香港城市大學
City University of Hong Kong

Tutorial 6: Paging

CS3103

Operating Systems

專業 創新 胸懷全球
Professional · Creative
For The World

Disclaimer: This slide was initially developed for Operating System course in Department of Computer Science at Hanyang University. This slide is for OSTEP book written by Remzi and Andrea at University of Wisconsin.

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.

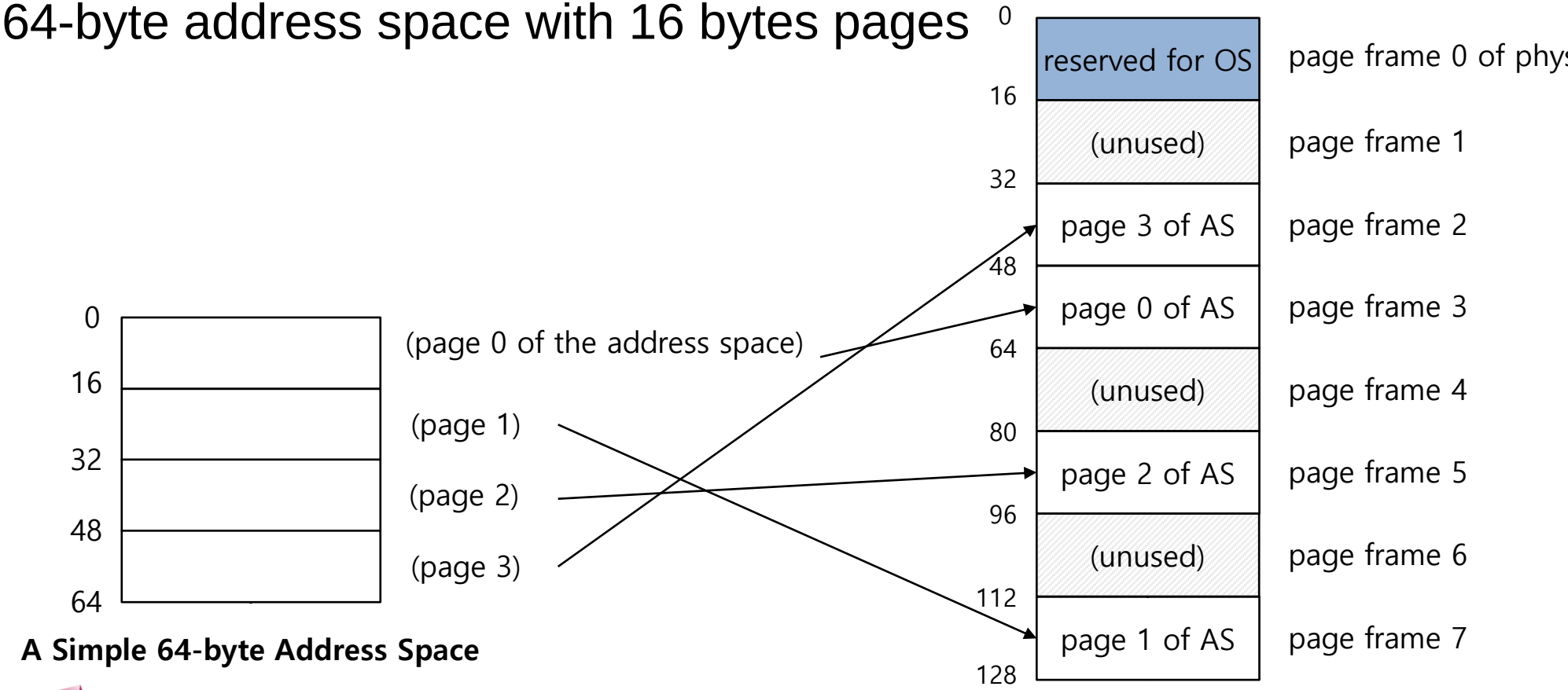
Advantages Of Paging

- **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

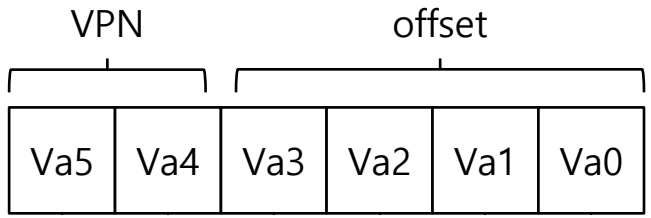
