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

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

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 **stall** processor by
blocking instruction retirement from
Reorder Buffer (ROB)

SAFARI

The Key Problem

Long-latency off-chip load requests

Often **stall** processor by
blocking instruction retirement from
Reorder Buffer (ROB)

Limit performance

SAFARI

Traditional Solutions

Traditional Solutions

1

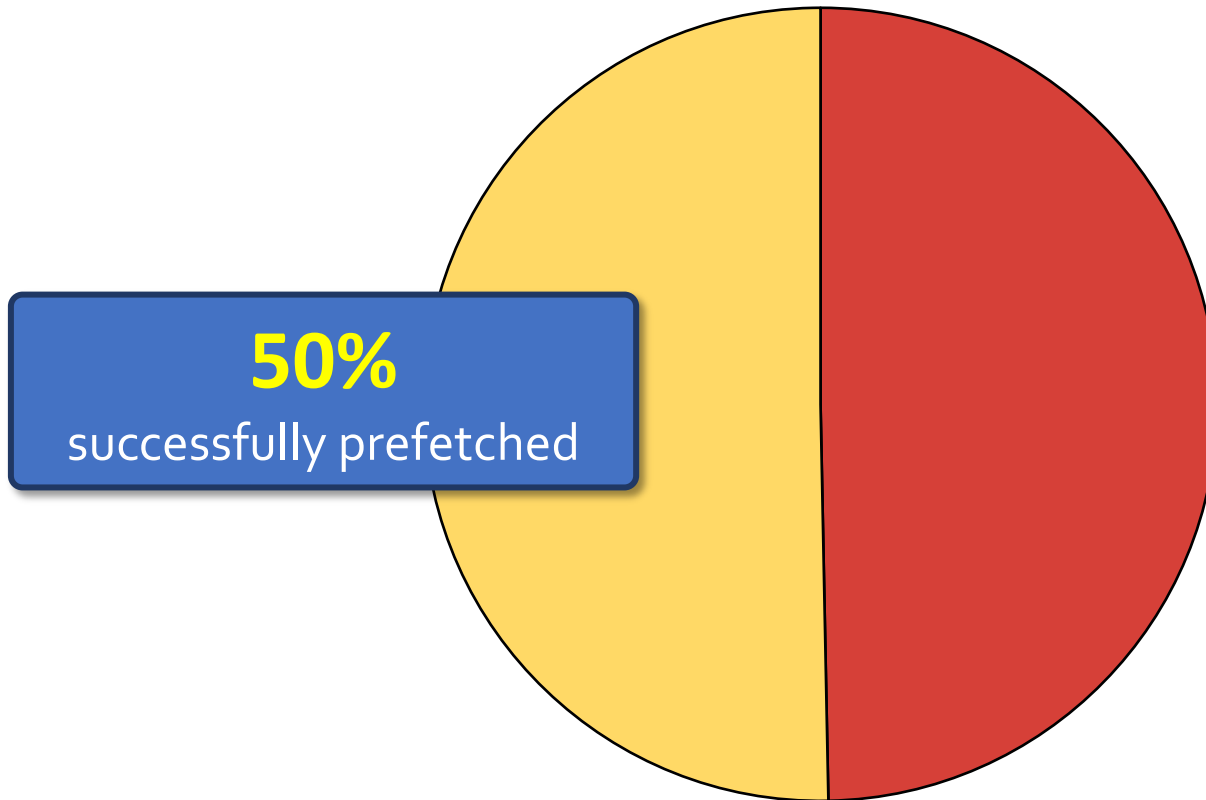Employ sophisticated **prefetchers**

Traditional Solutions

1 Employ sophisticated **prefetchers**

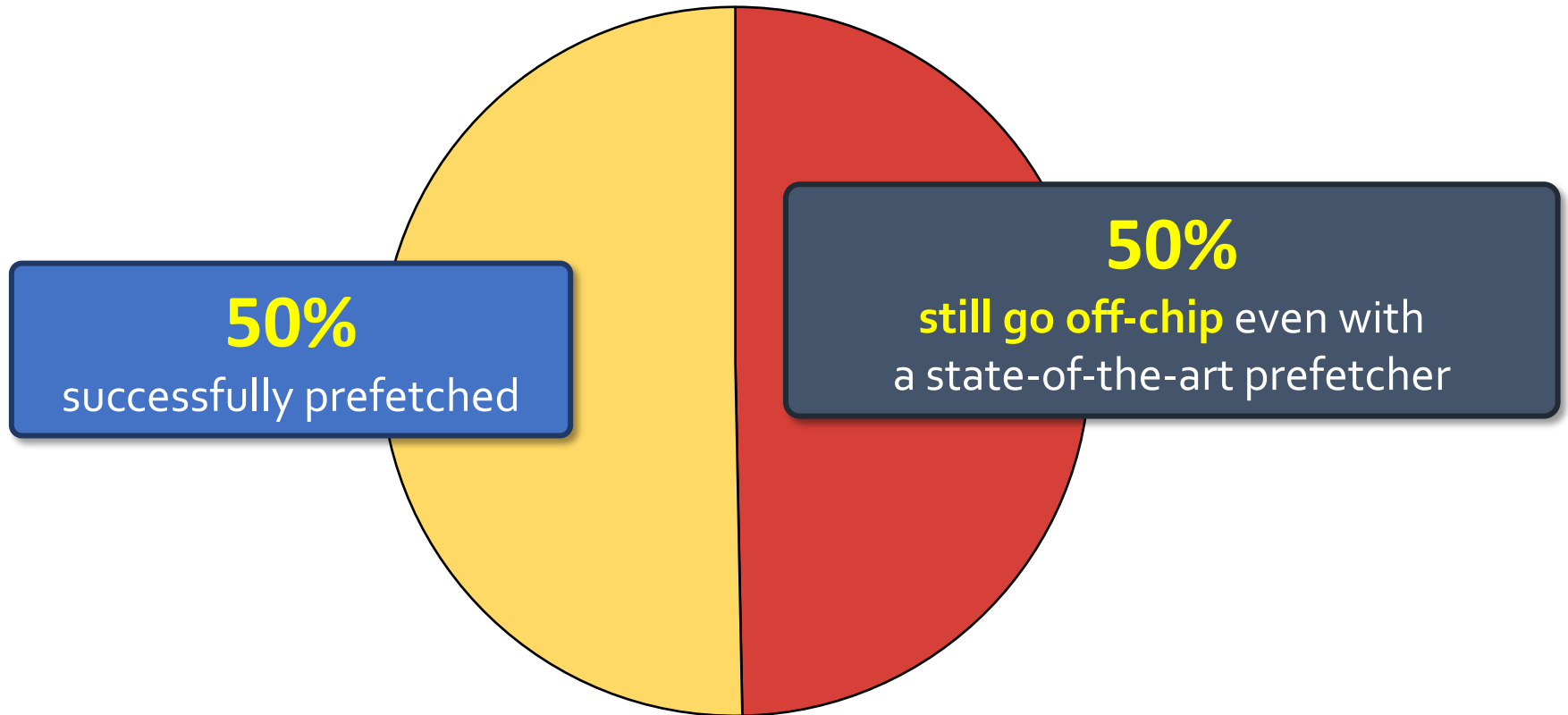**2** **Increase size** of on-chip caches

Key Observation 1

Many loads still go off-chip

50%
successfully prefetched

 # off-chip loads without any prefetcher

Key Observation 1
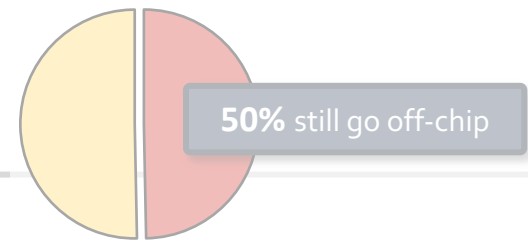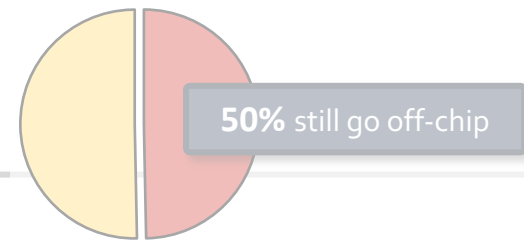
Many loads still go off-chip

50%
successfully prefetched

50%
still go off-chip even with
a state-of-the-art prefetcher

 # off-chip loads without any prefetcher

SAFARI

Key Observation 1

Many loads still go off-chip

50%
successfully prefetched

50%
still go off-chip even with
a state-of-the-art prefetcher

70% of the off-chip loads
block the ROB

 # off-chip loads without any prefetcher

SAFARI

Key Observation 2

50% still go off-chip

SAFARI

Key Observation 2

> **On-chip cache access latency**
> significantly contributes to off-chip load latency

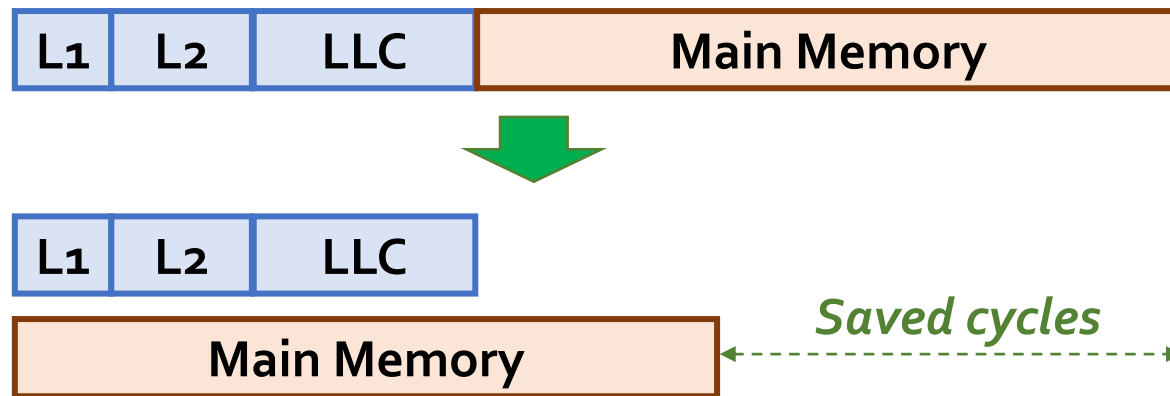| L1	L2	LLC	Main Memory

SAFARI

Key Observation 2

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

| L1 | L2 | LLC | Main Memory |

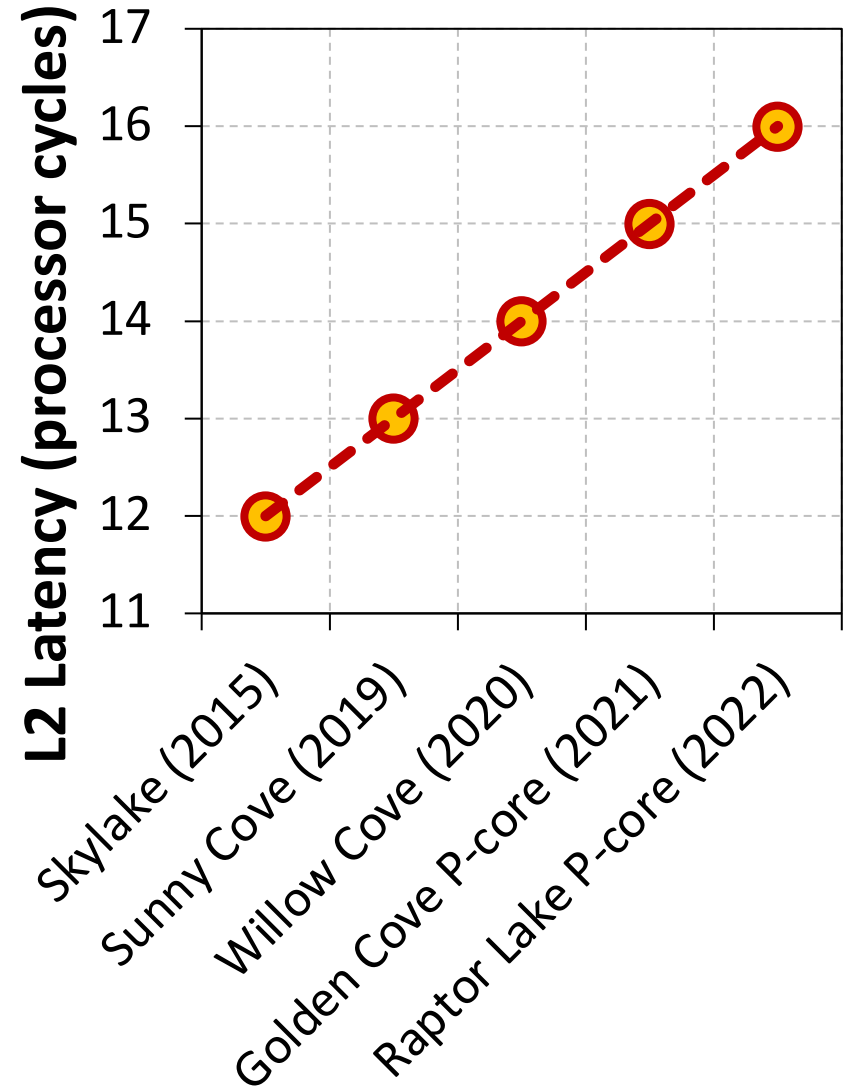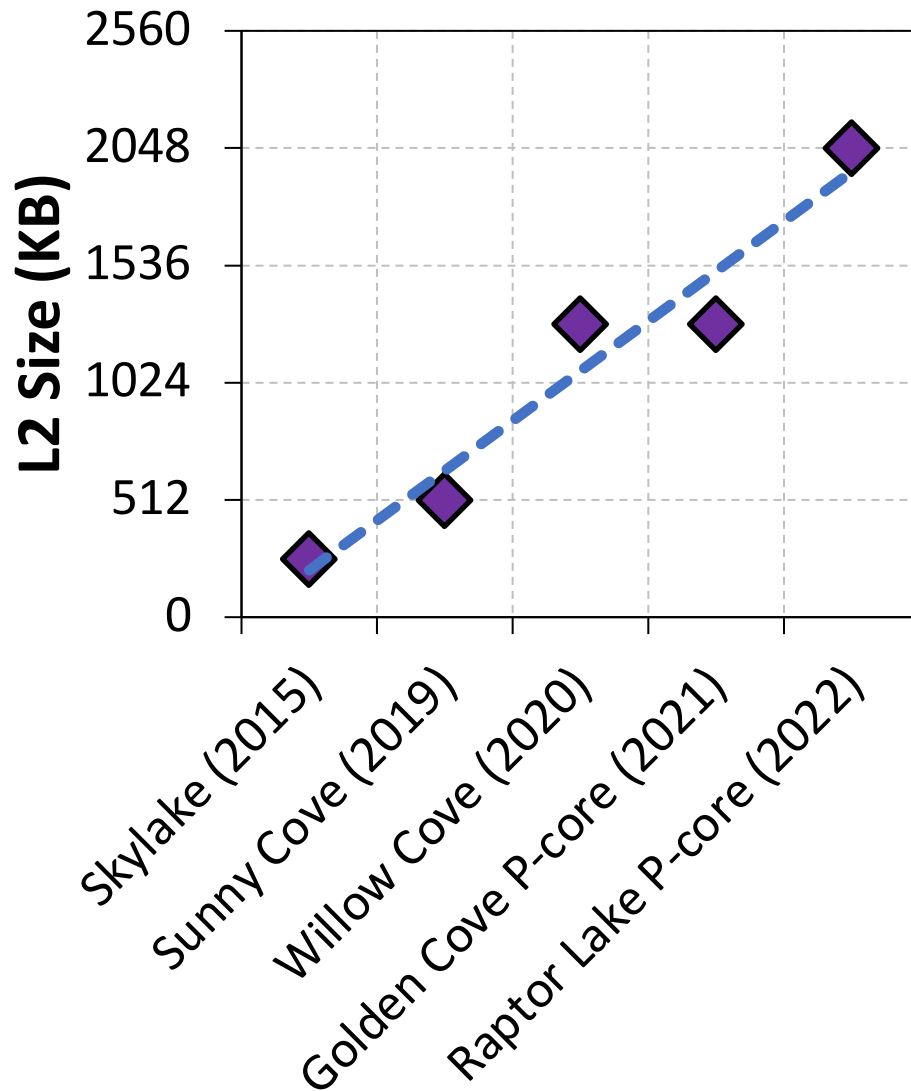| L1 | L2 | LLC |

| Main Memory | *Saved cycles*
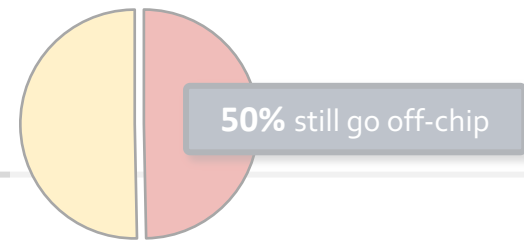
40% of the **stalls** can be eliminated by **removing on-chip cache access latency from critical path**

SAFARI

13

Caches are Getting Bigger and Slower…
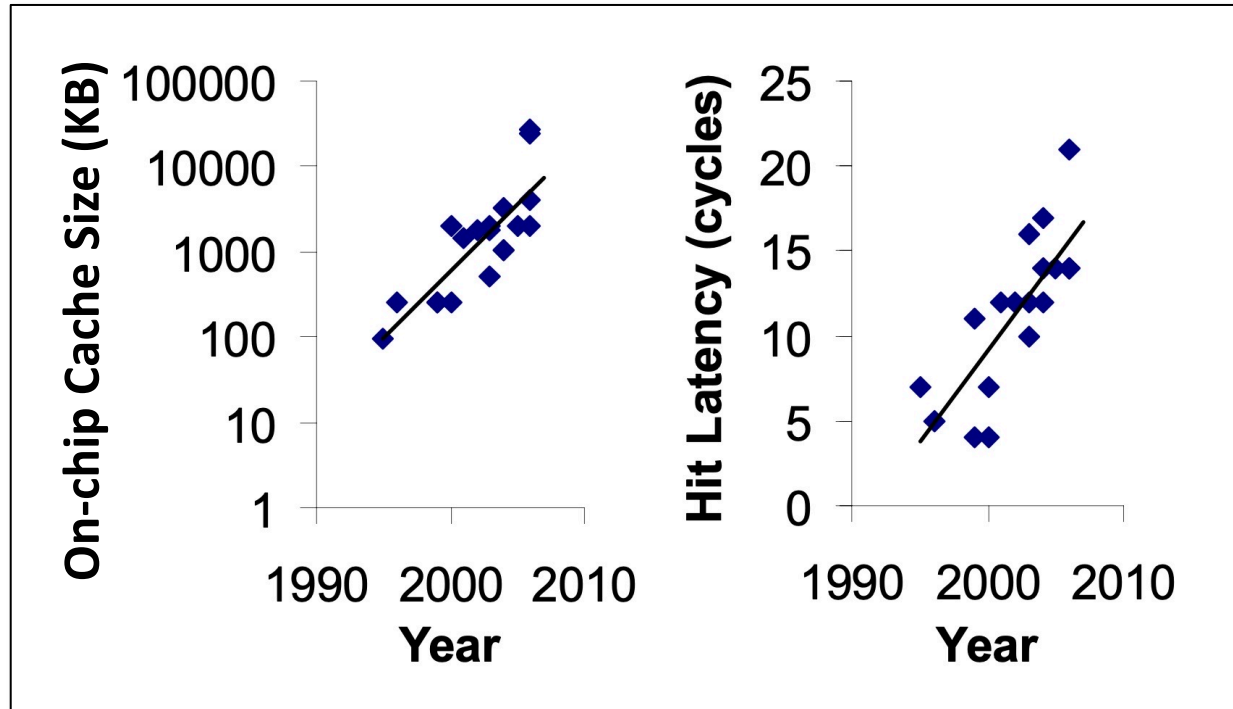
Hardavellas+, "Database Servers on Chip Multiprocessors: Limitations and Opportunities", CIDR, 2007

SAFARI

Caches are Getting Bigger and Slower…

Our Goal

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

SAFARI

Predicts which load requests
are likely to **go off-chip**

SAFARI

Predicts which load requests
are likely to **go off-chip**

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

Key Contribution

> **1** Hermes employs **the first perceptron-based** off-chip load predictor

Key Contribution

Hermes employs **the first perceptron-based** off-chip load predictor

That predicts which loads are likely to **go off-chip**

SAFARI

Key Contribution

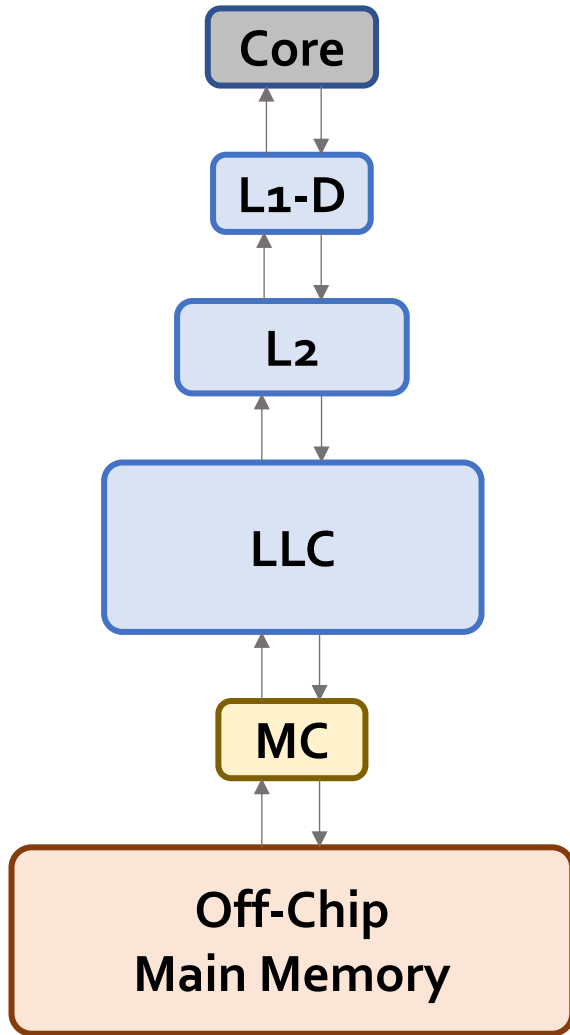Hermes employs **the first perceptron-based** off-chip load predictor

That predicts which loads are likely to **go off-chip**

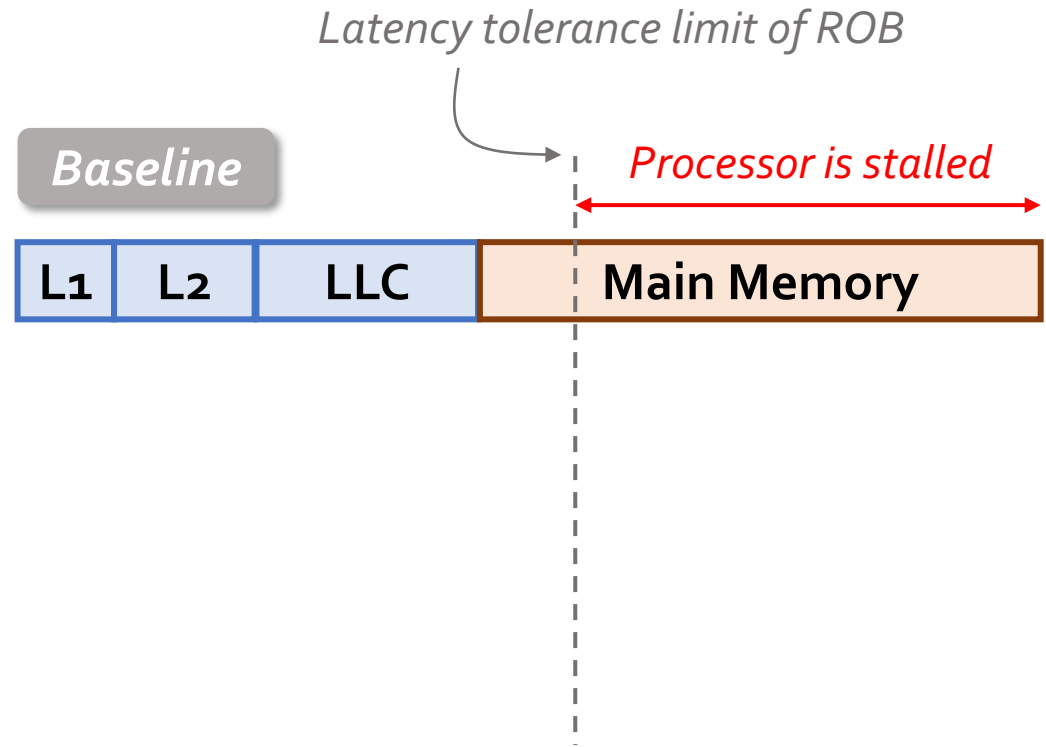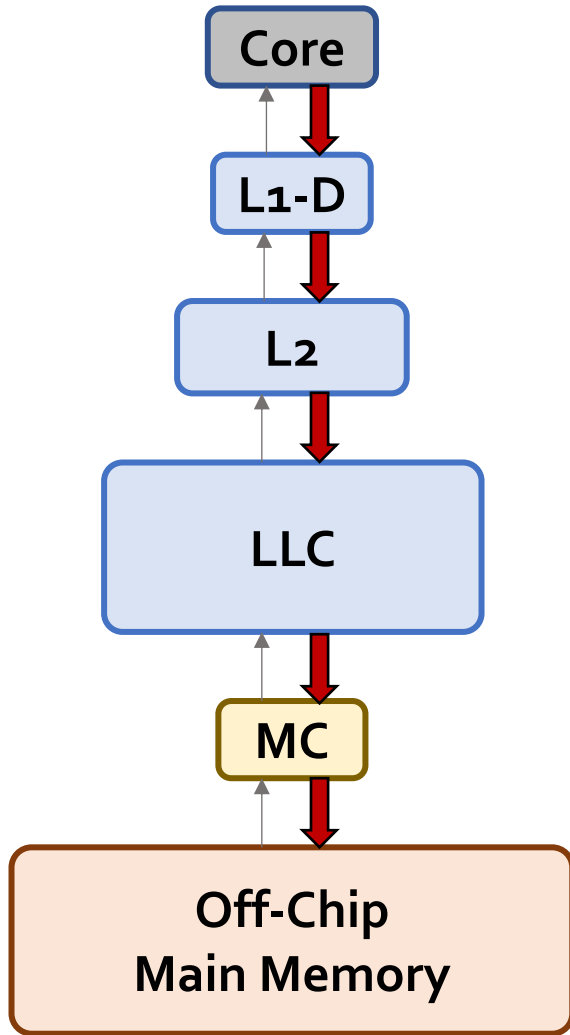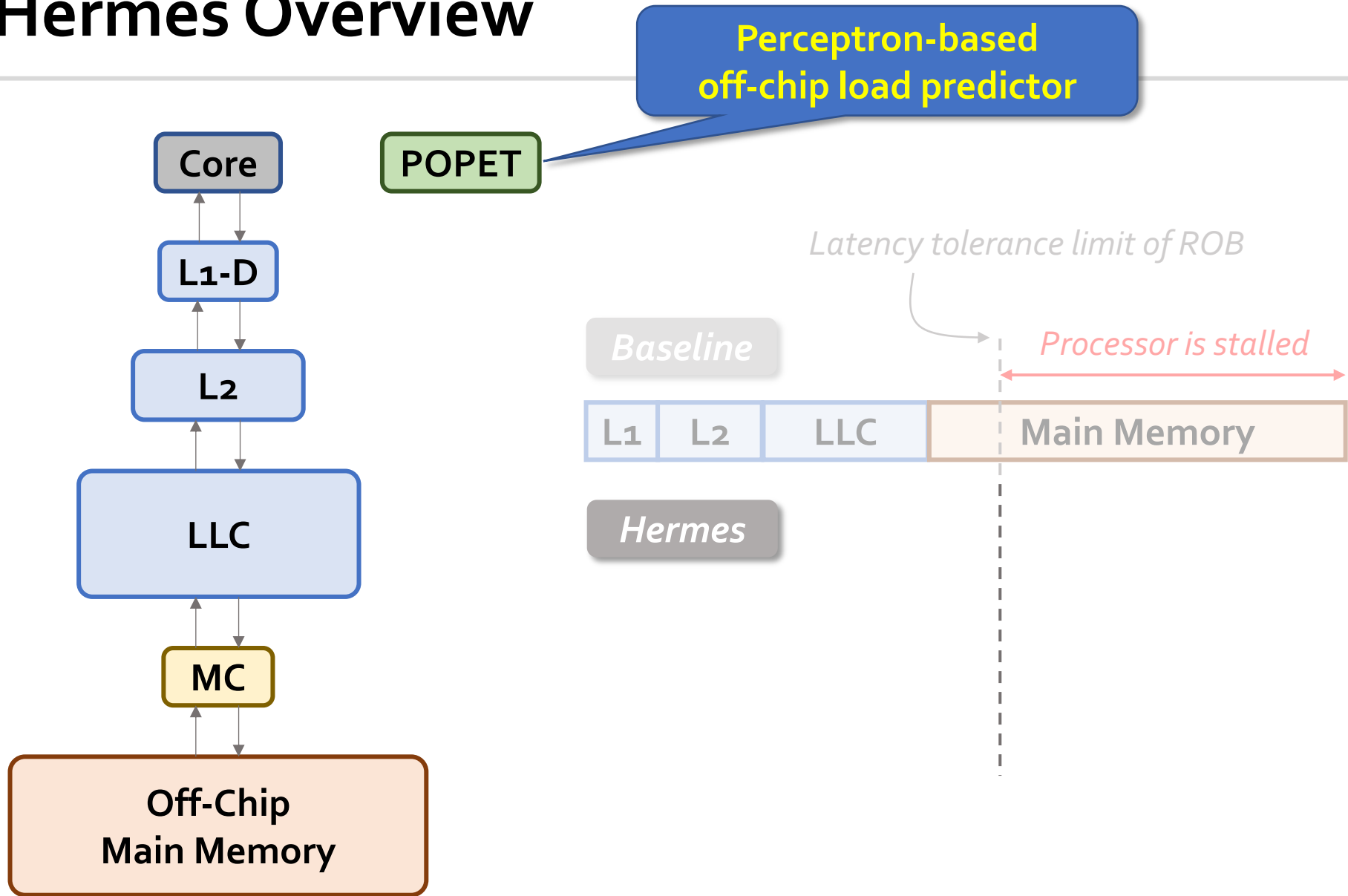By **learning** from **multiple program context information**
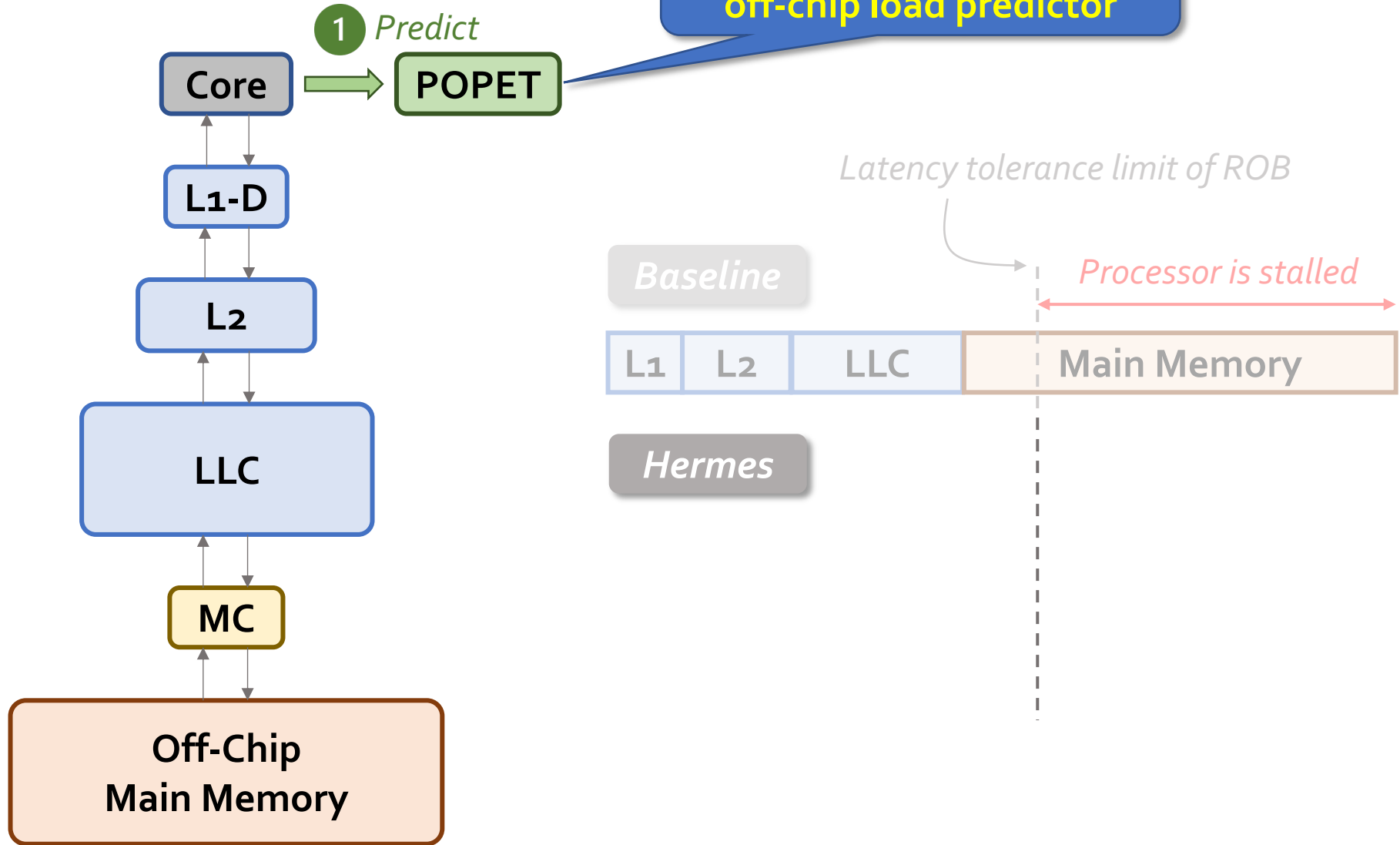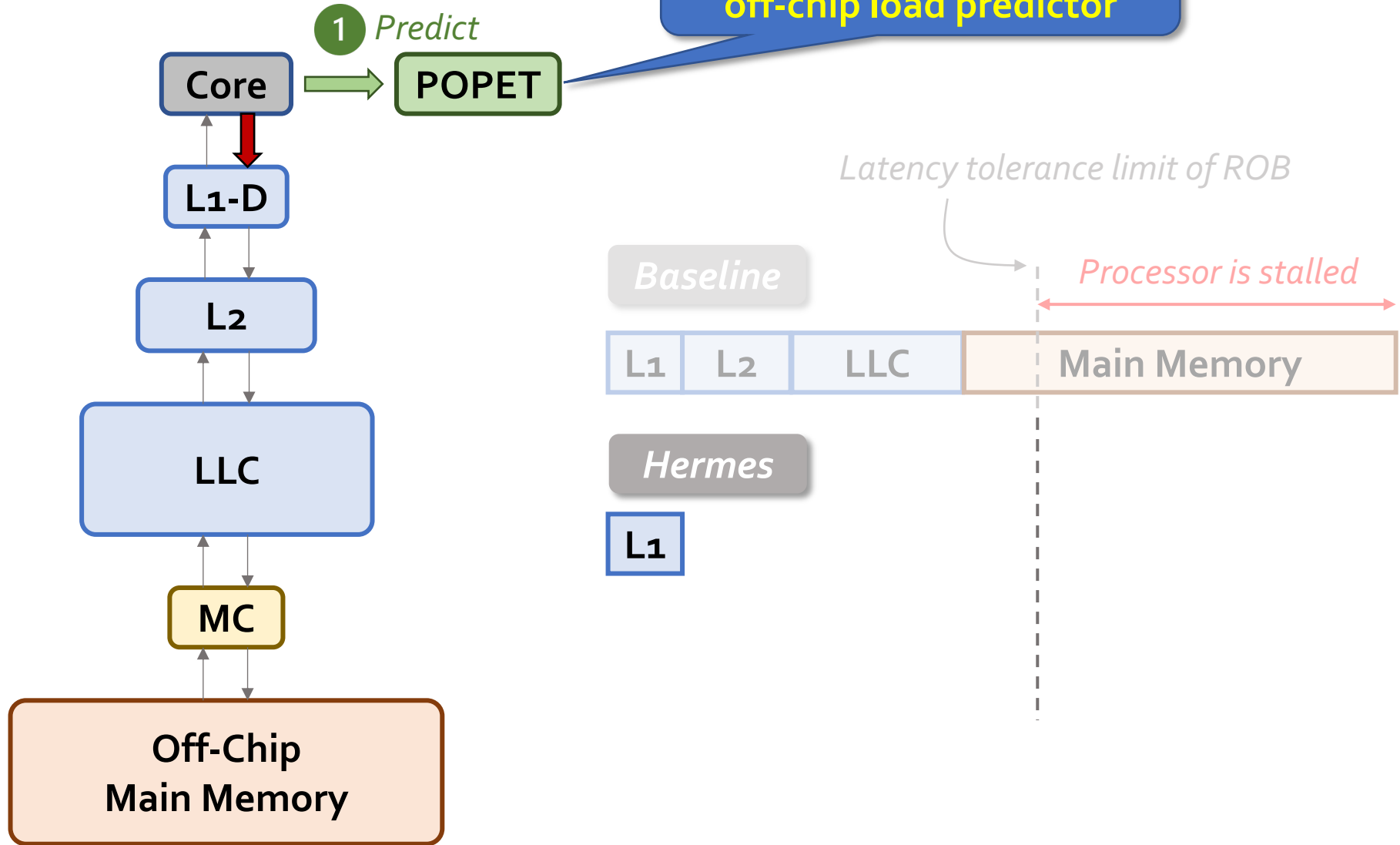
Hermes Overview

