OPERATING SYSTEMS

Memory Segmentation & Paging

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

中山大學 SUN YAT-SEN UNIVERSITY

Memory Segmentation & Paging

Contents

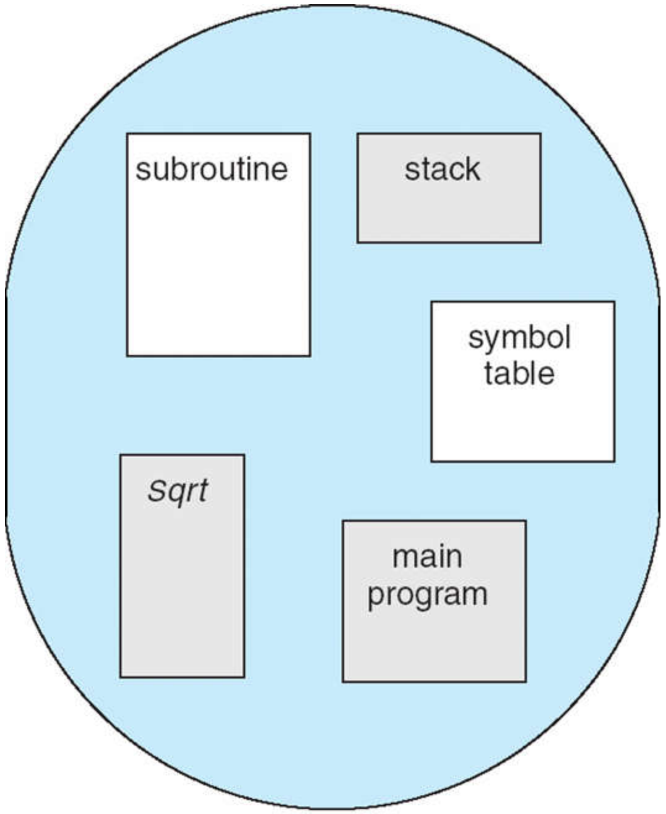
Simple Segmentation

Segmentation Concepts

- Segmentation is 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
    - symbol table
    - arrays.

Segmentation Concepts

- User View of a Program.



Logical address space.

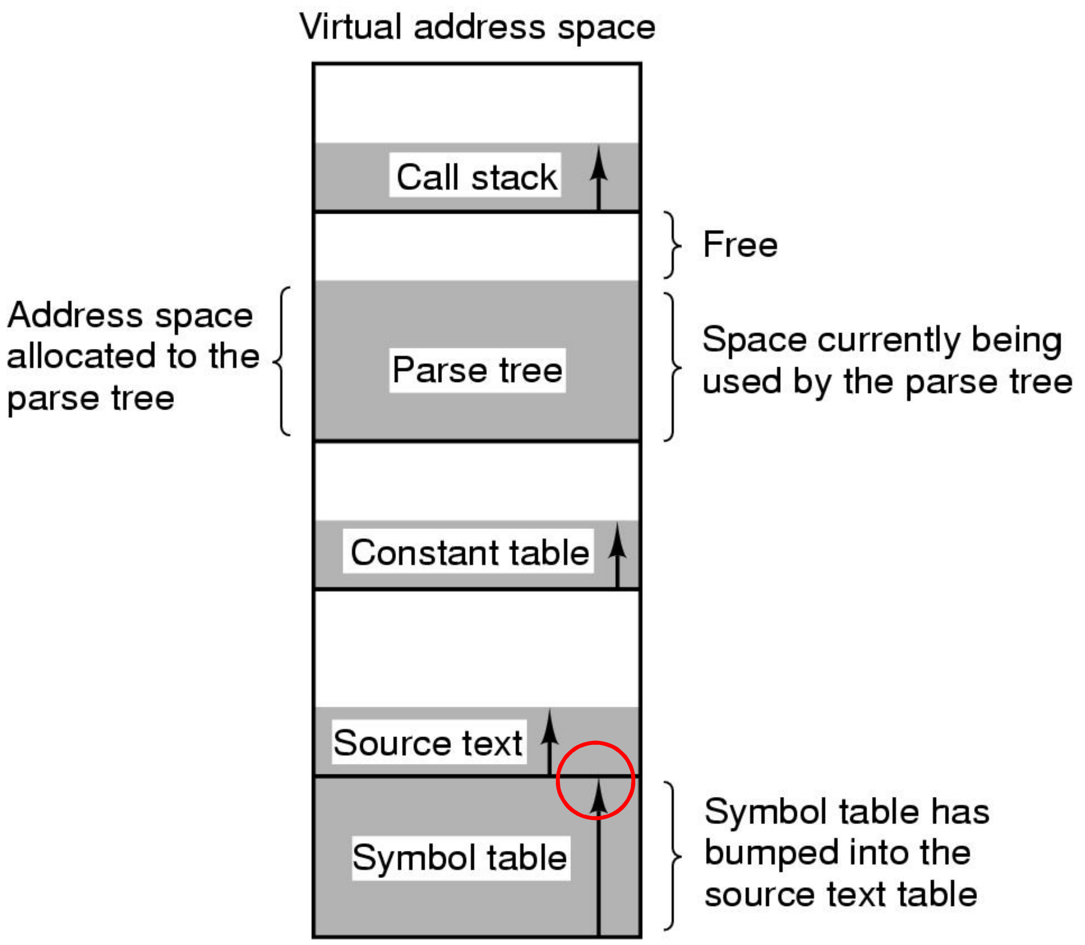
Simple Segmentation

Segmentation Concepts

- User View of a Program – Example.
  - A compiler has many tables that are built up as compilation proceeds, possibly including:
    - The source text being saved for the printed listing (on batch systems).
    - The symbol table – the names and attributes of variables.
    - The table containing integer, floating-point, constants used.
    - The parse tree (语法树), the syntactic analysis of the program.
    - The stack used for procedure calls within the compiler.

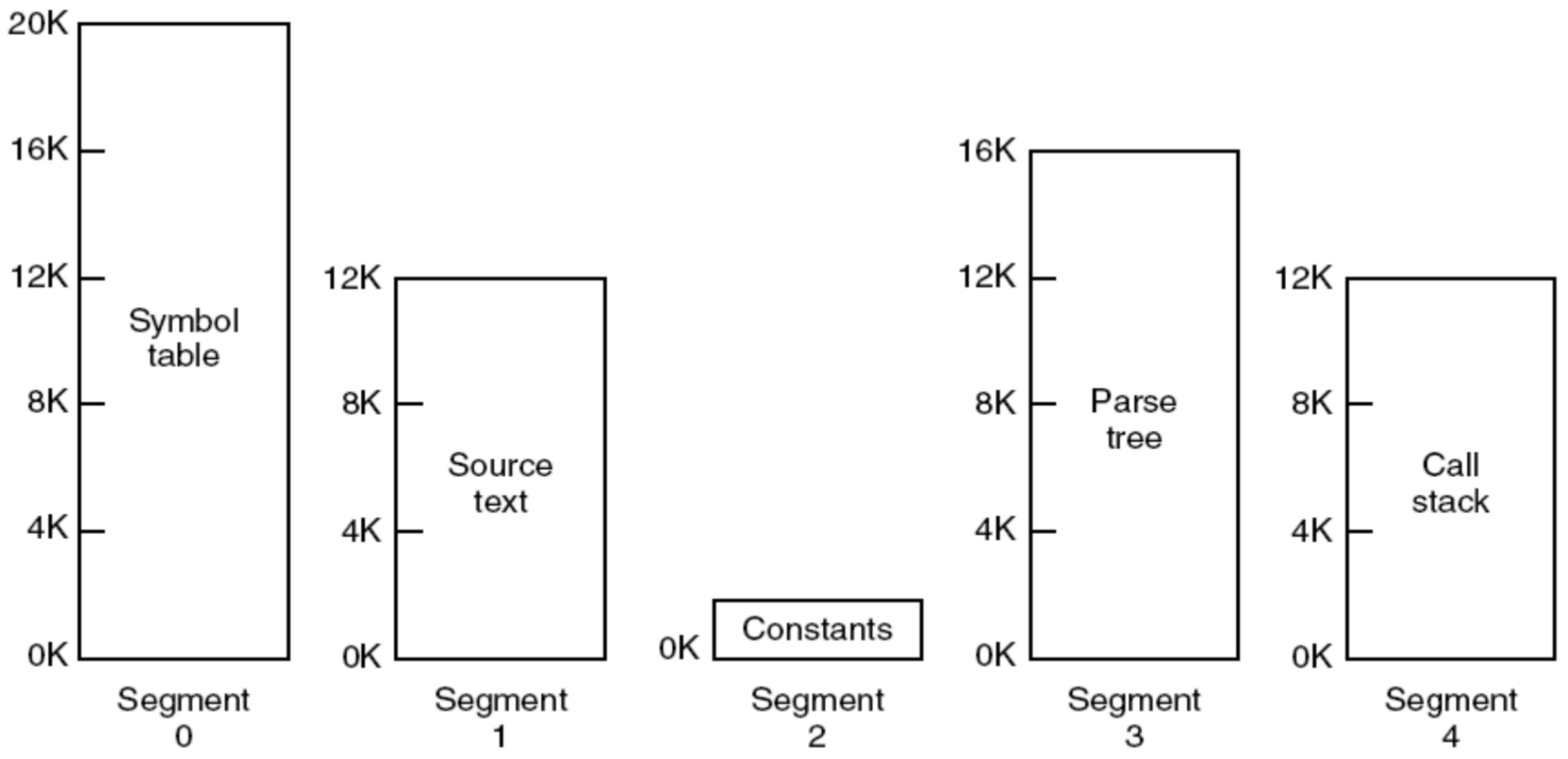
Segmentation Concepts

- One-dimensional address space
  - In a one-dimensional address space with growing tables, one table may bump into another.


Virtual address space
Call stack
Free
Address space allocated to the parse tree
Parse tree
Space currently being used by the parse tree
Constant table
Source text
Symbol table
Symbol table has bumped into the source text table

Simple Segmentation

- **Segmentation Concepts**
  - Segmentation Solution
    - Segmentation allows each table to grow or shrink independently of the other tables.



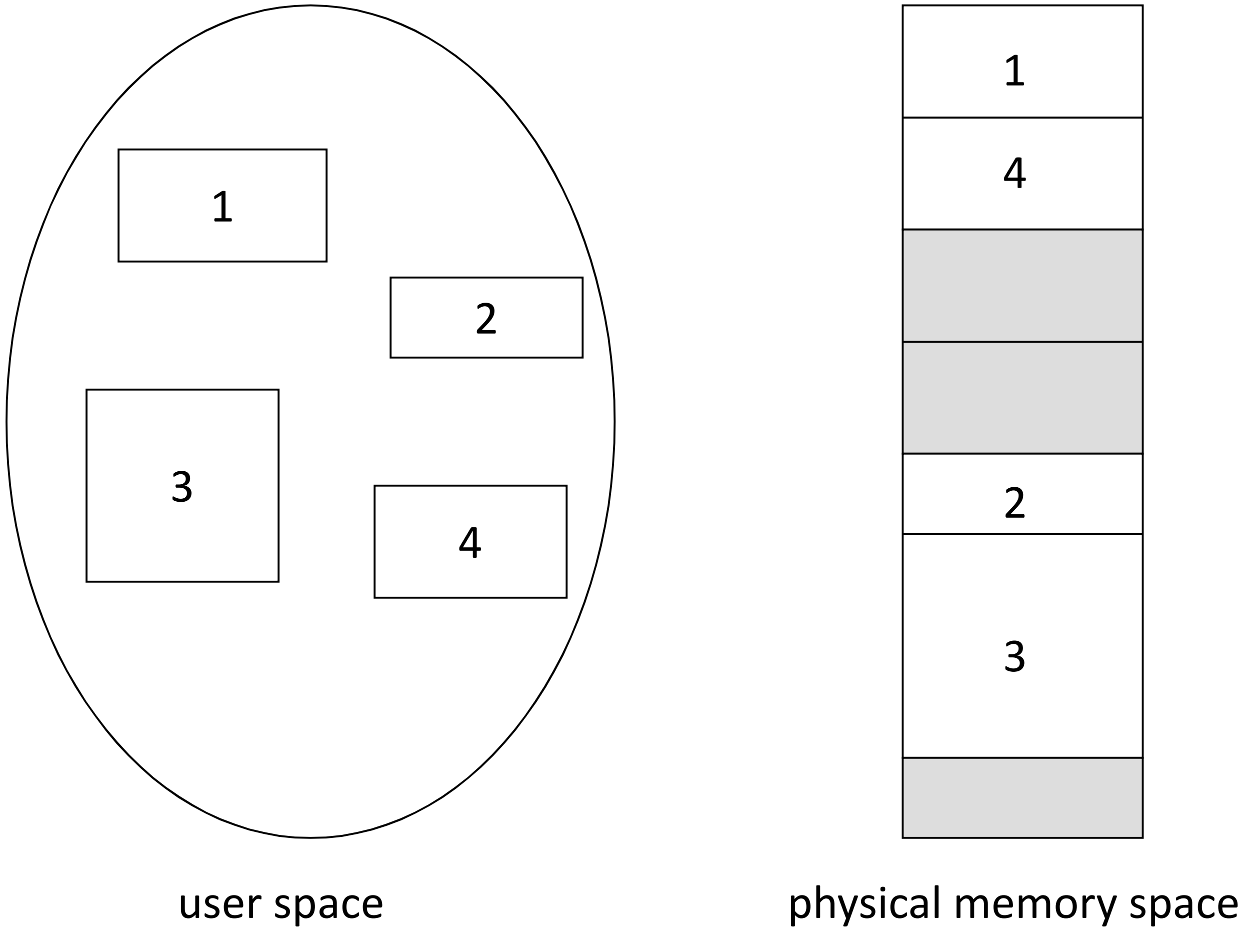
Simple Segmentation

Dynamics of Simple Segmentation

- Each program is subdivided into blocks of non-equal size called segments.
- When a process gets loaded into main memory, its different segments can be located anywhere.
- Each segment is fully packed with instructions/data; no internal fragmentation.
- There is external fragmentation; it is reduced when using small segments.
- In contrast with paging, segmentation is visible to the programmer.
  - provided as a convenience to organize logically programs (e.g., data in one segment, code in another segment).
  - must be aware of segment size limit.
- OS maintains a *segment table* (段表) for each process. Each entry contains:
  - the starting *physical addresses* of that segment
  - the length of that segment for protection.