Example: A Simple Paging

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

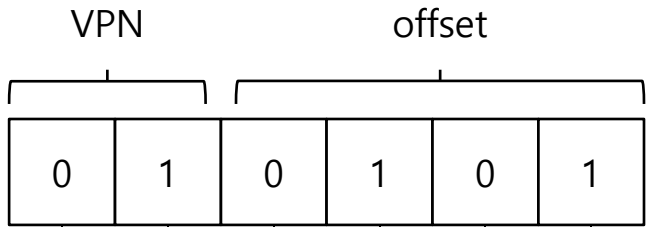


Address Translation

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



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

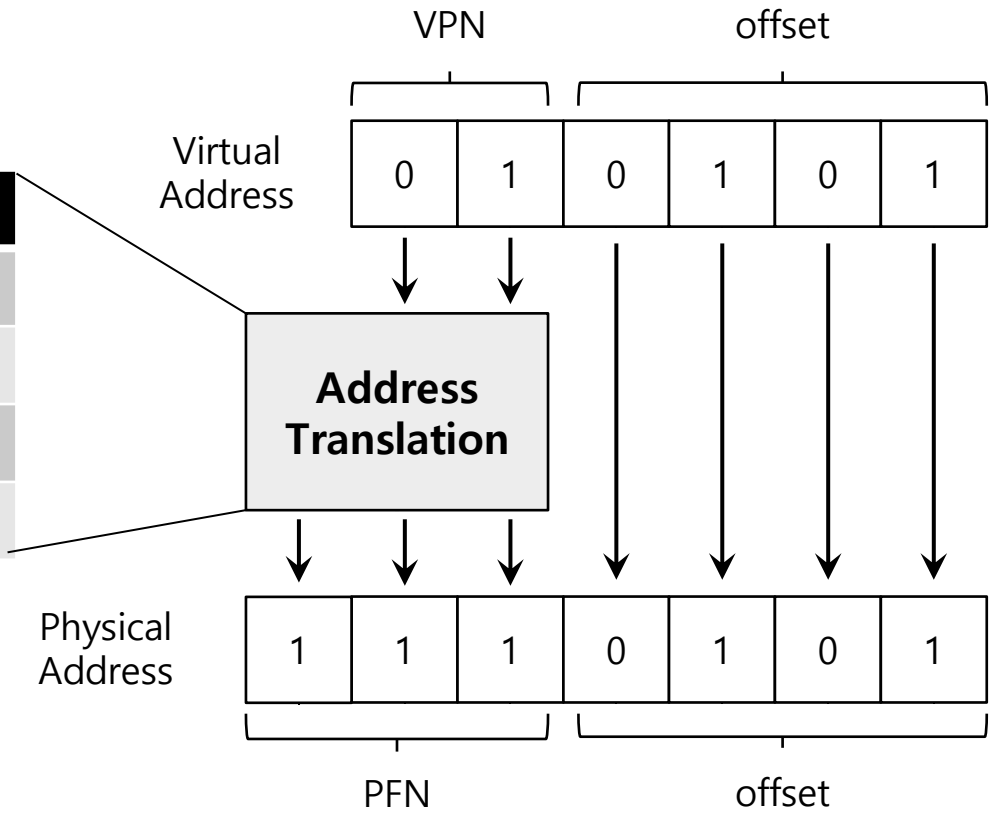


Example: Address Translation

- The virtual address 21 in 64-byte address space

The page table would have the following four entries:

Virtual Page No.	Physical Frame No.
VP 0	PF 3
VP 1	PF 7
VP 2	PF 5
VP 3	PF 2



What Is In The Page Table?

