UNIVERSITY of WASHINGTON

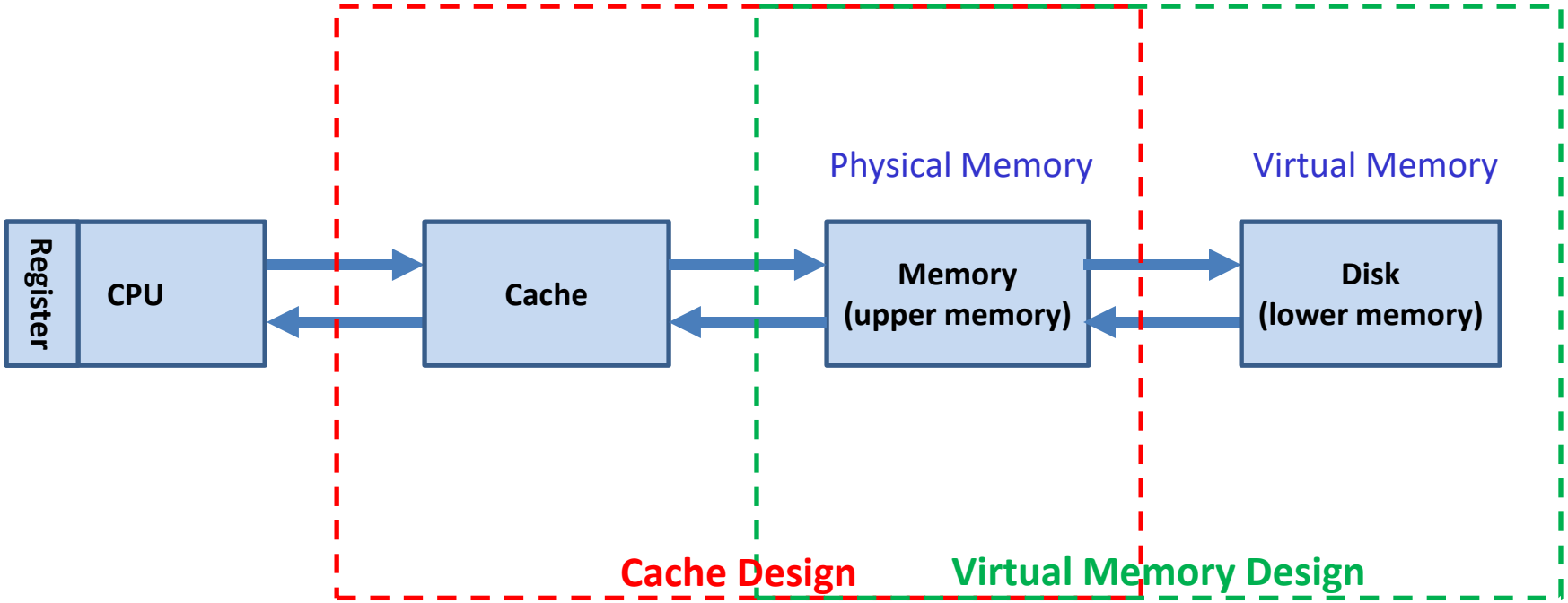
CSS 422 Hardware and Computer Organization

LRU Algorithm

Professor: Yang Peng

The slides are re-produced by the courtesy of
Dr. Arnie Berger, Dr. Ross Ortega and Dr. Wooyoung Kim

Accessing Memories in a Computer System



Replacement Policy

- When all the blocks in cache are used, we need to find a "***victim***" block to refill, where we need a ***Replacement Policy***

- *Optimal replacement policy*: impossible to implement as it should predict all the future cache references completely

Three Most Common Replacement Policies

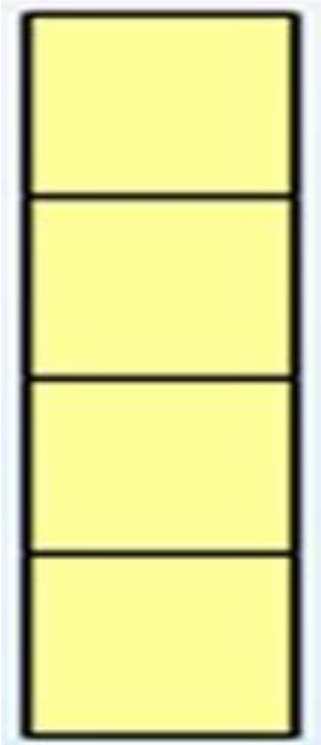
- ***First-In, First-Out* (FIFO)**
  - Throw away the frame that has been in the cache the longest, regardless of when it was last used

- ***Random***
  - Picks a frame at random and replaces it with a new one
  - Disadvantage: Can certainly evict a frame that will be needed often or needed soon

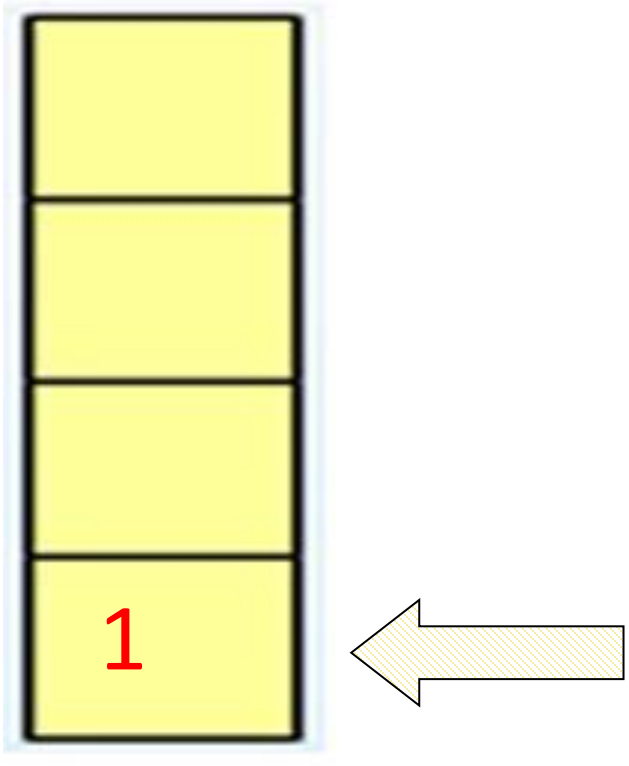
- ***Least Recently Used* (LRU)**
  - Evicts the frame that has been unused for the longest period of time
  - Disadvantage: Complex, as it has to maintain an access history for each block, which ultimately slows down the cache.

