

# CS3210: Virtual memory

*Taesoo Kim w/ minor updates K. Harrigan*

# Administrivia

- Lab2?
- Lab3 is out!
- (Oct 4) Quiz #1. Lab1-3, Ch 0-3, Appendix A/B
  - We will provide more information on Thursday
- (Oct 6) Time to brainstorm project ideas

# Summary of last lectures

- Power-on → BIOS → bootloader → kernel → init (+ user bins)
- OS: abstraction, multiplexing, isolation
- OS designs: monolithic (xv6) vs. micro kernels (jos)
- Isolation mechanisms: CPL (aka ring), address space (aka process)

# Today: virtual memory

- Isolation: process (unit of isolation)
  - address space
  - virtual memory (today's focus)
- Disclaimer: more than isolation (next lecture)

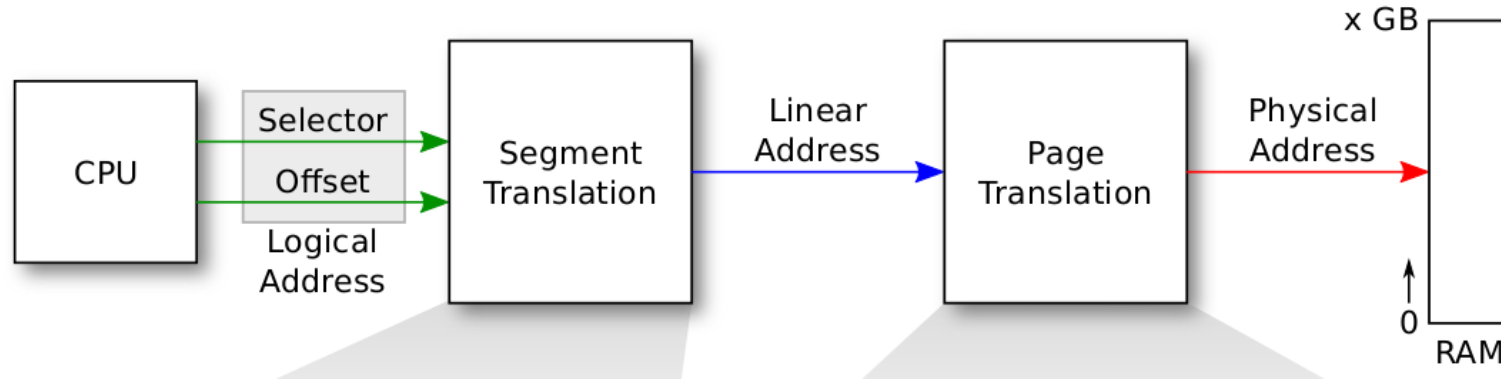
# Basic questions 1

- Q: can we use ring0 for userspace, ring3 for kernel (yes/no)?
- Q: kernel can not read userspace memory (yes/no)?
- Q: userspace can not read kernel's memory (yes/no)?

# Basic questions 2

- Q: pointers in kernel are using physical address (yes/no)?
- Q: processes requiring 3GB memory cannot run on a machine with 2GB RAM (yes/no)?

# Big picture: address translation



- Q: segmentation vs. paging?
- Q: virtual address?

# DEMO: finding physical pages (64-bit Linux)

```
$ ./test
```

```
/proc/24823/pagemap, buf=0x7ffd224c4e90
```

```
$ sudo ./readvirt /proc/24823/pagemap 0x7ffd224c4e90  
0x860000000000ace9, pfn=44265
```

```
$ gdb -c /proc/kcore
```

```
(gdb) x/30x 0xFFFF880000000000 + 44265 * 4096 + 3728
```

```
0xffff88000ace9e90: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
```

```
0xffff88000ace9e98: 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
```

```
0xffff88000ace9ea0: 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17
```

```
0xffff88000ace9ea8: 0x18 0x19 0x1a 0x1b 0x1c 0x1d
```



