Chapter 9: MainMemory

Outline

- Background
- Contiguous Memory Allocation
- Paging

- Structure of the Page Table
- Swapping
- Example: The Intel 32 and 64-bit Architectures ▪ Example: ARMv8 Architecture

Objectives

- To provide a detailed description of various ways of organizing memory hardware

- To discuss various memory-management techniques, - To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentationwith paging

Background

- Program is permanently kept on **backing store** (disk) - For a program to be run it must be brought frombackingstoreinto memory and placed within

a process : fetch, decode, execute, increment PC or EIP

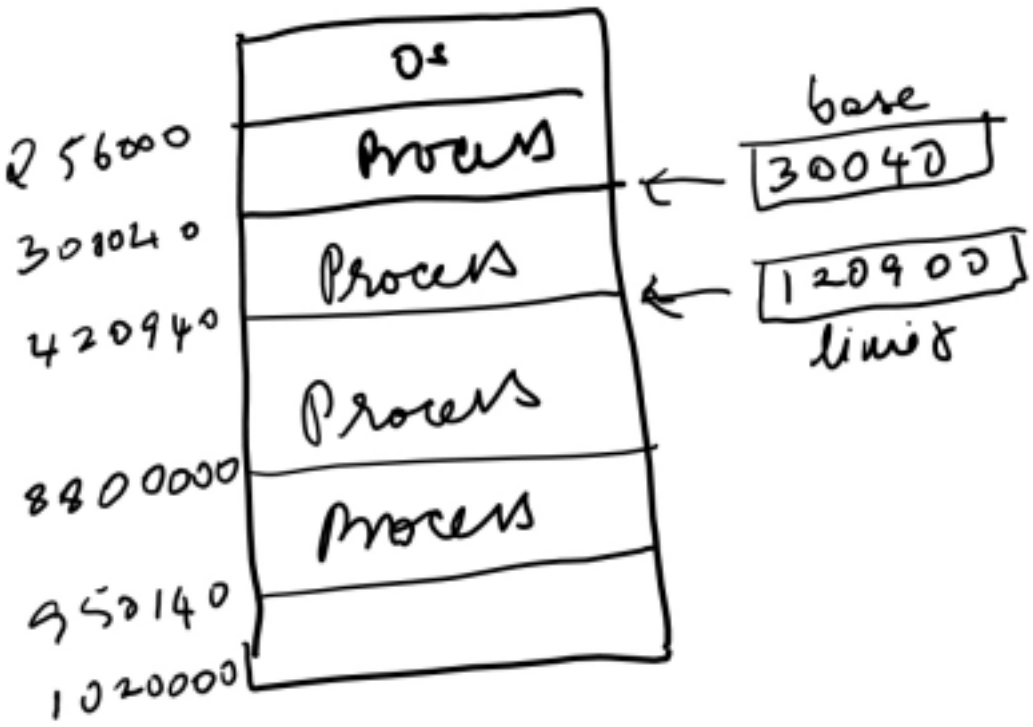
- Main memory and registers are the only storage devicestheCPU can access directly
- Memory unit only sees a stream of: • addresses + read requests, or
  - address + data and write requests
- Register access is done in one CPU clock (or less) ▪ Main memory can take many cycles, causing a **stall** ▪ **Cache** sits between main memory and CPU registers ▪ Protection of memory is required to ensure correct operation; hardware-level

Protection

- Need to censure that a process can access only access thoseaddresses in it address space.
- to protect processes from each; each process has its ownmemory▪ We can provide this protection by using a pair of **base**(holdssmallestlegal PA) and

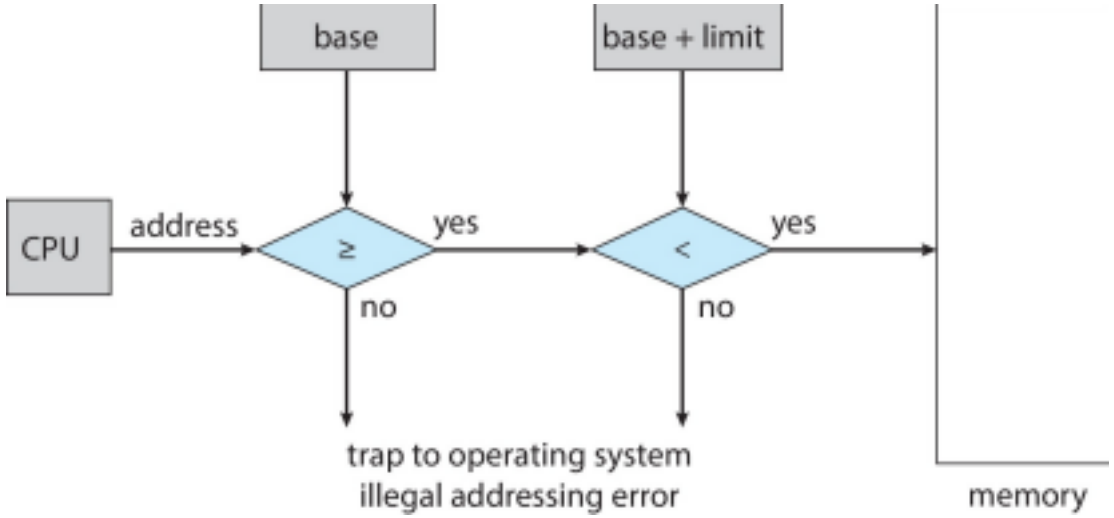
limit registers (specifies the range of accessibleaddresses)

- Every memory access generated in user mode is checked



Hardware AddressProtection

- CPU must check every memory access generated in user modetobe sure it is between base and limit for that user



- The instructions to loading the base and limit registers areprivileged

Address Binding

- Programs on disk, ready to be brought into memory toexecute, areplaced in an **input queue**
  - Without support, must be loaded into address 0000▪ Inconvenient to have

first user process physical address alwaysat0000• How can it not be?

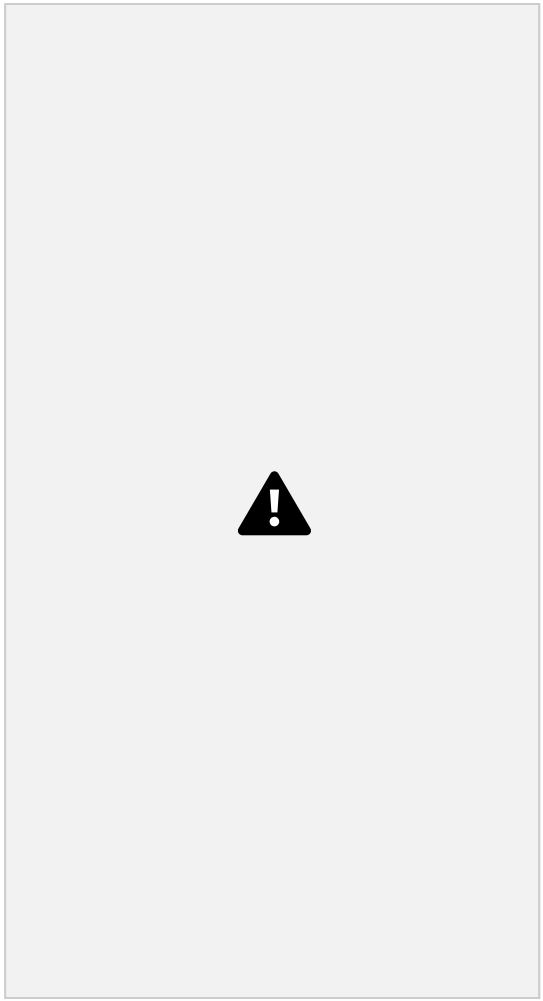
- Addresses represented in different ways at different stagesof aprogram's life
  - Source code addresses are usually symbolic • Compiled code addresses **bind** to relocatable addresses[4] i.e., "14 bytes from beginning of this module
    "
  - Linker or loader will bind relocatable addresses to absoluteaddresses [4] i.e., 74014
  - Each binding maps one address space to another

Binding of InstructionsandDatatoMemory

- Address binding of instructions and data to memory addressescan happen at three different stages
  - **Compile time**: If memory location known a priori, **absolutecode** can be generated; must recompile code if startinglocation changes

- **Load time**: Must generate **relocatable code** if memorylocation is not known at compile time
- **Execution time**: Binding delayed until run time if theprocess can be moved during its execution fromonememory segment to another
  - Need hardware support for address maps (e.g., baseand limit registers)

Multistep Processingof



aUserProgram

Logical vs. Physical AddressSpace

- The concept of a logical address space that is bound toaseparate**physical**

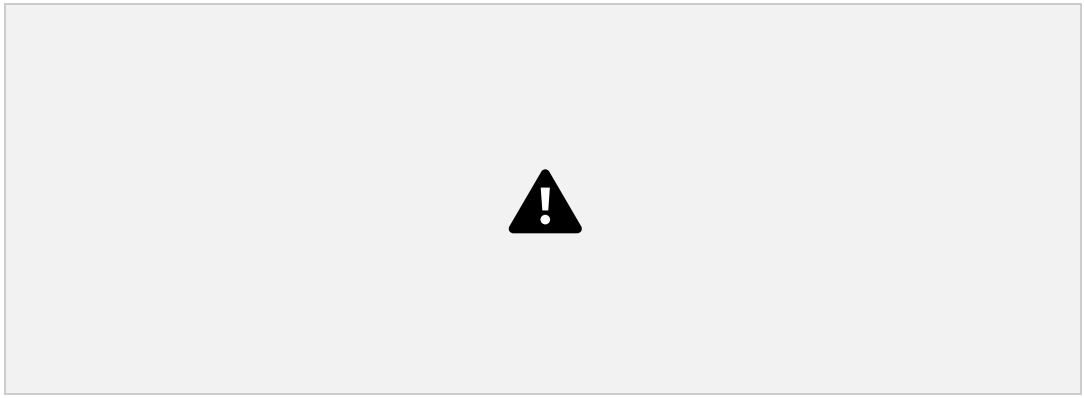
address space is central to proper memory management• **Logical address** – generated by the CPU; also referredtoas**virtual address**

