Memory Management

Bojan Nokovic

Based on: "Operating Systems Concepts", 10th Edition Silberschatz Et al.

Feb. 2021

Background

Role in programming language

- instructions - read-only
- variables - read and write
- constants - read-only

Memory unit only sees a stream of:

- addresses + read requests
- address + data and write requests

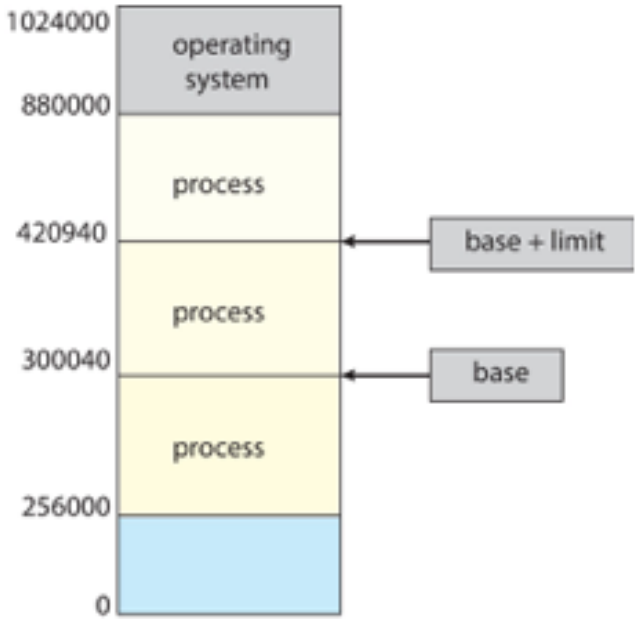
Main memory and registers are only storage CPU can access directly.

Protection of memory required to ensure correct operation

Protection

OS has to be protected from access by user processes, as well as protect user processes from one another.

We can provide this protection by base and limit registers.

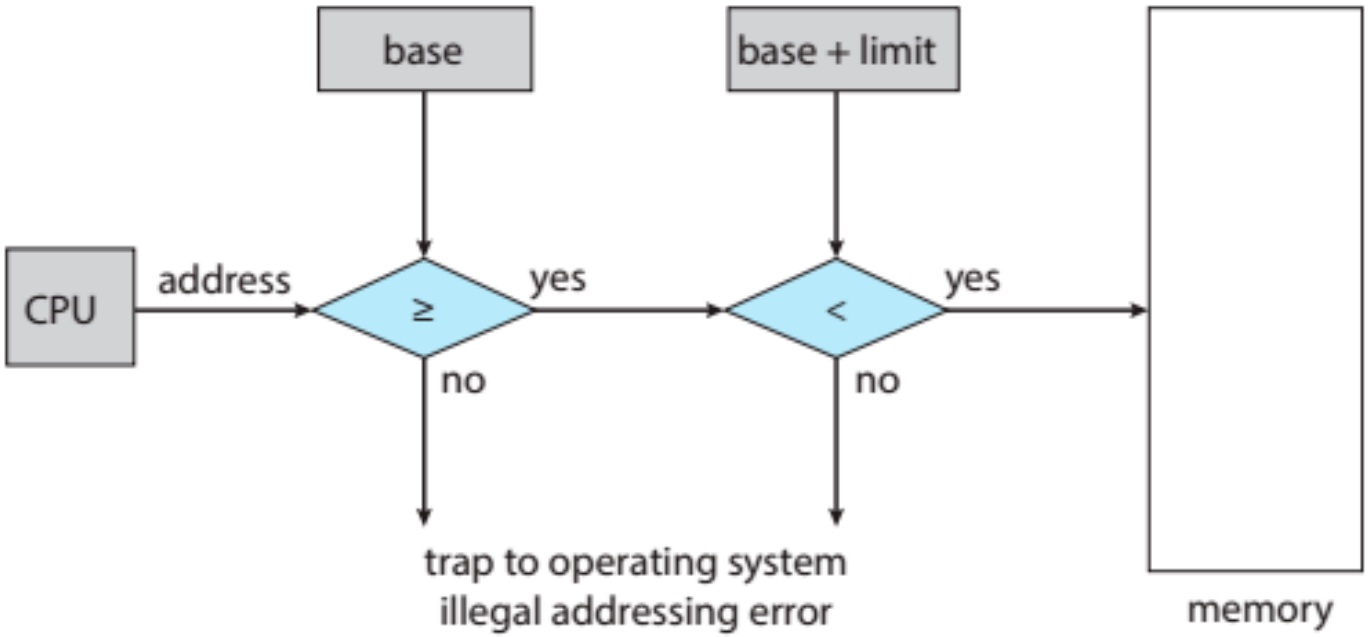


A base and a limit register define a logical address space.

Hardware Address Protection

CPU must check every memory access generated in user mode to be sure it is between base and limit for that user.

Protection of memory space is accomplished by having the CPU hardware.



The instructions to loading the base and limit registers are privileged (can be used only by the OS).

Address Binding

Program resides on a disk as a binary executable file.

Must be loaded into memory address 0000 (inconvenient)

Addresses represented in different ways at different stages of a program's life

- Addresses in the source program are generally symbolic (such as the variable `count`).
- 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!

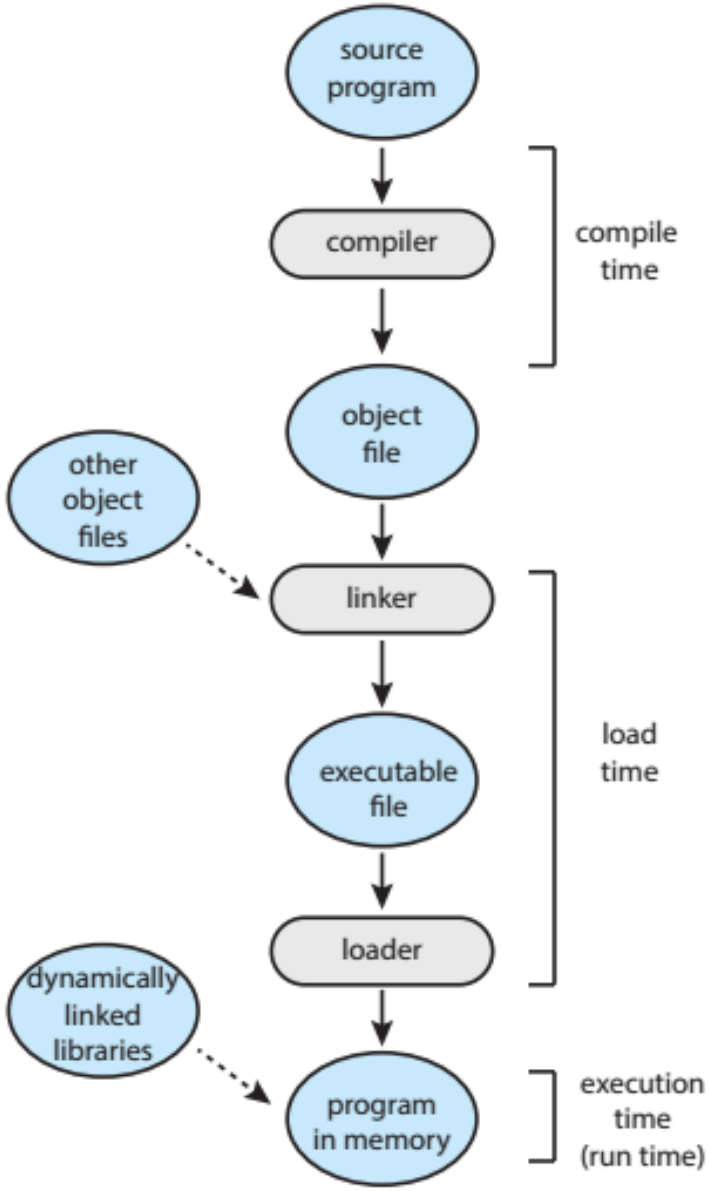
Binding of Instructions and Data to Memory

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.
- Load time: Must generate relocatable code if memory location is not known at compile time.
- Execution time: If the process can be moved during its execution from one memory segment to another, binding must be delayed until run time.
  - Need hardware support for address maps.

