Project #4: Xswap: Compressed Swap for xv6

Injae Kang
(abcinje@snu.ac.kr)
Systems Software & Architecture Lab.
Seoul National University

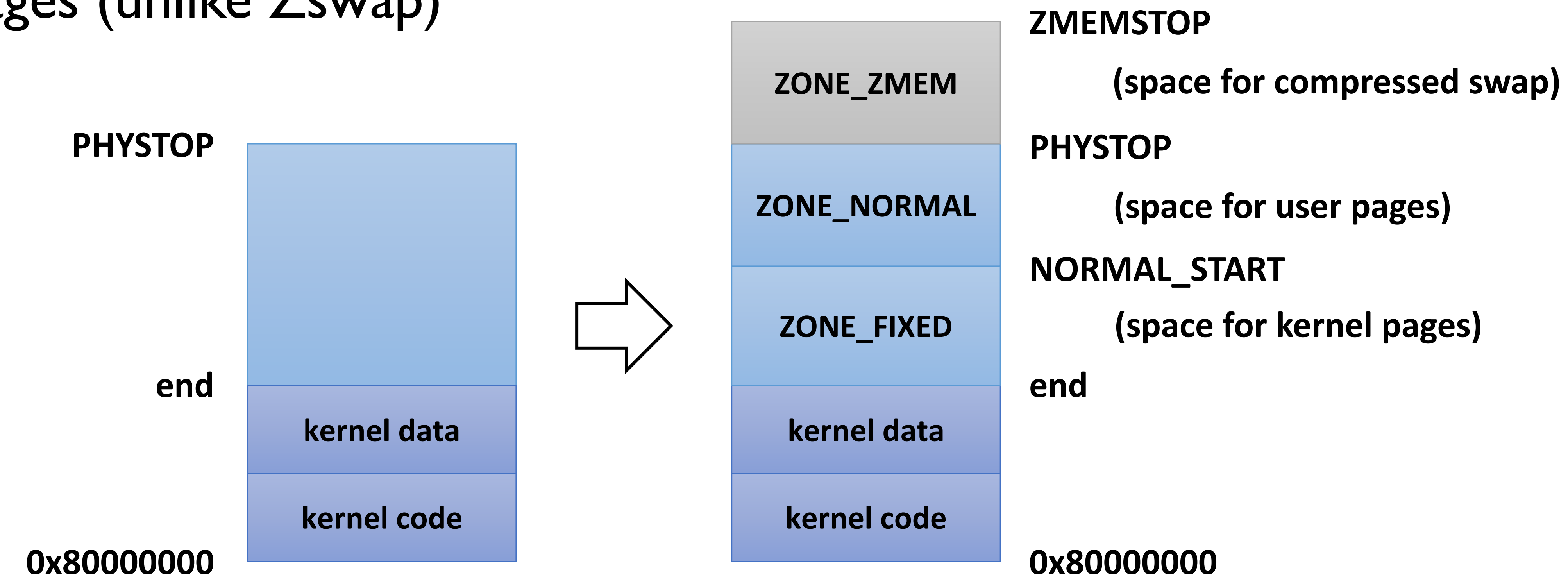
2024.11.11.

Linux Zswap

- Zswap is a lightweight compressed cache for swap pages
- It takes pages that are in the process of being swapped out and attempts to compress them into a memory pool
- It basically trades CPU cycles for potentially reduced swap I/O

Xswap

- Similar to Linux's Zswap, Xswap provides a compressed memory cache for swap pages
- xv6 reserves a portion of physical memory for storing swapped-out pages (unlike Zswap)



LZO Compression

- The default compressor for Linux Zswap

- You can use the following LZO routines available in ./kernel/lzo.c

```
int lzo1x_compress(const unsigned char *src, uint32 src_len,
                   unsigned char *dst, uint32 *dst_len, void *wrkmem);
int lzo1x_decompress(const unsigned char *src, uint32 src_len,
                     unsigned char *dst, uint32 *dst_len);
```

- The algorithm requires 16KB of working memory, and the parameter named wrkmem specifies its starting address

