Operating System (OS)
CS232

Memory Management: Paging

Dr. Muhammad Mobeen Movania

Outlines

- Issues with Segmentation
- Paging with example
- Paging Advantages
- Address Translation with example
- Where are page tables stored?
- Is paging slow?
- Summary

Issues with Segmentation

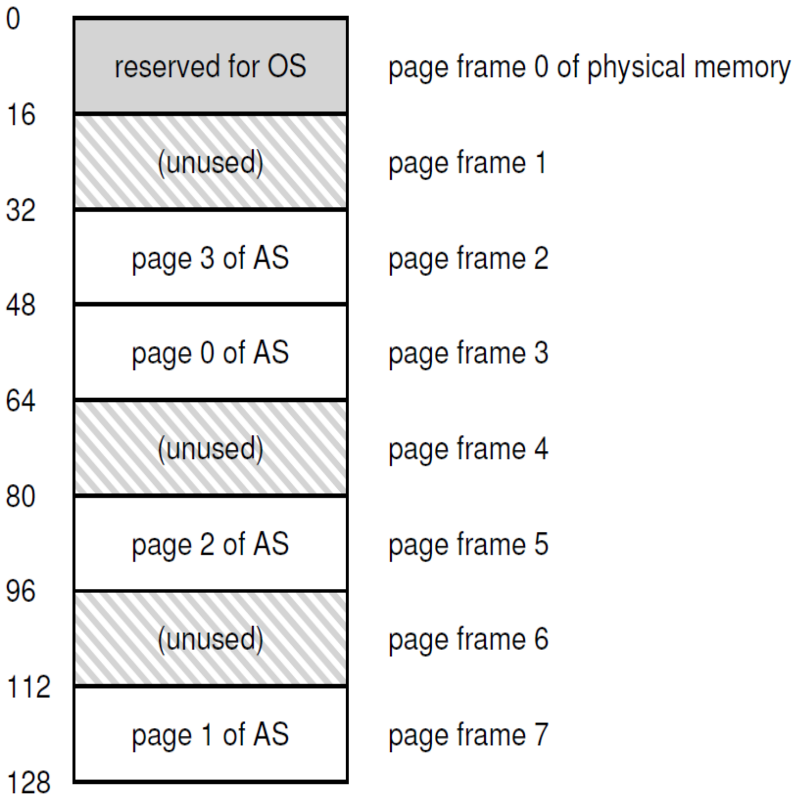
- Segmentation though useful in saving memory but it suffers from external fragmentation
- Why?
  - Because segments have unequal size, some may fit the free space well others might not
- Solution (Paging)
  - Divide the address space into equal sized regions

Paging

- Divide virtual address into fix-size units called ***pages***
- Physical memory is viewed as an array of fix-sized slots called ***page frames***
- Each page frame (physical) contains a page (virtual)

Example

- A 64-byte address space
- Divided into pages of 16 bytes each
- Virtual address = page_no. + offset
- Pages can be placed anywhere in physical memory.

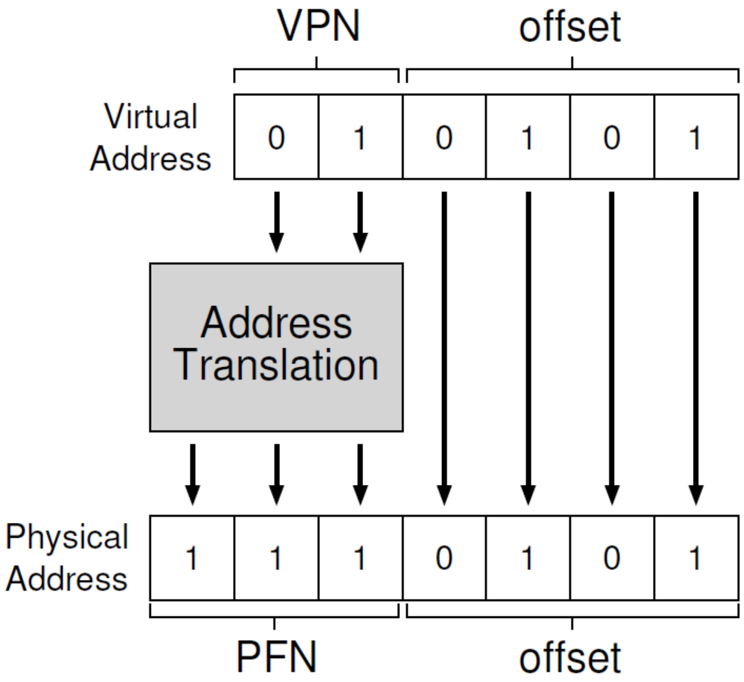


Paging - Advantages

- Flexibility
  - System can support abstraction of an address space effectively
- Simplicity of free-space management
  - Page frame size == page size, OS simply keeps information of free page frames in free list

