PROJECT

SUNG KYUN KWAN UNIVERSITY

SSE3044 Introduction to Operating Systems
Prof. Jinkyu Jeong

Project 4. Page Replacement

TA)

송원석(wonsuk.song@csi.skku.edu)

진승우(seungwoo.jin@csi.skku.edu)

Project Plan

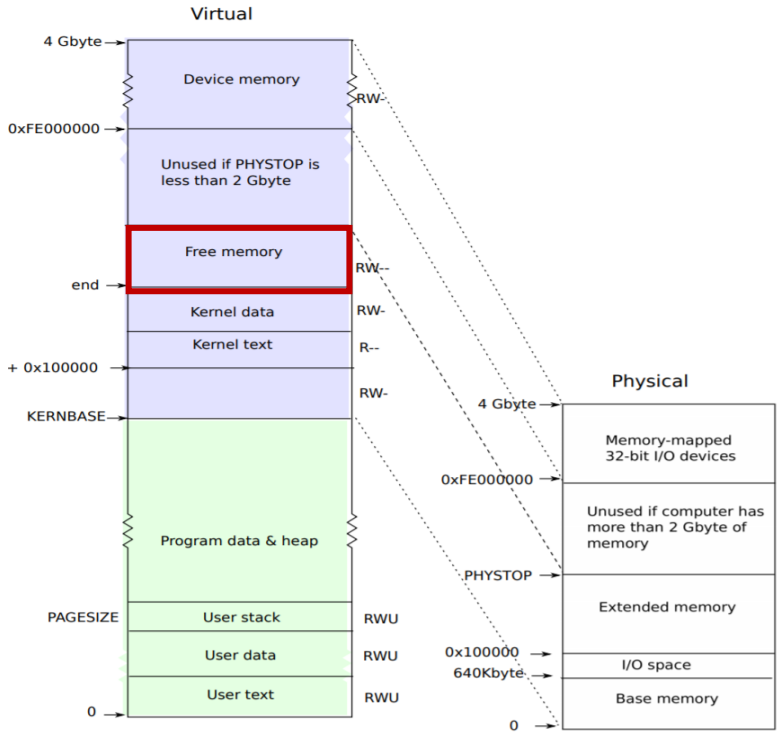
- Total 6 projects
  0. Booting xv6 operating system
  1. System call
  2. CPU scheduling
  3. Virtual memory
  4. Page replacement
     - Swapping
  5. File systems

Project Objective

- Implement page-level swapping
  - Swap-in operation
  - Swap-out operation
  - Manage swappable pages with LRU list
    - Page replacement policy: clock algorithm

Physical Memory Initialization in xv6

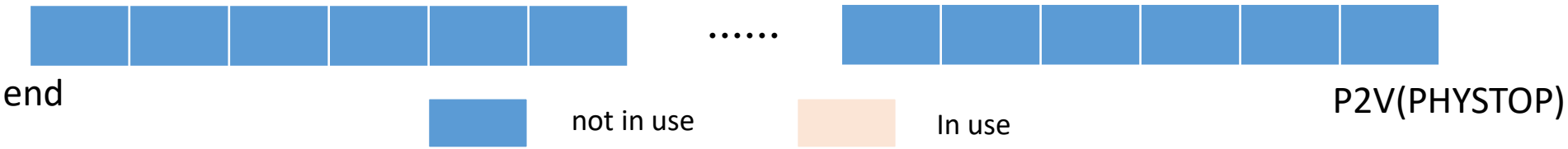
- When xv6 boots up, **kinit1()** and **kinit2()** to initialize free memory from end to P2V(PHYSTOP)
  - Uses a free page list to manage this free memory
    - To allocate this free page, use **kalloc**()
    - To free this allocated page, use **kfree**()



Memory Allocation in xv6

1. kinit1() & kinit2() : initialize free pages to be allocatable
2. kalloc();
3. kalloc();
4. kfree();

