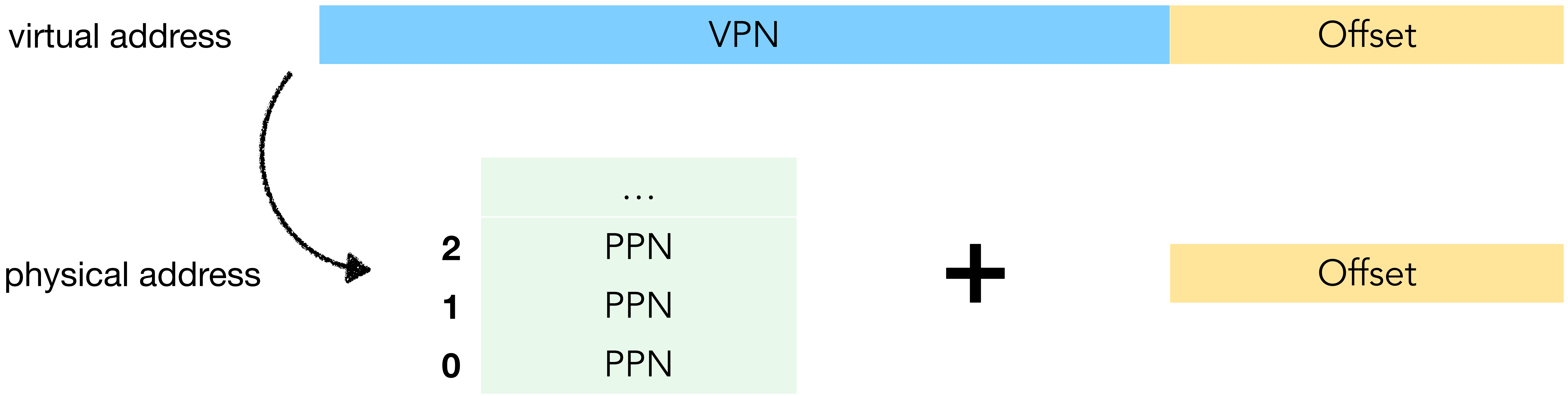
CS202 (003): Operating Systems
Virtual Memory II

Instructor: Jocelyn Chen

Most of the materials covered in this slide come from the lecture notes of Mike Walfish’s CS202

