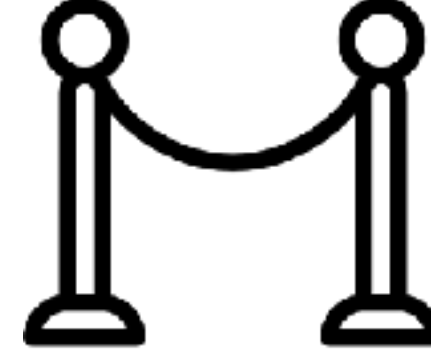


Advanced Buffer Management

Niv Dayan - CSC2525: Research Topics in Database Management



UB-Tree



**FIFO/LRU
Review**



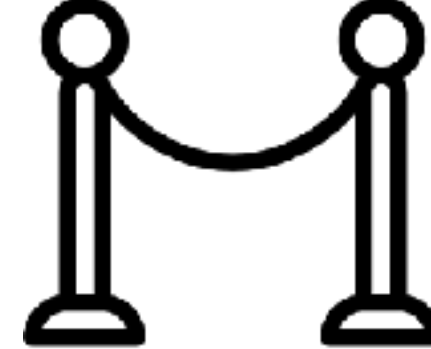
LRU/K



2Q



UB-Tree



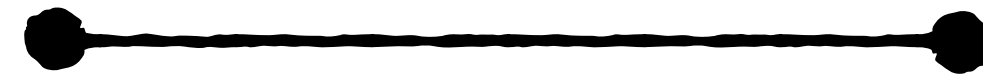
FIFO/LRU
Review



LRU/K



2Q



2 weeks ago

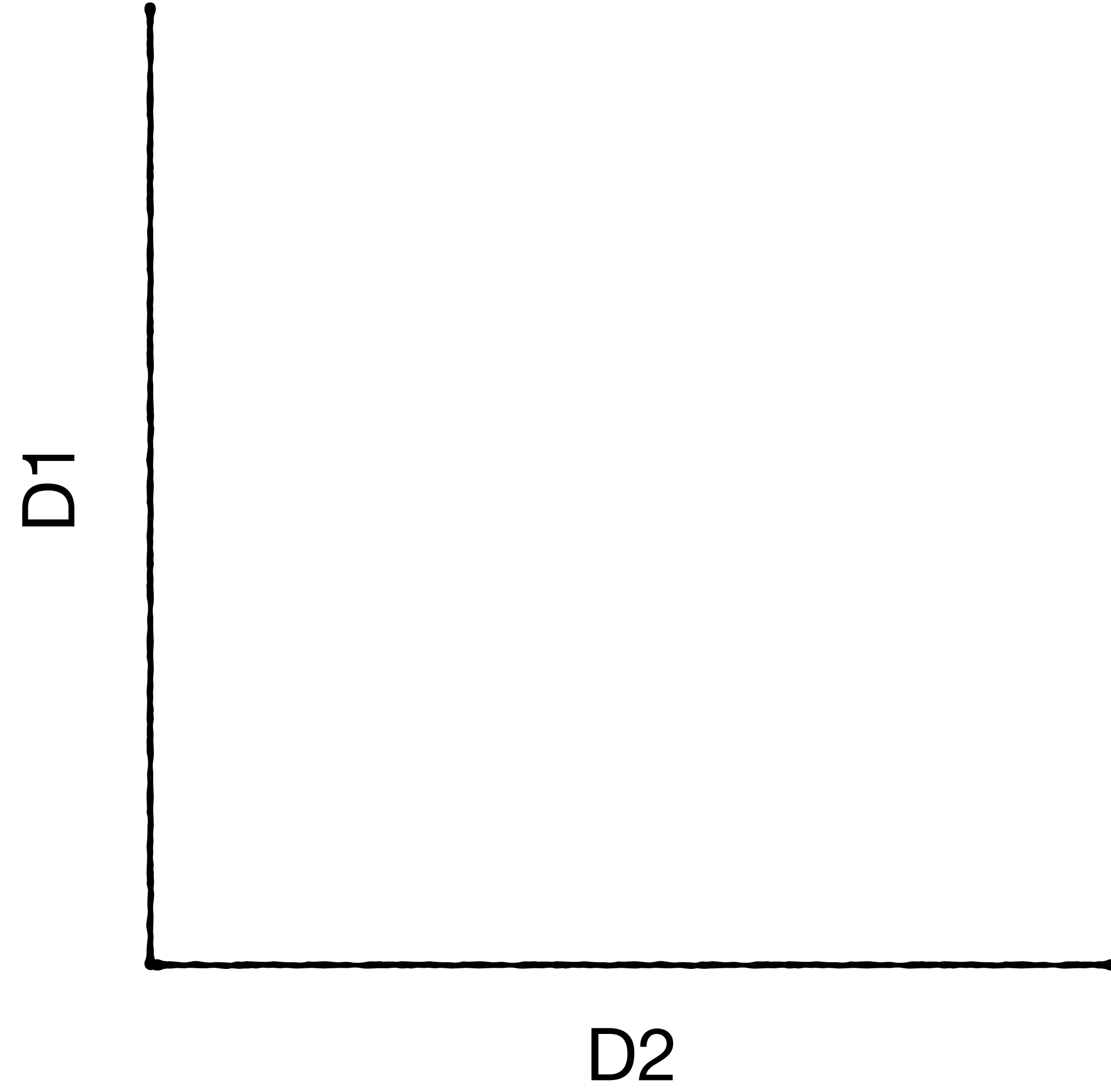


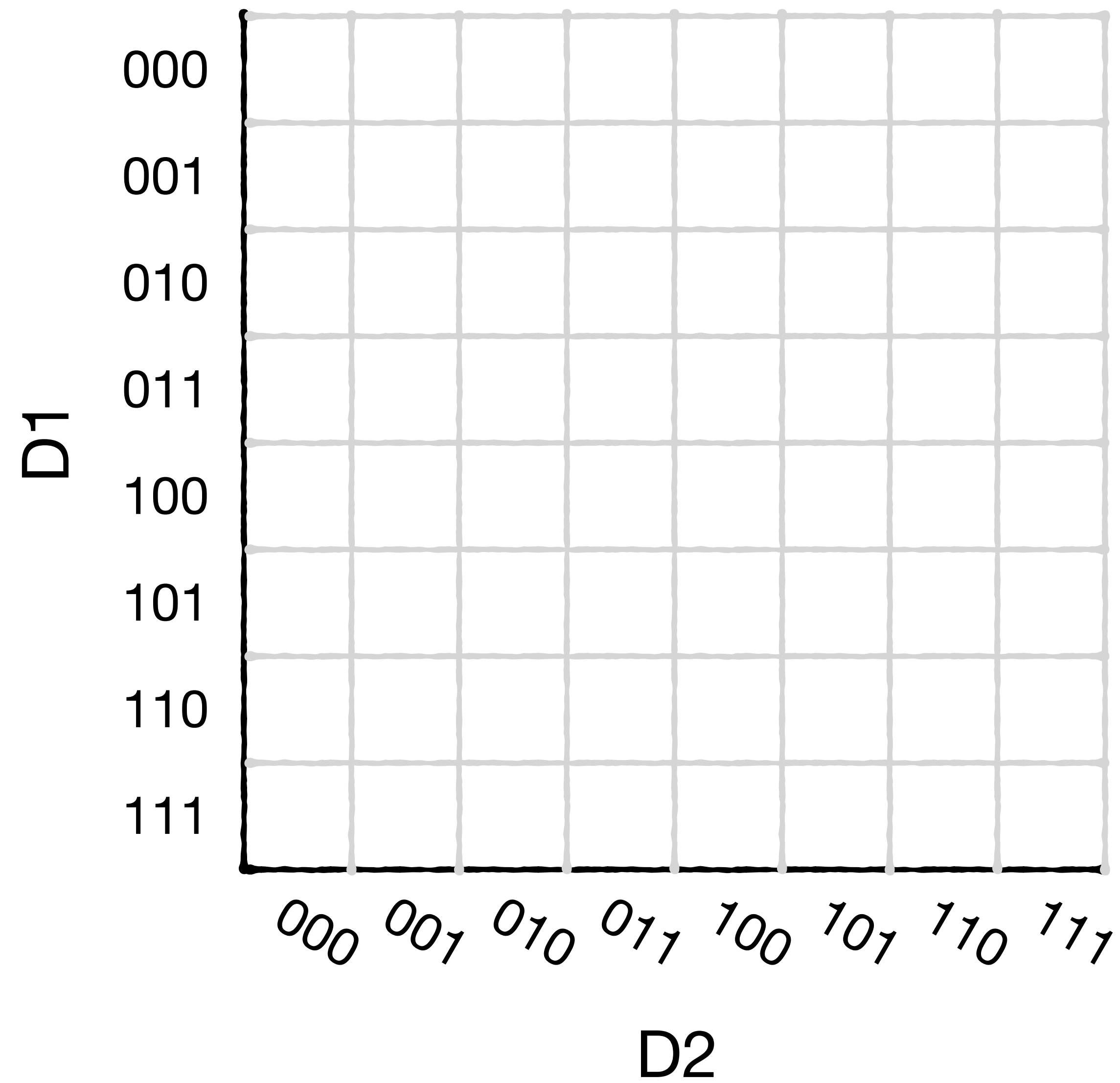
This week

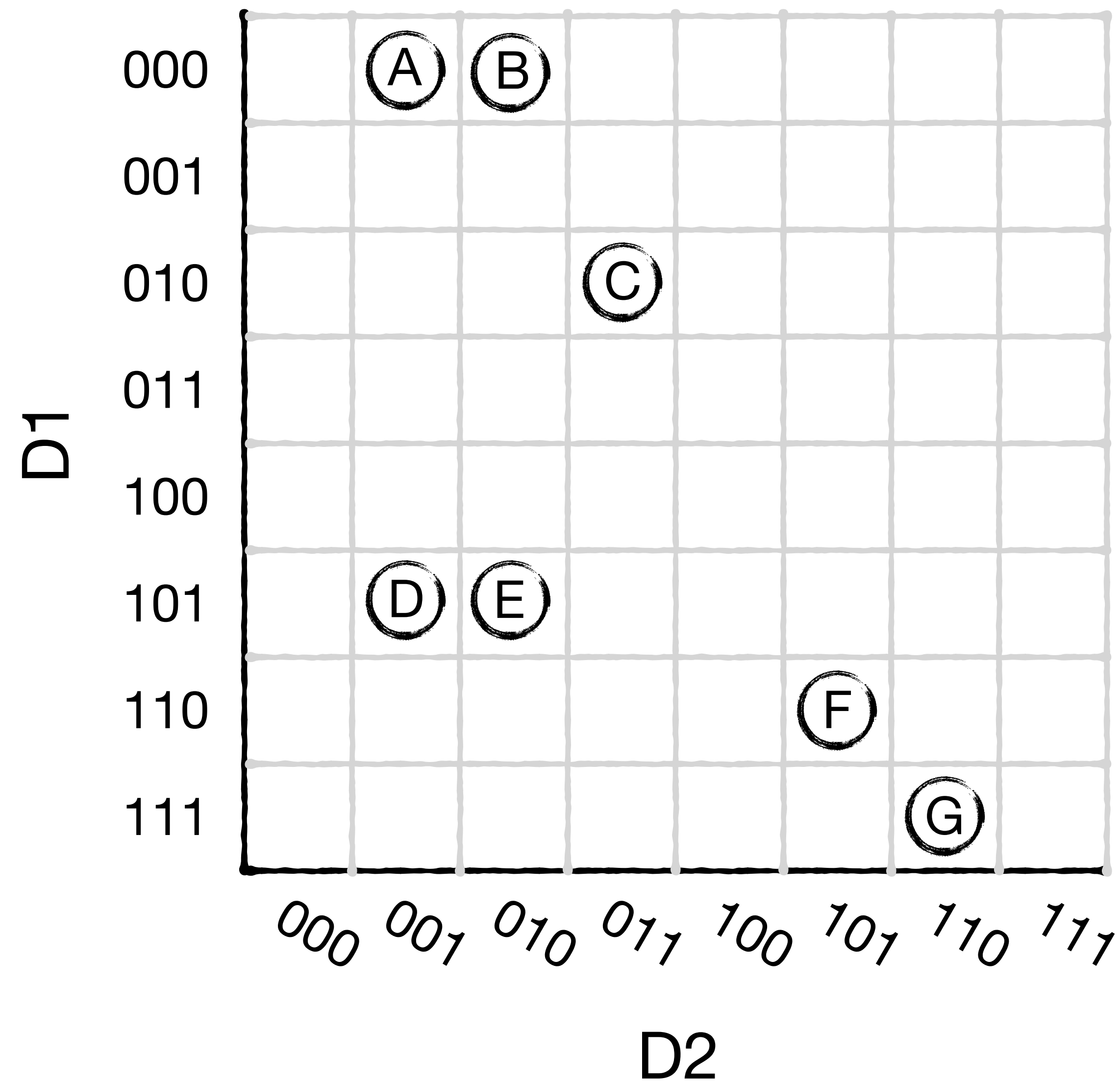
Integrating the UB-Tree into a Database System Kernel

Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhardt, Rudolf Bayer

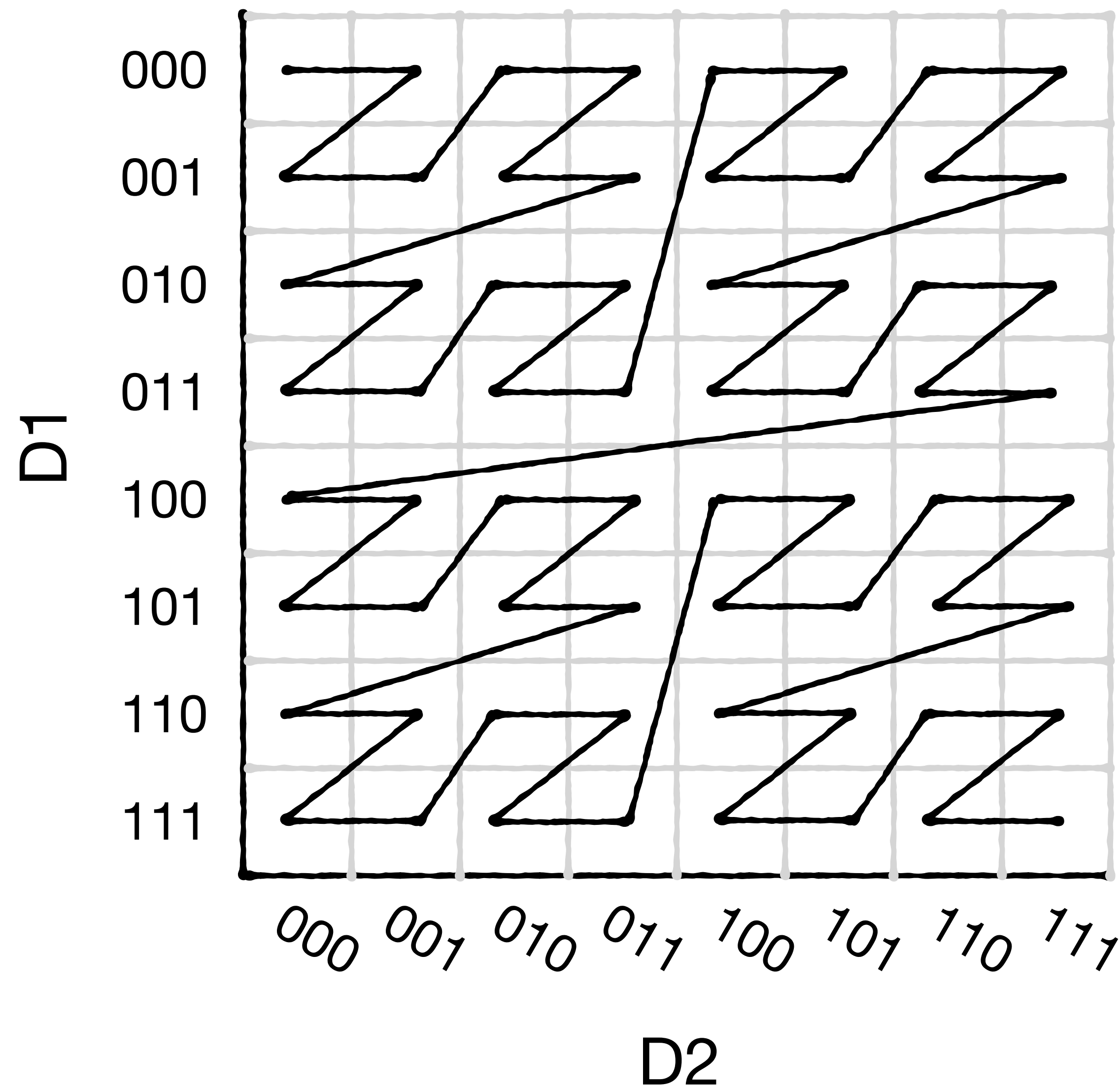
VLDB 2000



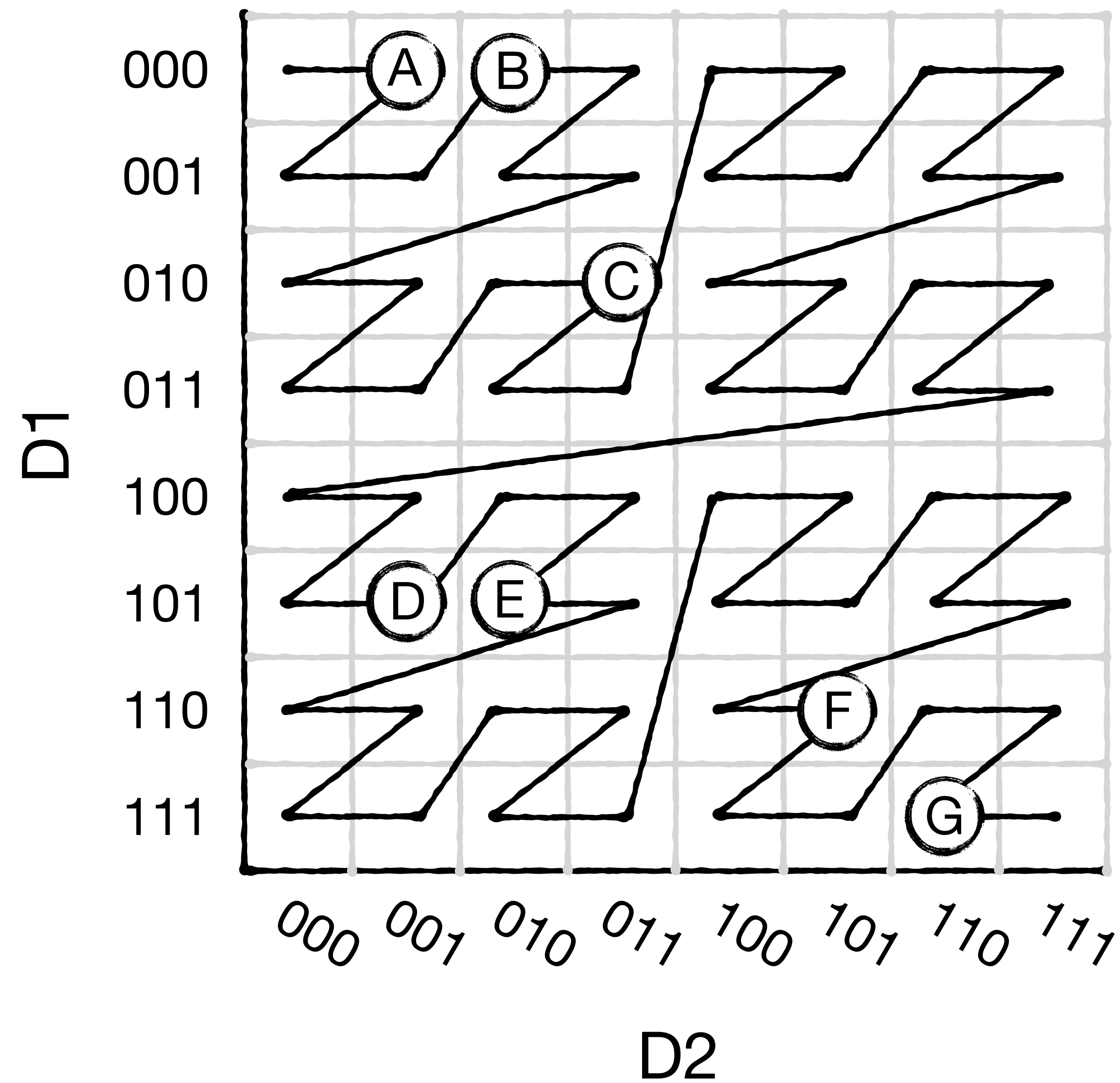


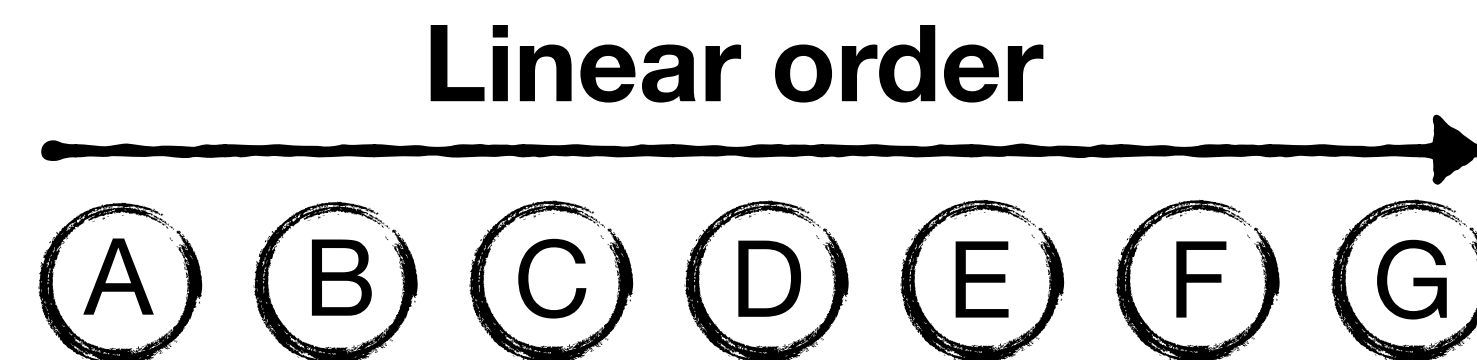
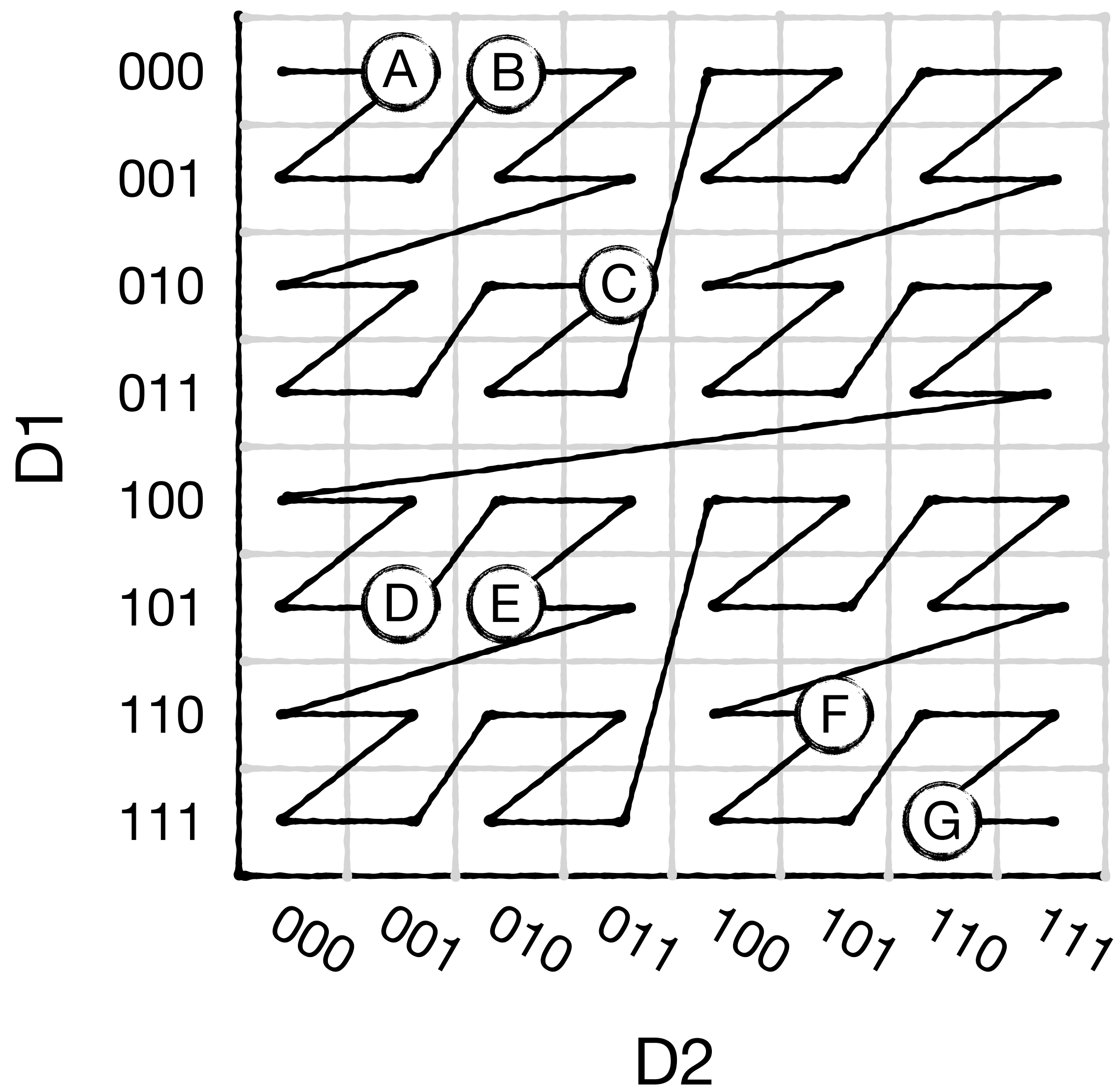


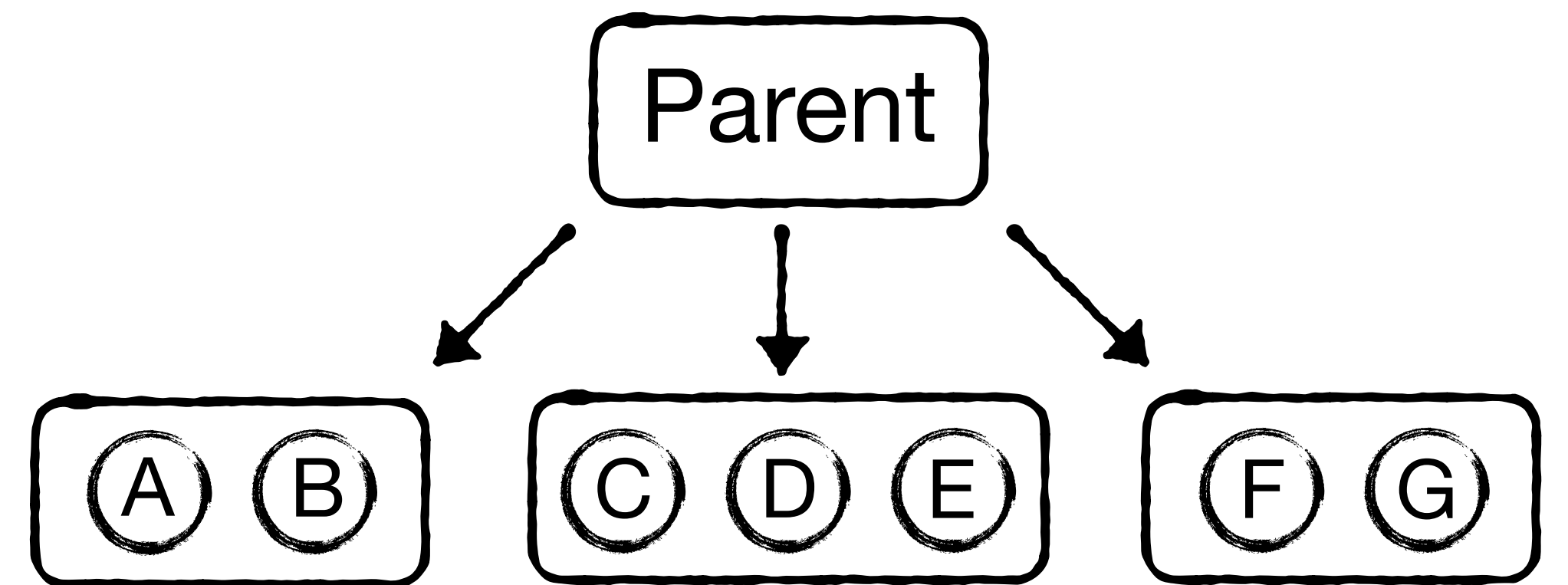
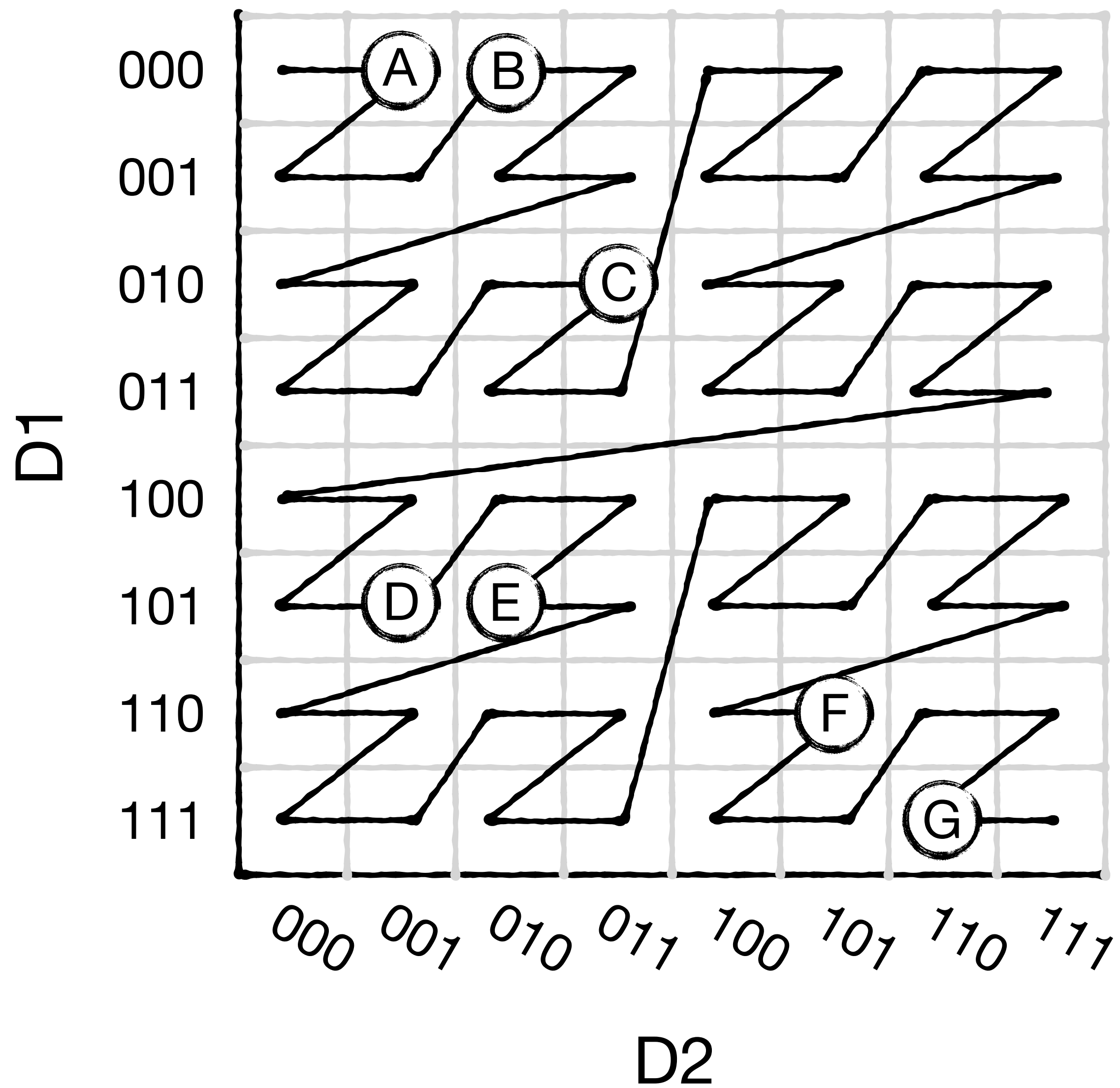
Z-Order Curves



Z-Order Curves

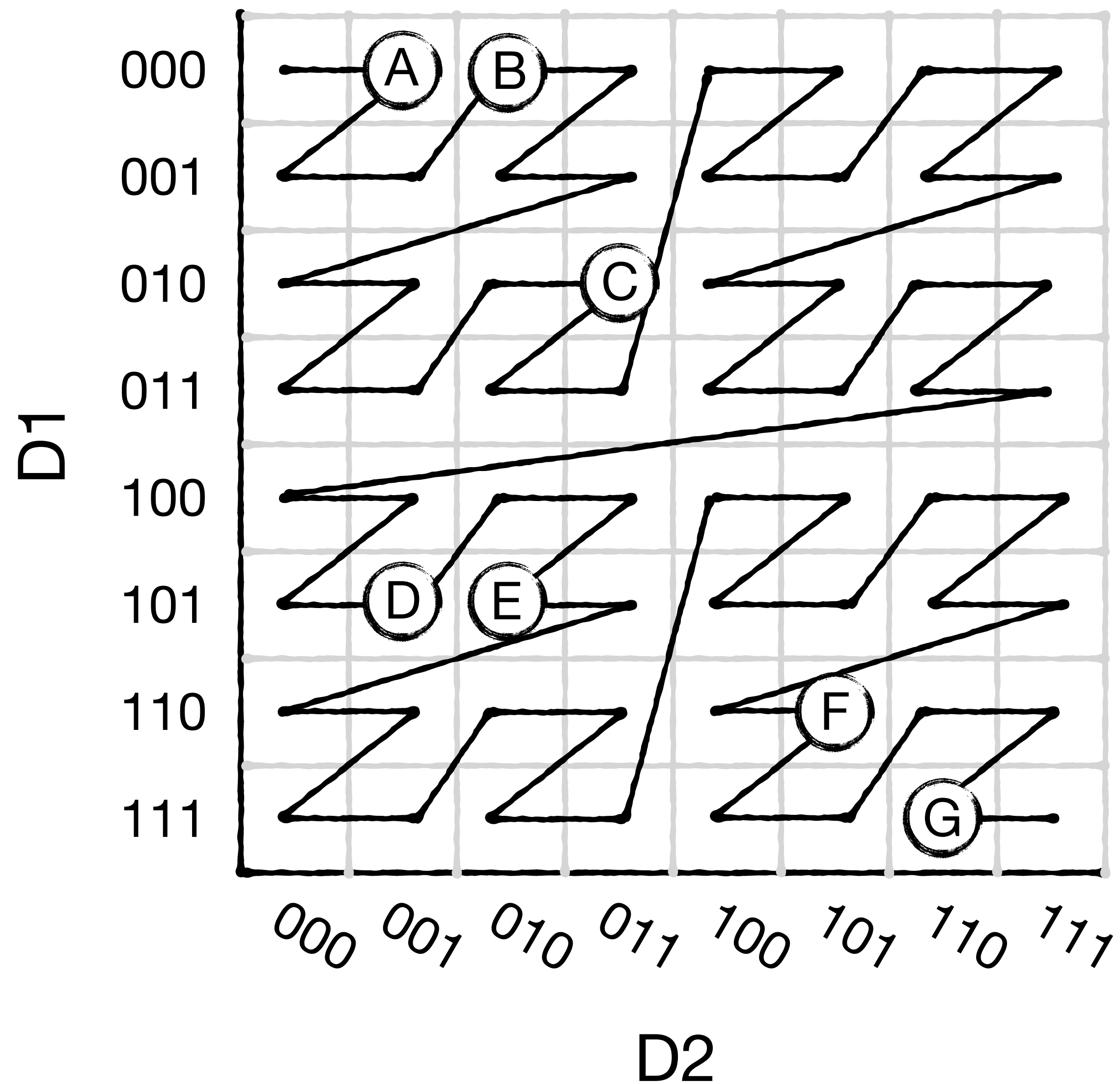






Construct a B-tree

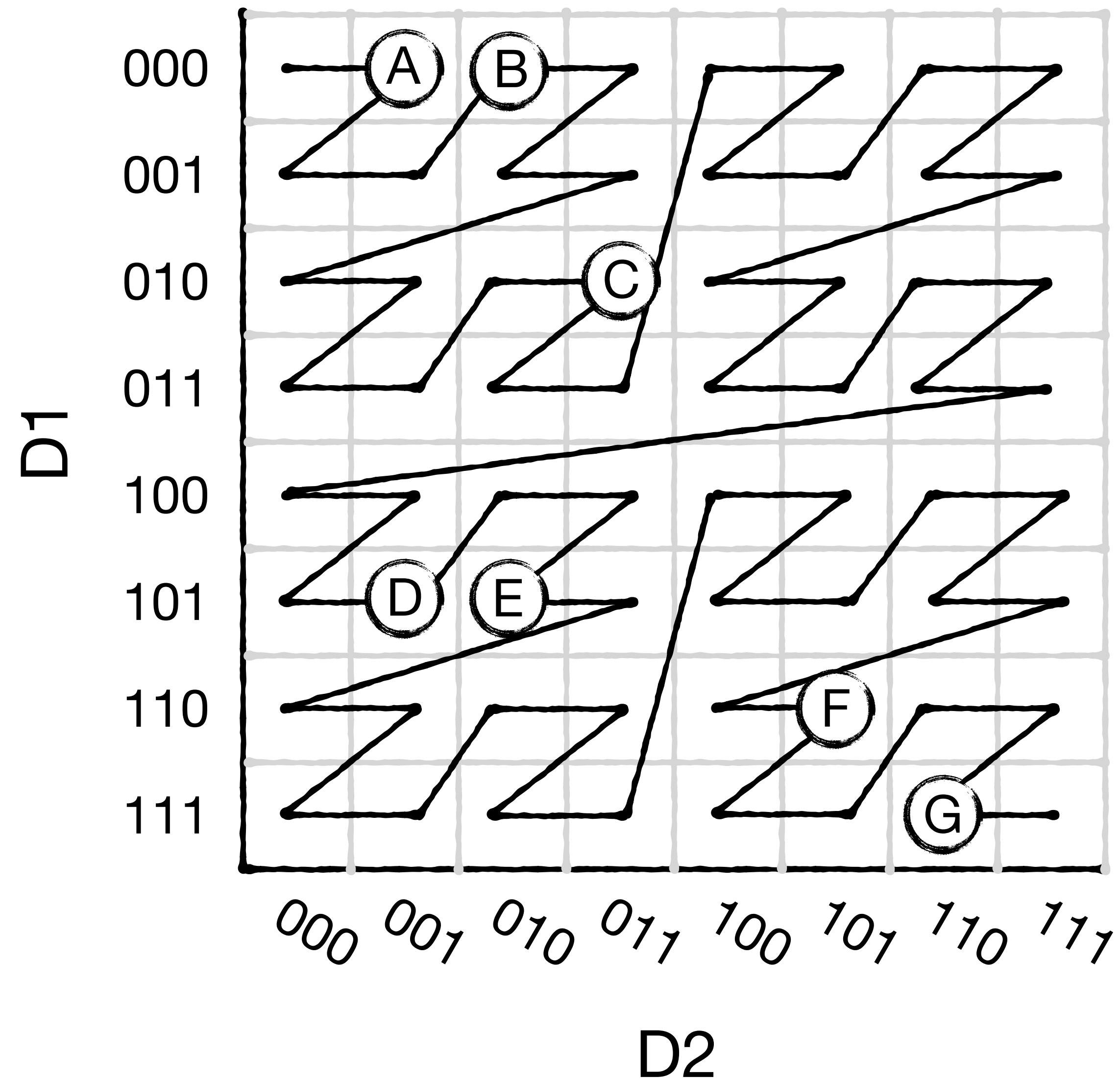
How to compare keys in the Z-Order?



Ⓐ <? Ⓑ

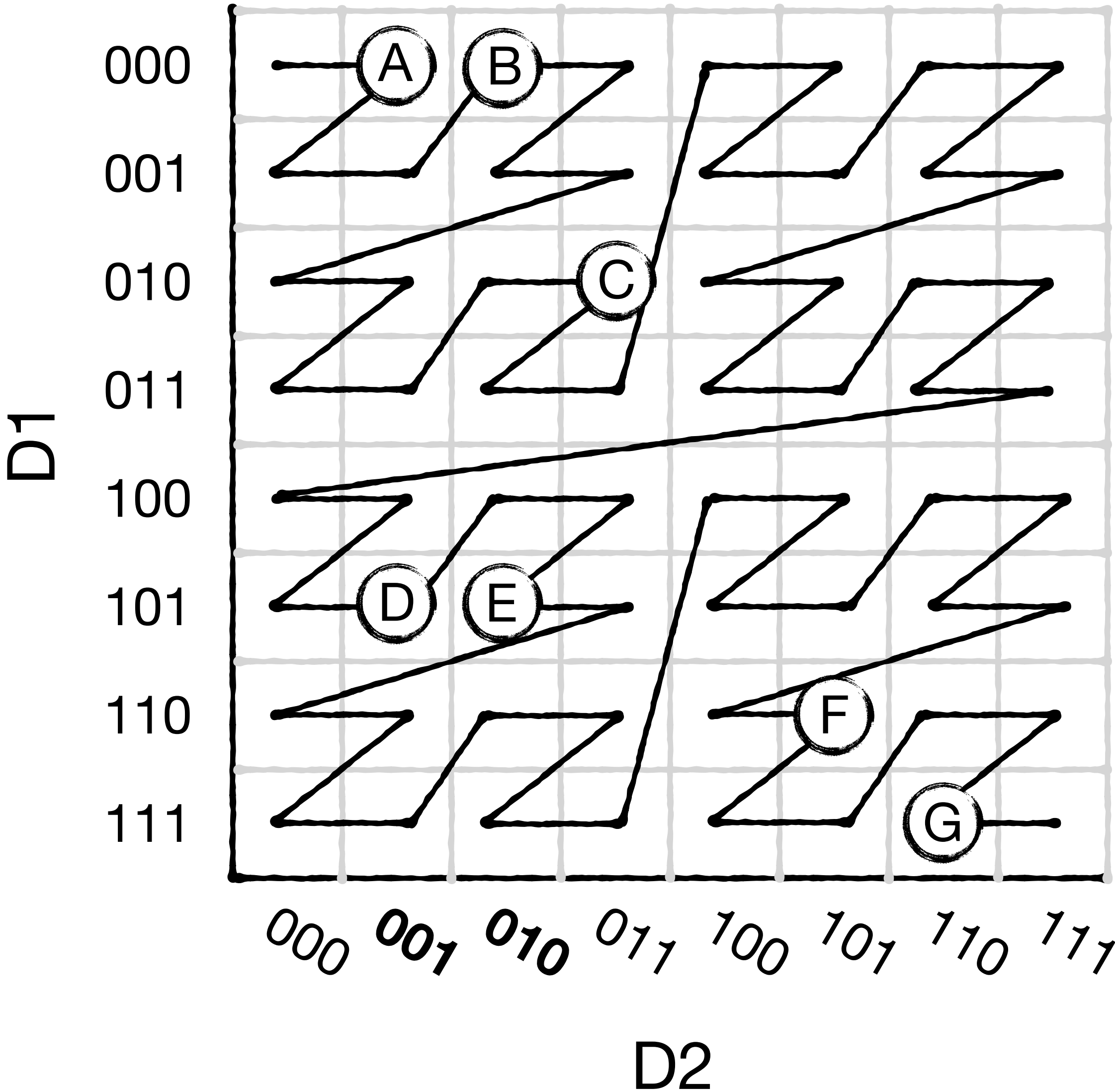
How to compare keys in the Z-Order?

Interleaving dimension bits



Ⓐ <? Ⓑ

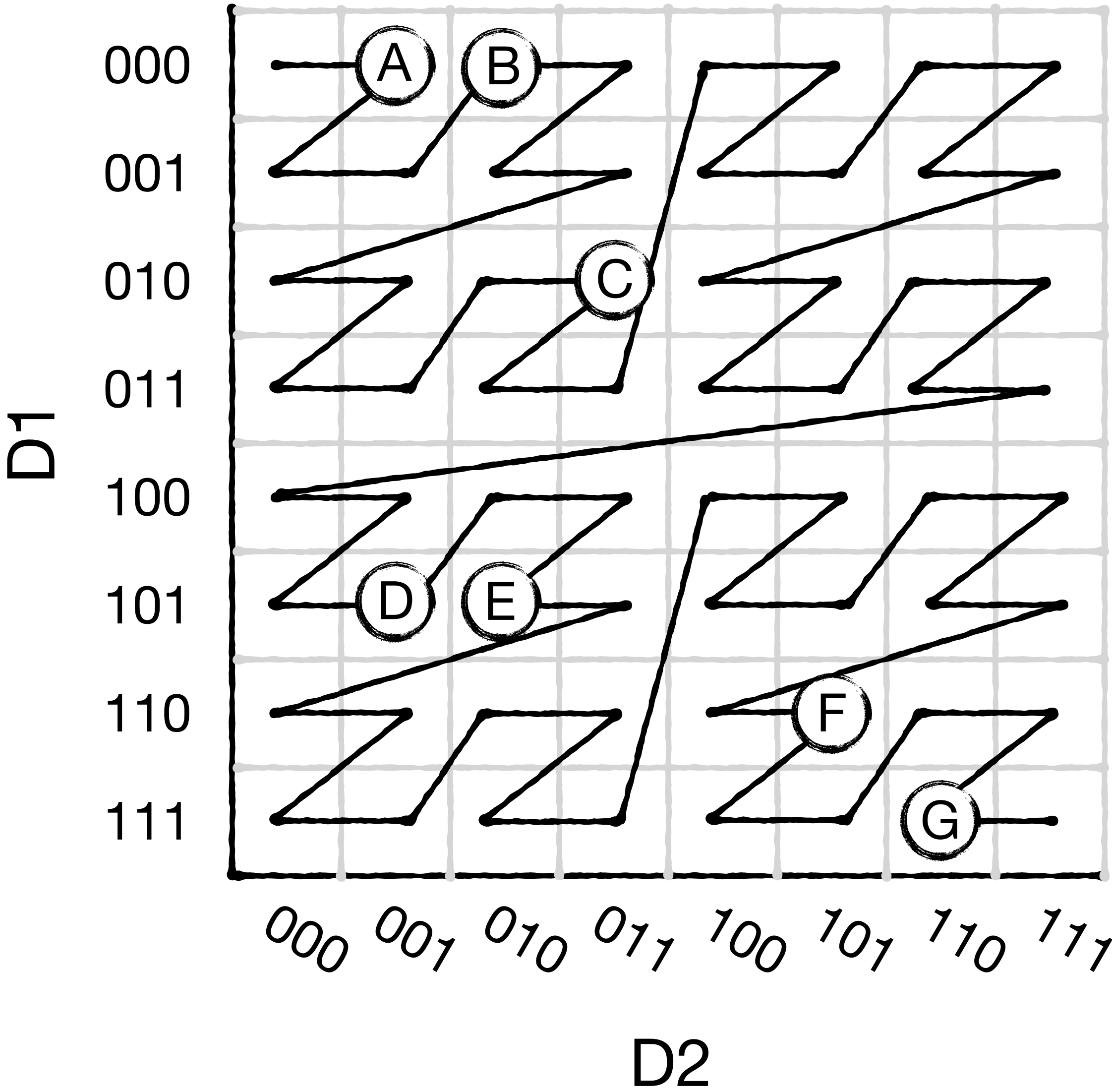
Interleaving dimension bits



D1
D2

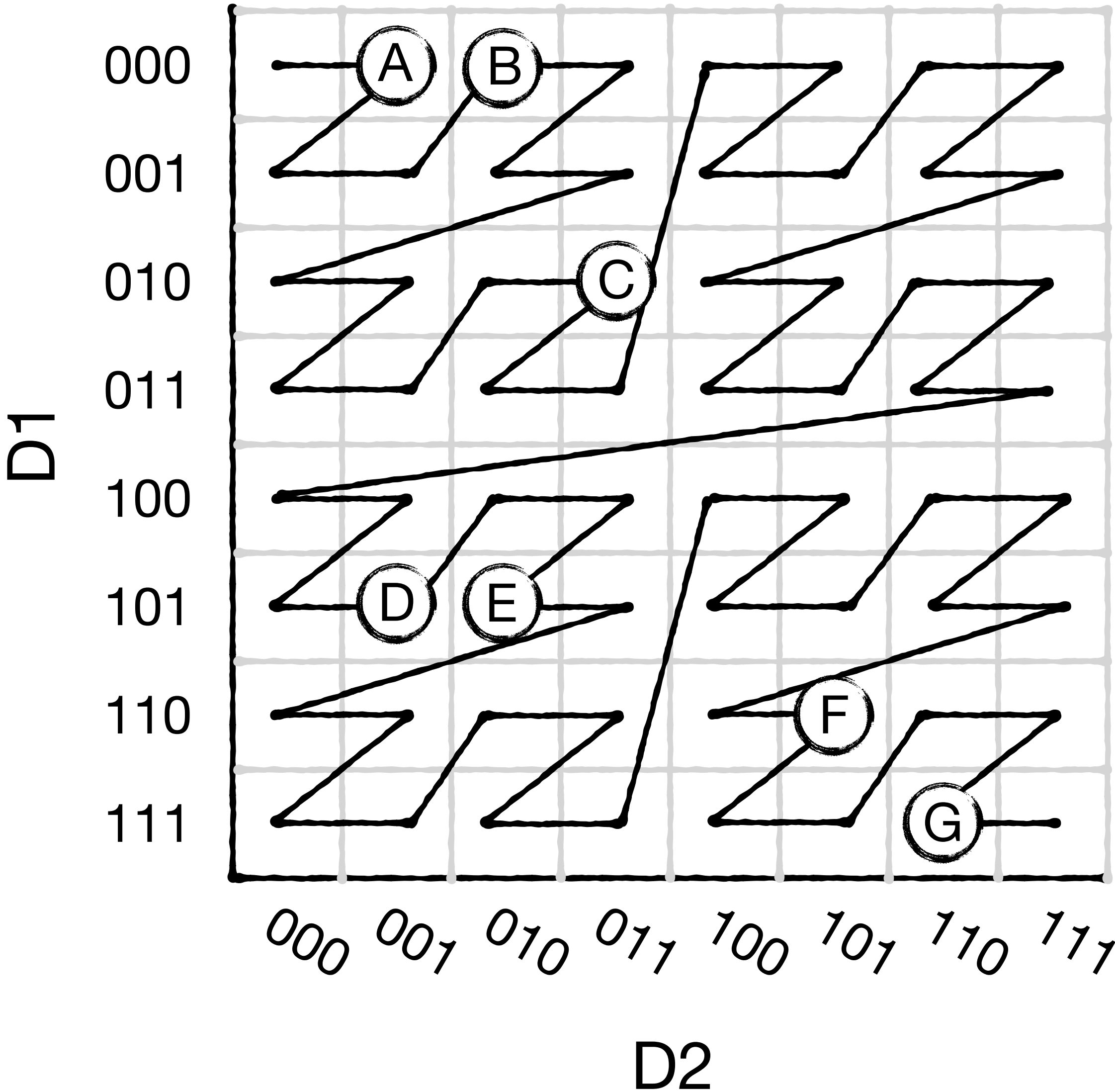
000		000
001		010
(A)	<?	(B)

Interleaving dimension bits



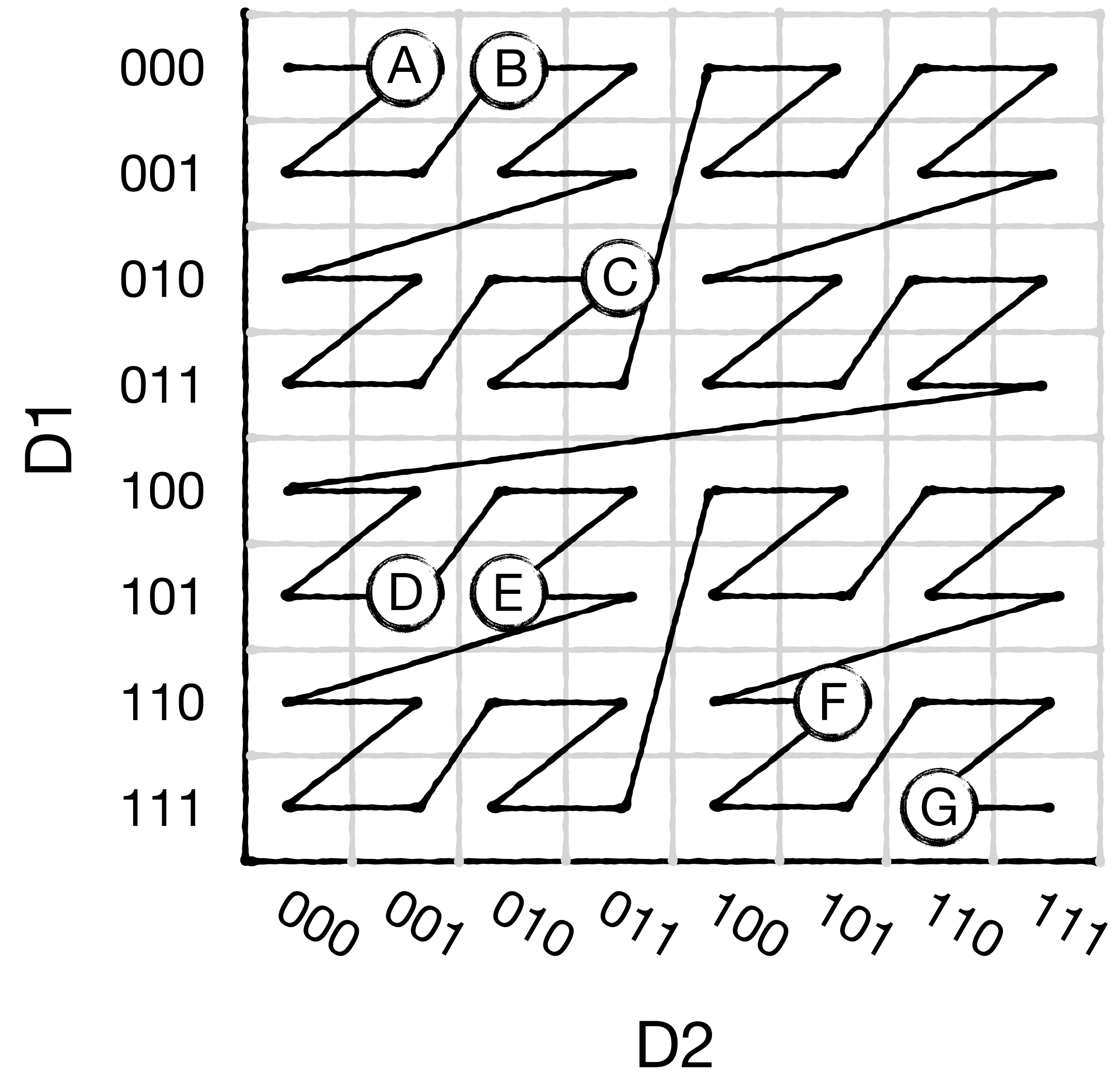
D1	000	000
D2	001	010
Z-Key	(A)	(B)
	<?	

Interleaving dimension bits



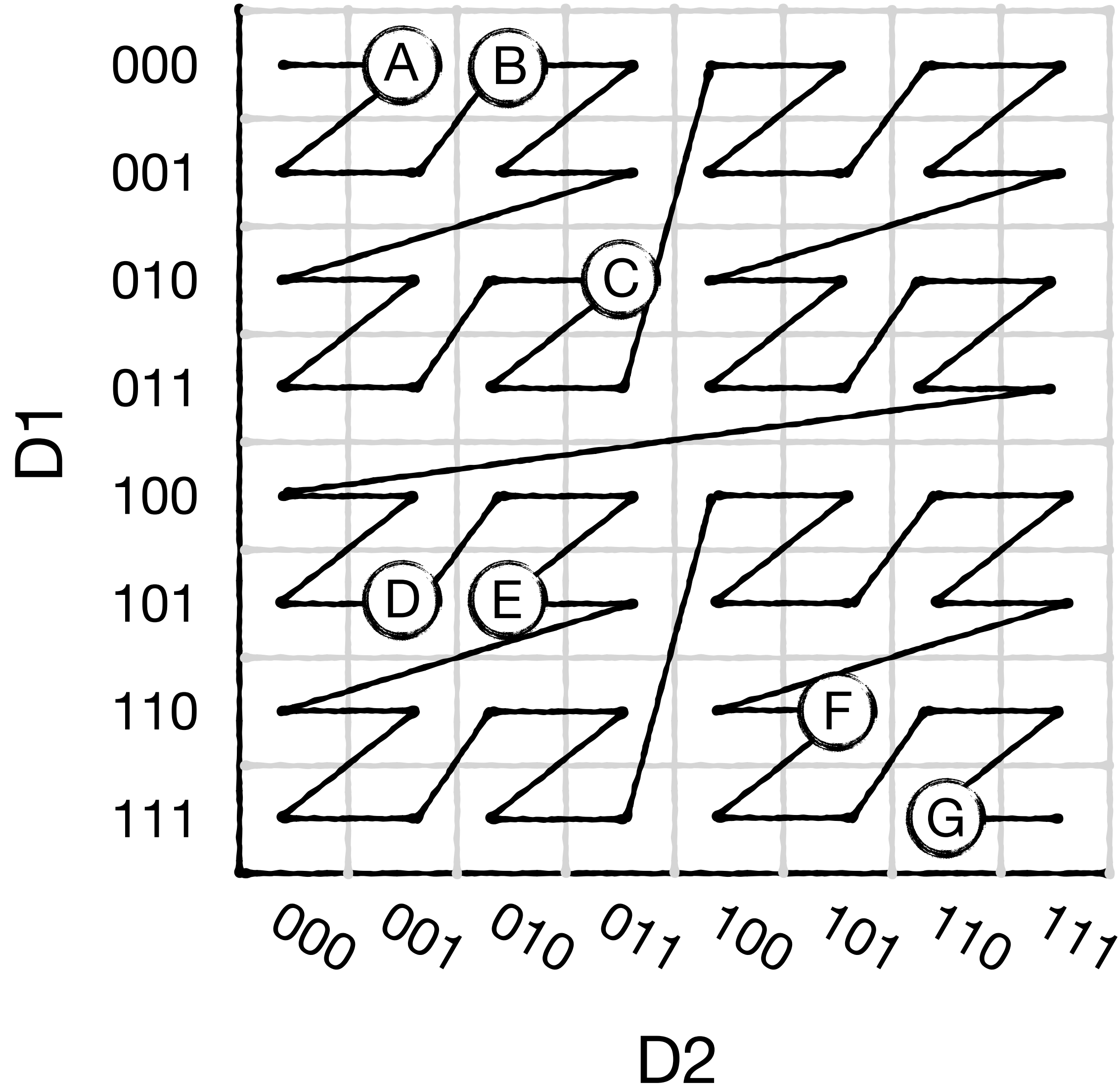
D1	000	000
D2	001	010
Z-Key	00	
	A	B
	<?	

Interleaving dimension bits



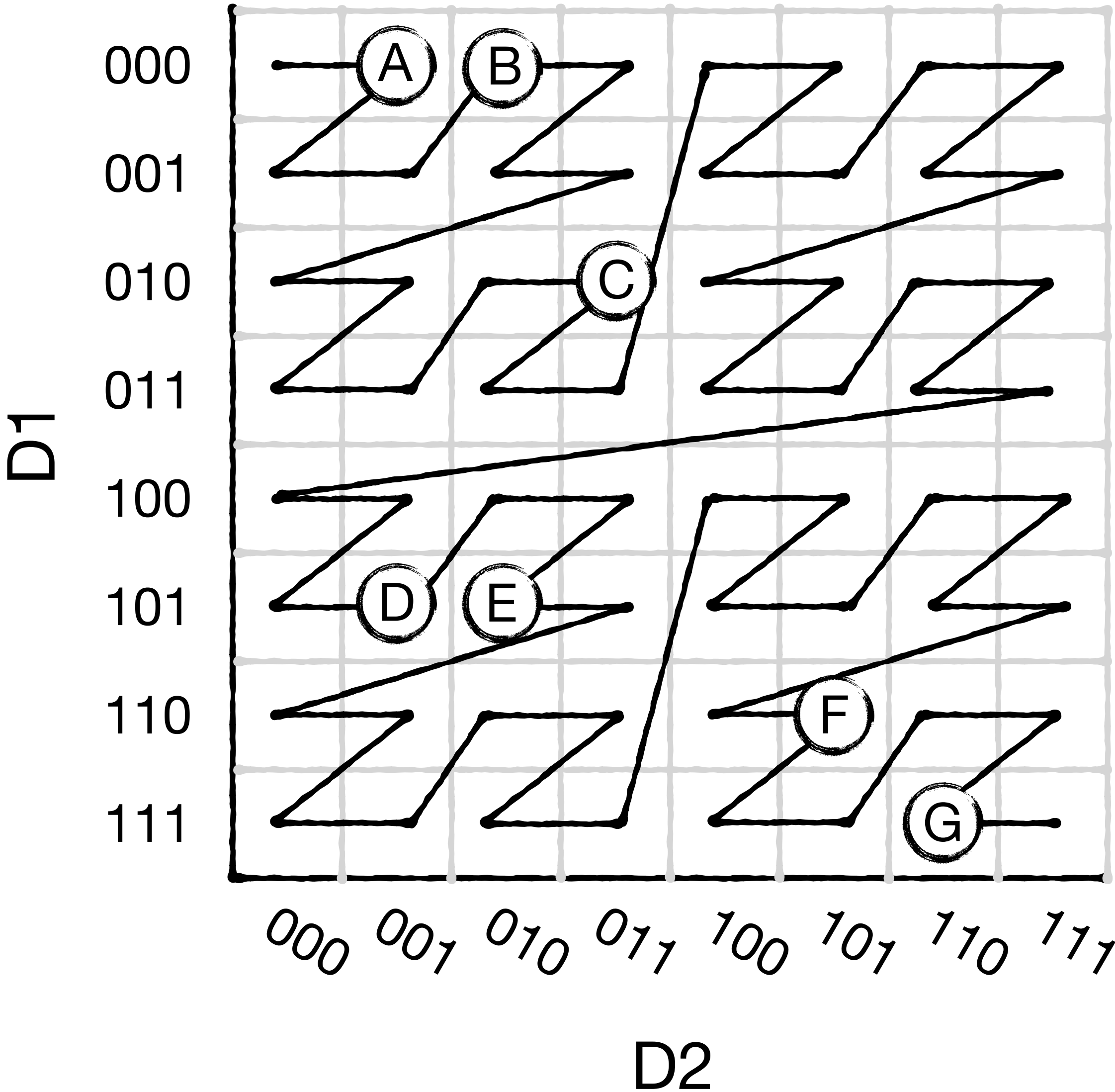
D1	000	000
D2	001	010
Z-Key	00 00	
	(A)	<? (B)

Interleaving dimension bits



D1	000	000
D2	001	010
Z-Key	00 00 01	
	(A)	(B)

Interleaving dimension bits

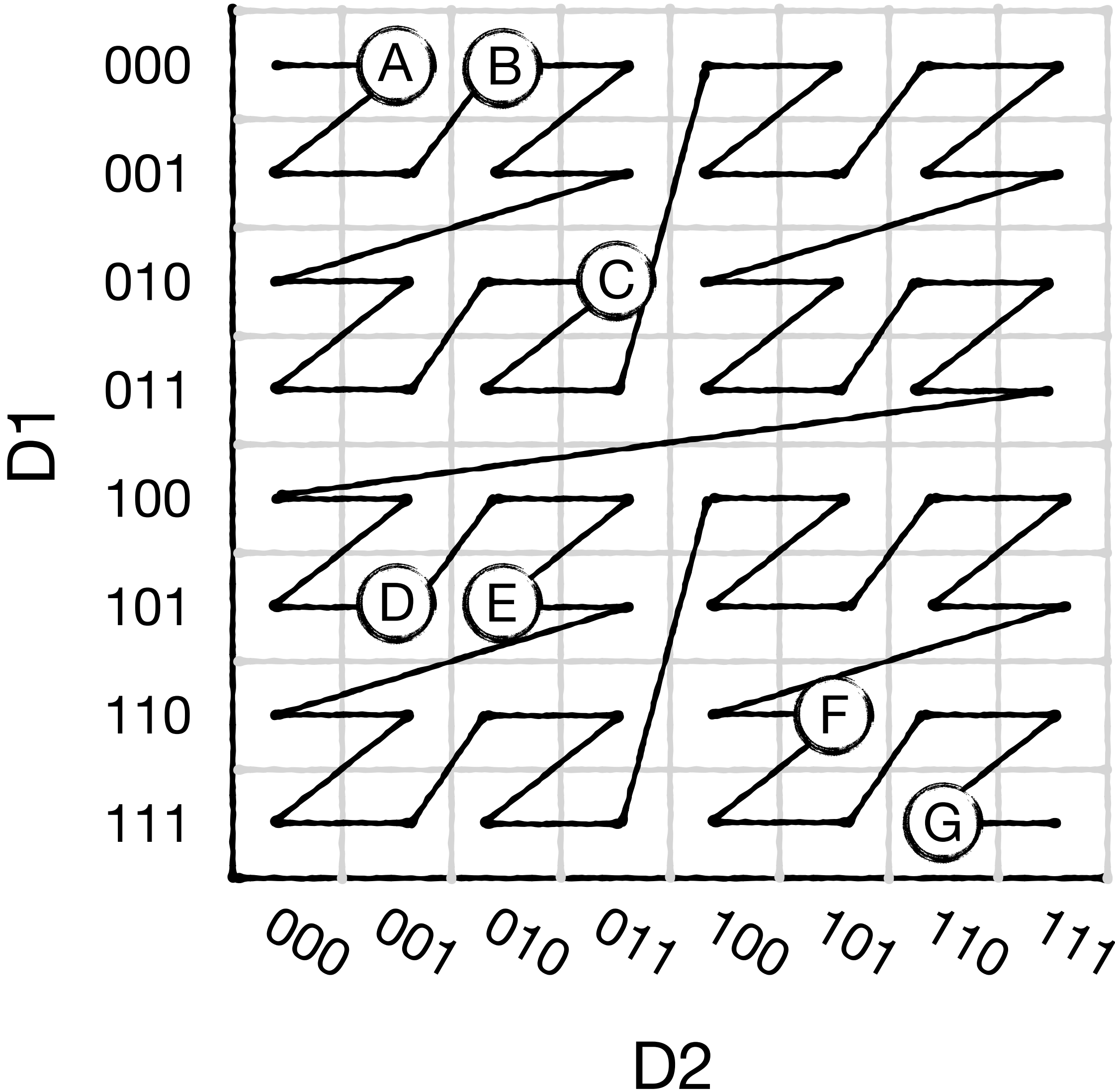


D1 000
D2 001
Z-Key 00 00 01

A <? **B**

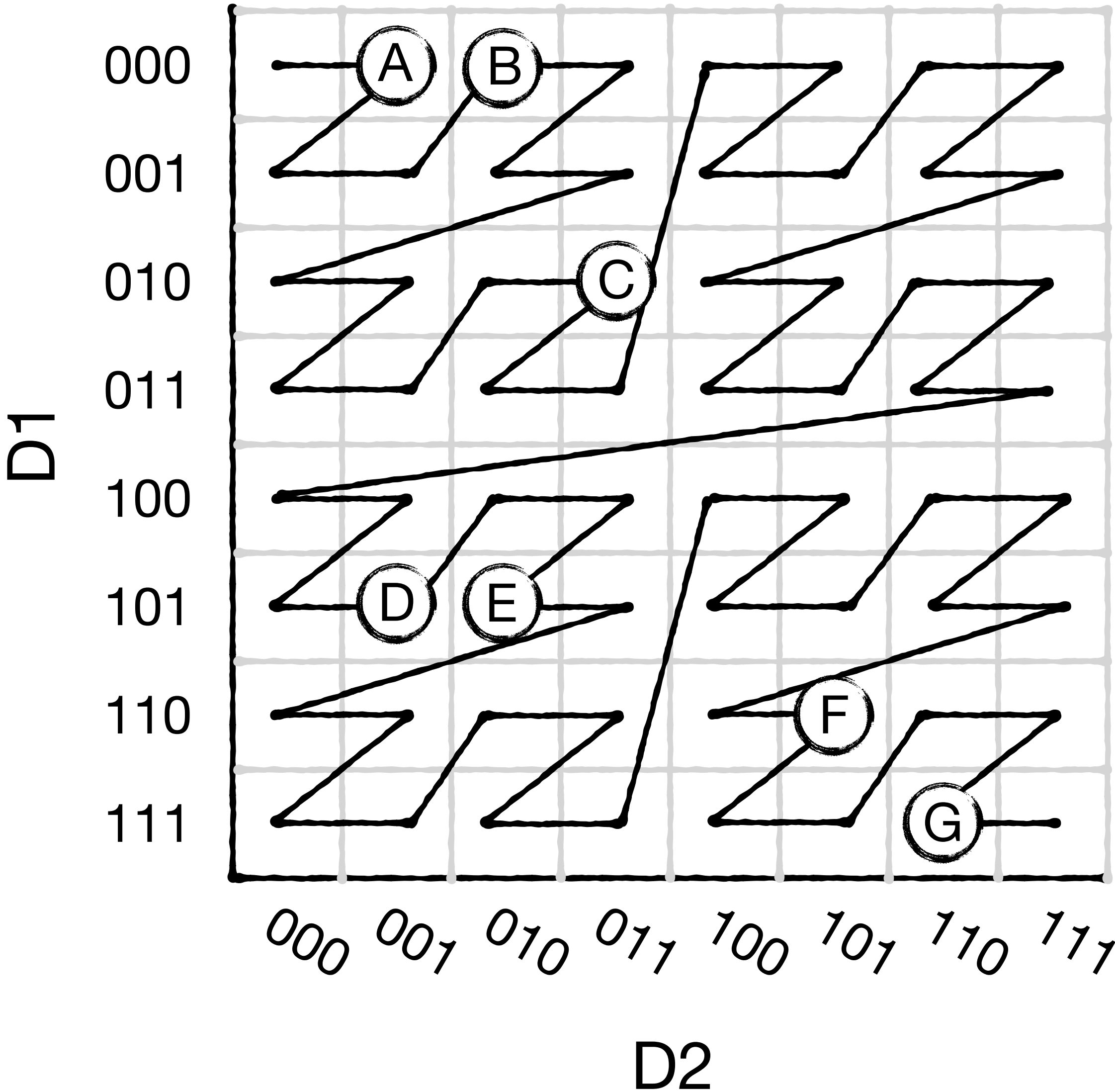
000
010
00

Interleaving dimension bits

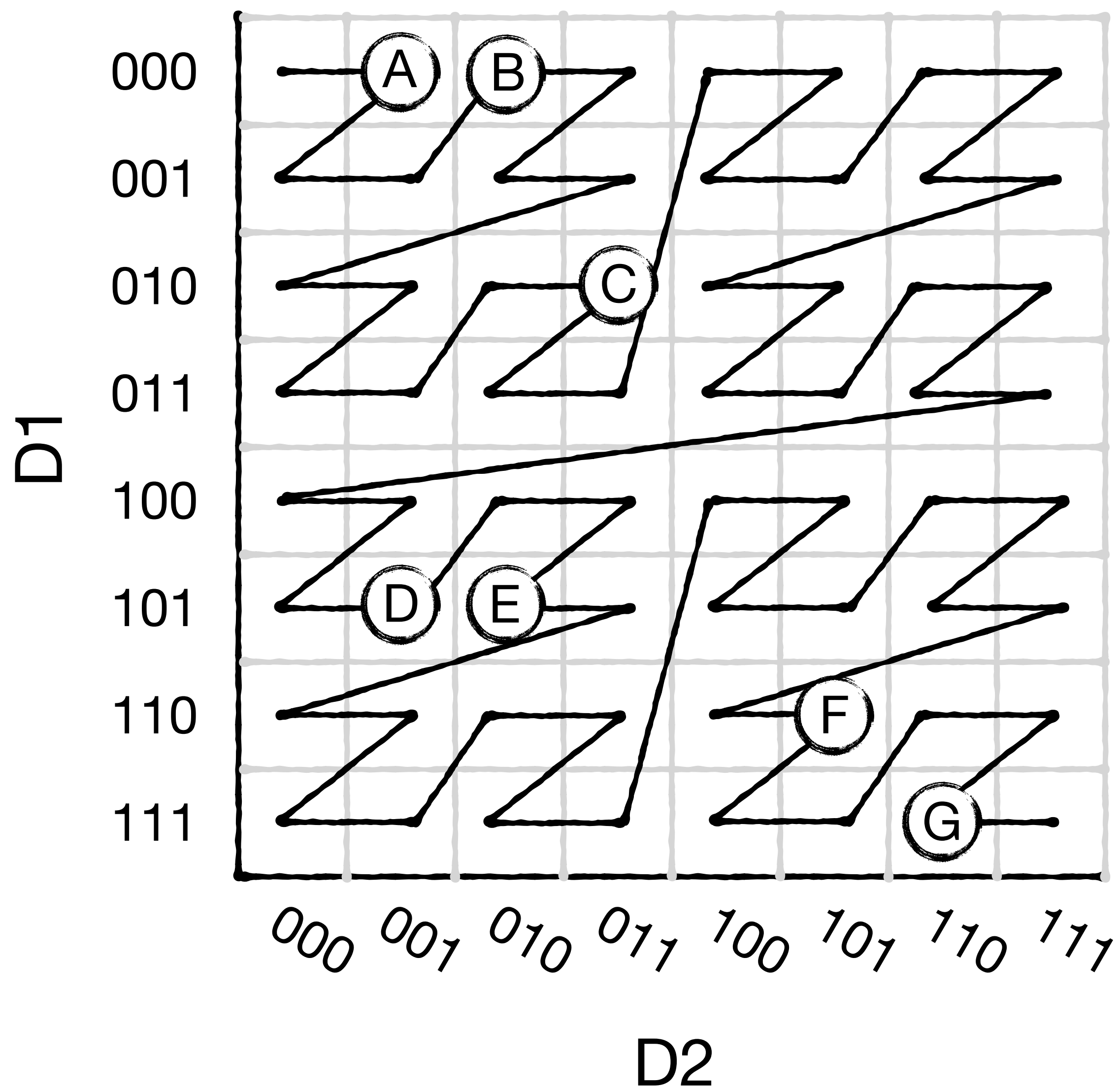


D1	000	000
D2	001	010
Z-Key	00 00 01	00 01
	A	B
	<?	