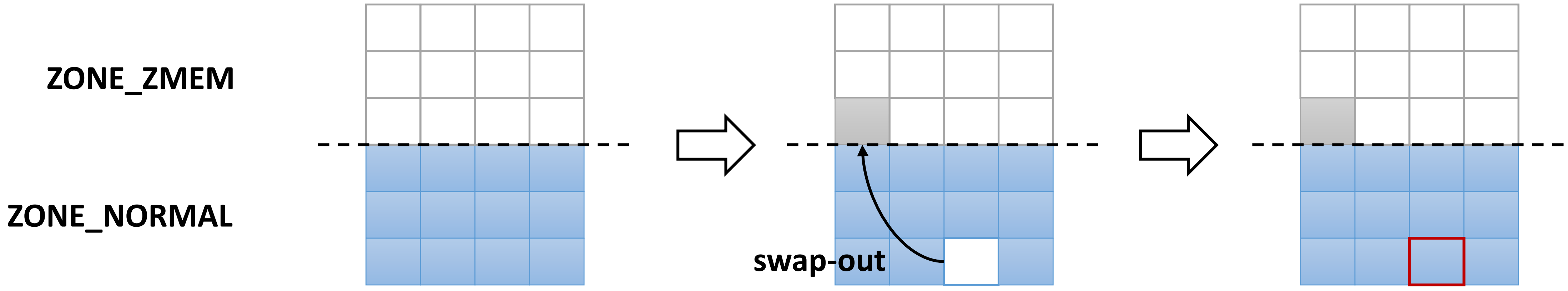
What to Swap

- Page frames in ZONE_NORMAL
  - User code/data/stack/heap pages
  - Different from swappable pages in Linux

- During swapping, the victim page is selected based on the FIFO replacement policy among all pages in ZONE_NORMAL

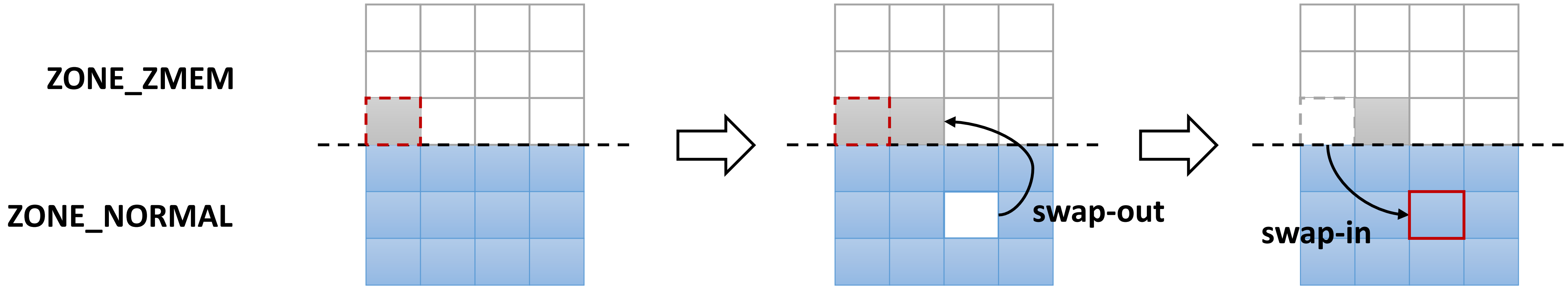
When to Swap-out

- If the kernel calls kalloc() but there are no available frames in ZONE_NORMAL
- The kalloc() function internally calls swapout() when it detects that there are no available frames



When to Swap-in

- If the swapped-out pages are later needed, the kernel calls the swapin() function to restore the pages
- Note that swapin() may require an additional swapout()



Traps in RISC-V

- scause

Exception code	Description
…	…
8	Environment call (syscall)
9 - 11	Reserved
12	**Instruction page fault**
13	**Load page fault**
14	Reserved
15	**Store page fault**
>= 16	Reserved

- stval
  - When a page fault exception occurs on an instruction fetch, load, or store, stval will contain the faulting virtual address