Key data structure: page table

a map from VPN to PPN



virtual address

VPN

Offset

physical address

…

2 PPN

1 PPN

0 PPN

+

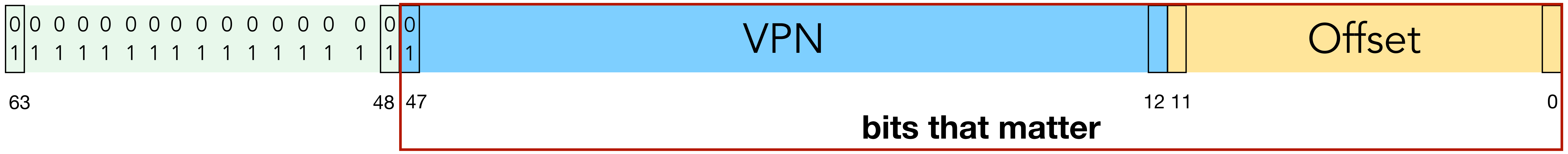
Offset

Each page table entry expresses a mapping about a contiguous group of address

x86-64

x86 architecture is 64-bits

Virtual Address

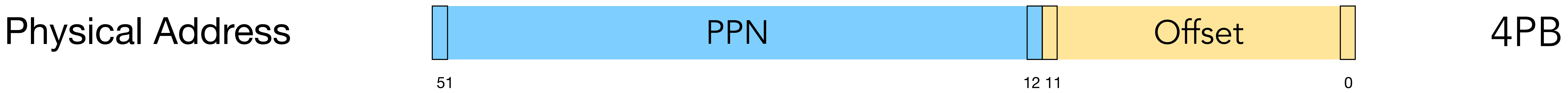


Bit patterns that are valid addresses are called **Canonical Addresses**

48-bit usable bits = 2^{48} possible addresses = 256 TB

x86-64

x86 architecture is 64-bits

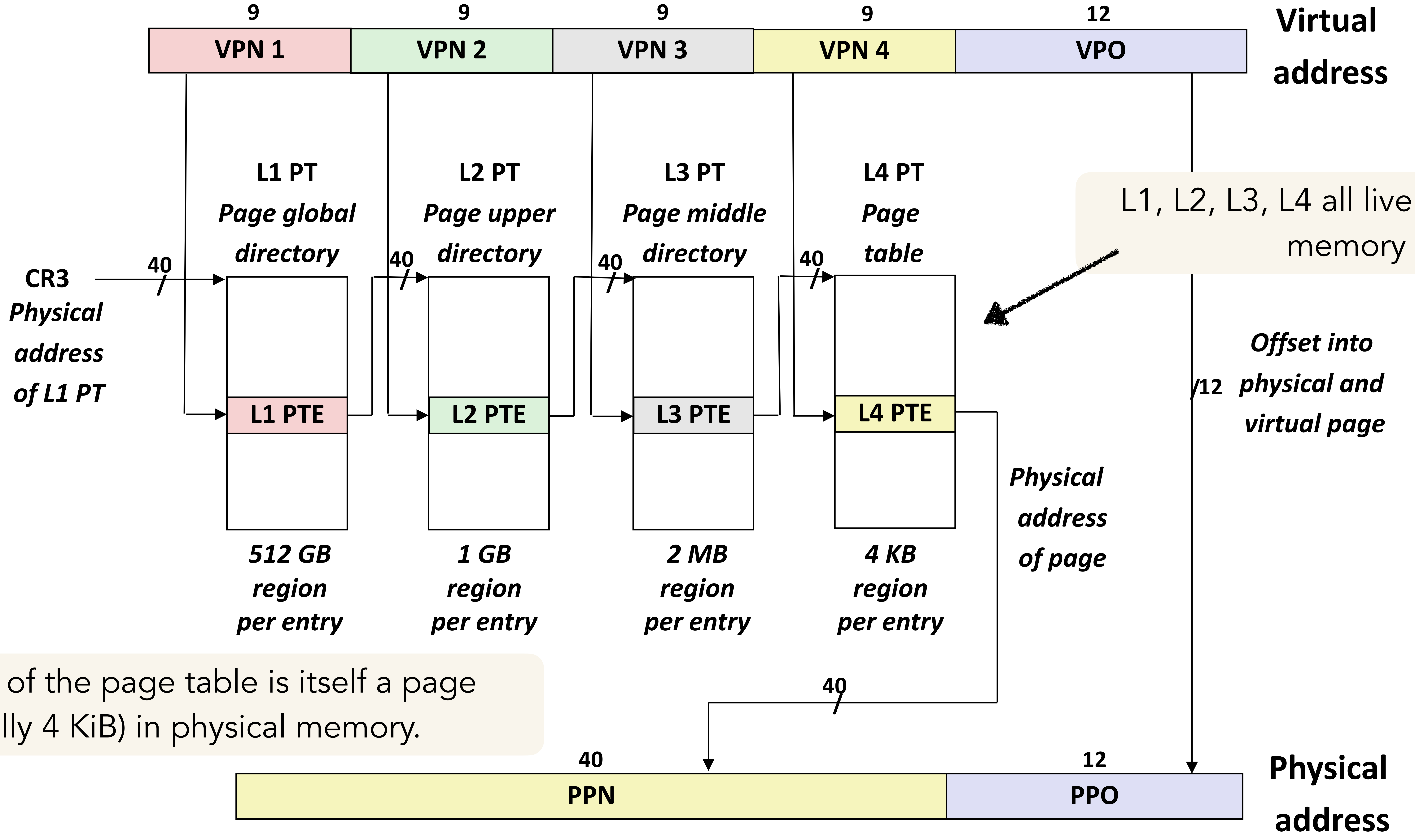


What happen if we only have 16 GB of memory?

(roughly) only 34 bits that matters!
the top 18 bits will (generally be) zero

We are mapping 48-bit number to 52-bit number, at a granularity of ranges of 2^{12}

Core i7 Page Table Translation

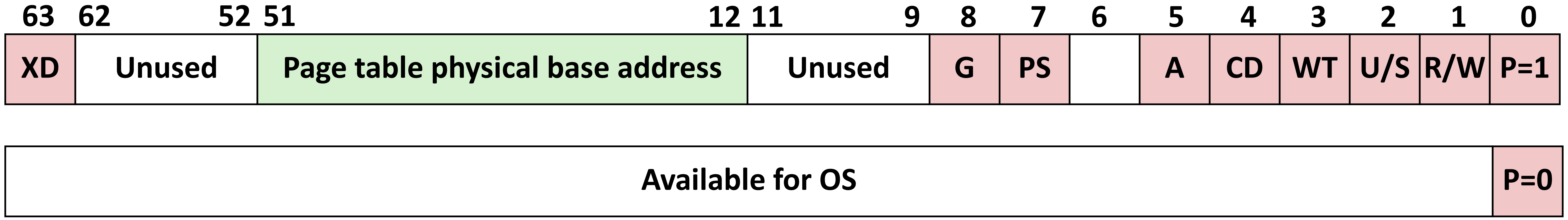


L1, L2, L3, L4 all live in physical memory

Each level of the page table is itself a page (typically 4 KiB) in physical memory.