Simple Segmentation

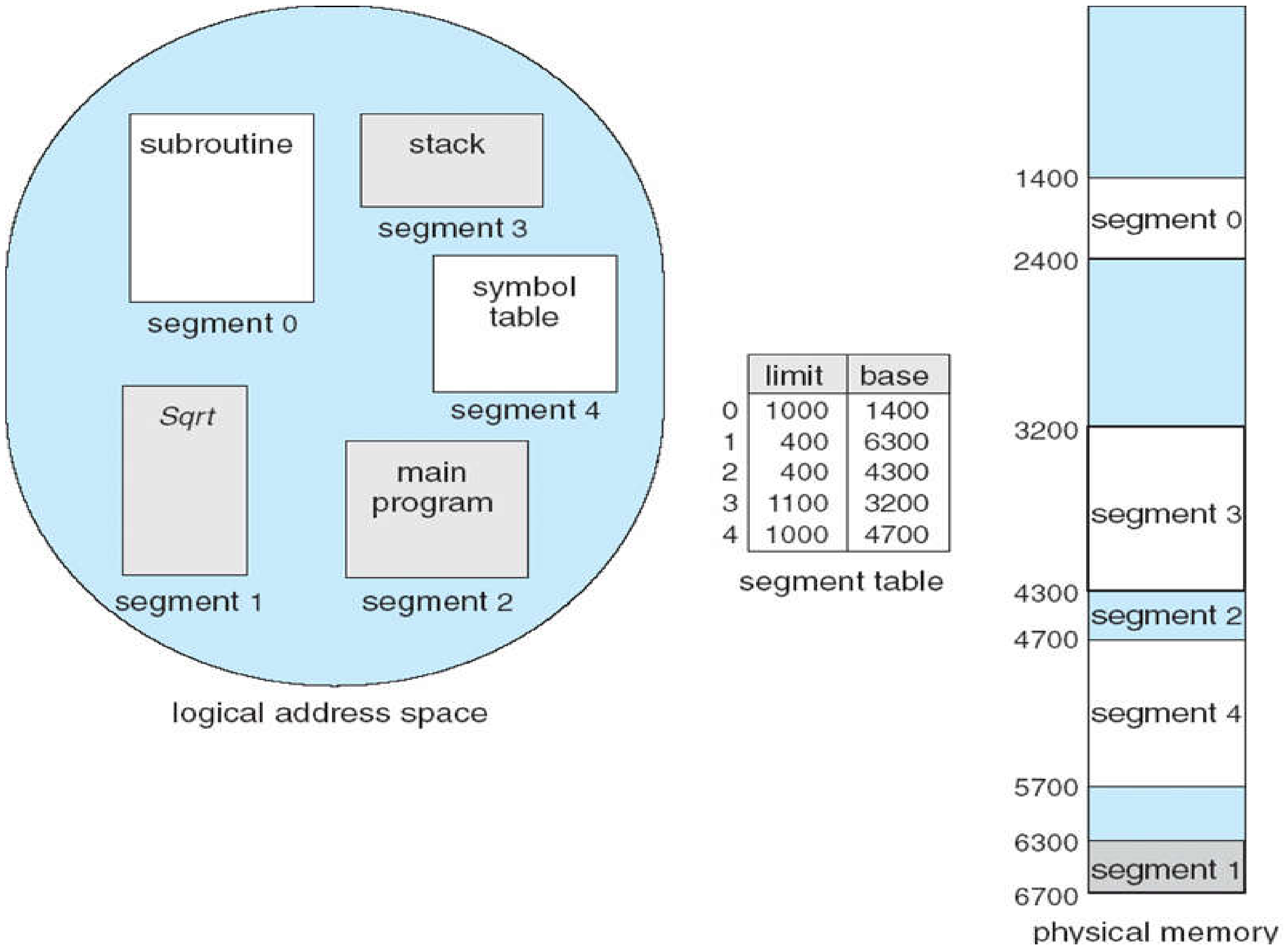
- **Logical view of simple segmentation**



Simple Segmentation

Address Translation in segmentation

- Example.



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Simple Segmentation

Segmentation Architecture

- A logical address consists of a two-tuple:

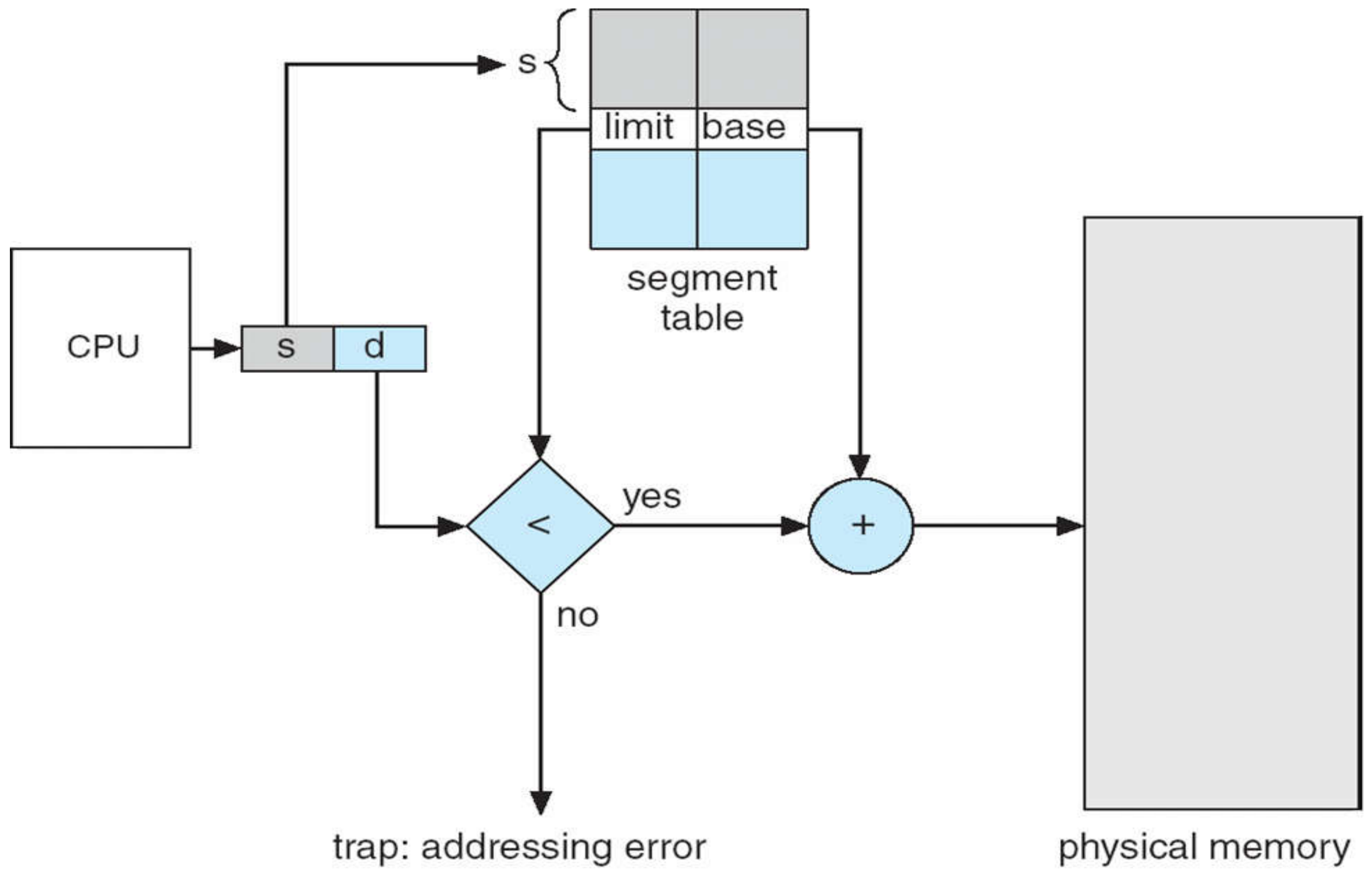
  <segment_number, offset>,

  段号 偏移

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

Simple Segmentation

- **Segmentation Architecture**
  - Address Translation Architecture - Segmentation Hardware.



Segmentation Architecture

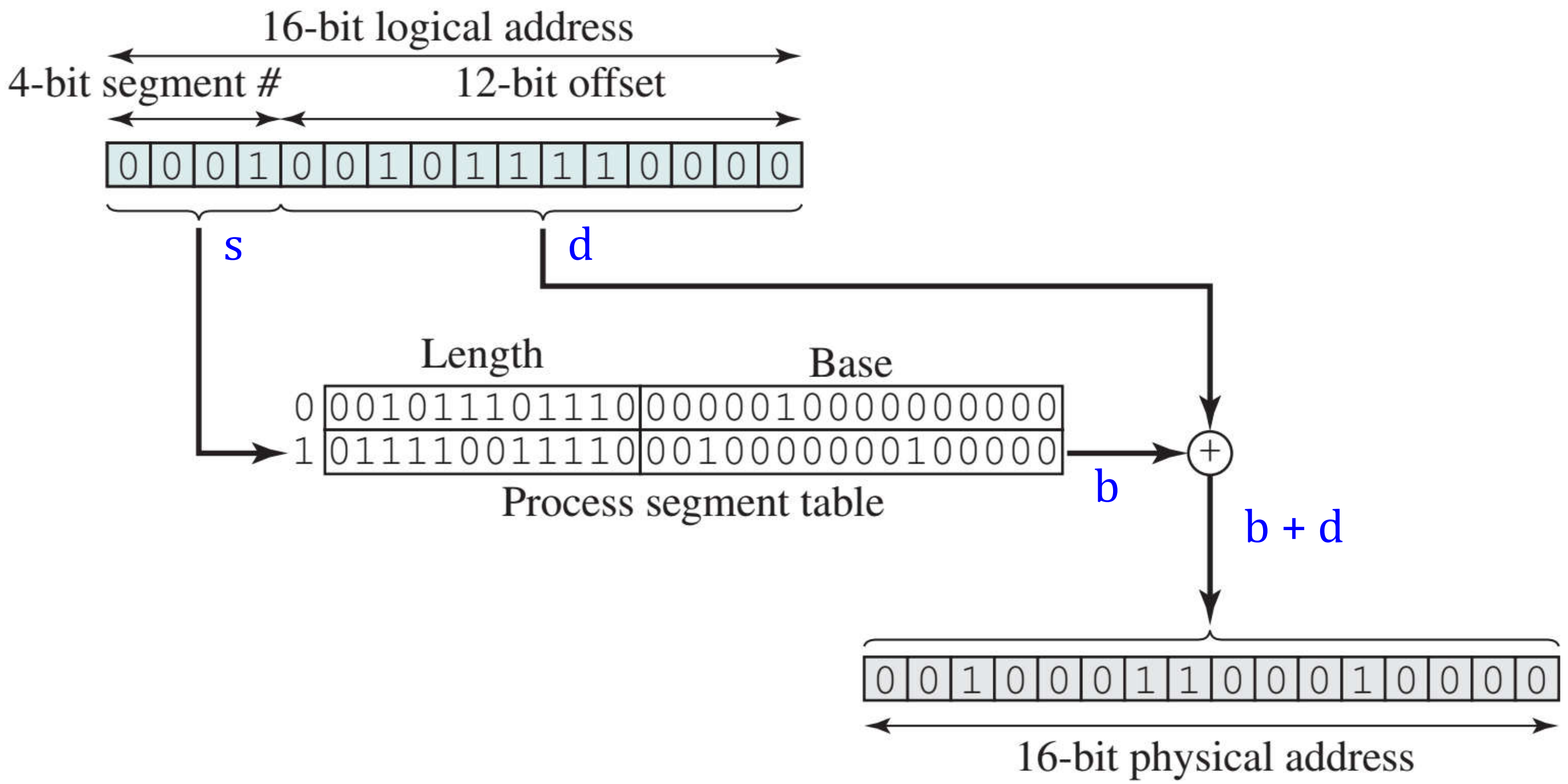
- Address Translation in Segmentation
  - When a process enters the Running state, a dedicated register gets loaded with the starting address of the process's segment table.
  - Presented with a logical address

    <segment_number, offset> = <s, d>,

    the CPU indexes (with s) the segment table to obtain the starting physical address b and the length limit l of that segment.
  - The physical address is obtained by adding d to b.
    - The hardware also compares the offset/displacement d with the limit l of that segment to determine if the address is valid.

Simple Segmentation

- **Segmentation Architecture**
  - Address Translation in Segmentation
    - Example.



Simple Segmentation

Segmentation Architecture

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges.
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.

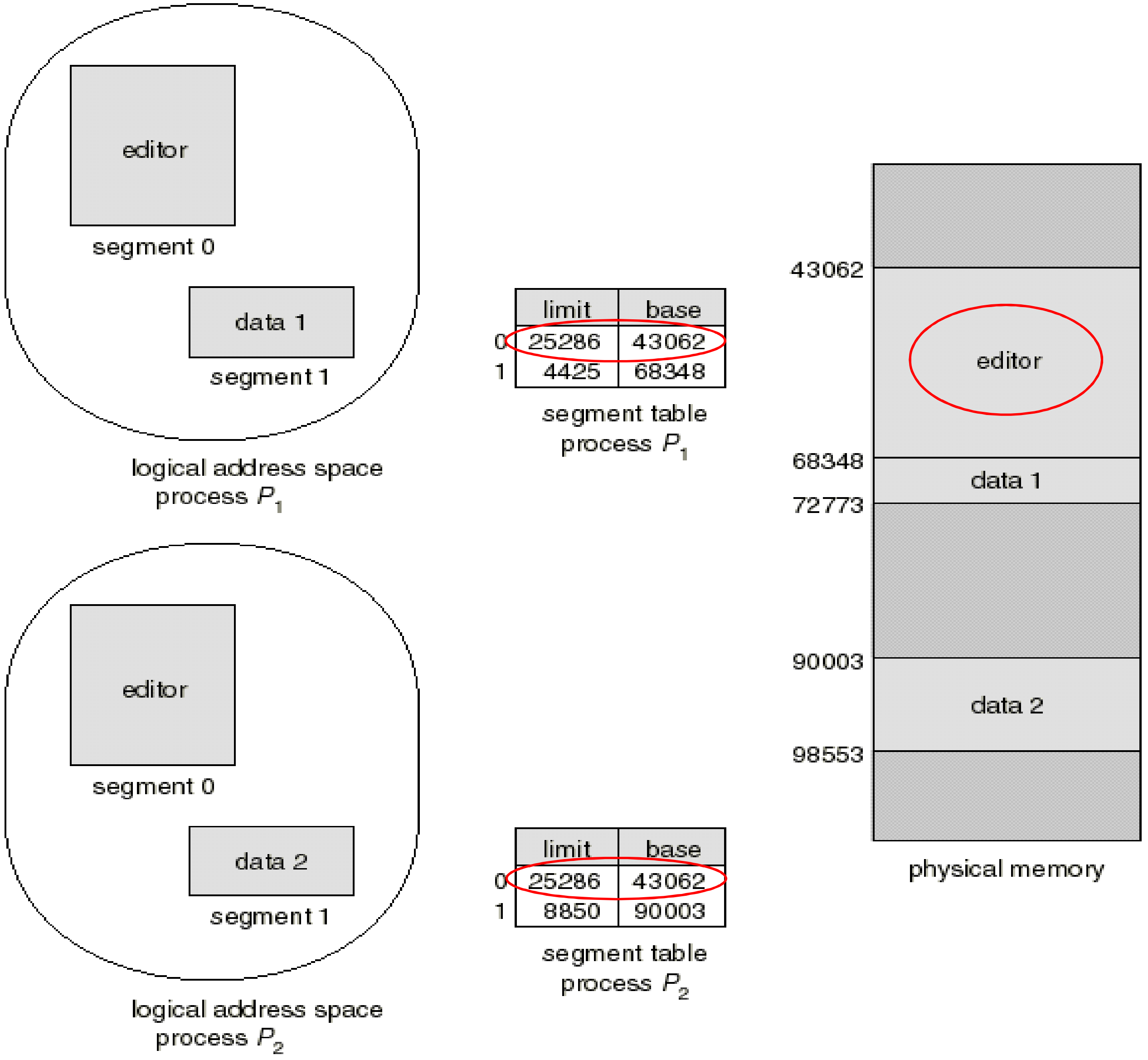
Simple Segmentation

- **Shared Segments**
  - Sharing in Segmentation Systems
    - Segments are shared when entries in the segment tables of 2 different processes point to the same physical locations.
    - Example.
      - The same code of a text editor can be shared by many users.
      - Only one copy is kept in main memory.
    - But each user would still need to have its own private data segment.

