33rd International Conference on Massive Storage Systems and Technology (MSST 2017)
Santa Clara, CA, May 15 — 19, 2017

FRD: A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance

Sejin Park* and Chanik Park**

2017. 05. 18

*SK telecom Corporate R&D Center | Network IT Convergence R&D Center,
New Computing Lab

**POSTECH Department of Computer Science and Engineering,
System Software Lab

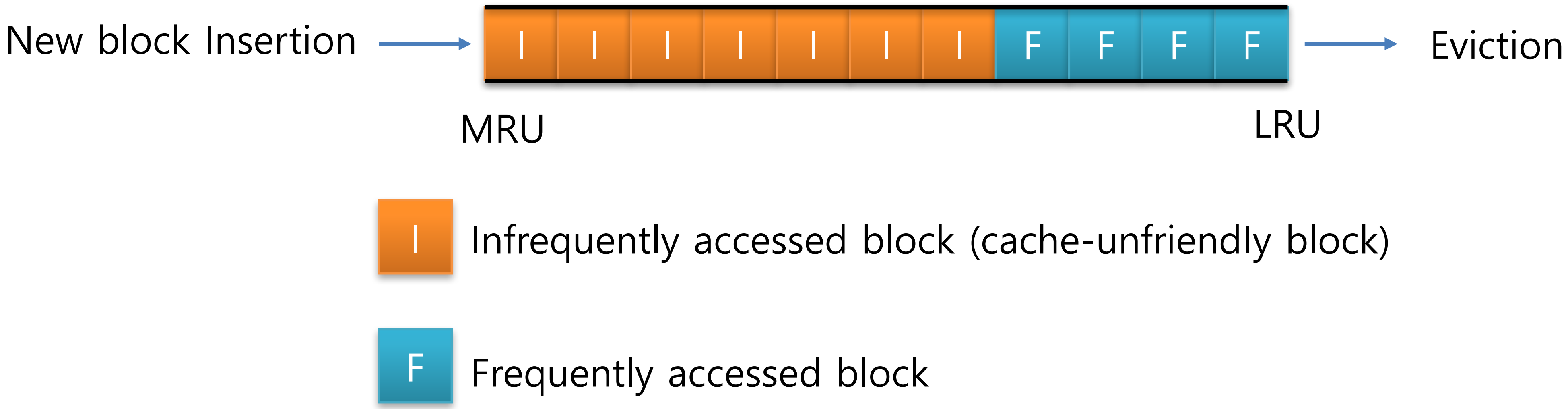
Table of Contents

Motivation

- Buffer cache management algorithm is one of the oldest topic in computer science area
- Existing buffer cache algorithm concentrates on how to maintain meaningful blocks?
  - LRU, LFU, OPT, …
  - LIRS (ACM SIGMETRICS 2002, S. Jiang. et. al.)
    - Two LRU Stacks (LIRS, HIRS)
      - Reuse distance ordering
  - ARC (USENIX FAST 03, Megiddo. et. al.)
    - Two LRU Stacks (Recency-T1, Frequency-T2)
      - Adaptive resizing
- In this study, we concentrate on how to exclude the cache-unfriendly blocks
  - We analyzed real-world workload and found characteristics of cache-unfriendly blocks

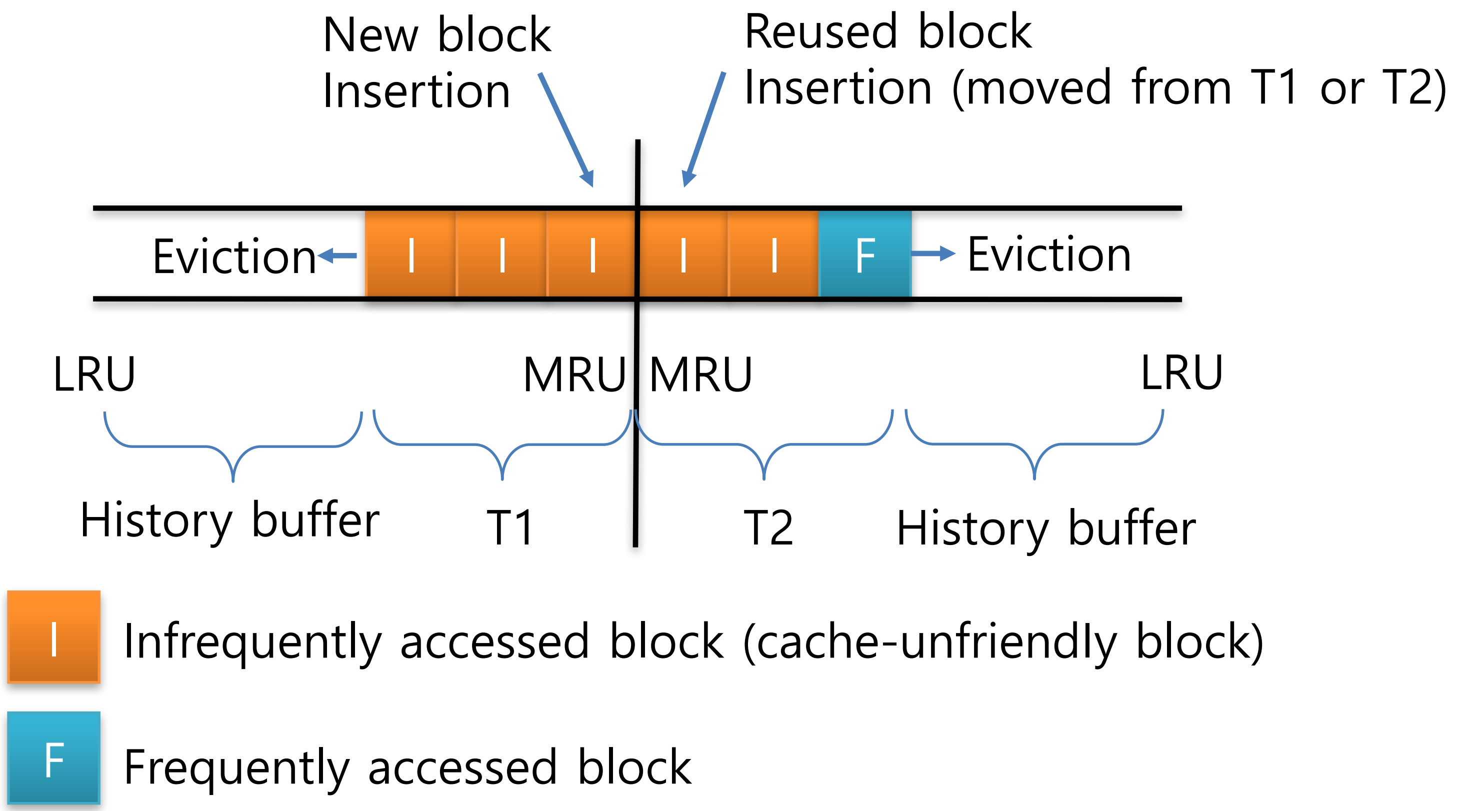
Example: LRU

- Depending on their eviction policy, blocks that can make cache pollution could be maintained in cache space
- LRU believes that recently used blocks will make more cache hit
  - **If the recently used blocks are infrequently accessed and rarely used, it causes cache pollution!**