Multistep Processing of a User Program

- Compile time - absolute code.
- Load time - relocatable code.
- Execution time - run time binding.



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.

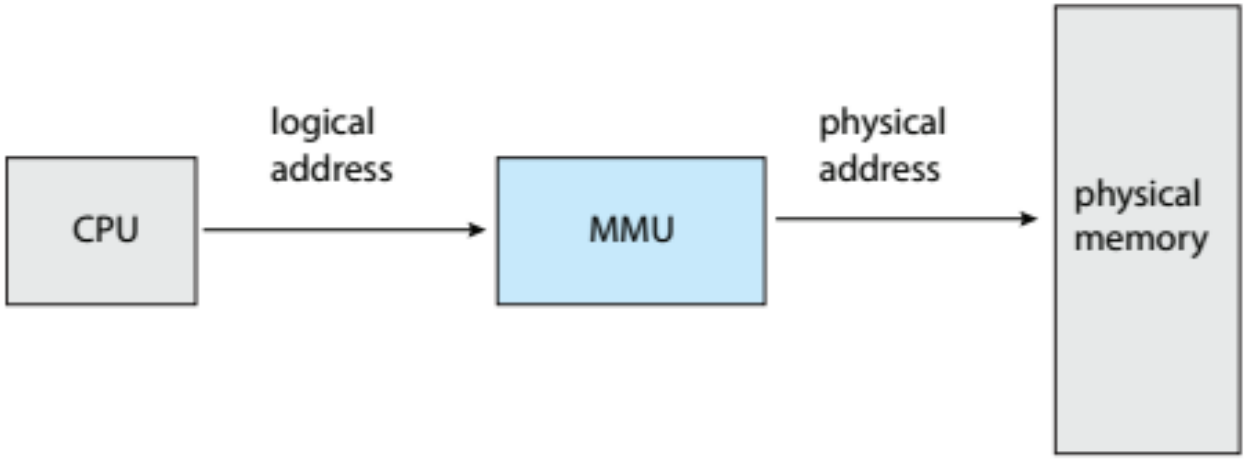
- Logical address - generated by the CPU, also referred to as virtual address.
- Physical address - address seen by the memory unit.

Logical and physical addresses are the same in compile-time and load-time address-binding schemes.

Logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

The run-time mapping from virtual to physical addresses is done by a hardware device memory-management unit (MMU)



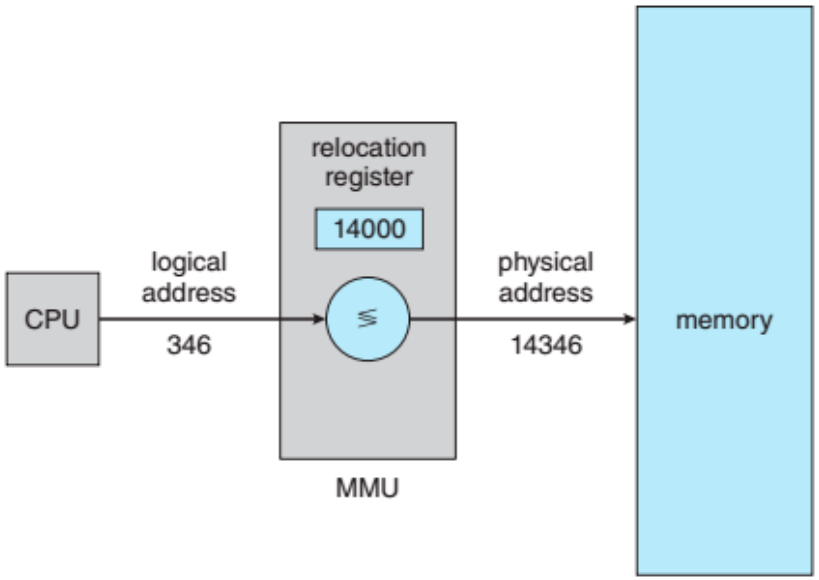
We can choose from many different methods to accomplish run-time mapping.

Memory-Management Unit (Cont.)

A generalization of the base-register scheme.

The base register now called relocation register

The value in the relocation register is added to every address generated by a user process at the time it is sent to memory



If the base is at 14000, then an attempt by the user to address location 346 is dynamically relocated to location 14346.

Dynamic Loading

Initially we should have entire program and all data of a process to be in physical memory for the process to execute - size of a process has been limited to the size of physical memory.

For 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.

No special support from the OS is required

- Implemented through program design
- OS can help by providing libraries to implement dynamic loading

Dynamic Linking

Dynamically linked libraries (DLLs) are system libraries that are linked to user programs when the programs are run.

Static linking - system libraries and program code combined by the loader into the binary program image.

Dynamic linking - linking postponed until execution time.

Without this facility, each program on a system must include a copy of its language library (or at least the routines referenced by the program) in the executable image.

- Decrease size of executable image.
- Libraries can be shared by multiple processes, only one instance of DLL in main memory.

Contiguous Memory Allocation

Main memory must support both OS and user processes - we need to allocate main memory in the most efficient way possible.

Contiguous allocation is one early method.

Main memory usually divided into two partitions:

- Resident OS usually held in low memory with interrupt vector
- User processes then held in high memory

Each process contained in single contiguous section of memory

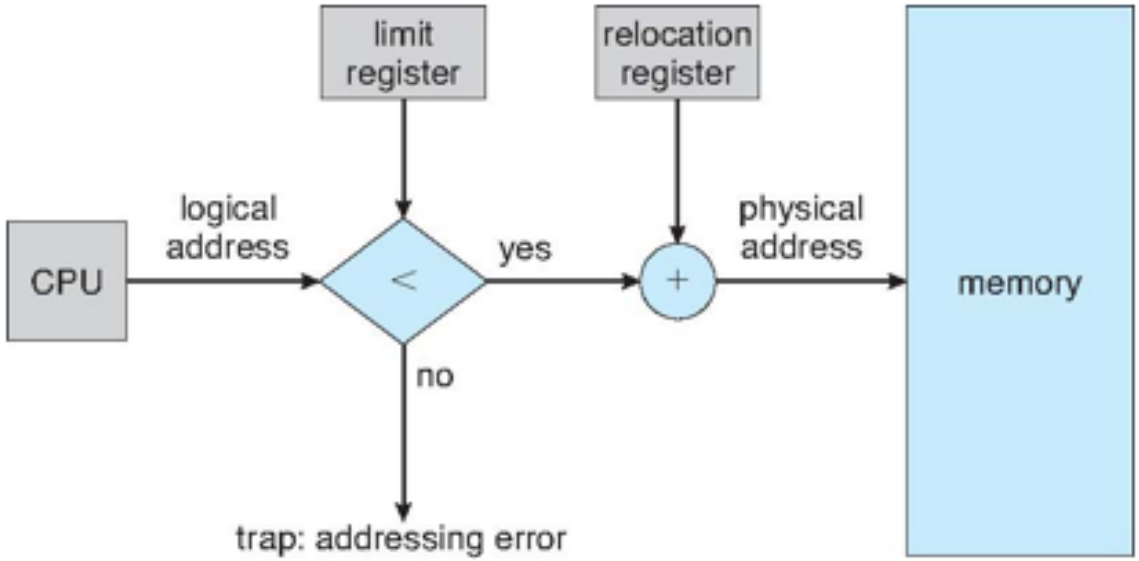
Contiguous Allocation

Relocation registers used to protect user processes from each other, and from changing operating-system code and data

- 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
- Can then allow actions such as kernel code being transient and kernel changing size

Hardware Support for Relocation and Limit Registers

Memory protection



The relocation register contains the value of the smallest physical address (i.e. relocation = 100040)
The limit register contains the range of logical addresses (i.e. limit = 74960)