SAFARI

Hermes Overview

Hermes Overview

Perceptron-based off-chip load predictor

Core

L1-D

L2

LLC

MC

Off-Chip Main Memory

POPET

Latency tolerance limit of ROB

Baseline

Processor is stalled

| L1 | L2 | LLC | Main Memory |

Hermes

SAFARI

Hermes Overview

Perceptron-based
off-chip load predictor

① *Predict*

Core → **POPET**

L1-D

L2

LLC

MC

**Off-Chip
Main Memory**

Latency tolerance limit of ROB

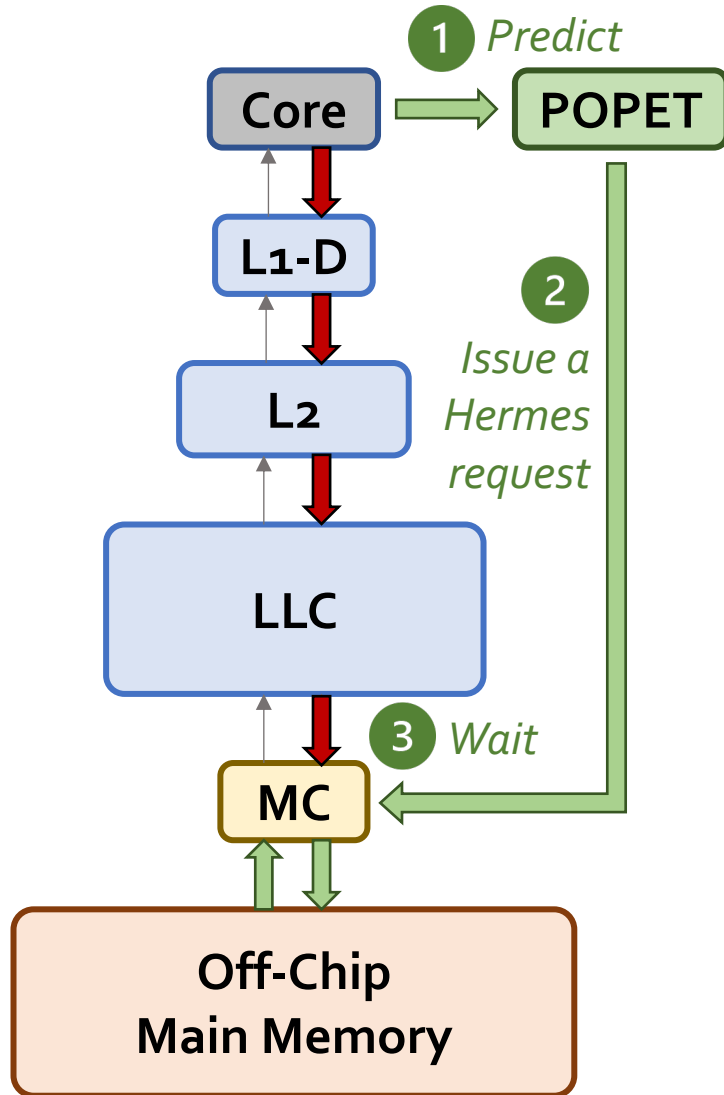**Baseline**

Processor is stalled
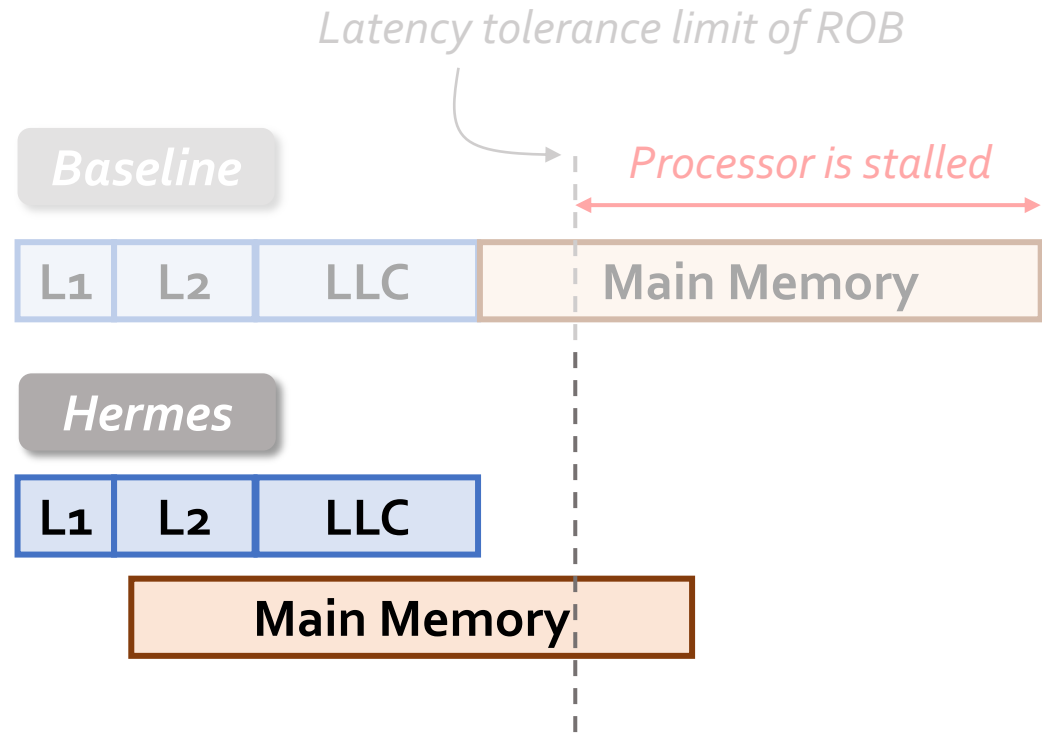
| L1 | L2 | LLC | Main Memory |

Hermes

Hermes Overview

Hermes Overview

Perceptron-based off-chip load predictor

Core → **POPET**
1 *Predict*

L1-D

2 *Issue a Hermes request*

L2

LLC

3 *Wait*

MC

Off-Chip Main Memory

Latency tolerance limit of ROB

Processor is stalled

Baseline

| L1 | L2 | LLC | Main Memory |

Hermes

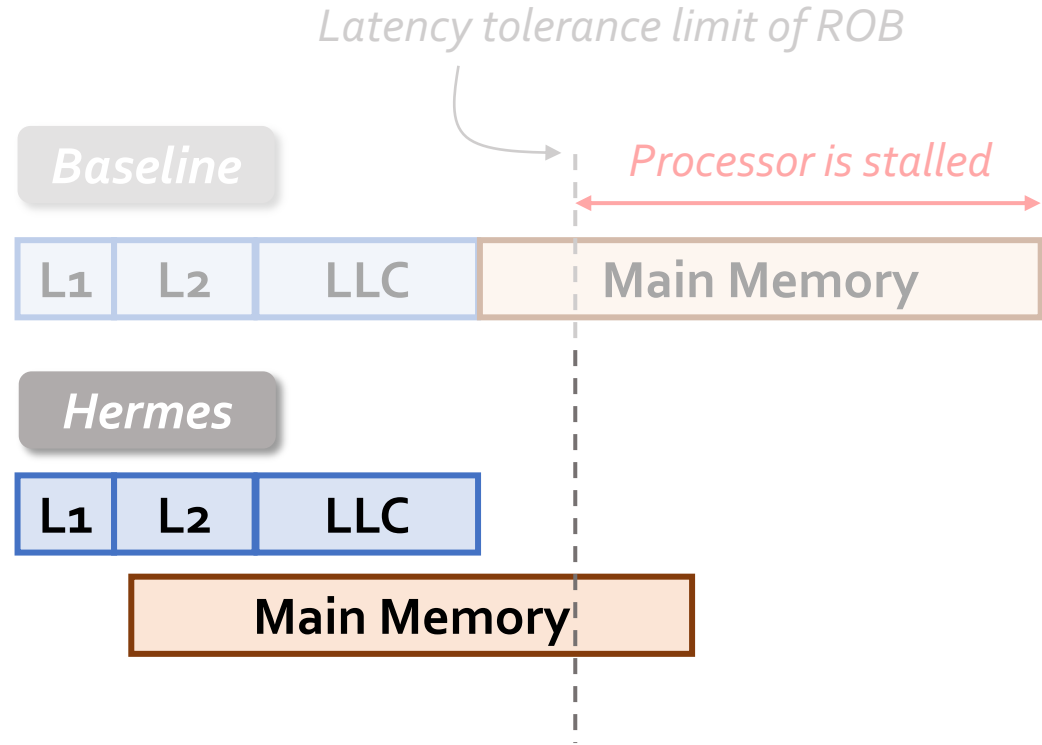| L1 | L2 | LLC |

Main Memory

SAFARI

Hermes Overview

SAFARI

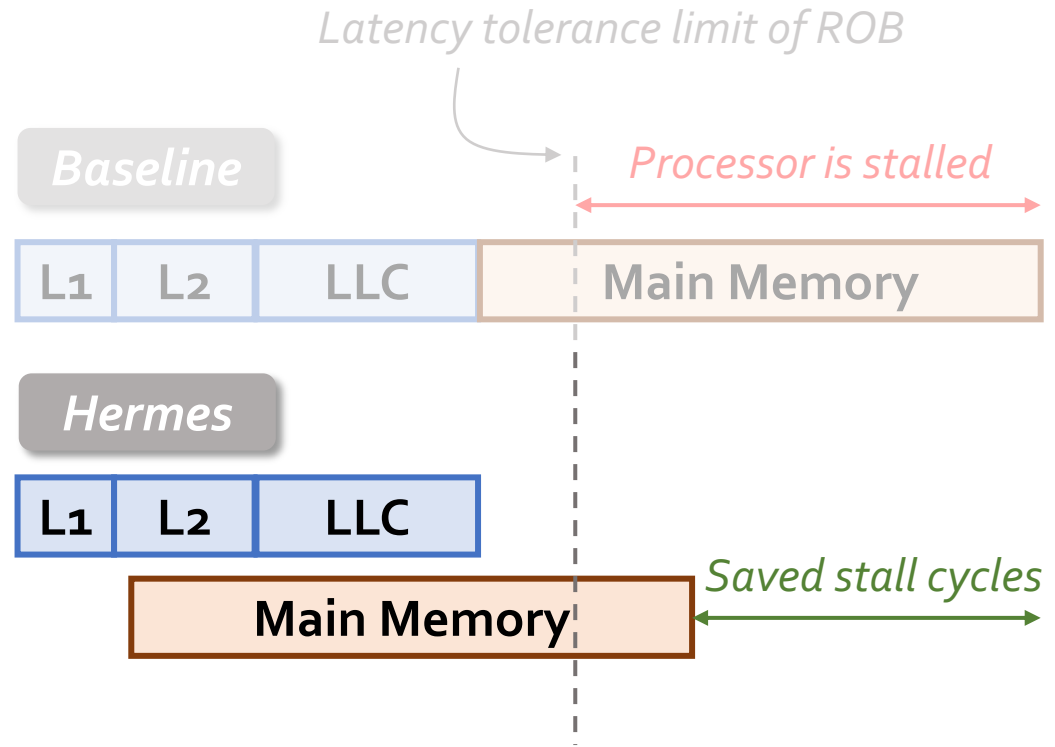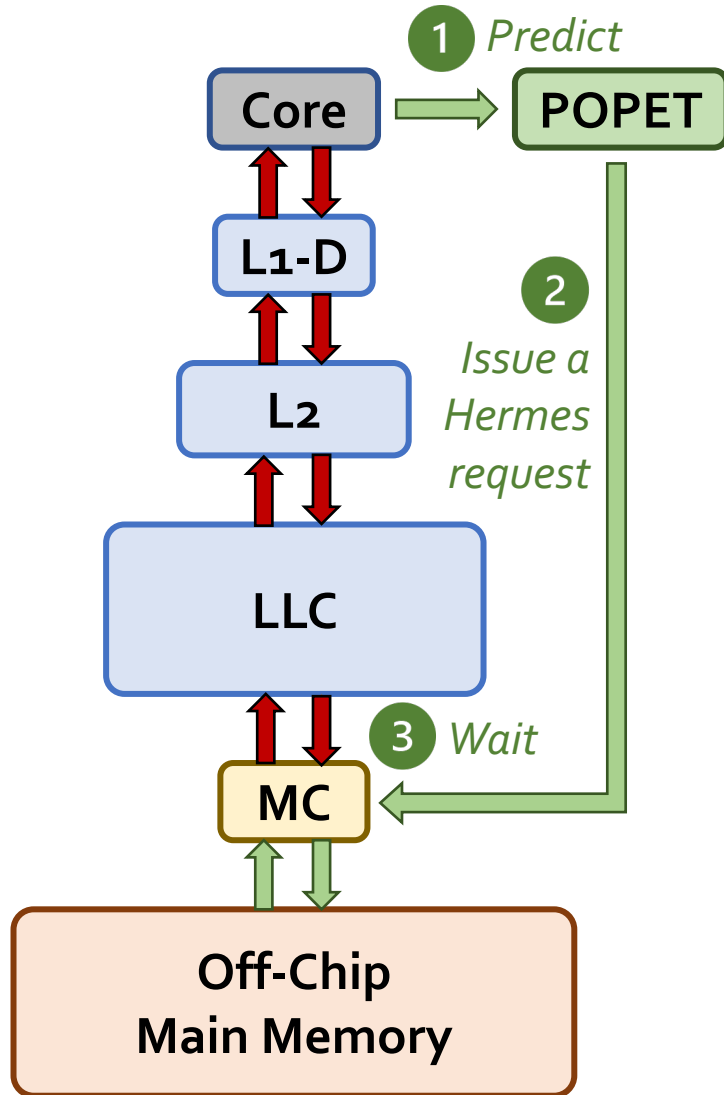# Hermes Overview

Perceptron-based
off-chip load predictor

1 *Predict*

Core → POPET

L1-D

2 *Issue a Hermes request*

L2

LLC

3 *Wait*

MC

Off-Chip
Main Memory

Latency tolerance limit of ROB

Baseline

Processor is stalled

| L1 | L2 | LLC | Main Memory |

Hermes

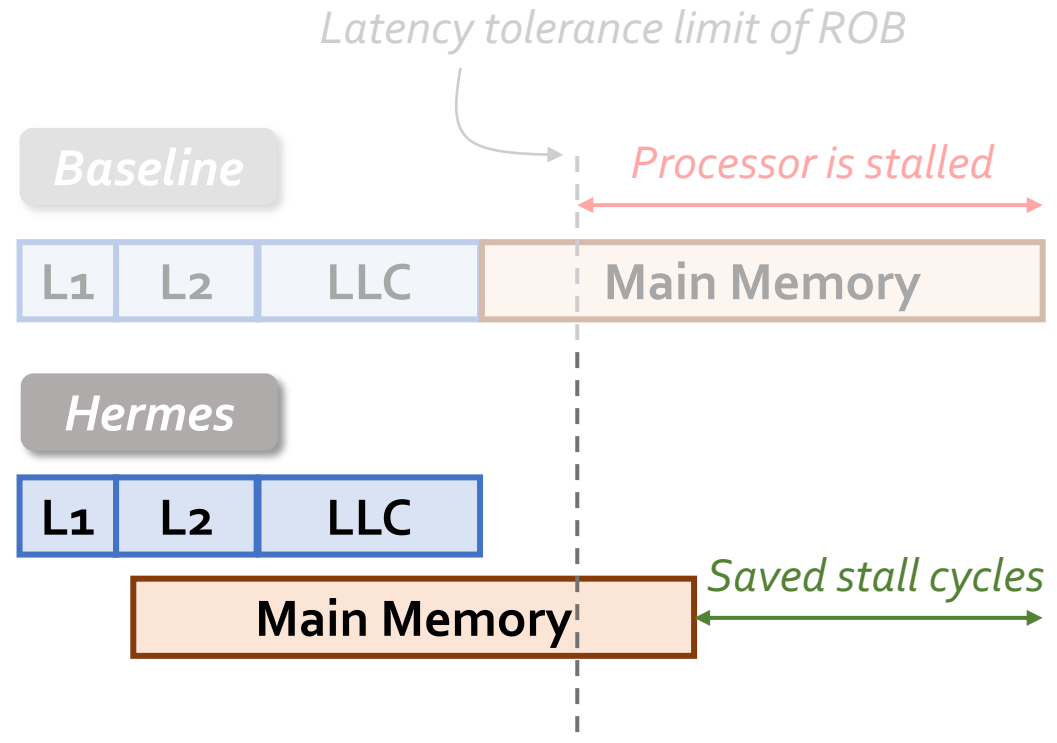| L1 | L2 | LLC |

Main Memory

Saved stall cycles

SAFARI

Hermes Overview
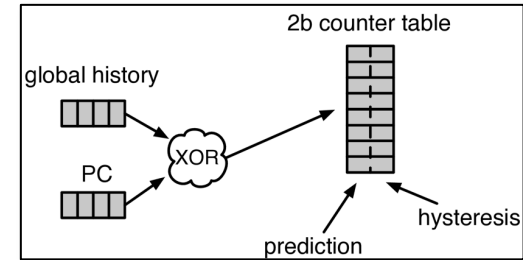

Perceptron-based off-chip load predictor

SAFARI

Designing the Off-Chip Load Predictor

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Tracking cache contents

MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

Tracking cache contents

MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

Designing the Off-Chip Load Predictor
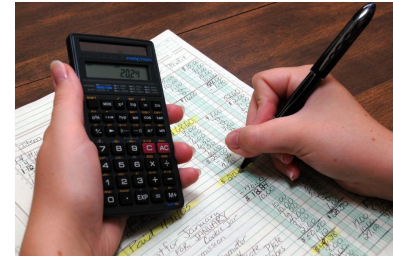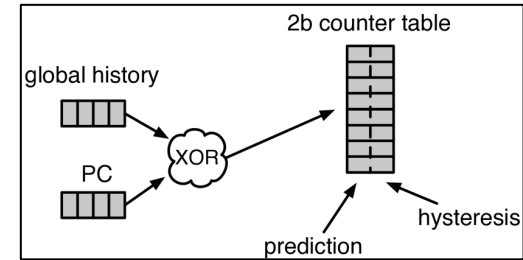
History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

Tracking cache contents

MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

❌ **Large** metadata
  ▪ Metadata size increases with cache hierarchy size

❌ May need to track **all** cache operations
  ▪ Gets complex depending on the cache hierarchy
    configuration (e.g., inclusivity, bypassing,…)

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

Tracking cache contents

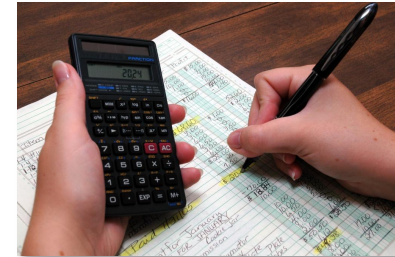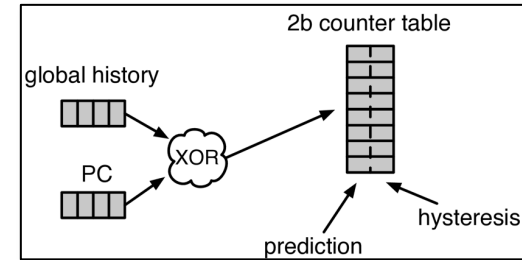MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

❌ **Large** metadata
- Metadata size increases with cache hierarchy size

❌ May need to track **all** cache operations
- Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,…)

Learning from program behavior

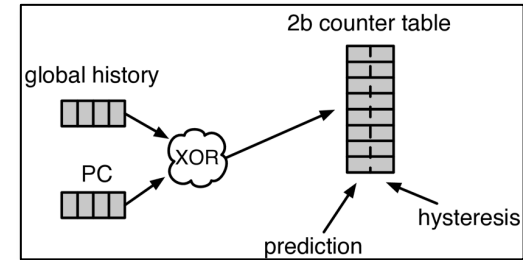Correlate different program features with off-chip loads

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

Tracking cache contents

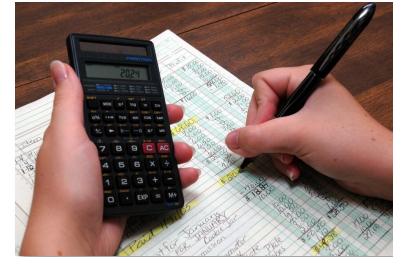MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

❌ **Large** metadata
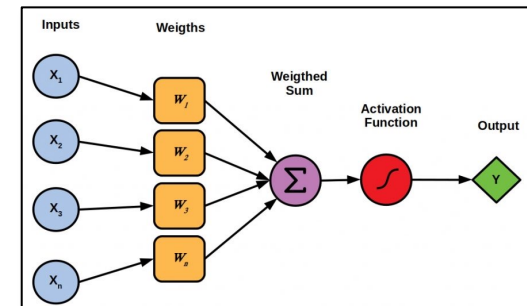- Metadata size increases with cache hierarchy size

❌ May need to track **all** cache operations
- Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,…)

Learning from program behavior

Correlate different program features with off-chip loads
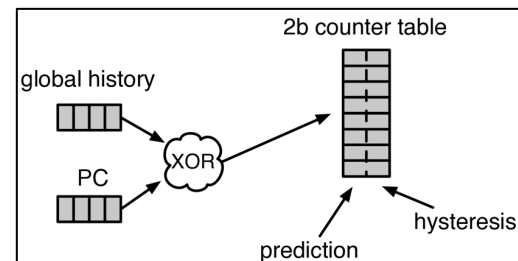
✅ Low storage overhead ✅ Low design complexity

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

Tracking cache contents

MissMap [Loh+, MICRO'11] for the **DRAM cache**,
D2D [Sembrant+, ISCA'14], D2M [Sembrant+, HPCA'17], LP [Jalili+, HPCA'22] for the **cache hierarchy**

❌ **Large** metadata
  ▪ Metadata size increases with cache hierarchy size

❌ May need to track **all** cache operations
  ▪ Gets complex depending on the cache hierarchy configuration (e.g., inclusivity, bypassing,…)
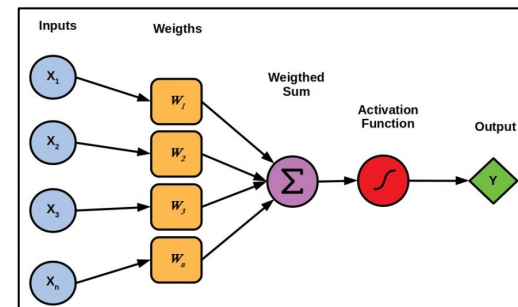
Learning from program behavior

Correlate different program features with off-chip loads

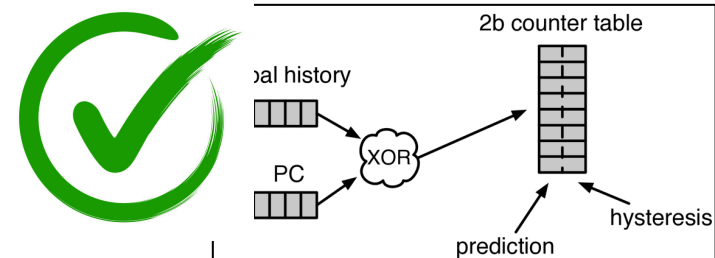✅ Low storage overhead   ✅ Low design complexity

Designing the Off-Chip Load Predictor

History-based prediction

HMP [Yoaz+, ISCA'99] for the **L1-D cache**

Using **branch-predictor-like** hybrid predictor:
Global, Gshare, and GSkew

2b counter table

global history

PC

XOR

hysteresis

POPET provides
both **higher accuracy** and **higher performance**
than predictors inspired from these previous works

Metadata size increases with cache hierarchy size

❌ May need to track **all** cache operations

Gets complex depending on the cache hierarchy
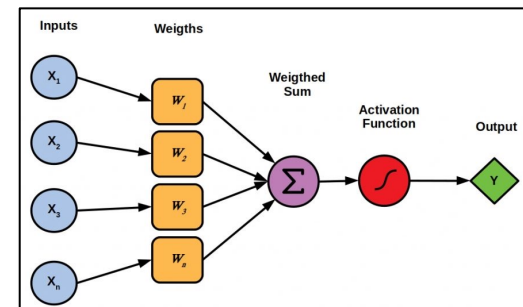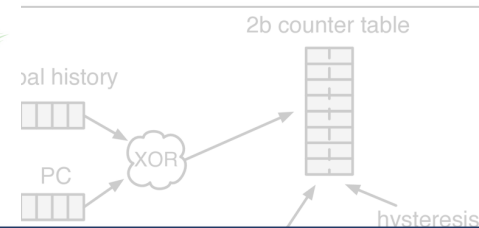configuration (e.g., inclusivity, bypassing,…)

Learning from program behavior

Correlate different program features with off-chip loads

✅ Low storage overhead   ✅ Low design complexity

Inputs   Weigths

x_1

x_2

x_3

x_n

w_1

w_2

w_3

w_4

Weigthed
Sum

Activation
Function

Output

Σ

∫

Y

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
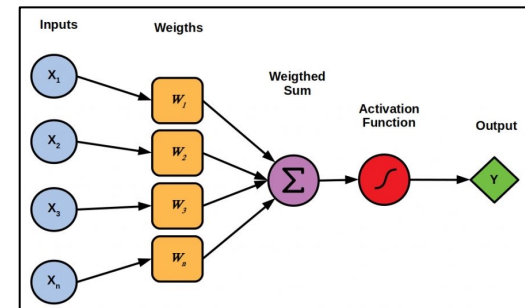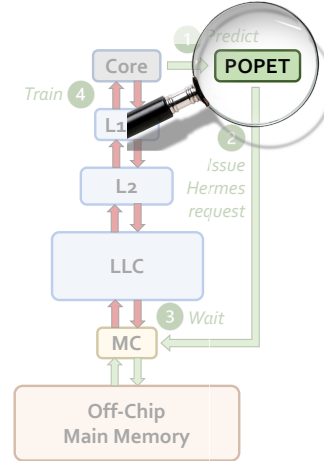
POPET: Perceptron-Based Off-Chip Predictor

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

Feature$_1$

(e.g., PC + offset)

Weight Table$_1$

Feature$_2$

Weight Table$_2$

Feature$_N$

Weight Table$_N$

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

SAFARI
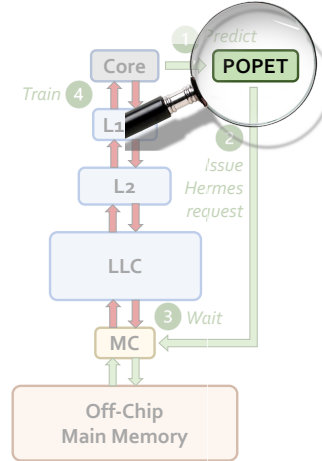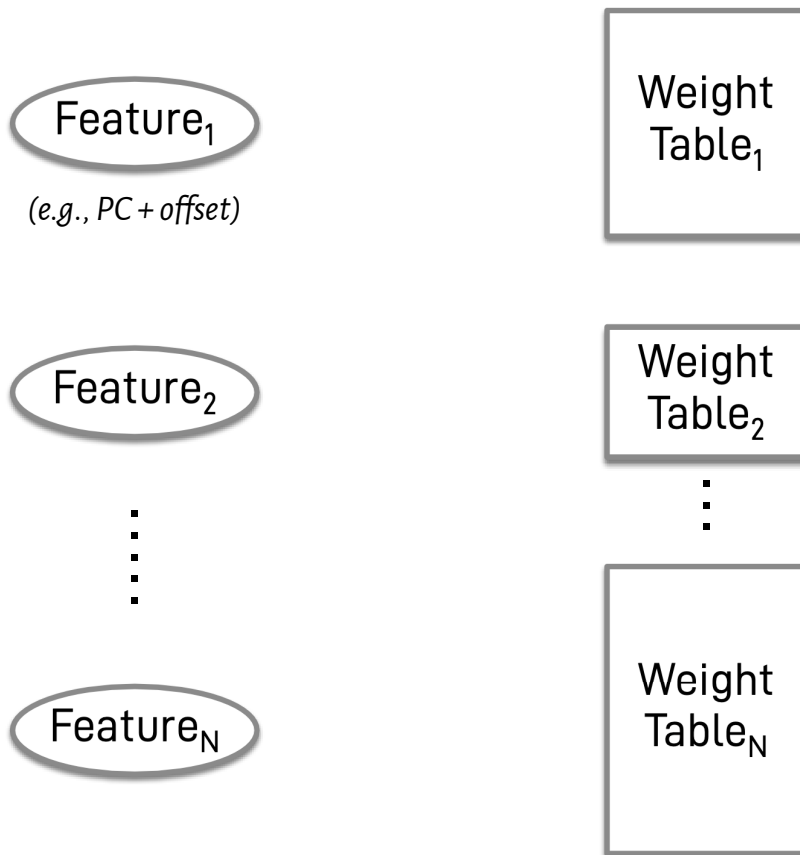
Predicting using POPET

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

Feature$_1$

(e.g., PC + offset)

Feature$_2$

Feature$_N$

Weight Table$_1$

Weight Table$_2$

Weight Table$_N$

SAFARI

Predicting using POPET

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

Stage 1

Extract features from the load request

$Feature_1$

(e.g., PC + offset)

0x7ffe0+12

$Feature_2$

$Feature_N$

$Weight Table_1$

$Weight Table_2$

$Weight Table_N$

Predicting using POPET

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

SAFARI

Predicting using POPET

- Uses simple **table lookups**, **addition**, and **comparison**
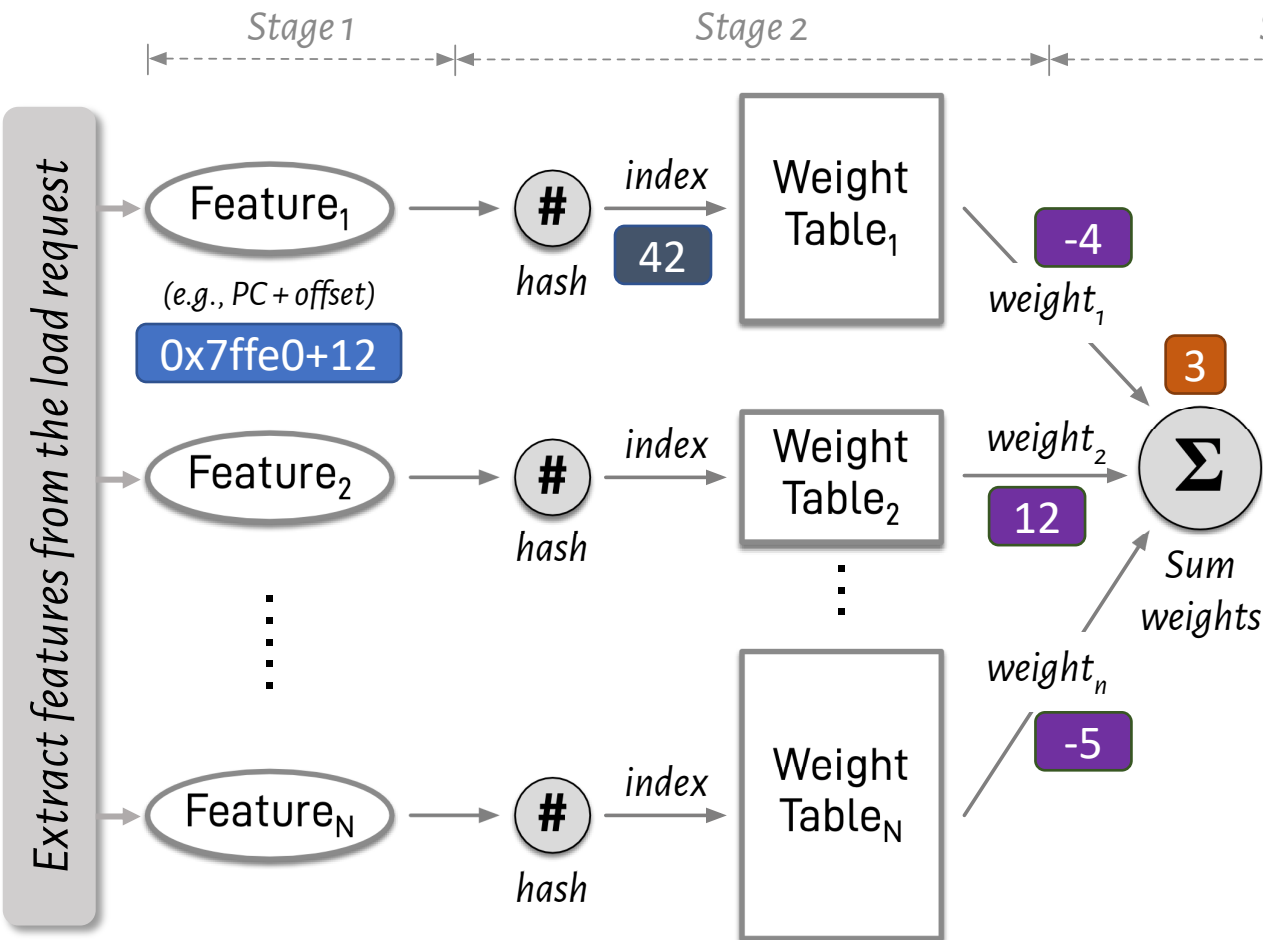
Predicting using POPET

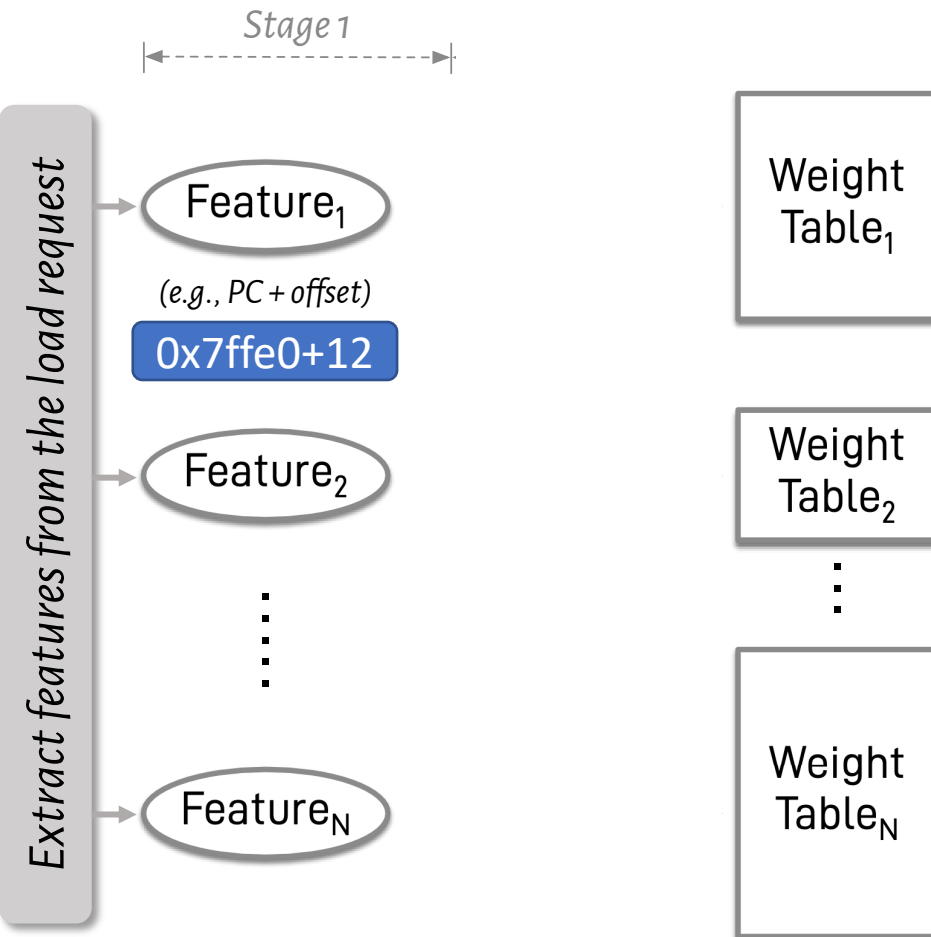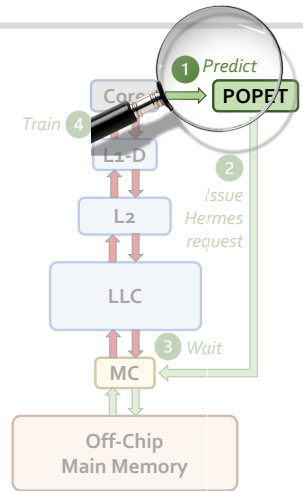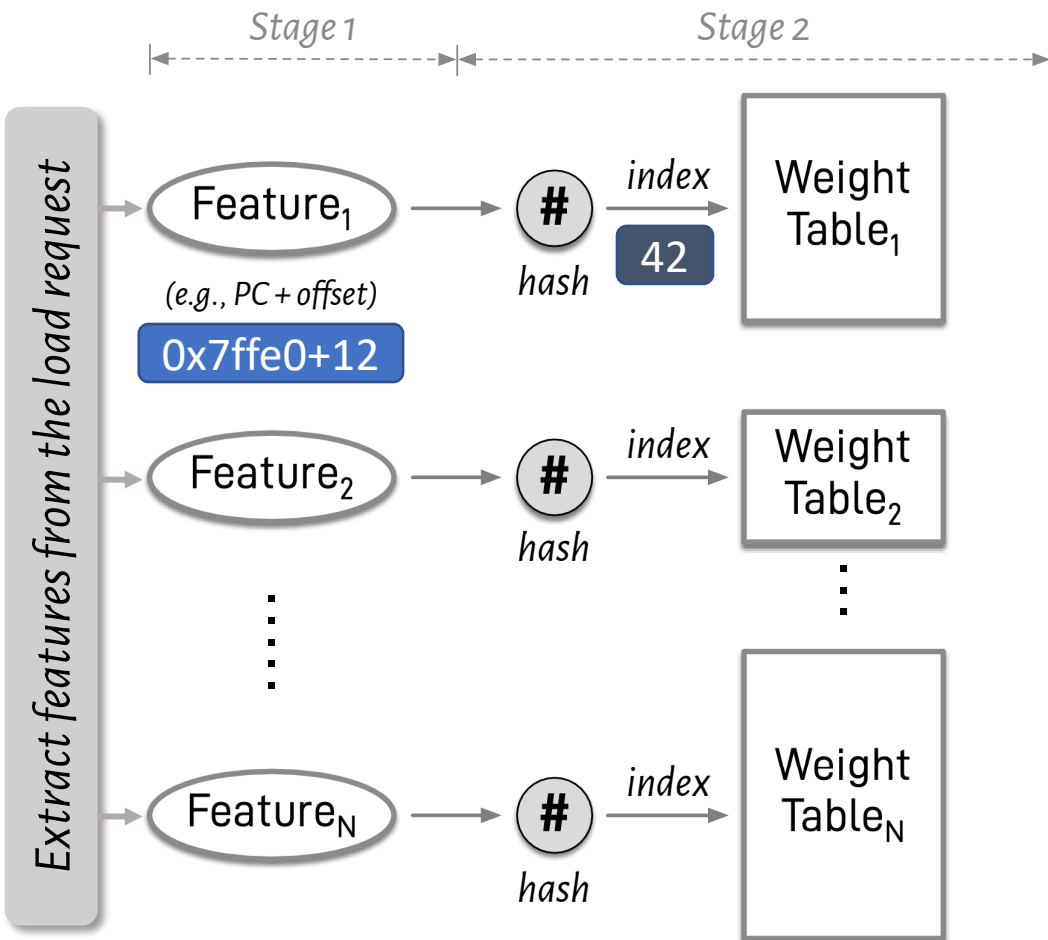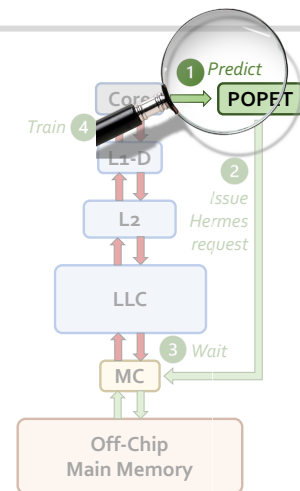- Uses simple **table lookups**, **addition**, and **comparison**
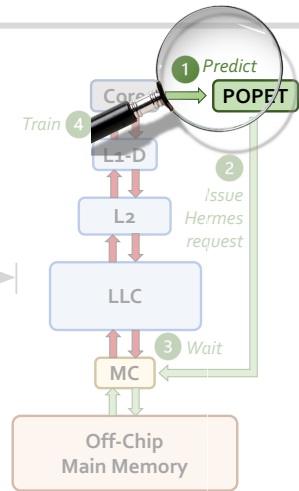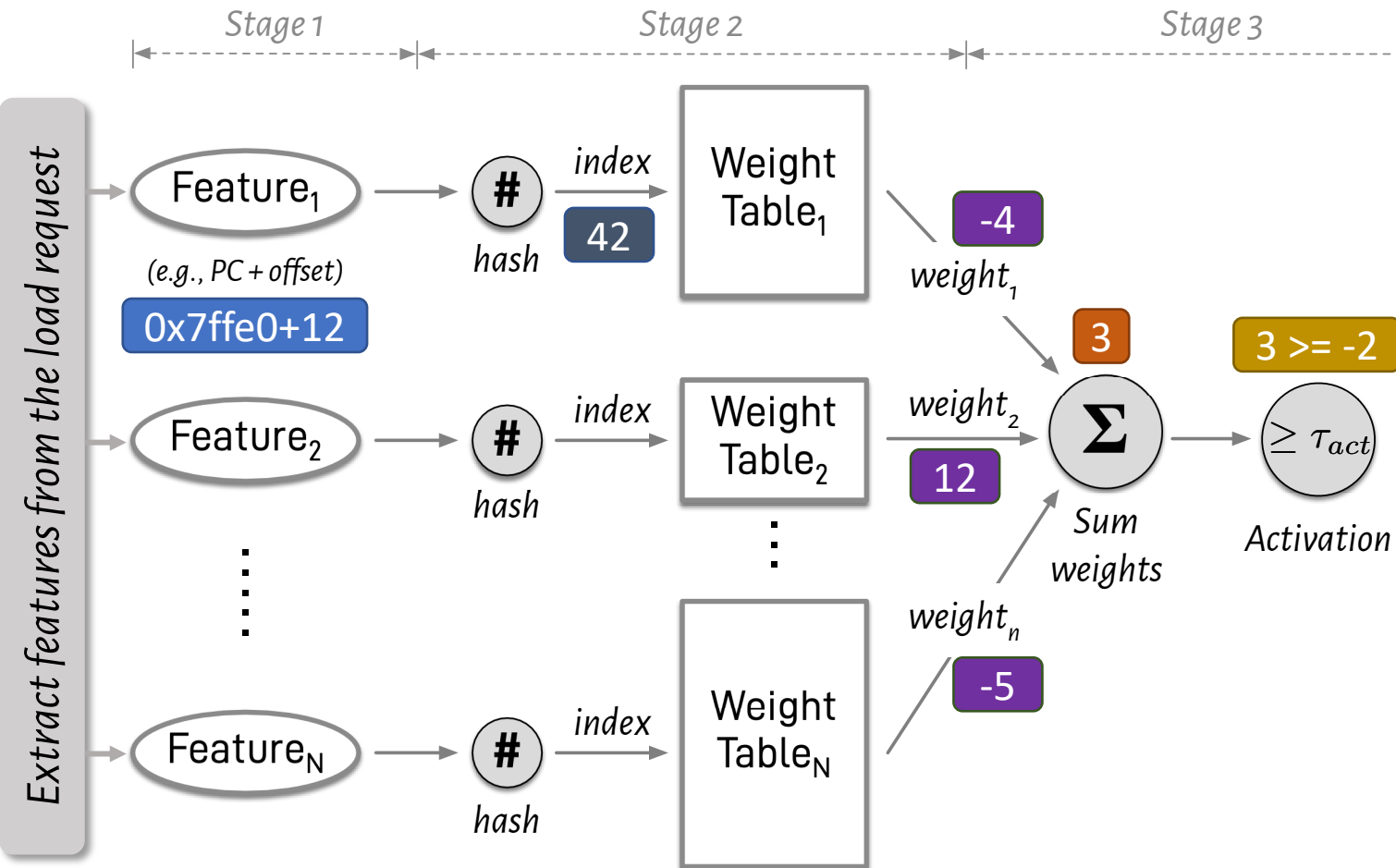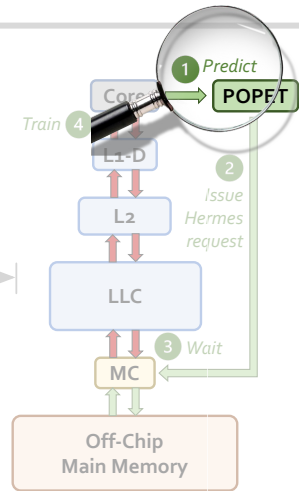
Predicting using POPET