Interleaving dimension bits



D1	000	000	000
D2	001	001	010
Z-Key	00	00	01
	(A)	<?	(B)



Z-Key

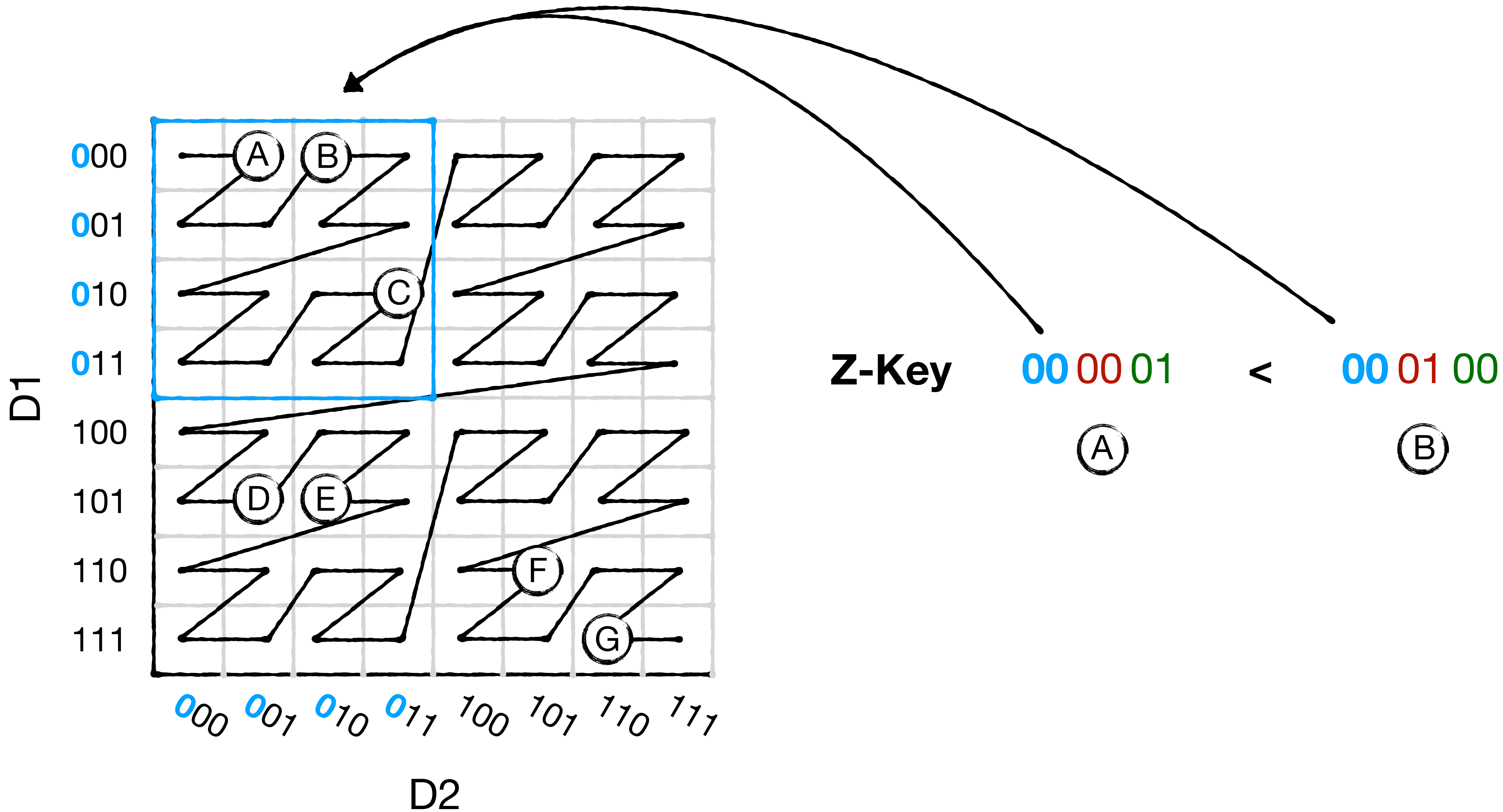
00 00 01

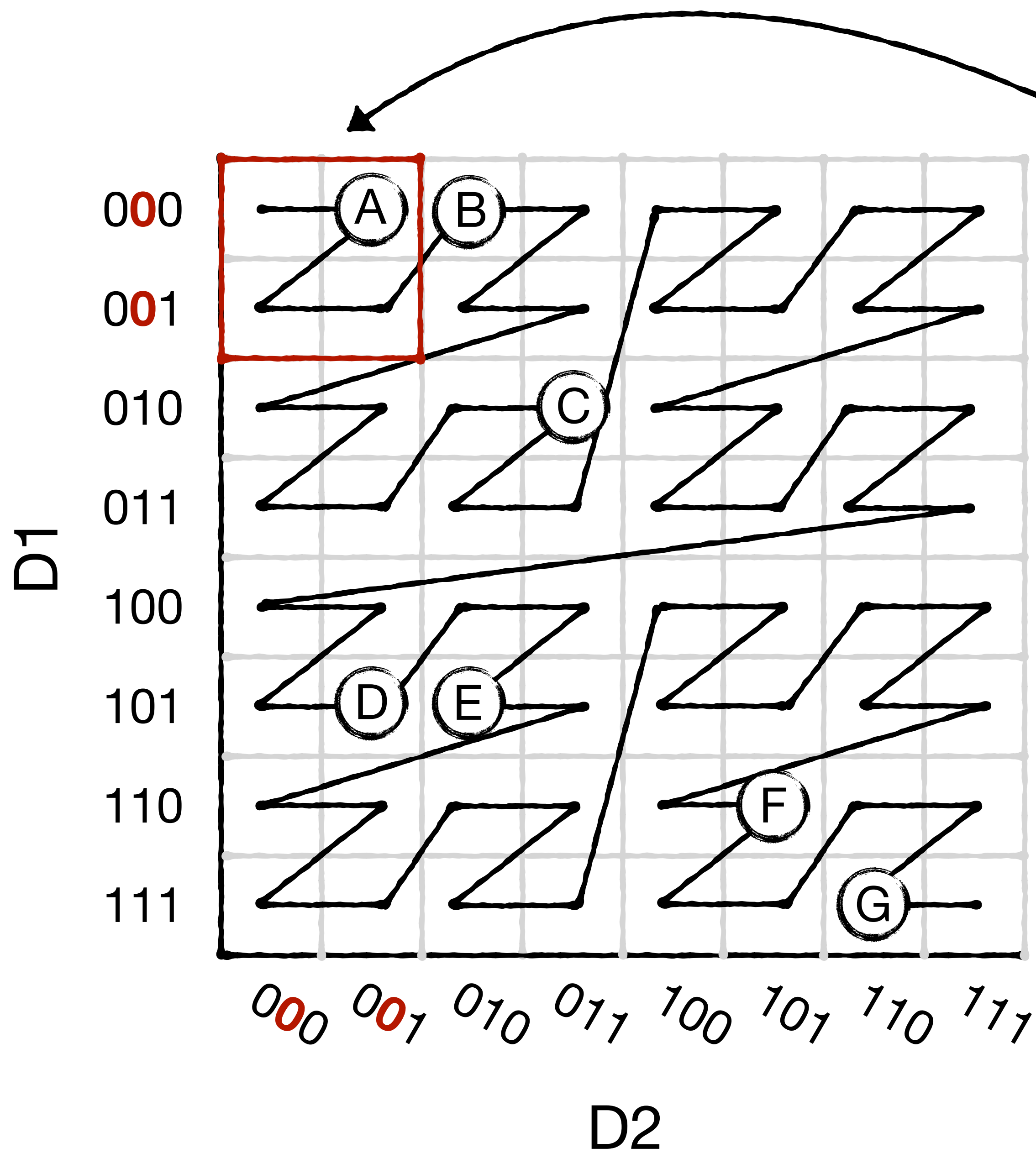
<

00 01 00

(A)

(B)





Z-Key

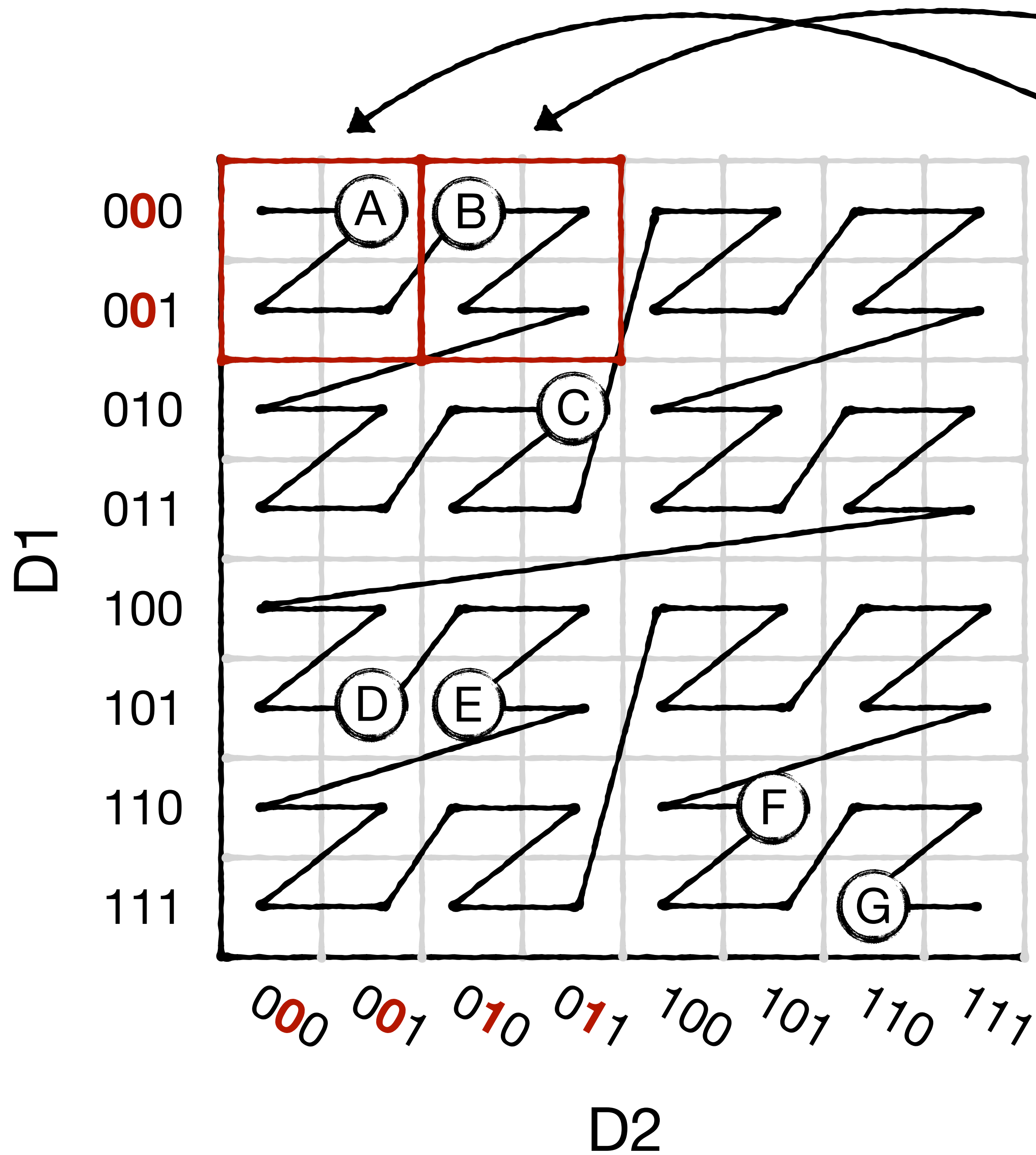
00 00 01

<

00 01 00

A

B



Z-Key

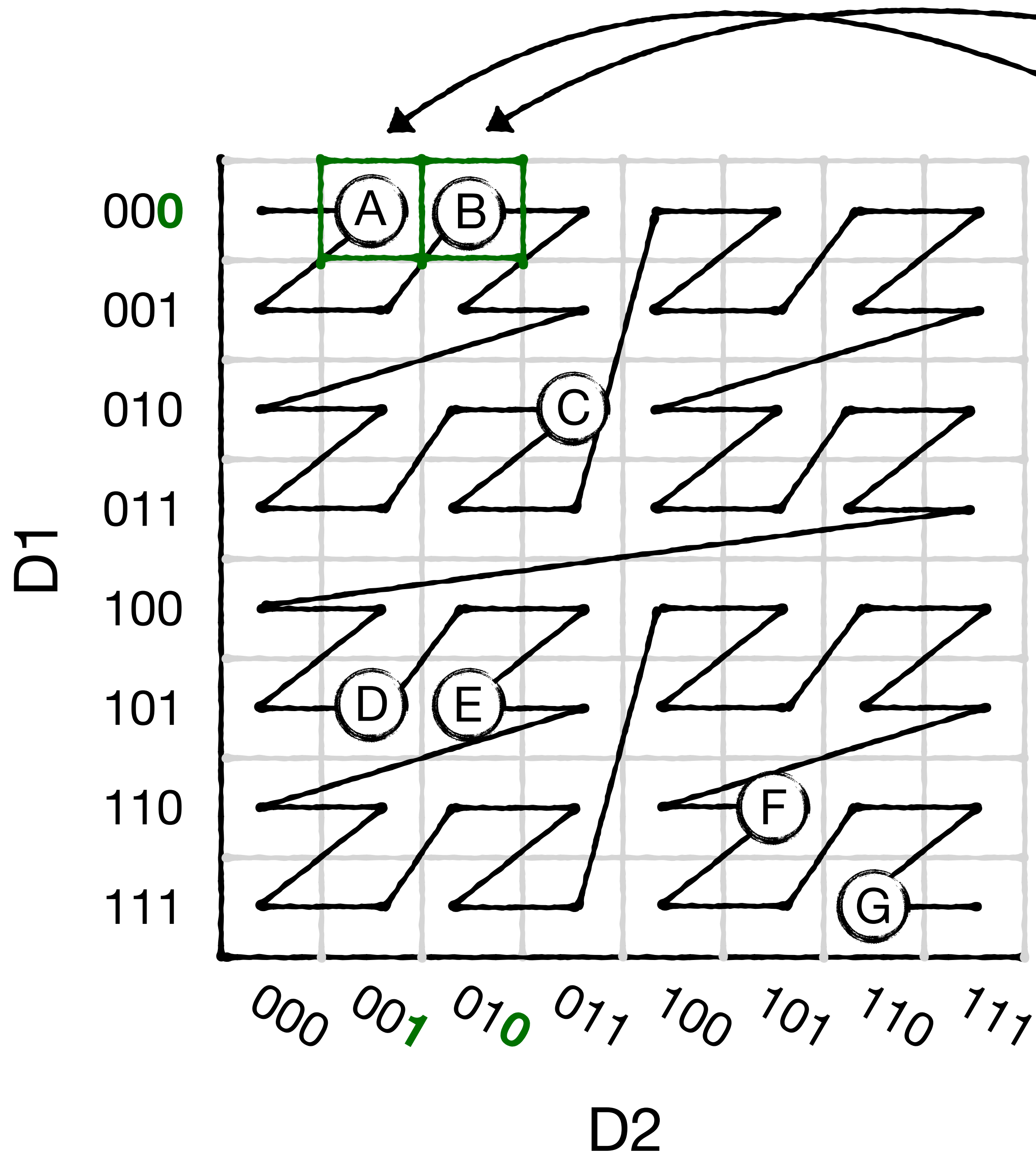
00 00 01

<

00 01 00

A

B



Z-Key

00 00 01

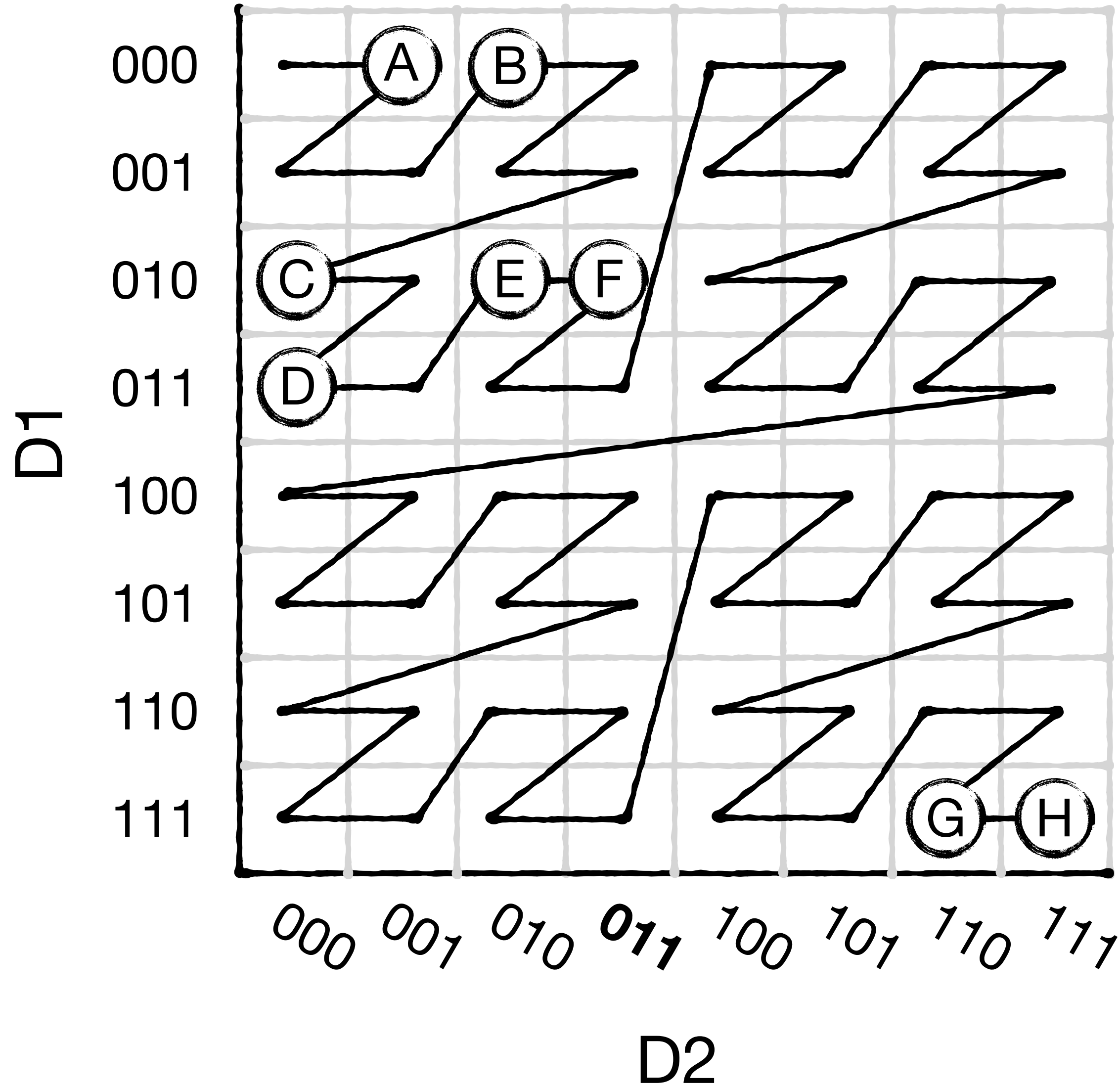
<

00 01 00

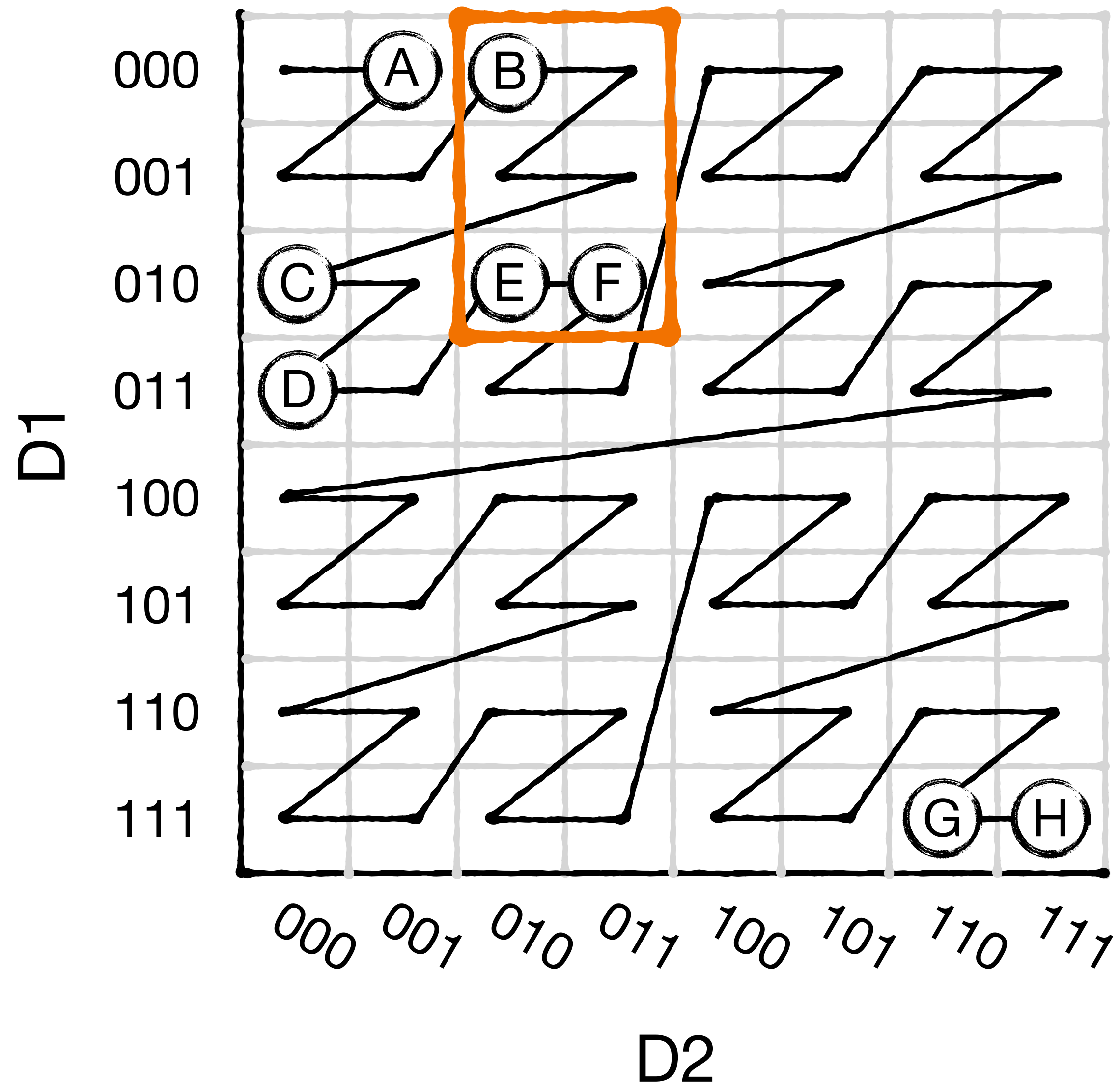
A

B

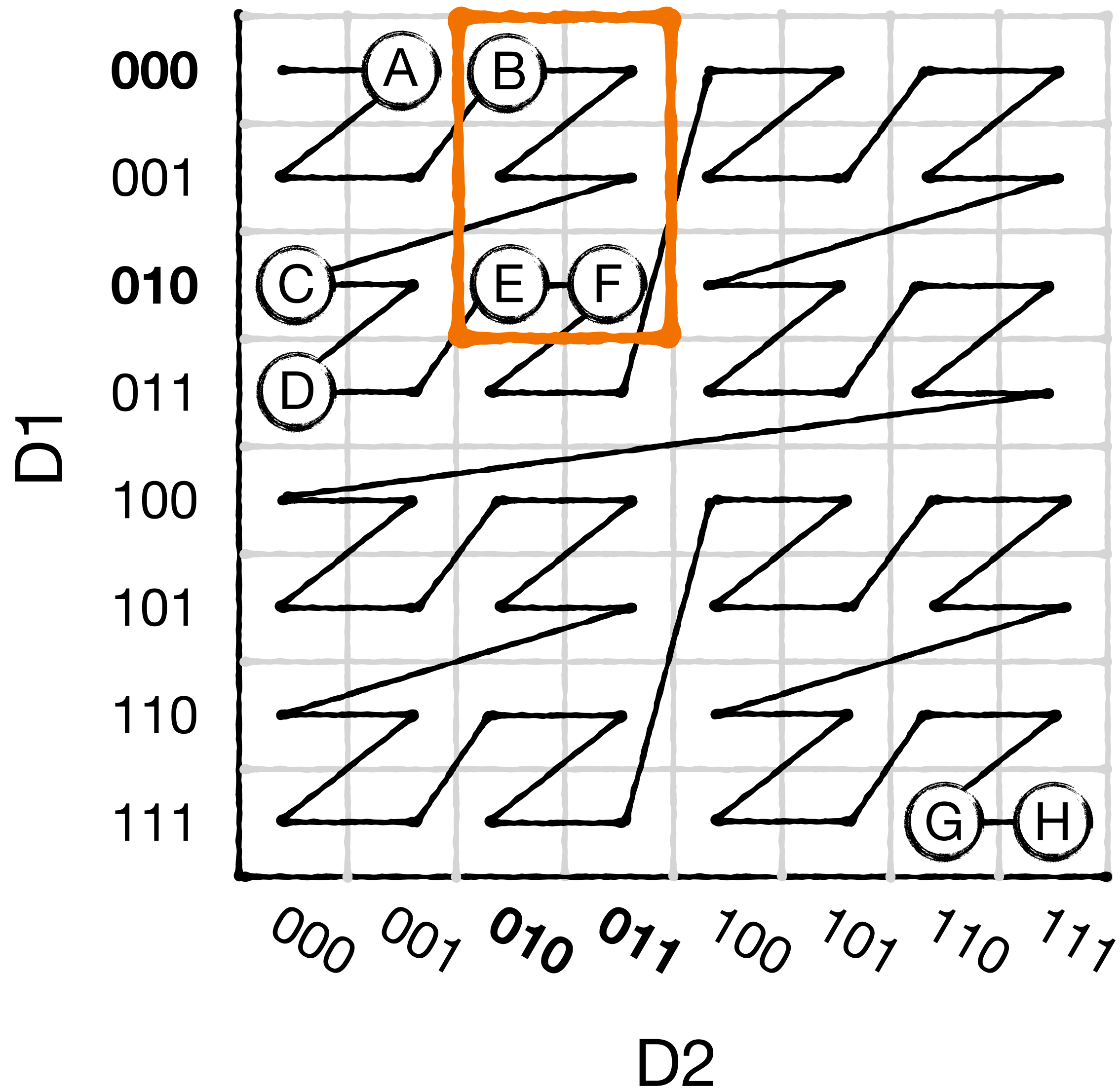
Query Handling



Query Handling



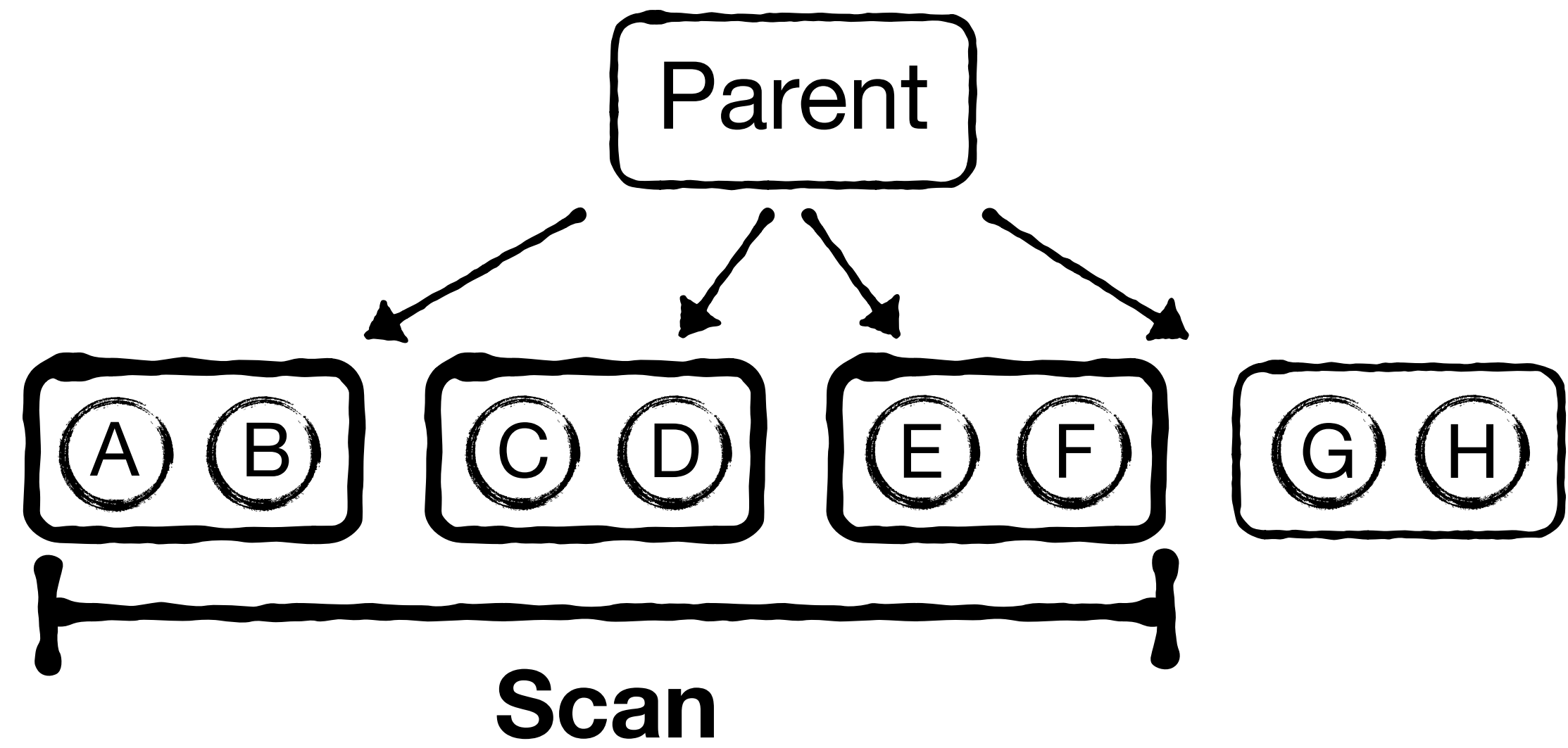
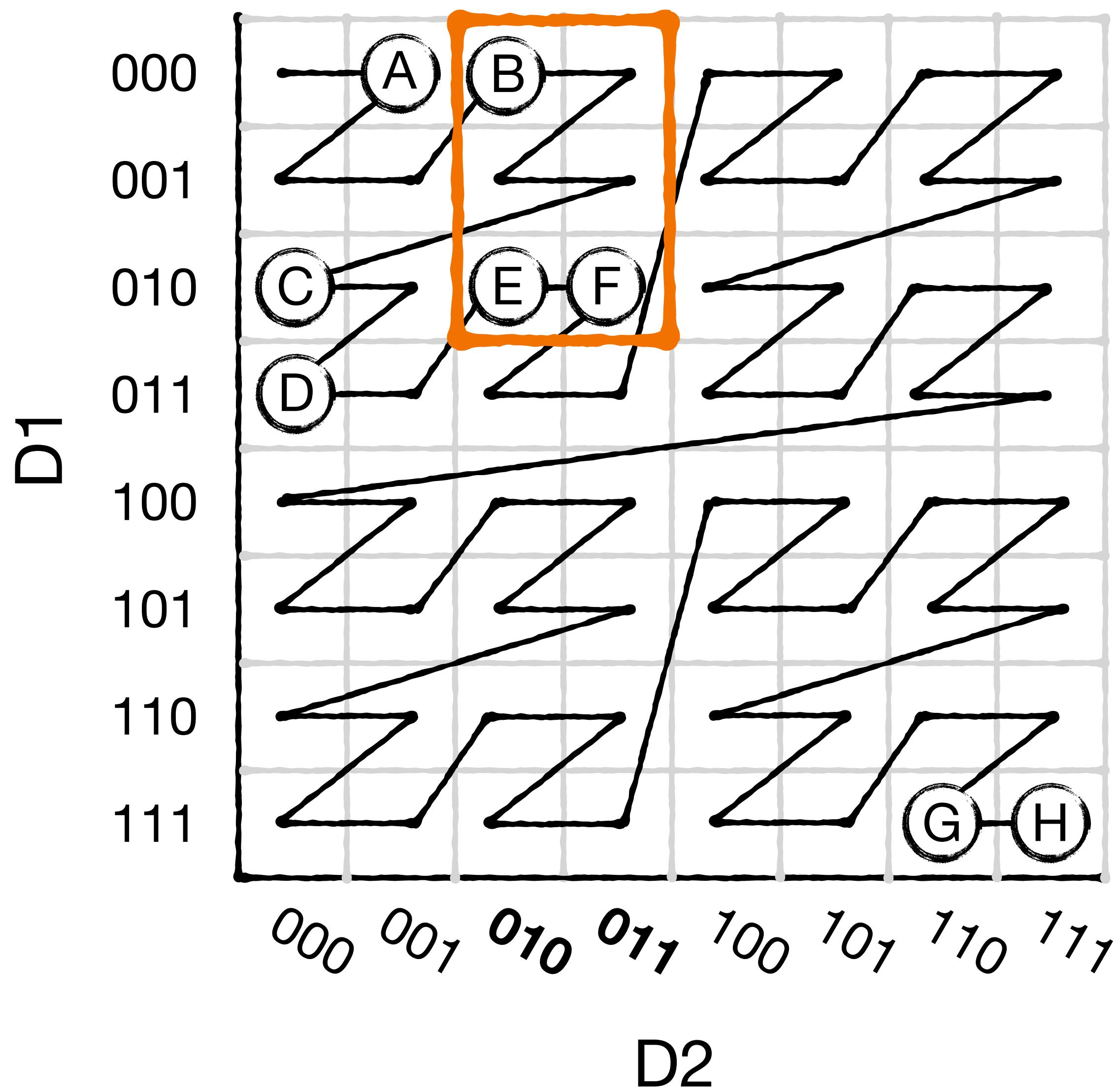
Query is a rectangle defined by start and end Z-key



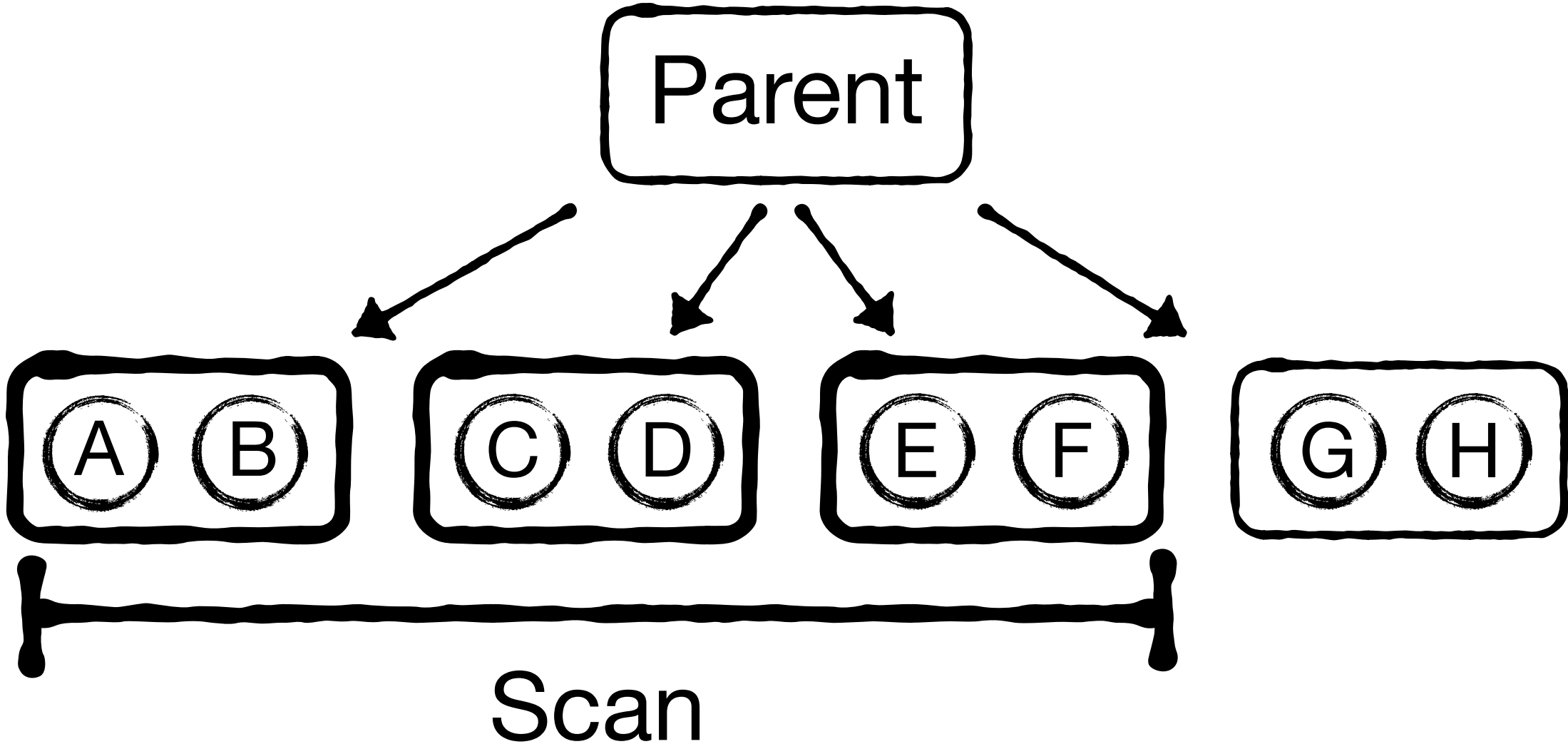
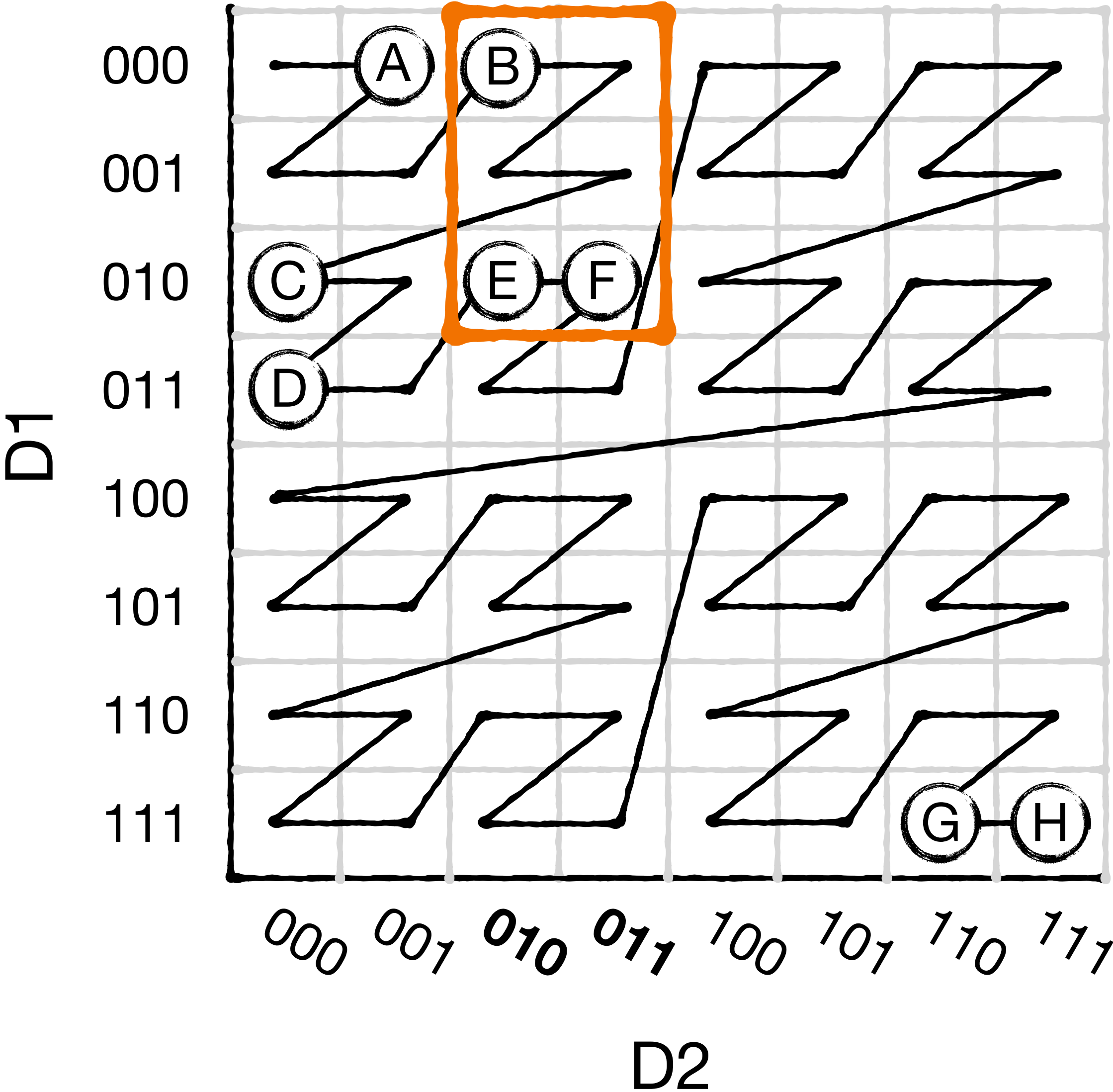
Example

start: 00 01 00

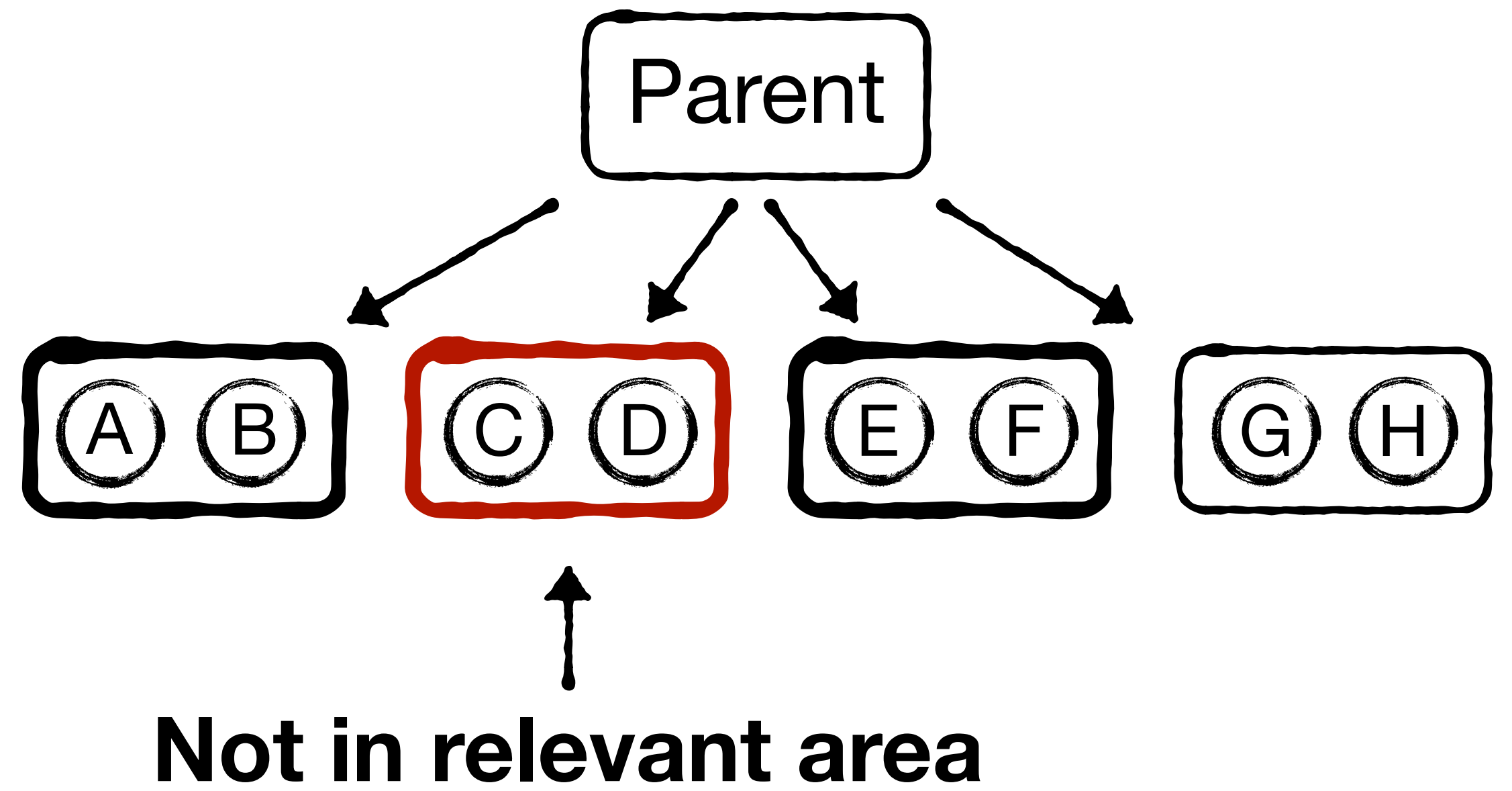
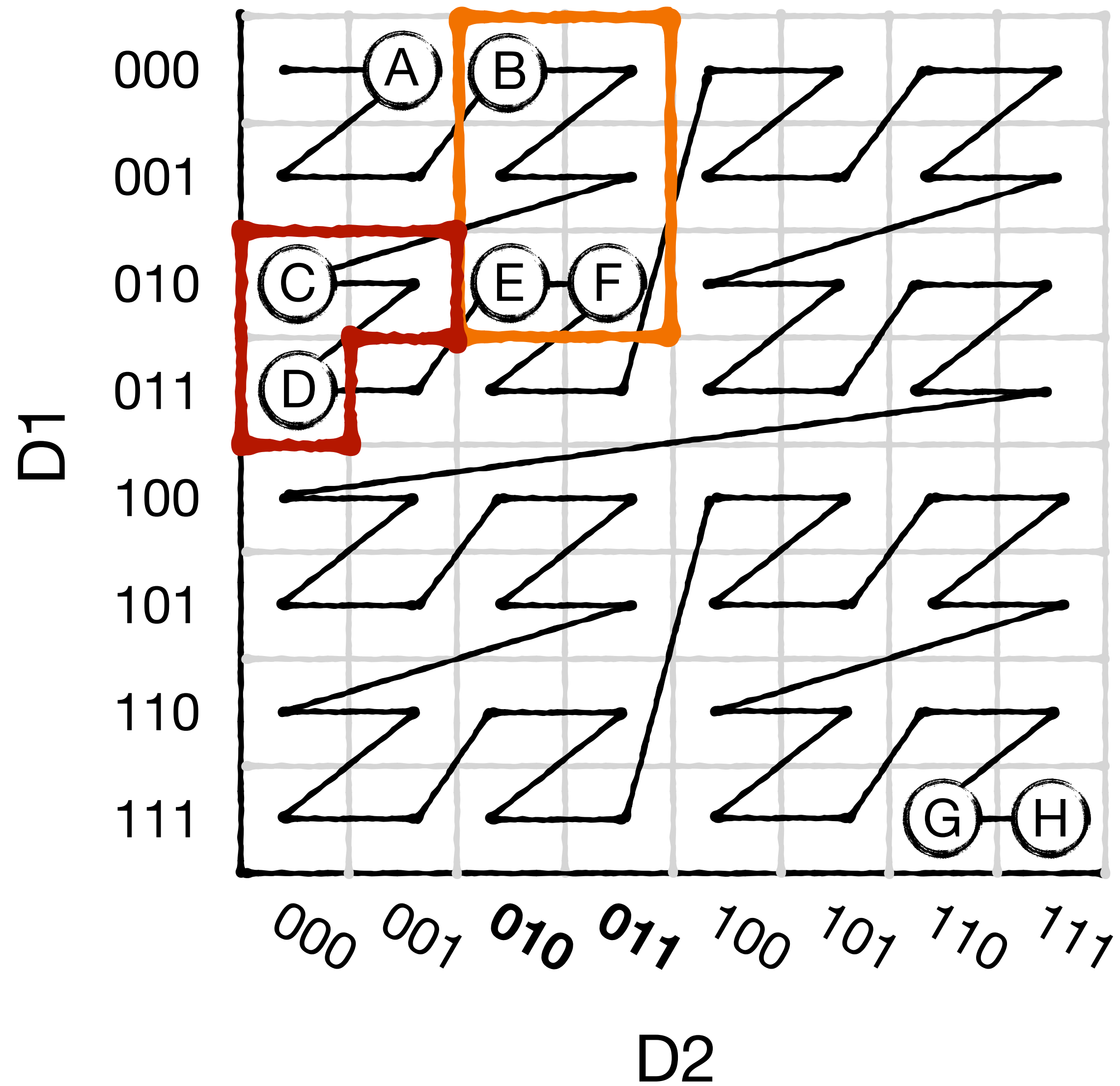
end: 00 11 01



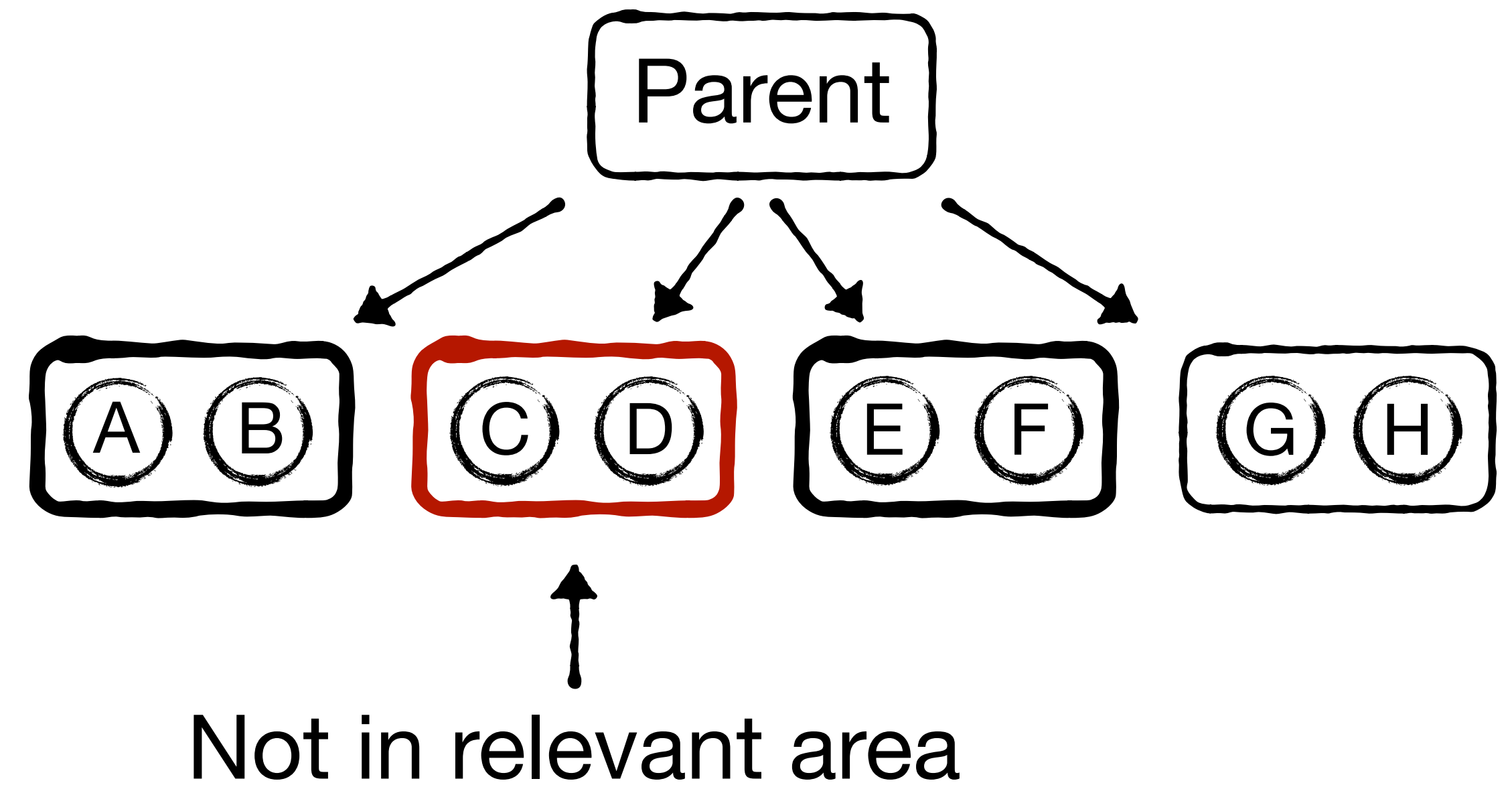
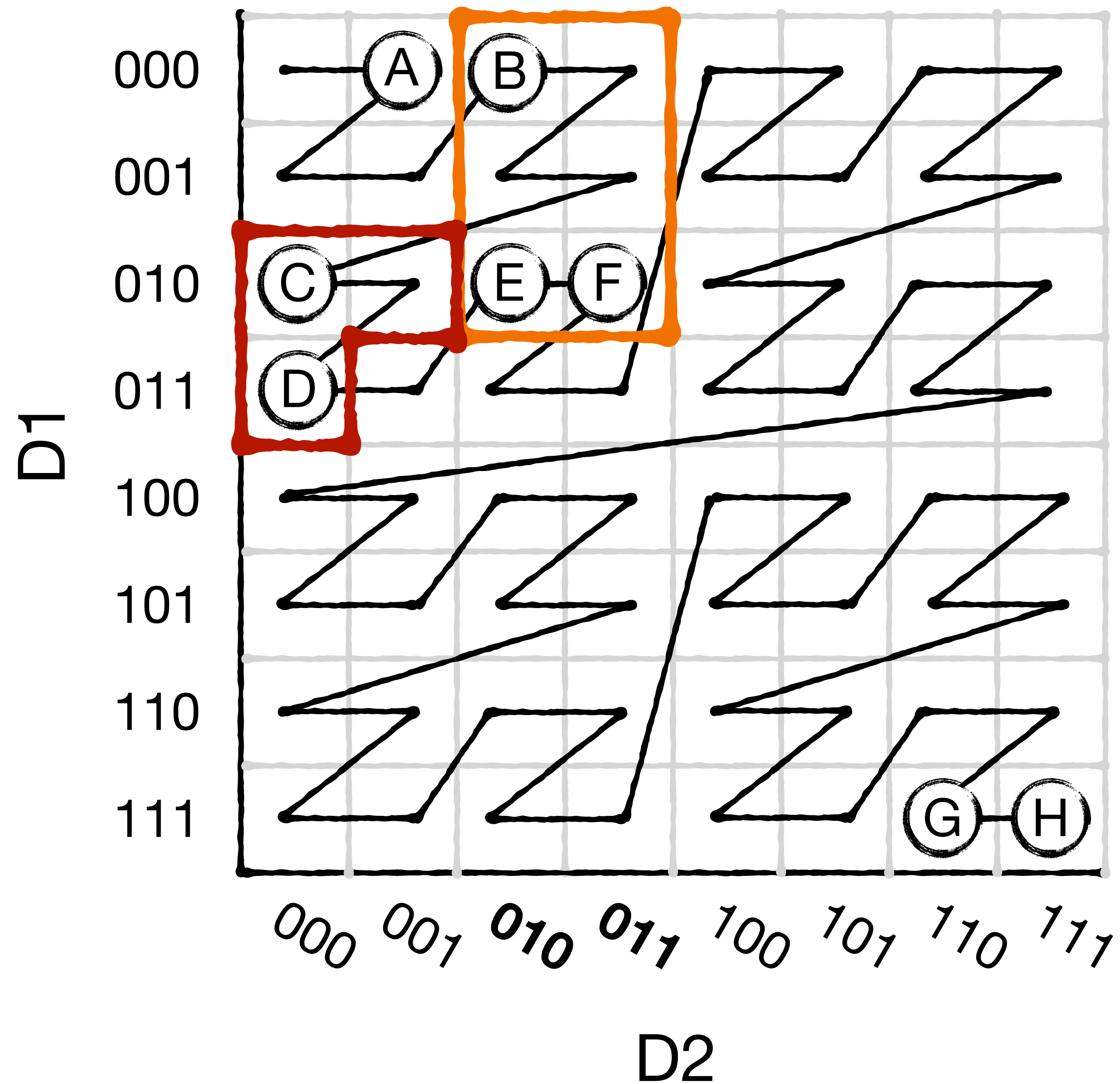
Issues?



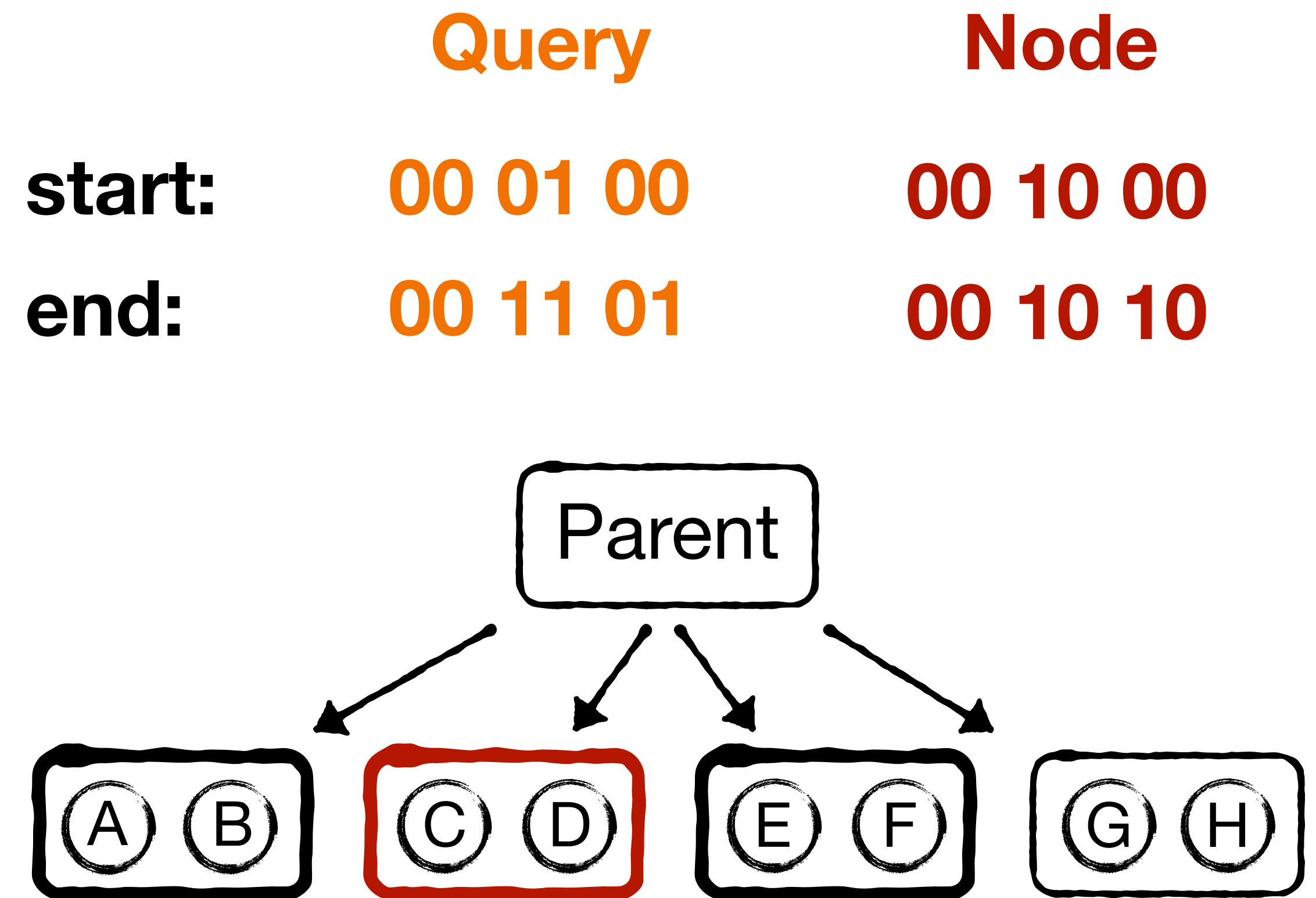
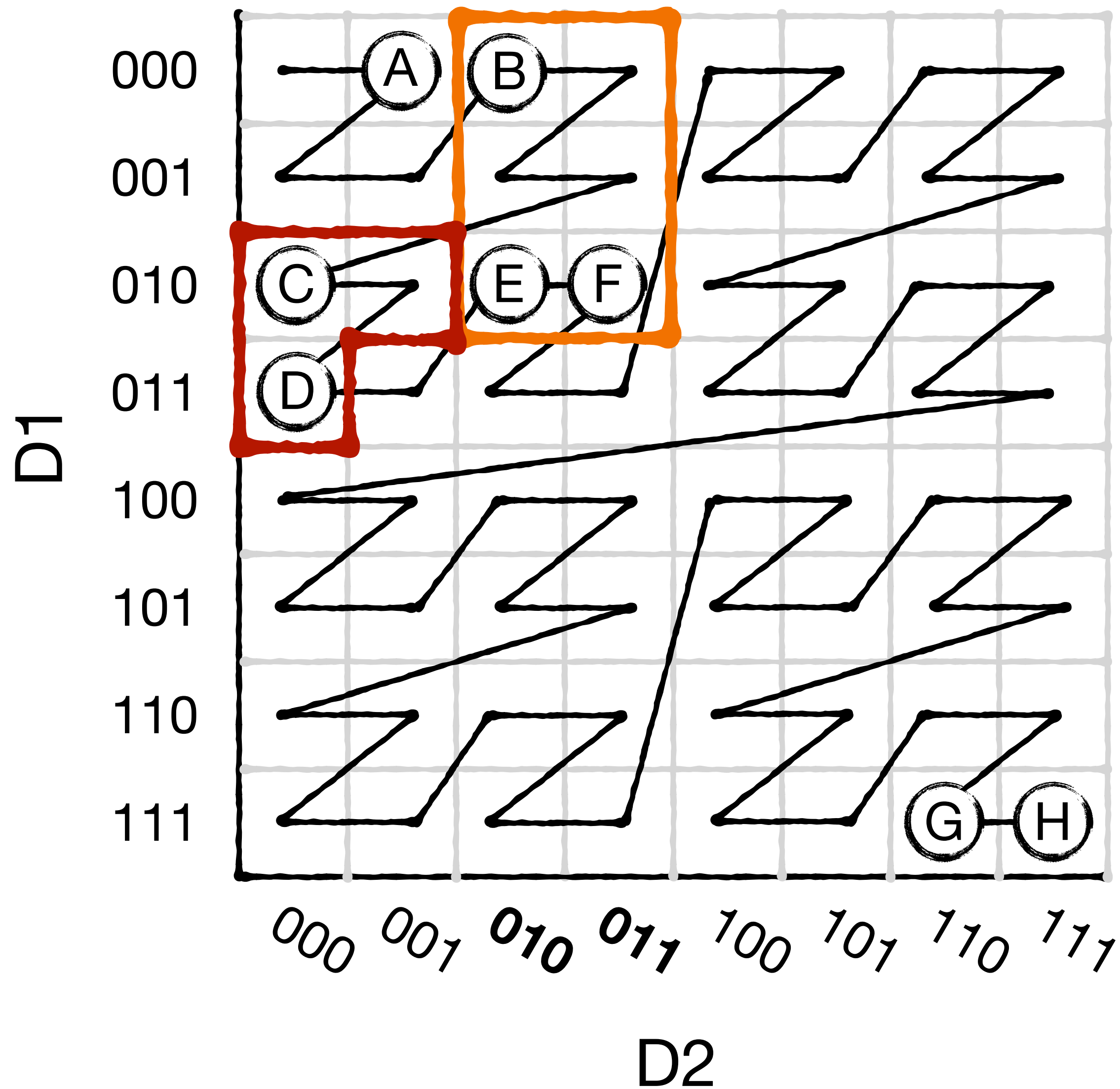
Issues?



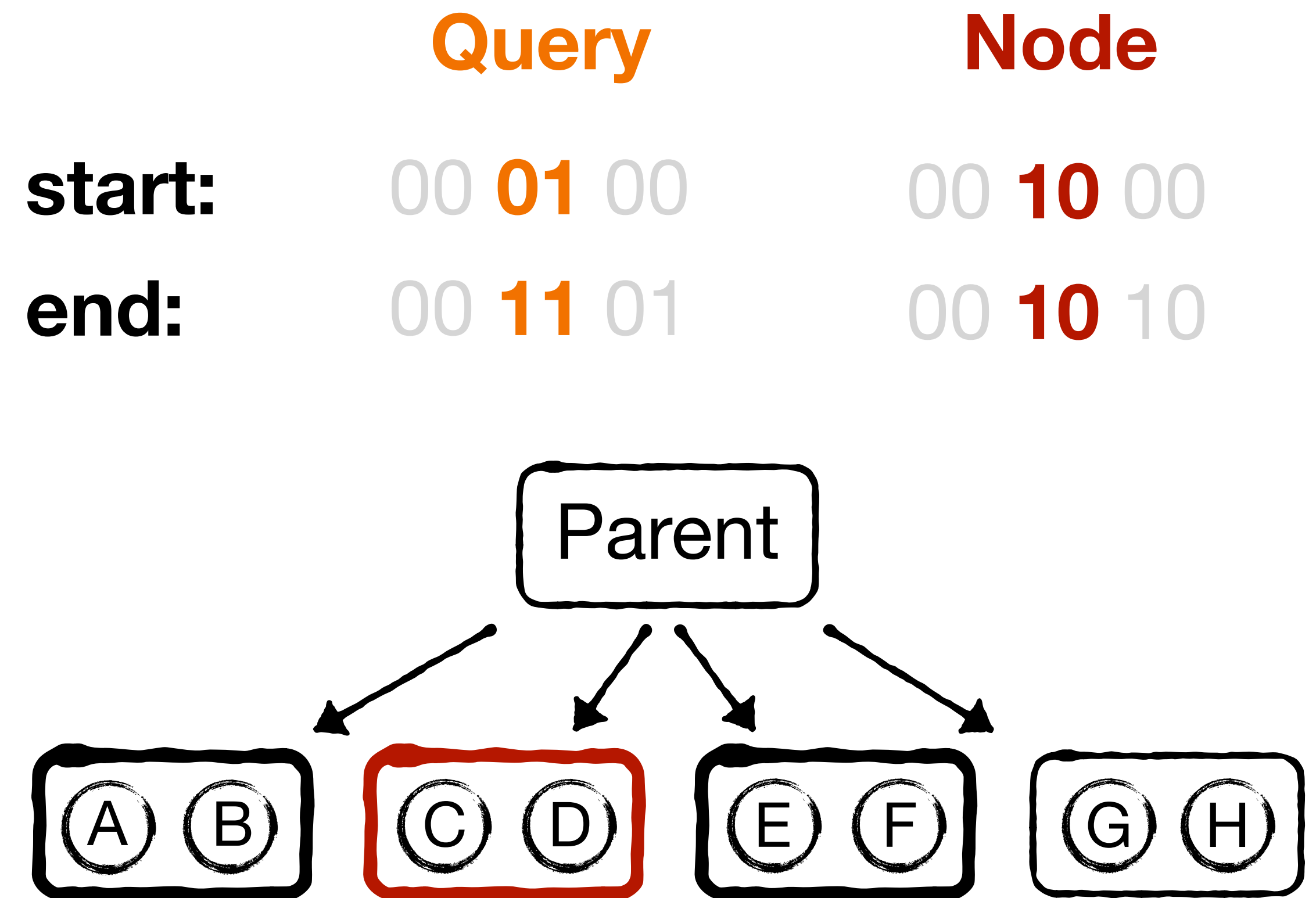
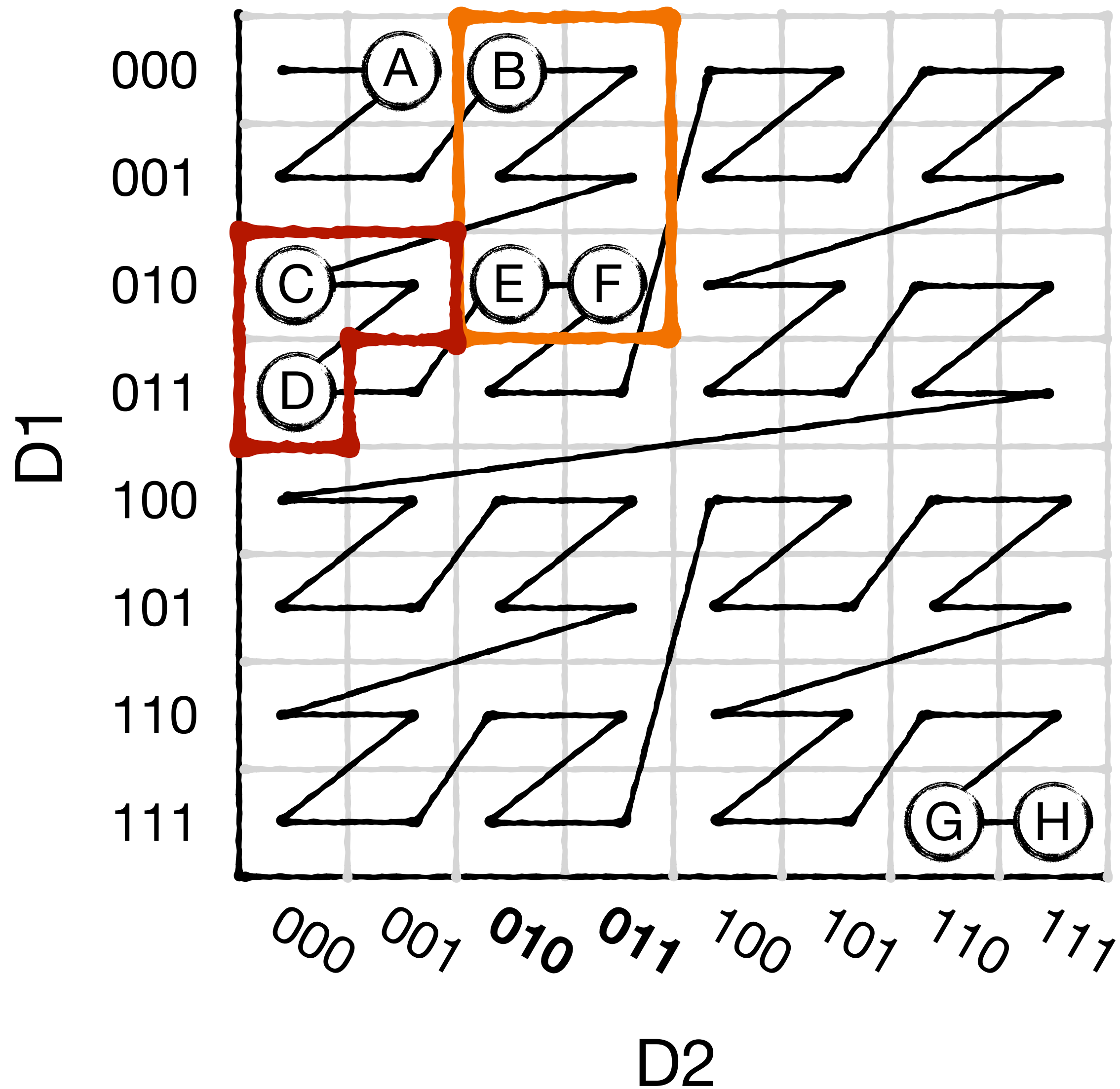
How to skip such nodes to speed up search?