Simple Segmentation

Shared Segments

- Example.



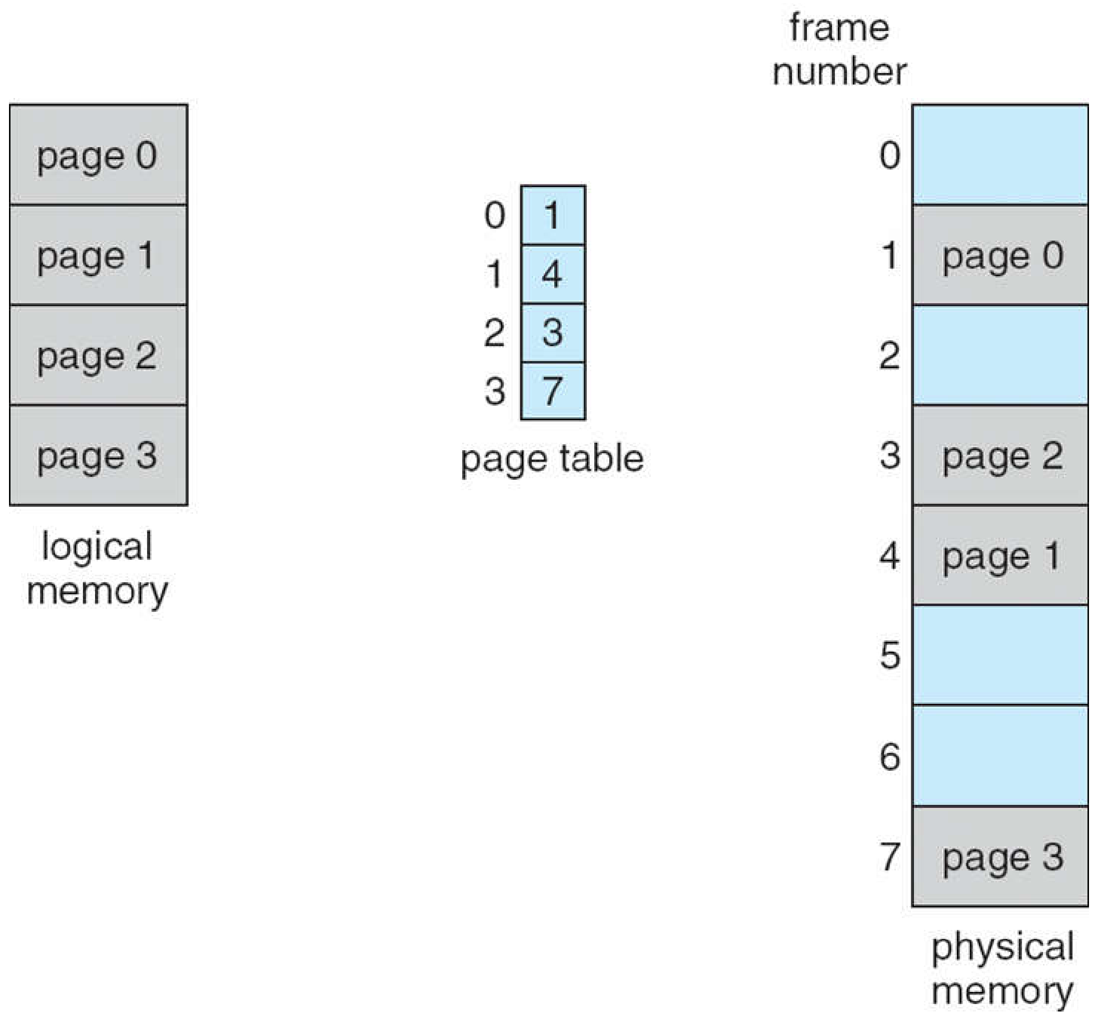
Simple Paging Concepts

- Idea: Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available in size.
    - Avoids external fragmentation.
    - Avoids problem of varying sized memory chunks.
- Divide physical memory into fixed-sized chunks/blocks called *frames* (页帧，页框，物理页), size of power of 2, usually between 512 bytes and 16 MB.
- Divide logical memory into blocks of the same size as physical frames, called *pages* (页，页面，逻辑页).
- Process pages can thus be assigned to any free frames in main memory; a process does not need to occupy a contiguous portion of physical memory. To run a program of size *n* pages, we need to find *n* free frames of physical memory and load the program.
    - So we need to use a *free-frame list* to keep track of all free frames in physical memory.
    - And we need to set up a *page table* (页表) to translate logical pages to physical frames.
- Internal fragmentation possible only for the page at the end of program.

Simple Paging

Simple Paging Concepts

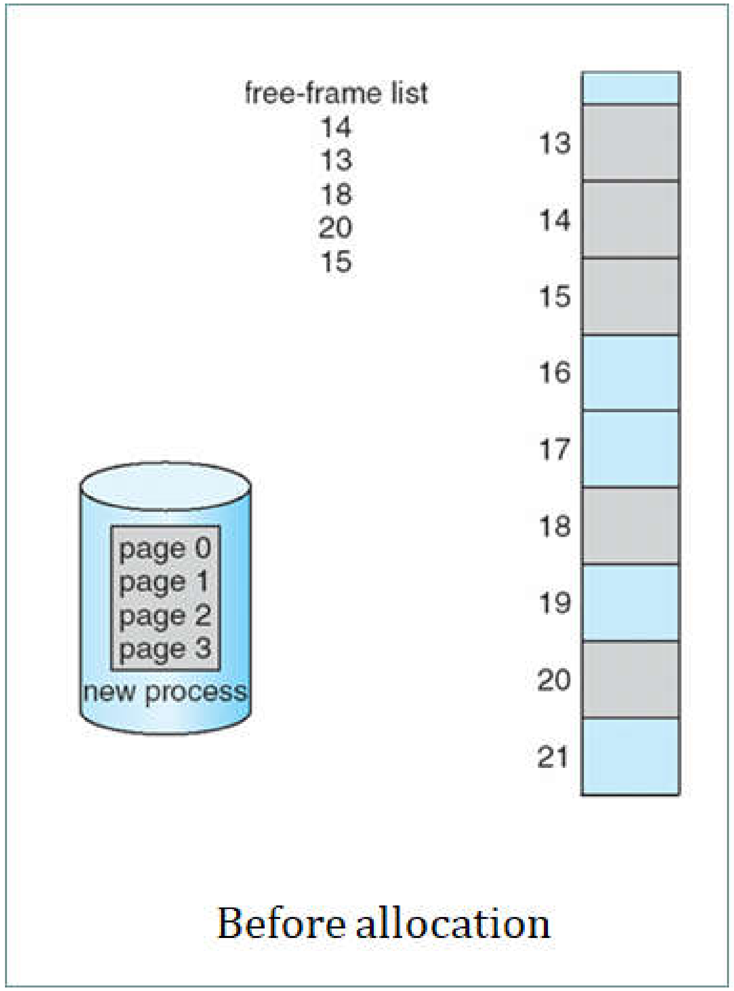
- Paging Example.

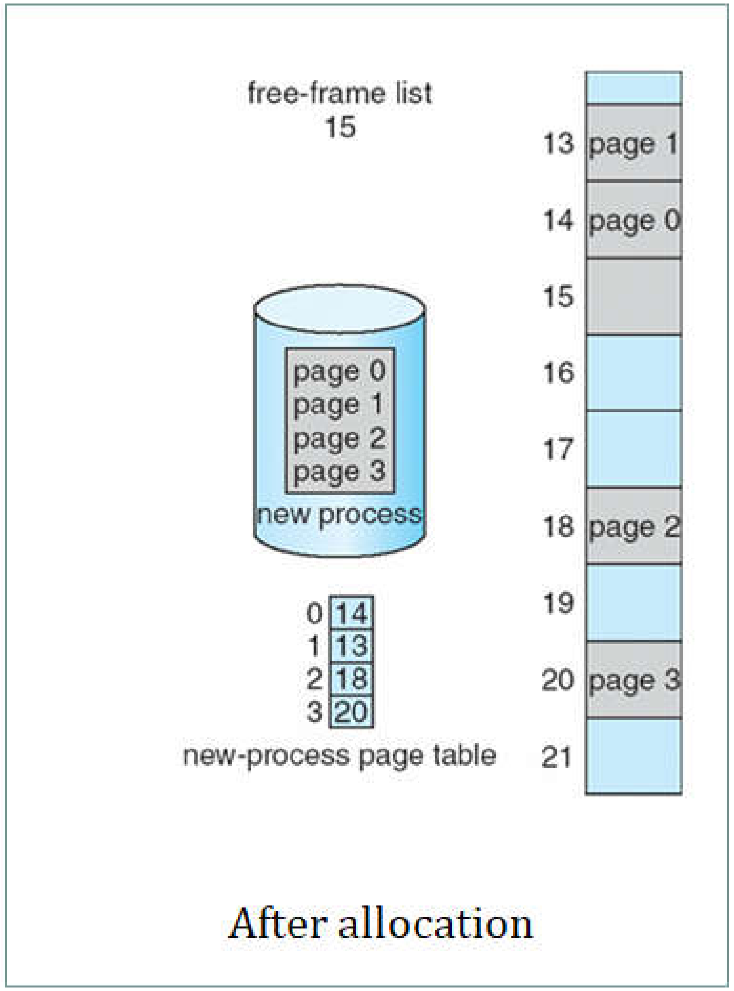


Simple Paging

Simple Paging Concepts

- Free-frame list example.

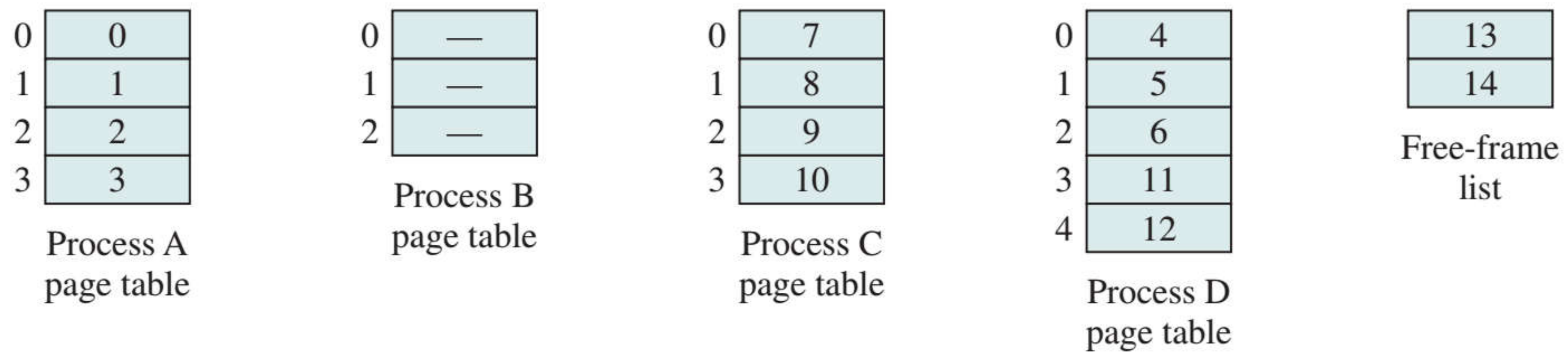




Simple Paging

Simple Paging Concepts

- OS now needs to maintain in main memory a *page table* for each process.
    - Each entry of a page table consists of the frame number where the corresponding page is physically located.
    - The corresponding page table is indexed by the page number to obtain the frame number.
- A *free frame table/list*, of available frames, is also maintained.
    - Example.



Simple Paging

Calculating Internal Fragmentation

- Example.
  - Page/frame size: 2,048 bytes
  - Suppose some process has size: 72,766 bytes
    - $72766 = 2048 \times 35 + 1{,}086$.
    - Need to allocate 36 frames to the process.
    - Internal fragmentation: 2,048 - 1,086 = 962 (bytes)
  - Worst case fragmentation: 2048 - 1 (byte)
    - For processes having size of $2048p + 1$ bytes.
  - On average internal fragmentation: 1/2 frame size.
- So small frame sizes desirable?
  - But each page table entry takes memory to track.
  - Page sizes growing over time:
    - Linux page sizes: 4KB / 2MB

