Computer Architecture

Lecture 18a:
Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

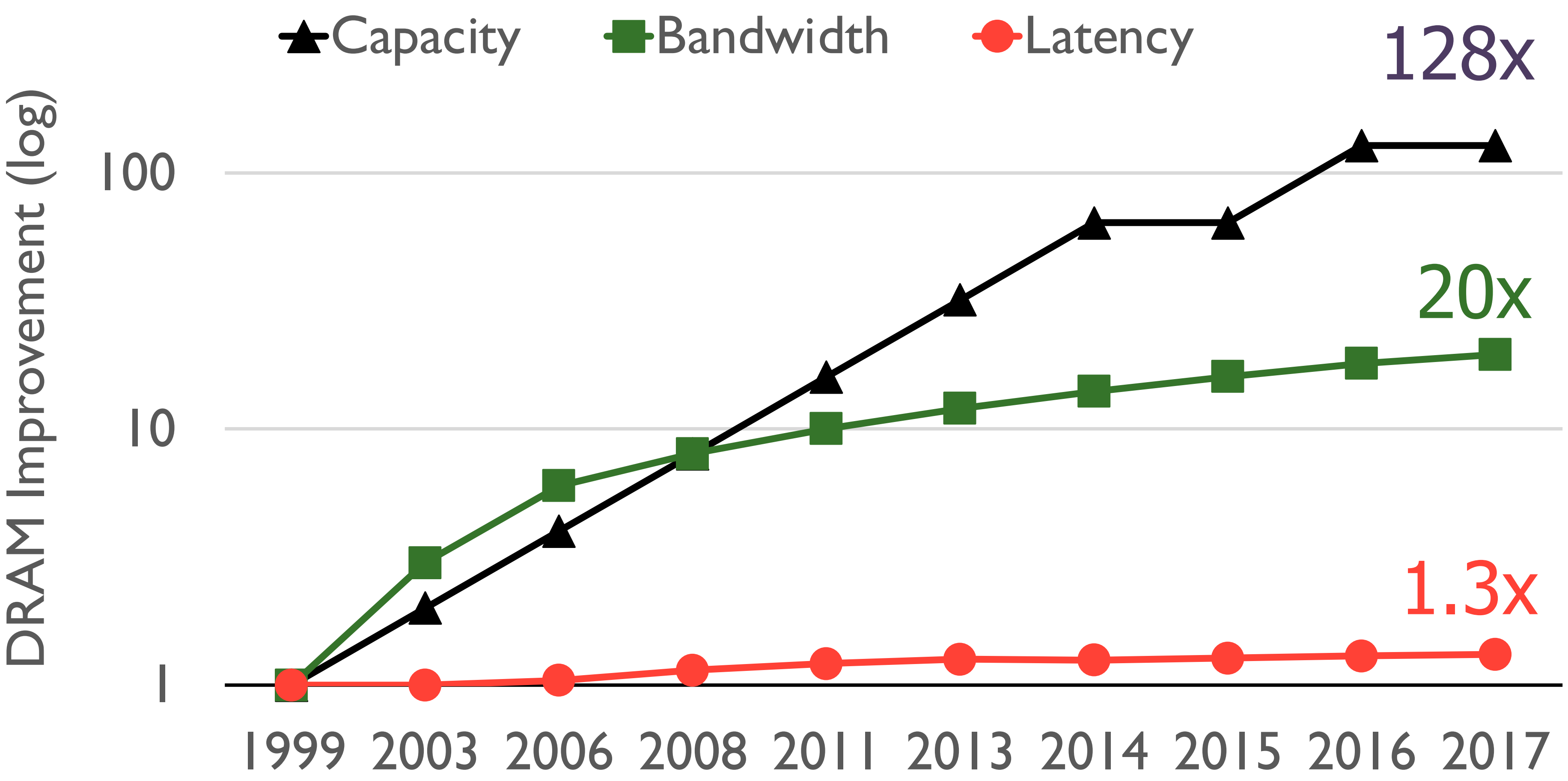
Rahul Bera
ETH Zürich

Fall 2022

25 November 2022

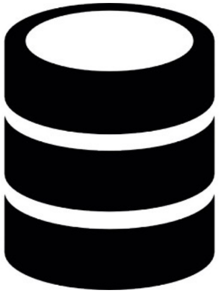
The (Memory) Latency Problem

Recall: Memory Latency Lags Behind



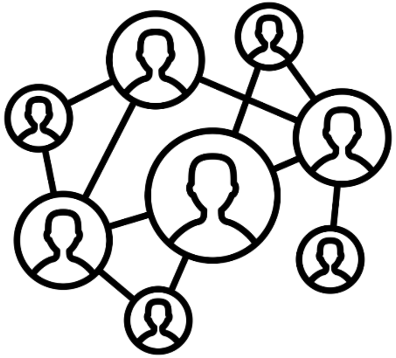
Memory latency remains almost constant

DRAM Latency Is Critical for Performance



In-memory Databases

[Mao+, EuroSys'12;
Clapp+ (**Intel**), IISWC'15]

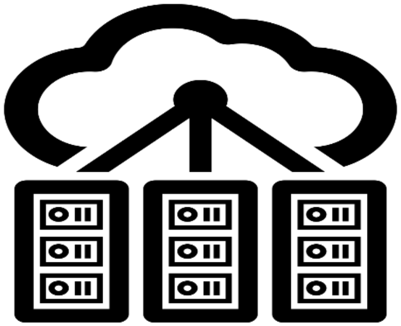


Graph/Tree Processing

[Xu+, IISWC'12; Umuroglu+, FPL'15]

In-Memory Data Analytics

[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]



Datacenter Workloads

[Kanev+ (**Google**), ISCA'15]

DRAM Latency Is Critical for Performance

In-memory Databases

Graph/Tree Processing

Long memory latency → performance bottleneck

In-Memory Data Analytics

[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]

Datacenter Workloads

[Kanev+ (**Google**), ISCA'15]

Conventional Latency Tolerance Techniques

- Out-of-order execution [initially by Tomasulo, 1967]
  - Tolerates cache misses that cannot be prefetched
  - Requires extensive hardware resources for tolerating long latencies
- Multithreading [initially in CDC 6600, 1964]
  - Works well if there are multiple threads
  - Improving single thread performance using multithreading hardware is an ongoing research effort
- Caching [initially by Wilkes, 1965]
  - Widely used, simple, effective, but inefficient, passive
  - Not all applications/phases exhibit temporal or spatial locality
- Prefetching [initially in IBM 360/91, 1967]
  - Works well for regular memory access patterns
  - Prefetching irregular access patterns is difficult, inaccurate, and hardware-intensive

Prefetching

Prefetching

- Idea: Fetch the data before it is needed (i.e. pre-fetch) by the program

- Why?
  - Memory latency is high. If we can prefetch accurately and early enough we can reduce/eliminate that latency

- Involves predicting which address will be needed in the future
  - Works if programs have predictable address patterns
  - Might mispredict if the program has irregular access patterns

Prefetcher Evaluation Metrics

- Coverage
  - Used prefetches / total demanded memory accesses from core
  - The higher the better
- Accuracy
  - Used prefetches / sent prefetches
  - The higher the better
- Timeliness
  - Memory access latency saved by a prefetch
  - The higher the better

- Bandwidth consumption
- Cache pollution
- Energy consumption, …

Prefetching: The Three Questions

- What
  - What addresses to prefetch

- When
  - When to initiate a prefetch request

- How
  - Software, execution-based, hardware

Prefetching: The Three Questions

- What
  - What addresses to prefetch
- When
  - When to initiate a prefetch request
- How
  - Software, execution-based, hardware

Challenges in Prefetching: How

- Software prefetching
  - Programmer or compiler inserts prefetch instructions

- Execution-based prefetchers
  - A “thread” is executed to prefetch data for the main program

- Hardware prefetching
  - Hardware monitors processor accesses
  - Memorizes or finds patterns/strides
  - Generates prefetch addresses accordingly

Challenges in Prefetching: How

- Software prefetching
  - Programmer or compiler inserts prefetch instructions

- Execution-based prefetchers
  - A “thread” is executed to prefetch data for the main program

- Hardware prefetching
  - Hardware monitors processor accesses
  - Memorizes or finds patterns/strides
  - Generates prefetch addresses accordingly

Hardware Prefetching

- An instruction with program counter (PC) X is accessing the following addresses:
  - A, A+D, A+2D, A+3D, …
  - Learning: PC_X is has a strided access pattern with stride D
  - Prediction: If PC_X accesses B, prefetch (B+D)

- The last few cacheline accesses are
  - A, A+3, A+5, A+8, A+10, A+13, …
  - Learning: Cacheline deltas +3 and +2 is repeating alternatively
  - Prediction: If last delta is +3 (or +2), predict next delta to be +2 (or +3)

Hardware Prefetching

- PC, Sequence of cacheline deltas, …
  - Program features
  - Represents execution "context" of the program

- Associates access patterns from past memory requests with program features

Program feature → Access Pattern

- More program features
  - Branch PCs
  - Page number
  - Page offset
  - …
  - Or a combination of these attributes

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

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

Executive Summary

- **Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with program context (called **feature**)

- **Problem**: Three key shortcomings of prior prefetchers:
  - Predict mainly using a **single program feature**
  - Lack **inherent system awareness** (e.g., memory bandwidth usage)
  - Lack **in-silicon customizability**

- **Goal**: Design a prefetching framework that:
  - Learns from **multiple features** and **inherent system-level feedback**
  - Can be **customized in silicon** to use different features and/or prefetching objectives

- **Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
  - Takes **adaptive** prefetch decisions using multiple features and system-level feedback
  - Can be **customized in silicon** for target workloads via simple configuration registers
  - Proposes **a realistic and practical** implementation of RL algorithm in hardware

- **Key Results**:
  - Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
  - Outperforms best prefetcher (in 1-core config.) by **3.4%, 7.7% and 17%** in 1/4/bw-constrained cores
  - Up to **7.8% more performance** over basic Pythia across Ligra workloads via simple customization

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

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

1 Mainly use **one** program feature for prediction

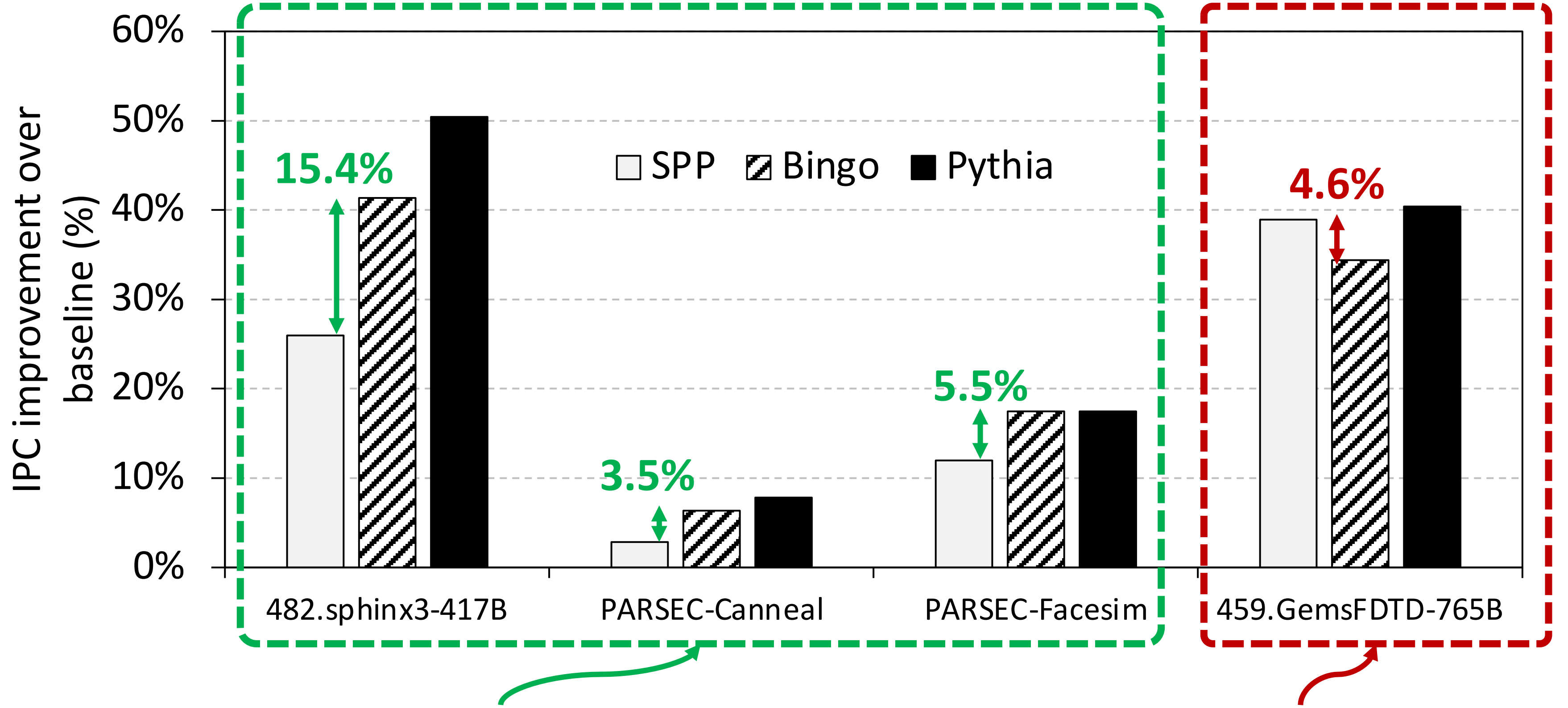
2 Lack **inherent system awareness**

3 Lack **in-silicon customizability**

Why do prefetchers not perform well?

