18. Paging: Introduction

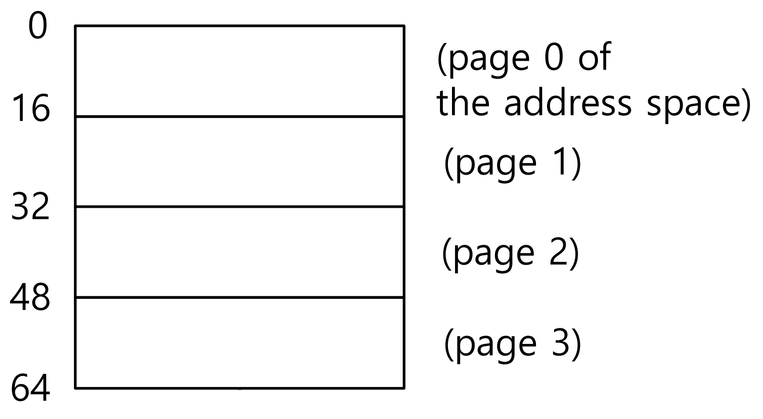
Operating System: Three Easy Pieces

Concept of Paging

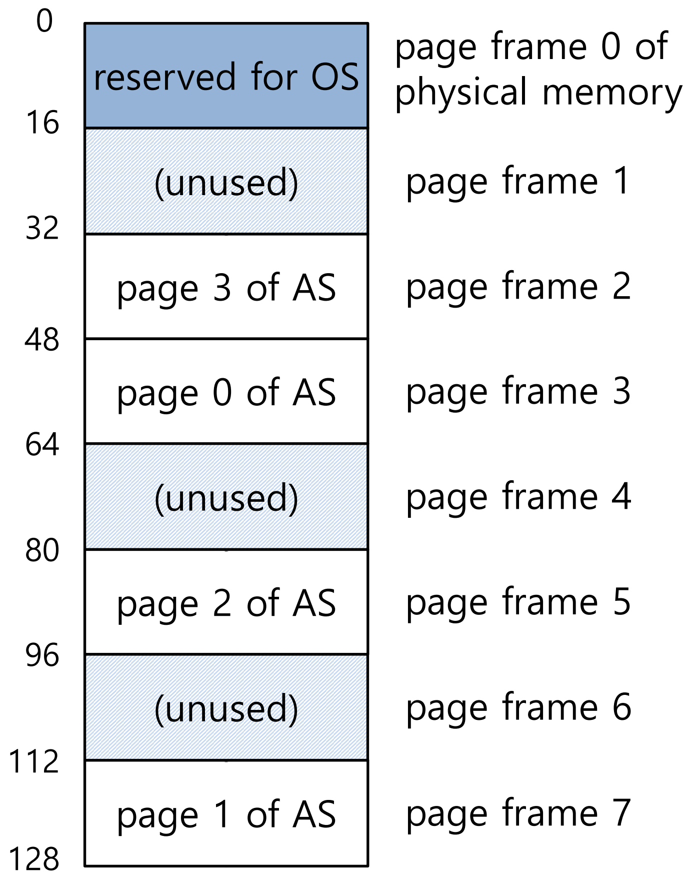
- Paging **splits up** address space into **fixed-zed** unit called a **page**.
  - Segmentation: variable size of logical segments(code, stack, heap, etc.)
- With paging, **physical memory** is also **split** into some number of pages called a **page frame**.
- **Page table** per process is needed **to translate** the virtual address to physical address.

Toy Example: A Simple Paging

- 128-byte physical memory with 16 bytes page frames
- 64-byte address space with 16 bytes pages, 1 byte addressable



A Simple 64-byte Address Space



64-Byte Address Space Placed In Physical Memory

Advantages Of Paging

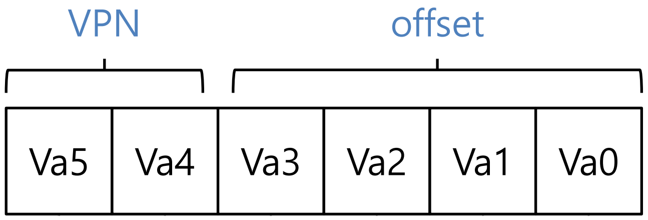
- **Flexibility:** Supporting the abstraction of address space effectively
  - Don't need assumption how heap and stack grow and are used.

