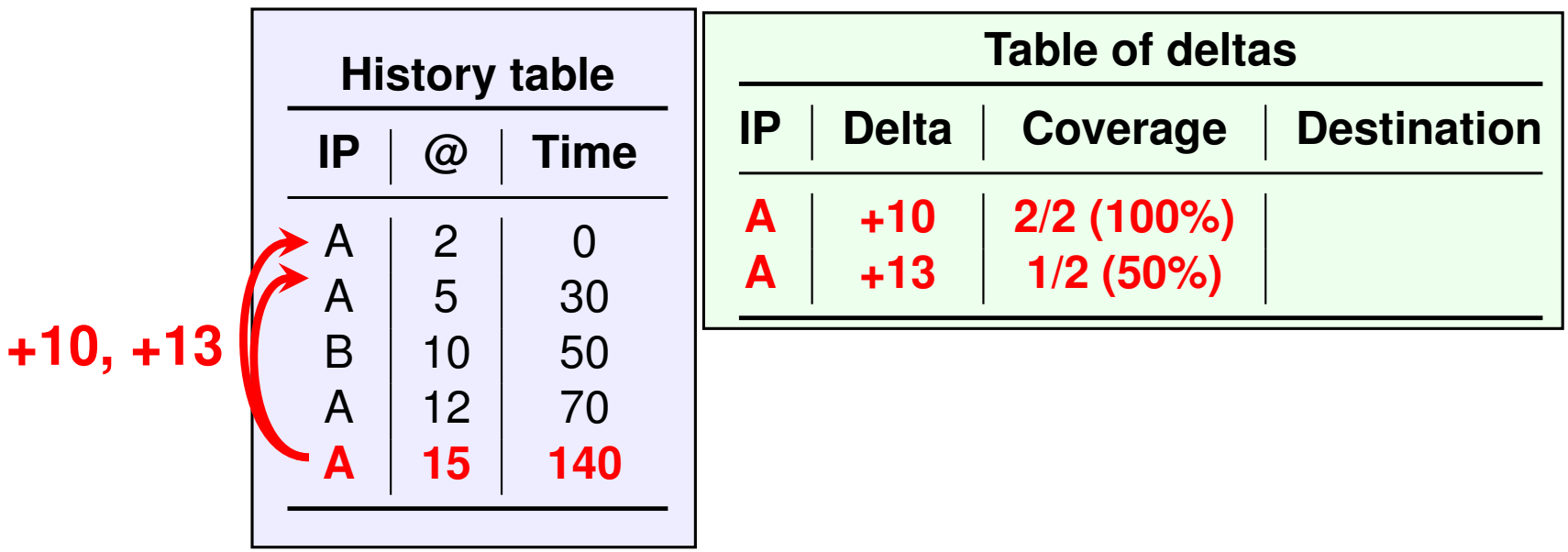
IP	@	Time
A	2	0
A	5	30
B	10	50
A	**12**	**70**

Table of deltas

IP	Delta	Coverage	Destination
A	**+10**	**1/1 (100%)**	

TRAINING

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas



ISSUING PREFETCH REQUESTS

1. Select deltas
2. Orchestration

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

IP	Delta	Coverage	Destination
A	+10	2/2 (100%)	
A	+13	1/2 (50%)	

ISSUING PREFETCH REQUESTS

1. Select deltas
2. Orchestration

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

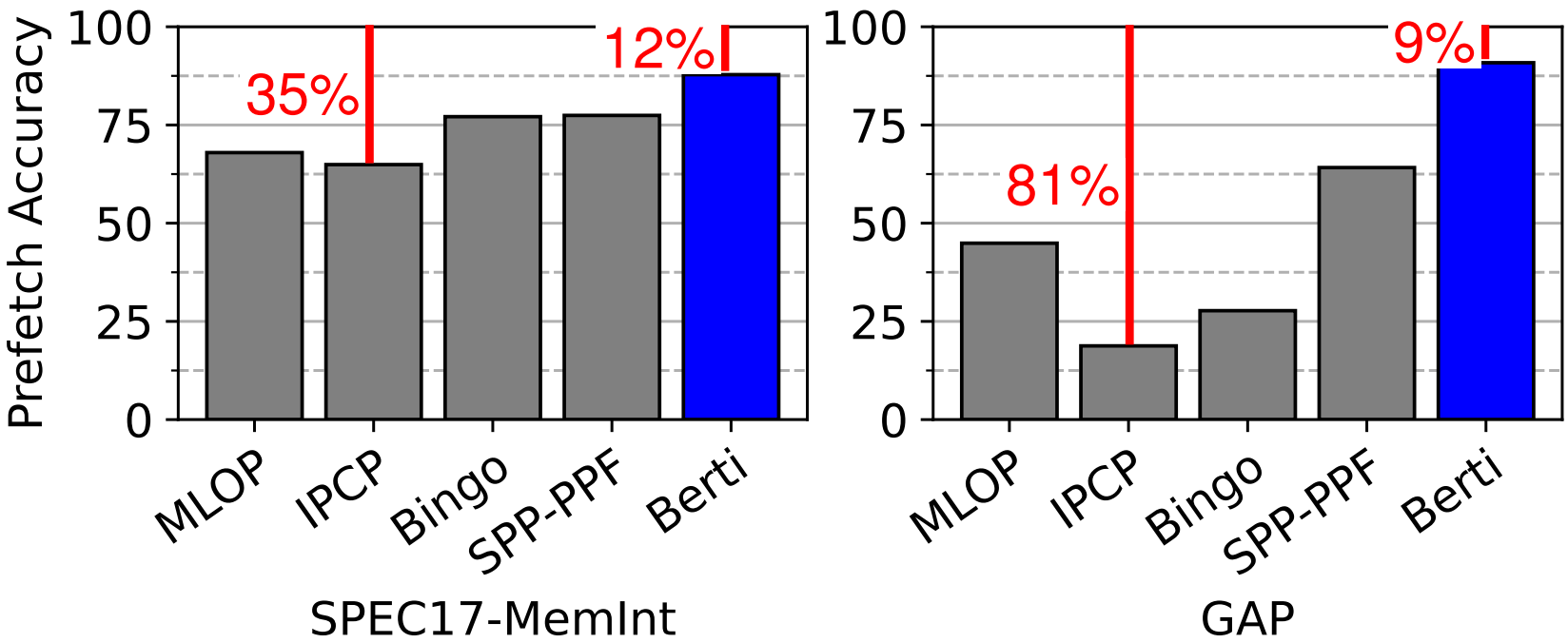
IP	Delta	Coverage	Destination
A	+10	2/2 (100%)	L1D
A	+13	1/2 (50%)	L2

Coverage

> 65% $\rightarrow$ *L1D*

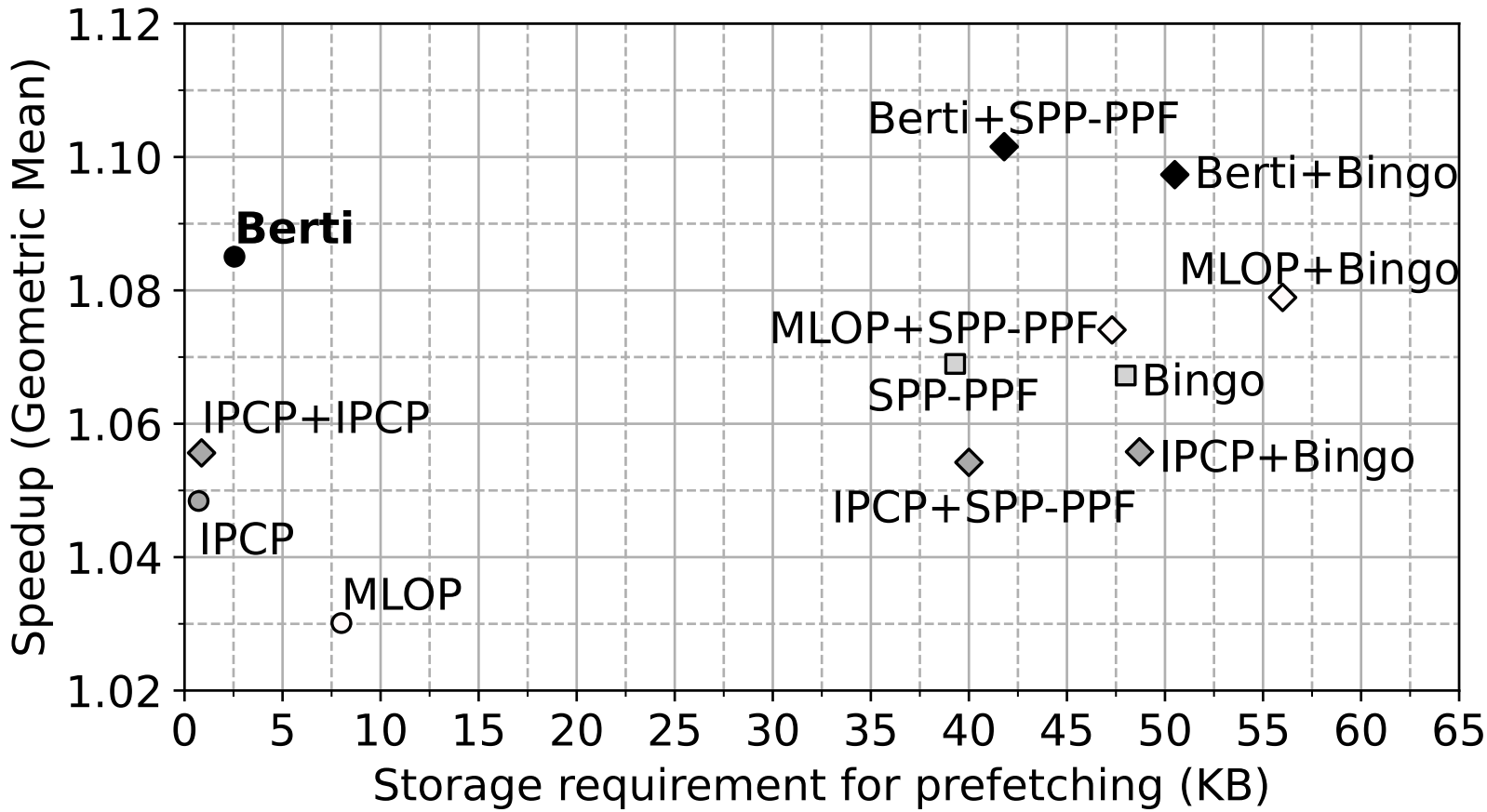
> 35% $\rightarrow$ *L2*

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER



Improved accuracy reduces BW overhead and L1D pollution

BERTI: OVERALL PERFORMANCE



IS BERTI RESTRICTED TO L1D PREFETCHING?

Alberto Ros
BLUE: A Timely, IP-based Data Prefetcher
[1st ML-Based Data Prefetching Competition 2021]

- A prefetcher targeting the last level cache (LLC)
- Combines three techniques (B L ∪ E):
  - BERTI → For spatial prefetching (version adapted from the 3rd DPC)
  - LINNEA (Link next address) → To overcome discontinuity of physical pages
  - ENTANGLING → For temporal prefetching (adapted version)
- Winner of the champsionship!

AND WHAT ABOUT SECURITY?

Sumon Nath, A. Navarro-Torres, Alberto Ros, Biswa Panda
Secure Prefetching for Secure Cache Systems
[MICRO'24]

- A secure version of the Berti prefetcher
  - The key is that it is trained and triggered on commit
  - And can easily adjust timeliness
- But it also avoids wrong-path noise!

Effective Context-Sensitive Memory Dependence Prediction

(Addressing bad speculation)

Sebastian S. Kim Alberto Ros

University of Murcia, Spain

Overview

- Memory dependence prediction (MDP) is a fundamental technique to enable memory level parallelism
  - Loads efficiently execute out-of-order w.r.t previous stores
  - That efficiency depends on the accuracy of the MDP

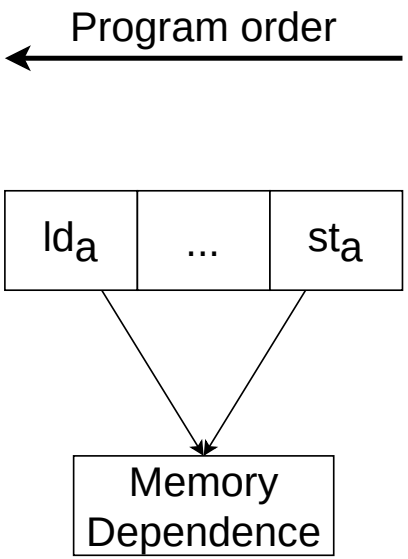
Overview

- Memory dependence prediction (MDP) is a fundamental technique to enable memory level parallelism
  - Loads efficiently execute out-of-order w.r.t previous stores
  - That efficiency depends on the accuracy of the MDP

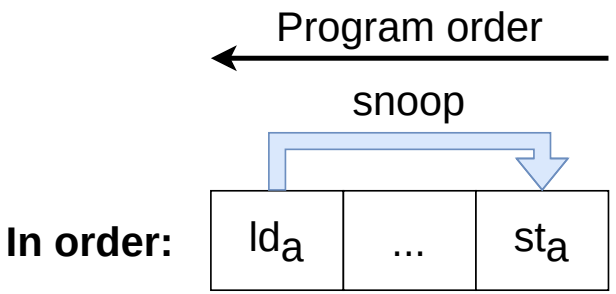
- A novel MDP: PHAST (**P**at**H**-**A**ware **ST**ore-distance)

  $\Rightarrow$ Accurate: Reduces mispredictions by 62.5%
  - Precisely identifies a single dependent store
  - Learns the *proper* context information for each conflict

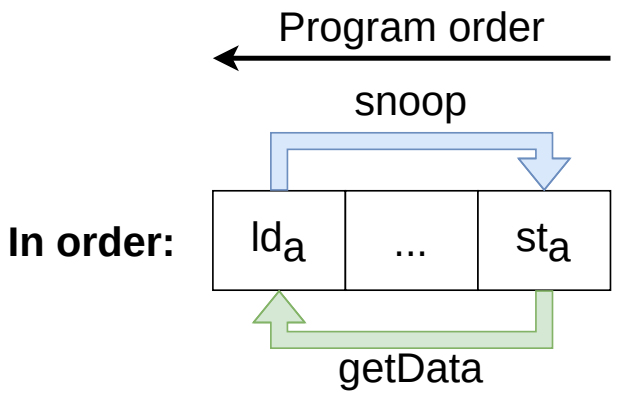
MDP: CRASH COURSE



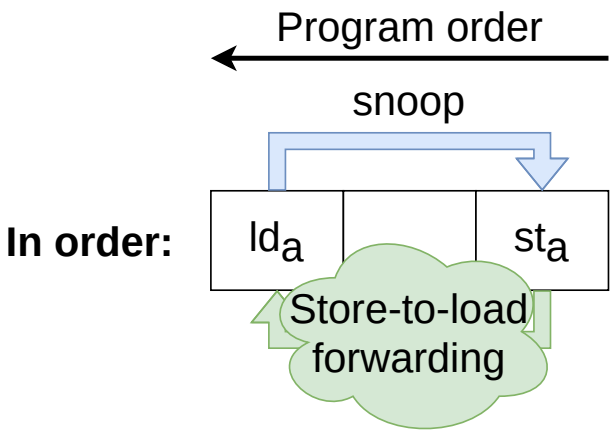
MDP: CRASH COURSE



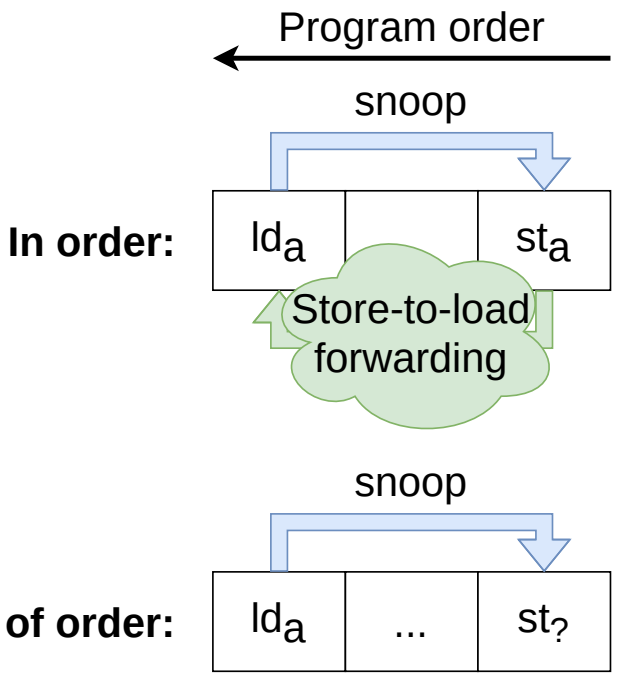
MDP: CRASH COURSE



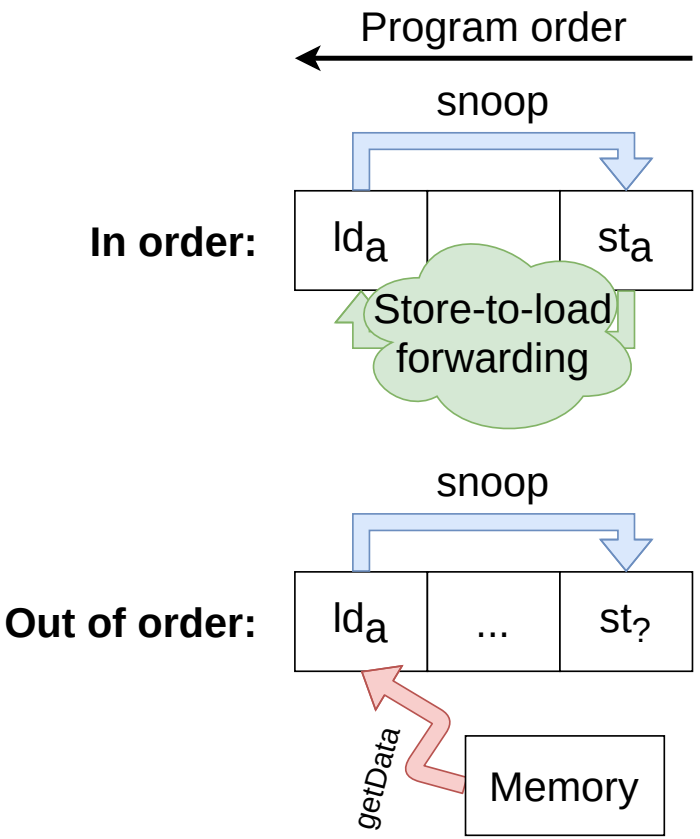
MDP: CRASH COURSE



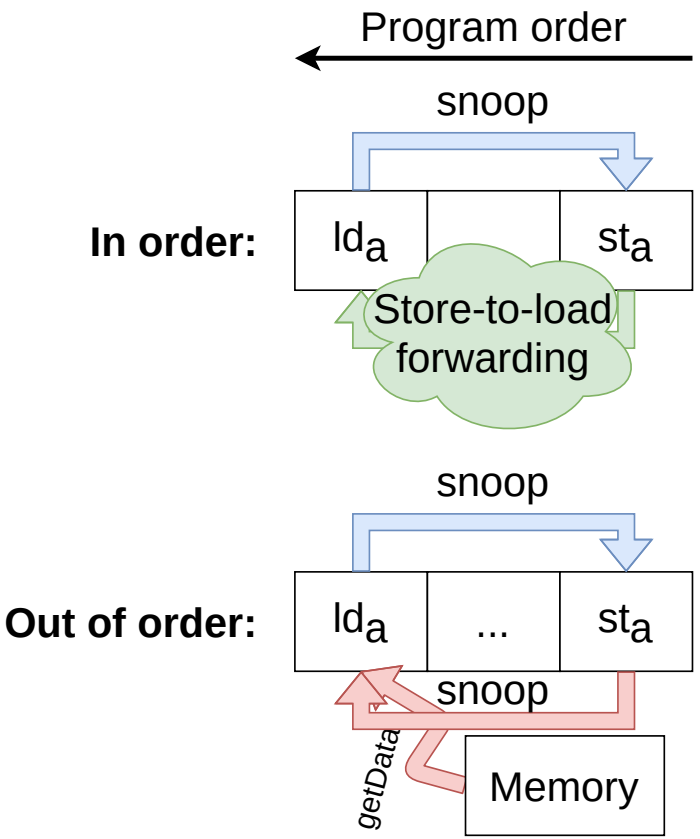
MDP: CRASH COURSE



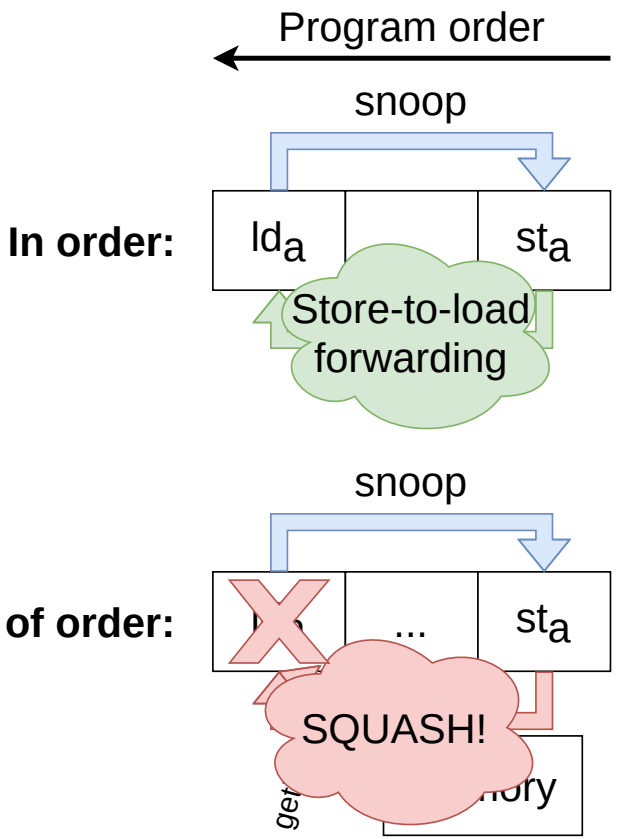
MDP: CRASH COURSE



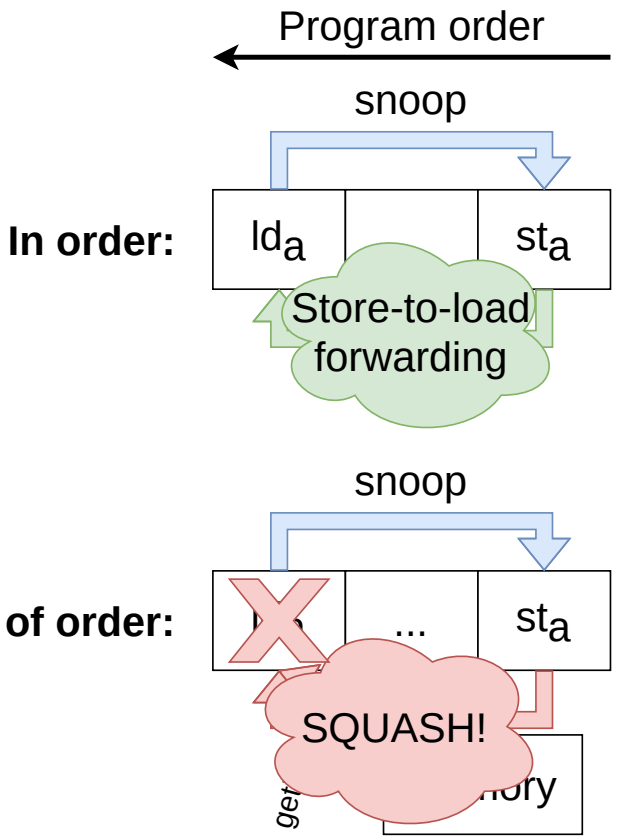
MDP: CRASH COURSE



MDP: Crash course



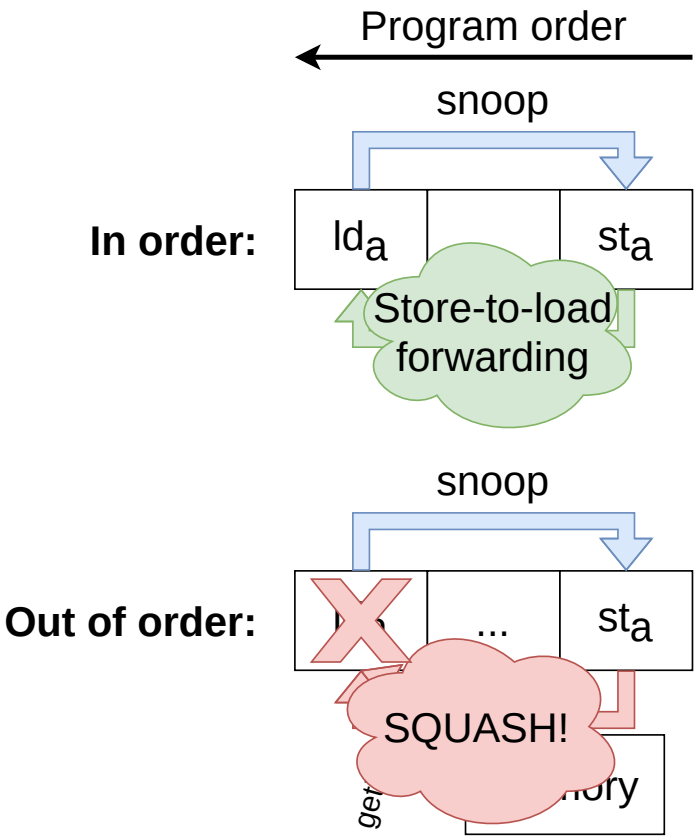
MDP: CRASH COURSE



THE PREDICTION

Is they any previous dependent store?

MDP: CRASH COURSE

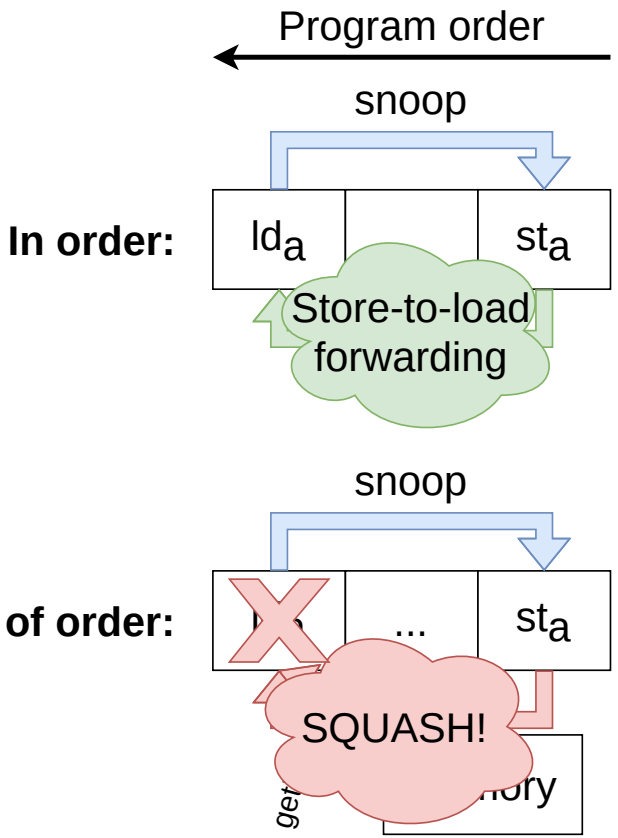


THE PREDICTION

Is they any previous dependent store?

- NO: Execute the load
- YES: Stall (wait) until it executes

MDP: CRASH COURSE



THE PREDICTION

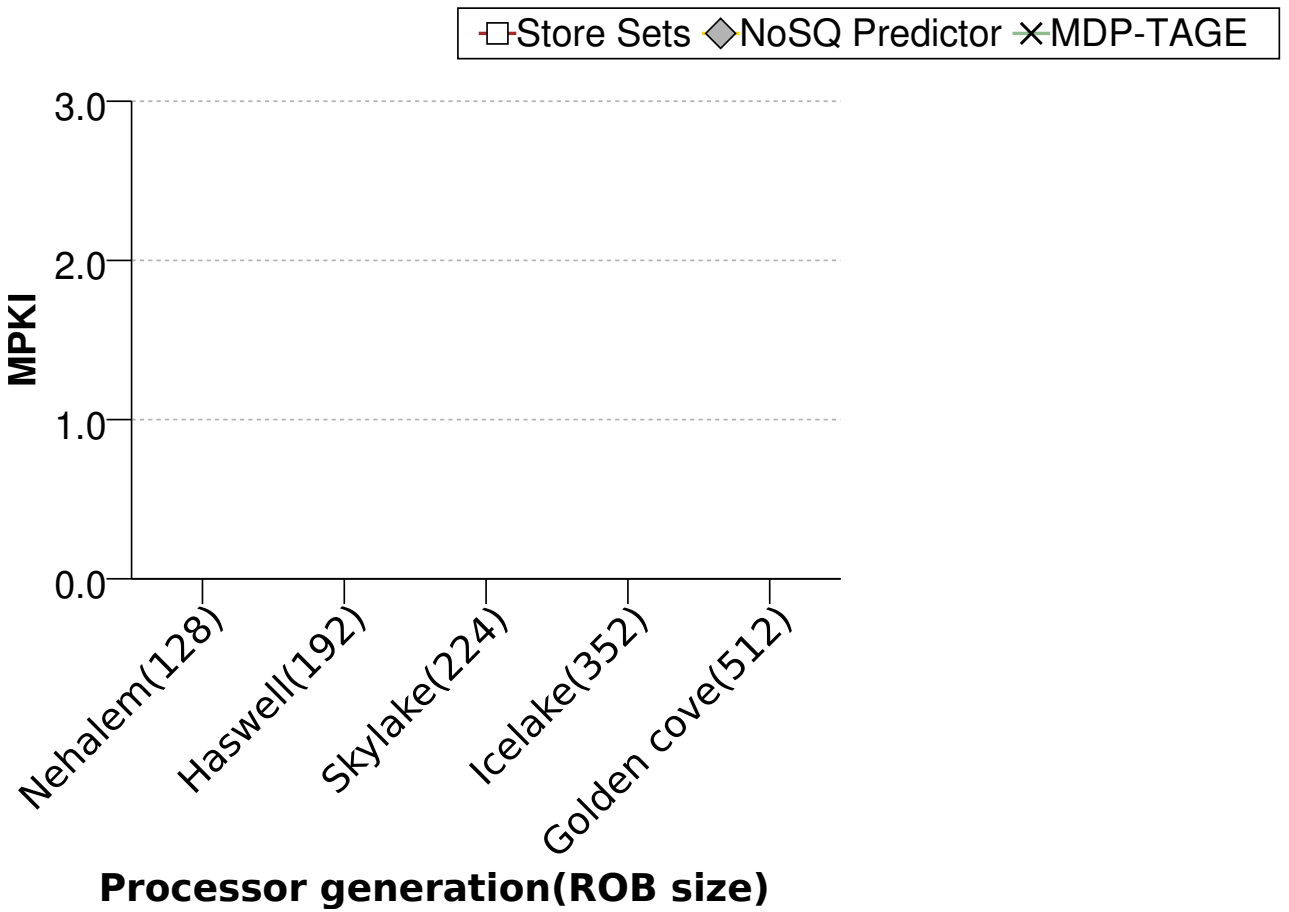
Is they any previous dependent store?

