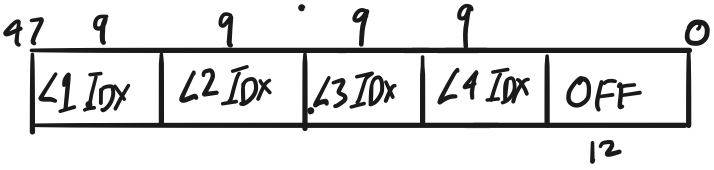
# CS202: Virtual Memory 3

- Last Class
  - Walking the Page Tables



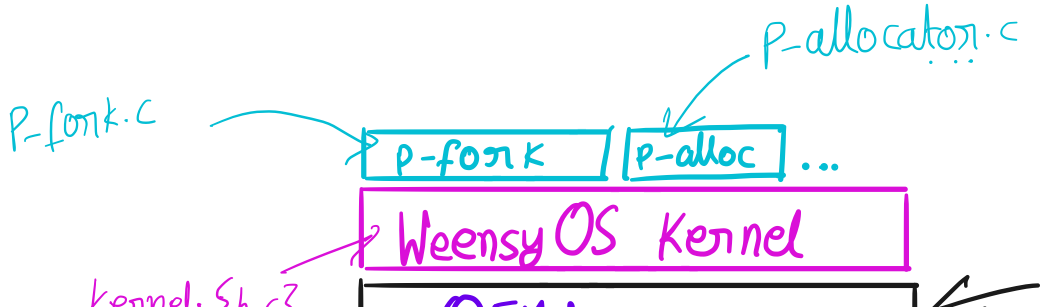

  - TLB: Translation Look Aside Buffer
    - ↳ Cache Past Translation Results
    - → Save Time!

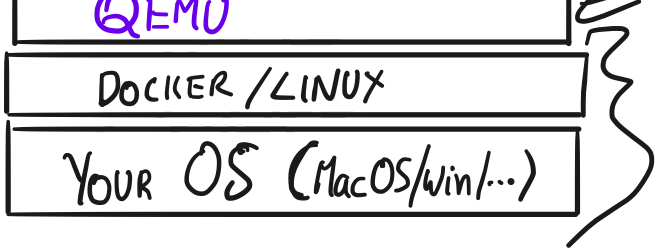
## Today

- Weensy OS (Lab4)
- Page Faults & Their Uses.

## Weensy OS

- Start Early ⟷ Don't
- Review Session: ~~Wed~~ 7:15 PM 10/31
- Goal Here: Provide Some Context.






# PROCESS CONTROL BLOCK (PCB)

struct proc (kernel.h)

↳ ... x86-64_pagetable* p_pagetable

↑ What you will often be manipulating

# Metadata to Track Physical Pages

physical_pageinfo pageinfo[0x20051 2] (kernel.c)

sys_page_alloc

assert

↳ owner → FREE
RESERVED
KERNEL

↳ refcnt → How many page tables map this page.

Manipulated by assign_physical_page.

# MANIPULATING VIRTUAL ADDRESSES

- FROM USERSPACE:

sys_page_alloc(void* addr) [process.h]
↳ Allocate a page at VA addr

- From Kernel à

virtual-memory-map ( x86-64... * page table,

uintptr_t va, → 0x401...

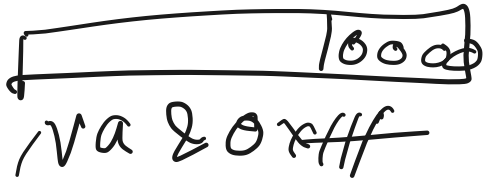
uintptr_t pa, ↳

size_t sz, → 0x1000

int perm,

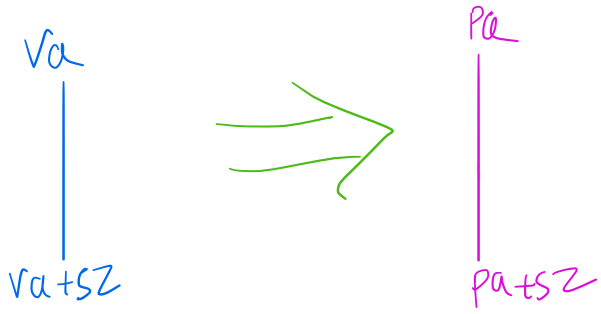
function pointer to allocator )

"Page Aligned"



