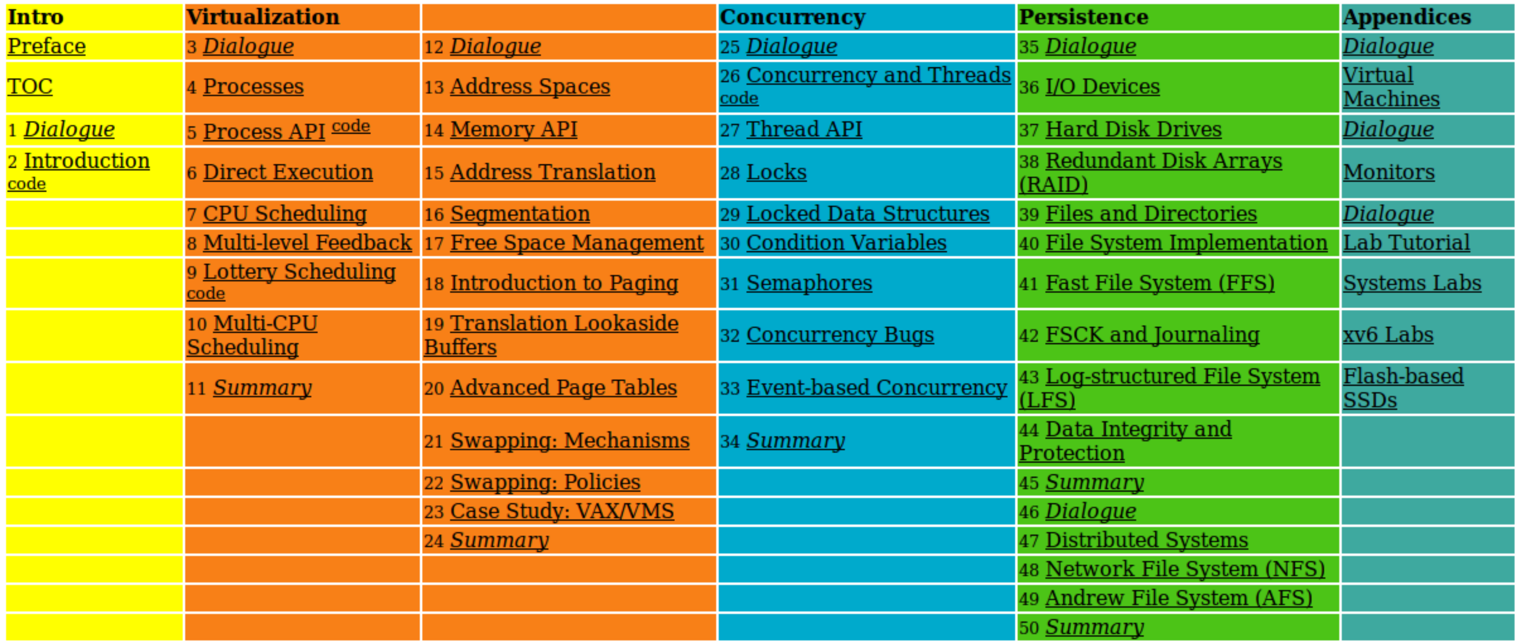
CSIS 429 Operating Systems

Lecture 7: Paging

September 28th 2020

Textbook chapters

Read “Intro to Paging” and “Translation L B”



Intro	Virtualization		Concurrency	Persistence	Appendices
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	*Dialogue*
TOC	4 Processes	13 Address Spaces	26 Concurrency and Threads code	36 I/O Devices	Virtual Machines
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API	37 Hard Disk Drives	*Dialogue*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks	38 Redundant Disk Arrays (RAID)	Monitors
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	*Dialogue*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables	40 File System Implementation	Lab Tutorial
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores	41 Fast File System (FFS)	Systems Labs
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	xv6 Labs
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	Flash-based SSDs
		21 Swapping: Mechanisms	34 *Summary*	44 Data Integrity and Protection	
		22 Swapping: Policies		45 *Summary*	
		23 Case Study: VAX/VMS		46 *Dialogue*	
		24 *Summary*		47 Distributed Systems	
				48 Network File System (NFS)	
				49 Andrew File System (AFS)	
				50 *Summary*	

Remember: Virtual Addresses

Every address generated by a process is treated as a "virtual address" by the OS

OS translates each virtual address to a DRAM address.

