

Jin-Soo Kim  
([jinsoo.kim@snu.ac.kr](mailto:jinsoo.kim@snu.ac.kr))

Systems Software &  
Architecture Lab.  
Seoul National University

Fall 2020

# ARC

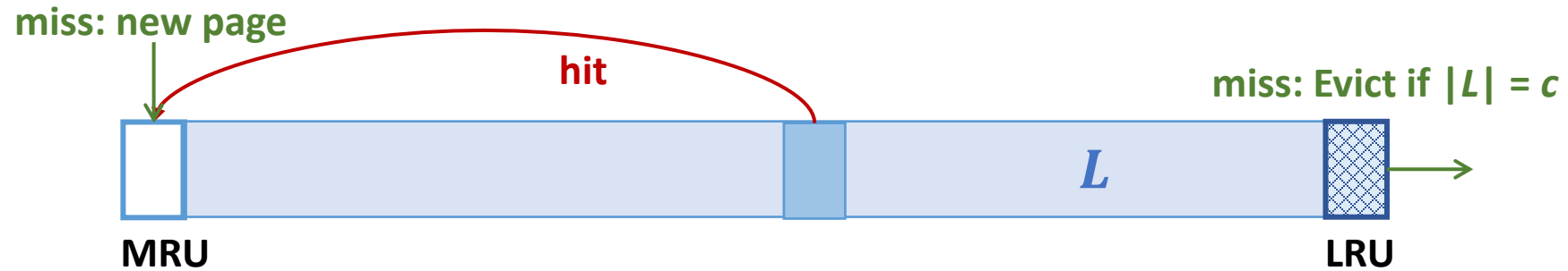
(N. Megiddo and D. S. Modha, FAST '03)



# LRU vs. LFU

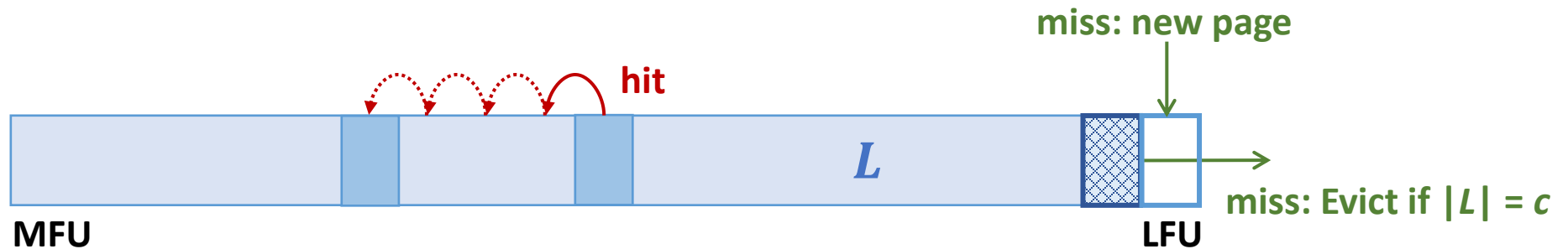
- LRU (Least Recently Used)

- $O(1)$  complexity, no frequency considered



- LFU (Least Frequently Used):

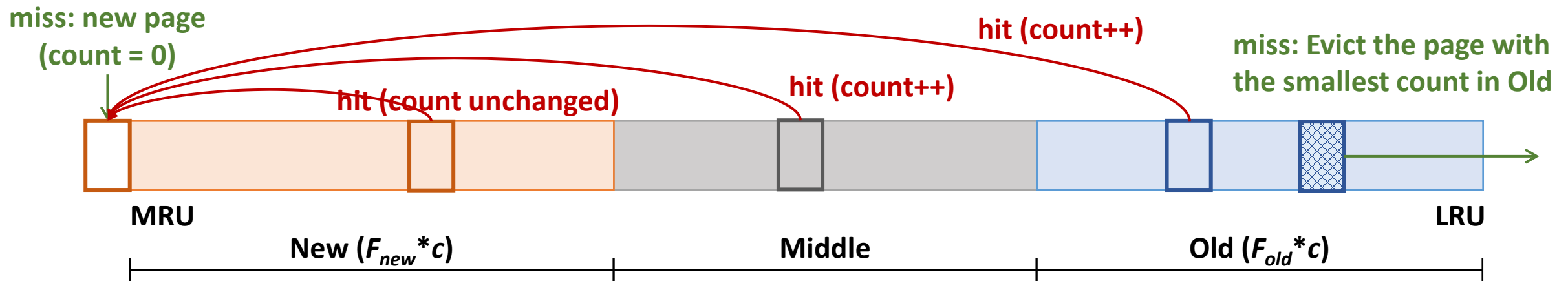
- $O(\log n)$  complexity, no attention to recent history, no adaptability



# FBR [SIGMETRICS '90]

## Frequency-Based Replacement

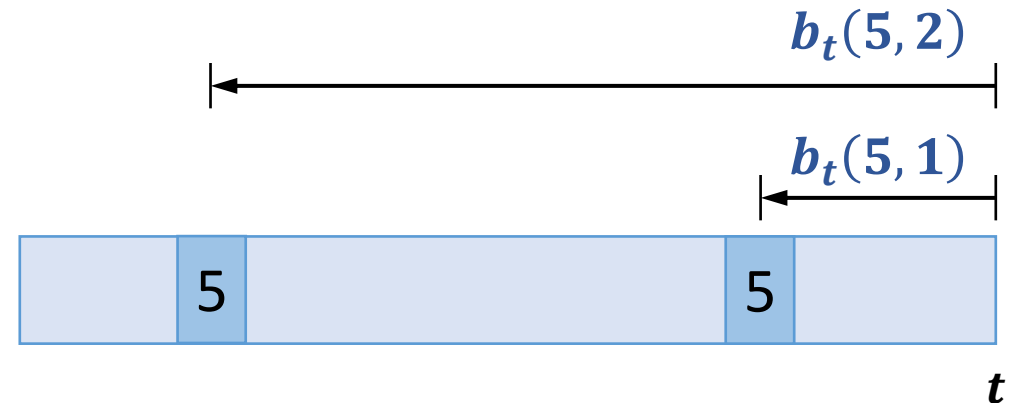
- Three sections in the LRU list: New, Middle, and Old
- Replaces the page with the smallest reference count in the Old section ( $O(\log n)$ )
- Need to rescale the reference counters: Whenever the average reference counter exceeds  $A_{max}$ , every reference counter is reduced to  $\lceil C/2 \rceil$
- Tunable parameters:  $F_{new}$ ,  $F_{old}$ ,  $A_{max}$



# LRU-2 [SIGMOD '93]

## ■ LRU-K

- Backward  $K$ -distance  $b_t(p, K)$ :  
the distance backward from the time  $t$  to the  $K$ -th most recent reference to the page  $p$  ( $\infty$  if  $p$  does not appear  $K$  times)
- Replace the page whose backward  $K$ -distance is the maximum of all pages
- LRU-1 = classical LRU



