CS 3650 Computer Systems – Spring 2023

OS Kernels, Booting, xv6 (2)

Week 11

Northeastern University

* Acknowledgements: created based on Christo Wilson's lecture slides for the same course.

We will see what we learned so far in xv6

- Week 6 file accesses
- Week 7 virtual memory
- Week 8, 9 concurrency

xv6 file accesses

File access

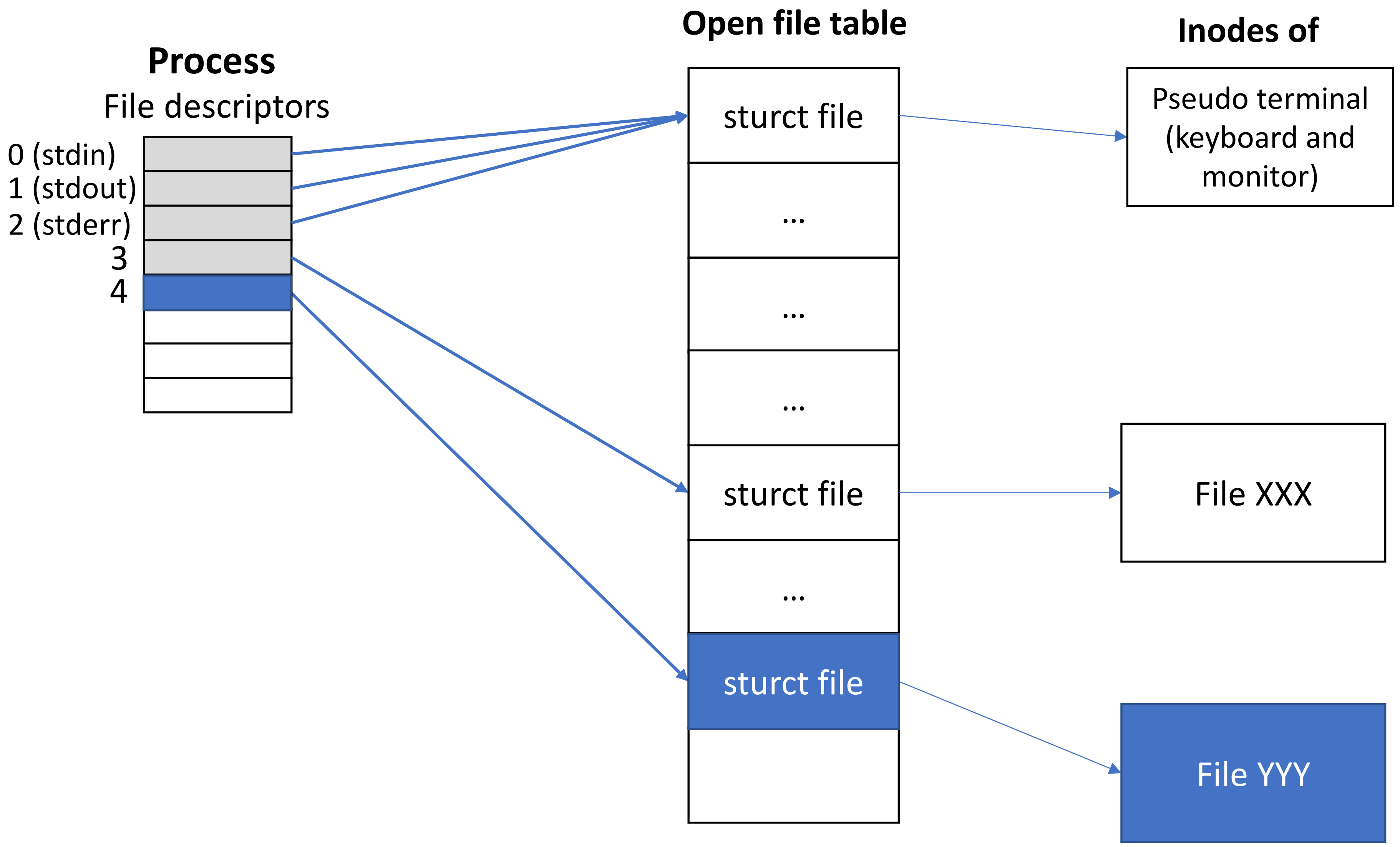
```
// system-wide open files maintained by the OS
struct {
        struct spinlock lock;
        struct file file[NFILE];
} ftable;
```

```
struct file {
        enum {
                FD_NONE,
                FD_PIPE,
                FD_INODE}
        type;
        int ref;
        char readable;
        char writable;
        struct inode *ip;
        struct pipe *pipe;
        uint off;
};
```

```
struct proc {
        …
        struct file *ofile[NOFILE]; // open files
                                    // NOFILE: max # open files
        …
};

// in xv6, file descriptor is the index of ofile
```