Implementing an LRU Stack

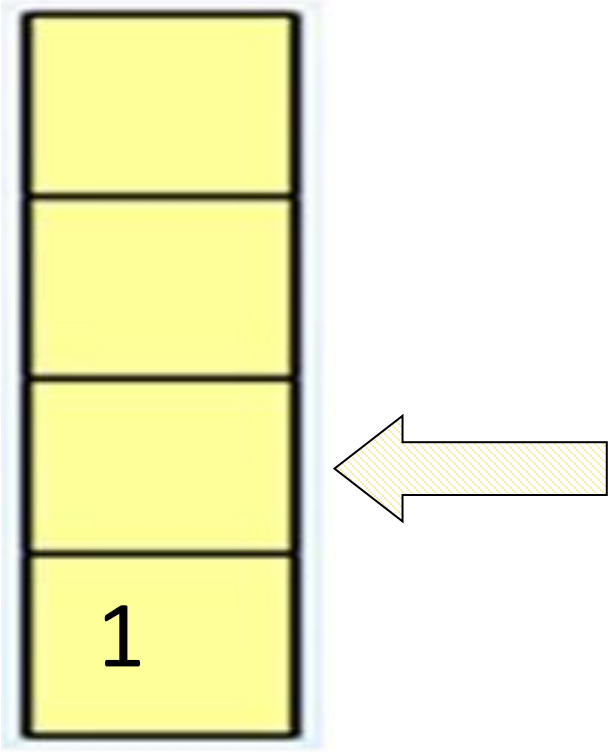
REF string: 1 5 3 1 6 2 3



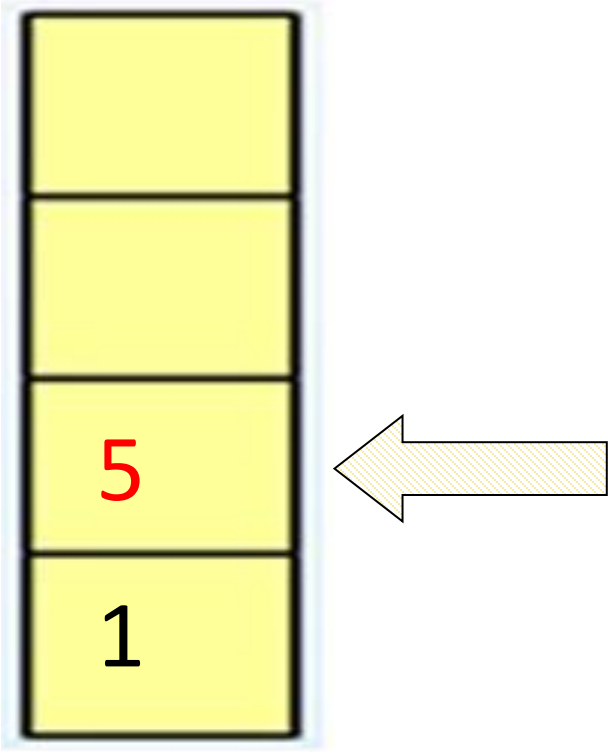
REF string: 1 5 3 1 6 2 3



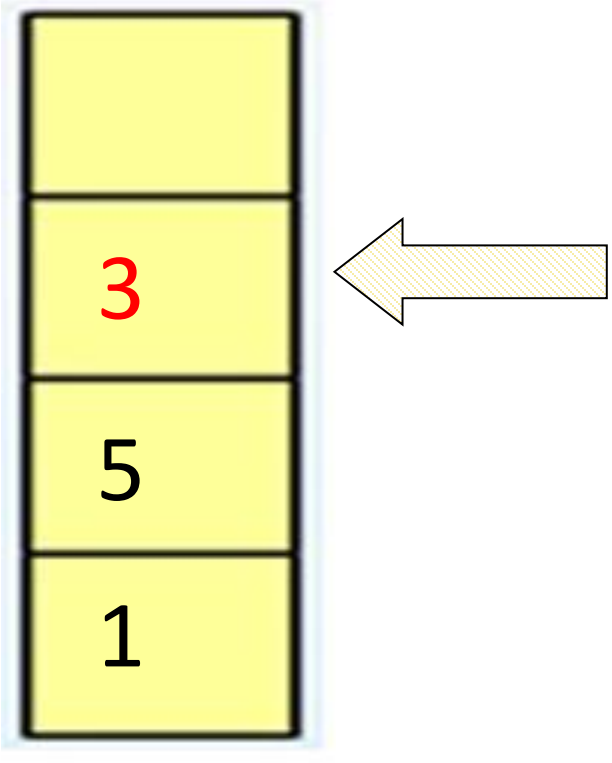
REF string: 1 5 3 1 6 2 3



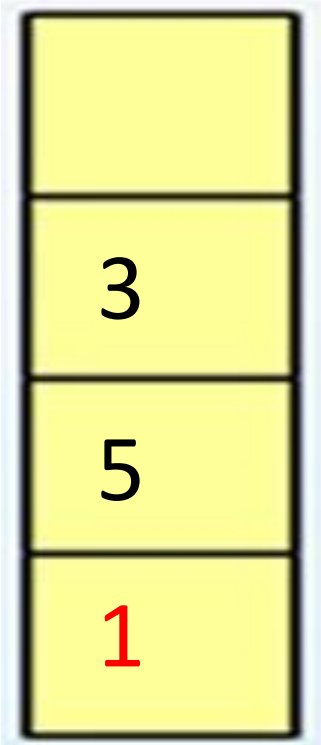
REF string: 1 5 3 1 6 2 3



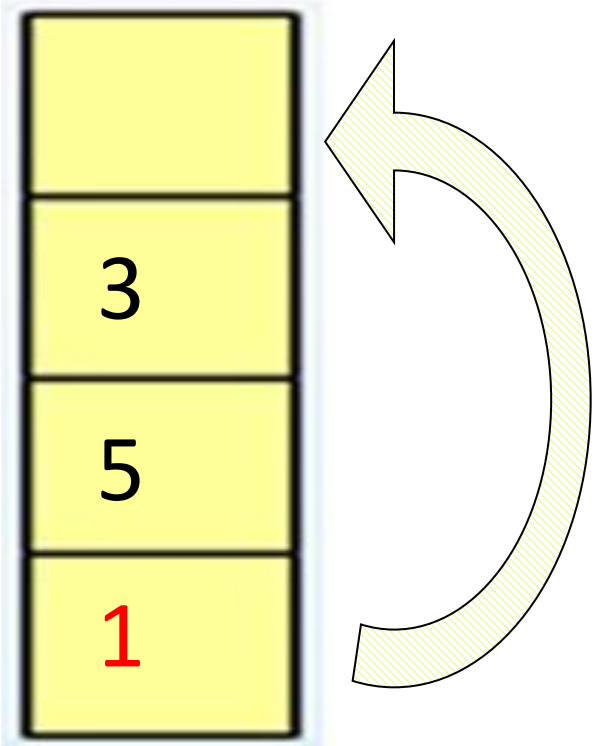
REF string: 1 5 3 1 6 2 3



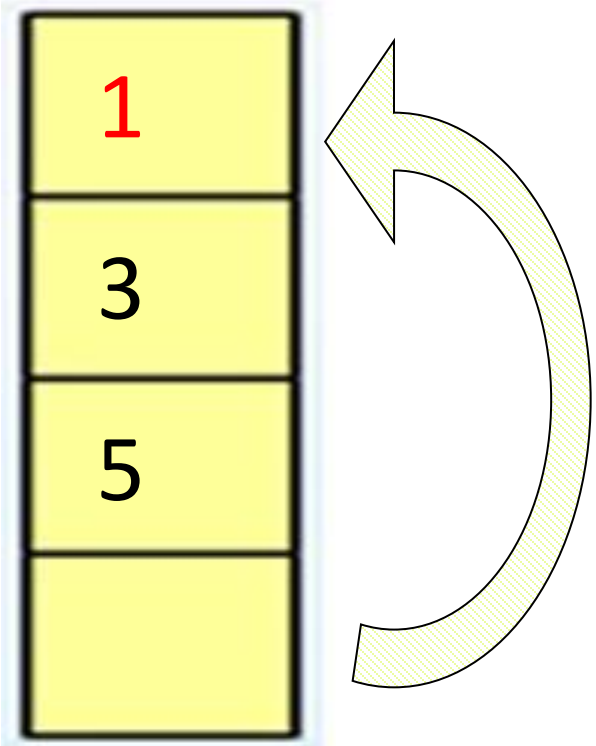
REF string: 1 5 3 1 6 2 3



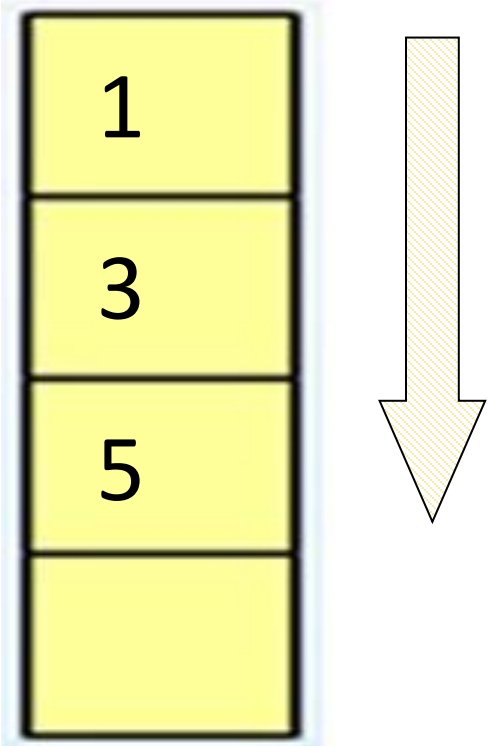
REF string: 1 5 3 1 6 2 3



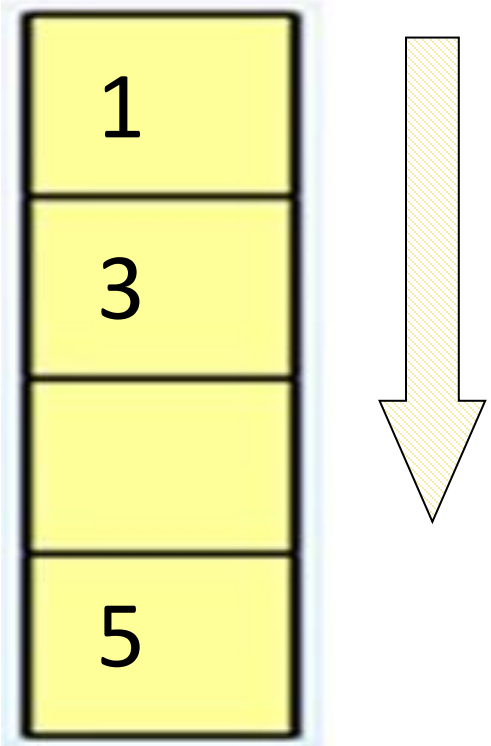
REF string: 1 5 3 1 6 2 3



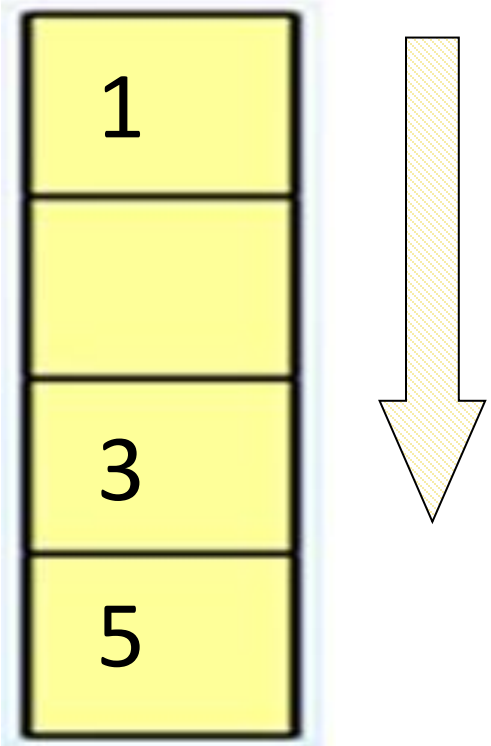
REF string: 1 5 3 1 6 2 3



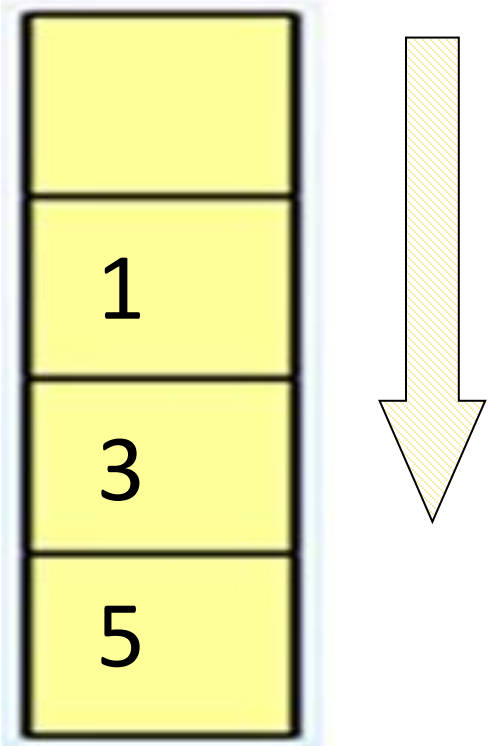
REF string: 1 5 3 1 6 2 3



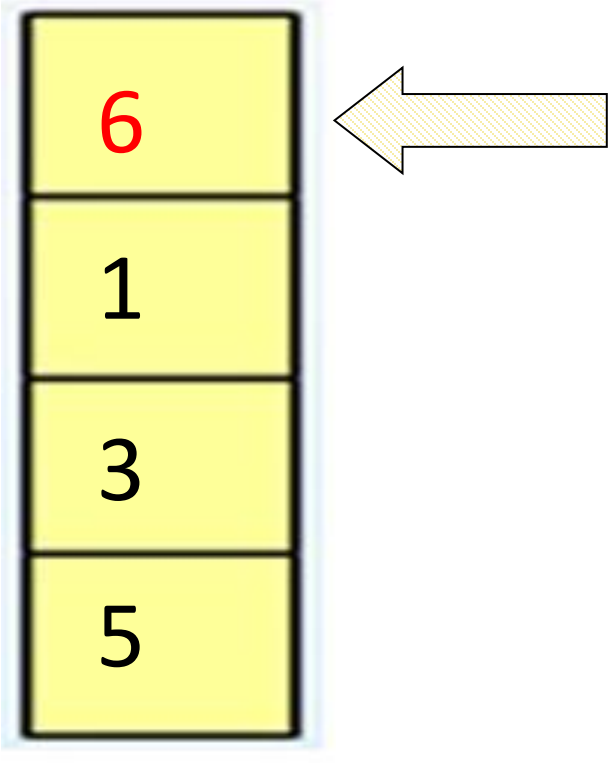