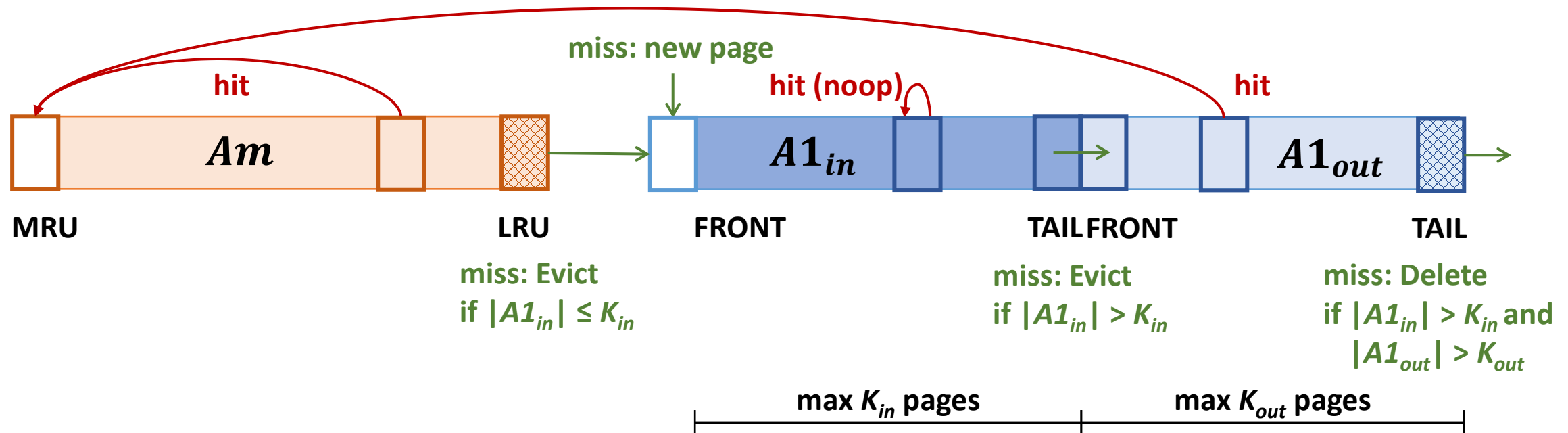
## ■ $O(\log n)$ complexity

## ■ A crucial tunable parameter: Correlated Information Period (CIP)

- The time a page that has only been seen once recently should be kept in the cache to avoid "Early Page Replacement"

# 2Q [VLDB '94]

- $A_m$  for hot pages
- $A1_{in}$  for pages of potentially correlated accesses
- $A1_{out}$  for pages that have been accessed once (metadata only)
- $O(1)$  complexity, two tunable parameters:  $K_{in}$  and  $K_{out}$



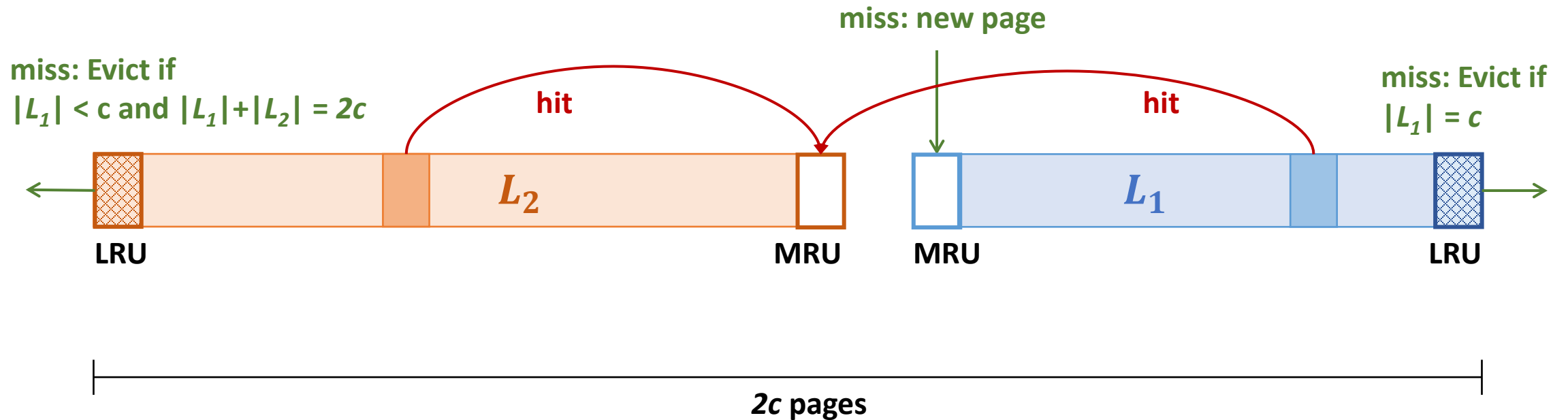
# LRFU [SIGMETRICS '99]

- Least Recently/Frequently Used
- Each block  $x$  is associated with the CRF (Combined Recency and Frequency) value or  $C(x)$ :
$$C(x) = \begin{cases} 1 + 2^{-\lambda}C(x) & \text{if } x \text{ is referenced at time } t \\ 2^{-\lambda}C(x) & \text{otherwise} \end{cases}$$
- Replace the page with the smallest  $C(x)$ 
  - As  $\lambda$  approaches to 0,  $C(x)$  is simply the number of occurrences of  $x \rightarrow$  LFU
  - As  $\lambda$  approaches to 1,  $C(x)$  emphasizes recency  $\rightarrow$  LRU
- The performance depends crucially on  $\lambda$  (adaptive version exists)
  - Other tunable parameter:  $c$  (for correlated references)
- Computation overhead (up to 50x over LRU)

# LIRS [SIGMETRICS '02]

- **Low Inter-reference Recency Set**
  - Inter-Reference Recency (IRR) or reuse distance: the number of other blocks accessed between two consecutive references to the block
  - High IRR blocks are candidates for replacements
- **Two stacks are maintained**
  - A large LRU stack ( $L_{lirs}$ ) for low IRR (LIR) resident blocks
  - A small LRU stack ( $L_{hirs}$ ) for high IRR (HIR) blocks
  - The large stack also records resident/nonresident high IRR blocks
- **Stack pruning operation removes the HIR blocks in the stack bottom**
  - Average-case rather than worst-case constant-time overhead
- **Tunable parameters:  $L_{hirs}$  (normally 1%),  $R_{max}$  (for LIR/HIR switching)**

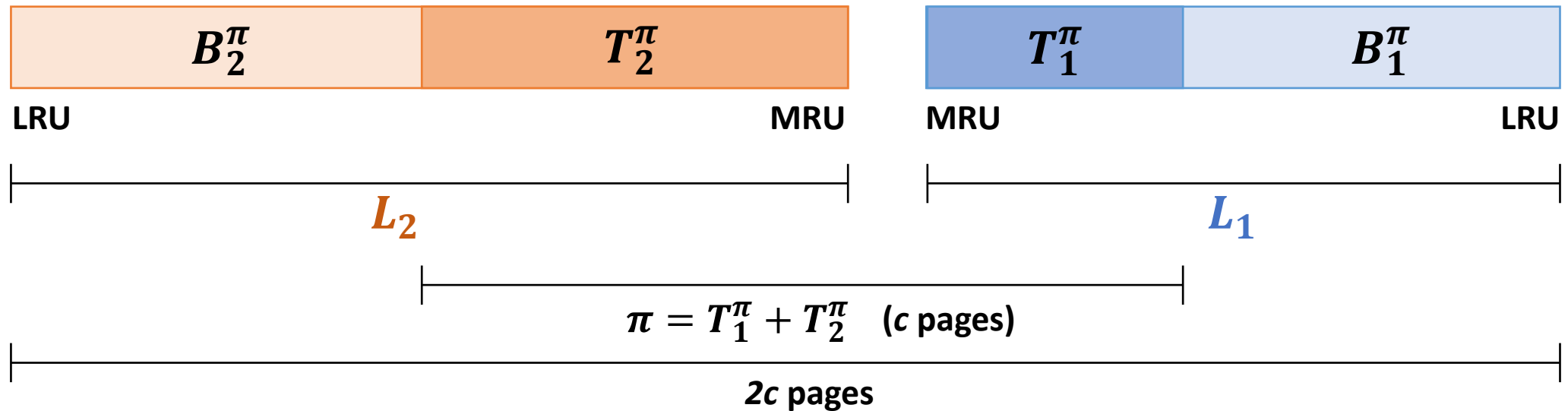
# DBL(2c)



