CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

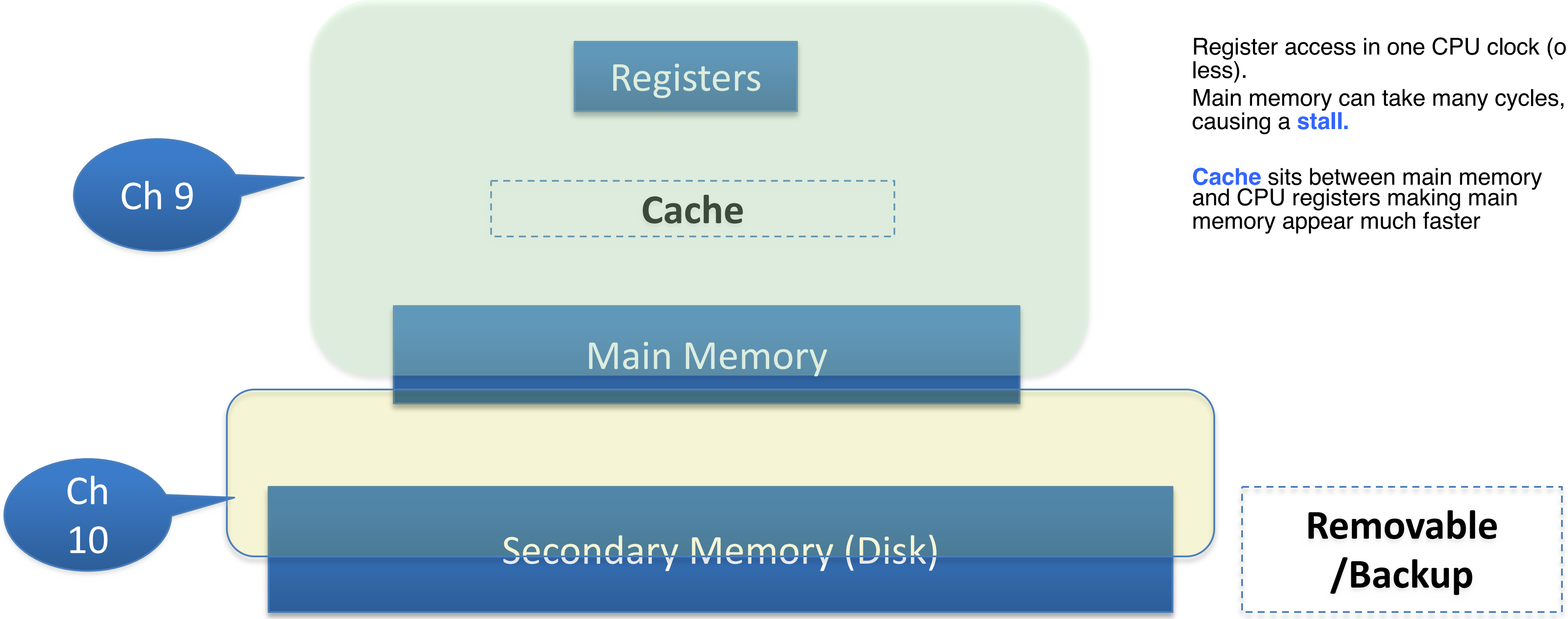
Spring 2022 L17

Main Memory

Slides based on

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

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, 13,14,15: Disk, file system   Cache Memory: CS470

Colorado State University

FAQ

- Why partition the address space/Why have multiple separate address spaces
  - Multiprogramming
  - Each process needs separate memory
- Why do processes/kernel need protection (separation)
  - Users can be careless/malicious.
- Why have logical (virtual) and physical addresses?

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 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, function, 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.

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**

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

Pages: Outlines

- 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

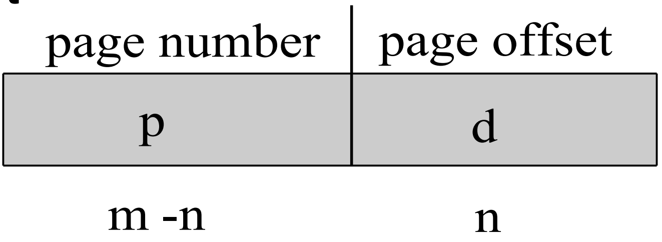
Page refers to a block of information, frame refers to a physical memory block. Frame is sometimes called a page frame or just a page.

Paging

- Divide physical memory into fixed-sized blocks called **frames** **(or page frames)**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
  - To run a program of size ***N*** pages, need to find ***N*** free frames and load program
  - Still have Internal fragmentation
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks

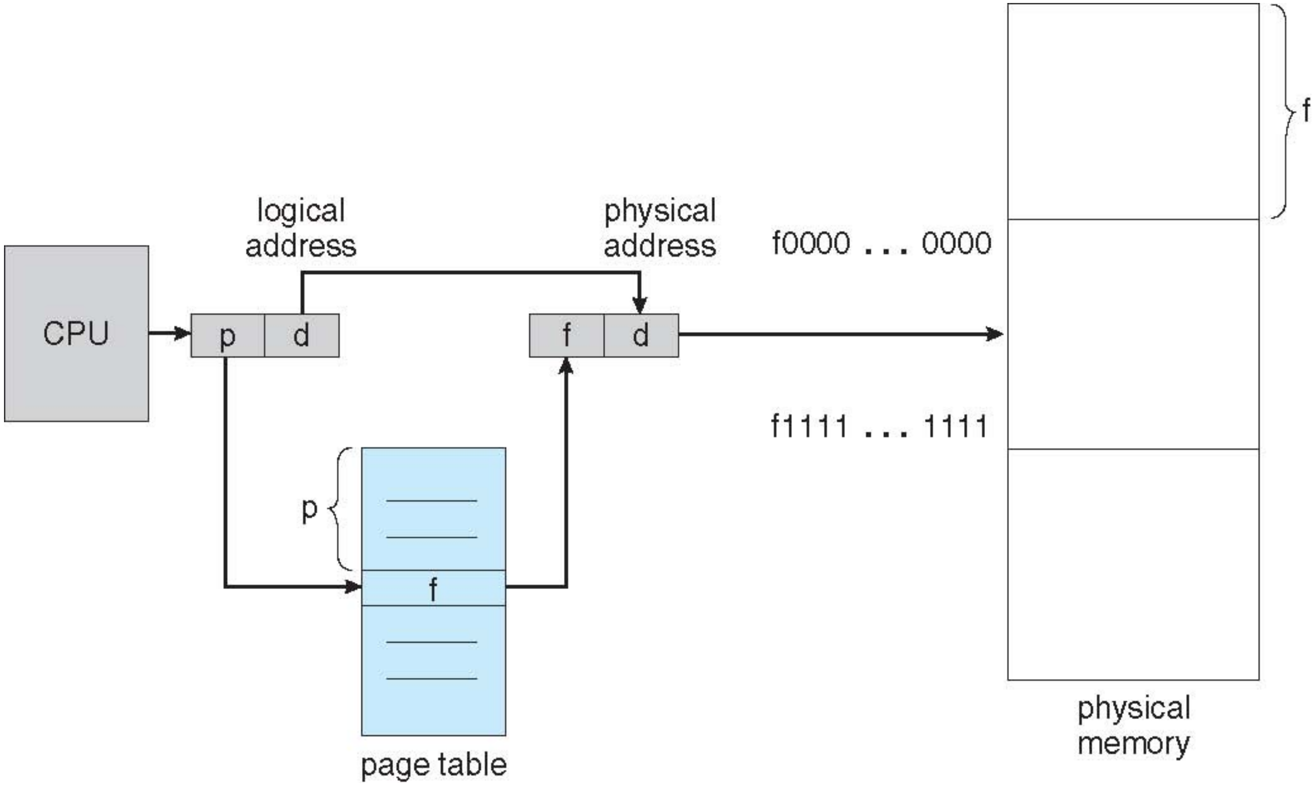
Address Translation Scheme

- 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

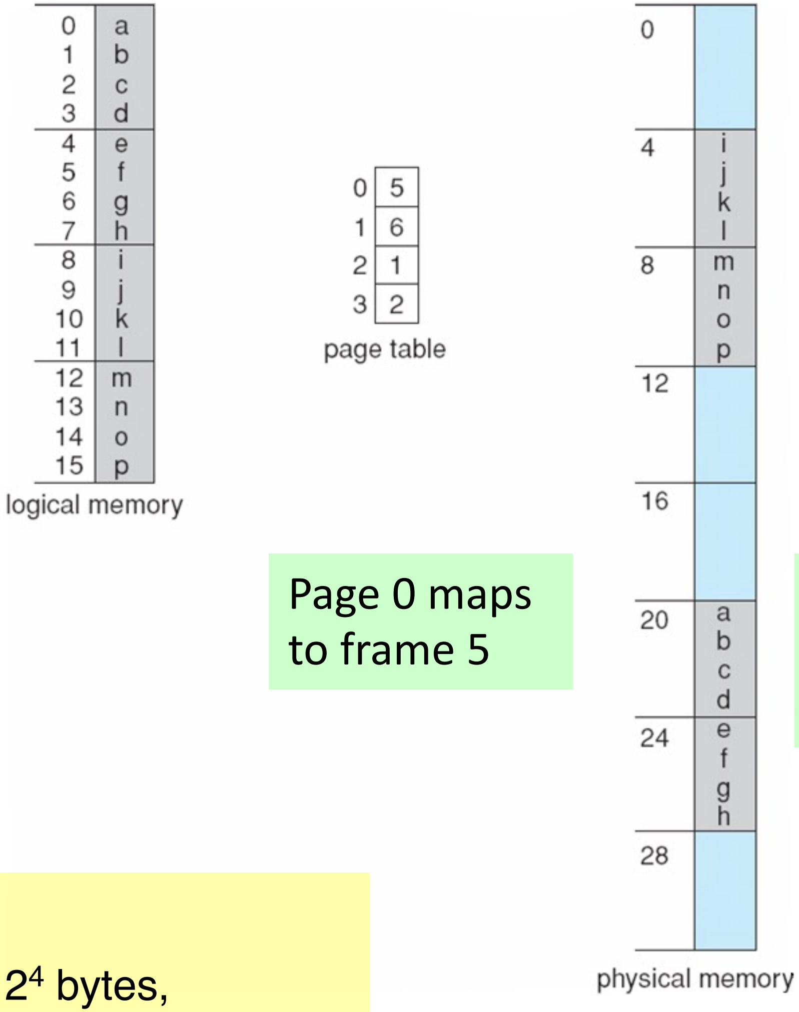
Paging Hardware



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)

