CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

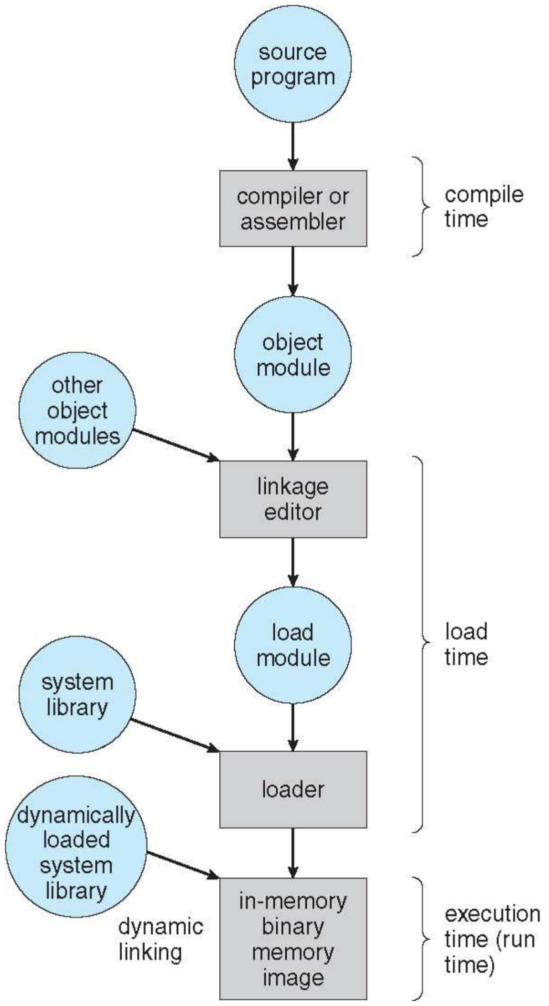
Fall 2025 L16

Main Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Multistep Processing of a User Program



Colorado State University

Address Binding Questions

- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Addresses represented in different ways at different stages of a program's life
  - **Source code** addresses are symbolic
  - **Compiled code** addresses **bind** to relocatable addresses
    - i.e., "14 bytes from beginning of this module"
  - **Linker or loader** will bind relocatable addresses to absolute addresses
    - i.e., 74014
  - Each binding maps one address space to another

Colorado State University

Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Linking: Static vs Dynamic

- **Linking**
  - Takes some smaller executables and joins them together as a single larger executable.
- **Static linking** – system libraries and program code combined by the loader into the binary image
  - Every program includes library: wastes memory
- **Dynamic linking** –linking postponed until execution time
  - Operating system locates and links the routine at run time

Dynamic Linking

- **Dynamic linking** –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for
  - shared libraries

Dynamic loading of routines

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- OS can help by providing libraries to implement dynamic loading
- Static library
    - Linux. .a (archive)
    - Windows .lib (Library)
- Dynamic Library
    - Linux .so (Shared object)
    - Windows .dll (Dynamic link library)

Join the UTA Staff!

Apply to work as an Undergraduate Teaching Assistant (UTA) today!

Develop Great Professional Skills!

- Working on a team
- Debugging and Code Comprehension
- Communication

Improve your technical knowledge!

- The best way to refine your knowledge is to teach it

Great for Resumes!

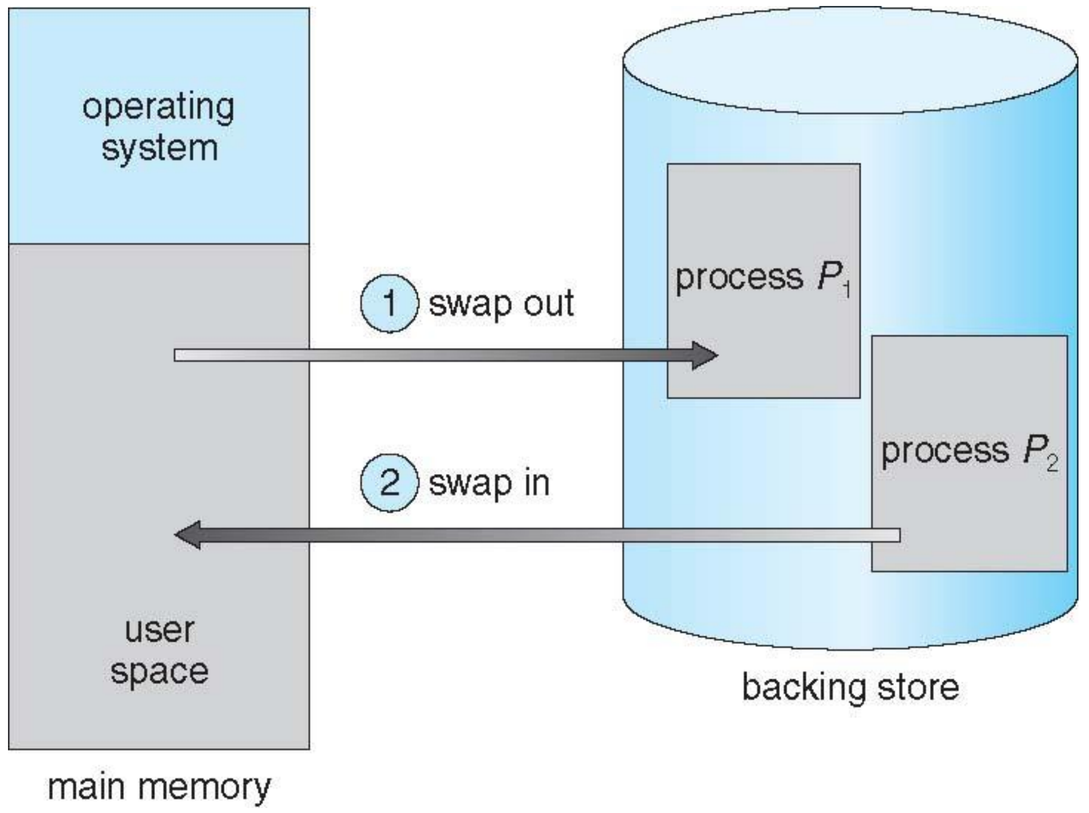
Build rapport with department instructors!

Earn more than $15 per hour!

Swapping a process

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Do we really need to keep the entire process in the main memory? Stay tuned.

Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 100MB/50MB/s = 2 seconds
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4 seconds + some latency
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used by a process

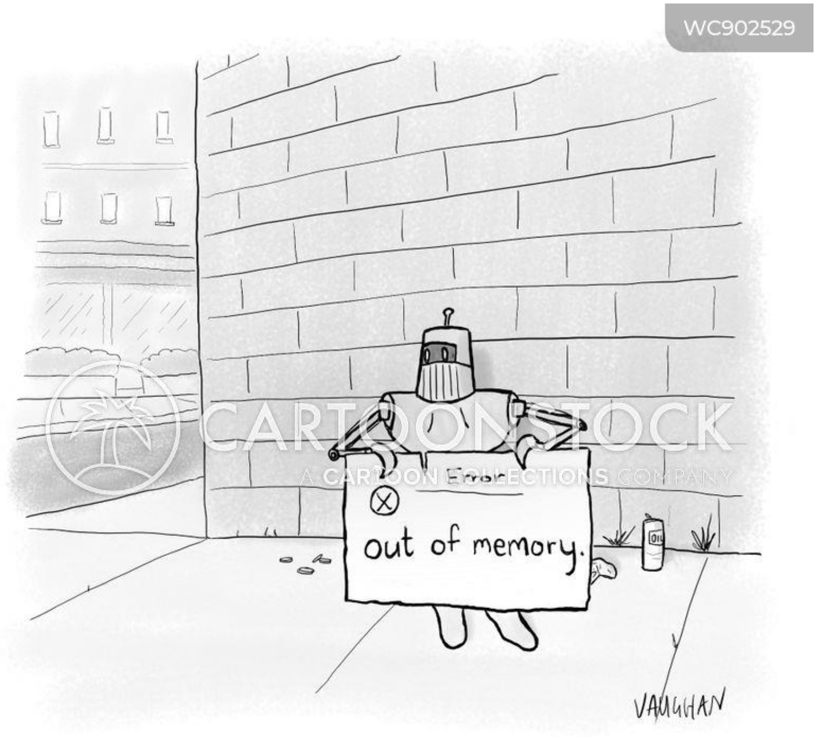
Context Switch Time and Swapping (Cont.)

- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Course Notes: HW4

- **Help Session Today**: 5 PM Room CSB 130
- HW4 output formatting: The RAMDesk team and the our GTAs have been working together on developing the autograder script. The output format needs to be revised for autograder to work. Will be available soon in the revised HW4 document.

Memory Allocation



CX920051
"AFTER YOU'VE WISHED FOR ADDITIONAL MEMORY, YOU CAN STOP WISHING FOR MORE UPGRADES."

Memory Allocation Approaches

- **Contiguous allocation**: entire memory for a program in a single contiguous memory block. Find where a program will "fit". earliest approach
- **Segmentation**: program divided into logically divided "segments" such as main program, functions, stack etc.
  - Need table to track segments.
- **Paging**: program divided into fixed size "pages", each placed in a fixed size "frame".
  - Need table to track pages.

Colorado State University

Contiguous Allocation

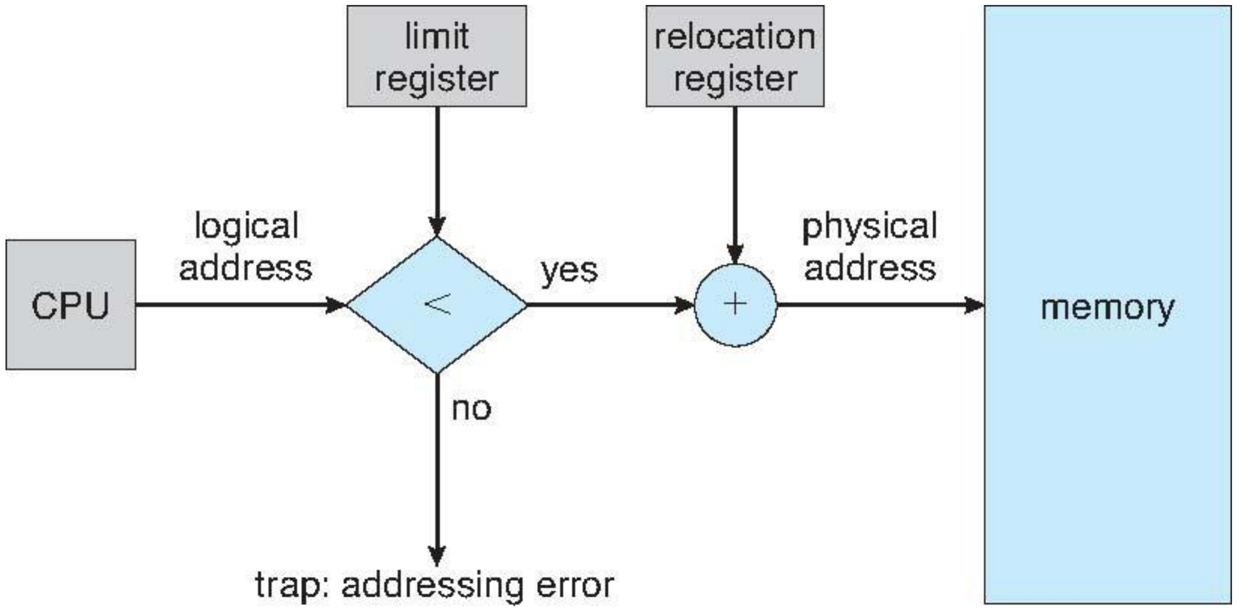
- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vectors
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

- Registers used to protect user processes from each other, and from changing operating-system code and data
  - Relocation (Base) register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers

Contiguous Allocation



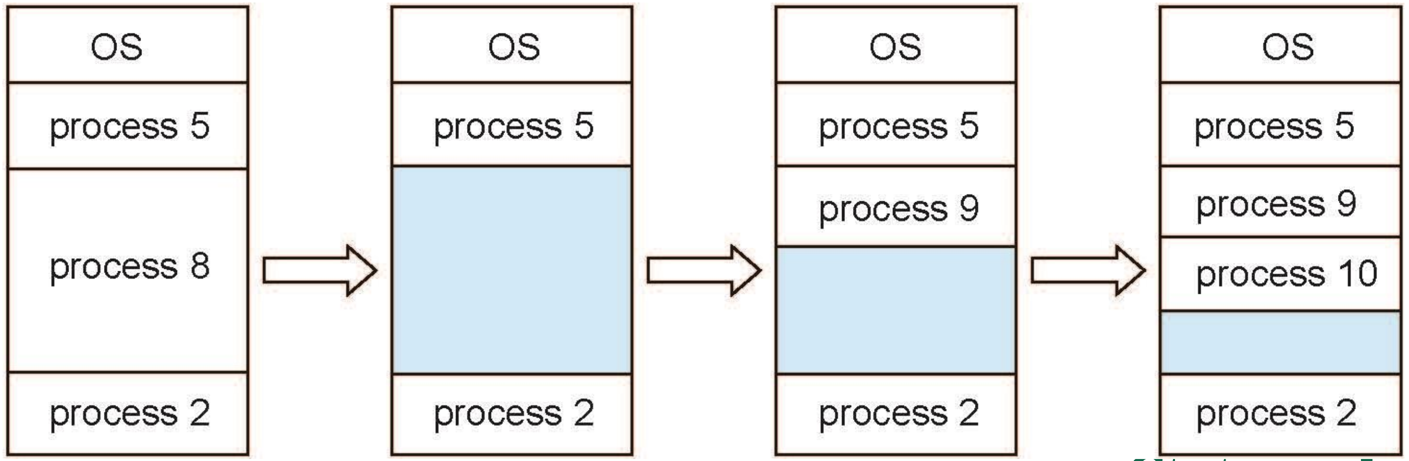
MMU maps logical address *dynamically*
Physical address = relocation reg + valid logical address

