CHROME: Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning

Xiaoyang Lu, Hamed Najafi, Jason Liu, Xian-He Sun

ILLINOIS TECH

FIU FLORIDA INTERNATIONAL UNIVERSITY

Cache Management

Cache Management: Essential for bridging the performance gap between fast CPU and slower main memory

Cache Replacement

- Determines which cache blocks to evict when new data needs to be loaded

Cache Bypassing

- Decides whether incoming data should be stored in the cache

Prefetching

- Predictively loads data into the cache before it is actually requested by the CPU

Examples of Cache Management Schemes

- Hawkeye [ISCA'16]
  - Cache replacement based on reuse prediction
  - Formulated as a binary classification problem (cache friendly or cache averse)
  - PC based predictor
- Glider [MICRO'19]
  - Cache replacement
  - Use LSTM for offline training and SVM for online decision making
- Mockingjay [HPCA'22]
  - Holistic approach encompassing cache replacement and bypassing, and support prefetching (with distinction between demand and prefetch accesses)
  - Extends Hawkeye -> multiclass reuse prediction
  - Policies are statically designed based on fixed assumptions
- CARE [HPCA'23]
  - Cache replacement considering both data locality and access concurrency

Limitations of Current Cache Management Schemes

We observe there are **two common limitations** faced by these cache management techniques:

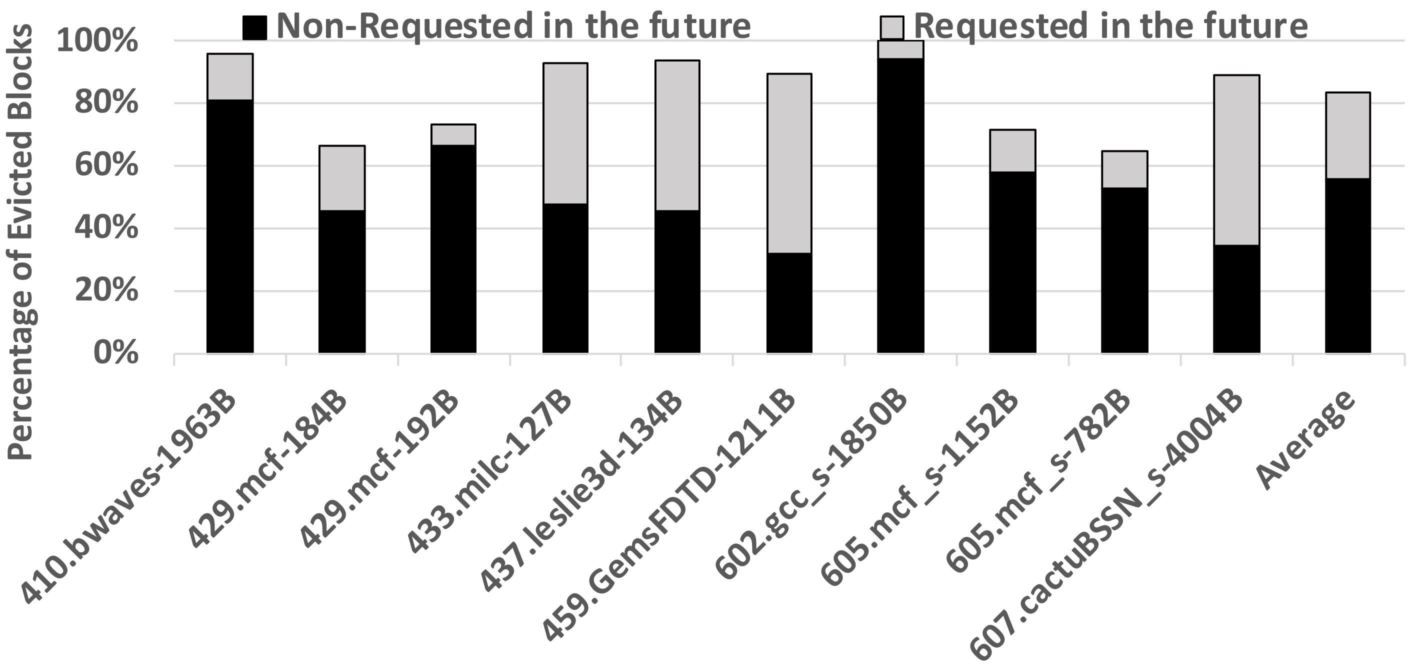
1 **Lack of Holistic View**

- Current schemes often examine cache replacement, bypassing, and prefetching in isolation, overlooking the potential benefits that could arise from a joint optimization strategy

2 **Lack of Adaptability**

- Current schemes often rely on fixed heuristics that don't account for the changing access patterns of modern applications and system configurations

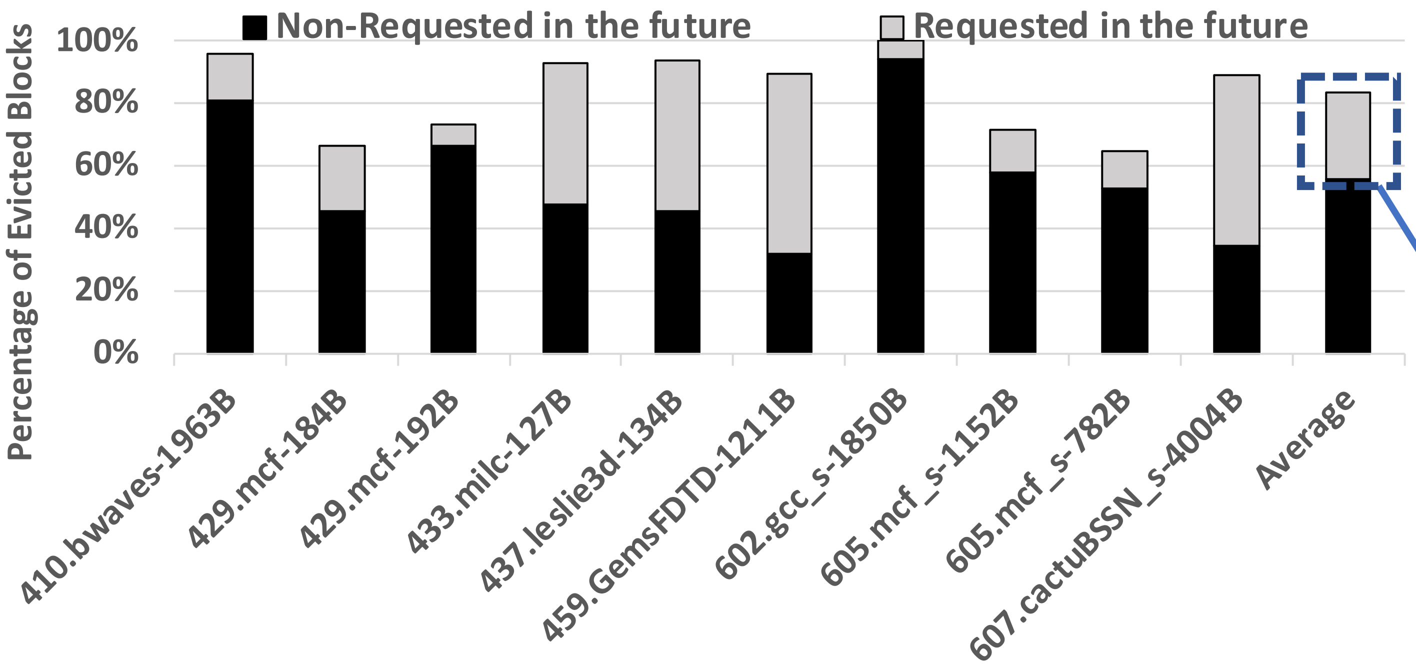
Lack of Holistic View



83.7% of evicted blocks in shared LLC are **not reused before eviction**;

Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO'19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

Lack of Holistic View

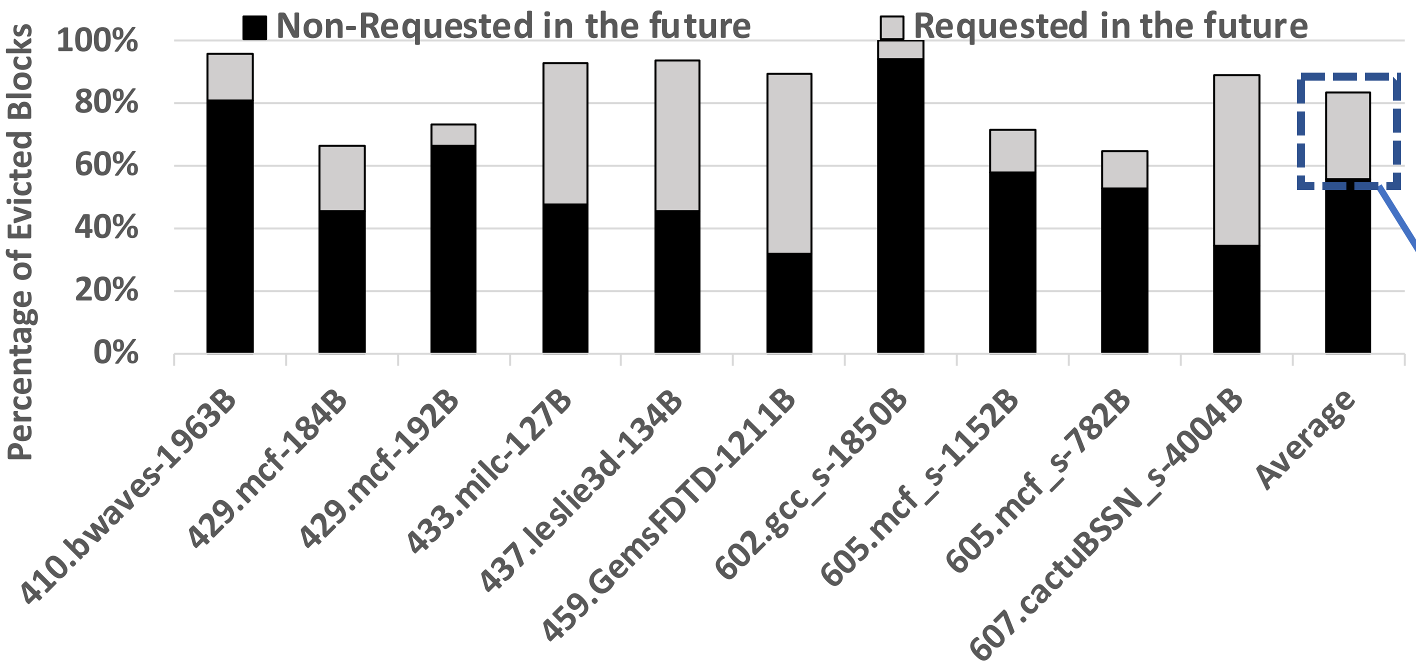


83.7% of evicted blocks in shared LLC are **not reused before eviction**;

28.0% of evicted blocks are not reused before eviction, but are **requested again** in the future

Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO'19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

Lack of Holistic View



Inspecting Unresued Blocks in LLC with Gilder management scheme [MICRO'19]. Next-line prefetcher is used at L1 and stride prefetcher is used at L2.

83.7% of evicted blocks in shared LLC are **not reused before eviction**;

28.0% of evicted blocks are not reused before eviction, but are **requested again** in the future

70.0% of the blocks that are not reused before eviction are attributed to **prefetching**

