CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2024 L16

Main Memory

Slides based on

- **Text by Silberschatz, Galvin, Gagne**
- **Various sources**

Where we were: Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
  - Deadlock Prevention
  - Deadlock Avoidance resource-allocation
  - Deadlock Detection
  - Recovery from Deadlock
- Livelock

Colorado State University

Recovery from Deadlock: Process Termination

Choices

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated

In which order should we choose to abort?

1. Priority of the process
2. How long process has computed, and how much longer to completion
3. Resources the process has used
4. Resources process needs to complete
5. How many processes will need to be terminated
6. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollbacks in cost factor

Colorado State University

Deadlock recovery through rollbacks

- **Checkpoint** process periodically
  - Contains memory image and resource state
- Deadlock detection tells us *which* resources are needed
- Process owning a needed resource
  - **Rolled back** to before it acquired needed resource
    - Work done since rolled back checkpoint discarded
  - **Assign** resource to deadlocked process

Livelocks

In a livelock two processes need each other's resource

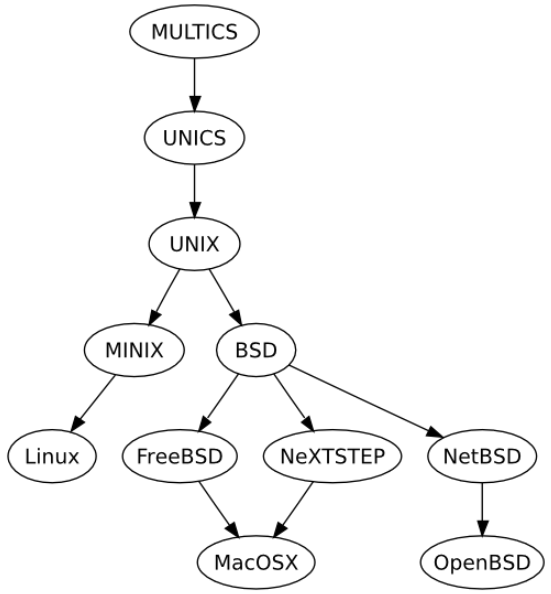
- Both run and make no progress, but neither process blocks
- Use CPU quantum over and over without making progress

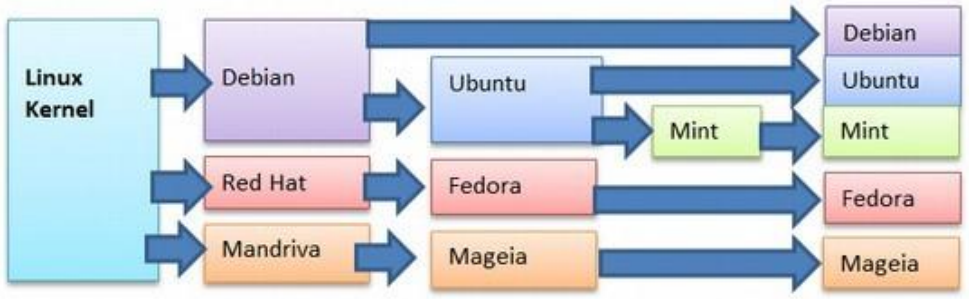
Ex: If fork fails because process table is full

- Wait for some time and try again
- But there could be a collection of processes each trying to do the same thing
- Avoided by ensuring that only one process (chosen randomly or by priority) takes action

Two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass.
But they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.

Some OS History Lessons 1: UNIX

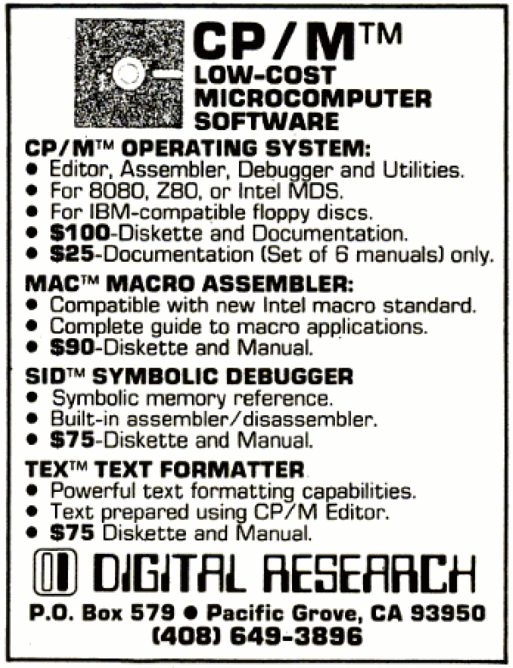




- **Unix** 1969 at AT&T's Bell Laboratories by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. Initially released in 1971 and written in assembly language, Unix was re-written in C in 1973 by Dennis Ritchie, making it more portable across platforms.
- **BSD** (Berkeley Software Distribution) is a Unix variant developed at UC Berkeley. Derivatives like FreeBSD, OpenBSD, and NetBSD have emerged from BSD. OS X (macOS) and PS4 also have roots in BSD.
- **Linux**, released by Linus Torvalds in 1991, open-source operating system. Initially built for Intel x86 PCs, Linux has since been ported to more platforms than any other OS. It is now widely used on servers, supercomputers, mobile phones (Android), and gaming consoles like the Nintendo Switch.