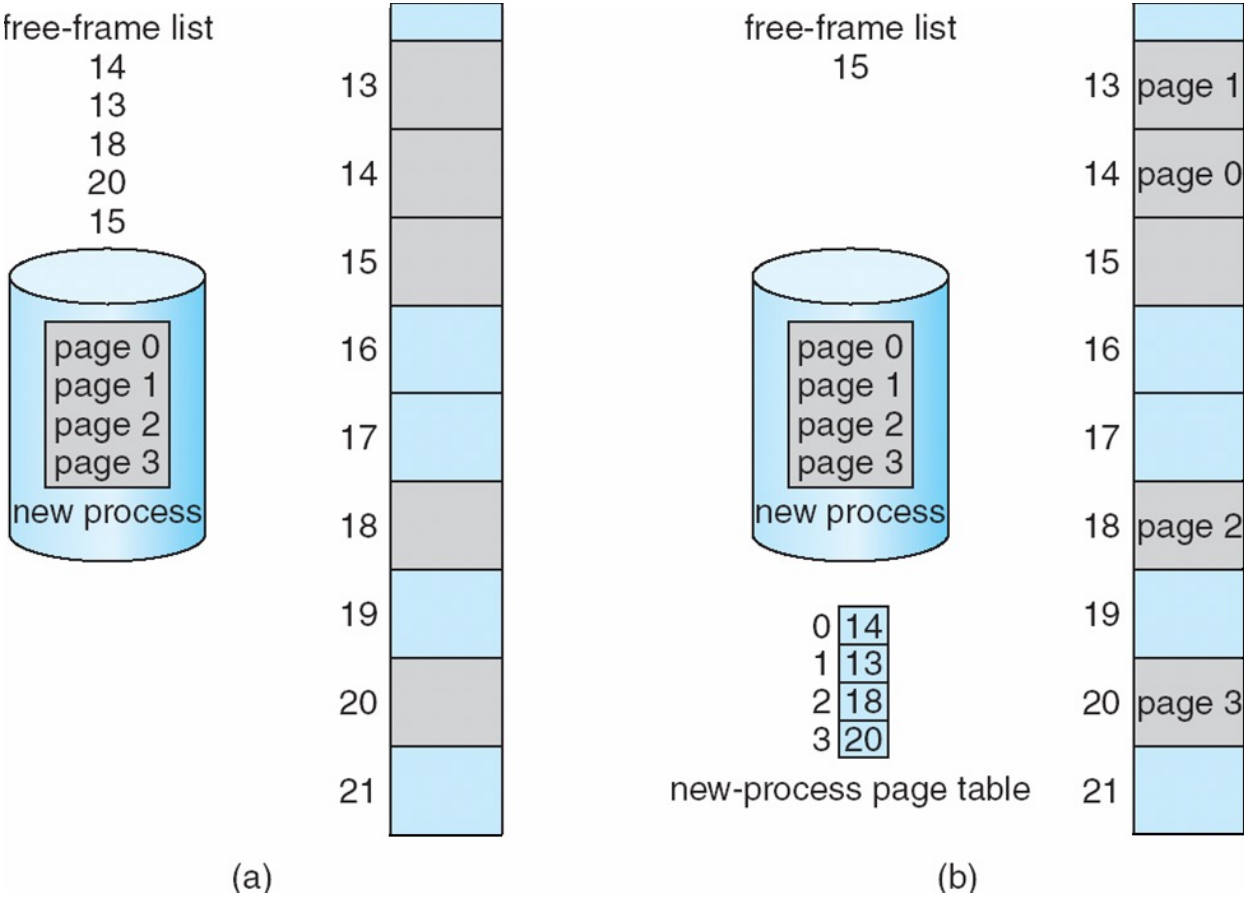
Ex: m=4   and   *n*=2

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

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 ..

Free Frame allocation



Before allocation

After allocation

A new process arrives

That needs four pages

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

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

Colorado State University

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

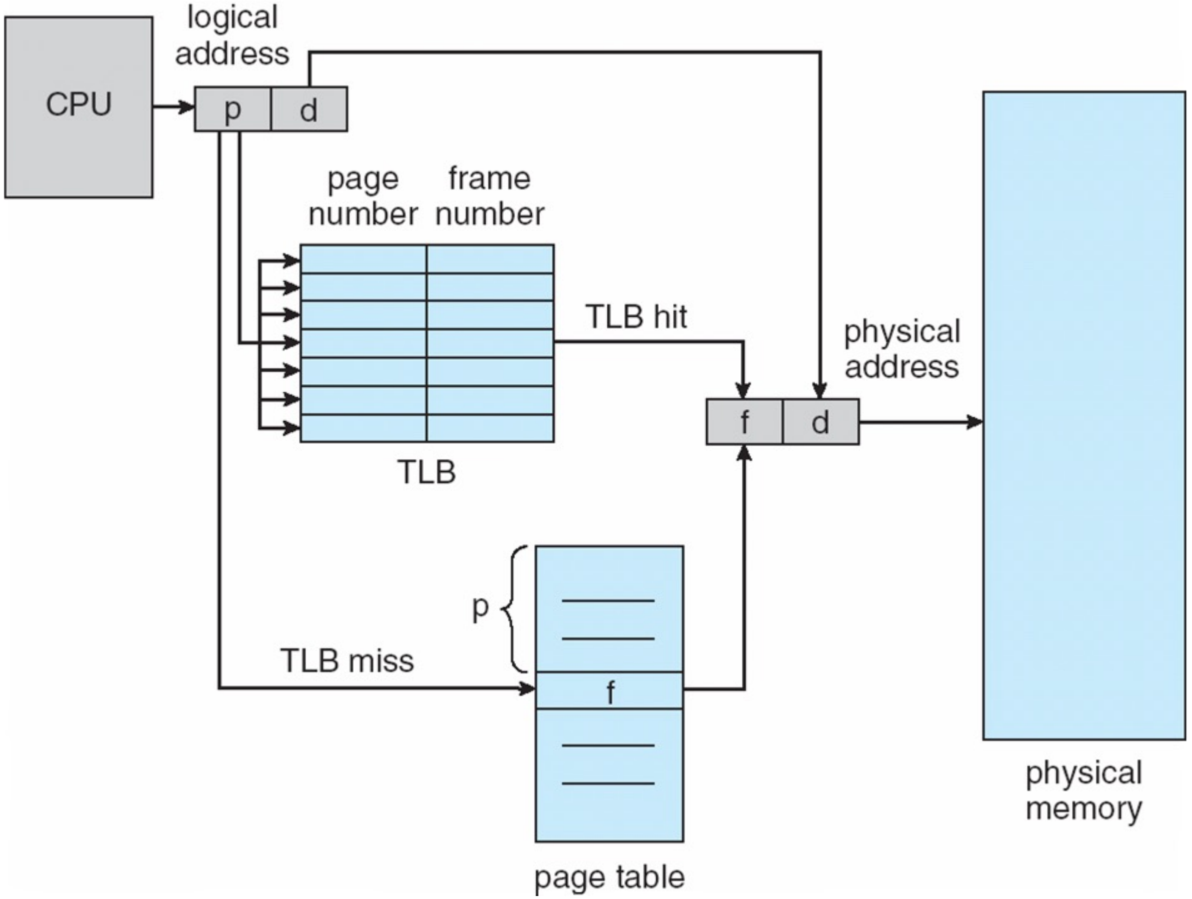
- Associative memory –parallel search using hardware
  - "Content addressable memory": Electronics is very expensive

