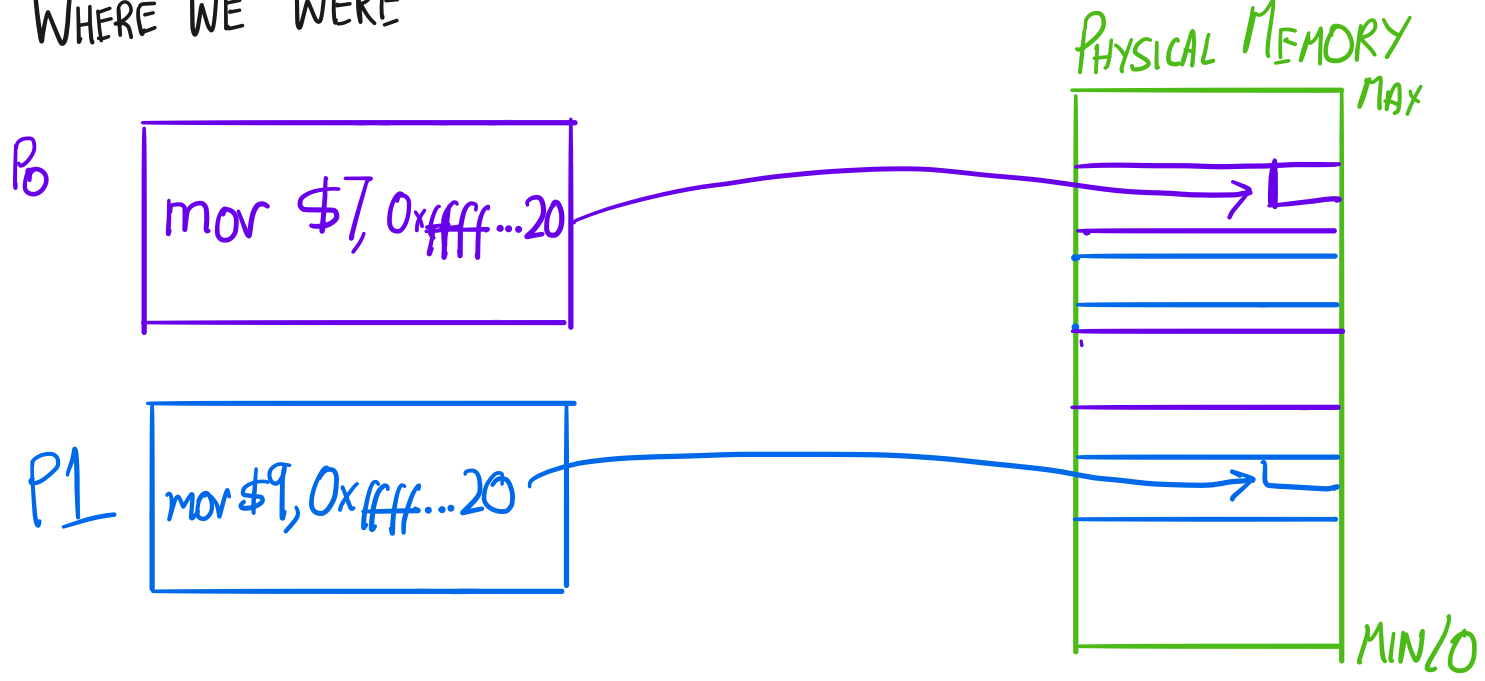