Base + Bounds MMU approach.

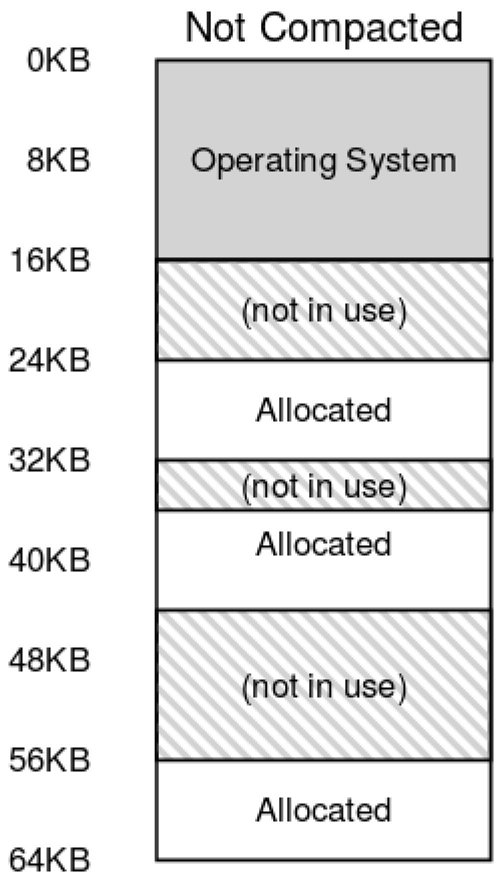
Generalizes to Segmentation.

H/w involved in every memory reference!

OS needs to track changes at certain times.

Problems with Segmentation

- Not flexible:
  - what if a heap segment is “boxed in”?
  - what if we swap a segment out to disk while a process is running?
- External fragmentation



Solution: Paging

Basic idea:

- ✔ Divide all of physical memory into fixed size page **frame**s
- ✔ Divide logical memory into **page**s and OS tracks each page
- ✔ With help from MMU and CPU hardware

The Crux of the Problem

Use pages to avoid the problems of segmentation

How do we do this?

What techniques can we use while minimizing time & memory overhead?

What are Page Frames?

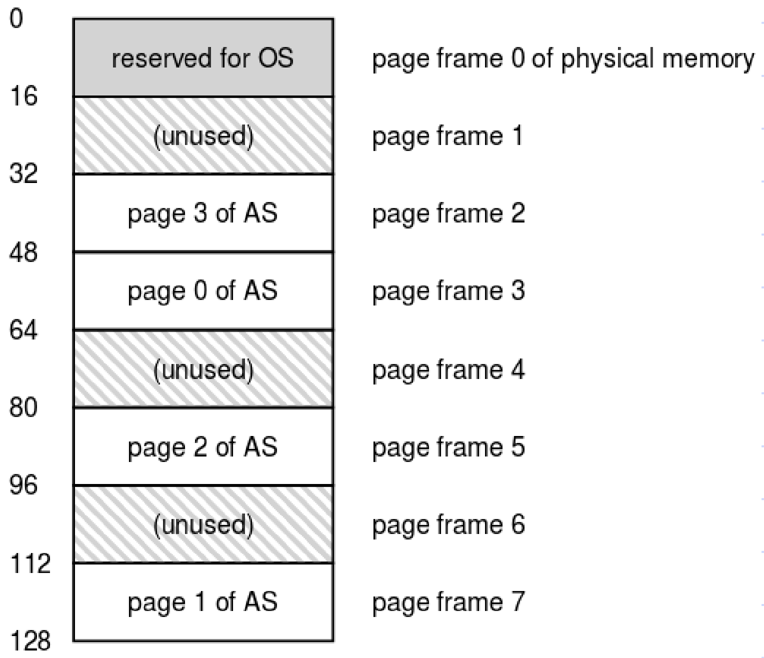
Physical memory is divided up into fixed size slots called page **frame**s.