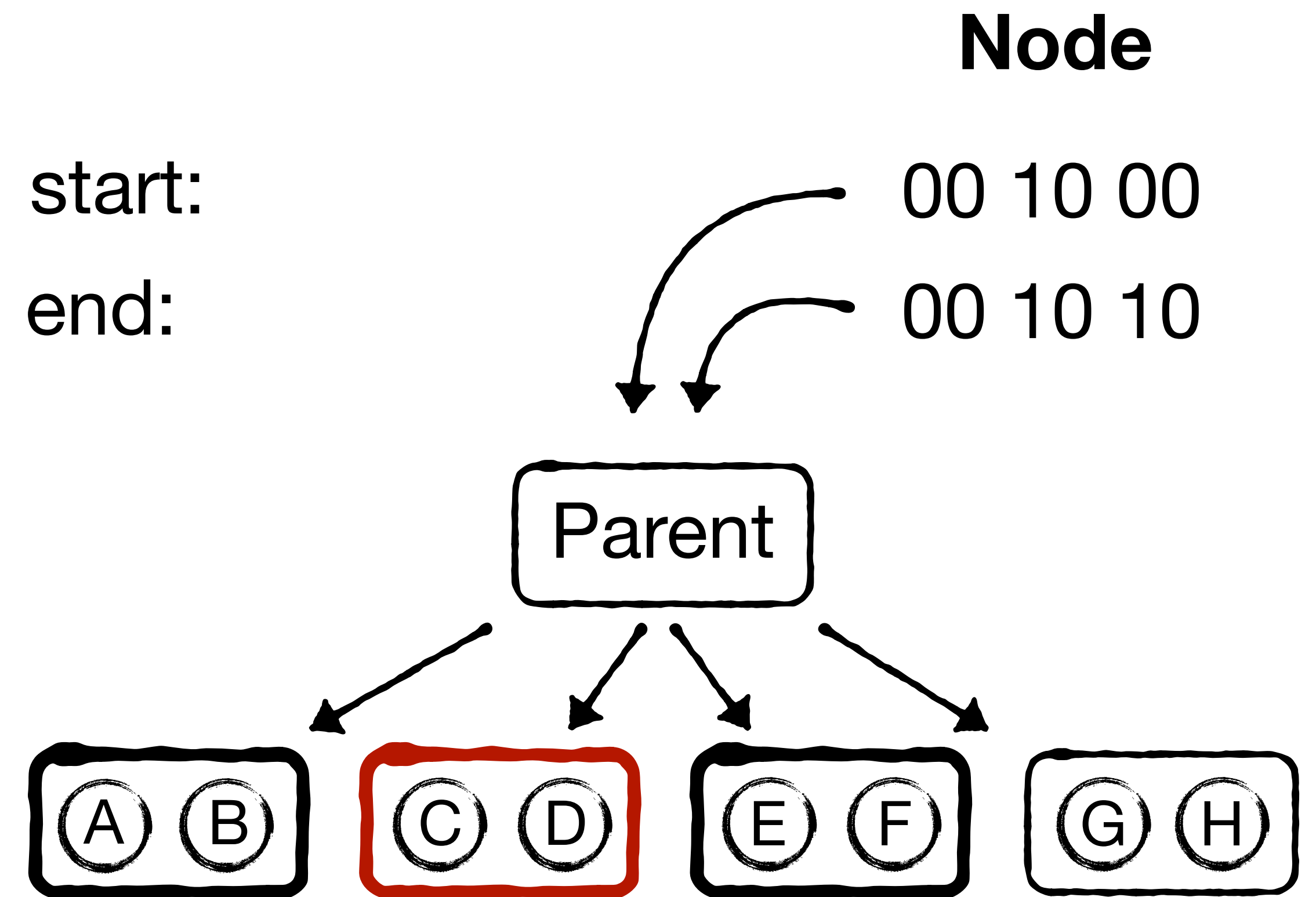
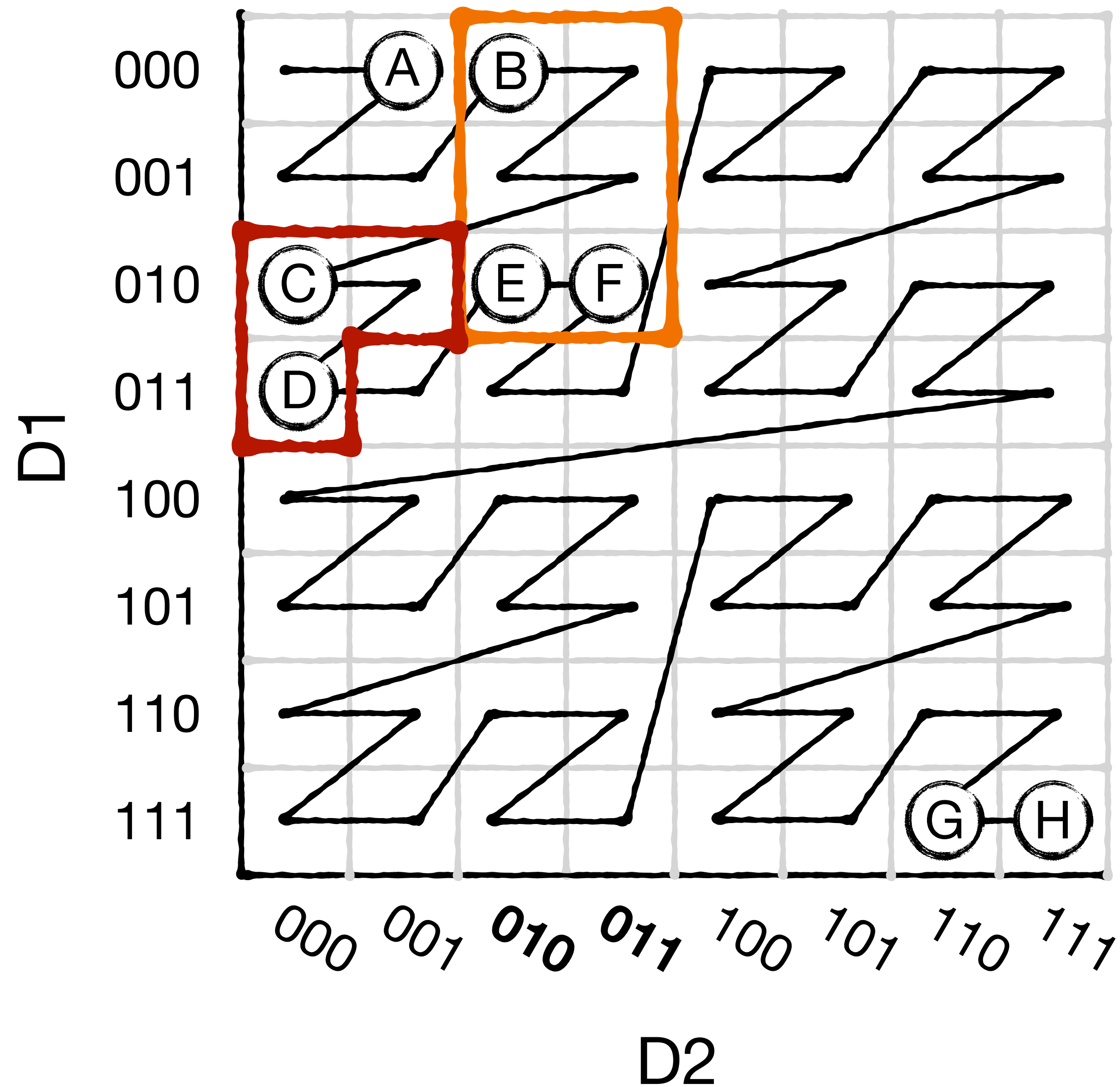
How to skip such nodes to speed up search?



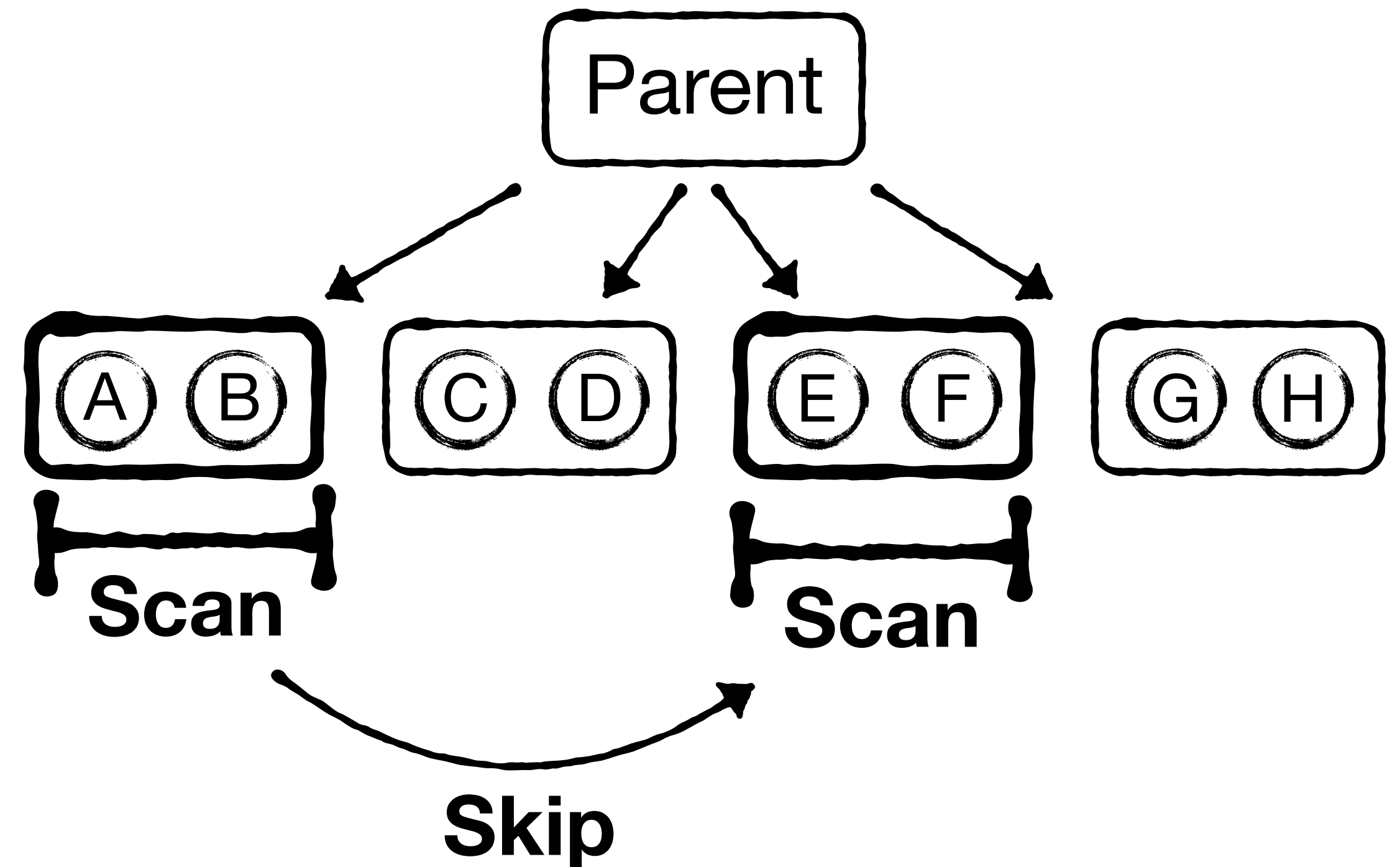
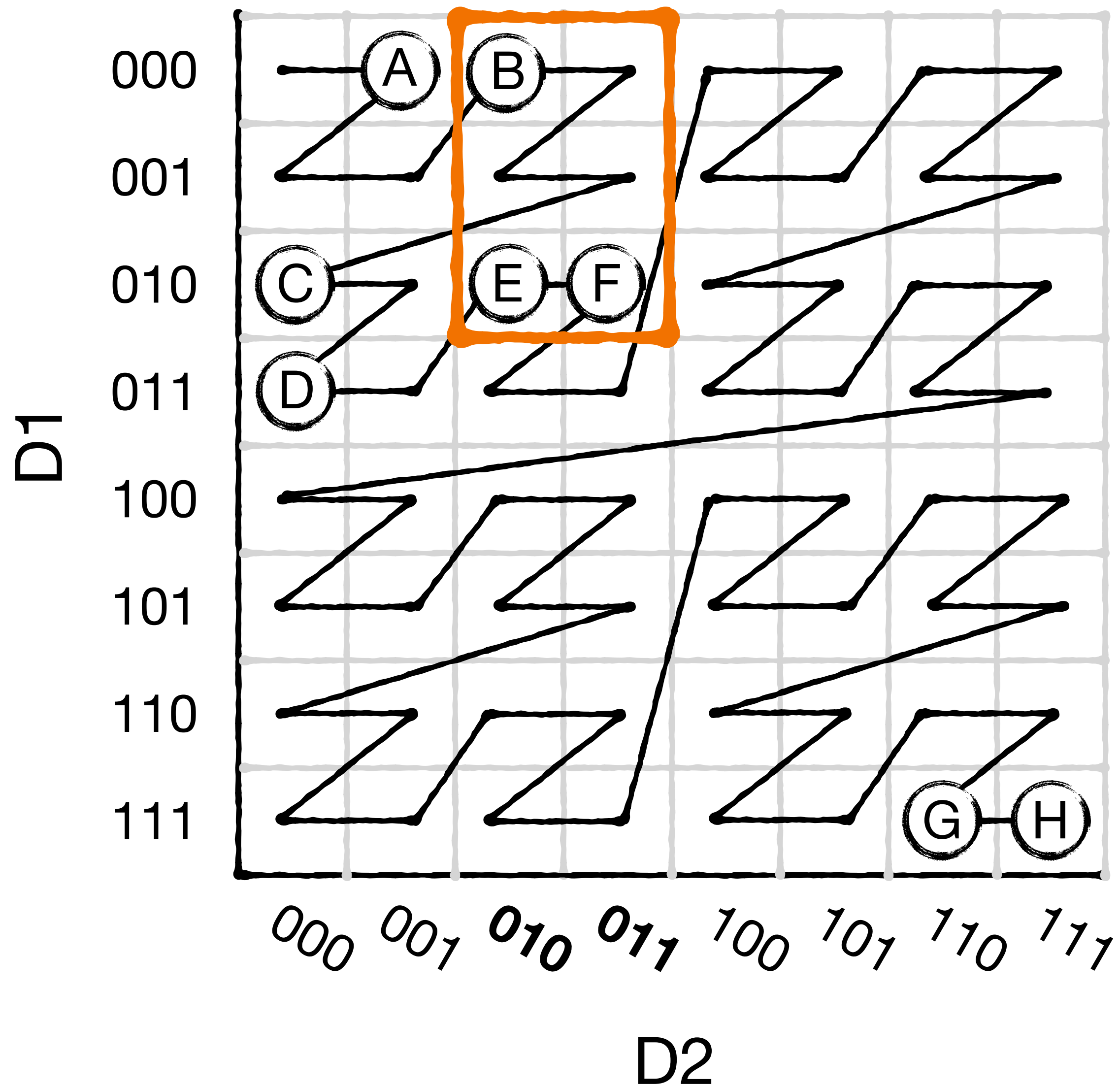
How to skip such nodes to speed up search?



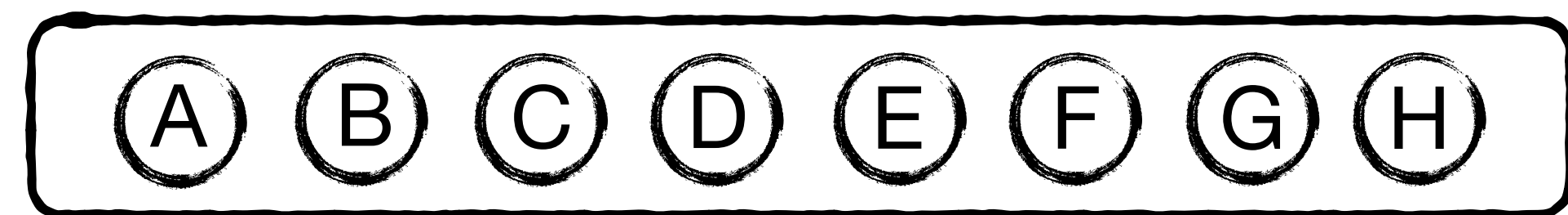
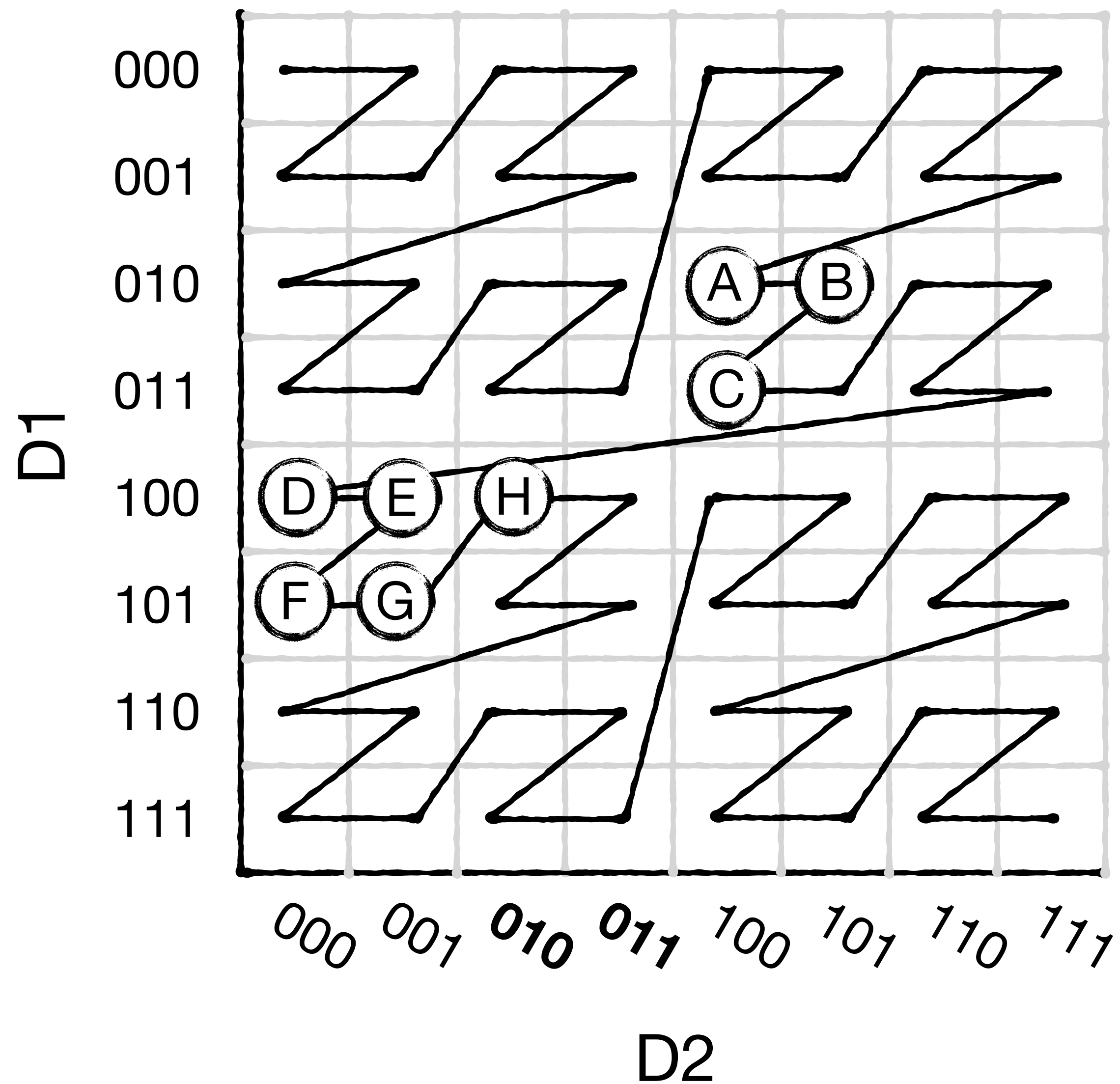
If parent contains start & end of each node, we can infer what to skip



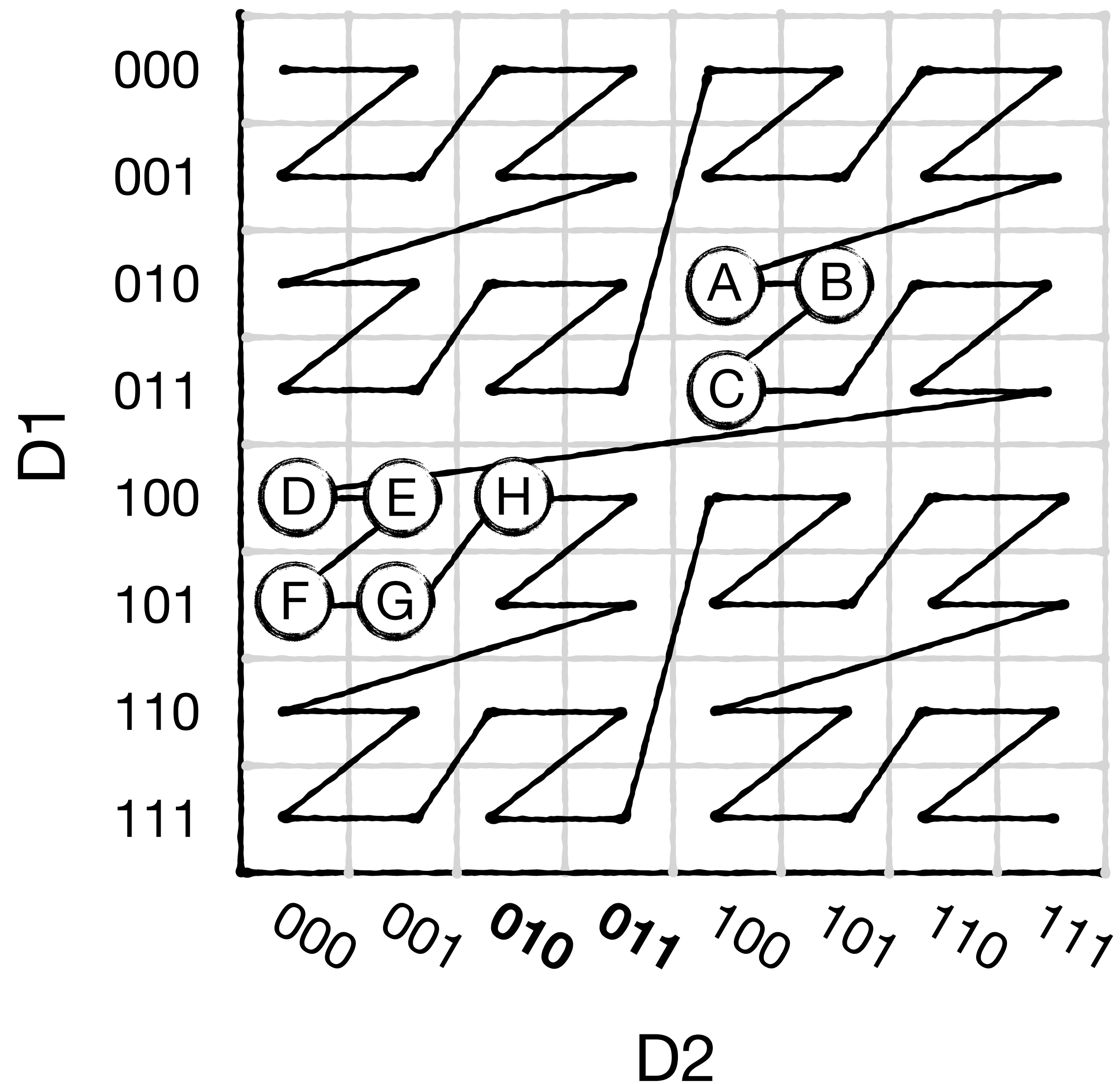
If parent contains start & end of each node, we can infer what to skip



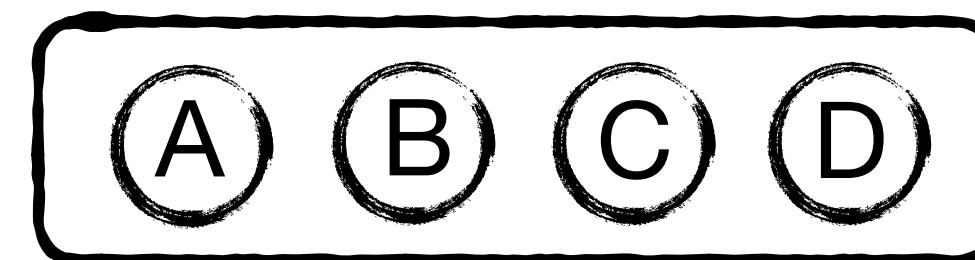
Consider the following node, which is full and about to split



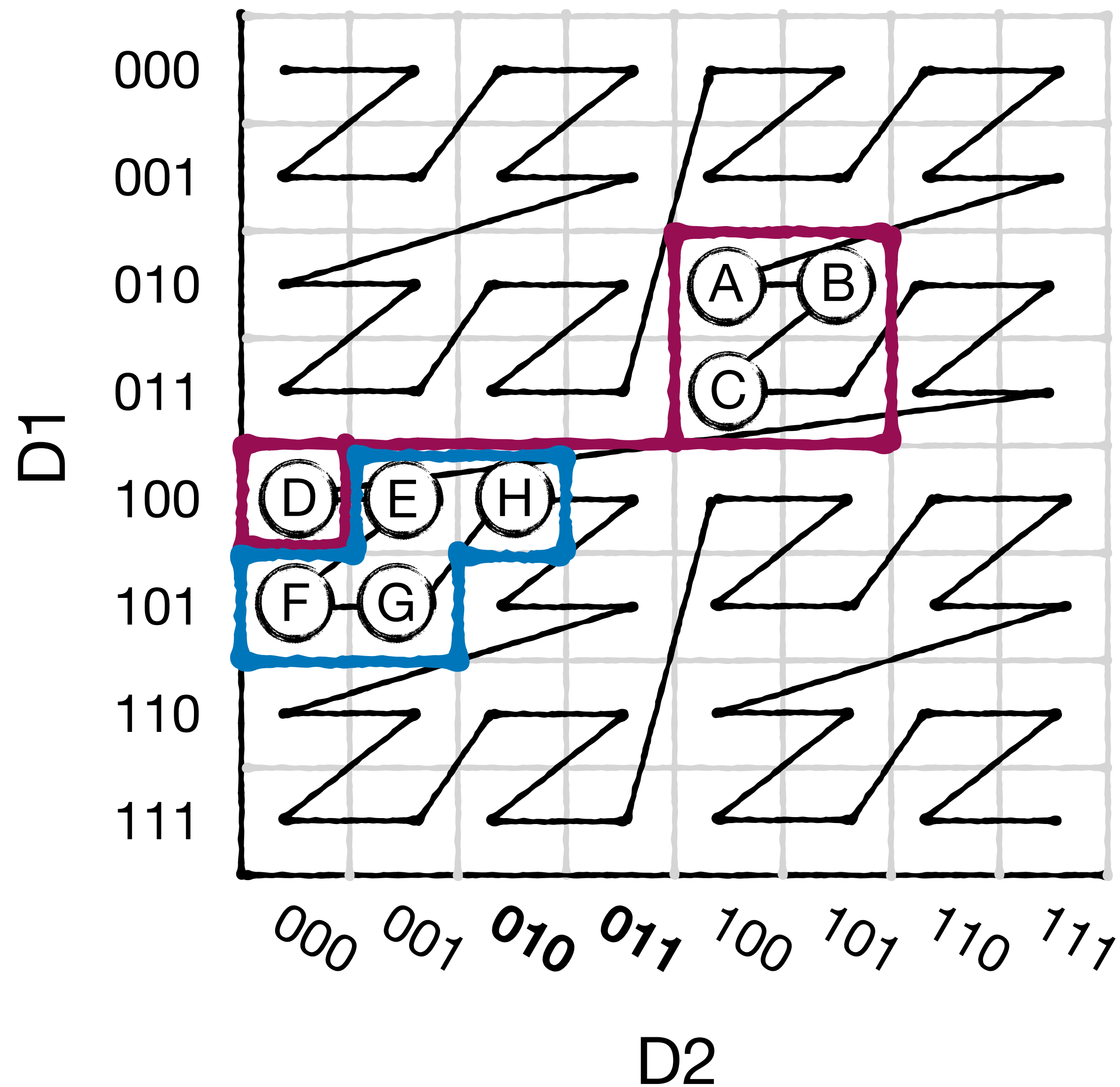
Consider the following node, which is full and about to split



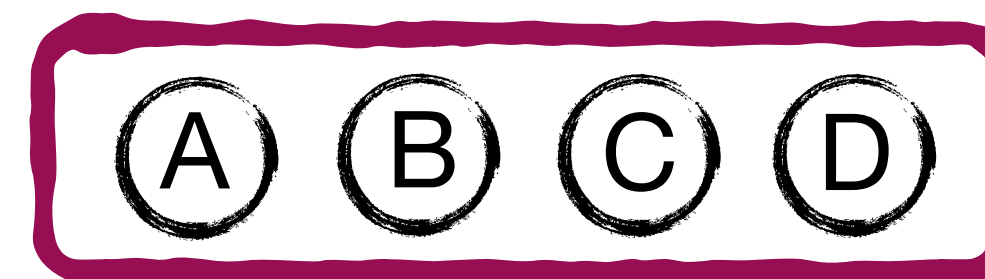
What's wrong with a fifty-fifty split?



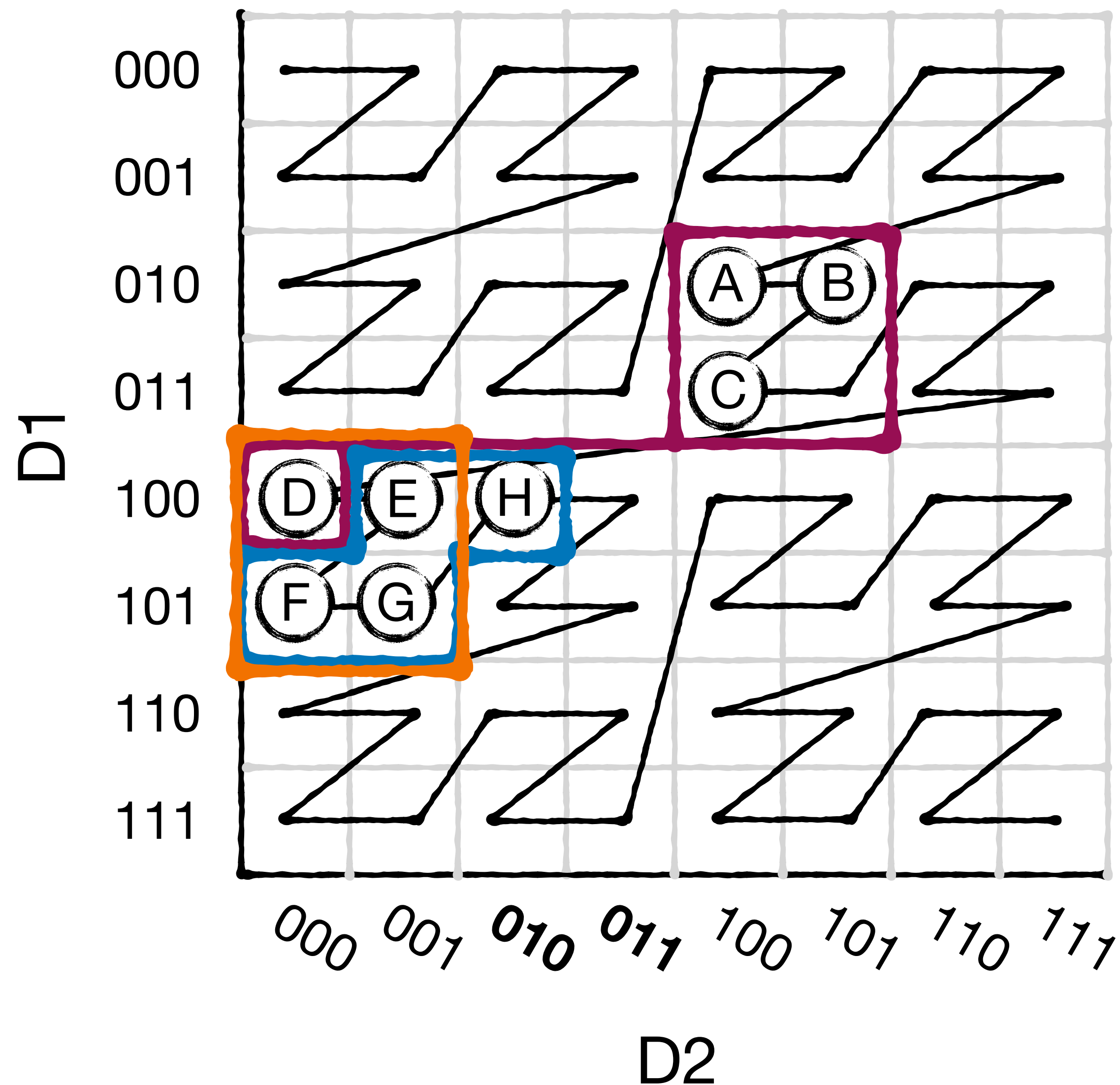
Consider the following node, which is full and about to split



What's wrong with a fifty-fifty split?



Consider the following node, which is full and about to split

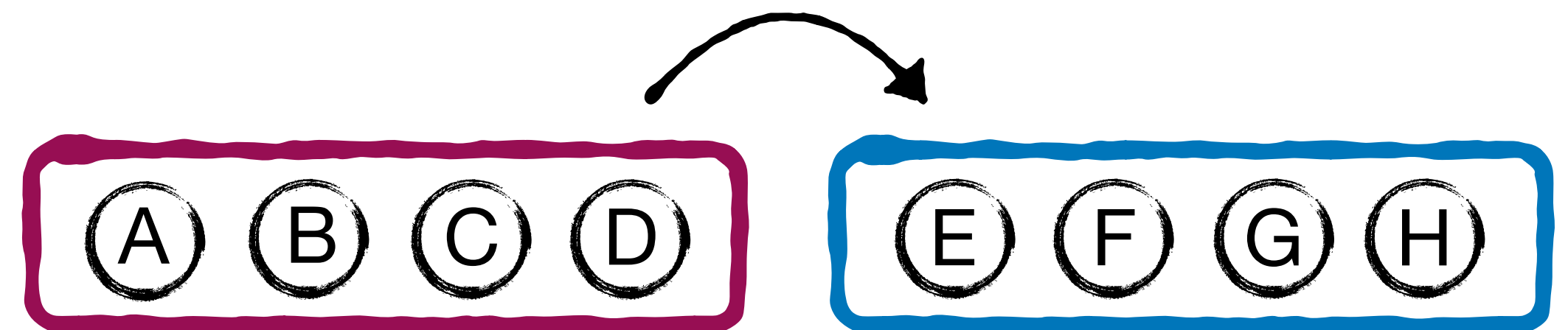
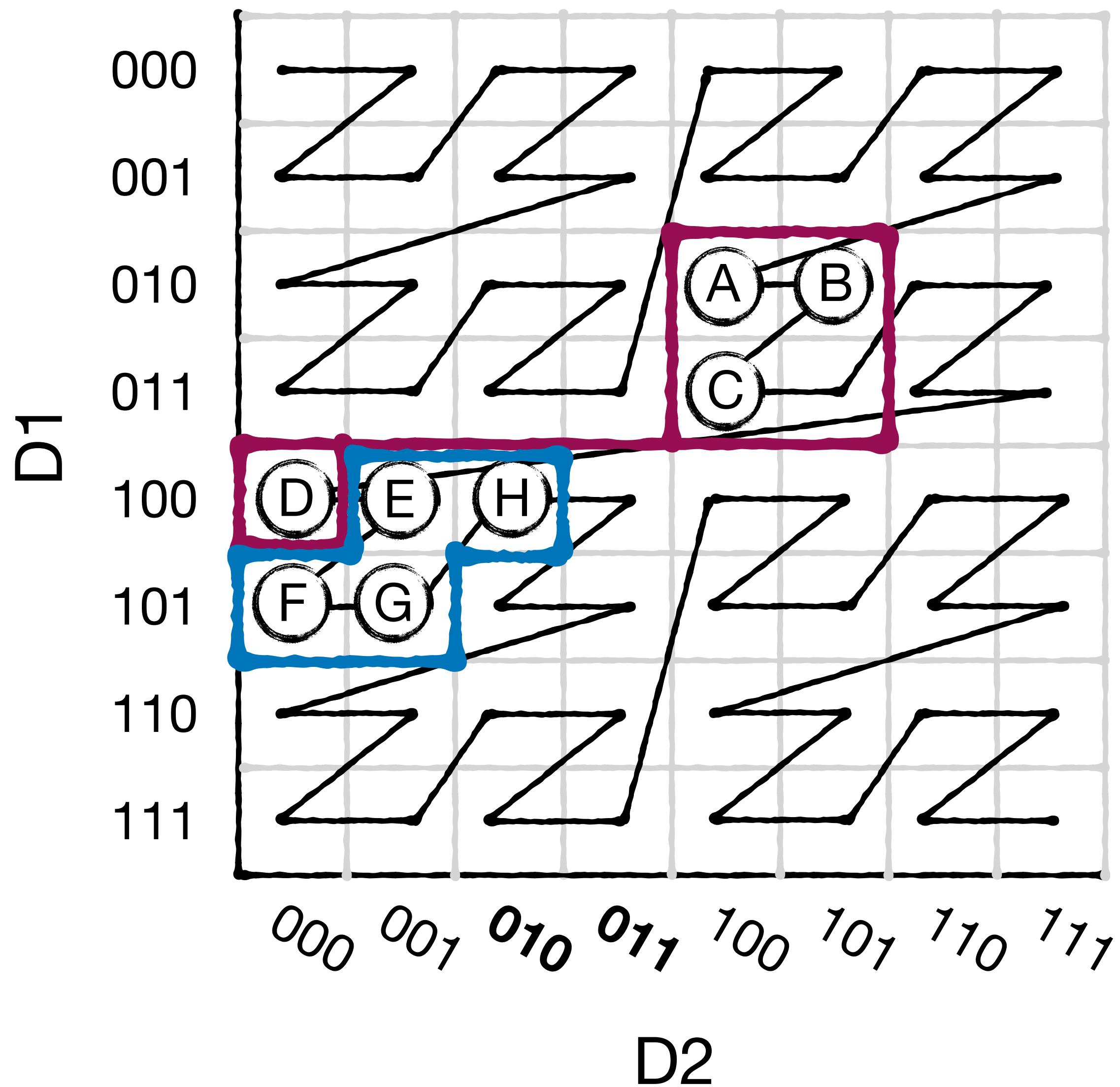


What's wrong with a fifty-fifty split?

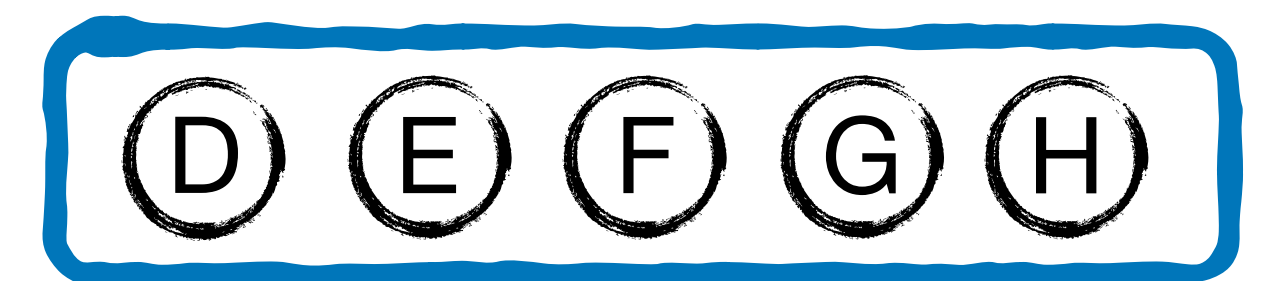
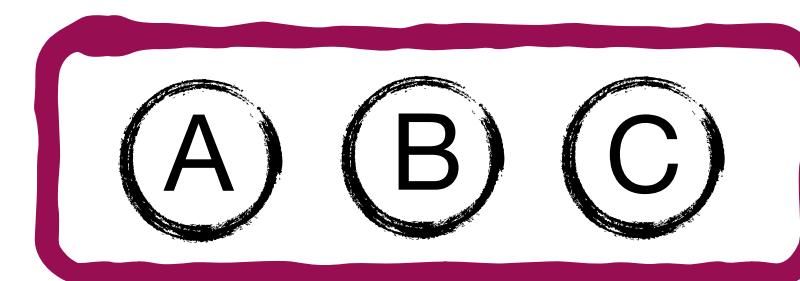
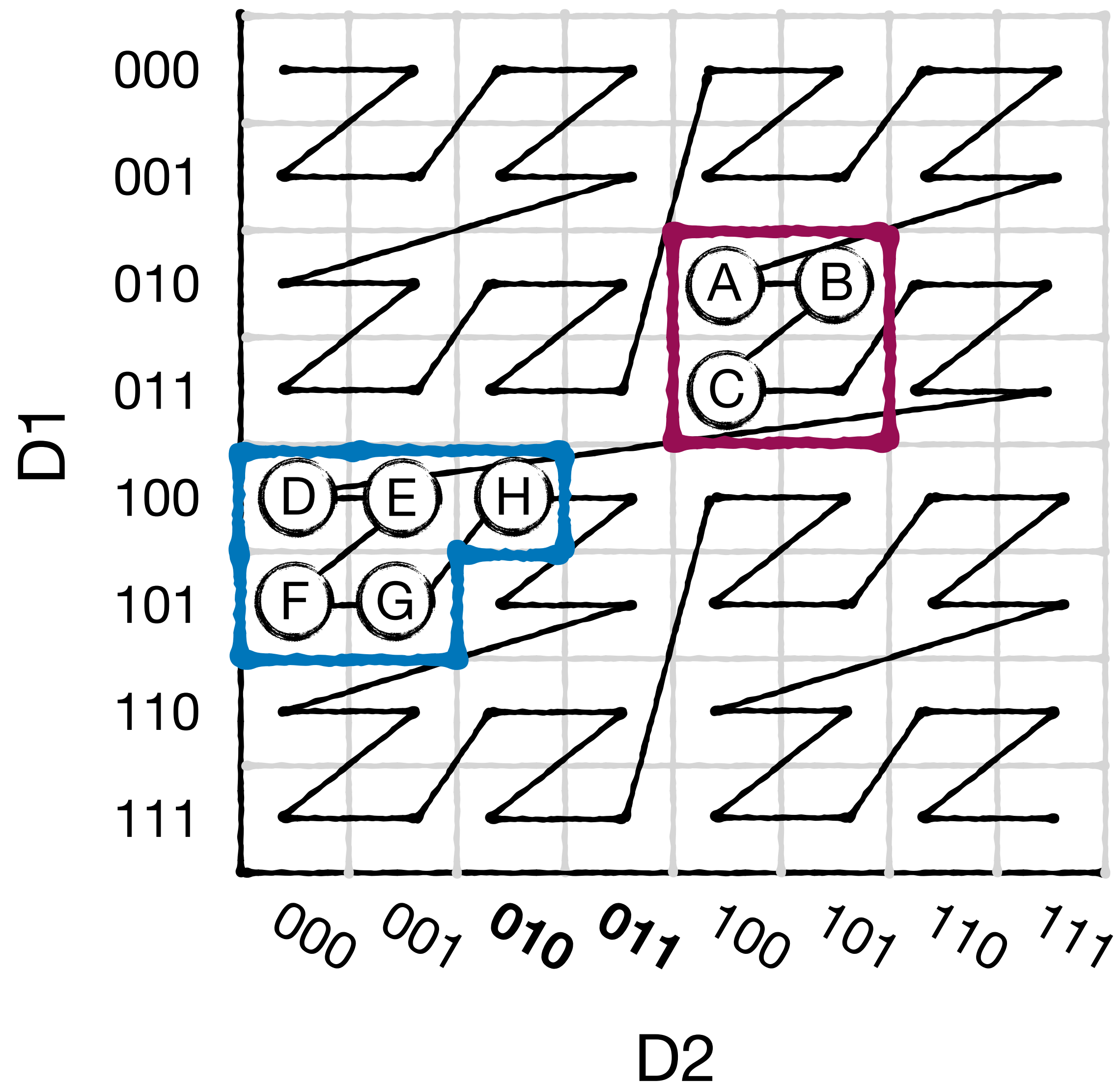


Queries are more likely to intersect with irregular node shapes

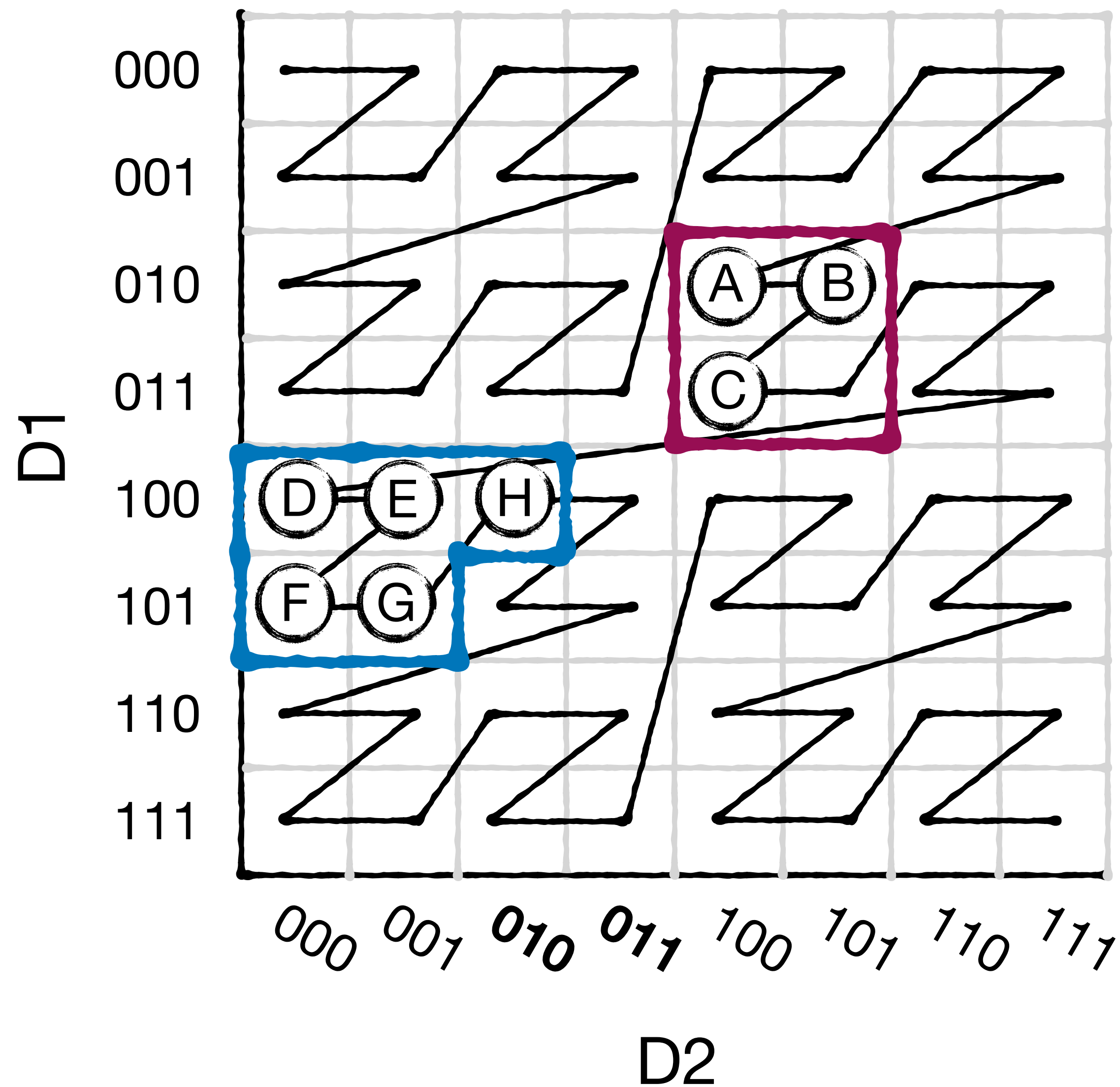
Insight: Picking some other close splitting point alleviates the issue



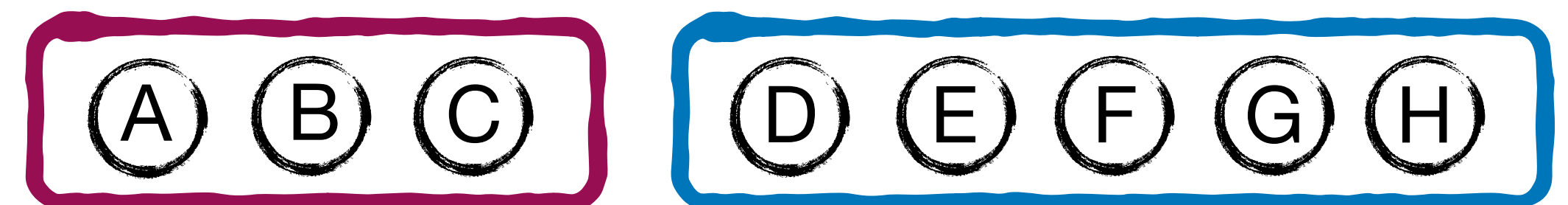
Insight: Picking some other close splitting
point alleviates the issue



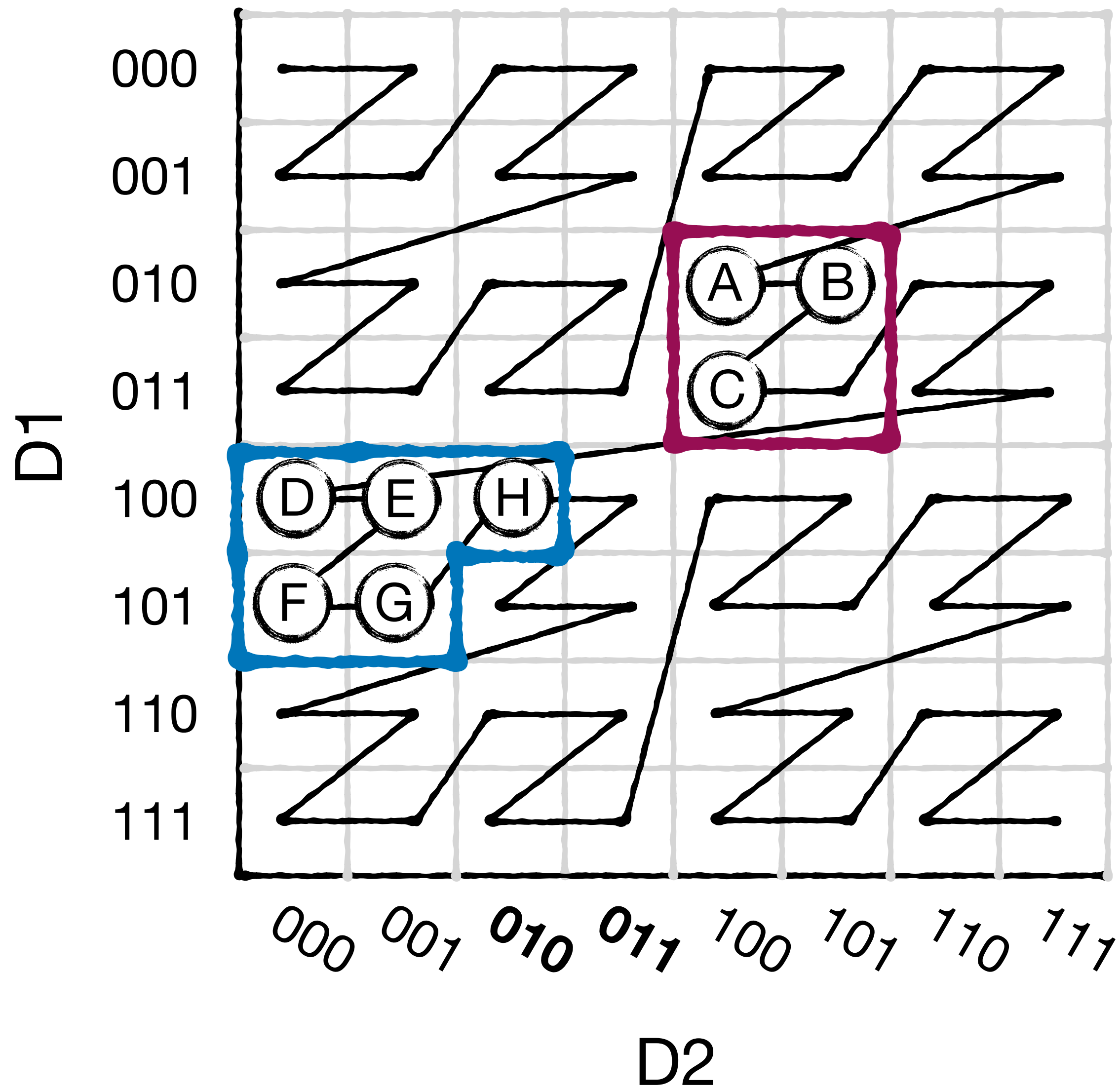
Insight: Picking some other close splitting point alleviates the issue



What's the trade-off?

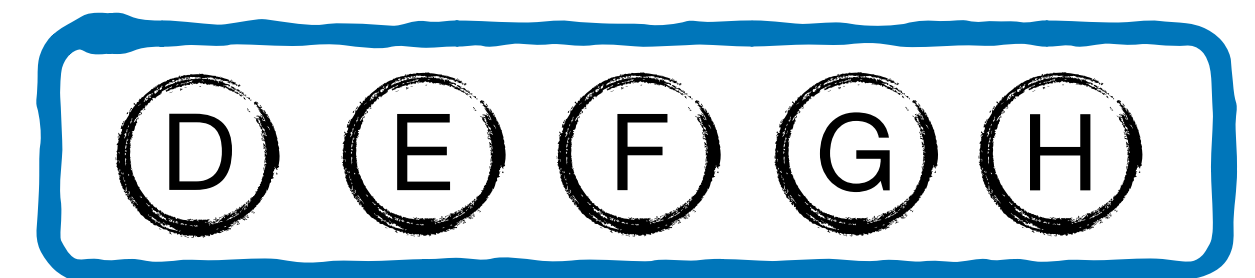
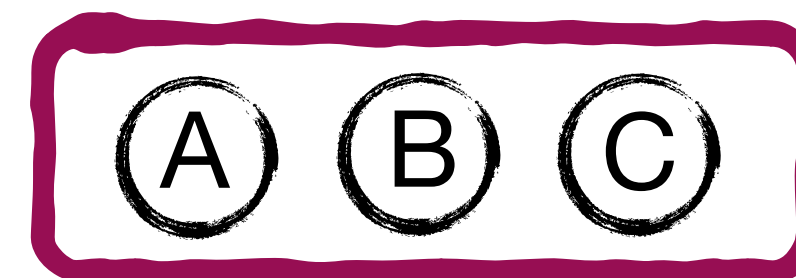


Insight: Picking some other close splitting
point alleviates the issue

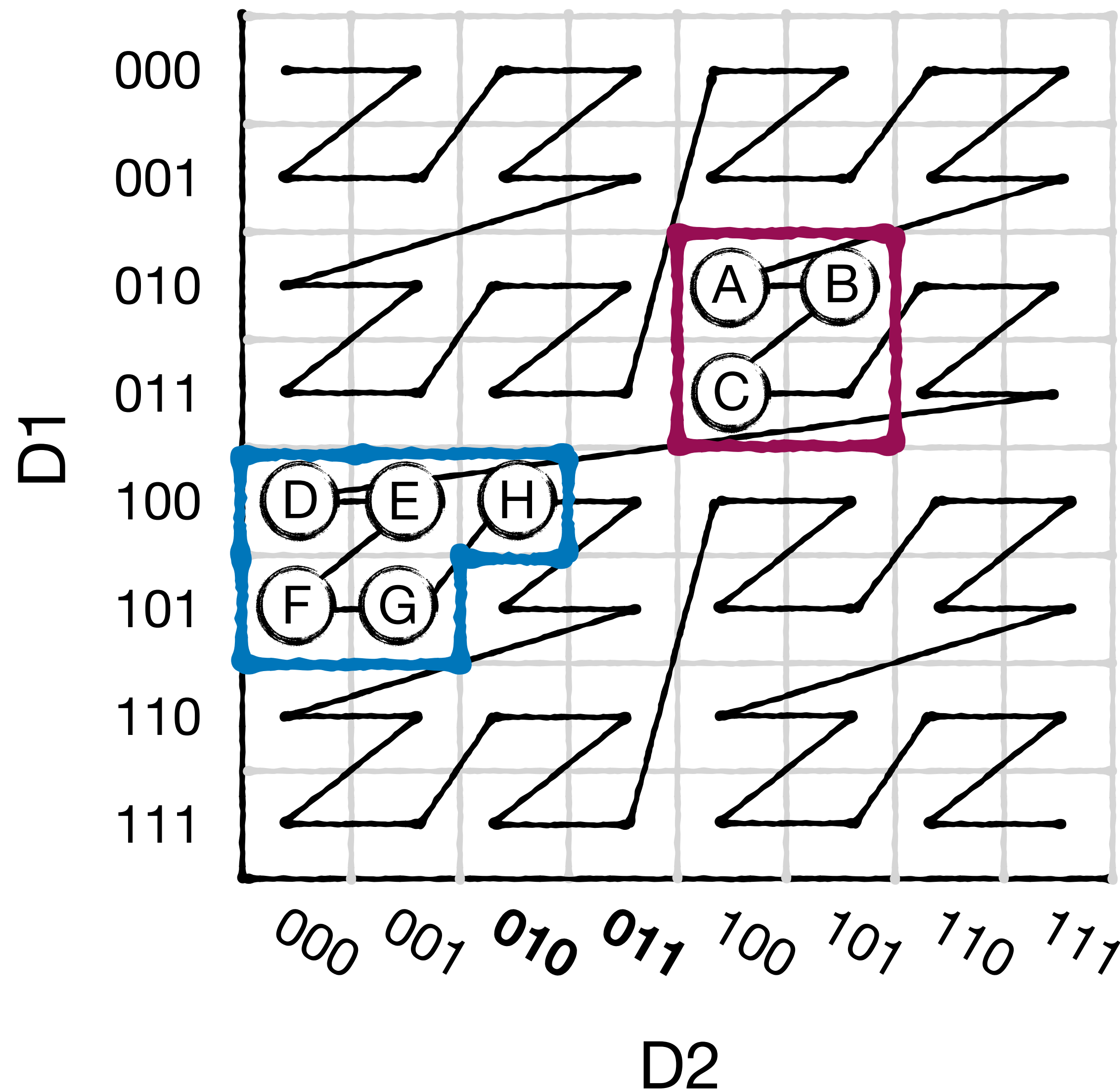


What's the trade-off?

Some nodes are less utilized

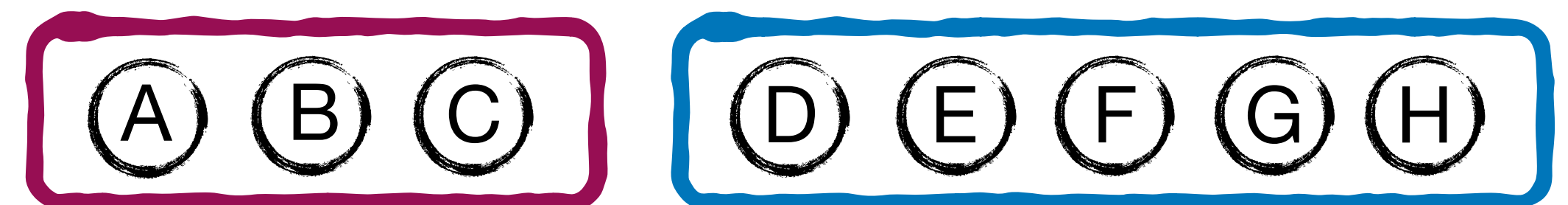


Insight: Picking some other close splitting
point alleviates the issue



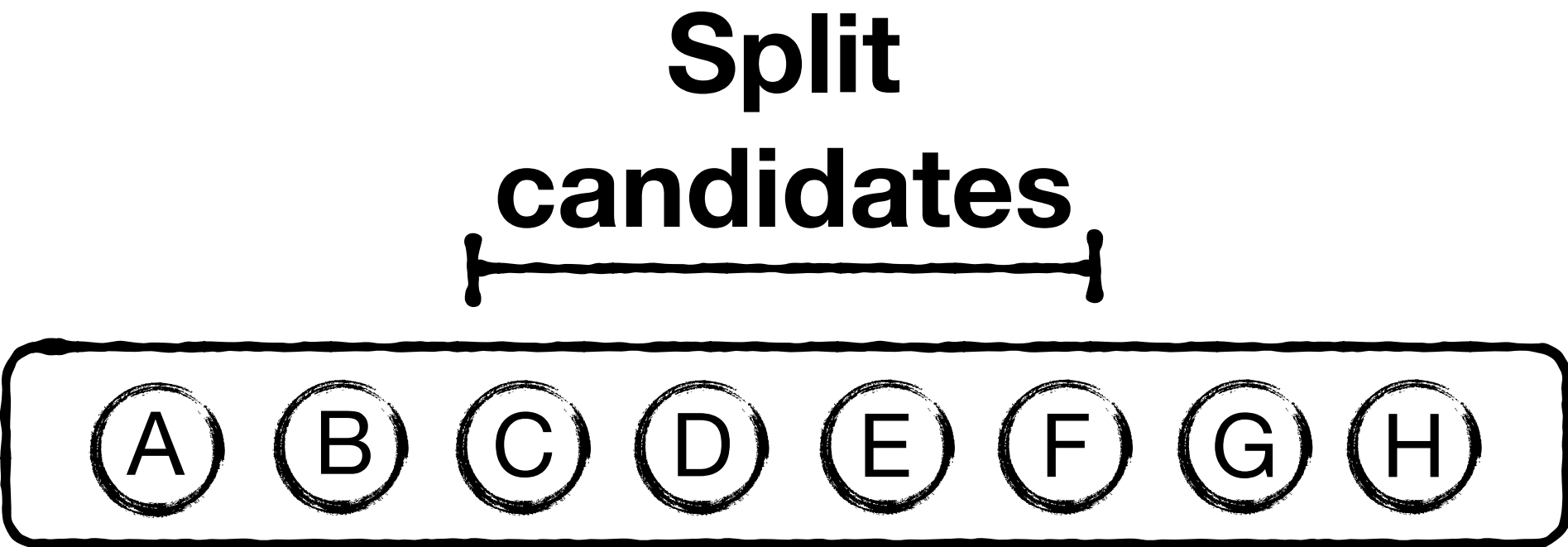
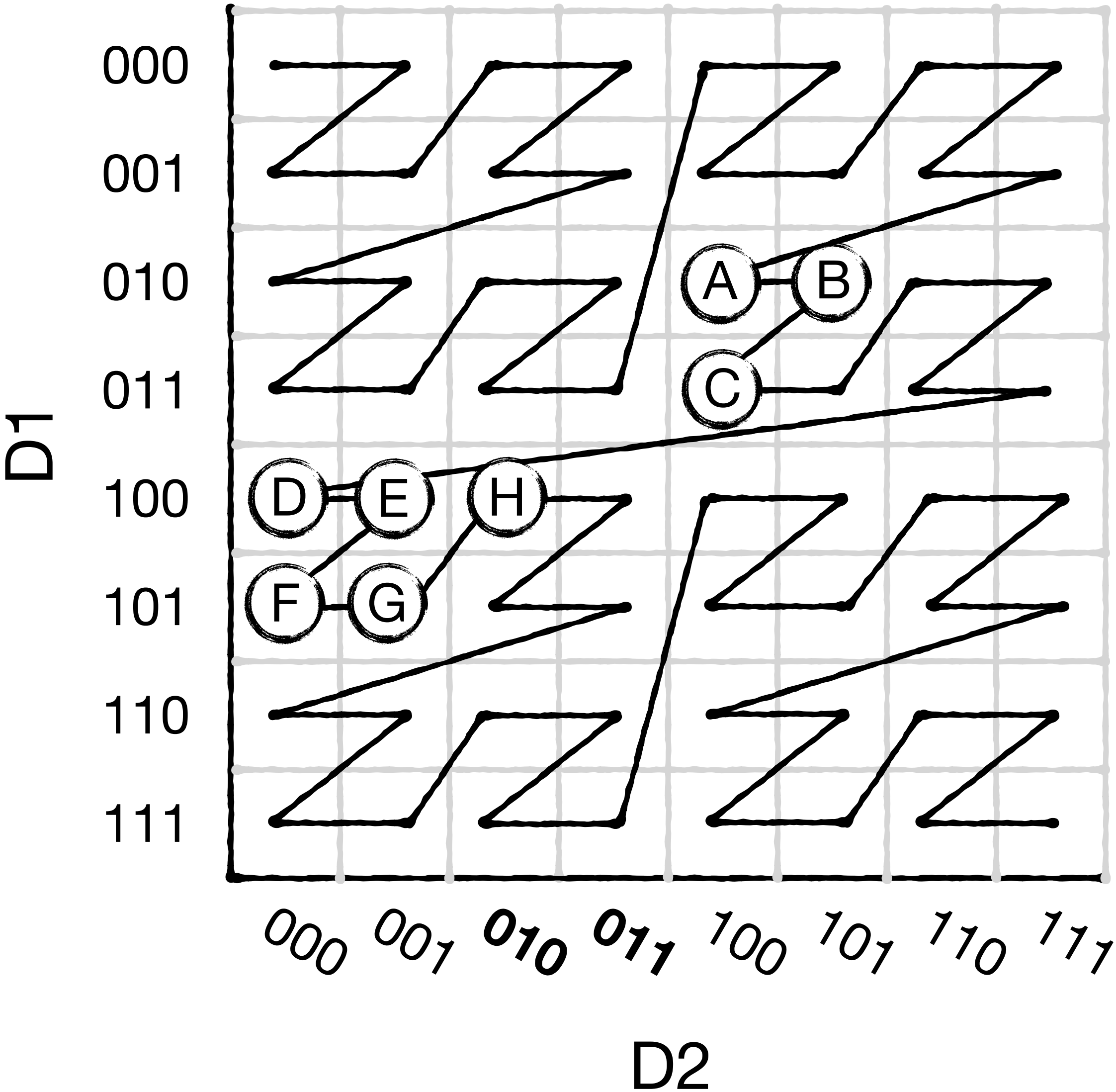
What's the trade-off?

