Advanced Buffer Management

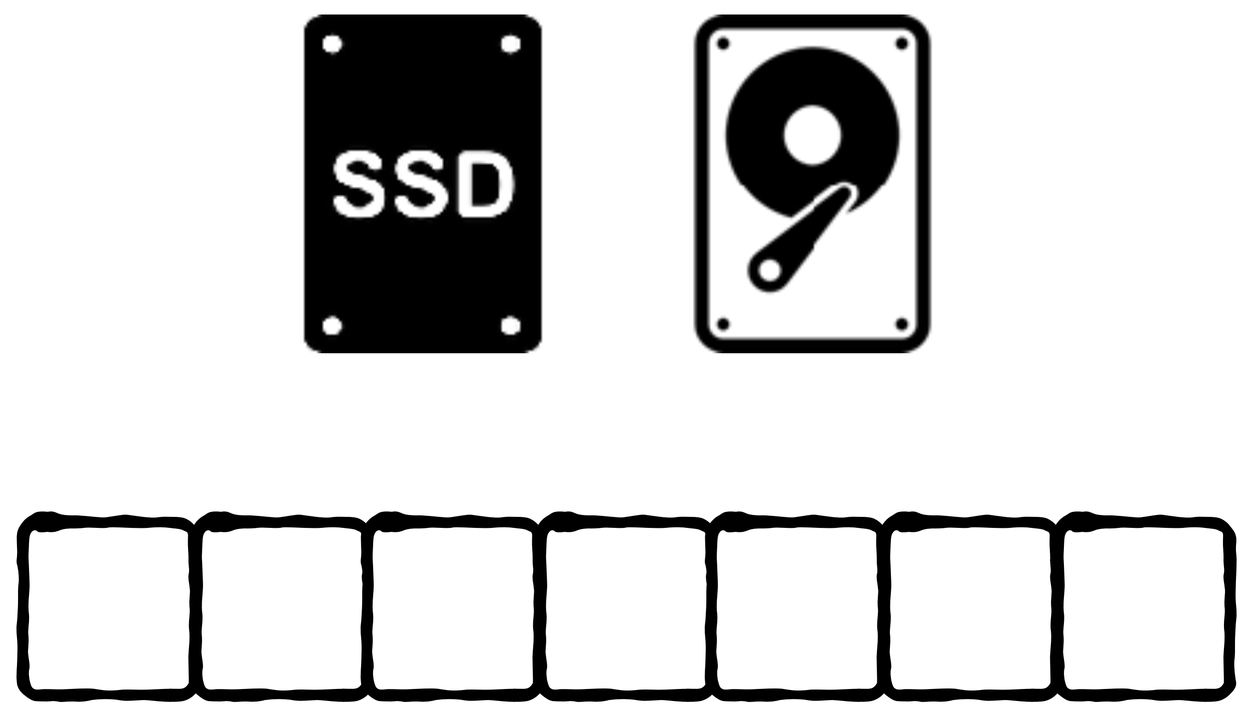
Niv Dayan - CSC2525: Research Topics in Database Management

FIFO/LRU
Review

LRU/K

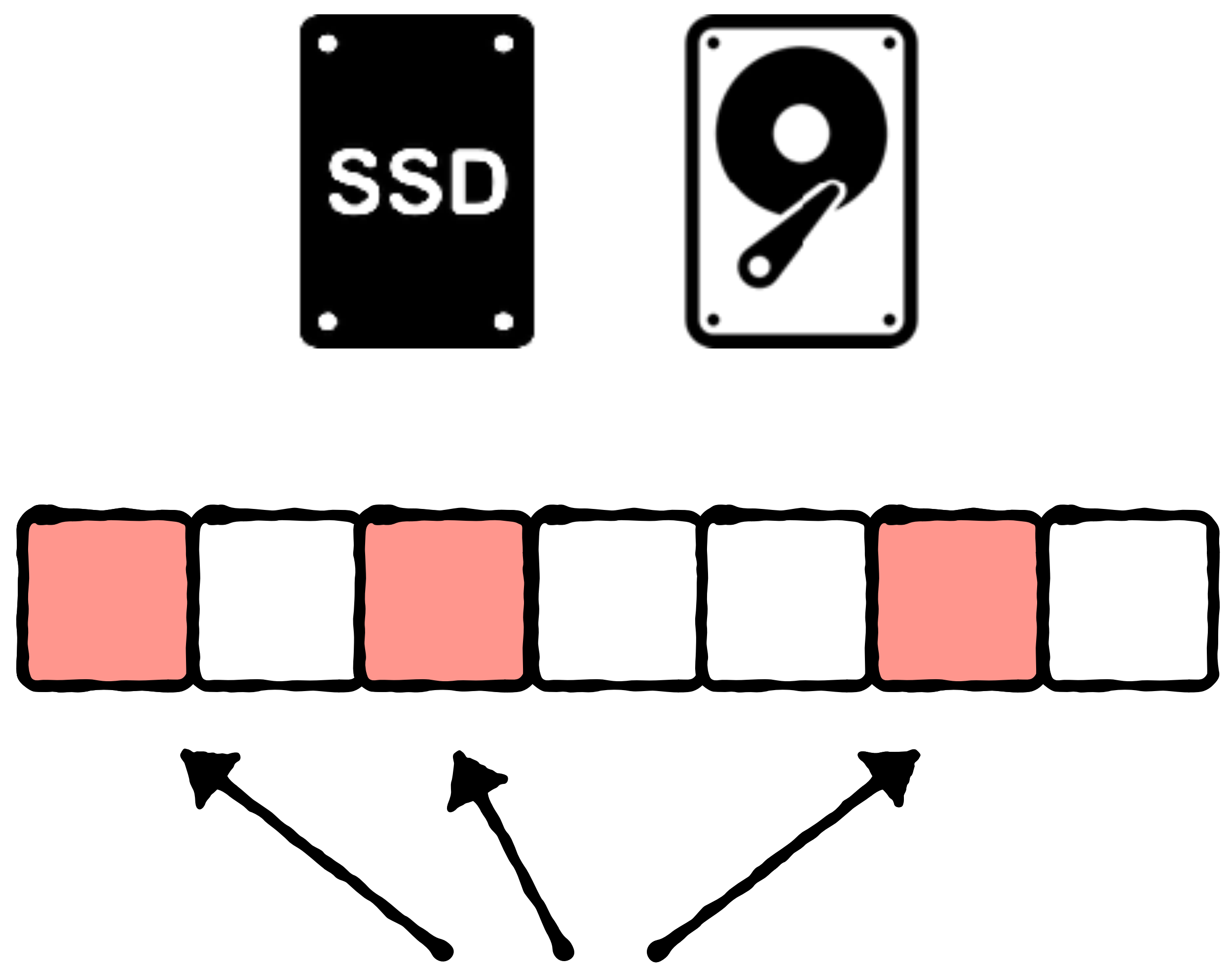
2Q

Review



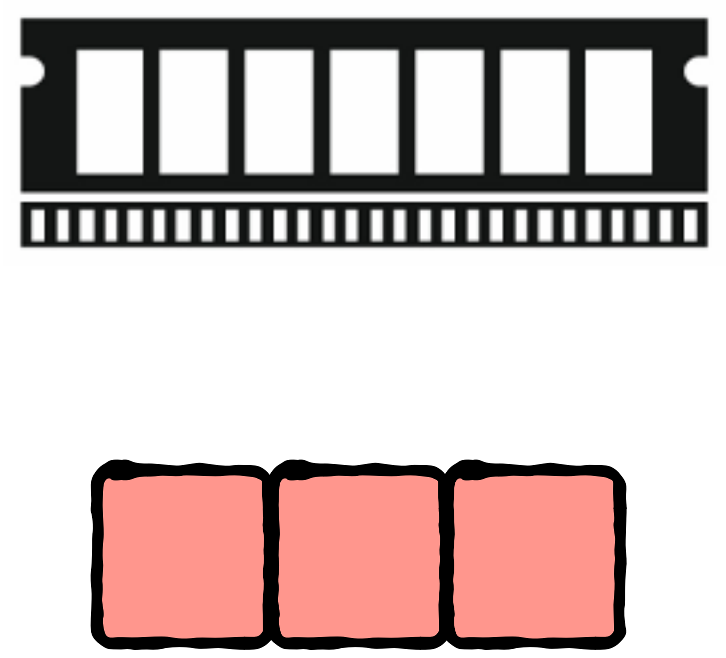
4KB pages

Review

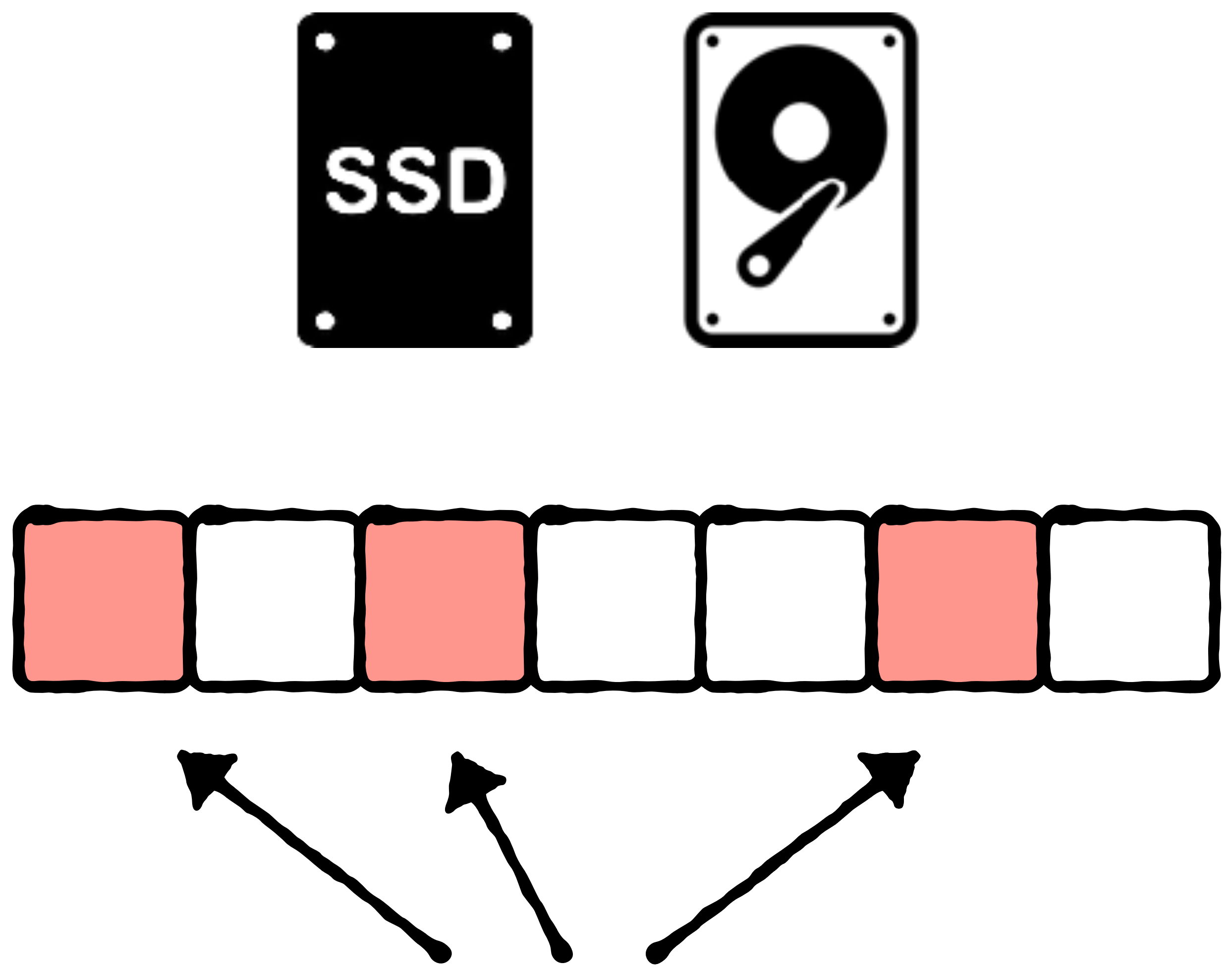


Frequently accessed (hot)

Review

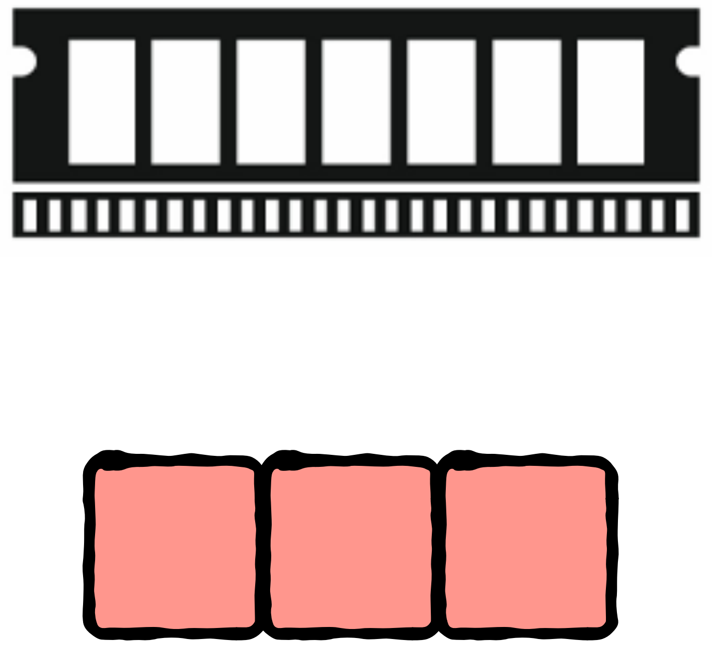


Buffer pool / block cache

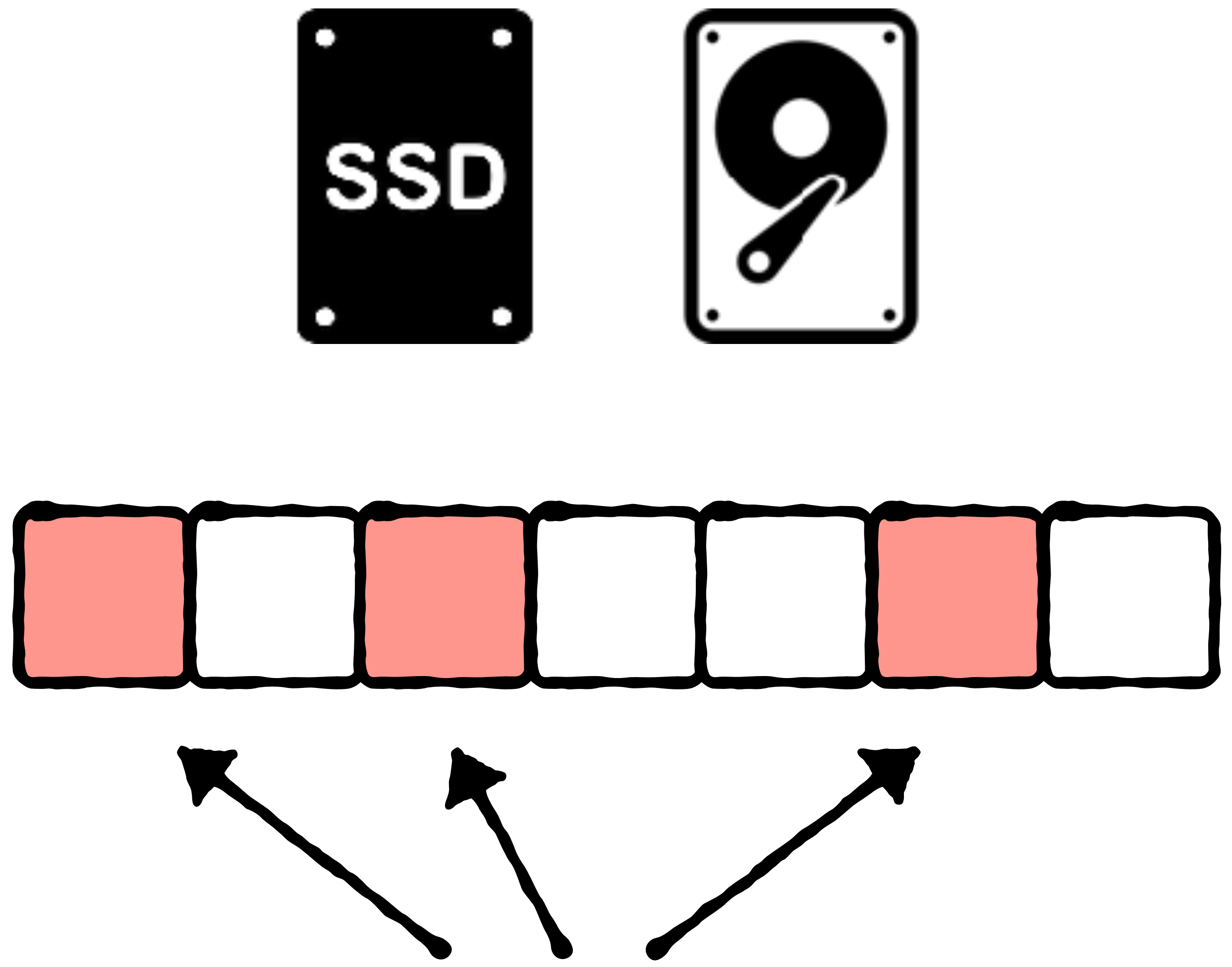


Frequently accessed (hot)

Review



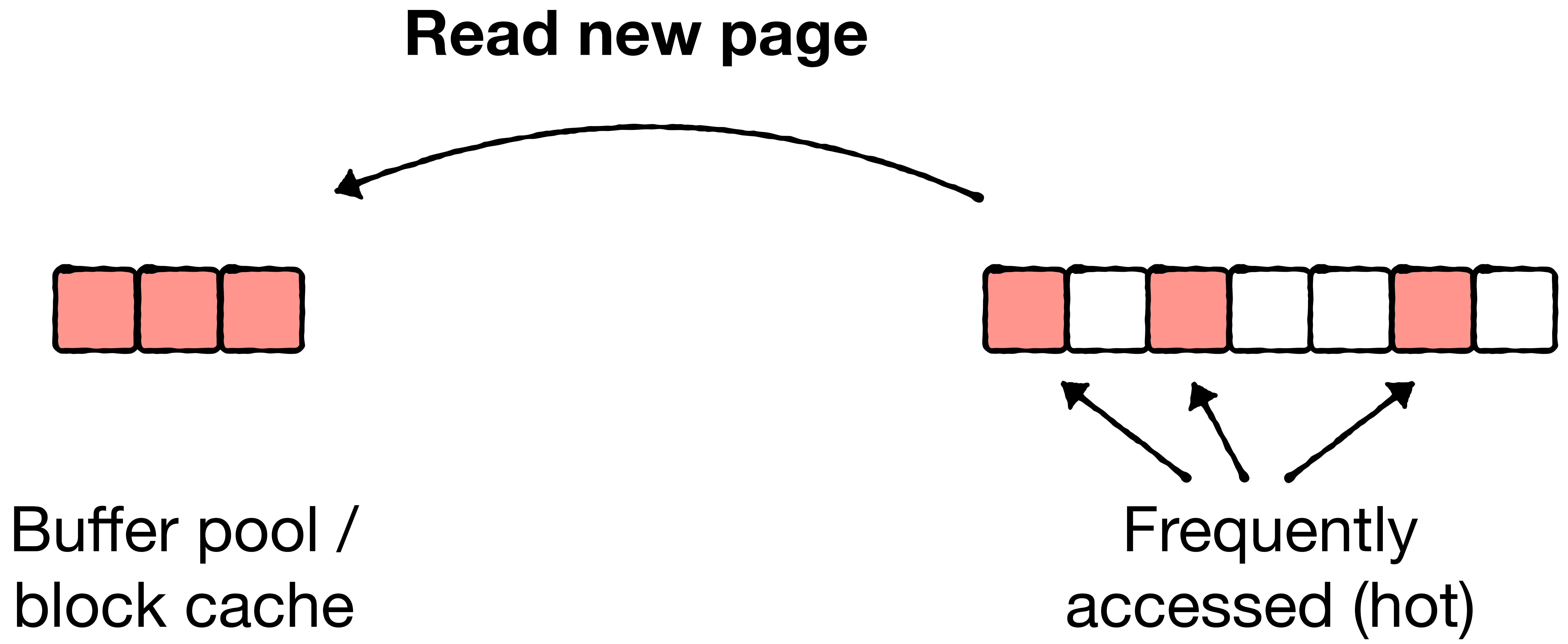
Buffer pool /
block cache



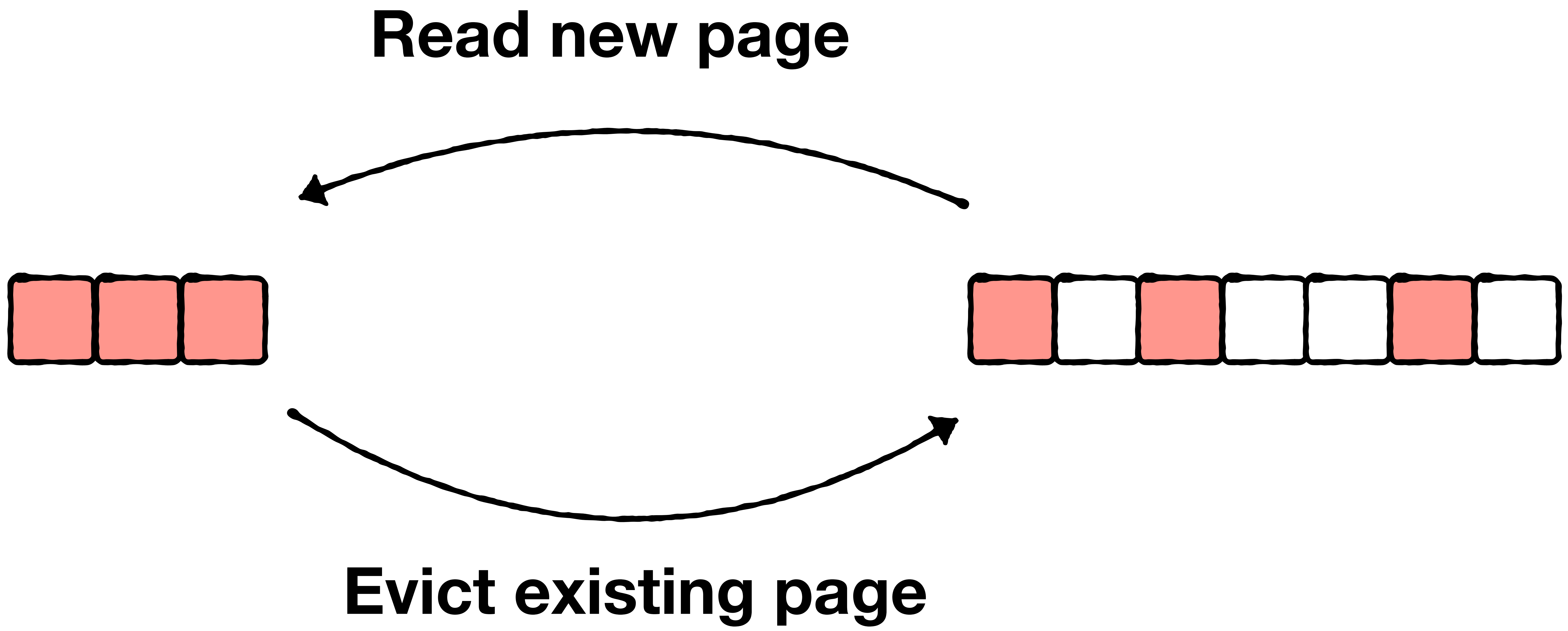
Frequently
accessed (hot)

Implemented as a hash table

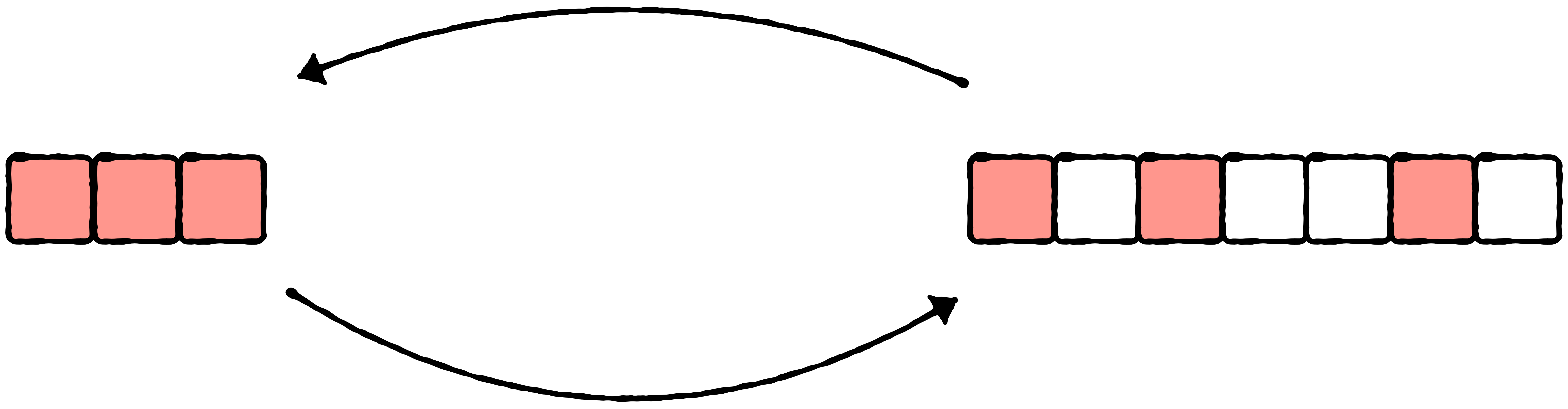
Review


Read new page
Buffer pool /
block cache
Frequently
accessed (hot)

Review



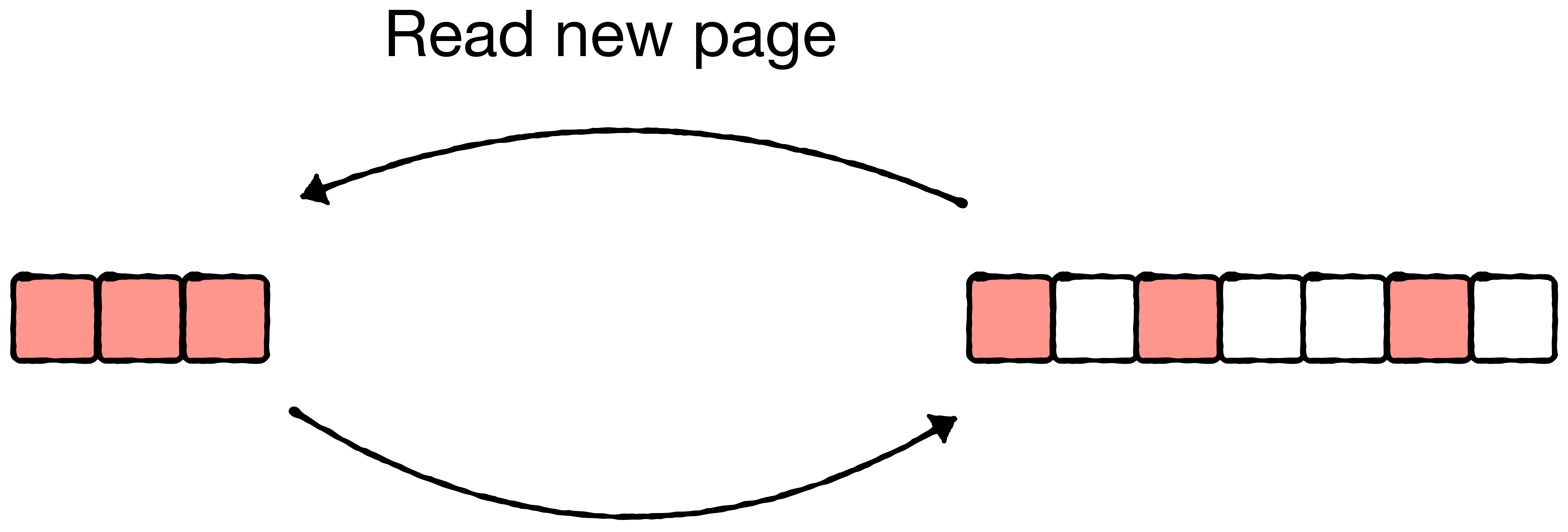
Review



Evict existing page

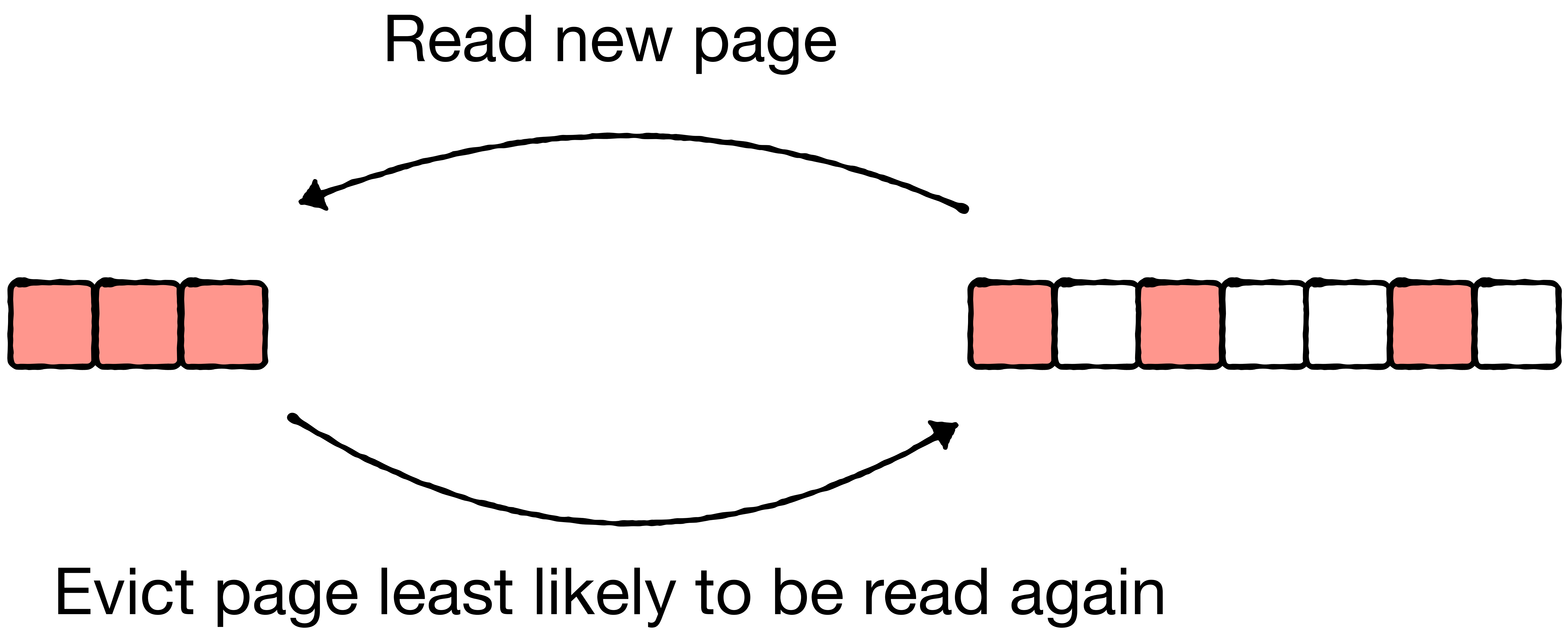
But which?

Review



Evict page least likely to be read again

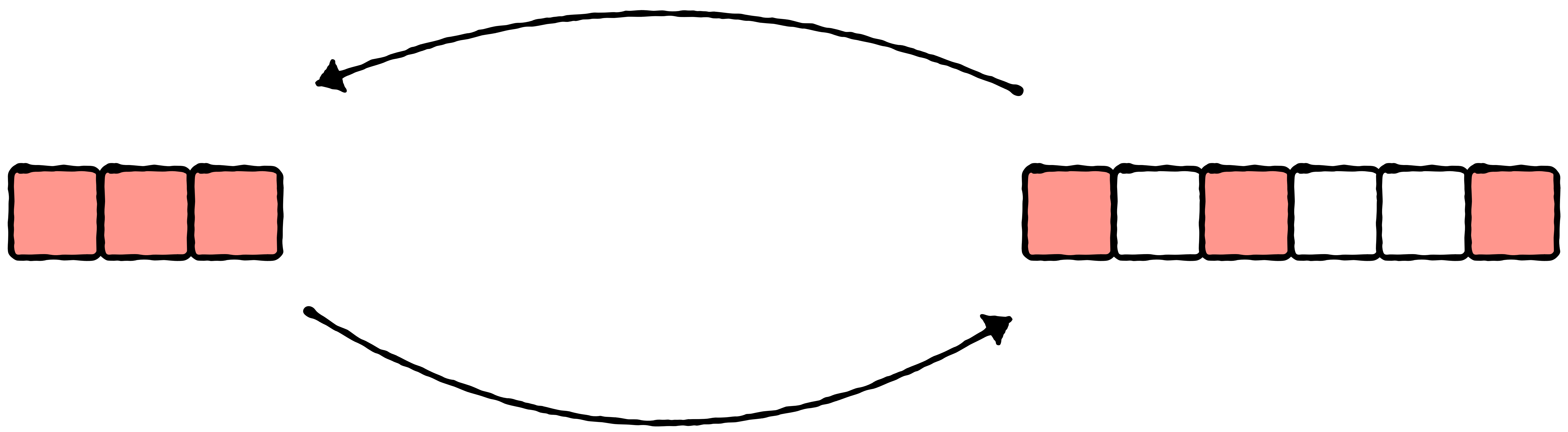
Review



Must approximate

FIFO - First In First Out

Evict page read longest ago

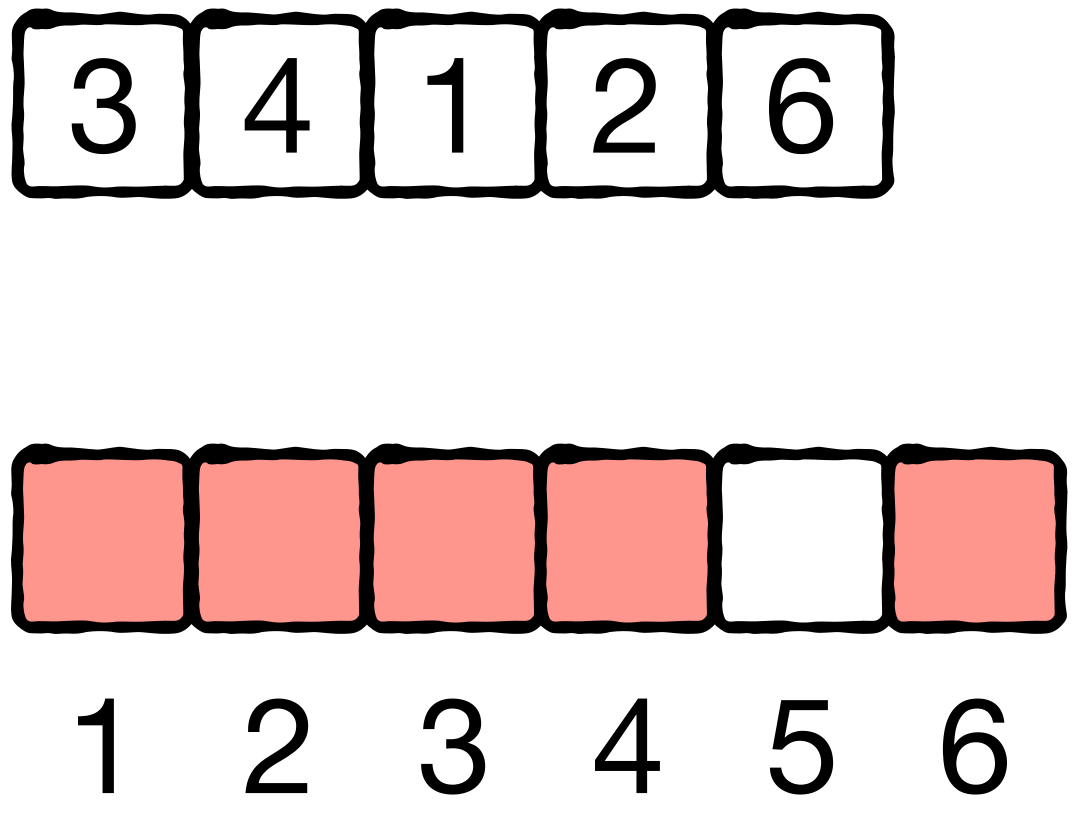


FIFO - First In First Out

Evict page read longest ago

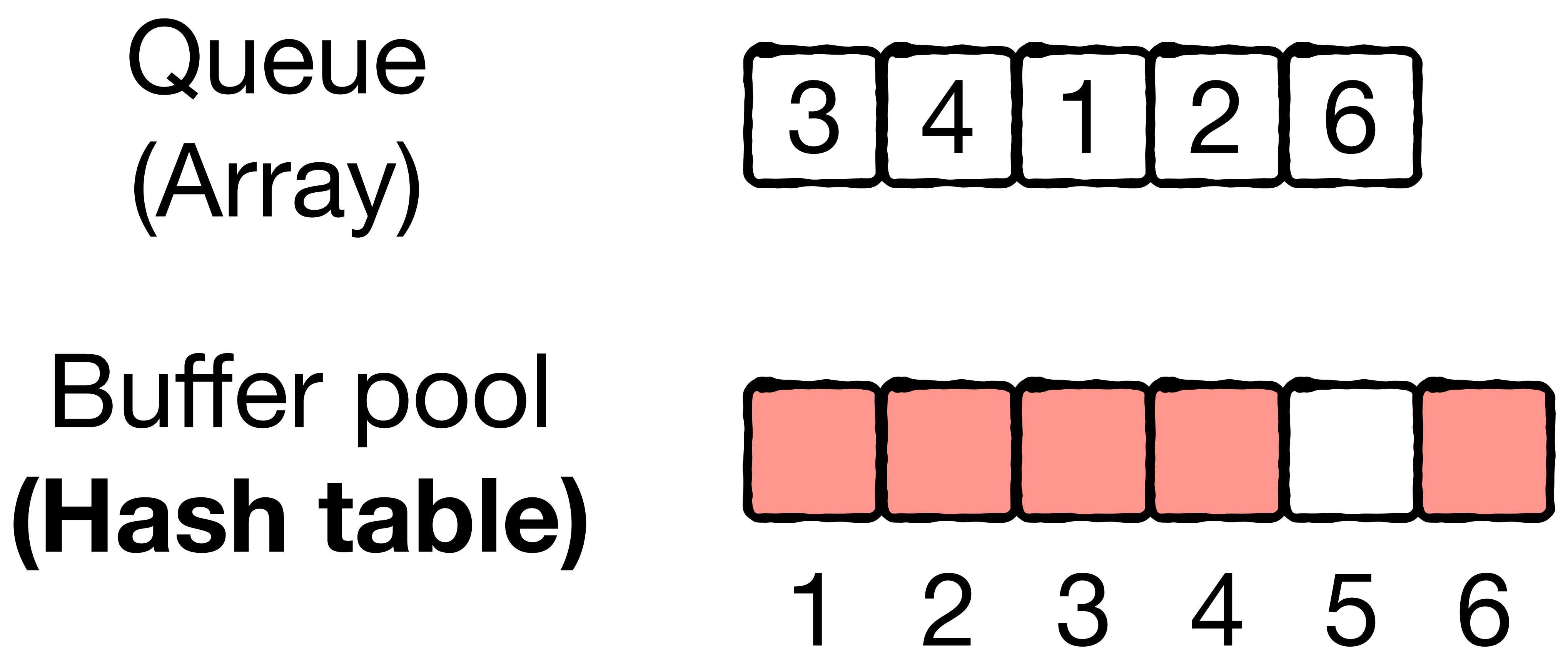
Queue
(Array)

Buffer pool
(Hash table)



FIFO - First In First Out

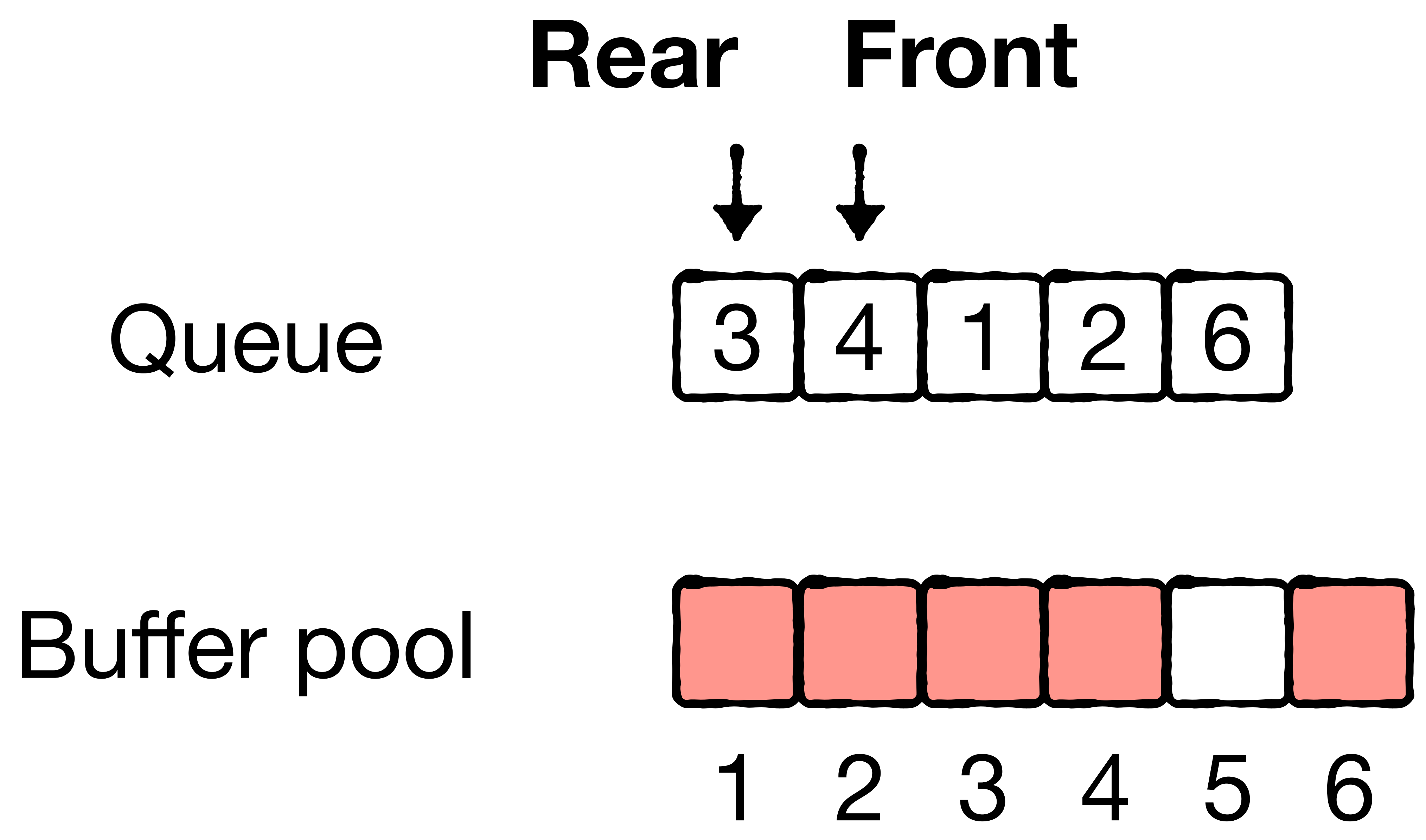
Evict page read longest ago



Hash collisions (not shown in slides) are addressed through standard techniques (e.g., chaining, linear probing etc)

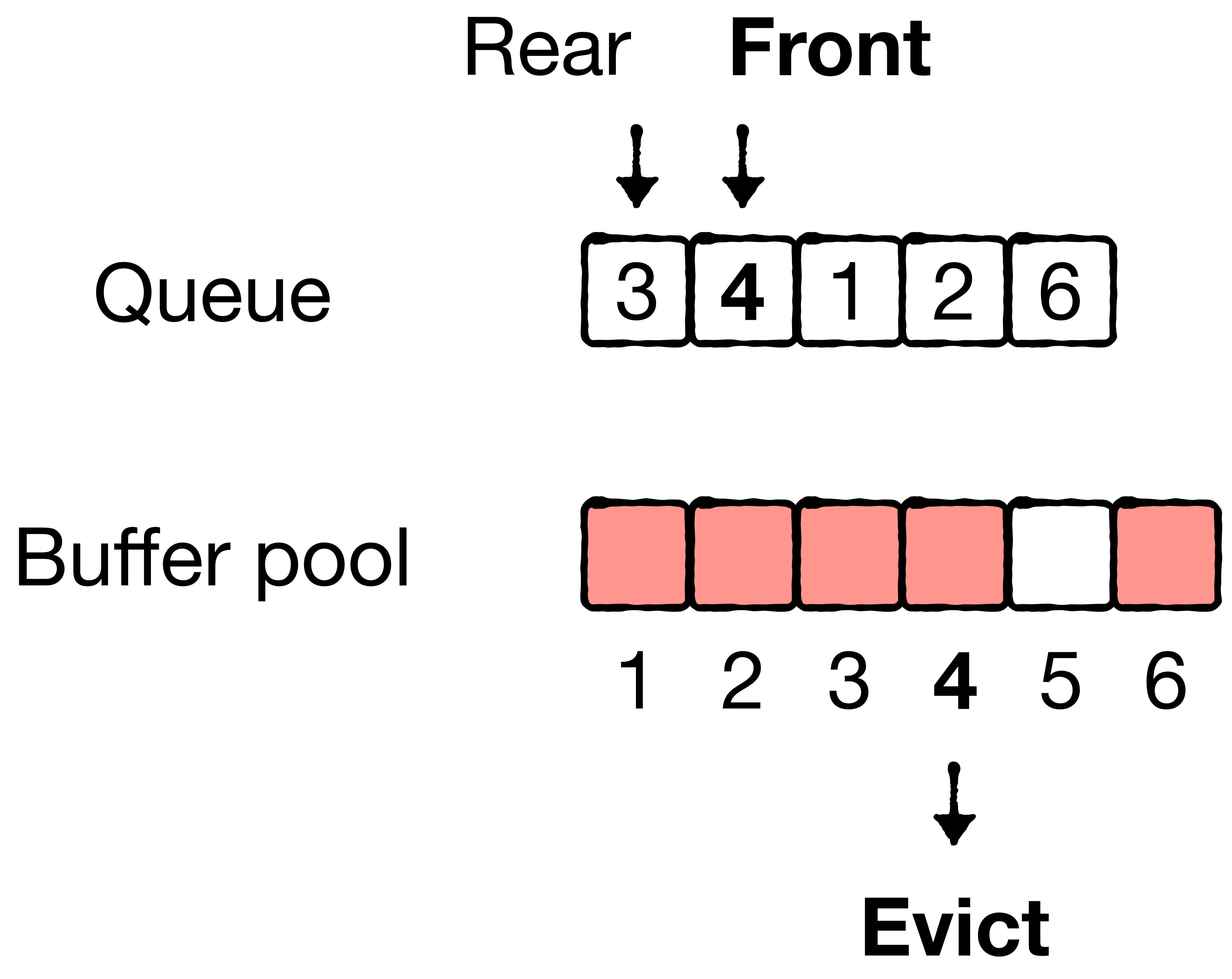
FIFO - First In First Out

Evict page read longest ago



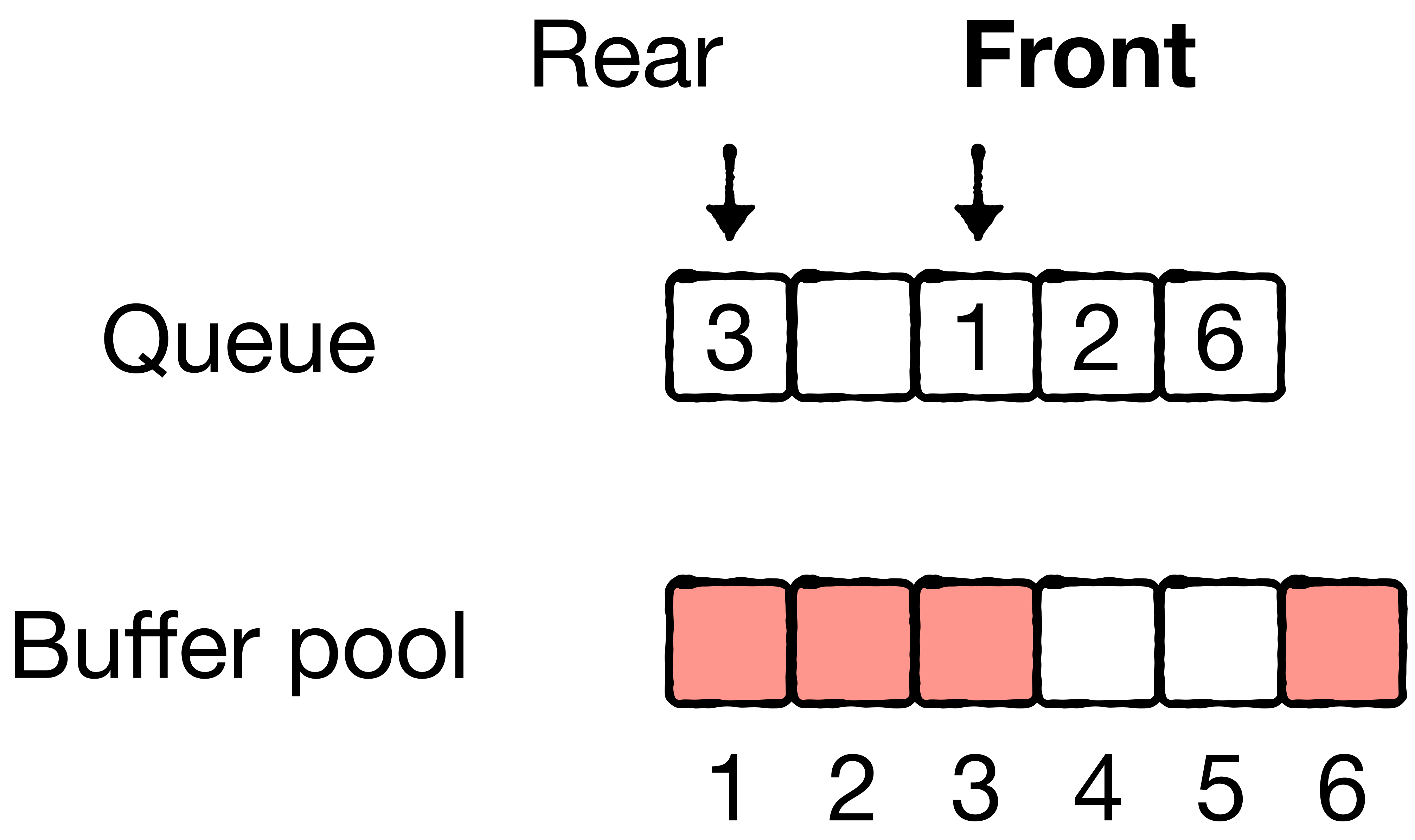
FIFO - First In First Out

Evict page read longest ago



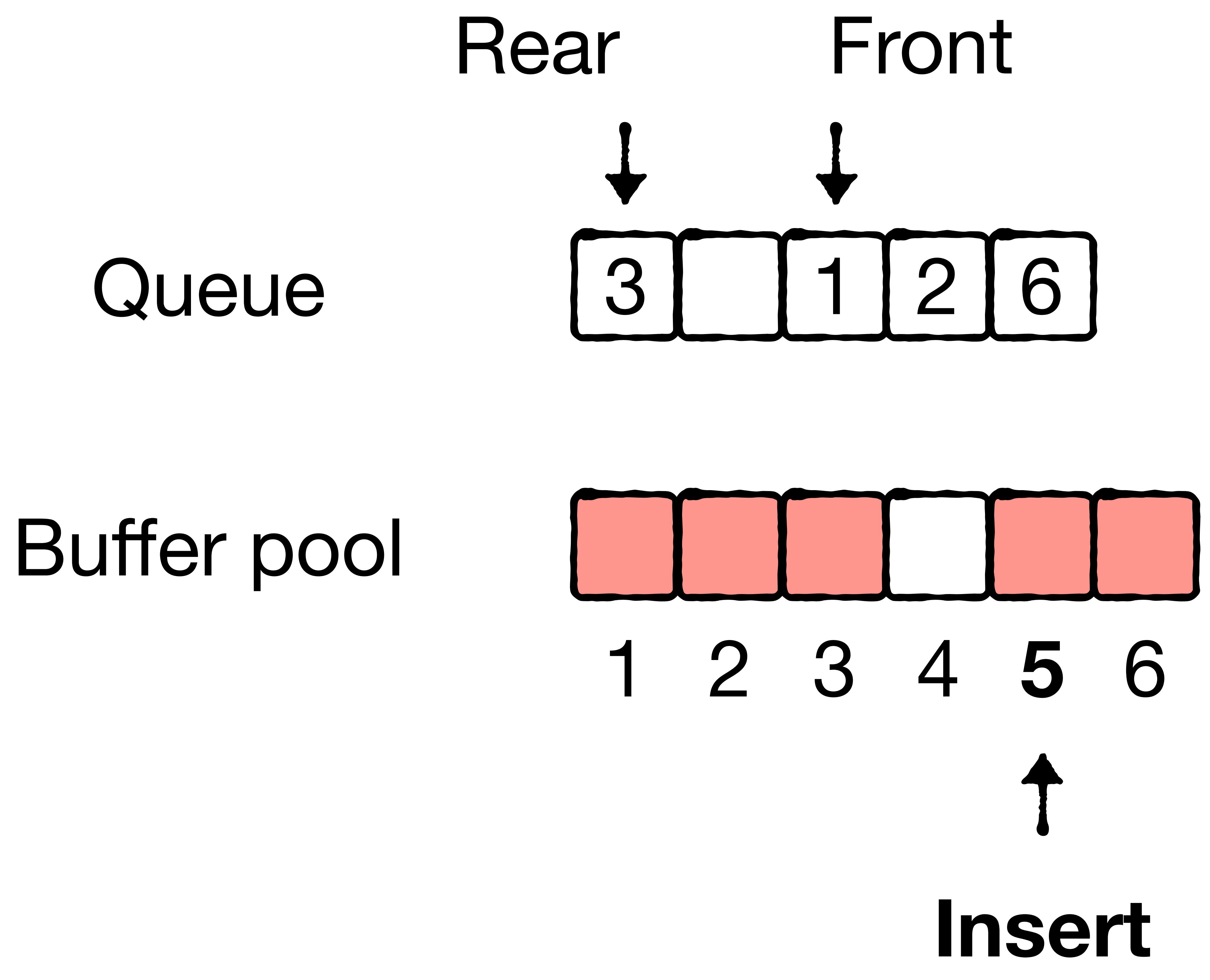
FIFO - First In First Out

Evict page read longest ago



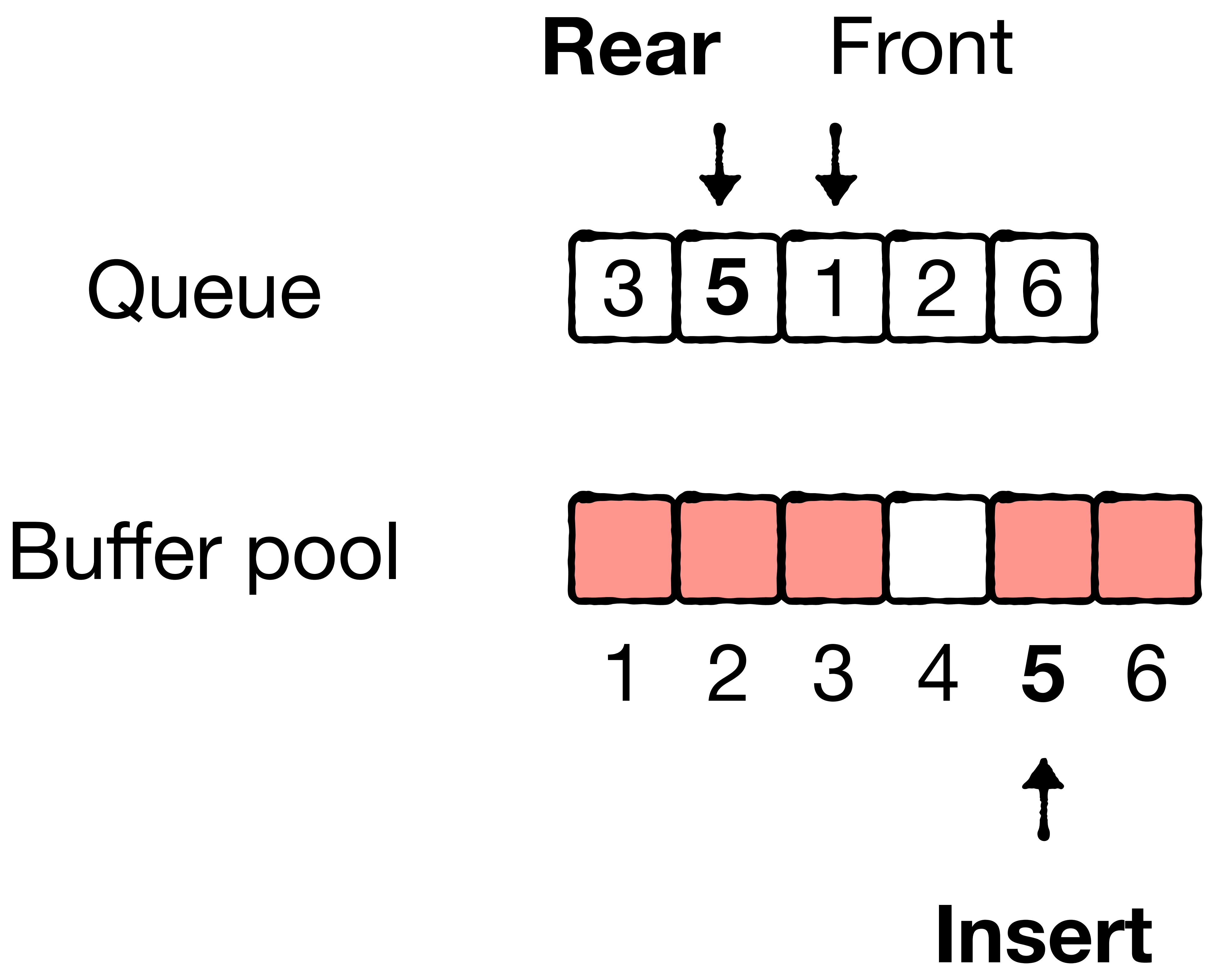
FIFO - First In First Out

Evict page read longest ago



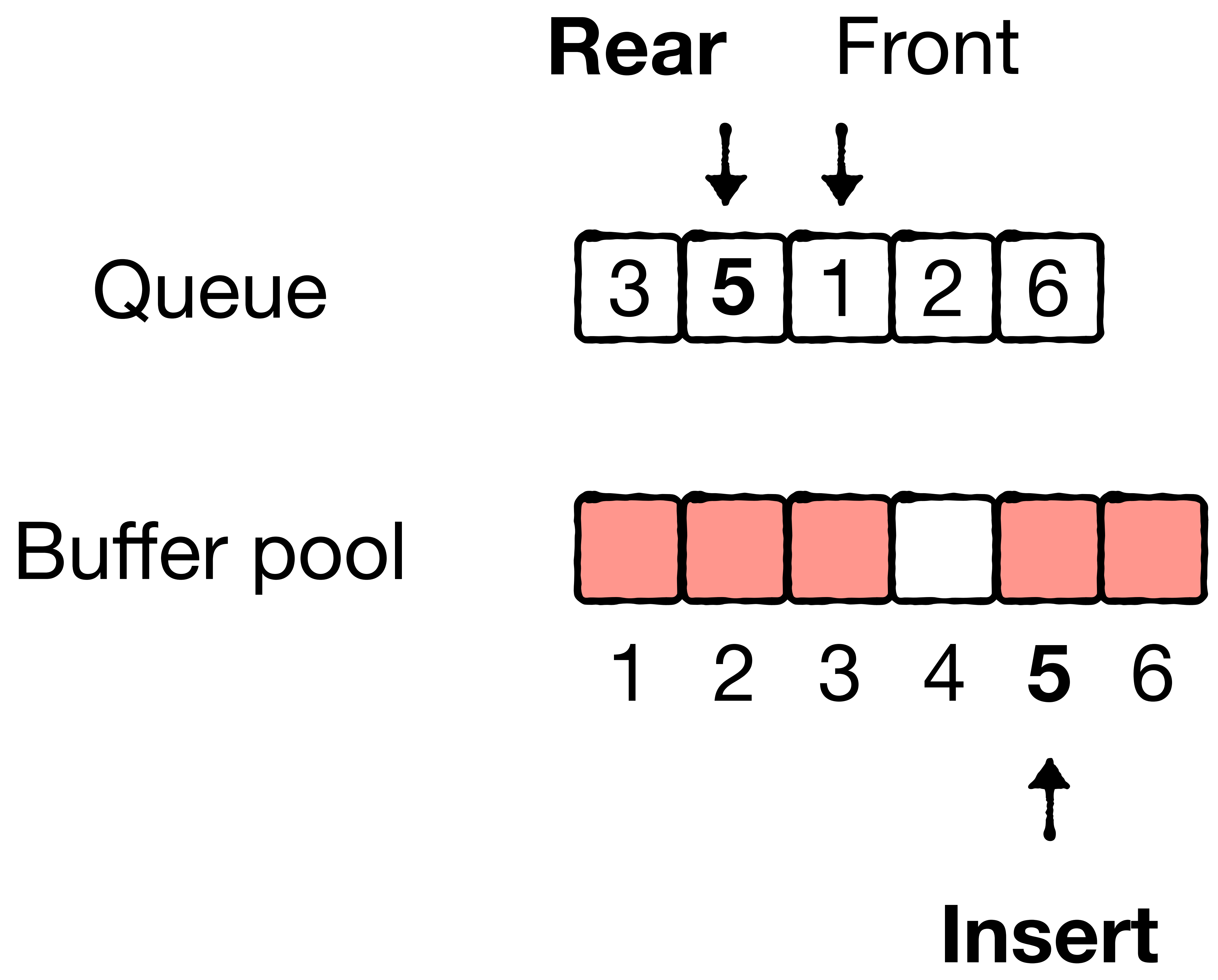
FIFO - First In First Out

Evict page read longest ago



FIFO - First In First Out

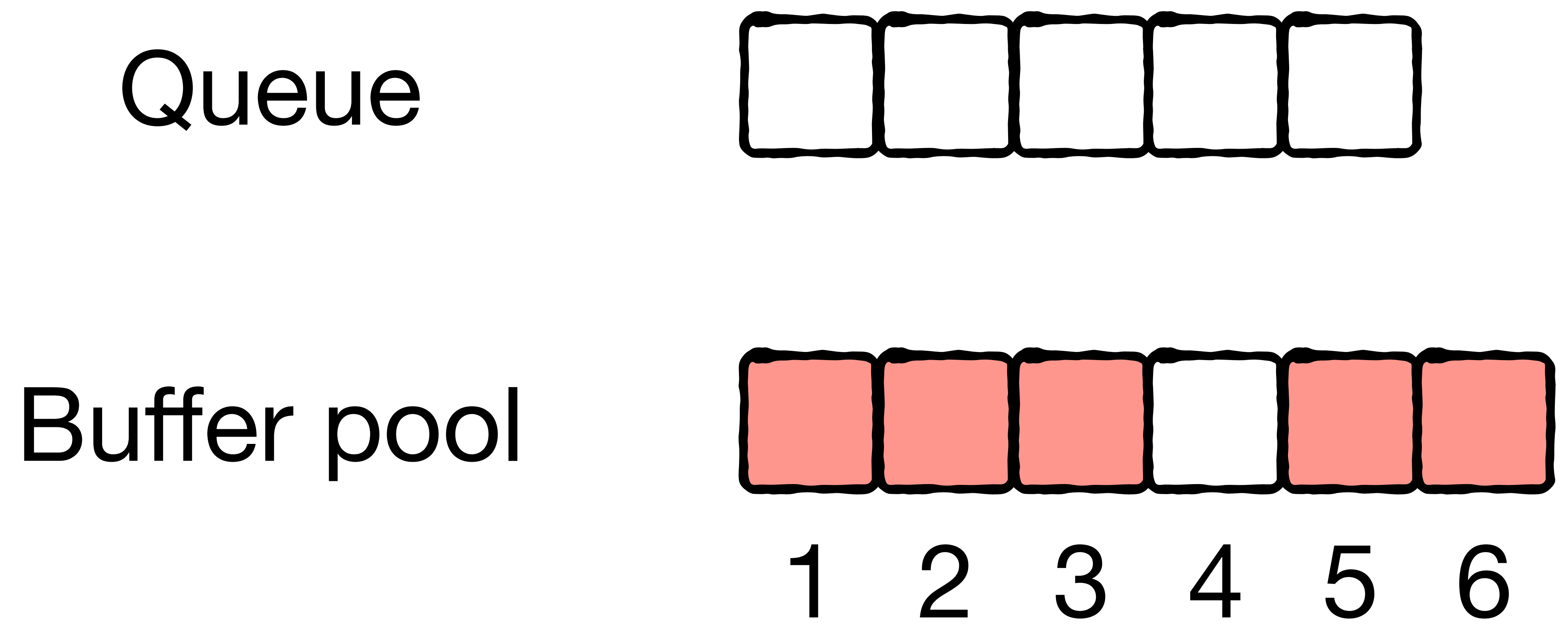
Evict page read longest ago



Can evict frequently accessed page!