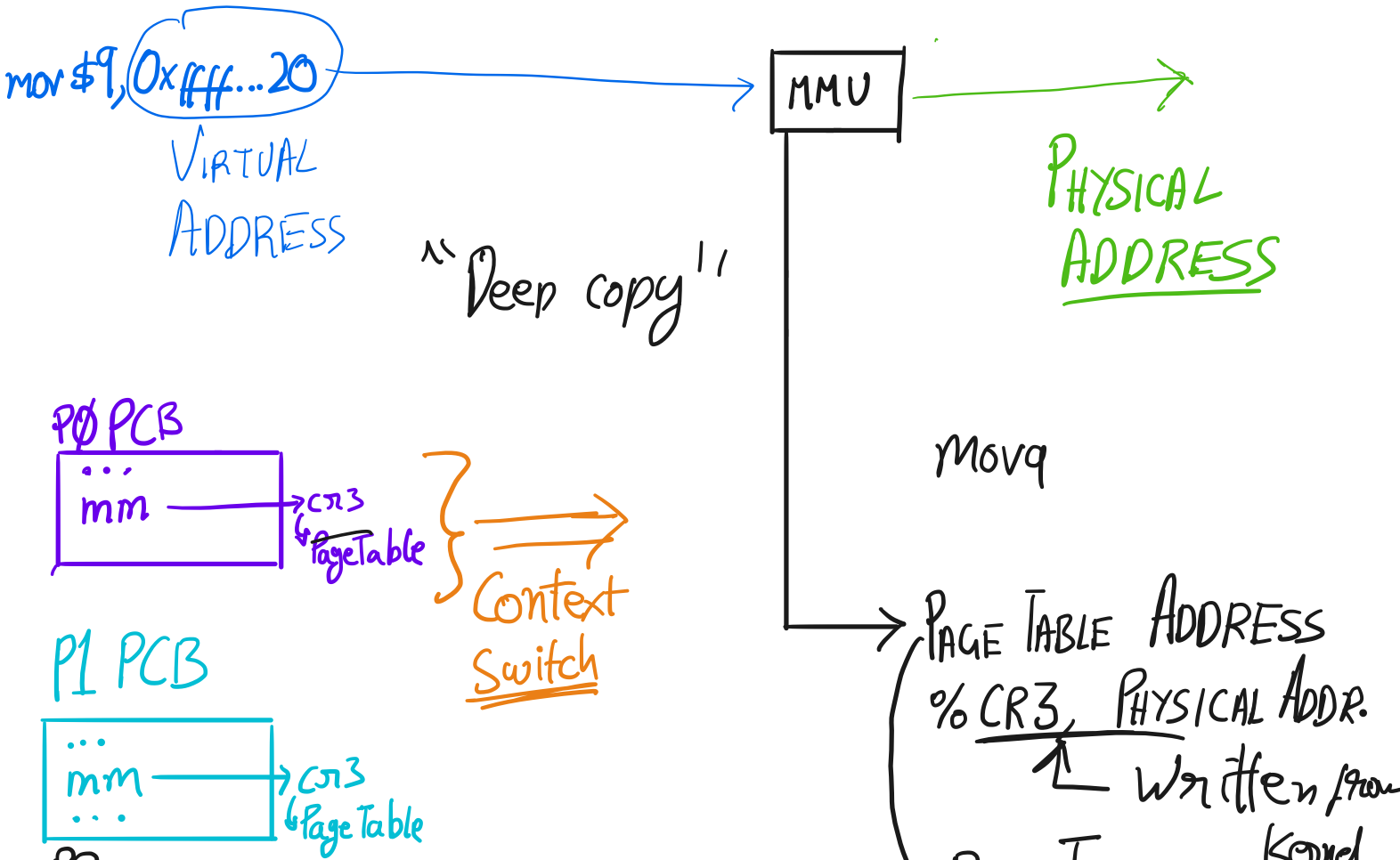
CS202 – Virtual Memory

