

LIPA: A Learning-based Indexing and Prefetching Approach for Data Deduplication

Guangping Xu*, Bo Tang*, Hongli Lu*, Quan Yu# and Chi Wan Sung+

* Tianjin University of Technology, Tianjin, China

Wuhan University of Technology, Tianjin, China

+City University of Hong Kong, Hong Kong, China



Santa Clara, May 24, MSST 2019

Outline

Introduction

Our approach

Experimental results

Conclusion

Data streams

In cloud storage system, **streams** are logical data channels that transfer data from the client to the storage backend.

When the data streams are received by the Media Agent, the data is divided into chunks that are written to Storage Media.



Source:



Data streams

Example: Google cloud storage API

You can perform streaming data transfers by using the following command-line tool:
gsutil.

```
collect_measurements | gsutil cp - gs://my_app_bucket/data_measurements
```

data source: application/process

stored target: object

Key Features



Single API across
storage classes



Scalable to exabytes
of data



Designed for
99.999999999%
durability

Source:

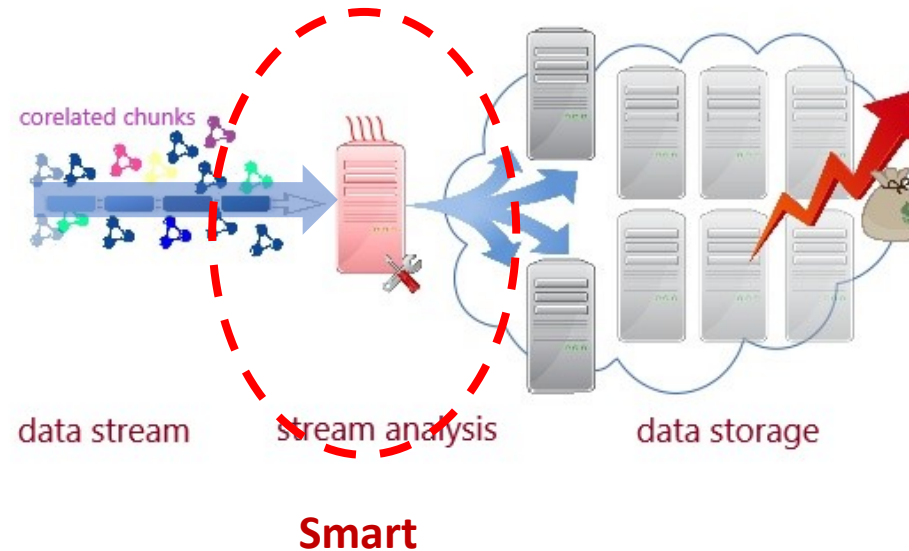


Deduplication and Goal

To save the storage cost, the deduplication technique is used:
The data stream is divided into chunks to check duplicate or not. We only store their metadata for duplicate chunks.

The accuracy of correlated chunks in data stream analysis affects the ratio.

The storage cost depends on the deduplication ratio.



Challenges and Contribution

Challenge 1

Accuracy: ensure the analysis could automatically capture the dynamics of data stream.

Challenge 2

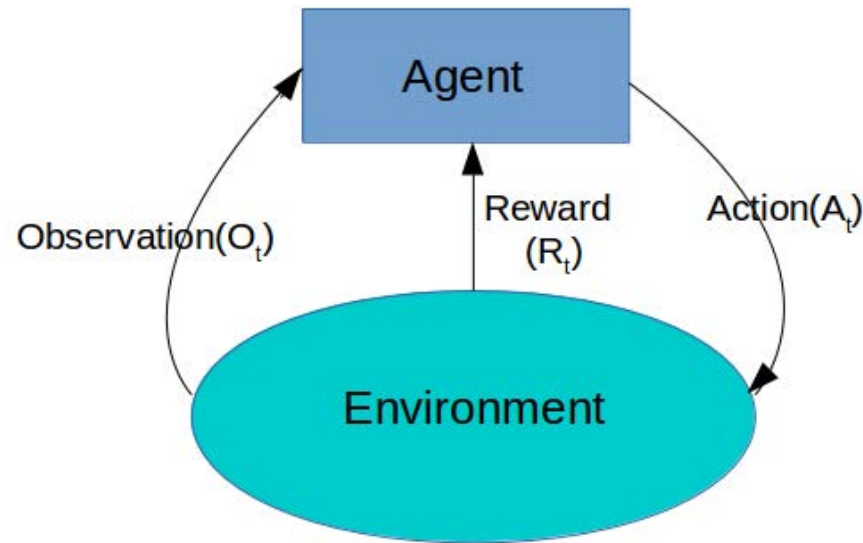
Efficiency: ensure the analysis could work in an efficient way

Solution: A Learning-based Indexing and Prefetching Approach, called LIPA, which uses the reinforcement learning framework.

Reinforcement Learning (RL)

RL method:

- ◆ a goal-directed online learning approach from interaction by an agent over time;
- ◆ The interaction between a learning agent and its environment in terms of **states**, **actions**, and **rewards**.