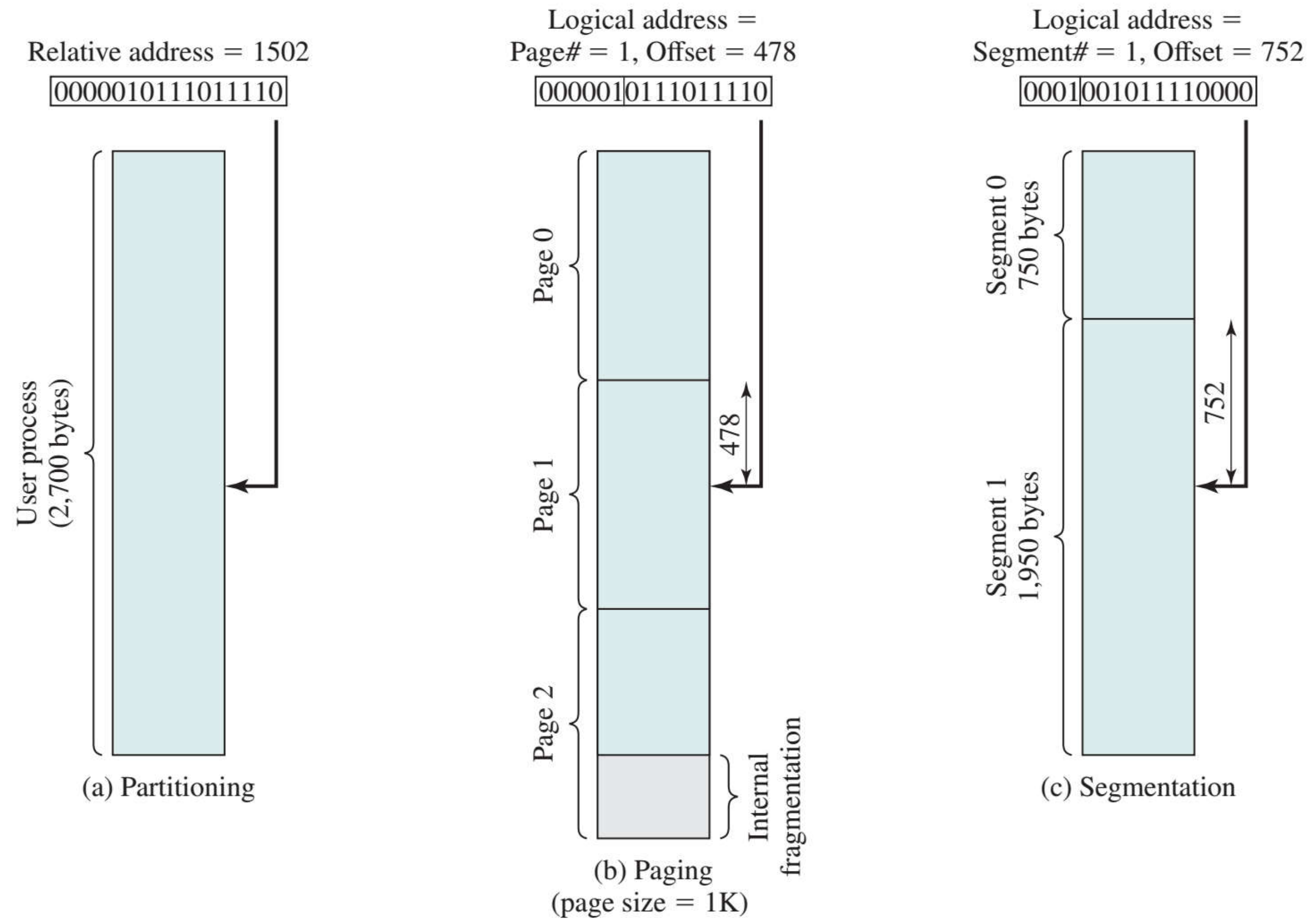
```
$getconf PAGESIZE
$cat /proc/meminfo | grep Huge
```

    - Solaris page sizes: 8KB / 4MB.

Simple Paging

Logical Address in Paging

- The logical address becomes a relative address when the page size is a power of 2.
  - Comparison of Partitioning, Paging and Segmentation.



Simple Paging

Logical Address in Paging

- The logical address becomes a relative address when the page size is a power of 2.
  - Example.
    - If 16-bit addresses are used and page size = 1K, we need 10 bits for offset and have 6 bits available for page number.
    - Then the 16 bit address, obtained with the 10 least significant bits as offset and 6 most significant bits as page number, is a location relative to the beginning of the process.

Simple Paging

Logical Address in Paging

- Within each program, each logical address must consist of a page number and an offset/displacement within the page

  <page_number, offset> = <p, d>.

- A dedicated register always holds the starting physical address of the page table of the currently running process.
- Presented with the logical address

  <page_number, offset> = <p, d>,

  the processor accesses the page table to obtain the physical address

  <frame_number, offset> = <f, d>.

Simple Paging

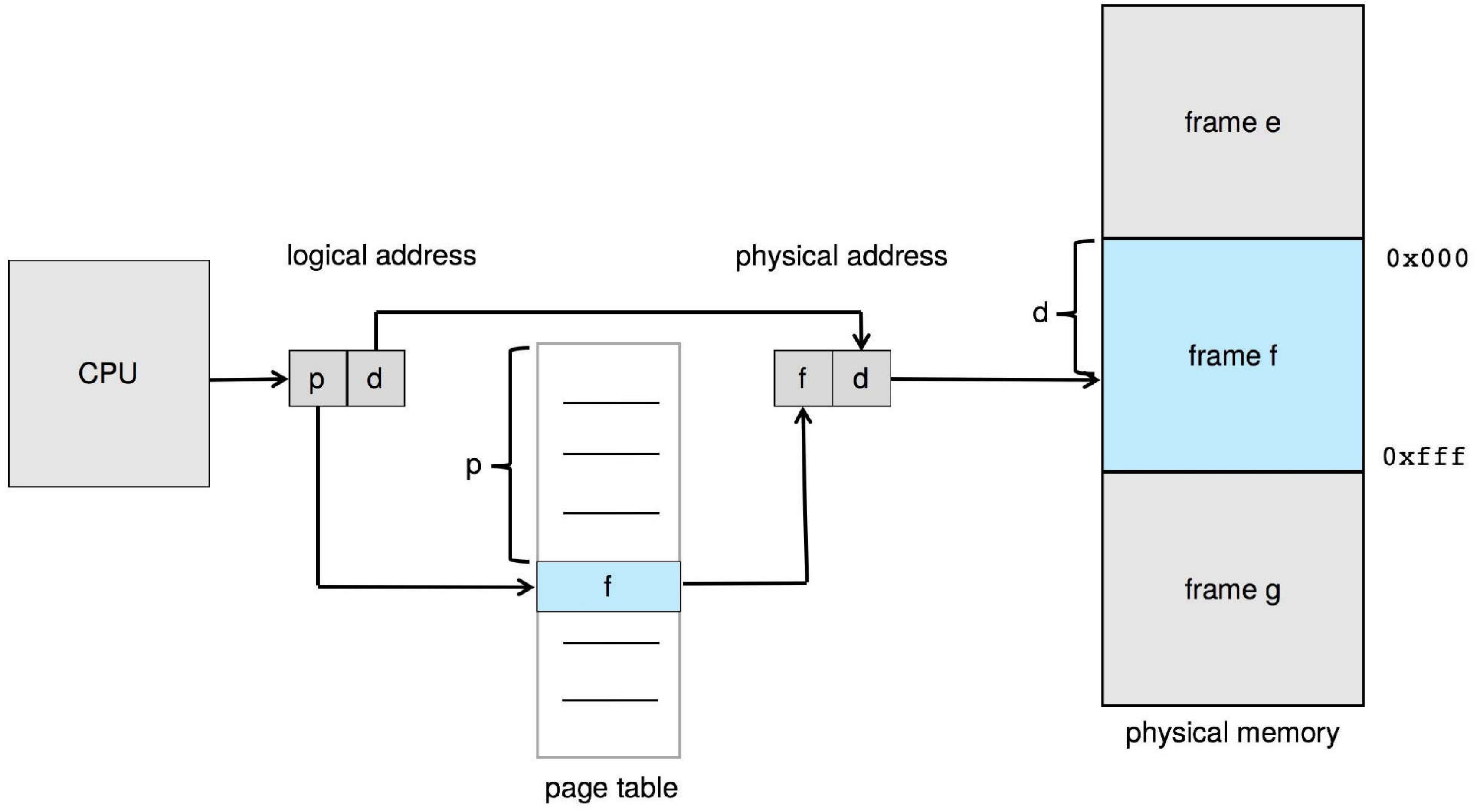
Address Translation Scheme

- Logical address generated by CPU is divided into two parts:
  - Page number (p) – used as an index into a page table which contains the base address of each page in physical memory.
  - Page offset/displacement (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 ($n < m$),

page number	page offset
p	d
$m - n$	*n*

- By using a page size of a power of 2, the pages are *invisible* to the programmer, compiler/assembler, and the linker.
- Address translation at run-time is then easy to implement in hardware:
  - Logical address <p, d> gets translated to physical address <f, d> by indexing the page table with p and *appending* the same offset d to the frame of number f.

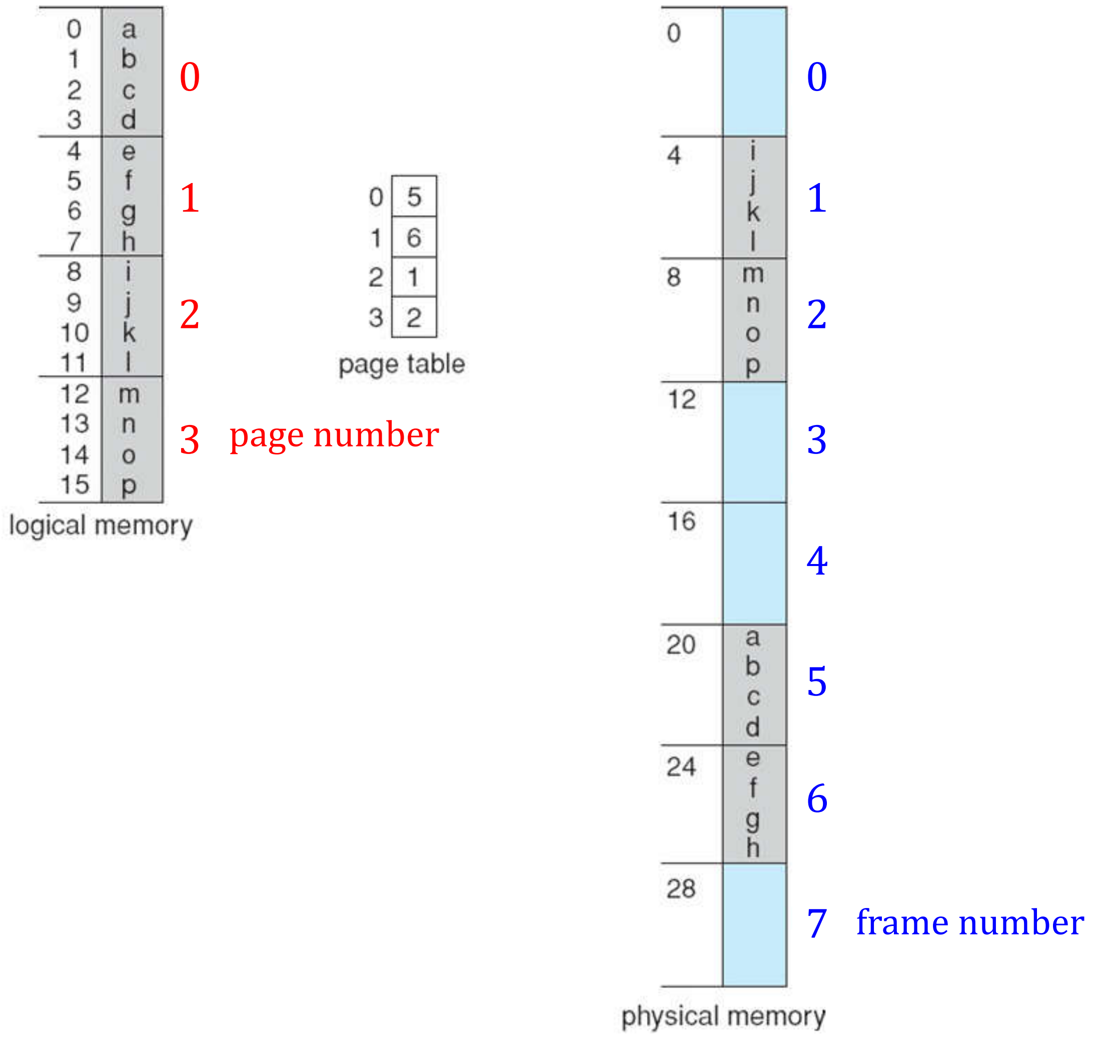
Address Translation Architecture



Simple Paging

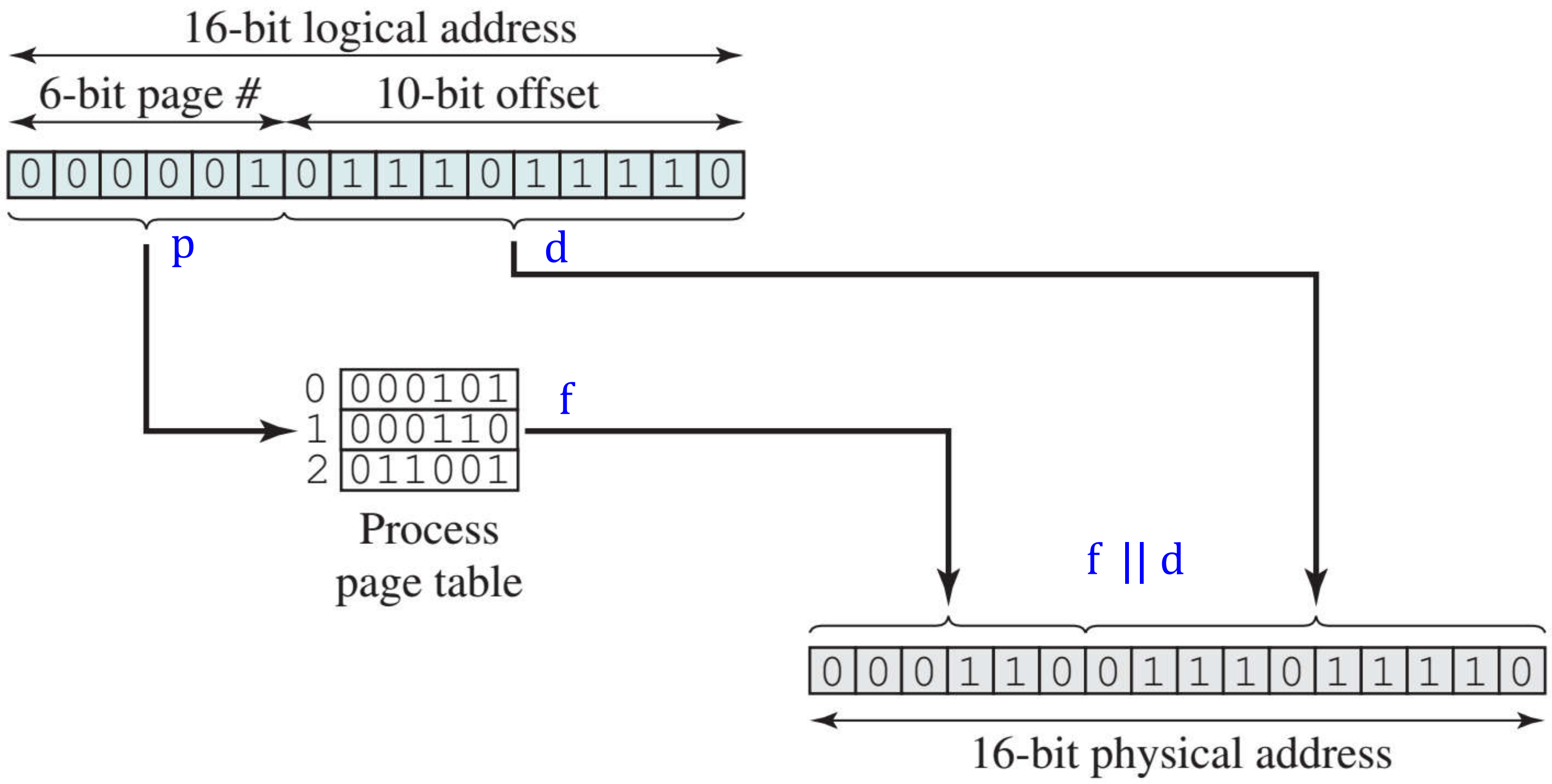
Address Translation Architecture

- Example.
  - $m = 4$, $n = 2$, page size = $2^2 = 4$ (bytes), memory size = 32 (bytes).



