Operating System (OS) CS232

Memory Management: Smaller Page Tables

Dr. Muhammad Mobeen Movania

Outlines

- Issues with Page Tables
- Bigger Pages
- Hybrid Approach
- Multi-level Page Tables
- Example, Advantages and Disadvantages
- Inverted Page Tables
- Summary

Issues with Page Tables

- Page tables are linear structures that rapidly consume a lot of memory
- Example
  - 32-bit address space (2^{32}=4294967296 bytes), 4KB=2^{12} page size
  - 4 bytes page table entry
  - Total pages= $2^{32}/2^{12}=2^{20}$ =1048576
  - Total page table size = 2^{20}*4=4194304=4MB
  - We have one page table per process, if hundred processes running in system, memory requirements will increase drastically
- Major Concern
  - How can we reduce the page table size?

Simple Solution: Bigger Pages

- Make pages bigger, less page table entries and less storage requirement of page tables
- Example
  - 32-bit address space (2^{32}=4294967296 bytes), 16KB=2^{14} page size
  - 4 bytes page table entry
  - Total pages= $2^{32}/2^{14}=2^{18}$ =262144
  - Total page table size = $2^{18}*4$=1048576=1MB
  - Reduce page table size to a quarter as compared to that with page of 4 KB size

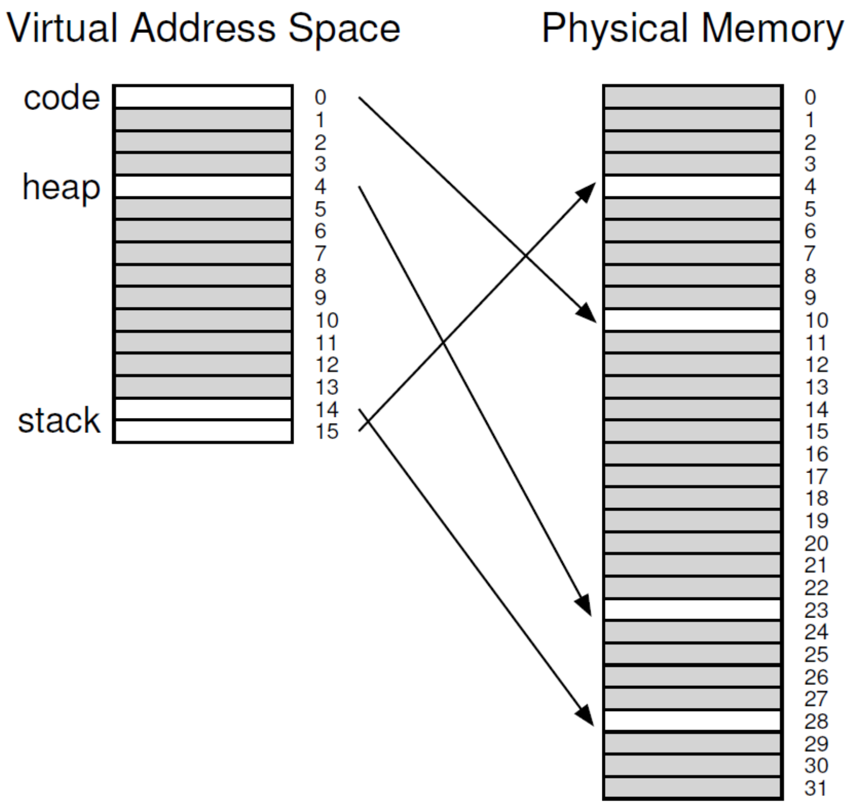
Second Solution: Hybrid Approach

- Key Observation
  - Most of the space inside page table is free
  - Combine segmentation with paging to only store entries with valid data
- Key Idea
  - Have one page table per logical segment
  - Base register: contains physical address of page table of the segment
  - Bounds register: indicates the end of the page table