Example: ARC

- Recency buffer T1 and Frequency buffer T2 in ARC works as LRU cache
- If a block is reused, it moves into T2 even if it is infrequently accessed block
  - **This can cause cache pollution for T2**



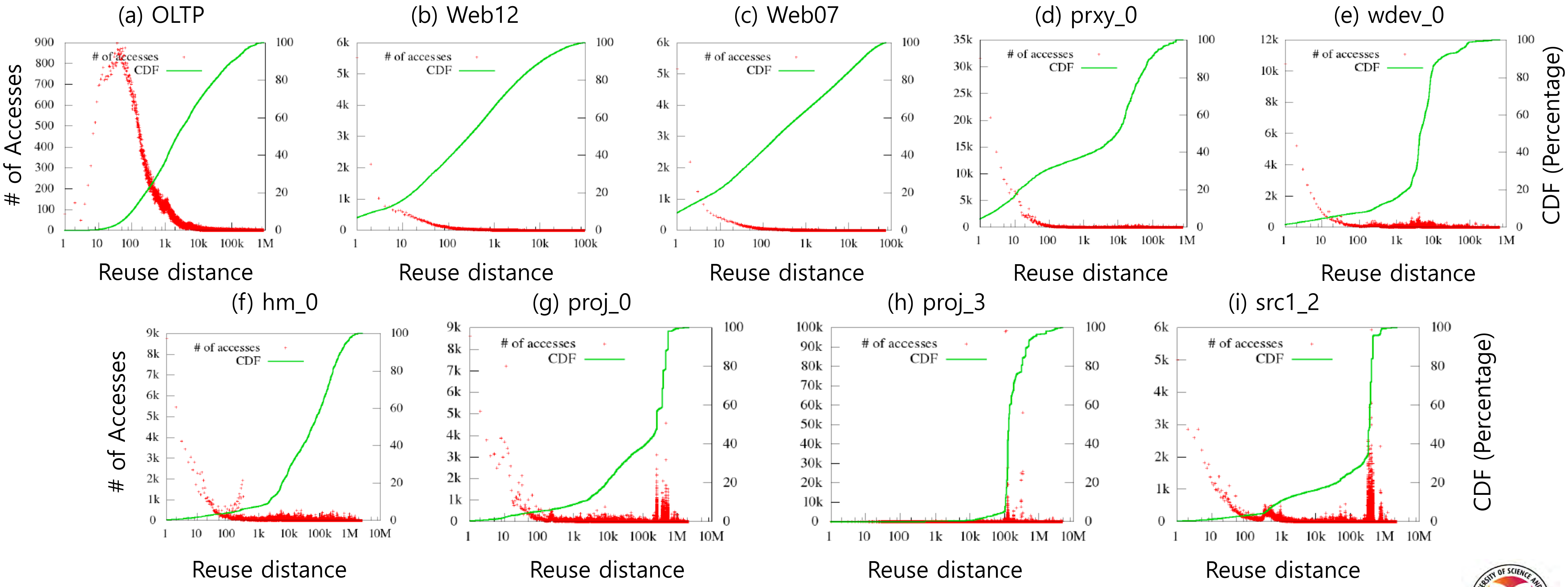
Workload Description

- Real-world workloads downloaded from SNIA.

Name	Type	Description
OLTP	Application	Online transaction processing
Web12	Web server	A typical retail shop
Web07	Web server	A typical retail shop
prxy_0	Data center	Firewall/web proxy
wdev_0	Data center	Test web server
hm_0	Data center	Hardware monitoring
proj_0	Data center	Project directories
proj_3	Data center	Project directories
src1_2	Data center	Source control

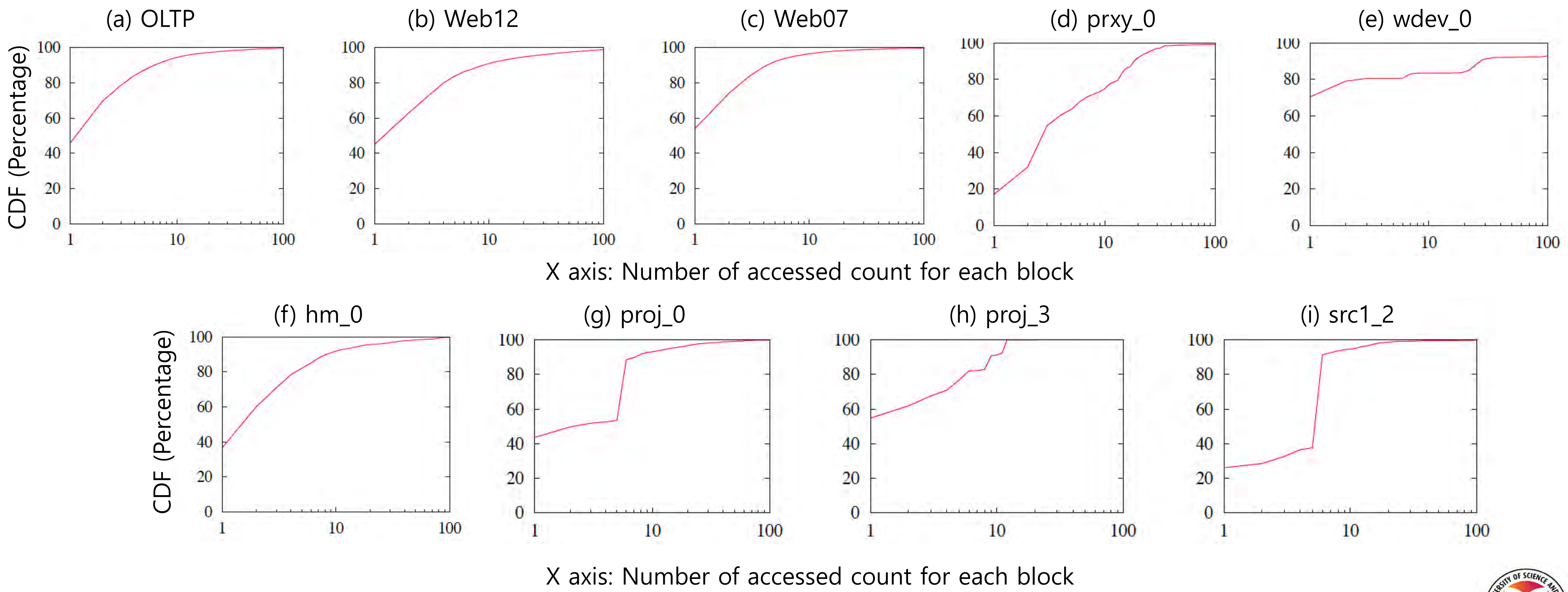
Workload Analysis

- Reuse Distance Distribution
  - Reuse Distance: # of unique blocks between the same blocks request