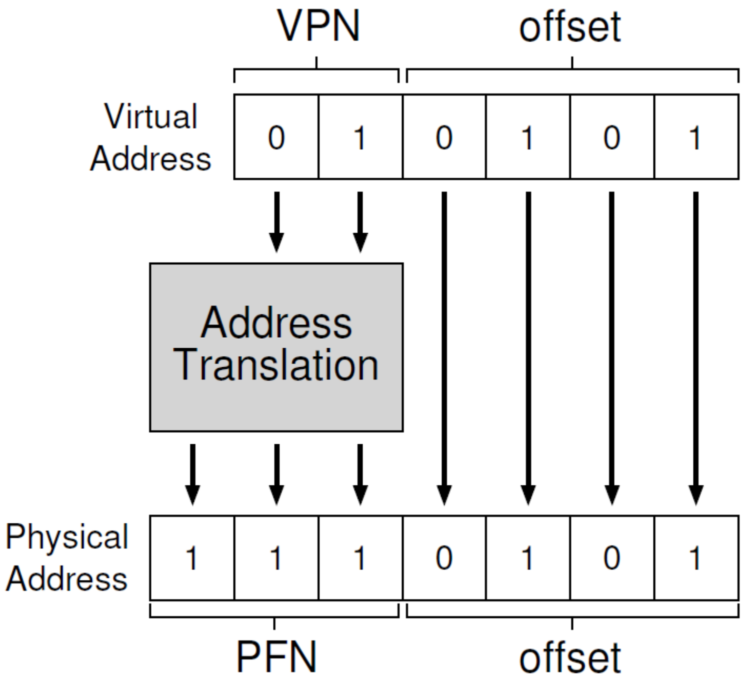
Address Translation

- The OS needs to store *page tables*
- Page tables store address translations for virtual pages
- Page tables are stored *per-process*



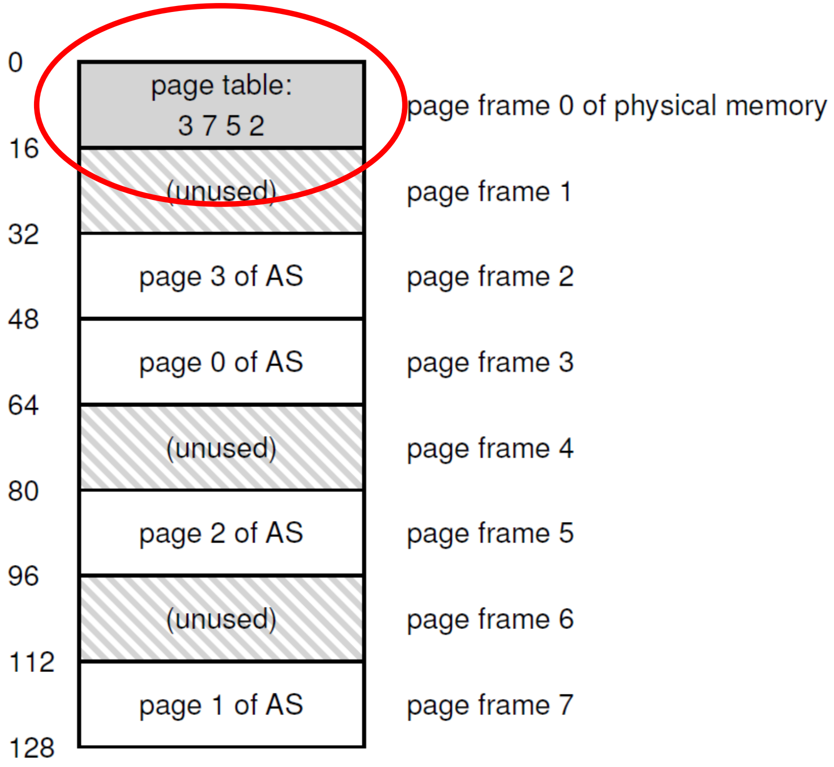
Example

- VA: 21=010101
- VPN: Virtual page no.
- PFN: physical frame no.
- VPN is used as index into page table to get PFN
- Note
  - offset remains the same as it tells us which byte within the page we want



Where are page tables stored?