# DEMO: finding physical pages (32-bit Linux)

```
$ ./test
```

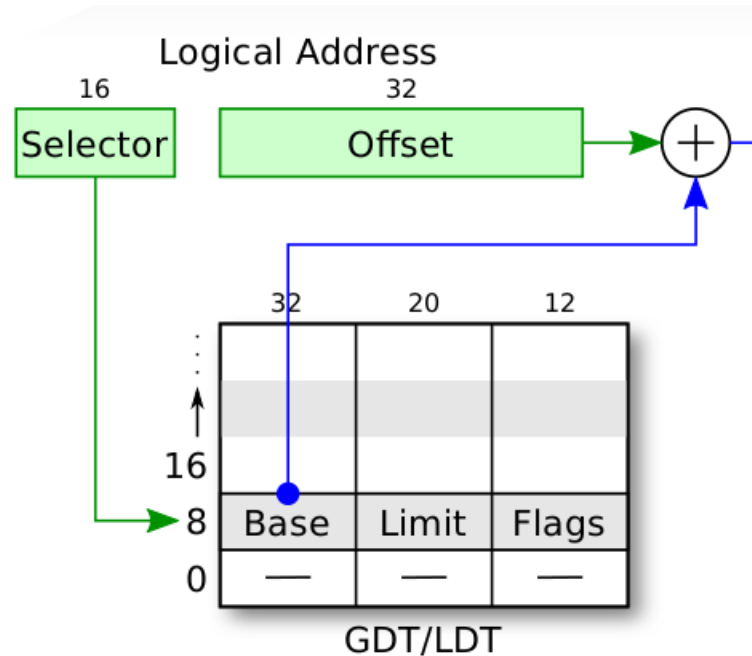
```
/proc/30118/pagemap, buf=0xbff4bfdc, main=0x80484cd
```

```
$ sudo ./read_virt /proc/30118/pagemap 0xbff4bfdc  
0x86000000000070ba, pfn=28858
```

```
$ sudo gdb -c /proc/kcore
```

```
(gdb) x/30b 0xc0000000 + 28858*4096 + 4060  
0xc70bafdc: 0  1  2  3  4  5  6  7  
0xc70bafec: 8  9 10 11 12 13 14 15  
0xc70bafec: 16 17 18 19 20 21 22 23  
0xc70baff4: 24 25 26 27 28 29
```

# Segmentation



- Q: Flags?, Selector?
- Q: What's output?

# Code: segmentation in xv6/bootloader

```
# Bootstrap GDT
.p2align 2                                # force 4 byte alignment
gdt:
    SEG_NULLASM                            # null seg
    SEG_ASM(STA_X|STA_R, 0x0, 0xffffffff) # code seg
    SEG_ASM(STA_W, 0x0, 0xffffffff)      # data seg
```

- Q: how do we specify to use code/data segments?

