# Lecture 2, Problemset 2

## Virtual Memory

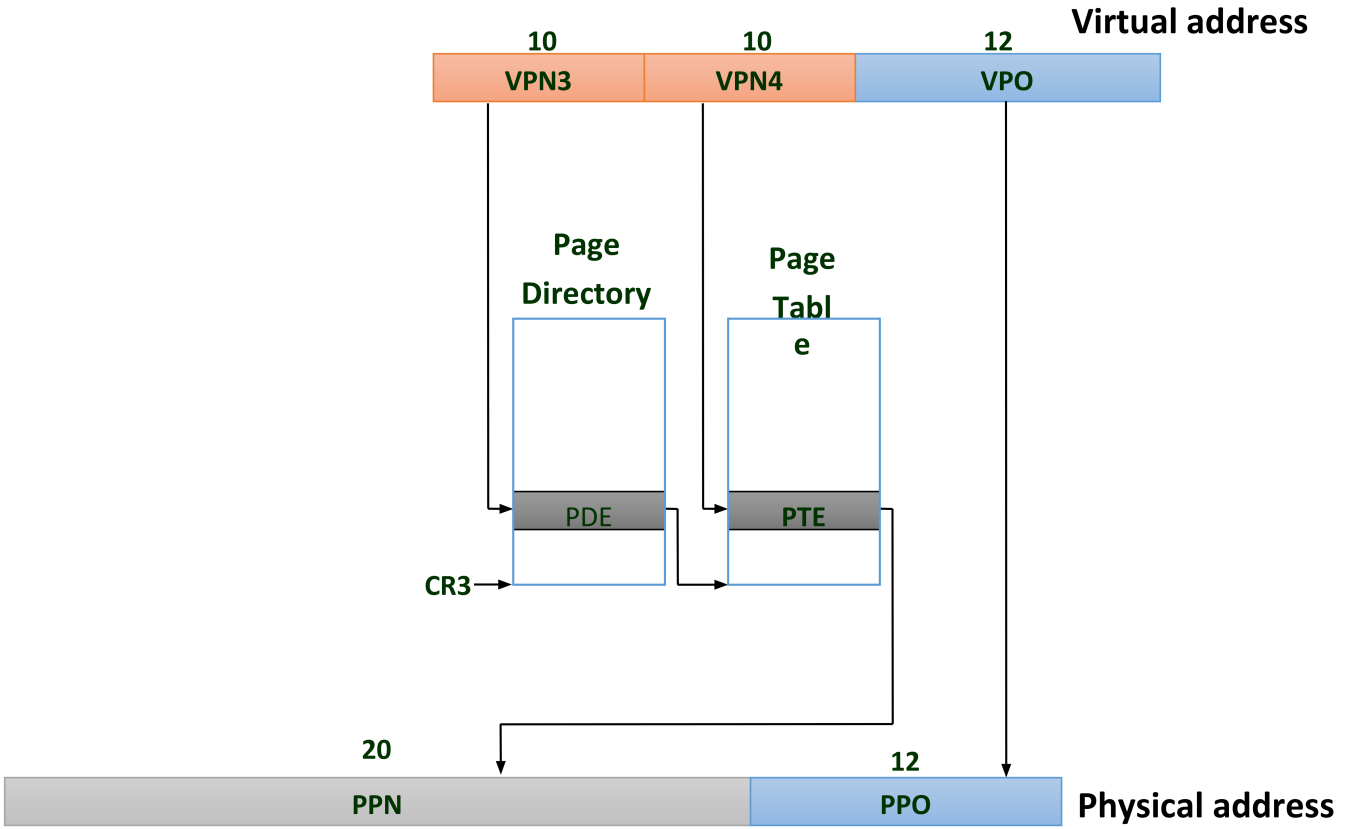
# Agenda

1. X86 Virtual Memory
2. xv6 Code Reading
3. Discussion of Problemset

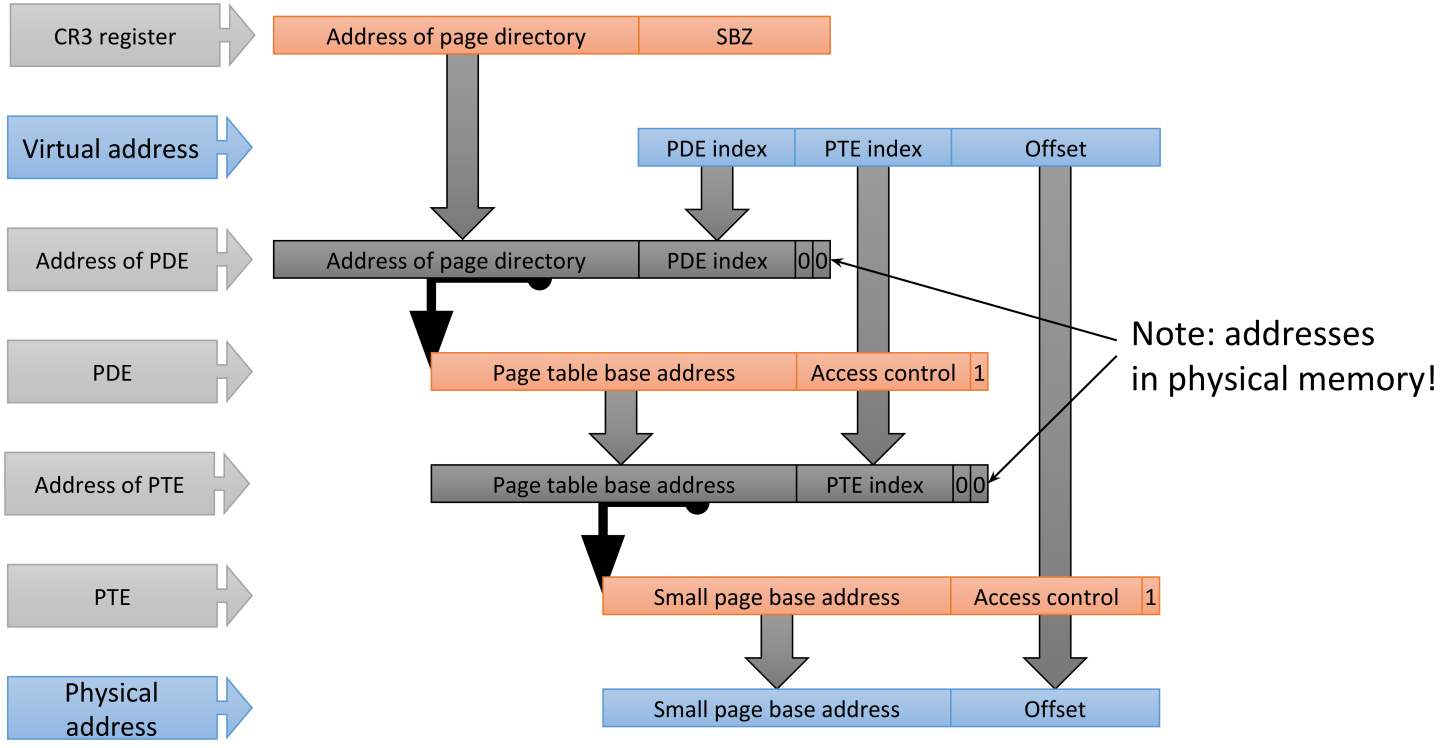
# X86 Virtual Memory Overview

- MMU supports:
  - Small pages (4 KBytes)
  - Large pages (4 MBytes)
- 2 level page table
  - Page directory (top level)
  - Page table (bottom level)