- NO: Execute the load
- YES: Stall (wait) until it executes

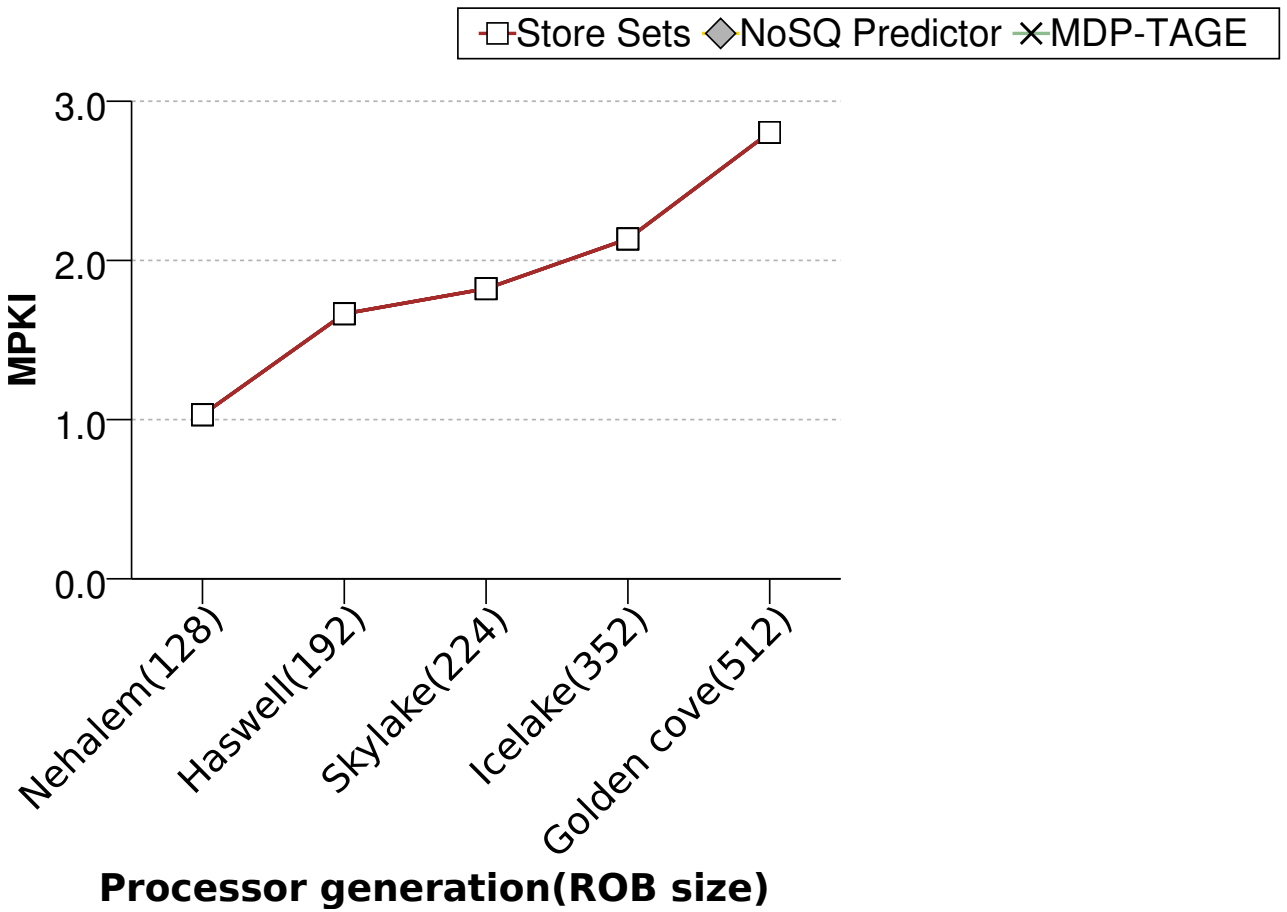
THE MISPREDICTION

- Execute, but conflict ☹
- Stall, but no dependence 😐

MDP, WAS NOT SOLVED IN THE 90'S?

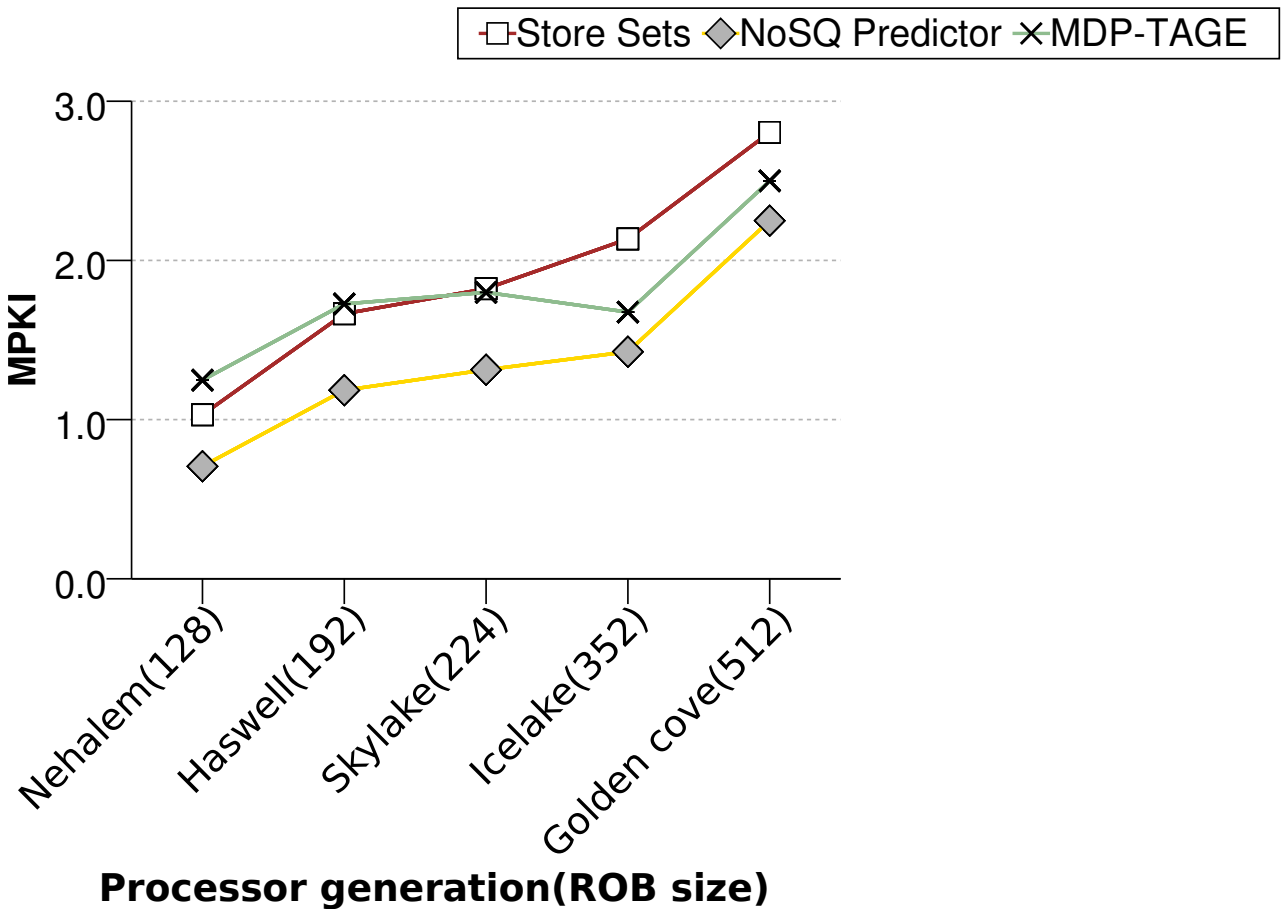


MDP, WAS NOT SOLVED IN THE 90'S?



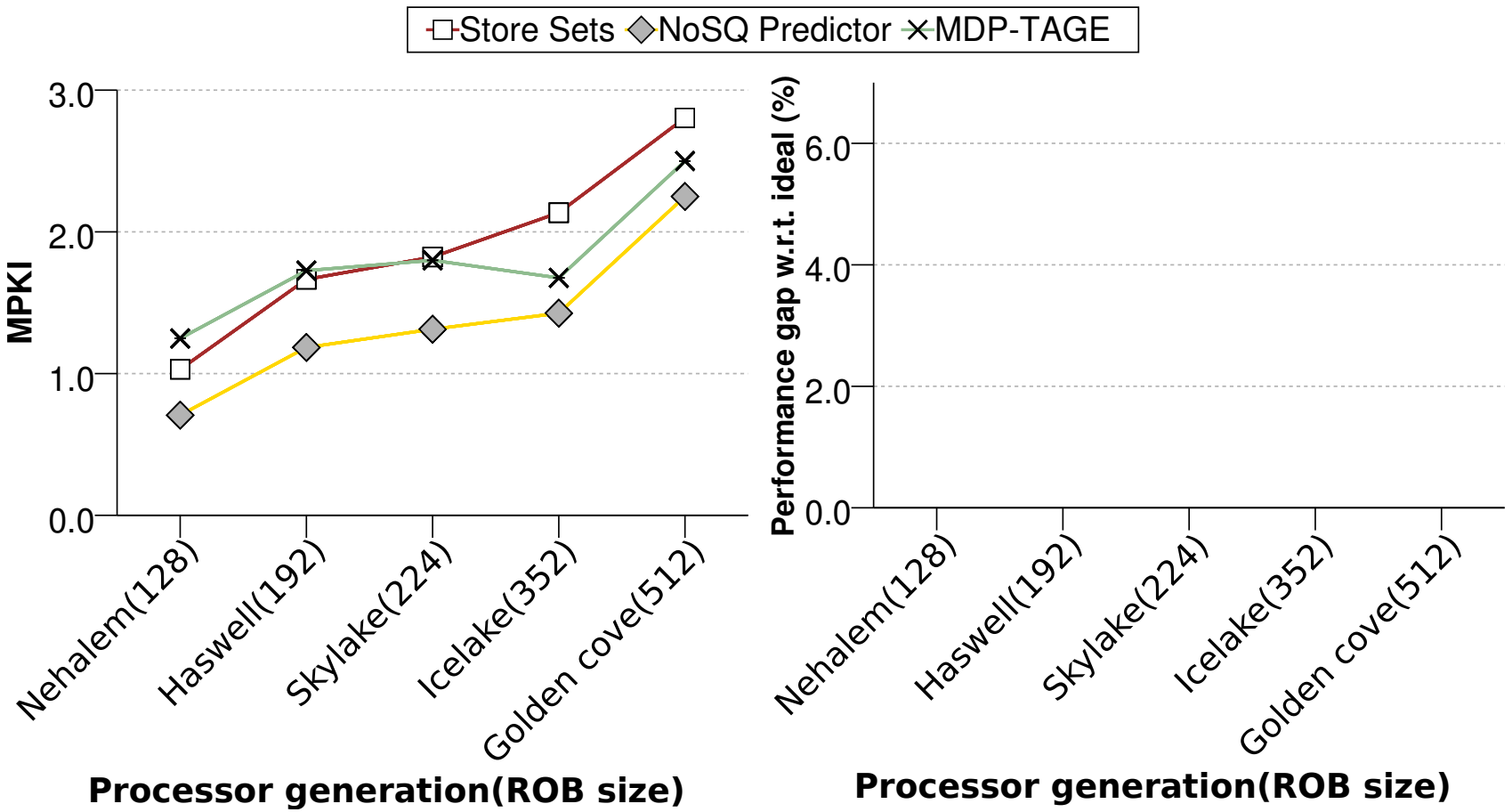
Memory dependence mispredictions increase with processor aggressiveness

MDP, WAS NOT SOLVED IN THE 90'S?



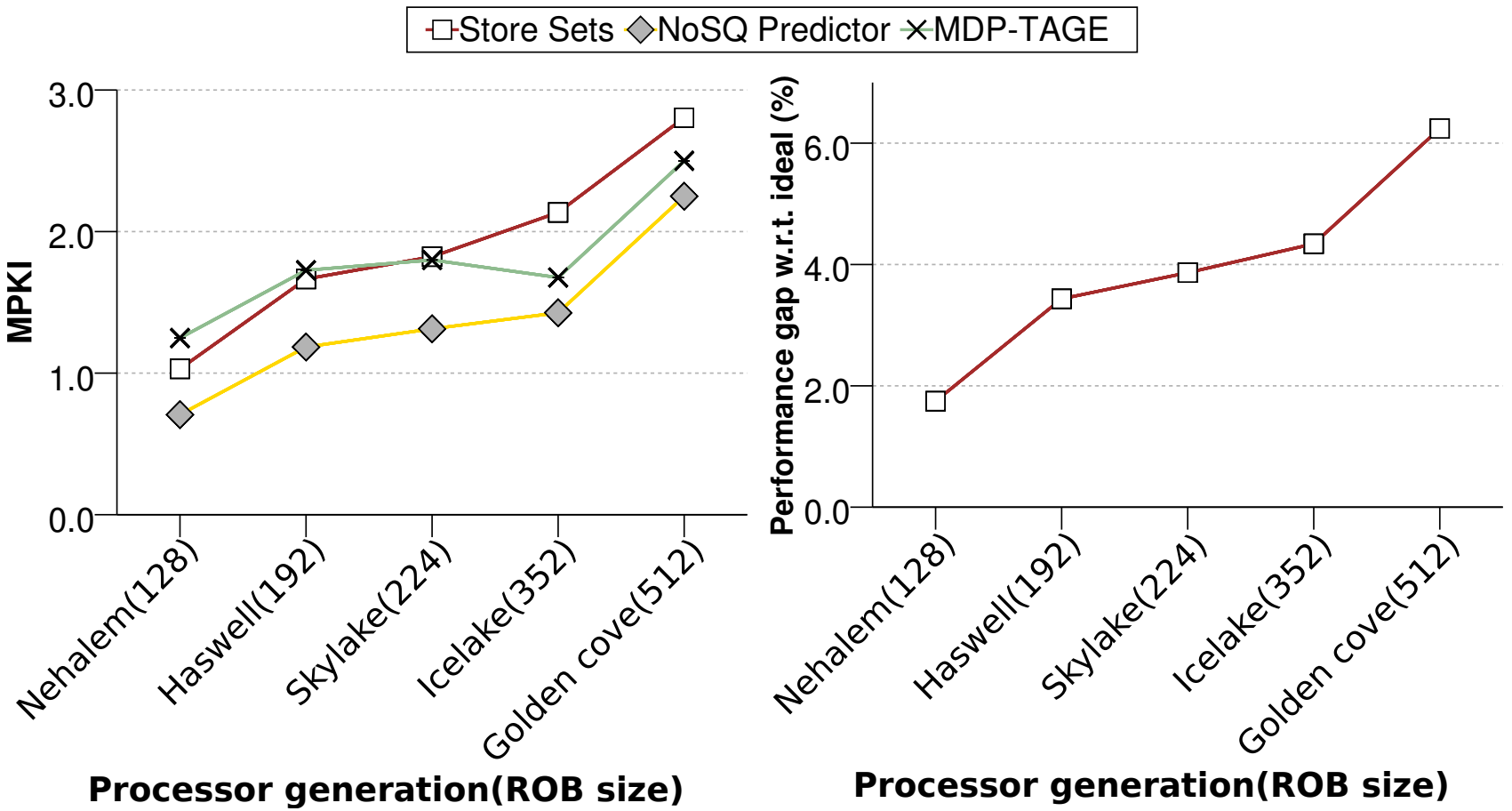
Memory dependence mispredictions increase with processor aggressiveness

MDP, WAS NOT SOLVED IN THE 90'S?



Memory dependence mispredictions increase with processor aggressiveness

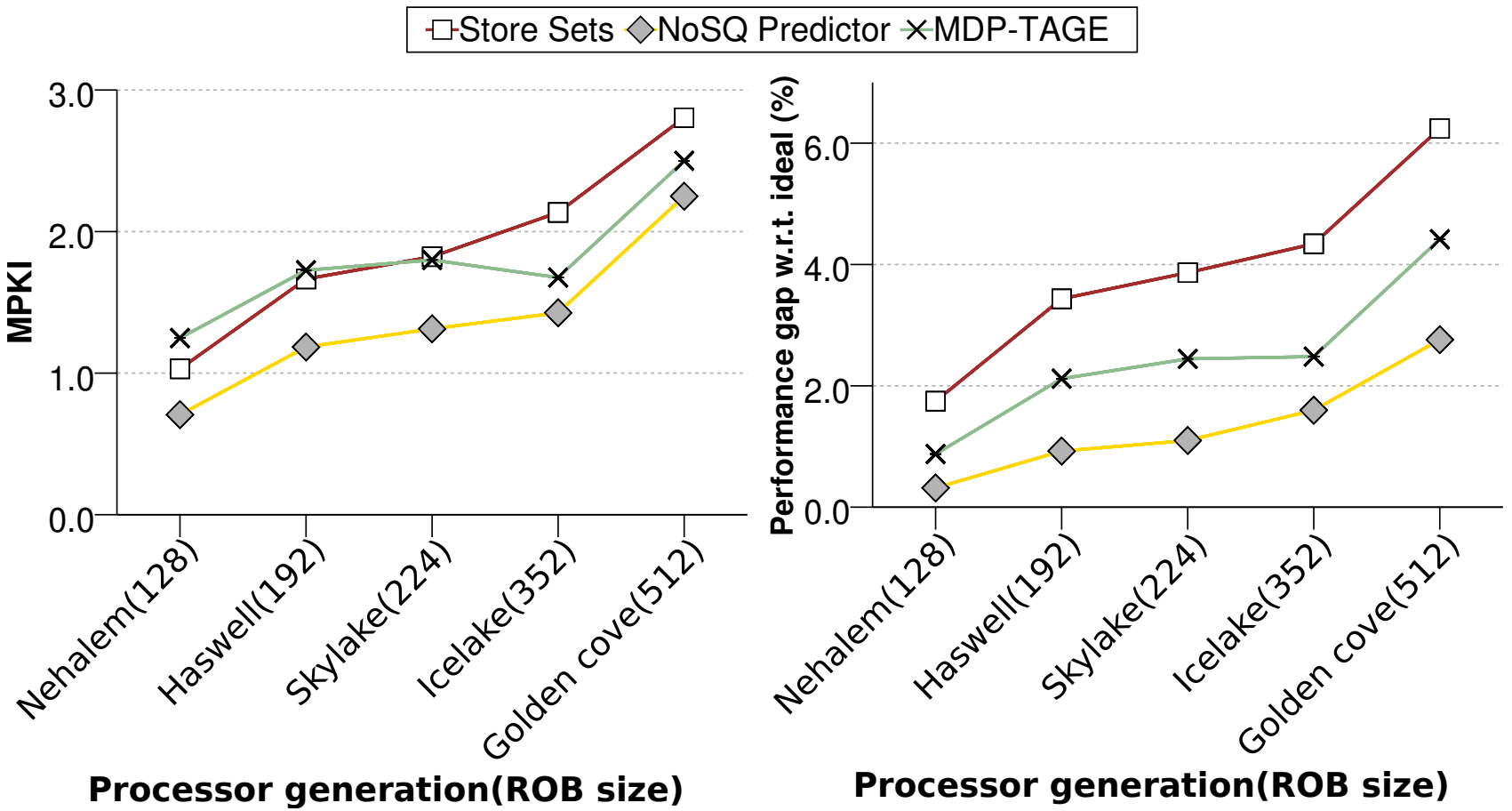
MDP, WAS NOT SOLVED IN THE 90'S?



Memory dependence mispredictions increase with processor aggressiveness

Squashes impact performance more

MDP, WAS NOT SOLVED IN THE 90'S?



Memory dependence mispredictions increase with processor aggressiveness

Squashes impact performance more

PHAST: THE TWO PILLARS

1. Precisely identifies a single dependent store
2. Learns the *proper* context information for each conflict

PHAST: IDENTIFYING A SINGLE STORE PRECISELY

- Loads are quite frequently dependent on a single store

PHAST: IDENTIFYING A SINGLE STORE PRECISELY

- Loads are quite frequently dependent on a single store

1. Exceptions (0.04% of total loads):
   - E.g., narrow stores followed by a wide load

   $\Rightarrow$ 70% of those situation stores execute in order, so identifying the younger store is enough

PHAST: IDENTIFYING A SINGLE STORE PRECISELY

- Loads are quite frequently dependent on a single store

1. Exceptions (0.04% of total loads):
   - E.g., narrow stores followed by a wide load

   $\Rightarrow$ 70% of those situation stores execute in order, so identifying the younger store is enough
2. Several stores targeting the same address as the load
   - Only the one closer to the load matters

PHAST: IDENTIFYING A SINGLE STORE PRECISELY

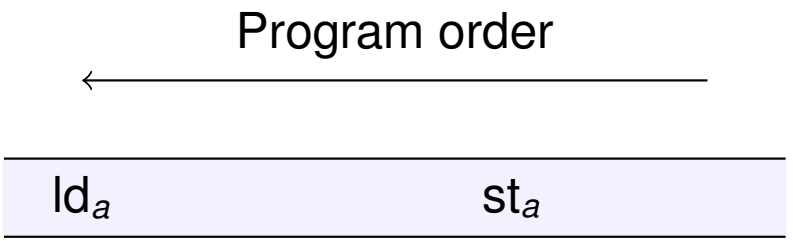
- Loads are quite frequently dependent on a single store

1. Exceptions (0.04% of total loads):
   - E.g., narrow stores followed by a wide load

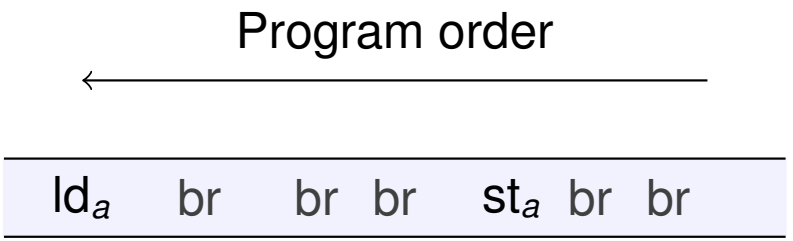
   $\Rightarrow$ 70% of those situation stores execute in order, so identifying the younger store is enough
2. Several stores targeting the same address as the load
   - Only the one closer to the load matters

SOLUTION Track precisely a single dependent store distance

PHAST: LEARNING THE PROPER CONTEXT



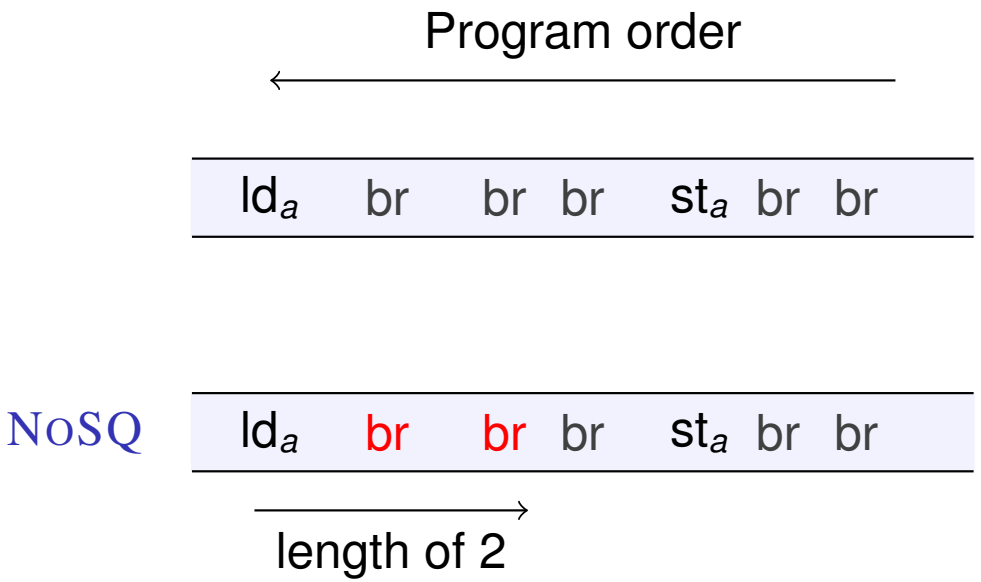
PHAST: LEARNING THE PROPER CONTEXT



Use load's branch history

But which history length?

PHAST: LEARNING THE PROPER CONTEXT

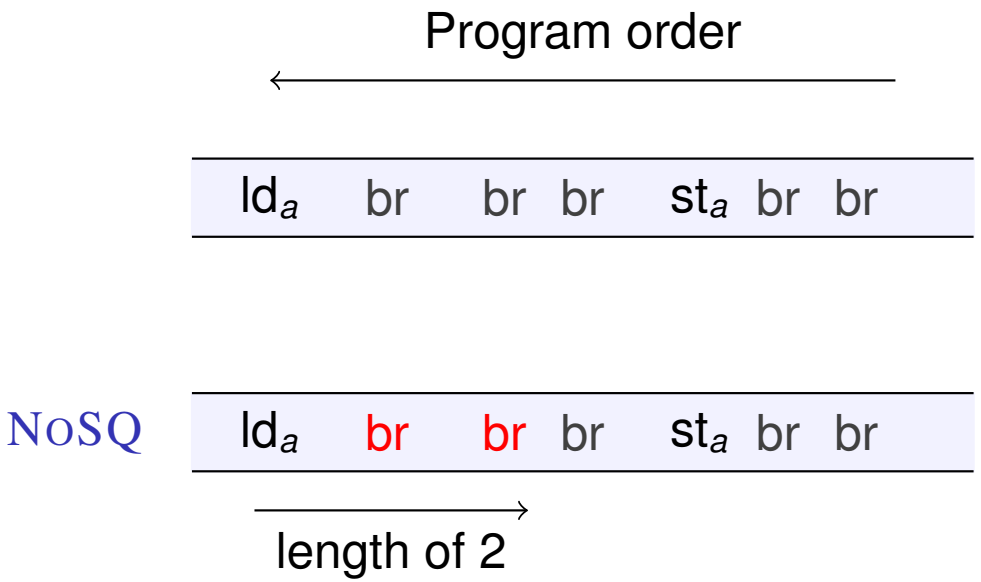


Use load's branch history

But which history length?

Predetermined history length

PHAST: LEARNING THE PROPER CONTEXT



Use load's branch history

But which history length?

Predetermined history length

Too short? Mispredictions

Too long? Unnecessary paths

PHAST: LEARNING THE PROPER CONTEXT

Use load's branch history

But which history length?

Predetermined history length

Too short? Mispredictions

Too long? Unnecessary paths

MDP-TAGE

Geometrically increasing history lengths

Brute-force method to find the right lenght

PHAST: LEARNING THE PROPER CONTEXT

Use load's branch history

But which history length?