Some OS History Lessons 2: Windows

- 1974: CP/M Intel 8080, Gary Kildall, Digital Research
  - 8-bit, min 16 kB RAM, floppy
- 1980: 86-DOS, Intel 8086, Tim Paterson, Seattle Computer Products
  - Inspired by CP/M?
- 1981: PC DOS, Bill Gates, Microsoft
  - 86-DOS licensed for $25,000, hired Paterson
- 1985: Windows, Bill Gates, Microsoft
  - GUI inspired by MAC? Xerox PARC Star?



CP/M™
LOW-COST MICROCOMPUTER SOFTWARE

CP/M™ OPERATING SYSTEM:
- Editor, Assembler, Debugger and Utilities.
- For 8080, Z80, or Intel MDS.
- For IBM-compatible floppy discs.
- **$100**-Diskette and Documentation.
- **$25**-Documentation (Set of 6 manuals) only.

MAC™ MACRO ASSEMBLER:
- Compatible with new Intel macro standard.
- Complete guide to macro applications.
- **$90**-Diskette and Manual.

SID™ SYMBOLIC DEBUGGER
- Symbolic memory reference.
- Built-in assembler/disassembler.
- **$75**-Diskette and Manual.

TEX™ TEXT FORMATTER
- Powerful text formatting capabilities.
- Text prepared using CP/M Editor.
- **$75** Diskette and Manual.

DIGITAL RESEARCH
P.O. Box 579 • Pacific Grove, CA 93950
(408) 649-3896

Gary Kildall net worth $1.9 Million at death
Tim Paterson Net Worth: $250,000

Perspective

- First half: Processes and Threads
  - Creation and termination
  - Interactions
    - Communications
    - Synchronization
    - Deadlocks
- Second half: Where is stuff? How are processes implemented? Isolated?
  - Main memory
  - Storage
  - File systems
  - Containers and virtual machines

Help Session this Thursday
HW4 Due Oct 23 Wed

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2024

Main Memory

Slides based on
- Text by Silberschatz, Galvin, Gagne
- Various sources

Chapter 8: Main Memory

Objectives:

- Organizing memory for multiprogramming environment
  - Partitioned vs separate address spaces
- Memory-management techniques
  - Virtual vs physical addresses
  - Chunks
    - segmentation
    - Paging: page tables, caching (“TLBs”)
- Examples: the Intel (old/new) and ARM architectures

What we want

- Memory capacities have been increasing
  - But programs are getting bigger faster
  - Parkinson's Law*: Programs expand to fill the memory available to hold
- What we would like
  - Memory that is
    - infinitely large, infinitely fast
    - Non-volatile
    - Inexpensive too
- Unfortunately, no such memory exists as of now

*work expands so as to fill the time available for its completion. 1955

How much memory is enough?

"My fortune says 'you can't be too thin, or too rich, or have too much computer memory'."