LRU - Least Recently Used

Evict page accessed longest ago



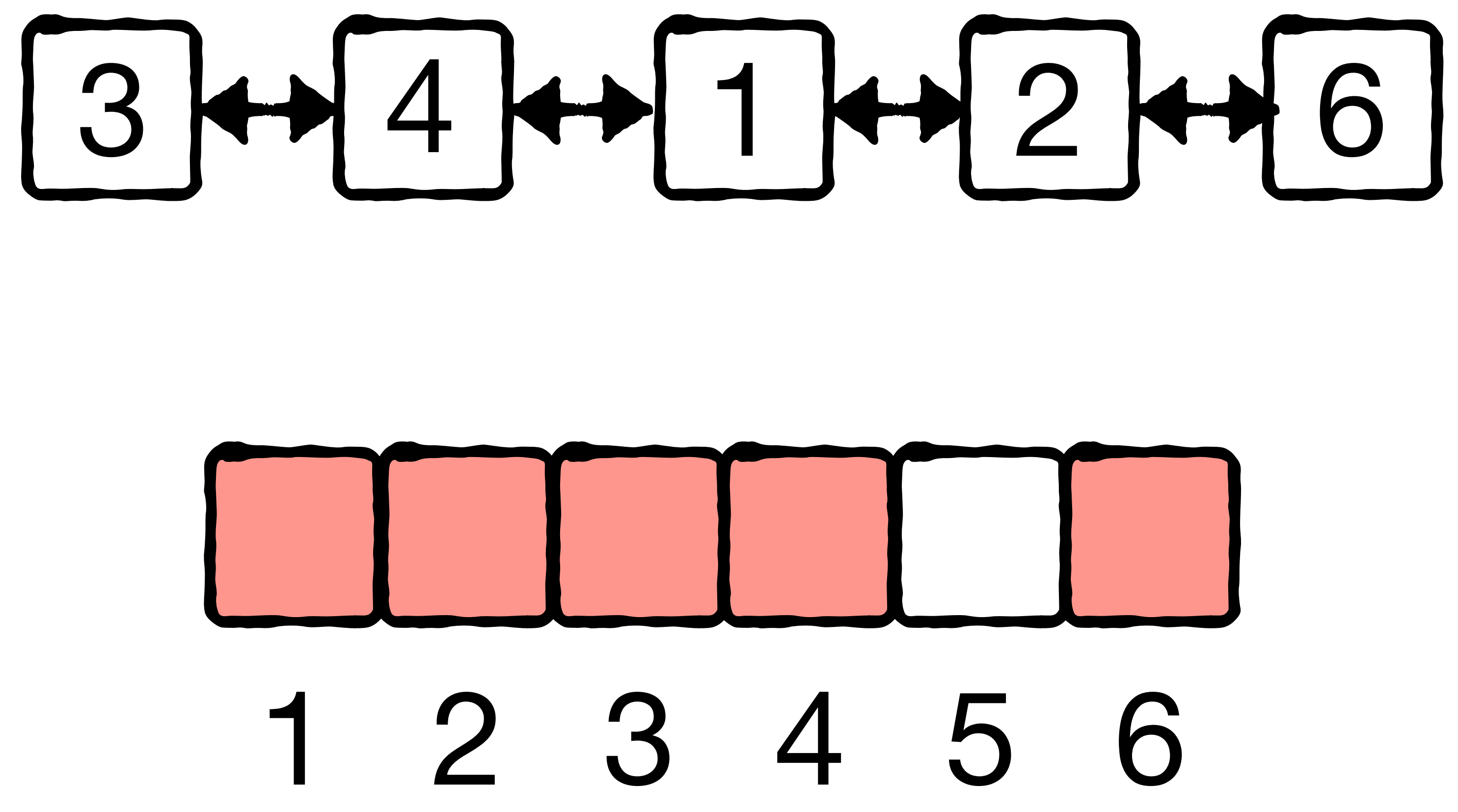
LRU - Least Recently Used

Evict page accessed longest ago

Queue

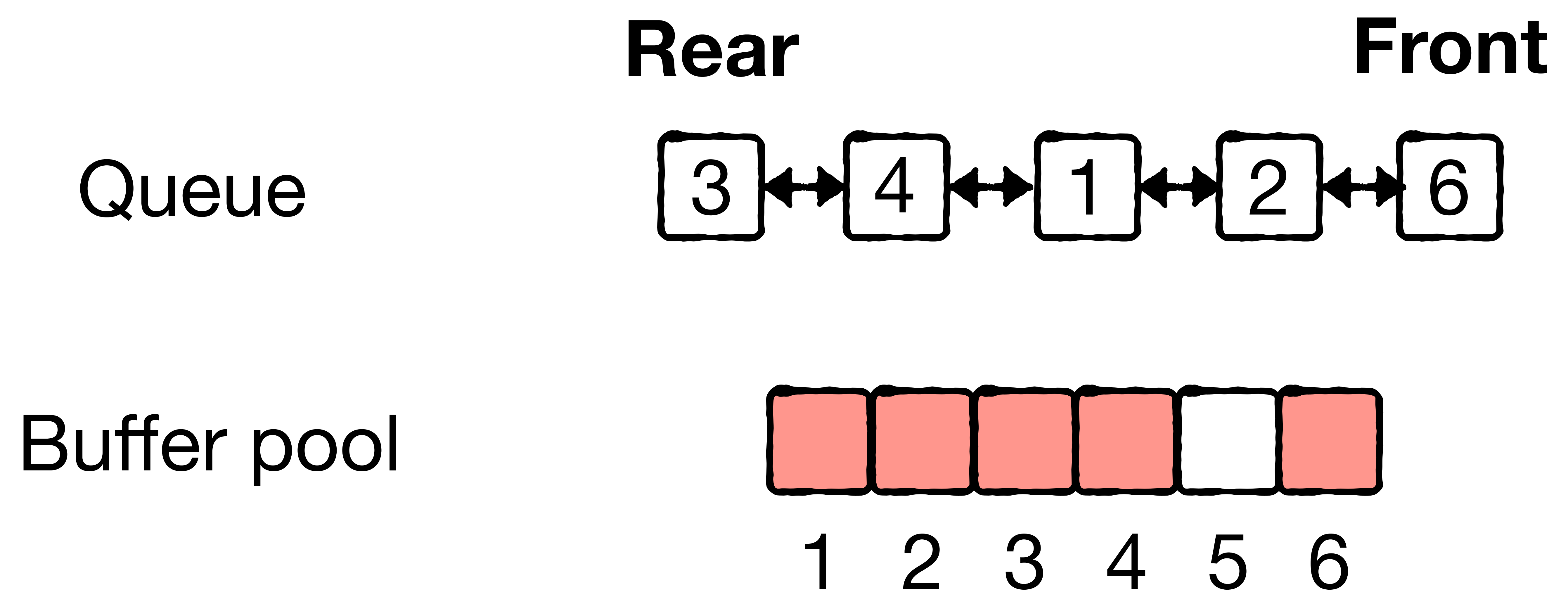
Doubly Linked list

Buffer pool



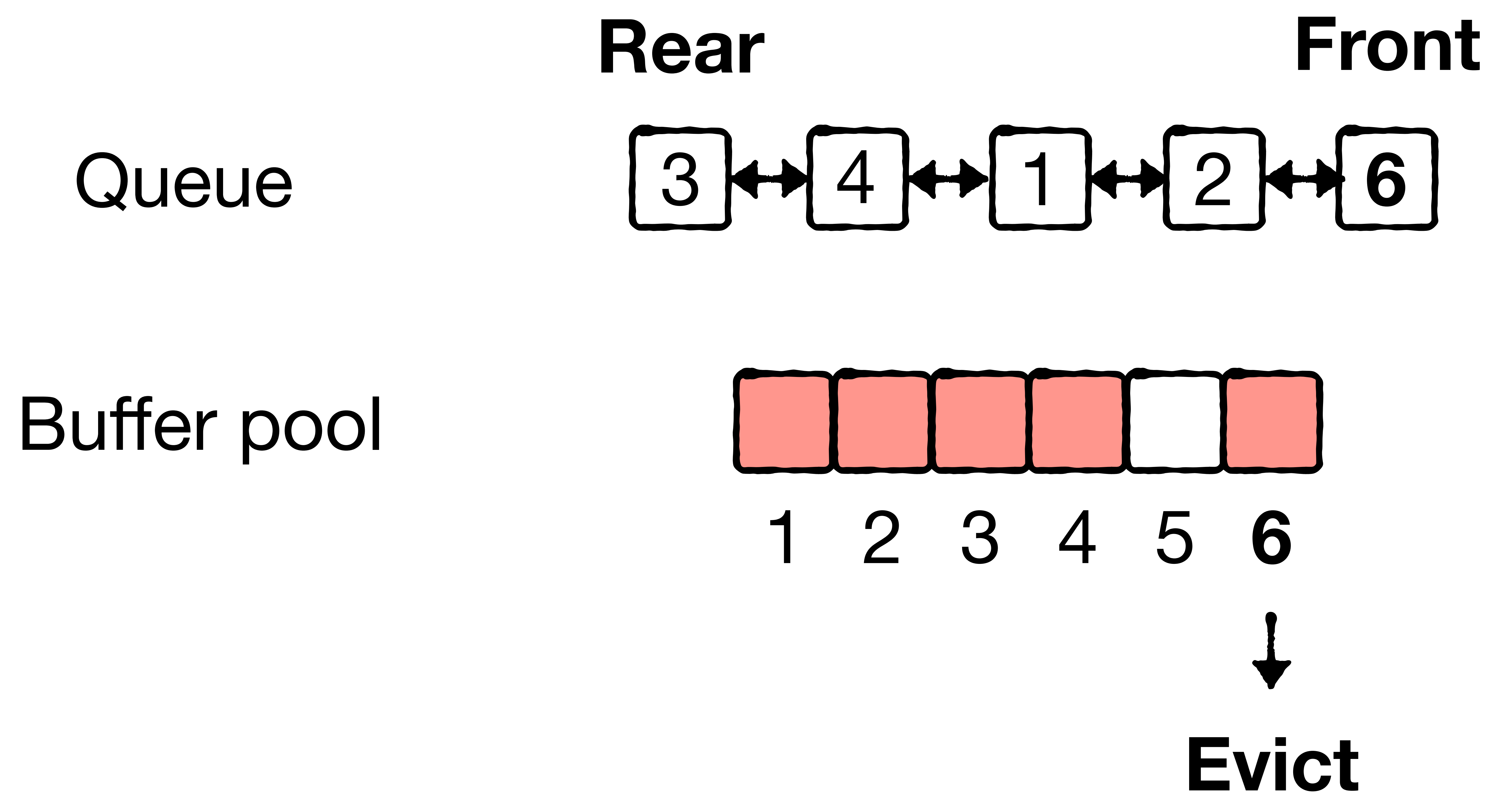
LRU - Least Recently Used

Evict page accessed longest ago



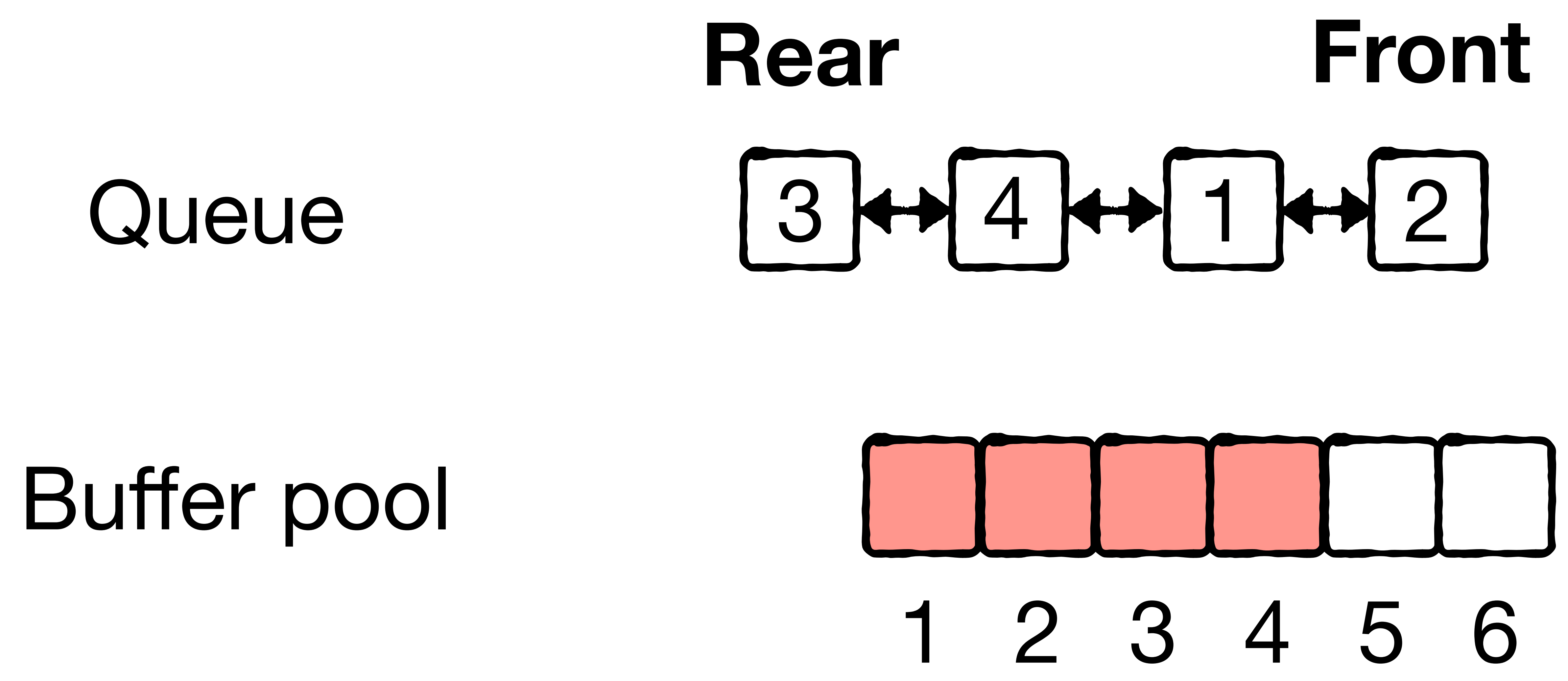
LRU - Least Recently Used

Evict page accessed longest ago



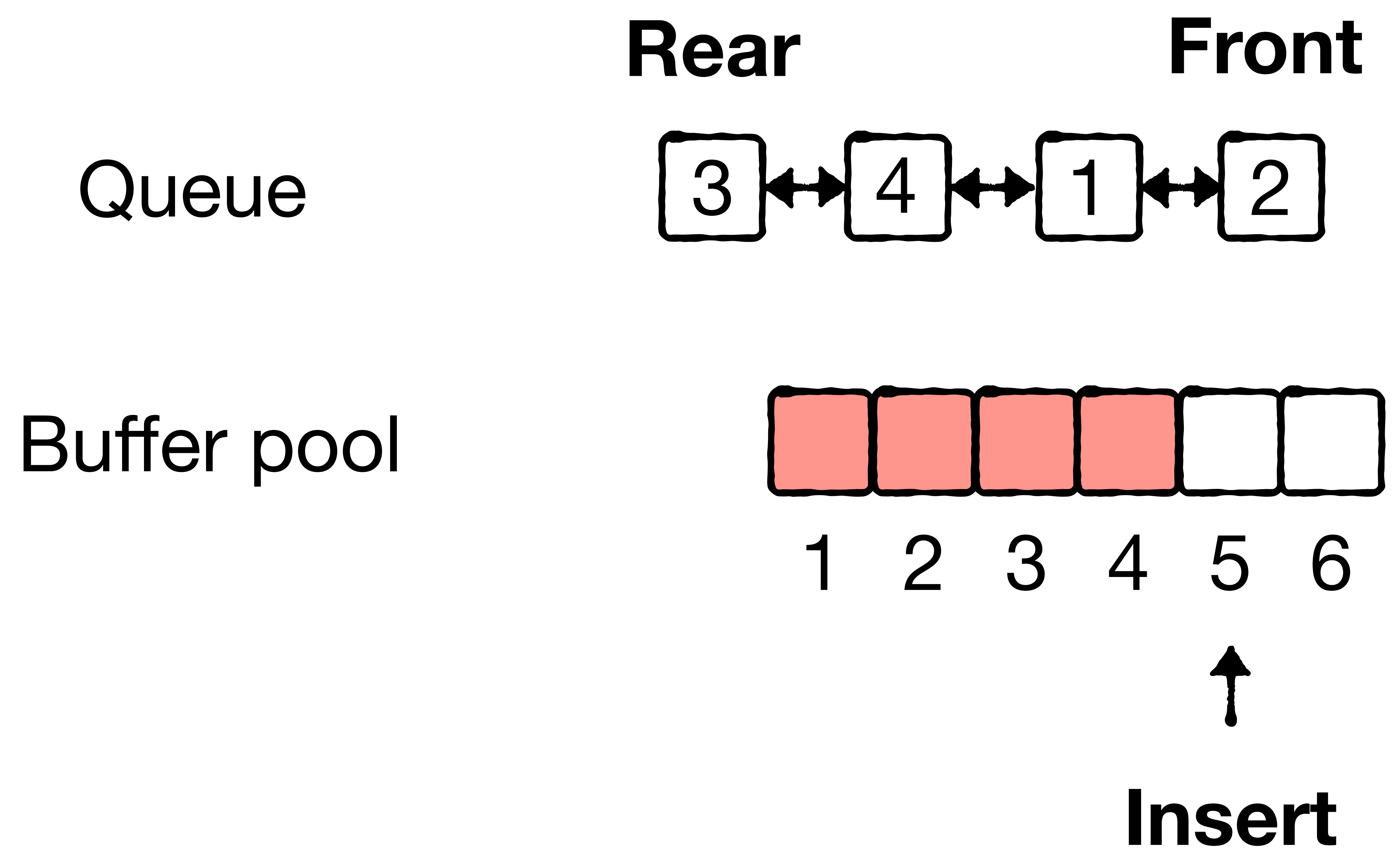
LRU - Least Recently Used

Evict page accessed longest ago



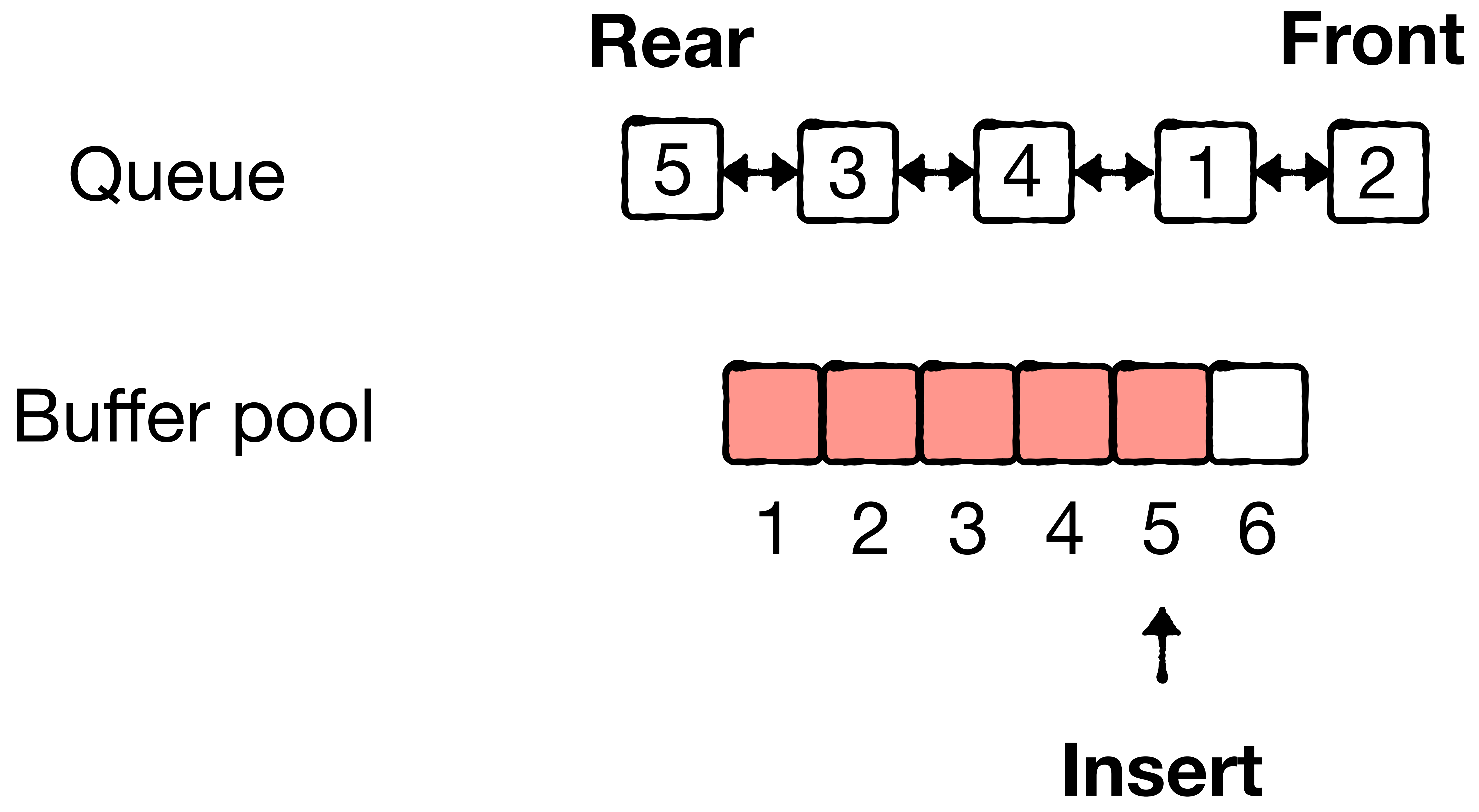
LRU - Least Recently Used

Evict page accessed longest ago



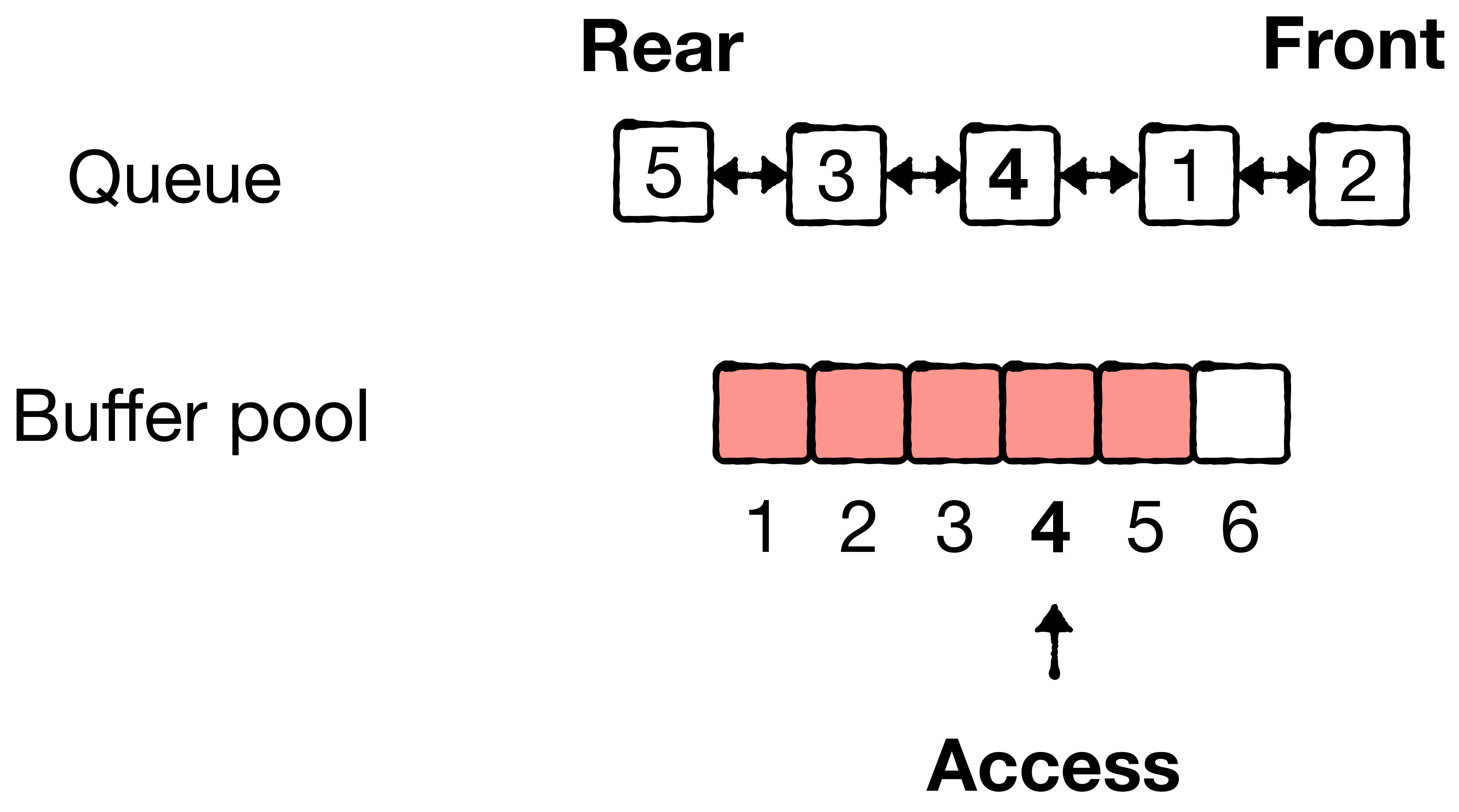
LRU - Least Recently Used

Evict page accessed longest ago



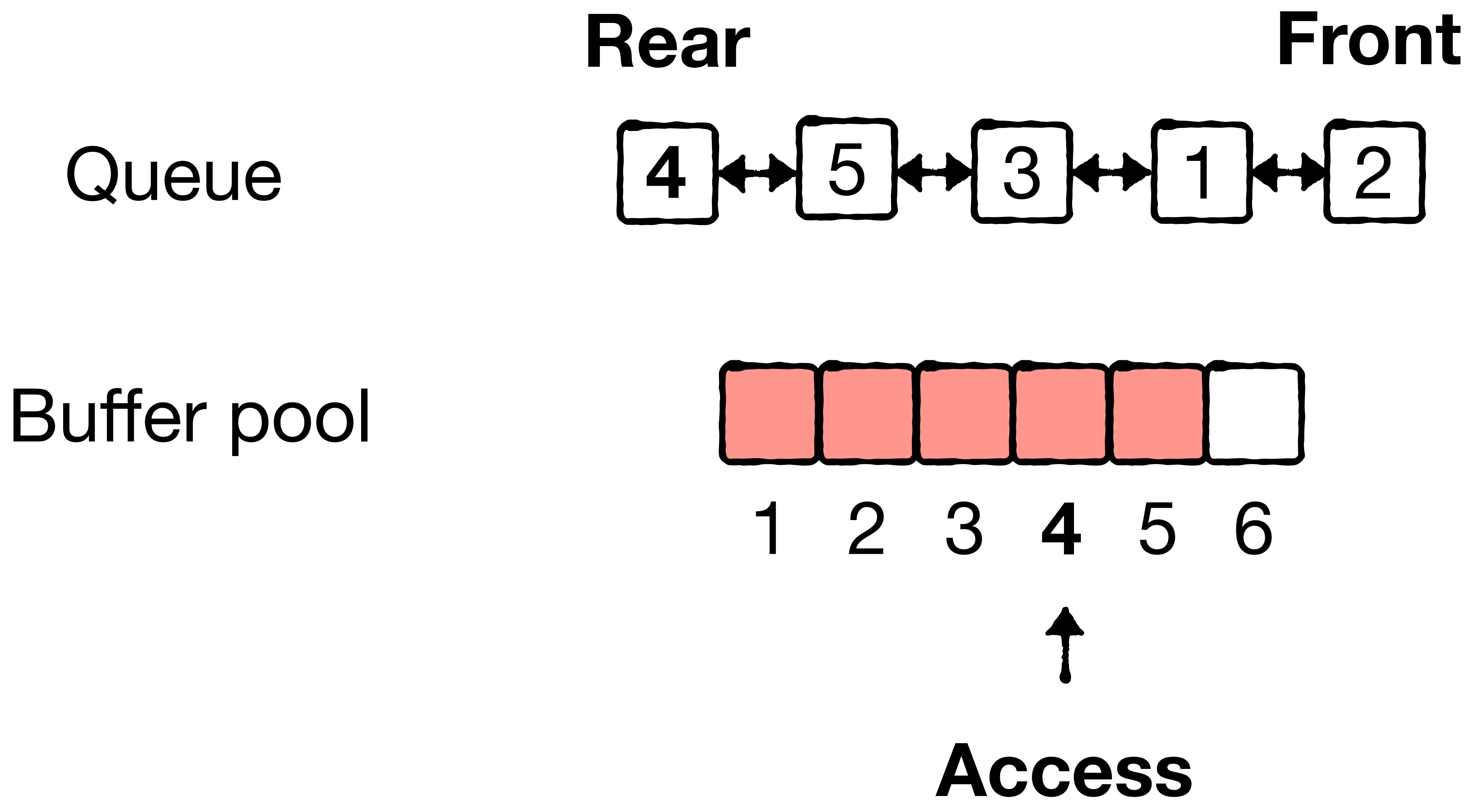
LRU - Least Recently Used

Evict page accessed longest ago



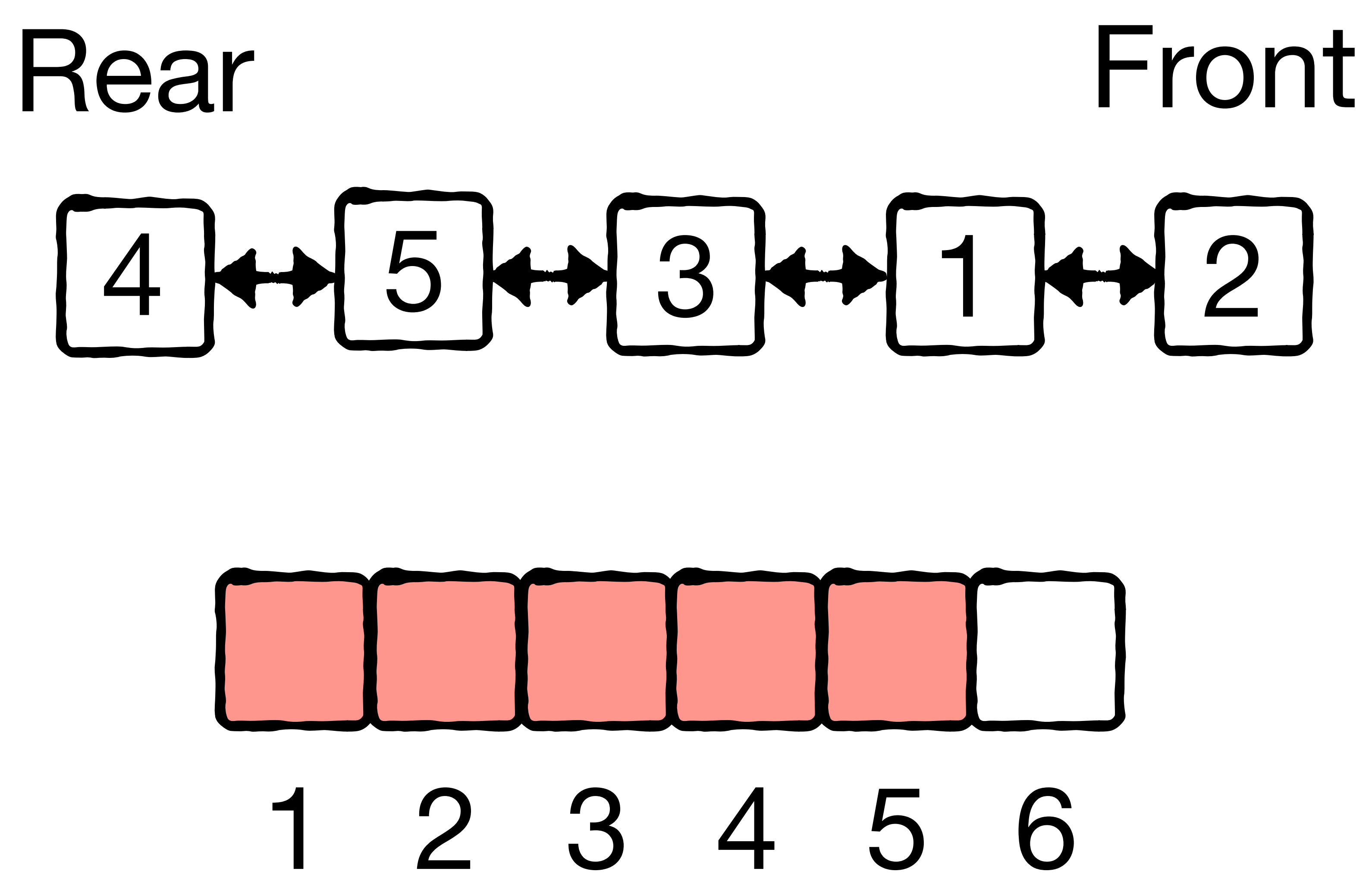
LRU - Least Recently Used

Evict page accessed longest ago

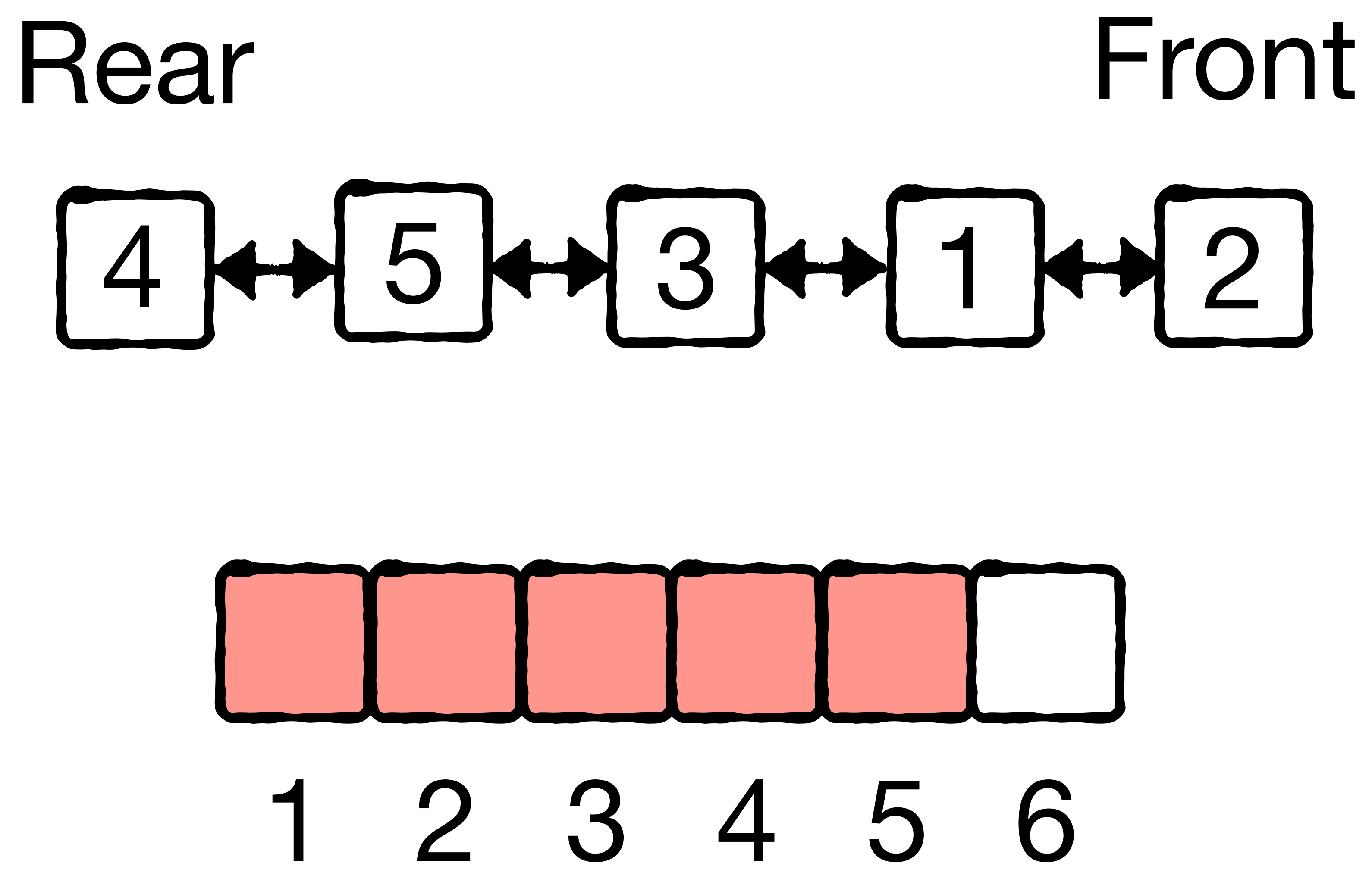


LRU - Least Recently Used

Problems?



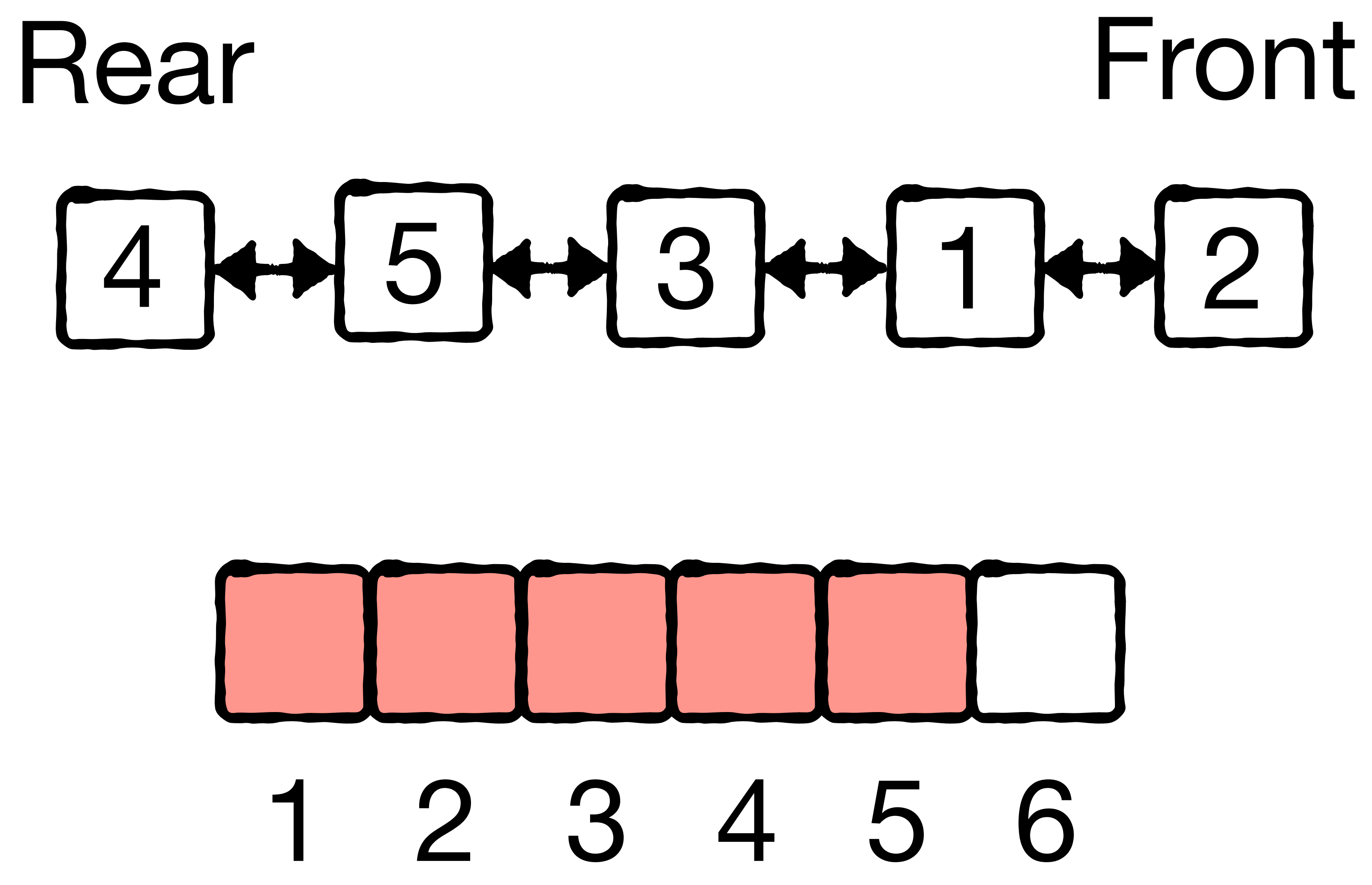
LRU - Least Recently Used



Problems?

(1) CPU - addressed with clock

LRU - Least Recently Used

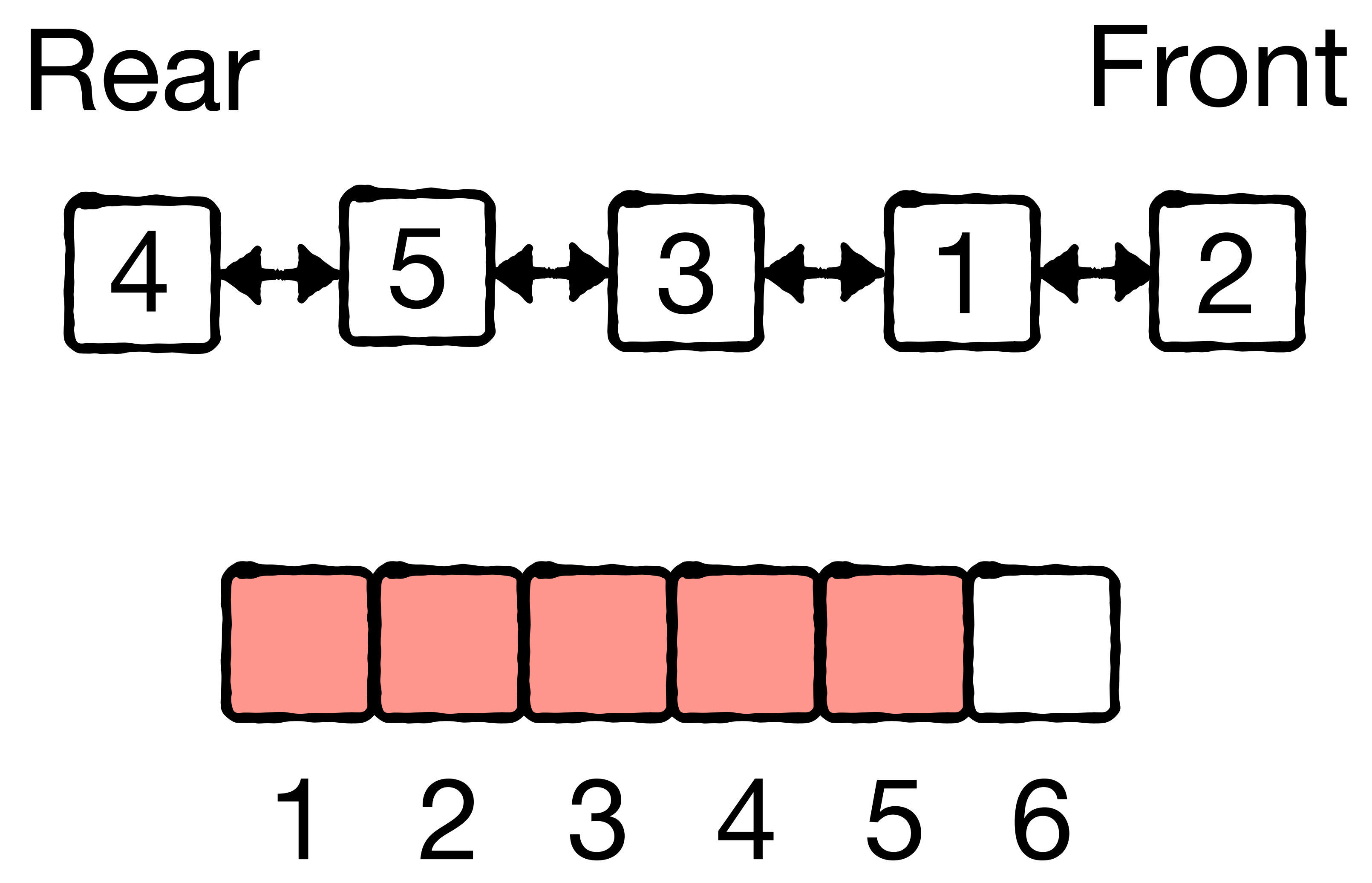


Problems?

(1) CPU - addressed with clock

(2) sequential flooding

LRU - Least Recently Used



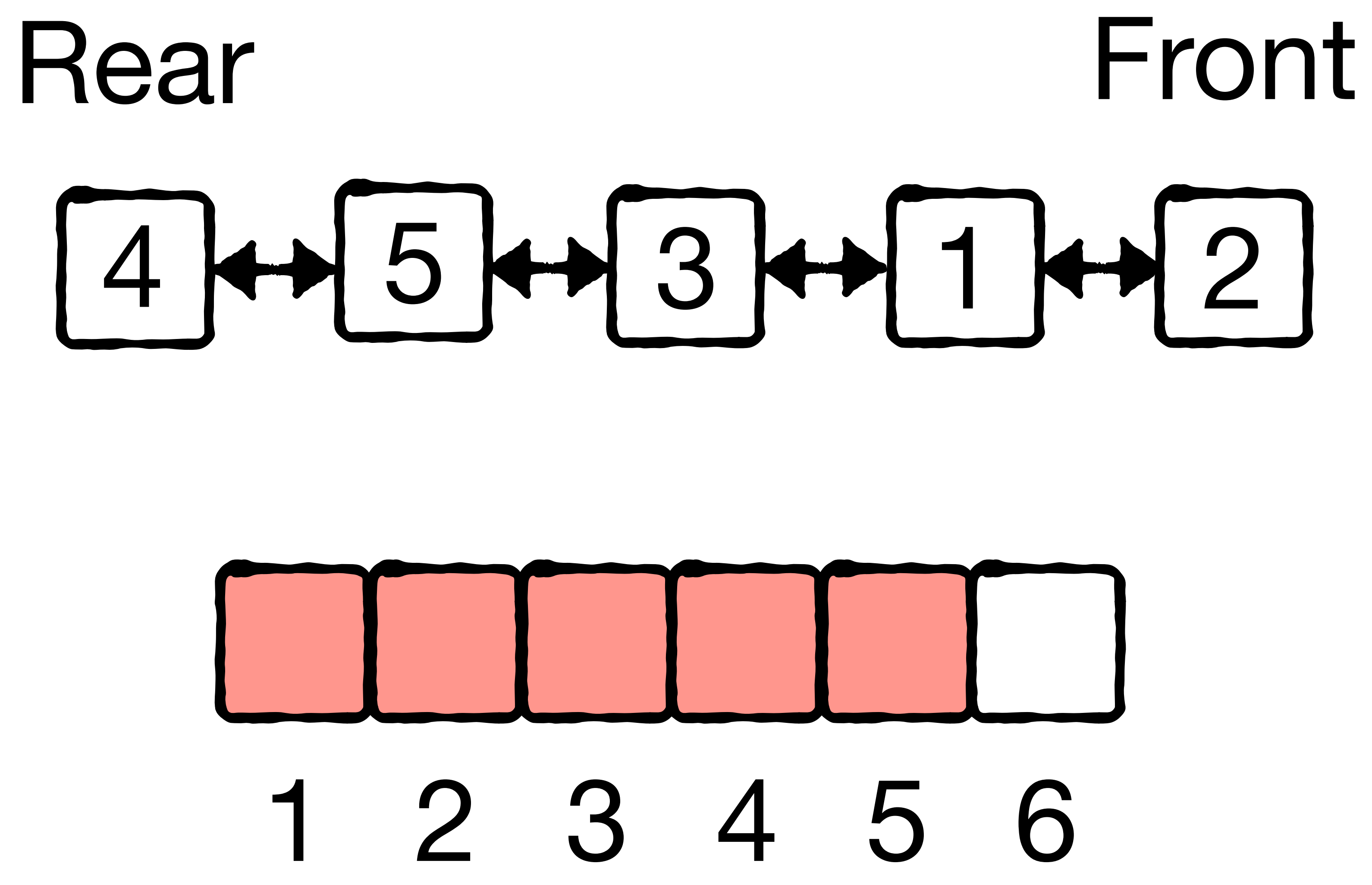
Problems?

(1) CPU - addressed with clock

(2) sequential flooding

(3) cold page read once may evict hotter page & take space in pool for long time

LRU - Least Recently Used



Problems?

(1) CPU - addressed with clock

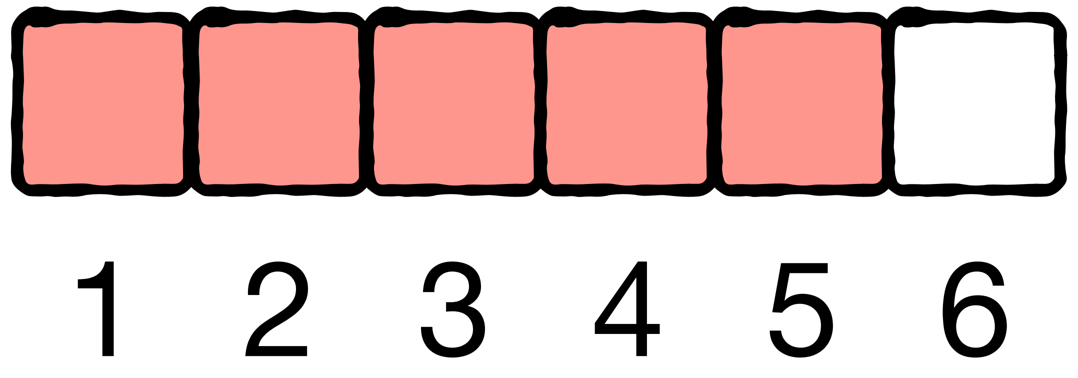
(2) sequential flooding

(3) cold page read once may evict hotter page & take space in pool for long time

Let's address (3)

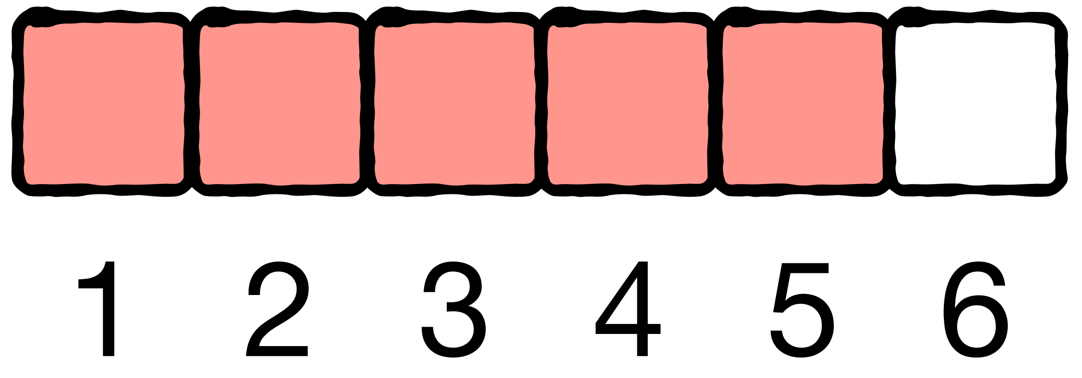
LRU/K - K^{th} Least Recently Used

Evict page whose K^{th} most recent access is longest ago



LRU/K - K^th Least Recently Used

Evict page whose K^th most recent access is longest ago



The LRU-K page replacement algorithm for database disk buffering

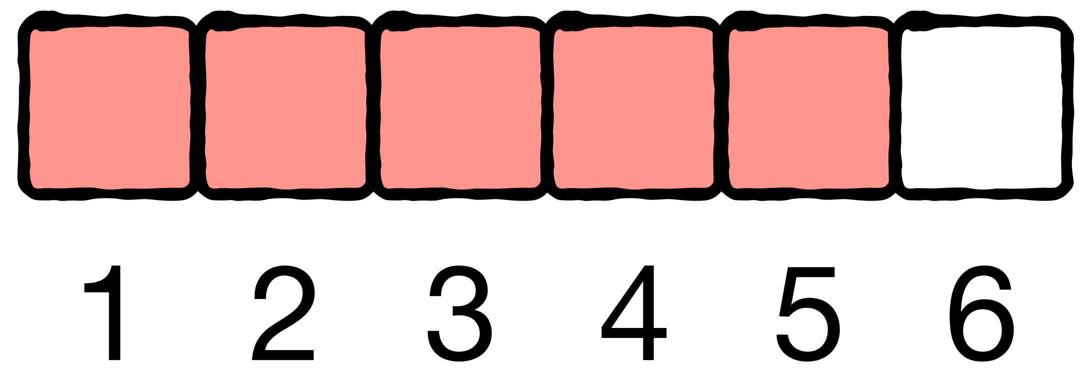
Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum

SIGMOD Record 1993

LRU/K - Kth Least Recently Used

Evict page whose Kth most recent access is longest ago

Generalizes LRU (as LRU/1)

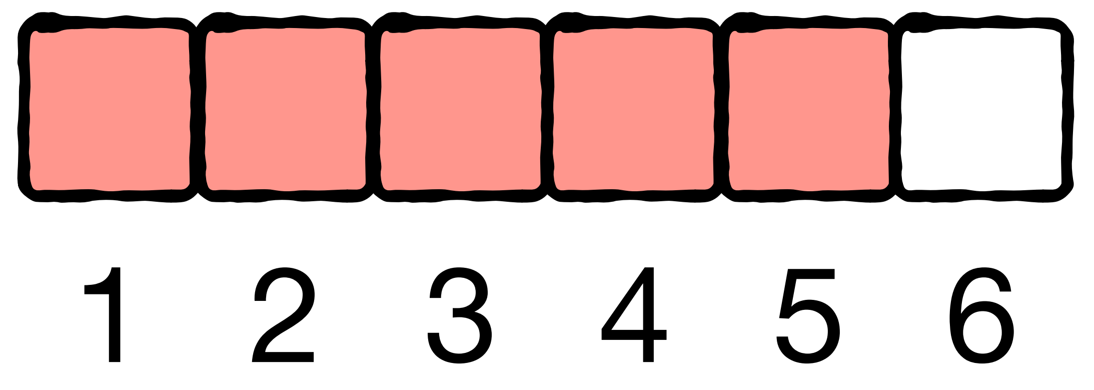


LRU/K - K^th^ Least Recently Used

Evict page whose K^th^ most recent access is longest ago

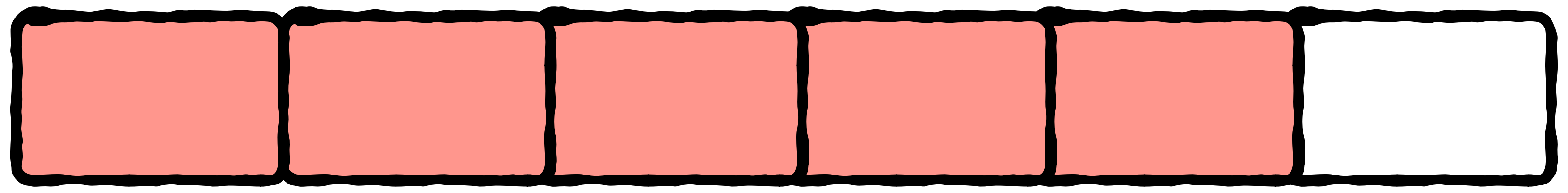
Generalizes LRU (as LRU/1)

LRU/2 achieves most of the benefits, so let's focus on that



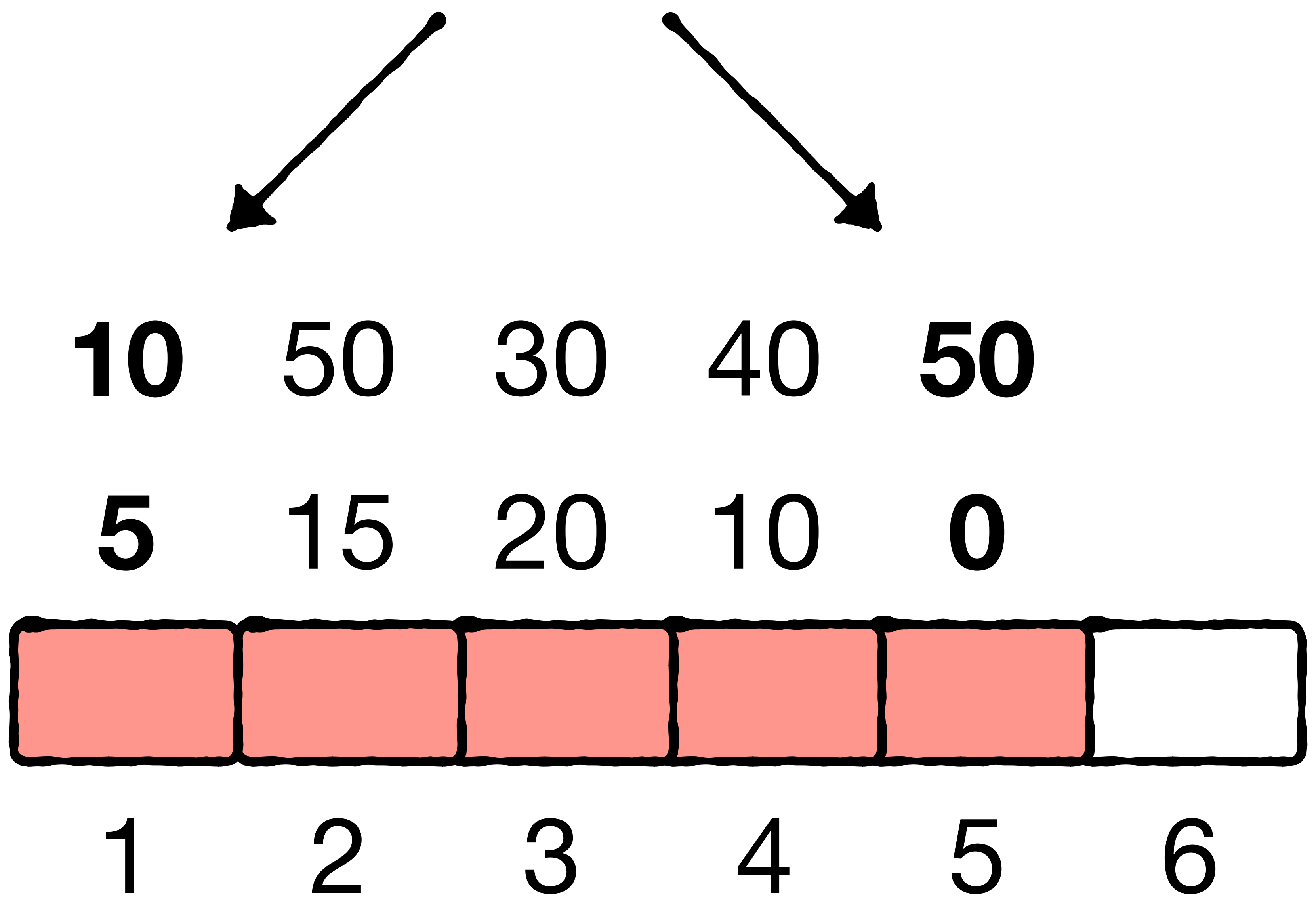
LRU/2 - 2nd Least Recently Used

Last access	10	50	30	40	50	
2nd last access	5	15	20	10	0	
	1	2	3	4	5	6



LRU/2 - 2nd Least Recently Used

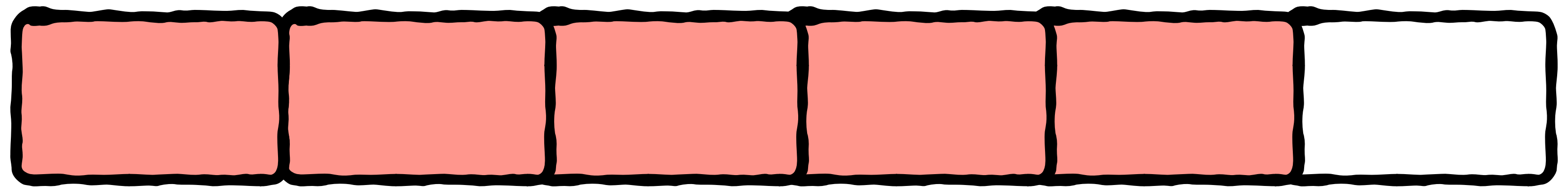
Which of these pages is less hot?