(1) Single-Feature Prefetch Prediction

- Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation exists**

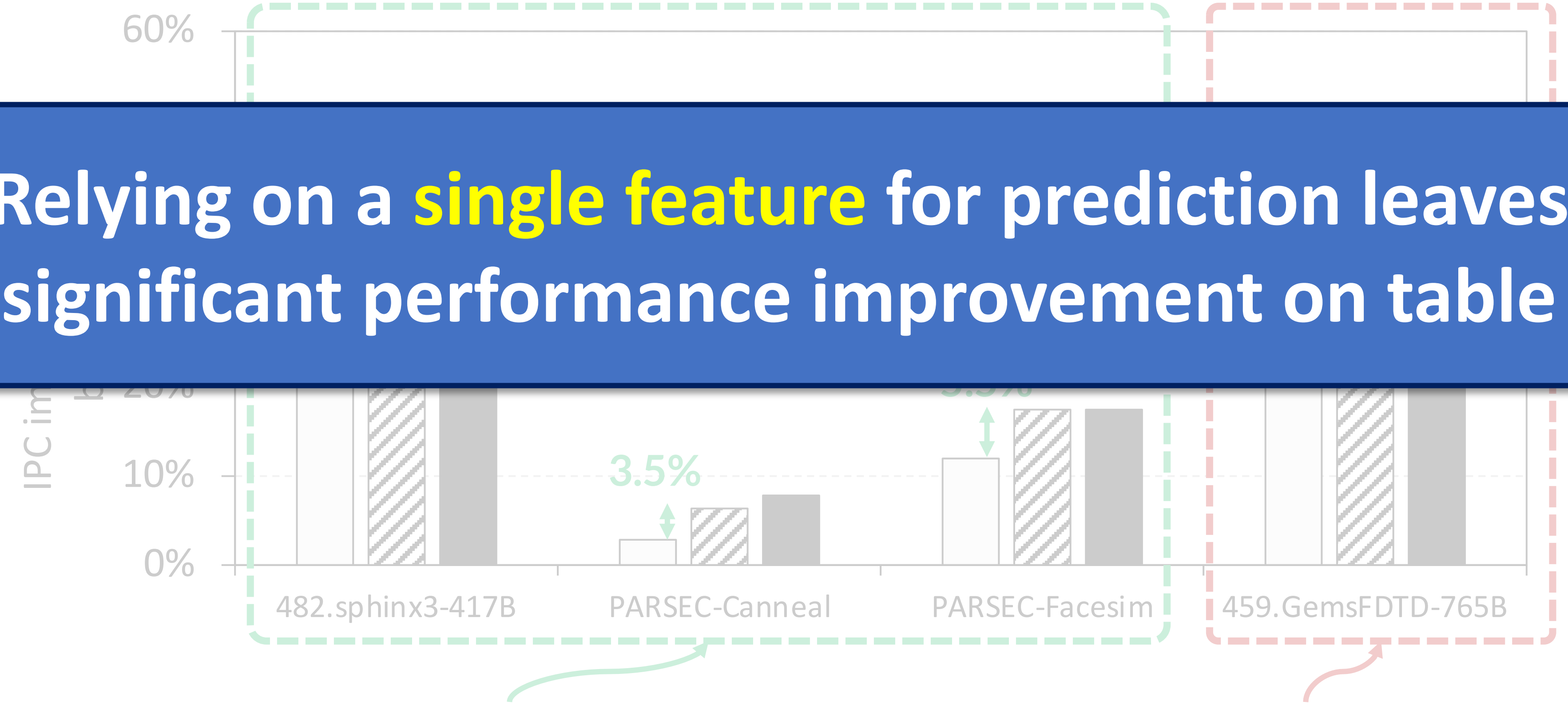


Bingo [1] performs better **SPP [2] performs better**

[1] Bakshalipour et al., HPCA'19 *[2] Kim et al., MICRO'16*

(1) Single-Feature Prefetch Prediction

- Provides **good** performance **gains** mainly on workloads where the **feature-to-pattern correlation exists**



Relying on a single feature for prediction leaves significant performance improvement on table

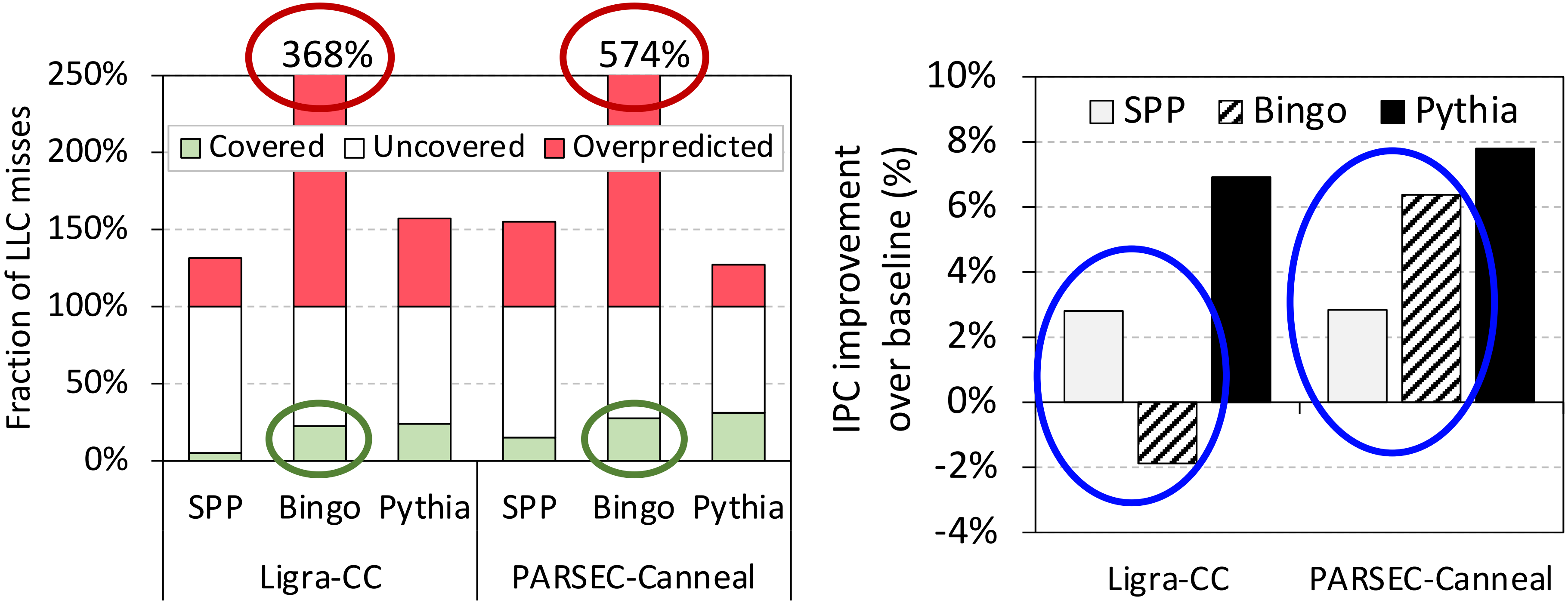
Bingo [1] **performs better**

SPP [2] **performs better**

[1] Bakshalipour et al., HPCA'19 *[2] Kim et al., MICRO'16*

(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, …)
  - Performance loss in **resource-constrained** configurations



Similar coverage

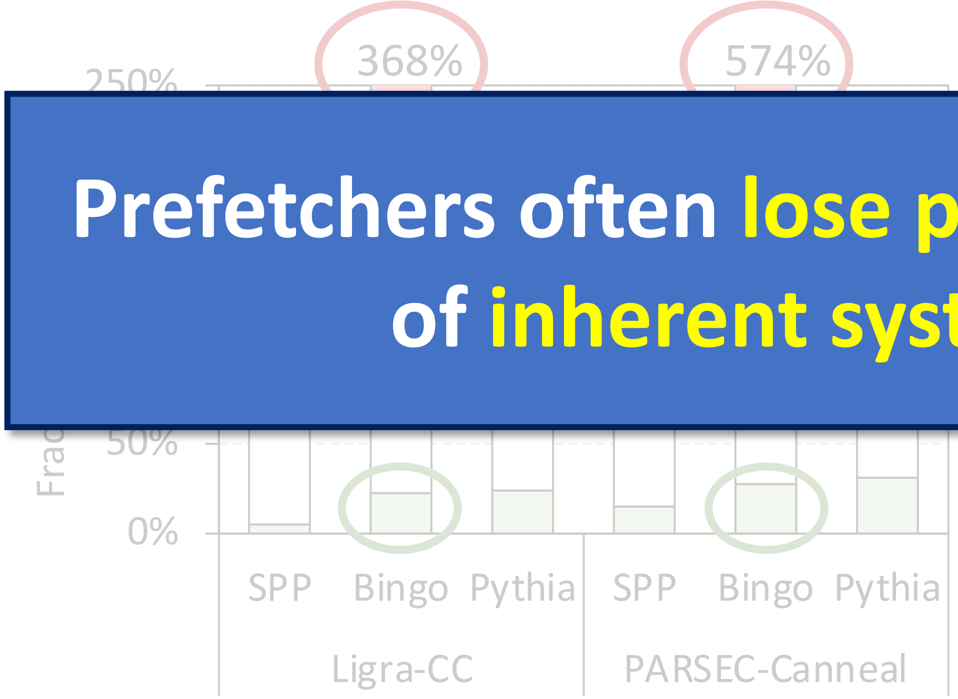
Lower overpredictions

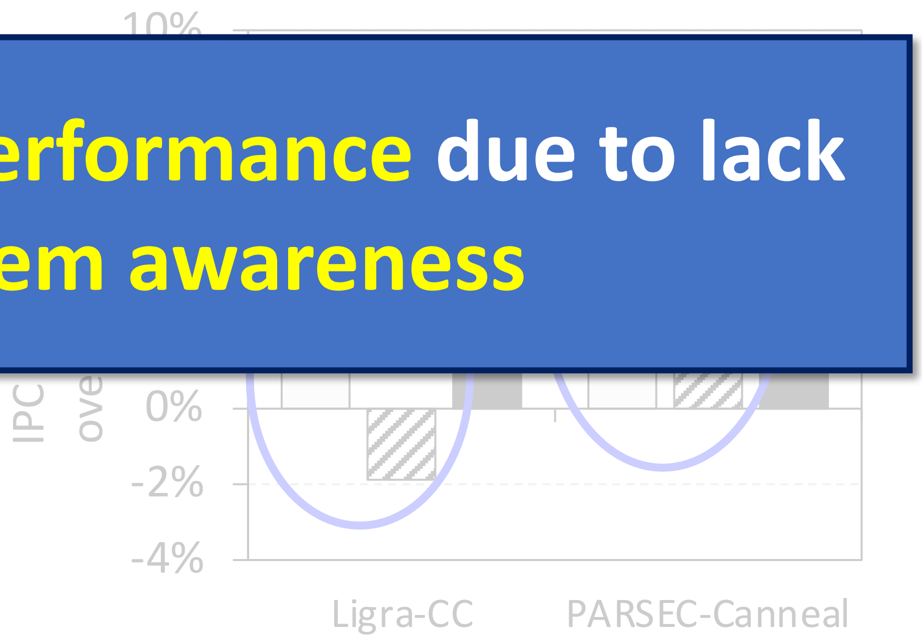
Yet, lower performance

SAFARI

(2) Lack of Inherent System Awareness

- Little understanding of **undesirable effects** (e.g., memory bandwidth usage, cache pollution, …)
  - Performance loss in **resource-constrained** configurations





Prefetchers often lose performance due to lack of inherent system awareness

Similar coverage

Lower overpredictions

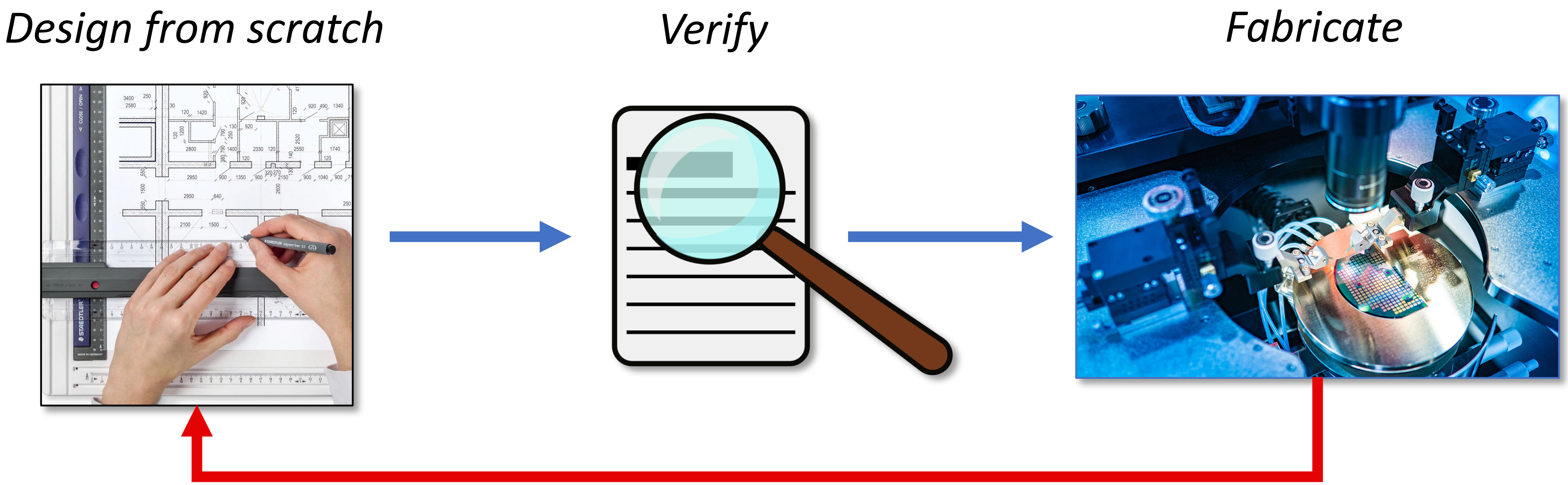
Yet, lower performance

SAFARI

(3) Lack of In-silicon Customizability

- Feature **statically** selected at design time
  - **Rigid hardware** designed specifically to exploit that feature

- **No way to change** program feature and/or change prefetcher’s objective **in silicon**
  - **Cannot adapt** to a wide range of workload demands


Design from scratch

Verify

Fabricate

Our Goal

1

Autonomously learns to prefetch using **multiple program context information** and **system-level feedback**