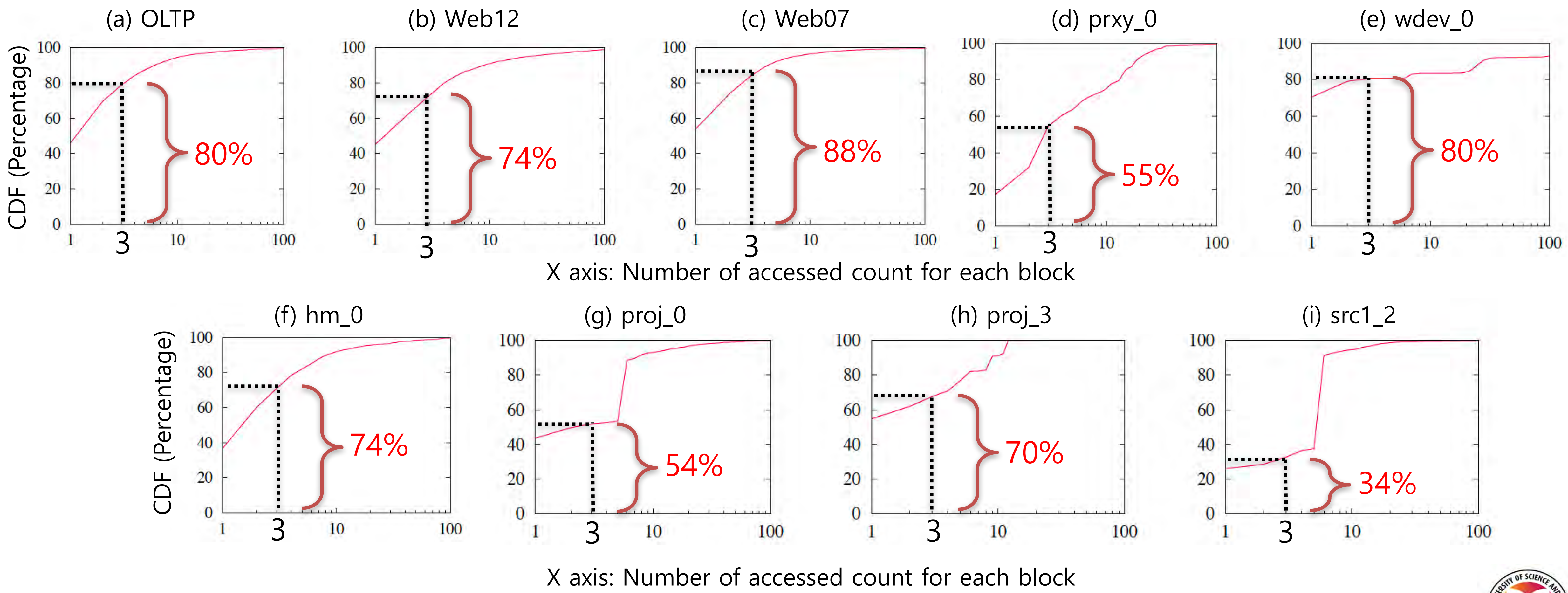
Workload Analysis

- CDF of Number of accessed count for each block



Workload Analysis

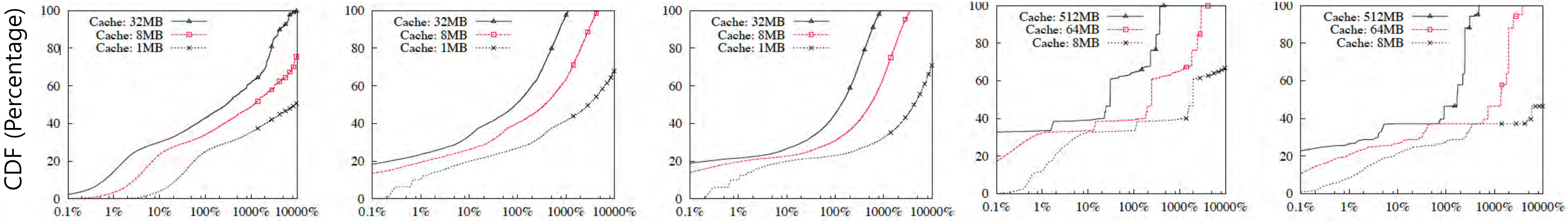
- Observation #1: Most blocks (about 50 – 90%) are infrequently accessed in the real-world workload.



Workload Analysis

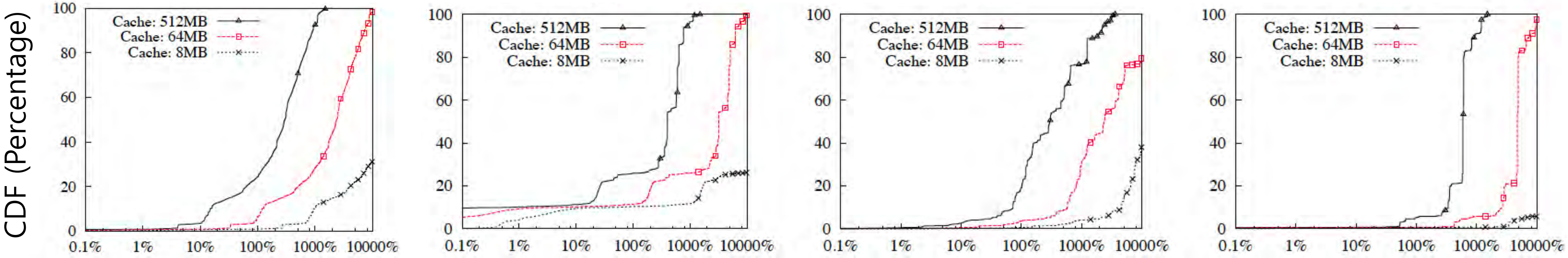
- CDF of reuse distance distribution for the infrequently accessed blocks (represented by percentage of cache size)

(a) OLTP (b) Web12 (c) Web07 (d) prxy_0 (e) wdev_0



X axis: Reuse distance (represented with percentage of given cache size)