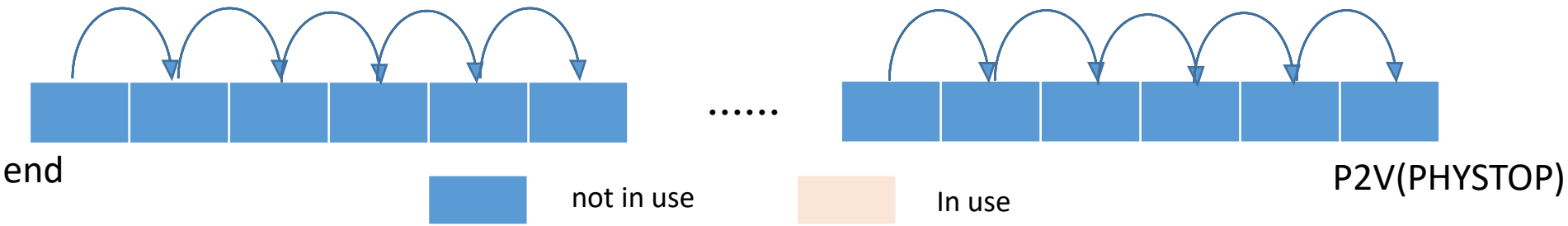
⋮



Memory Allocation in xv6

1. kinit1() & kinit2() : initialize free pages to be allocatable
2. kalloc();
3. kalloc();
4. kfree();

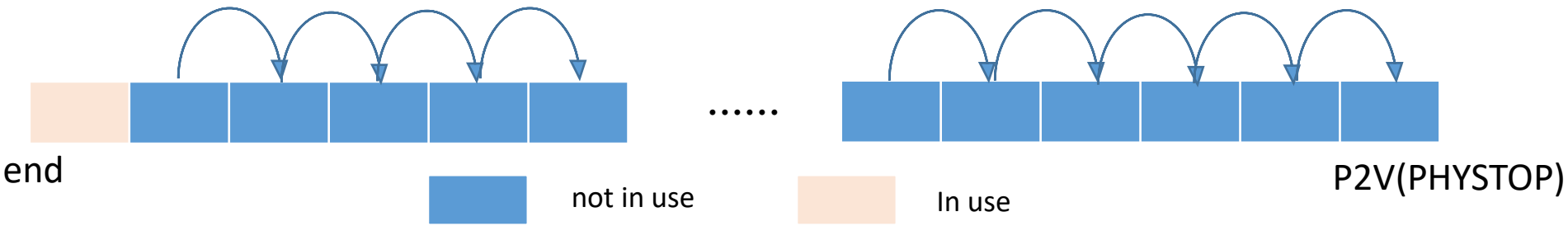
⋮



Memory Allocation in xv6

1. kinit1() & kinit2() : initialize free pages to be allocatable
2. kalloc();
3. kalloc();
4. kfree();

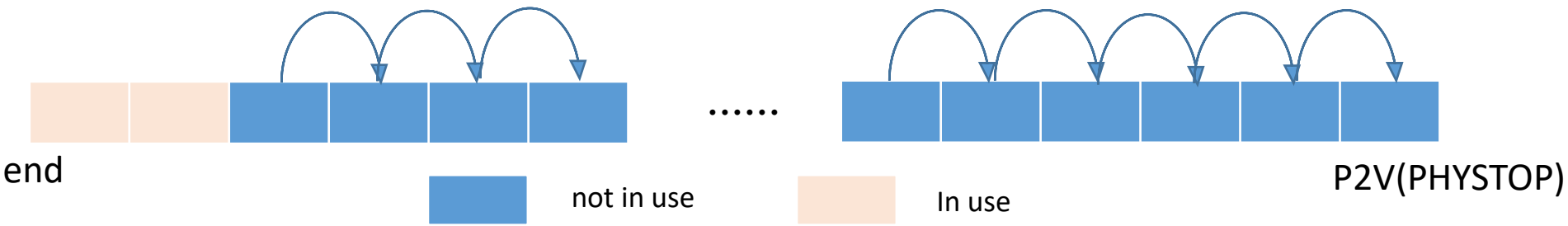
⋮



Memory Allocation in xv6

1. kinit1() & kinit2() : initialize free pages to be allocatable
2. kalloc();
3. kalloc();
4. kfree();

