Advanced Page Tables

Chapter 20

Previously in CS212…

- With the TLB for caching we can increase the speed of our page-based address translations
  - On a TLB hit, we essentially pay no penalty, on a miss, we must perform extra expensive memory lookup operations
- While the TLB helps with speed-based optimizations, linear page tables (as we've been discussing so far) take up quite a bit of precious memory.
- We need a way to shrink the footprint of the page tables

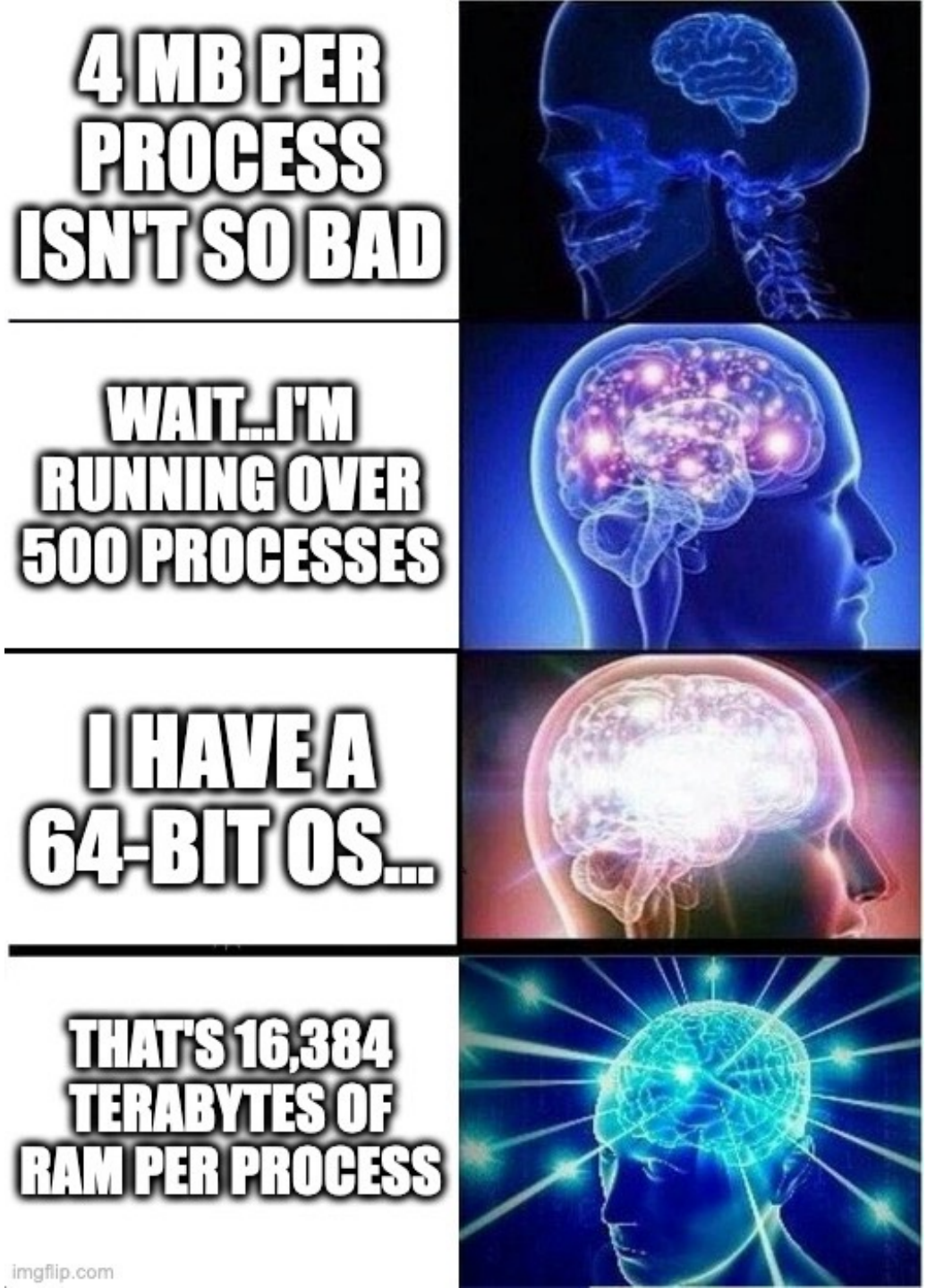
Abbreviation Quick Reference

- Virtual Page Number (**VPN**) – portion of virtual address used to index page tables/directories
- Physical Frame Number (**PFN**) – page that contains the data/code needed for an operation
- Page Table Entry (**PTE**) – a record stored in a page table to reference pages of memory that contain data/code and other metadata (valid, permissions, present, etc.)
- Page Directory Entry (**PDE**) – a record in the page directory to indicate the location of a requested page table

Remember that when using paging everything is stored in pages (that's out unit of storage), the page table/directory is a management structure to keep track of used and unused pages.