- Page tables are stored per-process in memory (could be in OS managed physical memory or in OS virtual memory)



What’s stored in a page table?

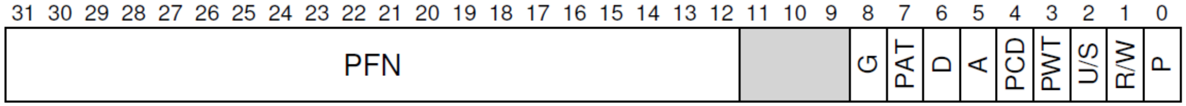


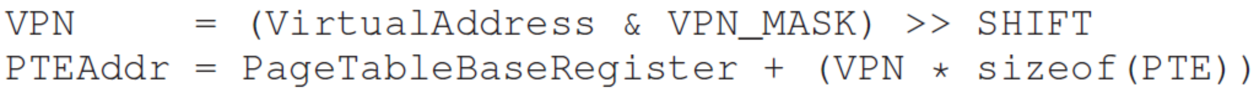
Figure 18.5: **An x86 Page Table Entry (PTE)**

- A → accessed bit (tells if page was accessed recently)
- D → dirty bit (tells if page was modified after it was brought in memory)
- P → present bit (if 0, the PTE is not valid for address translation)
- (PWT,PCD,PAT,G) → determine how caching works
- R/W → read/write bit tells if writes are allowed on this page
- U/S → user/supervisor bit tells if user-mode processes can access this page

Is paging slow?

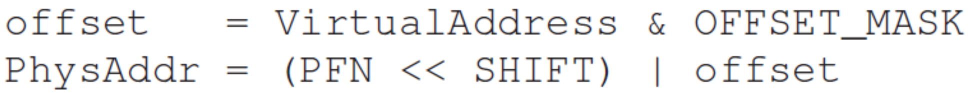
- Where is my page table located in memory?
  - Page Table Base Register

- Address Translation steps:



```
VPN     = (VirtualAddress & VPN_MASK) >> SHIFT
PTEAddr = PageTableBaseRegister + (VPN * sizeof(PTE))
```

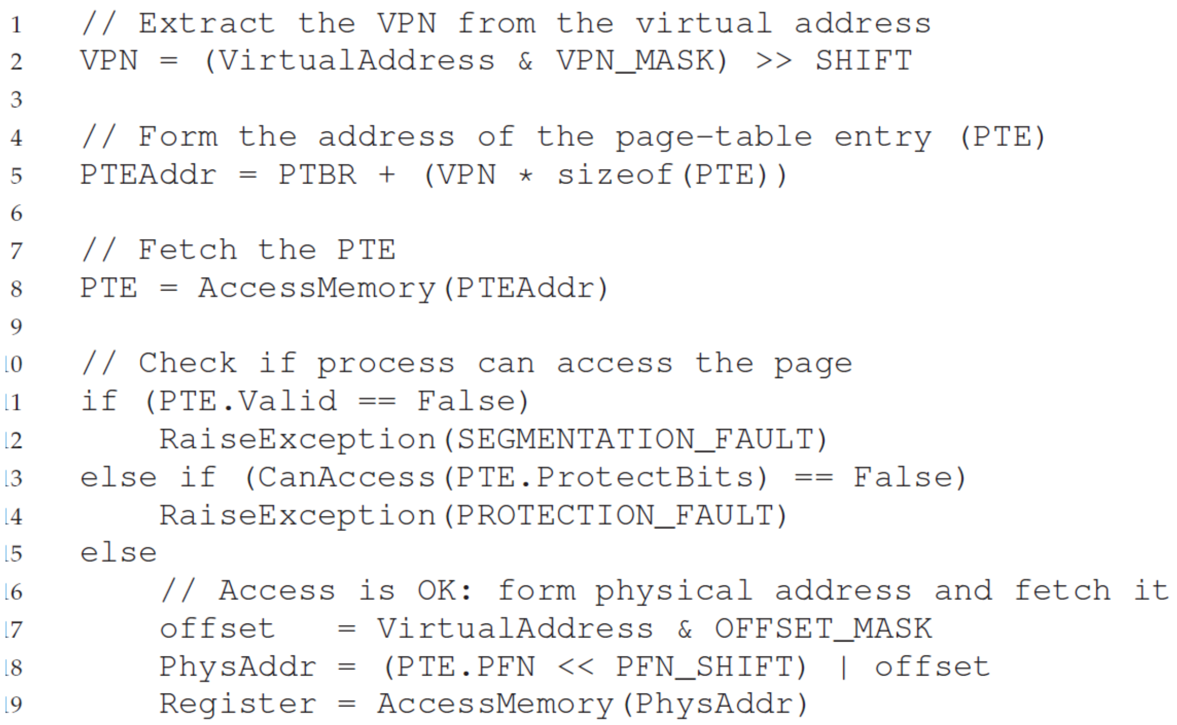
  - read physical Page Frame Number from PTEAddr and use it to get Physical Address:



```
offset   = VirtualAddress & OFFSET_MASK
PhysAddr = (PFN << SHIFT) | offset
```