• **Physical address** – address seen by the memory unit ▪ Logical and physical addresses are the same in compile-timeandload-time address-binding schemes; logical (virtual) andphysical addresses differ in execution-time address-binding scheme▪ **Logical address space** is the set of all logical addressesgeneratedby a program

▪ **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit(MMU)▪

Hardware device that at run time maps virtual to physical address

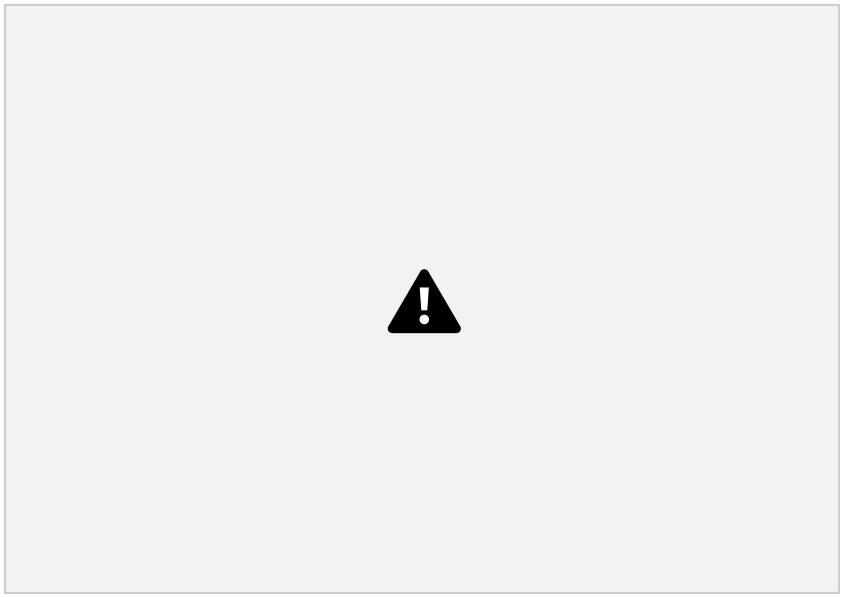


- Many methods possible, covered in the rest of this chapter

RelocationRegister

- Consider simple scheme. which is a generalization of thebase register scheme.

- The base register now called **relocation register** - The value in the relocation register is added to every addressgenerated by a user process at the time it is sent to memory



Relocation Register (Cont.)

- 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 Loading

- The program consist of main part and a number of routines
- The entire

program does need to be in memory to execute▪ 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 handleinfrequently occurring cases

▪ No special support from the operating systemis required• Implemented through program design • OS can help by providing libraries to implement dynamic loading

Dynamic Linking

▪ **Static linking** – system libraries and programcode combinedbytheloader into the binary program image ▪ Dynamic linking –linking postponed until execution

time▪ Small piece of code, called **stub**, is used to locate the appropriatememory-resident library routine

- Stub replaces itself with the address of the routine, andexecutestheroutine
- Operating system checks if routine is in processes ’memoryaddress
- If not in address space, add to address space ▪ Dynamic linking is particularly useful for libraries ▪ System also known as **shared libraries** ▪ Consider applicability to patching system libraries • Versioning may be needed

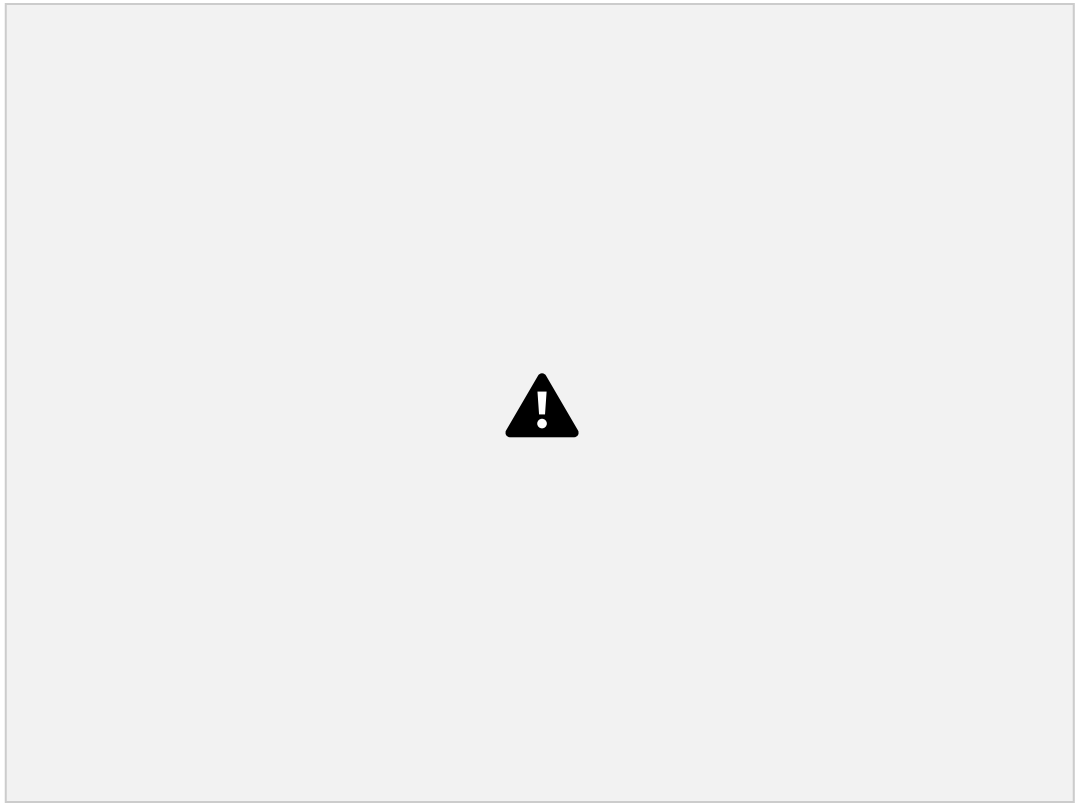
Swapping

- A process can be **swapped** temporarily out of memory toabackingstore, and then brought **back** into memory for continuedexecution• Total physical memory space of processes can exceedphysical memory
- **Backing store** – fast disk large enough to

accommodatecopiesofallmemory images for all users; must provide direct accesstothesememory images

- **Roll out, roll in** – swapping variant used for priority-basedschedulingalgorithms; lower-priority process is swapped out so higher-priorityprocess can be loaded and executed
- Major part of swap time is transfer time; total transfer timeisdirectlyproportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processeswhichhave memory images on disk

Schematic Viewof Swapping



Context Switch Time includingSwapping

▪ If next processes to be obtain CPU is not in memory, needto 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 rateof 50MB/sec

- Swap out time of 2000 milliseconds
- Plus swap in of same sized process
- Total context switch swapping component time of 4000milliseconds (4 seconds)

Memory Allocation ▪ Main memory must support both

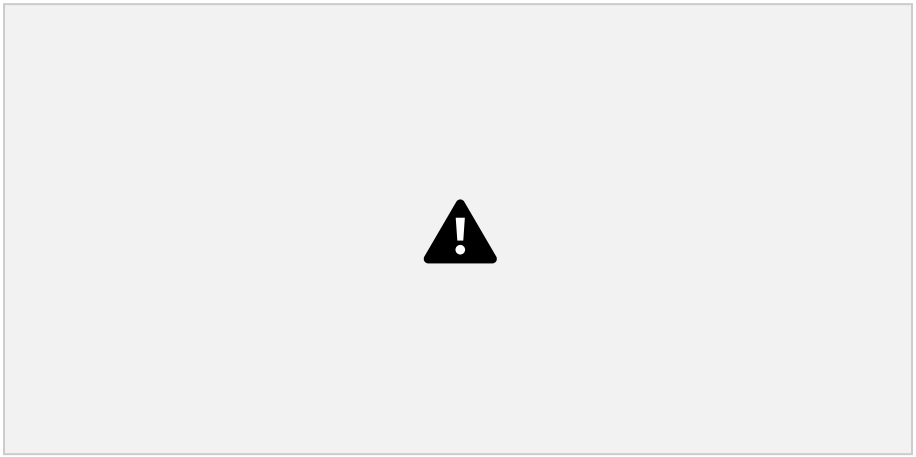
OS and user processes▪ Limited resource, must allocate efficiently ▪ **Contiguous allocation** is one early method ▪ Main memory usually into two **partitions**:

- Resident operating system, usually held in lowmemorywith interrupt vector
- User processes then held in high memory • Each process contained in single contiguous sectionof memory

ContiguousAllocation

▪ Relocation registers used to protect user processes fromeachother, andfrom changing operating-system code and data • Base register contains value of smallest physical address

- Limit register contains range of logical addresses – eachlogical address must be less than the limit register
- MMU maps logical address *dynamically*

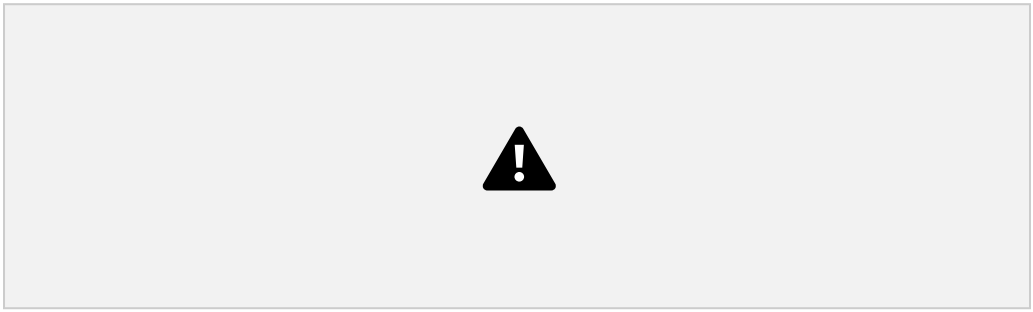


Variable PartitionAllocation

- Fixed allocation: Degree of multiprogramming limited by number of partitions (restricts both the number of simultaneous processes andthemaximum size of each process)
- **Variable-partition** sizes for efficiency (sized to a given process' needs) ▪ **Hole** – block of available memory; holes of various size are scatteredthroughout memory
- When a process arrives, it is allocated memory from a hole largeenoughto 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-AllocationProblem

How to satisfy a request of size ***n*** from a list of free holes?
- **First-fit**: Allocate the ***first*** hole that is big enough. Whateverfractionofthe hole not needed by the request is left on thefreelist asasmallerhole.
- **Best-fit**: Allocate the ***smallest*** hole that is big enough. Thissaveslargeholes for other process requests that may needthemlater,buttheresulting unused portions of holes may be too small tobeofanyuse,and will therefore be wasted; must search entire list, unlessorderedby

size.