(f) hm_0 (g) proj_0 (h) proj_3 (i) src1_2

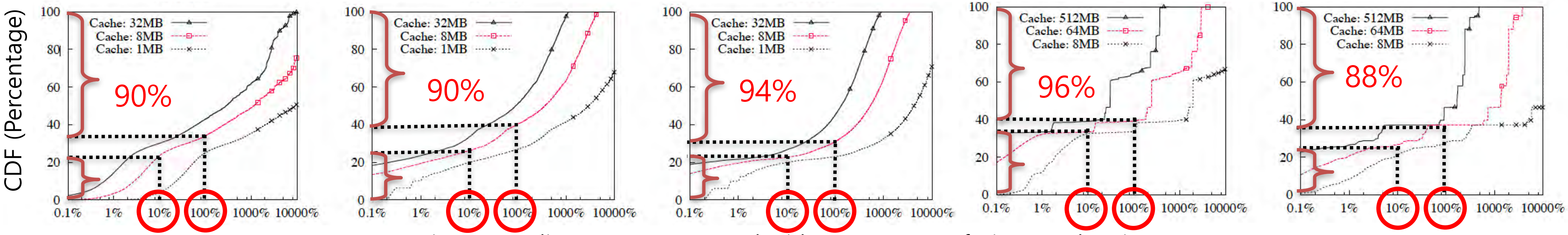


X axis: Reuse distance (represented with percentage of given cache size)

Workload Analysis

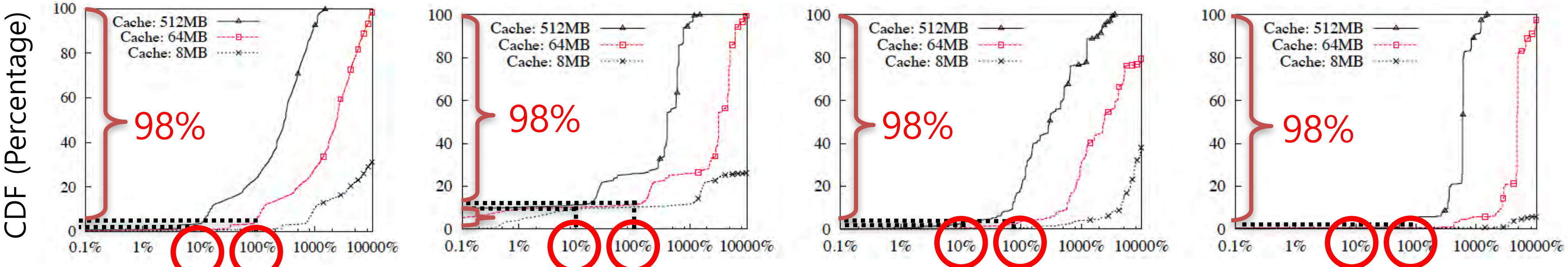
- Observation #2: Reuse distance for the infrequently accessed blocks is extremely long or extremely short
  - In terms of cache size: under 10% and over 100% of cache size are dominant

(a) OLTP (b) Web12 (c) Web07 (d) prxy_0 (e) wdev_0



X axis: Reuse distance (represented with percentage of given cache size)

(f) hm_0 (g) proj_0 (h) proj_3 (i) src1_2



X axis: Reuse distance (represented with percentage of given cache size)

Observations

- Observation #1: Most blocks are infrequently accessed in the real-world workload
  - These blocks are cache-unfriendly blocks that cause cache pollution
- Observation #2: Reuse distance for the infrequently accessed blocks is extremely long or extremely short
  - The cache-unfriendly blocks have distinct characteristics
- Therefore,
  - "Frequency" and "Reuse distance" are the key metrics to filter out the cache-unfriendly blocks

Design

- Block Classification

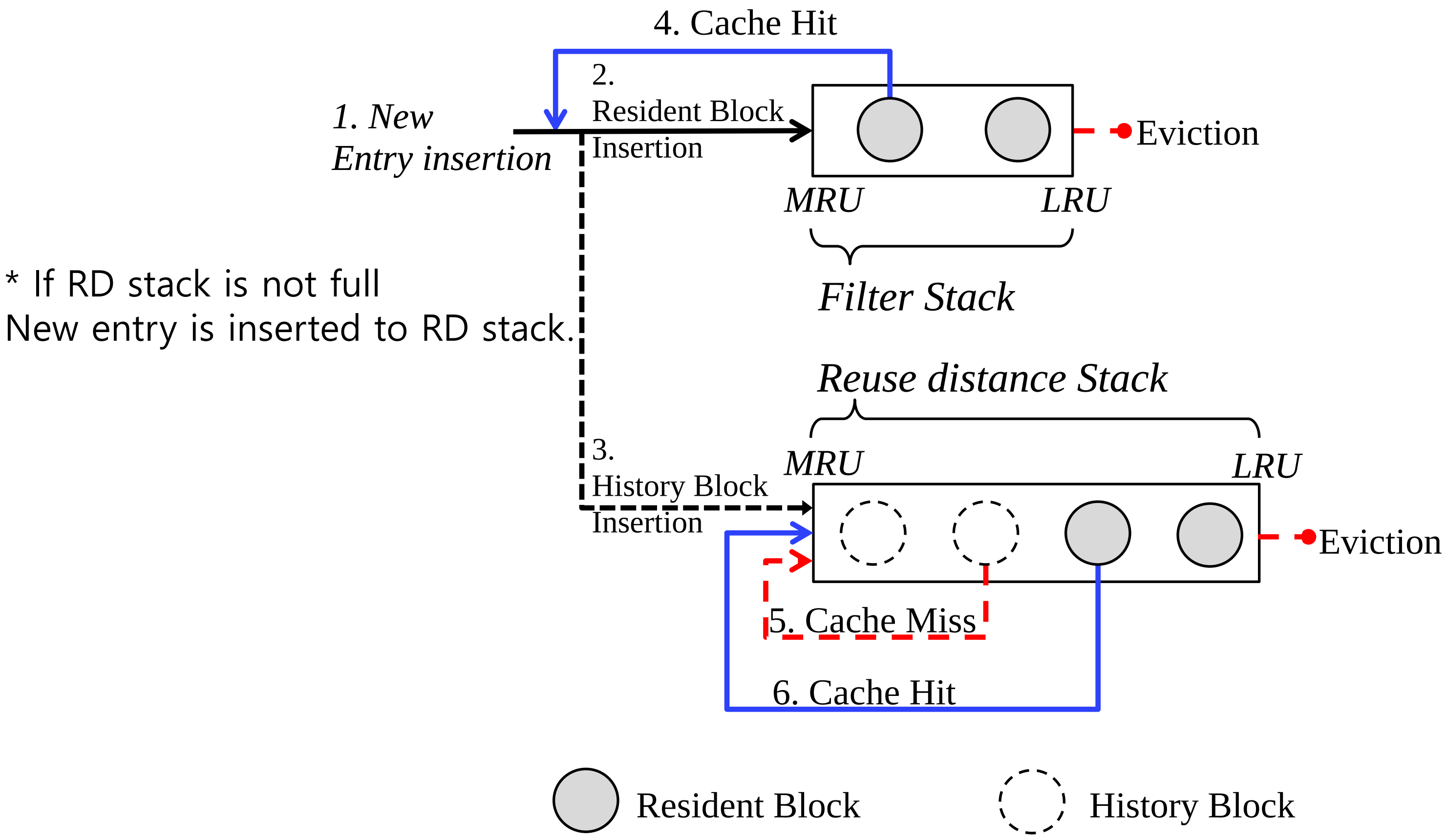
Class	Accessing Frequency	Reuse Distance	Cache-Hit Target	Cache Pollution (Filtering target)
Class 1 (FS)	Frequent	Short	V	-
Class 2 (FL)	Frequent	Long	V	-
Class 3 (IS)	Infrequent	Short	V	V
Class 4 (IL)	Infrequent	Long	-	V