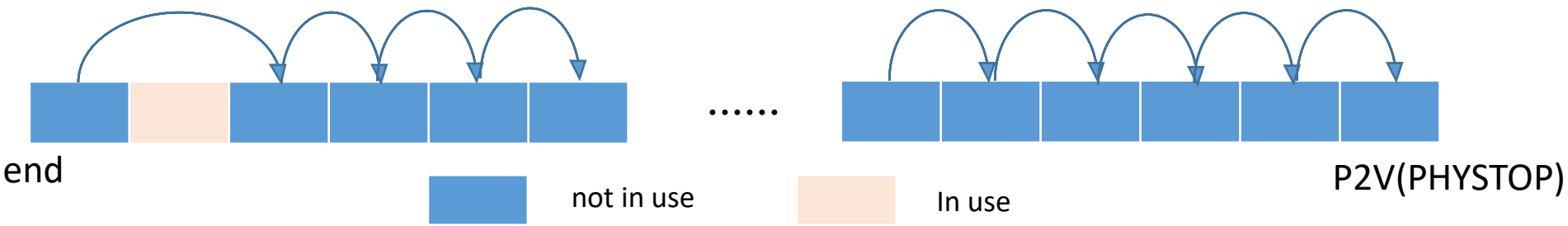
⋮



Memory Allocation in xv6

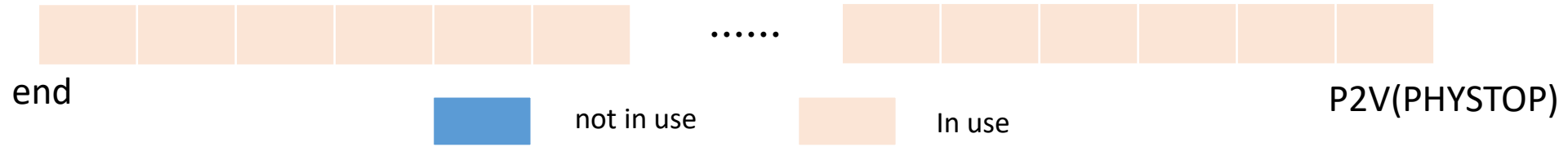
1. kinit1() & kinit2() : initialize free pages to be allocatable
2. kalloc();
3. kalloc();
4. kfree();

⋮



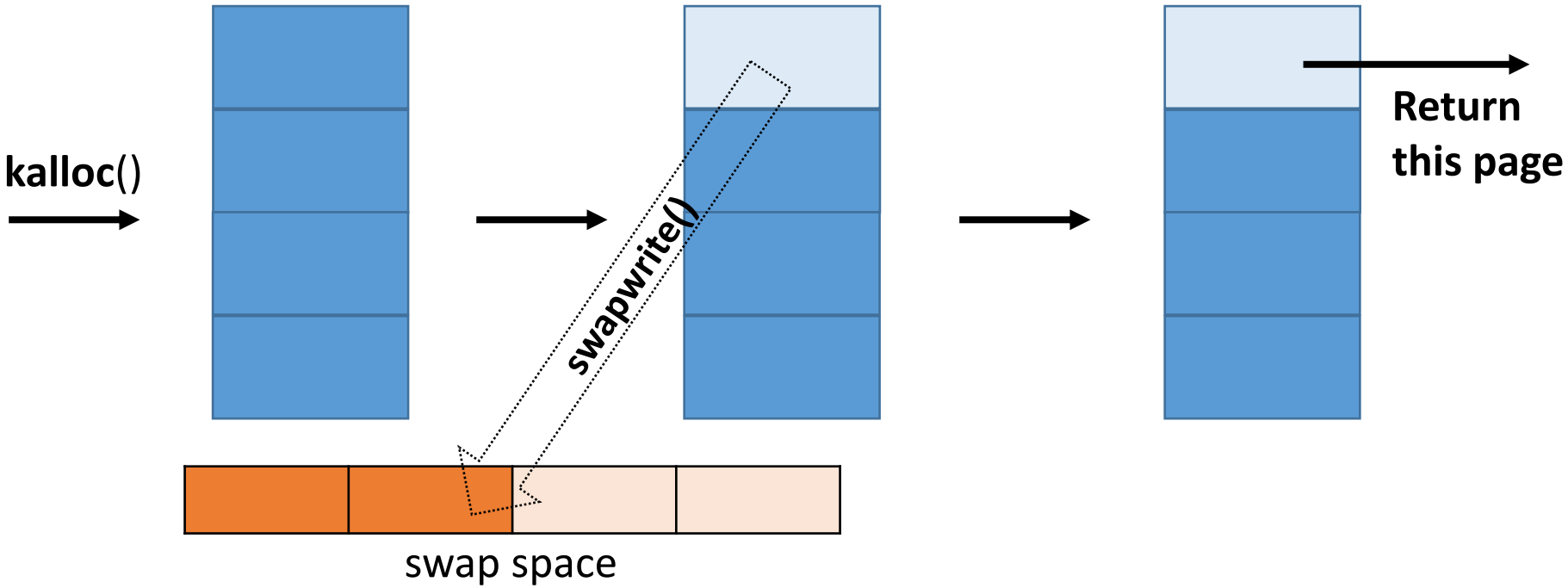
Memory Allocation in xv6

- If there is no free page, **kalloc**() just returns zero and it occurs an error consequently
  - To prevent this, we are going to implement a **swapping** for xv6