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Core i7 Level 1-3 Page Table Entries



63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page table physical base address	Unused	G	PS		A	CD	WT	U/S	R/W	P=1

Available for OS	P=0

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.

Core i7 Level 4 Page Table Entries

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address	Unused	G		D	A	CD	WT	U/S	R/W	P=1

Available for OS (for example, if page location on disk)	P=0

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

P: **Virtual** 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)

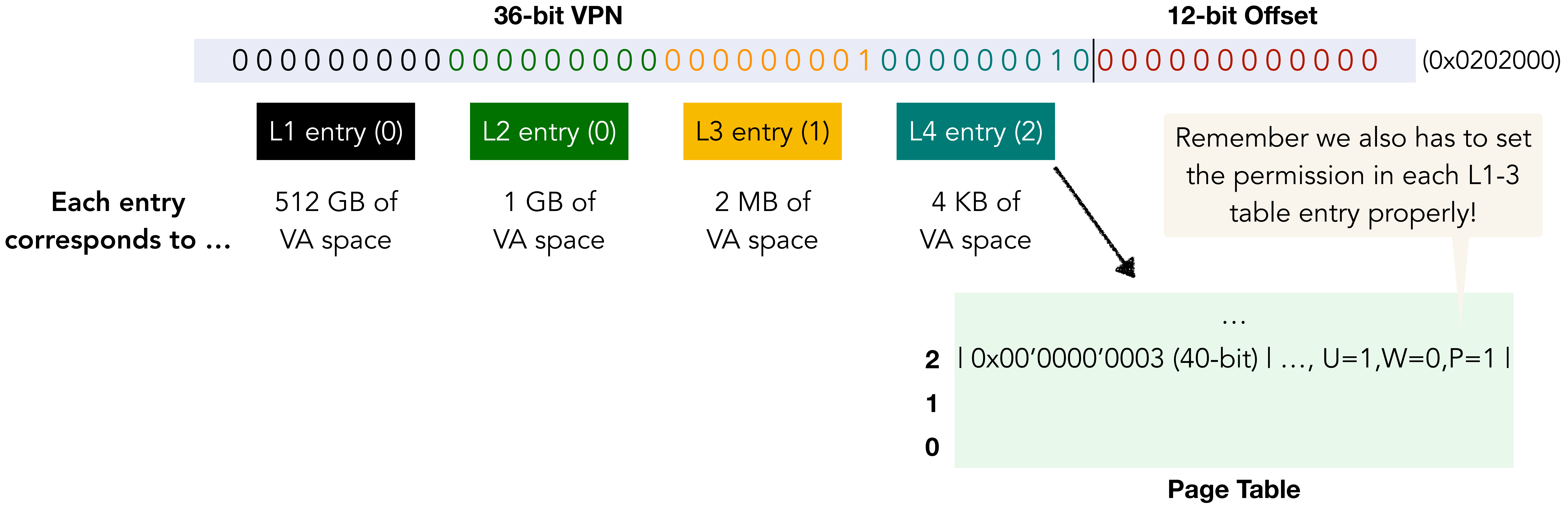
D: Dirty bit (set by MMU on writes, cleared by software)

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.

x86-64 address translation

What happens if we want to map a process's from VA 0x0202000 to PA 0x3000, while making it accessible to user-level but read-only?



36-bit VPN **12-bit Offset**

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 | 0 0 0 0 0 0 0 0 0 0 0 0 (0x0202000)

Each entry corresponds to …	L1 entry (0)	L2 entry (0)	L3 entry (1)	L4 entry (2)
	512 GB of VA space	1 GB of VA space	2 MB of VA space	4 KB of VA space

Remember we also has to set the permission in each L1-3 table entry properly!

…

2 | 0x00'0000'0003 (40-bit) | …, U=1,W=0,P=1 |

1

0

Page Table

x86-64 address translation

What is the minimum number of physical pages required on x86-64 to allocate the following allocations?