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

Training POPET

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

SAFARI

Training POPET

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

Feature$_1$ (e.g., PC + offset) 0x7ffe0+12 → # hash → index 42 → Weight Table$_1$ → weight$_1$ -4

Feature$_2$ → # hash → index → Weight Table$_2$ → weight$_2$ 12

Feature$_N$ → # hash → index → Weight Table$_N$ → weight$_n$ -5

3 → Σ Sum weights → 3 >= -2 → $\geq \tau_{act}$ Activation → ❌

Shouldn't be activated

Predict that the ... ould go off-chip

SAFARI

Training POPET

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

Feature$_1$ (e.g., PC + offset)

0x7ffe0+12

hash → index → 42 → Weight Table$_1$ → weight$_1$ → -4

Feature$_2$ → hash → index → Weight Table$_2$ → weight$_2$ → 12

Feature$_N$ → hash → index → Weight Table$_N$ → weight$_n$ → -5

3

Σ Sum weights

3 >= -2

$\geq \tau_{act}$ Activation

Predict that the ... ould go off-chip

Shouldn't be activated

Cumulative weight $< \tau_{act}$

SAFARI

50

Training POPET

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

$Feature_1$

(e.g., PC + offset)

`0x7ffe0+12`

hash $\#$ → index → Weight Table$_1$ → 42

$weight_1$ → -4

$Feature_2$ → hash $\#$ → index → Weight Table$_2$ → $weight_2$ → 12

$Feature_N$ → hash $\#$ → index → Weight Table$_N$ → $weight_n$ → -5

Σ → 3

Sum weights

$\geq \tau_{act}$ → 3 >= -2

Activation

Shouldn't be activated

Cumulative weight $< \tau_{act}$

Predict that the req should go off-chip

SAFARI

51

Training POPET

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

Evaluation

Simulation Methodology

- **ChampSim** trace driven simulator

SAFARI

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** memory-intensive trace mixes

SAFARI

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** memory-intensive trace mixes

LLC Prefetchers

- Pythia [Bera+, MICRO'21]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

SAFARI

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** memory-intensive trace mixes

LLC Prefetchers

- Pythia [Bera+, MICRO'21]
- Bingo [Bakshalipour+, HPCA'19]
- MLOP [Shakerinava+, 3rd Prefetching Championship'19]
- SPP + Perceptron filter [Bhatia+, ISCA'20]
- SMS [Somogyi+, ISCA'06]

Off-Chip Predictors

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

- **Ideal Off-chip Predictor**

Latency Configuration

- **Cache *round-trip* latency**

SAFARI

Latency Configuration

- **Cache *round-trip* latency**
  - L1-D: **5** cycles

SAFARI

Latency Configuration

- **Cache *round-trip* latency**
  - L1-D: **5** cycles
  - L2: **15** cycles

SAFARI

Latency Configuration

- **Cache _round-trip_ latency**
  - L1-D: **5** cycles
  - L2: **15** cycles
  - LLC: **55** cycles

SAFARI

Latency Configuration

- **Cache *round-trip* latency**
  - L1-D: **5** cycles
  - L2: **15** cycles
  - LLC: **55** cycles

- **Hermes request issue latency**
  (incurred after address translation)

  Depends on
  - **Interconnect** between POPET and MC

Latency Configuration

- **Cache *round-trip* latency**
  - L1-D: **5** cycles
  - L2: **15** cycles
  - LLC: **55** cycles

- **Hermes request issue latency**
  (incurred after address translation)

  Depends on
  - **Interconnect** between POPET and MC

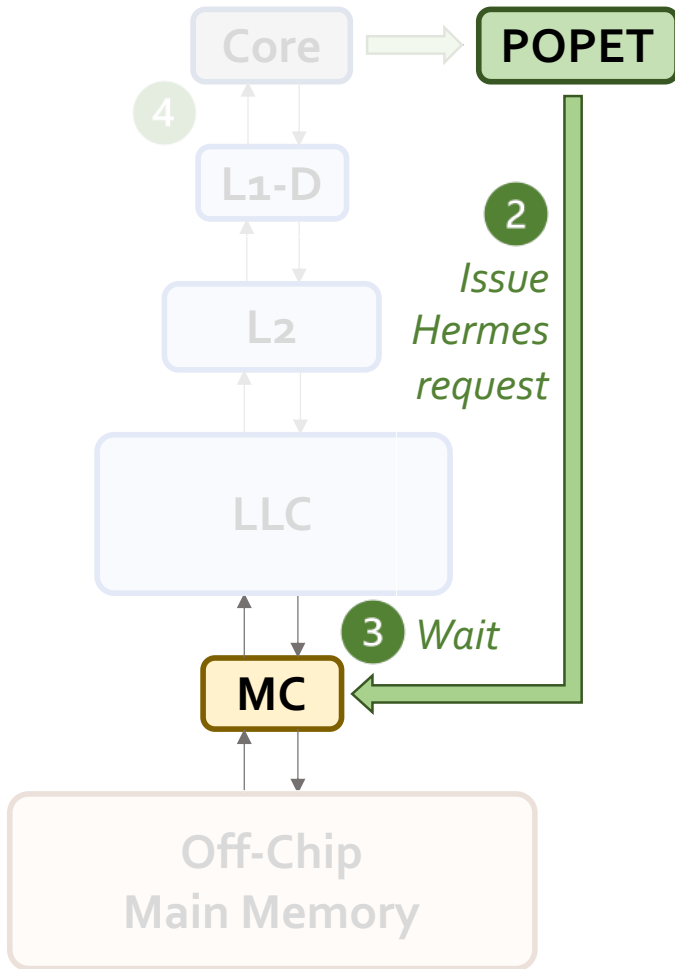 **0** *cycles* ———————————— **24** *cycles*

Latency Configuration
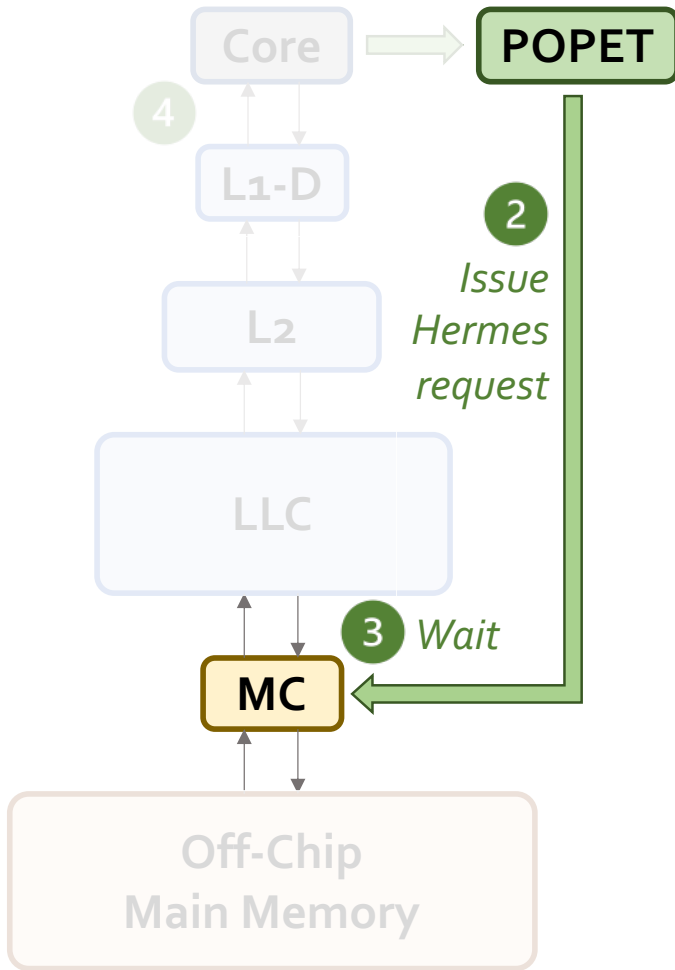
- **Cache *round-trip* latency**
    - L1-D: **5** cycles
    - L2: **15** cycles
    - LLC: **55** cycles
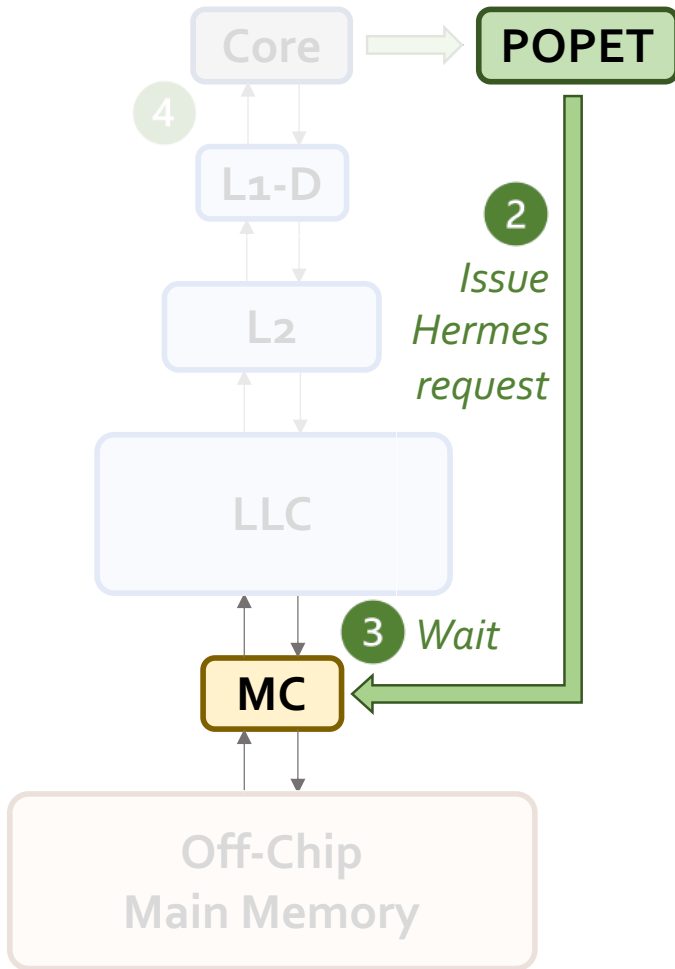
- **Hermes request issue latency**
  (incurred after address translation)

  Depends on
    - **Interconnect** between POPET and MC

Single-Core Performance Improvement
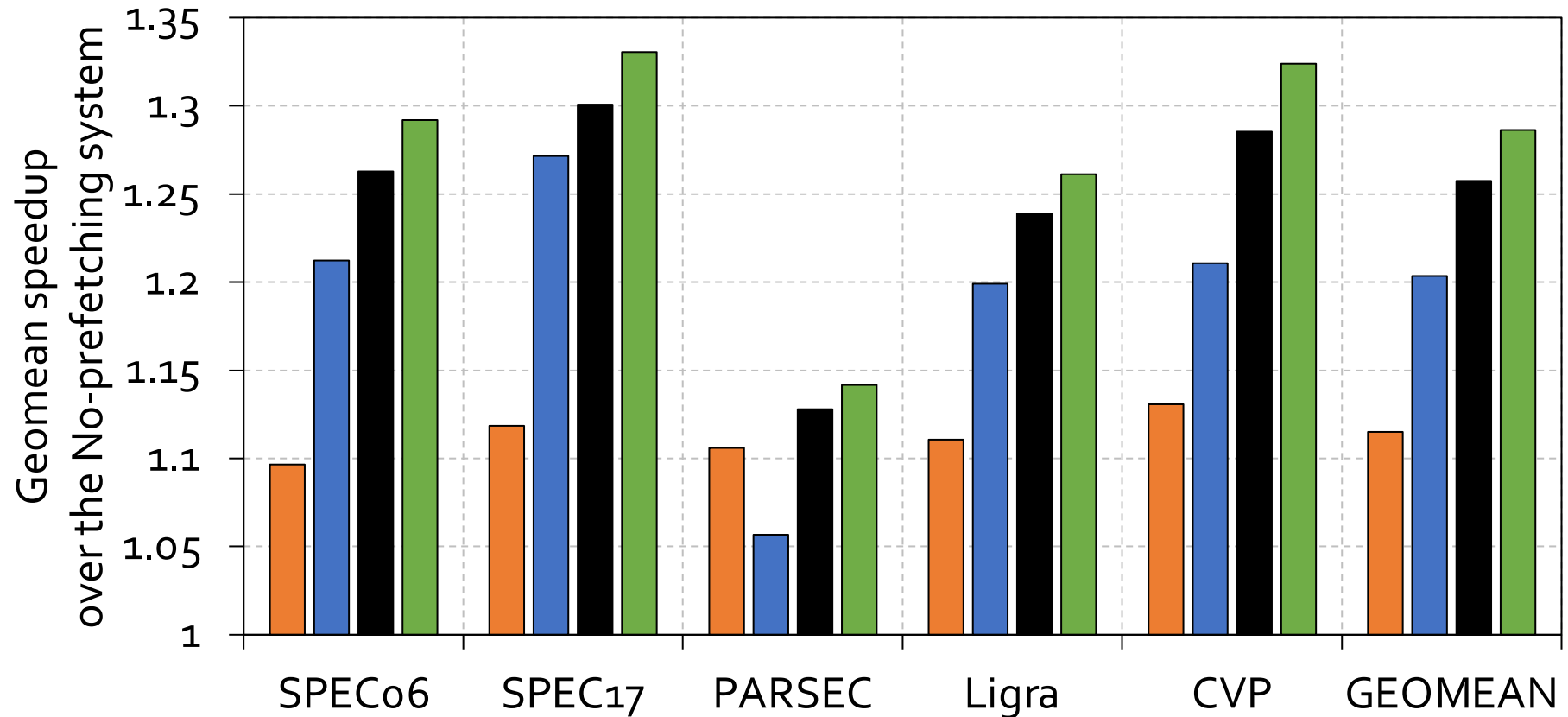
Single-Core Performance Improvement
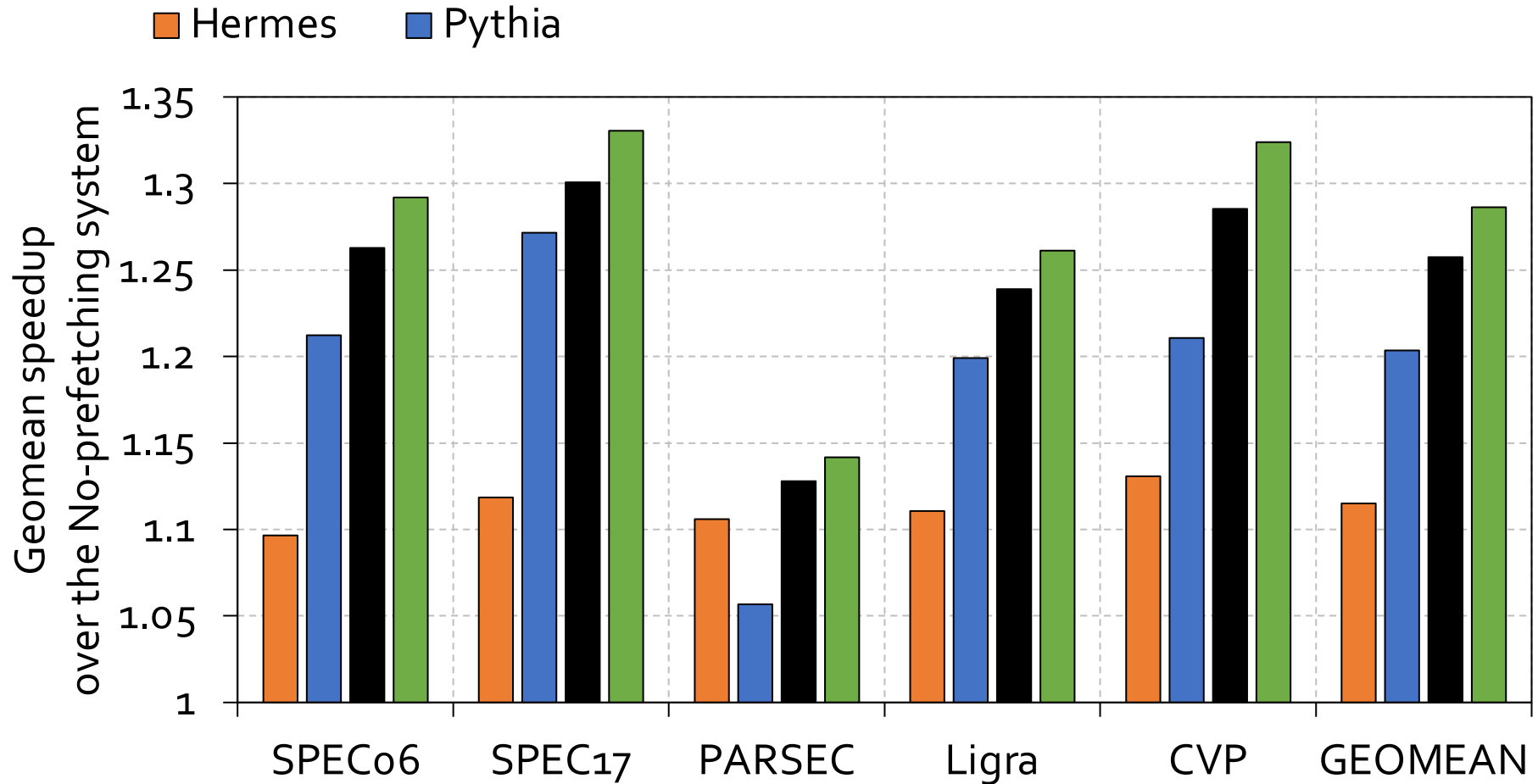
Single-Core Performance Improvement
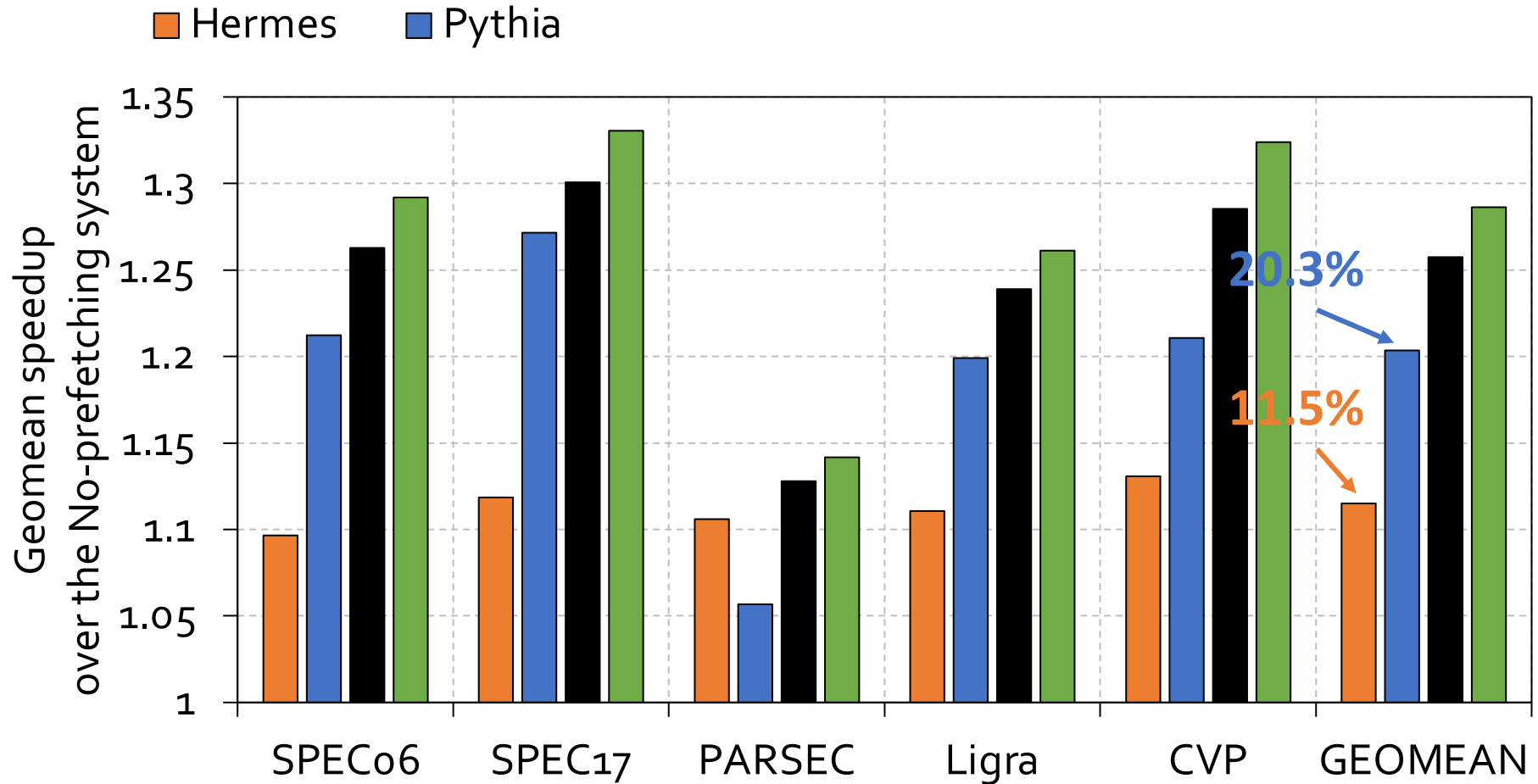
Single-Core Performance Improvement
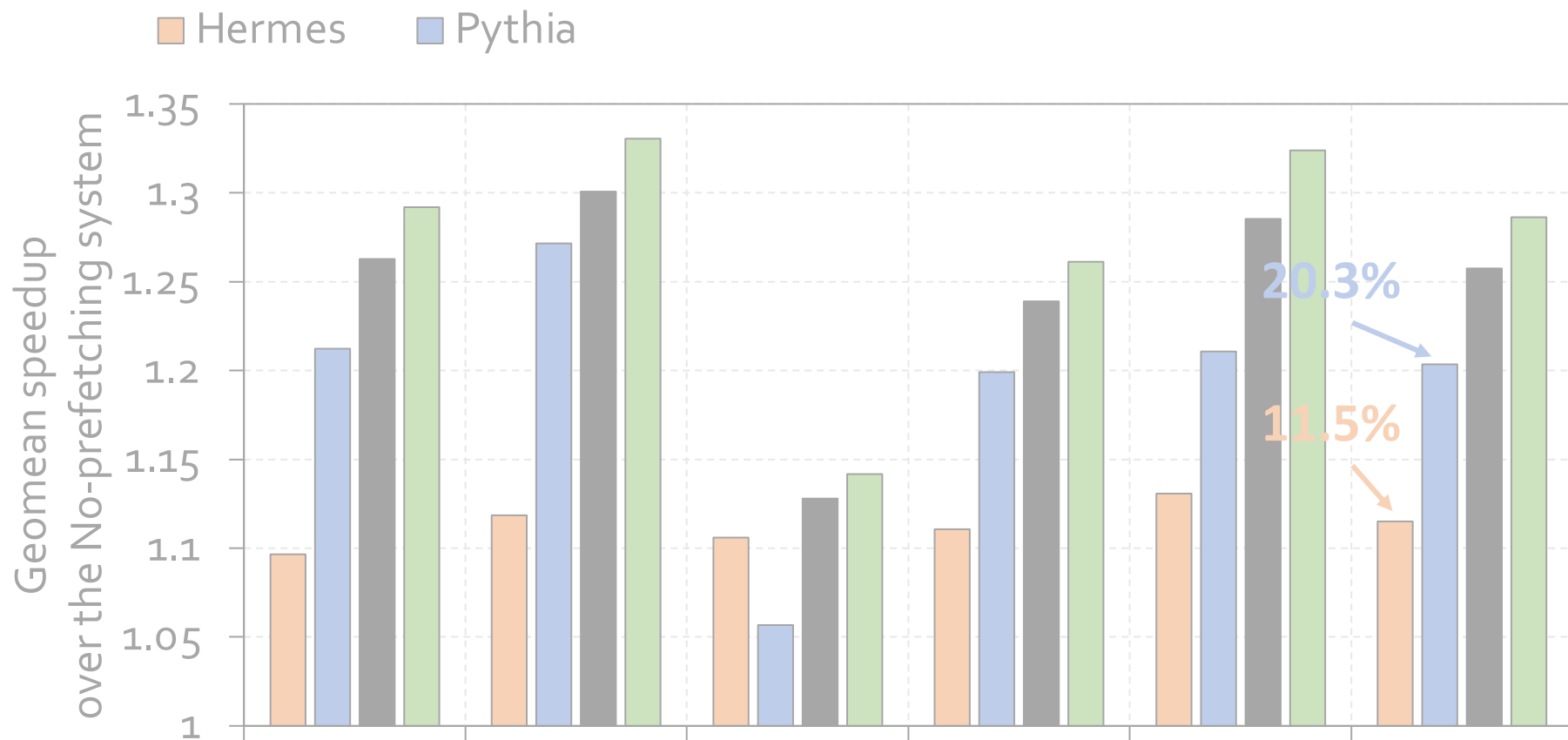
Hermes alone provides nearly 50% performance benefits of Pythia with only 1/5th storage overhead

Single-Core Performance Improvement

SAFARI

Single-Core Performance Improvement

Legend: ■ Hermes ■ Pythia ■ Pythia + Hermes

Y-axis: Geomean speedup over the No-prefetching system

X-axis categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

5.4%

Single-Core Performance Improvement

Legend: ■ Hermes ■ Pythia ■ Pythia + Hermes

Y-axis: Geomean speedup over the No-prefetching system (1 to 1.35)

X-axis categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

Annotation: 5.4%
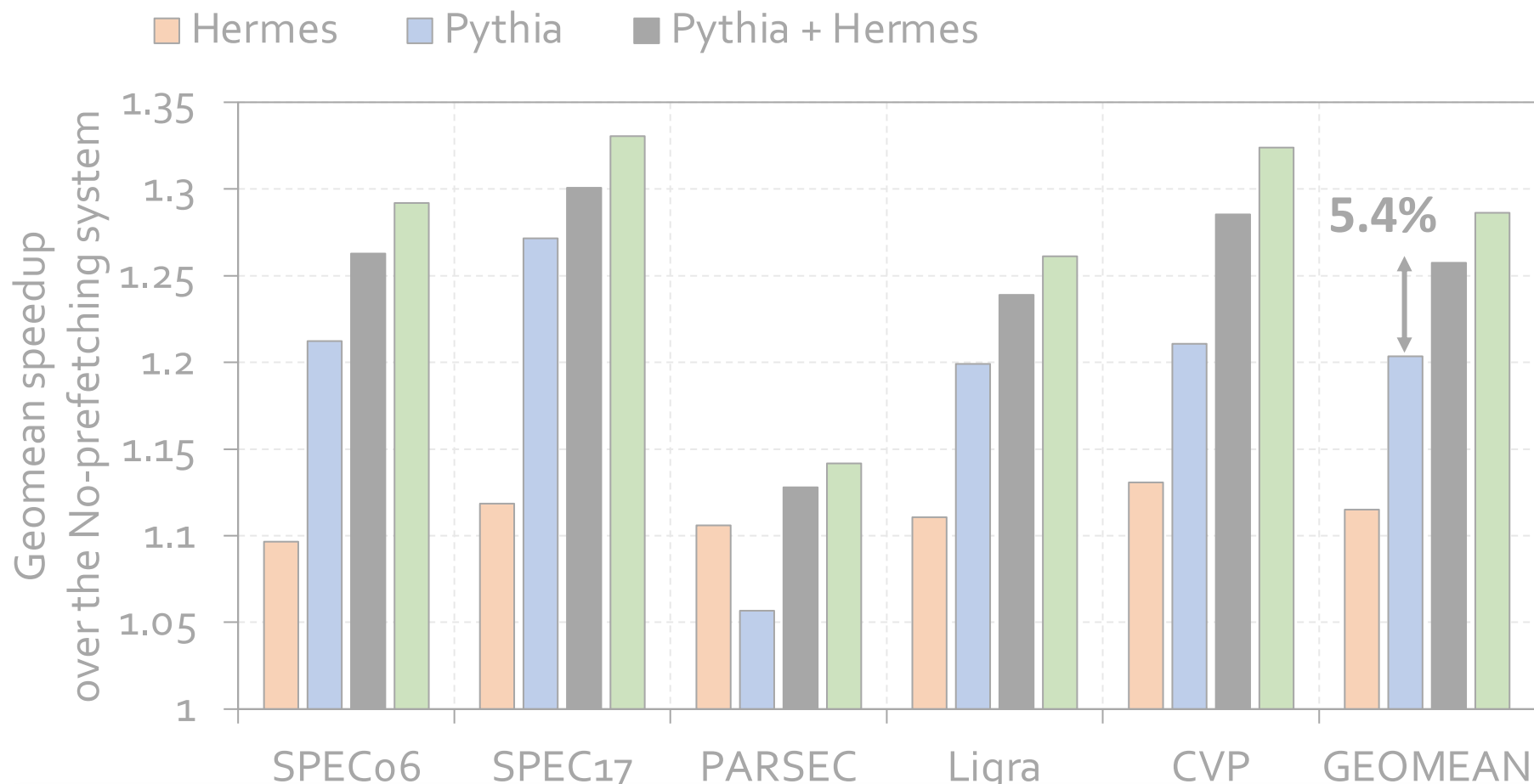
Hermes on top of Pythia outperforms Pythia alone in every workload category

Single-Core Performance Improvement

Legend: ■ Hermes ■ Pythia ■ Pythia + Hermes ■ Pythia + Ideal Hermes

Y-axis: Geomean speedup over the No-prefetching system (1 to 1.35)

X-axis categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

Single-Core Performance Improvement

SAFARI

Single-Core Performance Improvement

Legend: ■ Hermes ■ Pythia ■ Pythia + Hermes ■ Pythia + Ideal Hermes

Y-axis: Geomean speedup over the No-prefetching system (1 to 1.35)

X-axis categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN
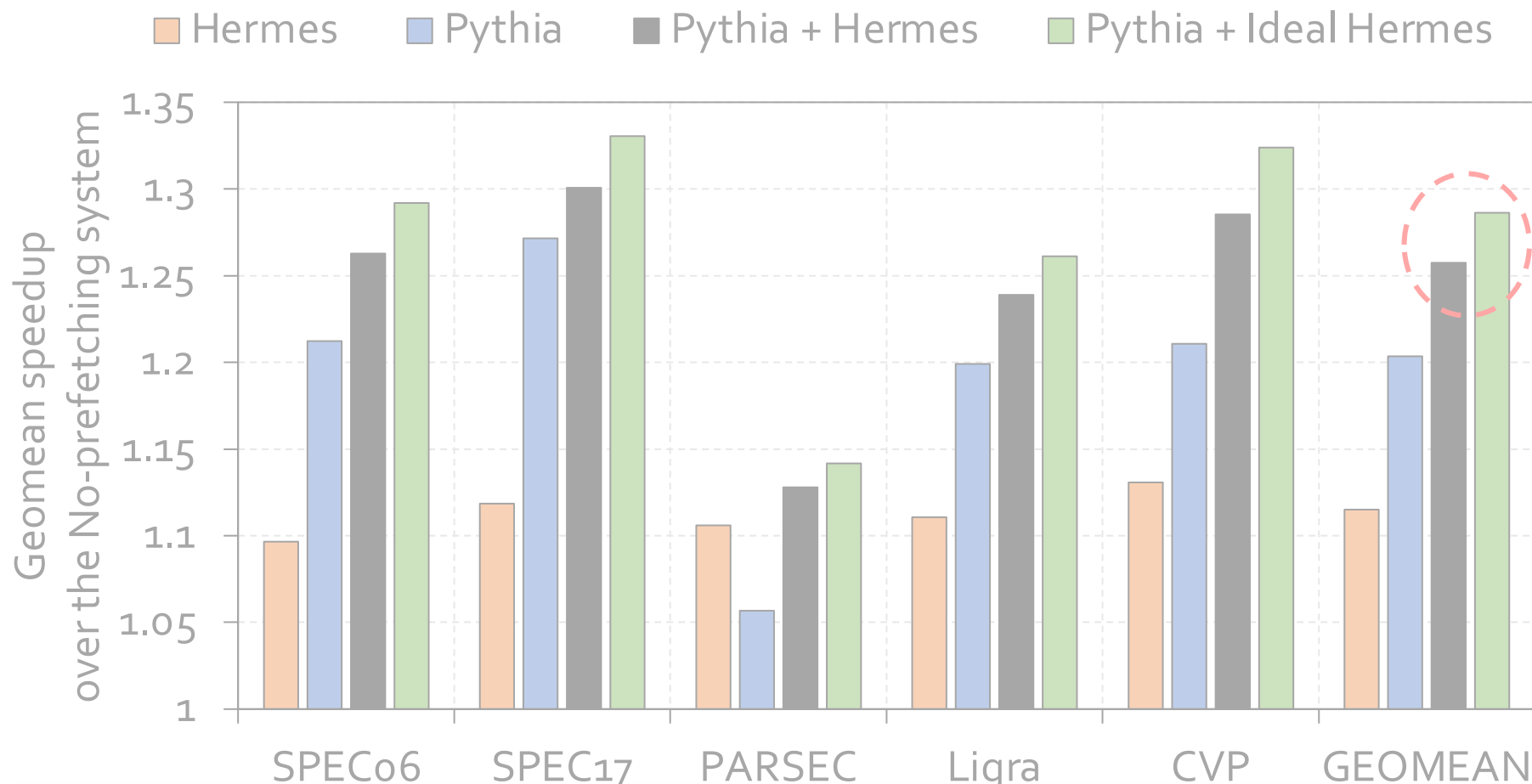
Hermes provides nearly **90%** performance benefit of **Ideal Hermes** that has an **ideal off-chip load predictor**

Increase in Main Memory Requests

Legend: Hermes (orange), Pythia (blue), Pythia + Hermes (black)

Y-axis: % increase in main memory requests over the No-prefetching system

X-axis categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG

Increase in Main Memory Requests

Chart: % increase in main memory requests over the No-prefetching system

Legend: Hermes (orange), Pythia (blue), Pythia + Hermes (black)

Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG

Annotations on AVG: Hermes 5.5%, Pythia 38.5%, Pythia + Hermes 5.9%

Increase in Main Memory Requests

SAFARI

Increase in Main Memory Requests

For **every 1% performance** benefit,
increase in main memory requests

Chart: % increase in main memory requests over the No-prefetching system, across SPEC06, SPEC17, PARSEC, Ligra, CVP, and AVG.