Colorado State University

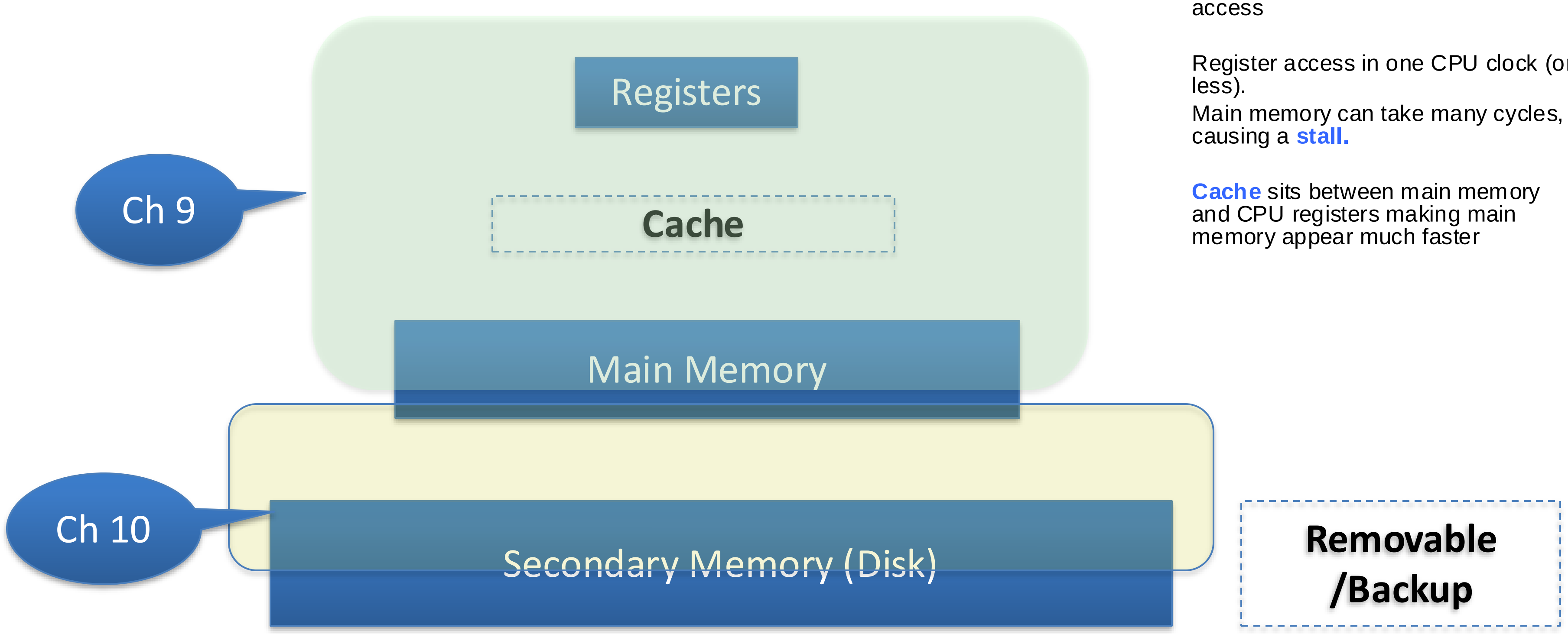
Background

- Program must be brought (from disk) into memory and run as a process
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of
  - addresses + read requests, or
  - address + data and write requests
- n-bit address: address space of size 2^n bytes.
  - Ex: 32 bits: addresses 0 to (2^{32} -1) bytes
  - Addressable unit is always 1 byte.
- Access times:
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles, causing a **stall**
  - **Cache** sits between main memory and CPU registers making main memory appear much faster
- **Protection** of memory required to ensure correct operation

2^{10}=1,024 ≈ K
2^{20} = 1,048,576 ≈ M
2^{30} ≈ G

Colorado State University

Hierarchy



Main memory and registers are only storage CPU can access directly access

Register access in one CPU clock (or less).
Main memory can take many cycles, causing a **stall.**

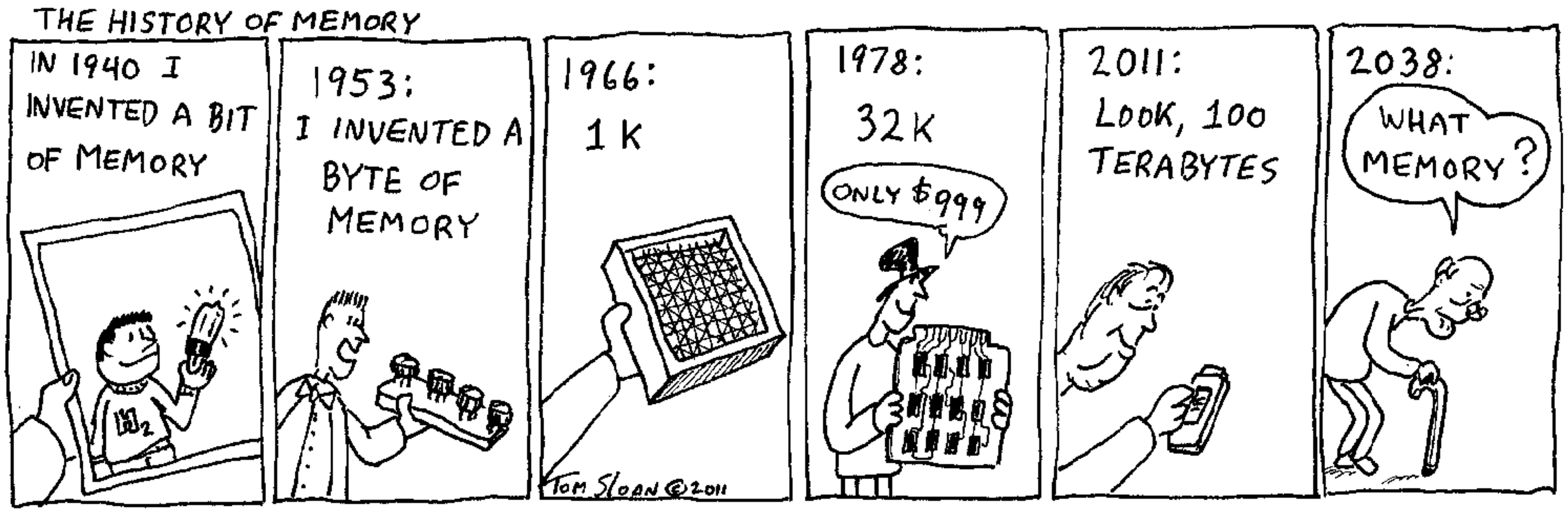
Cache sits between main memory and CPU registers making main memory appear much faster

Ch 11,13,14,16: Disk, file system   Cache: CS470

Colorado State University

Memory Technology

somewhat inaccurte


THE HISTORY OF MEMORY
IN 1940 I INVENTED A BIT OF MEMORY
1953: I INVENTED A BYTE OF MEMORY
1966: 1 K
TOM SLOAN ©2011
1978: 32 K
ONLY $999
2011: LOOK, 100 TERABYTES
2038:
WHAT MEMORY?