Where We Were



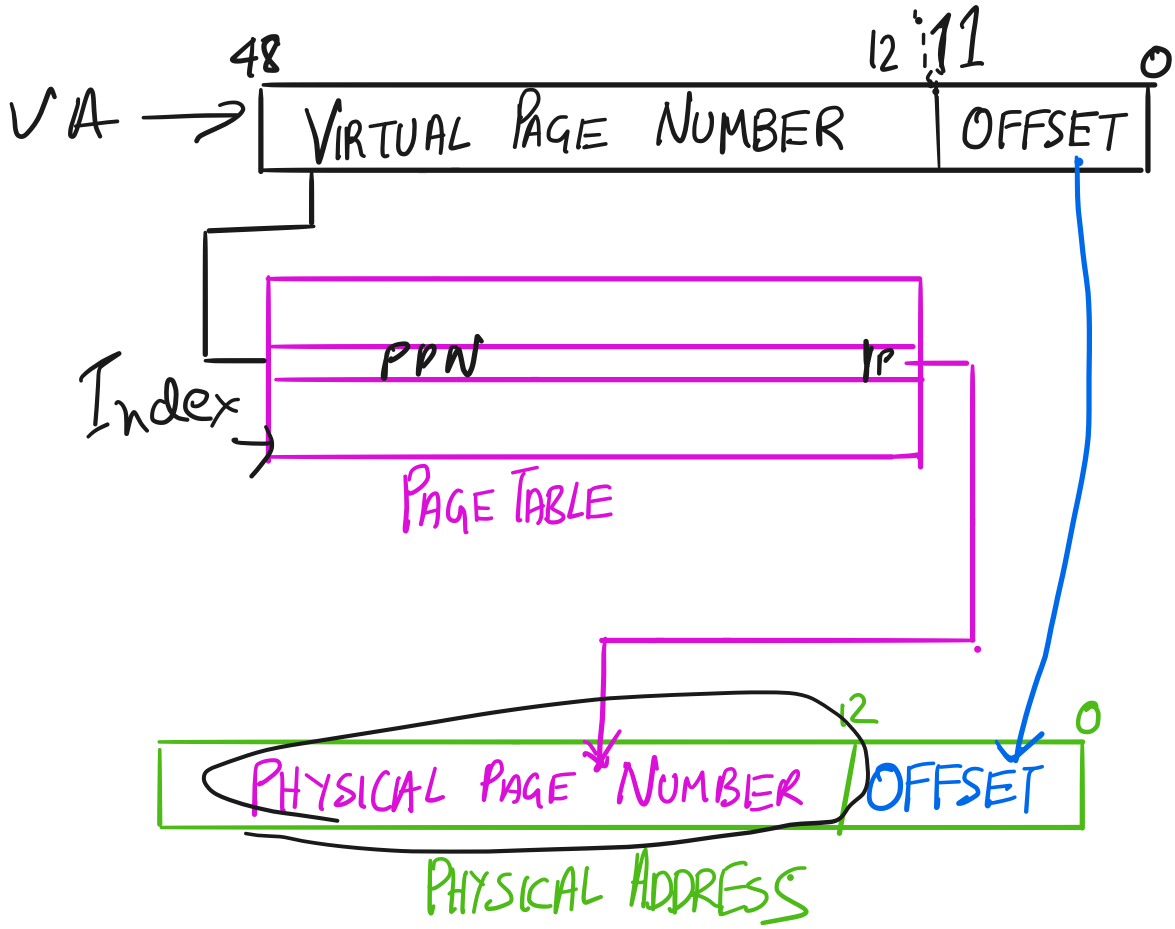
How?



mm

Today: Page Tables

Virtual Address (For Now Assume 48-Bits)



Note Physical & Virtual Addresses Might Have Different Lengths

AMD64 (some):- Virtual 48 bits
Physical 52 bits

Physical

NOTE REMAINING 64-48 ≡ 16 bits?

ALL Ø OR ALL 1

PAGE TABLES

CORE PROBLEM: LOTS OF UNUSED VIRTUAL ADDRESES

$2^{36} \equiv$ DON'T WANT TO WASTE SPACE.