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

Effective Access Time

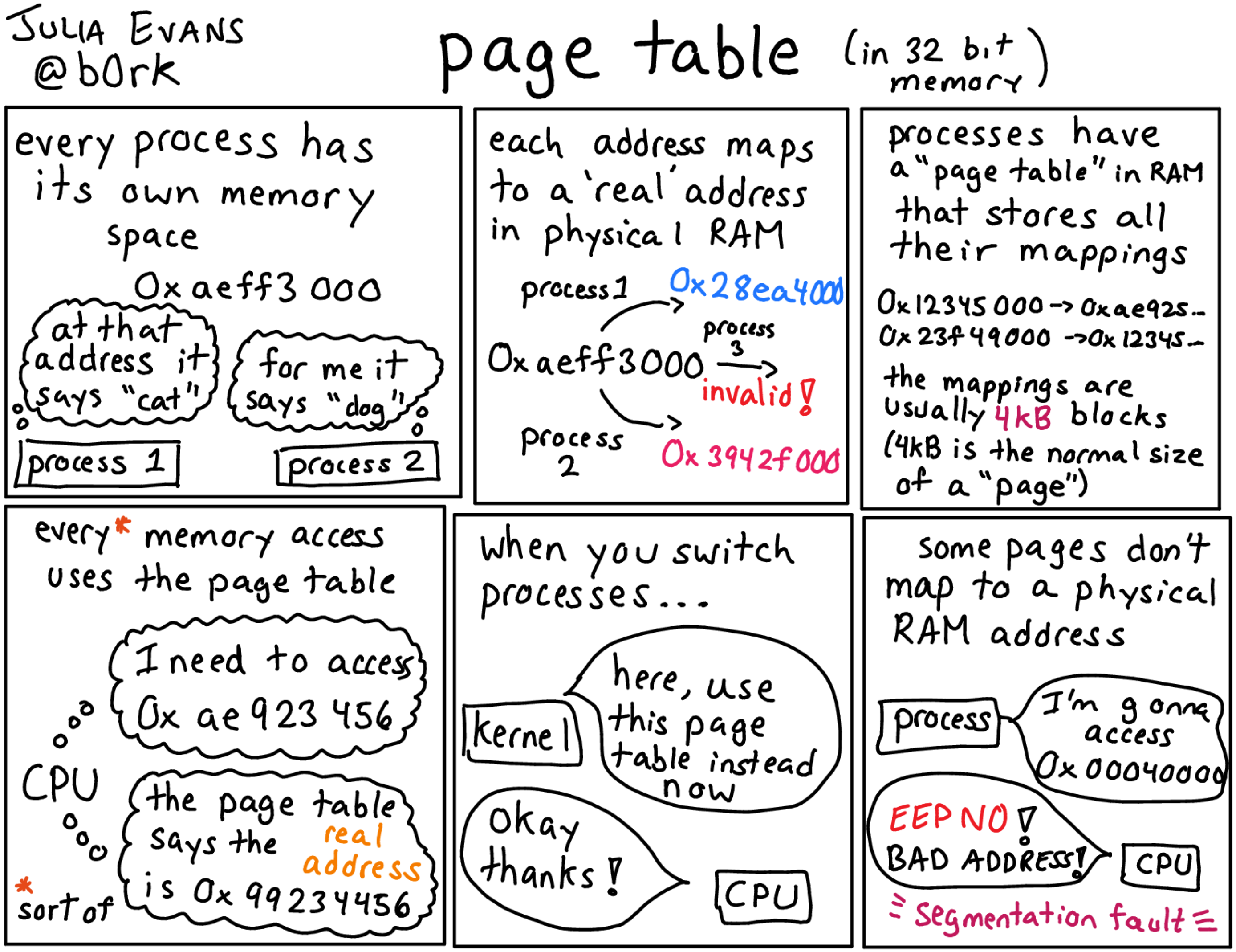
On average how long does a memory access take?

- 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 $\alpha = 90\%$, 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 -> $\alpha = 99\%$,
  - 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

each address maps to a 'real' address in physical RAM

processes have a "page table" in RAM that stores all their mappings

0x12345 000 -> 0x ae925..
0x 23f49000 -> 0x 12345..

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 ae923 456
CPU
the page table says the real address is 0x 99234456

* sort of

when you switch processes...

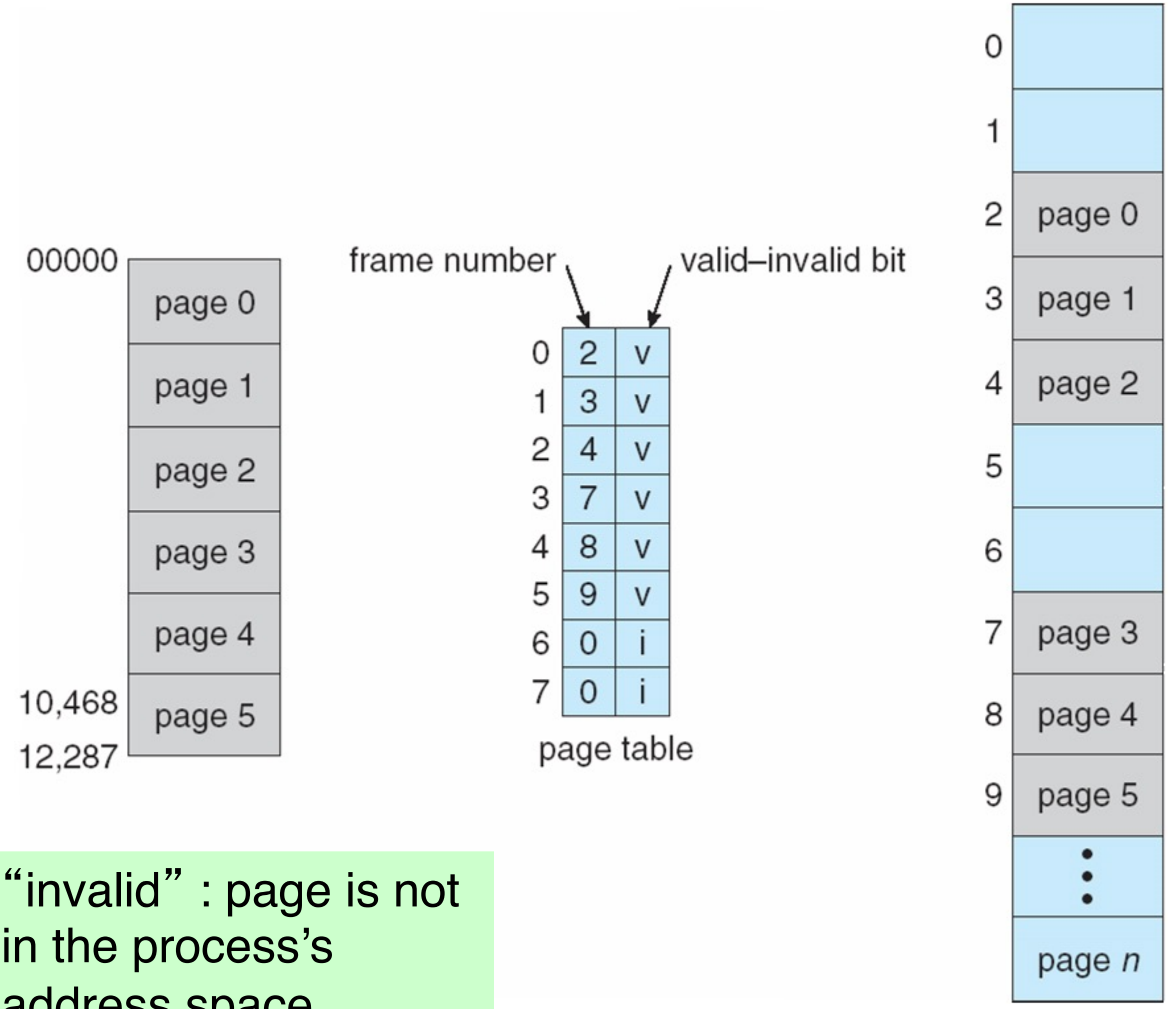
some pages don't map to a physical RAM address

