18. Paging: Introduction

Operating System: Three Easy Pieces

Concept of Paging

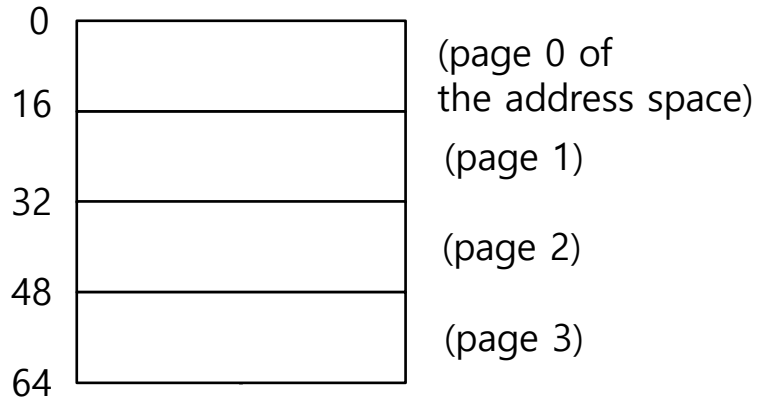- Paging **splits up** address space into **fixed-zed** unit called a **page**.

  - ◆ 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.
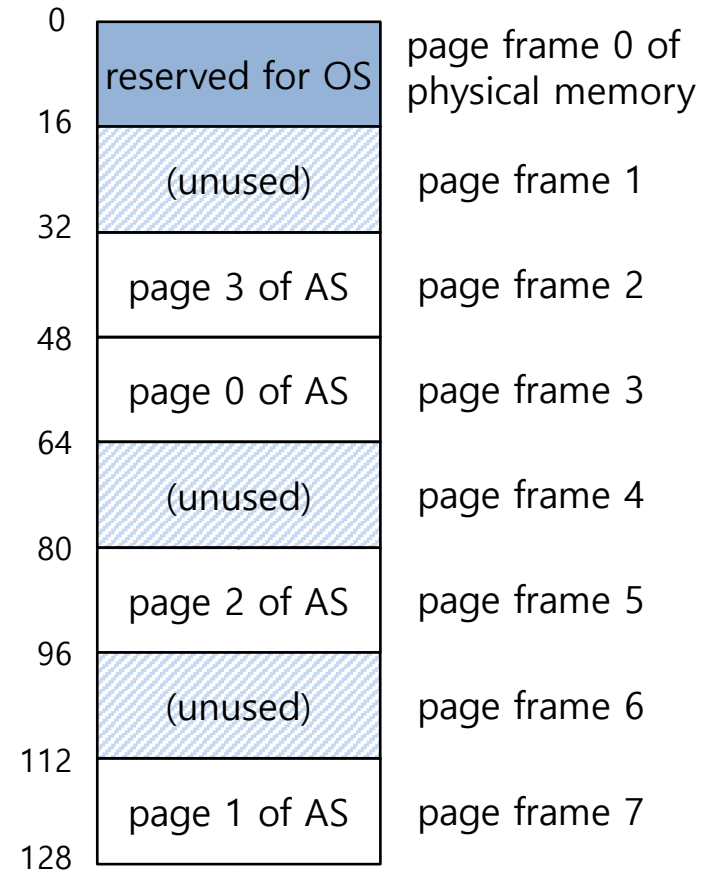
Advantages Of Paging

- **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
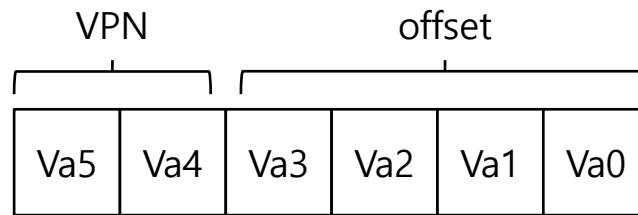
- 64-byte address space with 16 bytes pages

Address Space		Physical Memory	
0		0	
	(page 0 of the address space)	reserved for OS	page frame 0 of physical memory
16		16	
	(page 1)	(unused)	page frame 1
32		32	
	(page 2)	page 3 of AS	page frame 2
48		48	
	(page 3)	page 0 of AS	page frame 3
64		64	
		(unused)	page frame 4
		80	
		page 2 of AS	page frame 5
		96	
		(unused)	page frame 6
		112	
		page 1 of AS	page frame 7
		128	