- $0 \leq |L_1| + |L_2| \leq 2c, 0 \leq |L_1| \leq c, 0 \leq |L_2| \leq 2c$

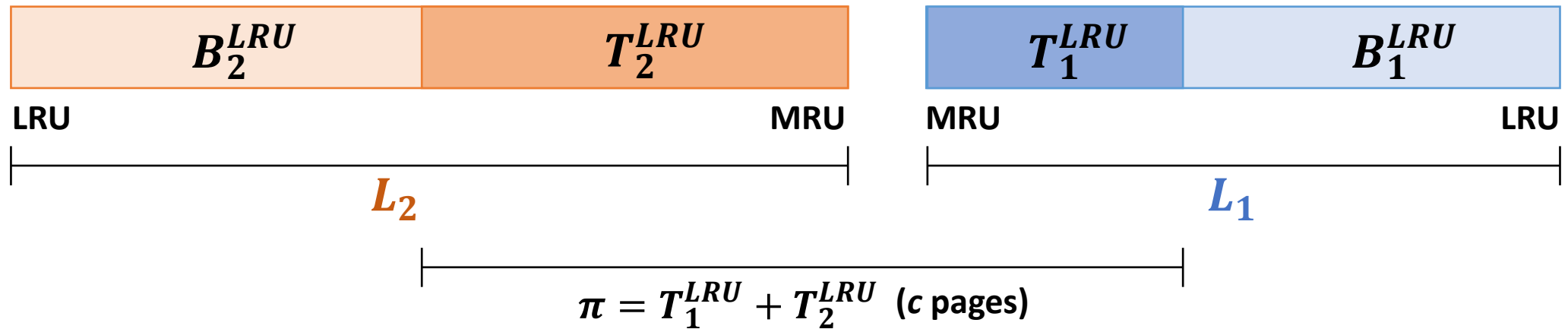


$$\pi(c) \in \Pi(c)$$



- If  $|L_1 \cup L_2| < c$ , then  $B_1^\pi = B_2^\pi = \emptyset$
- If  $|L_1 \cup L_2| \geq c$ , then  $|T_1^\pi \cup T_2^\pi| = c$

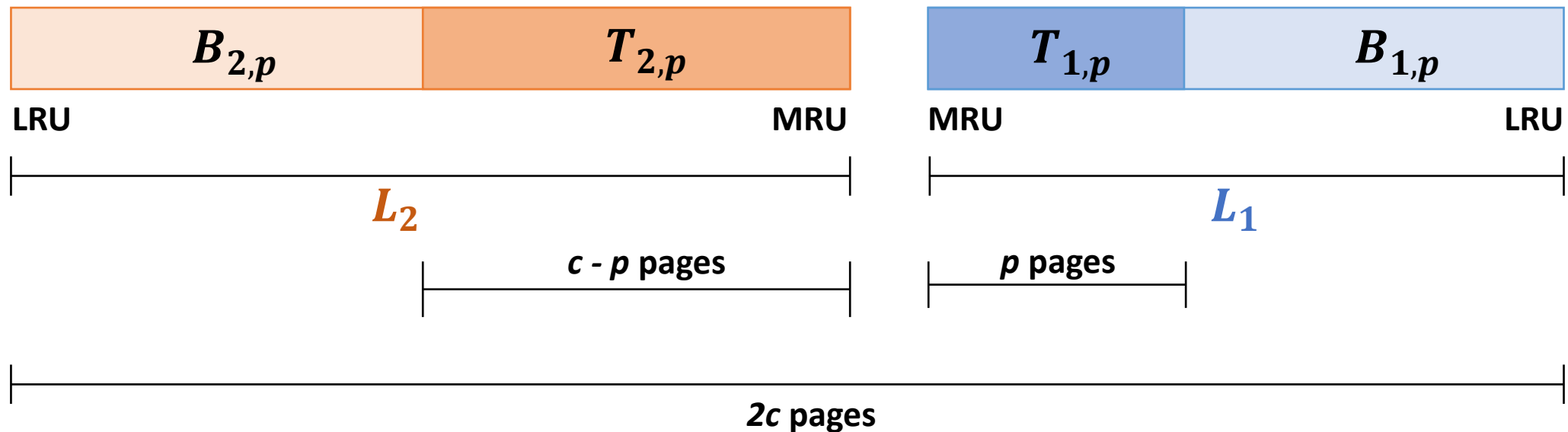
# LRU( $c$ ) $\in \Pi(c)$



- The most recent  $c$  pages will always be in  $DBL(2c)$ 
  - When the LRU item in  $L_1$  is evicted:  $L_1$  must contain exactly  $c$  items
  - When the LRU item in  $L_2$  is evicted:  $L_2$  must contain at least  $c$  items
- There must exist a dynamic partition of lists  $L_1$  and  $L_2$  into lists  $T_1^{LRU}$ ,  $B_1^{LRU}$ ,  $T_2^{LRU}$ , and  $B_2^{LRU}$  such that the conditions for  $\pi(c)$  hold

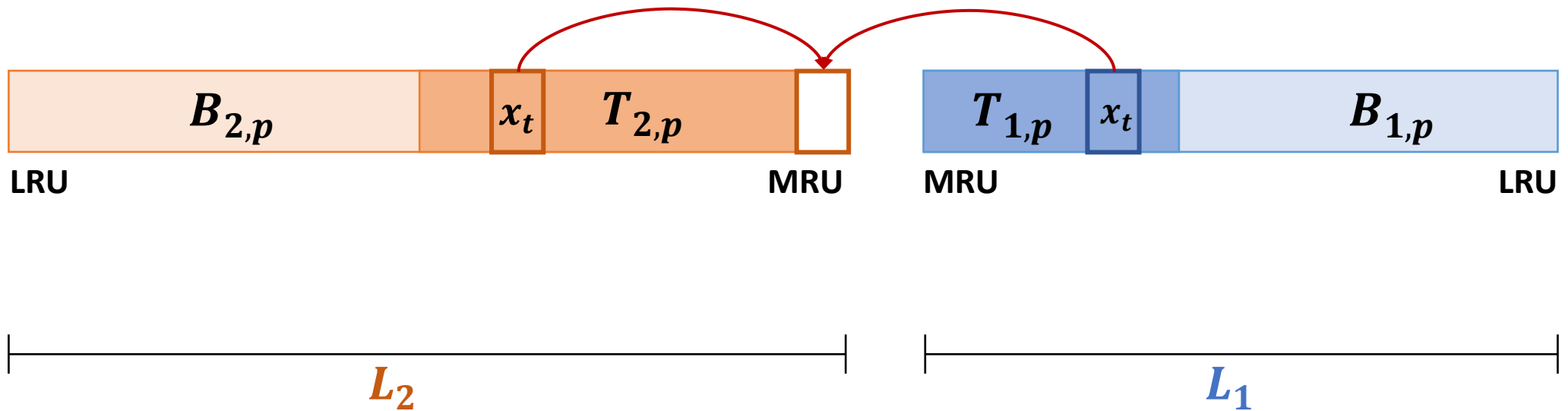
# Fixed Replacement Cache: $FRC_p(c)$

- Keep exactly  $p$  pages in  $T_{1,p}$  and exactly  $c - p$  pages in  $T_{2,p}$
- Parameter  $p$ : target size for  $T_{1,p}$