2

Can be **customized in silicon** to change program context information or prefetching objective on the fly

Our Proposal

Pythia

Formulates prefetching as a
reinforcement learning problem

Pythia is named after the oracle of Delphi, who is known for her accurate prophecies
https://en.wikipedia.org/wiki/Pythia

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

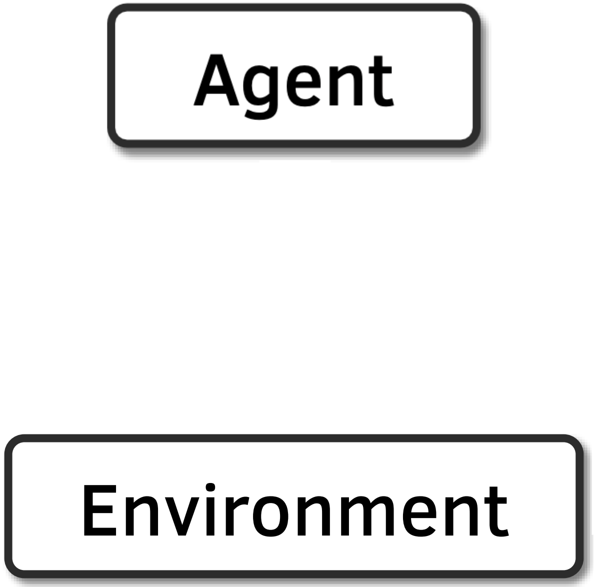
Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Basics of Reinforcement Learning (RL)

- Algorithmic approach to learn to take an **action** in a given **situation** to maximize a numerical **reward**



Agent

Environment

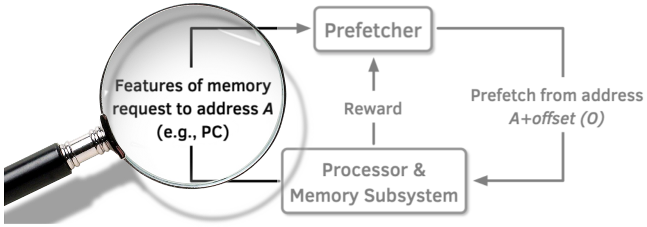
- Agent stores **Q-values** for *every* state-action pair
  - **Expected reward** for taking an action in a state
  - Given a state, selects action that provides **highest** Q-value

Formulating Prefetching as RL

What is State?

- ***k*-dimensional** vector of features

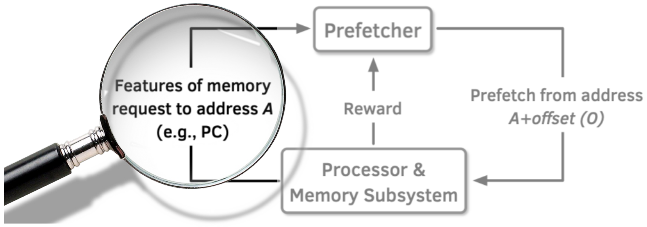
$$S \equiv \{\phi_S^1, \phi_S^2, \ldots, \phi_S^k\}$$


Features of memory request to address *A* (e.g., PC)
Prefetcher
Reward
Prefetch from address *A+offset (O)*
Processor & Memory Subsystem

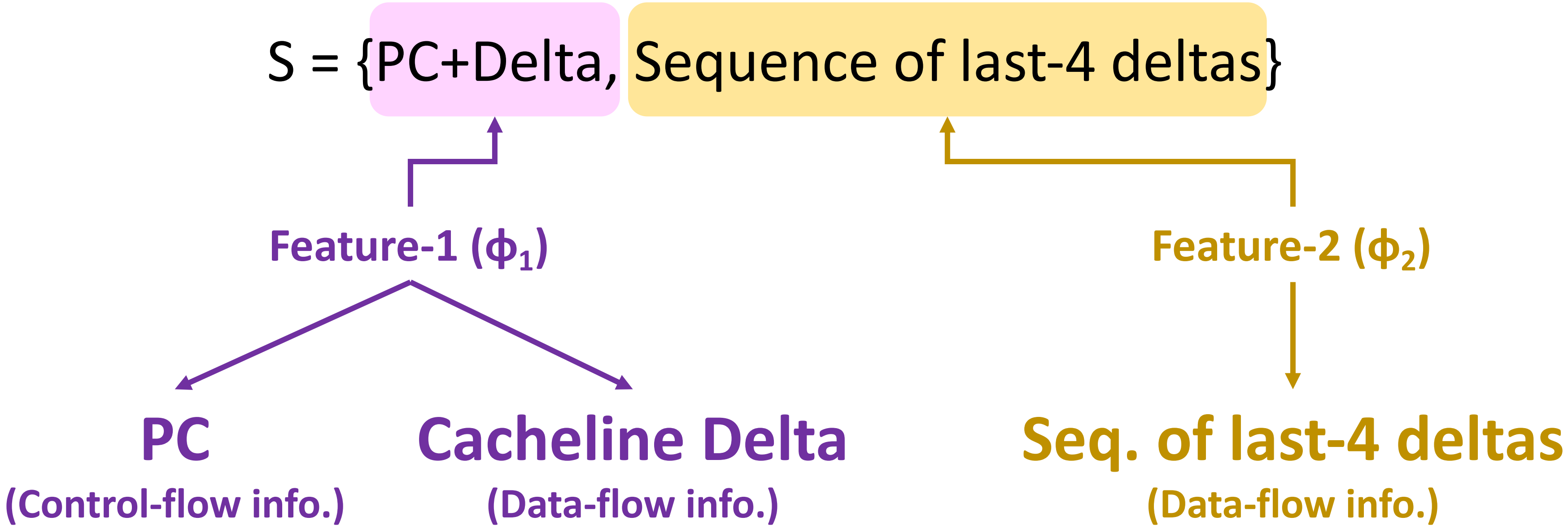
- Feature = control-flow + data-flow

- **Control-flow examples**
  - PC
  - Branch PC
  - Last-3 PCs, …
- **Data-flow examples**
  - Cacheline address
  - Physical page number
  - Delta between two cacheline addresses
  - Last 4 deltas, …

What is State?


Features of memory request to address *A* (e.g., PC)
Prefetcher
Reward
Prefetch from address *A+offset (O)*
Processor & Memory Subsystem

Example of a state information



S = {PC+Delta, Sequence of last-4 deltas}

Feature-1 (ϕ_1)

Feature-2 (ϕ_2)

PC
(Control-flow info.)

Cacheline Delta
(Data-flow info.)

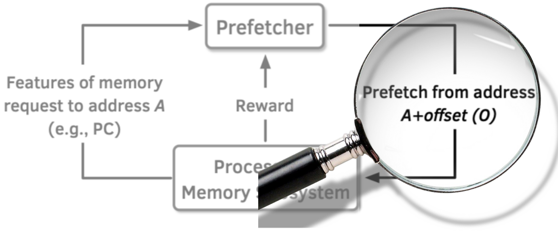
Seq. of last-4 deltas
(Data-flow info.)

What is Action?

Given a demand access to address A

the action is to **select prefetch offset "O"**

- Issue prefetch to (A+O)


Prefetcher
Features of memory request to address *A* (e.g., PC)
Reward
Processor and Memory Subsystem
Prefetch from address *A+offset (O)*

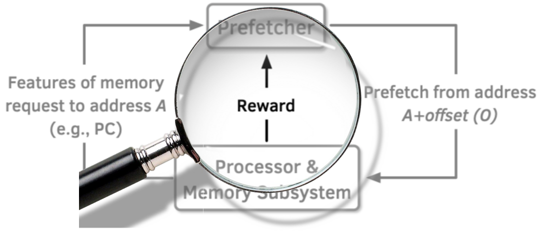
- **Action-space**: 127 actions in the range [-63, +63]
  - For a processor with 4KB page and 64B cacheline

- Upper and lower limits ensure prefetches do not cross **physical page boundary**

- A **zero offset** means **no prefetch** is generated

What is Reward?

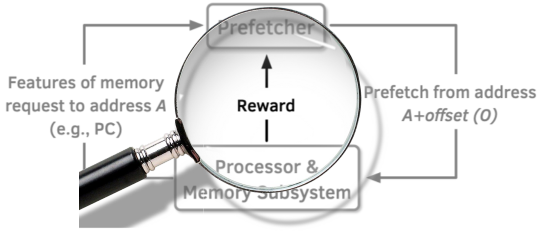
- Defines the **objective** of Pythia


Prefetcher
Features of memory request to address *A* (e.g., PC)
Reward
Prefetch from address *A+offset (O)*
Processor & Memory Subsystem

- Encapsulates two metrics:
  - **Prefetch usefulness** (e.g., accurate, late, out-of-page, …)
  - **System-level feedback** (e.g., mem. b/w usage, cache pollution, energy, …)

- We demonstrate Pythia with **memory bandwidth usage** as the system-level feedback in the paper

What is Reward?

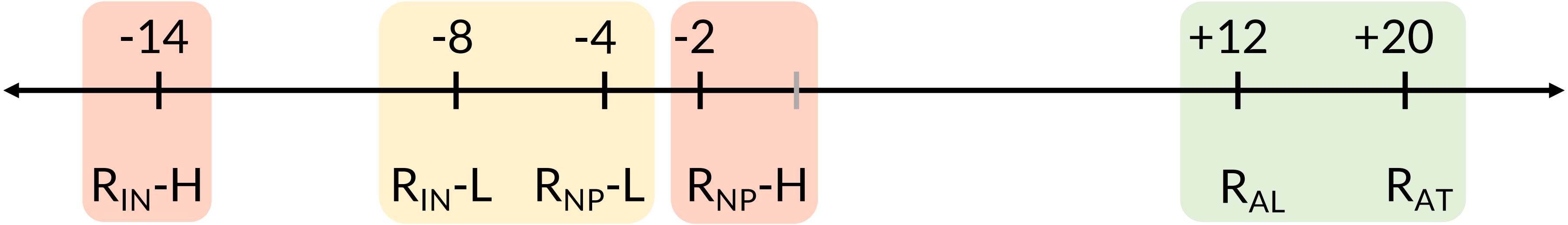

Prefetcher
Features of memory request to address A (e.g., PC)
Reward
Prefetch from address A+offset (O)
Processor & Memory Subsystem

- **Seven** distinct reward levels
  - *Accurate and timely* (R_{AT})
  - *Accurate but late* (R_{AL})
  - *Loss of coverage* (R_{CL})
  - *Inaccurate*
    - With low memory b/w usage (R_{IN}-L)
    - With high memory b/w usage (R_{IN}-H)
  - *No-prefetch*
    - With low memory b/w usage (R_{NP}-L)
    - With high memory b/w usage(R_{NP}-H)

- Values are set at design time via **automatic design-space exploration**
  - Can be **customized** further in silicon for higher performance

Steering Pythia's Objective via Reward Values

- Example reward configuration for
  - Generating **accurate prefetches**
  - Making **bandwidth-aware** prefetch decisions



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

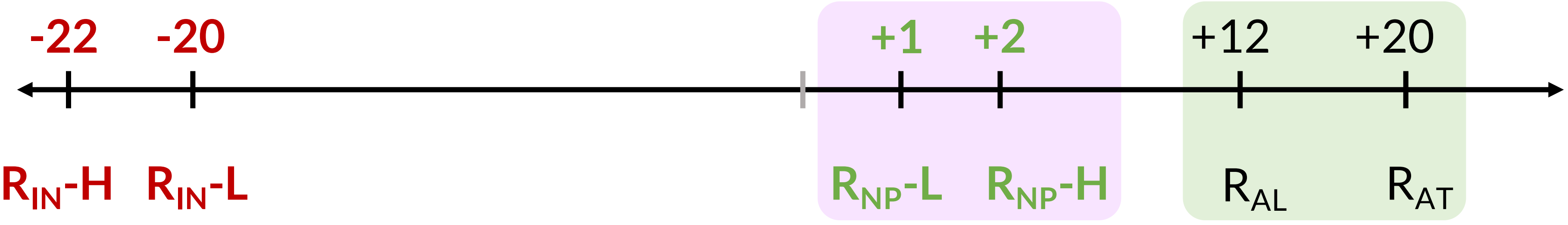
1 **Highly prefers to generate accurate prefetches**