Example 1

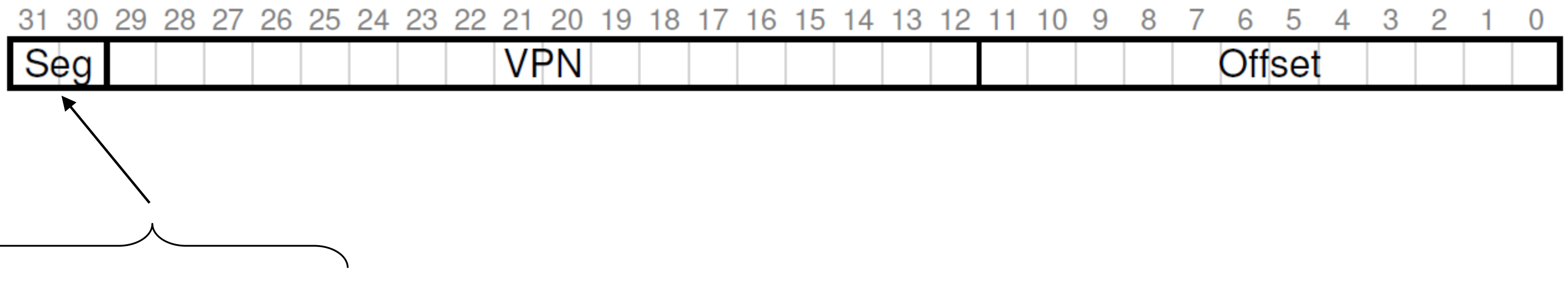
- 16KB address space with 1KB pages and small segments

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



Example 2

- 32-bit virtual address space split into three segments (need 3 base/bounds pair) and 4KB pages