Predetermined history length

Too short? Mispredictions

Too long? Unnecessary paths

MDP-TAGE

Geometrically increasing history lengths

Brute-force method to find the right lenght

- Previous context-sensitive proposals borrow branch-prediction techniques
  - Use predetermined history lengths ⇒ sub-optimal

PHAST: Learning the proper context

- PHAST uses the proper history length for each conflict
  - $\Rightarrow$ The one that defines the path from the store to the load
    - Intuition: If the exact execution path repeats, it is likely that the dependence repeats, too

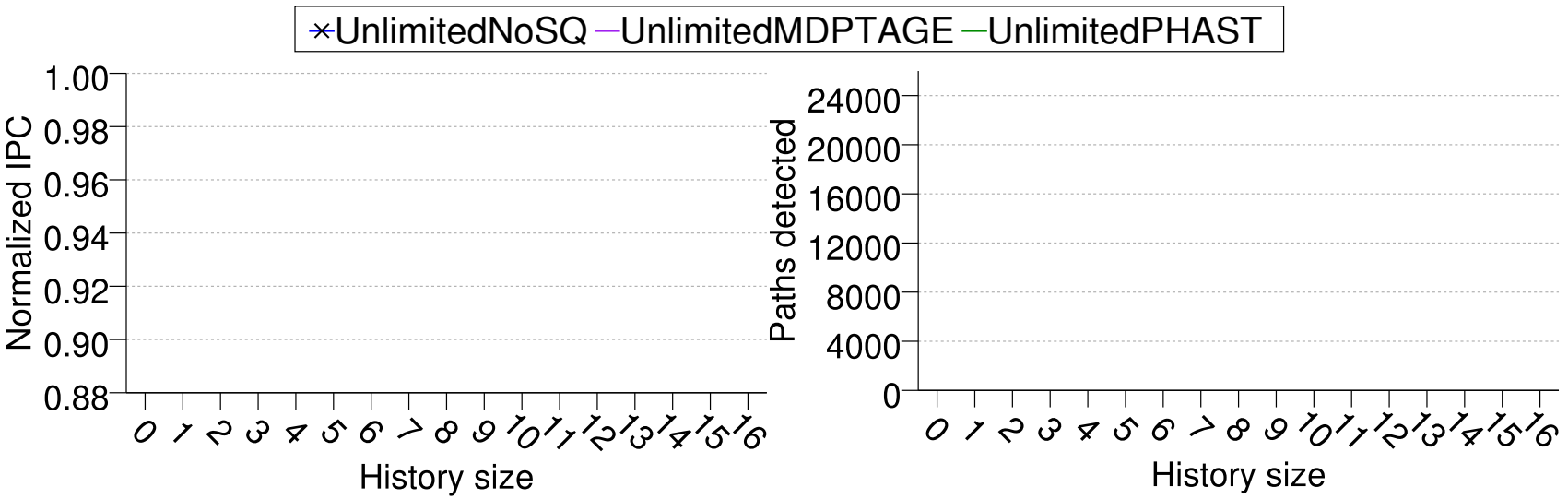
PHAST: Learning the proper context

- PHAST uses the proper history length for each conflict
  - $\Rightarrow$ The one that defines the path from the store to the load
    - Intuition: If the exact execution path repeats, it is likely that the dependence repeats, too

- To identify the path, PHAST only considers divergent branches (conditional and indirect branches):
  - Conditional branches $\rightarrow$ taken/not taken bit
  - Indirect branches $\rightarrow$ target address

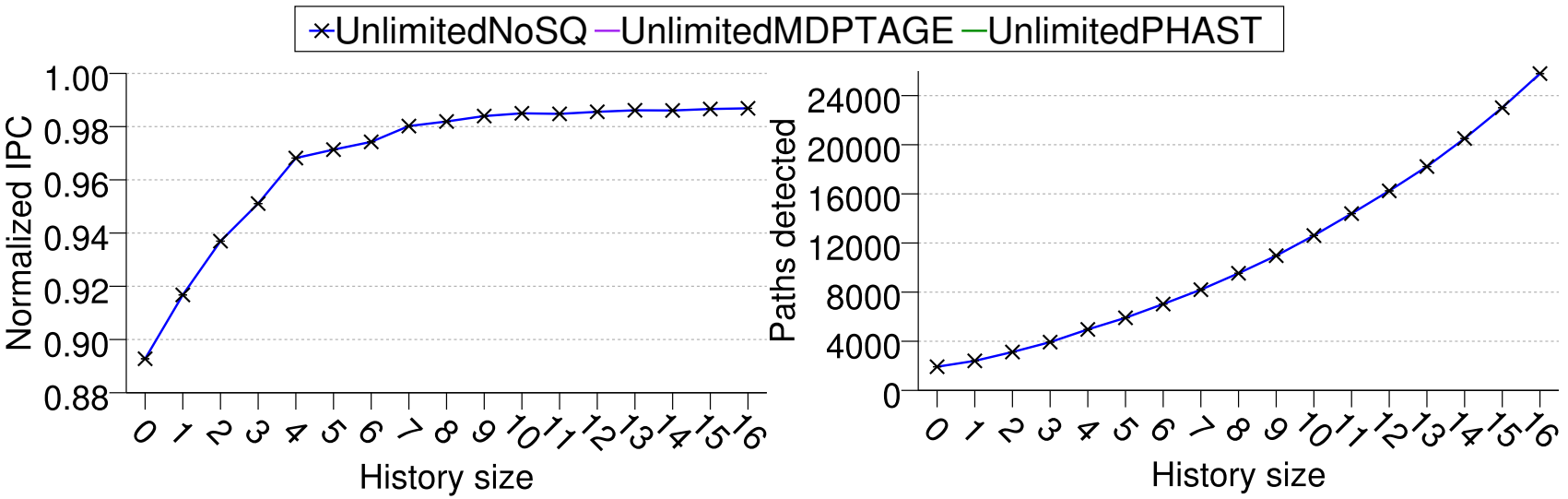
COMPARISON OF IDEAL CONTEXT-SENSITIVE MDPS

- IPC normalized to an ideal predictor!



COMPARISON OF IDEAL CONTEXT-SENSITIVE MDPS

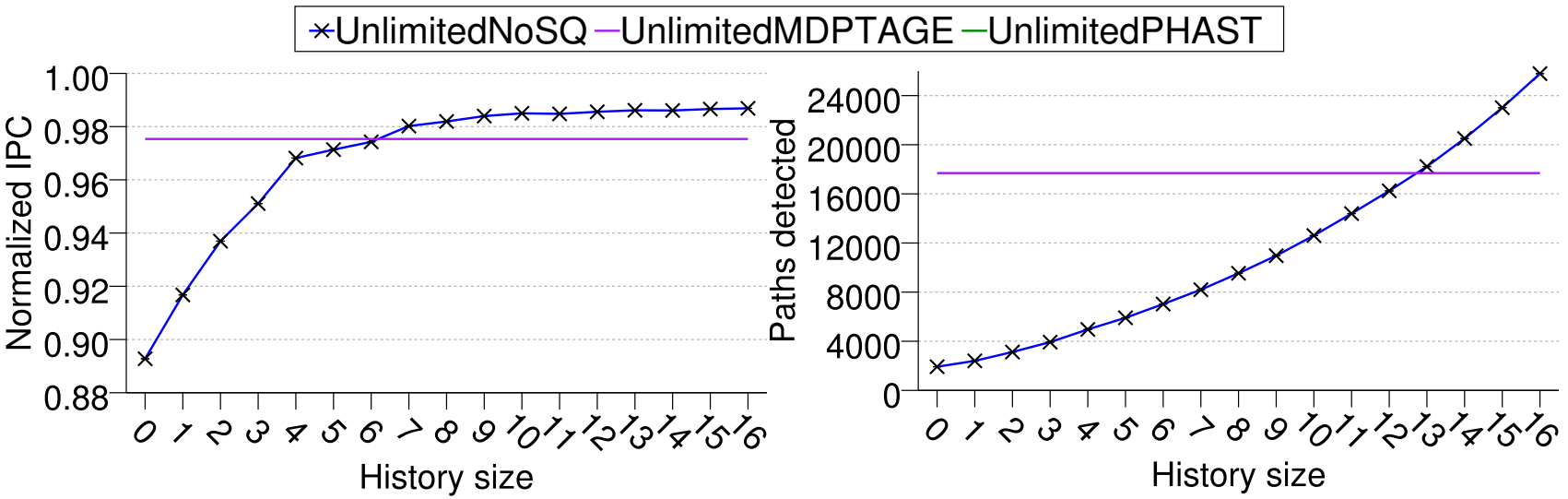
- IPC normalized to an ideal predictor!



- The larger is the history, the more is the accuracy, but the more are the hardware requirements

COMPARISON OF IDEAL CONTEXT-SENSITIVE MDPS

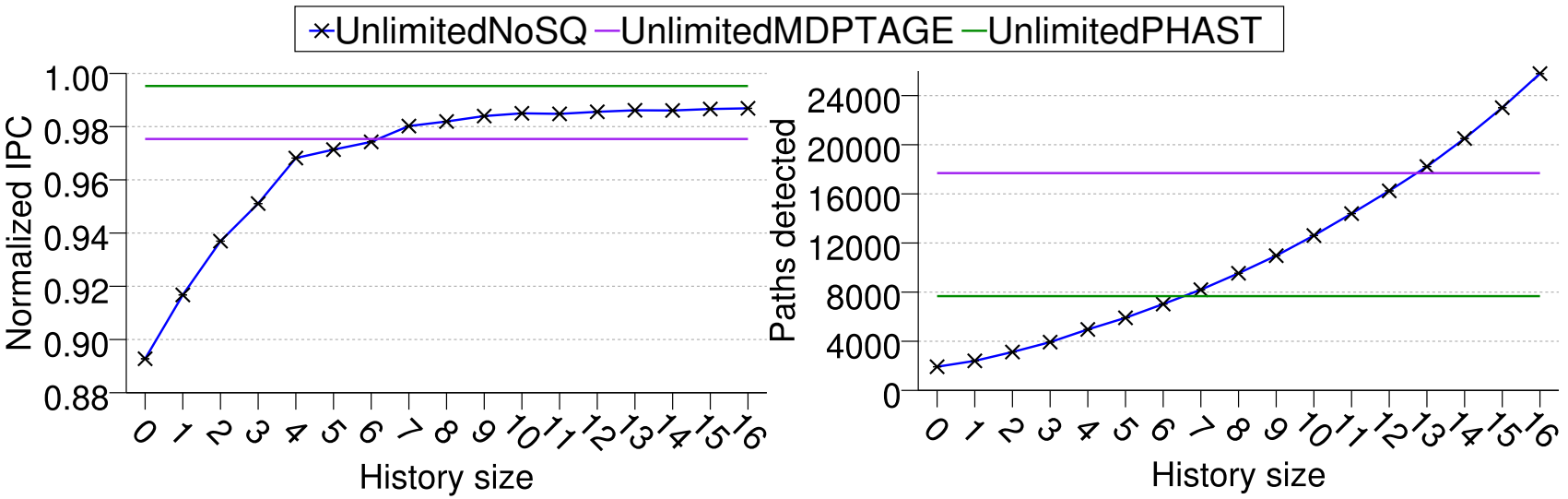
- IPC normalized to an ideal predictor!



- The larger is the history, the more is the accuracy, but the more are the hardware requirements

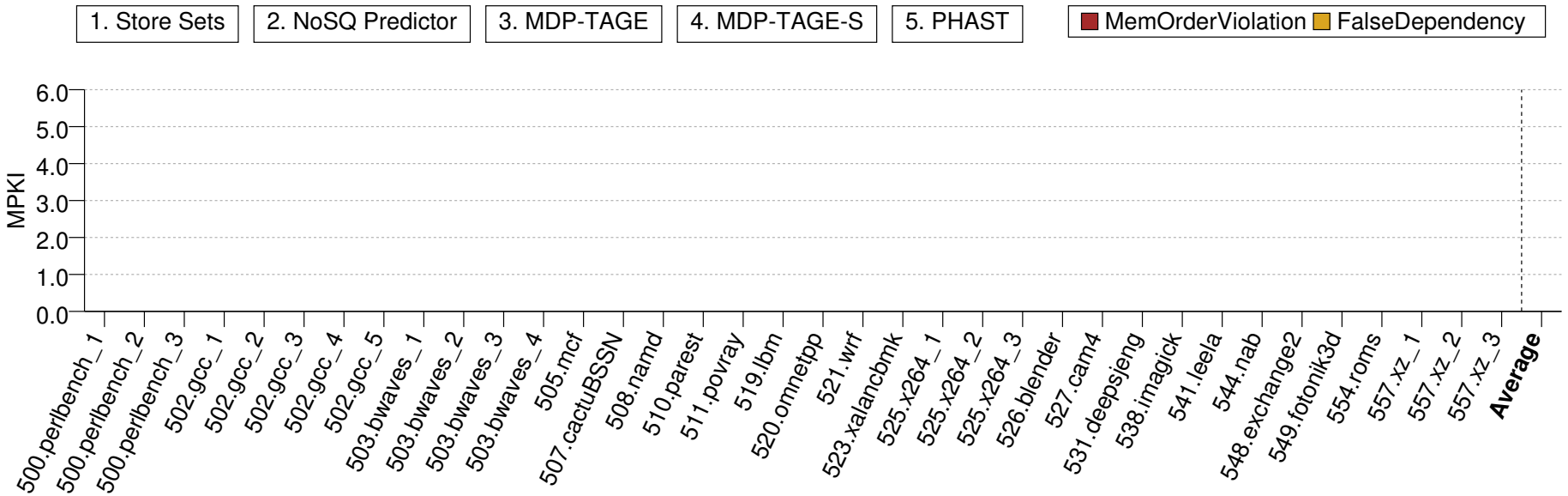
COMPARISON OF IDEAL CONTEXT-SENSITIVE MDPS

- IPC normalized to an ideal predictor!

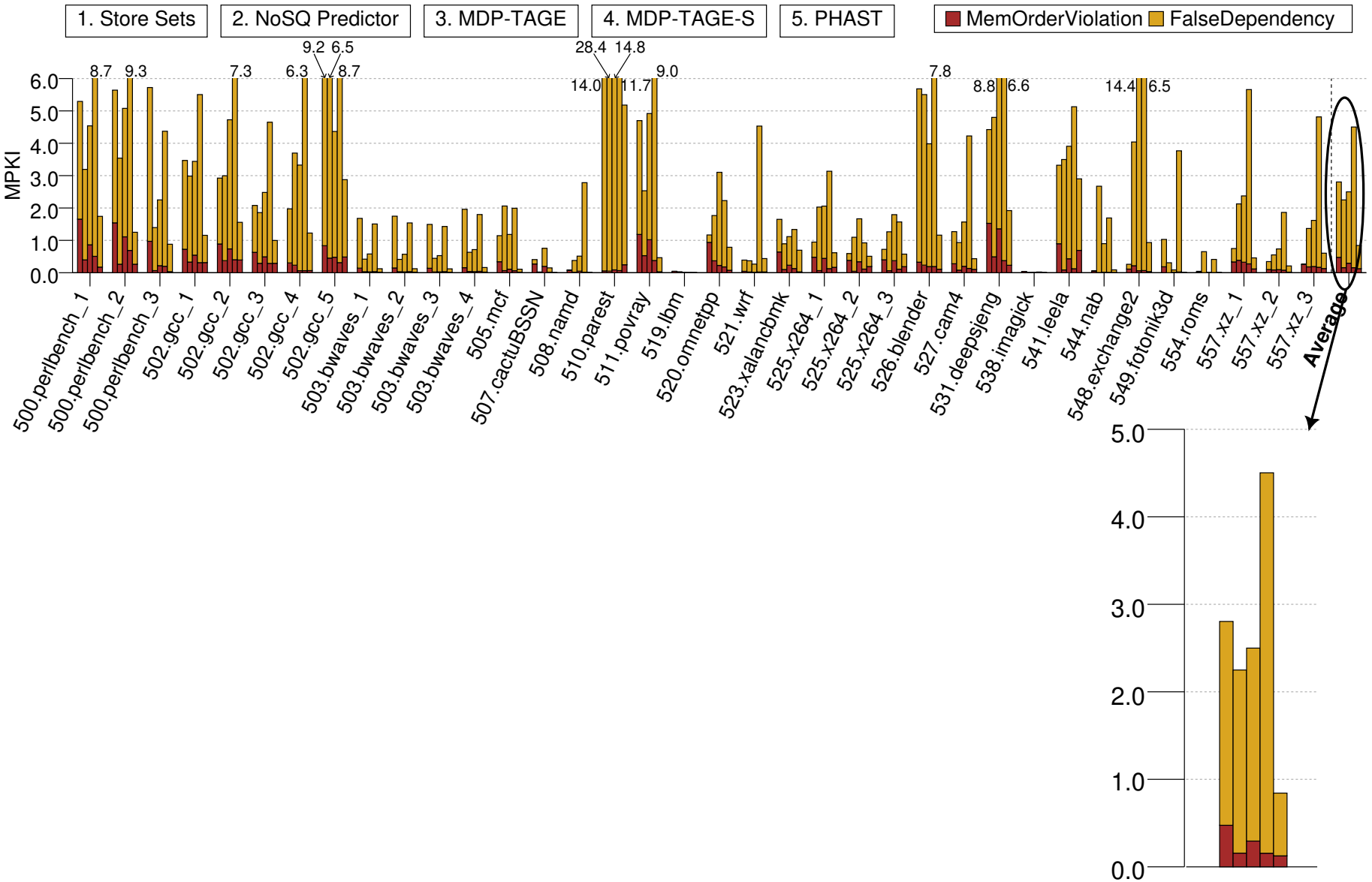


- The larger is the history, the more is the accuracy, but the more are the hardware requirements
- Unlimited PHAST peformance falls 0.5% behind an ideal predictor! $\Rightarrow$ Paths are a good proxy for conflicts

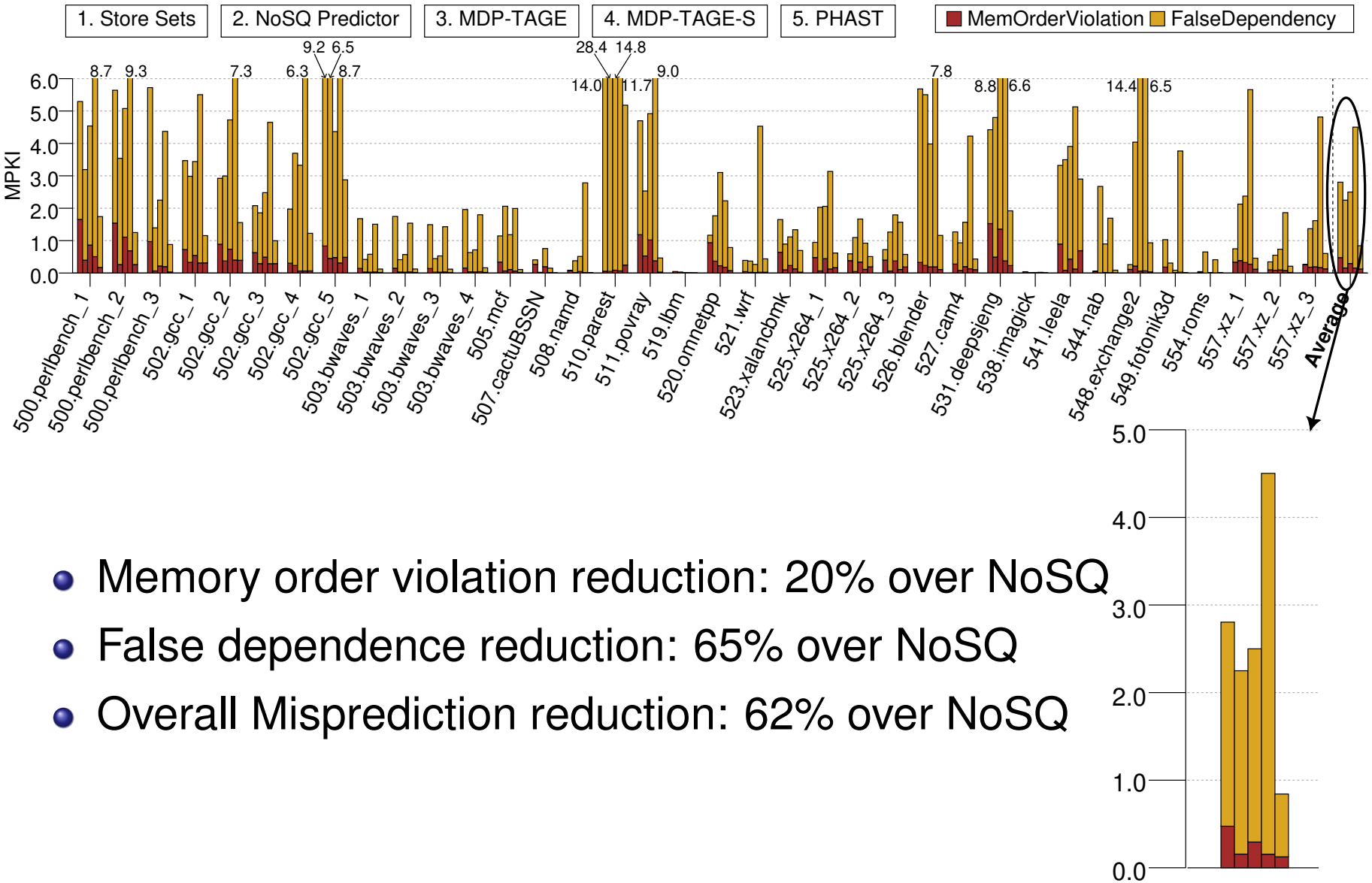
RESULTS: MPKI



RESULTS: MPKI



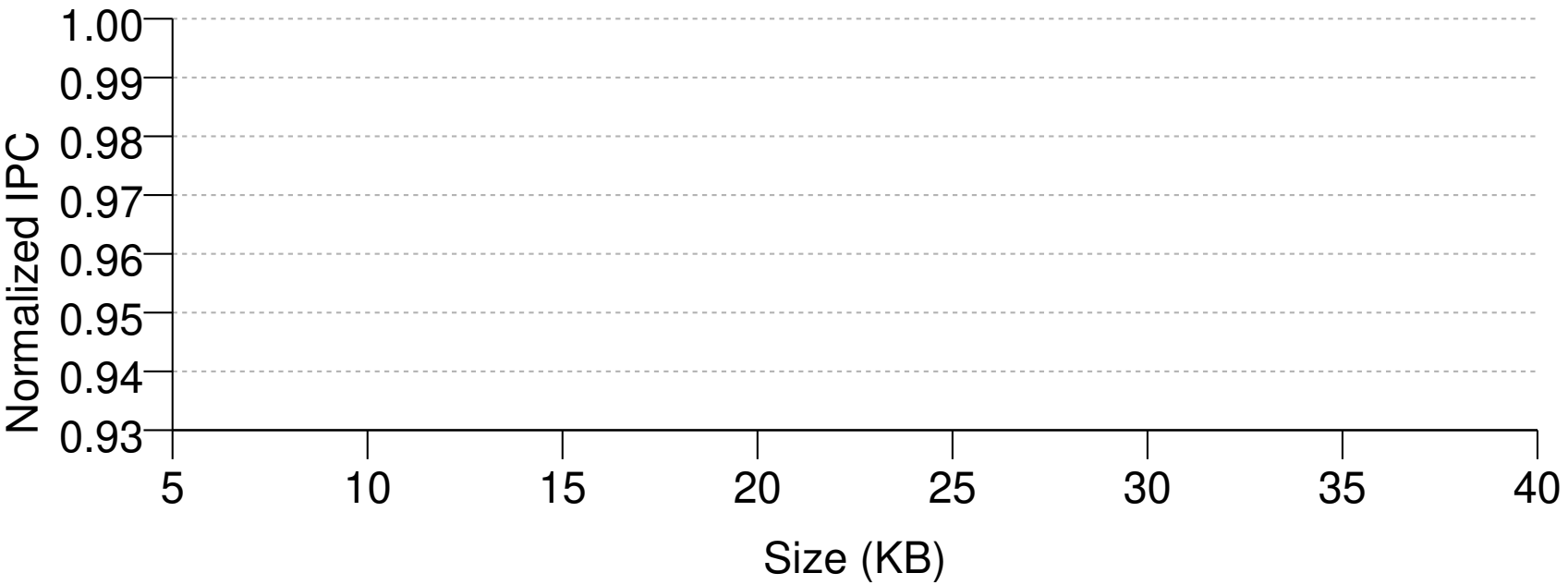
RESULTS: MPKI



- Memory order violation reduction: 20% over NoSQ
- False dependence reduction: 65% over NoSQ
- Overall Misprediction reduction: 62% over NoSQ

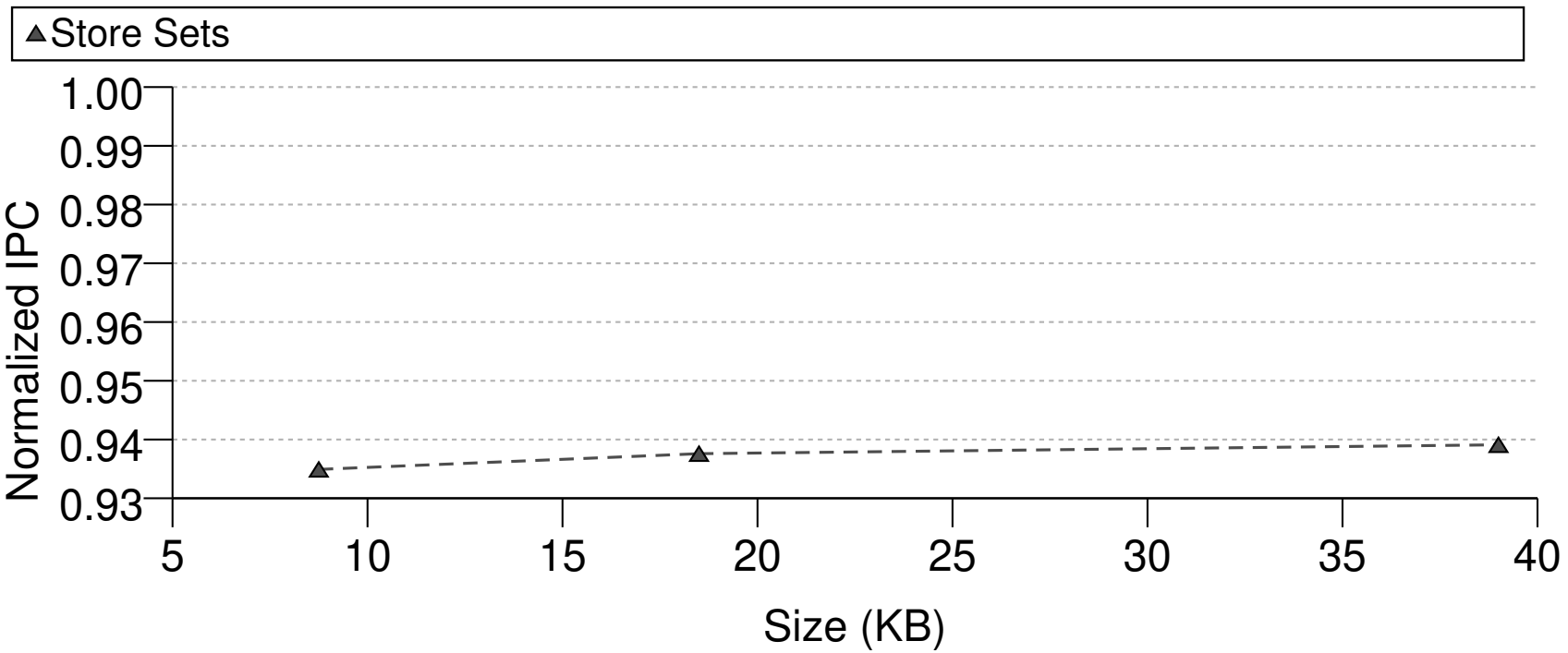
RESULTS: IPC VS SIZE

- IPC normalized to an ideal predictor!



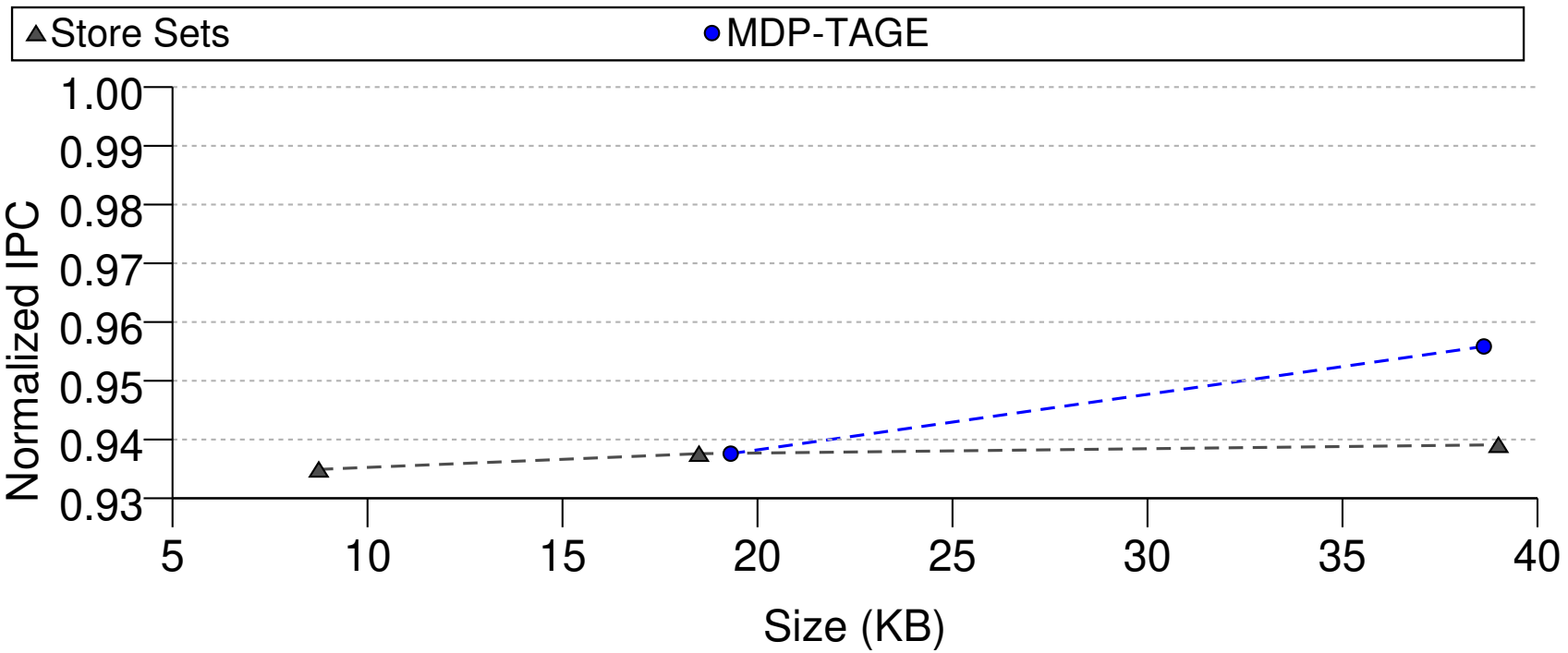
RESULTS: IPC VS SIZE

- IPC normalized to an ideal predictor!