Outline

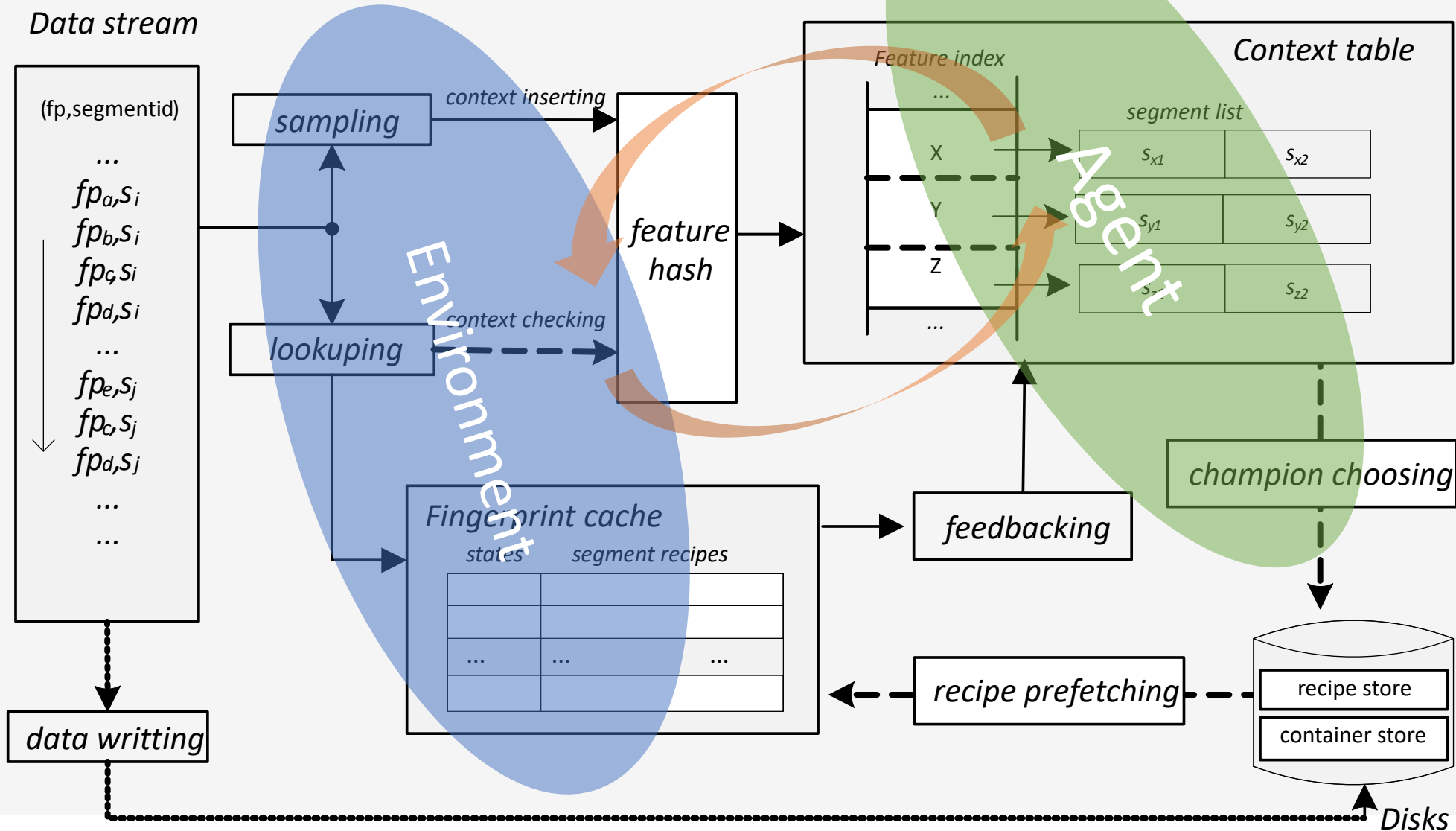
Introduction

Our approach

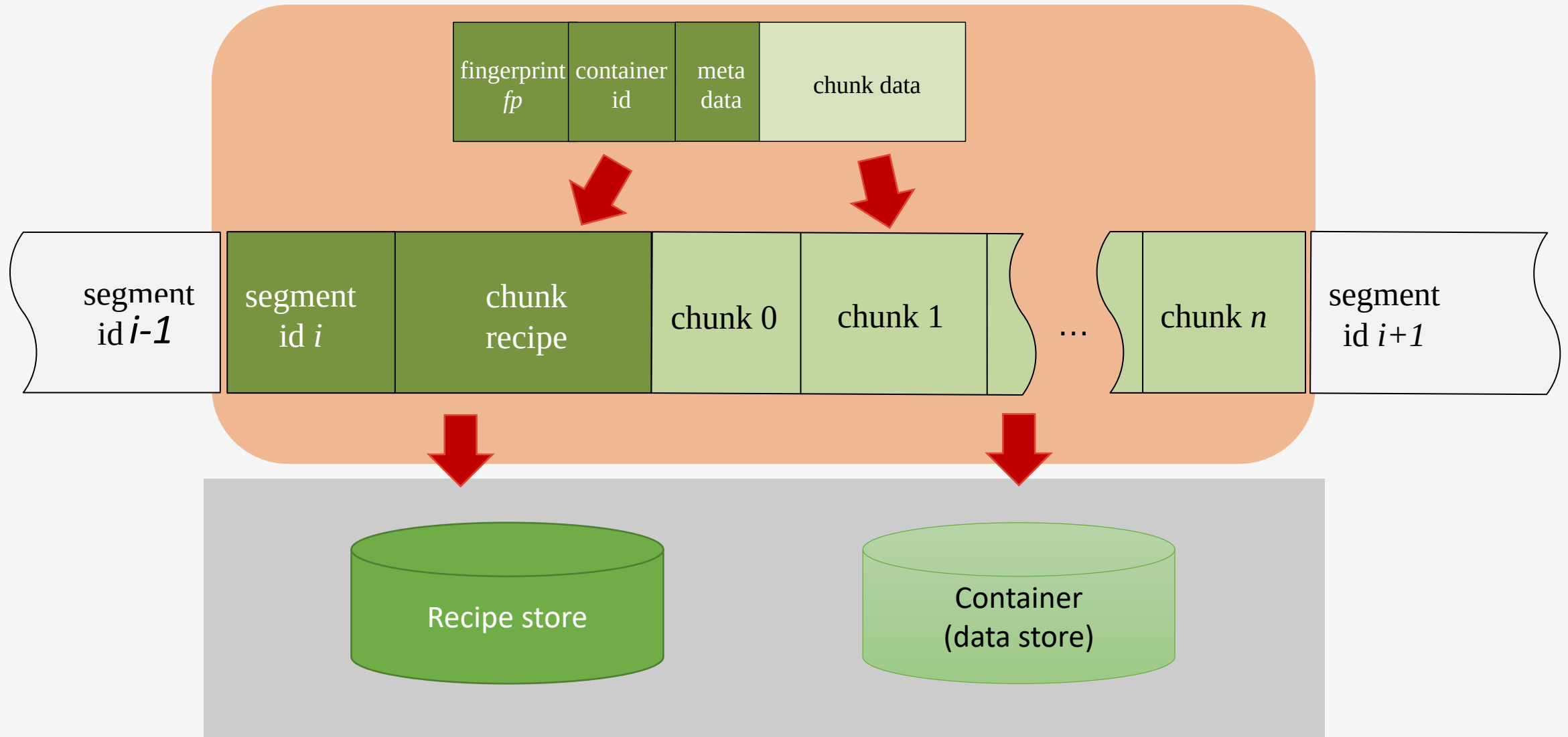
Experimental results

Conclusion

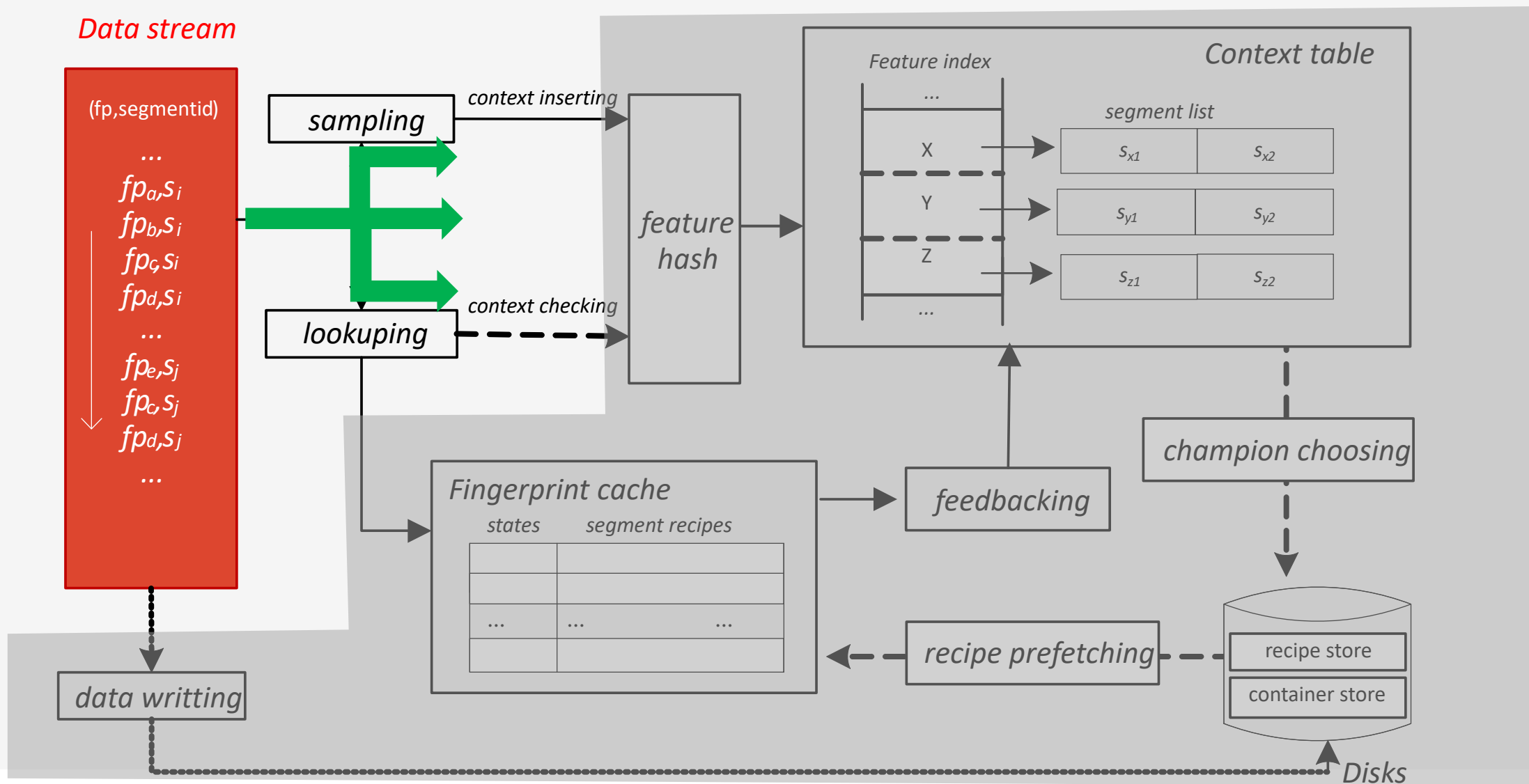
Architecture Overview



Basic terms



1. Chunking and segmenting



2. What are segment features?

Information of chunks may include:

Fingerprints of chunks

- ◆ Max/min fingerprints

Semantic information of a segment and its associated objects may include:

1) Application/process metadata

- ◆ such as file type, file size, application software (process name), etc.;

