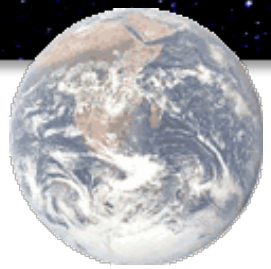
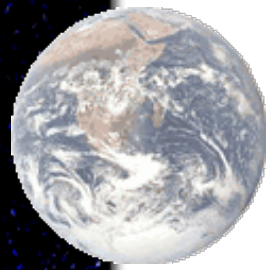
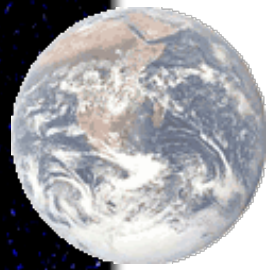


Memory Management



Background

- Program must be brought (from disk) into memory and placed within a process for it to be run.
- Memory unit only sees a stream of (addresses + read requests), or (address + data and write requests).
- Main memory and registers are only storage CPU can access directly.
- Register access in one CPU clock (or less).
- Main memory can take many cycles, causing a stall on CPU.
- Cache sits between main memory and CPU registers.
- Protection of memory required to ensure correct operation.

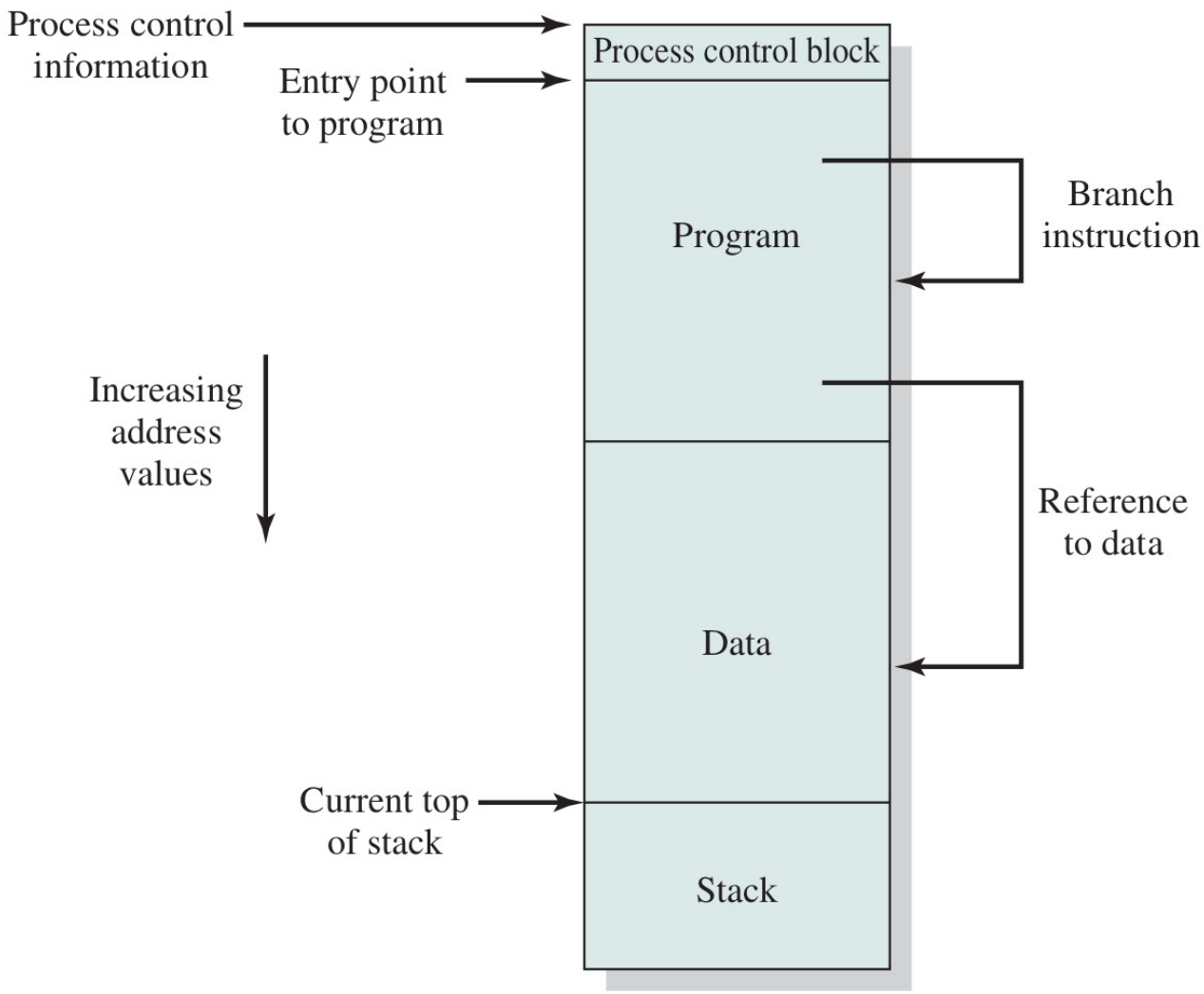


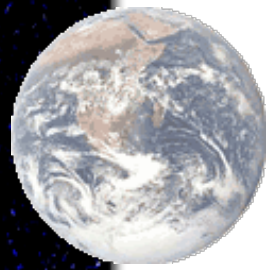
Memory Management Requirements

- Multiprogramming: subdividing memory to accommodate multiple processes.
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

- Memory management intends to satisfy following requirements:
  1. Relocation
  2. Protection
  3. Sharing
  4. Logical organization
  5. Physical organization

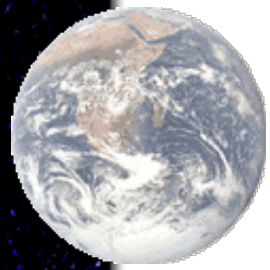
Address Requirement for a Process





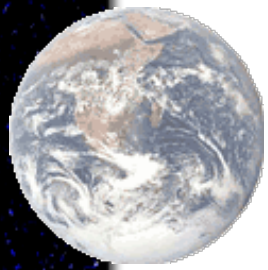
1. Relocation

- Programmer does not know where the program will be placed in memory when it is executed
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
- Memory references must be translated in the code to actual physical memory address.



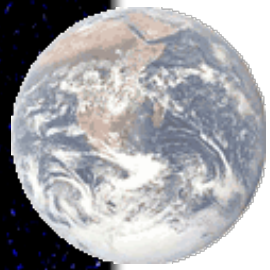
2. Protection

- Each process should be protected against unwanted interference by other process, whether accidental or intentional.
- It is only possible to assess the permissibility of a memory reference (data access or branch) at the time of execution of the instruction making the reference.
- The processor must be able to abort the violated reference at the point of execution.
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
  - Operating system cannot anticipate all of the memory references a program will make



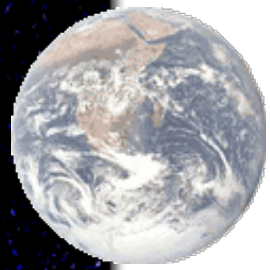
3. Sharing

- Any protection mechanism must have the flexibility to allow several processes to access the same portion of memory.
  - Number of processes execute the same program.
  - Cooperating processes share access to the same data structure.
- Better to allow each process access to the same copy of the program rather than have their own separate copy.



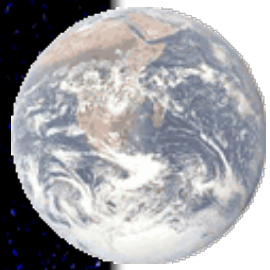
4. Local Organization

- Most programs are organized into modules which are written and compiled independently:
  - All references from one module to another resolved by the system at run time.
  - Different degrees of protection can be given to different modules
  - Provide sharing on a module level among processes



5. Physical Organization

- Computer memory is organized into at least two levels: main memory & second memory with different characteristics and purpose
- Memory available for a program plus its data may be insufficient
  - Programmer does not know how much space will be available
- The organization of the flow of information between the two levels should be a system responsibility.
  - Bring process into main memory for excution (or swap in)
  - Swap out of suspended process to second memory to release space in main memory

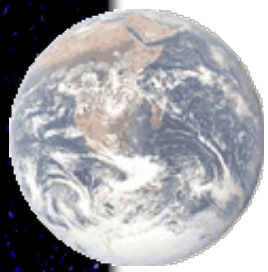


Memory Management Techniques

Main memory usually into two partitions:

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

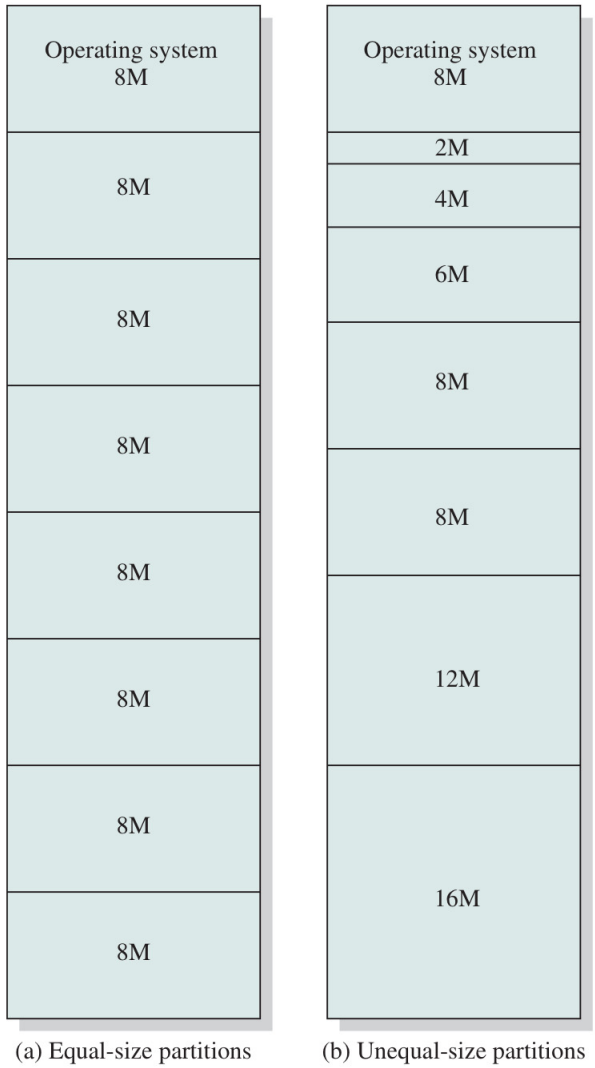
1. Contiguous memory allocation
    1. Fixed Partitioning
    2. Dynamic Partitioning
2. Non-contiguous memory allocation
    1. Simple Paging
    2. Simple Segmentation
    3. Virtual Memory Paging
    4. Virtual Memory Segmentation



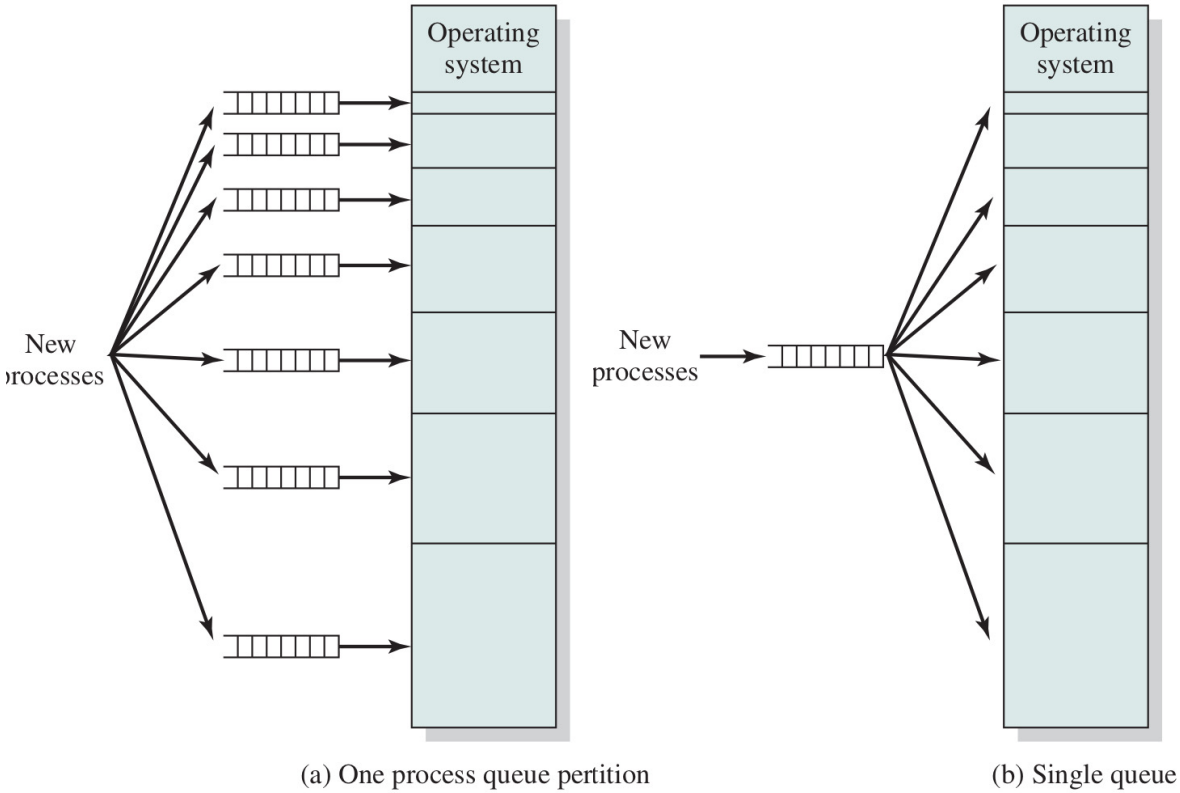
1.1. Fixed Partitioning

- Main memory is divided into a number of static partitions.
- A process may be loaded into a partition of eqaul or greater size.
- Strength:
  - Simple to implement
  - Little system overhead
- Weakness:
  - Inefficient use of memory due to internal fragmentation.
  - Maximum number of active processes is fixed.