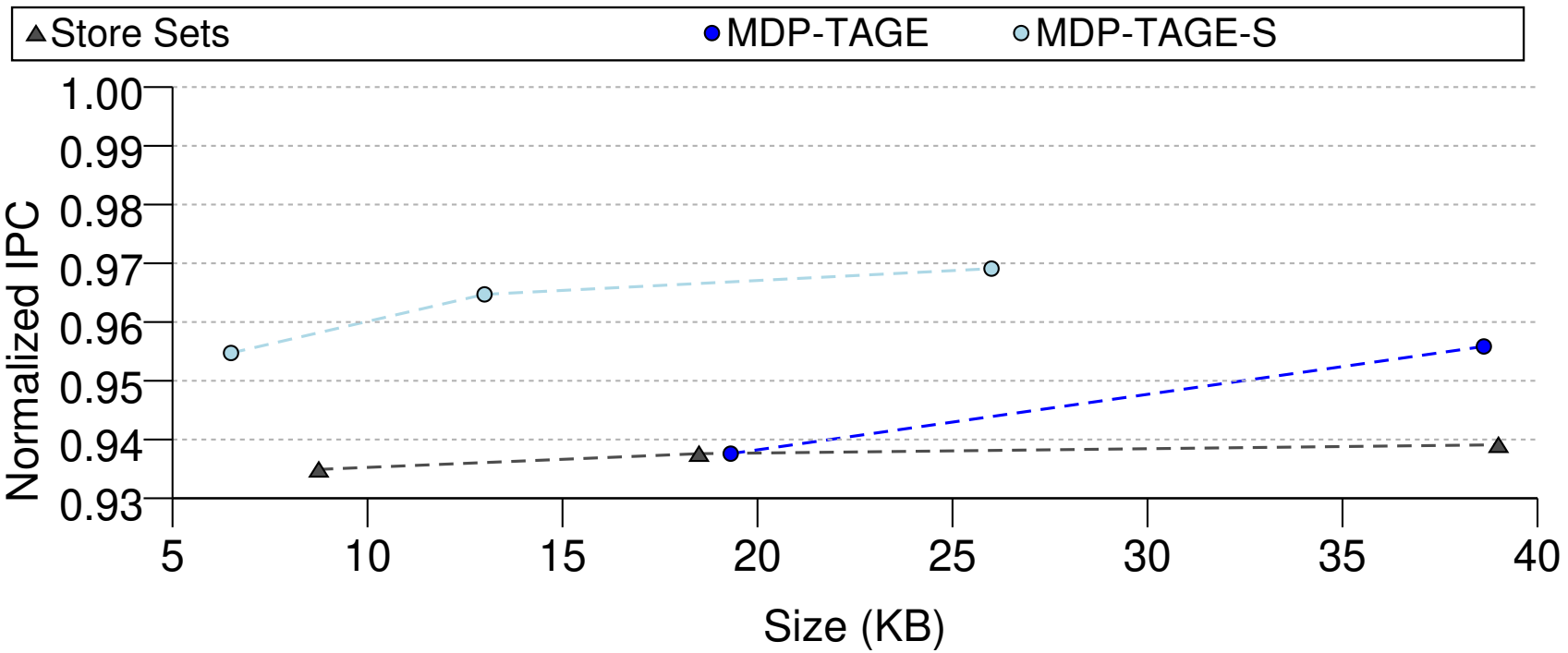
RESULTS: IPC VS SIZE

- IPC normalized to an ideal predictor!



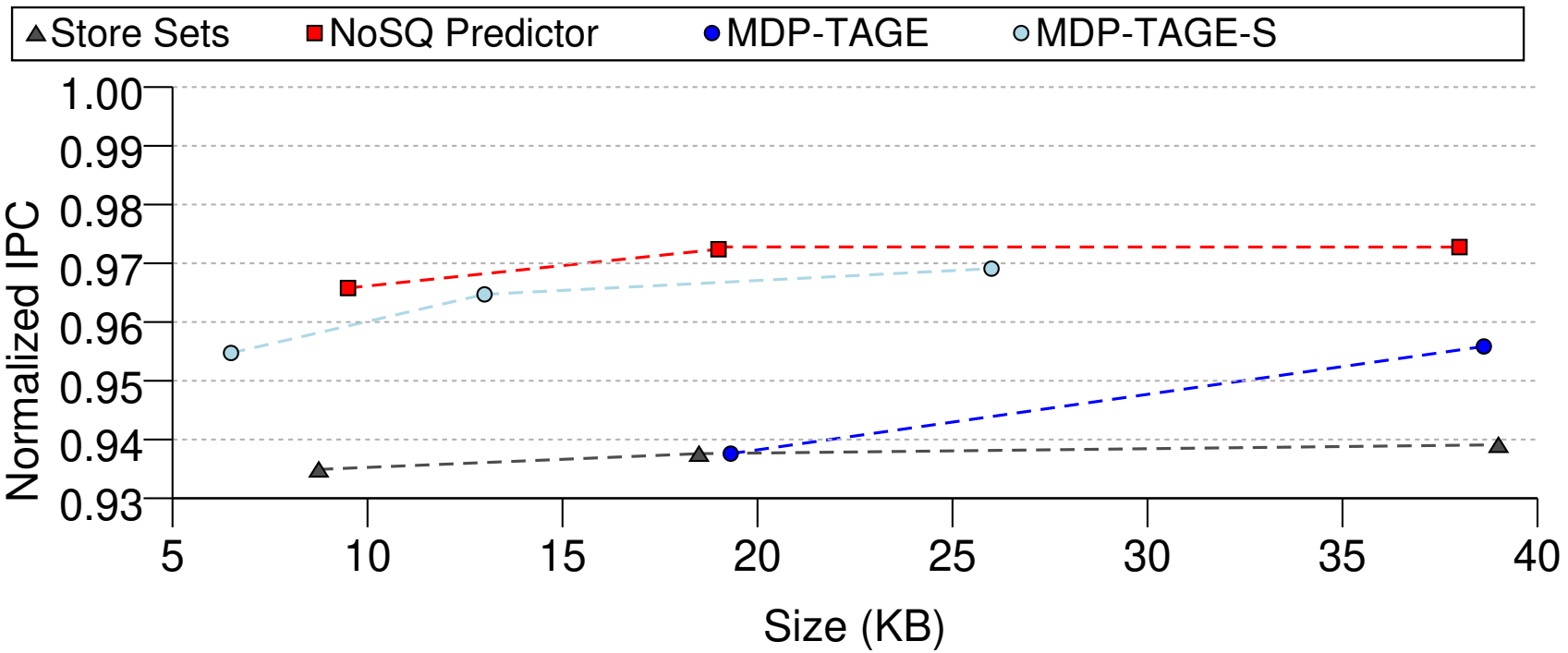
RESULTS: IPC VS SIZE

- IPC normalized to an ideal predictor!



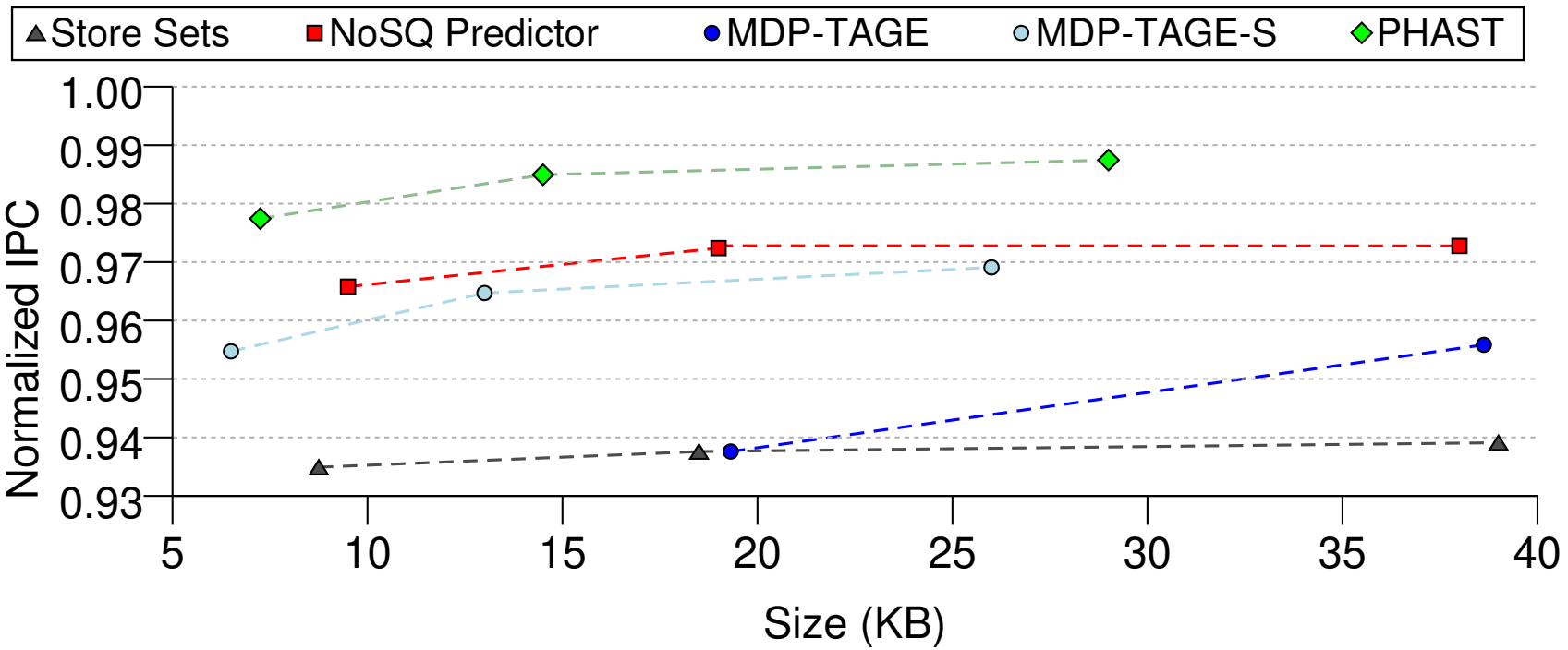
RESULTS: IPC VS SIZE

- IPC normalized to an ideal predictor!



Results: IPC vs Size

- IPC normalized to an ideal predictor!



FINAL REMARKS

$\Rightarrow$ Accurate predictions is key for high performance

1. Prefetching
   - Both intructions and data are amenable to timely prefetching
   - Ideally, it is desirable to prefetch to L1
2. Memory dependence prediction
   - Use just the branch history from the load to the store (+1)

Advanced Prediction Mechanisms for High Performance Computers

Alberto Ros

University of Murcia, Spain

Thank you!

ECHO, ERC Consolidator Grant (No 819134)

BUT WE SHOULD NOT FORGET ABOUT PARALLELISM!

- Eduardo J. Gómez-Hernández, Juan M. Cebrian, Stefanos Kaxiras, Alberto Ros
  Bounding Speculative Execution of Atomic Regions to a Single Retry
  [ASPLOS'24]

- Víctor Nicolas-Conesa, Rubén Titos-Gil, Ricardo Fernández-Pascual, Manuel E. Acacio, Alberto Ros
  Chaining Transactions for Effective Concurrency Management in Hardware Transactional Memory
  [MICRO'24]

- Juan M. Cebrian, Magnus Jarhe, Alberto Ros
  Temporarily Unauthorized Stores: Write First, Ask for Permission Later
  [MICRO'24]

Overview: Instruction prefetcher

- Server and cloud apps getting larger, far from fitting in L1I
  - $\Rightarrow$ stalls processor front-end, performance degradation

Overview: Instruction prefetcher

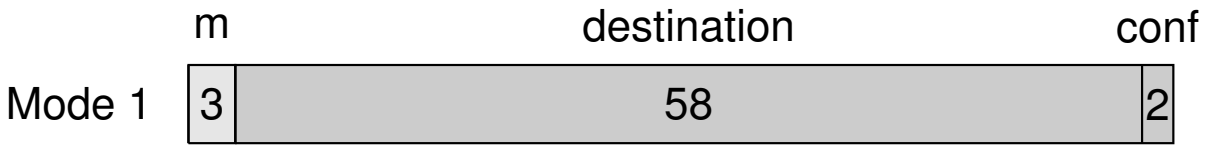
- Server and cloud apps getting larger, far from fitting in L1I
  - $\Rightarrow$ stalls processor front-end, performance degradation
- Prefetching instructions is fundamental for performance
  - Even when a decoupled front-end is implemented

Overview: Instruction prefetcher

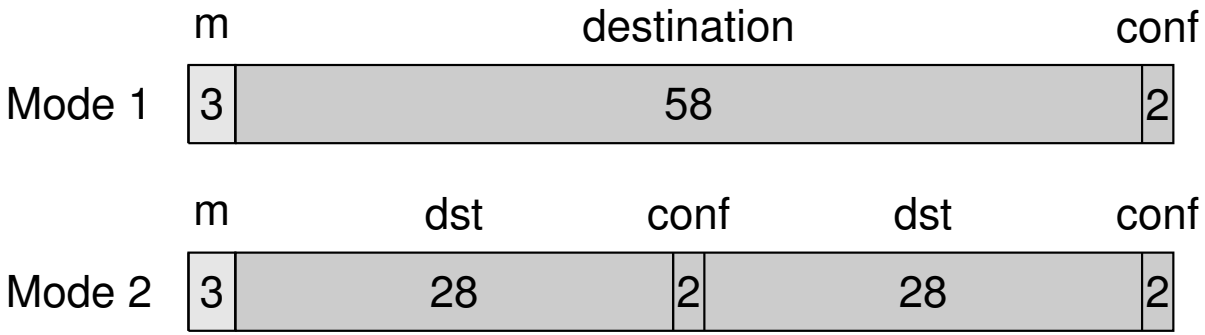
- Server and cloud apps getting larger, far from fitting in L1I
  - $\Rightarrow$ stalls processor front-end, performance degradation
- Prefetching instructions is fundamental for performance
  - Even when a decoupled front-end is implemented
- Our contribution: An Entangling prefetcher
  - Entangling: adaptive correlation based on latency
  - Winner of the 1st Instruction Prefetching Championship
  - A cost-effective prefetcher
  - Prefetcher code is available[1]

[1] https://github.com/alberto-ros/EntanglingInstructionPrefetcher

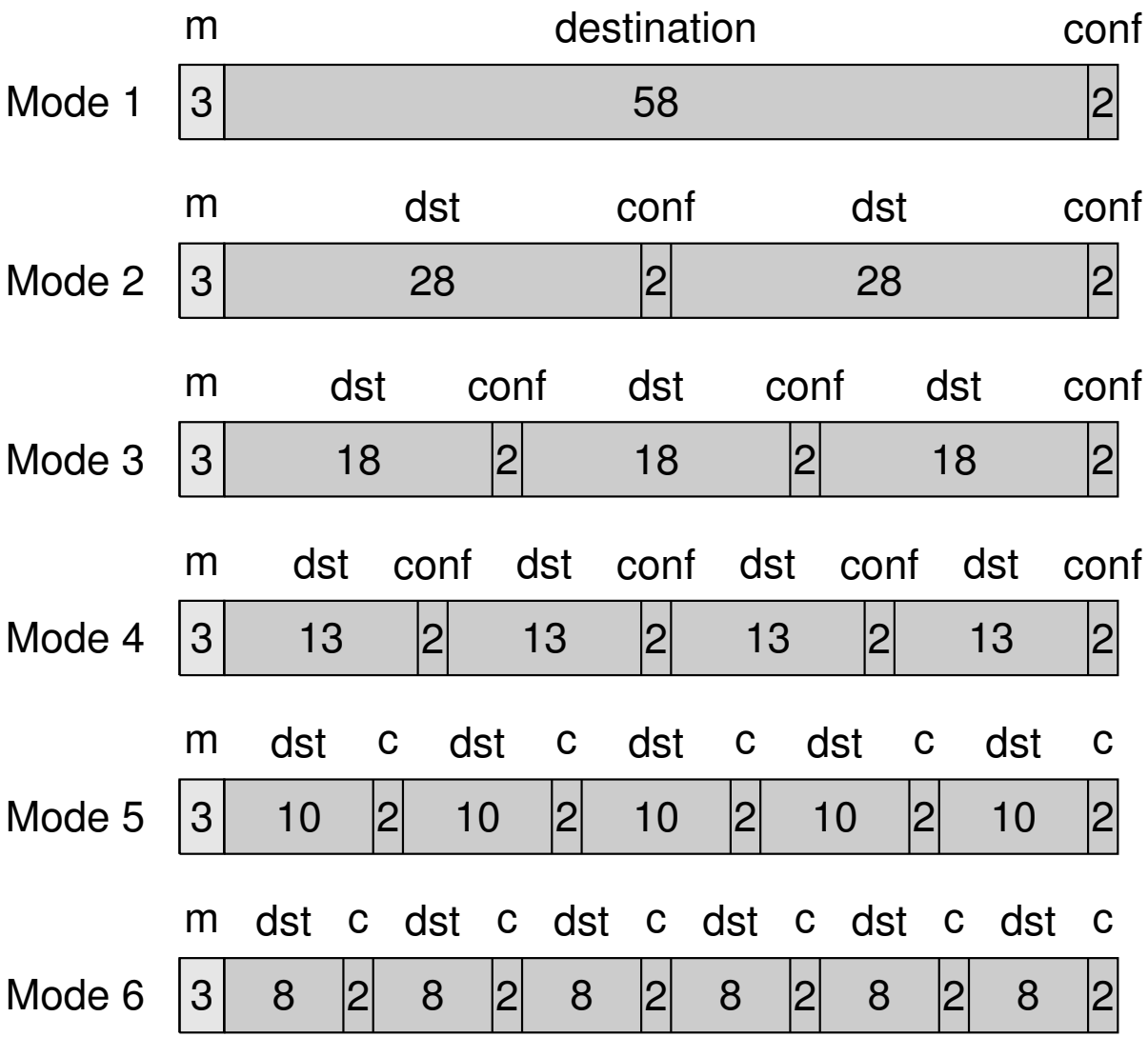
COMPRESSING DESTINATIONS



COMPRESSING DESTINATIONS



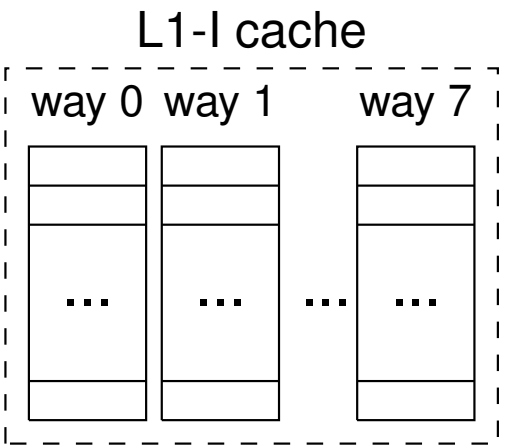
COMPRESSING DESTINATIONS



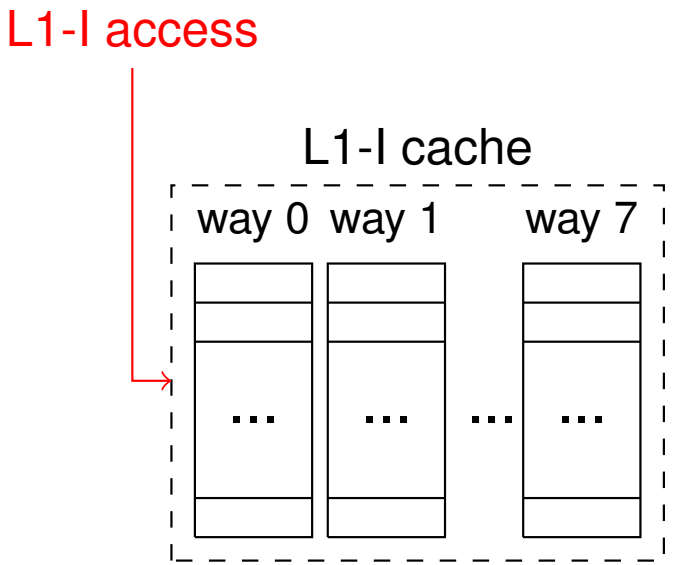
METHODOLOGY

- ChampSim develop branch (nov 2020)
- Baseline:
  - Sunny Cove-like system
  - Decoupled front-end (64-entry fetch queue)
  - 32KB L1I
- ENTANGLED:
  - *History buffer*: 16 entries
  - *Entangled table*: 2K, 4K and 8K entries
- Applications
  - 959 traces from the Championship Value Prediction (provided by Qualcomm)
  - Cloud Suite
- Analysis both for virtual and physical prefetching

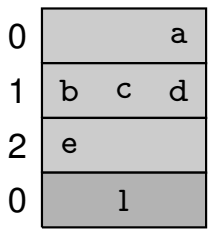
Design of the Entangling Prefetcher

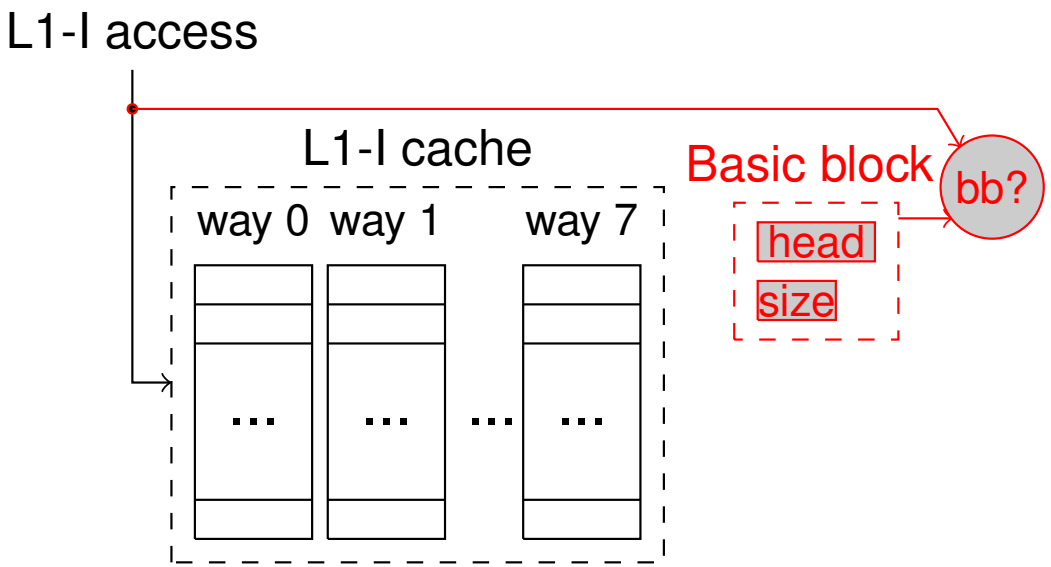


Design of the Entangling Prefetcher

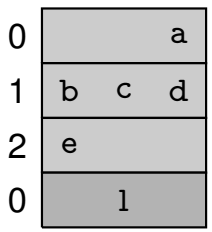


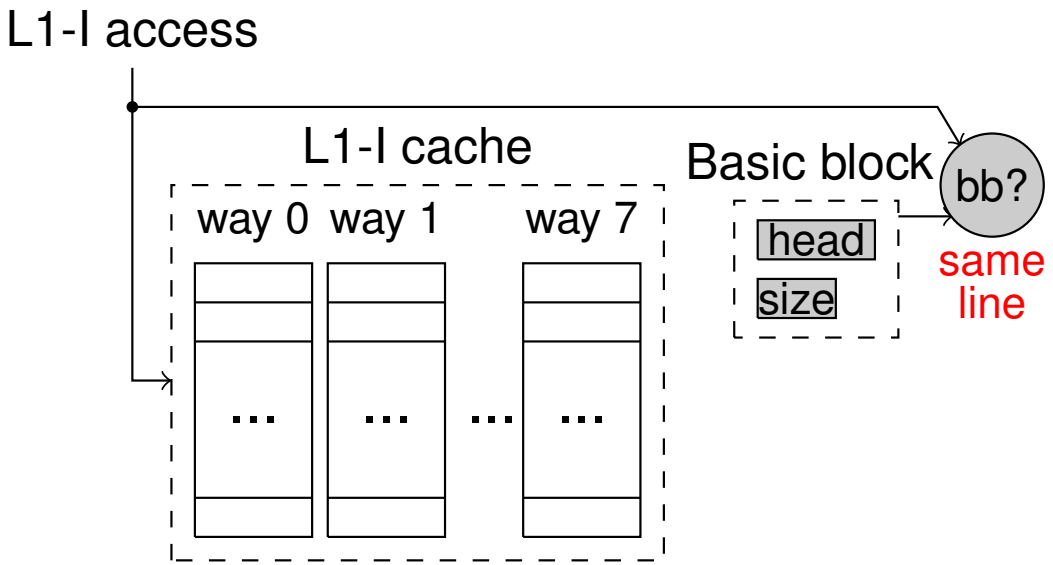
Design of the Entangling Prefetcher - Finding basic blocks



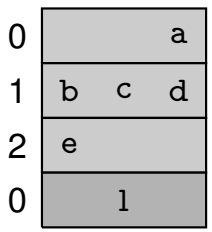


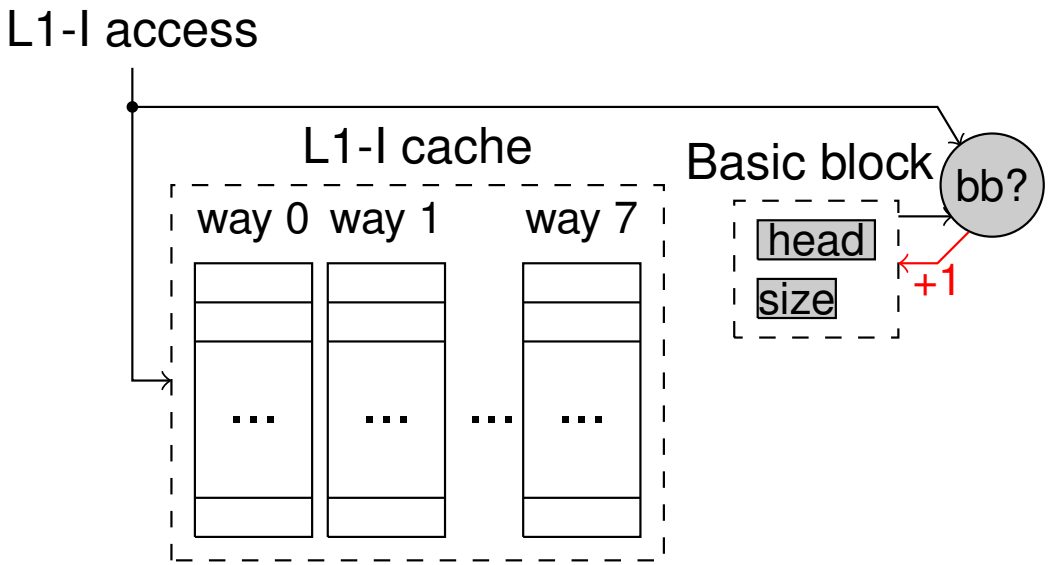
Design of the Entangling Prefetcher - Finding basic blocks