- **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

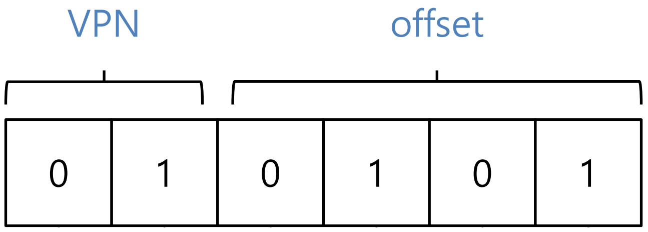
- *Con: the semantic meaning of the content is lost…*

Address Translation

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



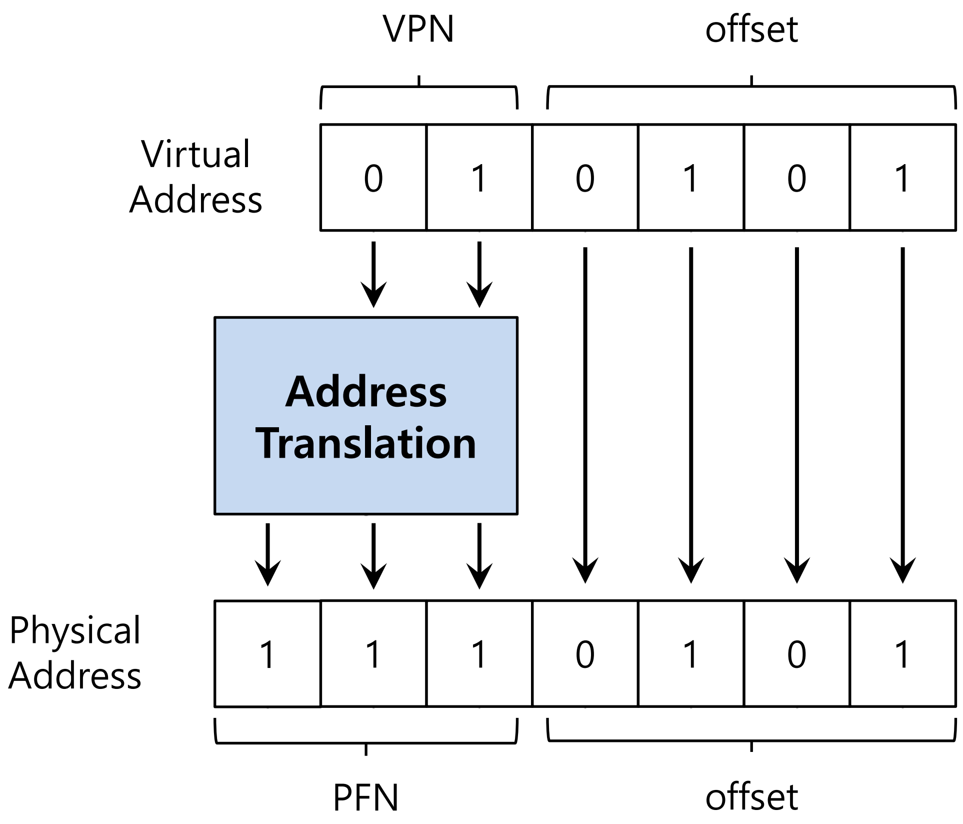
- Example: virtual address 21 in 64-byte address space



```
movl 21, %eax
```

