HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM

The Key Problem

Long-latency off-chip load requests

The Key Problem

Long-latency off-chip load requests

Often **block instruction retirement** from Reorder Buffer (ROB)

The Key Problem

Long-latency off-chip load requests

Often **block instruction retirement** from Reorder Buffer (ROB)

Limits performance

Traditional Solutions

Traditional Solutions

1. Employ sophisticated **prefetchers**

Traditional Solutions

1. Employ sophisticated **prefetchers**
2. **Increase size** of on-chip caches

Key Observations

Observation 1

Nearly **50%** of the off-chip requests
in a no-prefetching system
still go to the main memory
even in presence of state-of-the-art prefetcher

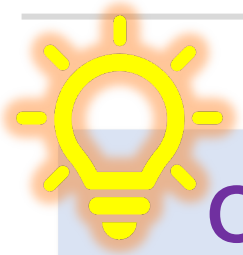
Key Observations

Observation 1

Nearly **50%** of the off-chip requests
in a no-prefetching system
still go to the main memory
even in presence of state-of-the-art prefetcher

70% of these off-chip loads **block the ROB**

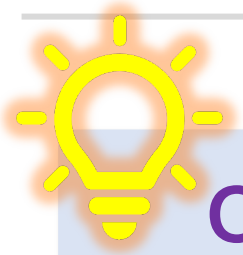
Key Observations



Observation 2

On-chip cache access latency significantly contributes to total off-chip load latency

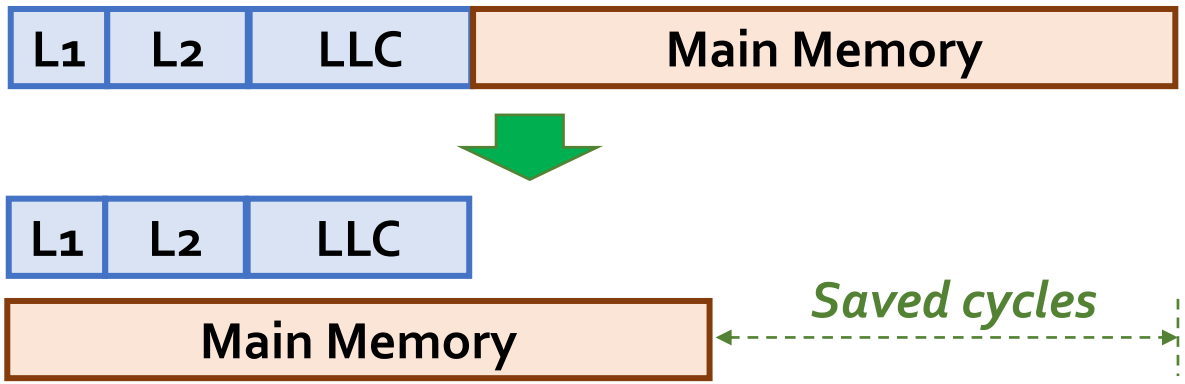
Key Observations



Observation 2

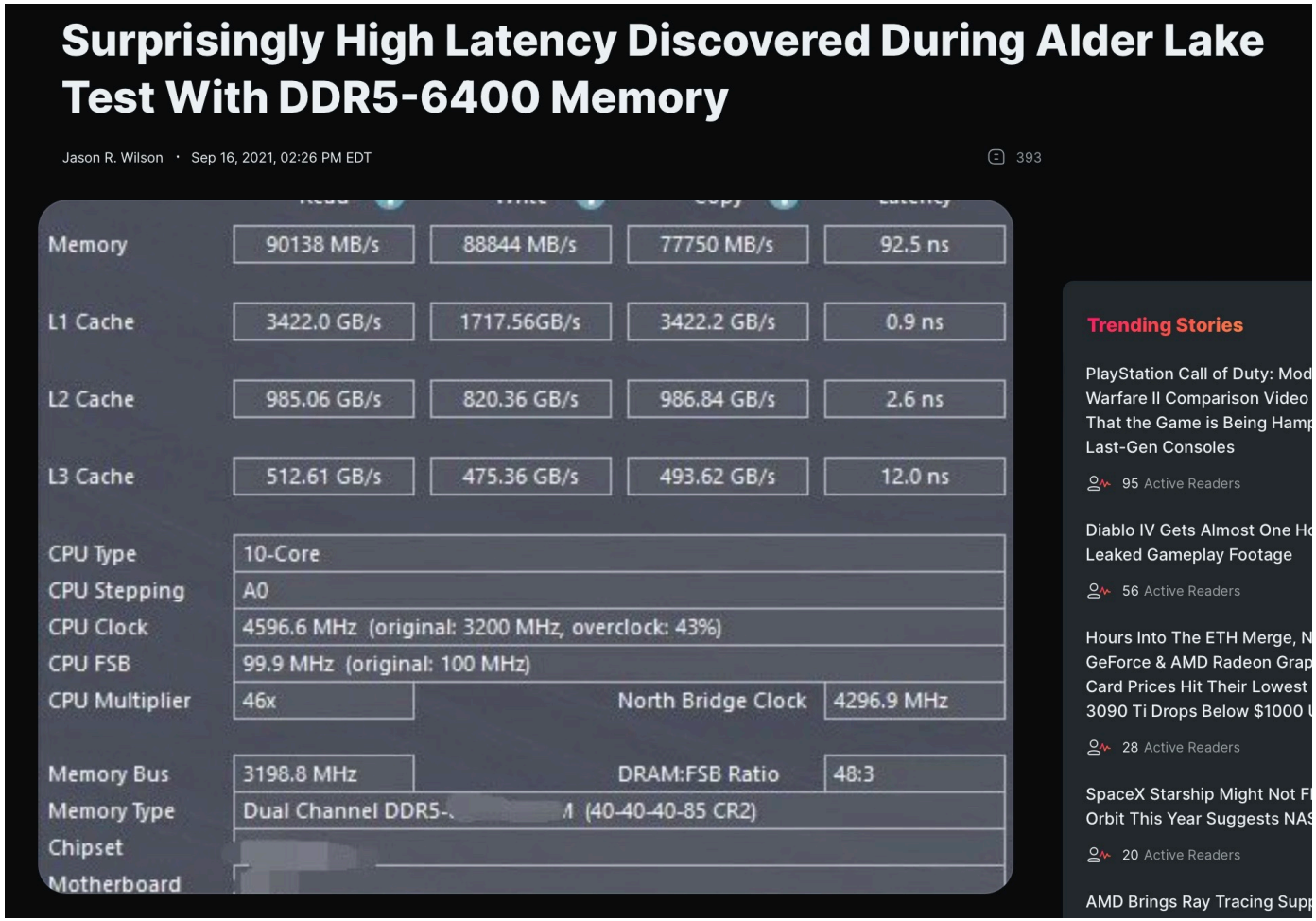
On-chip cache access latency significantly contributes to total off-chip load latency

40% of the stalls caused by an off-chip load can be reduced by **removing on-chip cache access latency from its critical path**



And Things are Getting Worse…

And Things are Getting Worse…



Surprisingly High Latency Discovered During Alder Lake Test With DDR5-6400 Memory

Jason R. Wilson · Sep 16, 2021, 02:26 PM EDT 393

	Read	Write	Copy	Latency
Memory	90138 MB/s	88844 MB/s	77750 MB/s	92.5 ns
L1 Cache	3422.0 GB/s	1717.56GB/s	3422.2 GB/s	0.9 ns
L2 Cache	985.06 GB/s	820.36 GB/s	986.84 GB/s	2.6 ns
L3 Cache	512.61 GB/s	475.36 GB/s	493.62 GB/s	12.0 ns

CPU Type	10-Core		
CPU Stepping	A0		
CPU Clock	4596.6 MHz (original: 3200 MHz, overclock: 43%)		
CPU FSB	99.9 MHz (original: 100 MHz)		
CPU Multiplier	46x	North Bridge Clock	4296.9 MHz
Memory Bus	3198.8 MHz	DRAM:FSB Ratio	48:3
Memory Type	Dual Channel DDR5-[redacted] (40-40-40-85 CR2)		
Chipset	[redacted]		
Motherboard	[redacted]		

Trending Stories

PlayStation Call of Duty: Mod
Warfare II Comparison Video
That the Game is Being Hamp
Last-Gen Consoles
95 Active Readers

Diablo IV Gets Almost One Ho
Leaked Gameplay Footage
56 Active Readers

Hours Into The ETH Merge, N
GeForce & AMD Radeon Grap
Card Prices Hit Their Lowest
3090 Ti Drops Below $1000 U
28 Active Readers

SpaceX Starship Might Not F
Orbit This Year Suggests NAS
20 Active Readers

AMD Brings Ray Tracing Supp

https://wccftech.com/surprisingly-high-latency-discovered-during-alder-lake-test-with-ddr5-6400-memory

And Things are Getting Worse…

Surprisingly High Late… Test With DDR5-640…

Jason R. Wilson · Sep 16, 2021, 02:26 PM EDT

Memory	90138 MB/s	88844		
L1 Cache	3422.0 GB/s	1717.56GB/s	3422.2 GB/s	0.9 ns
L2 Cache	985.06 GB/s	820.36 GB/s	986.84 GB/s	2.6 ns
L3 Cache	512.61 GB/s	475.36 GB/s	493.62 GB/s	12.0 ns

CPU Type	10-Core		
CPU Stepping	A0		
CPU Clock	4596.6 MHz (original: 3200 MHz, overclock: 43%)		
CPU FSB	99.9 MHz (original: 100 MHz)		
CPU Multiplier	46x	North Bridge Clock	4296.9 MHz
Memory Bus	3198.8 MHz	DRAM:FSB Ratio	48:3
Memory Type	Dual Channel DDR5-… (40-40-40-85 CR2)		
Chipset			
Motherboard			

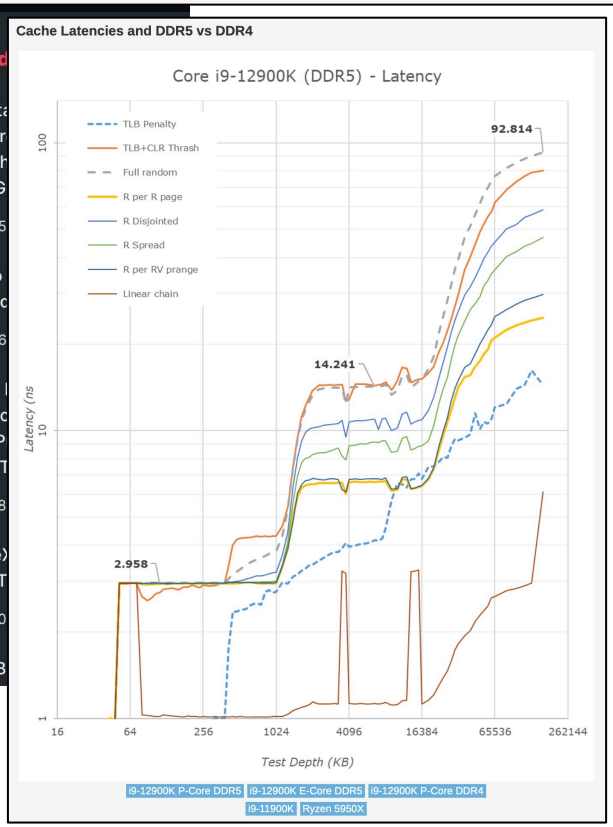
The Intel 12th Gen Core i9-12900K Review: Hybrid Performance Brings Hybrid Complexity

474 Comments

+ Add A Comment

by Dr. Ian Cutress & Andrei Frumusanu on November 4, 2021 9:00 AM EST

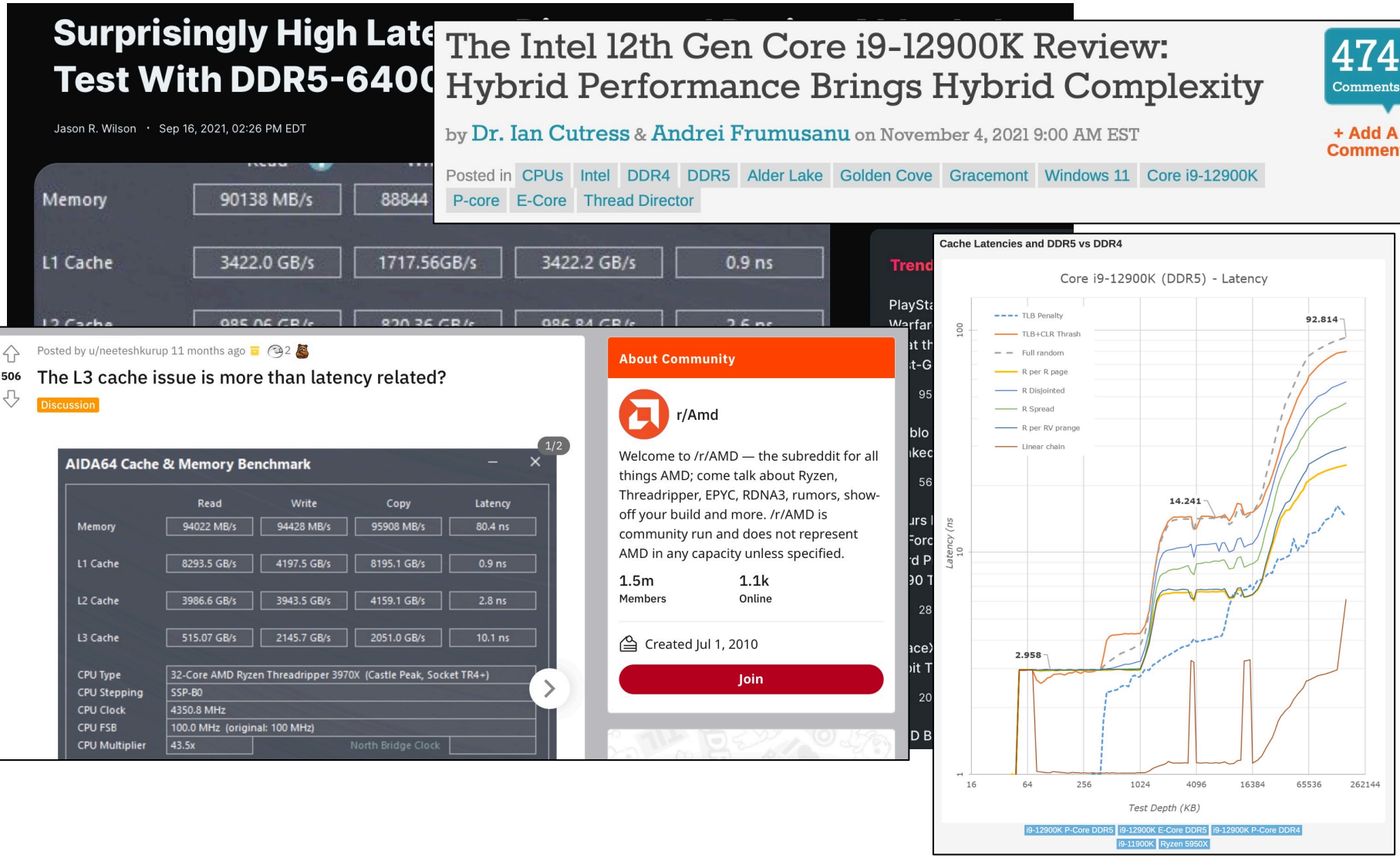
Posted in CPUs | Intel | DDR4 | DDR5 | Alder Lake | Golden Cove | Gracemont | Windows 11 | Core i9-12900K | P-core | E-Core | Thread Director



https://wccftech.com/surprisingly-high-latency-discovered-during-alder-lake-test-with-ddr5-6400-memory

https://www.anandtech.com/show/17047/the-intel-12th-gen-core-i912900k-review-hybrid-performance-brings-hybrid-complexity/6

And Things are Getting Worse…



https://wccftech.com/surprisingly-high-latency-discovered-during-alder-lake-test-with-ddr5-6400-memory
https://www.anandtech.com/show/17047/the-intel-12th-gen-core-i912900k-review-hybrid-performance-brings-hybrid-complexity/6
https://www.reddit.com/r/Amd/comments/q8k3u7/the_l3_cache_issue_is_more_than_latency_related

Our Goal

Improve processor performance
by **removing on-chip cache access latency**
from the **critical path of off-chip loads**

HERMES

HERMES

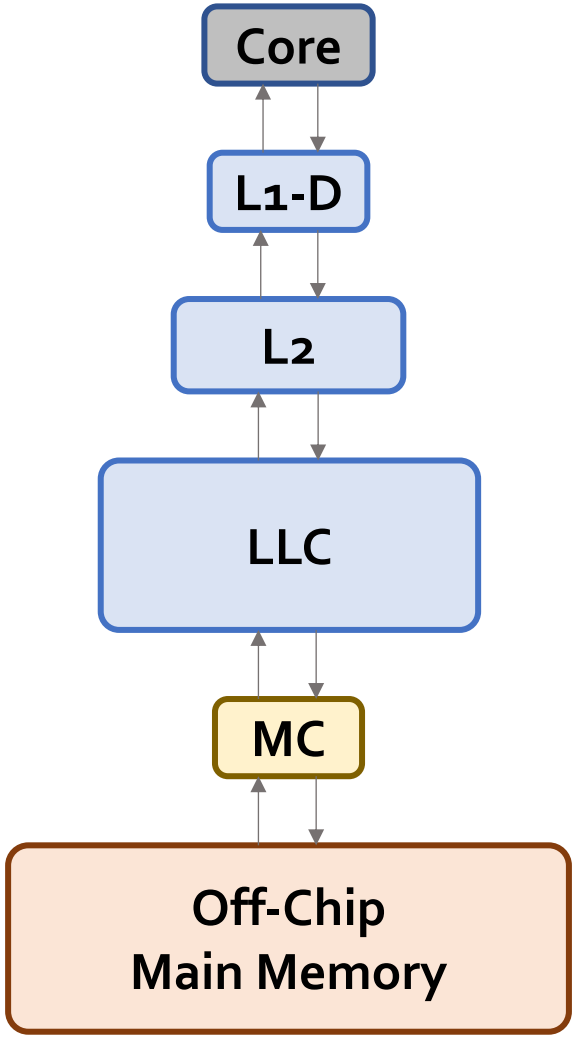
Predicts which load requests might go off-chip

HERMES

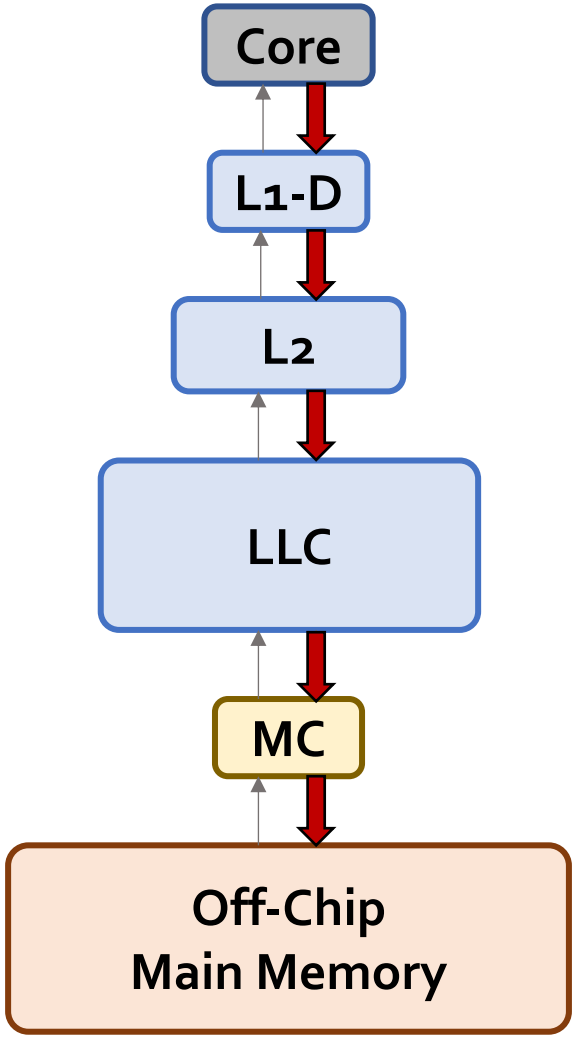
Predicts which load requests might go off-chip

Starts **fetching** data **directly** from main memory while concurrently accessing the cache hierarchy