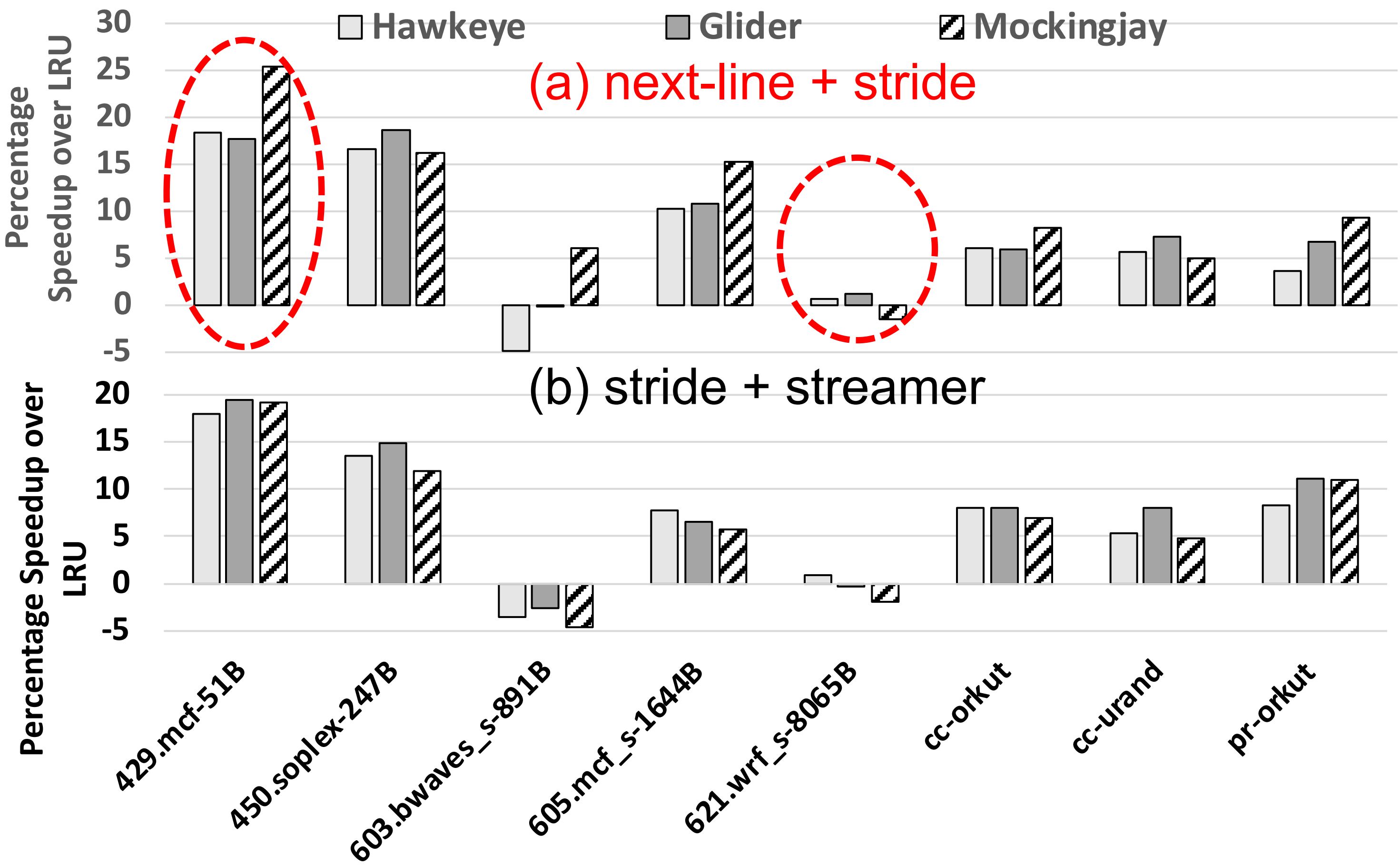
Lack of Holistic View

83.7% of evicted blocks in shared LLC are **not reused before eviction**; **70.0%** of

A holistic cache management scheme is needed:

- **Cache bypassing** can help identify blocks accessed only once
- **Cache replacement** needs to be aware of **prefetching**, to avoid the eviction of vital data

Lack of Adaptivity

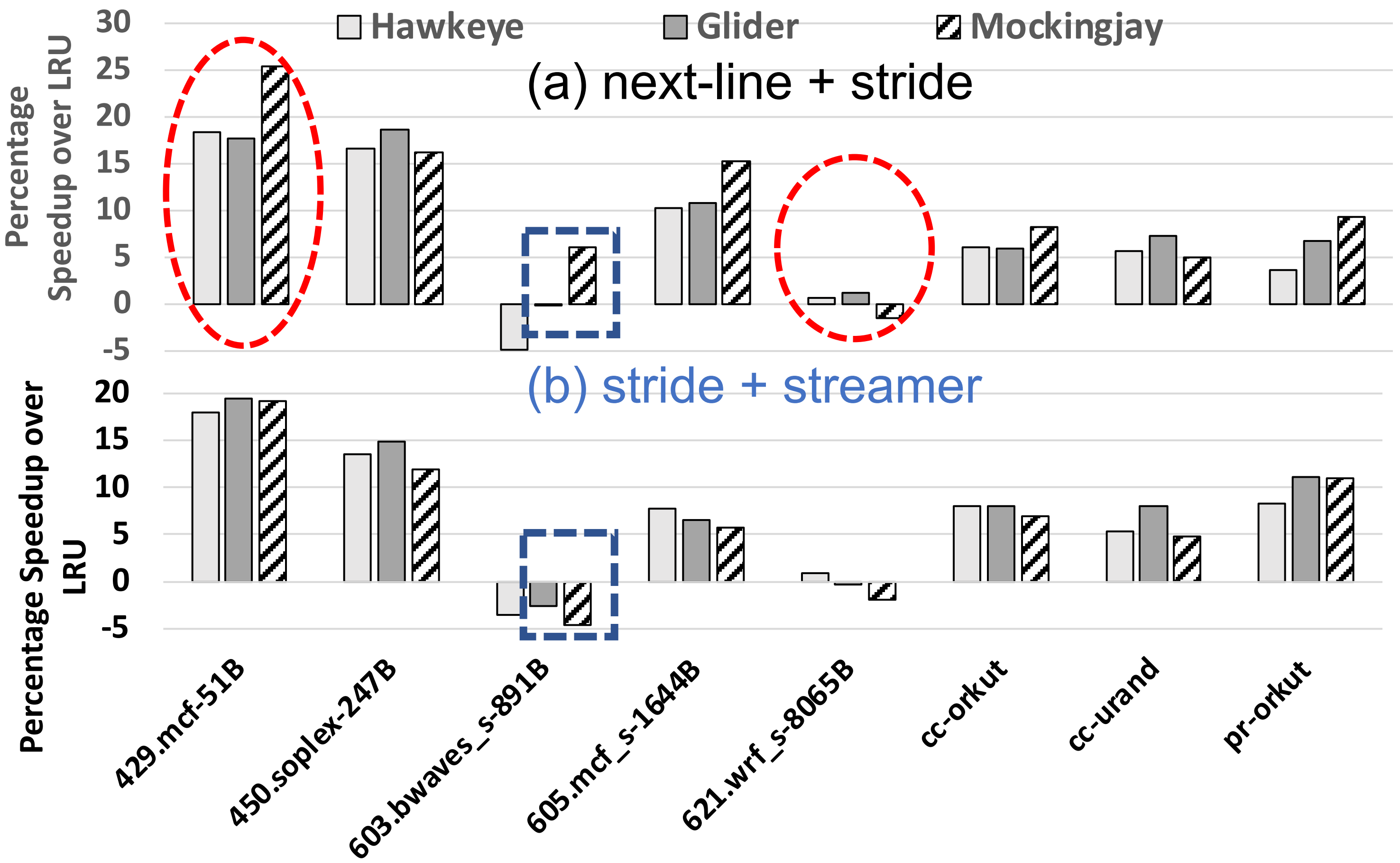


Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Three state-of-the-art cache management schemes:
Hawkeye [ISCA'16]
Glider [MICRO'19]
Mockingjay [HPCA'22]