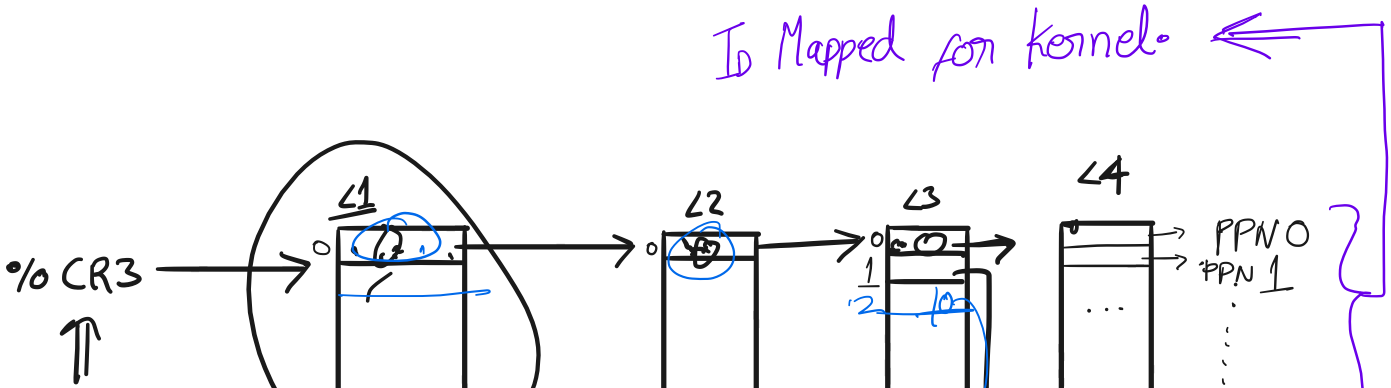

Goal:

Map

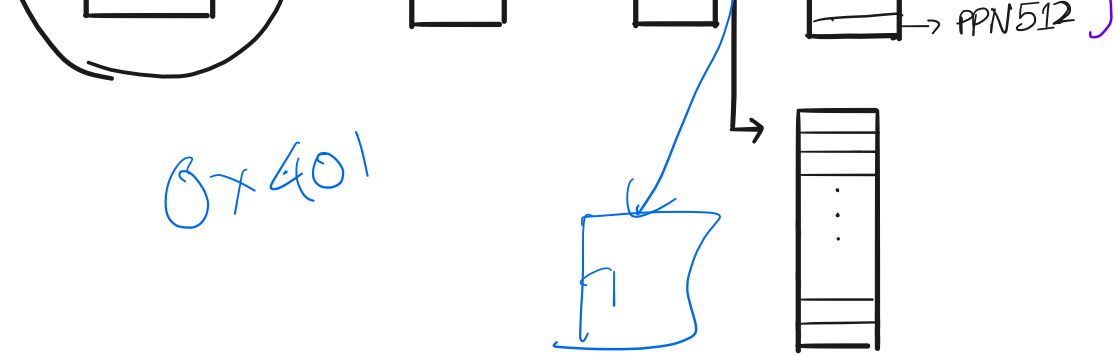



Assumption: Caller has already checked [pa... pa+sz] can be safely mapped.

How Is The Allocator Used?



PA for proc → p-pagetable

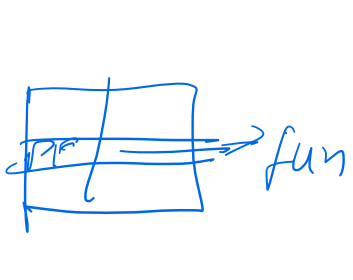


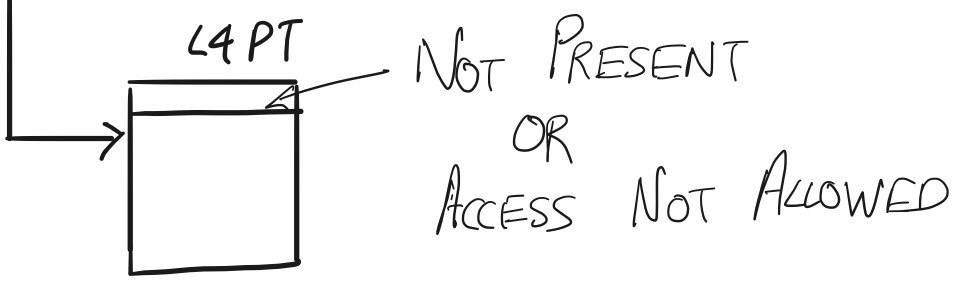

(ASSUMING YOU COPY INITIALIZATION FROM kernel-pagetable)