≡ 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



	frame number	valid–invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

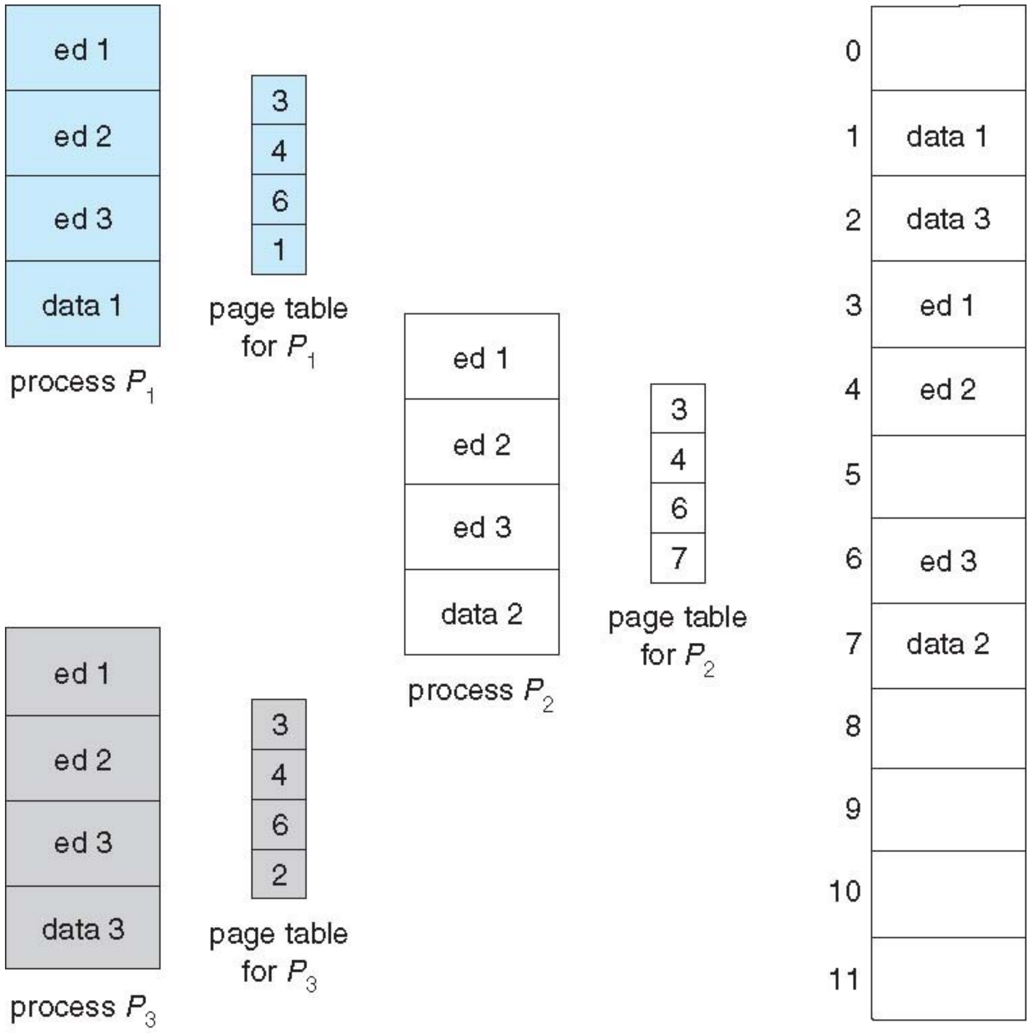
page table

“invalid” : page is not in the process’s address space.

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:

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

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

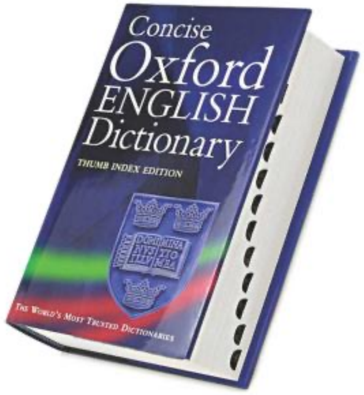
Issues with large page tables

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

Hierarchical Page Tables

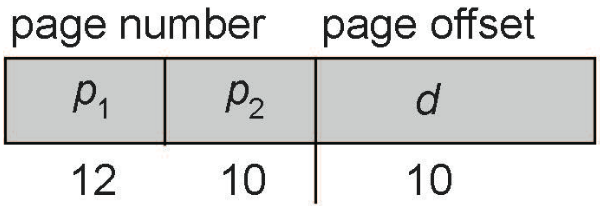
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

