CS 3502
Operating Systems

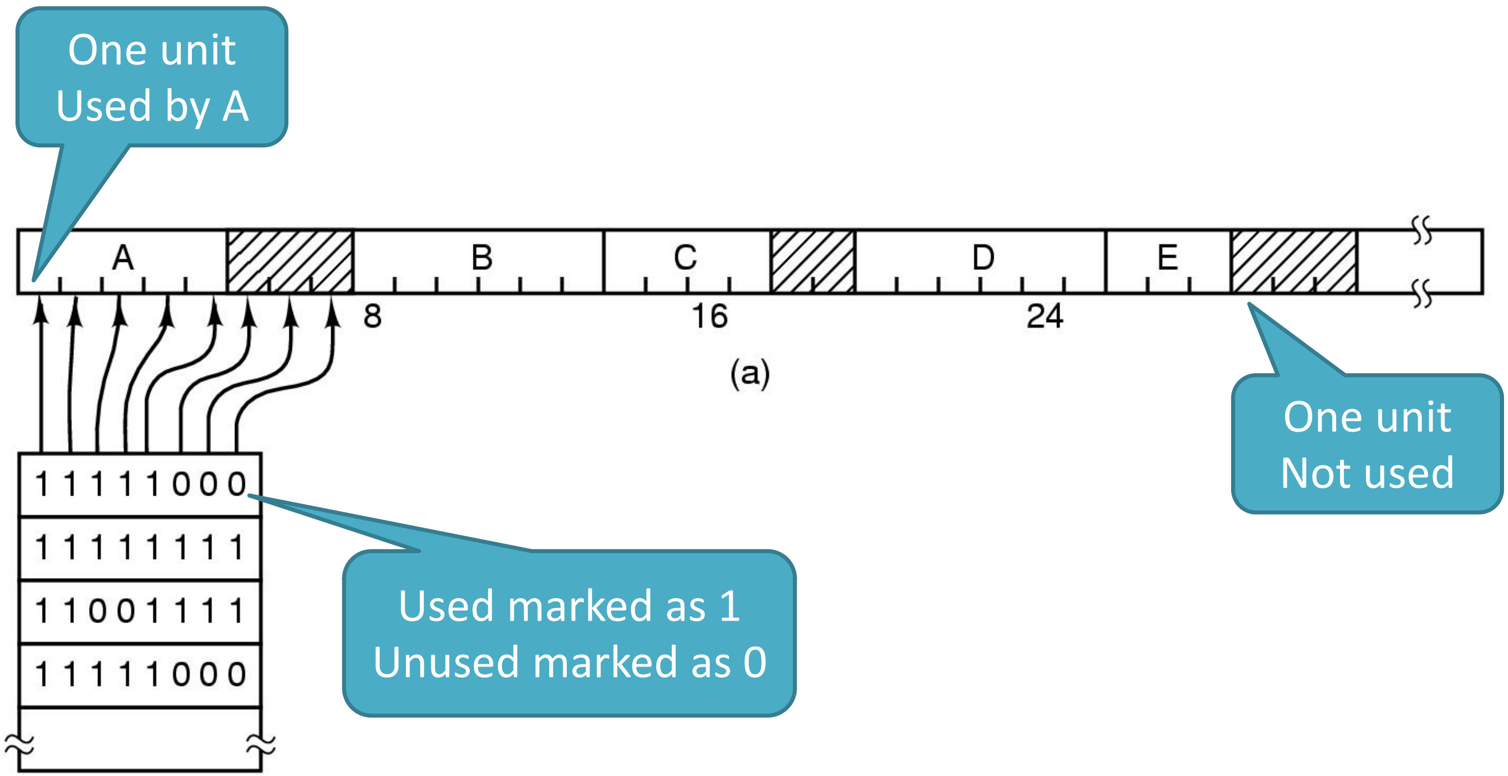
Page replacement

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/

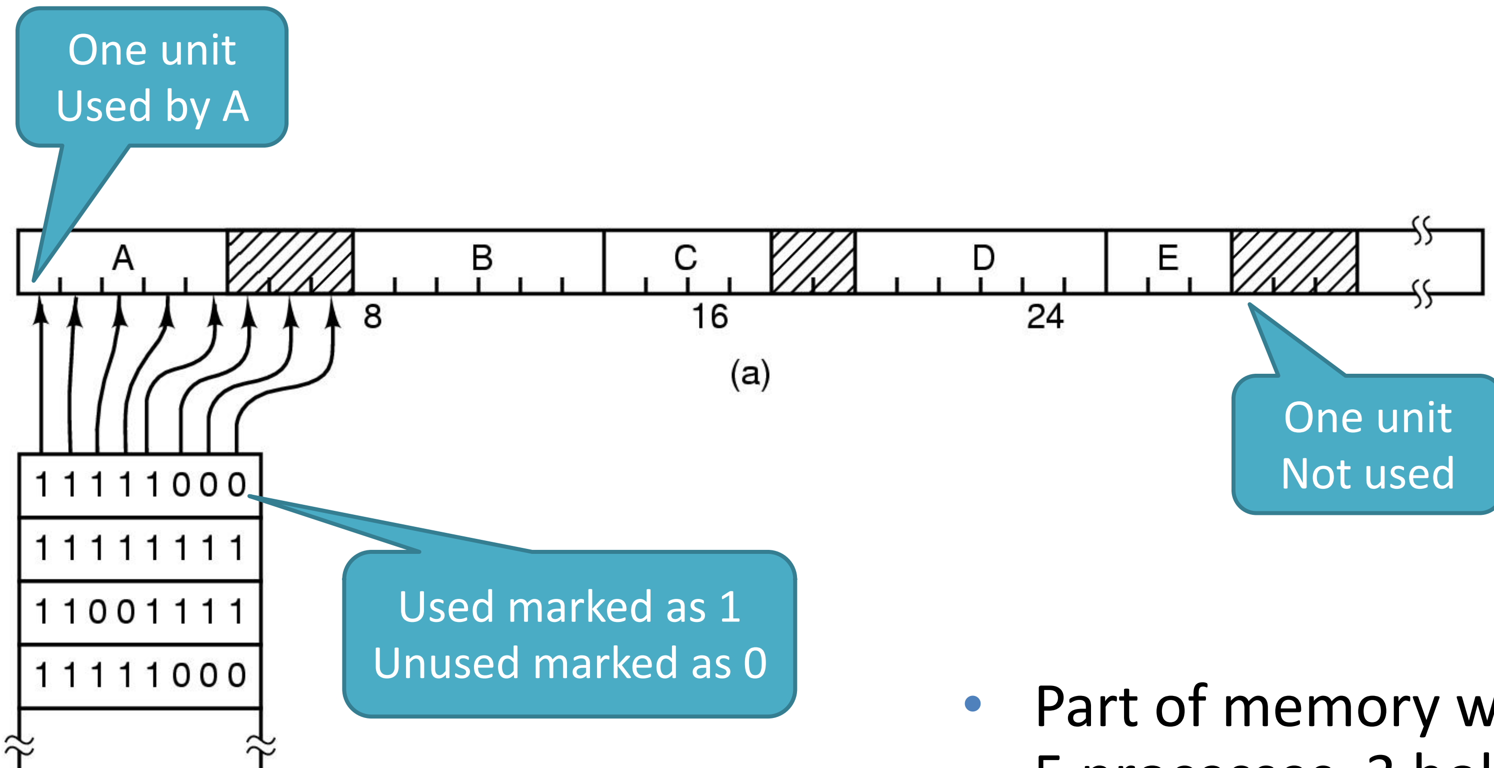
Outline

- Memory management data structure
  - Bit maps vs. Linked lists
- Page replacement algorithm
  - OPR, FIFO, LRU
  - NFU, NRU
  - Second chance, Clock
  - Aging

Memory Management with Bit Maps

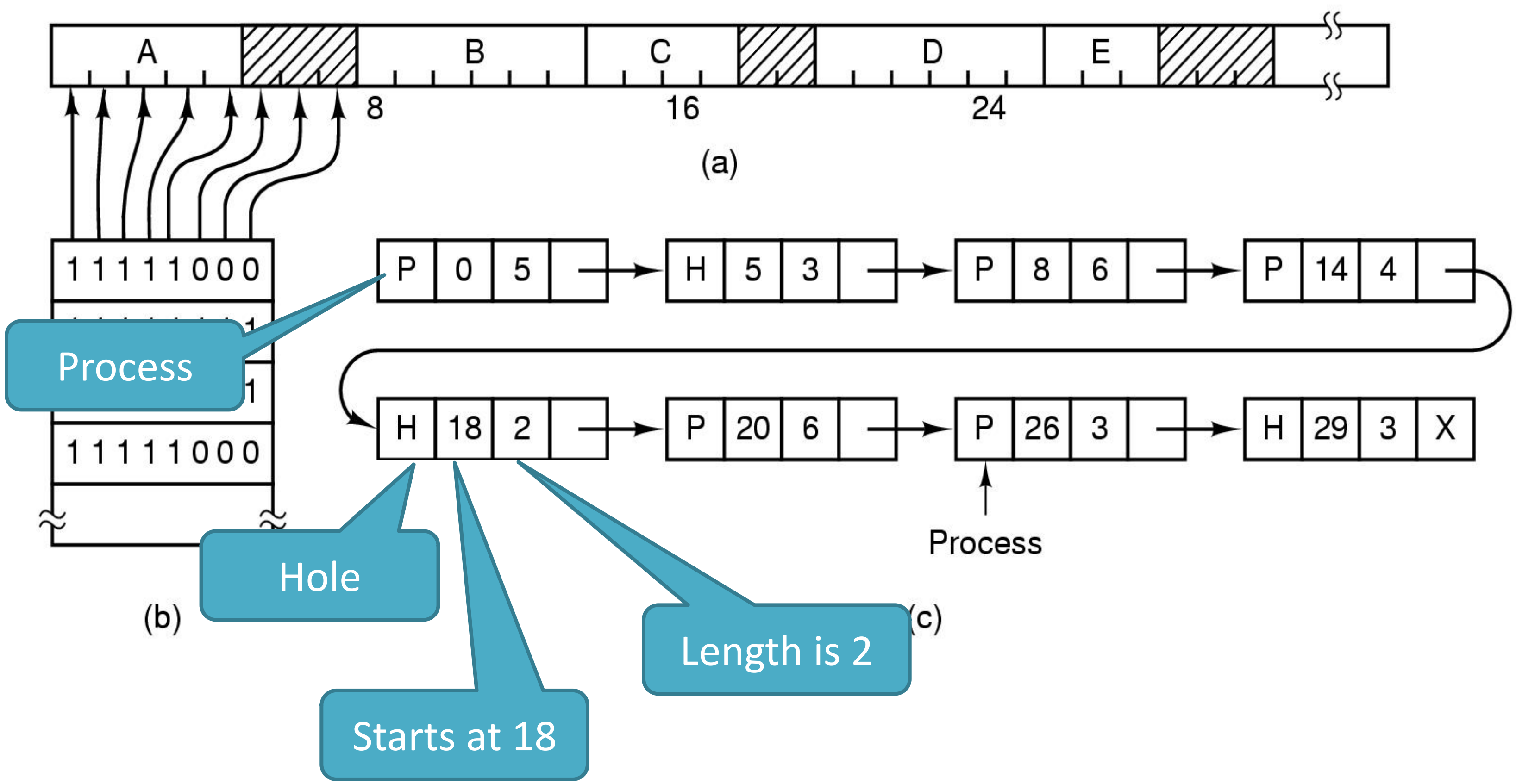


Memory Management with Bit Maps



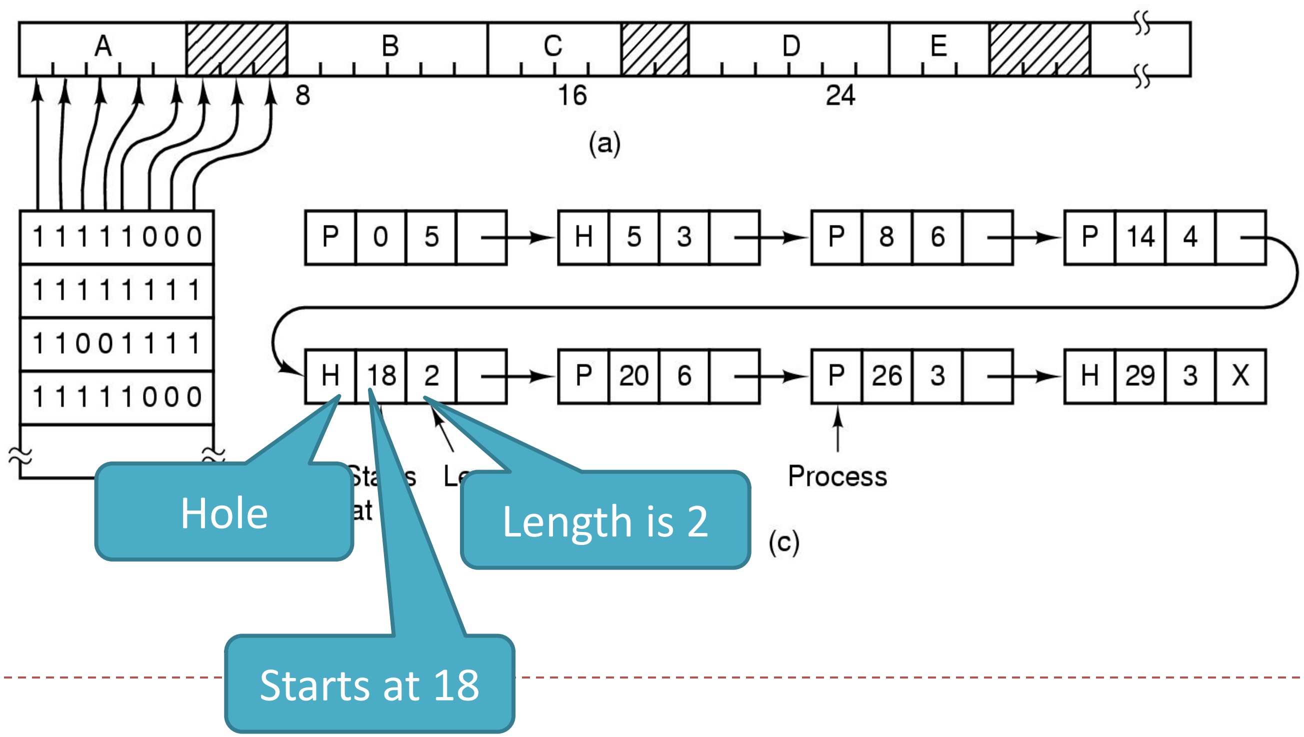
- Part of memory with 5 processes, 3 holes