Hermes Overview

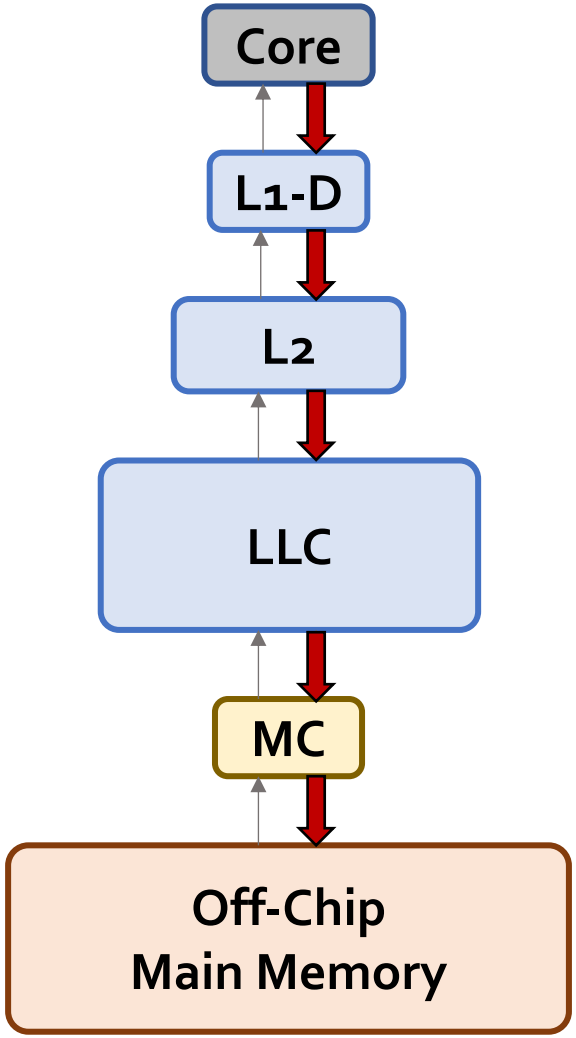


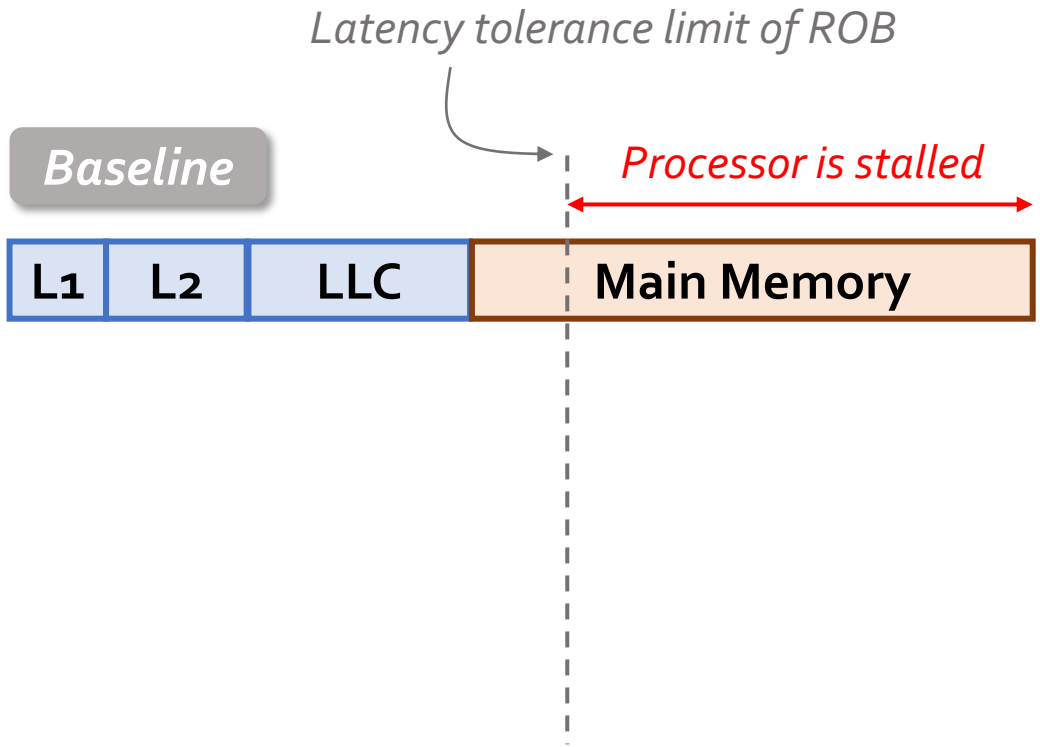
Hermes Overview



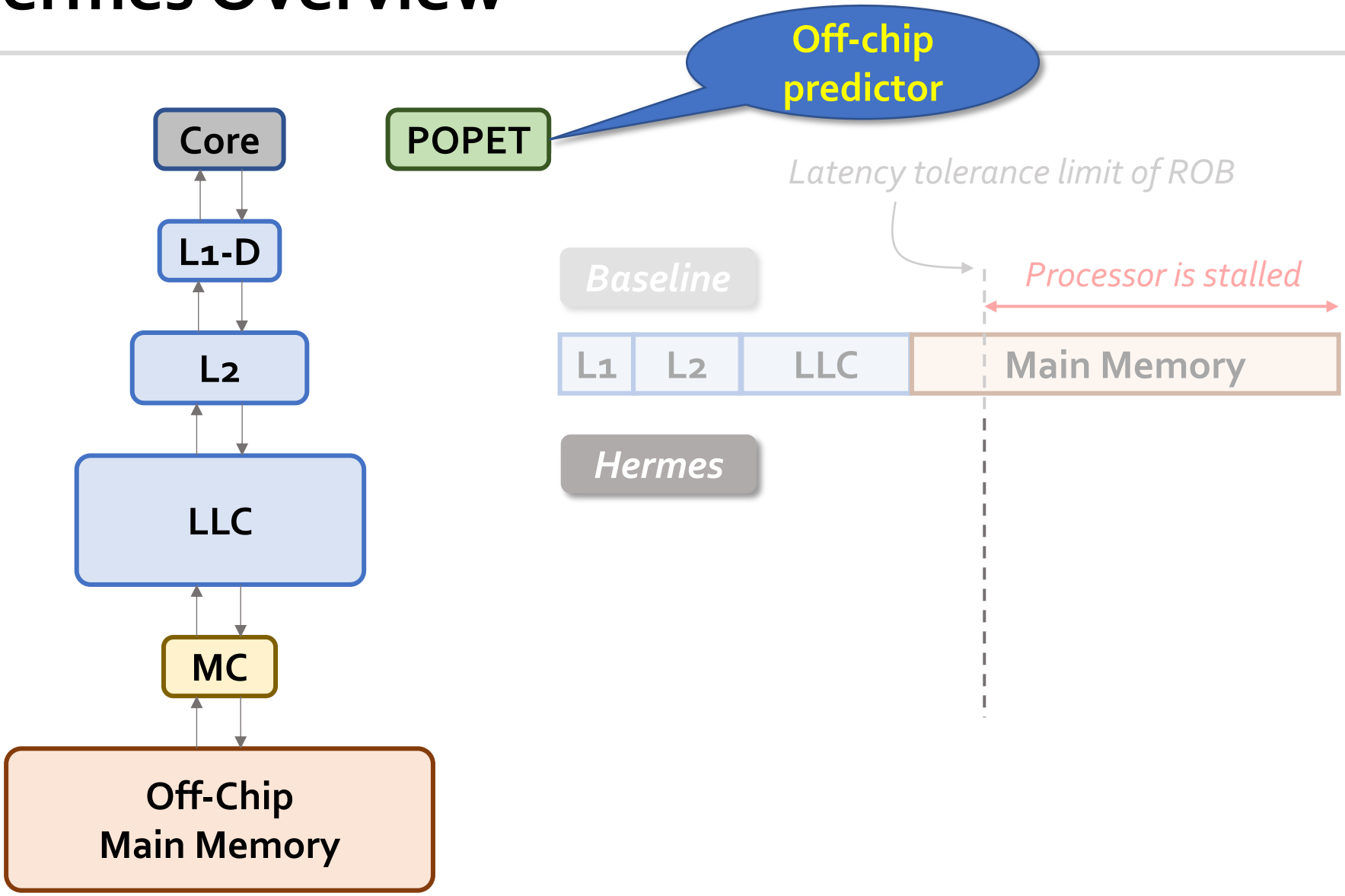
Baseline

Hermes Overview

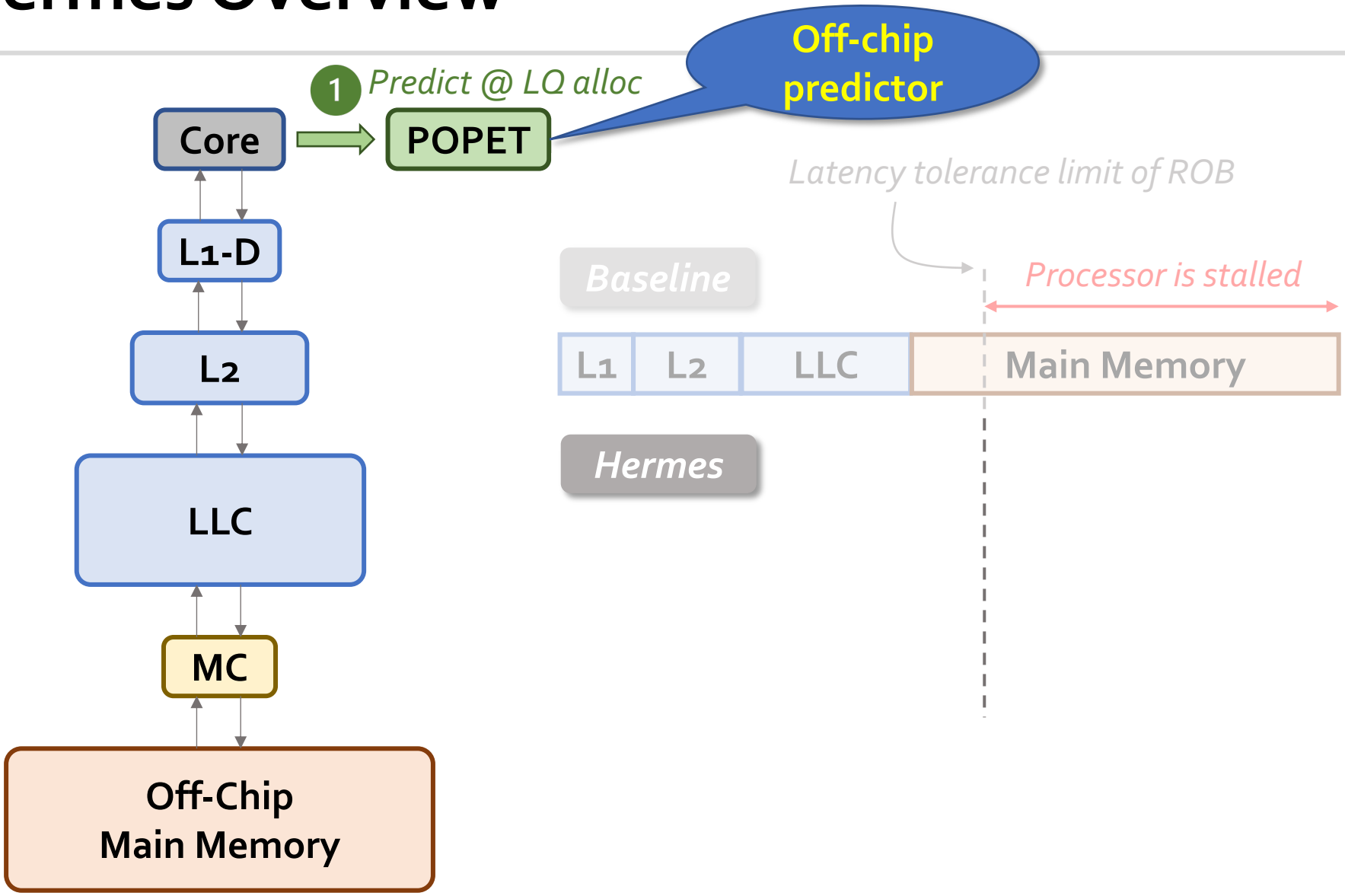




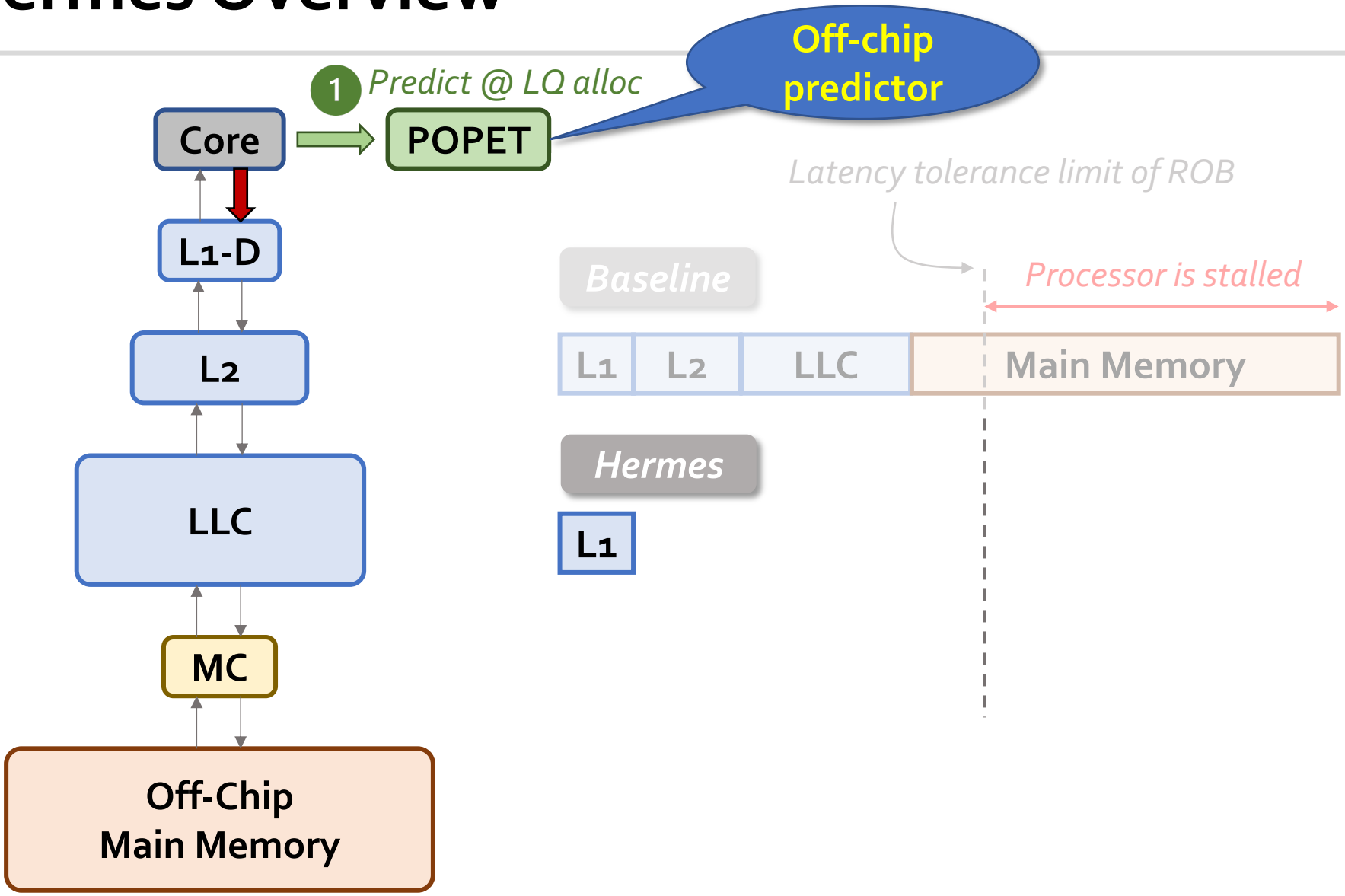
Hermes Overview



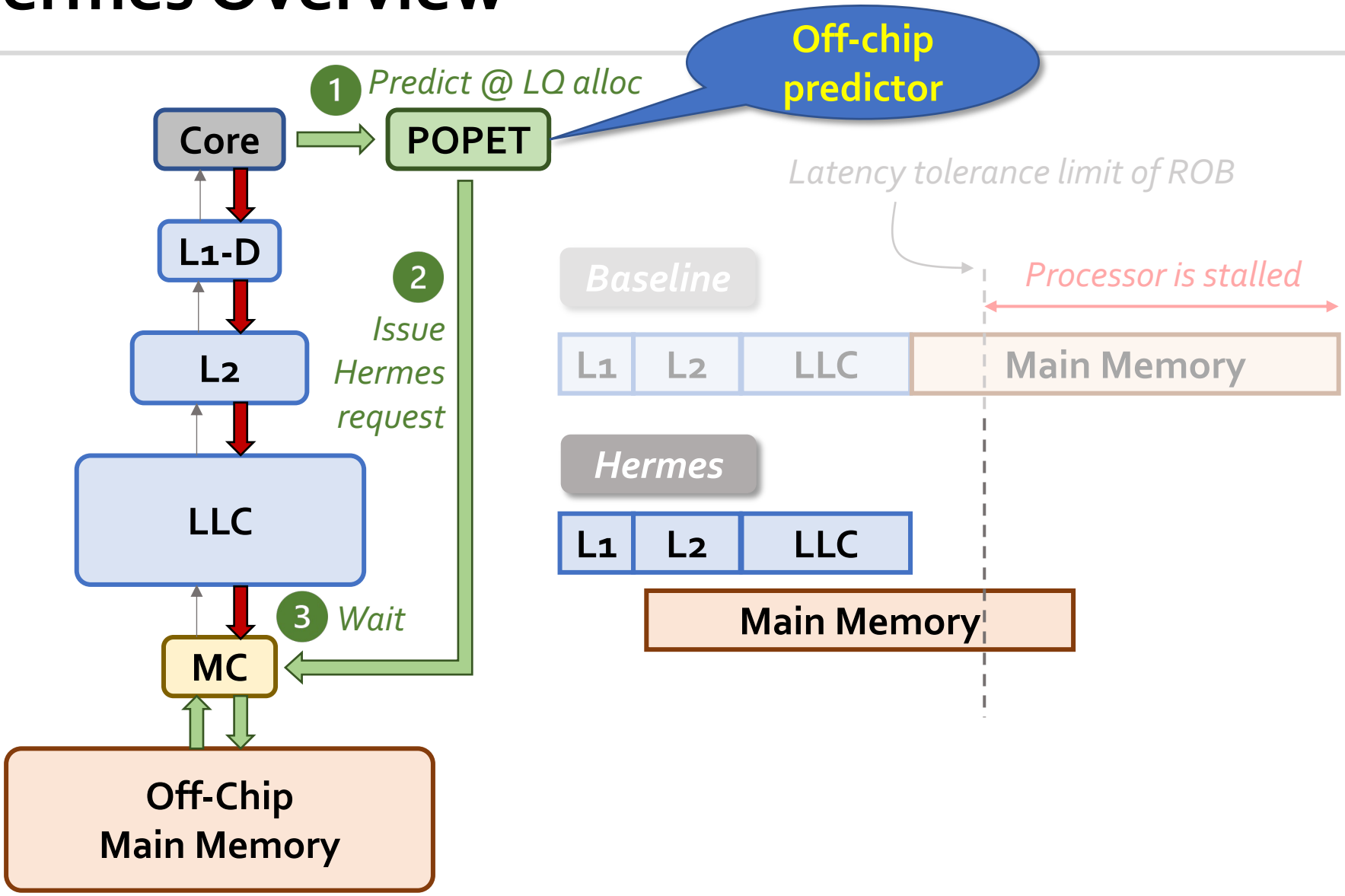
Hermes Overview



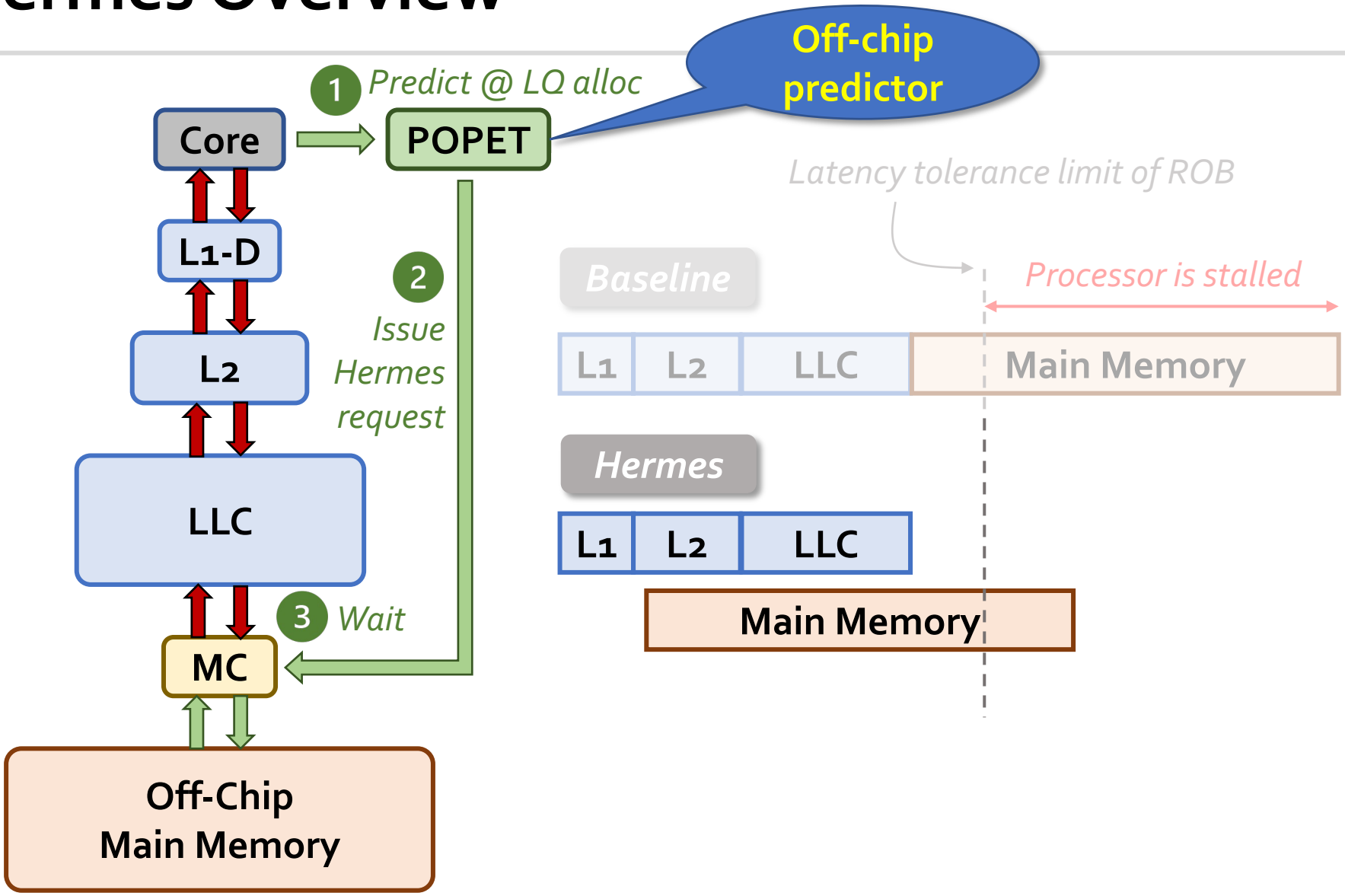
Hermes Overview



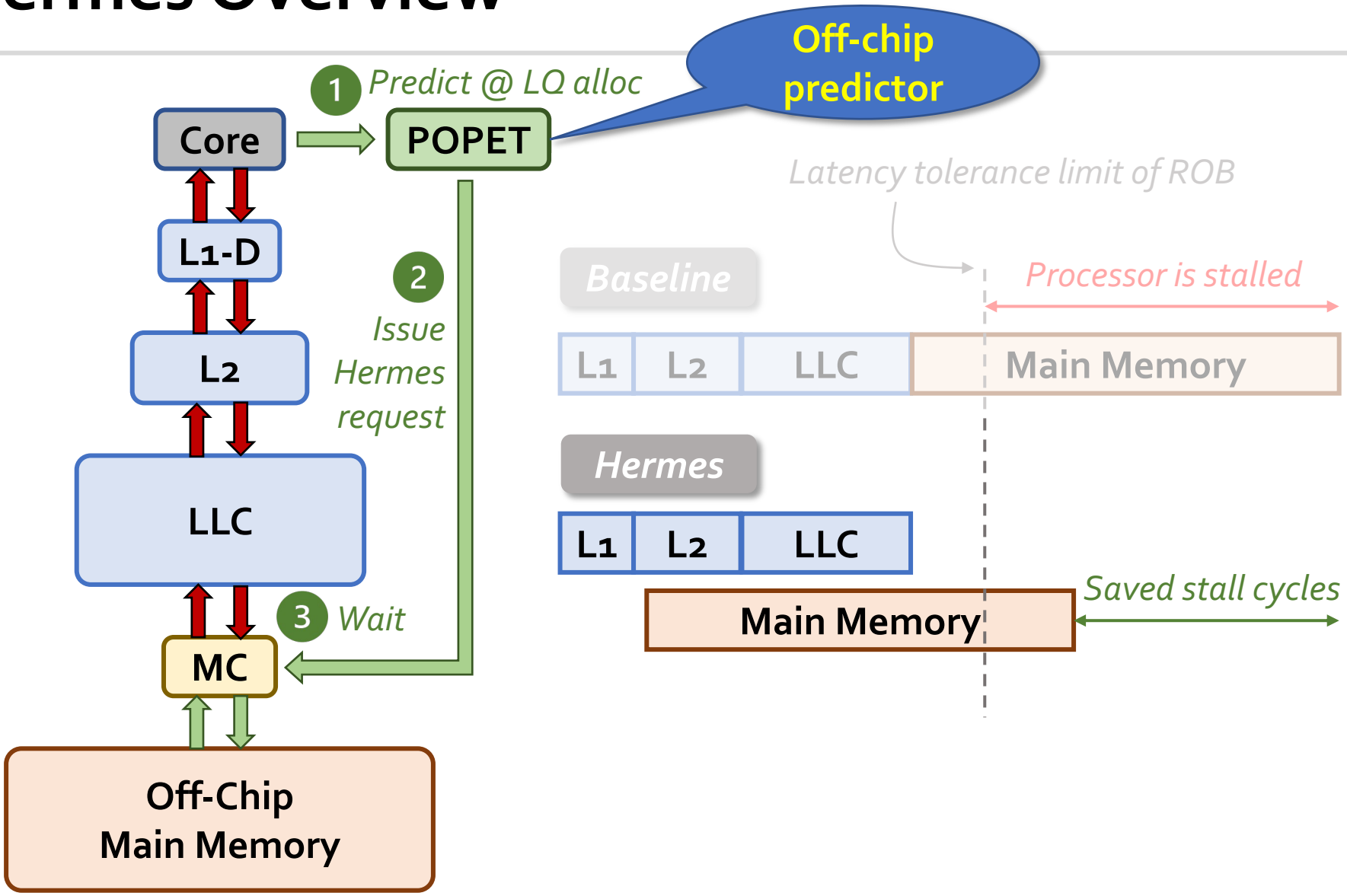
Hermes Overview



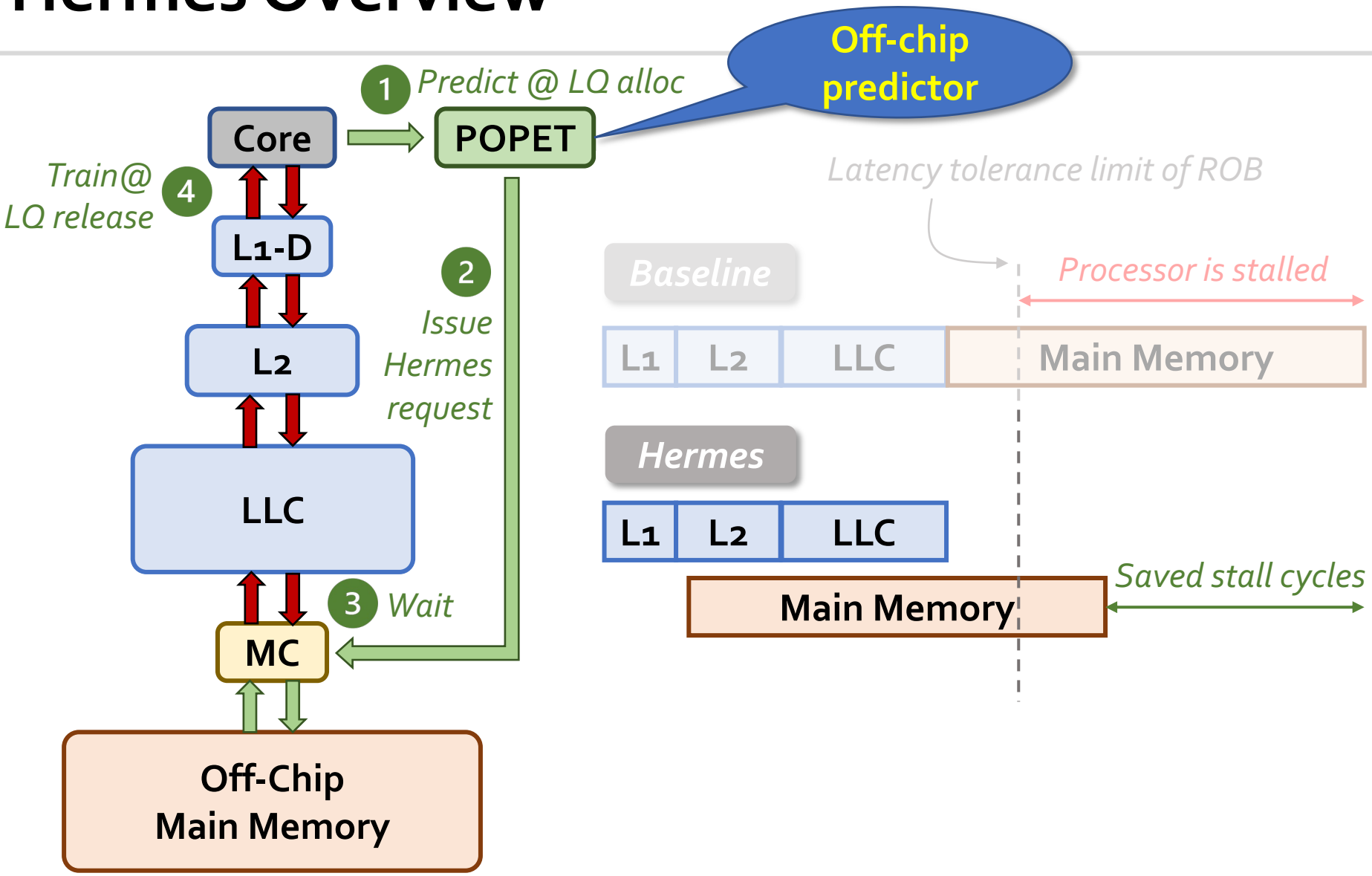
Hermes Overview



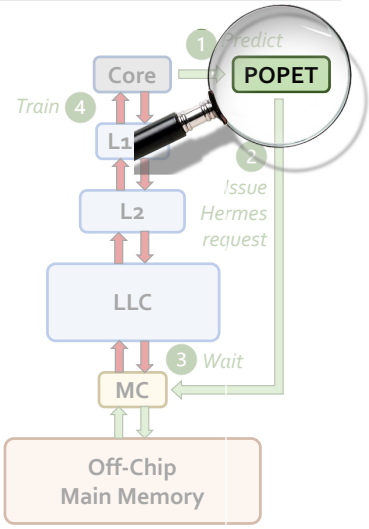
Hermes Overview



Hermes Overview

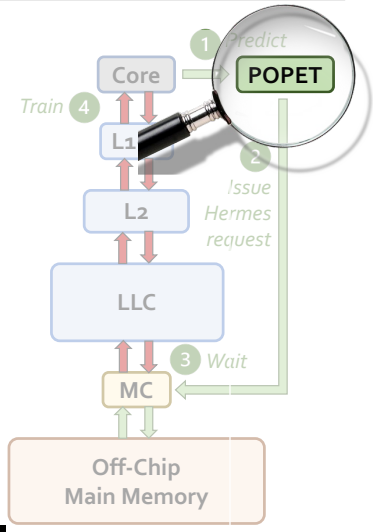


Designing The Off-Chip Predictor

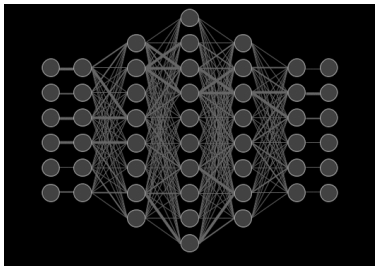


Designing The Off-Chip Predictor

Tracking cache contents



Learning from program behavior

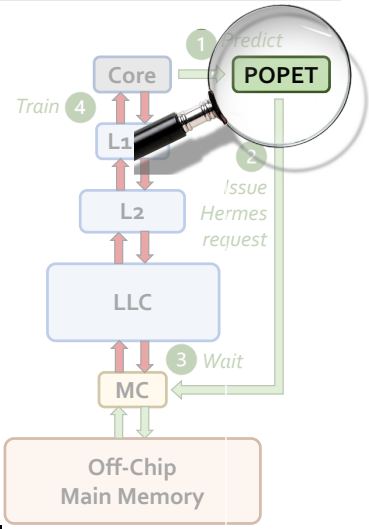


Designing The Off-Chip Predictor

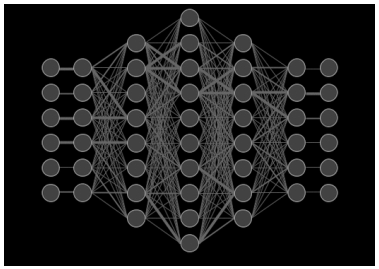
Tracking cache contents

- ✕ Large metadata
  - Metadata size increases with cache hierarchy size
- ✕ Track every cache operations
  - Gets tricky based on the cache hierarchy configuration

(e.g., inclusivity, bypassing,…)

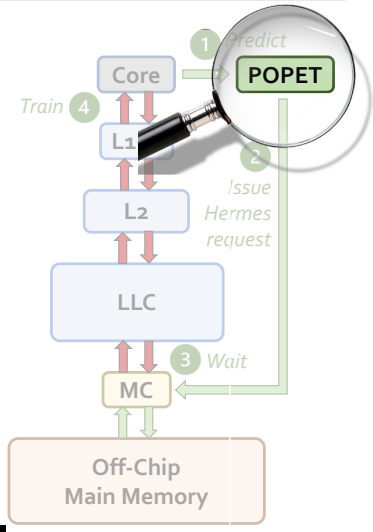


Learning from program behavior



Designing The Off-Chip Predictor

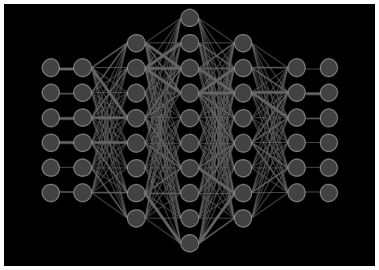
Tracking cache contents



- ✗ Large metadata
  - Metadata size increases with cache hierarchy size
- ✗ Track every cache operations
  - Gets tricky based on the cache hierarchy configuration

(e.g., inclusivity, bypassing,…)

Learning from program behavior



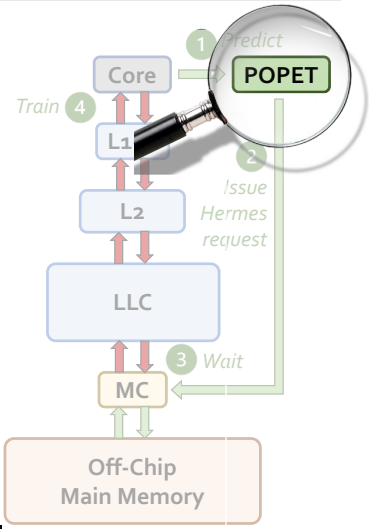
Correlate different program features with off-chip loads

Designing The Off-Chip Predictor

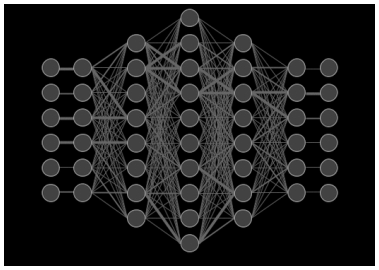
Tracking cache contents

- ✗ Large metadata
  - Metadata size increases with cache hierarchy size
- ✗ Track every cache operations
  - Gets tricky based on the cache hierarchy configuration

(e.g., inclusivity, bypassing,…)



Learning from program behavior

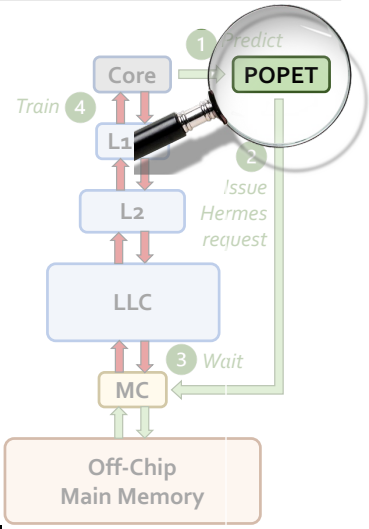


Correlate different program features with off-chip loads

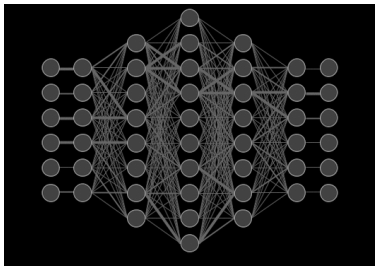
- ✓ Lower storage overhead
- ✓ Lower design complexity

Designing The Off-Chip Predictor

Tracking cache contents



- ✗ Large metadata
  - Metadata size increases with cache hierarchy size
- ✗ Track every cache operations
  - Gets tricky based on the cache hierarchy configuration (e.g., inclusivity, bypassing,...)



