HIGH-PERFORMANCE TIMELY PREFETCHING

Alberto Ros

University of Murcia, Spain

Dec 13, 2023

About me

1. **Academia**
   - PhD from the University of Murcia (UMU) in 2009
   - Postdoctoral positions at
     - Technical University of Valencia, Spain (2009-2011)
     - Uppsala University, Sweden (2011-2012)
   - Faculty at UMU since 2012
2. **Research**
   - Researcher in the Computer Architecture and Parallel Systems (CAPS) group
   - Since 2018 running an European Research Council (ERC) Consolidator Grant to improve the performance of multicore architectures

Improving next-generation architectures

1. Improving multi-core architectures
   - Fast communication among cores
     - $\rightarrow$ Efficicent cache coherence
   - Increasing core count
     - $\rightarrow$ Scalable cache coherence

IMPROVING NEXT-GENERATION ARCHITECTURES

1. Improving multi-core architectures
   - Fast communication among cores
     - $\rightarrow$ Efficicent cache coherence
   - Increasing core count
     - $\rightarrow$ Scalable cache coherence
2. Easy and efficient parallel programming
   - Memory consistency models and hardware implementation
   - Hardware transactional memory

Improving next-generation architectures

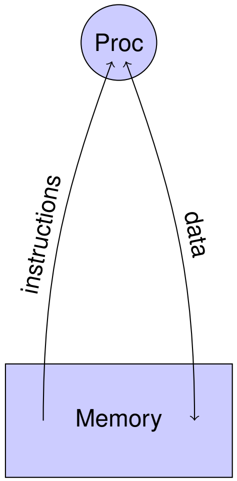
1. Improving multi-core architectures
   - Fast communication among cores
     - $\rightarrow$ Efficicent cache coherence
   - Increasing core count
     - $\rightarrow$ Scalable cache coherence
2. Easy and efficient parallel programming
   - Memory consistency models and hardware implementation
   - Hardware transactional memory
3. Software-hardware co-design
   - Help hardware with compile-time information

Improving next-generation architectures

1. Improving multi-core architectures
   - Fast communication among cores
     - → Efficicent cache coherence
   - Increasing core count
     - → Scalable cache coherence
2. Easy and efficient parallel programming
   - Memory consistency models and hardware implementation
   - Hardware transactional memory
3. Software-hardware co-design
   - Help hardware with compile-time information
4. Single-thread performance is still fundamental
   - Processor design and prediction mechanisms
   - Prefetching: Feed cores with enough instructions and data

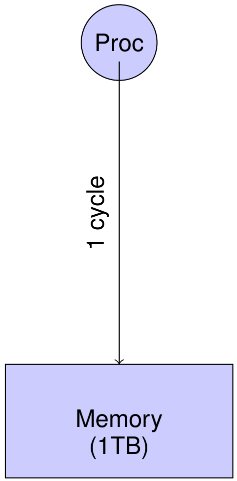
PREFETCHING

- Processors need to access instructions and data from memory



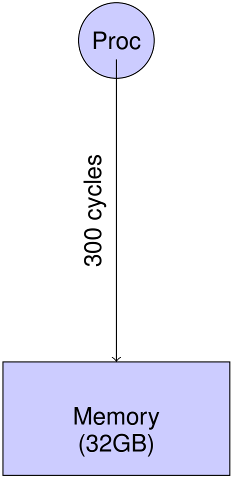
PREFETCHING

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



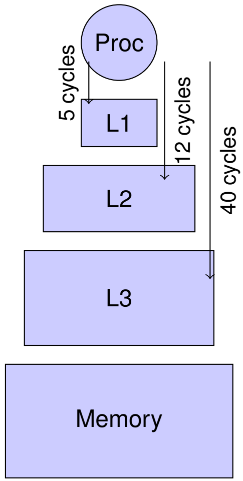
PREFETCHING

- 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



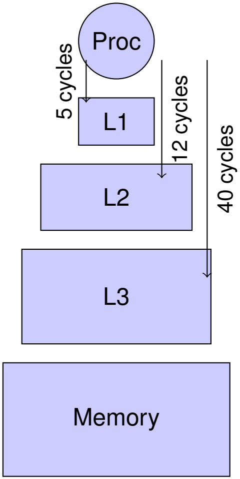
PREFETCHING

- 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



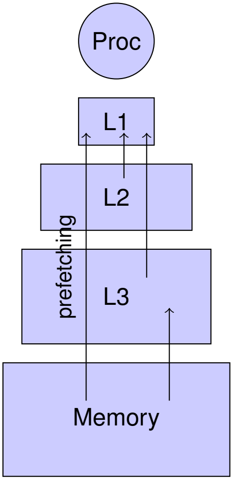
PREFETCHING

- 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
  - → *Predict* which memory addresses will be accessed by the processor and fetch them before the processor requests them



Instruction and data prefetching

- Prefetching is very different for instructions and data

INSTRUCTION AND DATA PREFETCHING

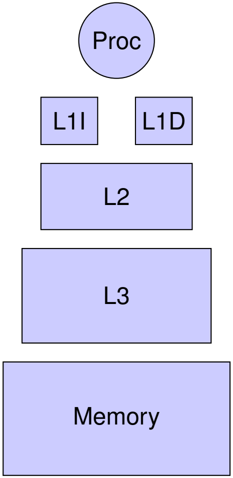
- Prefetching is very different for instructions and data
- Instructions
  - → Accesses to contiguous data blocks
  - → Branches, function calls, etc break contiguity

INSTRUCTION AND DATA PREFETCHING

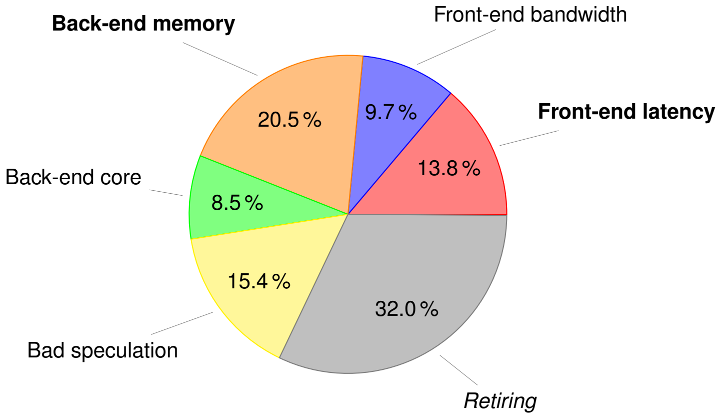
- Prefetching is very different for instructions and data
- Instructions
  - → Accesses to contiguous data blocks
  - → Branches, function calls, etc break contiguity
- Data
  - → Predictable when iterating regular data structures
  - → Hard to predict when using pointers or indirections

INSTRUCTION AND DATA PREFETCHING

- Prefetching is very different for instructions and data
- Instructions
  - → Accesses to contiguous data blocks
  - → Branches, function calls, etc break contiguity
- Data
  - → Predictable when iterating regular data structures
  - → Hard to predict when using pointers or indirections
- The L1 caches are separate for instructions and data
  - → Eases separating instruction and data prefetching, at least at the L1 cache



IMPORTANCE OF PREFETCHING



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

Importance of prefetching

Front-end latency (13.8%)

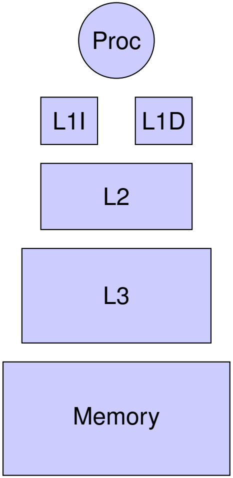
- Dominated by instruction cache (L1I) misses
  - Server and cloud apps getting larger, far from fitting in L1I
  - Hiting in the L2 or L3
- Latency more important than bandwidth
- Critical as processors need to keep the pipeline full

Back-end memory (20.5%)

- Due to data cache (L1D) misses
  - Many of them reaching main memory
- Cause significant stalls
  and late detection of BAD SPECULATION (15.4%)

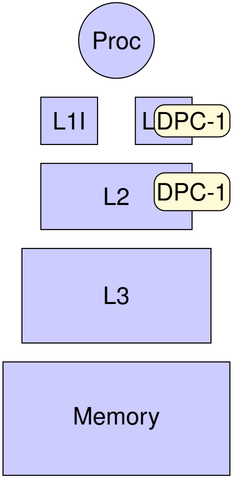
PREFETCHING CHAMPIONSHIPS

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



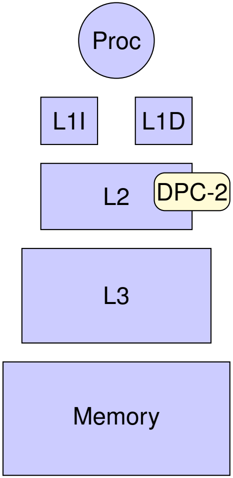
Prefetching Championships

- 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)



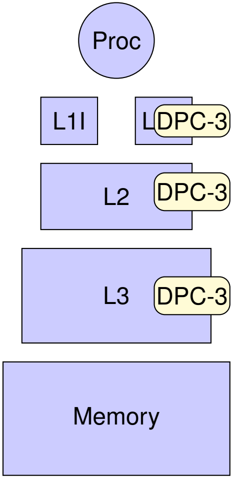
PREFETCHING CHAMPIONSHIPS

- 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)