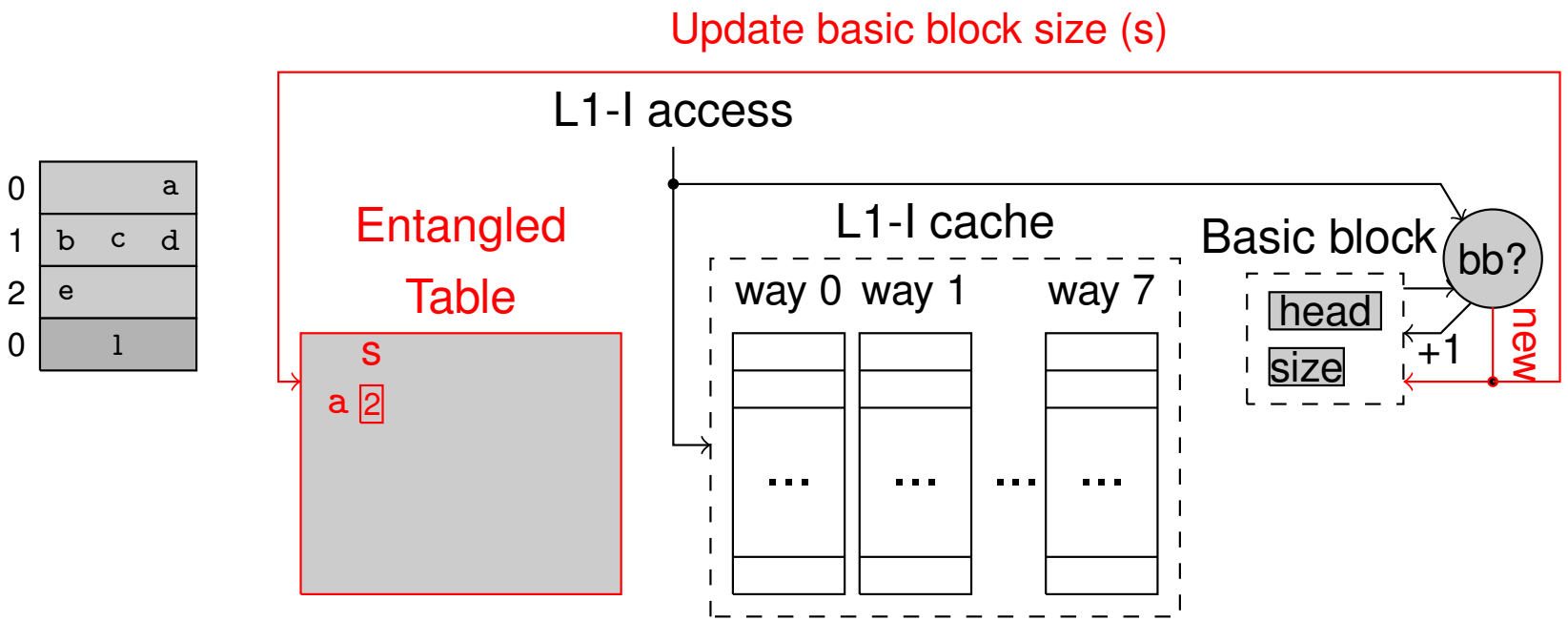


Design of the Entangling Prefetcher - Finding basic blocks

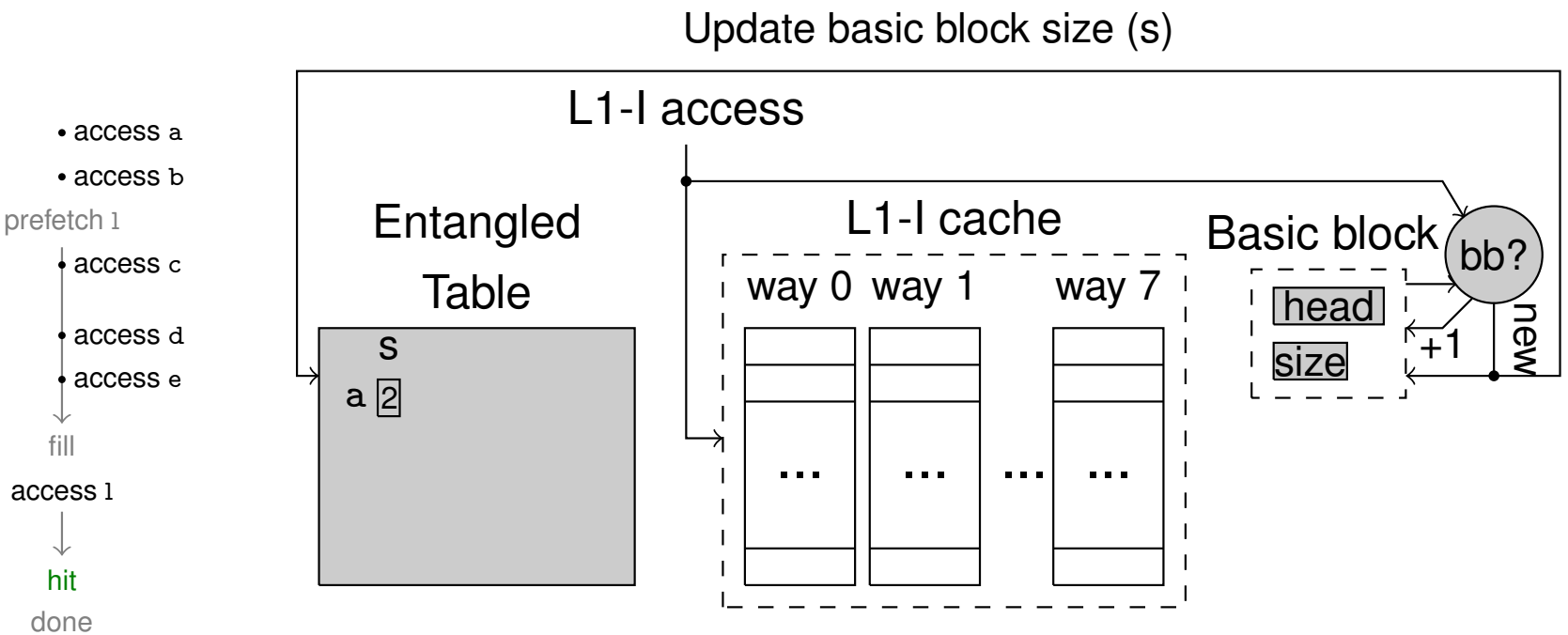




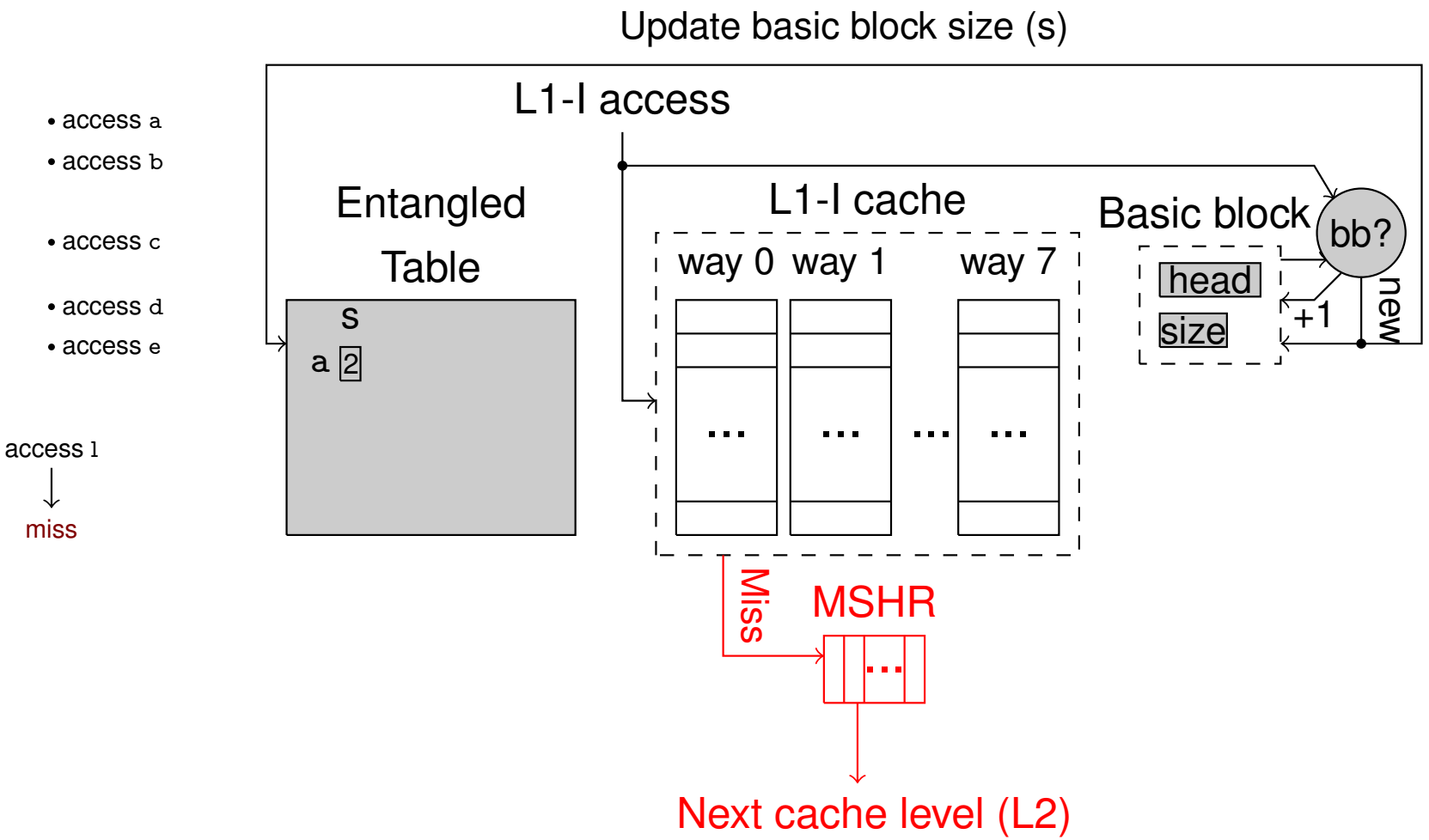
Design of the Entangling Prefetcher - Finding basic blocks



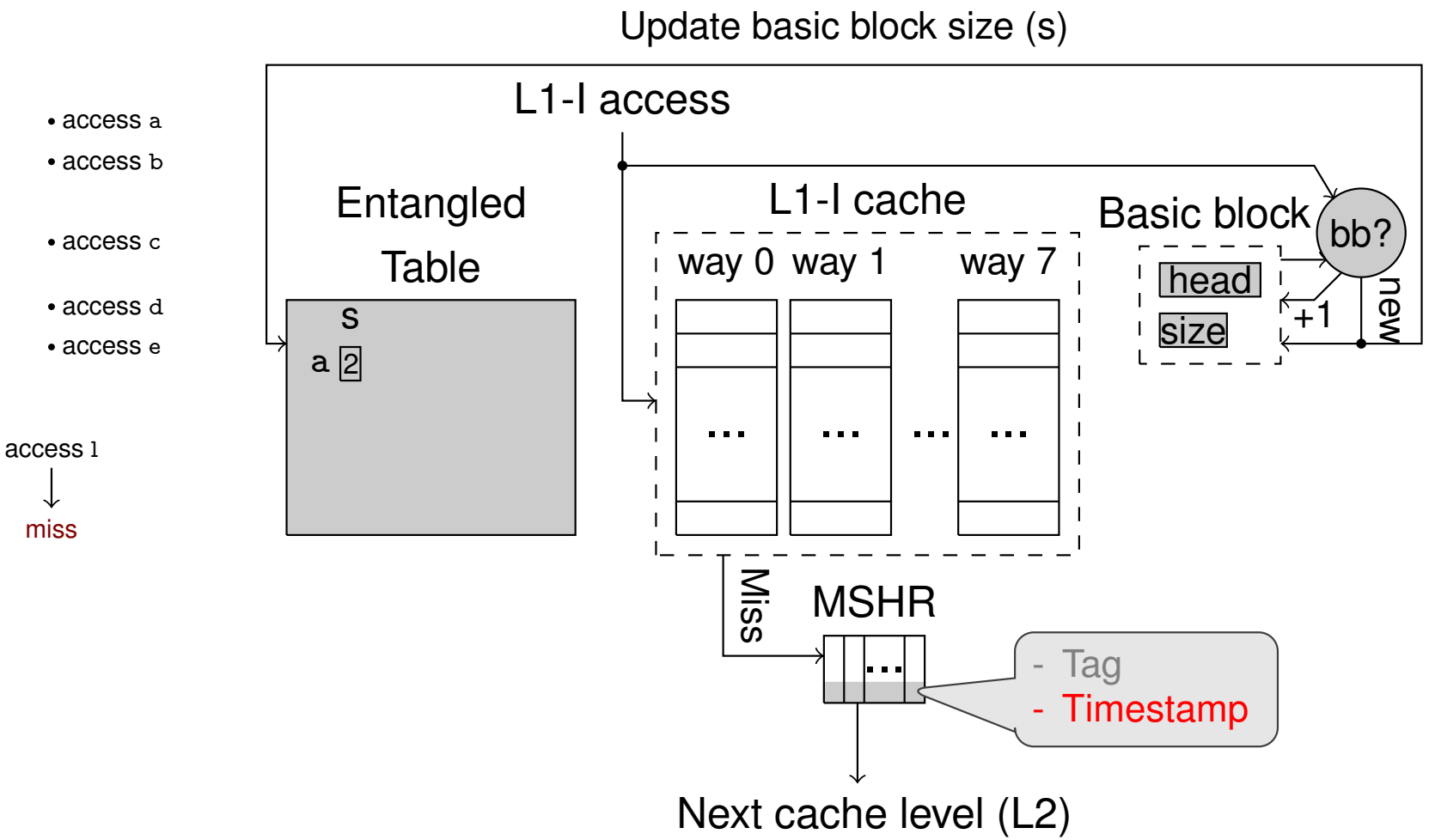
Design of the Entangling Prefetcher - Entangling cache lines



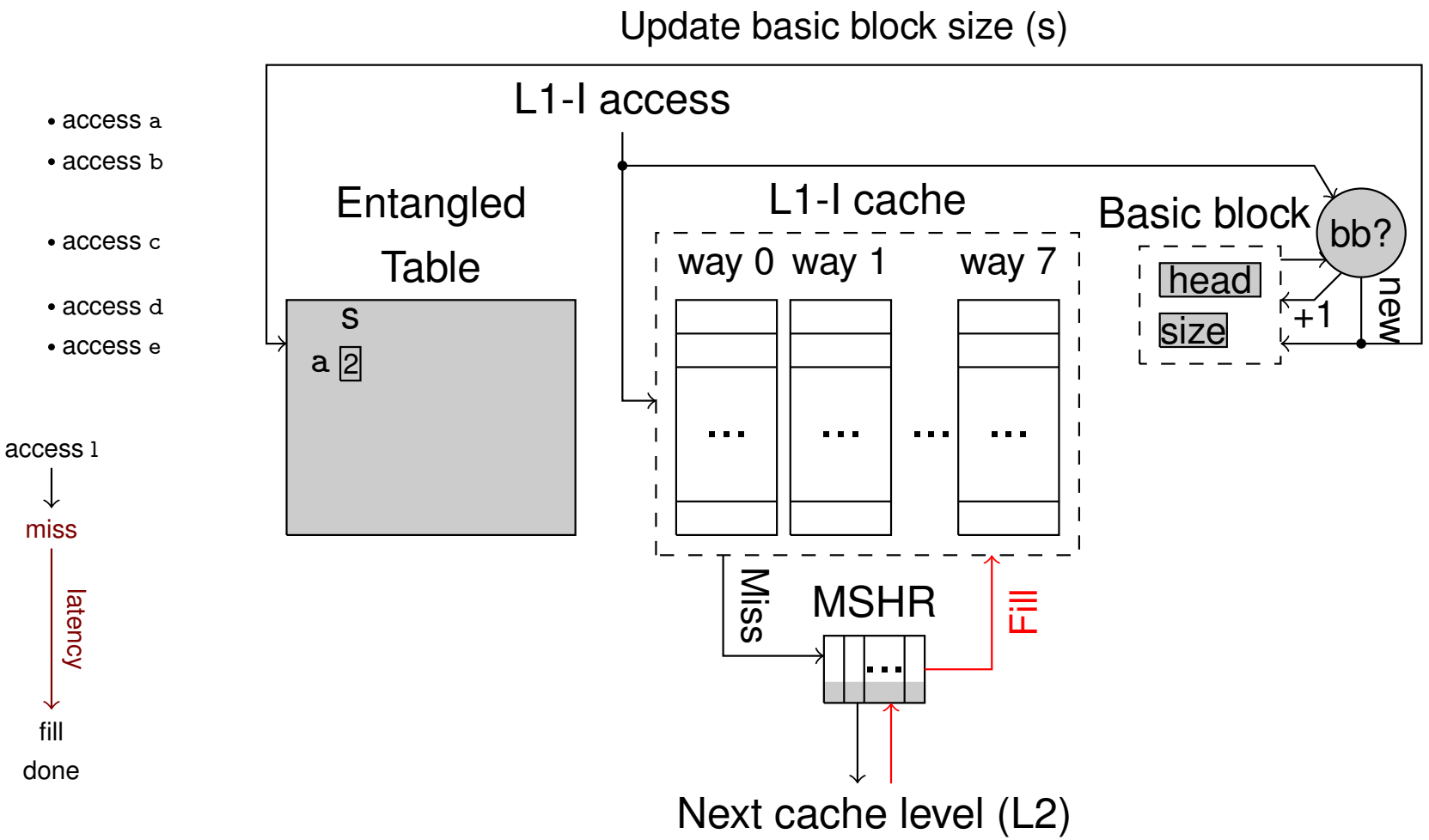
Design of the Entangling Prefetcher - Entangling cache lines



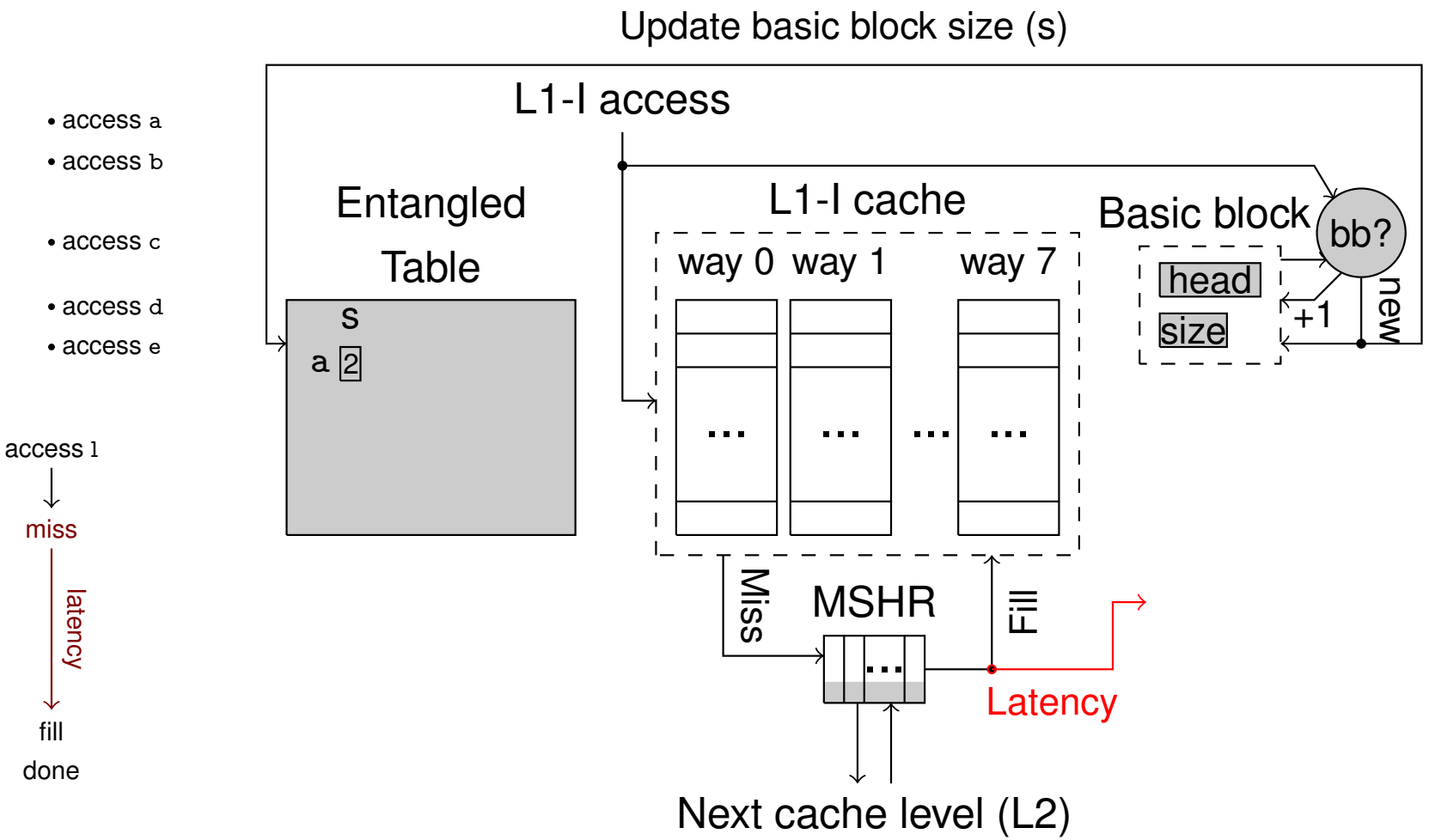
Design of the Entangling Prefetcher - Entangling cache lines



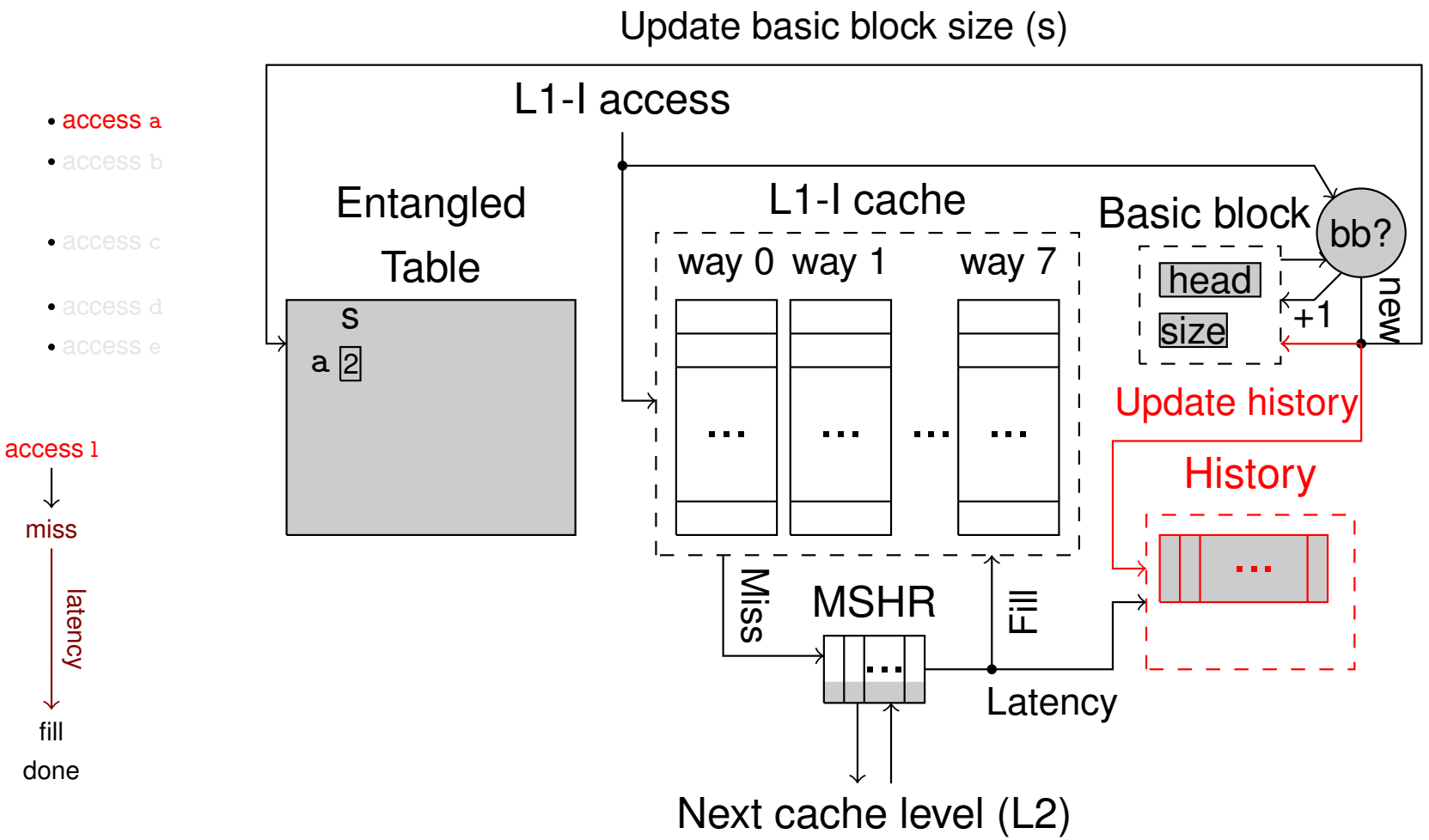
Design of the Entangling Prefetcher - Entangling cache lines



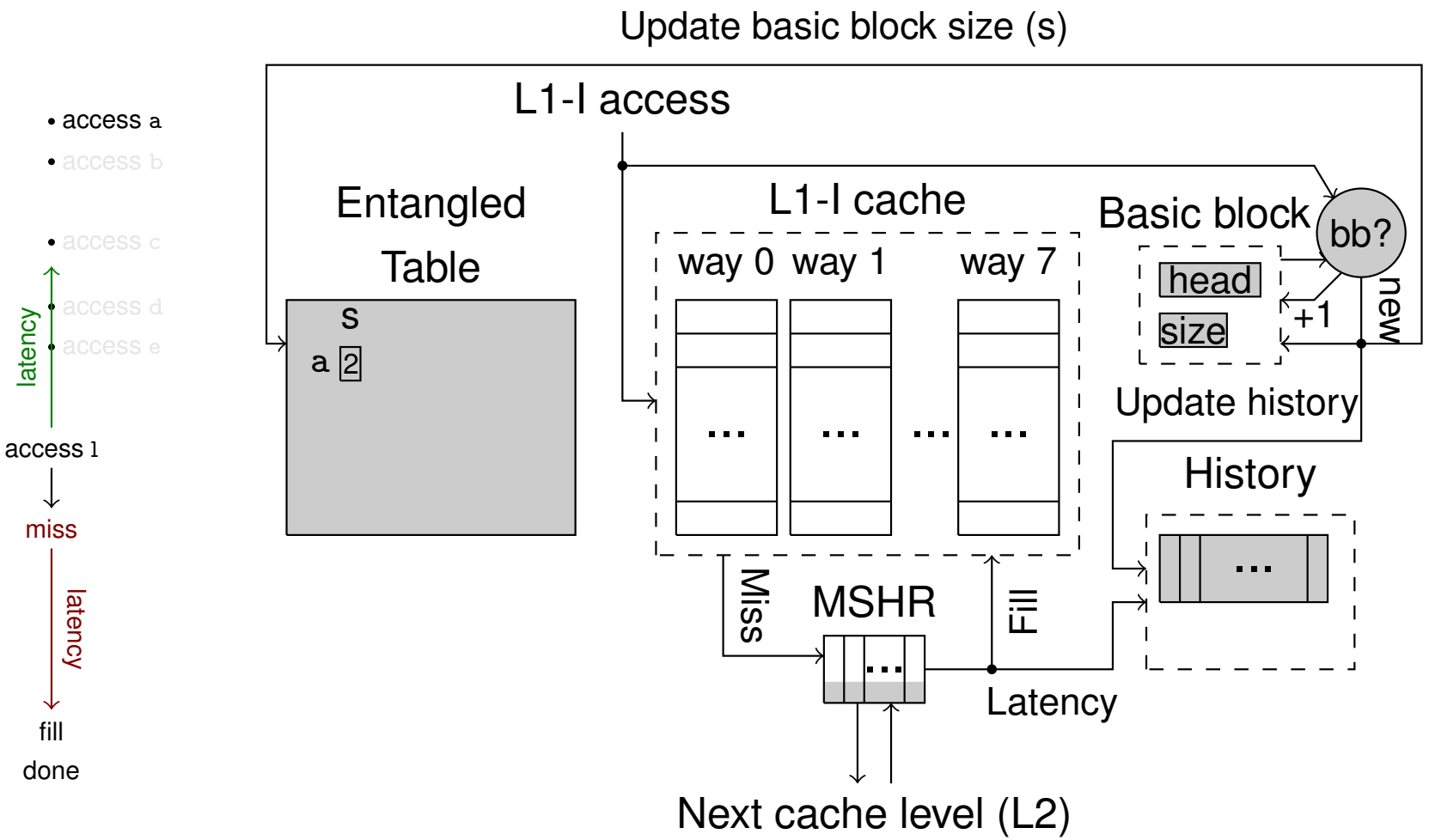
Design of the Entangling Prefetcher - Entangling cache lines



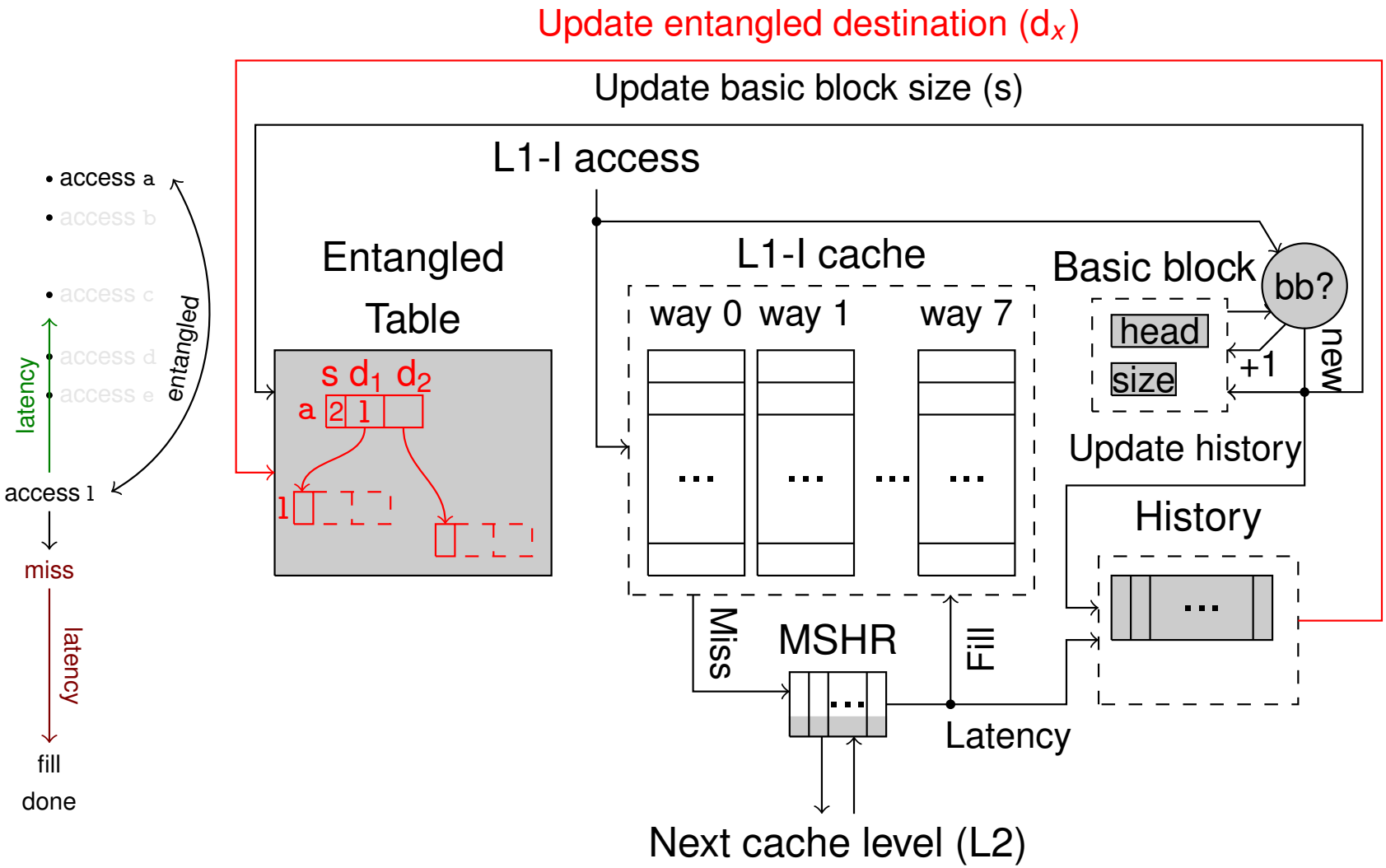
Design of the Entangling Prefetcher - Entangling cache lines