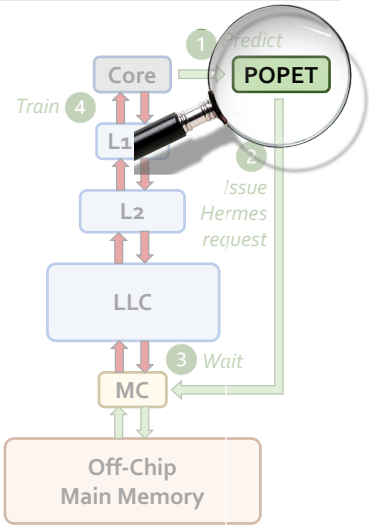
Learning from program behavior

Correlate different program features with off-chip loads

- ✓ Lower storage overhead
- ✓ Lower design complexity

POPET: Perceptron-Based Off-Chip Predictor

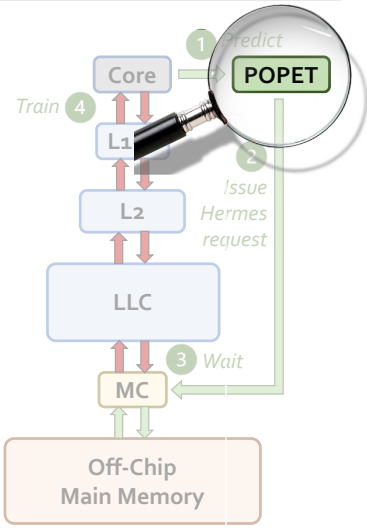
- Multi-feature hashed perceptron model[1]

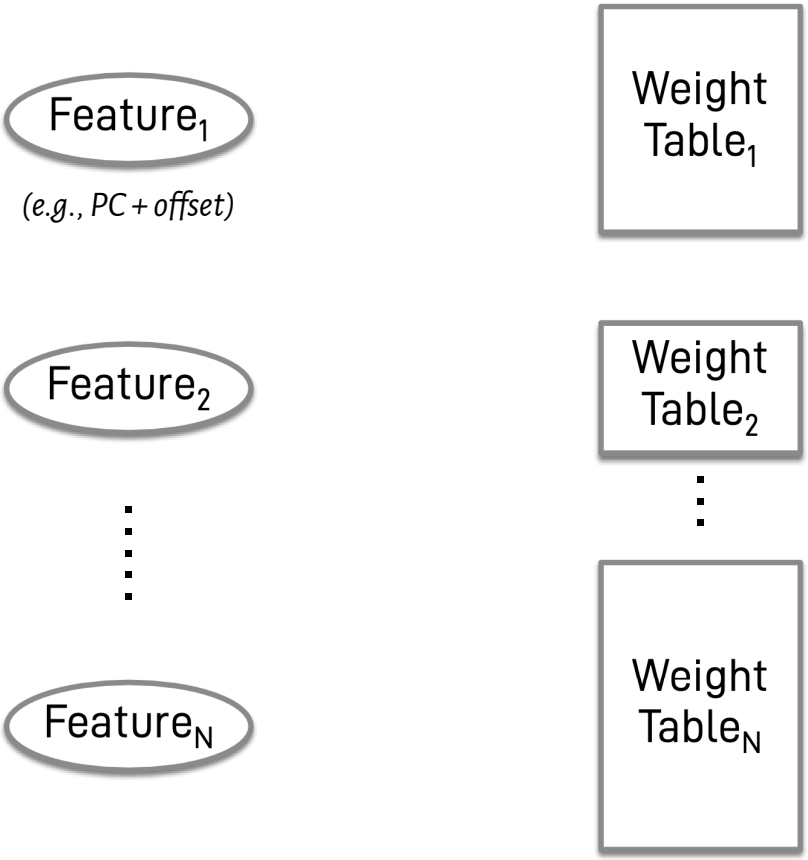


[1] D. Tarjan and K. Skadron, "Merging Path and Gshare Indexing in Perceptron Branch Prediction," TACO, 2005

POPET: Perceptron-Based Off-Chip Predictor

- Multi-feature hashed perceptron model[1]
  - Each feature has its own *weight table*
    - Stores correlation between feature value and off-chip prediction

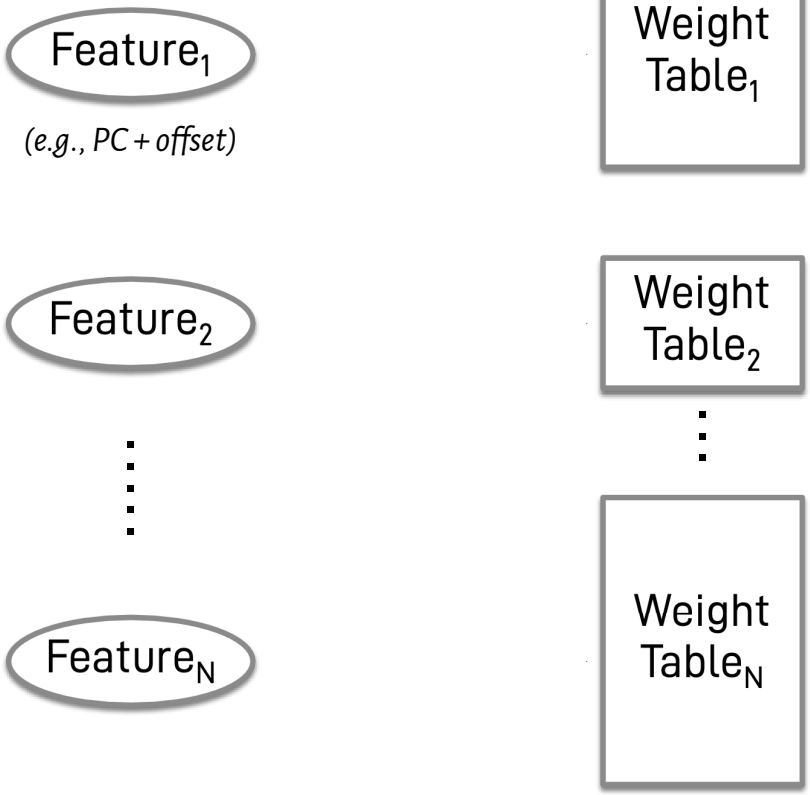


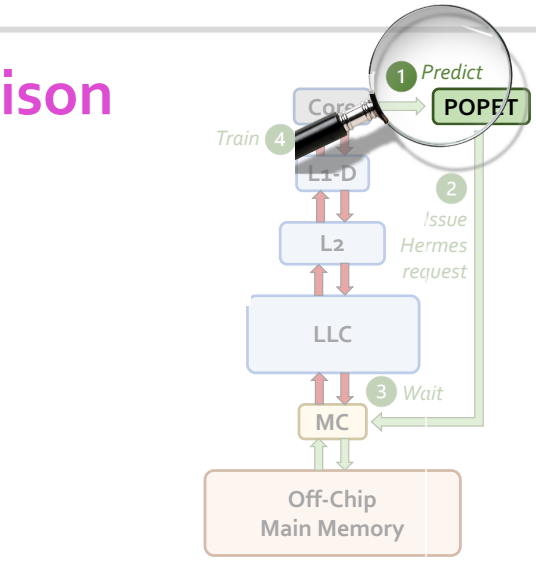


[1] D. Tarjan and K. Skadron, "Merging Path and Gshare Indexing in Perceptron Branch Prediction," TACO, 2005

Predicting using POPET

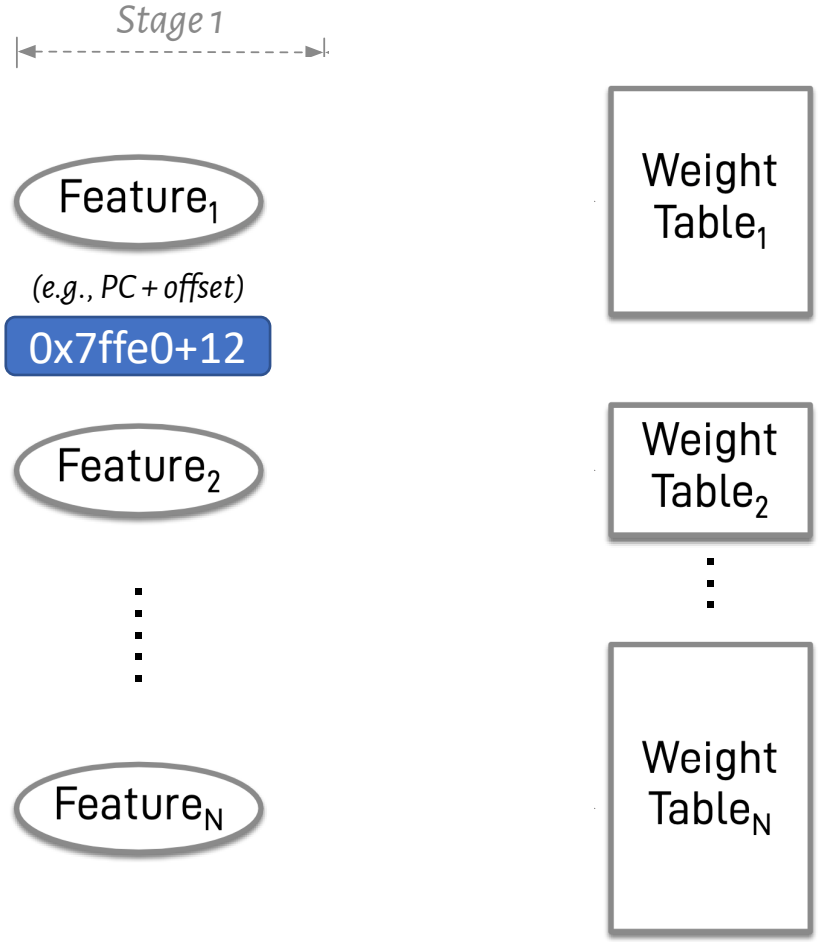
- Using simple **table lookups**, **addition**, and **comparison**

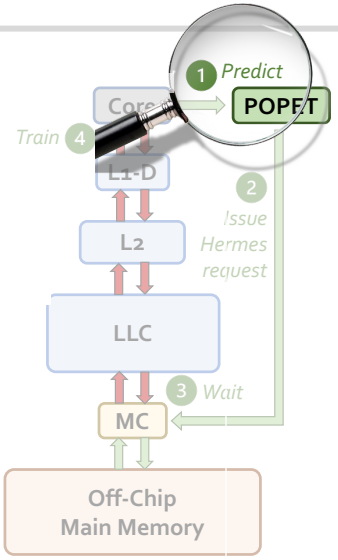




Predicting using POPET

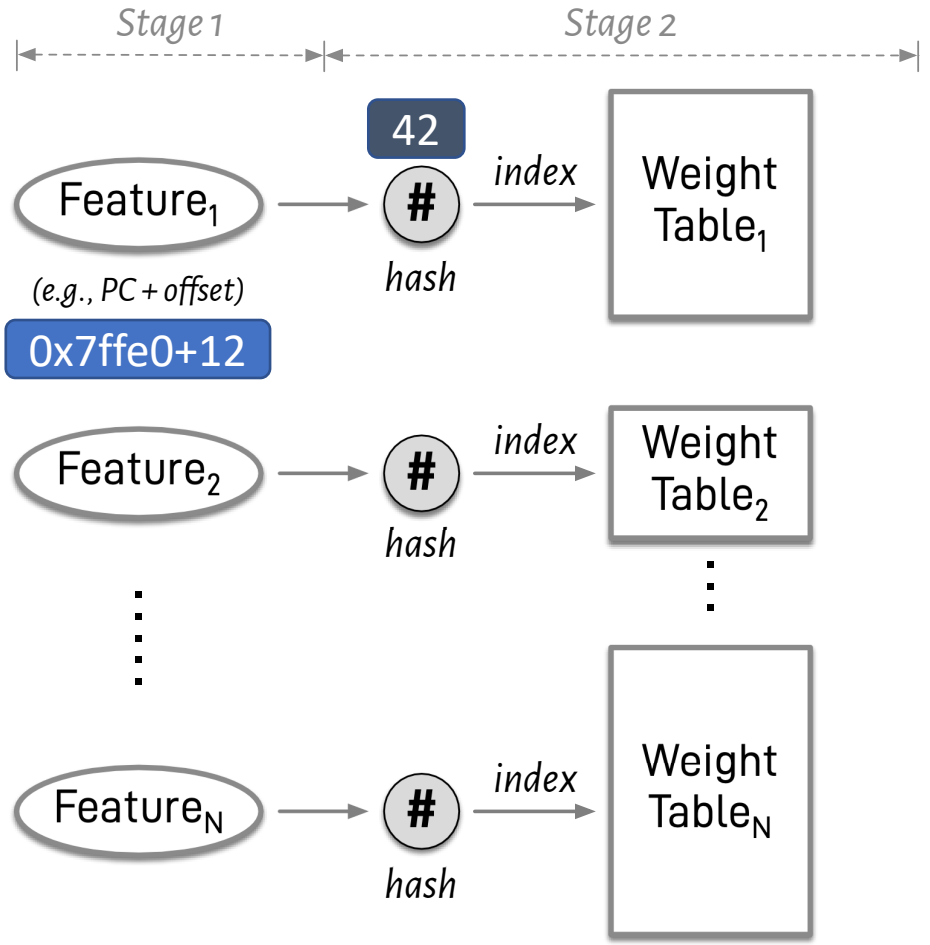
- Using simple **table lookups**, **addition**, and **comparison**

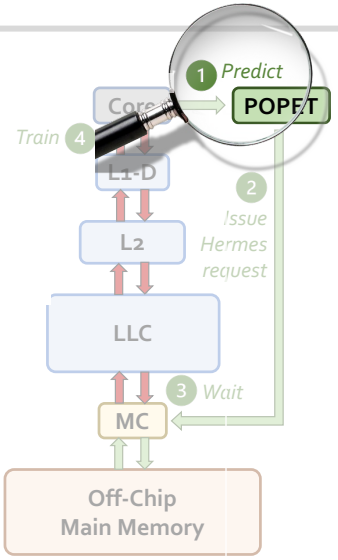




Predicting using POPET

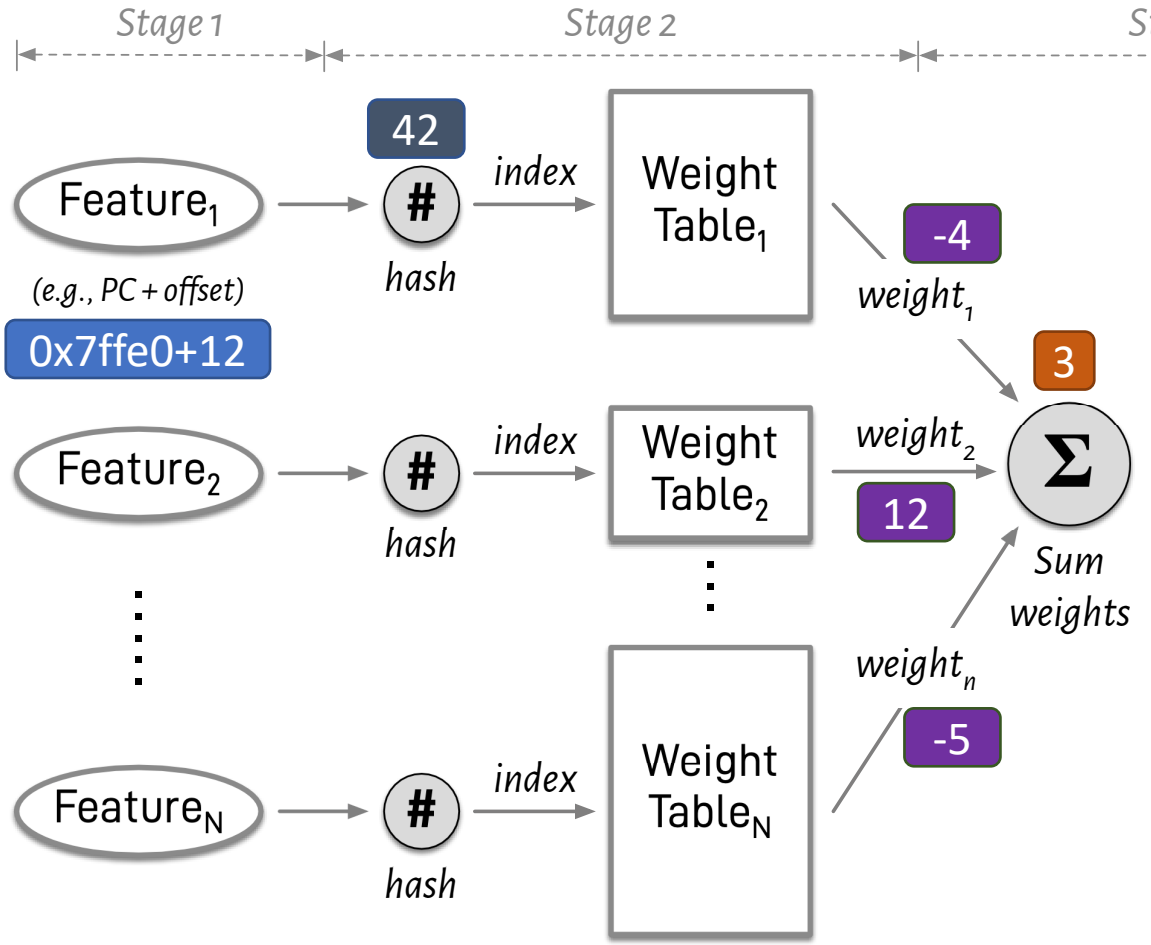
- Using simple **table lookups**, **addition**, and **comparison**

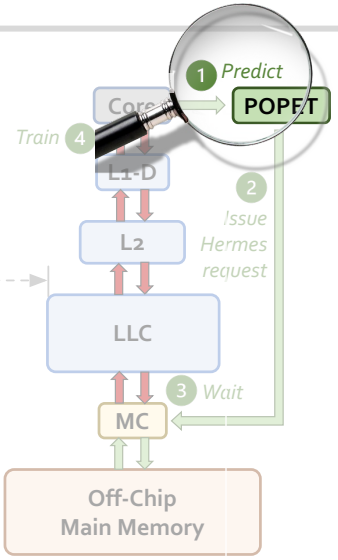




Predicting using POPET

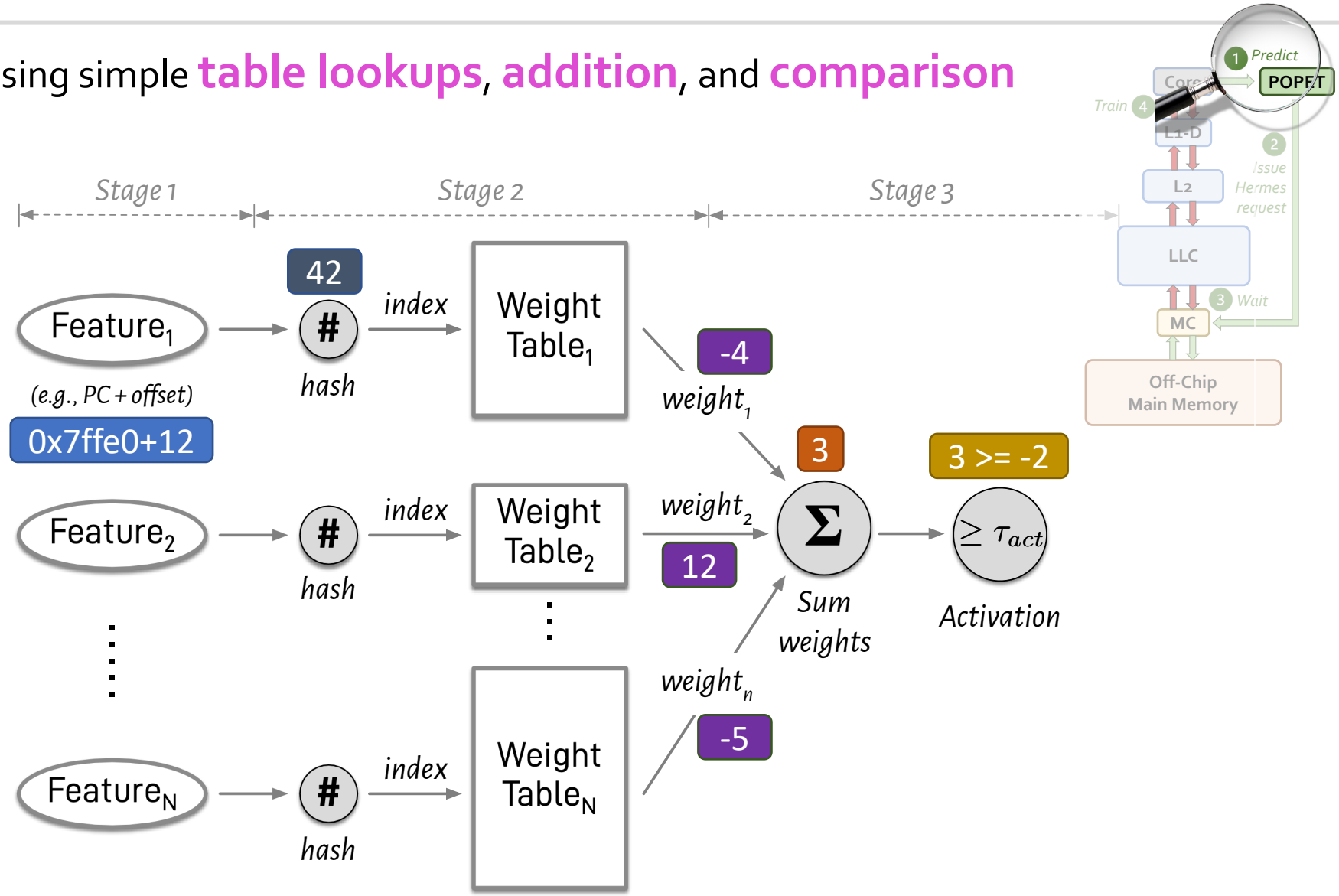
- Using simple **table lookups**, **addition**, and **comparison**