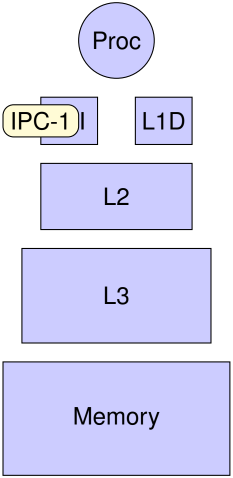
PREFETCHING CHAMPIONSHIPS

- 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)



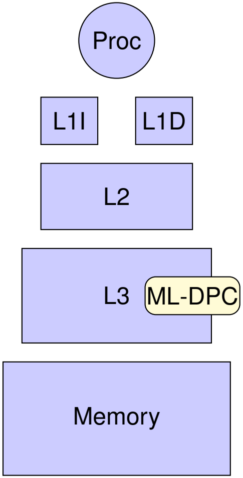
PREFETCHING CHAMPIONSHIPS

- 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)

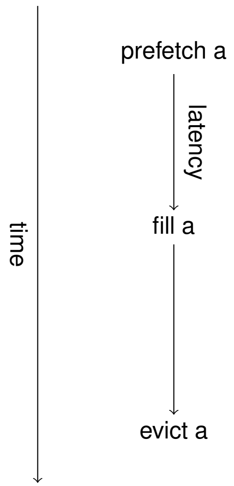


Prefetching Championships

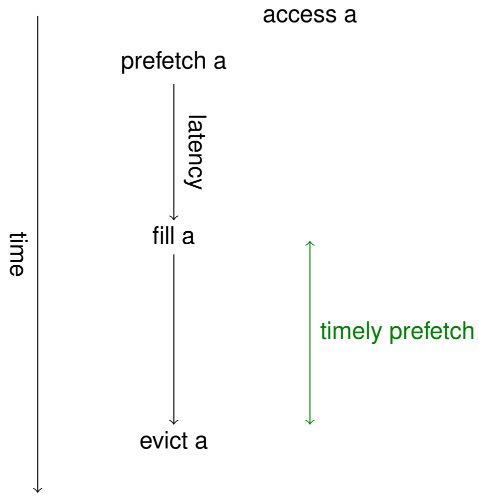
- Commonly promoted/sponsored by industry: e.g., Intel, Google
- All contestants following the same rules and criteria
- Championships
    - 2009 – 1st Data Prefetching Championship (DPC-1)
    - 2015 – 2nd Data Prefetching Championship (DPC-2)
    - 2019 – 3rd Data Prefetching Championship (DPC-3)
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



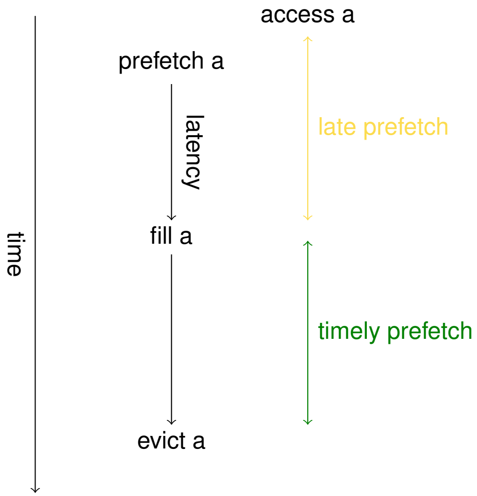
WHICH AND WHEN (TIMELINESS) TO PREFETCH



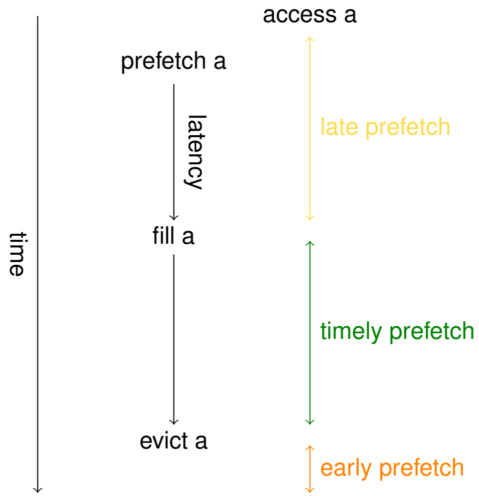
Which and when (timeliness) to prefetch



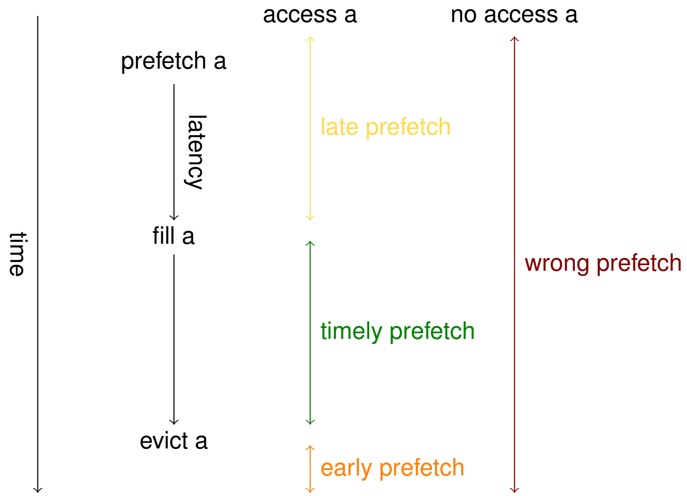
Which and when (timeliness) to prefetch



WHICH AND WHEN (TIMELINESS) TO PREFETCH



Which and when (timeliness) to prefetch



PREFETCHING METRICS

- COVERAGE
  - Fraction of cache misses covered by the prefetch

$$Coverage = \frac{timely\_prefs}{timely\_prefs + cache\_misses}$$

  - Indicator of performance benefits

PREFETCHING METRICS

- COVERAGE
  - Fraction of cache misses covered by the prefetch

$$Coverage = \frac{timely\_prefs}{timely\_prefs + cache\_misses}$$

  - Indicator of performance benefits
- ACCURACY
  - Fraction of useful prefetches

$$Accuracy = \frac{timely\_prefs + late\_prefs}{prefs\_to\_next\_cache\_level}$$

  - Indicator of energy efficiency

Timeliness is the key property

- Two prefetchers with a strong focus on timeliness

TIMELINESS IS THE KEY PROPERTY

- Two prefetchers with a strong focus on timeliness

1. The ENTANGLING Intruction Prefetcher
    - An L1I prefetcher
    - Winner of the IPC-1
    - Follow up papers published at ISCA'21 and IEEE TC'24

TIMELINESS IS THE KEY PROPERTY

- Two prefetchers with a strong focus on timeliness

1. The ENTANGLING Intruction Prefetcher
    - An L1I prefetcher
    - Winner of the IPC-1
    - Follow up papers published at ISCA'21 and IEEE TC'24
2. The BERTI Data Prefetcher
    - An L1D prefetcher published at MICRO'22
    - An early version participated in the DPC-3
    - A follow up version won the ML-DPC (without using ML)

TIMELINESS IS THE KEY PROPERTY

- Two prefetchers with a strong focus on timeliness

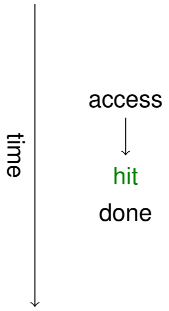
1. The ENTANGLING Intruction Prefetcher
   - An L1I prefetcher
   - Winner of the IPC-1
   - Follow up papers published at ISCA'21 and IEEE TC'24
2. The BERTI Data Prefetcher
   - An L1D prefetcher published at MICRO'22
   - An early version participated in the DPC-3
   - A follow up version won the ML-DPC (without using ML)