00	Unused
01	Code segment
10	Heap segment
11	Stack segment

```
SN           = (VirtualAddress & SEG_MASK) >> SN_SHIFT
VPN          = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

Hybrid Approach – Pros and Cons

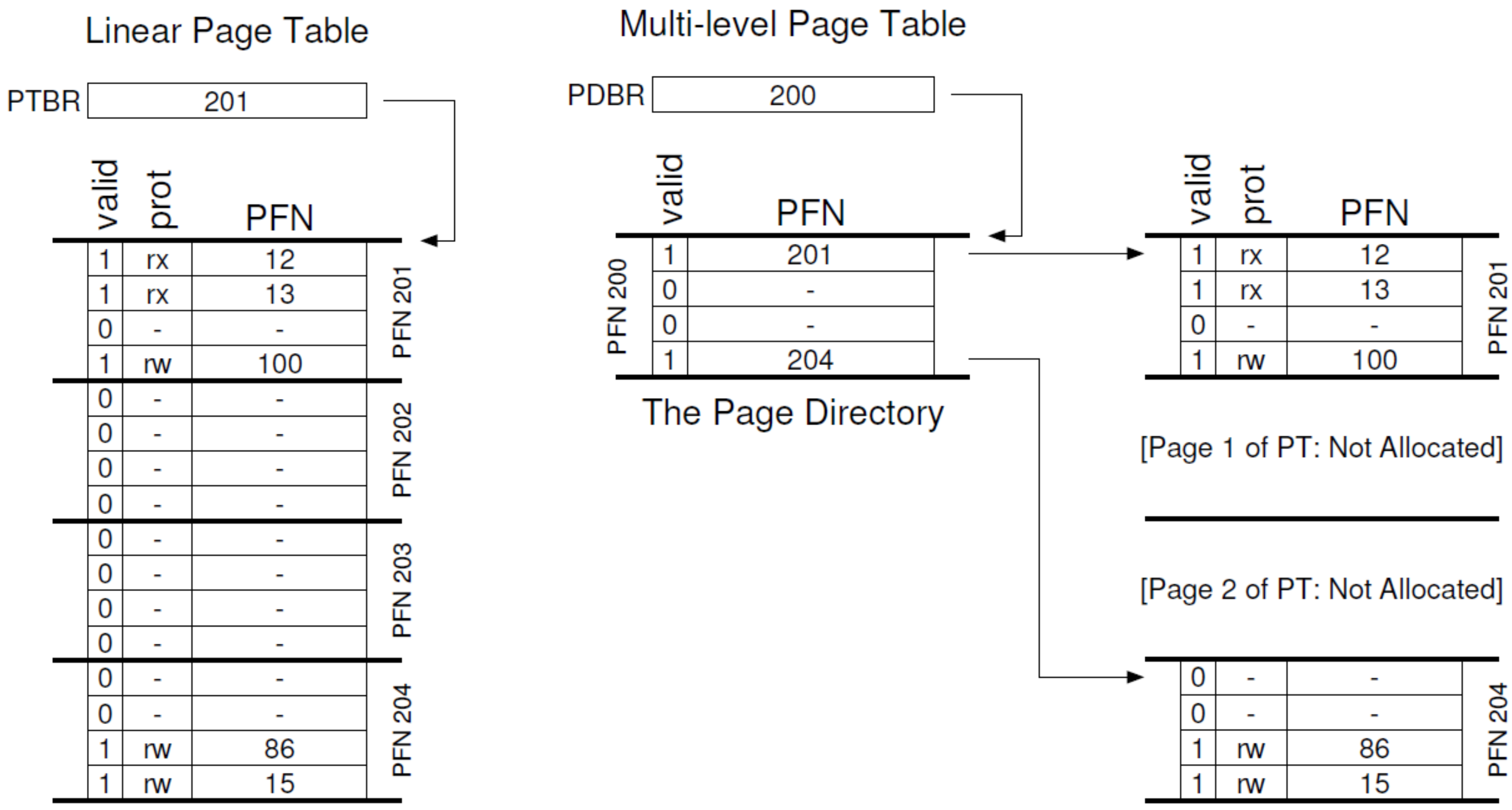
- Pros
  - Reduces page table sizes
- Cons
  - Have multiple base and bound registers
  - Dividing address space into fixed size portions is not flexible
  - Variable sized page tables depending on the segment size

Third Solution – Multi-level page tables

- Chop up the *page table* into page sized fragments
- If an entire page of page-table entries in invalid, don’t allocate space for that page in page table RAM
- A new structure, page directory, tells you:
  - which *pages of the page table* are valid!
  - where in RAM would you find the valid pages of the page table

Example – Multi-level Page Tables

- Comparing linear PT (L) vs multi-level PT (R)



Linear Page Table

PTBR 201

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	
0	-	-	PFN 202
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 203
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Multi-level Page Table

PDBR 200

	valid	PFN
PFN 200	1	201
	0	-
	0	-
	1	204

The Page Directory

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	

[Page 1 of PT: Not Allocated]

[Page 2 of PT: Not Allocated]

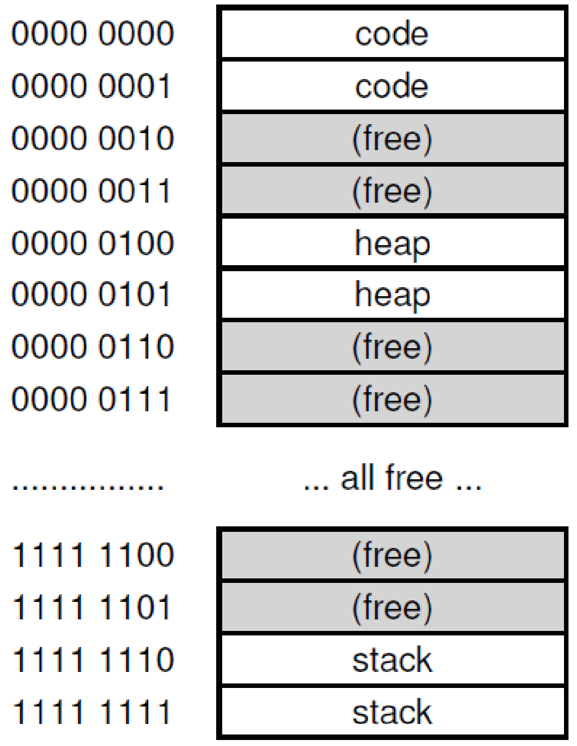
valid	prot	PFN	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Multi-level Page Tables – Pros and Cons

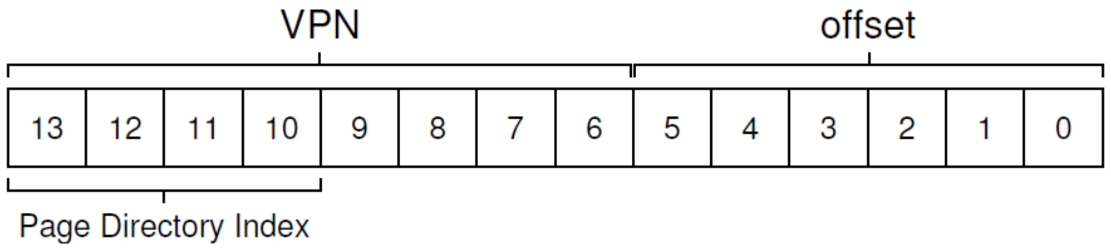
- Pros
  - Page table sizes are reduced
  - Page tables can be distributed in non contiguous pages
    - Previous schemes required contiguous series of pages in memory
- Cons
  - Complexity of implementation
  - On a TLB miss, we've to access the RAM 3 times
    - example of Time-space tradeoff

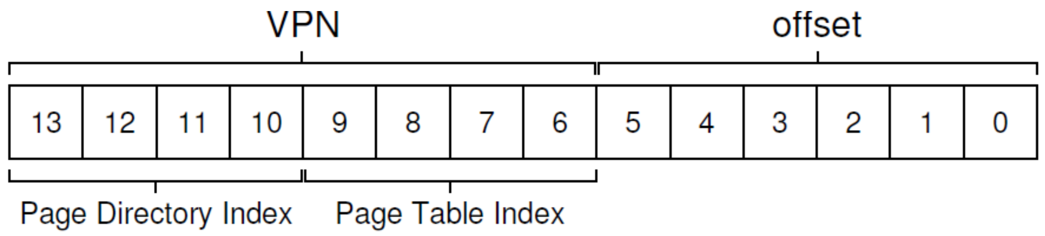
Example: Multi-level Page Table

- 16KB address space, 64 byte pages
- Linear page table size
  - 16 KB address space (2^{14}=16384 bytes)
  - 64 byte page size = 2^6
  - Total pages= $2^{14}/2^6=2^8$ =256
  - Total page table size = 2^8*4 =1024=1KB
  - Page table split into 16, 64 byte pages (16 PTEs)



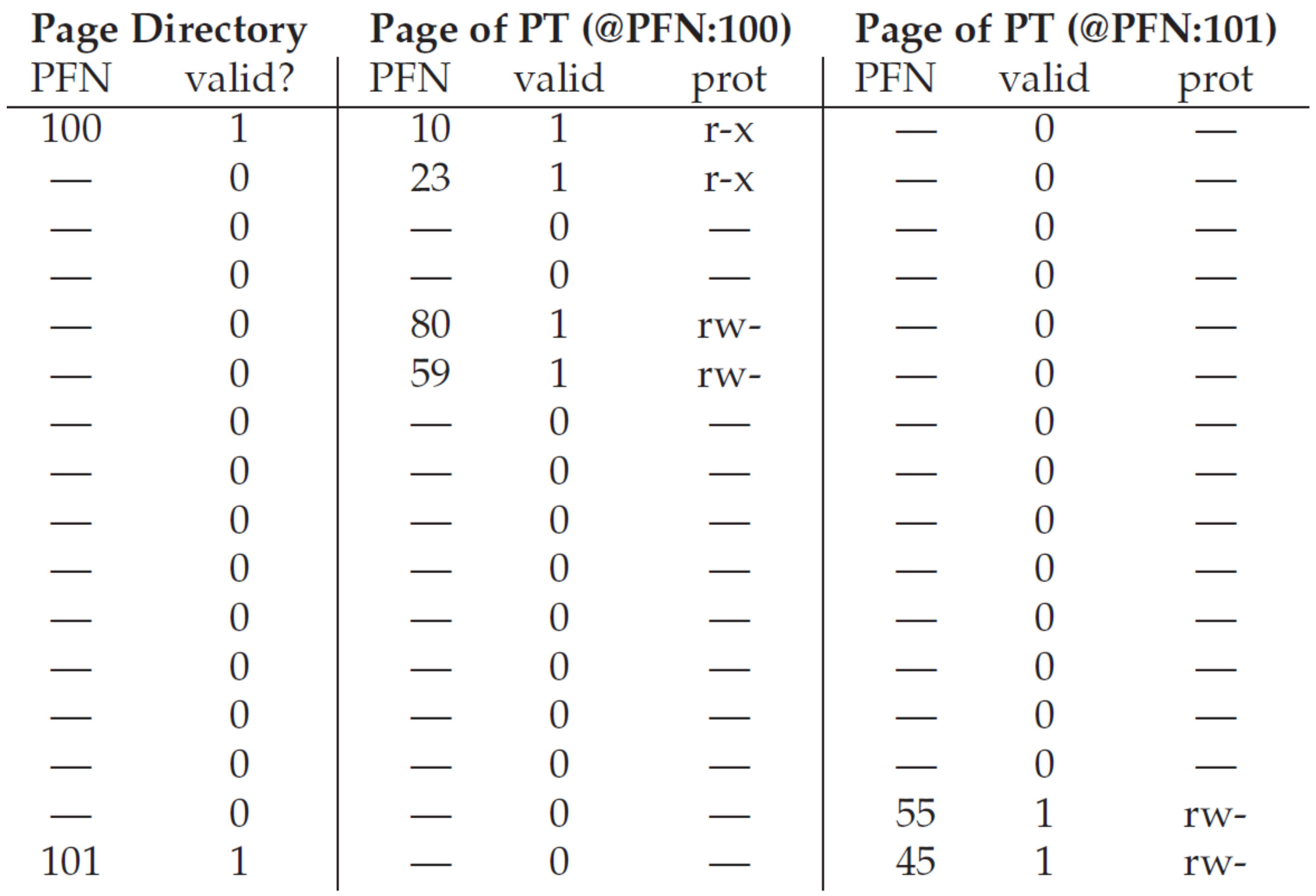
Example: Multi-level Page Table