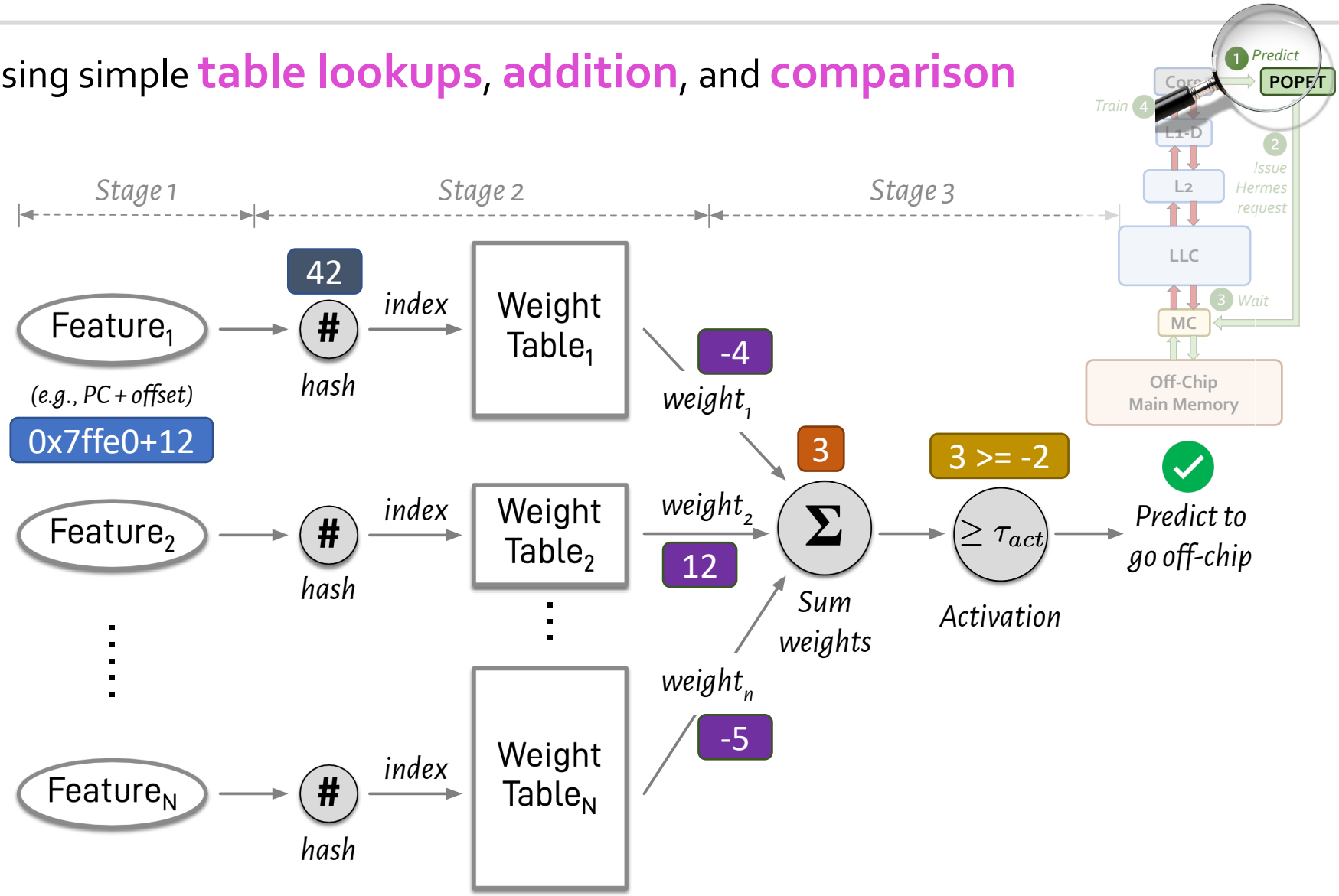
Predicting using POPET

- Using simple **table lookups**, **addition**, and **comparison**



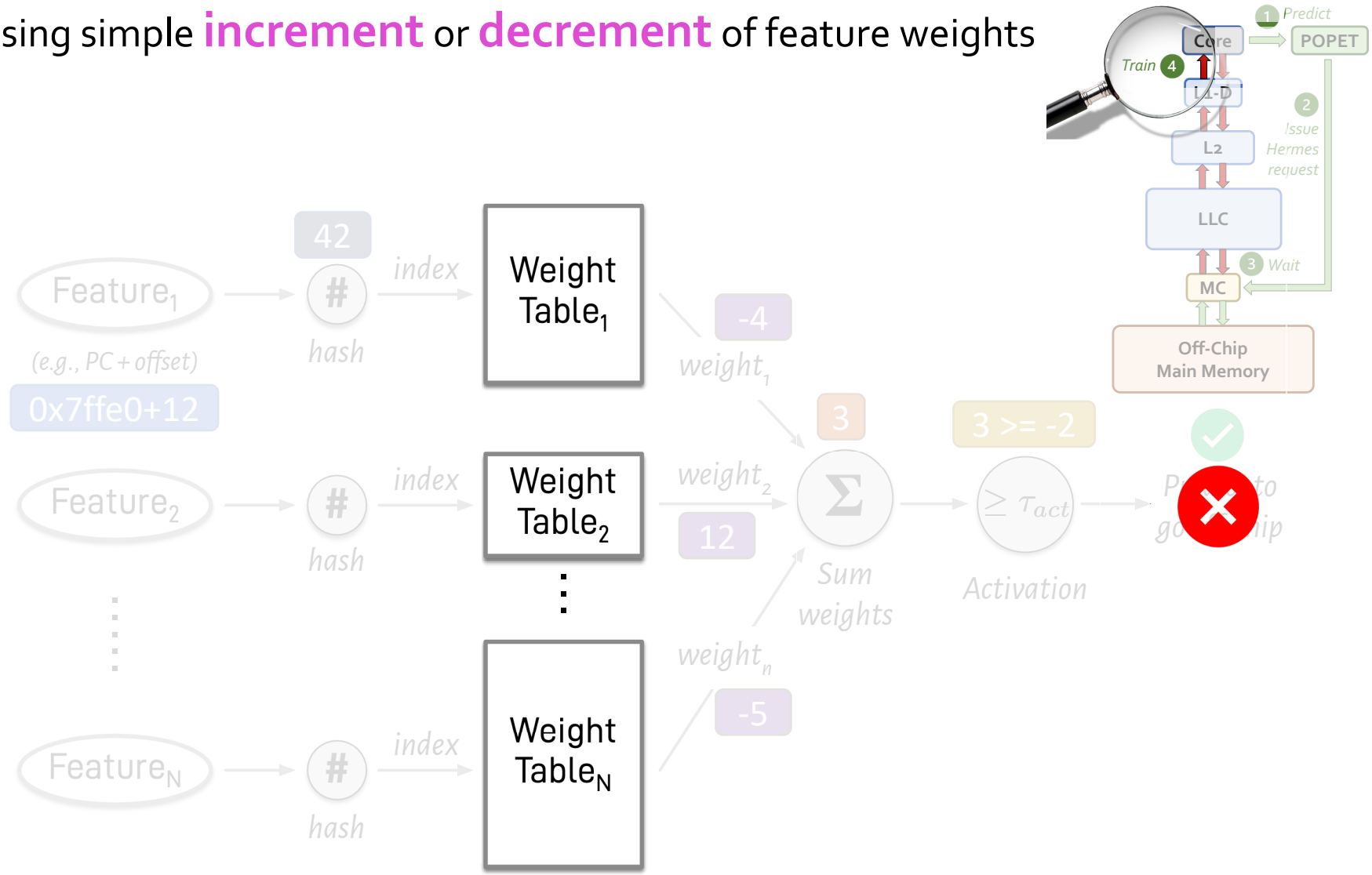
Predicting using POPET

- Using simple **table lookups**, **addition**, and **comparison**



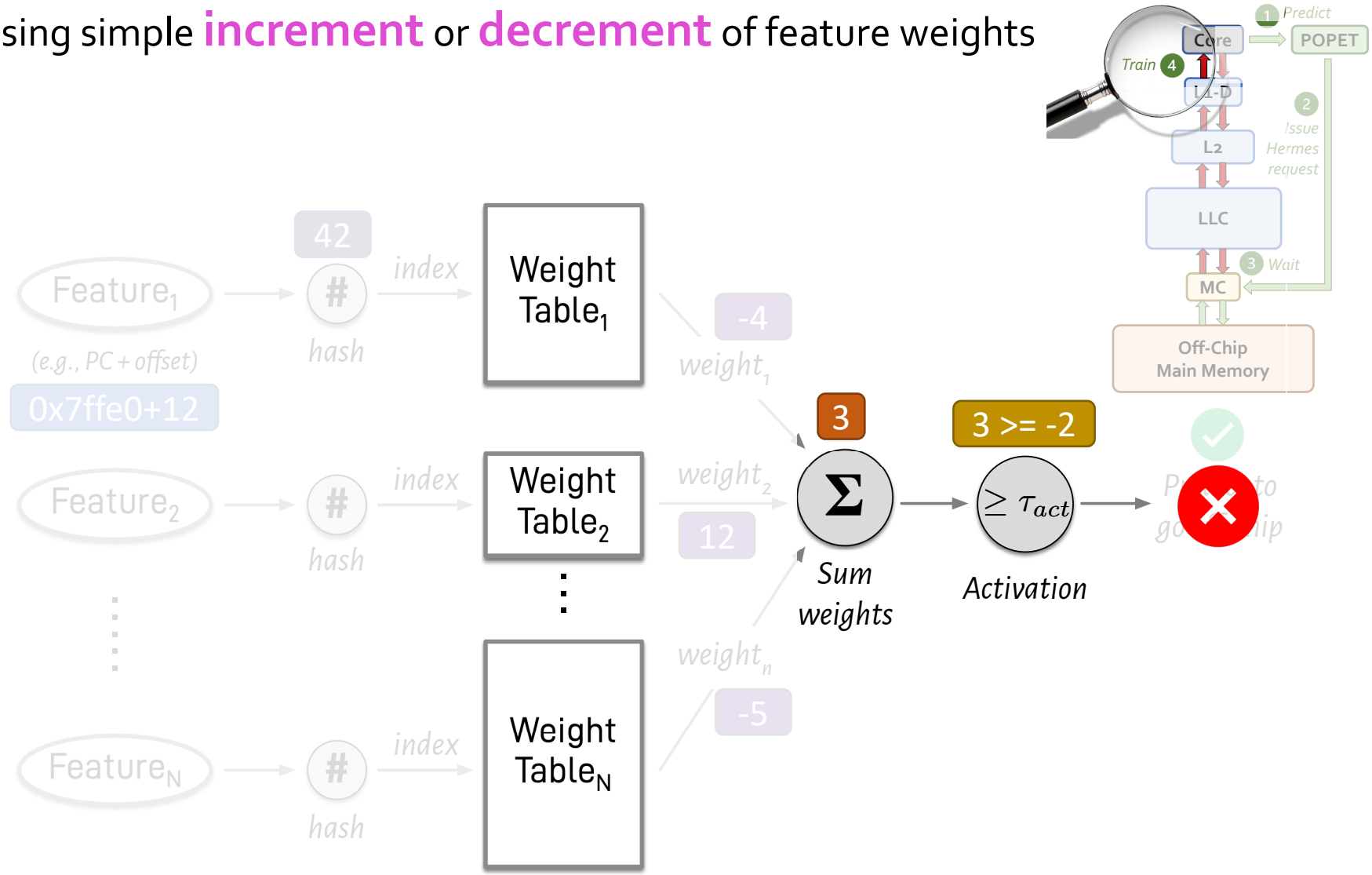
Training POPET

- Using simple **increment** or **decrement** of feature weights



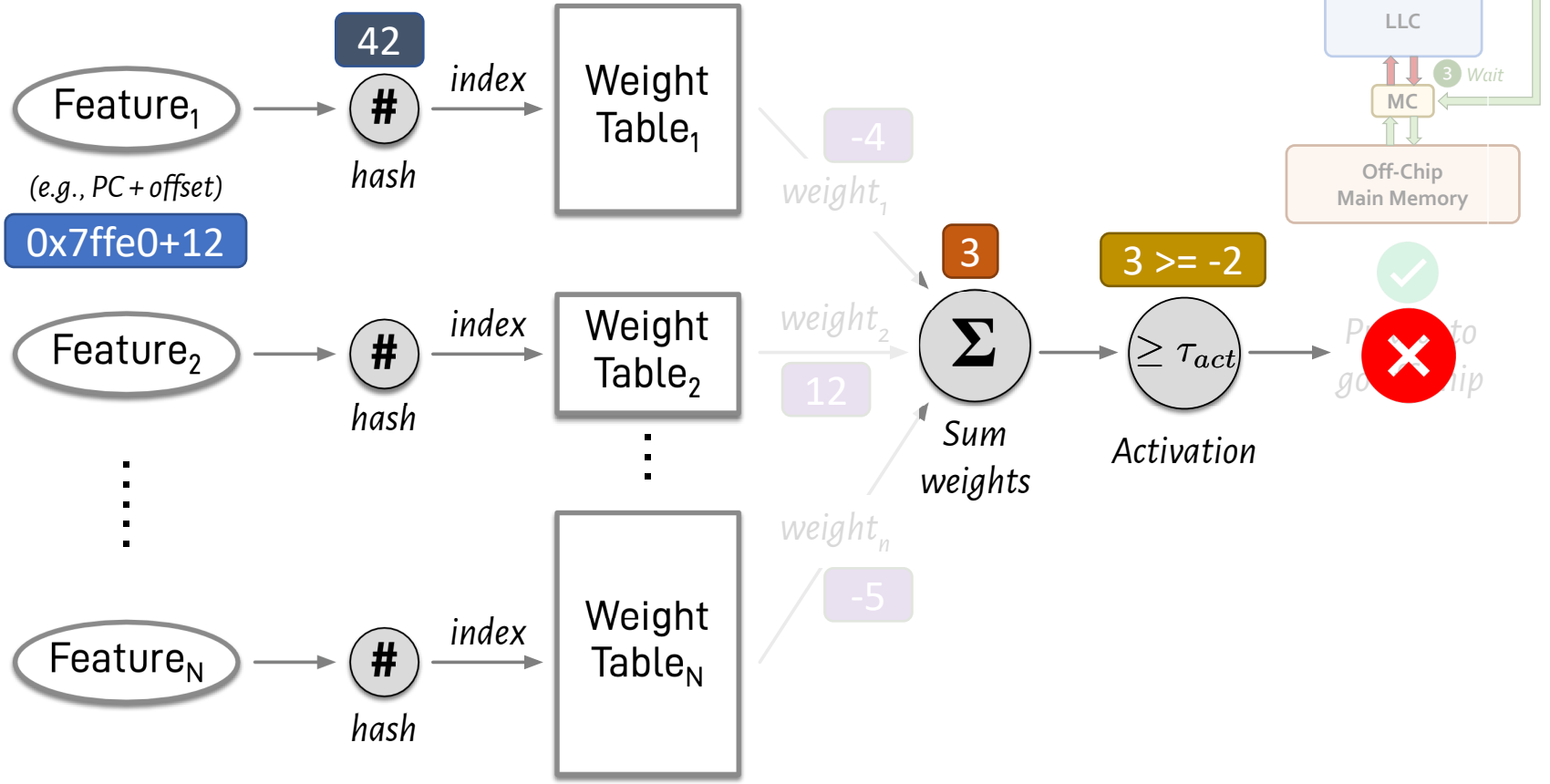
Training POPET

- Using simple **increment** or **decrement** of feature weights



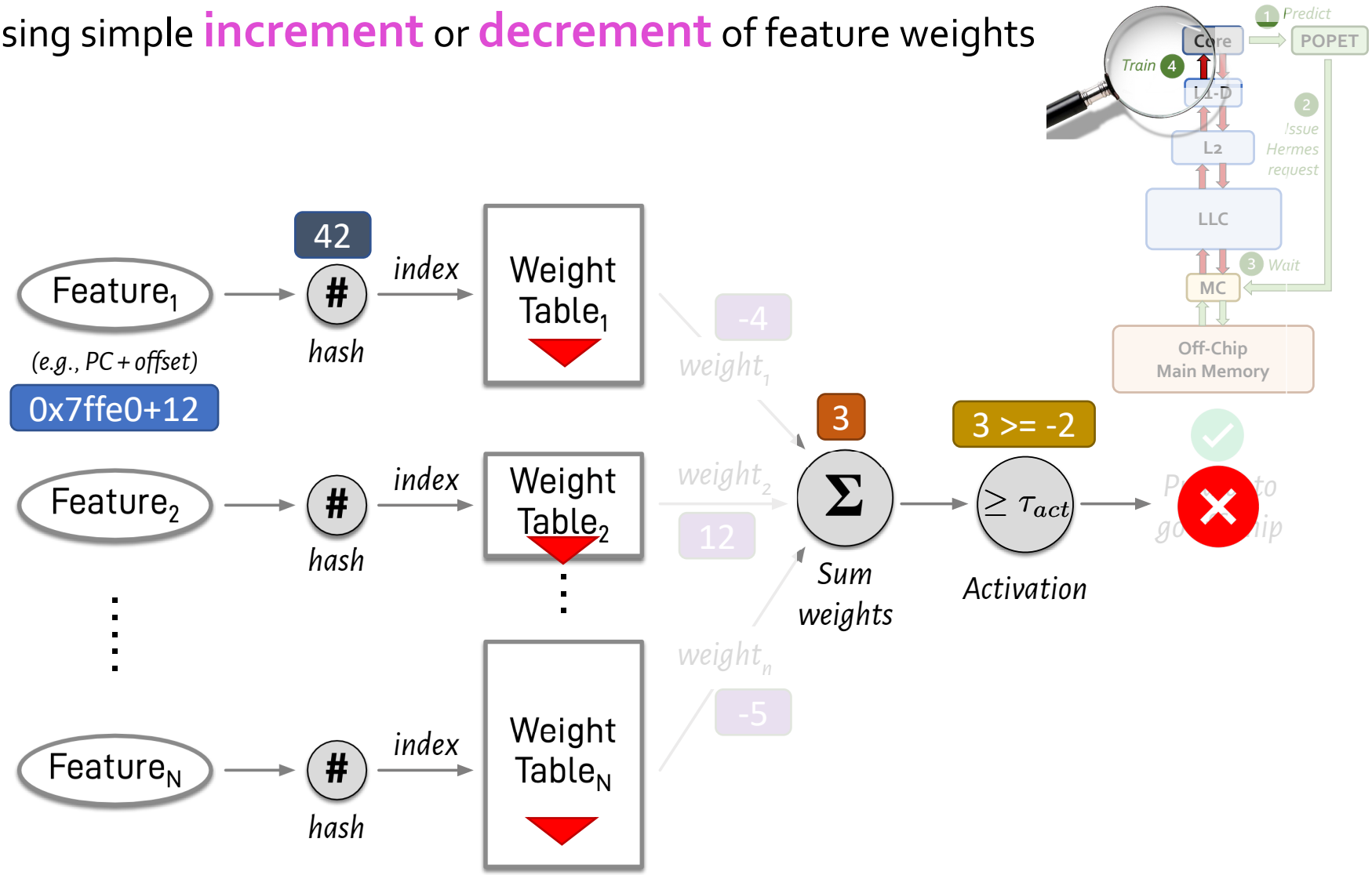
Training POPET

- Using simple **increment** or **decrement** of feature weights



Training POPET

- Using simple **increment** or **decrement** of feature weights



Evaluation

Simulation Methodology

- **ChampSim** trace driven simulator

Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** homogeneous and heterogeneous trace mixes

Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** homogeneous and heterogeneous trace mixes

LLC Prefetchers

- Pythia [Bera+, MICRO'22]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

Simulation Methodology

- **ChampSim** trace driven simulator
- **110 single-core** memory-intensive traces
  - SPEC CPU 2006 and 2017
  - PARSEC 2.1
  - Ligra
  - Real-world applications
- **220 eight-core** homogeneous and heterogeneous trace mixes

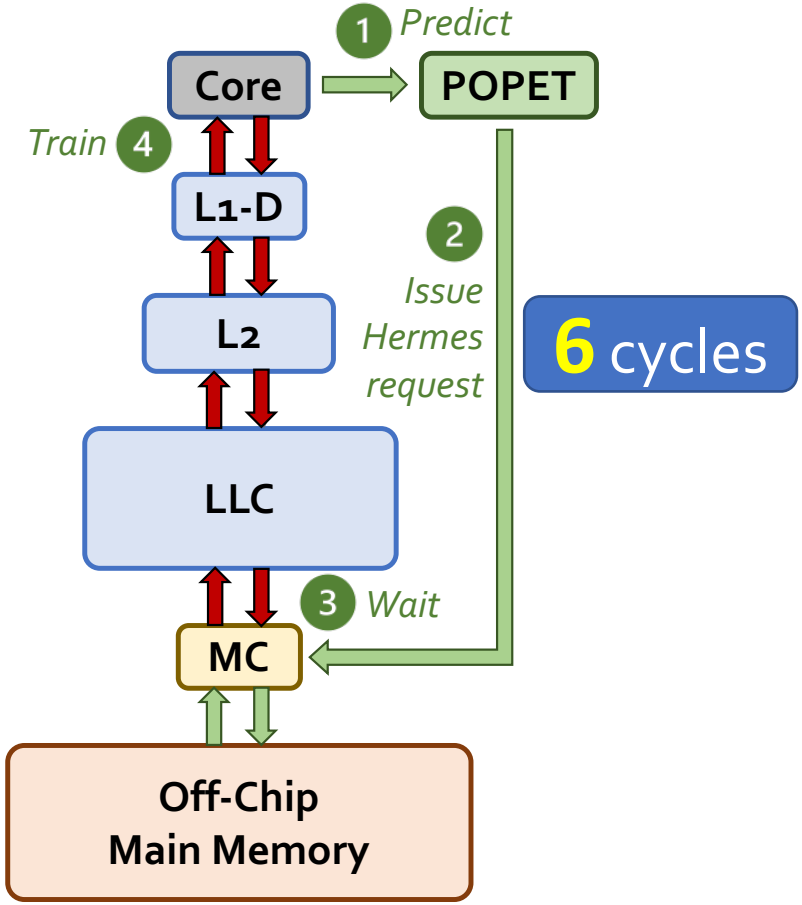
LLC Prefetchers

- Pythia [Bera+, MICRO'22]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

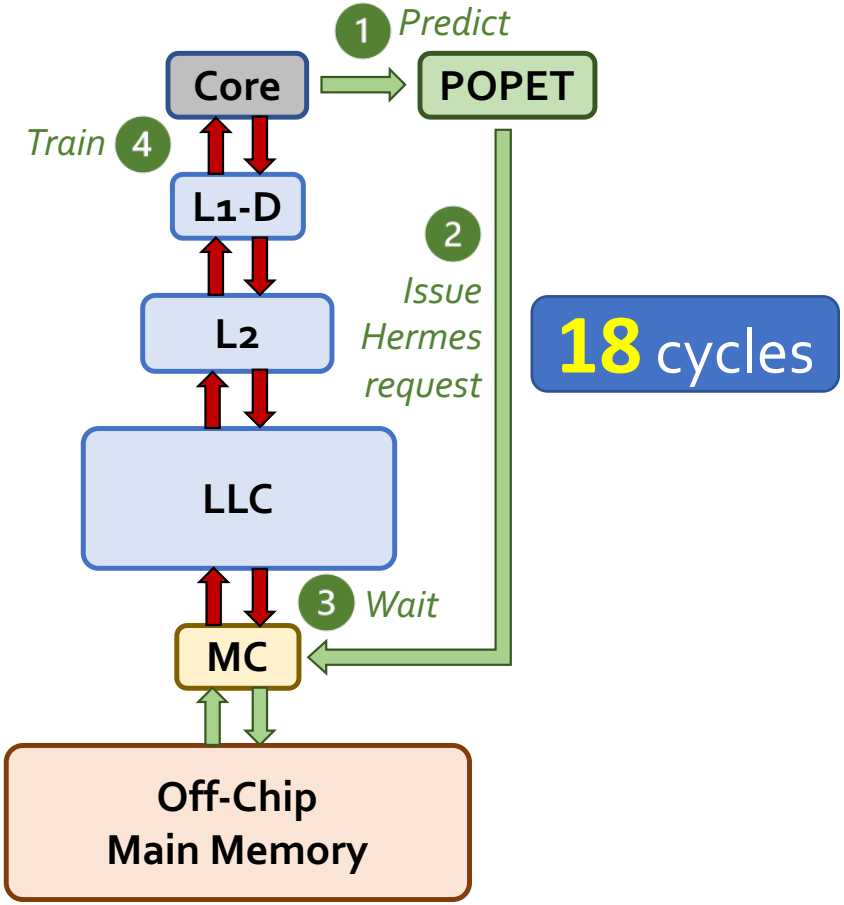
Off-Chip Predictors

- HMP [Yoaz+, ISCA'99]
- Address Tag-Tracking based Predictor (TTP)

