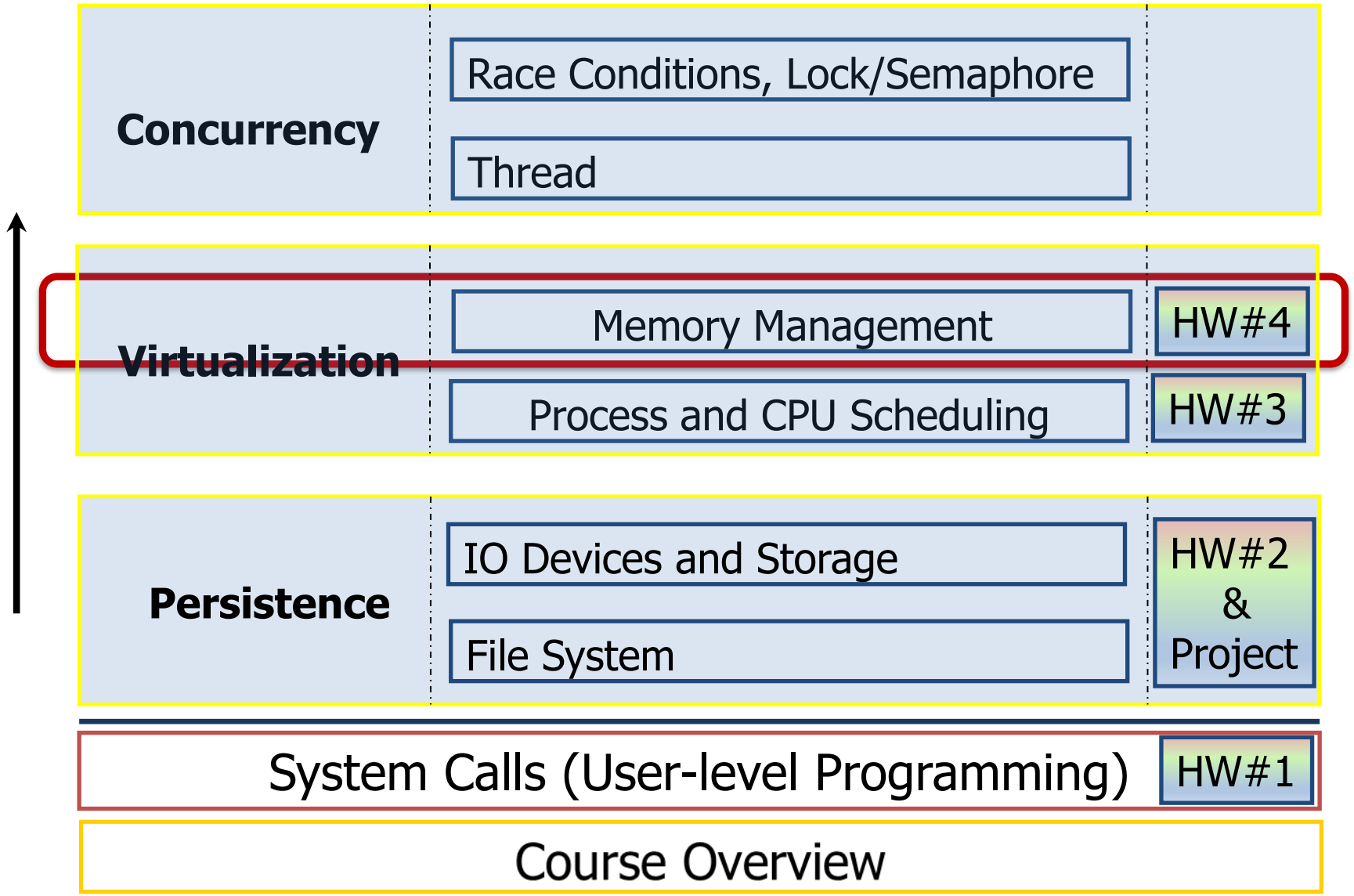
Lecture 11: Virtualizing Memory – TLB and Advanced Paging

The Course Organization (Bottom-up)

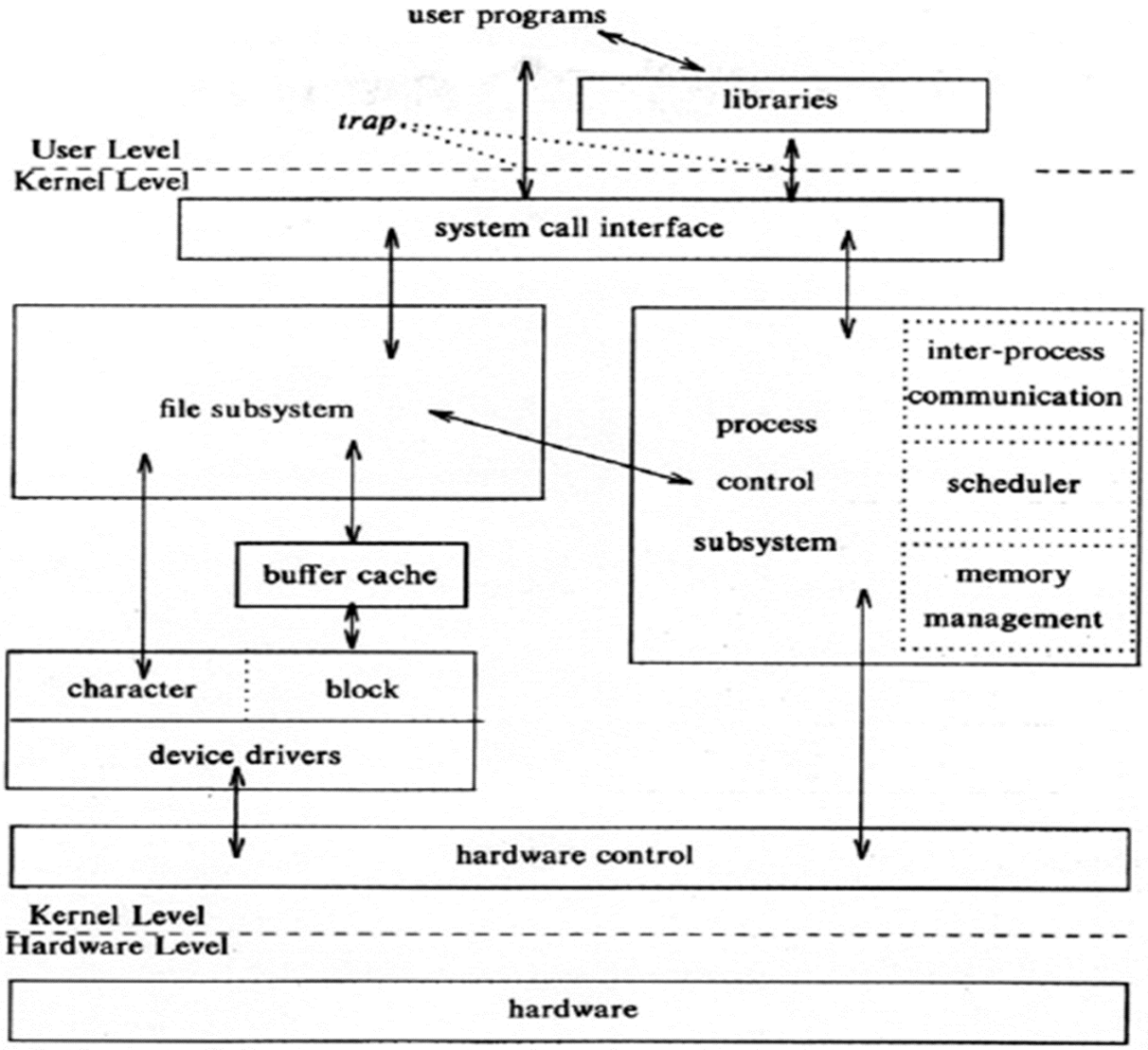


OS – Resource management via virtualization

OS provides services via **System Call** (typically a few hundred) to run **process**, access memory/devices/files, etc.

The OS **manages resources** such as *CPU*, *memory* and *disk* via **virtualization**.

