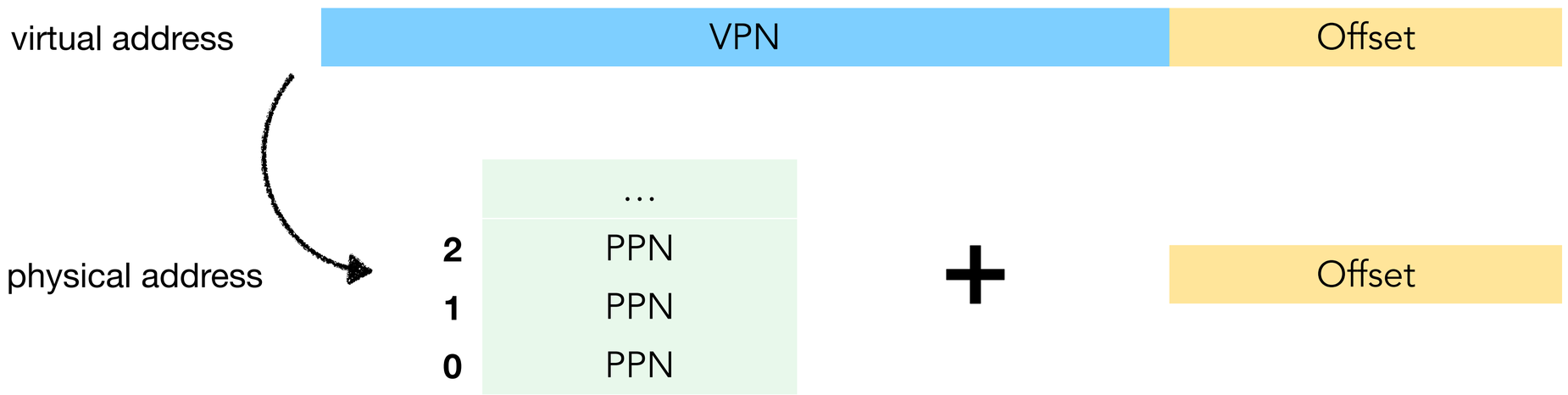
CS202 (003): Operating Systems
Virtual Memory II

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



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

Multi-level page table

Represent a linear page table as a hierarchy of smaller page tables

Each level uses a portion of the virtual address to index into its table

a) The system starts with the root page table.
b) It uses the first part of the address to find an entry in this table.
c) This entry points to a second-level table.
d) The next part of the address is used to index into this second table.
e) This process continues through all levels.
f) The final level provides the actual physical page number.

A virtual address is divided into several parts:

- Multiple segments (often 9 bits each) for indexing each level of tables
- A final segment (often 12 bits) for the offset within the physical page

This tree is space: only fill in parts that are actually in-use!

Alternatives and tradeoffs

Large/small page size

Large page size: waste actual memory

Small page size: lots of page table entries

Many/few level of mapping

Many level of mapping: Less space spent on page structures when address space is space, but more costly for hardware to walk the page table

Few level of mapping: Need to allocate larger pages, which cost more space, but the hardware has fewer levels of mapping