  - Read data from PhysAddr

Is paging slow? … contd.



```
// 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)
```

Figure 18.6: **Accessing Memory With Paging**

Is paging slow? … contd.2

- For each memory access we access RAM twice
  - Once to read PFN from the page table
  - Second to read the actual data

Example:

address of array (variable array)

index of array (variable i)

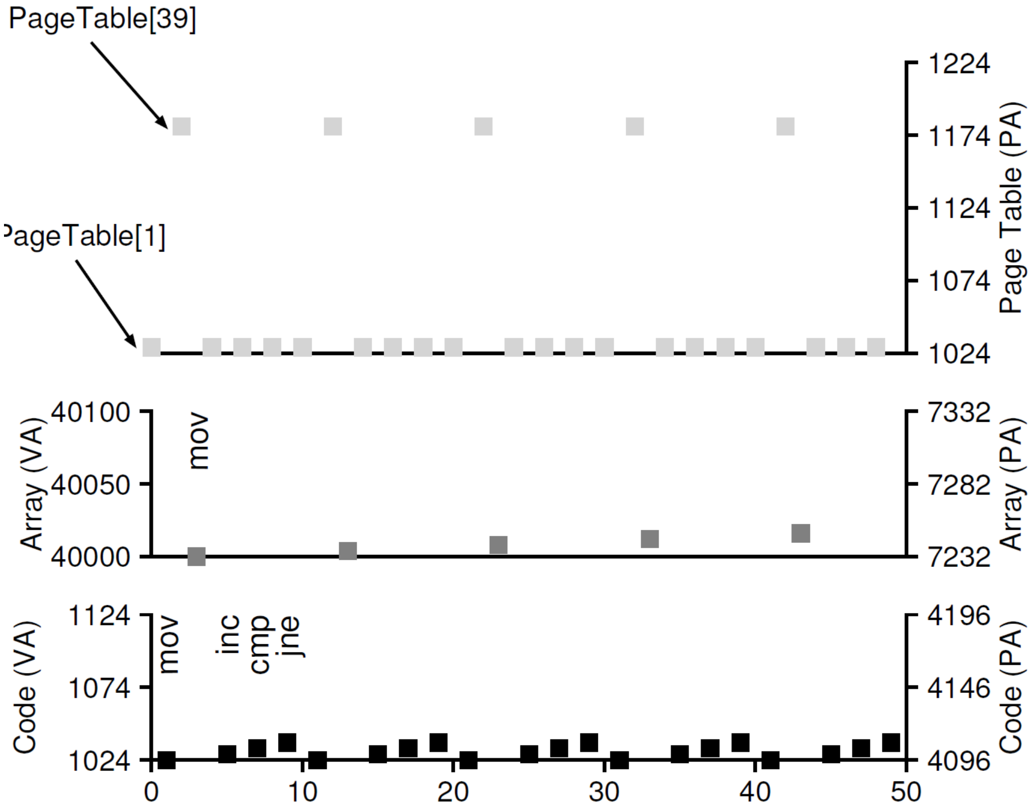
```
int array[1000];
...
for (i = 0; i < 1000; i++)
    array[i] = 0;
```

```
1024 movl  $0x0,(%edi,%eax,4)
1028 incl  %eax
1032 cmpl  $0x03e8,%eax
1036 jne   0x1024
```

```
Line 1024 == *(ptr+i)=0
Line 1028 == i++
Line 1032 and Line 1036 == if(i != 1000) goto line 1024
```

Is paging slow? contd… 3

	V.A	VPN	P.A	PFN
P.Table			1024	1
Code	1024	1	4096	4
Array	40000	39…42	7232	7…10



Summary

- We have introduced paging as an improvement over segmentation
- Pros
  - Does not lead to external fragmentation
  - Allows sparse use of virtual address spaces
- Cons
  - Slows the machine down due to more memory accesses
  - Waste memory (for storing page tables)