Q. Want to map a page at virt. address 0x401000 (VPN 1025)?

## BACK TO PAGE FAULTS

movq 0x401008, %rax

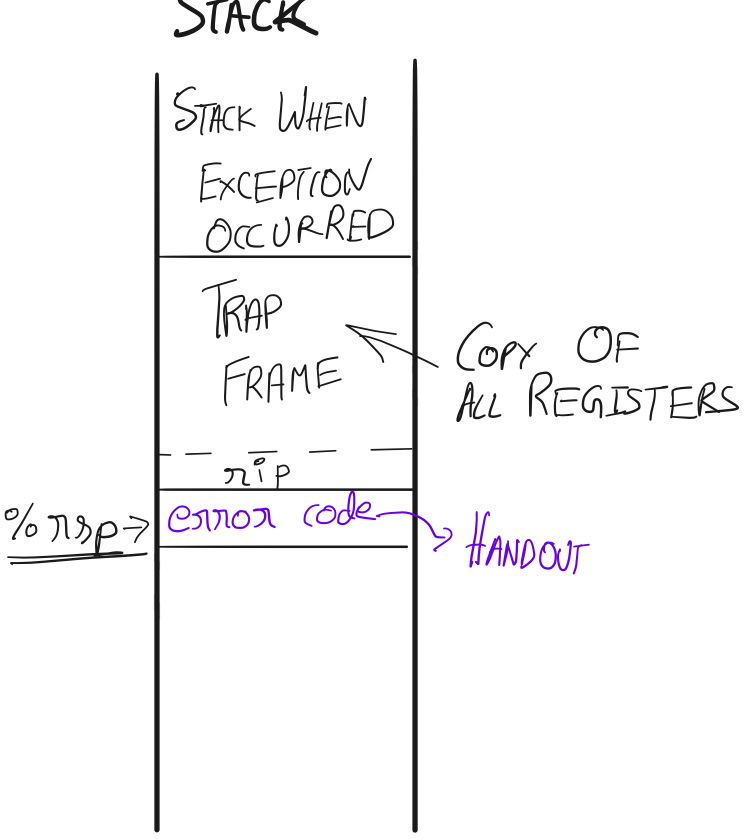



## PROCESSOR THROWS AN EXCEPTION (TRAP)

%RIP → trap handler (in kernel)

%CR2 → ADDRESS WHICH CAUSED FAULT (0x4010008)