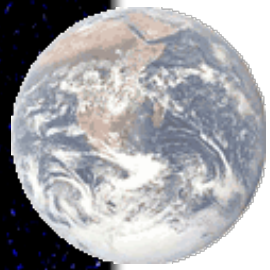
Equal-Size vs. Unequal-Size Partitioning



(a) Equal-size partitions (b) Unequal-size partitions

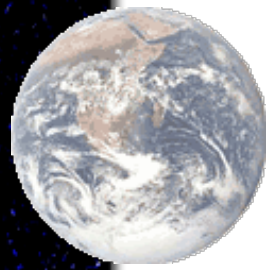


(a) One process queue pertition (b) Single queue



1.2. Dynamic Partitioning

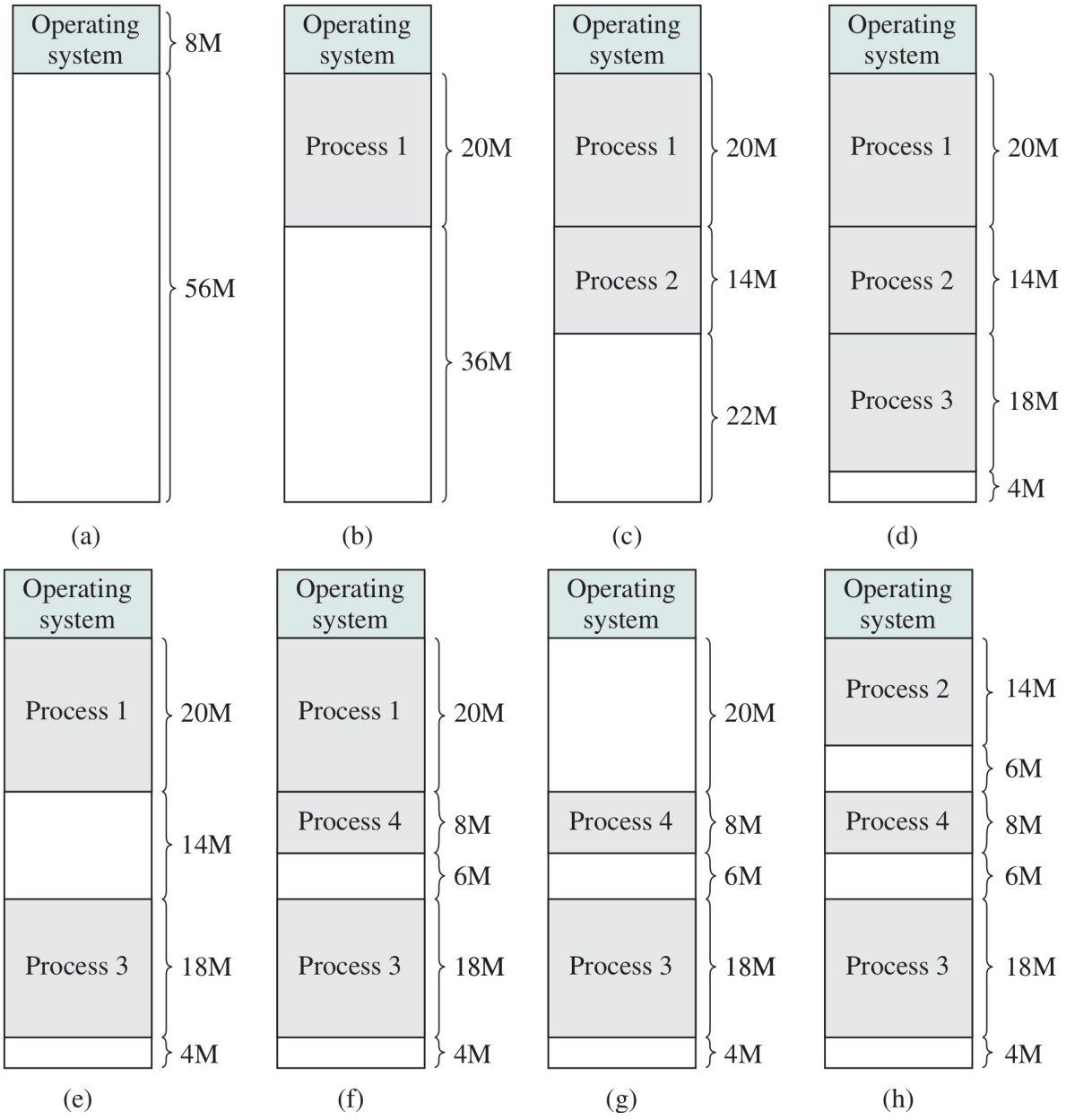
- Partitions are created dynamically.
- Each process is loaded into a partition of exactly the same size as that process
- Strength:
  - No internal fragmentation
  - More efficient use of main memory
- Weakness:
  - Inefficient use of processor due to the need for compaction to counter external fragmentation.

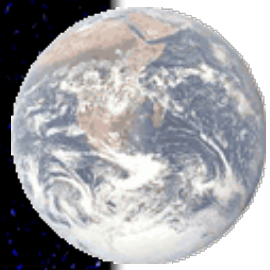


1.2. Dynamic Partitioning

- Degree of multiprogramming limited by number of partitions
- Variable-partition sizes for efficiency
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Process exiting frees its partition, adjacent free partitions combined
- Operating system maintains information about:
  - Allocated partitions
  - Free partitions (hole)

Effect of Dynamic Partitioning



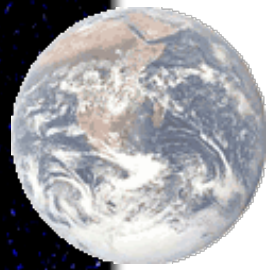


Dynamic-Storage Allocation

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.
- First fit analysis reveals that given n blocks allocated, $n/2$ blocks lost to fragmentation → 50-percent rule

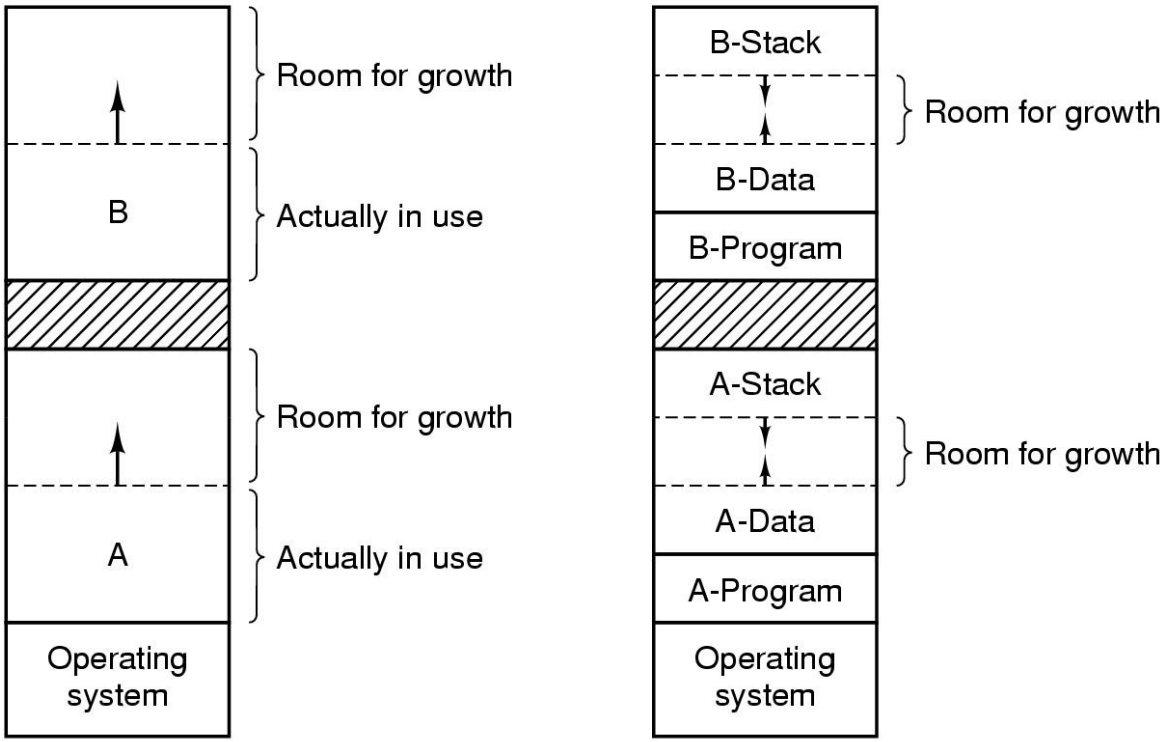


Internal vs. External Fragmentation

- 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; this size difference is memory internal to a partition but not being used.
- 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
  - Involved I/O and must do I/O only into OS buffers

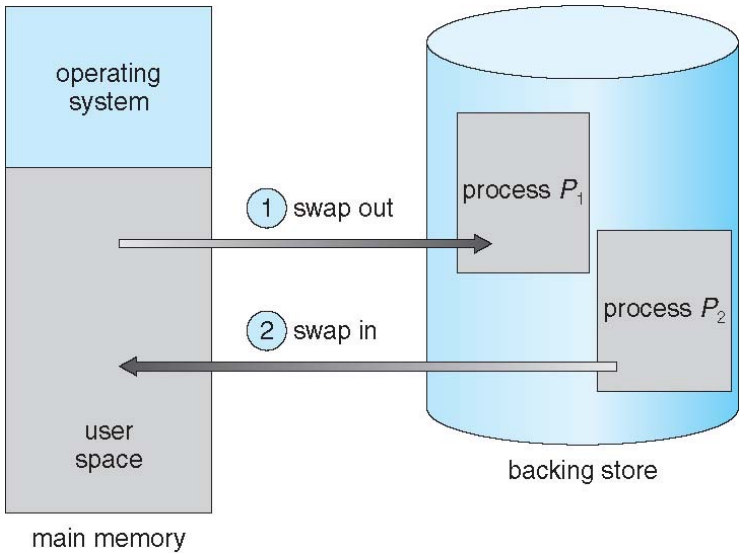
Space Allocation

- Allocating space for growing data segment
- Allocating space for growing stack & data segment

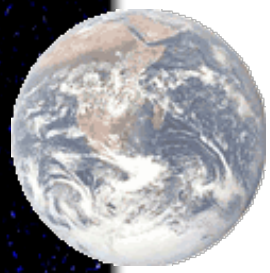


Swapping

- 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.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped



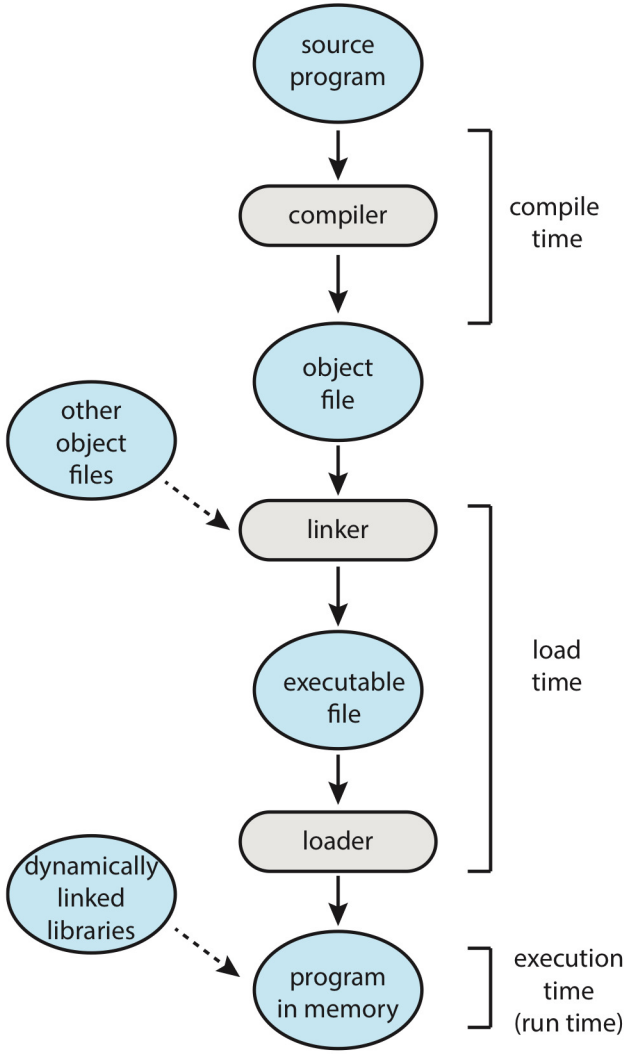
- Does the swapped-out process need to swap back into same physical addresses?
  - Depends on address binding method

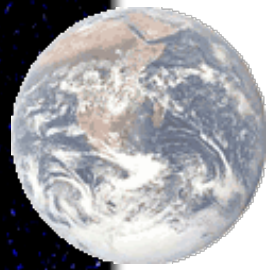


Address Binding

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

- Source code addresses usually symbolic
- Compiled code addresses bind to relocatable addresses
  - i.e., "14 bytes from beginning of this module"
- Linker or loader will bind relocatable addresses to absolute addresses
  - i.e., 74014
- Each binding maps one address space to another