SOLUTION: BUILD A TREE

(SWITCH TO SHEET)

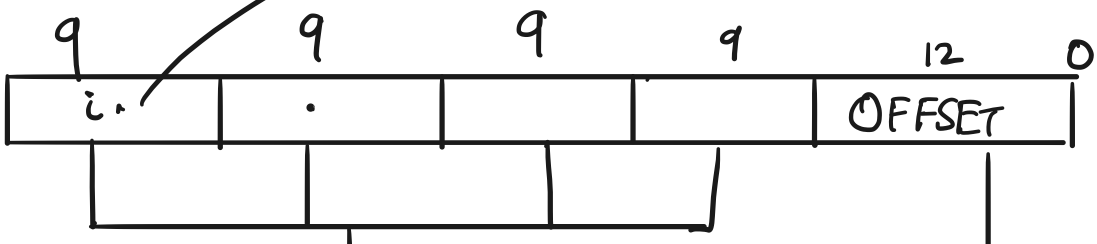
WHAT IS IMPORTANT

- FAULTS: NOT PRESENT OR
  VIOLATE ACCESS CONTROL PERMISSIONS

- How to translate virtual address
  - VPN, offset
  - Use VPN to walk page table
- Why the tree helps.

uint64_t a[512]

Resolving addresses — a[i]



- Red → 4 memory accesses to locate PPN
  + 1 actual access

- 5x extra accesses

- Want to reduce overheads.
  - ↳ Cache recent VPN → PPN

TRANSLATION LOOKASIDE BUFFER

TRANSLATIONS

→ SIZE: DICTATED BY HARDWARE

USUALLY B/W 64 – 1024 entries

- ON MISS?

TLB QUESTION

16 ENTRIES.

PROGRAM THAT (AFTER INITIAL START) MISSES TLB ON EACH INSTRUCTION?

a [4096 x 17]

~~for~~ int i=0; P''

while (1)

puts(a [i]);

i += 4096.

Process Memory Layout.

2^0 Byte

2^{10} Kilobyte (1024)

2^{20} MEGA BYTE

2^{30} GIGA BYTE

...

