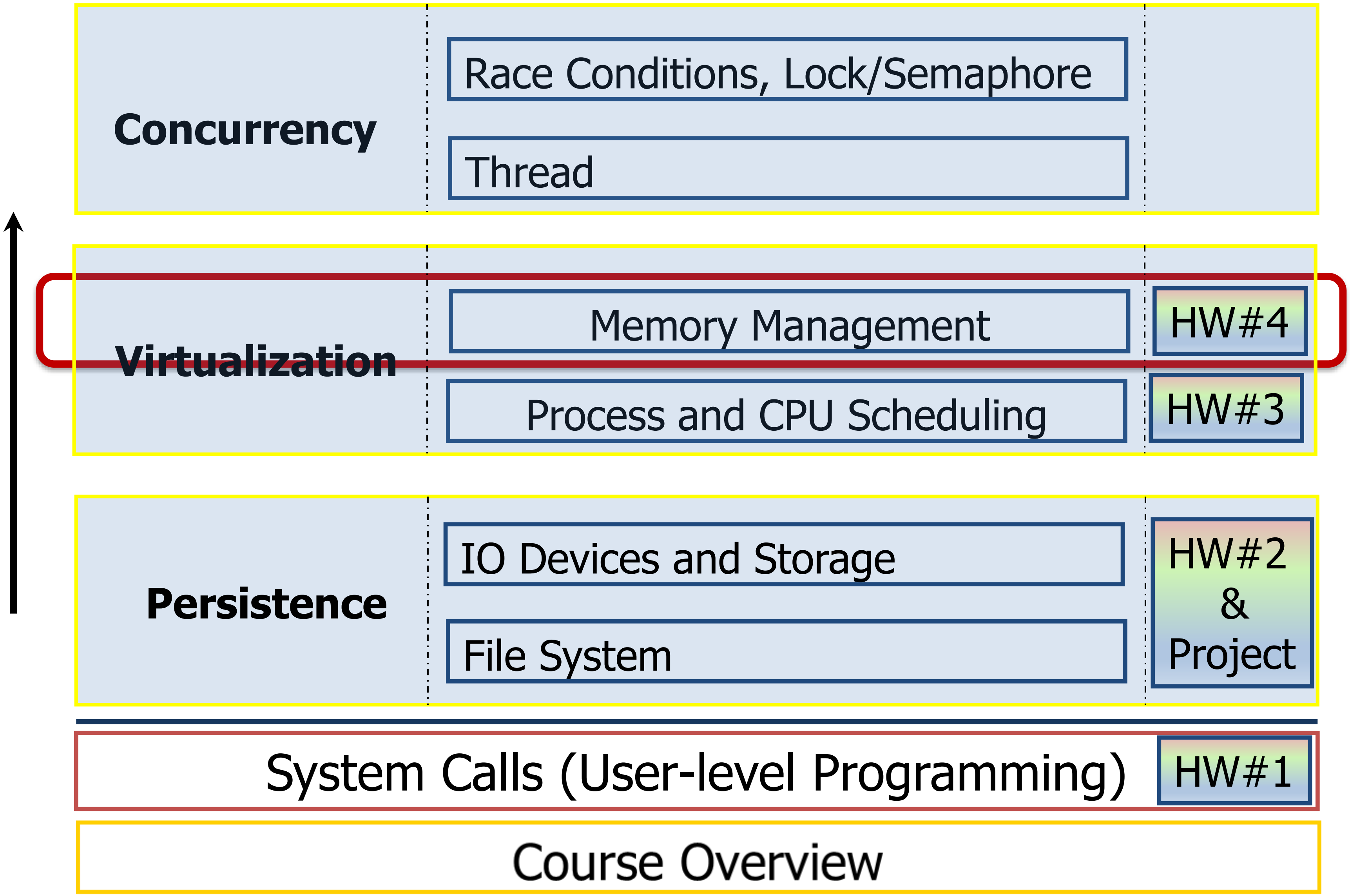
Lecture 10: Virtualizing Memory – Segmentation and Paging

The Course Organization (Bottom-up)