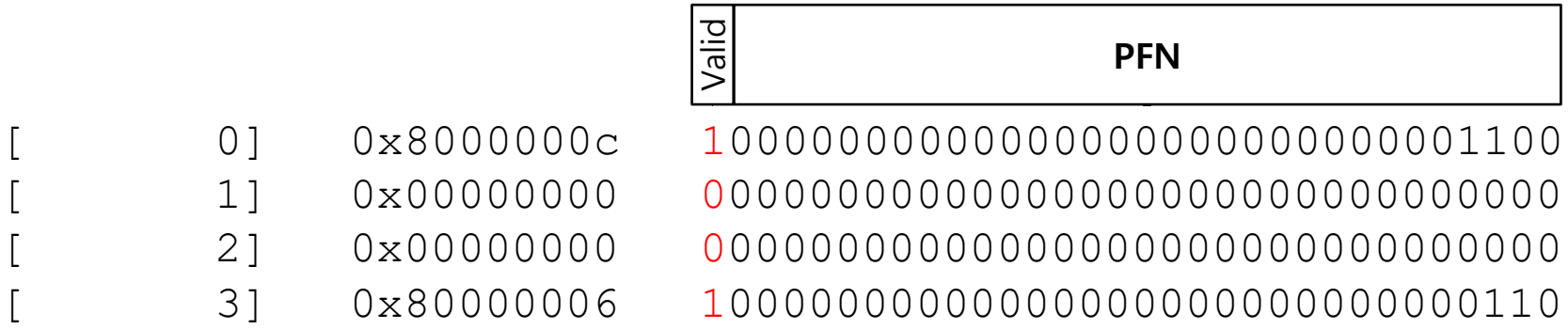
- The page table is just a **data structure** that is used to map the virtual address to physical address.
  - Simplest form: a linear page table, an array
- The OS **indexes** the array by VPN, and looks up the page-table entry.

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

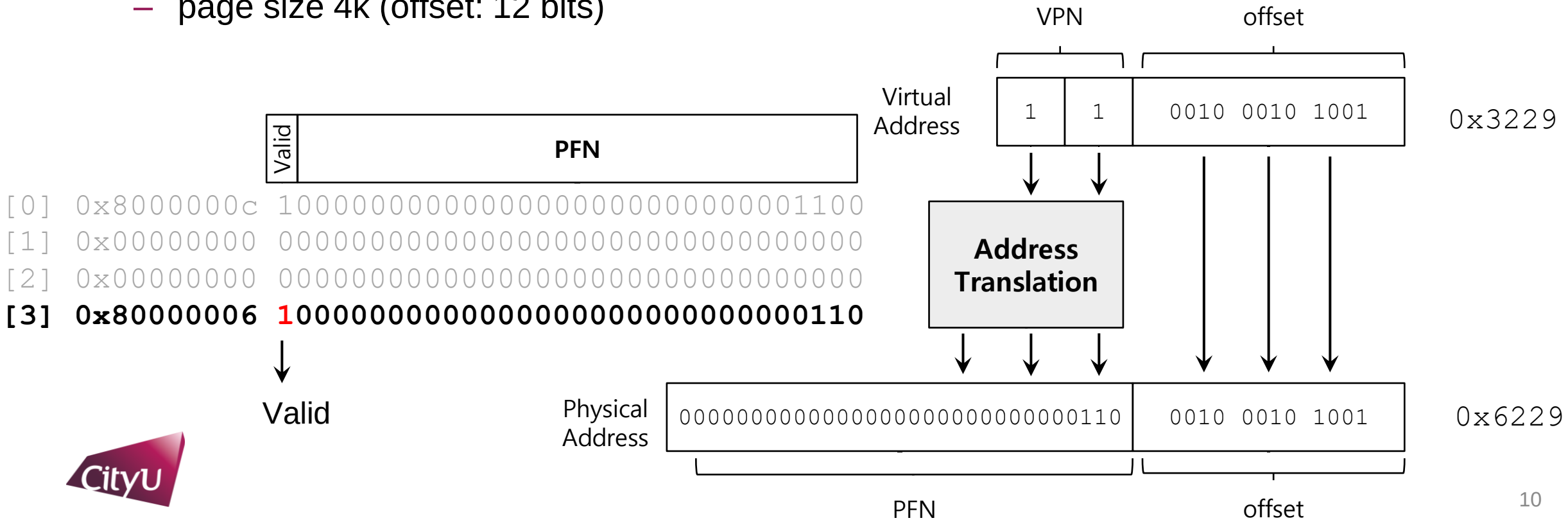
Example: `paging-linear-translate.py`

- The format of the page table is simple:
- The high-order (left-most) bit is the VALID bit.
- If the bit is 1, the rest of the entry is the PFN.
- If the bit is 0, the page is not valid.