Linear Page Table Sizes

- Is this really an issue?
- Assume we have:
  - A 32-bit address space
  - 4KB pages (4096 bytes)
  - 4 Byte page table entries
- This means there are 2^32 possible addresses (~4 million)
- We squeeze those into 4KB pages (2^12 addresses per page)
- 2^32 / 2^12 = 2^20 page entries (~1 million)
- 2^20 * 4 bytes = ~4MB of space used **PER PROCESS**



We need to fix this...

Quick Fix: Make the pages bigger!

- Assume we have:
  - A 32-bit address space
  - 16KB pages (16,384 bytes)
  - 4 Byte page table entries

- 2^32 / 2^14 * 4 Bytes = ~1MB of space used per process

- Better, but what's the drawback?
  - **Big pages == internal fragmentation**

But surely, we aren't using all that space?

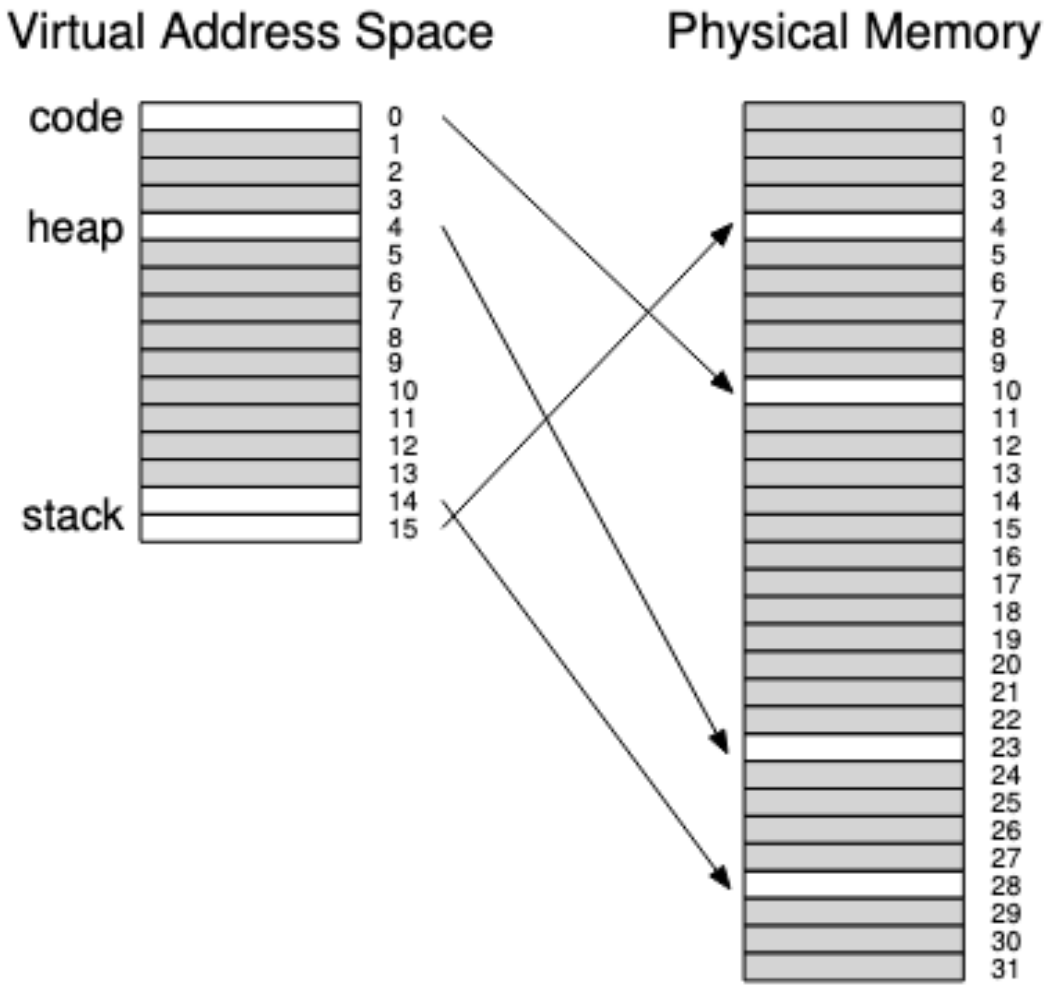