AVG annotations: 5.5%, 38.5%, 5.9%

78

Increase in Main Memory Requests

For **every 1% performance** benefit,
increase in main memory requests

| Pythia | 2% |

% increase in main memory requests over the No-prefetching system

SPEC06 SPEC17 PARSEC Ligra CVP AVG

5.9%
38.5%
5.5%

SAFARI

Increase in Main Memory Requests

For **every 1% performance** benefit,
increase in main memory requests

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

Increase in Main Memory Requests

For **every 1% performance** benefit, increase in main memory requests

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

% increase in main memory requests over the No-prefetching system

- 60%
- 50%
- 40%
- 30%
- 20%
- 10%
- 0%

SPEC06   SPEC17   PARSEC   Ligra   CVP   AVG

5.9%
38.5%
5.5%
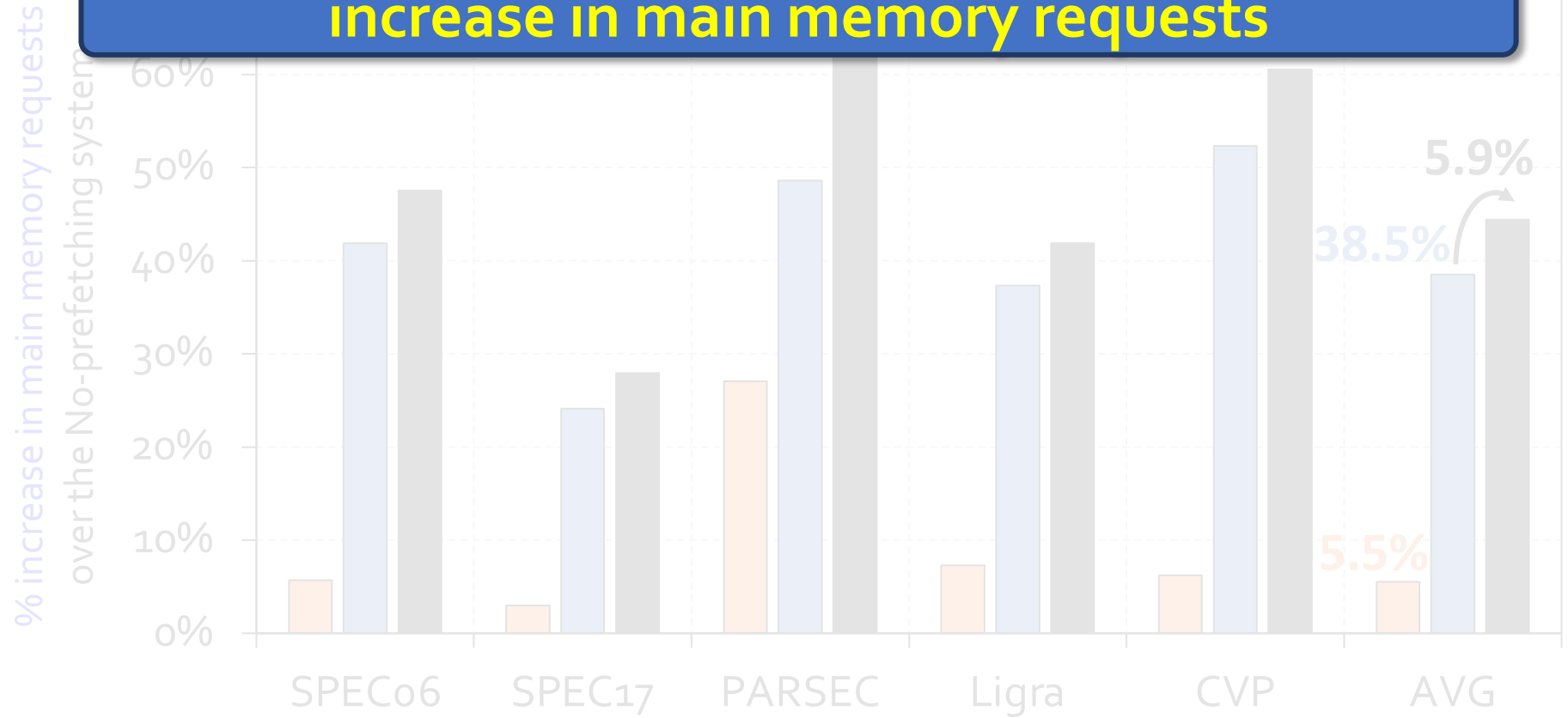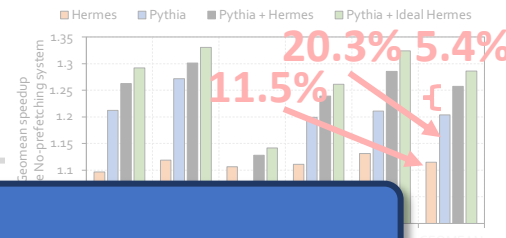
Increase in Main Memory Requests

For **every 1% performance** benefit,
increase in main memory requests

Pythia	2%
Hermes on top of Pythia	1%
Hermes alone	0.5%
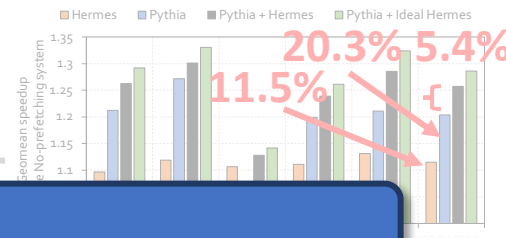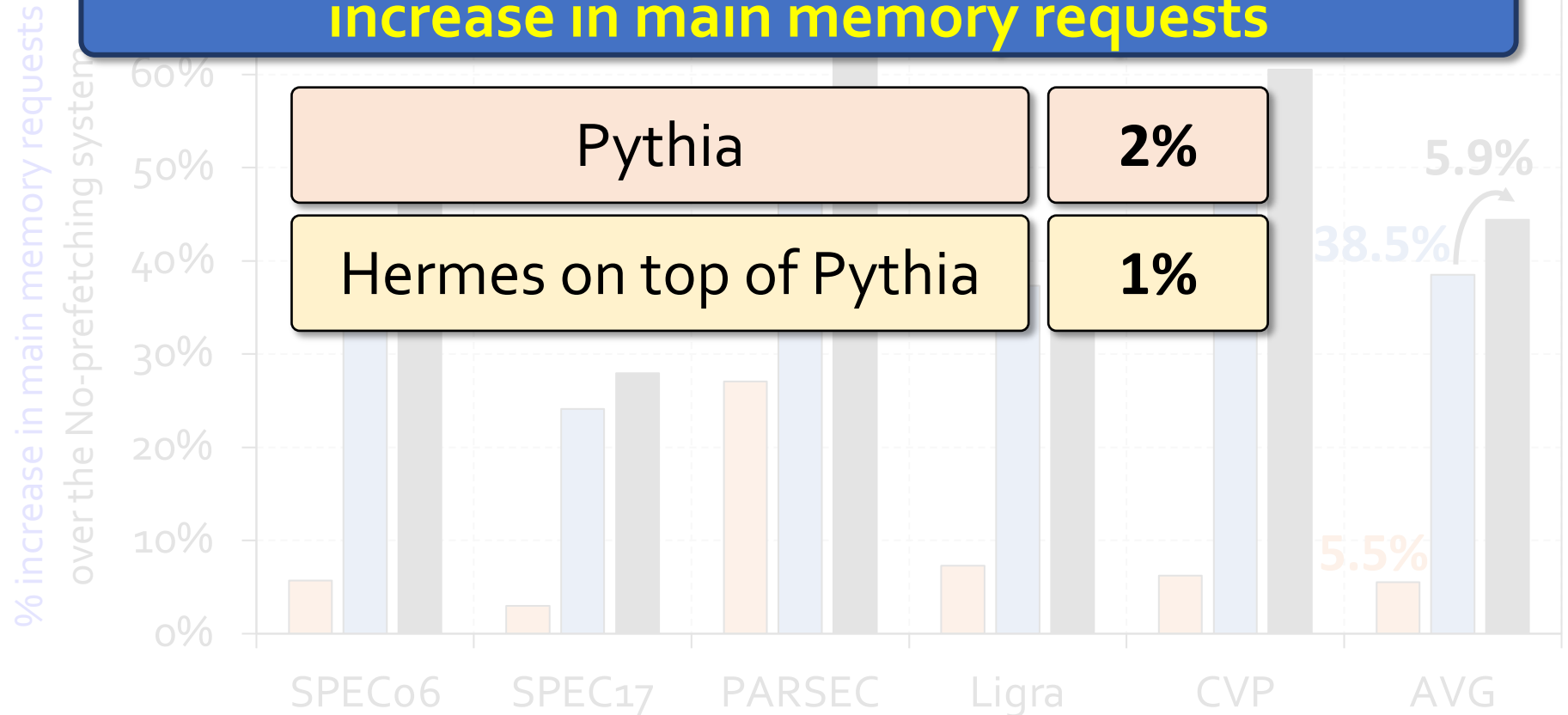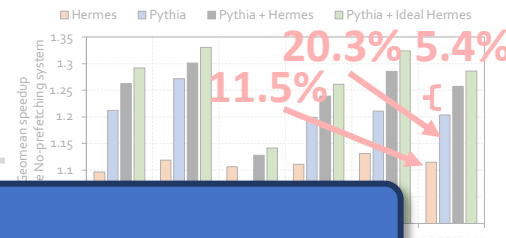
Hermes is more **bandwidth-efficient**
than even an efficient prefetcher like Pythia

SAFARI

Performance with Varying Memory Bandwidth

Performance with Varying Memory Bandwidth

Geomean speedup over the No-prefetching system vs. Main Memory Bandwidth (in MT/s). Pythia and Hermes curves shown.

Performance with Varying Memory Bandwidth

SAFARI

Performance with Varying Memory Bandwidth

Performance with Varying Memory Bandwidth

Geomean speedup over the No-prefetching system

Pythia

Hermes

Baseline

~Intel Xeon 6258R (Cascade Lake, 28C/6ch, 2020)

~AMD EPYC Rome 7702P (Zen 2, 64C/8ch, 2019)

~AMD Threadripper 3990x (Zen 2, 64C/4ch, 2020)

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

Performance with Varying Memory Bandwidth
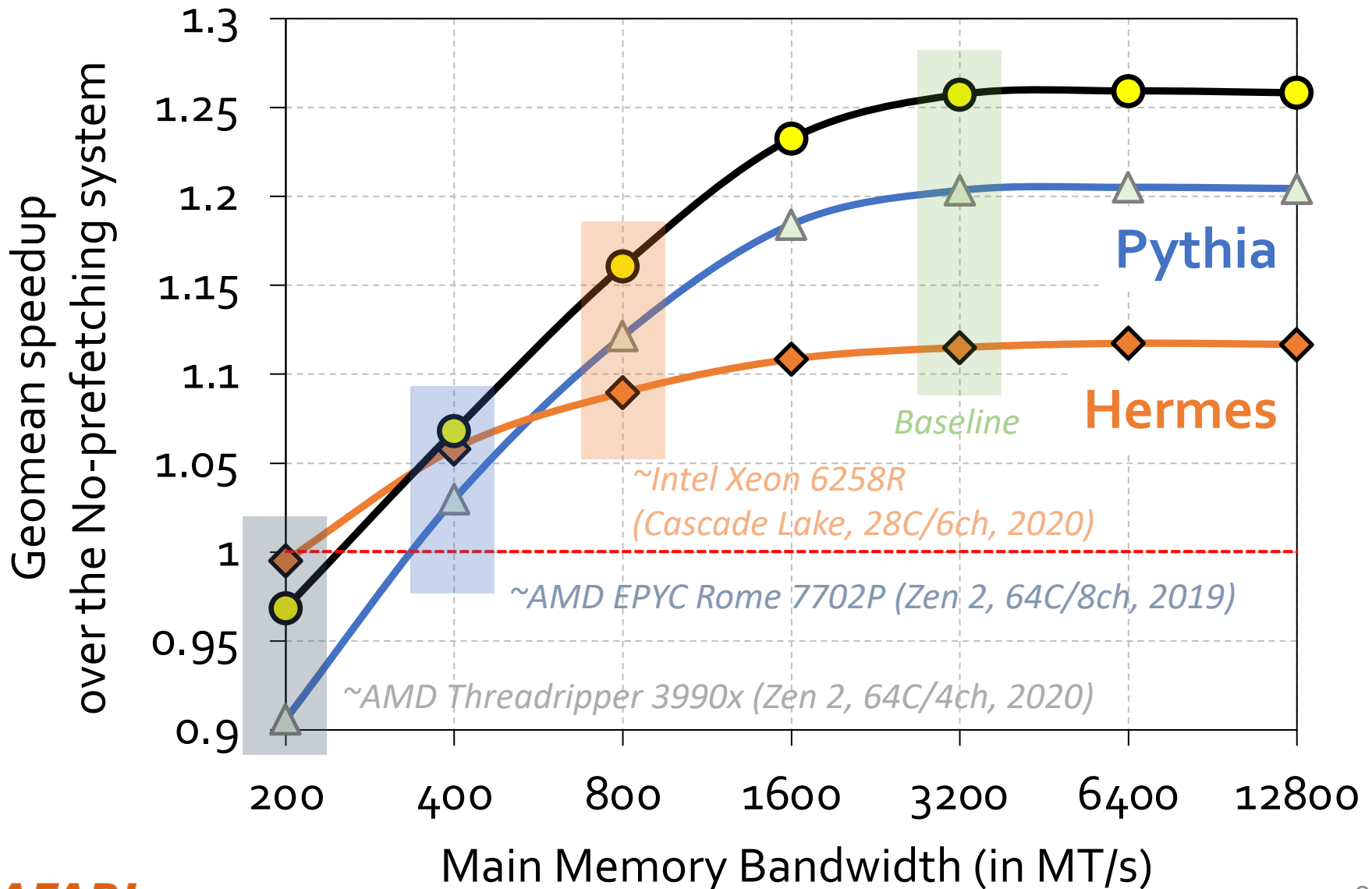
Performance with Varying Memory Bandwidth

Geomean speedup over the No-prefetching system vs. Main Memory Bandwidth (in MT/s)

Pythia+Hermes

Pythia

Hermes

Baseline

~Intel Xeon 6258R (Cascade Lake, 28C/6ch, 2020)

~AMD EPYC Rome 7702P (Zen 2, 64C/8ch, 2019)

~AMD Threadripper 3990x (Zen 2, 64C/4ch, 2020)

Performance with Varying Memory Bandwidth

Pythia+Hermes

Pythia

Hermes

Baseline

Geomean speedup over the No-prefetching system

~Intel Xeon 6258R
(Cascade Lake, 28C/6ch, 2020)

~AMD EPYC Rome 7702P (Zen 2, 64C/8ch, 2019)

~AMD Threadripper 3990x (Zen 2, 64C/4ch, 2020)
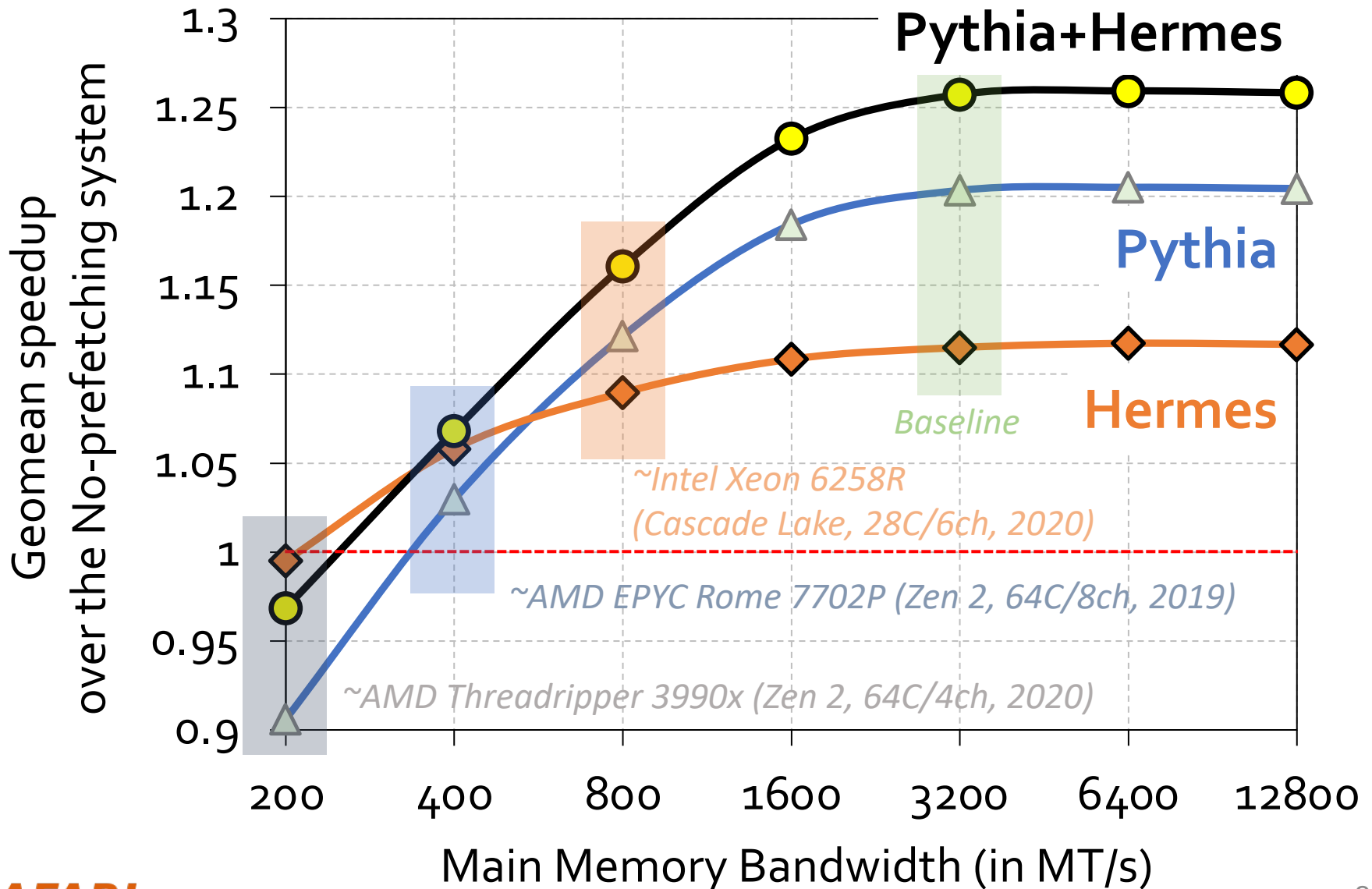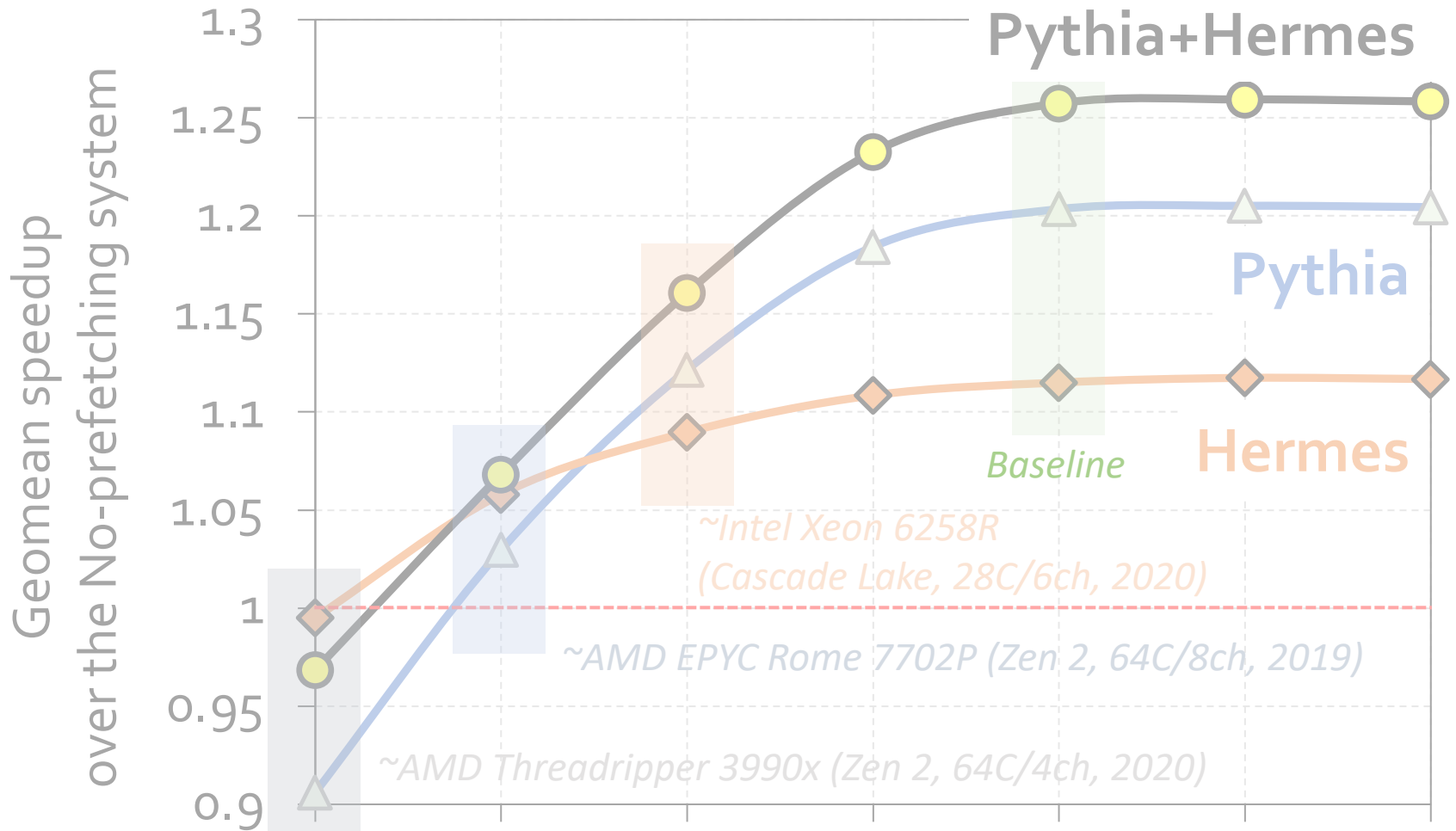
Hermes+Pythia outperforms Pythia
across all bandwidth configurations

Performance with Varying Baseline Prefetcher
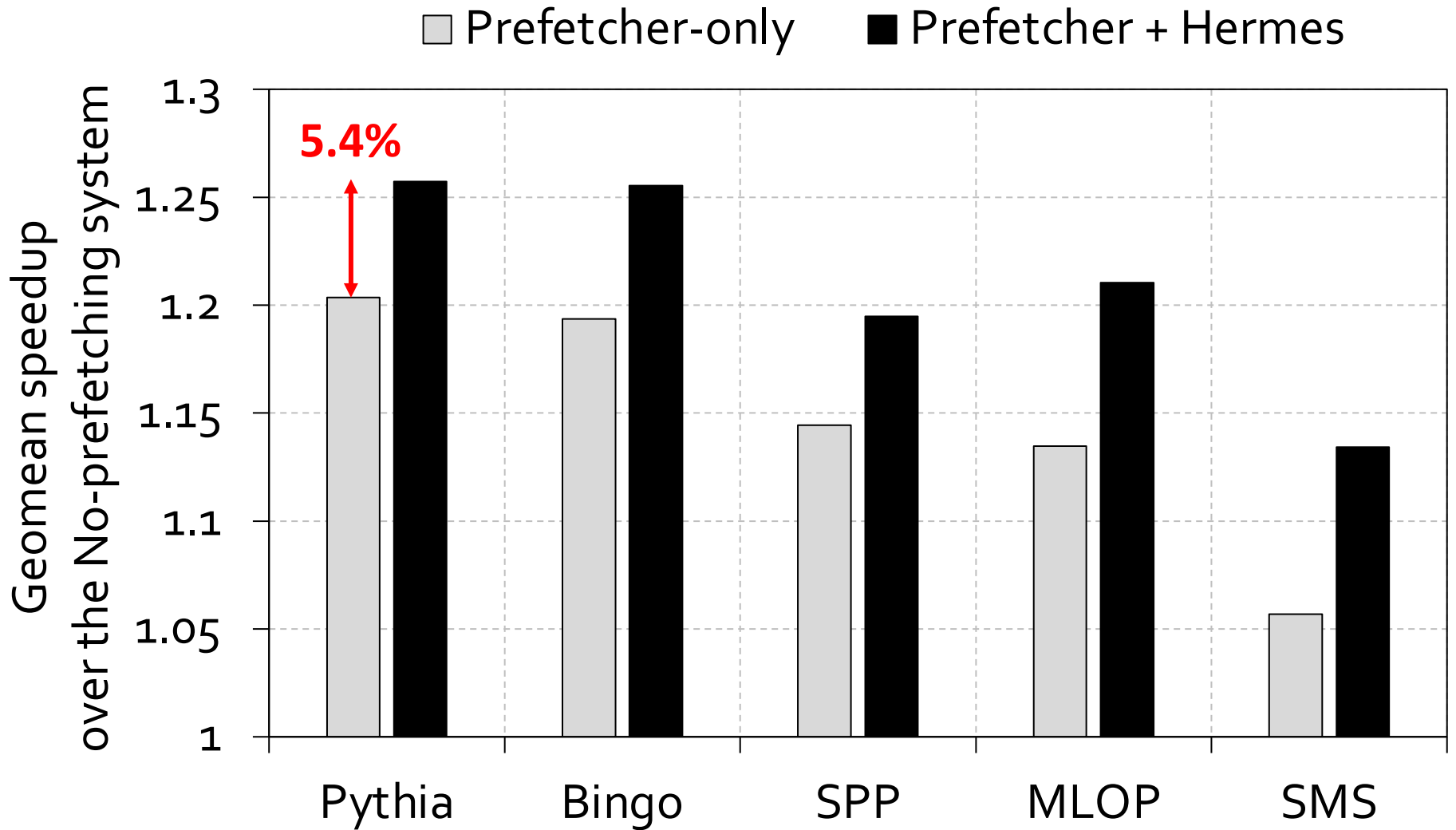
Prefetcher-only   Prefetcher + Hermes

5.4%

Geomean speedup over the No-prefetching system

Pythia   Bingo   SPP   MLOP   SMS

Performance with Varying Baseline Prefetcher

SAFARI

Performance with Varying Baseline Prefetcher

Legend: ☐ Prefetcher-only   ■ Prefetcher + Hermes

Y-axis: Geomean speedup over the No-prefetching system

Values: 5.4% (Pythia), 6.2% (Bingo), 5.1% (SPP), 7.6% (MLOP)

X-axis labels: Pythia, Bingo, SPP, MLOP, SMS

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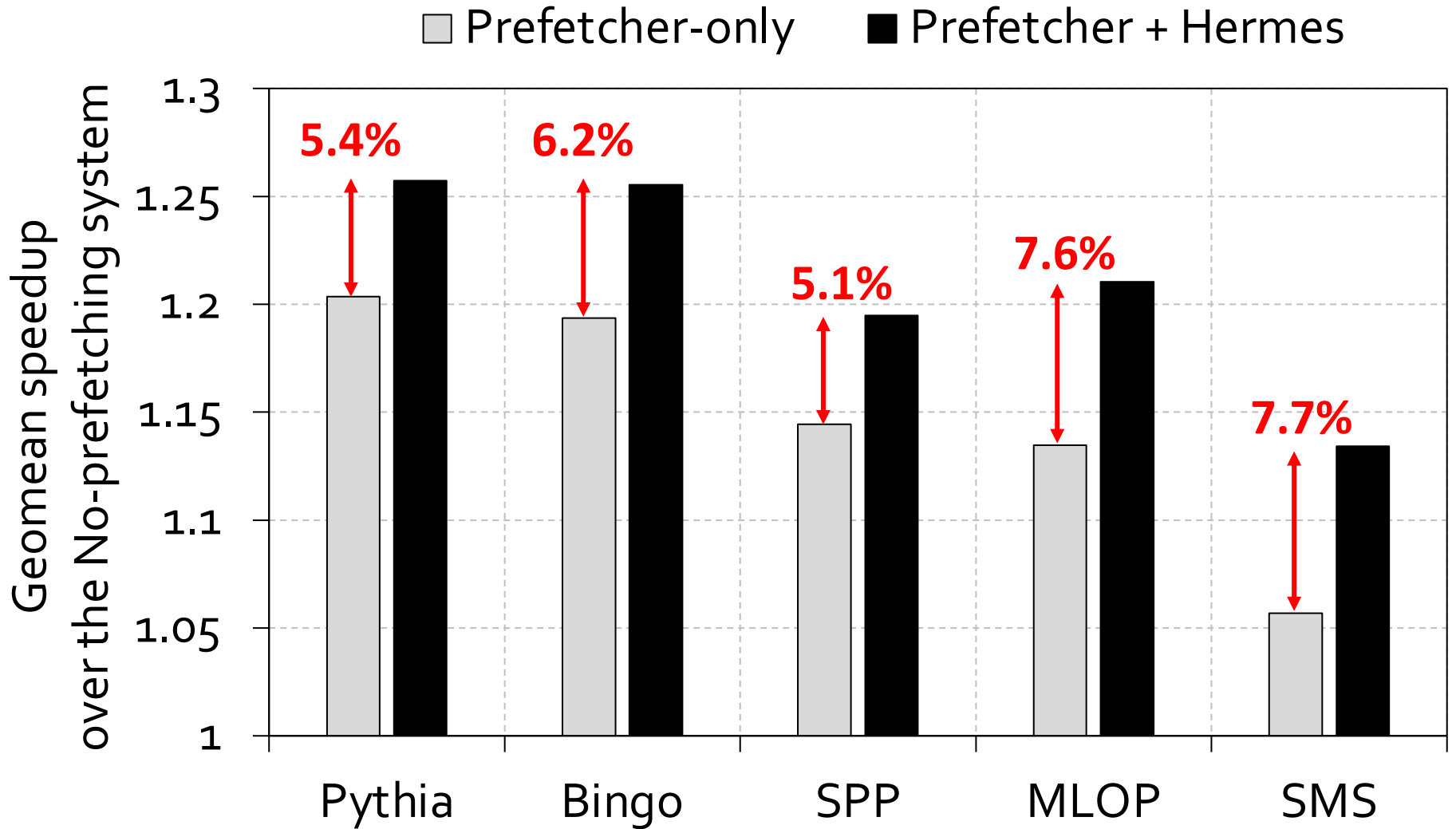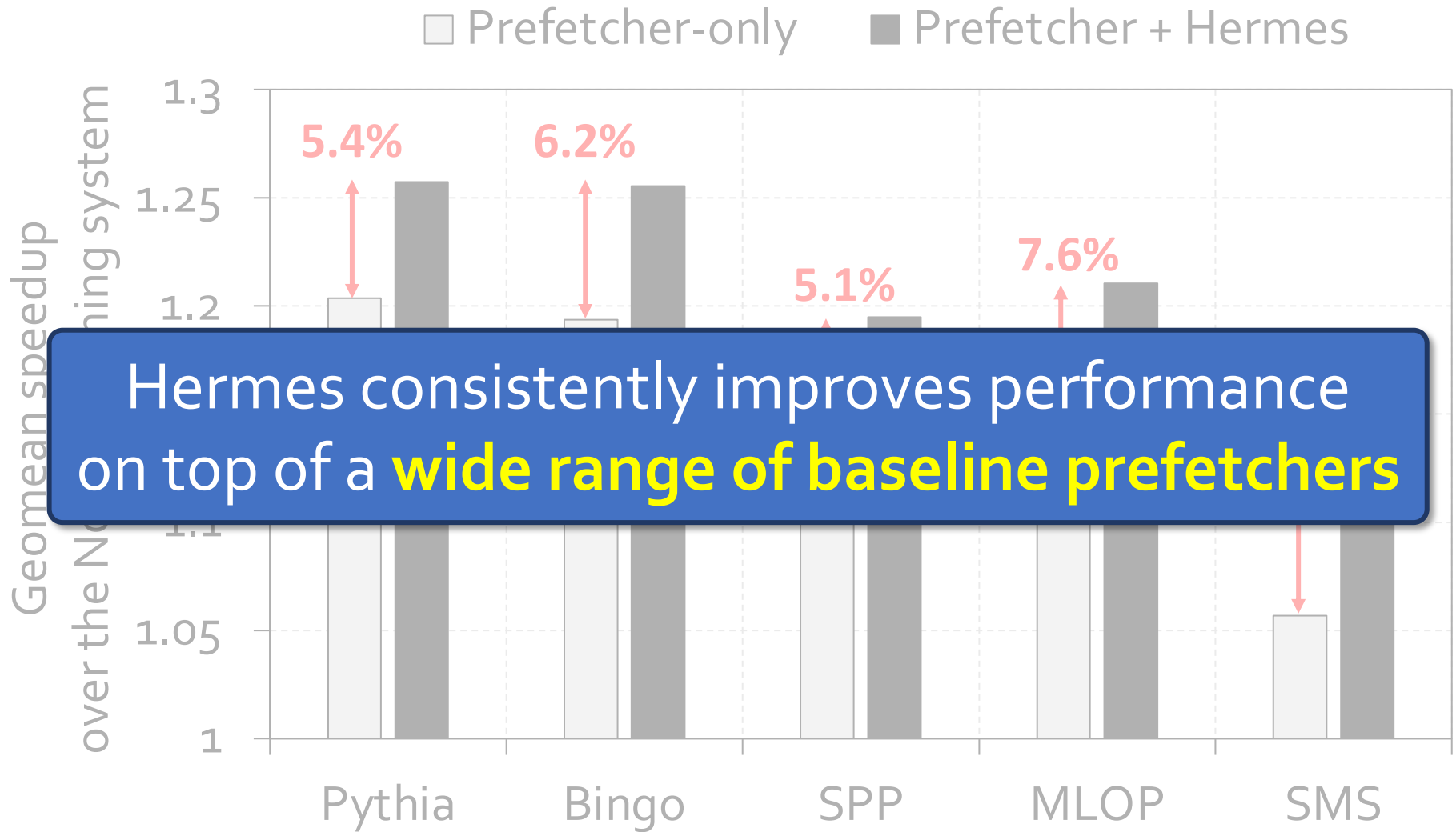
Overhead of Hermes

✅ **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

SAFARI

More in the Paper

- Performance sensitivity to:
    - Cache hierarchy access latency
    - Hermes request issue latency
    - Activation threshold
    - ROB size (**in extended version on arXiv**)
    - LLC size (**in extended version on arXiv**)

- **Accuracy, coverage, and performance analysis** against **HMP** and **TTP**

- Understanding **usefulness of each program feature**

- Effect on **stall cycle reduction**

- **Performance** analysis on an **eight-core** system

More in the Paper

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

Rahul Bera[1]   Konstantinos Kanellopoulos[1]   Shankar Balachandran[2]   David Novo[3]

Ataberk Olgun[1]   Mohammad Sadrosadati[1]   Onur Mutlu[1]

[1]ETH Zürich   [2]Intel Processor Architecture Research Lab   [3]LIRMM, Univ. Montpellier, CNRS

Long-latency load requests continue to limit the performance of modern high-performance processors. To increase the latency tolerance of a processor, architects have primarily relied on two key techniques: sophisticated data prefetchers and large on-chip caches. In this work, we show that: (1) even a sophisticated state-of-the-art prefetcher can only predict half of the off-chip load requests on average across a wide range of workloads, and (2) due to the increasing size and complexity of on-chip caches, a large fraction of the latency of an off-chip load request is spent accessing the on-chip cache hierarchy to solely determine that it needs to go off-chip.

The goal of this work is to accelerate off-chip load requests by removing the on-chip cache access latency from their critical path. To this end, we propose a new technique called Hermes, whose key idea is to: (1) accurately predict which load requests

off-chip main memory (i.e., an *off-chip load*) often stalls the processor core by blocking the instruction retirement from the re-order buffer (ROB), thus limiting the core's performance [88, 91, 92]. To increase the latency tolerance of a core, computer architects primarily rely on two key techniques. First, they employ increasingly sophisticated hardware prefetchers that can learn complex memory address patterns and fetch data required by future load requests before the core demands them [28, 32, 33, 35, 75]. Second, they significantly scale up the size of the on-chip cache hierarchy with each new generation of processors [10, 11, 16].

Key problem. Despite recent advances in processor core design, we observe two key trends in new processor designs that leave a significant opportunity for performance improvement on the table. First, even a sophisticated state-of-the-art

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

To Summarize…

Summary

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

SAFARI

Summary

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

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

SAFARI

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Low storage overhead
(4KB/core)

SAFARI

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Low storage overhead
(4KB/core)

High performance improvement
over best prior baseline
(5.4%)

Summary

Hermes employs **the first perceptron-based** off-chip load predictor

High accuracy
(77%)

High coverage
(74%)

Low storage overhead
(4KB/core)

High performance improvement
over best prior baseline
(5.4%)

High performance per bandwidth

Hermes is Open Sourced

Hermes is Open Sourced

All workload traces

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]

Artifacts Available V1.1

Artifacts Evaluated Functional V1.1

Results Reproduced V1.1

CMU-SAFARI Public

<> Code

main

Rahul Bera Minor changes in experiment file   9eaeca8 5 days ago   21 commits

2 months ago

5 days ago

2 months ago

experiments   Minor changes in experiment file   5 days ago

All experiment files and rollup script...   6 days ago

2 months ago

2 months ago

2 months ago

experiements in parallel when loc...   25 days ago

5 days ago

2 months ago

2 months ago

ct_traces.csv   2 months ago

an example python script to genera...   25 days ago

2 months ago

2 months ago

A speculative mechanism to accelerate long-latency off-chip load requests by removing on-chip cache access latency from their critical path, as described by MICRO 2022 paper by Bera et al. (https://arxiv.org/pdf/2209.00188.pdf)

🔗 arxiv.org/abs/2209.00188

machine-learning   cache   perceptron
computer-architecture   microarchitecture
perceptron-learning-algorithm   prefetching

Readme
MIT, Unknown licenses found
6 stars
3 watching
1 fork

Releases 5
v1.2 Latest
6 days ago
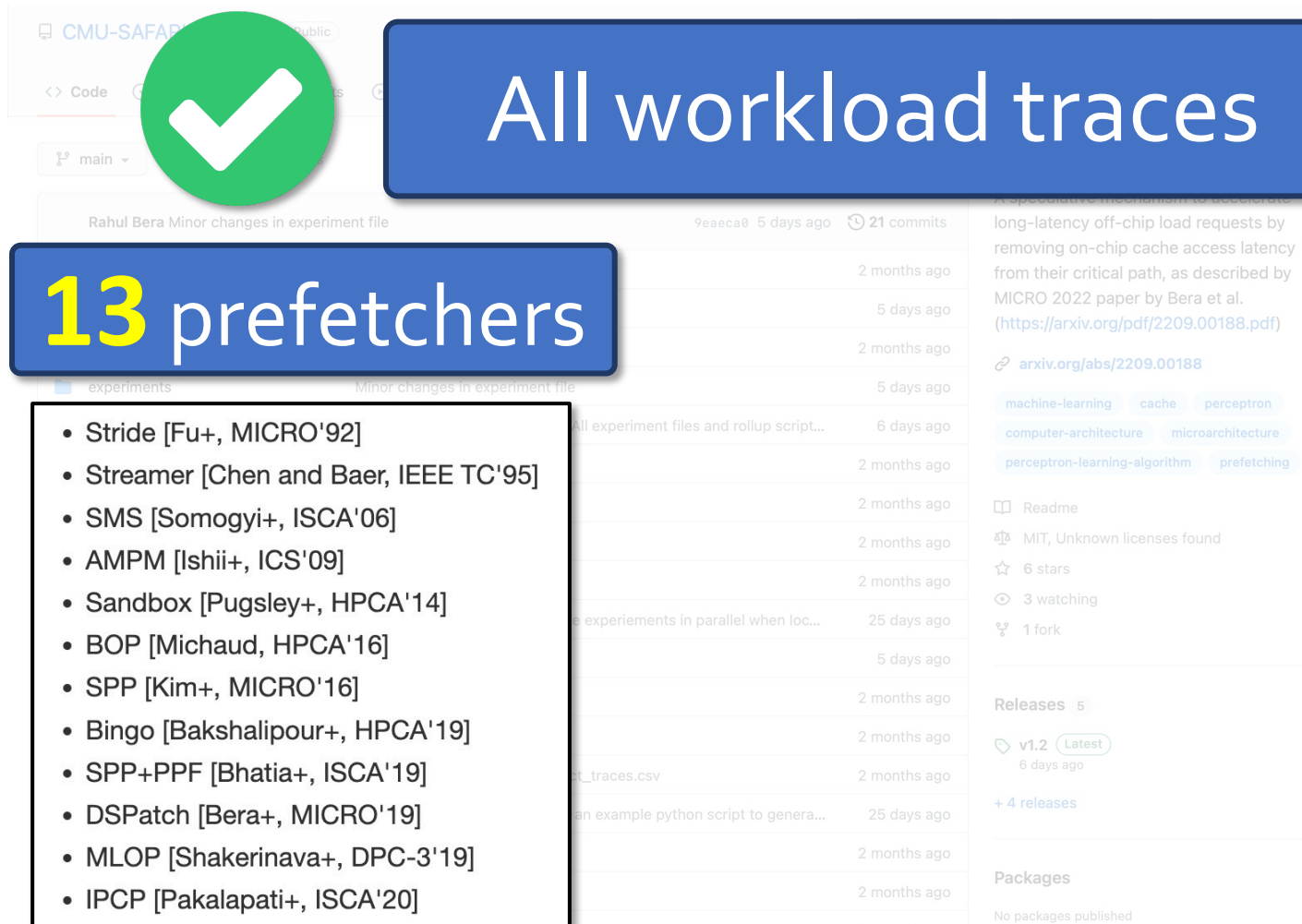+ 4 releases

Packages
No packages published

SAFARI   https://github.com/CMU-SAFARI/Hermes   108

Hermes is Open Sourced

All workload traces

13 prefetchers

9 off-chip predictors

- 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]

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

SAFARI

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

Easy To Define Your Own Off-Chip Predictor

- Just extend the OffchipPredBase class

```cpp
 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
```