How many address bits do we need to address each byte?

Minimum number of bits needed to refer to each page?

Small, managable examples:

A 128-Byte Physical Memory

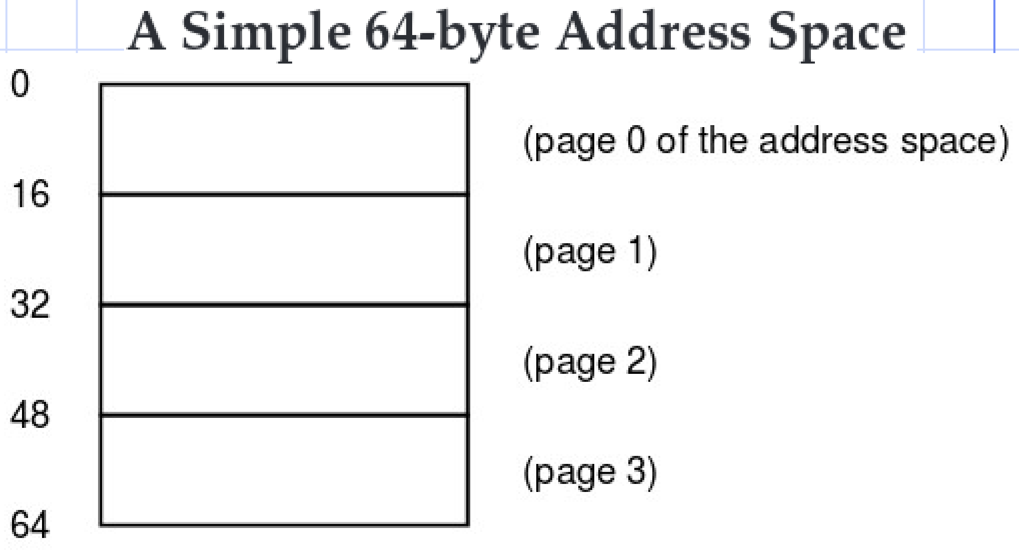


What are Pages?