- many programs to run (processes) → Sharing the CPU
- many processes to *concurrently* access their own instructions and data → Sharing memory
- many processes to access devices → Sharing disks

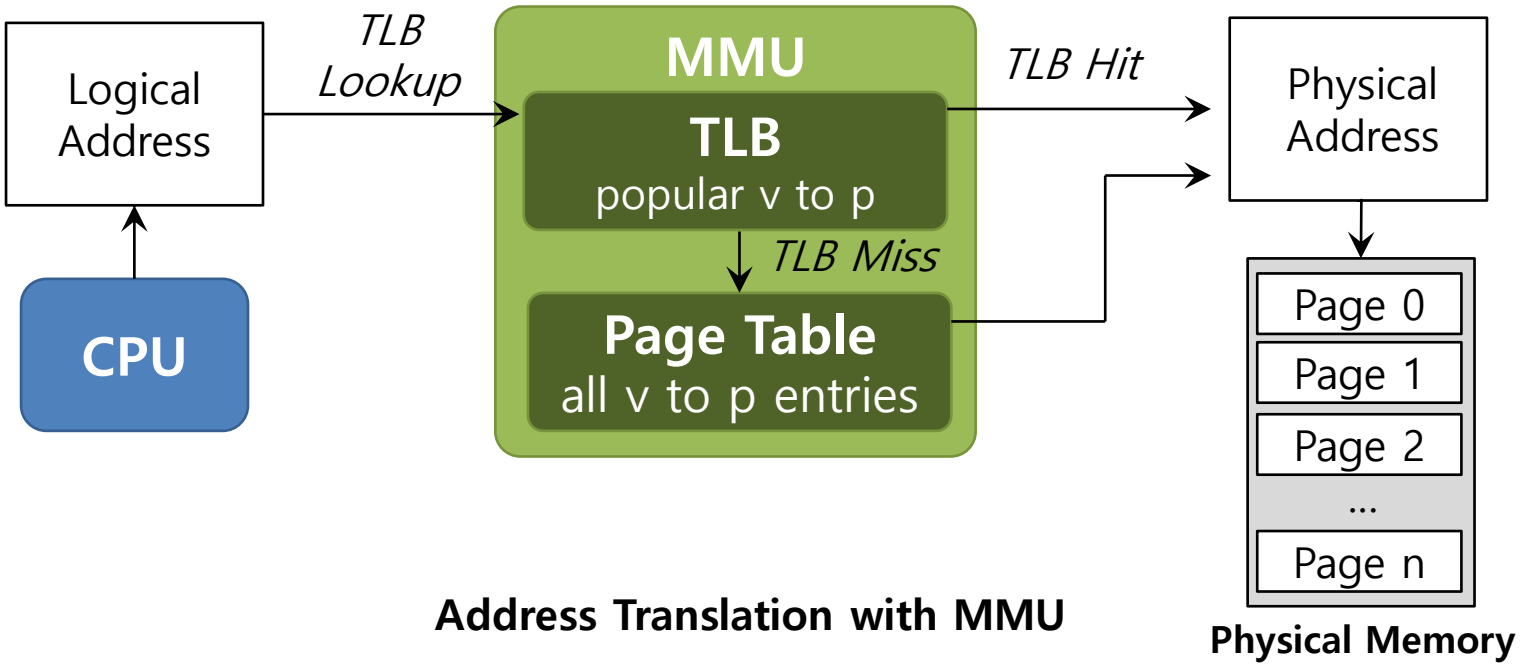


The Design Of The Unix Operating System (Maurice Bach, 1986)

Part I: TLB (Translation Lookaside Buffer)

TLB

- Part of the chip's memory-management unit(MMU).
- A hardware cache of **popular** virtual-to-physical address translation.



Address Translation with MMU

TLB Basic Algorithm

```
1: VPN = (VirtualAddress & VPN_MASK ) >> SHIFT
2: (Success , TlbEntry) = TLB_Lookup(VPN)
3: if(Success == Ture){ // TLB Hit
4:     if(CanAccess(TlbEntry.ProtectBit) == True ){
5:         offset = VirtualAddress & OFFSET_MASK
6:         PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:         AccessMemory( PhysAddr )
8:     }else RaiseException(PROTECTION_ERROR)
```

- (1 lines) extract the virtual page number(VPN).
- (2 lines) check if the TLB holds the transalation for this VPN.
- (5-8 lines) extract the page frame number from the relevant TLB entry, and form the desired physical address and access memory.

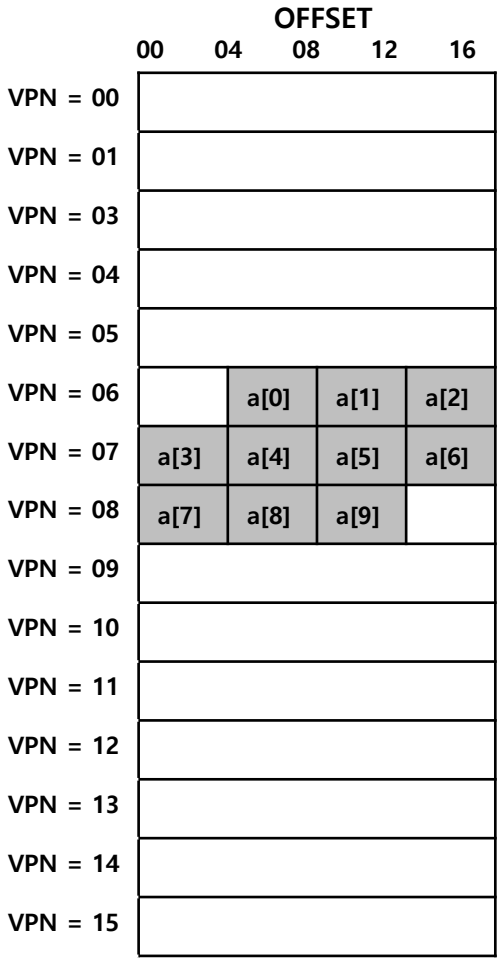
TLB Basic Algorithm (Cont.)

```
11:}else{ //TLB Miss
12:         PTEAddr = PTBR + (VPN * sizeof(PTE))
13:         PTE = AccessMemory(PTEAddr)
14:         (…)
15:         }else{
16:         TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
17:         RetryInstruction()
18:         }
19:}
```

- (11-12 lines) The hardware accesses the page table to find the translation.
- (16 lines) updates the TLB with the translation.

Example: Accessing An Array

- How a TLB can improve its performance.