Simple Paging

- **Address Translation Architecture**
  - Example.
    - m = 16, n = 10, page size = 2^{10} = 1024 (bytes).



Page Tables

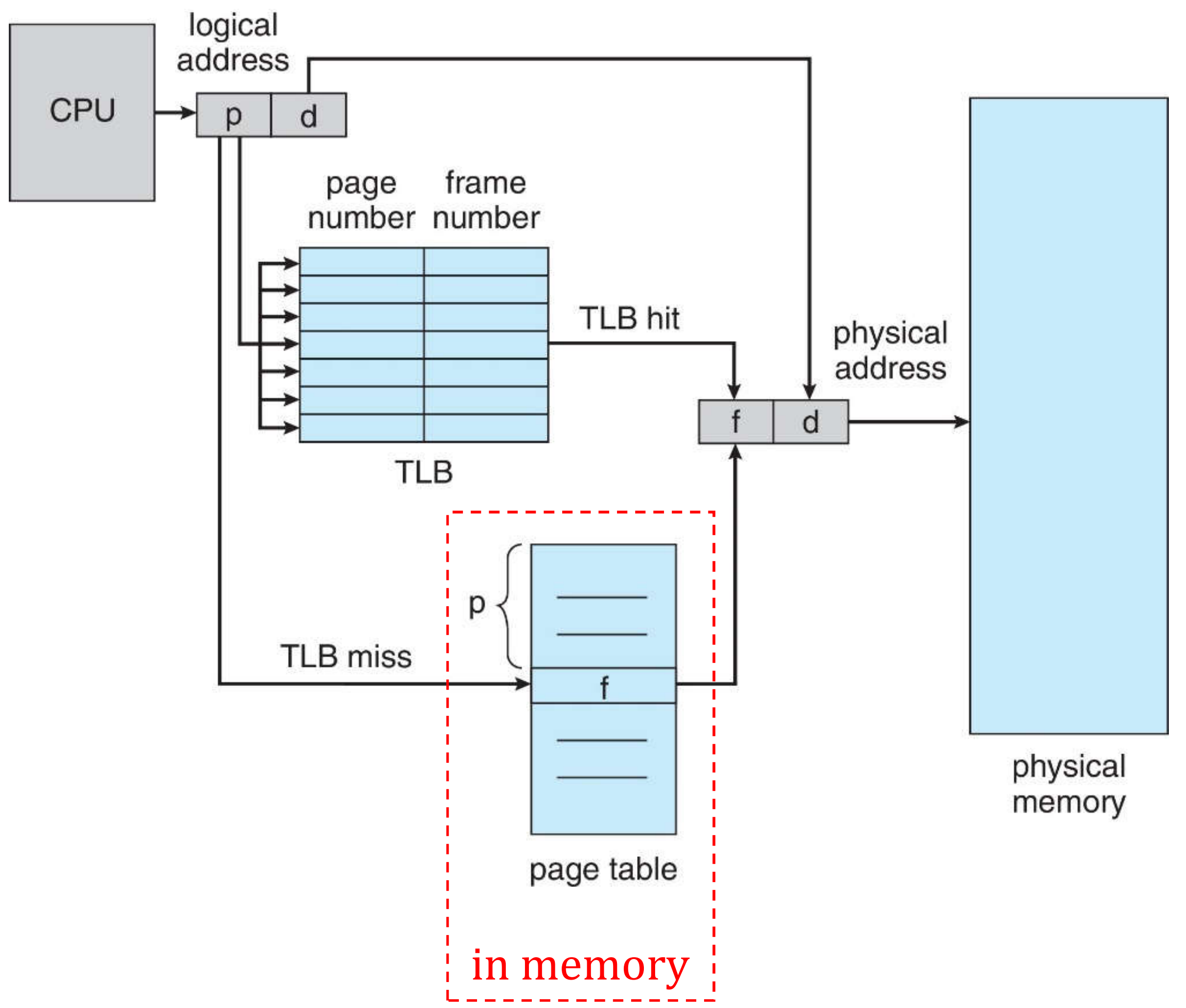
Implementing Page Table

- If we keep Page Table in main memory
  - *Page-table base register* (*PTBR*) points to the location of the page table in memory.
  - *Page-table length register* (*PTLR*) indicates the size of the page table.
  - Every data/instruction access requires *two memory accesses.*
    - one for the page table, and one for the data/instruction
- If we keep Page Table in hardware (in MMU?)
  - However, page table can be large – too expensive.
- The two memory accesses problem can be solved by combining these two mechanisms.
  - Use a special fast-lookup hardware cache called *Associative Memory* (Registers) or *Translation Look-aside Buffer* (TLB, 快表) – enables fast parallel search.
  - Address translation <p, d>
    - If p is in associative register, get frame number out.
    - Otherwise get frame number from page table in memory.

Page Tables

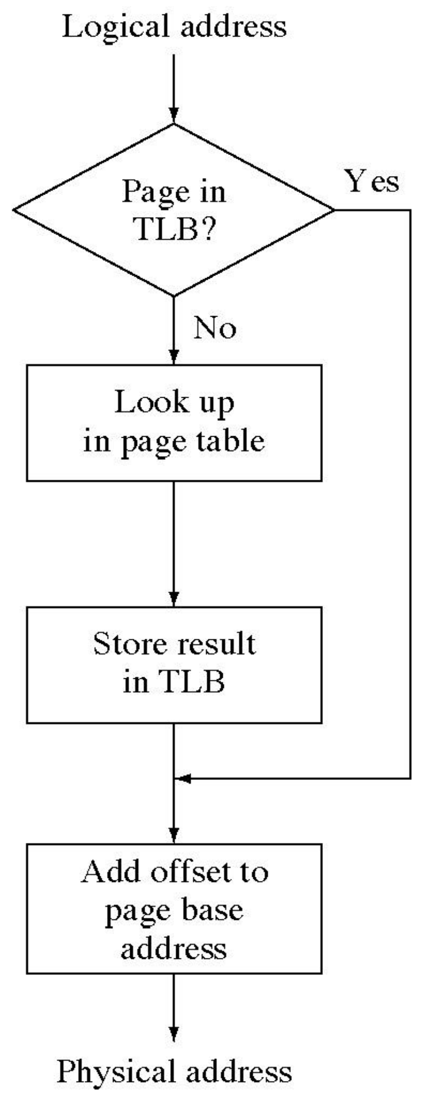
Implementing Page Table

- Paging Hardware with TLB.



Implementing Page Table

- Paging Hardware with TLB.
  - TLB Flow Chart.



Page Tables

Implementing Page Table

- Paging Hardware with TLB
  - TLB takes advantage of the *Locality Principle* (局部性原理).
  - TLB uses associative mapping hardware to simultaneously interrogate all TLB entries to find a match/hit on page number.
  - TLB hit rates are 90%+.
  - TLB must be flushed (erased) each time a new process enters the running state.
    - There is only one global TLB serving for all processes.
  - TLB information maybe kept/loaded in/from process context.

Page Tables

Implementing Page Table

- Paging Hardware with TLB
  - Each associative (memory) lookup time = ε time unit.
    - time unit: memory access time
    - can be < 10% of (far faster than) memory access time
  - Hit ratio = α
    - percentage of times that a page number is found in the associative memory;
    - related to number of associative registers.
  - Effective Access Time (EAT)

    $$EAT = \alpha(\varepsilon + 1) + (1 - \alpha)(\varepsilon + 2) = 2 + \varepsilon - \alpha.$$

    Unit: memory access time.
  - EAT is between $1 + \varepsilon$ and $2 + \varepsilon$ access times.
    - should be closer to 1.

Page Tables

- **Implementing Page Table**
  - Paging Hardware with TLB
    - Effective Access Time (EAT)

      EAT = $\alpha(\varepsilon + 1) + (1 - \alpha)(\varepsilon + 2) = 2 + \varepsilon - \alpha$.

      Unit: memory access time.
    - Examples: Assume that memory cycle time is 100ns and ε = 20ns for TLB search.
      - Consider α = 80%. Then

        EAT = 0.80 x (20 + 100) + 0.20 x (20 + 200) = 140(ns), or

        EAT = 2 x 100 + 20 – 1 x 100 x 0.8 = 140(ns)

        So it is (140 – 100) / 100 = 40% slowdown in memory access time.
      - Consider more realistic hit ratio α = 98%. Then

        EAT = 0.98 x 120 + 0.02 x 220 = 122(ns), or

        EAT = 2 x 100 + 20 – 1 x 100 x 0.98 = 122(ns)

        So it is only (122 – 100) / 100 = 22% slowdown in memory access time.

Page Tables

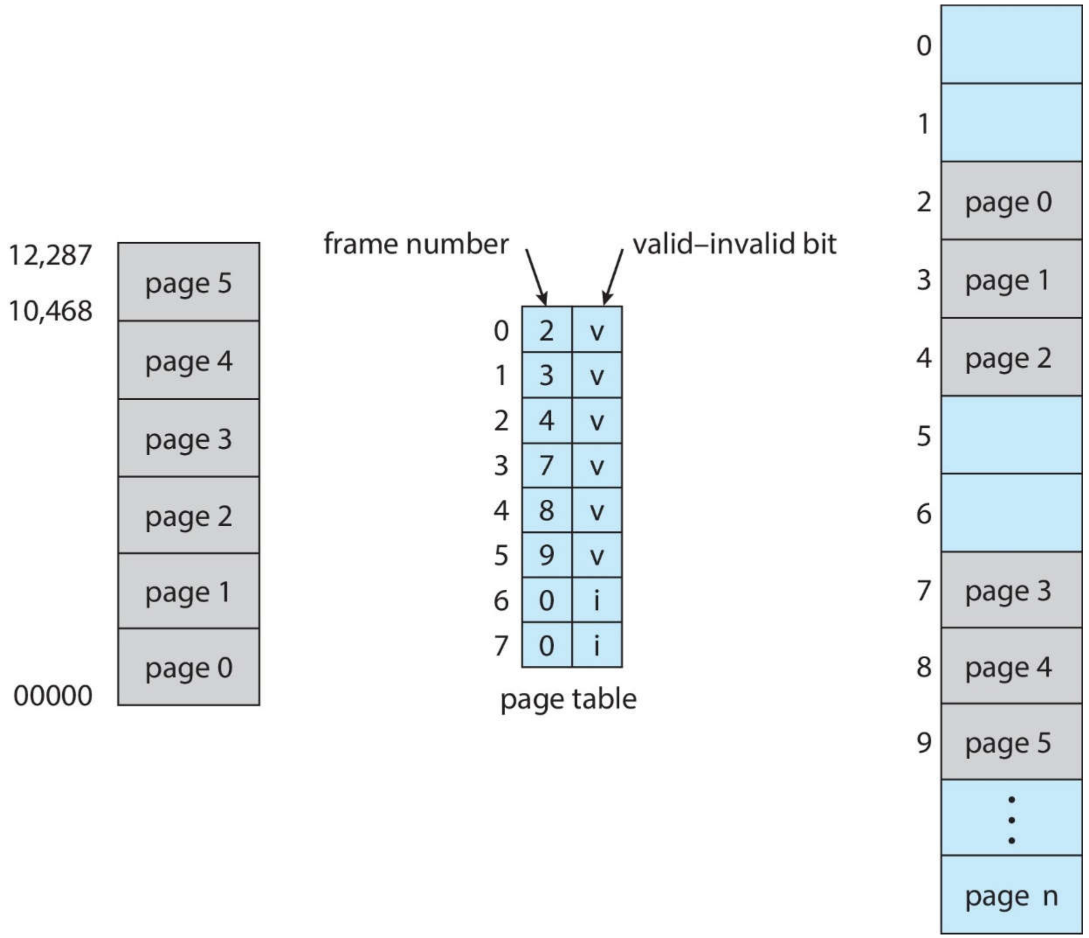
- **Implementing Page Table**
  - Paging Hardware with TLB
    - Advanced TLB Aspects
      - Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process:
        - Otherwise need to flush at every context 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* (not removed from the TLB) for permanent fast access.

Page Tables

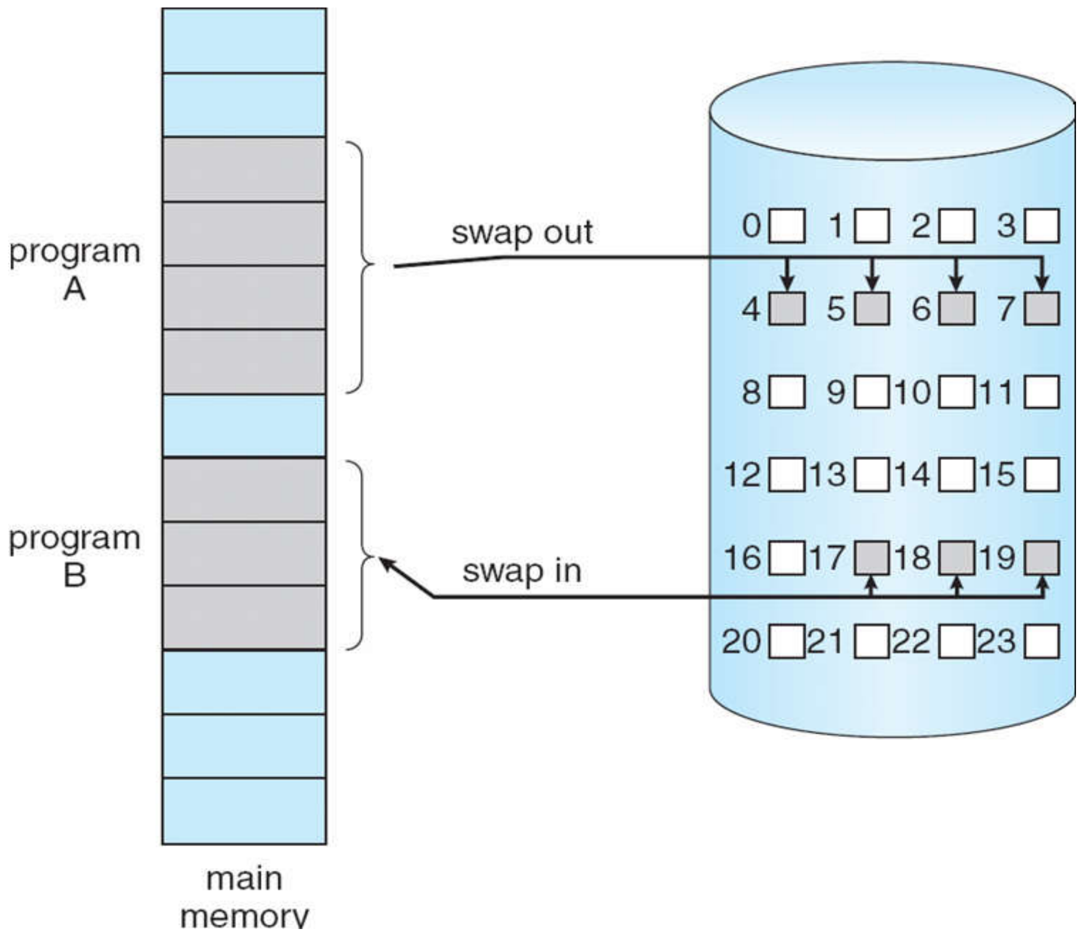
- **Memory Protection**
  - Memory protection is implemented by associating protection bit with *each frame* to indicate if read-only or read-write access is allowed.
    - can also add more bits to indicate page execute-only, and so on.
  - Valid-invalid bit is attached to each entry in the page table.
    - “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.
    - Or use *page-table length register* (PTLR)
  - Any violations result in a trap to the kernel.