```
PDEAddr = PDBaseAddr + (PDIndex * sizeof(PDE))
PTEAdd = (PDE.PFN<<SHIFT) + (PTIndex*sizeof(PTE))
```

Example: Multi-level Page Table



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**

Example: Multi-level Page Table

- Crux
  - Instead of allocating 16 pages of a linear page table, we allocate 3 pages
    - One for page directory
    - Two for page table chunks (PFN:100, PFN:101)

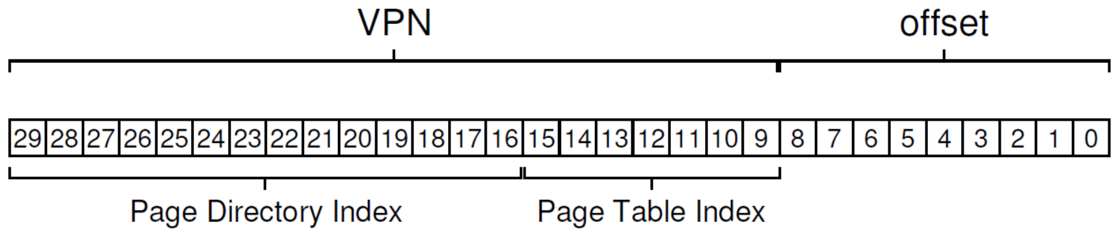
Multi-level Page Table- Algorithm

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)   // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                  // TLB Miss
    // first, get page directory entry
    PDIndex = (VPN & PD_MASK) >> PD_SHIFT
    PDEAddr = PDBR + (PDIndex * sizeof(PDE))
    PDE     = AccessMemory(PDEAddr)
    if (PDE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else
        // PDE is valid: now fetch PTE from page table
        PTIndex = (VPN & PT_MASK) >> PT_SHIFT
        PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
        PTE     = AccessMemory(PTEAddr)
        if (PTE.Valid == False)
            RaiseException(SEGMENTATION_FAULT)
        else if (CanAccess(PTE.ProtectBits) == False)
            RaiseException(PROTECTION_FAULT)
        else
            TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
            RetryInstruction()
```

