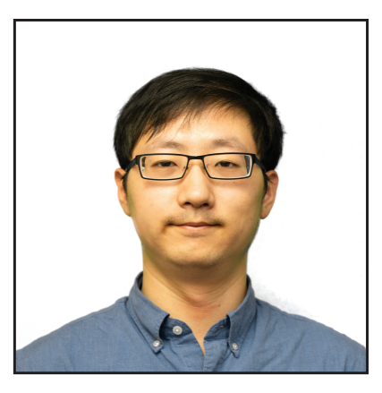
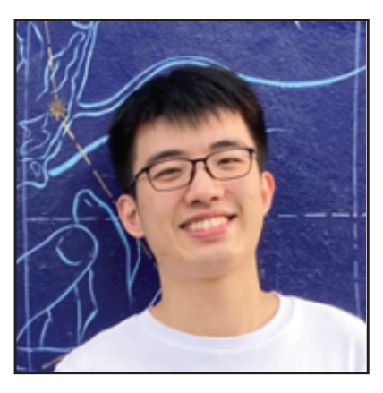
GL-Cache: Group-level Learning for Efficient and High-performance Caching

Juncheng Yang (Carnegie Mellon), Ziming Mao (Yale University), Yao Yue (Pelikan Foundation), K. V. Rashmi (Carnegie Mellon)

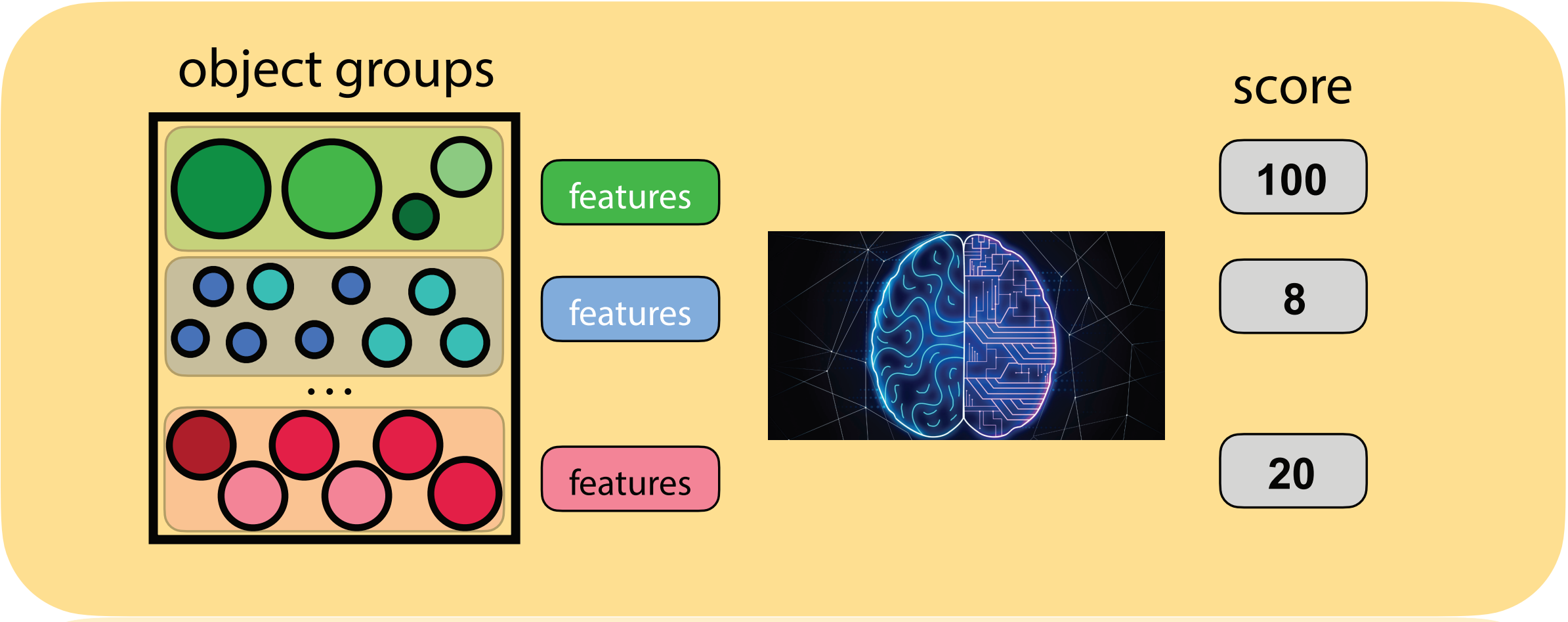
Introduction

- Cache is widely deployed to support the modern Internet
- Two metrics are important for a cache: efficiency (measured by hit ratio) and throughput performance
- Many recent works improve the efficiency of caches using machine learning