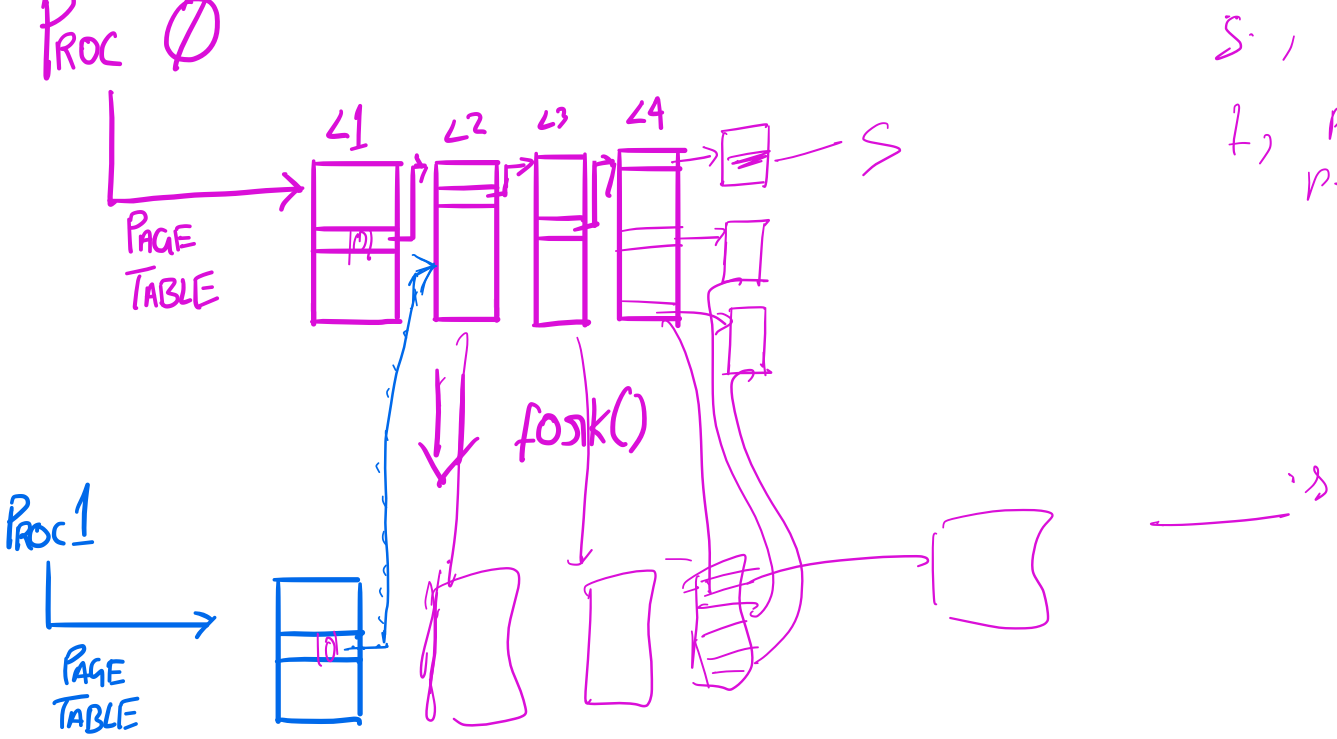

## HOW USED

(1) DEMAND PAGING TO OVERCOMMIT MEMORY

(NEXT CLASS)

(2) COPY ON WRITE.

vm (proc-ng)