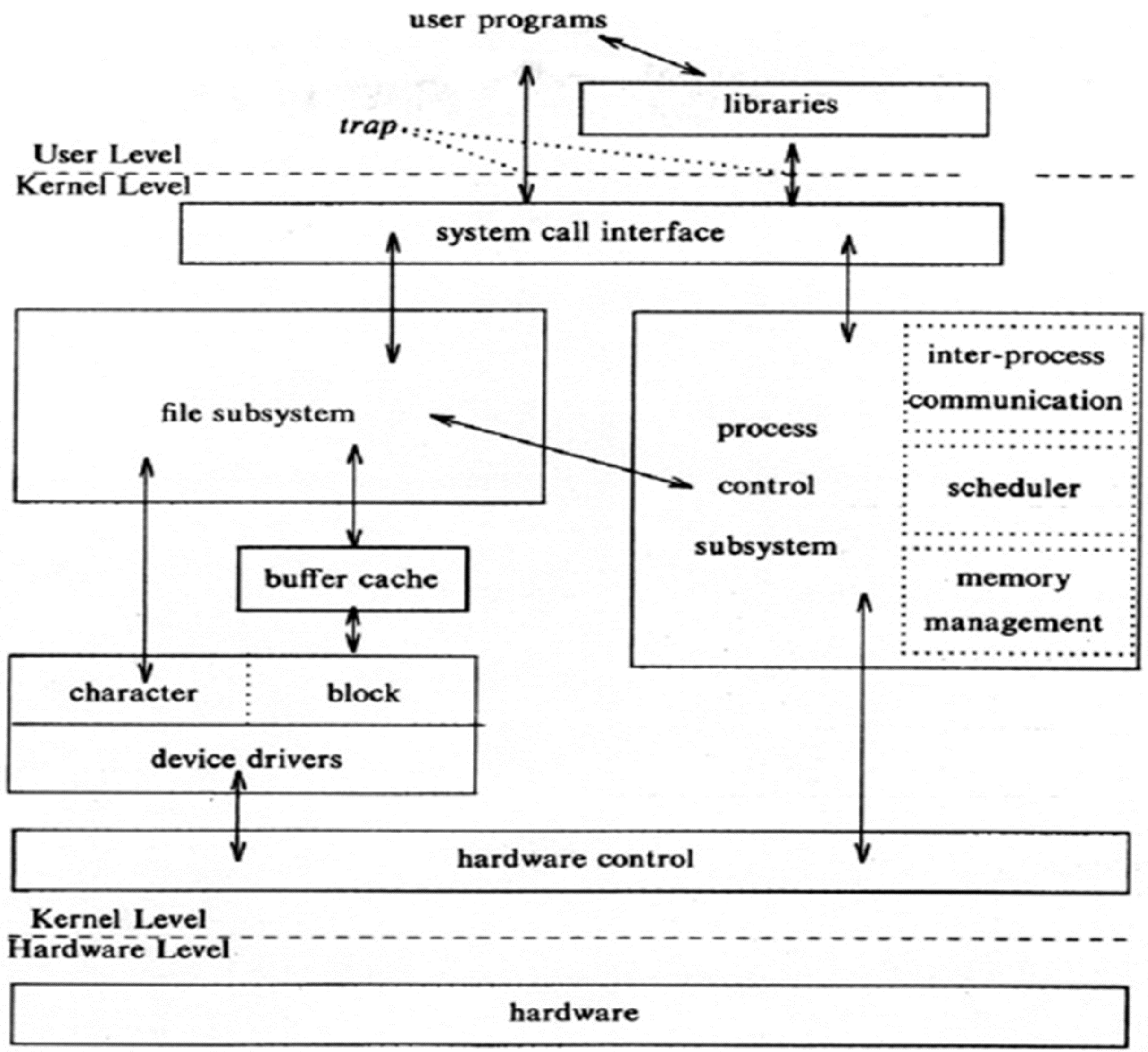


OS – Resource management via virtualization

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**.

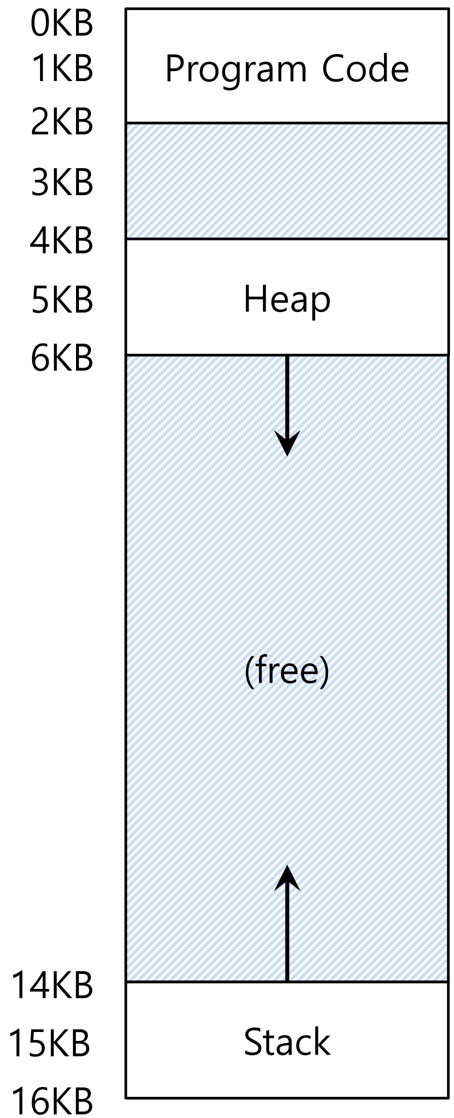
- many programs to run (processes) → Sharing the CPU
- many processes to *concurrently* access their own instructions and data → Sharing memory
- many processes to access devices → Sharing disks



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

Part I: Segmentation

Inefficiency of the Base and Bound Approach



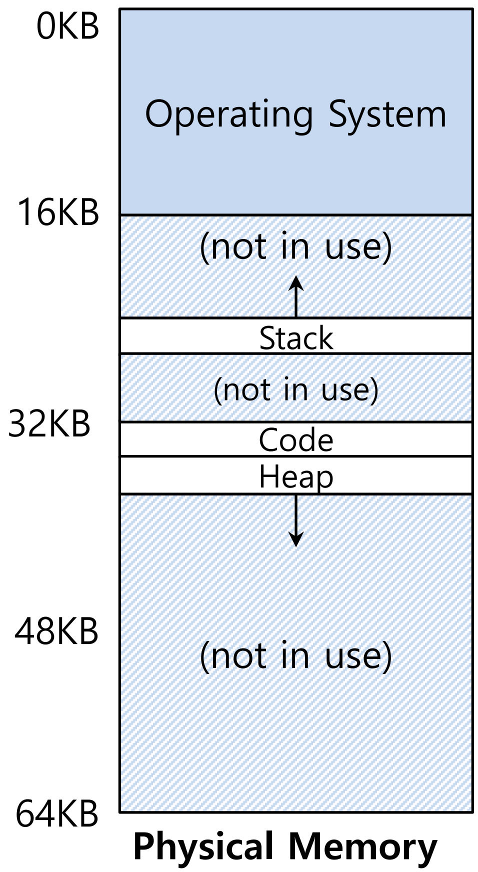
- **Big chunk of "free"** space
- "free" space **takes up** physical memory.
- Hard to run when an address space **does not fit** into physical memory

Segmentation

- Segment is just **a contiguous portion** of the address space of a particular length.
  - Logically-different segment: code, stack, heap

- Each segment can be **placed** in **different part of physical memory**.
  - **Base** and **bounds** exist **per each segment**.