SAFARI

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   }
```

SAFARI

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 accuracy (stat name *precision*) and coverage (stat name *recall*) 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...

SAFARI

Off-Chip Prediction Can Further Enable…

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

SAFARI

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

Other ideas to improve **performance** and **fairness** in multi-core system design…

Discussion

- **FAQs**
  - What are the selected set of program features?
  - Can you provide some intuition on why these features work?
  - What happens in case of a misprediction?
  - What's the performance headroom for off-chip prediction?
  - Do you see a variance of different features in final prediction accuracy?

- **Simulation Methodology**
  - System parameters
  - Evaluated workloads

- **More Results**
  - Percentage of off-chip requests
  - Reduction in stall cycles by reducing the critical path
  - Fraction of off-chip load requests
  - Accuracy and coverage of POPET
  - Effect of different features
  - Are all features required?
  - 1C performance
  - 1C performance line graph
  - 1C performance against prior predictors
  - Effect on stall cycles
  - 8C performance
  - Sensitivity:
    - Hermes request issue latency
    - Cache hierarchy access latency
    - Activation threshold
    - ROB size
    - LLC size
  - Power overhead
  - Accuracy without prefetcher
  - Main memory request overhead with different prefetchers

SAFARI

BACKUP

Initial Set of Program Features

Features without control-flow information	Features with control-flow information
	8. Load PC
1. Load virtual address	9. PC $\oplus$ load virtual address
2. Virtual page number	10. PC $\oplus$ virtual page number
3. Cacheline offset in page	11. PC $\oplus$ cacheline offset
4. First access	12. PC + first access
5. Cacheline offset + first access	13. PC $\oplus$ byte offset
6. Byte offset in cacheline	14. PC $\oplus$ word offset
7. Word offset in cacheline	15. Last-4 load PCs
	16. Last-4 PCs

Selected Set of Program Features

Five features

- PC $\oplus$ cacheline offset
- PC $\oplus$ byte offset
- PC + first access
- Cacheline offset + first access
- Last-4 load PCs

SAFARI

Selected Set of Program Features

Five features

- PC ⊕ cacheline offset
- PC ⊕ byte offset
- PC ⊕ first access
- Cacheline offset ⊕ first access
- Last-4 load PCs

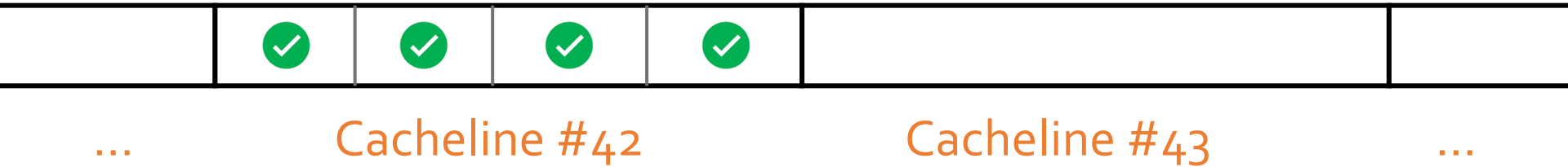A **binary hint** that represents whether or not a cacheblock has been recently touched

SAFARI

When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

...	Cacheline #42	Cacheline #43	...

When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

... Cacheline #42 Cacheline #43 ...

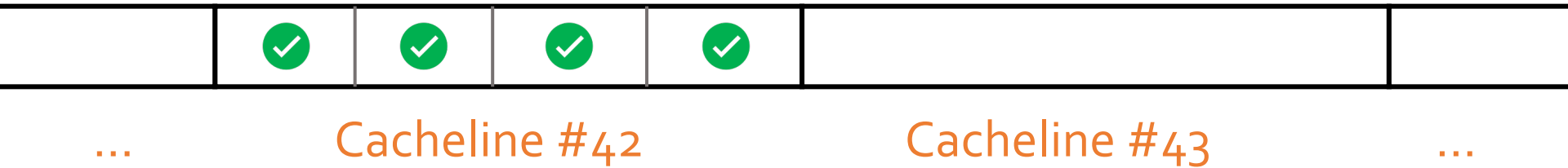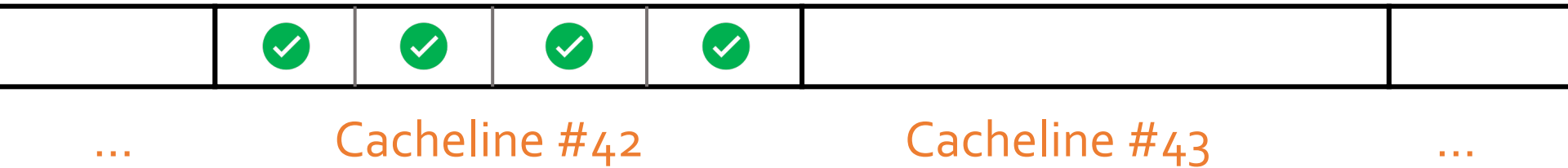# When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

...   ✅ ✅ ✅ ✅   ...

Cacheline #42   Cacheline #43

Without prefetcher

- PC + first access
- Cacheline offset + first access

SAFARI

When A Feature Works/Does Not Work?

Trace: 462.libquantum-1343B

PC: 0x401442

... Cacheline #42 Cacheline #43 ...

Without prefetcher

- PC + first access
- Cacheline offset + first access

With a simple stride prefetcher

- Cacheline offset + first access

SAFARI

What Happens in case of a Misprediction?

- Two cases of mispredictions:

What Happens in case of a Misprediction?

- Two cases of mispredictions:

- **Predicted on-chip but actually goes off-chip**

- **Predicted off-chip but actually is on-chip**

What Happens in case of a Misprediction?

- Two cases of mispredictions:

- **Predicted on-chip but actually goes off-chip**
  - Loss of performance improvement opportunity

- **Predicted off-chip but actually is on-chip**

What Happens in case of a Misprediction?

- Two cases of mispredictions:

- **Predicted on-chip but actually goes off-chip**
  - Loss of performance improvement opportunity

No need for misprediction detection and recovery

- **Predicted off-chip but actually is on-chip**