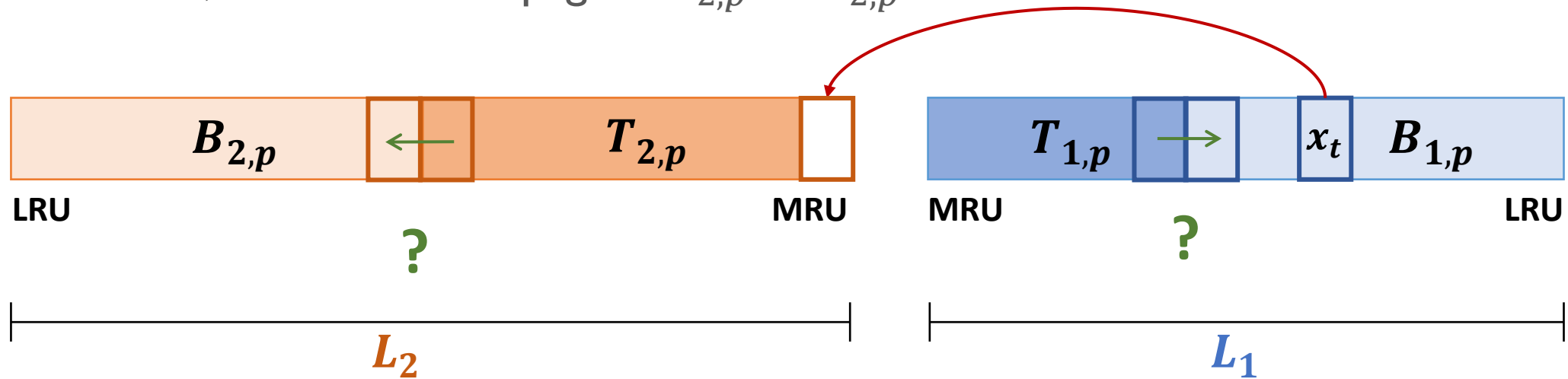
# $FRC_p(c)$ : Cache Hit

- $x_t \in T_{1,p} \cup T_{2,p}$   
→ Move  $x_t$  to MRU position of  $T_{2,p}$



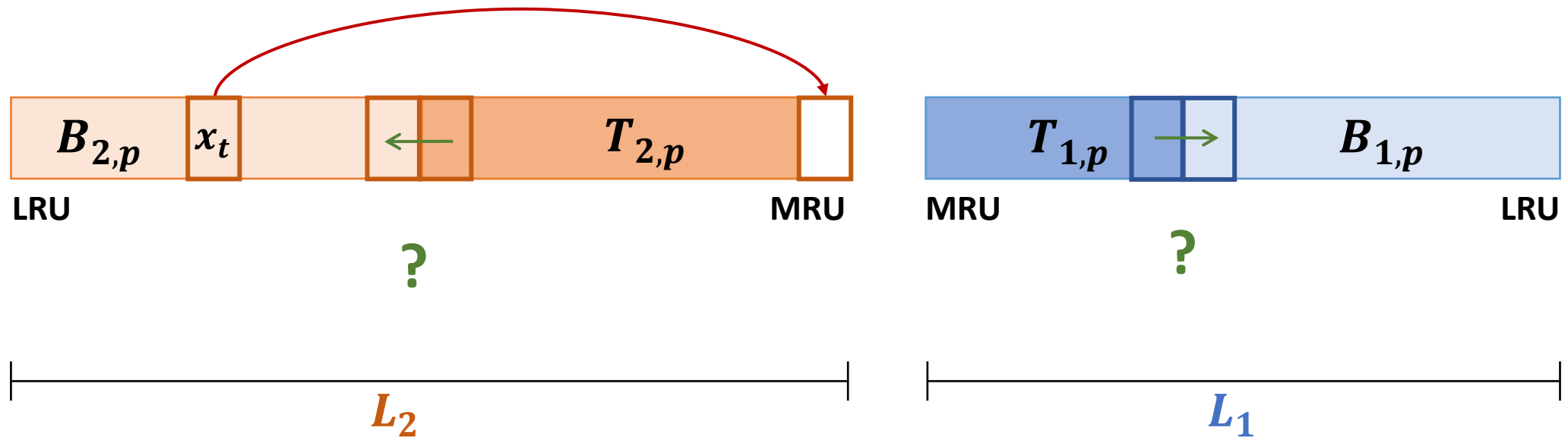
# FRC<sub>p</sub>(c): Cache Miss in $B_{l,p}$

- $x_t \in B_{1,p}$   
 $\rightarrow$   $\text{Replace}(x_t, p)$ , and move  $x_t$  from  $B_{1,p}$  to MRU position of  $T_{2,p}$
- $\text{Replace}(x_t, p)$ 
  - If ( $T_{1,p} \neq \emptyset$  and ( $(|T_{1,p}| > p)$  or ( $x_t \in B_{2,p}$  and  $|T_{1,p}| = p$ ))),  
move the LRU page in  $T_{1,p}$  to  $B_{1,p}$
  - Otherwise, move the LRU page in  $T_{2,p}$  to  $B_{2,p}$



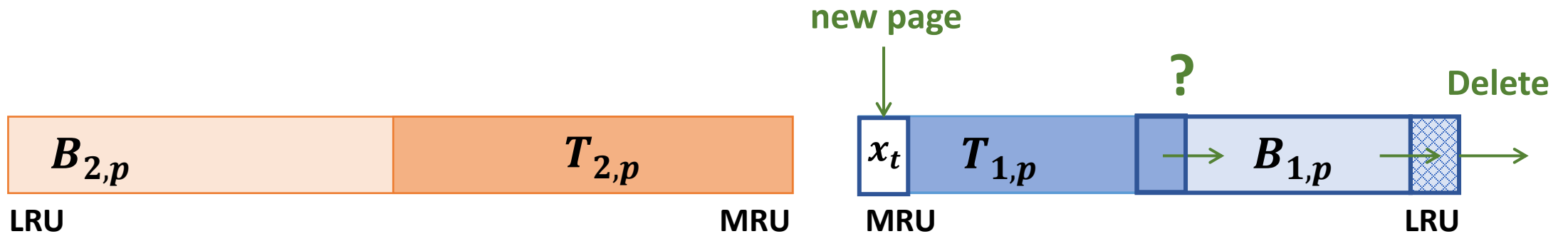
# $FRC_p(c)$ : Cache Miss in $B_{2,p}$

- $x_t \in B_{2,p}$   
→ Replace( $x_t, p$ ), and move  $x_t$  from  $B_{2,p}$  to MRU position of  $T_{2,p}$

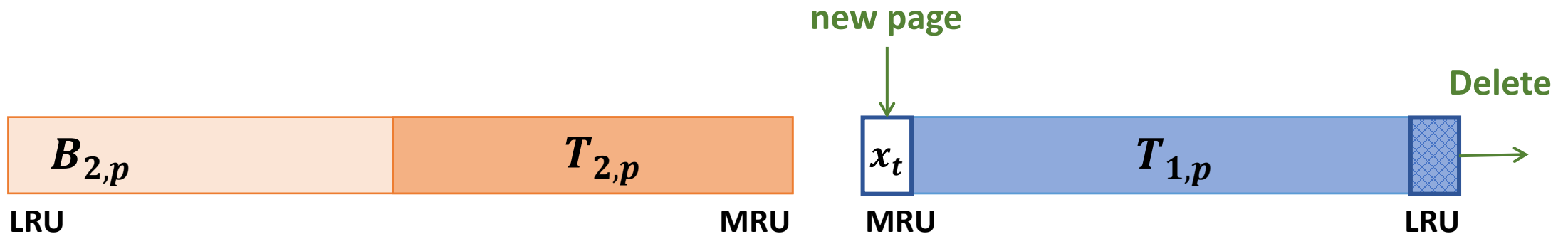


# FRC<sub>p</sub>(c): Other Cache Miss I

- $x_t \notin L_1 \cup L_2$  and  $|L_1| = c$ 
  - if ( $|T_{1,p}| < c$ ): Delete LRU page in  $B_{1,p}$  and Replace( $x_t, p$ )