Open files


Process
File descriptors
0 (stdin)
1 (stdout)
2 (stderr)
3
4
Open file table
sturct file
…
…
…
sturct file
…
sturct file
Inodes of
Pseudo terminal (keyboard and monitor)
File XXX
File YYY

Proc.h

- Struct proc
  - Contains a context for a process
  - Pid
  - Registers when context switched out
  - List of open files

Console

- Console.c
  - Implements a console
  - Reading and writing from and to the console
    - Calls uartputc/uartgetc
- Uart.c
  - Takes care of input/output through serial port

File operations

- Sysfile.c
    - fdalloc
        - File descriptor allocation
    - Sys_dup
    - Sys_read
    - Sys_write
    - Sys_close
    - Sys_open
    - …

File related structs

- File.h
  - Defines a file struct
  - Inode

- File.c
  - Defines the file table

Sys_open

- Filealloc (file.c)
    - Allocates the file to the file table

- Fdalloc (sysfile.c)
    - Allocates file descriptor in the open file table of the process

File accesses

- Sys_read/write (sysfile.c)
  - Fetches system call arguments
  - Calls fileread/write

- Fileread/write (file.c)
  - Calls readi/writei
    - Inode read/write request
    - This operation can depend on file system implementation

Dup

- Sys_dup (sysfile.c)
    - Calls fdalloc with the given file
    - Calls filedup

- Filedup (file.c)
    - Simply increments ref count

Pipe

- Sys_pipe (sysfile.c)
  - Calls pipealloc
- Struct pipe (pipe.c)
  - 512byte circular buffer
  - Read/write index
- Pipealloc (pipe.c)
  - Fileallocs two files
  - Creates one pipe instance
  - Linkes the same pipe to two file instances
- Piperead/write (pipe.c)
  - Reads and writes from and to pipe

xv6 address translation

Page tables

- Let’s assume
  - 4GB address space
  - 4KB page size

- How many bits do we need to address 4GB address space?

- How many pages can fit into the address space?

- How many bits do we need to address pages in the address space?

- How many bits do we need to address 4KB page?