Some nodes are less utilized

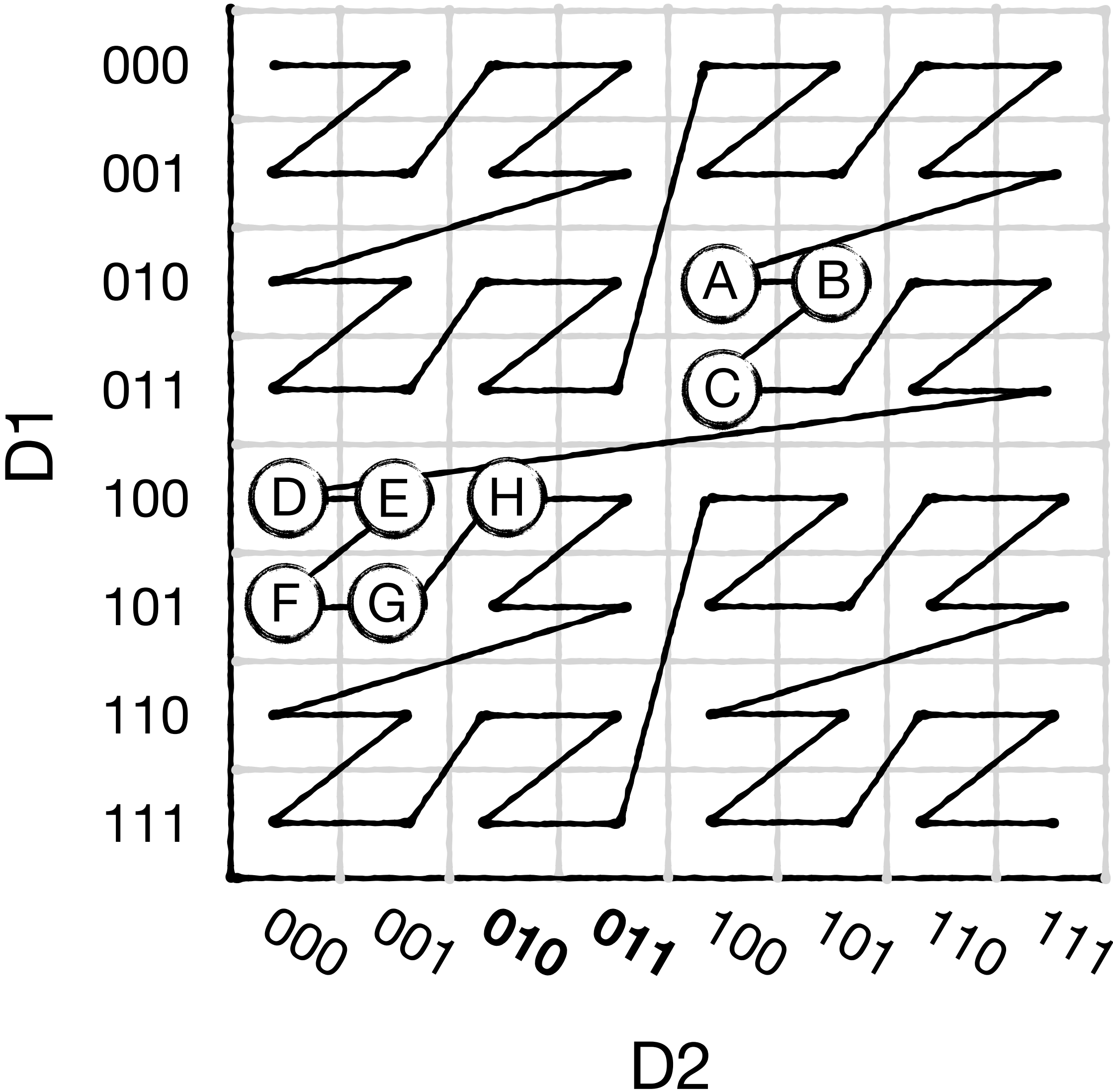


**Can lead to deeper tree with
more space wastage**

Balance by picking best split from a middle nodes



Balance by picking best split from a middle nodes



Split candidates

A B C D E F G H

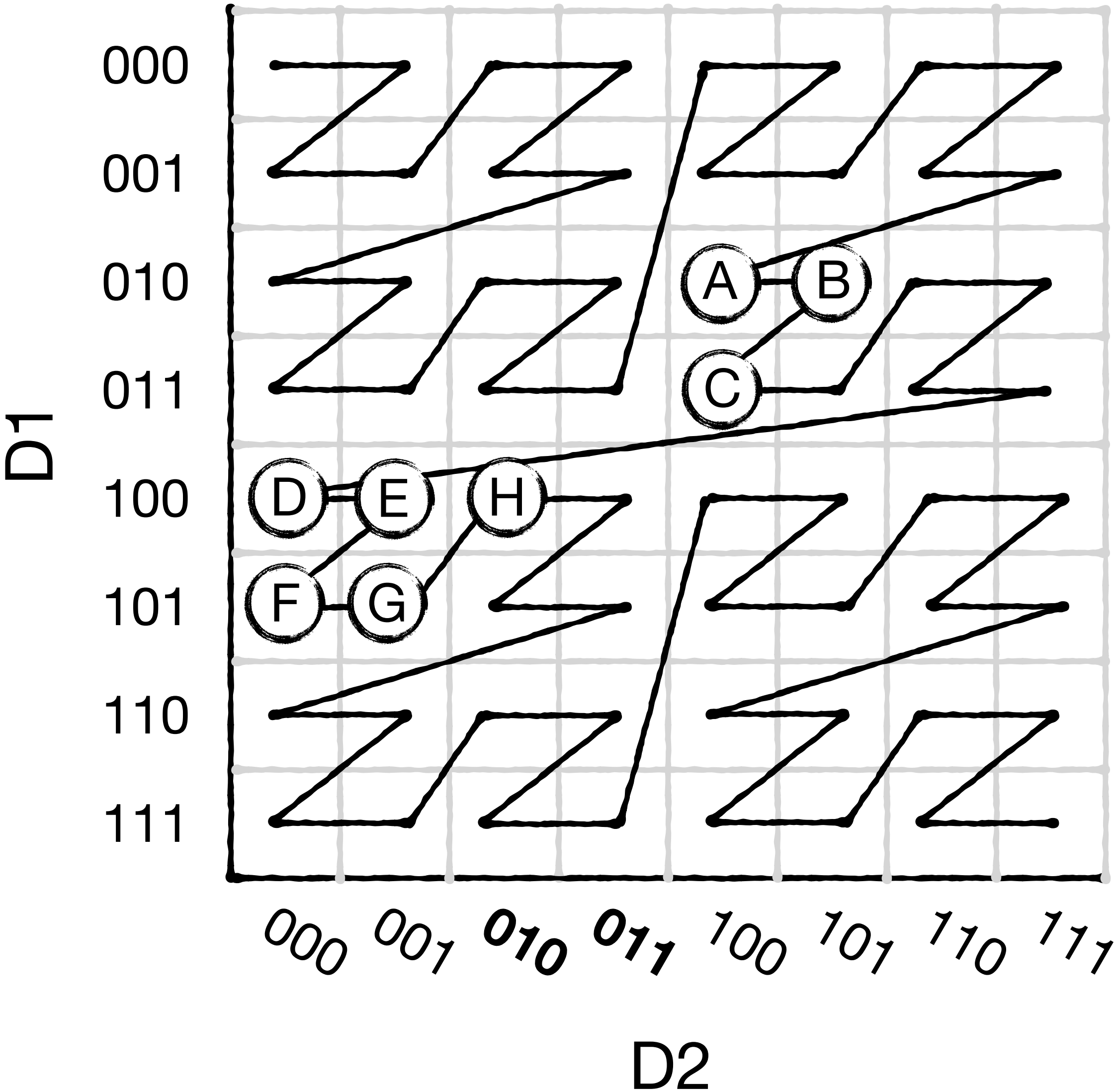
C01 00 10

D10 00 00

E10 00 01

F10 00 10

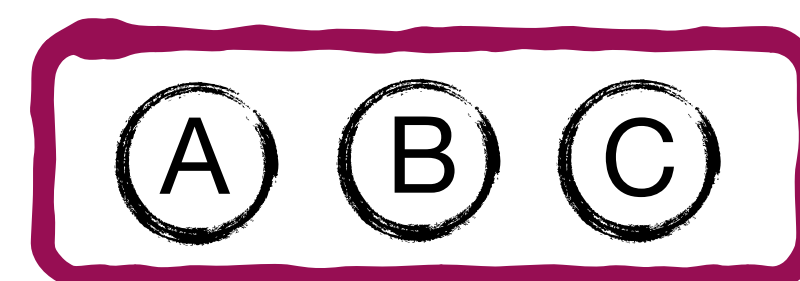
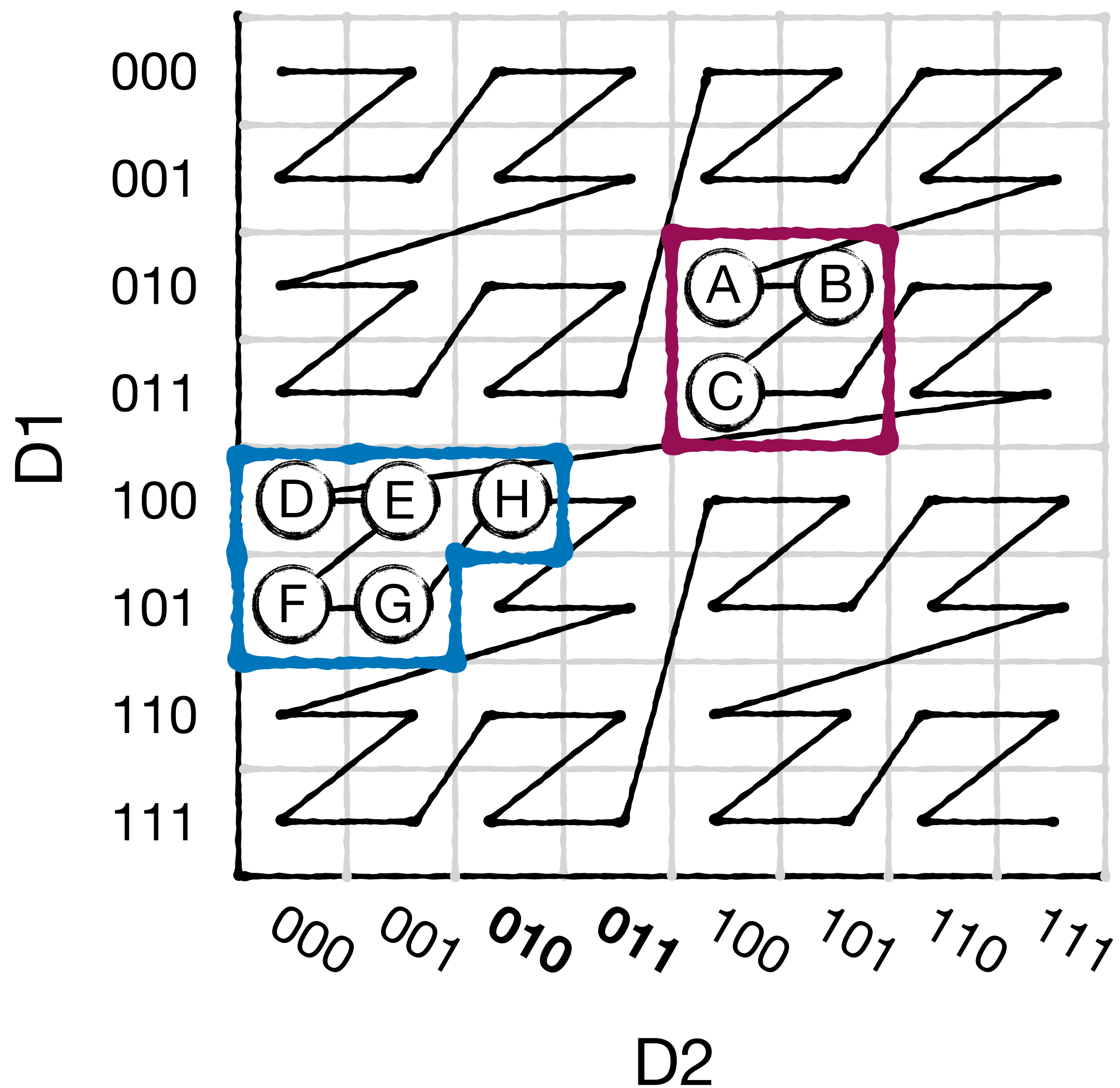
Balance by picking best split from a middle nodes



Split candidates

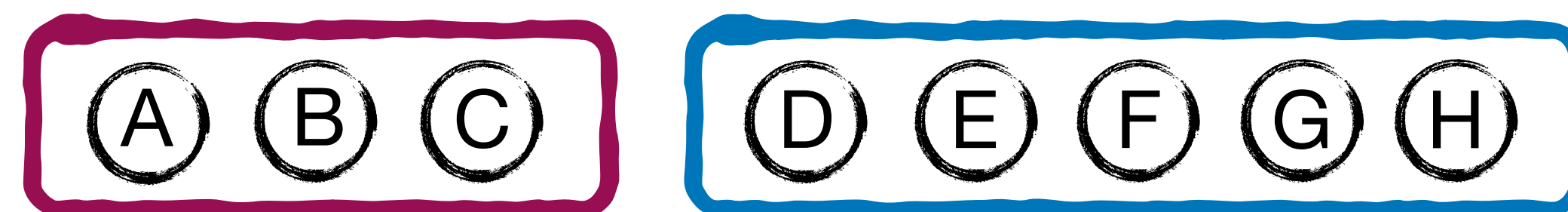
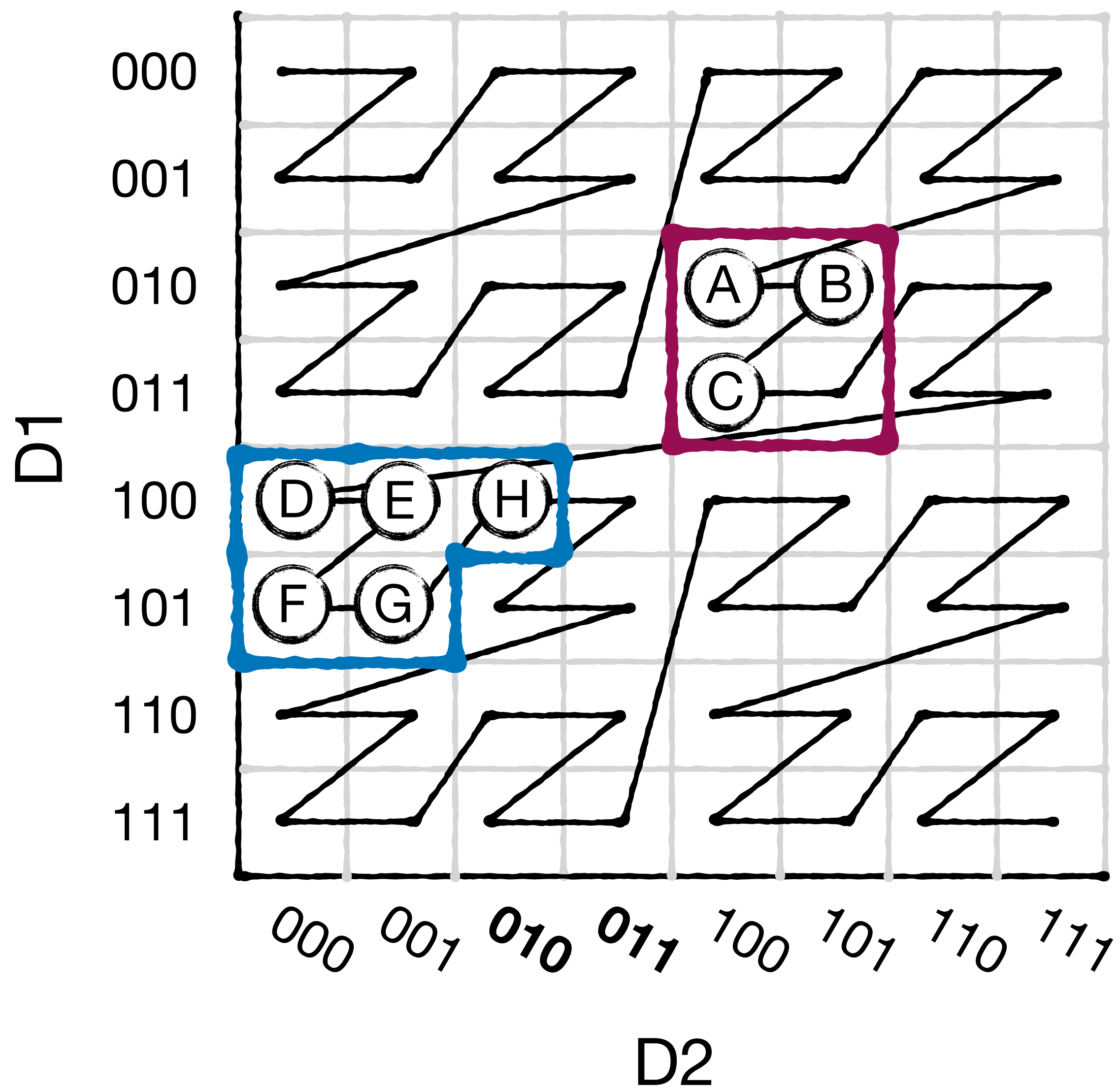
A B C D E F G H

(C) 01 00 10
(D) 10 00 00 ← **Most trailing zeros, best splitter**
(E) 10 00 01
(F) 10 00 10



- (C) 01 00 10
- (D) 10 00 00
- (E) 10 00 01
- (F) 10 00 10

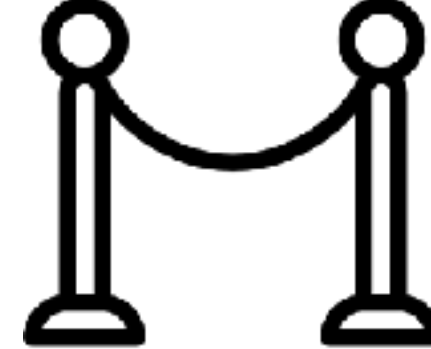
← Most trailing zeros, best splitter



**Balances goals of having
rectangular nodes against
having shallow and space-
efficient tree**



UB-Tree



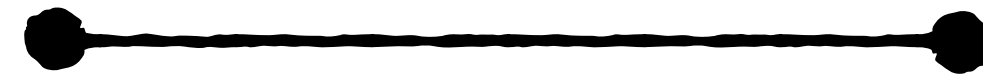
**FIFO/LRU
Review**



LRU/K



2Q

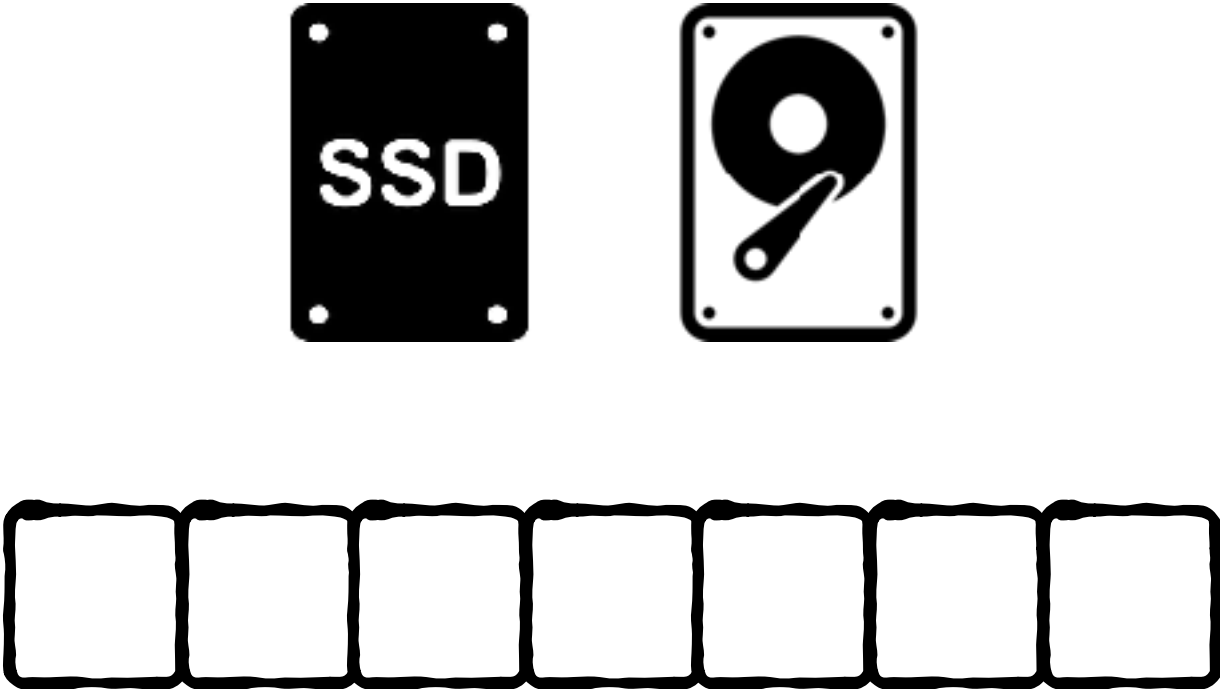


2 weeks ago



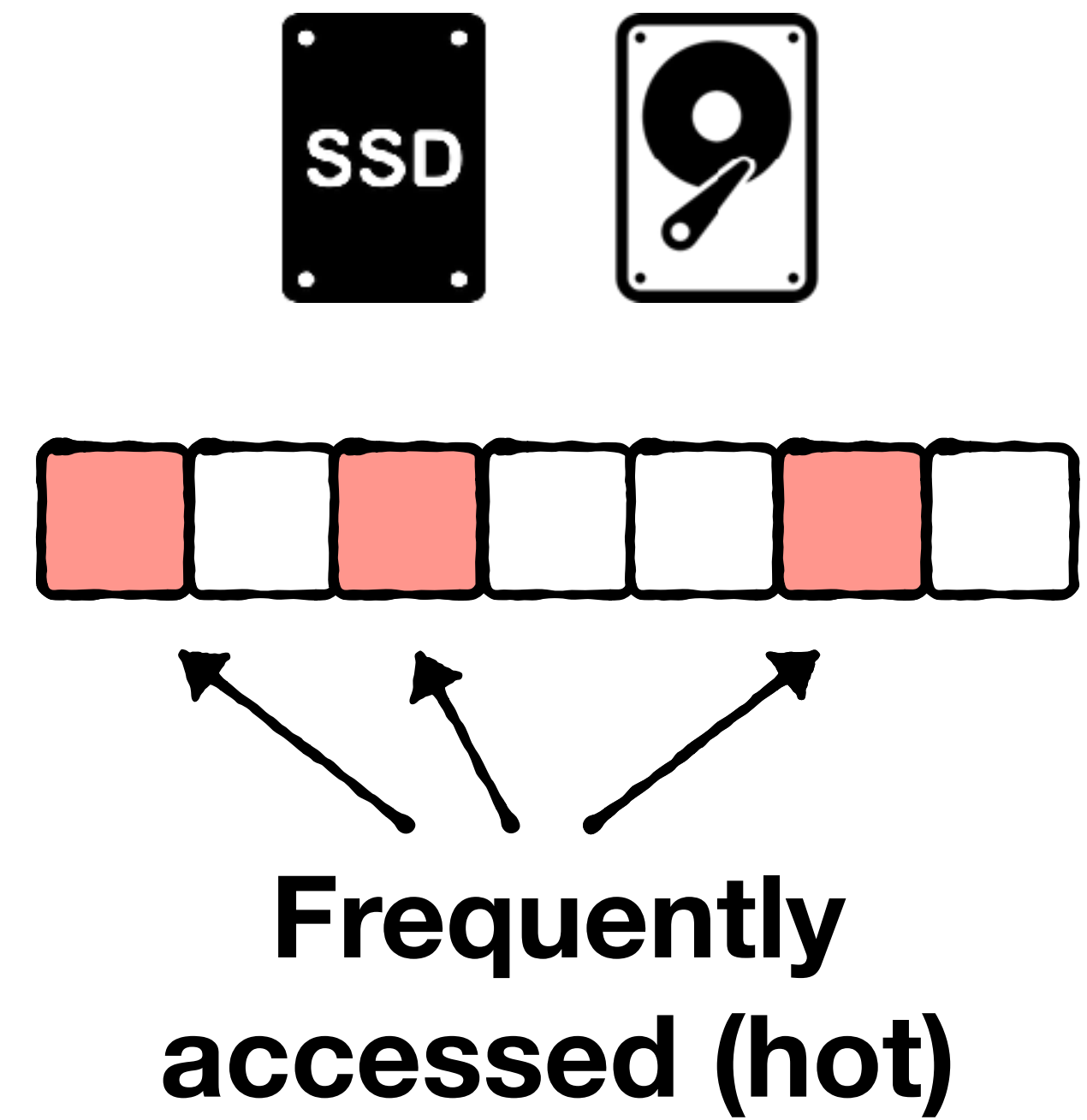
This week

Review

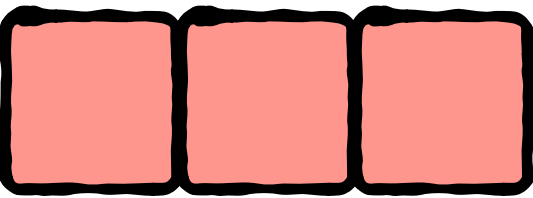


4KB pages

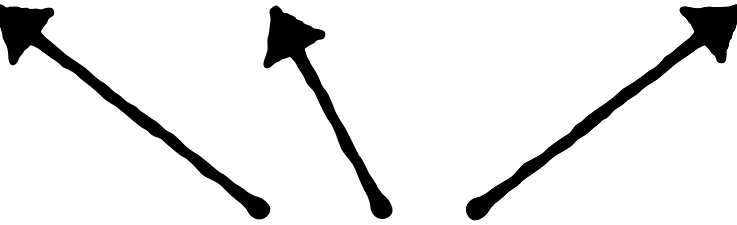
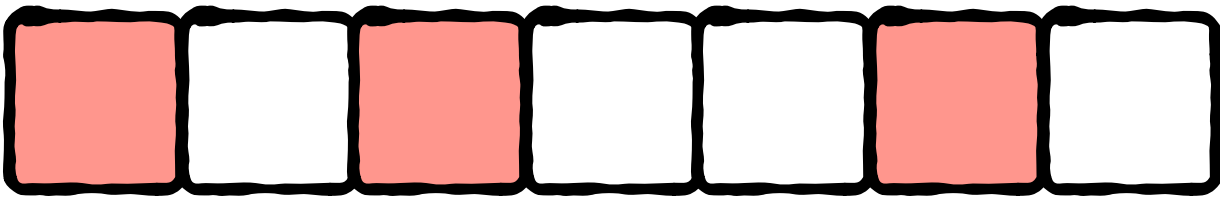
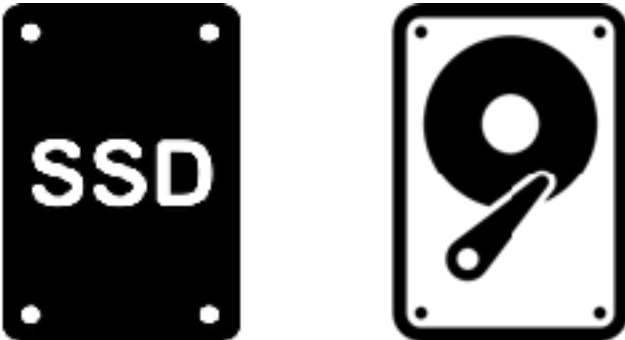
Review



Review

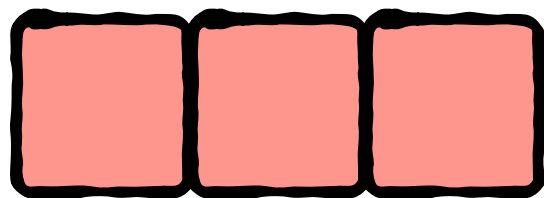


**Buffer pool /
block cache**



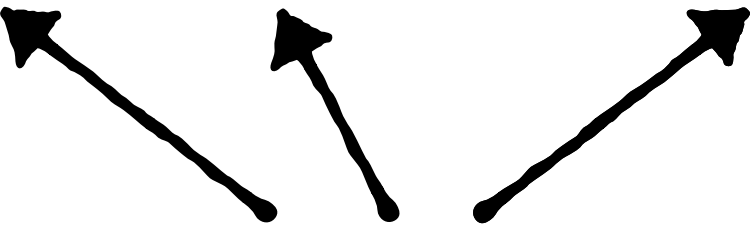
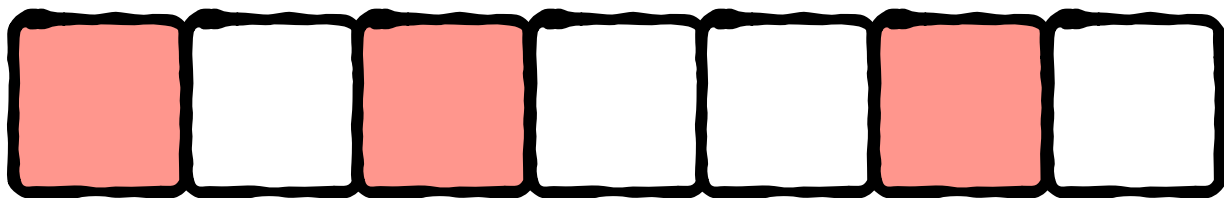
Frequently
accessed (hot)

Review



Buffer pool /
block cache

Implemented as a hash table



Frequently
accessed (hot)

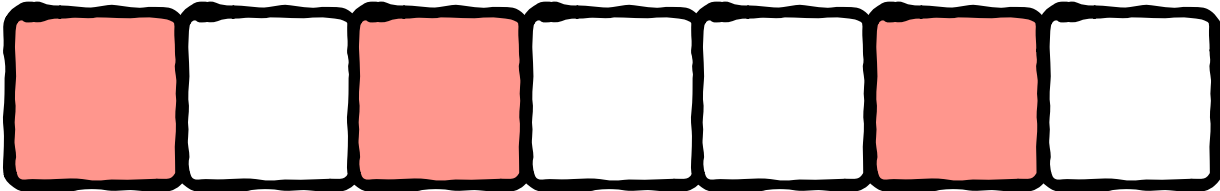
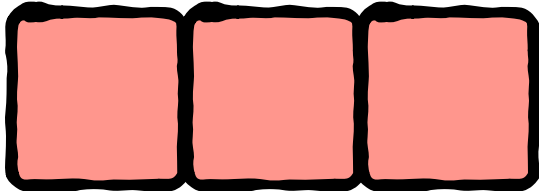
Review

Read new page



Review

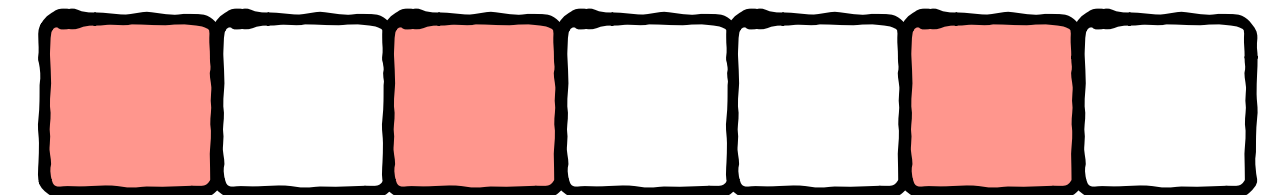
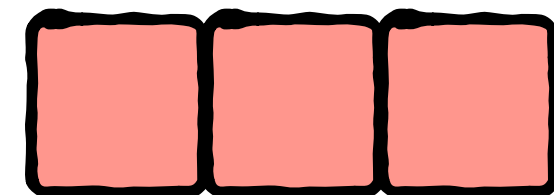
Read new page



Evict existing page

Review

Read new page



Evict existing page

But which?

Review

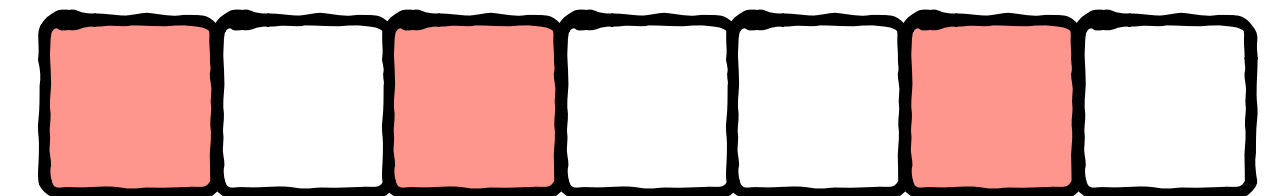
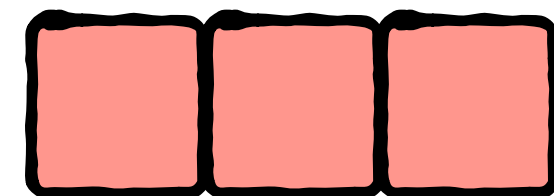
Read new page



Evict page least likely to be read again

Review

Read new page

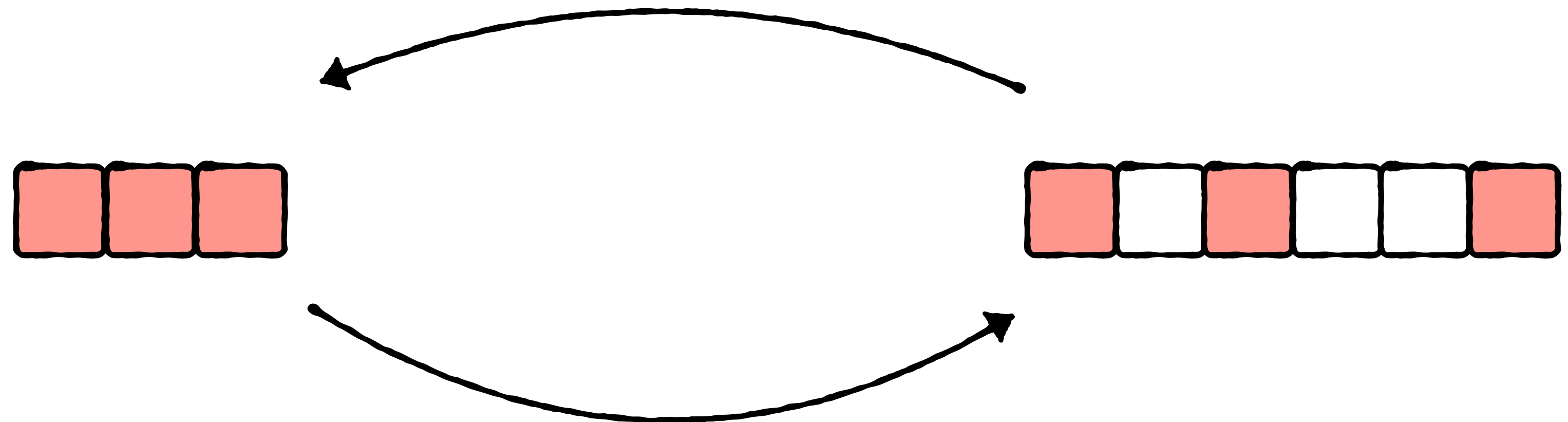


Evict page least likely to be read again

Must approximate

FIFO - First In First Out

Evict page read longest ago



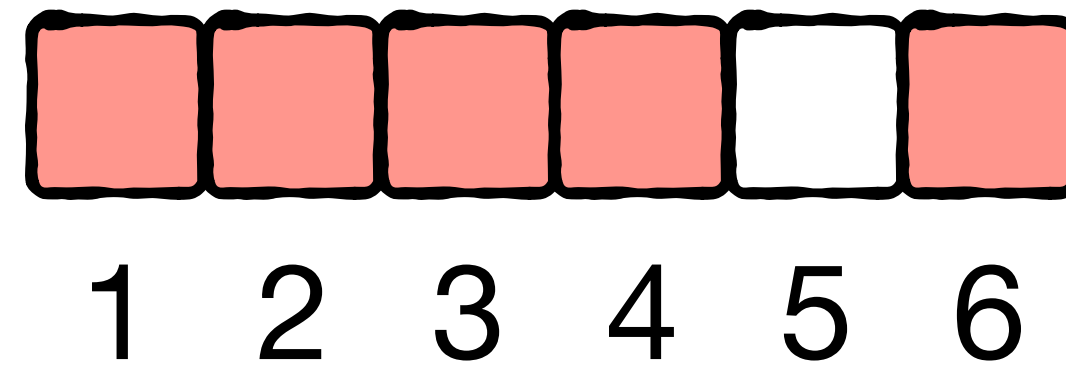
FIFO - First In First Out

Evict page read longest ago

Queue
(Array)



Buffer pool
(Hash table)



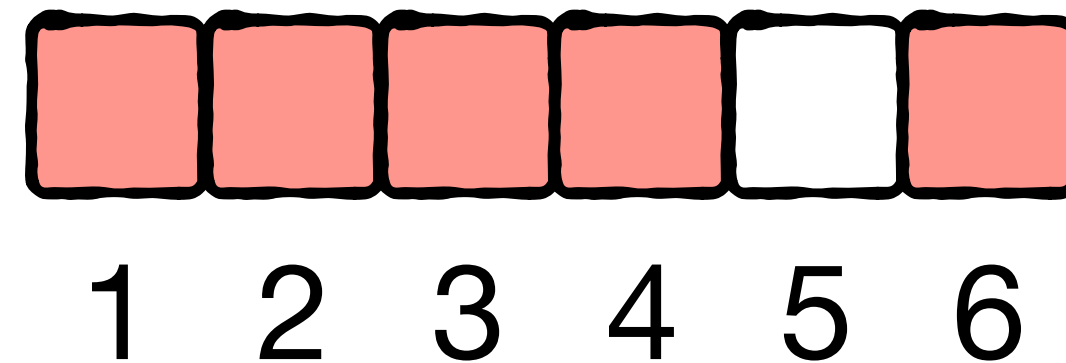
FIFO - First In First Out

Evict page read longest ago

Queue
(Array)



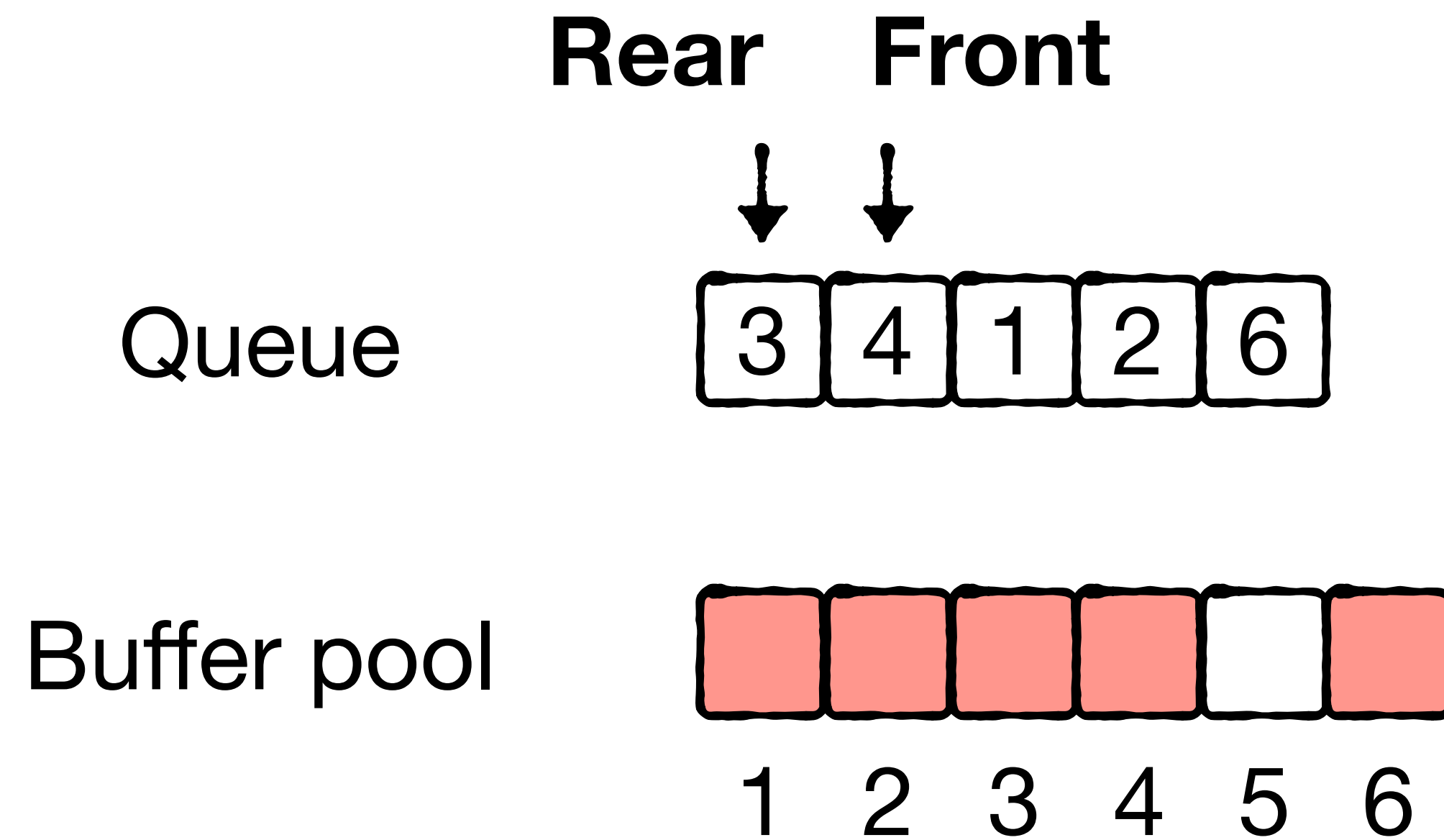
Buffer pool
(Hash table)