Memory Management with Linked Lists



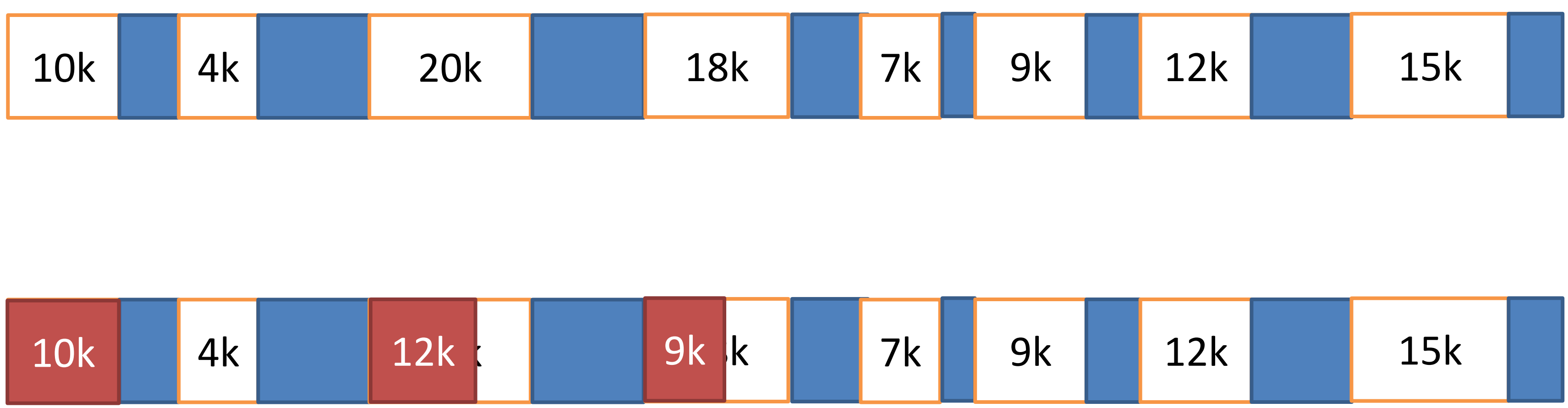
Memory Management with Linked Lists: Allocation

- How to allocate memory for a newly created process (or swapping) ?
  - First fit: allocate the first hole that is big enough
    - Search hole nodes, if large enough, return the start address of first hole node



Question: First fit

- Consider a system in which memory consists of the following hole sizes in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K and 15K.
- Which hole is taken for successive segment requests of (a) 12K, (b) 10K, (c) 9K for first fit?



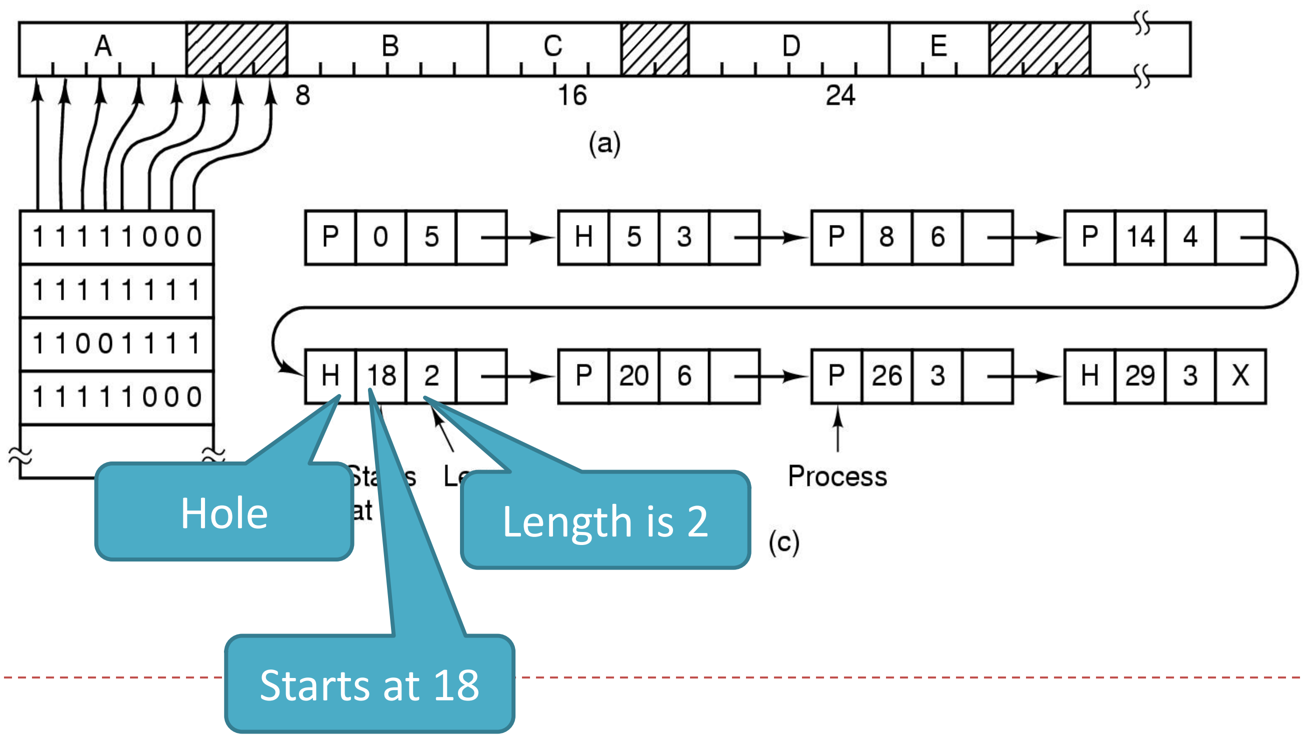
Advantage & Disadvantage of First Fit

- Advantage:
  - Simple
  - Fast match

- Disadvantage:
  - Fragmentation

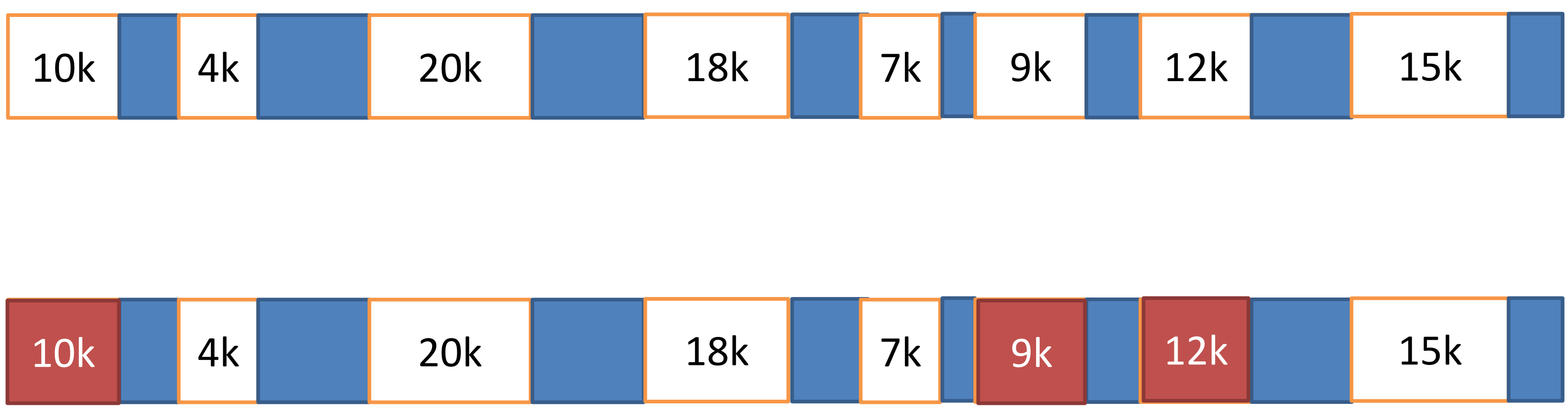
Memory Management with Linked Lists: Allocation

- How to allocate memory for a newly created process (or swapping) ?
  - Best fit: allocate the smallest hole that is big
    - Search every hole node, return start address of the smallest hole that fits



Question: Best fit

- Consider a system in which memory consists of the following hole sizes in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K and 15K.
- Which hole is taken for successive segment requests of (a) 12K, (b) 10K, (c) 9K for best fit?

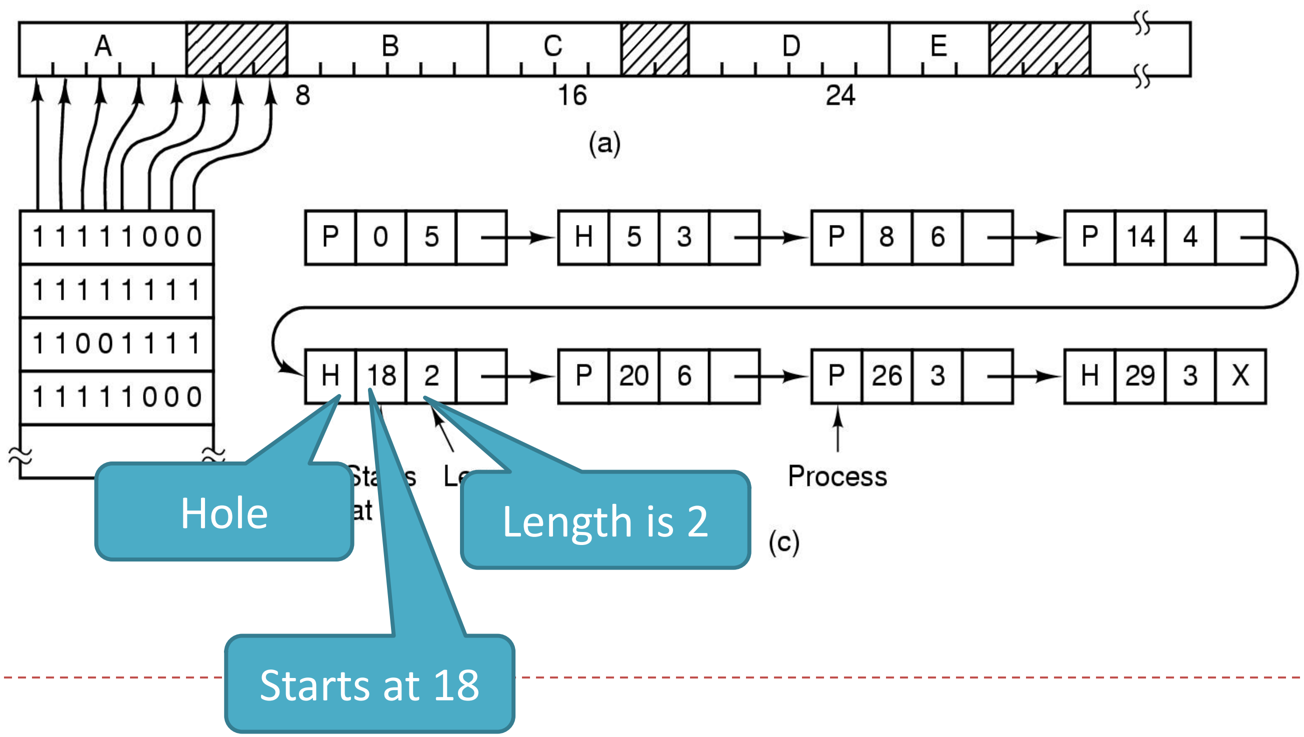


Advantage & Disadvantage of Best Fit

- Advantage:
  - Works well when most allocations are of small size
  - Least fragmentation
  - Relatively simple
- Disadvantage:
  - Slow allocation