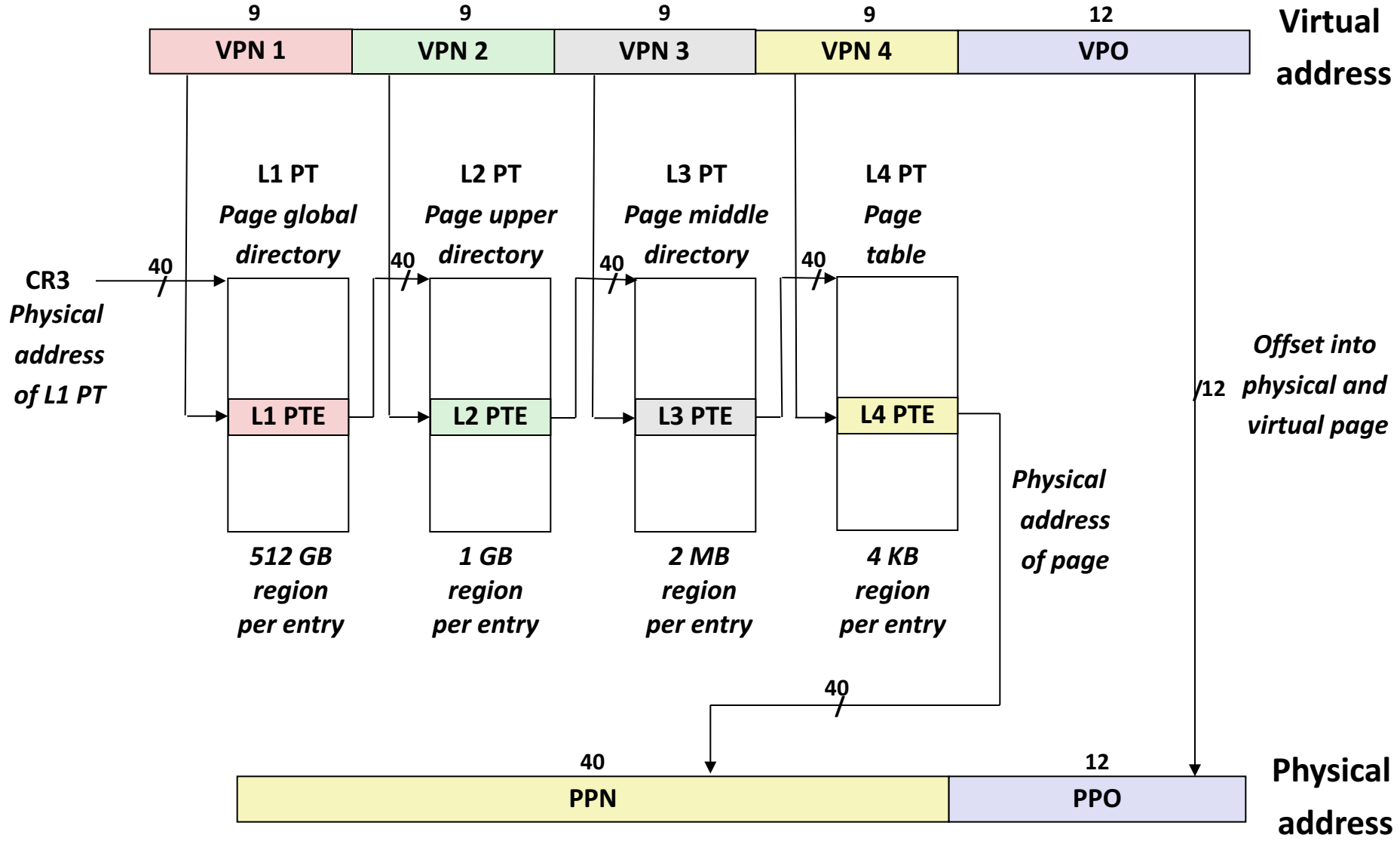

CORE IDEA

- DELAY ACTUALLY COPYING UNTIL NECESSARY
- WHEN NECESSARY?
  - ON MODIFICATION

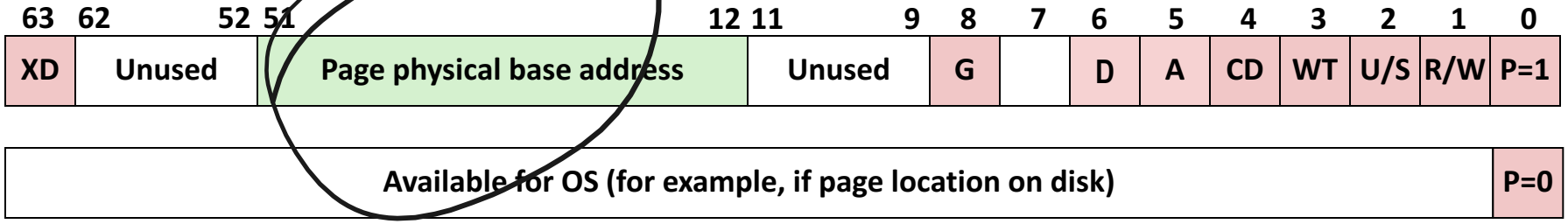
③ ACCOUNTING

# Linux VA Layout

# Core i7 Page Table Translation



# Core i7 Level 4 Page Table Entries




**Each entry references a 4K child page. Significant fields:**

**P:** Child page is present in memory (1) or not (0)

**R/W:** Read-only or read-write access permission for this page

**U/S:** User or supervisor mode access

**WT:** Write-through or write-back cache policy for this page

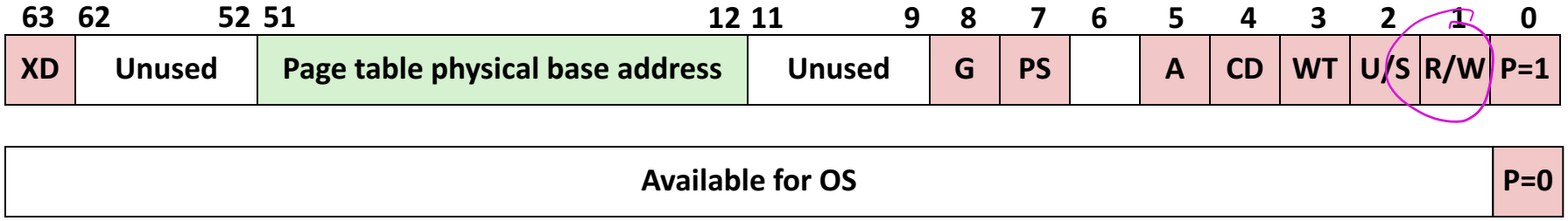
**A:** Reference bit (set by MMU on reads and writes, cleared by software)