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

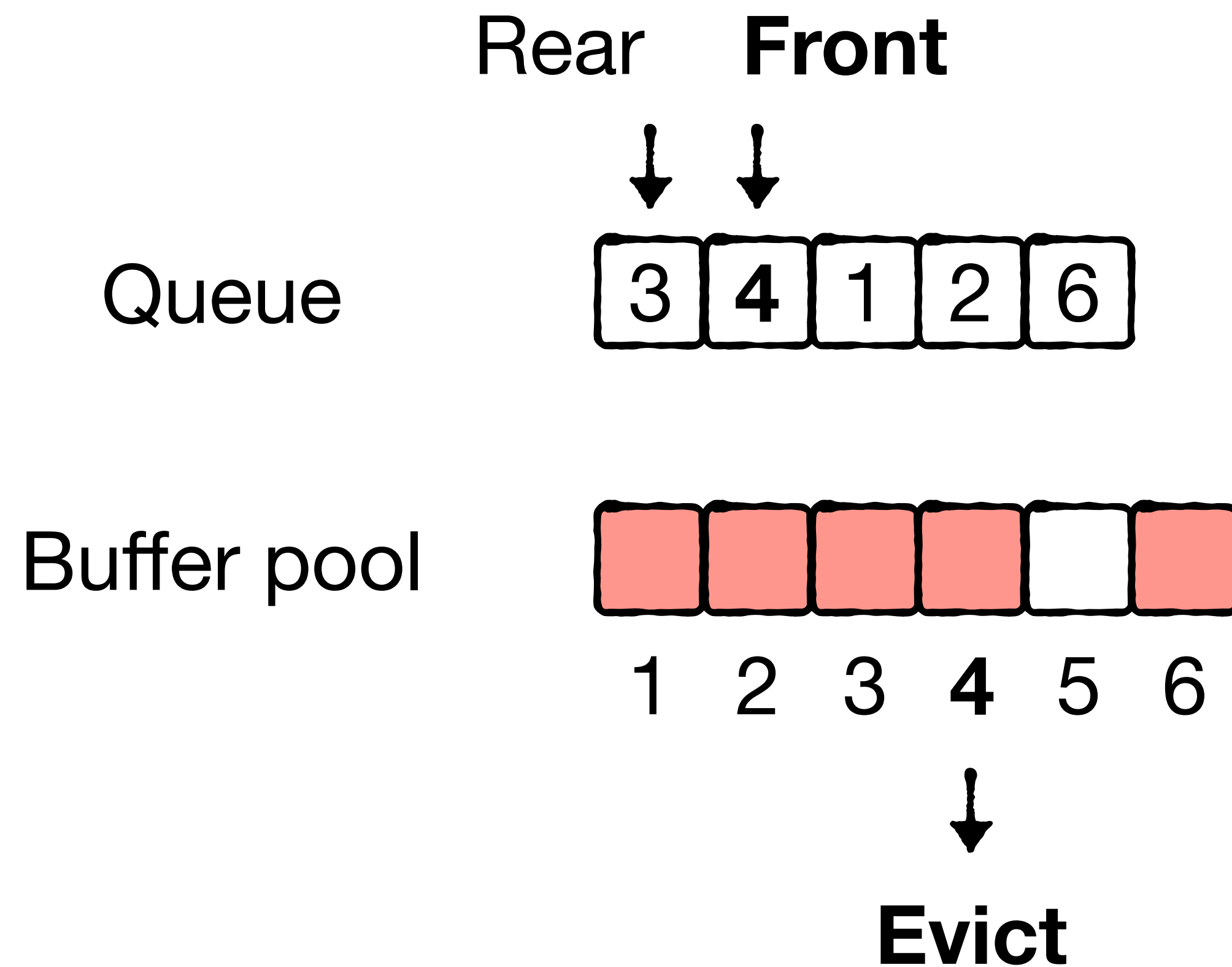
FIFO - First In First Out

Evict page read longest ago



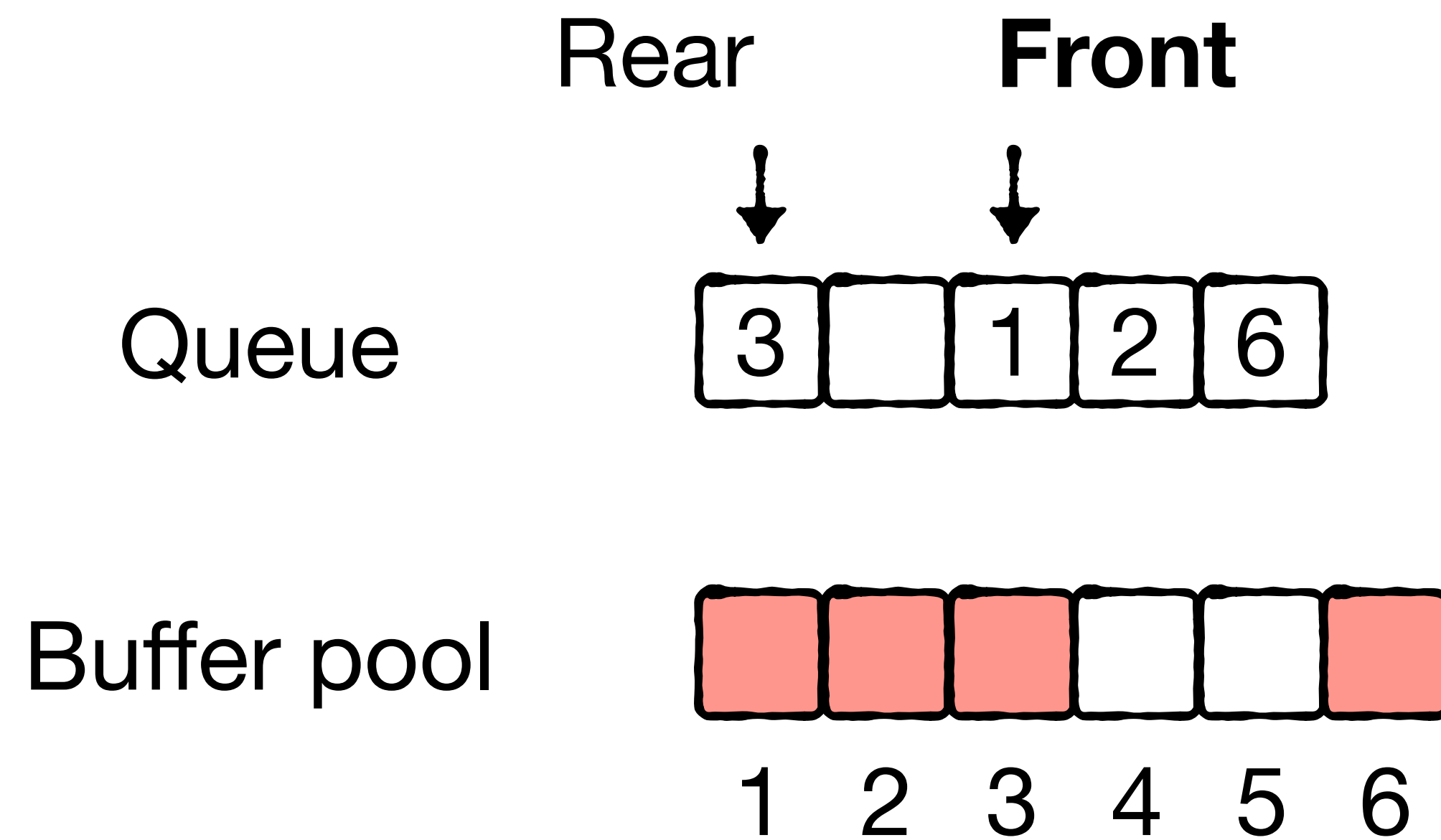
FIFO - First In First Out

Evict page read longest ago



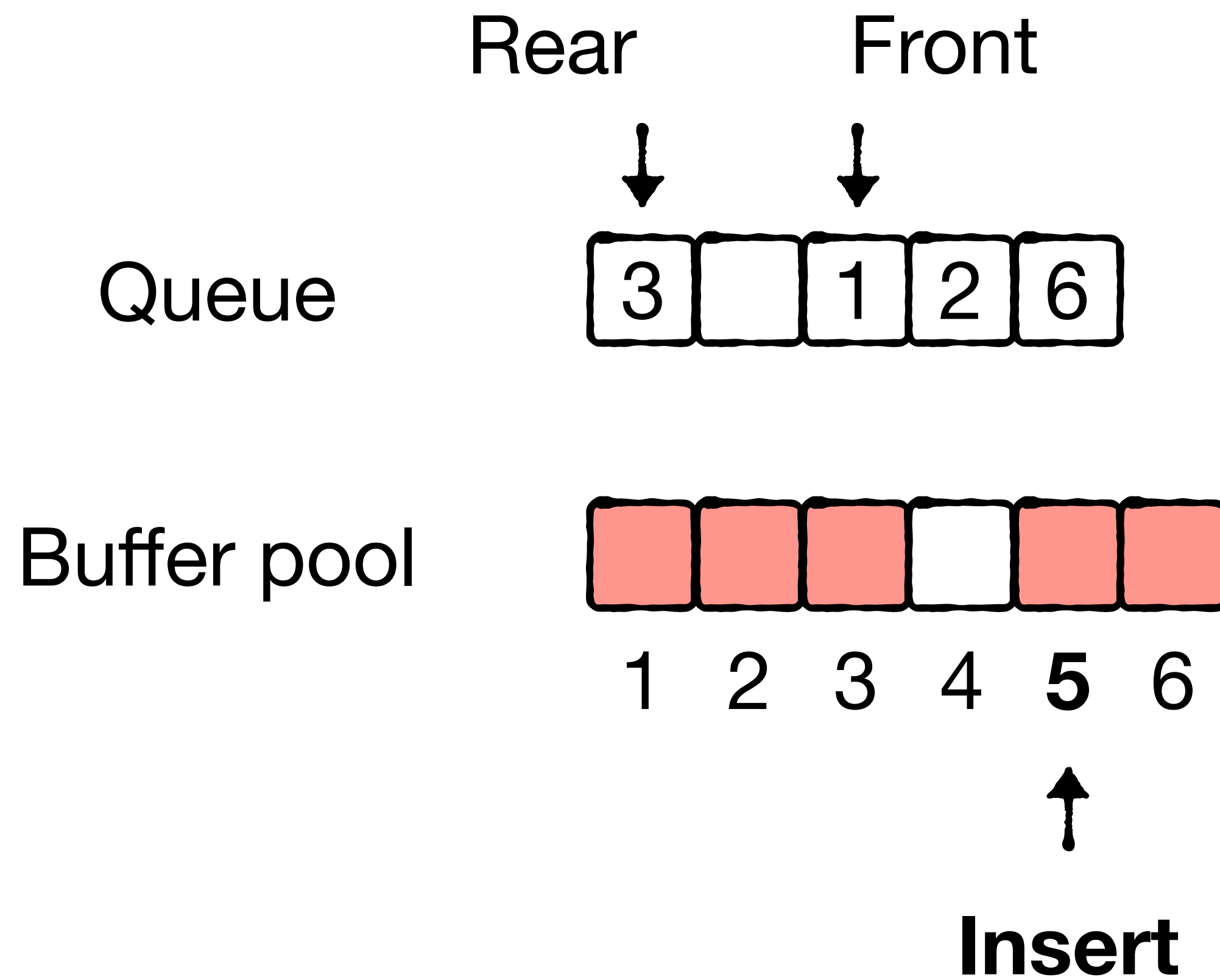
FIFO - First In First Out

Evict page read longest ago



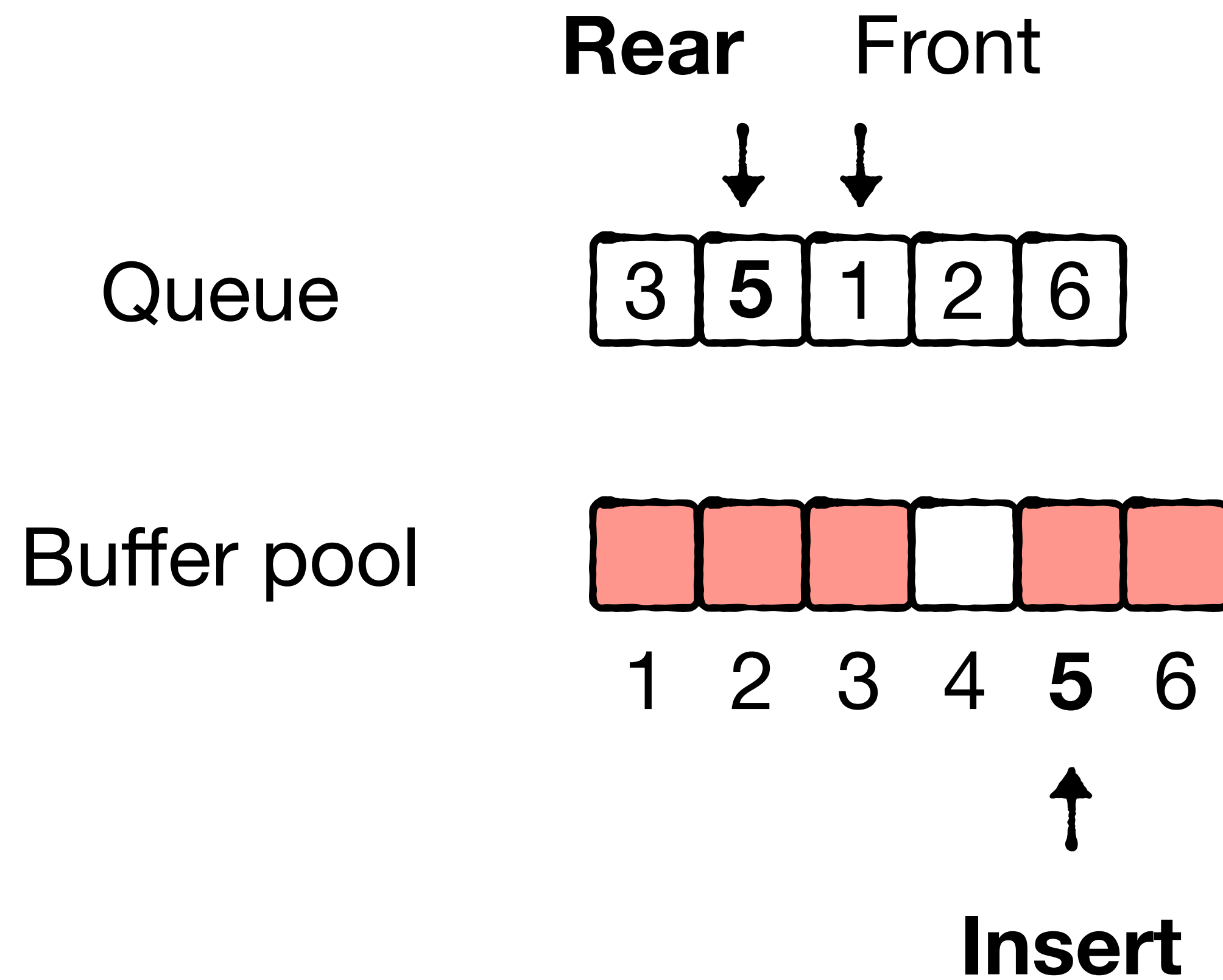
FIFO - First In First Out

Evict page read longest ago



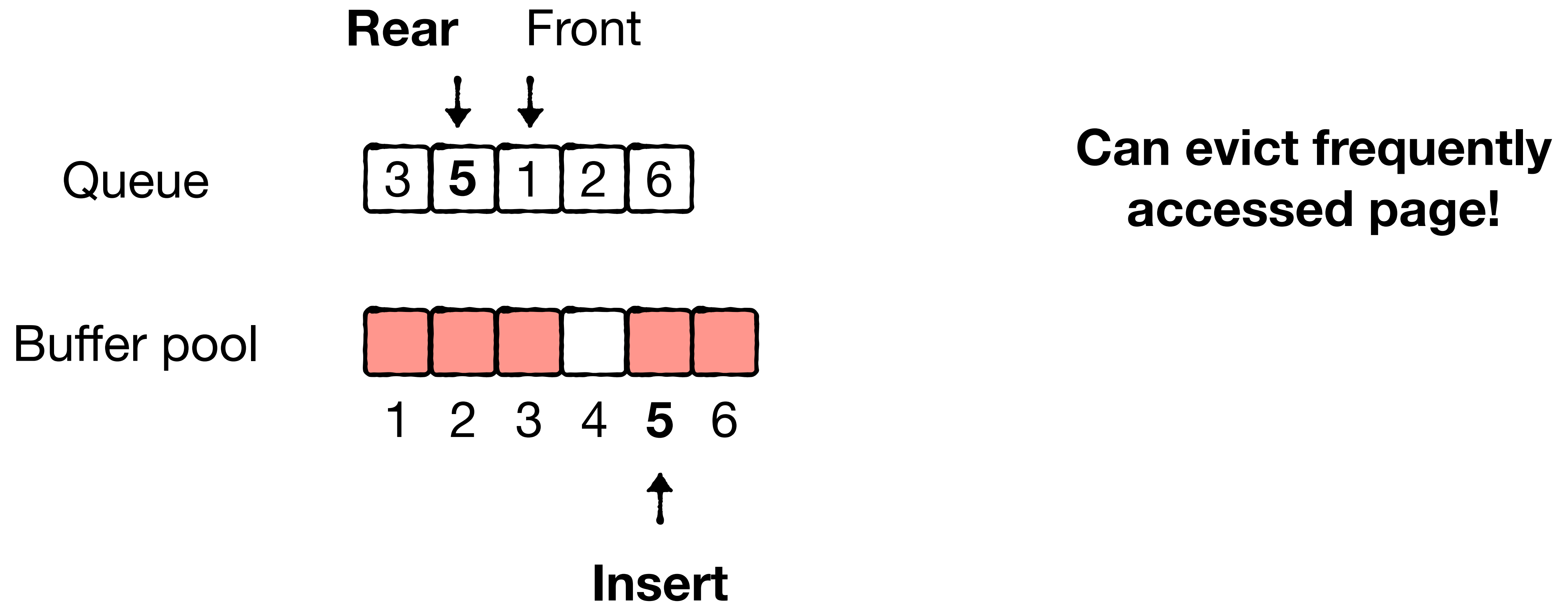
FIFO - First In First Out

Evict page read longest ago



FIFO - First In First Out

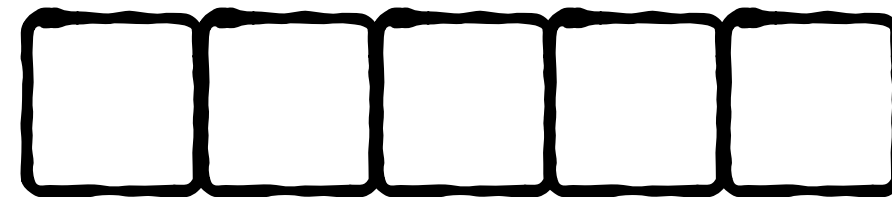
Evict page read longest ago



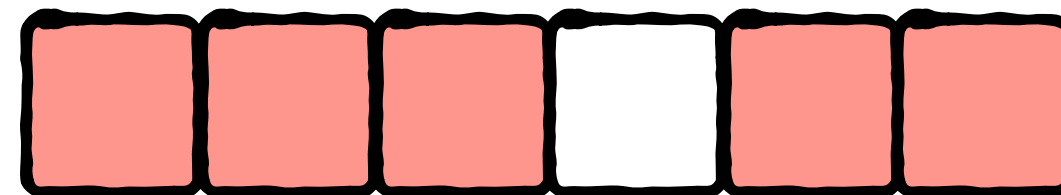
LRU - Least Recently Used

Evict page accessed longest ago

Queue



Buffer pool

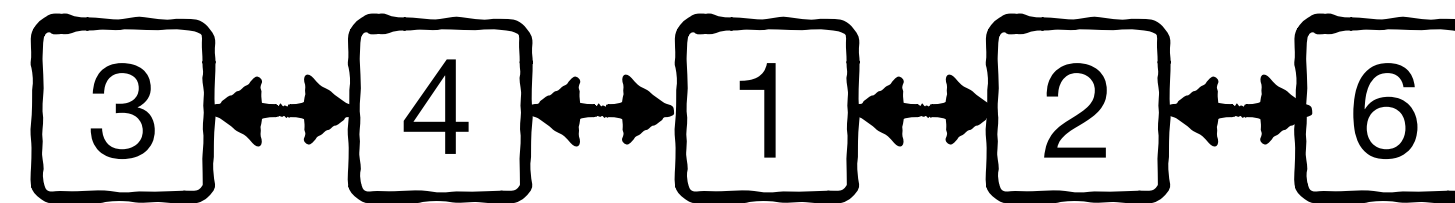


1 2 3 4 5 6

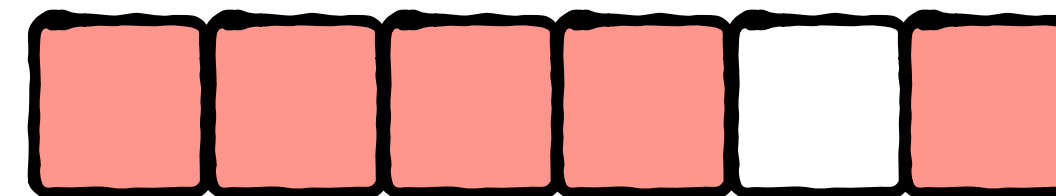
LRU - Least Recently Used

Evict page accessed longest ago

Queue
Doubly Linked list



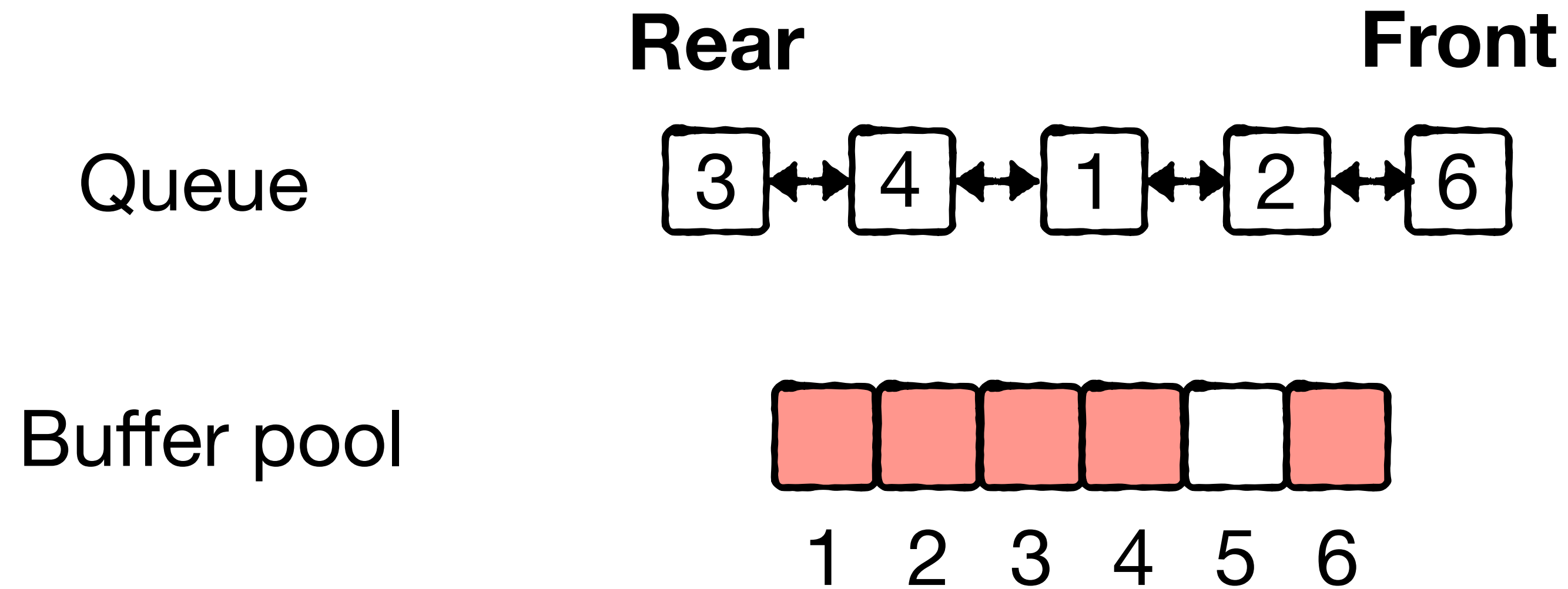
Buffer pool



1 2 3 4 5 6

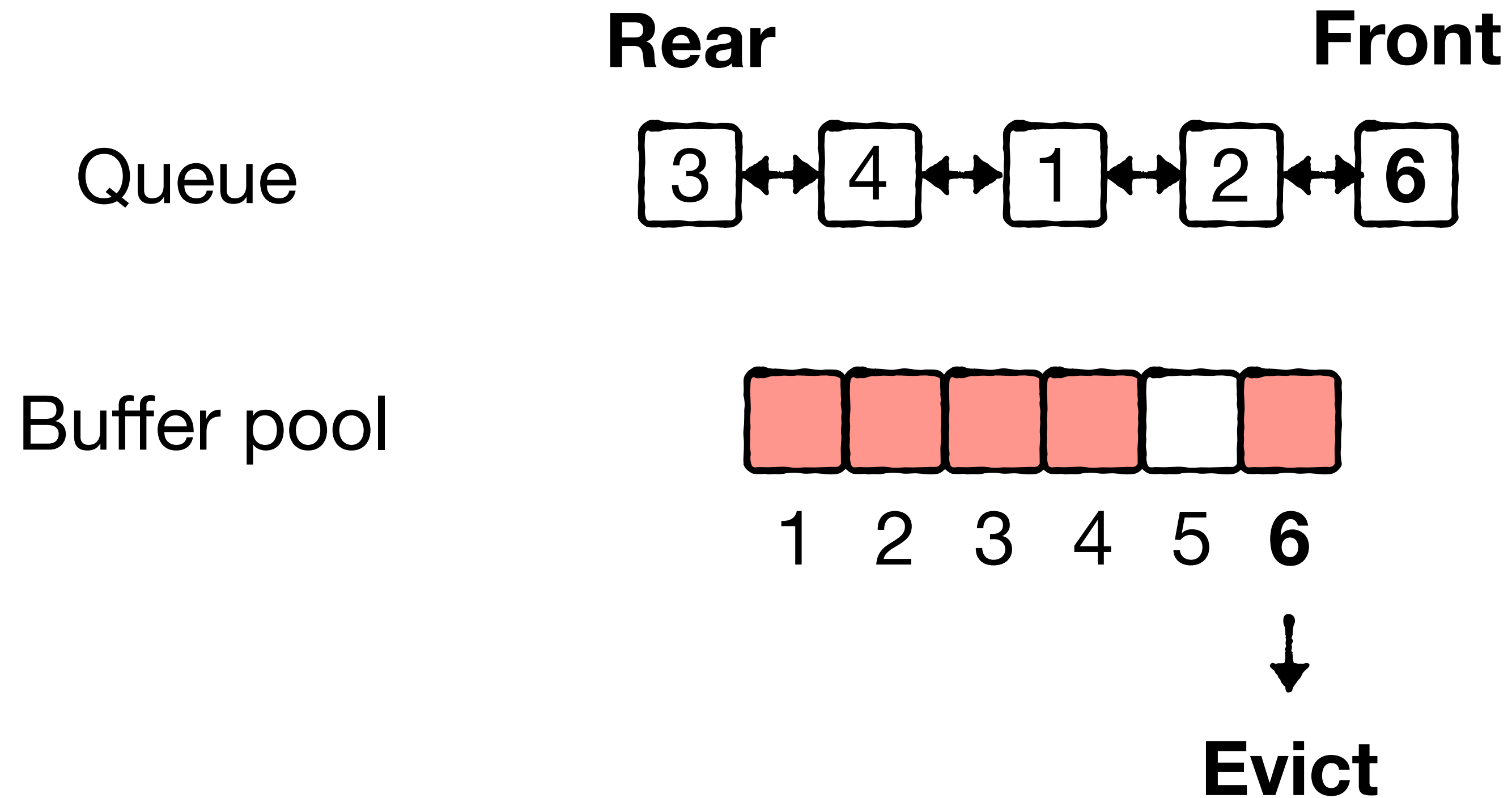
LRU - Least Recently Used

Evict page accessed longest ago



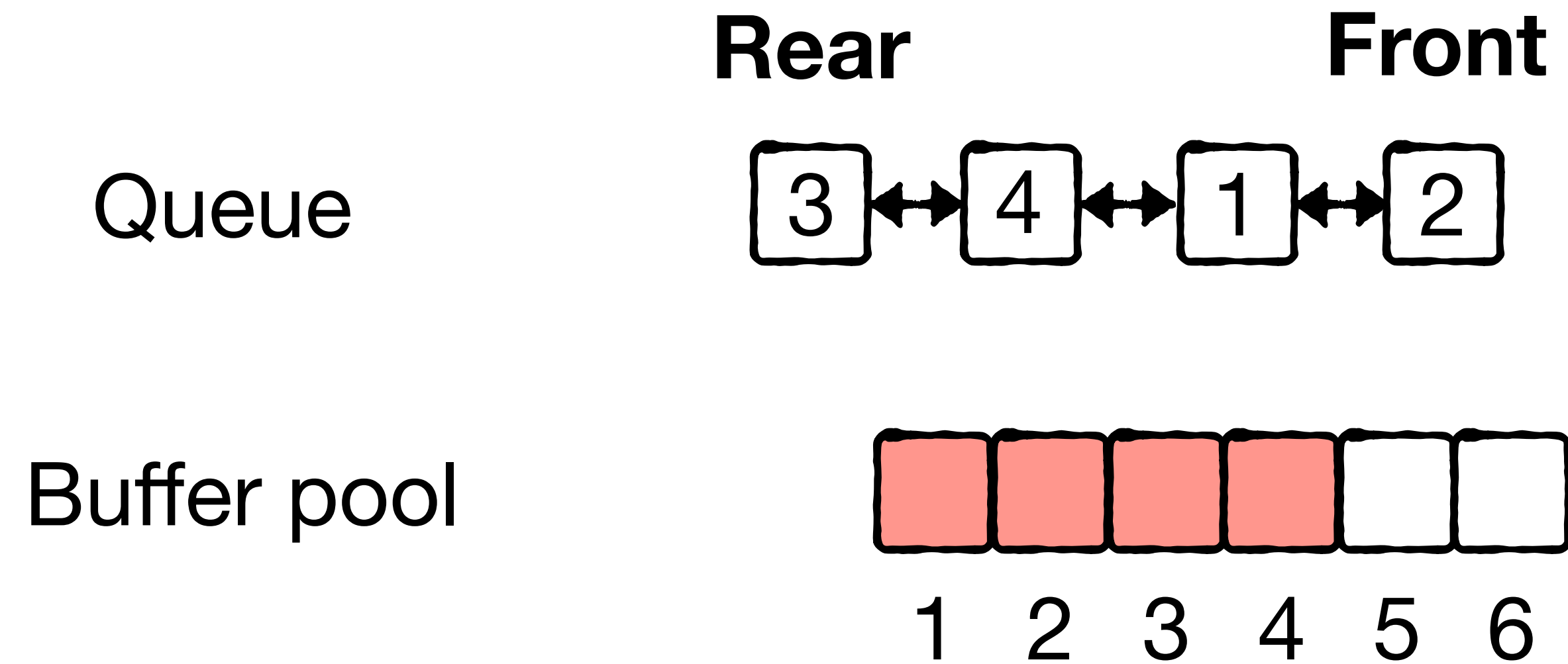
LRU - Least Recently Used

Evict page accessed longest ago



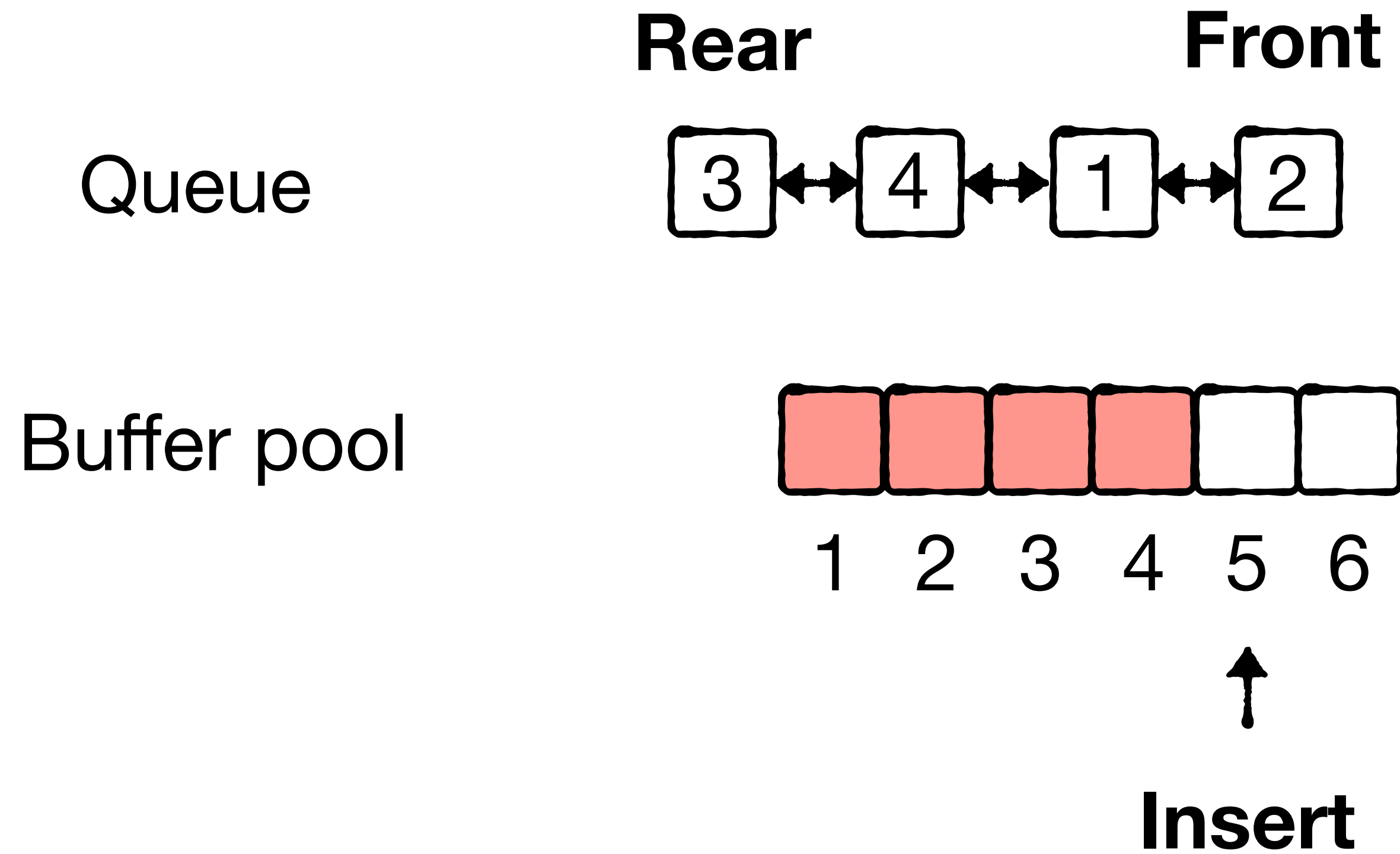
LRU - Least Recently Used

Evict page accessed longest ago



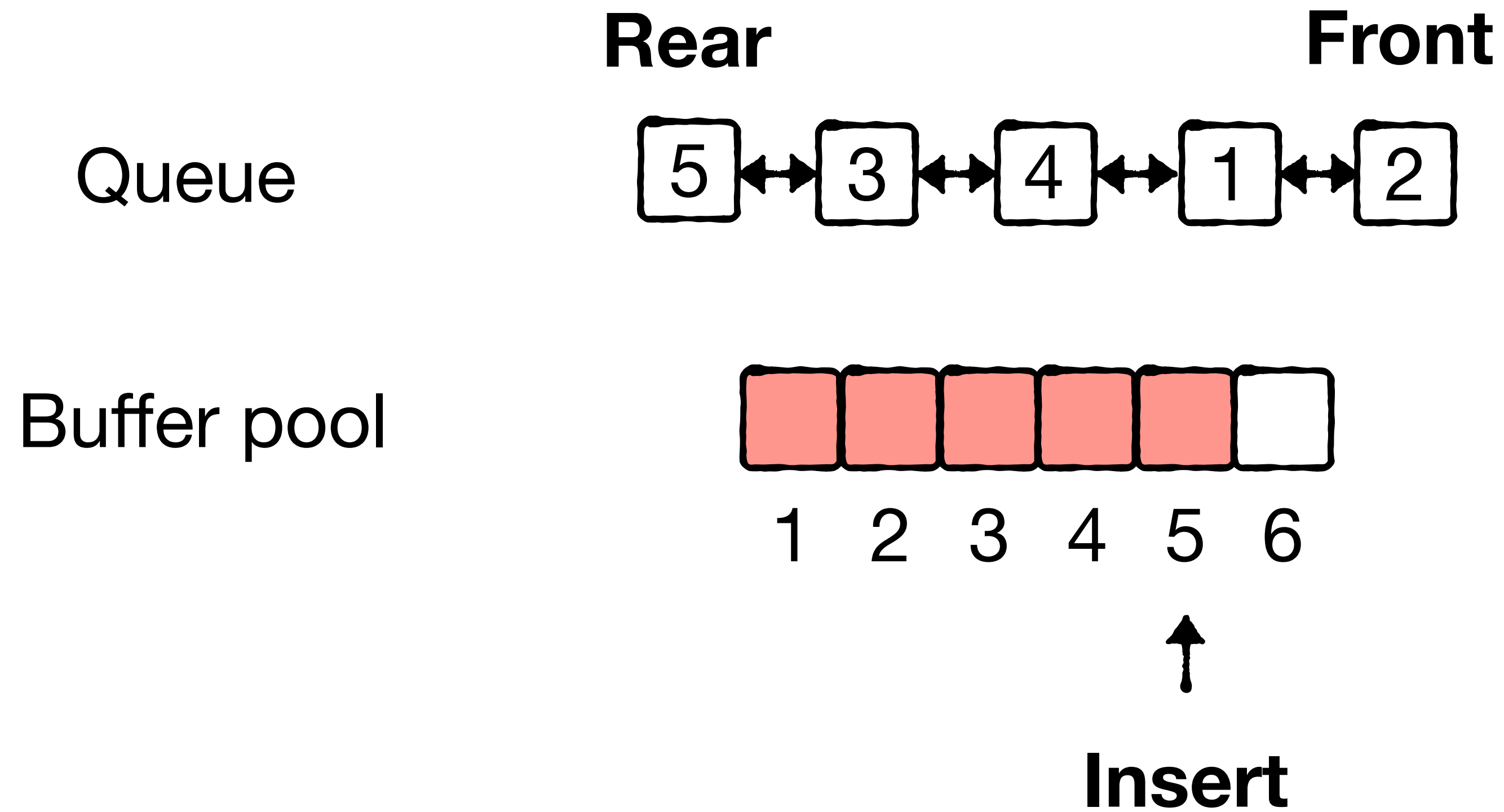
LRU - Least Recently Used

Evict page accessed longest ago



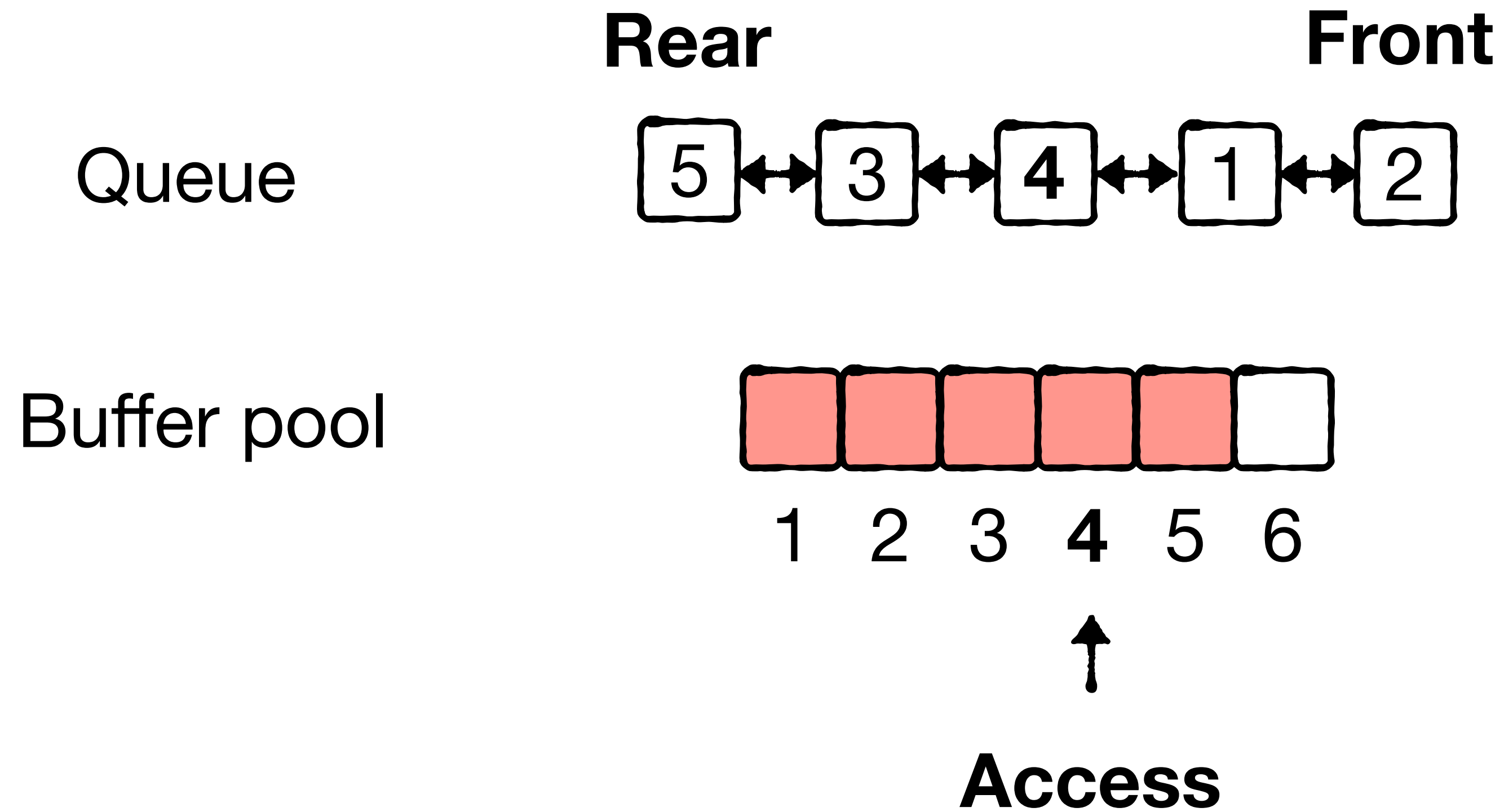
LRU - Least Recently Used

Evict page accessed longest ago



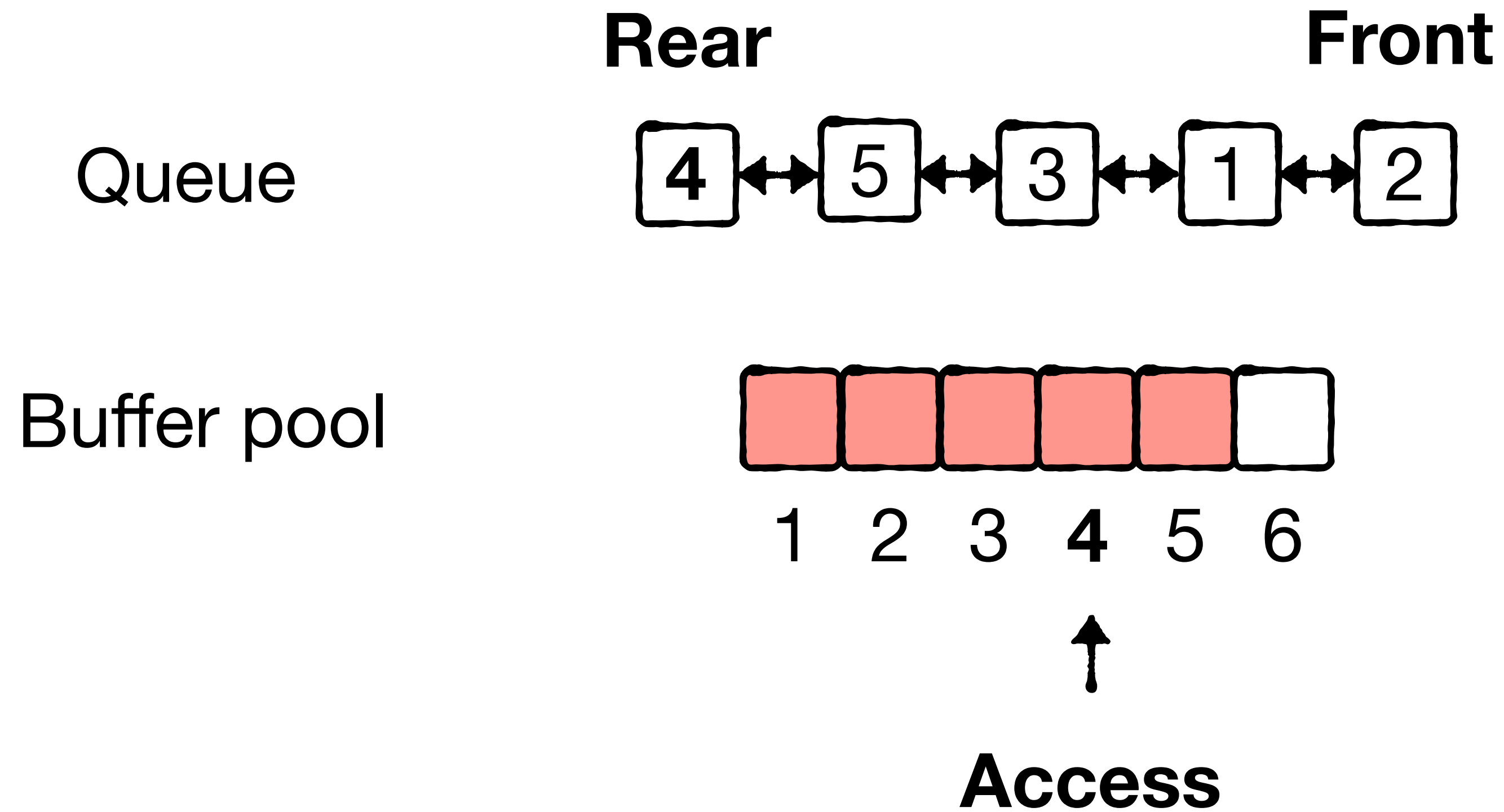
LRU - Least Recently Used

Evict page accessed longest ago



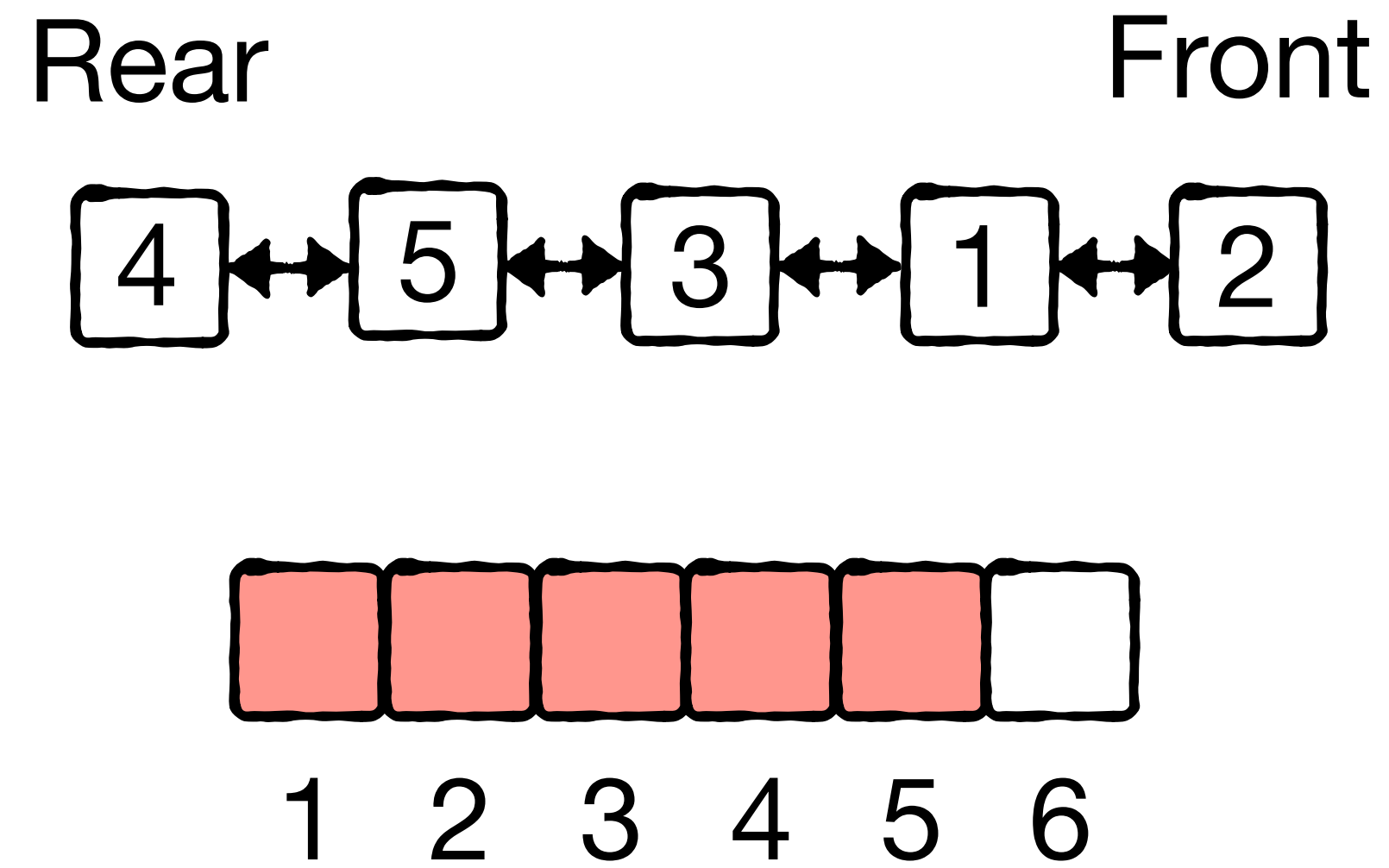
LRU - Least Recently Used

Evict page accessed longest ago

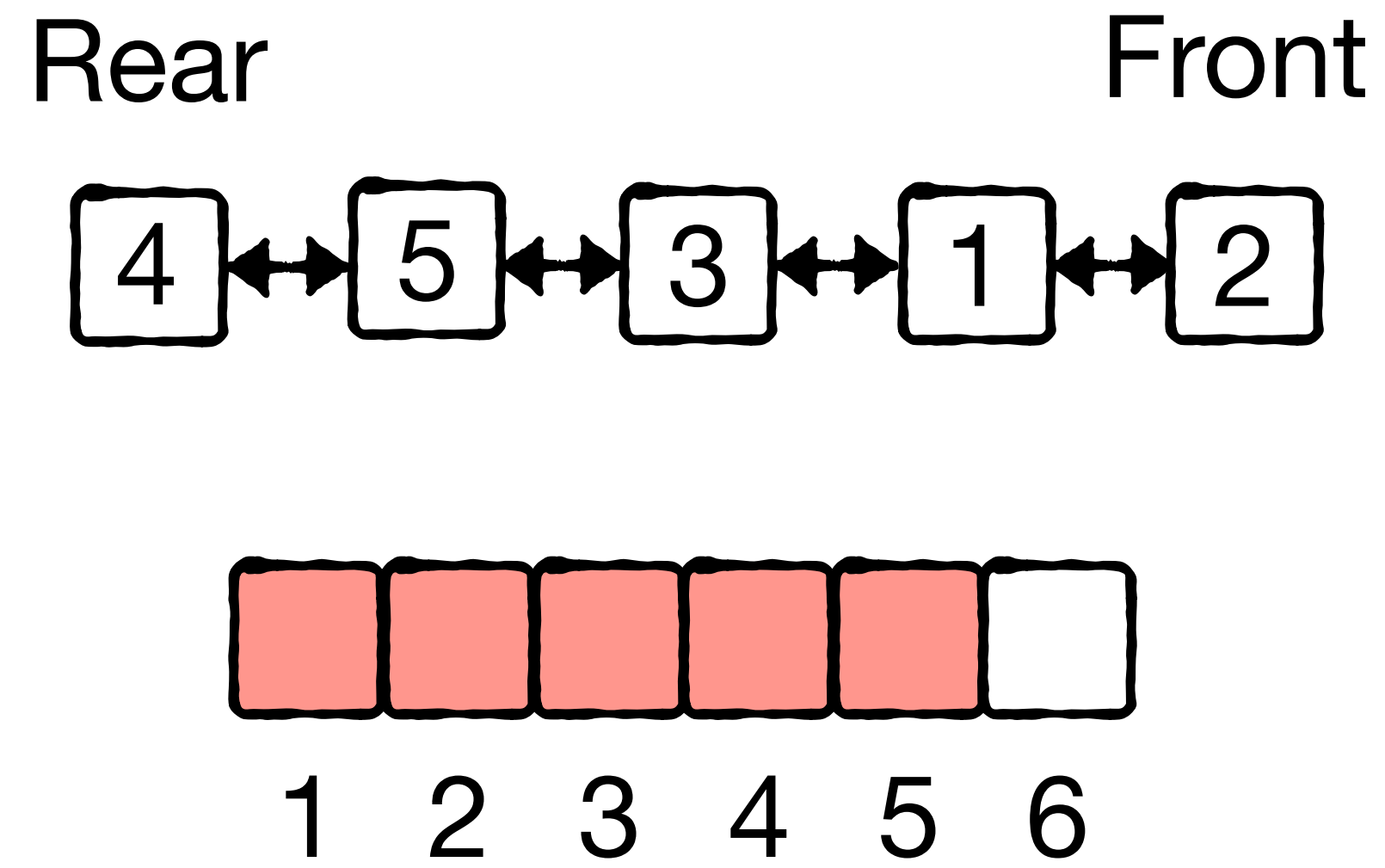


LRU - Least Recently Used

Problems?



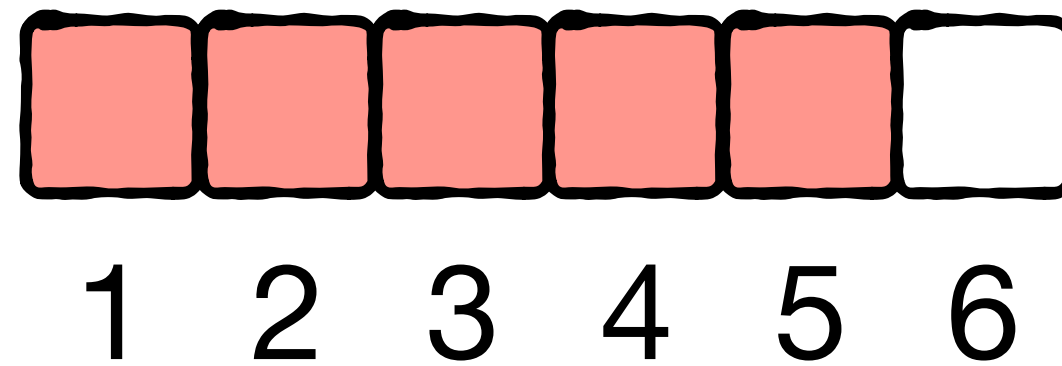
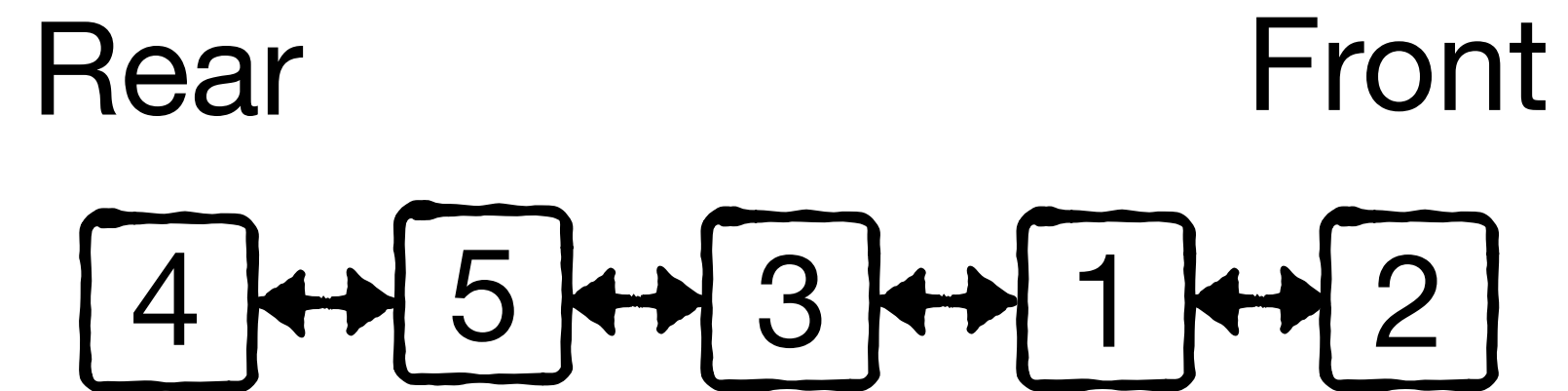
LRU - Least Recently Used



Problems?

(1) CPU - addressed with clock

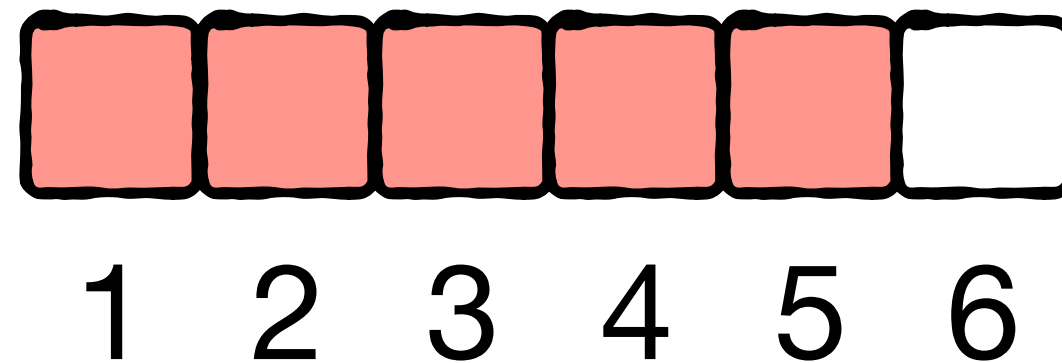
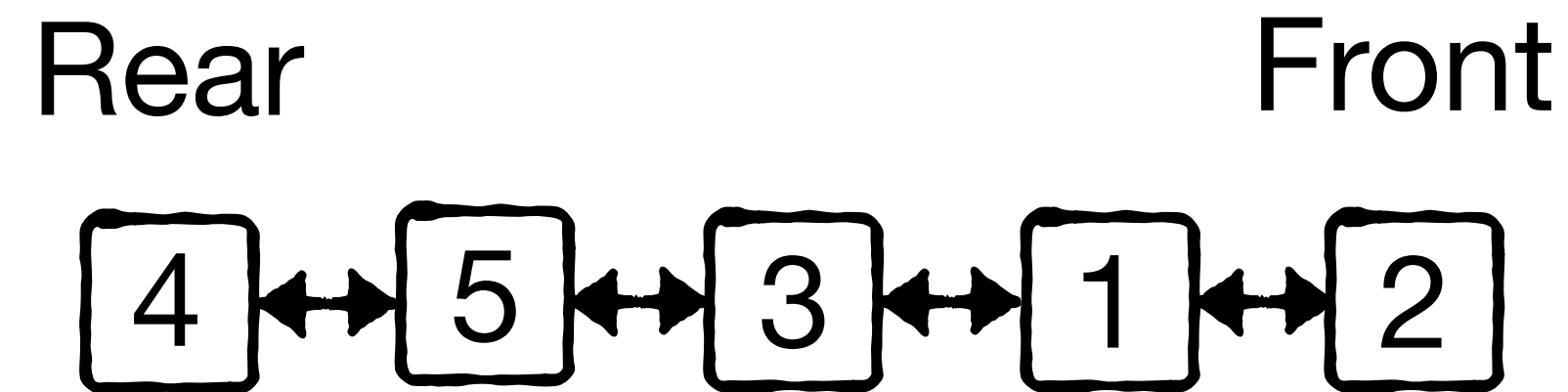
LRU - Least Recently Used



Problems?

- (1) CPU - addressed with clock
- (2) sequential flooding**

LRU - Least Recently Used



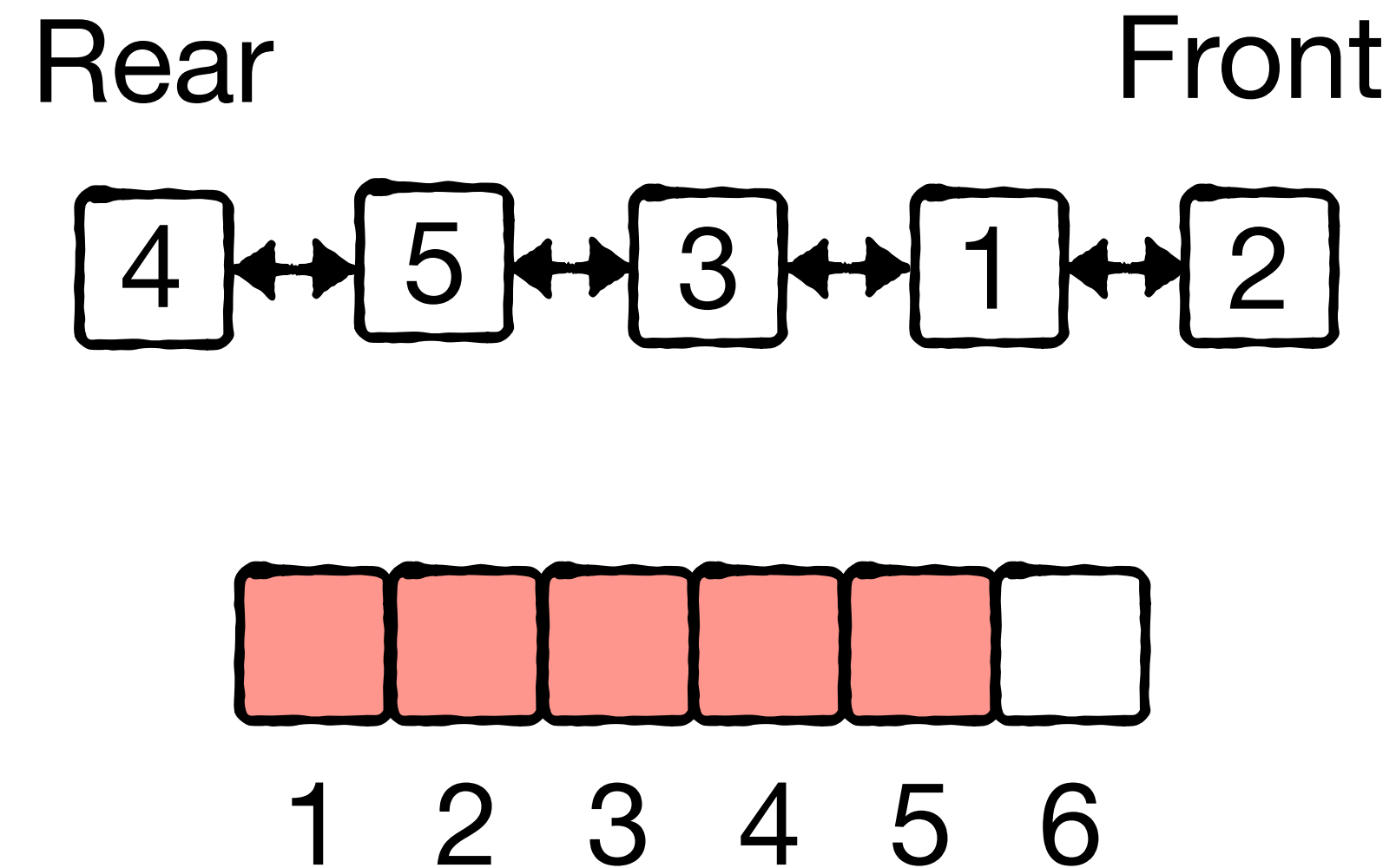
Problems?

(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



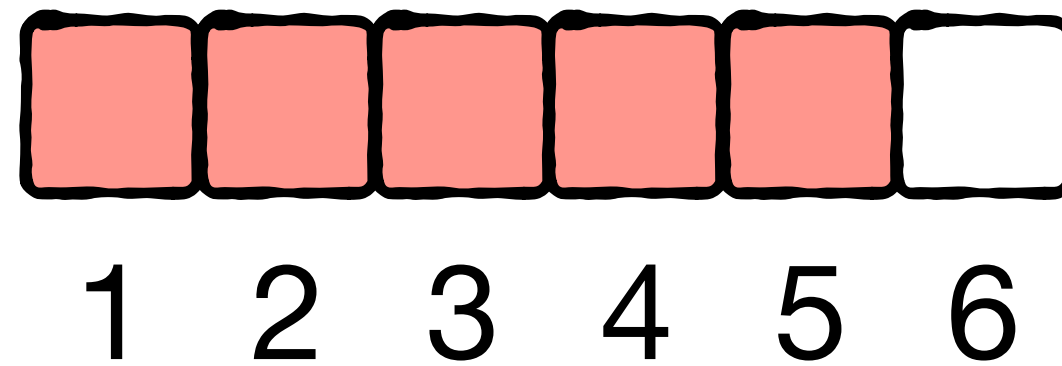
Problems?

- (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



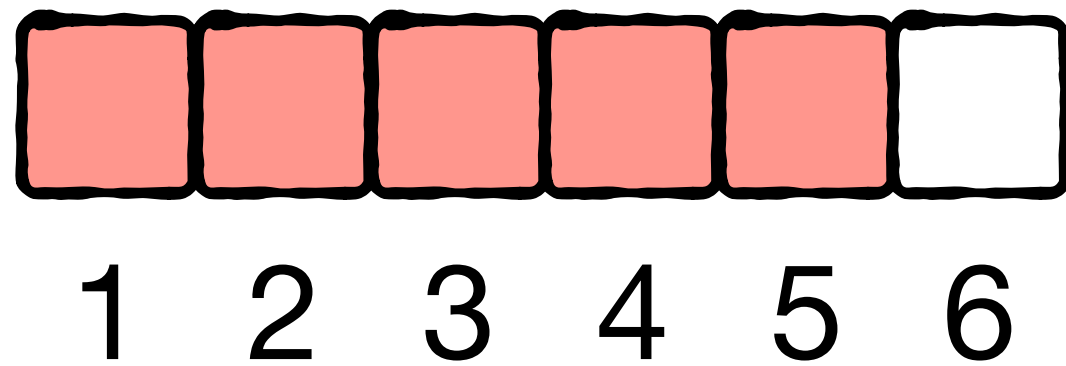
LRU/K - K^{th} Least Recently Used

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

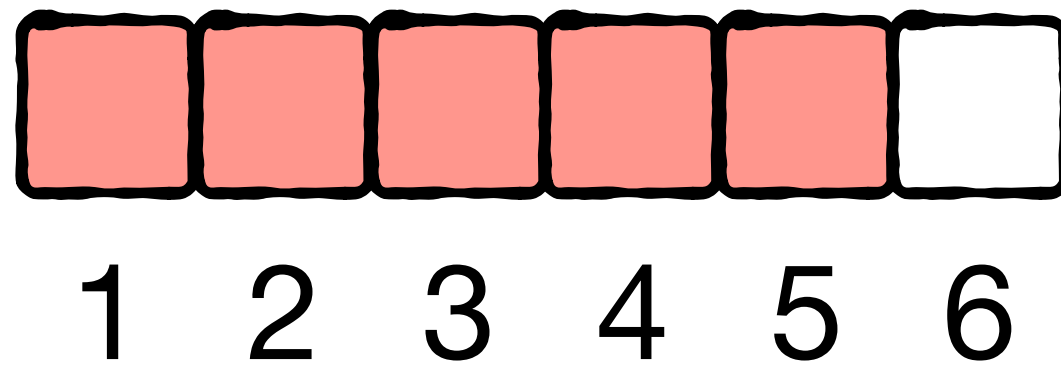
SIGMOD Record 1993



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

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

Generalizes LRU (as LRU/1)

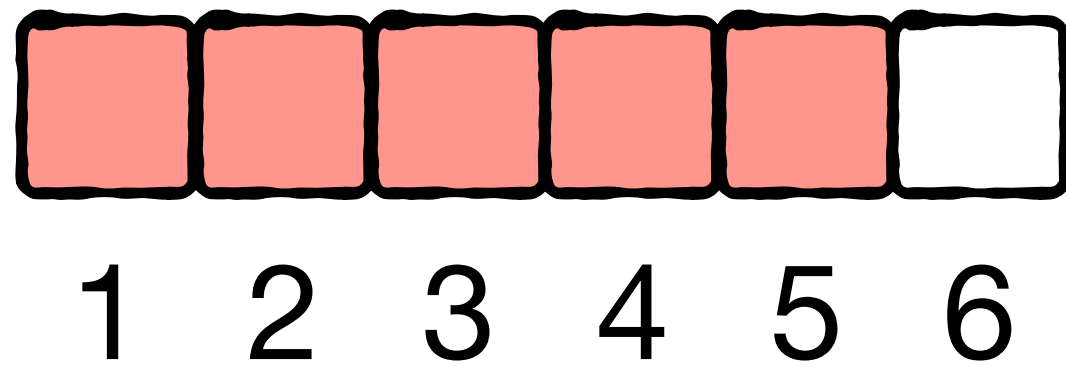


LRU/K - K^{th} Least Recently Used

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



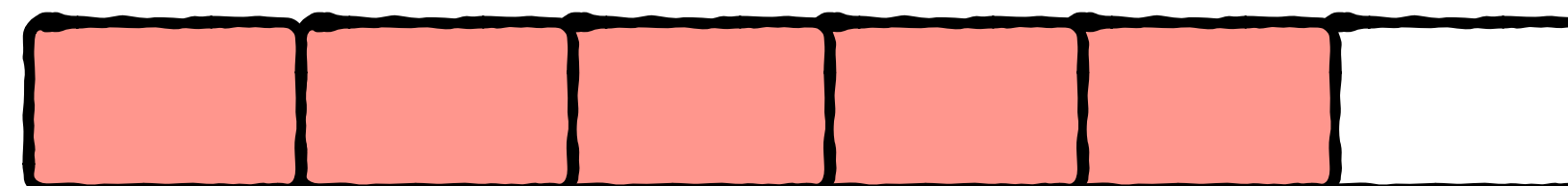
LRU/2 - 2nd Least Recently Used

Last access

10 50 30 40 50

2nd last access

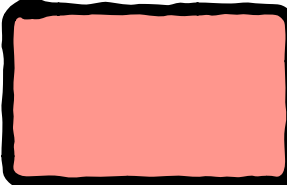
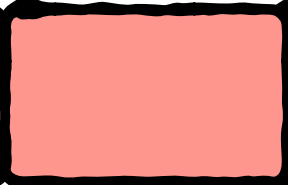
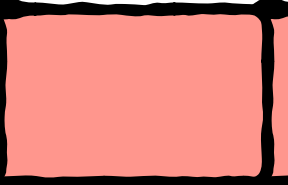
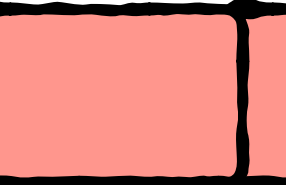
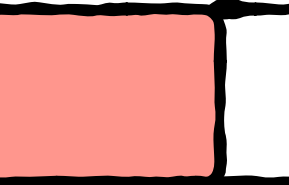
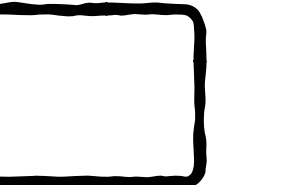
5 15 20 10 0



1 2 3 4 5 6

LRU/2 - 2nd Least Recently Used

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 :)

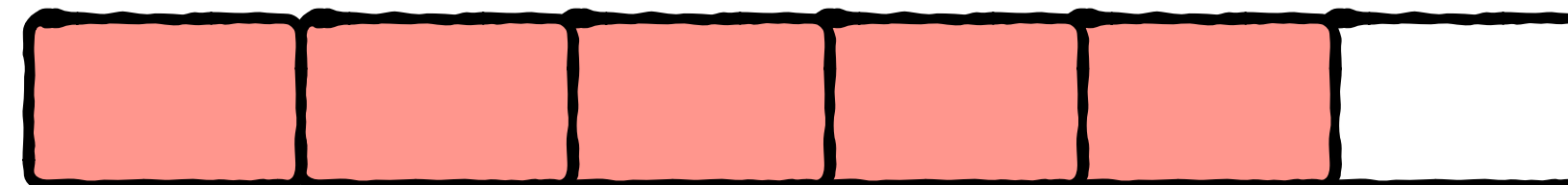


Last access

10 50 30 40 **50**

2nd last access

5 15 20 10 0



1 2 3 4 5 6

Last access
2nd last access

10	50	30	40	50	
5	15	20	10	0	
1	2	3	4	5	6

↑
Evict

Last access
2nd last access

10	50	30	40		
5	15	20	10		
1	2	3	4	5	6

↑
Evict

Last access
2nd last access

10	50	30	40		60
5	15	20	10		$-\infty$
1	2	3	4	5	6



Insert

Time now:
60

Last access

2nd last access

2nd last access

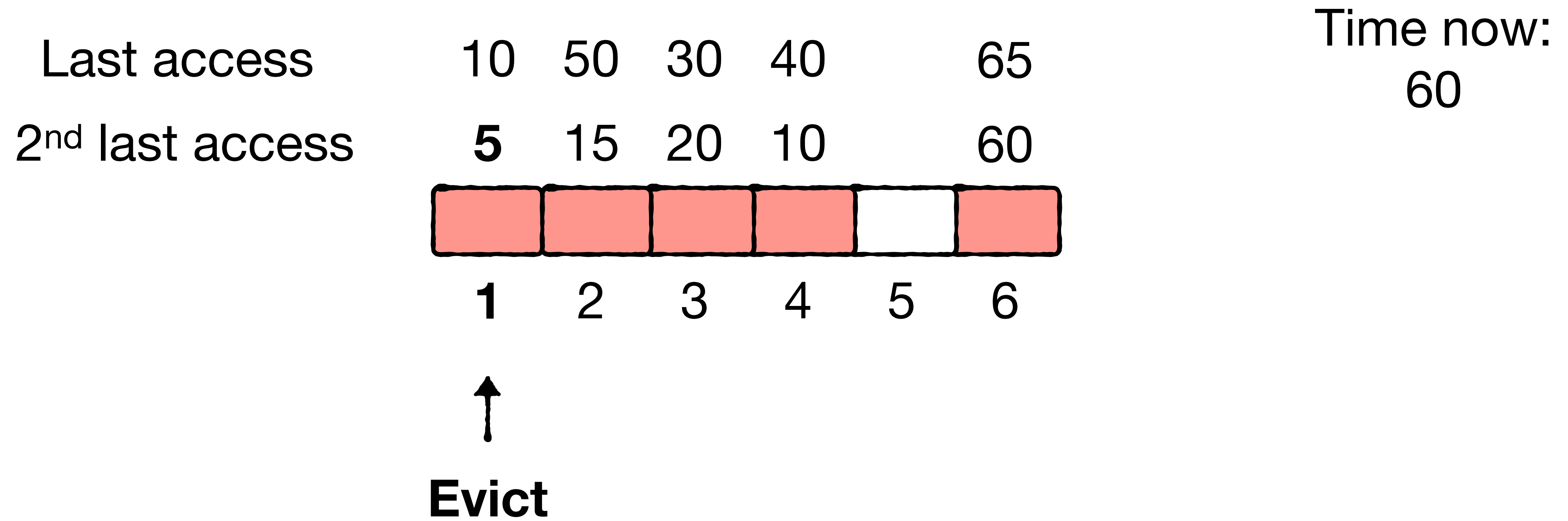
5 15 20 10 60

5 15 20 10 60

1 2 3 4 5 **6**

Access

Time now:
65



Last access

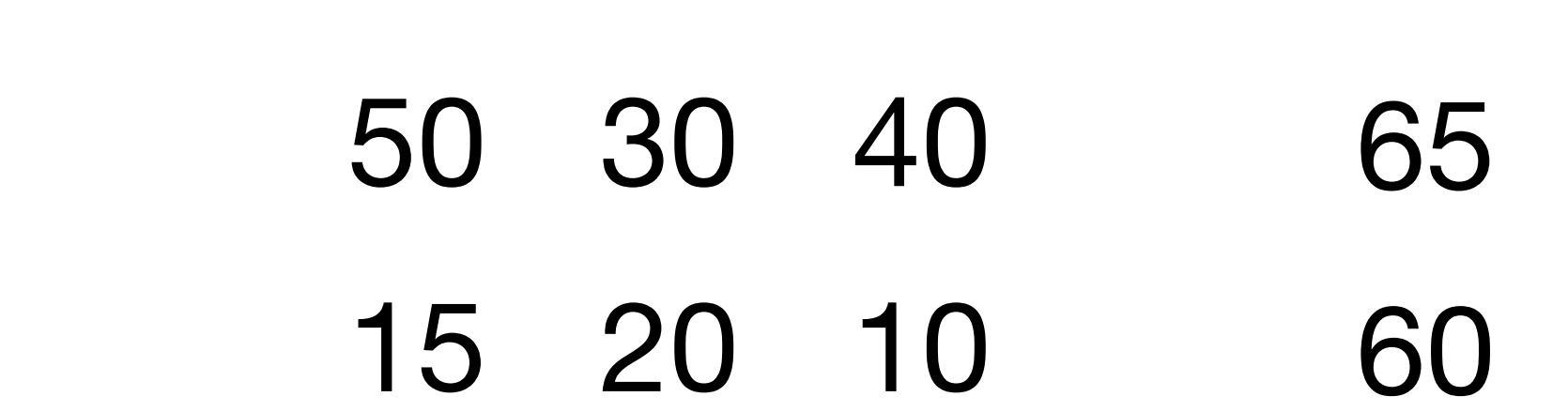
2nd last access

Last access

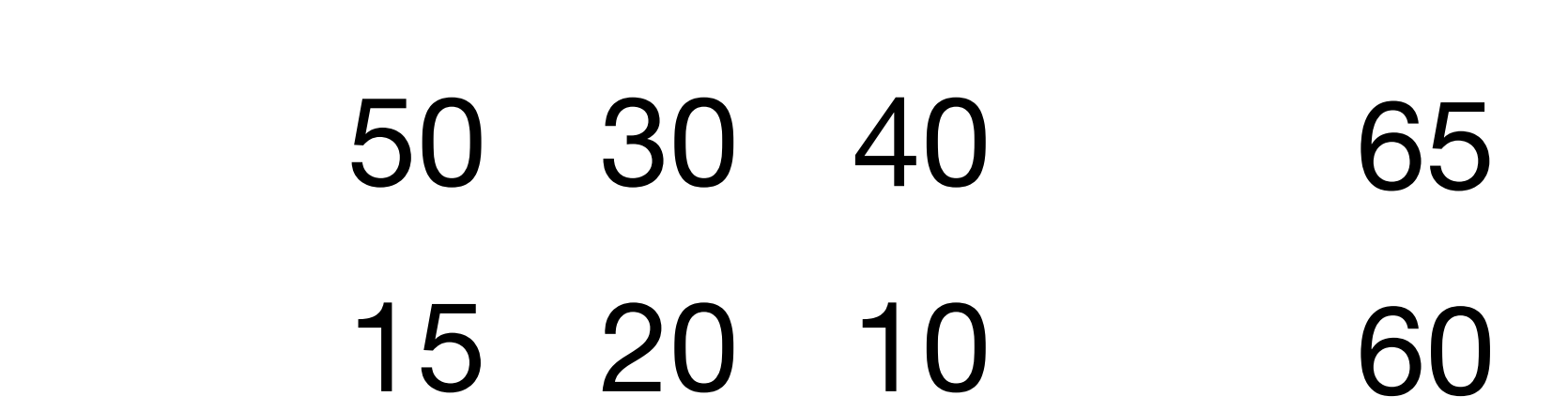
2nd last access

50	30	40	65
15	20	10	60

50	30	40	65
15	20	10	60

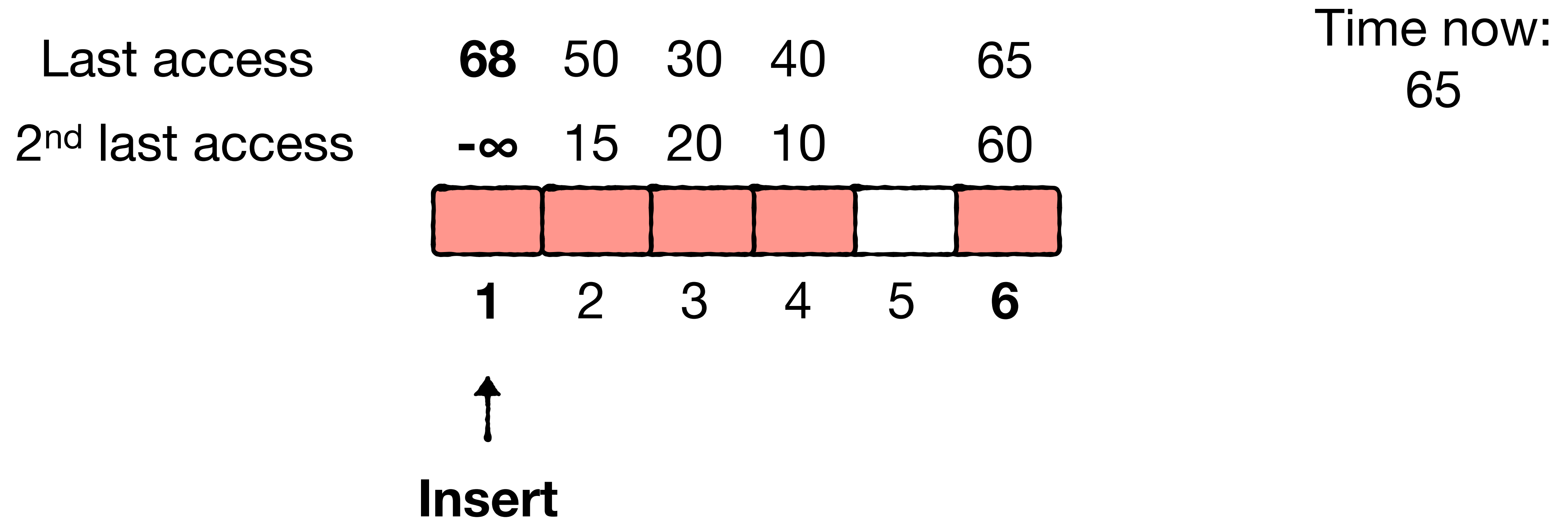


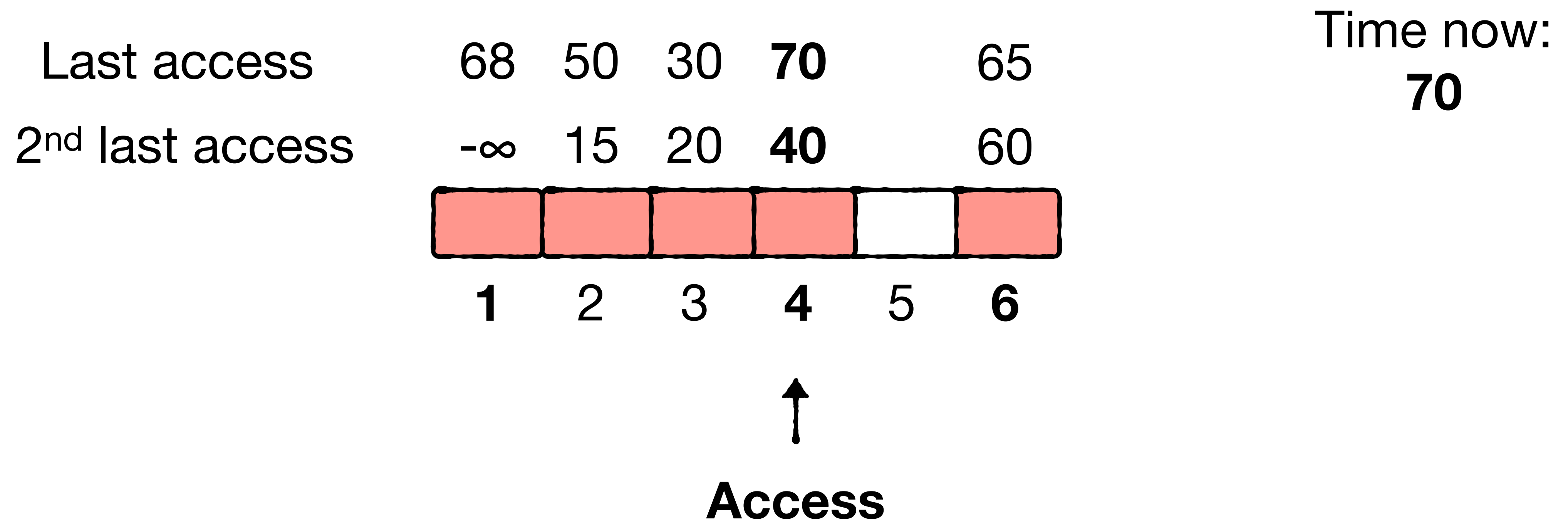
50	30	40	65
15	20	10	60



50	30	40	65
15	20	10	60

Time now:
60





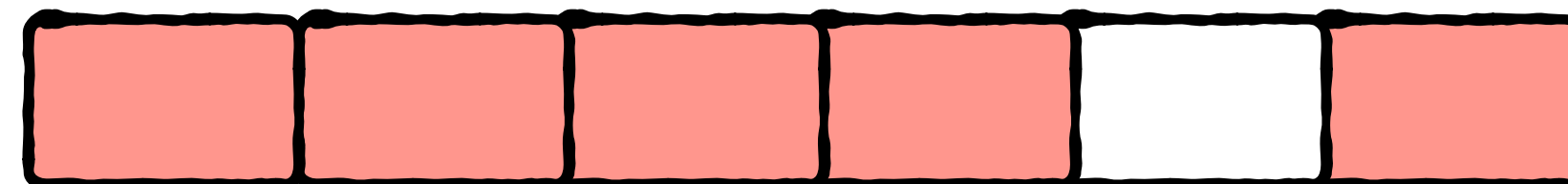
Last access

2nd last access

2nd last access

65

60



6

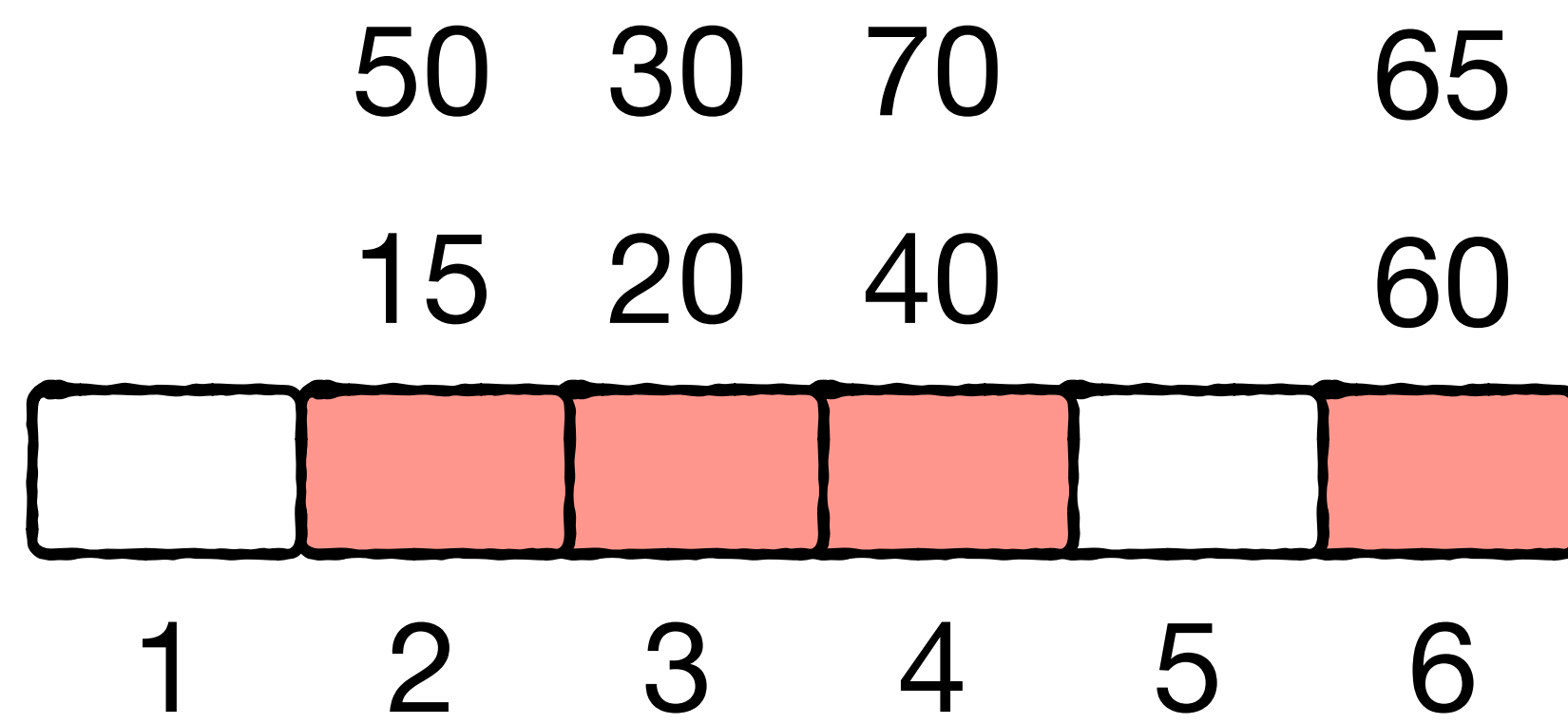
Time now:
70

Last access

2nd last access

Last access

2nd last access



Evict

Time now:
70

Last access

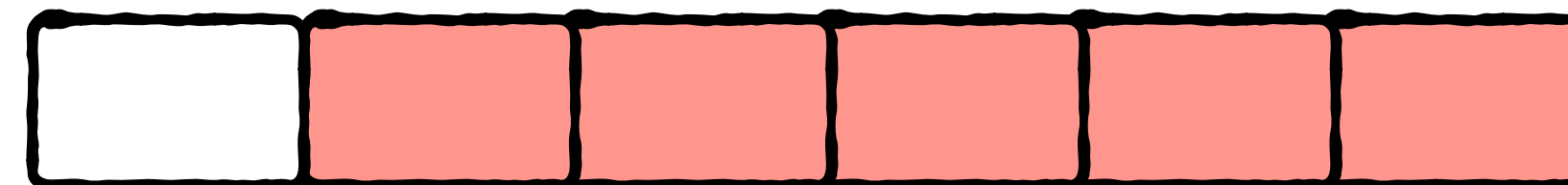
2nd last access

Last access

2nd last access

50 30 70 **75** 65

15 20 40 $-\infty$ 60



1 2 3 4 **5** 6



Insert

Time now:
75

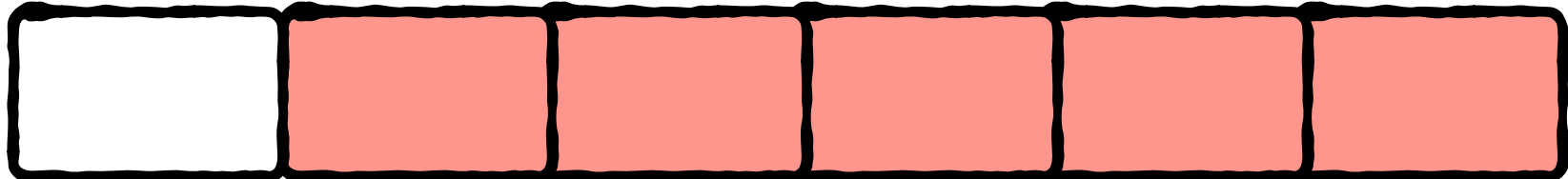
How to find 2nd LRU entry?

Last access

50 30 70 75 65

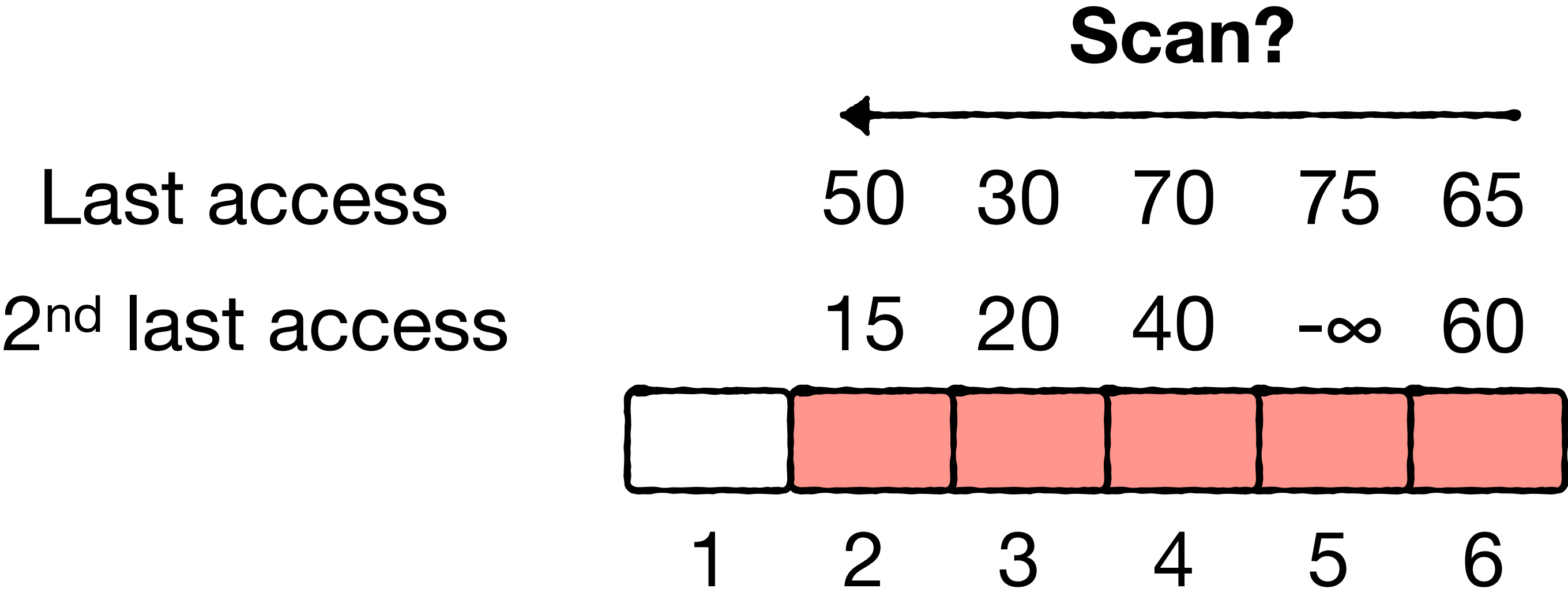
2nd last access

15 20 40 -∞ 60

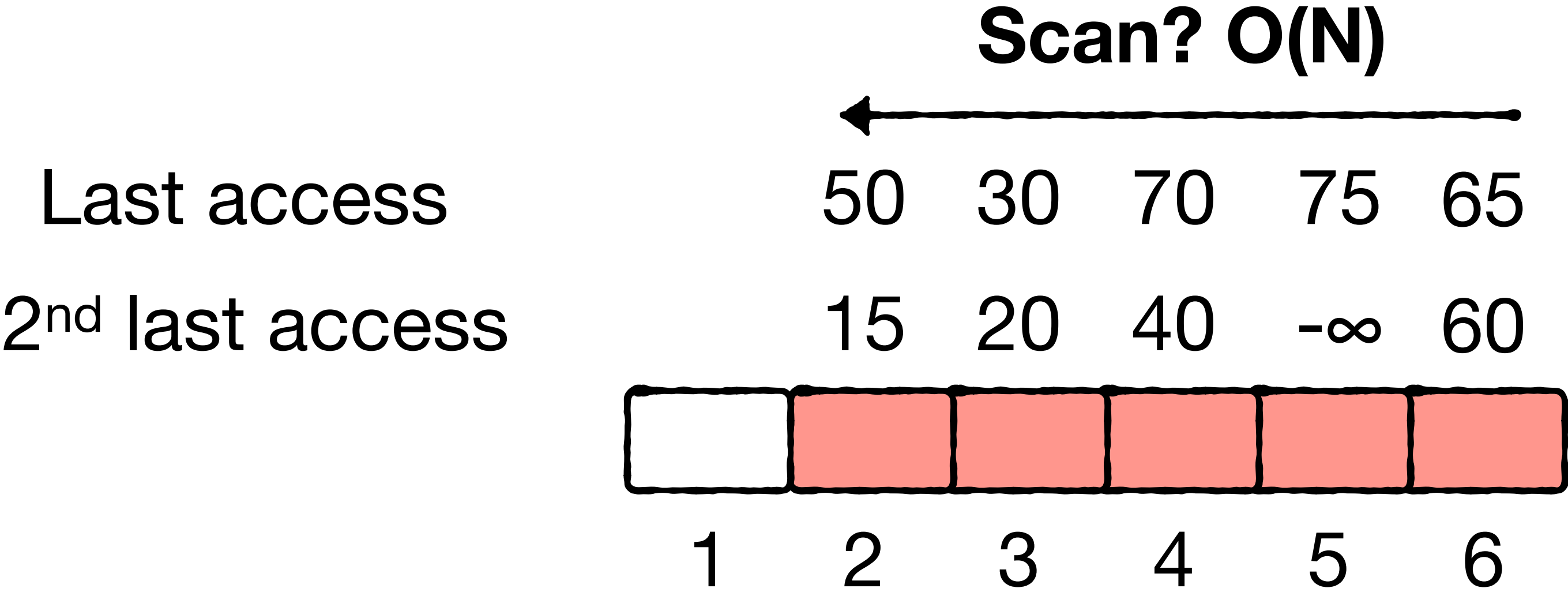


1 2 3 4 5 6

How to find 2nd LRU entry?

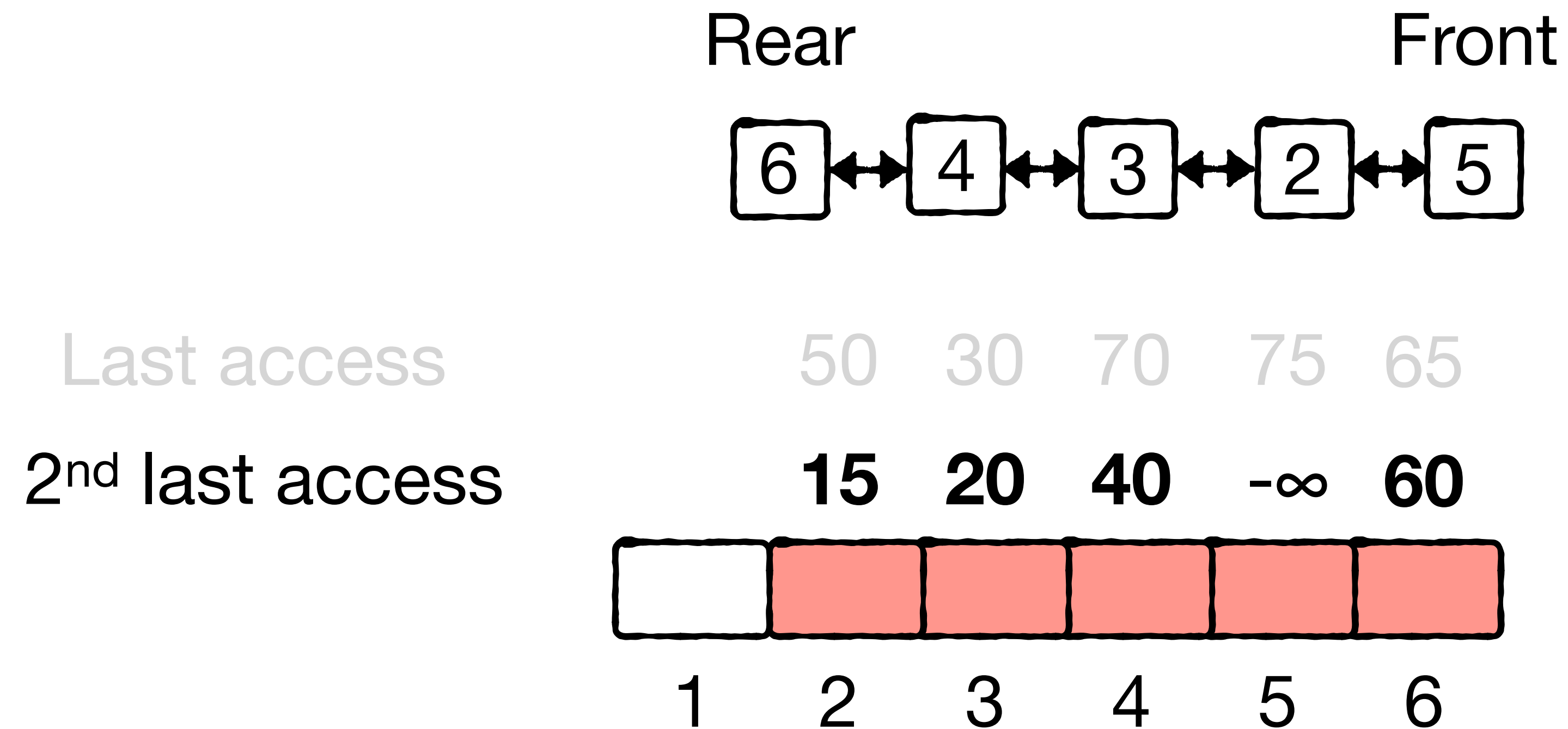


How to find 2nd LRU entry?



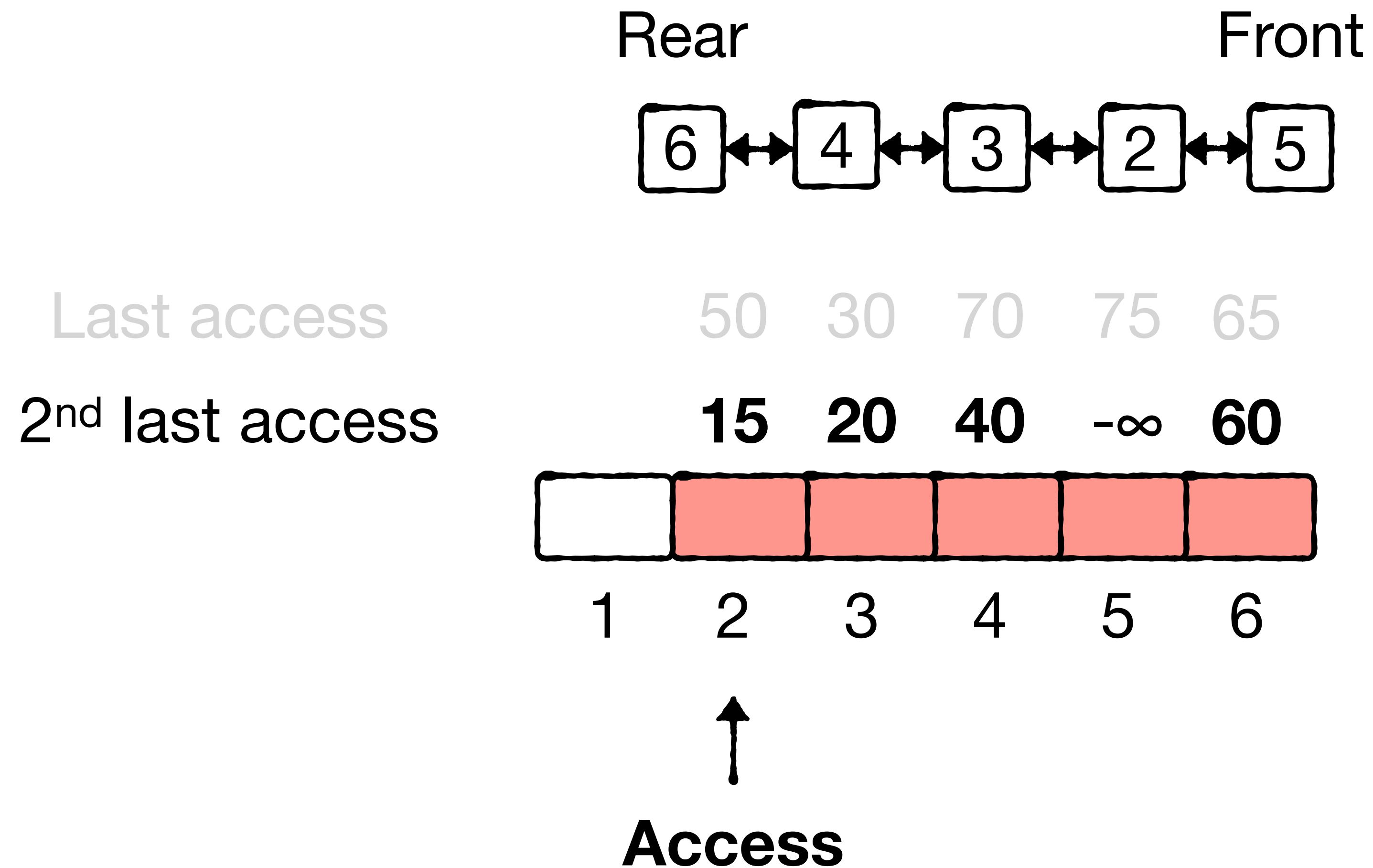
How to find 2nd LRU entry?

Consider linked list



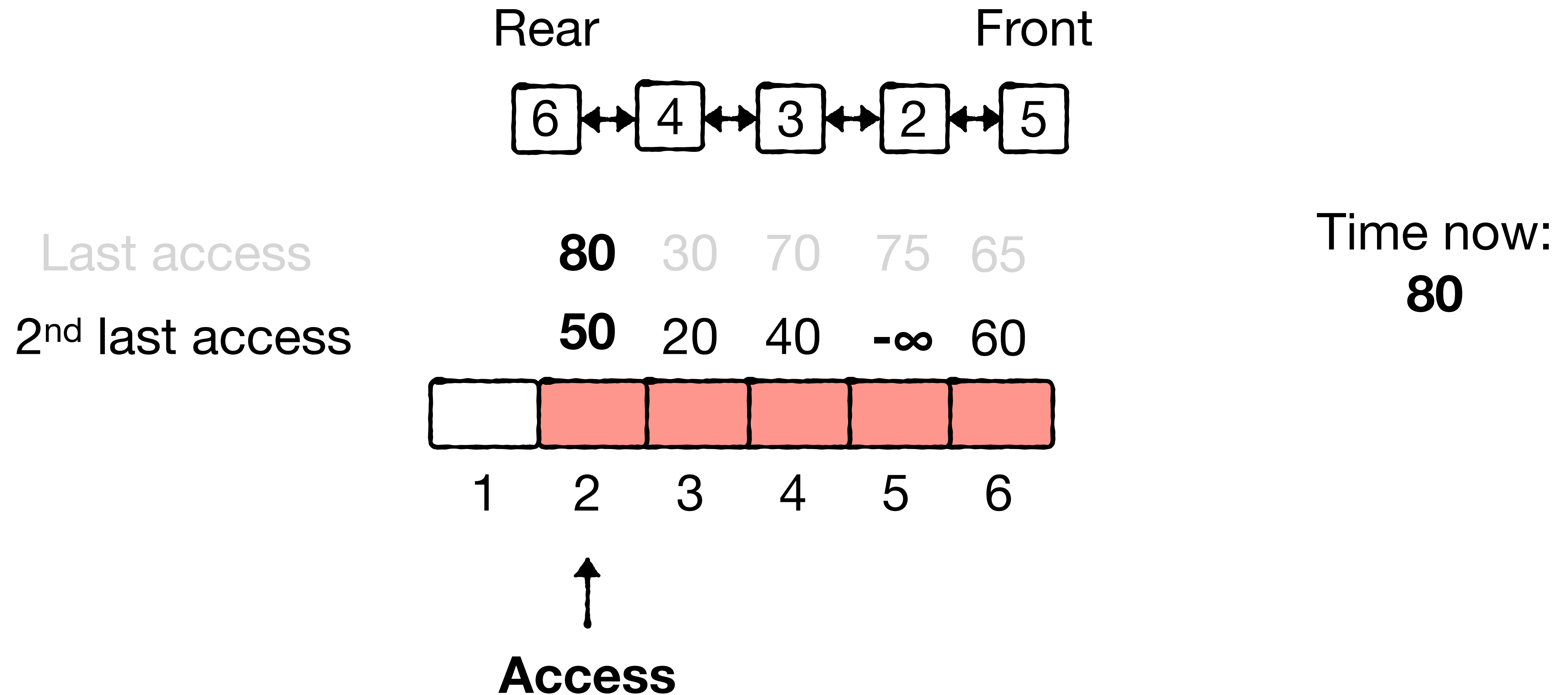
How to find 2nd LRU entry?

Consider linked list



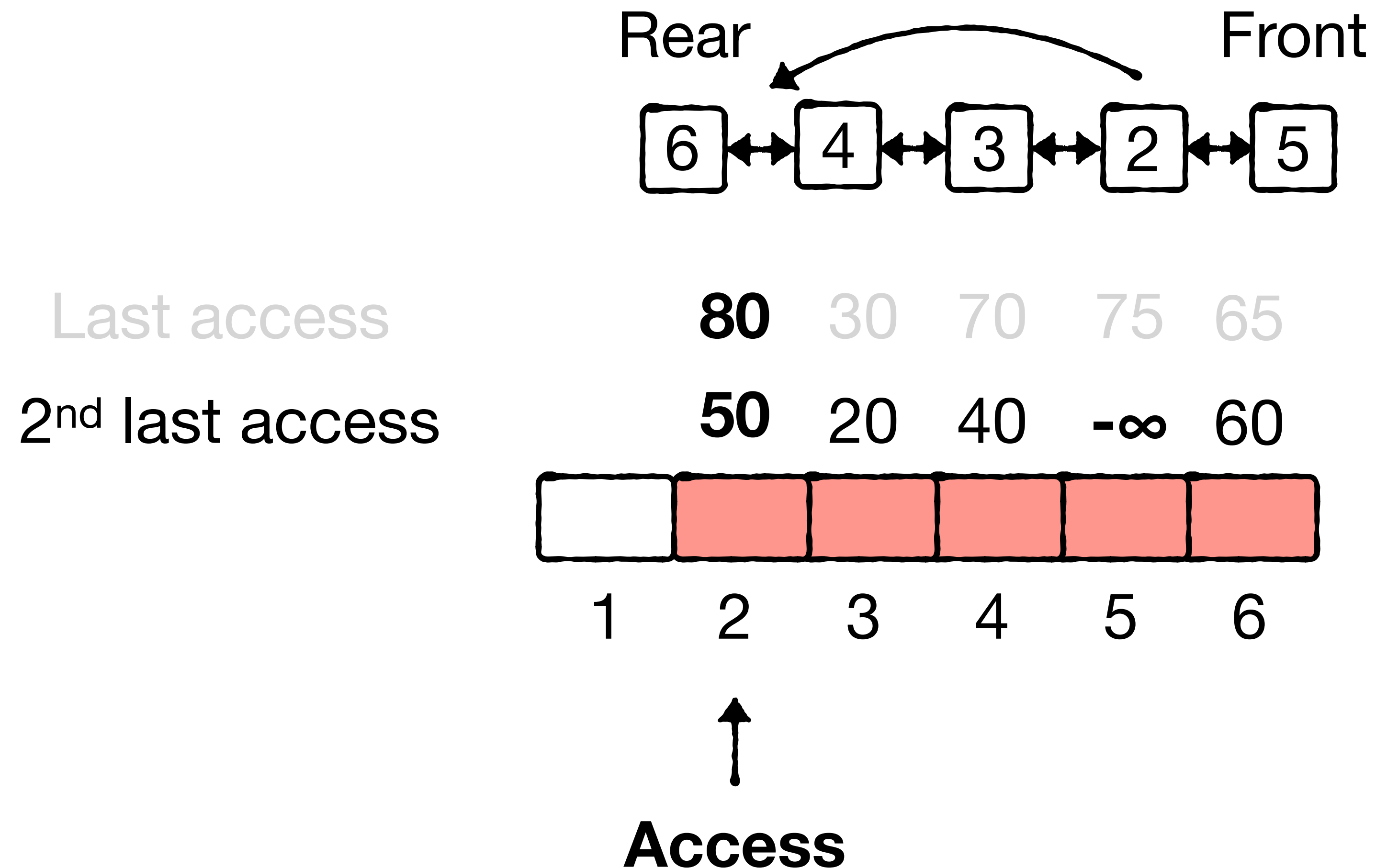
How to find 2nd LRU entry?

Consider linked list



How to find 2nd LRU entry?