x86-64

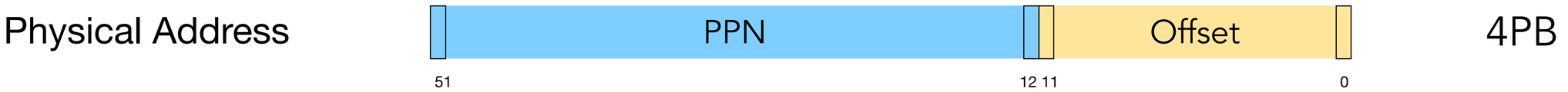
x86 architecture is 64-bits

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

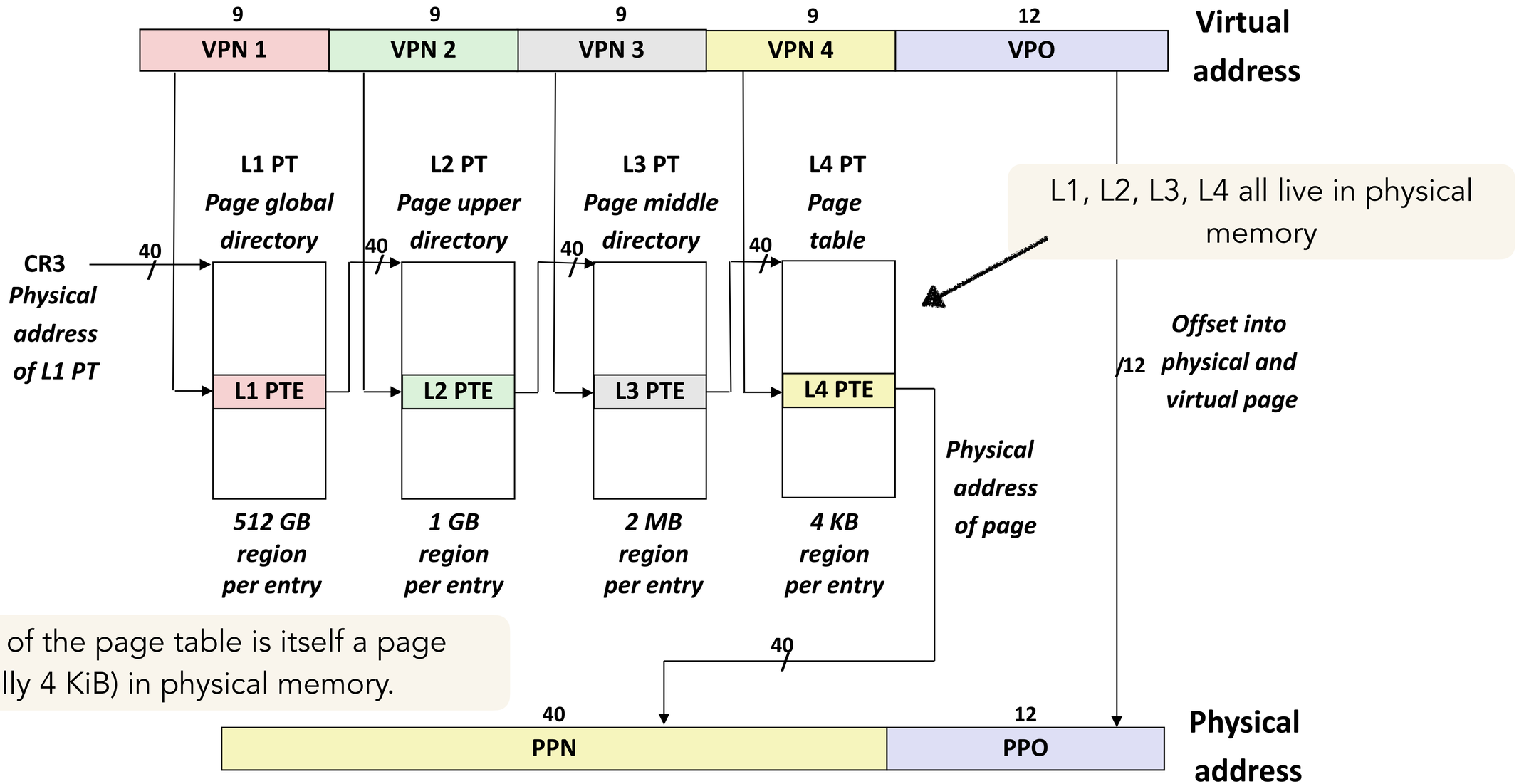


What happen if we only have 16 GB of memory?

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

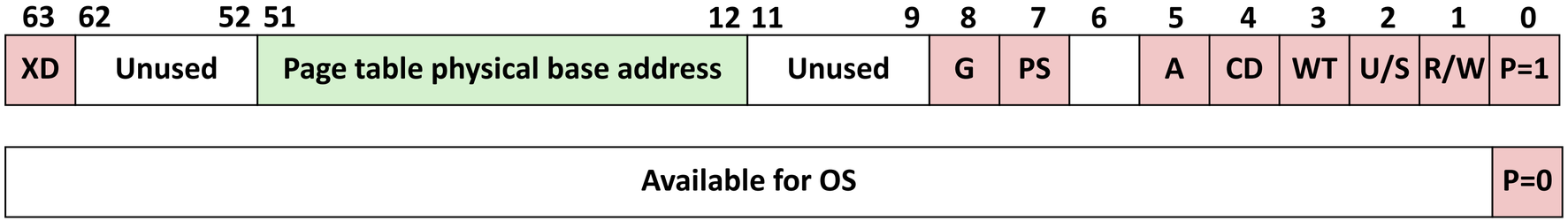
We are mapping 48-bit number to 52-bit number, at a granularity of ranges of 2^{12}

Core i7 Page Table Translation



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

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.