1 byte of memory

1 L1, L2, L3 and L4 pages + 1 physical page for the actual memory = 5

1 allocation of size 2^{12} bytes of memory

same as previous question, because 2^{12} = 4 KB = 1 page size

2^{9} allocations of size of 2^{12} bytes of memory each

2^{9} (physical pages for the memory) + 4 (L1, L2, L3, L4)

(2^{9}+1) allocations of size of 2^{12} bytes of memory each

(2^{9}+1) (physical pages for the memory) + 3 (L1, L2, L3) + 2 L4

(2^{18}+1) allocations of size of 2^{12} bytes of memory each

(2^{18}+1) (physical pages for the memory) + 2 (L1, L2)
+ 2 L3
+ (2^{9} + 1) L4

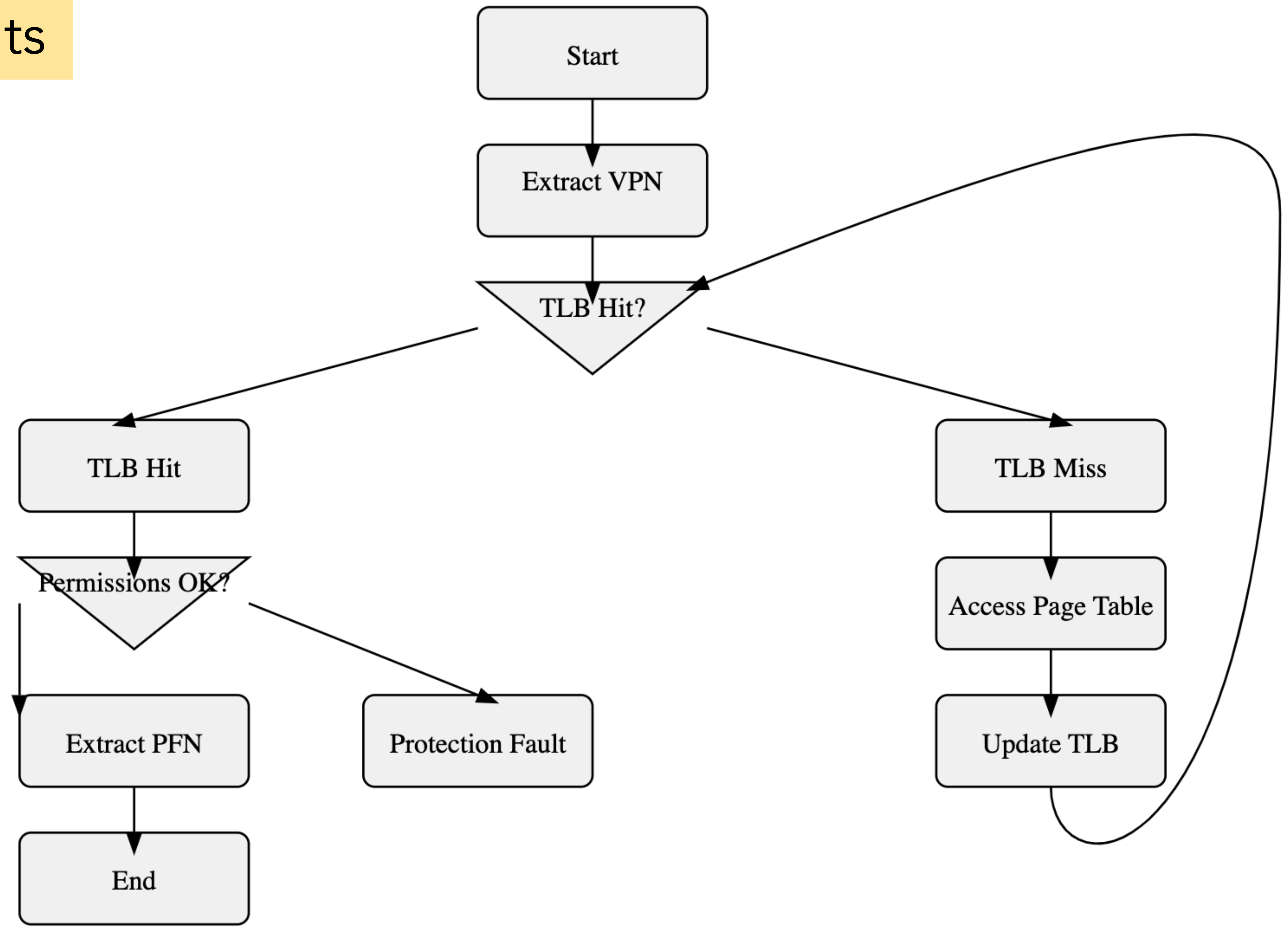
How to speed up address translation?