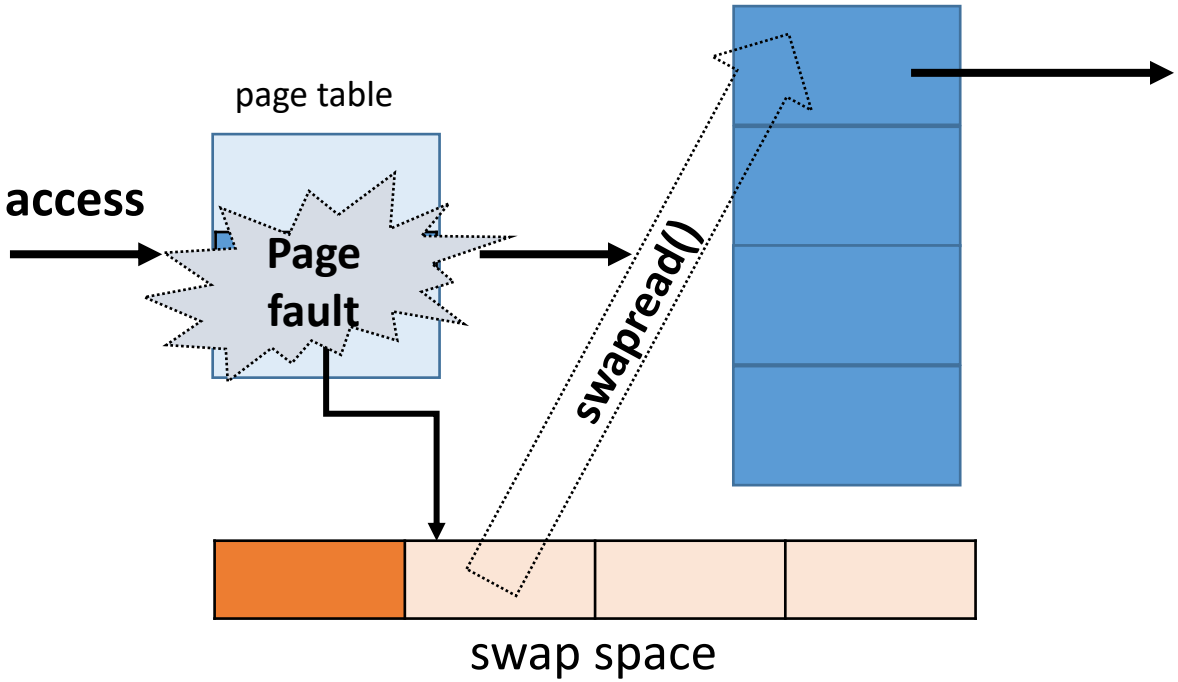
Project 4: Swapping (1)

- If **kalloc**() is called but there is no free page, selects a victim page and swaps temporarily out of memory to a swap space
  - This process is called swap-out
  - Use the **swapwrite**() function which is provided in skeleton code



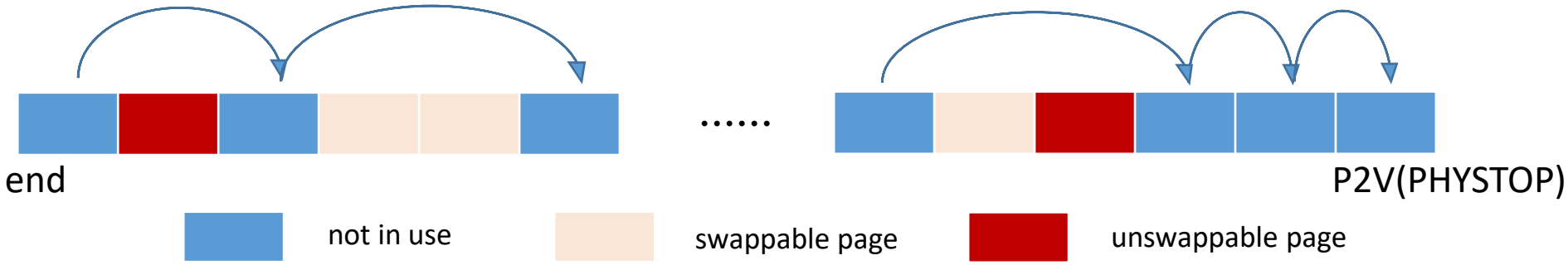
Project 4: Swapping (2)