Simulation Methodology

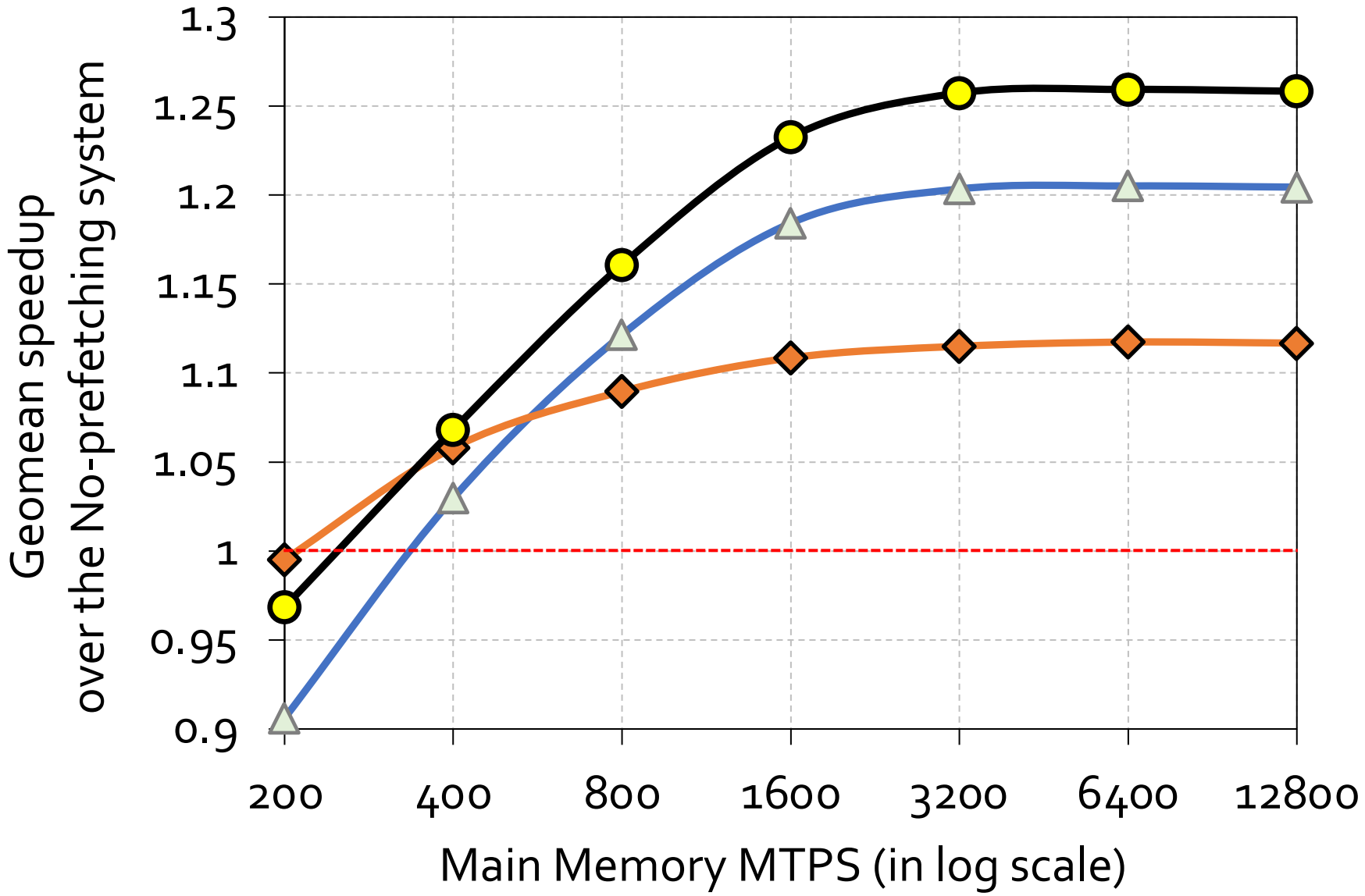


Hermes-**Optimistic**

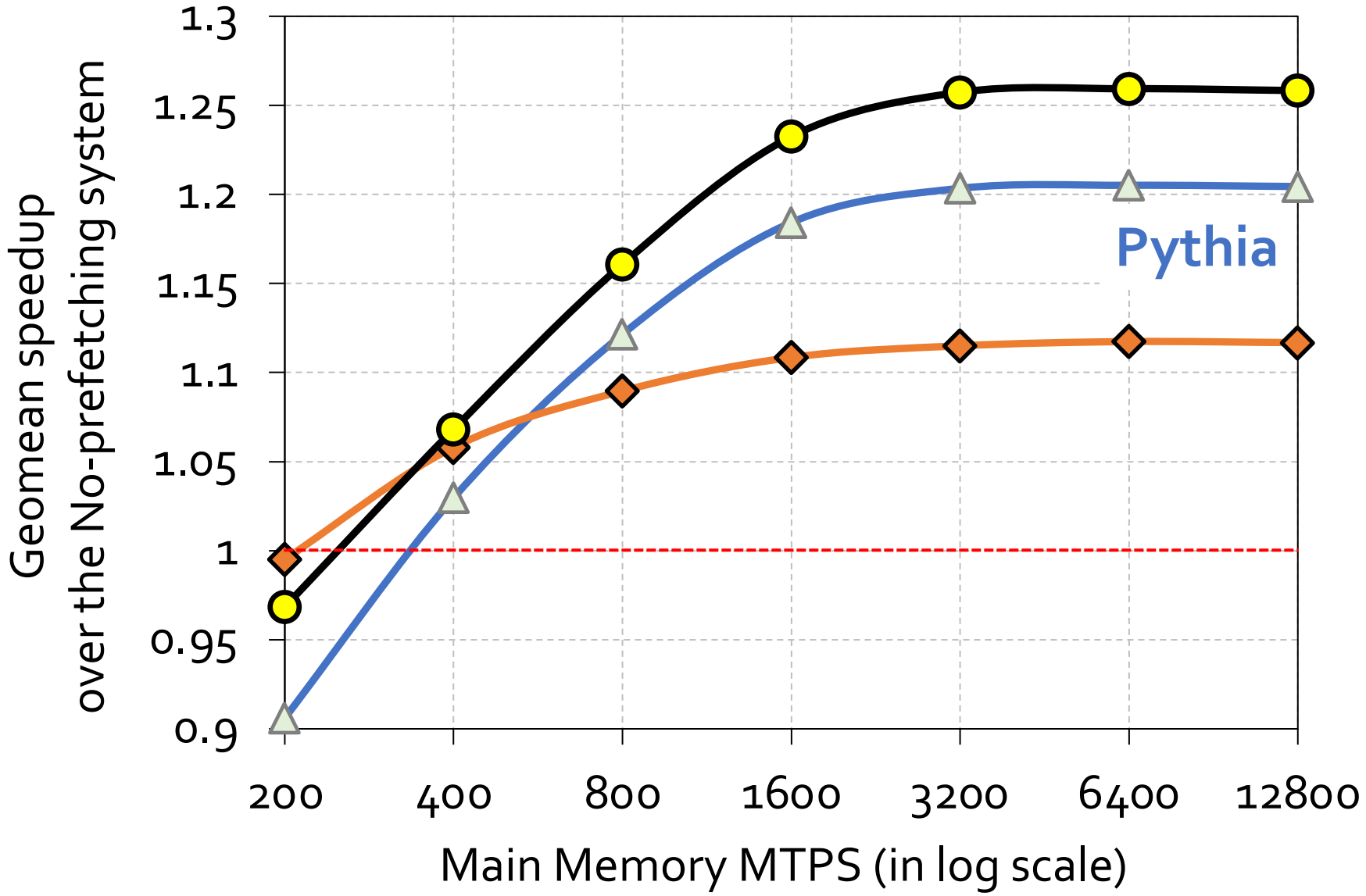


Hermes-**Pessimistic**

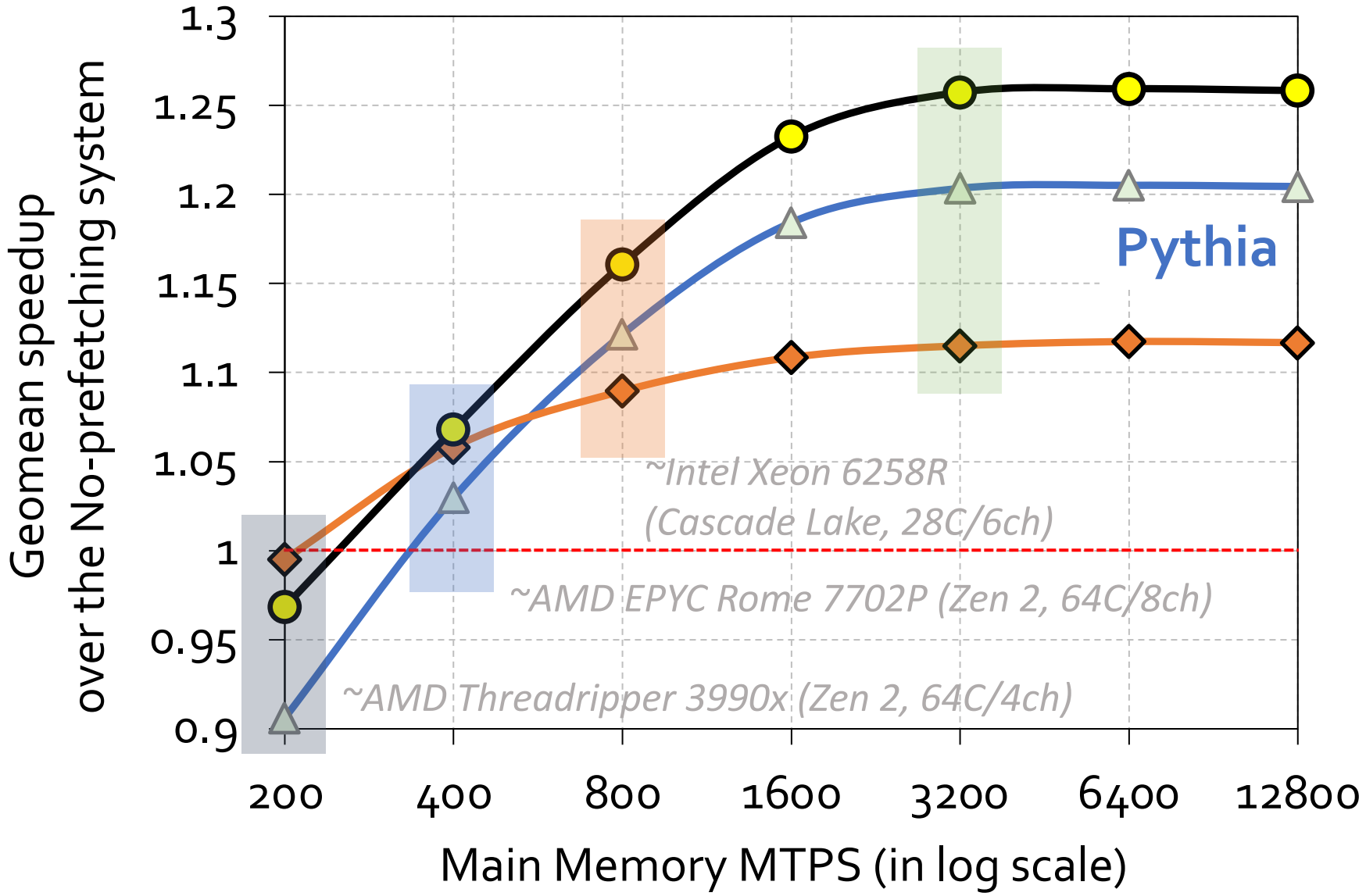
Performance with Varying Memory Bandwidth



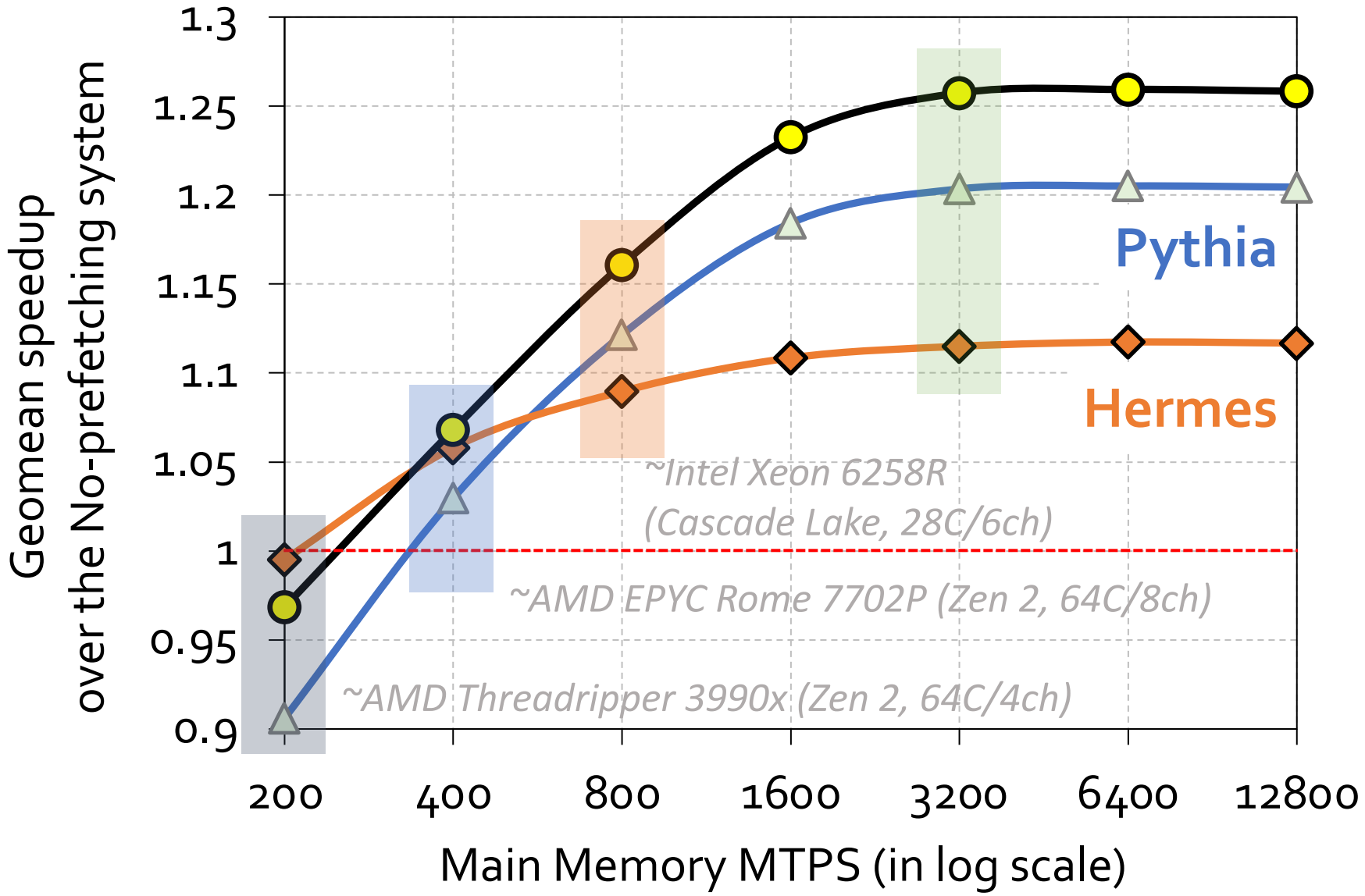
Performance with Varying Memory Bandwidth



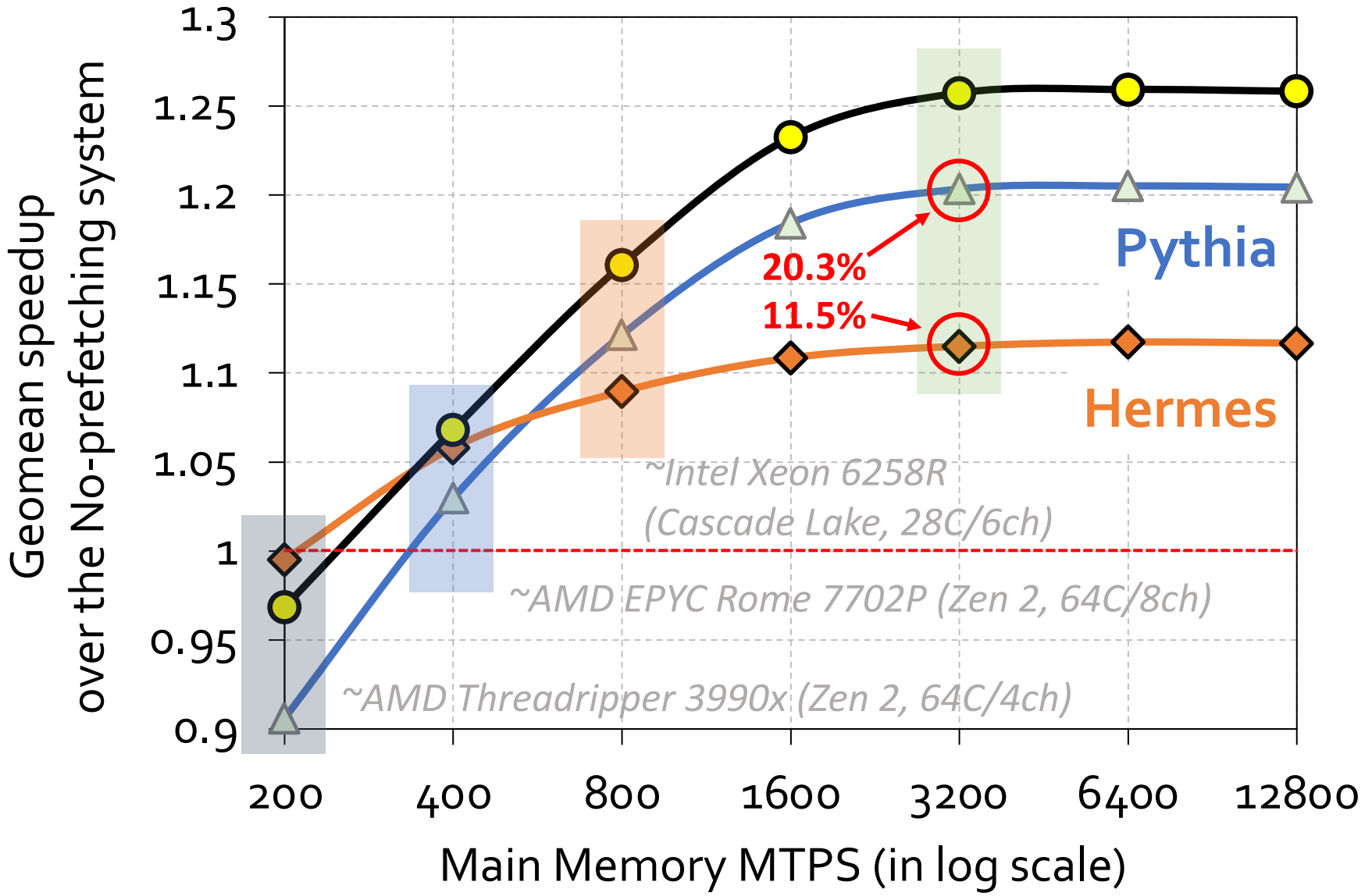
Performance with Varying Memory Bandwidth



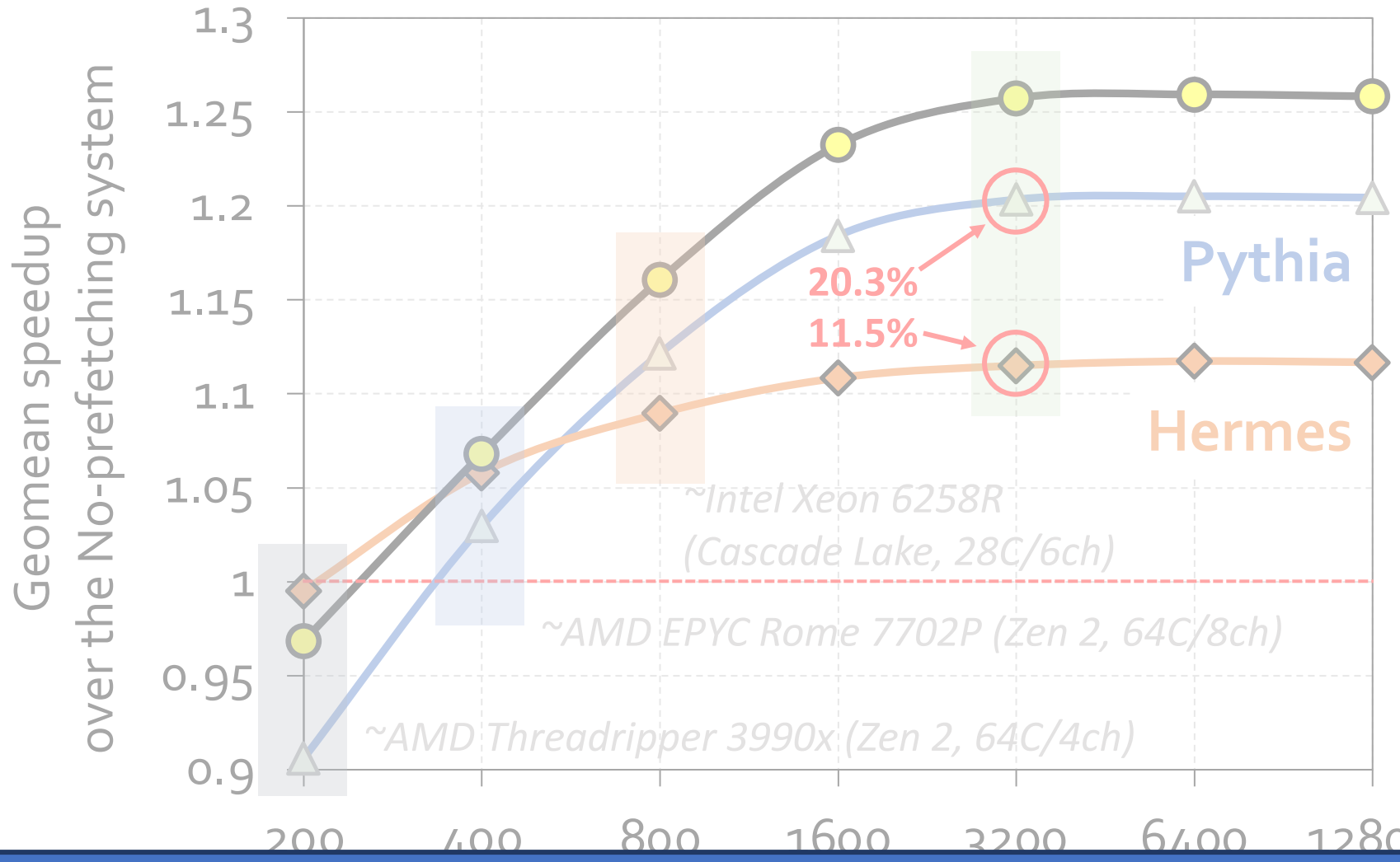
Performance with Varying Memory Bandwidth



Performance with Varying Memory Bandwidth

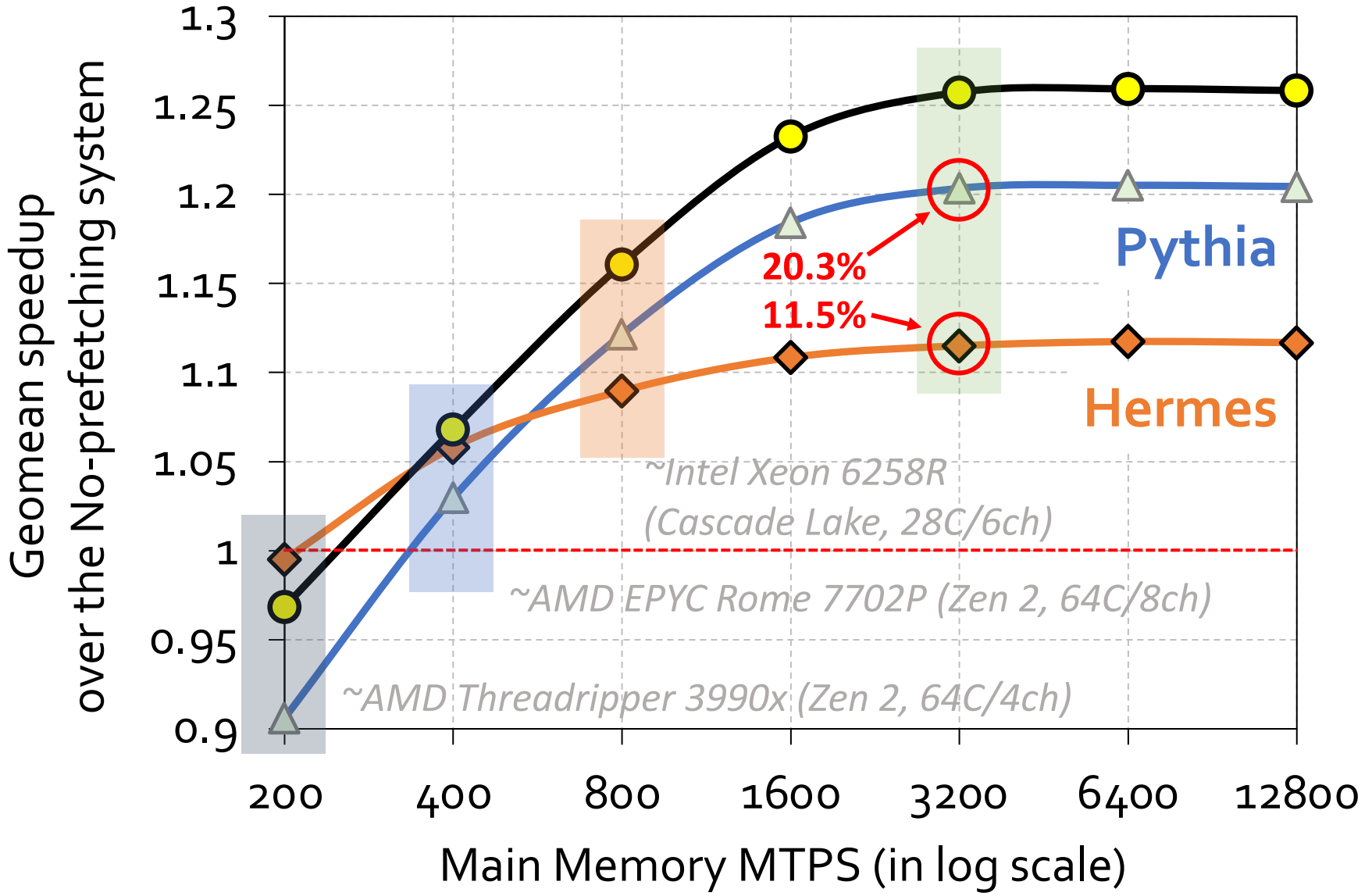


Performance with Varying Memory Bandwidth

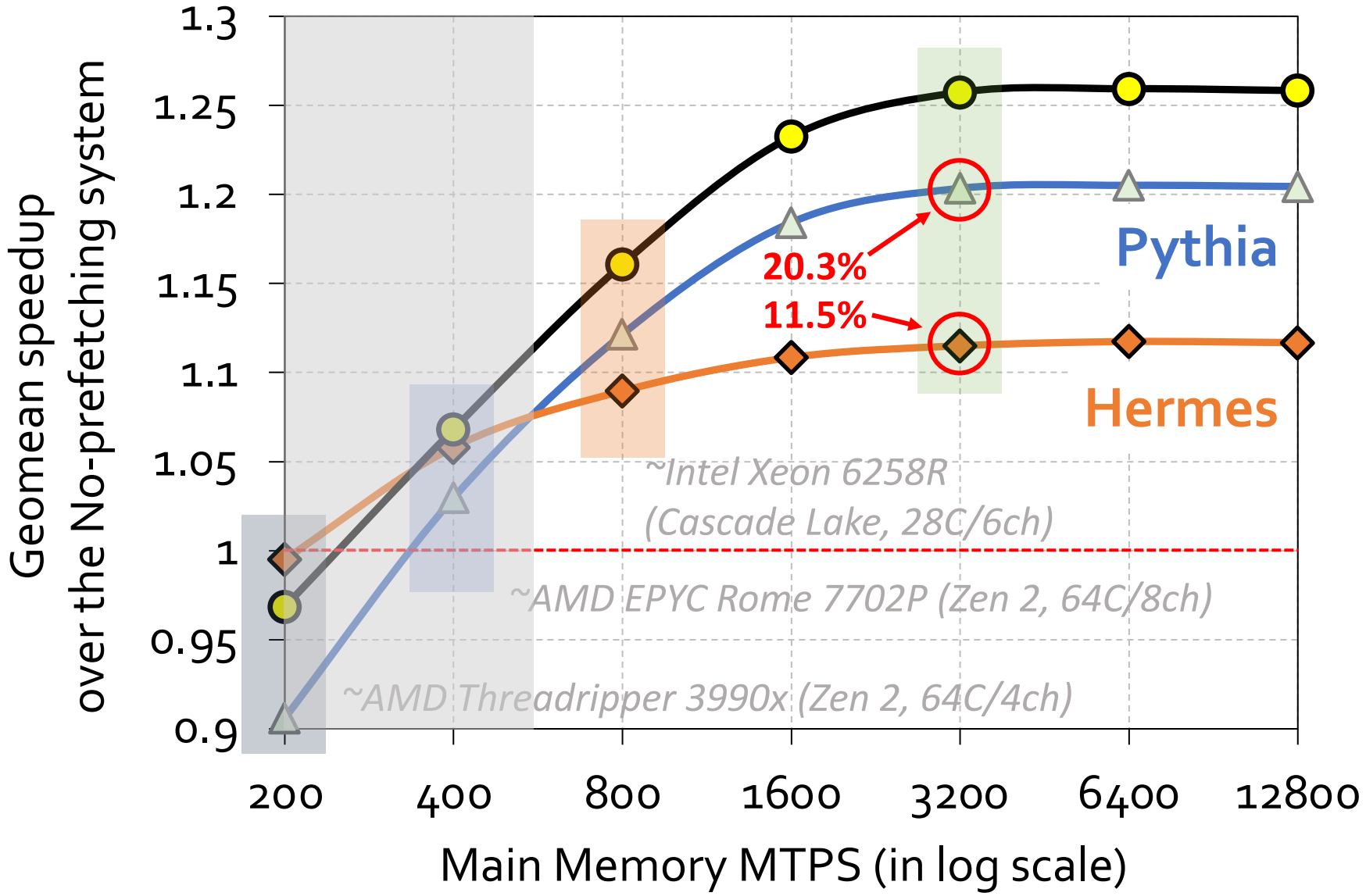


Hermes alone provides nearly **50%** performance of **Pythia**

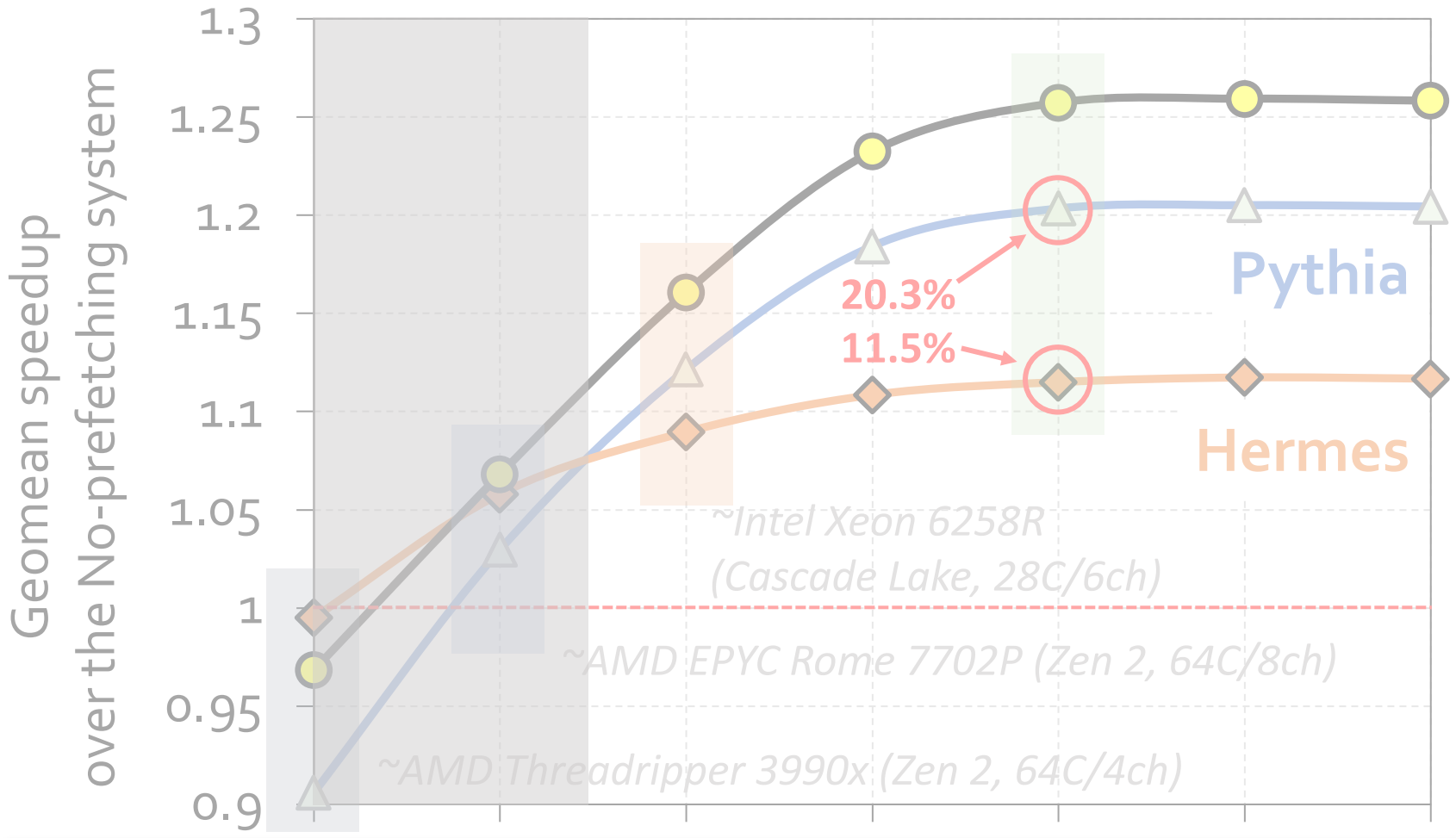
Performance with Varying Memory Bandwidth