REF string: 1 5 3 1 6 2 3



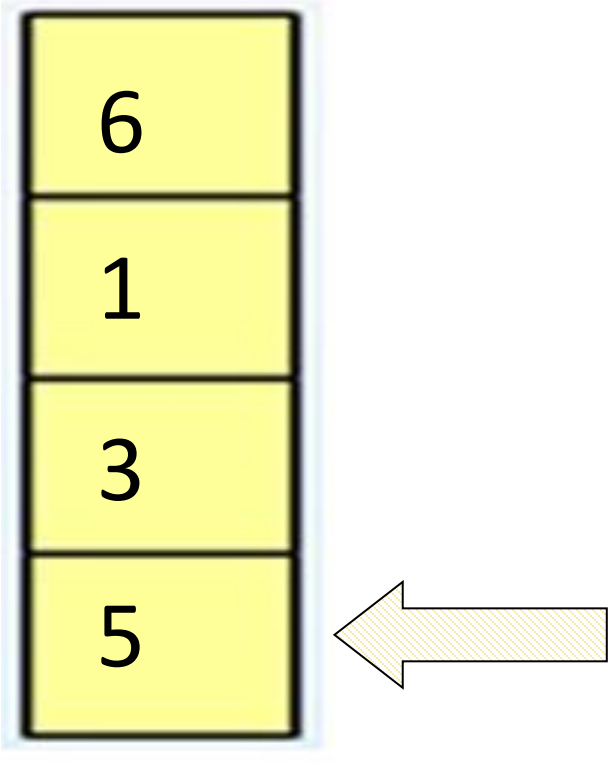
REF string: 1 5 3 1 6 2 3



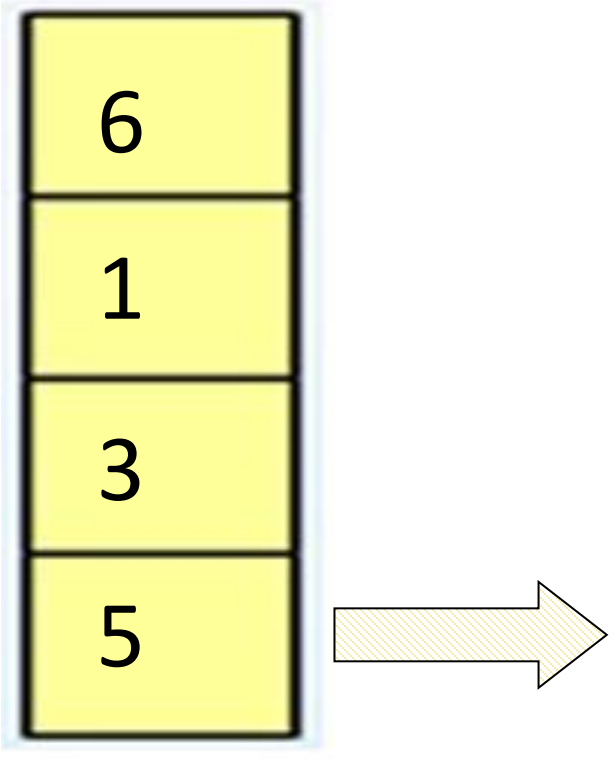
REF string: 1 5 3 1 6 2 3



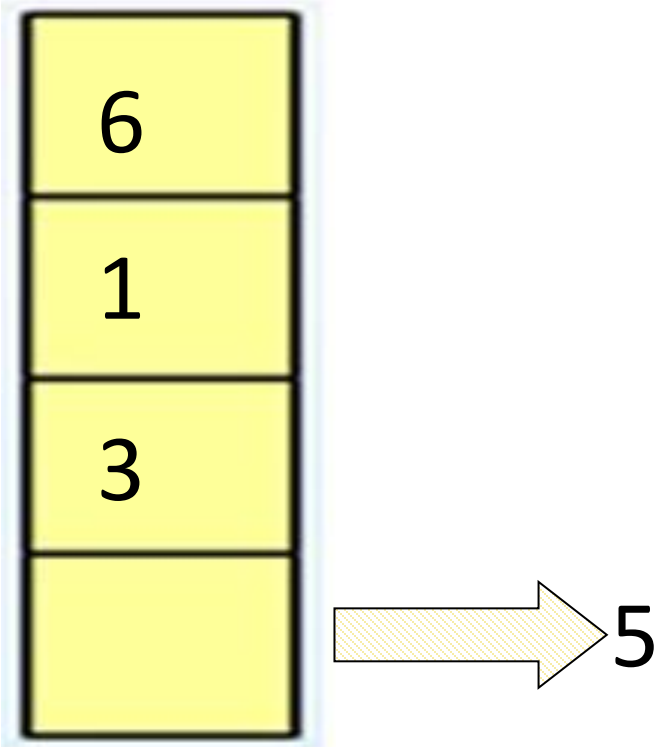
REF string: 1 5 3 1 6 2 3



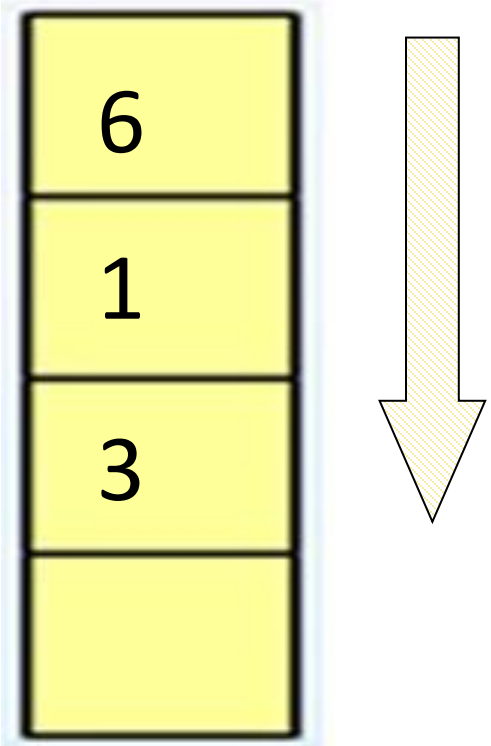
REF string: 1 5 3 1 6 2 3



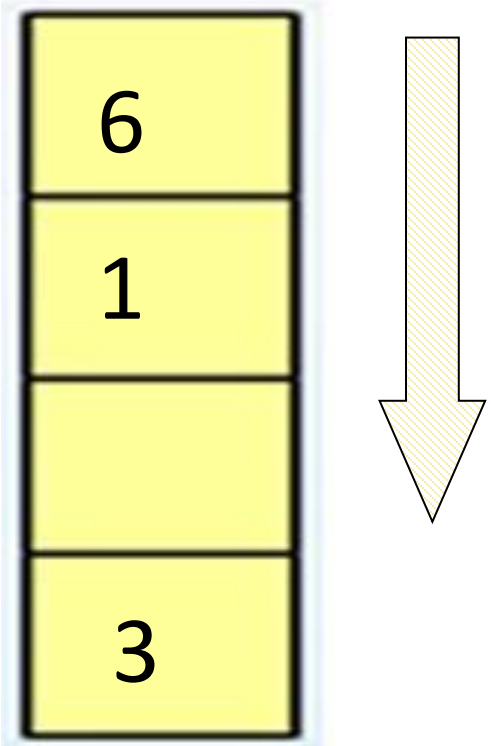
REF string: 1 5 3 1 6 2 3



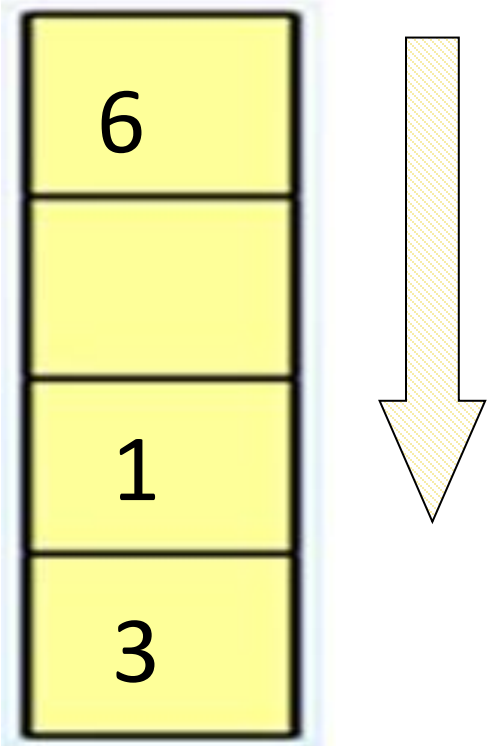
REF string: 1 5 3 1 6 2 3



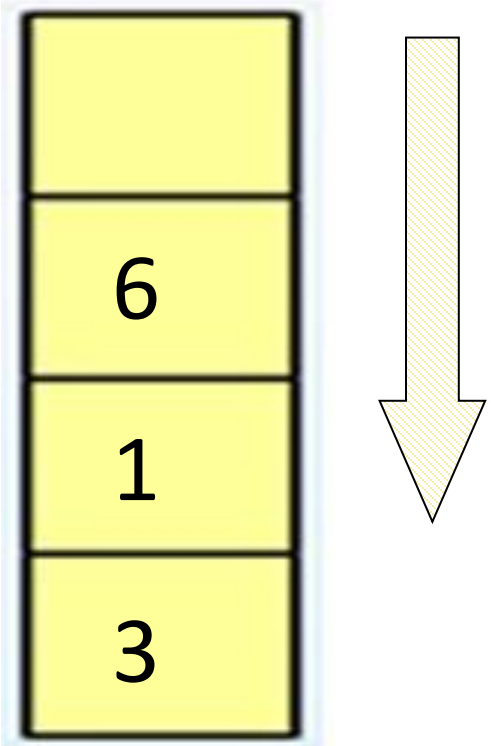
REF string: 1 5 3 1 6 2 3



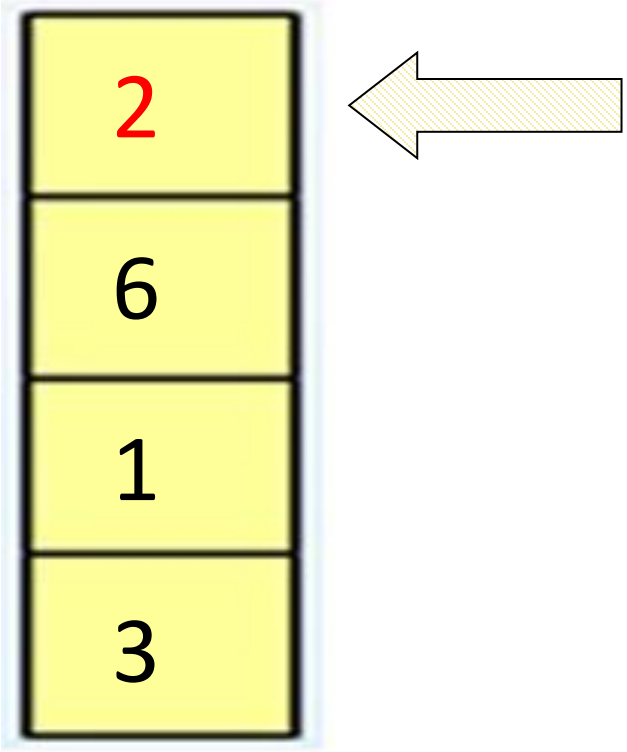
REF string: 1 5 3 1 6 2 3



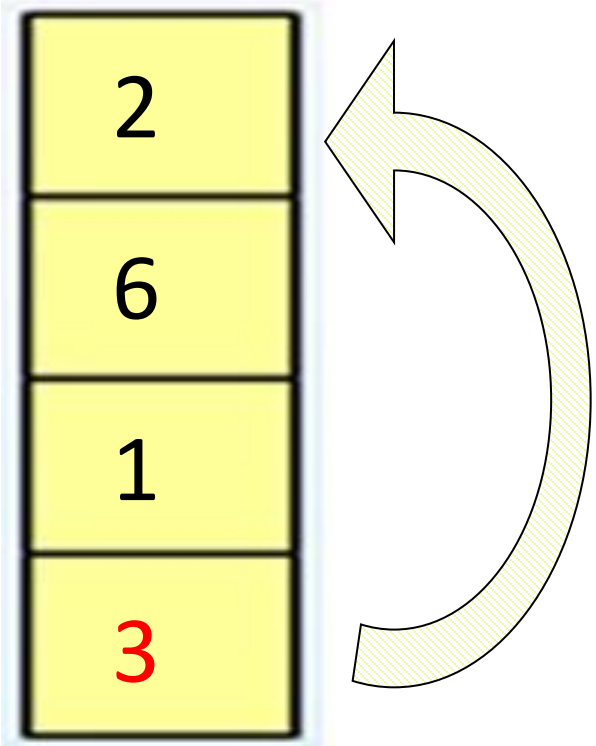
REF string: 1 5 3 1 6 2 3



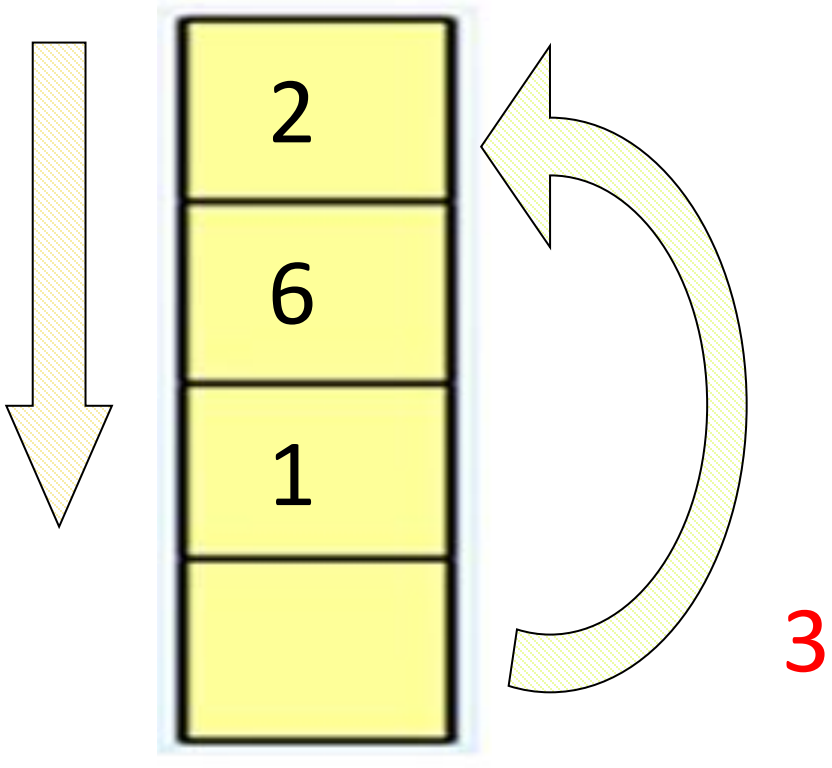
REF string: 1 5 3 1 6 2 3