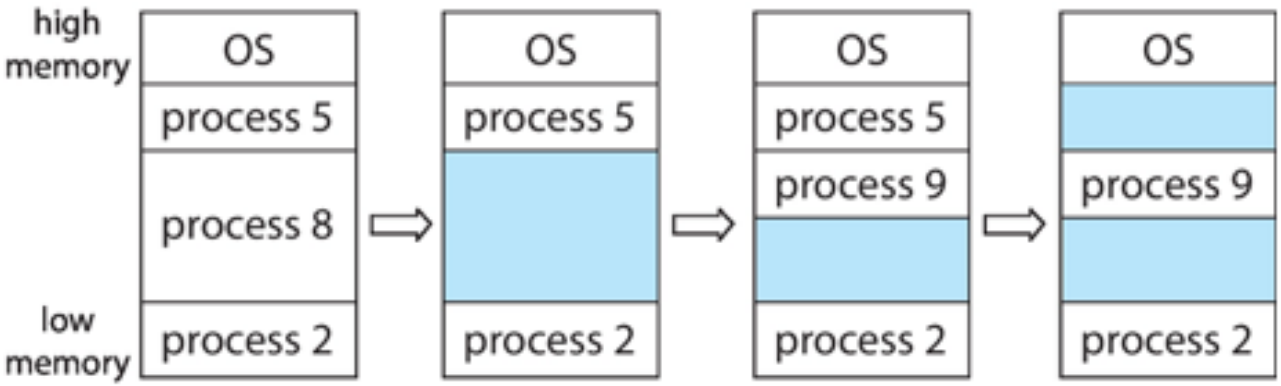
Variable Partition

The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically.

Operating system maintains information about:

a) allocated partitions

b) free partitions (hole)



Allocation

What happens when there isn't sufficient memory to satisfy the demands of an arriving process?

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.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.

External Fragmentation - total memory space exists to satisfy a request, but it is not contiguous.

Internal Fragmentation - allocated memory may be slightly larger than requested memory.

First-fit analysis reveals that given N blocks allocated, another $0.5N$ blocks lost to fragmentation

$\frac{1}{3}$ may be unusable $\rightarrow$ 50% 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

The simplest compaction algorithm is to move all processes toward one end of memory.

Another possible solution is to permit the logical address space of processes to be noncontiguous thus allowing a process to allocate physical memory wherever such memory is available - **paging**.

Paging

Physical address space of a process can be noncontiguous.

- 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.

To run a program of size N pages, need to find N free frames.

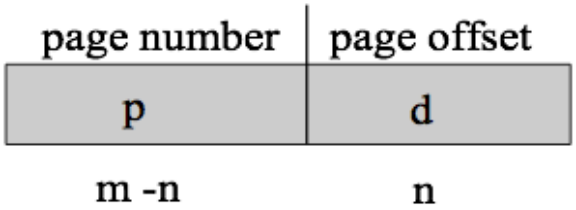
Set up a page table to translate logical to physical addresses.

Backing store likewise split into pages.

Address Translation Scheme

Address generated by CPU is divided into two parts:

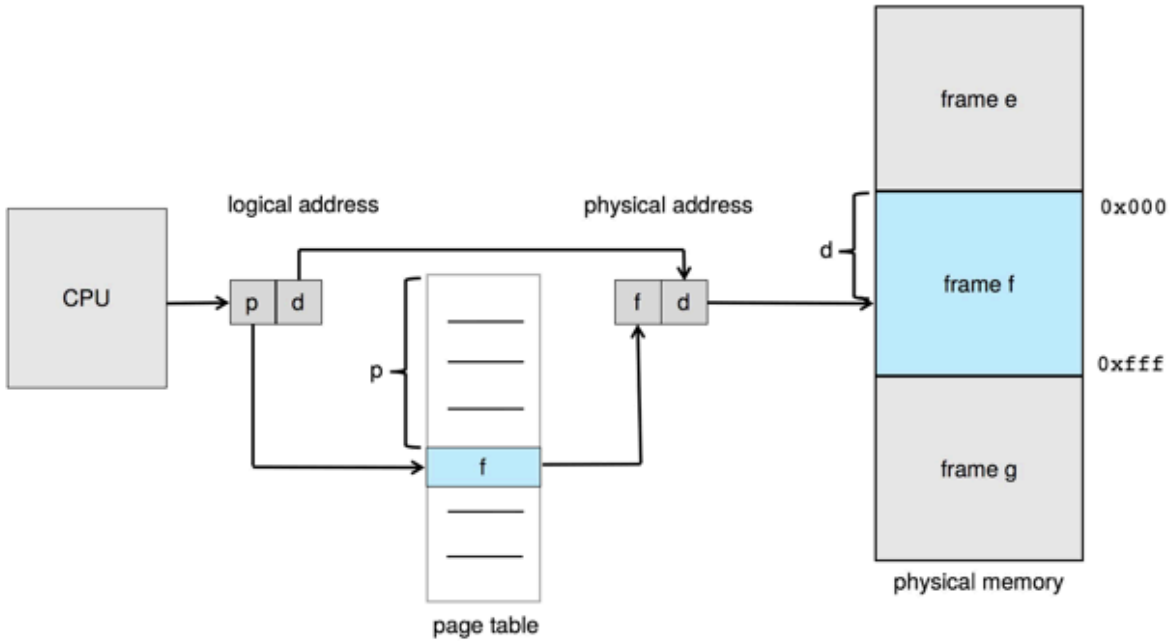
- Page number (p) - used as an index into a page table which contains base address of each page in physical memory
- Page offset (d) - combined with base address to define the physical memory address that is sent to the memory unit



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

Logical address space is now totally separated from the physical address space.

Paging Hardware

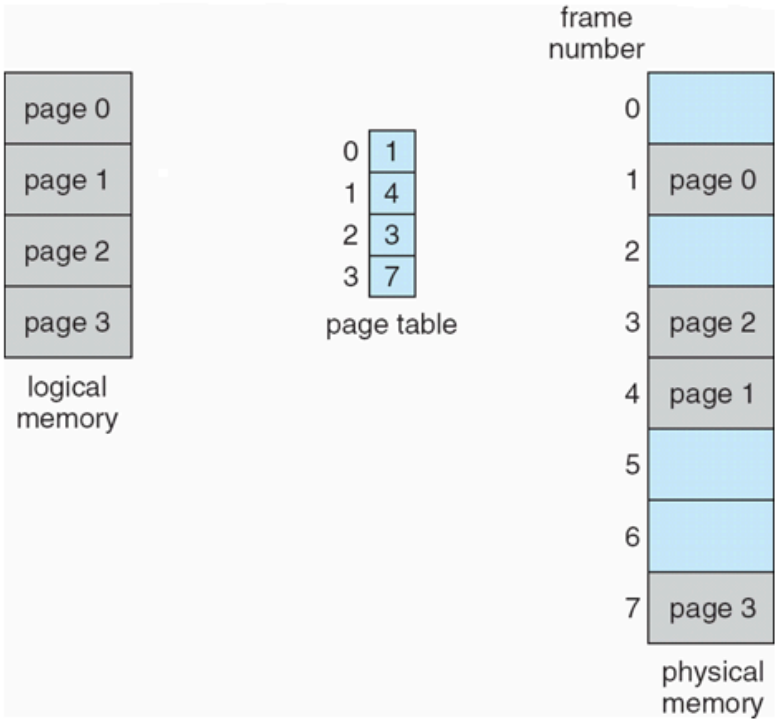


The page number is used as an index into a per-process page table.

The page table contains the base address of each frame in physical memory.

Offset *d* does not change.

Paging Model of Logical and Physical Memory



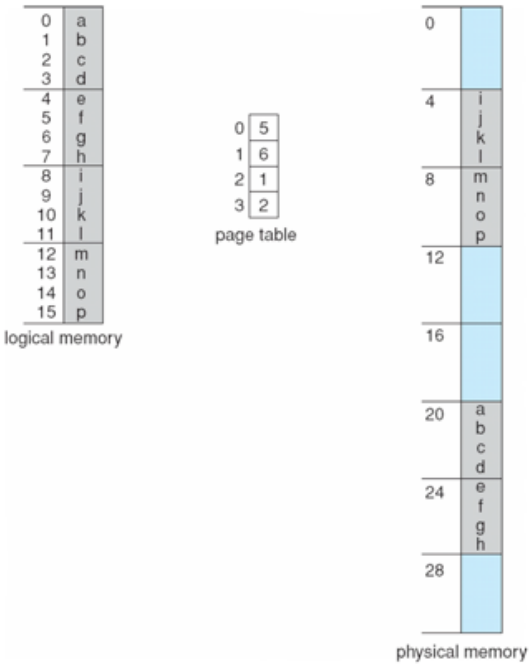
Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 1.