- Produces the smallest leftover hole

- **Worst-fit**: Allocate the ***largest*** hole thereby increasingthelikelihoodthat the remaining portion will be usable for satisfyingfuturerequests;must also search entire list

- First-fit and best-fit better than worst-fit in terms of speedandstorageutilization; but first fit is faster

Fragmentation

- **External Fragmentation** – total memory space exists tosatisfyarequest, but it is not contiguous; holes present in betweentheprocess- **Internal Fragmentation** – allocated memory may be slightlylargerthan requested memory; this size difference is memory internal toapartition, but not being used; holes are present within theprocessitself- Internal fragmentation occurs with all memory allocationstrategies.This is caused by the fact that memory is allocated

in blocksof afixedsize, whereas the actual memory needed will rarely bethat exact size.

Fragmentation(Cont.)

- Reduce external fragmentation by **compaction** • Shuffle memory contents to place all free memory together inonelarge block
  - Compaction is possible *only* if relocation is dynamic (usingexecution-time address binding), and is done at executiontime.• moving all processes down to one end of physical memorysoasto place all free memory together

to get a large freeblock. • This only involves updating the relocation register for eachprocess, as all internal work is done using logical addresses.

Paging

- Physical address space of a process can be noncontiguous; processis allocated physical memory whenever the latter is available• Avoids external fragmentation
  - Avoids problem of varying sized memory chunks ▪ Divide physical memory into fixed-sized blocks called **frames**• Size is power of 2, between 512 bytes and 16 Mbytes▪ Divide logical memory into blocks of same size called

pages▪ Keep track of all free frames

- To run a program of size ***N*** pages, need to find ***N***free framesandload program
- Set up a **page table** to translate logical to physical addresses▪

Backing store likewise split into pages

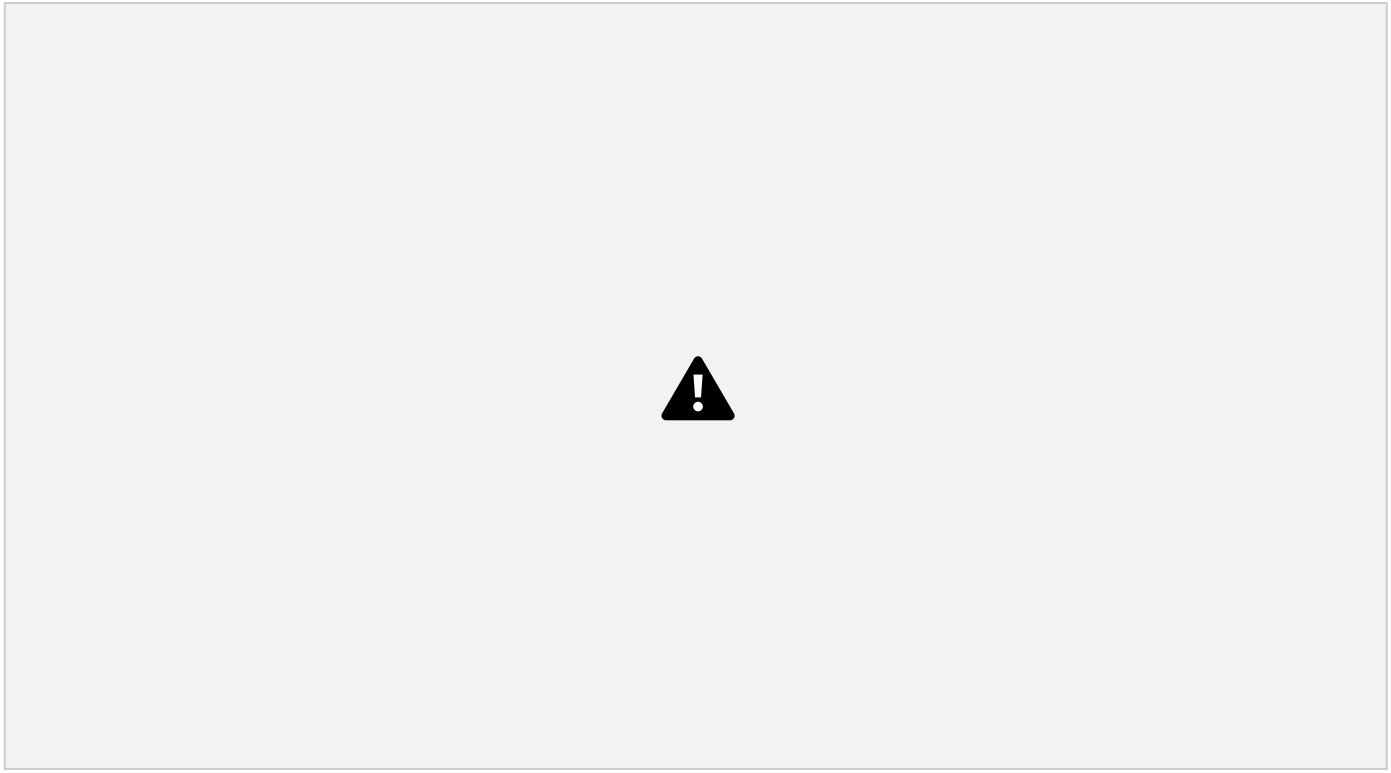
- Still have Internal fragmentation

Address TranslationScheme

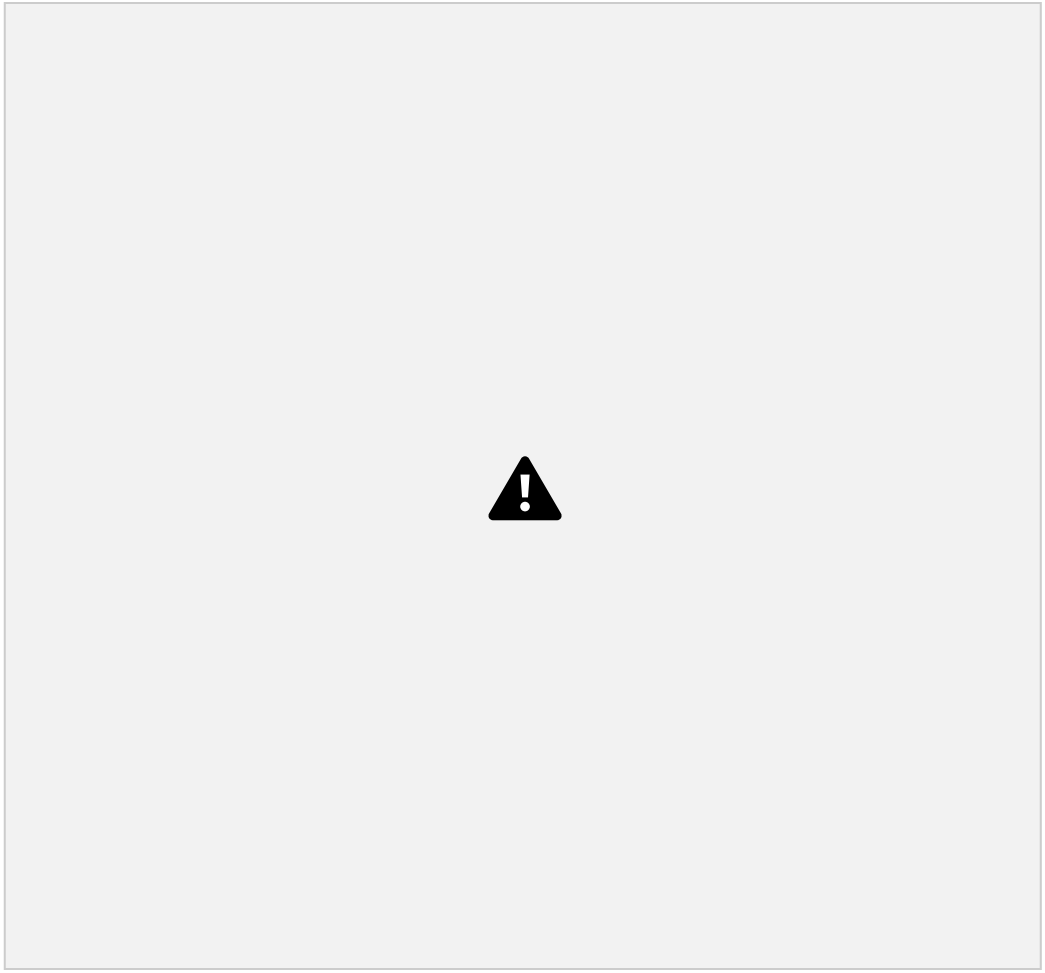
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **pagetable**whichcontains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address todefinethephysical memory address that is sent to the memoryunit