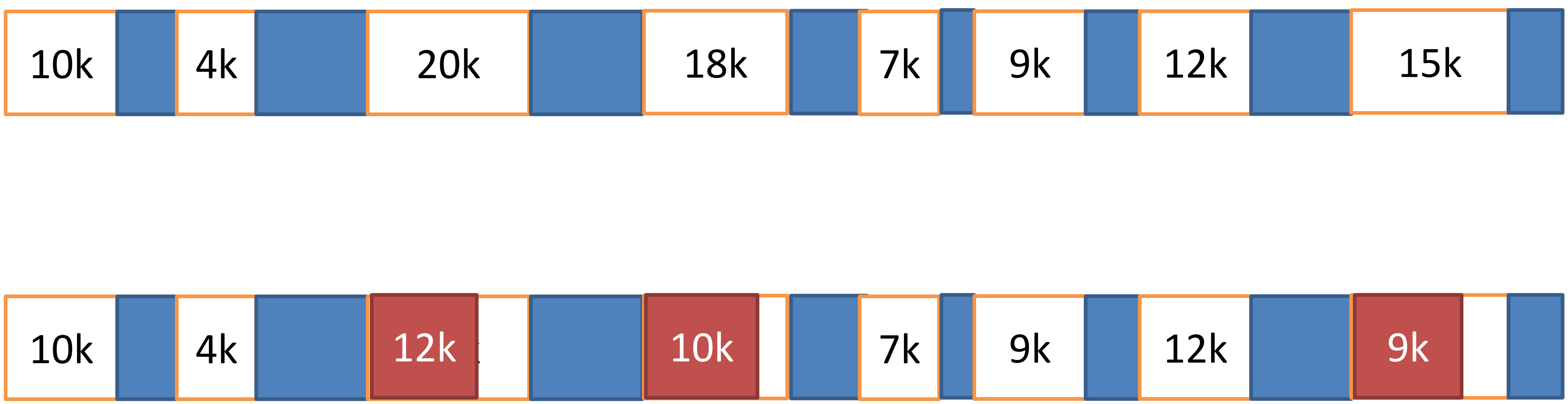
Memory Management with Linked Lists: Allocation

- How to allocate memory for a newly created process (or swapping) ?
  - Worst fit: allocate the largest hole
    - Search every hole node, return start address of the largest hole that fits



Question: Worst fit

- Consider a system in which memory consists of the following hole sizes in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K and 15K.
- Which hole is taken for successive segment requests of (a) 12K, (b) 10K, (c) 9K for worst fit?



Advantage & Disadvantage of Worst Fit

- Advantage:
  - Works well if allocations are of medium sizes

- Disadvantage:
  - Fragmentation
  - Tends to break large free blocks into useless small ones

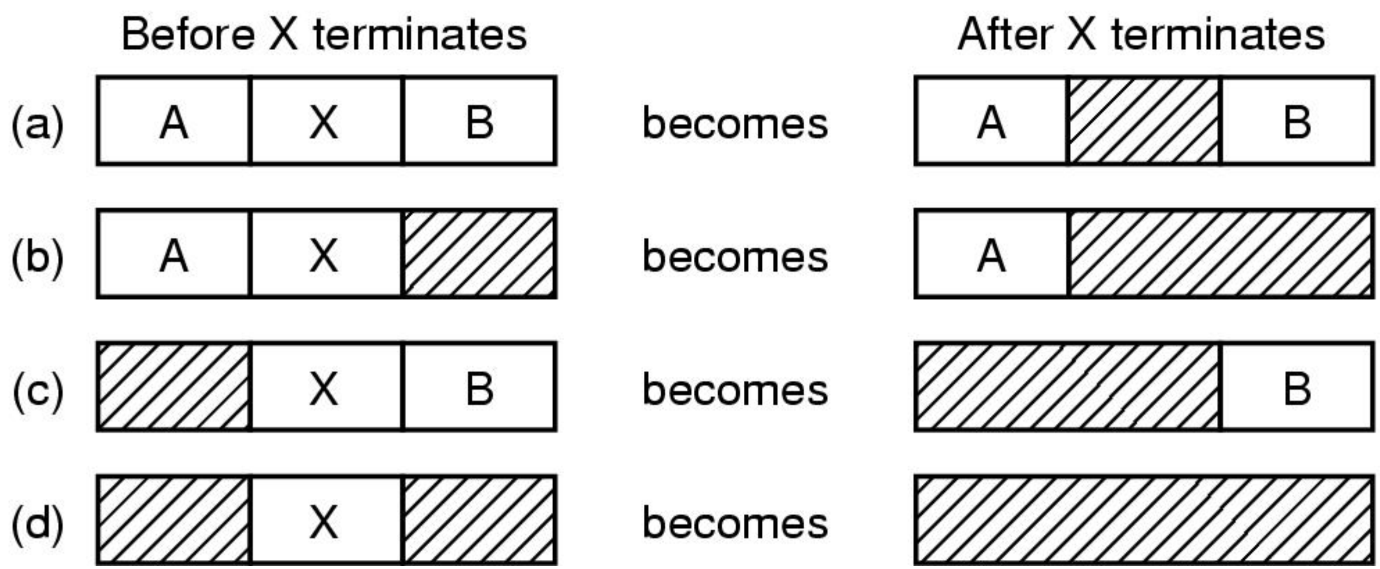
Memory Management with Linked Lists: Allocation

- How to allocate memory for a newly created process (or swapping) ?
  - First fit: allocate the first hole that is big enough
  - Best fit: allocate the smallest hole that is big
  - Worst fit: allocate the largest hole

- Which strategy is the best?
  - The first fit works faster
  - The best fit has the least memory fragments

Memory Management with Linked Lists: Deallocation

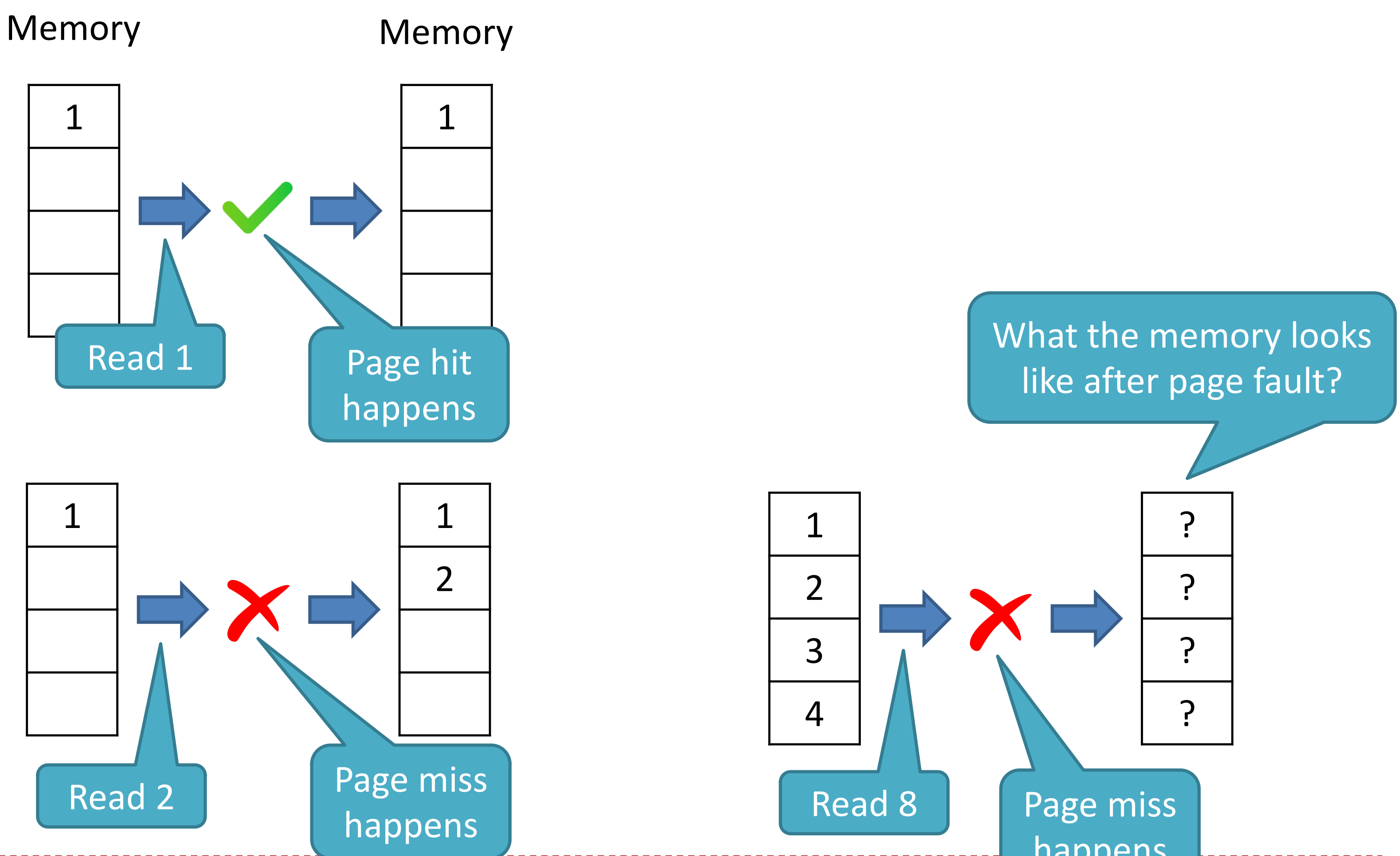
- De-allocating memory is to update the list
- Four neighbor combinations for the terminating process X



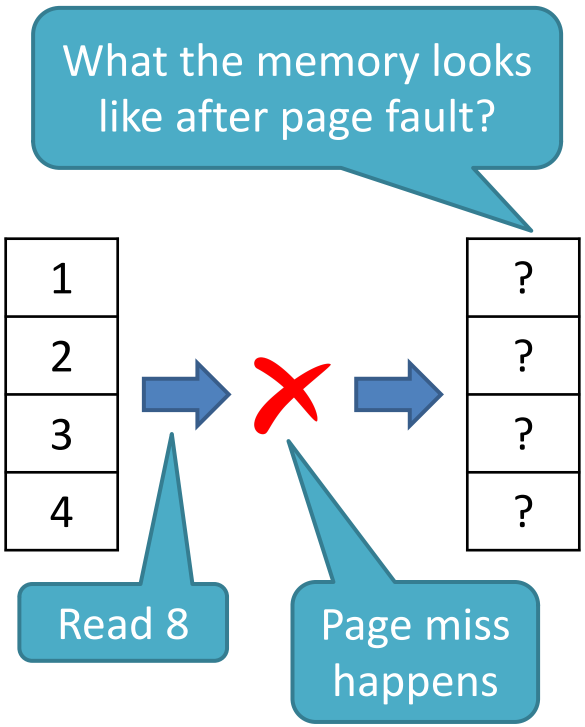
Outline

- Memory management data structure
  - Bit maps vs. Linked lists
- Page replacement algorithm
  - OPR, FIFO, LRU
  - NFU, NRU
  - Second chance, Clock
  - Aging

Page Hit and Page Miss(fault)



Page Replacement Algorithms



- After page fault, which page is removed
- Better not to choose an often-used page or pages which will be used in near future (God view)
- Low page-fault rate is the metrics for good algorithms

Optimal Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5

6 page faults

Optimal Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5

6 page faults

We replace 4 with 5 because in the following three reads: 1,2,3 will be visited

Optimal Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5
✗	✗	✗	✗			✗				✗	

6 page faults

We replace 1 with 4 because 1 is the oldest in memory while 1 is not used in future

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future, keep page needed <u>at the nearest future (God view)</u>

- Advantage
  - Good as a benchmark for comparison

- Disadvantage
  - Optimal but unrealizable
  - OS must know when each of the pages will be referenced next

FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in the order they came into memory
- Idea:
  - Page at beginning of list replaced (the oldest one)

FIFO Page Replacement Algorithm

Reference string:

1 2 3 4 1 2 5 1 2 3 4 5

1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3

10 page faults

FIFO Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3

10 page faults

We replace 1 because 1 is the oldest in memory

FIFO Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
✗	✗	✗	✗			✗	✗	✗	✗	✗	✗

10 page faults

We replace 2 because 2 is the oldest in memory

FIFO Page Replacement Algorithm

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3

10 page faults

We replace 3 because 3 is the oldest in memory

FIFO Page Replacement Algorithm

- Idea:
  - Page at beginning of list replaced (the oldest one)
- Advantage
  - Design and implementation is simple
- Disadvantage
  - Page in memory the longest (oldest) may be often used, something not optimal

Least Recently Used (LRU)

- Idea:
  - Assume pages used recently will be used again soon
  - throw out page that has been least used recently
- Design: keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference
    - finding, removing, and moving it to the front

Least Recently Used (LRU)

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

8 page faults

Least Recently Used (LRU)

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3
✗	✗	✗	✗			✗			✗	✗	✗

8 page faults

We replace 3 because we visited 2, 1, 4 recently and 3 is the least recently used

Least Recently Used (LRU)

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

8 page faults

We replace 4 because we visited 2, 1, 5 recently and 4 is the least recently used

Least Recently Used (LRU)

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

8 page faults

Least Recently Used (LRU)

Reference string:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3
✗	✗	✗	✗			✗			✗	✗	✗

8 page faults

Not Recently Used (NRU)

- Each page has R bit (referenced) and M bit (modified)
  - bits are ***set*** when page is referenced and modified
  - OS ***clears*** R bits periodically (by clock interrupts)