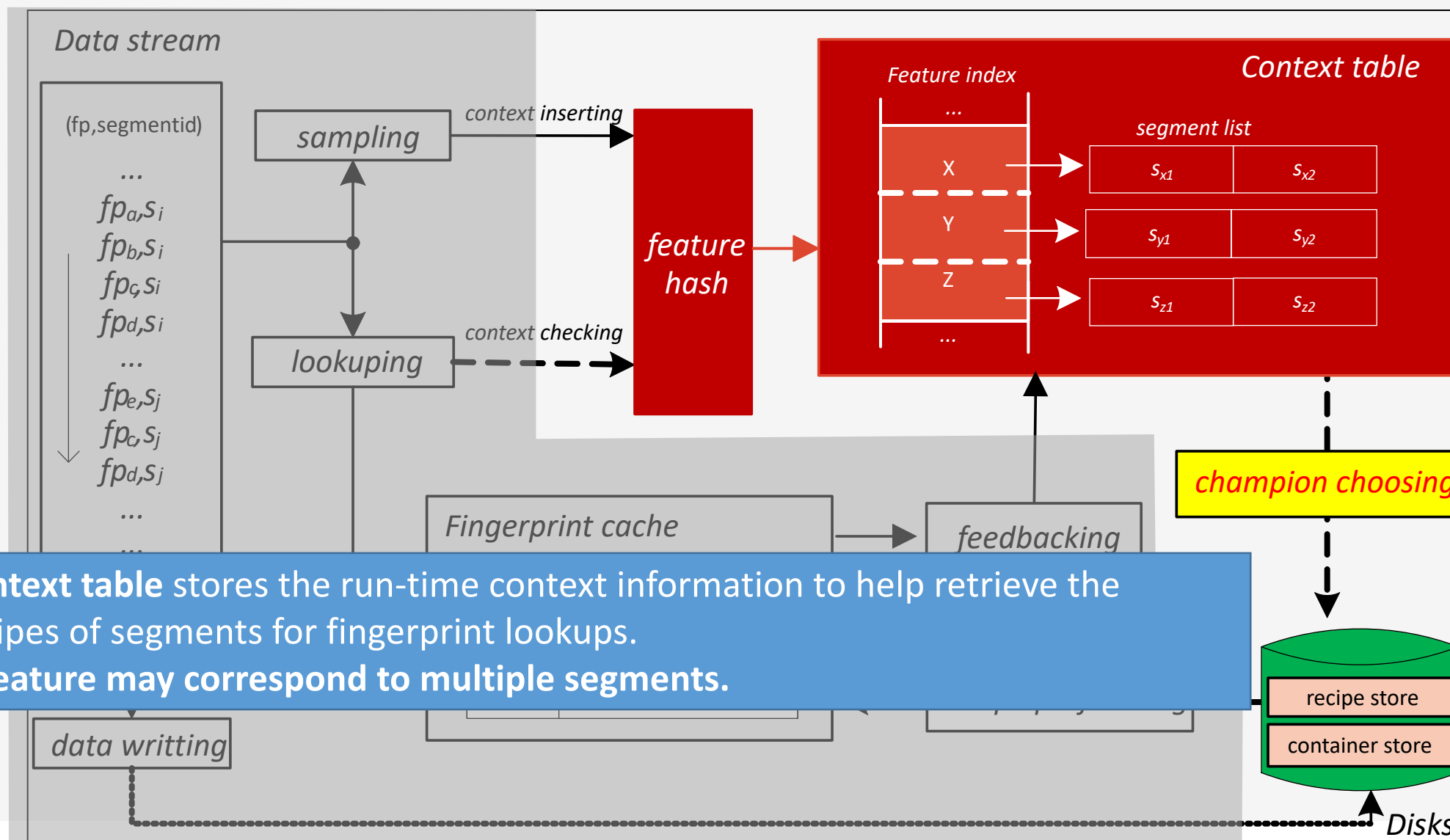
2) User tags

- ◆ such as the tags used to describe the characteristics of blogs, images or multimedia;

3) Filesystem metadata

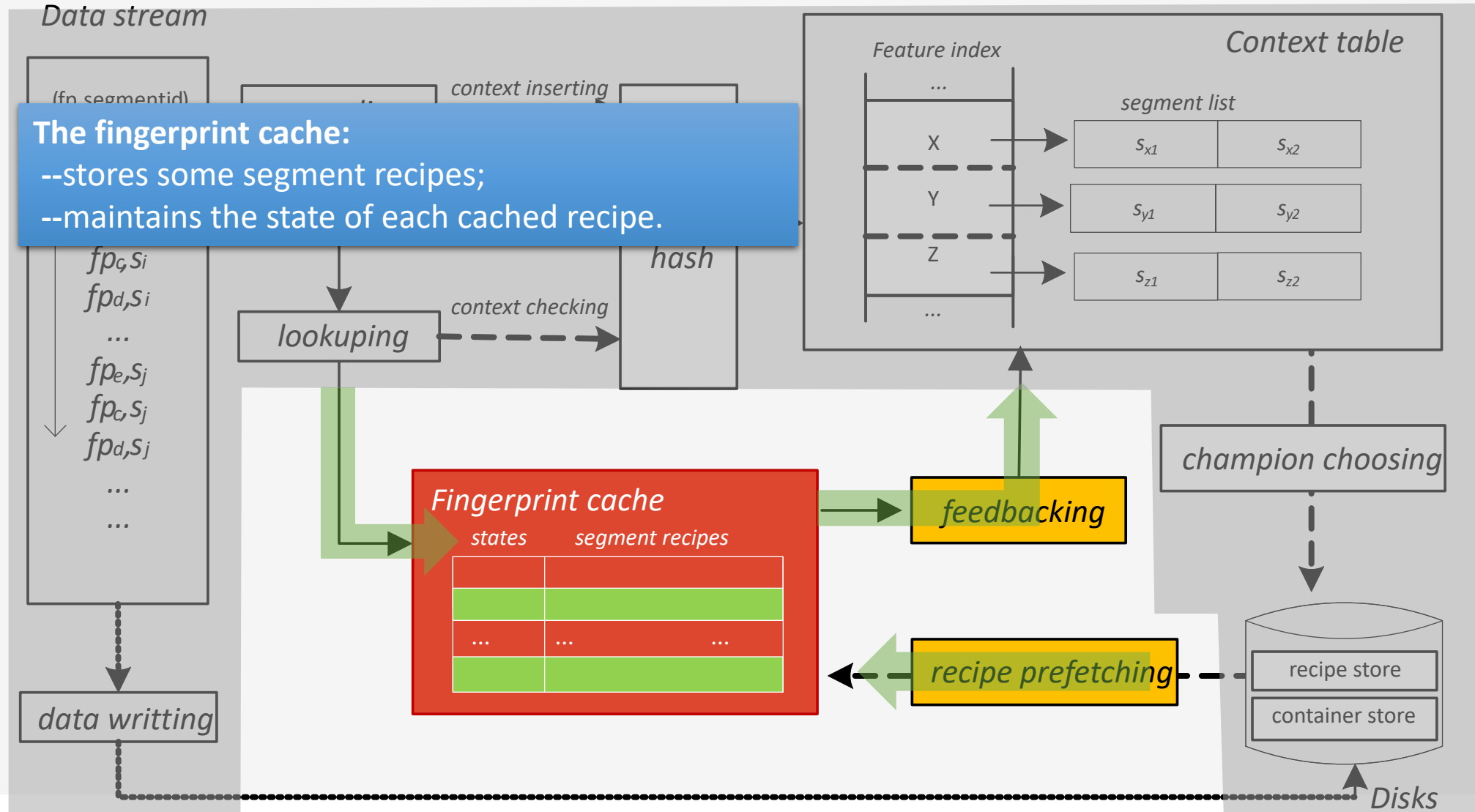
- ◆ such as directory entries, i-node information of a file.