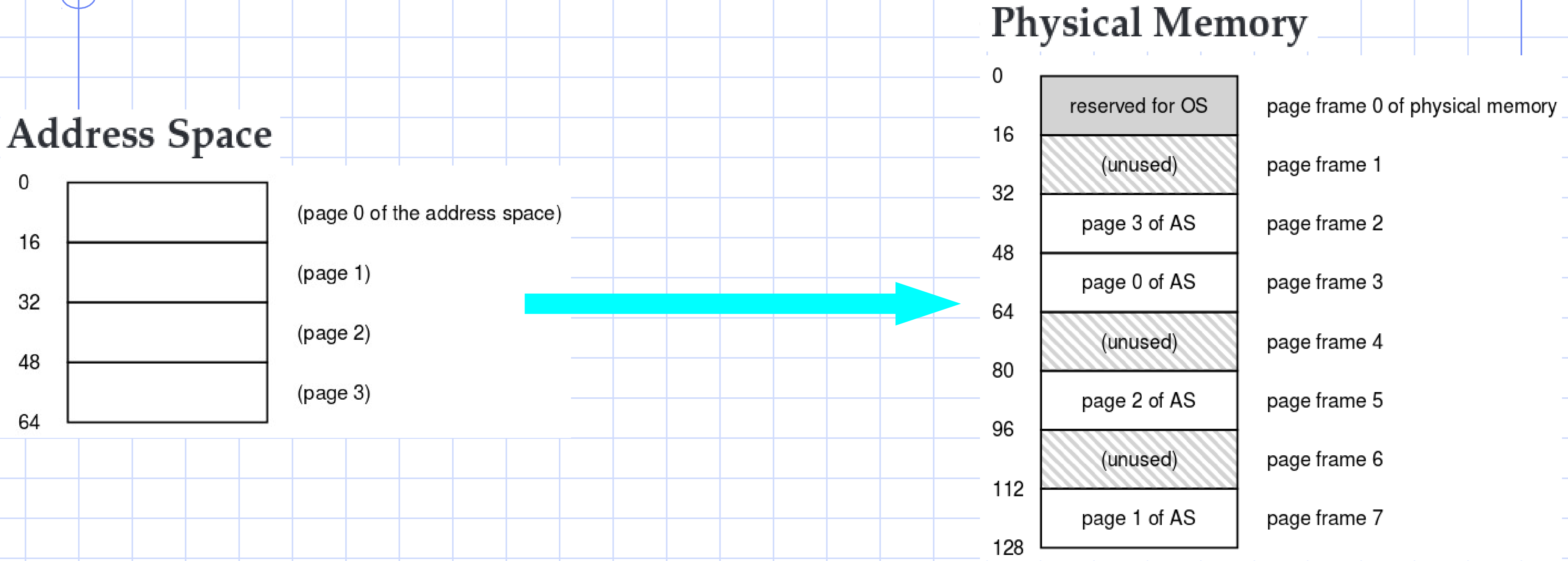
- Logical memory (or address space) is divided up into fixed size slots called pages.

How many address bits do we nee
to address each byte of address space?

Minimum number of bits needed to refer to each page?



How are Pages and Page Frames related?



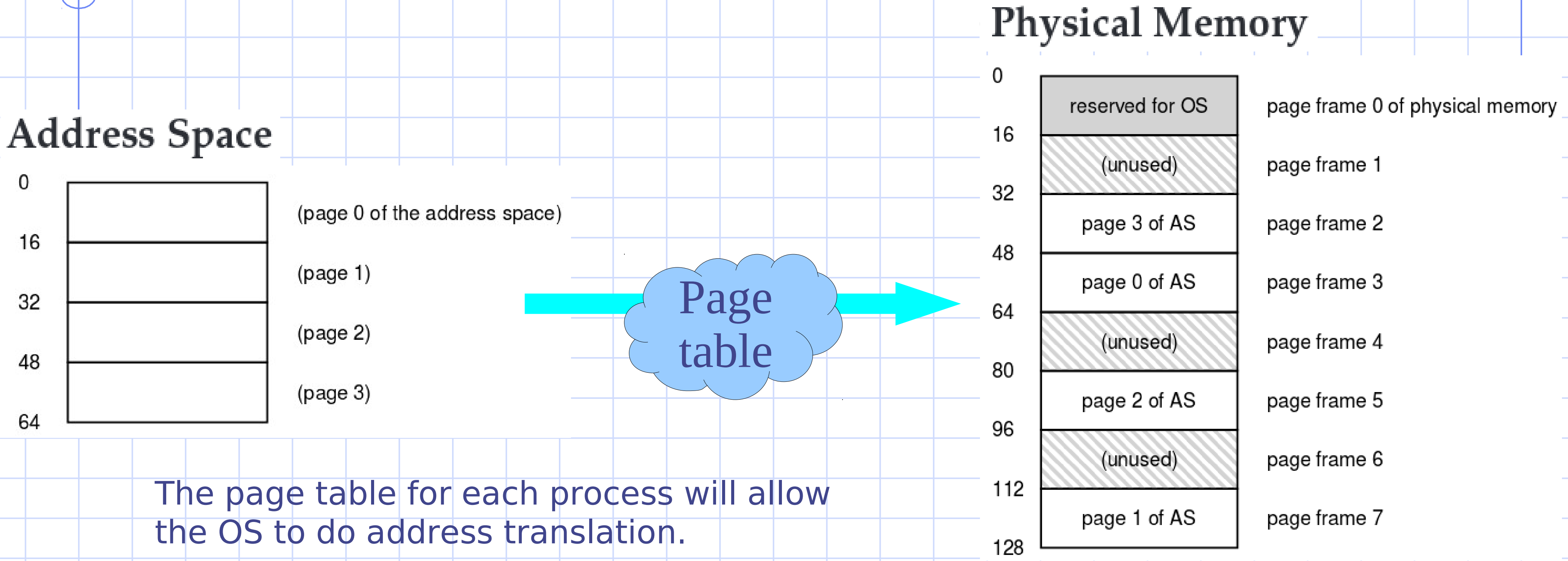
Notice: frames are either in use or not → solves external fragmentation!