R	M	item
0	1	A
1	0	B
1	1	C
0	0	D

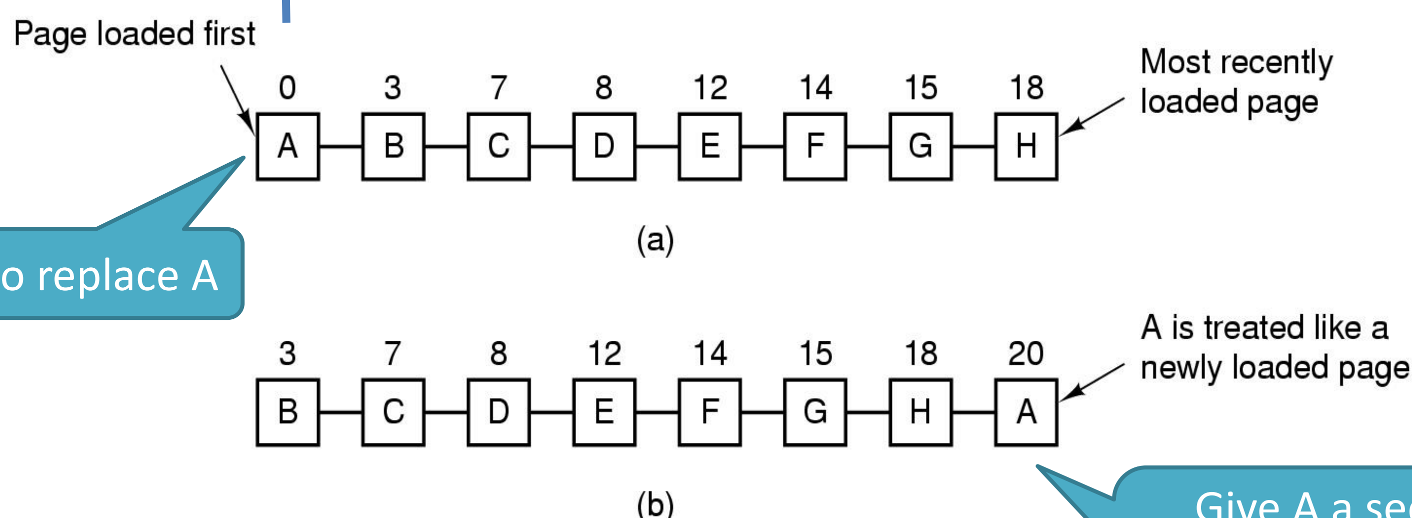
- Pages are classified
  - not referenced, not modified (0 class)
  - not referenced, modified (1 class)
  - referenced, not modified (2 class)
  - referenced, modified (3 class)

- NRU removes a page at random
  - From the lowest numbered non-empty class (from 0 class to 3 class)

Second Chance Page Replacement Algorithm

R	item
1	A
0	B
1	C
0	D

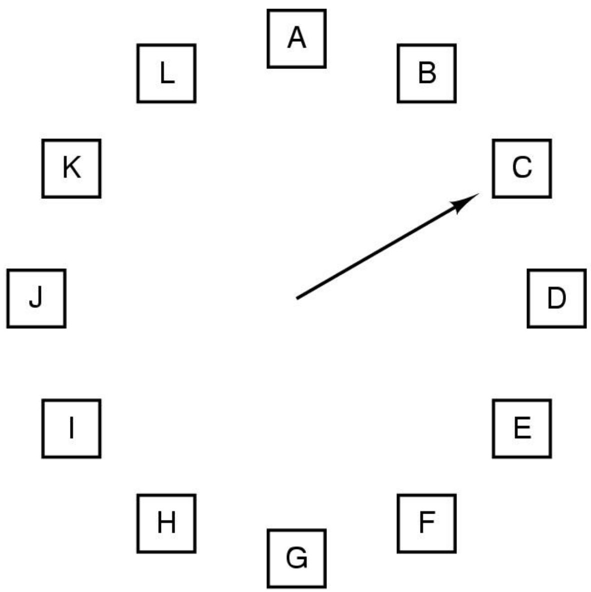
- Every page has a reference bit
- Check the R bit of oldest page:
  - if it is 0, replace it;
  - if it is 1, set the R bit to 0 and then put the page to the tail of linked list.



Try to replace A

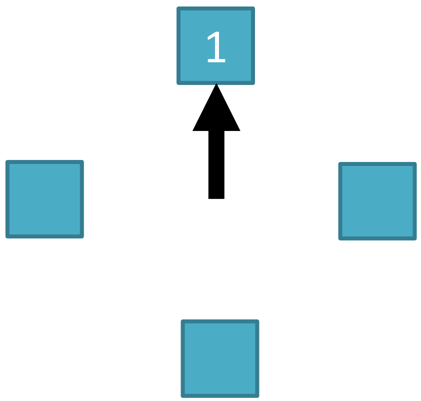
Give A a second chance: set R bit as 0 and put it to the tail

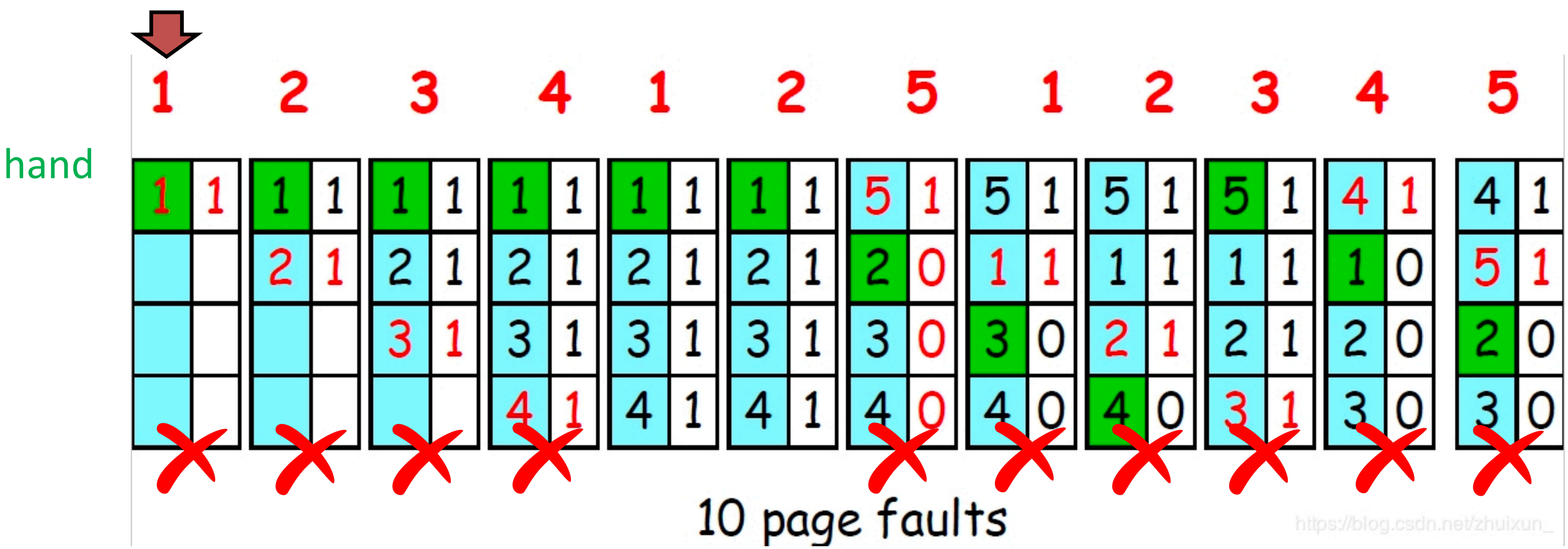
The Clock Page Replacement Algorithm



- Similar as Second Chance
- Algorithm:
  - When a page fault occurs, the page the hand is pointing to is inspected:
    - If the R bit is 0,
      - replace the page, advance hand;
    - If the R bit is 1,
      - clear R bit,
      - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
      - advance hand

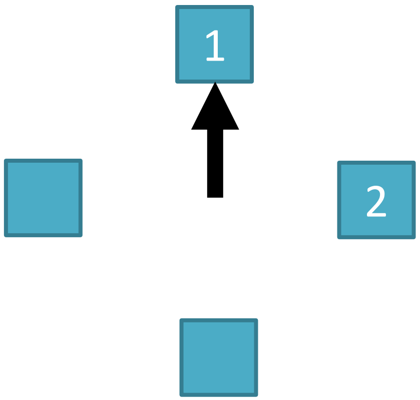
The Clock Page Replacemer



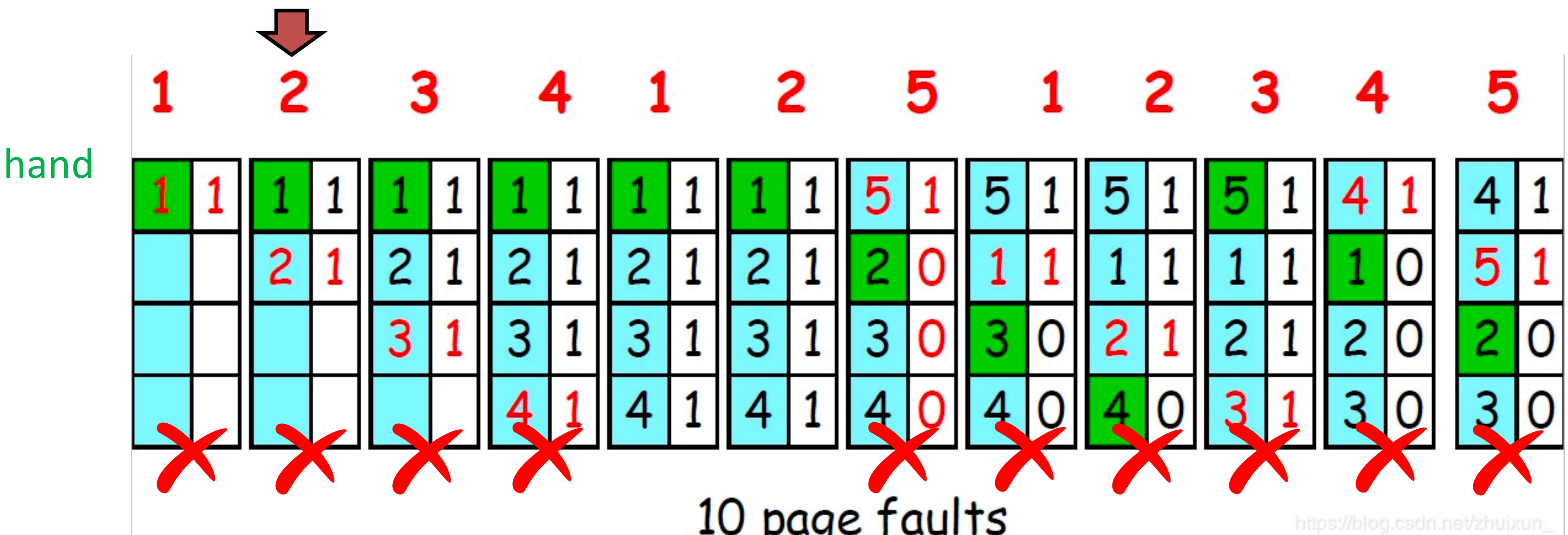


- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

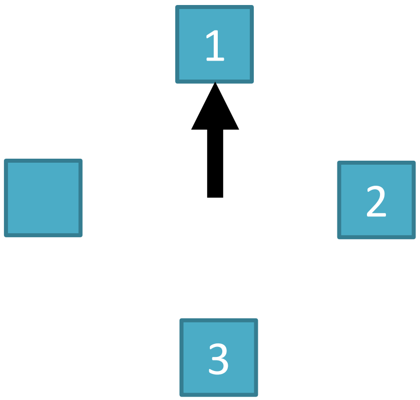
The Clock Page Replacemer



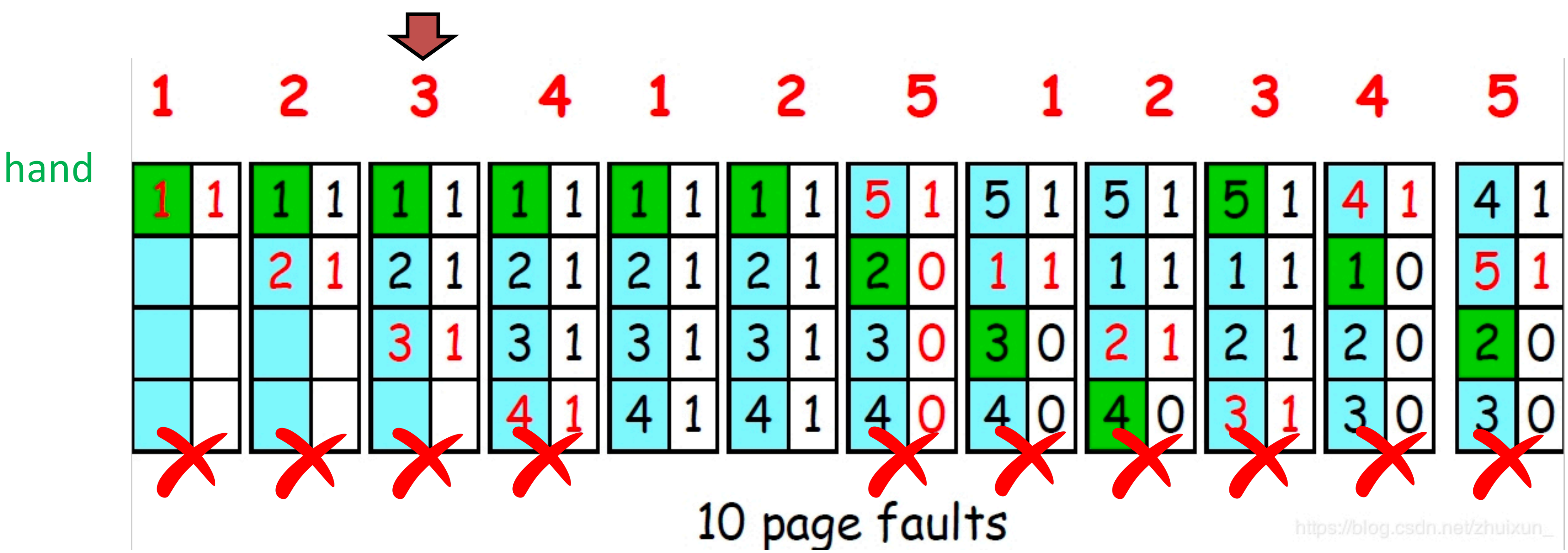
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