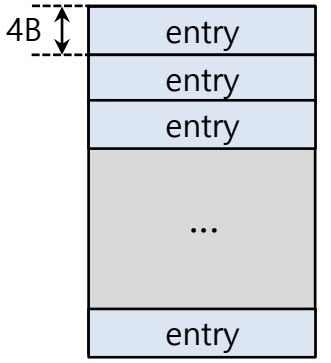
A Detailed Linear Page Table Example

- Some basic assumptions:
  - address space size 16k (VA: 14 bits)
  - physical memory size 64k
  - page size 4k (offset: 12 bits)
- Translate VA: 0x00003229:
  - Binary: 11 0010 0010 1001

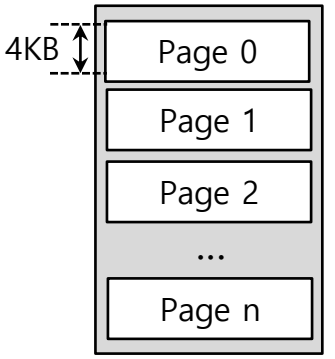


Paging: Linear Tables

- Page tables for each process are stored in memory.
- Page table are too big and thus consume too much memory.
  - 32-bit address space with 4-KB pages, 20 bits for VPN
    - $4MB = 2^{20}\ entries * 4\ Bytes\ per\ page\ table\ entry$
    - Now imagine there are 100 processes running: this means the OS would need 400MB of memory just for all those address translations!



Page Table of Process A

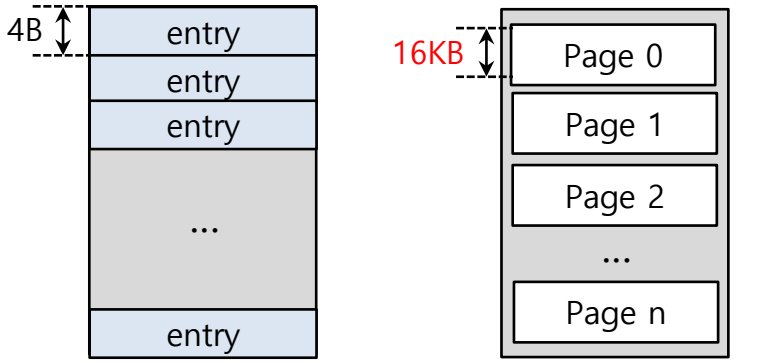


Physical Memory

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

Paging: Bigger Pages

- We could reduce the size of the page table in one simple way: use bigger pages.
  - Assume that 32-bit address space with 16KB pages and 4-byte page-table entry.