THE ENTANGLING INSTRUCTION PREFETCHER

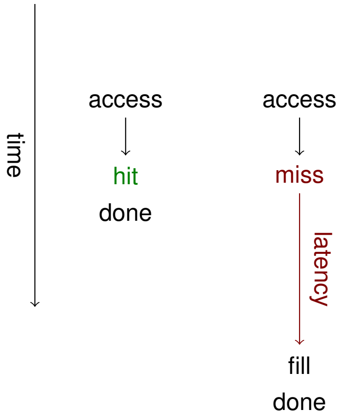
Alberto Ros   Alexandra Jimborean

University of Murcia, Spain

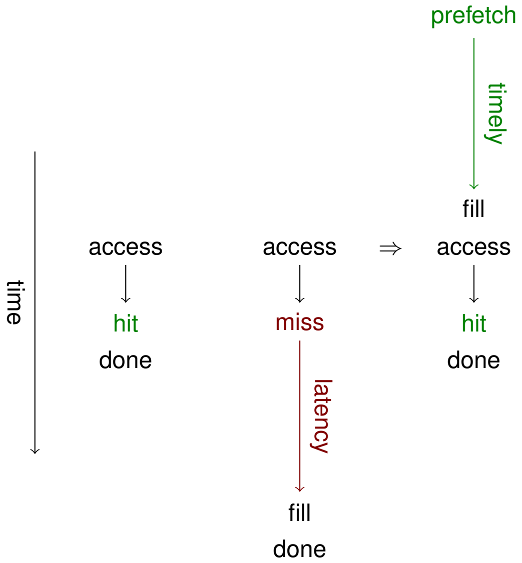
Motivation: Timeliness



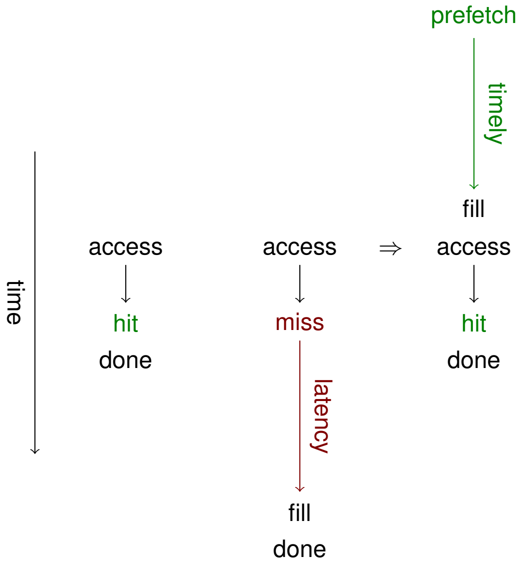
Motivation: Timeliness



Motivation: Timeliness



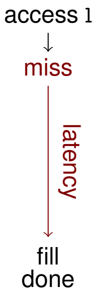
MOTIVATION: TIMELINESS



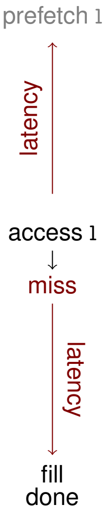
Timely prefetches
for all misses:
Coverage 100%

And only for misses:
Accuracy 100%

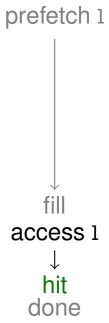
CONCEPT OF ENTANGLED ACCESSES



Concept of entangled accesses



Concept of entangled accesses



CONCEPT OF ENTANGLED ACCESSES

- access `a`
- access `b`

prefetch `l`

- access `c`
- access `d`
- access `e`

fill

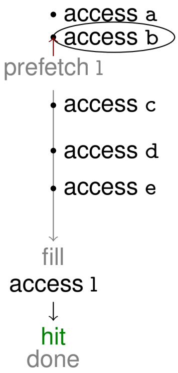
access `l`

↓

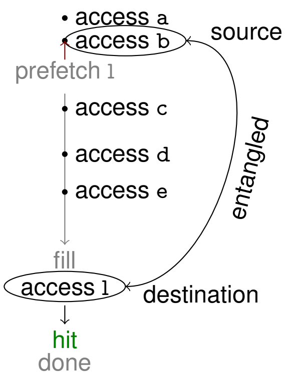
hit

done

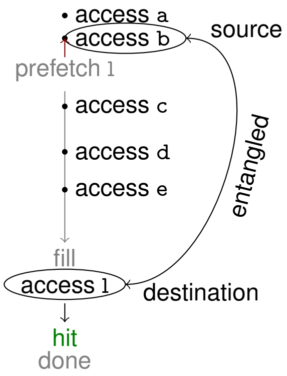
Concept of entangled accesses

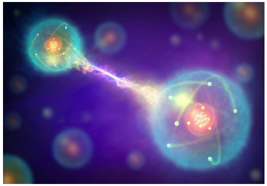


Concept of entangled accesses



CONCEPT OF ENTANGLED ACCESSES

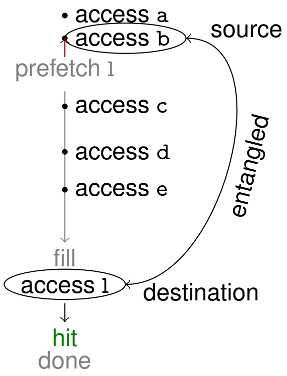


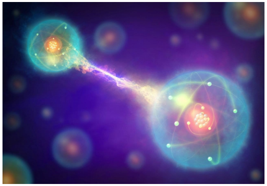


Quantum entanglement

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

CONCEPT OF ENTANGLED ACCESSES



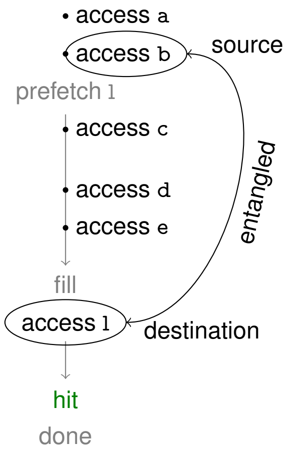


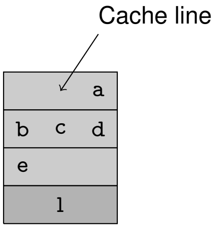
Quantum entanglement

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

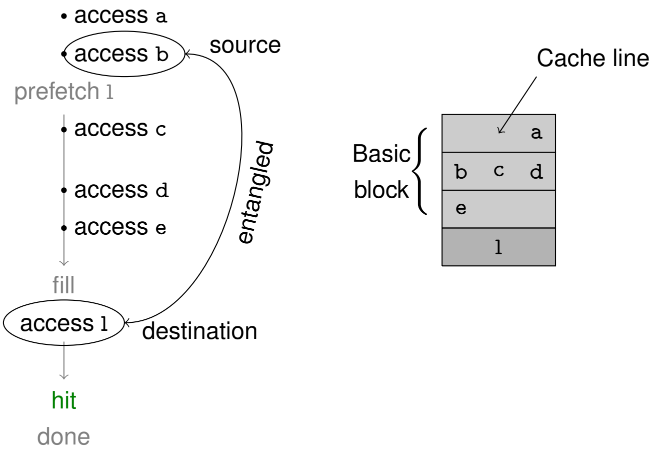
THE ENTANGLING PREFETCHER FOR INSTRUCTIONS

Entangling cache lines head of basic blocks

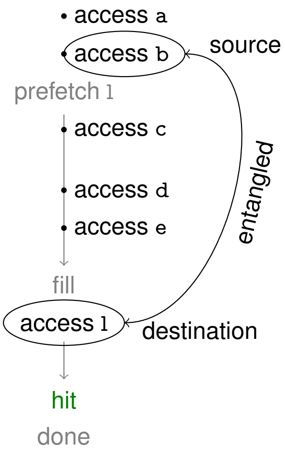


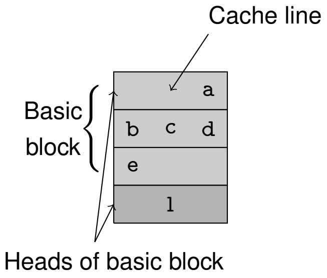


ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS

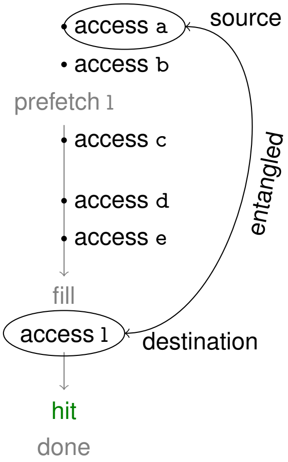


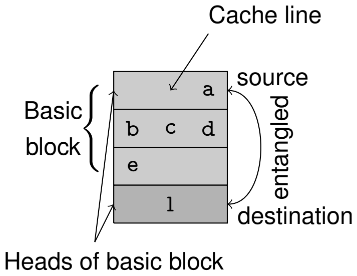
ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS



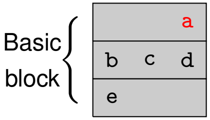


ENTANGLING CACHE LINES HEAD OF BASIC BLOCKS

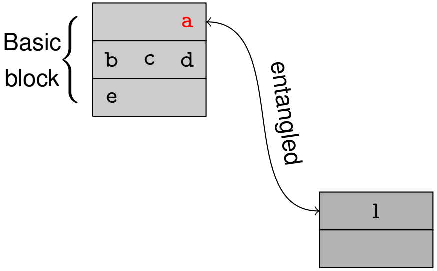




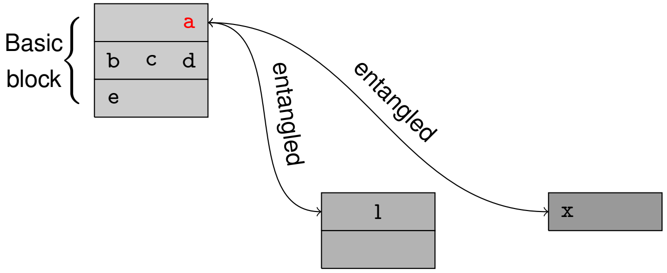
What to prefetch on an access to a?



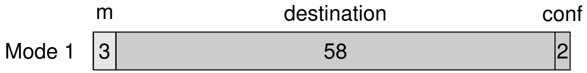
What to prefetch on an access to a?