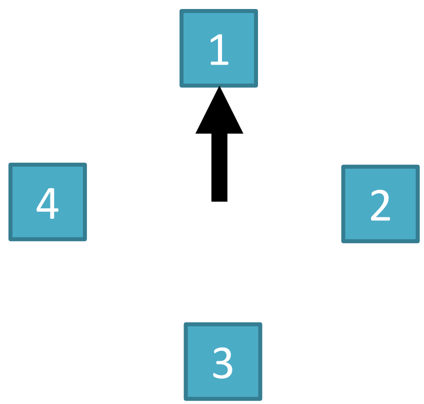
The Clock Page Replacemer



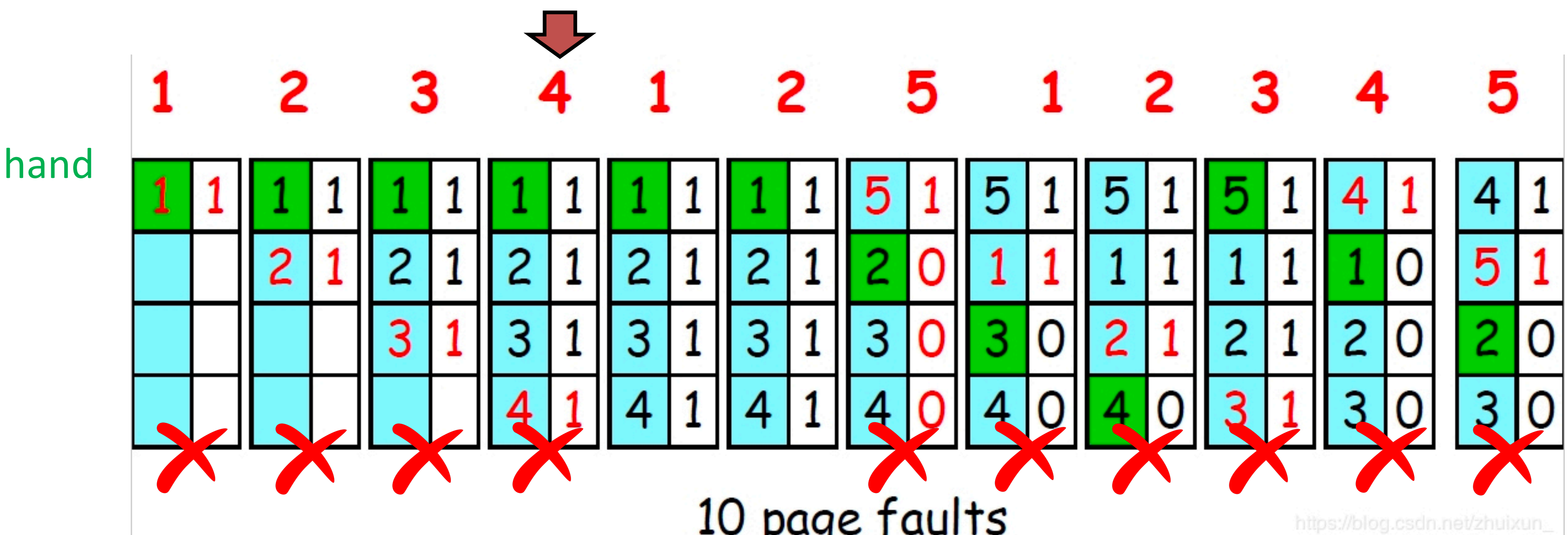
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



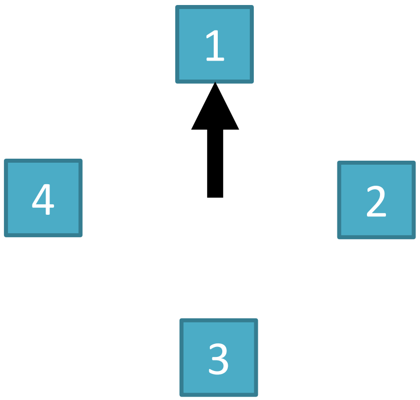
The Clock Page Replacemer



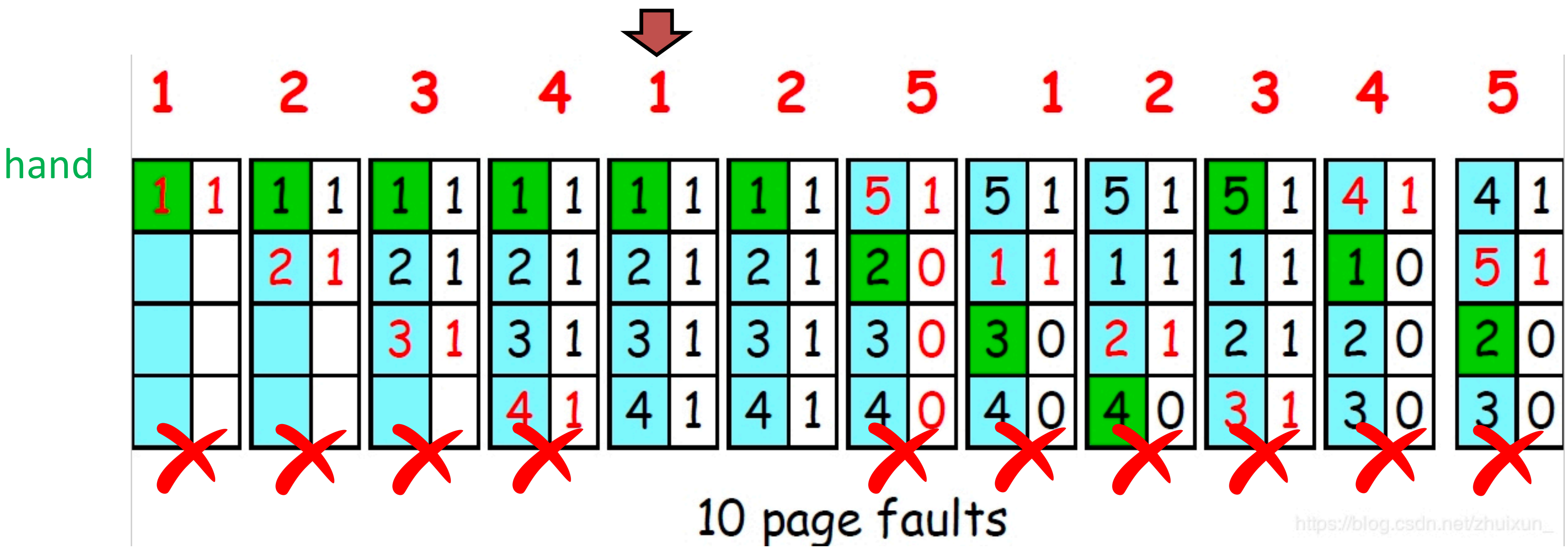
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



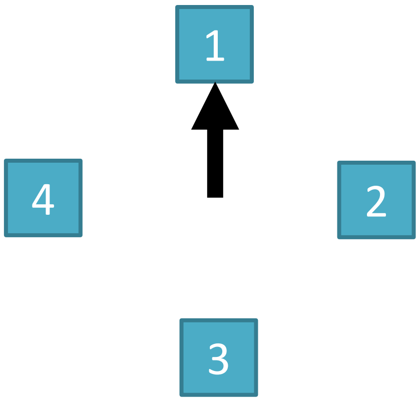
The Clock Page Replacemer



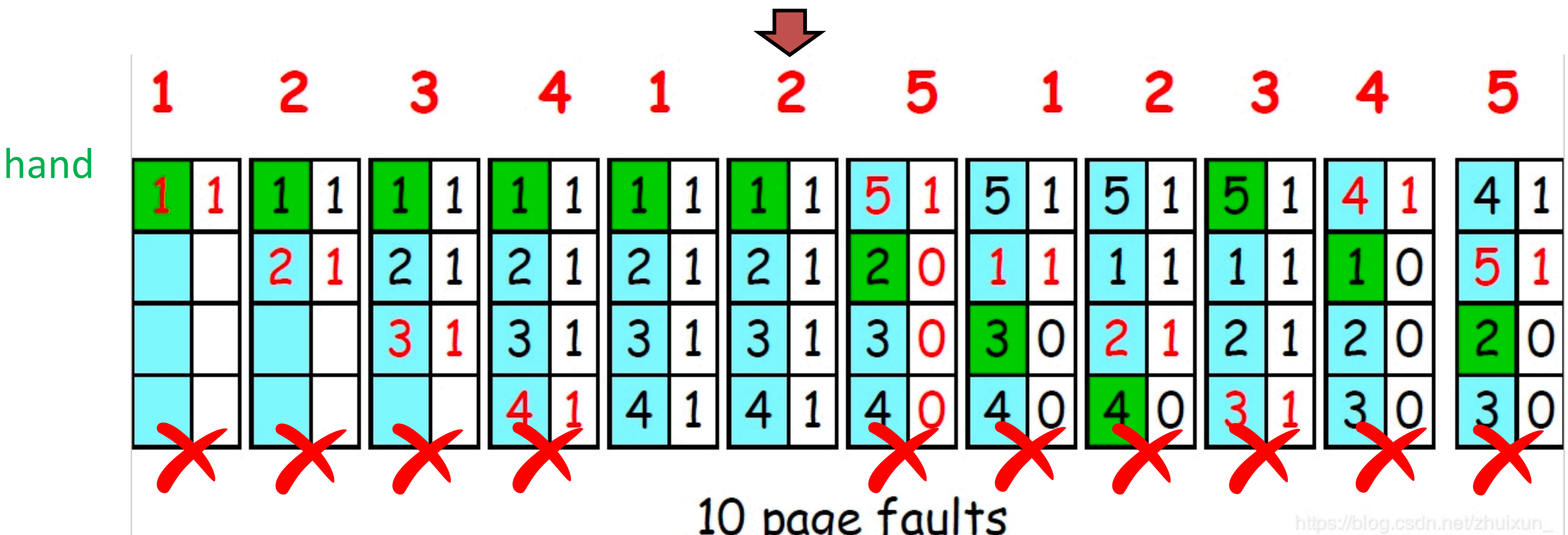
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



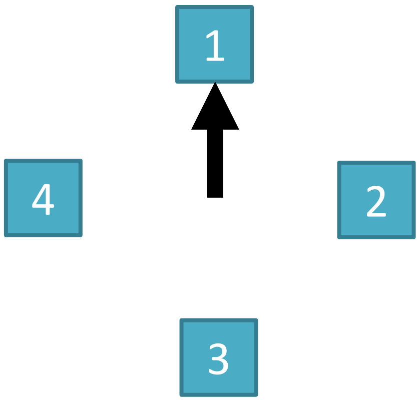
The Clock Page Replacemer



- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

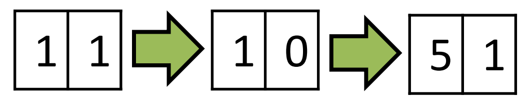


The Clock Page Replacemer



Page fault happens! Check the element which the hand points to (1);

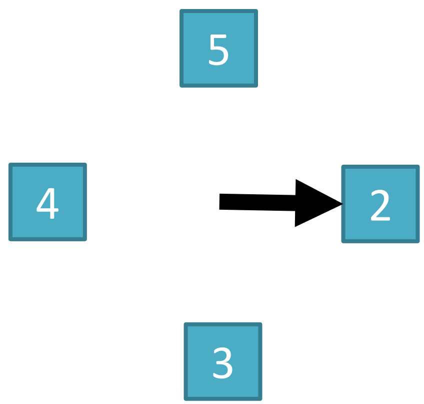
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