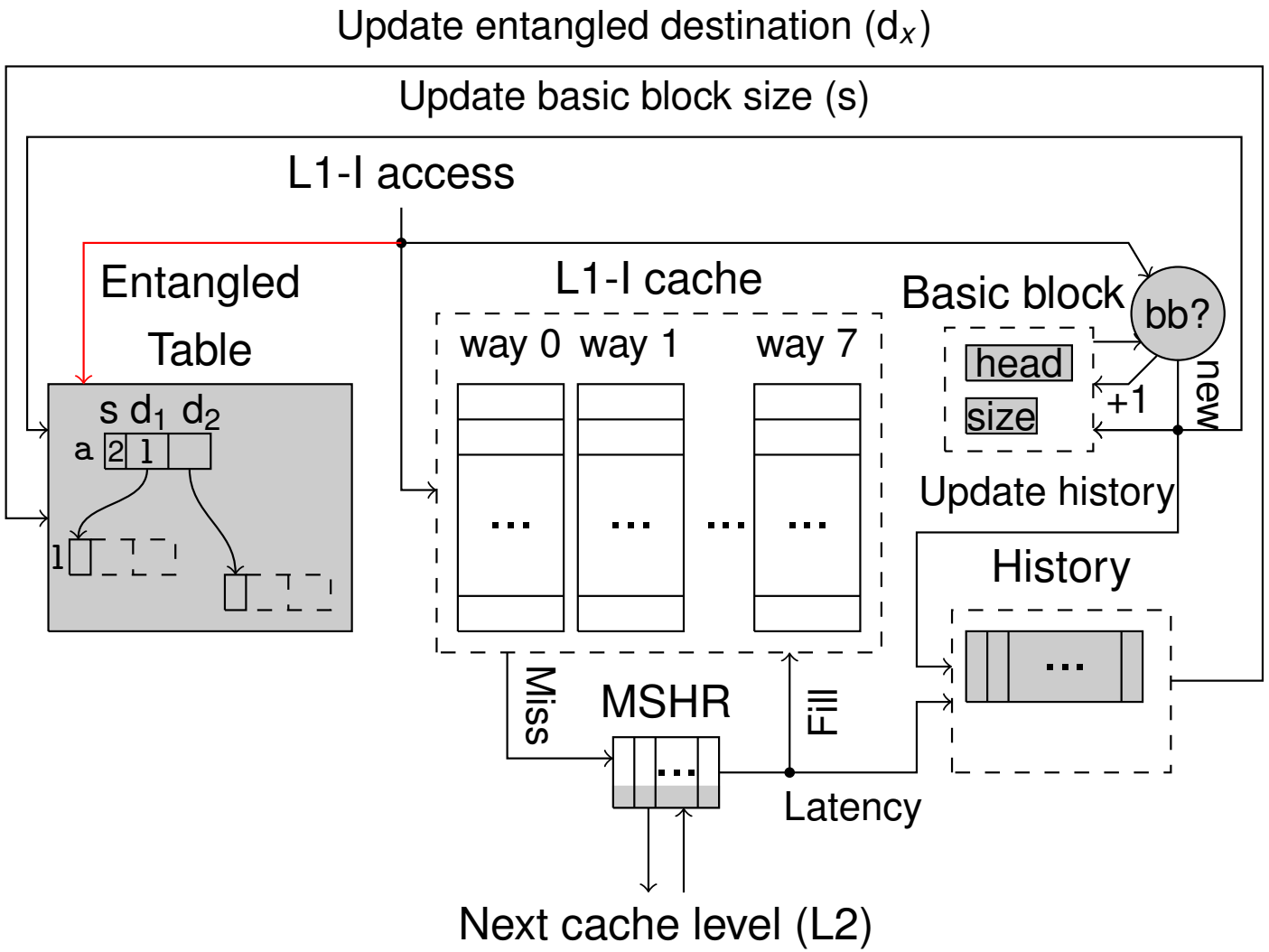
Design of the Entangling Prefetcher - Entangling cache lines



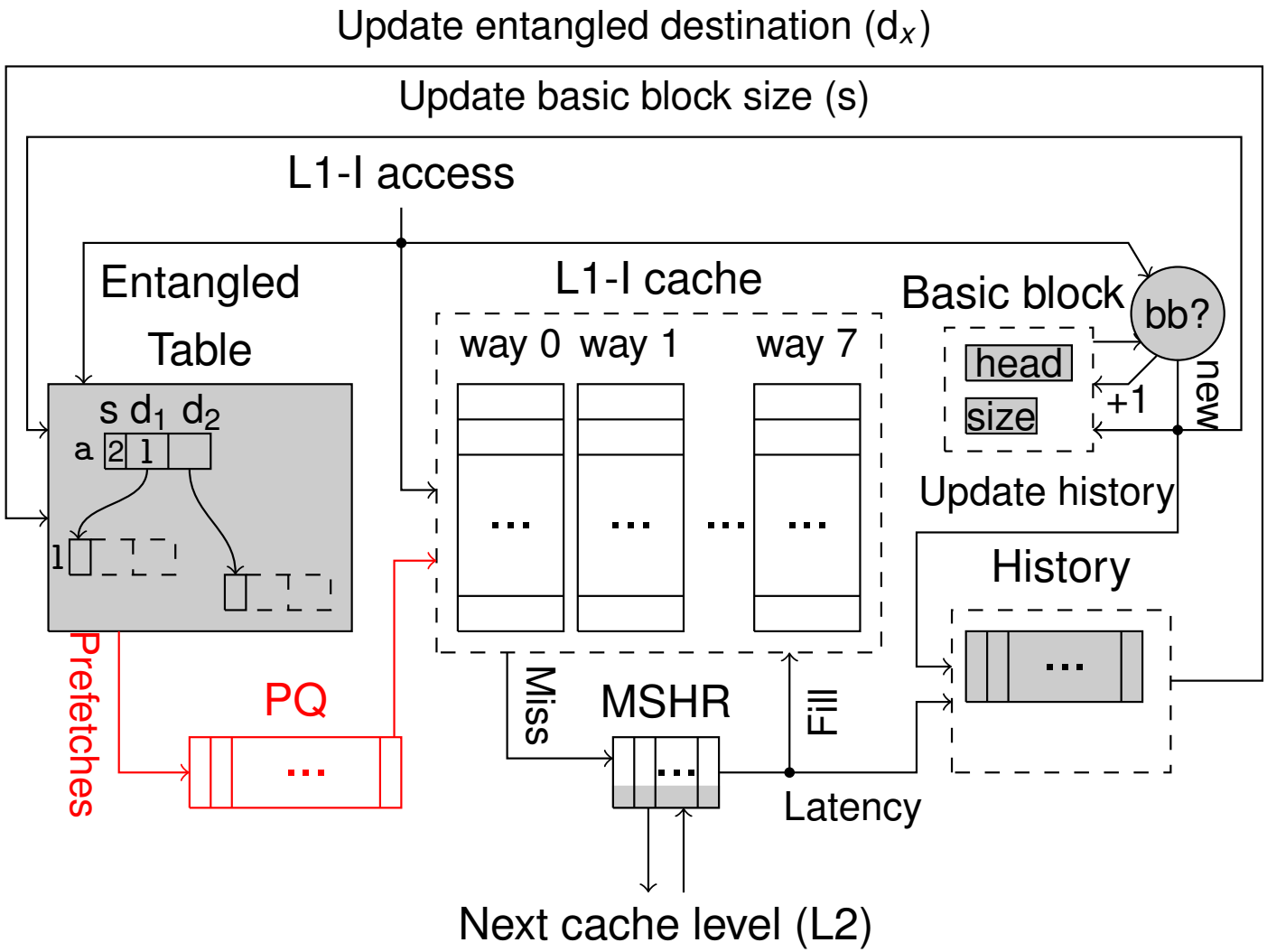
Design of the Entangling Prefetcher - Entangling cache lines



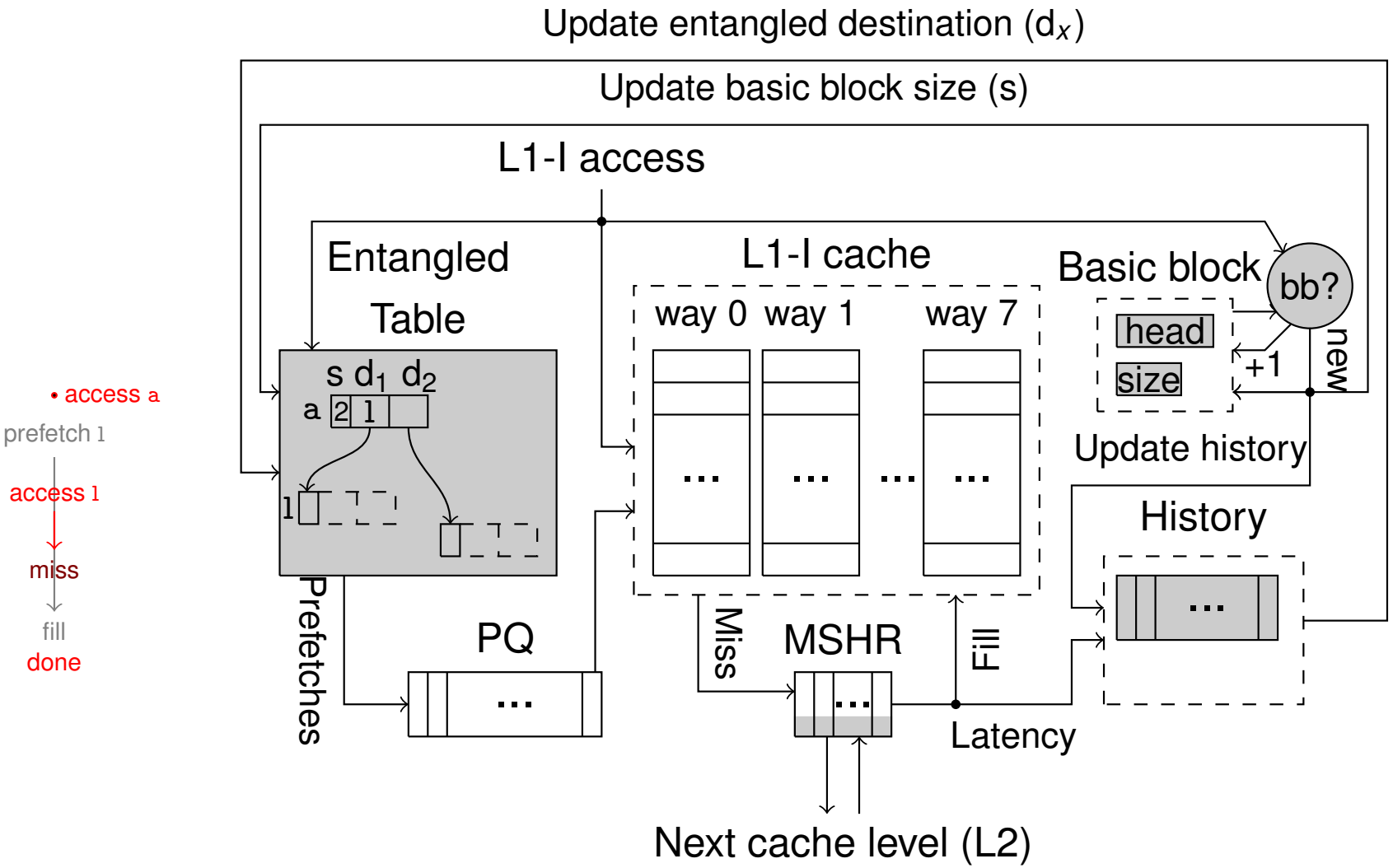
Design of the Entangling Prefetcher - Issuing prefetches



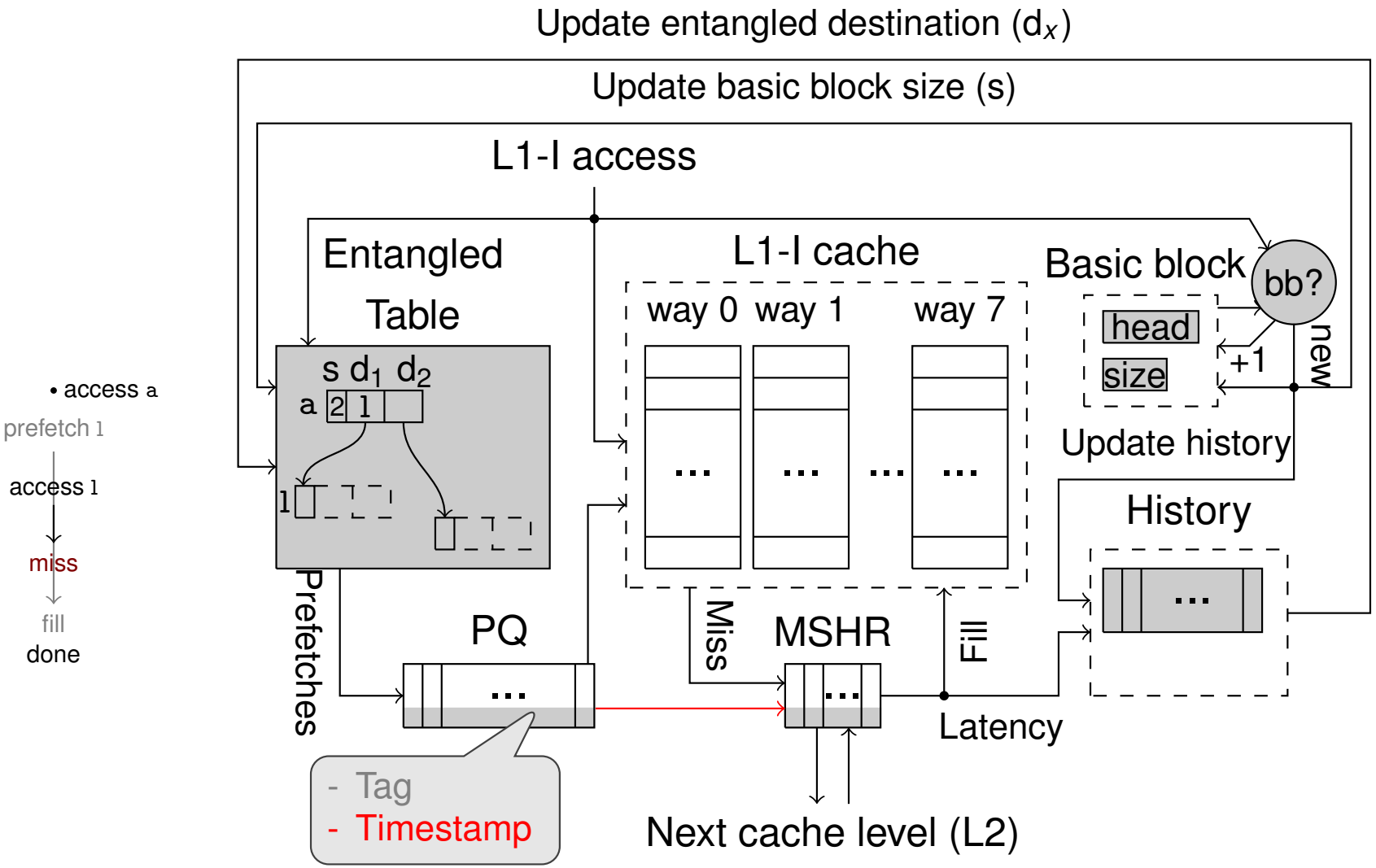
Design of the Entangling Prefetcher - Issuing prefetches



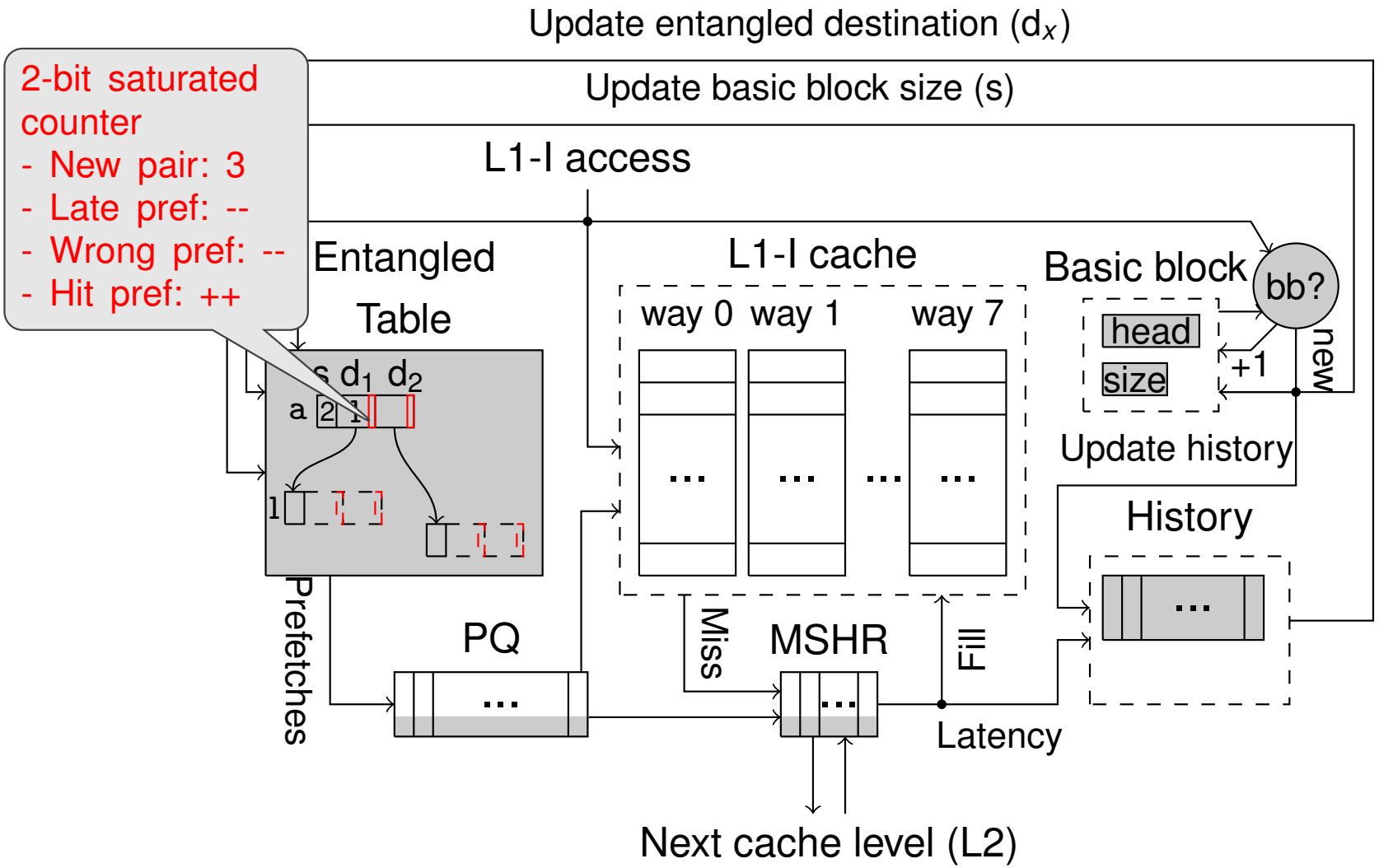
Design of the Entangling Prefetcher - Fixing Late Prefetches



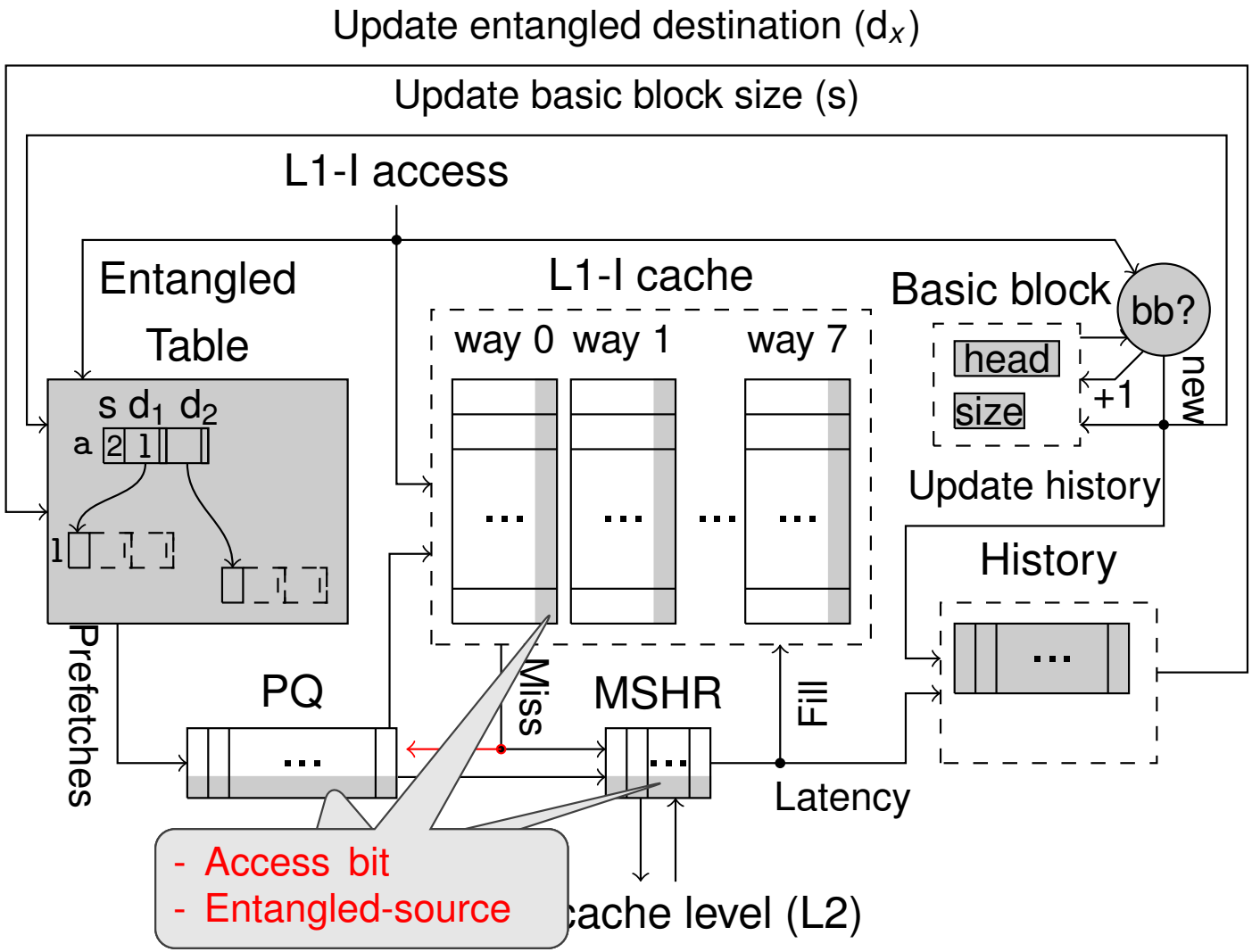
Design of the Entangling Prefetcher - Fixing late prefetches



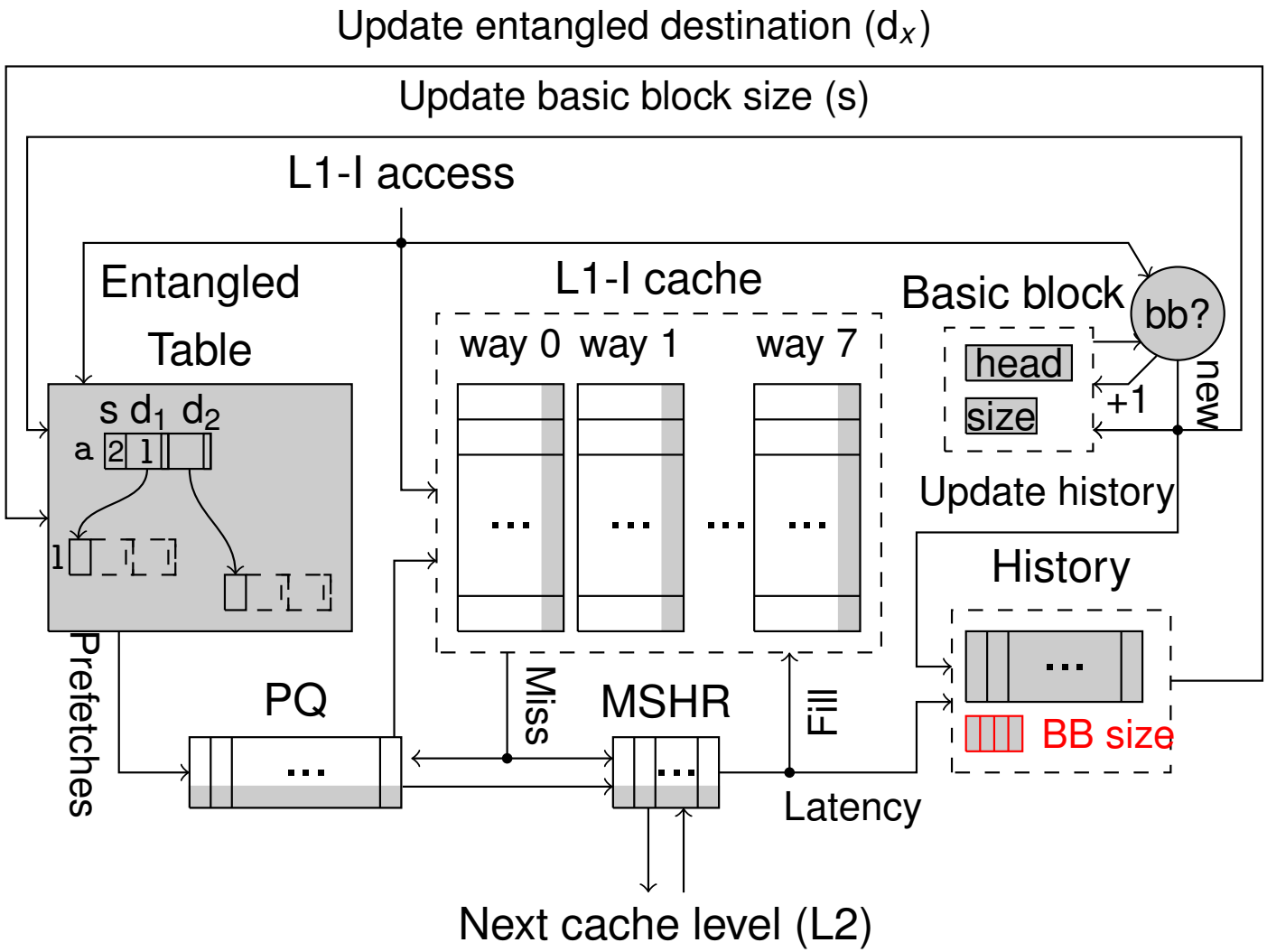
Design of the Entangling Prefetcher - Confidence for entangled pairs



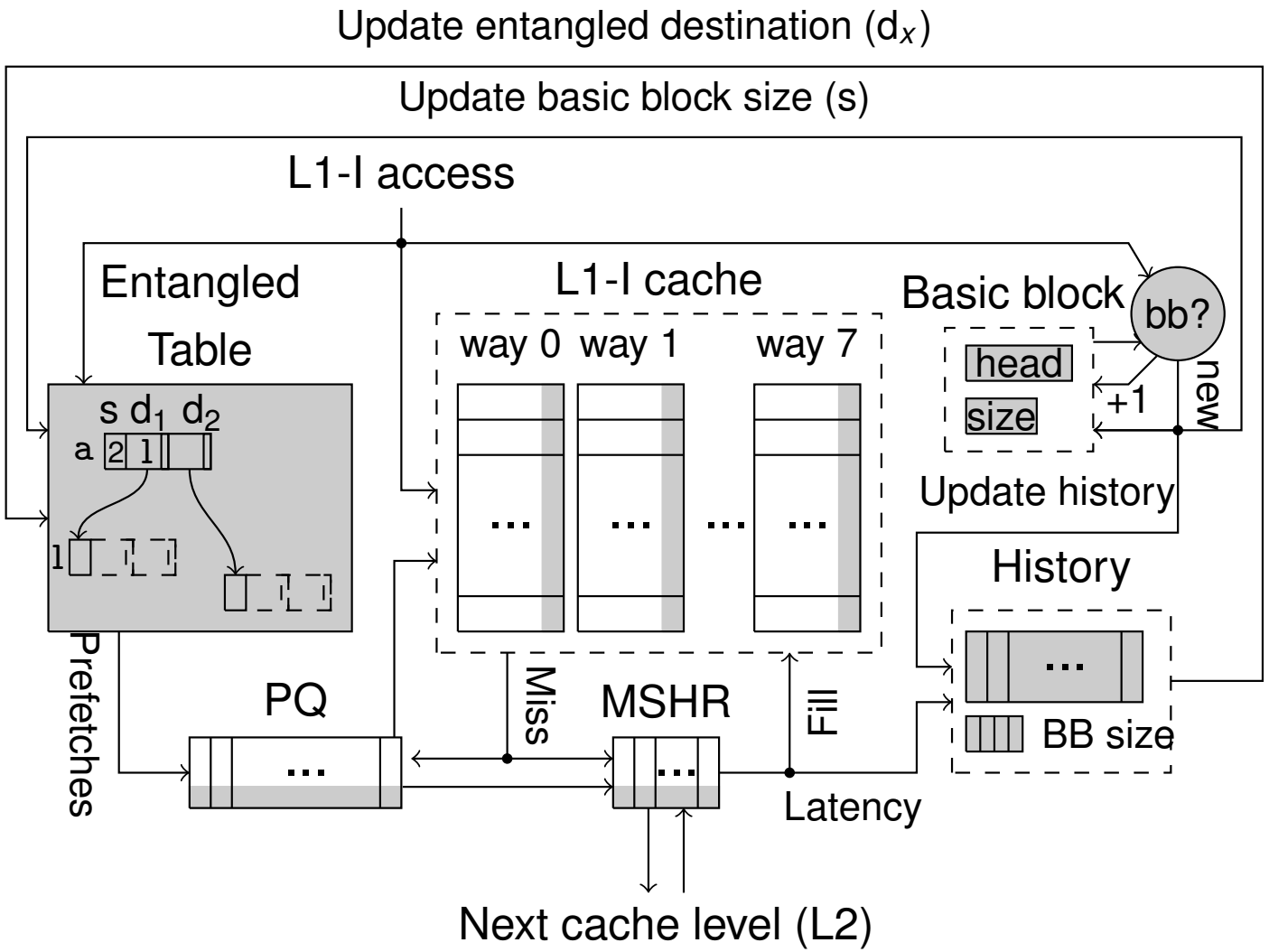
Design of the Entangling Prefetcher - Confidence for entangled pairs