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

Paging Hardware

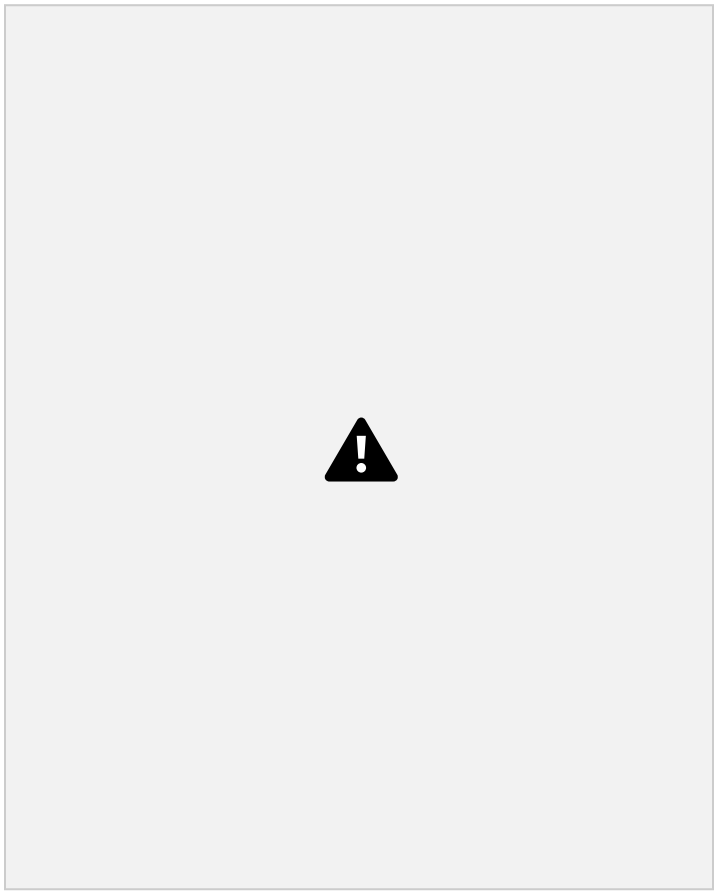


Paging Model of Logical andPhysical Memory



Paging Example

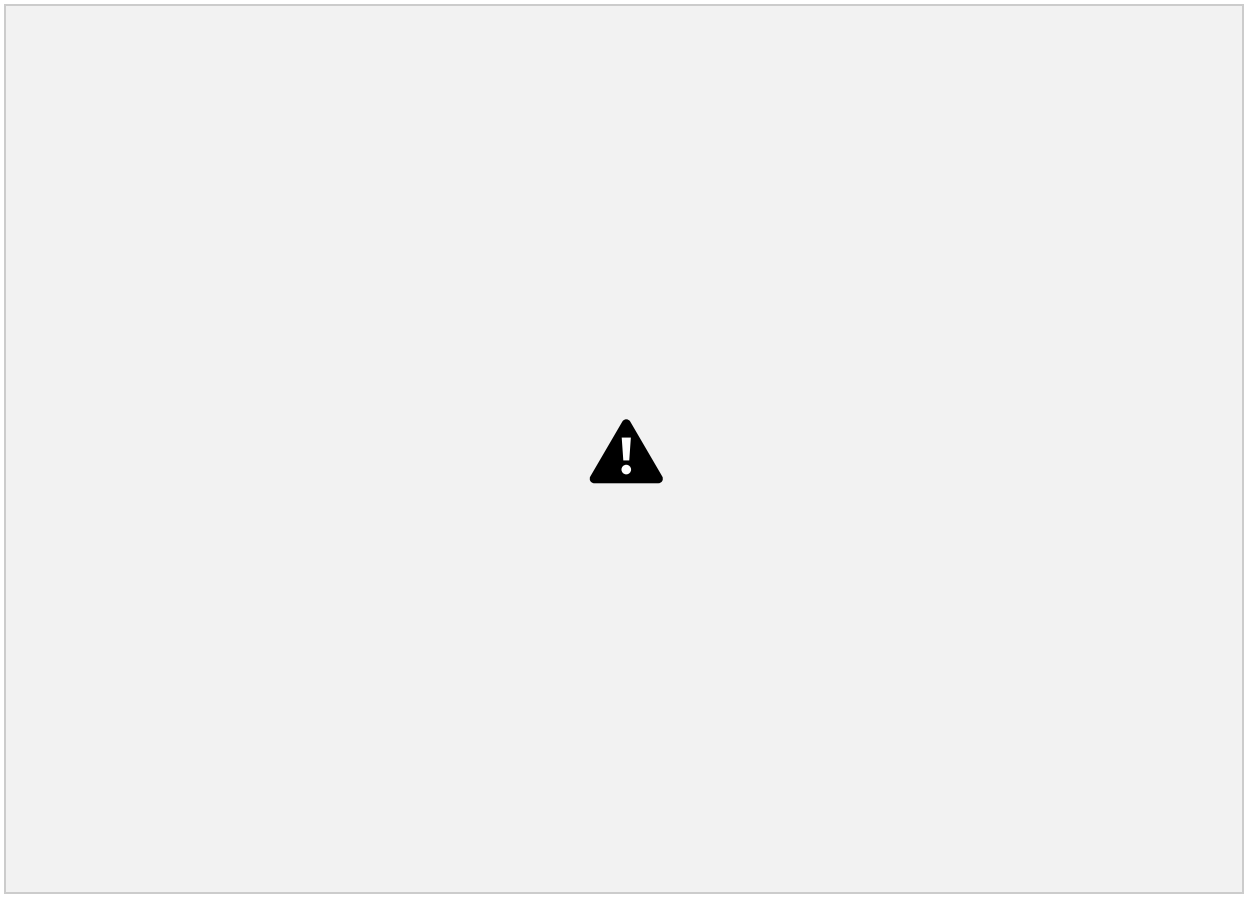
- Logical address: n = 2 and m = 4. Using a page size of 4bytesanda physical memory of 32 bytes (8 pages)



Paging -- Calculating internal fragmentation

- Page size = 2,048 bytes
- Process size = 72,766 bytes
- 35 pages + 1,086 bytes
- Internal fragmentation of 2,048 - 1,086 = 962 bytes ▪ 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 sizes growing over time
  - Solaris supports two page sizes – 8 KB and 4 MB

Free Frames



Before allocation After allocation

Implementationof PageTable

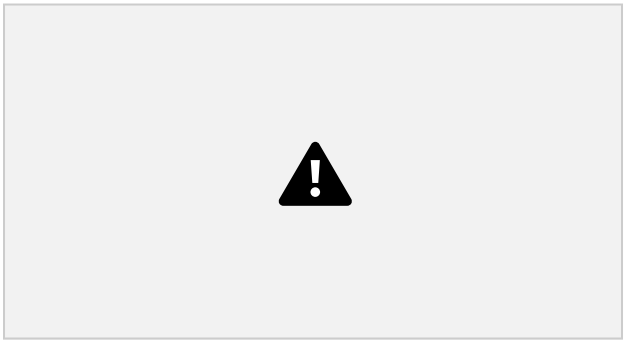
- Page table is kept in main memory
  - **Page-table base register** (**PTBR**) points to the pagetable• **Page-table length register** (**PTLR**) indicates size of thepage table
- In this scheme every data/instruction access requires twomemory accesses
  - One for the page table and one for the data / instruction▪ The two-memory access problem can be solved by theuseof aspecial fast-lookup hardware cache called **translationlook- aside buffers** (**TLBs**) (also called **associative memory**).

Translation Look-AsideBuffer

- 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
- Some TLBs store **address-space identifiers** (**ASIDs**) ineach TLB entry – uniquely identifies each process to provide address-space protection for that process • Otherwise need to flush the TLB at every context switch

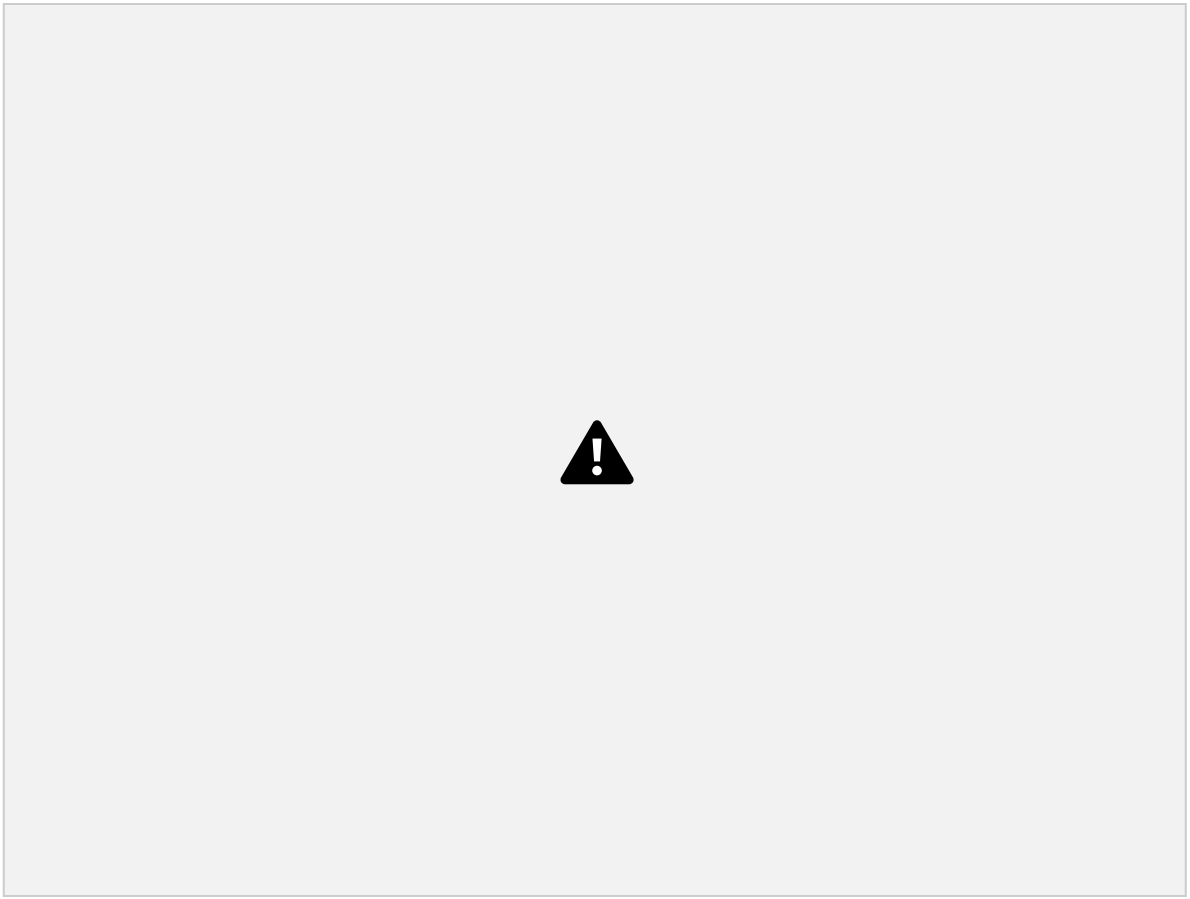
Hardware

- Associative memory – parallel search



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

Paging HardwareWithTLB



Effective AccessTime

- Hit ratio – percentage of times that a page number is foundintheTLB
- An 80% hit ratio means that we find the desired page number intheTLB 80% of the time.
- Suppose that it takes 10 nanoseconds to access memory.
  - If we find the desired page in TLB then a mapped-memoryaccesstake 10 nanoseconds
  - Otherwise we need two memory access so it is 20 nanoseconds
- **Effective Access Time** (**EAT**)