Sv39

- xv6 uses 39-bit address system called Sv39
- 3-level page table

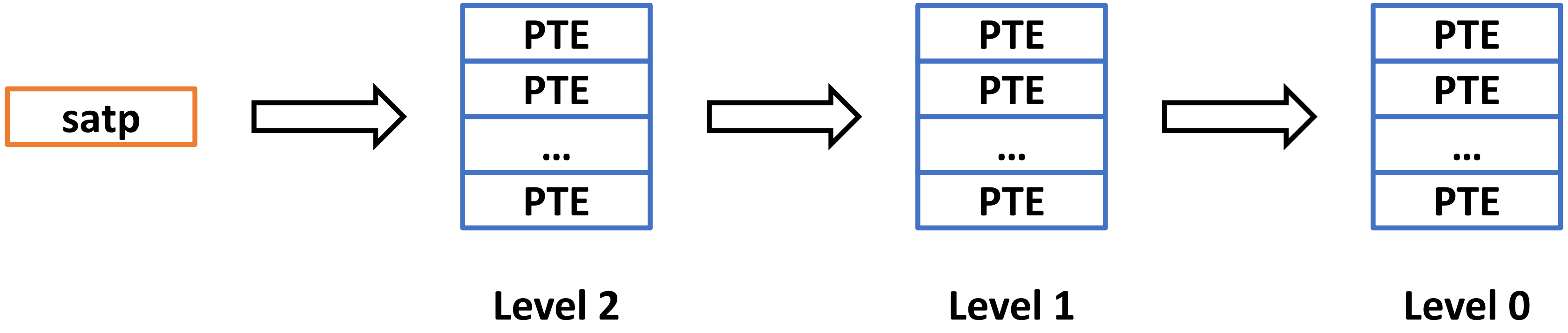
PPN[1]	PPN[1]	PPN[0]	Page Offset
9	9	9	12

Sv39 Virtual Address

PPN[2]	PPN[1]	PPN[0]	Page Offset
26	9	9	12

Sv39 Physical Address

Sv39 Page Table



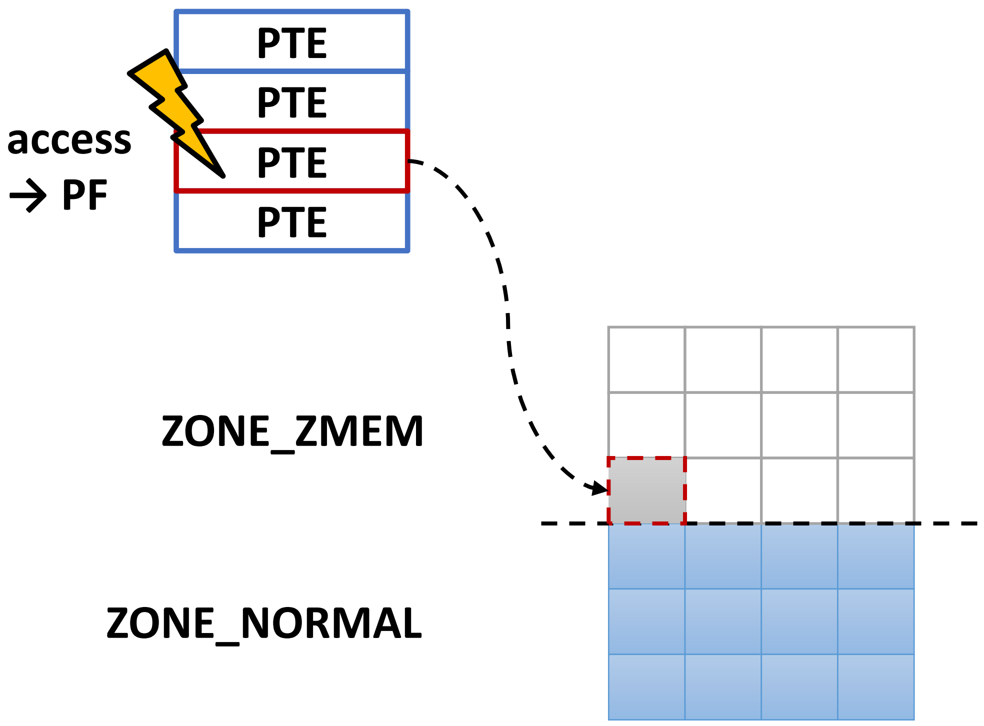
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