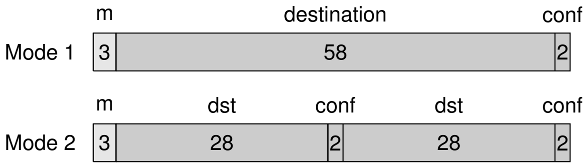
What to prefetch on an access to a?



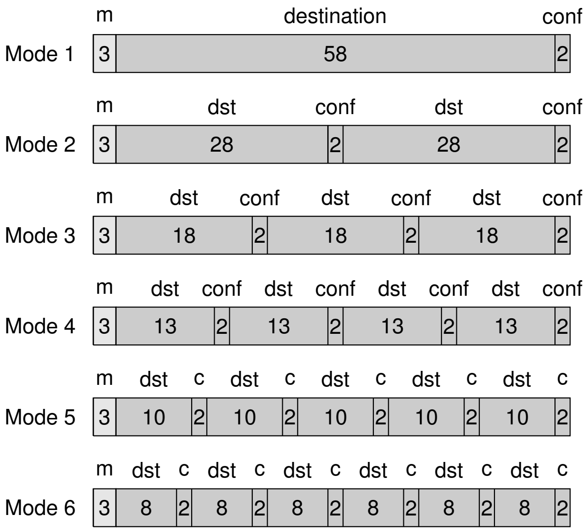
Compressing destinations



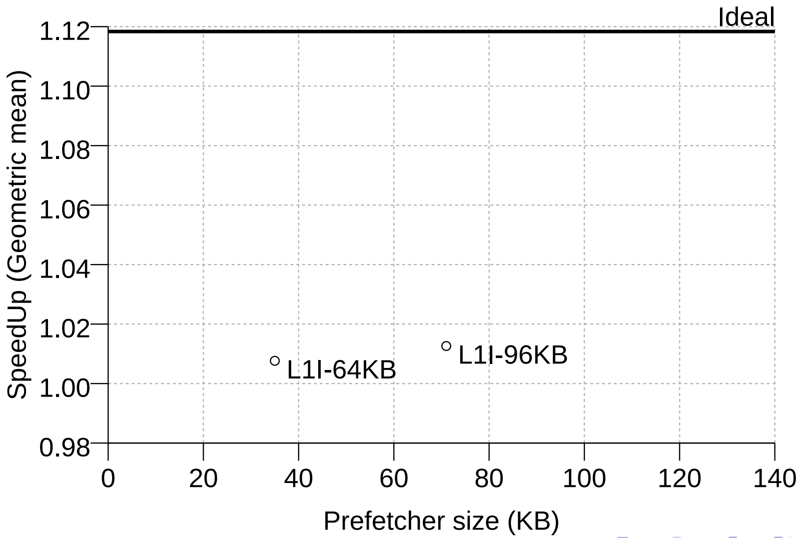
COMPRESSING DESTINATIONS



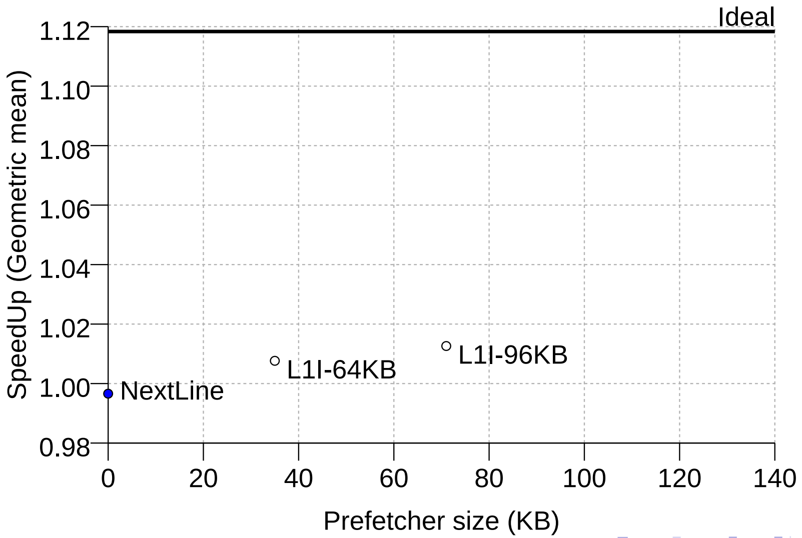
COMPRESSING DESTINATIONS



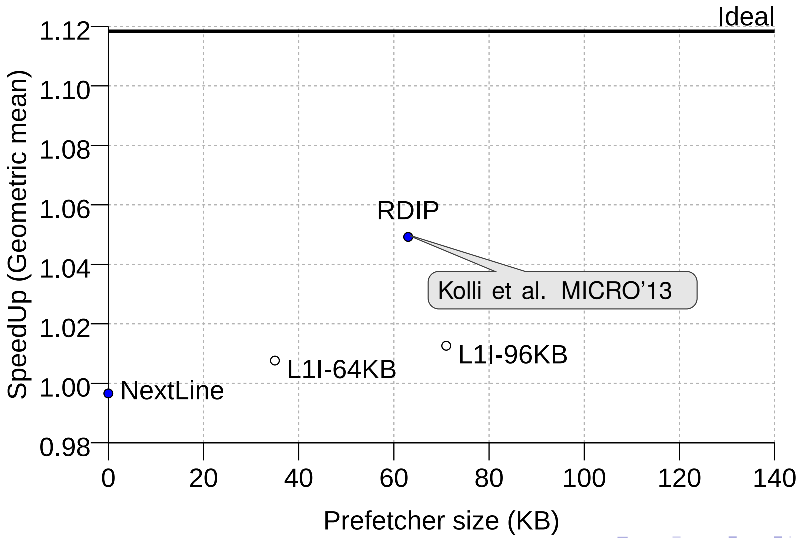
Results: IPC vs memory overhead



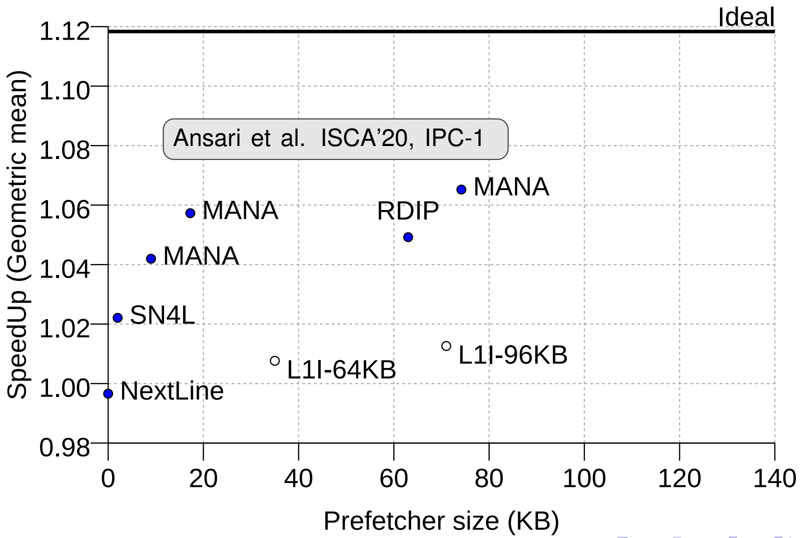
Results: IPC vs memory overhead



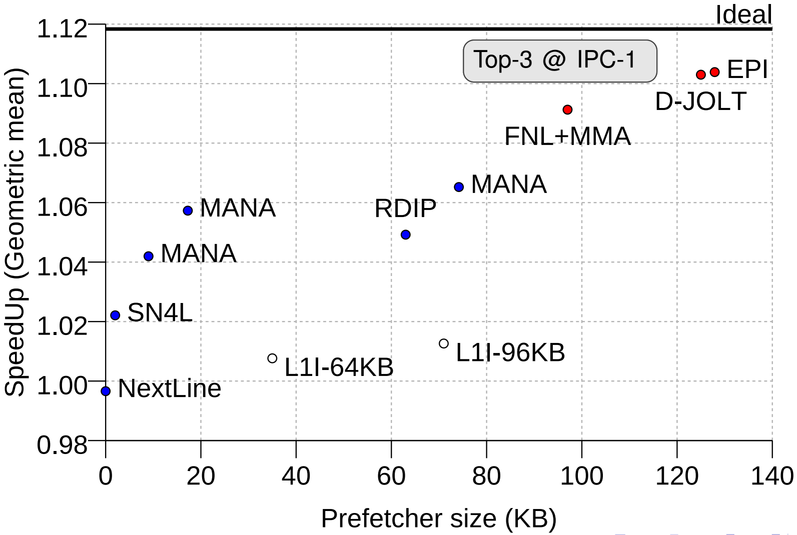
Results: IPC vs memory overhead



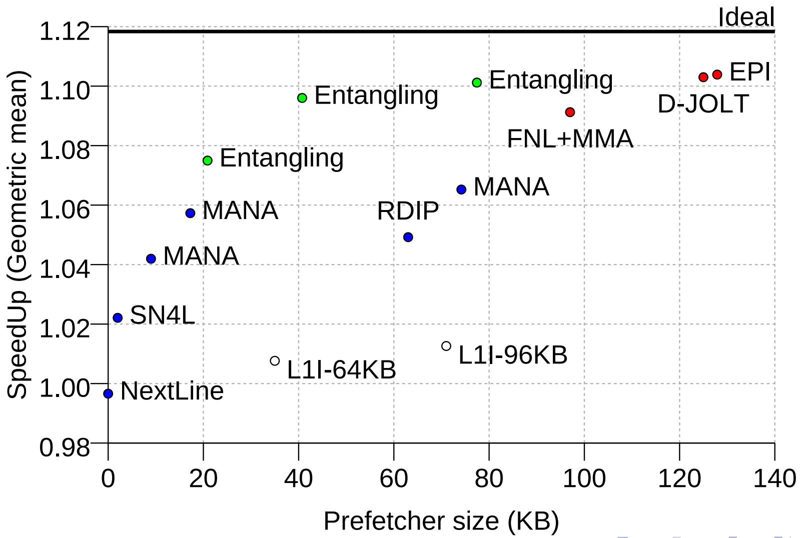
RESULTS: IPC VS MEMORY OVERHEAD



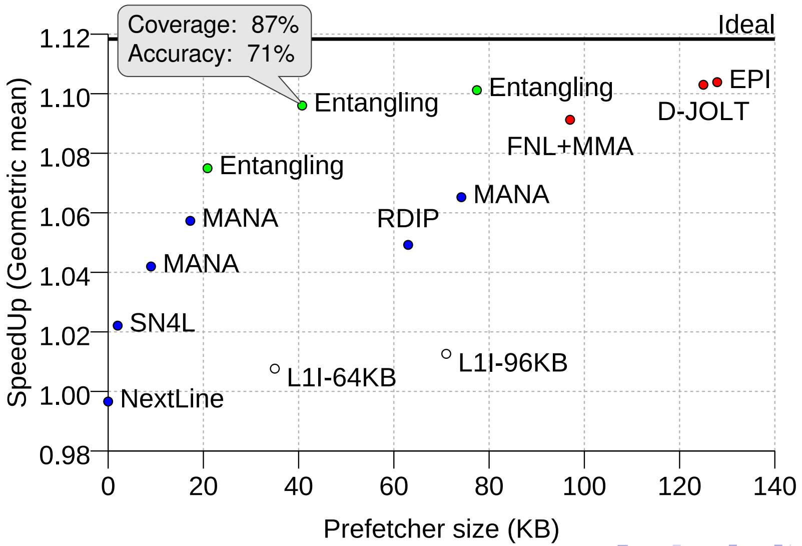
RESULTS: IPC VS MEMORY OVERHEAD



RESULTS: IPC VS MEMORY OVERHEAD



Results: IPC vs memory overhead



BERTI: AN ACCURATE LOCAL-DELTA DATA PREFETCHER

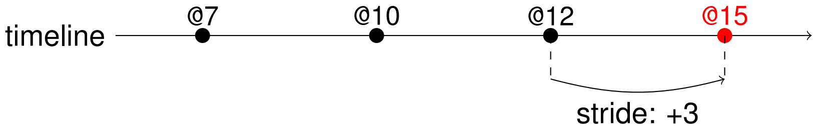
Agustín Navarro-Torres[1] Biswabandan Panda[2]
Jesús Alastruey-Benedé[3] Pablo Ibañez[3]
Víctor Viñals-Yúfera[3] Alberto Ros[1]

[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

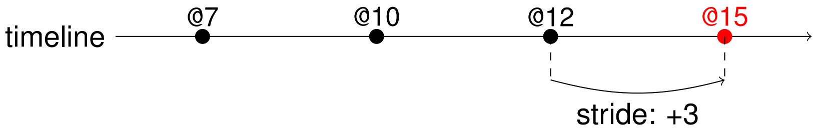
- An L1D prefetcher that orchestrates data prefetching from all cache levels