Paging is a form of dynamic relocation.

Paging Example

Logical address: $n = 2$ and $m = 4$.

Page size of 4 bytes, physical memory of 32 bytes (8 pages).



Logical address 0 maps to physical address $20 [= (5 \times 4) + 0]$.
3 (page 0, offset 3) $\rightarrow 23 [= (5 \times 4) + 3]$.
4 (page 1, offset 0) $\rightarrow 24 [= (6 \times 4) + 0]$.

Paging - Calculating Internal Fragmentation

Internal Fragmentation Example

Given page size = 2048 bytes.
For process size = 72766 bytes we need 36 pages.
Internal fragmentation = $36 \times 2048 - 72766 = 962$ bytes

Worst case fragmentation = frame - 1 byte

Average fragmentation = 1/2 frame size

Pages are typically either 4 KB or 8 KB in size.

Some CPUs and operating systems even support multiple page sizes (Windows 10 4KB and 2MB, Linux 4KB and *huge pages*)

Example

Frequently, on a 32-bit CPU, each page-table entry is 4 bytes long.

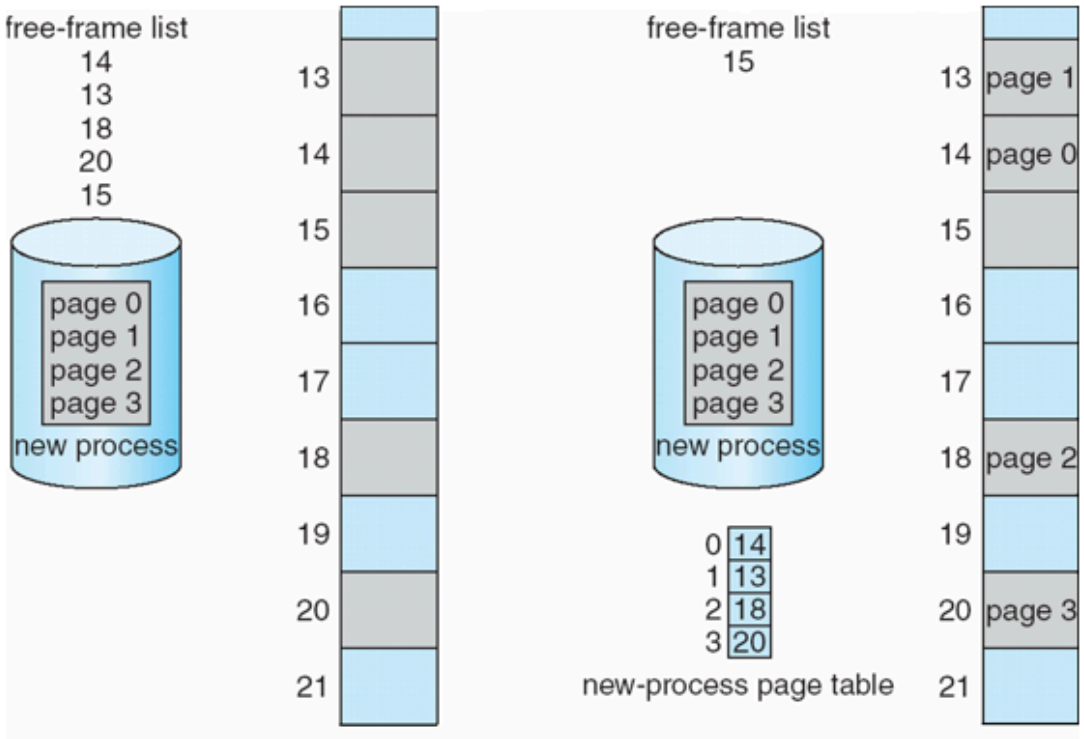
A 32-bit entry can point to one of 2^{32} physical page frames.

If the frame size is 4 KB (2^{12}), then a system can address 16 TB (2^{44}) bytes of physical memory!

Free Frames

When a process arrives in the system to be executed, its size, expressed in pages, is examined.

Each page of the process needs one frame.



Free frames before and after 4 pages allocation.

Implementation of Page Table

The hardware implementation of the page table can be done in several ways.

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 access requires two steps one for the page table and one for the actual data.

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

Translation Look-Aside Buffer

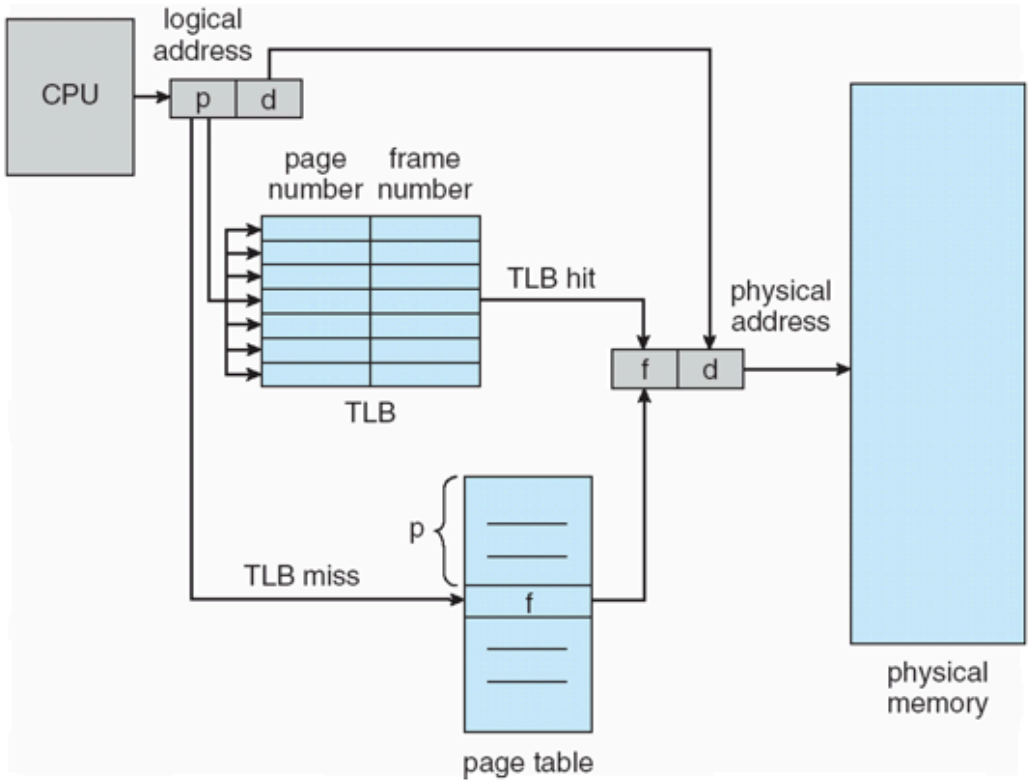
Some TLBs store address-space identifiers (ASIDs) in each TLB entry - uniquely identifies each process used to provide address-space protection for that process.

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 (if full)
- Some entries can be wired down for permanent fast access

Paging Hardware With TLB



When the frame number is obtained, we can use it to access memory.

We add the page number and frame number to the TLB, so that they will be found quickly on the next reference.

Effective Access Time

Percentage of times that a page number is found in the TLB is called hit ratio.

An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.

Suppose that takes 10 ns to access memory if we find the desired page in TLB, otherwise we need two memory access so it is 20 ns

Effective Access Time (EAT) = 0.80 x 10 + 0.20 x 20 = 12ns
20% slowdown in access time

Question

Calculate slowdown in access time for hit ratio of 99%.

Memory Protection

Memory protection in a paged environment is accomplished by protection bits associated with each frame.