Colorado State University

Multiple-partition allocation

Contiguous Allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Colorado State University

Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

Simulation studies:

- First-fit and best-fit better than worst-fit in terms of speed and storage utilization
- Best fit is **slower** than first fit . Surprisingly, it also results in more **wasted memory** than first fit
  - Tends to fill up memory with tiny, useless holes

Fragmentation

- **External Fragmentation** – External fragmentation: memory wasted due to small chunks of free memory interspersed among allocated regions
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Simulation analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Paging vs Segmentations

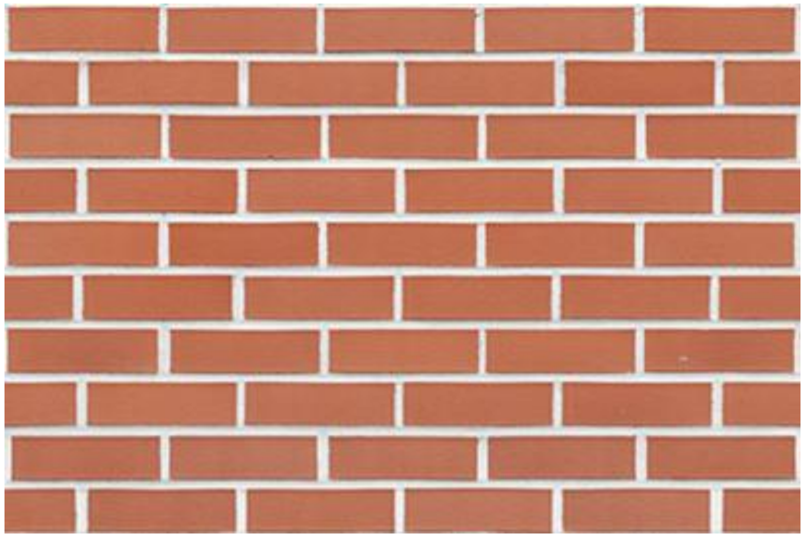
Segmentation: program divided into logically divided "segments" such as main program, function, stack etc.

- Need table to track segments.
- Term "segmentation fault occurs": improper attempt to access a memory location

Paging: program divided into fixed size "pages", each placed in a fixed size "frame".

- Need table to track pages.
- No external fragmentation
- Increasingly more common

Paging vs Segmentations



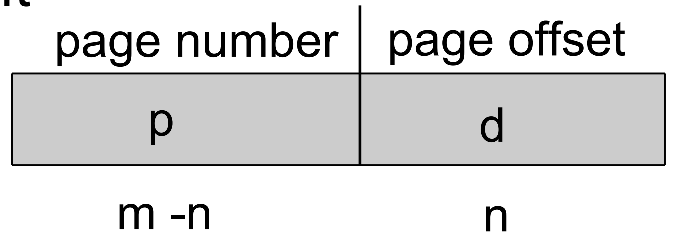
Colorado State University

Pages

- Pages and frames
  - Addresses: page number, offset
- Page tables: mapping from page # to frame #
  - TLB: page table caching
- Memory protection and sharing
- Multilevel page tables

Address Translation Scheme

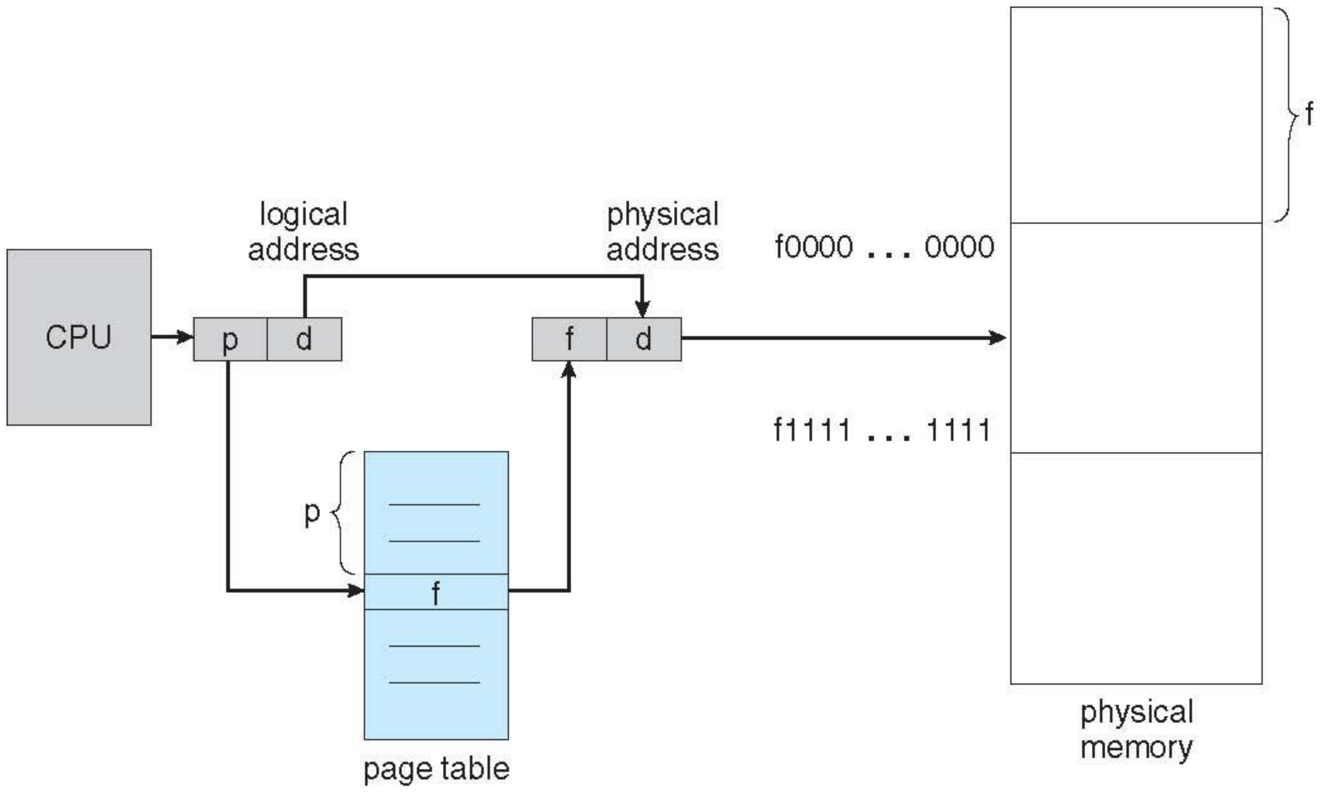
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