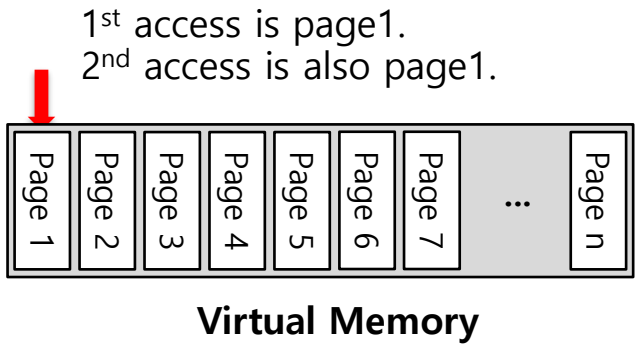

```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```

The TLB improves performance due to spatial locality

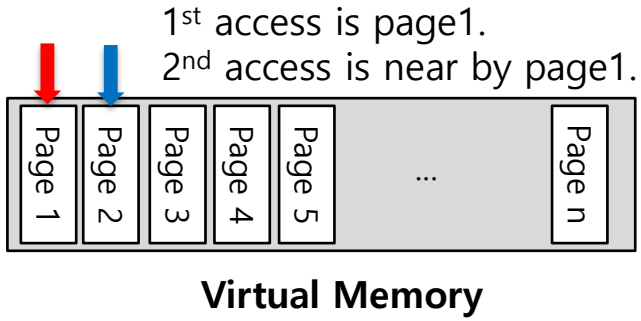
3 misses and 7 hits.
Thus TLB hit rate is 70%.

Locality

- Temporal Locality
  - An instruction or data item that has been recently accessed will likely be re-accessed soon in the future.



- Spatial Locality
  - If a program accesses memory at address `x`, it will likely soon access memory near `x`.

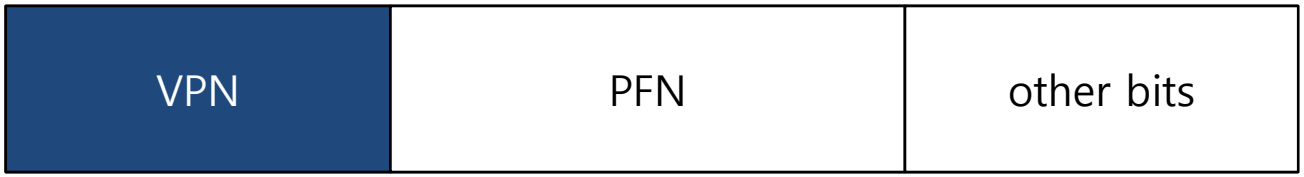


Who Handles The TLB Miss?

- Hardware handle the TLB miss entirely on CISC.
  - The hardware has to know exactly where the page tables are located in memory.
  - The hardware would “walk” the page table, find the correct page-table entry and extract the desired translation, update and retry instruction.
  - **hardware-managed TLB.**
- RISC have what is known as a **software-managed TLB.**
  - On a TLB miss, the hardware raises exception(trap handler).
    - **Trap handler is code** within the OS that is written with the express purpose of handling TLB miss.

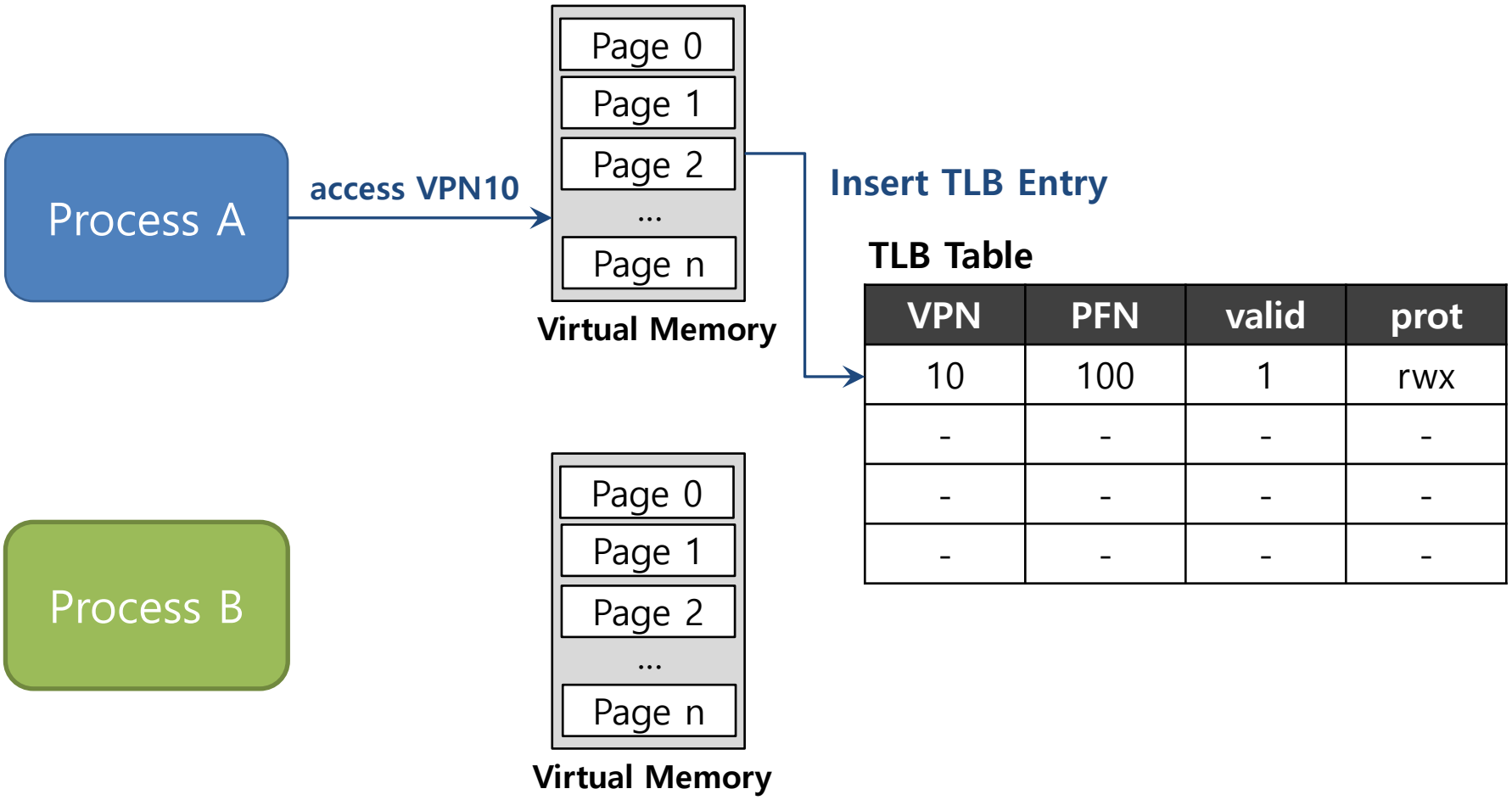
TLB entry

- TLB is managed by **Full Associative** method.
  - A typical TLB might have 32,64, or 128 entries.
  - Hardware search the entire TLB in parallel to find the desired translation.
  - other bits: valid bits , protection bits, address-space identifier, dirty bit



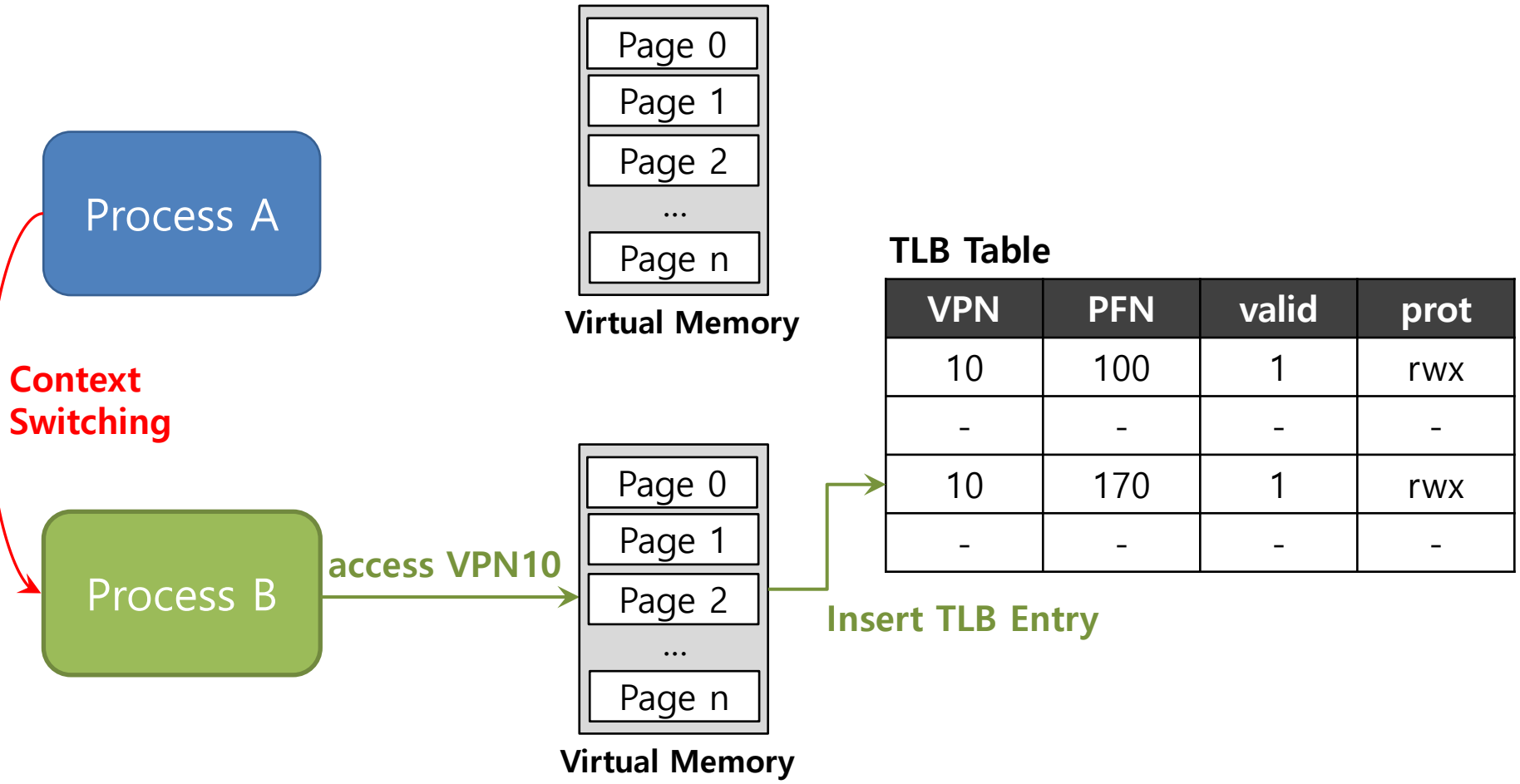
Typical TLB entry look like this

TLB Issue: Context Switching



VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
-	-	-	-
-	-	-	-

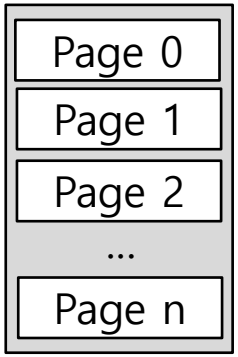
TLB Issue: Context Switching



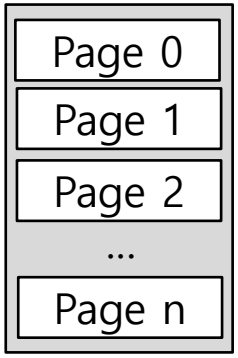
TLB Issue: Context Switching

Process A

Process B



Virtual Memory



Virtual Memory

TLB Table

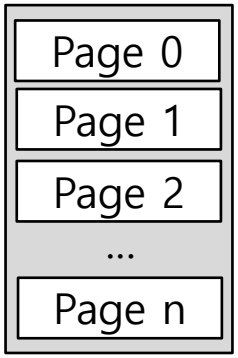
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

Can't distinguish which entry is meant for which process

To Solve Problem

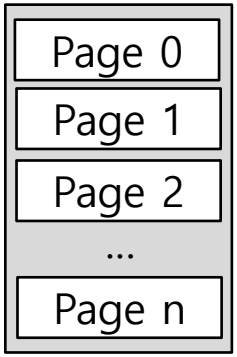
- Provide an address space identifier(ASID) field in the TLB.

Process A



Virtual Memory

Process B



Virtual Memory

TLB Table

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-

Another Case

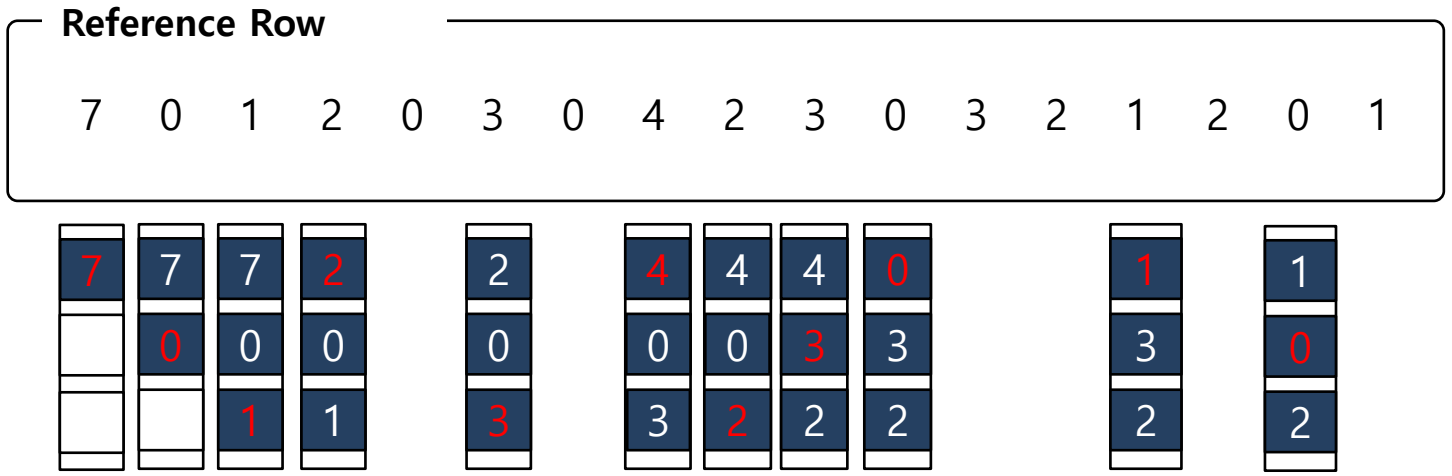
- Two processes share a page.
  - Process 1 is sharing physical page 101 with Process2.
  - P1 maps this page into the 10th page of its address space.
  - P2 maps this page to the 50th page of its address space.

VPN	PFN	valid	prot	ASID
10	101	1	rwx	1
-	-	-	-	-
50	101	1	rwx	2
-	-	-	-	-

Sharing of physical pages is useful as it reduces the number of physical pages in use.

TLB Replacement Policy

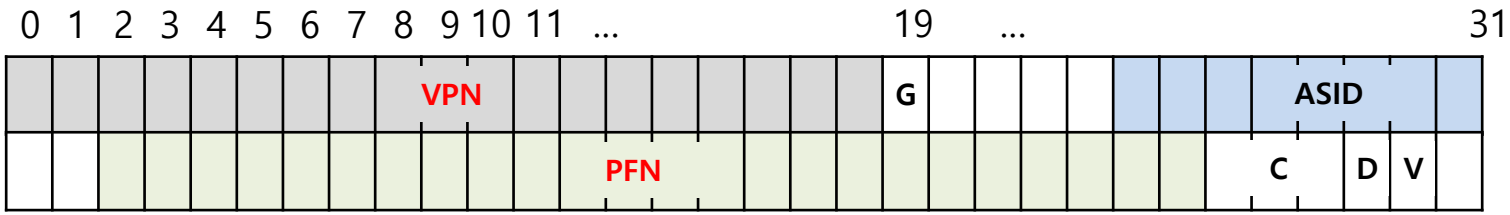
- LRU(Least Recently Used)
  - Evict an entry that has not recently been used.
  - Take advantage of *locality* in the memory-reference stream.



Total 11 TLB miss

A Real TLB Entry

All 64 bits of this TLB entry(example of MIPS R4000)

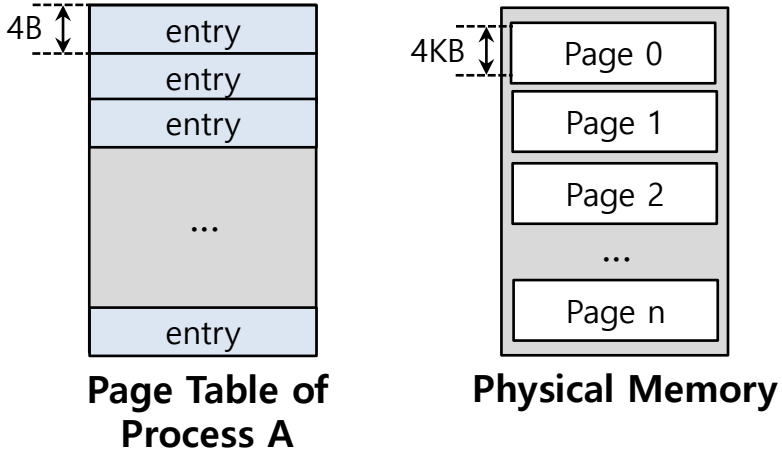


Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory($2^{24} * 4KB$ pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.