- Design Goal
  - Maintains Class 1 and 2 blocks in cache
  - Maintains Class 3 blocks but preventing it from polluting cache
  - Filters out Class 4 blocks from cache

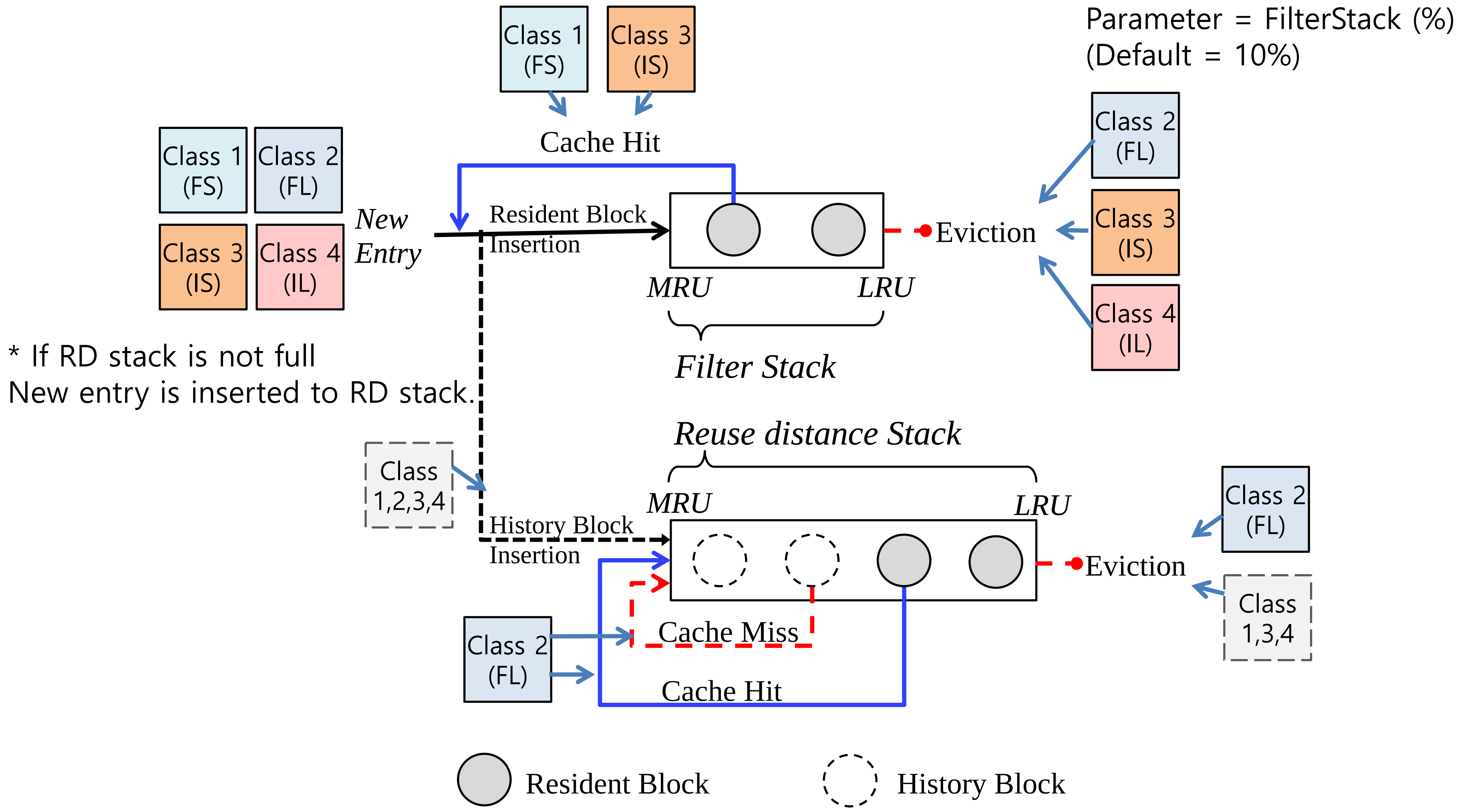
FRD Algorithm

- A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance

Parameter = FilterStack (%)
(Default = 10%)



Analysis of FRD Algorithm



Evaluation

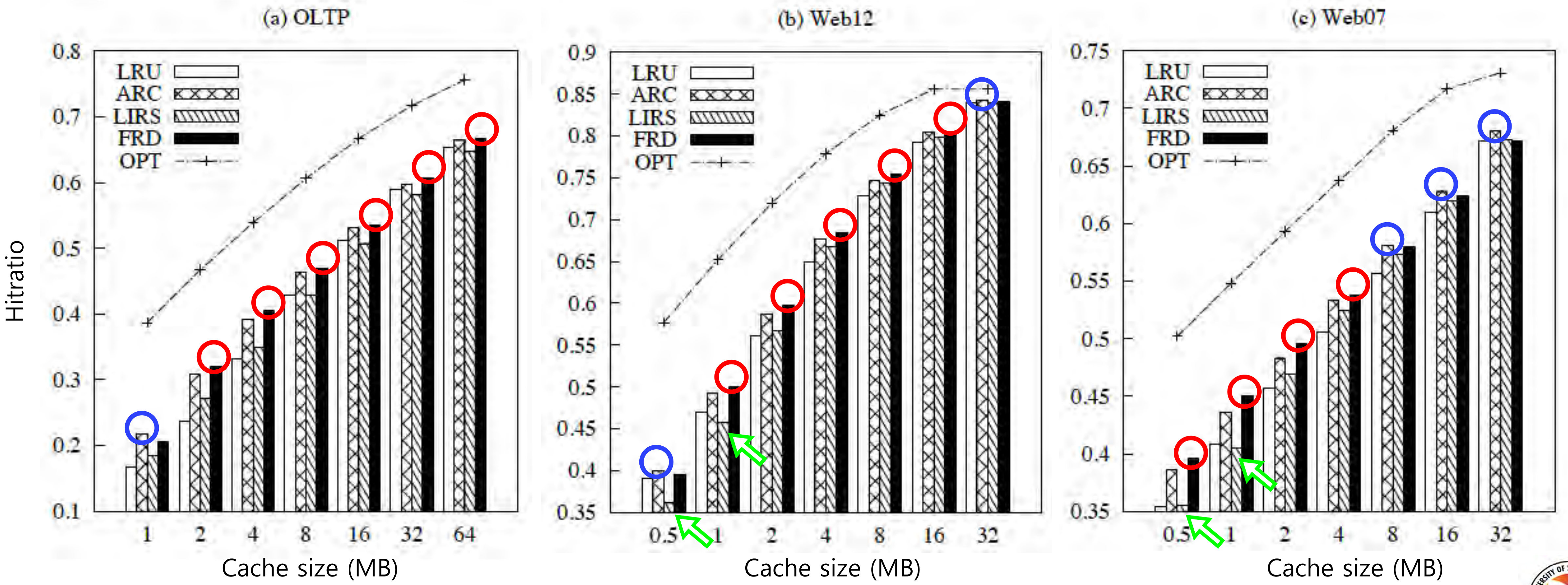
- Environment
  - Simulation based evaluation
  - Compared with OPT, LRU, ARC, LIRS