TLB (translation-lookaside buffer) inside MMU, is a **hardware cache** of popular virtual-to-physical address translation

VPN	PFN	protection bits

Who manages TLB?

Hardware-managed (x86, ARM)
Software-managed (MIPS)



How to speed up address translation?

TLB (translation-lookaside buffer) inside MMU, is a **hardware cache** of popular virtual-to-physical address translation

TLB miss => page fault?

No. It might just means we don’t have the cache.

page fault => TLB miss?

No, the process might request some operations that violates permission. It is a page fault, but not a TLB miss.

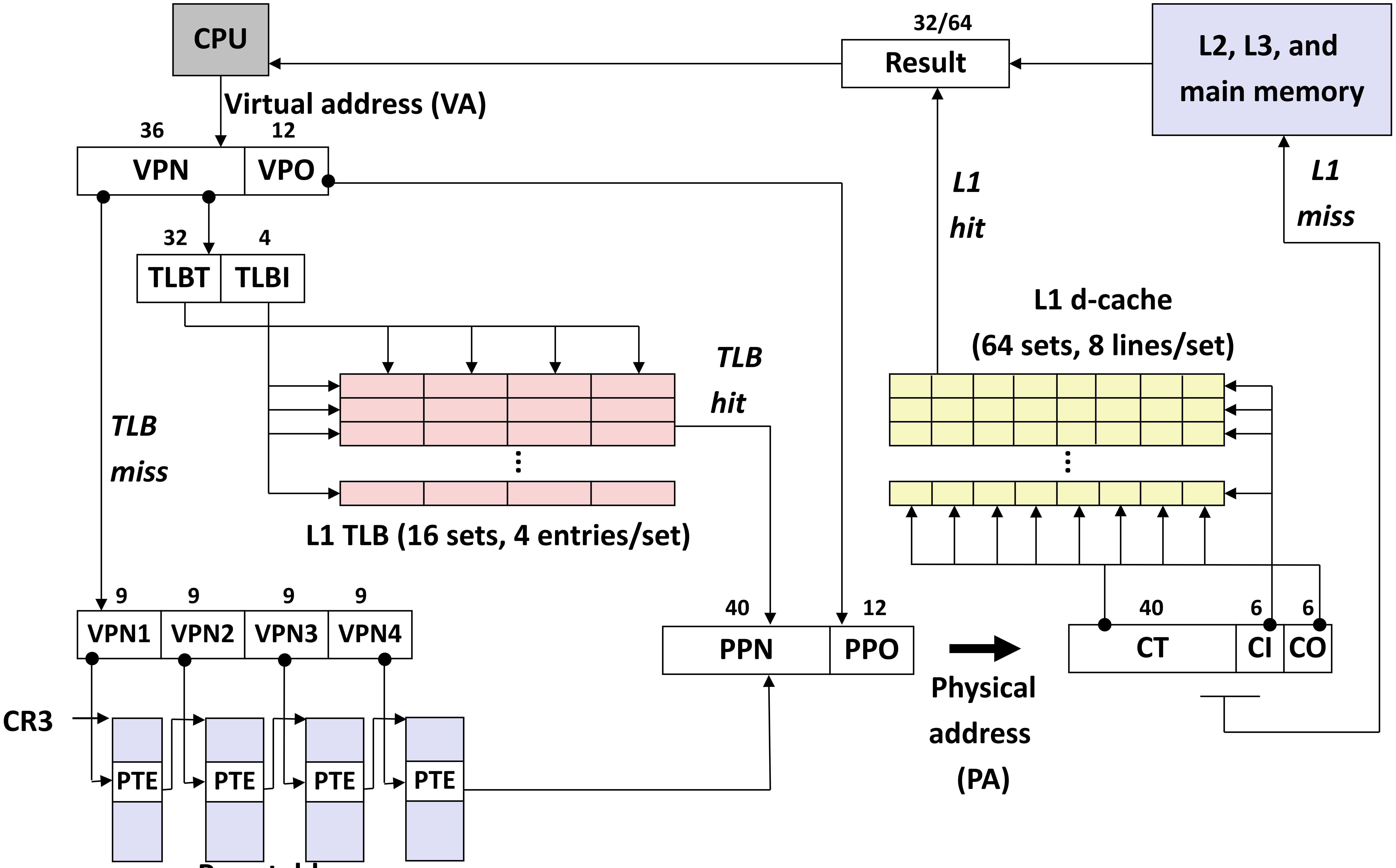
What happens to TLB when %cr3 is loaded?