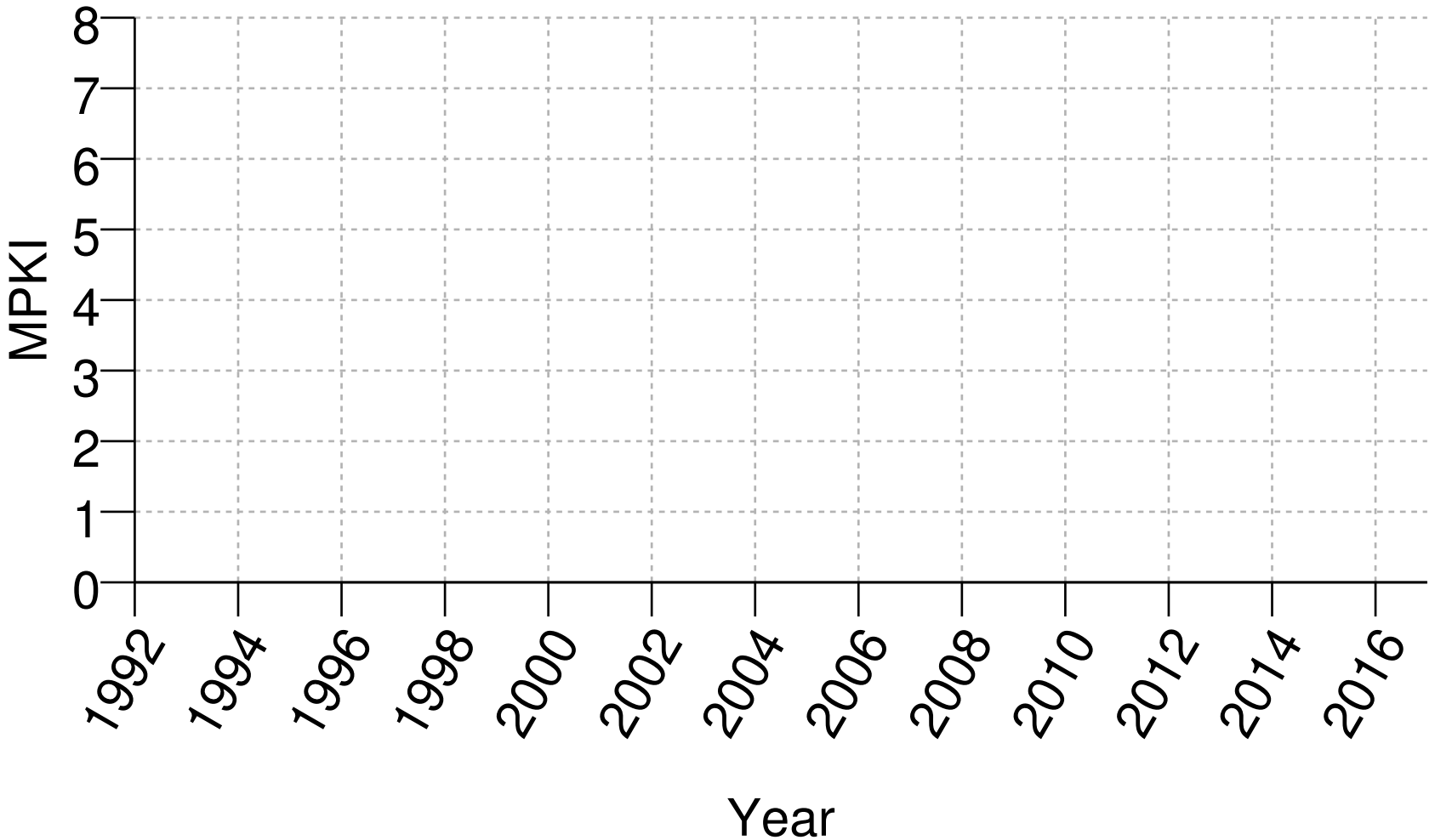
Design of the Entangling Prefetcher - Merging basic blocks



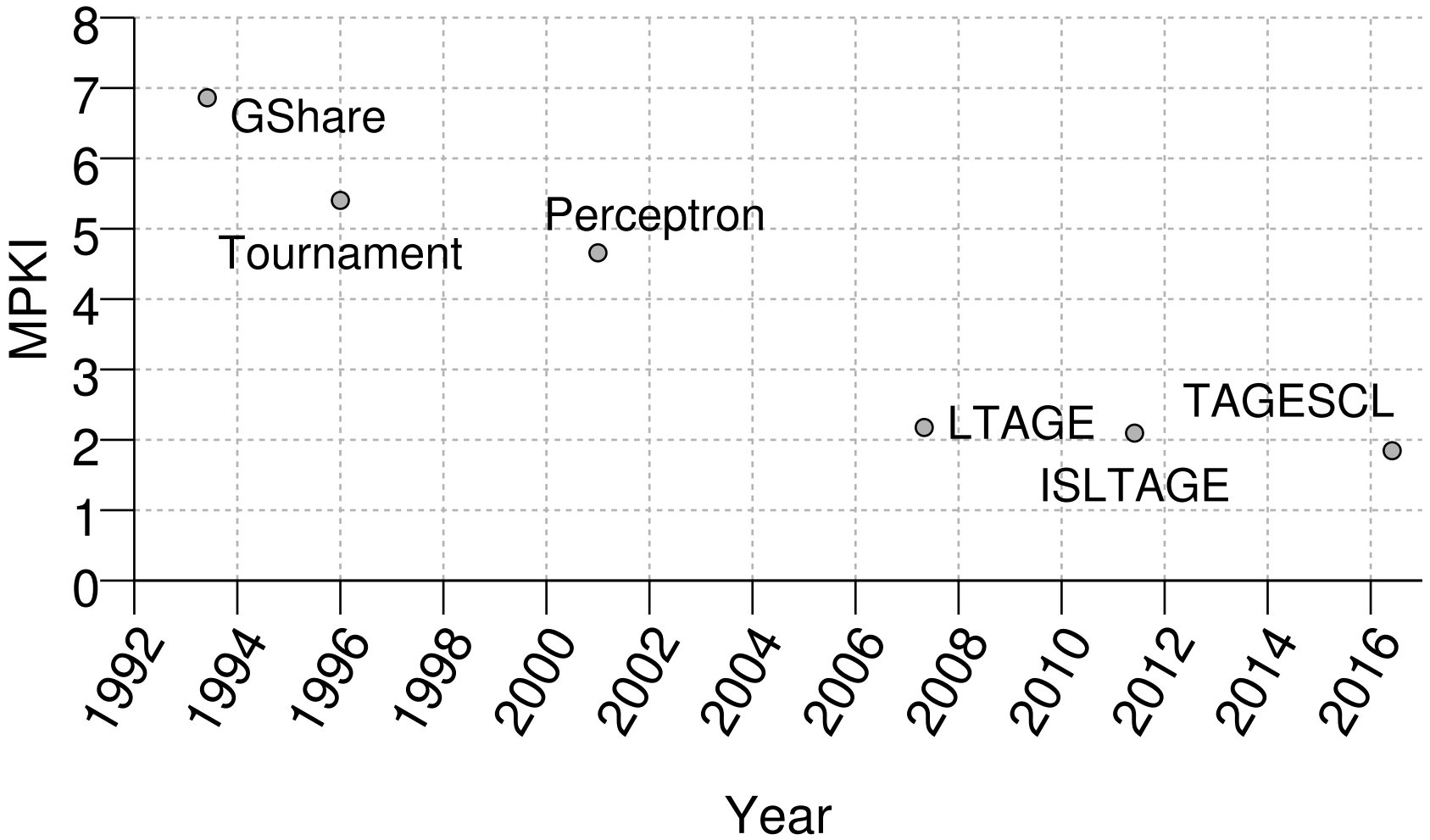
Design of the Entangling Prefetcher



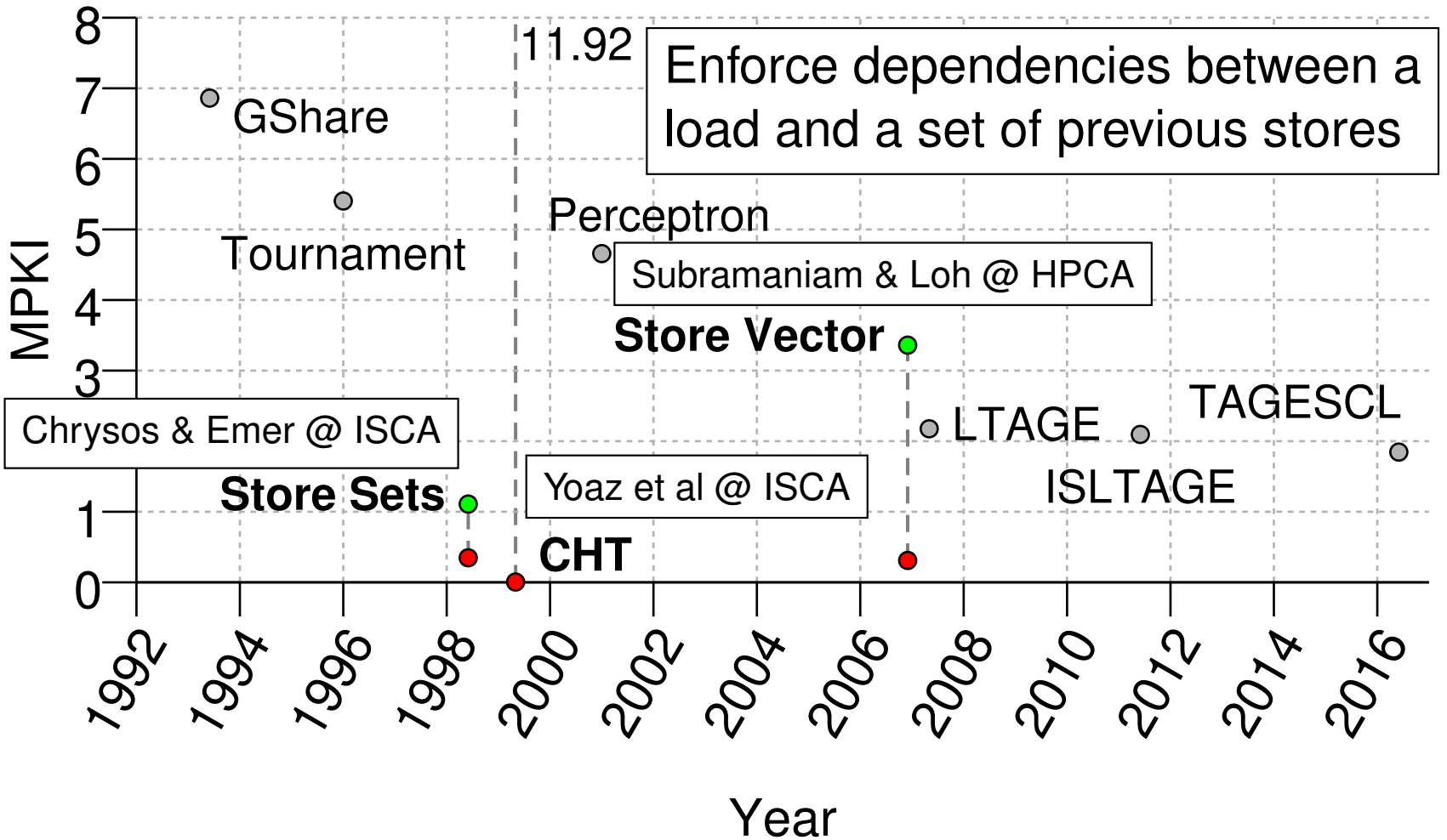
MDP, WAS NOT SOLVED IN THE 90'S?



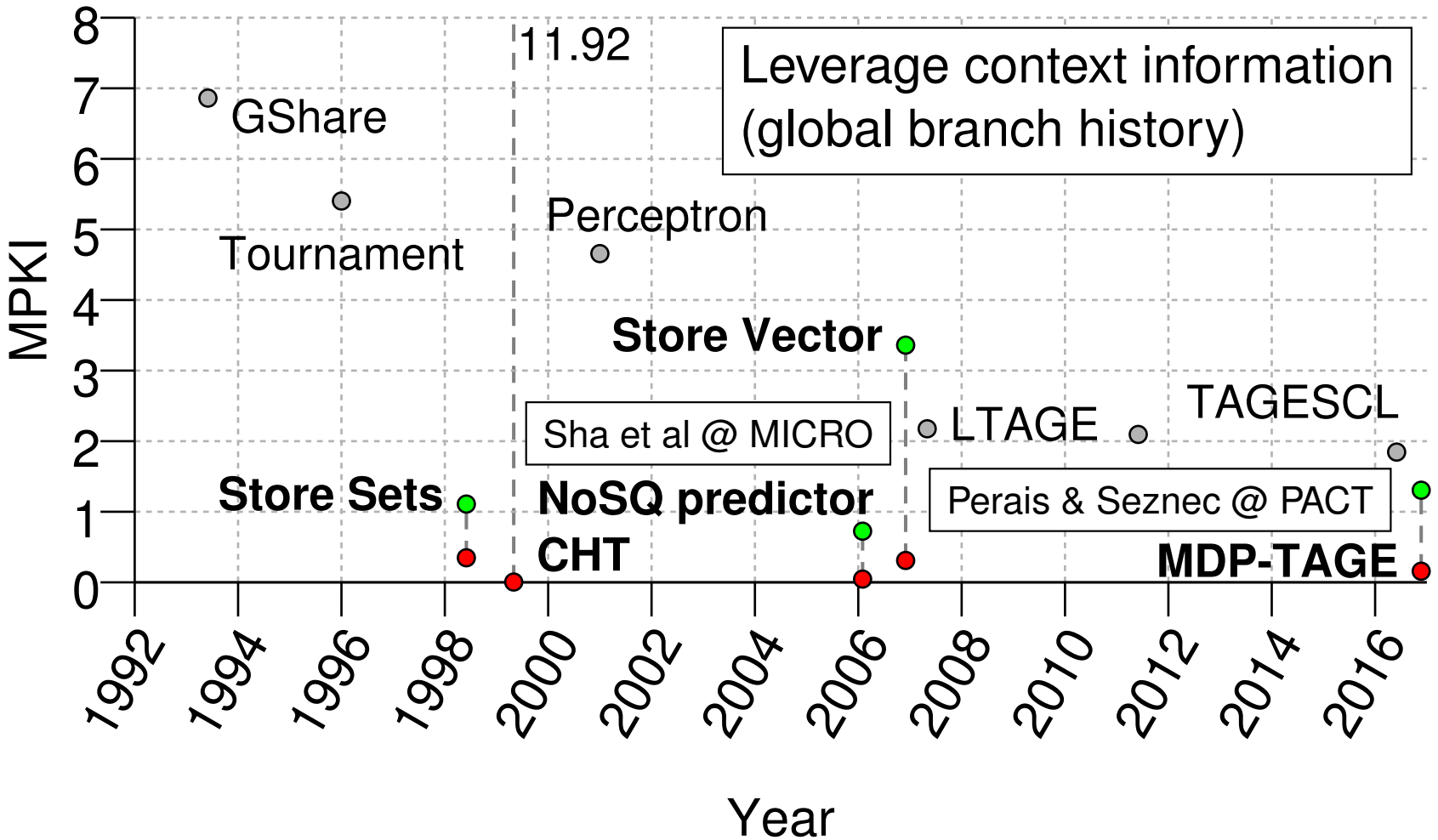
MDP, WAS NOT SOLVED IN THE 90'S?



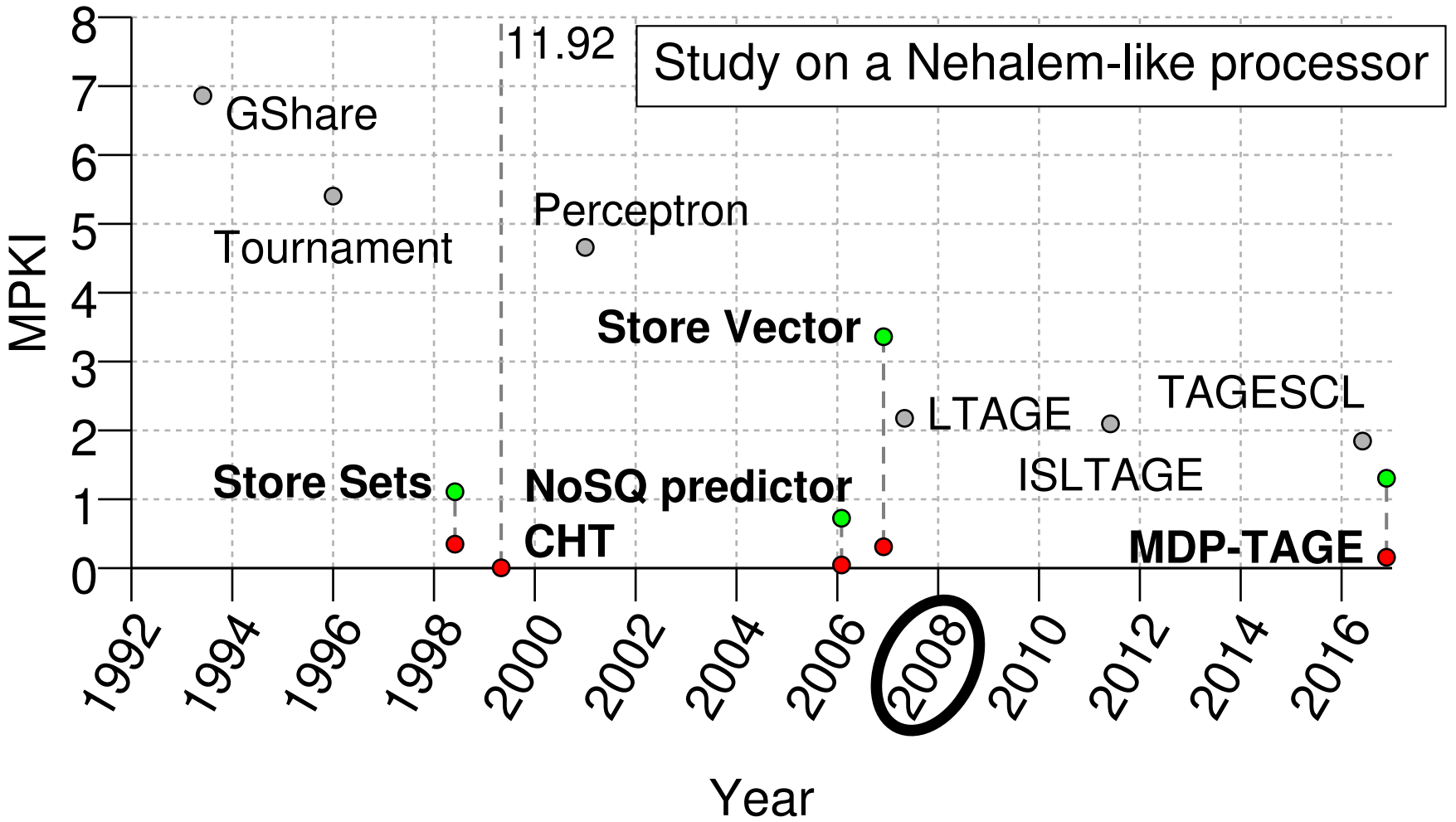
MDP, WAS NOT SOLVED IN THE 90'S?



MDP, WAS NOT SOLVED IN THE 90'S?



MDP, WAS NOT SOLVED IN THE 90'S?

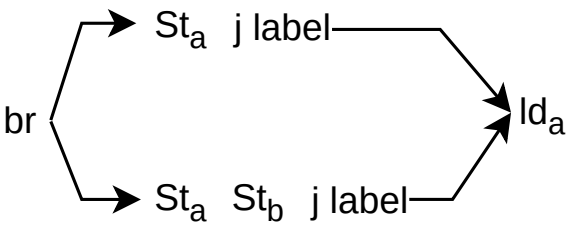


PHAST: LEARNING THE PROPER CONTEXT

- PHAST defines the *proper* length as $N + 1$, where N is the number of divergent branches between the store and the load

PHAST: Learning the proper context

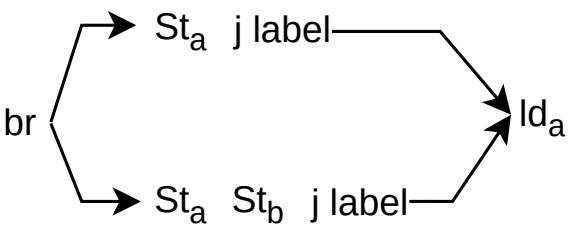
- PHAST defines the *proper* length as $N + 1$, where N is the number of divergent branches between the store and the load
- Why $+1$?



Conflict	Store Distance	History
Up	1	label
Down	2	label

PHAST: Learning the proper context

- PHAST defines the *proper* length as $N + 1$, where N is the number of divergent branches between the store and the load
- Why $+1$?

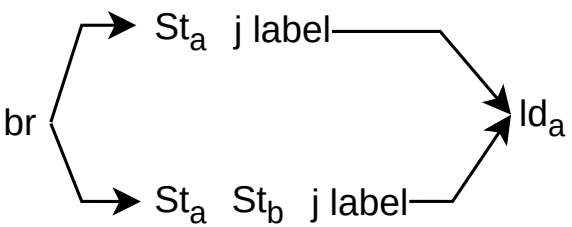


Conflict	Store Distance	History
Up	1	label
Down	2	label

Same history, different distances!

PHAST: LEARNING THE PROPER CONTEXT

- PHAST defines the *proper* length as $N + 1$, where N is the number of divergent branches between the store and the load
- Why $+1$?



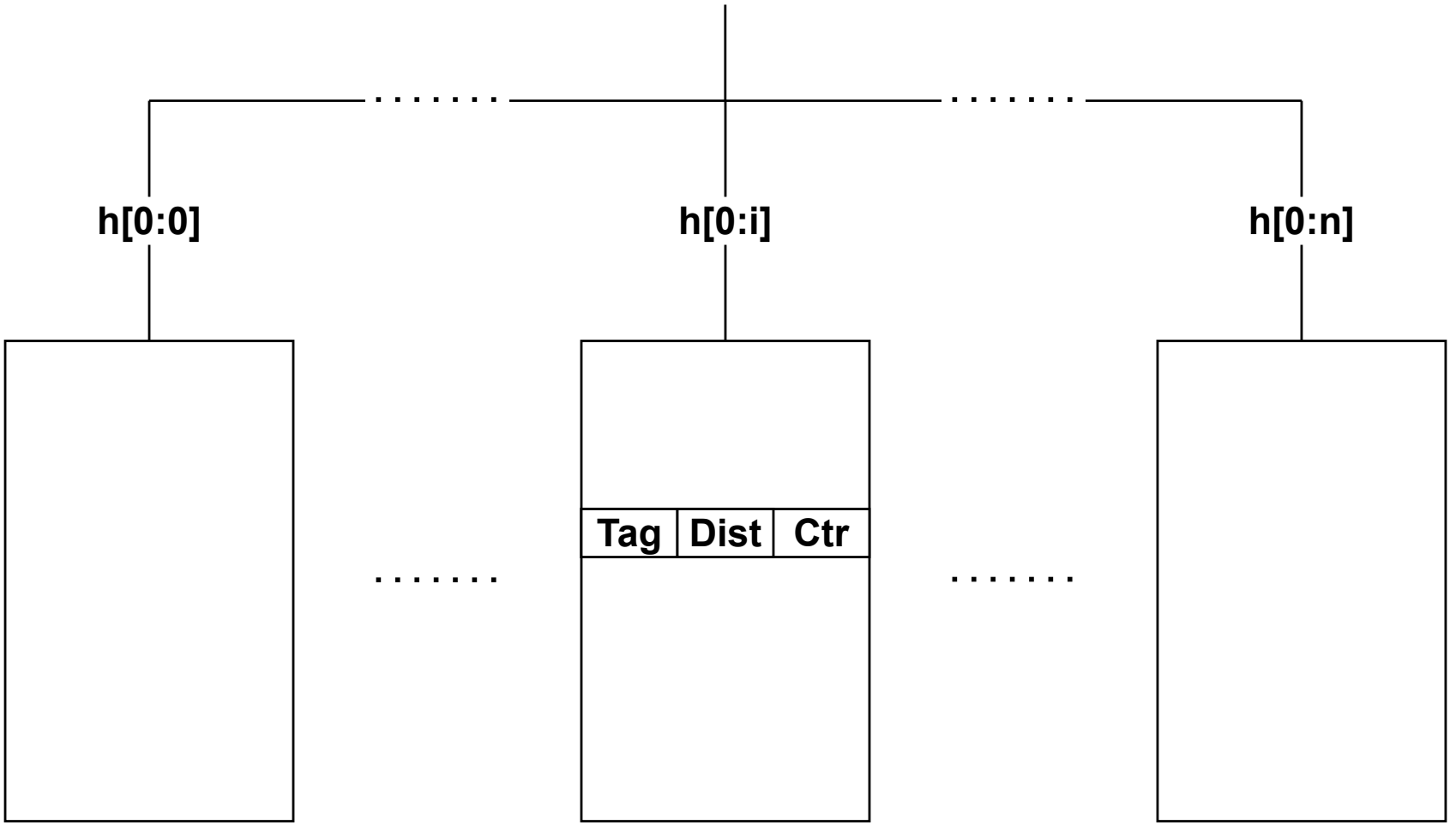
Conflict	Store Distance	History
Up	1	label
Down	2	label

Same history, different distances!