$2^3, 2^9$ $\angle$ 1 PT

↳ Entries

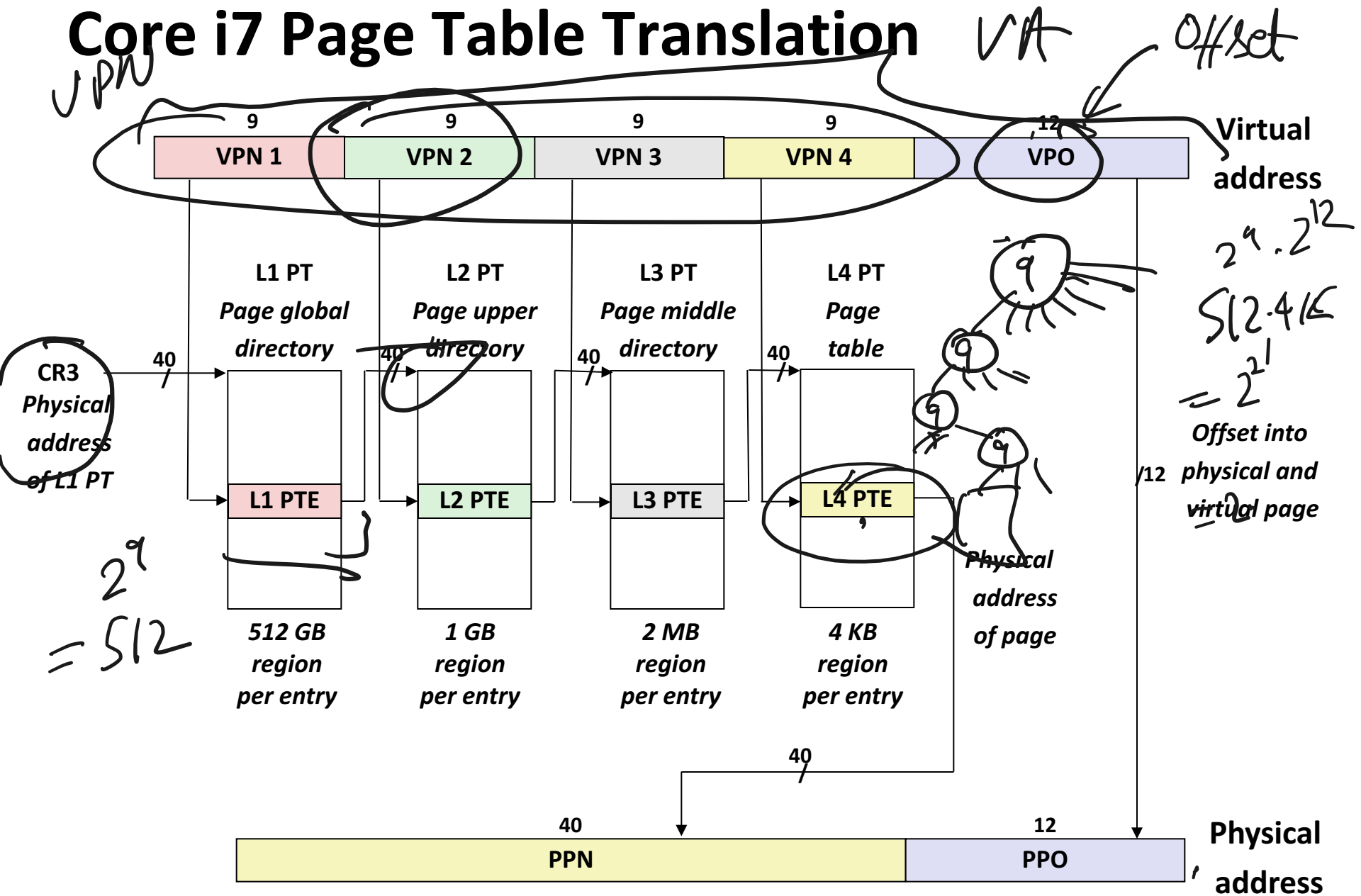
12 bits

$= 2^{12}$

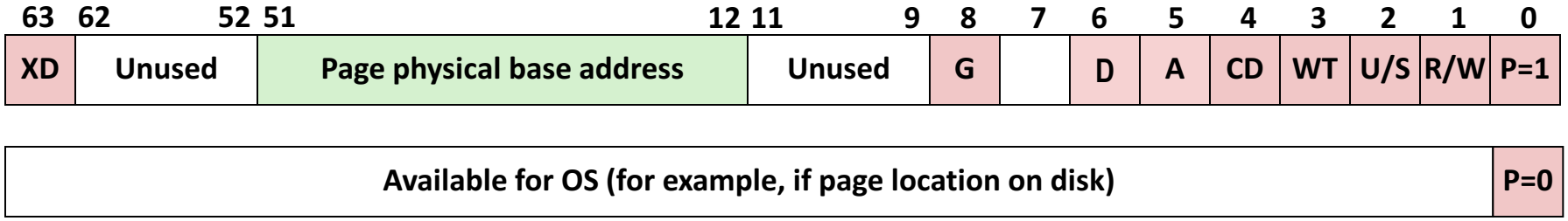
$= 2^2 \cdot 2^{10}$

$= 4K$

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

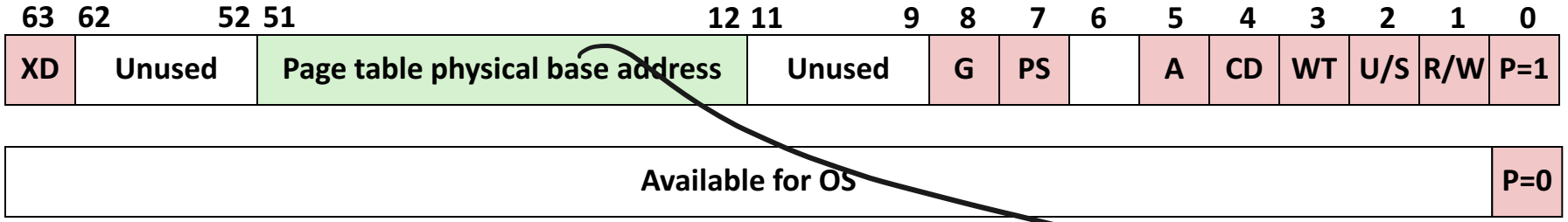
A: Reference bit (set by MMU on reads and writes, cleared by software)

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).