- If access to swapped out page occurs, load it into memory by demand paging
  - This process is called swap-in
  - Use the **swapread**() function which is provided in skeleton code



Project 4: Swappable Pages

- Only user pages are swappable
  - Generated when invoking the **inituvm**(), **allocuvm**(), **copyuvm**()
  - Some of the physical pages should not be swapped out
    - E.g. page table, page directory, kernel stack
    - You don't need to worry about kernel text and data and because they do not include between end and P2V(PHYSTOP)

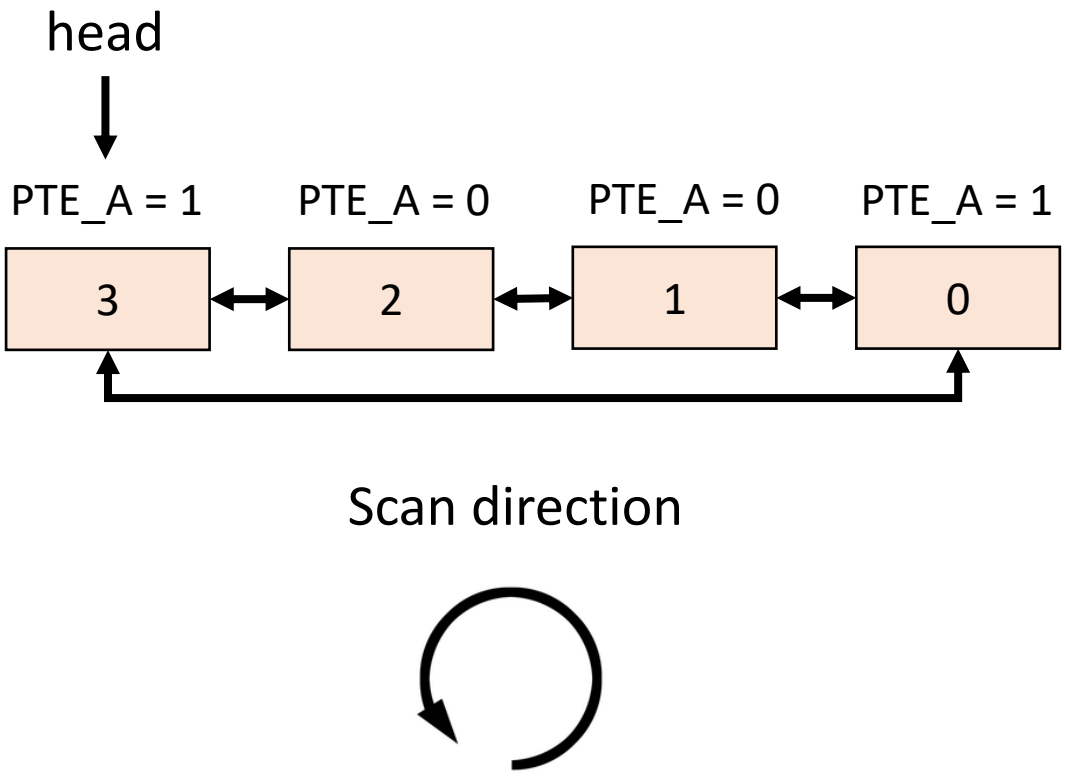


Project 4: Page Replacement

- Policy : clock algorithm
  - Use accessed bit in PTE (PTE_A : 0x20)
    - QEMU automatically sets PTE_A bit when accessed
  - Manage swappable pages using circular doubly linked list in a LRU manner
    - Implement using a **struct page** defined on all pages in skeleton code
    - Insert the new node into head->next
    - From the head of the list, select a victim page by scanning in the direction of the previous node
      - If PTE_A == 1, clear the flag and go to previous page
      - If PTE_A == 0, swap-out this page and go to previous page