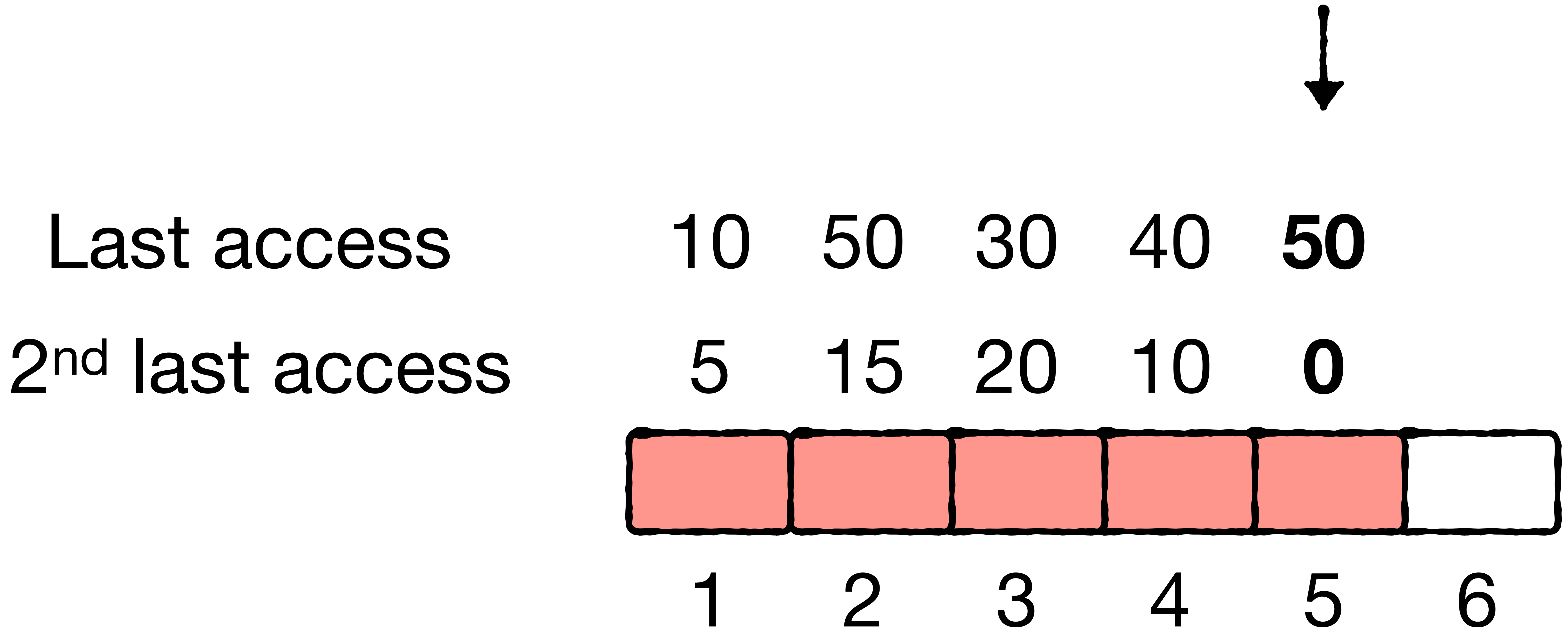
LRU/2 - 2nd Least Recently Used

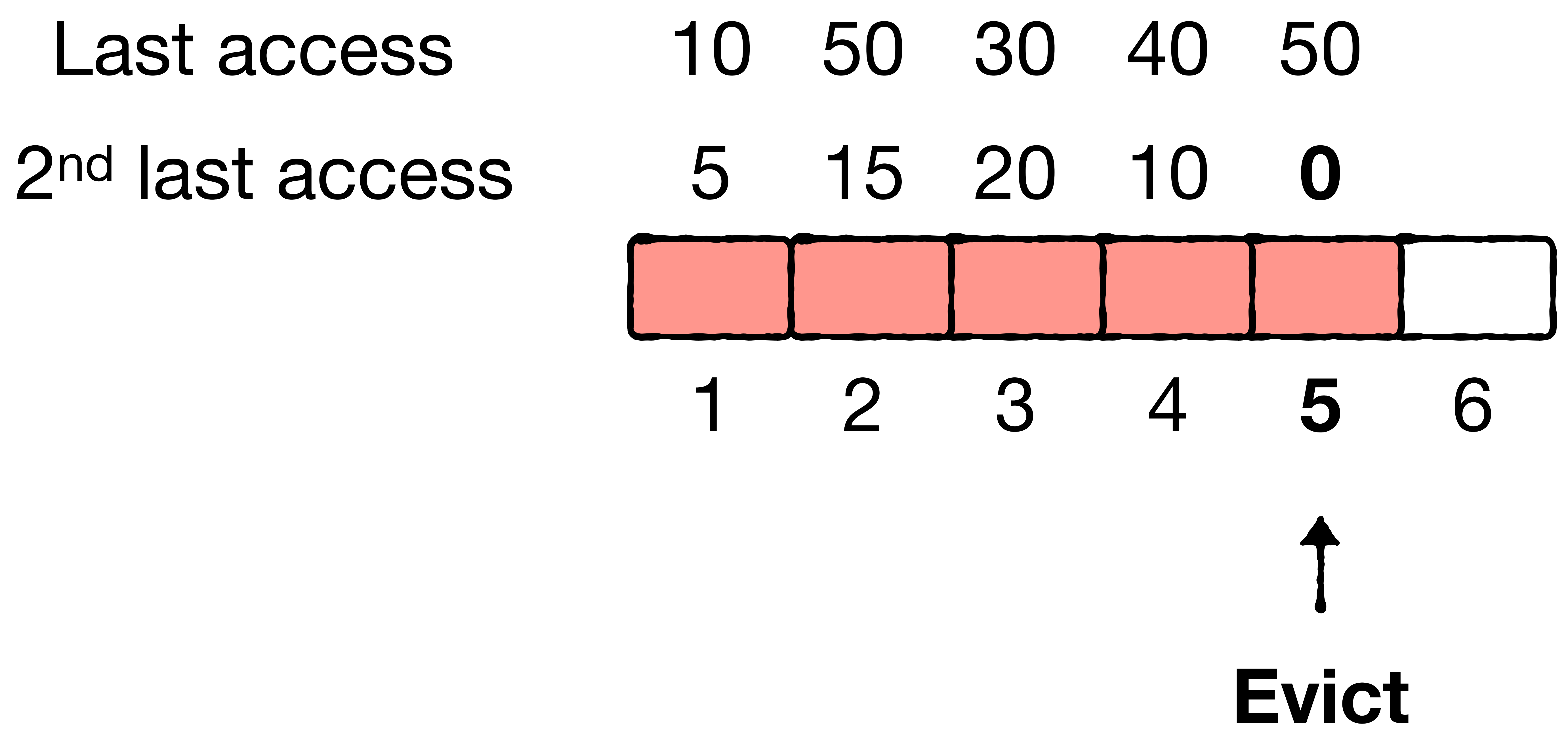
Intuition: evicting page whose penultimate access is longest ago implies the page is seldom read

Last access	10	50	30	40	50	
2nd last access	5	15	20	10	0	
	1	2	3	4	5	6

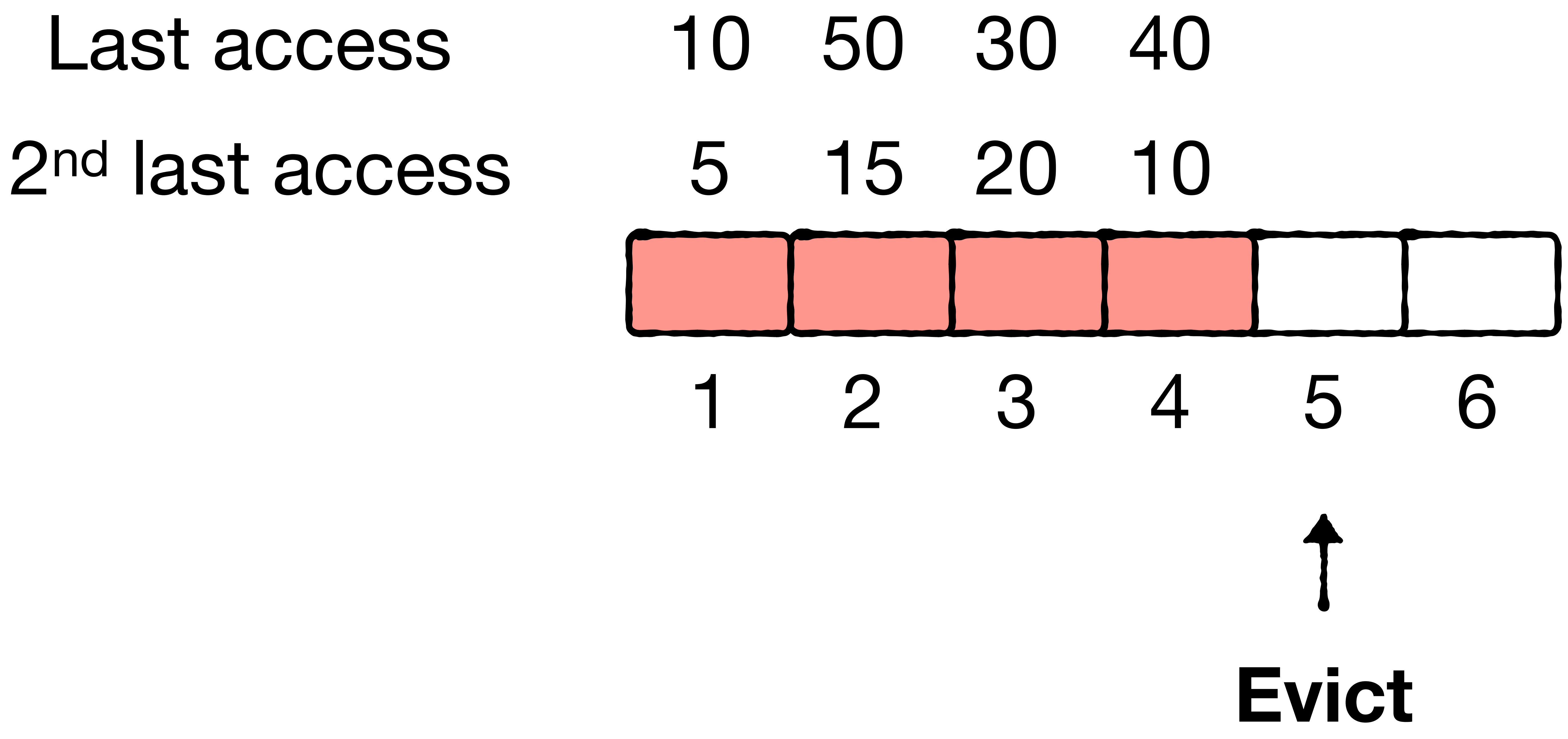


A recently read page can still be cold :)





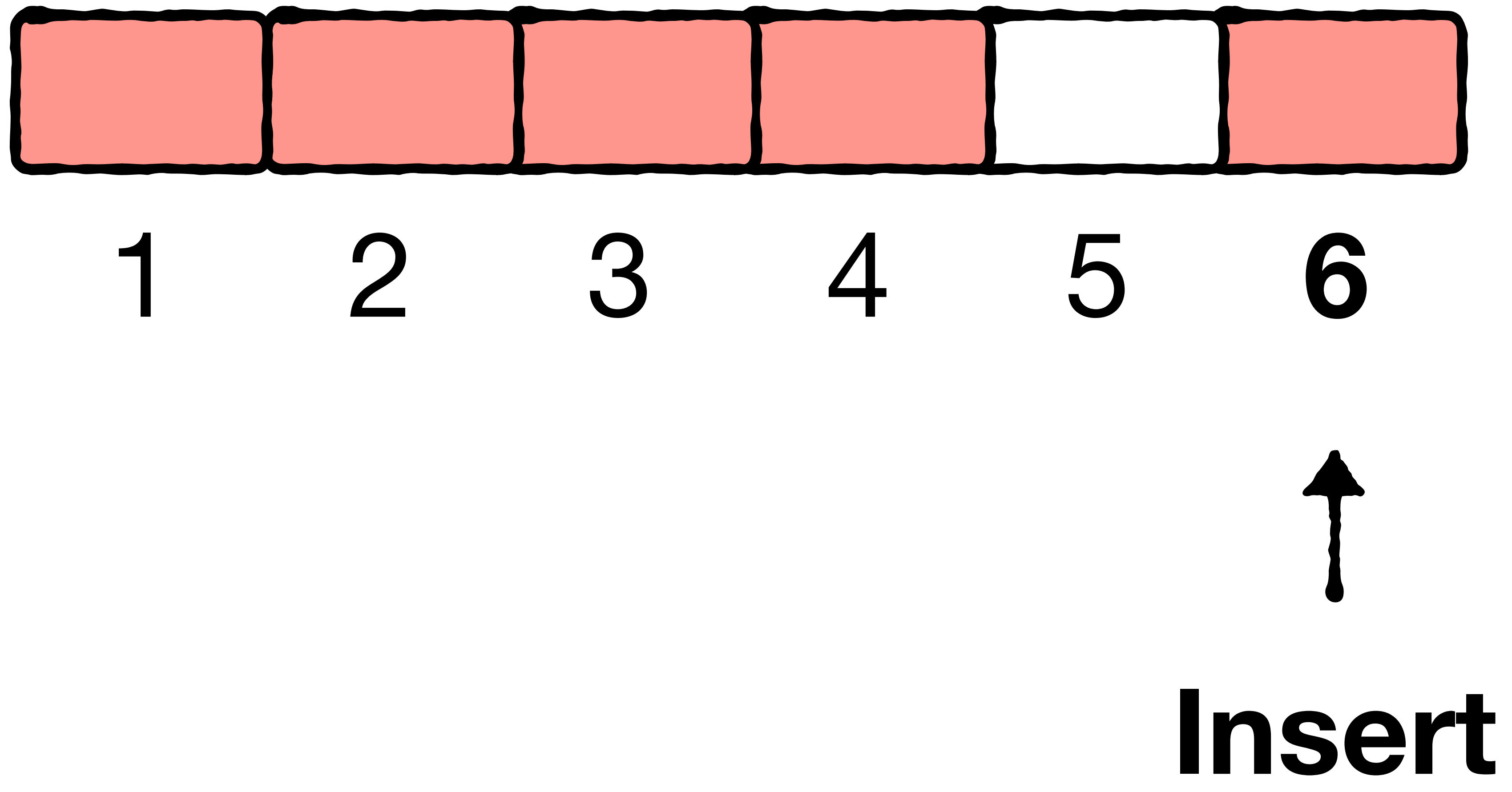
	1	2	3	4	**5**	6
Last access	10	50	30	40	50	
2nd last access	5	15	20	10	**0**	


Last access
10
50
30
40
2nd last access
5
15
20
10
1
2
3
4
5
6
Evict

Time now:

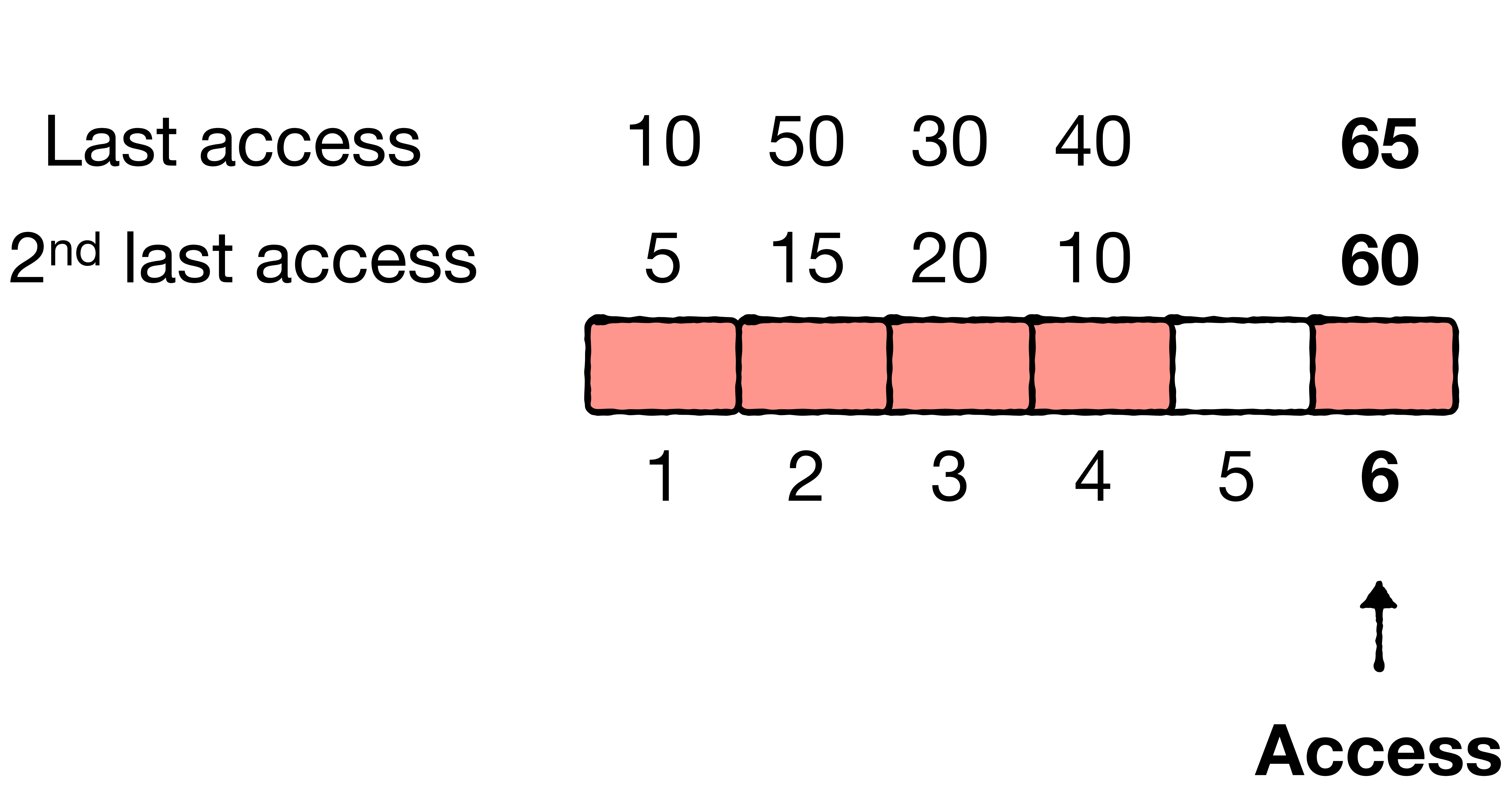
60

Last access	10	50	30	40		**60**
2nd last access	5	15	20	10		$-\infty$
	1	2	3	4	5	**6**



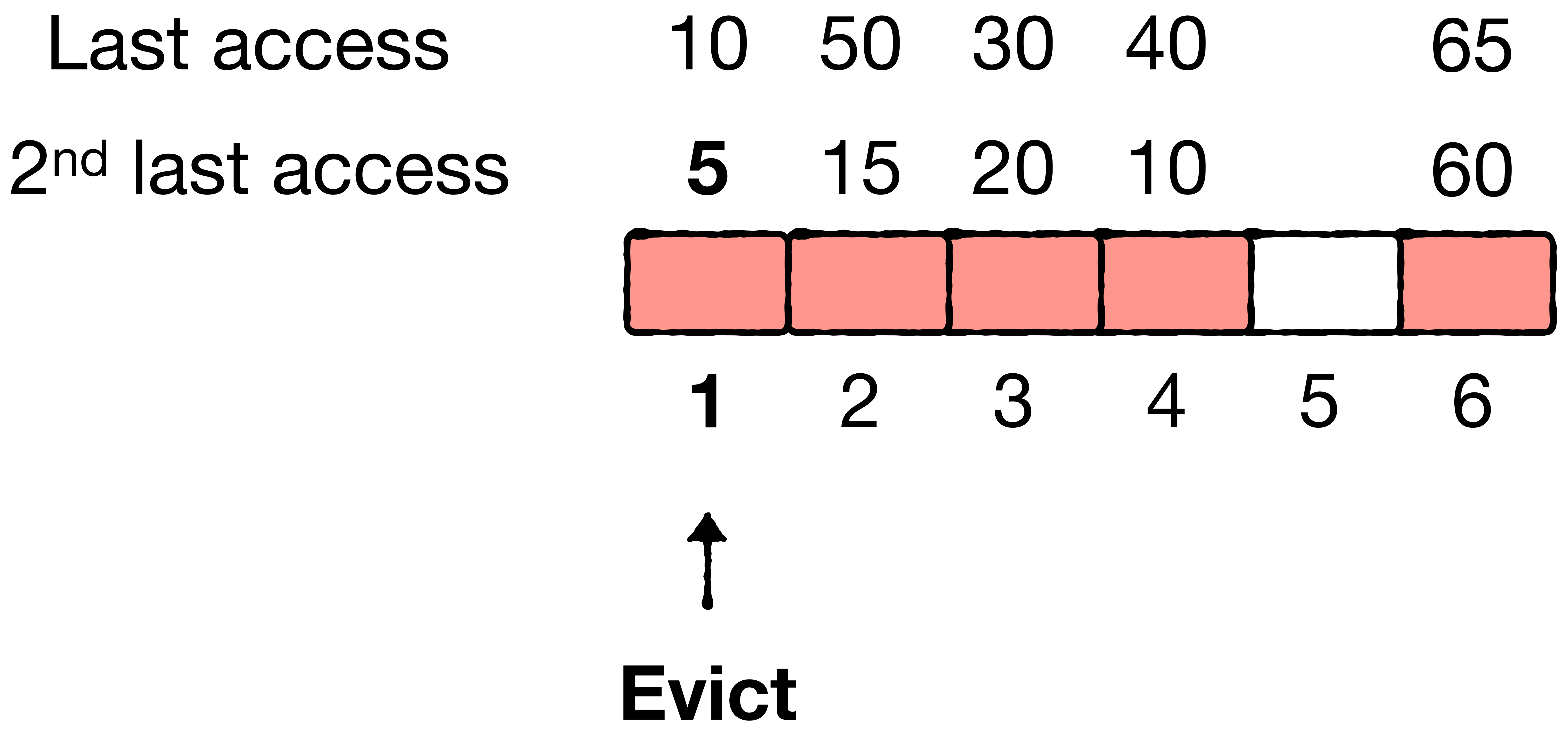
Time now:

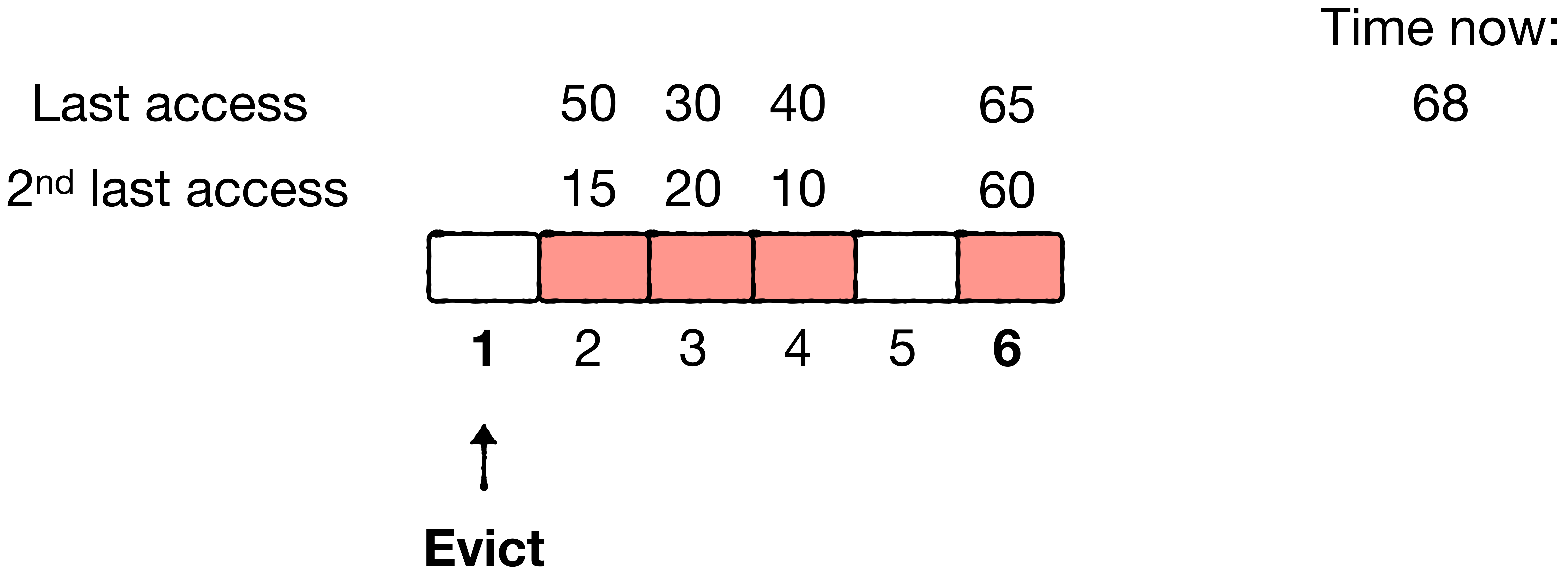
65



Last access
10 50 30 40 **65**
2nd last access
5 15 20 10 **60**
1 2 3 4 5 **6**
Access

Time now:
68


Last access 10 50 30 40 65
2nd last access 5 15 20 10 60
1 2 3 4 5 6
Evict

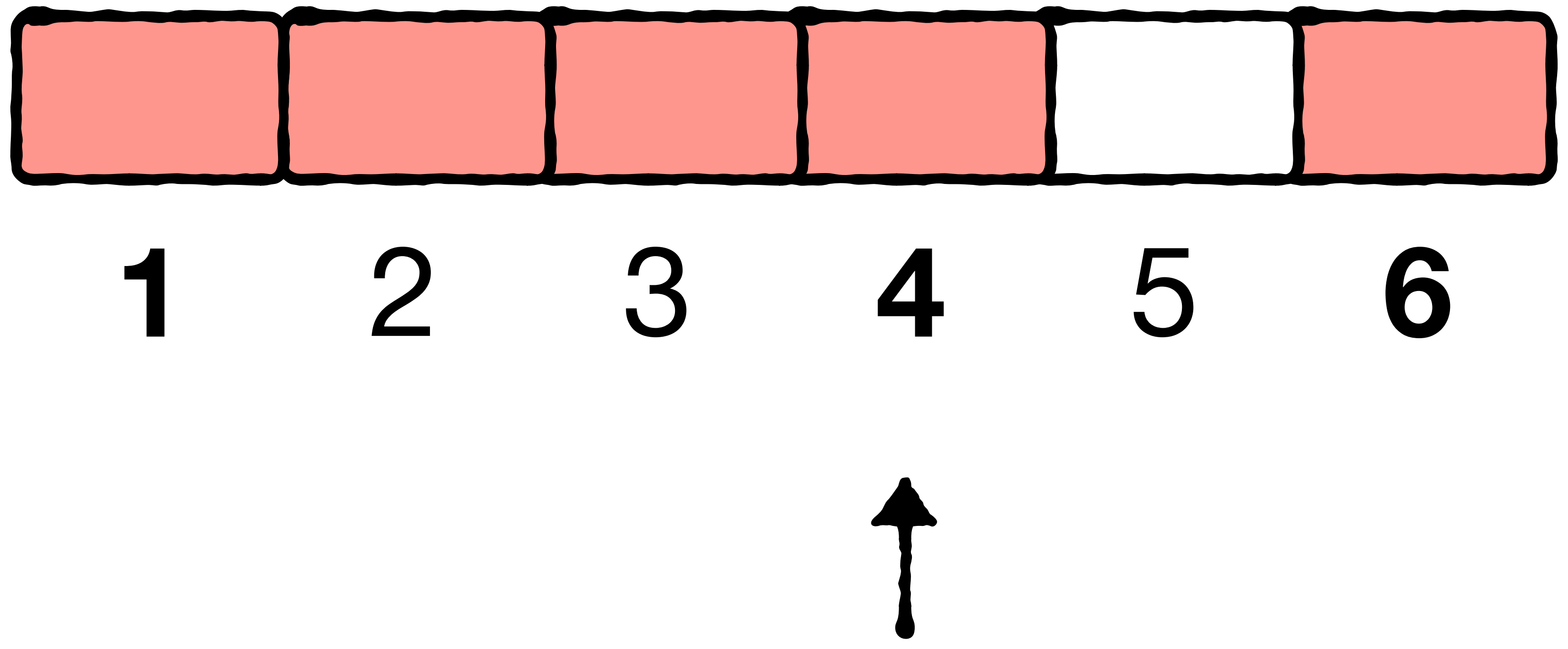

Time now:
68
Last access
50
30
40
65
2nd last access
15
20
10
60
1
2
3
4
5
6
Evict

Time now:
68

Last access	**68**	50	30	40		65
2nd last access	**-∞**	15	20	10		60
	🟥	🟥	🟥	🟥	⬜	🟥
	1	2	3	4	5	**6**
	↑					
	Insert					

Time now:
70

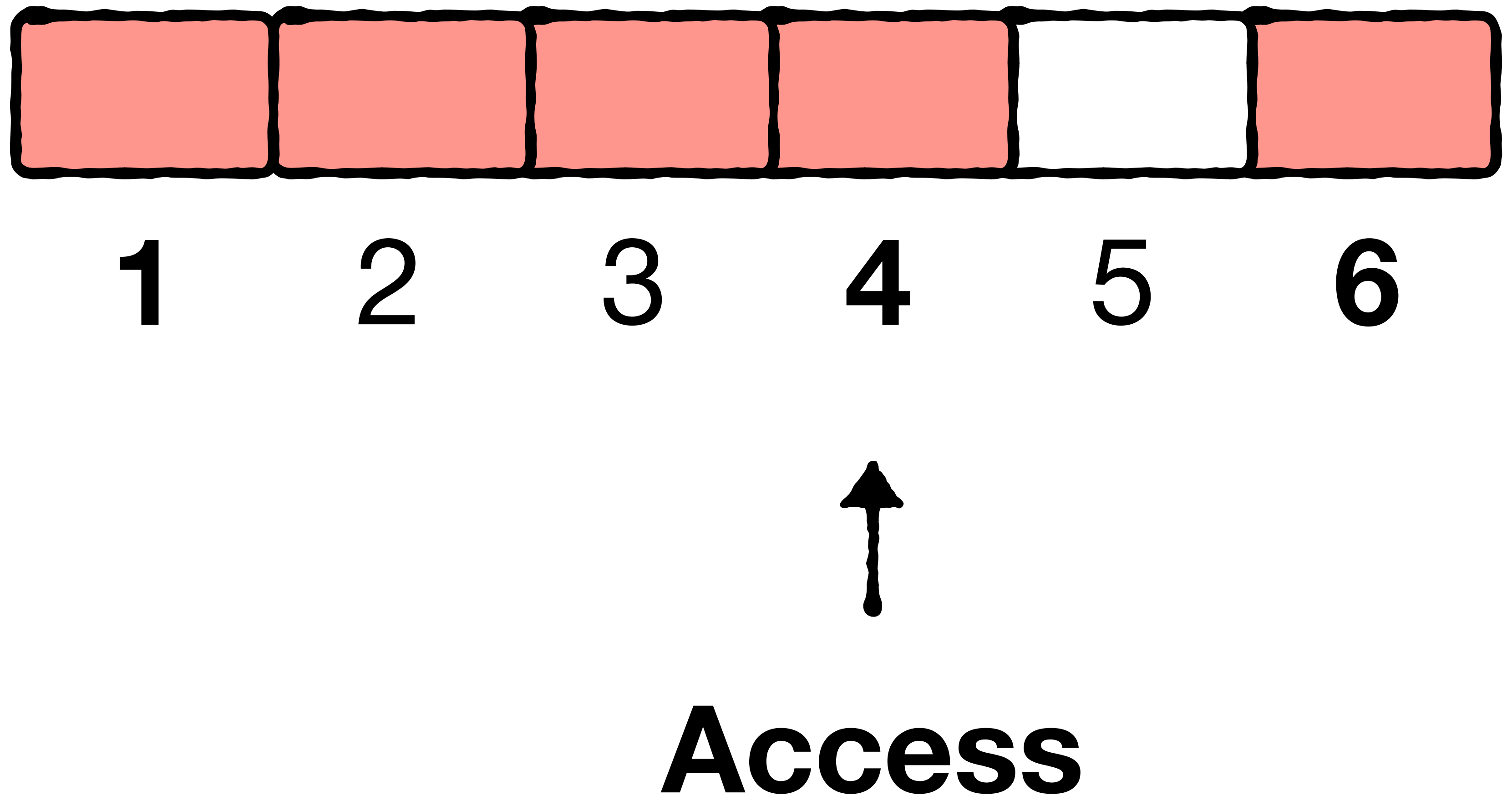
Last access	68	50	30	**40**		65
2nd last access	-∞	15	20	**10**		60

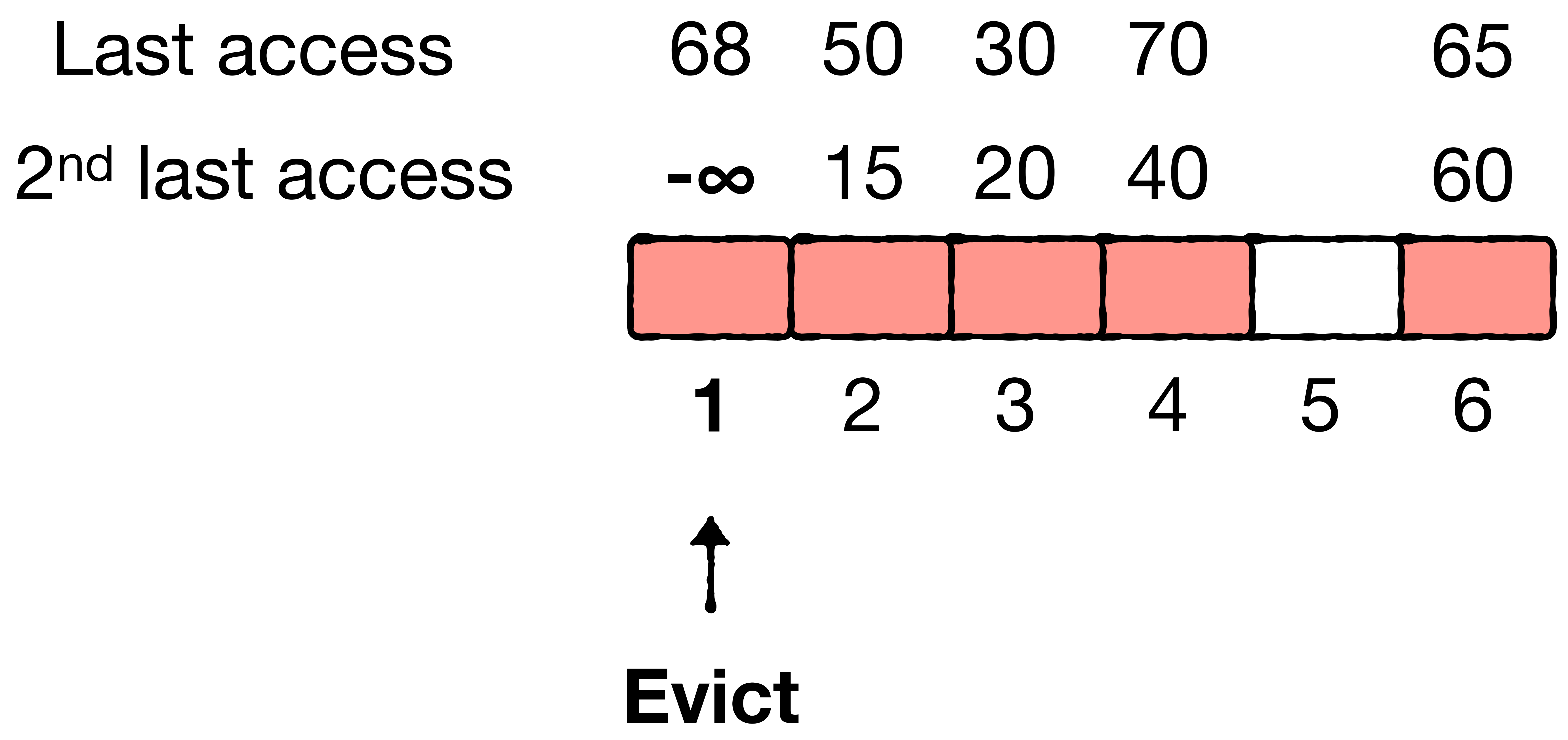


Access

Time now: 70

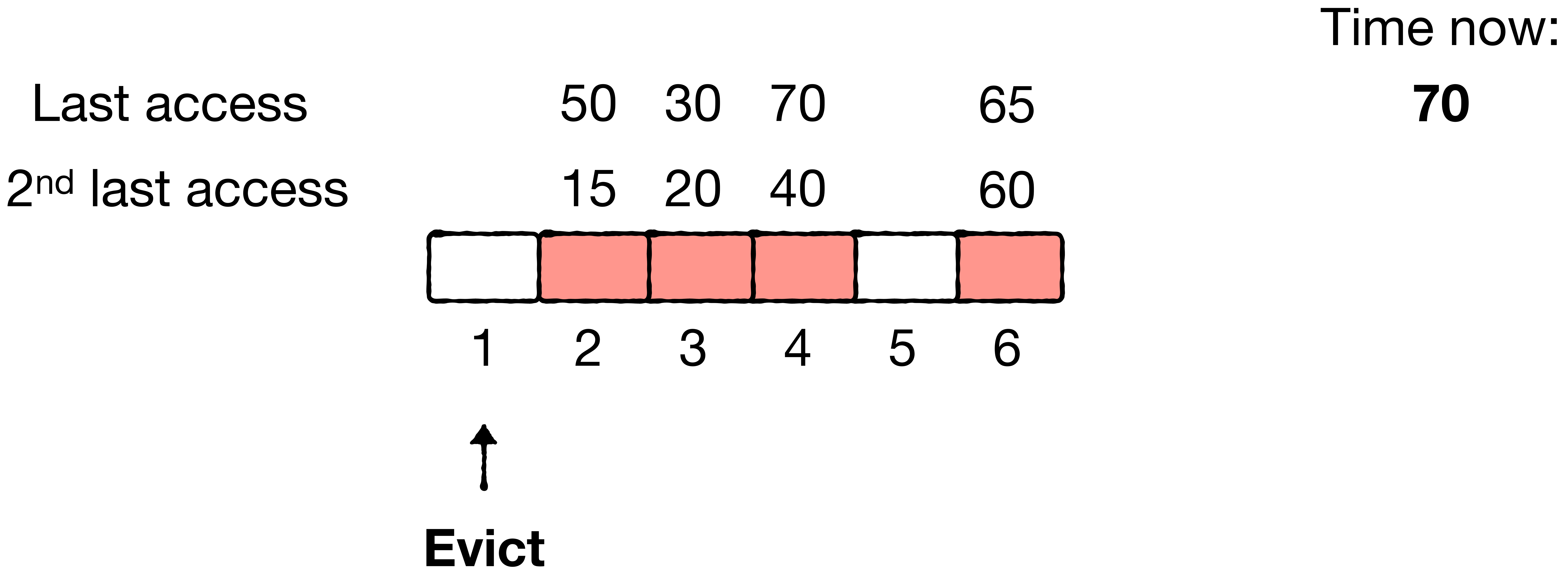
Last access	68	50	30	**70**		65
2nd last access	$-\infty$	15	20	**40**		60
	1	2	3	**4**	5	**6**




Last access
68 50 30 70 65
2nd last access
-∞ 15 20 40 60
1 2 3 4 5 6
Evict

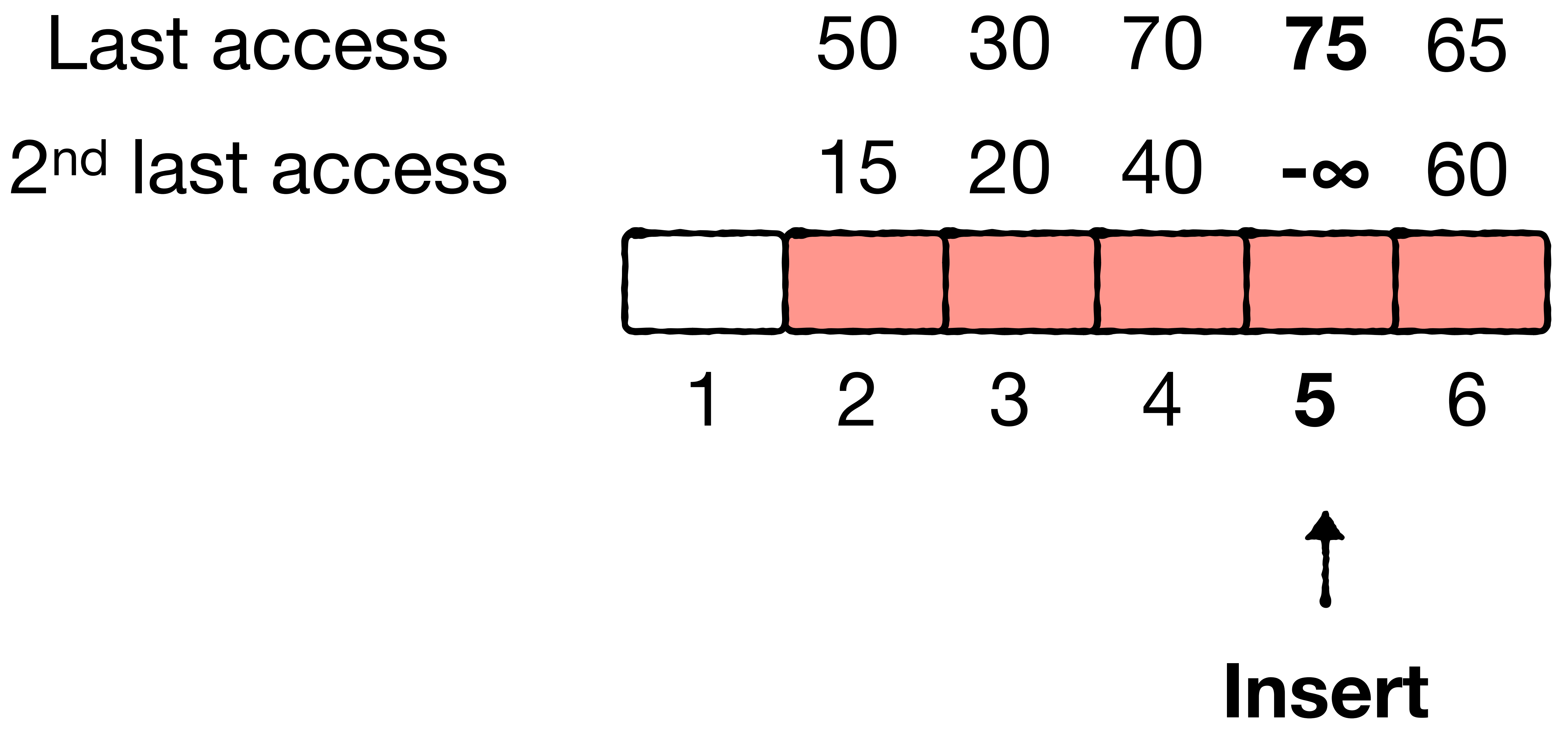
Time now:

70

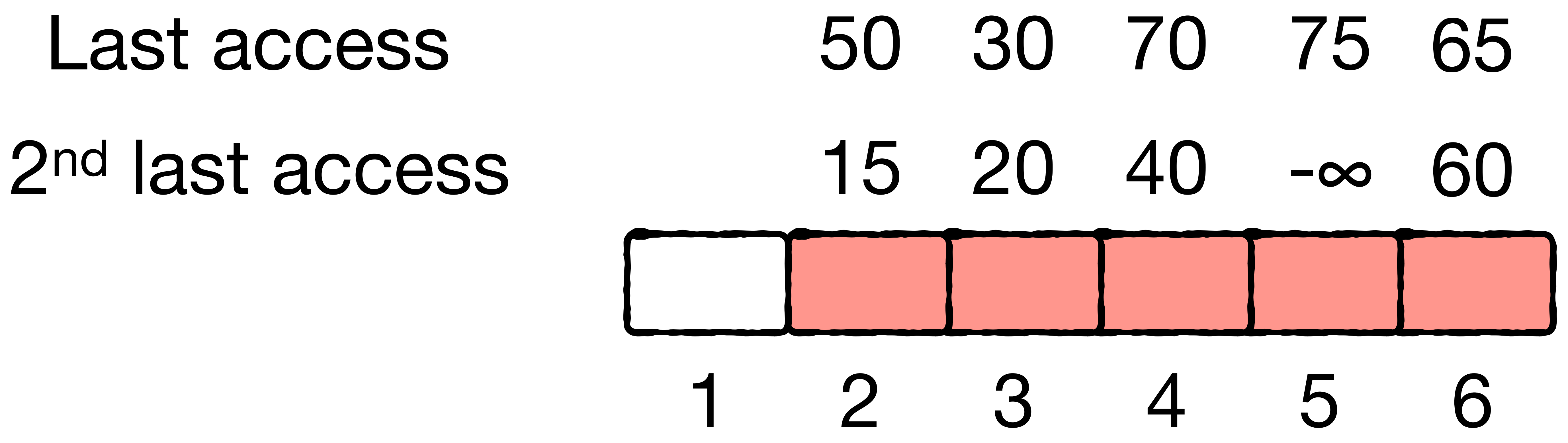

Time now:
70
Last access
50 30 70 65
2nd last access
15 20 40 60
1 2 3 4 5 6
Evict

Time now:

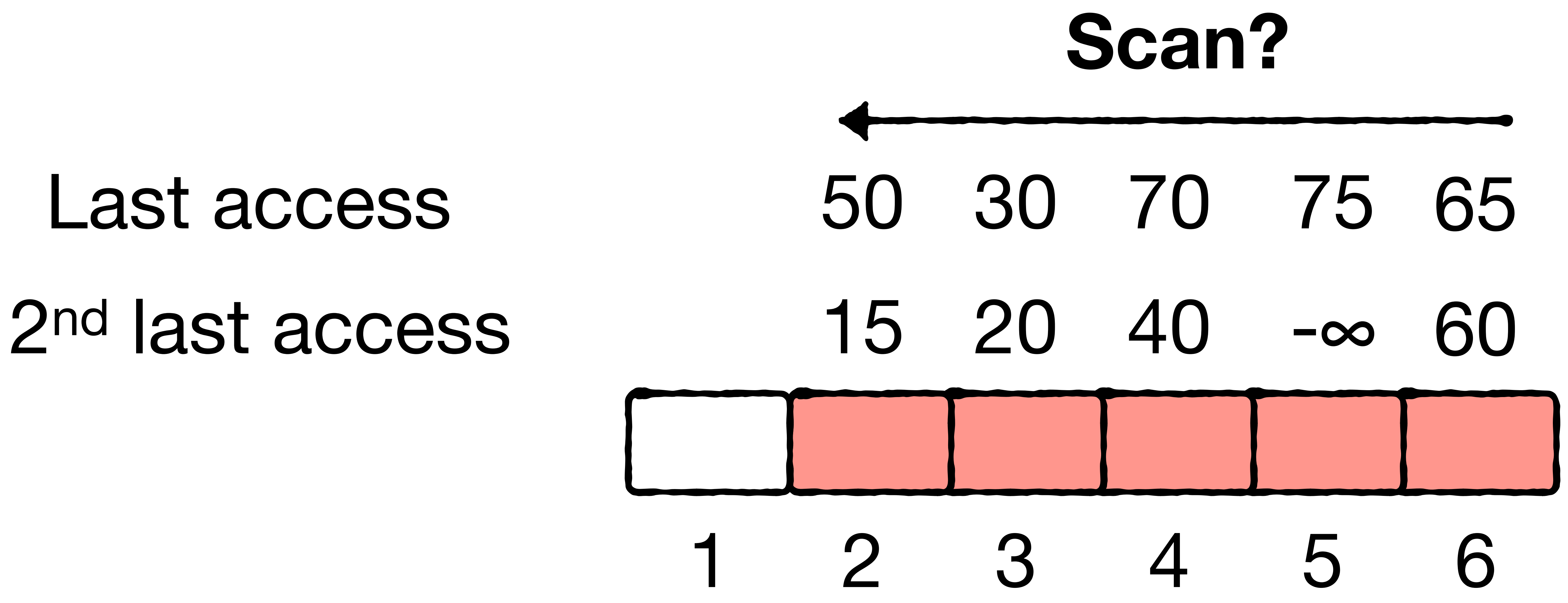
75


Last access
50 30 70 **75** 65
2nd last access
15 20 40 **-∞** 60
1 2 3 4 **5** 6
Insert

How to find 2nd LRU entry?

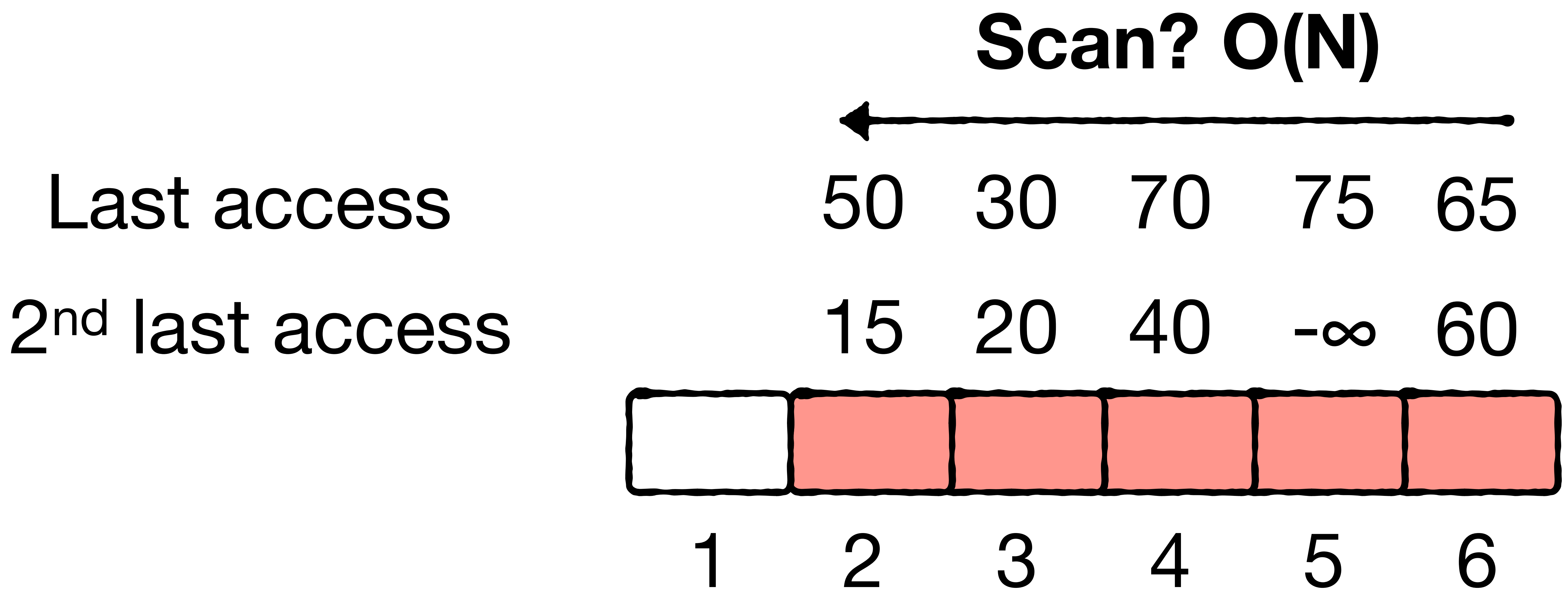


How to find 2nd LRU entry?



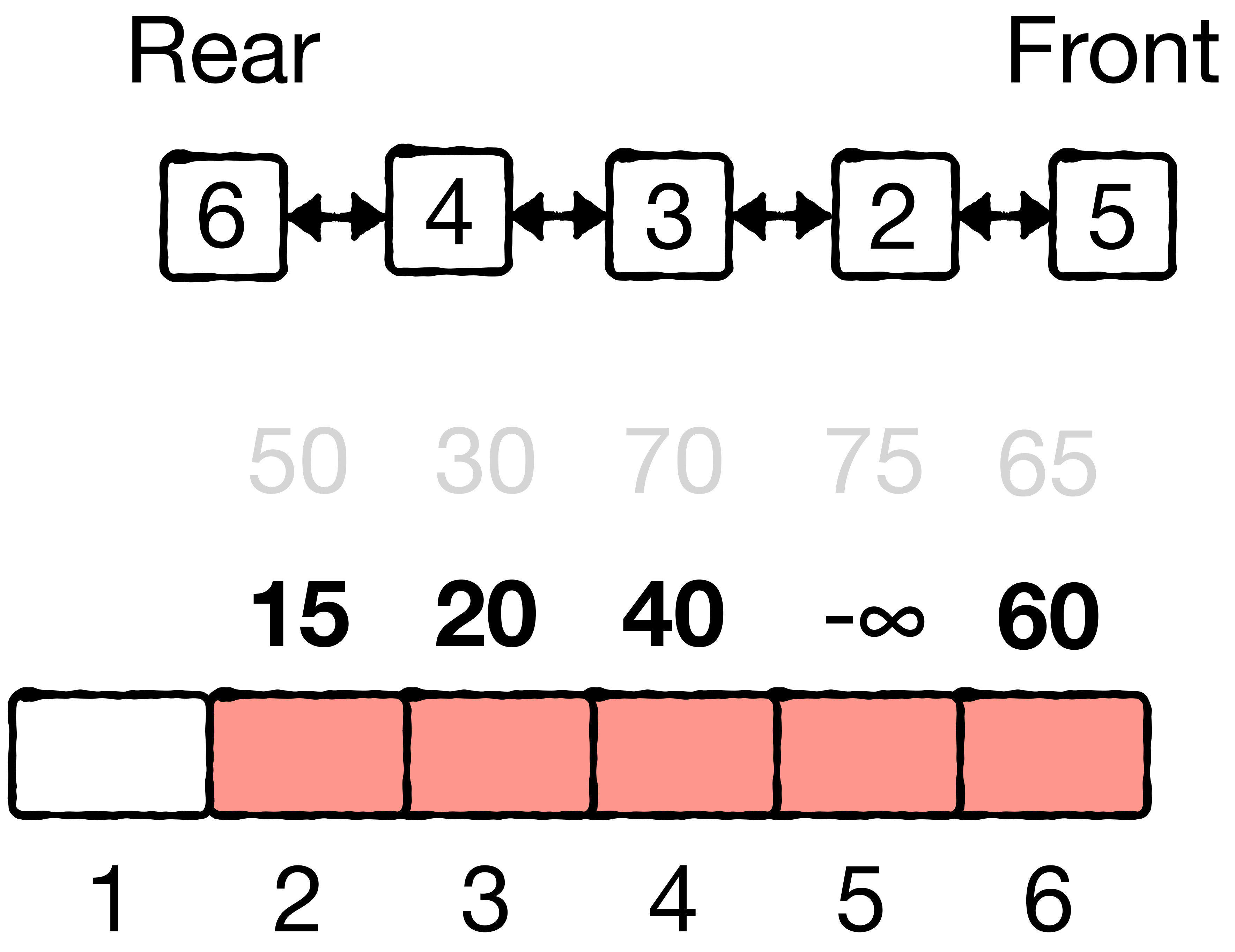
	1	2	3	4	5	6
Last access		50	30	70	75	65
2nd last access		15	20	40	-∞	60
	⬜	🟥	🟥	🟥	🟥	🟥

How to find 2nd LRU entry?

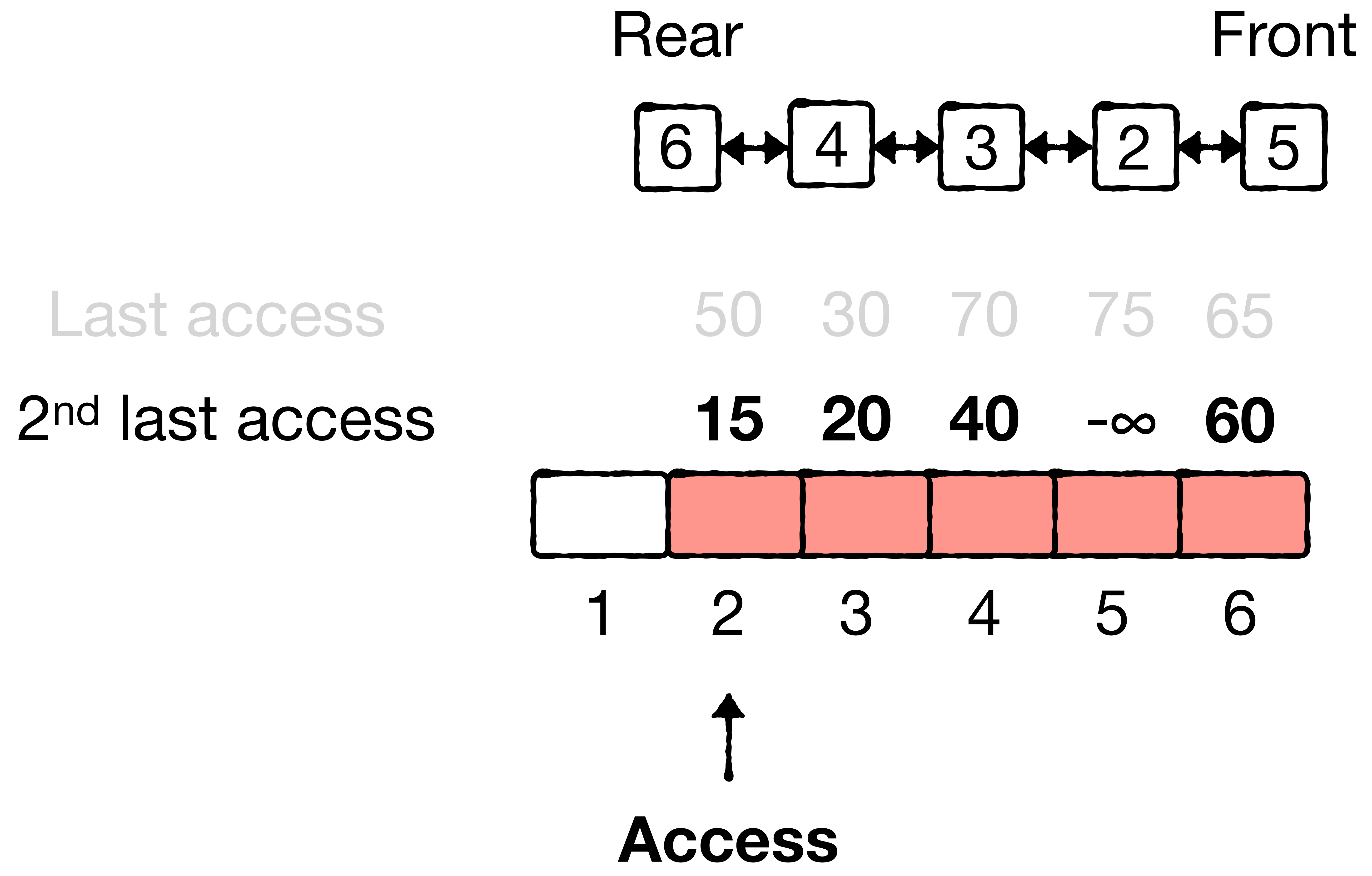


How to find 2nd LRU entry? **Consider linked list**

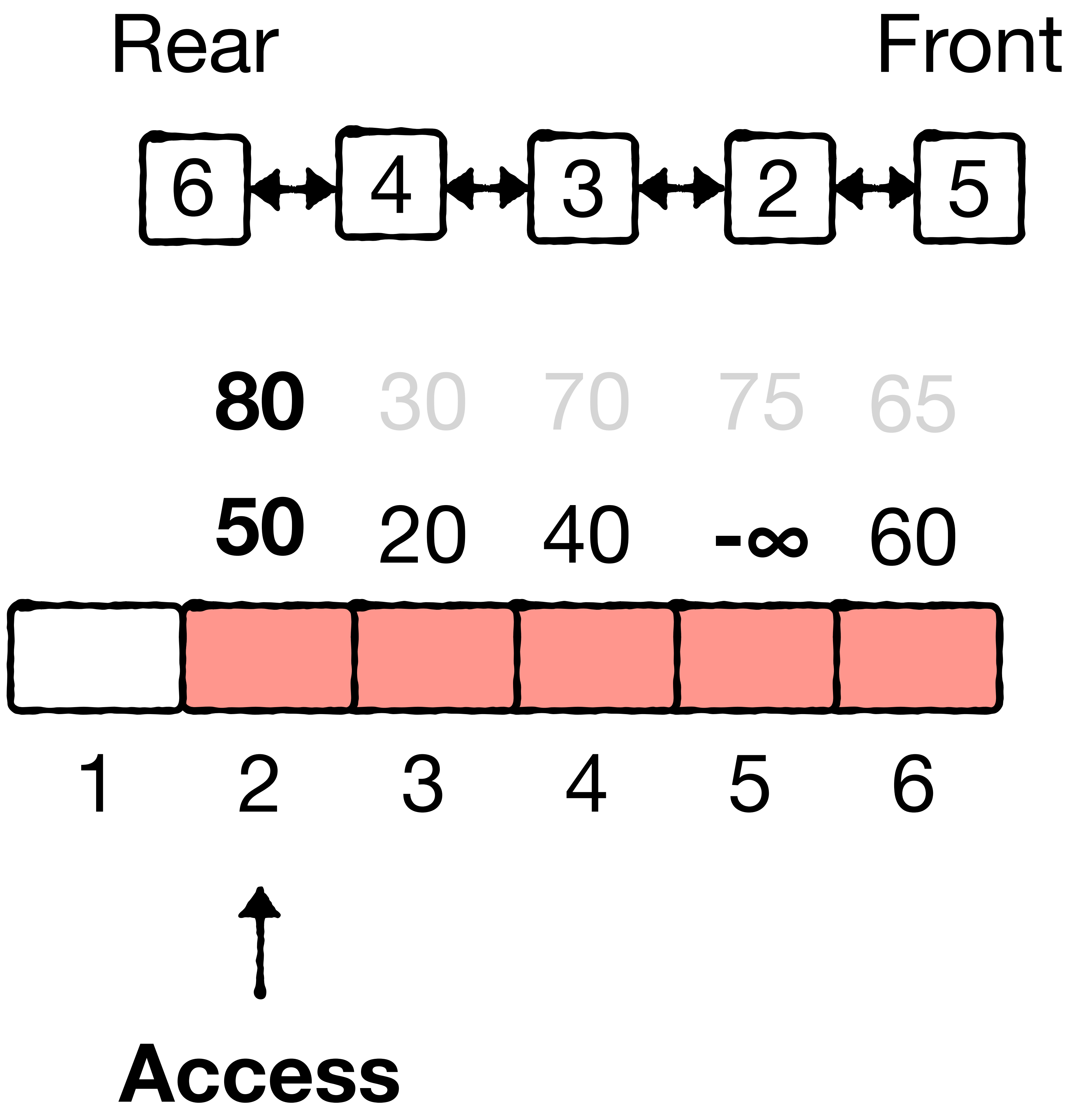
(Ordered by 2nd last access)