# What Happens in case of a Misprediction?

- Two cases of mispredictions:

- **Predicted on-chip but actually goes off-chip**
  - Loss of performance improvement opportunity

No need for misprediction detection and recovery

- **Predicted off-chip but actually is on-chip**
  - Memory controller forwards the data to LLC **if and only if** a load to the same address have already missed LLC and arrived at the memory controller

What Happens in case of a Misprediction?

- Two cases of mispredictions:

- **Predicted on-chip but actually goes off-chip**
  - Loss of performance improvement opportunity

No need for misprediction detection and recovery

- **Predicted off-chip but actually is on-chip**
  - Memory controller forwards the data to LLC **if and only if** a load to the same address have already missed LLC and arrived at the memory controller

No need for misprediction detection and recovery

SAFARI

Performance Headroom of Off-Chip Prediction
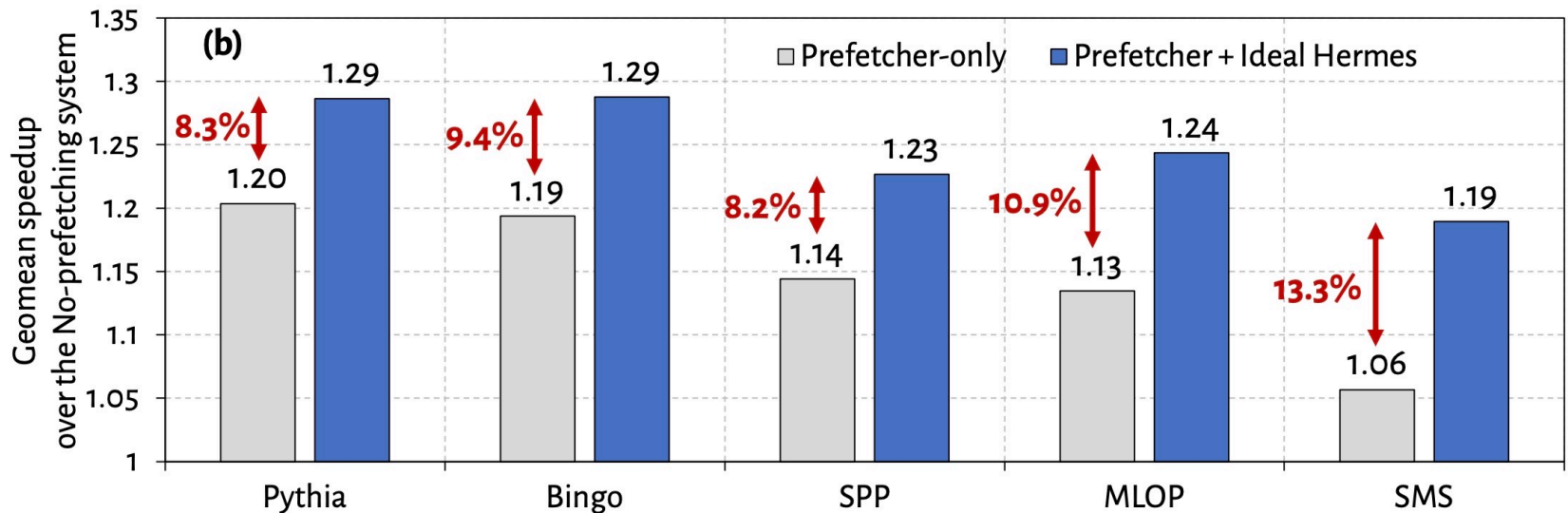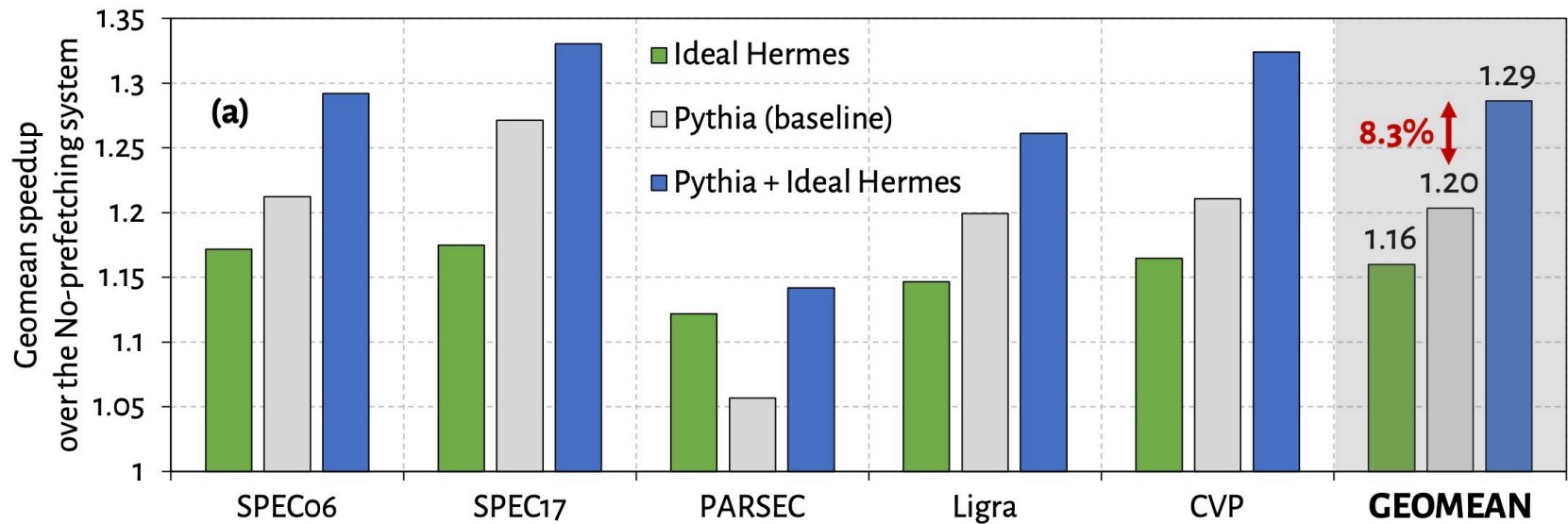
System Parameters

Table 4: Simulated system parameters

|	
Core	1 and 8 cores, 6-wide fetch/execute/commit, 512-entry ROB, 128/72-entry LQ/SQ, Perceptron branch predictor [61] with 17-cycle misprediction penalty
L1/L2 Caches	Private, 48KB/1.25MB, 64B line, 12/20-way, 16/48 MSHRs, LRU, 5/15-cycle round-trip latency [25]
LLC	3MB/core, 64B line, 12 way, 64 MSHRs/slice, SHiP [122], 55-cycle round-trip latency [24, 25], **Pythia** prefetcher [32]
Main Memory	**1C:** 1 channel, 1 rank per channel; **8C:** 4 channels, 2 ranks per channel; 8 banks per rank, DDR4-3200 MTPS, 64b data-bus per channel, 2KB row buffer per bank, tRCD=12.5ns, tRP=12.5ns, tCAS=12.5ns
Hermes	**Hermes-O/P**: 6/18-cycle Hermes request issue latency

SAFARI

Evaluated Workloads

Table 5: Workloads used for evaluation

Suite	#Workloads	#Traces	Example Workloads
SPEC06	14	22	gcc, mcf, cactusADM, lbm, ...
SPEC17	11	23	gcc, mcf, pop2, fotonik3d, ...
PARSEC	4	12	canneal, facesim, raytrace, ...
Ligra	11	20	BFS, PageRank, Radii, ...
CVP	33	33	integer, floating-point, server, ...

Observation: Not All Off-Chip Loads are Prefetched

Observation: Not All Off-Chip Loads are Prefetched

SAFARI
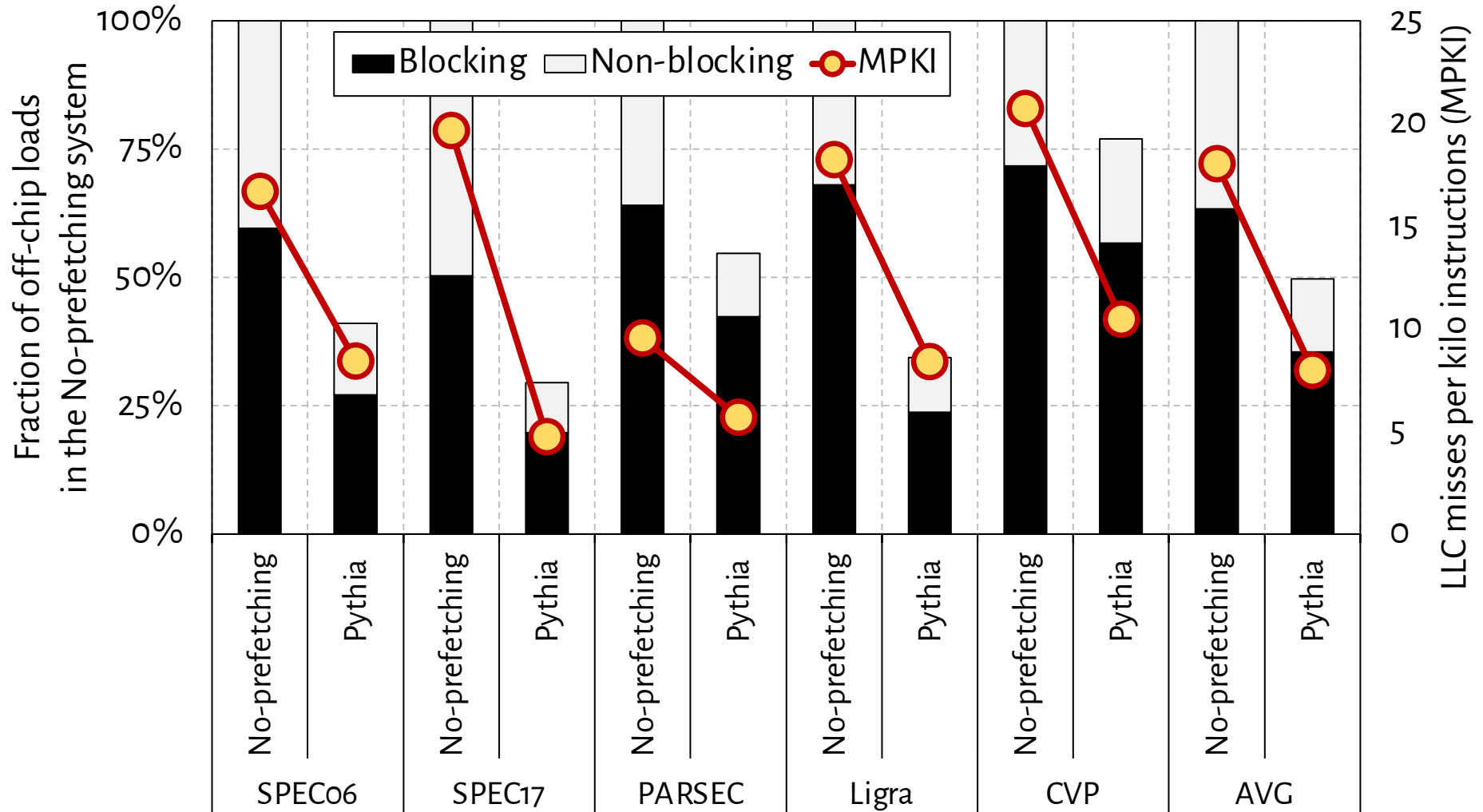
Observation: Not All Off-Chip Loads are Prefetched
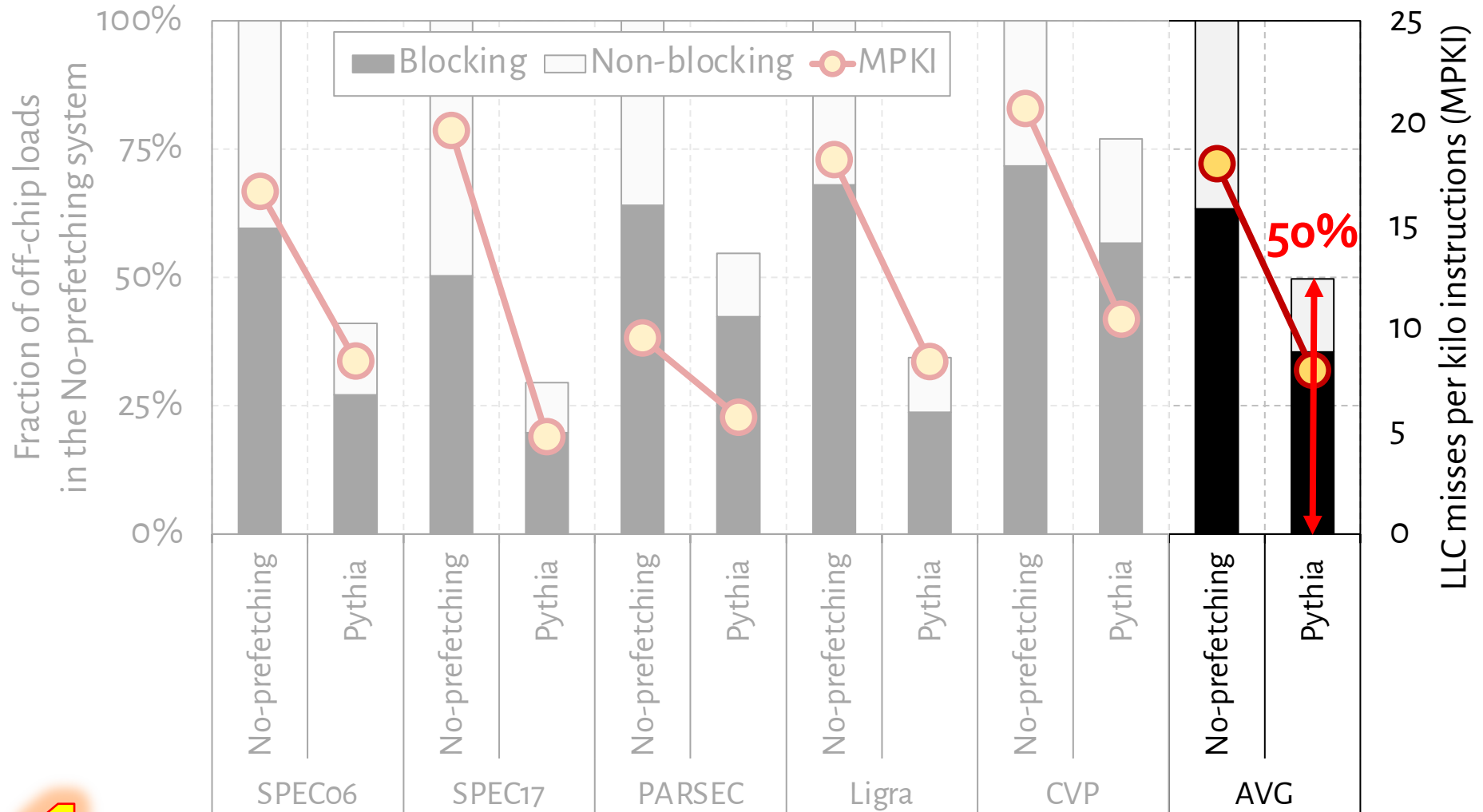
1 Nearly **50%** of the loads are still **not prefetched**

Observation: Not All Off-Chip Loads are Prefetched
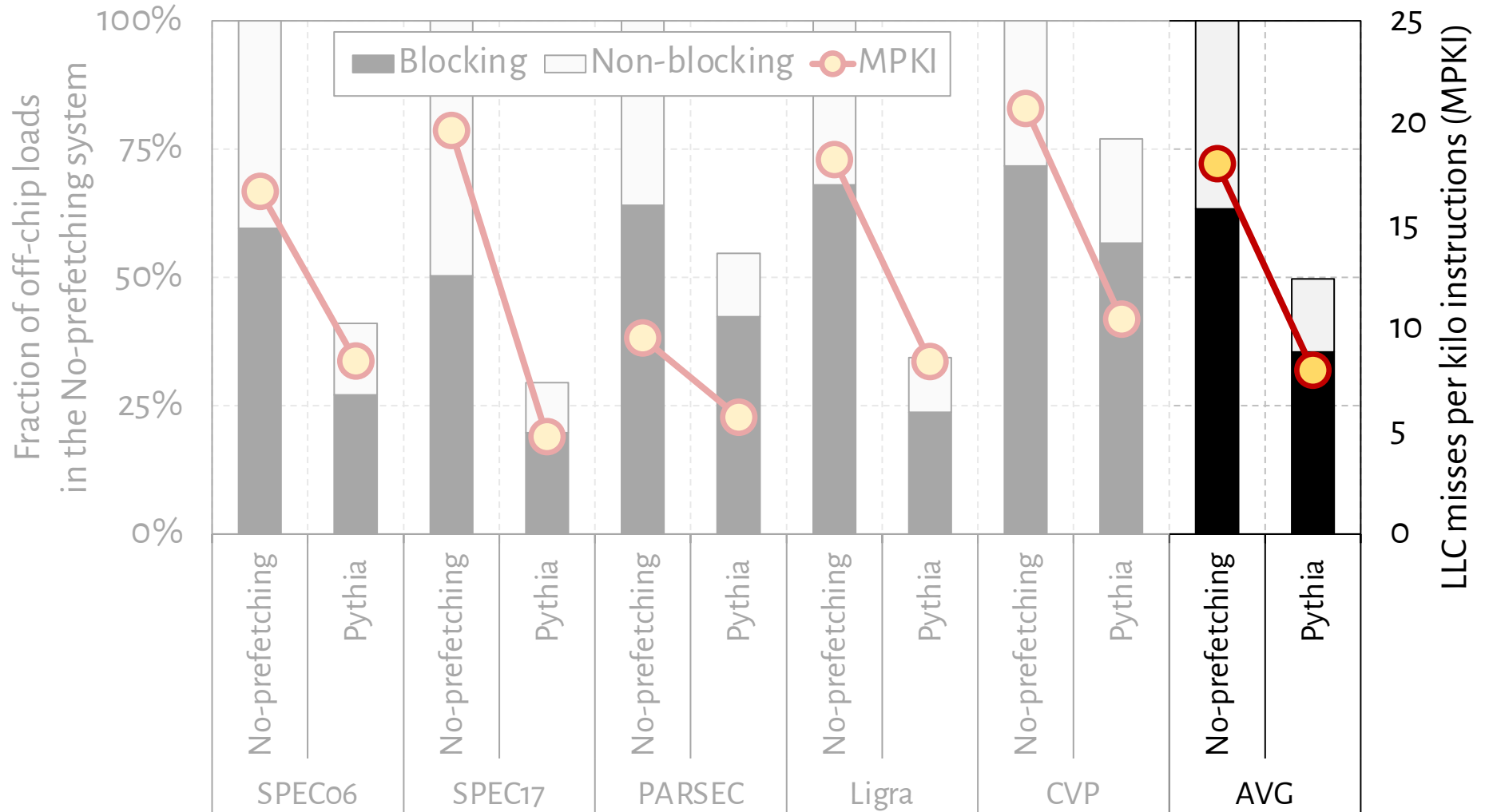
Observation: **Not All Off-Chip Loads are Prefetched**
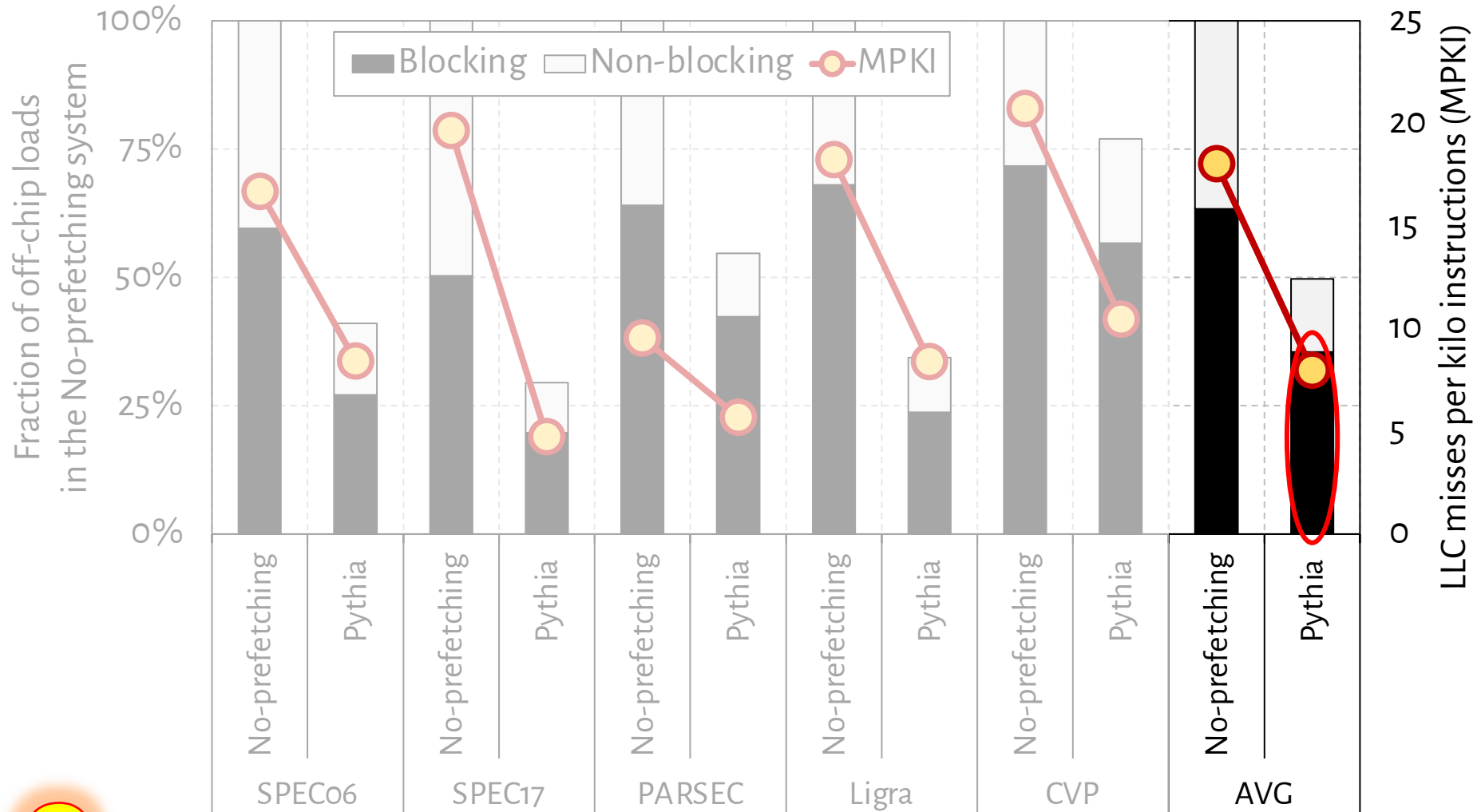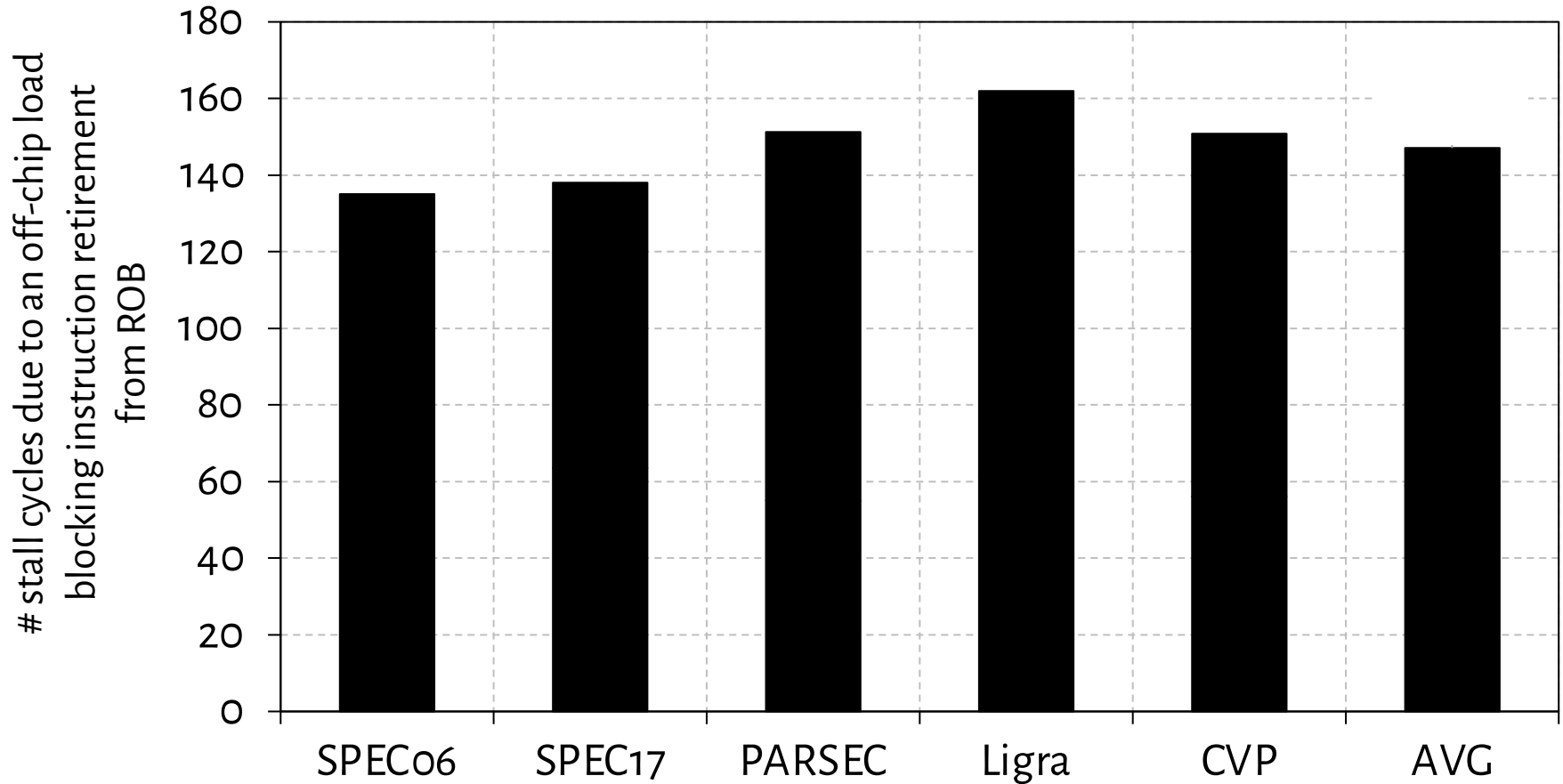

2

70% of these off-chip loads **blocks ROB**

Observation: With Large Cache Comes Longer Latency

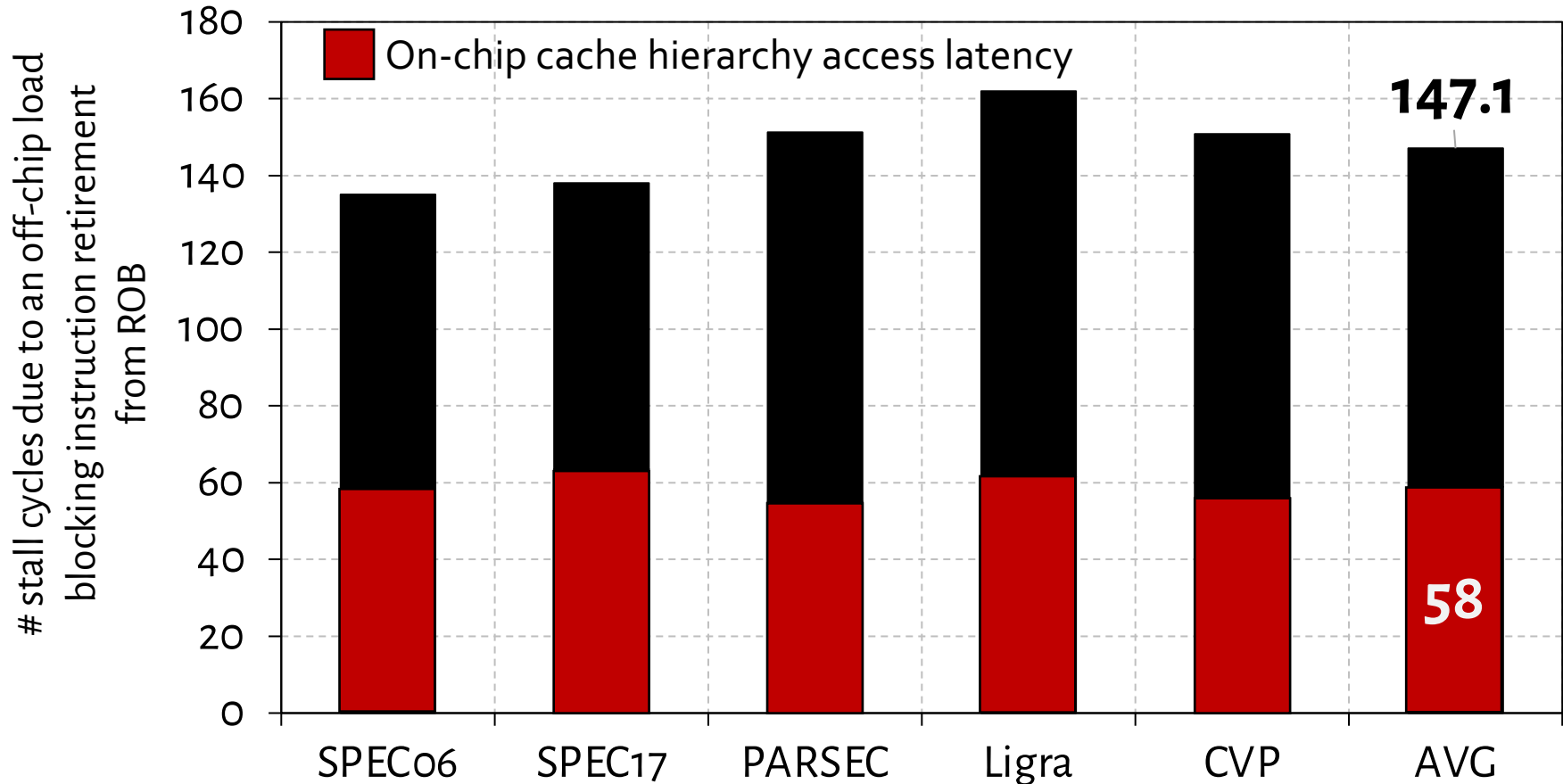- On-chip cache access latency significantly contributes to the latency of an off-chip load
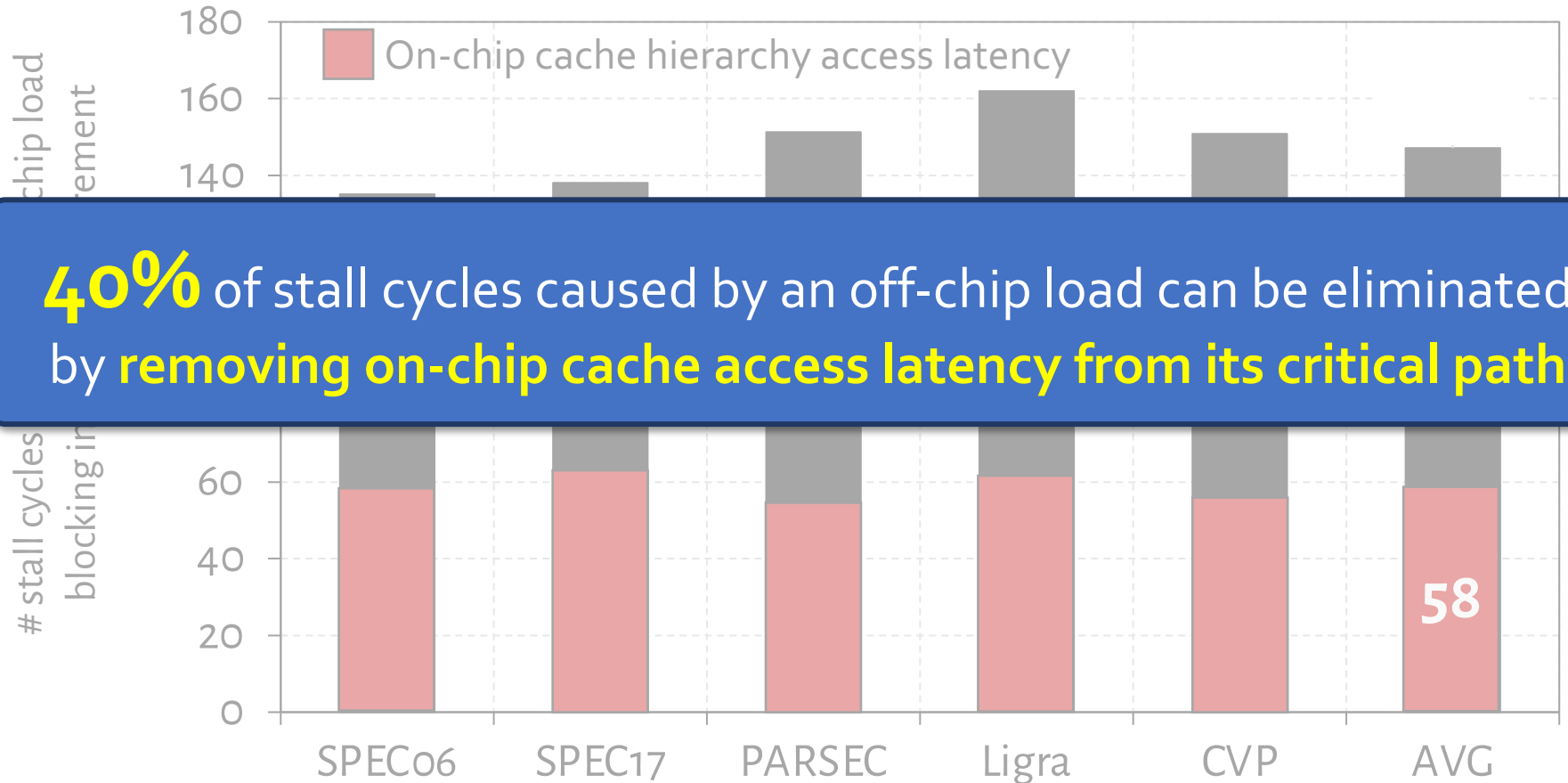
SAFARI

Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load

Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load
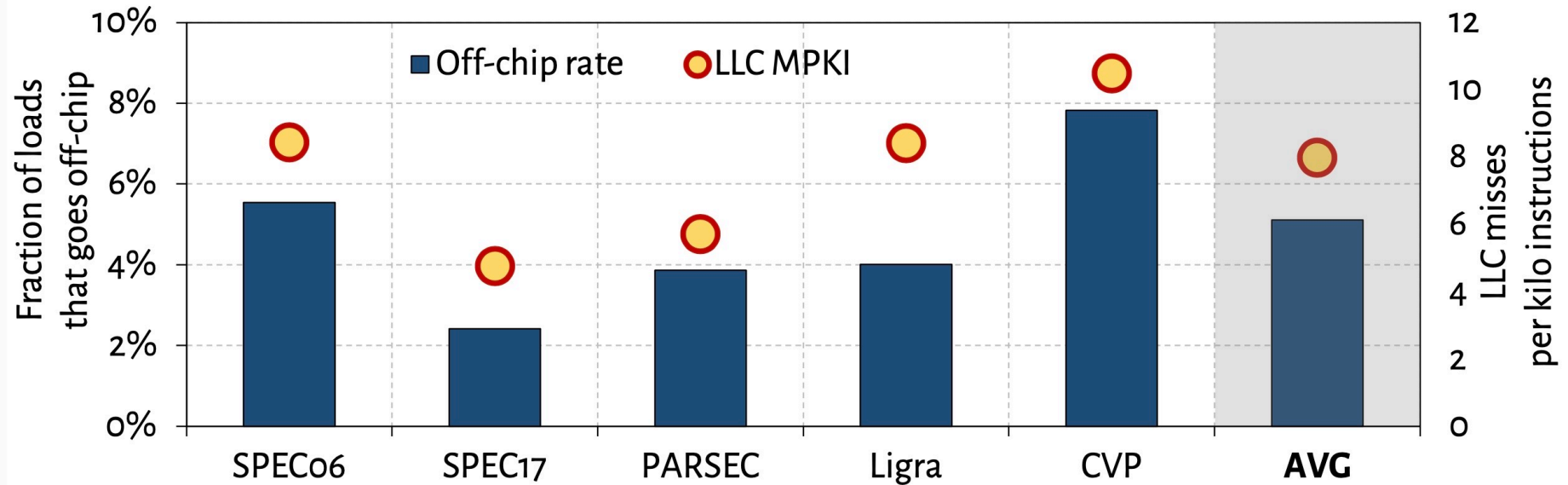
SAFARI

Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load

Observation: With Large Cache Comes Longer Latency

- On-chip cache access latency significantly contributes to the latency of an off-chip load
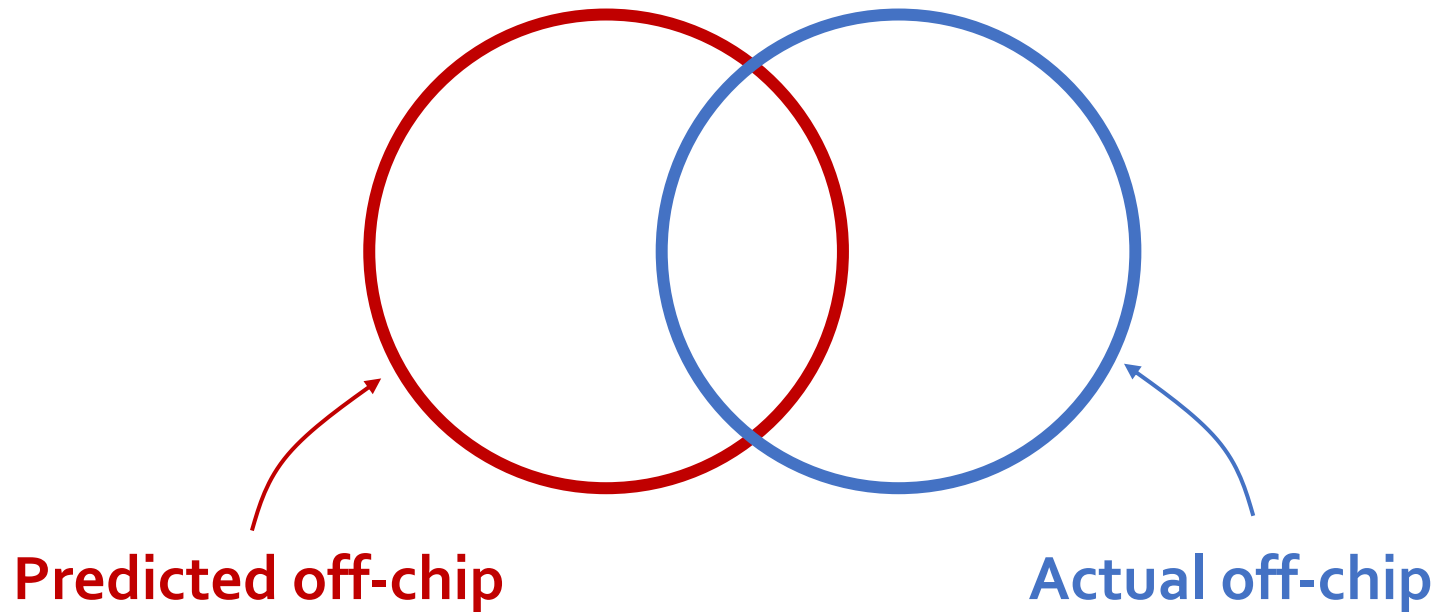

40% of stall cycles caused by an off-chip load can be eliminated by **removing on-chip cache access latency from its critical path**

What Fraction of Load Requests Goes Off-Chip?

SAFARI

Off-Chip Prediction Quality: *Defining Metrics*

Predicted off-chip

Actual off-chip

SAFARI

Off-Chip Prediction Quality: *Defining Metrics*

Predicted off-chip

Actual off-chip

Predicted *and* actual off-chip

Off-Chip Prediction Quality: *Defining Metrics*

Accuracy

Predicted off-chip

Actual off-chip

Predicted *and* actual off-chip

Off-Chip Prediction Quality: *Defining Metrics*

Accuracy

Coverage

Predicted off-chip

Actual off-chip

Predicted *and* actual off-chip

SAFARI

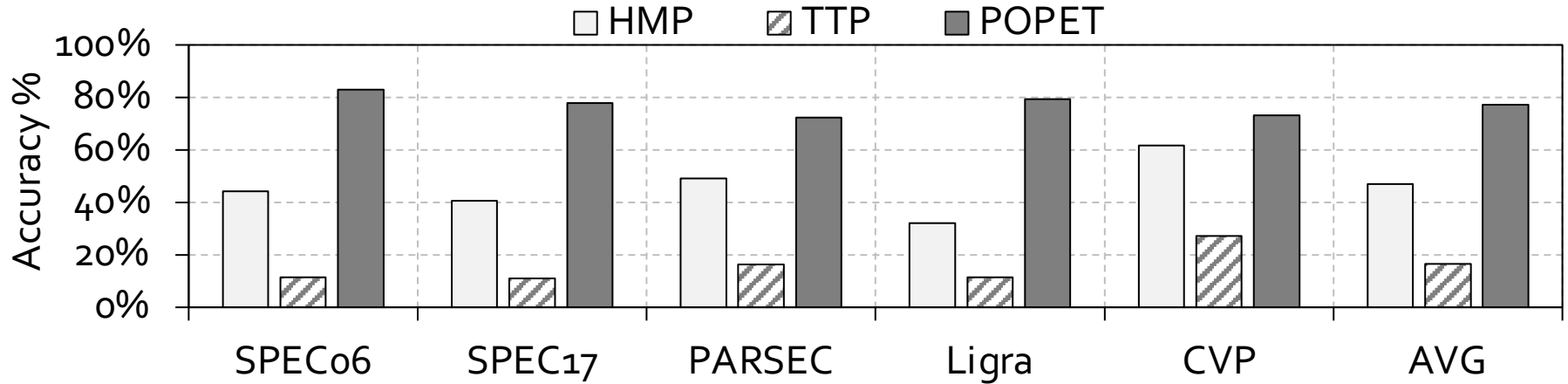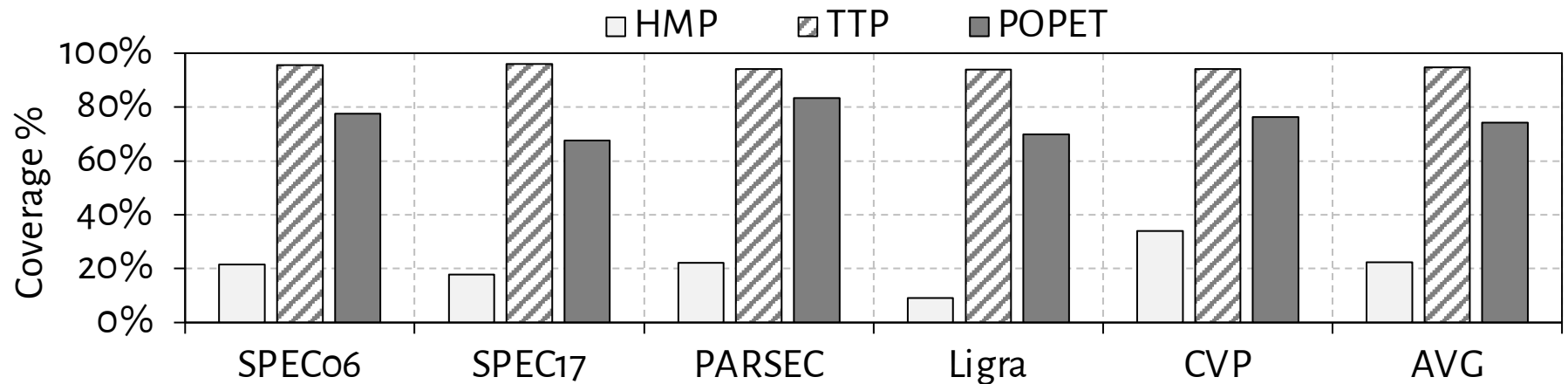# Off-Chip Prediction Quality: *Analysis*