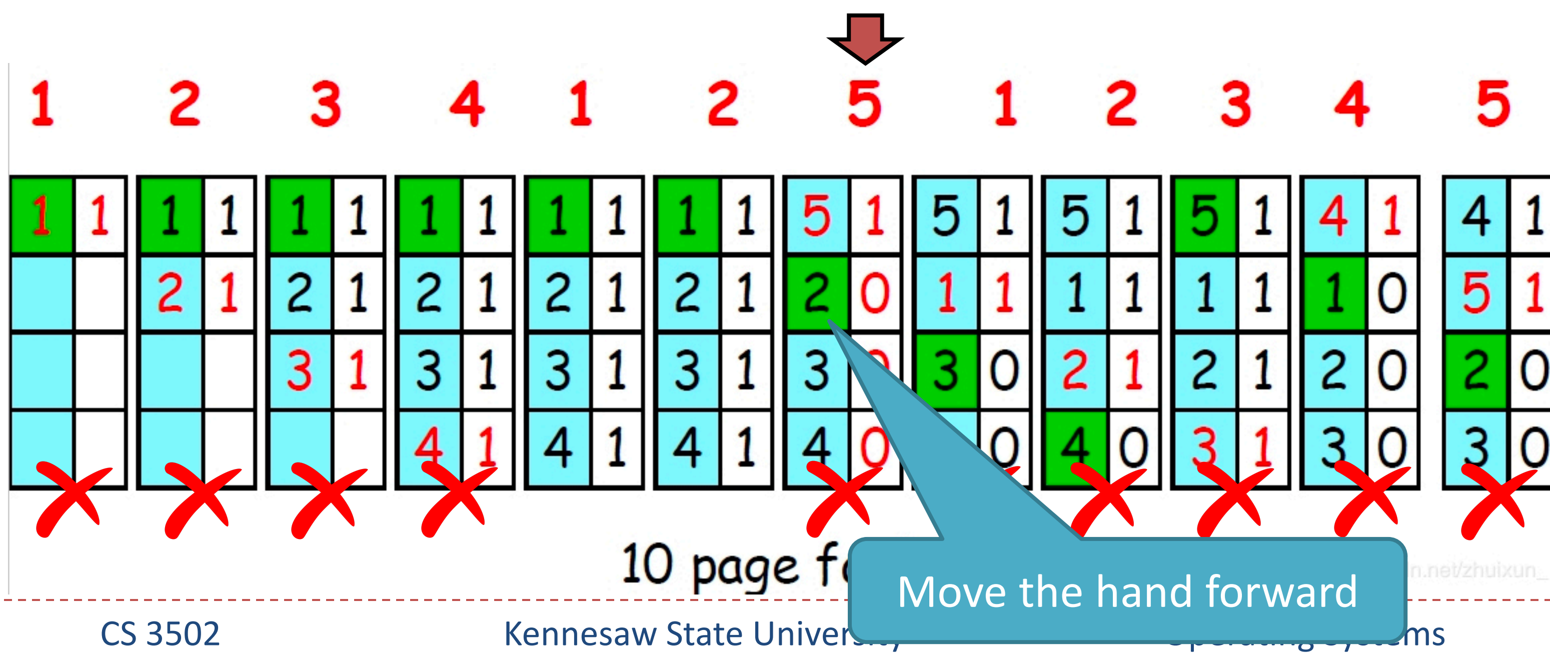


R bits all change to 0 because all R bits are 1s

The Clock Page Replacemen

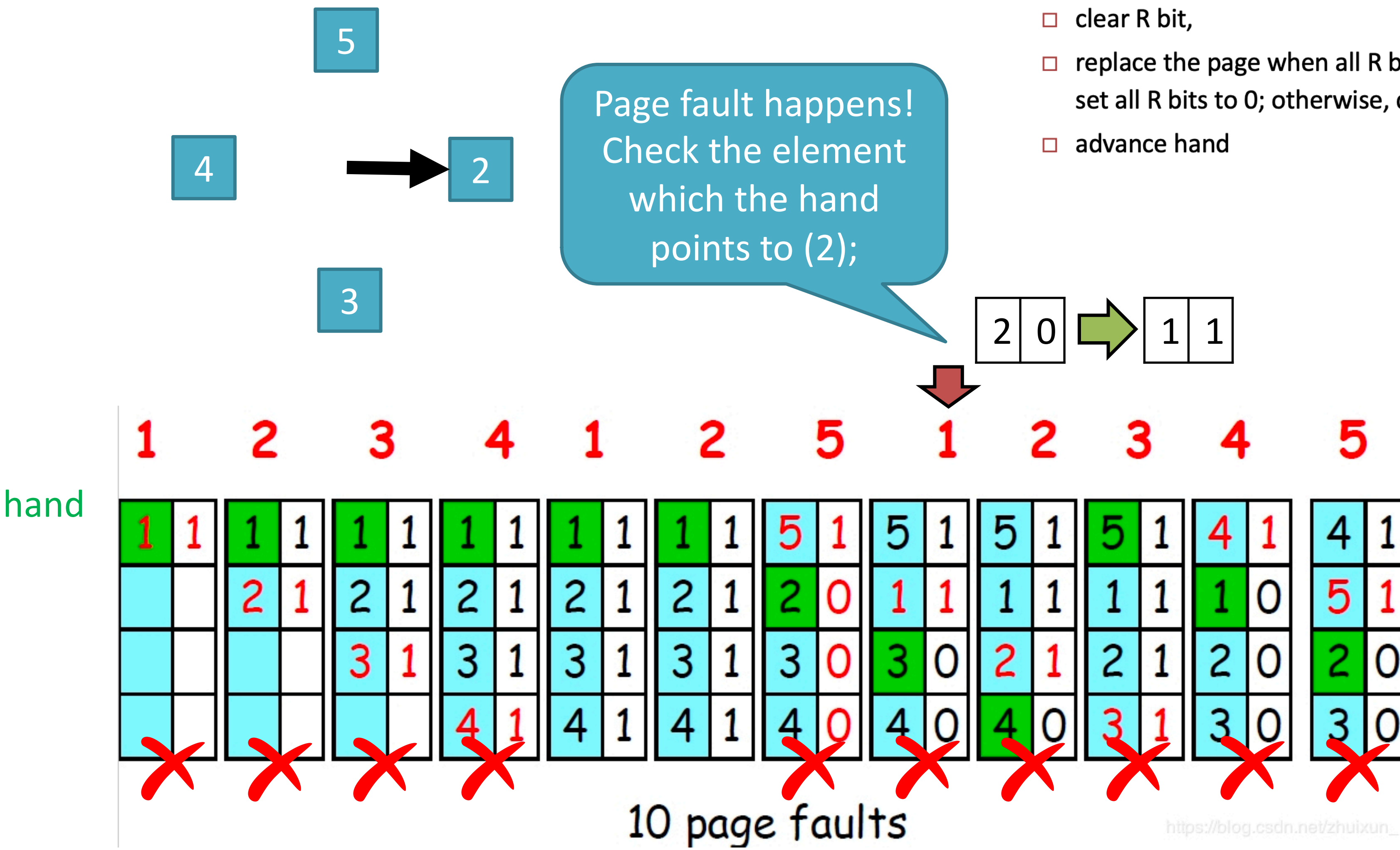
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



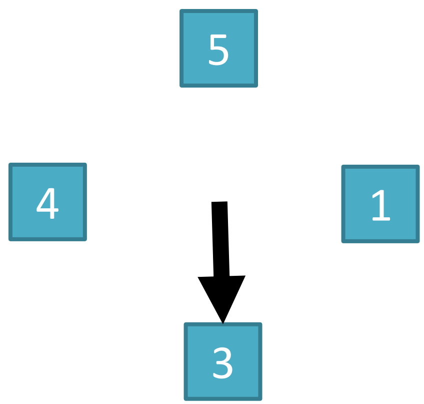


The Clock Page Replacemer

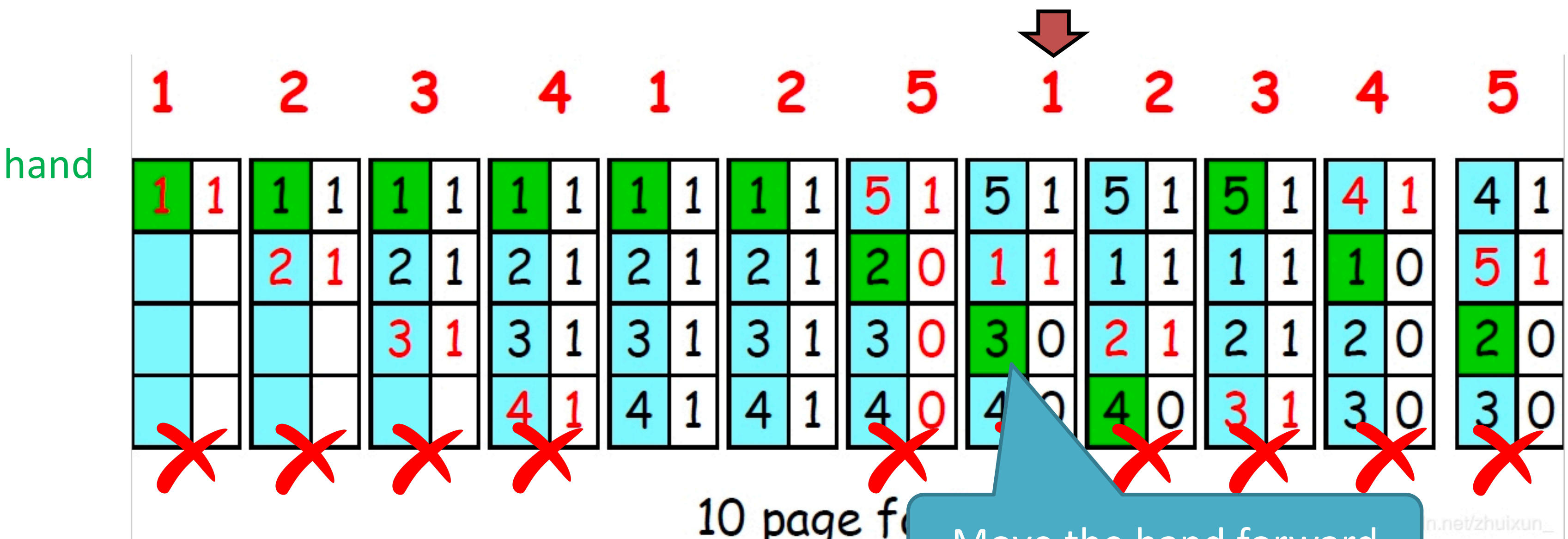
- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand



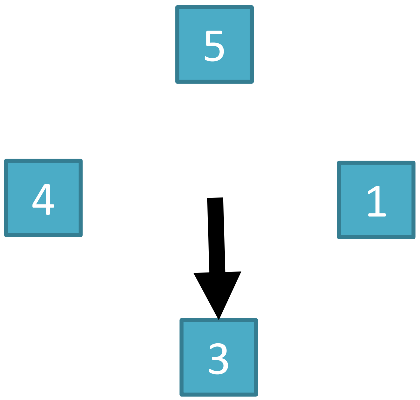
The Clock Page Replacemer



- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

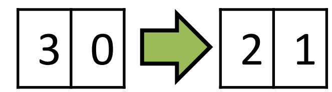


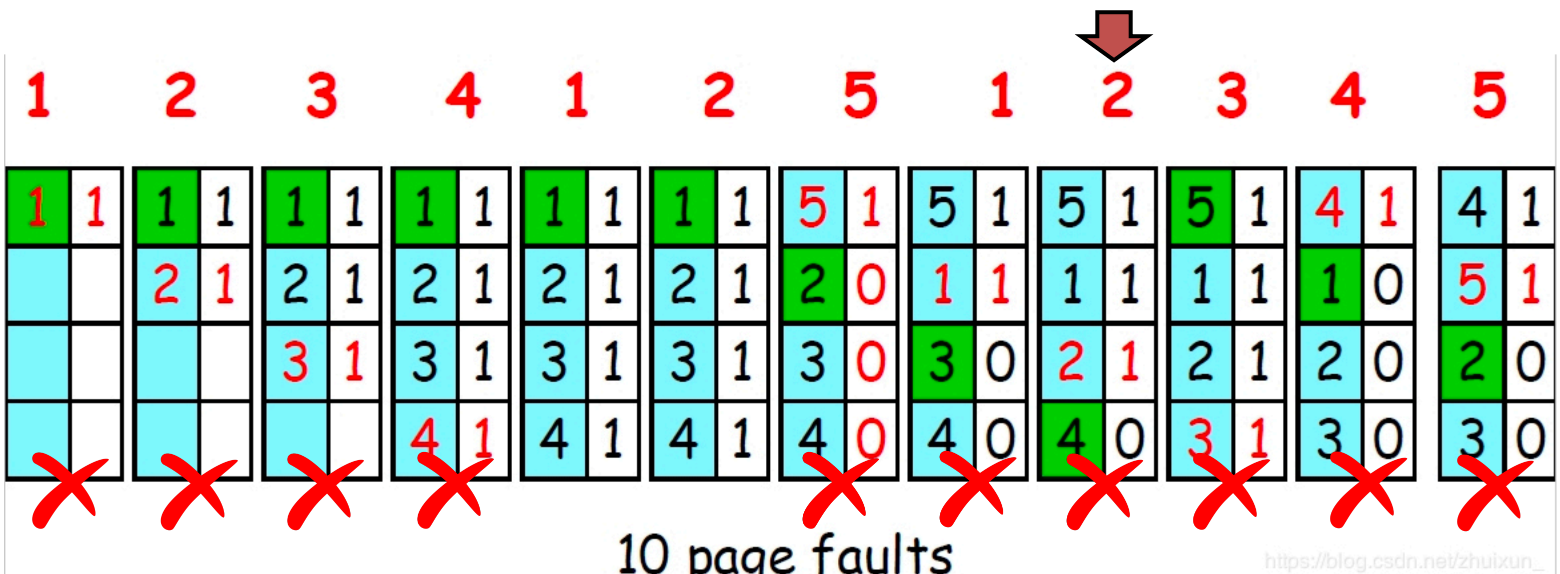
The Clock Page Replacement Algorithm



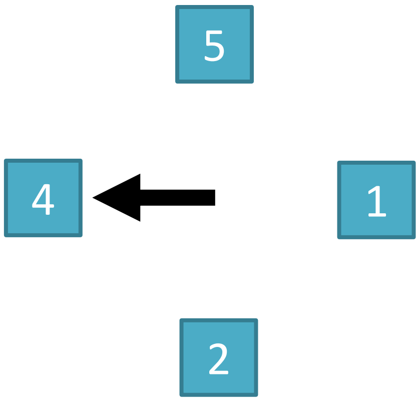
Page fault happens! Check the element which the hand points to (3);