Inconsistent performance across different workloads

Lack of Adaptivity



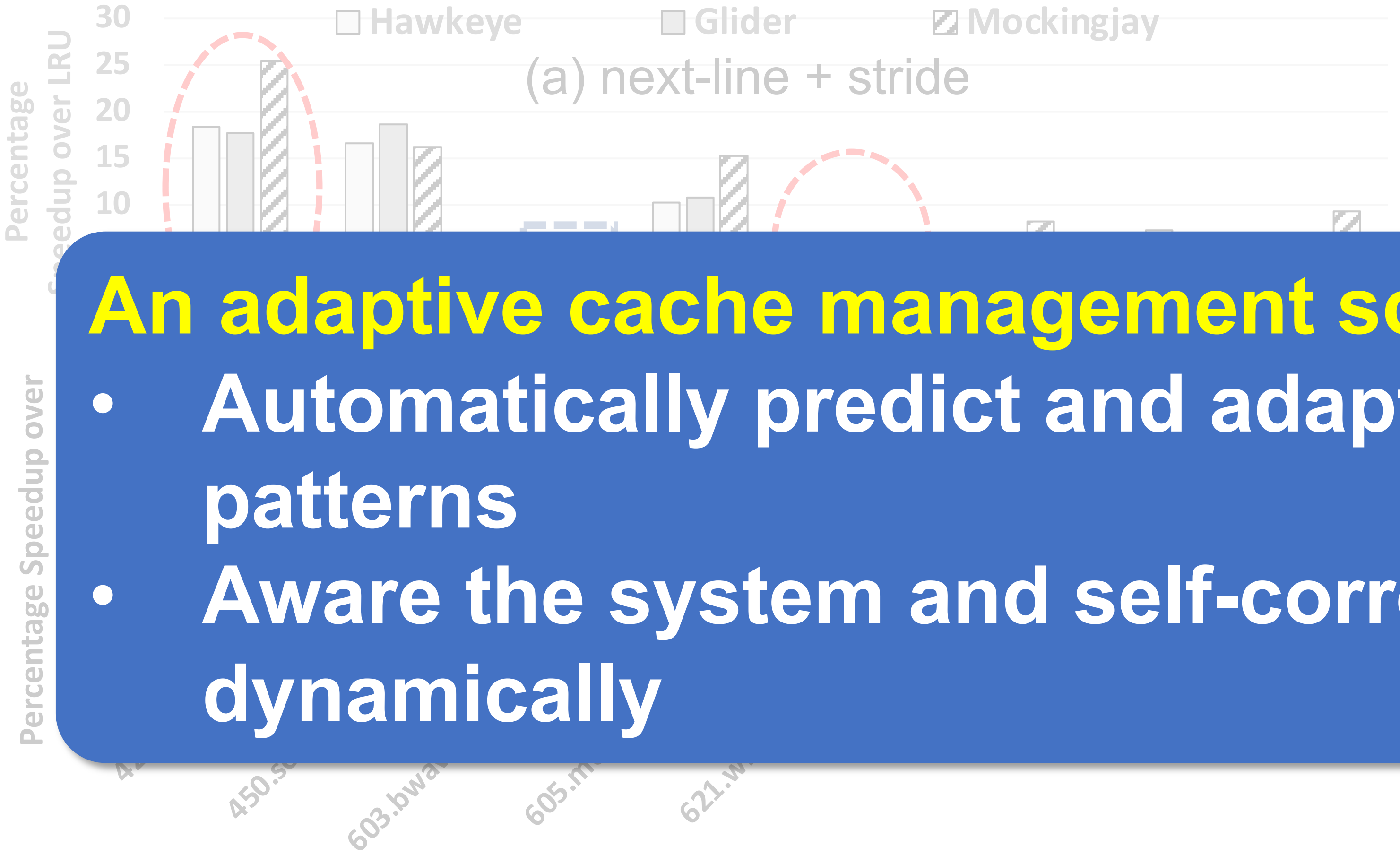
Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Three state-of-the-art cache management schemes:
Hawkeye [ISCA'16]
Glider [MICRO'19]
Mockingjay [HPCA'22]

Inconsistent performance across different workloads

Performance varies among diverse system configurations

Lack of Adaptivity



Comparing speedup over LRU on a 4-core system between: (a) using next-line prefetcher at L1 and stride prefetcher at L2, and (b) using stride prefetcher at L1 and streamer prefetcher at L2.

Inconsistent performance across different workloads

An adaptive cache management scheme is needed:

- **Automatically predict and adapt to various access patterns**
- **Aware the system and self-correct decisions dynamically**

and system configurations

Our Solution

A **holistic** cache management framework that **dynamically adapts** to various workloads and system configurations

Key Contributions: CHROME

Holistic Integration: Integrate cache bypassing and replacement with pattern-based prefetching

Dynamic Online Learning: Utilizes online reinforcement learning to adapt cache management to varying workloads and system configurations

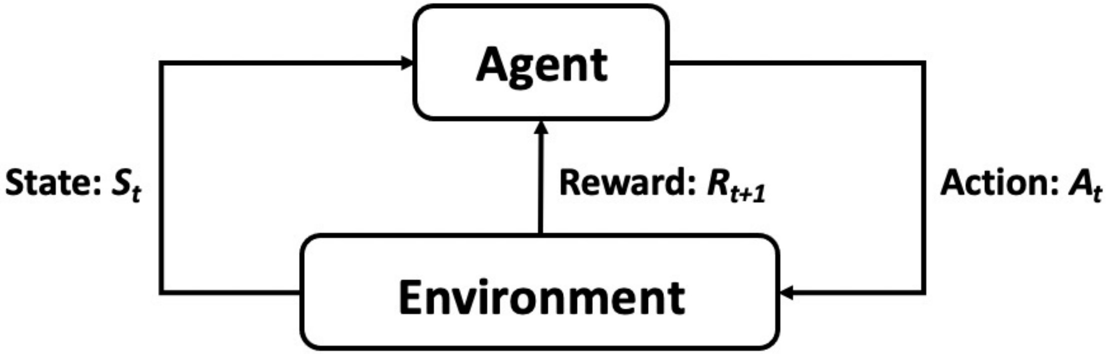
Multiple Program Features: Employs multiple program features to achieve a better understanding of memory access patterns

Concurrency-Aware Rewards: Implements a reward system that is aware of concurrent accesses, factoring in system-level feedback for decision-evaluation

Efficient Design: Achieves a minimal hardware overhead

Reinforcement Learning (RL)

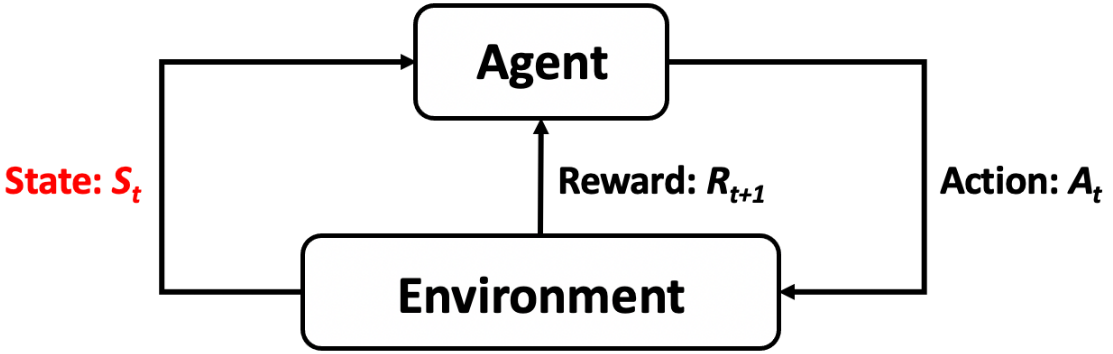
- **Autonomously** learn through **feedback** from actions and experiences in an **interactive** environment
- Agent learns to take an **action** in a given **situation** to **maximize** a numerical **reward**



- To do that, agent stores **Q-values** for **every** state-action pair
  - **Expected return** for taking an action in a state
  - Given a state, selects action that provides **highest** Q-value

Formulating Cache Management as an RL Problem

What is State?