How to find 2nd LRU entry? Consider linked list

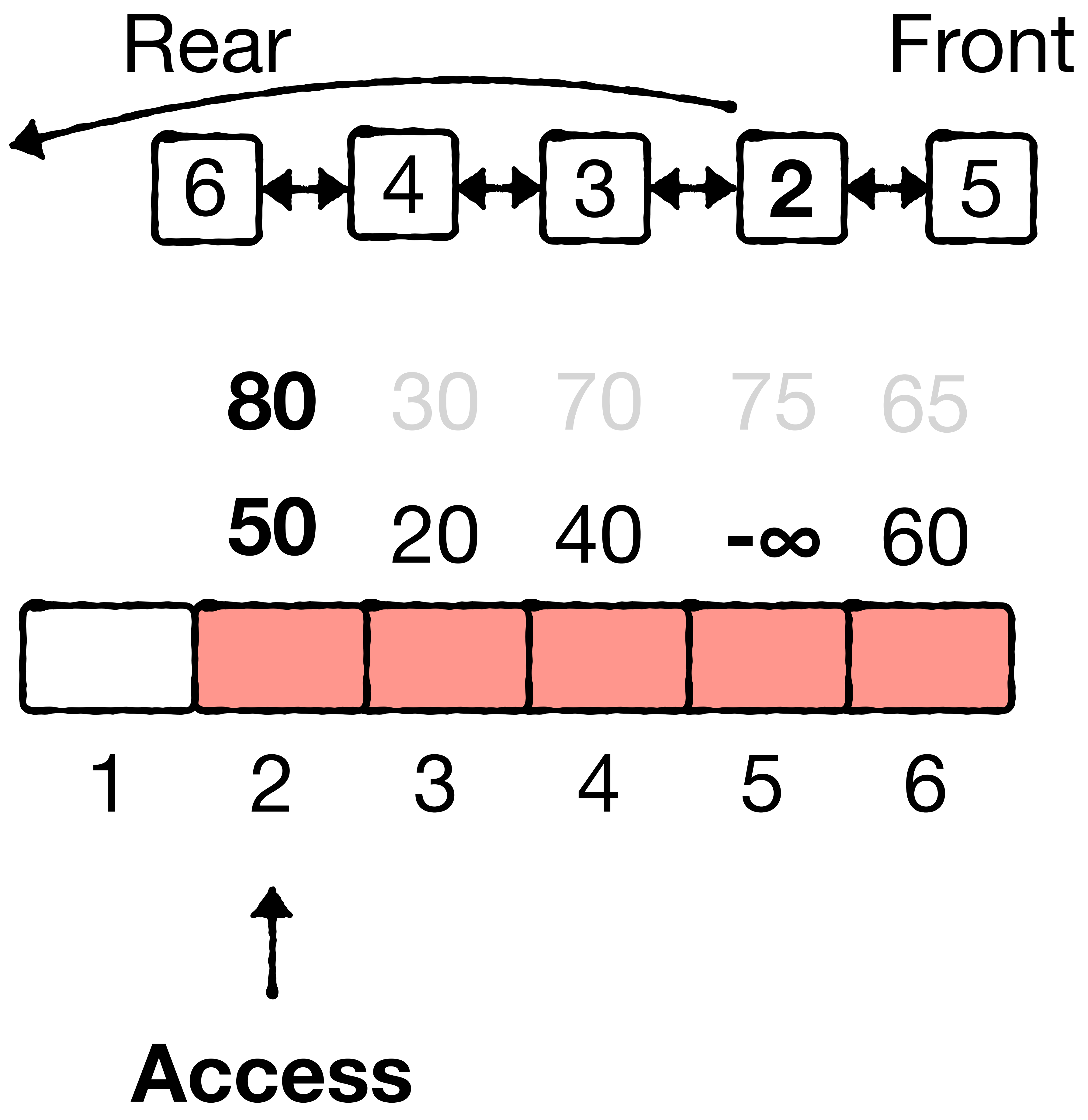


How to find 2nd LRU entry?　　Consider linked list



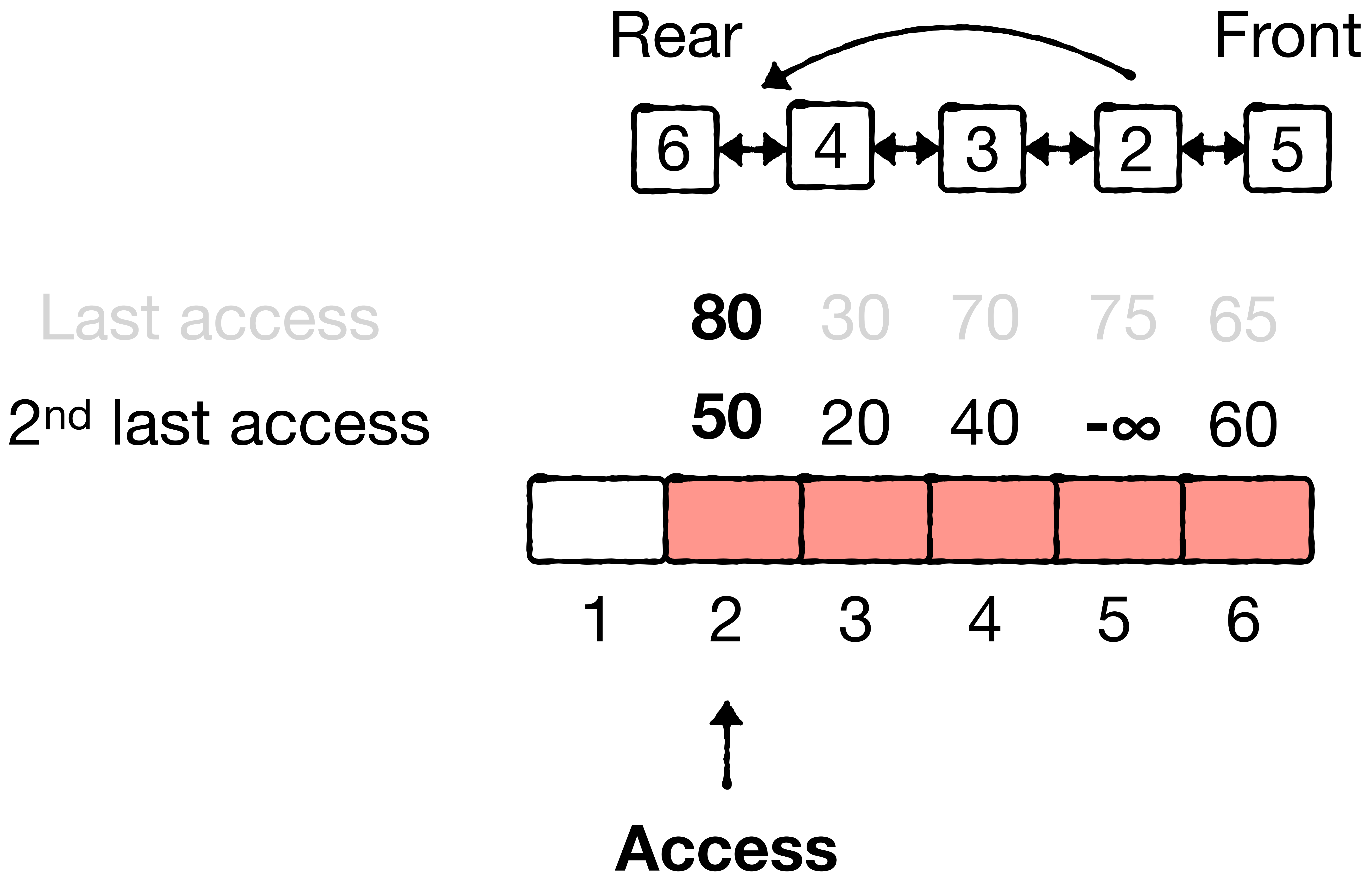
Time now:
80

Recall that in regular LRU queue, accessed item always moves to rear



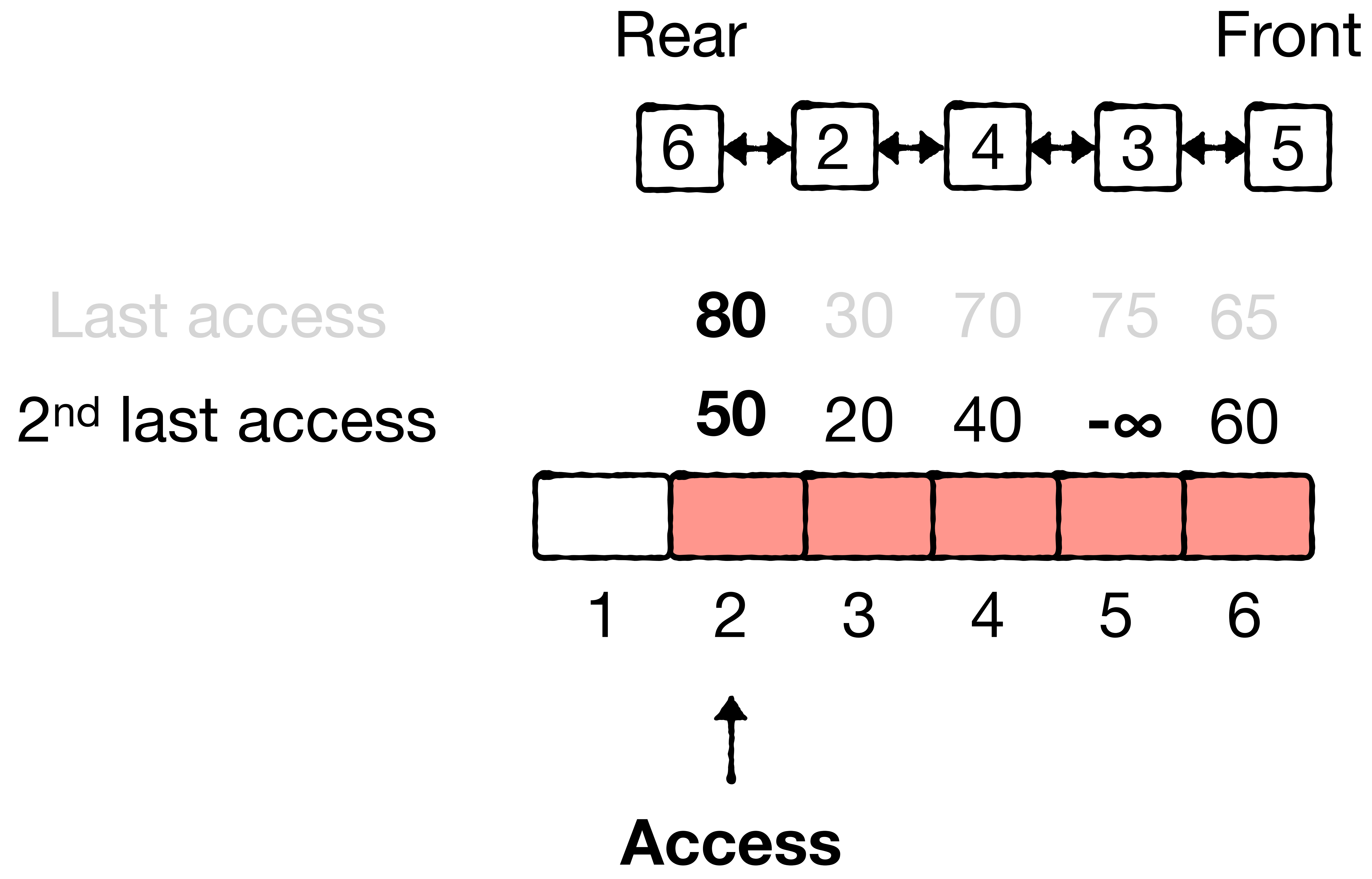
Time now:
80

Recall that in regular LRU queue, accessed item always moves to rear



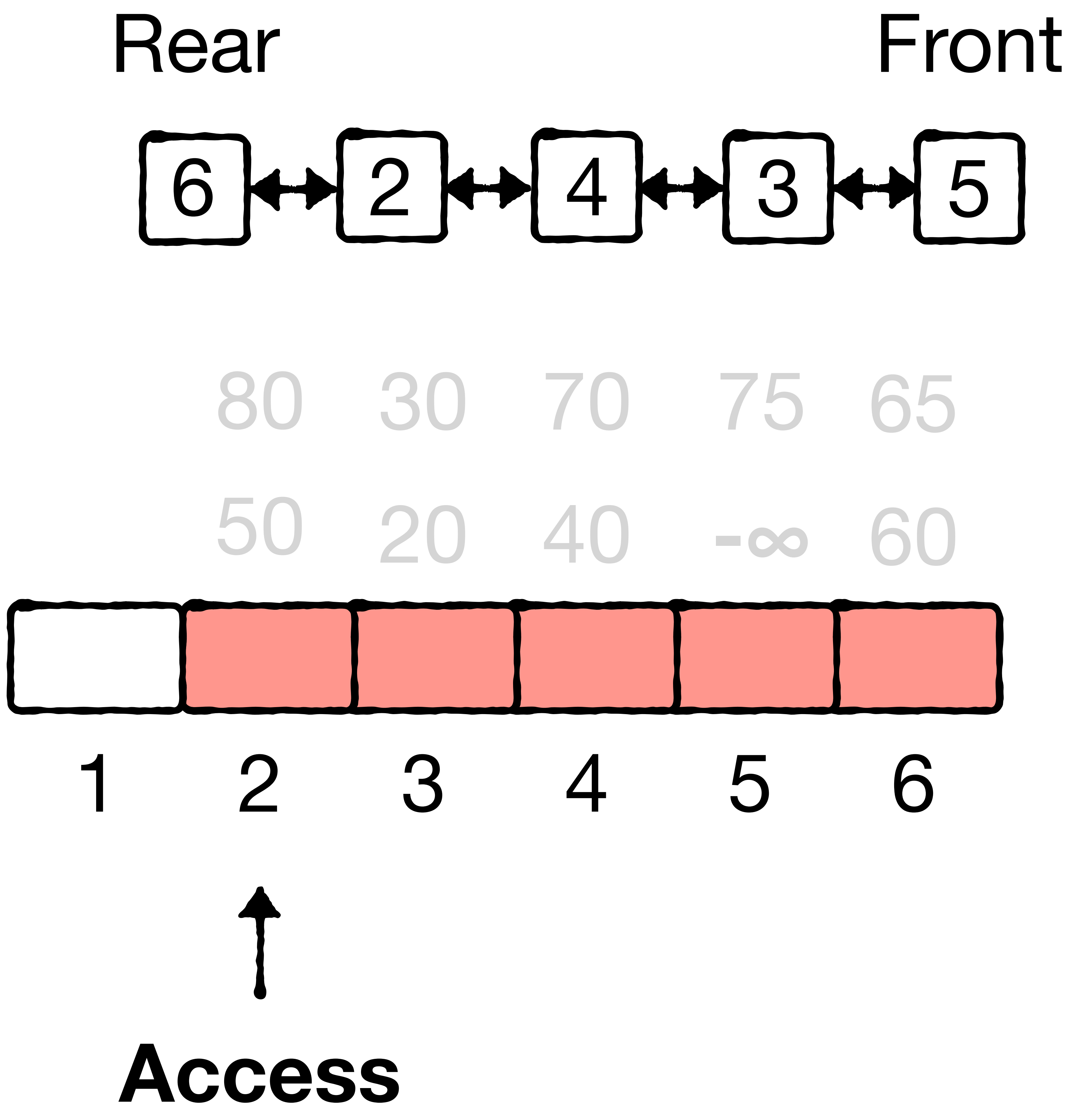
But slot 2 does not necessarily move to rear. Finding its right location requires linear pass in O(N)

Recall that in regular LRU queue, accessed item always moves to rear



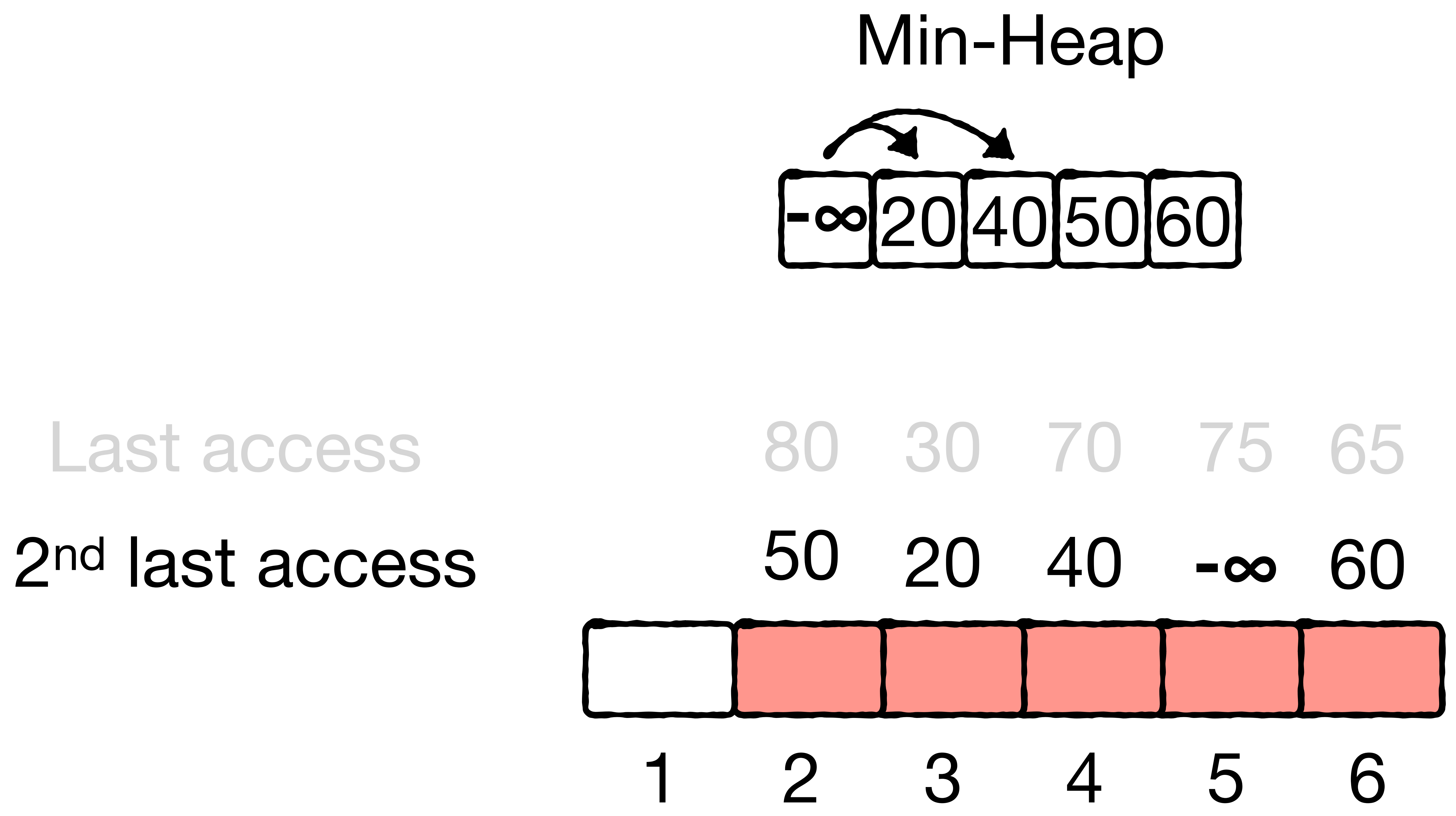
But slot 2 does not necessarily move to rear. Finding its right location requires linear pass in O(N)

Can we improve on O(N) for each access?

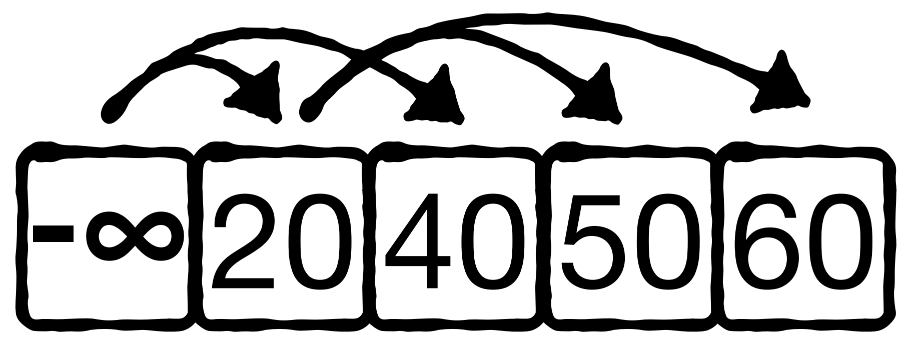


Min-Heap

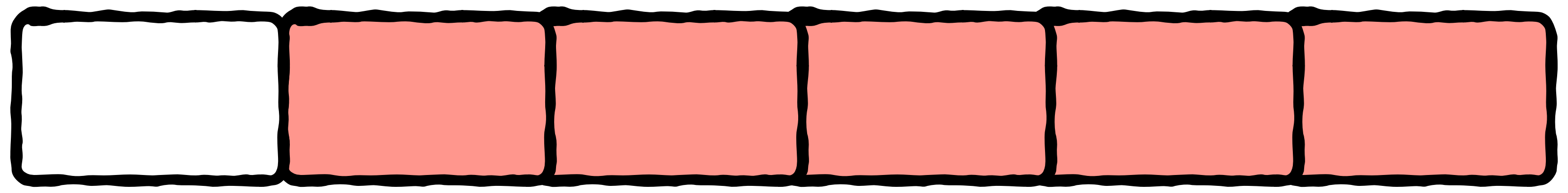
Last access		80	30	70	75	65
2nd last access		50	20	40	$-\infty$	60
	1	2	3	4	5	6

Min-Heap
-∞ 20 40 50 60
Last access
80 30 70 75 65
2nd last access
50 20 40 -∞ 60
1 2 3 4 5 6

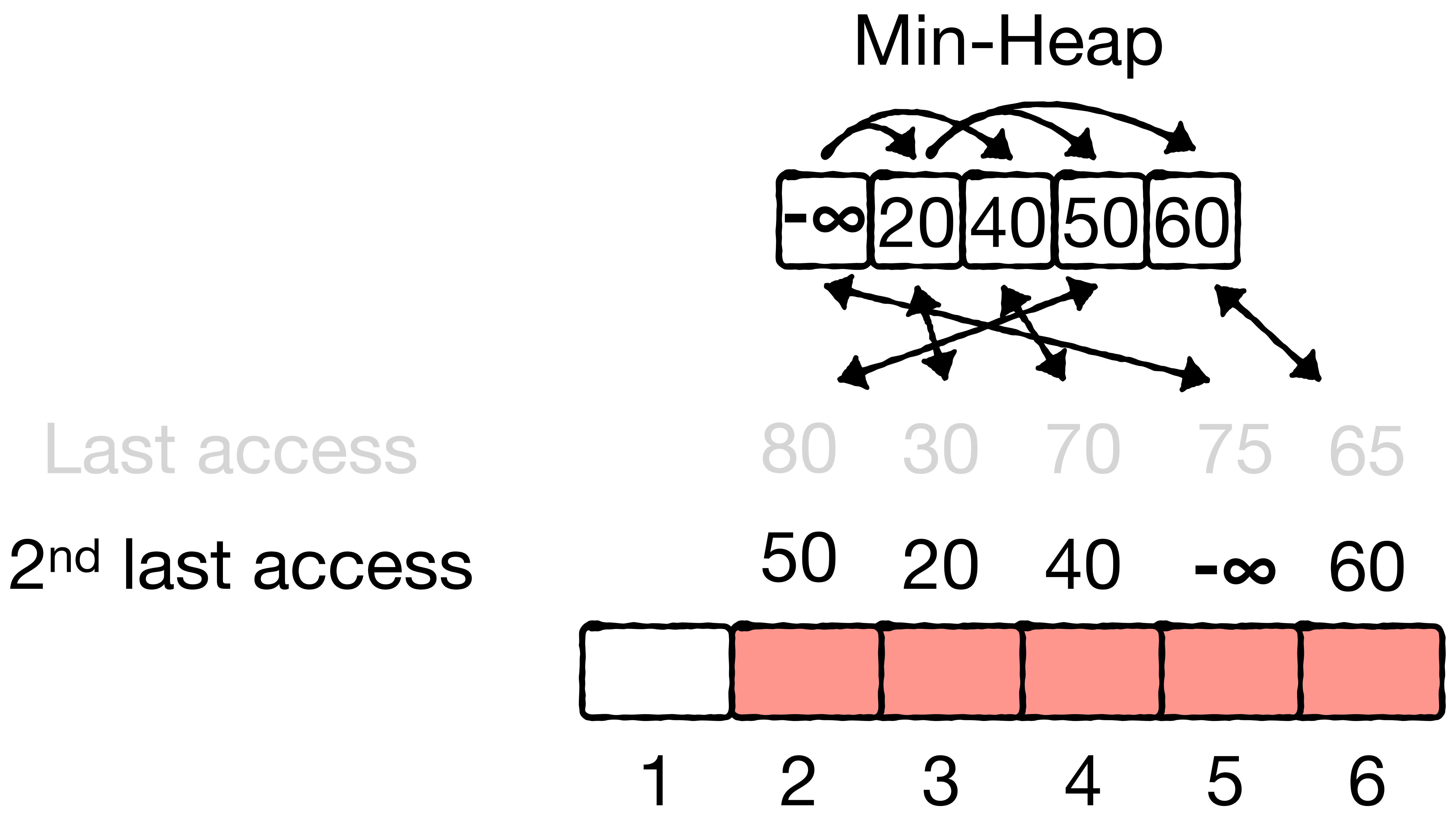
Min-Heap

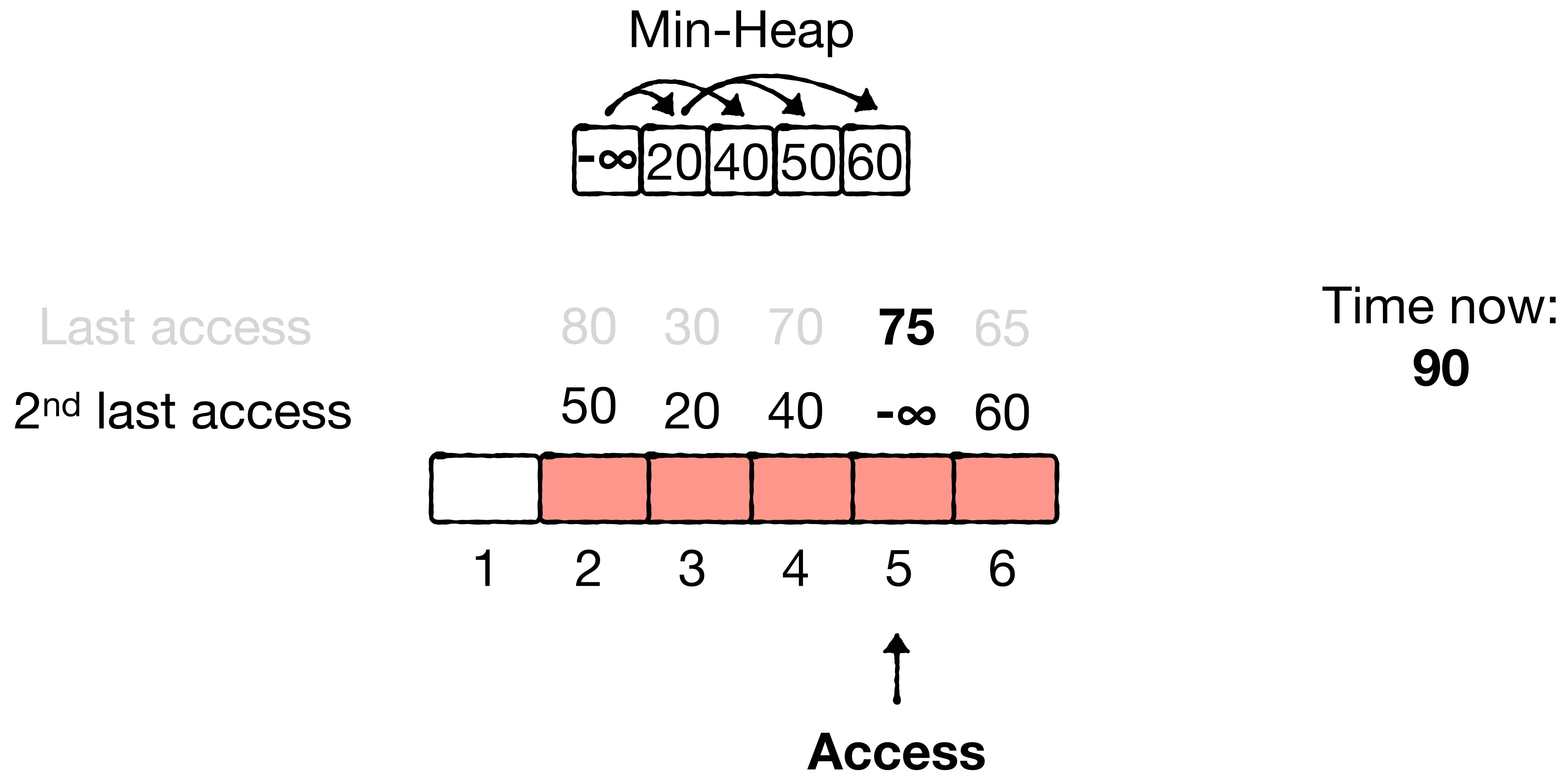


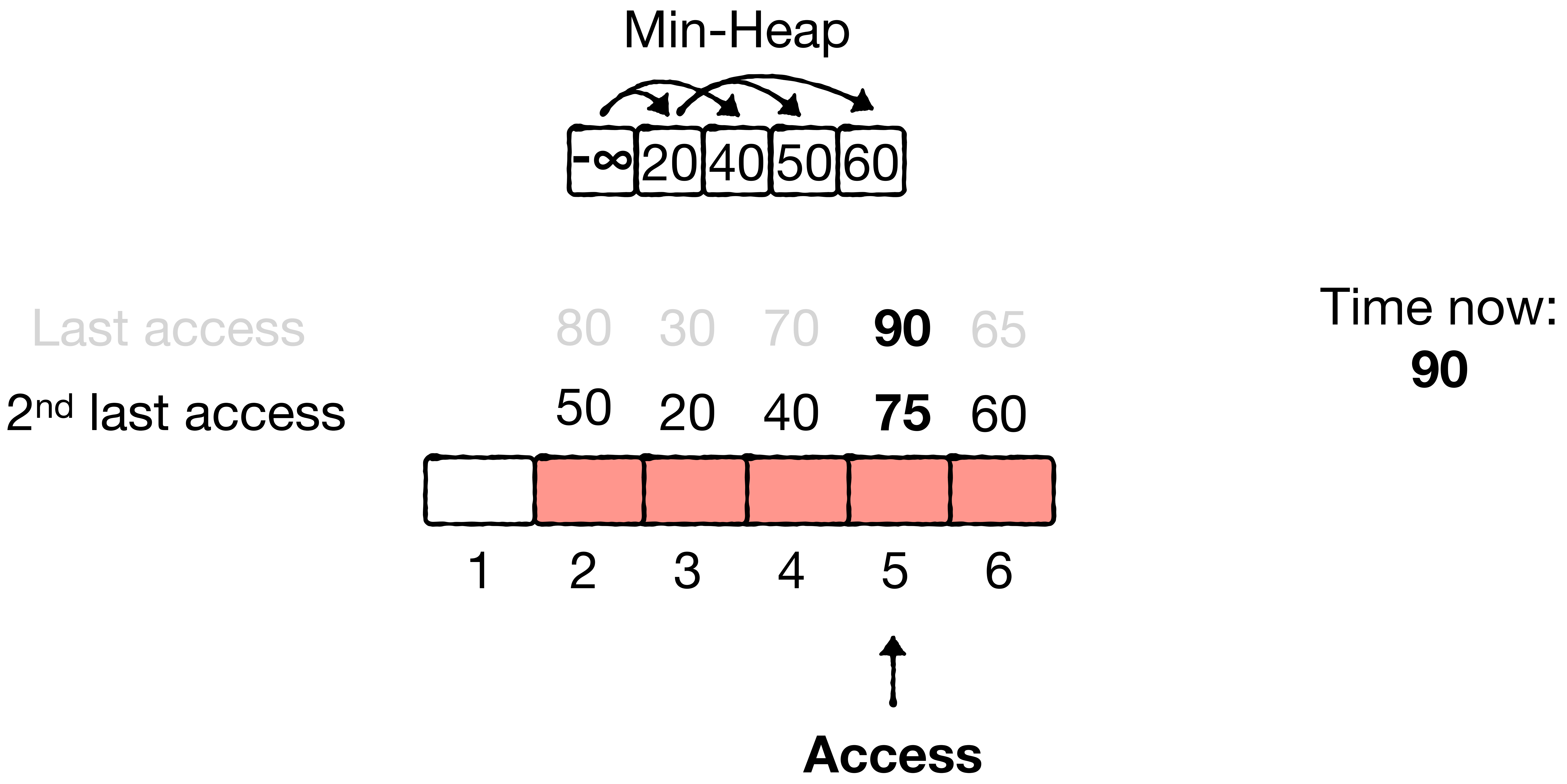
Last access		80	30	70	75	65
2nd last access		50	20	40	-∞	60
	1	2	3	4	5	6



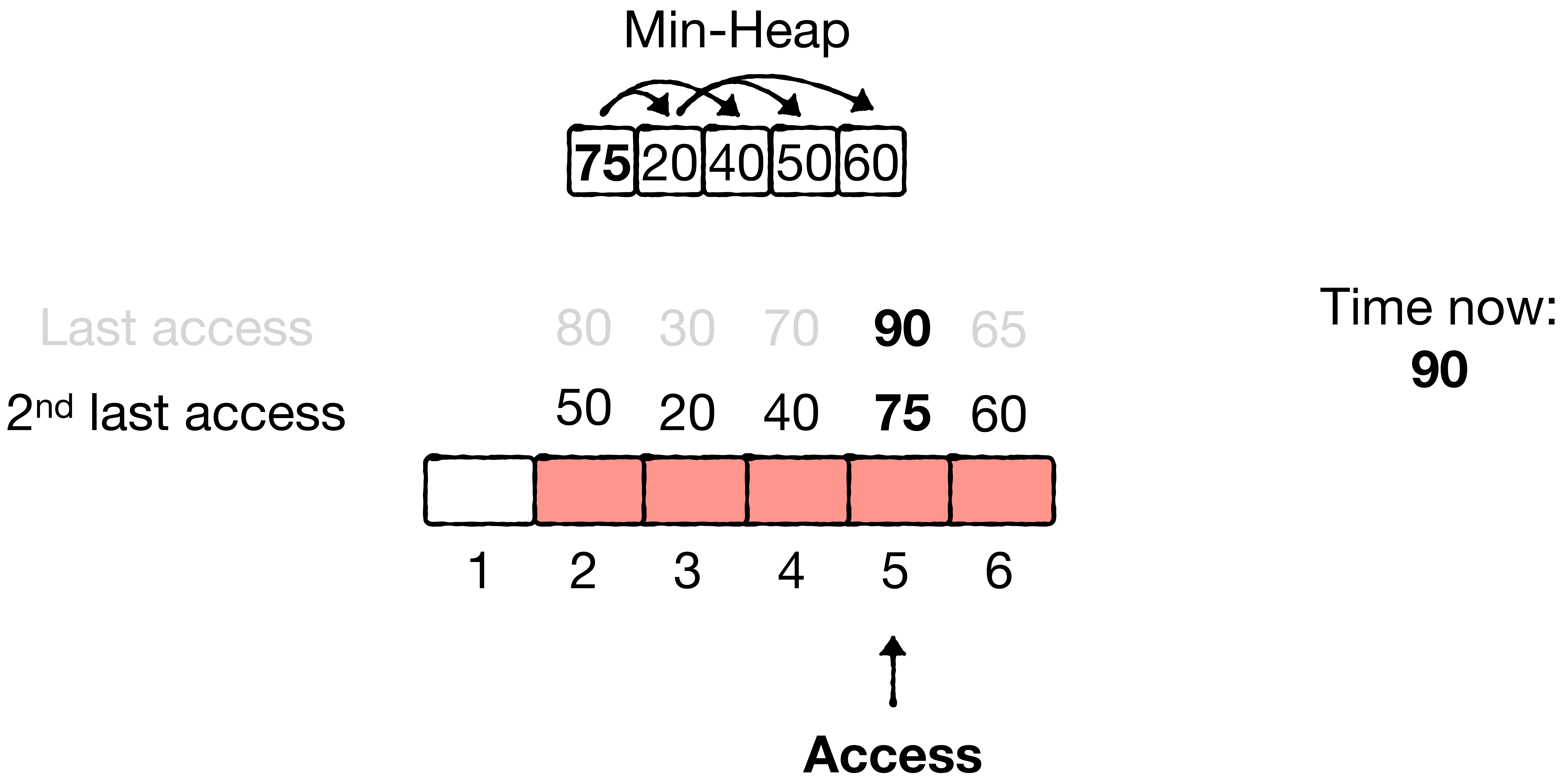
+ doubly-directed pointers



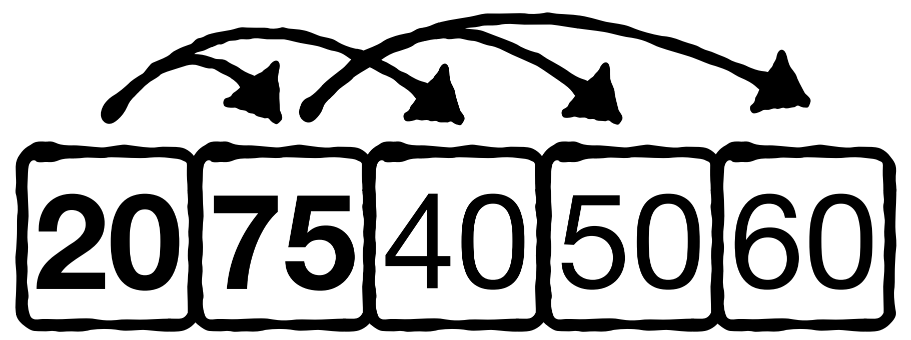

Min-Heap
-∞
20
40
50
60
Last access
80
30
70
75
65
2nd last access
50
20
40
-∞
60
1
2
3
4
5
6
Access
Time now:
90



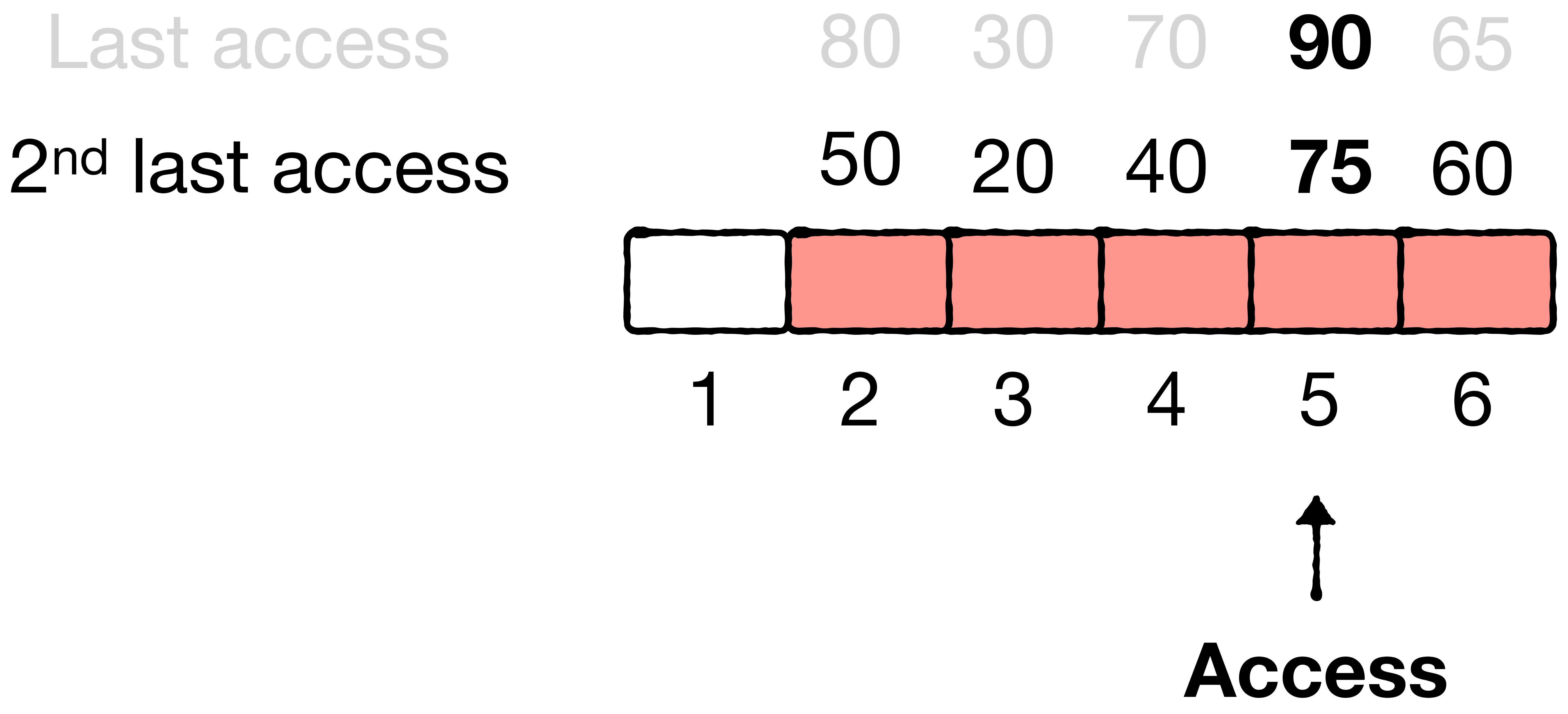
Min-Heap
-∞
20
40
50
60
Last access
80
30
70
90
65
2nd last access
50
20
40
75
60
1
2
3
4
5
6
Access
Time now:
90


Min-Heap
75
20
40
50
60
Last access
80
30
70
90
65
2nd last access
50
20
40
75
60
1
2
3
4
5
6
Access
Time now:
90

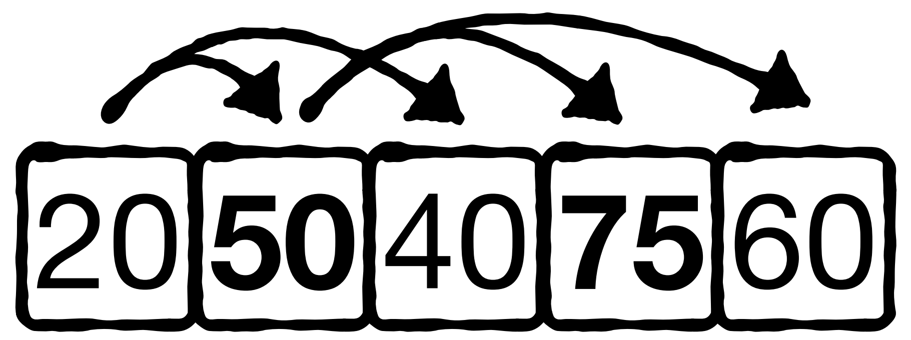
Min-Heap

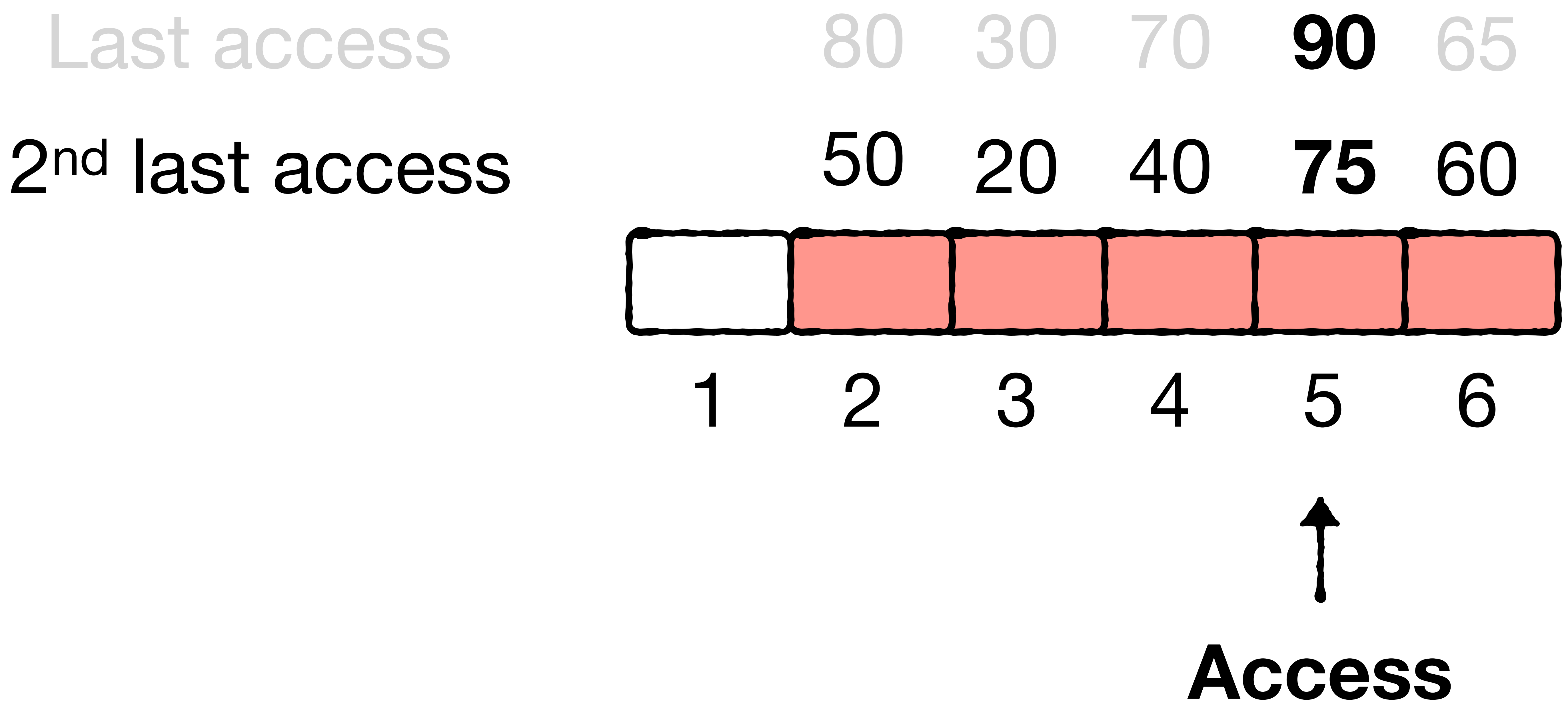


Time now:
90

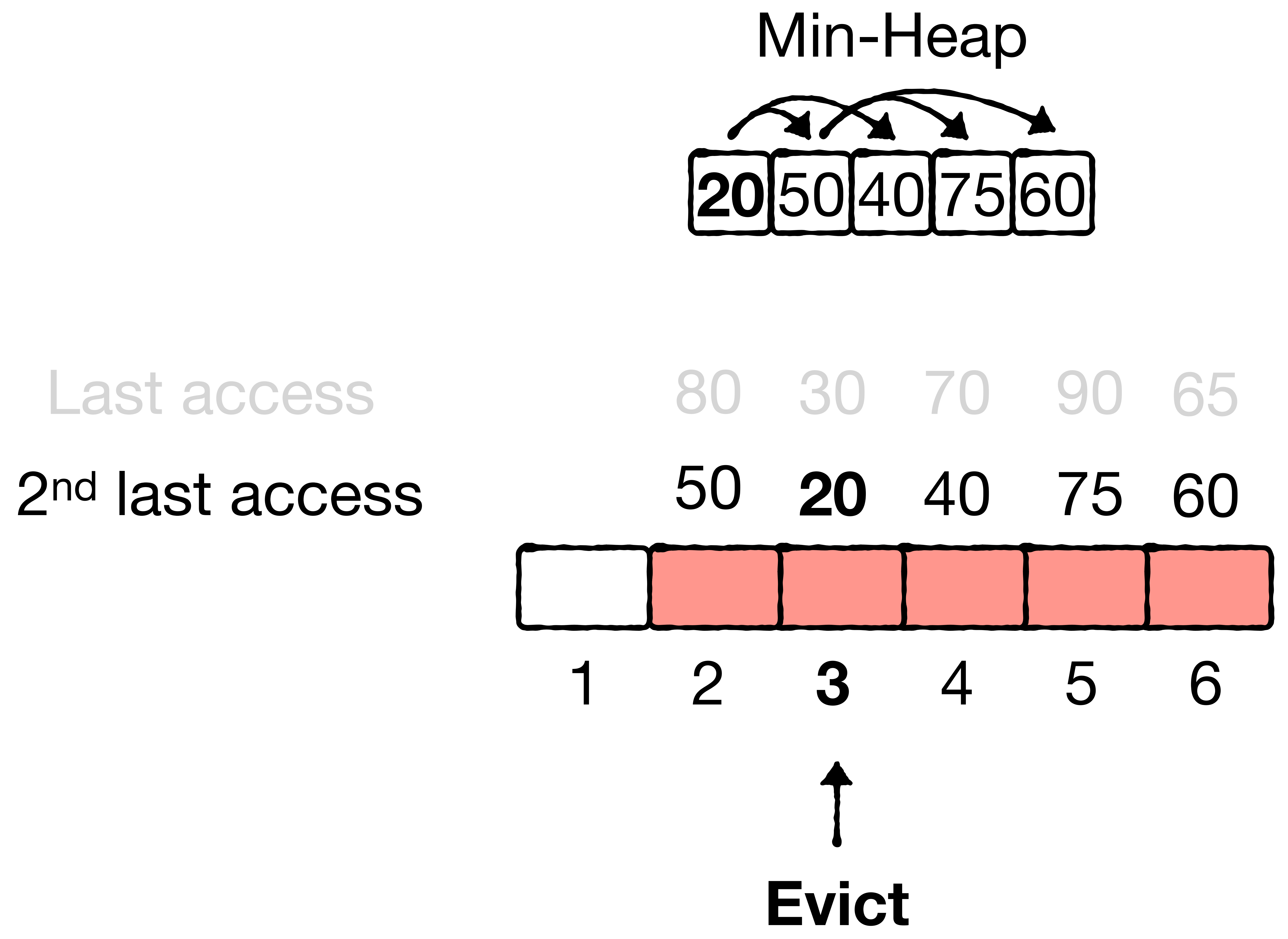


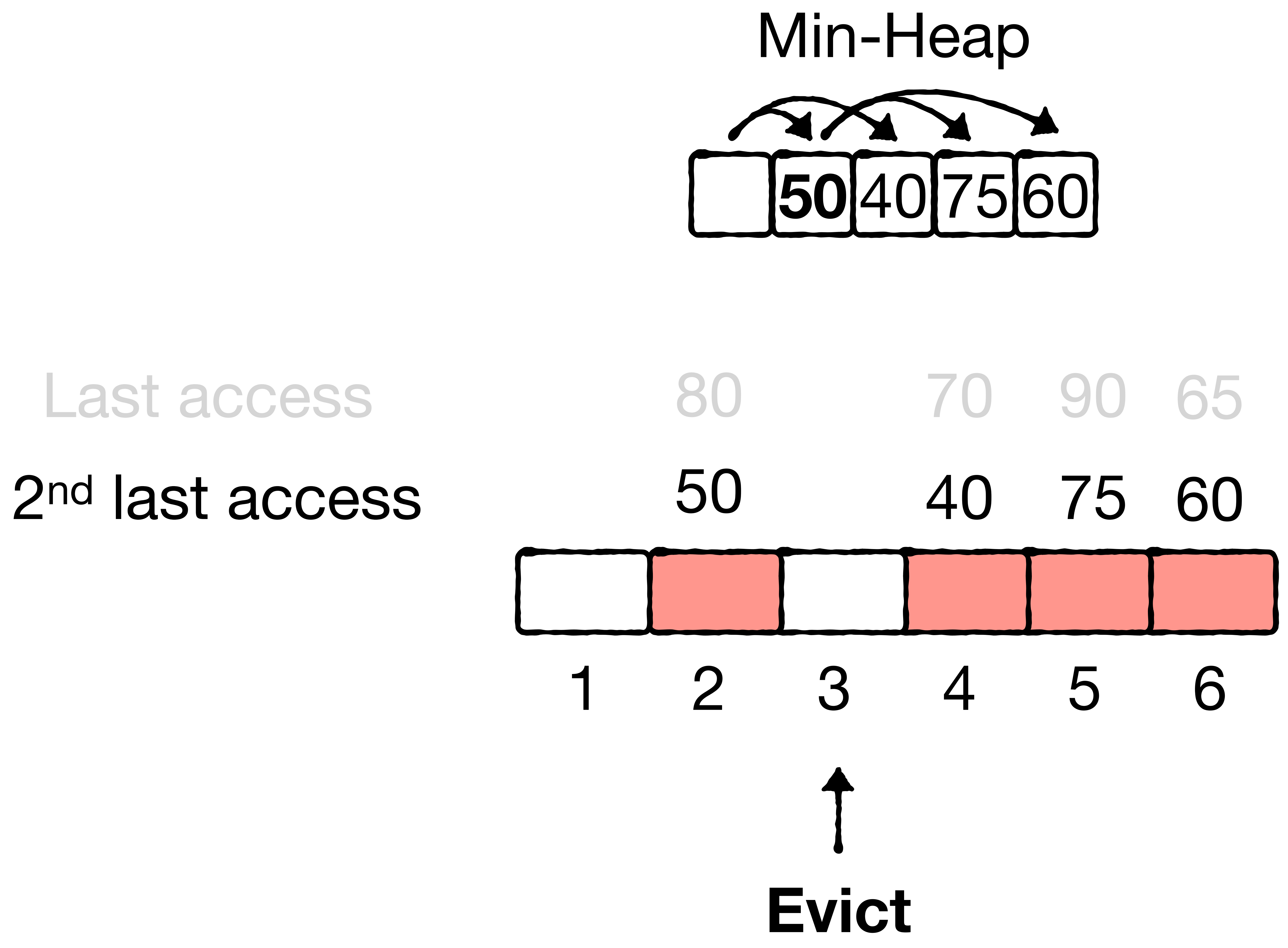
Min-Heap



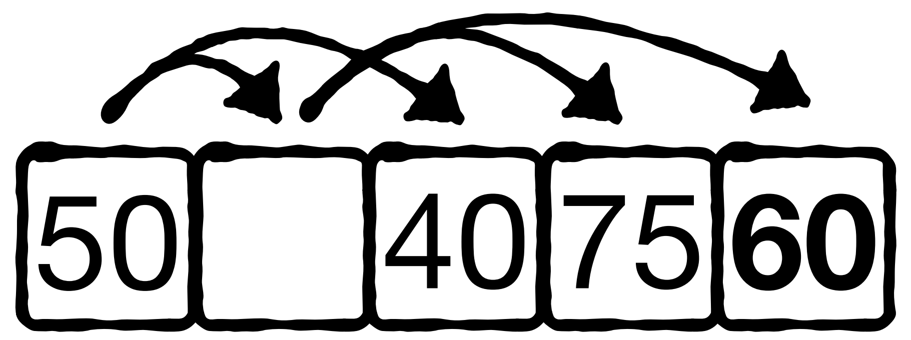


Time now:
90

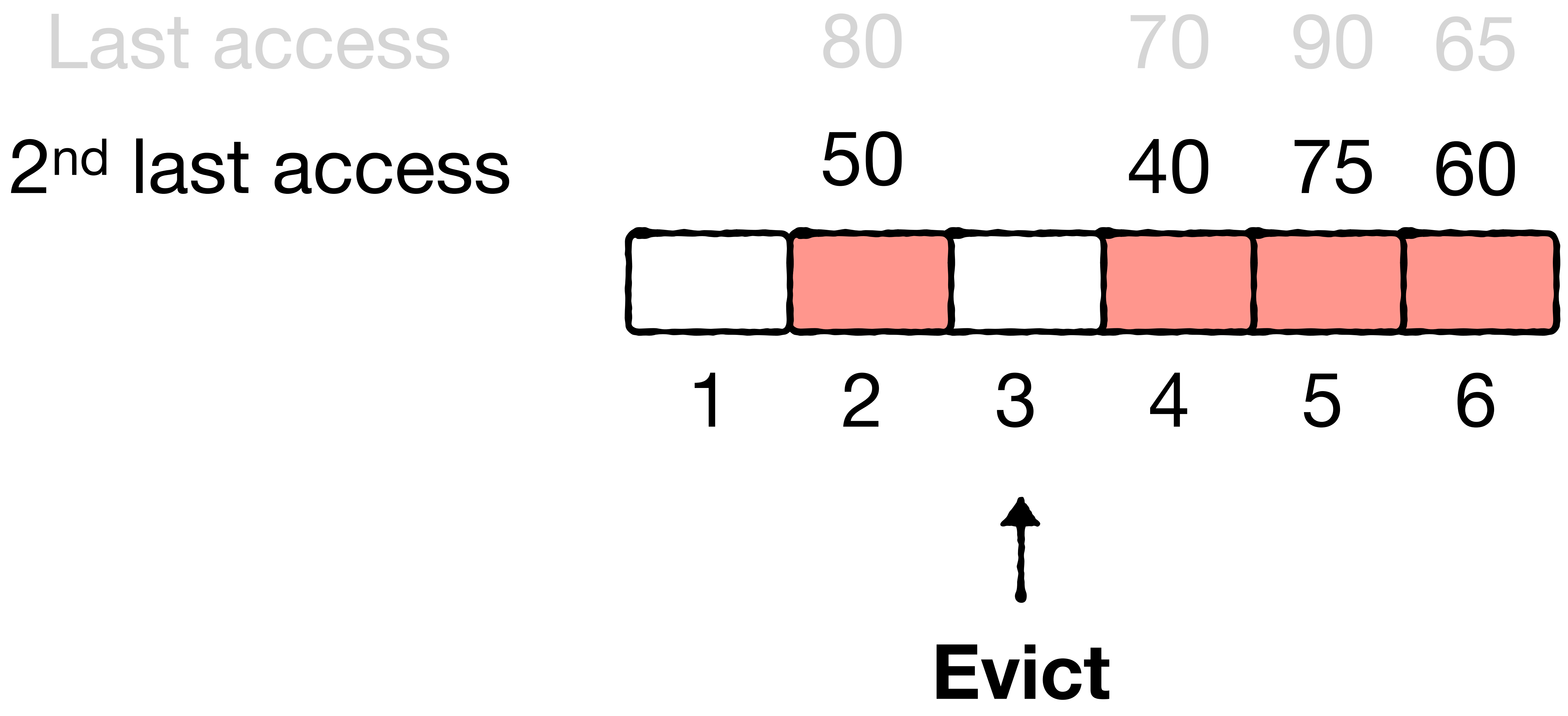

Min-Heap
20
50
40
75
60
Last access
80
30
70
90
65
2nd last access
50
20
40
75
60
1
2
3
4
5
6
Evict


Min-Heap
50
40
75
60
Last access
80
70
90
65
2nd last access
50
40
75
60
1
2
3
4
5
6
Evict

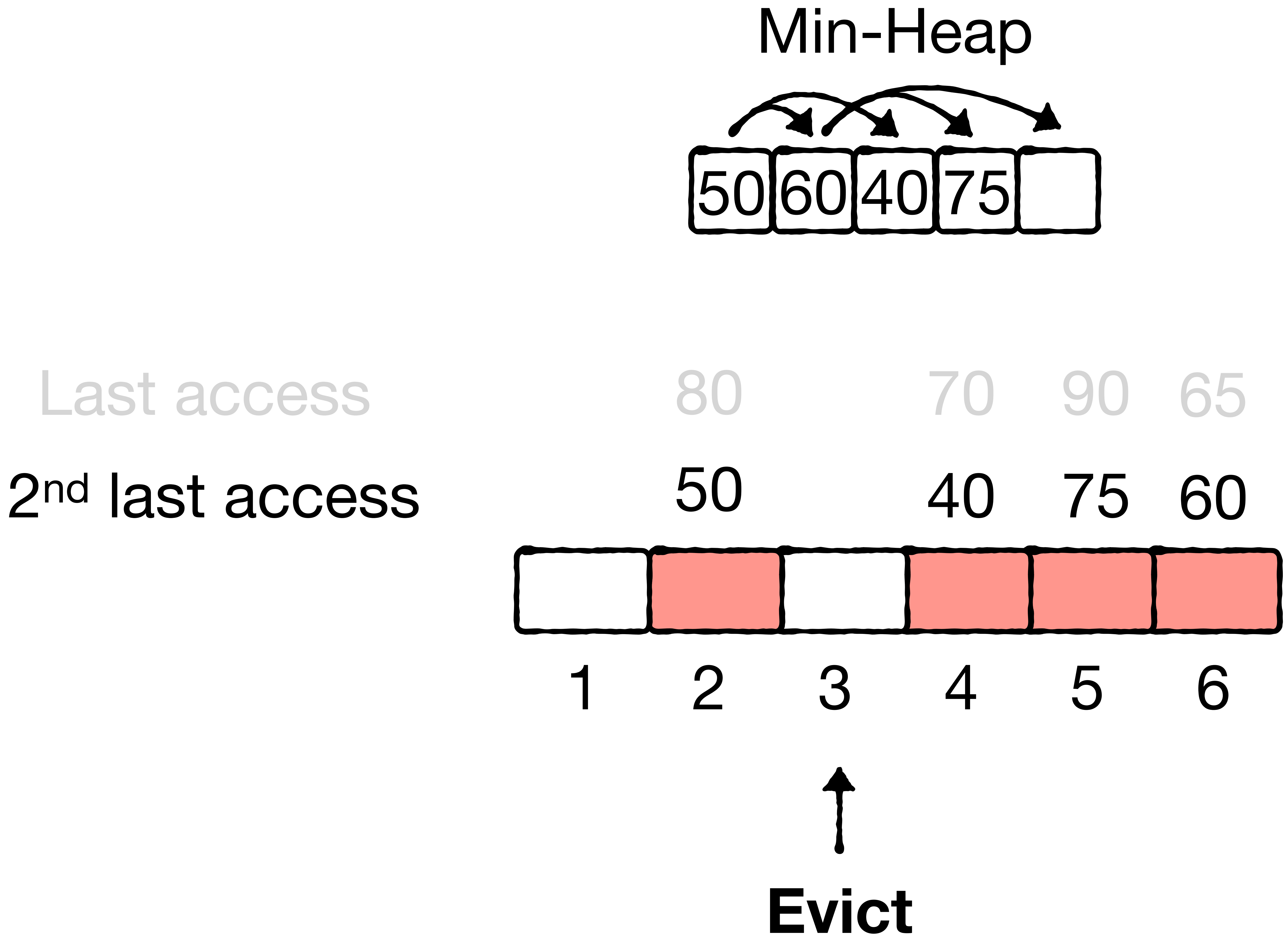
Min-Heap



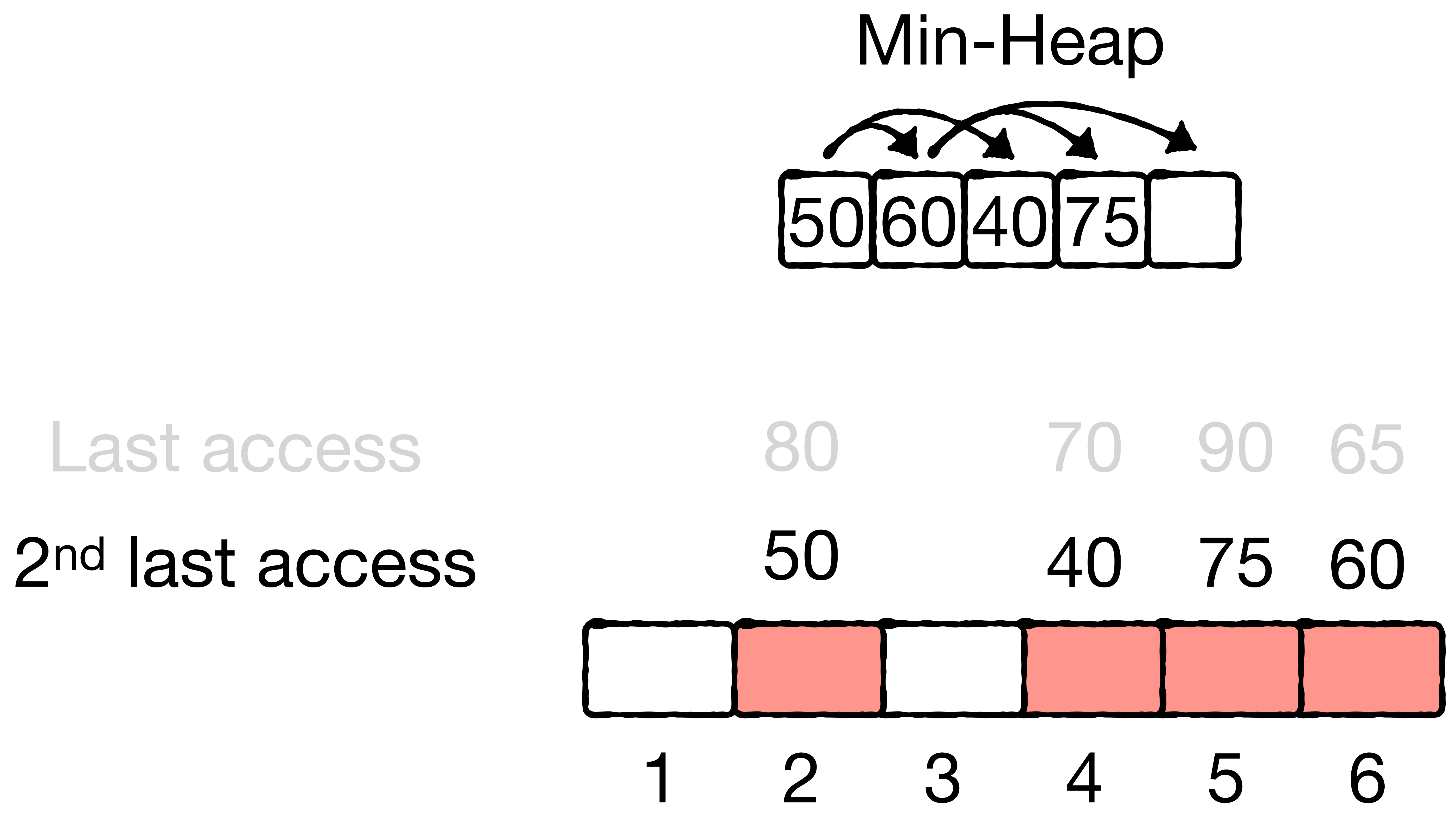
50
40
75
60



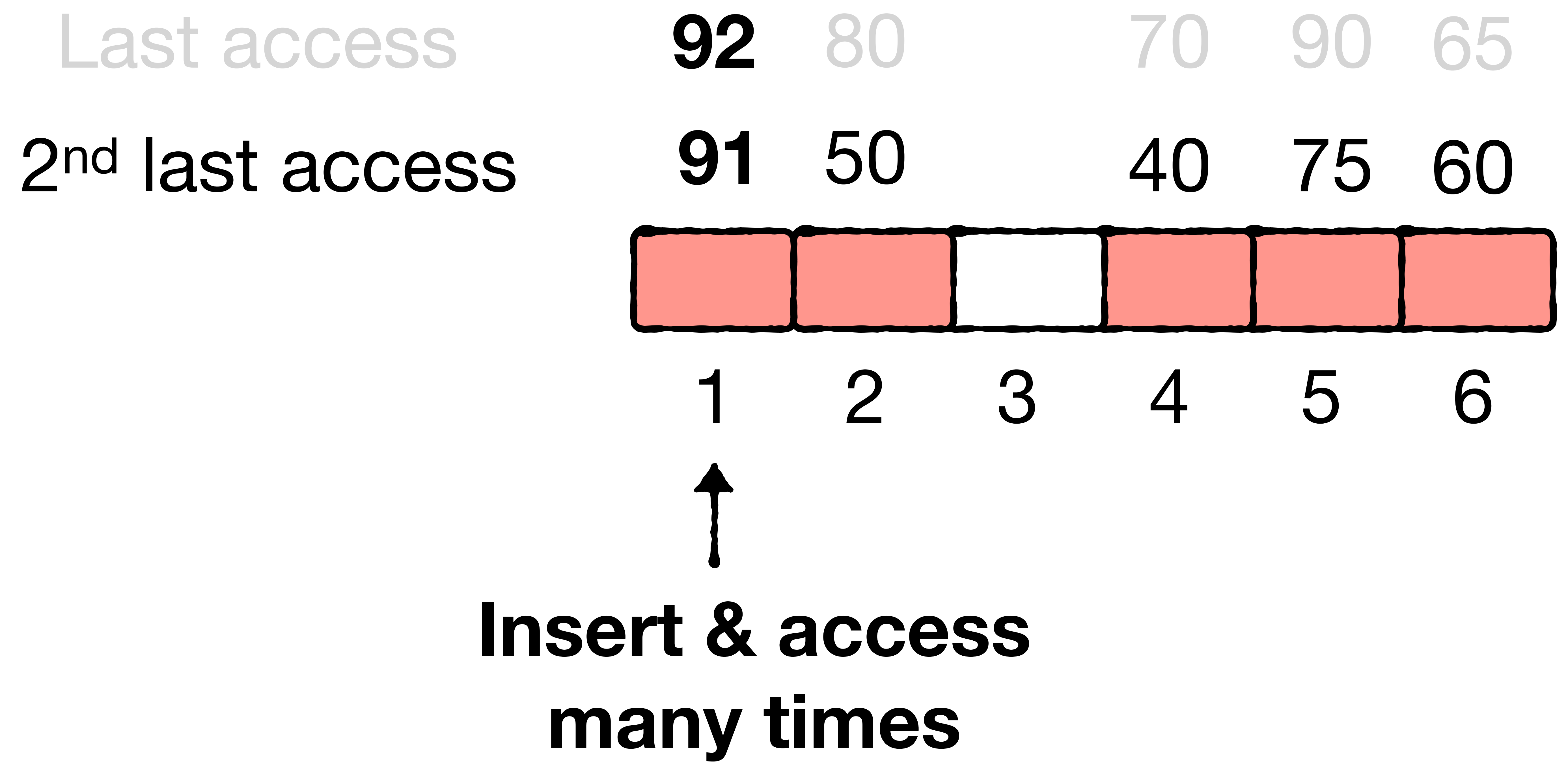
Last access
80
70
90
65
2nd last access
50
40
75
60
1
2
3
4
5
6
Evict