These bits are kept in the page table.

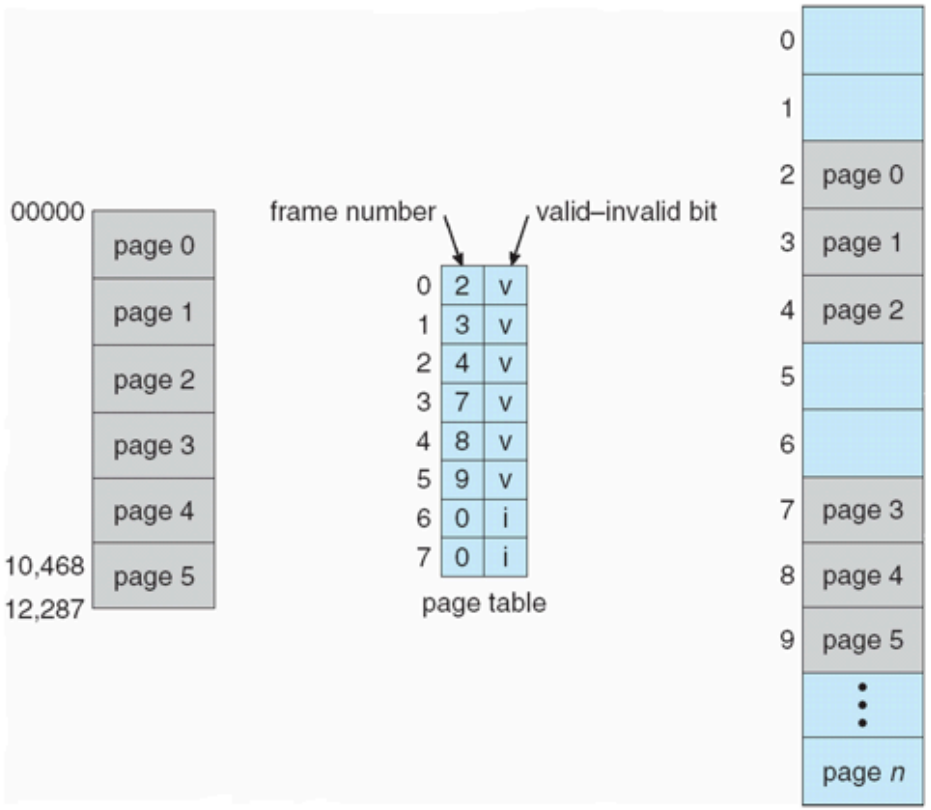
One bit can define a page to be read-write or read-only.

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
- Or use page-table length register (PTLR)

Any violations result in a trap to the kernel!

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



Any attempt to generate an address in pages 6 or 7, however, will find that the valid-invalid bit is set to invalid, and the computer will trap to the OS.

Shared Pages

Shared code

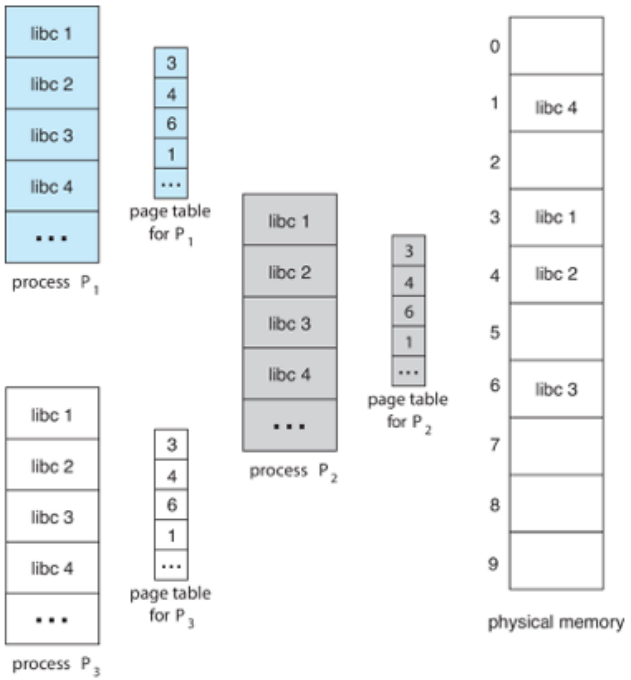
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
- Similar to multiple threads sharing the same process space
- Also useful for interprocess communication if sharing of read-write pages is allowed

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



Three processes sharing the pages for the standard C library `libc`.

Only one copy of the standard C library need be kept in physical memory.

The page table for each user process maps onto the same physical copy of `libc`.

Structure of the Page Table

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

- Consider a 32-bit logical address space as on modern computers
- Page size of 4 KB (2^{12})
- Page table would have 1 million entries ($2^{20} = 2^{32} / 2^{12}$)
- If each entry is 4 bytes $\rightarrow$ each process 4 MB of physical address space for the page table alone

We don't want to allocate the page table contiguously in main memory.

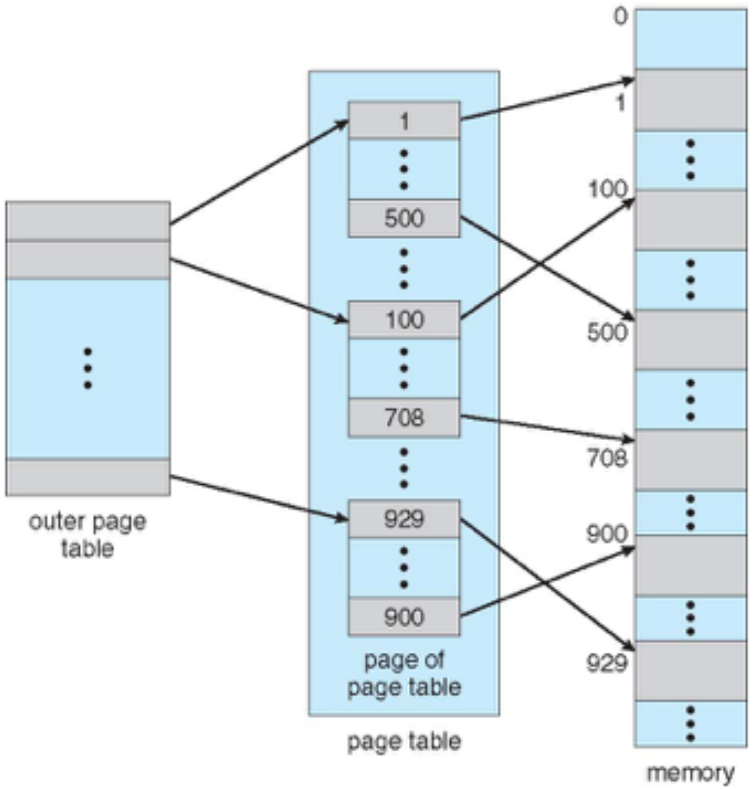
One simple solution is to divide the page table into smaller units - Hierarchical Paging, Hashed Page Tables, Inverted Page Tables.