3. Context table

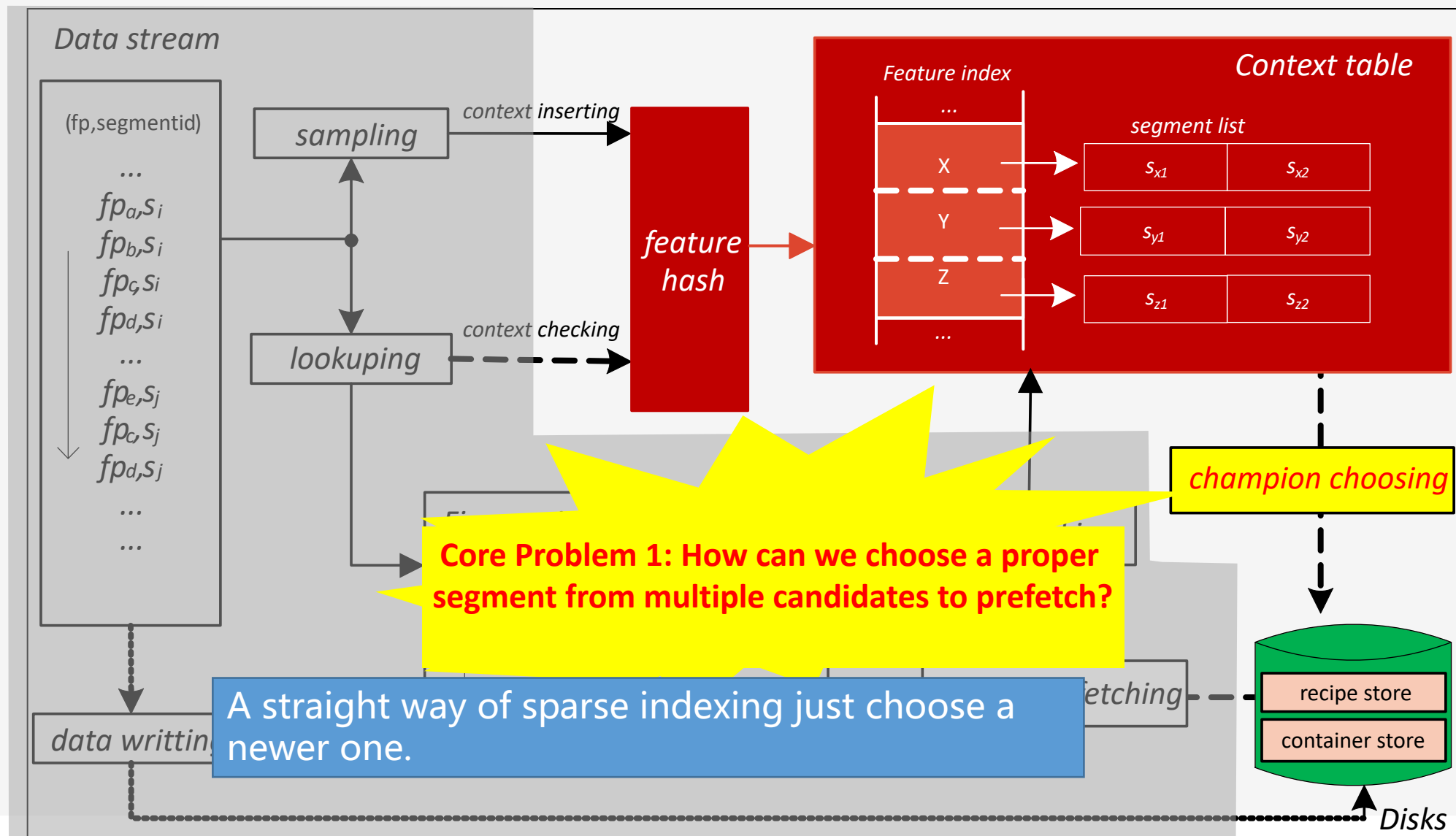


Context table stores the run-time context information to help retrieve the recipes of segments for fingerprint lookups.
A feature may correspond to multiple segments.