Hitratio Result

- Case of LIRS' unstable hitratio result

<Legend>

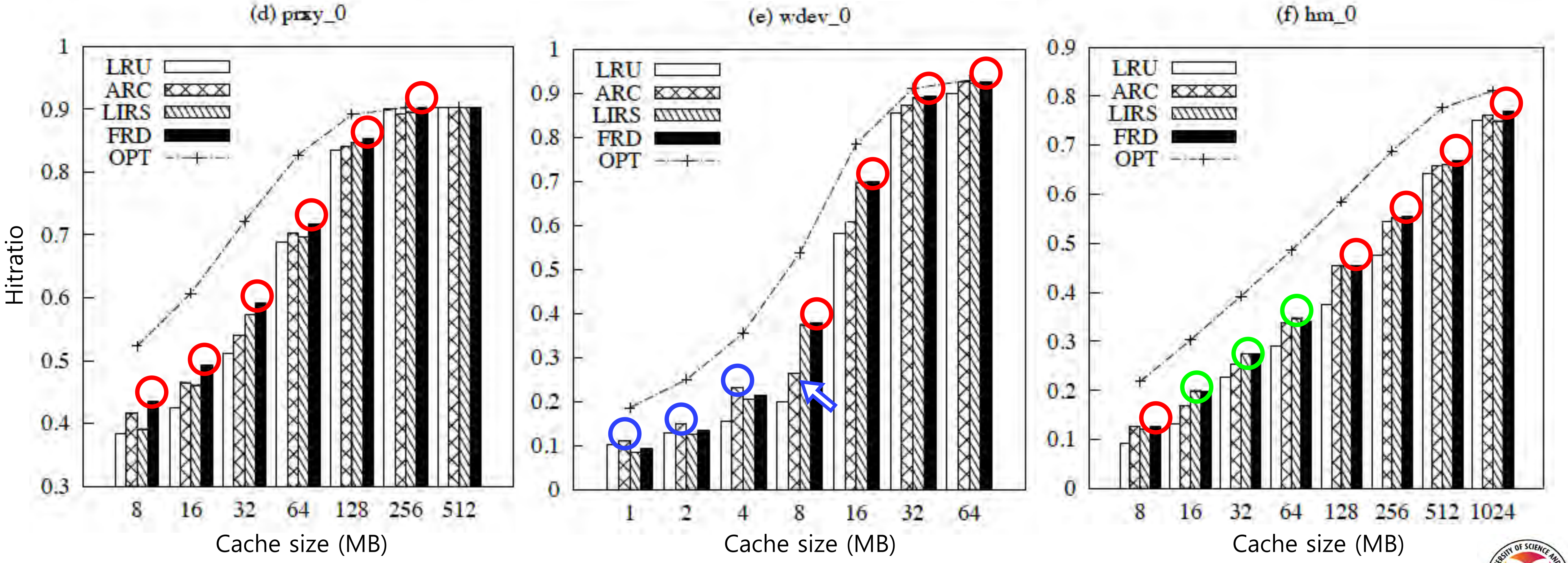
- FRD is highest
- LIRS is highest
- ARC is highest
- LIRS is unstable
- ARC is unstable



Hitratio Result

<Legend>

- FRD is highest
- LIRS is highest
- ARC is highest
- LIRS is unstable
- ARC is unstable



Hitratio Result

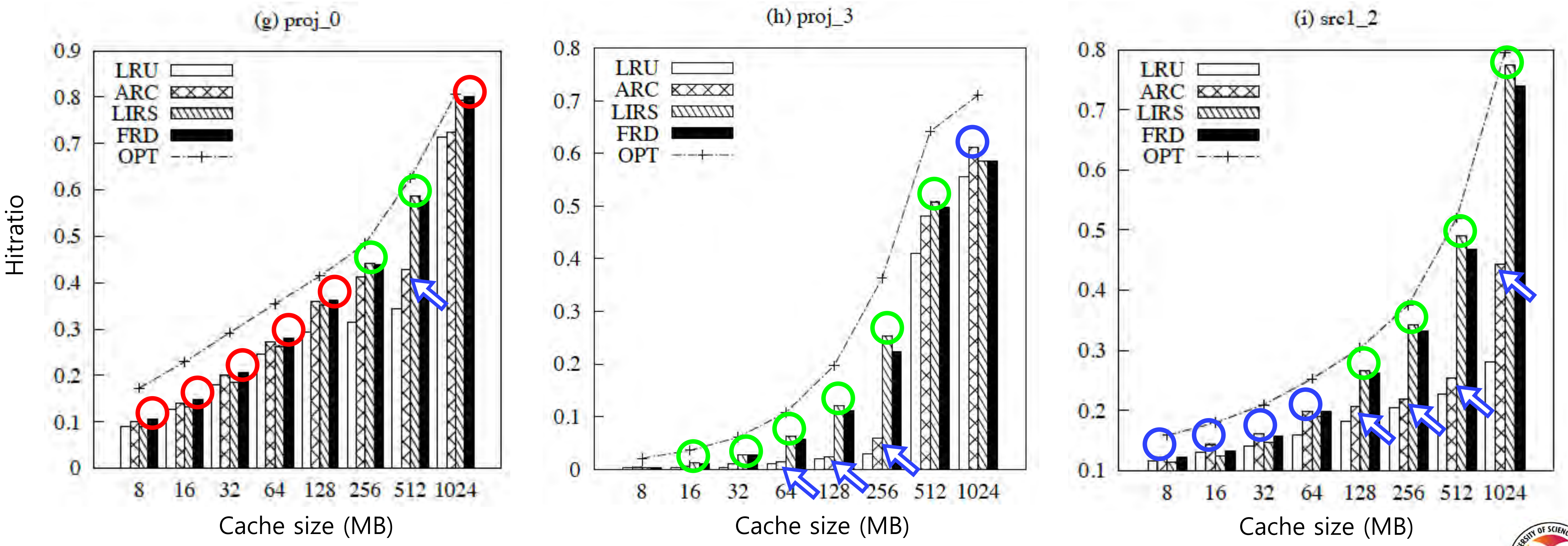
- Case of ARC's unstable hitratio result