  - For given logical address space 2^m and page size 2^n

Colorado State University

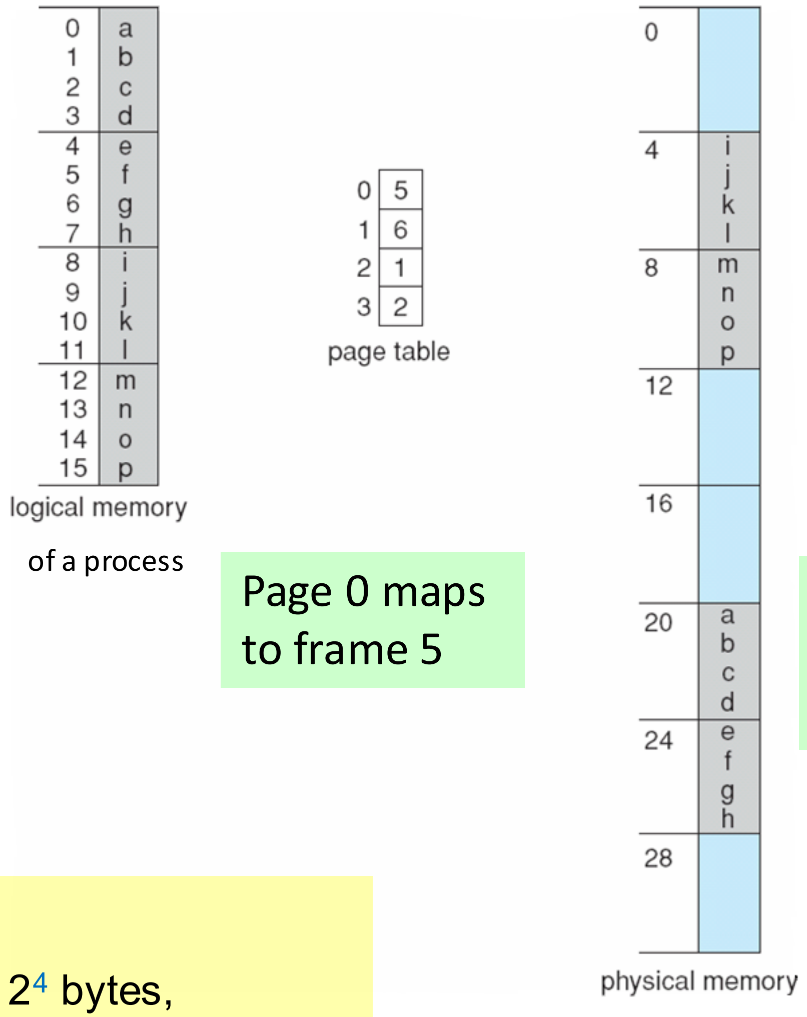
Paging Hardware



Page number p mapped into the frame number f.
The offset d needs no mapping.

Colorado State University

Paging Example



of a process

Page 0 maps to frame 5

8 frames
Frame number 0-to-7

Example:
Logical add: **00** 10 (2)
Phyical Add: **101** 10 (22)

Ex: m=4 and *n*=2

- Logical add. space = 2^4 bytes,
- 2^2=4-byte pages
- 32-byte physics memory with 8 frames

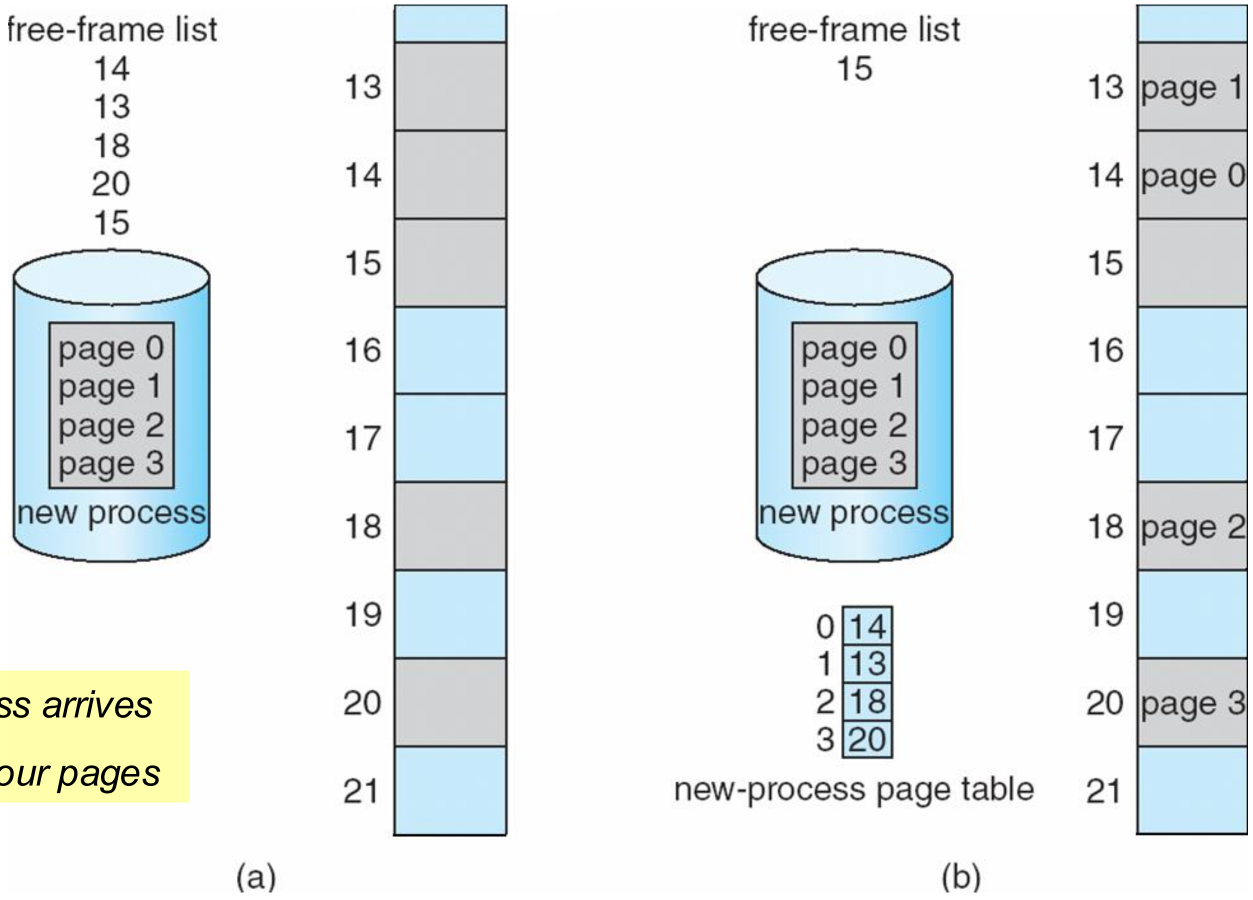
Colorado State University

Paging (Cont.)