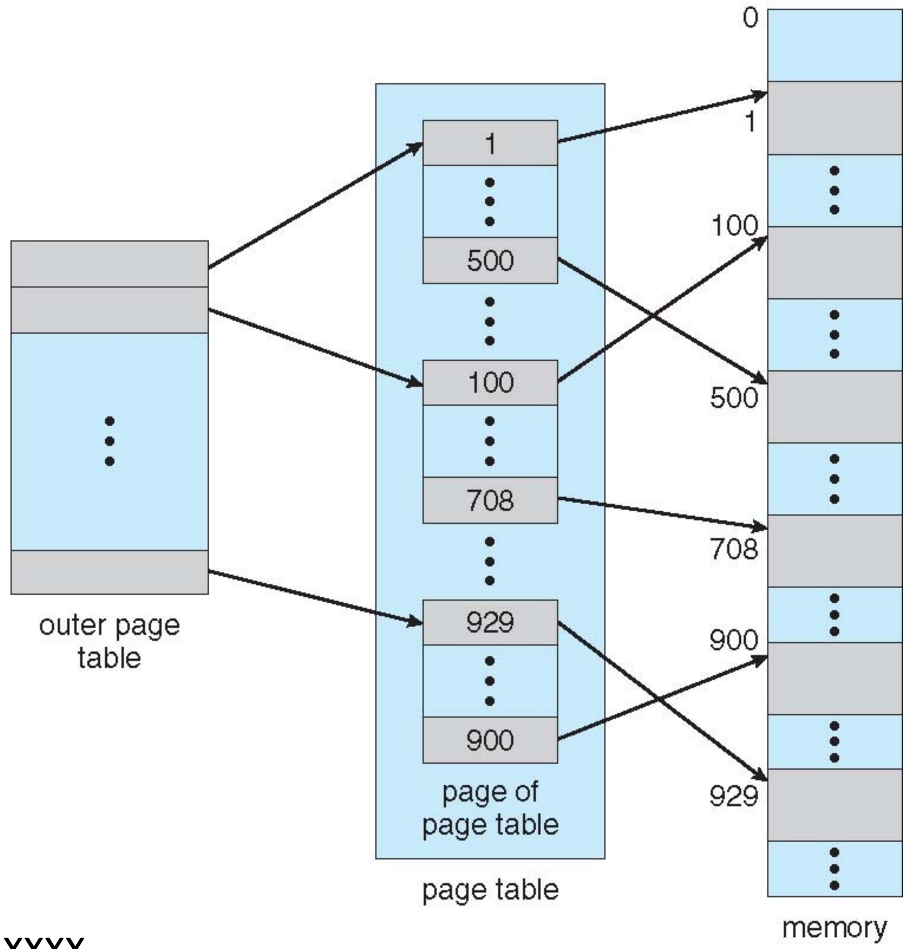
page number		page offset
p_1	p_2	d
12	10	10



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

Two-Level Page-Table Scheme



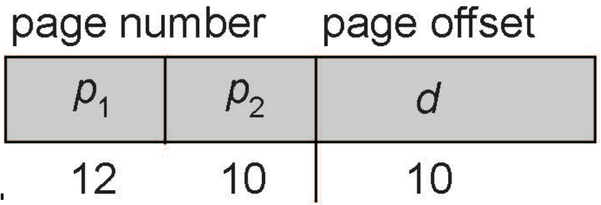


xxxx xxxx xxxx xxxx xxxx xx xx xxxx xxxx

Outer Page table page table offset within page

Two-Level Paging Example

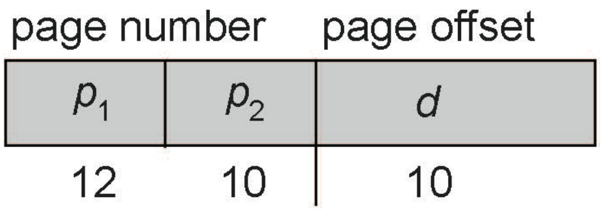
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address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

Two-Level Paging Example

- A logical address is as follows:



page number page offset

p_1 p_2 d

12 10 10

- 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})

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

ICQ

Page Offset

Q1. Consider a logical address with a page size of 4 KB. How many bits must be used to represent the page offset in the logical address?

A. 16
B. 10
C. 8
D. 12

Page number

Q2. Given the logical address 0xAEF9 (in hexadecimal) with a page size of 256 bytes, what is the page number?

A. 0xF9
B. x00F9
C. xA
D. 0xAE

Each hex digit represents 4 bits

External fragmentation

Q3. In paging-based memory allocations, the physical memory is subject to external fragmentation.

A. True

B. False

Answers

Page Offset

Q1. Consider a logical address with a page size of 4 KB. How many bits must be used to represent the page offset in the logical address?

A. 16
B. 10
C. 8
D. 12 since $2^{12} = 4K$

Page number

Q2. Given the logical address 0xAEF9 (in hexadecimal) with a page size of 256 bytes, what is the page number?

A. 0xF9
B. x00F9
C. xA
D. 0xAE 2^8 = 256. Thus 8 LSBs or 2 hex digits are used for page offset

External fragmentation

Q3. In paging based memory allocations, the physical memory is subject to external fragmentation.