Placing Segment In Physical Memory



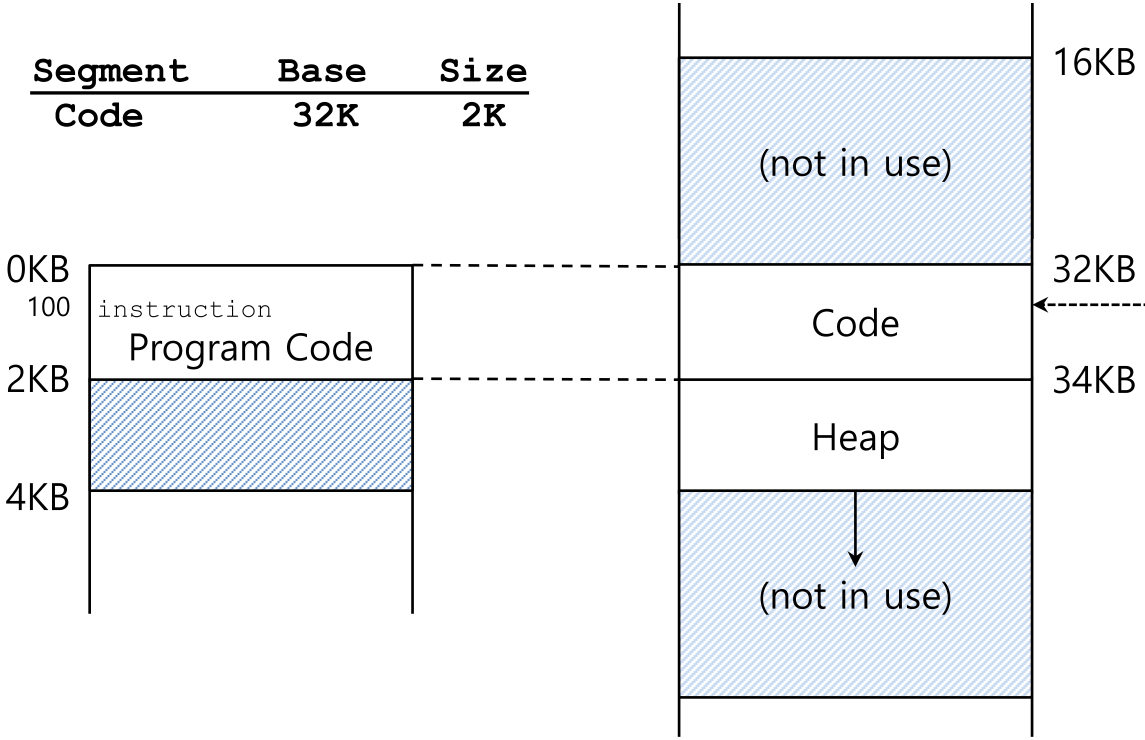
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Address Translation on Segmentation

$$physical\ address = offset + base$$

- The `offset` of virtual address 100 is `100`.
  - The code segment **starts at virtual address 0** in address space.

Segment	Base	Size
Code	32K	2K



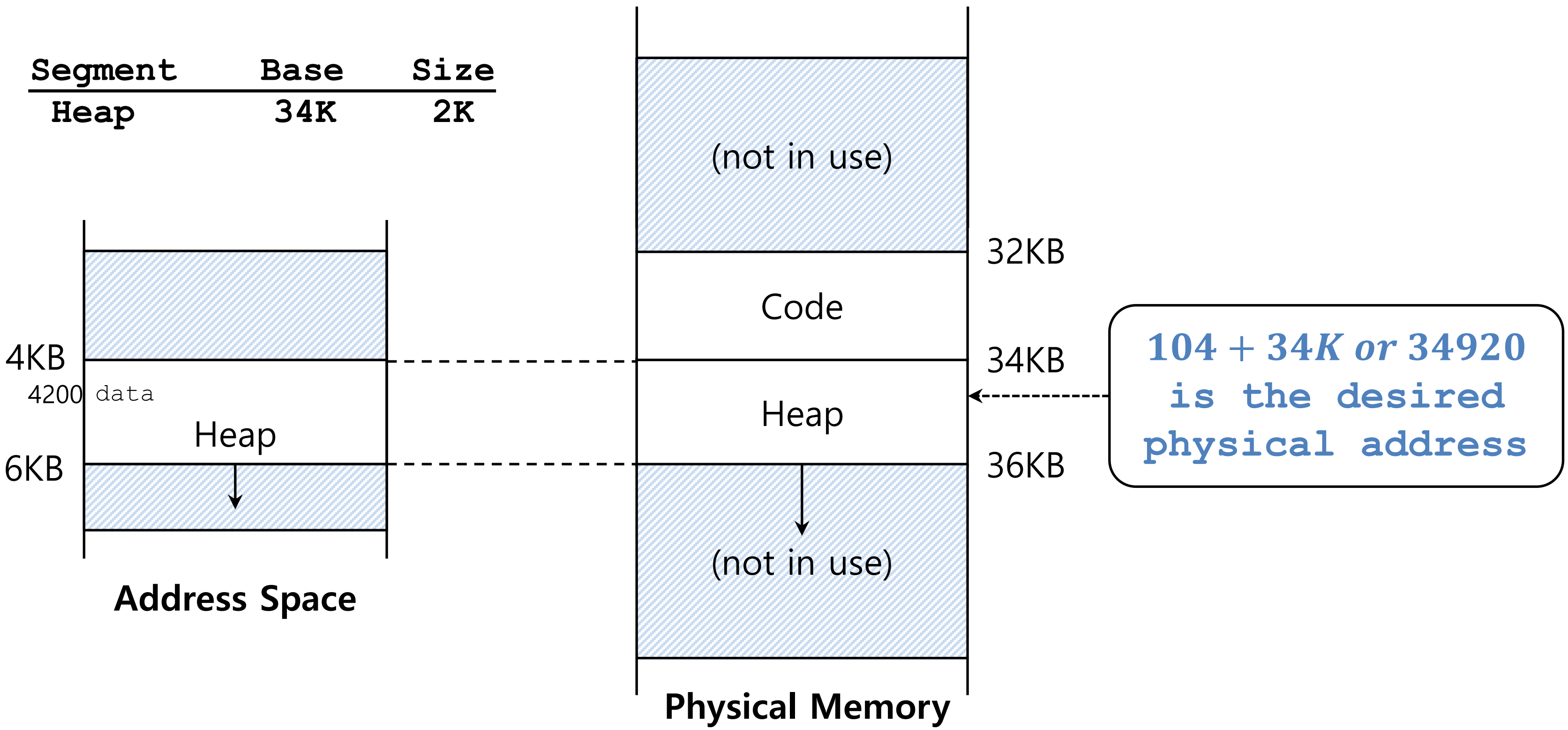
$100 + 32K$ or 32868 **is the desired physical address**

Address Translation on Segmentation(Cont.)