<Legend>

FRD is highest

LIRS is highest — LIRS is unstable

ARC is highest — ARC is unstable



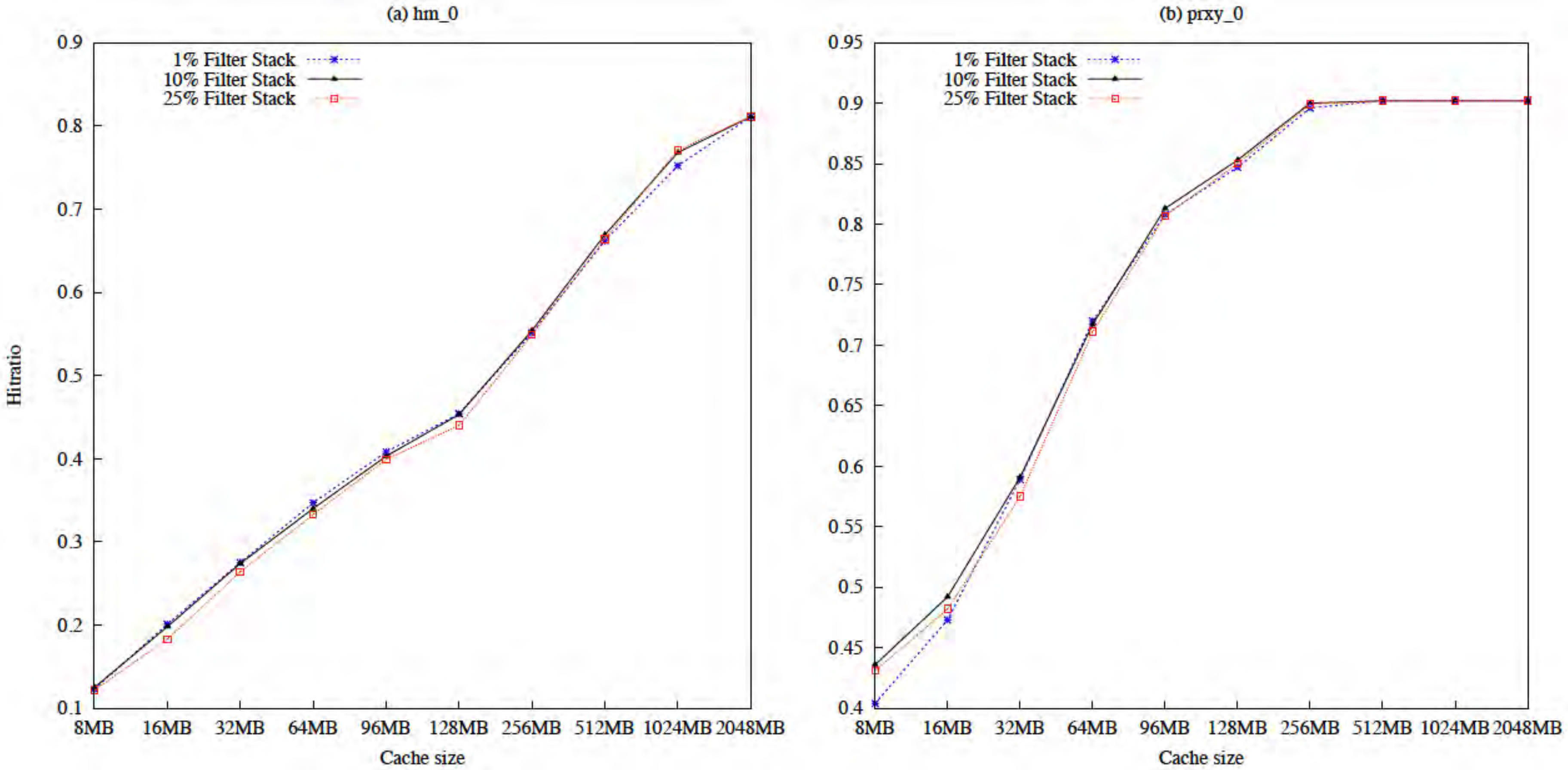
Evaluation

- Overall Average Result (1.0 is OPT's hitratio)

Workload	LRU	ARC	LIRS	FRD
OLTP	0.674	0.746	0.691	**0.753**
Web12	0.829	0.852	0.827	**0.857**
Web07	0.800	0.839	0.812	**0.847**
prxy_0	0.844	0.870	0.870	**0.898**
wdev_0	0.647	0.723	0.728	**0.745**
hm_0	0.598	0.700	0.723	**0.724**
proj_0	0.612	0.722	0.740	**0.780**
proj_3	0.172	0.241	**0.516**	0.478
src1_2	0.620	0.697	0.799	**0.813**

Parameter Sensitivity (Size of the Filter stack)

- Variation of filter stack size from 1% to 25% of cache size.
- 10% shows the best performance on average but the difference is negligible.