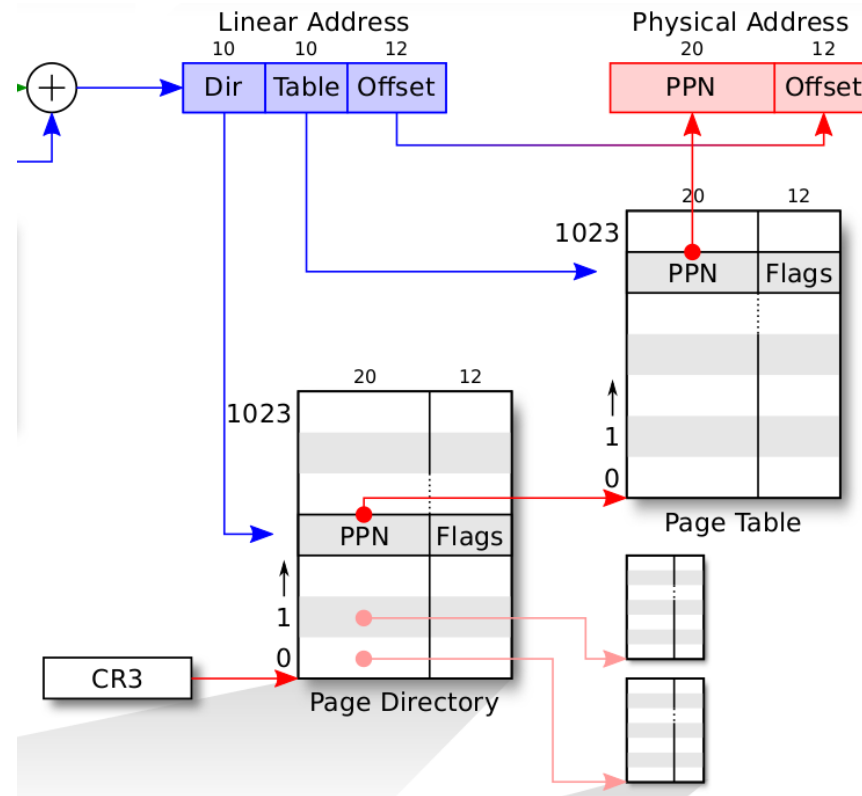
# Code: segmentation in xv6

```
void seginit(void) {
    struct cpu *c = &cpus[cpunum()];
    c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_KERN);
    c->gdt[SEG_KDATA] = SEG(STA_W, 0, 0xffffffff, DPL_KERN);
    c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
    c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);

    lgdt(c->gdt, sizeof(c->gdt));
    ...
}
```

- Q: what happens if userspace ptr points to 0xff000000?

# Page Translation



# Page Translation

- Q: CR3 should be virtual address? (yes/no)
- We divide a 32 bit address into [dir=10|tbl=10|off=12]
  - [dir=05|tbl=15|off=12]?
  - [dir=15|tbl=05|off=12]?
  - [dir=10|tbl=00|off=22]?
  - [dir=00|tbl=20|off=12]?

# Single-level paging [dir=10|tbl=00|off=22]

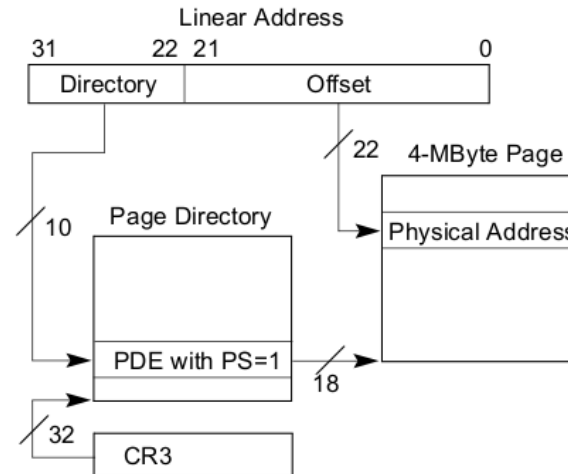


Figure 4-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

- Q: page size?
- Q: trade-off (better/worse)?