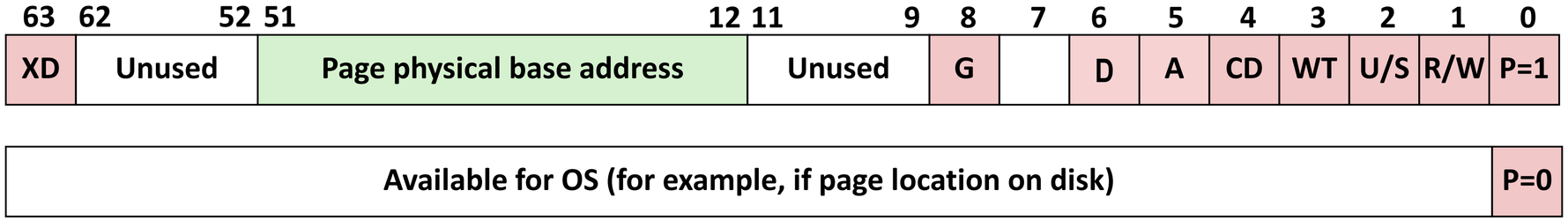
A: Reference bit (set by MMU on reads and writes, cleared by software).

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.

Core i7 Level 4 Page Table Entries



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)

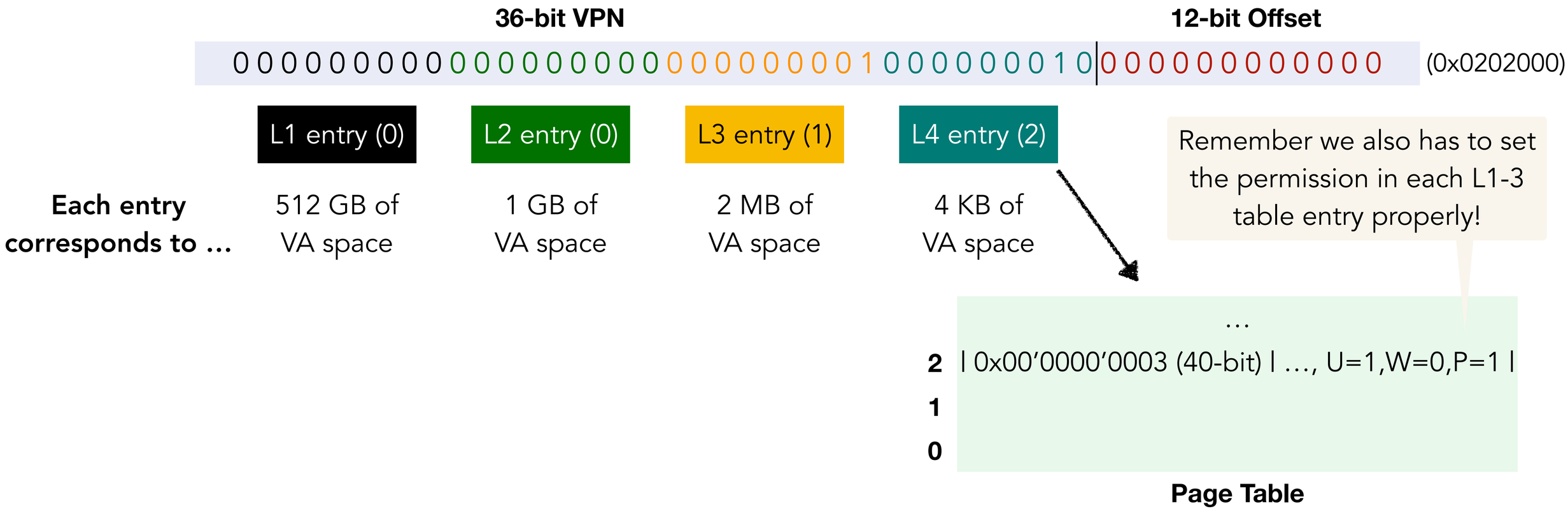
D: Dirty bit (set by MMU on writes, cleared by software)

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?



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

Quiz Time!

Lab 3 is Due Tomorrow!
(at the end of the day)