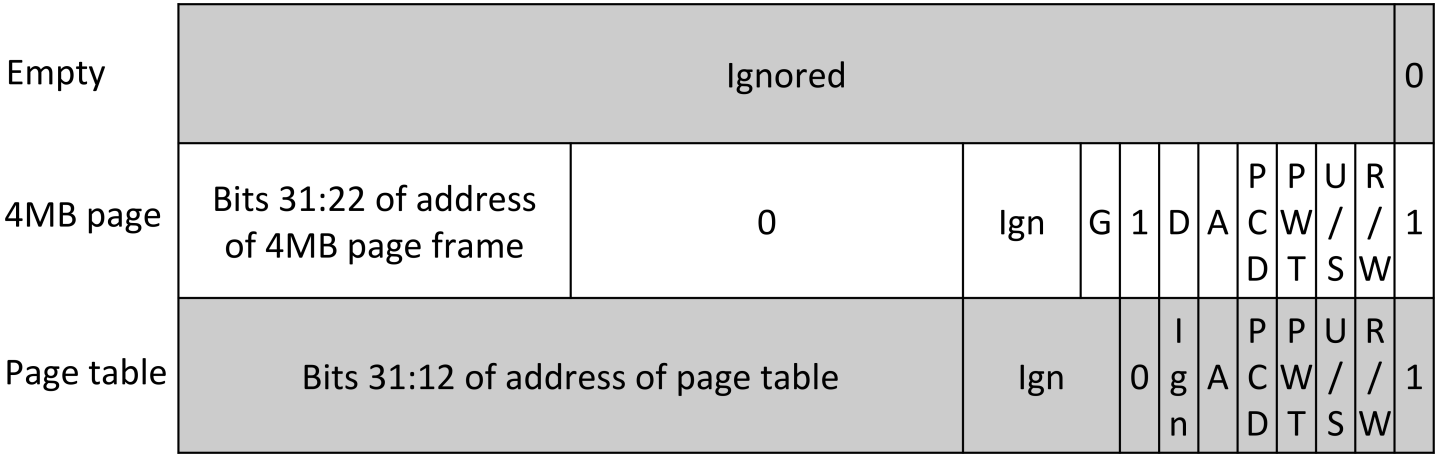
# X86 Virtual Memory Translation



# X86 4KB Page Translation

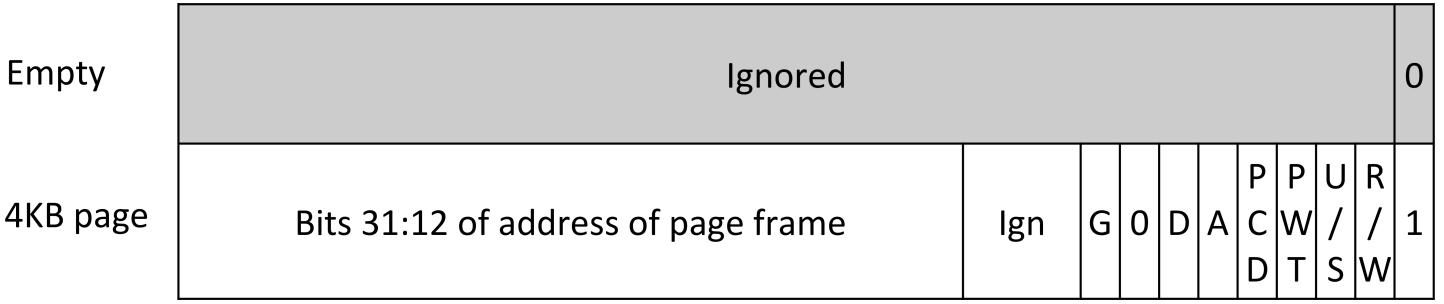


# X86 Page Directory Entry




- Each PD has 1024 entries, 32 bits each
- 4 KBytes total size
- `#define PTE_*` in `mmu.h` for `xv6`

# X86 Page Table Entry




- Each PT has 1024 entries, 32 bits each
- 4 KBytes total size
- `#define PTE_*` in `mmu.h` for xv6