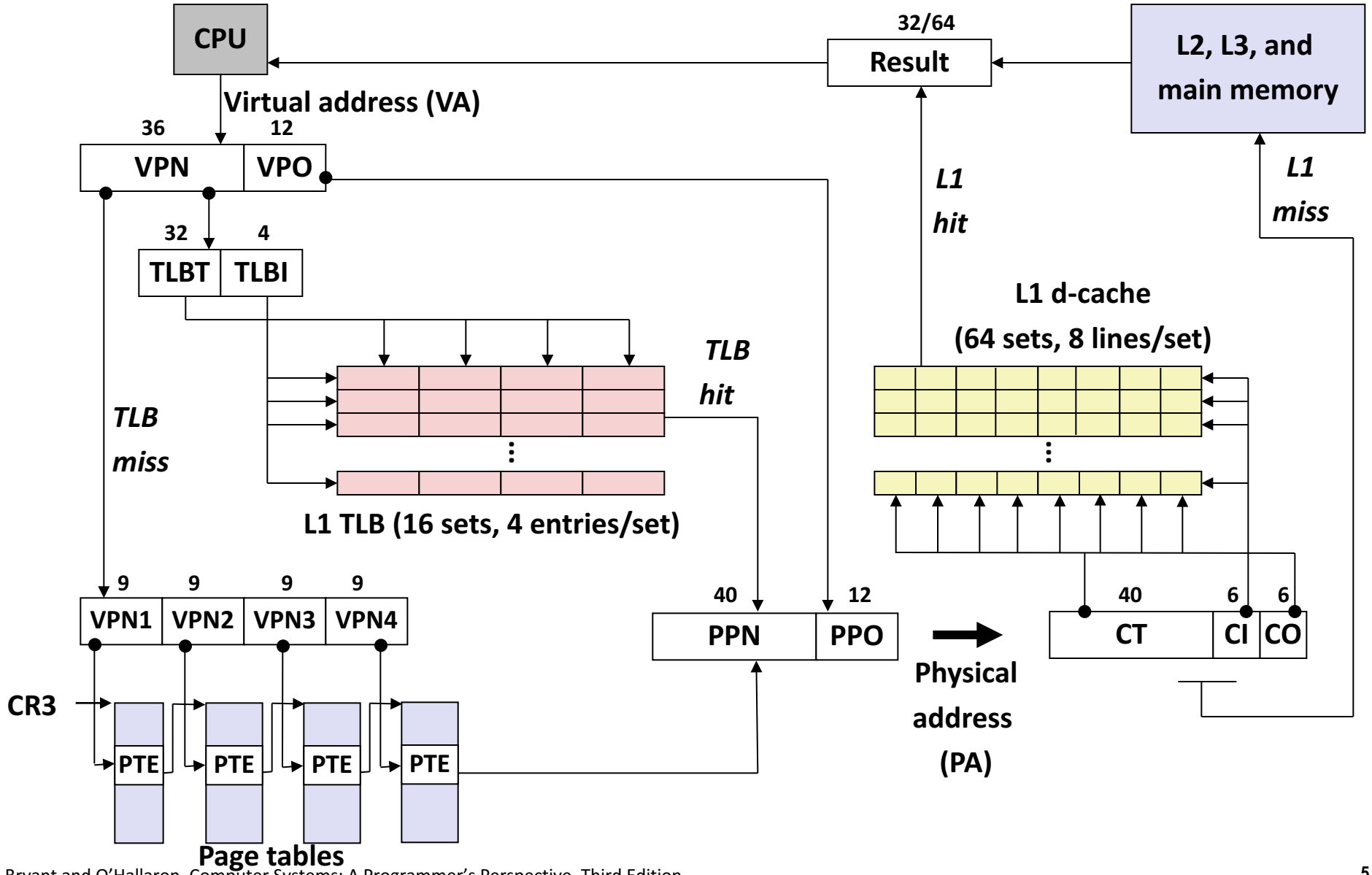
Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