**D:** Dirty bit (set by MMU on writes, cleared by software)

**Page physical base address:** 40 most significant bits of physical page address (forces pages to be 4KB aligned)

**XD:** Disable or enable instruction fetches from this page.

# Core i7 Level 1-3 Page Table Entries




## Each entry references a 4K child page table. Significant fields:

**P:** Child page table present in physical memory (1) or not (0).

**R/W:** Read-only or read-write access access permission for all reachable pages.

**U/S:** user or supervisor (kernel) mode access permission for all reachable pages.

**WT:** Write-through or write-back cache policy for the child page table.

**A:** Reference bit (set by MMU on reads and writes, cleared by software).

**PS:** Page size: if bit set, we have 2 MB or 1 GB pages (bit can be set in Level 2 and 3 PTEs only).

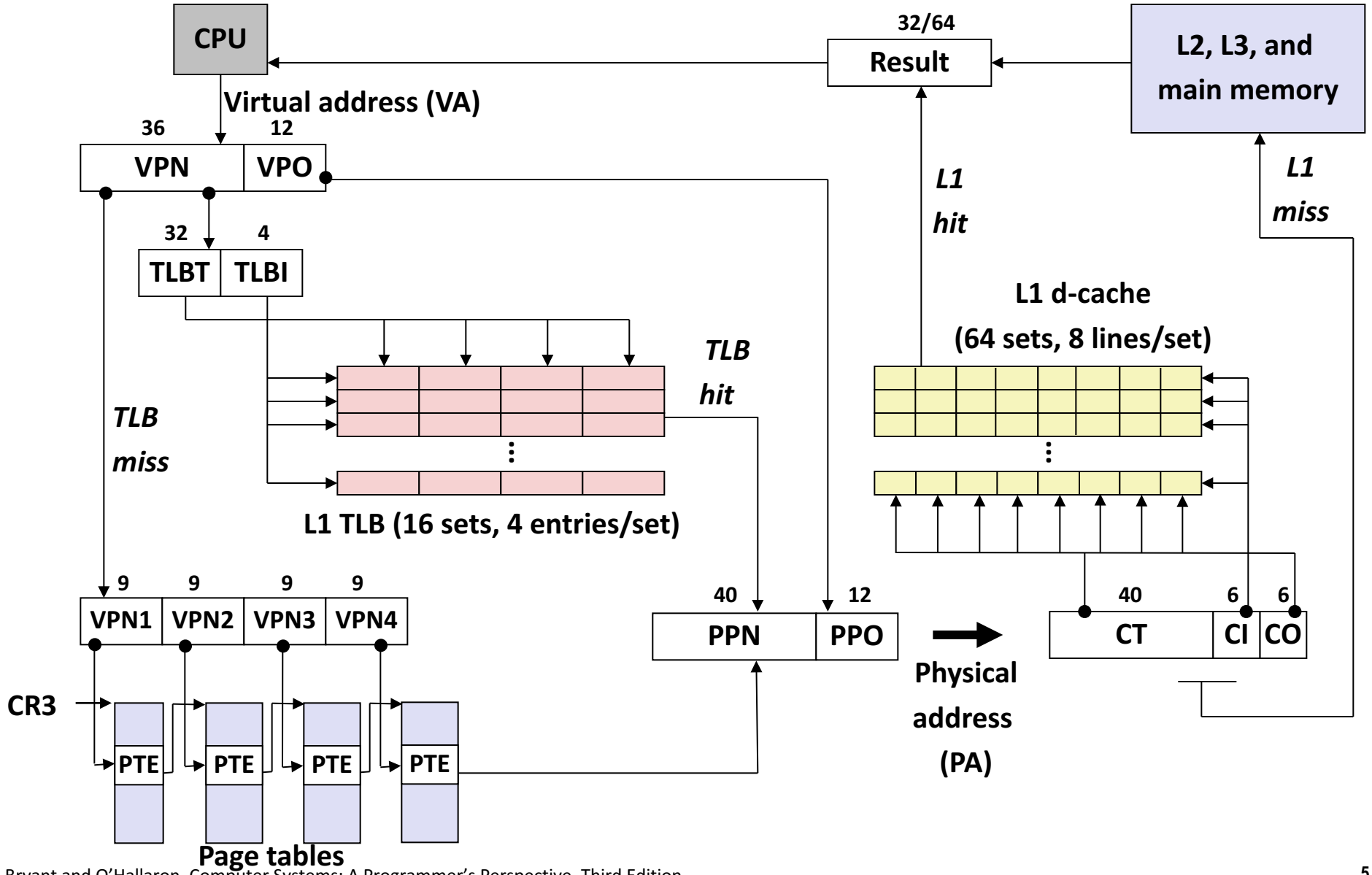
**Page table physical base address:** 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