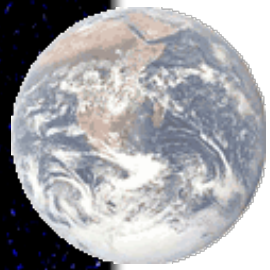




Binding of Instructions and Data to Memory

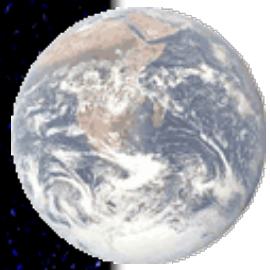
Address binding of instructions and data to memory addresses can happen at three different stages

- Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
- Load time: Must generate relocatable code if memory location is not known at compile time
- Execution time: Binding delayed until run time when reference is made to location in memory
  - Need hardware support for address maps (e.g., base and bound (or limit) registers)



Logical vs. Physical Address Space

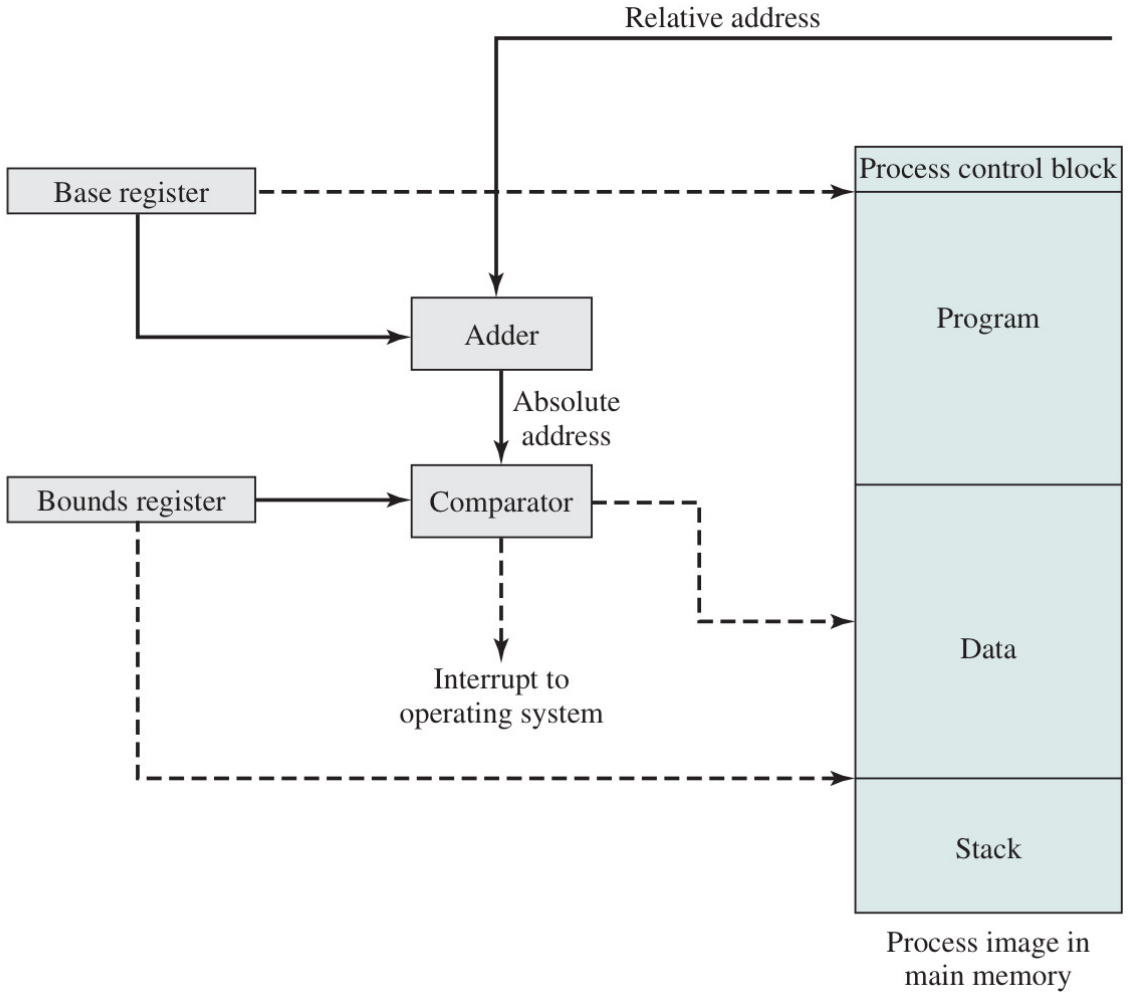
- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - Logical address – generated by the CPU; also referred to as virtual address
  - Physical address – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- Logical address space is the set of all logical addresses generated by a program
- Physical address space is the set of all physical addresses generated by a program

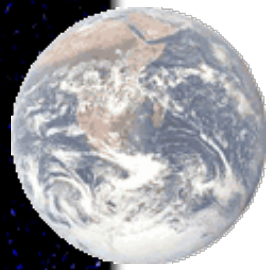


Base and Bound (or Limit) Registers

- A pair of base and bound registers define the logical address space.
- Base register contains starting address in memory of the process.
- Bound register indicates the ending location of the process
- CPU must check every memory access generated in user mode to be sure it is between base and bound for that user.
- If the absolute address is within bounds, then proceed execution.
- Otherwise, an interrupt is generated to the OS, which must respond to an error.

Base and Bound Registers

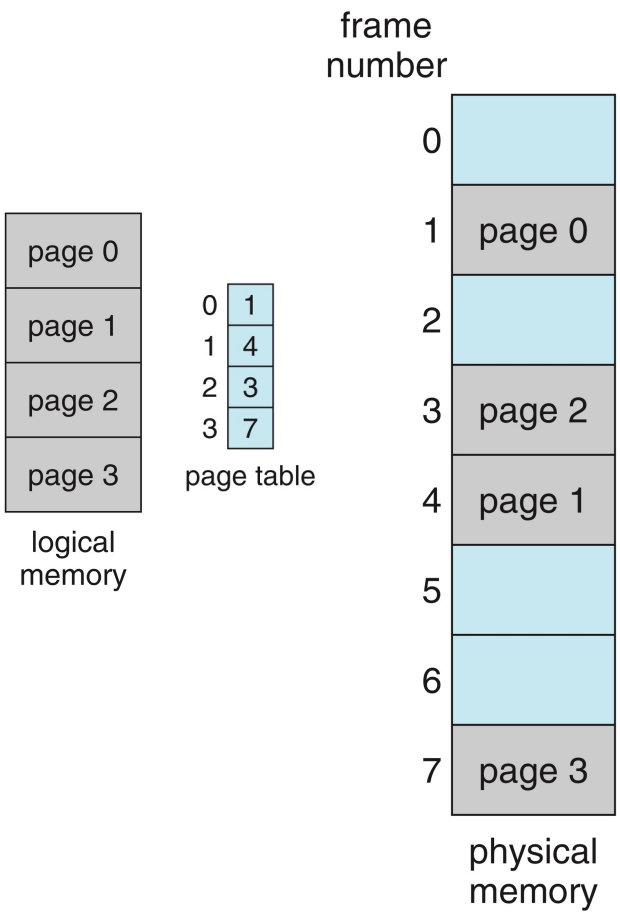


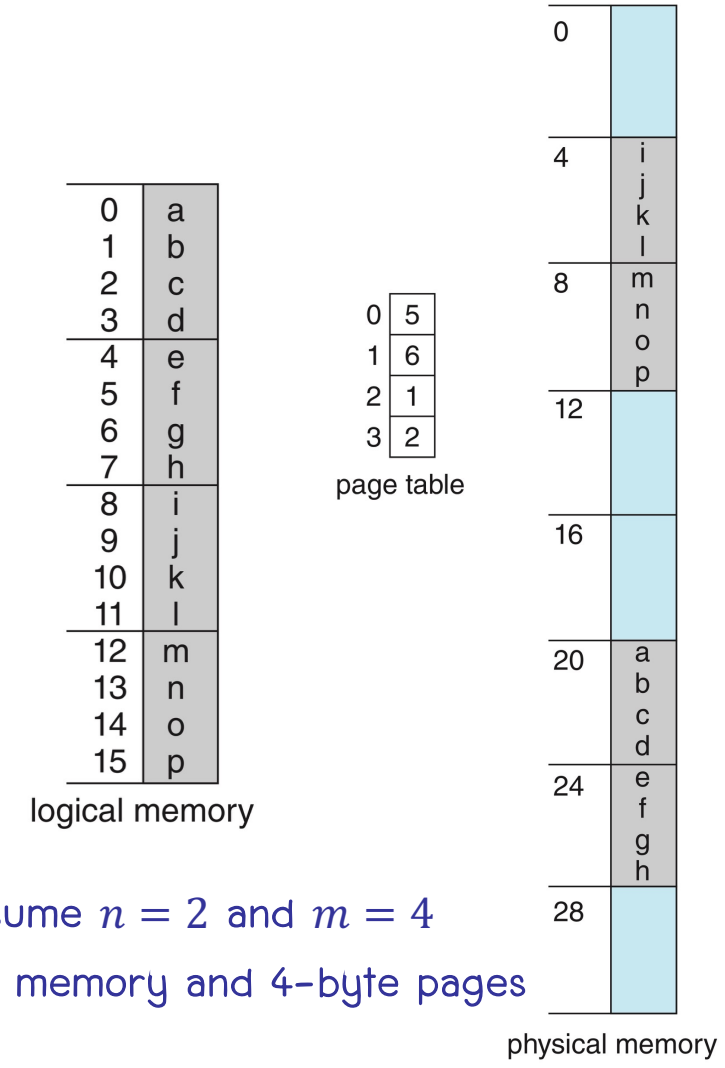


2.1. Simple Paging

- Main memory is divided into a number of equal-size frames.
- Each process is divided into a number of equal-size pagess of the same length as frames.
- A process is loaded by loading all of its pages into available, no necessarily contiguous frames.
- Strength:
  - No external fragmentation
  - Avoids problem of varying sized memory chunks
- Weakness:
  - A small amount of internal fragmentation, depending on page size

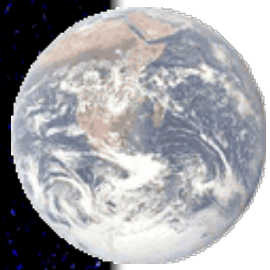
Paging Model of Logical Memory and Physical Memory





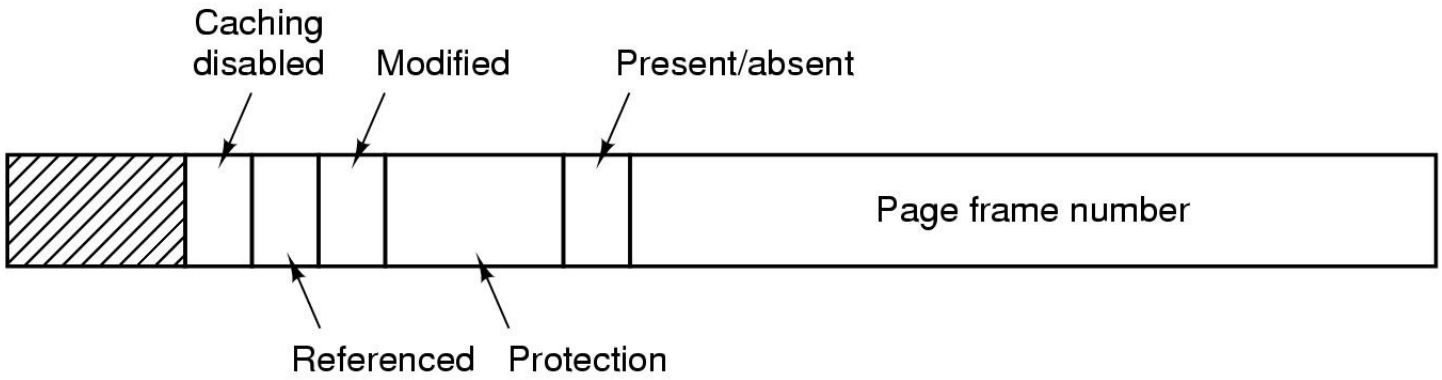
Assume $n = 2$ and $m = 4$

32-byte memory and 4-byte pages

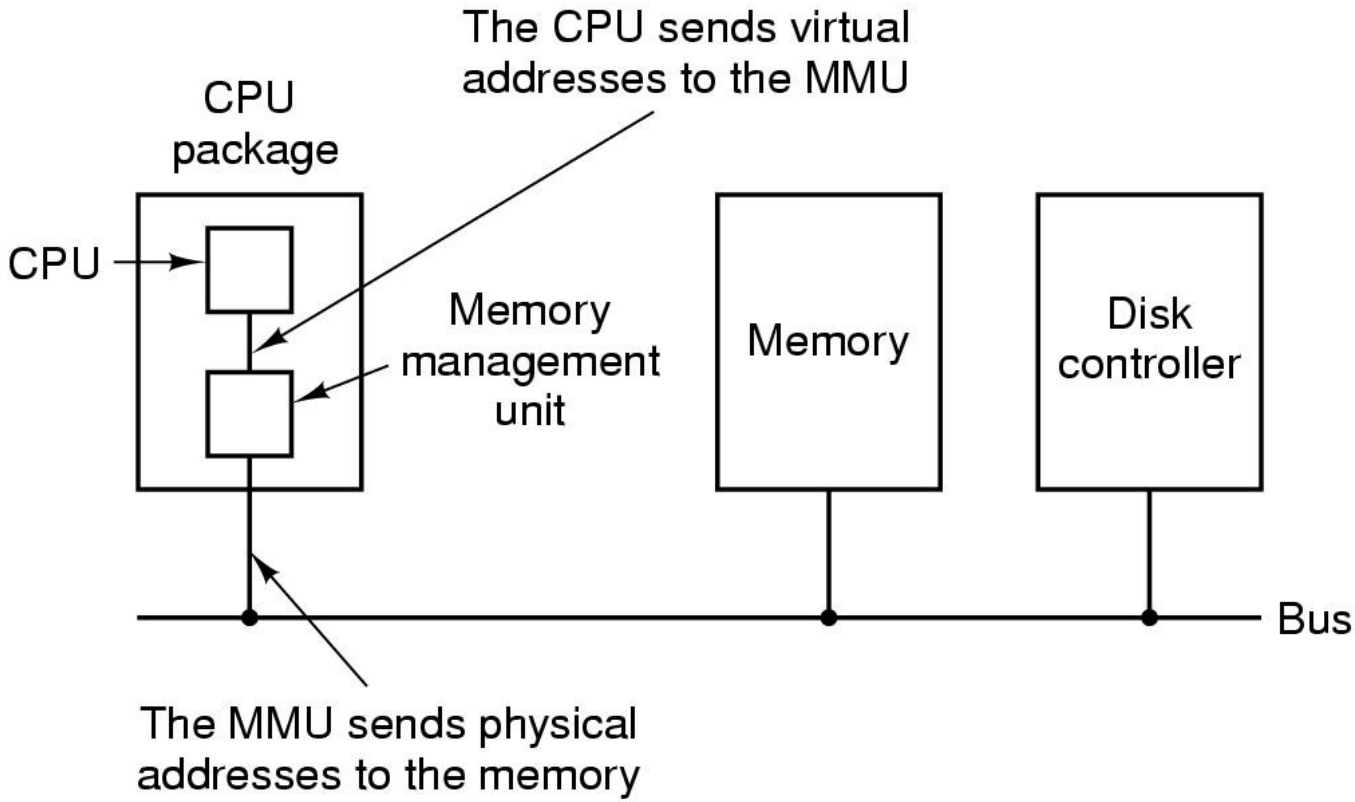


Paging System

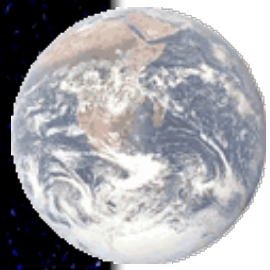
- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page
- Page table is used to map logical addresses to physical addresses.
- Page table entry structure:



Memory Management Unit (MMU)



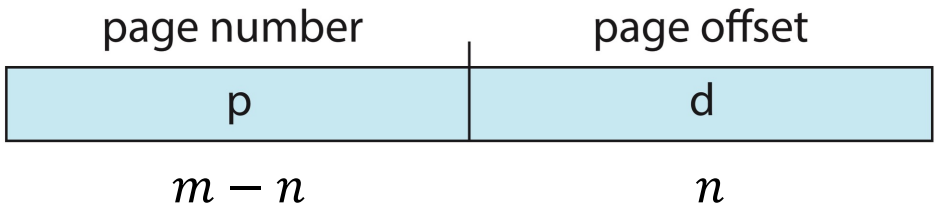
MMU – Hardware device that at run time maps virtual to physical address.



Address Translation Scheme

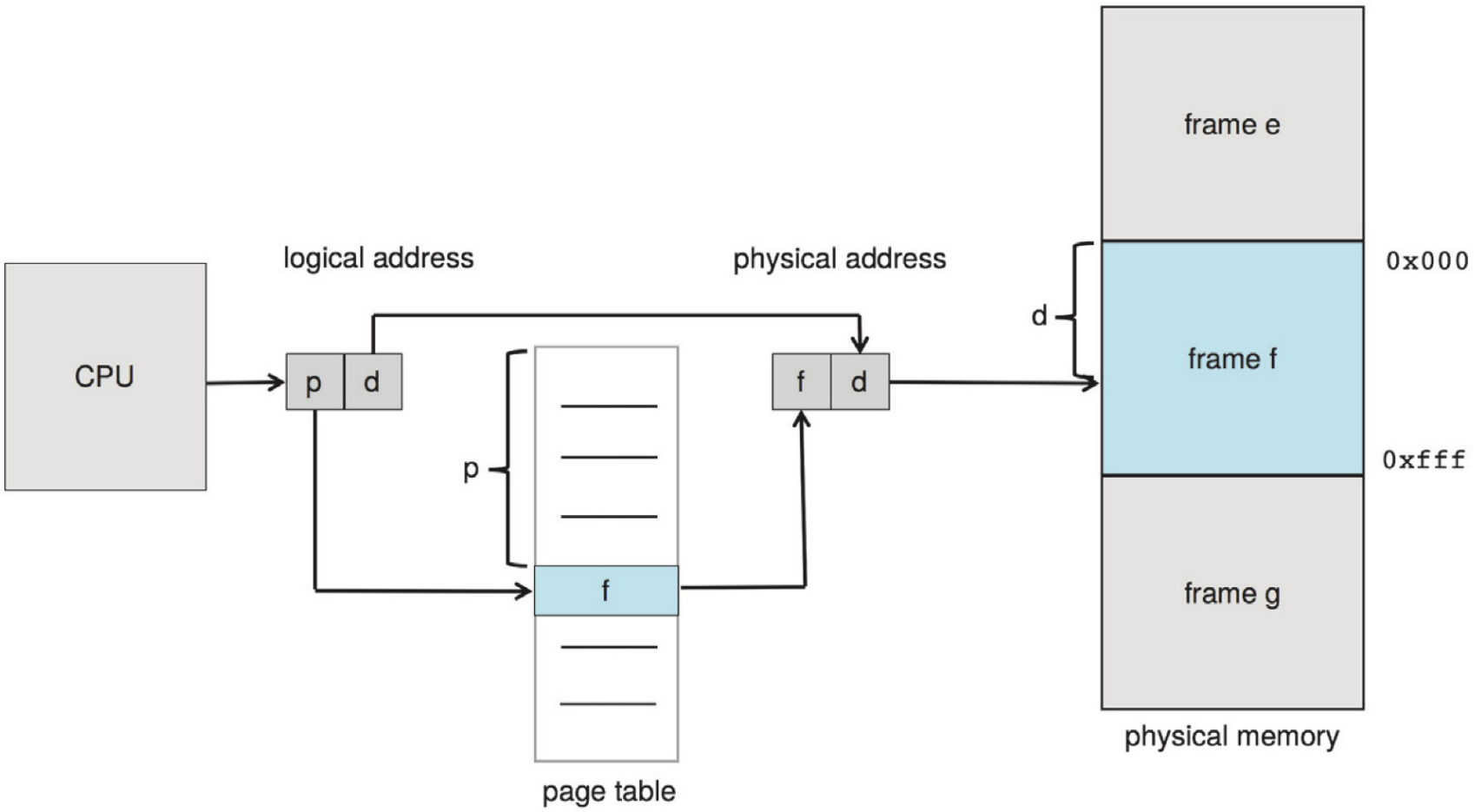
Address generated by CPU is divided into:

- 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



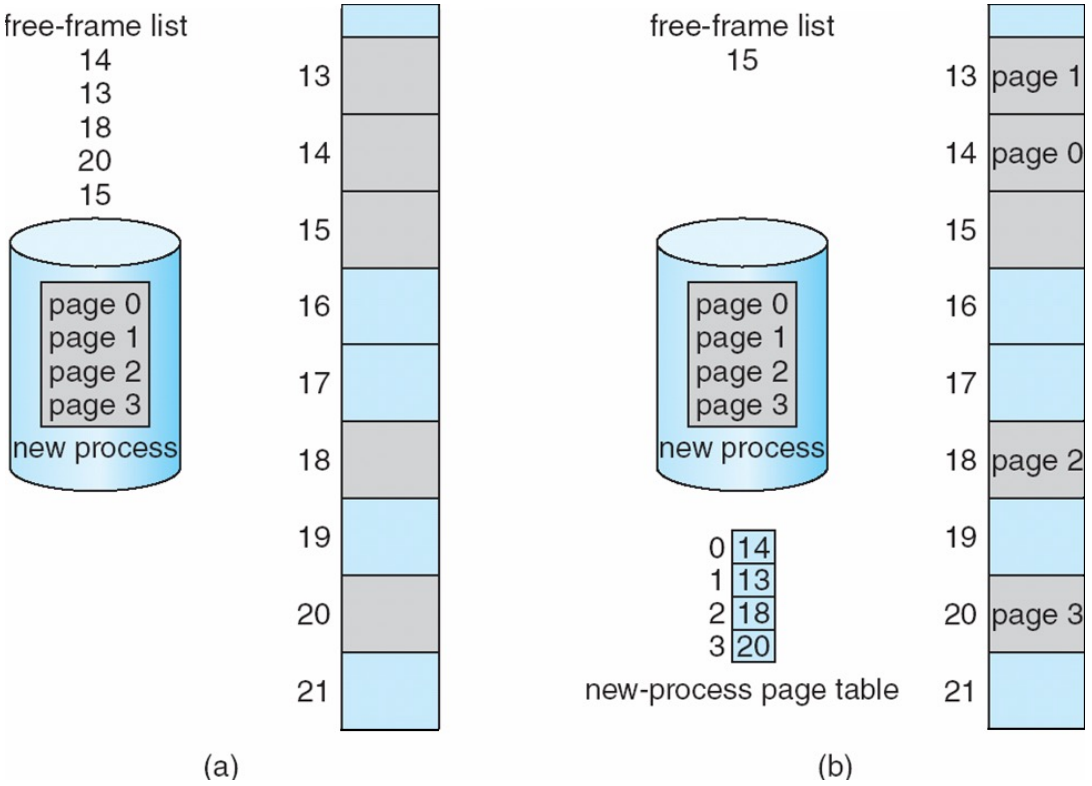
Logical address space is 2^m and page size is 2^n.

Paging Hardware



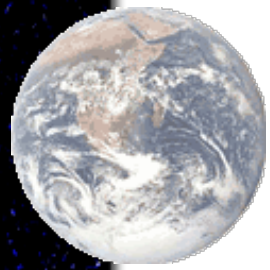
Free Frames

- To run a program of size N pages, need to find N free frames
  - Must keep track of all free frames



Before allocation

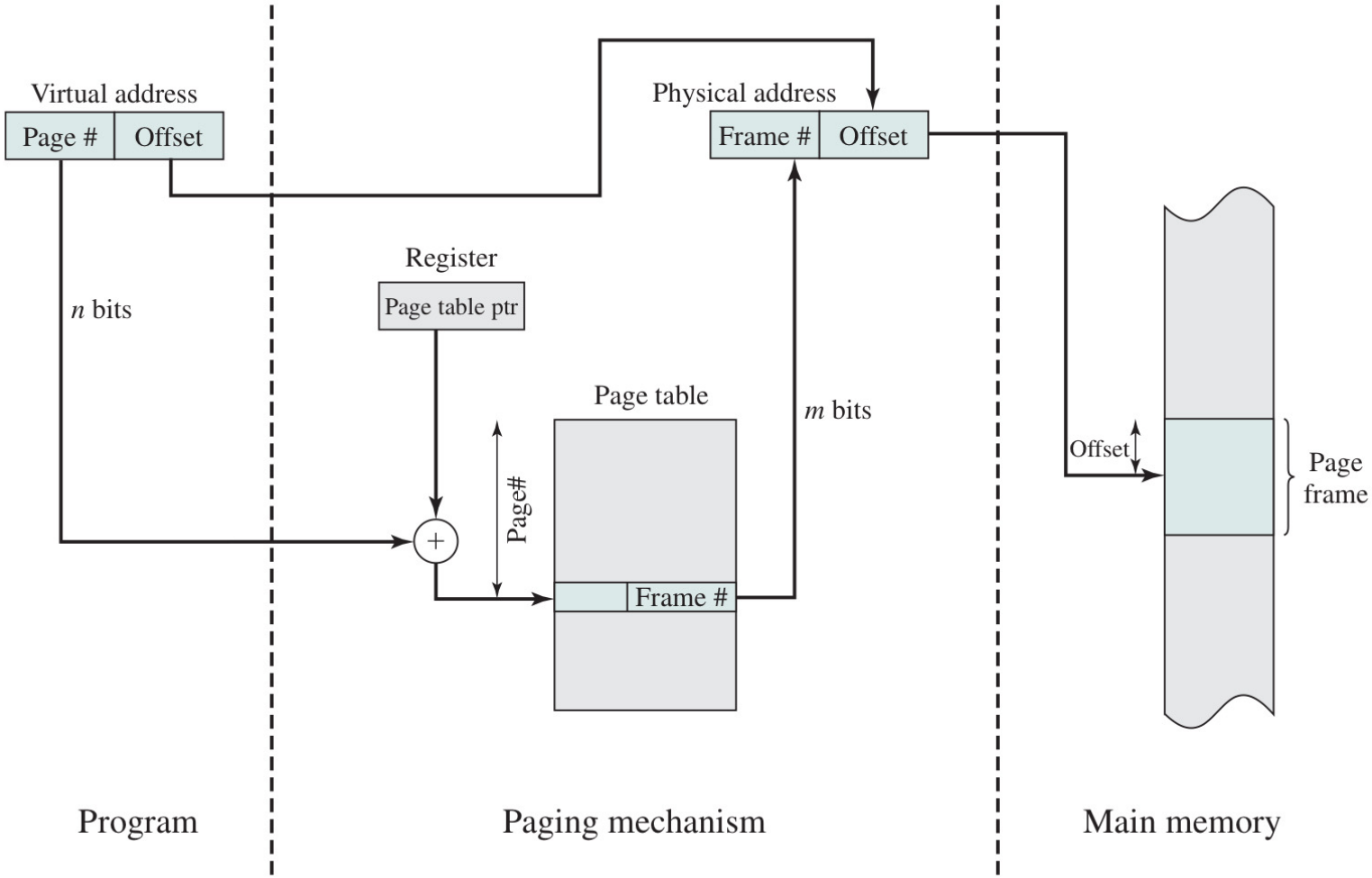
After allocation

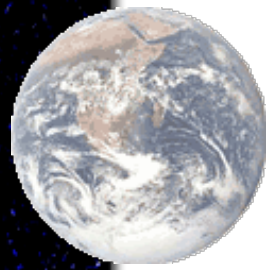


Implementation of Page Table

- 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
- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a special fast-lookup hardware cache is set up for page table entries
  - Called associative memory or translation look-aside buffers (TLBs).