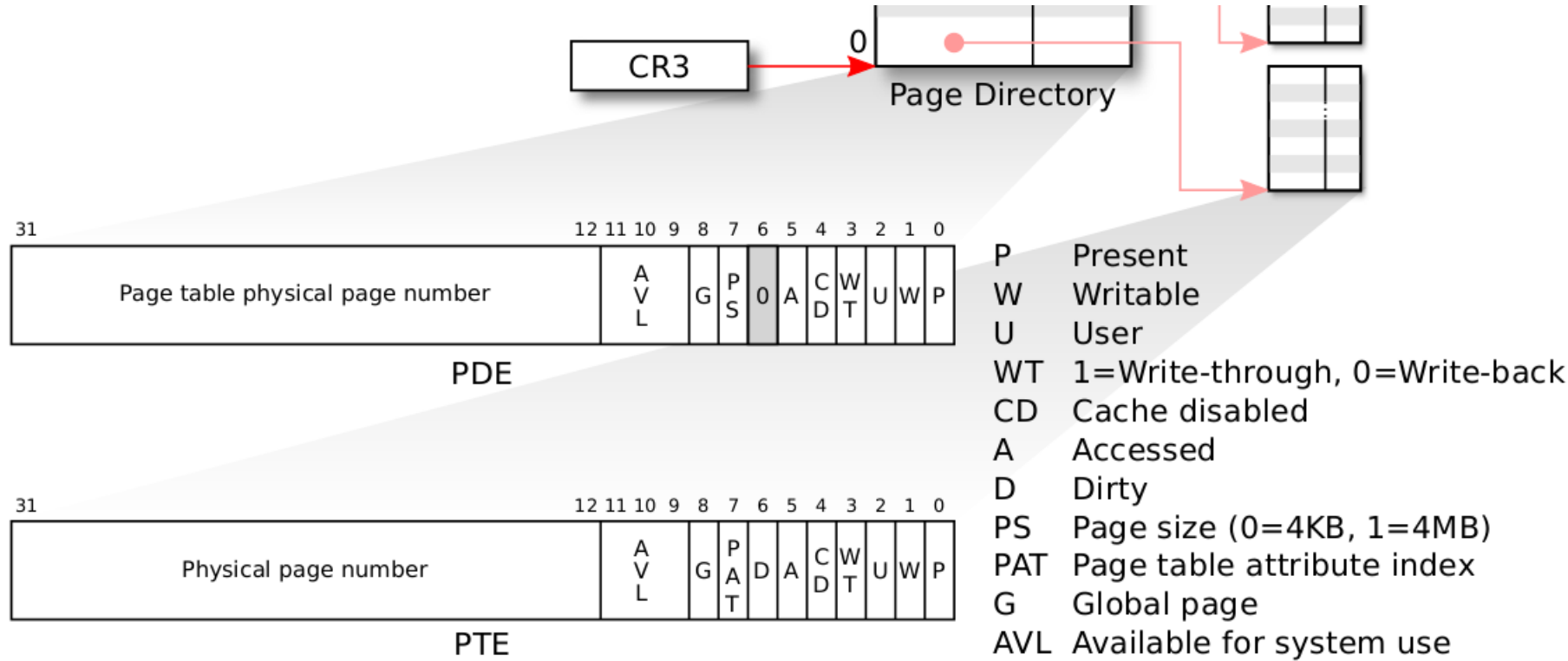
# Single-level paging in xv6

```
// main.c
__attribute__((__aligned__(PGSIZE)))
pde_t entrypgdir[NPDENTRIES] = {
    // Q?
    [0] = (0) | PTE_P | PTE_W | PTE_PS,
    // Q?
    [KERNBASE>>PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
};
```