Figure 20.1: **A 16KB Address Space With 1KB Pages**

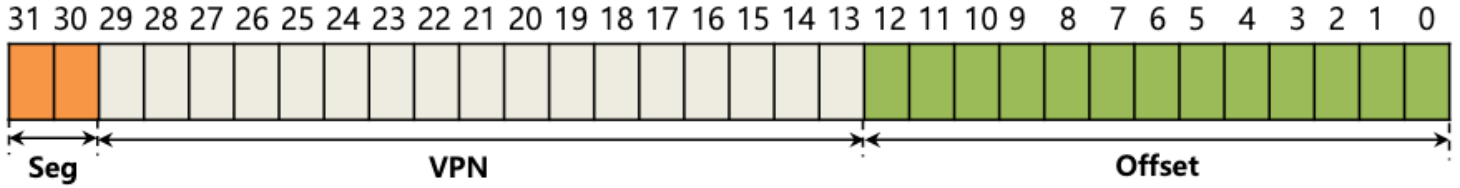
PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
23	1	rw-	1	1
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
28	1	rw-	1	1
4	1	rw-	1	1

Correct, but don't call me Shirley.

What if we just didn’t save invalid entries?

Hybrid Approach: Segments and Paging

- What if each process had a page table **per segment**
- Each segment gets a base and bounds value
  - Base indicate where that segment's page table is located
  - Bounds indicates how many pages are being used
  - So, if we are using pages 0-2 the base points to the physical address of the page table, and the bounds would be 3



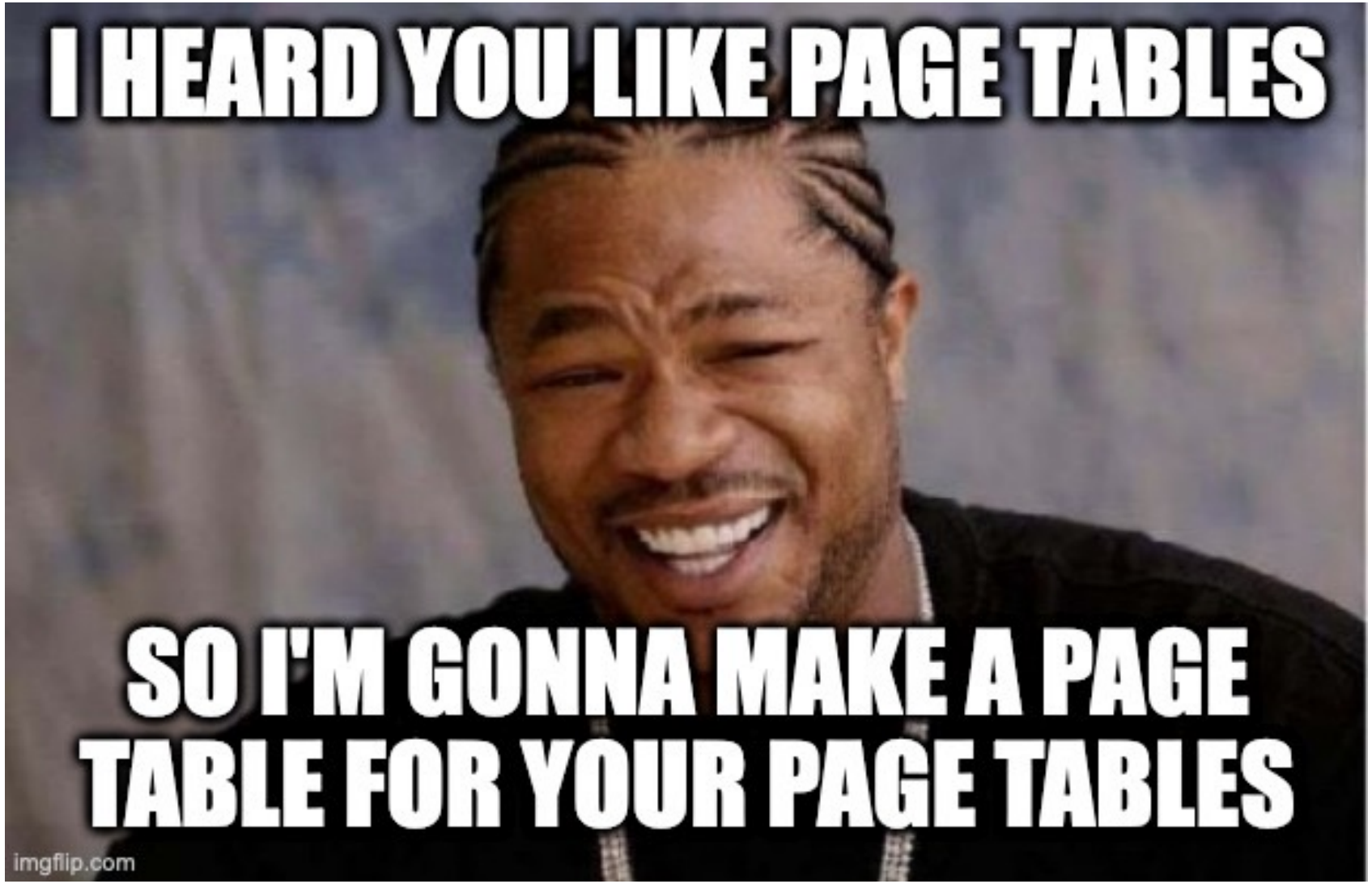
32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