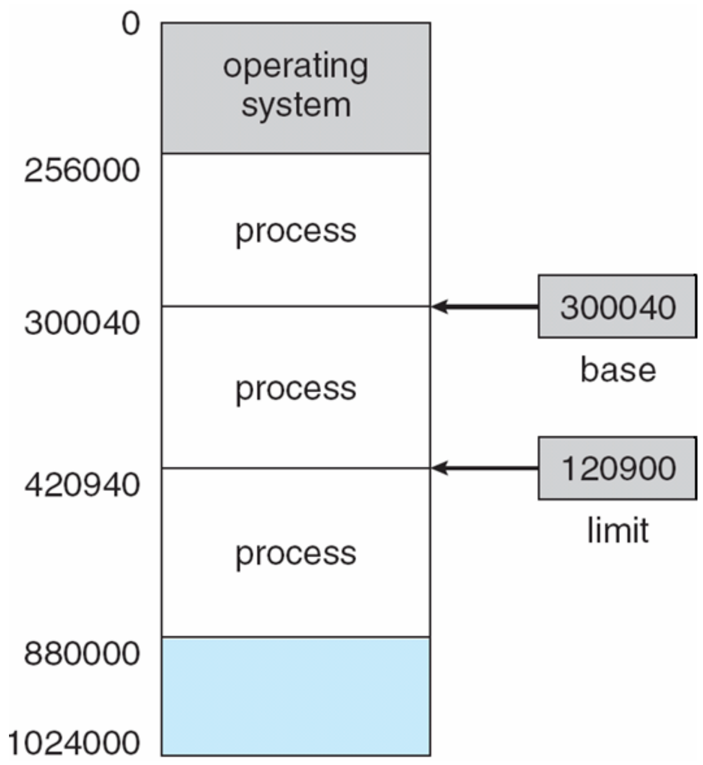
Colorado State University

Protection: Making sure each process has separate memory spaces

- OS must be protected from accesses by user processes
- User processes must be protected from one another
  - Determine range of legal addresses for each process
  - Ensure that process can access only those
- Approaches:
  - **Partitioning address space** (early system)
  - **Separate address spaces** (modern practice)

Partitioning Address Space: Base & Limit Registers

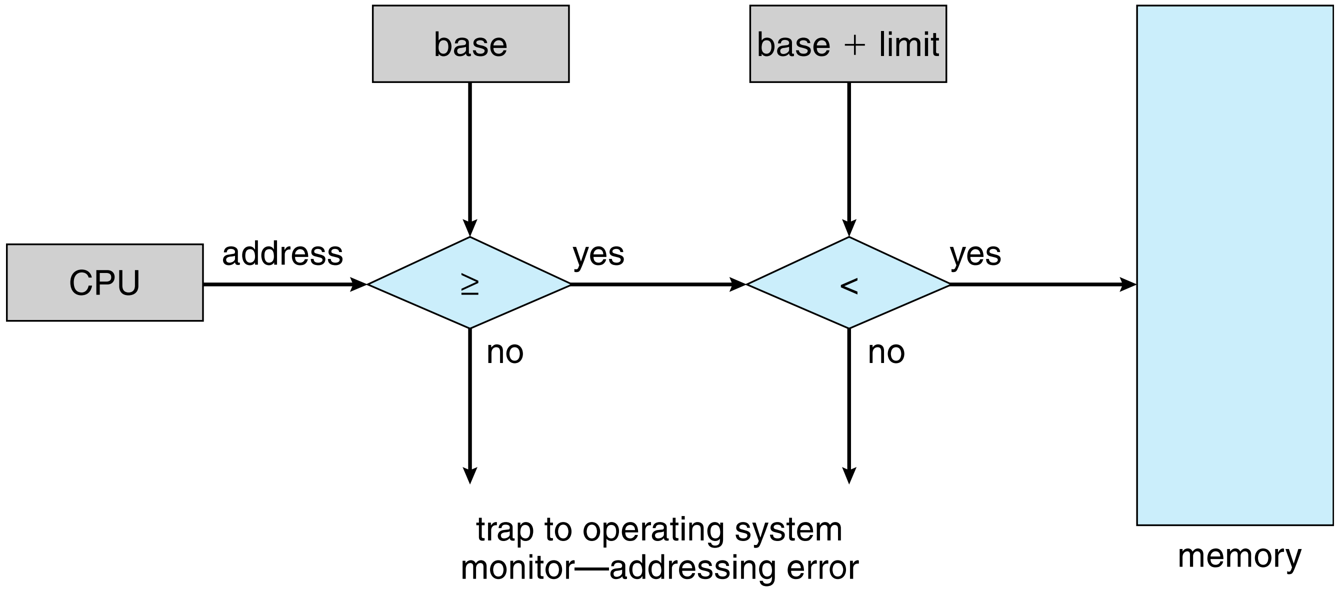
- Base and Limit for a process
  - **Base**: Smallest legal physical address
  - **Limit**: Size of the range of physical address
- A pair of **base** and **limit registers** define the logical address space for a process
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user
- Base: **Smallest** legal physical address
- Limit: Size of the **range** of physical address
- Eg: Base = 300040 and limit = 120900
- Legal: 300040 to (300040 + 120900 -1) = 420939



Addresses: decimal, hex/binary

Colorado State University

Hardware Address Protection



Legal addresses: **Base address** to **Base address + limit -1**

Colorado State University

Separate Address Spaces Modern

- Each process has its own private address space.
  - **Logical address space** is the set of all logical addresses used by a process.
- However, the physical memory has just one address space.
  - **Physical address space** is the set of all physical addresses
- Need to map one to the other.