- A vector of features for each access
- Feature selection is tradeoff between performance and overhead
- Feature: **{control-flow, data access}**
  - Control-flow of demands examples: PC, seq. of PCs, call stack, …
  - Data-access examples: memory address, page number, page offset, delta, …
  - **S = (PC signature, page number)**
- **PC signature:**
  - Hashed PC and hit/miss information
  - One bit to distinguish demand vs. prefetch access
  - Incorporate core number (to differentiate behaviors among the cores)

Formulating Cache Management as an RL Problem

What is Action?

- Eviction Priority Value (EPV)
  - Reflects the eviction priorities of the cache block
  - Three possible EPVs: low, moderate, high
- Cache miss (4 possible actions):
  - **Bypass** LLC
  - **Insert** the corresponding block in LLC with an **EPV of low, moderate, or high**
- Cache hit (3 possible actions):
  - **Update** the EPV of the corresponding block **to low, moderate, or high**

Formulating Cache Management as an RL Problem

What is Reward?

- CHROME provides 8 distinct rewards
- **Accuracy**: Reward hits and penalize misses; ALSO, for blocks not to be accessed in near future, incentivize bypass on misses or assign high EPV on hits
- **Prefetching Awareness:** Prioritize blocks likely to be requested next by demand accesses over those that might be requested by prefetch accesses
- **Concurrency-Aware System Feedback:** Identifies cores causing LLC obstruction at runtime, promoting actions that mitigate the obstruction (LLC obstruction = if C-AMAT is greater than memory access time)

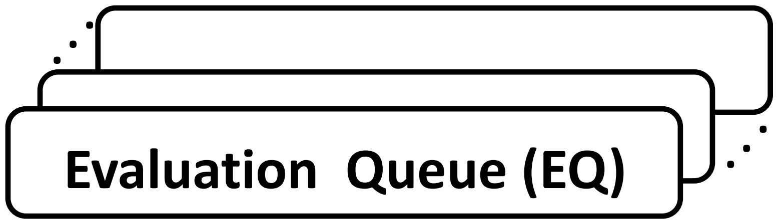
CHROME Design

Q-Table

	A1	A2	A3	A4
S1				
S2				
⋮				
Sn				

Q-Table:

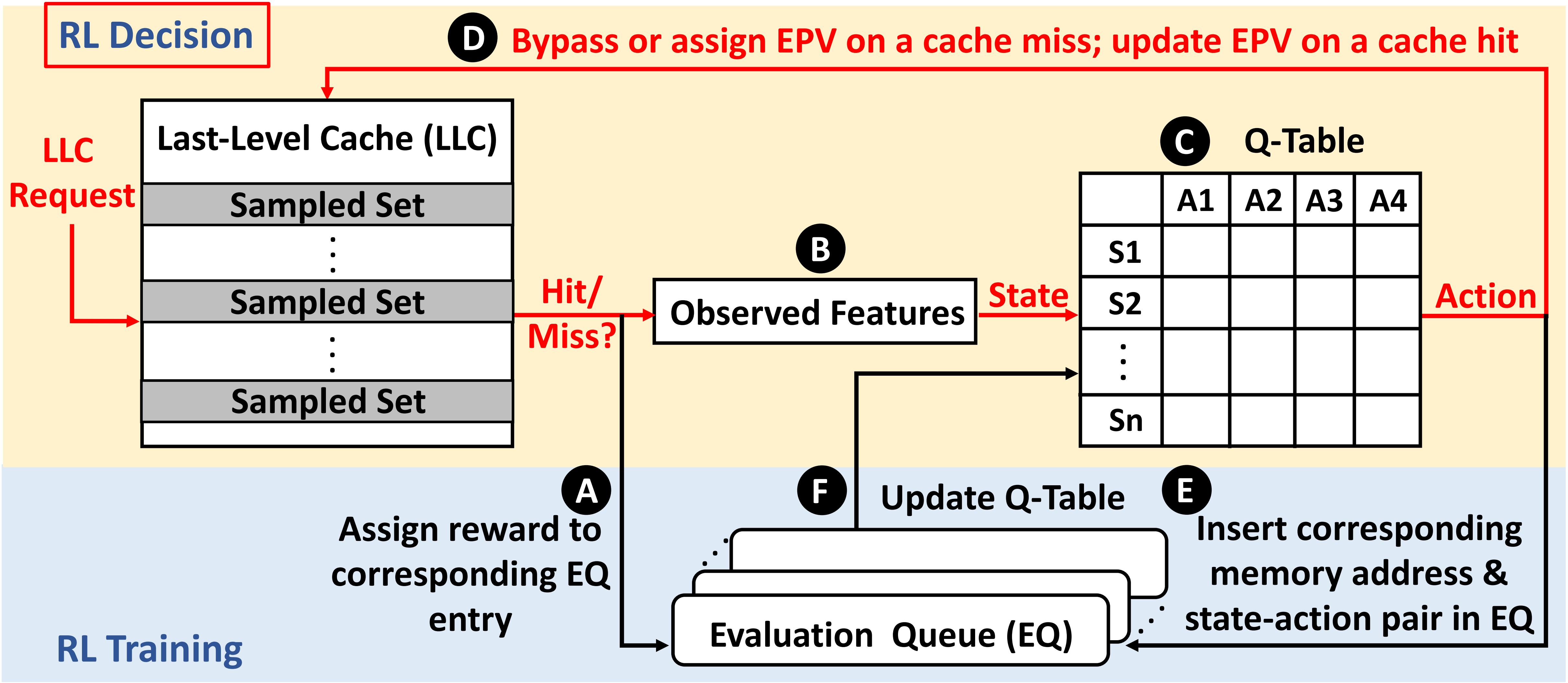
- Tracks the Q-values of all observed state-action pairs
- Given a state, CHROME picks a reasonable action based on the Q-Table