- DEMO: write to `0x1000` and read from `0x1000 + KERNBASE`



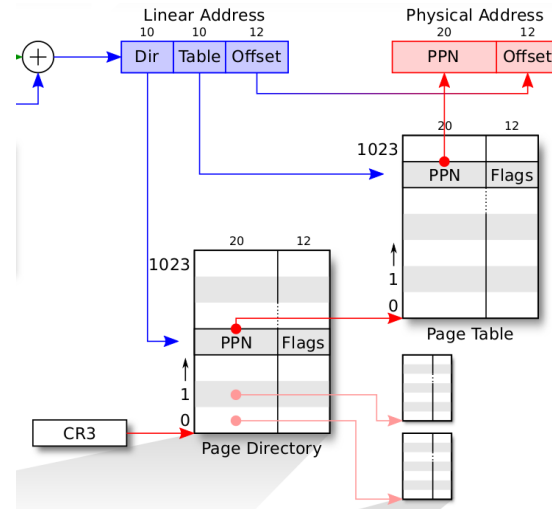
# PTE/PDE



# PTE/PDE: how to interpret?

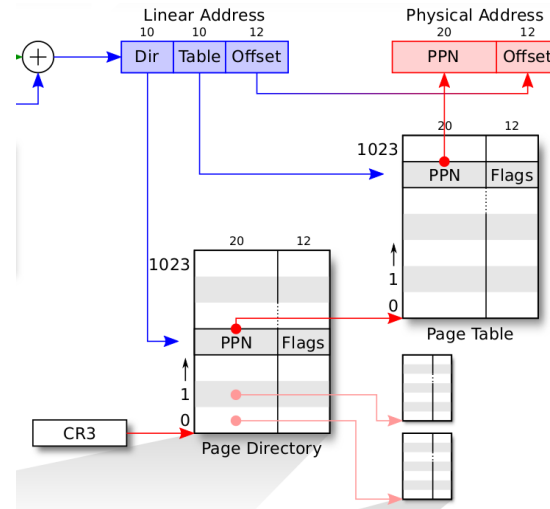
```
// main.c
__attribute__((__aligned__(PGSIZE)))
pde_t entrypgdir[NPDENTRIES] = {
    // Q?
    [0] = (0) | PTE_P | PTE_W | PTE_PS,
    // Q?
    [KERNBASE>>PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
};
```

# Multi-level paging



- Q: why not just 4K single-level paging?
  - i.e.,  $2^{20} \times \text{size(pte)} = ?? \text{ MB}$
- Q: why problematic?

# Page Translation (two-level)



- Q: given a virtual address, how many memory lookups to translate it to a physical address (aka page walk)?