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

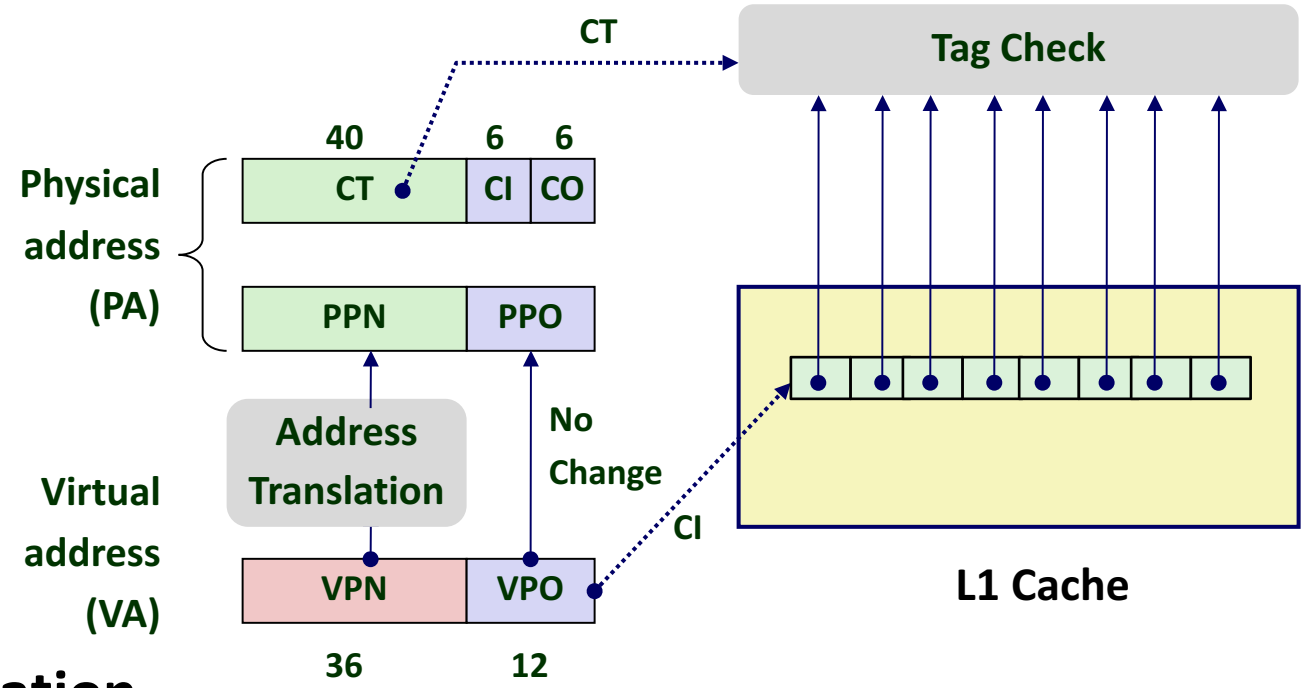
P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The 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