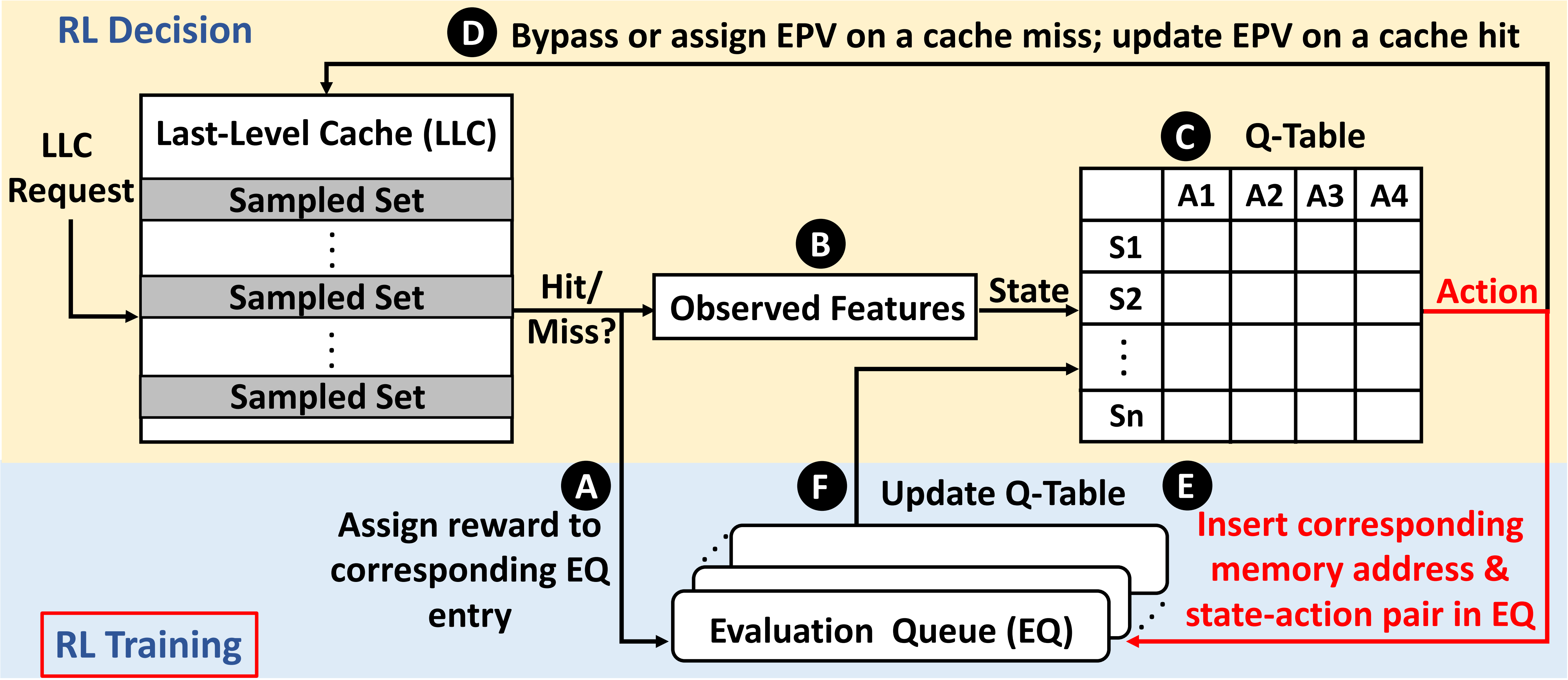
Evaluation Queue:

- Several first-in-first-out queues, each with a fixed capacity
- Records the actions of CHROME within a temporal window, which assists in rewarding