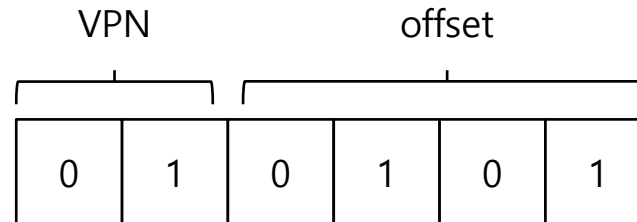**A Simple 64-byte Address Space**

64-Byte Address Space Placed In Physical Memory

Address Translation

- Two components in the virtual address

  - VPN: virtual page number
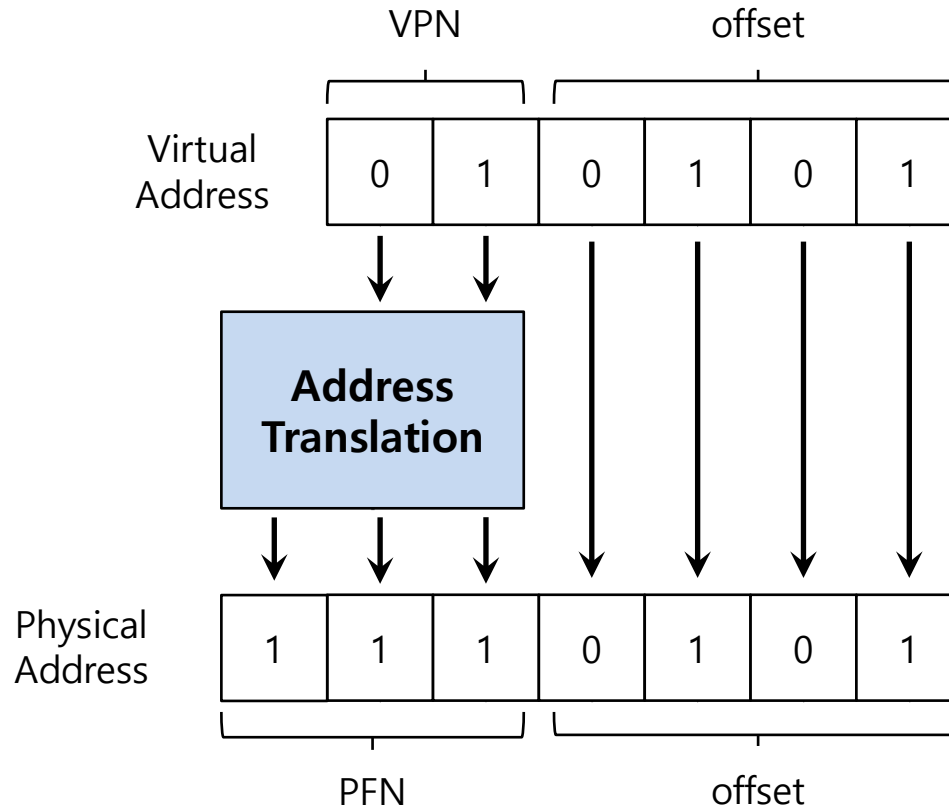
  - Offset: offset within the page