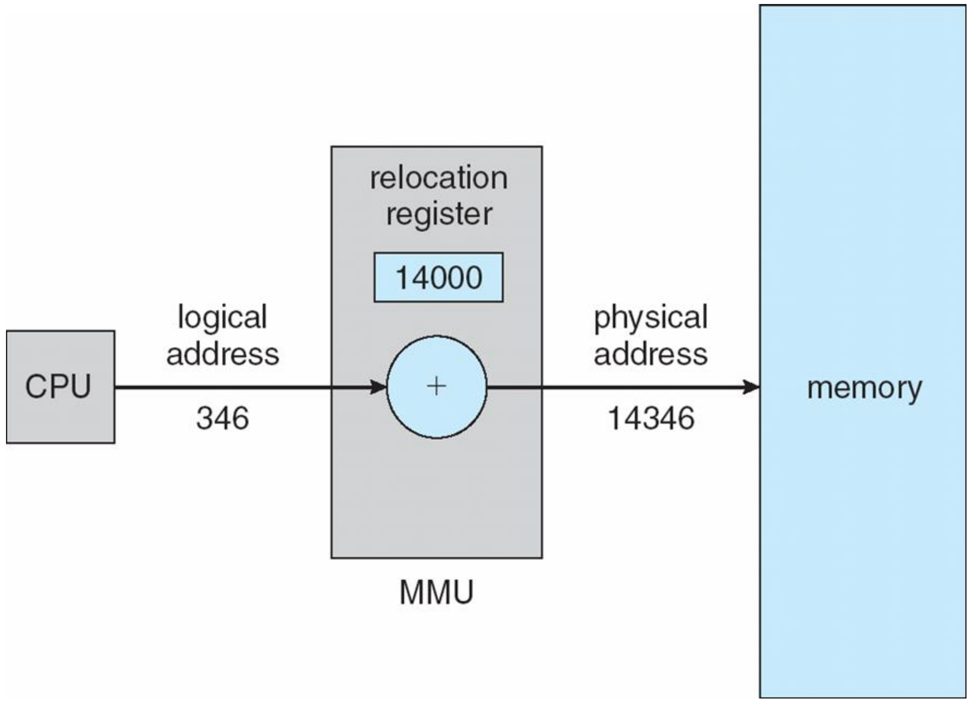
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses

Memory-Management Unit (MMU)

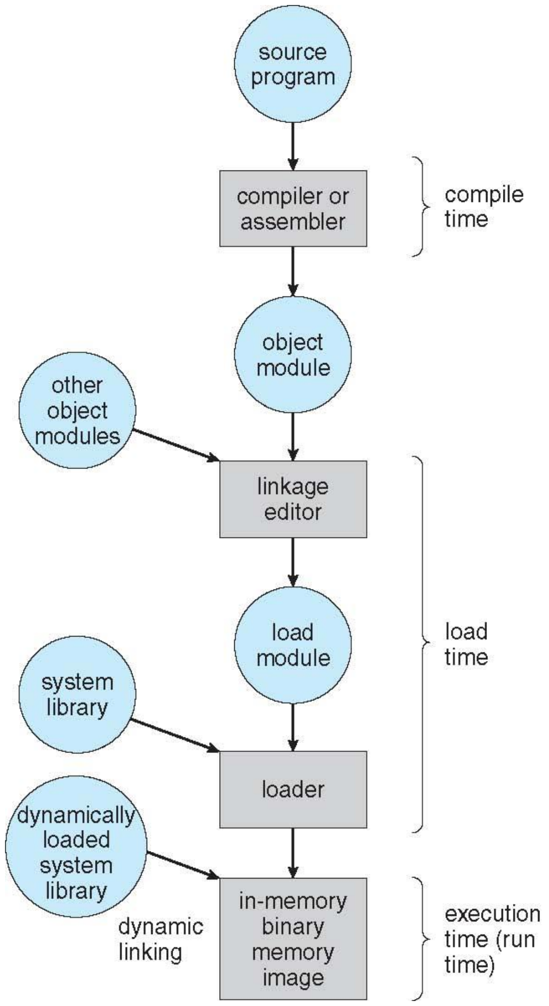
- Hardware device that at run time maps virtual to physical address
  - Many methods possible, we will see them soon
- Consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Dynamic relocation using a relocation register