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



Two-Level Paging Example

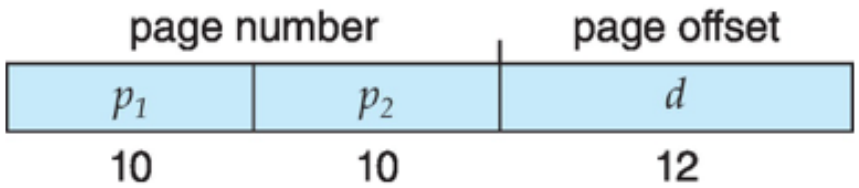
A logical address (on 32-bit machine with 4K page size) is divided into:

- 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:

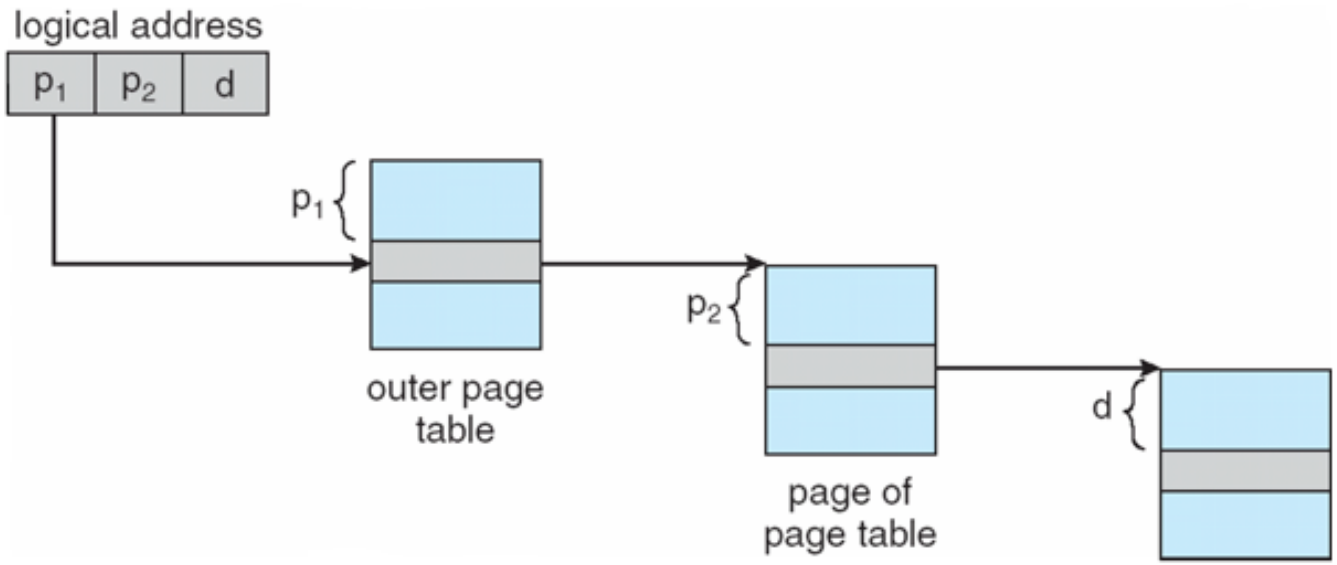
- a 10-bit page number
- a 12-bit page offset

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-Translation Scheme



Address translation for a two-level 32-bit paging architecture.

64-bit Logical Address Space

Even two-level paging scheme not sufficient!

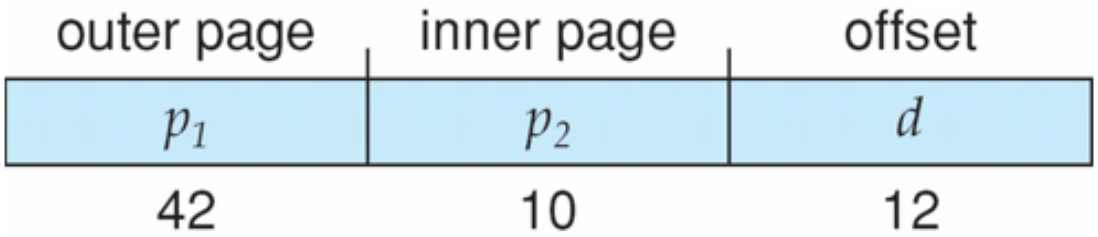
If page size is 4 KB (2^{12}) then page table has 2^{52} entries.

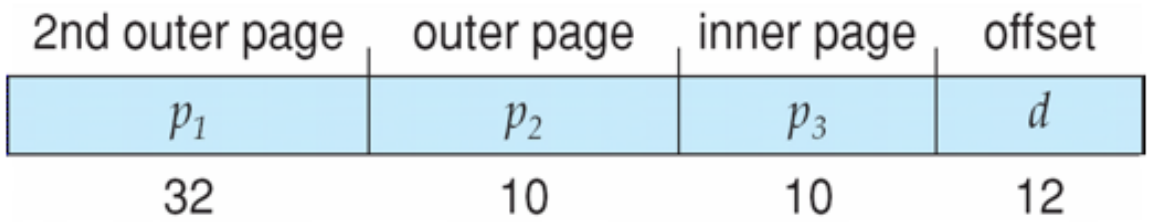
If two level scheme, inner page tables could be 2^{10} 4-byte entries.

Outer page table has 2^{42} entries. One solution is to add a 2nd outer page table.

Three-level Paging Scheme

In the following example the 2nd outer page table is still 2^{32} bytes in size.





The next step would be a four-level paging scheme, where the second-level outer page table itself is also paged, and so forth.

Hashed Page Tables

Common in address spaces > 32 bits

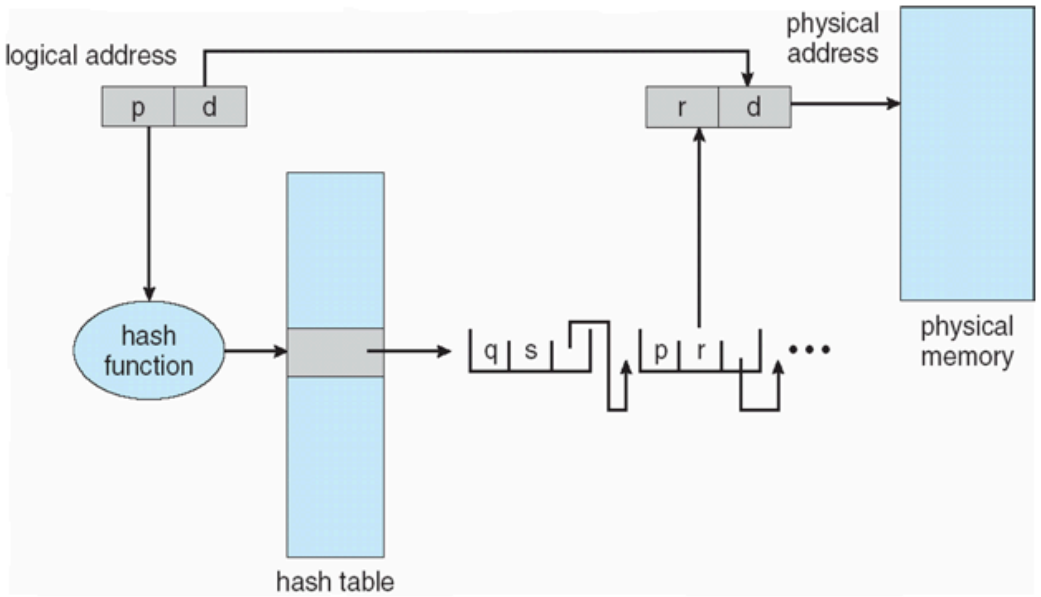
The virtual page number is hashed into a page table.

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.

Hashed Page Table



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.

Inverted Page Table

Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.

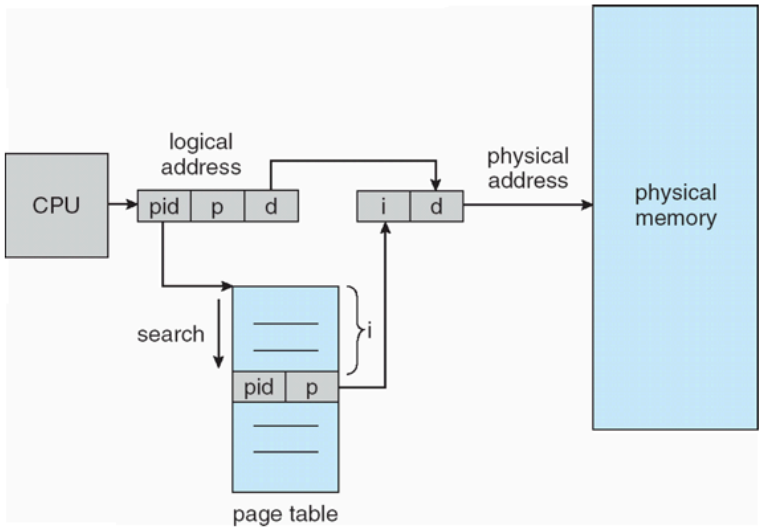
One entry for each real page of memory. An inverted page table has one entry for each real page (or frame) of memory.