2 **Prefers not to prefetch if memory bandwidth usage is low**

3 **Strongly prefers not to prefetch if memory bandwidth usage is high**

Steering Pythia's Objective via Reward Values

- Customizing reward values to make Pythia **conservative** towards prefetching



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

1 **Highly prefers to generate accurate prefetches**

2 **Otherwise prefers not to prefetch**

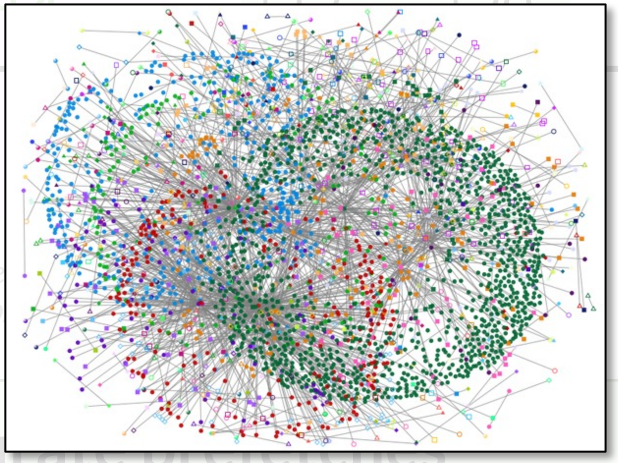
SAFARI

Steering Pythia’s Objective via Reward Values

- Customizing reward values to make Pythia conservative towards p

Strict Pythia configuration

Server-class processors



Bandwidth-sensitive workloads

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

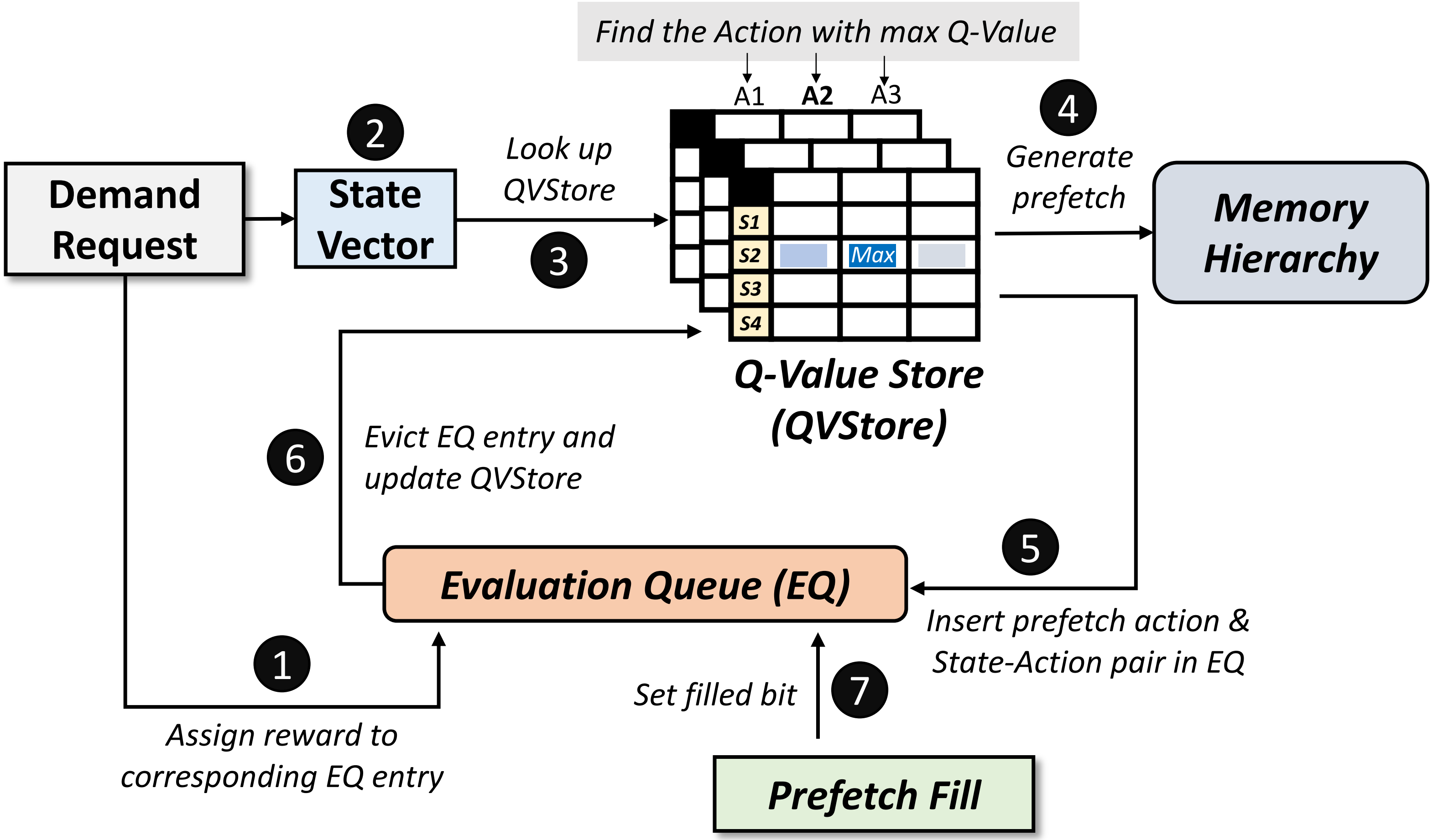
Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Pythia Overview

- **Q-Value Store**: Records Q-values for *all* state-action pairs
- **Evaluation Queue**: A FIFO queue of recently-taken actions



SAFARI

More in the Paper

- **Pipelined search** operation for QVStore

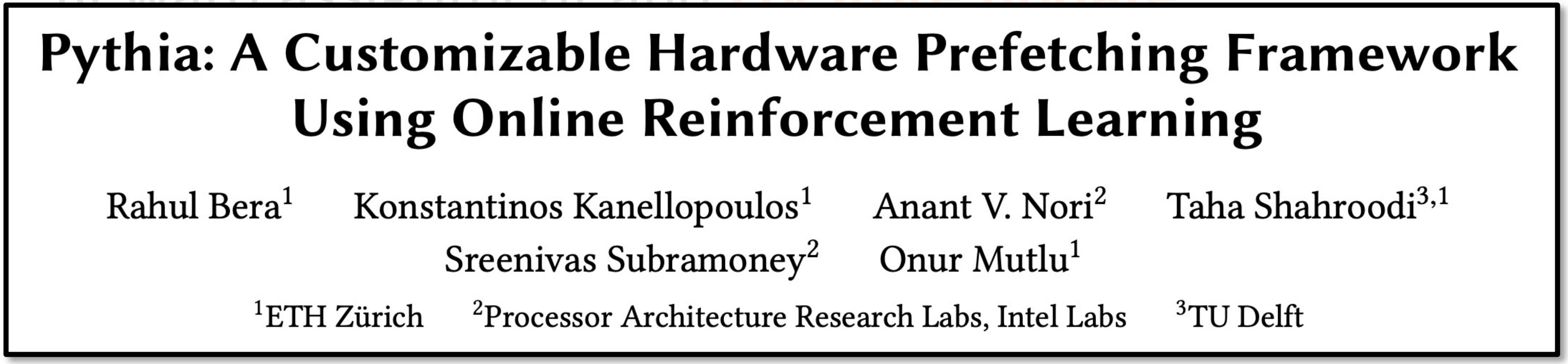
- Reward assignment and **QVStore update**

- **Automatic design-space exploration**
  - Feature types
  - Action
  - Reward and Hyperparameter values

More in the Paper

- Pipelined search operation for QVStore

- Reward assignment and QVStore update



Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]

[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

  - Reward a

https://arxiv.org/pdf/2109.12021.pdf

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

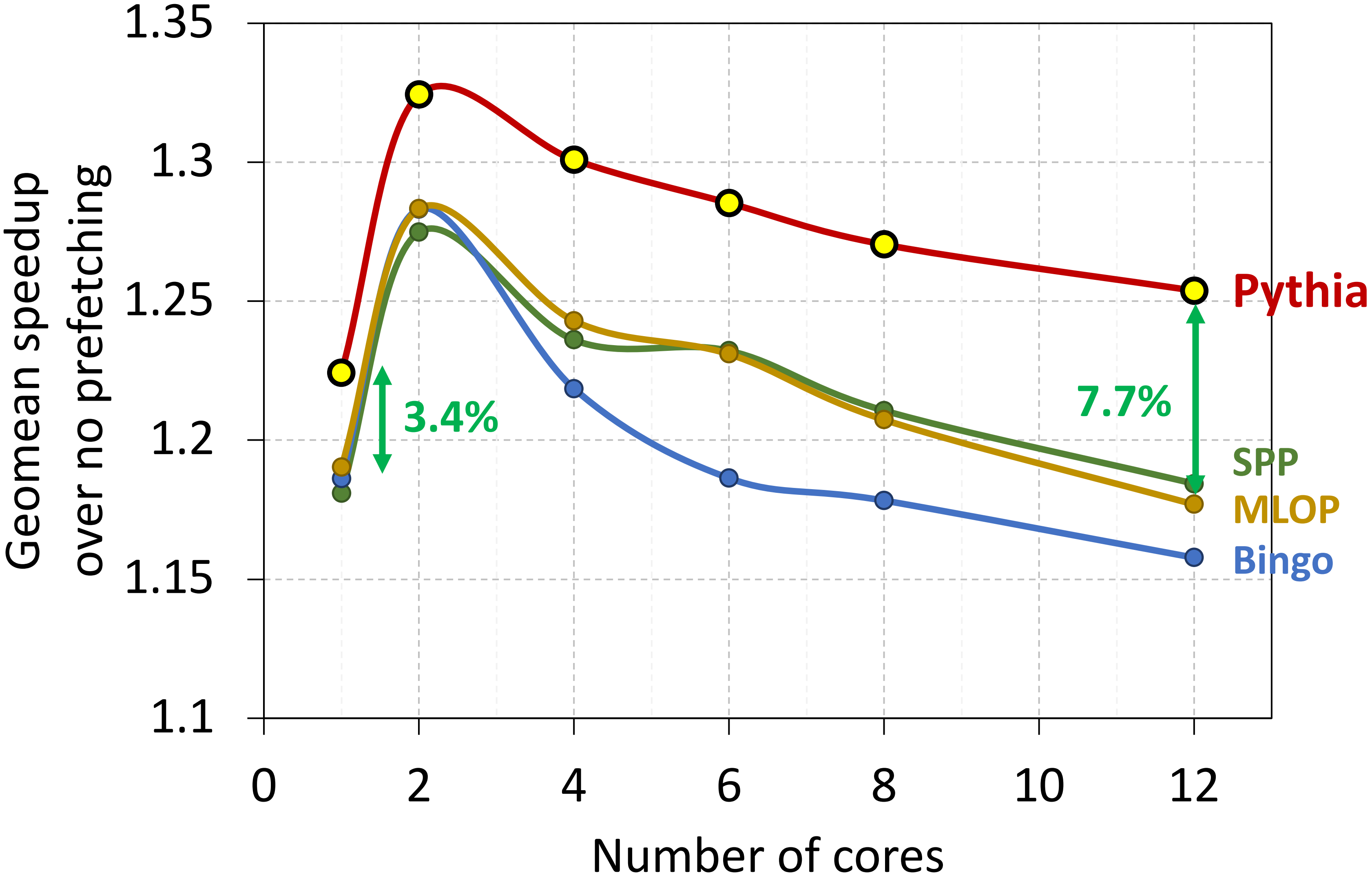
Conclusion

Simulation Methodology

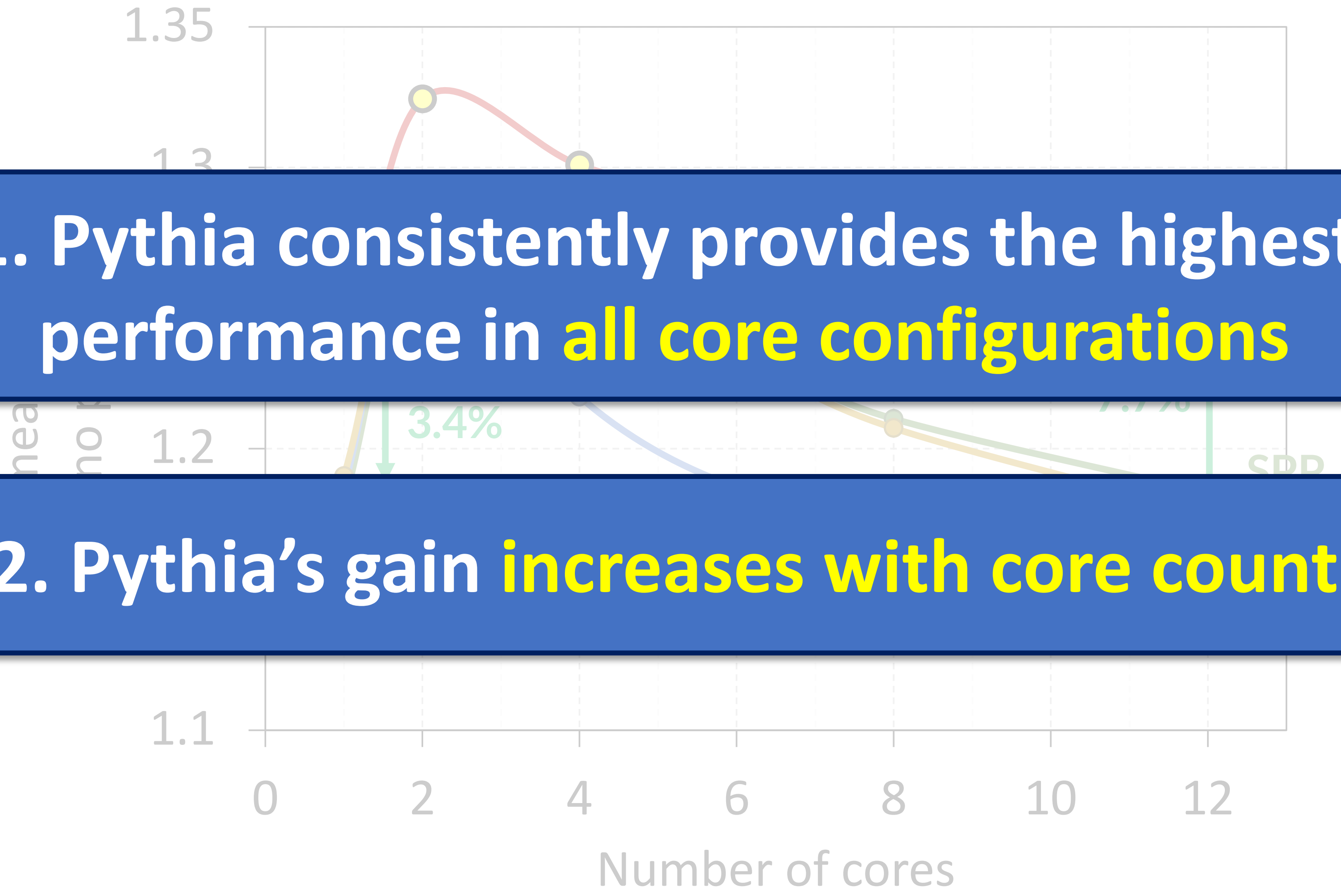
- **Champsim** [3] trace-driven simulator
- **150** single-core memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - PARSEC 2.1
  - Ligra
  - Cloudsuite
- Homogeneous and heterogeneous multi-core mixes
- **Five** state-of-the-art prefetchers
  - SPP [Kim+, MICRO'16]
  - Bingo [Bakhshalipour+, HPCA'19]
  - MLOP [Shakerinava+, 3rd Prefetching Championship, 2019]
  - SPP+DSPatch [Bera+, MICRO'19]
  - SPP+PPF [Bhatia+, ISCA'20]