Xv6 addressing and page tables

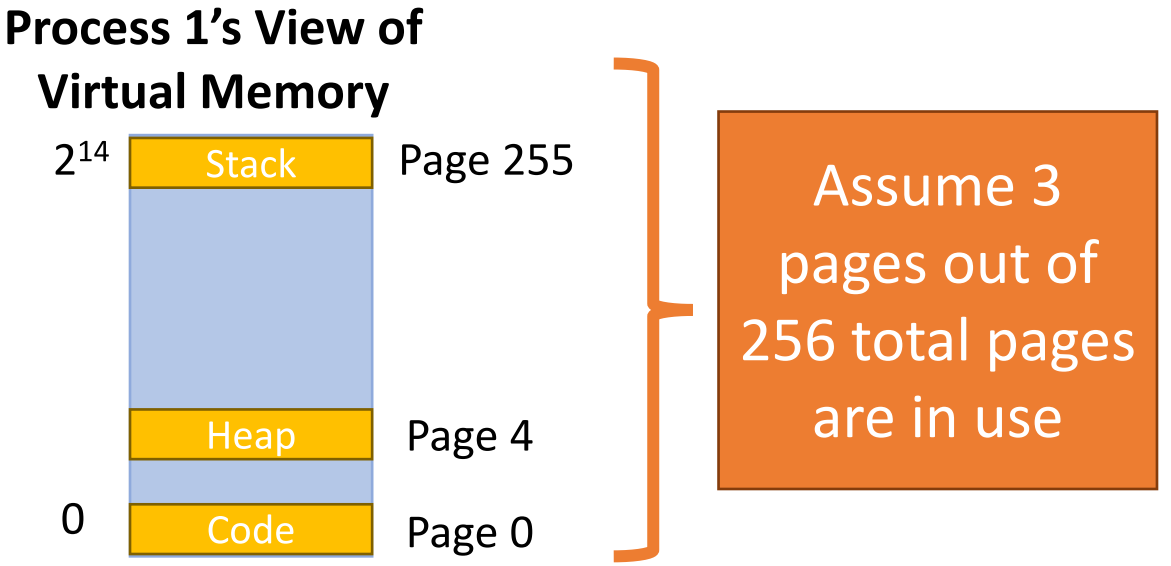
- 32 bit memory address
  - First 20 bits: (physical/virtual) page number
  - 12 bits: page offset
- Logically an array of 2^20 (= 1M) page table entries (PTE)
- Each PTE stores 20 bit physical page number (PPN) + some flags

Multi-Level Page Tables

- Key idea: split the linear page table into a tree of sub-tables
  - Benefit: empty branches (i.e., pointers to invalid pages) can be pruned
- Multi-level page tables are a space/time tradeoff
  - Pruning reduces the size of the table (saves space)
  - But now the tree must be traversed to translate virtual addresses (increased access time)
- Technique used by modern x86 CPUs
  - 32-bit: two-level tables
  - 64-bit: four-level tables