VPN		offset			
Va5	Va4	Va3	Va2	Va1	Va0

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

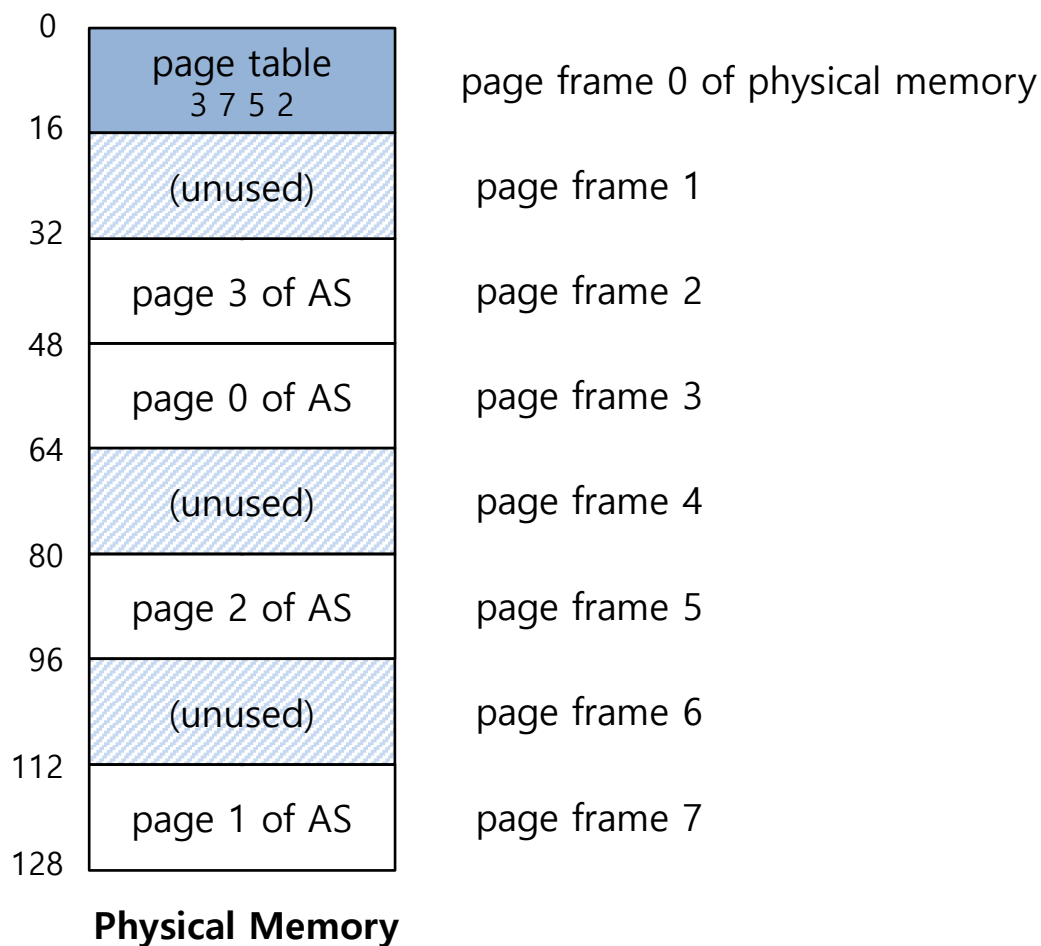| VPN		offset			
0	1	0	1	0	1

- 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.

0	
page table 3 7 5 2	page frame 0 of physical memory
16	
(unused)	page frame 1
32	
page 3 of AS	page frame 2
48	
page 0 of AS	page frame 3
64	
(unused)	page frame 4
80	
page 2 of AS	page frame 5
96	
(unused)	page frame 6
112	
page 1 of AS	page frame 7
128	

Physical Memory
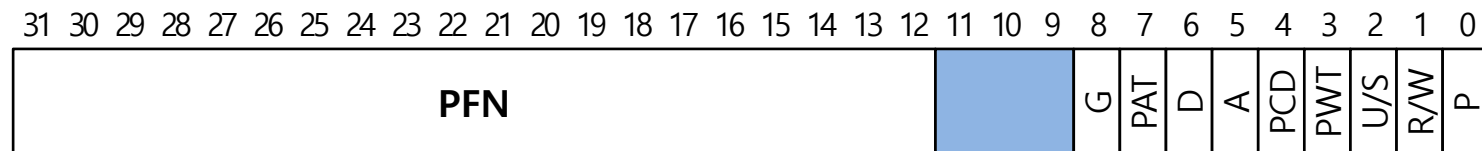
□ The page table is just a **data structure** that is used to map the virtual address to physical address.

  ◆ 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

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12	11 10 9	8	7	6	5	4	3	2	1	0
PFN		G	PAT	D	A	PCD	PWT	U/S	R/W	P

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

- 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**.

```
1        // Extract the VPN from the virtual address
2        VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4        // Form the address of the page-table entry (PTE)
5        PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7        // Fetch the PTE
8        PTE = AccessMemory(PTEAddr)
9
10       // Check if process can access the page
11       if (PTE.Valid == False)
12               RaiseException(SEGMENTATION_FAULT)
13       else if (CanAccess(PTE.ProtectBits) == False)
14               RaiseException(PROTECTION_FAULT)
15       else
16               // Access is OK: form physical address and fetch it
17               offset = VirtualAddress & OFFSET_MASK
18               PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19               Register = AccessMemory(PhysAddr)
```