Page Tables

- **Memory Protection**
  - Valid (v) or Invalid (i) Bit in Page Table.



Transfer of a Paged Memory to Contiguous Disk Space

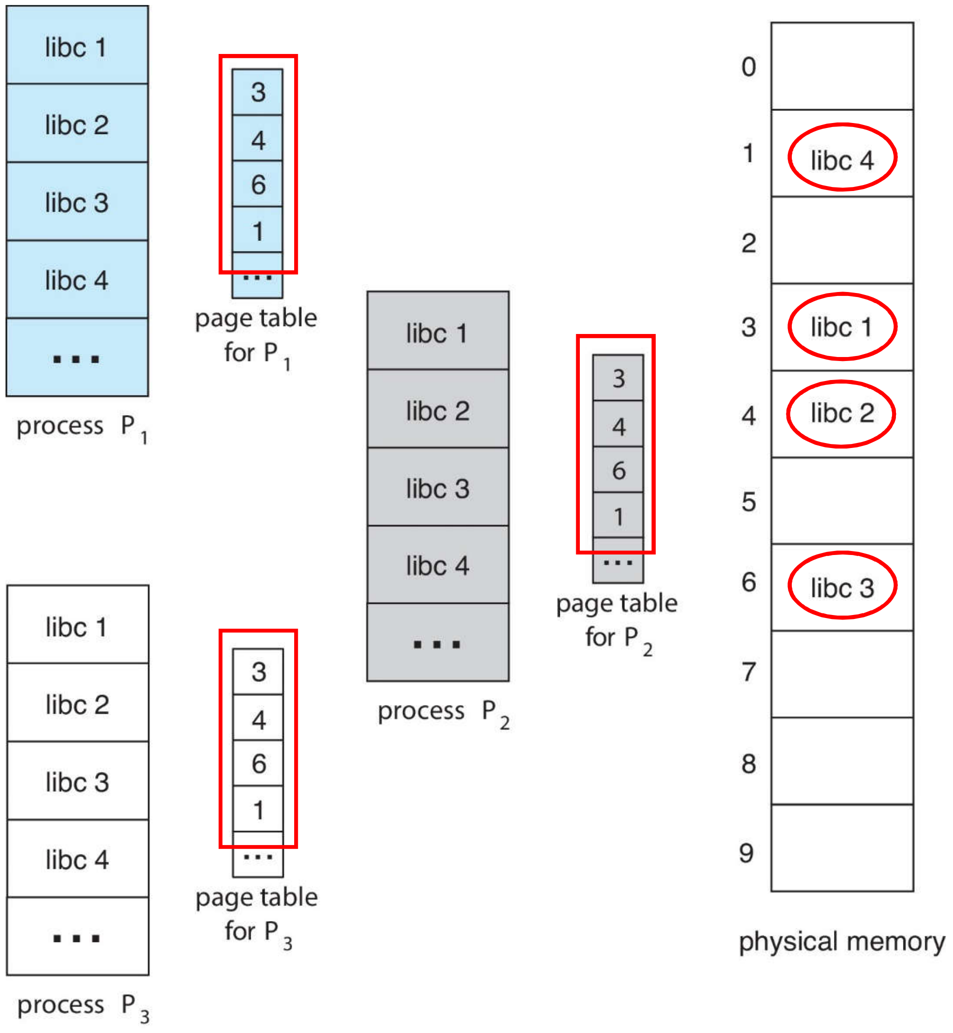


Page Tables

- **Shared Pages**
  - Shared code
    - One copy of read-only (*reentrant*) code shared among processes (e.g., text editors, compilers, window systems).
      - Similar to multiple threads sharing the same process space.
    - Shared code must appear in the same physical location in the logical address spaces of all processes.
  - Private code and data
    - Each process keeps separate copy of code and data.
    - The pages for the private code and data can appear anywhere in the logical address space.

Page Tables

- **Shared Pages**
  - Shared Pages Example.



Page Tables

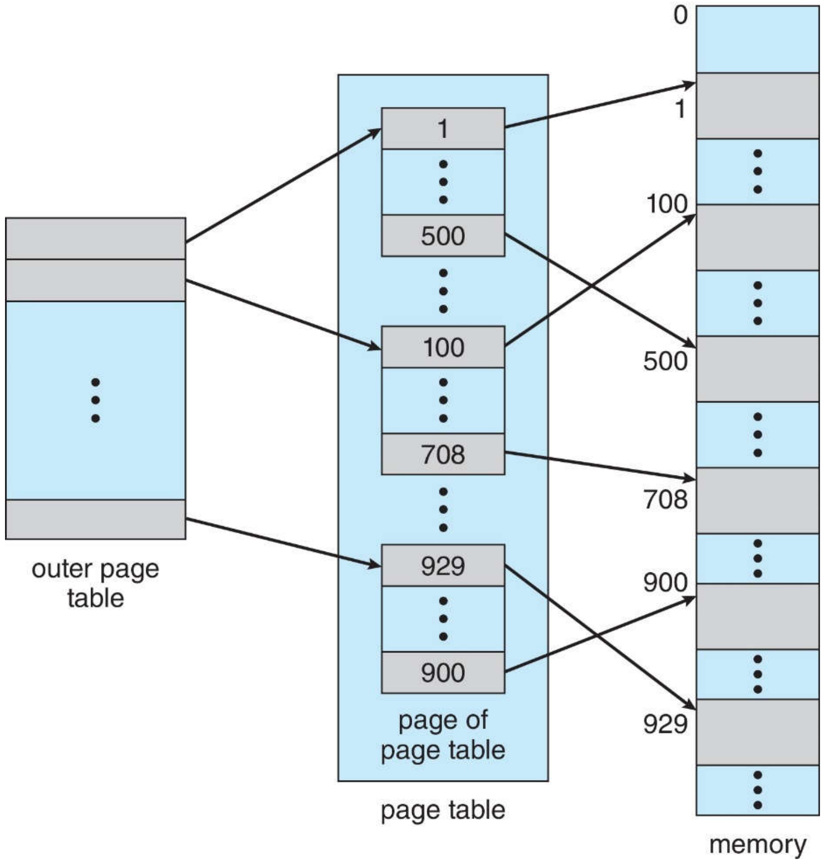
Structures of Page Tables

- Memory structures for paging can get huge using straight-forward methods.
  - Consider a 32-bit logical address space as on modern computers with page size of 4KB (2^{12}).
  - Page table would have 1 million entries ($2^{32-12} = 2^{20}$).
  - If each entry is 4 bytes, then 4MB of physical address space is needed for page table alone.
    - That amount of memory used to cost a lot.
    - It is hard to allocate that contiguously in main memory.
- There are three structures for page tables.
  - Hierarchical Page Tables
  - Hashed Page Tables
  - Inverted Page Tables.

Page Tables

Structures of Page Tables

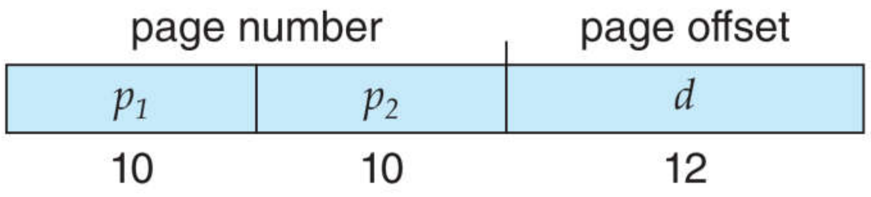
- Hierarchical Page Tables
  - Break up the logical address space into multiple page tables.
    - A simple technique is a two-level page table.



A two-level page-table scheme

Page Tables

- **Structures of Page Tables**
  - Hierarchical Page Tables
    - Consider a 32-bit logical address space.
      - A logical address (on 32-bit machine with 2^{12} = 4K page size) is divided into:
        - a page number consisting of 20 bits
        - a page offset consisting of 12 bits.
      - Since the page table is paged, the page number is further divided into:
        - a 10-bit outer page number
        - a 10-bit inner page offset.
      - Thus, a logical address is as follows:

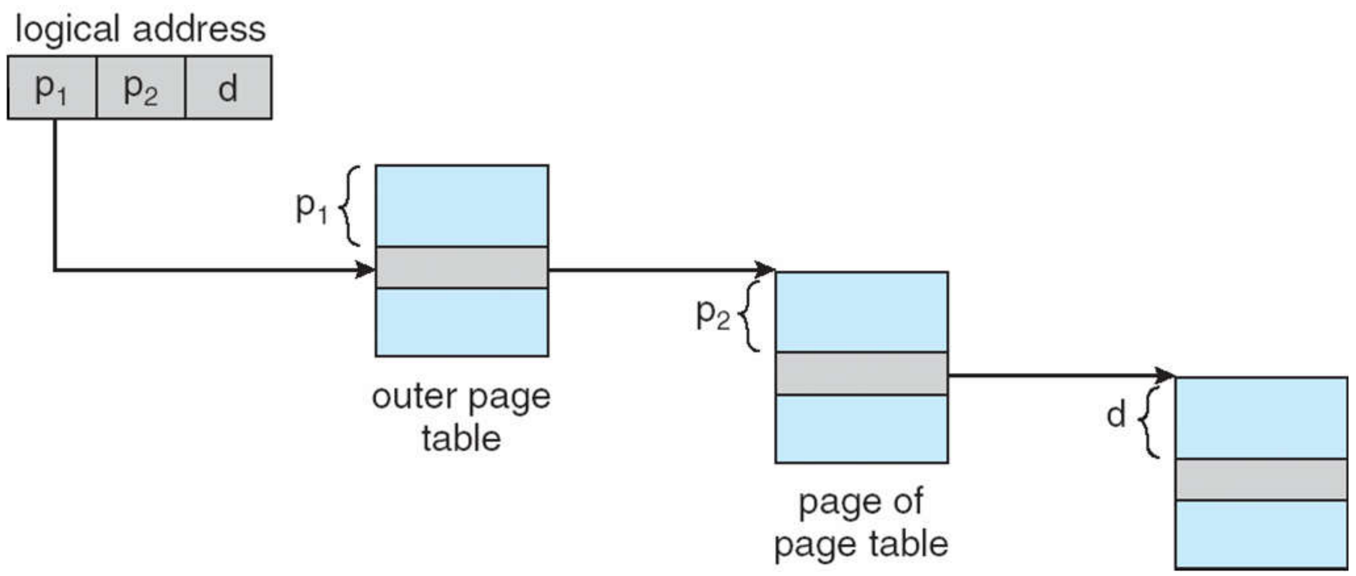


where p_1 is an index into the outer page table, and p_2 is the *displacement* within the page of the inner page table.

- Known as *forward-mapped page table*

Structures of Page Tables

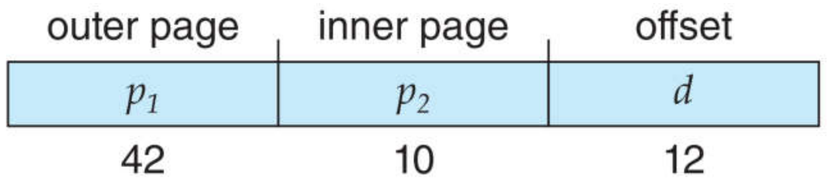
- Hierarchical Page Tables.



Two-level Paging Scheme.

Structures of Page Tables

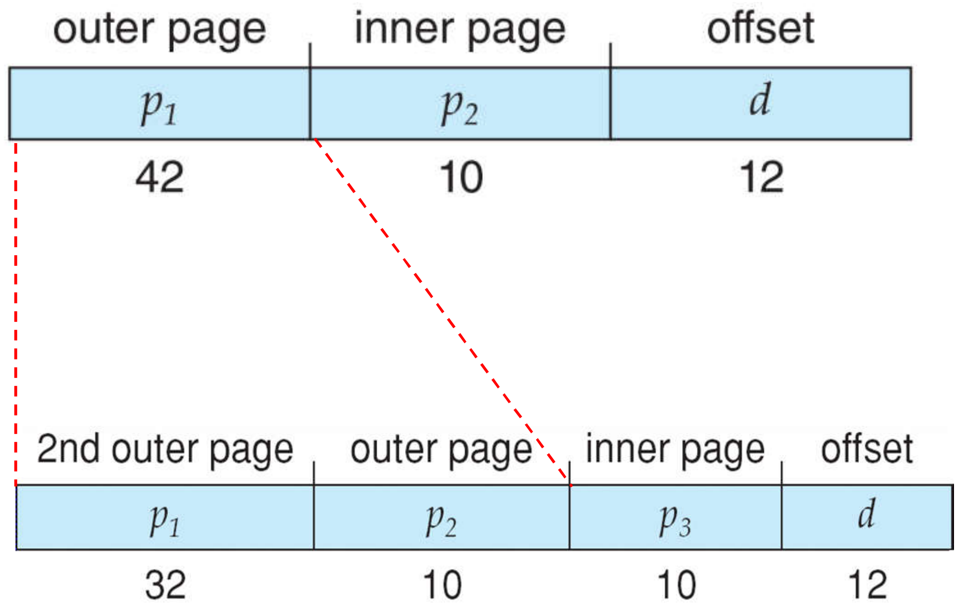
- Hierarchical Page Tables
  - Consider a 64-bit Logical Address Space.
    - Even two-level paging scheme is not sufficient.
    - If page size is 4KB (= 2^{12}), then page table has 2^{52} entries.
      - For two level scheme, inner page tables could have 2^{10} entries of 4 bytes length. Address would look like



      - Outer page table has 2^{42} entries (2^{44} bytes, 4 bytes for each entry).
    - 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.

Page Tables

- **Structures of Page Tables**
  - Hierarchical Page Tables
    - Three-level Paging Scheme.