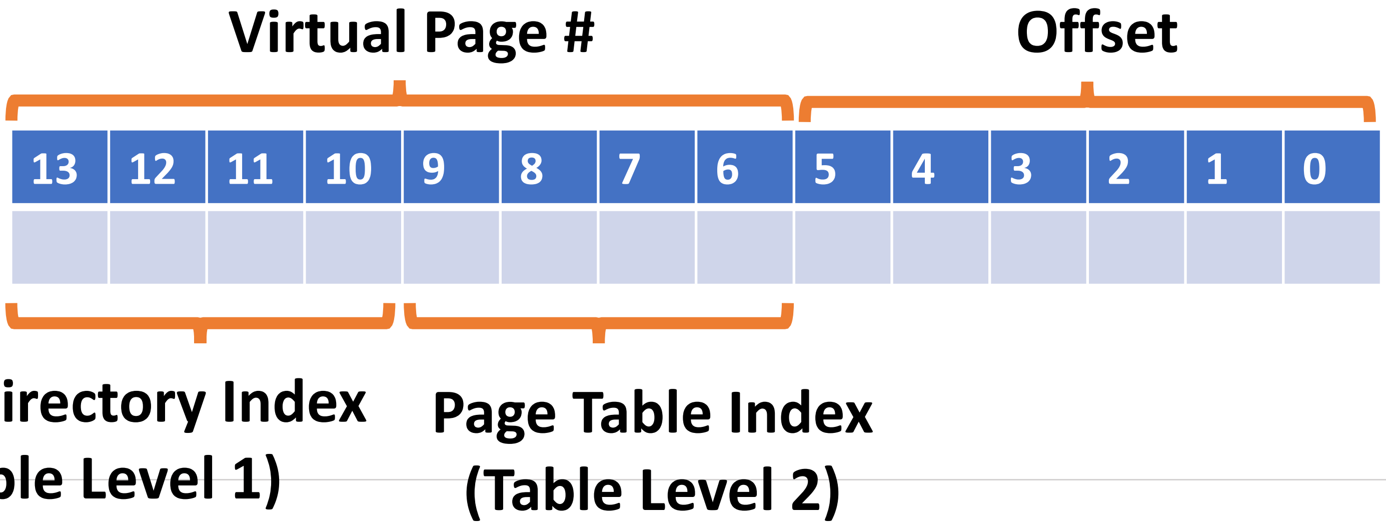
Multi-Level Table Toy Example

- Imagine a small, 16KB address space
  - 64-byte pages
  - 14-bit virtual addresses, 8 bits for the VPN and 6 for the offset
- How many entries does a linear page table need?
  - 16K / 64 = $2^{14} / 2^{6} = 2^{8}$ = 256 entries

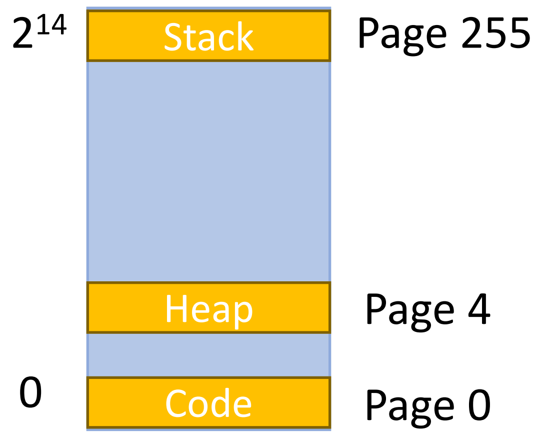

Process 1's View of Virtual Memory
2^{14}
Stack
Page 255
Heap
Page 4
0
Code
Page 0
Assume 3 pages out of 256 total pages are in use

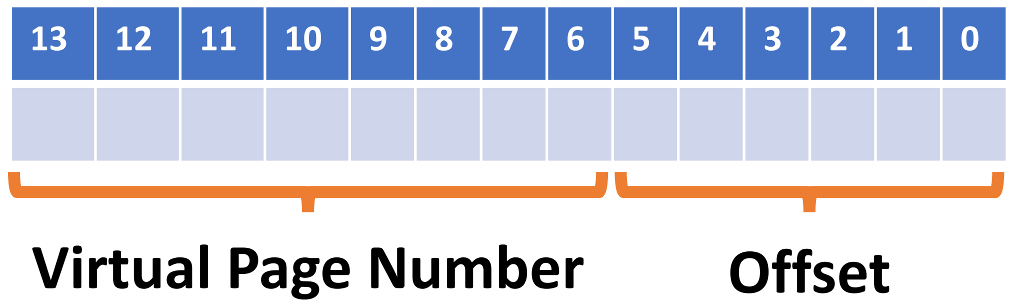
From Linear to Two-levels Tables

- How do you turn a linear table into a multi-level table?
  - Break the linear table up into page-size units
- 256 table entries
  - Assume each entry is 4 bytes large
  - 256 * 4 bytes = 1KB linear page tables
- 1KB linear table can be divided into 16 x 64-byte (page size) tables
  - Each sub-table holds 16 (= 64B / 4B) page table entries