Performance with Varying Memory Bandwidth

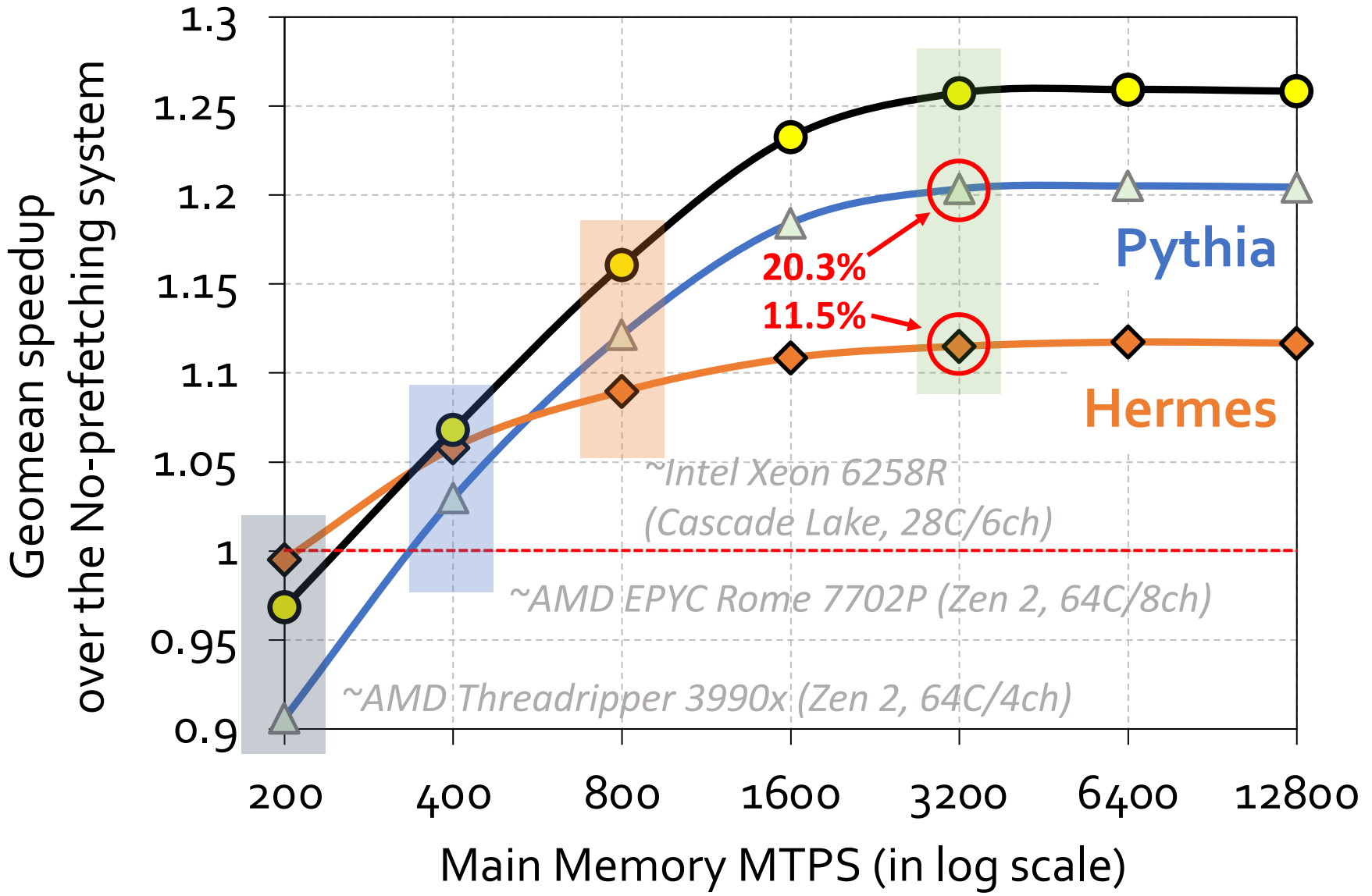


Performance with Varying Memory Bandwidth

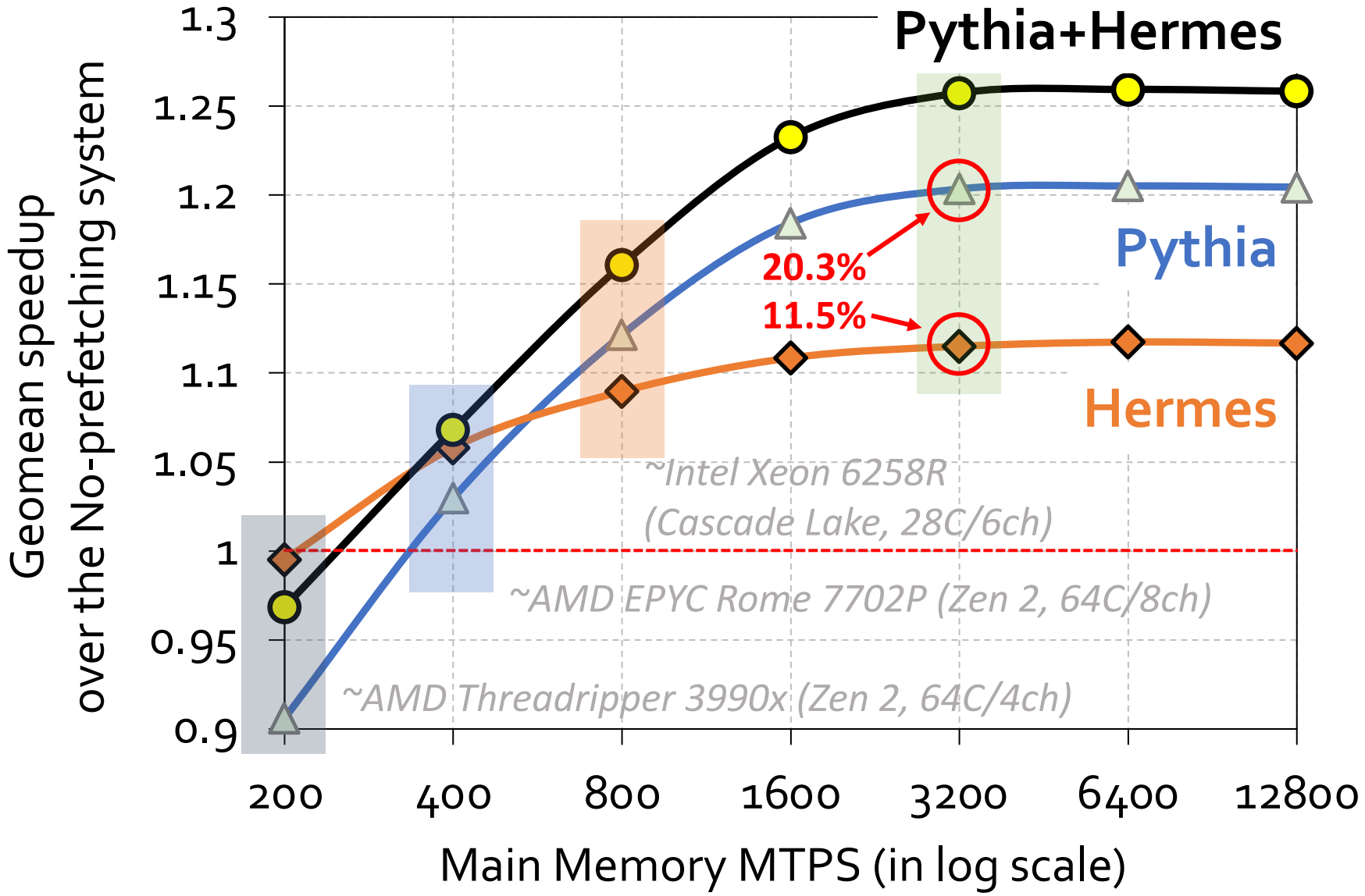


In **extremely bandwidth-constrained** configurations, **Hermes alone outperforms Pythia**

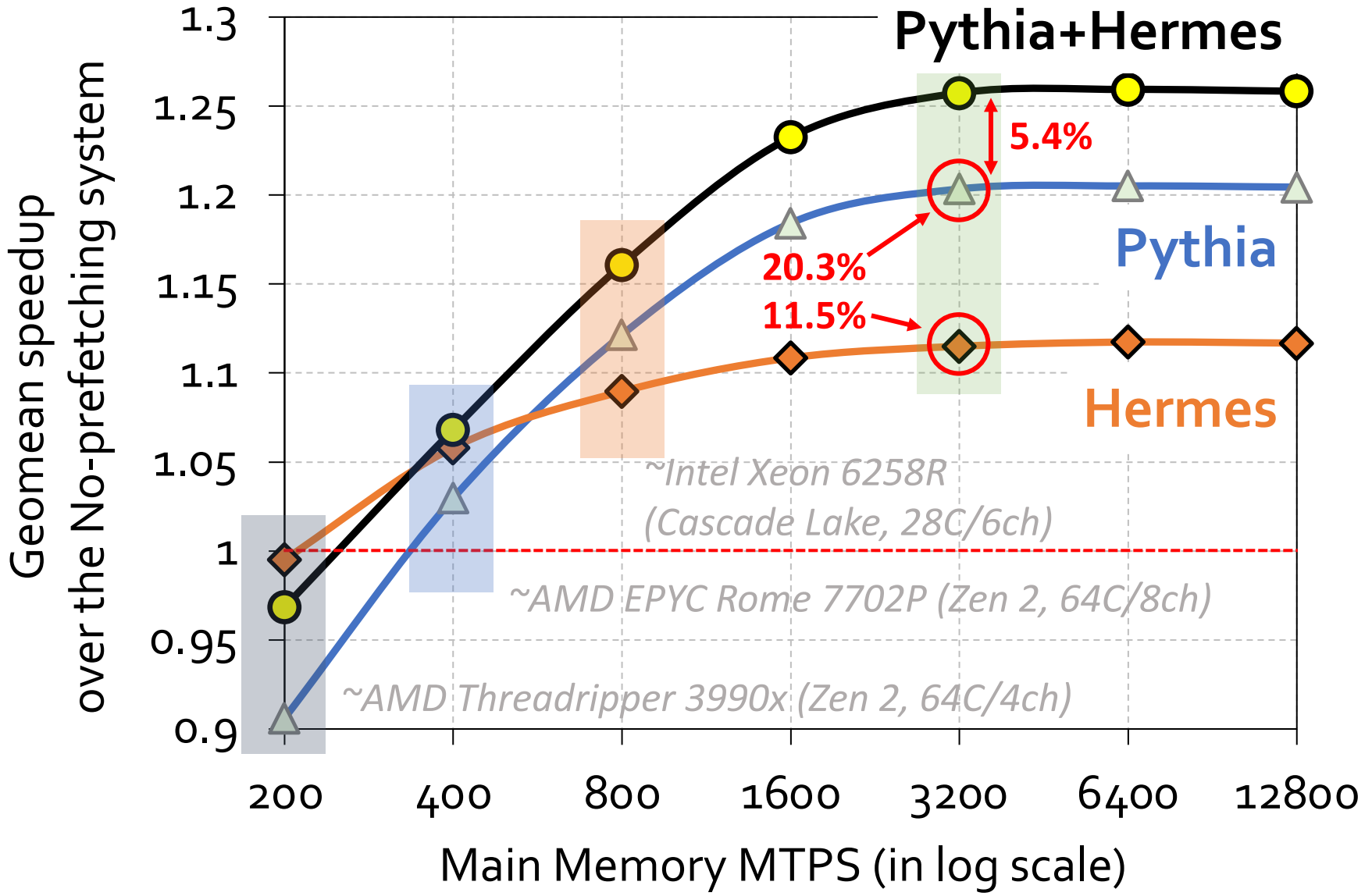
Performance with Varying Memory Bandwidth



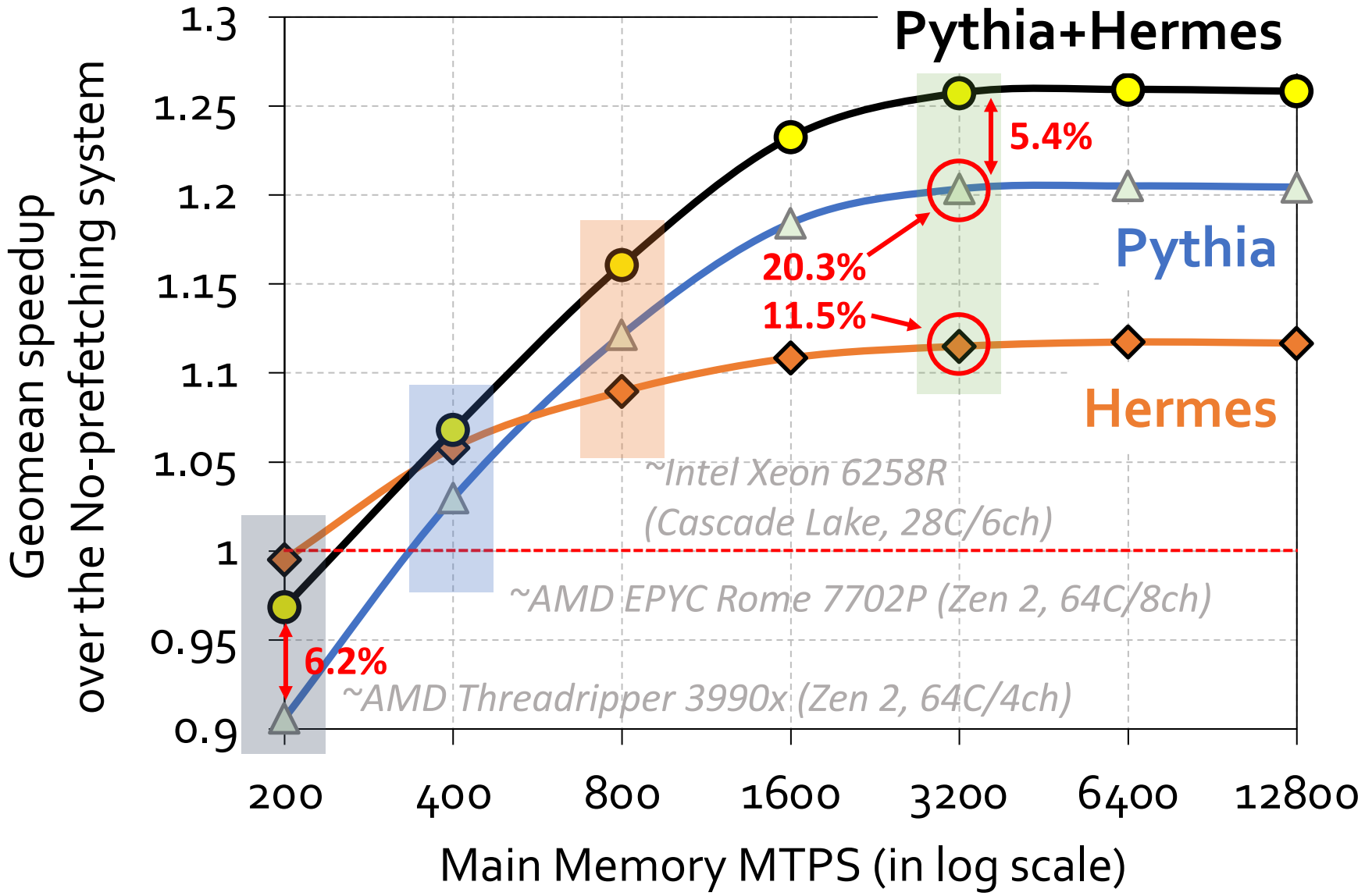
Performance with Varying Memory Bandwidth



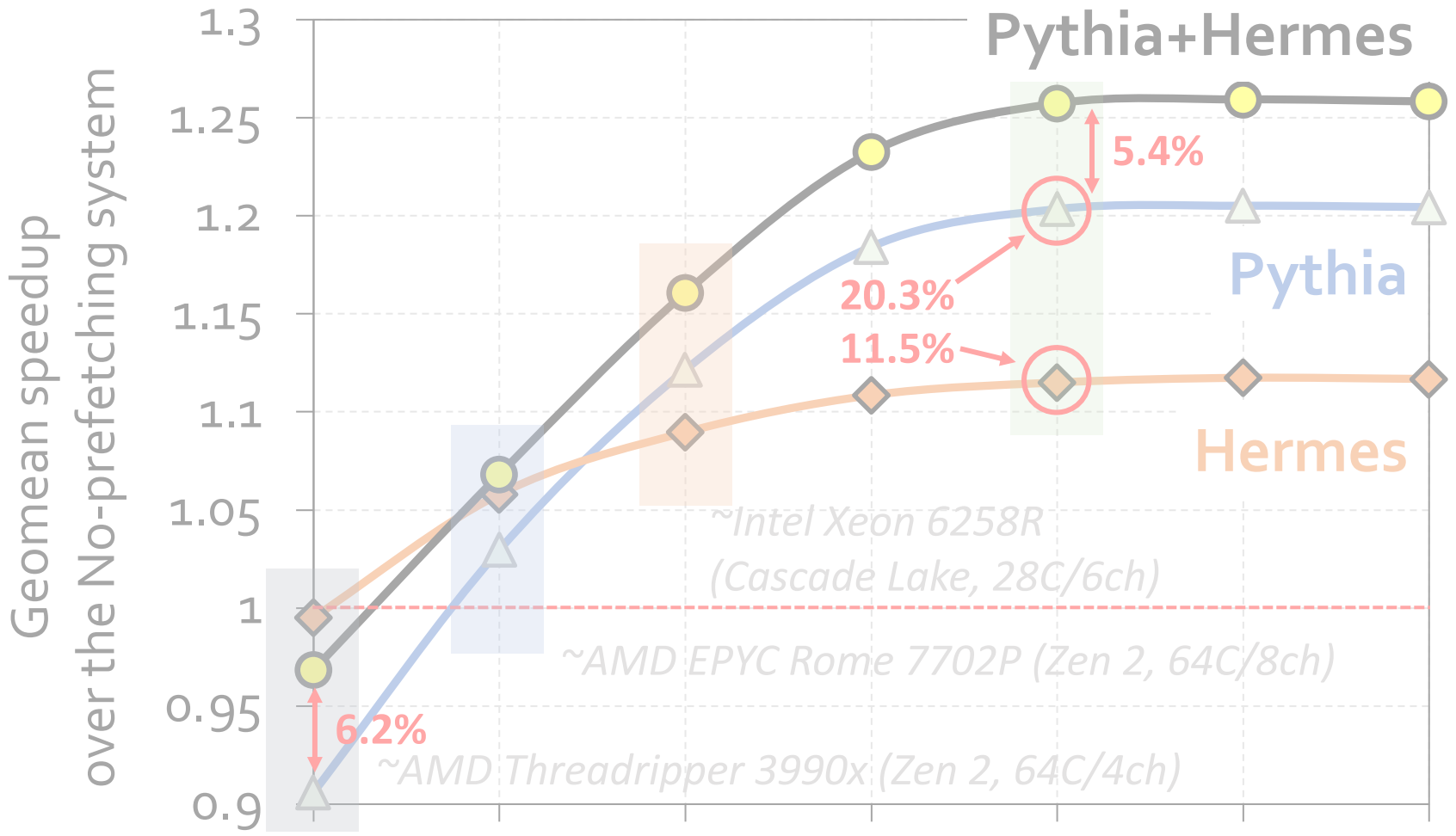
Performance with Varying Memory Bandwidth