Part II: Advanced Paging

Paging: Linear Tables

- We usually have one page table for every process in the system.
  - Assume that 32-bit address space with 4KB pages and 4-byte page-table entry.

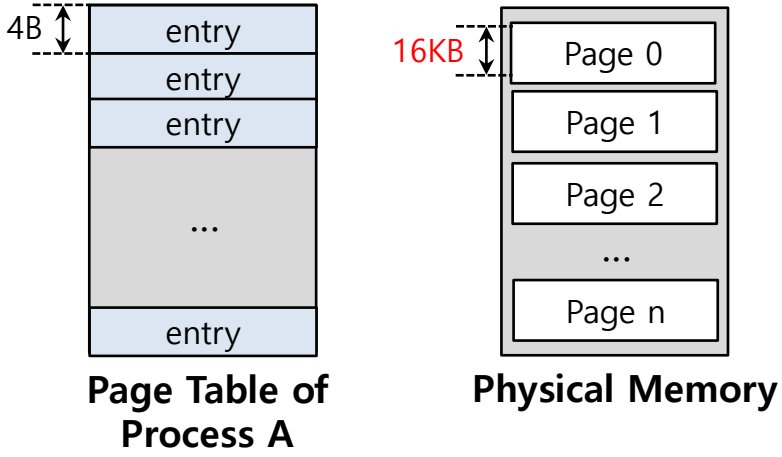


Page table size = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

Page tables are too big and thus consume too much memory.

Paging: Smaller Tables

- Page table are too big and thus consume too much memory.
  - Assume that 32-bit address space with 16KB pages and 4-byte page-table entry.

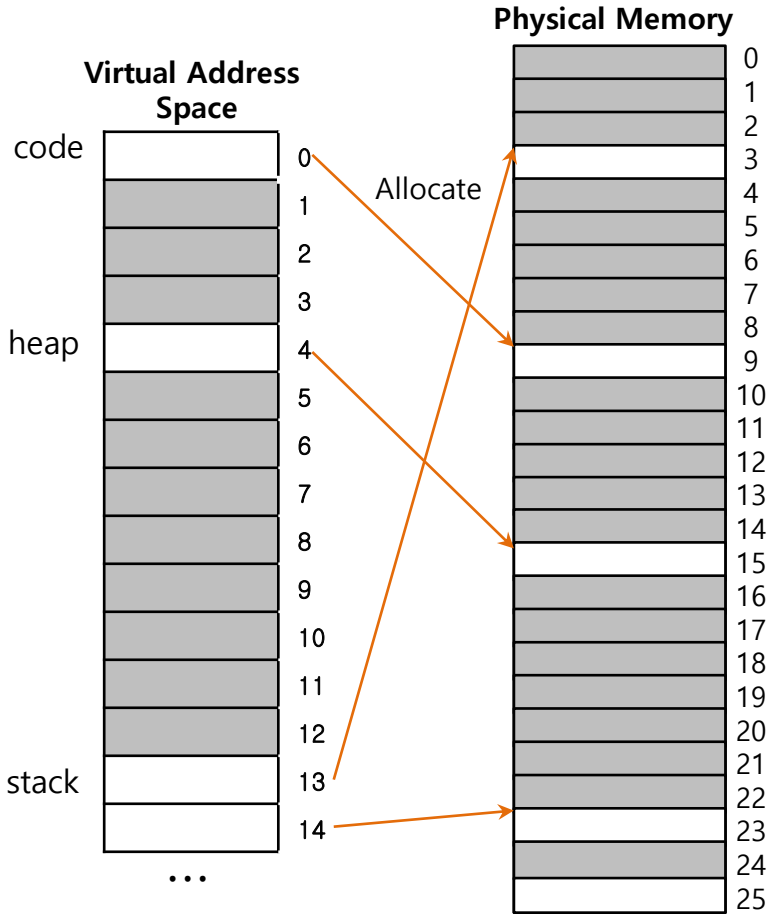


$\frac{2^{32}}{2^{14}} * 4 = 1MB$ per page table

Big pages lead to internal fragmentation.

Problem

- Single page table for the entries address space of a process.



A 16KB Address Space with 1KB Pages

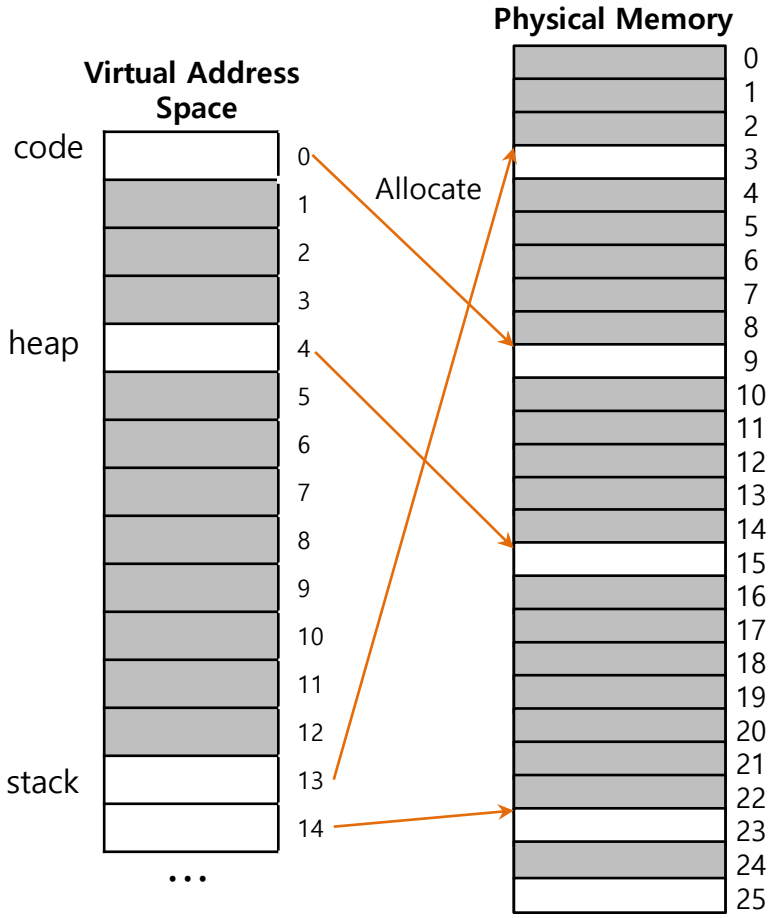
PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

...

A Page Table For 16KB Address Space

Problem

- Most of the page table is **unused**, full of invalid entries.



A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

...

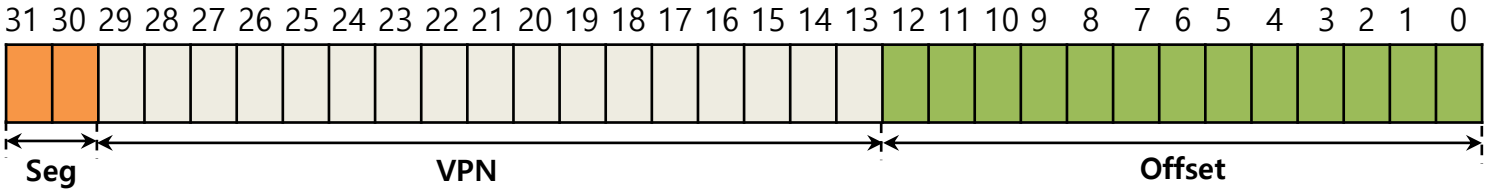
A Page Table For 16KB Address Space

Hybrid Approach: Paging and Segments

- In order to reduce the memory overhead of page tables.
  - Using base not to point to the segment itself but rather to hold the physical address of the page table of that segment.
  - The bounds register is used to indicate the end of the page table.

Simple Example of Hybrid Approach

- Each process has three page tables associated with it.