Cool, did we fix it?

- Sort of…
  - we no longer need to record unallocated pages!
- However…
  - page tables are now arbitrary in size resulting in external fragmentation
  - pages that are sparsely used have internal fragmentation

Multi-level Page Tables



PTBR = Page Table Base Register
PDBR = Page Directory Base Register

Multi-level Page Tables

- Take a linear page table and turn it into a tree structure
- We chop up the page table into page-sized units
  - Each unit holds multiple Page Table Entries (PTEs)
- We use a Page Directory to indicate where a Page Table is located, and whether is is valid
  - Valid if **at least one** PTE is valid
  - Invalid if a page contains no valid PTEs (NOT ALLOCATED)

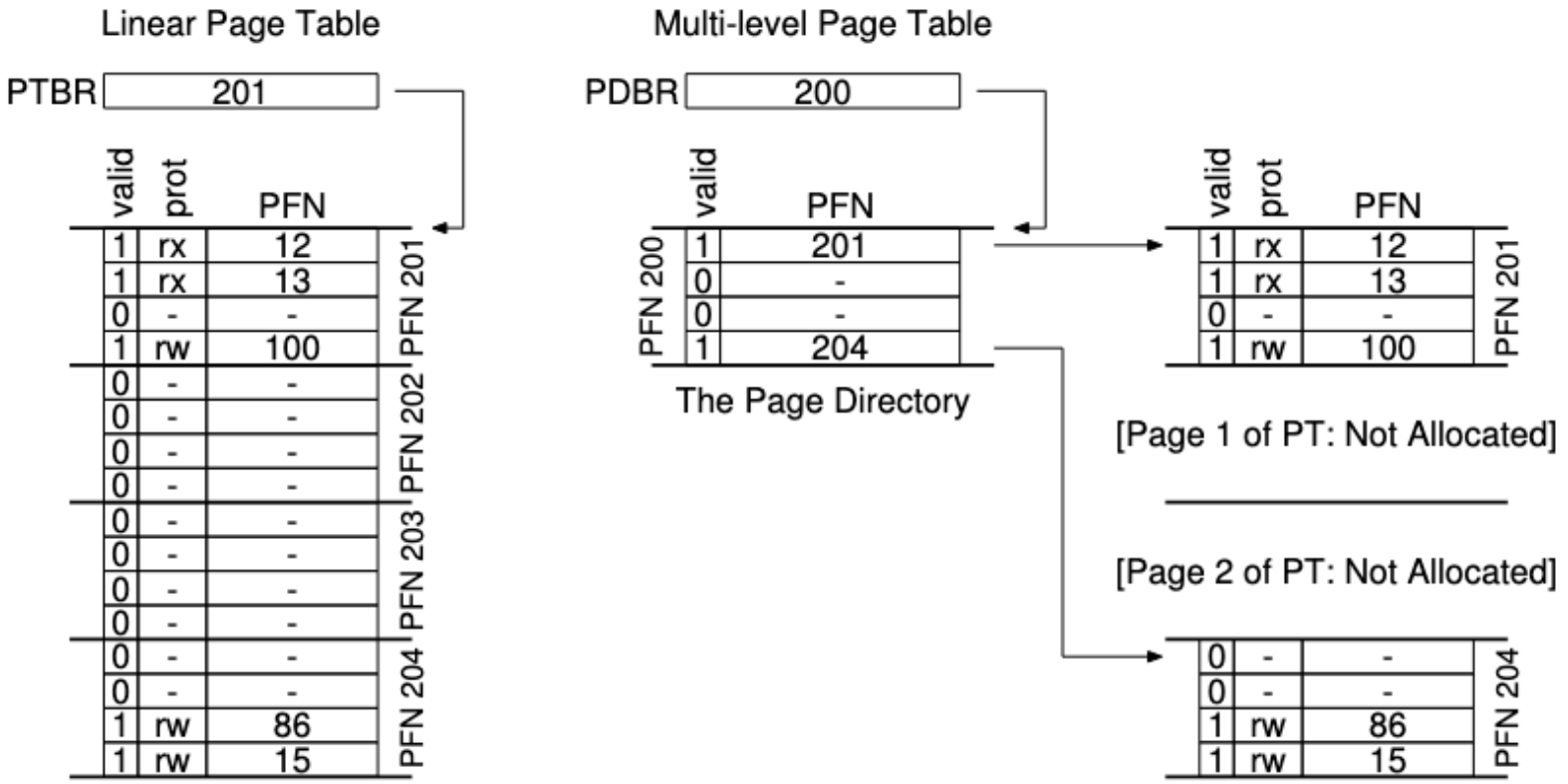


Figure 20.3: **Linear (Left) And Multi-Level (Right) Page Tables**

Advantages

- Uses less memory than a linear page table
  - Pages that would containing only invalid page table entries (PTEs) are simply not allocated and referenced by the page directory
  - Compact and supports sparse address spaces
- If constructed carefully such that each portion of the page table fits within a page, the OS can simply allocate a new page table and update the page directory to reference it
- The directory serves as a level of indirection so we can store the page tables anywhere and they no longer need to be contiguous (as linear page table required)

Disadvantages

- While we are saving space, his introduces add performance overhead
  - Time-space trade-off
- We still use a TLB and cache recently used physical page frame
  - On a TLB hit, performance is the same as the linear page table
  - On a miss, we now need to perform additional memory look-ups to one (or more) page directories to find our page table, so we incur additional memory access penalties
- Complexity (hardware or software) to facilitate the page table lookups

Example: Linear Page Table

Assume a 16KB address space with 64-byte pages

- 16KB is 2^14 so we need 14-bits for the virtual address
- A linear page table requires 256 (2^8) entries 16KB / 64-bytes (2^14 / 2^6)
  - Regardless of how much of that is used by our process
- This means we have an 8-bit VPN and a 6-bit offset
- Assuming each page table entry is 4 bytes, this page table 256 * 4 or 1KB in size