- Advantages of L1D prefetching
  - Training with all processor references
  - Using the Instruction Pointer (IP) information
  - Operating with virtual addresses: cross-page prefetching

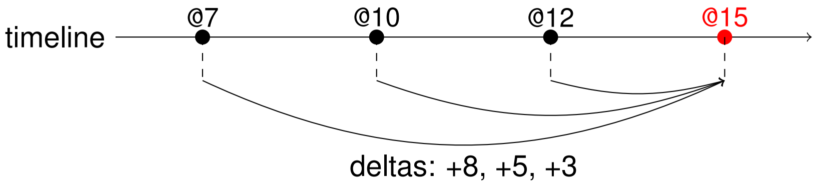
Definition of delta



BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

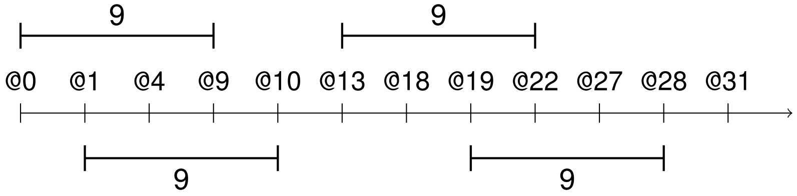
Definition of delta





BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

Stride: **+1, +3, +5**

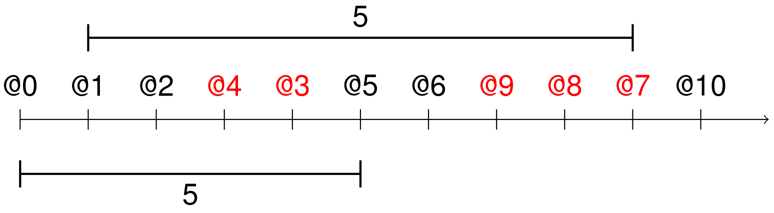


With which delta should I prefetch?

delta = 1 + 3 + 5 = 9 $\rightarrow$ **always hit**

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

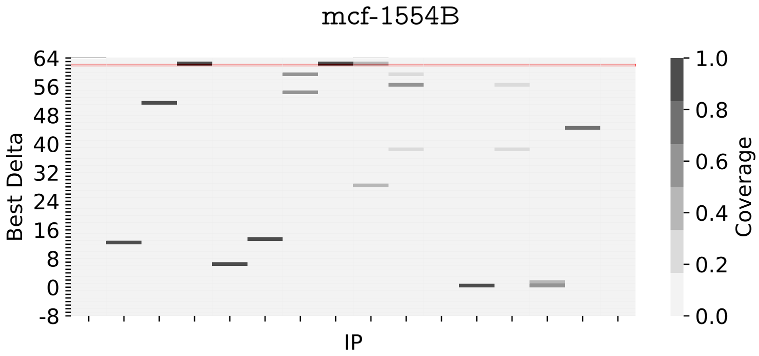
Addresses reordered by the out-of-order processor



Stride prefetch requires specific order

Berti can prefetch with delta $=$ 5, for example

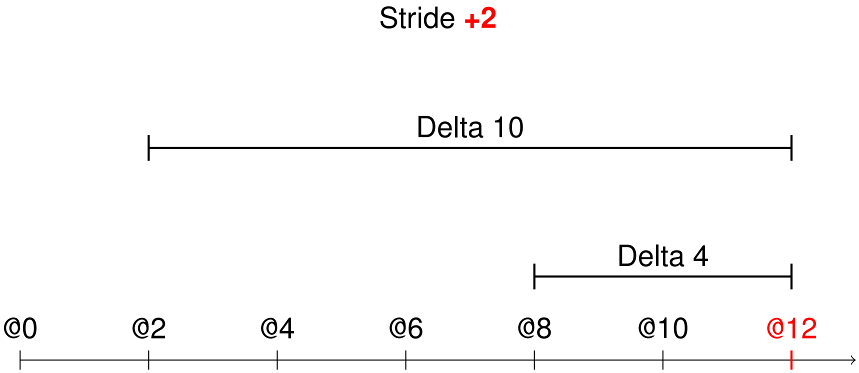
BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER



- **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)

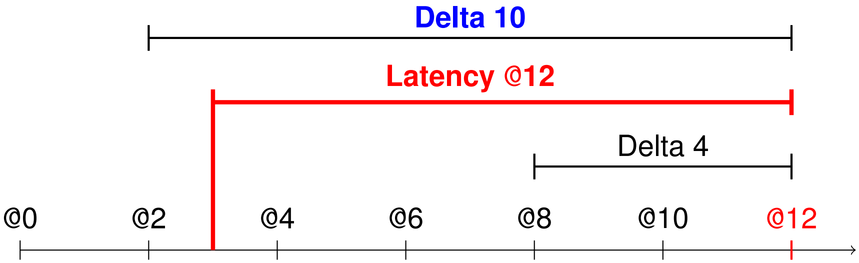
Berti: Accurate and timely local delta L1D prefetcher



How far in advance should I prefetch address 12?