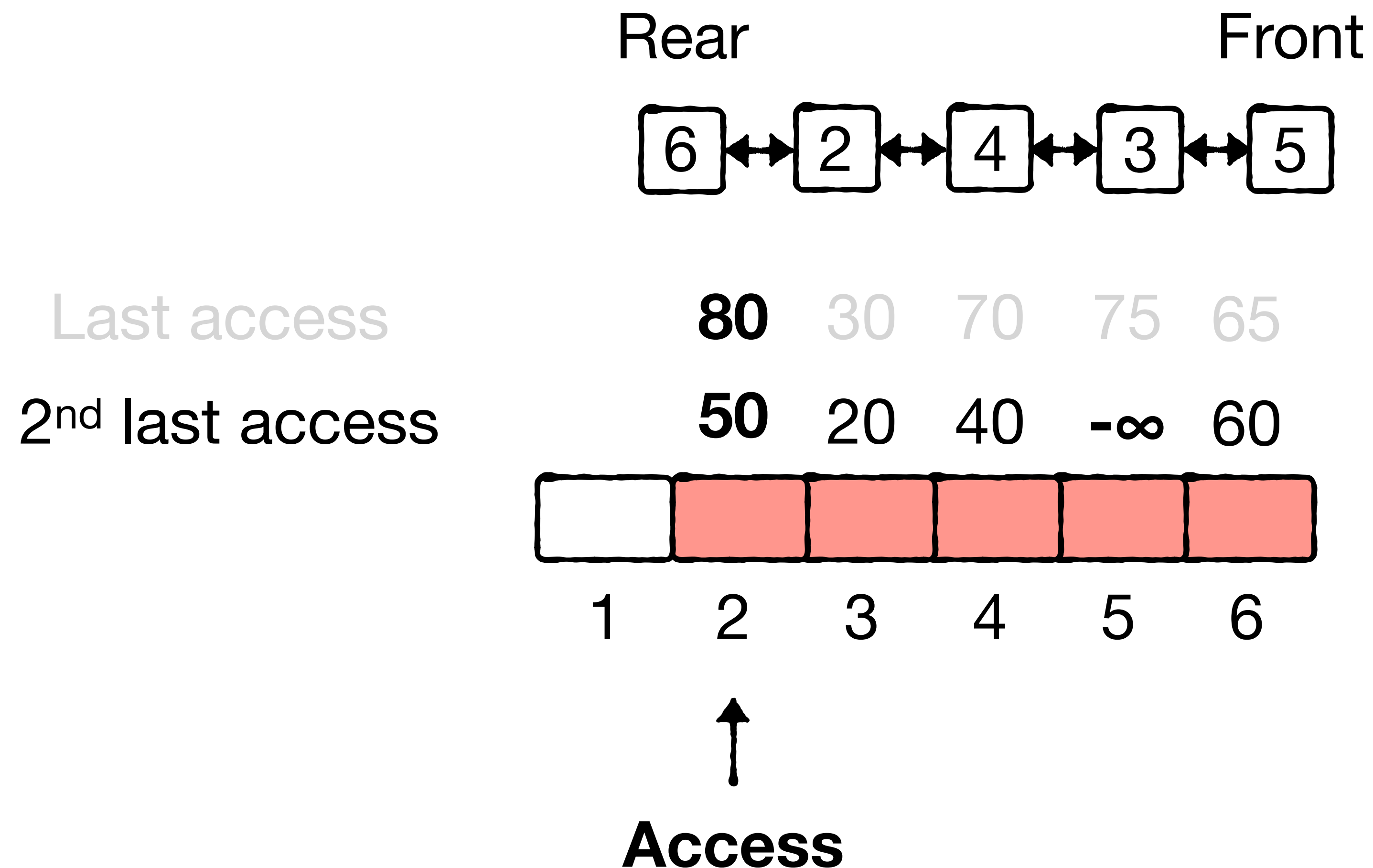
Consider linked list



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

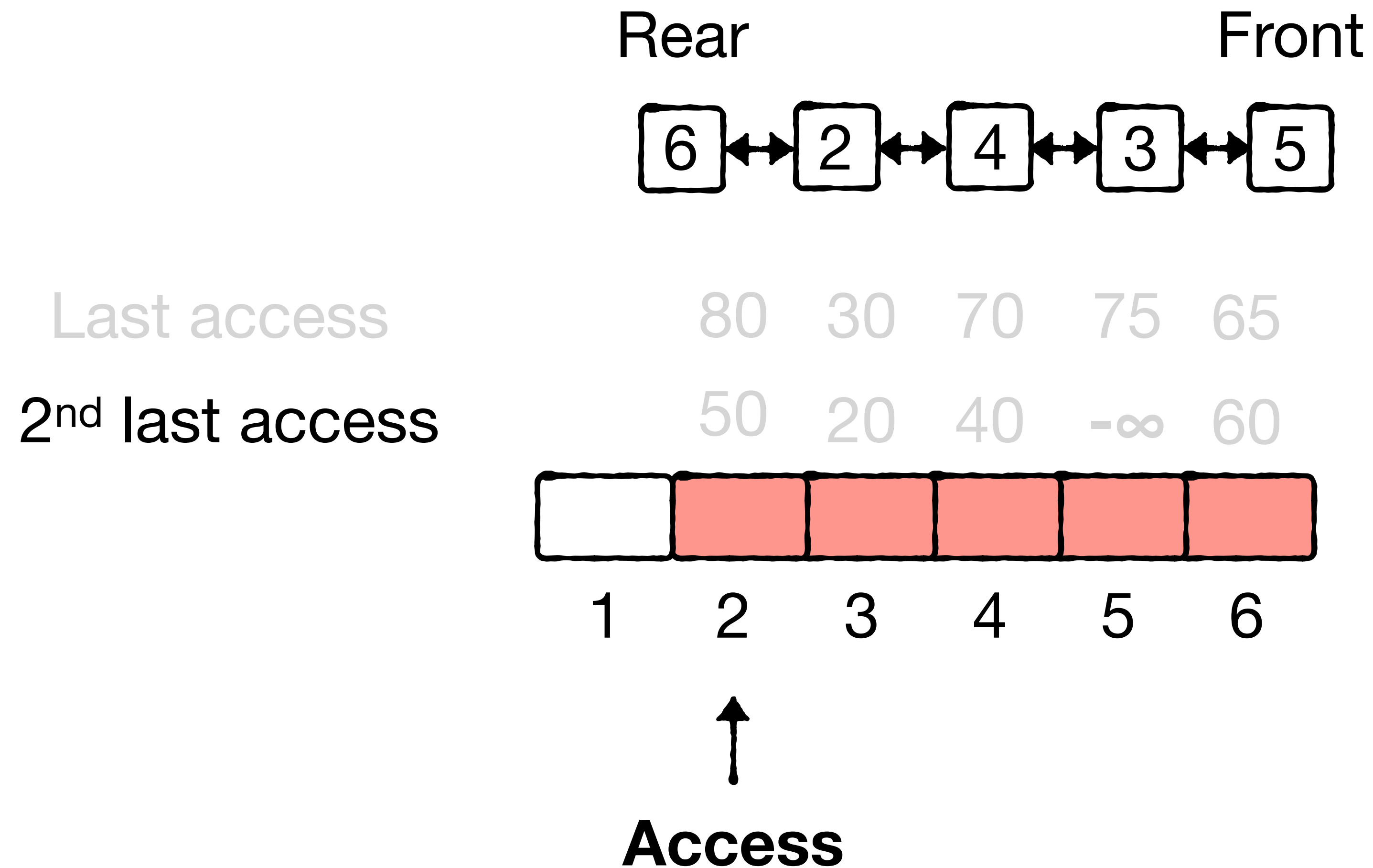
How to find 2nd LRU entry?

Consider linked list



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?



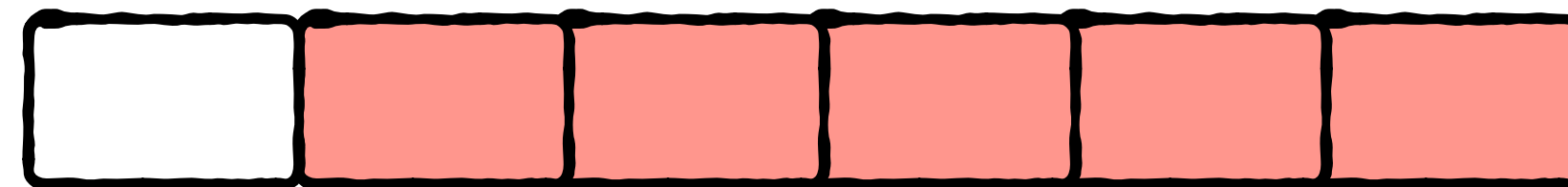
Min-Heap

Last access

80 30 70 75 65

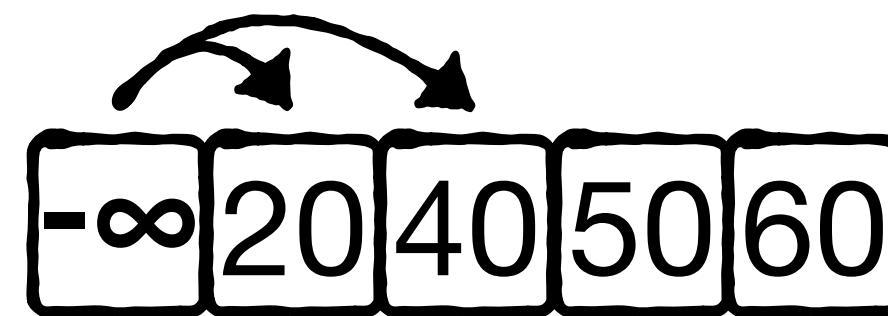
2nd last access

50 20 40 $-\infty$ 60



1 2 3 4 5 6

Min-Heap

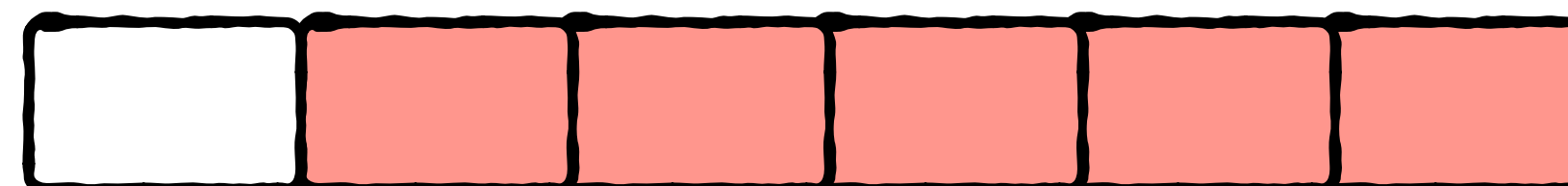


Last access

80 30 70 75 65

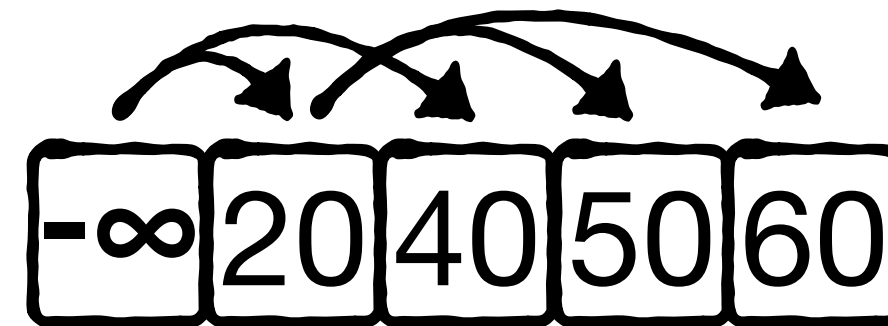
2nd last access

50 20 40 $-\infty$ 60



1 2 3 4 5 6

Min-Heap

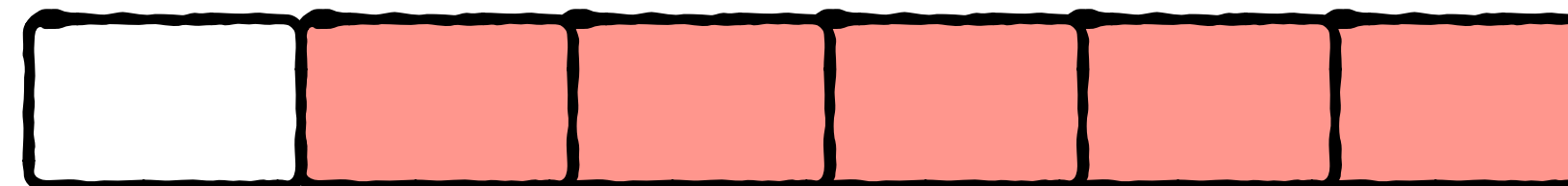


Last access

80 30 70 75 65

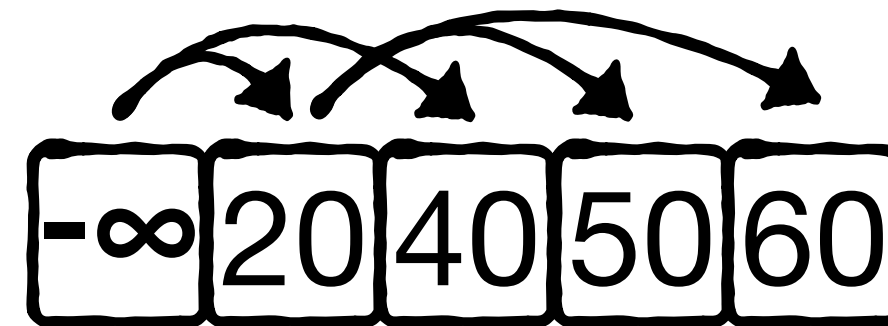
2nd last access

50 20 40 $-\infty$ 60



1 2 3 4 5 6

Min-Heap

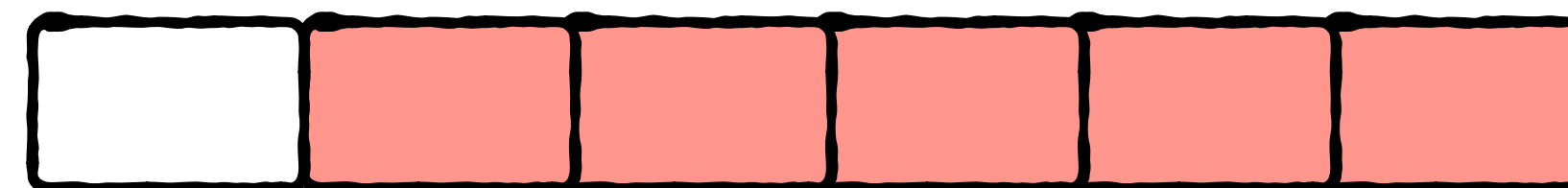


Last access

80 30 70 **75** 65

2nd last access

50 20 40 $-\infty$ 60



1 2 3 4 5 6

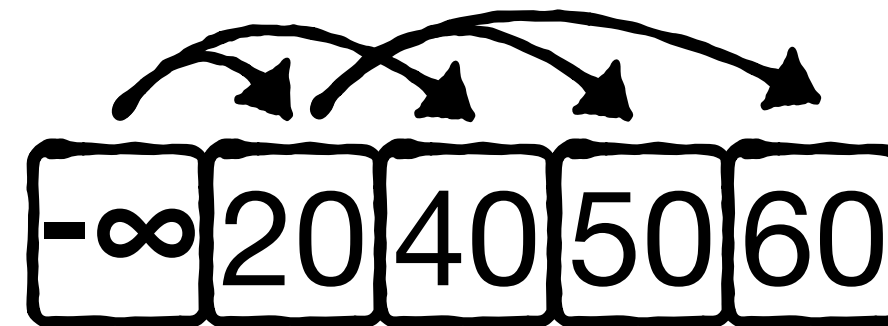


Access

Time now:

90

Min-Heap

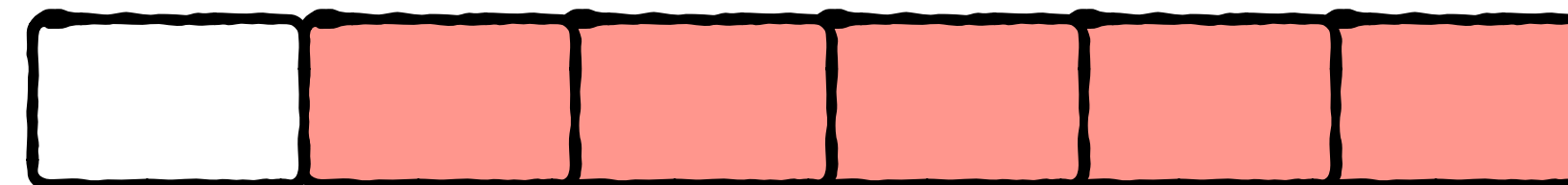


Last access

80 30 70 **90** 65

2nd last access

50 20 40 **75** 60



1 2 3 4 5 6

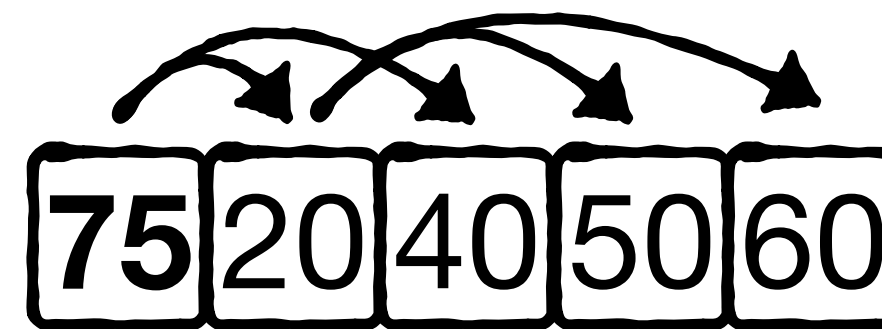


Access

Time now:

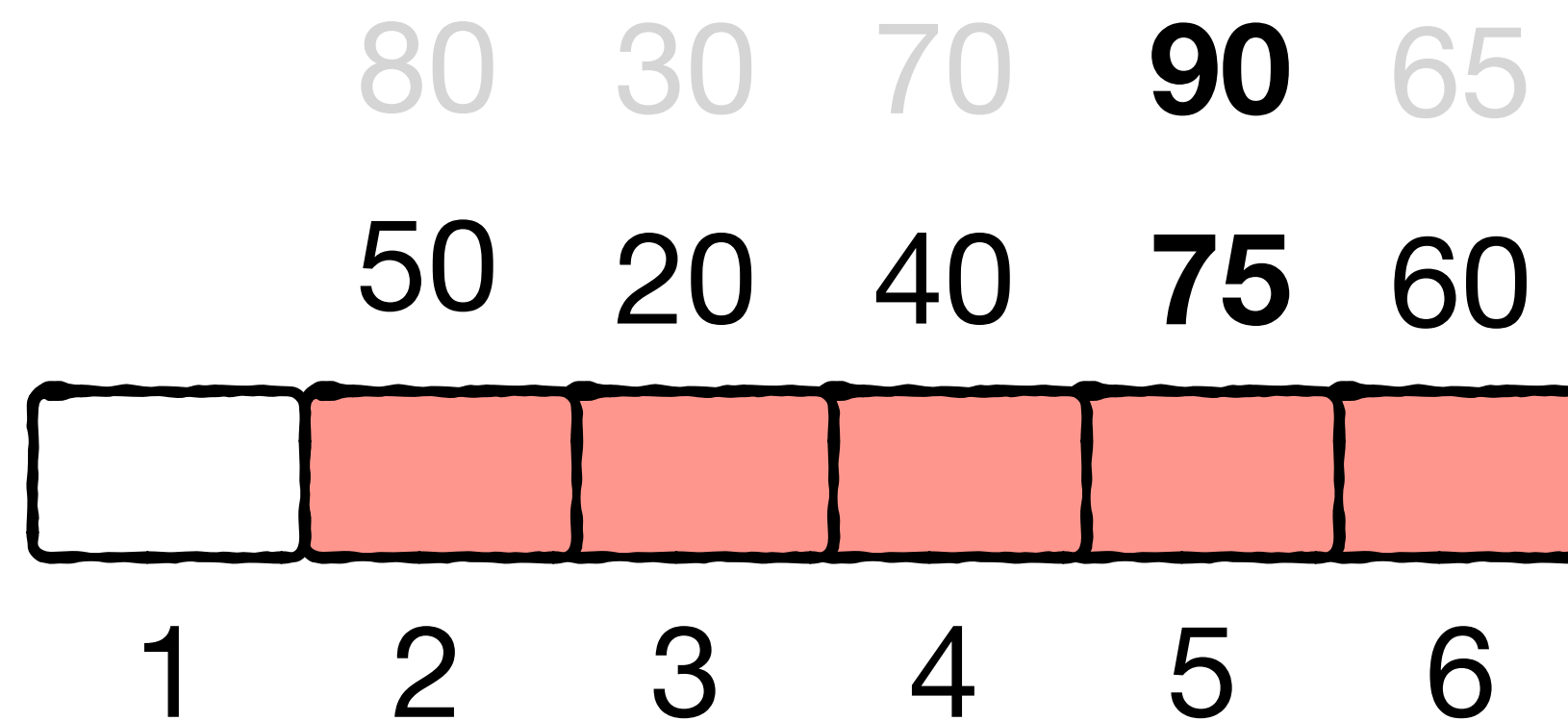
90

Min-Heap



Last access

2nd last access

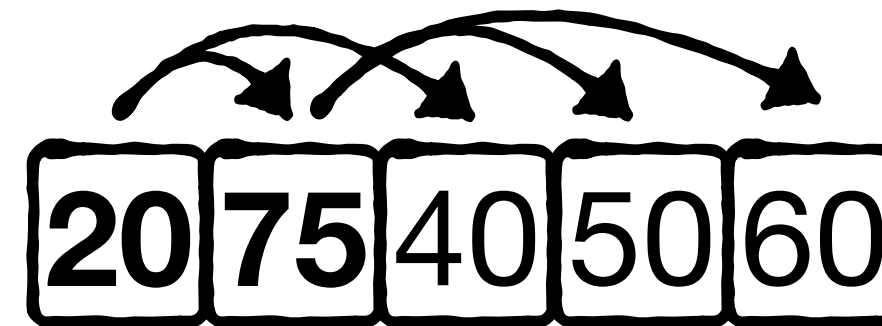


Access

Time now:

90

Min-Heap

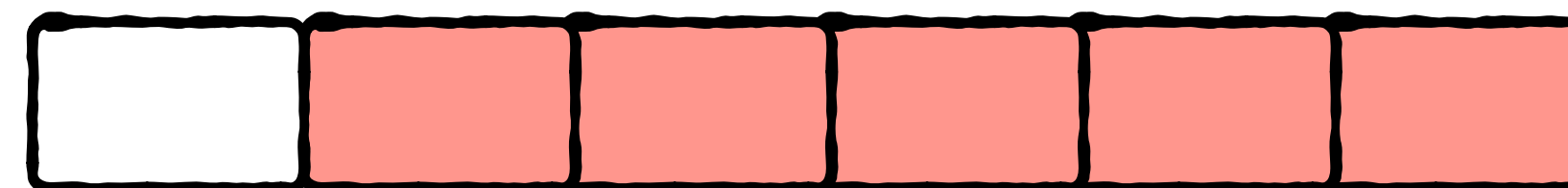


Last access

2nd last access

80 30 70 **90** 65

50 20 40 **75** 60



1 2 3 4 5 6

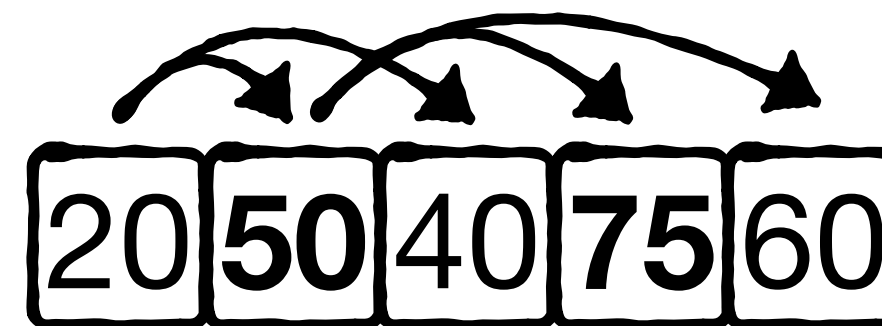


Access

Time now:

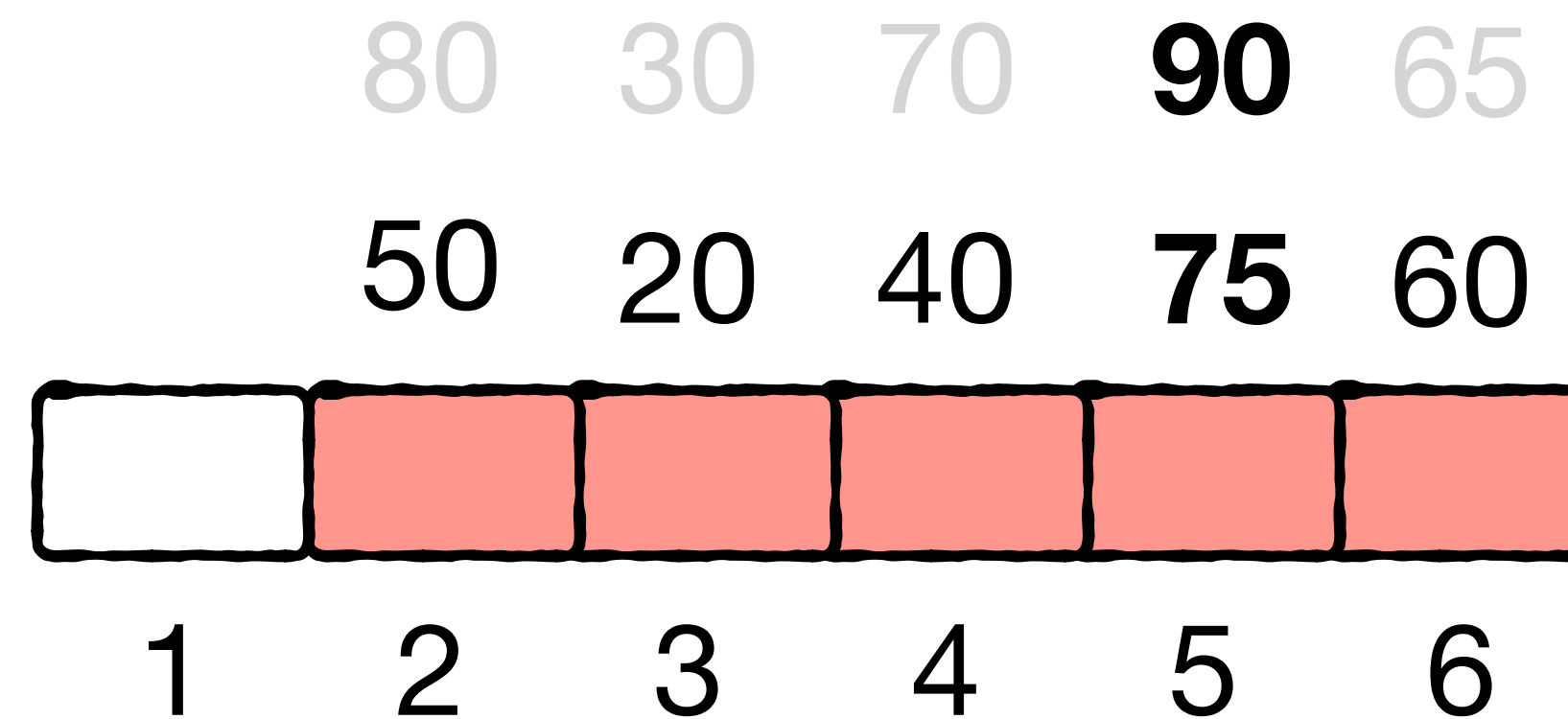
90

Min-Heap



Last access

2nd last access

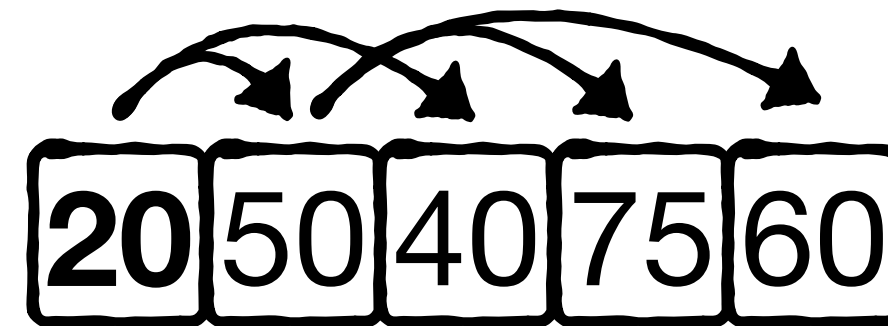


Access

Time now:

90

Min-Heap

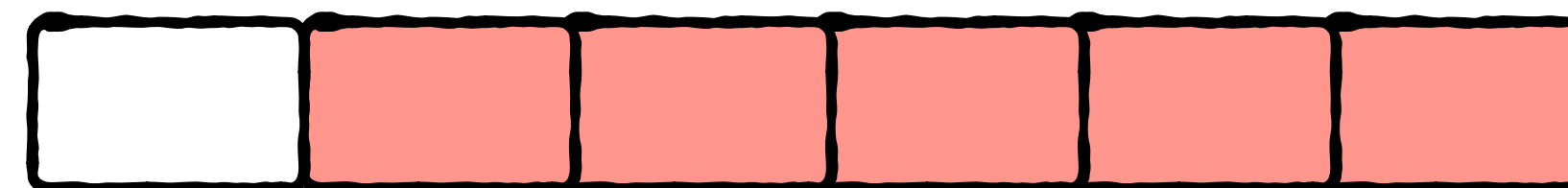


Last access

80 30 70 90 65

2nd last access

50 **20** 40 75 60

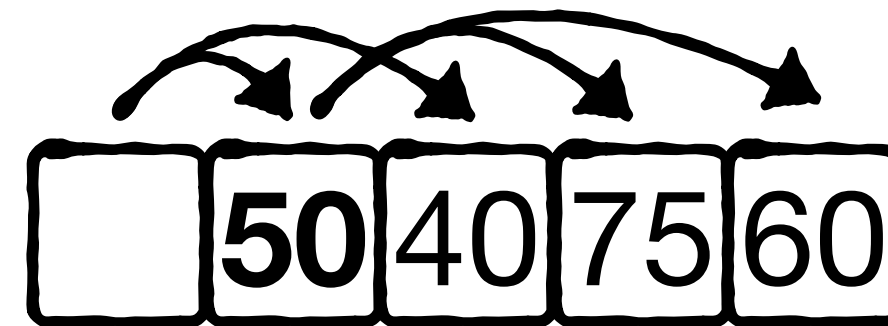


1 2 3 4 5 6



Evict

Min-Heap

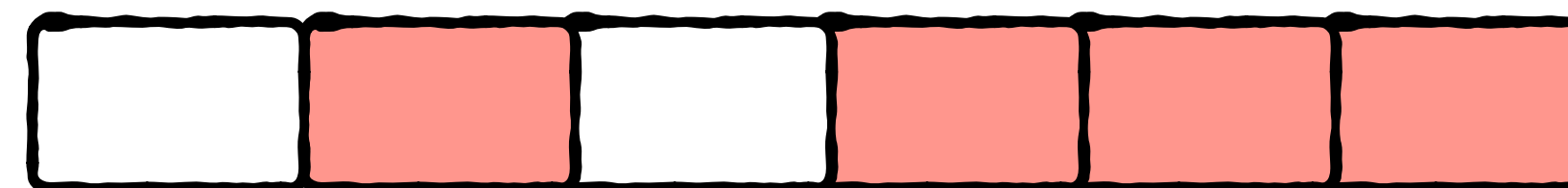


Last access

80 70 90 65

2nd last access

50 40 75 60

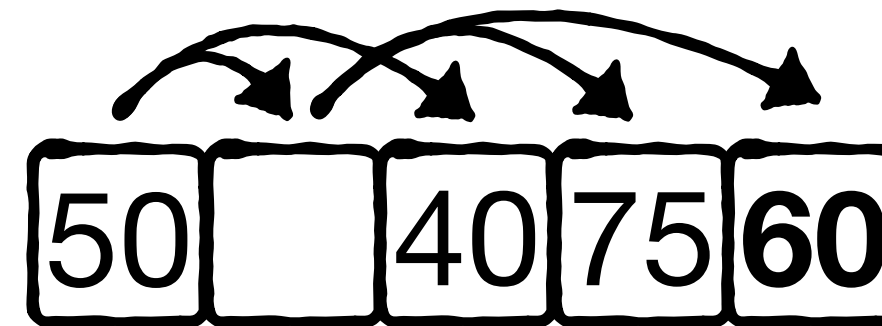


1 2 3 4 5 6



Evict

Min-Heap

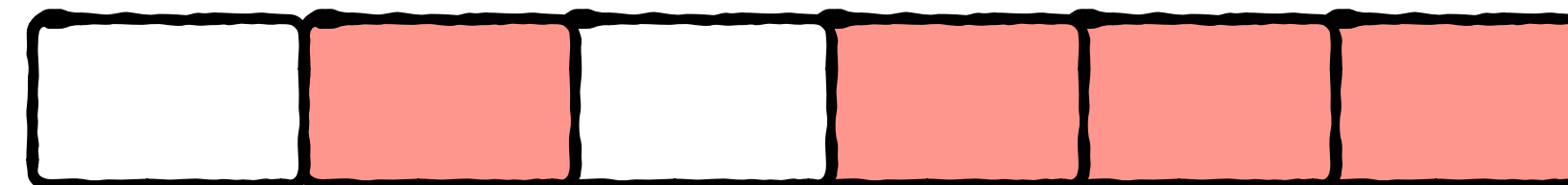


Last access

80 70 90 65

2nd last access

50 40 75 60

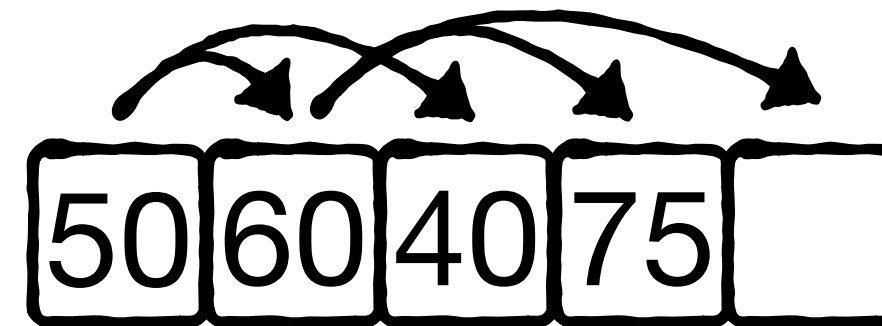


1 2 3 4 5 6



Evict

Min-Heap

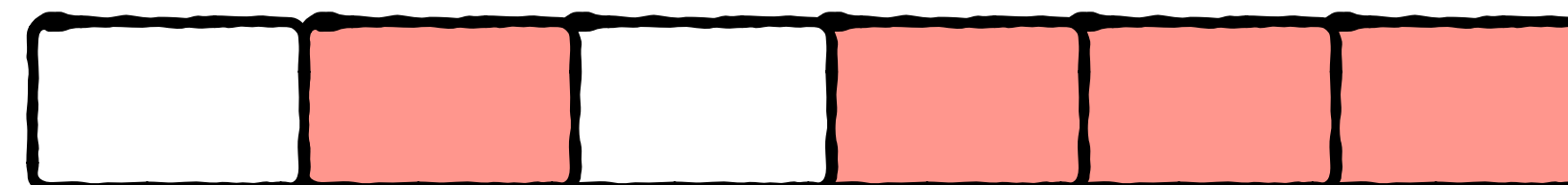


Last access

80 70 90 65

2nd last access

50 40 75 60



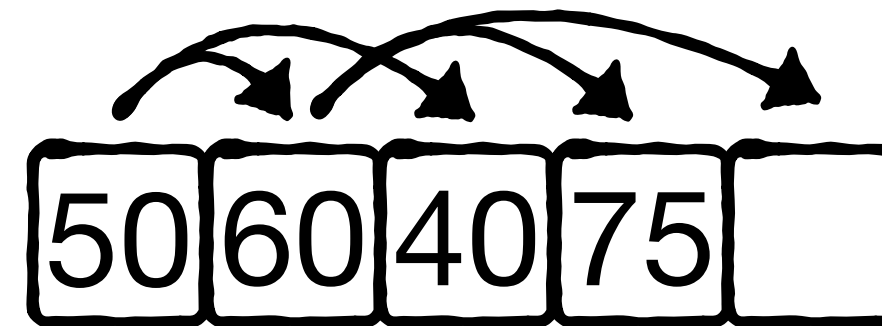
1 2 3 4 5 6



Evict

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

Min-Heap

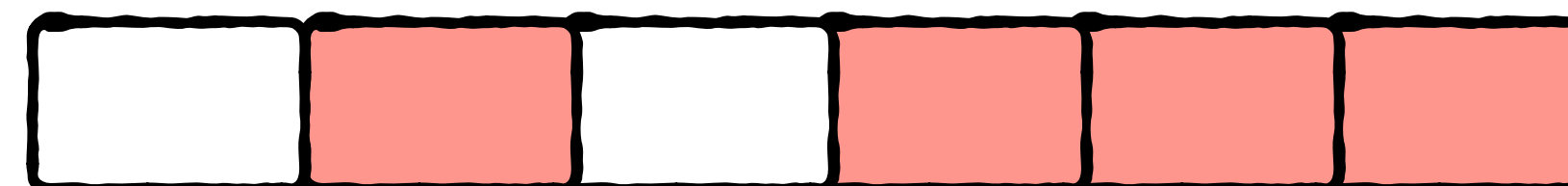


Last access

80 70 90 65

2nd last access

50 40 75 60



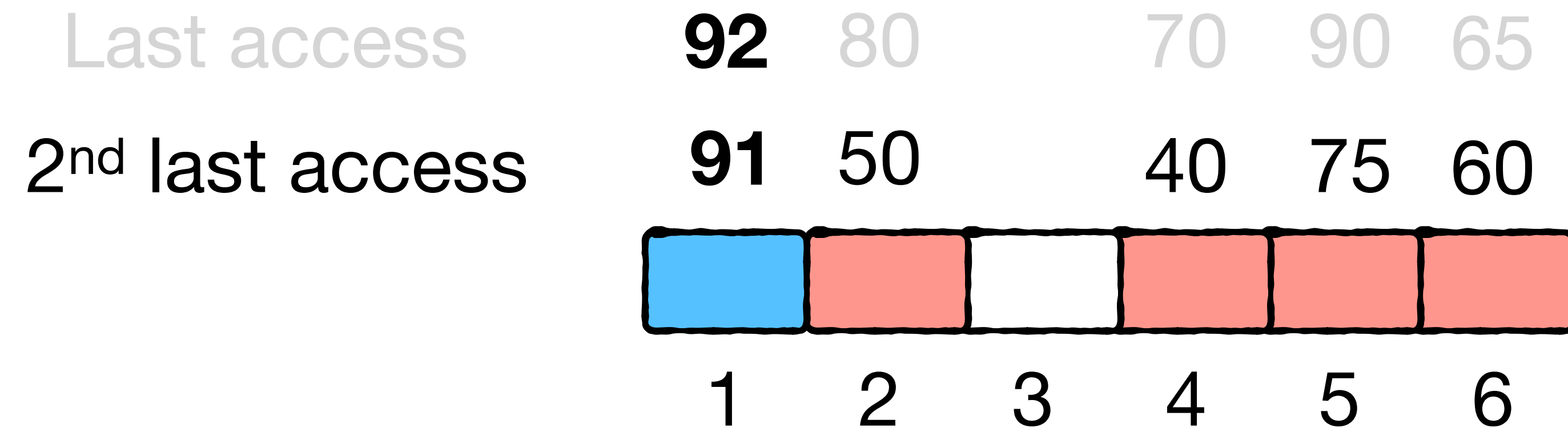
1 2 3 4 5 6

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)

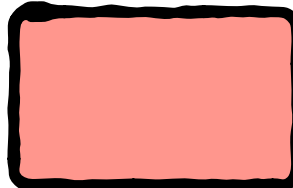
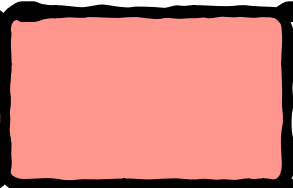
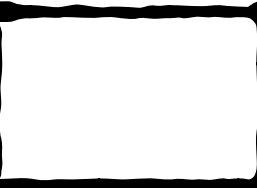
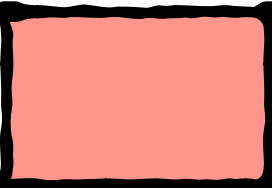
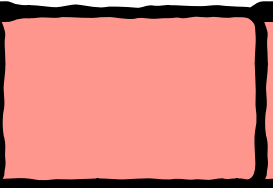
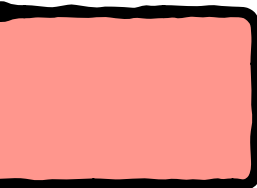
After transaction is over, page won't be evicted despite being cold



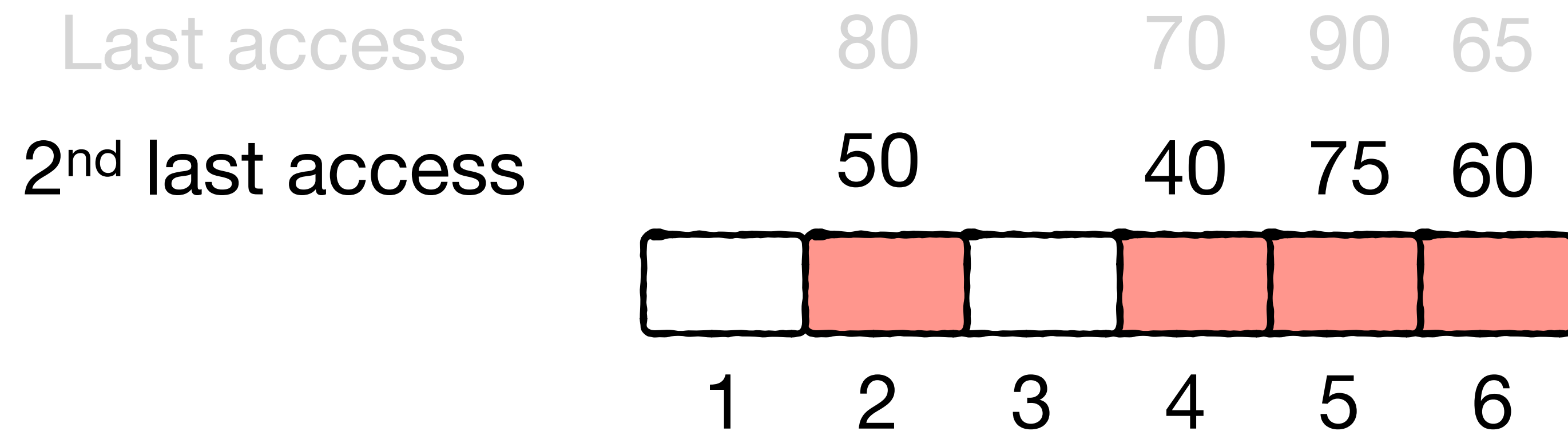
Concern 1: cold entry may get access spurt (e.g. by same transaction)

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

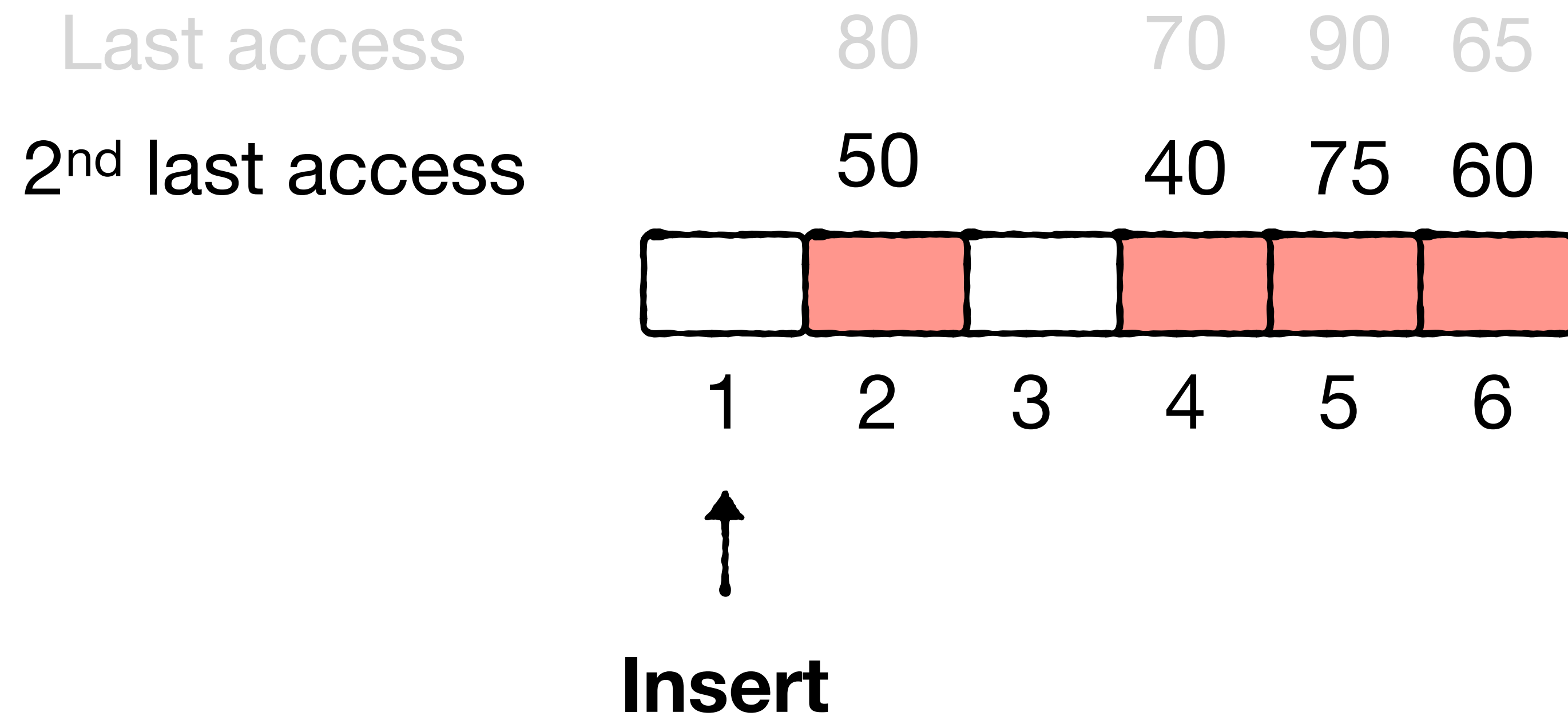
Solution?

Last access	92	80		70	90	65
2 nd last access	91	50		40	75	60
						
	1	2	3	4	5	6

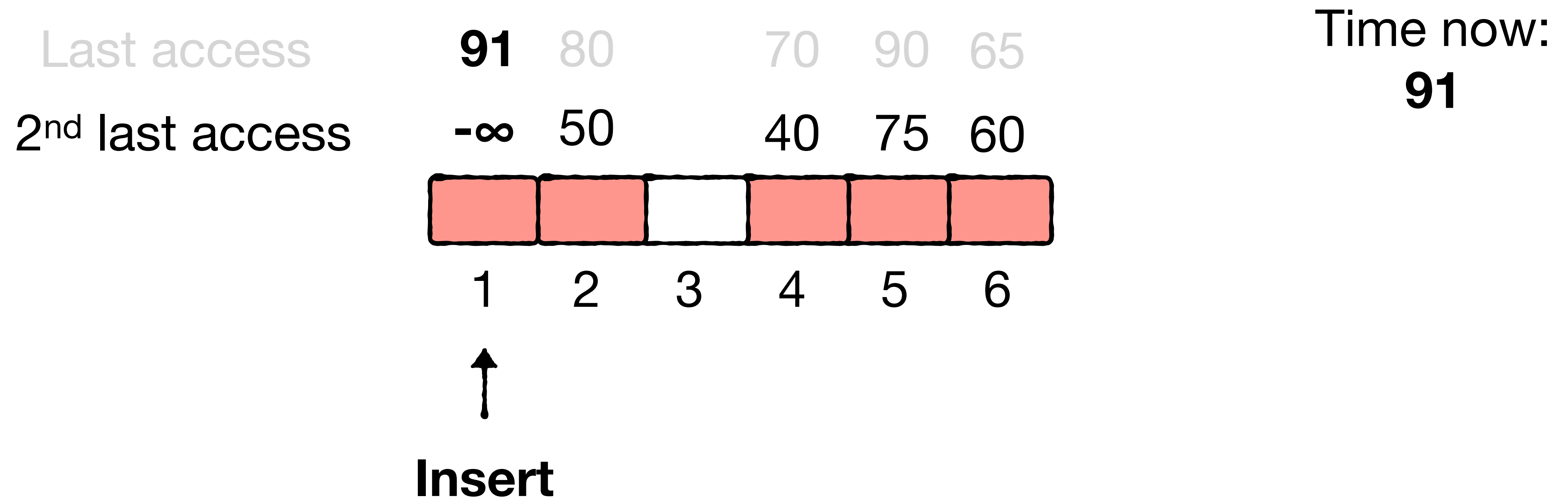
Solution: do not update entry's access time if new access is within given window of last access



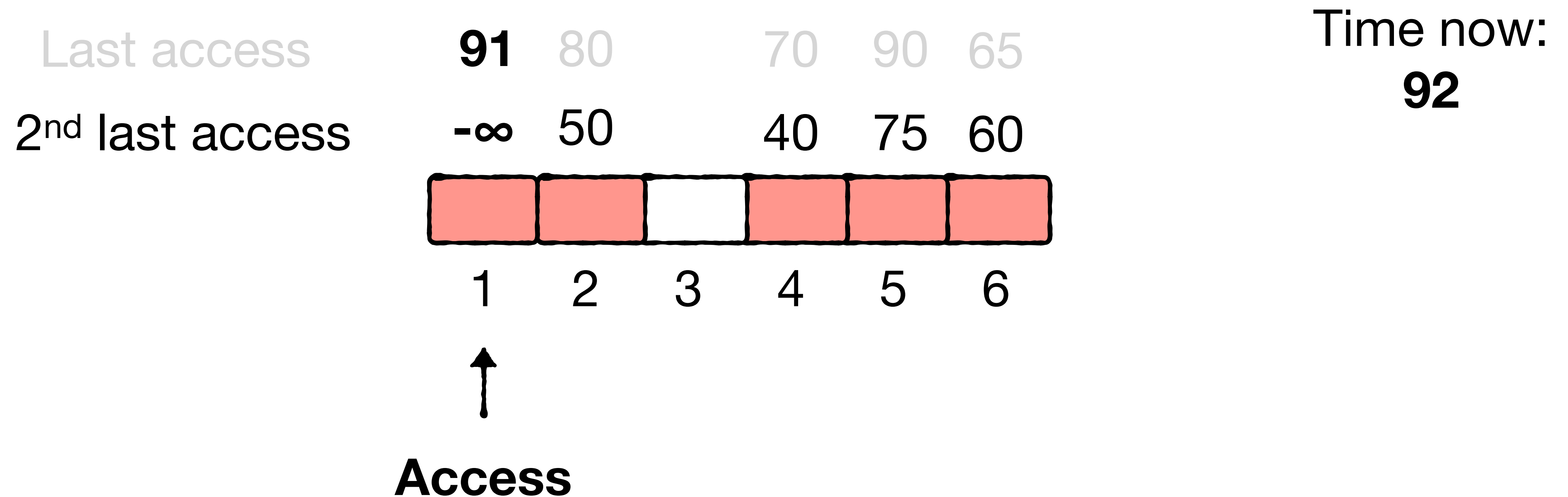
Solution: do not update entry's access time if new access is within given window of last access



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



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

Last access	91	80		70	90	65
2 nd last access	$-\infty$	50		40	75	60
	1	2	3	4	5	6



Access

Time now:
92

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

Window size: 5

Last access	91	80		70	90	65
2 nd last access	$-\infty$	50		40	75	60
	1	2	3	4	5	6



Access

Time now:
93

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

Window size: 5

Last access	91	80		70	90	65
2 nd last access	$-\infty$	50		40	75	60
	1	2	3	4	5	6

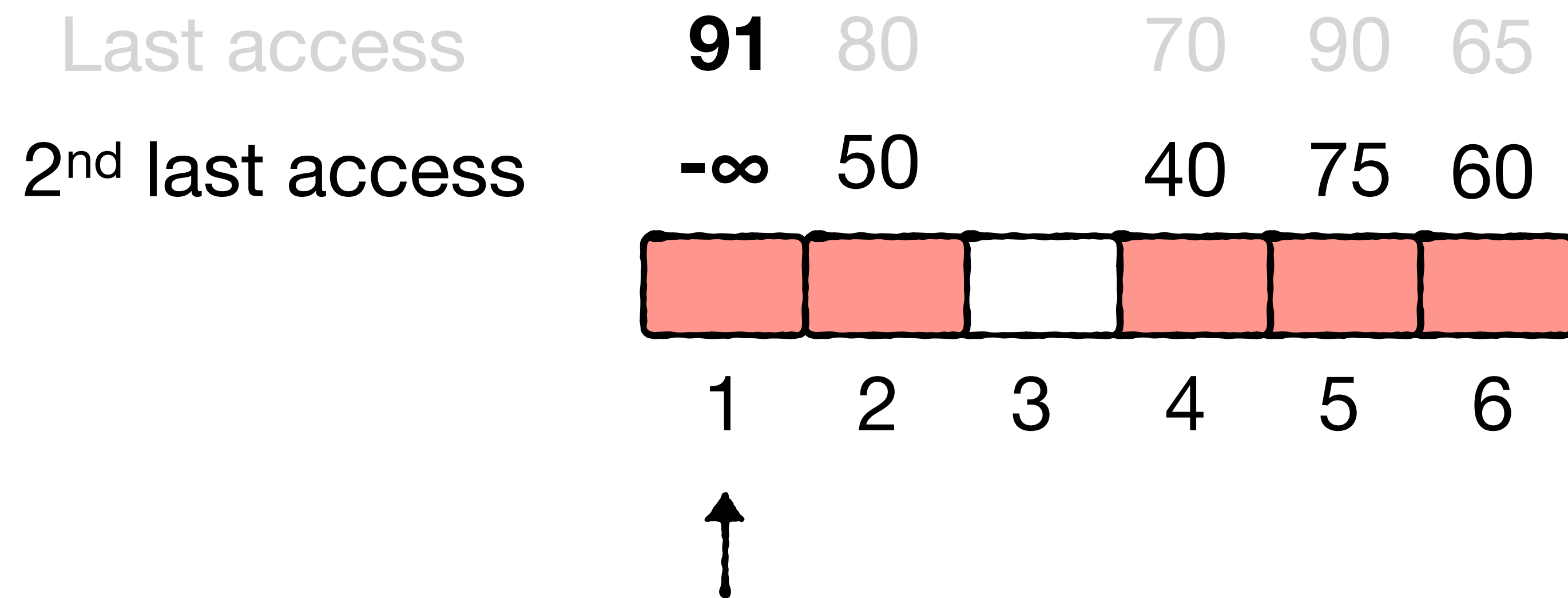


Access

Time now:
94

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

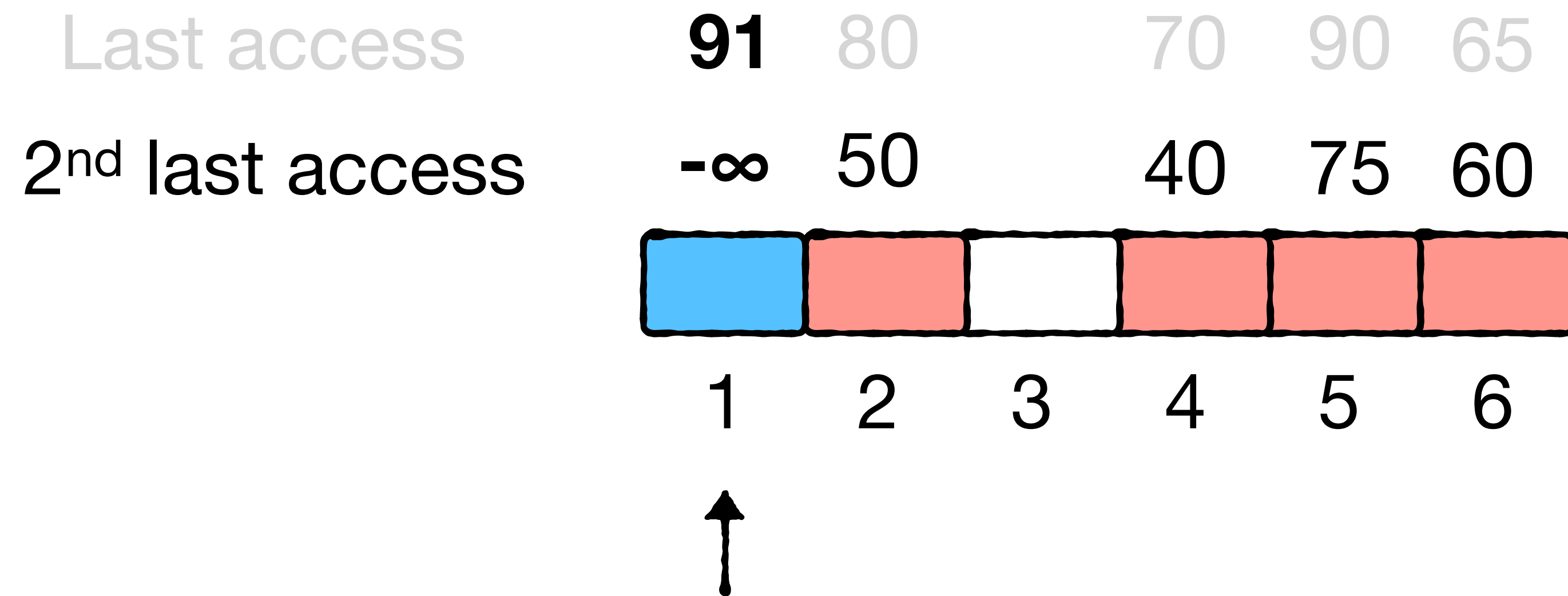
Window size: 5



Time now:
95

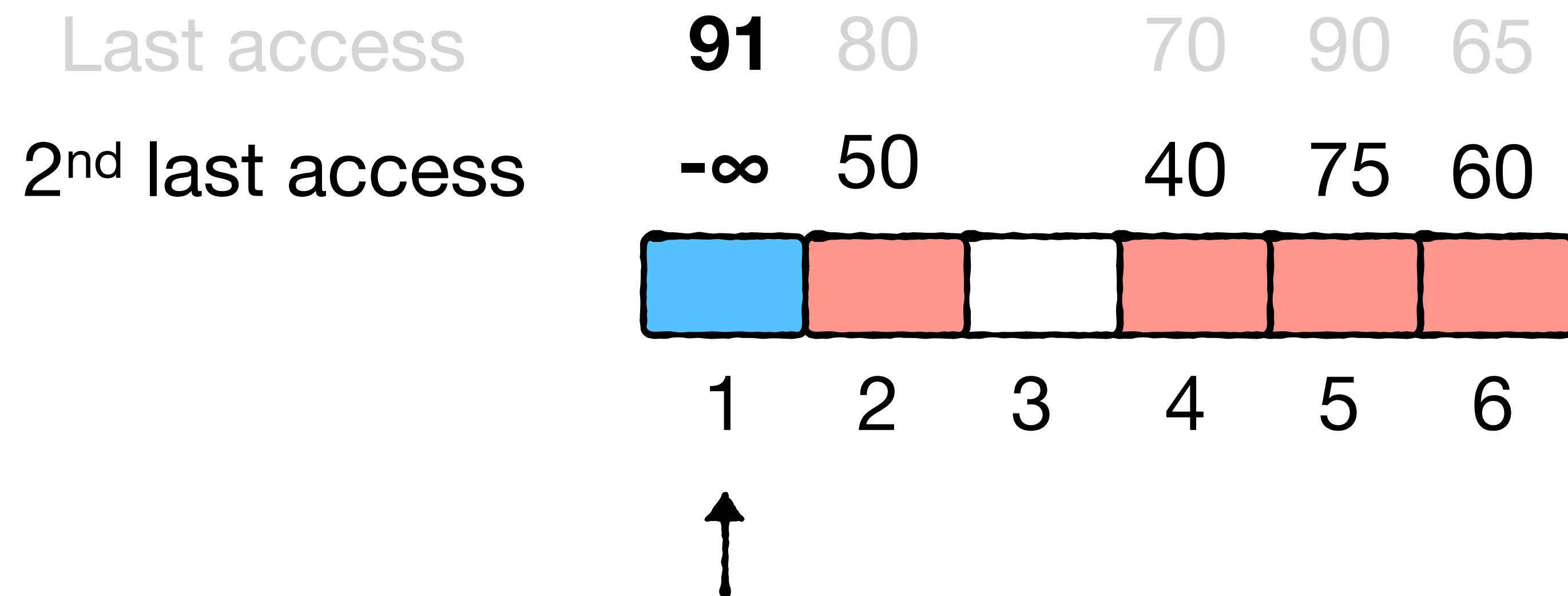
Solution: do not update entry's access time if new access is within given window of last access

Window size: 5



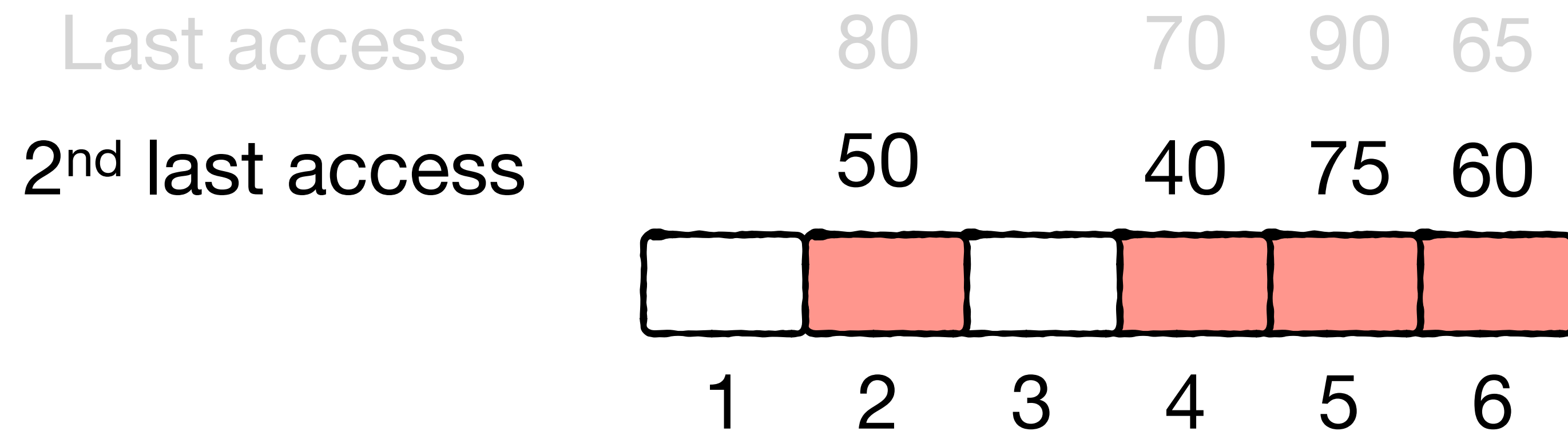
Solution: do not update entry's access time if new access is within given window of last access

Window size: 5

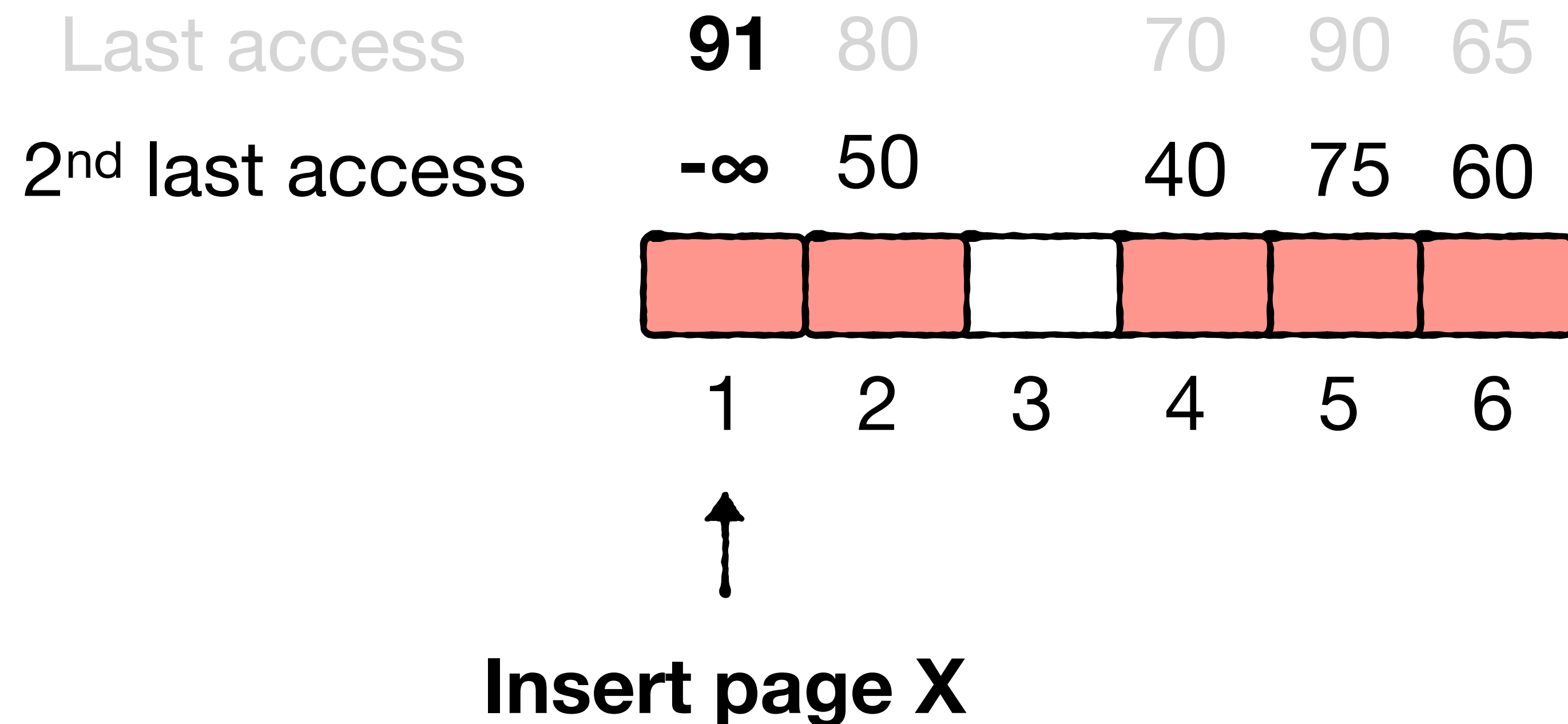


Page returns to being cold, and it is now prioritized for eviction

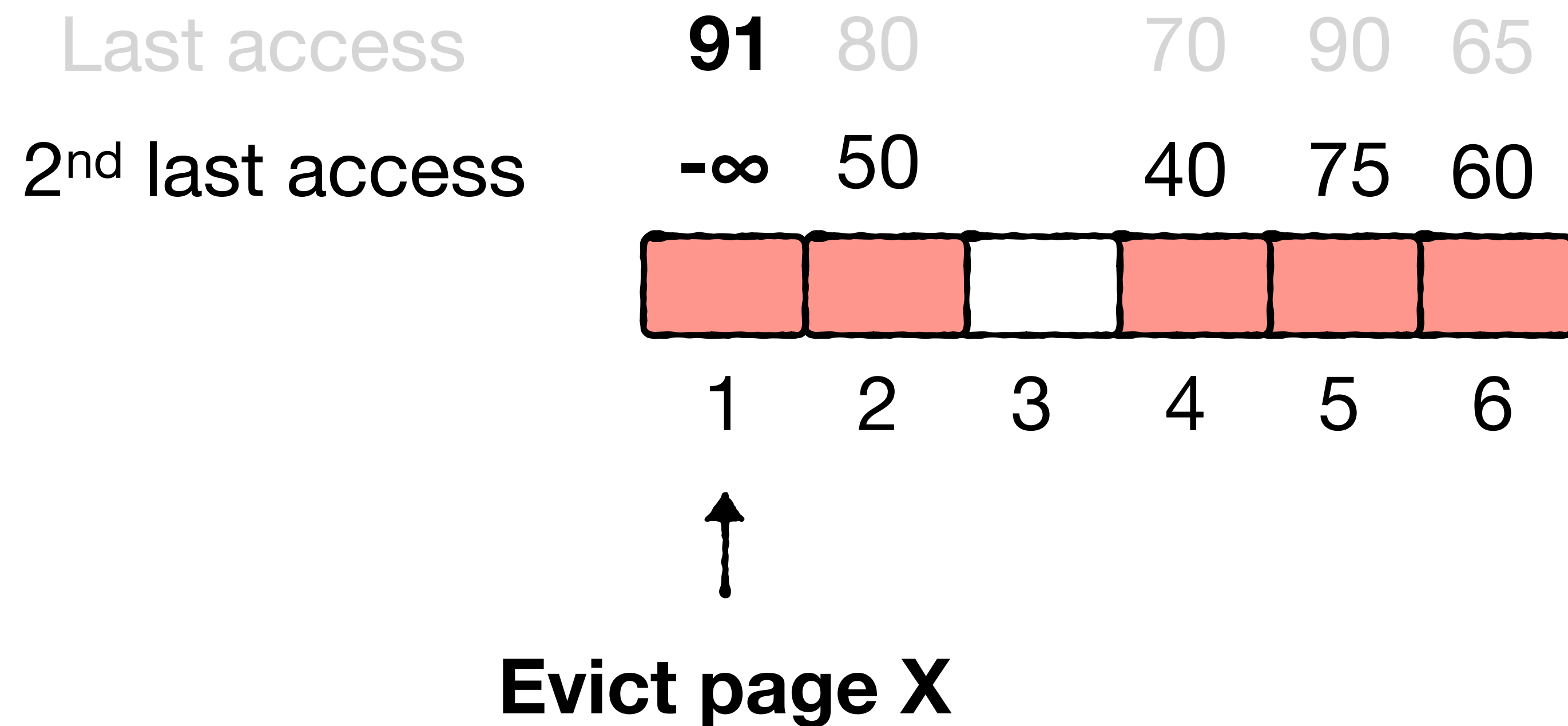
Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