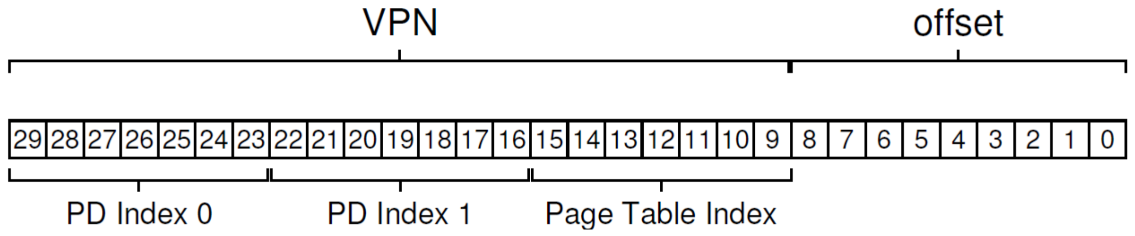
Figure 20.6: **Multi-level Page Table Control Flow**

Multi-level page tables – more than 2 levels

- 30-bit address space, 512 byte pages



- Page directory doesn't fit in one page



Inverted Page Tables

- One page table for the whole system
- Indices are PFNs instead of VPNs
- PTE contains: PID, VPN
- Have to do a linear search
  - Hash tables may help reduce search time

Example: Inverted Page Table

View Shockwave Animation

Summary

- We saw how non-linear page tables are built
- Small tables
  - used in memory constrained systems
- Large tables
  - used with a reasonable amount of memory and with workloads that actively use a large number of pages help speeds up TLB misses
- Leftover issues
  - What if size of total number of pages in a system exceeds RAM?
  - What if there are so many page tables that the RAM space allocated to kernel falls short?