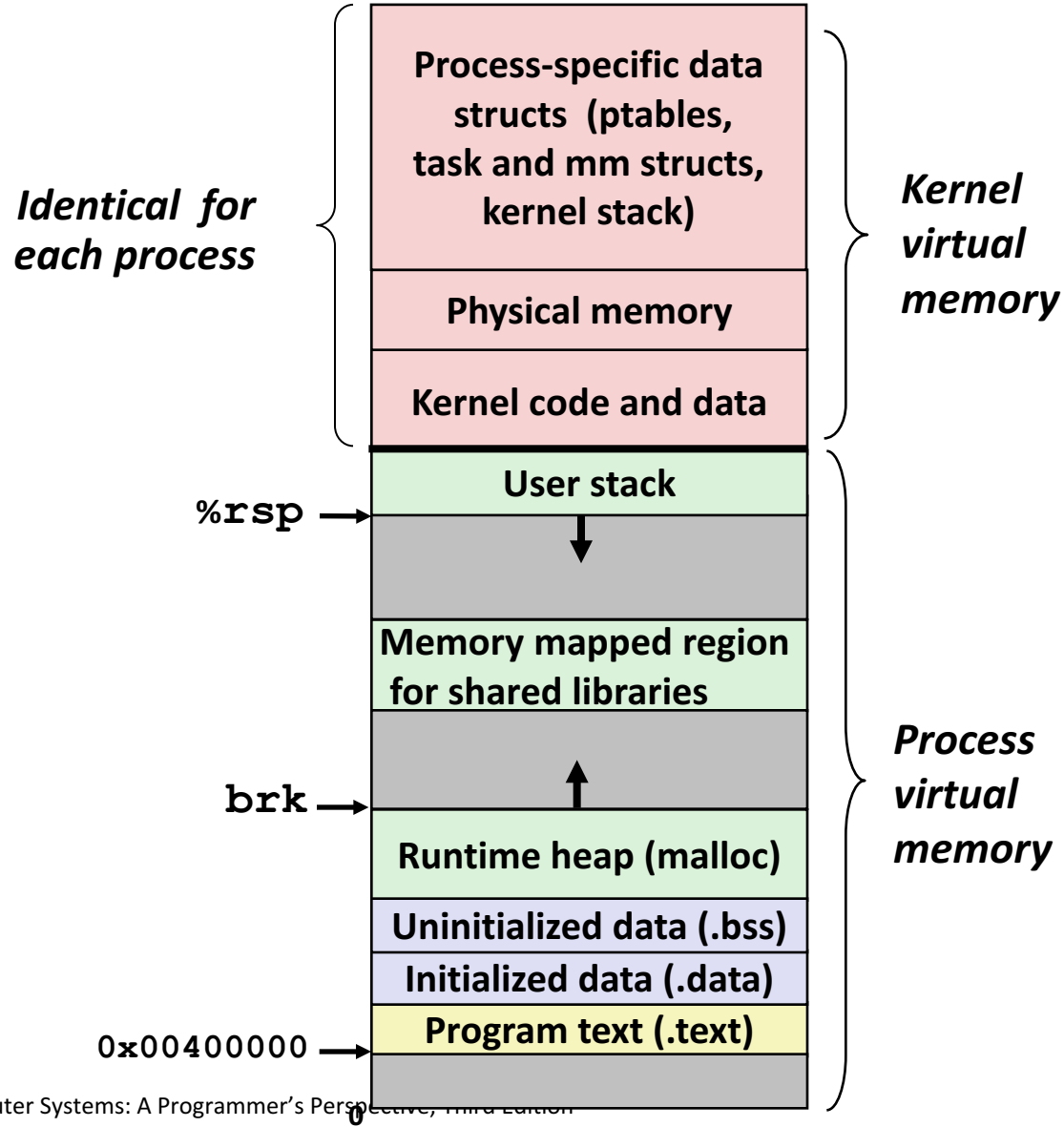


Cute Trick for Speeding Up L1 Access



- **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



Q. Pages Required To Allocate

- 1 Byte

- 4 KB (Page Size) $\equiv 2^{12}$ Bytes

- $512 \cdot 4\,KB \equiv 2^9 \cdot 2^{12}$ Bytes $\equiv 2MB$

- $512 \cdot 2MB \equiv 1GB \equiv 2^9 \cdot 2^9 \cdot 2^{12}$ Bytes