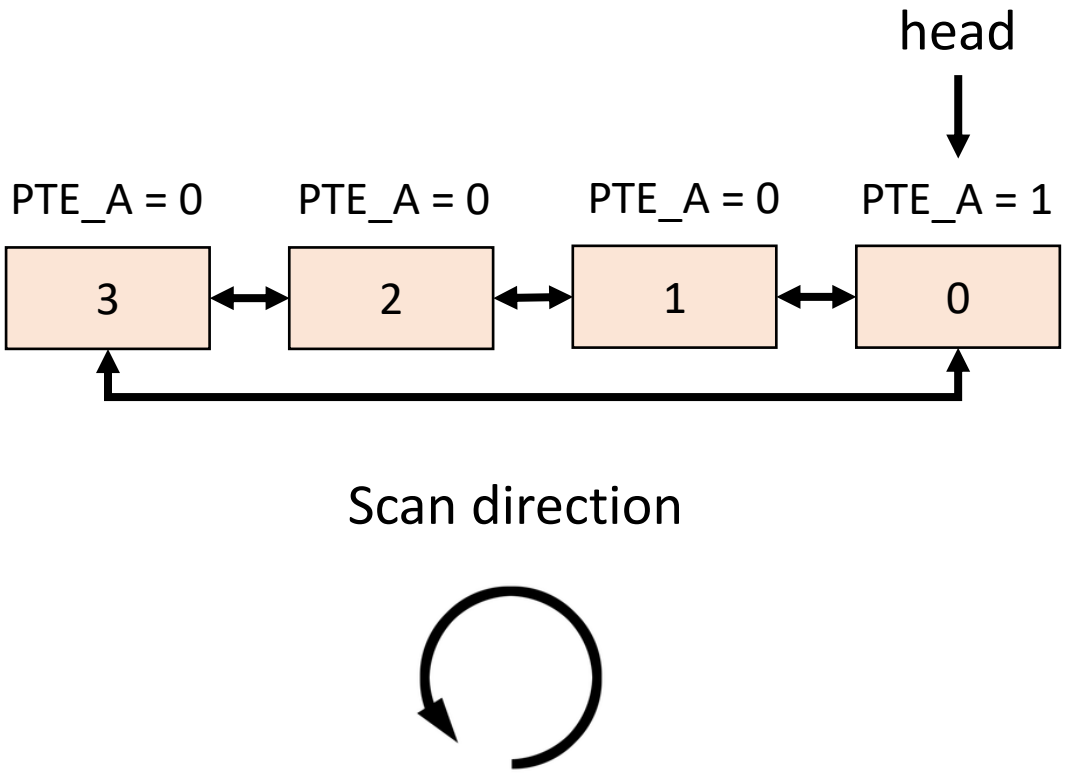
Clock Algorithm Example

1. Check page 3's PTE_A



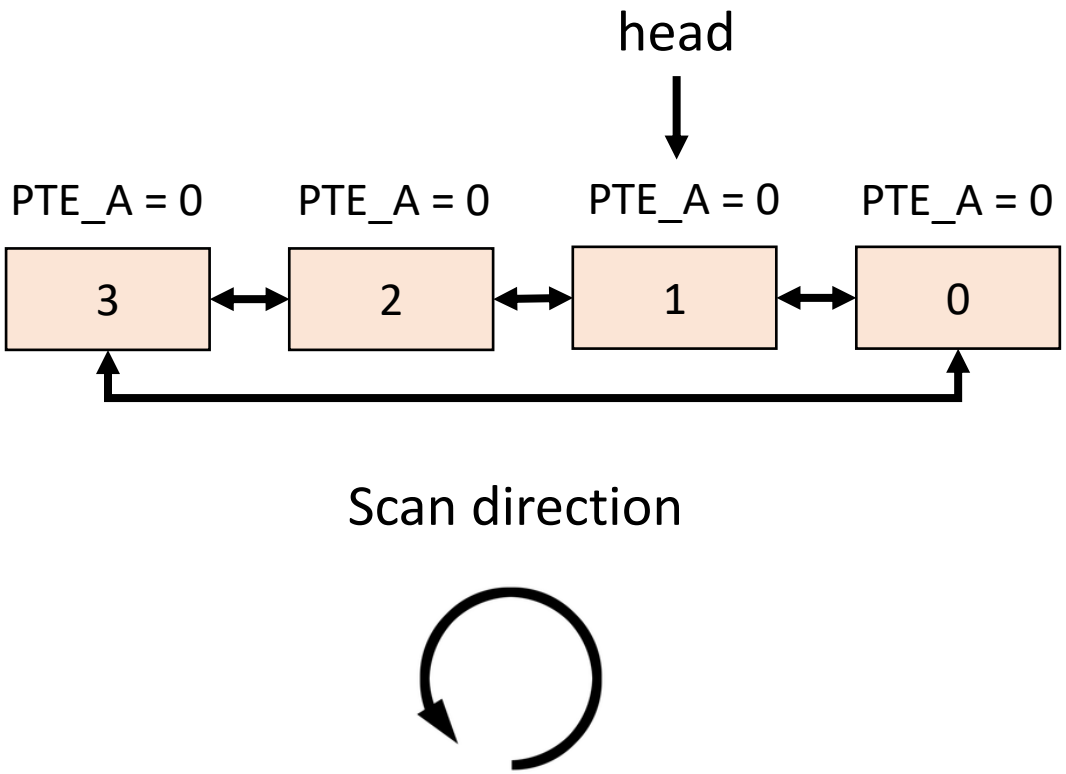
Clock Algorithm Example

1. Check page 3's PTE_A -> clear the flag and go to previous page