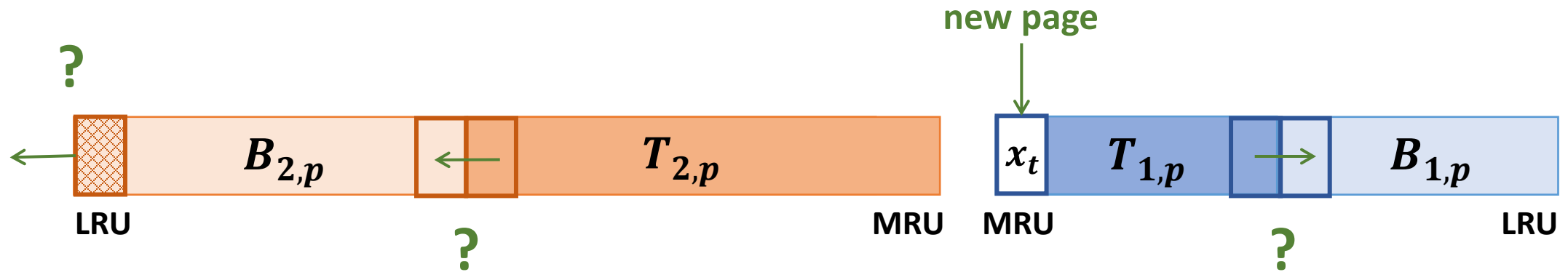


- if ( $|T_{1,p}| = c$ ): Delete LRU page in  $T_{1,p}$

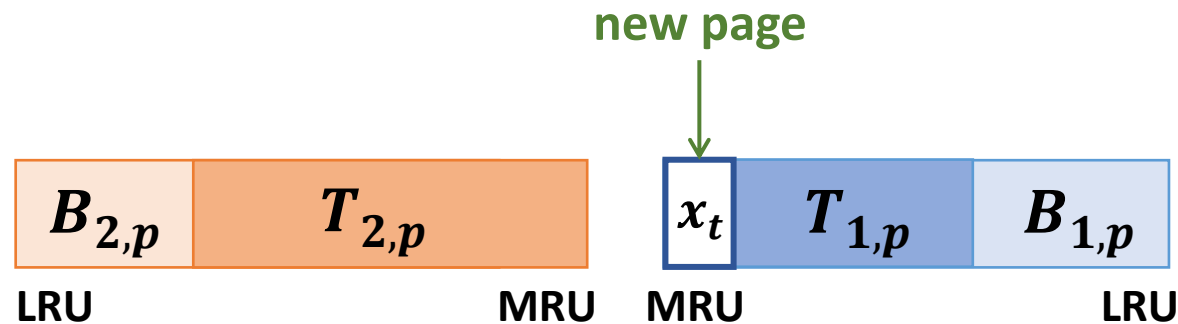


# FRC<sub>p</sub>(c): Other Cache Miss II

- $x_t \notin L_1 \cup L_2$  and  $|L_1| < c$ 
  - if ( $|L_1| + |L_2| \geq c$ ): Delete LRU page in B2 if  $|L_1| + |L_2| = 2c$ , and Replace( $x_t, p$ )



- if ( $|L_1| + |L_2| < c$ ):



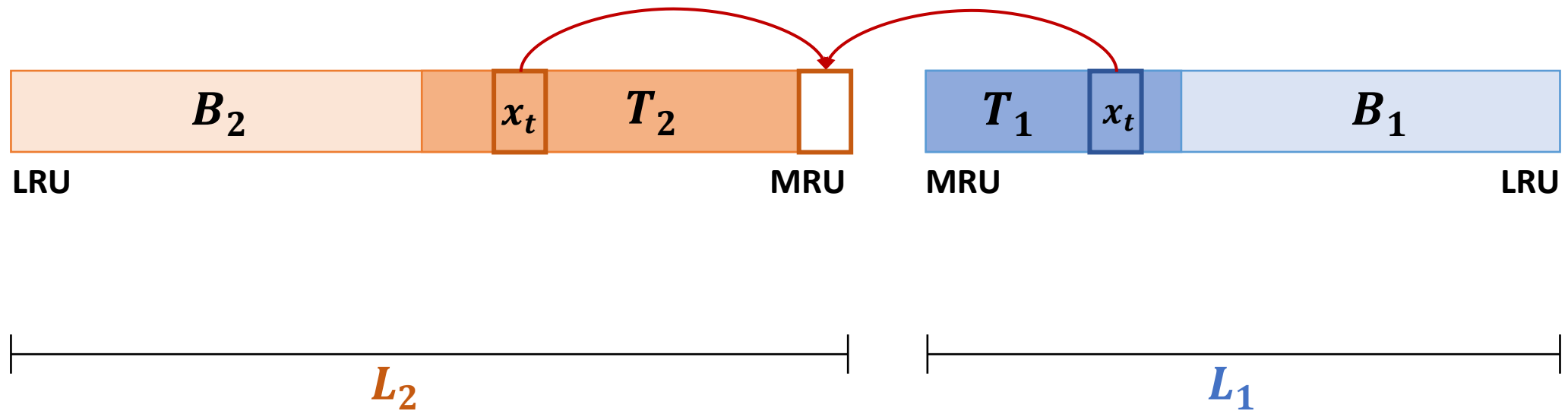


# ARC

- For a given value of  $p$ , ARC behaves exactly like  $FRC_p$
- ARC continuously adapts and tunes  $p$  in response to a workload
- Learning rates
  - "Invest" in the list that is performing the best
  - On a hit in  $B_1$ , we need to increase  $T_1 \rightarrow p$  increased by  $\delta_1 = \begin{cases} 1 & \text{if } |B_1| \geq |B_2| \\ |B_2|/|B_1| & \text{otherwise} \end{cases}$
  - On a hit in  $B_2$ , we need to increase  $T_2 \rightarrow p$  decreased by  $\delta_2 = \begin{cases} 1 & \text{if } |B_2| \geq |B_1| \\ |B_1|/|B_2| & \text{otherwise} \end{cases}$