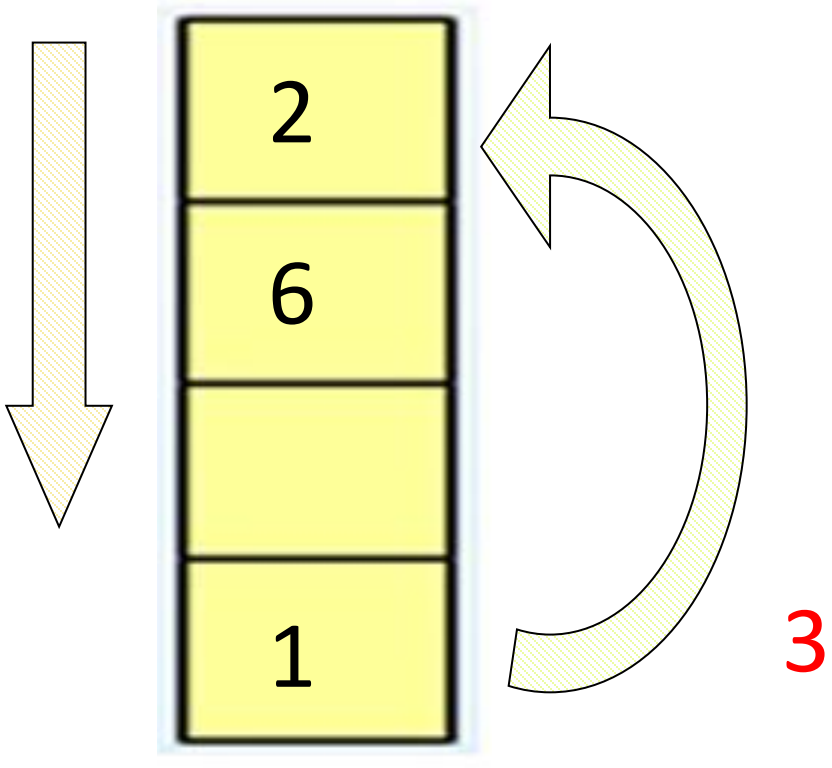
REF string: 1 5 3 1 6 2 3



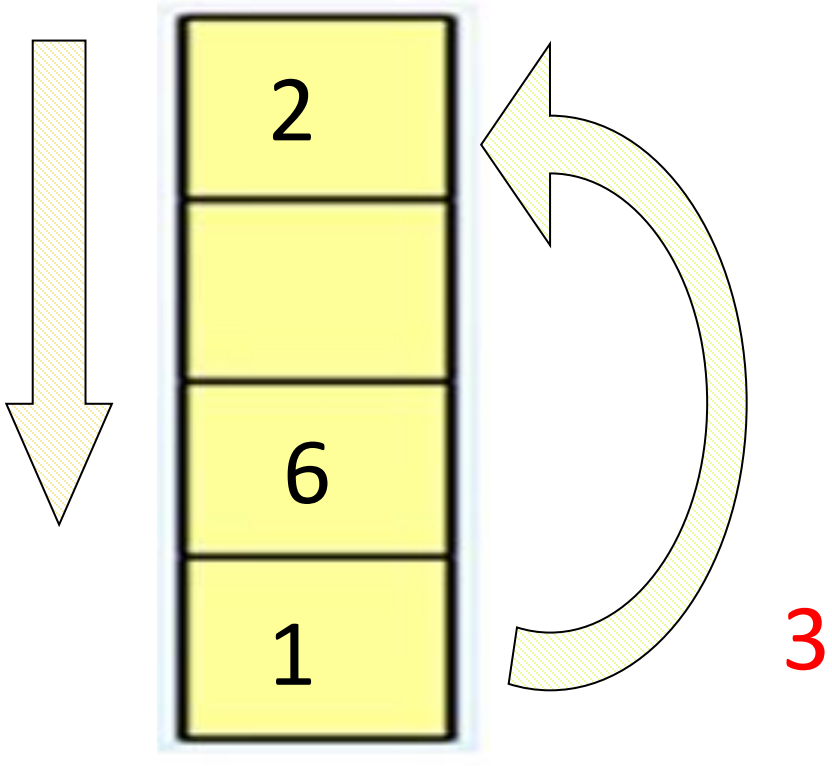
REF string: 1 5 3 1 6 2 3



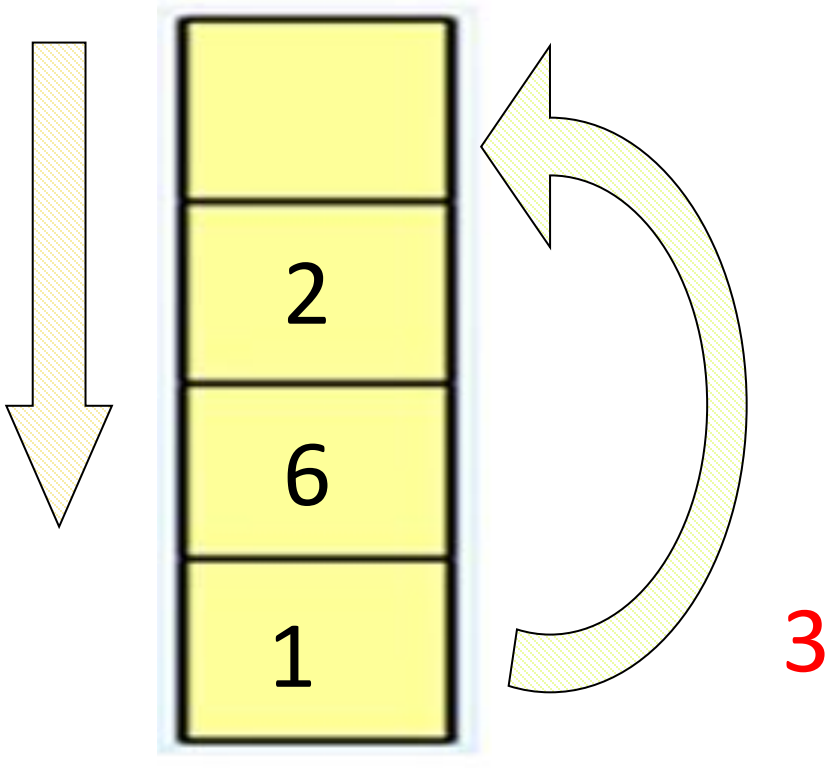
REF string: 1 5 3 1 6 2 3



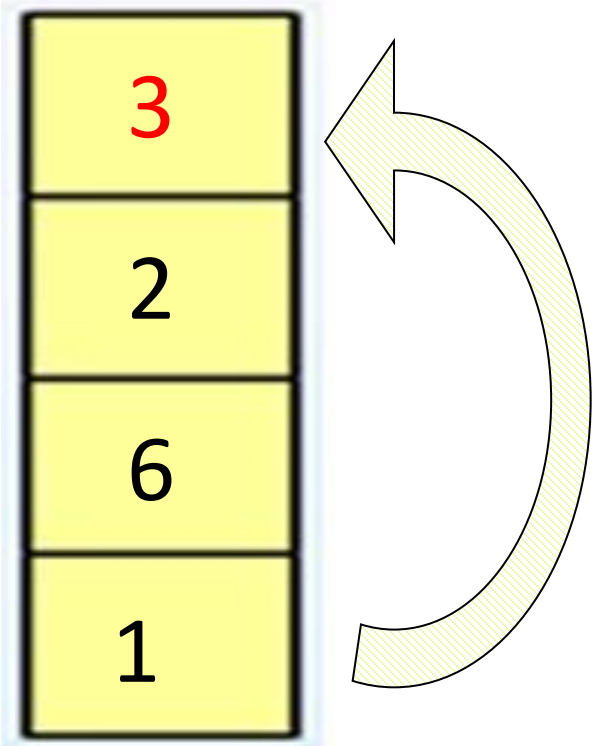
REF string: 1 5 3 1 6 2 3



REF string: 1 5 3 1 6 2 3



REF string: 1 5 3 1 6 2 3



Cache LRU Stack

- Cache LRU stack – example for **associative cache**
  - When Cache is full, the victim block is the block at the bottom of the Cache LRU stack
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

Cache LRU Stack

5
1
7
4
3
2
0
6

Address with tag value 18 is referenced

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	2	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for **associative cache**