Process 1's View of Virtual Memory

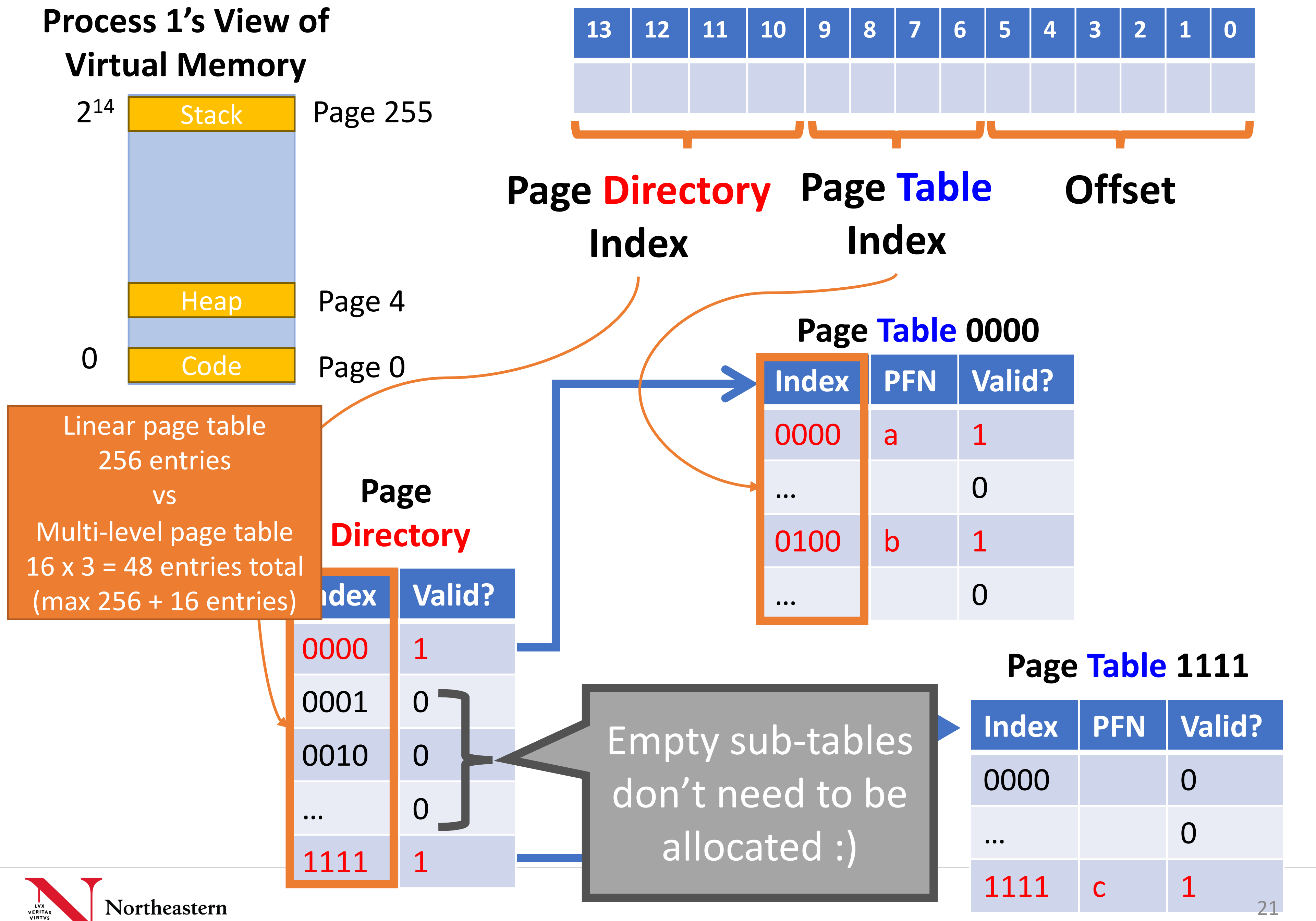




Linear Page Table

VPN	PFN	Valid?
00000000	a	1
…		0
00000100	b	1
…		0
11111111	c	1

253 tables entries are empty
Space is wasted :(



Linear page table
256 entries
vs
Multi-level page table
16 x 3 = 48 entries total
(max 256 + 16 entries)