$$EAT = 0.80 \times 10 + 0.20 \times 20 = 12 \text{ nanoseconds}$$

implying 20% slowdown in access time

- Consider a more realistic hit ratio of 99%,

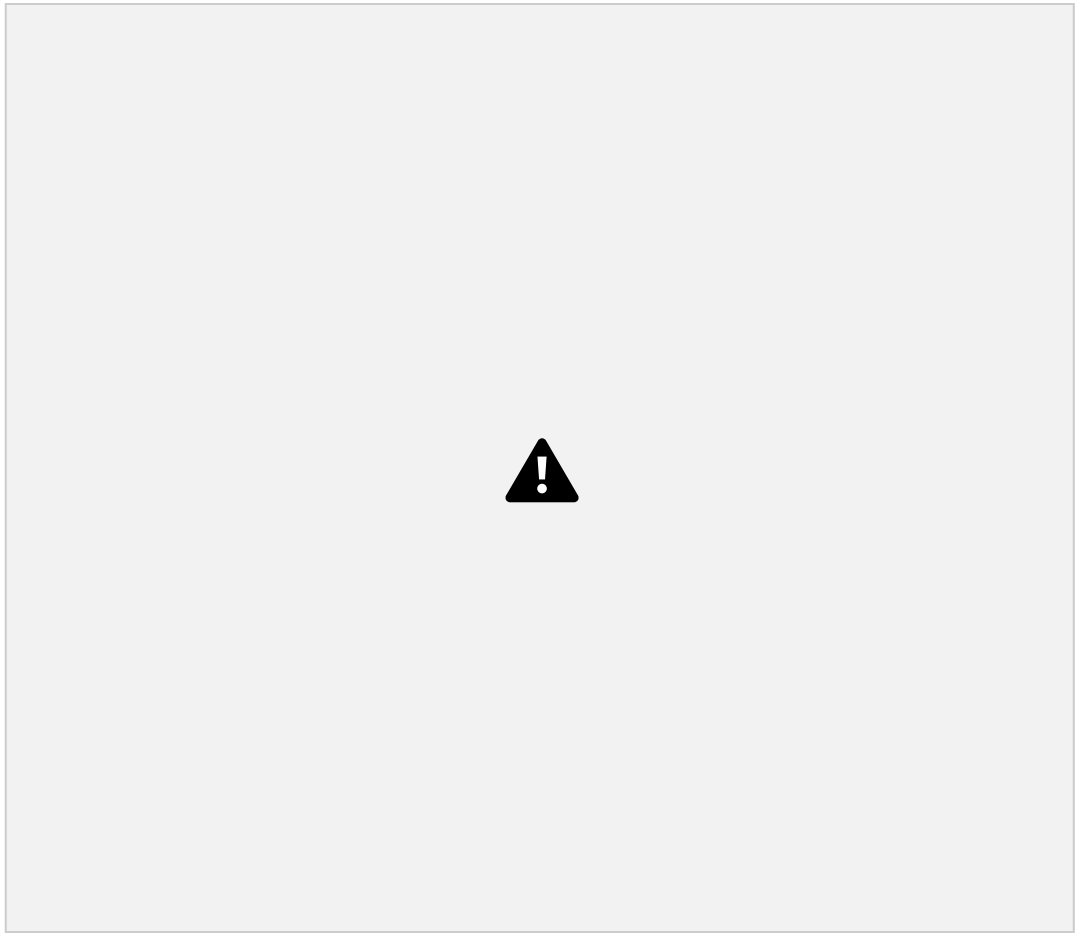
$$EAT = 0.99 \times 10 + 0.01 \times 20 = 10.1 \text{ nanoseconds}$$

implying only 1% slowdown in access time.

Memory Protection

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

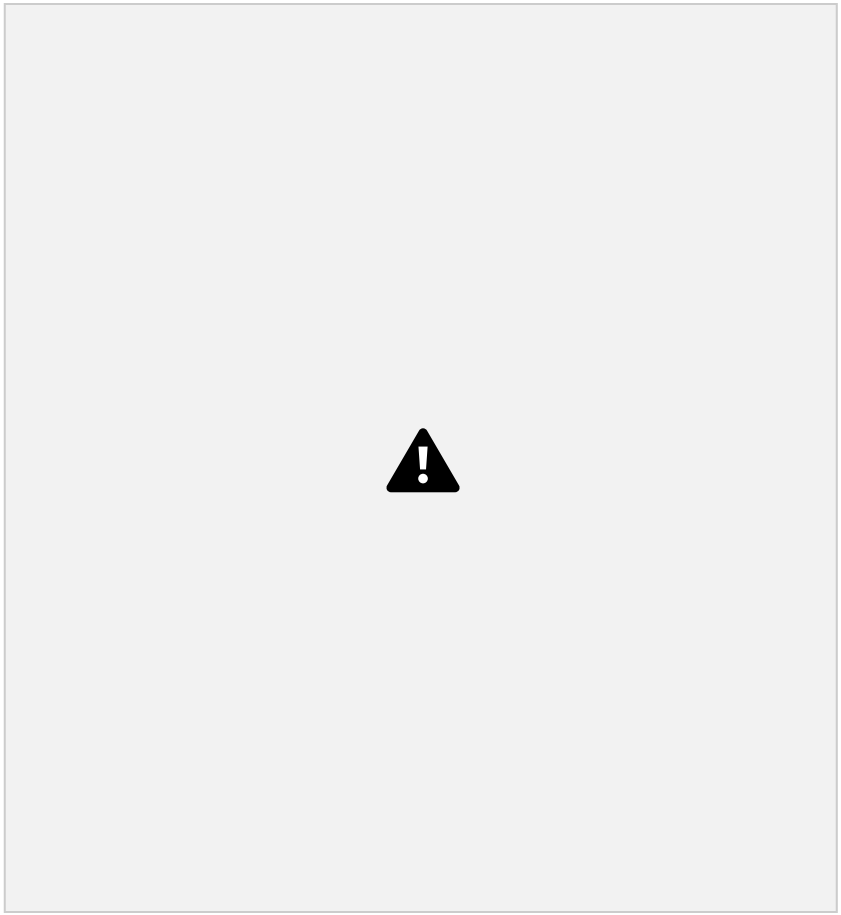
Valid (v) or Invalid (i) Bit InAPageTable