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

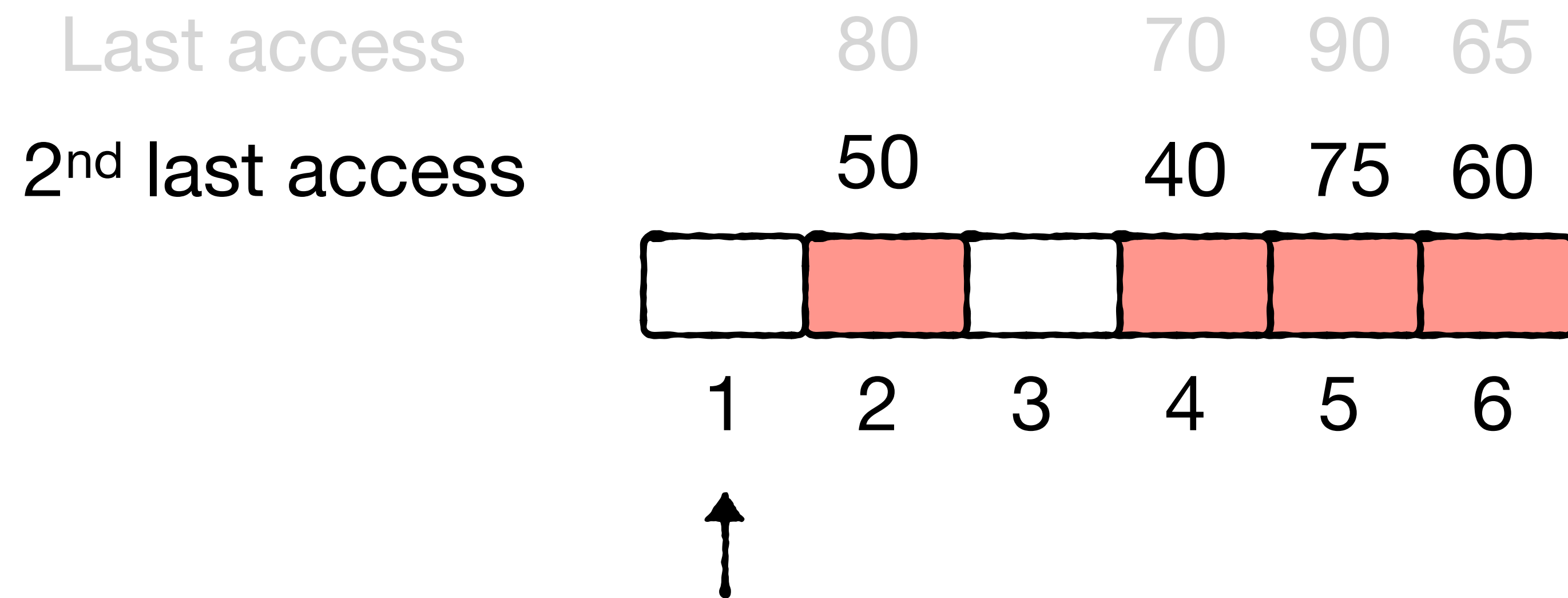


Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



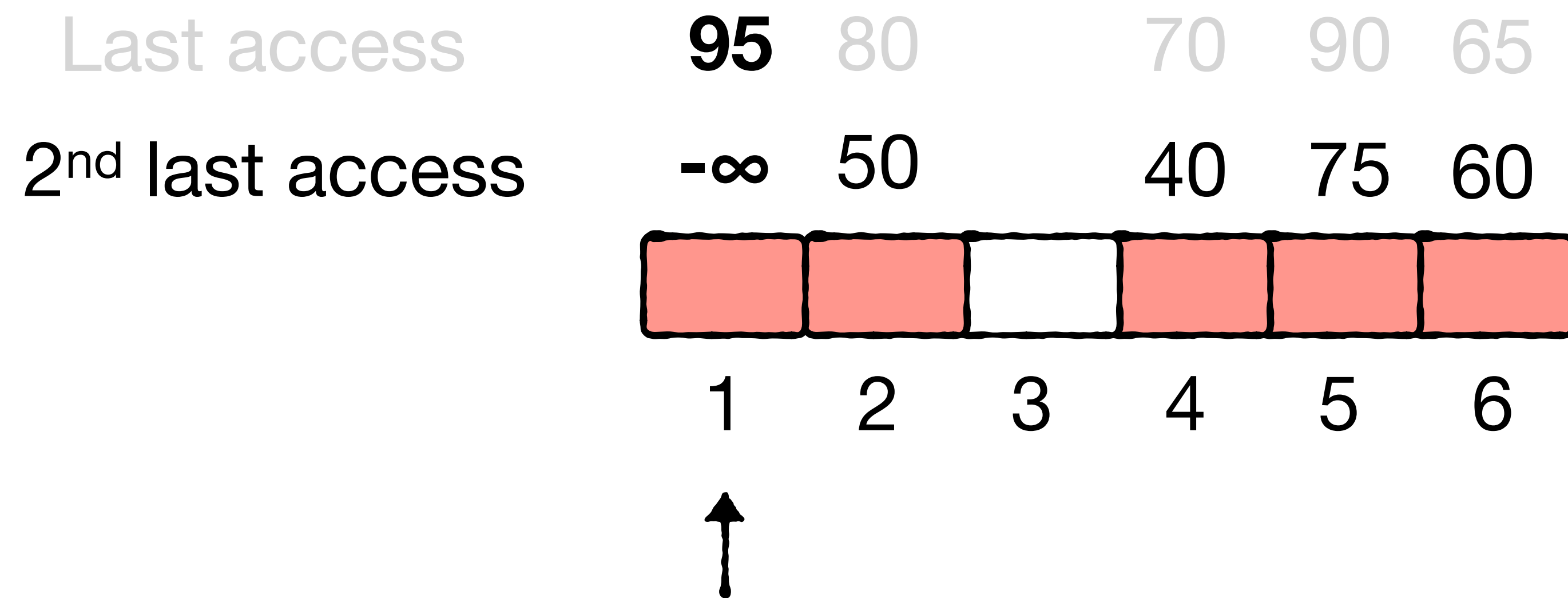
Evict page X

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



Shortly after, Insert page X again

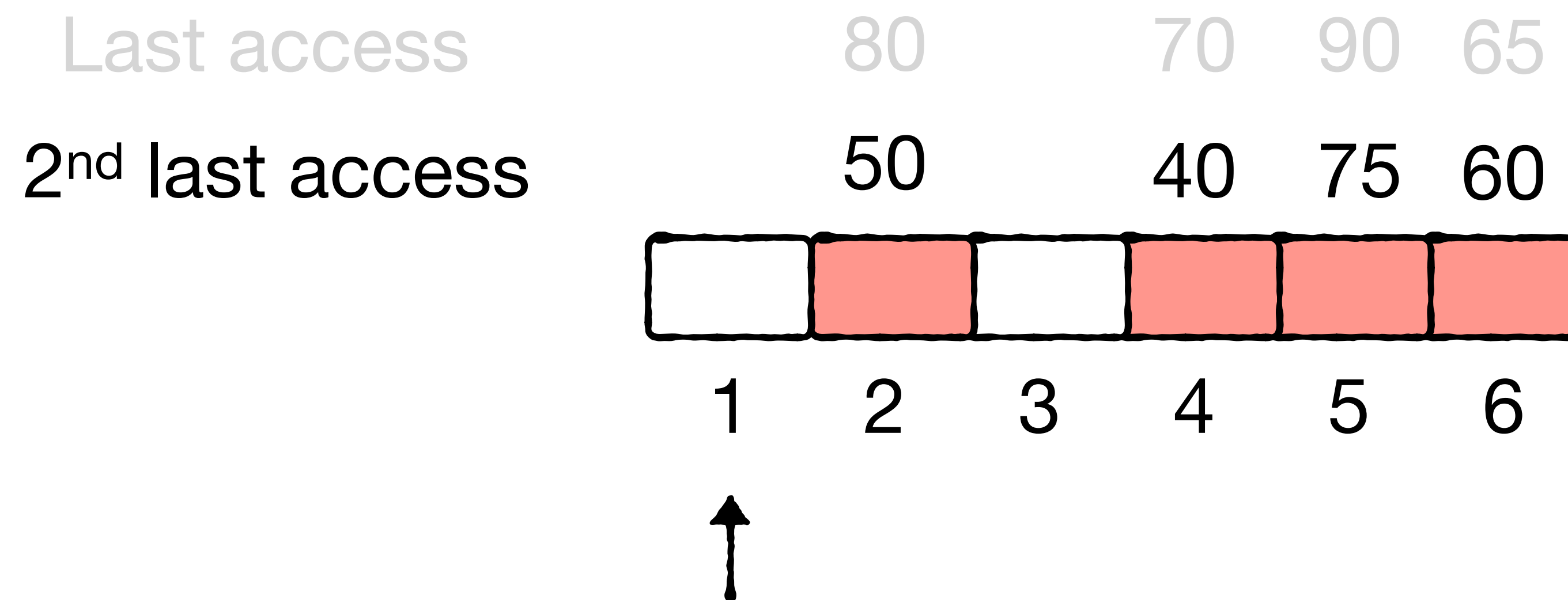
Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent



Shortly after, Insert page X again

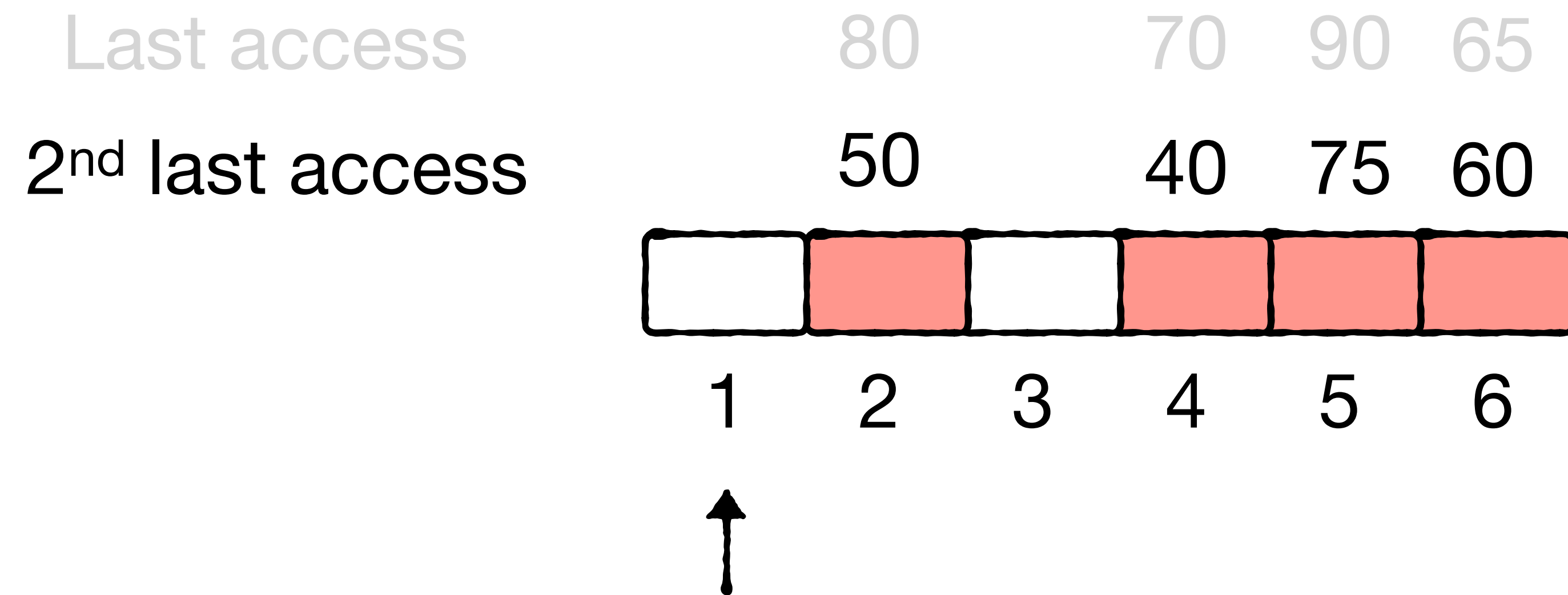
Concern 2: Newly inserted pages are prioritized for eviction, even when subsequent access may be imminent

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 $\rightarrow$ Last K access times

Last access

80

70

90

65

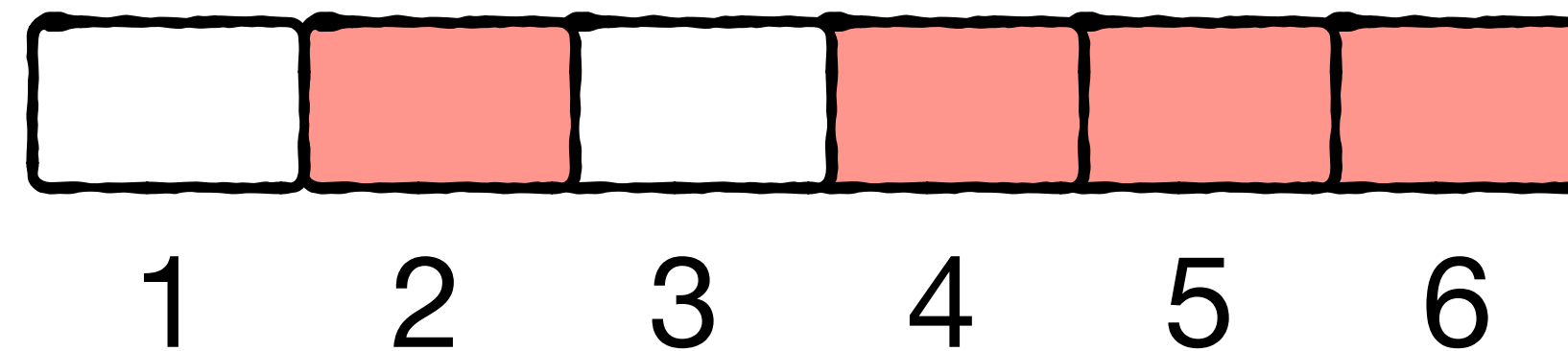
2nd last access

50

40

75

60



Solutions: Keep access times for evicted pages in memory

Access HT

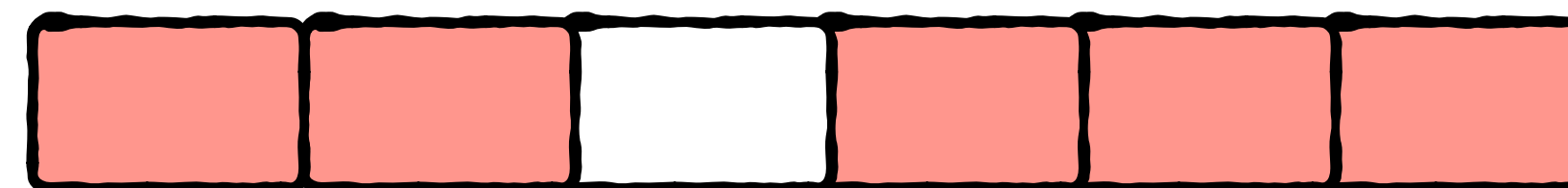
Page ID $\rightarrow$ Last K access times

Last access

95 80 70 90 65

2nd last access

$-\infty$ 50 40 75 60



1 2 3 4 5 6



Insert page X

Solutions: Keep access times for evicted pages in memory

Access HT

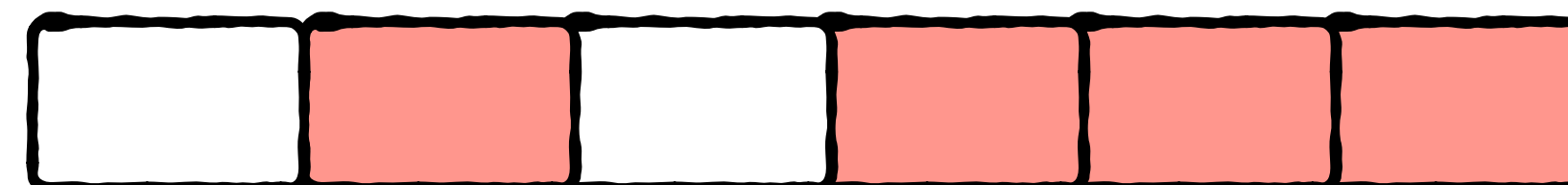
X → **95, $-\infty$**

Last access

80 70 90 65

2nd last access

50 40 75 60



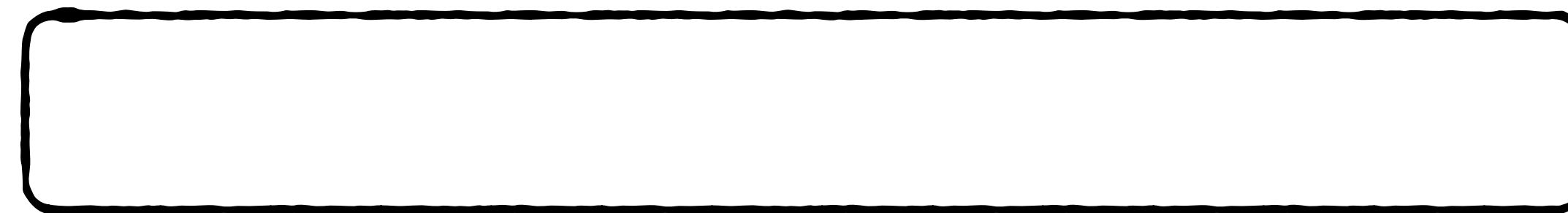
1 2 3 4 5 6



Evict page X

Solutions: Keep access times for evicted pages in memory

Access HT

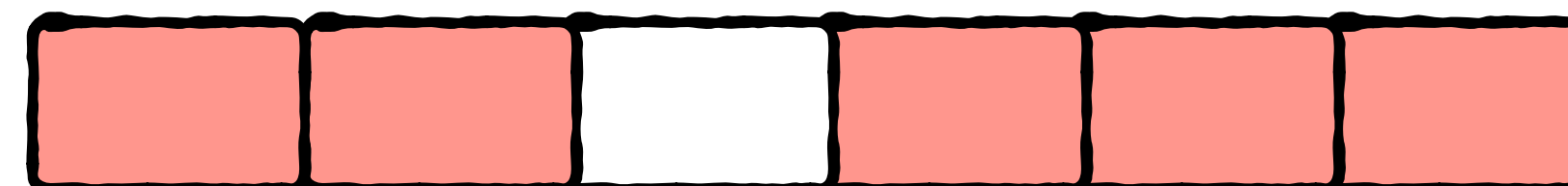


Last access

100 80 70 90 65

2nd last access

95 50 40 75 60

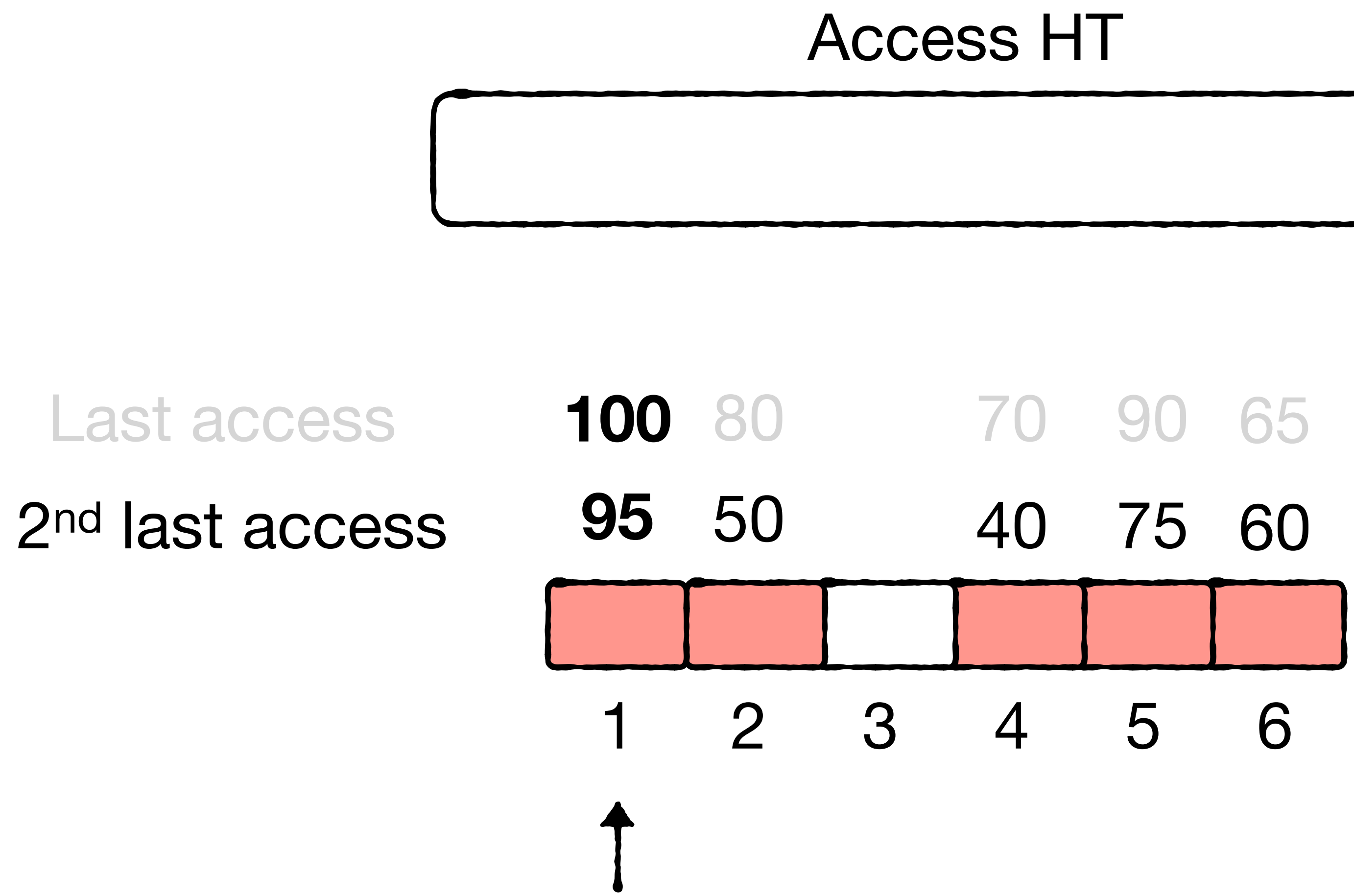


1 2 3 4 5 6



Shortly after, Insert page X again (time 100)

Solutions: Keep access times for evicted pages in memory

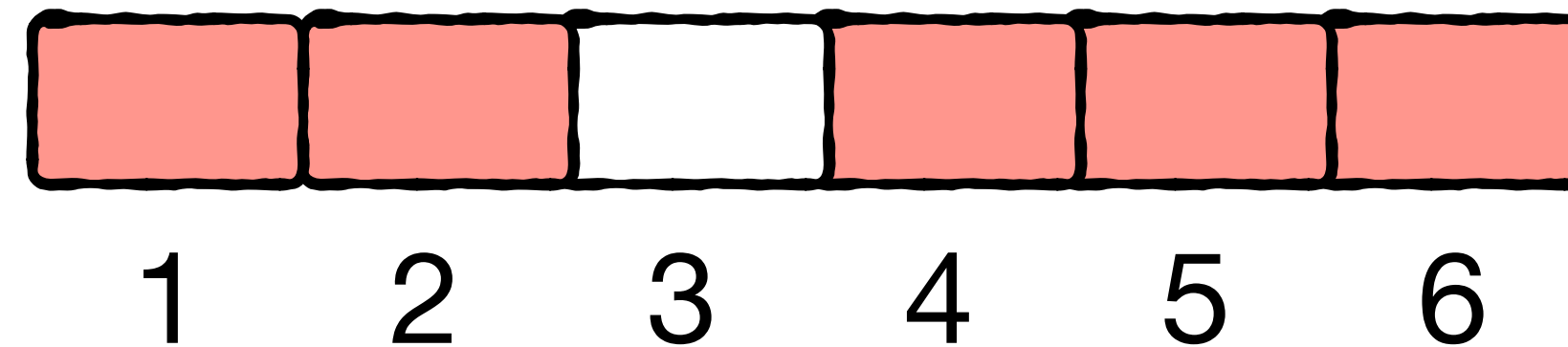


How to implement this?



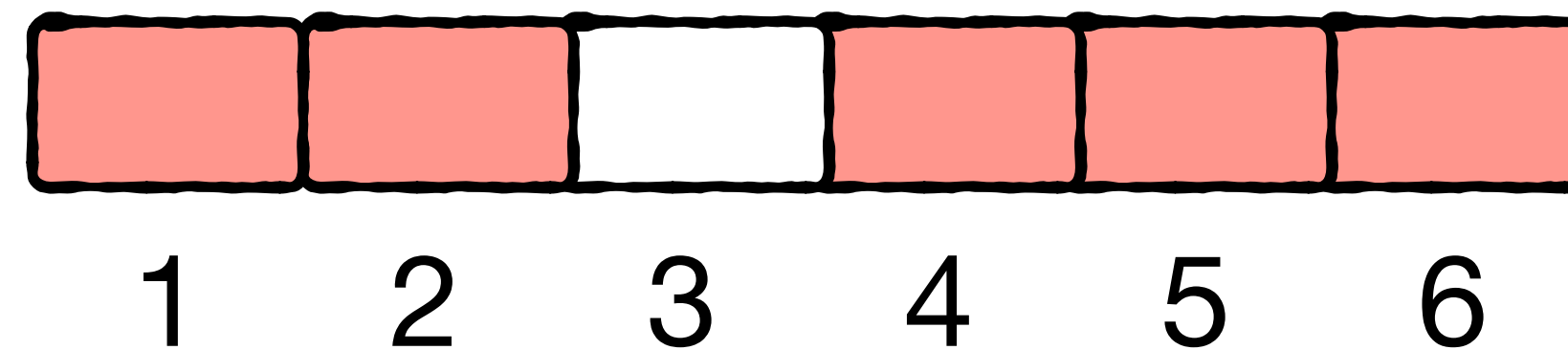
Access HT

Page ID → Last 2 access times



Access HT + FIFO

Page ID → Last 2 access times



Picture so far:

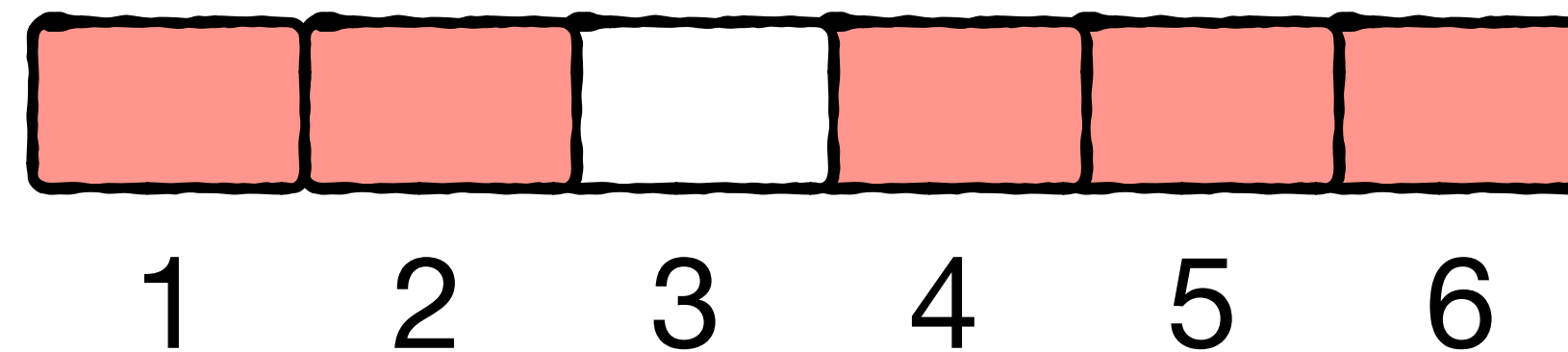
Access HT + FIFO

Page ID $\rightarrow$ Last 2 access times

Min-Heap

LRU/2

Buffer pool

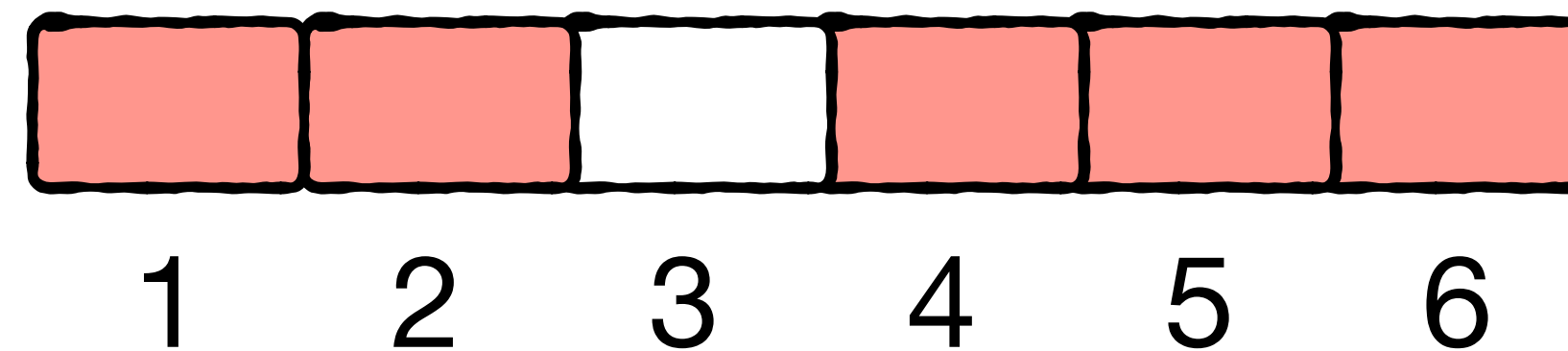


Can we improve on the $O(\log N)$ cost of updating min-heap?

Min-Heap



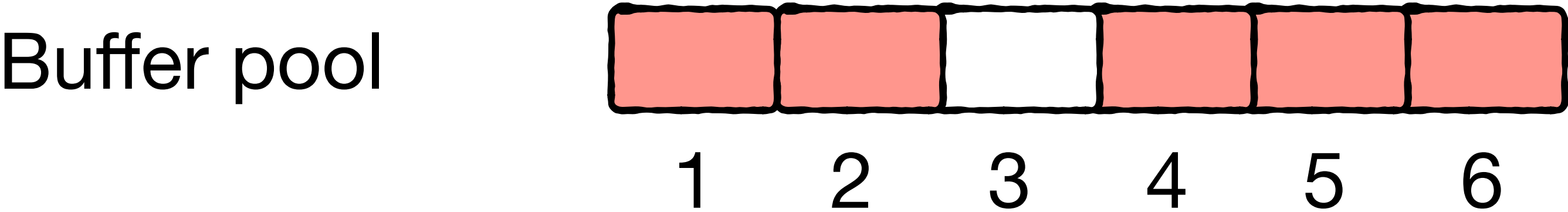
Buffer pool



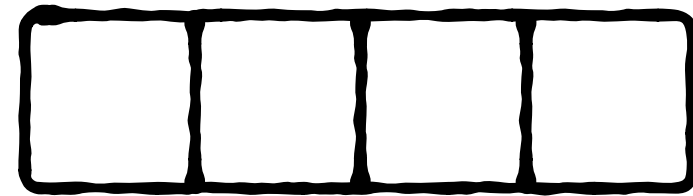
2Q: a low overhead high performance buffer management replacement algorithm

Theodore Johnson, Dennis Shasha

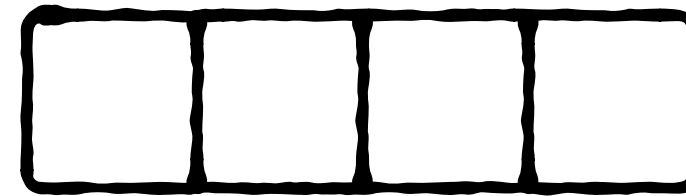
VLBD 1994



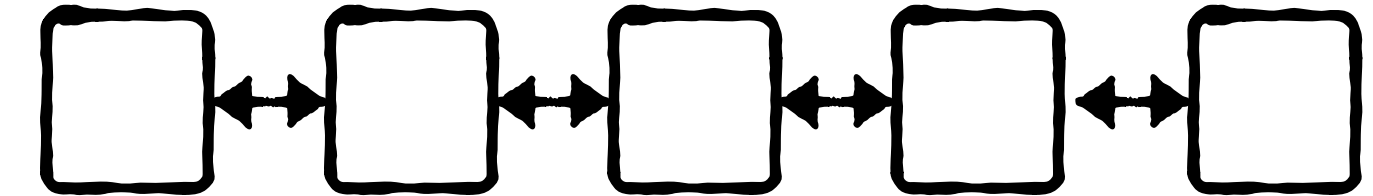
**Recently evicted
(HT + FIFO)**



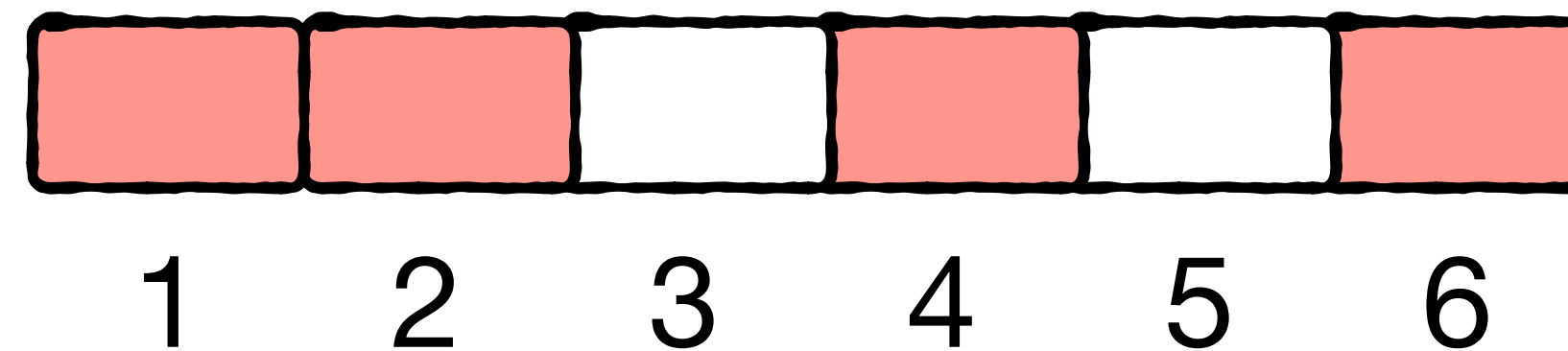
In queue (FIFO)



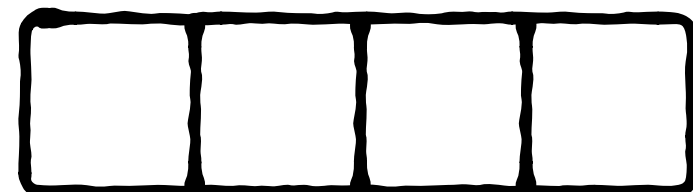
Main queue (LRU)



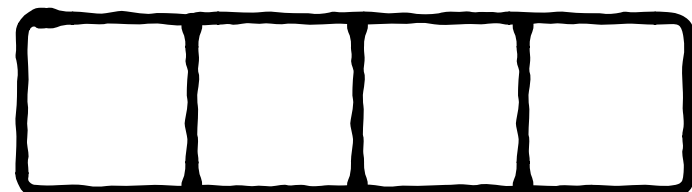
Buffer pool



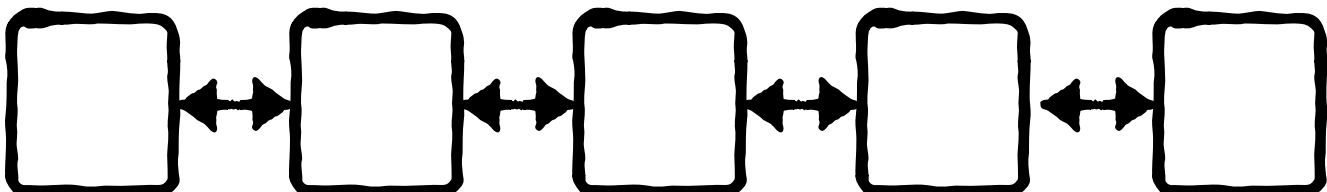
Recently evicted
(HT + FIFO)



In queue (FIFO)

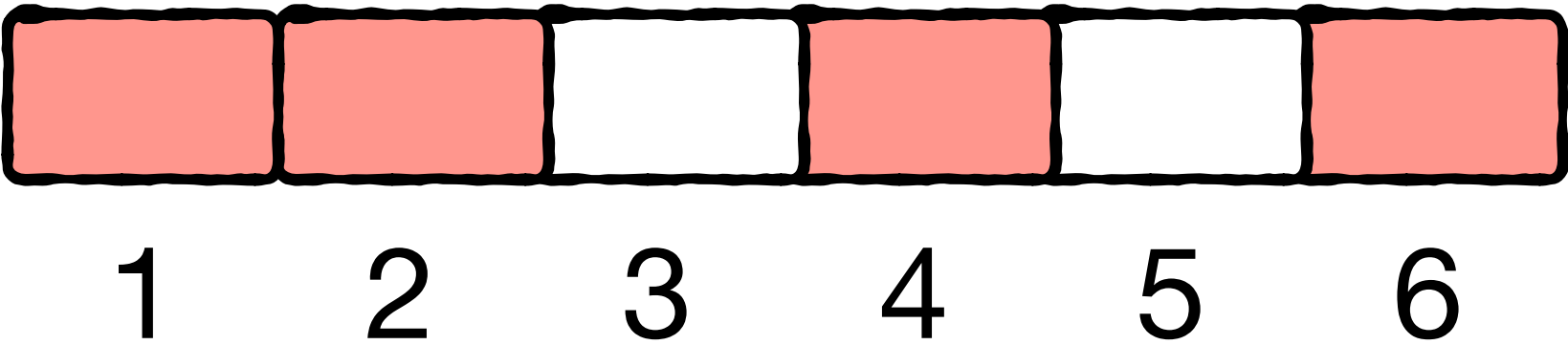


Main queue (LRU)

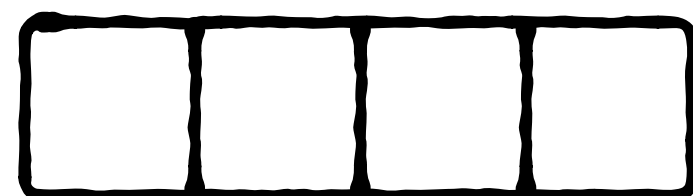


newly read page IDs

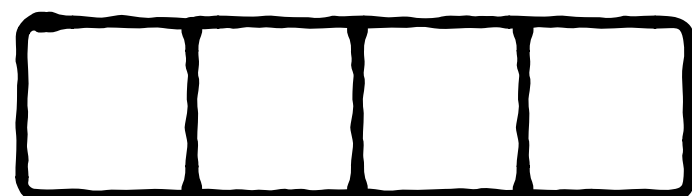
Buffer pool



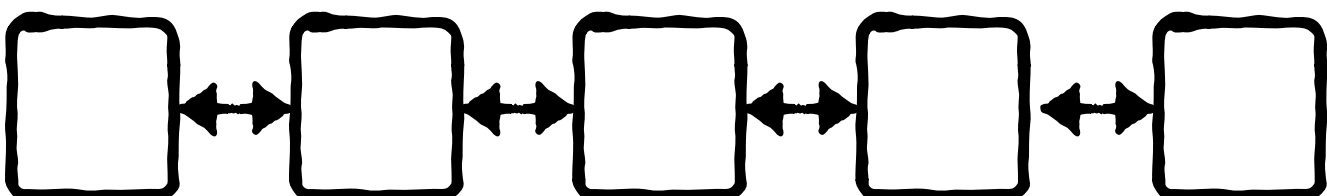
Recently evicted
(HT + FIFO)



In queue (FIFO)

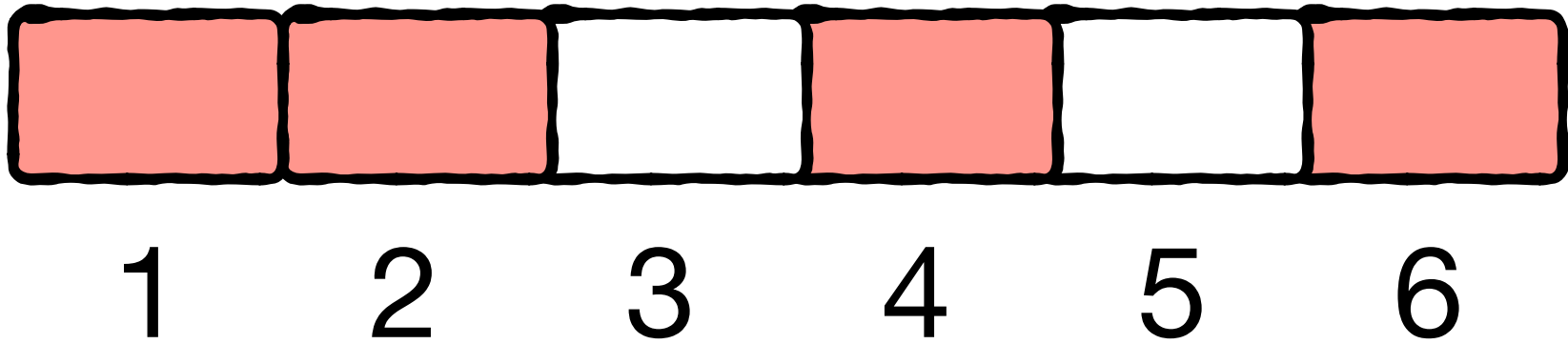


Main queue (LRU)

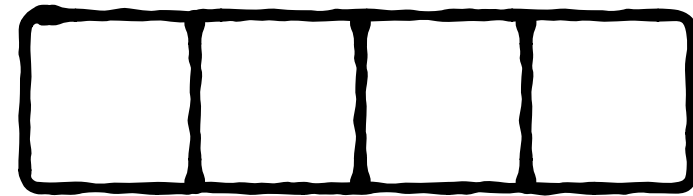


hot page IDs

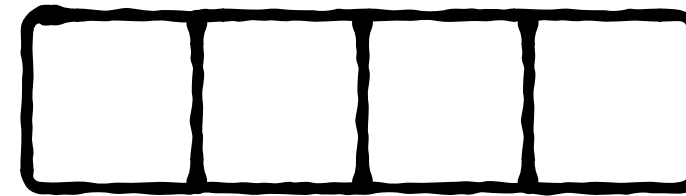
Buffer pool



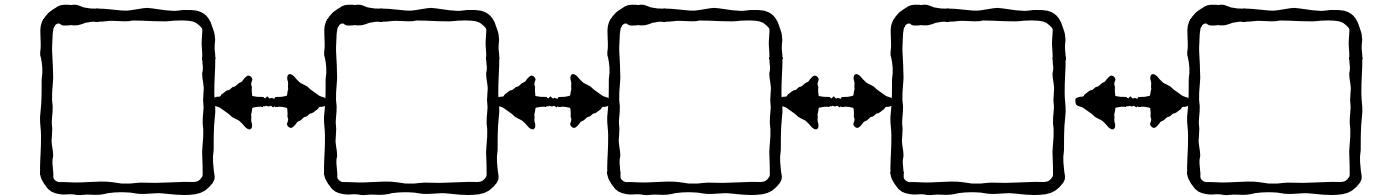
Recently evicted
(HT + FIFO)



In queue (FIFO)

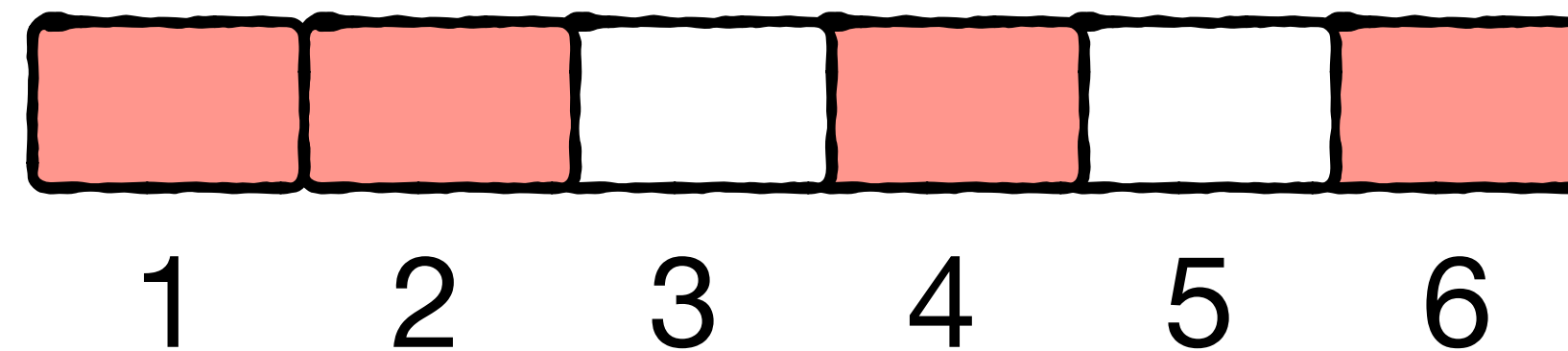


Main queue (LRU)

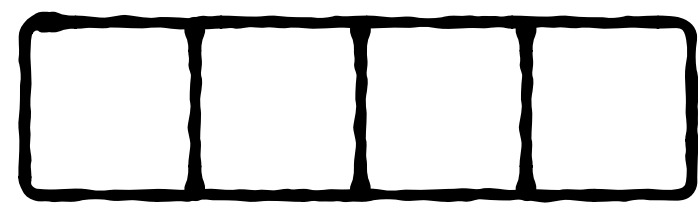


Recently evicted page IDs

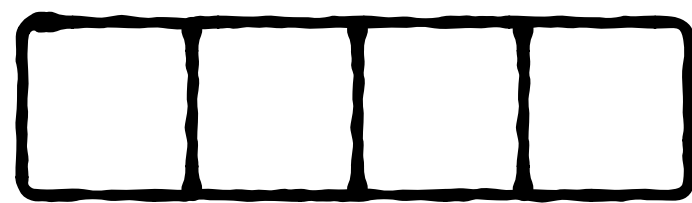
Buffer pool



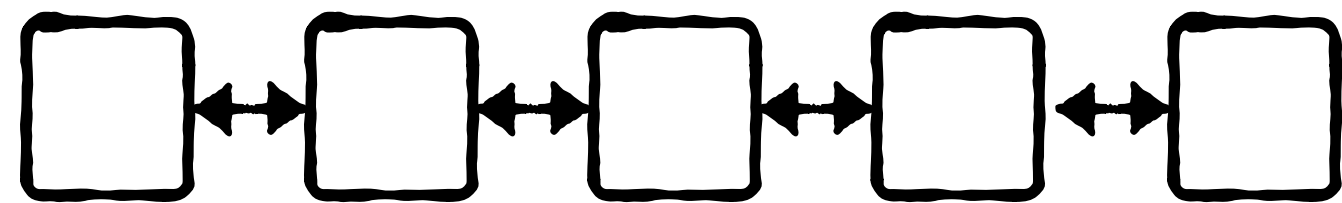
Recently evicted
(HT + FIFO)



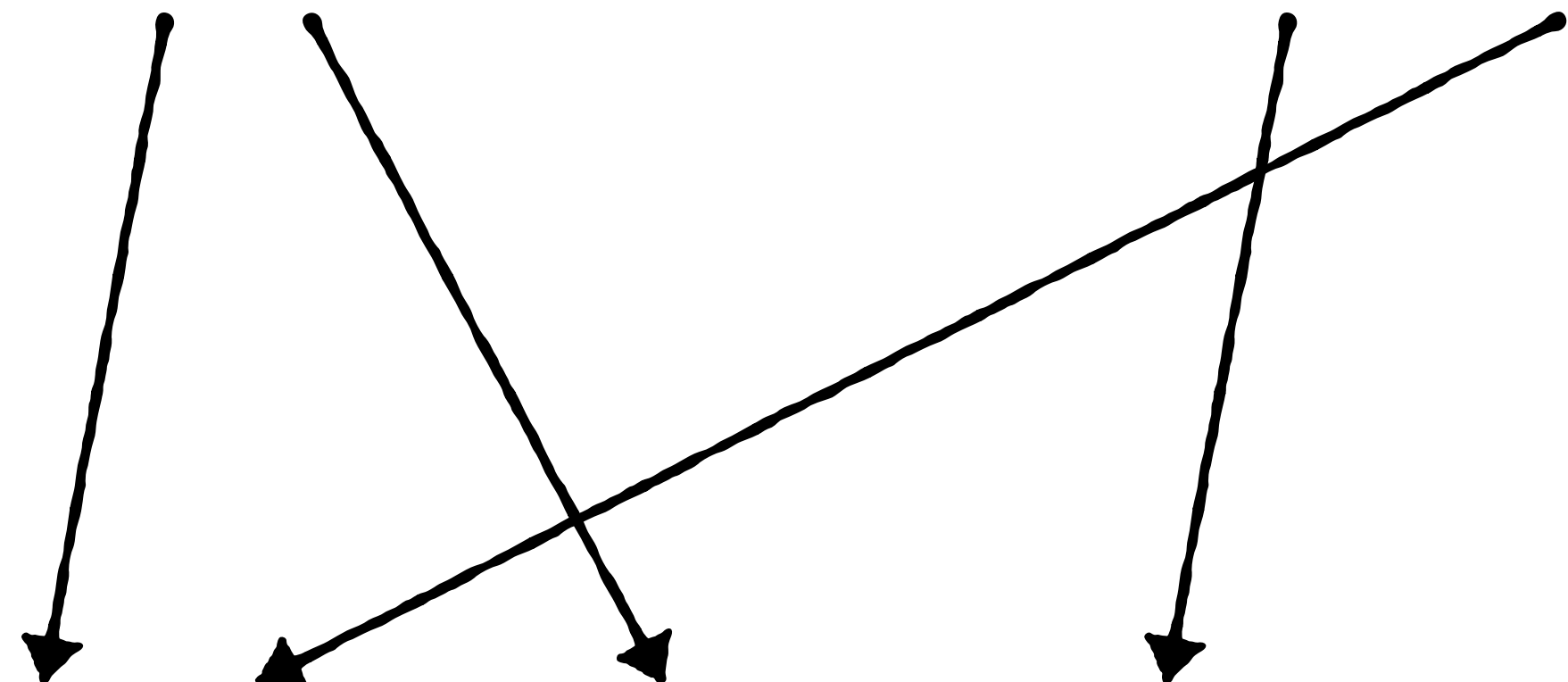
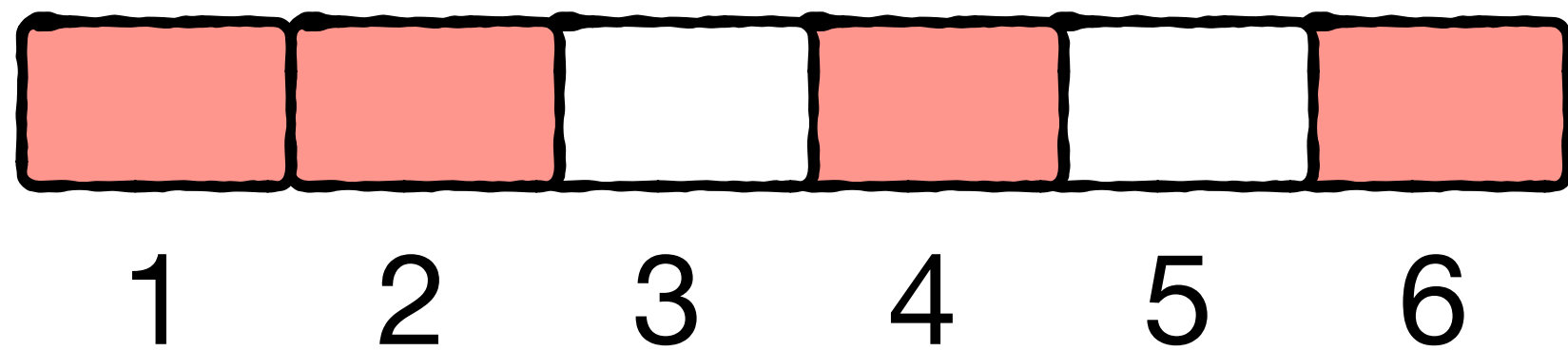
In queue (FIFO)



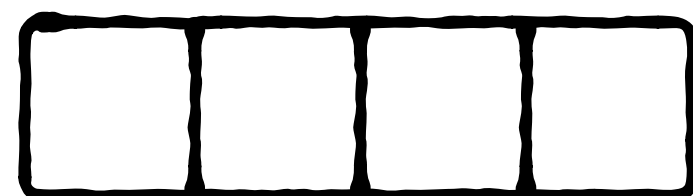
Main queue (LRU)



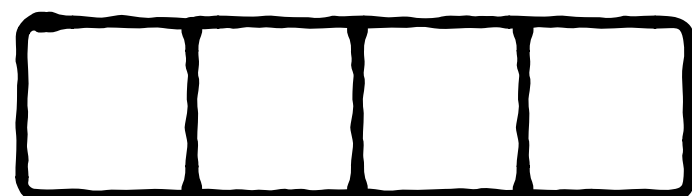
Buffer pool



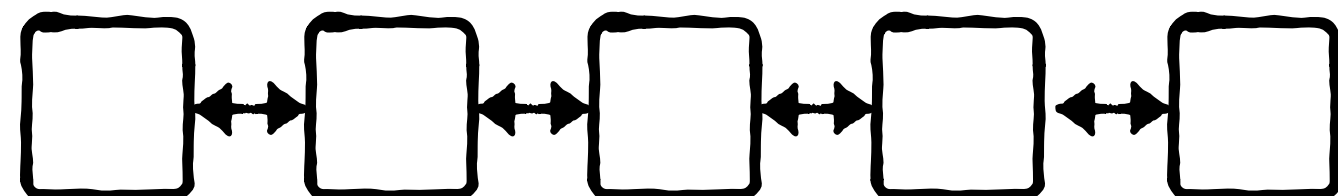
Recently evicted
(HT + FIFO)



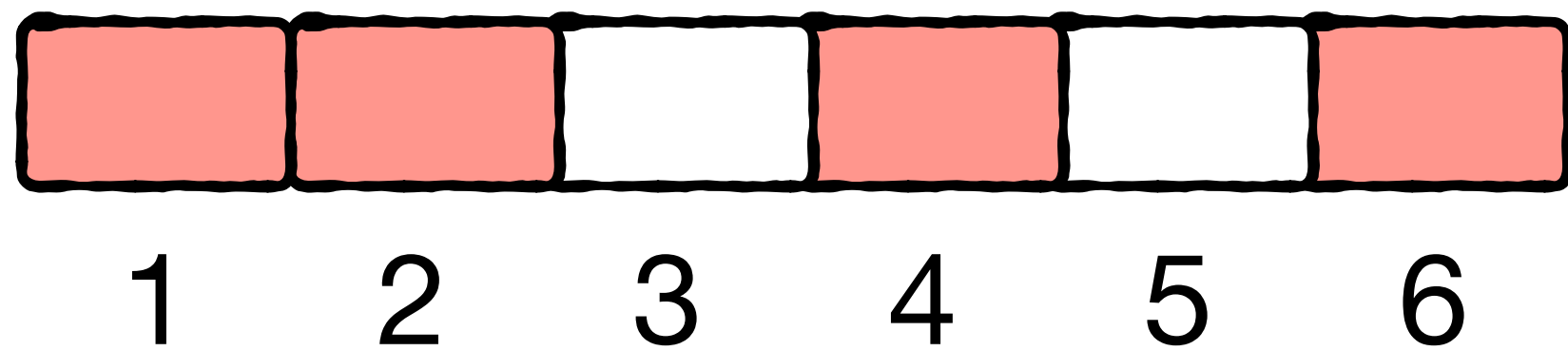
In queue (FIFO)



Main queue (LRU)

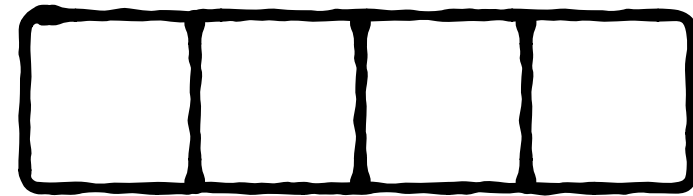


Buffer pool

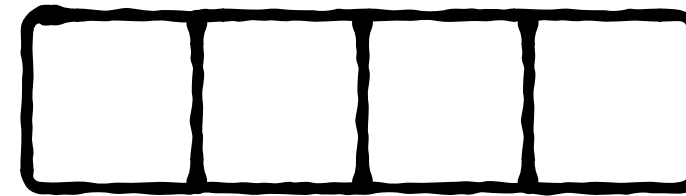


Insert page X

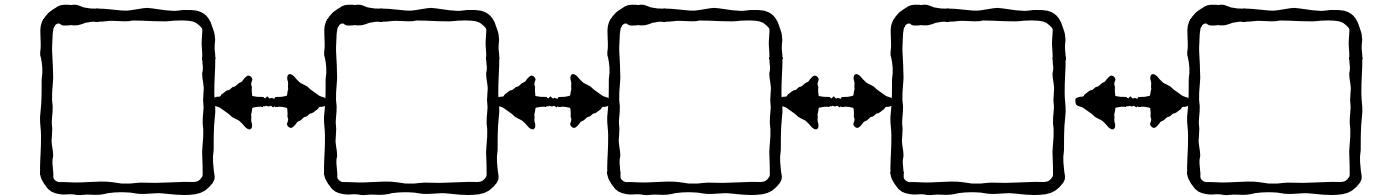
**Recently evicted
(HT + FIFO)**



In queue (FIFO)

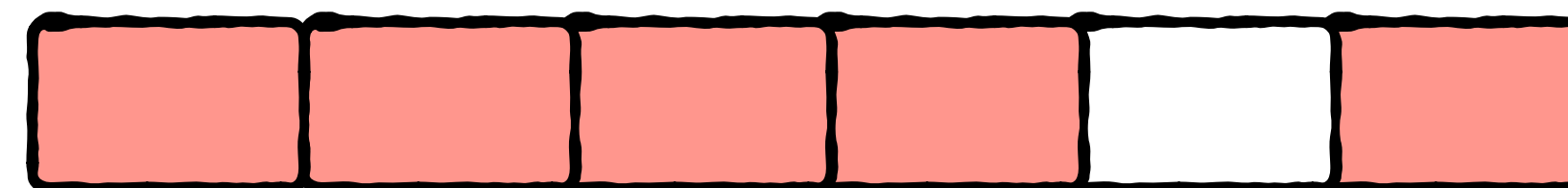


Main queue (LRU)



**Check for X,
doesn't exist**

Buffer pool

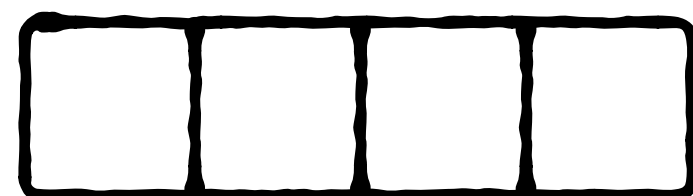


1 2 3 4 5 6

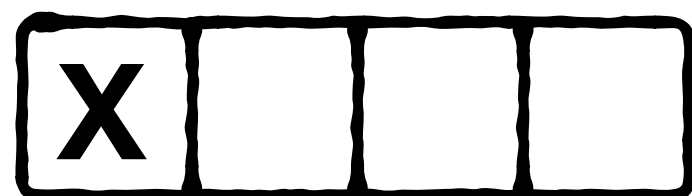


Insert page X

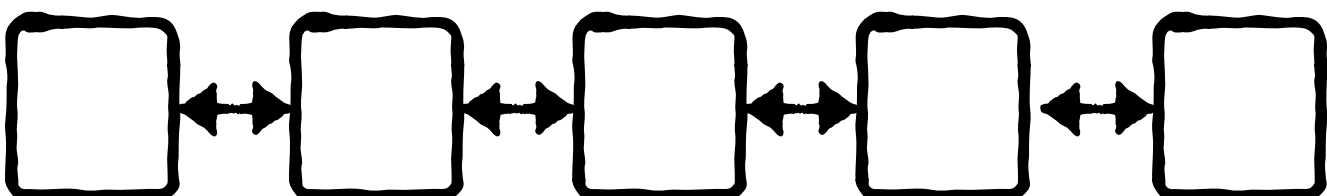
Recently evicted
(HT + FIFO)



In queue (FIFO)



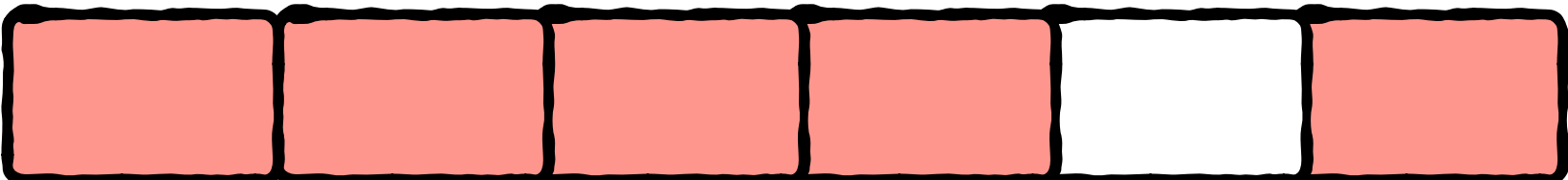
Main queue (LRU)



Insert X



Buffer pool

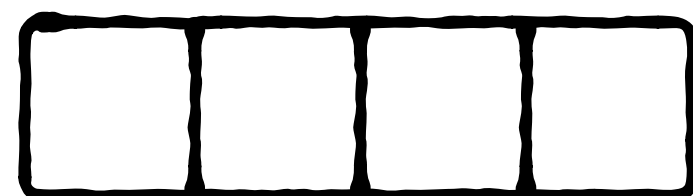


1 2 3 4 5 6

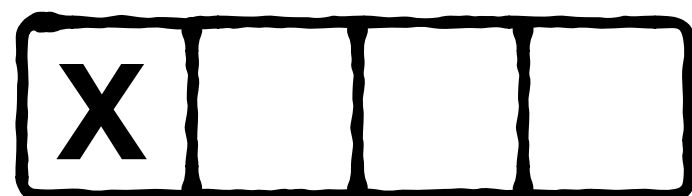


Insert page X

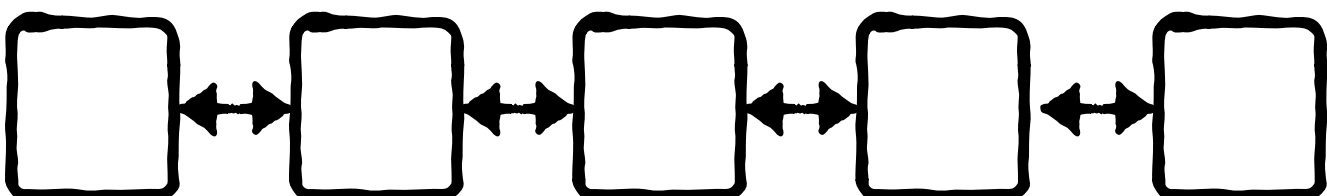
Recently evicted
(HT + FIFO)



In queue (FIFO)

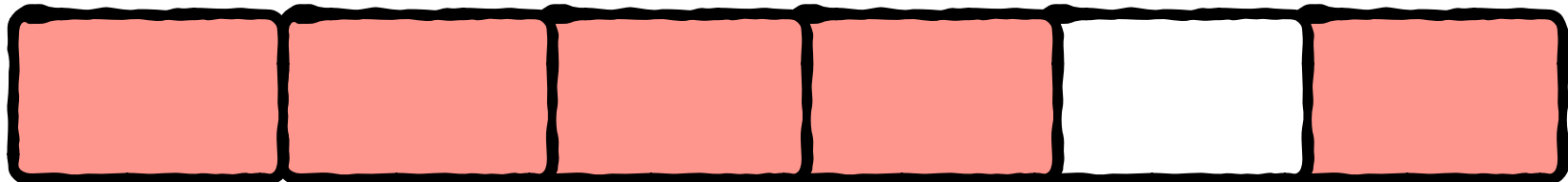


Main queue (LRU)



Do not promote (could
be a temporary spurt)

Buffer pool

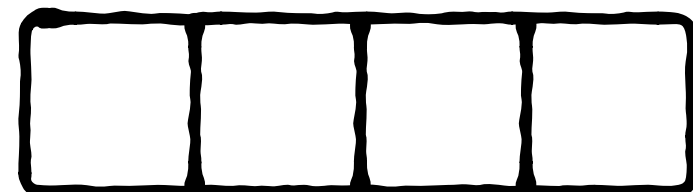


1 2 3 4 5 6

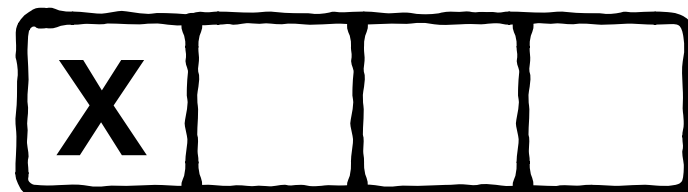


Access

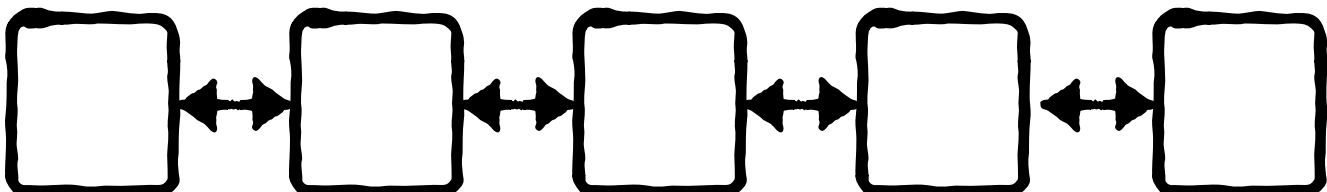
Recently evicted
(HT + FIFO)



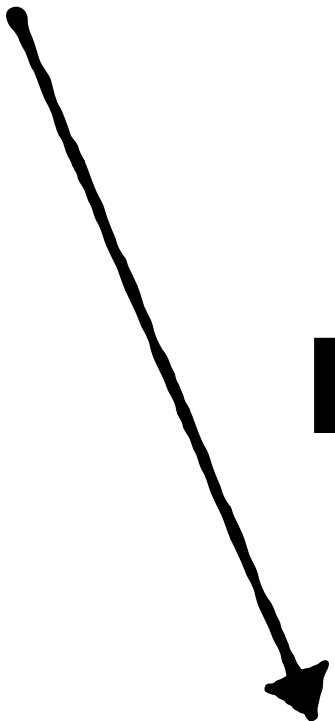
In queue (FIFO)



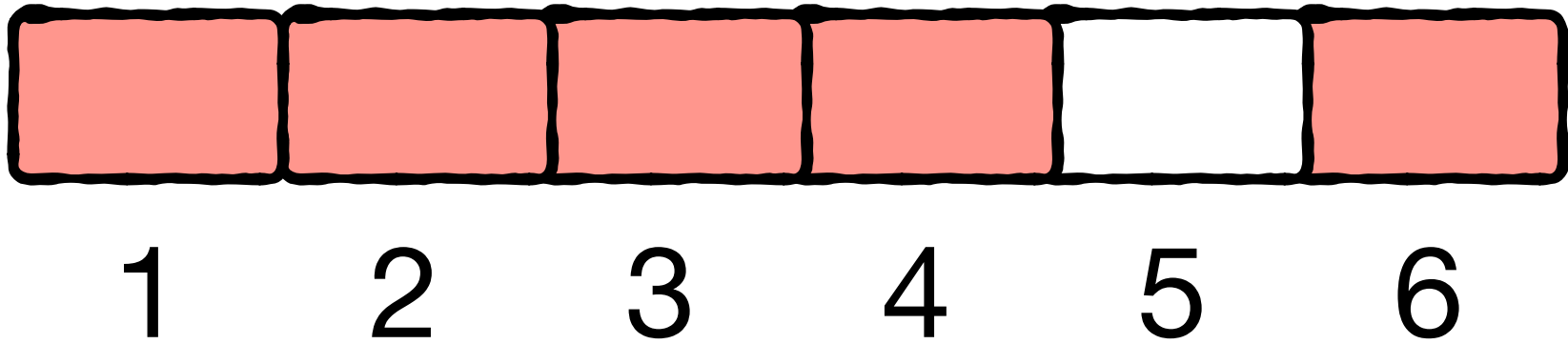
Main queue (LRU)



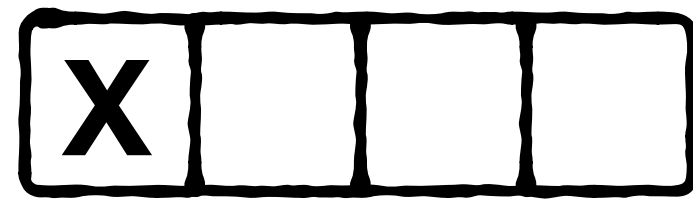
Evict eventually



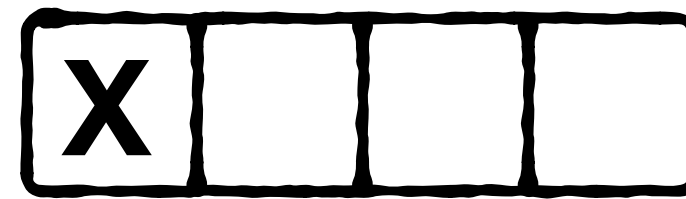
Buffer pool



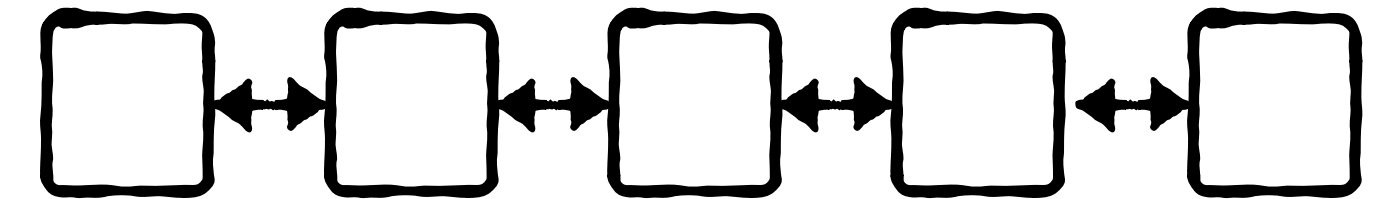
Recently evicted
(HT + FIFO)



In queue (FIFO)

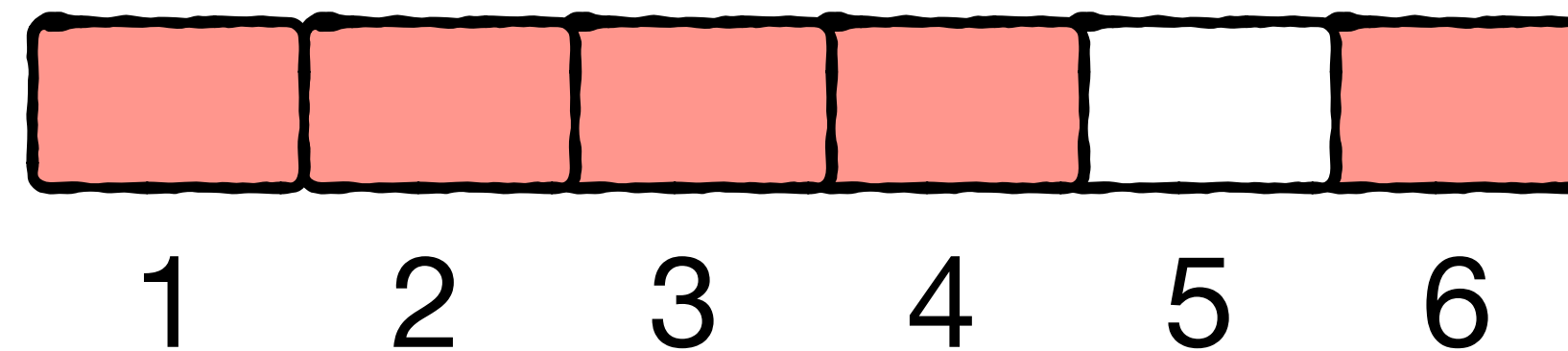


Main queue (LRU)

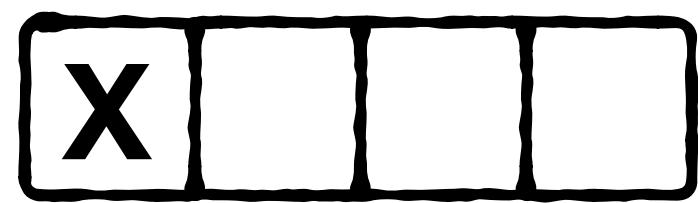


Evict eventually

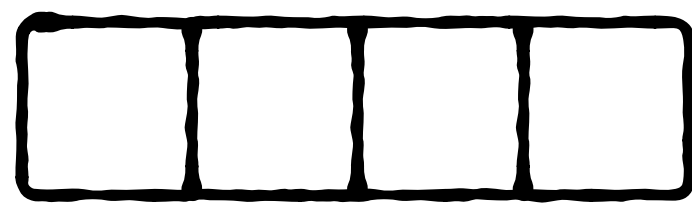
Buffer pool



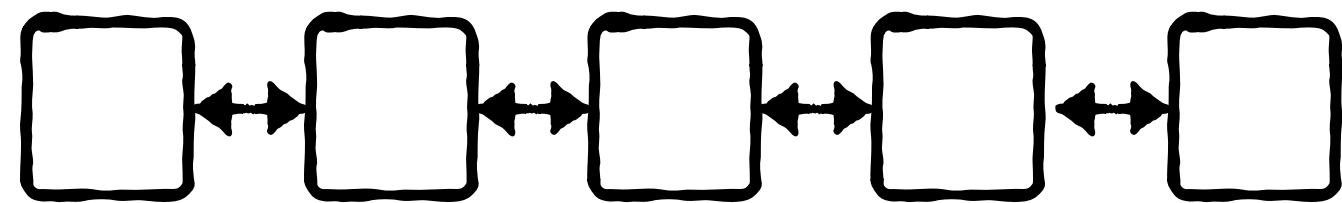
Recently evicted
(HT + FIFO)



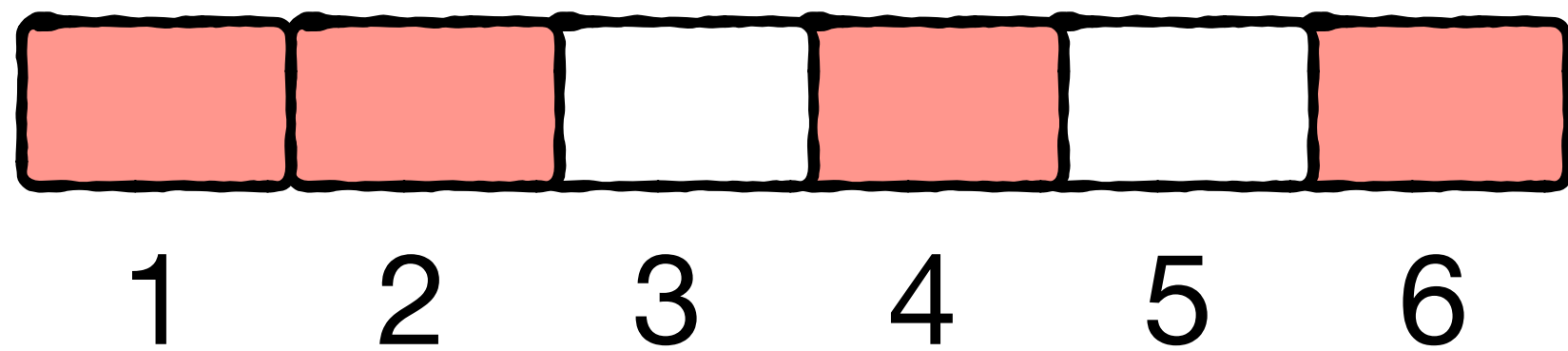
In queue (FIFO)



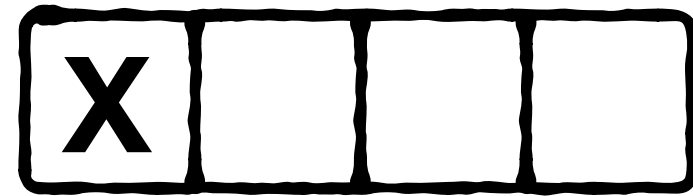
Main queue (LRU)



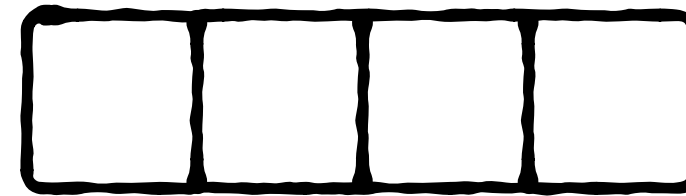
Buffer pool



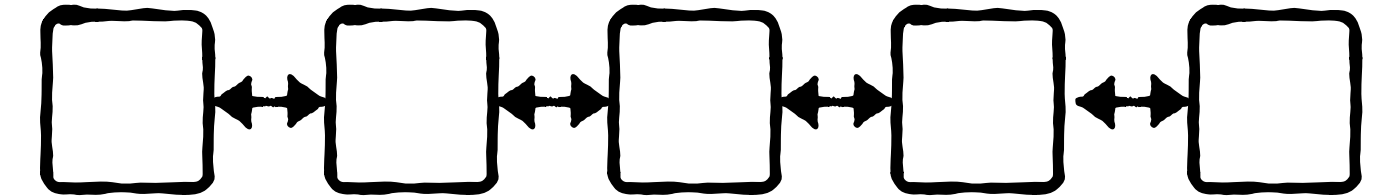
Recently evicted
(HT + FIFO)