Group-level Learning

4. move this section after background

- Amortizes the cost of learning across multiple objects
- Can accumulate more information for learning since most objects have very few requests



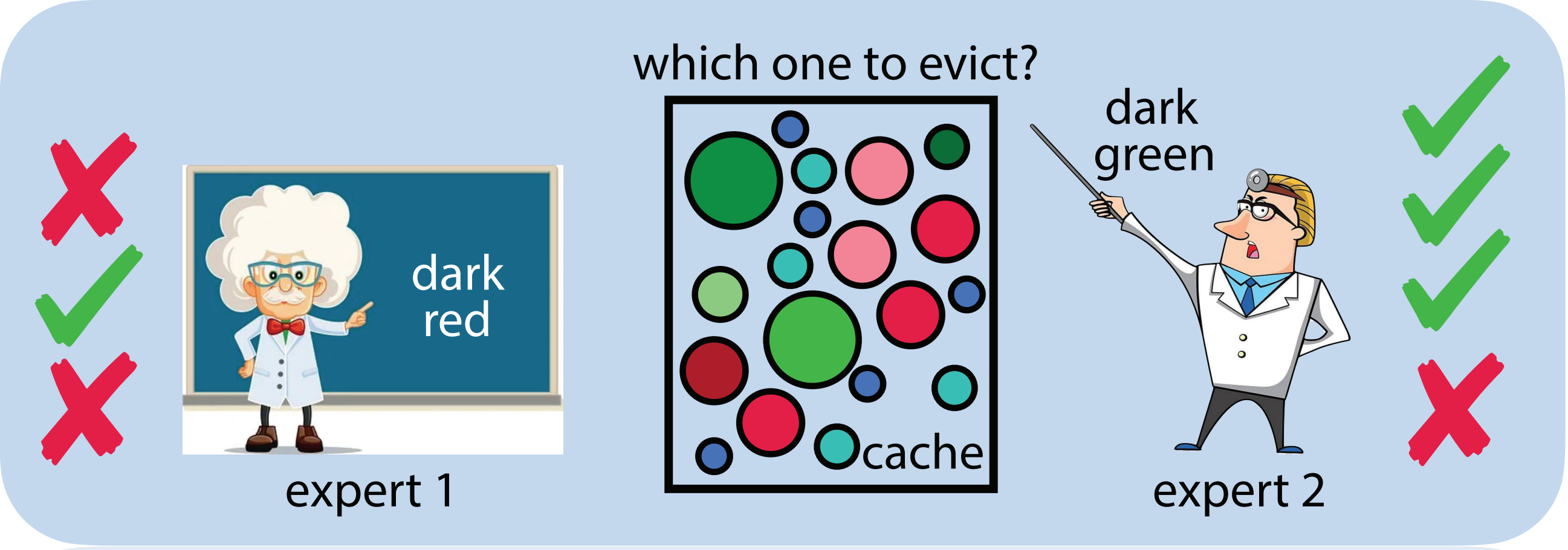
utilizes multiple features, while amortizes overheads
groups accumulate more information and are easier to learn

Group-level learned cache

Background: Learned Caches

- We categorize the existing learned caches into 3 types:

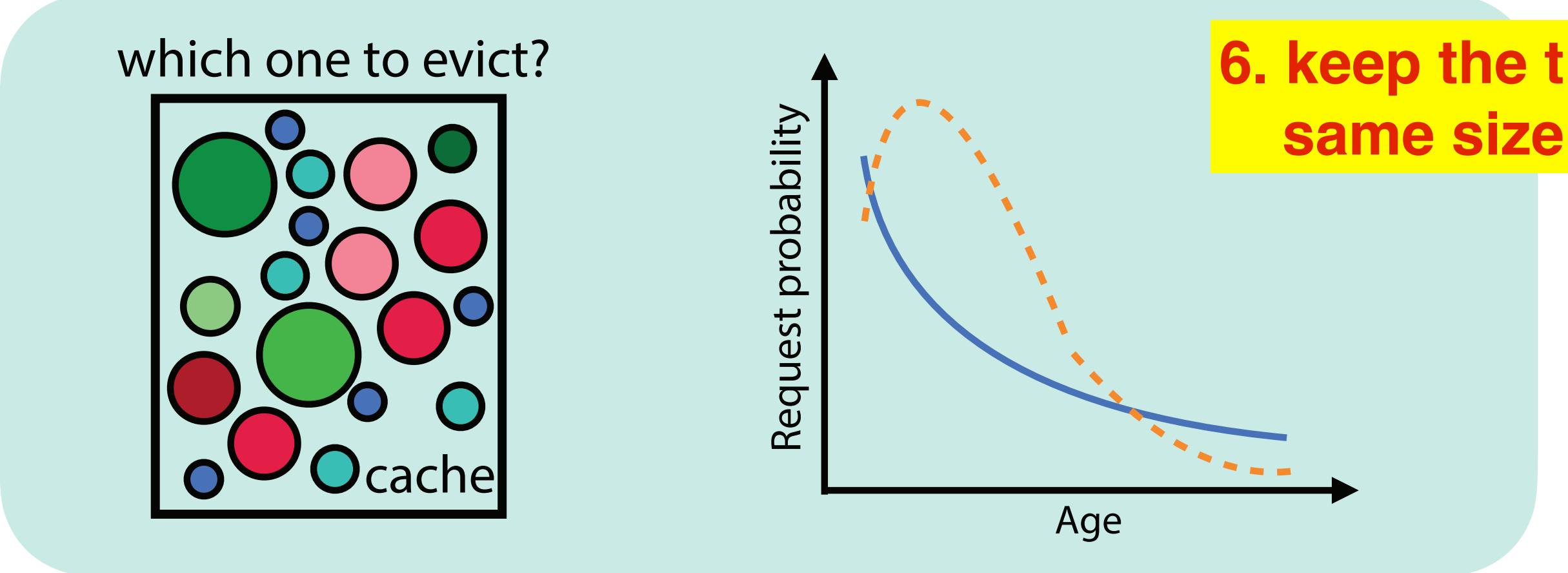
1. Learning from simple experts, e.g., LeCaR ~~and CACHEUS~~ 5. remove "and cacheus"



both efficiency and throughput depend on the experts chosen
maintain two sets of metadata is expensive and complex
delayed reward

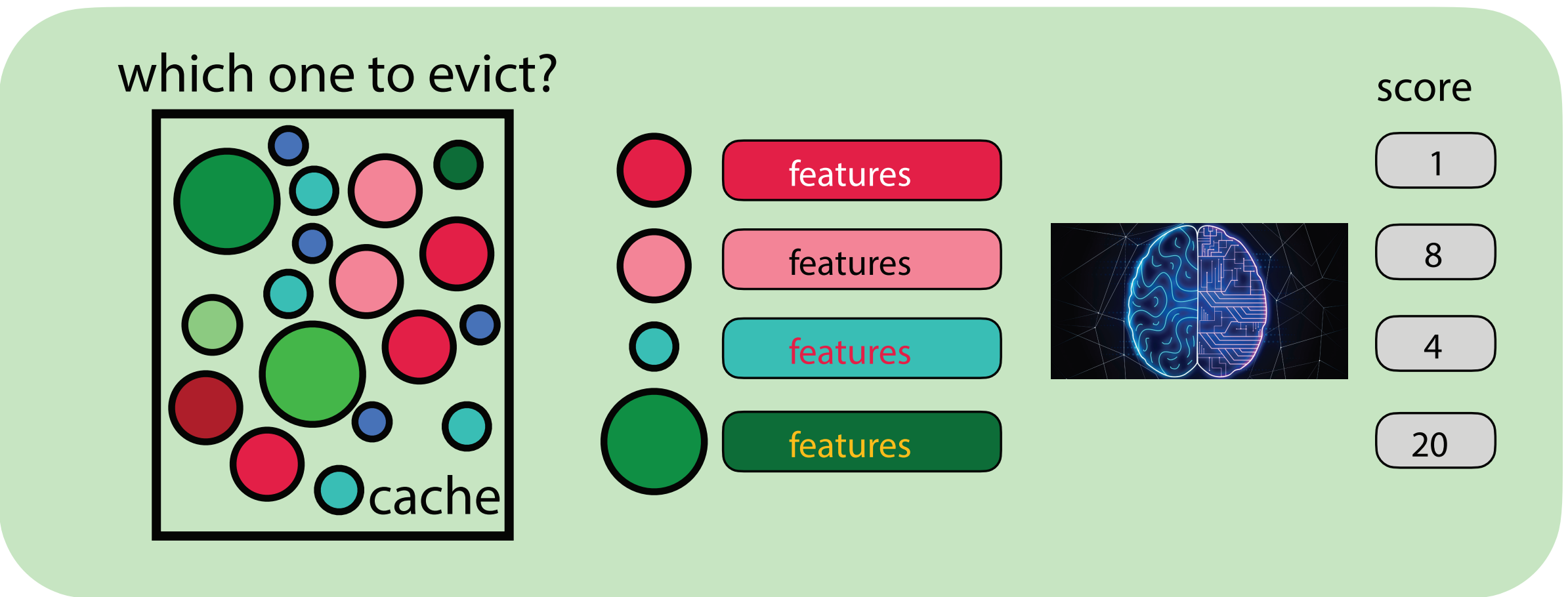
2. Learning from probability distribution, e.g., LHD