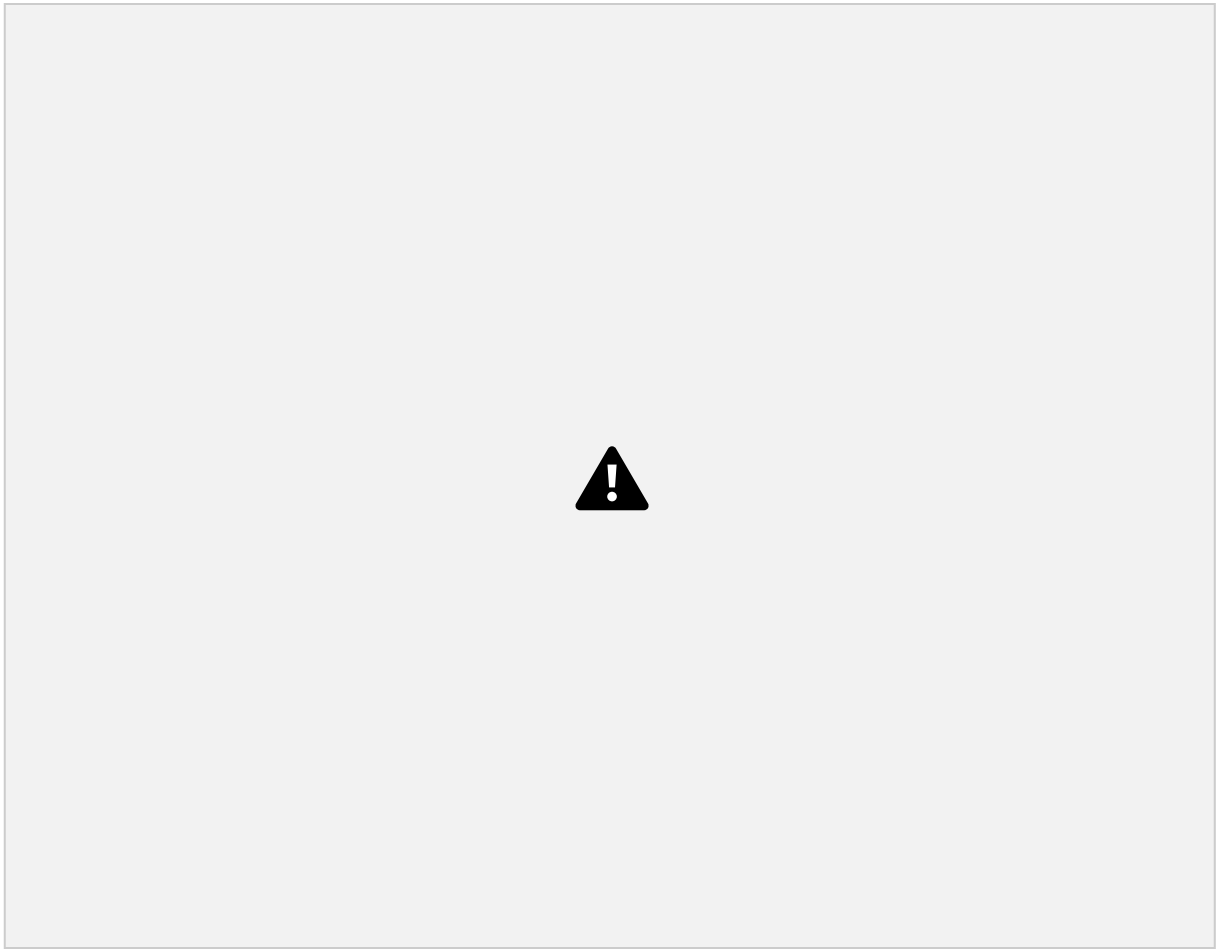
SharedPages

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

Shared PagesExample



Shared PagesExample



Structure of thePageTable

- LAS = 4GB = 2^32 = 32 bits LA
- PAS = 64MB = 2^26 = 26 bits PA
- Page size = 4KB = 2^12
- No. of pages ? = 2^32/2^12 = 2^20
- No. of frames ? = 2^26 / 2^12 = 2^14
- 32 bits LA= page number(32-12=20) : offset (12)
- 26 bits PA = frame no. (26-12=14) : offset (12)
- No. of entries in page = 2^20
- Page table size = 14 X 2^20

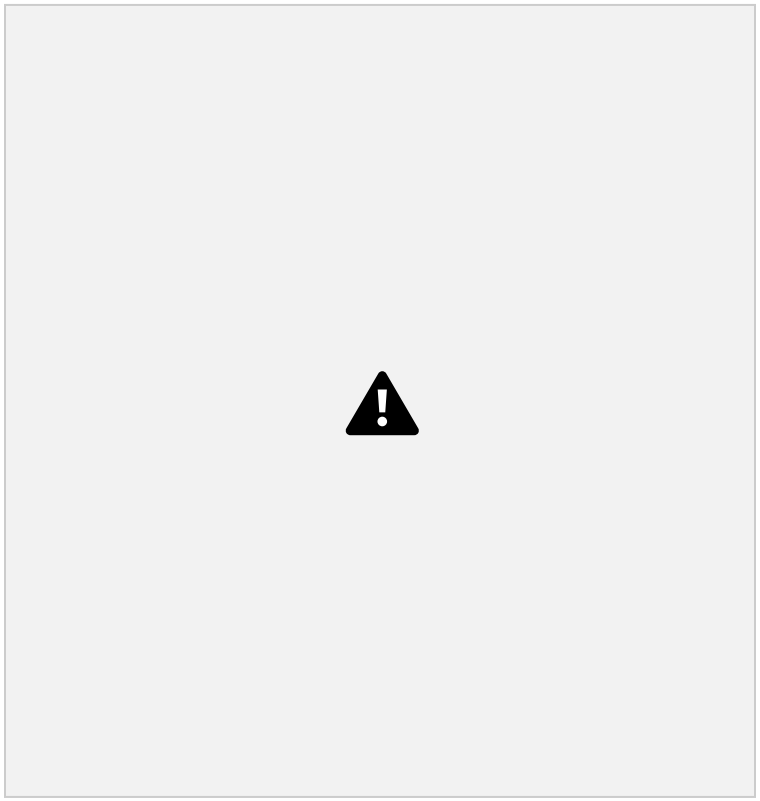
Structure of thePageTable

- Memory structures for paging can get huge using straight-forwardmethods
  - Consider a 32-bit logical address space as on moderncomputers
  - Page size of 1 KB (2^{10})
  - Page table would have 1 million entries ($2^{32} / 2^{10}$)
  - If each entry is 4 bytes each process requires 16MBof physical address space for the page table alone
    - Don't want to allocate that contiguously in mainmemory
    - One simple solution is to divide the page table intosmaller units
    - Hierarchical Paging

- Hashed Page Tables
- Inverted Page Tables

Hierarchical PageTables

- 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



Two-Level PagingExample

- A logical address (on 32-bit machine with 1K page size) isdividedinto:
  - 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 dividedinto:
  - a

12-bit page number

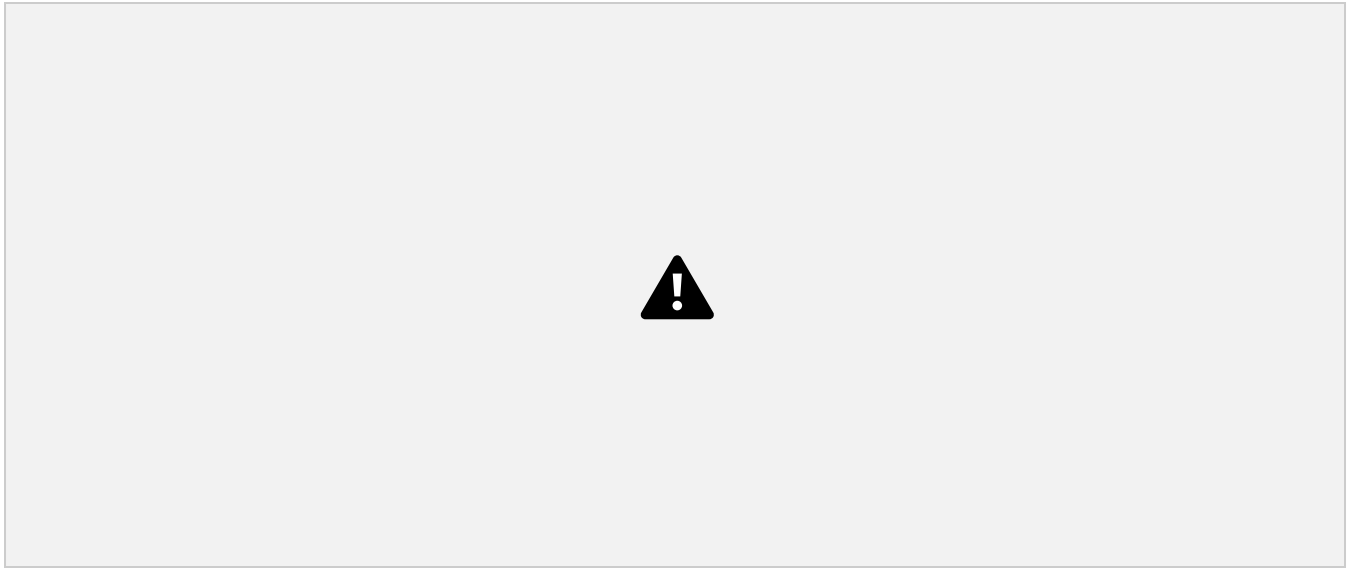
- a 10-bit page offset

- 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**

Address-TranslationScheme



64-bit Logical AddressSpace

- 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-byteentries
  - Address would look like

  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page tableisstill 2^{34}bytes in size

4 And possibly 4 memory access to get to one physical memorylocation

Three-level PagingScheme

Hashed PageTables

- Used in architecture with address spaces > 32 bits
- The virtual page number

is hashed into a page table • This page table contains a chain of elements hashingtothe same location

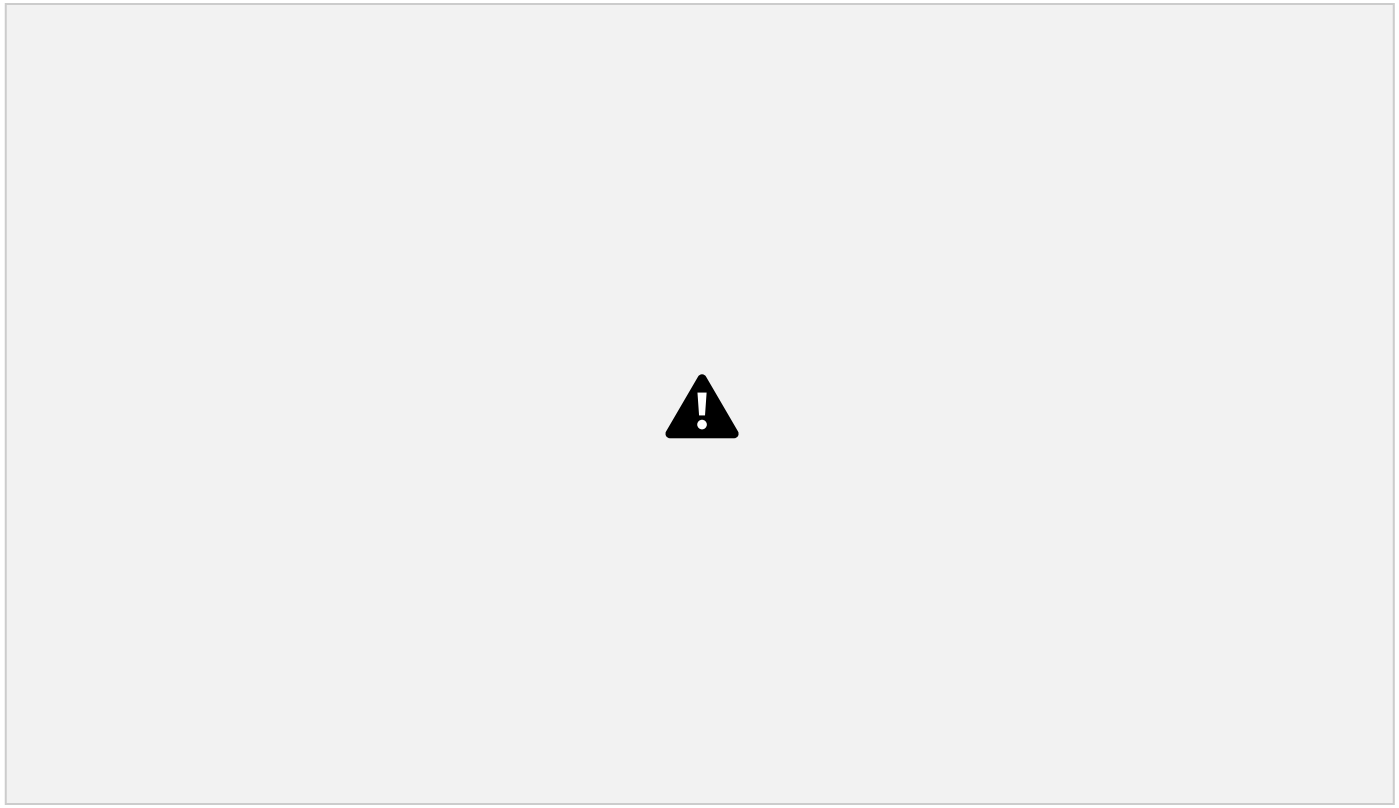
- Each element contains
  1. The virtual page number
  2. The value of the mapped page frame
  3. A pointer to the next element
- Virtual page numbers are compared in this chain searchingfor a match
  - If a match is found, the corresponding physical frameis extracted