Virtual address + base `is not the correct physical address.`

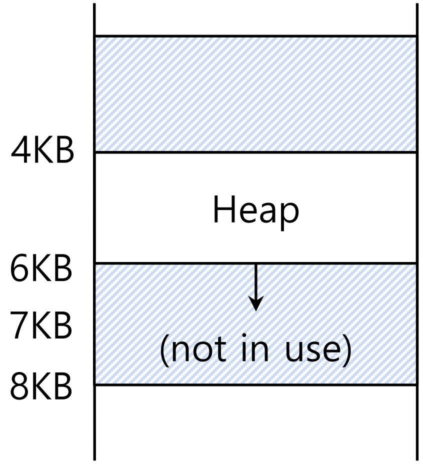
- The `offset` of virtual address 4200 is `104`.
  - The heap segment **starts at virtual address 4096** in address space.

Segment	Base	Size
Heap	34K	2K



Segmentation Fault or Violation

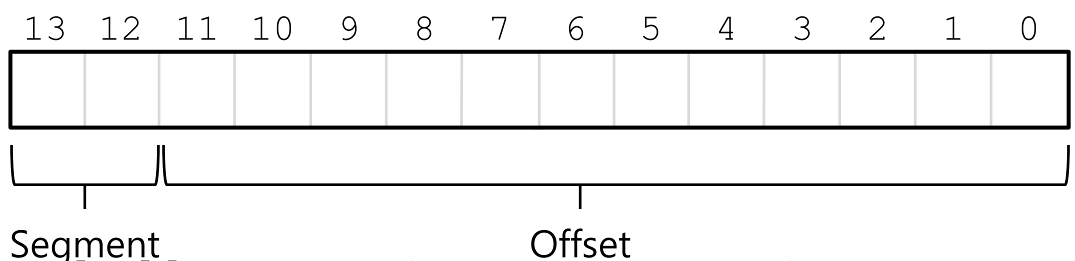
- If an **illegal address** such as 7KB which is beyond the end of heap is referenced, the OS occurs **segmentation fault**.
  - The hardware detects that the address is **out of bounds**.



Address Space

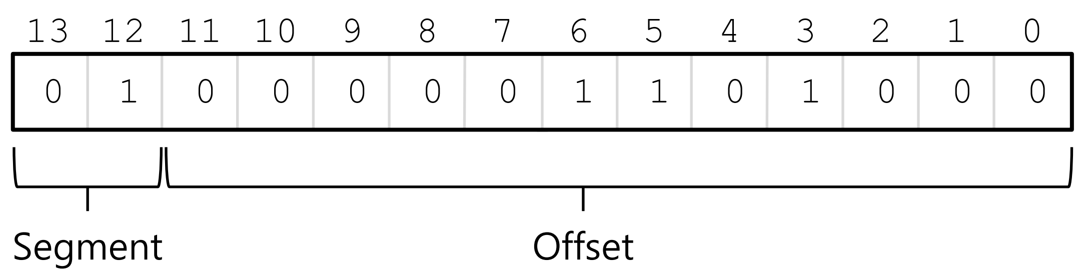
Referring to Segment

- **Explicit approach**
  - Chop up the address space into segments based on the **top few bits** of virtual address.



- Example: virtual address 4200 (01000001101000)

Segment	bits
Code	00
Heap	01
Stack	10
-	11



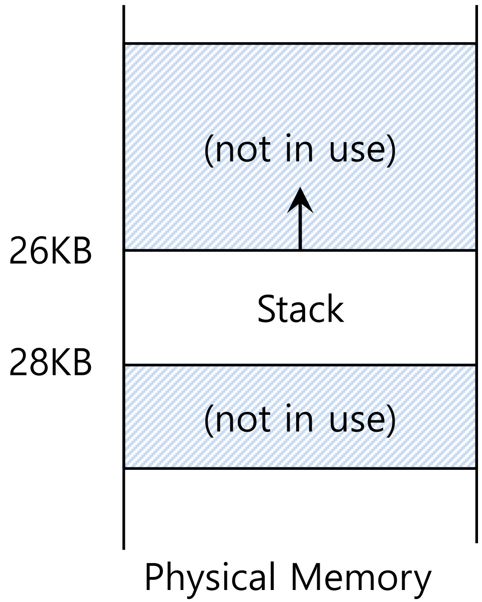
Referring to Segment(Cont.)

```
// get top 2 bits of 14-bit VA
Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
// now get offset
Offset = VirtualAddress & OFFSET_MASK
if (Offset >= Bounds[Segment])
     RaiseException(PROTECTION_FAULT)
else
     PhysAddr = Base[Segment] + Offset
     Register = AccessMemory(PhysAddr)
```

- SEG_MASK = 0x3000(11000000000000)
- SEG_SHIFT = 12
- OFFSET_MASK = 0xFFF (00111111111111)

Referring to Stack Segment

- Stack grows **backward**.
- **Extra hardware support** is need.
  - The hardware checks which way the segment grows.
  - 1: positive direction, 0: negative direction



Segment Register(with Negative-Growth Support)

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Support for Sharing

- Segment can be **shared between address** space.
  - **Code sharing** is still in use in systems today.
  - by extra hardware support.
- Extra hardware support is need for form of **Protection bits.**
  - **A few more bits** per segment to indicate **permissions** of **read,** write and **execute**.