# Four-level paging in x86-64

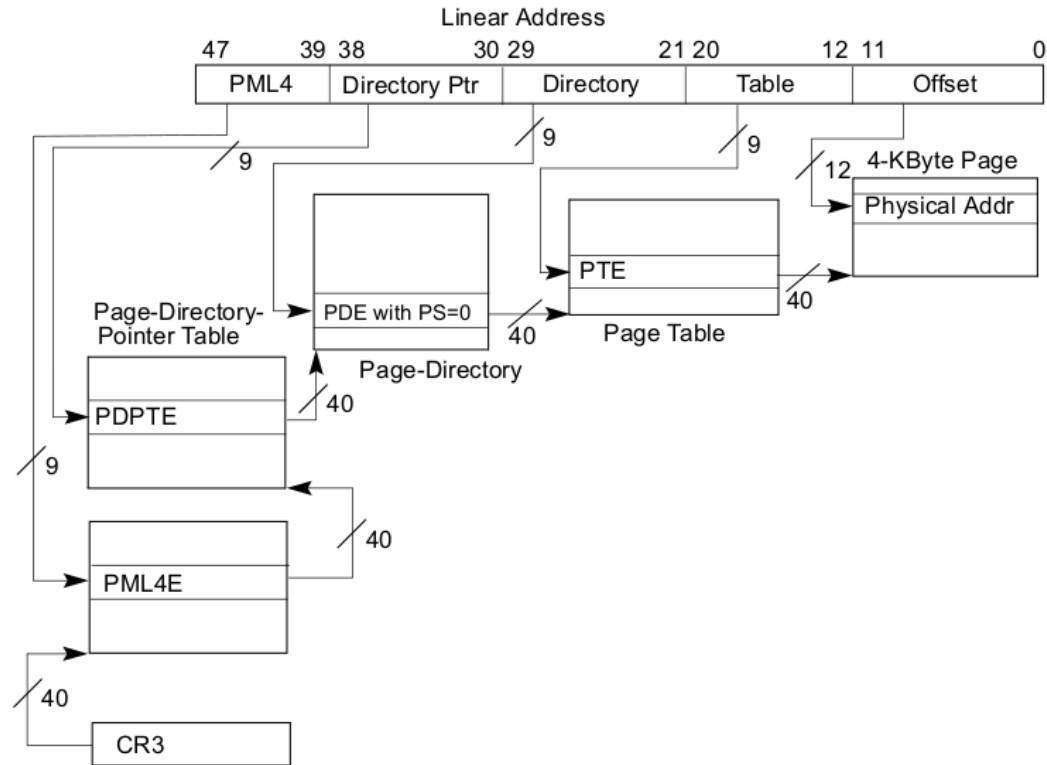
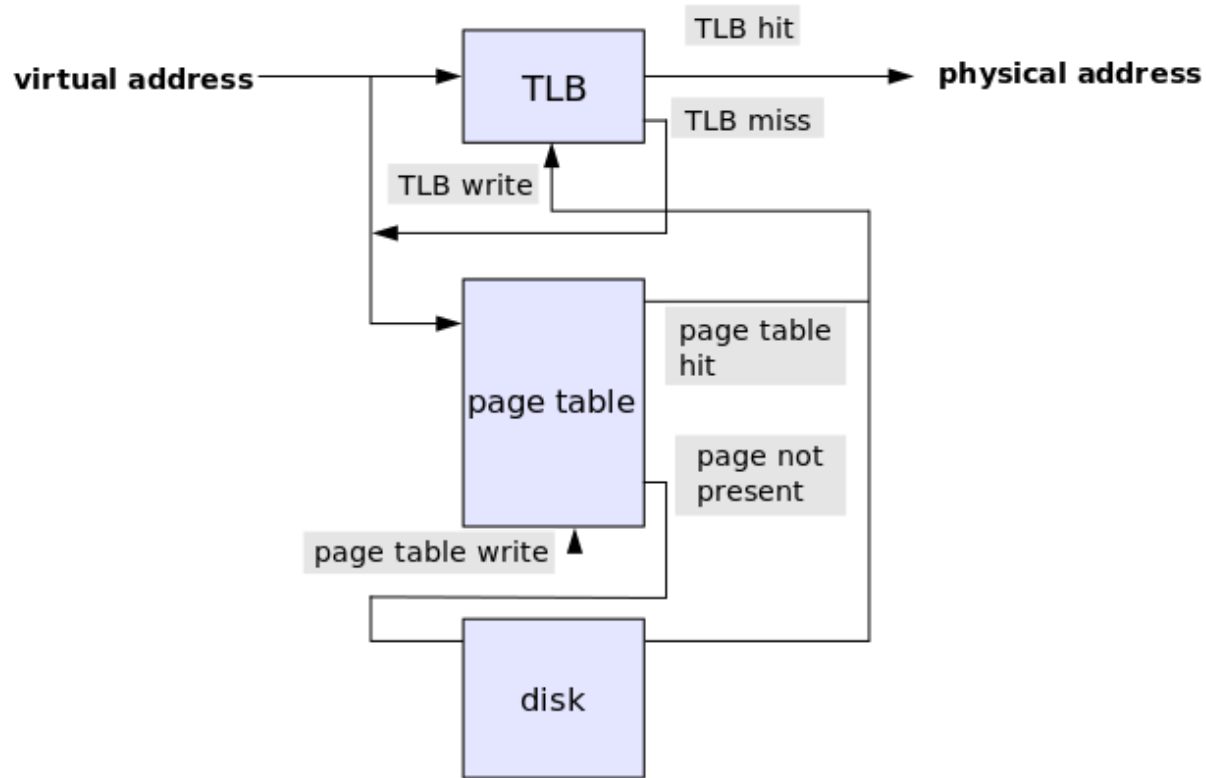


Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

# MMU (memory management unit)

- Hardware support: VA → PA translation
- Cache: TLB (translation lookaside buffer)
  - Caching mappings: virtual → physical addresses
  - Optimization: iTLB, dTLB

# Paging with TLB



# DEMO: how many TLB misses/hits?

```
$ perf stat -e iTLB-load-misses,iTLB-loads -- ls
```

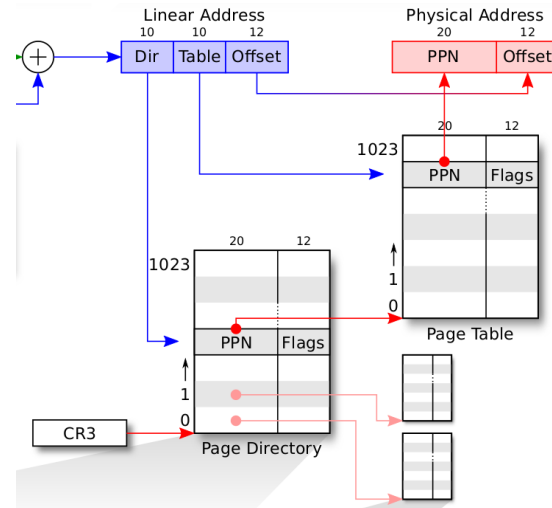
```
...
```

```
$ perf stat -e iTLB-load-misses,iTLB-loads -- gvim
```

```
...
```