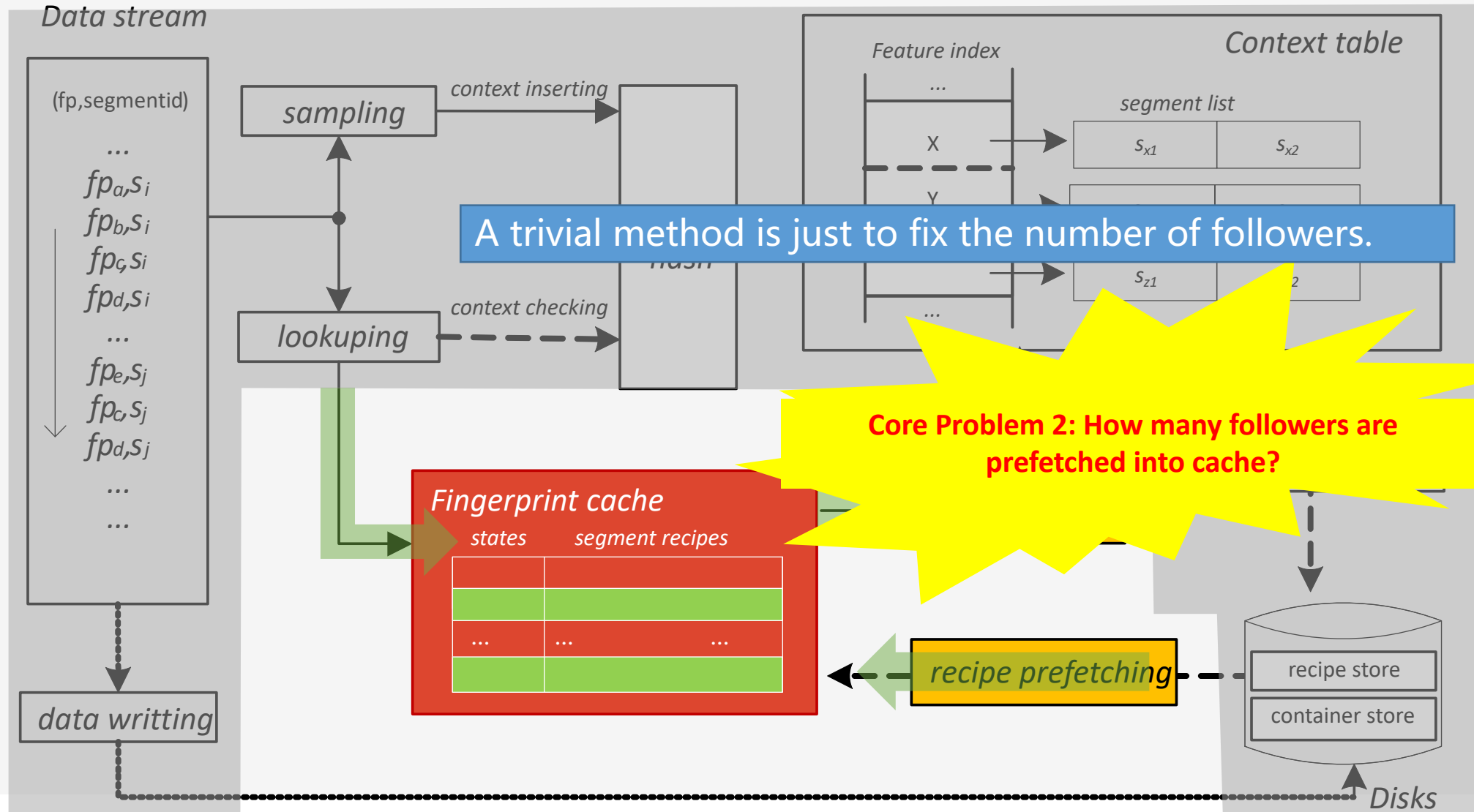
4. Fingerprint cache



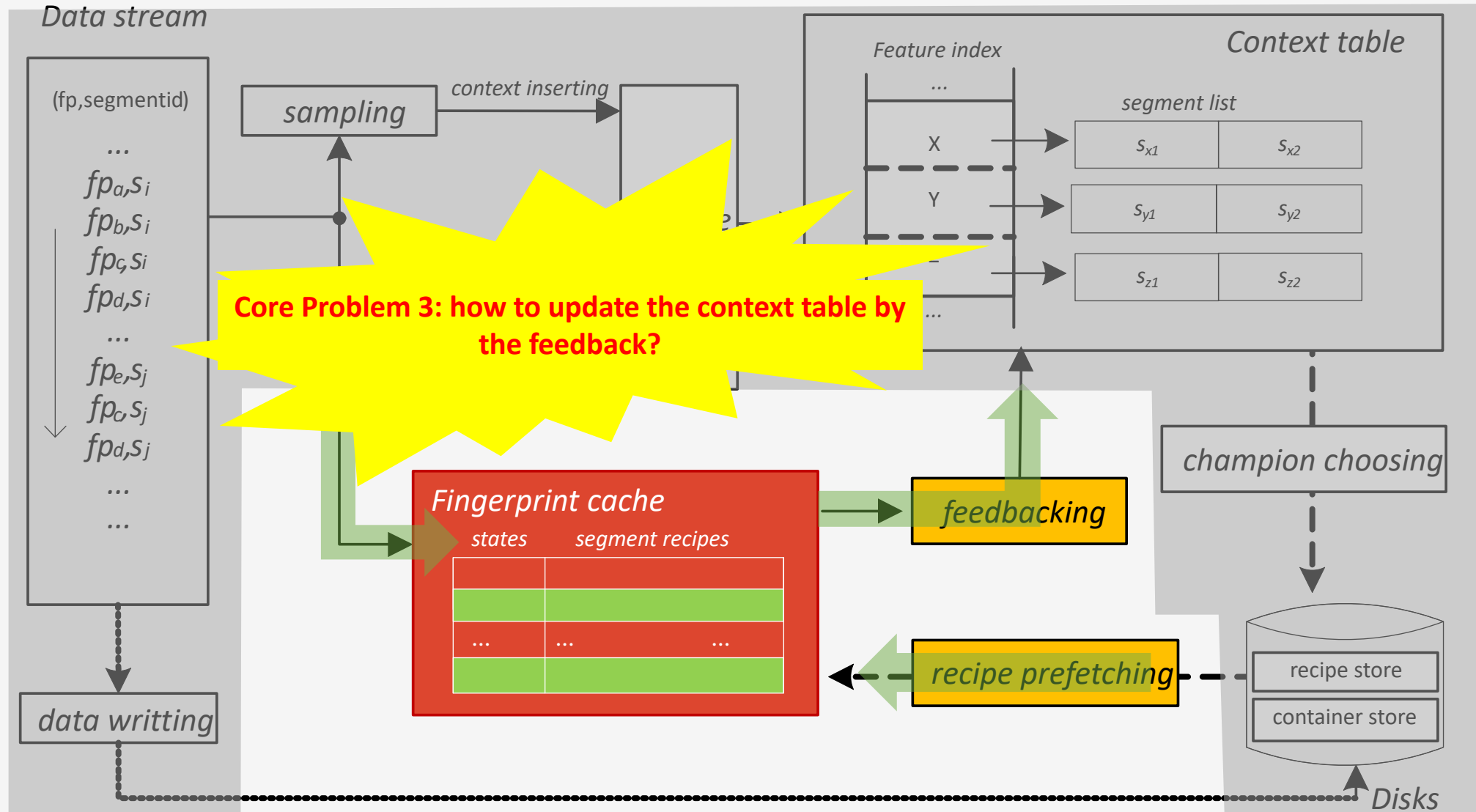
Core problem 1



Core problem 2



Core problem 3

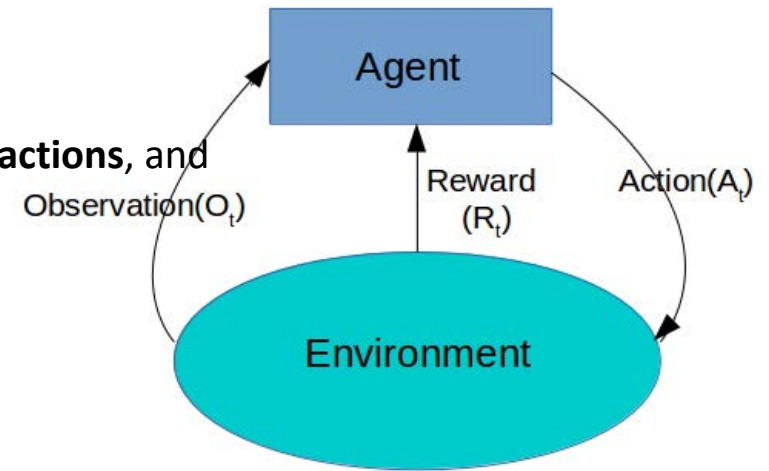


The 3 core problems & Reinforcement Learning

RL method:

- ◆ a goal-directed online learning approach from interaction by an agent over time;
- ◆ The interaction between a learning agent and its environment in terms of **states**, **actions**, and **rewards**.

RL clearly fits to address these three core problems [1, 2].



- ◆ **(Observation)** The prefetcher need to **observe** the stream context and cached segment states,
- ◆ **(feedback)** **update** the context table by feedbacking rewards,
- ◆ **(Action)** and **decide** which segment as champion and how many its followers to prefetch into cache.

[1] Renato Costa, Jose Pazos. MLCache: A Multi-Armed Bandit Policy for an Operating System Page Cache., University of British Columbia, 2018

[2] Leeor Peled; Shie Mannor; Uri Weiser; Yoav Etsion. Semantic Locality and Context-based Prefetching Using Reinforcement Learning. ISCA 2015.

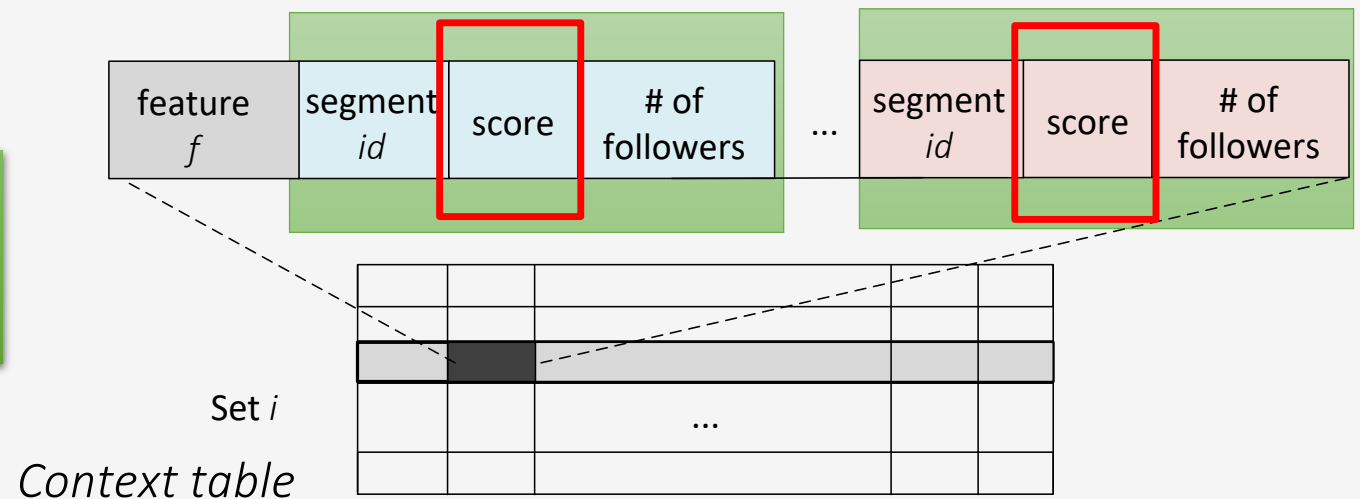
The 3 core problems & Reinforcement Learning (1)

Core Problem 1: How to choose a proper segment from multiple segment candidates?

A multi-armed bandit is a simple model. It could achieve a balance between exploration and exploitation for some options to select.

The **ϵ -greedy policy**: To determine a champion every time, it may choose a segment with **the highest score** with **$1-\epsilon$** probability (i.e., **exploitation** of the past); otherwise, it may choose one randomly (i.e., **exploration** of the new).

By acting greedily, we try to evict a segment which has the lowest score and keep segments that have higher reward.



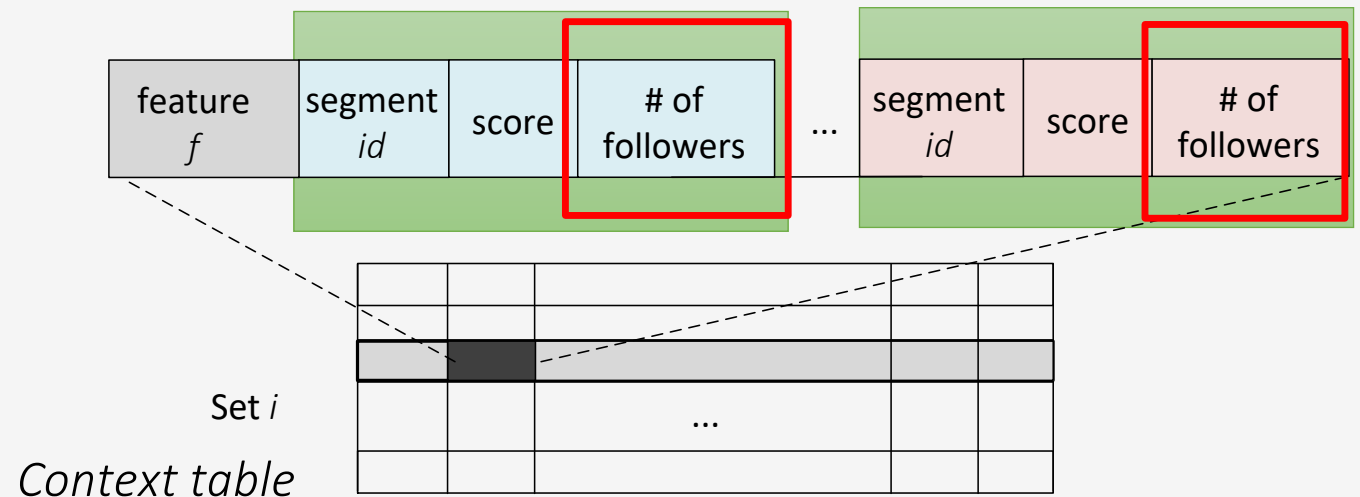
The 3 core problems & Reinforcement Learning (2)

Core Problem 2: How many follows are prefetched into cache?