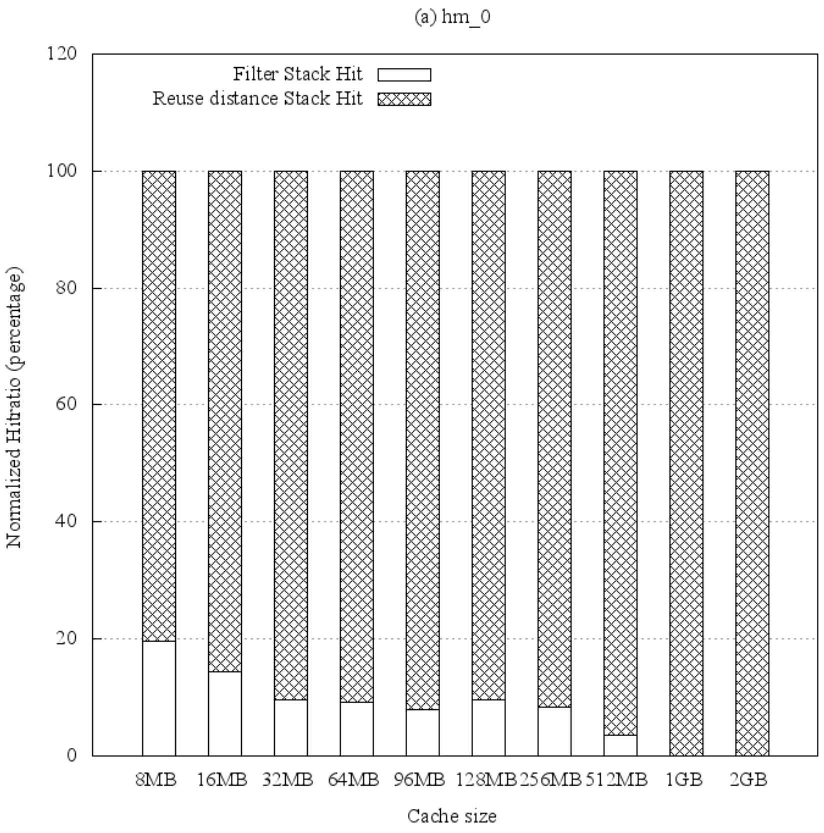
Summary

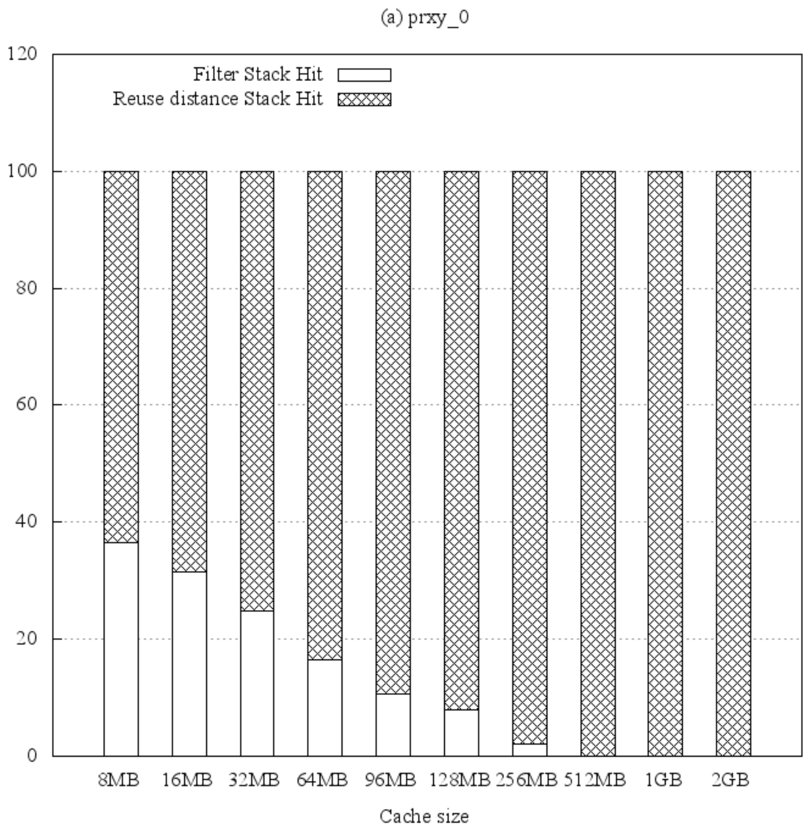
- FRD: A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance
  - A new buffer cache algorithm that filters out cache-unfriendly blocks
  - Careful analysis on real-world workload gives characteristics of cache-unfriendly blocks
  - The experimental result shows that it outperforms state-of-the-art cache algorithms like ARC or LIRS.

Backup slides

Hitratio Analysis

- Filter stack performance

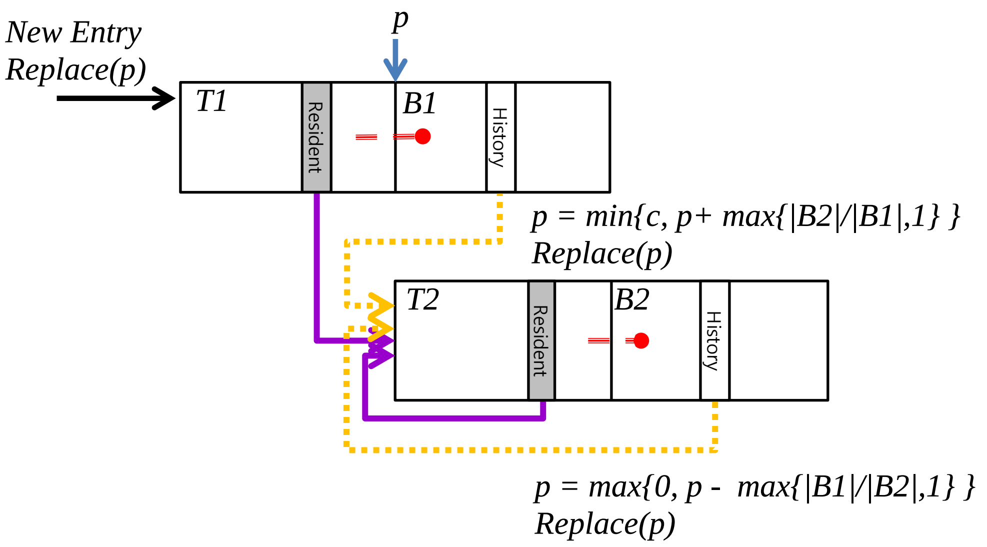




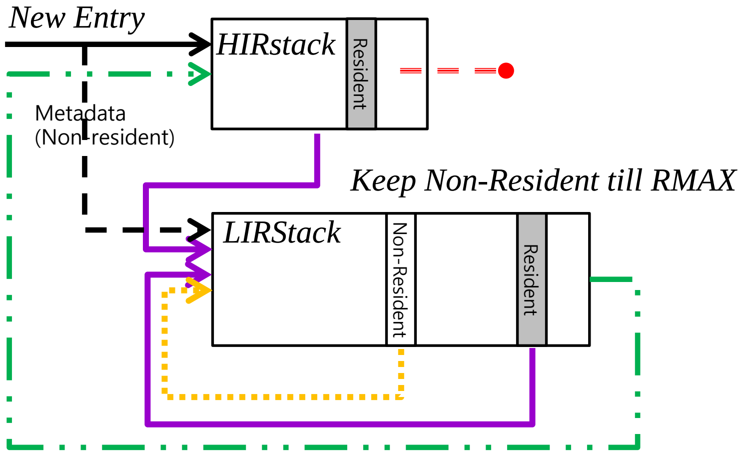
Revisiting LIRS and ARC

ARC (Initial: $T1 = T2 = B1 = B2 = 0, p = 0$)

$T1 + T2 + B1 + B2 \le 2C$



LIRS (HIRstack + LIRstack = c, 1:99)



Subroutine ***Replace***(*p*)

if ($|T1| \ge 1$) and (($x \in B2$ and $|T1| = p$) or ($|T1| > p$)) then move the **LRU** page of *T1* to the top of *B1* and remove it from the cache.

else move the **LRU** page in *T2* to the top of *B2* and remove it from the cache.

HIT MISS NEW ENTRY Metadata(History) Eviction Flow

Design comparison with ARC and LIRS

	ARC	LIRS	FRD
# LRU stack	Two	Two	Two
Adaptive Resizing	O	X	X
Eviction Point	Two (Two LRU stacks are isolated)	One (Two LRU stacks are **not** isolated)	Two (Two LRU stacks are isolated)
History size	Cache size x 2	Max resident block	Max resident block