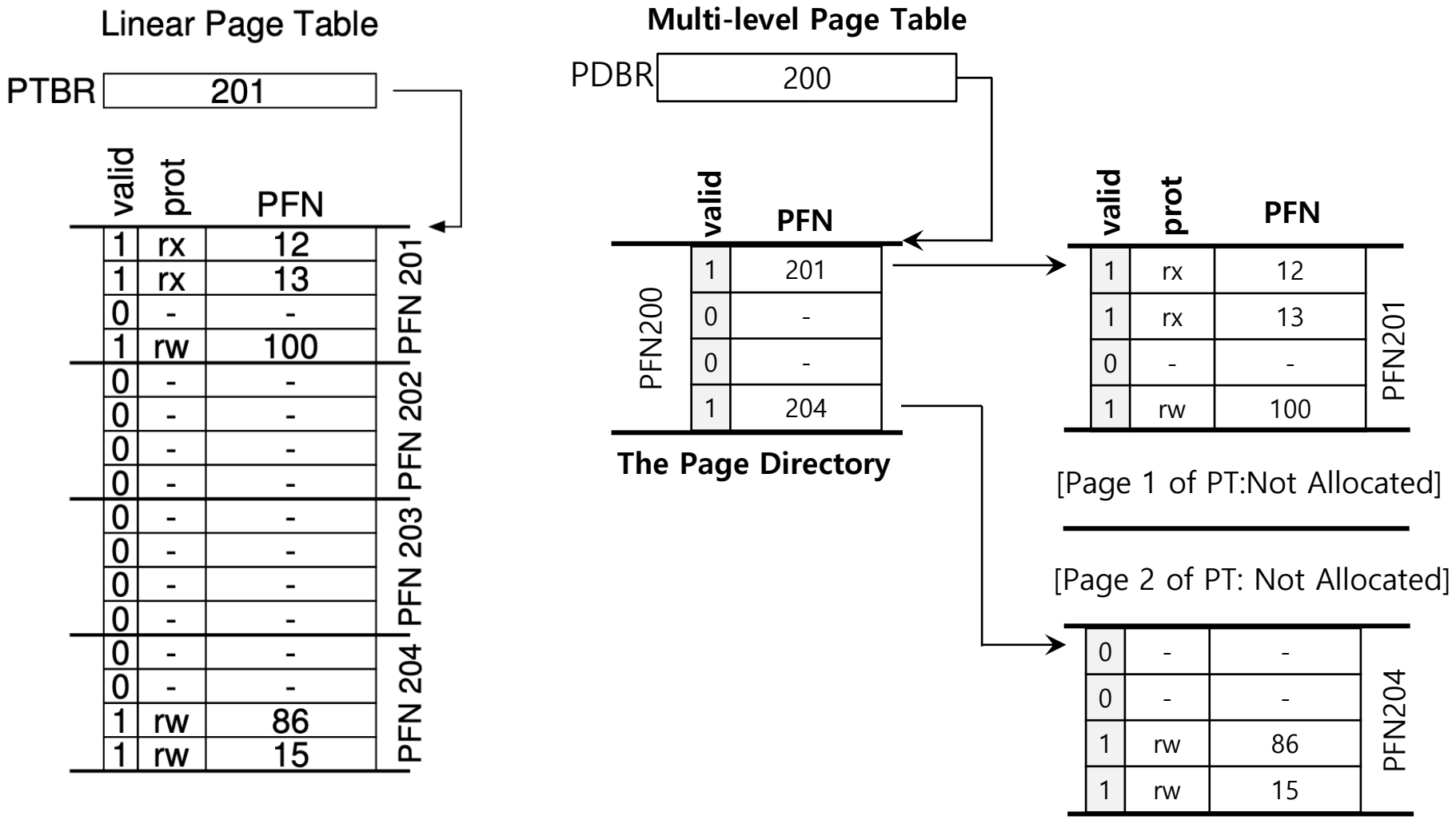
$\frac{2^{32}}{2^{14}} * 4 = 1MB$ **per page table**

Big pages lead to internal fragmentation.

Paging: Multi-level Page Tables

- In order to reduce the memory overhead of 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



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

PTBR: page table base register, PDBR: page directory base register.

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 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

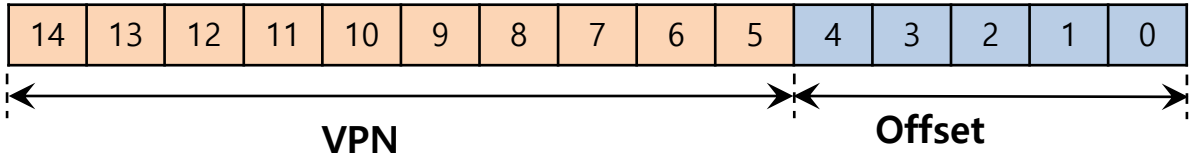
- A multi-level structure can adjust level of indirection through use of the page directory.
  - Indirection place page-table pages wherever we would like in physical memory.