Hashed Page Tables(Cont.)

- 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 (wherememory references are non-contiguous and scattered)

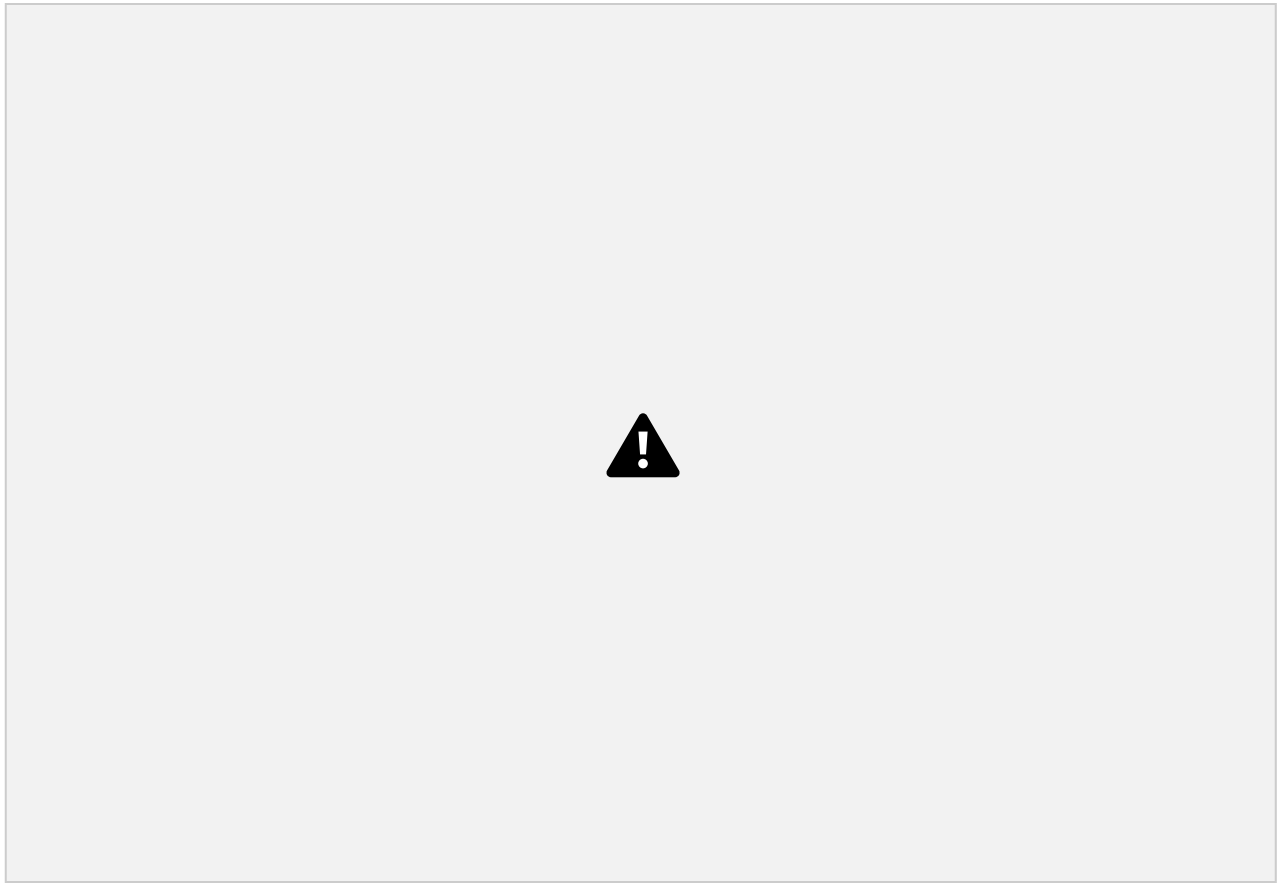
Hashed PageTable



Inverted PageTable

- Rather than having each process keep a page table andtrackof all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page storedinthat real memory location, with information about the process that ownsthatpage
- Decreases memory needed to store each page table, but increasestime needed to search the table when a page referenceoccurs
- Use hash table to limit the search to one (or at most a few) page table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

Inverted Page TableArchitecture



Segmentation

- Memory-management scheme that supports user viewof 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