# X86 Translation Lookaside Buffer (TLB)

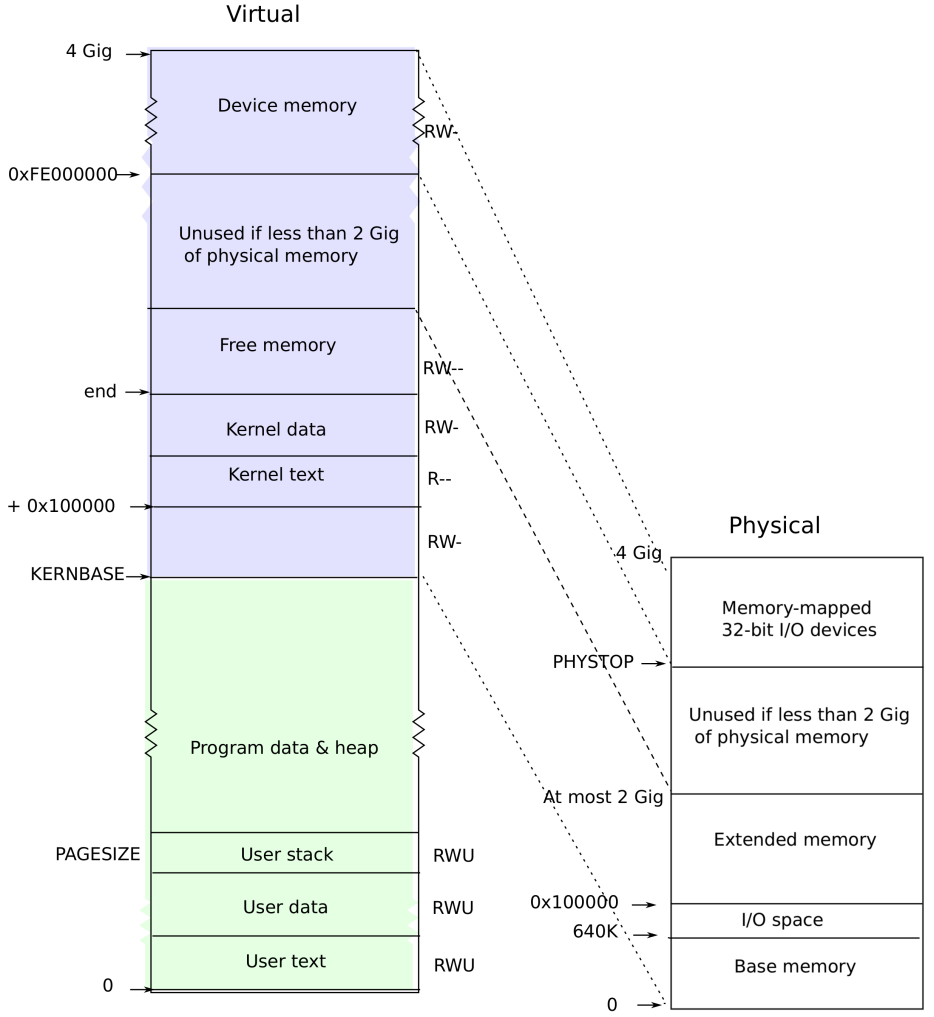
- CPU caches translation results after page walk
  - Cache is partially transparent, adding entries automatically
  - But does not track changes to page table
- Kernel needs to invalidate TLB manually
  - Only required when unmapping or changing permissions
- Two mechanisms for invalidation:
  - Flush: reloading `cr3` (page directory base pointer)
  - `invlpg` instruction invalidates individual page
- Harder with multi-threaded processes on multiple cores

# X86 Page fault

- Causes trap 14
- Also includes error code:

| Bit# | Label | Meaning |
|---|---|---|
| 0 | P | Page table entry was valid |
| 1 | W | Caused by write access |
| 2 | U | Caused by user space access |
| 3 | R | Page table entry with reserved bits set |
| 4 | I | Caused by instruction fetch |

# xv6 Address Space Layout



# Code reading

- Kernel startup
- VM page table manipulation (`vm.c`)
- Page fault handling
- exec implementation

# Problem set: Question 1

- Implement very simple `mmap` and `munmap`

```
mmap(addr, length, rw, fd, offset)
munmap(addr, length)
```

- Map files into memory
  - Applications can read and write file using memory operations
- Implement the simplest case
  - Everything aligned, application picks address, file only mapped in one process
- Don't forget cleanup on `exit`

# Problem set: Question 2

- Add demand paging for `mmap`
- Load pages "lazily" when accessed
  - Speeds up mapping large files
- You'll need to handle (some) page faults
  - And recover from them
- In problem set 4 you'll share files between processes