A Detailed Multi-Level Page Table Example

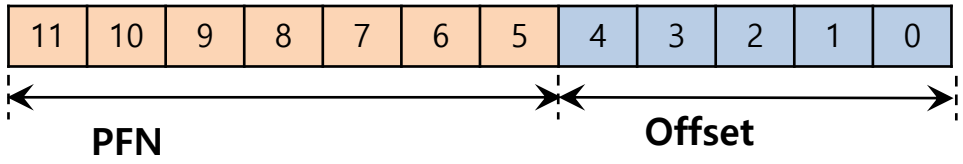
- To understand the idea behind multi-level page tables better, let's do an example.
- Some basic assumptions:

Flag	Detail
Address space	32 KB (1024 pages)
Page size	32 bytes
Virtual address	15 bits
VPN	10 bits
Offset	5 bits
Page entry per page	32 PTEs $\longrightarrow \log_2 32 = 5$
Physical memory size	4 KB (128 pages)

15-bits Virtual Address

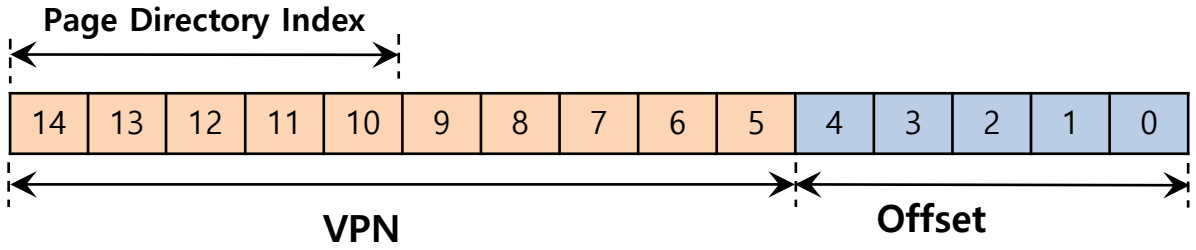


12-bits Physical Address



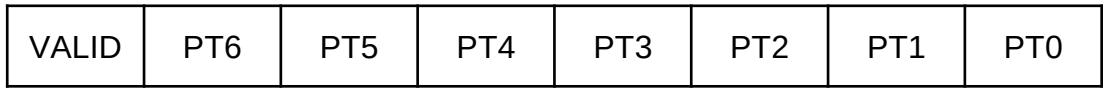
A Detailed Multi-Level Page Table Example: Page Directory Index

- The page directory needs one entry per page of the page table
  - it has 32 page-directory entries (PDEs).
- The page-directory entry is invalid → Raise an exception (The access is invalid)



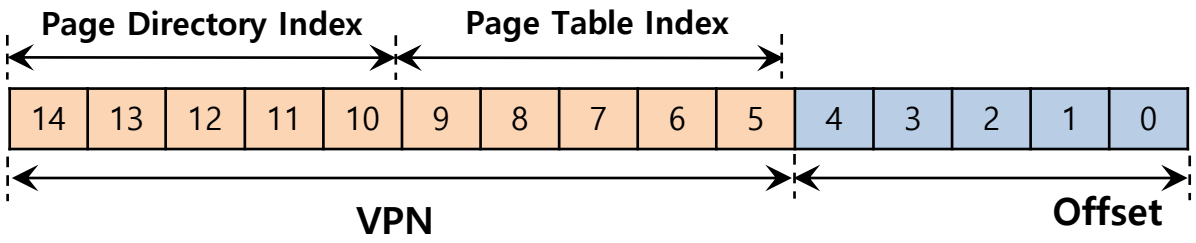
15-bits Virtual Address

- The format of a PDE is (8 bits or 1 byte):
  - `VALID | PT6 ... PT0`



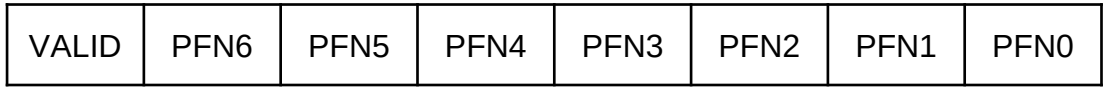
A Detailed Multi-Level Page Table Example: Page Table Index

- 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.