Sv39 Page Table Entry

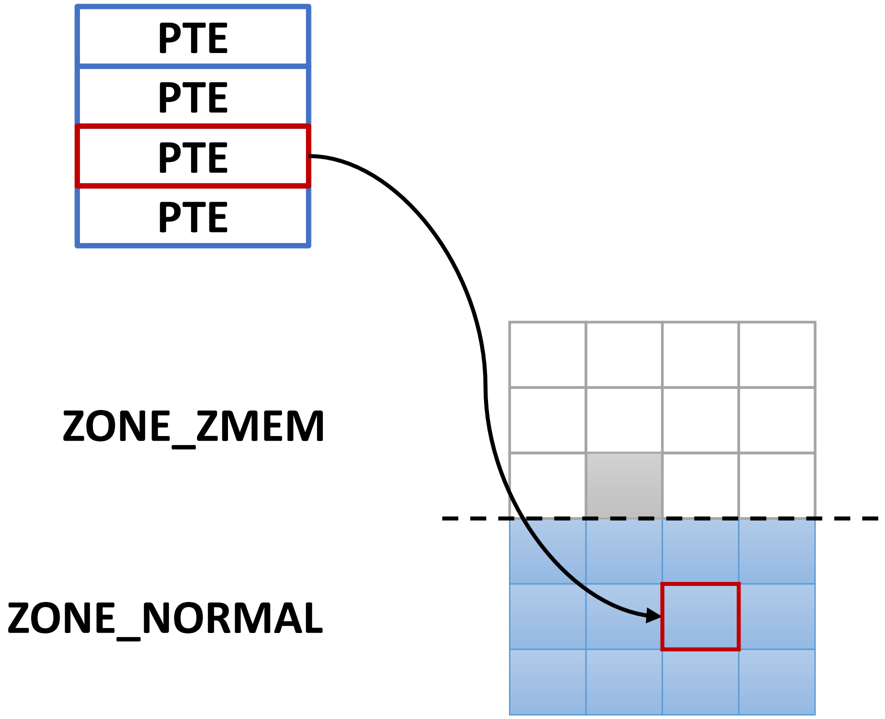
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

- Page table entry bits
  - D: Dirty bit
  - A: Access bit
  - G: Global bit
  - U: User bit
  - X: Execute bit
  - W: Write bit
  - R: Read bit
  - V: Valid bit
- If X, W, and R are all 0, the PTE is a pointer to next level
- The RSW field is reserved for use by supervisor software