A Memory Trace

- Example: A Simple Memory Access

```c
int array[1000];
...
for (i = 0; i < 1000; i++)
        array[i] = 0;
```
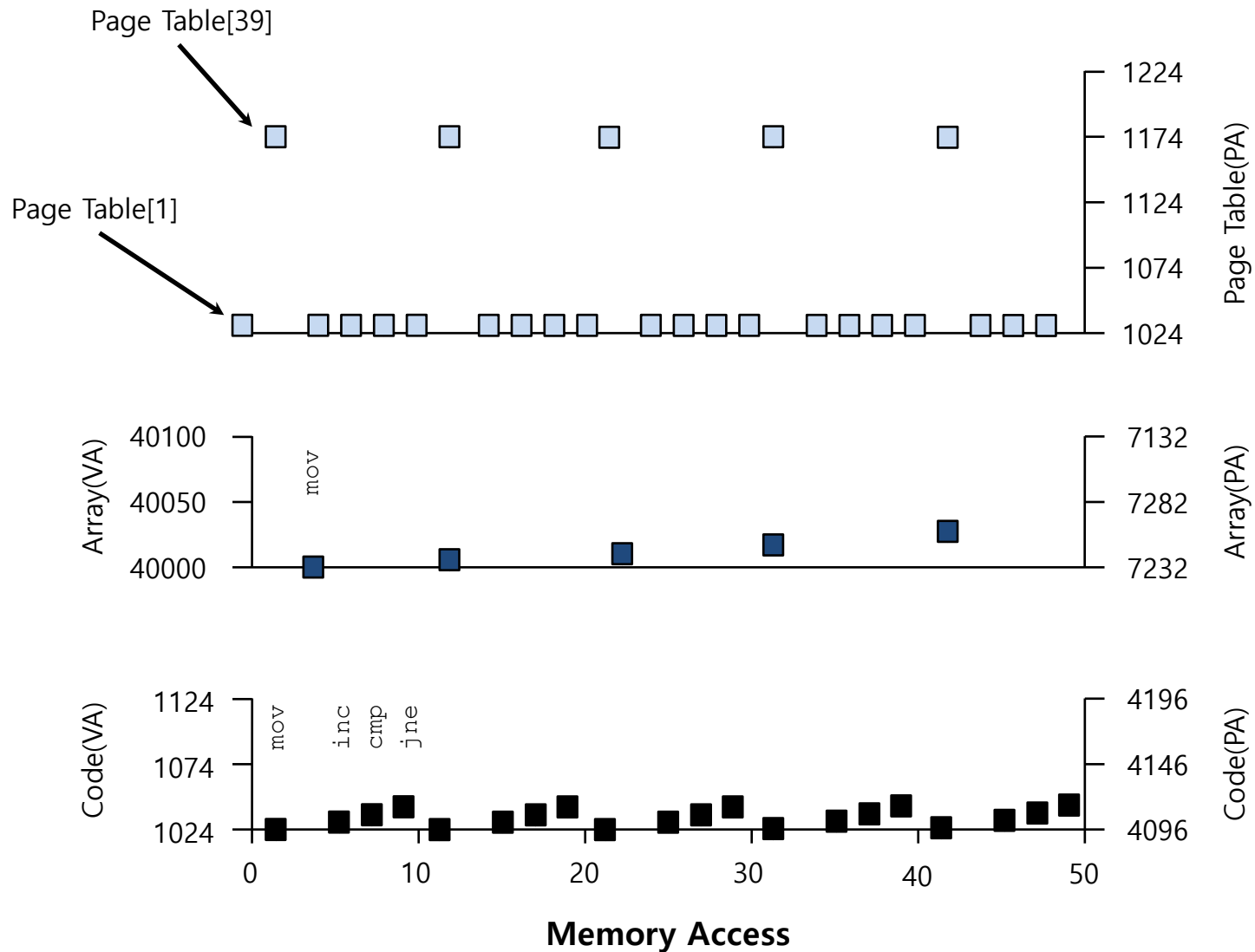
- Compile and execute

```
prompt> gcc –o array array.c –Wall –o
prompt>./array
```

- Resulting Assembly code

```
0x1024 movl $0x0,(%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
```

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