15-bits Virtual Address

- The format of a PTE is (8 bits or 1 byte):
  - `VALID | PFN6 ... PFN0`



A Detailed Multi-Level Page Table Example: Address Translation

- Page directory base register (PDBR) : This means the page directory is held in this page, e.g., 108 (decimal).
- A memory page dump:
  - shows the 32 bytes found on pages 0, 1, 2, and so forth. The first byte (0th byte) on page 0 has the value 0x1b, the second is 0x1d, the third 0x05, and so forth.

```
          00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31
--------: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
page   0: 1b 1d 05 05 1d 0b 19 00 1e 00 12 1c 19 09 19 0c 0f 0b 0a 12 18 15 17 00 10 0a 06 1c 06 05 05 14
page   1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   2: 12 1b 0c 06 00 1e 04 13 0f 0b 10 02 1e 0f 00 0c 17 09 17 17 07 1e 00 1a 0f 04 08 12 08 19 06 0b
...
```

A Detailed Multi-Level Page Table Example: Address Translation

- Run `./paging-multilevel-translate.py`
  - It gives the value of the PDBR (108), a complete dump of each page of memory, and a list of virtual addresses to translate (611c, 3da8, etc).
- Steps:
  - Use the PDBR to find the relevant page table entries for this virtual page.
  - Then find if it is valid. If so, use the translation to form a final physical address. Using this address, you can find the VALUE that the memory reference is looking for. Note that the virtual address may not be valid and thus generate a fault.

CityU

A Detailed Multi-Level Page Table Example: Address Translation

- Let's try to translate virtual address 0x611c to physical address and fetch its value if it is a valid address.

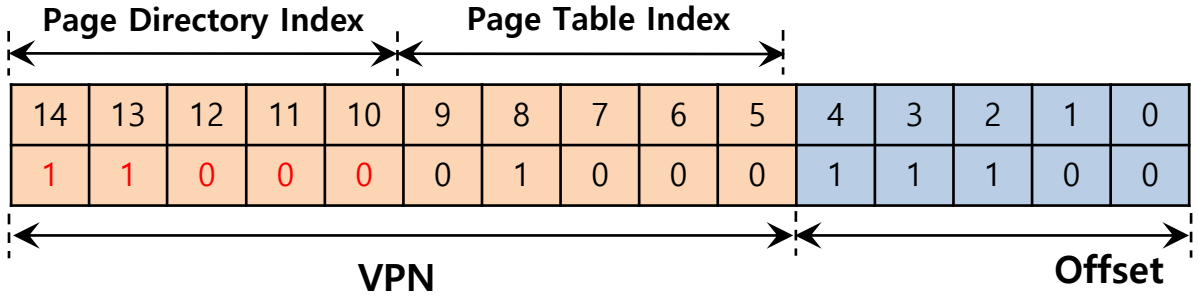
```
            00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31
--------: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
page   0: 1b 1d 05 05 1d 0b 19 00 1e 00 12 1c 19 09 19 0c 0f 0b 0a 12 18 15 17 00 10 0a 06 1c 06 05 05 14
page   1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   2: 12 1b 0c 06 00 1e 04 13 0f 0b 10 02 1e 0f 00 0c 17 09 17 17 07 1e 00 1a 0f 04 08 12 08 19 06 0b
....
page  33: 7f 7f 7f 7f 7f 7f 7f 7f b5 7f 9d 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f6 b1 7f 7f 7f 7f
....
page  53: 0f 0c 18 09 0e 12 1c 0f 08 17 13 07 1c 1e 19 1b 09 16 1b 15 0e 03 0d 12 1c 1d 0e 1a 08 18 11 00
....
page 108: 83 fe e0 da 7f d4 7f eb be 9e d5 ad e4 ac 90 d6 92 d8 c1 f8 9f e1 ed e9 a1 e8 c7 c2 a9 d1 db ff
....
page 127: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f df 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 95 7f 7f
```

A Detailed Multi-Level Page Table Example: Address Translation

- Step 1: Use the PDBR to find the page directory on page 108.

```
            00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31
--------: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
page 108: 83 fe e0 da 7f d4 7f eb be 9e d5 ad e4 ac 90 d6 92 d8 c1 f8 9f e1 ed e9 a1 e8 c7 c2 a9 d1 db ff
```

- Step 2: Convert virtual address 611c to binary (15 bits): 110000100011100.
- Step 3: PDE
  - PDE index: 0b11000 = 0x18 = 24
  - PDE content: a1, 24th byte on page 108
    - 0xa1= 0b10100001
    - Valid bit: 1, PT (PFN): 0b0100001=0x21=33