Segment Register Values(with Protection)

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Fine-Grained and Coarse-Grained

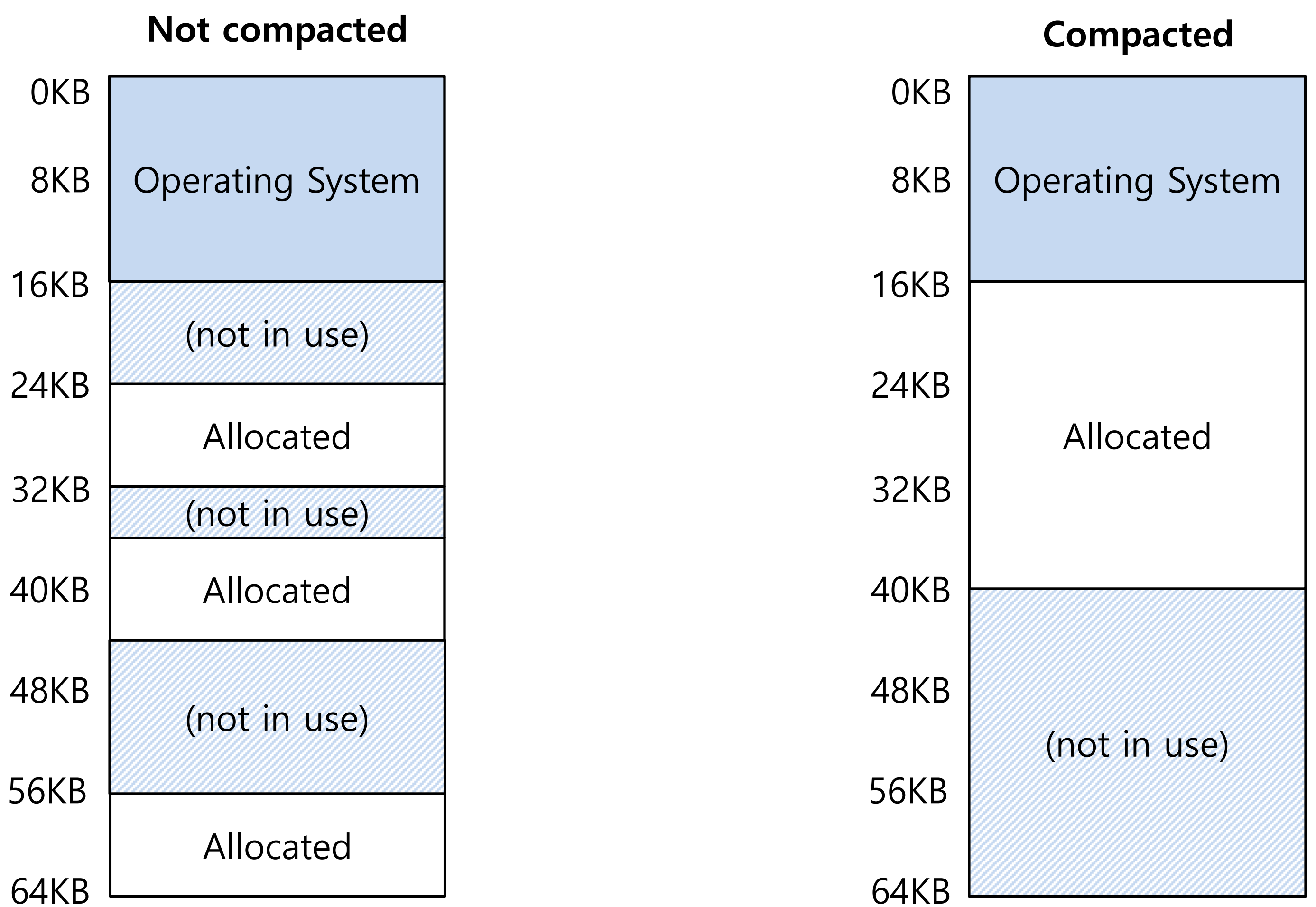
- **Coarse-Grained** means segmentation in a small number.
  - e.g., code, heap, stack.
- **Fine-Grained** segmentation allows **more flexibility** for address space in some early system.
  - To support many segments, Hardware support with a **segment table** is required.

OS support: Fragmentation

- **External Fragmentation**: little holes of **free space** in physical memory that make difficulty to allocate new segments.
  - There is **$\mathfrak{X}$KB free**, but **not in one contiguous** segment.
  - The OS **cannot** satisfy the **$\mathfrak{X}$ KB request**.

- **Compaction**: **rearranging** the exiting segments in physical memory.
  - Compaction is **costly**.
    - **Stop** running process.
    - **Copy** data to somewhere.
    - **Change** segment register value.