  - When process is running, the base register for each of these segments contains the physical address of a linear page table for that segment.



32-bit Virtual address space with 4KB pages

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

TLB miss on Hybrid Approach

- The hardware gets the **physical address** from the **page table**.
  - The hardware uses the segment bits(SN) to determine which base and bounds pair to use.
  - The hardware then takes the physical address therein and combines it with the VPN as follows to form the address of the page table entry(PTE) .

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

Problem of Hybrid Approach

- Hybrid Approach is not without problems.
  - If we have a large but sparsely-used heap, we can still end up with a lot of page table waste.
  - External fragmentation arises again.

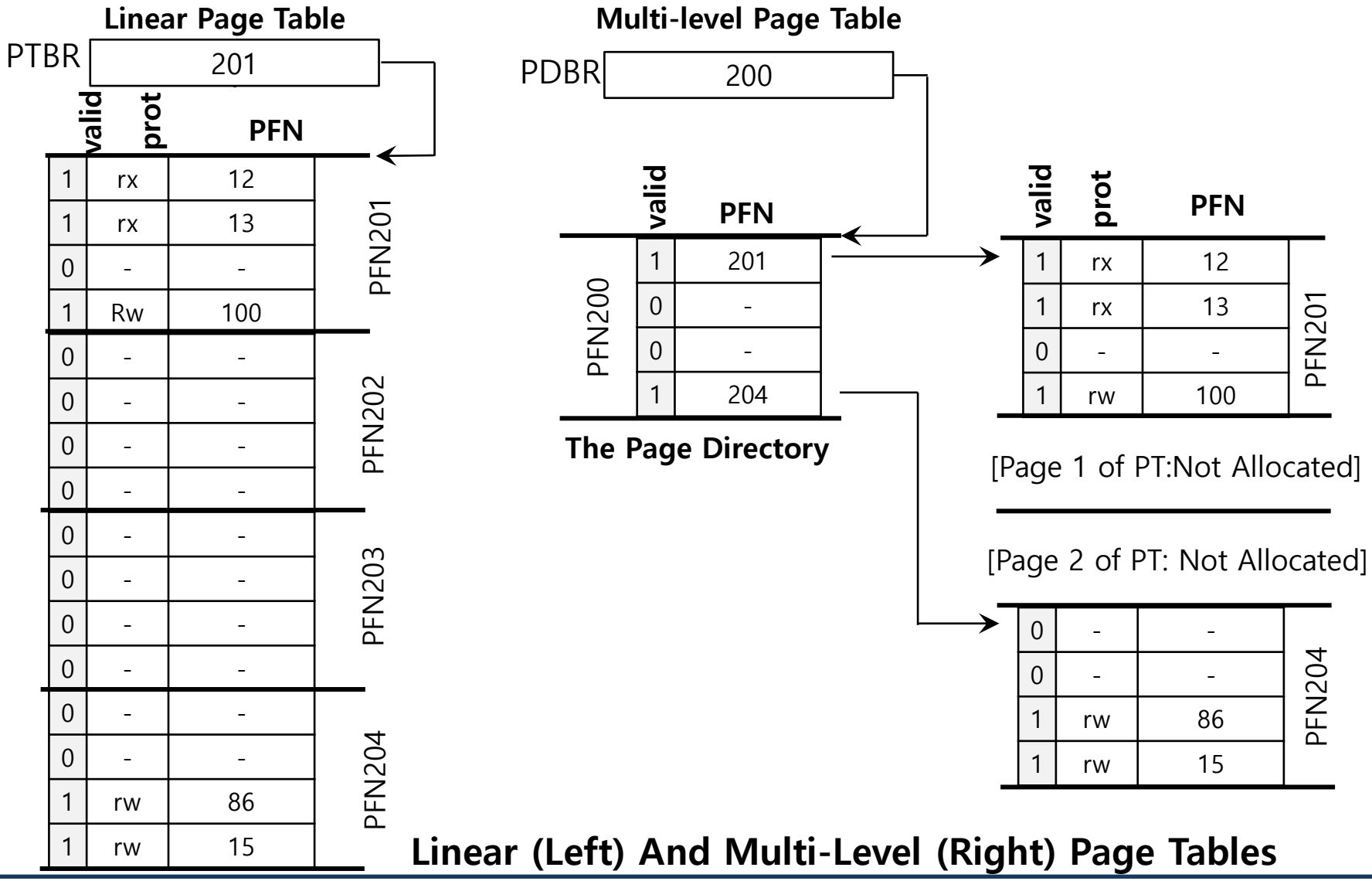
Multi-level Page Tables

- Turns the linear page table into something like a tree.
  - Chop up the page table into page-sized units.
  - If an entire page of page-table entries is invalid, don't allocate that page of the page table at all.
  - To track whether a page of the page table is valid, use a new structure, called page directory.

Multi-level Page Tables: Page directory

PTBR=Page Table Base Address

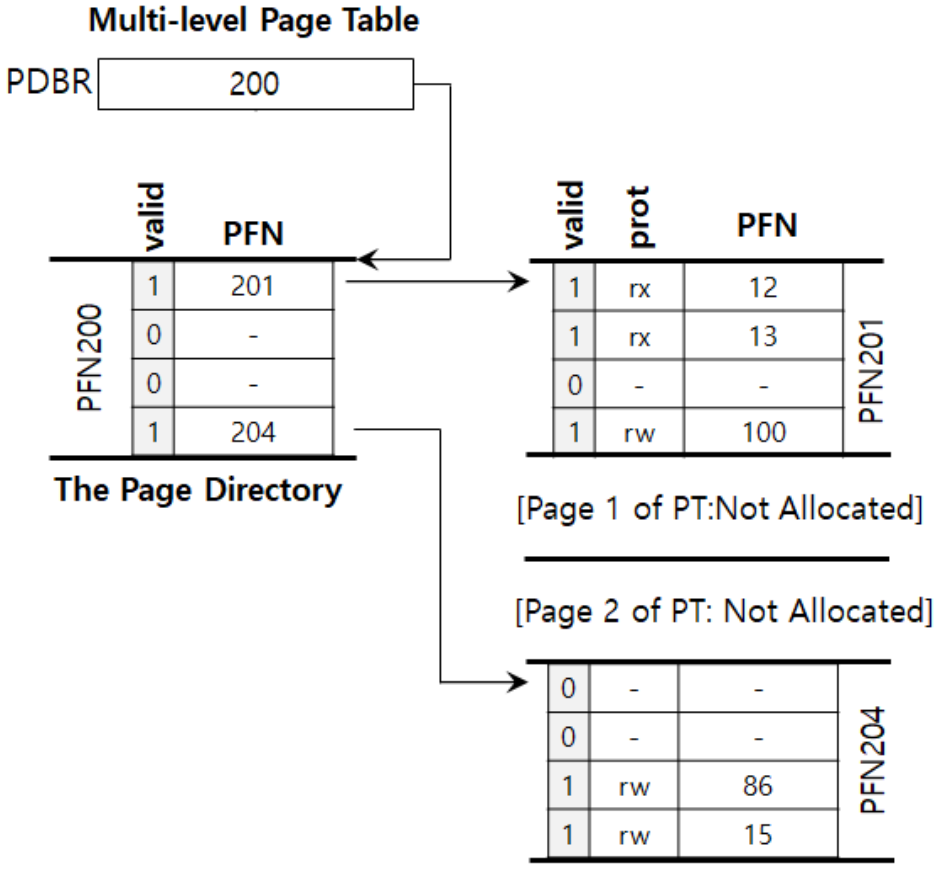
PDBR=Page Directory Base Address



Linear (Left) And Multi-Level (Right) Page Tables

Multi-level Page Tables: Page directory entries

- The page directory contains one entry per page of the page table.