Min-Heap
50
60
40
75
Last access
80
70
90
65
2nd last access
50
40
75
60
1
2
3
4
5
6
Evict

Can access & evict in $O(\log_2 N)$ for any LRU/K

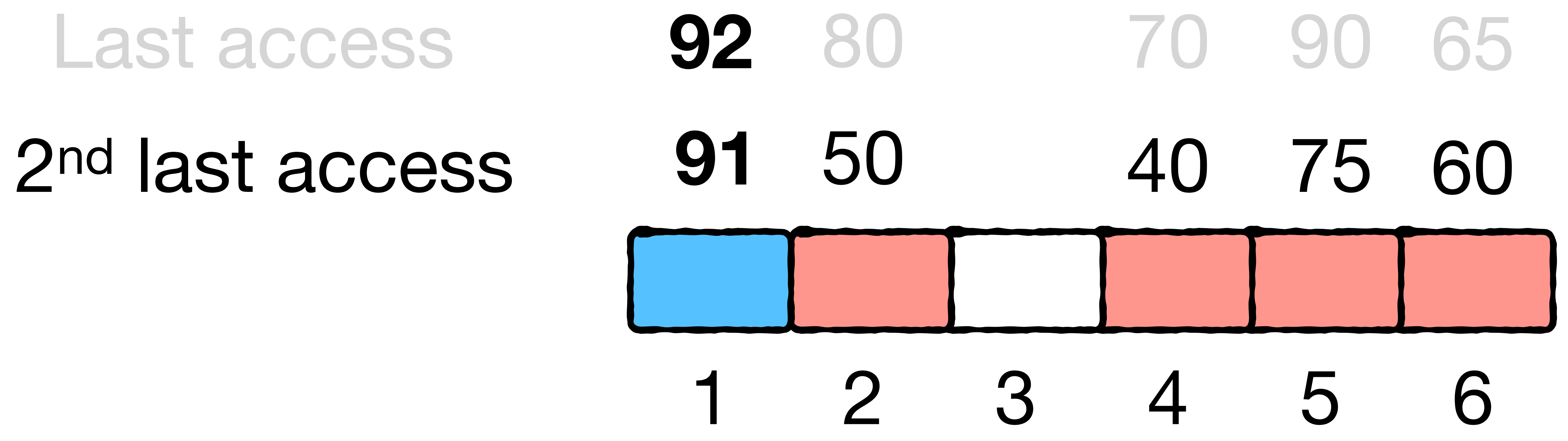


Concern 1: cold entry may get access spurt (e.g. by same transaction)



Concern 1: cold entry may get access spurt (e.g. by same transaction)

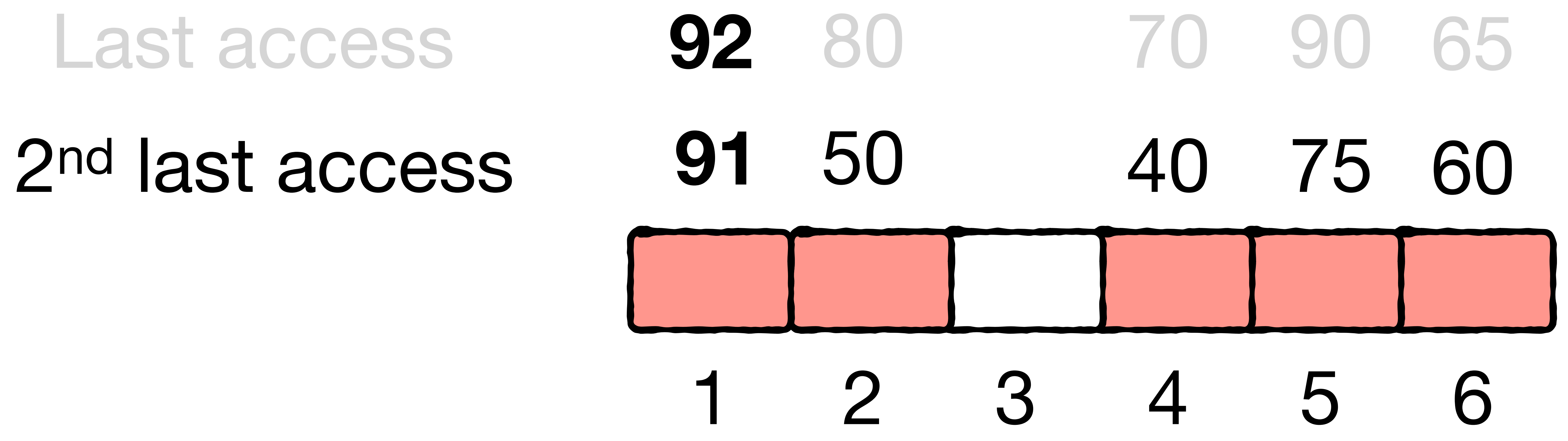
After transaction is over, page won't be evicted despite being cold



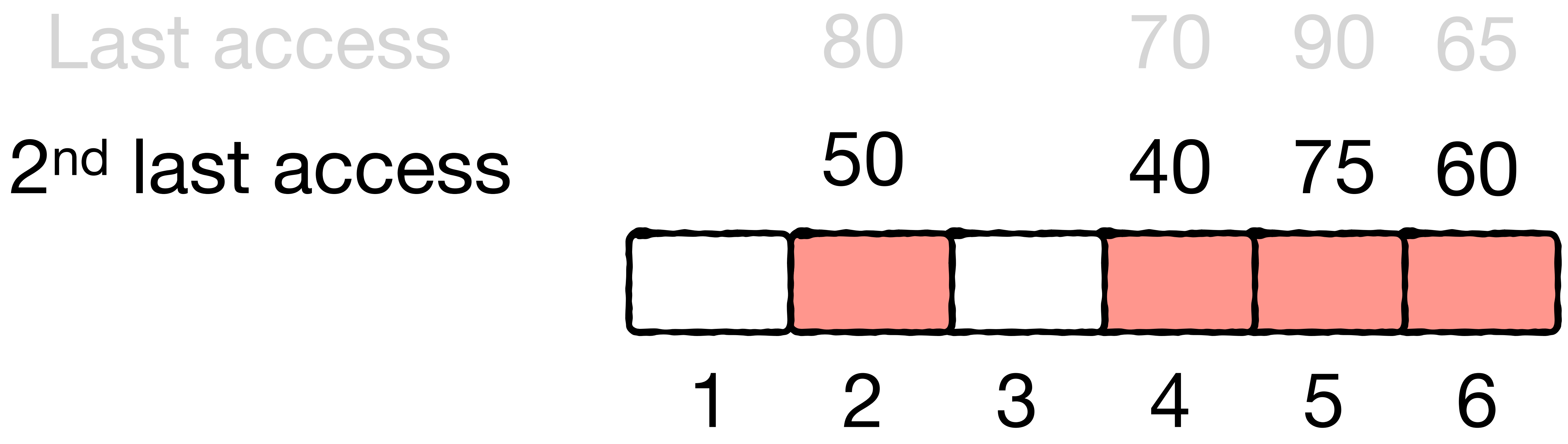
Concern 1: cold entry may get access spurt (e.g. by same transaction)

After transaction is over, page won't be evicted despite being cold

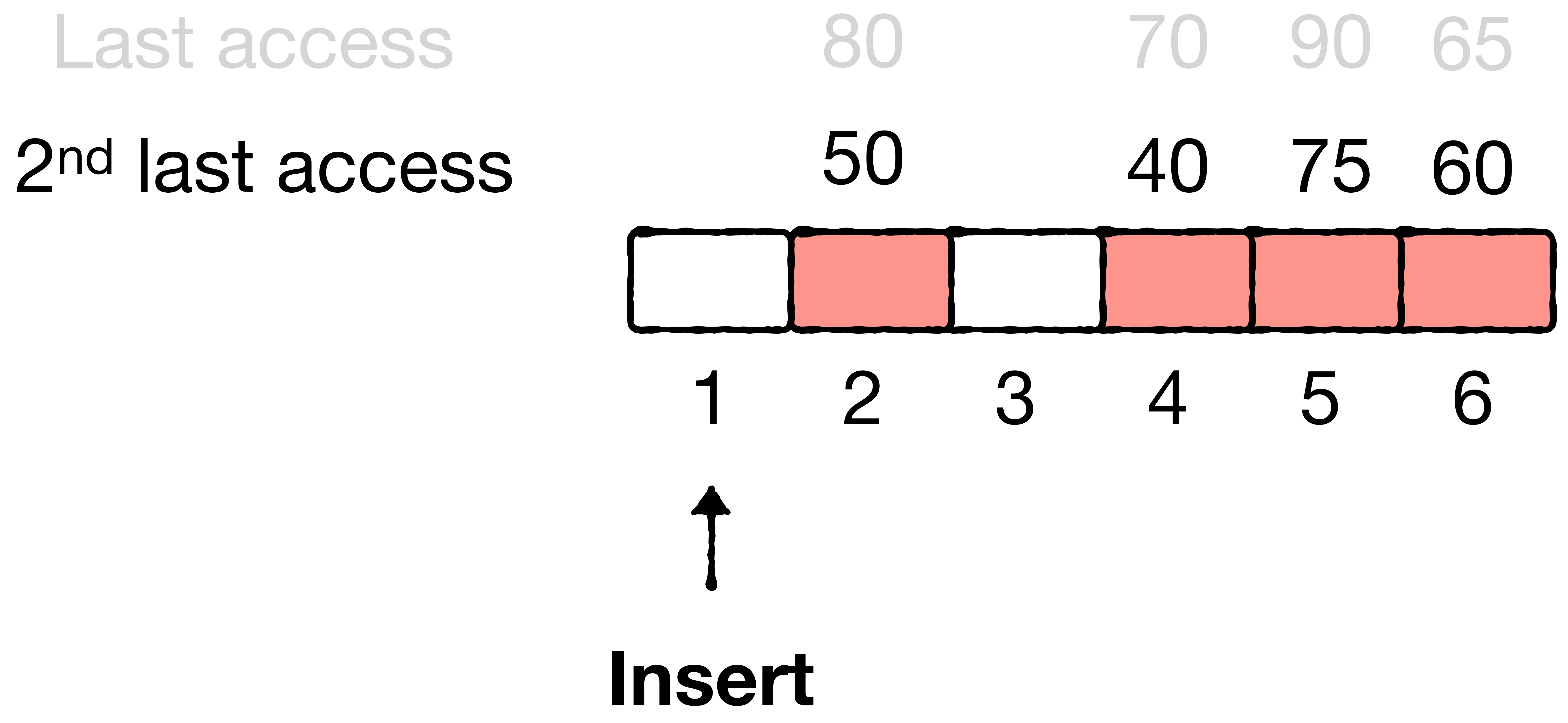
Solution?



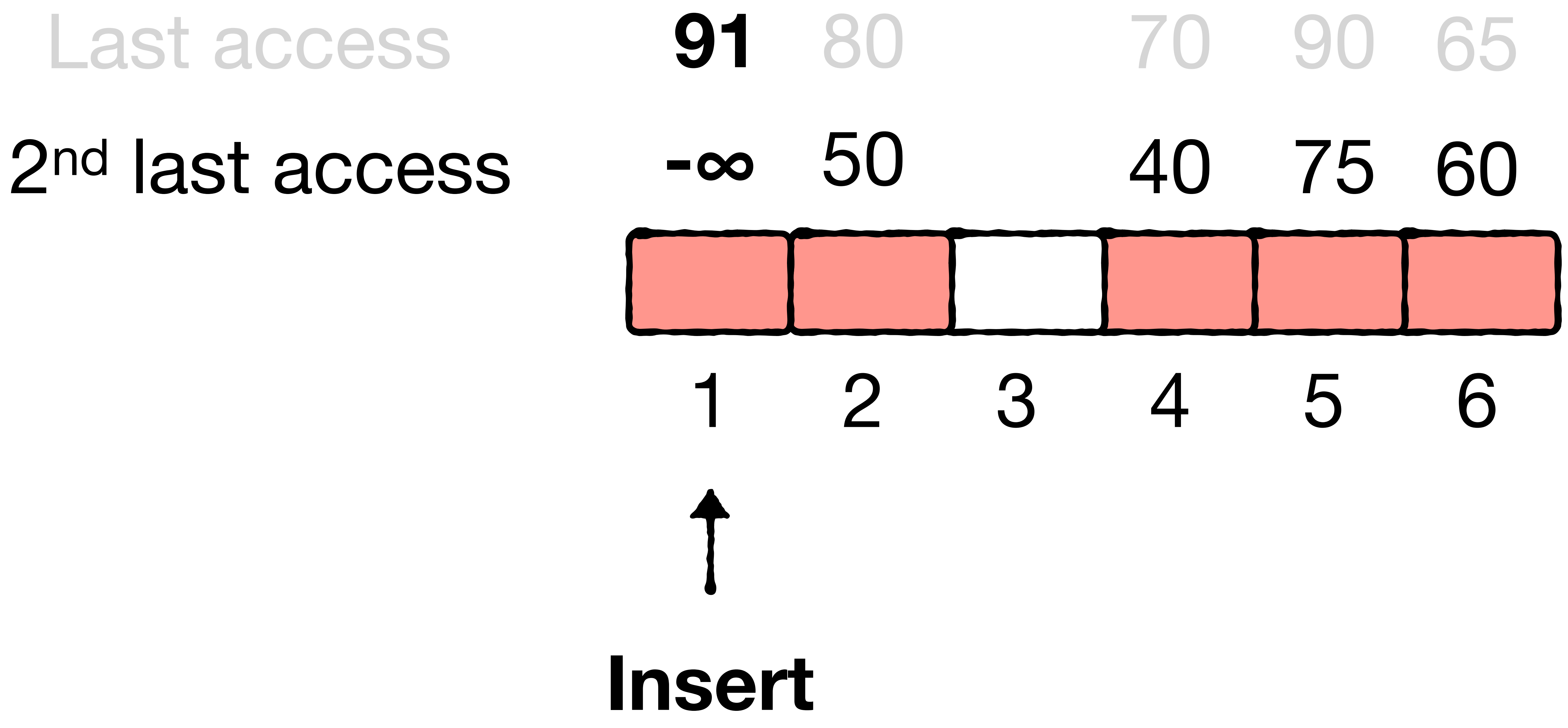
Solution: **do not update entry's access time if new access is within given window of last access**



Solution: do not update entry's access time if new access is within given window of last access

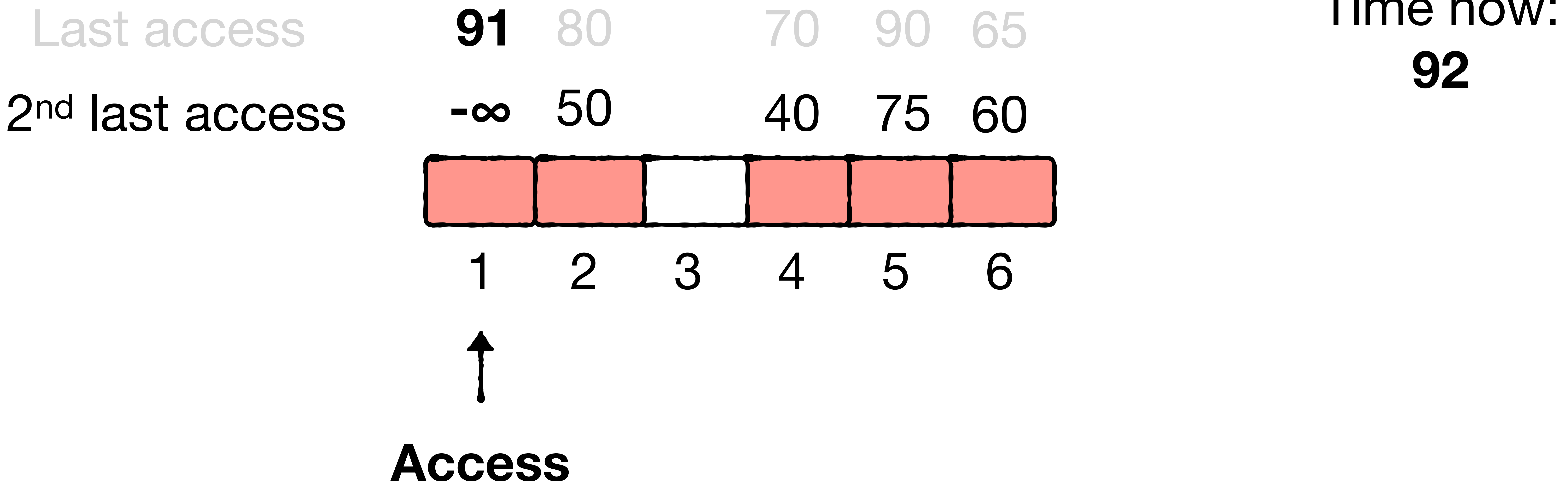


Solution: do not update entry's access time if new access is within given window of last access



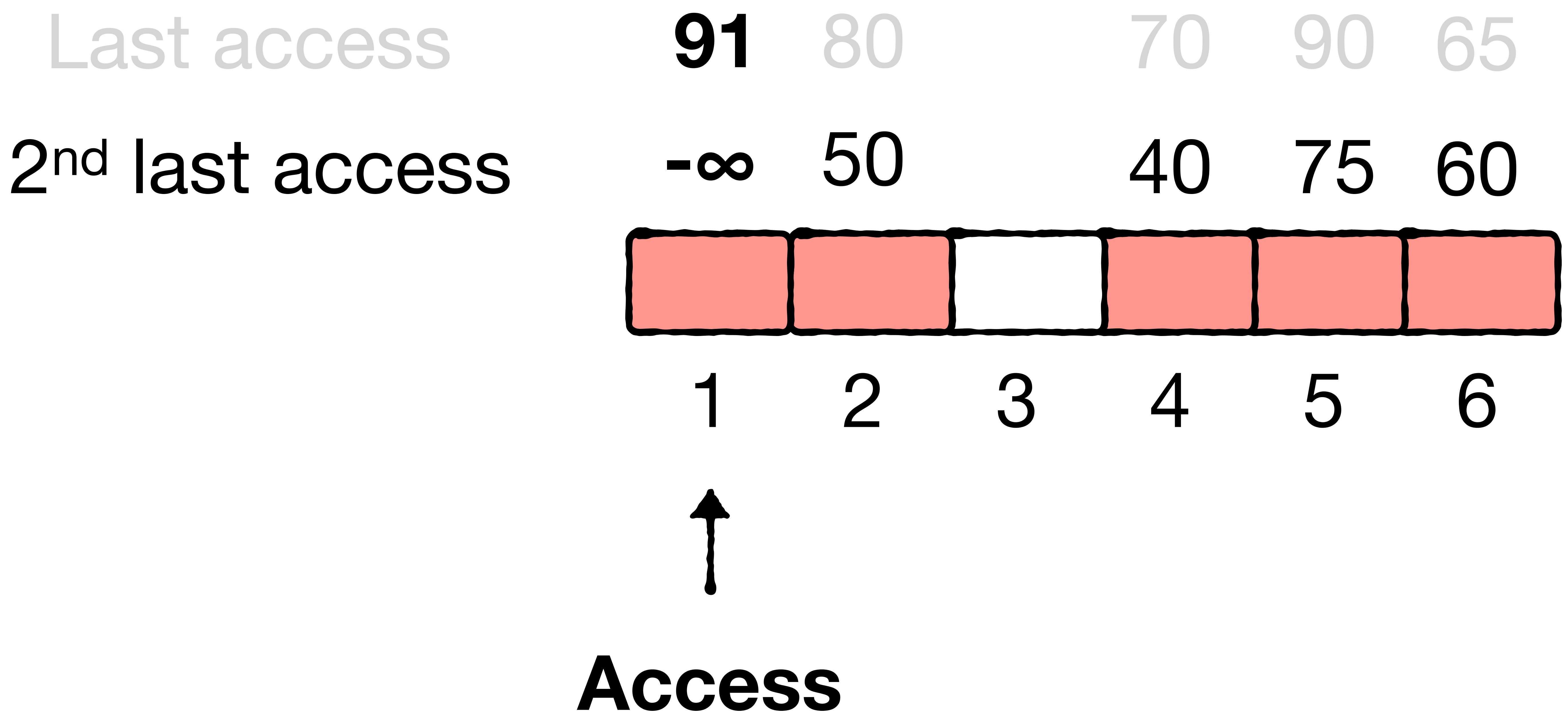
Time now:
91

Solution: do not update entry's access time if new access is within given window of last access



Solution: do not update entry's access time if new access is within given window of last access

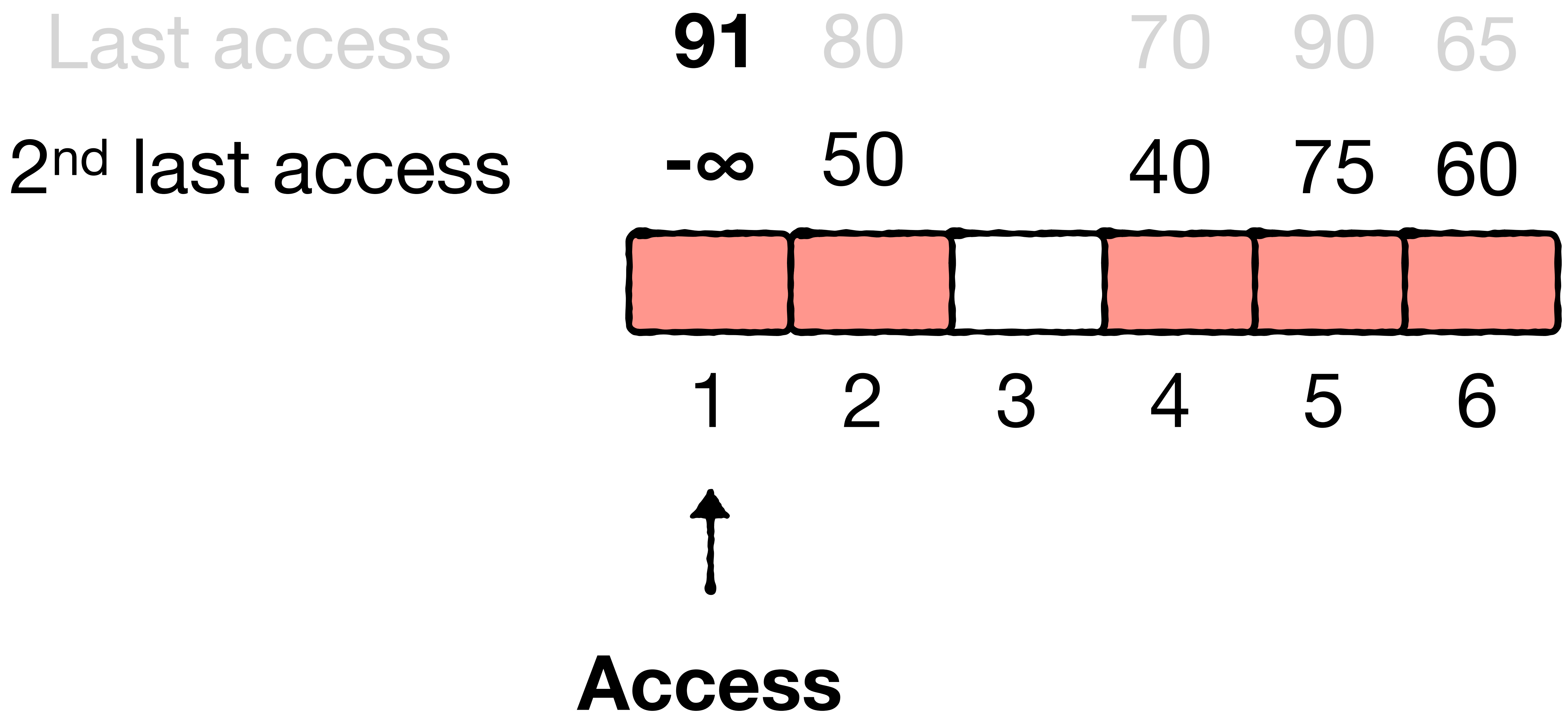
Window size: 5



Time now:
92

Solution: do not update entry's access time if new access is within given window of last access

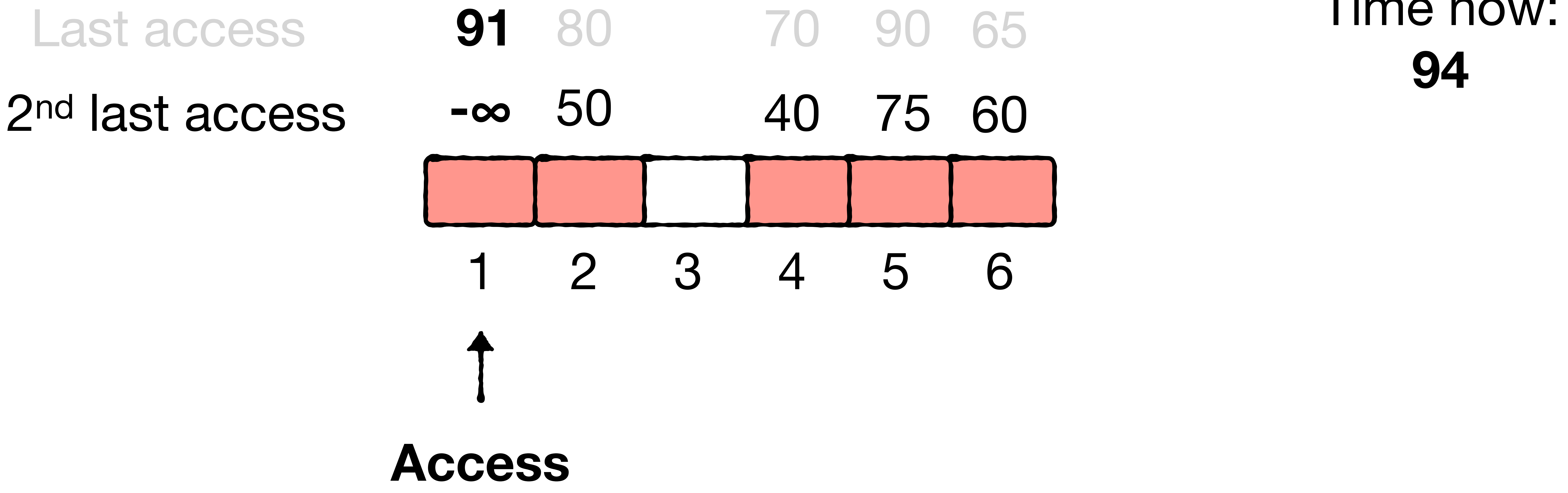
Window size: 5



Time now:
93

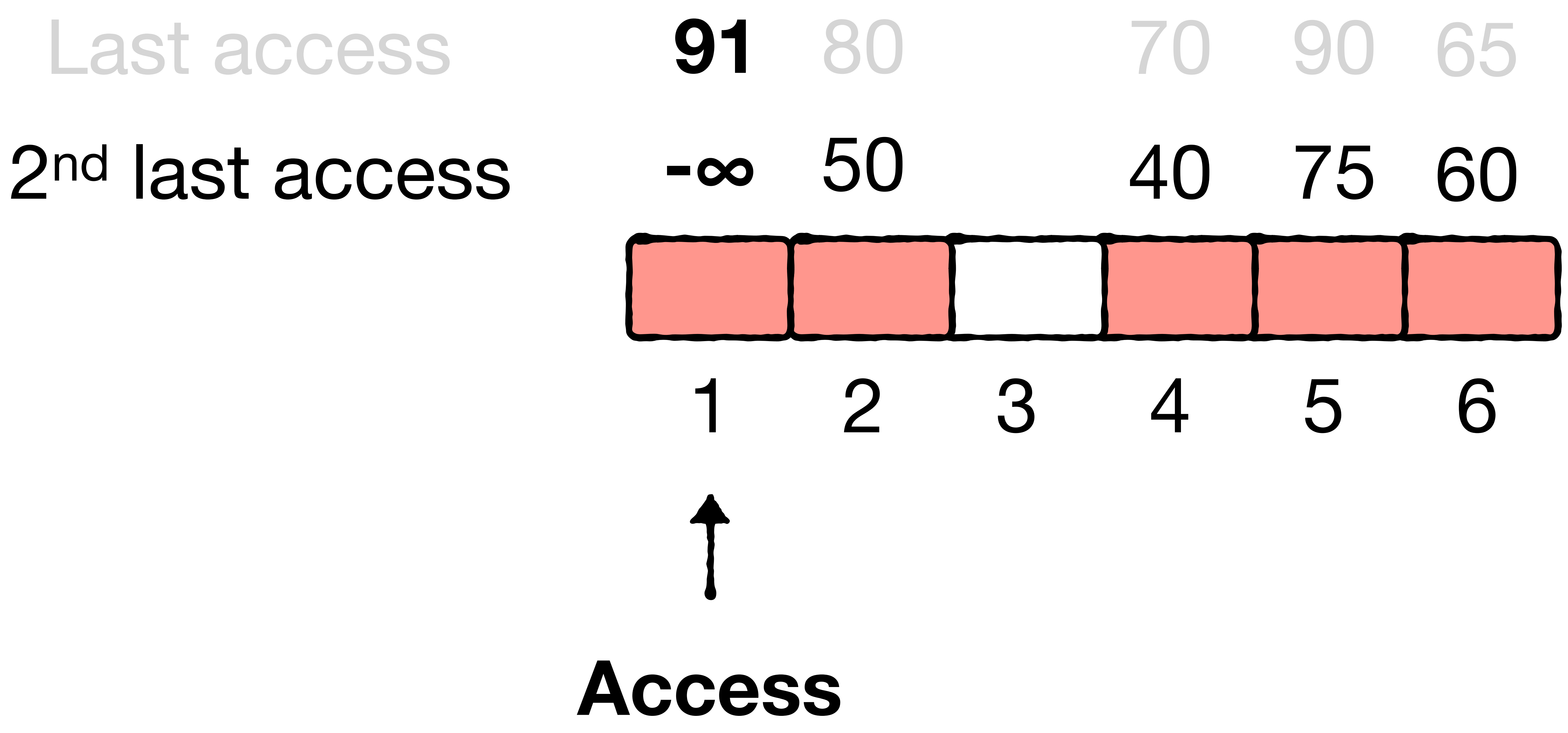
Solution: do not update entry's access time if new access is within given window of last access

Window size: 5



Solution: do not update entry's access time if new access is within given window of last access

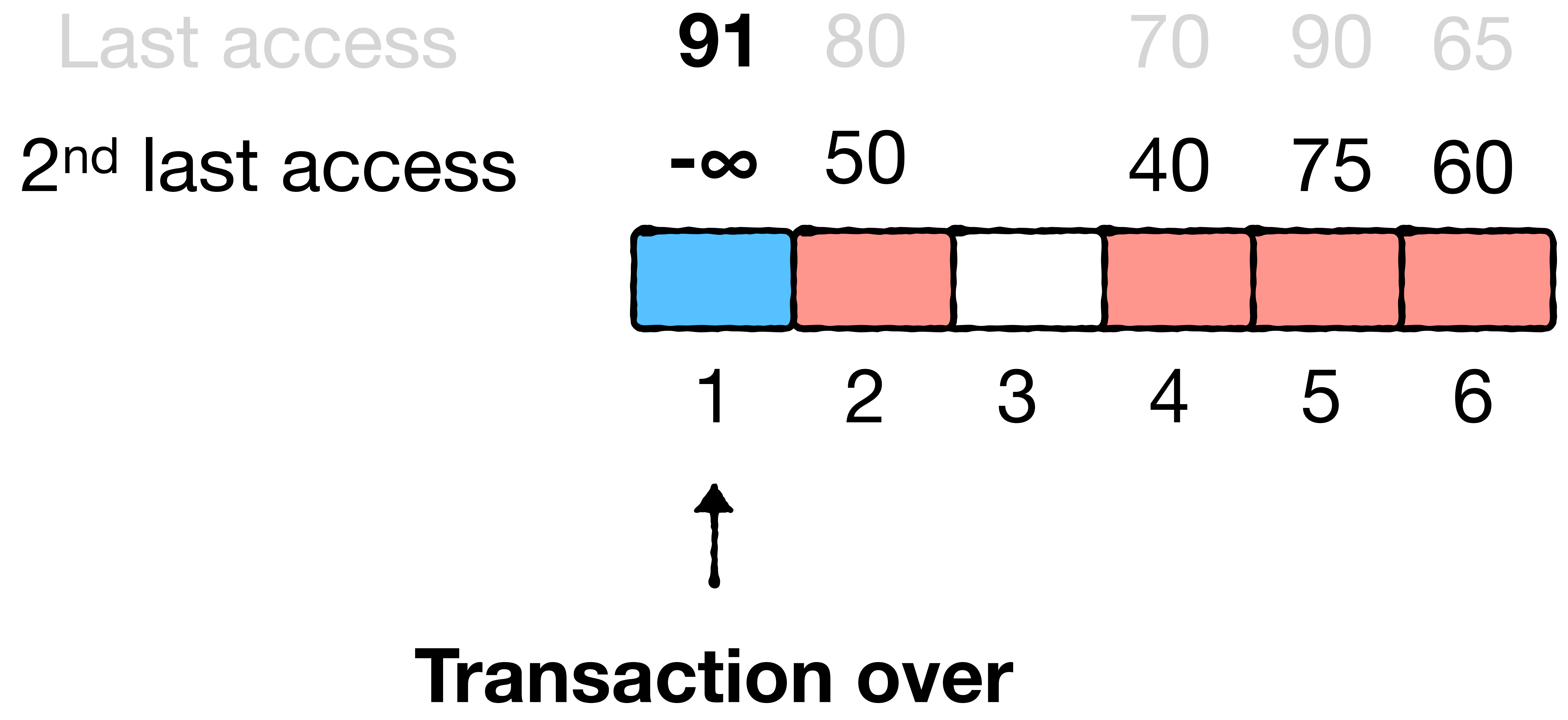
Window size: 5



Time now:
95

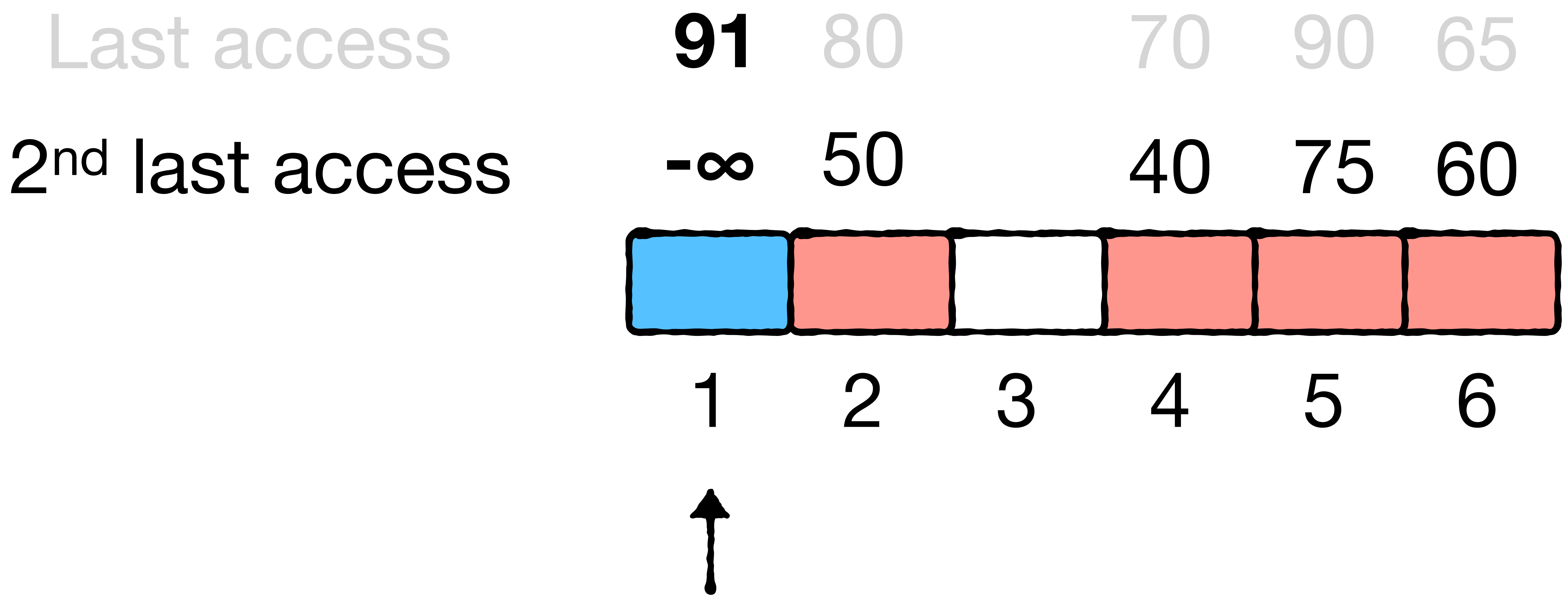
Solution: do not update entry’s access time if new access is within given window of last access

Window size: 5



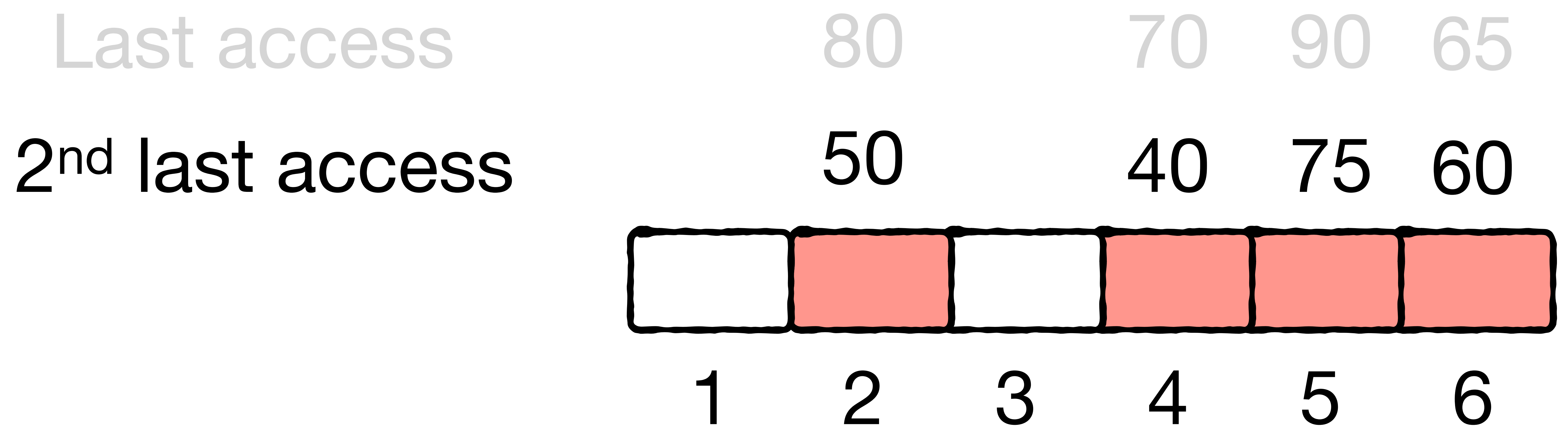
Solution: do not update entry's access time if new access is within given window of last access

Window size: 5

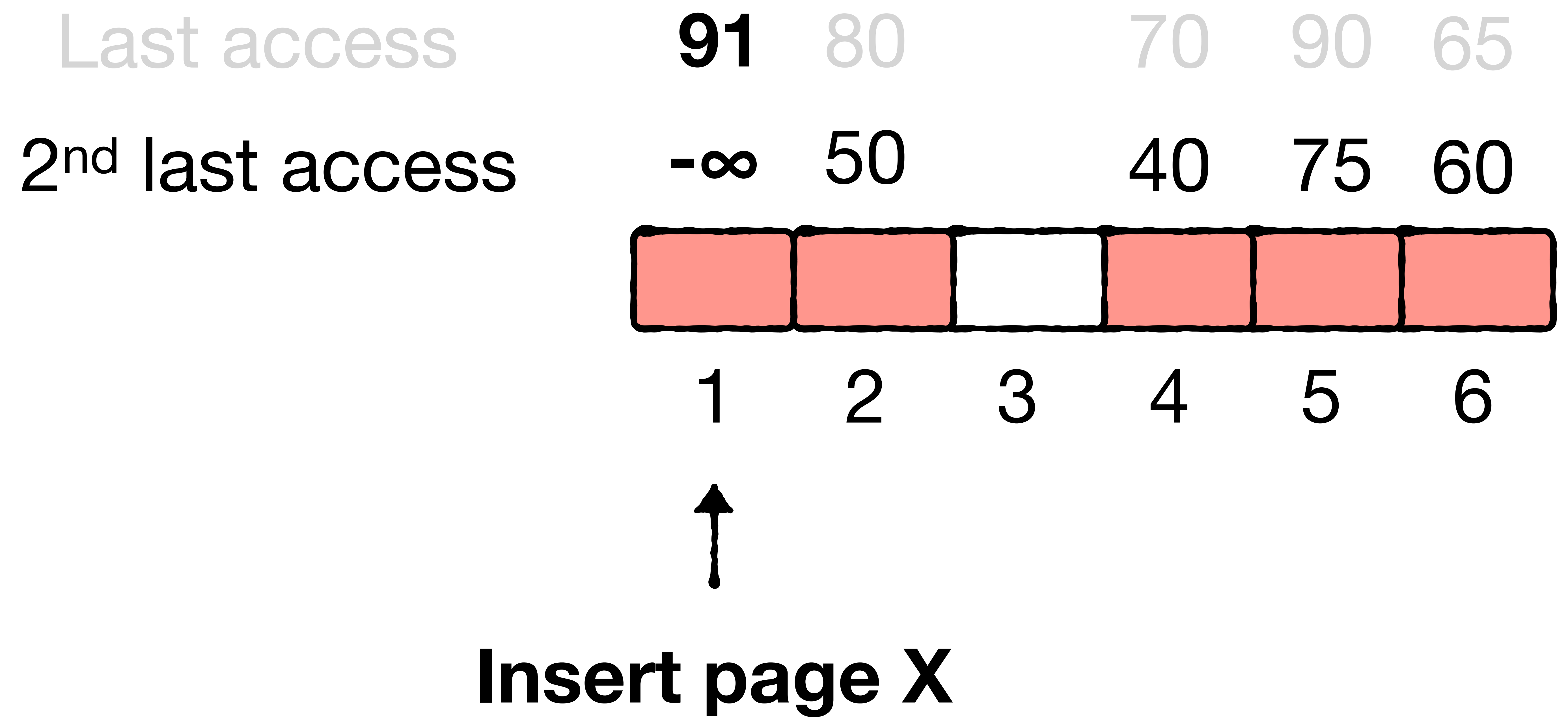


Page returns to being cold, and it is now prioritized for eviction

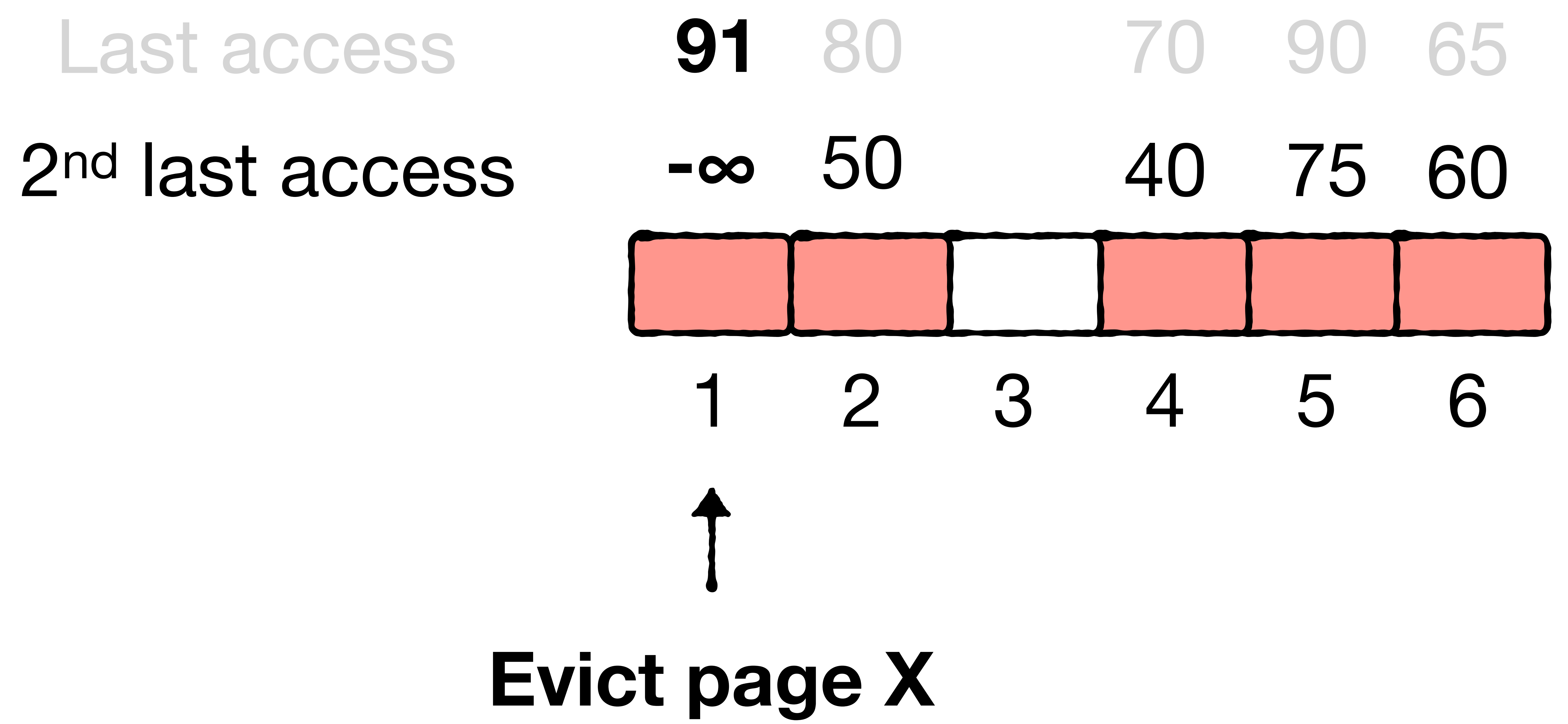
Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



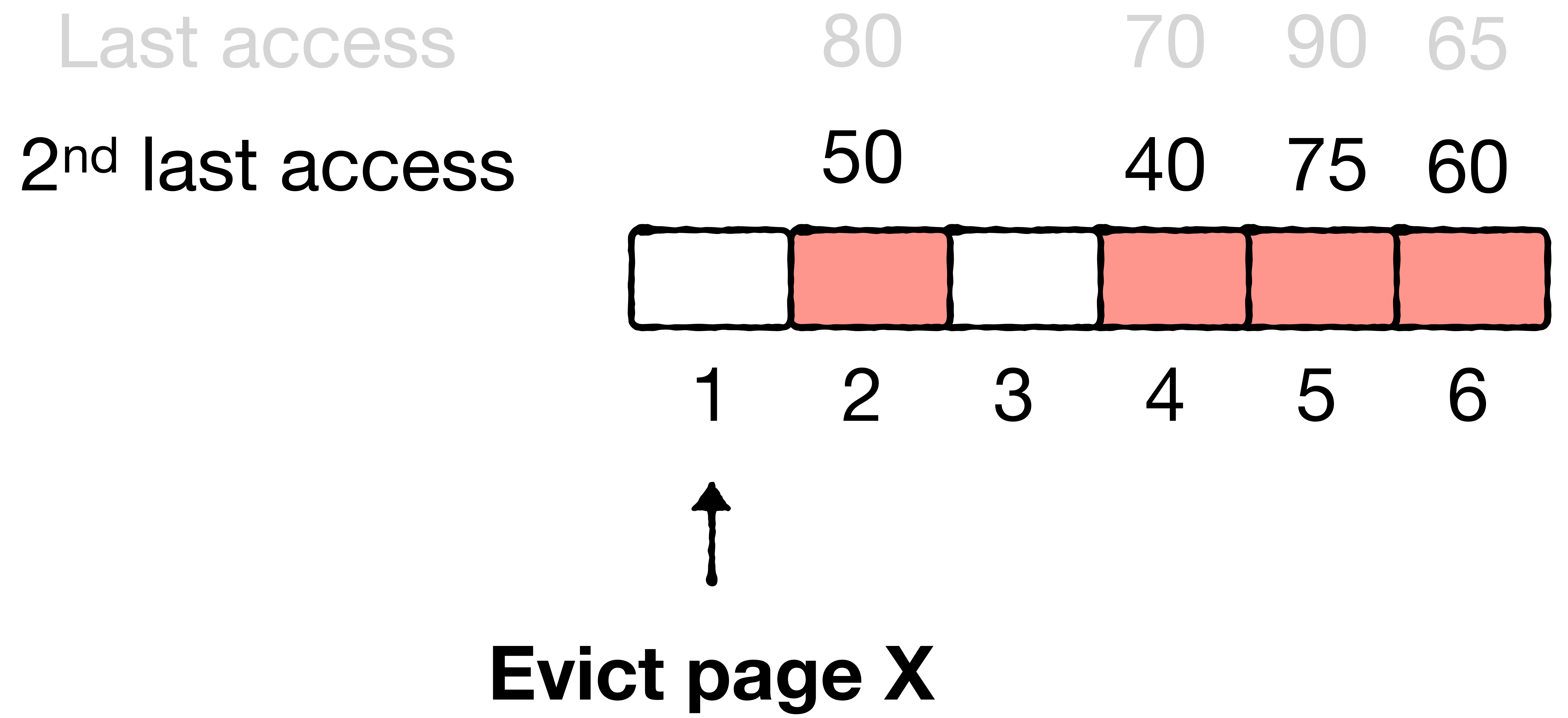
Concern 2: **Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent**



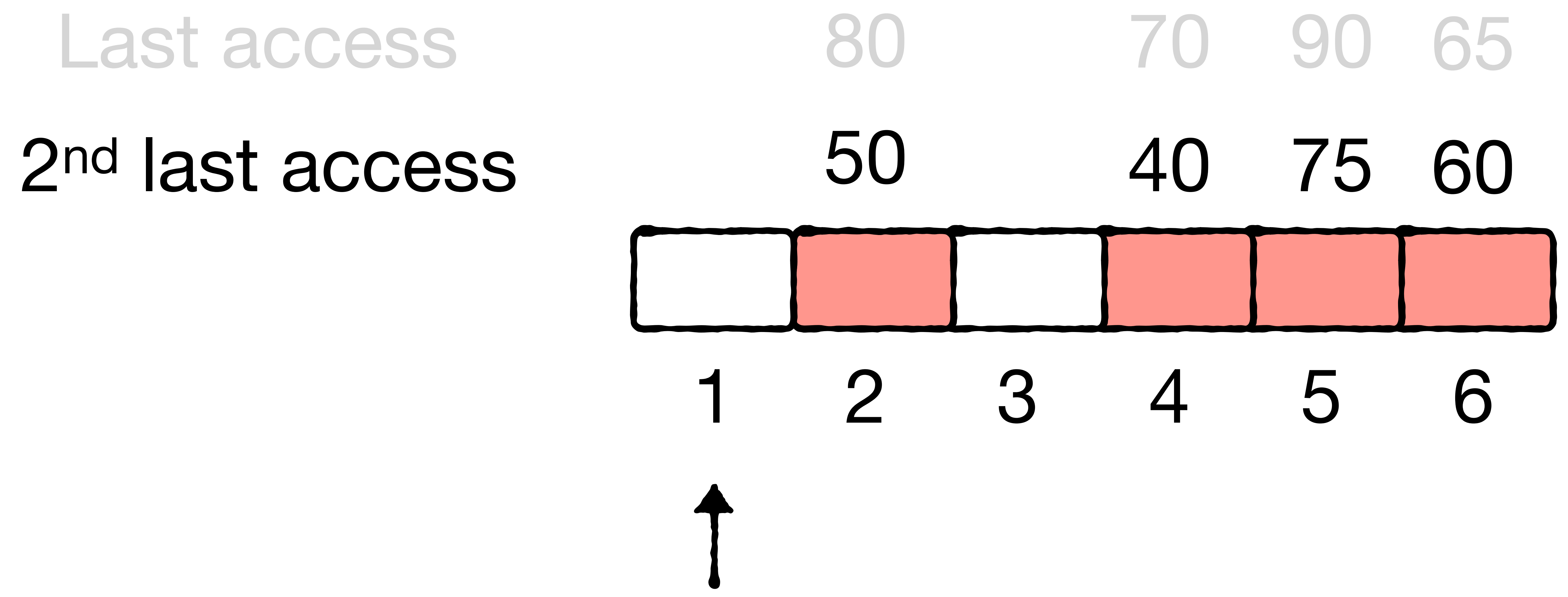
Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent

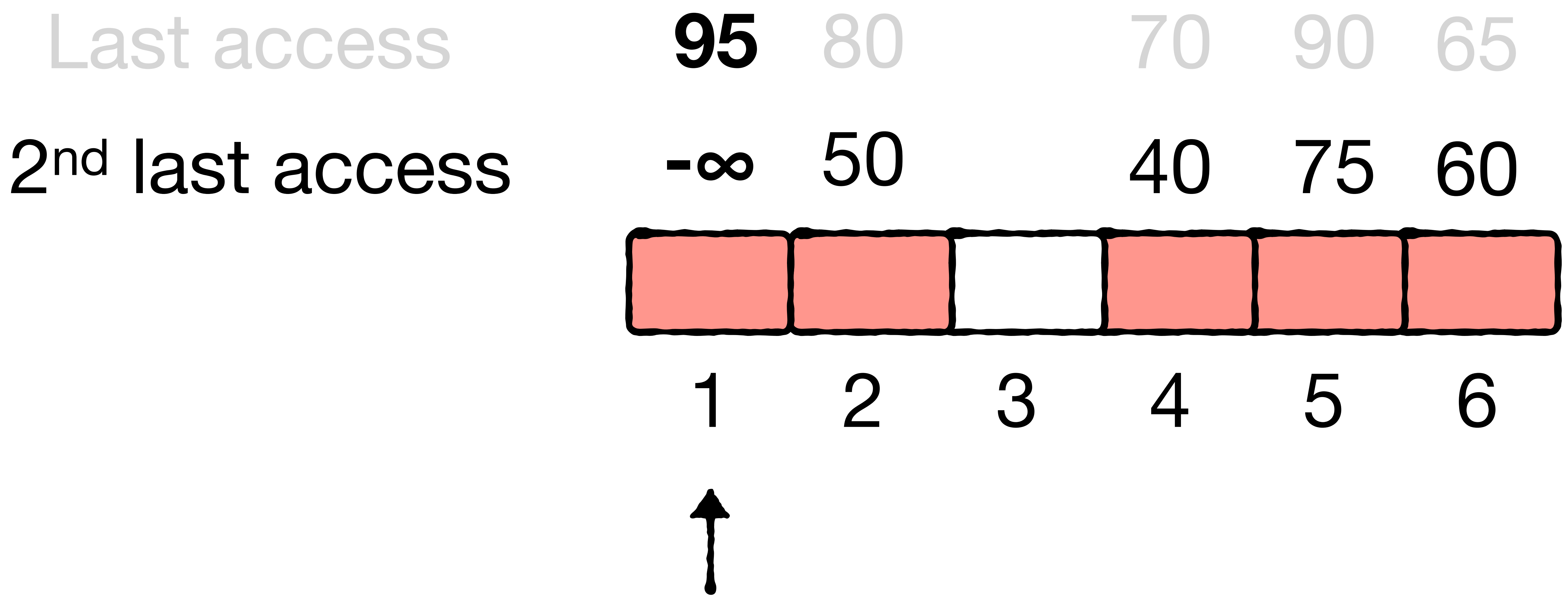


Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



Shortly after, Insert page X again

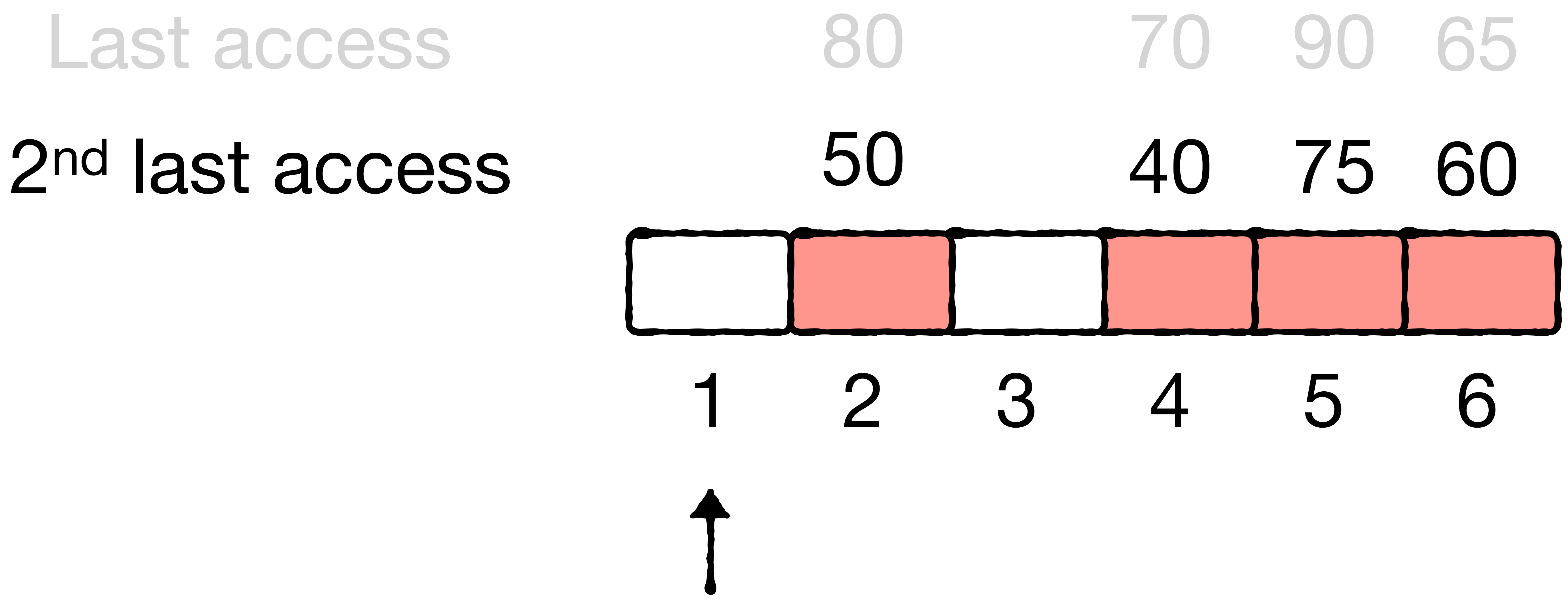
Concern 2: **Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent**



Shortly after, Insert page X again

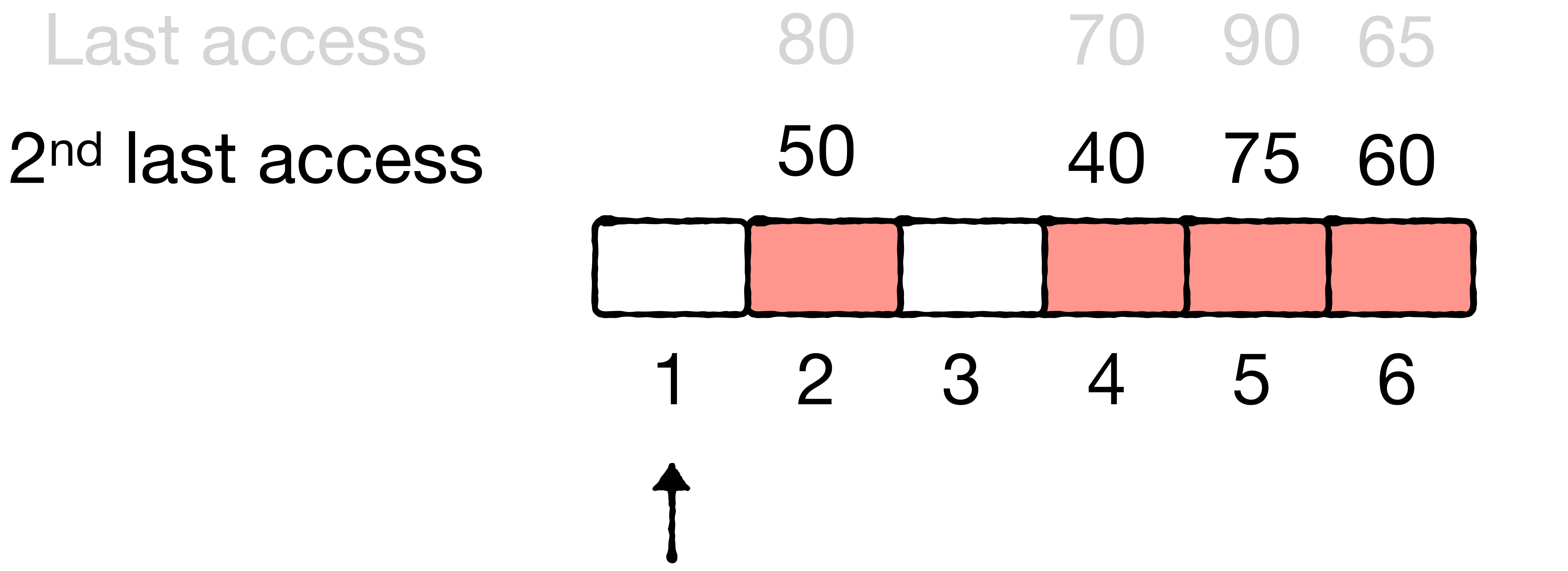
Concern 2: **Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent**

Solutions?