VALID	PT6	PT5	PT4	PT3	PT2	PT1	PT0
1	0	1	0	0	0	0	1

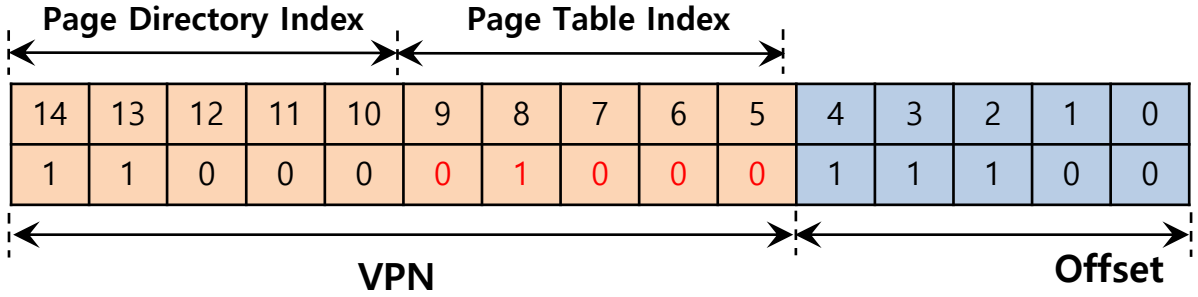
Note: Numbers are in base 10 representation by default. To distinguish the different number systems, suffixes will be used, 0b for binary numbers and 0x for hexadecimal numbers. We sometimes prefix decimal numbers with "0d" (zero d).

A Detailed Multi-Level Page Table Example: Address Translation

- Step 4: Locate memory page 33.

```
            00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31
--------: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
page  33: 7f 7f 7f 7f 7f 7f 7f 7f b5 7f 9d 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f6 b1 7f 7f 7f 7f
```

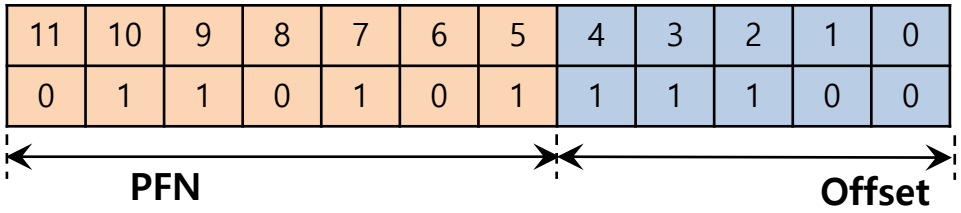
- Step 5: PTE
  - PTE index: 0b01000 = 0x8 = 8
  - PTE content: b5, 8th byte on page 33
    - 0xb5 = 0b10110101
    - Valid bit: 1, PFN: 0b0110101 = 0x35 = 53



VALID	PFN6	PFN5	PFN4	PFN3	PFN2	PFN1	PFN0
1	0	1	1	0	1	0	1

A Detailed Multi-Level Page Table Example: Address Translation

- Step 6: Physical Address
  - PFN: 0b0110101 = 53, Offset 0b11100 = 28
  - Physical Address: 0b011010111100 = 0x6bc



- Step 7: Locate memory page 53.

```
          00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31
--------: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
page  53: 0f 0c 18 09 0e 12 1c 0f 08 17 13 07 1c 1e 19 1b 09 16 1b 15 0e 03 0d 12 1c 1d 0e 1a 08 18 11 00
```

- Step 8: Load Data
  - Page offset: 0b11100 = 0x1c = 28
  - Value: 08, 28th byte on page 53
    - 0x08 = 0b00001000 = 8

A Detailed Multi-Level Page Table Example: Address Translation

- Summary:
- Virtual Address 0x611c:
  - pde index:0x18 [decimal 24] pde contents:0xa1 (valid 1, pfn 0x21 [decimal 33])
  - pte index:0x8 [decimal 8] pte contents:0xb5 (valid 1, pfn 0x35 [decimal 53])
  - Translates to Physical Address 0x6bc --> Value: 08
- Note that the virtual address may not be valid and thus generate a fault.
  - For example: when translating virtual address 0x3da8, the page table entry is not valid.

CityU

Reference: Decimal, Binary, Hex

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F