Paging Mechanism

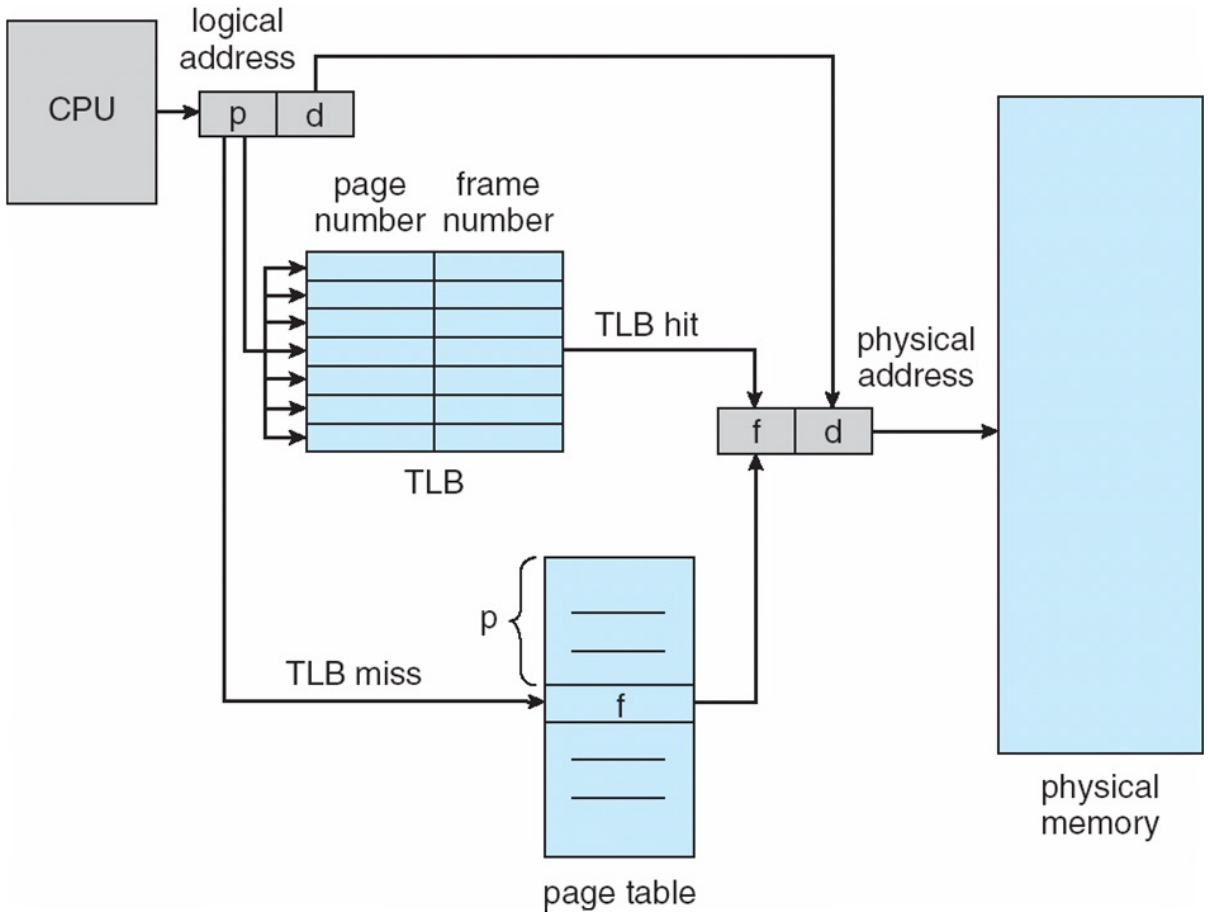


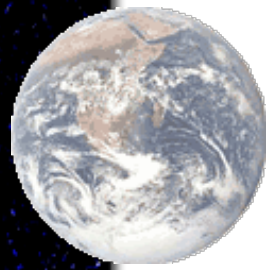


Translation Lookaside Buffer

- TLB contains page table entries that have been most recently used.
- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved, and the real address is formed.
- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table and get frame number from page table in memory.
- TLB lookup uses parallel search.

Address Translation with TLB





Effective Access Time

- TLB Lookup = ε time unit
  - Can be < 10% of memory access time
- 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)

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

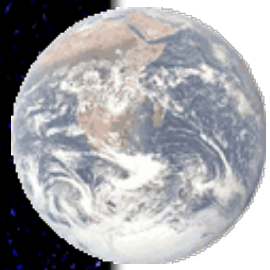
$$= 2 + \varepsilon - \alpha$$

- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access

$$EAT = 0.80 \times 100 + 0.20 \times 200 = 120ns$$

- Consider more realistic hit ratio: α = 99%, ε = 20ns for TLB search, 100ns for memory access

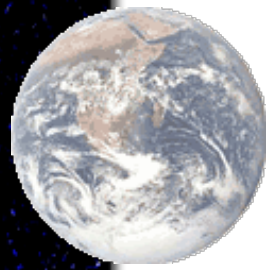
$$EAT = 0.99 \times 100 + 0.01 \times 200 = 101ns$$



TLB Speed Up Paging

TLBs typically small (64 to 1,024 entries)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

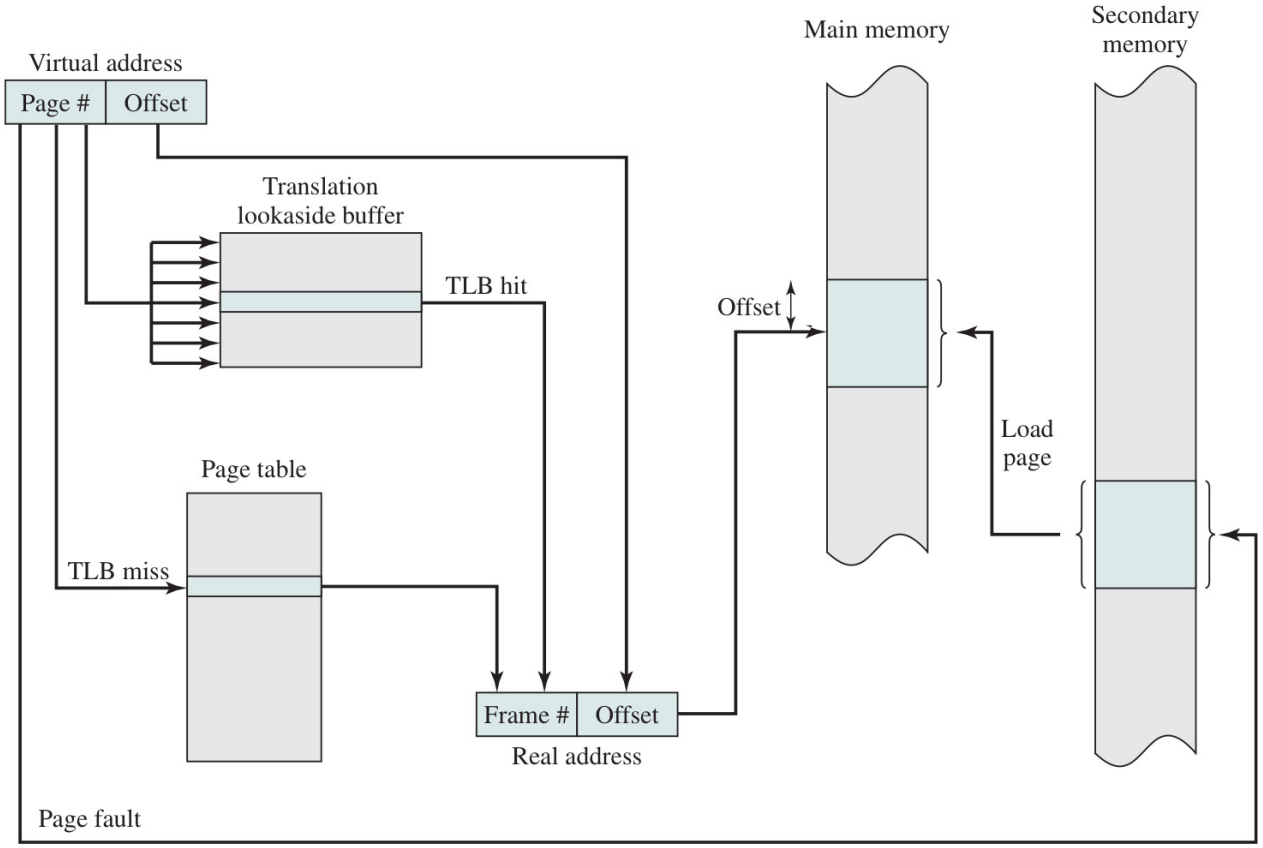


Translation Lookaside Buffer

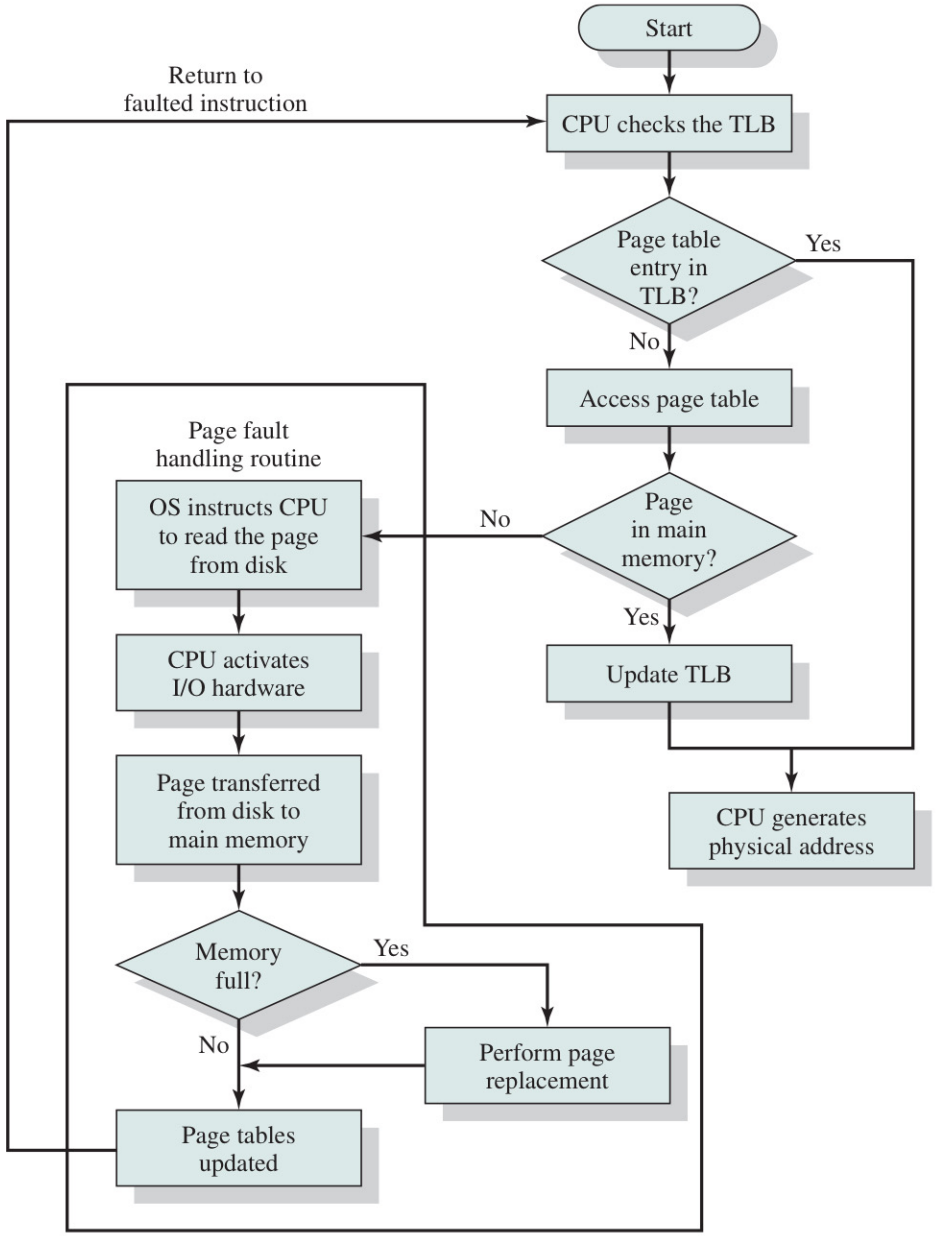
- Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered

Use of TLB

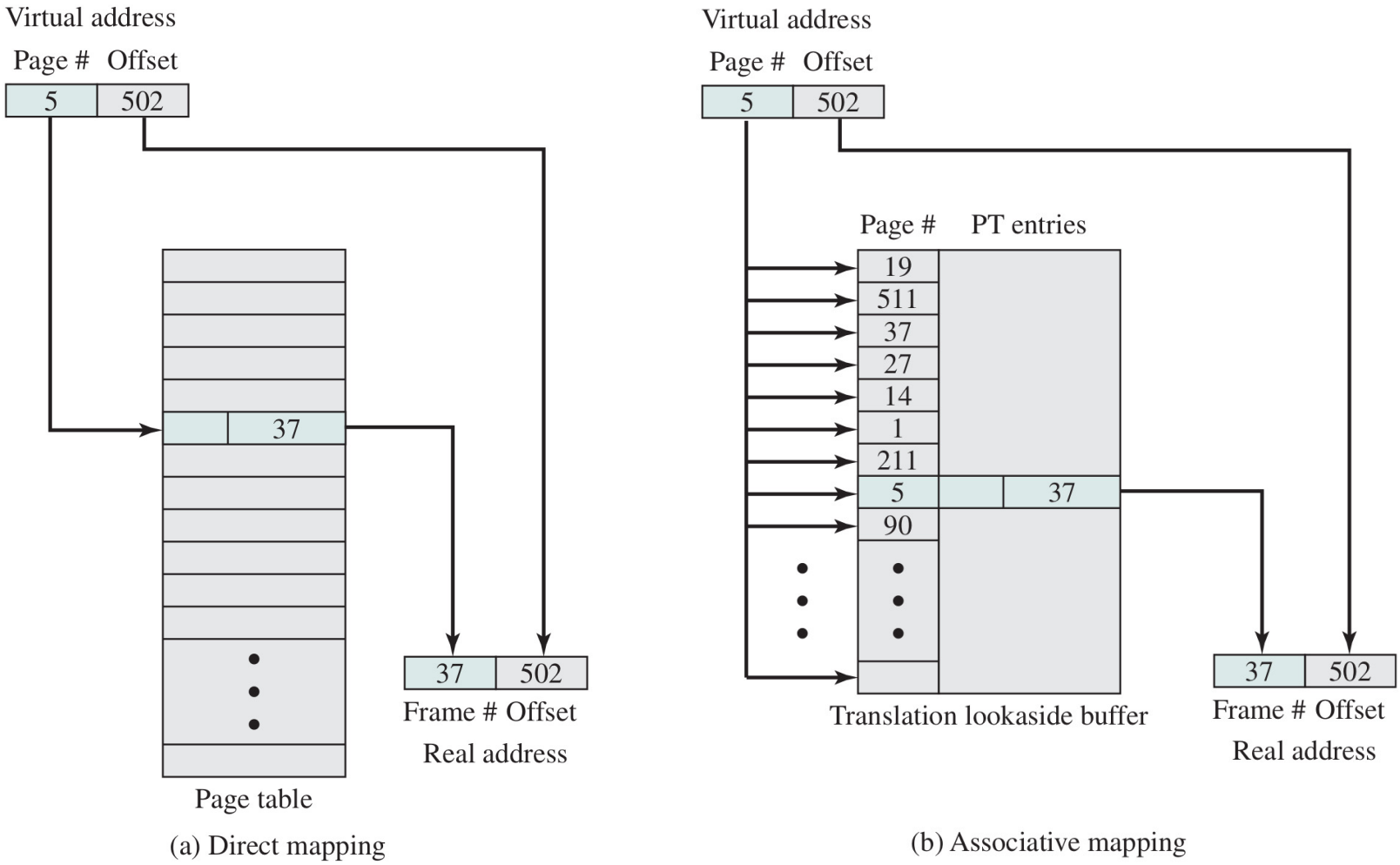
- First checks if page is already in main memory
  - If not in main memory a page fault is issued
- The TLB is updated to include the new page entry



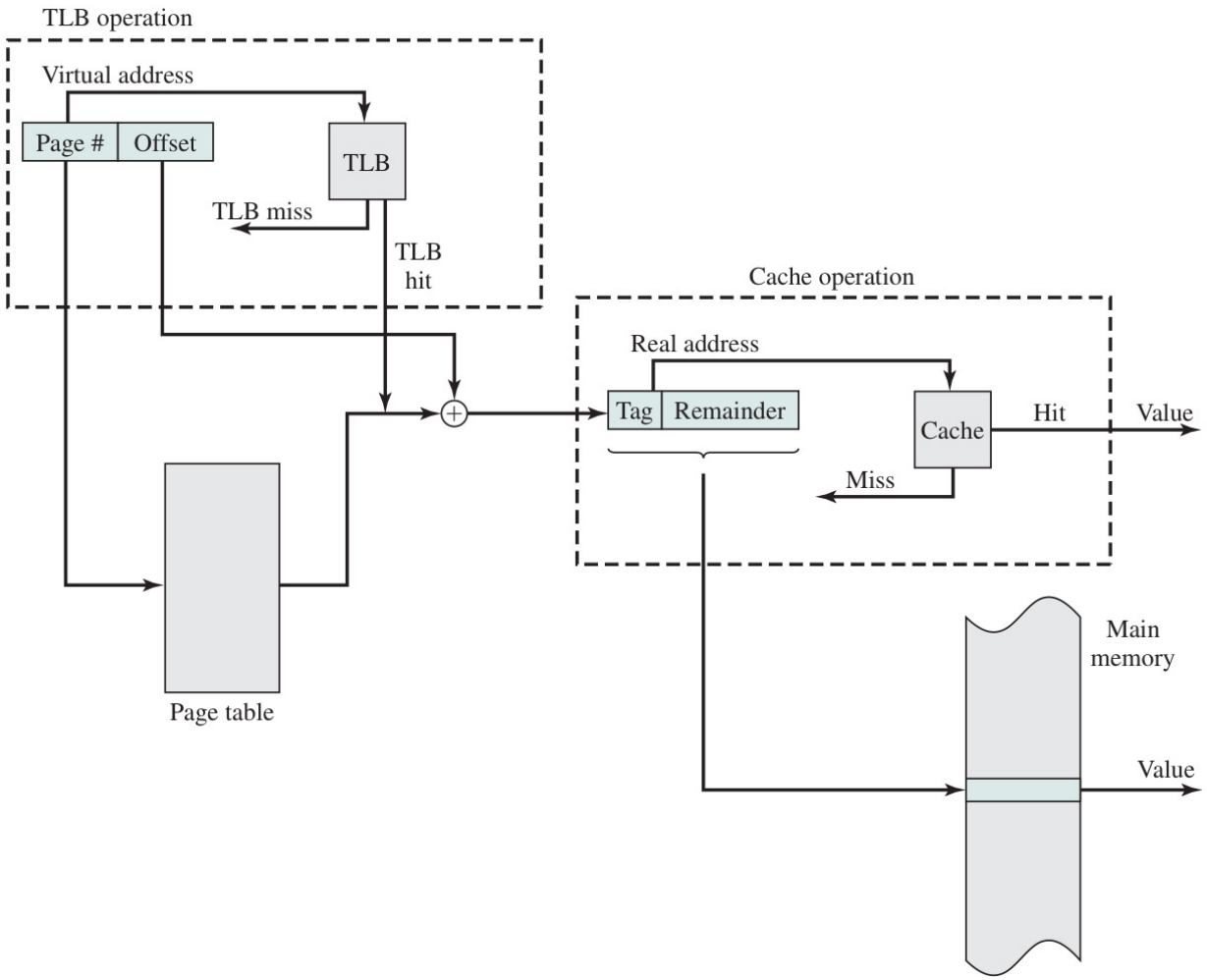
Operation of Paging and TLB

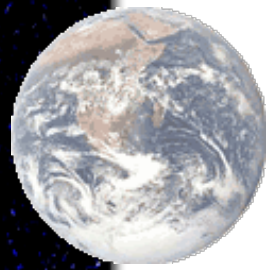


Direct vs. TLB Lookup



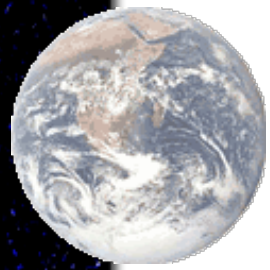
TLB and Cache Operation





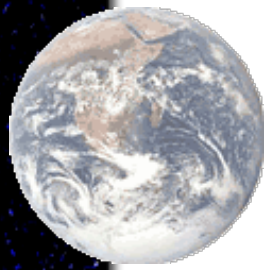
Structure of the Page Table

- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space
    - Logical address space is of …… in bytes.
  - Page size of 4 KB = …… in bytes.
  - Number of entries in the page table is ……
  - If each page table entry is 4 bytes
    - Page table size is ……
- That amount of memory used to cost a lot.
- Don't want to allocate that contiguously in main memory



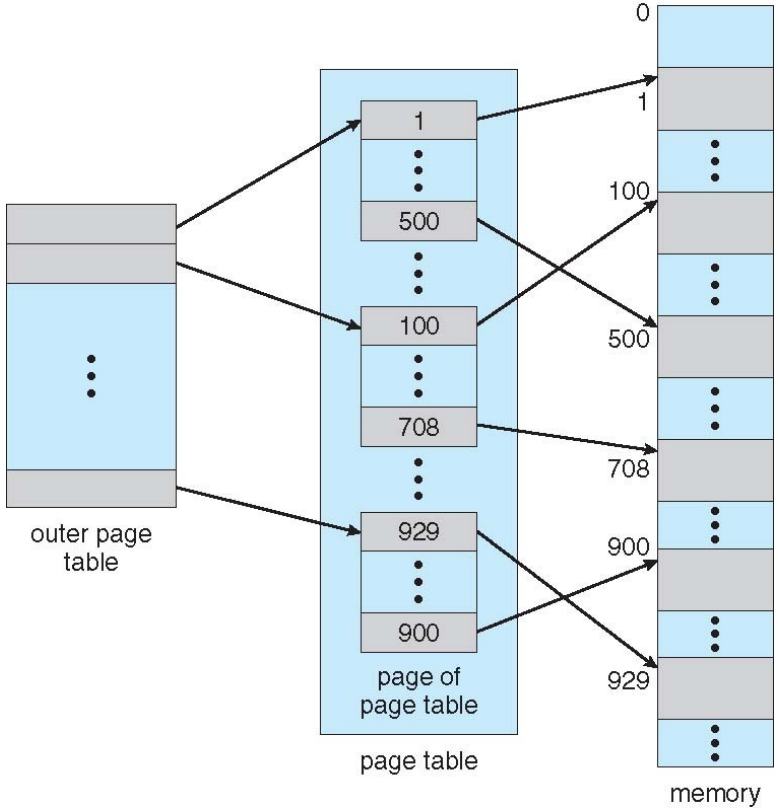
Structure of the Page Table

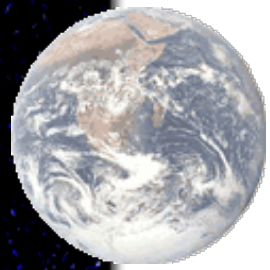
1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables



1. Hierarchical Page Tables

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



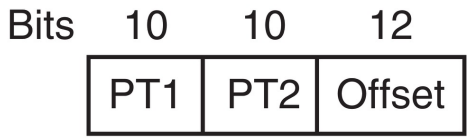


Two-Level Paging Example

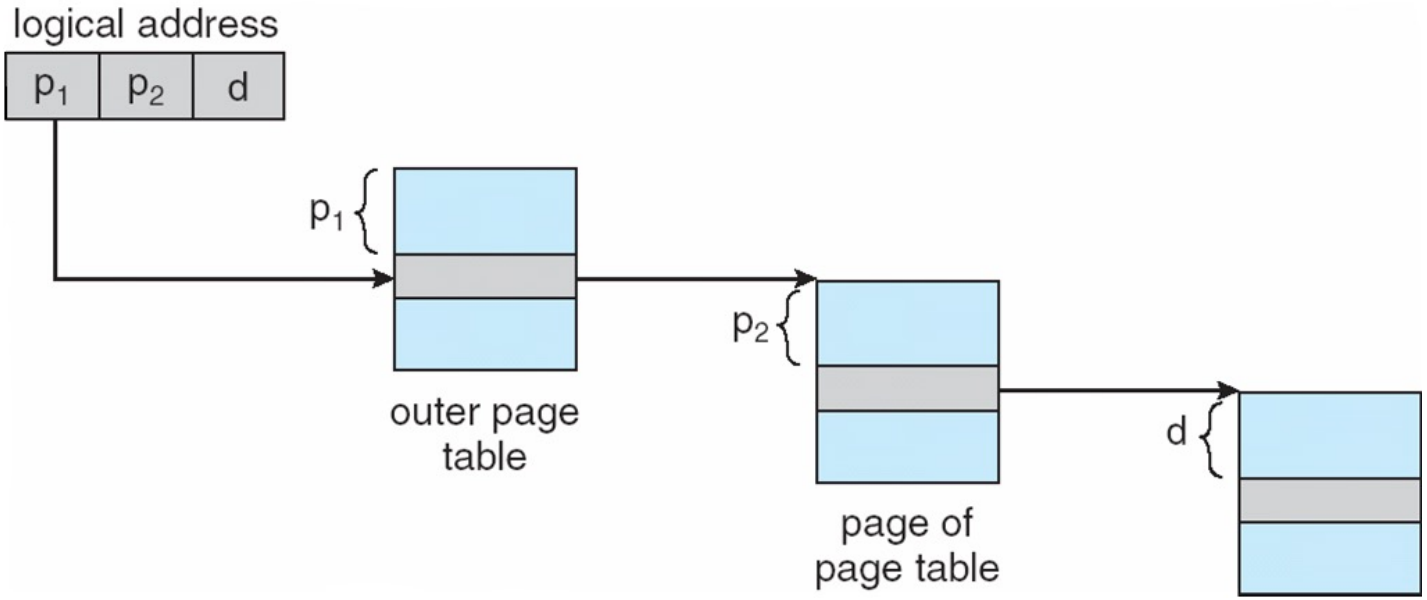
- Assume on a 32-bit address machine with 4KB page size
- A 32-bit logical address is divided into:
  - a page number consisting of 20 bits
    - Total number of pages is .....
  - a page offset consisting of 12 bits - where this come from?
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
    - Number of pages in the outer level is .....
  - a 10-bit page offset
    - Number of page tables of the inner level is .....