  - It consists of a number of page directory entries(PDE).
- PDE has a valid bit and page frame number(PFN).



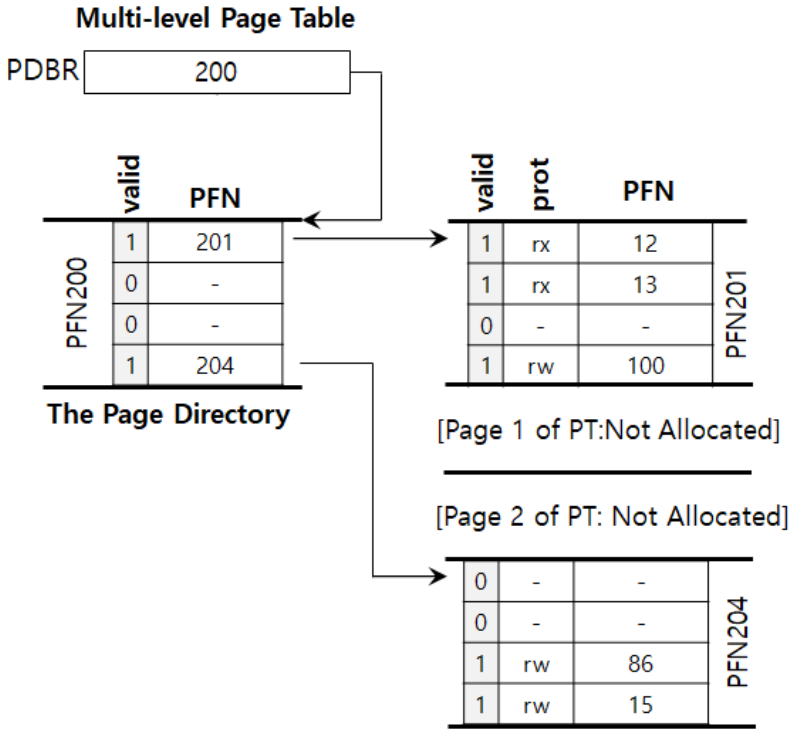
Multi-level Page Tables: Advantage & Disadvantage

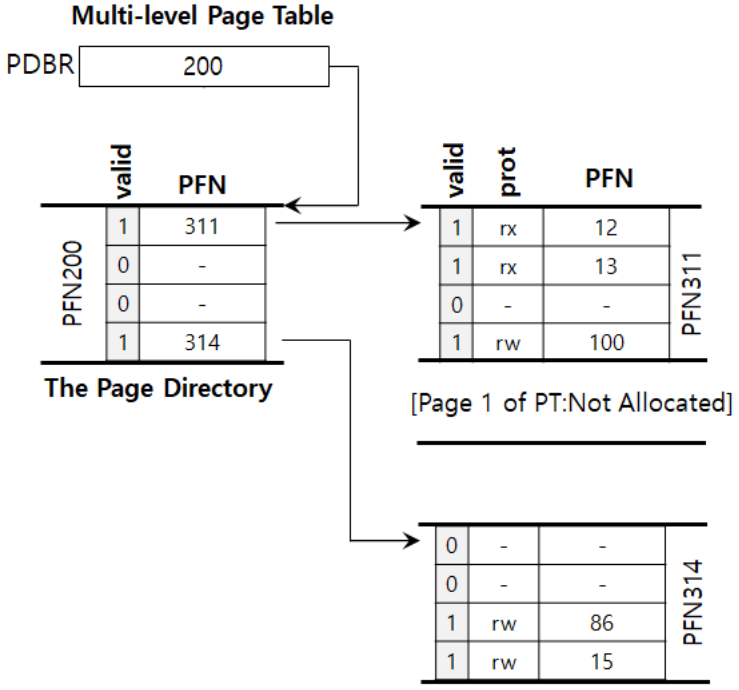
- Advantage
  - Only allocates the page-table space in proportion to the amount of address space you are using.
  - The OS can grab the next free page when it needs to allocate or grow a page table.

- Disadvantage
  - Multi-level table is a small example of a time-space trade-off.
  - Complexity.

Multi-level Page Table: Level of indirection

- A multi-level structure can adjust level of indirection through use of the page directory.
  - Indirection can help us place page-table pages wherever we like in physical memory.



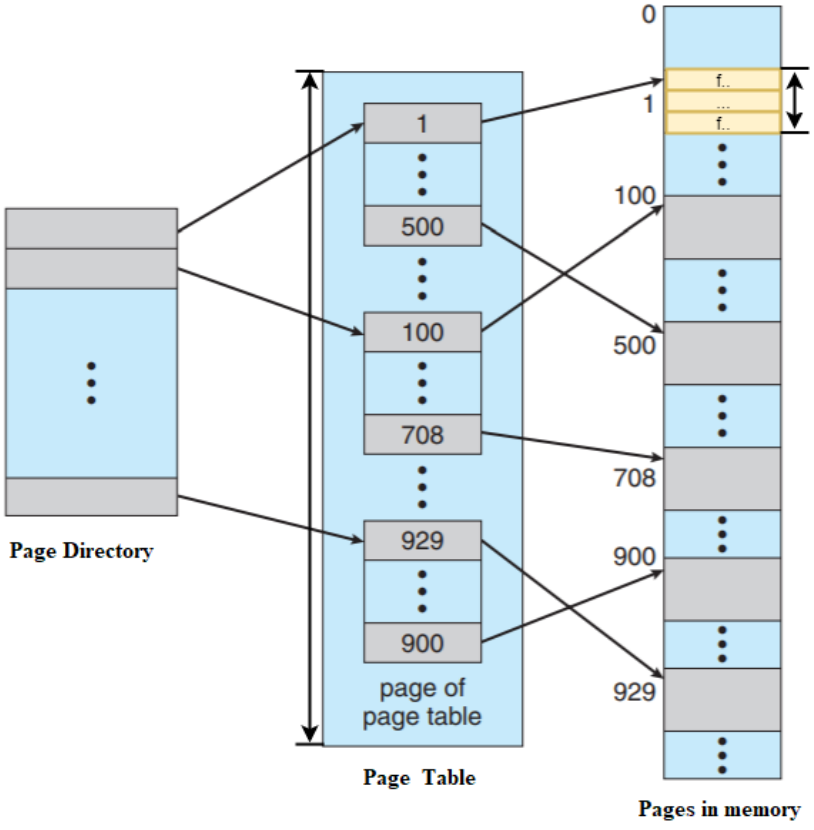


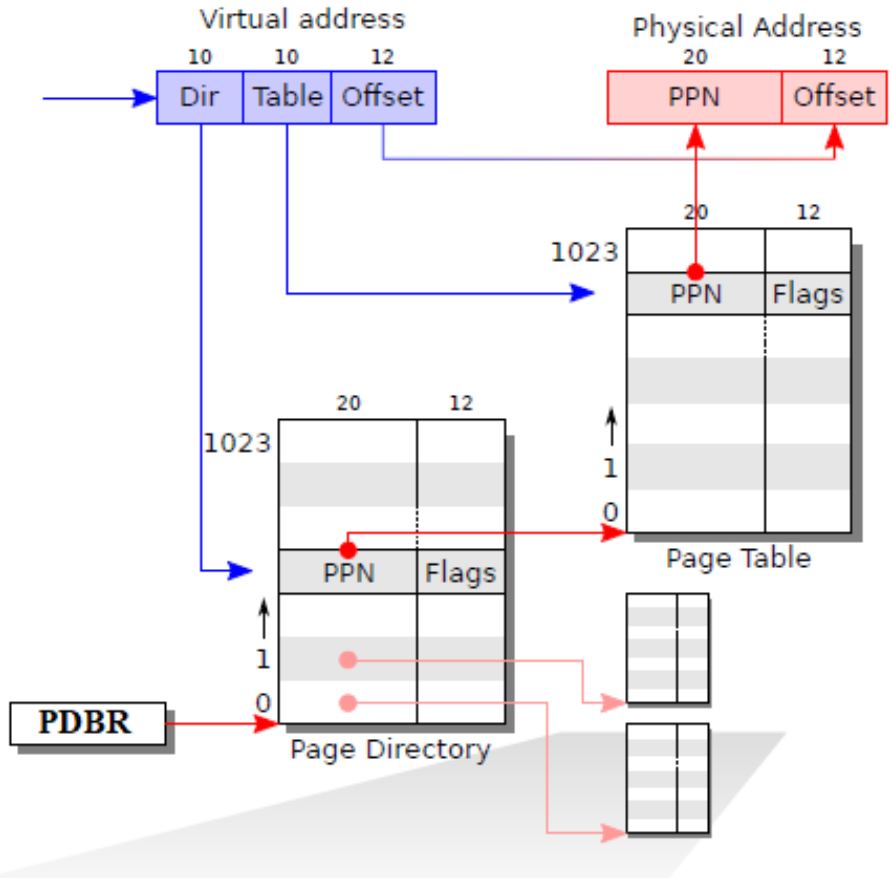
Example

- Assumptions:
  - Each page is 4 KB (12 bits to represent its offsets from 0x 000 - 0x fff)
  - The virtual space of a process: 4GB (32-bit to represent from 0x 00000000 – 0x ffffffff)
  - Each entry in the page table is 4 B
  - How big for the page table to one process (map 4GB virtual space to 4GB physical space)
    - How many entries in the page table: 4GB/4KB = 1 M
    - How big: 1M * 4 B = 4 MB
  - If we have one thousand of processes running
    - The page table of each process is 4MB
    - The page tables of one thousand of processes is 4GB (ONLY for tables)

Example

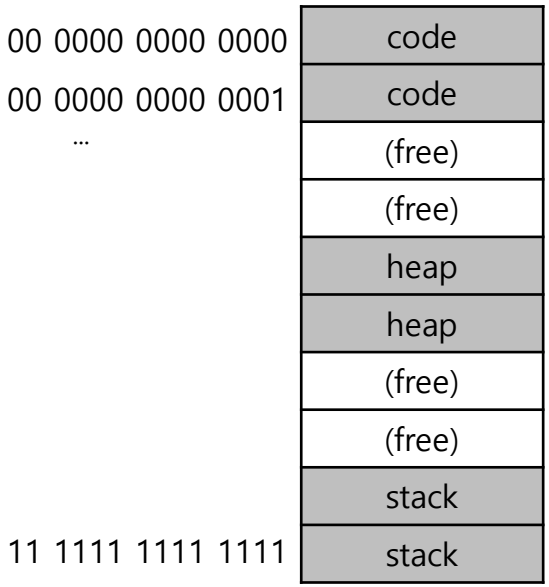
- Idea: direct mapping to indirect mapping





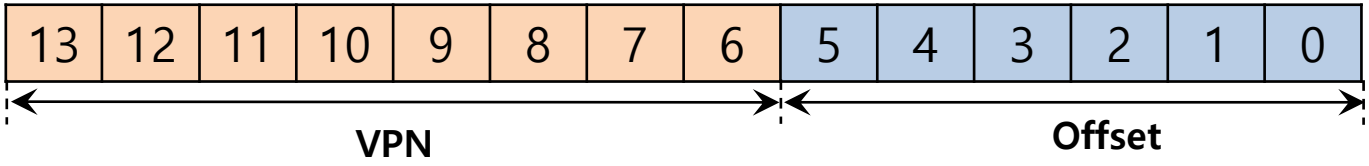