  - When Cache is full, the victim block is the block at the bottom of the Cache LRU stack
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

Cache LRU Stack

5
1
7
4
3
2
0
6

Address with tag value 18 is referenced

Cache miss!

Block 6 is the victim!

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	2	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for **associative cache**
  - When Cache is full, the victim block is the block at the bottom of the Cache LRU stack
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

Cache LRU Stack

6
5
1
7
4
3
2
0

Address with tag value 18 is referenced

Cache miss!

Block 6 is the victim!

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	18	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for **associative cache**
  - When Cache is full, the victim block is the block at the bottom of the Cache LRU stack
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

Cache LRU Stack

6
5
1
7
4
3
2
0

Address with tag value 7 is referenced.

Cache hit!
Block 2's position in LRU stack is updated.

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	18	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for **associative cache**
  - When Cache is full, the victim block is the block at the bottom of the Cache LRU stack
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

Cache LRU Stack

2
6
5
1
7
4
3
0

Address with tag value 7 is referenced

Cache hit!
Block 2's position in LRU stack is updated.

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	18	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for ***N-way set-associative*** cache
  - When ***a set in Cache is full***, the victim block is the block at the bottom of ***the set's LRU stack***
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

LRU Stack for set 0	LRU Stack for set 1
2	5
0	6
1	7
3	4

Address with set value 0 and tag value 21 is referenced

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	2	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for ***N-way set-associative*** cache
  - When ***a set in Cache is full***, the victim block is the block at the bottom of ***the set's LRU stack***
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

LRU Stack for set 0	LRU Stack for set 1
2	5
0	6
1	7
3	4

Address with set value 0 and tag value 21 is referenced

Cache miss!

Block 3 in set 0 is the victim!

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	9	*
4	3	*
5	15	*
6	2	*
7	12	*

Cache LRU Stack

- Cache LRU stack – example for ***N-way set-associative*** cache
  - When ***a set in Cache is full***, the victim block is the block at the bottom of ***the set's LRU stack***
  - The victim block is removed from the stack and the newly used one goes on top of the stack. The rest are shifted to down.

LRU Stack for set 0	LRU Stack for set 1
3	5
2	6
0	7
1	4

Address with set value 0 and tag value 21 is referenced

Cache miss!

Block 3 in set 0 is the victim!

Cache

Block ID	Tag Value	Data
0	10	*
1	0	*
2	7	*
3	21	*
4	3	*
5	15	*
6	2	*
7	12	*