Page Tables

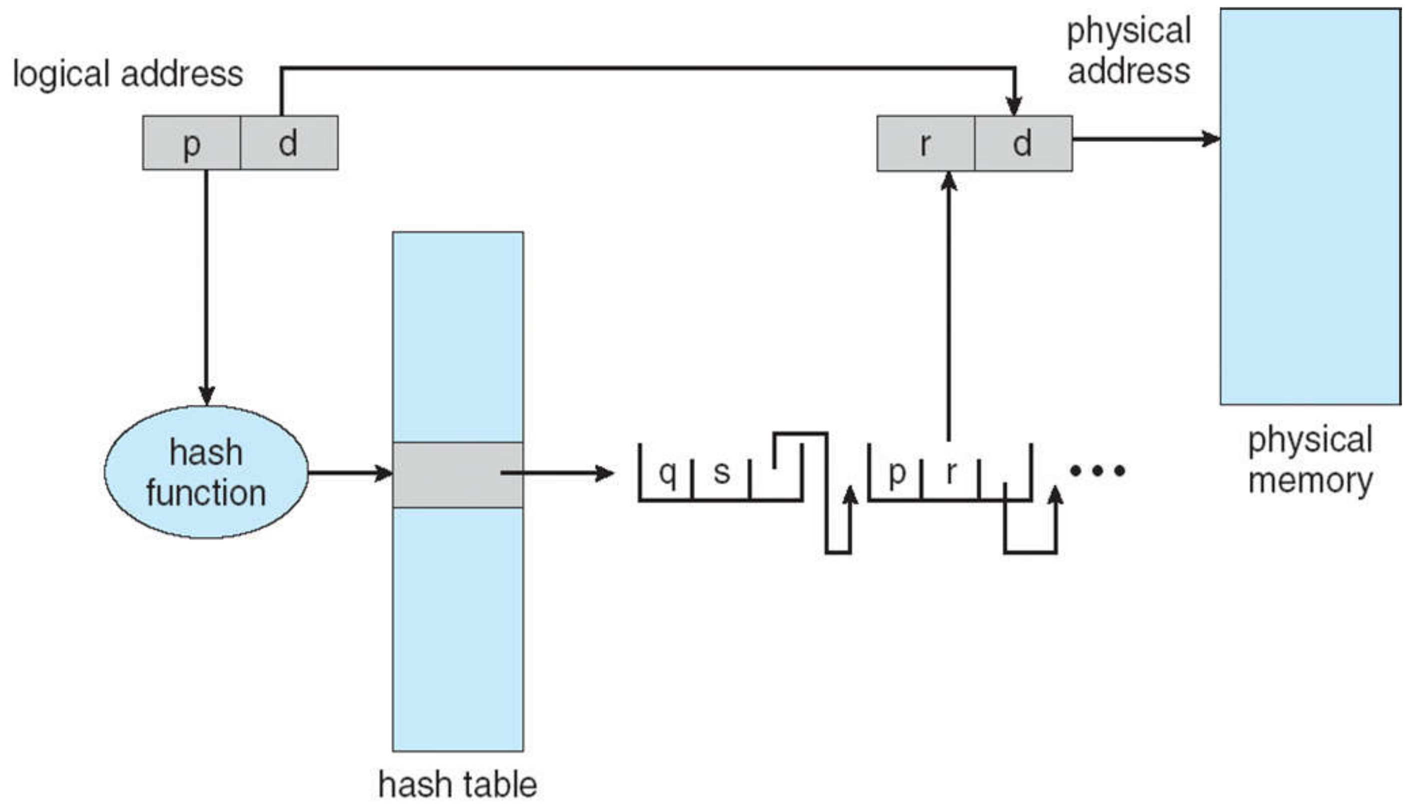
Structures of Page Tables

- 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 of the chain table 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).

Page Tables

Structures of Page Tables

- Hashed Page Tables



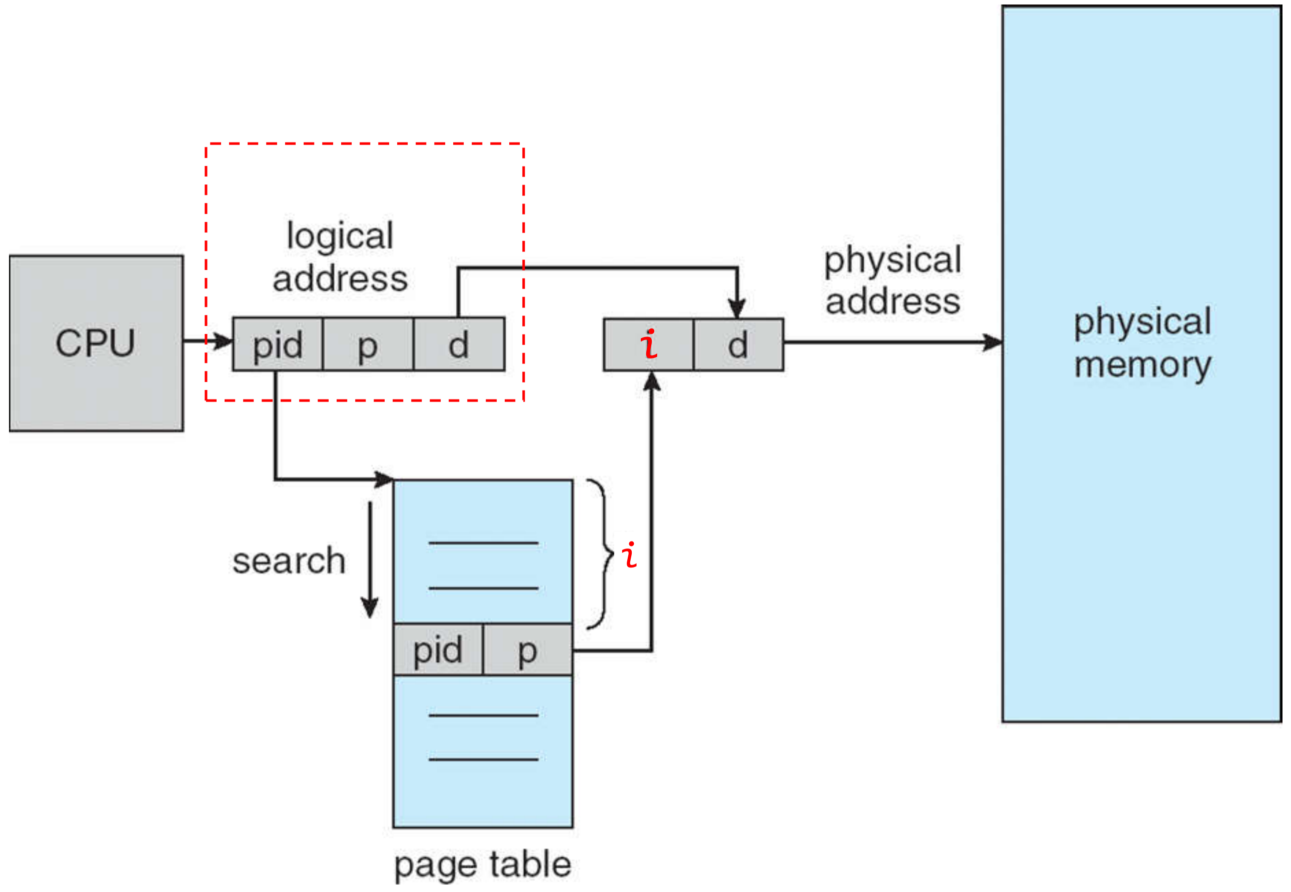
Structures of Page Tables

- Inverted Page Table
  - Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.
  - Entries of an inverted page table consists of the virtual address of the page stored in that real (physical) memory location, with information about the process that owns that page.
    - one entry for each real page of memory
      - *e.g.,* <process-id, page-number>
    - used by 64-bit Ultra SPARC, PowerPC, …
  - This scheme decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
  - 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.

Page Tables

Structures of Page Tables

- Inverted Page Table



Inverted page table architecture

Page Tables

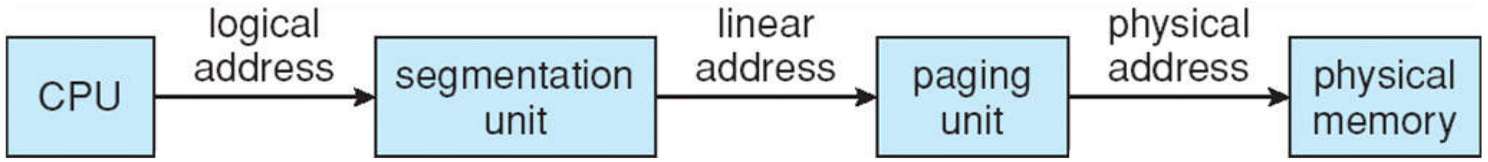
- **Simple Segmentation/Paging Comparison**
  - Segmentation is visible to the programmer whereas Paging is transparent.
  - Naturally supports protection/sharing.
  - Segmentation can be viewed as commodity offered to the programmer to logically organize a program into segments while using different kinds of protection (example: execute-only for code but read-write for data).
  - Segments are variable-size; Pages are fixed-size.
  - Segmentation requires more complicated hardware for address translation than Paging.
  - Segmentation suffers from external fragmentation. Paging only yields a small internal fragmentation.
  - Maybe combine Segmentation and Paging?

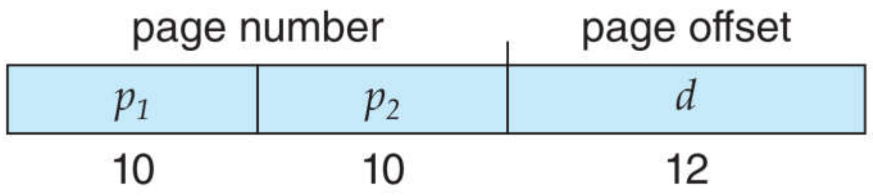
Operating System Examples

- **The Intel 32 Architecture**
  - Dominant industry chips
  - Pentium CPUs are 32-bit and called IA-32 architecture
    - Current Intel CPUs are 64-bit and called IA-64 architecture
  - Many variations in the chips, cover the main ideas here
  - Supports both segmentation and segmentation with paging
    - Each segment can be 4GB
    - Up to 16K segments per process
    - Divided into two partitions
      - First partition of up to 8K segments are private to process (kept in *local descriptor table* (LDT))
      - Second partition of up to 8K segments shared among all processes (kept in *global descriptor table* (GDT))
  - CPU generates logical address
    - Given to segmentation unit which produces linear addresses.
    - Linear address given to paging unit:
      - Which generates physical address in main memory.
      - Paging units form equivalent of MMU.
      - Pages sizes can be 4KB or 4MB.

Operating System Examples

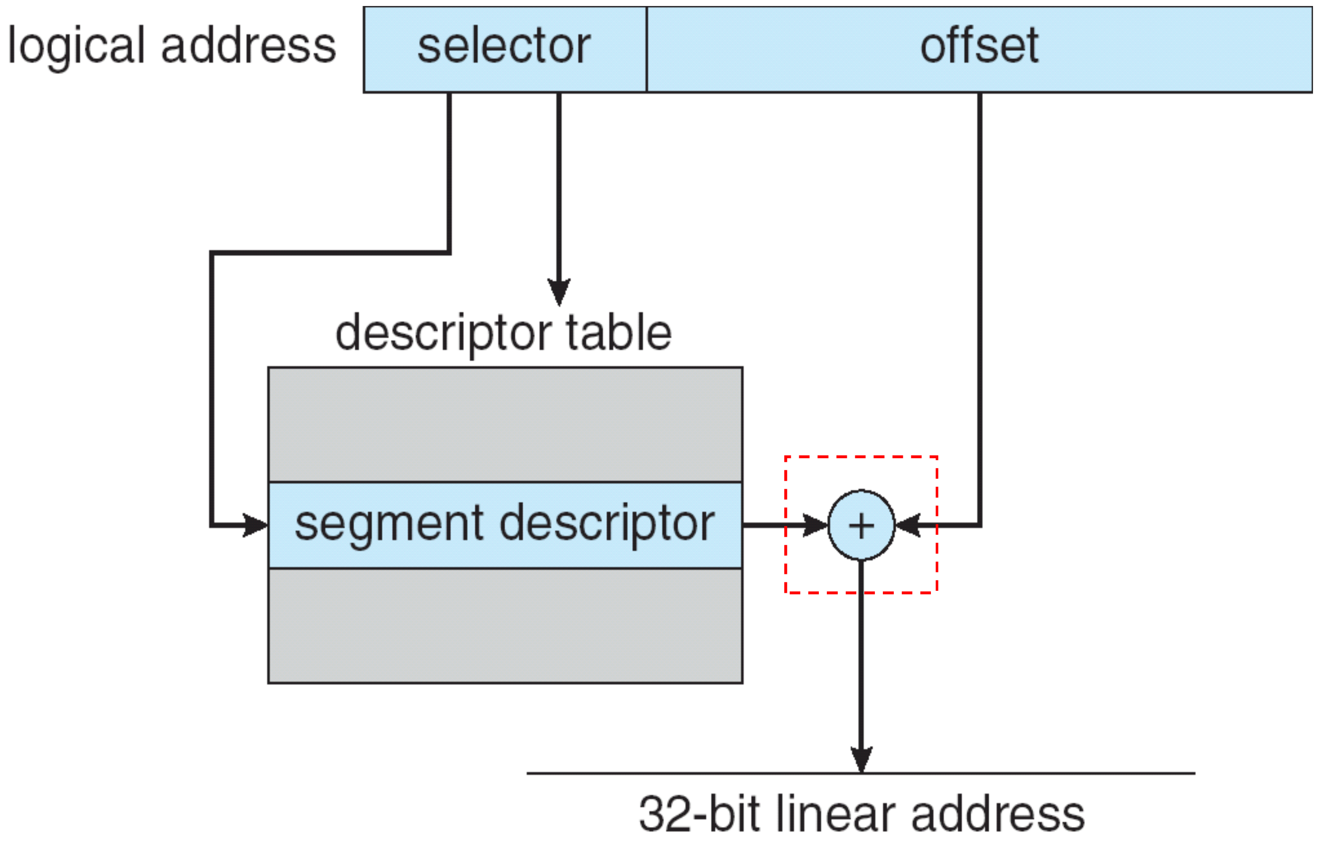
- **The Intel 32 Architecture**
  - Logical to Physical Address Translation.