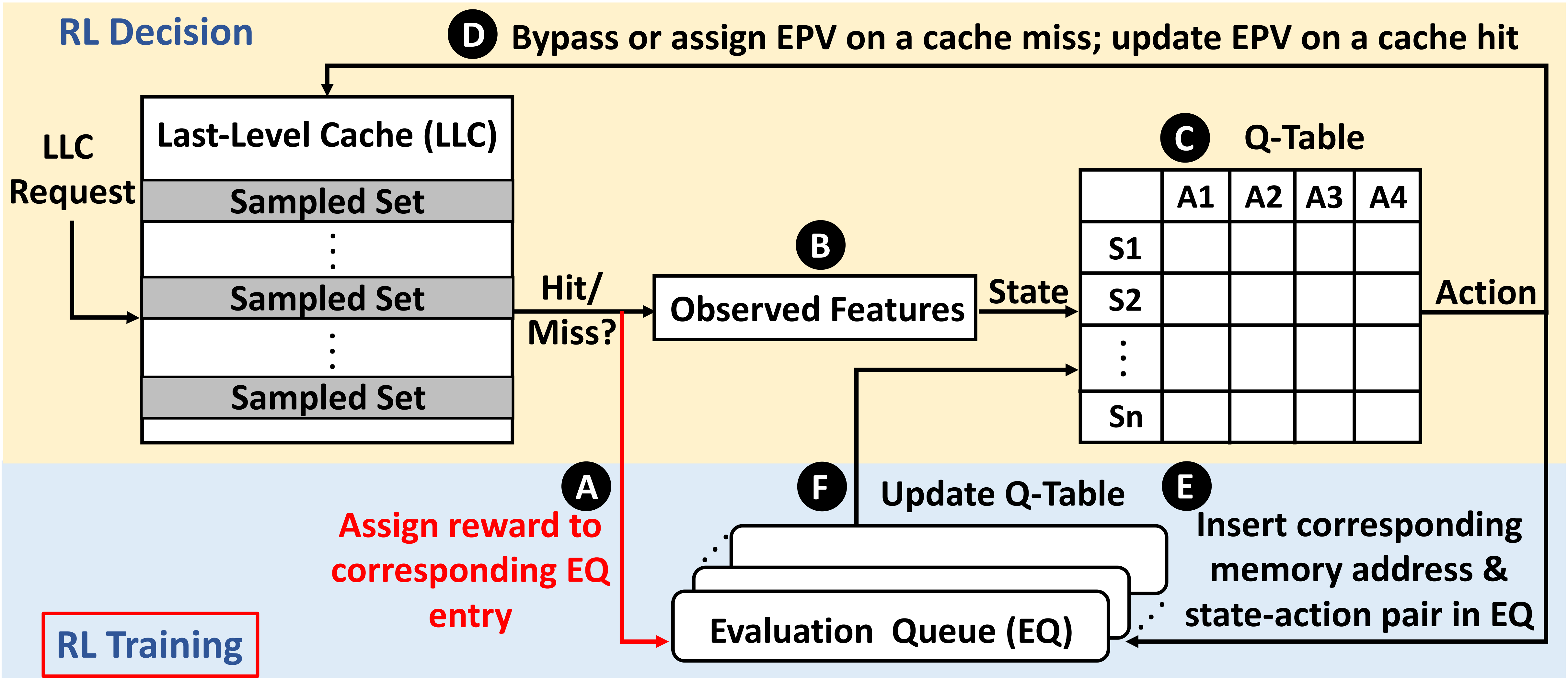
CHROME Workflow



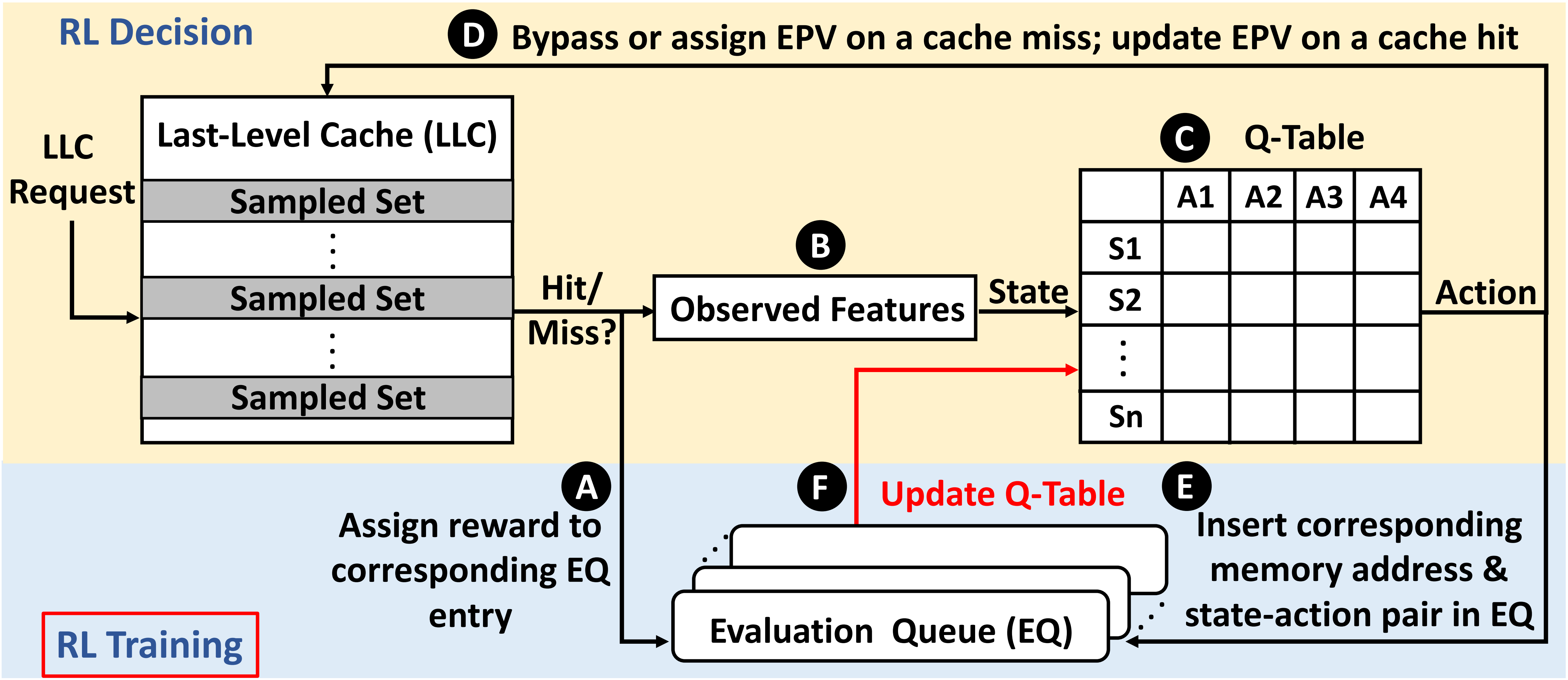
CHROME Workflow



CHROME Workflow



CHROME Workflow



More in the Paper

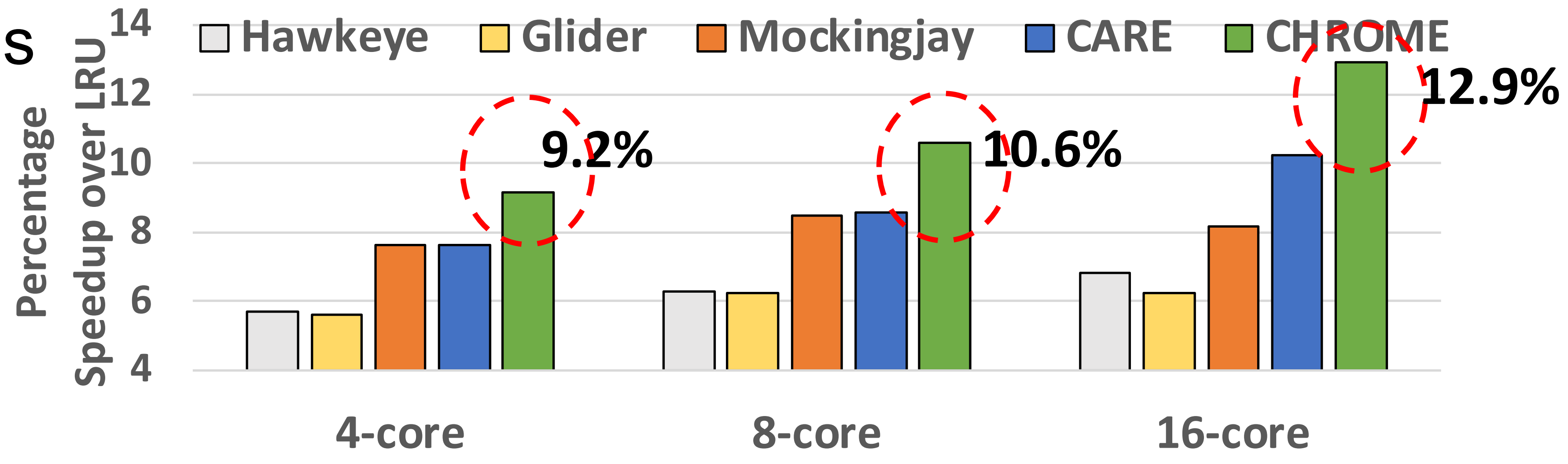
- Details on concurrency-aware system-level feedback
- Insights into the reward systems
- Pipelined organization of Q-Table
- EQ organization and Q-value update
- Turing of the hyper-parameter