6. keep the three figures the same size and aligned?



can only use limited number of features → poor efficiency
require sampling many objects to compare at each eviction → low throughput

3. Object-level learning

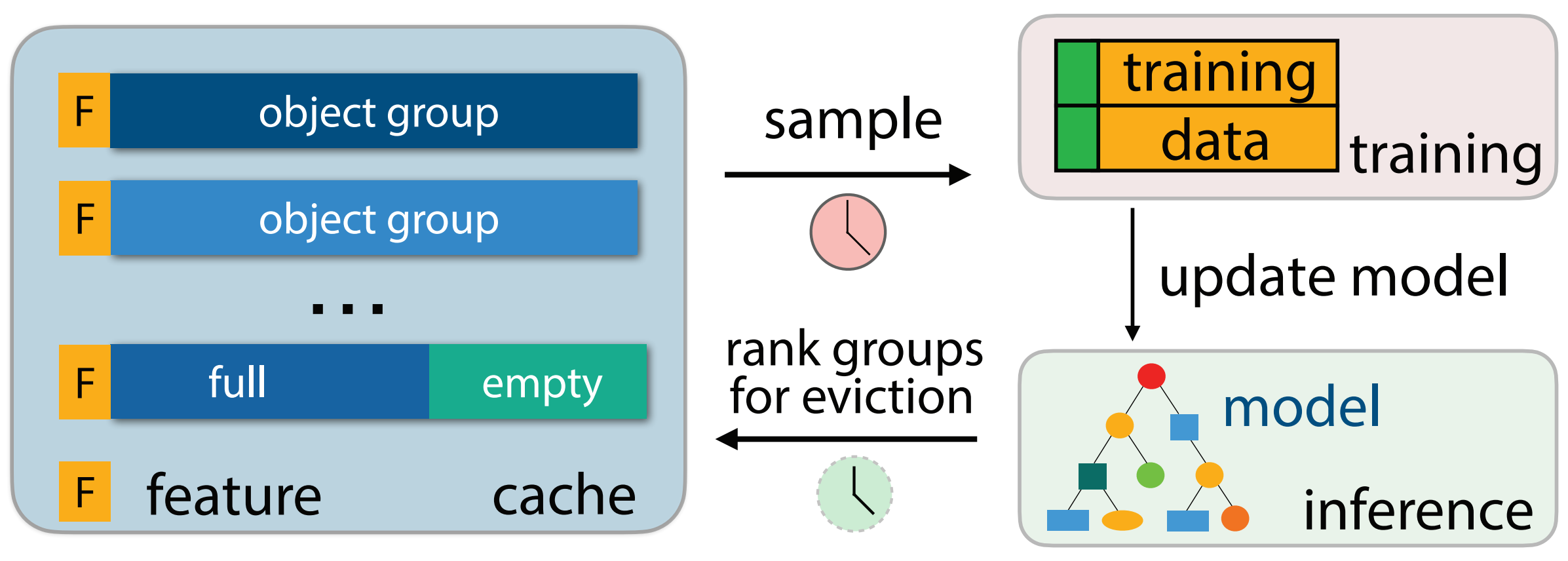


leverage more features than other learned caches
sampling and inference at each eviction → very very very slow