Memory Compaction



Part II: Paging

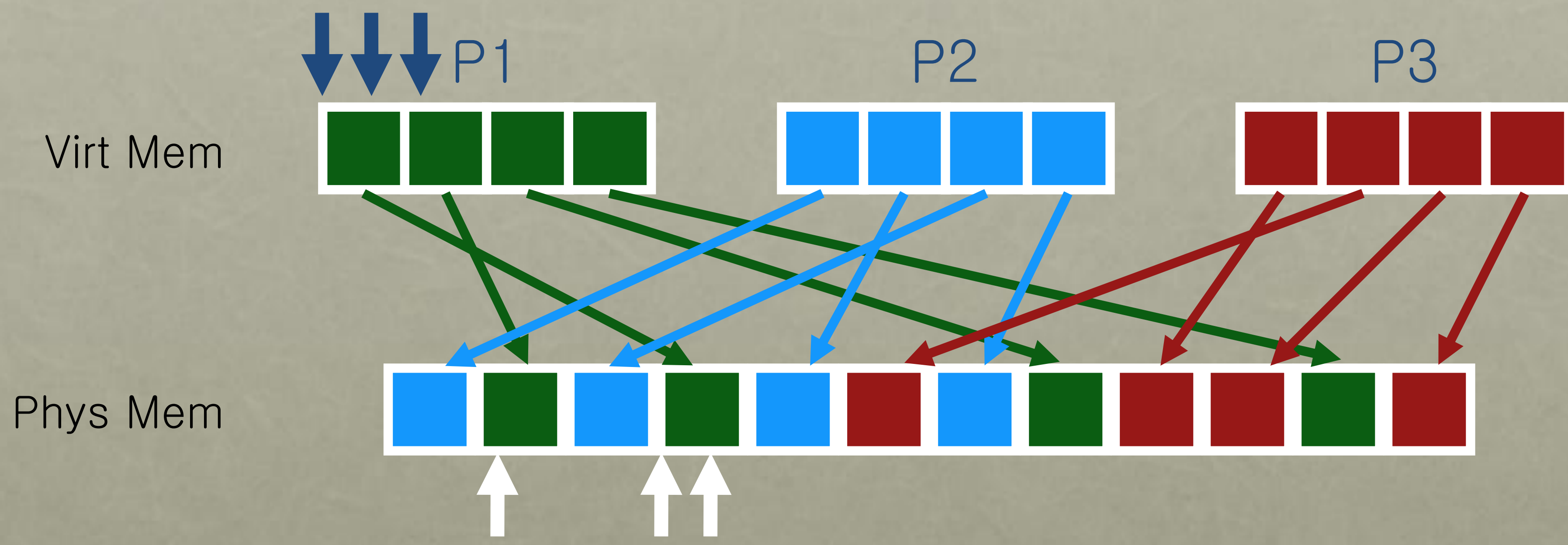
Concept of Paging

- Paging **splits up** address space into **fixed-sized** unit called a **page**.
  - Segmentation: variable size of logical segments(code, stack, heap, etc.)

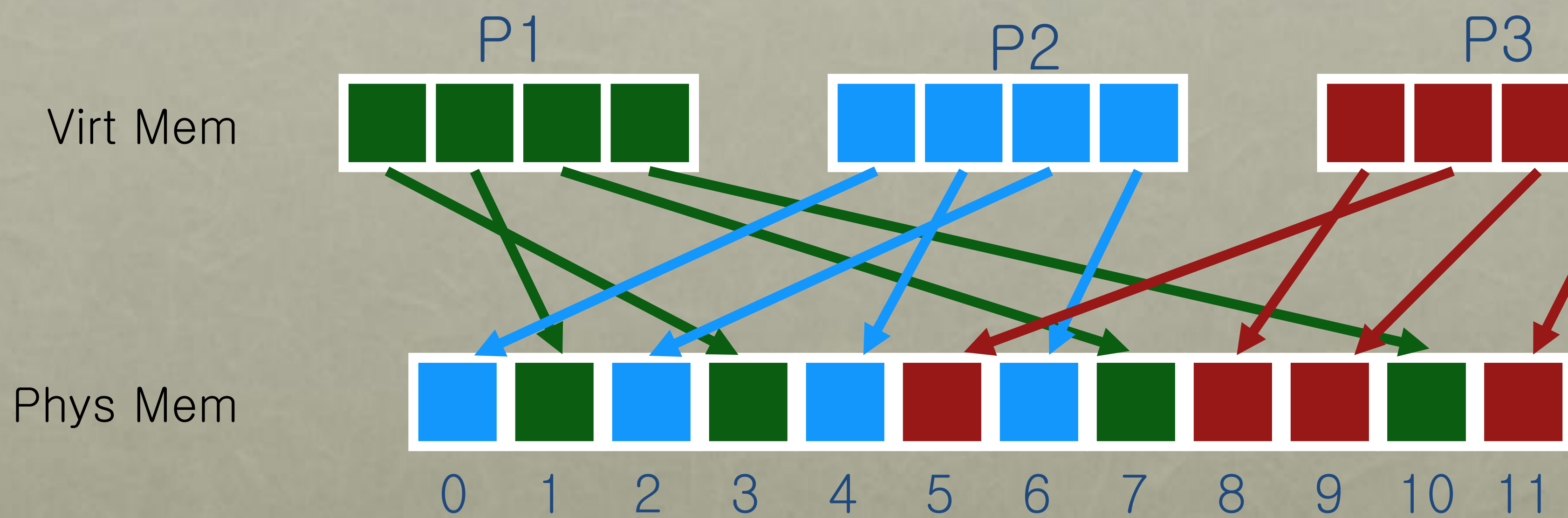
- With paging, **physical memory** is also **split** into some number of pages called a **page frame**.

- **Page table** per process is needed **to translate** the virtual address to physical address.

Paging



Fill in Page Table



Page Tables:

P1	P2	P3
3	0	8
1	4	5
7	2	9
10	6	11

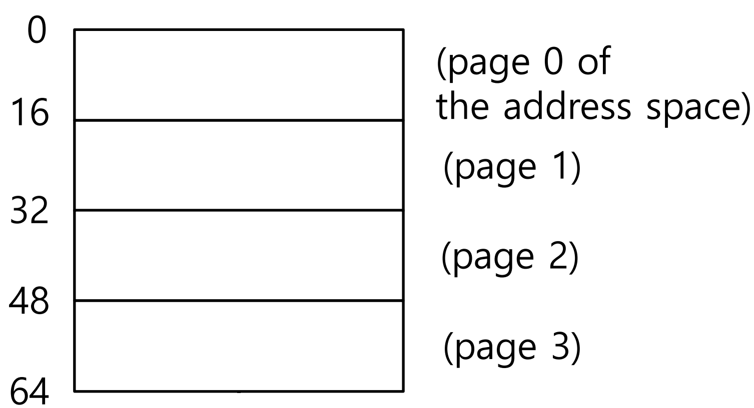
Advantages Of Paging

- **Flexibility:** Supporting the abstraction of address space effectively
  - Don’t need assumption how heap and stack grow and are used.