[3] https://github.com/ChampSim/ChampSim

Performance with Varying Core Count



Performance with Varying Core Count

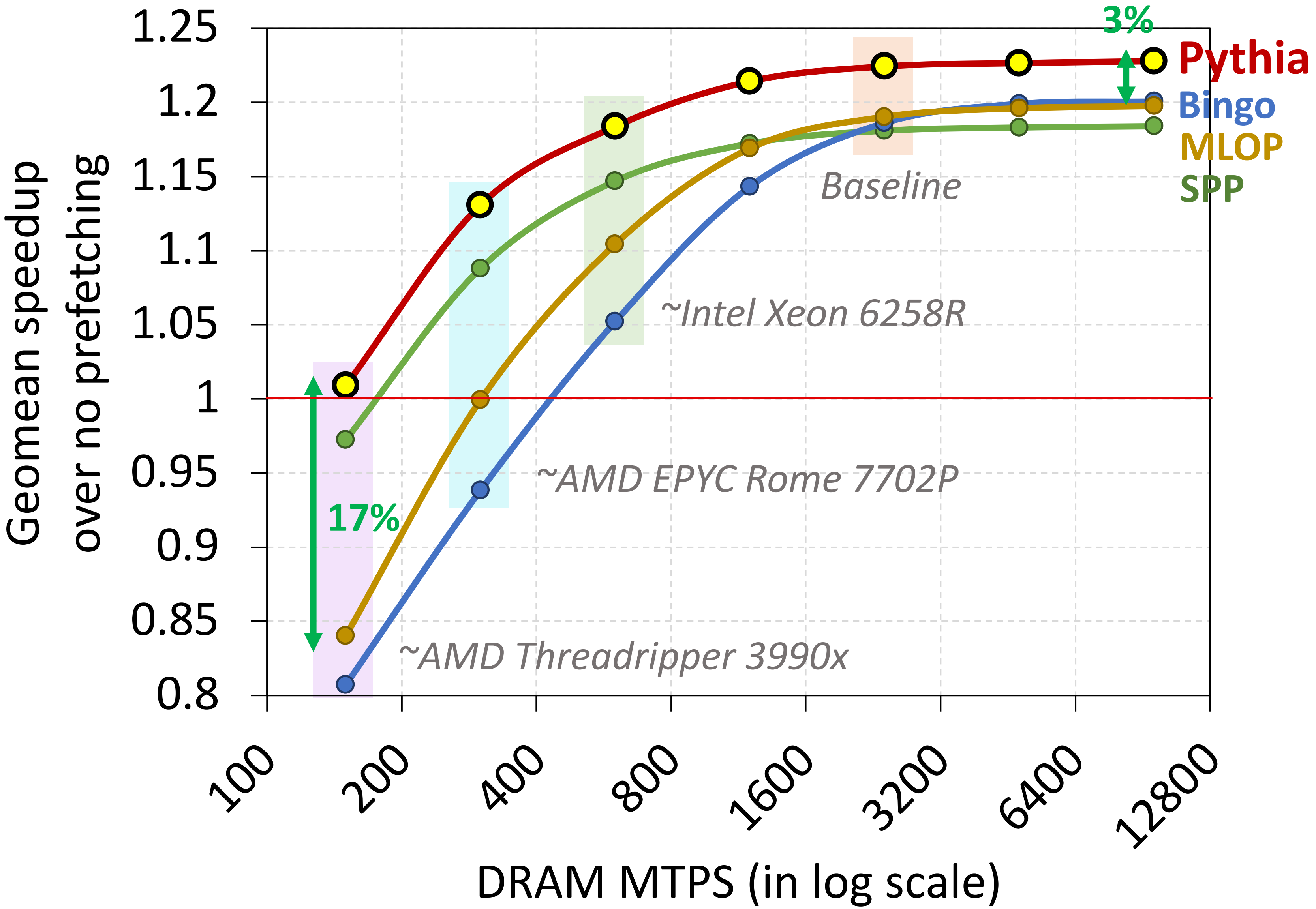


1. Pythia consistently provides the highest performance in **all core configurations**

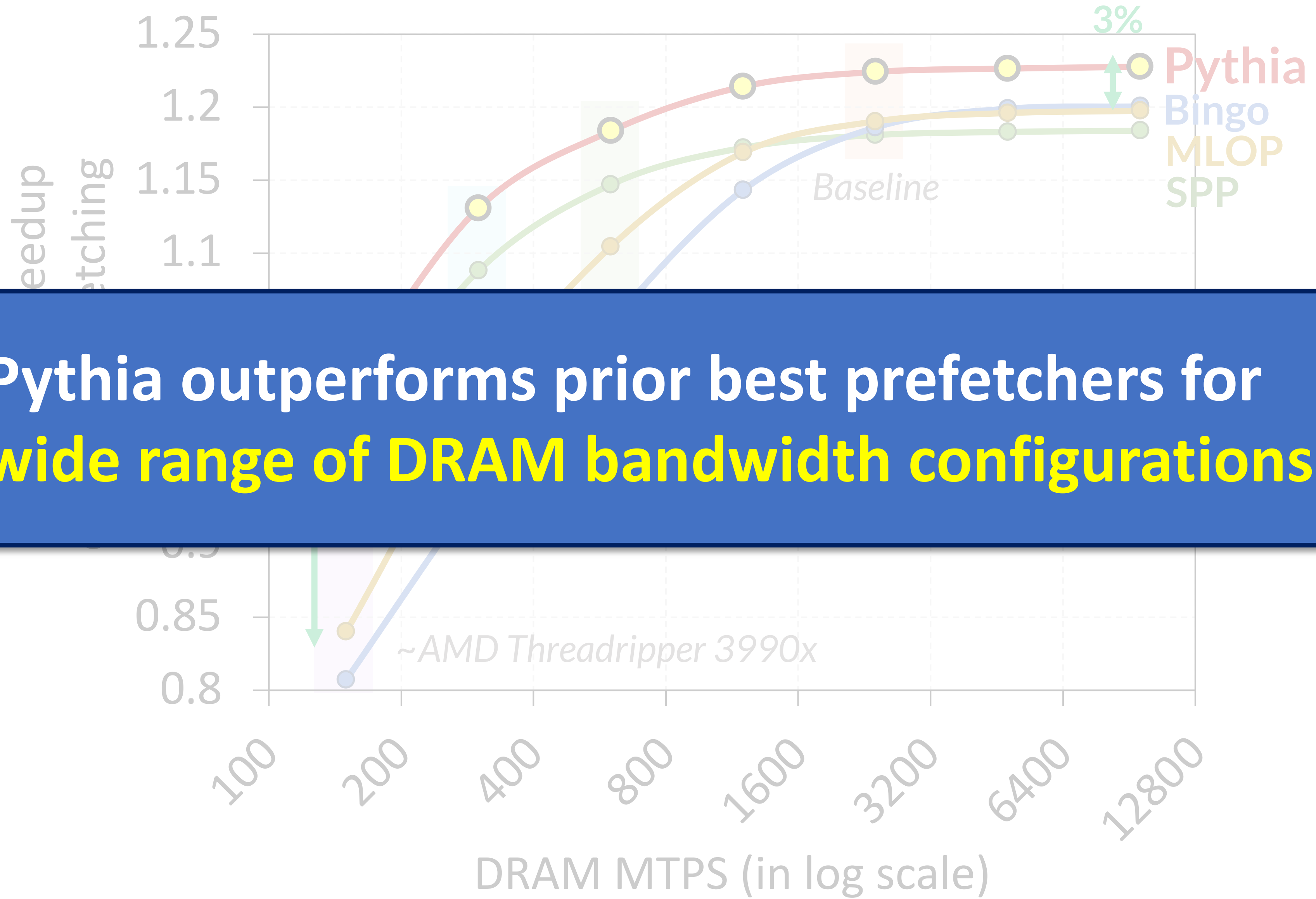
2. Pythia's gain **increases with core count**

SAFARI

Performance with Varying DRAM Bandwidth



Performance with Varying DRAM Bandwidth



Pythia outperforms prior best prefetchers for a wide range of DRAM bandwidth configurations

Pythia's Overhead

- **25.5 KB** of total metadata storage **per core**
  - Only simple tables
- We also model functionally-accurate Pythia with full complexity in **Chisel** [4] HDL

✓ **1.03% area overhead**

✓ **0.4% power overhead**

✓ **Satisfies prediction latency**

of a desktop-class 4-core Skylake processor (Xeon D2132IT, 60W)

[4] https://www.chisel-lang.org

Pythia is Open Source

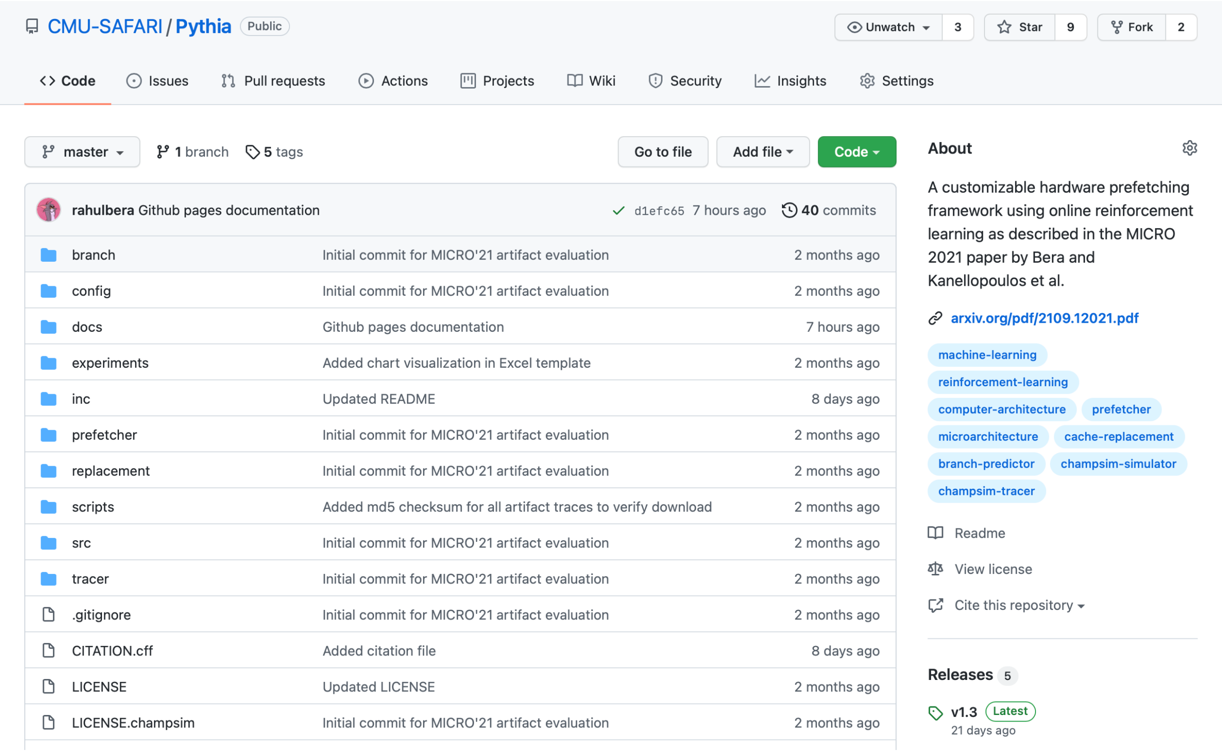
Artifacts Evaluated acm Functional V1.1

Artifacts Available acm V1.1

Results Reproduced acm V1.1

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

- MICRO'21 **artifact evaluated**
- **Champsim source** code + **Chisel** modeling code
- **All traces** used for evaluation



CMU-SAFARI / **Pythia** Public

Unwatch 3 | Star 9 | Fork 2

<> Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

master ▾ | 1 branch | 5 tags | Go to file | Add file ▾ | Code ▾

rahulbera Github pages documentation ✓ d1efc65 7 hours ago **40** commits

branch	Initial commit for MICRO'21 artifact evaluation	2 months ago
config	Initial commit for MICRO'21 artifact evaluation	2 months ago
docs	Github pages documentation	7 hours ago
experiments	Added chart visualization in Excel template	2 months ago
inc	Updated README	8 days ago
prefetcher	Initial commit for MICRO'21 artifact evaluation	2 months ago
replacement	Initial commit for MICRO'21 artifact evaluation	2 months ago
scripts	Added md5 checksum for all artifact traces to verify download	2 months ago
src	Initial commit for MICRO'21 artifact evaluation	2 months ago
tracer	Initial commit for MICRO'21 artifact evaluation	2 months ago
.gitignore	Initial commit for MICRO'21 artifact evaluation	2 months ago
CITATION.cff	Added citation file	8 days ago
LICENSE	Updated LICENSE	2 months ago
LICENSE.champsim	Initial commit for MICRO'21 artifact evaluation	2 months ago

About

A customizable hardware prefetching framework using online reinforcement learning as described in the MICRO 2021 paper by Bera and Kanellopoulos et al.

arxiv.org/pdf/2109.12021.pdf

machine-learning | reinforcement-learning | computer-architecture | prefetcher | microarchitecture | cache-replacement | branch-predictor | champsim-simulator | champsim-tracer

Readme

View license

Cite this repository ▾

Releases 5

v1.3 Latest
21 days ago

Talk Outline

Key Shortcomings of Prior Prefetchers

Formulating Prefetching as Reinforcement Learning

Pythia: Overview

Evaluation of Pythia and Key Results

Conclusion

Conclusion

- **Background**: Prefetchers predict addresses of future memory requests by associating memory access patterns with program context (called **feature**)

- **Problem**: Three key shortcomings of prior prefetchers:
  - Predict mainly using a **single program feature**
  - Lack **inherent system awareness** (e.g., memory bandwidth usage)
  - Lack **in-silicon customizability**

- **Goal**: Design a prefetching framework that:
  - Learns from **multiple features** and **inherent system-level feedback**
  - Can be **customized in silicon** to use different features and/or prefetching objectives

- **Contribution**: Pythia, which formulates prefetching as reinforcement learning problem
  - Takes **adaptive** prefetch decisions using multiple features and system-level feedback
  - Can be **customized in silicon** for target workloads via simple configuration registers
  - Proposes **a realistic and practical** implementation of RL algorithm in hardware