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

Stride **+2**



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

Berti: Accurate and timely local delta L1D prefetcher

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

+10

History table

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

Table of deltas

IP	Delta	Coverage	Destination

Berti: Accurate and timely local delta L1D prefetcher

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%)**	

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

TRAINING

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

+10, +13

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%)**	

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

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%)	

BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

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

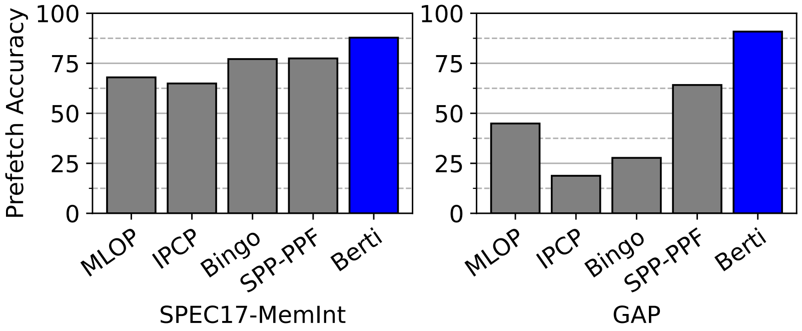
Table of deltas

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

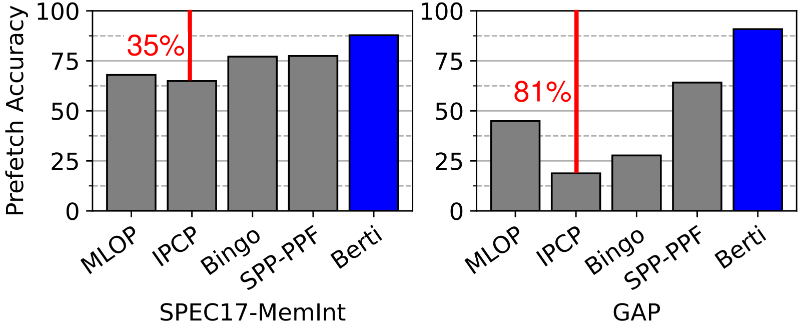
Coverage

> 65% $\rightarrow$ *L1D*
> 35% $\rightarrow$ *L2*

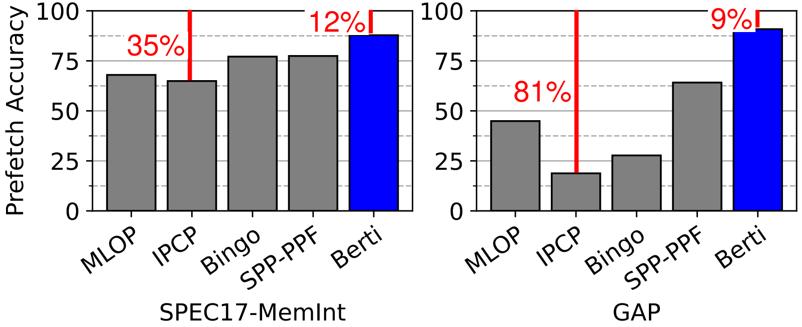
Berti: Accurate and timely local delta L1D prefetcher



BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER

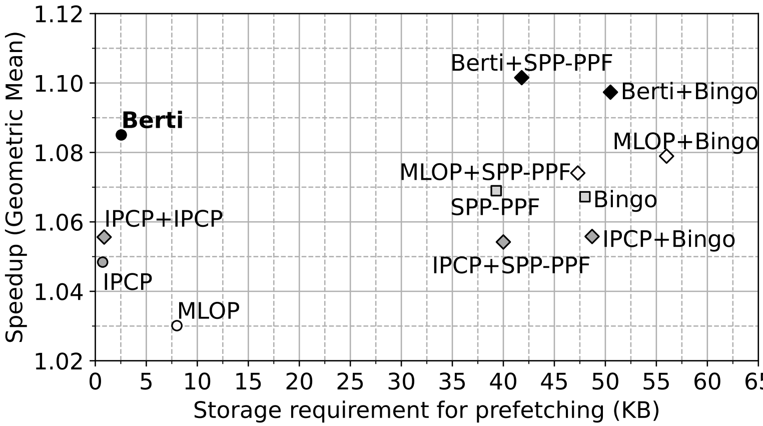


BERTI: ACCURATE AND TIMELY LOCAL DELTA L1D PREFETCHER



Improved accuracy reduces BW overhead and L1D pollution

Berti: Overall performance



TAKE AWAY MESSAGE

- Prefetching is fundamental techique for high-performance
- Both intructions and data are amenable to prefetching
- Ideally, it is desirable to prefetch to L1
- Fetch latency plays an important for timely prefetching

HIGH-PERFORMANCE TIMELY PREFETCHING

Alberto Ros

University of Murcia, Spain

Thank you!

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

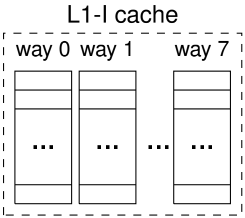
- Our contribution: An ENTANGLING prefetcher
  - ENTANGLING: adaptive correlation based on latency
  - Winner of the 1st Instruction Prefetching Championship
  - A cost-effective prefetcher
  - Prefetcher code is available[1]

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

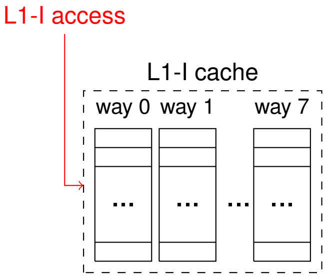
METHODOLOGY

- 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

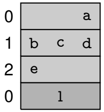
DESIGN OF THE ENTANGLING PREFETCHER

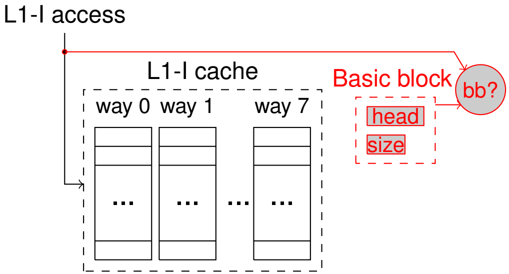


DESIGN OF THE ENTANGLING PREFETCHER

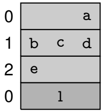


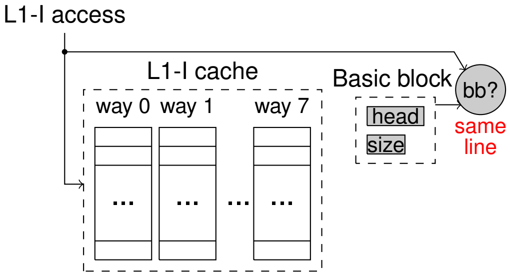
Design of the Entangling Prefetcher - Finding basic blocks



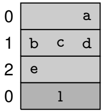


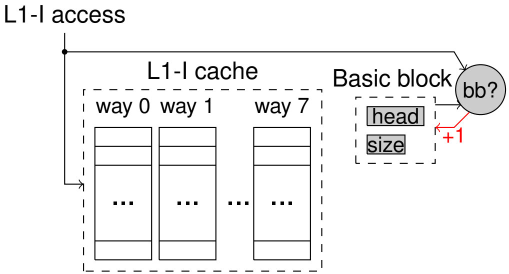
Design of the Entangling Prefetcher - Finding basic blocks