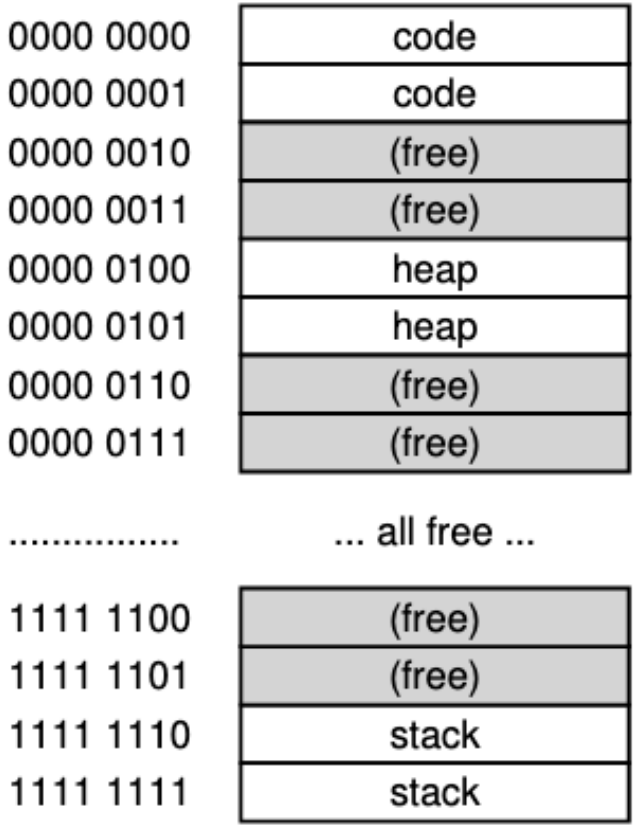
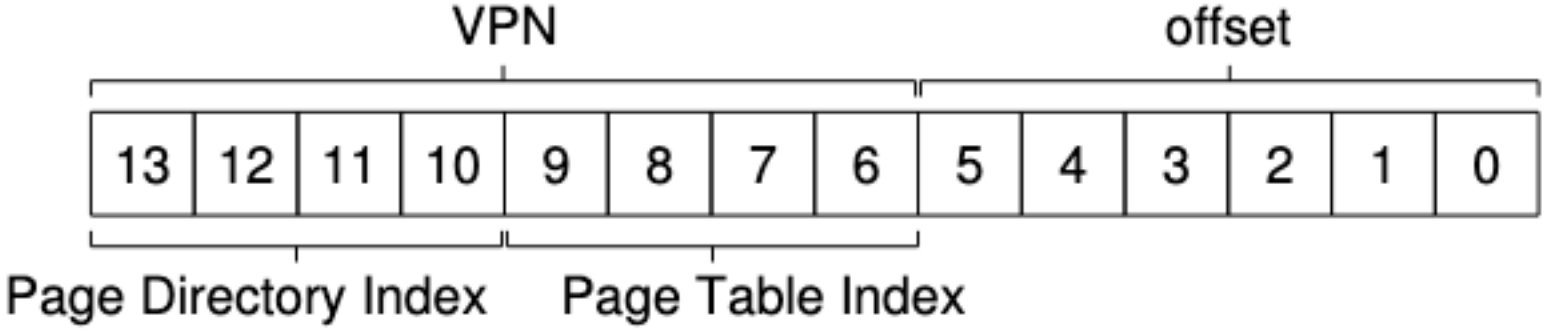


Figure 20.4: **A 16KB Address Space With 64-byte Pages**

Example: Linear to Multi-level

- Our linear page table was 1KB
  - Since we have 64-byte pages, 1KB can fit into 16 pages (1024 / 64)
  - Each page can hold 16 page table entries (PTEs) (64-bytes per page / 4-bytes per entry)
- Our virtual addresses are 14-bits (8-bit VPN and a 6-bit offset)
  - Since we have 256 entries (thus the 8-bit VPN) spread across 16 pages we need to split up the VPN to include the Page Directory
    - To reference 16 pages, we need 4-bits (2^4 = 16)
  - Since each page has 16 PTEs per page, we need 4 more bits from the VPN to be the page-table index



Example: Linear to Multi-level memory layout

Linear (1KB)

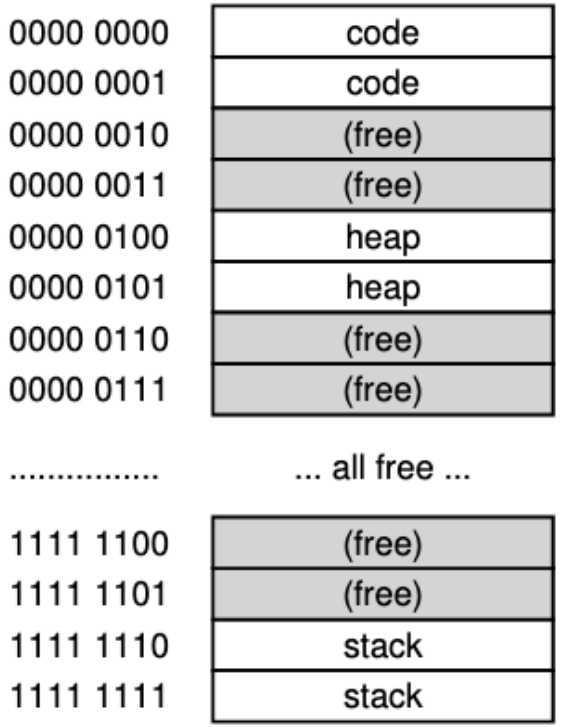


Figure 20.4: **A 16KB Address Space With 64-byte Pages**

Muti-level (192-bytes)