- **Key Results**:
  - Evaluated using a wide range of workloads from SPEC CPU, PARSEC, Ligra, Cloudsuite
  - Outperforms best prefetcher (in 1-core config.) by **3.4%, 7.7% and 17%** in 1/4/bw-constrained cores
  - Up to **7.8% more performance** over basic Pythia across Ligra workloads via simple customization

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

acm Artifacts Evaluated Functional V1.1

acm Artifacts Available V1.1

acm Results Reproduced V1.1

Pythia

A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori,
Taha Shahroodi, Sreenivas Subramoney, Onur Mutlu

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

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

TU Delft

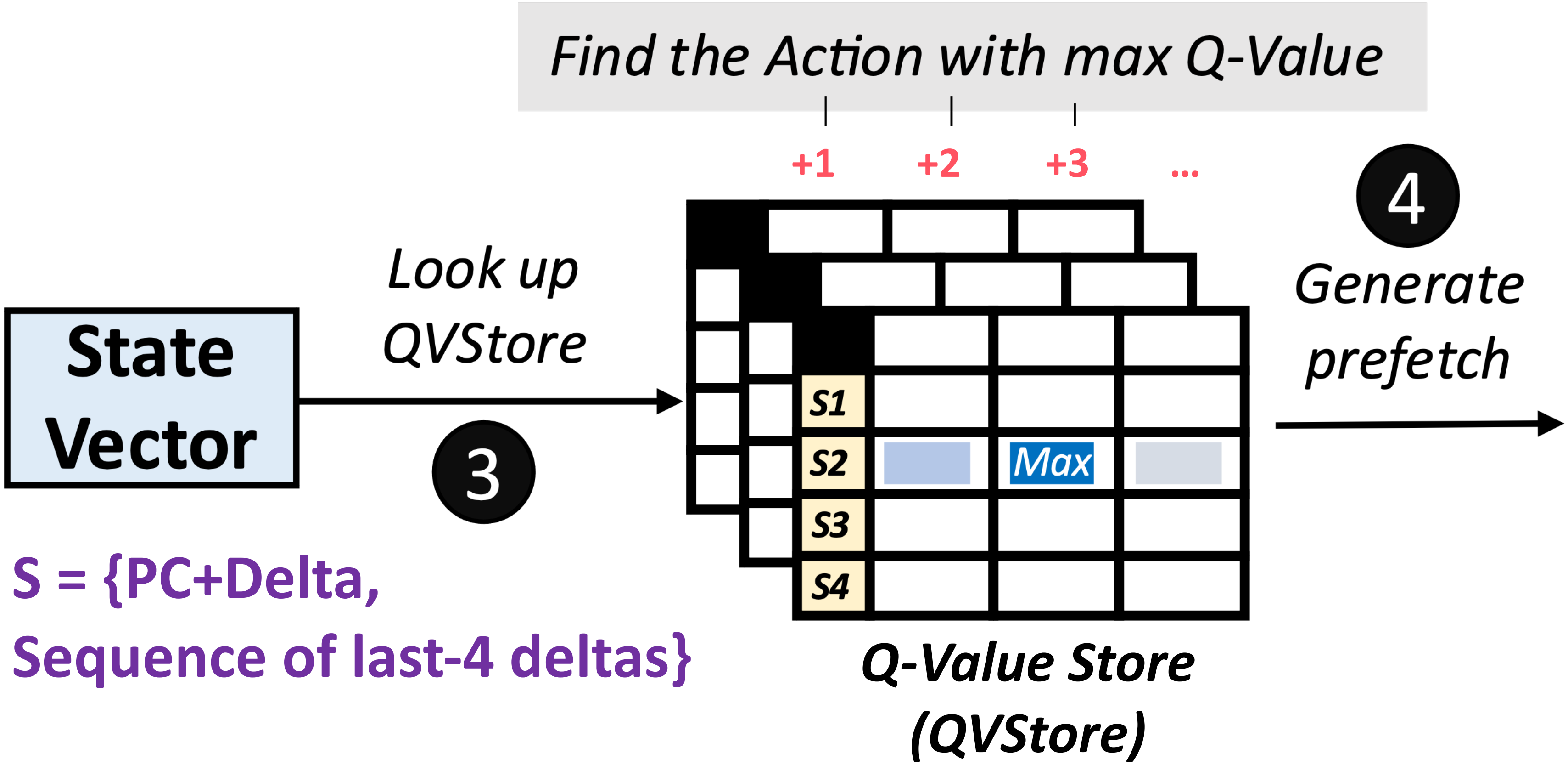
BACKUP

Why RL? Why Not Supervised Learning?

- Determining the **benefits of prefetching** (i.e., whether a decision was good for performance or not) is **not easy**
  - **Depends on a complex set of metrics**
    - Coverage, accuracy, timeliness
    - Effects on system: b/w usage, pollution, cross-application interference, …
  - **Dynamically-changing environmental conditions** change the benefit
  - **Delayed feedback due to long latency** (might not receive feedback at all for inaccurate prefetches!)

- Differs from classification tasks (e.g., branch prediction)
  - Performance strongly correlates mainly to accuracy
  - Does not depend on environment
  - Bounded feedback delay

Architecting QVStore



Find the Action with max Q-Value

State Vector

Look up QVStore

3

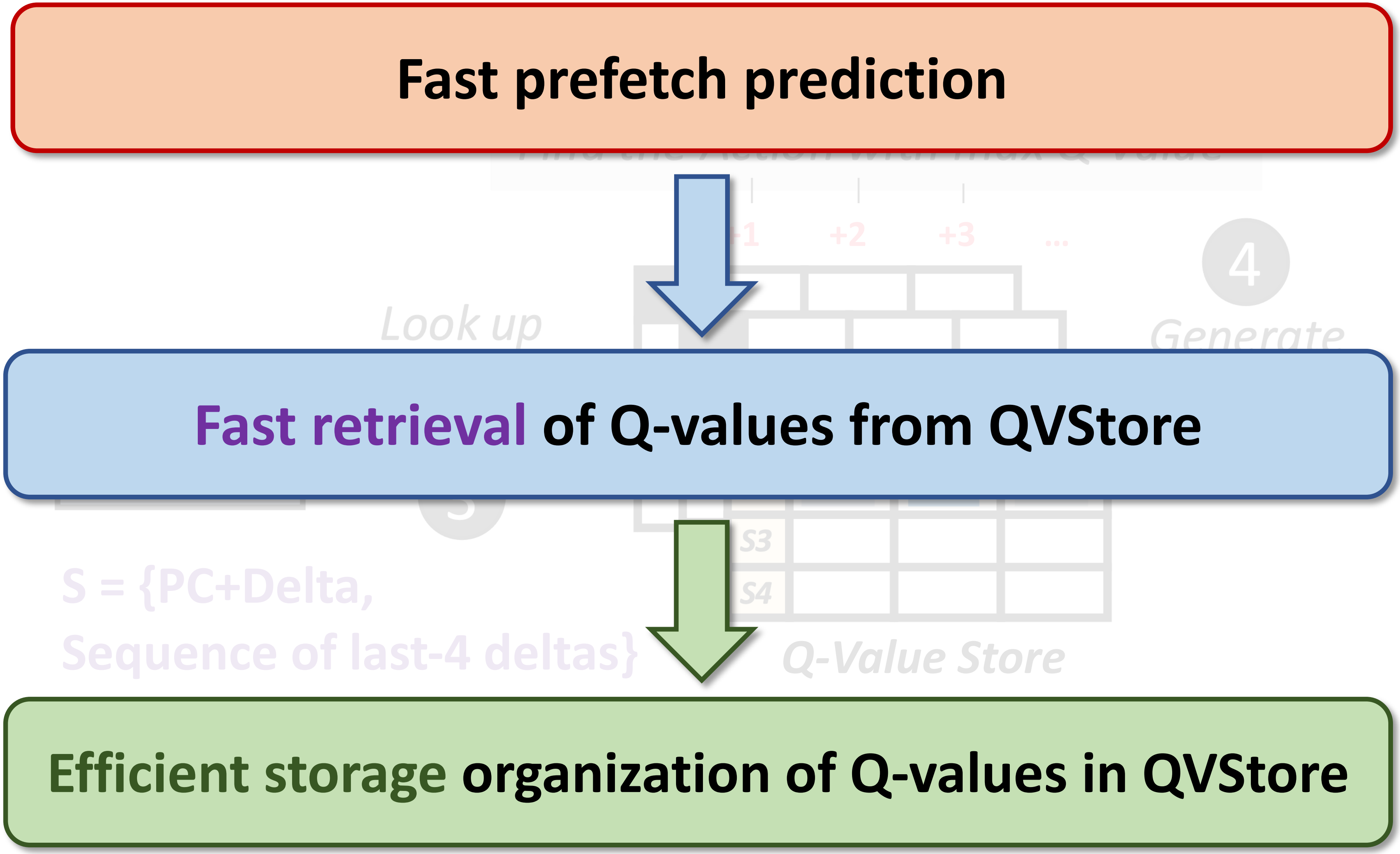
S = {PC+Delta,
Sequence of last-4 deltas}

Q-Value Store (QVStore)

4

Generate prefetch

Architecting QVStore



Fast prefetch prediction

Fast retrieval of Q-values from QVStore

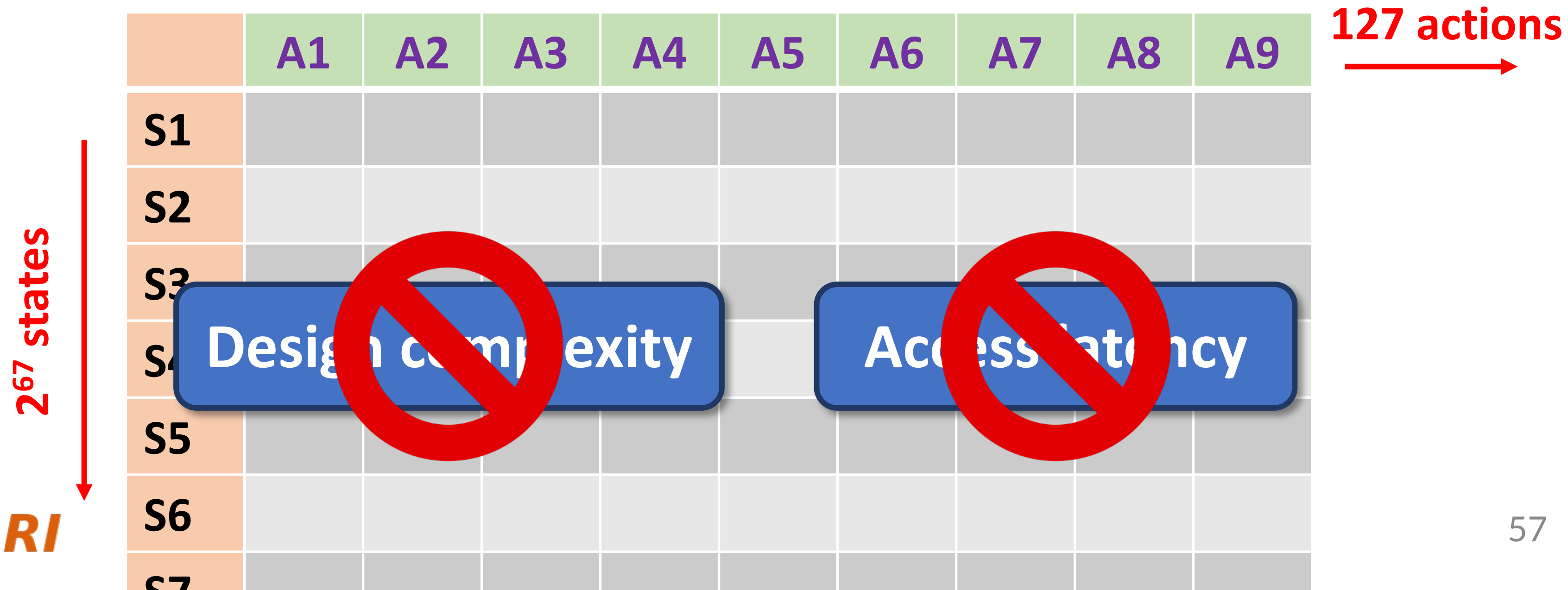
Efficient storage organization of Q-values in QVStore

Organization of QVStore

- A **monolithic** two-dimensional table?
  - Indexed by state and action values
- State-space increases **exponentially** with #bits

S = {PC+Delta, Sequence of last-4 deltas}

32b + 7b + 4x7b **= 67 bits**



	A1	A2	A3	A4	A5	A6	A7	A8	A9
S1									
S2									
S3									
S4									
S5									
S6									
S7									

127 actions

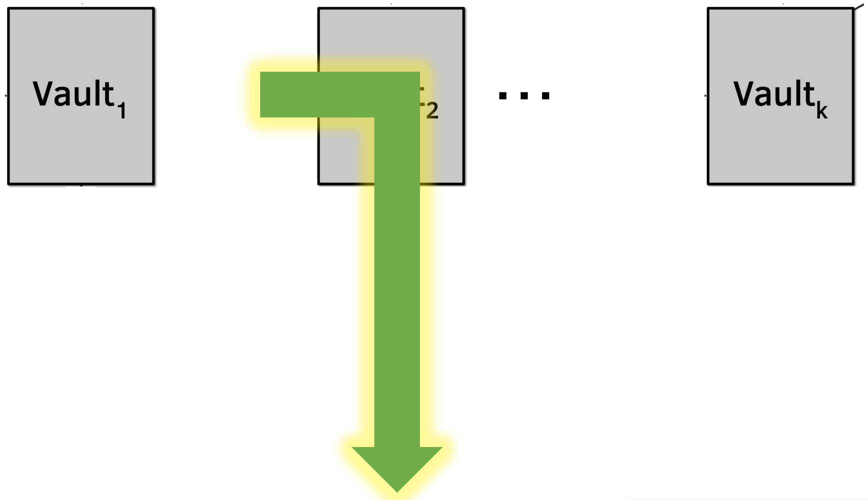
2^{67} states

Design complexity

Access latency

Organization of QVStore

- We partition QVStore into ***k* vaults** [*k* = number of features in state]
    - Each vault corresponds to one feature and stores the Q-values of **feature-action pairs**