Advantages of Paging

Each frame is either in use or not
 → no need for compacting or defragmenting memory
 → no more external fragmentation.

No need to track direction of segment growth
 – when we need more heap/stack space, just allocate one or more frames.

Where is a Page in Physical Memory?



The page table for each process will allow the OS to do address translation.

Address Translation with Paging

Example: Instruction to load data from memory to register

```
movl <virtual-addr>, %a
```

If our virtual address space is 64 bytes, how many bits do we need for the virtual address?

If our page size is 16 bytes, how many bits do we need for the “offset” within a page?

How many pages do we have in our address space? How many bits to choose a page?

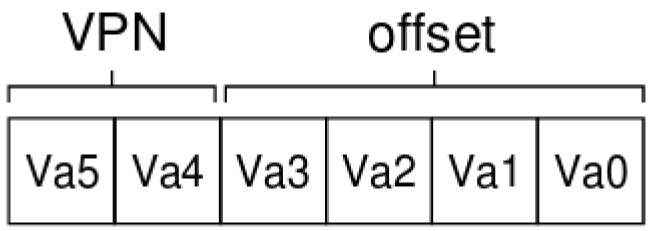
Address Translation with Paging

Example: Instruction to load data from memory to register

`movl <virtual-addr>, %a`

64 byte virtual address space; page size is 16 bytes

→ Virtual address:



Address Translation with Paging

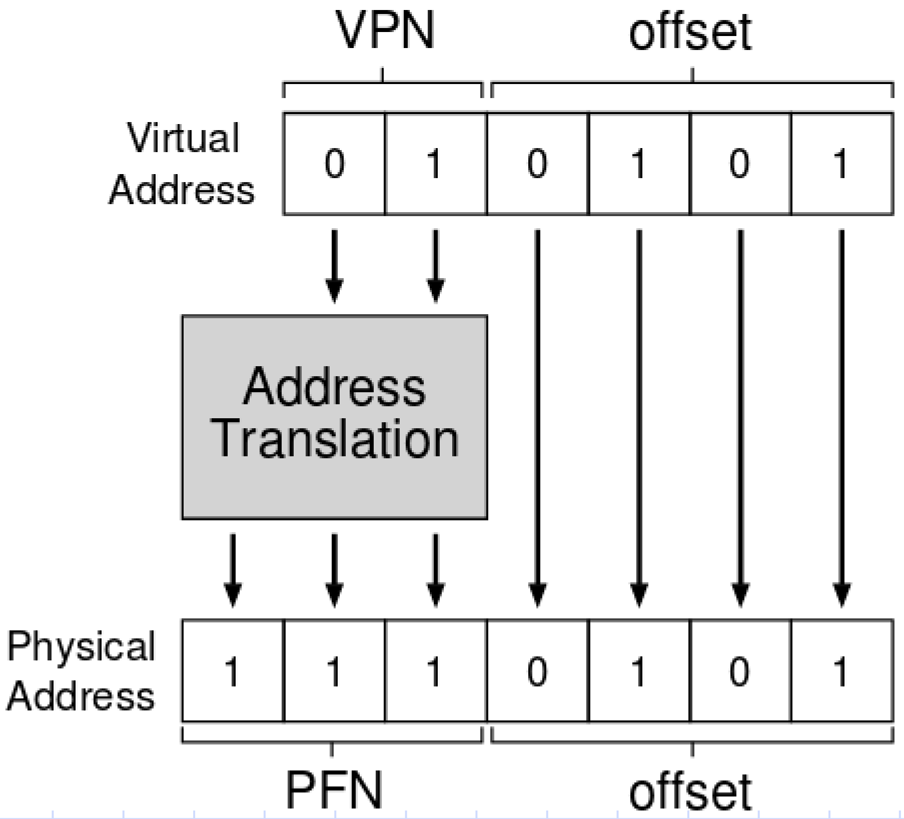
Example: Instruction to load data from memory to register

```
movl <virt-addr>, %eax
```