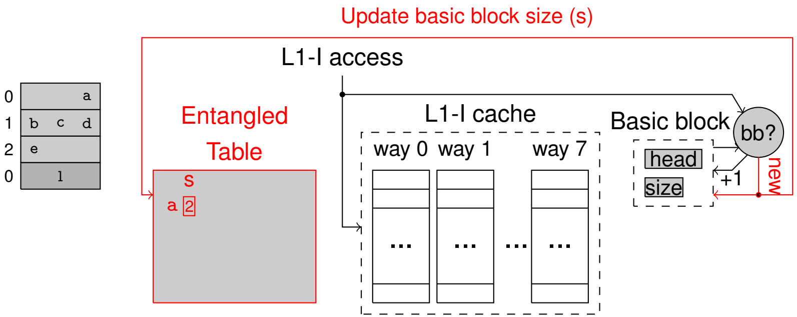


Design of the Entangling Prefetcher - Finding basic blocks





Design of the Entangling Prefetcher - Finding basic blocks



Design of the Entangling Prefetcher - Entangling cache lines

- access a
- access b

prefetch l

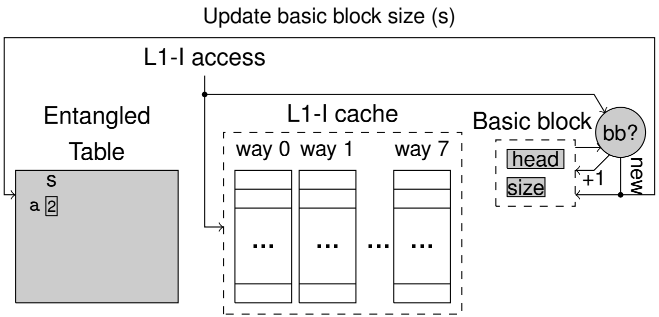
- access c
- access d
- access e

fill

access l

hit

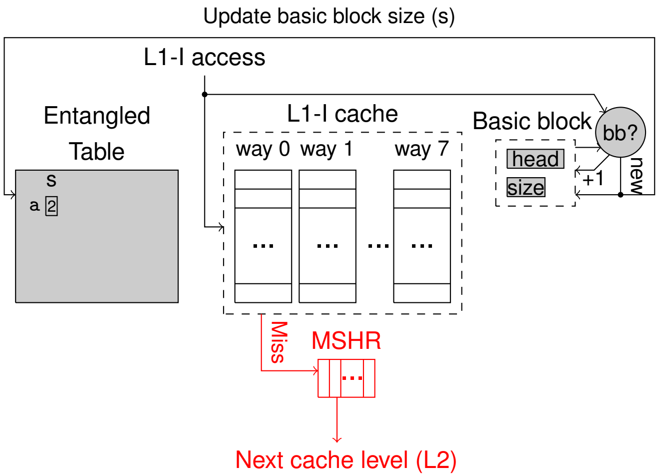
done



Design of the Entangling Prefetcher - Entangling cache lines

- access a
- access b
- access c
- access d
- access e

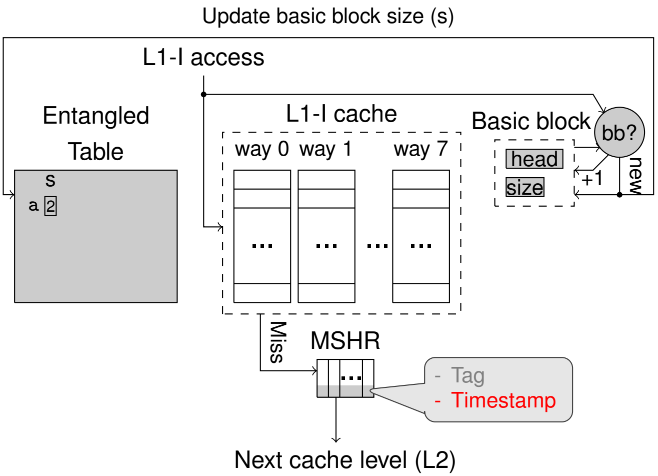
access l
↓
miss



Design of the Entangling Prefetcher - Entangling cache lines

- access a
- access b
- access c
- access d
- access e

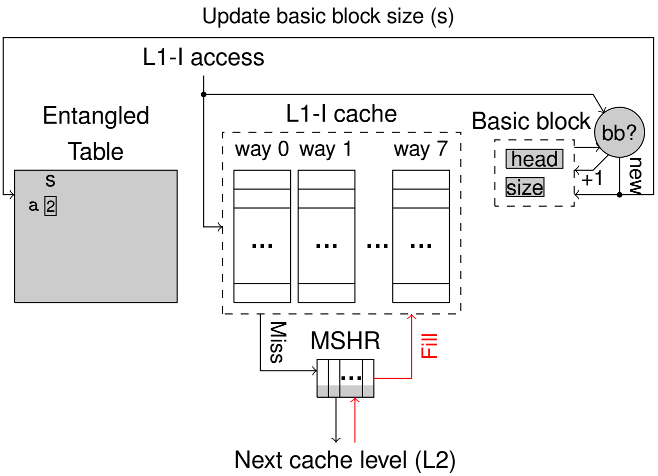
access l
↓
miss



Design of the Entangling Prefetcher - Entangling cache lines

- access `a`
- access `b`
- access `c`
- access `d`
- access `e`

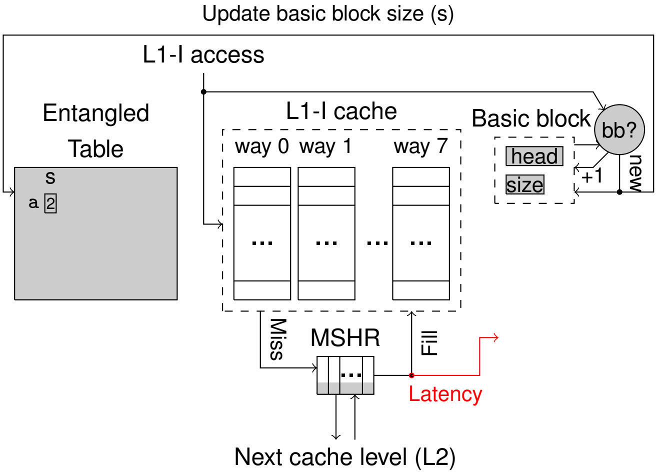
access `l`
↓
miss
latency
↓
fill
done



Design of the Entangling Prefetcher - Entangling cache lines

- access a
- access b
- access c
- access d
- access e

access l
↓
miss
latency
↓
fill
done



Design of the Entangling Prefetcher - Entangling cache lines

- access a
- access b
- access c
- access d
- access e

access l

↓

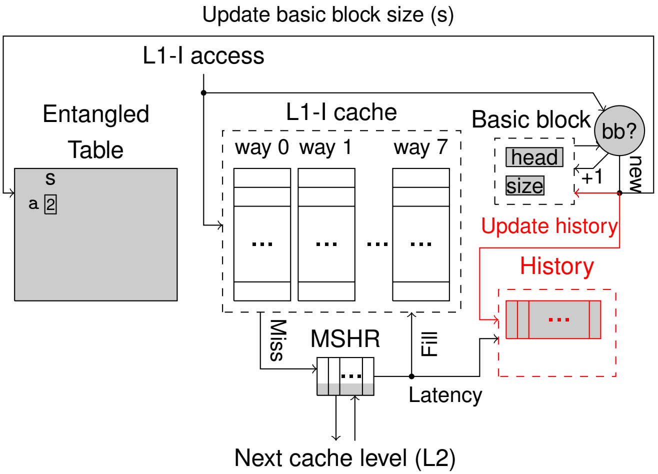
miss

latency

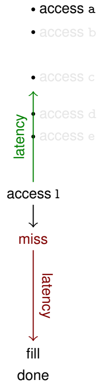
↓

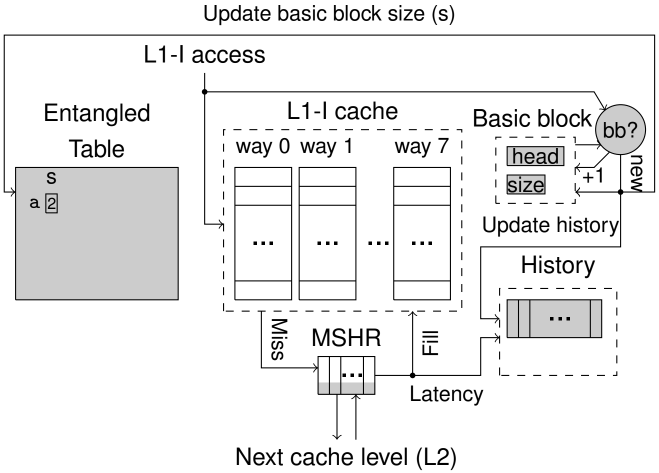
fill

done

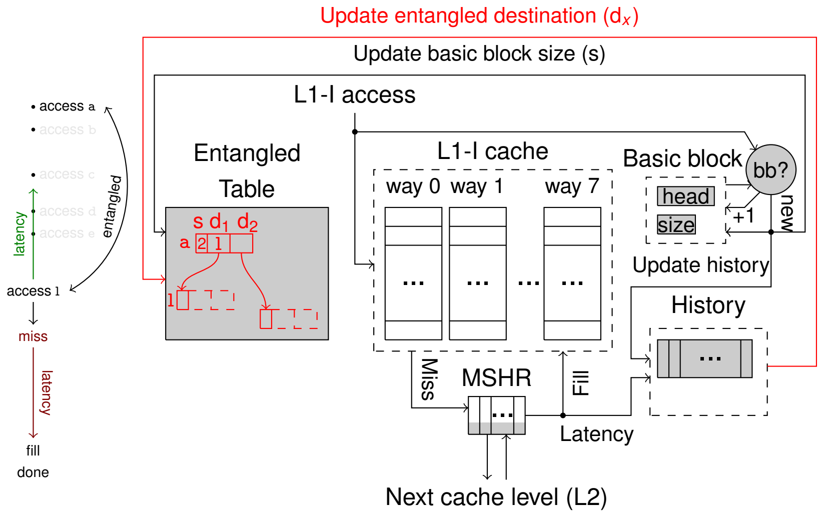


Design of the Entangling Prefetcher - Entangling cache lines

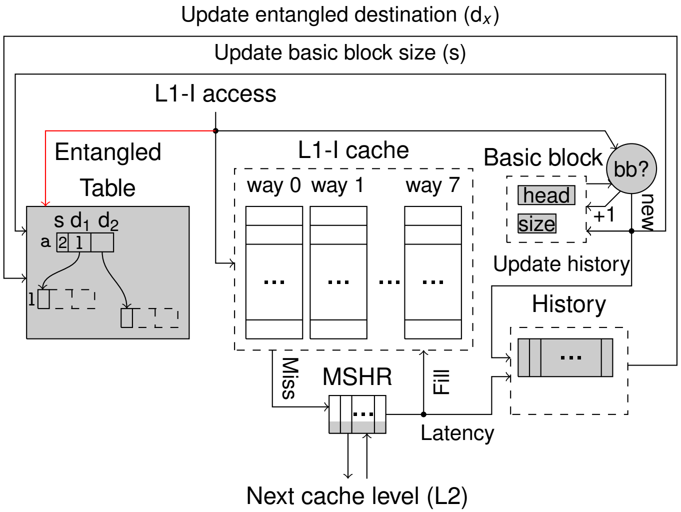




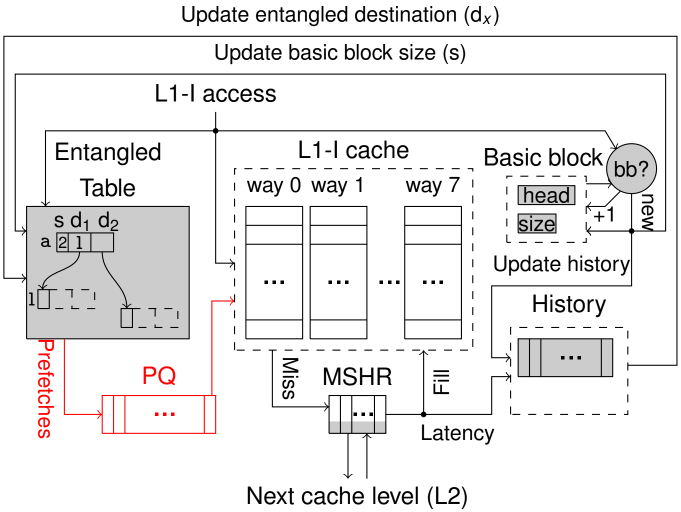
Design of the Entangling Prefetcher - Entangling cache lines