**XD:** Disable or enable instruction fetches from all pages reachable from this PTE.

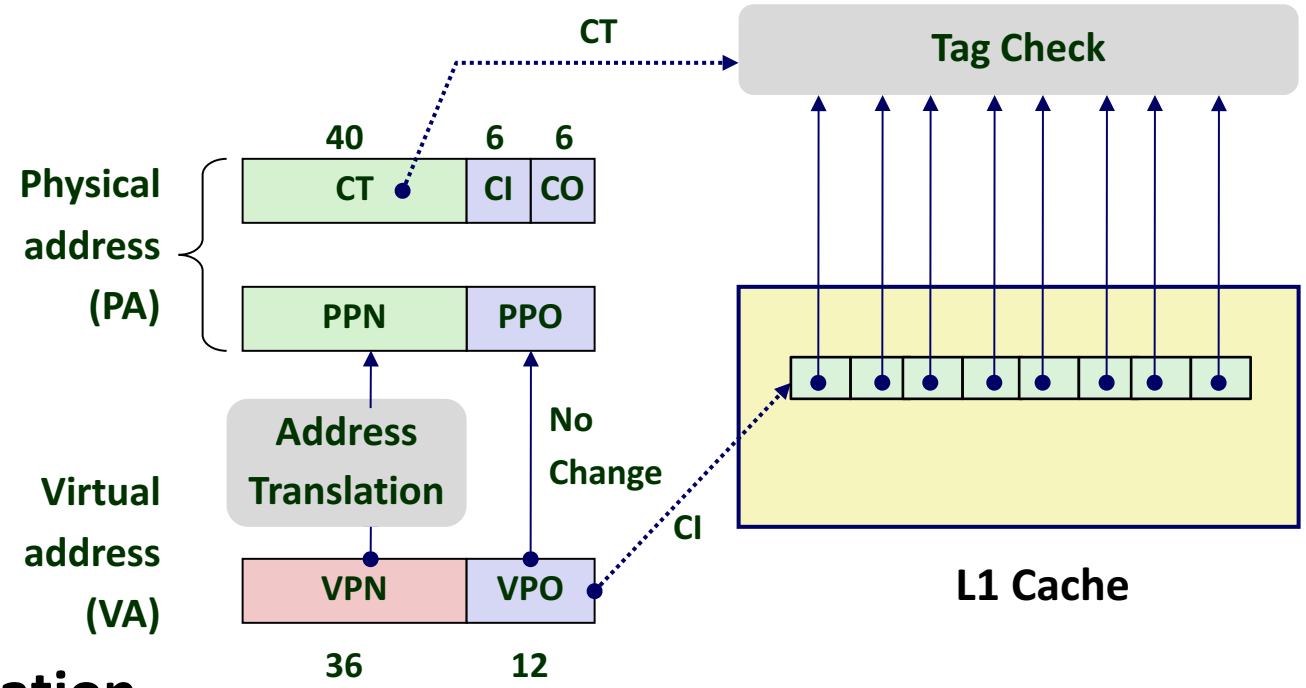

| | | |
|---|---|---|
| P | 0 | The fault was caused by a non-present page. |
| | 1 | The fault was caused by a page-level protection violation. |
| W/R | 0 | The access causing the fault was a read. |
| | 1 | The access causing the fault was a write. |
| U/S | 0 | A supervisor-mode access caused the fault. |
| | 1 | A user-mode access caused the fault. |
| RSVD | 0 | The fault was not caused by reserved bit violation. |
| | 1 | The fault was caused by a reserved bit set to 1 in some paging-structure entry. |
| I/D | 0 | The fault was not caused by an instruction fetch. |
| | 1 | The fault was caused by an instruction fetch. |
| PK | 0 | The fault was not caused by protection keys. |
| | 1 | There was a protection-key violation. |
| SGX | 0 | The fault is not related to SGX. |
| | 1 | The fault resulted from violation of SGX-specific access-control requirements. |

Figure 4-12. Page-Fault Error Code

# End-to-end Core i7 Address Translation



# Cute Trick for Speeding Up L1 Access




- **Observation**
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation taking place
  - Cache carefully sized to make this possible: 64 sets, 64-byte cache blocks
  - Means 6 bits for cache index, 6 for *cache* offset
  - That's 12 bits; matches *VPO, PPO* → One reason pages are $2^{12}$ bits = 4 KB

# Virtual Address Space of a Linux Process