Page Directory		Page of PT (@PFN:100)			Page of PT (@PFN:101)		
PFN	valid?	PFN	valid	prot	PFN	valid	prot
100	1	10	1	r-x	—	0	—
—	0	23	1	r-x	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	80	1	rw-	—	0	—
—	0	59	1	rw-	—	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	—	0	—	55	1	rw-
101	1	—	0	—	45	1	rw-

Figure 20.5: **A Page Directory, And Pieces Of Page Table**

A space savings of 832-bytes!

Example: Address Translation

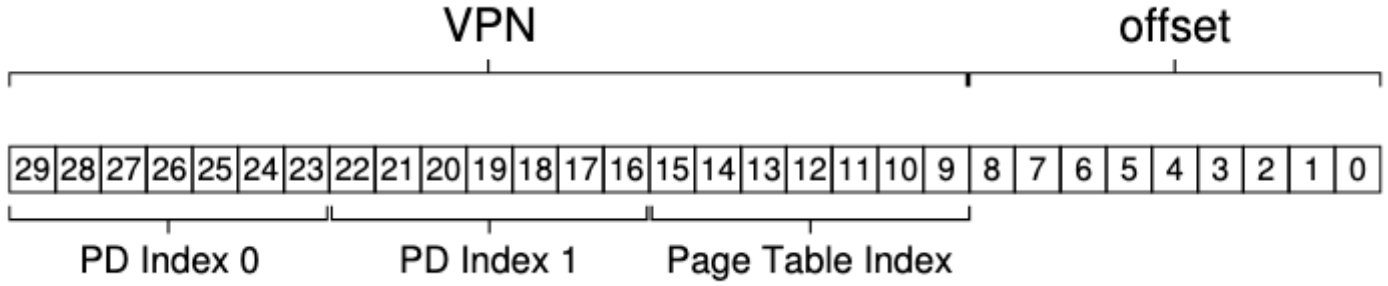
- Let's translate the address 0x3F80
  - 11 1111 1000 0000 (in binary)
- 8-bit VPN
  - Top four bits are the page directory index
    - 1111 or the 15th entry (zero based index)
  - The page is valid and points to PFN 101
  - The next four bits are the page table index
    - 1110 or the 14th entry (zero based index)
  - The page is valid, and we need PFN 55
- Take PFN 55 (0x37 or 0011 0111) and append it to the offset 00 0000
  - Phys Address: 00 1101 1100 0000 or 0x0DC0

Page Directory		Page of PT (@PFN:100)			Page of PT (@PFN:101)		
PFN	valid?	PFN	valid	prot	PFN	valid	prot
100	1	10	1	r-x	—	0	—
—	0	23	1	r-x	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	80	1	rw-	—	0	—
—	0	59	1	rw-	—	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	—	0	—	55	1	rw-
101	1	—	0	—	45	1	rw-

Figure 20.5: **A Page Directory, And Pieces Of Page Table**

When one level of indirection is not enough

- Multi-level page tables are not limited to having one page table directory
  - Why?
  - We need the page table pieces to fit nicely within a page
- Assume a 30-bit virtual address space and 512-byte pages
- Of the 30-bits, we need 21 for the VPN and 9 for the offset
  - 2^9 = 512 (offset) and 30 – 9 = 21 (VPN)
- Assume 4-byte PTEs, that's 128 PTEs per 512-byte page
- Since 128 PTEs requires 7-bits for each page table index that leaves 14 bits for the page directory index
  - If the directory has 2^14 entries and each entry is 512-bytes it requires 128 pages to hold the directory ((2^14 * 4-bytes) / 512-bytes)
- If we split the page directory in two, we could have 2^7 entries which can fit into two 512-bytes page directories



Next Time

- We figured out how to shrink the size of our page tables using multi-level page tables
- While this incurs additional memory look-up penalties, the TLB is still present and can help with that on TBL hits
- However, a lot of things are going on in an OS, and we still might not be able to store everything at once in RAM
- Next time we will look at ways to deal with storing things outside of RAM when we need more space