- Internal fragmentation
  - Ex: Page size = 2,048 bytes, Process size = 72,766 bytes
    - 35 pages + 1,086 bytes
    - Internal fragmentation of 2,048 - 1,086 = 962 bytes wasted
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
    - But each page table entry takes memory to track
  - Page size
    - X86-64: 4 KB (common), 2 MB (“huge” for servers), 1GB (“large”)
- Process view and physical memory now very different
- By implementation, a process can only access its own memory unless ..

Colorado State University

Free Frame allocation



A new process arrives

That needs four pages

Before allocation

After allocation

Colorado State University

Implementation of Page Table

Page table is kept in main memory

- Page-table base register (PTBR) points to the page table
- Page-table length register (PTLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction

One page-table
For each process

The *two memory access problem* can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)

TLB: cache for Page Table

Colorado State University

Caching: The General Concept

- Widely used concept:
  - keep small subset of information likely to needed in near future in a fast accessible place
  - Hopefully the "**Hit Rate**" is high

Challenges:
  - 1. Is the information in cache? 2. Where?
  - Hit rate vs cache size

Examples:
  - Cache Memory ("Cache"):
    Cache for Main memory Default meaning for this class
  - Browser cache: for browser
  - Disk cache
  - Cache for Page Table: TLB

Implementation of Page Table (Cont.)

- Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush TLB at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

TLB: cache for page Table

Associative Memory (hardware block)

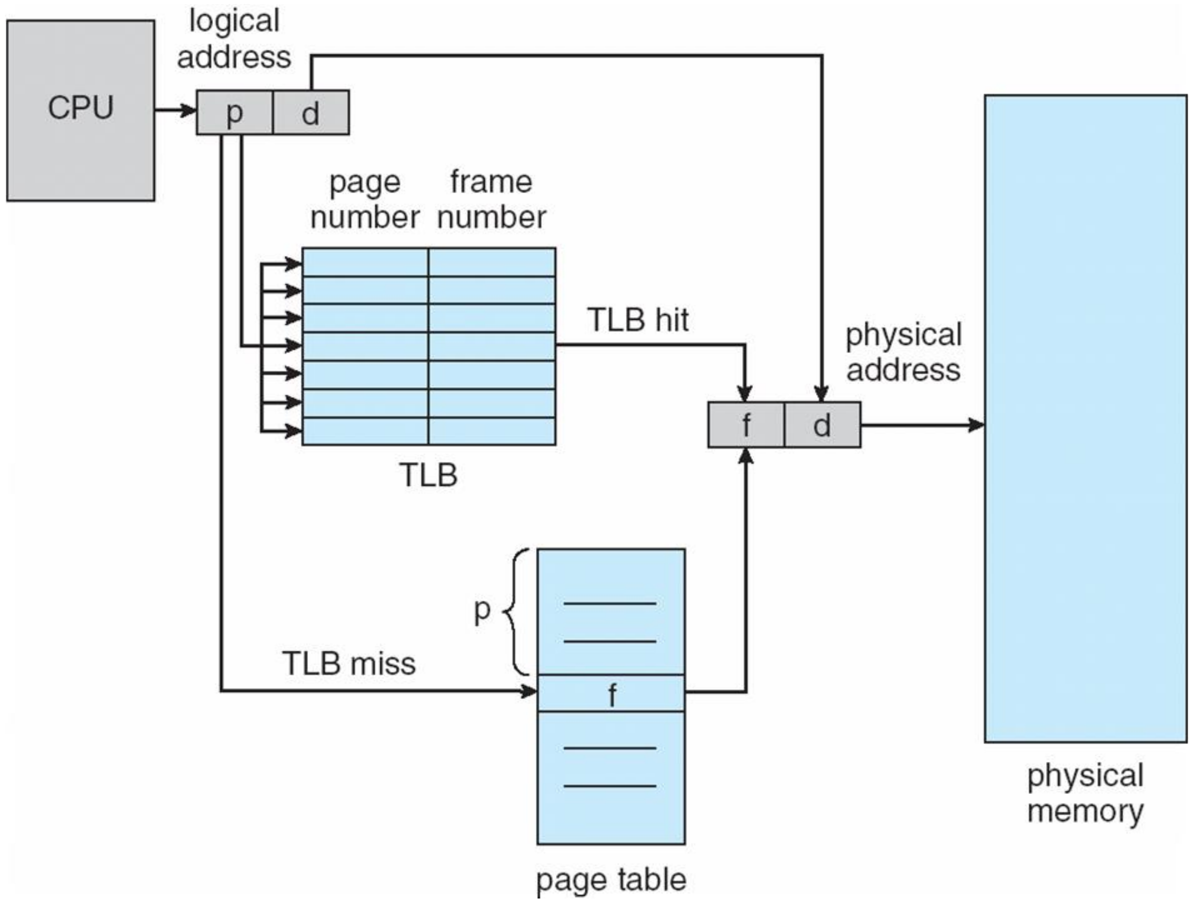
- Associative memory –parallel search using hardware
  - "Content addressable memory": Electronics is very expensive

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out ("Hit")
  - Otherwise get frame # from page table in memory ("Miss")