DESIGN OF THE ENTANGLING PREFETCHER - ISSUING PREFETCHES



Design of the Entangling Prefetcher - Issuing prefetches



Design of the Entangling Prefetcher - Fixing late prefetches

- access a

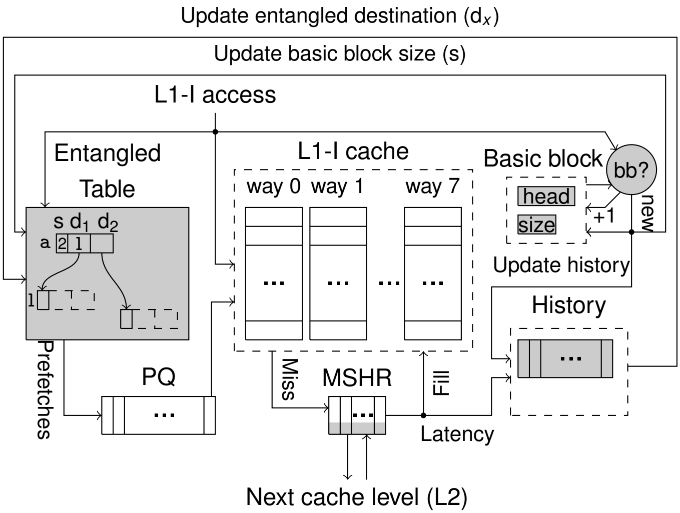
prefetch 1

access 1

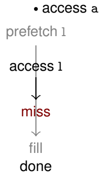
miss

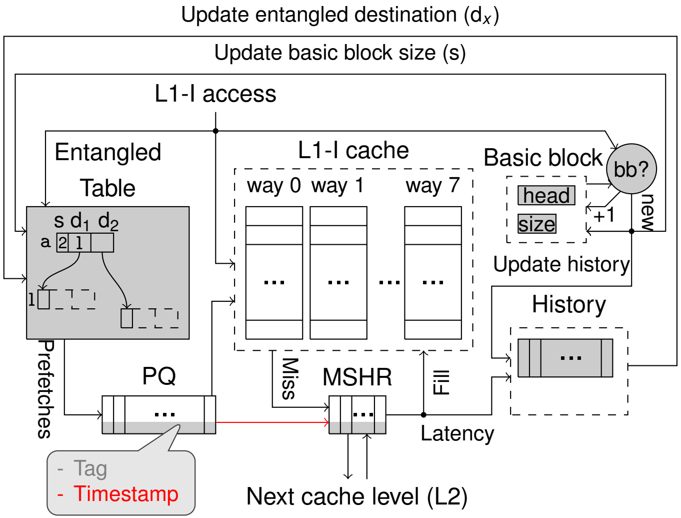
fill

done

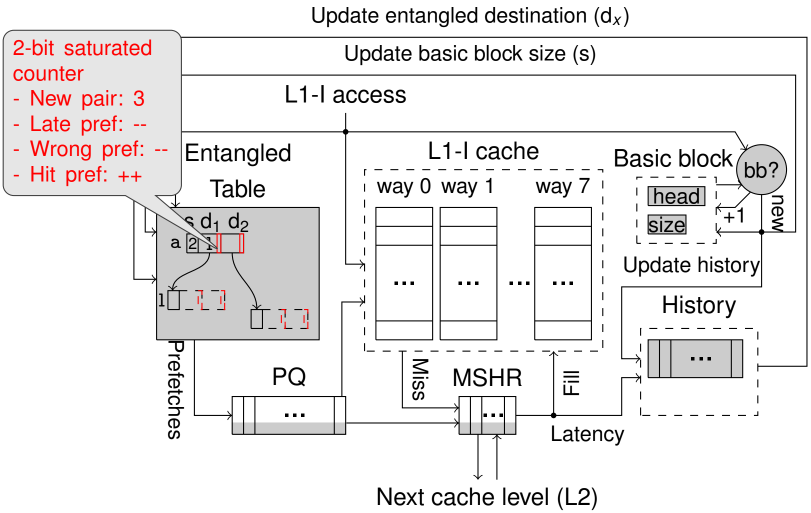


Design of the Entangling Prefetcher - Fixing late prefetches

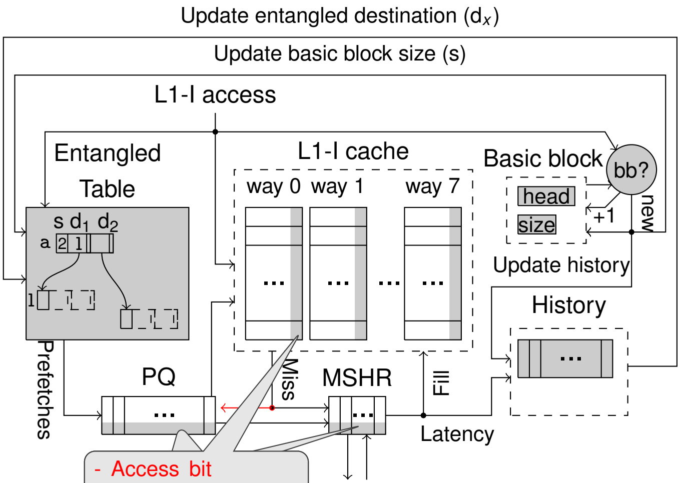




Design of the Entangling Prefetcher - Confidence for entangled pairs

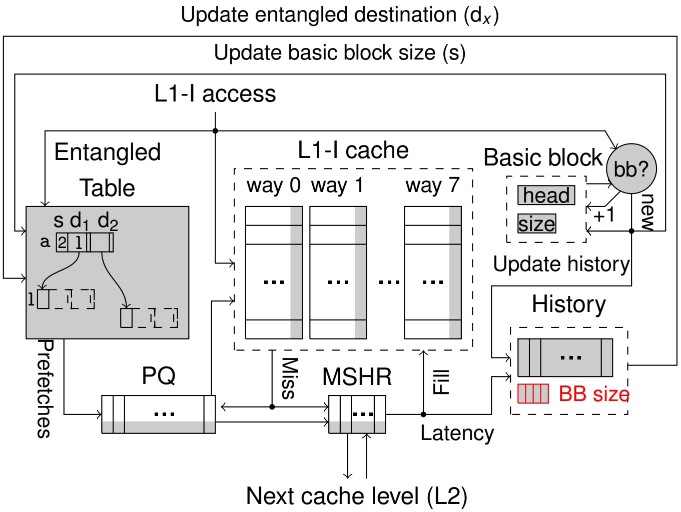


Design of the Entangling Prefetcher - Confidence for entangled pairs

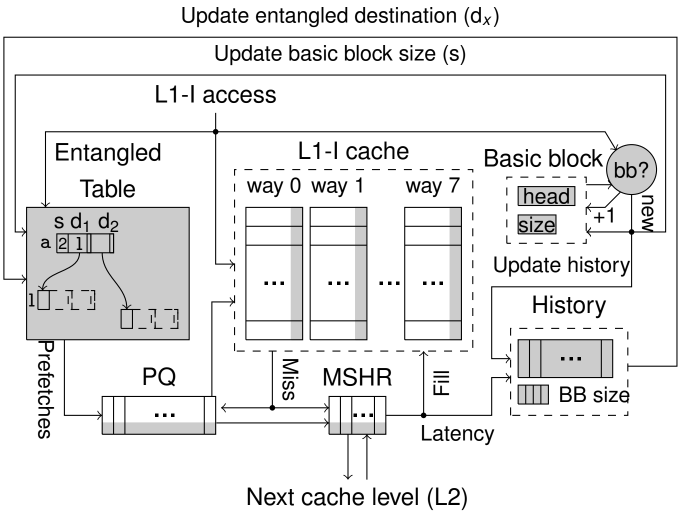


- Access bit
- Entangled-source

Design of the Entangling Prefetcher - Merging basic blocks



Design of the Entangling Prefetcher



Concluding remarks

- Timeliness as a key property
- Entangles heads of basic blocks to trigger timely prefetches
- Near ideal performance with just 40KB