2. Check page 0's PTE_A



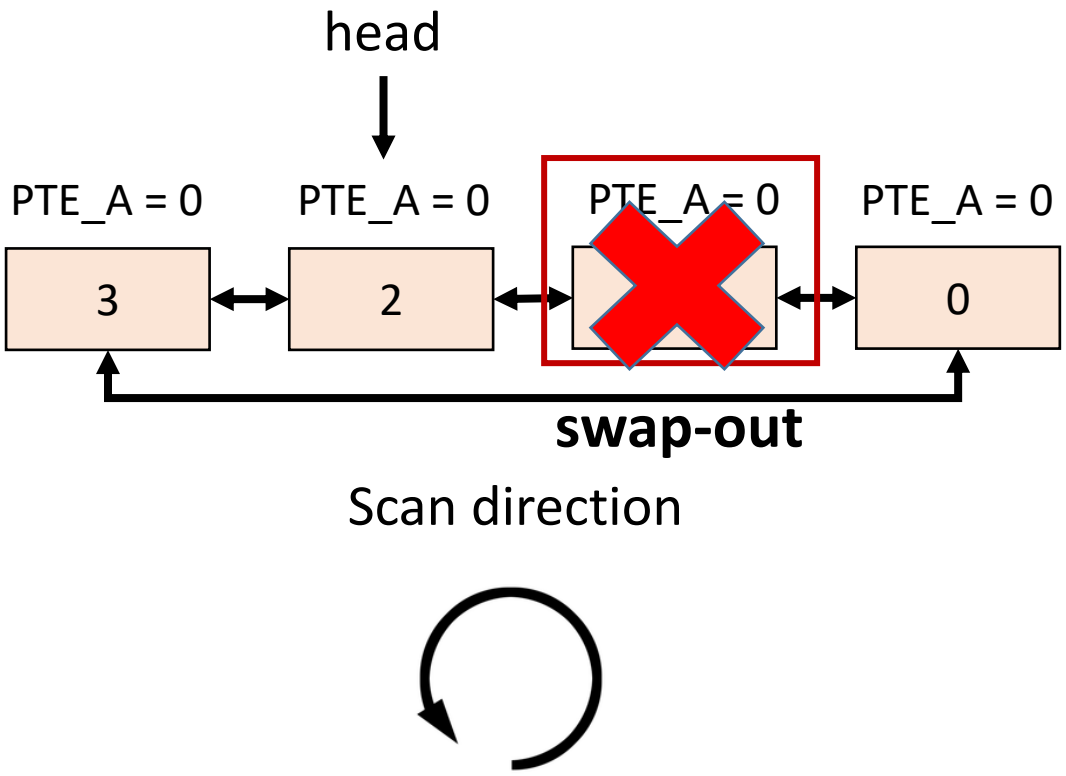
Clock Algorithm Example

1. Check page 3's PTE_A -> clear the flag and go to previous page
2. Check page 0's PTE_A -> clear the flag and go to previous page
3. Check page 1's PTE_A