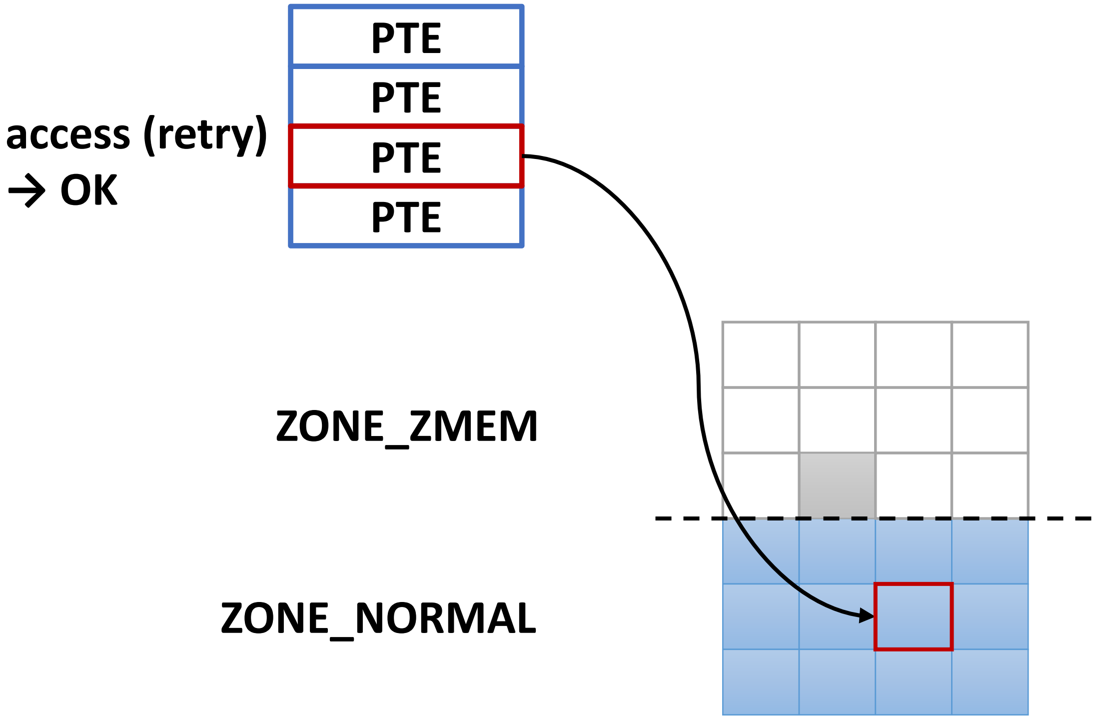
Page Fault Handling (1)



Page Fault Handling (2)



Page Fault Handling (3)



Project #4

1. Implement new physical memory allocators (20 points)

```
void *kalloc(int zone);
```

- It allocates a 4MB frame from the specified memory zone
- Returns the start address on success and 0 on failure

```
void kfree(void *pa, int zone);
```

- It frees the 4KB page frame starting at the specified address pa
- If the address pa does not belong to the specified zone or was not previously allocated, it should trigger panic("kfree")

Project #4

1. Implement new physical memory allocators (20 points) (cont'd)

```
void *zalloc(int type);
```

- It allocates a physical memory block with the specified type
- Returns the start address on success and 0 on failure