Address Translation Scheme

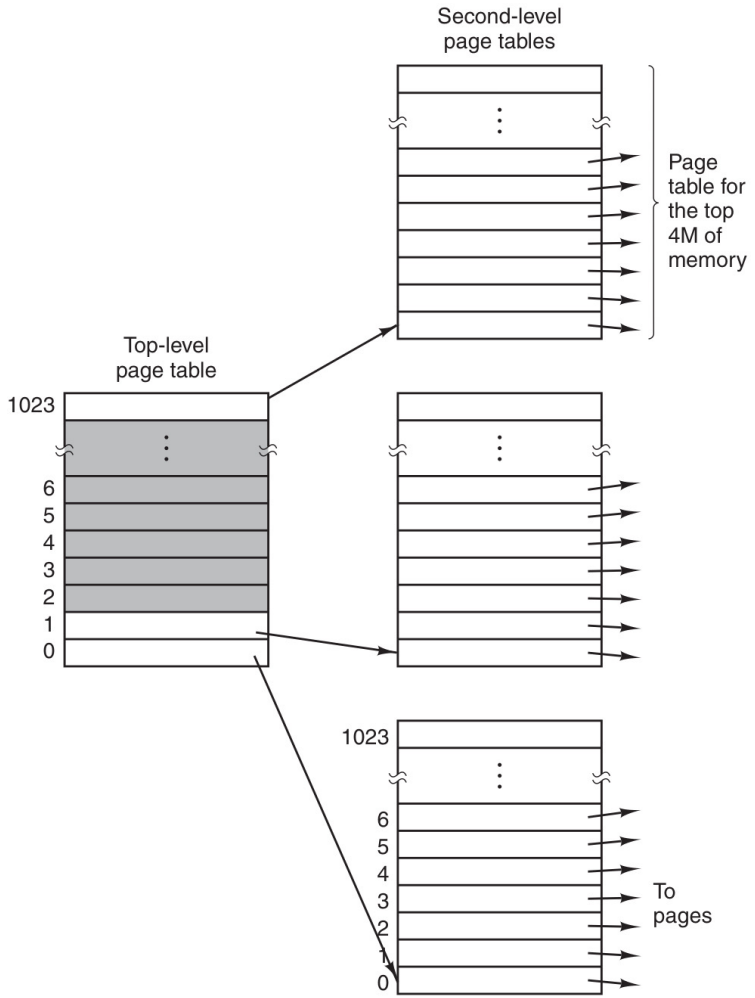
- Thus, a logical address is as follows:

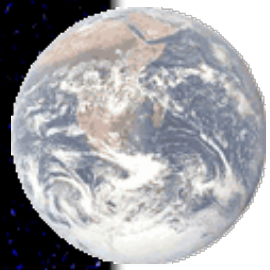


where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table.



Two-Level Page Table

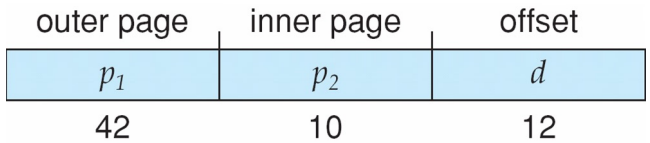




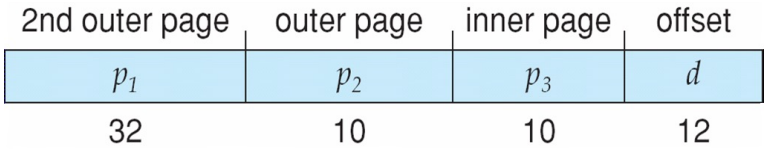
64-bit Logical Address Space

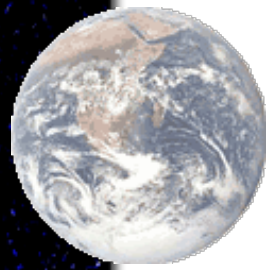
- Even two-level paging scheme not sufficient
- Assume on a 64-bit address machine with 4KB page size
  - Then page table has …… entries

- If two level scheme, inner page tables could be 2^{10} 4-byte entries
- 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 table is still 2^{34} bytes in size
- And possibly 4 memory access to get to one physical memory location

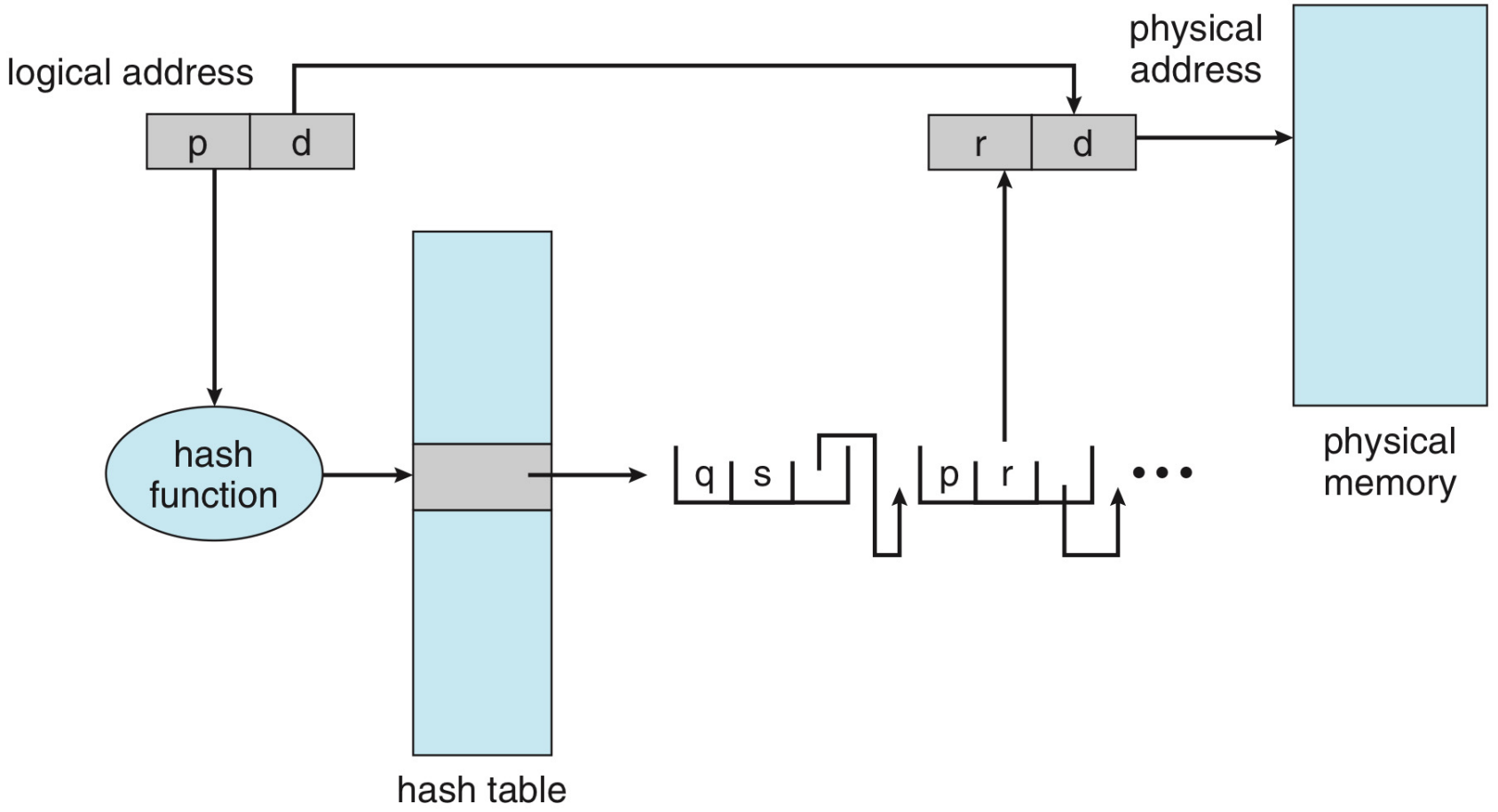


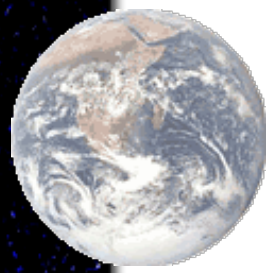


2. Hashed Page Tables

- Common in address > 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

Hashed Page Table

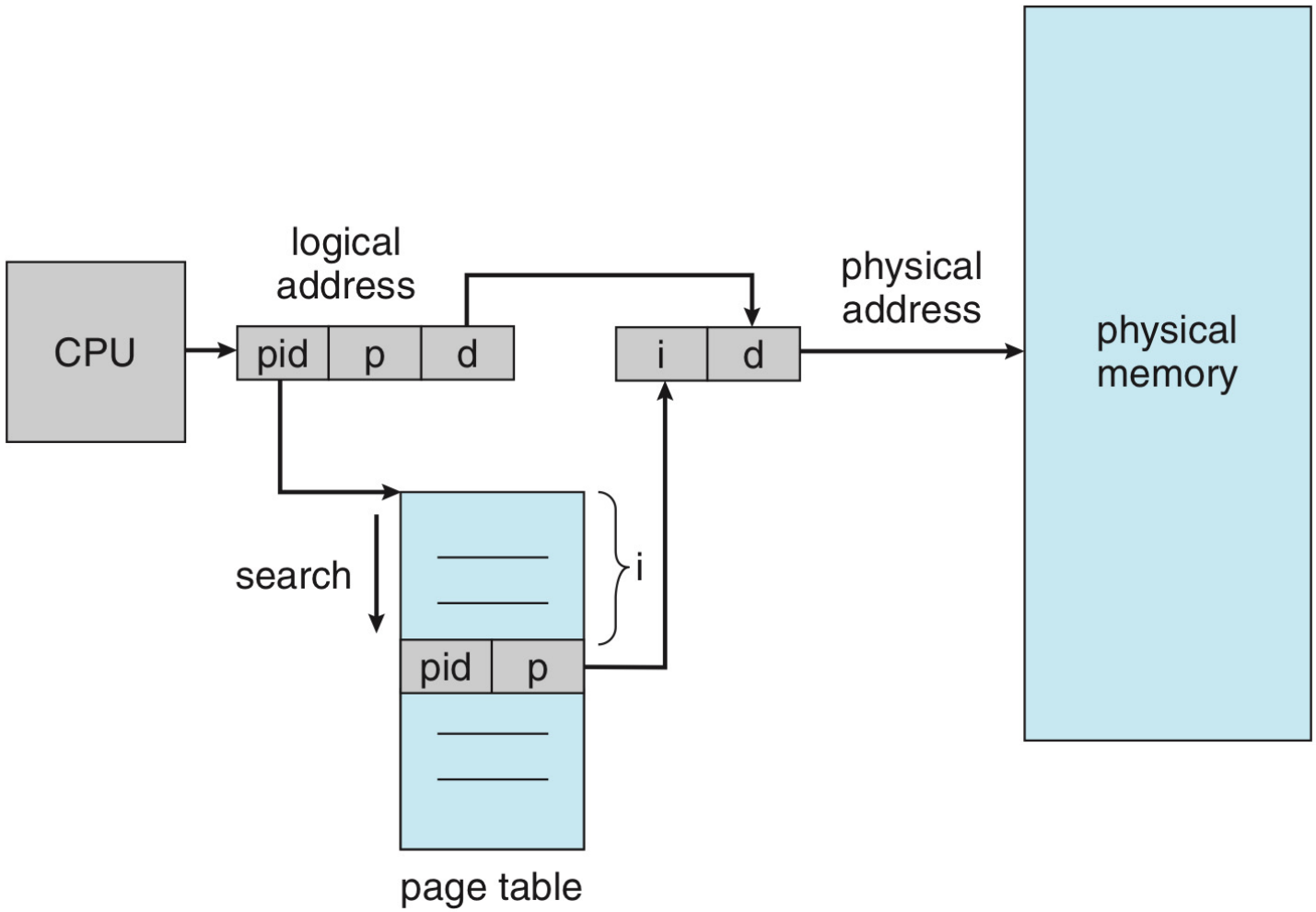




3. Inverted Page Table

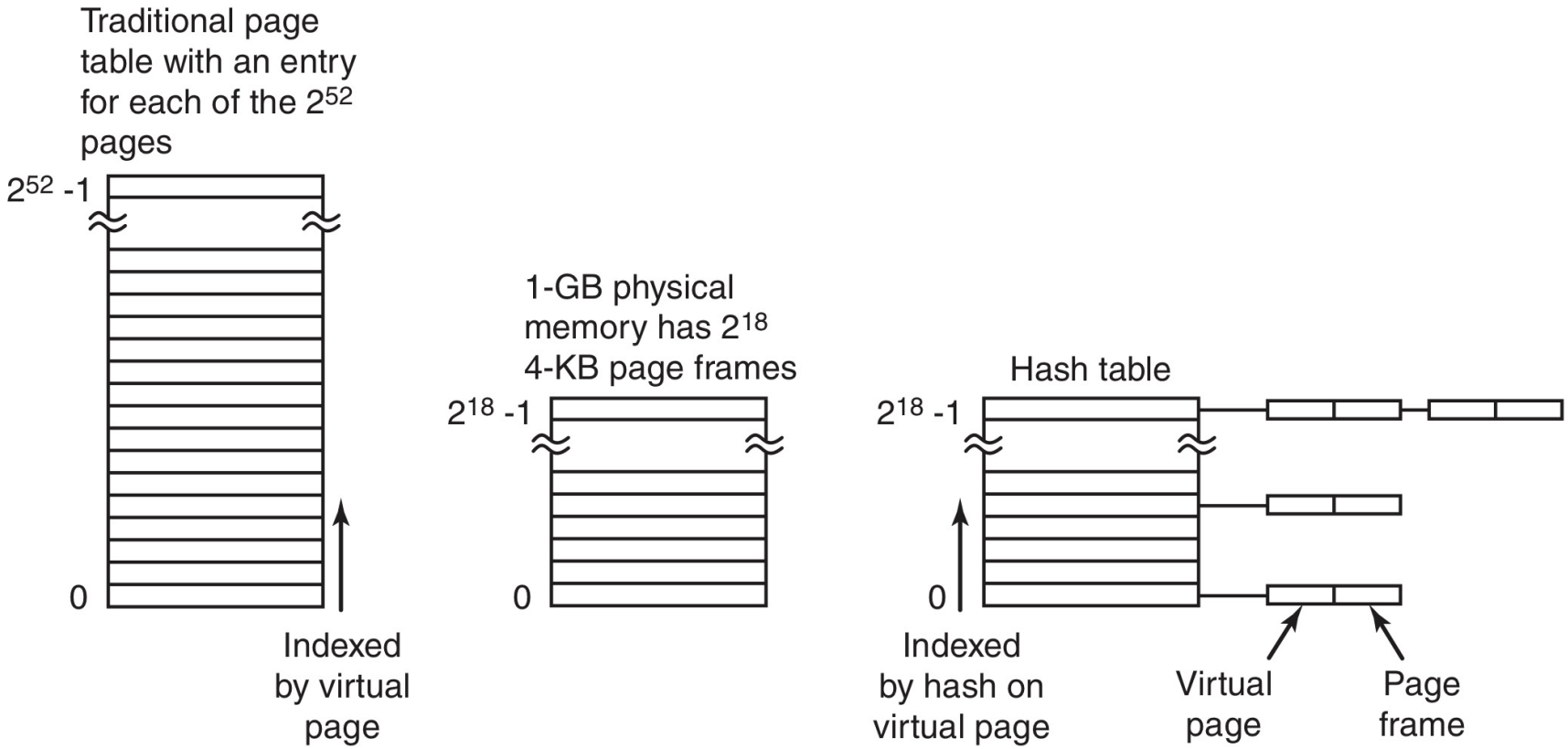
- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.
- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that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