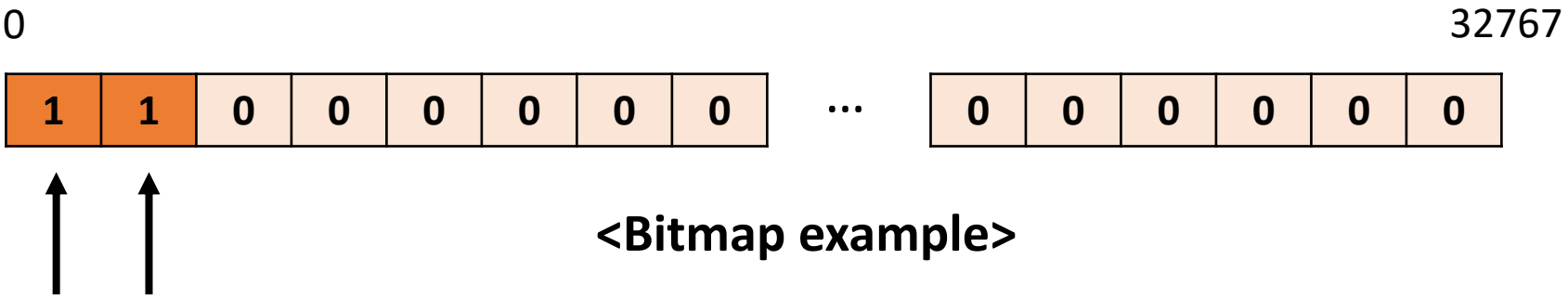
Clock Algorithm Example

1. Check page 3's PTE_A -> clear the flag and go to previous page
2. Check page 0's PTE_A -> clear the flag and go to previous page
3. Check page 1's PTE_A -> swap-out this page and go to previous page



Swap Space Management

- To check where swap space is used, use one physical page to implement a bitmap
  - Set up the bits when the page is swapped out
  - Clear the bits when the page is swapped in
  - The swap space is about 50M in the skeleton code, enough to represent as a single page(128M)
    - Please refer to SWAPMAX, SWAPBASE in skeleton code



<Bitmap example>

Pages are written on swap space 0, 1

Operations during Swap-out

- If **kalloc**() is called but there is no free page
  1. Use bitmap to find a free block
  2. Select a victim page using the clock algorithm
  3. Use **swapwrite**() to write the victim page to the swap space
  4. Update PTE of the victim page
     - Clear the PTE_P
     - Write the swap space offset into the PFN field of the PTE
     - Flush TLB
  5. Remove the victim page from LRU list
  6. Set up the corresponding bit in bitmap
  7. Use **kfree**() to free the victim page

Operations during Swap-in

- When a page fault is occurred and if the faulted PTE is non-zero and PTE_P is not set
  1. Allocate a new physical page
  2. Use **swapread**() to read the page from swap space
  3. Change PTE value with physical address & PTE_P set
     - Tip: do not need to call **mappages**(), because it is already mapped to a page table
  4. Insert the page into LRU list
  5. Clear the corresponding bit in the bitmap

Considerations & Hints

- When swap-out occurs, and there is no page in LRU list, OOM (out of memory) error will occur
  - Just use **panic**(“OOM Error”)
- Consider to user lock when to read or modify shared resources
  - Do not touch the number of CPUs and **yield**() function in trap.c
- When user virtual memory is copied,
  - Swapped out pages should be copied
- When user virtual memory is deallocated,
  - Remove from LRU list
  - Swapped out pages should be cleared in bitmap and set PTE to 0

Skeleton Code

- Provides three useful functions
  - Functions for read / write page to swap space are provided
    - void **swapread**(char* **ptr**, int **blkno**)
    - void **swapwrite**(char* **ptr**, int **blkno**)
  - Function for measuring swap space accesses
    - void **swapstat**(int* **nr_sectors_read**, int* **nr_sectors_write**)

- Extended the swap space more widely

```
xv6.img: bootblock kernel
        dd if=/dev/zero of=xv6.img count=10000
```

→

```
xv6.img: bootblock kernel
        dd if=/dev/zero of=xv6.img count=100000
```

Makefile

Self Swapping Check

1. There are too many free pages in the beginning, you need to reduce the PHYSTOP to reduce free pages

2. Use **malloc**() system call to consume physical memory

3. Use **swapstat**() system call to measure the swap space accesses
   – The less swap-out and swap-in occur, you can get a better score

4. Check the contents of the page are the same as before
   – If not, swapping was performed incorrectly

Submission

- Start using the skeleton code provided
- Compress your source code and upload on i-Campus
- How to compress your project
  - If you insert the user program, Modify the 'EXTRA' in Makefile
  - make dist
  - make tar
  - Then, tar.gz file will be generated automatically
  - Rename to studentID-project4.tar.gz

- Submit a report together. The file format of the report is limited to pdf
  - There is no limit to the format of the contents
  - But, include your description of your code
  - **Also, include result of self swapping check**

Submission

- File format
  - StudentID-project4.tar.gz
  - StudentID-project4.pdf

- PLEASE DO NOT COPY
  - YOU WILL GET F GRADE IF YOU COPIED

- Due date: 5/30(Sat.), 23:59:59 PM
  - -25% per day for delayed submission

Questions

- If you have questions, please ask in Piazza
  - Avoid questions about implementation ☺
  - Read the commentary before posting the question on piazza
    http://csl.skku.edu/uploads/SSE3044S20/book-rev11.pdf