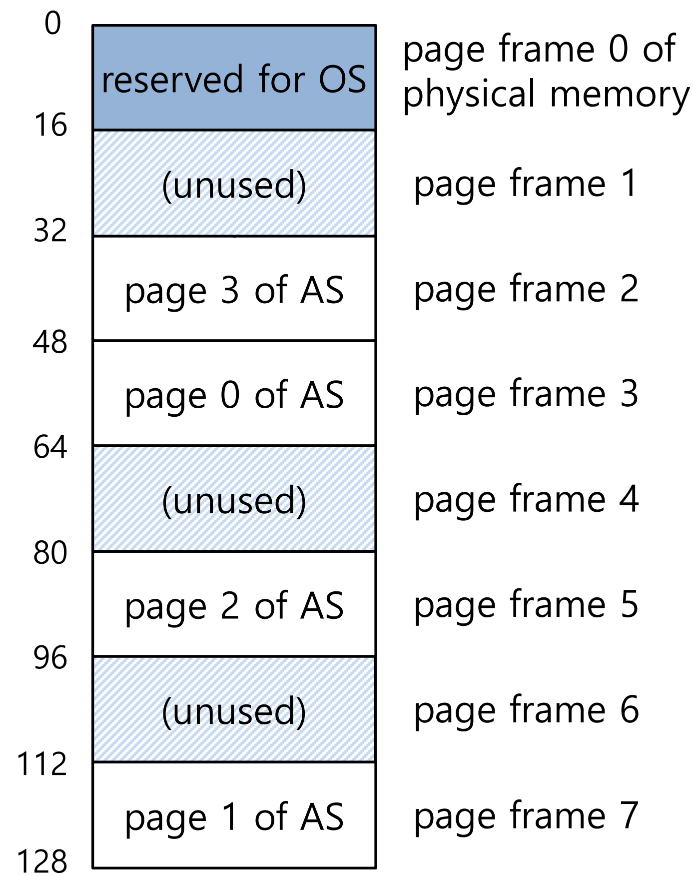
- **Simplicity**: ease of free-space management
  - The page in address space and the page frame are the same size.
  - Easy to allocate and keep a free list

Example: A Simple Paging

- 128-byte physical memory with 16 bytes page frames
- 64-byte address space with 16 bytes pages



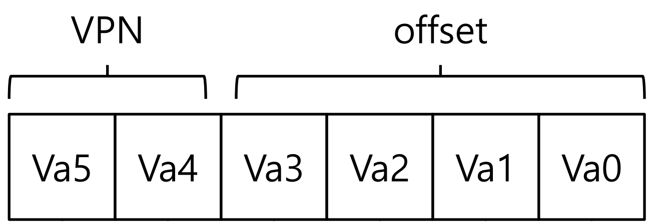
A Simple 64-byte Address Space



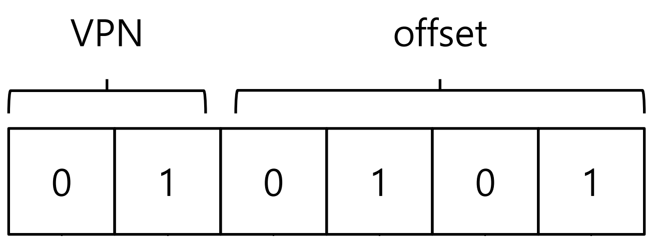
64-Byte Address Space Placed In Physical Memory

Address Translation

- Two components in the virtual address
  - VPN: virtual page number
  - Offset: offset within the page