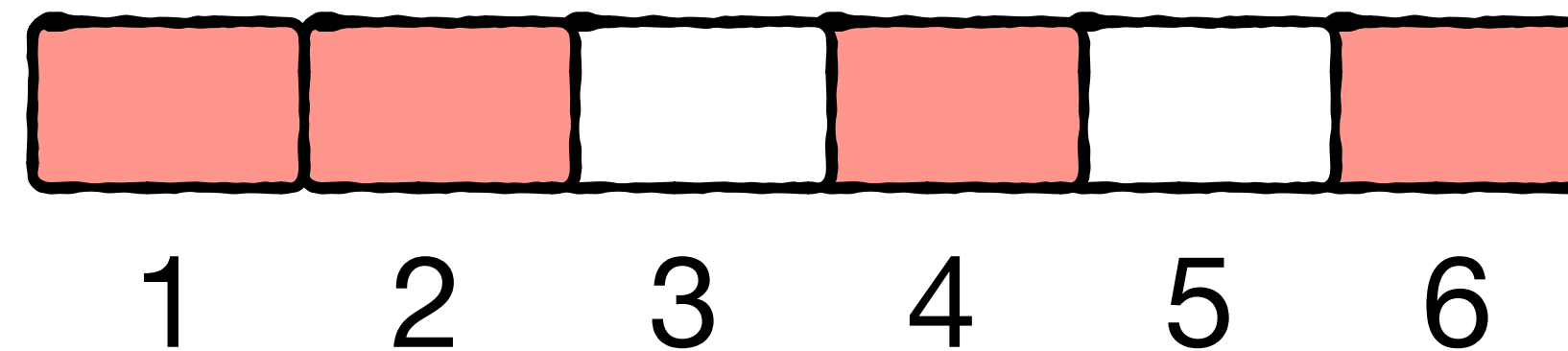
In queue (FIFO)



Main queue (LRU)

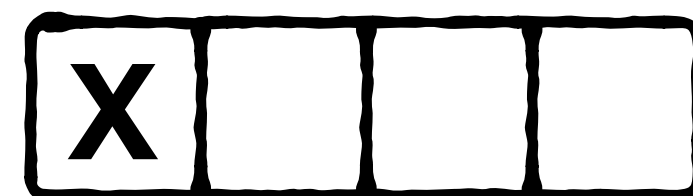


Buffer pool

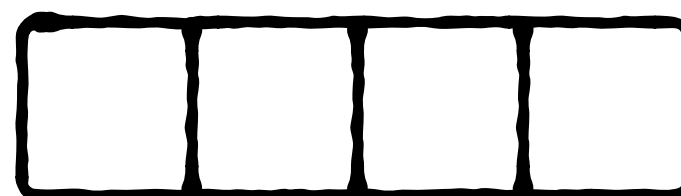


↑
Insert page X again

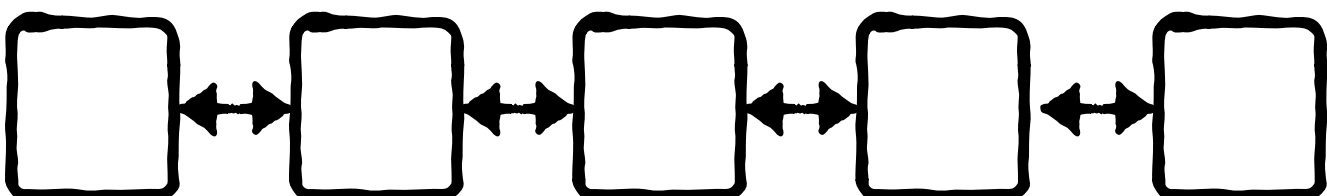
Recently evicted
(HT + FIFO)



In queue (FIFO)

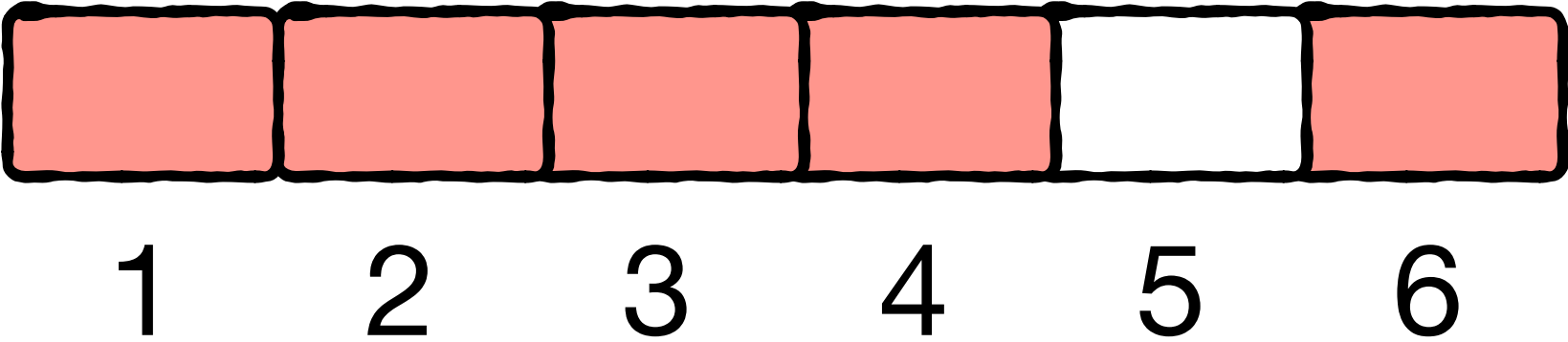


Main queue (LRU)

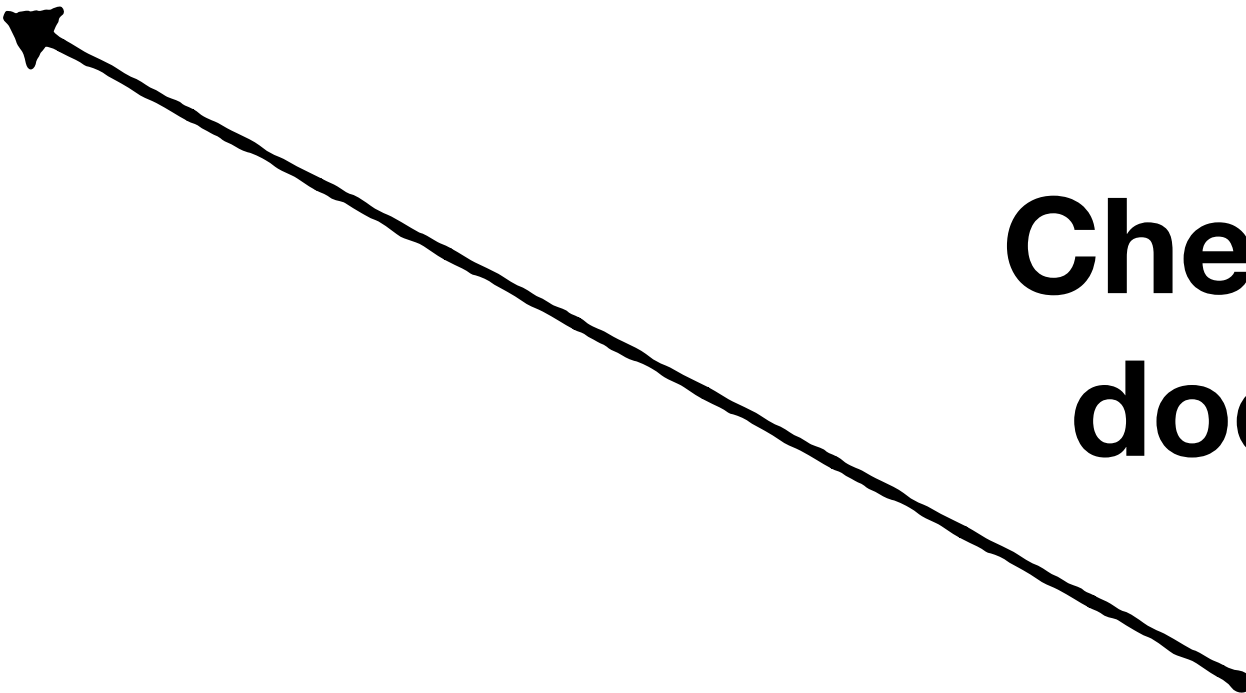


**Check for X,
does exist**

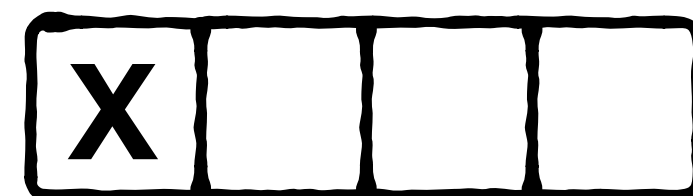
Buffer pool



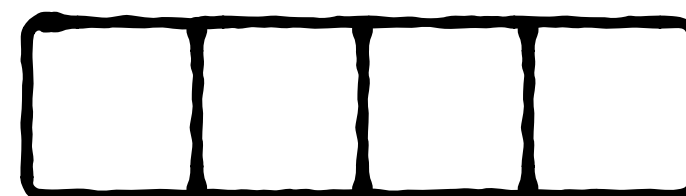
Insert page X again



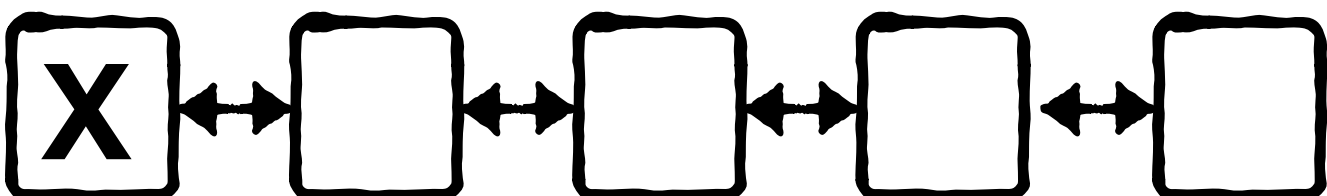
Recently evicted
(HT + FIFO)



In queue (FIFO)

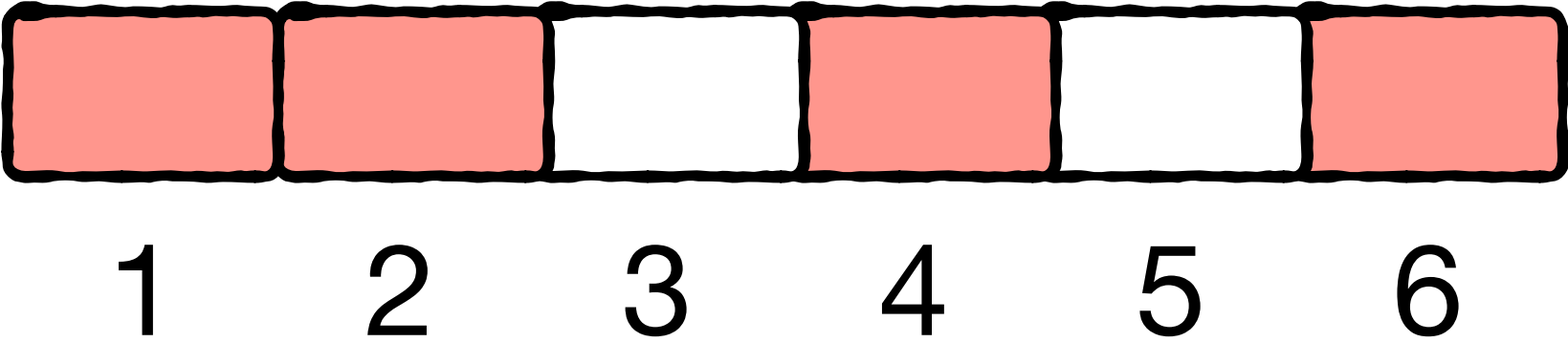


Main queue (LRU)



Insert

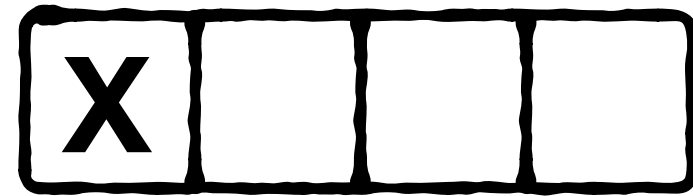
Buffer pool



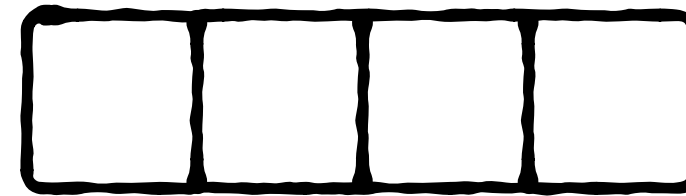
Insert page X again



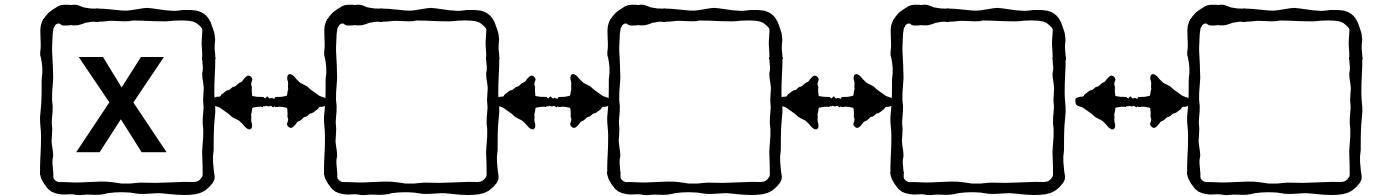
Recently evicted
(HT + FIFO)



In queue (FIFO)

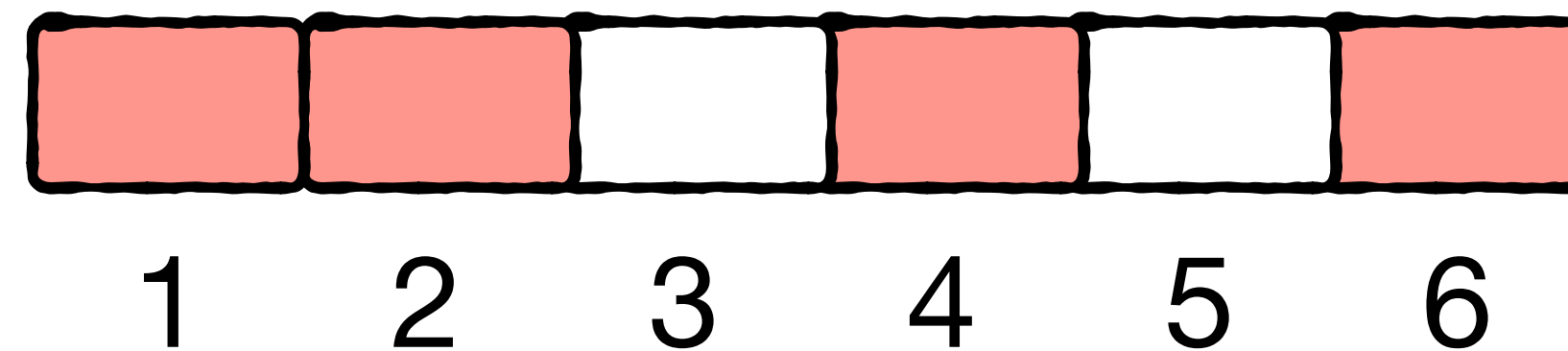


Main queue (LRU)

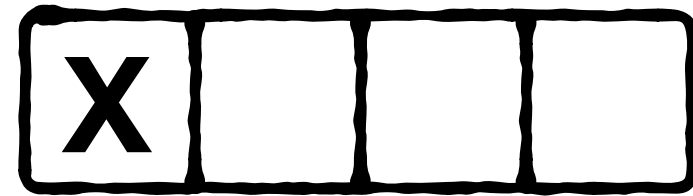


pages in main queue have had at least 2 recent accesses, so 2Q approximates LRU/K

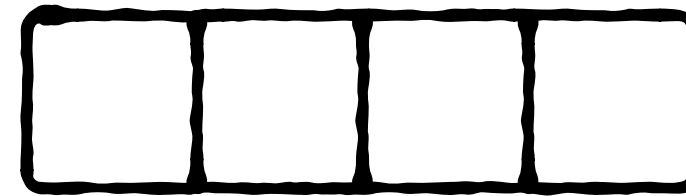
Buffer pool



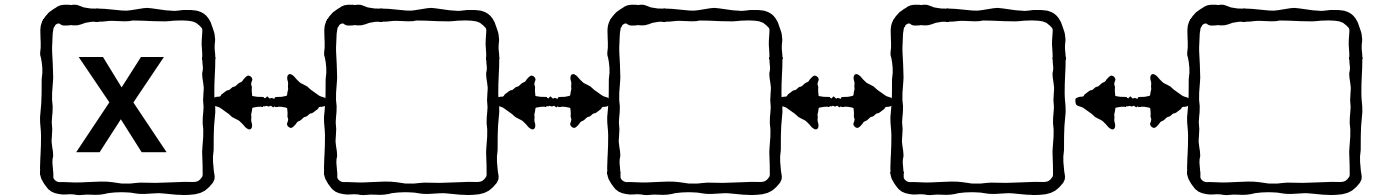
Recently evicted
(HT + FIFO)



In queue (FIFO)

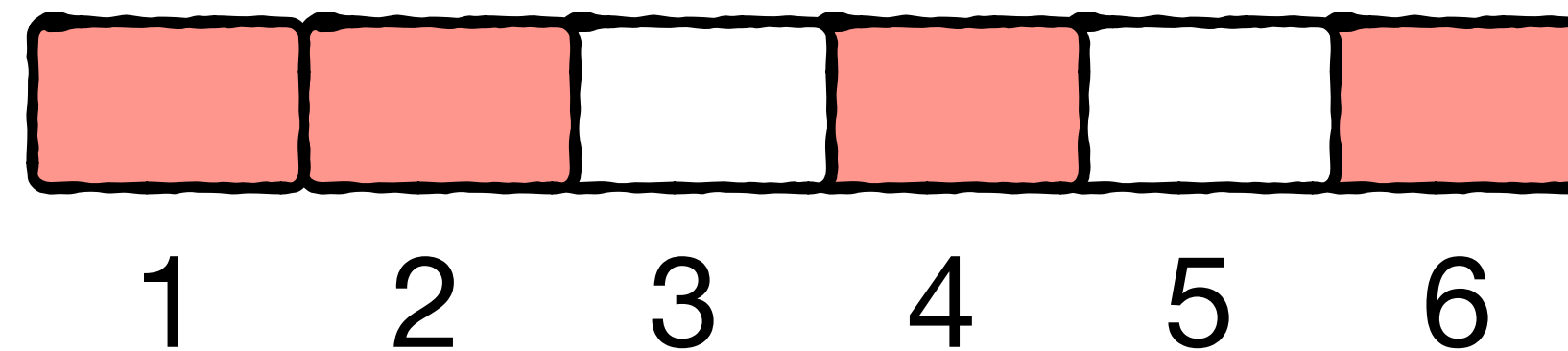


Main queue (LRU)



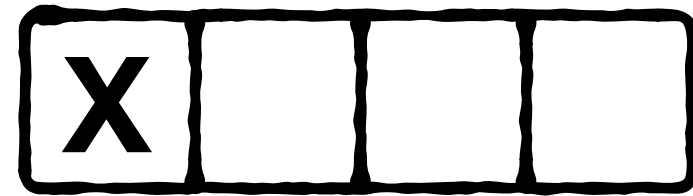
Each is a $O(1)$ data structure

Buffer pool

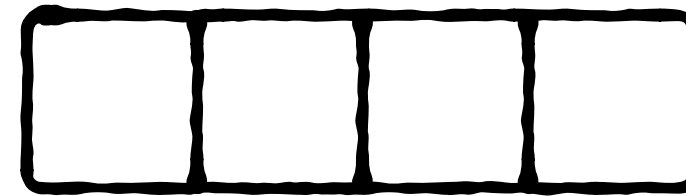


Problems?

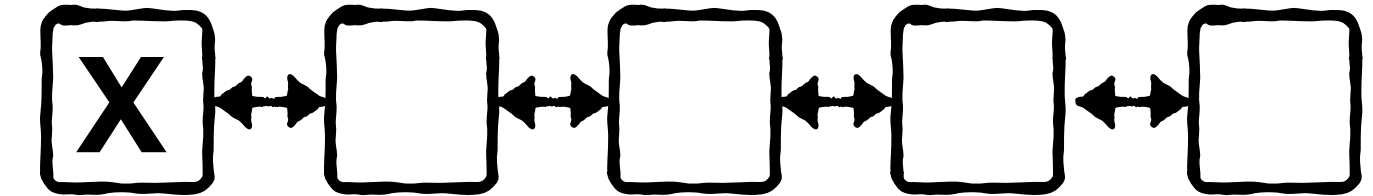
Recently evicted
(HT + FIFO)



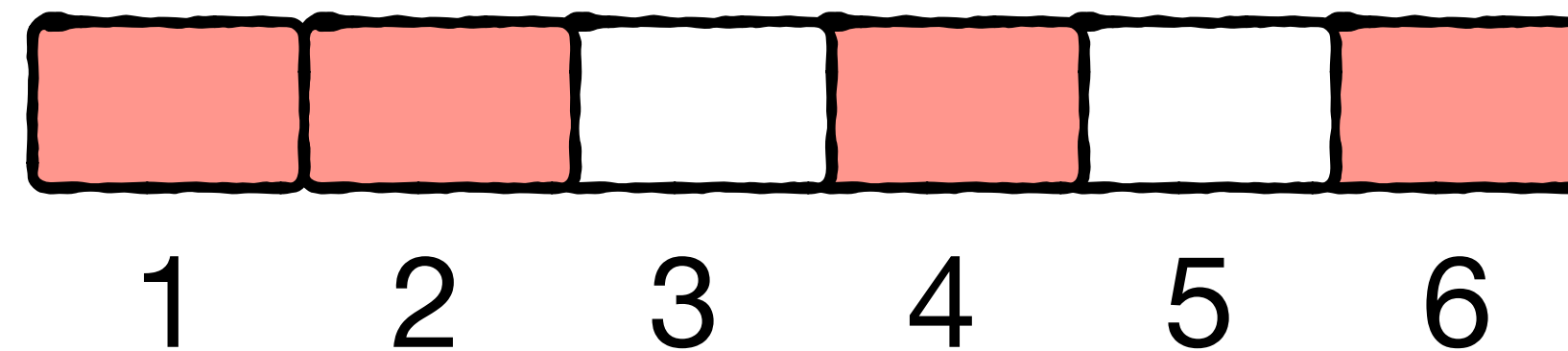
In queue (FIFO)



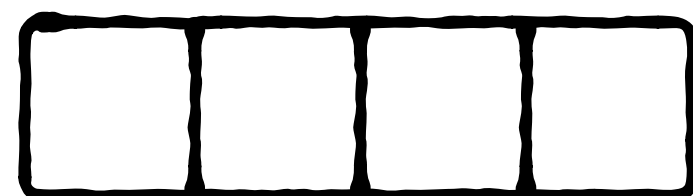
Main queue (LRU)



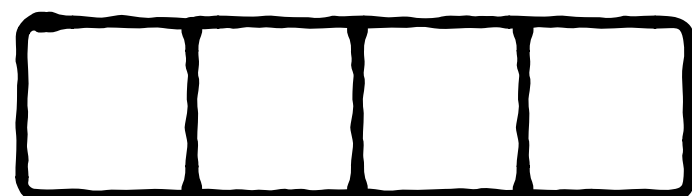
Buffer pool



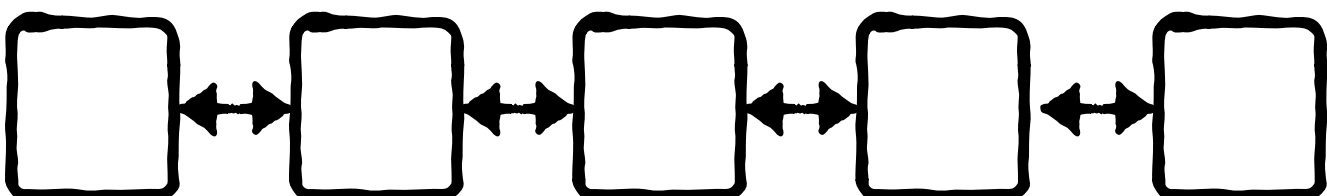
Recently evicted
(HT + FIFO)



In queue (FIFO)



Main queue (LRU)

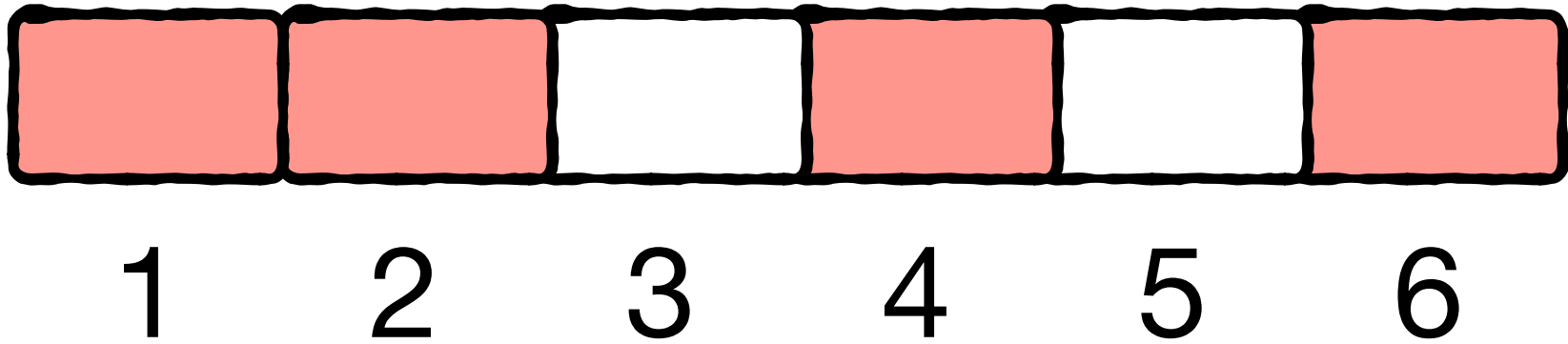


X%

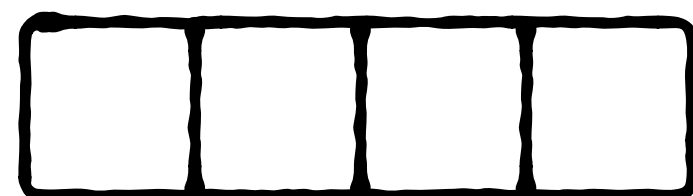


**What does this
size control?**

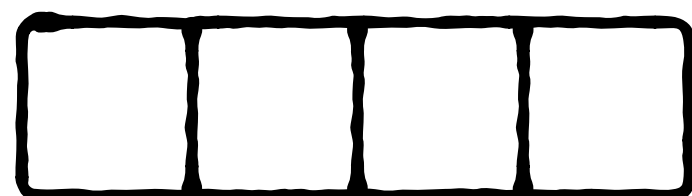
Buffer pool



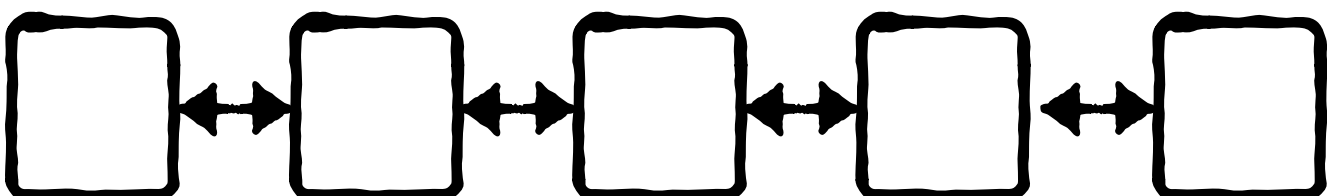
Recently evicted
(HT + FIFO)



In queue (FIFO)



Main queue (LRU)



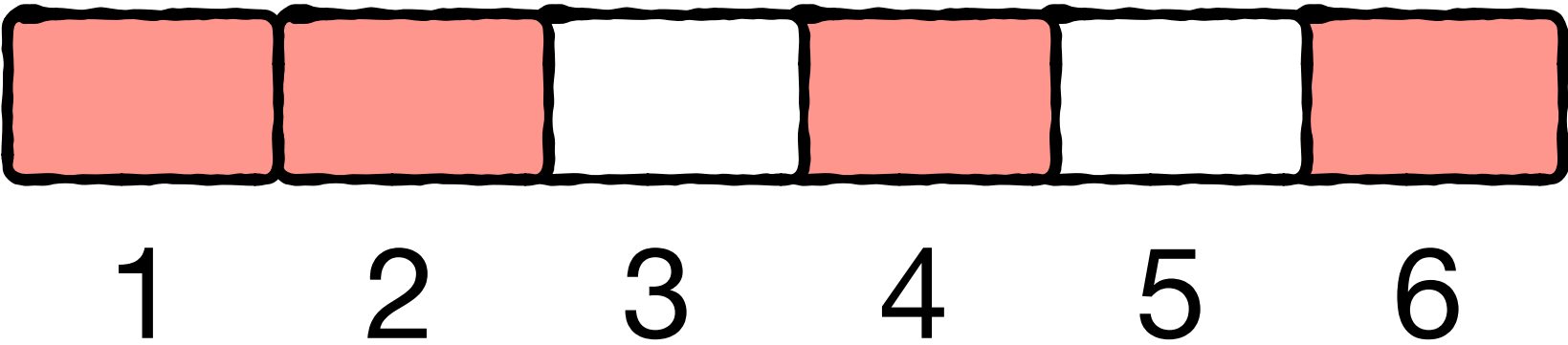
X%



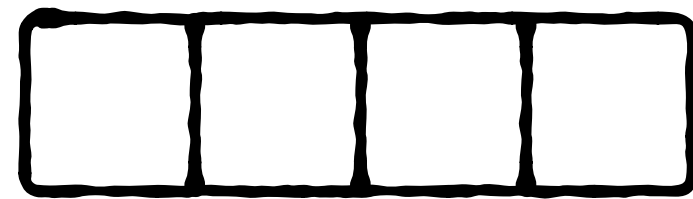
What does this
size control?

Spurt resistance.

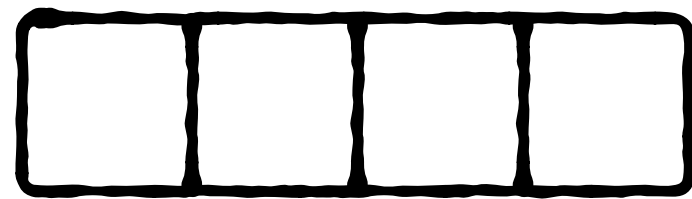
Buffer pool



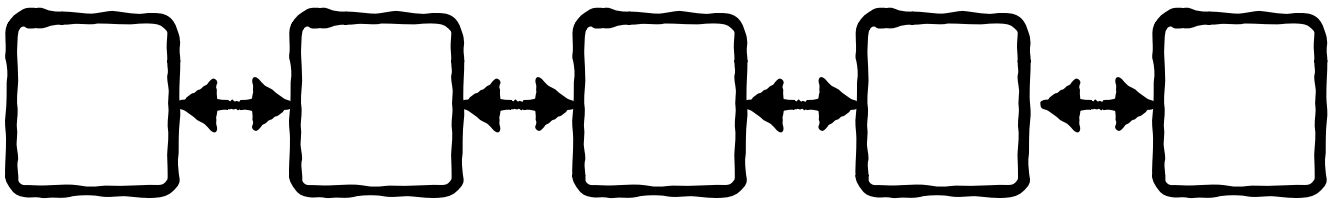
Recently evicted
(HT + FIFO)



In queue (FIFO)



Main queue (LRU)

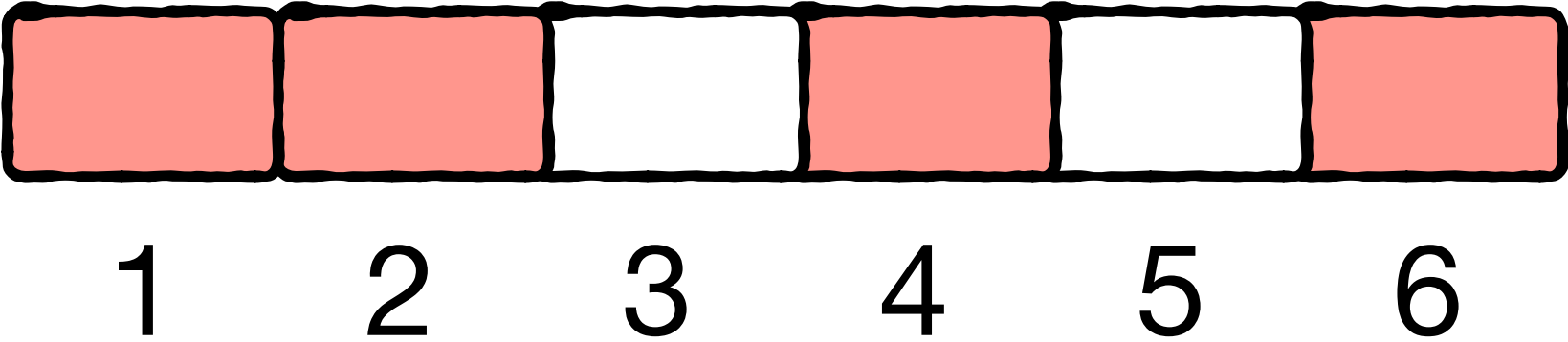


X%

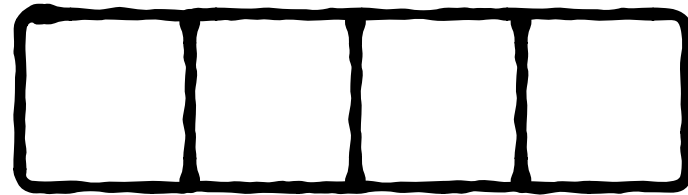


**If too small, spurling pages will be inserted into
main queue**

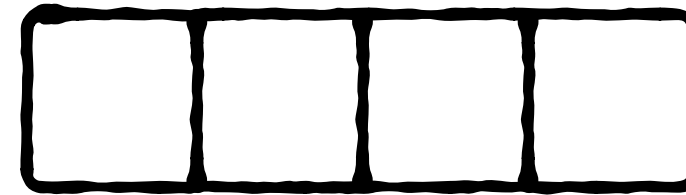
Buffer pool



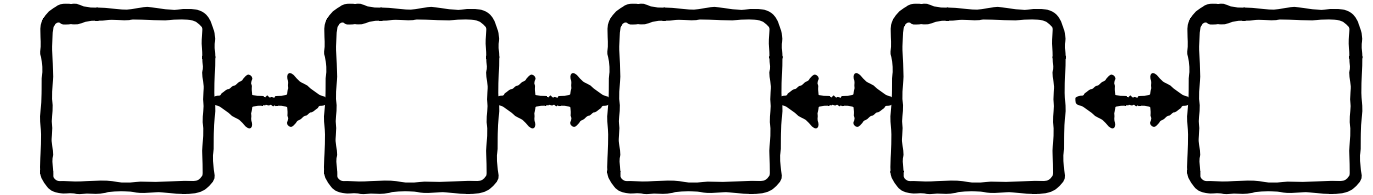
Recently evicted
(HT + FIFO)



In queue (FIFO)



Main queue (LRU)

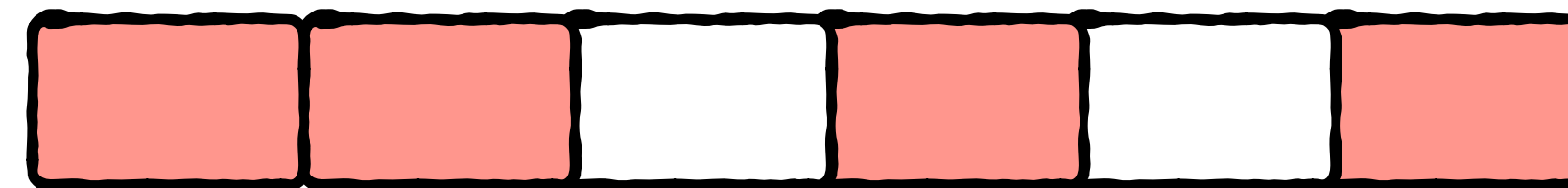


X%



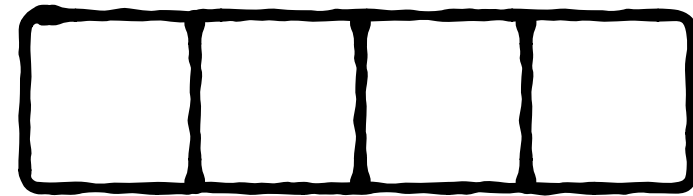
**If too large relative to main queue, scheme
degenerates into FIFO.**

Buffer pool

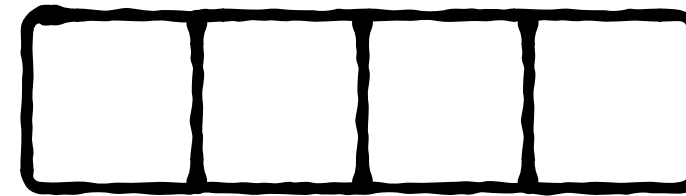


1 2 3 4 5 6

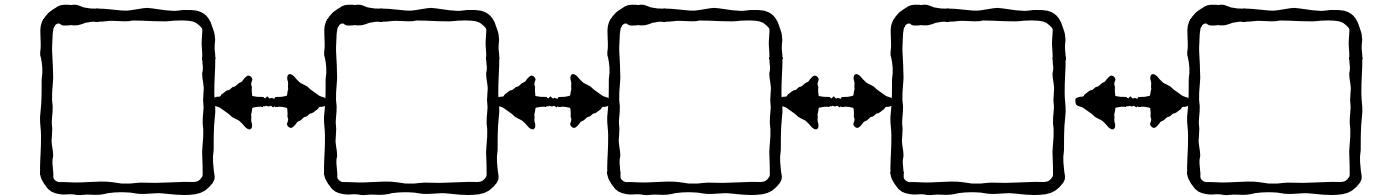
Recently evicted
(HT + FIFO)



In queue (FIFO)



Main queue (LRU)



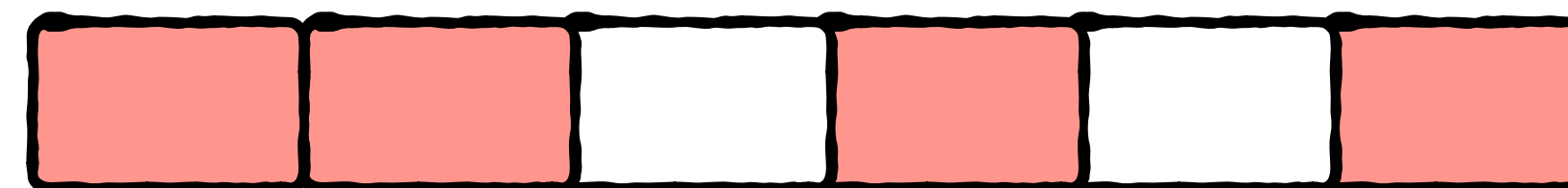
25%

75%



**Percentage slots
in buffer pool**

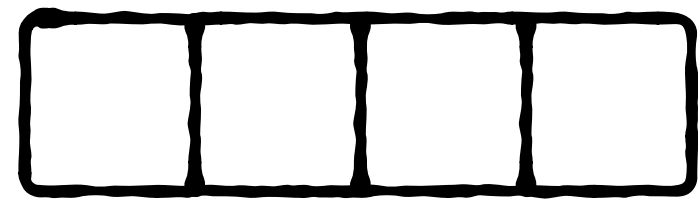
Buffer pool



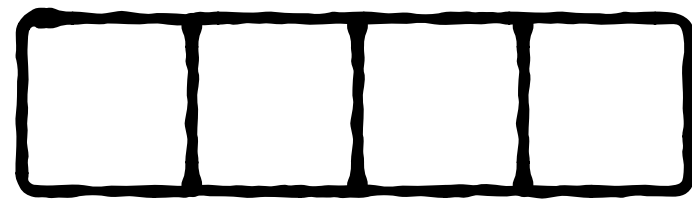
1 2 3 4 5 6

Other problems?

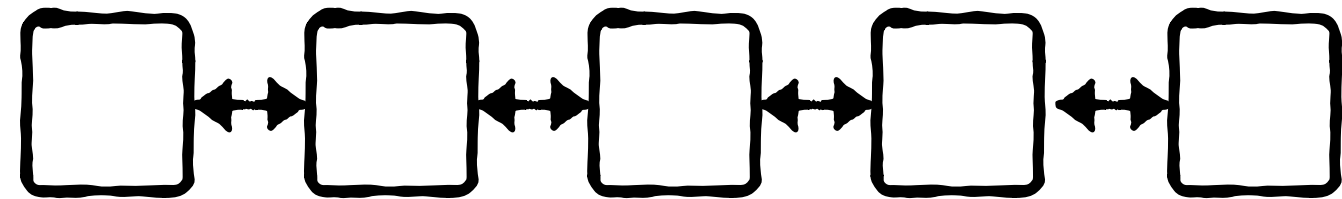
Recently evicted
(HT + FIFO)



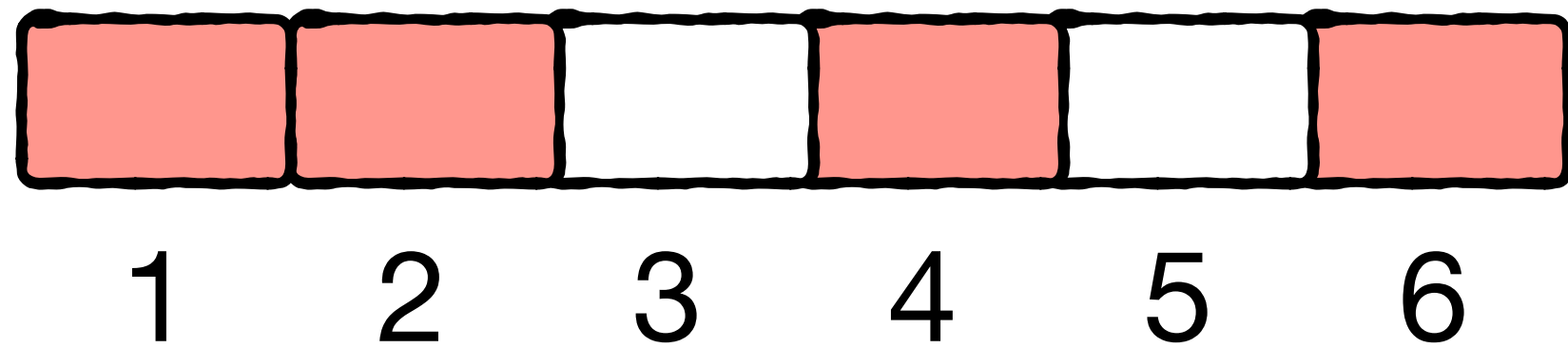
In queue (FIFO)



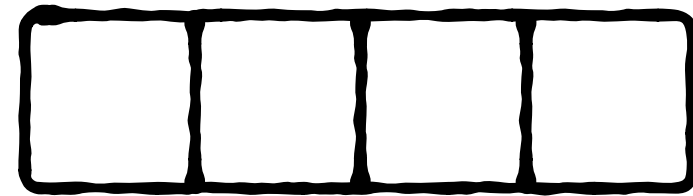
Main queue (LRU)



Buffer pool

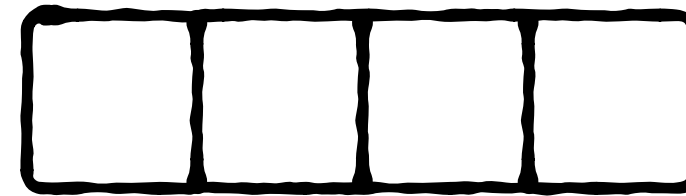


**Recently evicted
(HT + FIFO)**

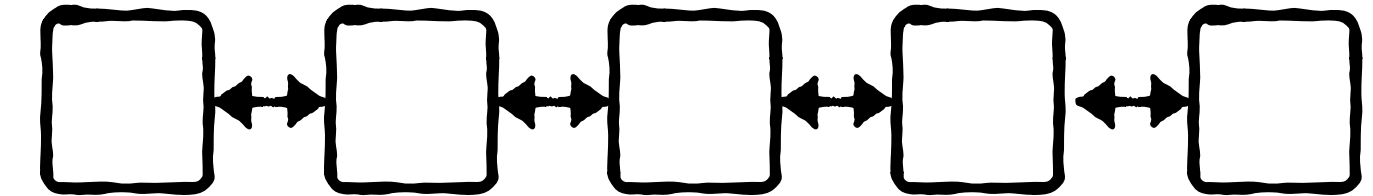


How to set size?

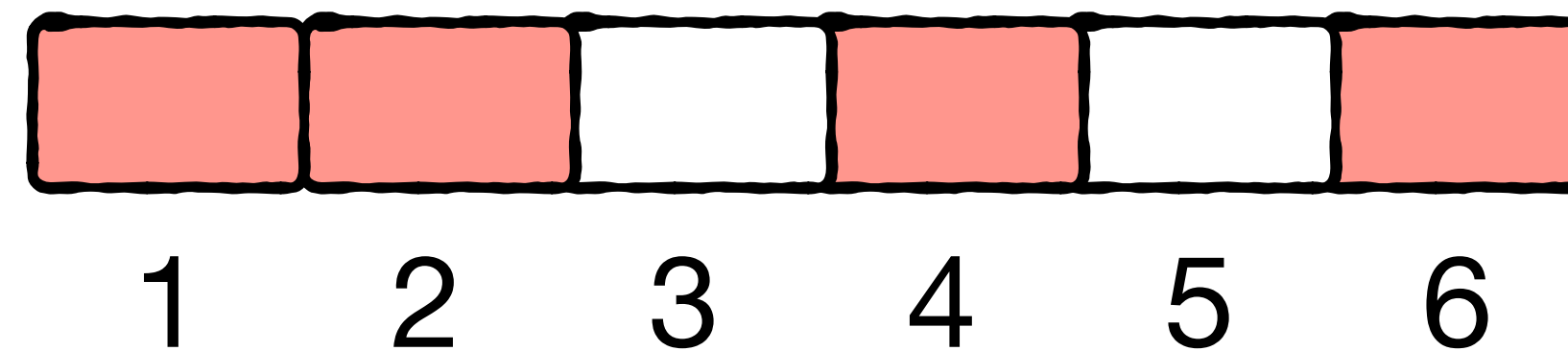
In queue (FIFO)



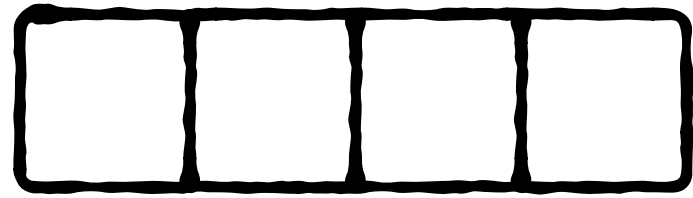
Main queue (LRU)



Buffer pool

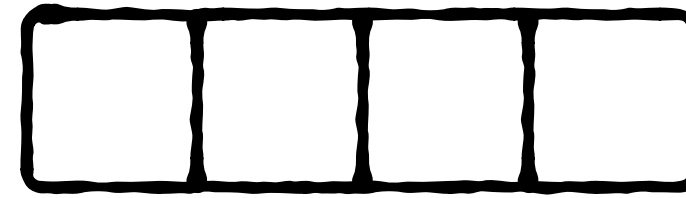


**Recently evicted
(HT + FIFO)**

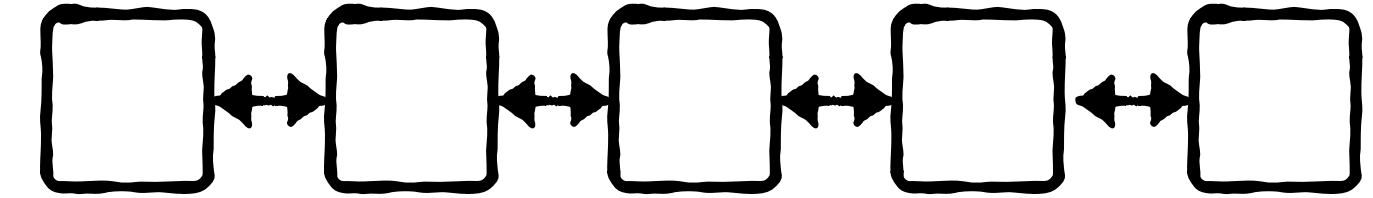


How to set size?

In queue (FIFO)



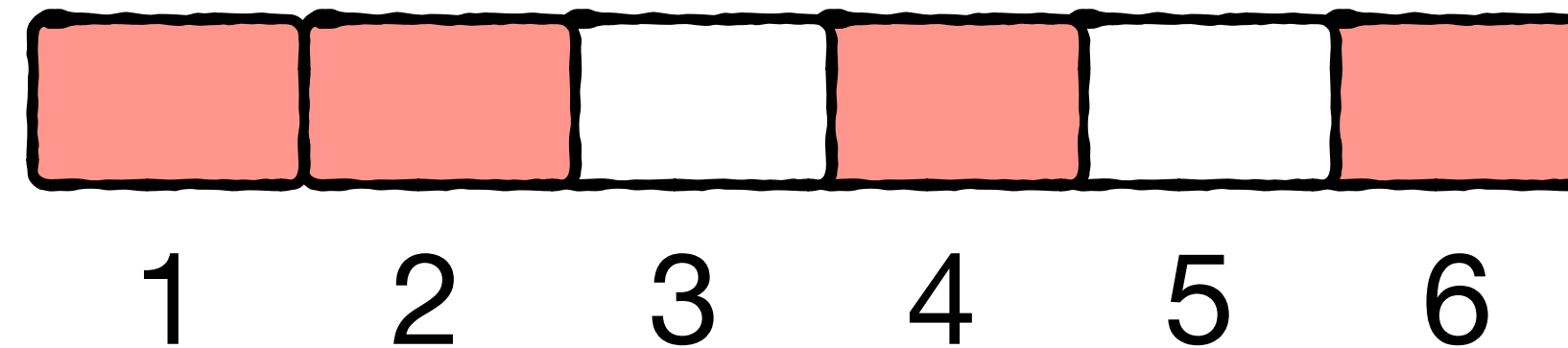
Main queue (LRU)



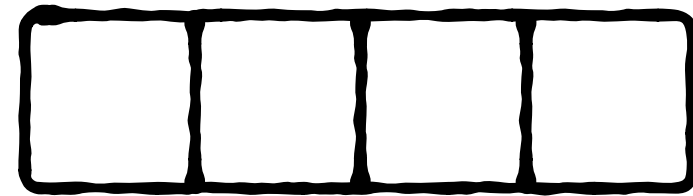
If too large:

If too small:

Buffer pool

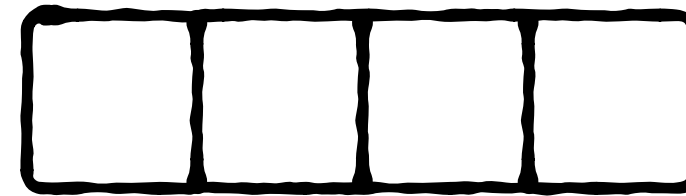


**Recently evicted
(HT + FIFO)**

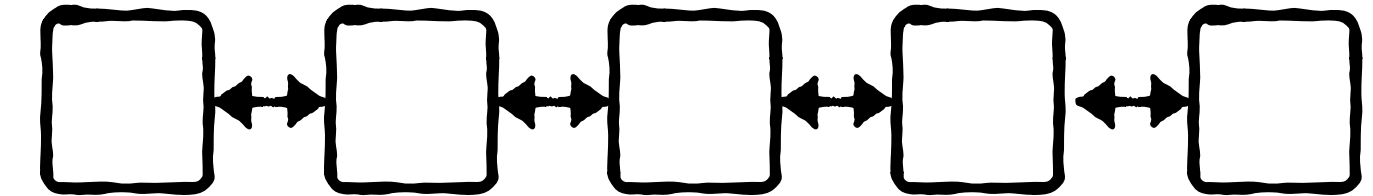


How to set size?

In queue (FIFO)



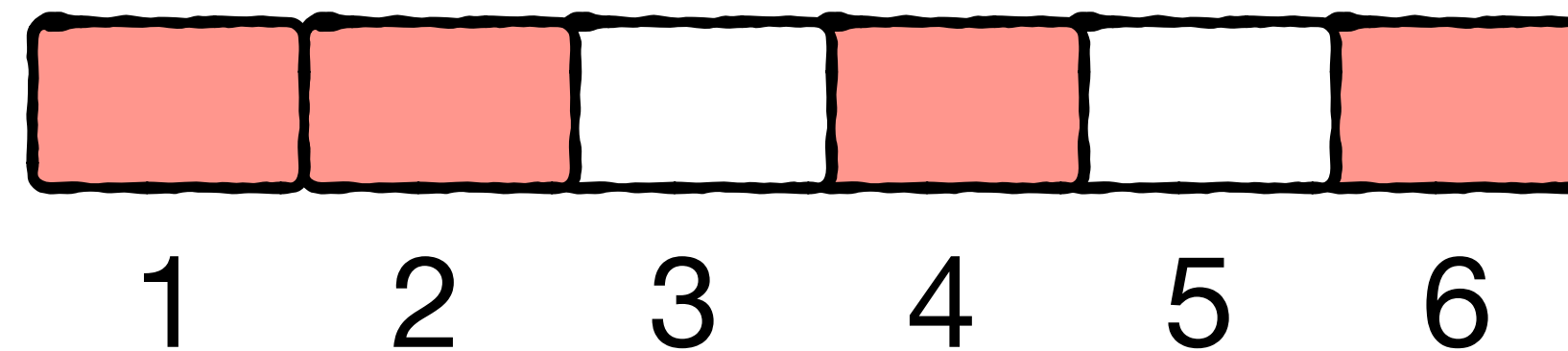
Main queue (LRU)



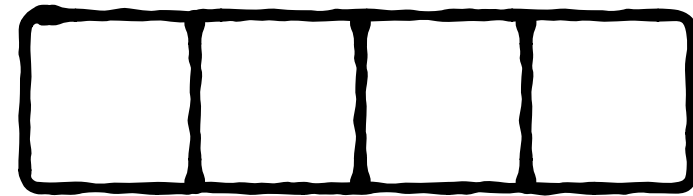
If too large: degenerate into LRU

If too small:

Buffer pool

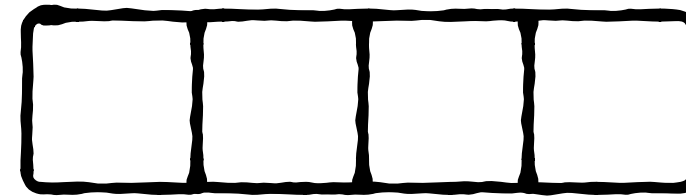


**Recently evicted
(HT + FIFO)**

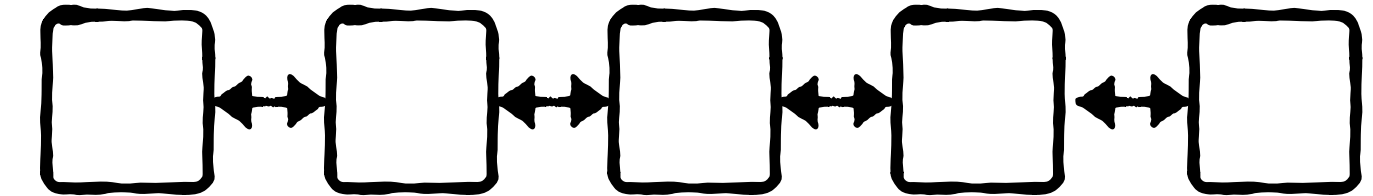


How to set size?

In queue (FIFO)



Main queue (LRU)



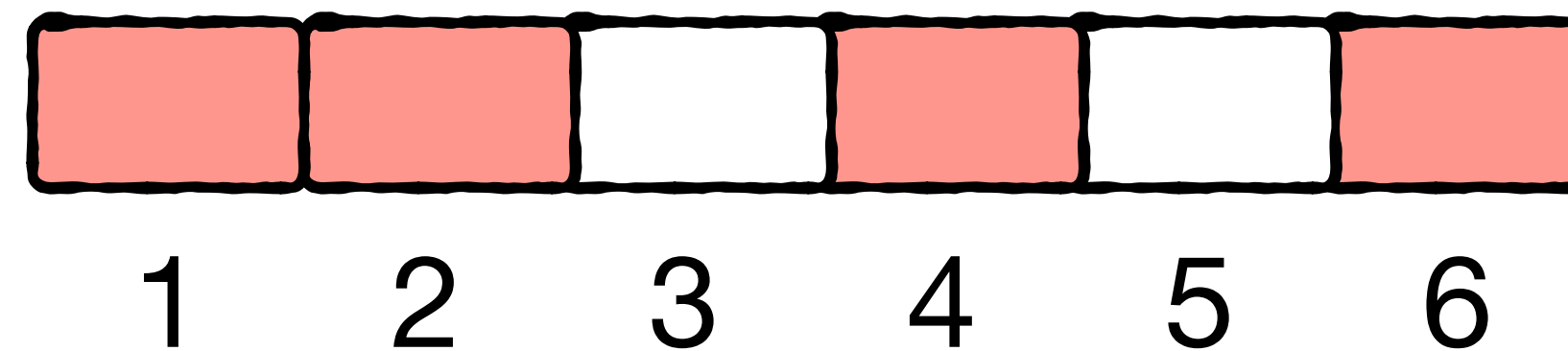
If too large:

degenerate into LRU

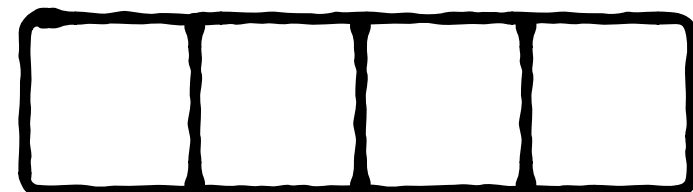
If too small:

degenerate into FIFO

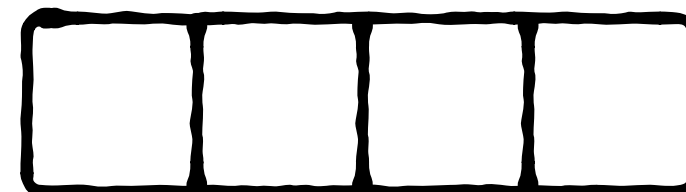
Buffer pool



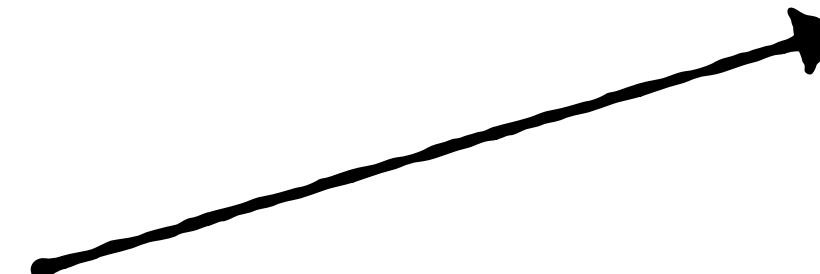
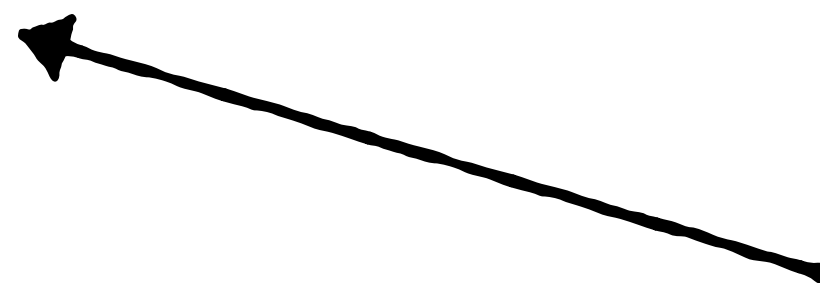
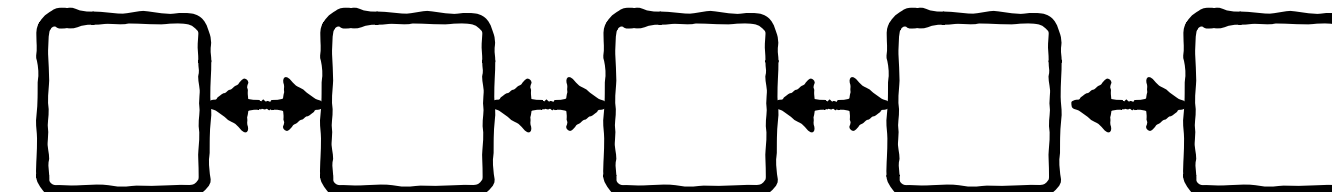
Recently evicted
(HT + FIFO)



In queue (FIFO)

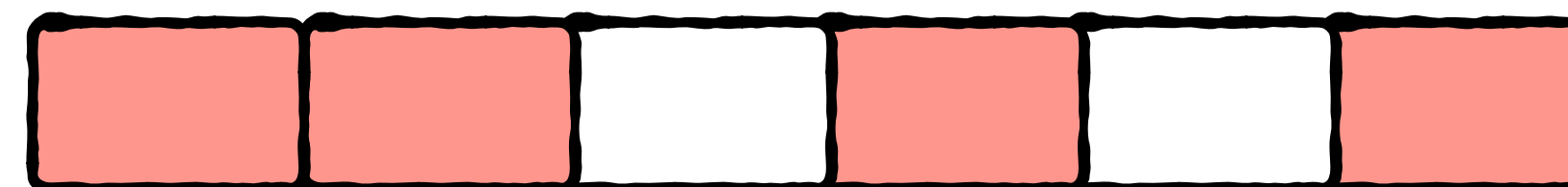


Main queue (LRU)



**Overall, tuning is
sensitive**

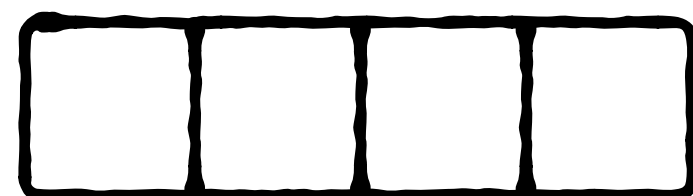
Buffer pool



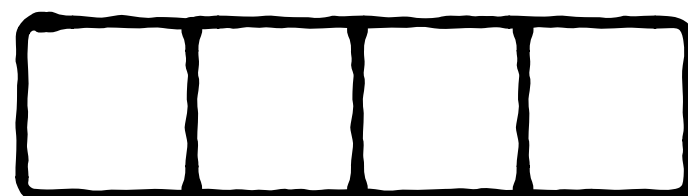
1 2 3 4 5 6

Any other problem?

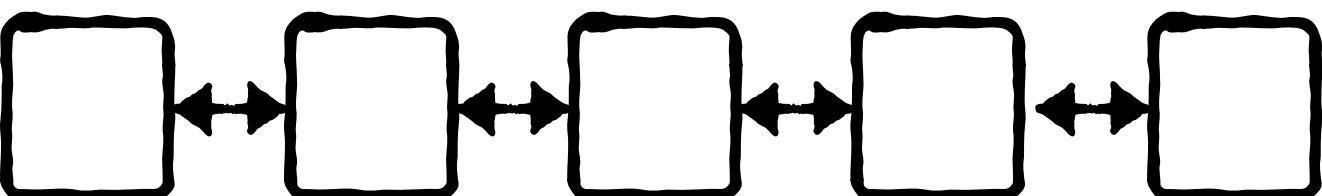
Recently evicted
(HT + FIFO)



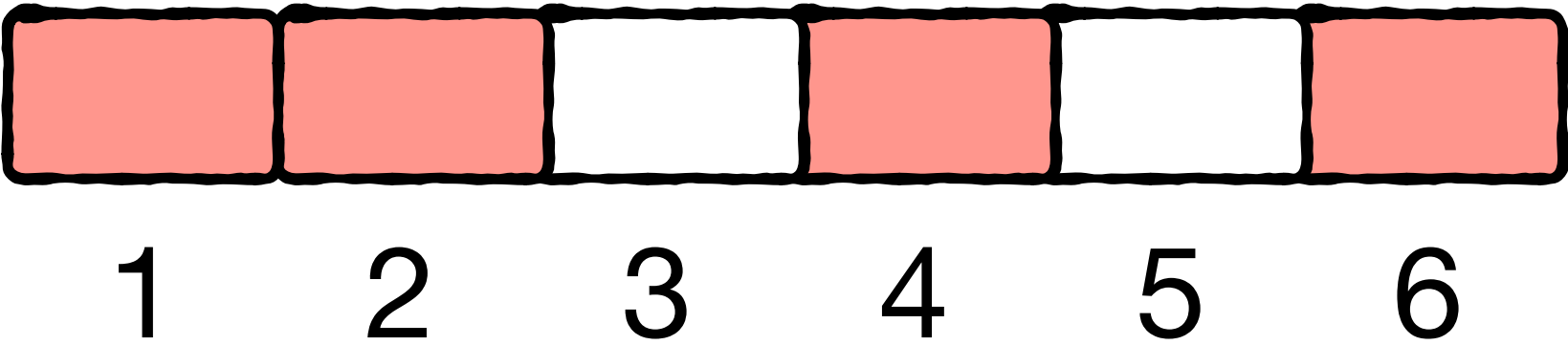
In queue (FIFO)



Main queue (LRU)

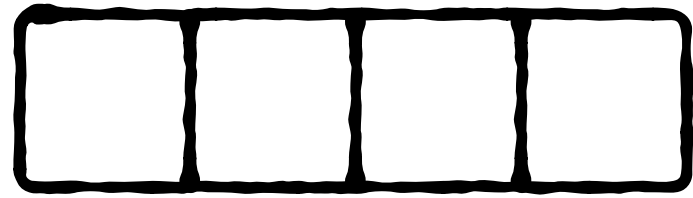


Buffer pool

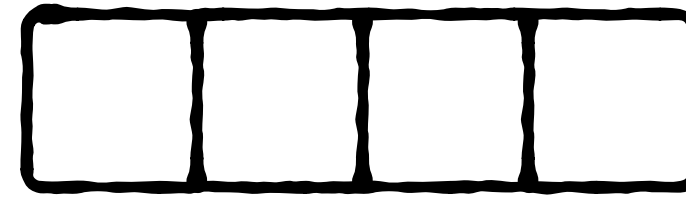


Any other problem?

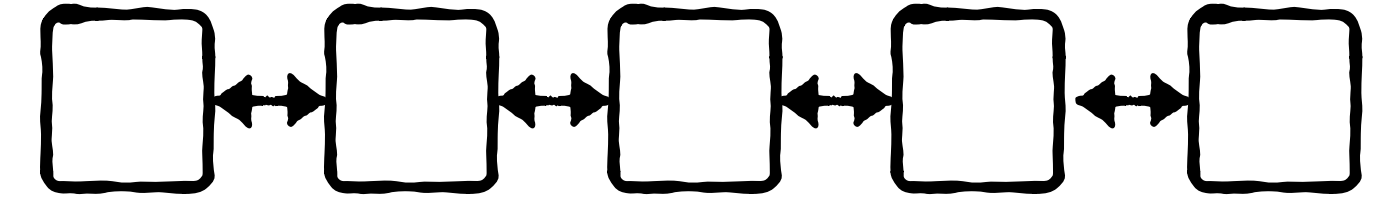
Recently evicted
(HT + FIFO)



In queue (FIFO)

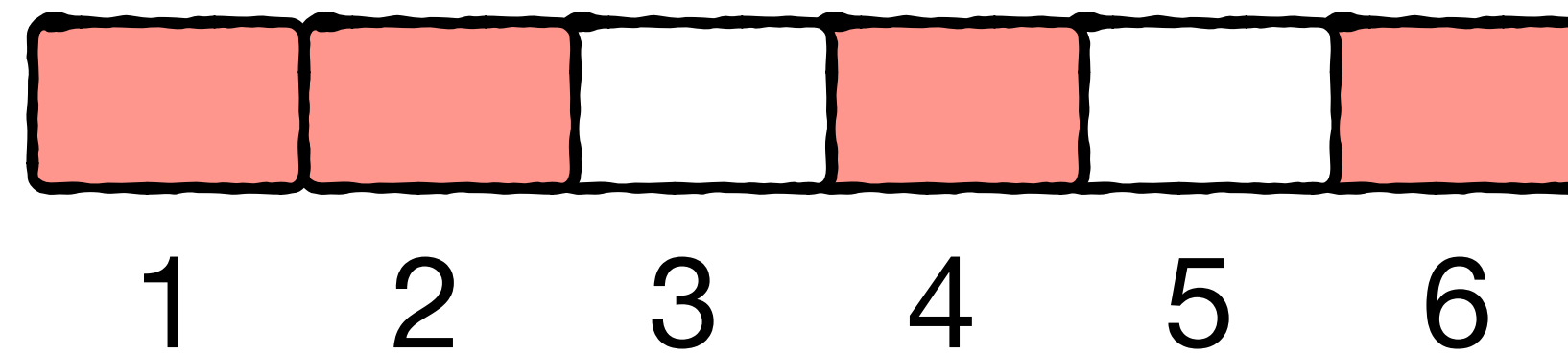


Main queue (LRU)

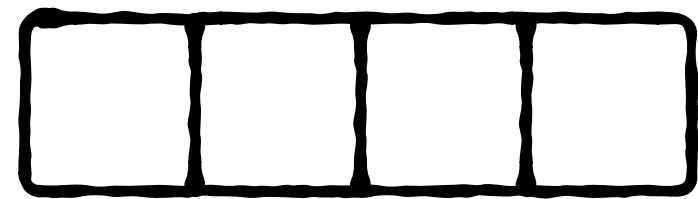


**Newly hot page must be evicted & reread
once to make it into main queue**

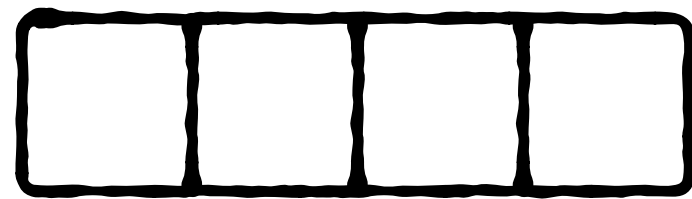
Buffer pool



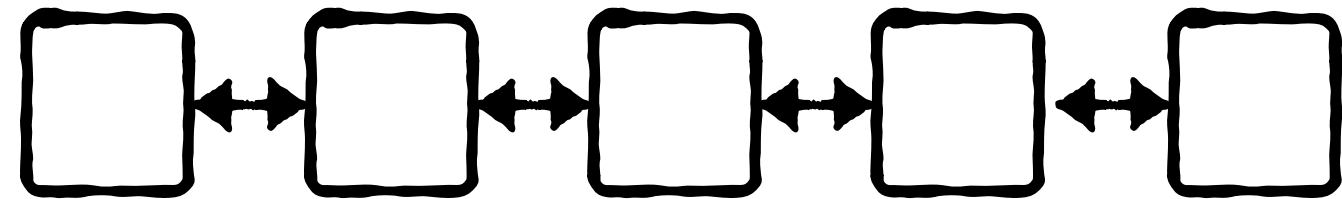
Recently evicted
(HT + FIFO)



In queue (FIFO)

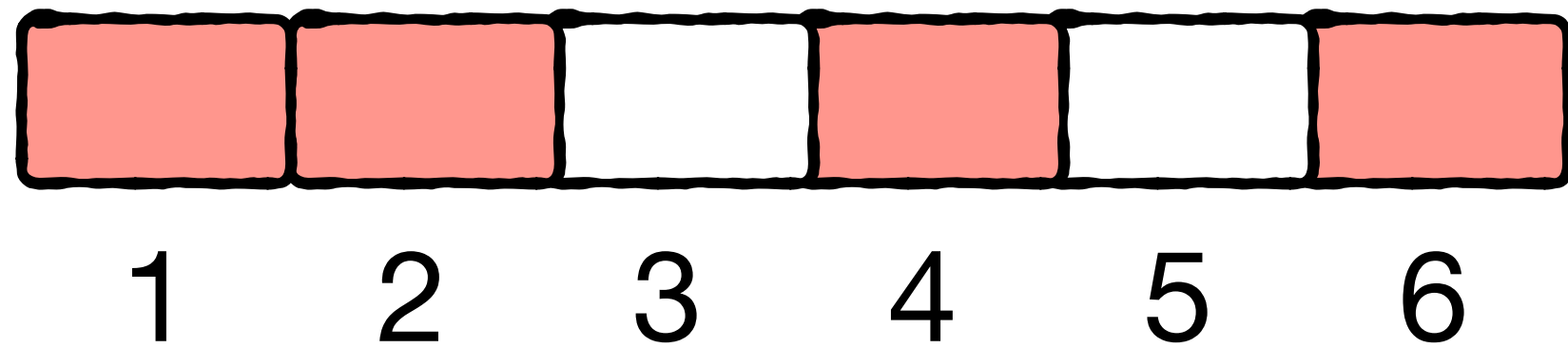


Main queue (LRU)



Questions?

Buffer pool



Thank you!