- Existing learned caches either compromise on throughput or cannot leverage multiple features

Carnegie Mellon University

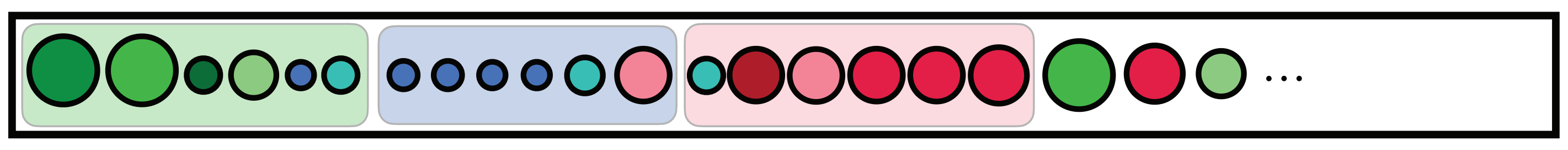
Design of GL-Cache



GL-Cache Architecture

- Many challenges:
  - › How does GL-Cache *group* objects?
  - › *What* and *How* does GL-Cache *learn*?
  - › How does GL-Cache *evict*?

INSERTION-TIME-BASED GROUPING



A NEW UTILITY FUNCTION

- Properties desired:
  - › Larger object → smaller utility
  - › Sooner-to-be-accessed → larger utility
  - › Group size one → Belady's MIN ~~(weighted by size)~~ 3. remove weighted by size
  - › Easy and accurate to track online

object utility at time *t*

$$U_o(t) = \frac{1}{T_o(t) \times s_o}$$

group utility

$$U_{group}(t) = \sum_{o \in group} \frac{1}{T_o(t) \times s_o}$$

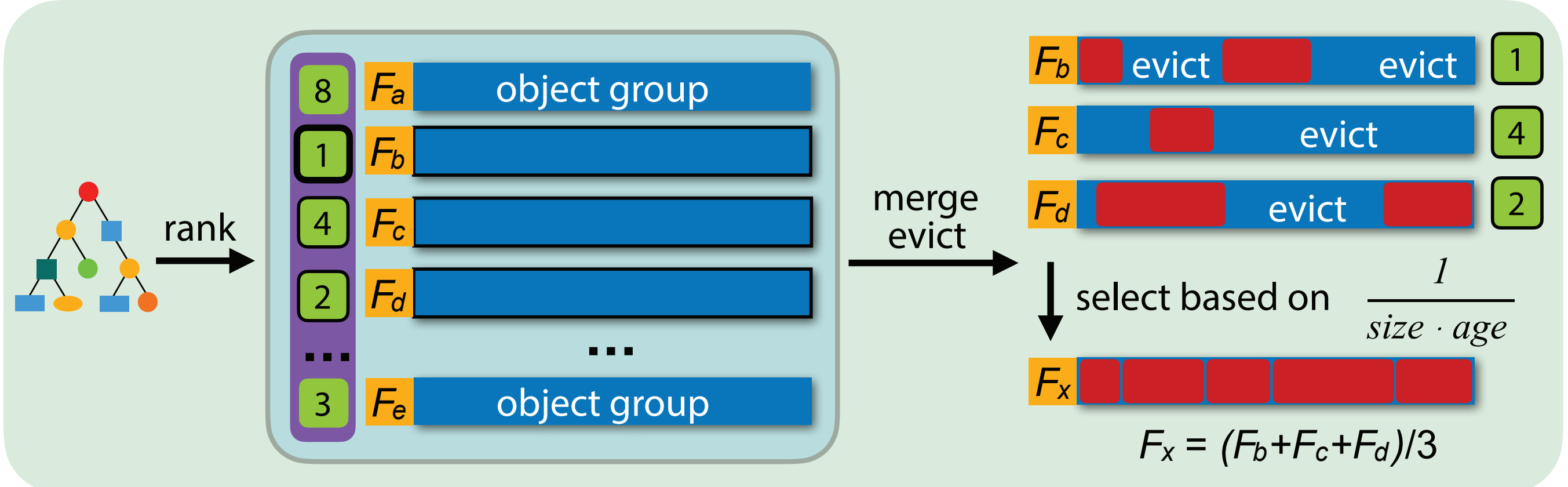
Merge-based eviction

LEARNING IN GL-CACHE

- Static + dynamic features: write rate, miss ratio, request rate, mean object size, age, # requests, # active objects
- Model: gradient-boosting trees
- Objective: regression
- Eviction in GL-Cache