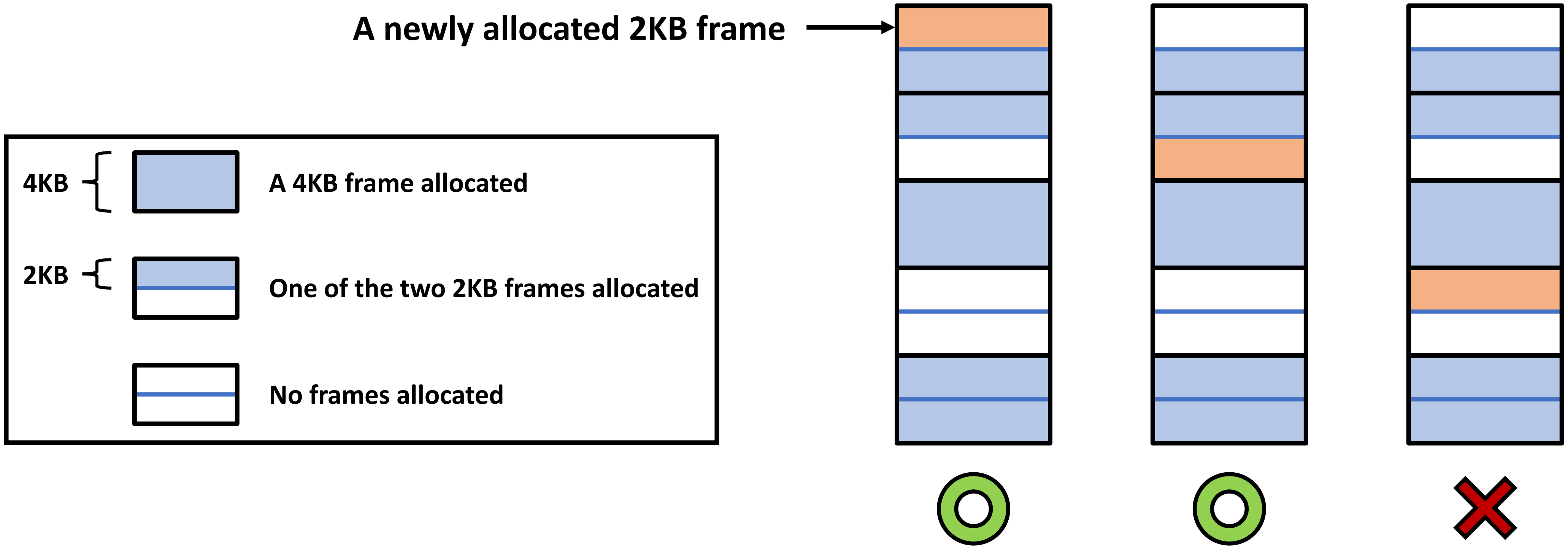
```
void zfree(void *pa, int type);
```

- It frees the page frame starting at the specified address pa
- If the address pa does not belong to ZONE_ZMEM or was not previously allocated, it should trigger panic("zfree")

Project #4

1. Implement new physical memory allocators (20 points) (cont'd)

- When zalloc() is called, you should maximize the number of allocatable 4KB frames



Project #4

1. Implement new physical memory allocators (20 points) (cont'd)

```
int memstat(int *n4k, int *z4k, int *z2k, int *swapin, int *swapout);
```

- It provides information on the status of memory allocation and swapping
- For Part 1, nswapin and nswapout can be left as zero

Project #4

2. Enable swapping functionality (50 points)

- Please refer to the contents in the previous pages
- Additionally, you need to ensure that nswapin and nswapout track the number of swap-in and swap-out operations performed, respectively

Project #4

3. Support compressed swapping and ensure no memory leaks (20 points)

- You will extend the swapping functionality to support compressed swapping using the LZO library
  - If the compressed size ≤ 2KB, the compressed data will be stored in a 2KB frame
  - Otherwise, the original data will be copied in a 4KB frame
- You must also ensure that your implementation is free from memory leaks
  - Whenever you return to the shell after executing a command, the sum of allocation counts should remain the same

Project #4

4. Design Document (10 points)

1. New data structures
2. Algorithm design
3. Testing and validation

Project #4

Bonus (up to additional 20 points)

- If you ensure that swapping works correctly on a multi-core system, you can earn an additional 20-point bonus
- We will use various testcases
- We will run your implementation multiple times to detect bugs that appear occasionally

Project #4

- Tips
  - Read Chap. 3 and 4 of the xv6 book to understand the virtual memory subsystem and page-fault exceptions in xv6
  - Use the following programs to test you implementation
    - $ forktest
    - $ swaptest
    - $ usertests –q
  - Understand the xv6 source code well enough before you start making fixes
  - Start the assignments early!

Project #4

- Restrictions
  - For Part 1 ~ 3, you may assume that xv6 is running on a single-core system
  - For Part 1 and 2, we will run your code without any special compiler options
  - For Part 3, we will use the –DPART3 compiler option
  - For the bonus, we will use the –DMULTI compiler option along with –DPART3
  - You should use the QEMU version 8.2.0 or higher
  - You are required to modify only the files in the ./kernel directory
  - If you have created your own test cases, place them in the ./user directory and mention them in your documentation

Project #4

- **Skeleton Code**
  - You should work on the pa4 branch of the `xv6-riscv-snu` repository as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa4
```

  - The skeleton code includes new files in the ./kernel directory
    - lzo.c: the LZO compression/decompression library
    - xswap.h, xswap.c: stuff related to swapping
  - The pa4 branch also includes a user-level program called swaptest, which can be built from ./user/swaptest.c

Project #4

- Due
  - 11:59 PM, November 24 (Sunday)

- Submission
  - Run `make submit` to generate a tarball named `xv6-pa4-{STUDENTID}.tar.gz` in the `xv6-riscv-snu` directory
  - Upload the compressed file to the submission server
  - The total number of submissions for this project will be limited to 30
  - Only the version marked FINAL will be considered for the project score
  - The grading server will no longer accept submissions after three days past the deadline

Thank you!