To retrieve Q(S,A) for each action

- Query **each vault in parallel** with feature and action
- **Retrieve feature-action Q-value** from each vault
- Compute **MAX** of all feature-action Q-values

MAX ensures the Q(S,A) is driven by the constituent feature that has highest Q(ϕ,A)

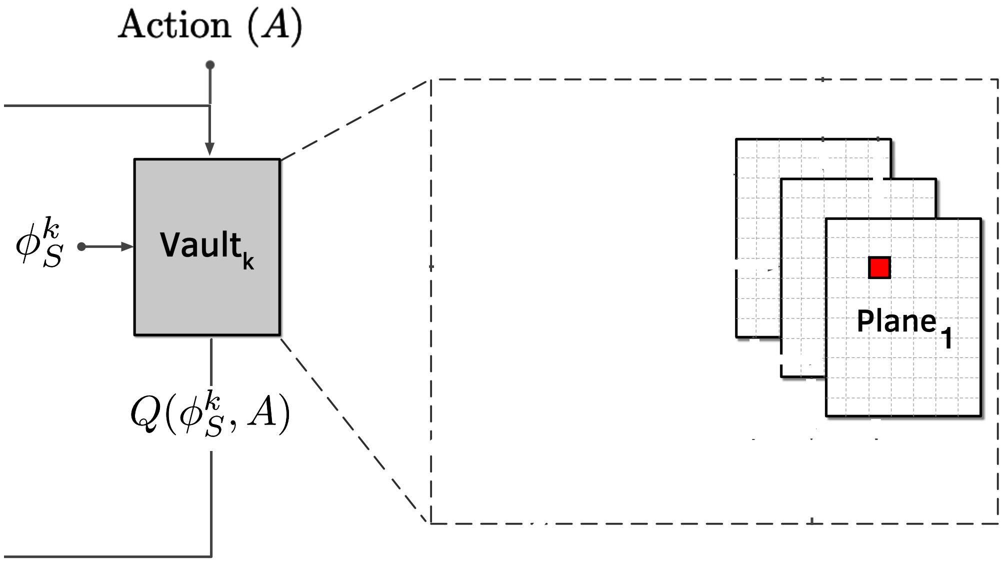
SAFARI

Organization of QVStore

- We further partition each vault into multiple **planes**
  - Each plane stores a **partial** Q-value of a feature-action pair

To retrieve Q(ϕ,A) for each action

- Query **each plane in parallel** with hashed feature and action
- **Retrieve partial feature-action Q-value** from each plane
- Compute **SUM** of all parital feature-action Q-values

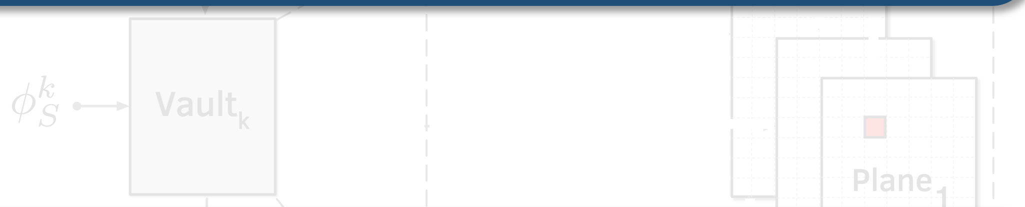


Organization of QVStore

- We further partition each vault into multiple **planes**
  - Each plane stores a **partial** Q-value of a feature-action pair

1. **Enables sharing** of partial Q-values between **similar feature values**, shortens prefetcher training time

- Query **each plane in parallel** with hashed feature and action

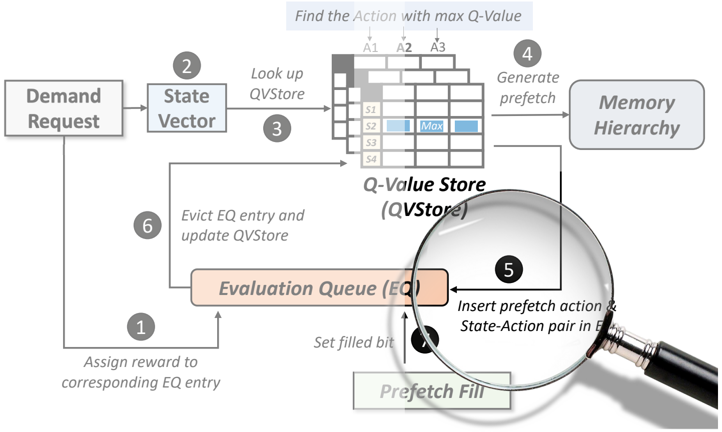


2. **Reduces chances** of sharing partial Q-values across widely **different feature values**

- Compute **SUM** of all partial feature-action Q-values

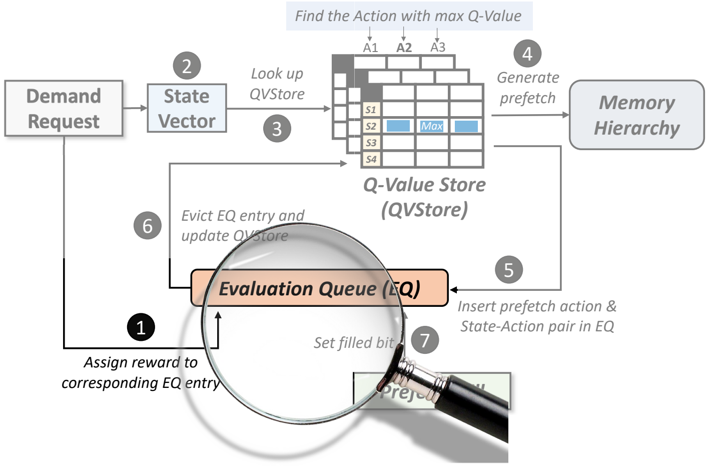
Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction
- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch



Reward Assignment to EQ Entry

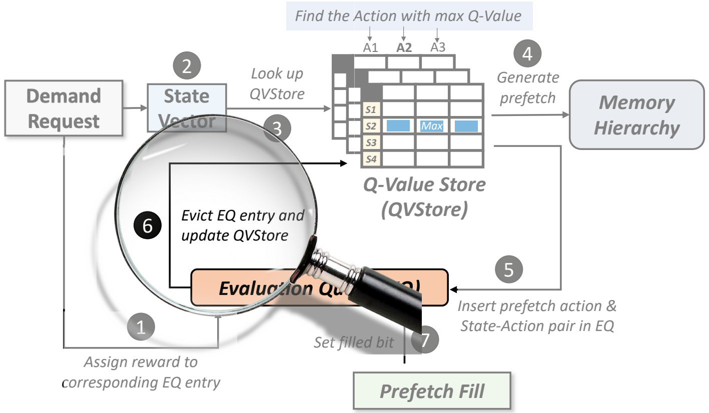
- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction



- **During EQ insertion**: for actions
  - Not to prefetch
  - Out-of-page prefetch
- **During EQ residency**:
  - In case address of a demand matches with address in EQ (*signifies accurate prefetch*)

Reward Assignment to EQ Entry

- **Every** action gets inserted into EQ
- Reward is assigned to each EQ entry **before or during** the eviction



Find the Action with max Q-Value

A1 A2 A3

Demand Request

2 State Vector

Look up QVStore

3

S1 S2 S3 S4

Max

Q-Value Store (QVStore)

4 Generate prefetch

Memory Hierarchy

6 Evict EQ entry and update QVStore

Evaluation Qu

5 Insert prefetch action & State-Action pair in EQ

1 Assign reward to corresponding EQ entry

Set filled bit 7

Prefetch Fill

- **During EQ insertion:** for actions
  - Not to prefetch
  - Out-of-page prefetch
- **During EQ residency:**
  - In case address of a demand matches with address in EQ (*signifies accurate prefetch*)
- **During EQ eviction:**
  - In case no reward is assigned till eviction (*signifies inaccurate prefetch*)

Basic Pythia Configuration

- Derived from **automatic design-space exploration**

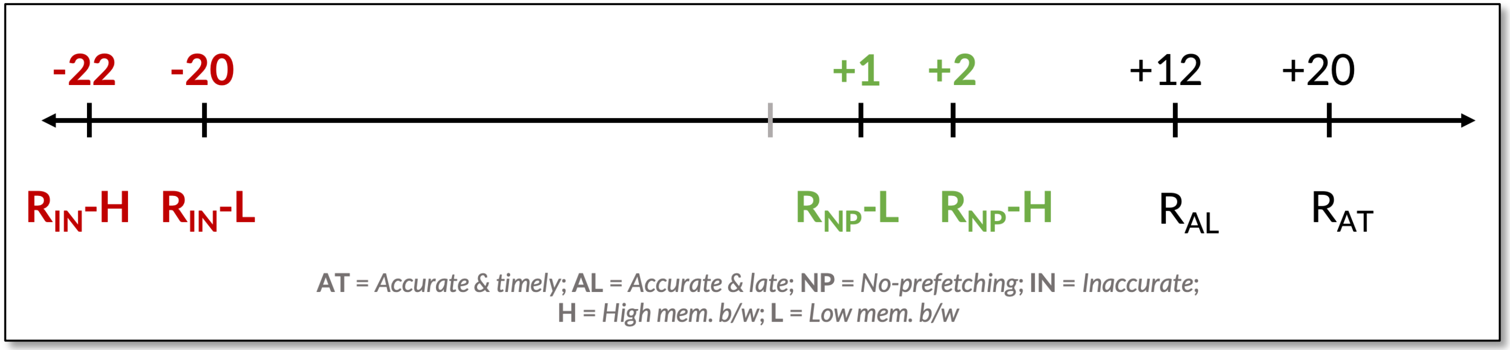
- **State:** 2 features
  - PC+Delta
  - Sequence of last-4 deltas

- **Actions:** 16 prefetch offsets
  - Ranging between -6 to +32. Including 0.

- **Rewards:**
  - R_{AT} = +20; R_{AL} = +12; R_{NP}-H=-2; R_{NP}-L=-4;
  - R_{IN}-H=-14; R_{IN}-L=-8; R_{CL}=-12

Performance Improvement via Customization

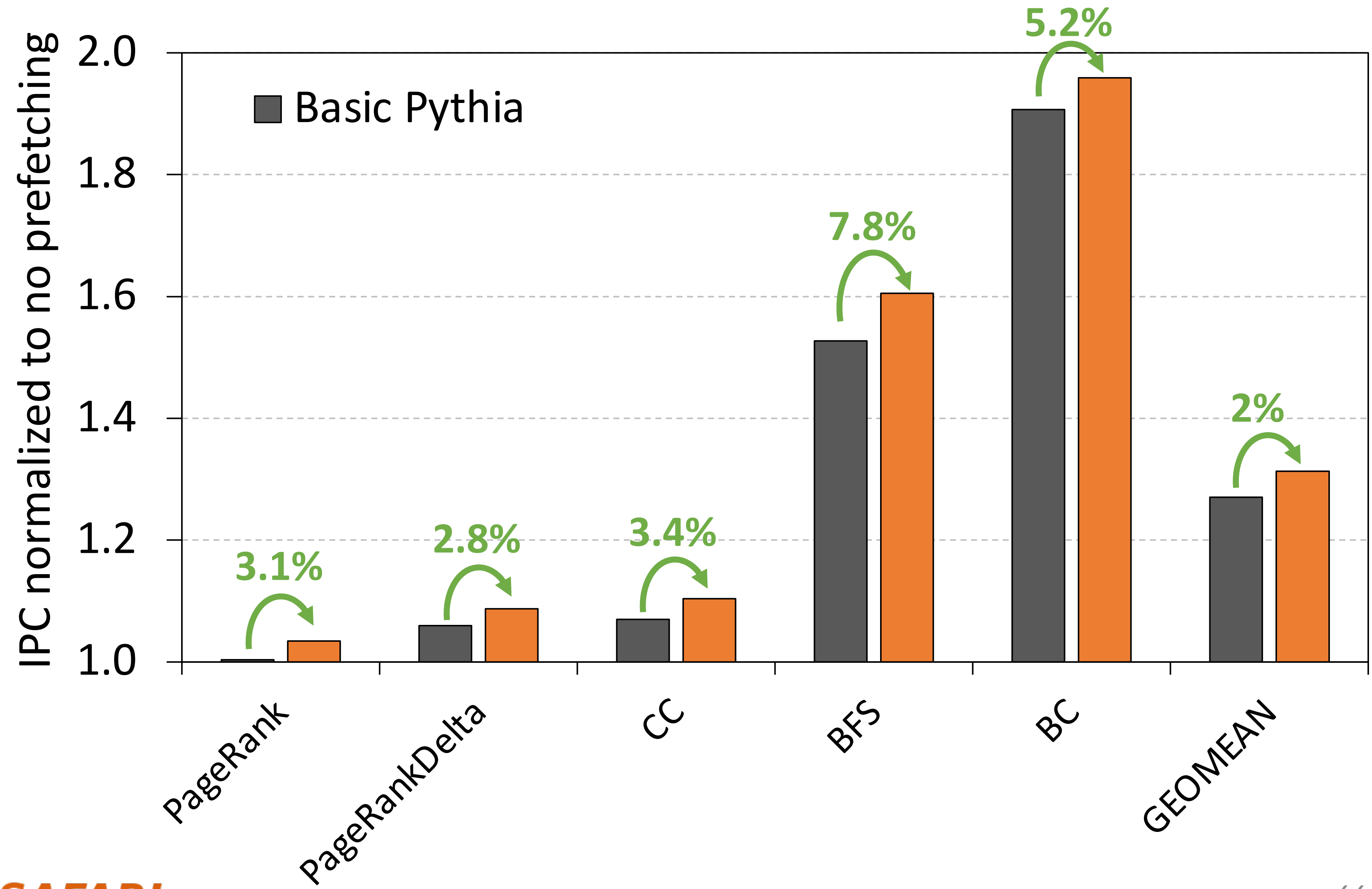
- Reward value customization
- **Strict Pythia configuration**
  - **Increasing** the rewards for **no prefetching**
  - **Decreasing** the rewards for **inaccurate prefetching**