A trivial method is just to fix the number of followers.

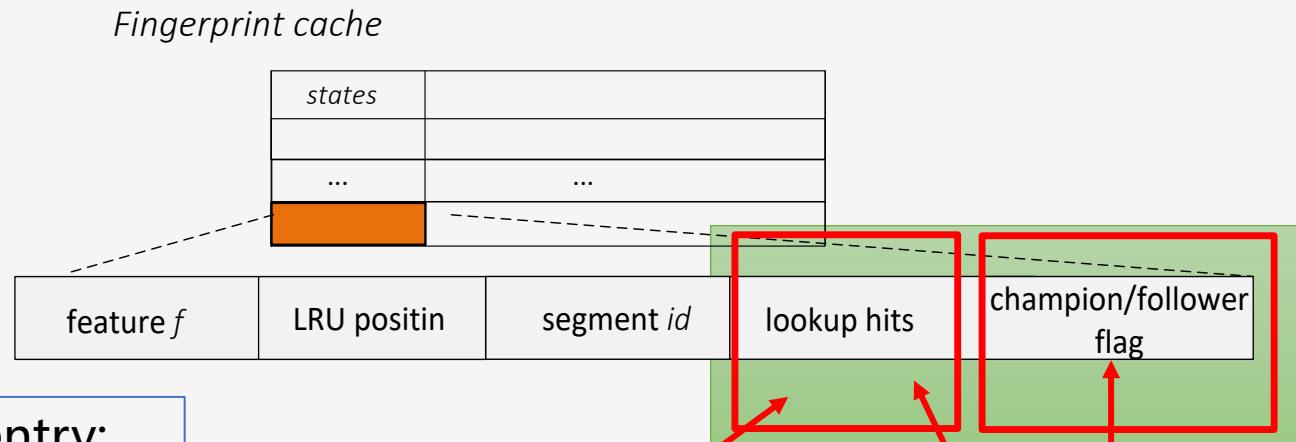
It would desire **an adaptive prefetching way** to **dynamically adjust** the number of followers.

The number of prefetched followers is needed as an attribute of a segment in each entry of the context table.



The 3 core problems & Reinforcement Learning (3)

Core Problem 3: how to update the context table by the feedback?



add two attributes to each cache entry:
lookup hits and **follower flag**.

A reward is calculated from the lookup hits.

The number of followers of a champion is determined from the flag.

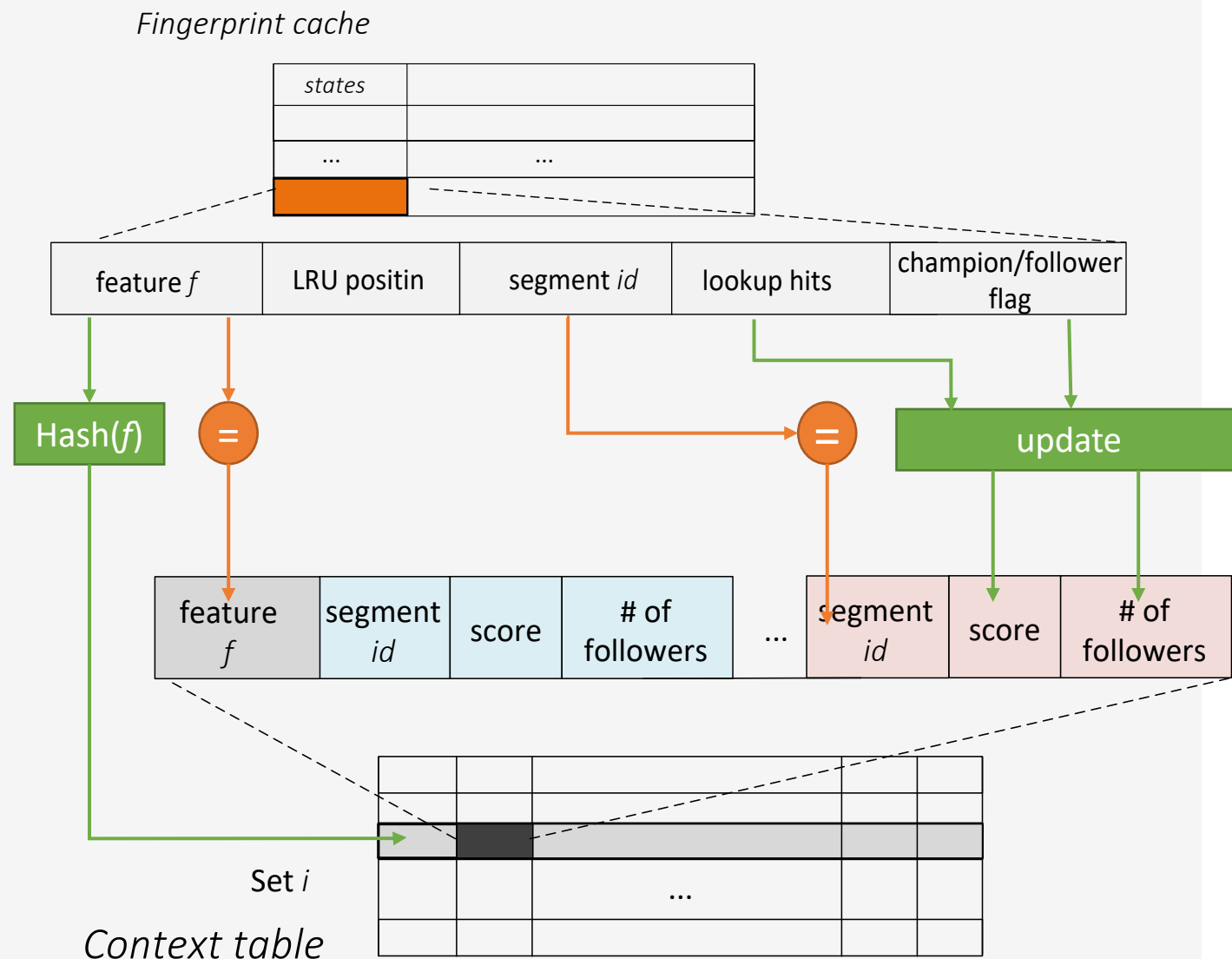
Implementation

The interaction between both components:

- ◆ Fingerprint cache
- ◆ Context table

For a segment s , $Q_n(s)$ is the estimated score after segment s is chosen as a champion n times with rewards $r_1, r_2, \dots, r_n$

$$Q_n(s) = \frac{1}{n} [(n-1) Q_{n-1}(s) + r_n]$$



Outline

Introduction

Our approach

Experimental results

Conclusion

Experiments and datasets

We implemented the learning-based deduplication approach in a prototype system.

It is running in a Linux operating system. The hardware configuration includes a quad-core CPU running at 2.4GHz with 4GB RAM and two 1TB 7200rpm hard disks.

TABLE I
THE CHARACTERISTICS OF FOUR REAL-WORLD DATASETS USED IN OUR
EXPERIMENTS.

Dataset name	number of version	Total size	Dupliation ratio
Kernel	115	24.9GB	95.06%
Vmdk	110	180.9GB	42.14%
Fslhomes	126	4.378TB	95.22%
Macos	18	1.628TB	96.69%

Experimental evaluation (1): the champion choosing policy

Effectiveness of the ϵ -greedy champion choosing method

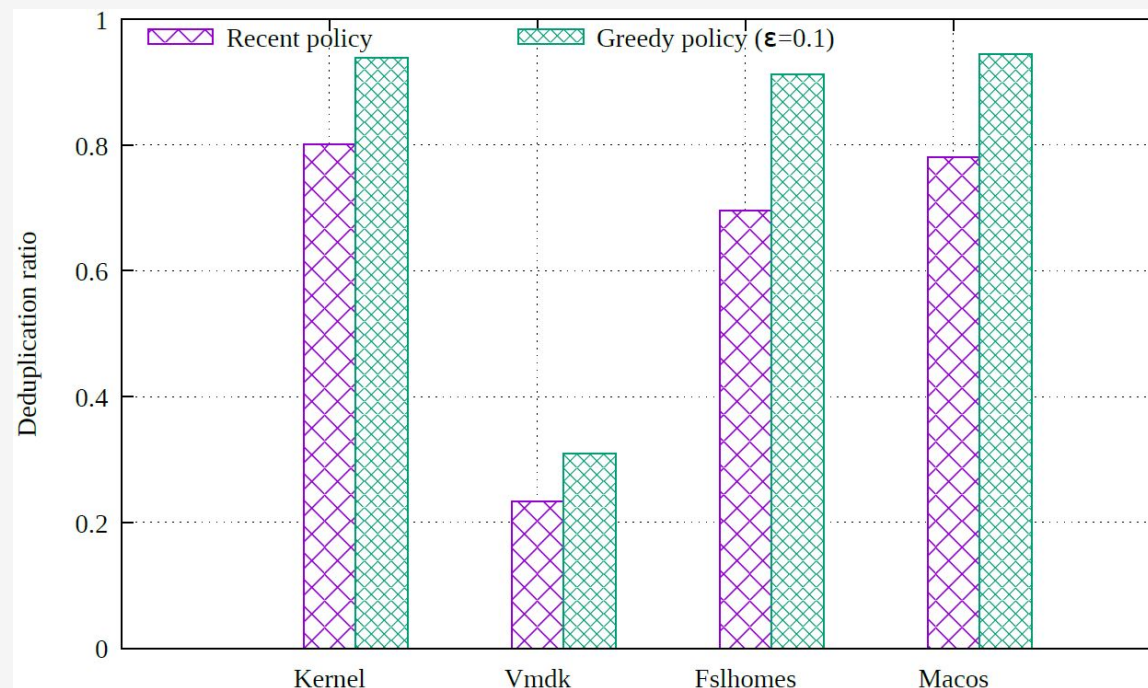
The base policy to choose a champion:

recent policy (Sparse Indexing)

- ◆ always chooses the newest segment as champion

ϵ -greedy policy

- ◆ chooses the segment with the highest score as champion with probability $(1-\epsilon)$



Experimental evaluation (2): the greedy parameter value (ϵ)

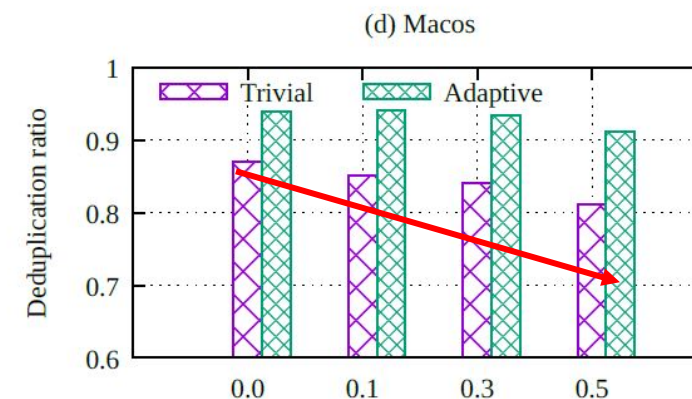
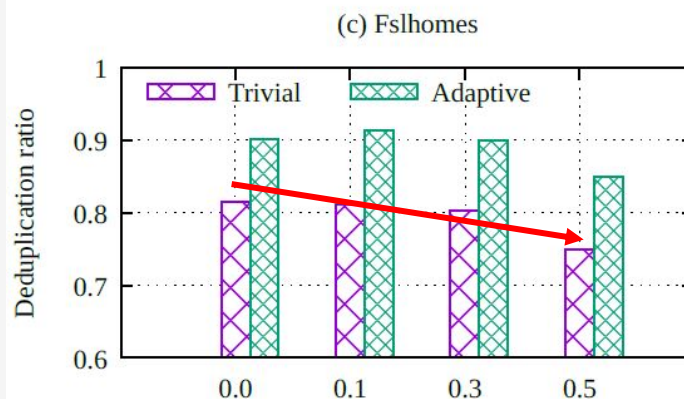
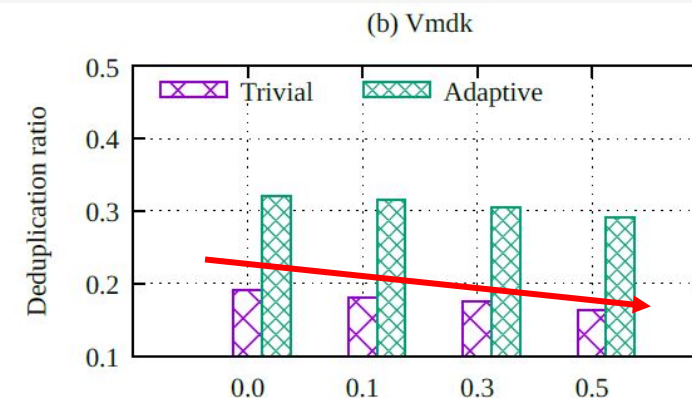
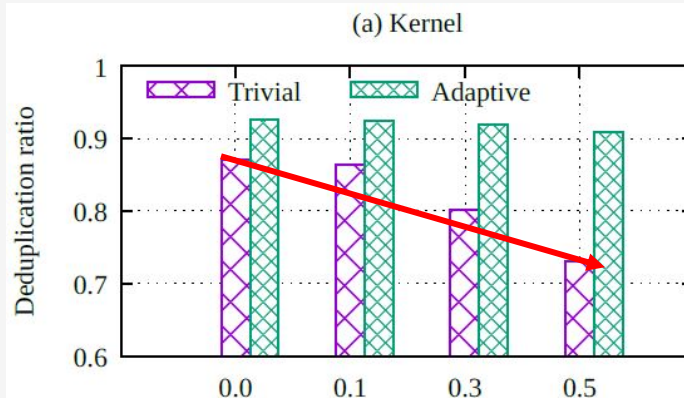
Evaluation of the greedy champion choosing method by varying $\epsilon = 0, 0.1, 0.3$ and 0.5 , respectively.

The trivial followers prefetching (Sparse indexing)

- ◆ It always prefetches a fixed number of followers ($n = 4$).

The adaptive followers prefetching

- ◆ It prefetches a dynamic number of followers which is initialized by the same n .



Experimental evaluation (3): the adaptive prefetch of followers

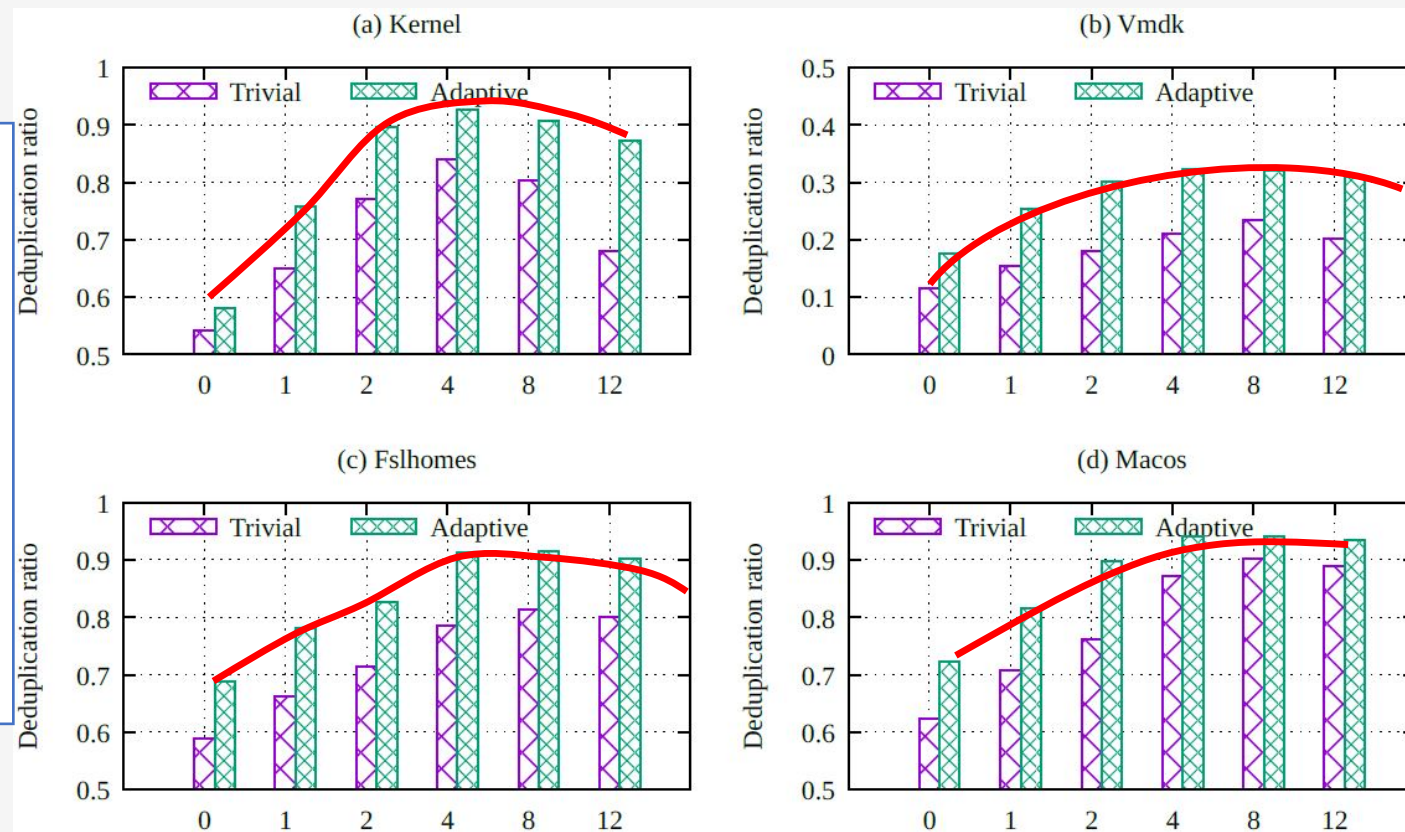
Comparison of the trivial prefetching way and the adaptive prefetching way, by varying $n = 0, 1, 2, 4, 8$ and 12 , respectively and fixing $\varepsilon = 0.1$.