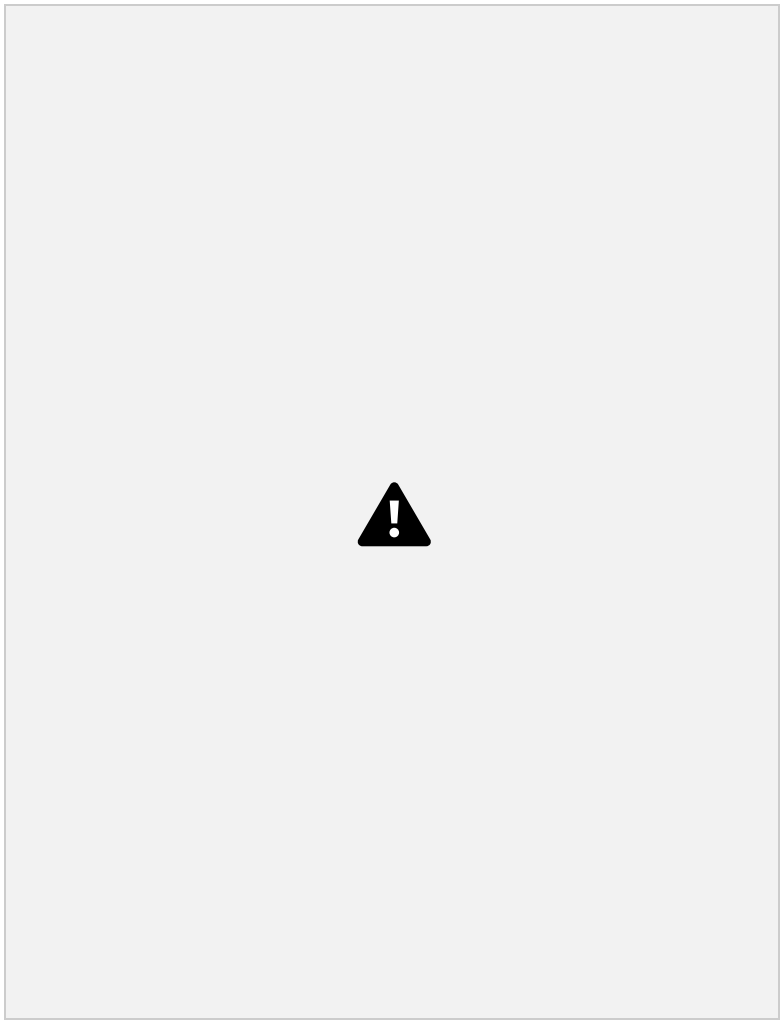
common block

stack

symbol table

arrays

User's Viewof aProgram



Logical Viewof Segmentation1

1

4

2

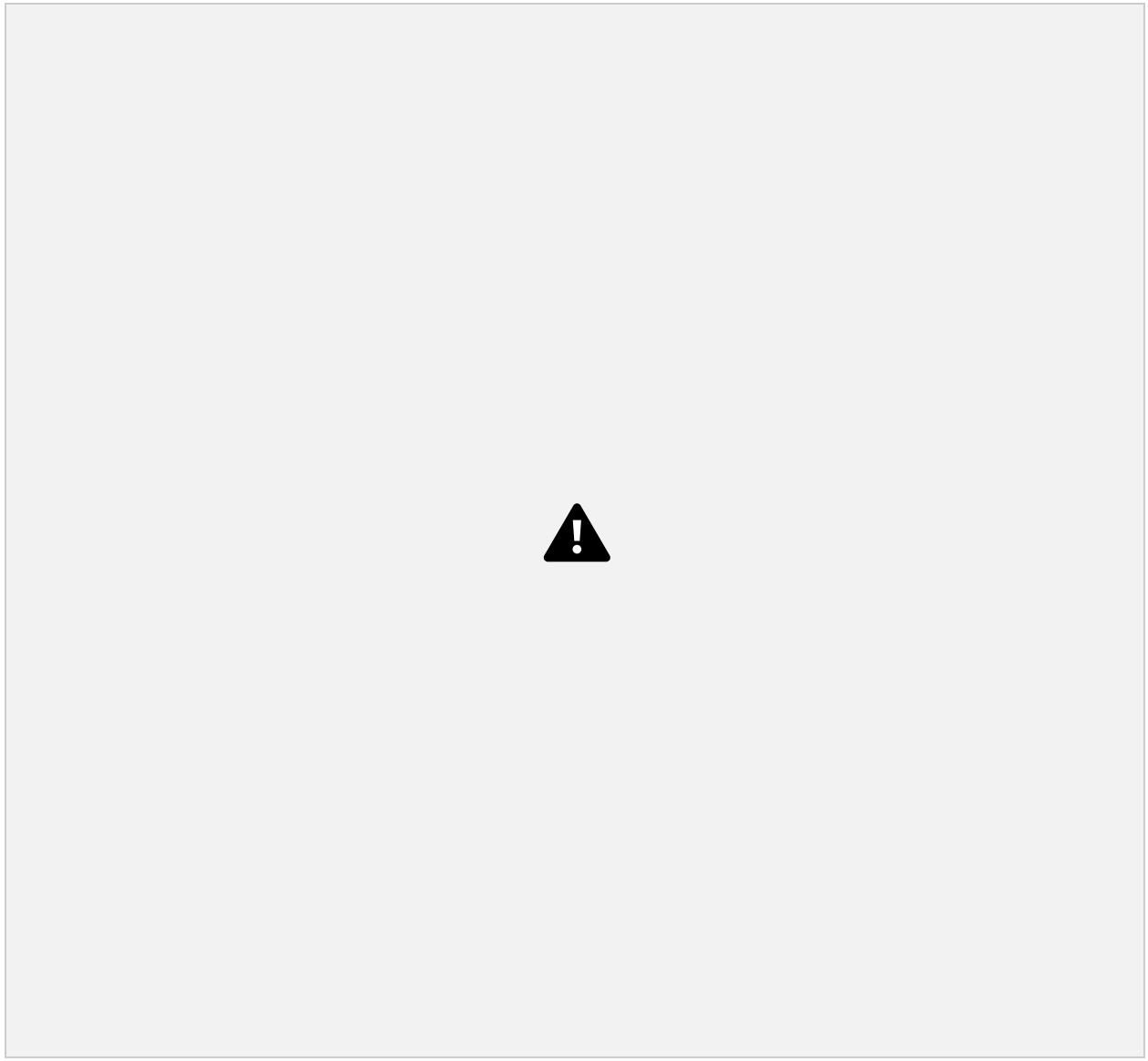
2 3

3

4

user space physical memory space

Logical Viewof Segmentation



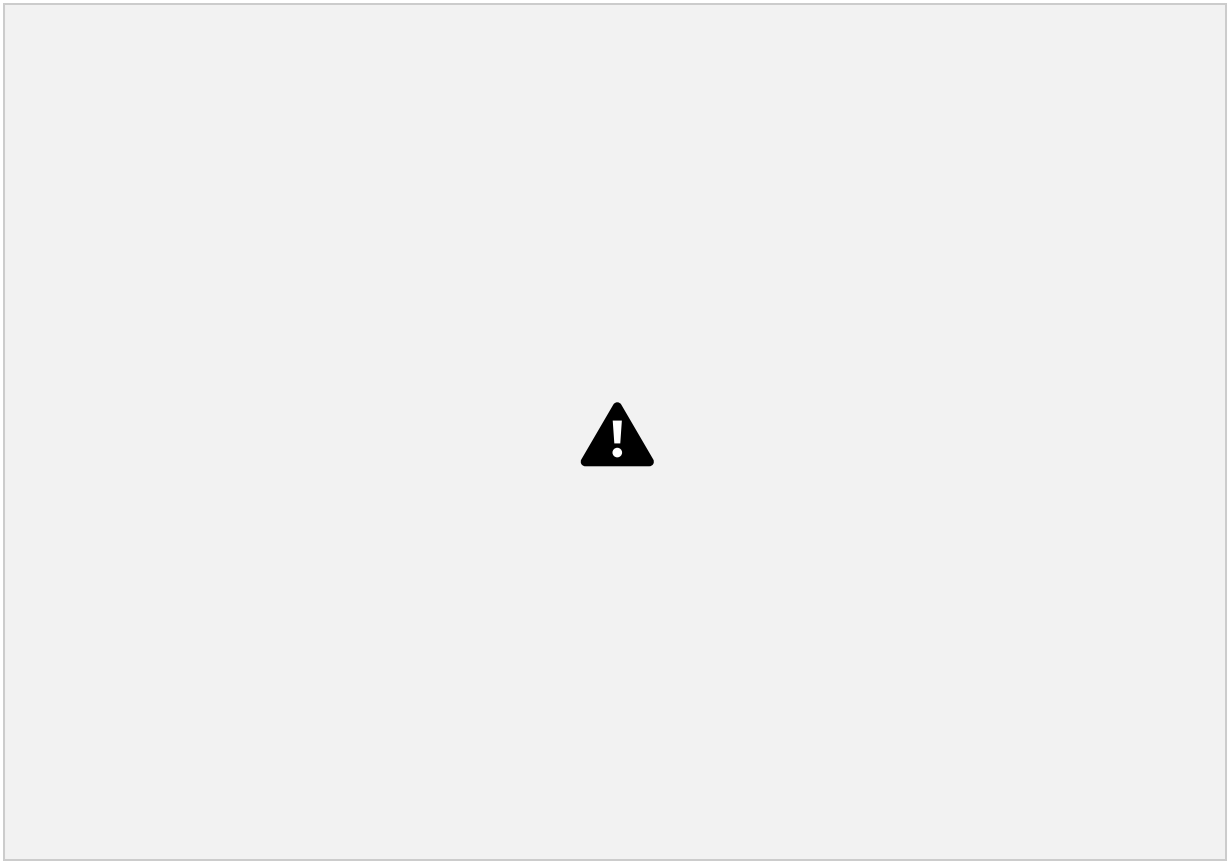
SegmentationArchitecture

Logical address consists of a two tuple:

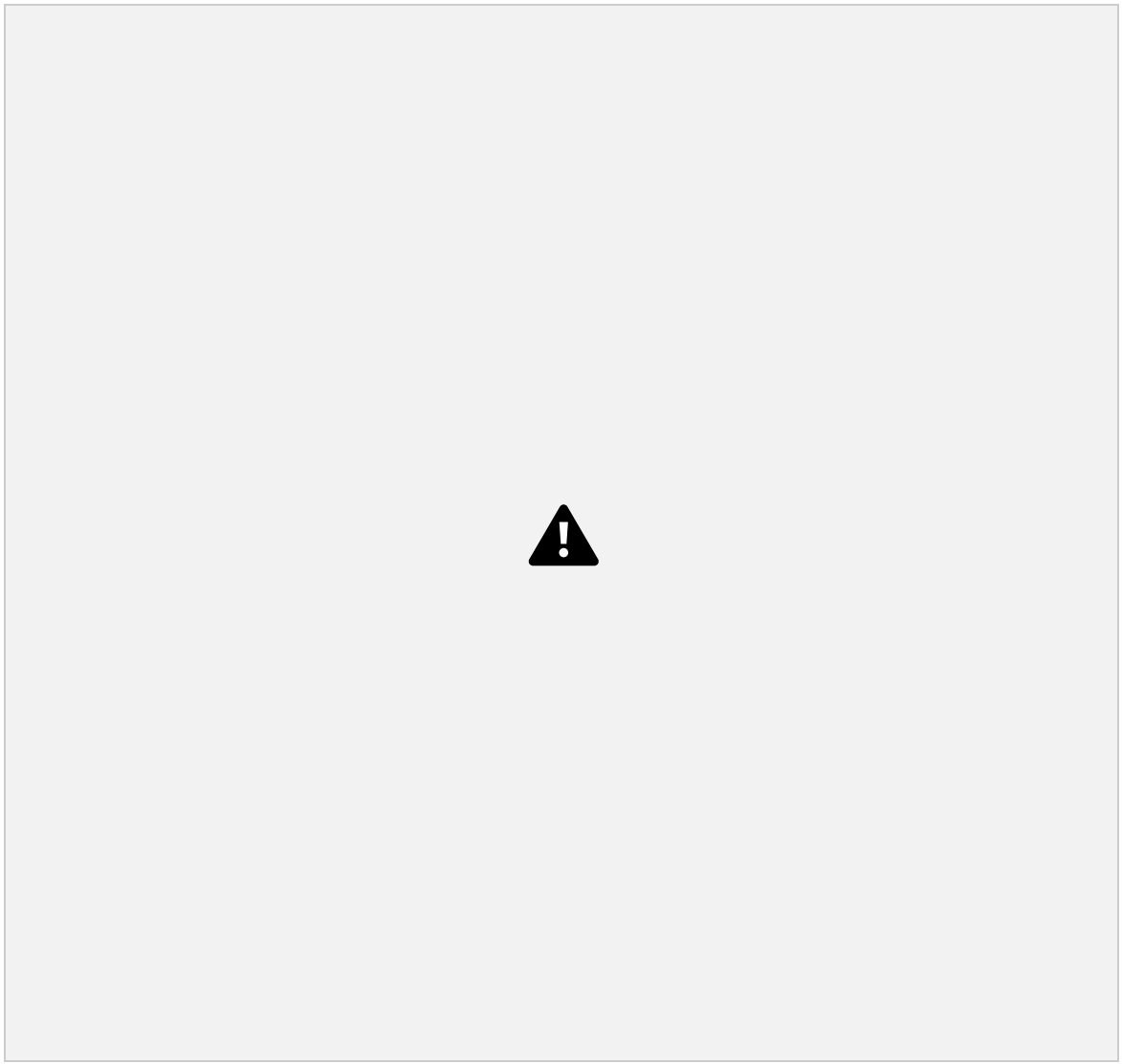
<segment-number, offset>

- **Segment table** – maps two-dimensional physical addresses; eachtable entry has:
  - **base** – contains the starting physical address wherethesegments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register** (**STBR**) points to the segment table's location in memory
- **Segment-table length register** (**STLR**) indicates number of segments used by a program
  - Segment number ***s*** is legal if ***s*** < **STLR**

SegmentationHardware



SegmentationHardware



Endof Chapter9