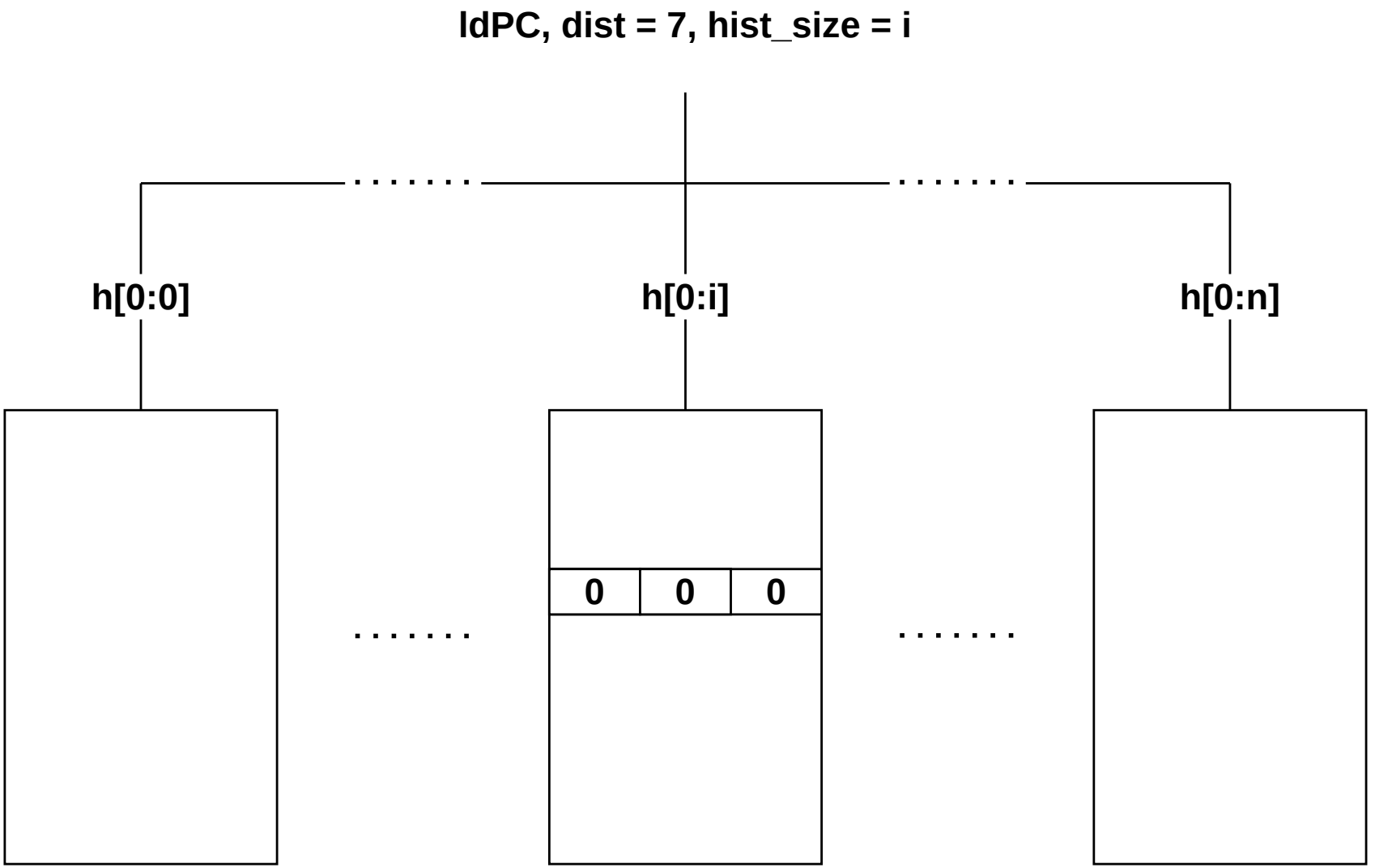
SOLUTION Use target of the branch previous to the store to differentiate the paths

ld_a	br	br	br	st_a	br	br

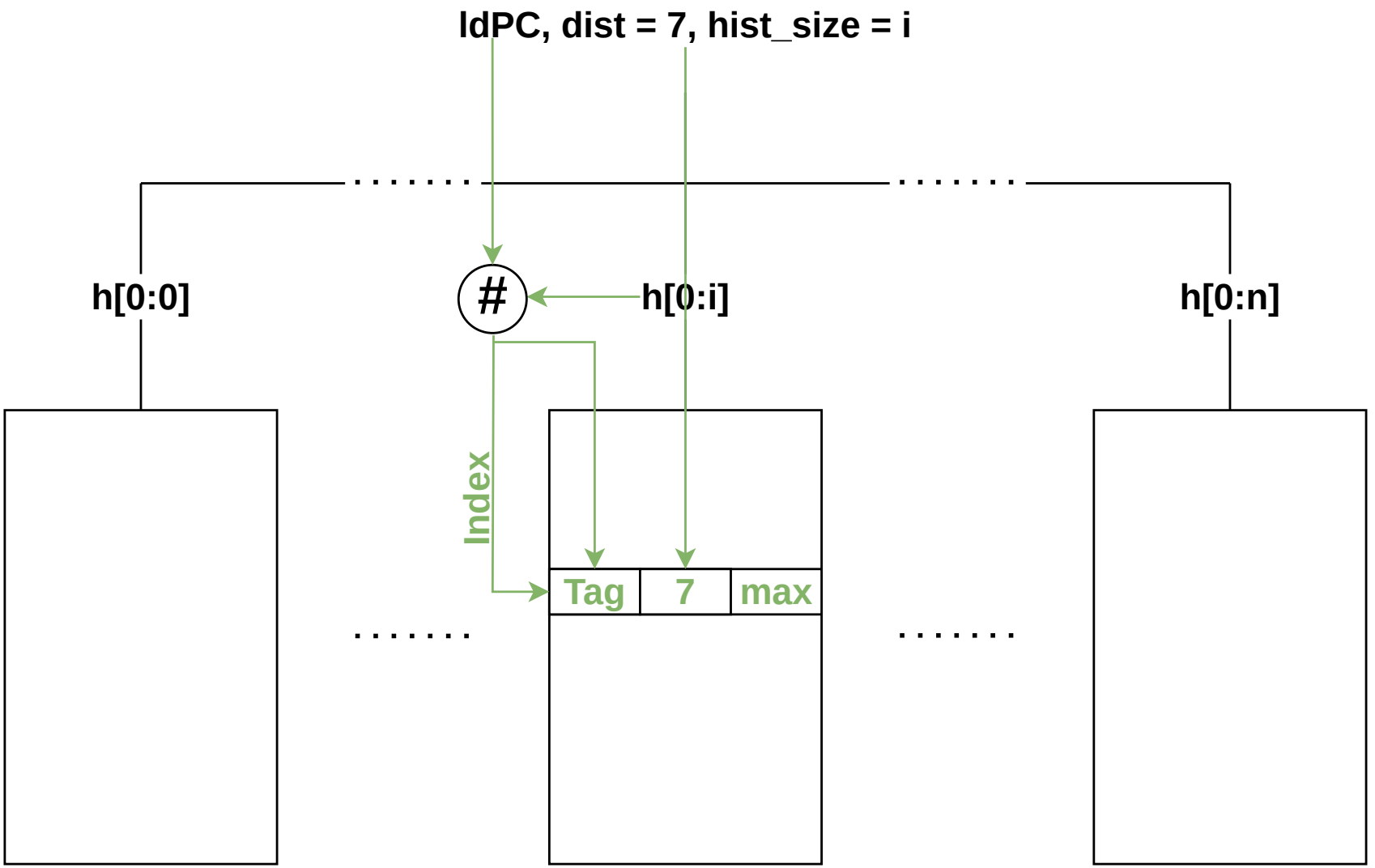
PHAST: STRUCTURE



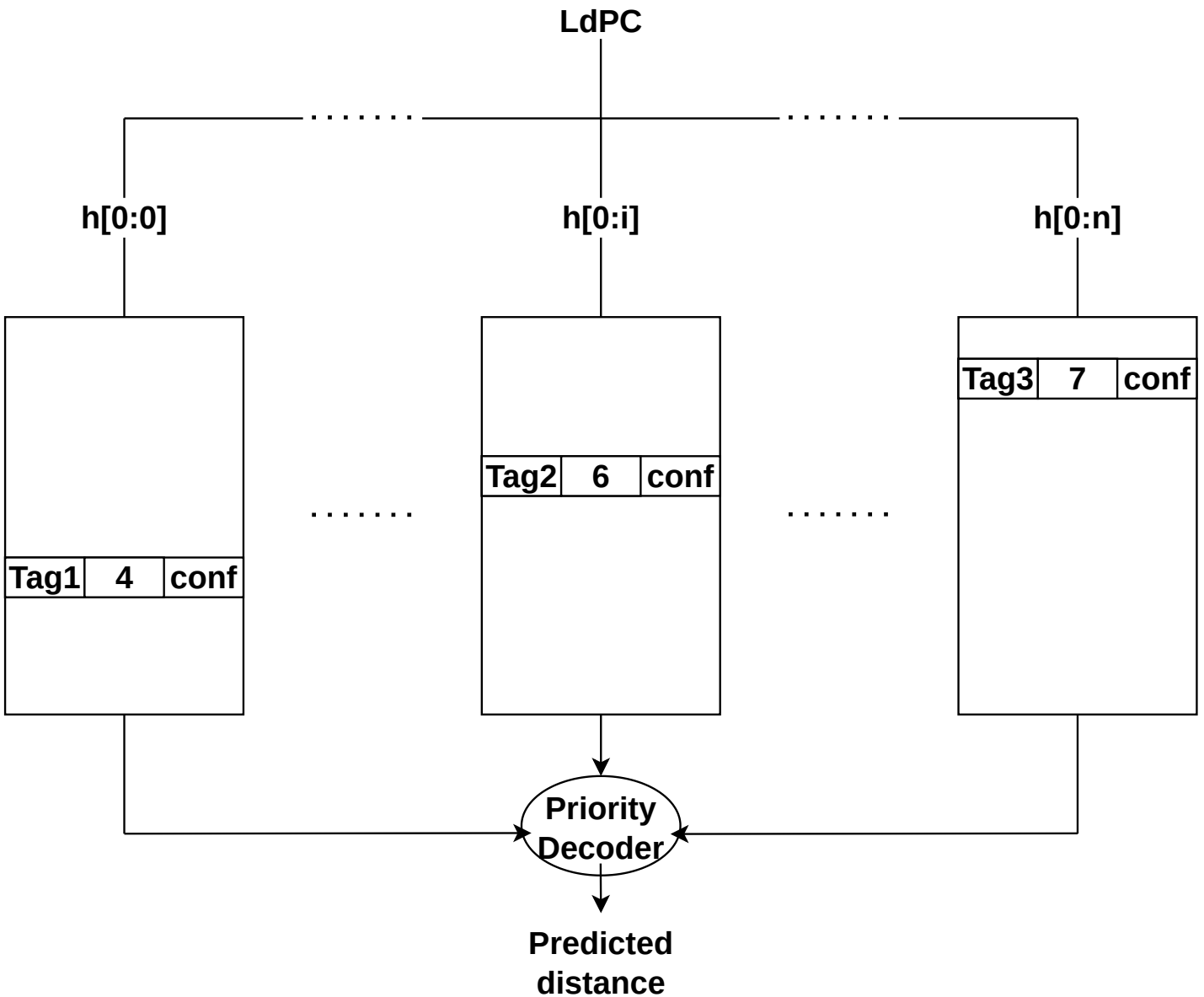
PHAST: UPDATE



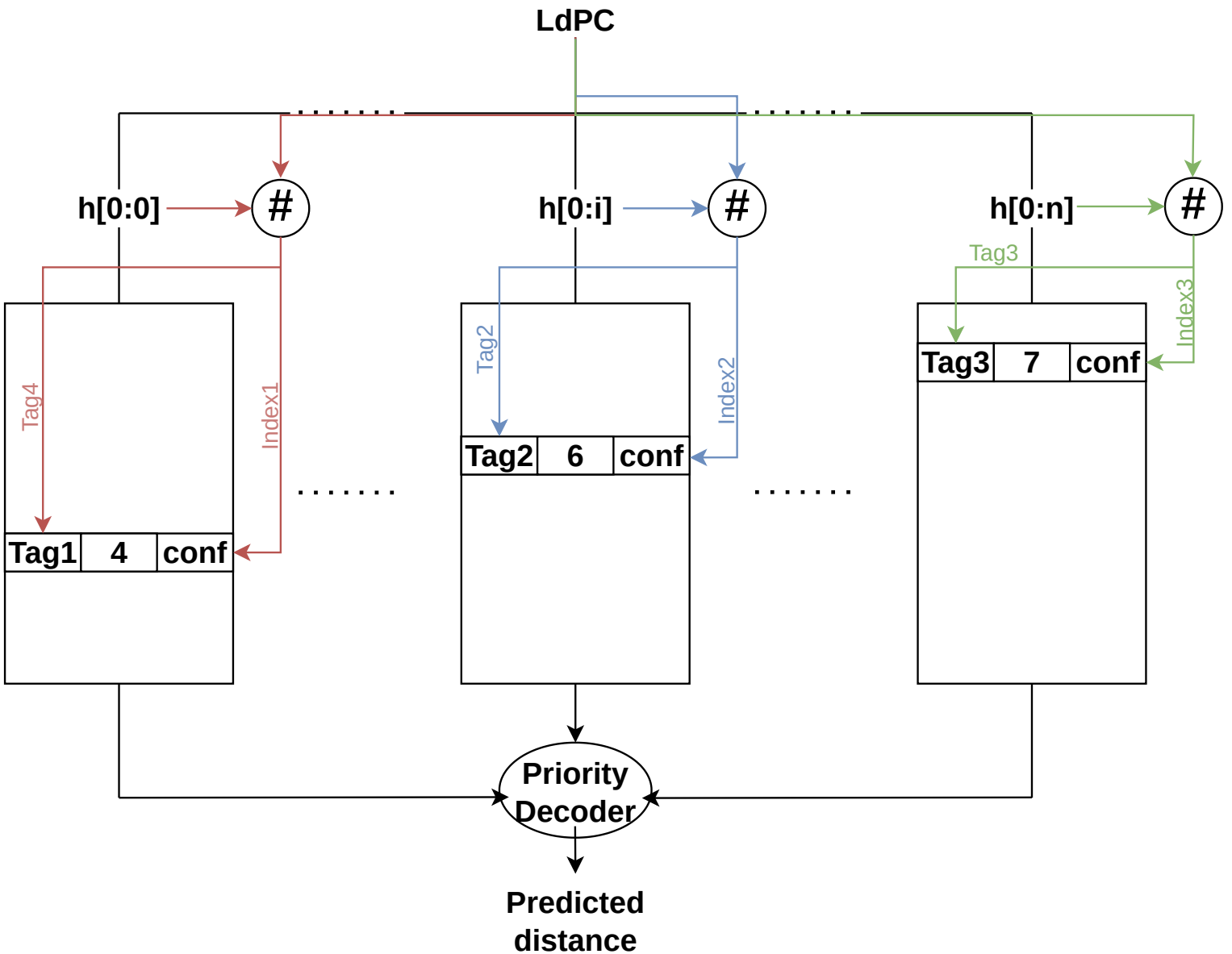
PHAST: UPDATE



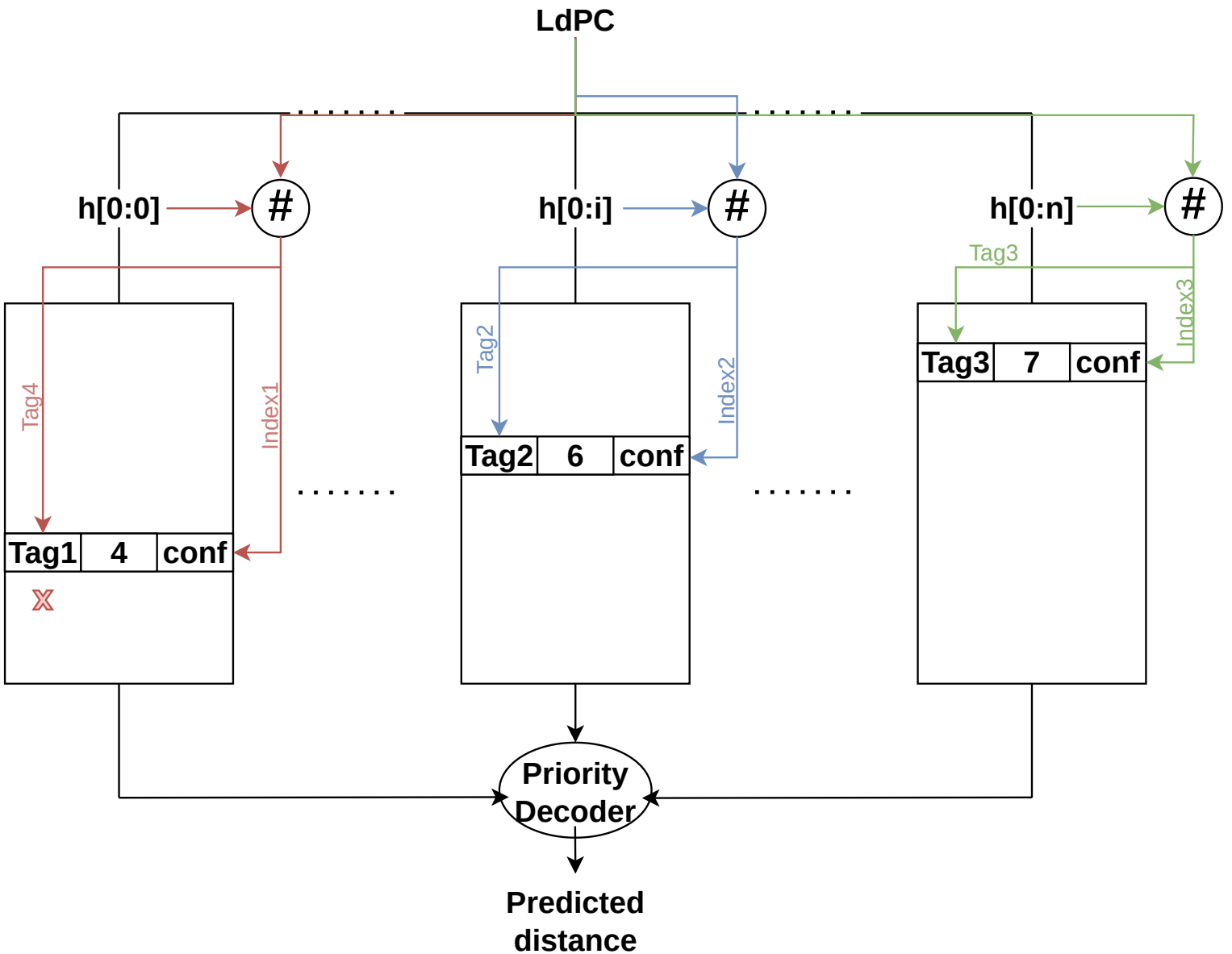
PHAST: PREDICTION



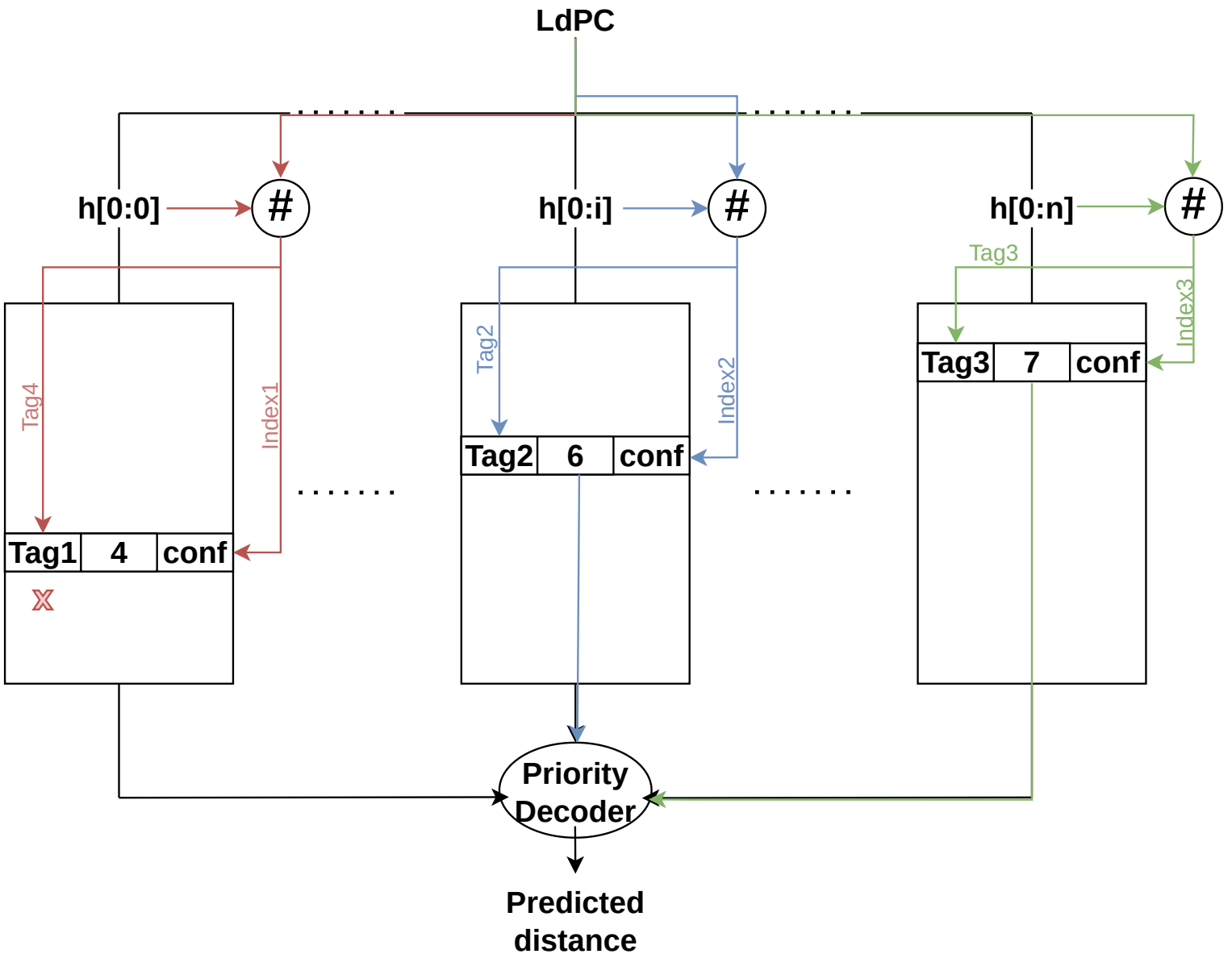
PHAST: PREDICTION



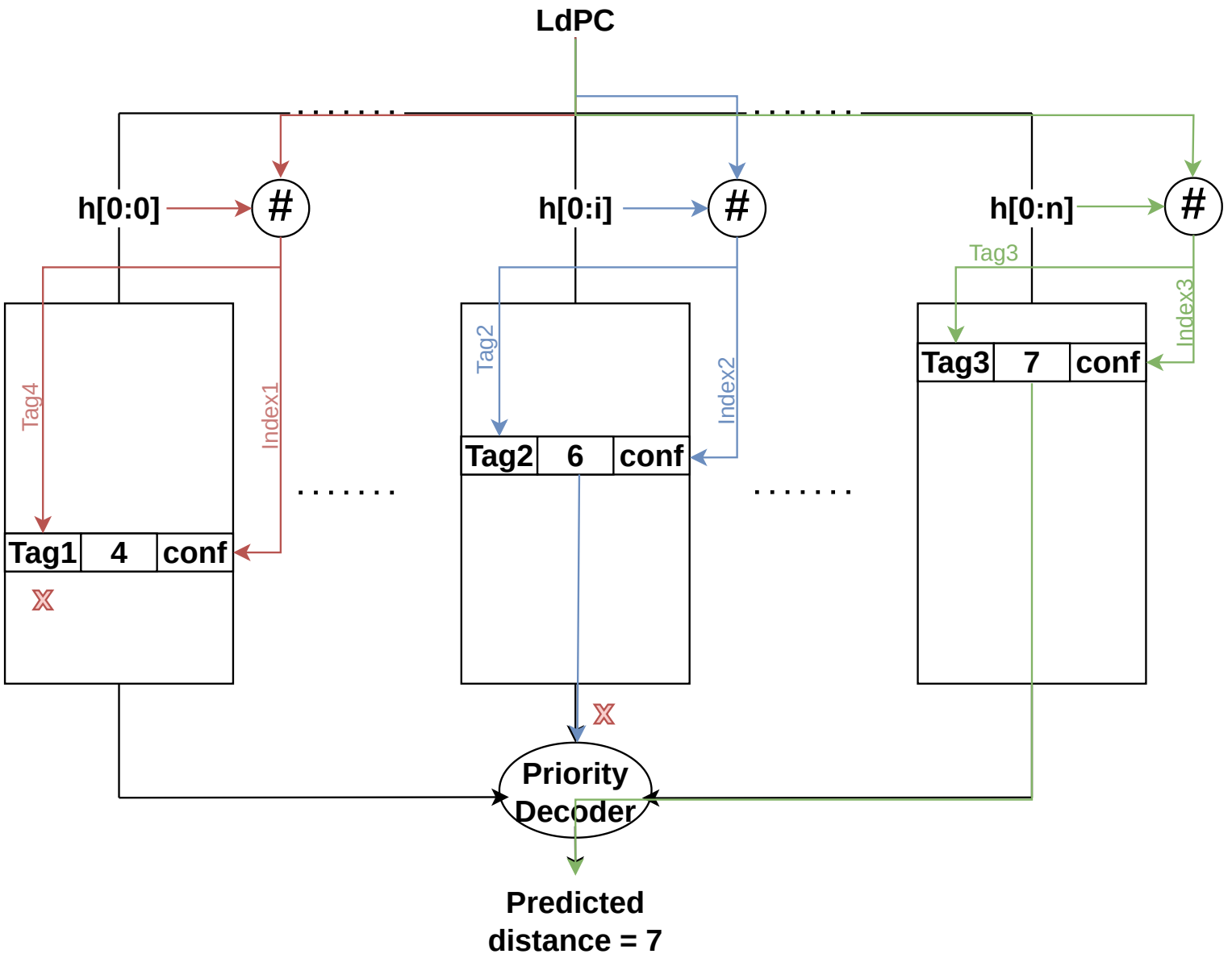
PHAST: PREDICTION



PHAST: PREDICTION



PHAST: PREDICTION



PHAST: IMPLEMENTATION DETAILS

- Confidence counter:
  - 4-bit saturated counter
  - On commit, the prediction is checked
    - Correct $\rightarrow$ Saturate confidence counter
    - False dependence $\rightarrow$ Decrease it by 1

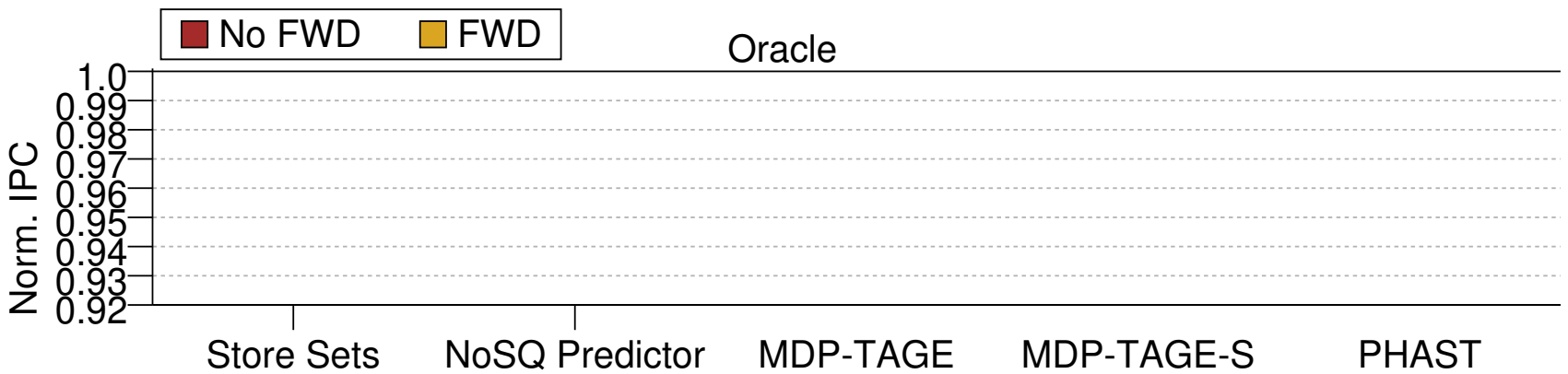
PHAST: Implementation details

- Confidence counter:
  - 4-bit saturated counter
  - On commit, the prediction is checked
    - Correct $\rightarrow$ Saturate confidence counter
    - False dependence $\rightarrow$ Decrease it by 1

- Number and configuration of tables:
  - 8 tables with geometric-like sequence of history lengths $\Rightarrow$ {0, 2, 4, 6, 8, 12, 16, 32}
  - Histories not covered are truncated
  - Each table is 4-way associative

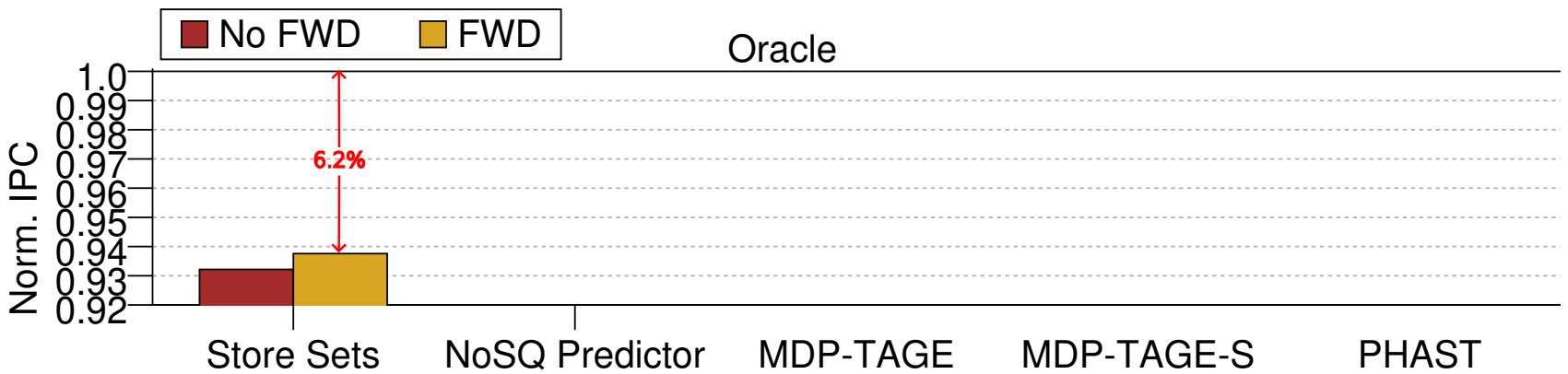
RESULTS: FORWARDING FILTERING

IDENTIFYING PRECISELY THE CONFLICTING STORE



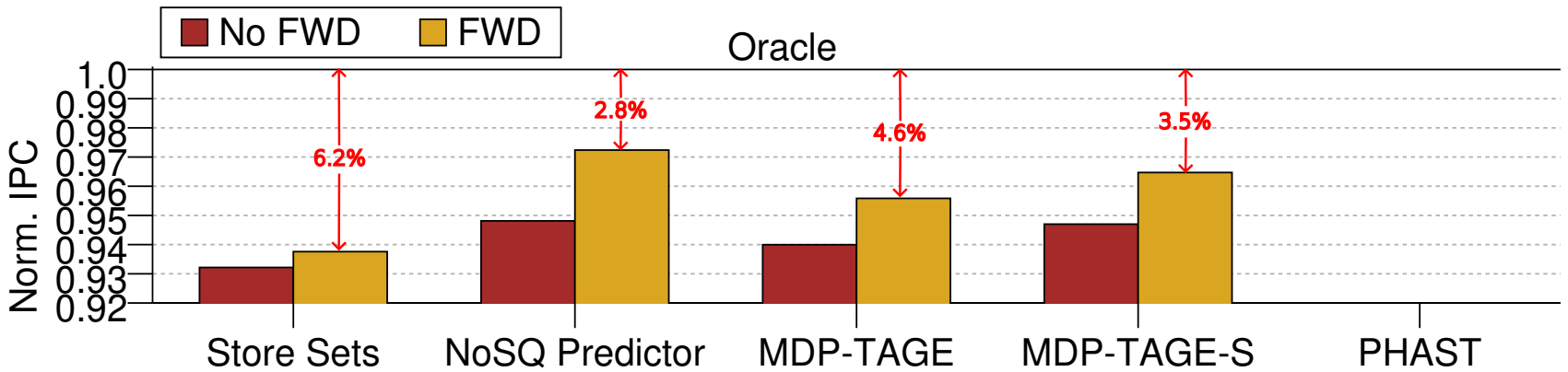
Results: Forwarding filtering

Identifying precisely the conflicting store



RESULTS: FORWARDING FILTERING

IDENTIFYING PRECISELY THE CONFLICTING STORE



Results: Forwarding filtering

Identifying precisely the conflicting store