The trivial followers prefetching (Sparse indexing)

- ◆ It always prefetches a fixed number of followers ($n=0, 1, 2, 4, 8$ and 12).

The adaptive followers prefetching

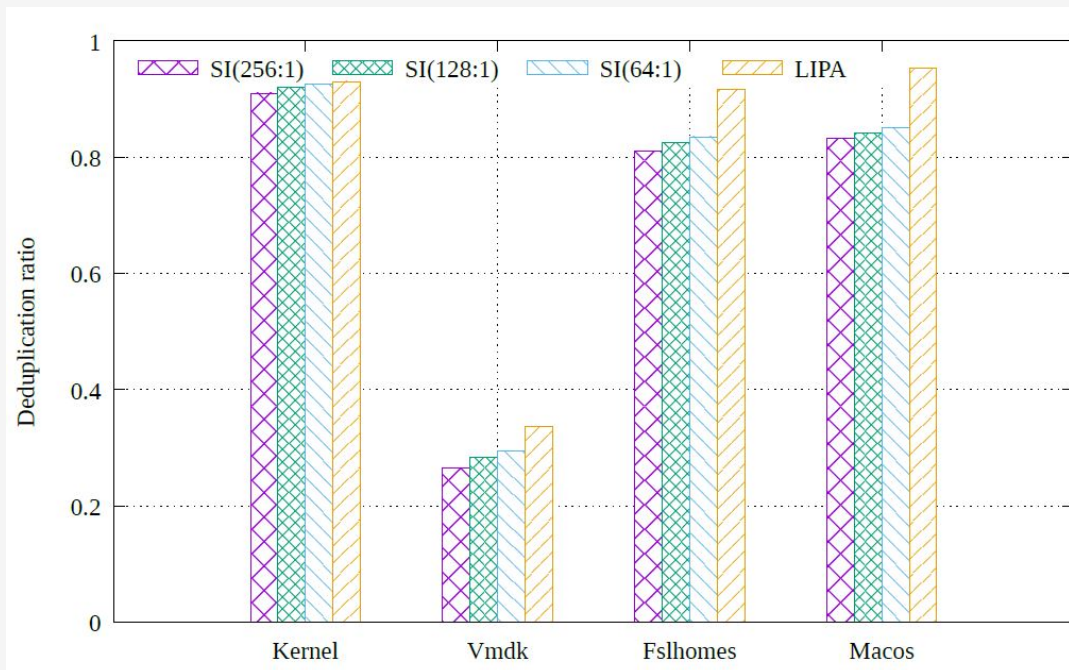
- ◆ It prefetches a dynamic number of followers which is initialized by the same n .



Comparison with Sparse Indexing (SI)

Comparison in deduplication ratio between our approach ($\epsilon=0.1$ and $n=4$) and Sparse Indexing with different sampling ratios ($r=256:1$; $128:1$ and $64:1$).

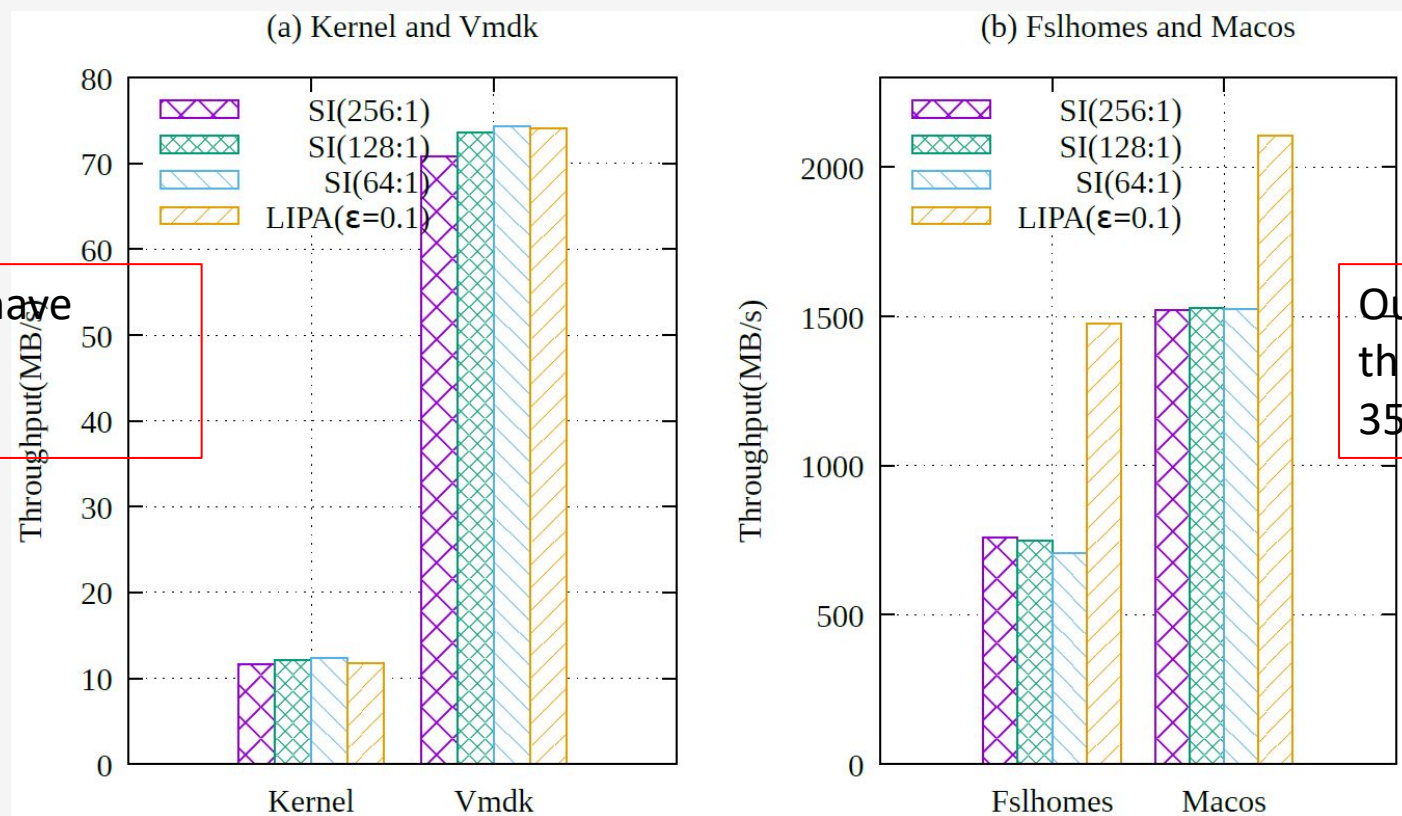
Our approach improves the deduplication ratio by 4%~10% than sparse indexing



Deduplication ratio

Comparison with Sparse Indexing (SI)

Comparison in data throughput between our approach ($\epsilon = 0.1$ and $n=4$) and Sparse Indexing with different sampling ratios ($r = 256:1$; $128:1$ and $64:1$)



both approaches have almost the same throughput

Our approach improve throughput by 50% and 35%, respectively

Data throughput

Conclusion

- ◆ We proposed a learning-based deduplication framework for deduplication.
 - To the best of our knowledge, this is the first learning-based deduplication.
- ◆ The benefits of our approach mainly relies on two features, the greedy champion choosing and the adaptive recipe prefetching, according to the reinforcement learning.
 - The experimental results demonstrate that our approach obtains better performance.

Thanks!

Q&A

Reward

time step $n > 0$:

$$\max_{1 \leq j \leq k} Q_{j,n} + \sqrt{\frac{2 \log(n)}{N_j}}$$

where $Q_{j,n}$ is the average hits obtained from action j , N_j is the number of times action j has been played.