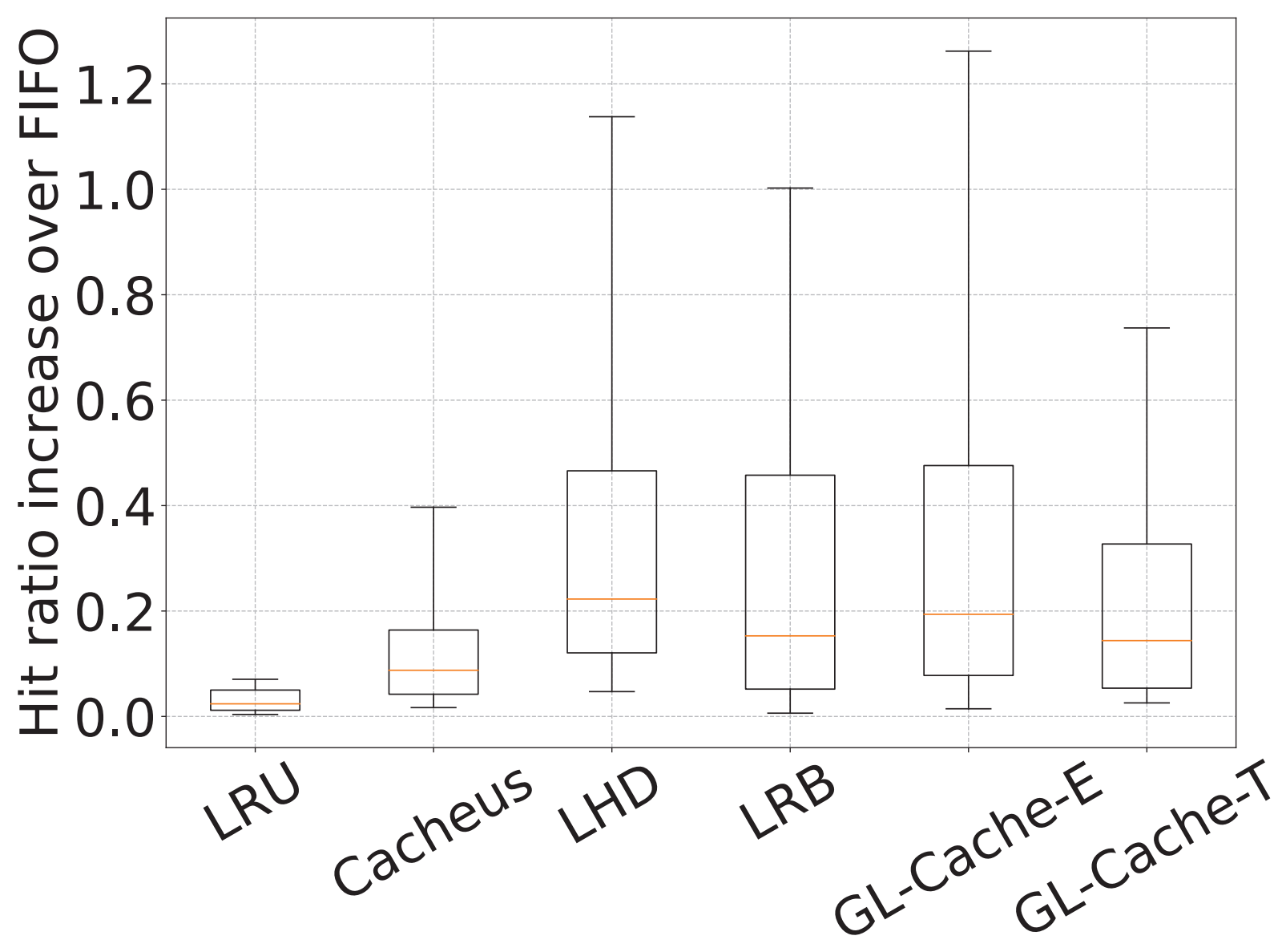
1. Let's make this the same foramt as "learning in GL-Cache"