64 byte virtual address space

page size is 16 bytes

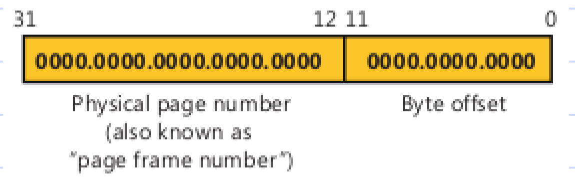
128 byte physical address space

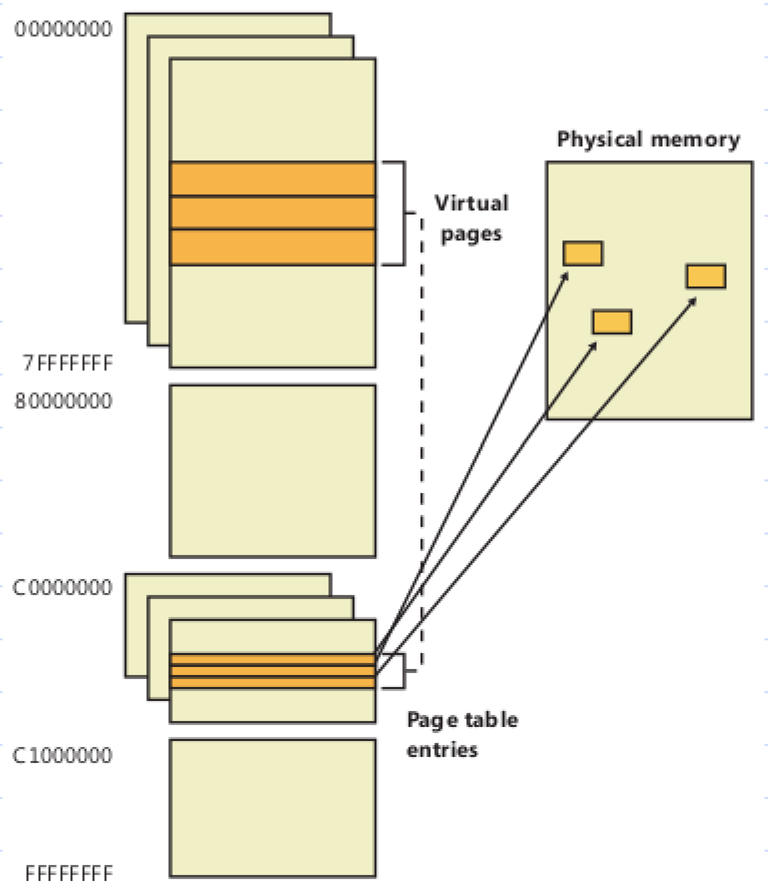


Note: offset stays the same in the virtual and physical addresses. Why?