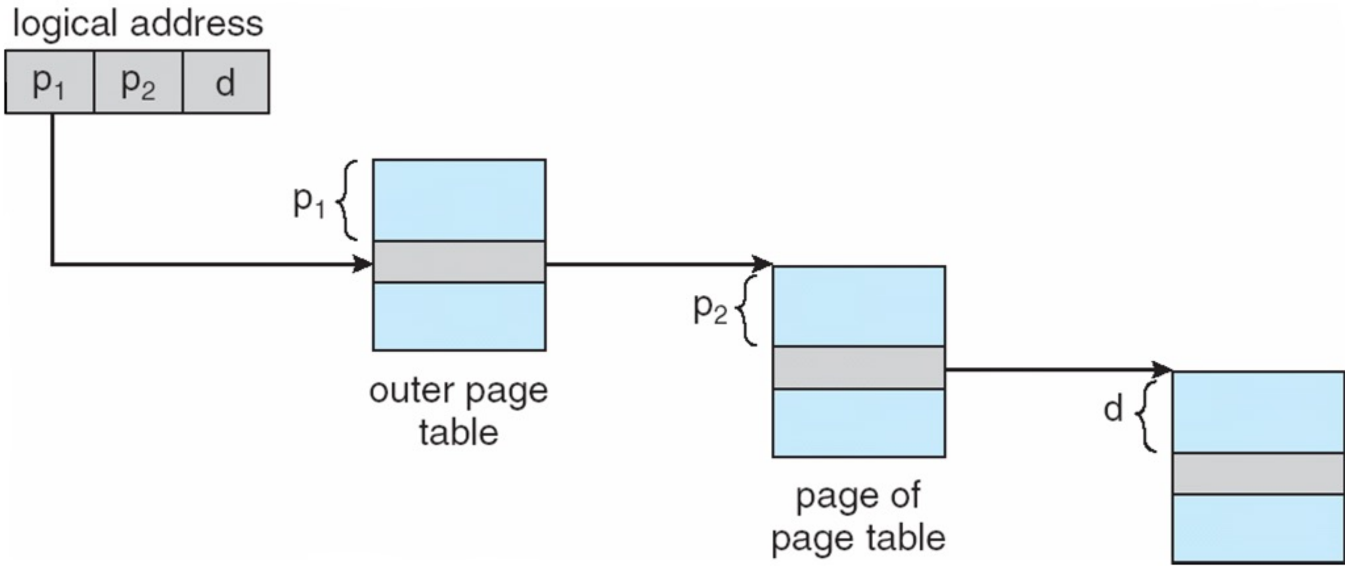
A. True

B. False Only internal fragmentation in page-based

CS370 Operating Systems

Colorado State University
Yashwant K Malaiya
Back from ICQ

Hierarchical Paging

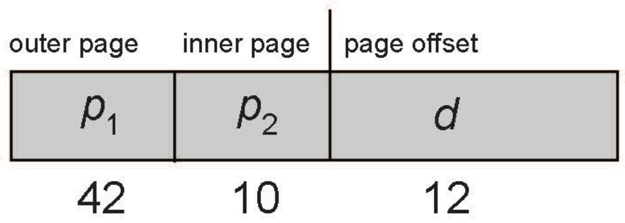


If there is a hit in the TLB (say 95% of the time), then average access time will be close to slightly more than one memory access time.

Colorado State University

64-bit Logical Address Space

- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like

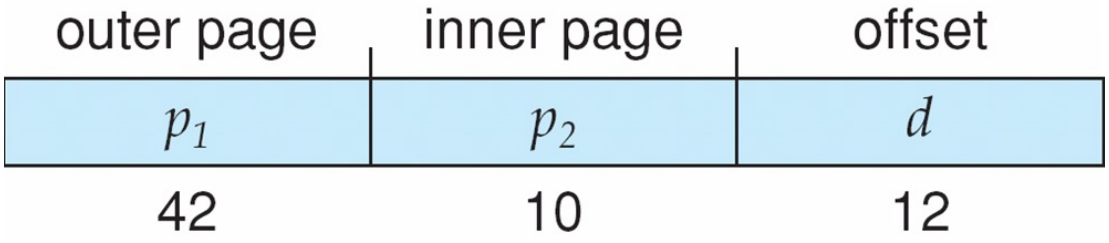


outer page	inner page	page offset
p_1	p_2	d
42	10	12

  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2nd outer page table
    - But in the following example the 2nd outer page table is still 2^{34} bytes in size
      - And possibly 4 memory access to get to one physical memory location!

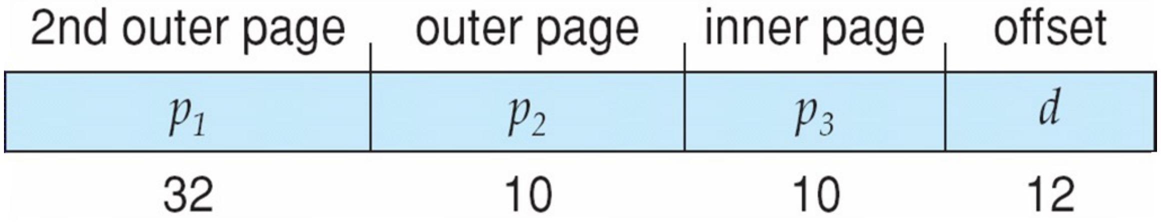
Full 64-bit physical memories not common yet

Three-level Paging Scheme



outer page	inner page	offset
p_1	p_2	d
42	10	12

- Outer page table has 2^{42} entries!
- Divide the outer page table into 2 levels
  - 4 memory accesses!