# ARC(c): Cache Hit

- $x_t \in T_1 \cup T_2$   
→ Move  $x_t$  to MRU position of  $T_2$

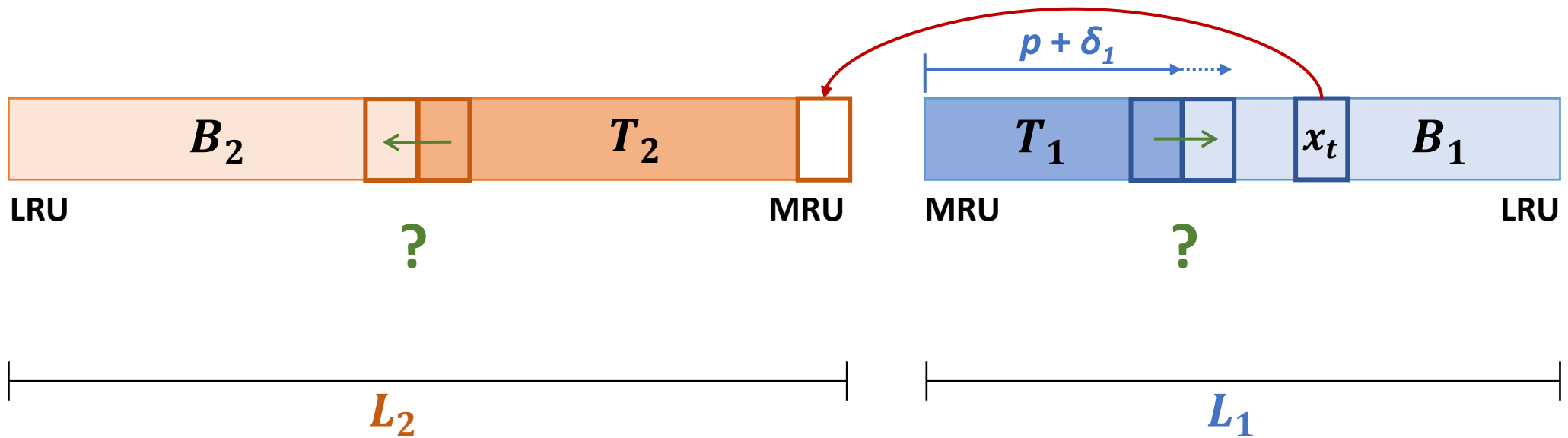


# ARC(c): Cache Miss in $B_1$

- $x_t \in B_1$

→ Update  $p = \min\{p + \delta_1, c\}$  where  $\delta_1 = \begin{cases} 1 & \text{if } |B_1| \geq |B_2| \\ |B_2|/|B_1| & \text{otherwise} \end{cases}$

→  $\text{Replace}(x_t, p)$ , and move  $x_t$  from  $B_1$  to MRU position of  $T_2$

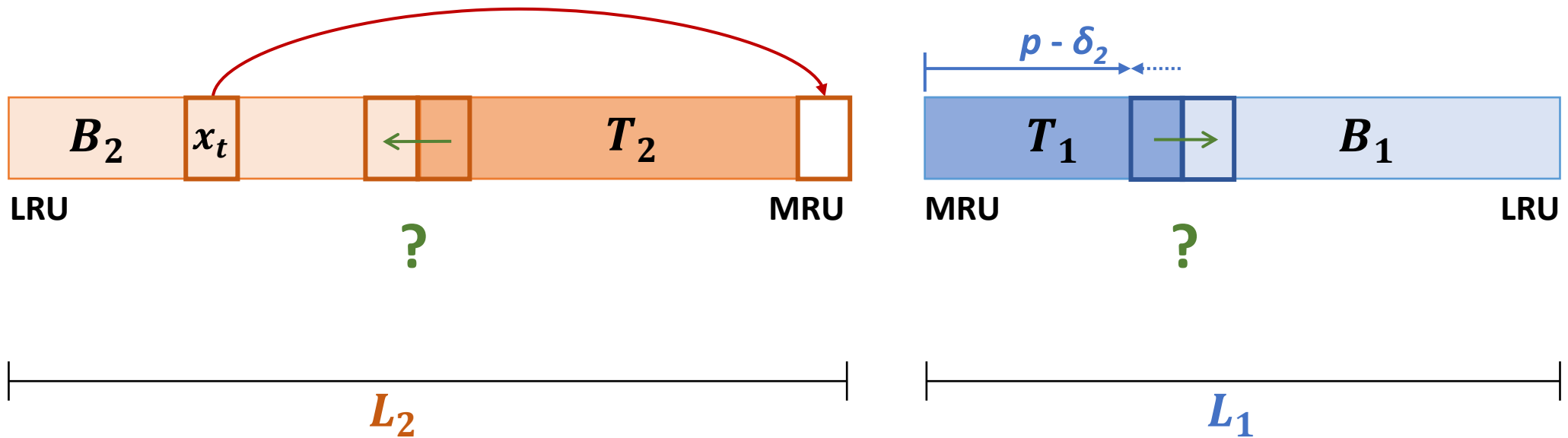


# ARC(c): Cache Miss in $B_2$

- $x_t \in B_2$

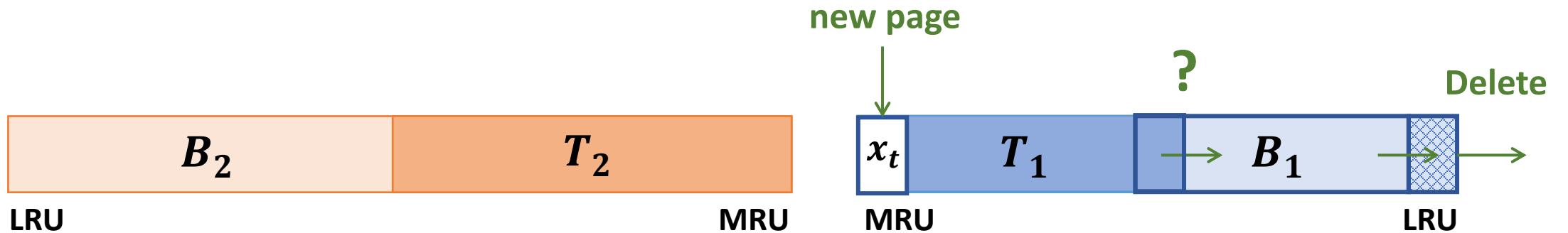
→ Update  $p = \max\{p - \delta_2, 0\}$  where  $\delta_2 = \begin{cases} 1 & \text{if } |B_2| \geq |B_1| \\ |B_1|/|B_2| & \text{otherwise} \end{cases}$

→ Replace( $x_t, p$ ), and move  $x_t$  from  $B_2$  to MRU position of  $T_2$

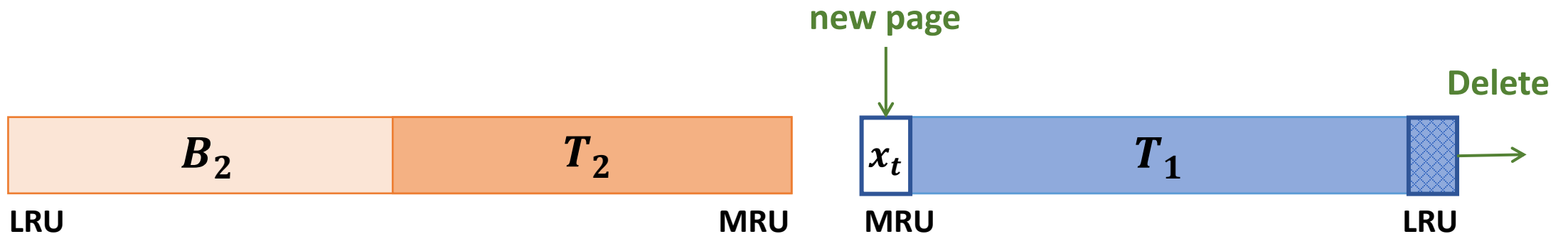


# ARC(c): Other Cache Miss I

- $x_t \notin L_1 \cup L_2$  and  $|L_1| = c$ 
  - if ( $|T_1| < c$ ): Delete LRU page in  $B_1$  and  $\text{Replace}(x_t, p)$