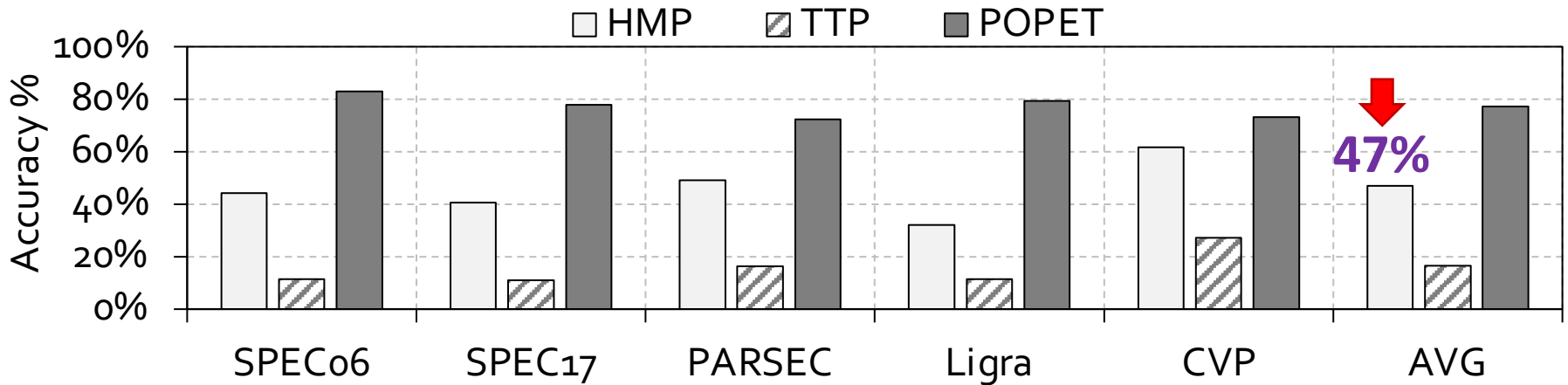
Accuracy

Coverage

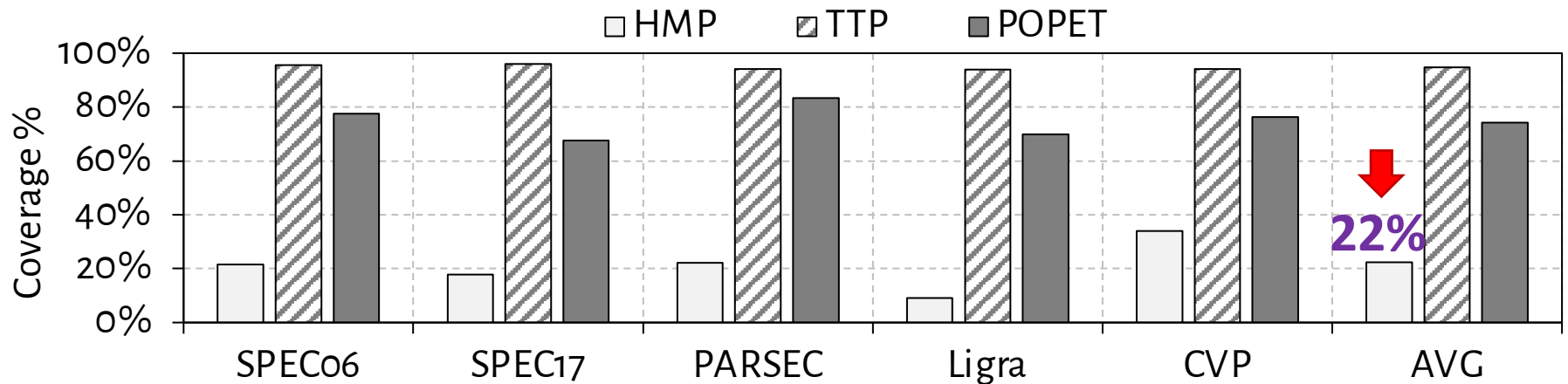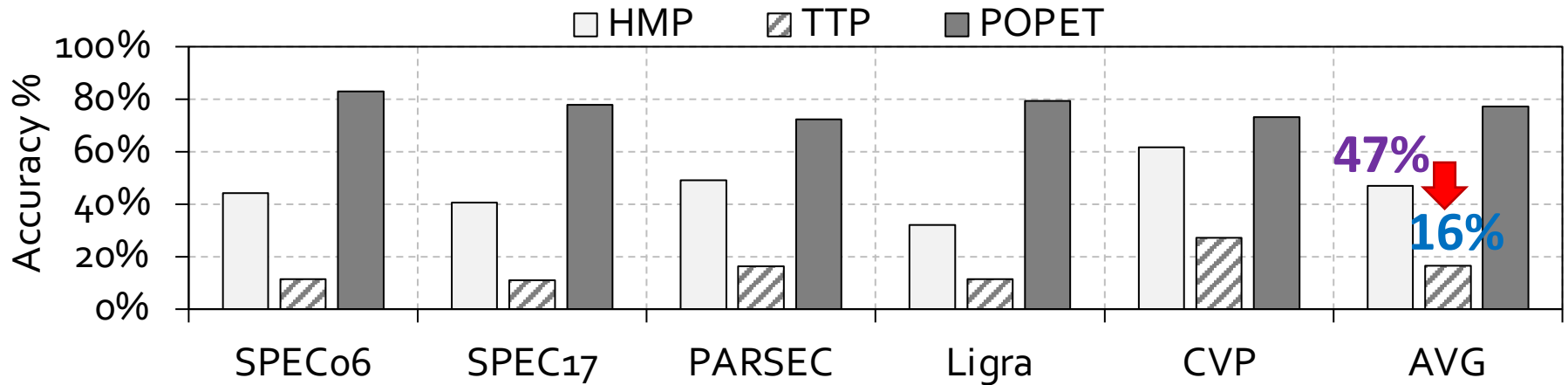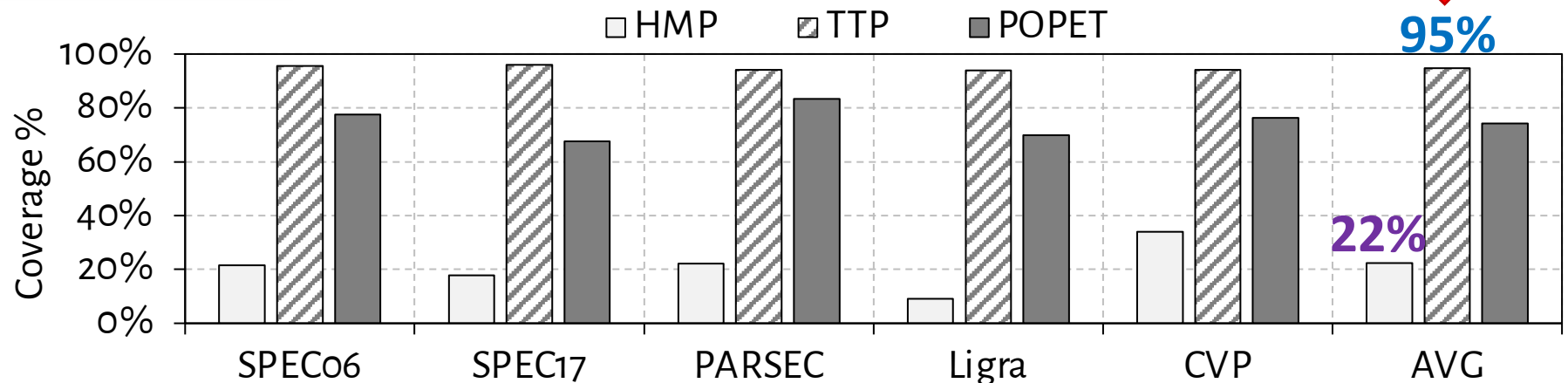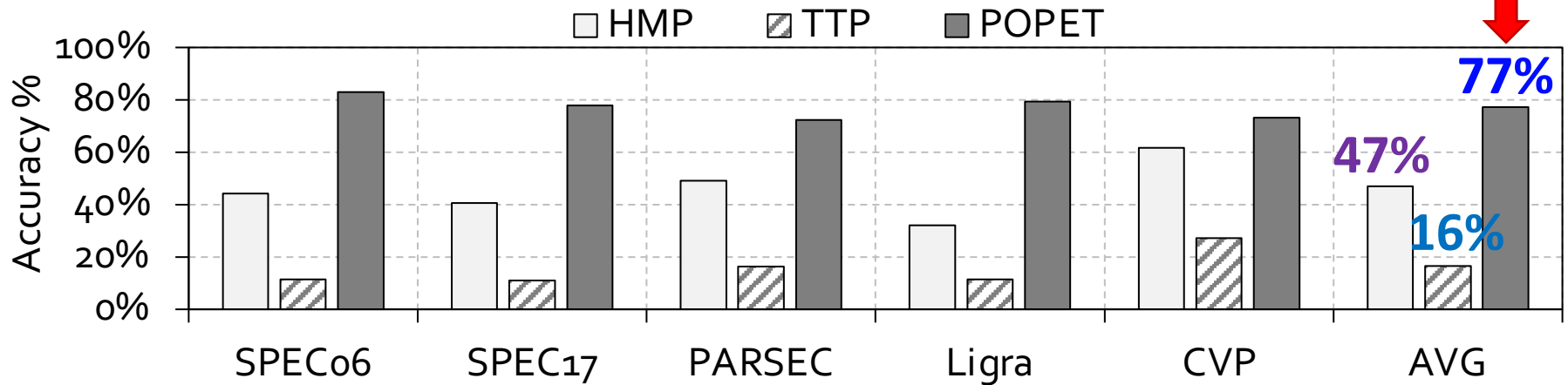# Off-Chip Prediction Quality: *Analysis*

Accuracy

Coverage

Off-Chip Prediction Quality: *Analysis*

Accuracy

Accuracy % — HMP, TTP, POPET across SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG. AVG: 47%

Coverage

Coverage % — HMP, TTP, POPET across SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG. AVG: 22%

SAFARI

Off-Chip Prediction Quality: *Analysis*

Accuracy

Legend: ☐ HMP ▨ TTP ■ POPET

Accuracy % (y-axis: 0% to 100%)

Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG

47% **16%**

Coverage

Legend: ☐ HMP ▨ TTP ■ POPET

Coverage % (y-axis: 0% to 100%)

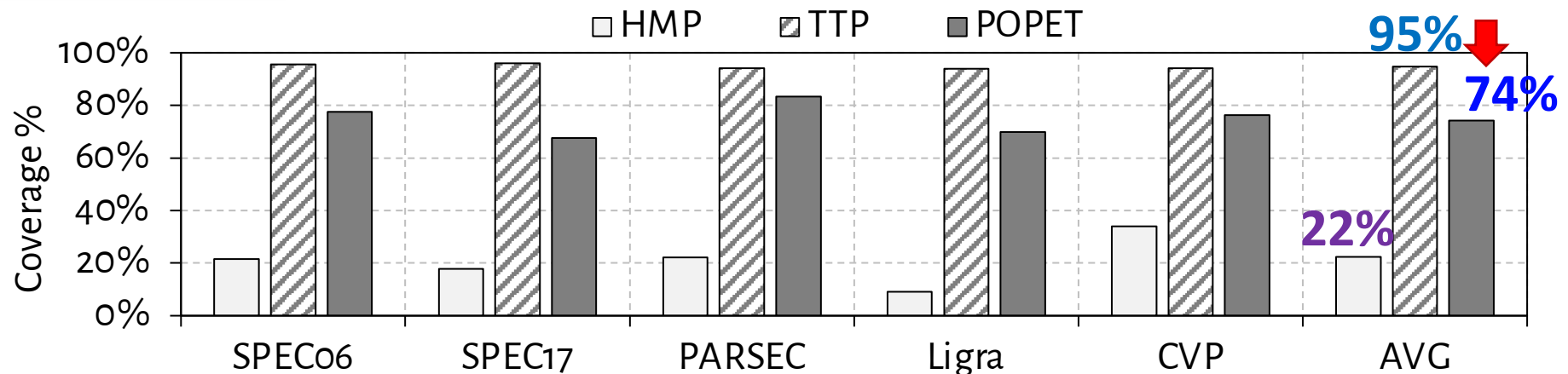Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, AVG

95% **22%**
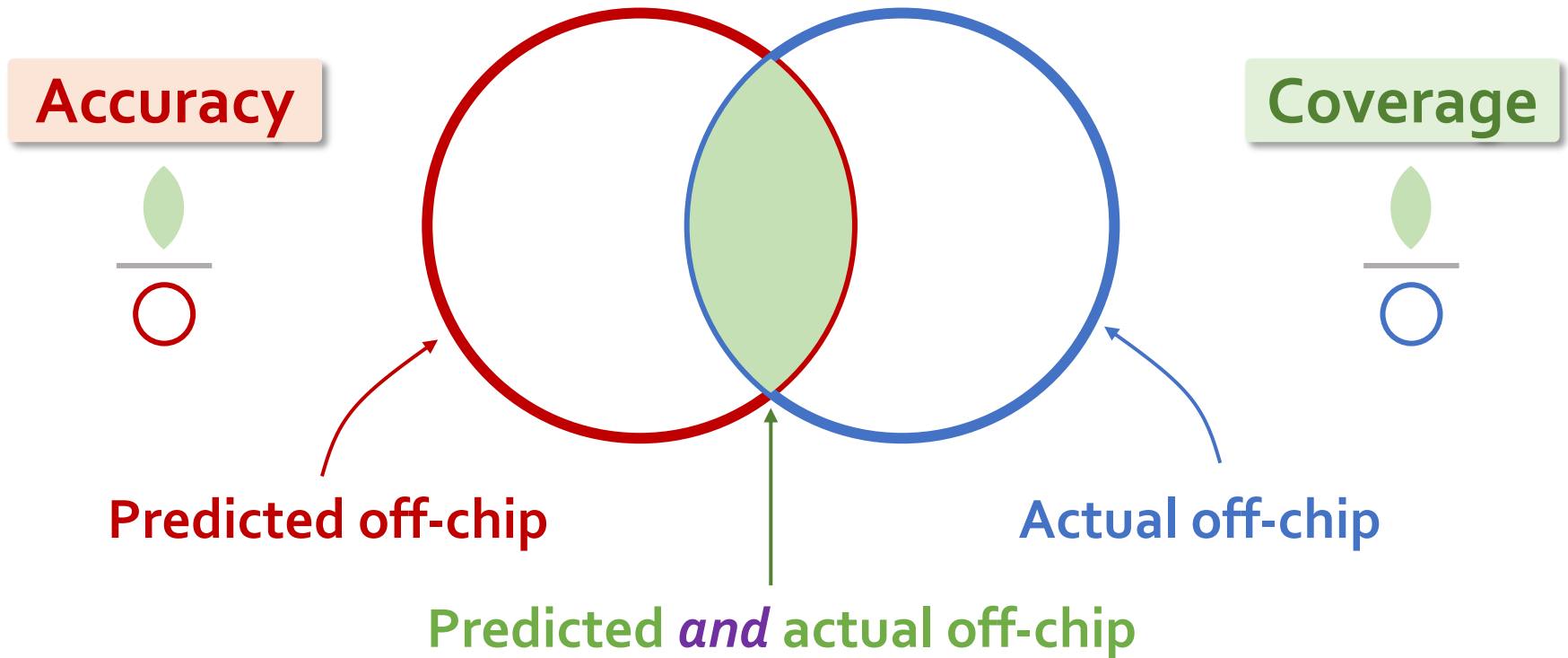
Off-Chip Prediction Quality: *Analysis*
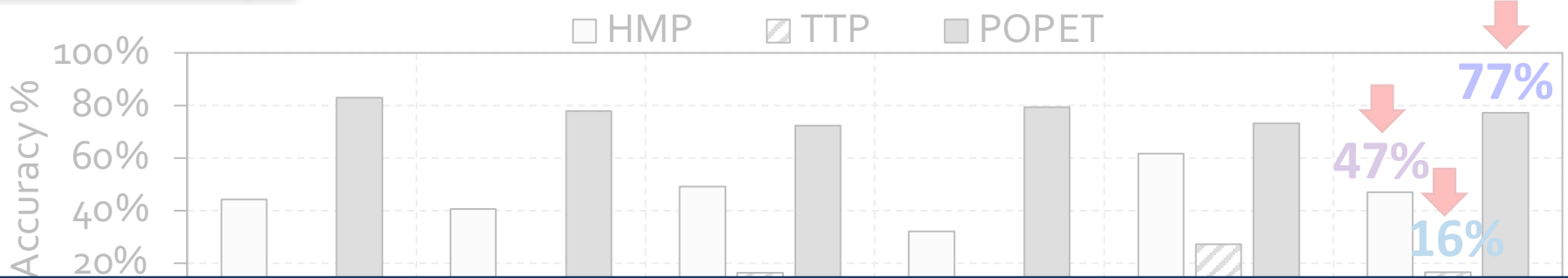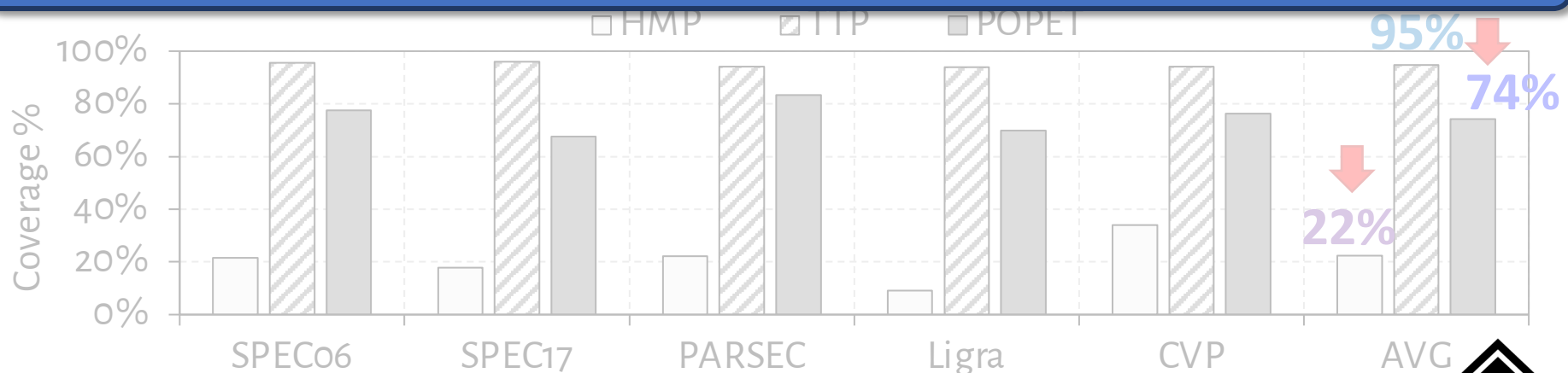
Accuracy

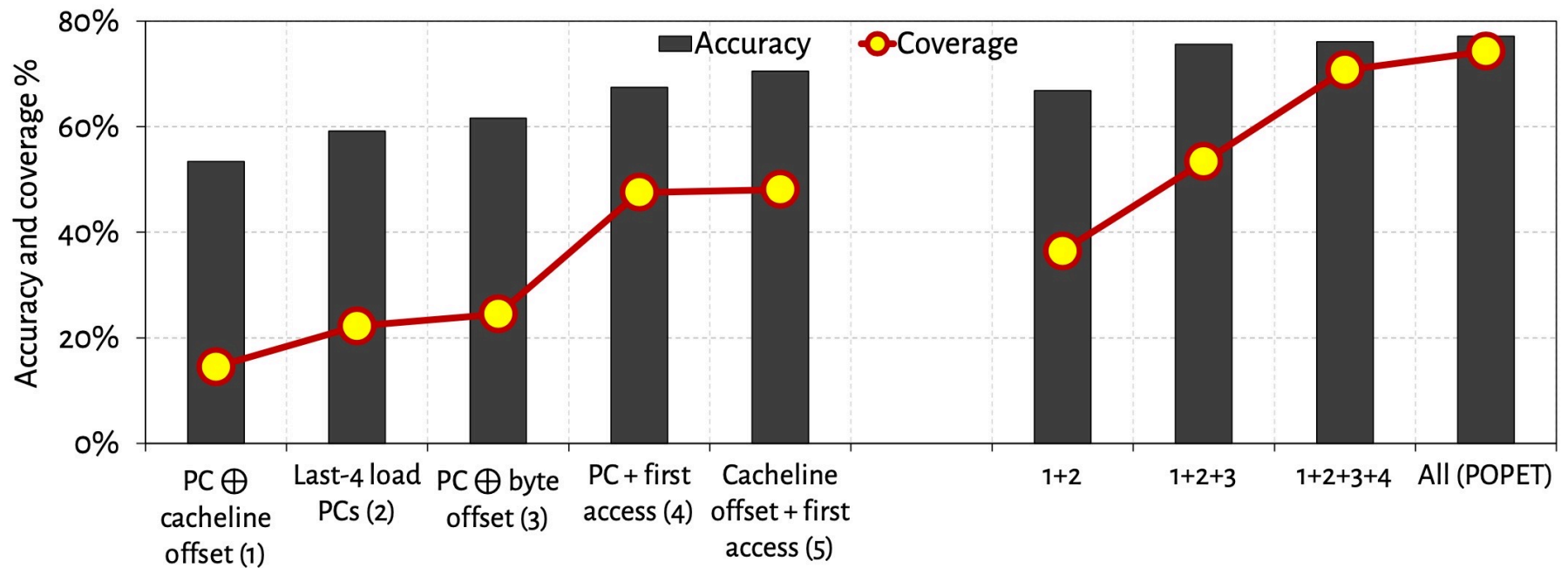**Coverage**

Off-Chip Prediction Quality: *Analysis*
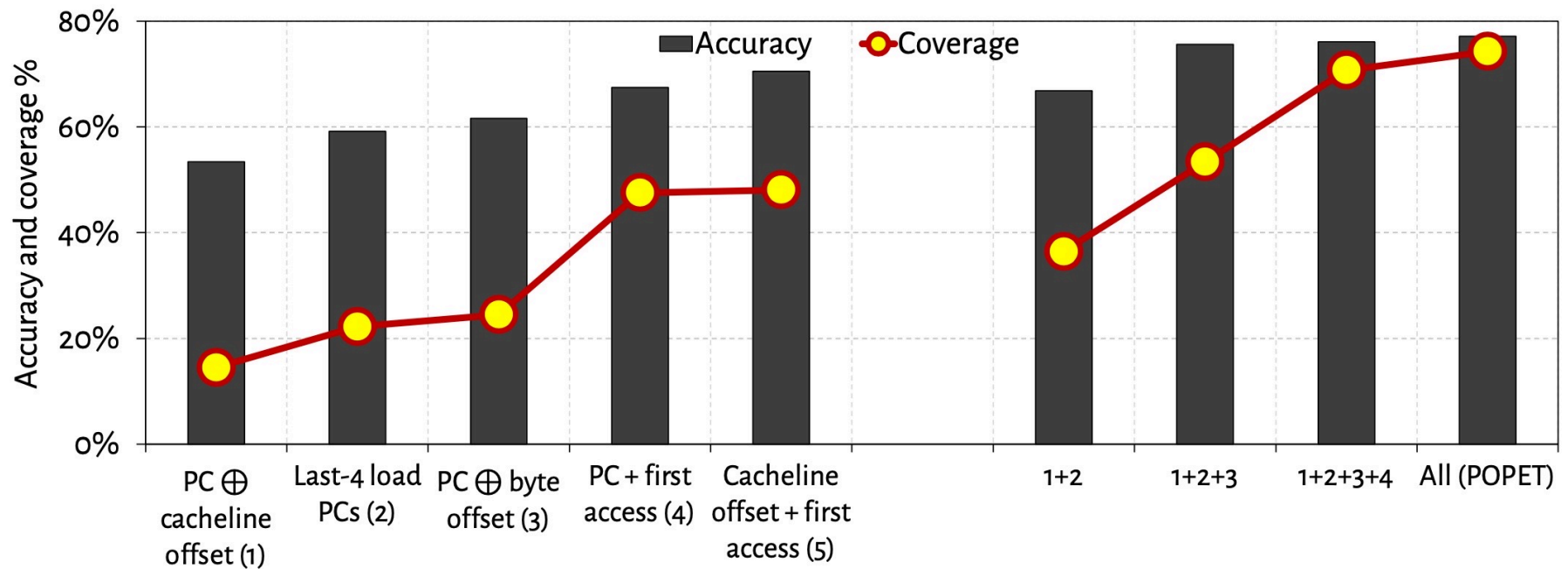
Accuracy

POPET provides off-chip predictions with **high-accuracy** and **high-coverage**
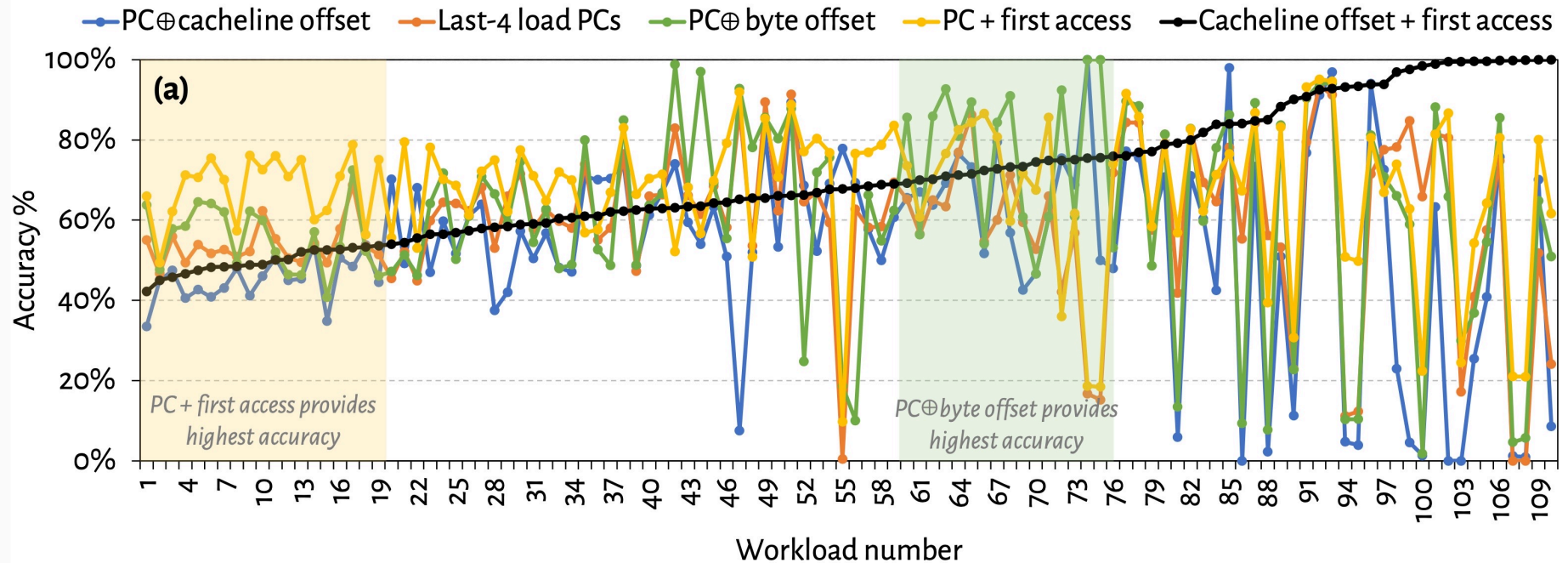
SAFARI

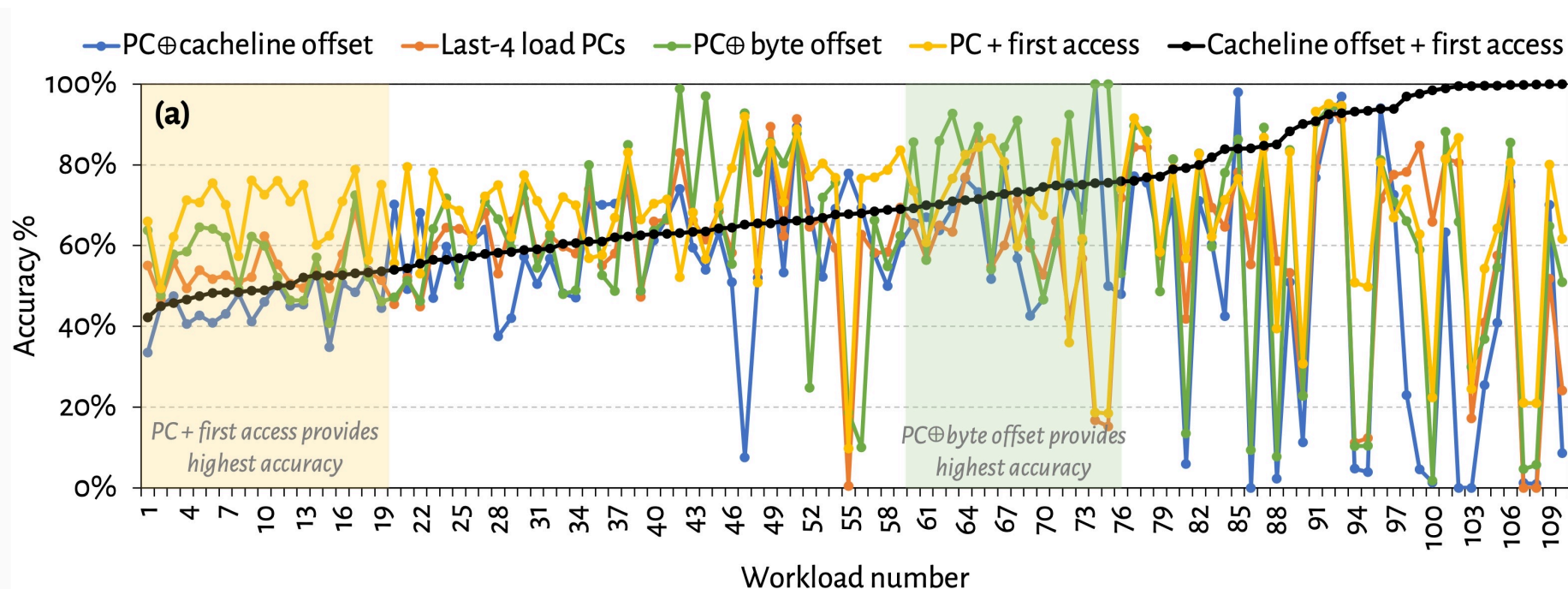# Effect of Different Features

Effect of Different Features

Combination of features provides both **higher accuracy and higher coverage** than any individual feature

SAFARI

Are All Features Required? (1)

SAFARI

Are All Features Required? (1)

(a)

Legend: PC⊕cacheline offset, Last-4 load PCs, PC⊕ byte offset, PC + first access, Cacheline offset + first access

PC + first access provides highest accuracy

PC⊕byte offset provides highest accuracy
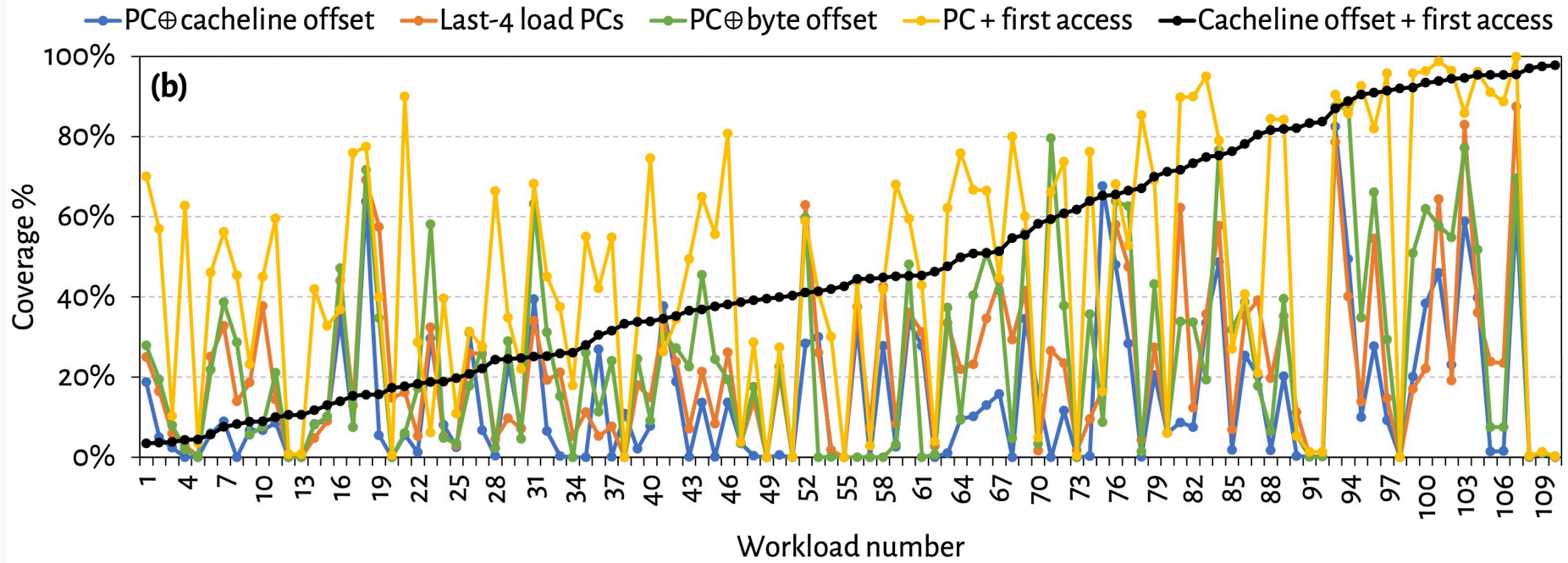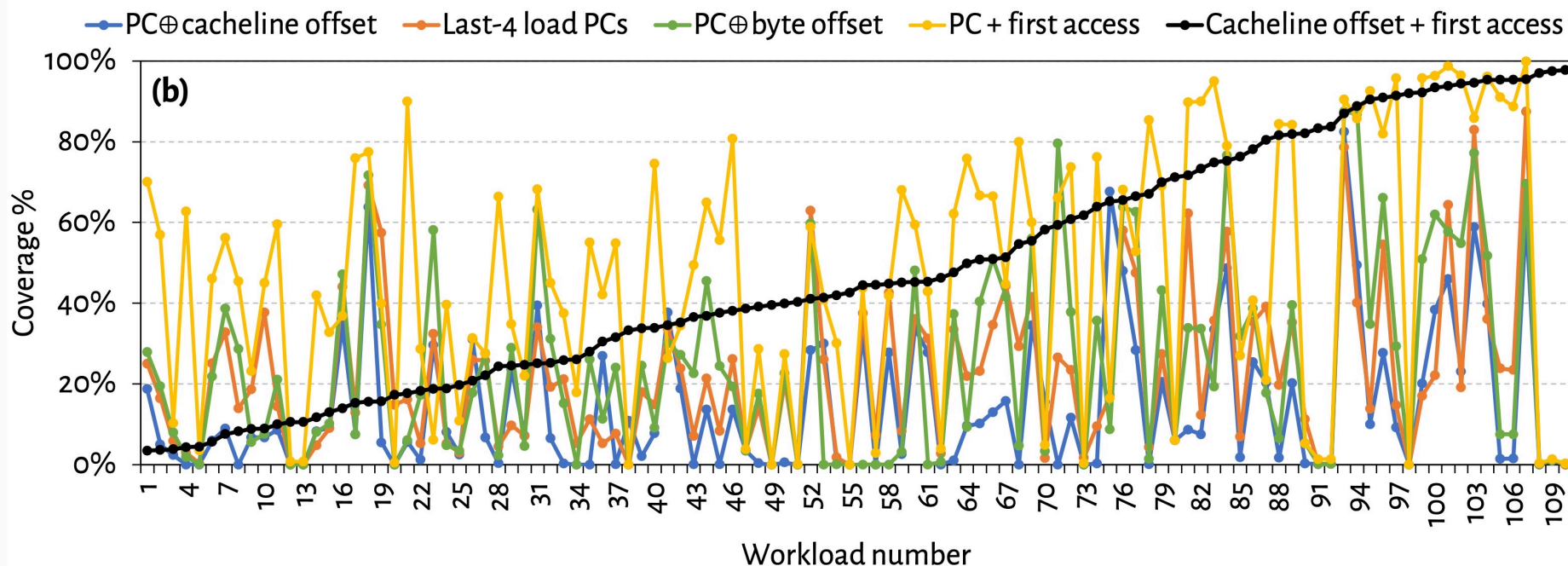
No single feature individually provides **highest prediction accuracy** across *all* workloads

SAFARI

Are All Features Required? (2)

SAFARI

Are All Features Required? (2)

Chart (b): Coverage % versus Workload number for five features: PC⊕cacheline offset, Last-4 load PCs, PC⊕byte offset, PC + first access, Cacheline offset + first access.
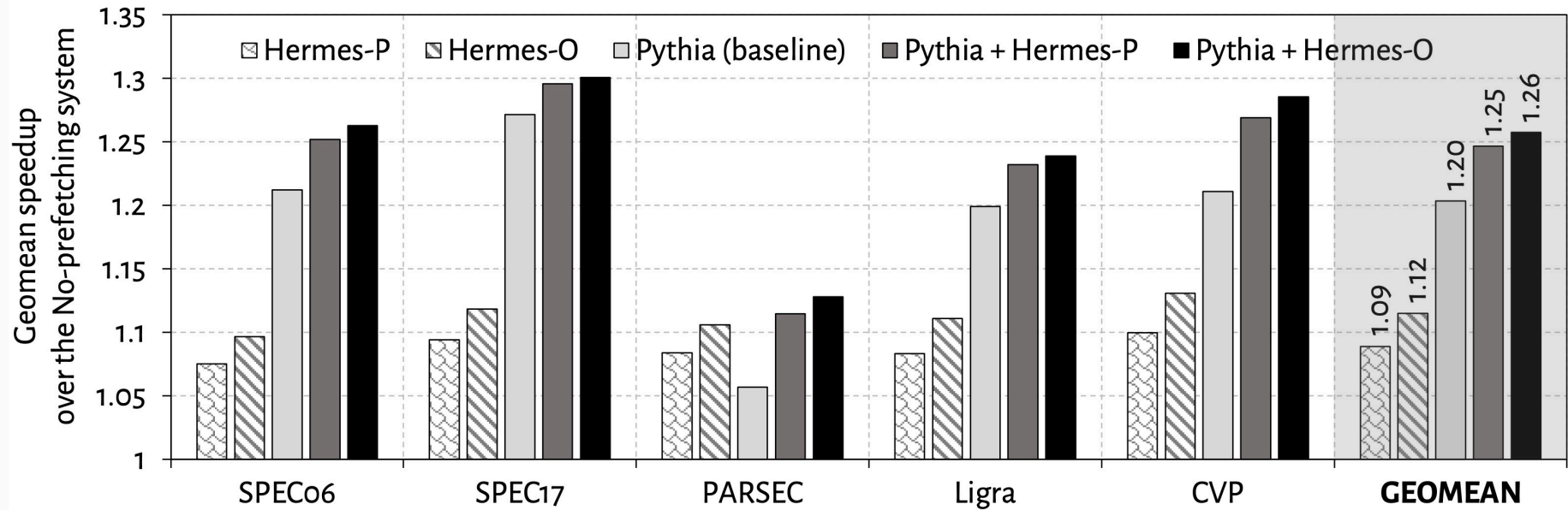
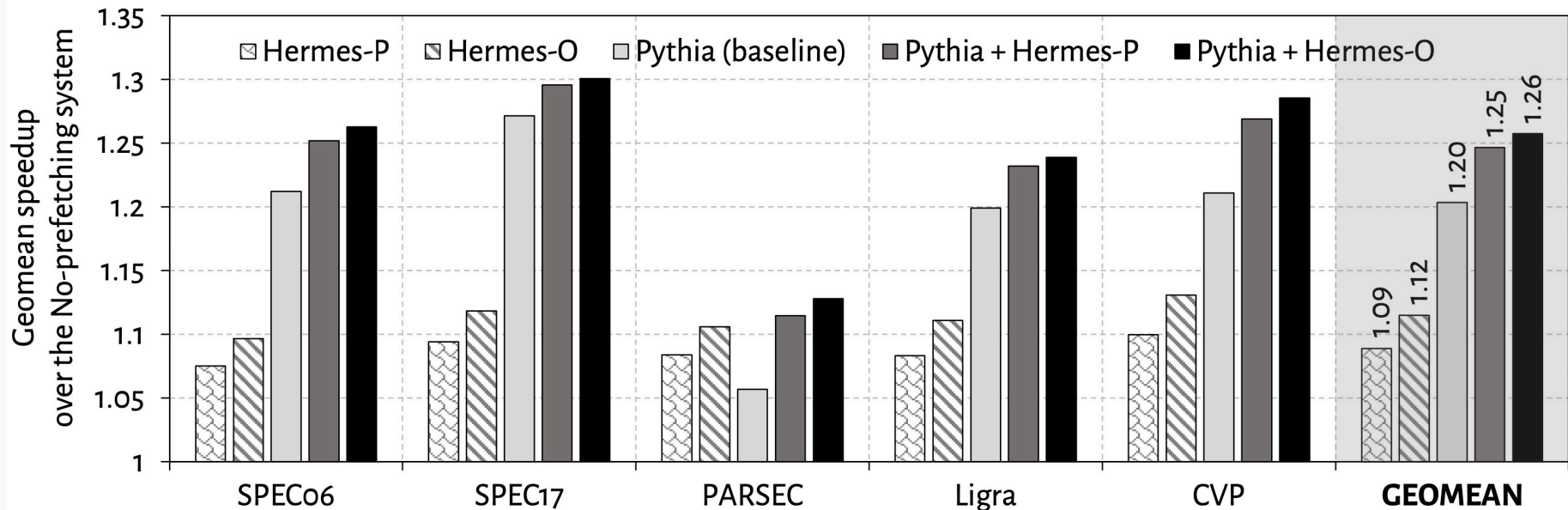**No single feature** individually provides
highest prediction coverage also across *all* workloads

Single-Core Performance

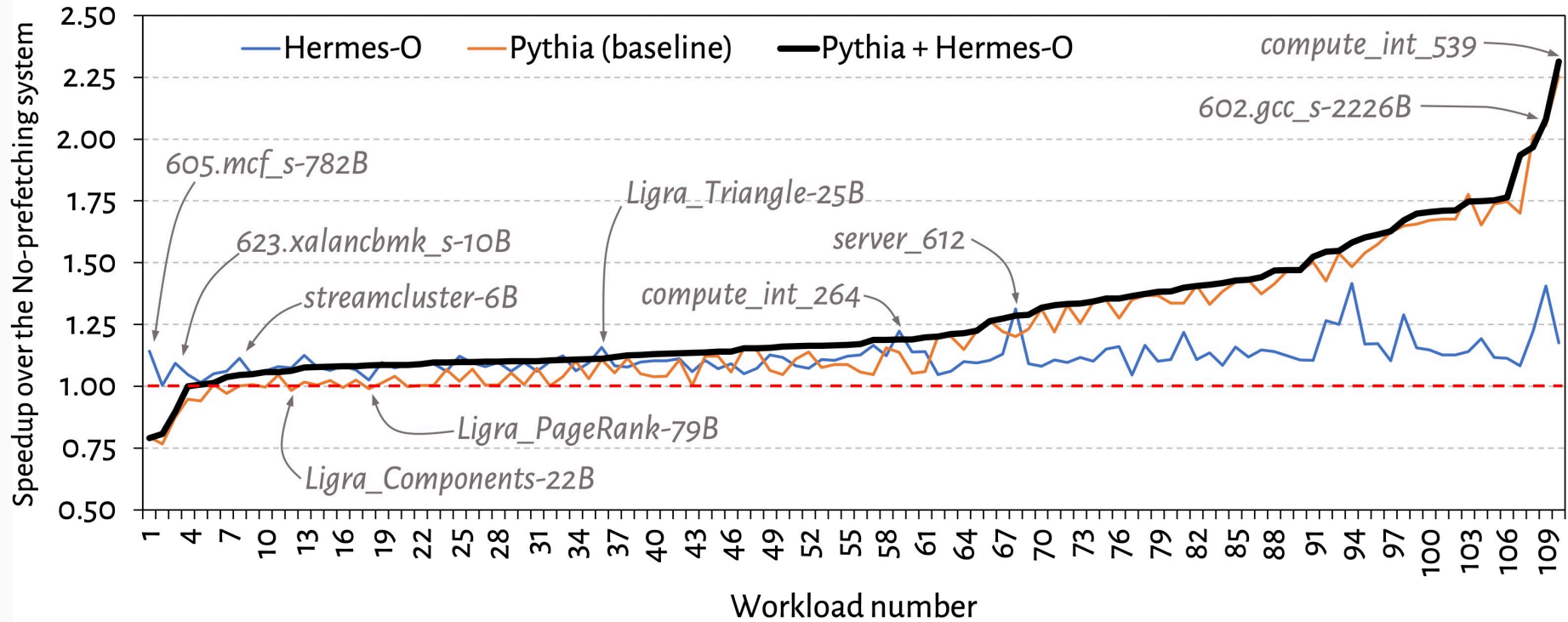Chart: Geomean speedup over the No-prefetching system across benchmarks (SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN) for five configurations: Hermes-P, Hermes-O, Pythia (baseline), Pythia + Hermes-P, Pythia + Hermes-O.

GEOMEAN values: 1.09, 1.12, 1.20, 1.25, 1.26

SAFARI

Single-Core Performance

Geomean speedup over the No-prefetching system

Legend: Hermes-P, Hermes-O, Pythia (baseline), Pythia + Hermes-P, Pythia + Hermes-O

Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

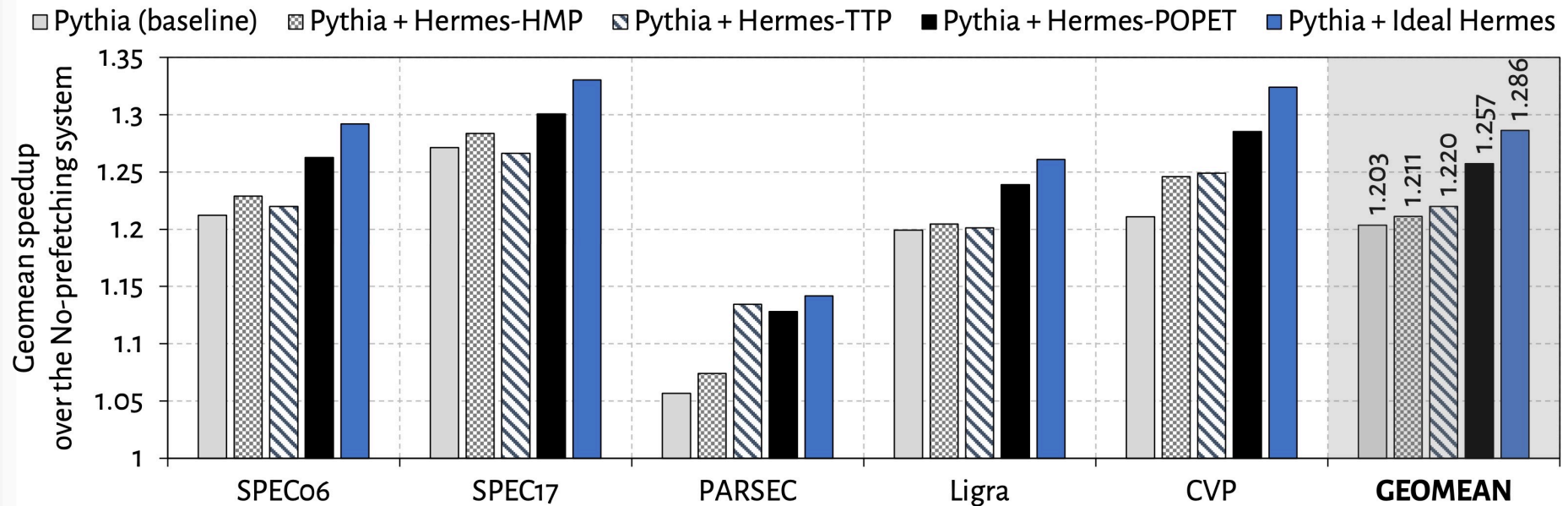GEOMEAN values: 1.09, 1.12, 1.20, 1.25, 1.26

Hermes in combination with Pythia outperforms **Pythia alone** in every workload category

SAFARI

Single-Core Performance Line Graph

Single-Core Performance Against Prior Predictors

Legend: Pythia (baseline), Pythia + Hermes-HMP, Pythia + Hermes-TTP, Pythia + Hermes-POPET, Pythia + Ideal Hermes

Y-axis: Geomean speedup over the No-prefetching system

GEOMEAN values: 1.203, 1.211, 1.220, 1.257, 1.286

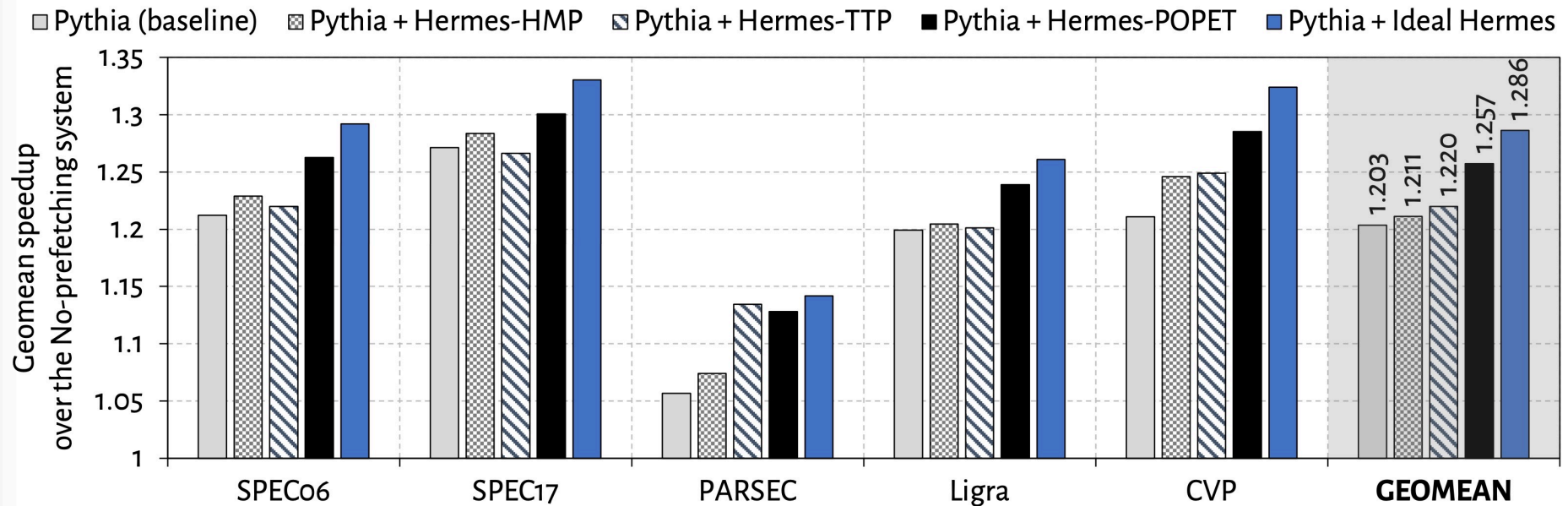Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

Single-Core Performance Against Prior Predictors

POPET provides higher performance benefit than prior predictors

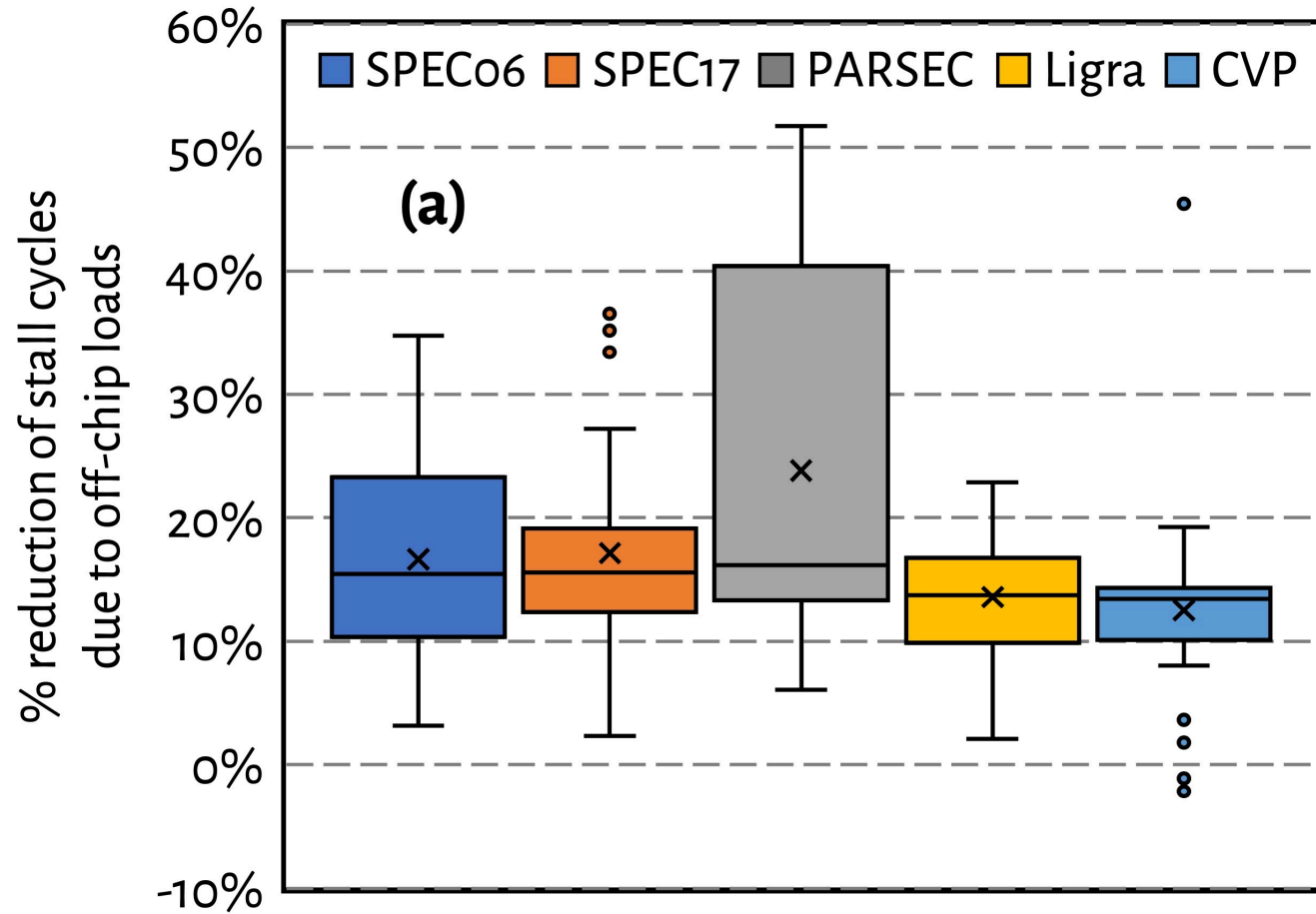# Single-Core Performance Against Prior Predictors

Legend: Pythia (baseline), Pythia + Hermes-HMP, Pythia + Hermes-TTP, Pythia + Hermes-POPET, Pythia + Ideal Hermes

Y-axis: Geomean speedup over the No-prefetching system