Colorado State University

Paging Hardware With TLB



TLB Miss: page table access may be done using hardware / software

Colorado State University

Effective Access Time

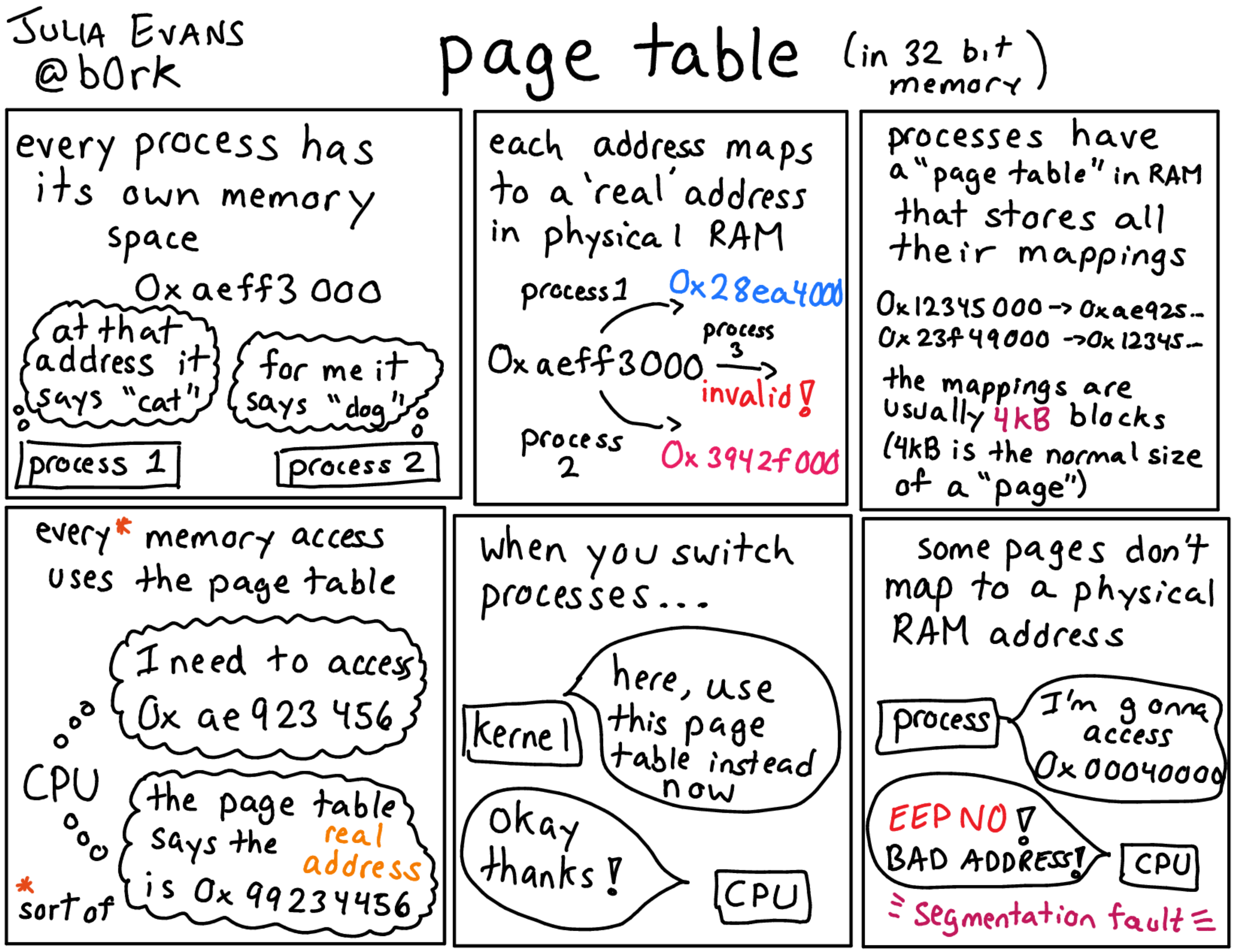
On average how long does a memory access take?

- Associative Lookup = ε time units
  - Can be < 10% of memory access time (*MAT*)
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Effective Access Time (EAT): probability weighted

  EAT = $\alpha\,(\varepsilon+\text{MAT}) + (1-\alpha)(\varepsilon+2.\text{MAT})$
- Ex:

  Consider α = 90%, ε = negligible for TLB search, 100ns for memory access time
  - EAT = 0.90 x 100 + 0.10 x 200 = 110ns
- Consider more realistic hit ratio -> α = 99%,
  - EAT = 0.99 x 100 + 0.01 x 200 = 101ns