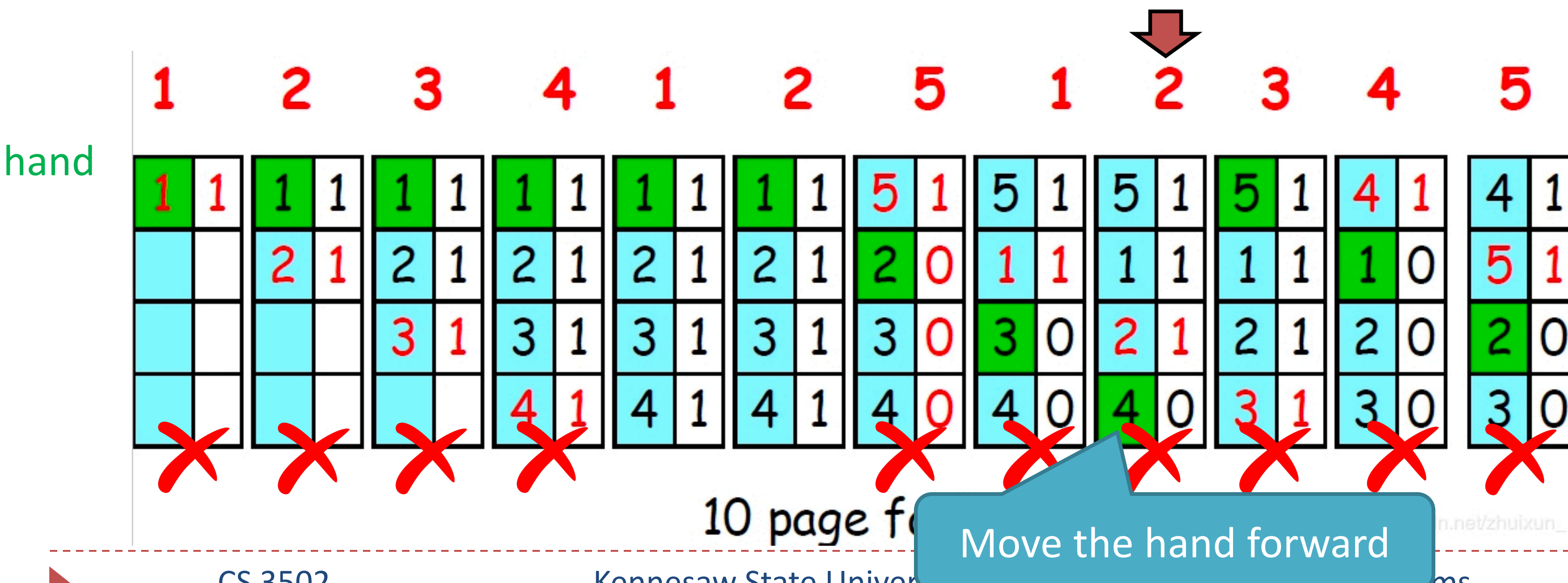
Check R bits: it is 0, replace the page and move hand forward



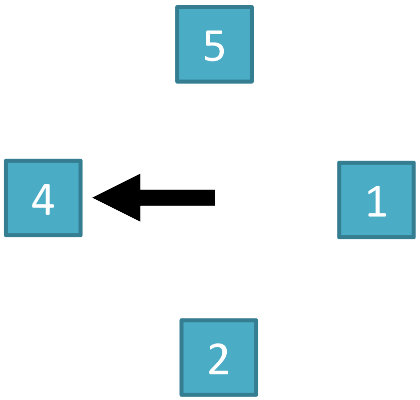


The Clock Page Replacement Algorithm



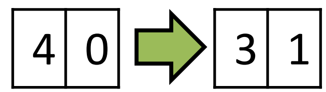


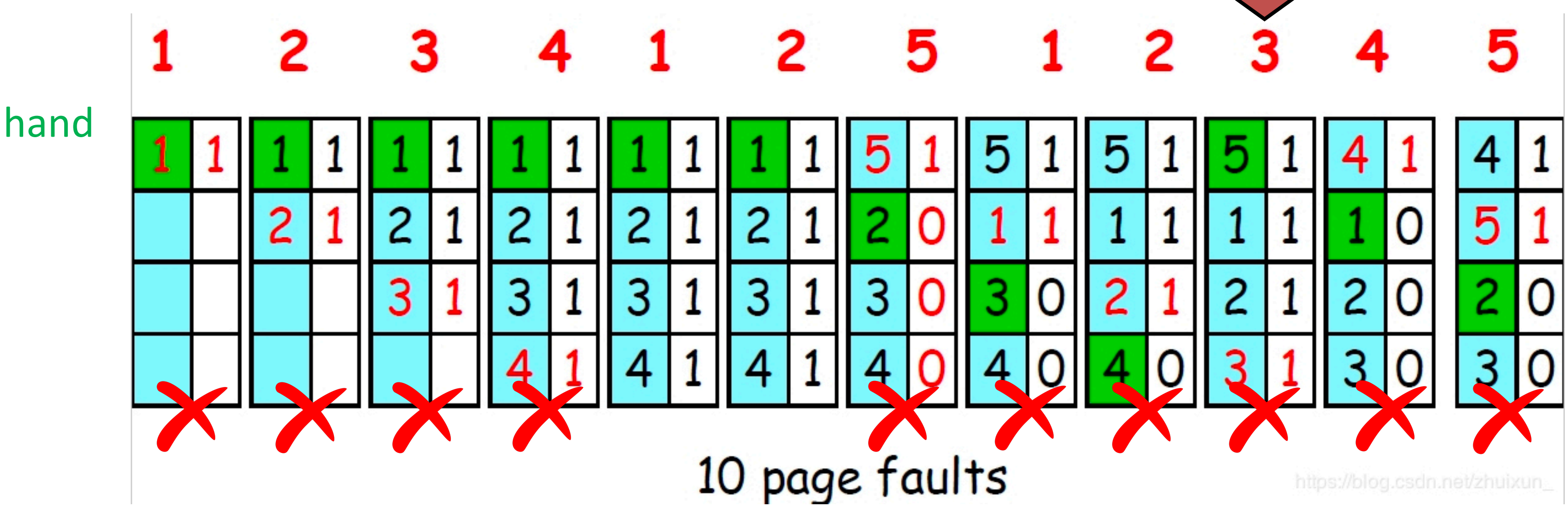
The Clock Page Replacement Algorithm



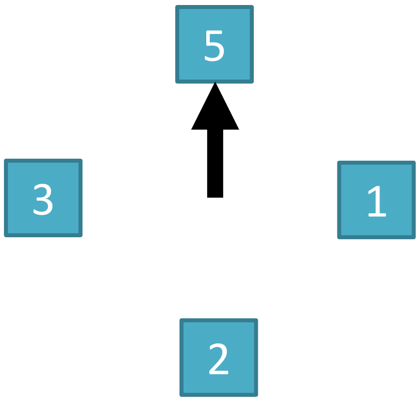
Page fault happens! Check the element which the hand points to (4);

Check R bits: it is 0, replace the page and move hand forward

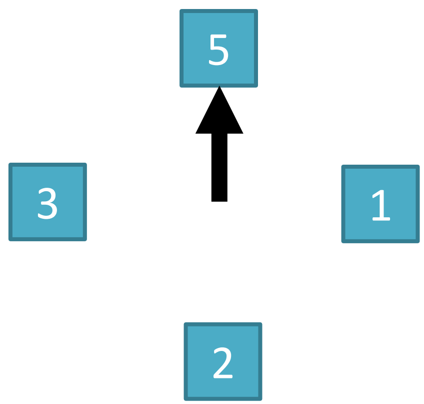




The Clock Page Replacement Algorithm

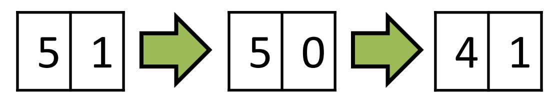


The Clock Page Replacemer



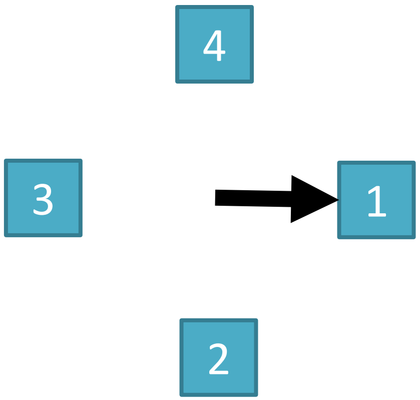
Page fault happens! Check the element which the hand points to (5);

- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

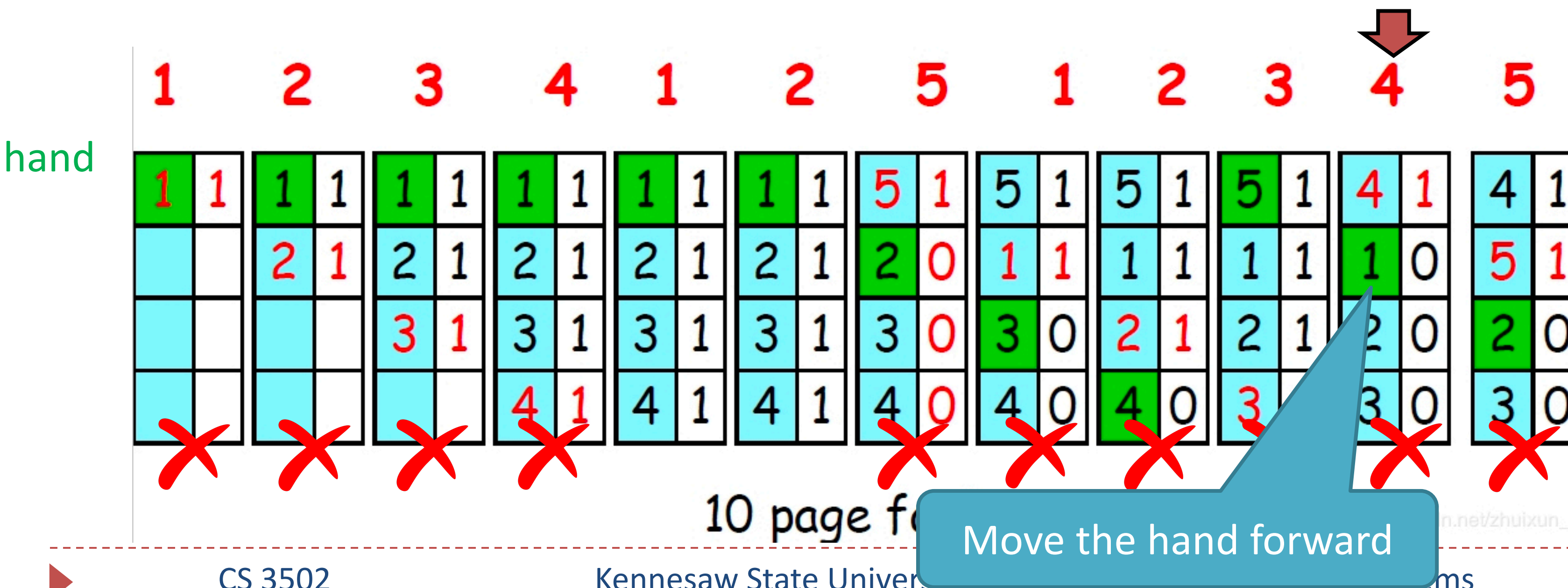


R bits all change to 0 because all R bits are 1s

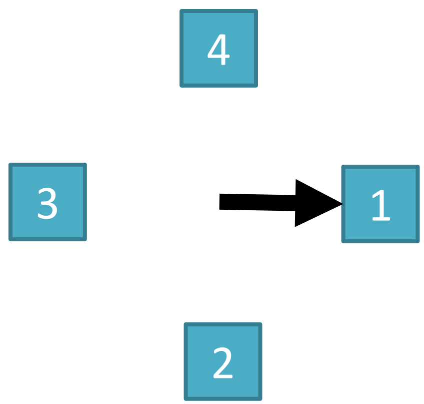
The Clock Page Replacemer



- When a page fault occurs, the page the hand is pointing to is inspected:
  - If the R bit is 0,
    - replace the page, advance hand;
  - If the R bit is 1,
    - clear R bit,
    - replace the page when all R bits are 1, and set all R bits to 0; otherwise, do nothing;
    - advance hand