Example: Address Translation

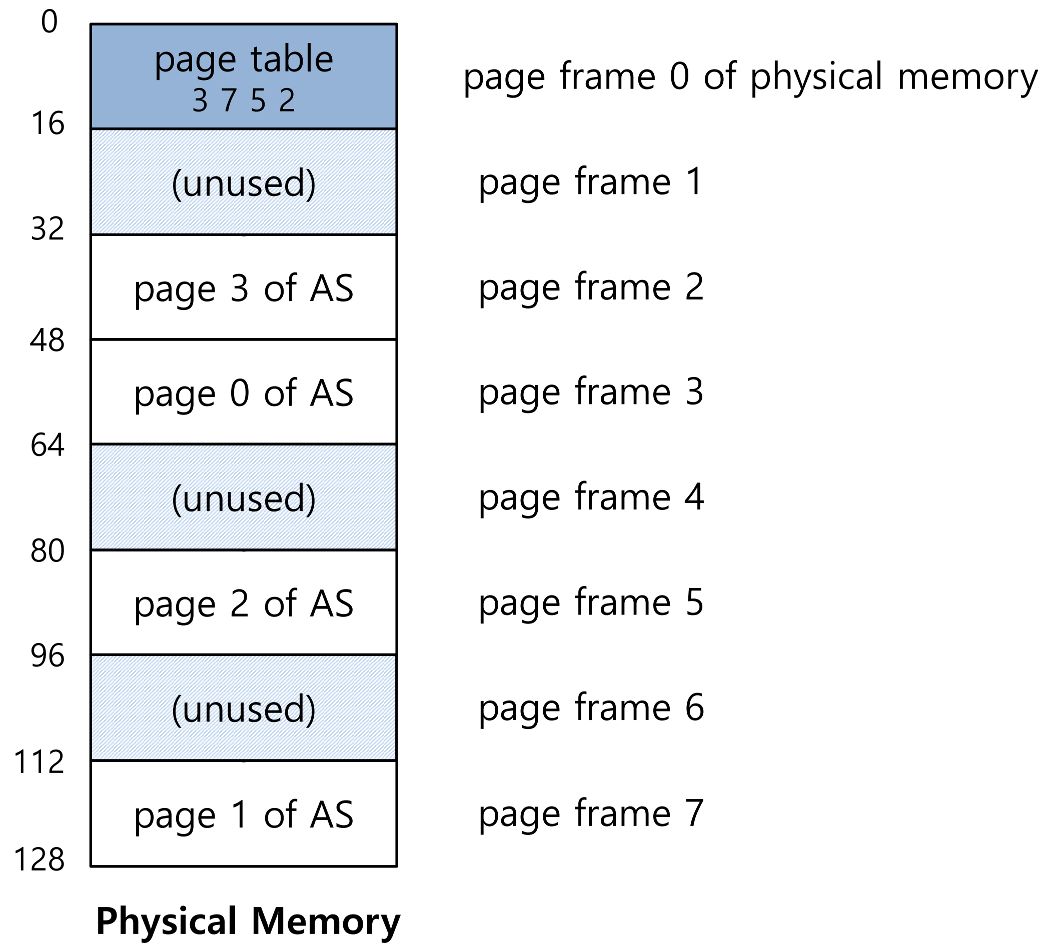
- The virtual address 21 in 64-byte address space



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 each process** are stored in memory.

Example: Page Table in Kernel Physical Memory



What Is In The Page Table?

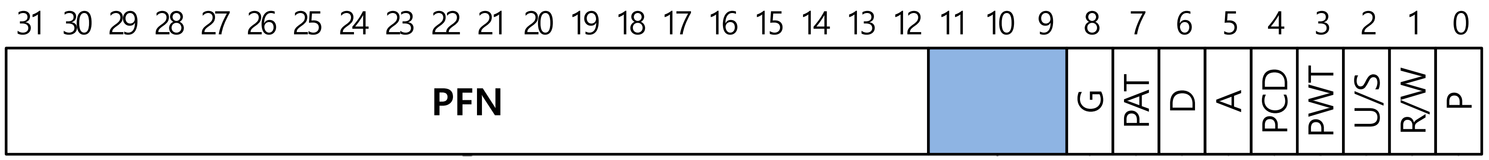
- The page table is just a **data structure** that is used to map the virtual address to physical address.
  - Simplest form: a linear page table, an array

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

Common Flags Of Page Table Entry (PTE)

- **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**.
  - Page tables are too big to be stored in MMU

- 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 SHIFT=4  VPN_MASK=0x30
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
// Form the address of the page-table entry (PTE)
PTEAddr = PTBR + (VPN * sizeof(PTE))

// Fetch the PTE ~PTBR[VPN]
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)
```

A Memory Trace

- Example: A Simple Memory Access