GEOMEAN values: 1.203, 1.211, 1.220, 1.257, 1.286

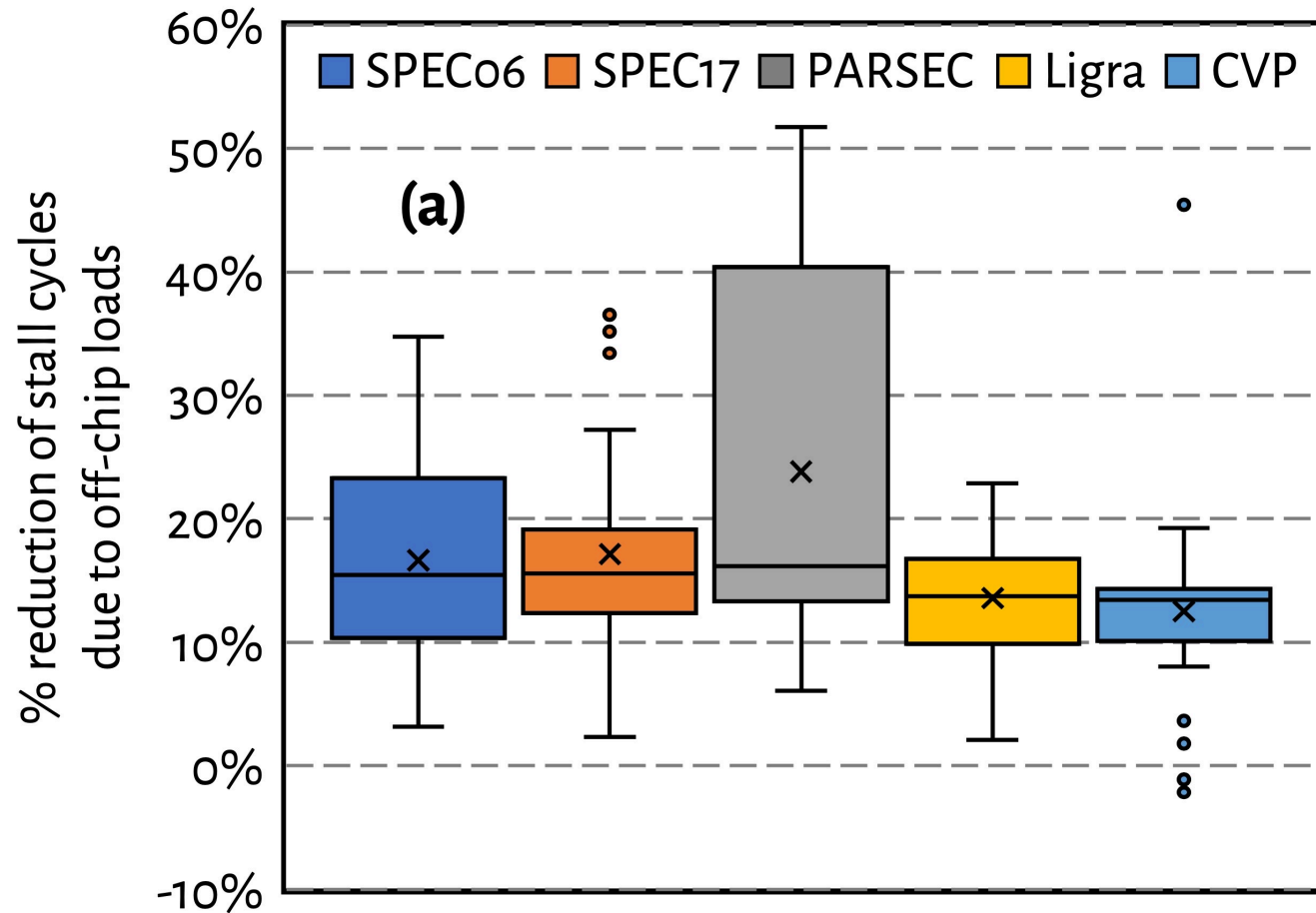Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, GEOMEAN

POPET provides higher performance benefit than prior predictors

Hermes with POPET achieves nearly **90%** performance improvement of the Ideal Hermes
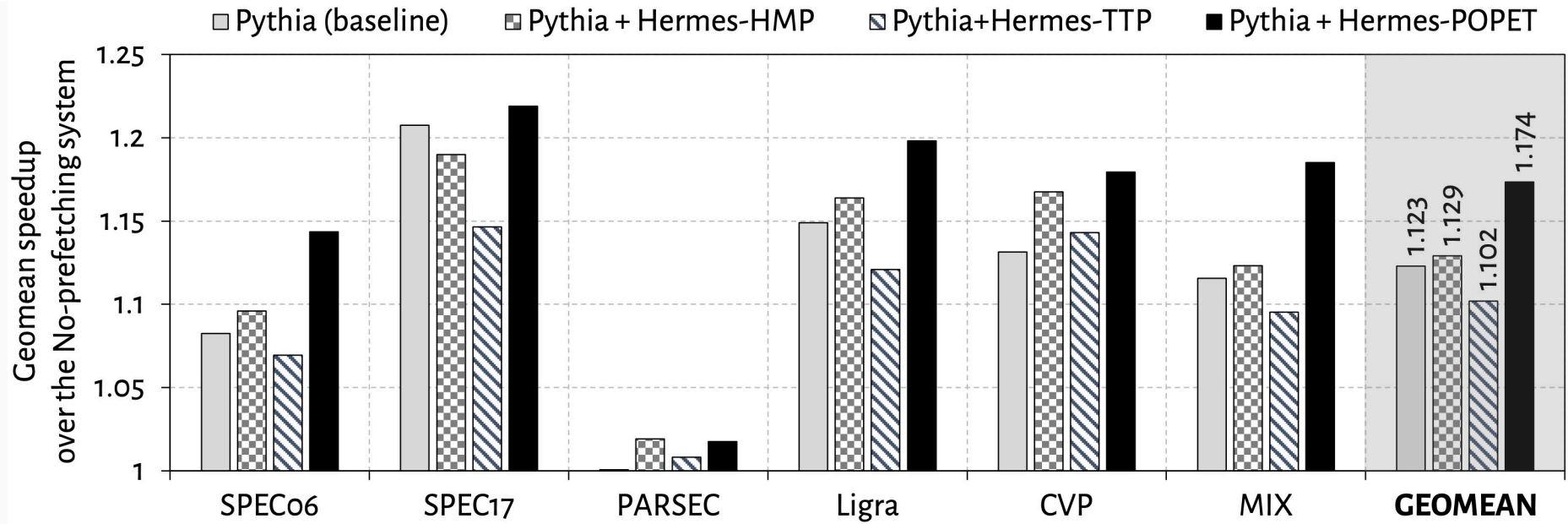
Effect on Stall Cycles
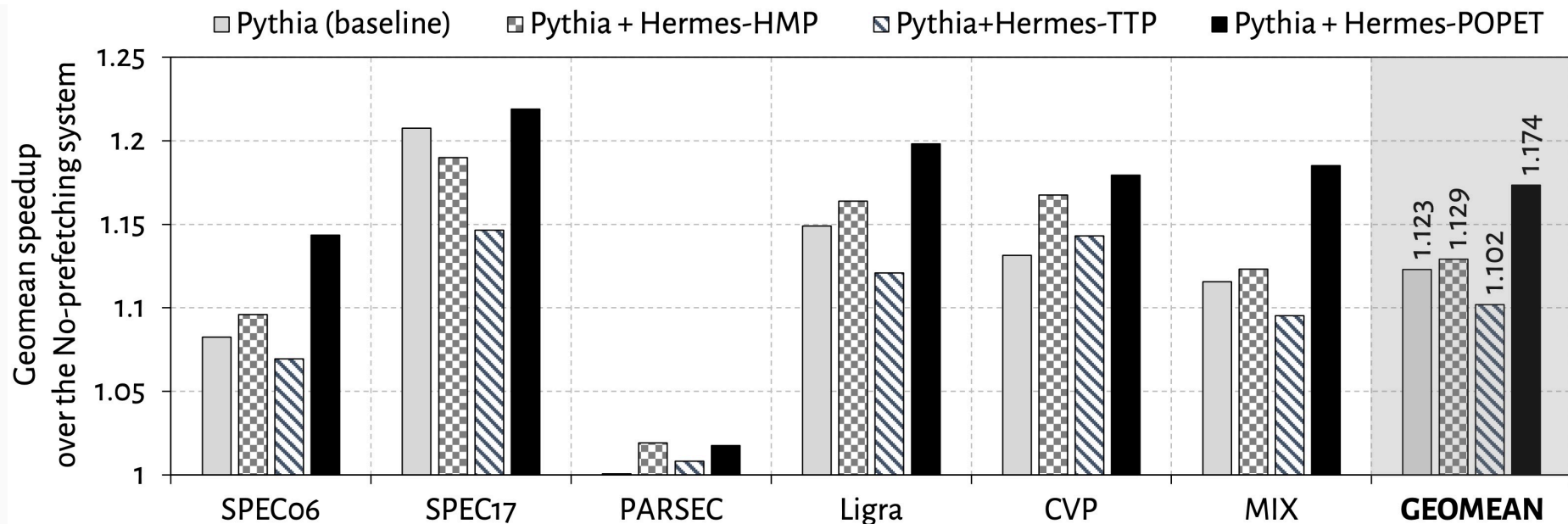
Effect on Stall Cycles

(a)

% reduction of stall cycles due to off-chip loads

Legend: SPEC06, SPEC17, PARSEC, Ligra, CVP

Hermes reduces off-chip load induced stall cycles on average by **16.2%** (up-to **51.8%**)

Eight-Core Performance

Legend: Pythia (baseline), Pythia + Hermes-HMP, Pythia+Hermes-TTP, Pythia + Hermes-POPET

Y-axis: Geomean speedup over the No-prefetching system

Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, MIX, GEOMEAN

GEOMEAN values: 1.123, 1.129, 1.102, 1.174

SAFARI

Eight-Core Performance

Legend: Pythia (baseline), Pythia + Hermes-HMP, Pythia+Hermes-TTP, Pythia + Hermes-POPET

Y-axis: Geomean speedup over the No-prefetching system

GEOMEAN values: 1.123, 1.129, 1.102, 1.174
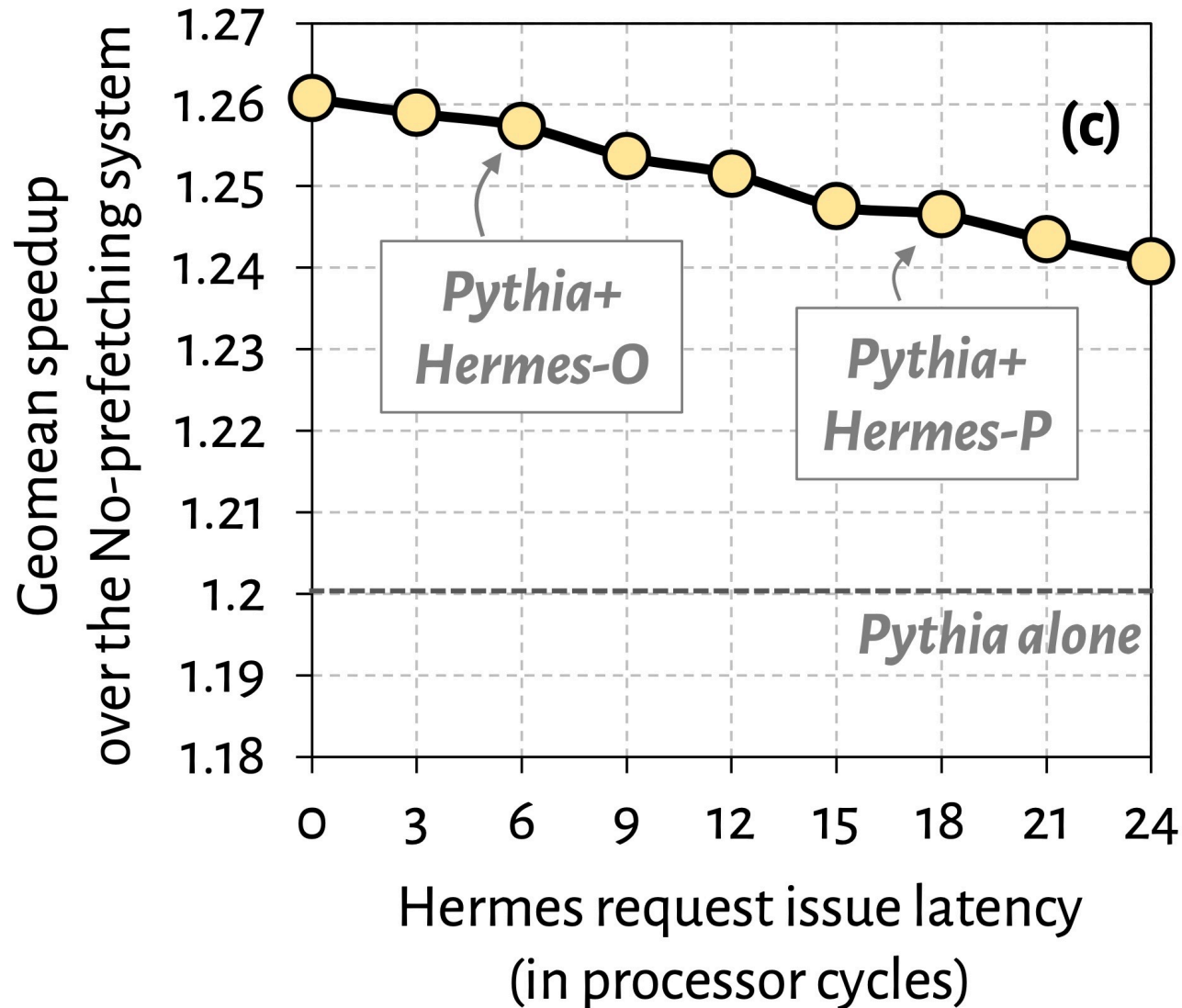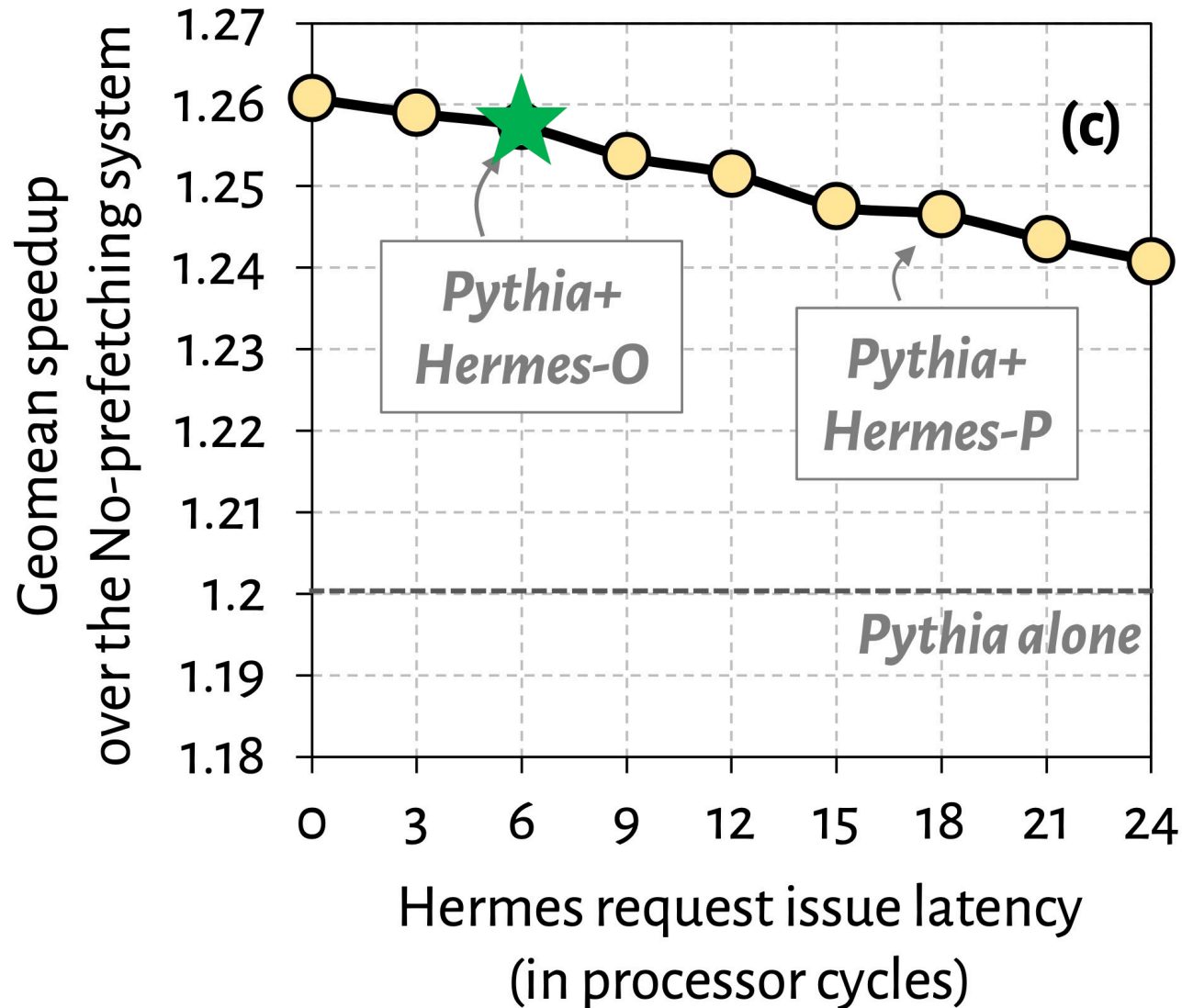
Categories: SPEC06, SPEC17, PARSEC, Ligra, CVP, MIX, GEOMEAN

Hermes in combination with Pythia outperforms Pythia alone by **5.1%** on average

SAFARI

Effect of Hermes Request Issue Latency

(c)

Geomean speedup over the No-prefetching system (y-axis)

Hermes request issue latency (in processor cycles) (x-axis)

Pythia+ Hermes-O

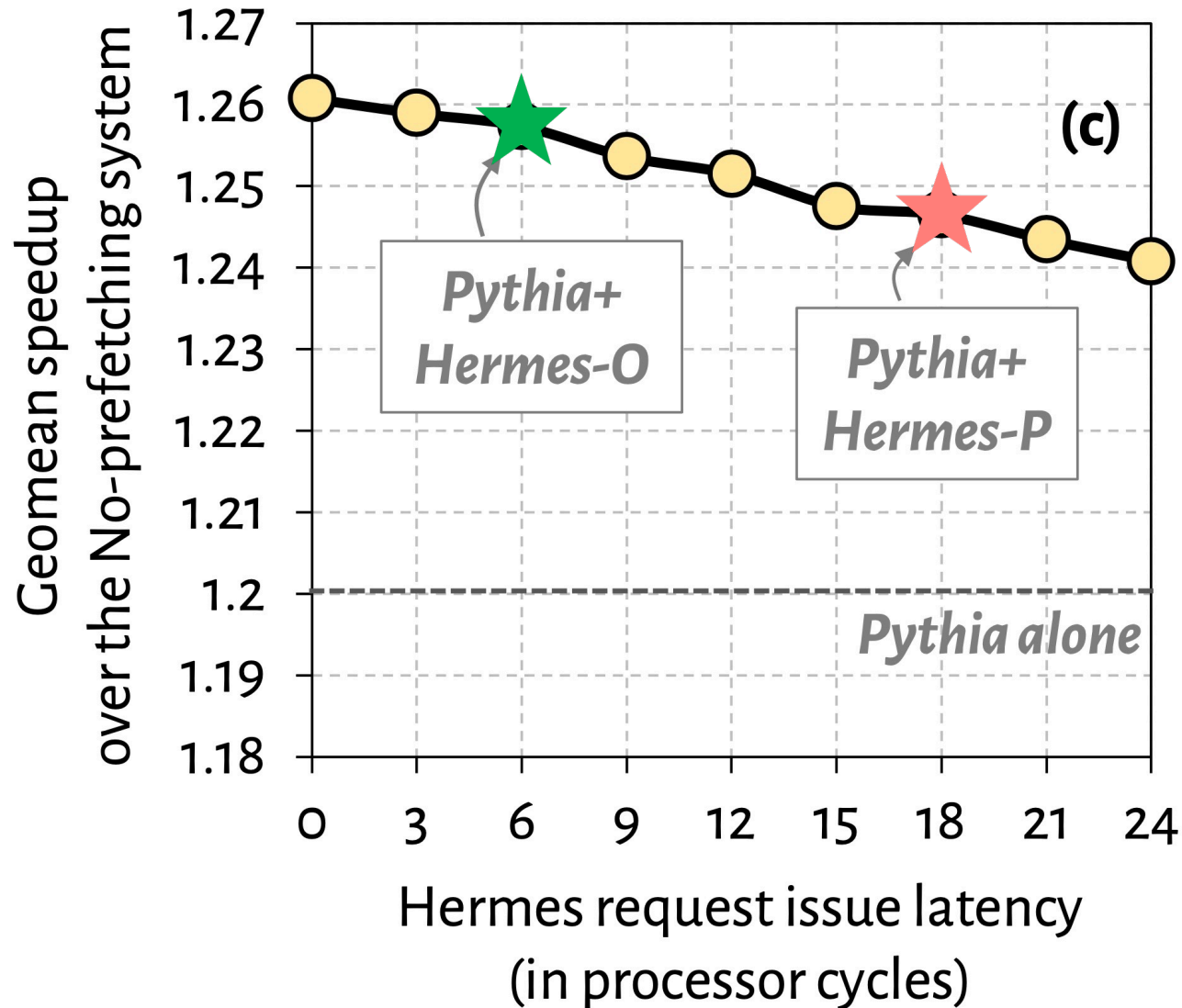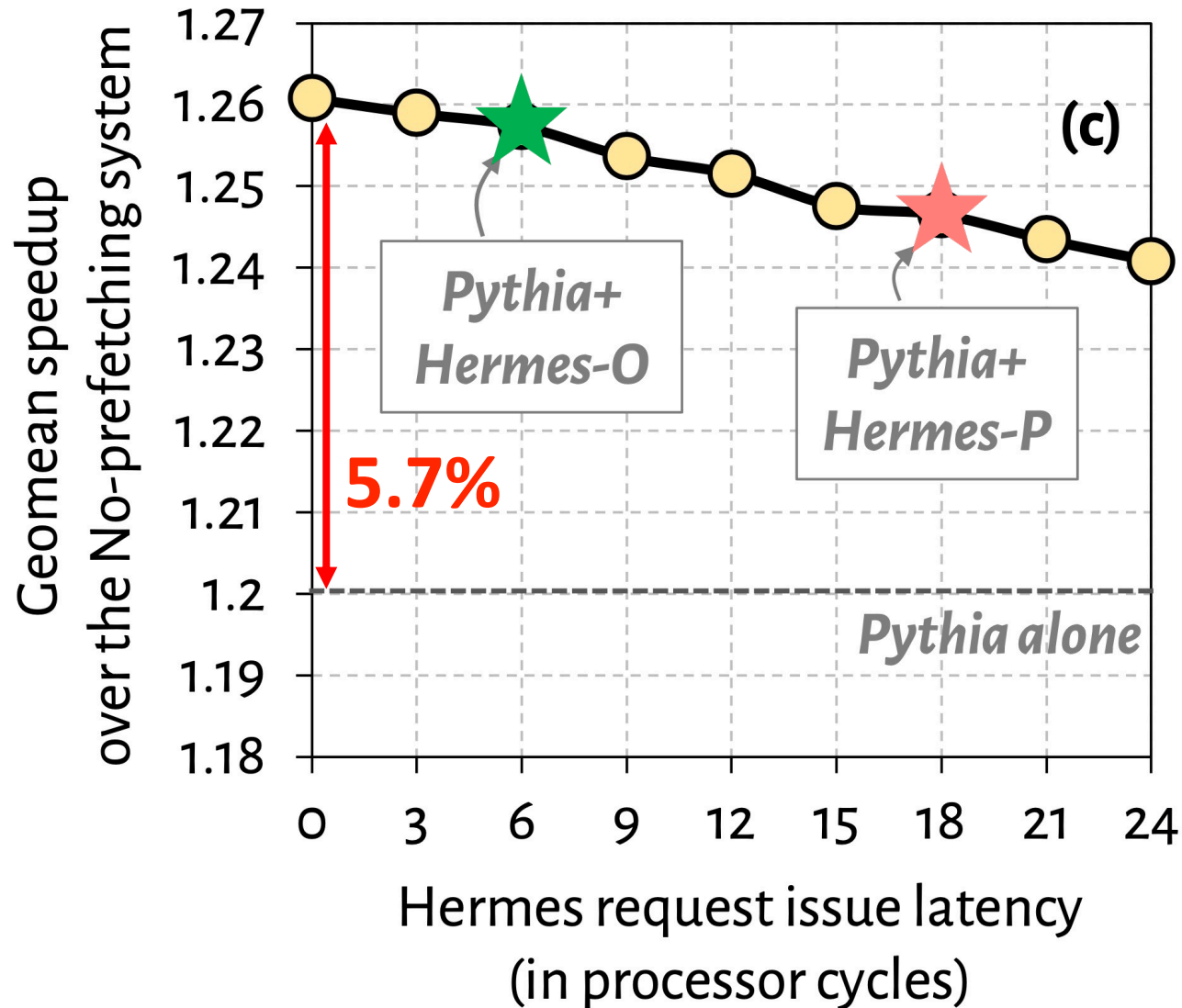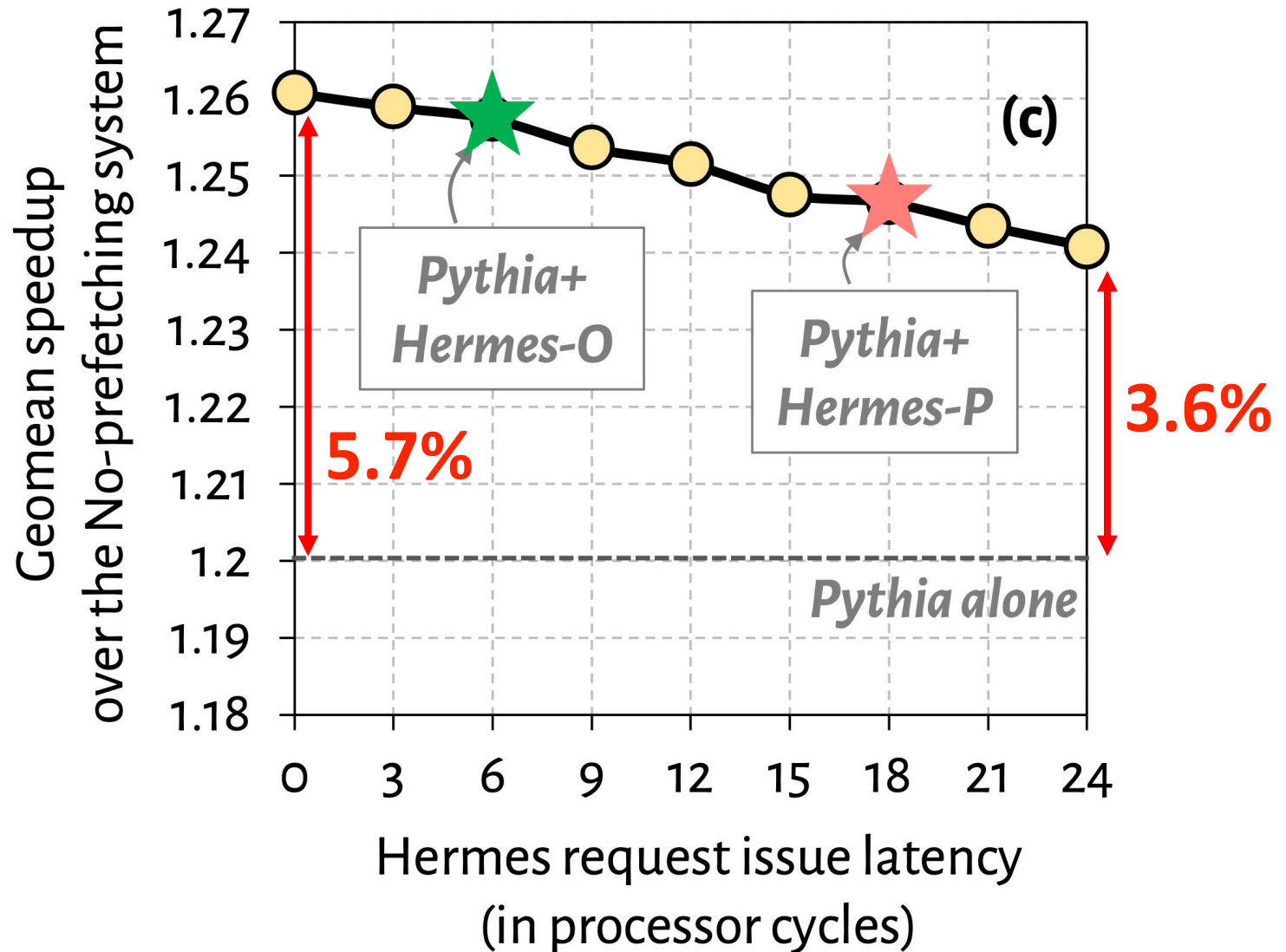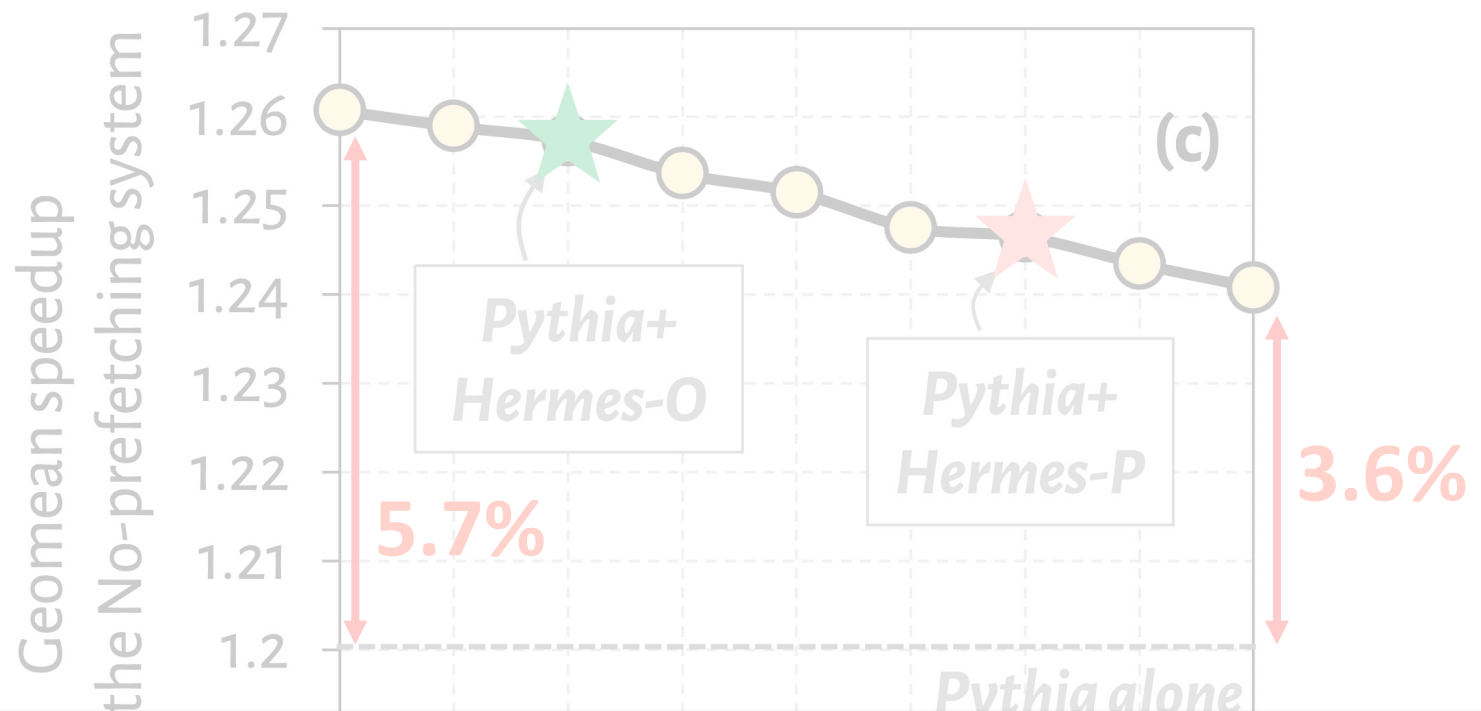Pythia+ Hermes-P

Pythia alone

Effect of Hermes Request Issue Latency

Effect of Hermes Request Issue Latency

Effect of Hermes Request Issue Latency
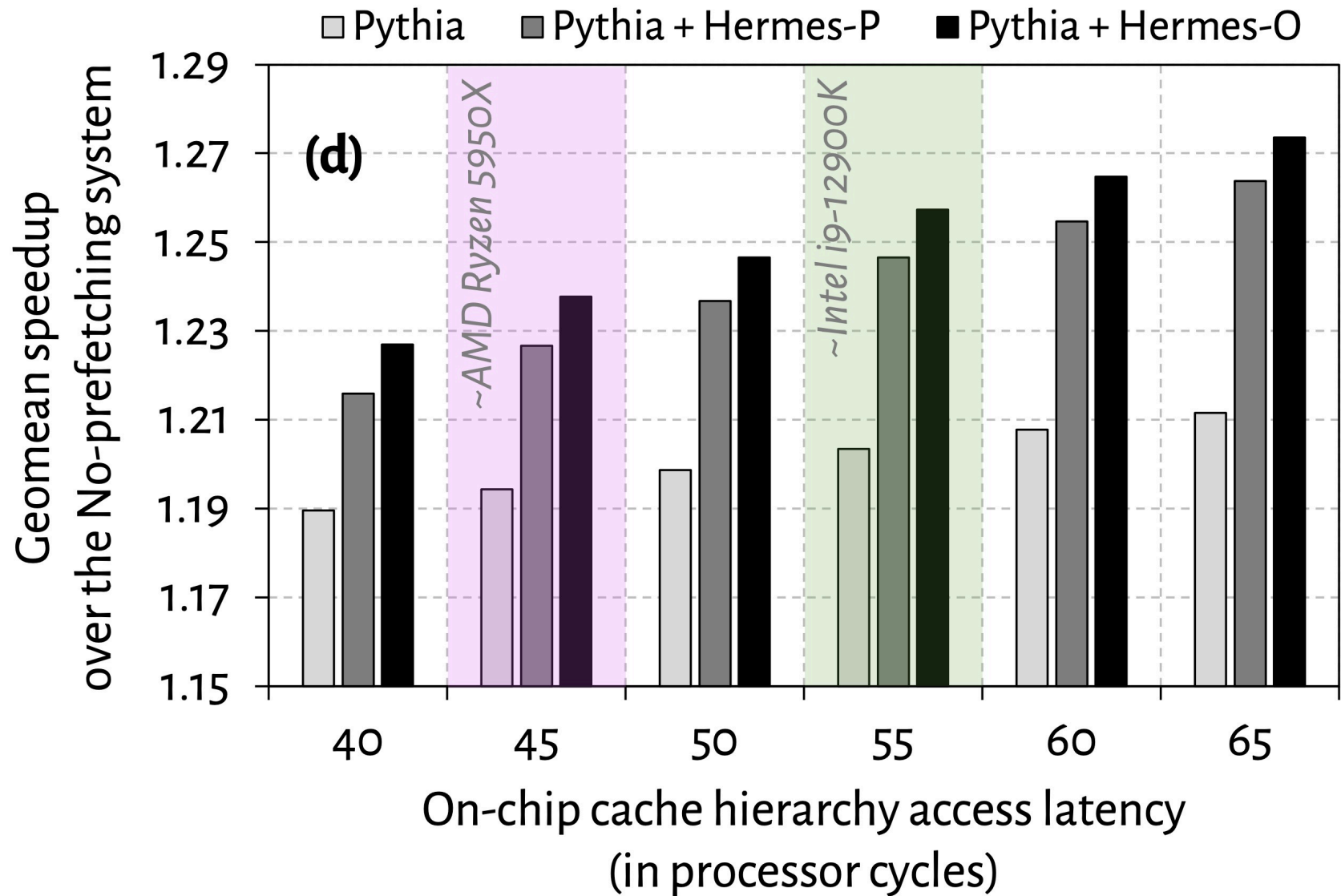
SAFARI

Effect of Hermes Request Issue Latency

Effect of Hermes Request Issue Latency

Geomean speedup over the No-prefetching system

1.27
1.26
1.25
1.24
1.23
1.22
1.21
1.2

(c)

Pythia+ Hermes-O

Pythia+ Hermes-P

5.7%

3.6%

Pythia alone

Hermes request issue latency
(in processor cycles)

Hermes in combination with Pythia outperforms Pythia alone even with a 24-cycle Hermes request issue latency

Effect of Cache Hierarchy Access Latency

SAFARI

Effect of Cache Hierarchy Access Latency

(d)

3.6%

~AMD Ryzen 5950X

~Intel i9-12900K

Legend: □ Pythia ■ Pythia + Hermes-P ■ Pythia + Hermes-O

Y-axis: Geomean speedup over the No-prefetching system

X-axis: On-chip cache hierarchy access latency (in processor cycles)

SAFARI

Effect of Cache Hierarchy Access Latency

(d)

Geomean speedup over the No-prefetching system

- Pythia
- Pythia + Hermes-P
- Pythia + Hermes-O

~AMD Ryzen 5950X

~Intel i9-12900K

3.6%

6.2%

On-chip cache hierarchy access latency (in processor cycles)

Effect of Cache Hierarchy Access Latency

Legend: □ Pythia   ■ Pythia + Hermes-P   ■ Pythia + Hermes-O

(d)

Y-axis: Geomean speedup the No-prefetching system

Y-axis values: 1.29, 1.27, 1.25, 1.23, 1.21, 1.19

~AMD Ryzen 5950X

~Intel i9-12900K

3.6%

6.2%

Hermes can provide even higher performance benefit in future processors with bigger and slower on-chip caches

On-chip cache hierarchy access latency
(in processor cycles)

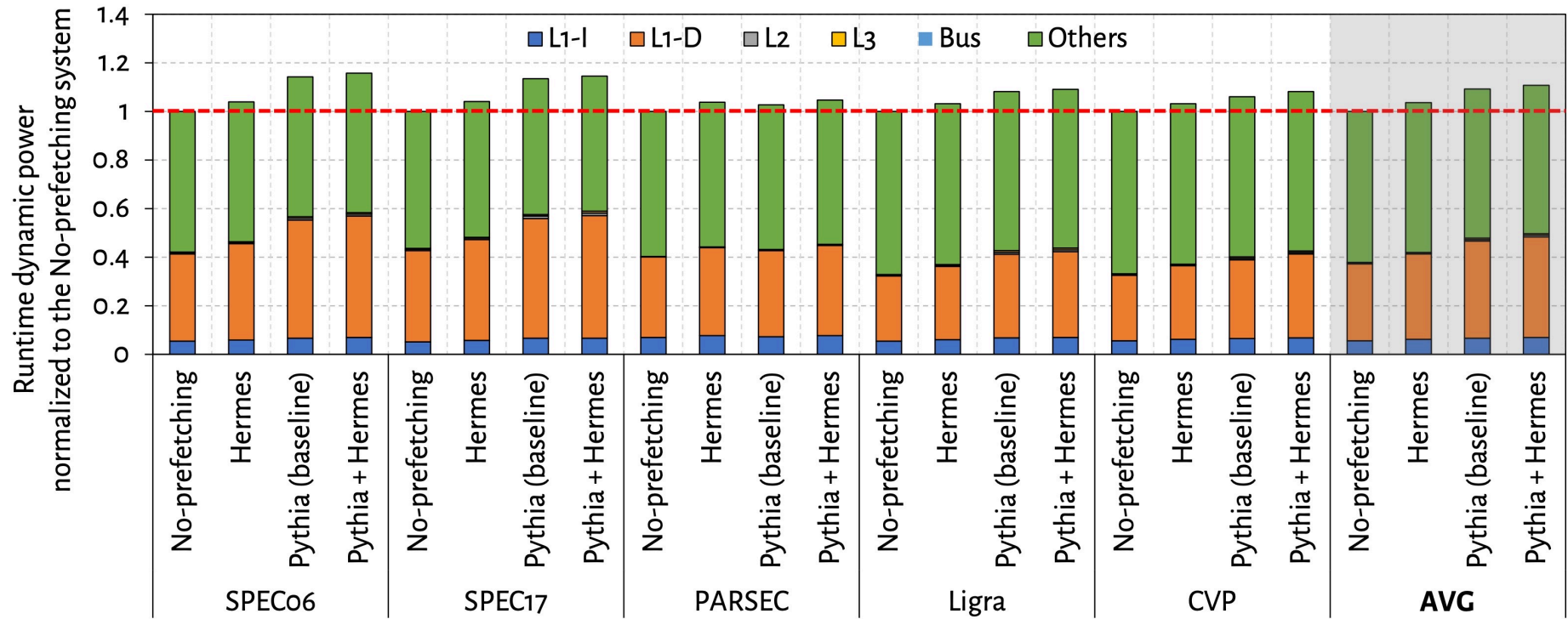SAFARI

183

Effect of Activation Threshold

SAFARI
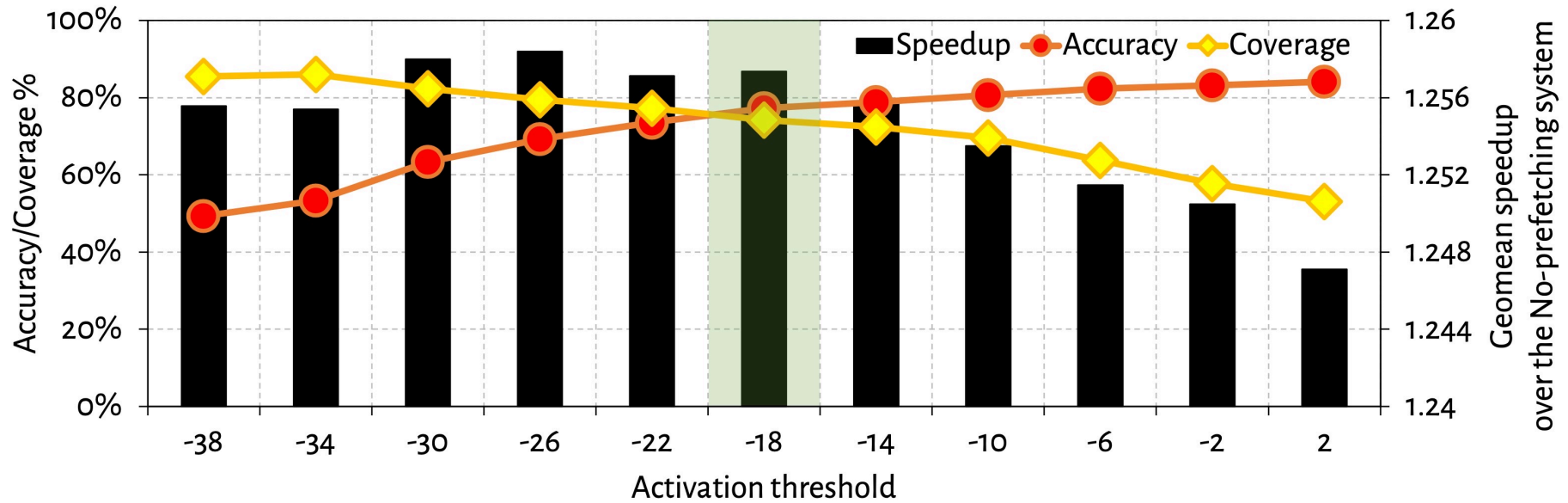
Effect of Activation Threshold
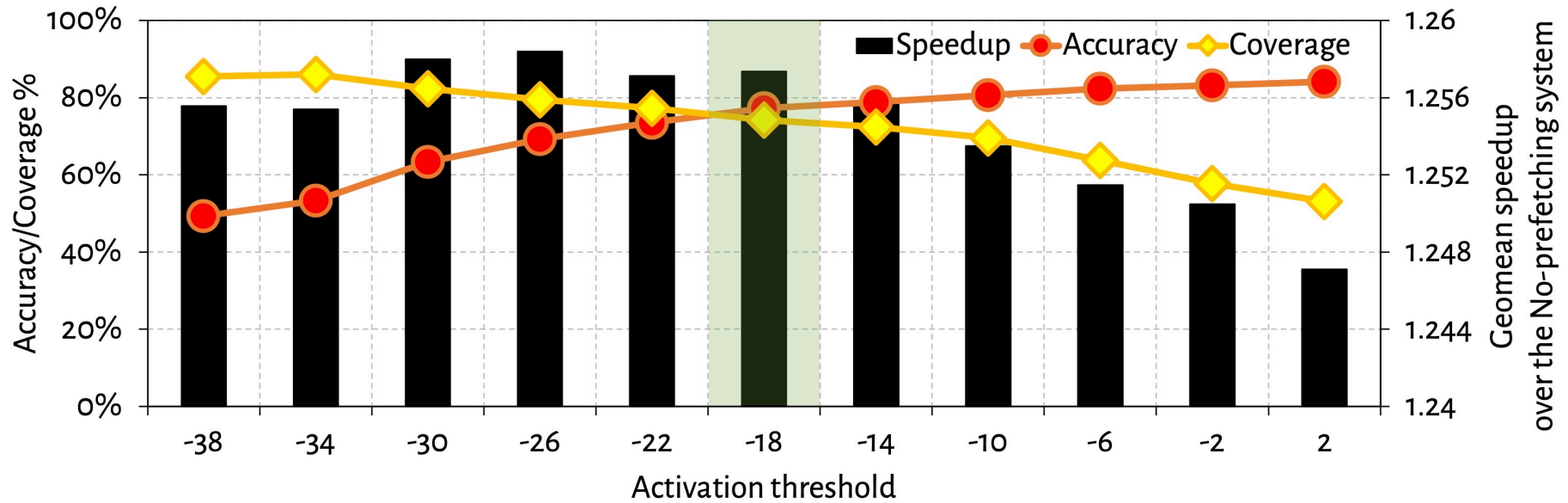
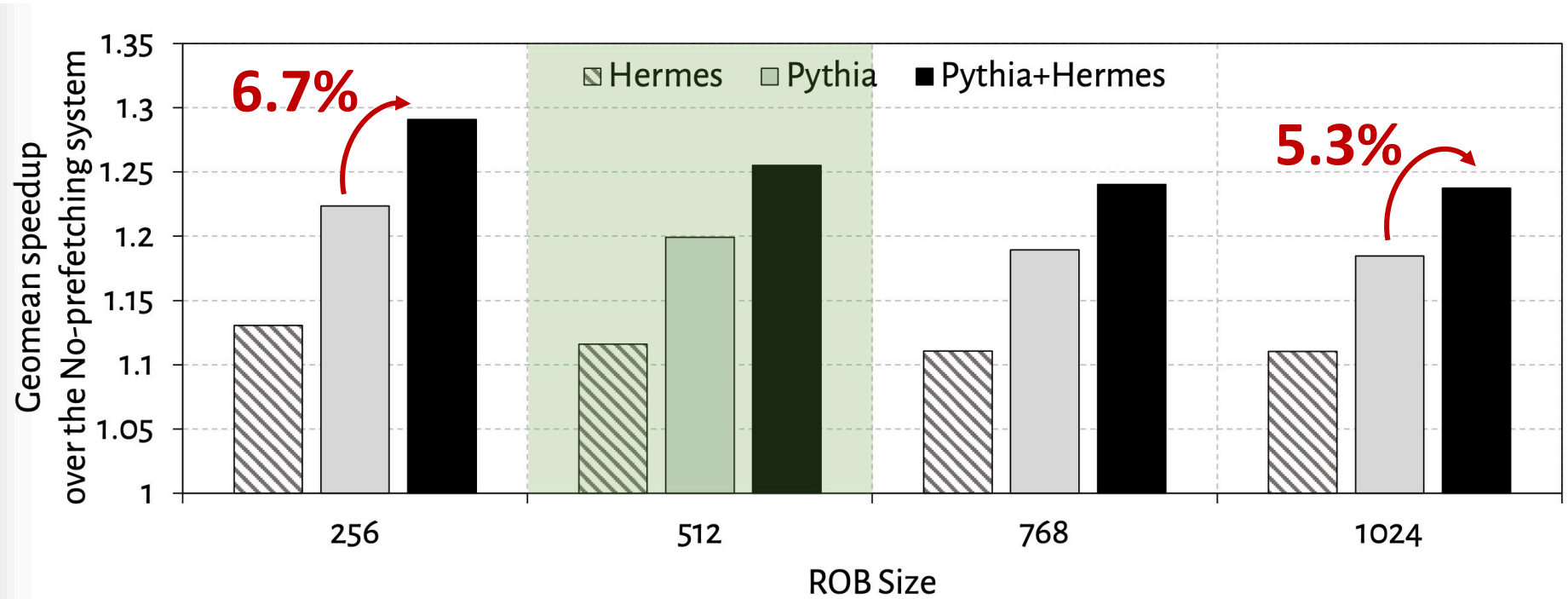With increase in activation threshold
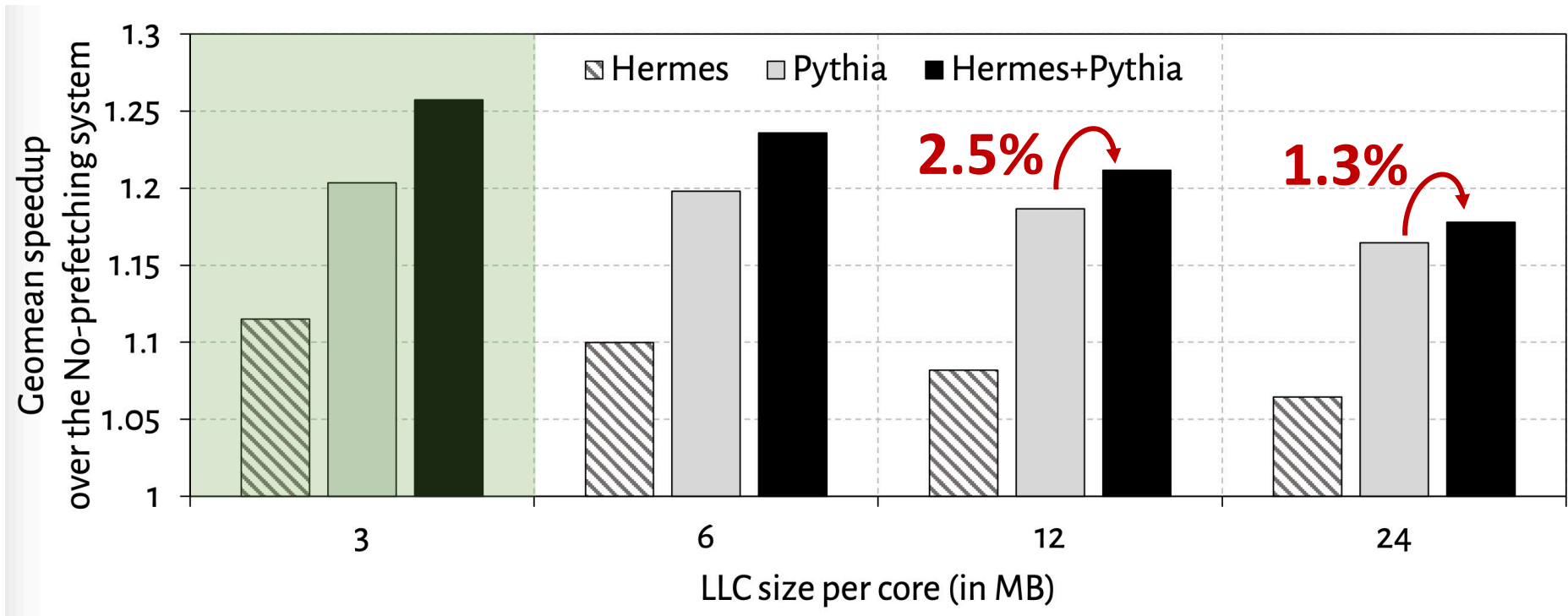1. Accuracy increases
2. Coverage decreases
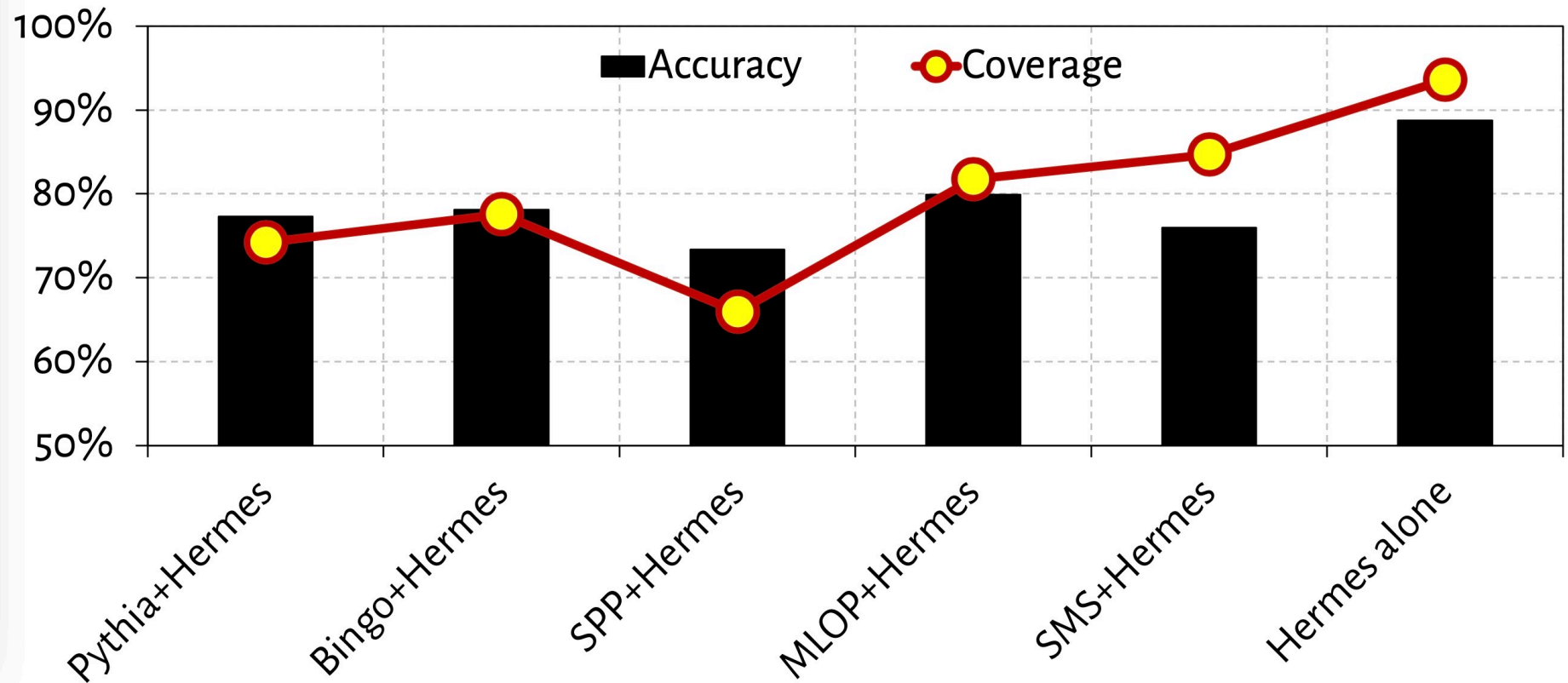
Power Overhead

Effect of ROB Size

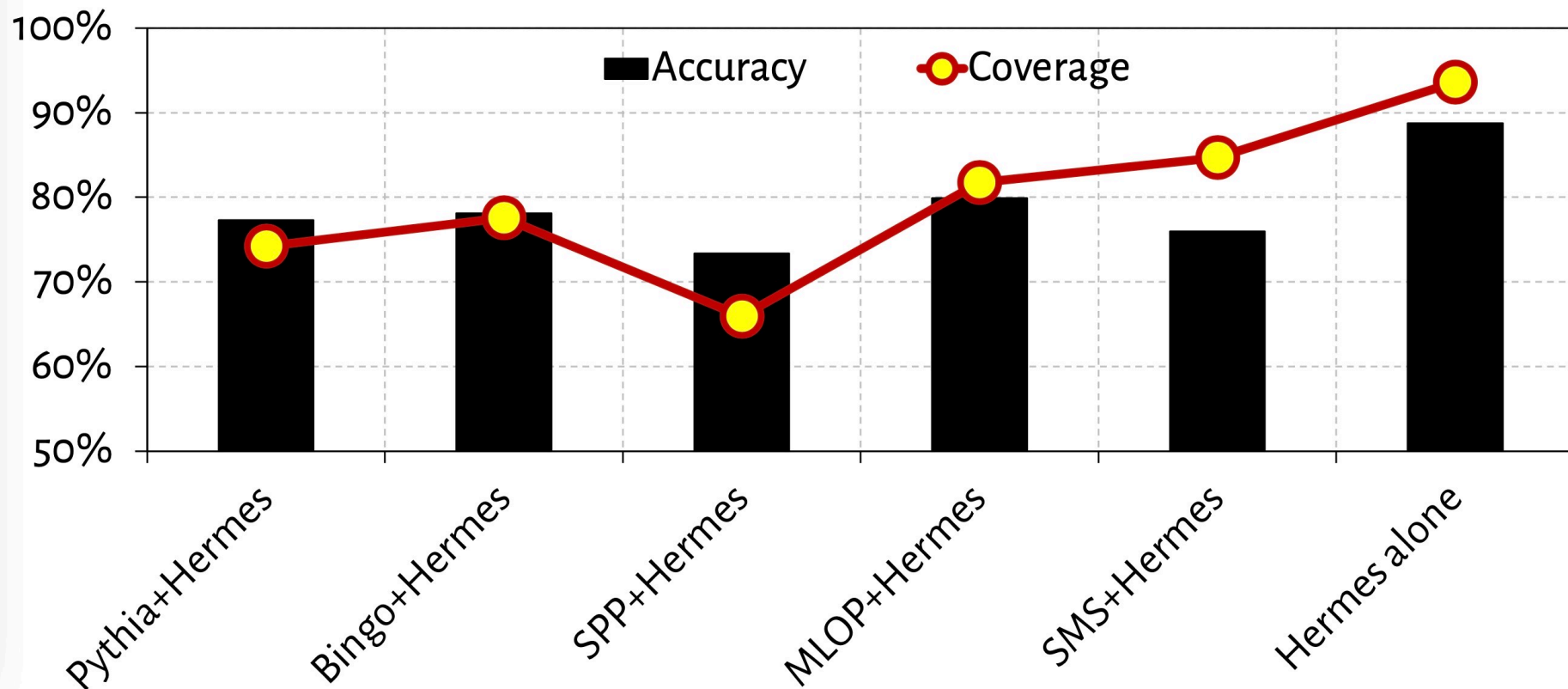**SAFARI**

Effect of LLC Size

SAFARI

Accuracy and Coverage with Different Prefetchers

SAFARI

Accuracy and Coverage with Different Prefetchers

POPET's **accuracy and coverage increases significantly** in absence of a data prefetcher

SAFARI

Increase in Main Memory Requests

SAFARI