Performance with Varying Memory Bandwidth

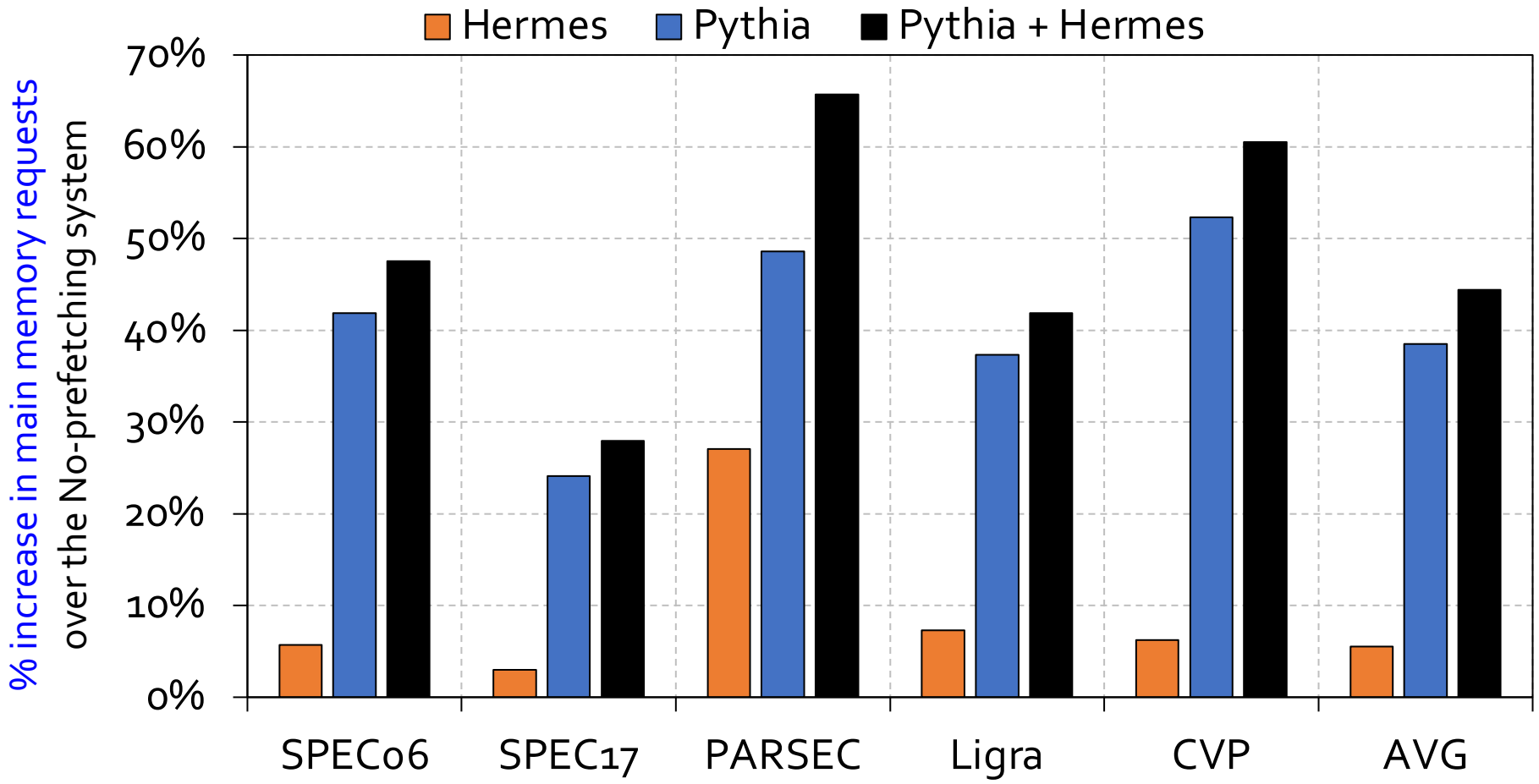


Performance with Varying Memory Bandwidth

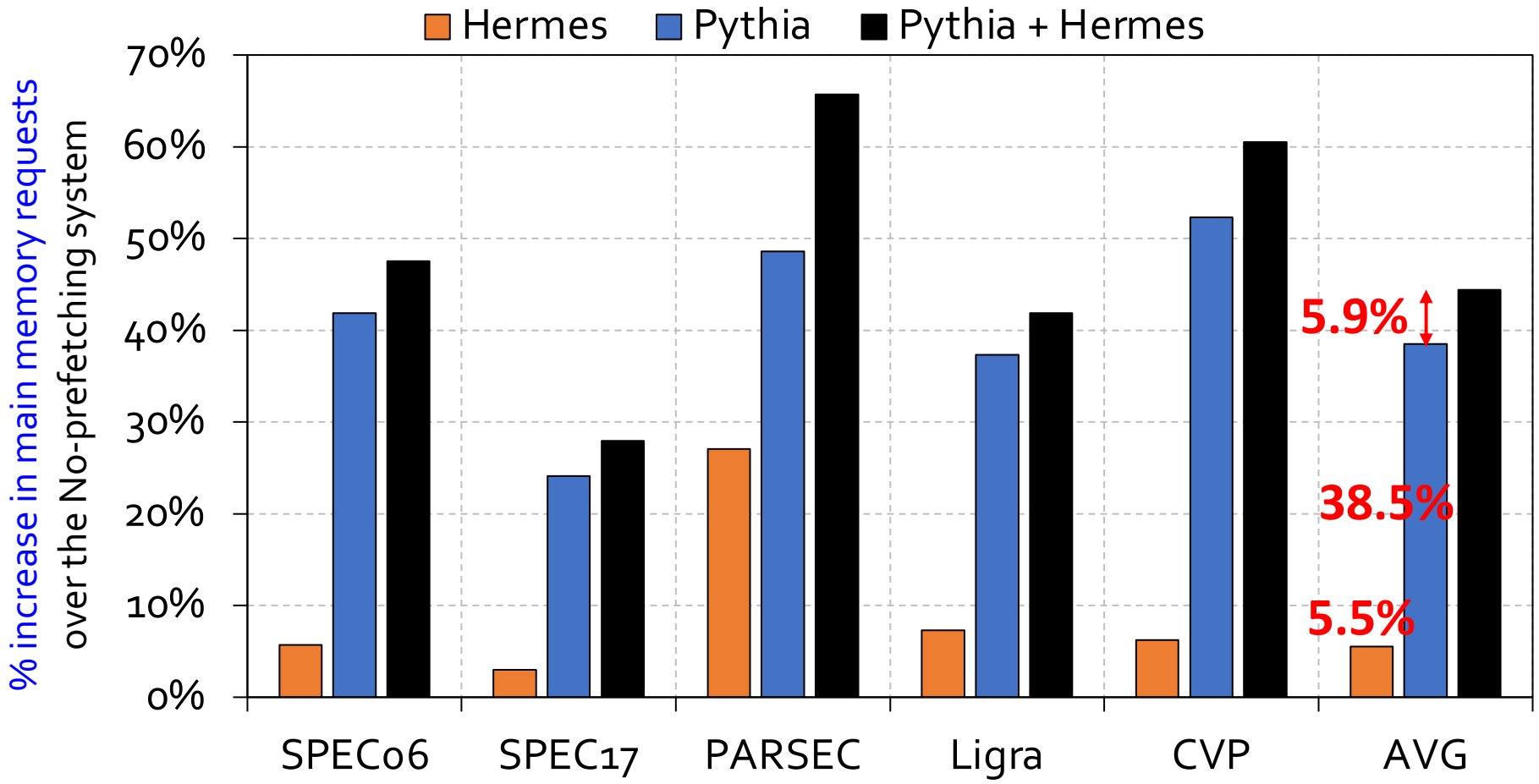


Hermes+Pythia outperforms Pythia
in a wide range of bandwidth configurations

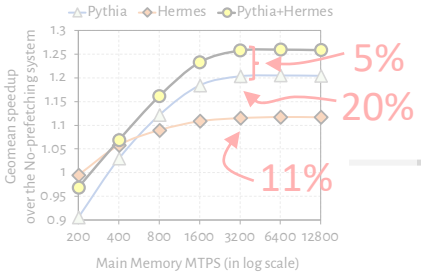
Bandwidth Efficiency

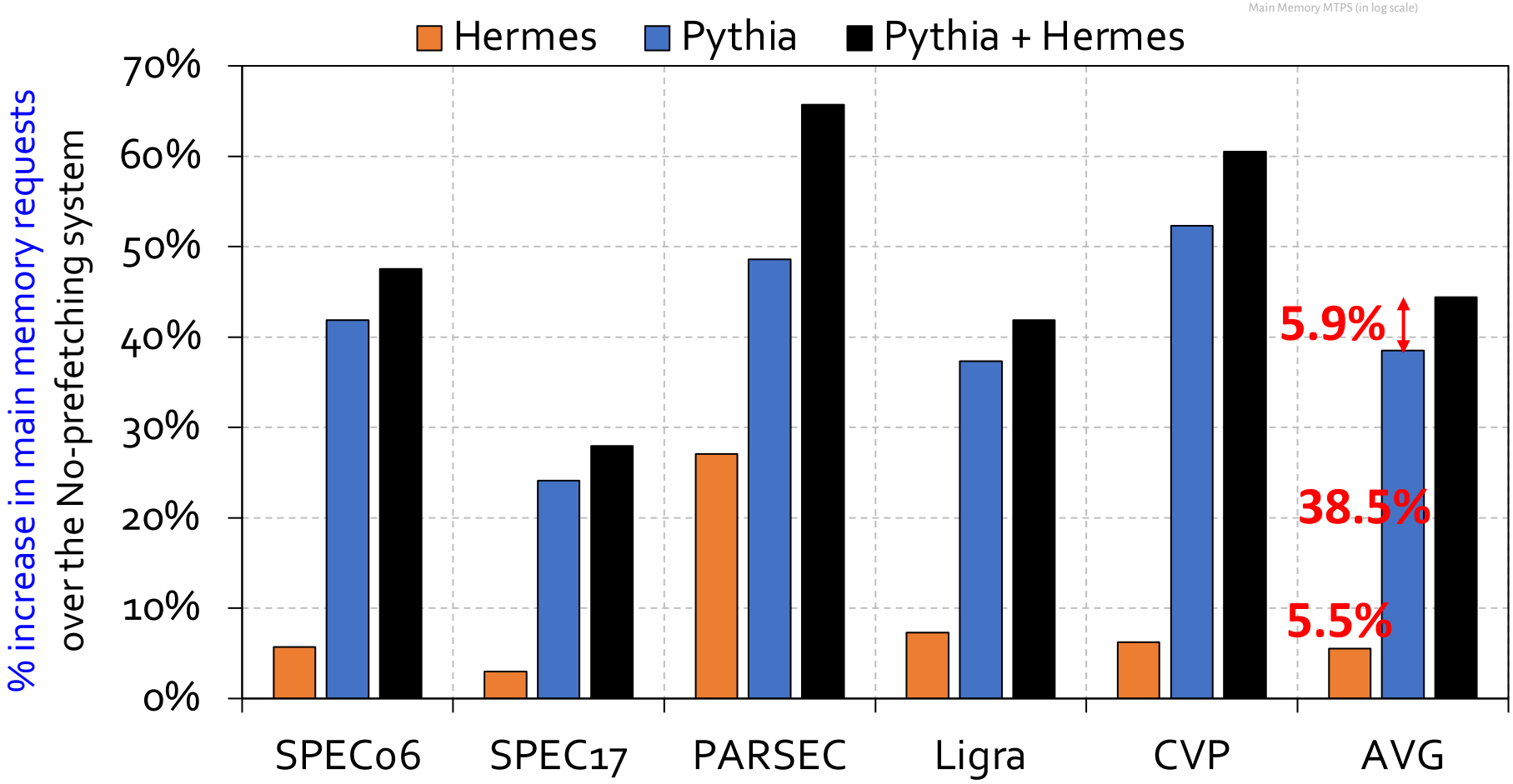


Bandwidth Efficiency

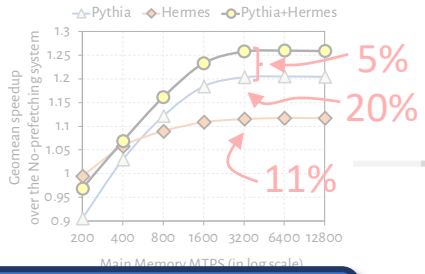


Bandwidth Efficiency

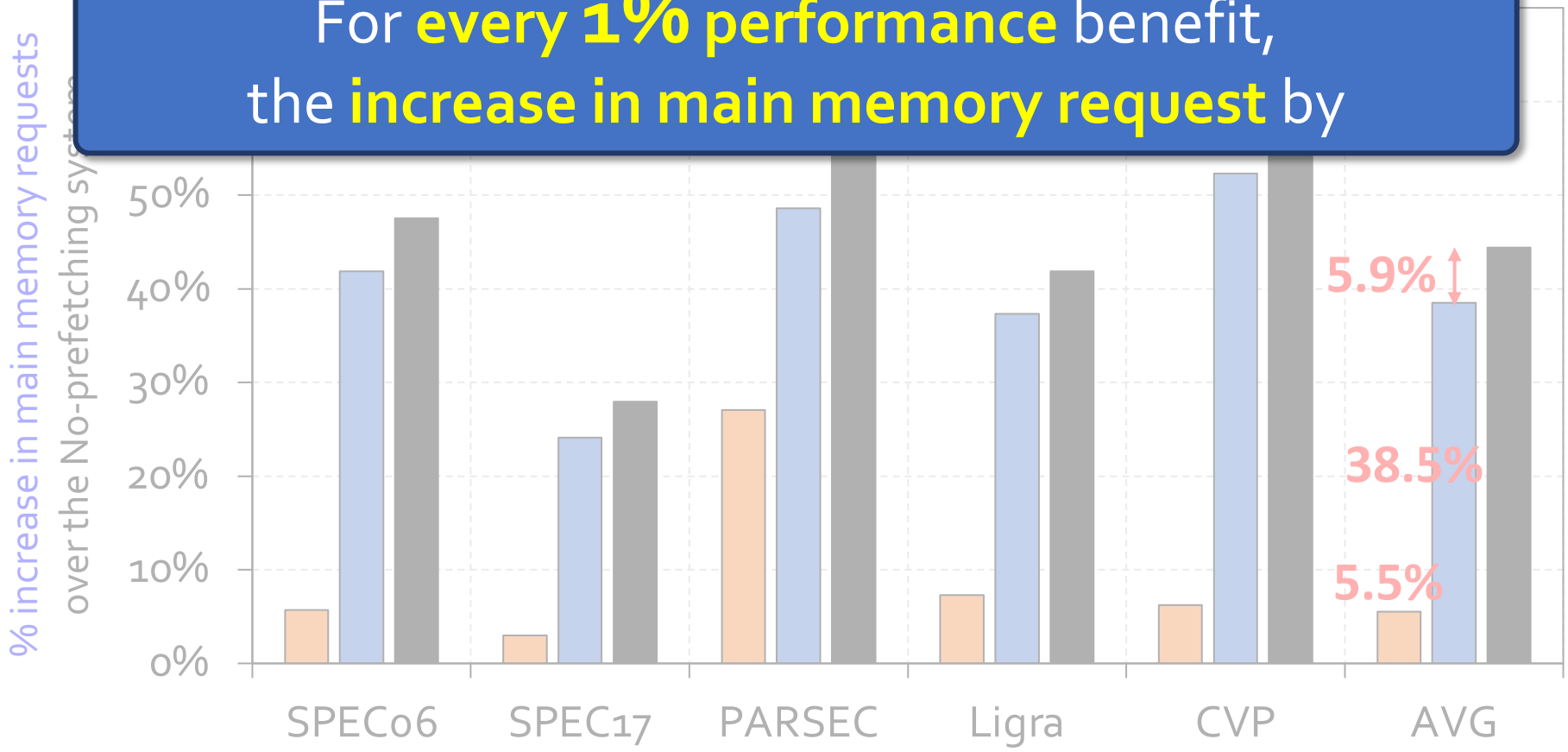




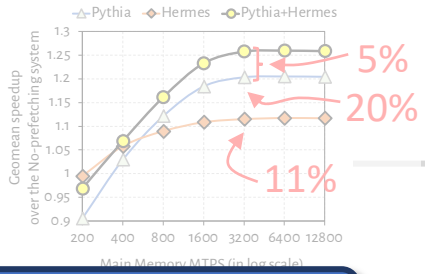
Bandwidth Efficiency



For **every 1% performance** benefit,
the **increase in main memory request** by



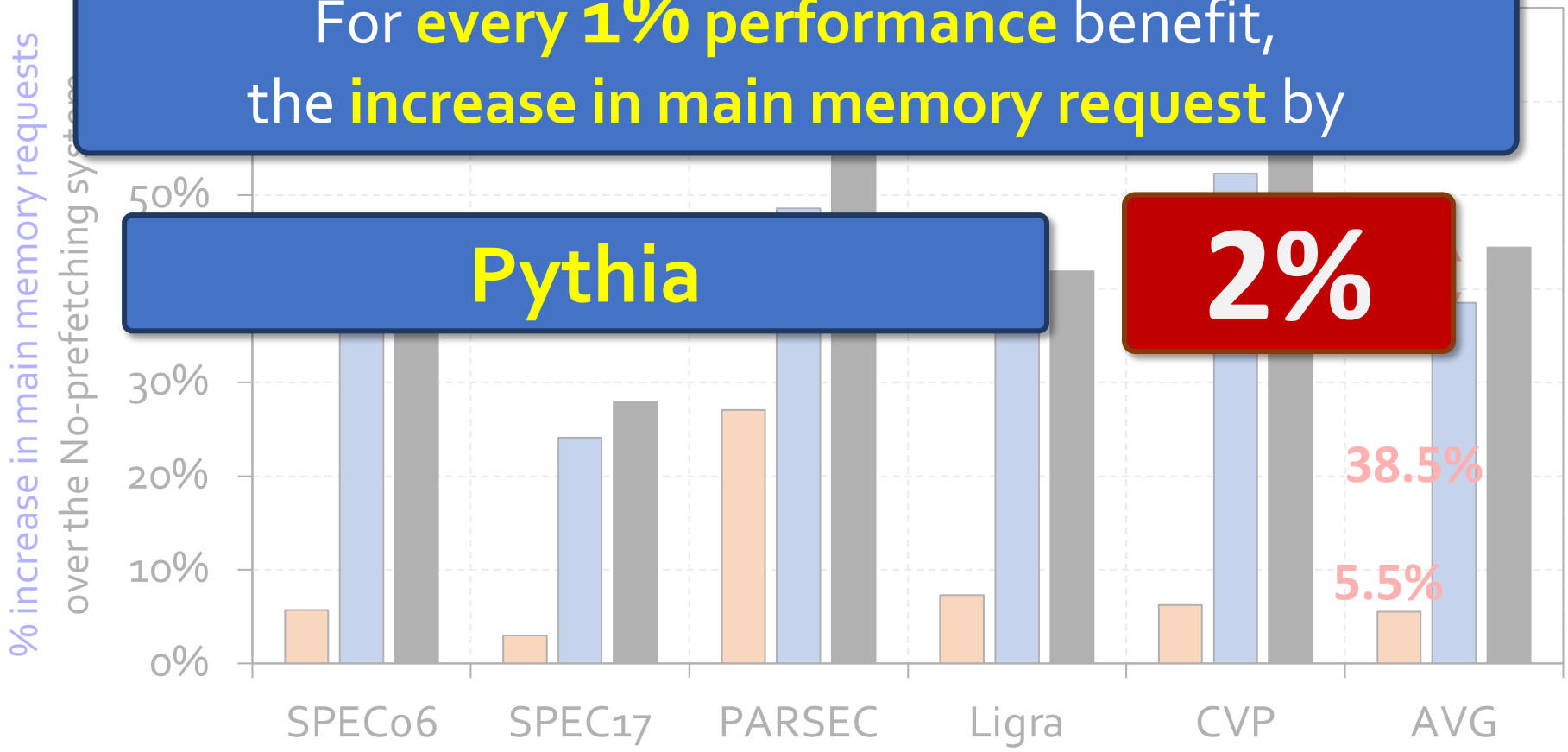
Bandwidth Efficiency



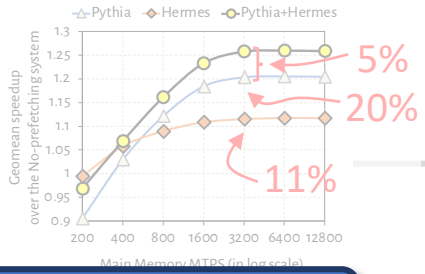
For **every 1% performance** benefit, the **increase in main memory request** by

Pythia

2%



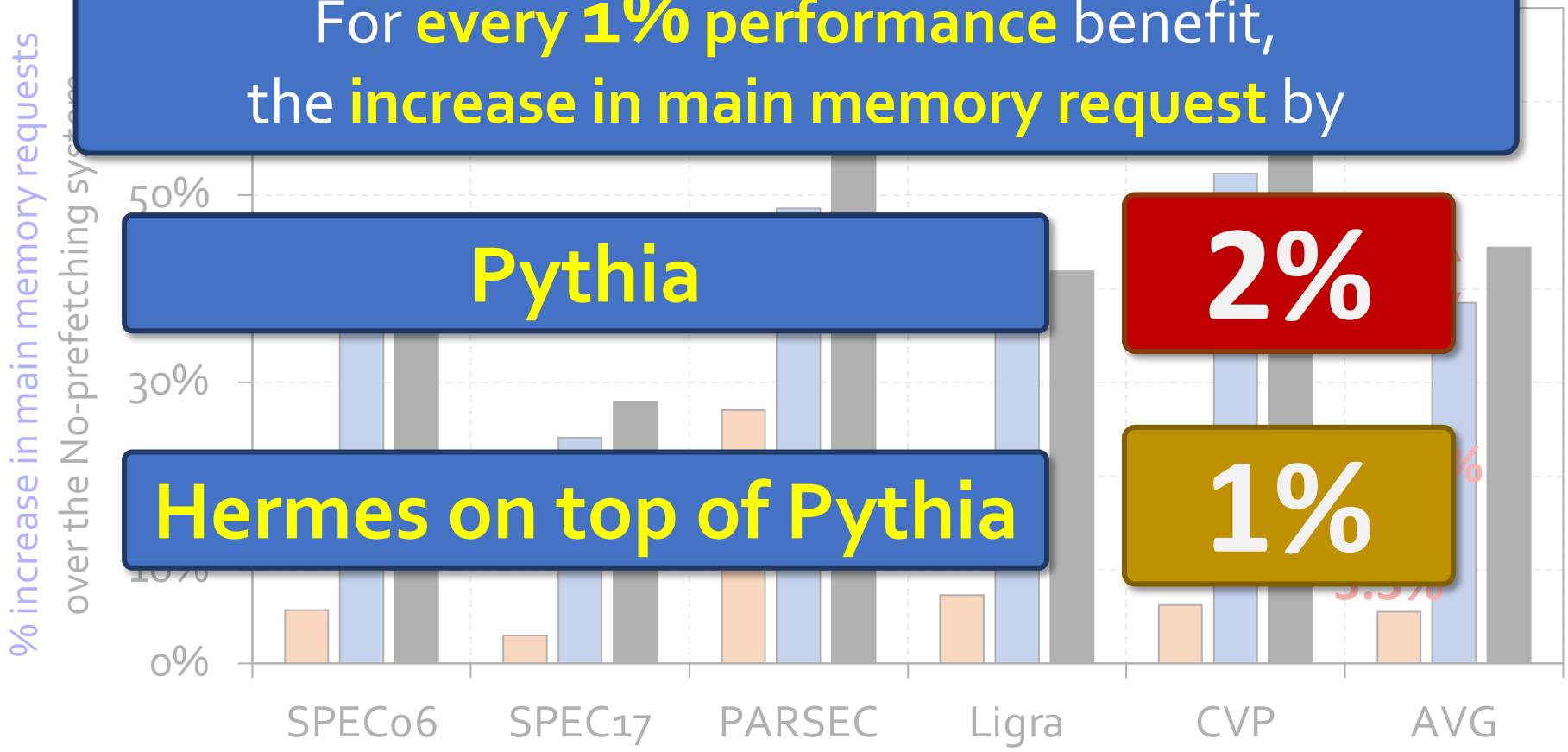
Bandwidth Efficiency



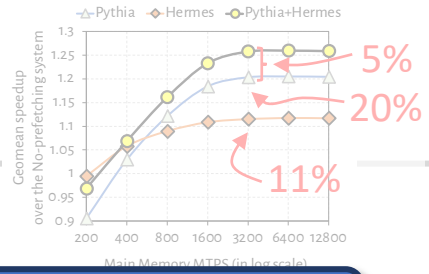
For **every 1% performance** benefit, the **increase in main memory request** by

Pythia **2%**

Hermes on top of Pythia **1%**



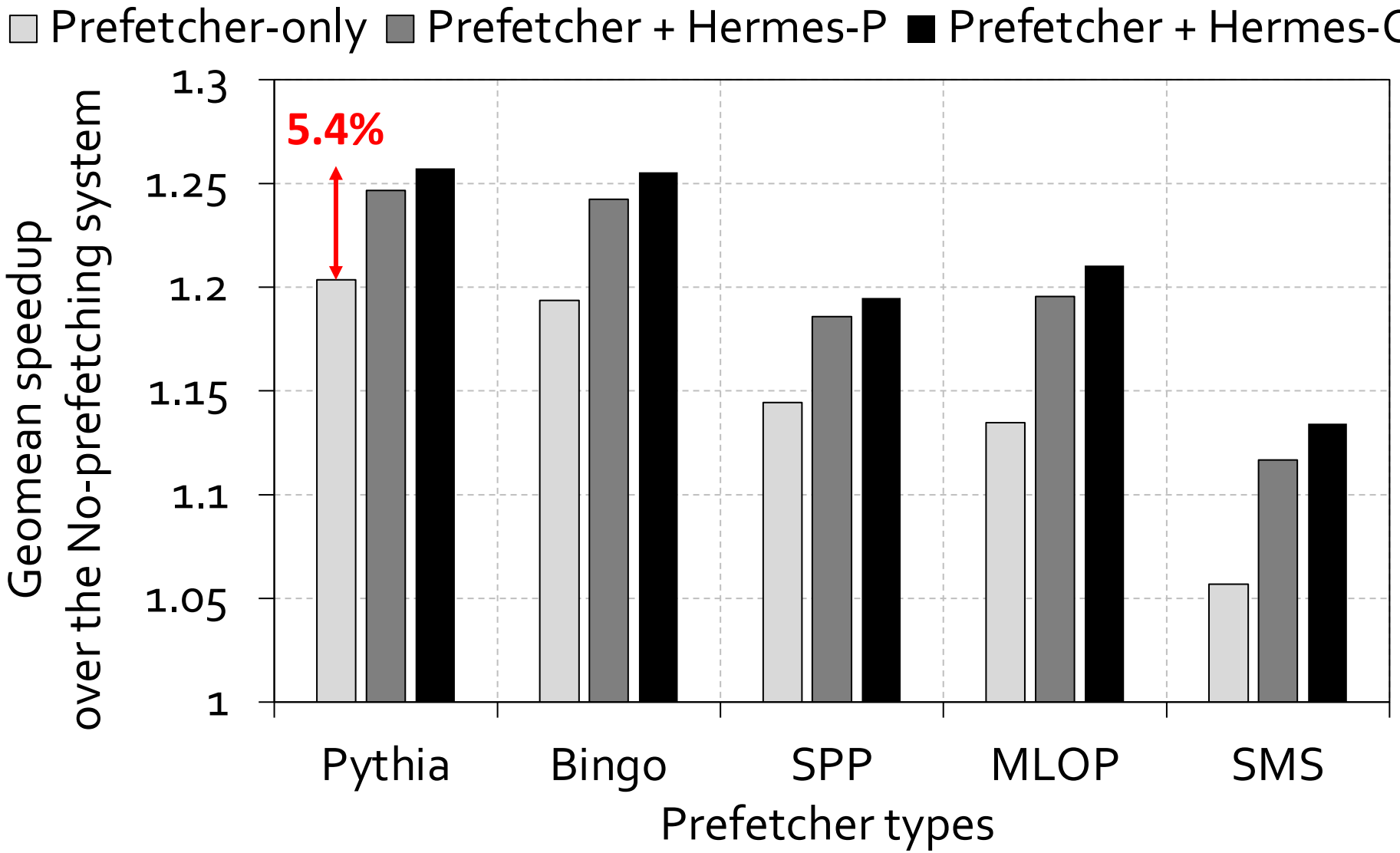
Bandwidth Efficiency



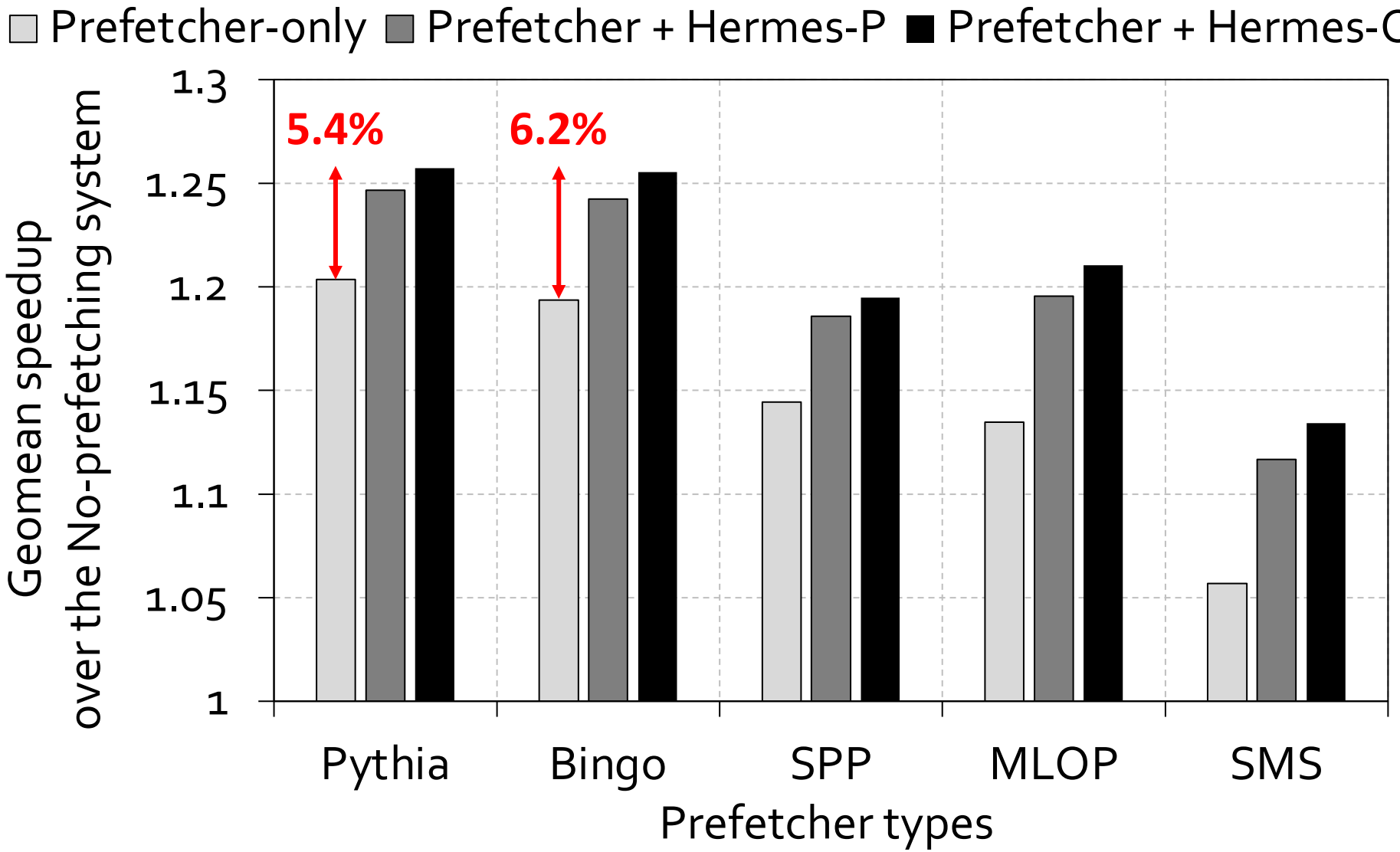
For **every 1% performance** benefit, the **increase in main memory request** by

Pythia	**2%**
Hermes on top of Pythia	**1%**
Hermes alone	**0.5%**

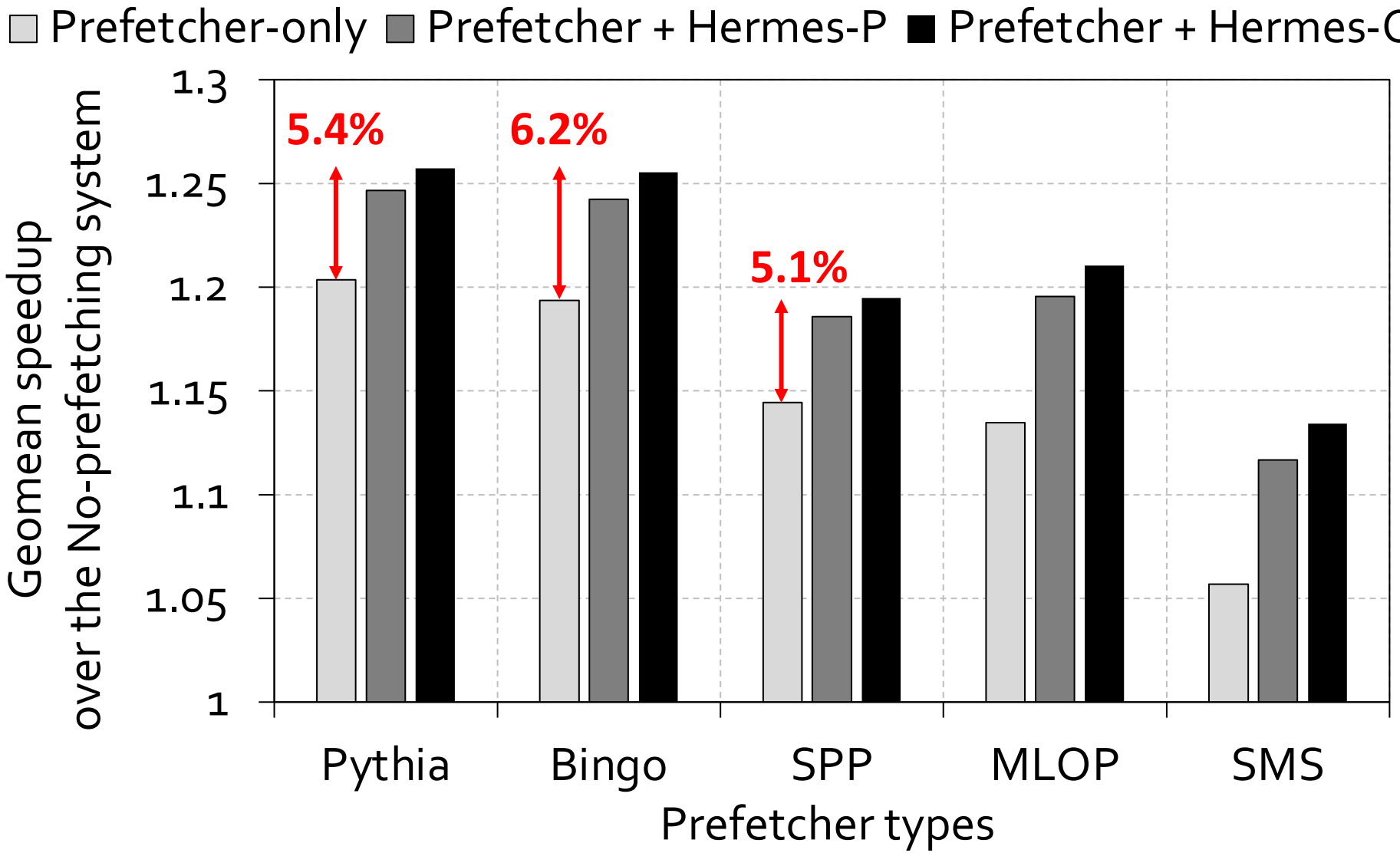
Performance with Varying Baseline Prefetcher