# Q: what if PTE is not present (P)?



- Q: what if we fail to translate? (next week)

# DEMO: how many page faults?

```
$ perf stat -- ls  
...  
$ perf stat -- gvim  
...
```

# So, why paging is good?

# So, why paging is good?

- Primary purpose: isolation
  - each process has its own address space
- Benefits:
  - memory utilization, fragmentation, sharing, etc.
- Level-of-indirection
  - provides kernel with opportunity to do cool stuff (example?)

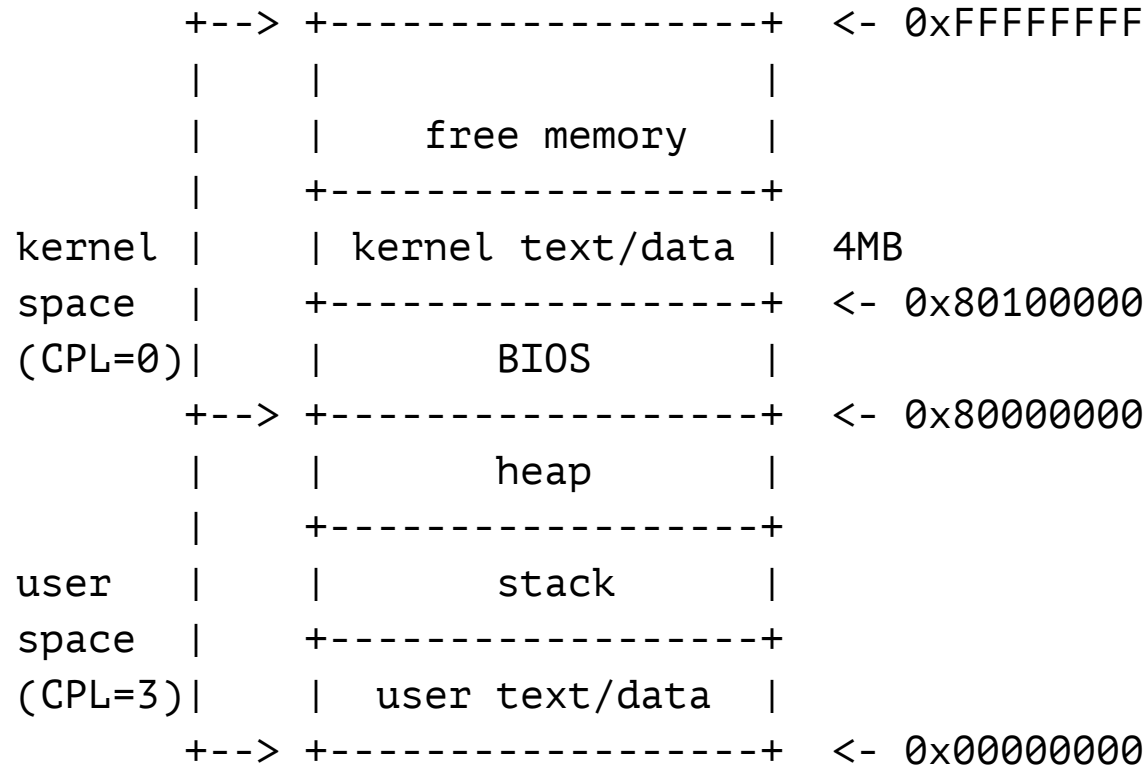
# Potential applications

- Kernel tricks (e.g., one zero-filled page)
- Faster system calls (e.g., copy-on-write fork)
- New features (e.g., memory-mapped files)
- **NOTE** : project idea?