Simulation Methodology

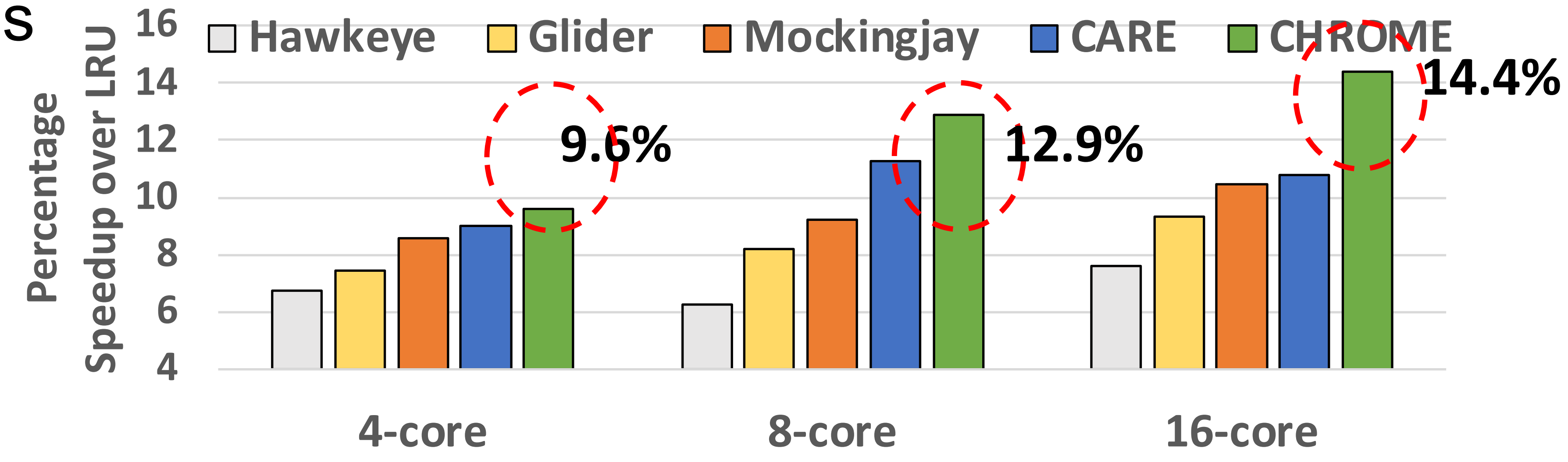
- **Champsim** trace-driven simulator
- **57** memory-intensive workload traces
  - SPEC CPU2006 and CPU2017
  - GAP
- **Homogeneous** and **heterogeneous** multi-core mixes
- **Prefetchers:**
  - L1D: Next-line prefetcher
  - L2: Stride prefetcher
- **Five** state-of-the-art LLC management schemes:
  - LRU
  - Hawkeye [ISCA'16]
  - Glider [MICRO'19]
  - Mockingjay [HPCA'22]
  - CARE [HPCA'23]

Performance with Varying Core Count

SPEC workloads homogeneous



SPEC workloads heterogeneous



Performance with Varying Core Count

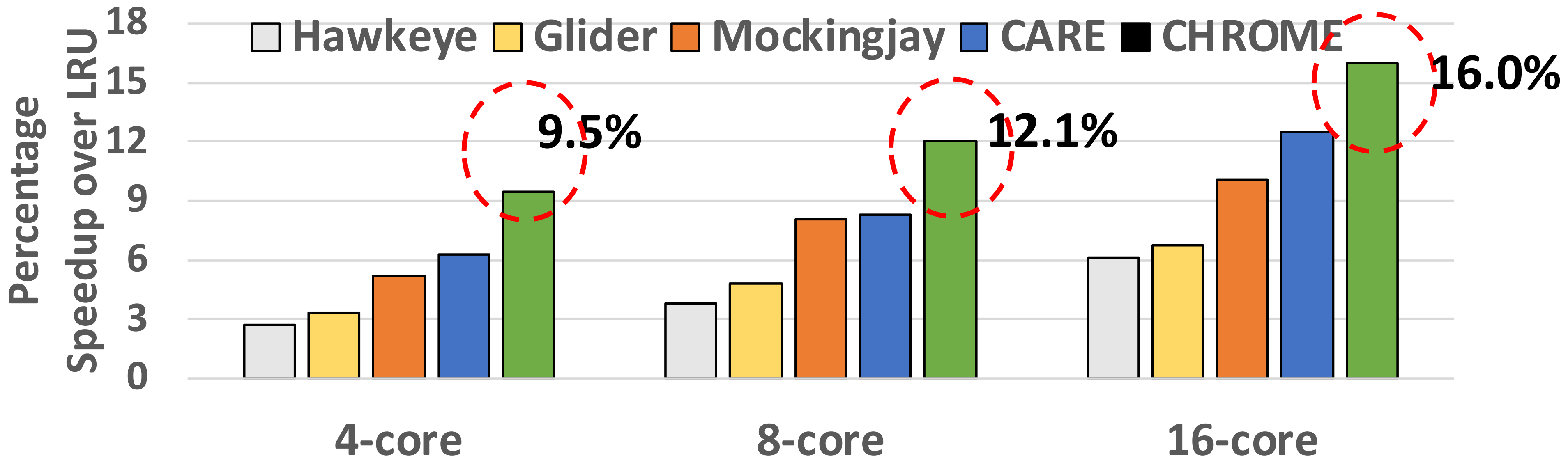
CHROME can accurately provide cache management for different workloads

CHROME outperforms all other schemes across all system configurations

Performance advantage of CHROME over others increases with more cores.

Performance on Unseen Traces

GAP workloads



Performance on Unseen Traces

GAP workloads

The holistic view provides consistent performance advantage

Online RL provides good adaptability and scalability

Cost

TABLE III: Storage overhead of CHROME.

Component	Details	Overhead
Q-Table	2 features; 4 sub-tables/feature; 2048 entries/sub-table; 16 bits/entry	**32KB**
EQ	64 queues; 28 entries/queue; 58 bits/entry (state: 33 bits, action: 2 bits, reward: 6 bits, hashed address: 16 bits, trigger: 1 bit)	**12.7KB**
Metadata	EPV (2-bit/LLC block)	**48KB**
Total		**92.7KB**

TABLE IV: Storage overhead for different schemes (4-core configuration, 12-way 12MB LLC).

	Holistic	Concurrency-aware	Overhead
Hawkeye [21]	No	No	146KB
Glider [44]	No	No	254KB
Mockingjay [43]	Yes	No	170.6KB
CARE [43]	No	Yes	130.5KB
CHROME	**Yes**	**Yes**	**92.7KB**

- The complexity of the prediction path is similar to that of the other SOTA prediction-based schemes
- We used CACTI 7.0 to estimate the latency for the Q-Table lookups, also the area and power consumption:
  - Q-Table lookup latency is ~2 cycles
  - Q-Table operations are off the critical path -> no interference with cache controller
- The area and power consumption of CHROME is rather modest

Summary

CHROME is a **holistic** cache management framework

CHROME continuously learns the policy by utilizing **online RL**

CHROME considers **multiple program features** and **concurrency-aware system-level feedback** information

CHROME **outperforms** state-of-the-art cache management schemes

CHROME: Concurrency-Aware Holistic Cache Management Framework with Online Reinforcement Learning

Xiaoyang Lu, Hamed Najafi, Jason Liu, Xian-He Sun

U.S. National Science Foundation

ILLINOIS TECH

FIU FLORIDA INTERNATIONAL UNIVERSITY