Colorado State University

Multistep Processing of a User Program



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

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

Colorado State University

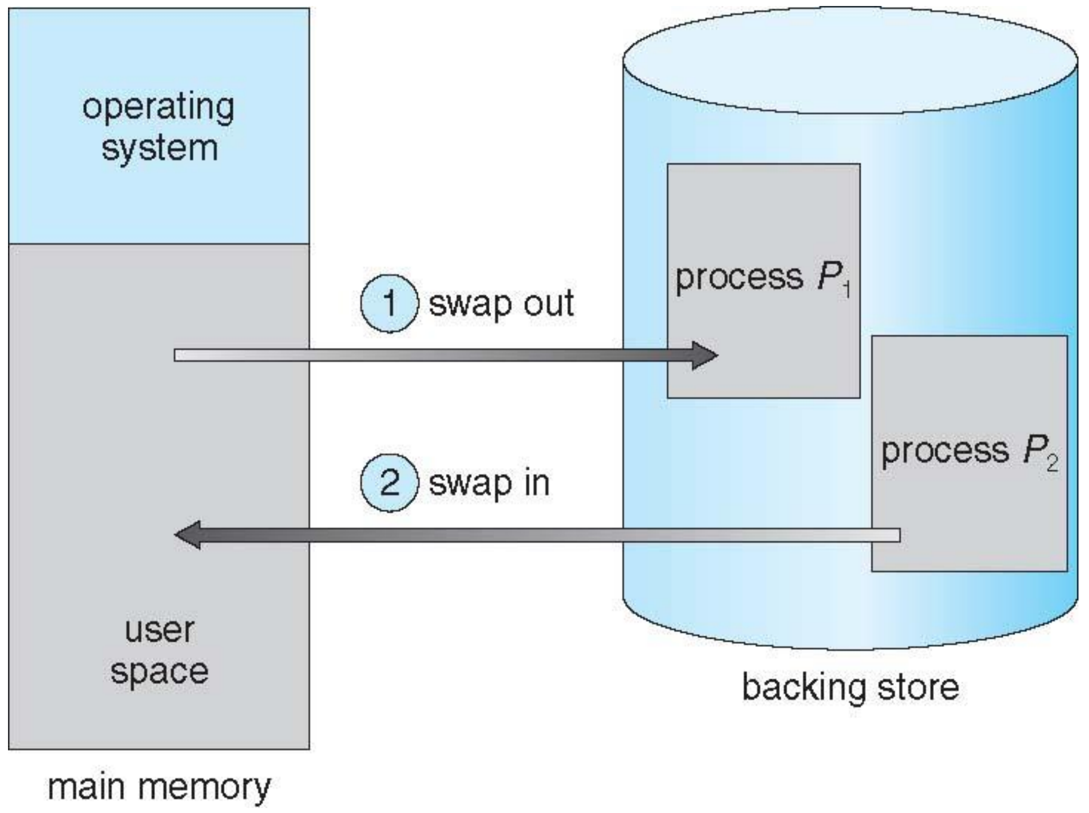
Dynamic loading of routines

- 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)

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

Colorado State University

Memory Allocation

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.