Win x86 Virtual Address Translation

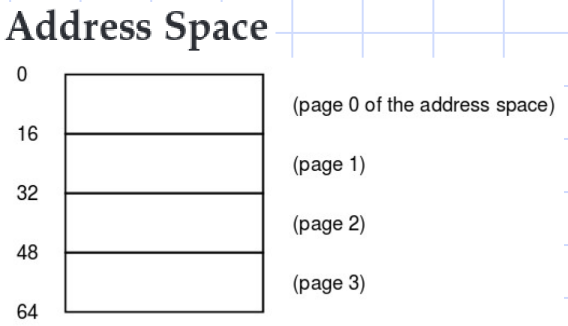
- 32-bit (x86) Windows has a 4GB process address space



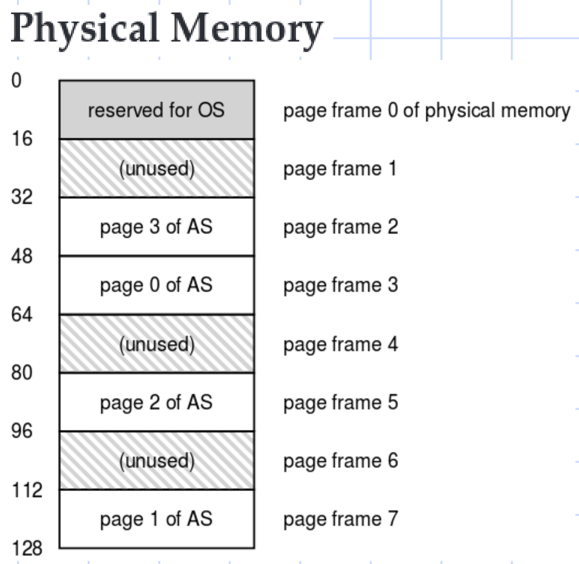


What do page tables look like?

- Actual page tables can be very big b/c addresses can be 32-48 bits. In our simple 64-byte example:



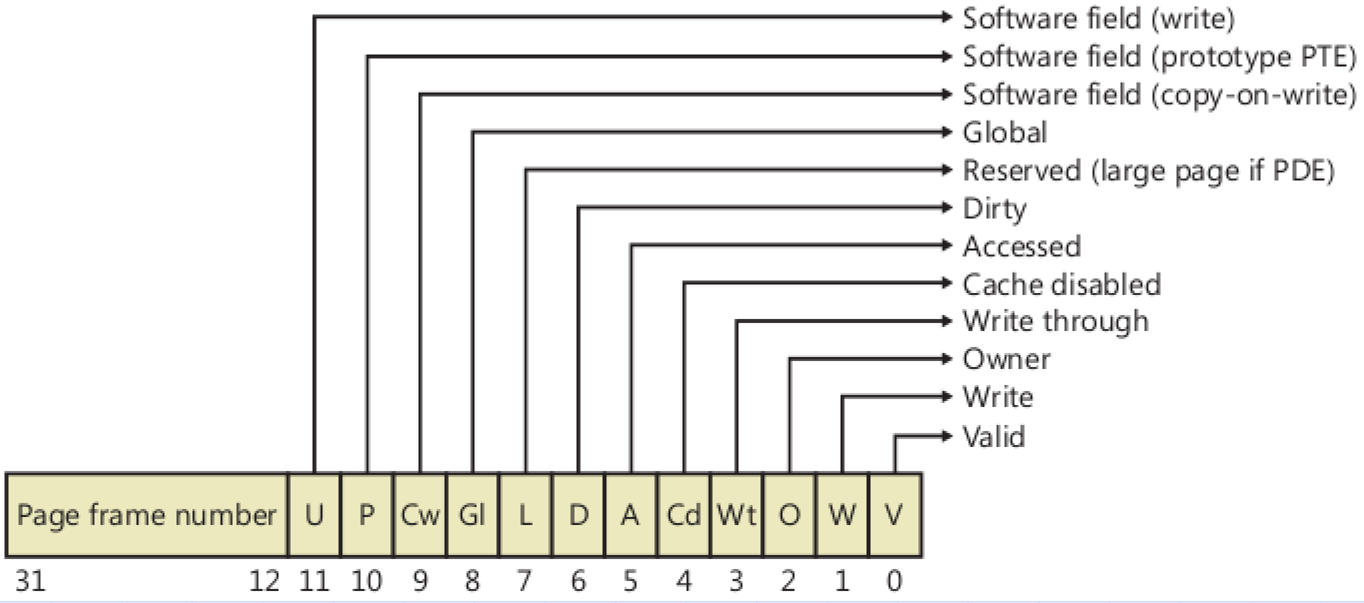
Virtual	Physical
00	011
01	111
10	101
11	010