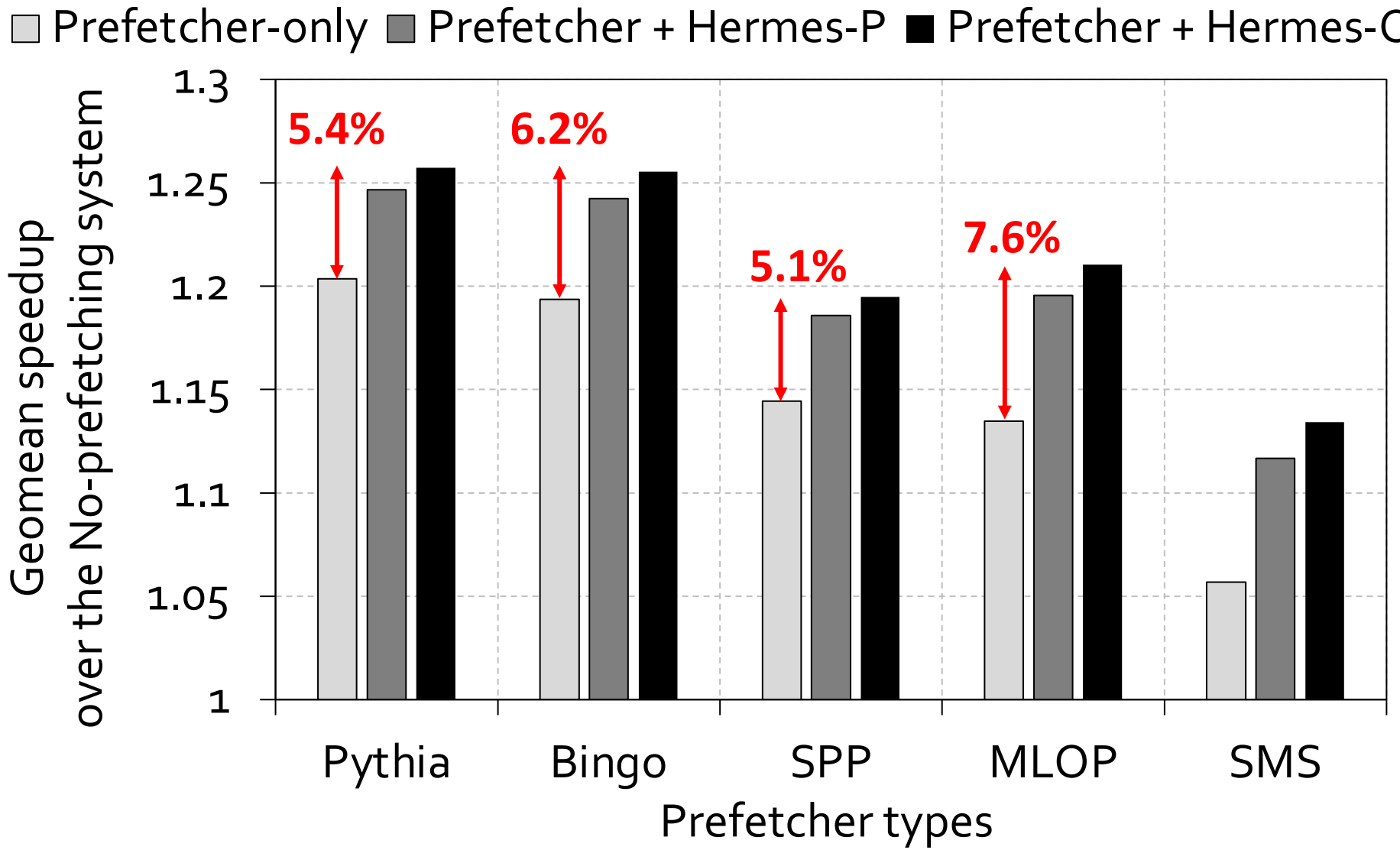
Performance with Varying Baseline Prefetcher



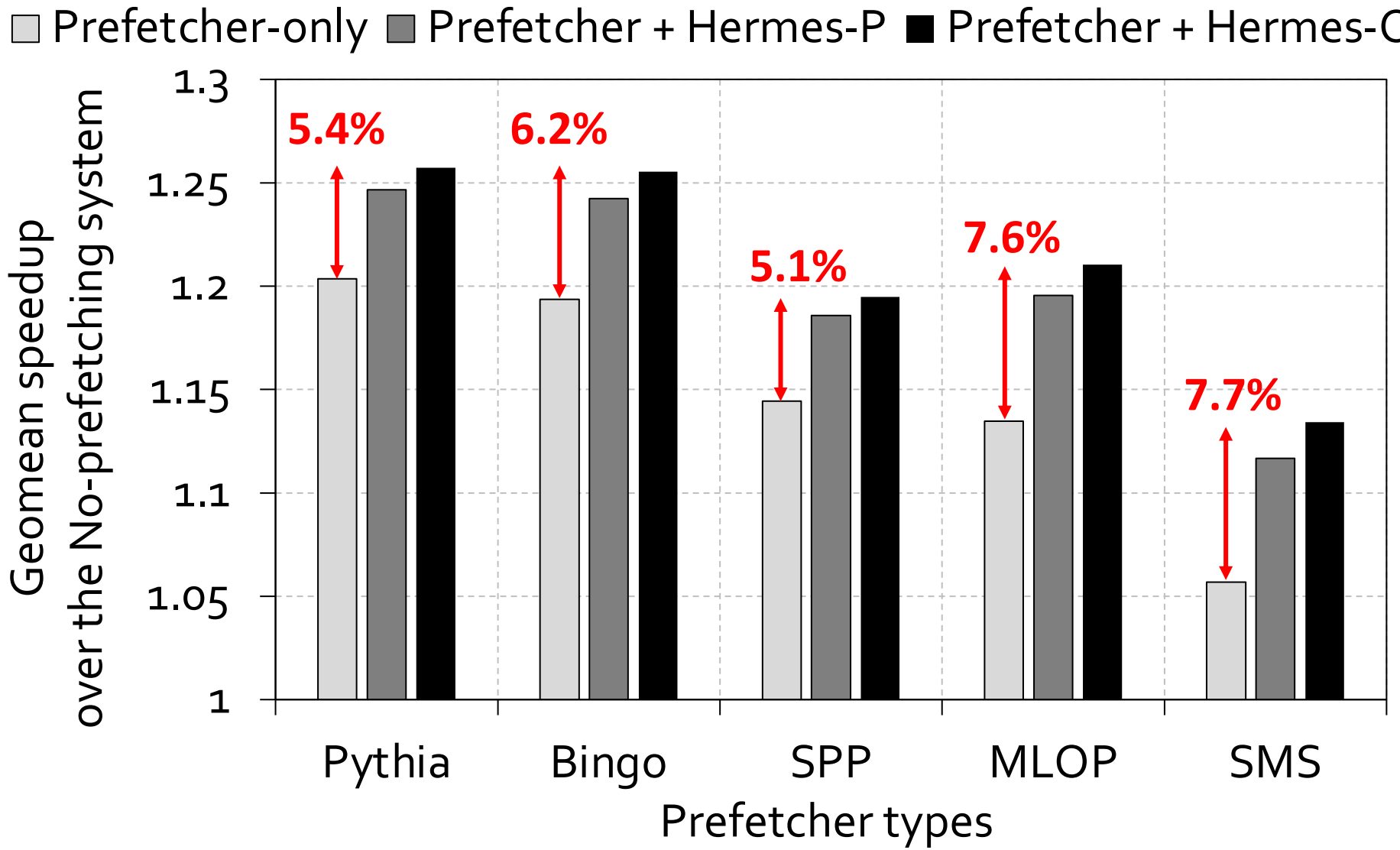
Performance with Varying Baseline Prefetcher



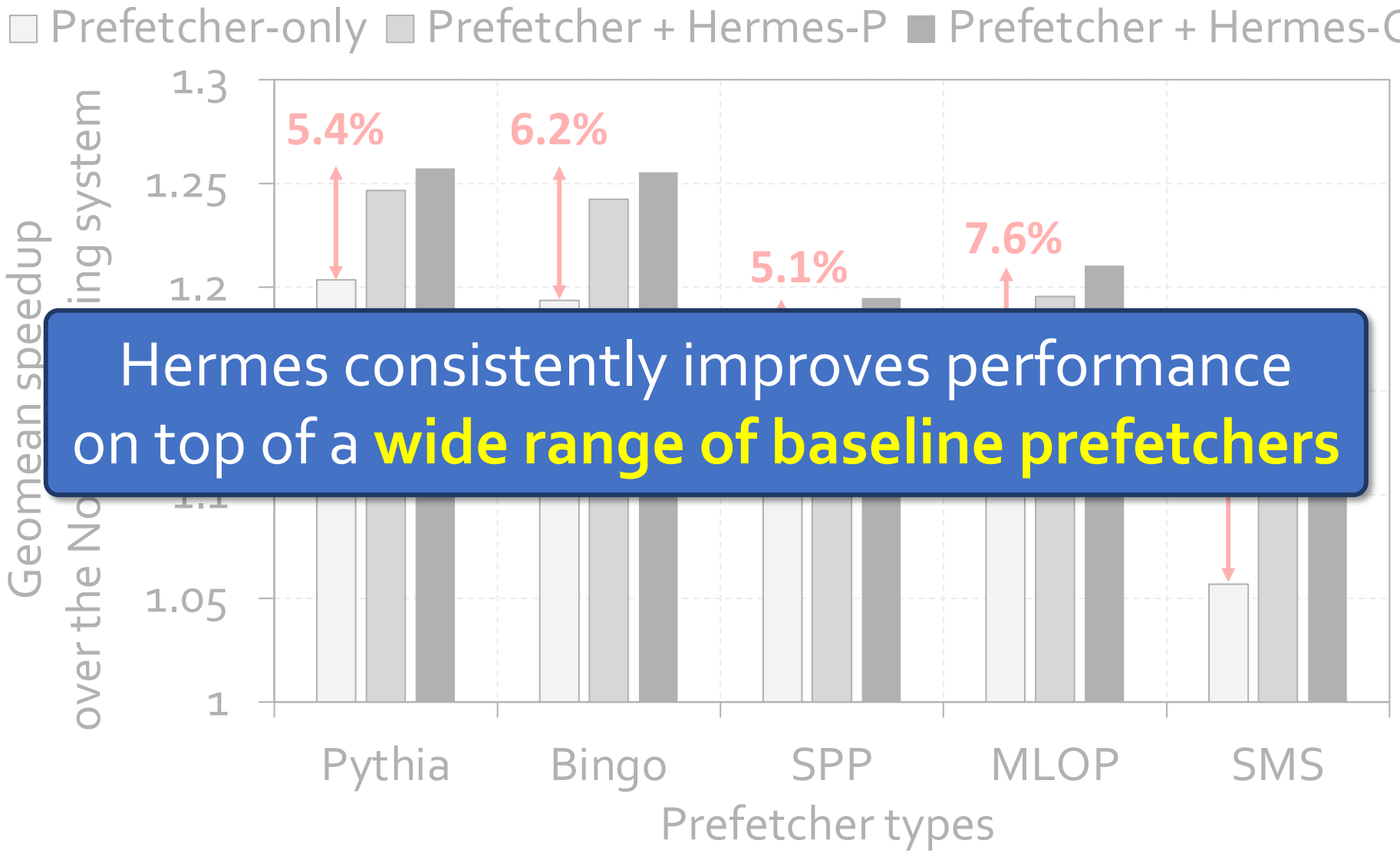
Performance with Varying Baseline Prefetcher



Performance with Varying Baseline Prefetcher



Performance with Varying Baseline Prefetcher



Hermes consistently improves performance on top of a **wide range of baseline prefetchers**

Overhead of Hermes

Overhead of Hermes

4 KB storage overhead

Overhead of Hermes

4 KB storage overhead

1.5% power overhead

On top of an Intel Alder Lake-like performance-core [2] configuration

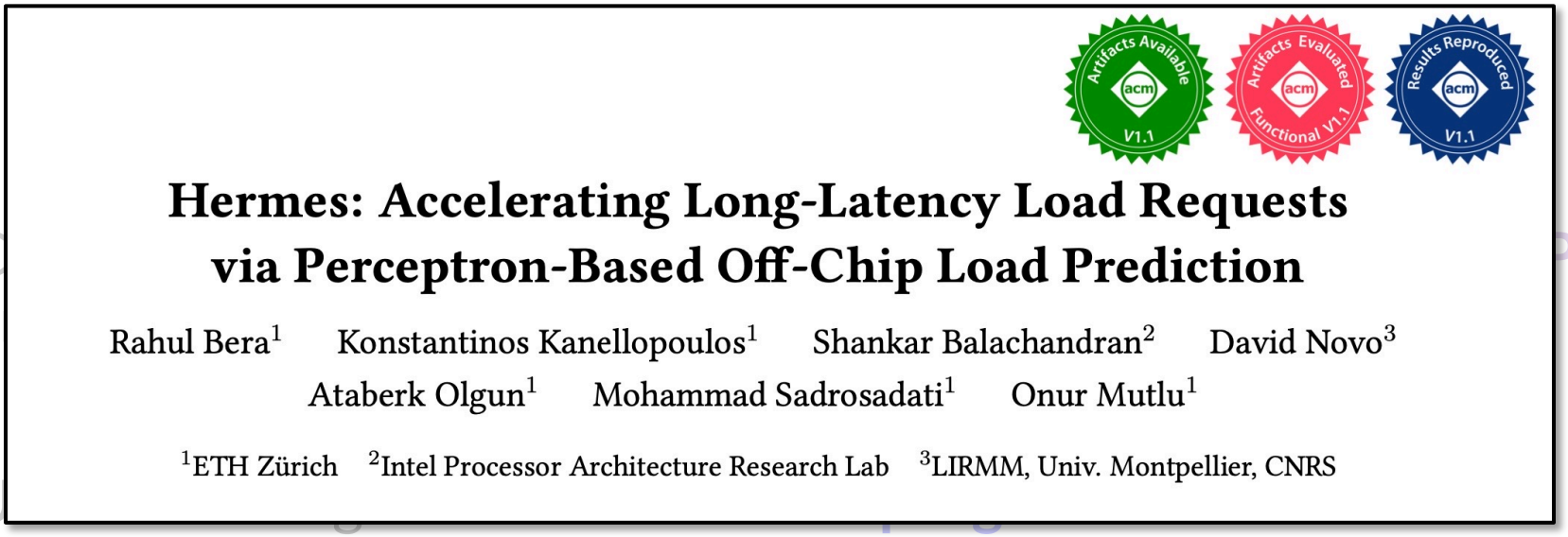
[2] https://www.anandtech.com/show/16881/a-deep-dive-into-intels-alder-lake-microarchitectures/3

More in the Paper

- Wide range of sensitivity studies by varying
  - Cache hierarchy access latency
  - Hermes request issue latency
  - Activation threshold
- Comparison of POPET's **accuracy** and **coverage** against **HMP** and **TTP**
- Understanding **usefulness of each program feature**
- Hermes's effect on **stall cycle reduction**
- **Performance** analysis in **eight-core** system

More in the Paper

- Wide range of sensitivity studies by varying
  - Cache hierarchy access latency
- C… a…
- U…
- Hermes's effect on stall cycle reduction
- Performance analysis in eight-core system



https://arxiv.org/pdf/2209.00188.pdf

To Summarize…

Summary

Summary

Hermes advocates **off-chip load prediction**,
a different form of speculation than
load address predictions by prefetchers

Summary

Hermes advocates **off-chip load prediction**,
a different form of speculation than
load address predictions by prefetchers

Off-chip load prediction can be applied **by itself**
or **combined with load address prediction**
to provide performance improvement

Summary

Hermes…

Summary

Hermes…

Identifies **74%** off-chip loads,

Summary

Hermes…

Identifies **74%** off-chip loads,

that misses a **4.5 MB** cache hierarchy using only **4 KB** storage overhead,

Summary

Hermes…

Identifies **74%** off-chip loads,

that misses a **4.5 MB** cache hierarchy using only **4 KB** storage overhead,

with **77%** accuracy,

Summary

Hermes…

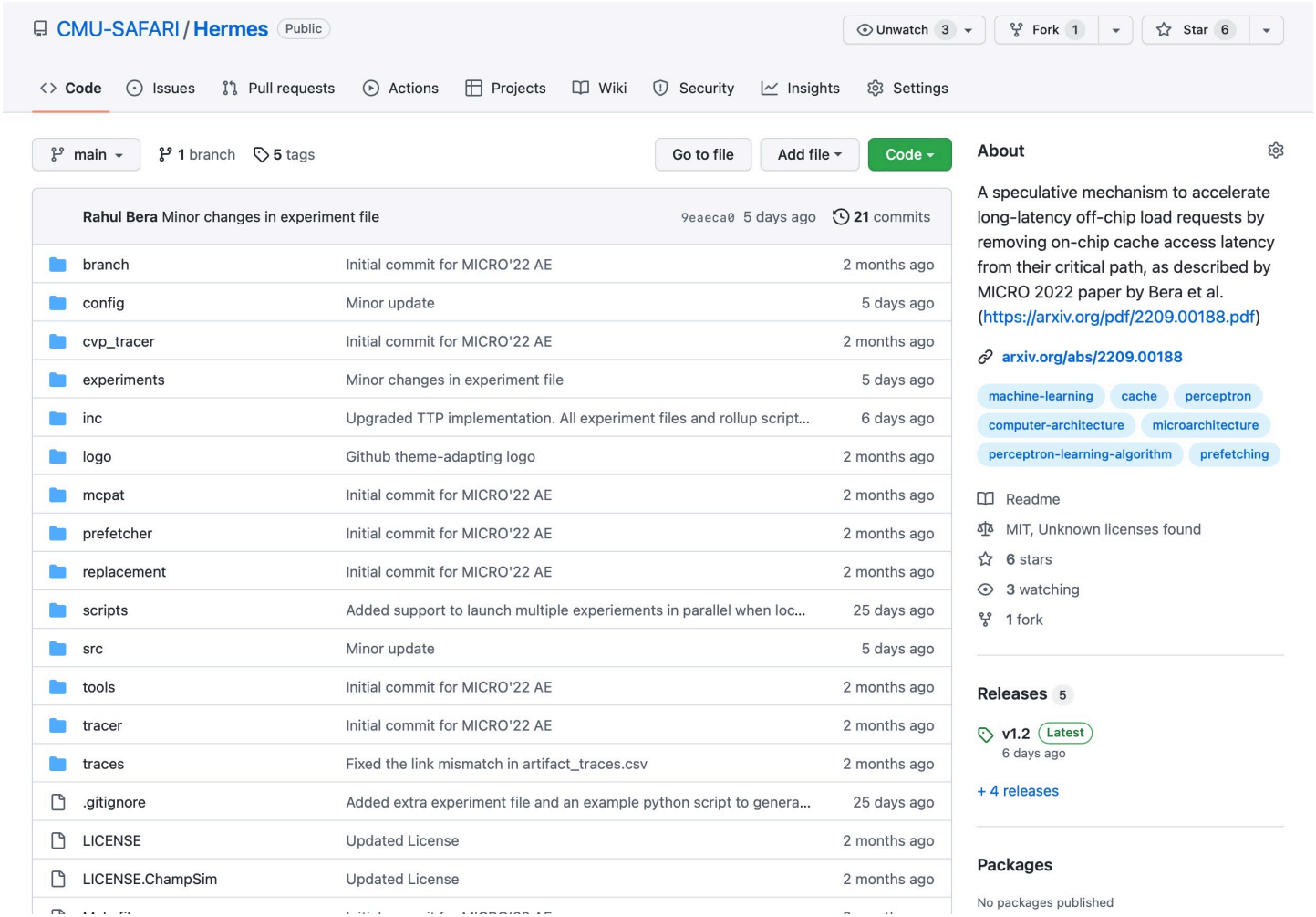
Identifies **74%** off-chip loads,

that misses a **4.5 MB** cache hierarchy using only **4 KB** storage overhead,

with **77%** accuracy,

and provides **5.4%** performance gain

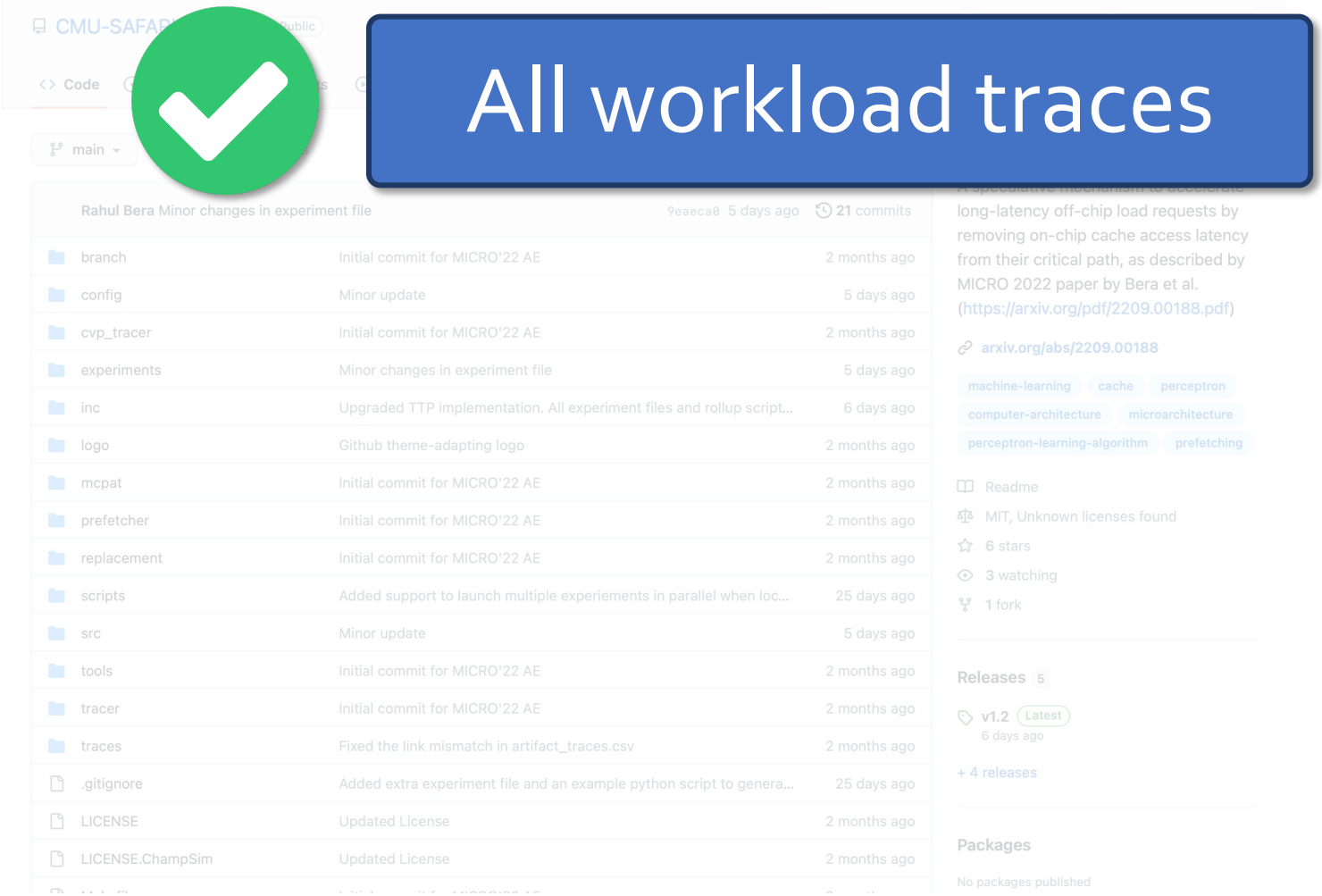
Hermes is Open Sourced



SAFARI

https://github.com/CMU-SAFARI/Hermes

Hermes is Open Sourced



https://github.com/CMU-SAFARI/Hermes

Hermes is Open Sourced

All workload traces

13 prefetchers

- Stride [Fu+, MICRO'92]
- Streamer [Chen and Baer, IEEE TC'95]
- SMS [Somogyi+, ISCA'06]
- AMPM [Ishii+, ICS'09]
- Sandbox [Pugsley+, HPCA'14]
- BOP [Michaud, HPCA'16]
- SPP [Kim+, MICRO'16]
- Bingo [Bakshalipour+, HPCA'19]
- SPP+PPF [Bhatia+, ISCA'19]
- DSPatch [Bera+, MICRO'19]
- MLOP [Shakerinava+, DPC-3'19]
- IPCP [Pakalapati+, ISCA'20]
- Pythia [Bera+, MICRO'21]

https://github.com/CMU-SAFARI/Hermes

Hermes is Open Sourced

All workload traces

13 prefetchers

- Stride [Fu+, MICRO'92]
- Streamer [Chen and Baer, IEEE TC'95]
- SMS [Somogyi+, ISCA'06]
- AMPM [Ishii+, ICS'09]
- Sandbox [Pugsley+, HPCA'14]
- BOP [Michaud, HPCA'16]
- SPP [Kim+, MICRO'16]
- Bingo [Bakshalipour+, HPCA'19]
- SPP+PPF [Bhatia+, ISCA'19]
- DSPatch [Bera+, MICRO'19]
- MLOP [Shakerinava+, DPC-3'19]
- IPCP [Pakalapati+, ISCA'20]
- Pythia [Bera+, MICRO'21]

9 off-chip predictors

Predictor type	Description
Base	Always NO
Basic	Simple confidence counter-based threshold
Random	Random Hit-miss predictor with a given positive probability
HMP-Local	Hit-miss predictor [Yoaz+, ISCA'99] with local prediction
HMP-GShare	Hit-miss predictor with GShare prediction
HMP-GSkew	Hit-miss predictor with GSkew prediction
HMP-Ensemble	Hit-miss predictor with all three types combined
TTP	Tag-tracking based predictor
Perc	Perceptron-based OCP used in this paper

https://github.com/CMU-SAFARI/Hermes

Easy To Define Your Own Off-Chip Predictor

- Just extend the OffchipPredBase class

```
8   class OffchipPredBase
9   {
10  public:
11      uint32_t cpu;
12      string type;
13      uint64_t seed;
14      uint8_t dram_bw; // current DRAM bandwidth bucket
15
16      OffchipPredBase(uint32_t _cpu, string _type, uint64_t _seed) : cpu(_cpu), type(_type), seed(_seed)
17      {
18          srand(seed);
19          dram_bw = 0;
20      }
21      ~OffchipPredBase() {}
22      void update_dram_bw(uint8_t _dram_bw) { dram_bw = _dram_bw; }
23
24      virtual void print_config();
25      virtual void dump_stats();
26      virtual void reset_stats();
27      virtual void train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
28      virtual bool predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry);
29  };
30
31  #endif /* OFFCHIP_PRED_BASE_H */
32
```

Easy To Define Your Own Off-Chip Predictor

- Define your own `train()` and `predict()` functions

```
19  void OffchipPredBase::train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
20  {
21      // nothing to train
22  }
23
24  bool OffchipPredBase::predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
25  {
26      // predict randomly
27      // return (rand() % 2) ? true : false;
28      return false;
29  }
```

Easy To Define Your Own Off-Chip Predictor

- Define your own `train()` and `predict()` functions

```
19  void OffchipPredBase::train(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
20  {
21      // nothing to train
22  }
23
24  bool OffchipPredBase::predict(ooo_model_instr *arch_instr, uint32_t data_index, LSQ_ENTRY *lq_entry)
25  {
26      // predict randomly
27      // return (rand() % 2) ? true : false;
28      return false;
29  }
```

- Get statistics like precision (aka accuracy) and recall (aka coverage) out of the box

```
Core_0_offchip_pred_true_pos 2358716
Core_0_offchip_pred_false_pos 276883
Core_0_offchip_pred_false_neg 132145
Core_0_offchip_pred_precision 89.49
Core_0_offchip_pred_recall 94.69
```

Off-Chip Prediction Can Further Enable…

Off-Chip Prediction Can Further Enable…

Prioritizing loads that are likely go off-chip in **cache queues** and **on-chip network routing**

Off-Chip Prediction Can Further Enable…

Prioritizing loads that are likely go off-chip in **cache queues** and **on-chip network routing**

Better instruction scheduling of data-dependent instructions

Off-Chip Prediction Can Further Enable…

Prioritizing loads that are likely go off-chip in **cache queues** and **on-chip network routing**

Better instruction scheduling of data-dependent instructions

and many more…

HERMES

Accelerating Long-Latency Load Requests via Perceptron-Based Off-Chip Load Prediction

Rahul Bera, Konstantinos Kanellopoulos, Shankar Balachandran, David Novo, Ataberk Olgun, Mohammad Sadrosadati, Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

LIRMM