Although this scheme decreases the amount of memory needed to store each page table, it increases the amount of 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.

Inverted Page Table Architecture



$< pid, pagenumber >$ is presented to the memory subsystem.

The inverted page table is then searched for a match. If a match is found at entry i then the physical address $< i, offset >$ is generated.

Observation

How to implement shared memory?
Map multiple virtual addresses to the same physical address?

Swapping

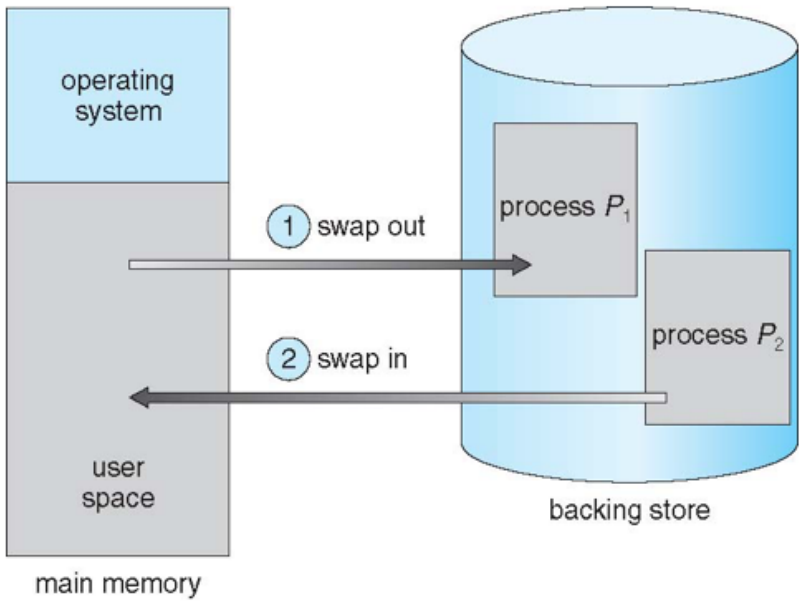
A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.

Backing store - fast disk large enough to accommodate copies of all memory images for all users.

Roll out, roll in - swapping variant used for priority-based scheduling algorithms.

Major part of swap time is transfer time. Total transfer time is directly proportional to the amount of memory swapped.

Schematic View of Swapping



Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system!

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 be very high.

Example: 100MB process swapping to hard disk with transfer rate of 50MB/sec

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

Can reduce if reduce size of memory swapped - by knowing how much memory really being used.

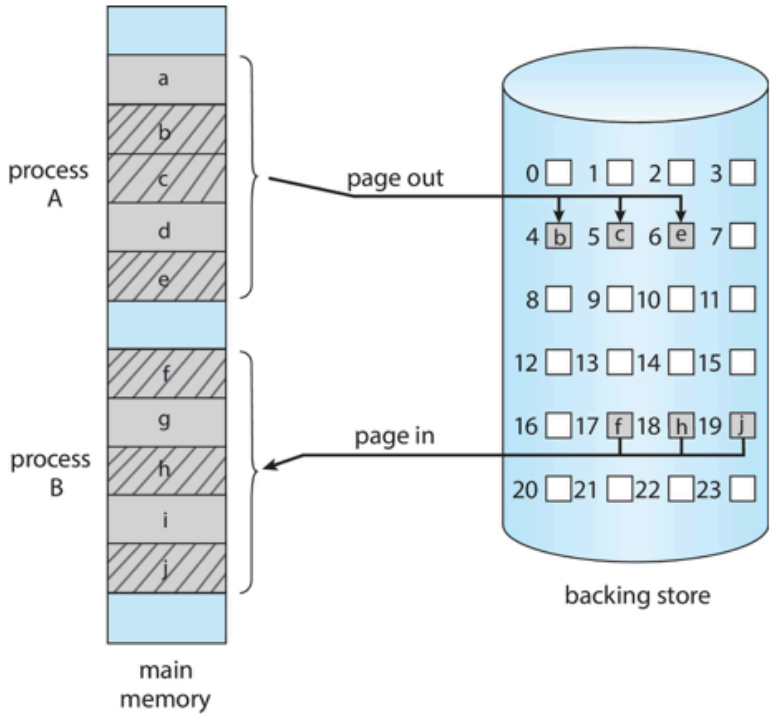
Swapping in Modern OS

Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

- Swapping normally disabled
- Started if more than threshold amount of memory allocated
- Disabled again once memory demand reduced below threshold

Schematic View of Swapping with Paging

A subset of pages for processes A and B are being paged-out and paged-in respectively.



Pages of a process rather than an entire process can be swapped.

Most systems, including Linux and Windows, now use swapping with paging.

Swapping on Mobile Systems

Not typically supported

Flash memory based: small amount of space, limited number of write cycles, poor throughput between flash memory and CPU on mobile platform

iOS asks apps to voluntarily relinquish allocated memory:

- Read-only data thrown out and reloaded from flash
- Failure to free can result in termination

Android terminates apps if low free memory, but first writes application state to flash for fast restart.

Both iOS and Android support paging

Implementation

The Intel 32 and 64-bit Architectures

Intel x86-64

ARM Architecture

Example: The Intel 32 and 64-bit Architectures

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

Example: The Intel IA-32 Architecture (Cont.)

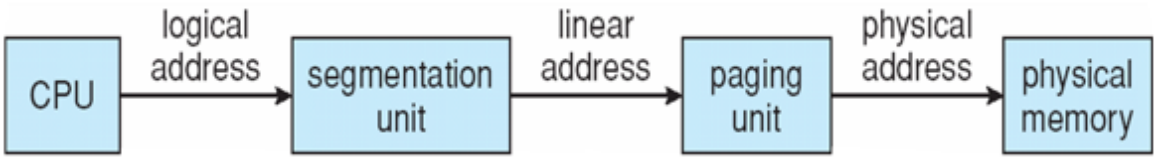
CPU generates logical address

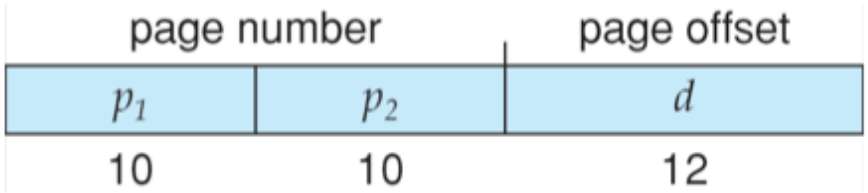
Selector given to segmentation unit which produces linear addresses

Linear address given to paging unit

- Which generates physical address in main memory
- Paging units form equivalent of MMU
- Pages sizes can be 4 KB or 4 MB

Logical to Physical Address Translation in IA-32





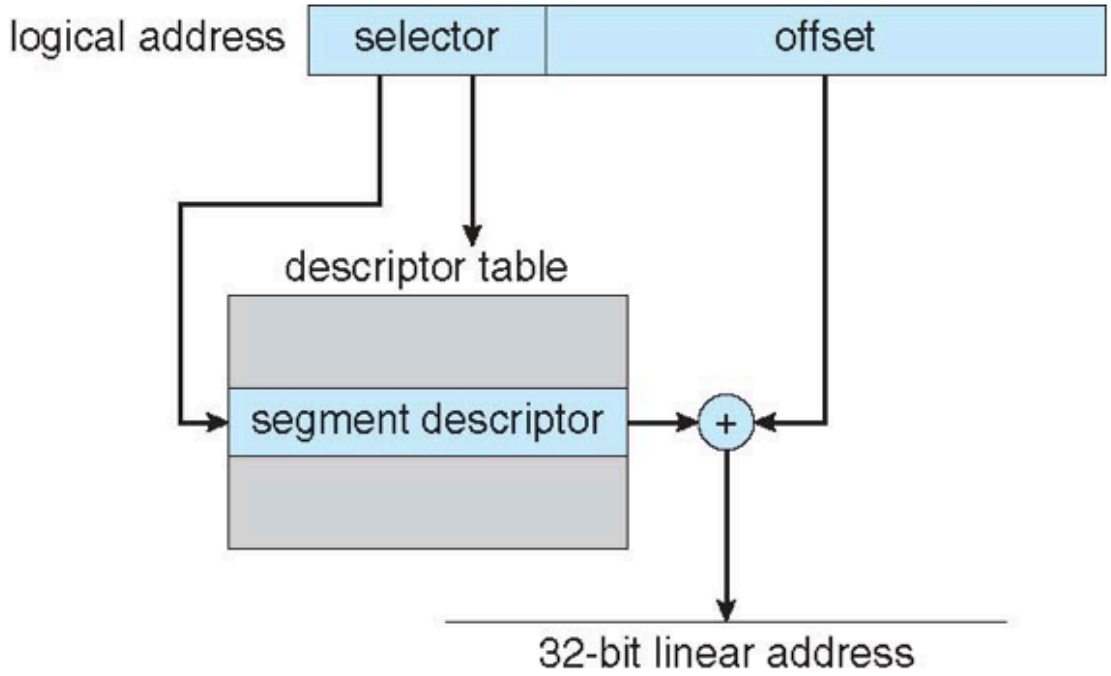
Memory management divided into two components: segmentation and paging.