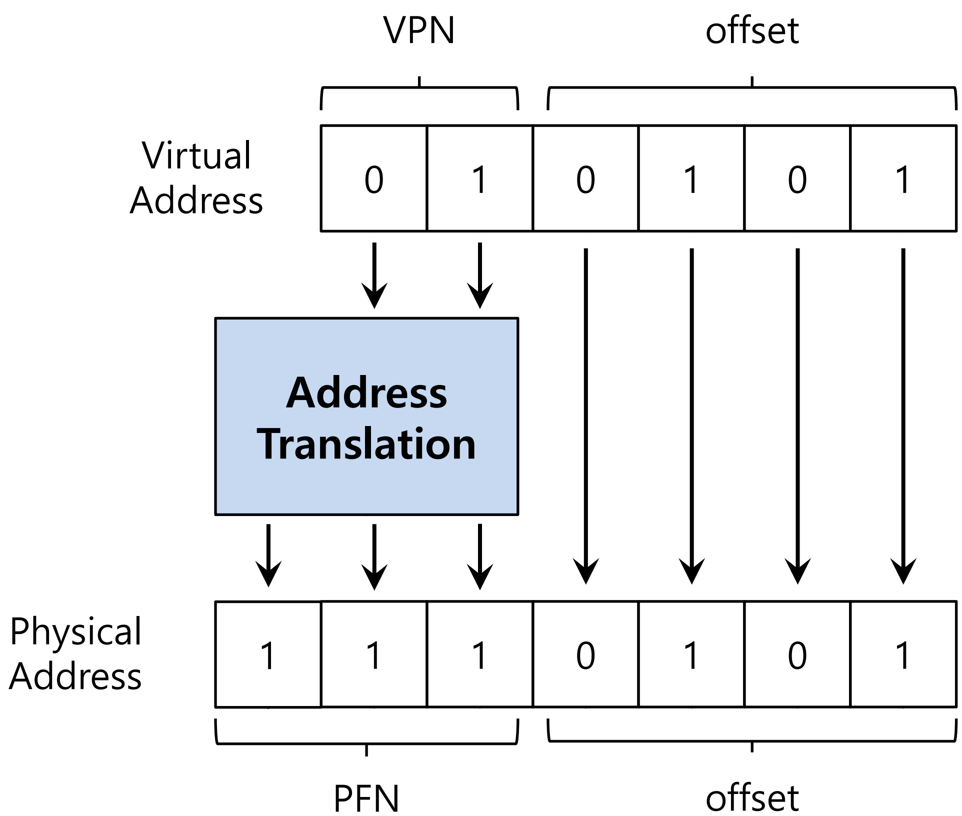


- Example: virtual address 21 in 64-byte address space



Example: Address Translation

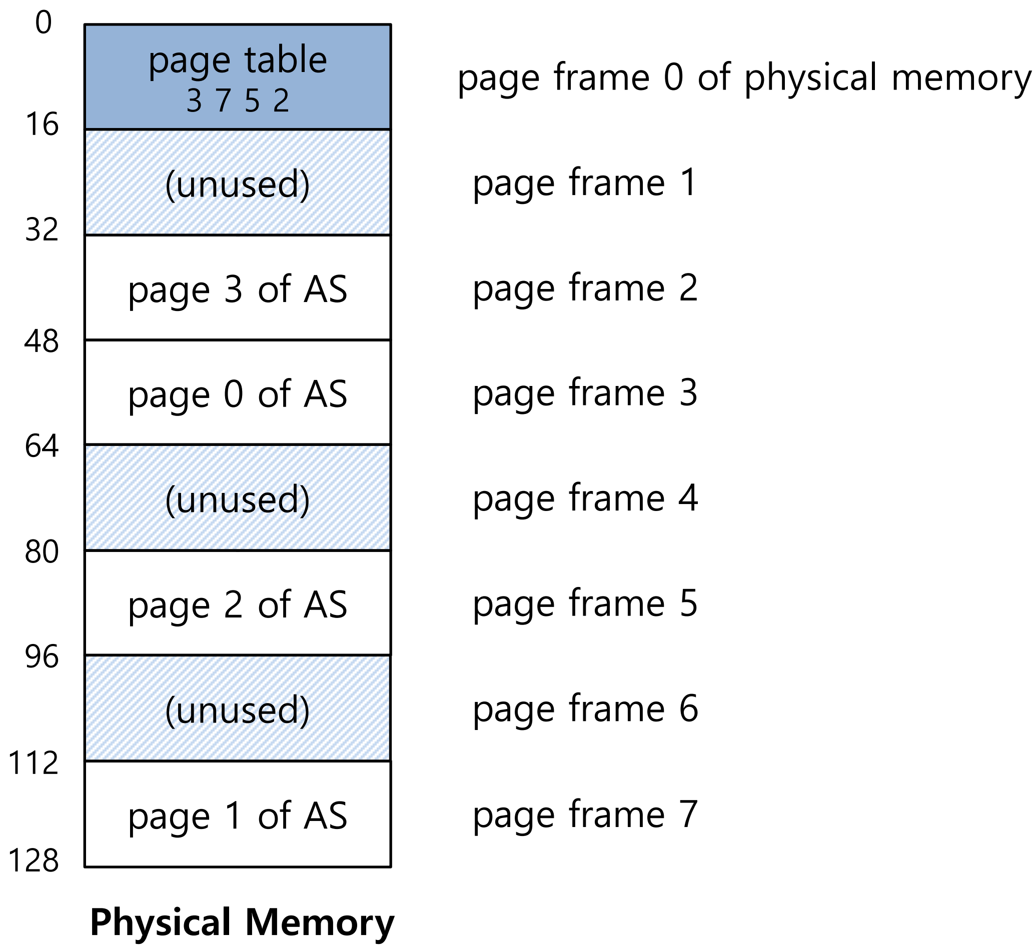
- The virtual address 21 in 64-byte address space



Where Are Page Tables Stored?

- Page tables can get awfully large
  - 32-bit address space with 4-KB pages, 20 bits for VPN
    - $4MB = 2^{20}\ entries \ * 4\ Bytes\ per\ page\ table\ entry$
- Page tables for peach process are stored in memory.

Example: Page Table in Kernel Physical Memory



What is in the page table?

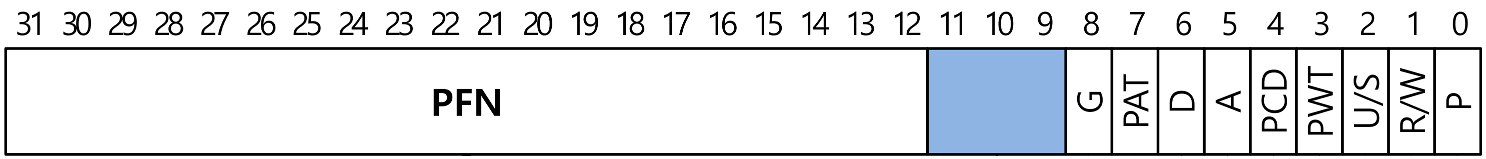
- The page table is just a **data structure** that is used to map the virtual address to physical address.
  - Simplest form: a linear page table, an array

- The OS **indexes** the array by VPN, and looks up the page-table entry.

Common Flags Of Page Table Entry

- **Valid Bit**: Indicating whether the particular translation is valid.
- **Protection Bit**: Indicating whether the page could be read from, written to, or executed from
- **Present Bit**: Indicating whether this page is in physical memory or on disk(swapped out)
- **Dirty Bit**: Indicating whether the page has been modified since it was brought into memory
- **Reference Bit(Accessed Bit):** Indicating that a page has been accessed

Example: x86 Page Table Entry



An x86 Page Table Entry(PTE)

- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

Paging: Too Slow

- To find a location of the desired PTE, the **starting location** of the page table is **needed**.

- For every memory reference, paging requires the OS to perform one **extra memory reference**.

Accessing Memory With Paging

```
// Extract the VPN from the virtual address
VPN = (VirtualAddress & VPN_MASK) >> SHIFT

// Form the address of the page-table entry (PTE)
PTEAddr = PTBR + (VPN * sizeof(PTE))

// Fetch the PTE
PTE = AccessMemory(PTEAddr)

// Check if process can access the page
if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
else
        // Access is OK: form physical address and fetch it
        offset = VirtualAddress & OFFSET_MASK
        PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
        Register = AccessMemory(PhysAddr)
```

Summary

- Segmentation
  - A program contains logically-different **segments**: code, stack, heap
  - Each segment can be **placed** in **different part of physical memory**.
    - **Base** and **bounds** exist **per each segment**.
- Paging
  - **physical memory** is **split** into some number of pages called a **page frame**.
  - **Page table** per process is used **to translate** the virtual address to physical address.
- Next: Free Space Management, Translation Lookaside Buffers, Advanced Page Tables, and Swapping (Chapters 17, 19, 20, 21, 22, 23)