A Detailed Multi-Level Example

- To understand the idea behind multi-level page tables better, let's do an example.



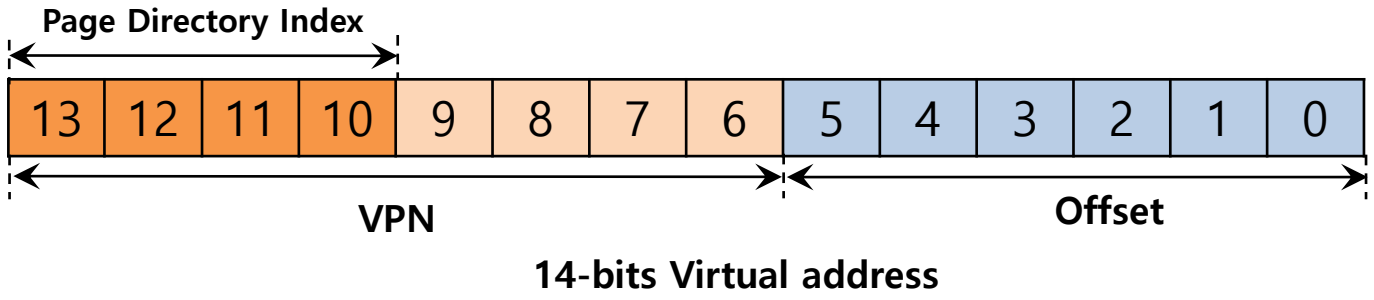
Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	$2^8(256)$

A 16-KB Address Space With 64-byte Pages



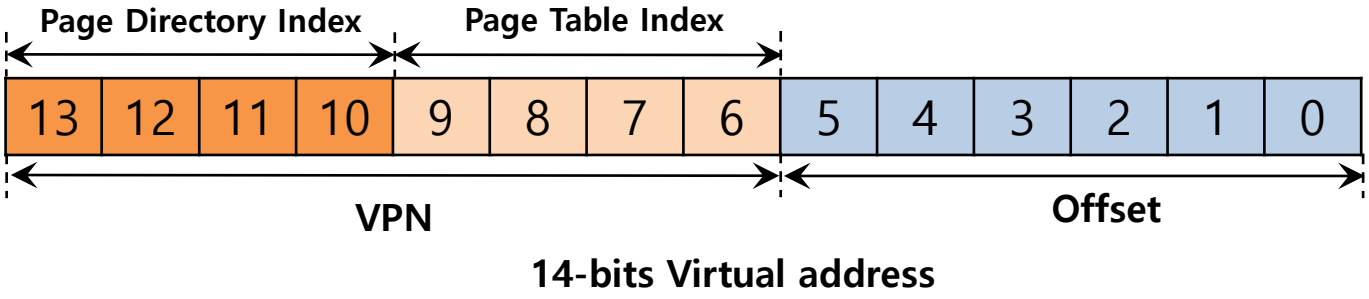
A Detailed Multi-Level Example: Page Directory Idx

- The page directory needs one entry (4 bytes) per page of the page table
  - it has 16 entries.
- The page-directory entry is invalid → Raise an exception (The access is invalid)

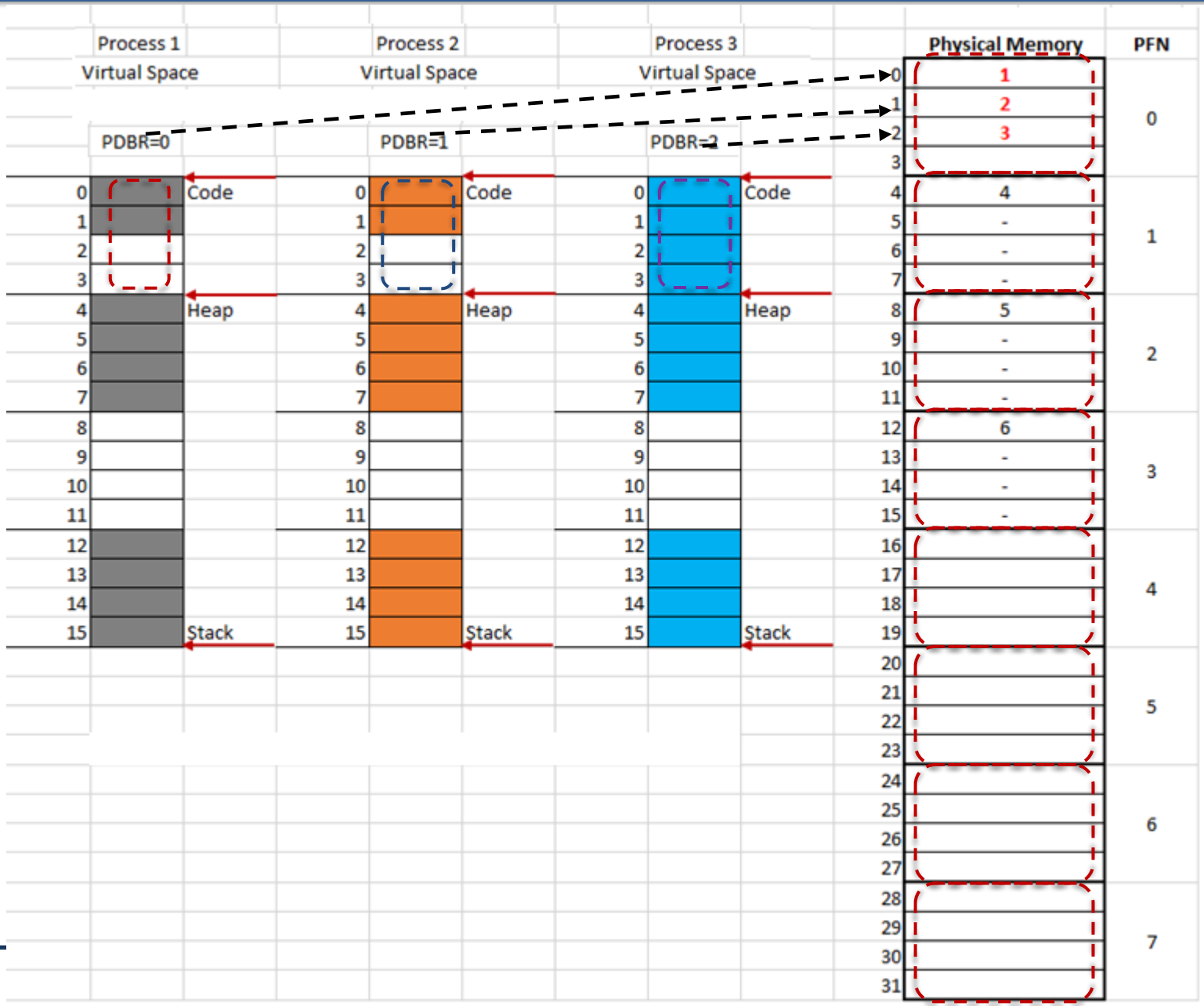


A Detailed Multi-Level Example: Page Table Idx

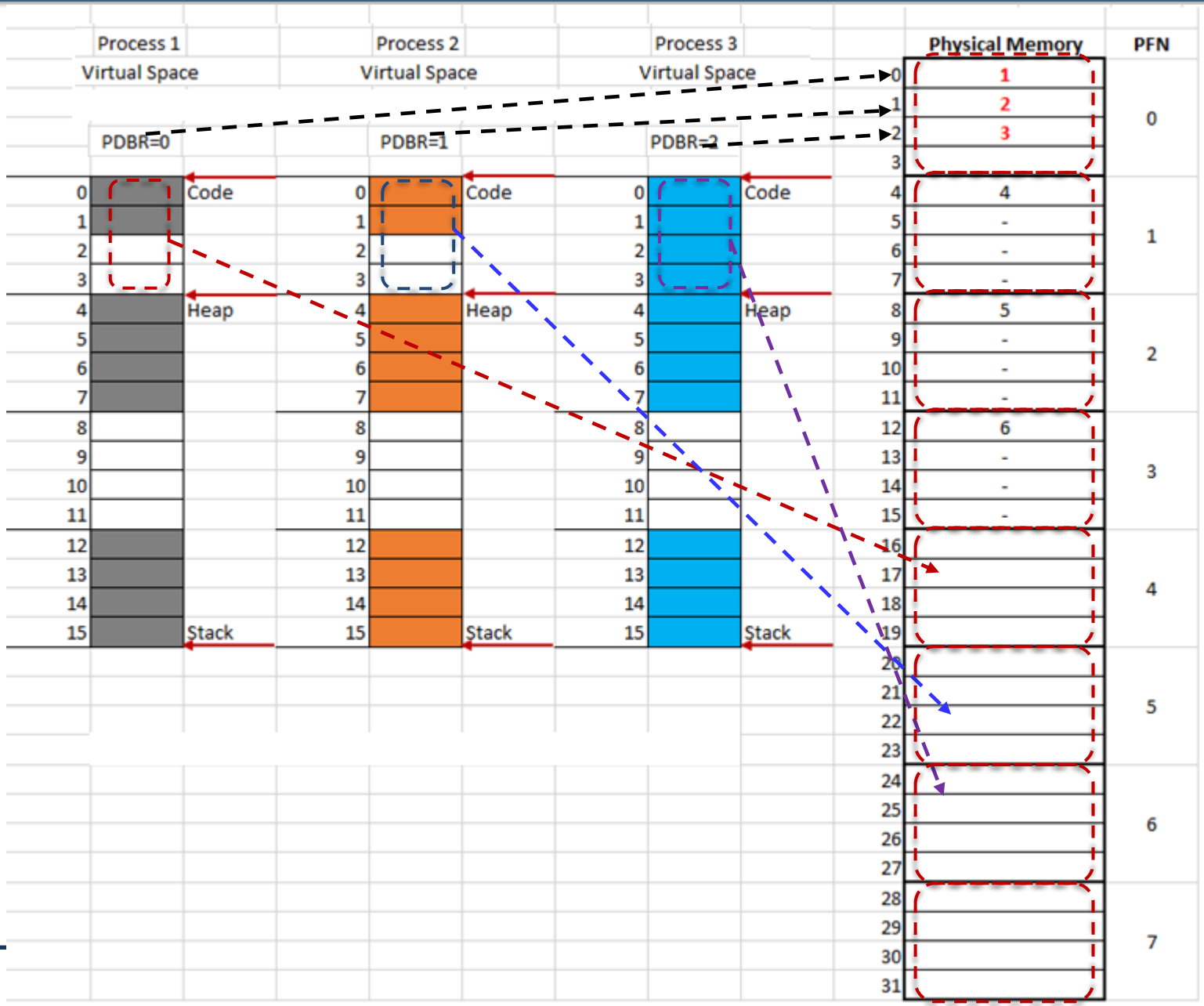
- The PDE is valid, we have more work to do.
  - To fetch the page table entry(PTE) from the page of the page table pointed to by this page-directory entry.
- This page-table index can then be used to index into the page table itself.



Example


Process 1
Virtual Space
PDBR=0
Process 2
Virtual Space
PDBR=1
Process 3
Virtual Space
PDBR=2
Code
Heap
Stack
Physical Memory
PFN
1
2
3
4
5
6
0
1
2
3
4
5
6
7

Example


Process 1
Virtual Space
PDBR=0
Process 2
Virtual Space
PDBR=1
Process 3
Virtual Space
PDBR=2
Code
Heap
Stack
Physical Memory
PFN
1
2
3
4
5
6
0
1
2
3
4
5
6
7

Summary

- TLB (Translation Lookaside Buffer)
  - Part of the chip's memory-management unit(MMU).
  - A hardware cache of popular virtual-to-physical address translation.
- Advanced Paging
  - Big page size
  - Hybrid approach (paging + segmentation)
  - Multi-level page table
- Next: Swapping (Chapters 21, 22)