Evict page X again

Solutions: Keep access times for evicted pages in memory

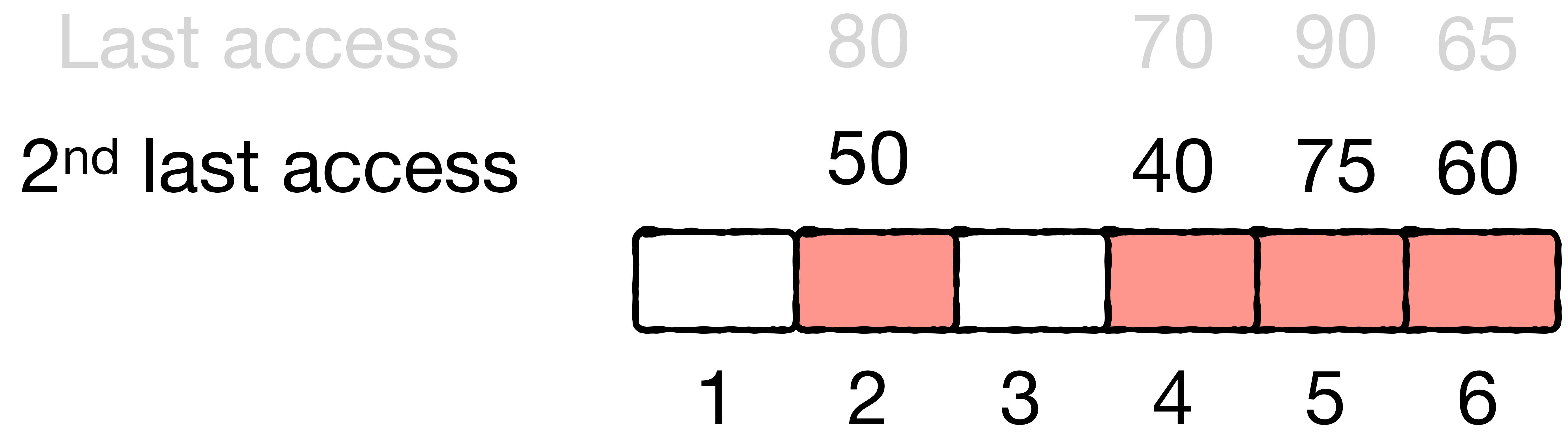


Evict page X again

Solutions: Keep access times for evicted pages in memory

Access Hash Table (HT)

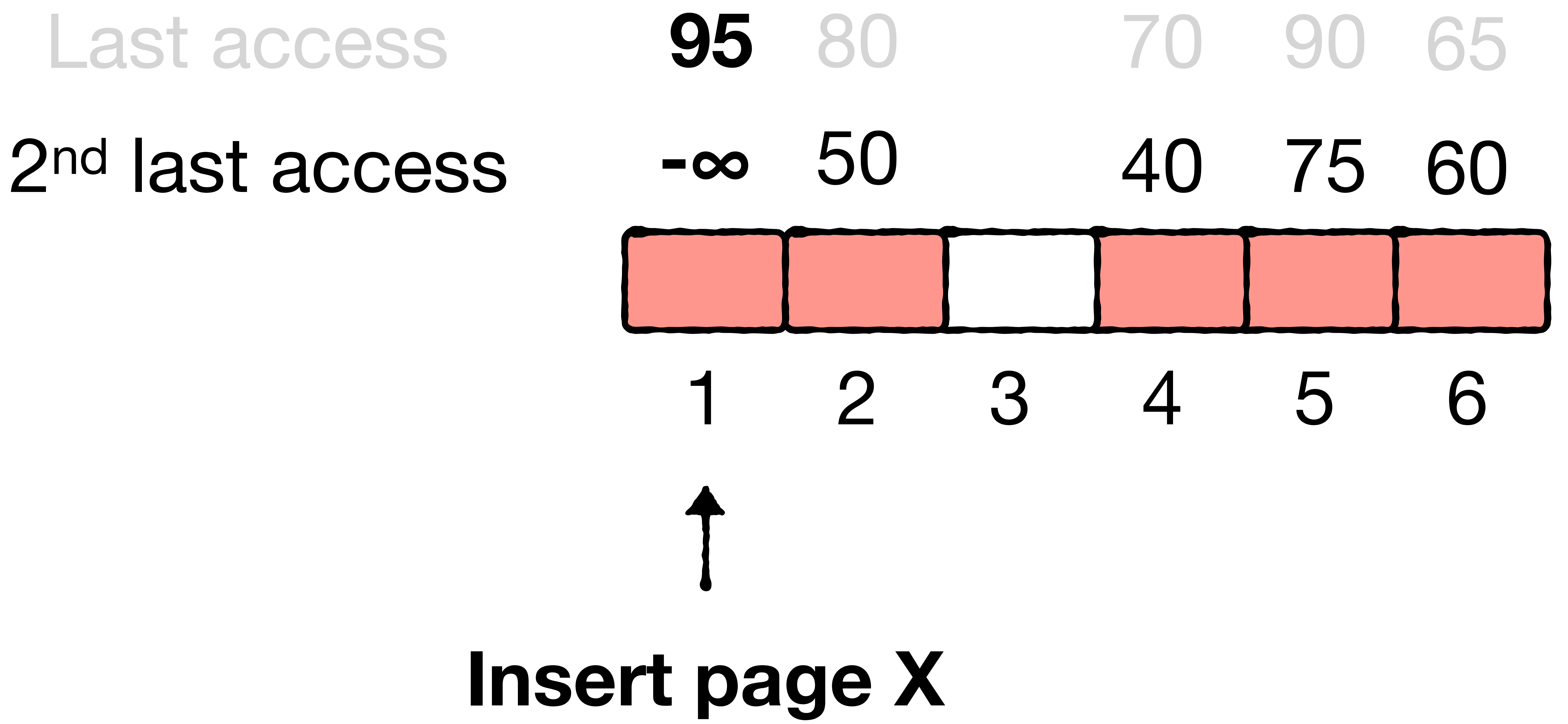
Page ID → Last K access times



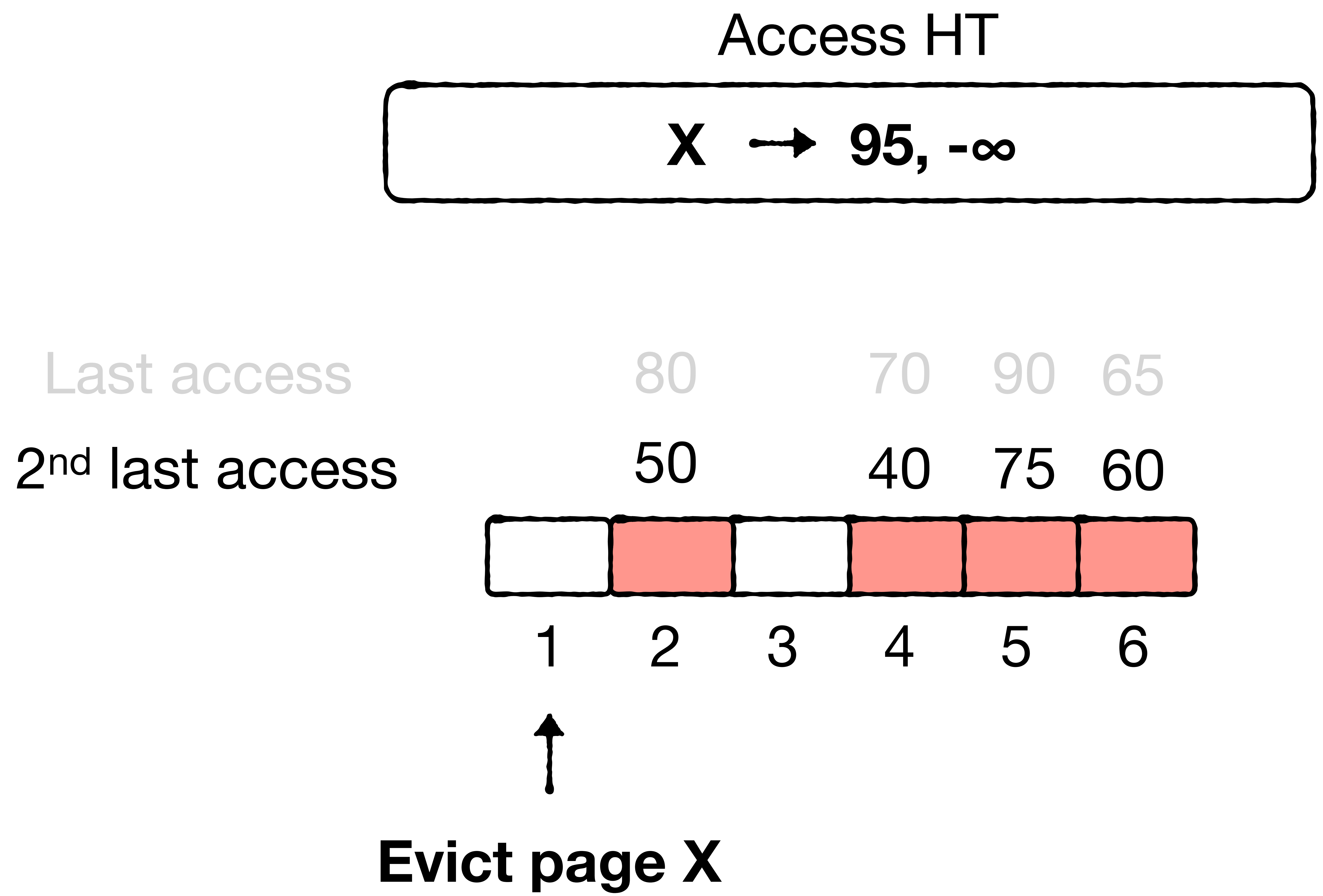
Solutions: Keep access times for evicted pages in memory

Access HT

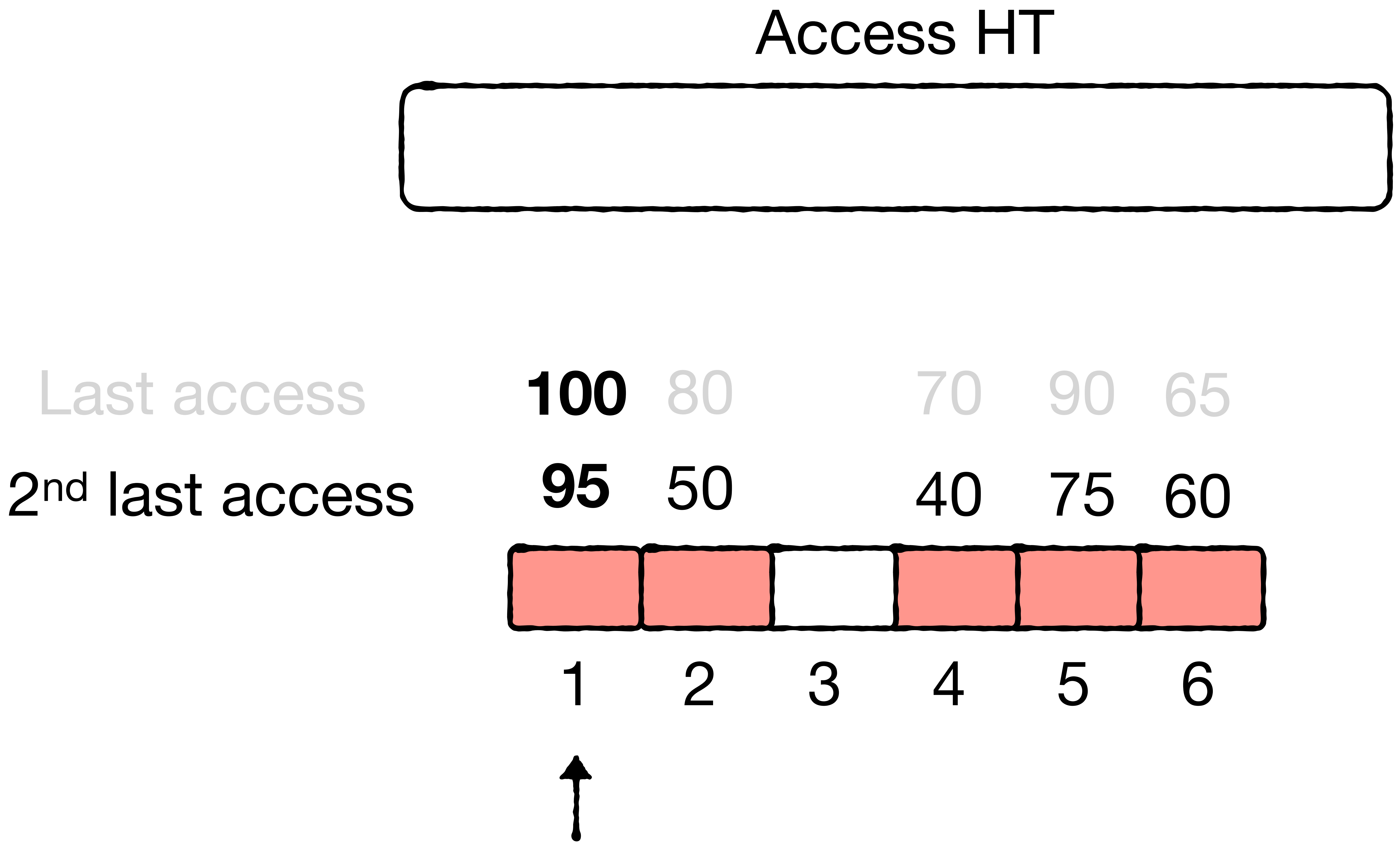
Page ID → Last K access times



Solutions: Keep access times for evicted pages in memory

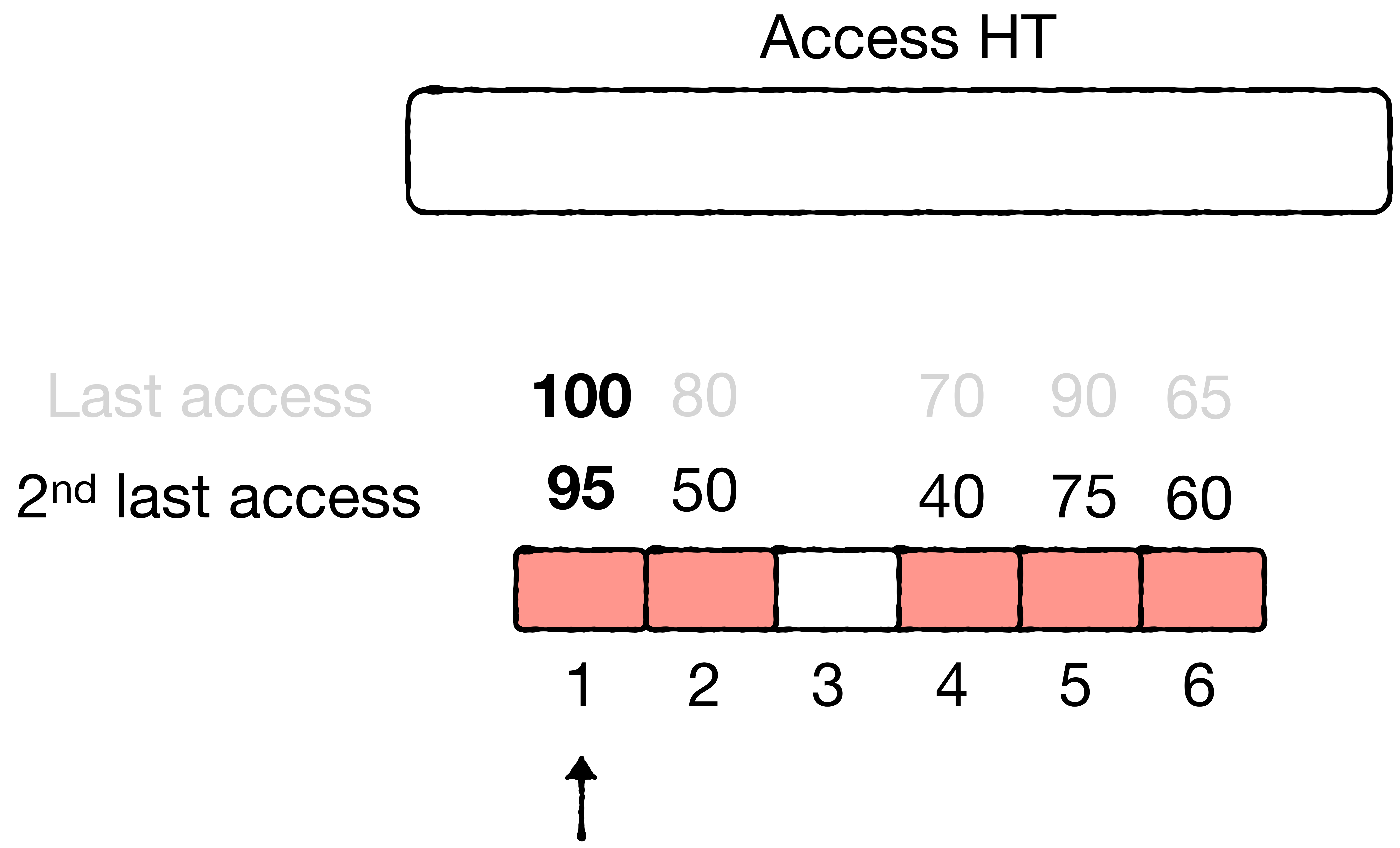


Solutions: Keep access times for evicted pages in memory



Shortly after, Insert page X again (time 100)

Solutions: Keep access times for evicted pages in memory



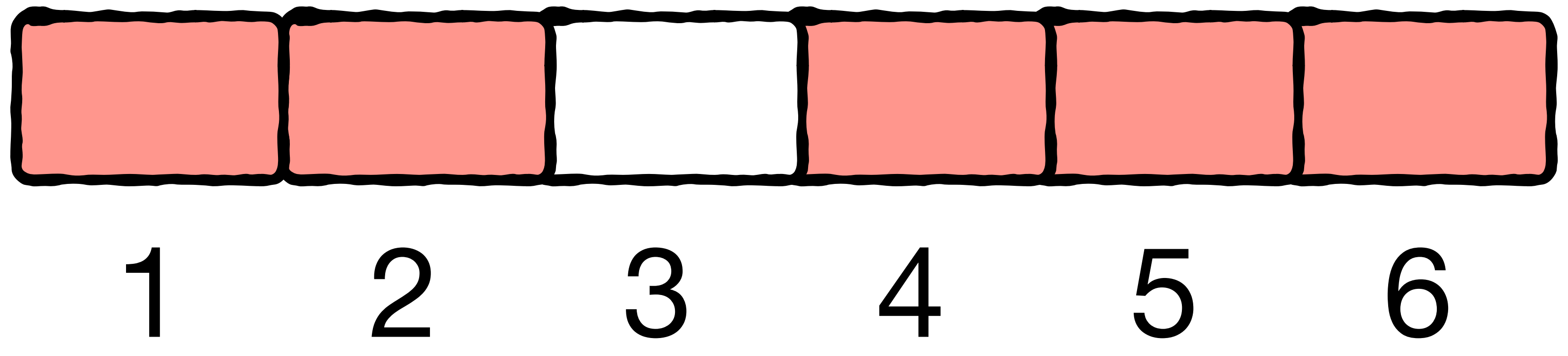
Page X is no longer a priority eviction candidate

How to implement this?

$\downarrow$

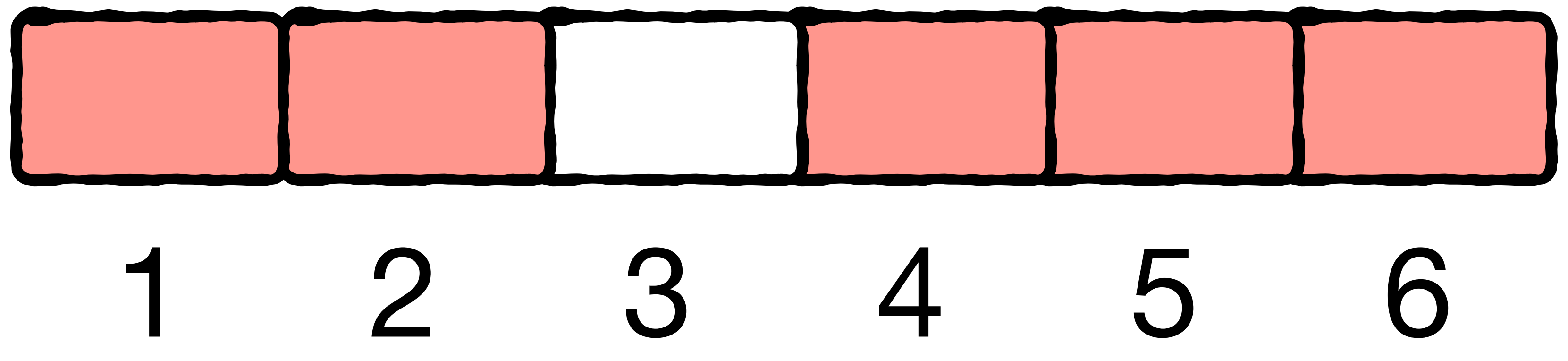
Access HT

$\boxed{\textbf{Page ID} \rightarrow \textbf{Last 2 access times}}$



Access HT + FIFO

Page ID $\rightarrow$ Last 2 access times



Picture so far:

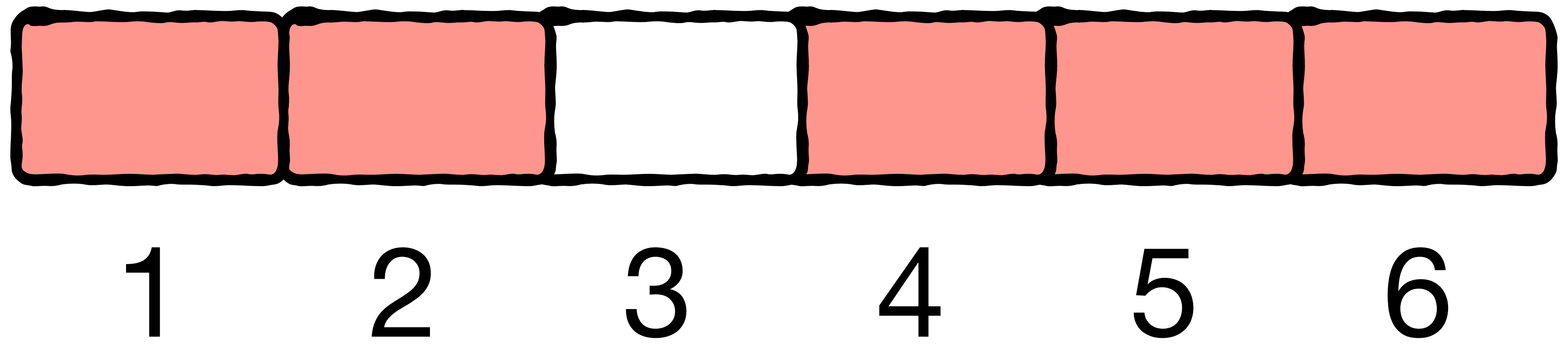
Access HT + FIFO

Page ID → Last 2 access times

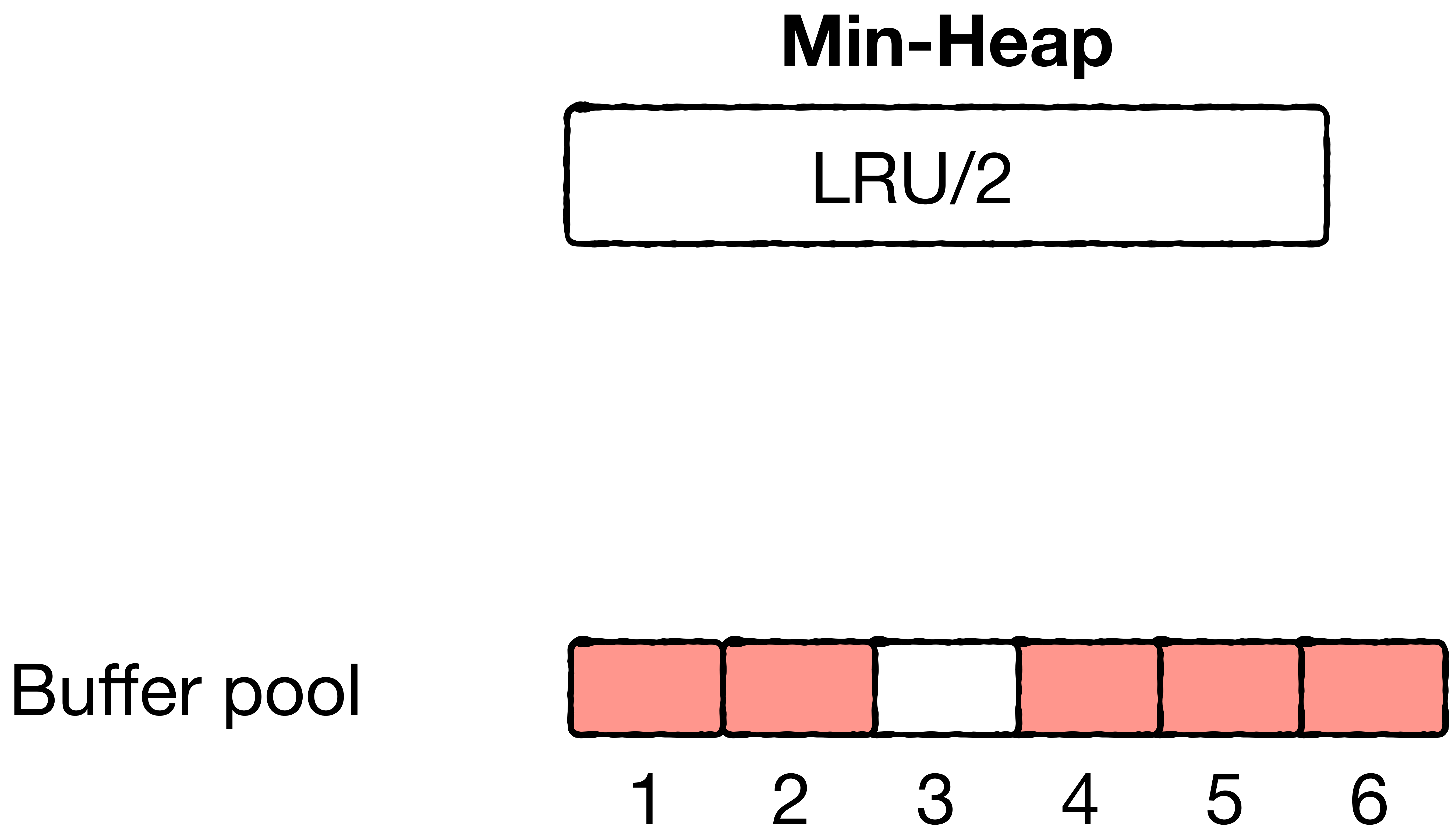
Min-Heap

LRU/2

Buffer pool

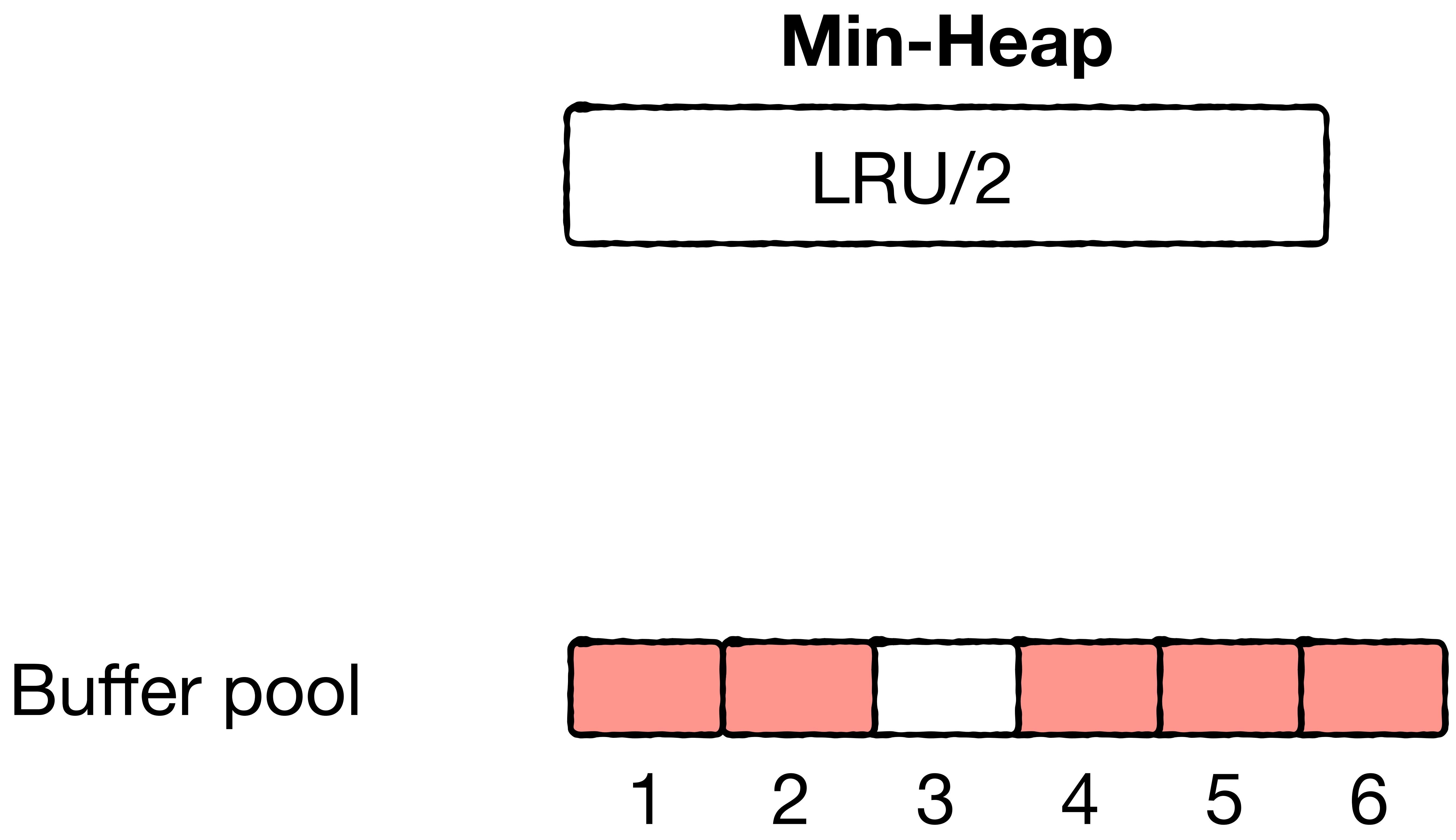


Can we improve on the O(log N) cost of updating min-heap?



Break

Can we improve on the O(log N) cost of updating min-heap in LRU-2

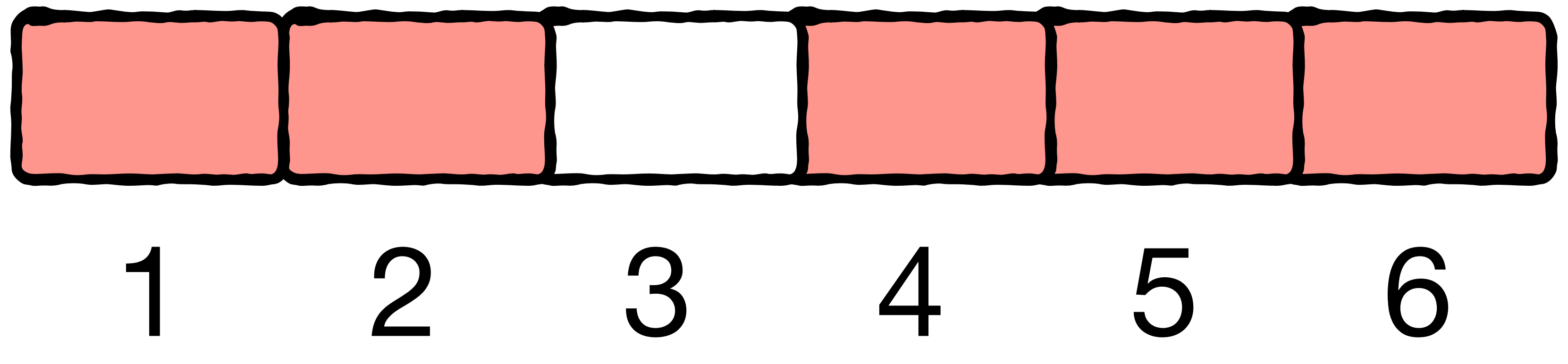


2Q: a low overhead high performance buffer management replacement algorithm

Theodore Johnson, Dennis Shasha

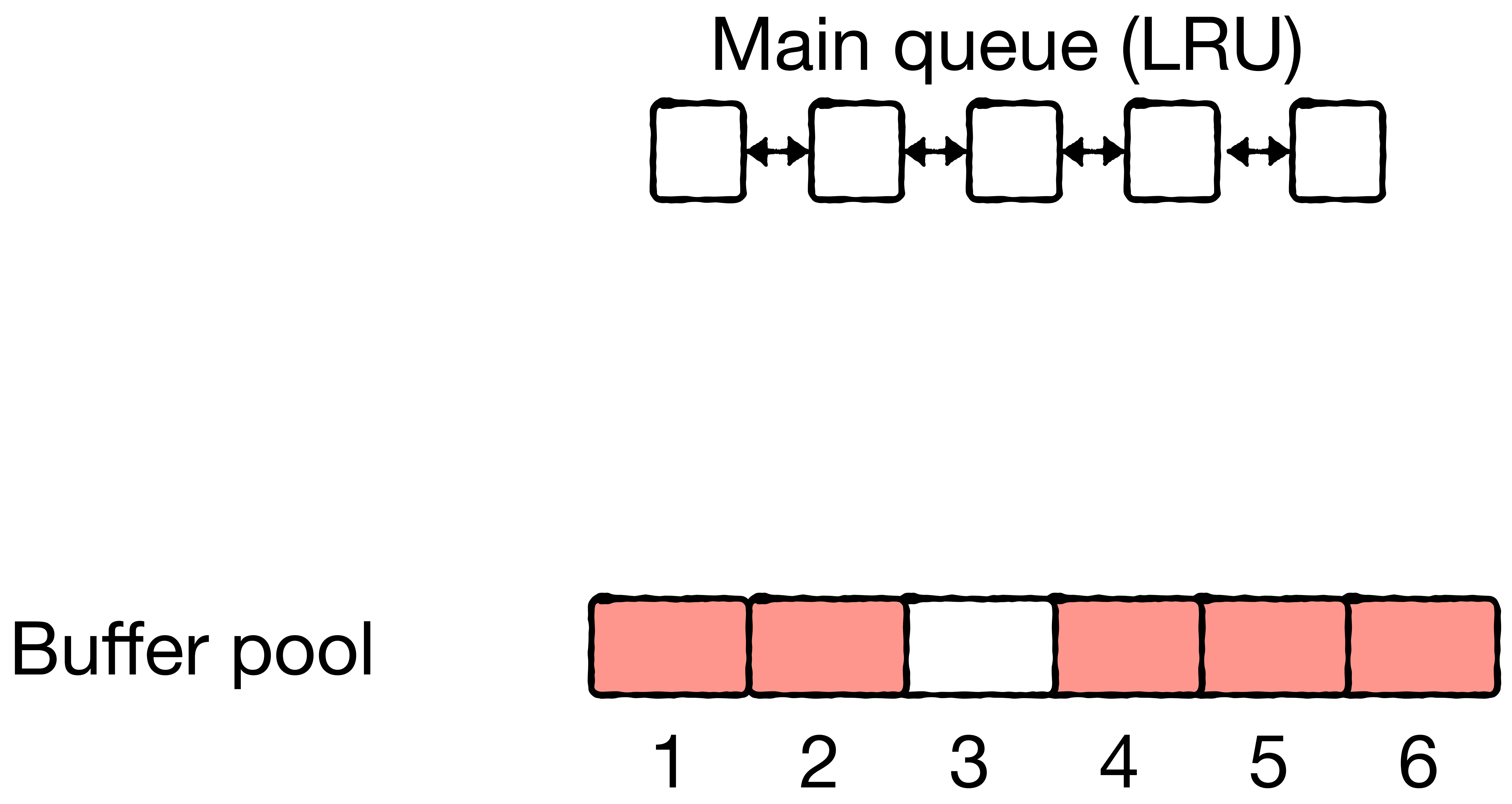
VLBD 1994

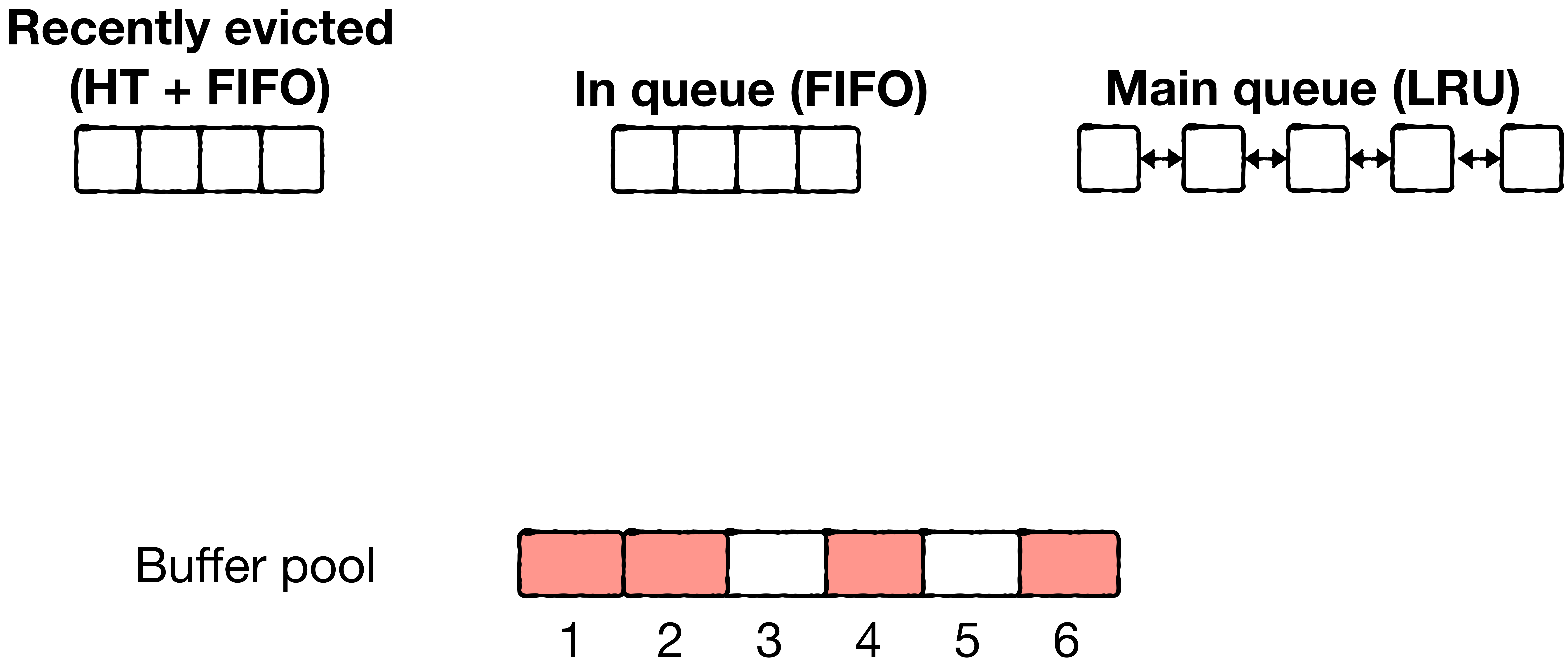
Buffer pool

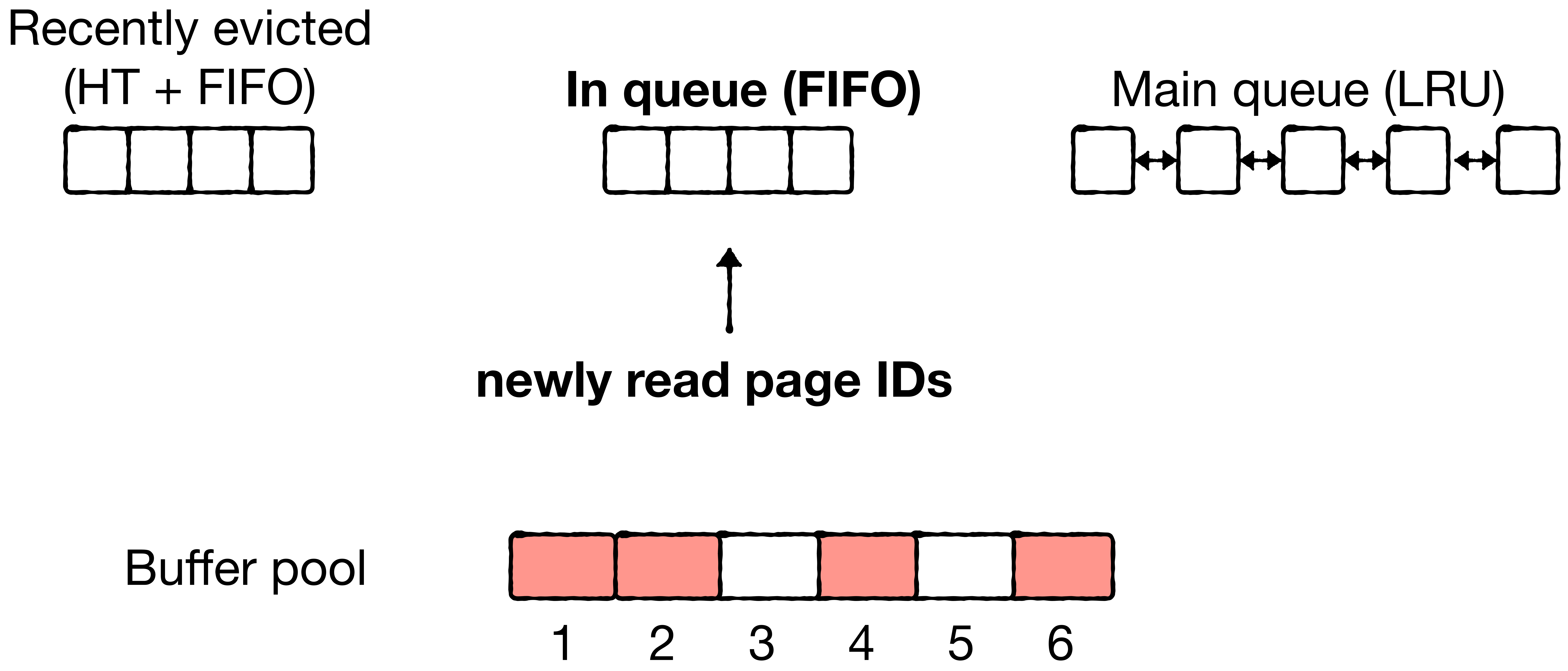


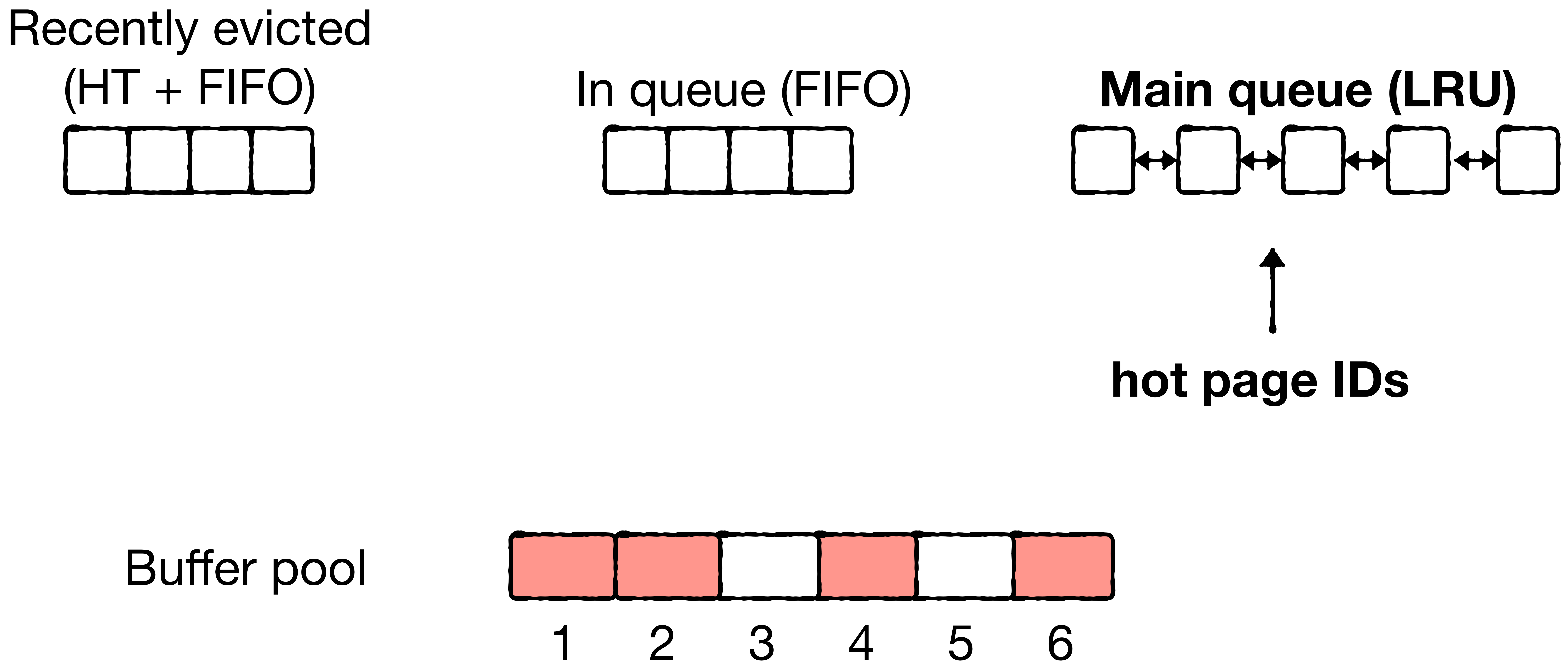
(1) use LRU queue as it has O(1) time

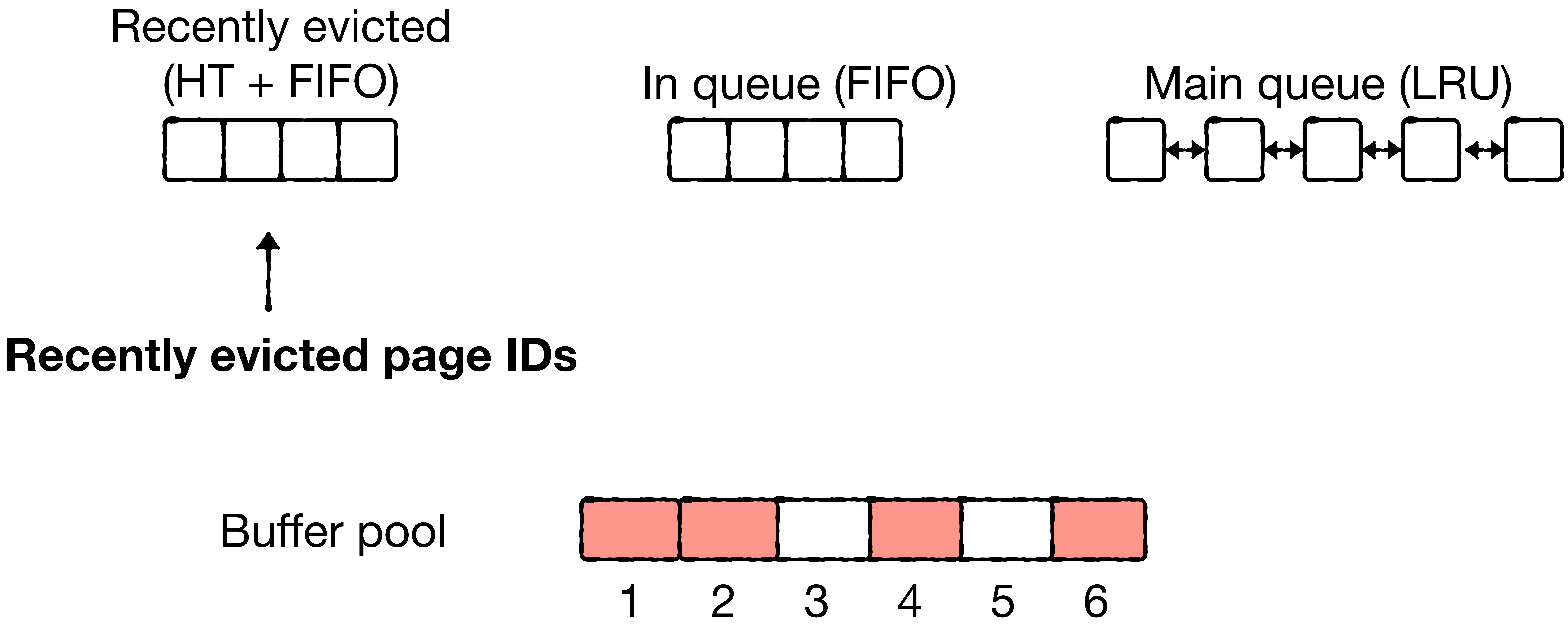
(2) Ensure all entries in queue had 2 recent accesses

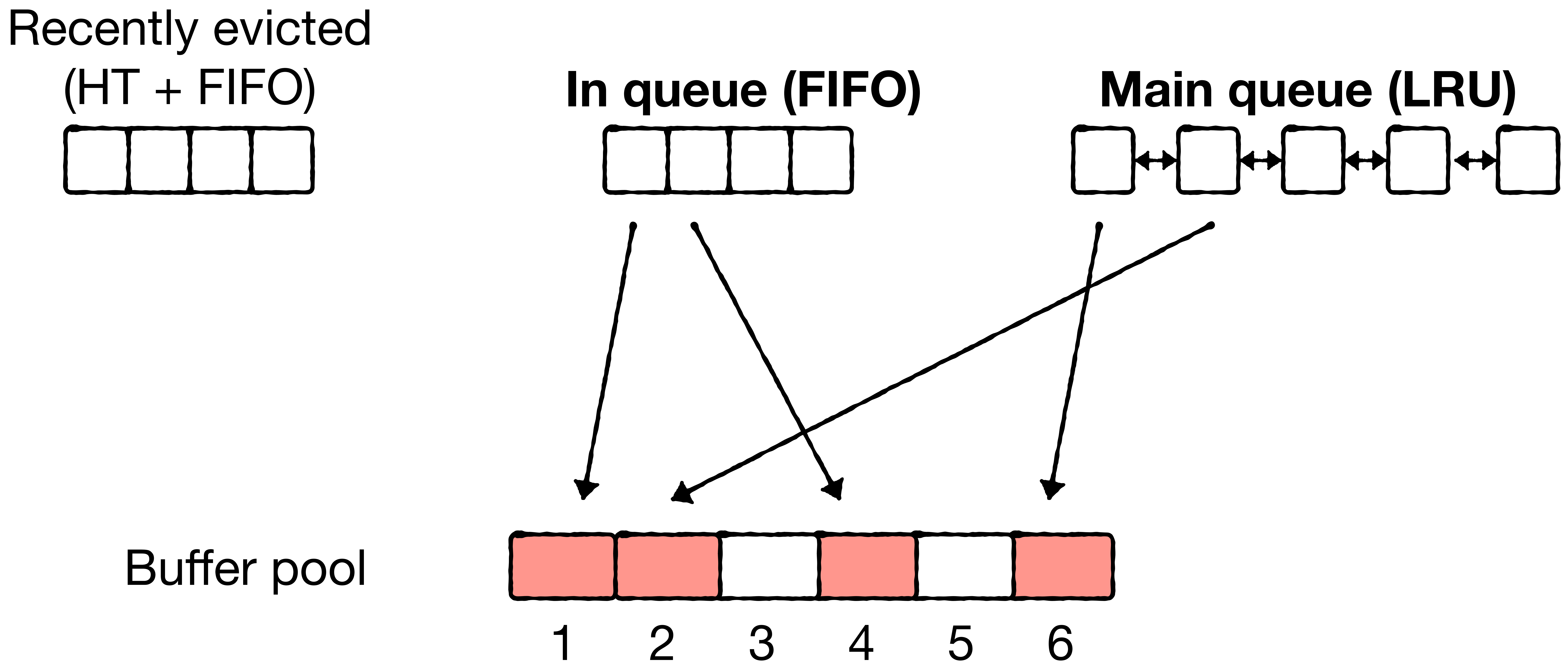


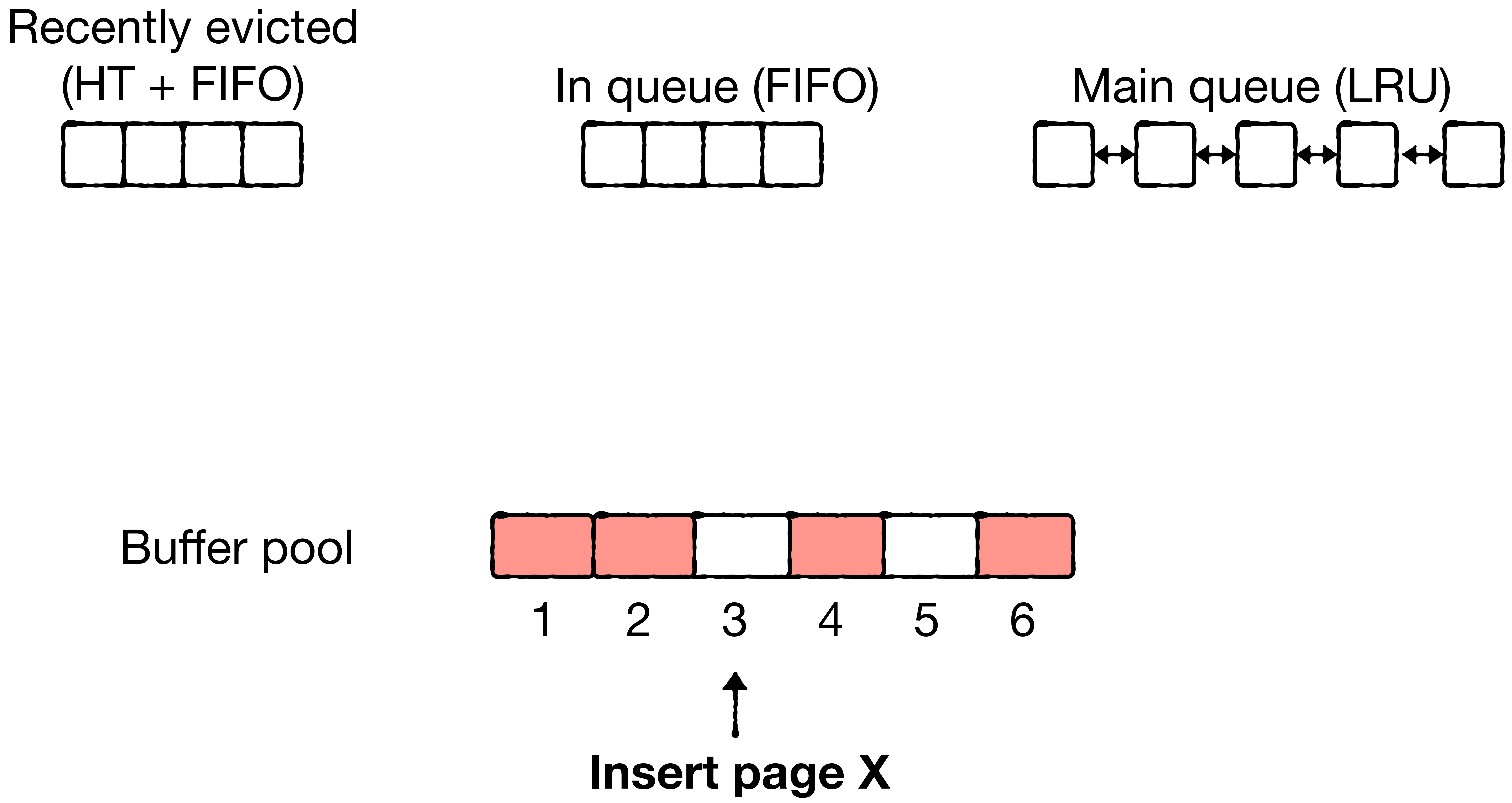

Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Buffer pool
1
2
3
4
5
6

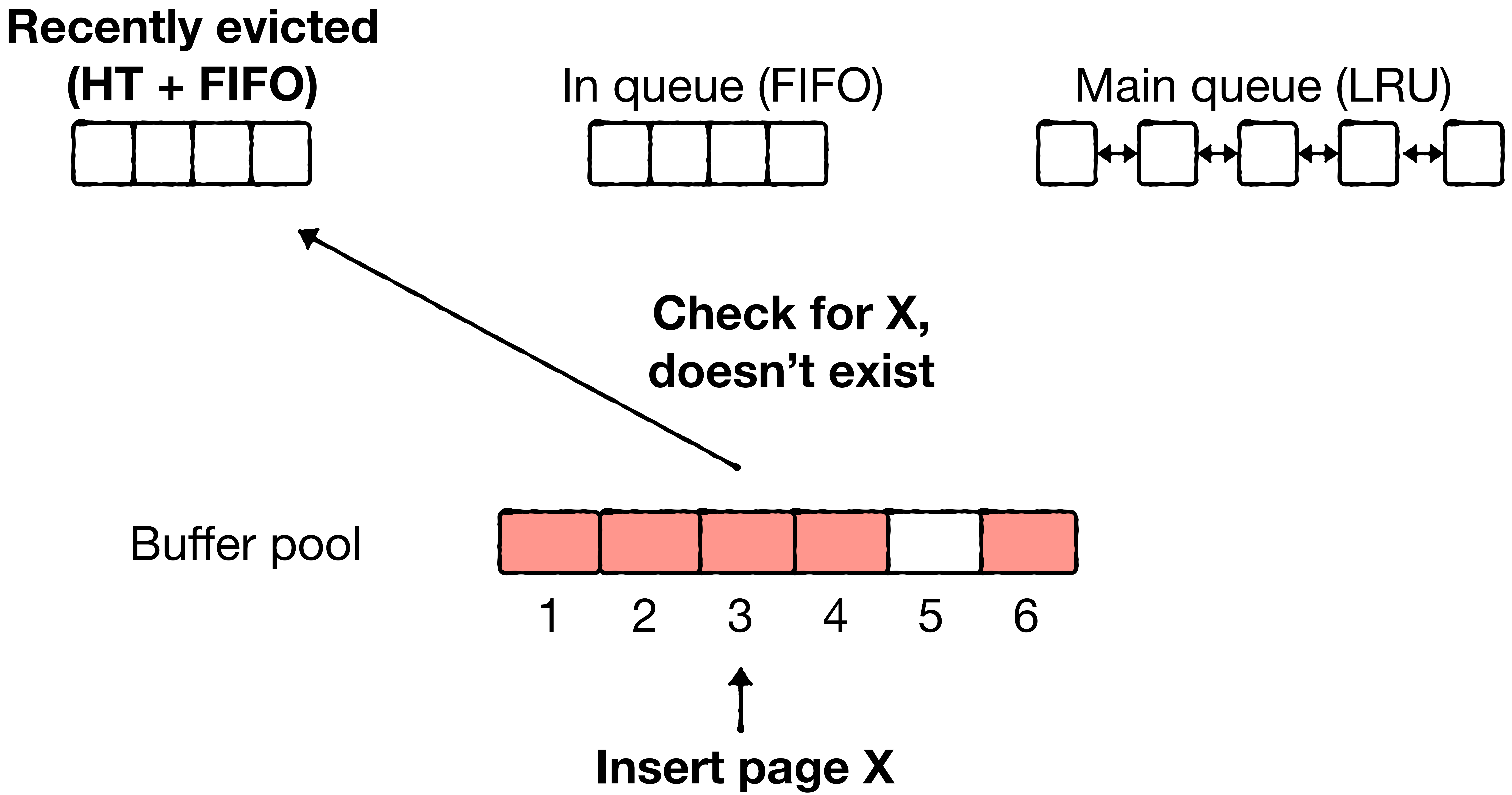

Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
newly read page IDs
Buffer pool
1
2
3
4
5
6

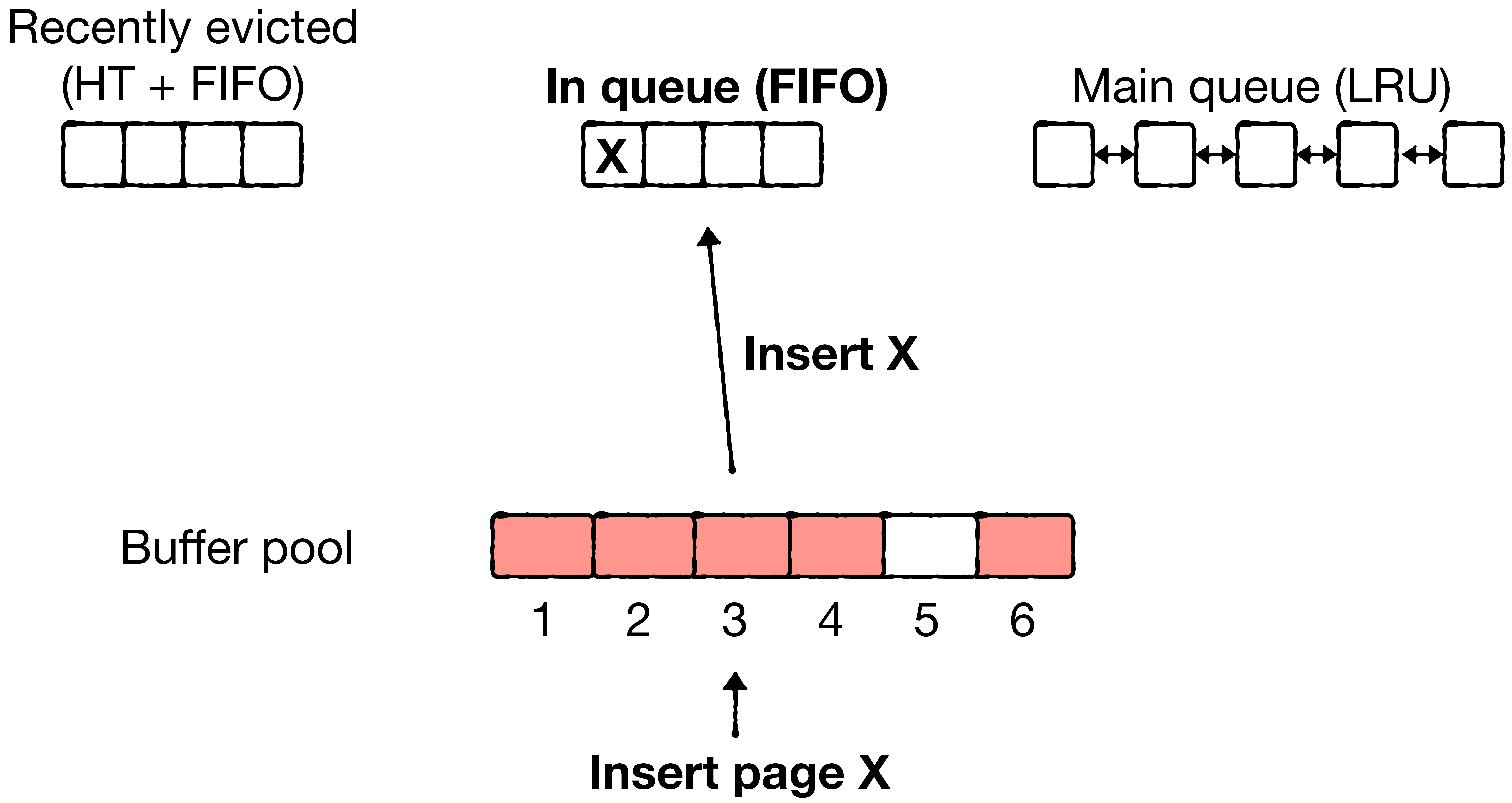

Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
hot page IDs
Buffer pool
1
2
3
4
5
6