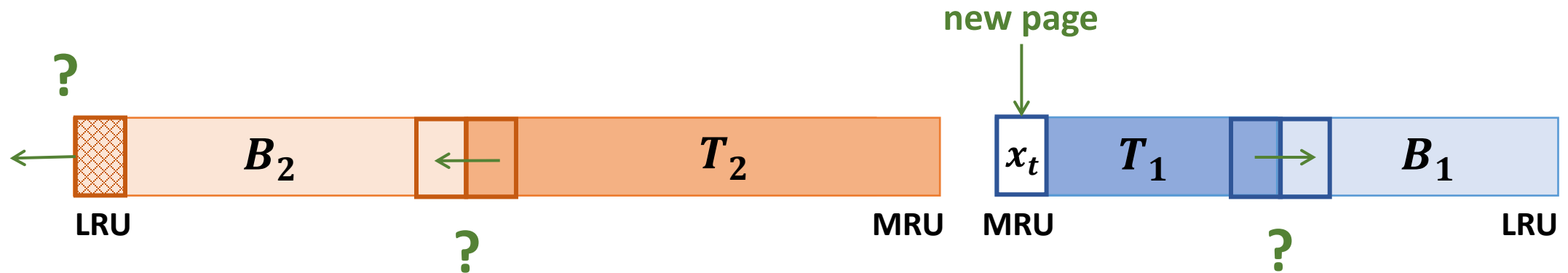


- if ( $|T_1| = c$ ): Delete LRU page in  $T_1$

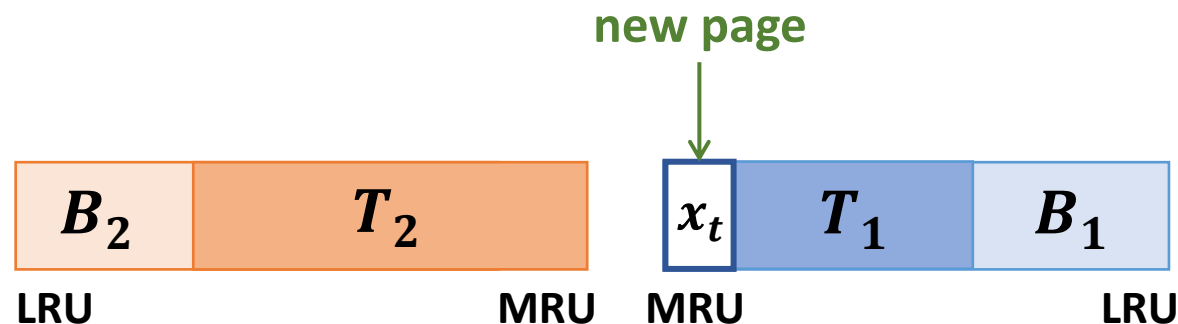


# ARC(c): Other Cache Miss II

- $x_t \notin L_1 \cup L_2$  and  $|L_1| < c$ 
  - if  $(|L_1| + |L_2| \geq c)$ : Delete LRU page in B2 if  $|L_1| + |L_2| = 2c$ , and  $\text{Replace}(x_t, p)$



- if  $(|L_1| + |L_2| < c)$ :



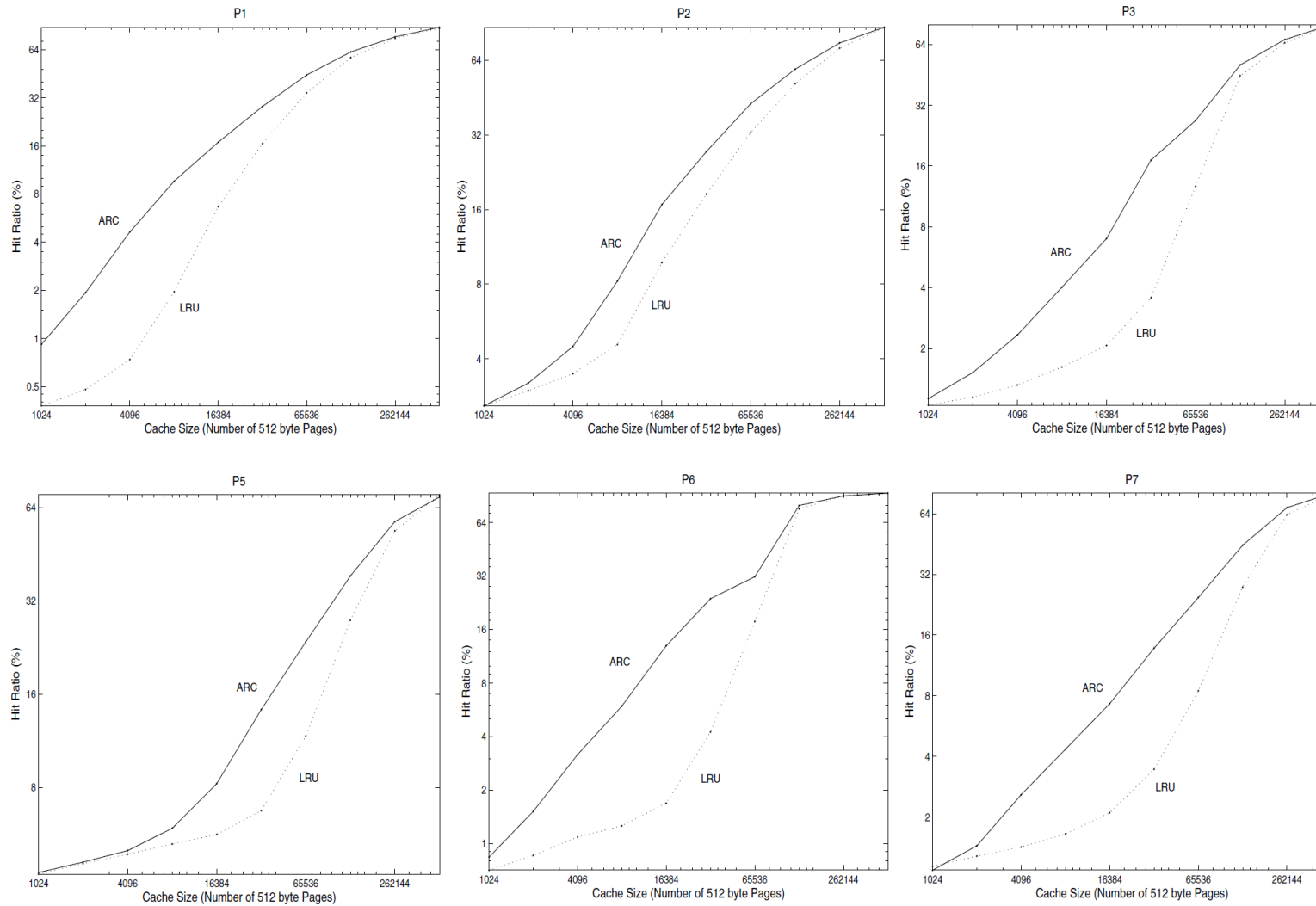
# Hit Ratios

- ARC outperforms online algorithms (LRU, FBR, LFU, LIRS, MQ)
- ARC performs as well as LRU-2, 2Q, LRFU which use the best offline parameters

OLTP

c	ONLINE						OFFLINE			MIN
	LRU	ARC	FBR	LFU	LIRS	MQ	LRU-2	2Q	LRFU	
1000	32.83	38.93	36.96	27.98	34.80	37.86	39.30	40.48	40.52	53.61
2000	42.47	46.08	43.98	35.21	42.51	44.10	45.82	46.53	46.11	60.40
5000	53.65	55.25	53.53	44.76	47.14	54.39	54.78	55.70	56.73	68.27
10000	60.70	61.87	62.32	52.15	60.35	61.08	62.42	62.58	63.54	73.02
15000	64.63	65.40	65.66	56.22	63.99	64.81	65.22	65.82	67.06	75.13