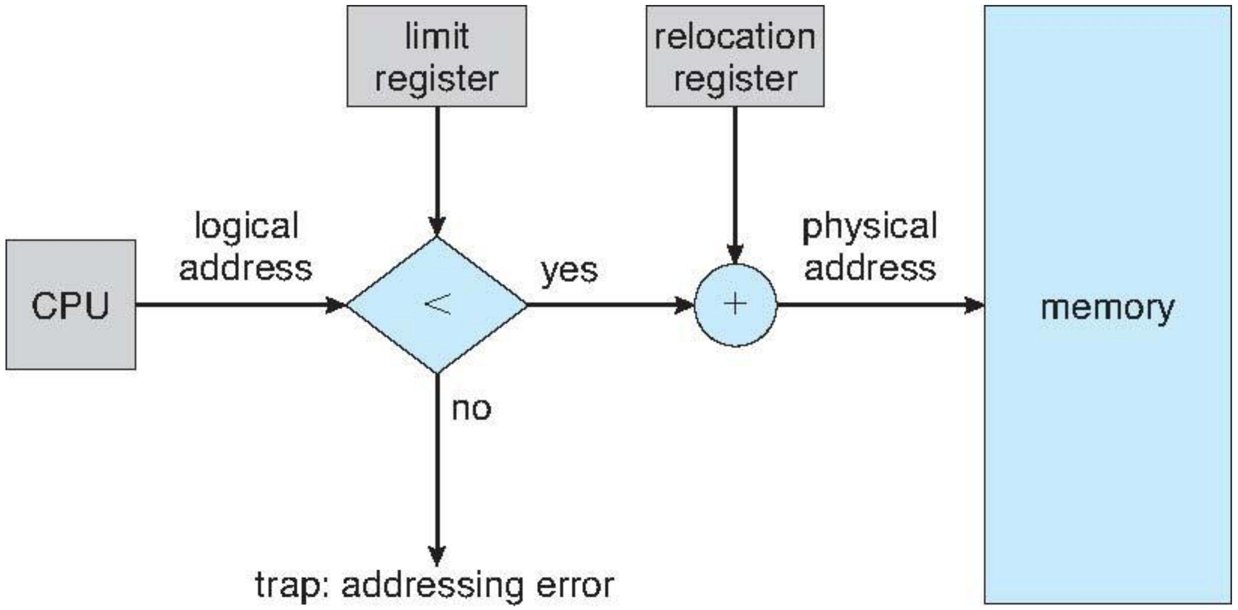
Contiguous Allocation

- 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



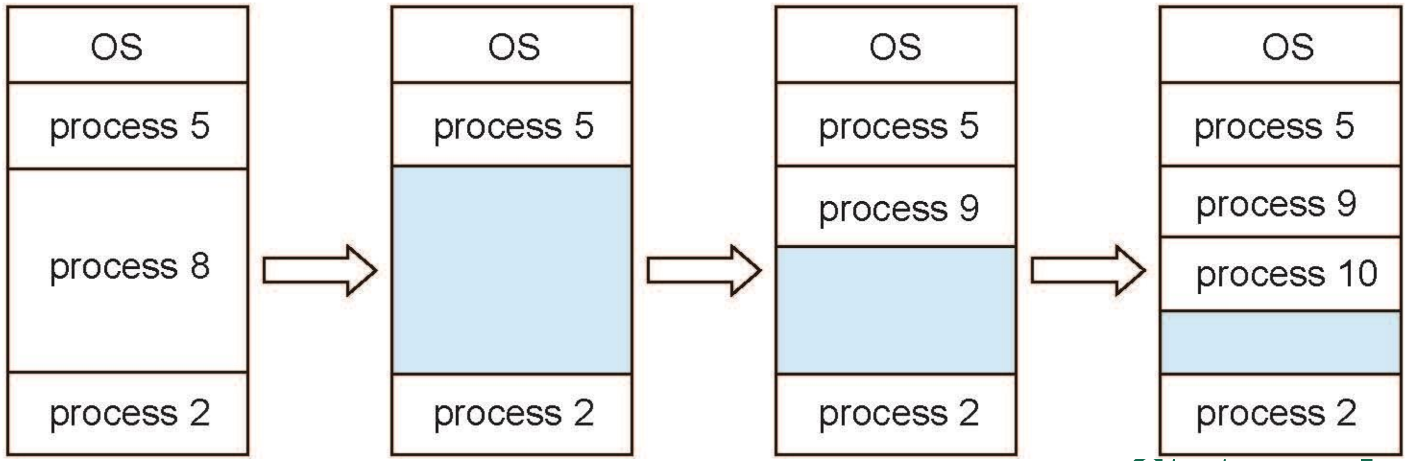
MMU maps logical address *dynamically*

Physical address = relocation reg + valid logical address

Colorado State University

Multiple-partition 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)



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

Colorado State University

Paging vs Segmentations

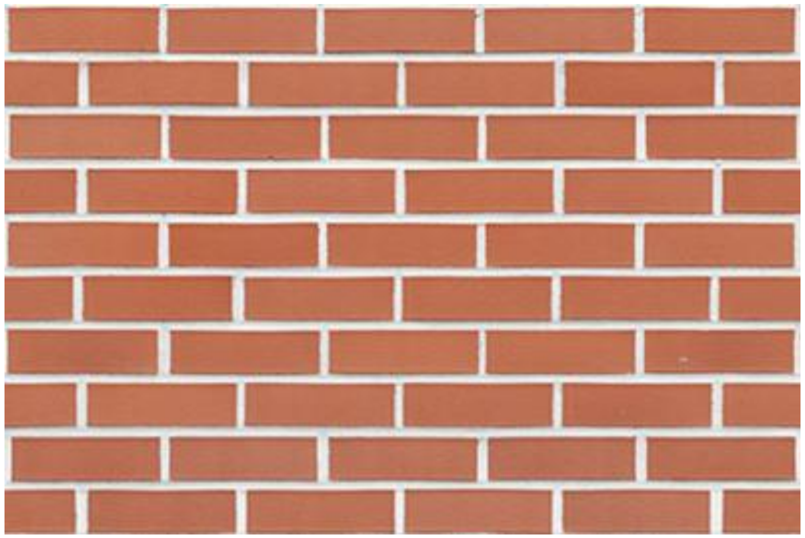
Segmentation: program divided into logically divided “segments” such as main program, function, stack etc.