FAQ

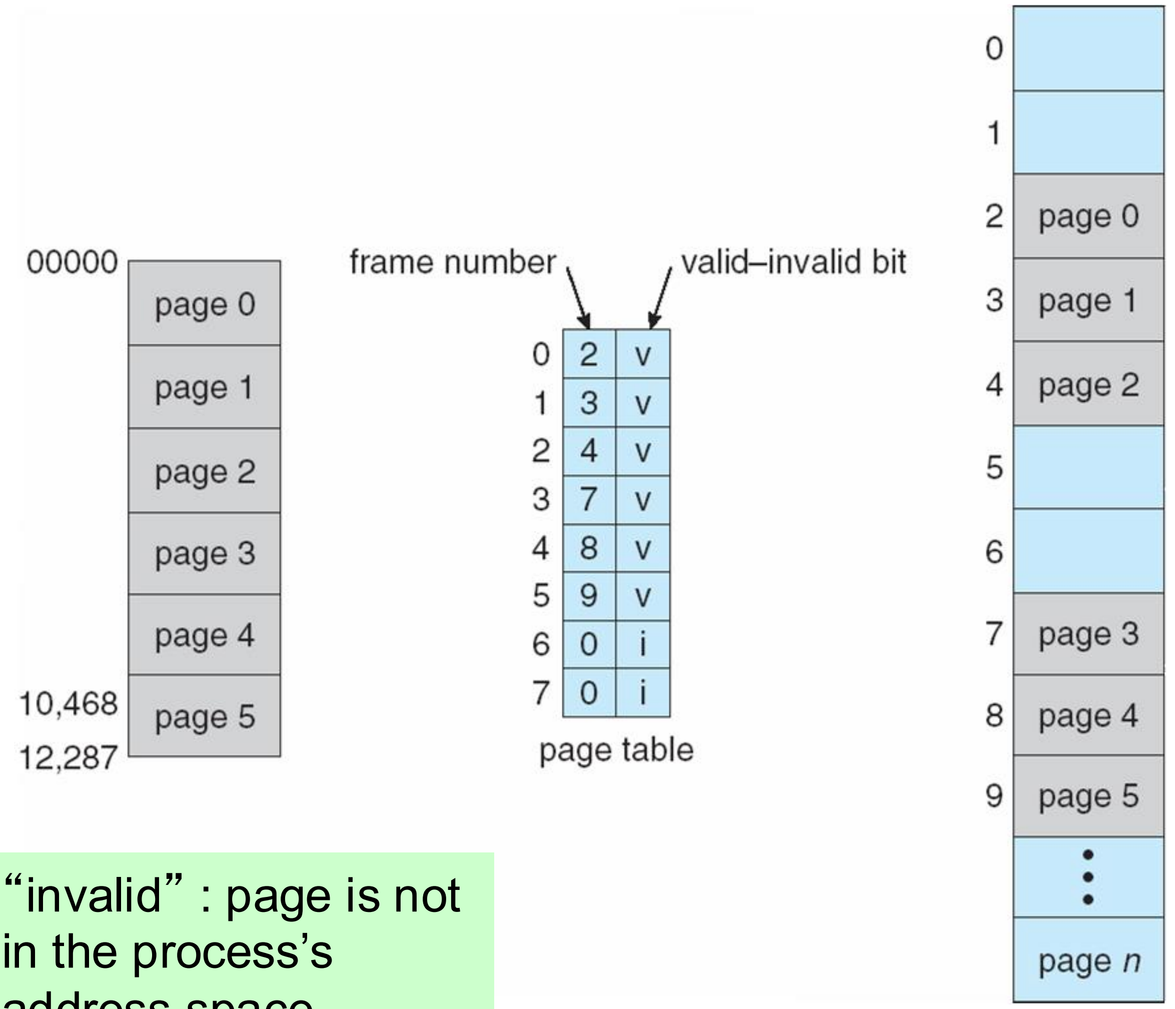


JULIA EVANS @b0rk
page table (in 32 bit memory)
every process has its own memory space
0x aeff3 000
at that address it says "cat"
for me it says "dog"
process 1
process 2
each address maps to a 'real' address in physical RAM
process 1
0x28ea4000
0x aeff3000
process 3
invalid!
process 2
0x 3942f000
processes have a "page table" in RAM that stores all their mappings
0x12345000 -> 0xae925..
0x23f49000 -> 0x12345..
the mappings are usually 4kB blocks (4kB is the normal size of a "page")
every* memory access uses the page table
I need to access 0x ae923456
CPU
the page table says the real address is 0x 99234456
* sort of
when you switch processes...
kernel
here, use this page table instead now
okay thanks!
CPU
some pages don't map to a physical RAM address
process
I'm gonna access 0x 00040000
EEP NO! BAD ADDRESS!
CPU
≡ Segmentation fault ≡

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
- Any violations result in a trap to the kernel

Valid (v) or Invalid (i) Bit In A Page Table



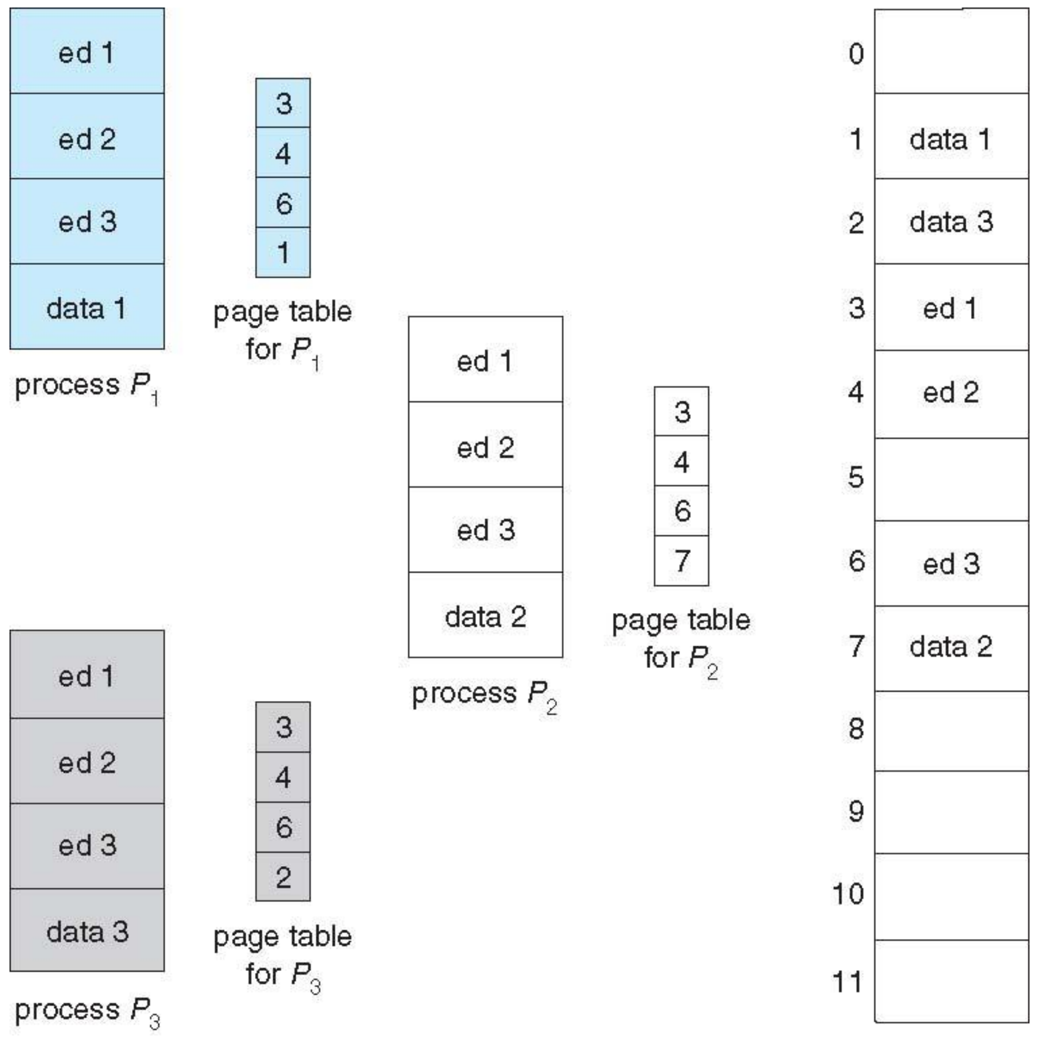
"invalid" : page is not in the process's address space.