# ARC vs. LRU



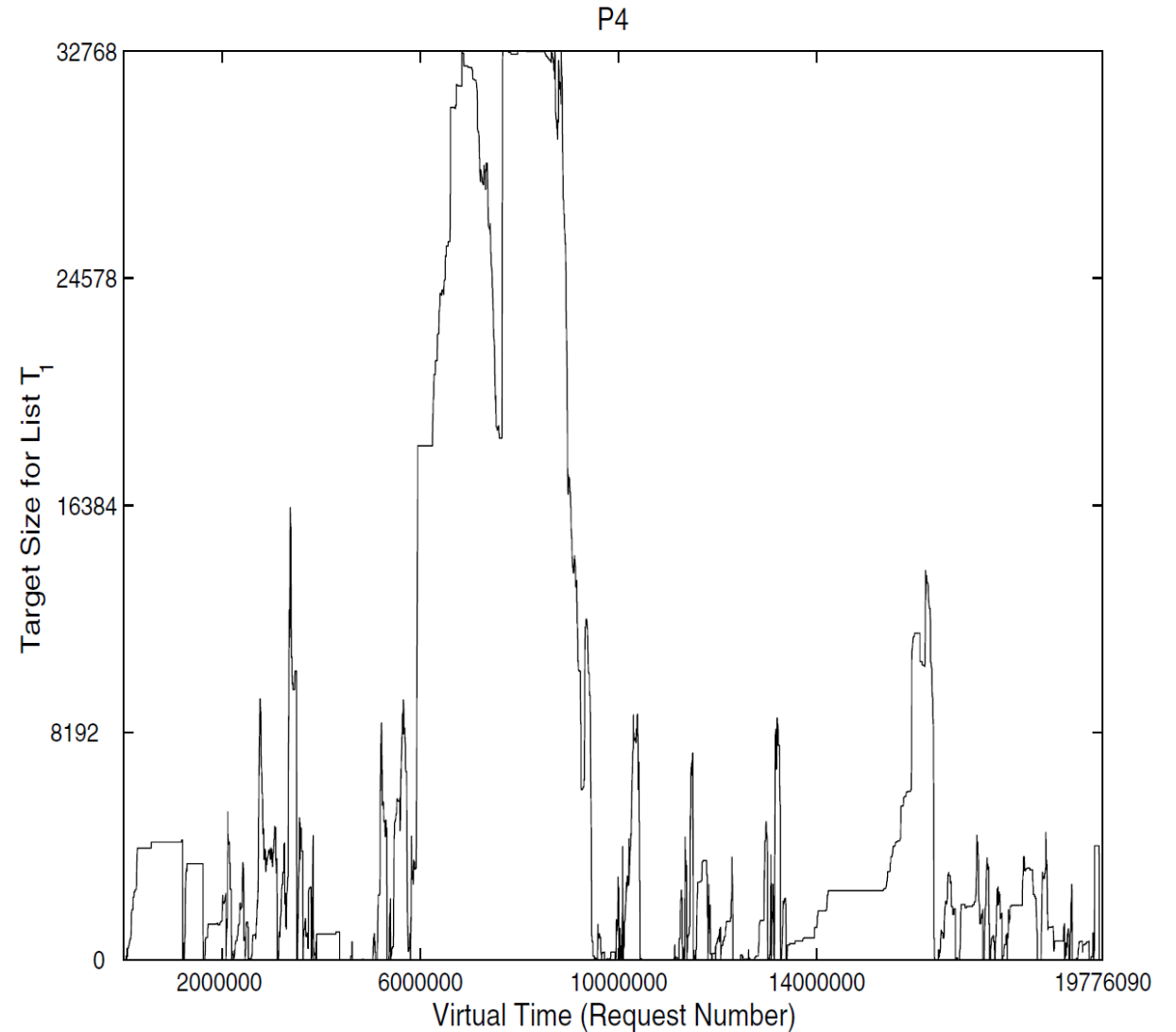


# ARC vs. FRC

Workload	c	space MB	LRU	ARC	FRC OFFLINE
P1	32768	16	16.55	28.26	29.39
P2	32768	16	18.47	27.38	27.61
P3	32768	16	3.57	17.12	17.60
P4	32768	16	5.24	11.24	9.11
P5	32768	16	6.73	14.27	14.29
P6	32768	16	4.24	23.84	22.62
P7	32768	16	3.45	13.77	14.01
P8	32768	16	17.18	27.51	28.92
P9	32768	16	8.28	19.73	20.28
P10	32768	16	2.48	9.46	9.63
P11	32768	16	20.92	26.48	26.57
P12	32768	16	8.93	15.94	15.97
P13	32768	16	7.83	16.60	16.81
P14	32768	16	15.73	20.52	20.55
ConCat	32768	16	14.38	21.67	21.63
Merge(P)	262144	128	38.05	39.91	39.40
DS1	2097152	1024	11.65	22.52	18.72
SPC1	1048576	4096	9.19	20.00	20.11
S1	524288	2048	23.71	33.43	34.00
S2	524288	2048	25.91	40.68	40.57
S3	524288	2048	25.26	40.44	40.29
Merge(S)	1048576	4096	27.62	40.44	40.18

# Adaptation

- Parameter  $p$  keeps fluctuating between  $0$  to  $c$  (32768 pages)
- Such fluctuations occur as many times as dictated by the nature of the workload without any *a priori* knowledge or offline tuning
- ARC continually adapts and reconfigures itself



# To ARC or Not to ARC...

← → ↻ 🔒 patents.google.com/patent/US6996676B2/en ☆ S ⚙

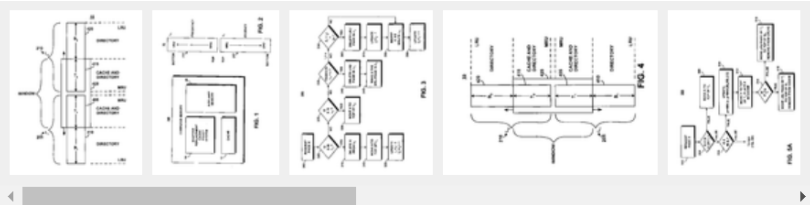
Google Patents 🔍

## System and method for implementing an adaptive replacement cache policy

### Abstract

An adaptive replacement cache policy dynamically maintains two lists of pages, a recency list and a frequency list, in addition to a cache directory. The policy keeps these two lists to roughly the same size, the cache size  $c$ . Together, the two lists remember twice the number of pages that would fit in the cache. At any time, the policy selects a variable number of the most recent pages to exclude from the two lists. The policy adaptively decides in response to an evolving workload how many top pages from each list to maintain in the cache at any given time. It achieves such online, on-the-fly adaptation by using a learning rule that allows the policy to track a workload quickly and effectively. This allows the policy to balance between recency and frequency in an online and self-tuning fashion, in response to evolving and possibly changing access patterns. The policy is also scan-resistant. It allows one-time-only sequential read requests to pass through the cache without flushing pages that have temporal locality. The policy is extremely simple to implement and requires only constant-time overhead per request. The policy has negligible space overhead.

### Images (12)



### Classifications

- **G06F12/123** Replacement control using replacement algorithms with age lists, e.g. queue, most recently used [MRU] list or least recently used [LRU] list

### US6996676B2

United States

Download PDF Find Prior Art Similar

**Inventor:** Nimrod Megiddo, Dharmendra Shantilal Modha  
**Current Assignee:** Intel Corp

#### Worldwide applications

2002 · [US](#) 2005 · [US](#)

#### Application US10/295,507 events

- 2002-11-14 • Application filed by International Business Machines Corp
- 2002-11-14 • Priority to US10/295,507
- 2002-11-14 • Assigned to INTERNATIONAL BUSINESS MACHINES CORPORATION
- 2004-05-20 • Publication of US20040098541A1
- 2006-02-07 • Application granted
- 2006-02-07 • Publication of US6996676B2
- 2013-04-16 • Assigned to INTEL CORPORATION

**Status** • Active

# Summary

	LRU	LFU	FBR	LRU-2	2Q	LRFU	LIRS	ARC
Recency	O	X	O	O	O	O	O	O
Frequency	X	O	O	O	O	O	O	O
Complexity	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)?$	$O(1)$
Scan resistance	X	O	O	O	O	O	O	O
Ghost cache	X	X	X	O	O	X	O	O
Self-tunable	—	—	X	X	X	X	X	O