What’s in each page table entry?

- No need to store Virtual Page Number
- Can index into table using Virtual Page num: PT[ivpn]
- If a virtual page has not been allocated a frame, need to indicate that → Valid Bit
- Protection bits for read, write, execute
- Present bit for “in RAM” vs. “on Disk” (swapped out)
- Modified since brought to memory → “Dirty” bit
- Reference Bit set if page was read, written or executed

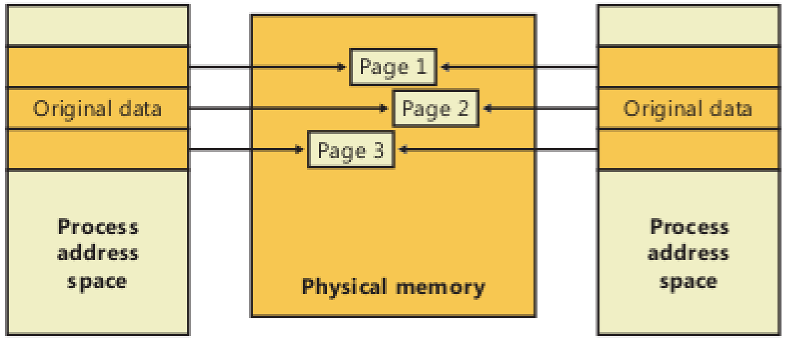
x86 Hardware Page Table Entry



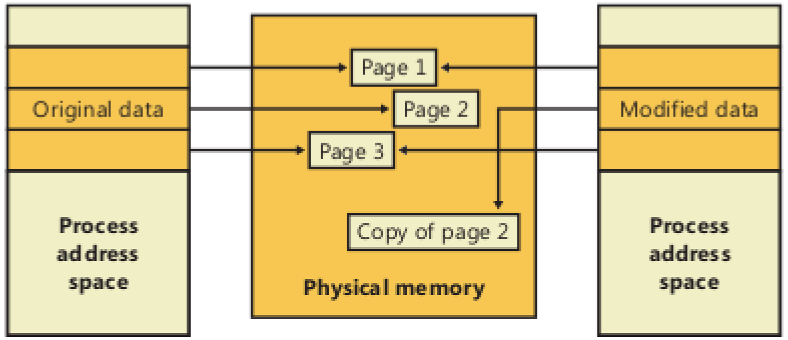
Example use of page table entry

- Modified since brought to memory → “Dirty” bit

Can be used to implement Window’s “Copy on Write” when 2 processes share pages



Before write



After write

How much time will paging take?

Use our old example:

Instruction to load data from memory to register

```
movl 21, %a
```

i.e. read contents of memory location 21 into register A

OS translates the virtual addr “21” to a physical address:

i. Get an entry from the process’s page table in memory

ii. Add the offset to the frame address → “117”

Two memory accesses plus processing to get 1 byte!

What about instructions?

In our old example - instruction to load data from memory to register

```
movl 21, %a
```

We did not account for time taken to read the instruction which is at some virtual address which has to be converted to a physical address – by accessing a Page Table!!

Can all this work?   Yes, but we need some help from H/w