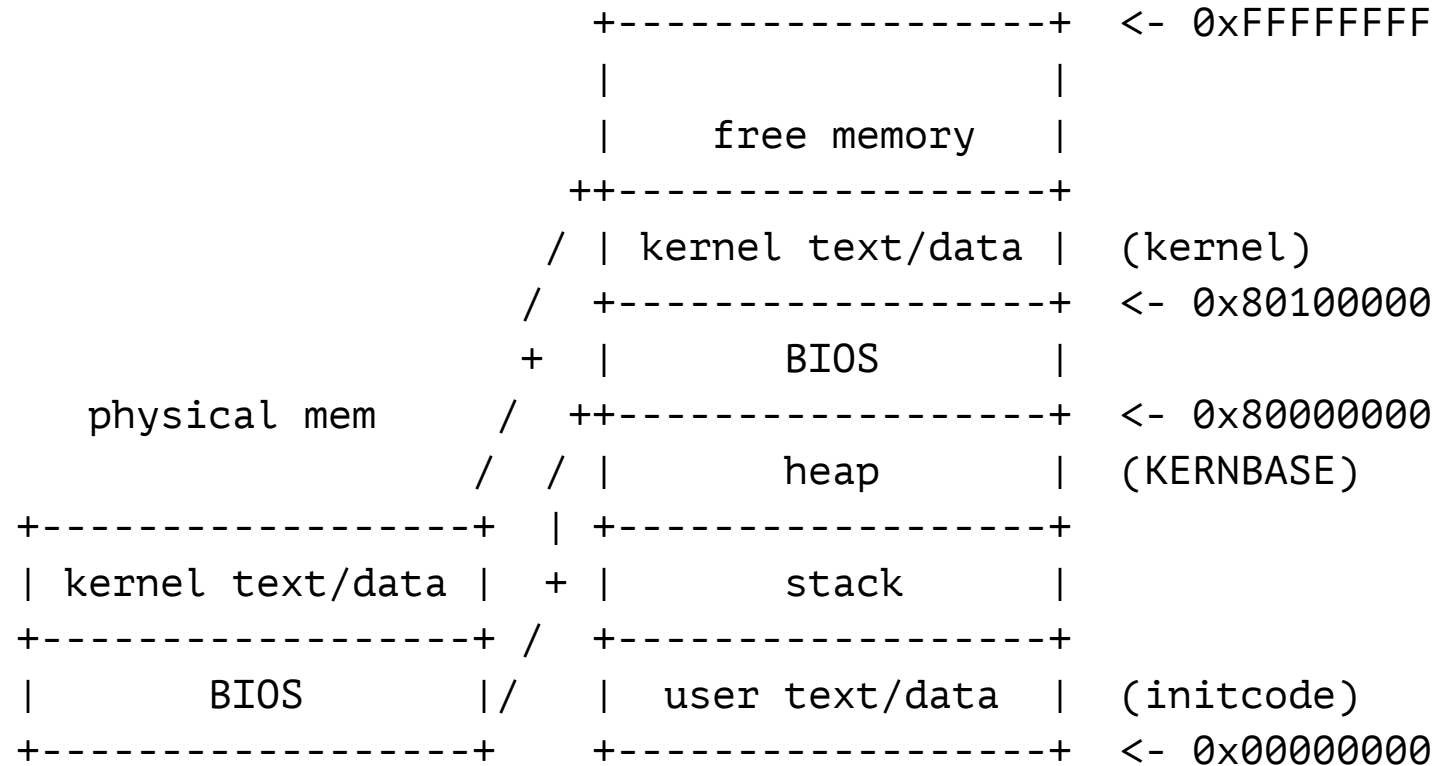
# Code: paging in xv6

- `kinit1()` in `main.c`
- `kvmalloc()` in `main.c`

# Virtual address space in xv6



# The first address space in xv6





# References

- [Intel Manual](#)
- [UW CSE 451](#)
- [OSPP](#)
- [MIT 6.828](#)
- Wikipedia
- The Internet