Page Directory

Index	Valid?
0000	1
0001	0
0010	0
…	0
1111	1

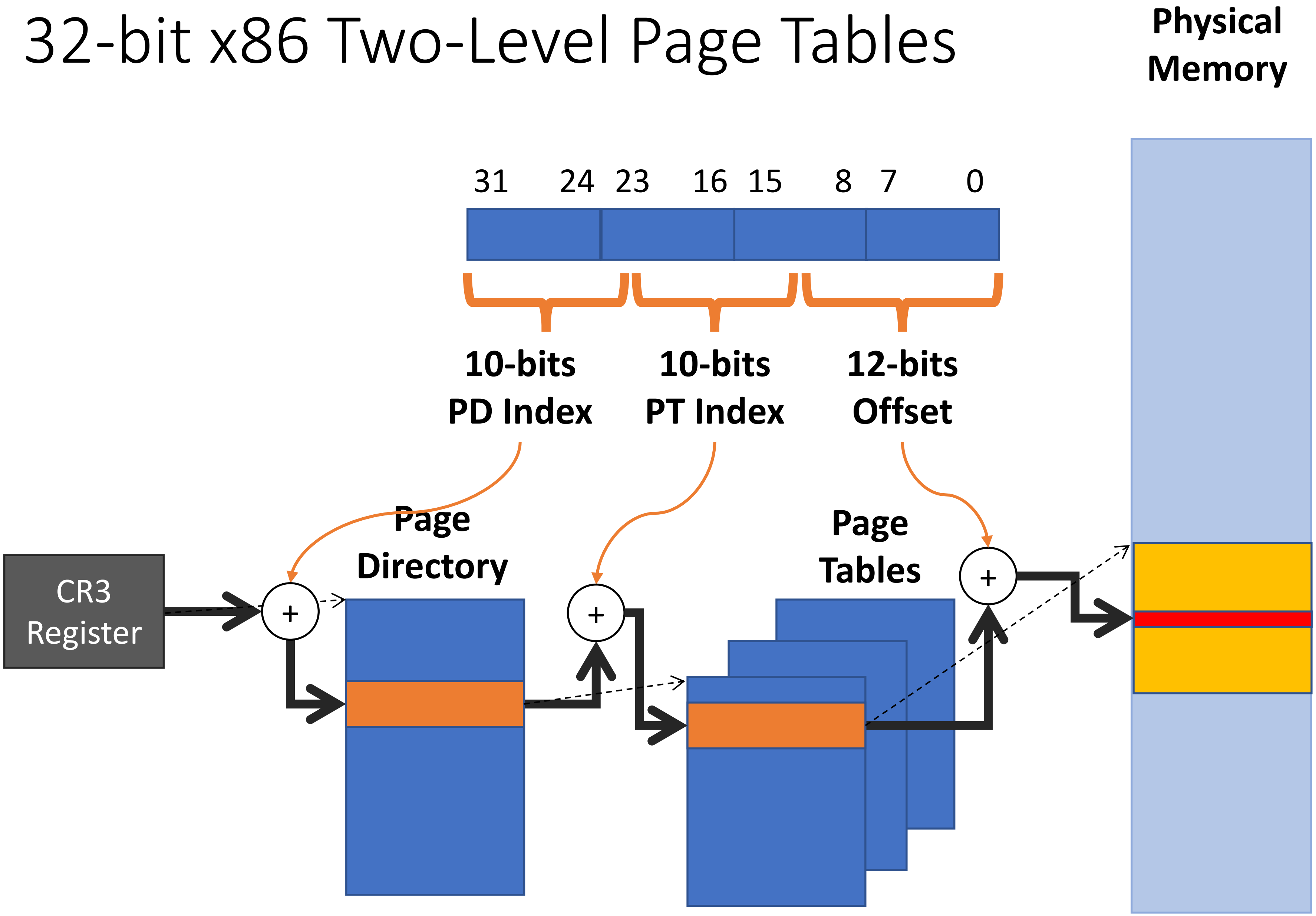
Page Table 0000

Index	PFN	Valid?
0000	a	1
…		0
0100	b	1
…		0

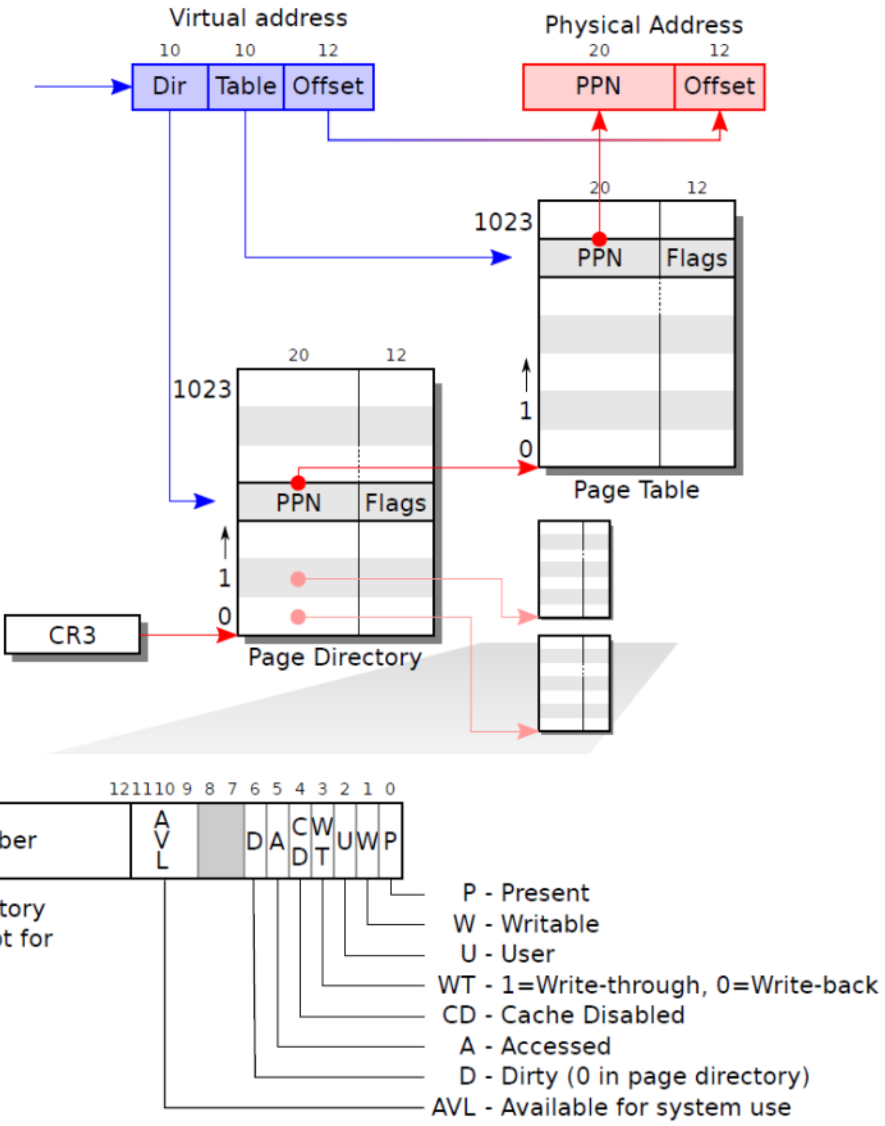
Page Table 1111

Index	PFN	Valid?
0000		0
…		0
1111	c	1

32-bit x86 Two-Level Page Tables



Page translation



Page table and page directory entries are identical except for the D bit.

Page translation

- mmu.h from line 65
  - Explains how an address is decomposed
  - How to extract page directory index
  - How to extract page table index

- Page directory and page tables are simple arrays of uint
  - typedef uint pte_t;
  - typedef uint pde_t;

- Page directory/page traversal is written down in walkpgdir in vm.c

xv6 physical memory management

Physical memory management

- Kalloc.c
  - Kalloc and kfree
    - Linked list managing free physical memory
  - Freerange calls kfree to add free physical memory to free list

sbrk

- sysproc.c
    - sys_sbrk system call
        - Calls growproc in proc.c

- Proc.c
    - Growproc
        - Calls allocuvm

- Vm.c
    - Allocuvm
        - Calls kalloc
        - Calls mappages

xv6 spinlock

Spinlock

- spinlock.h/c
  - Acquire/Release APIs

- x86.h
  - Defines x86 assembly code embeddings