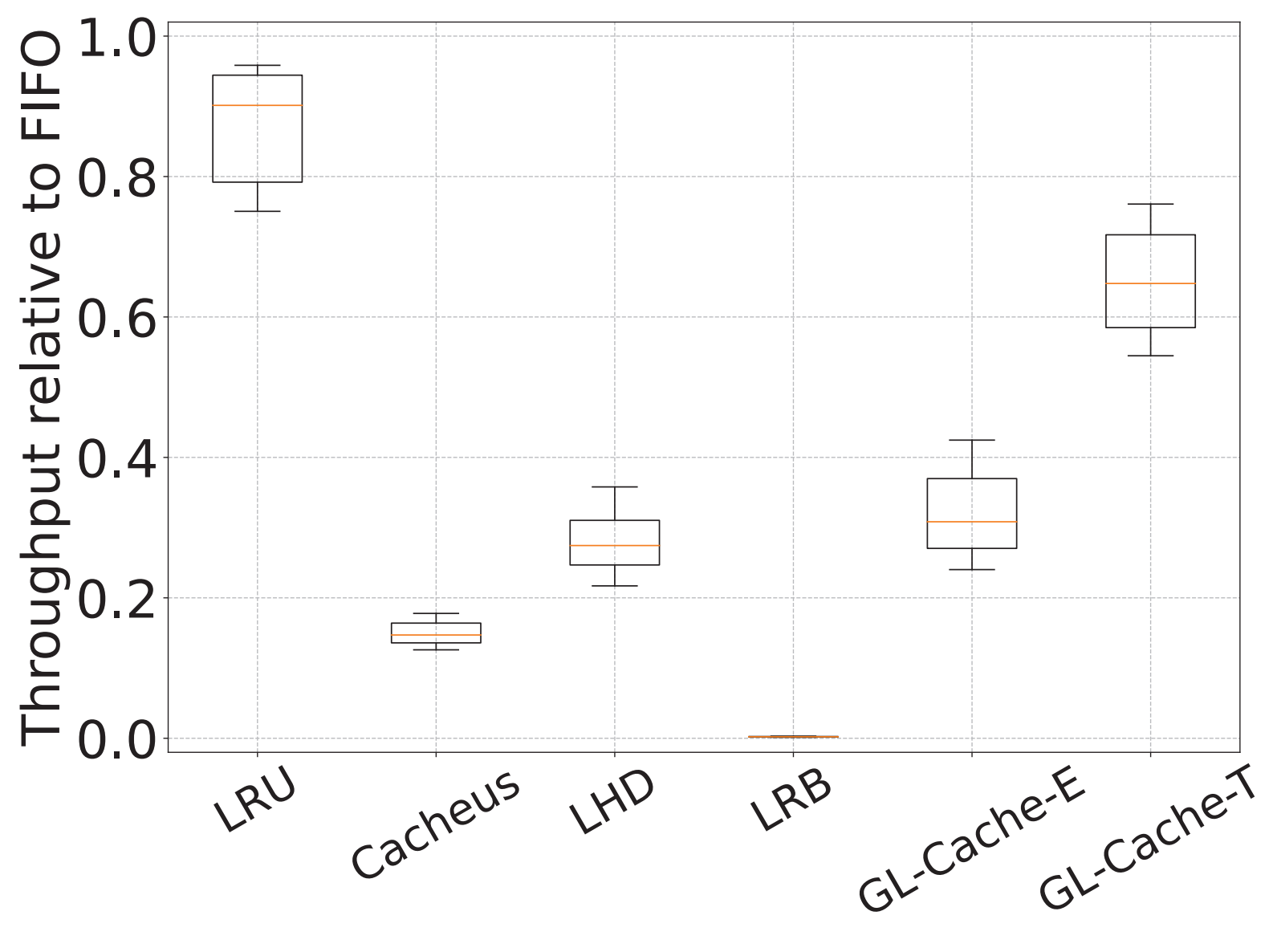
~~A UTILITY FUNCTION TO MEASURE~~ 2. remove this "a utility…"



Evaluation

- Efficiency
  - › GL-Cache-E is slightly better than state-of-the-art algorithms
  - › GL-Cache-T is close to LRB





- Throughput
  - › GL-Cache-E is faster than all state-of-the-art algorithms
  - › GL-Cache-T is significantly faster

Summary

- Group-level learning
  1. Amortizes the overhead of learning, and
  2. Accumulates more information for learning

Parallel Data Laboratory