- 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



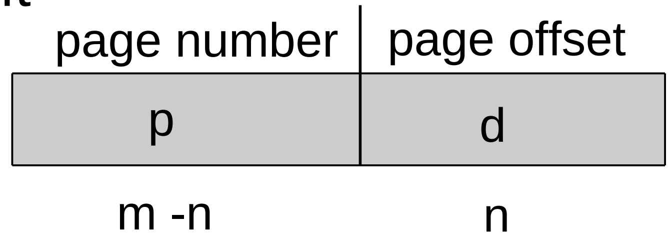
Colorado State University

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

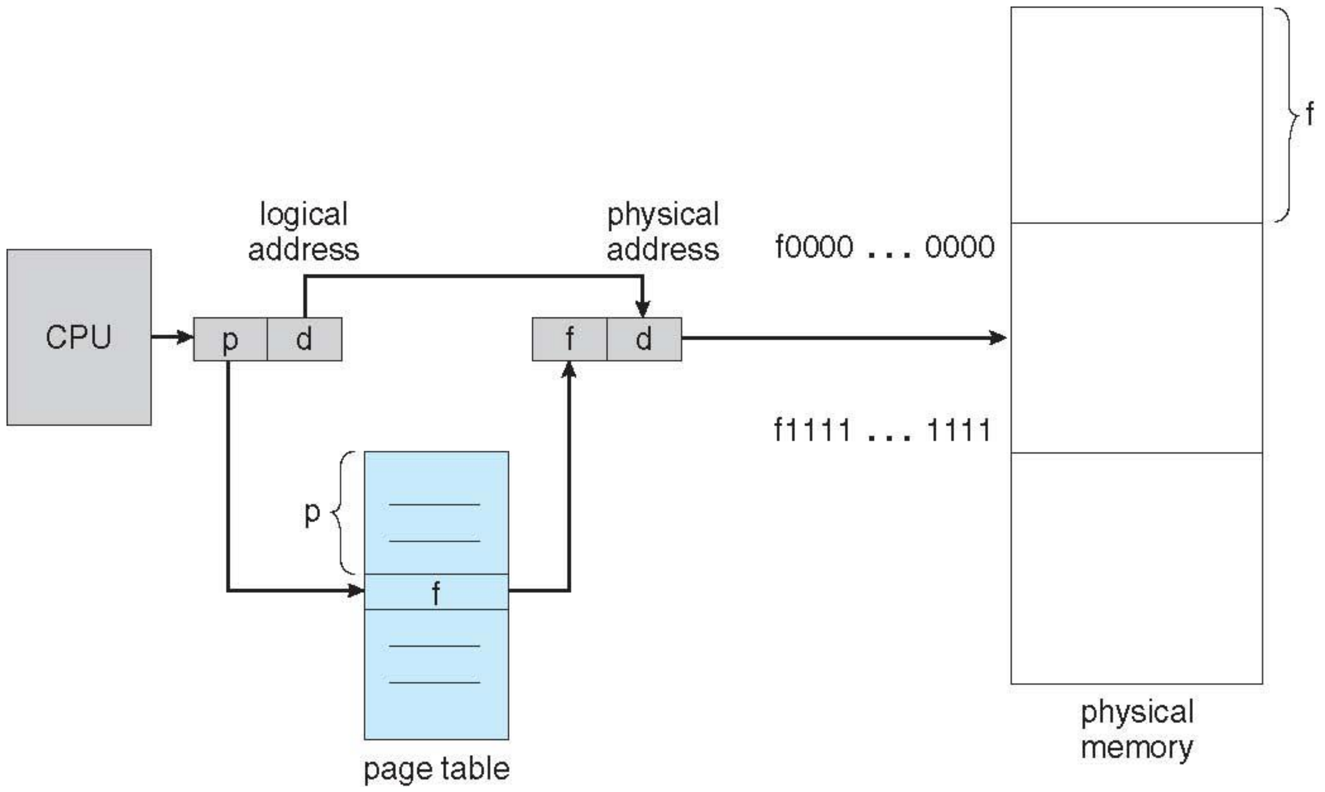
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



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

Colorado State University

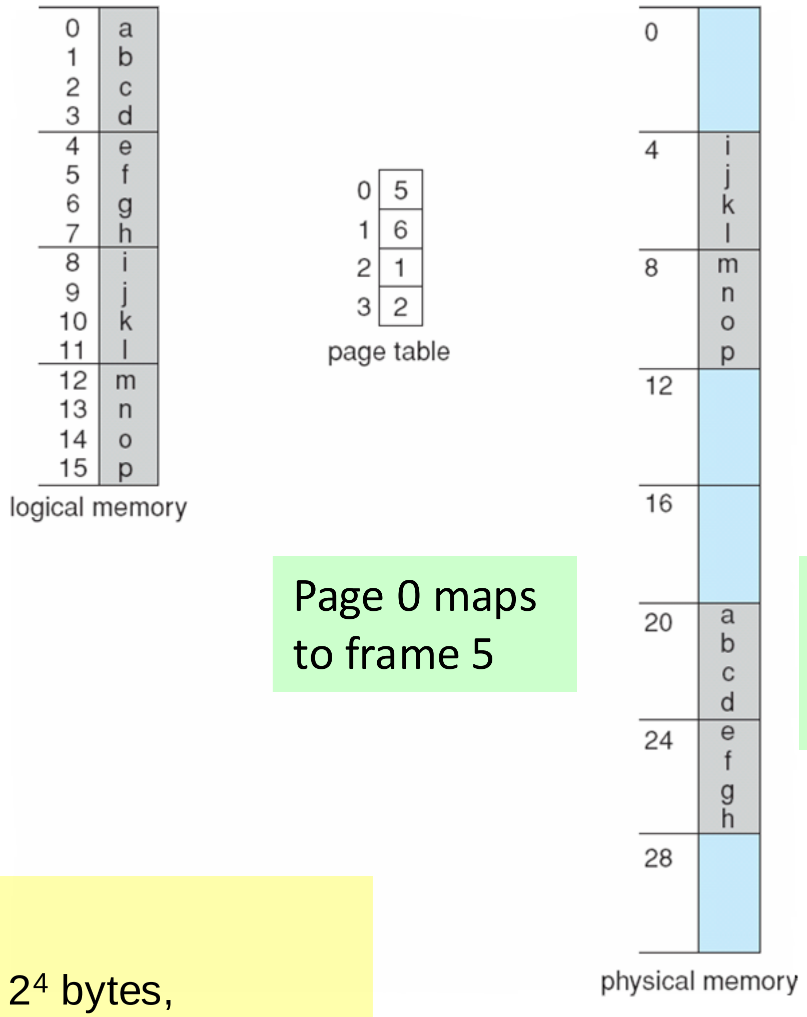
Paging Hardware



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

Colorado State University

Paging Example



8 frames
Frame number 0-to-7

Page 0 maps to frame 5

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