The entire TLB is flushed

Can we flush individual entries in the TLB otherwise?

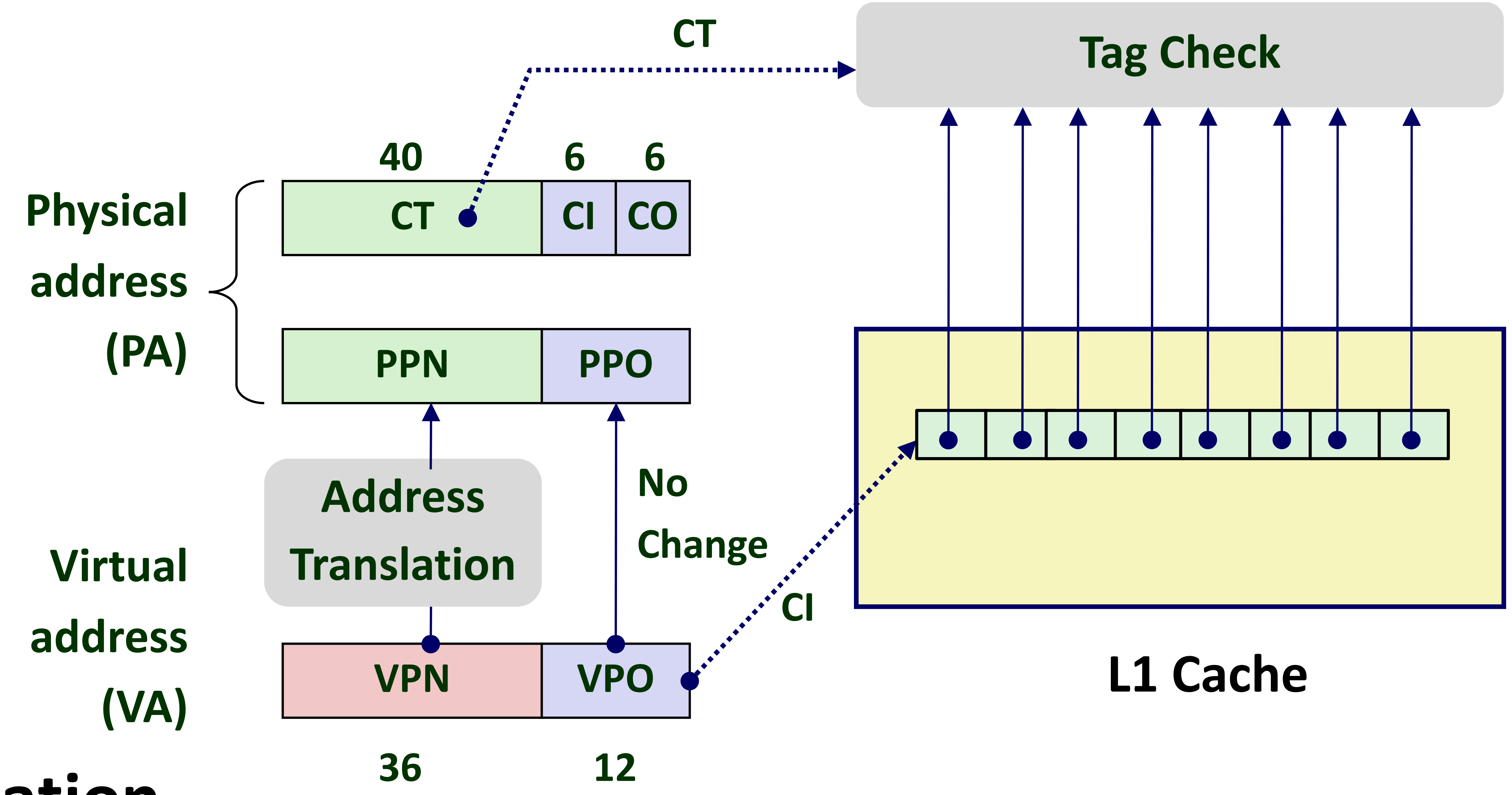
Yes, on x86 architectures, you can flush individual TLB entries using the `INVLPG` instruction

End-to-end Core i7 Address Translation



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

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

HW 6 is Released Today!

HW 5 is Due Tomorrow!