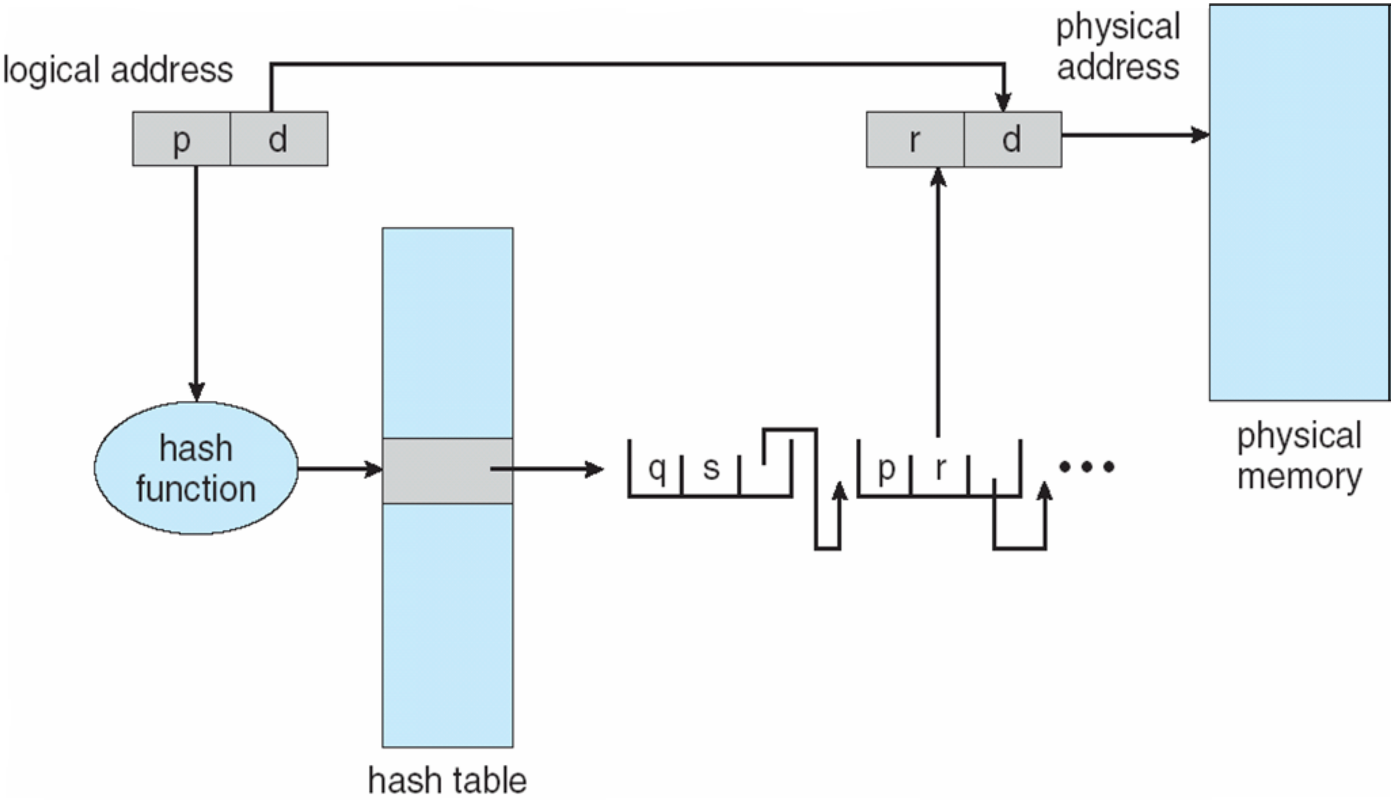


2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables**
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)

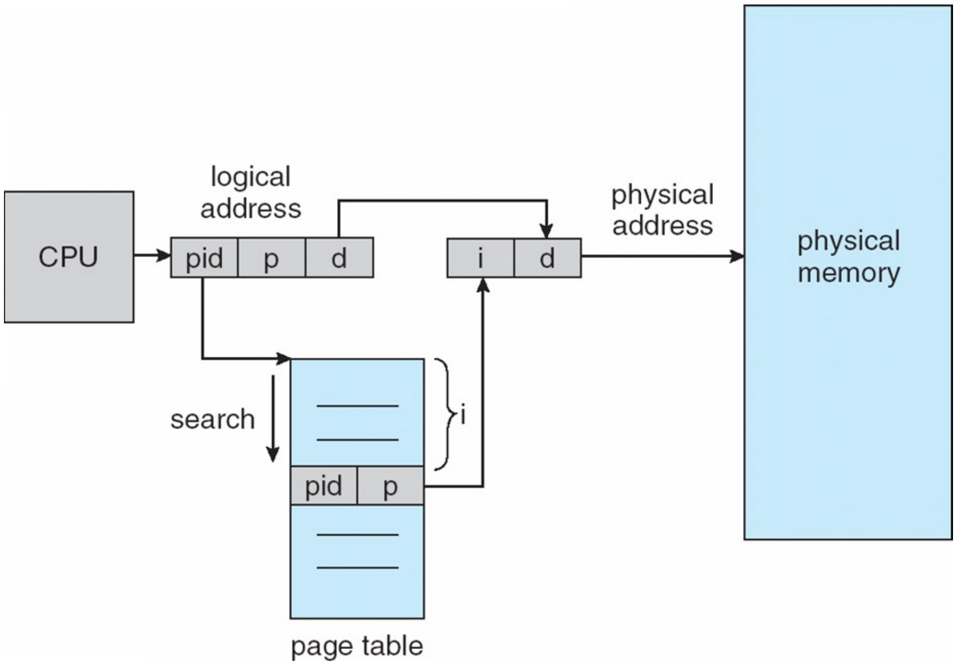
Hashed Page Table



This page table contains a chain of elements hashing to the same location.
Each element contains (1) the virtual page number (2) the value of the mapped page frame
(3) a pointer to the next element

Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
  - One entry for each real page of memory ("frame")
  - Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page



Search for pid, p, offset i is the physical frame address
Note: multiple processes in memory

Colorado State University

Inverted Page Table

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address. Not possible.

Used in IA-64 ..

Colorado State University

Segmentation Approach

Memory-management scheme that supports user view of memory

- A program is a collection of segments
  - A segment is a logical unit such as:

    main program

    procedure, function, method

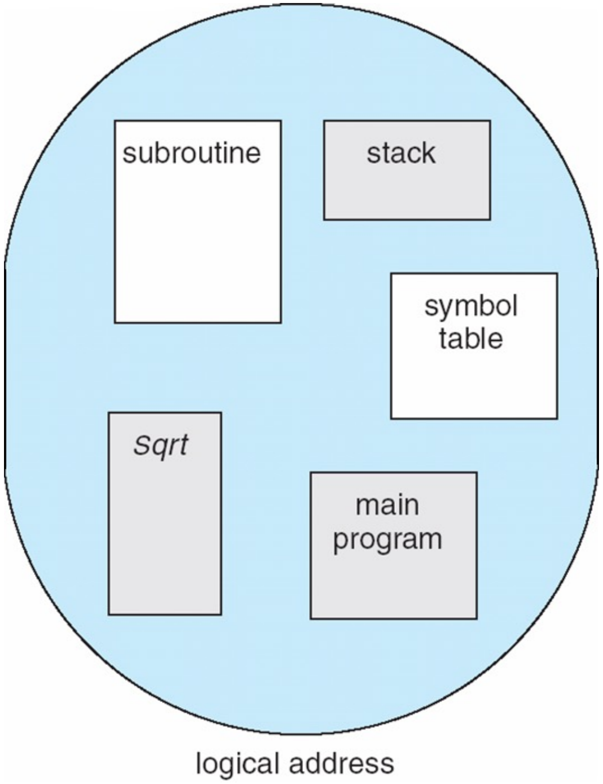
    object

    local variables, global variables

    common block

    stack, arrays, symbol table

- Segment table
  - Segment-table base register (STBR)
  - Segment-table length register (STLR)
- segments vary in length, can very dynamically
- Segments may be paged
- Used for x86-32 bit
- Origin of term "segmentation fault"


subroutine
stack
symbol table
Sqrt
main program
logical address