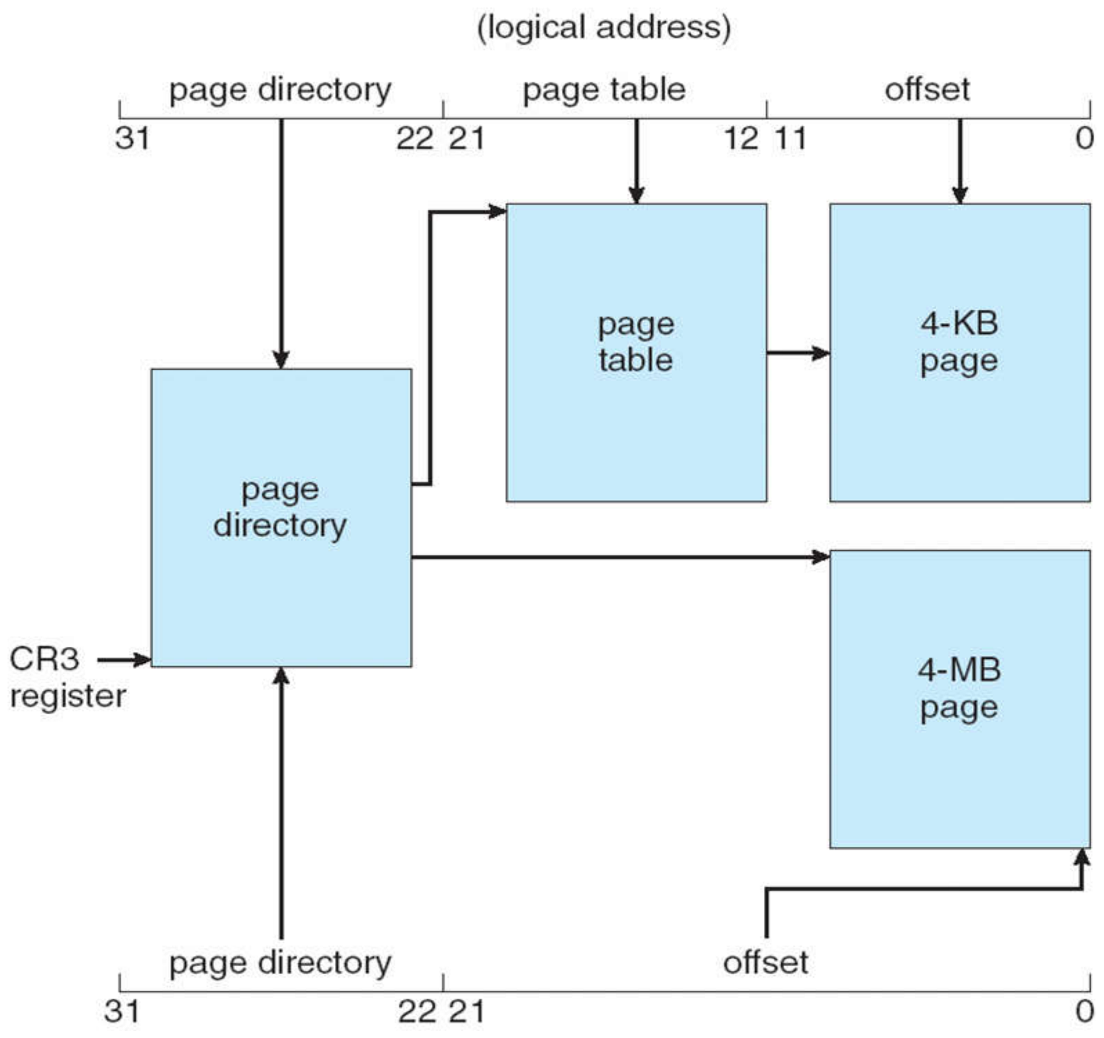
The Intel 32 Architecture

- Intel Pentium Segmentation.



Operating System Examples

- **The Intel 32 Architecture**
  - Intel Pentium Paging Architecture.

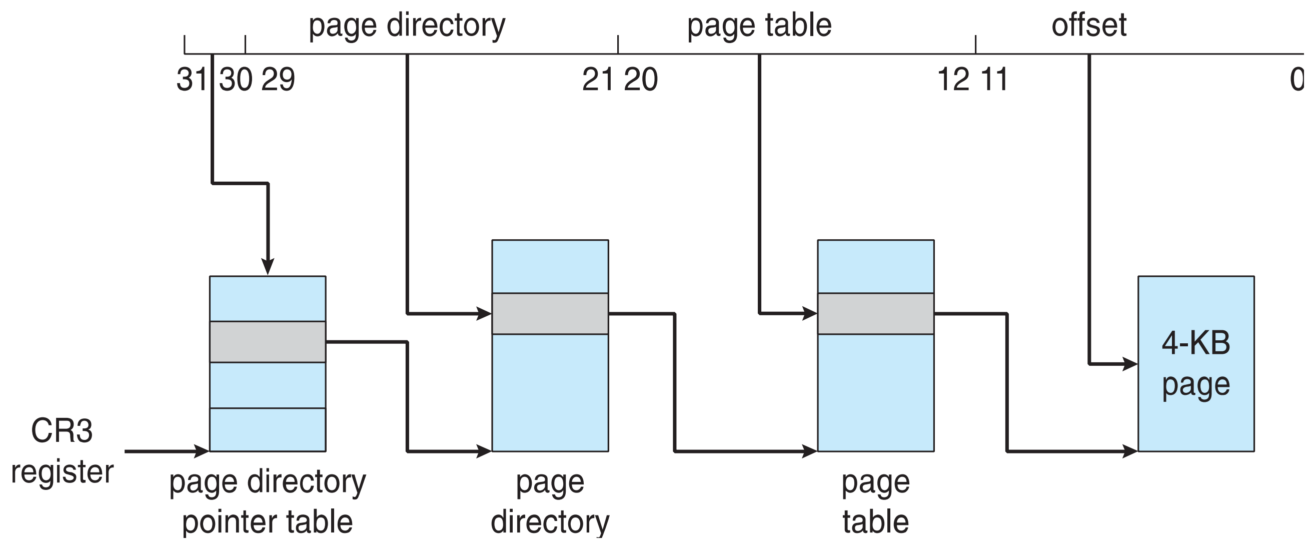


Two-level paging scheme for 4 KB page.

Direct paging scheme for 4 MB page.

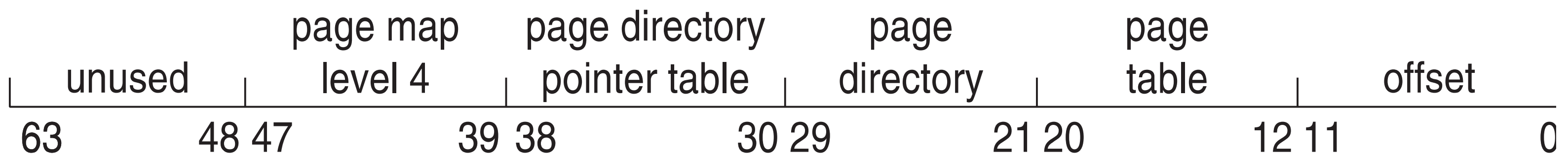
The Intel 32 Architecture

- Intel IA-32 Page Address Extensions
  - 32-bit address limits led Intel to create *page address extension* (PAE), allowing 32-bit apps access to more than 4GB of memory space.
    - Paging went to a three-level scheme
    - Top two bits refer to a *page directory pointer table*
    - Page-directory and page-table entries moved to 64-bits in size
    - Net effect is increasing address space to 36 bits – 64GB of physical memory



Operating System Examples

- **The Intel 32 Architecture**
  - Intel X86-64
    - current generation Intel x86 architecture
    - 64 bits is ginormous (> 16 exabytes)
    - in practice only implement 48 bit addressing
      - Page sizes of 4KB / 2MB / 1GB
      - Four levels of paging hierarchy
    - can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits

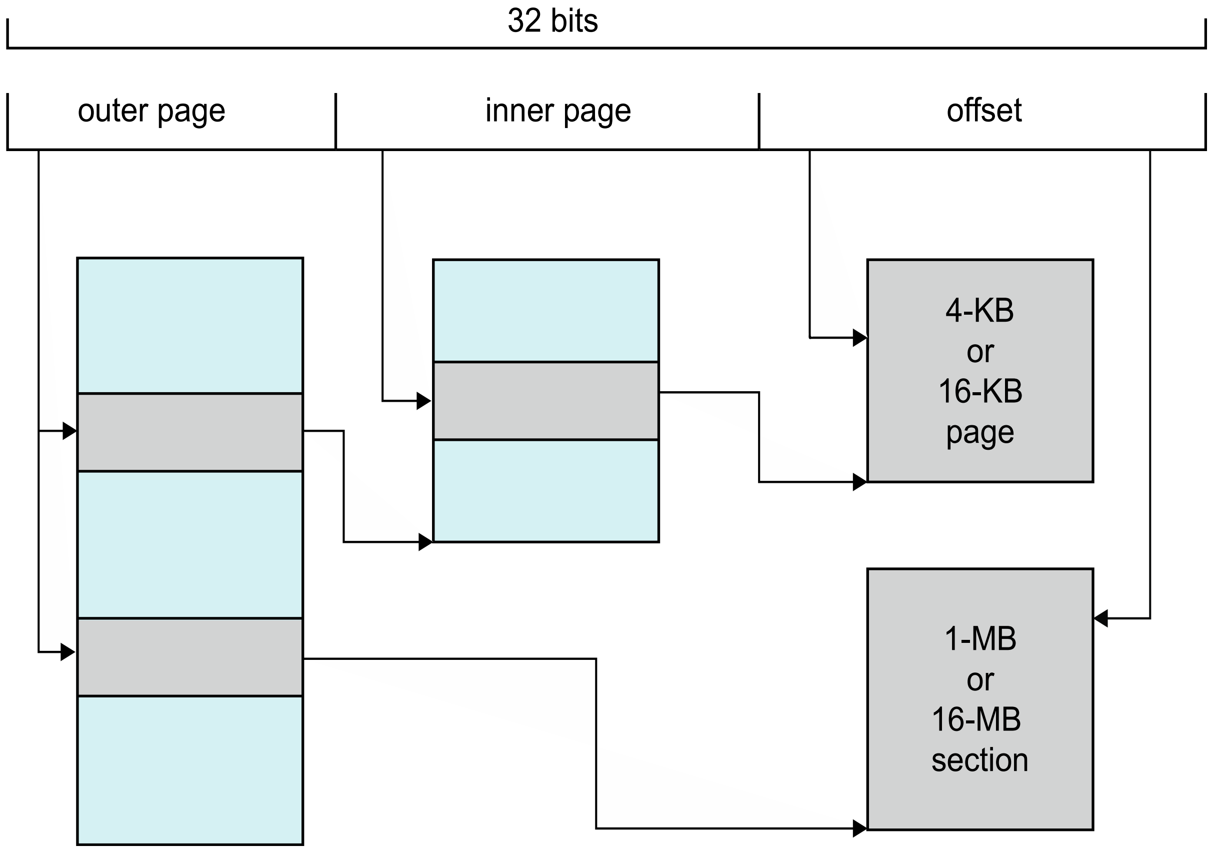


Operating System Examples

- **ARM Architecture**
  - Dominant mobile platform chip (Apple iOS and Google Android devices for example)
  - Modern, energy efficient, 32-bit CPU
  - 4KB / 16KB pages
  - 1MB / 16MB pages (termed *sections*)
  - One-level paging for sections, two-level for smaller pages
  - Two levels of TLBs
    - Outer level has two micro TLBs (one for data, one for instruction)
    - Inner is single main TLB
    - First inner is checked, on miss outers are checked, and on miss page table walk performed by CPU

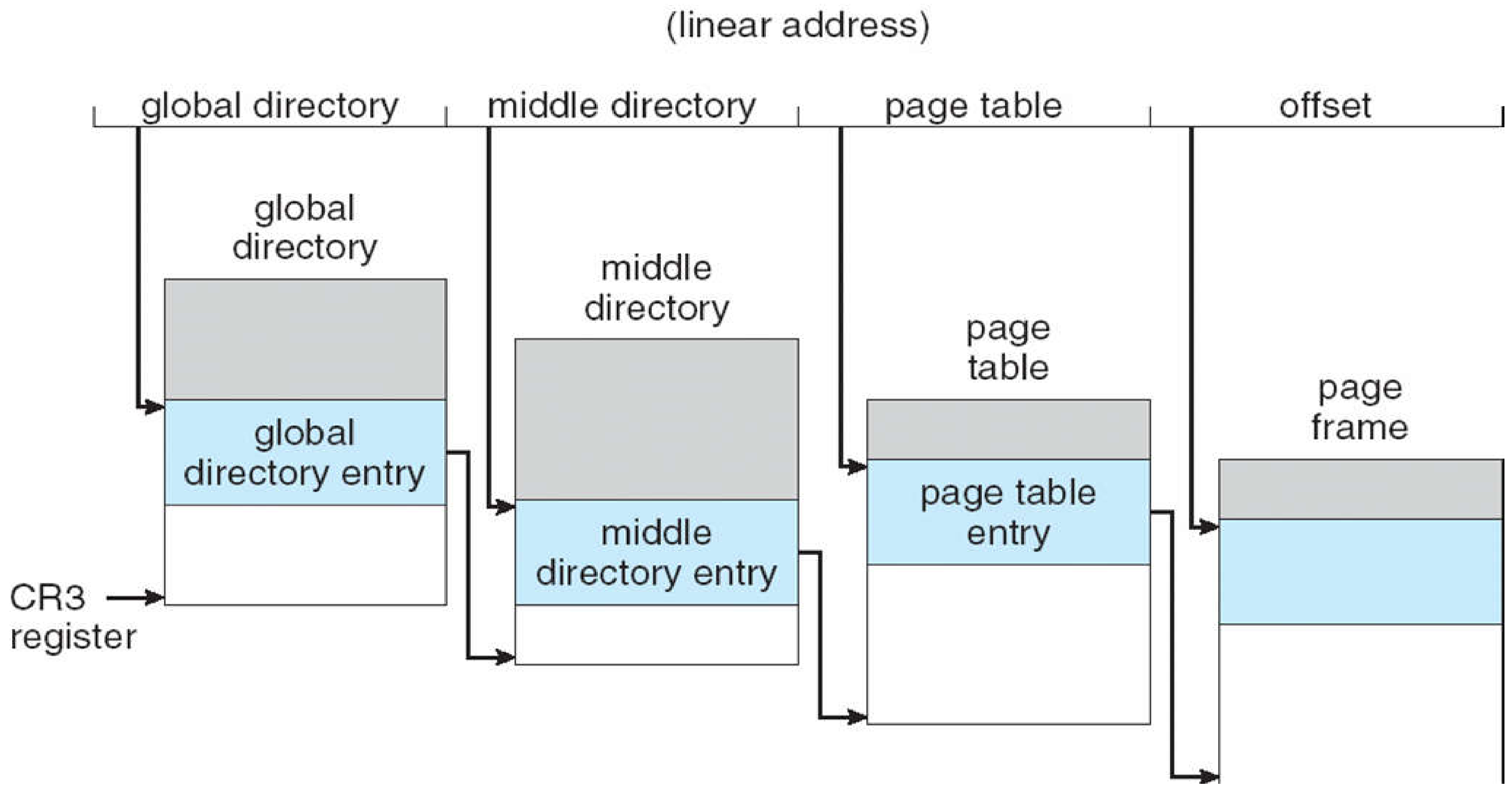
Operating System Examples

ARM Architecture



Operating System Examples

- **Linux**
  - Three-level Paging in Linux.


(linear address)
global directory
middle directory
page table
offset
global directory
global directory entry
middle directory
middle directory entry
page table
page table entry
page frame
CR3 register