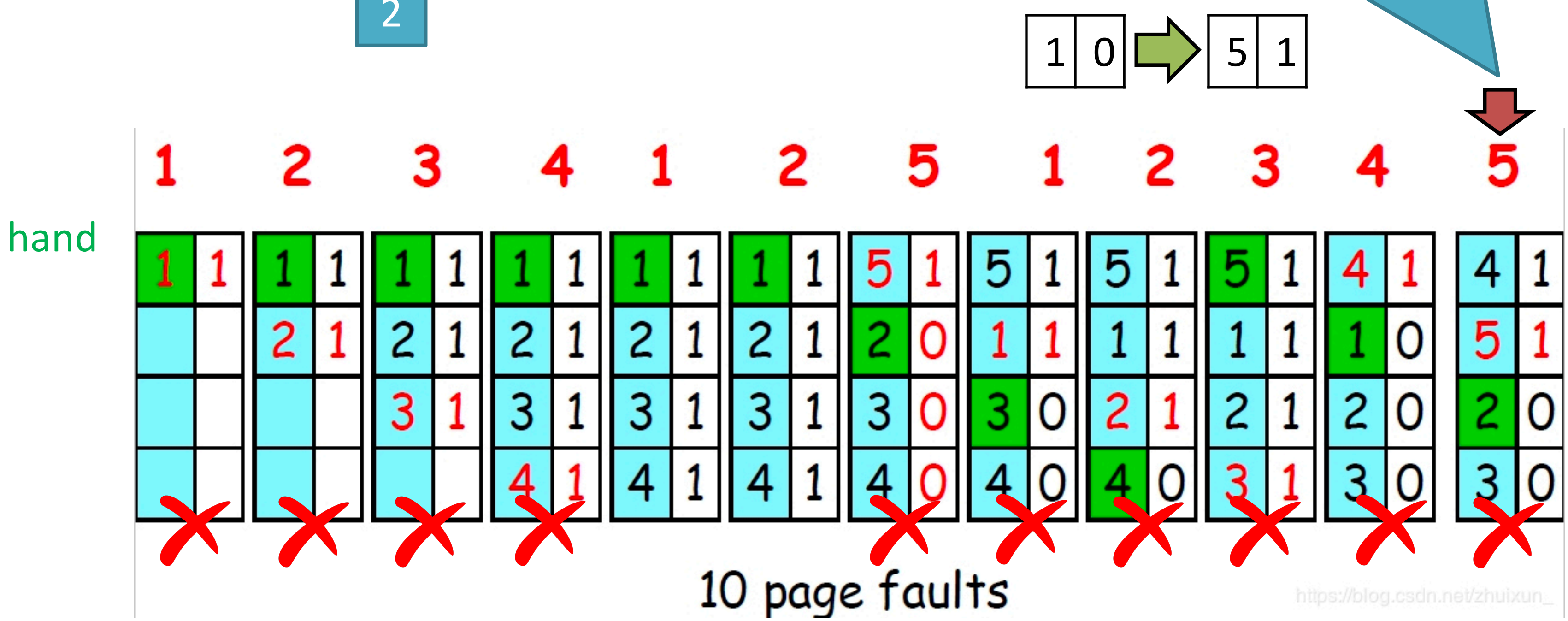


The Clock Page Replacement Algorithm

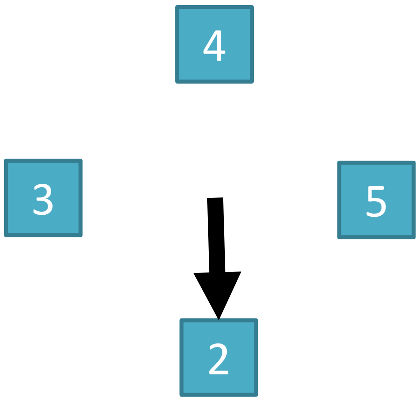


Page fault happens! Check the element which the hand points to (1);

Check R bits: it is 0, replace the page and move hand forward



The Clock Page Replacement Algorithm



Not Frequently Used (NFU)

- NFU (Not Frequently Used): uses a software counter per page to track how *often* each page has been referenced, and chose the least to kick out
  - OS adds R bit (0 or 1) to the counter at each clock interrupt
  - OS adds N-R bits to the counter at each clock interrupt

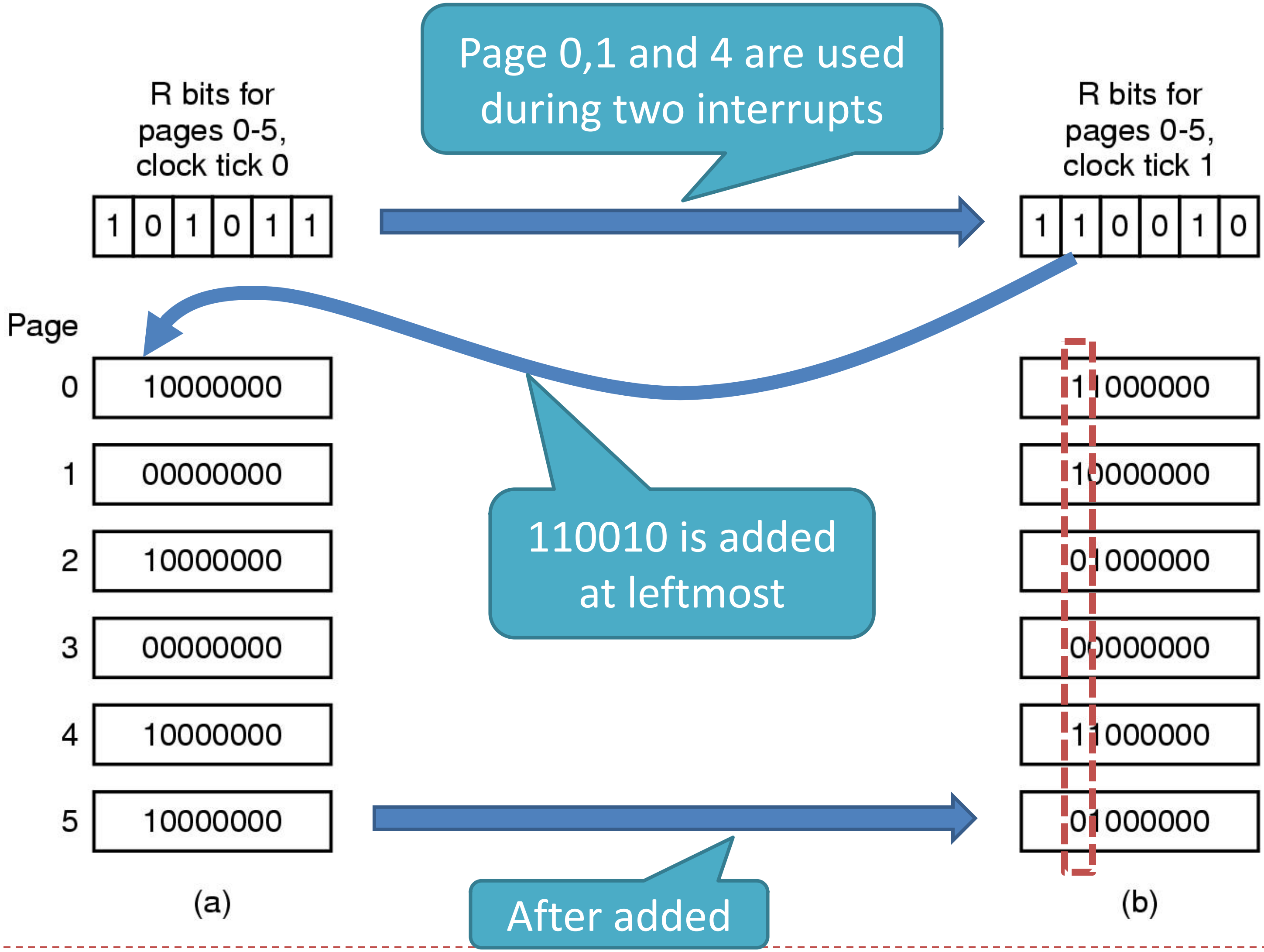
R	item
1	A
0	B
1	C
0	D

Replace page B or D

N-R bits	item
8	A
5	B
3	C
6	D

Replace page C

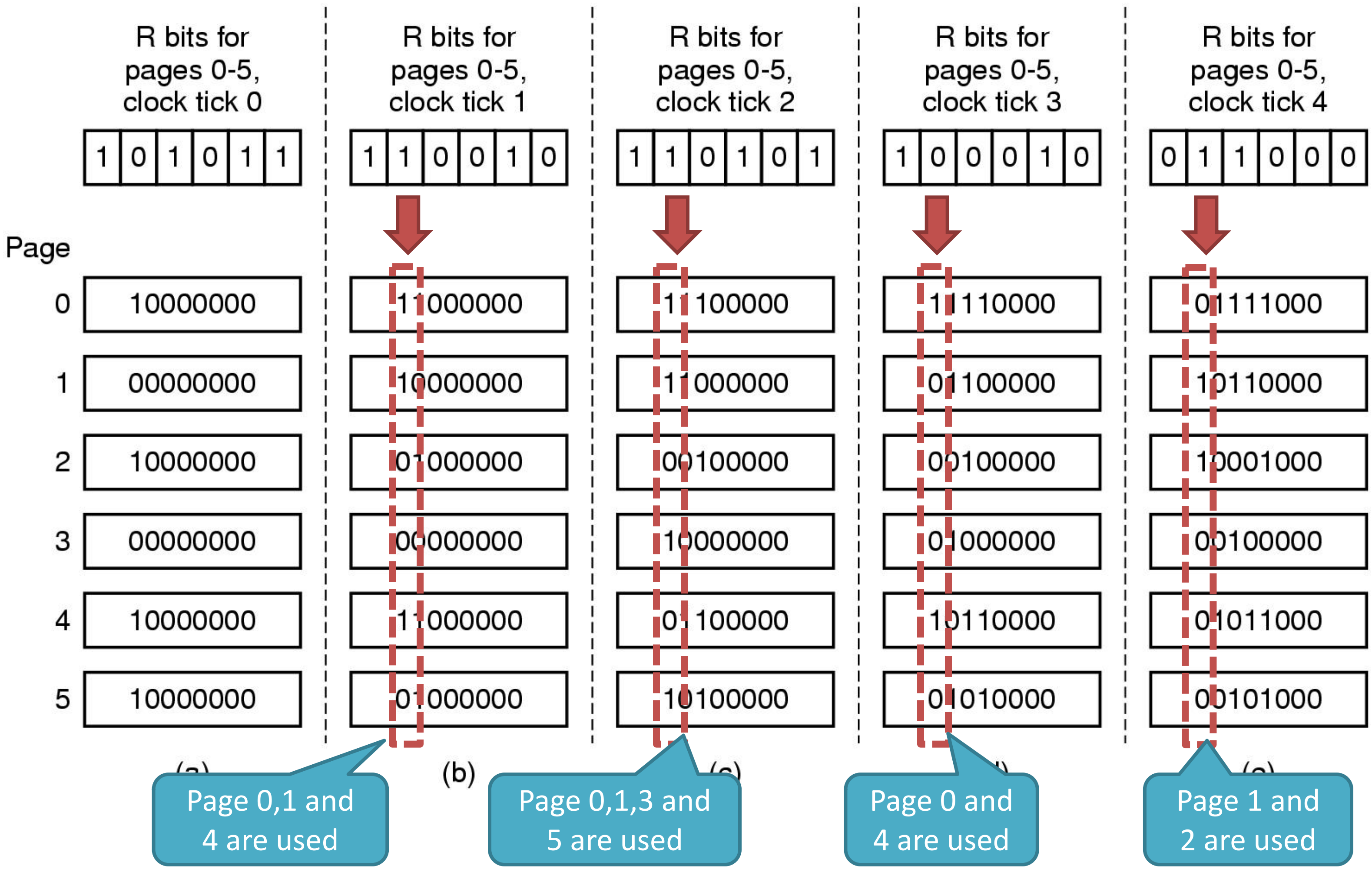
Aging - Simulating LRU/NFU in Software



Aging: the counters are each shifted right 1 bit before the R bit is added in; the R bit is then added to the leftmost

- The page whose counter is the lowest is removed when a page fault

The *aging* algorithm simulates LRU in software, 6 pages for 5 clock ticks, (a) – (e)



Comparison of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well

Conclusion

- Memory management data structure
  - Bit maps vs. Linked lists
- Page replacement algorithm
  - OPR, FIFO, LRU
  - NFU, NRU
  - Second chance, Clock
  - Aging

Question

- Suppose a reference page order: **2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2**. The memory size is **3**. What is the number of page faults and page fault rate by using **Optimal Page Replacement** Algorithm.

	2	**3**	**2**	**1**	**5**	**2**	**4**	**5**	**3**	**2**	**5**	**2**
Page 1												
Page 2												
Page 3												
Fault												

6/12*100% = 50%

Question

- Suppose a reference page order: **2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2**. The memory size is **3**. What is the number of page faults and page fault rate by using **FIFO Replacement** Algorithm.

	2	3	2	1	5	2	4	5	3	2	5	2
Page 1												
Page 2												
Page 3												
Fault												

9/12*100% = 75%

Question

- Suppose a reference page order: **2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2**. The memory size is **3**. What is the number of page faults and page fault rate by using **LRU Replacement** Algorithm.

	2	3	2	1	5	2	4	5	3	2	5	2
Page 1												
Page 2												
Page 3												
Fault												

7/12*100% = 58.3%