AT = *Accurate & timely*; **AL** = *Accurate & late*; **NP** = *No-prefetching*; **IN** = *Inaccurate*;
H = *High mem. b/w*; **L** = *Low mem. b/w*

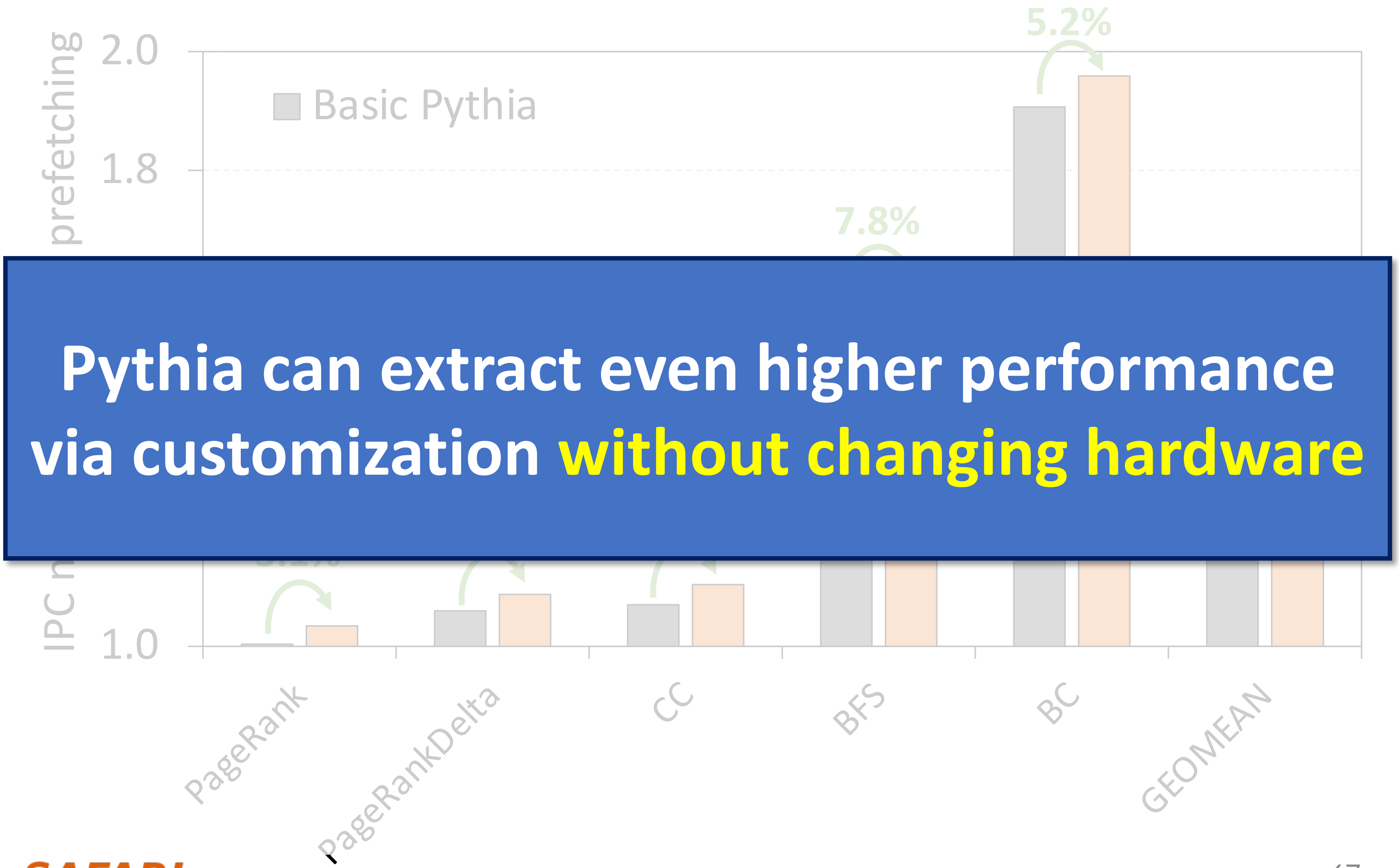
- Strict Pythia is **more conservative** in generating prefetch requests than the basic Pythia
- Evaluate on all **Ligra graph processing workloads**

Performance Improvement via Customization



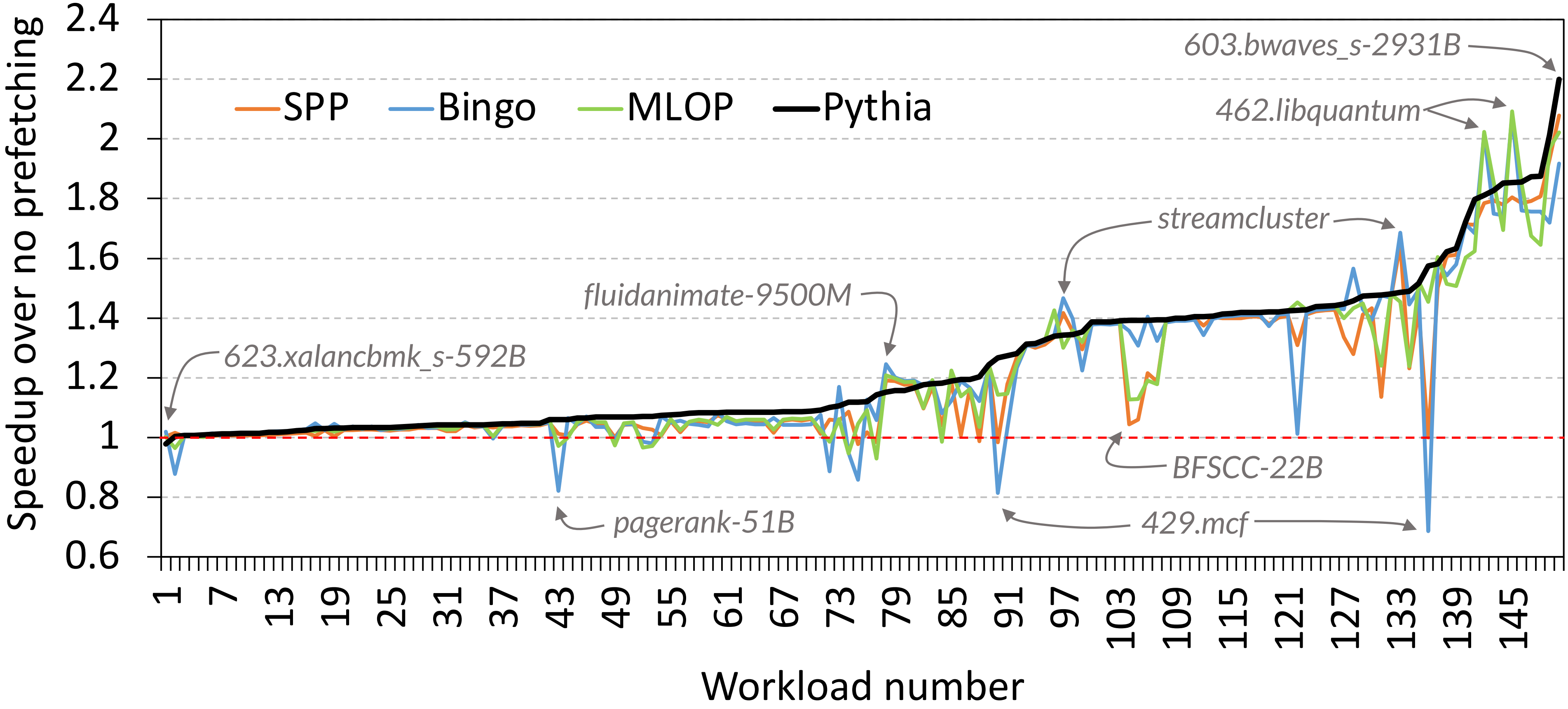
SAFARI

Performance Improvement via Customization



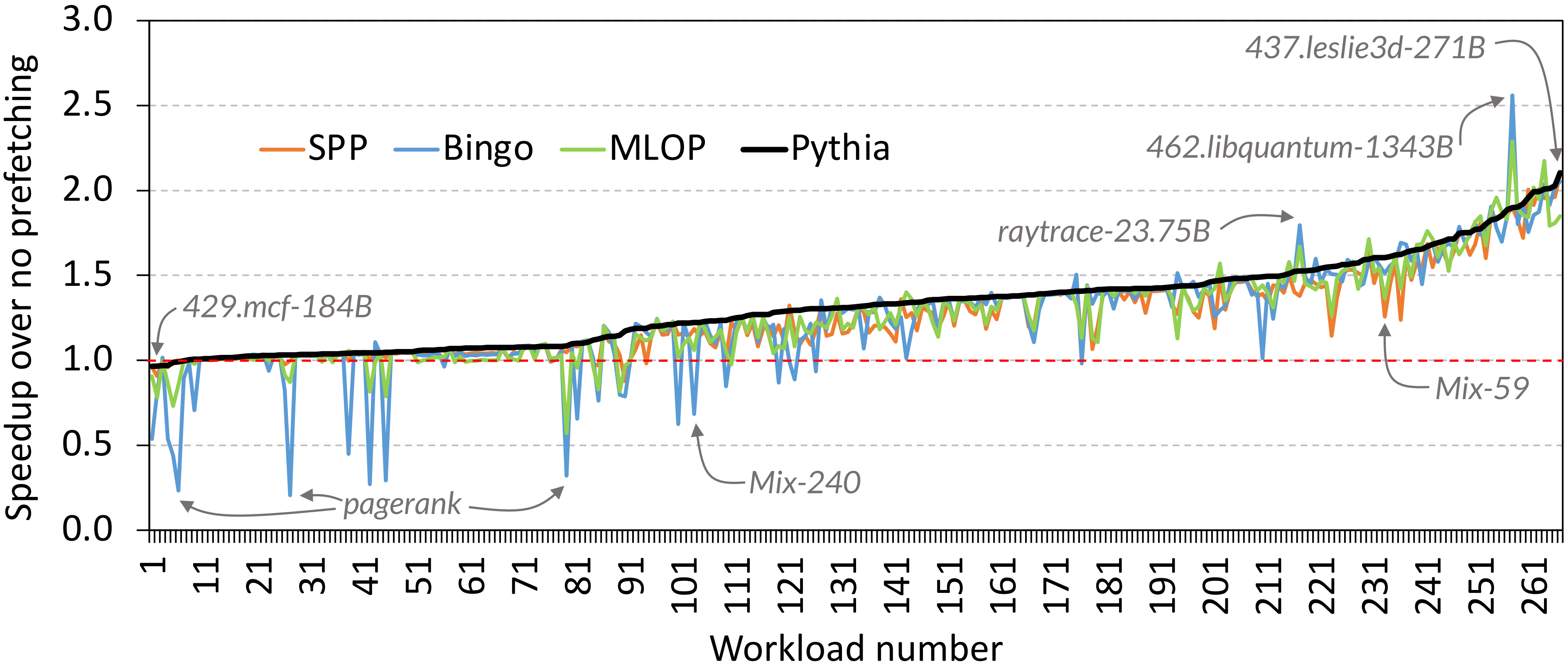
Pythia can extract even higher performance via customization without changing hardware

Performance S-curve: Single-core



SAFARI

Performance S-curve: Four-core



SAFARI

More in the Paper

- Performance comparison with **unseen traces**
  - Pythia provides **equally high** performance benefits

- Comparison against **multi-level prefetchers**
  - Pythia **outperforms** prior best multi-level prefetchers

- Understanding Pythia's learning with **a case study**
  - We reason towards **the correctness** of Pythia's decision

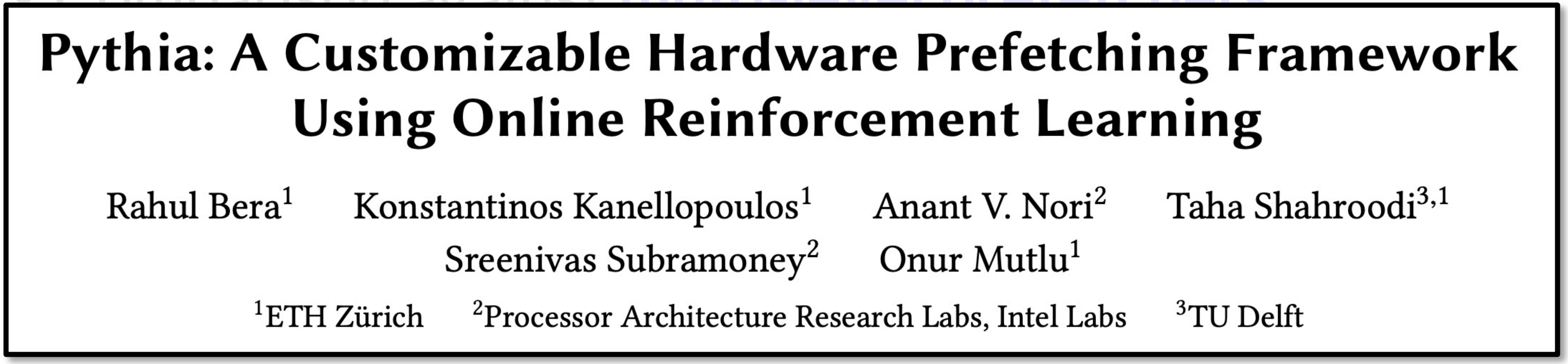
- **Performance sensitivity** towards different features and hyperparameter values

- Detailed single-core and four-core performance

More in the Paper

- Performance comparison with **unseen traces**
  - Pythia provides equally high performance benefits

- Comparison against **multi-level prefetchers**



Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning

Rahul Bera[1] Konstantinos Kanellopoulos[1] Anant V. Nori[2] Taha Shahroodi[3,1]
Sreenivas Subramoney[2] Onur Mutlu[1]
[1]ETH Zürich [2]Processor Architecture Research Labs, Intel Labs [3]TU Delft

https://arxiv.org/pdf/2109.12021.pdf

- **Performance sensitivity** towards different features and hyperparameter values

- Detailed single-core and four-core performance