Colorado State University

Shared Pages among Processes

- **Shared code**
  - One copy of read-only (**reentrant** **non-self modifying**) code *shared* among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



ed1, ed2, ed3 (frames 3, 4, 6) shared

Colorado State University

Overheads in paging: Page table and internal fragmentation

Optimal Page Size Computation:

page table size vs internal fragmentation tradeoff

- Average process size = ***s***
- Page size = ***p***
- Size of each entry in page table = ***e***
  - Pages per process = ***s/p***
  - ***se/p:*** Total page table space for average process
  - Total Overhead = Page table overhead + Internal fragmentation loss

    $= se/p + p/2$

Colorado State University

Optimal Page size: Page table and internal fragmentation

- Total Overhead = $se/p + p/\mathbf{2}$
- Optimal: Obtain derivative of overhead with respect to ***p,*** equate to 0

  $-se/p2 + 1/2 = 0$
- i.e. $p^2 = 2se$ or $p = (2se)^{0.5}$

Assume ***s*** = 128KB and ***e=8*** bytes per entry

- Optimal page size = 1448 bytes
  - In practice we will never use 1448 bytes
  - Instead, either 1K or 2K would be used
    - **Why?** Pages sizes are in powers of 2 i.e. $\mathbf{2^X}$
    - Deriving offsets and page numbers is also easier

Colorado State University

Page Table Size

Memory structures for paging can get huge using straight-forward methods

- Consider a 32-bit logical address space as on recent processors 64-bit on 64-bit processors
  - Assume page size of 4 KB (2^{12}) entries
  - Page table would have 1 million entries (2^{32} / 2^{12})
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone
    - Don't want to allocate that **contiguously** in main memory

2^{10}	**1024 or 1 kibibyte**
2^{20}	1M mebibyte
2^{30}	1G gigibyte
2^{40}	1T tebibyte

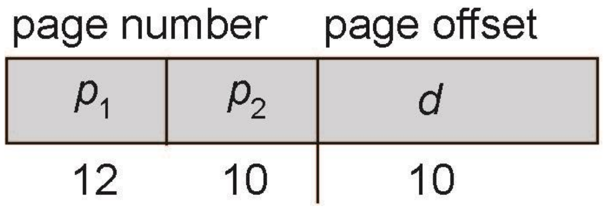
Colorado State University

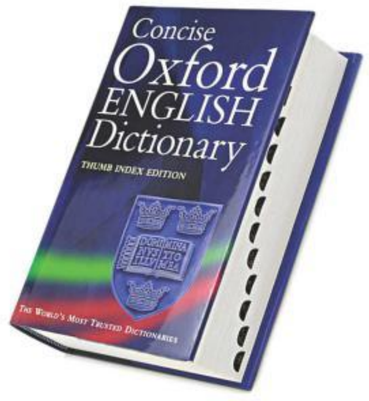
Issues with large page tables

- Cannot allocate page table **contiguously** in memory
- Solution:
  - Divide the page table into smaller pieces
  - **Page the page-table**
    - Hierarchical Paging

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

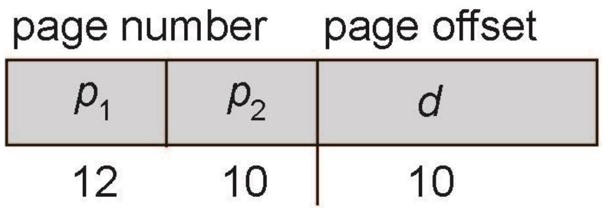


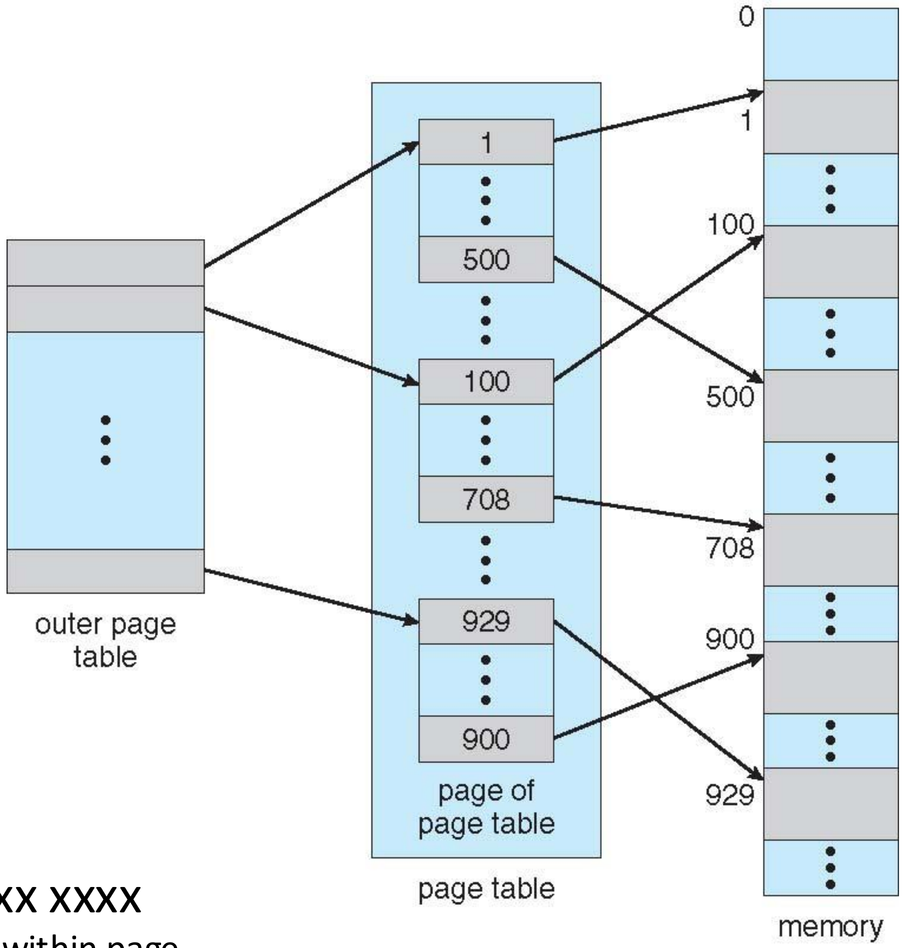


P1: indexes the outer page table
P2: page table: maps to frame

Colorado State University

Two-Level Page-Table Scheme



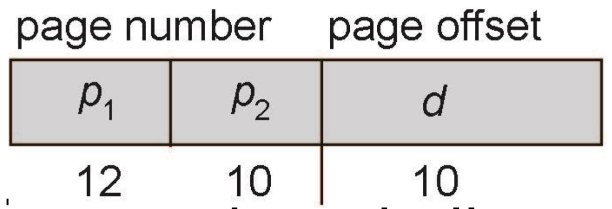


xxxx xxxx xxxx xxxx xxxx xx xx xxxx xxxx

Outer Page table page table offset within page

Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:

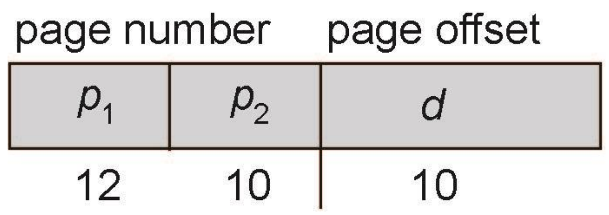


- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

Colorado State University

Two-Level Paging Example

- A logical address is as follows:



- One Outer page table: size 2^{12}
  entry: page of the page table
- Often only some of all possible 2^{12} Page tables needed (each of size 2^{10})

Colorado State University