The CPU generates logical addresses, which are given to the segmentation unit.

The segmentation unit produces a linear address for each logical address.

The linear address is then given to the paging unit, which in turn generates the physical address in main memory.

Intel IA-32 Segmentation



The logical address space of a process is divided into two partitions.

- up to 8 K segments that are private to that process (information kept in the (local descriptor table (LDT)).
- up to 8 K segments that are shared among all the processes (global descriptor table (GDT)).

Intel IA-32 Segmentation

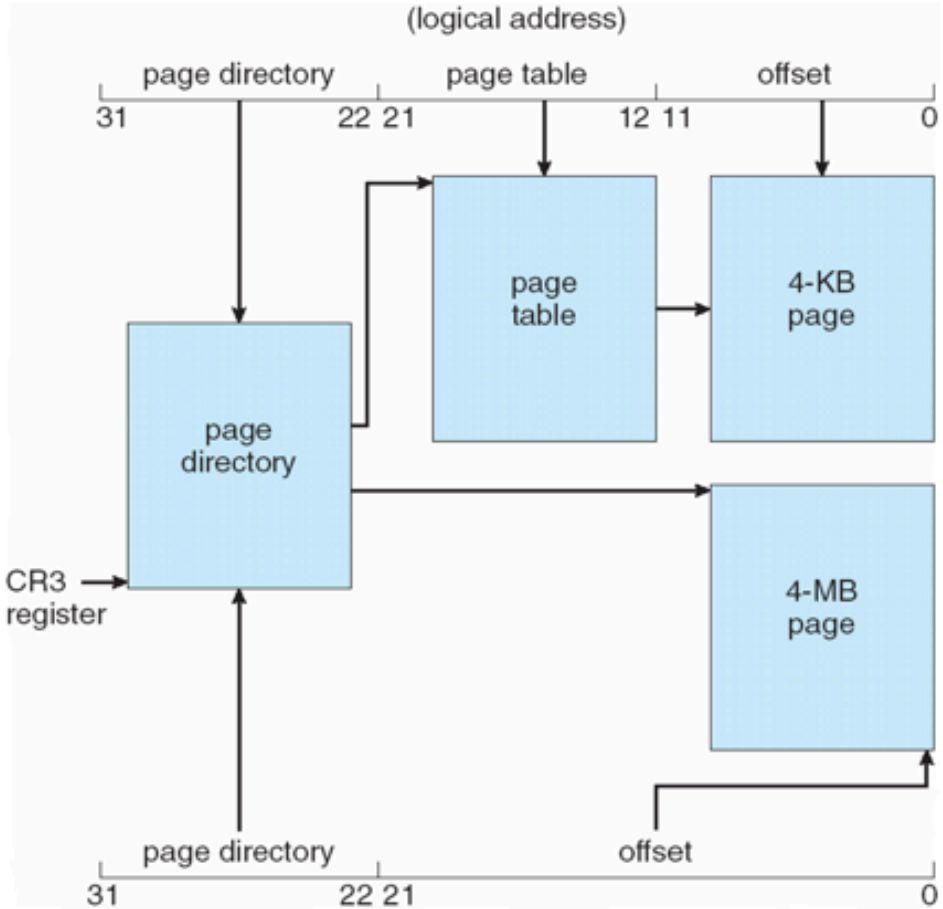
The logical address is a pair (selector, offset)

The selector is a 16-bit number:

s - the segment number
g - indicates whether the segment is in the GDT or LDT
p - deals with protection.

The offset is a 32-bit number specifying the location of the byte within the segment in question.

Intel IA-32 Paging Architecture



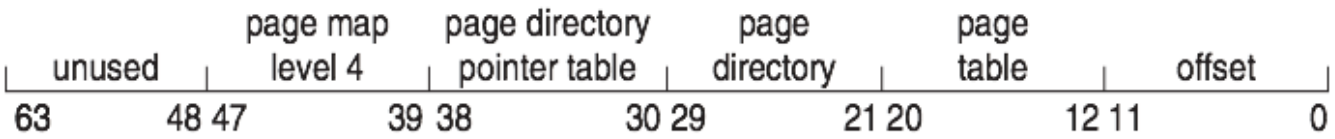
The address-translation scheme for this architecture is similar to the scheme shown in Figure on Page 39.

Intel x86-64

Current generation Intel x86 architecture

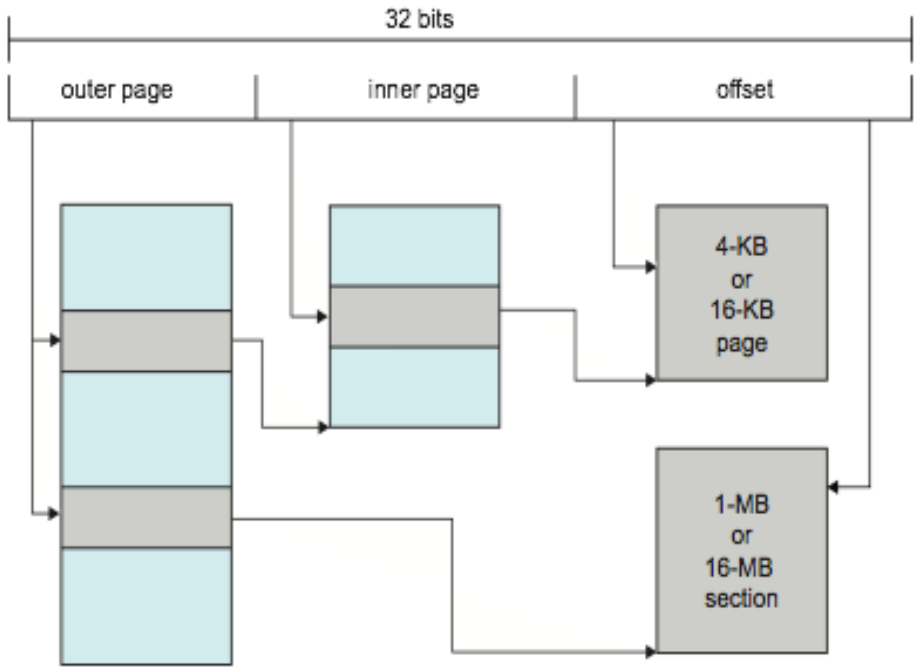
- 64 bits is ginormous (> 16 exabytes)
  1 EB = 1 000 000 000 000 000 000B
- In practice only implement 48 bit addressing - Page sizes of 4 KB, 2 MB, 1 GB; Four levels of paging hierarchy

Can also use page address extension (PAE) so virtual addresses are 48 bits and physical addresses are 52 bits



Example: ARM Architecture

- Dominant mobile platform chip (Apple iOS and Google Android devices for example)
- Modern, energy efficient, 32-bit CPU
- 4 KB and 16 KB pages
- 1 MB and 16 MB pages (termed sections)
- One-level paging for sections, two-level for smaller pages



Thank you !

Operating Systems are among the most complex pieces of software ever developed !