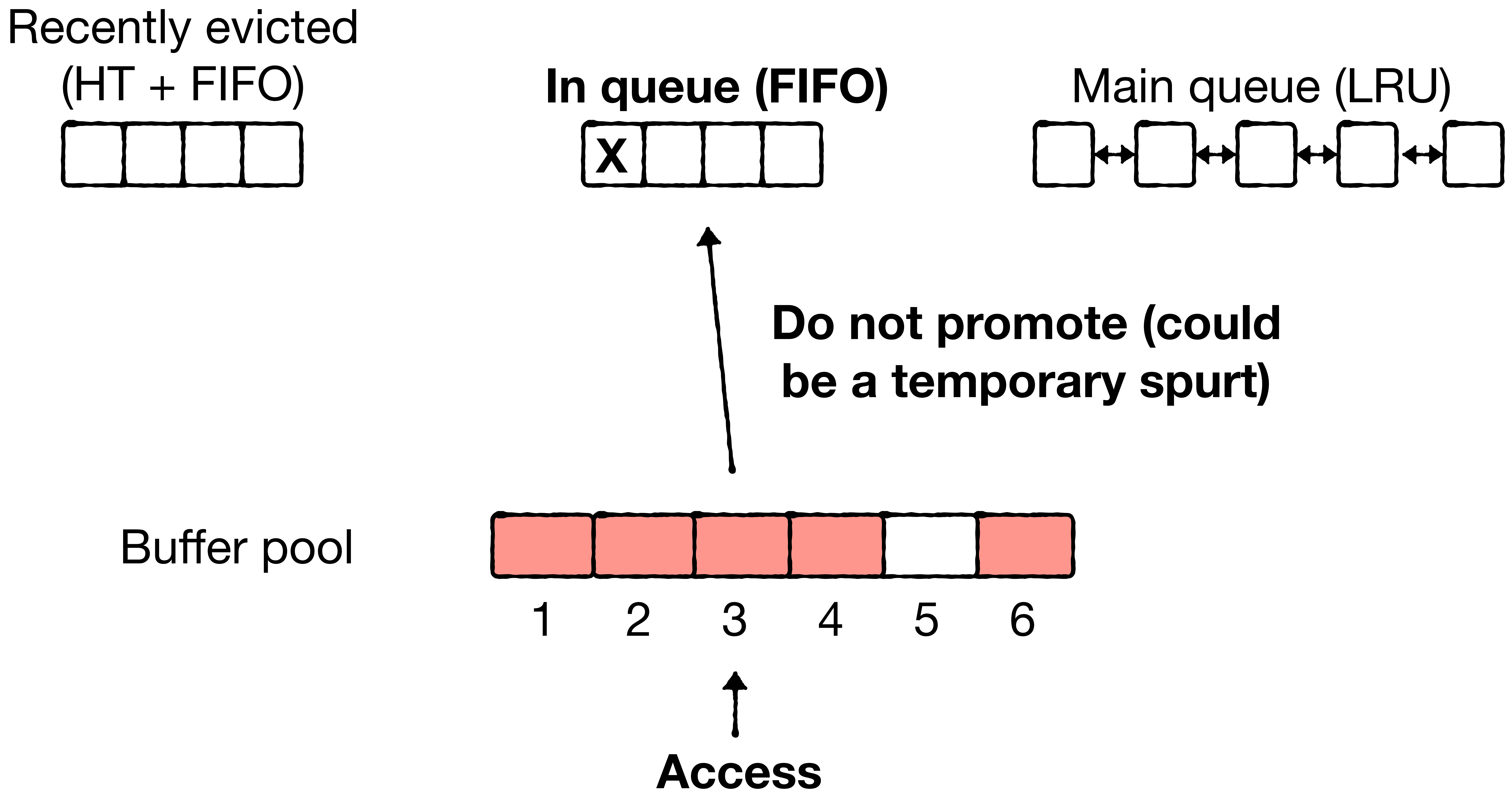

Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Recently evicted page IDs
Buffer pool
1
2
3
4
5
6

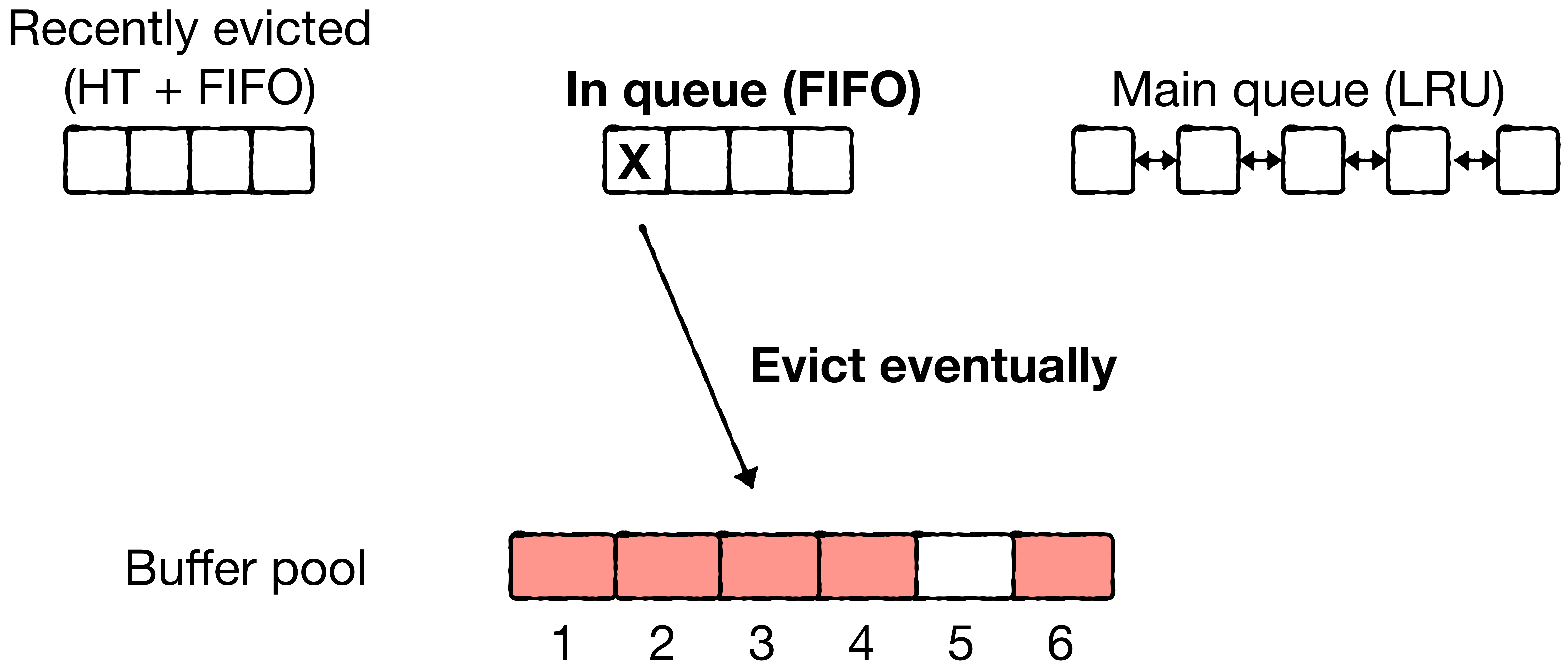

Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Buffer pool
1
2
3
4
5
6

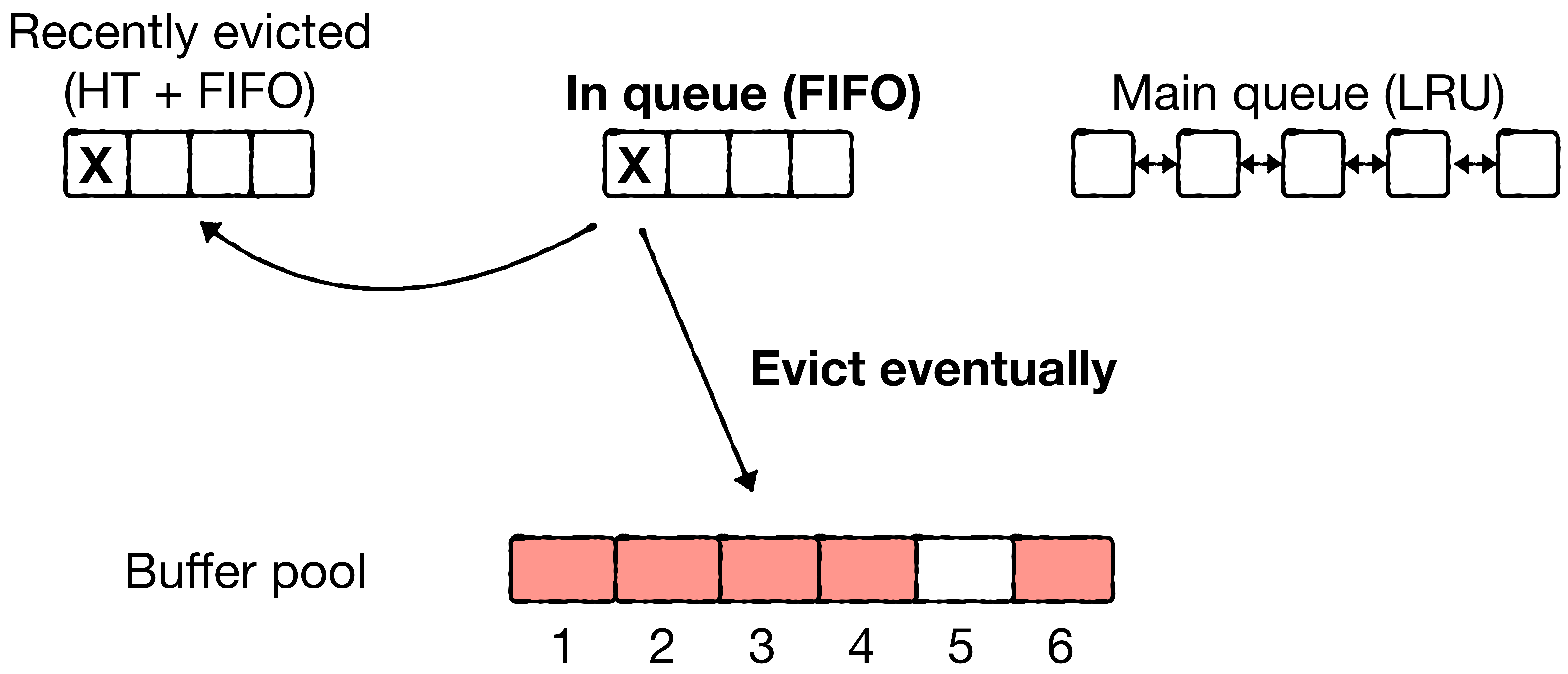

Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Buffer pool
1
2
3
4
5
6
Insert page X

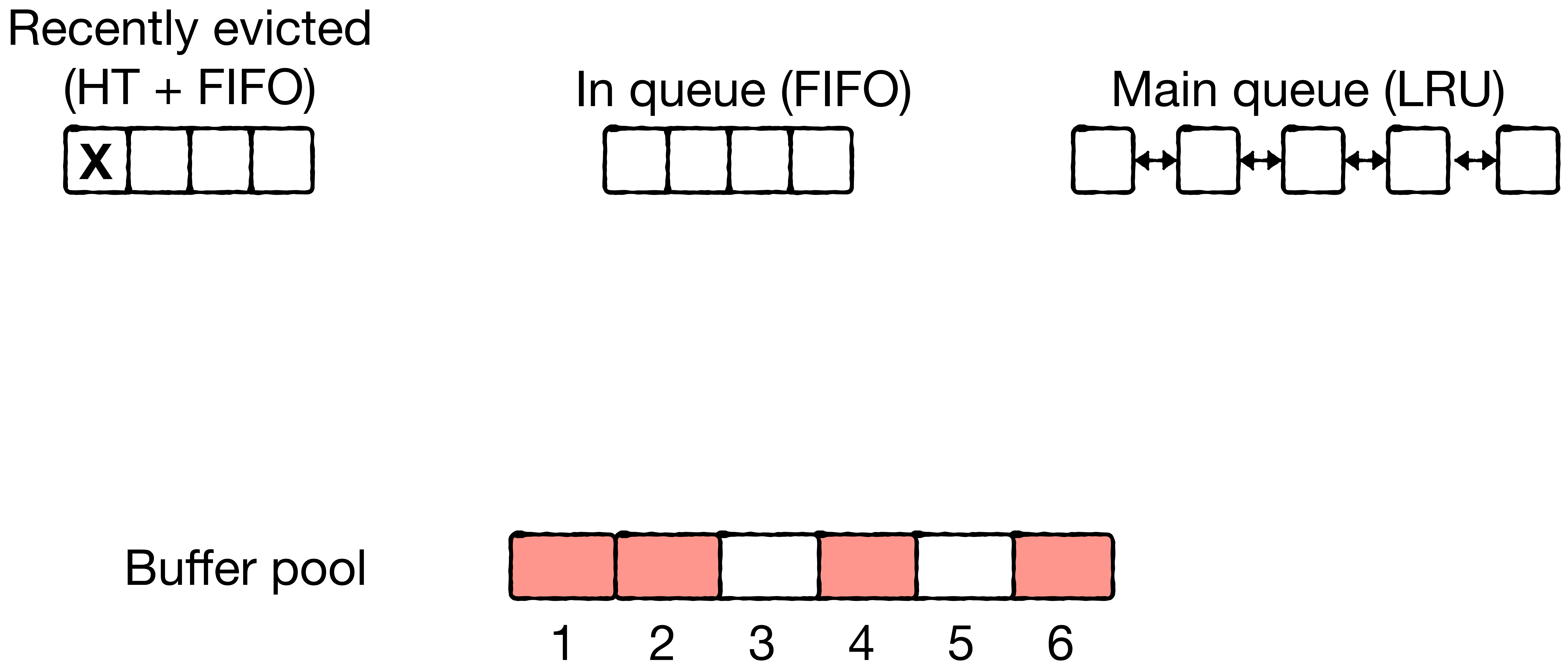

Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Check for X,
doesn't exist
Buffer pool
1
2
3
4
5
6
Insert page X

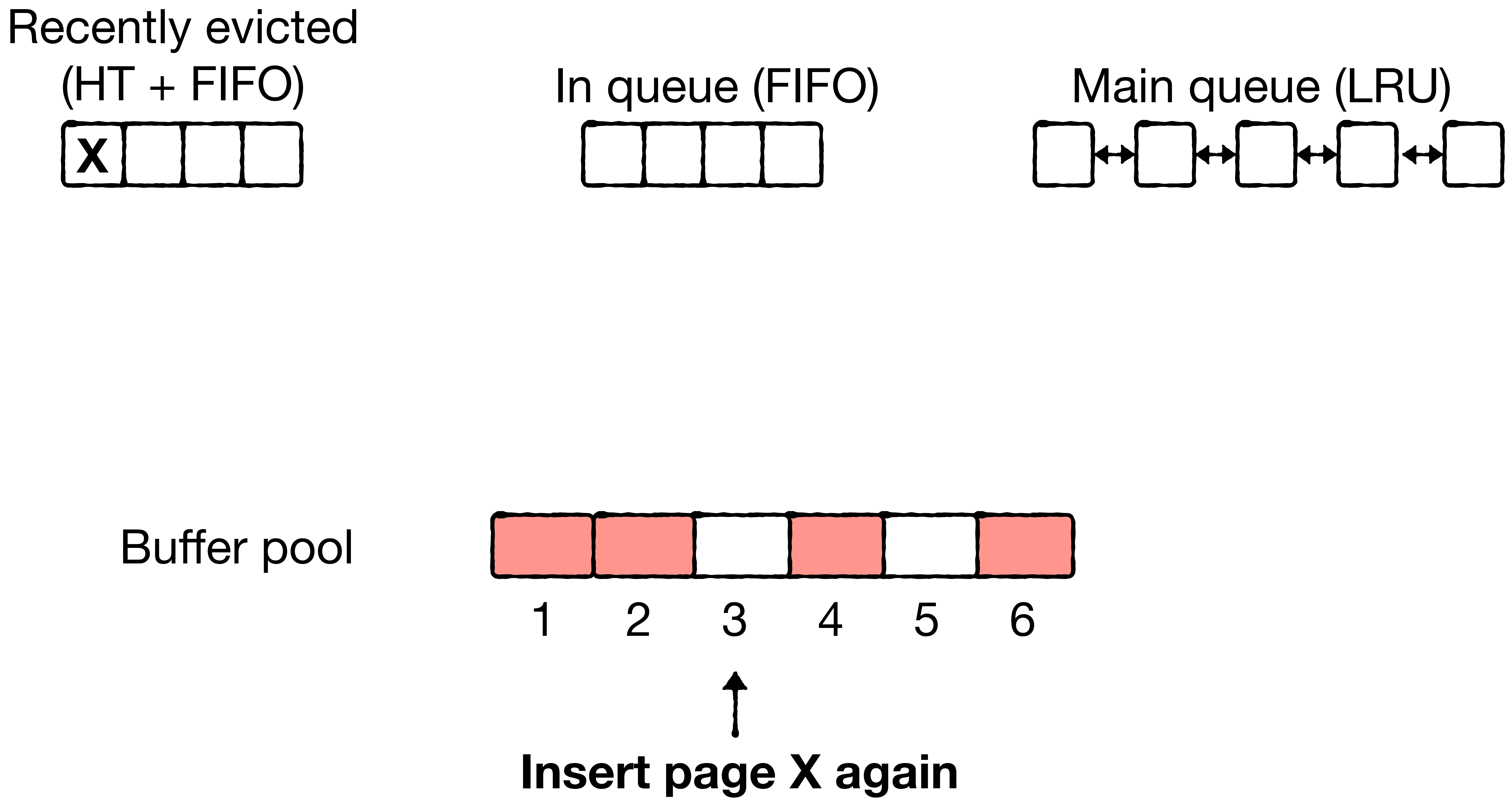

Recently evicted (HT + FIFO)
In queue (FIFO)
X
Main queue (LRU)
Insert X
Buffer pool
1
2
3
4
5
6
Insert page X

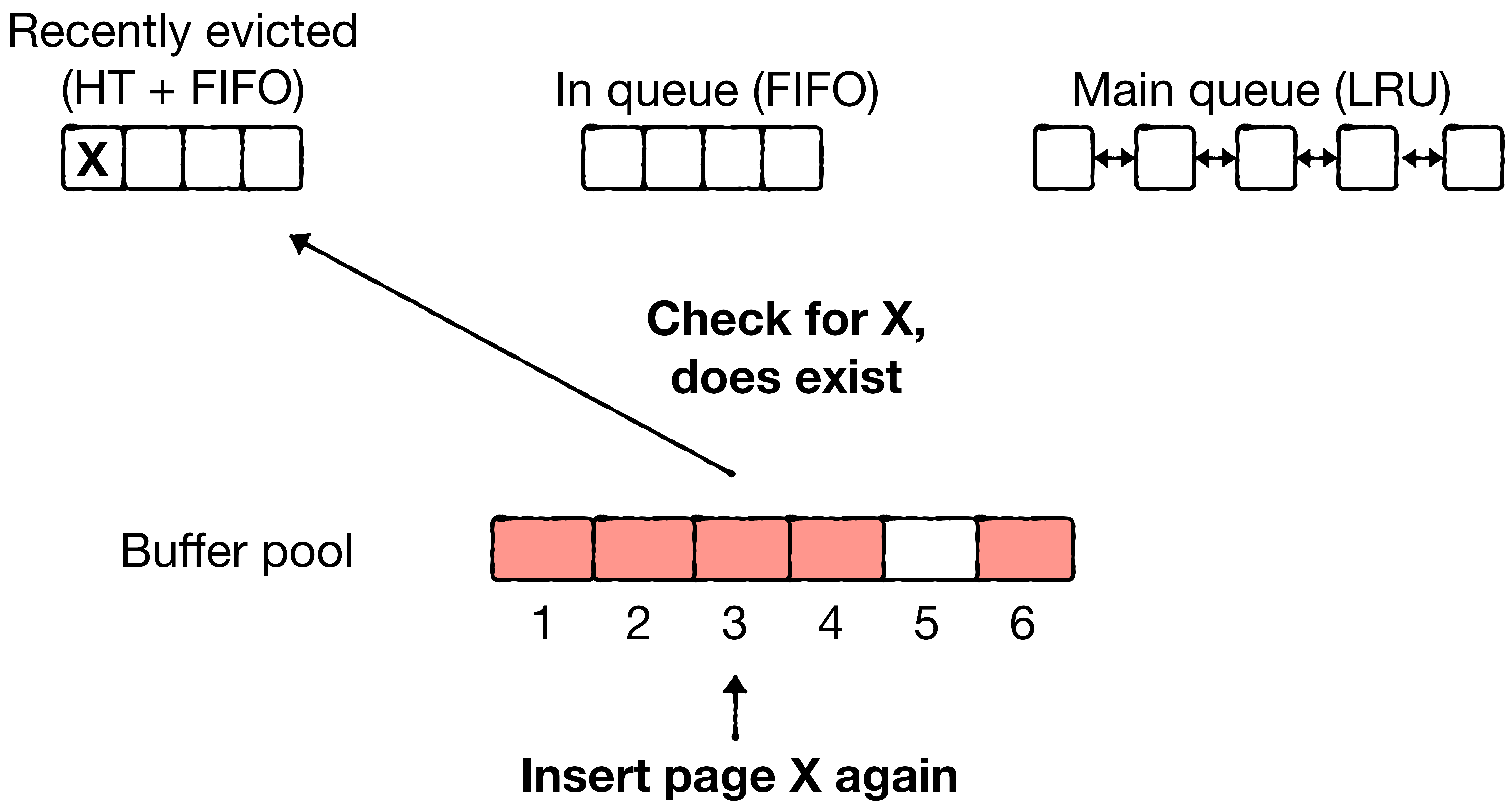

Recently evicted
(HT + FIFO)
In queue (FIFO)
X
Main queue (LRU)
Do not promote (could
be a temporary spurt)
Buffer pool
1
2
3
4
5
6
Access

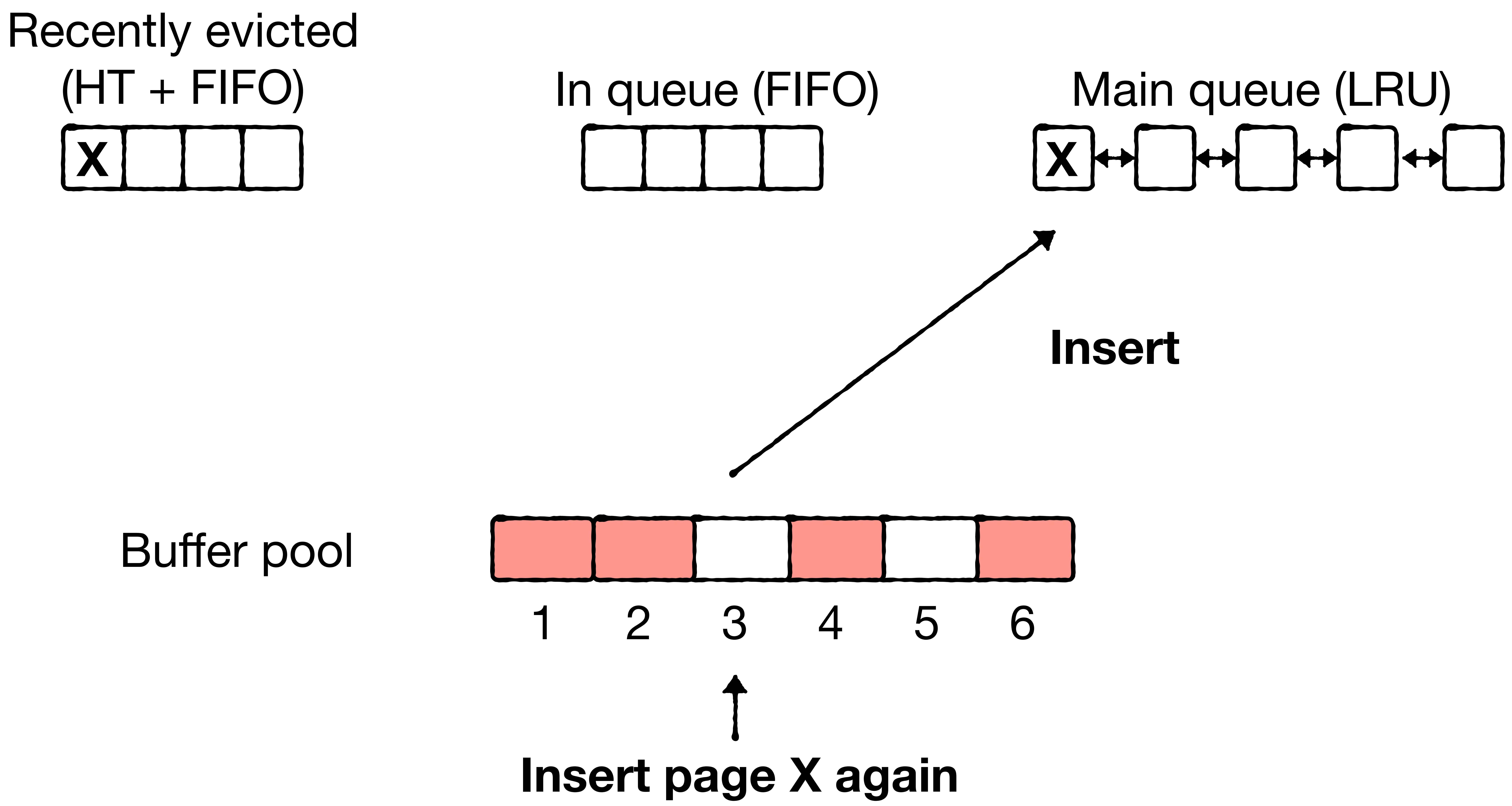

Recently evicted (HT + FIFO)
In queue (FIFO)
X
Main queue (LRU)
Evict eventually
Buffer pool
1
2
3
4
5
6

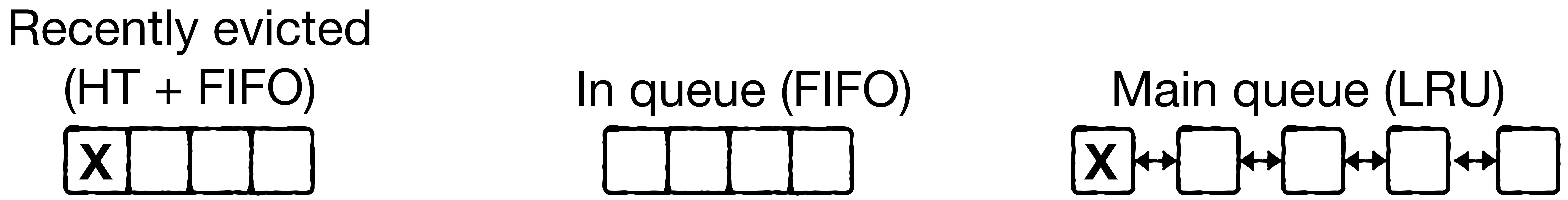

Recently evicted
(HT + FIFO)
X
In queue (FIFO)
X
Main queue (LRU)
Evict eventually
Buffer pool
1
2
3
4
5
6


Recently evicted
(HT + FIFO)
X
In queue (FIFO)
Main queue (LRU)
Buffer pool
1
2
3
4
5
6

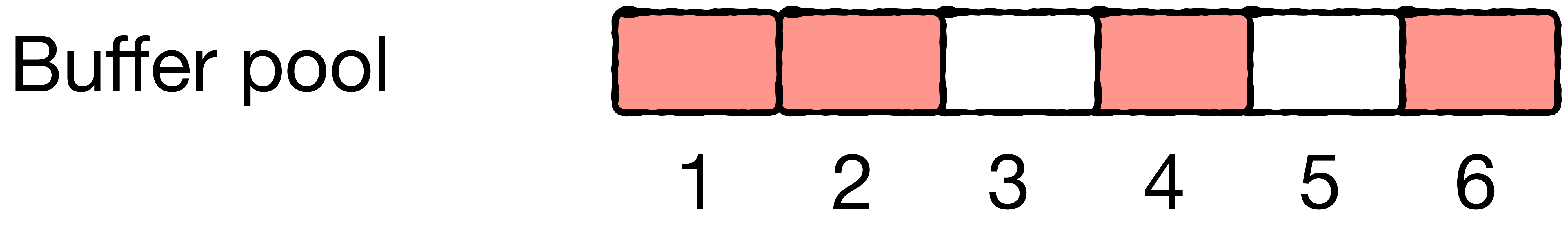

Recently evicted
(HT + FIFO)
X
In queue (FIFO)
Main queue (LRU)
Buffer pool
1
2
3
4
5
6
Insert page X again

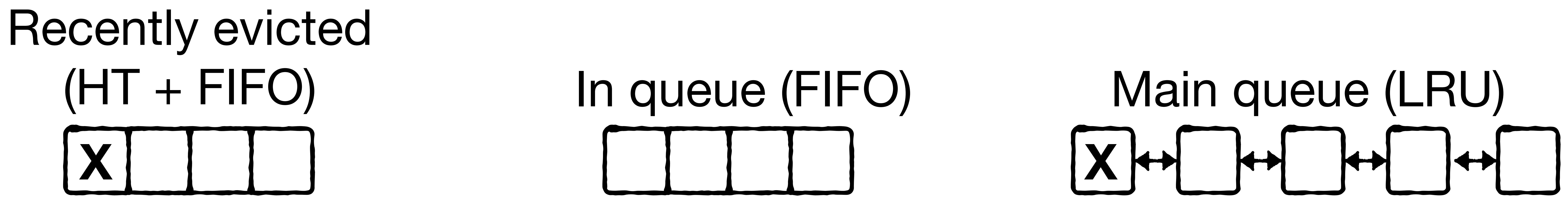

Recently evicted (HT + FIFO)
X
In queue (FIFO)
Main queue (LRU)
Check for X, does exist
Buffer pool
1
2
3
4
5
6
Insert page X again


Recently evicted (HT + FIFO)
X
In queue (FIFO)
Main queue (LRU)
X
Insert
Buffer pool
1
2
3
4
5
6
Insert page X again

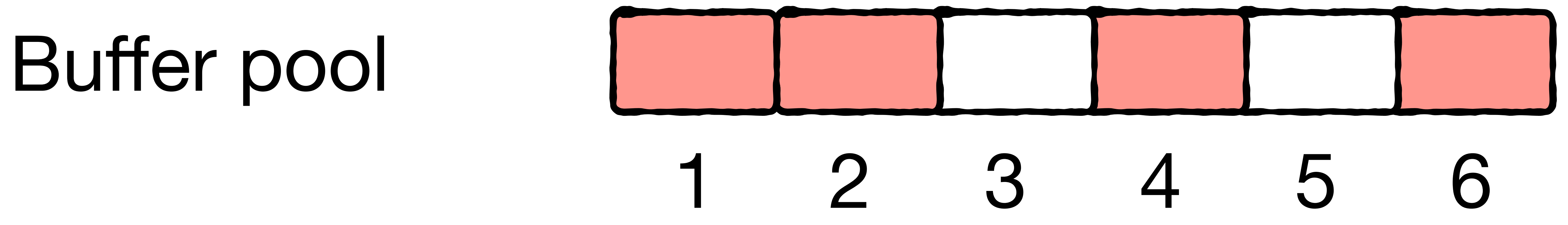


pages in main queue have had at least 2 recent accesses, so 2Q approximates LRU/K

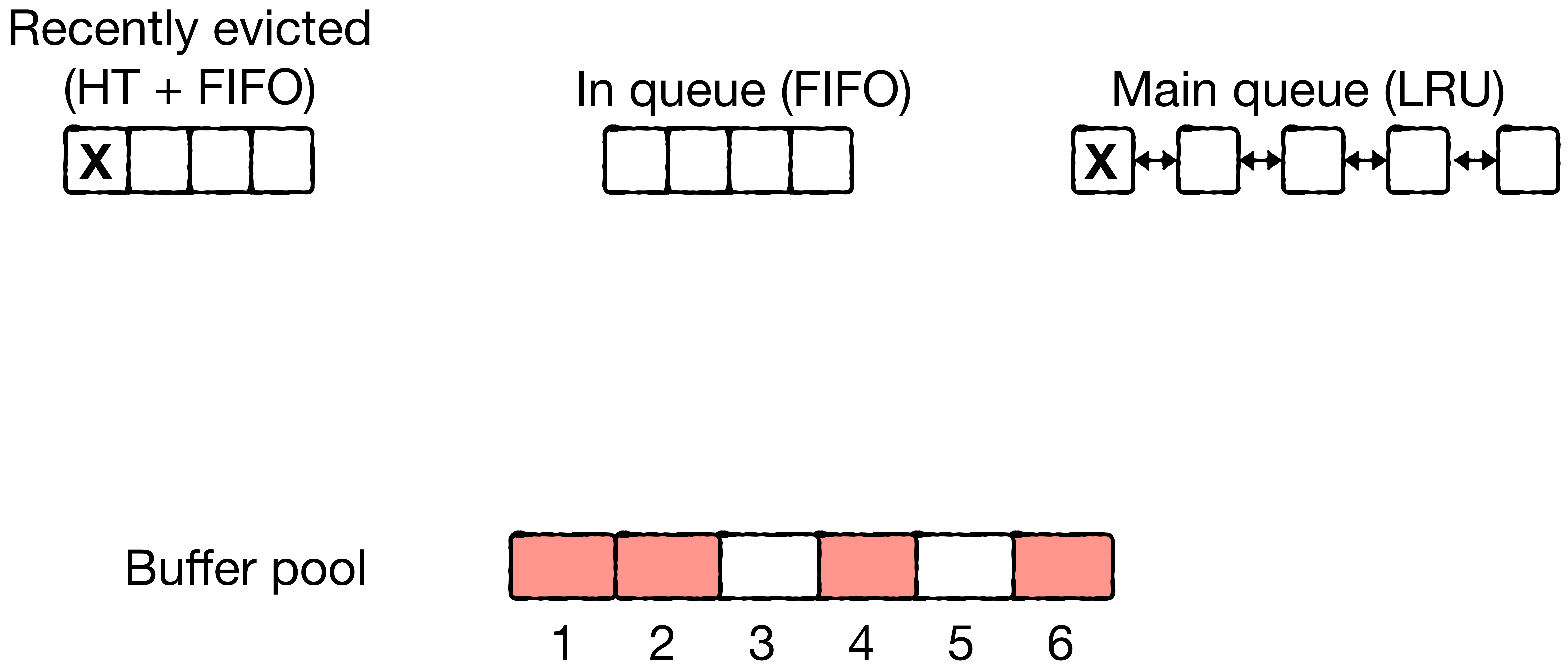


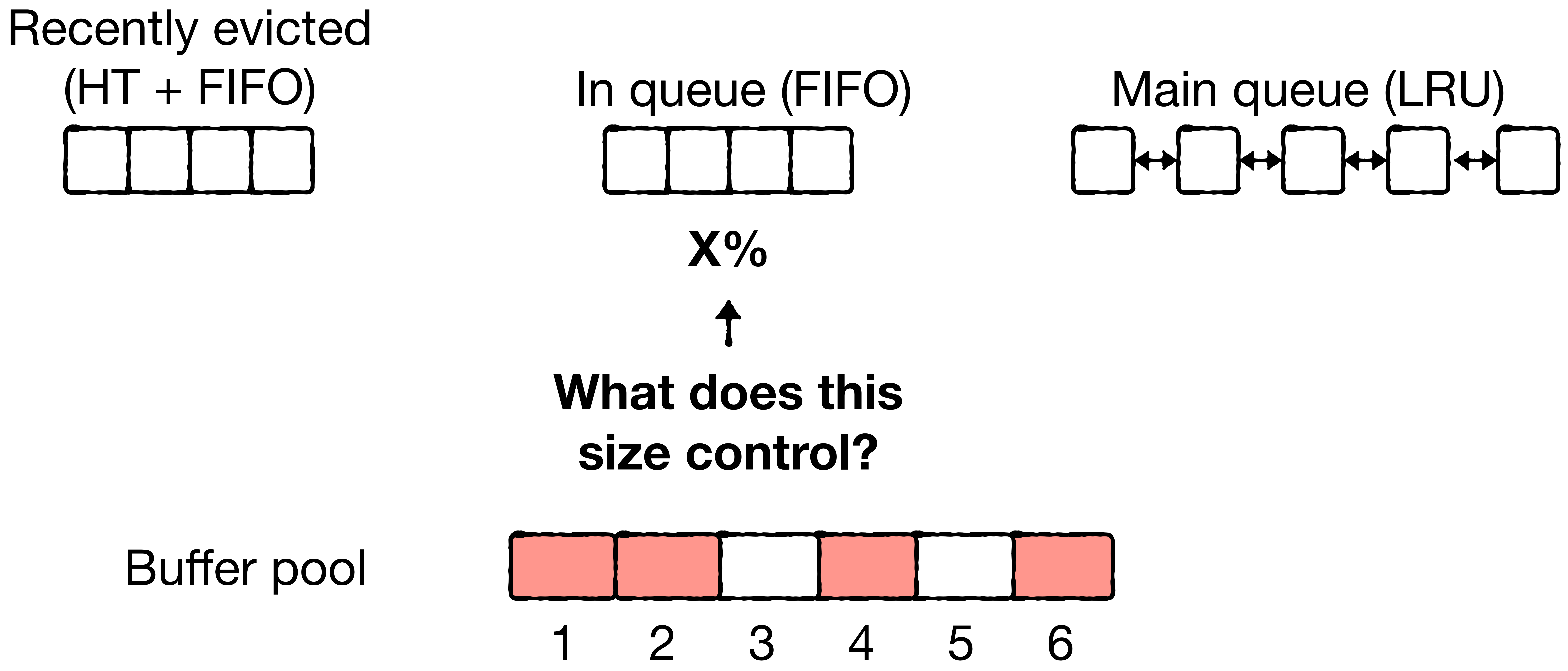


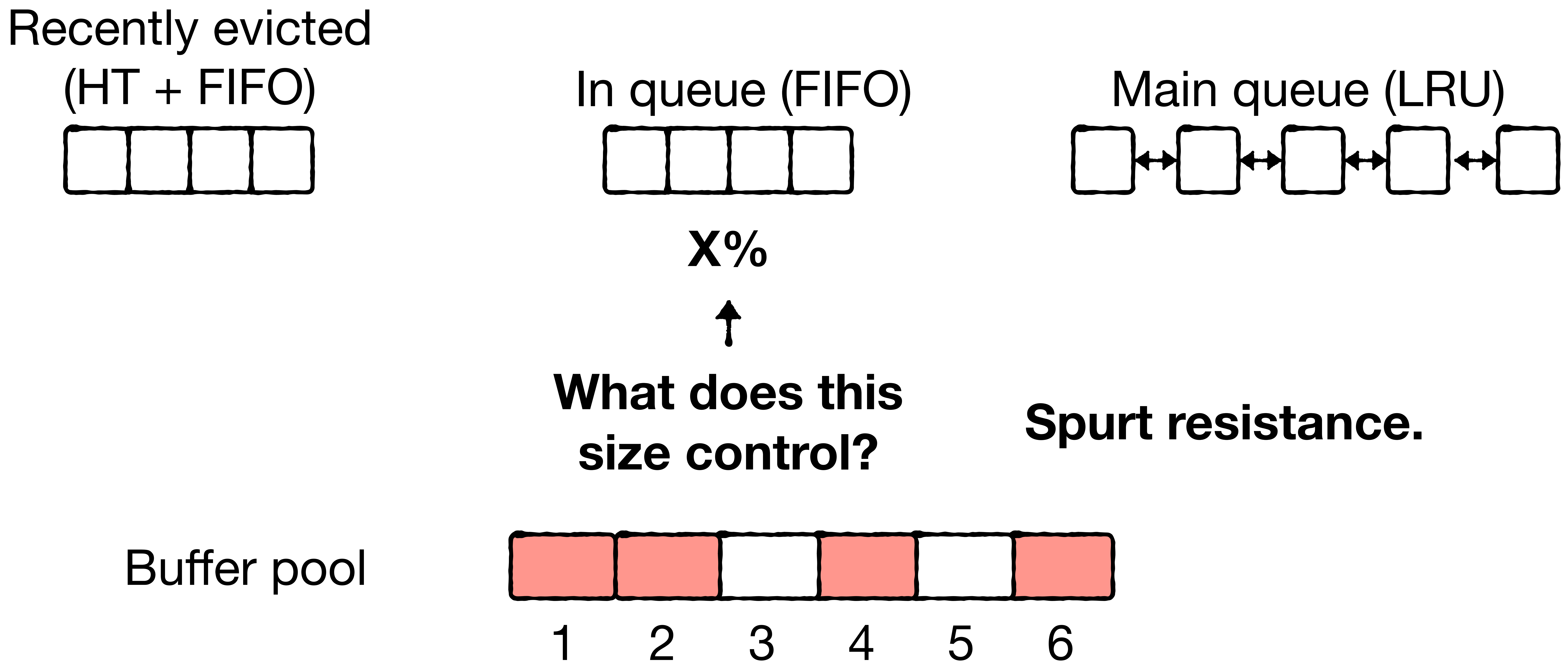
Each is a O(1) data structure



Problems?

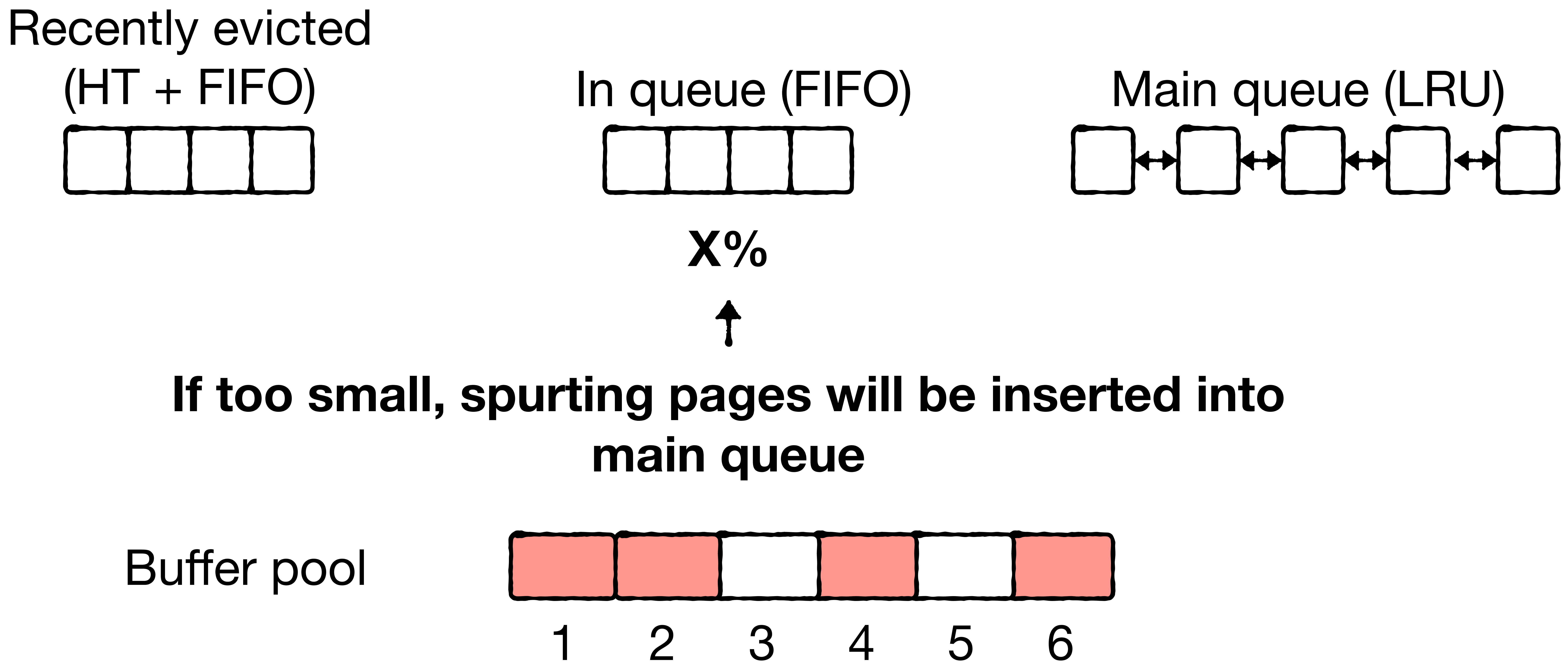

Recently evicted
(HT + FIFO)
X
In queue (FIFO)
Main queue (LRU)
X
Buffer pool
1
2
3
4
5
6


Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)
X%
What does this
size control?
Buffer pool
1
2
3
4
5
6

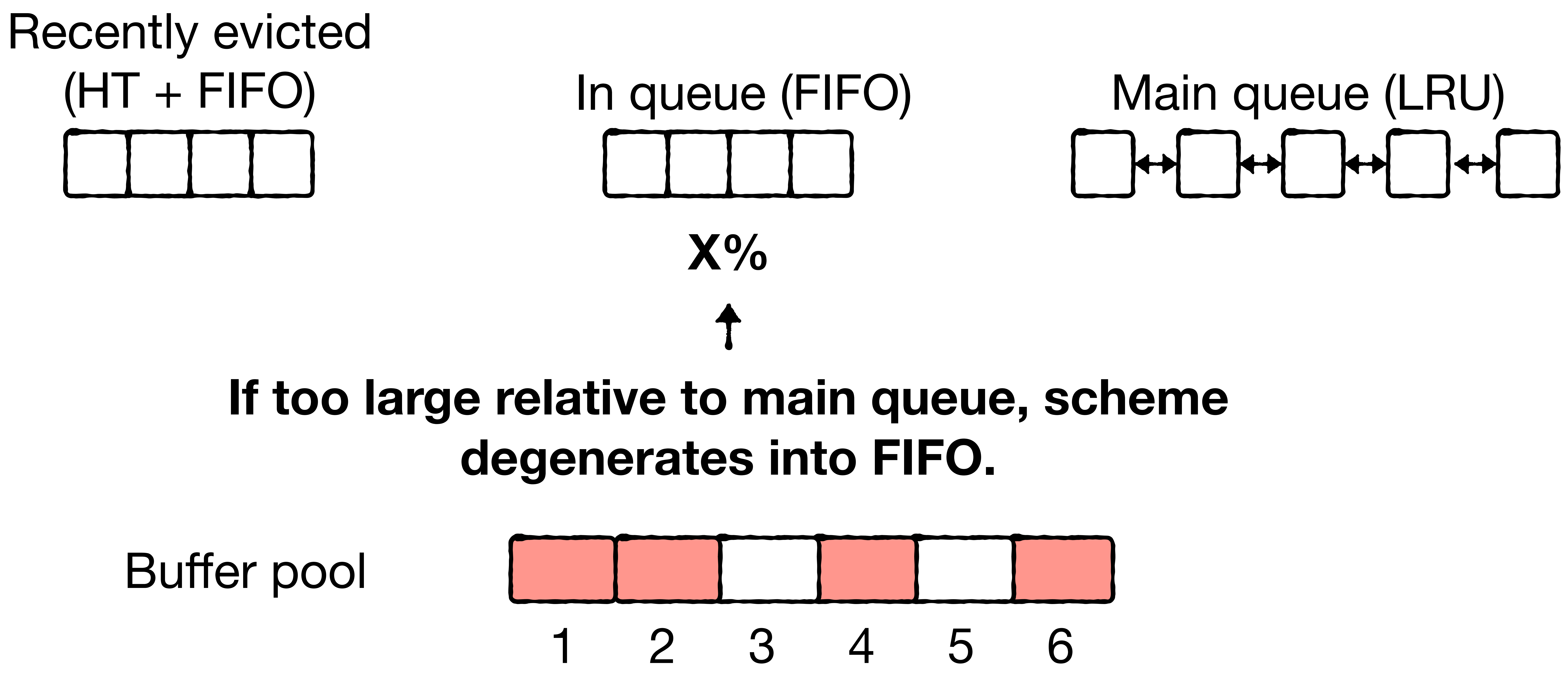


What does this size control?

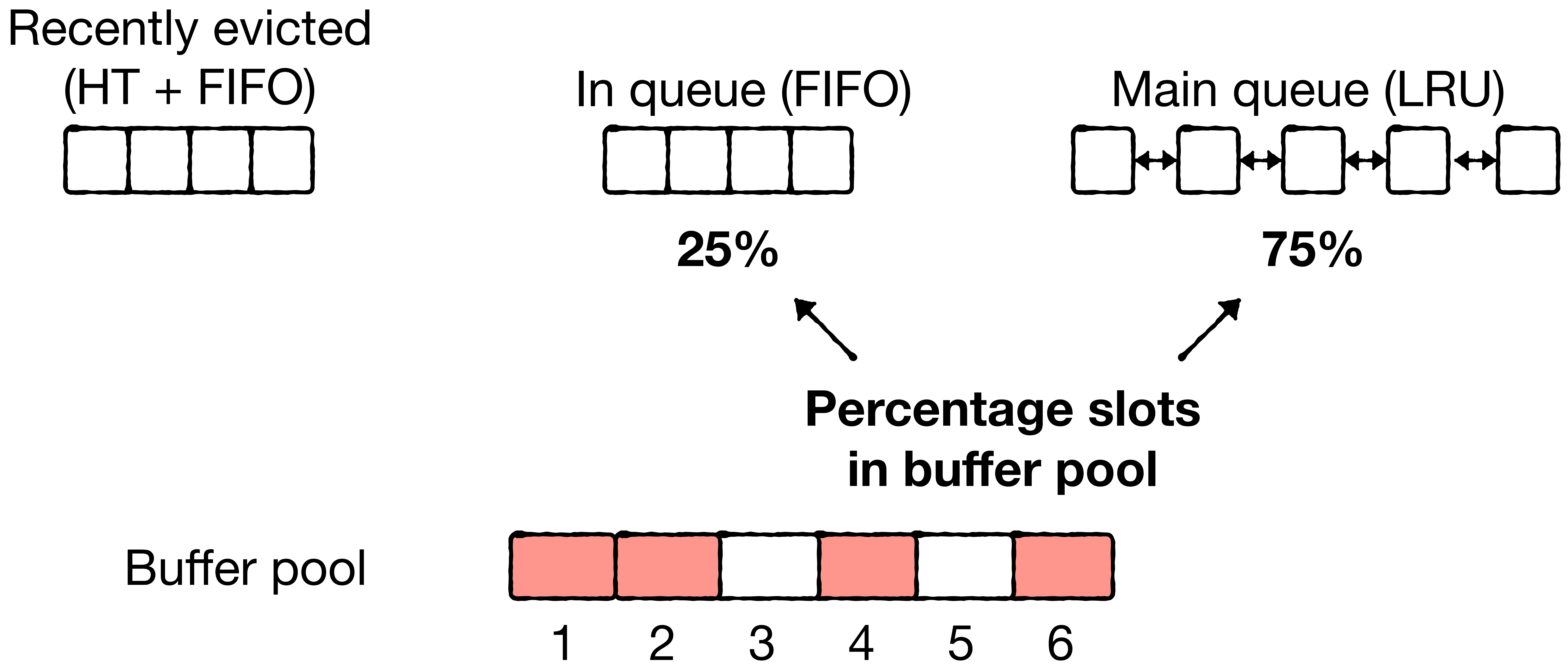
Spurt resistance.



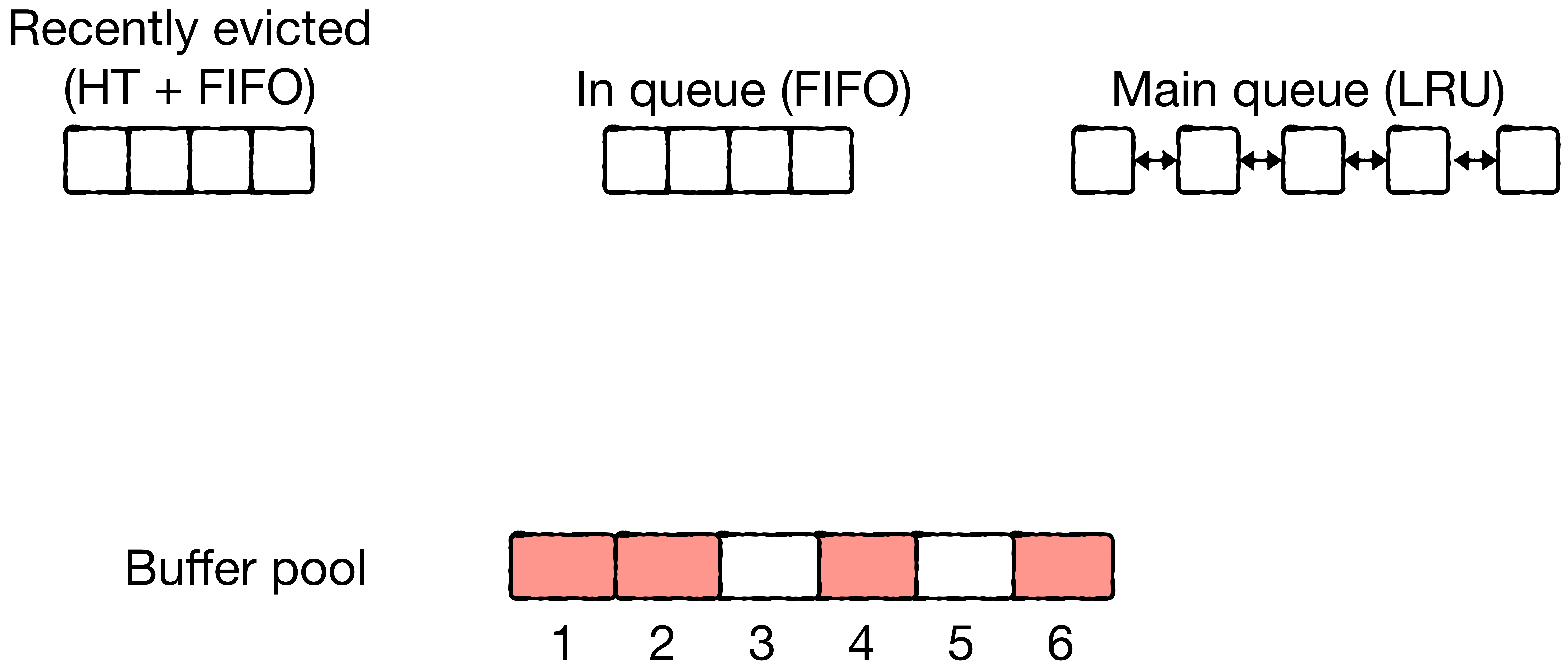
If too small, spurting pages will be inserted into main queue

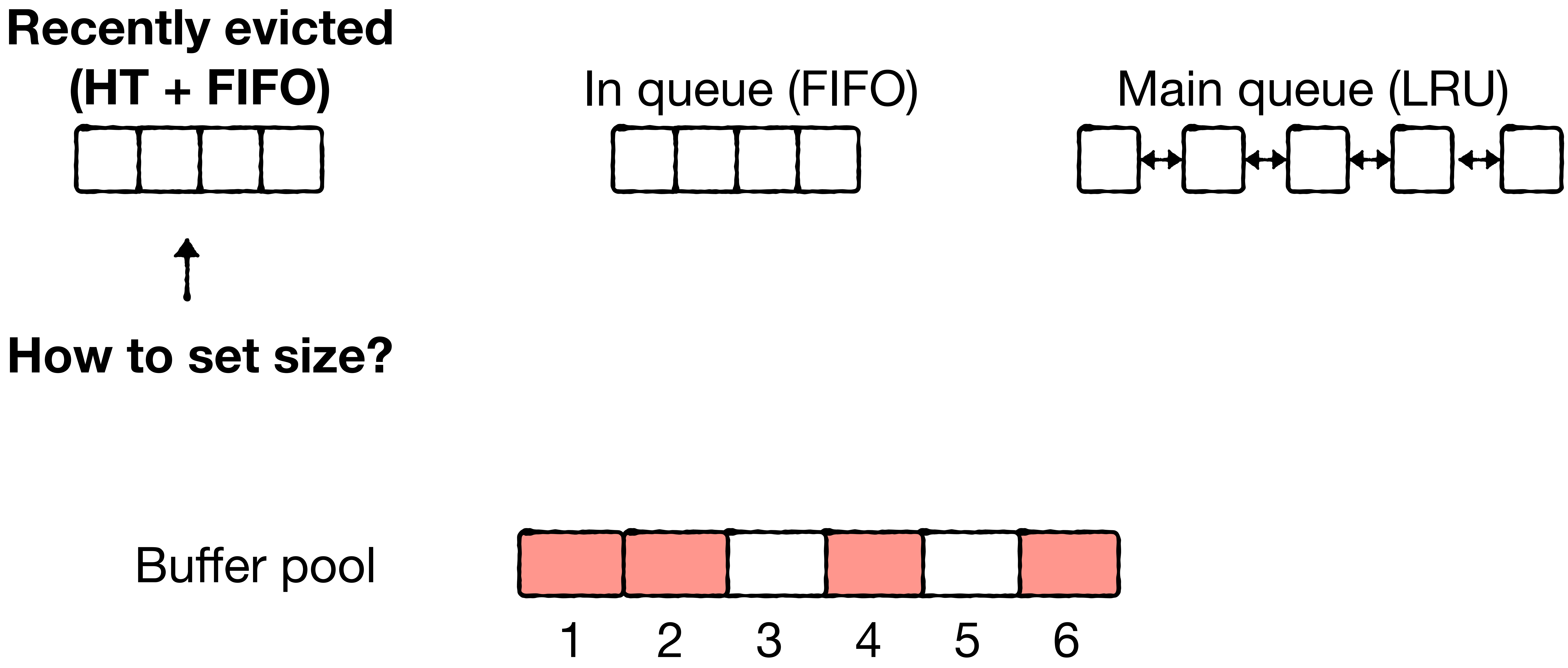


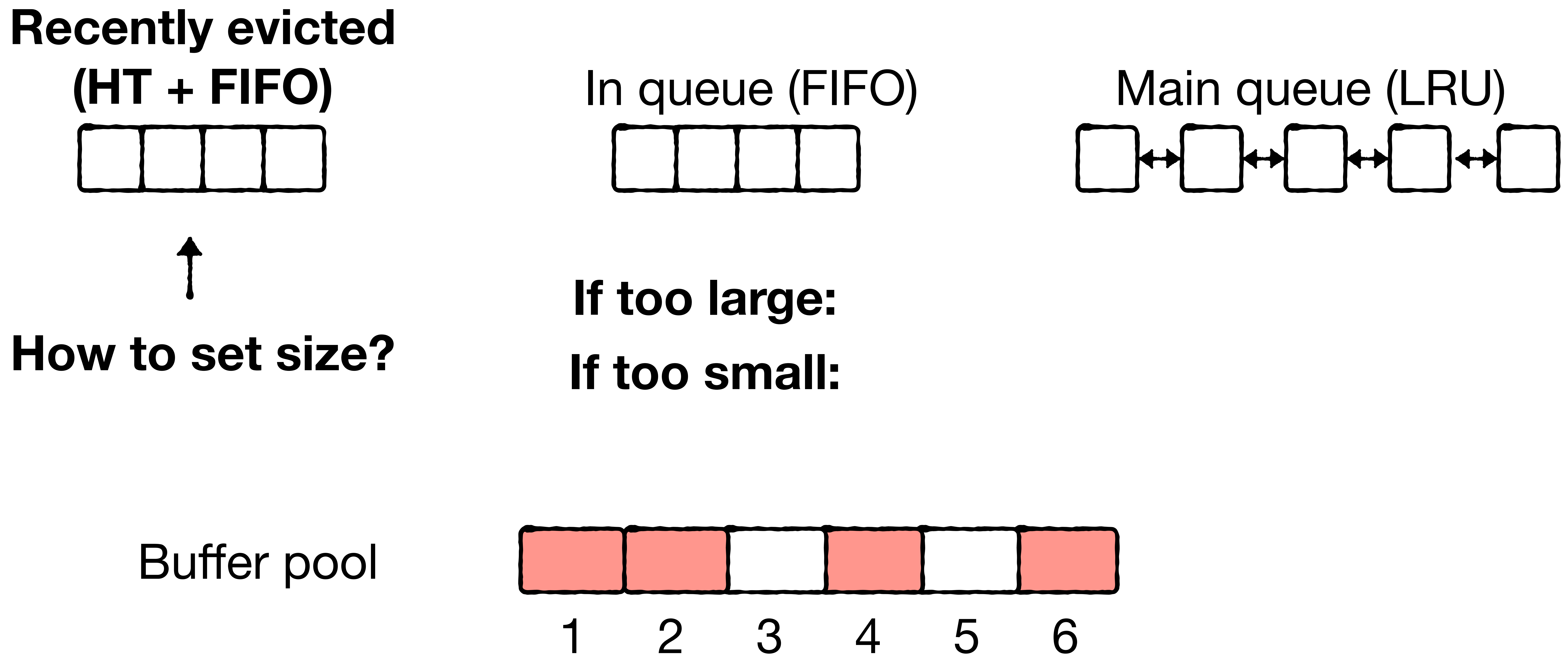
If too large relative to main queue, scheme degenerates into FIFO.


Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
25%
75%
Percentage slots in buffer pool
Buffer pool
1
2
3
4
5
6

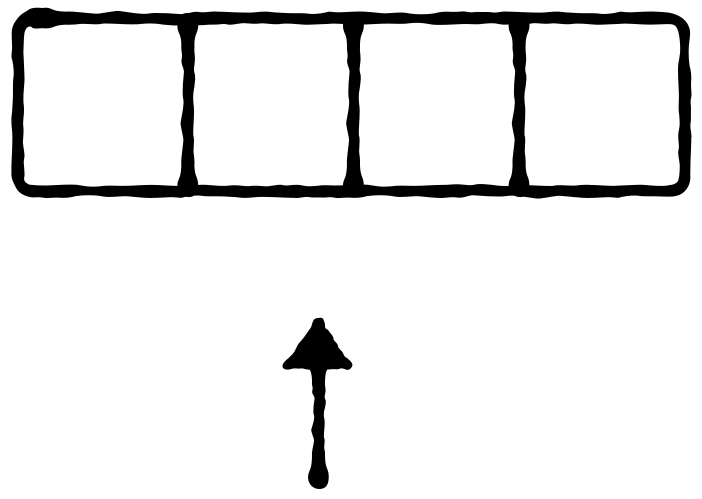
Other problems?




Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
How to set size?
Buffer pool
1
2
3
4
5
6

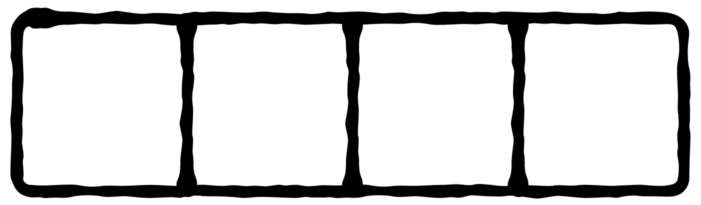

Recently evicted (HT + FIFO)
In queue (FIFO)
Main queue (LRU)
How to set size?
If too large:
If too small:
Buffer pool
1
2
3
4
5
6

Recently evicted (HT + FIFO)

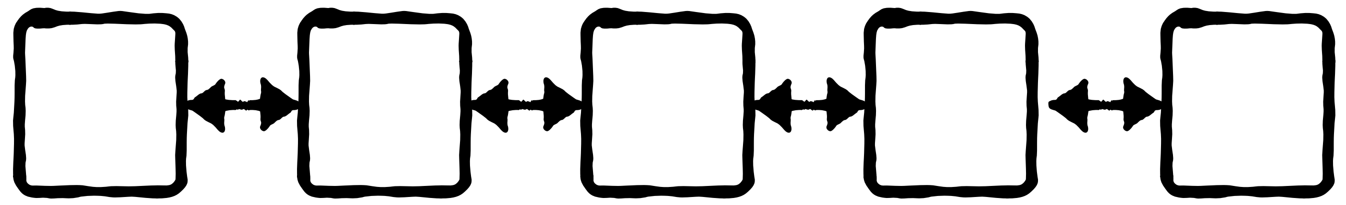


How to set size?

In queue (FIFO)



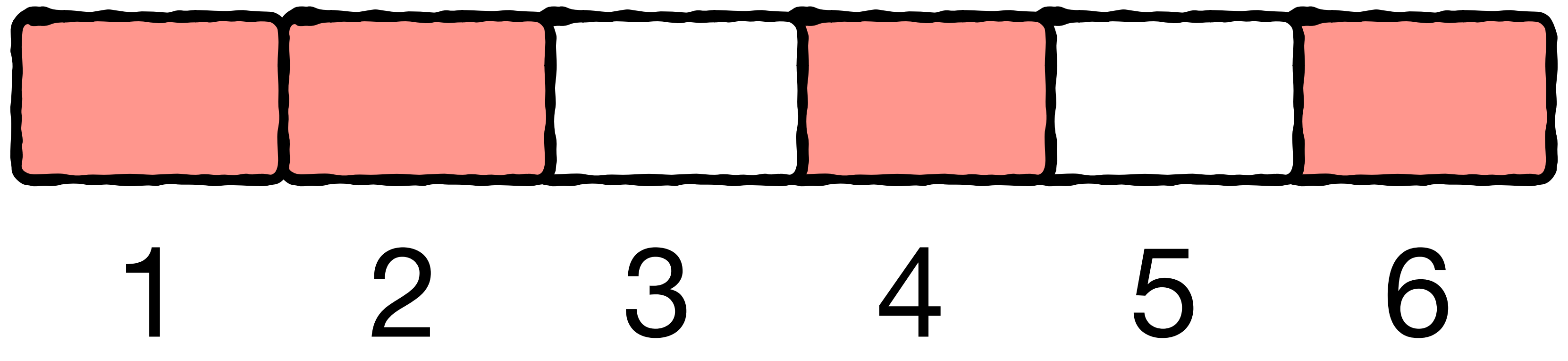
Main queue (LRU)



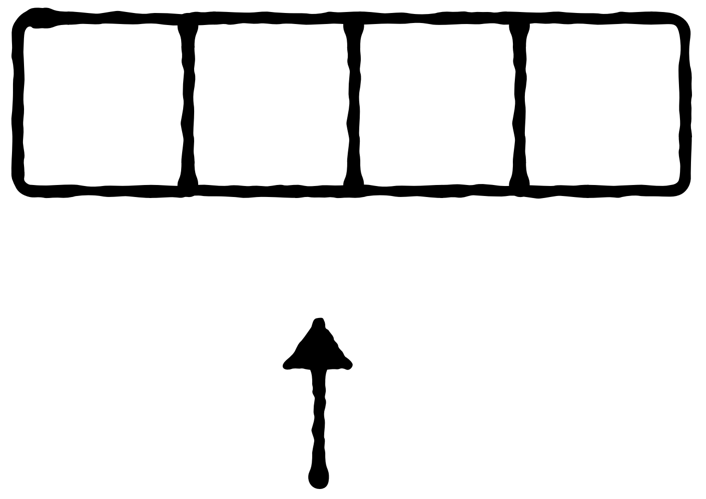
If too large: **degenerate into LRU**

If too small:

Buffer pool

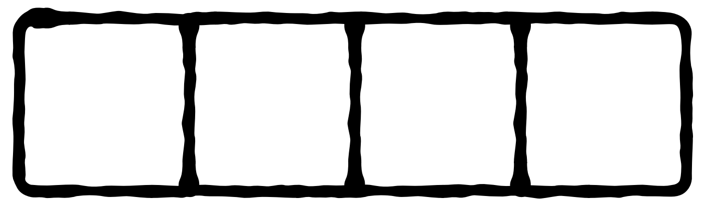


Recently evicted (HT + FIFO)

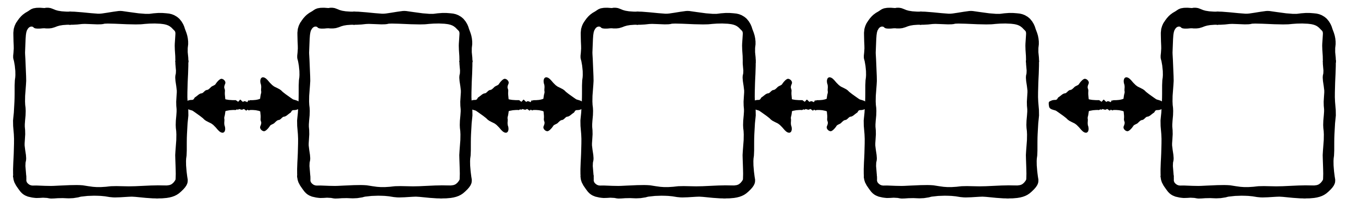


How to set size?

In queue (FIFO)



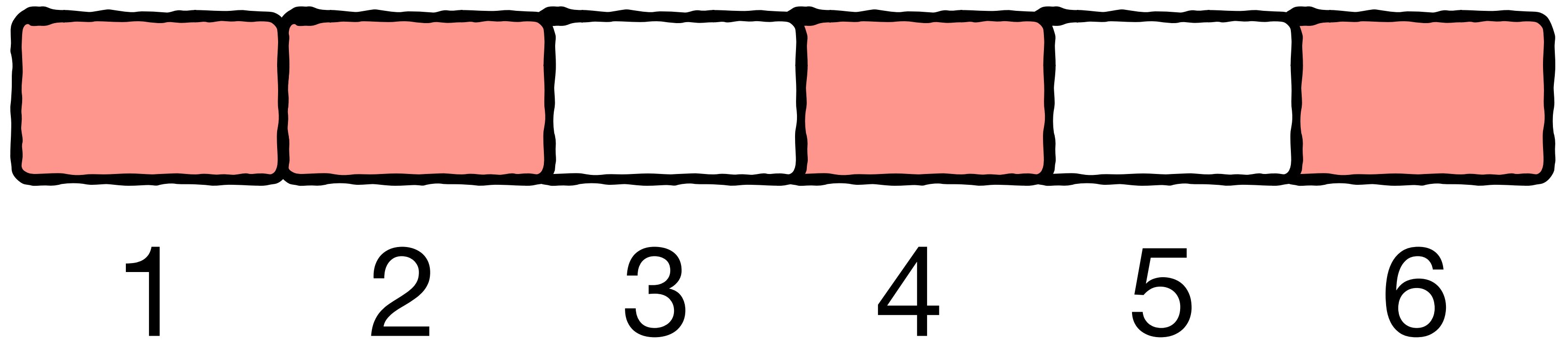
Main queue (LRU)

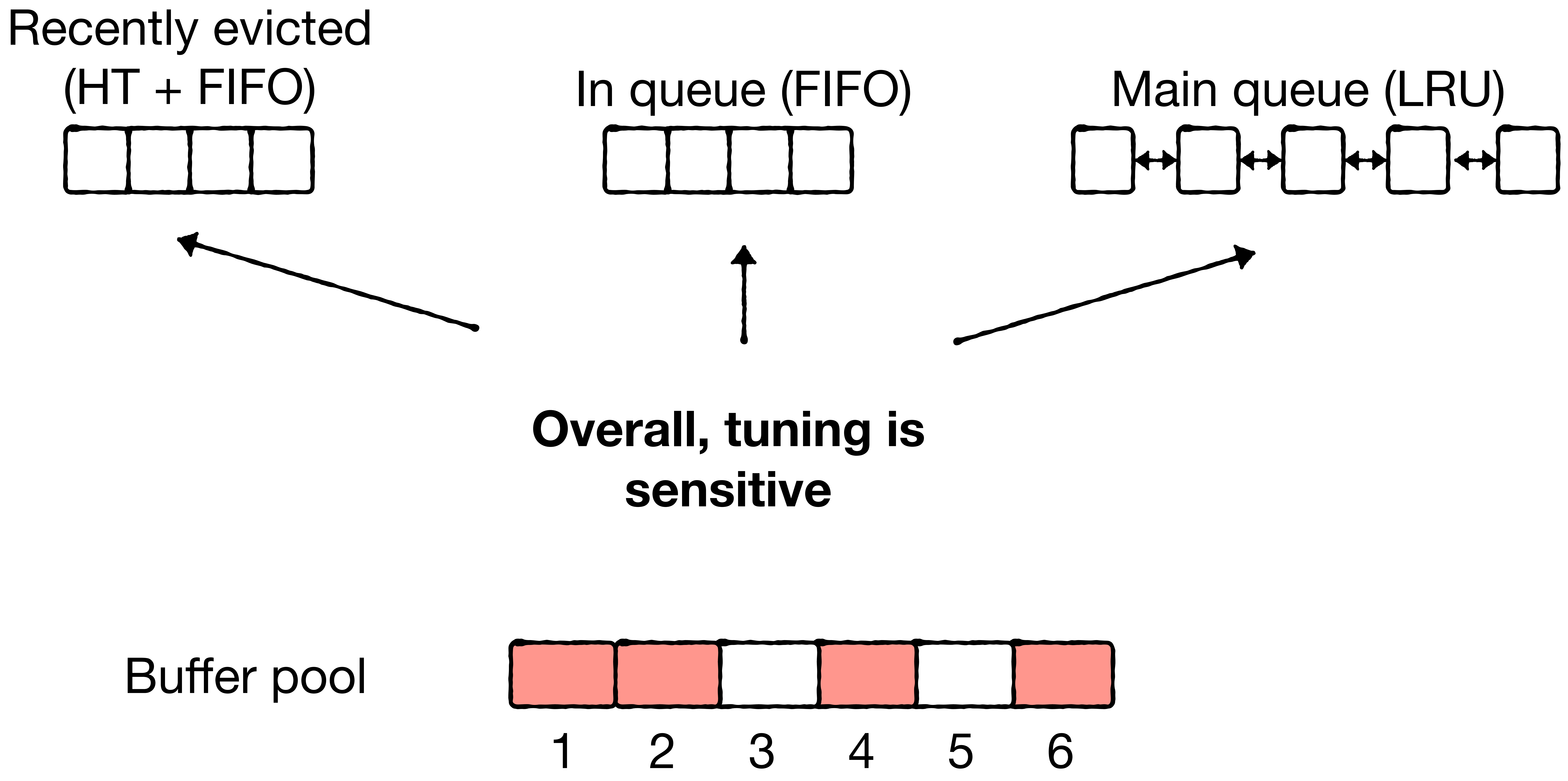


If too large: degenerate into LRU

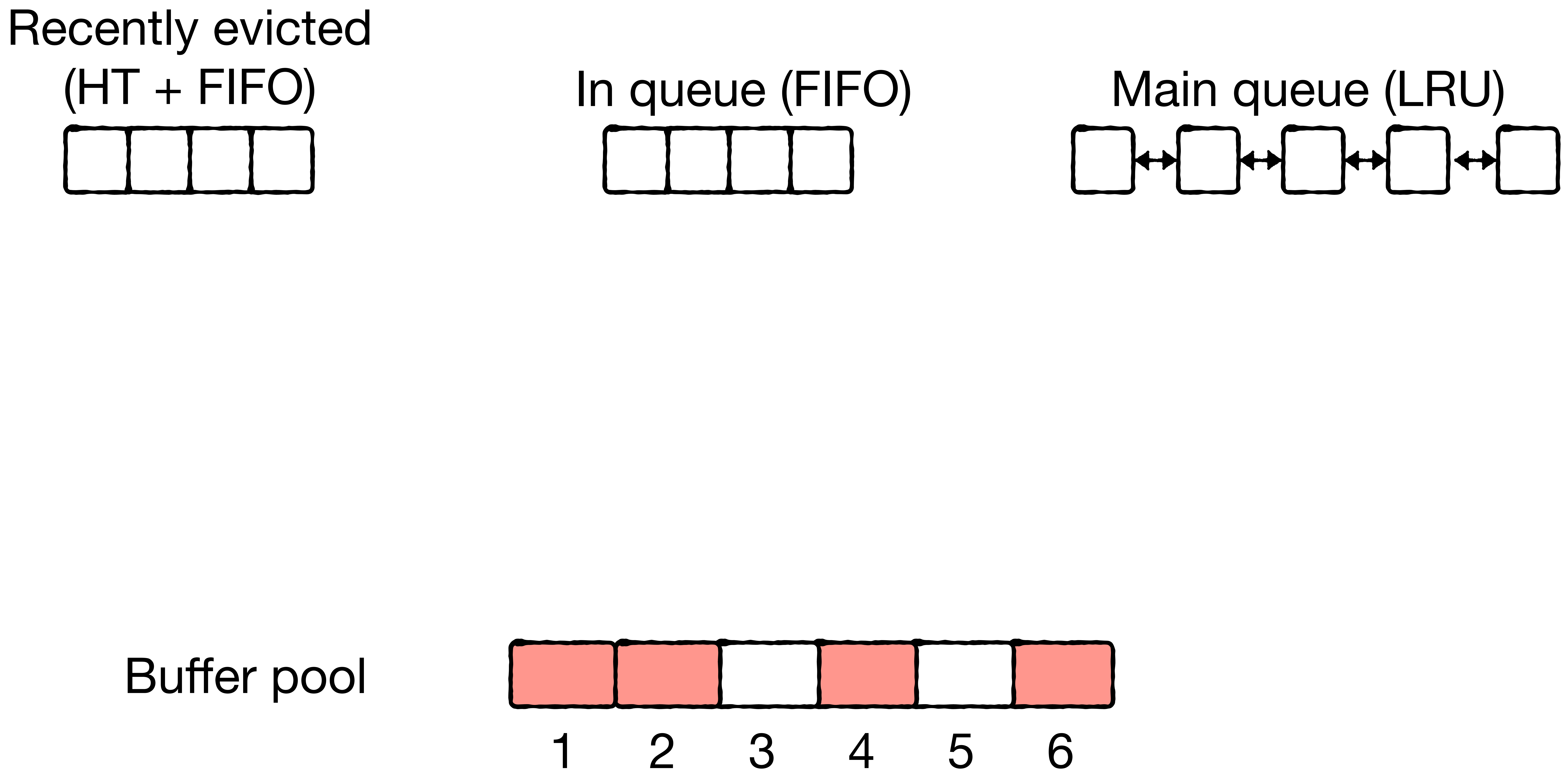
If too small: degenerate into FIFO

Buffer pool

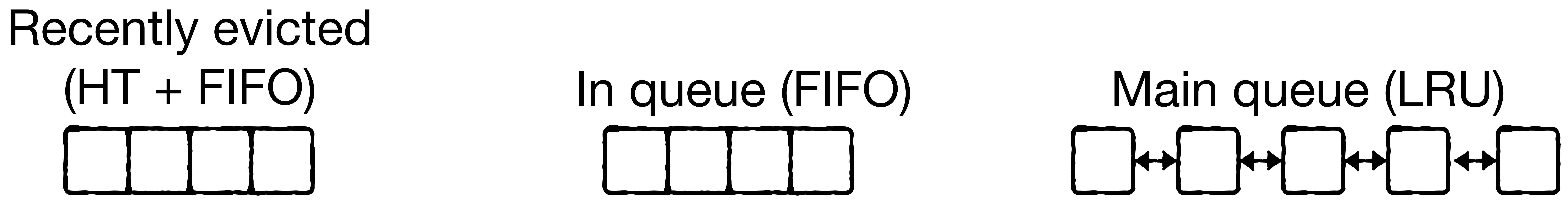



Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)
Overall, tuning is sensitive
Buffer pool
1
2
3
4
5
6

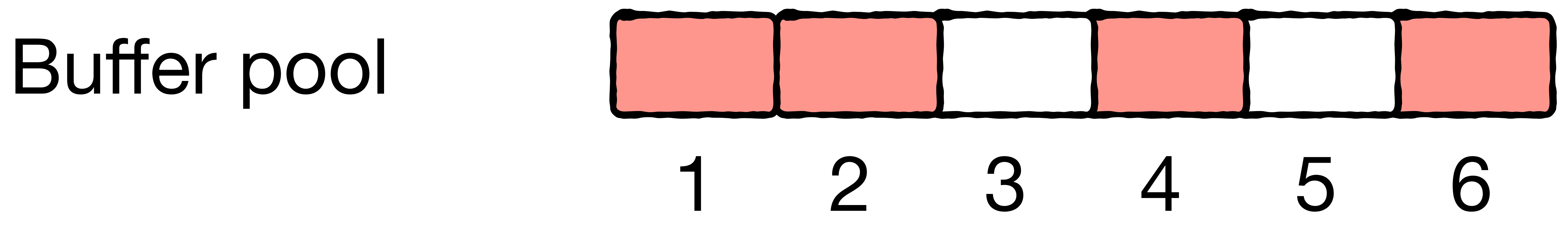
Any other problem?

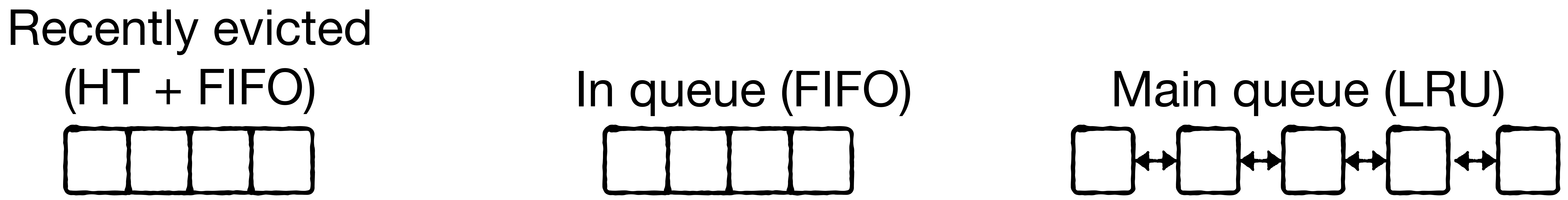


Any other problem?

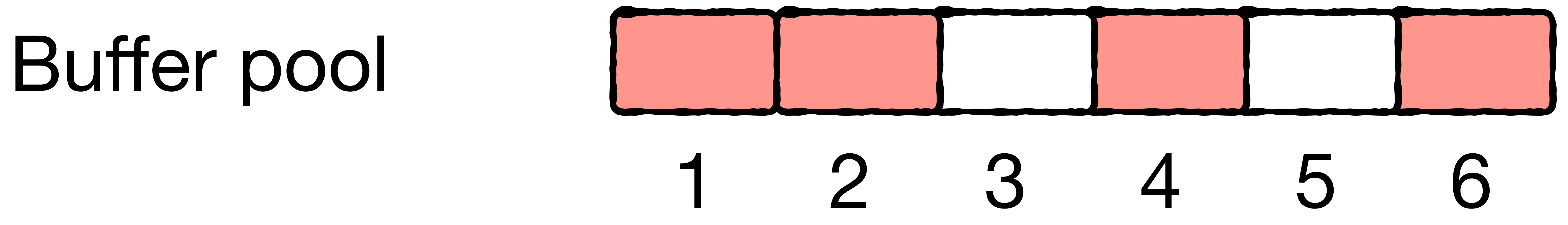


Newly hot page must be evicted & reread once to make it into main queue



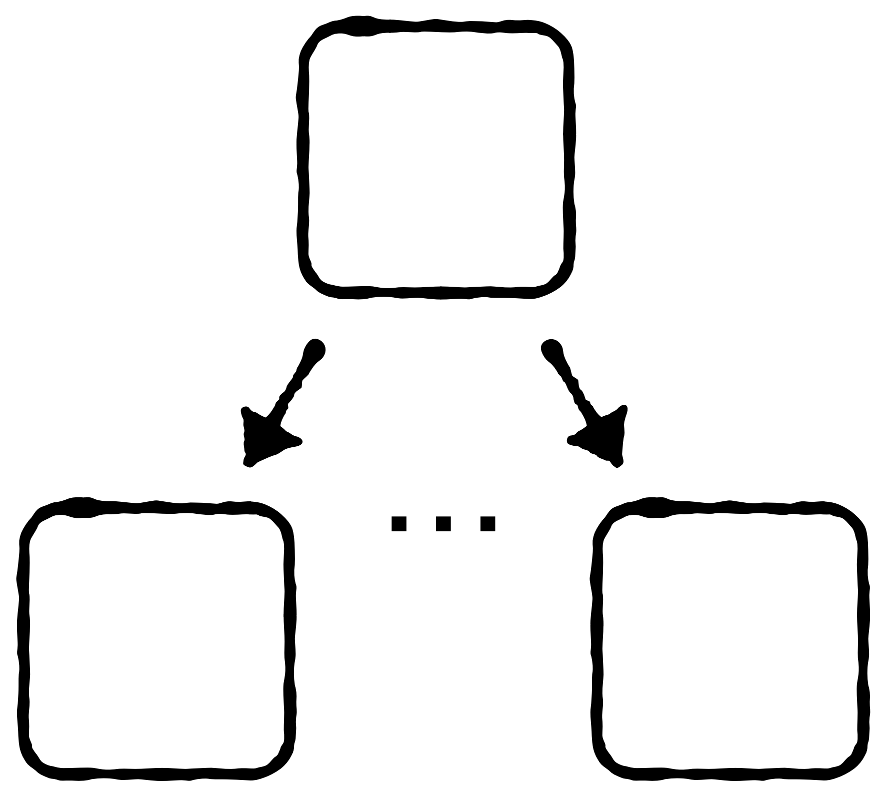

Recently evicted
(HT + FIFO)
In queue (FIFO)
Main queue (LRU)

Questions?


Buffer pool
1
2
3
4
5
6

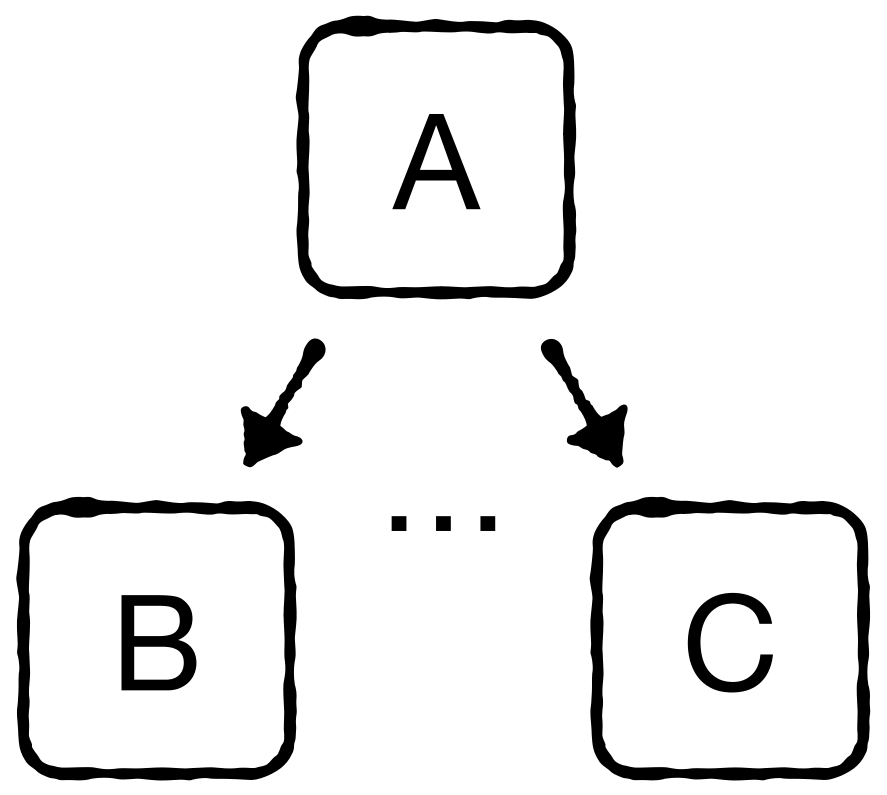
Pointer Swizzling

Pointer Swizzling



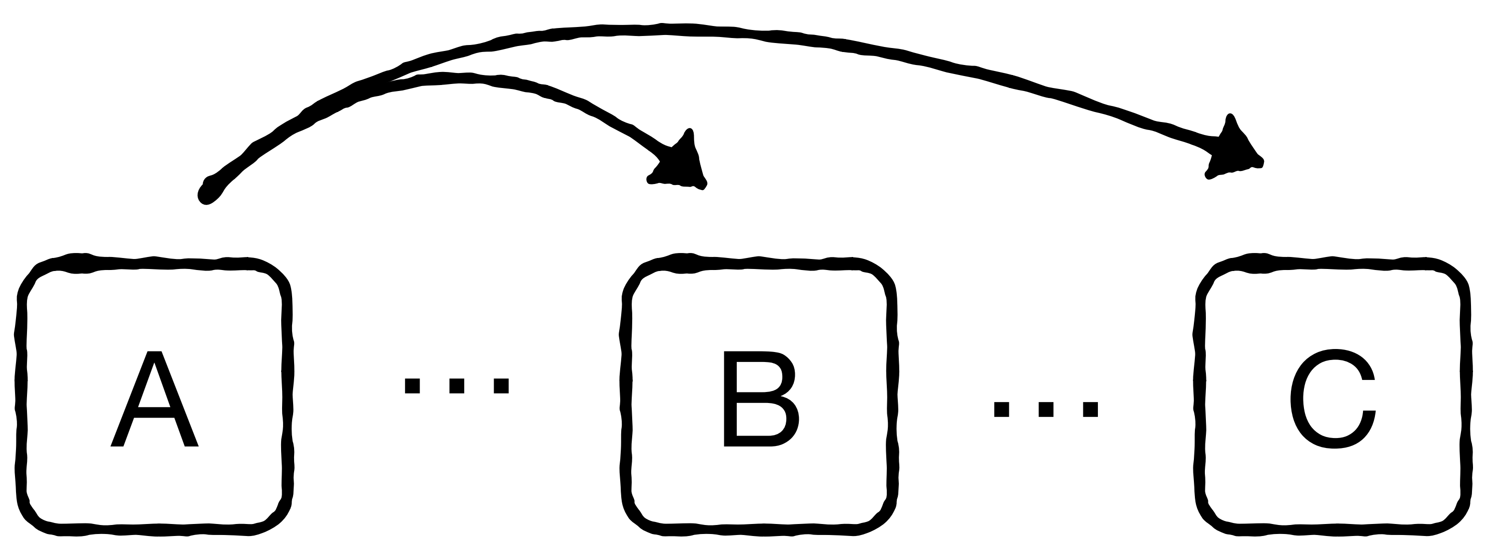
Consider a b-tree as example

Pointer Swizzling



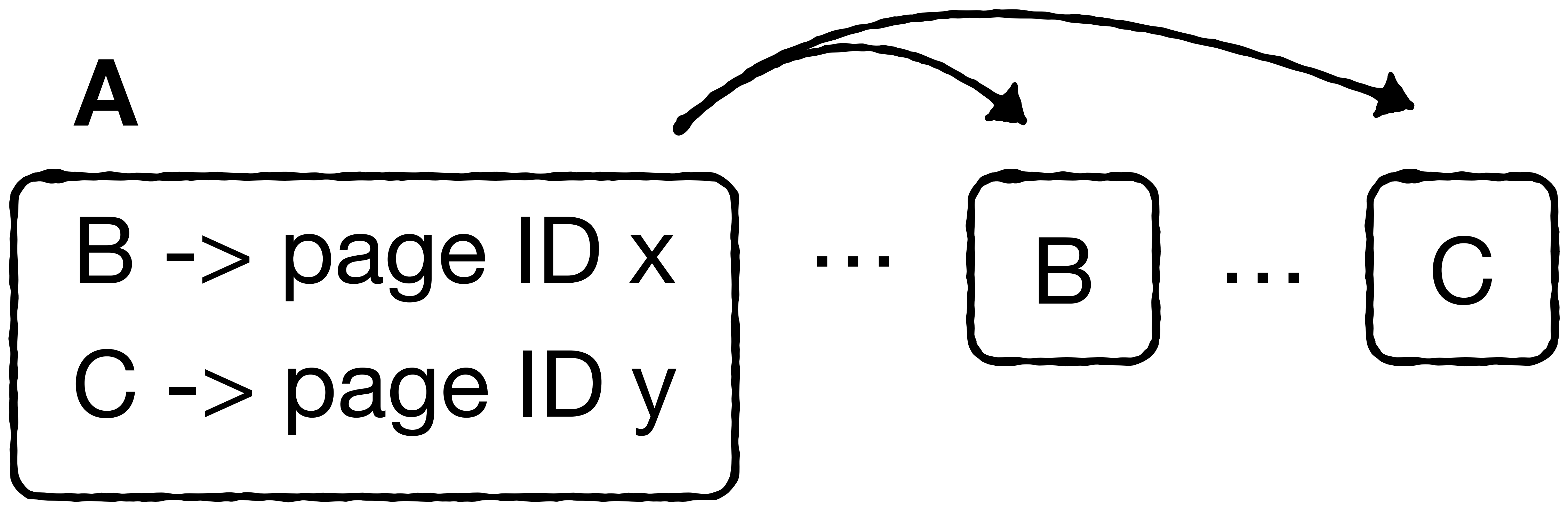
Consider a b-tree as example

Pointer Swizzling

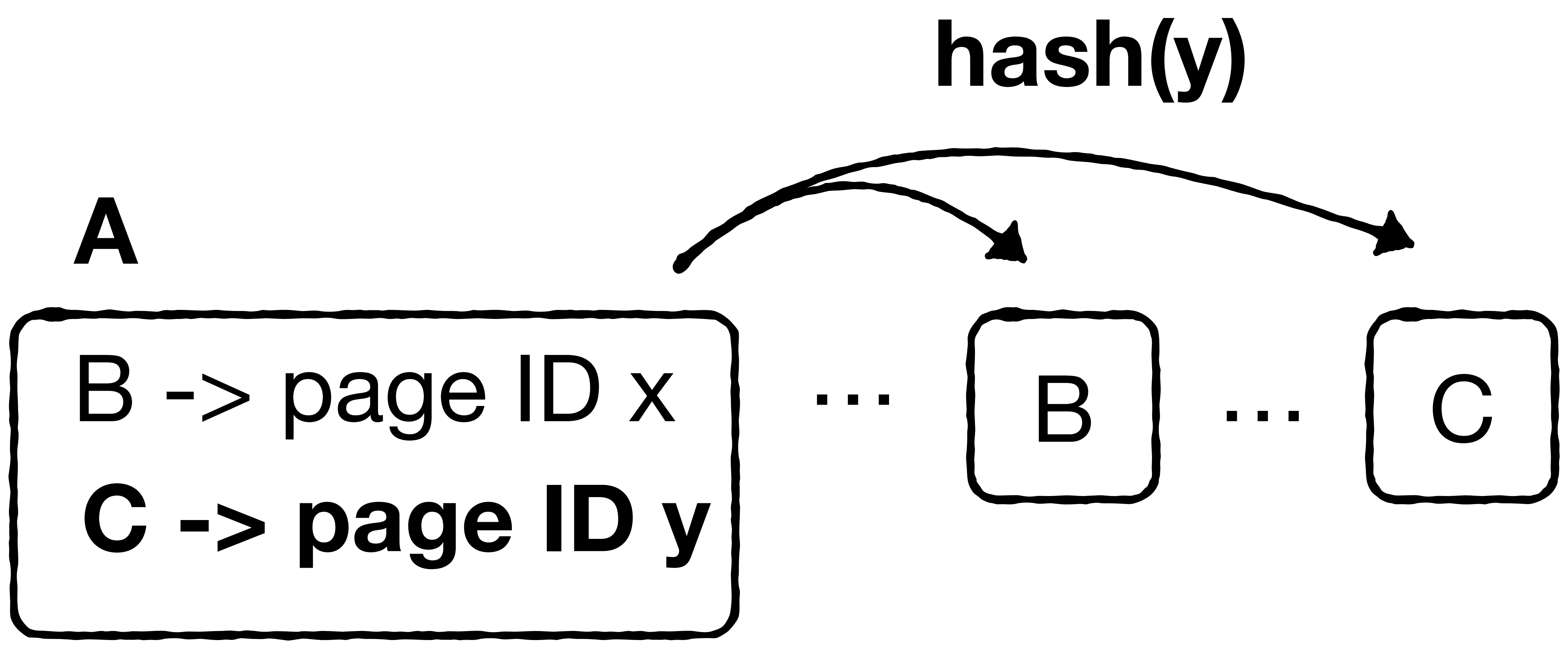


Suppose they are all in buffer pool

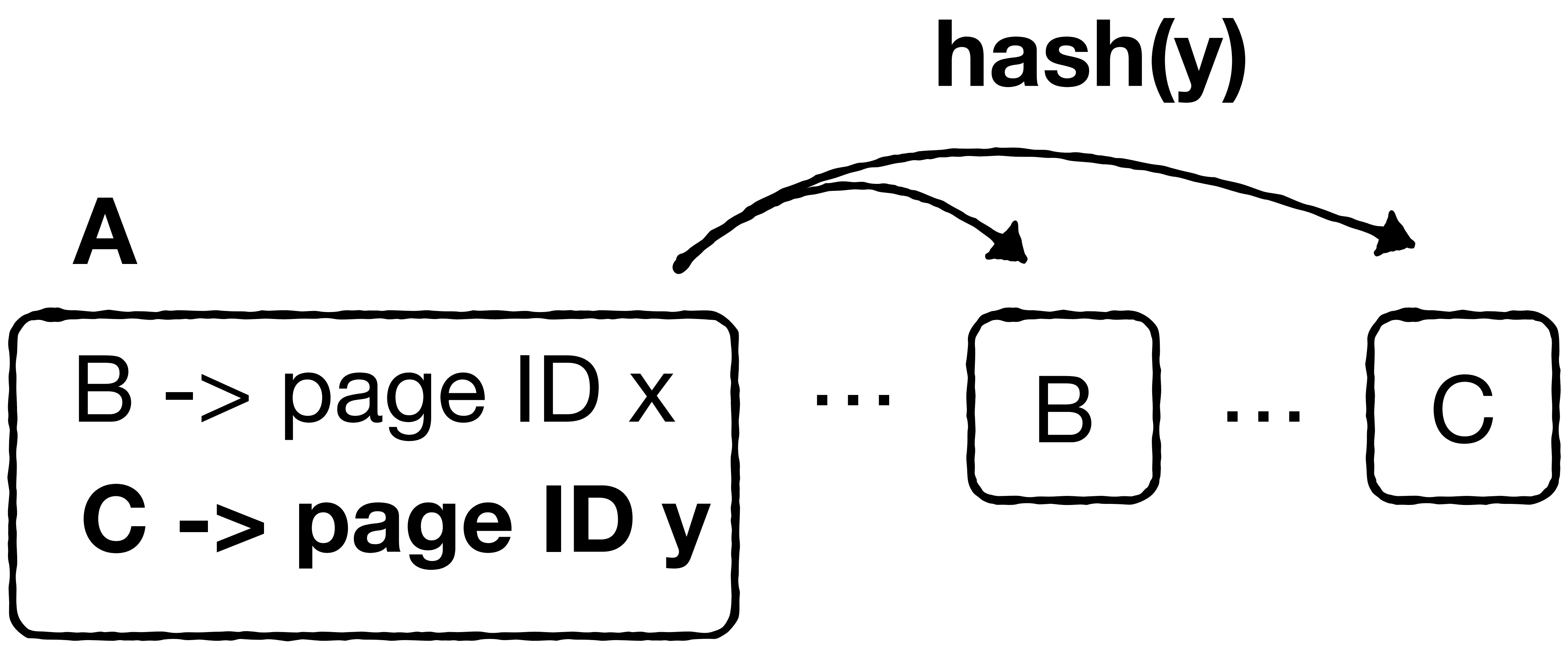
Pointer Swizzling



Pointer Swizzling

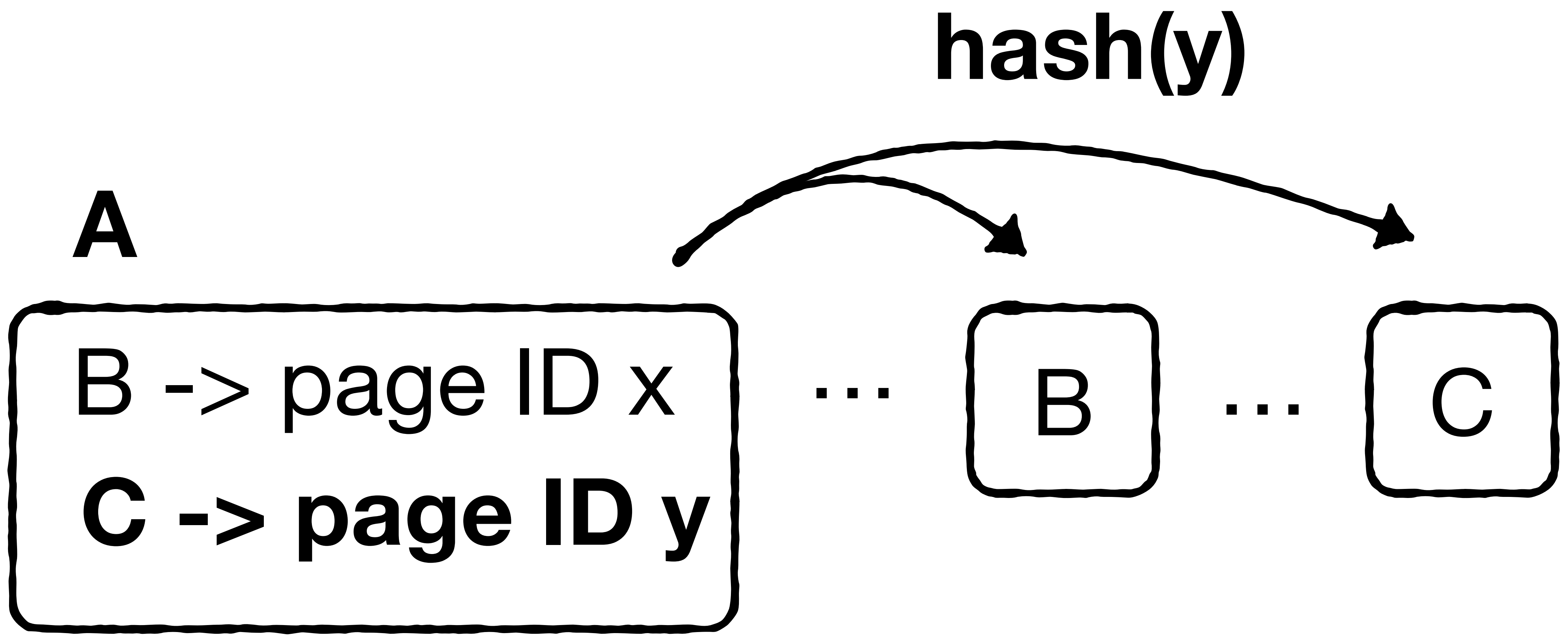


Pointer Swizzling



To follow from one page to another, we must rehash page IDs

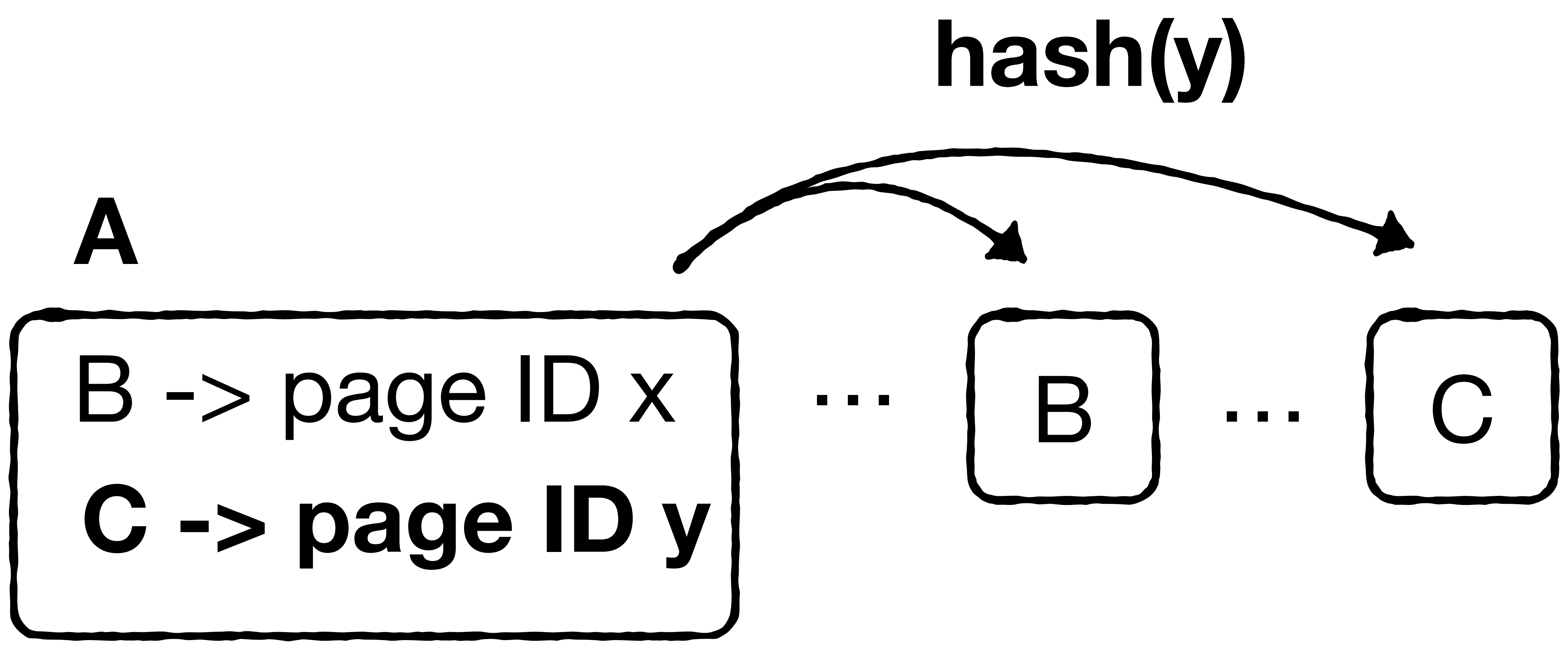
Pointer Swizzling



To follow from one page to another, we must rehash page IDs

This adds overhead.

Pointer Swizzling

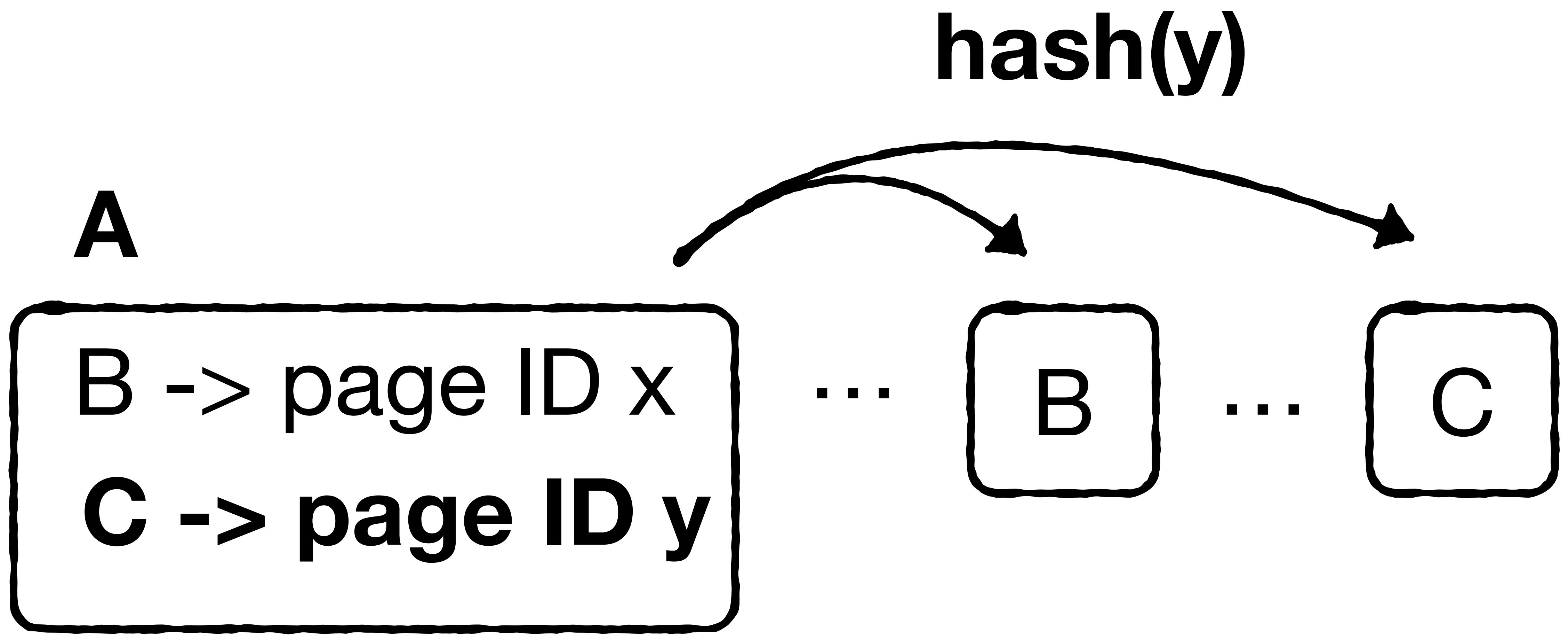


To follow from one page to another, we must rehash page IDs

This adds overhead.

Solutions?

Pointer Swizzling

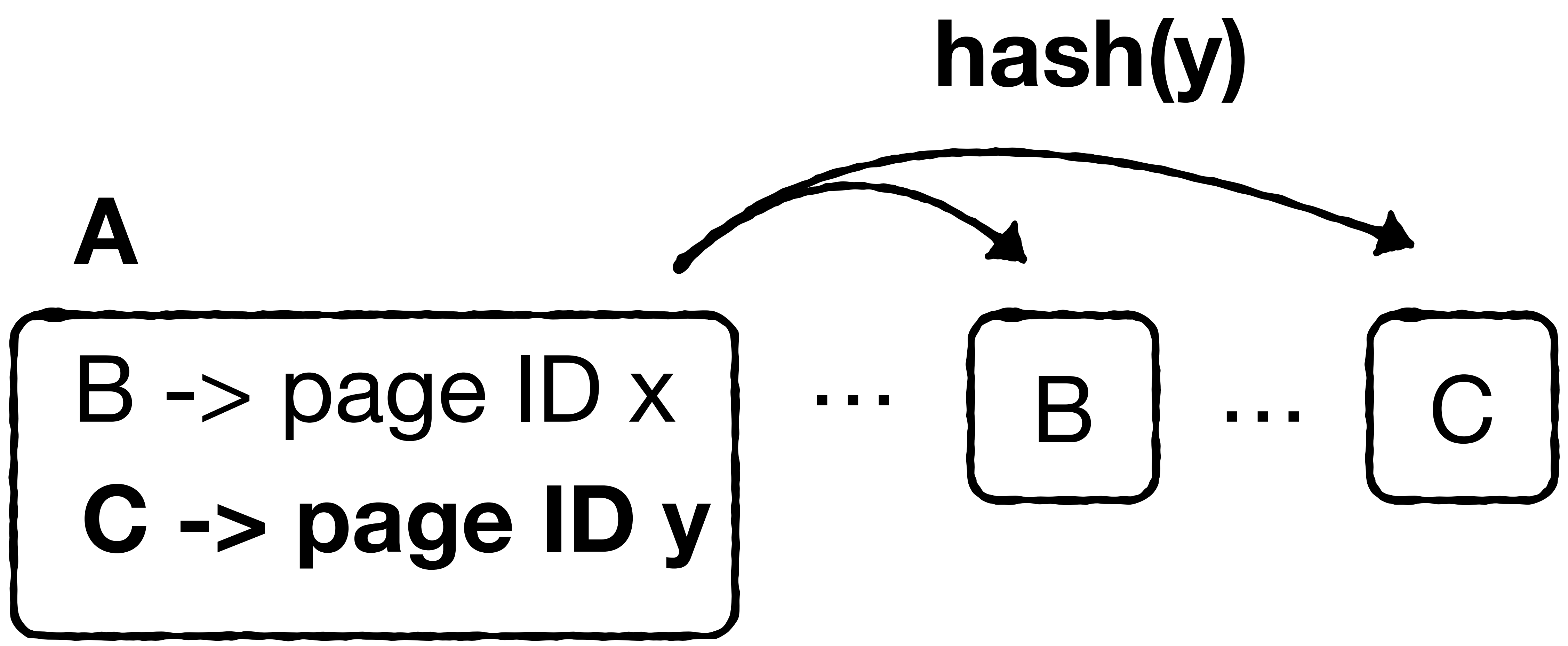


To follow from one page to another, we must rehash page IDs

This adds overhead.

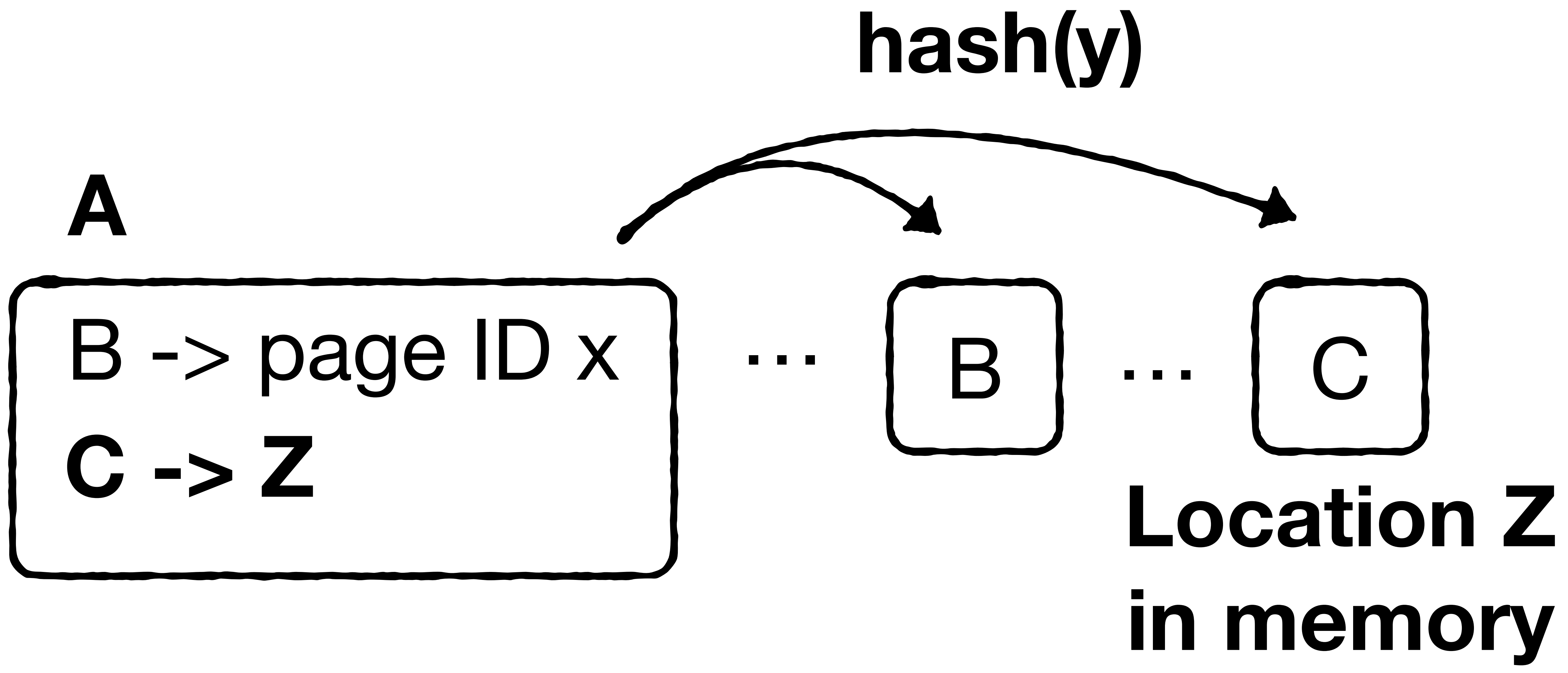
Solutions?

Pointer Swizzling



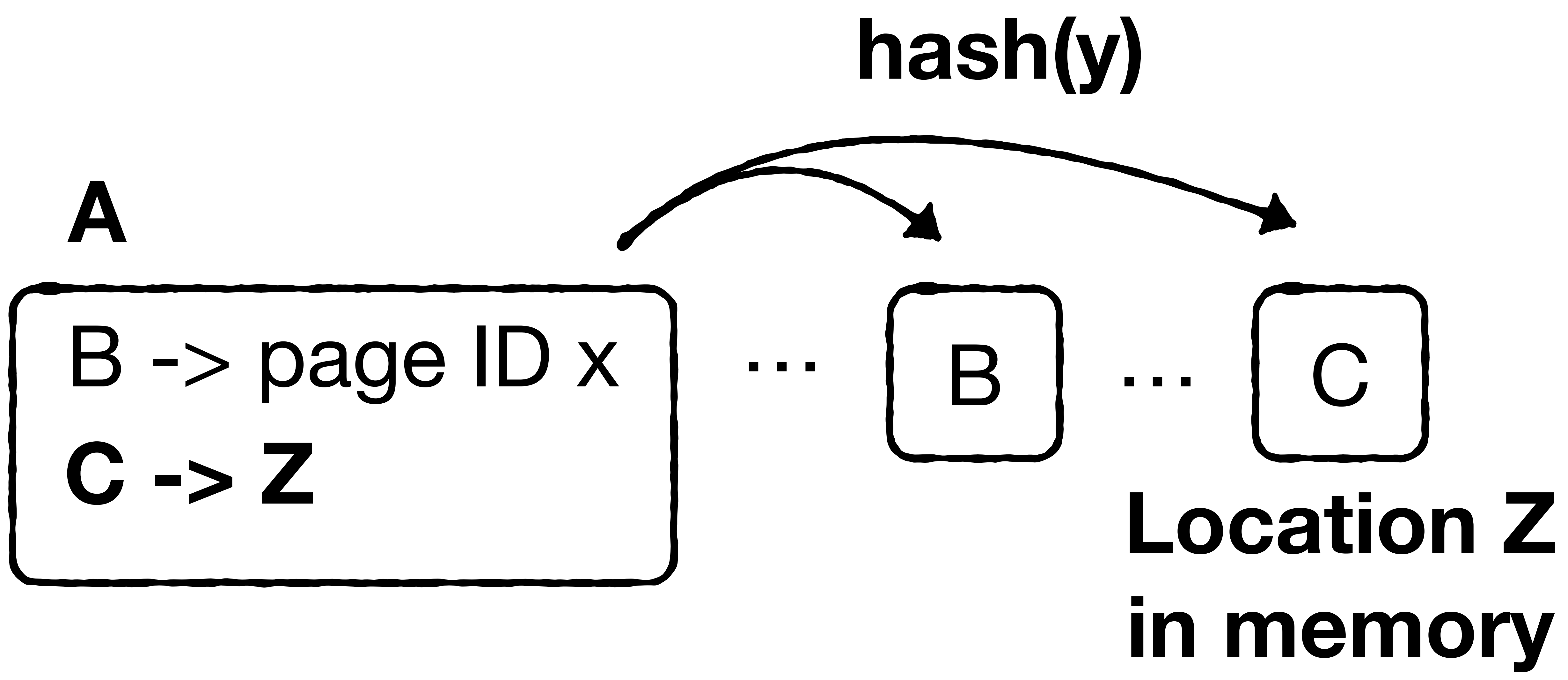
Solutions? **Replace logical page ID by direct in-memory pointer**

Pointer Swizzling



Solutions? **Replace logical page ID by direct in-memory pointer**

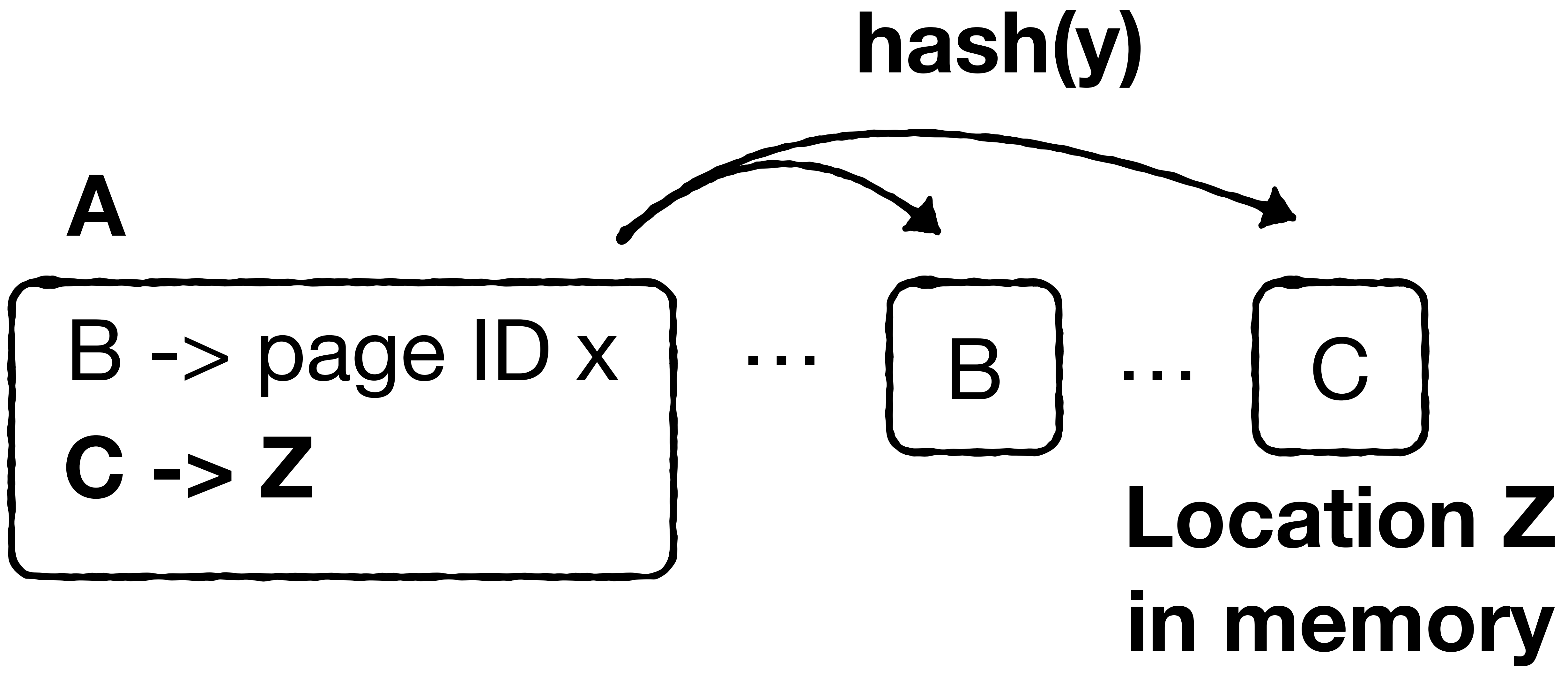
Pointer Swizzling



Solutions? Replace logical page ID by direct in-memory pointer

Question 1: any requirements for hash table?

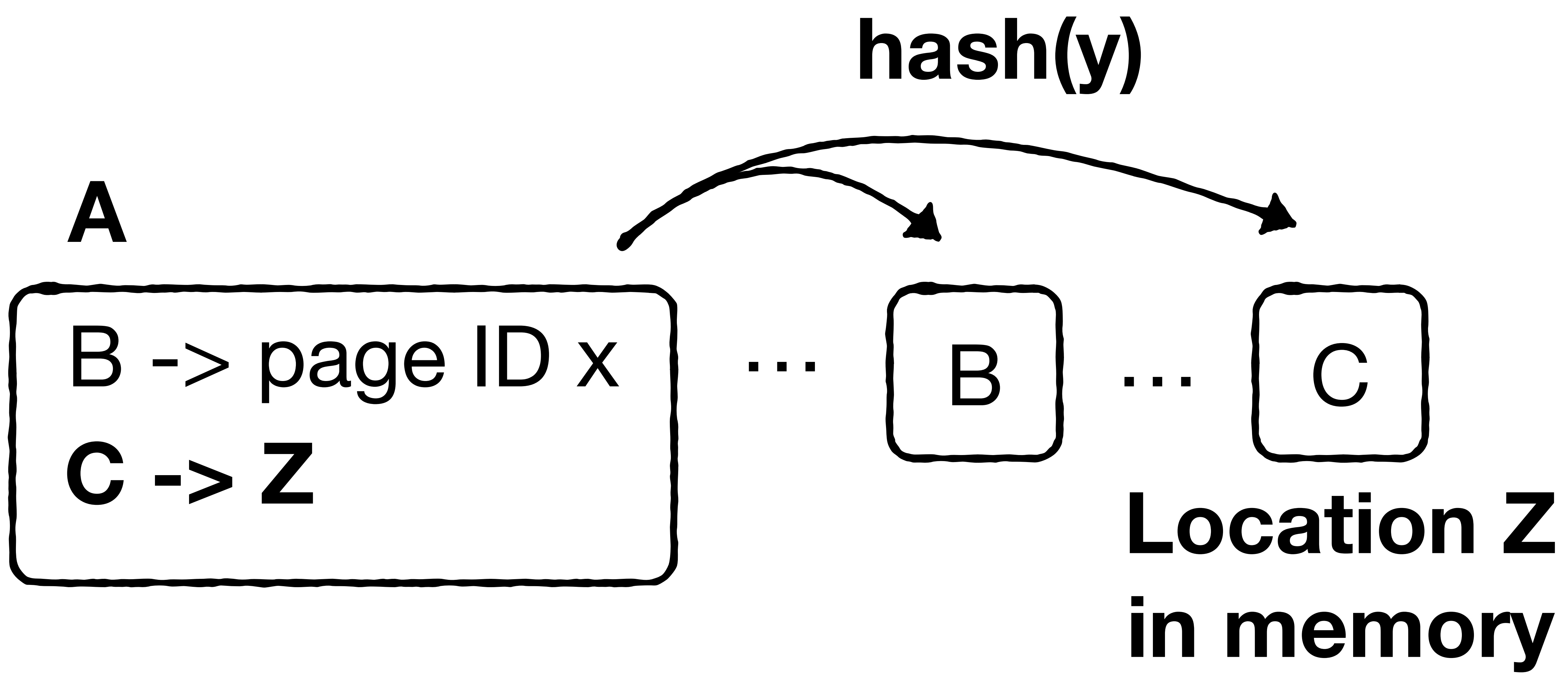
Pointer Swizzling



Question 1: any requirements for hash table?

Question 2: what happens if page C is evicted? :)

Pointer Swizzling



Question 1: any requirements for hash table?

Question 2: what happens if page C is evicted? :)

Question 3: should we swizzle eagerly or lazily?

Thank you!