```
int array[1000];
...
for (i = 0; i < 1000; i++)
        array[i] = 0;
```

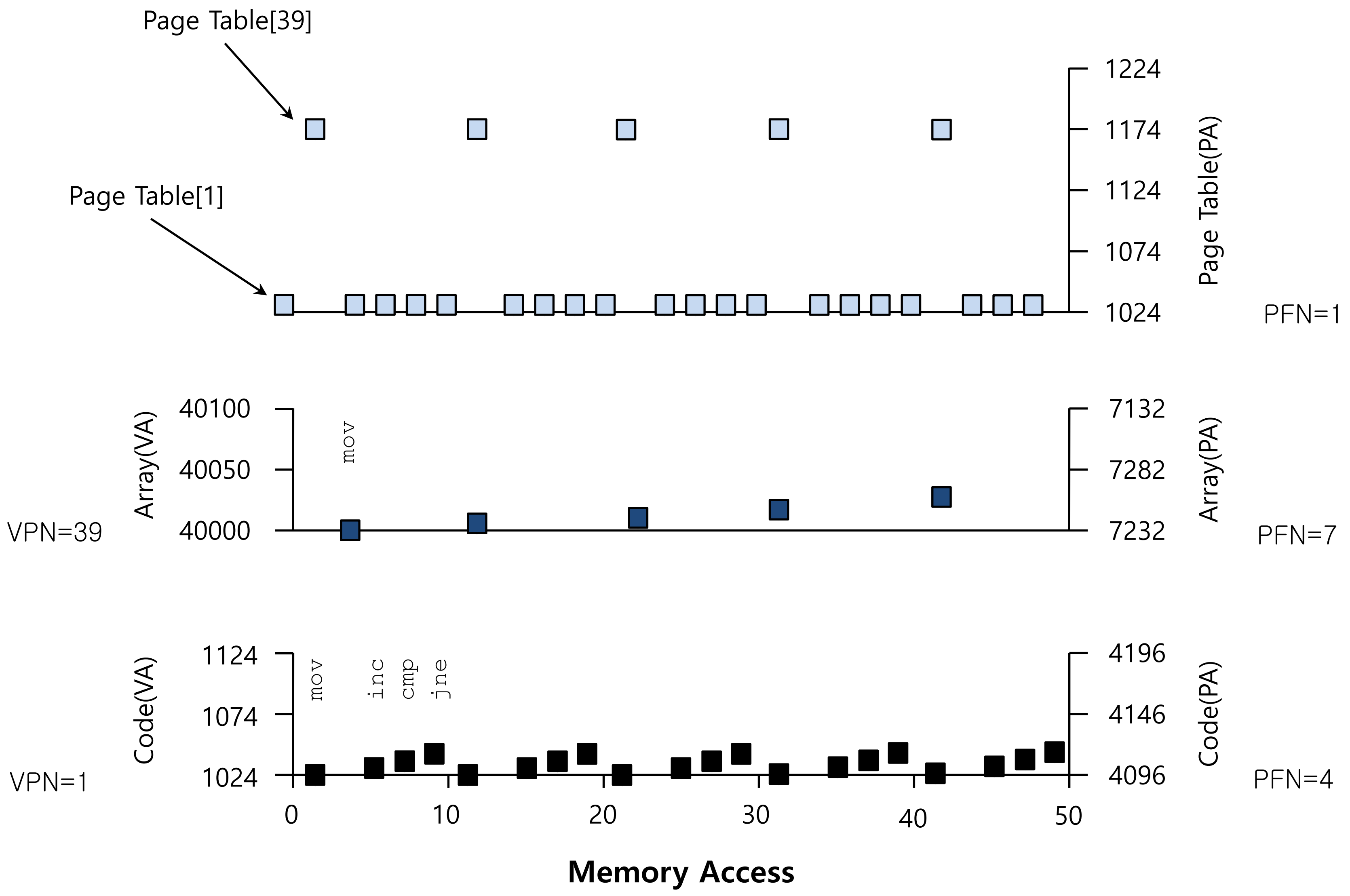
- Compile and execute

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

- Resulting Assembly code

```
0x1024 movl $0x0,(%edi,%eax,4)  #Power of CISC! edi+eax*4
0x1028 incl %eax                #Increase counter
0x102c cmpl $0x03e8,%eax        #Check if last element
0x1030 jne 0x1024               #Implicit (eflags) Zero bit access
```

A Virtual(And Physical) Memory Trace: 64KB VAS, 1KB pages



- Disclaimer: Disclaimer: This lecture slide set is used in AOS course at University of Cantabria. 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 Arpaci-Dusseau (at University of Wisconsin)