Inverted Page Table Architecture

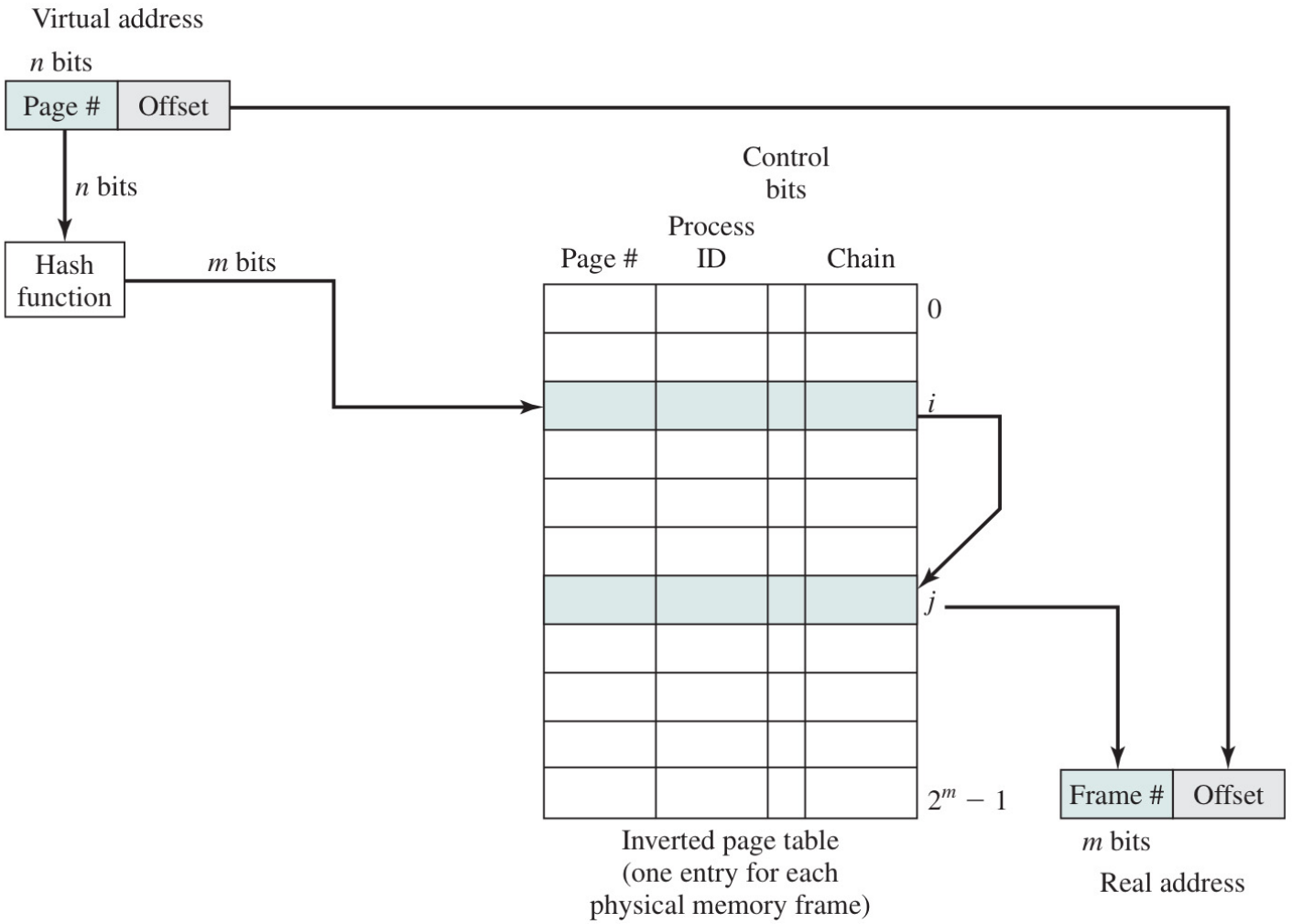


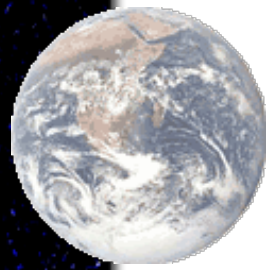
Inverted Page Table

Comparison of a traditional page table with an inverted page table.



Hashed Inverted Page Table

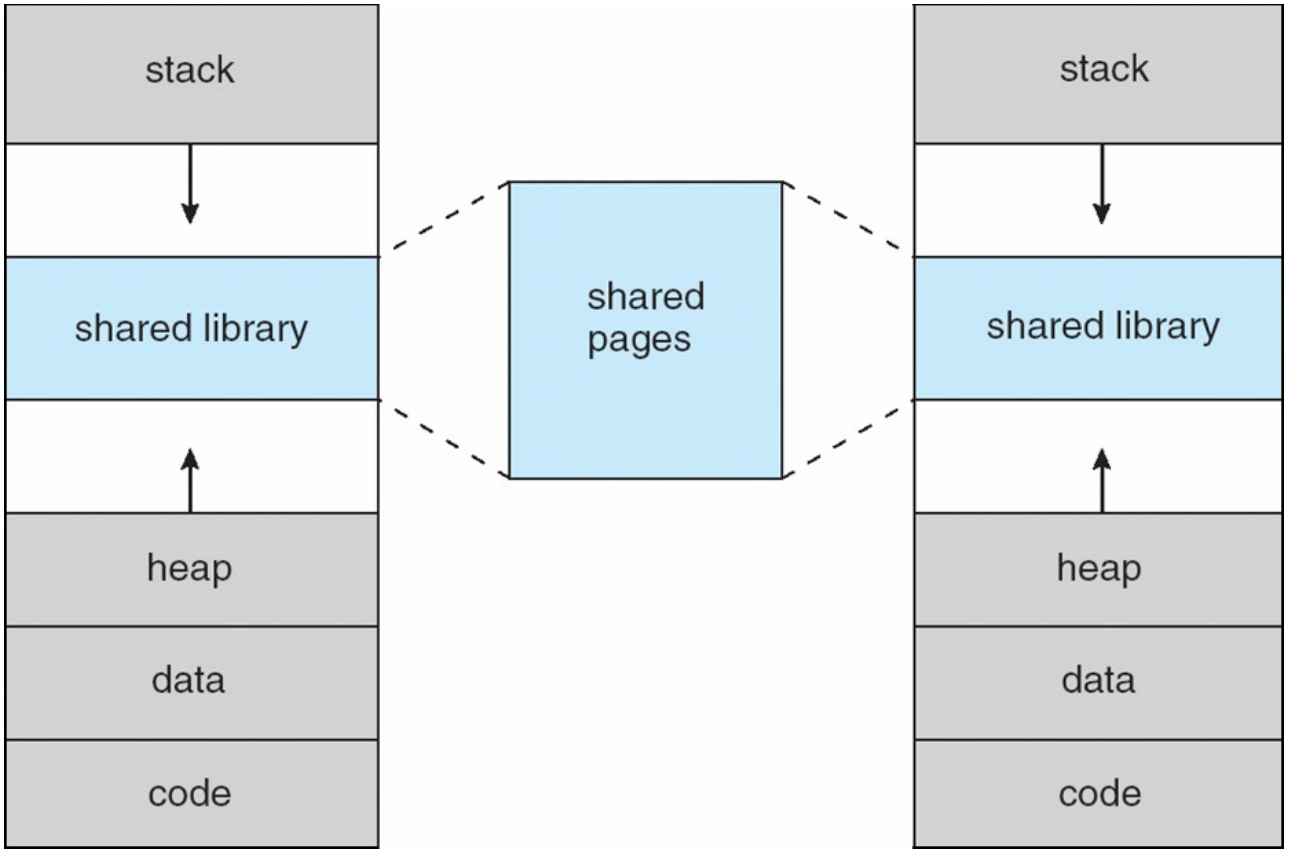




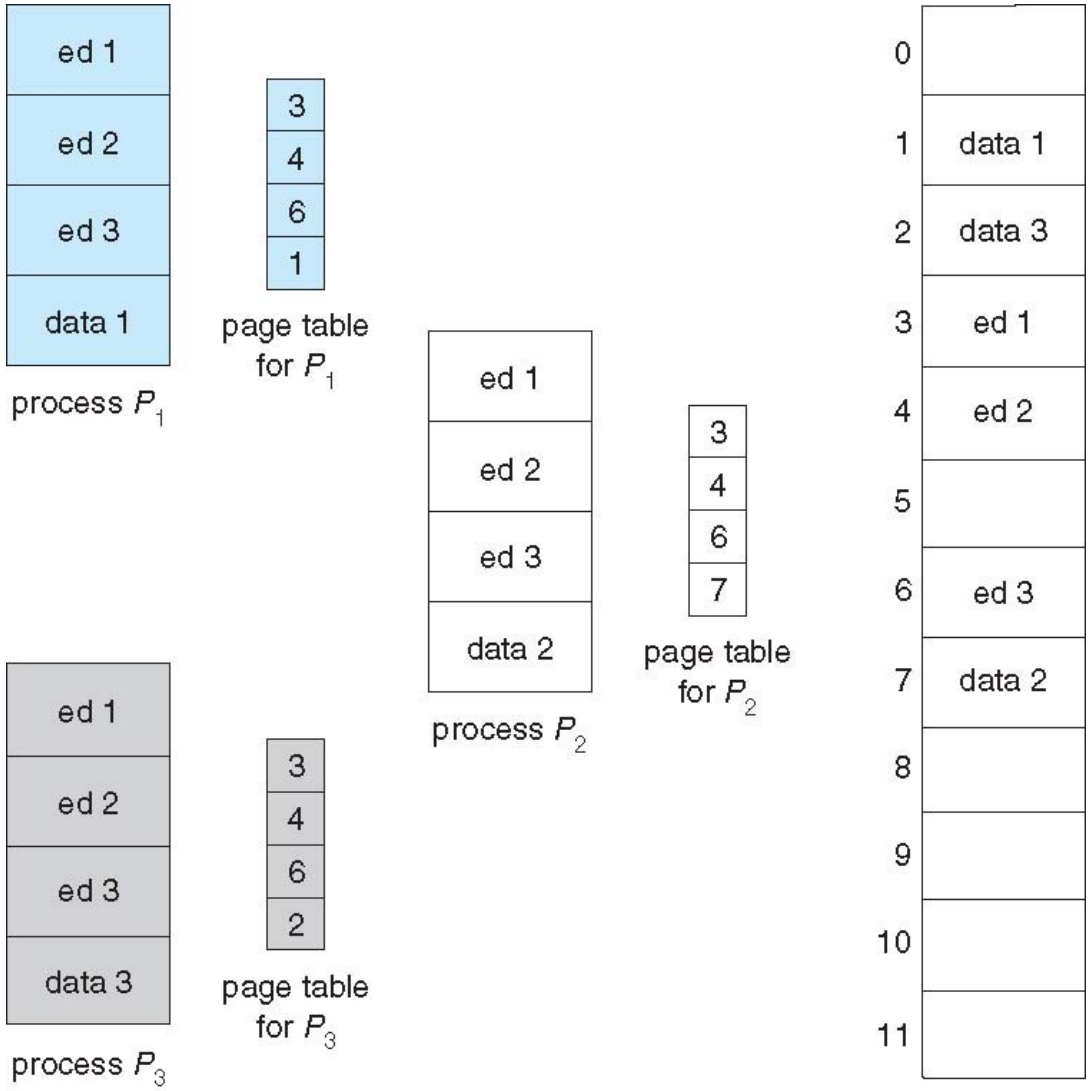
Shared Pages

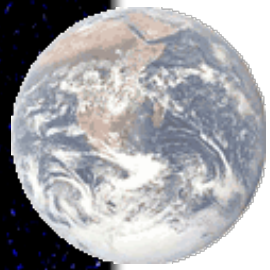
- Virtual memory allows files and memory to be shared by two or more processes through page sharing.
- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also 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 Library Using Virtual Memory



Shared Page Example



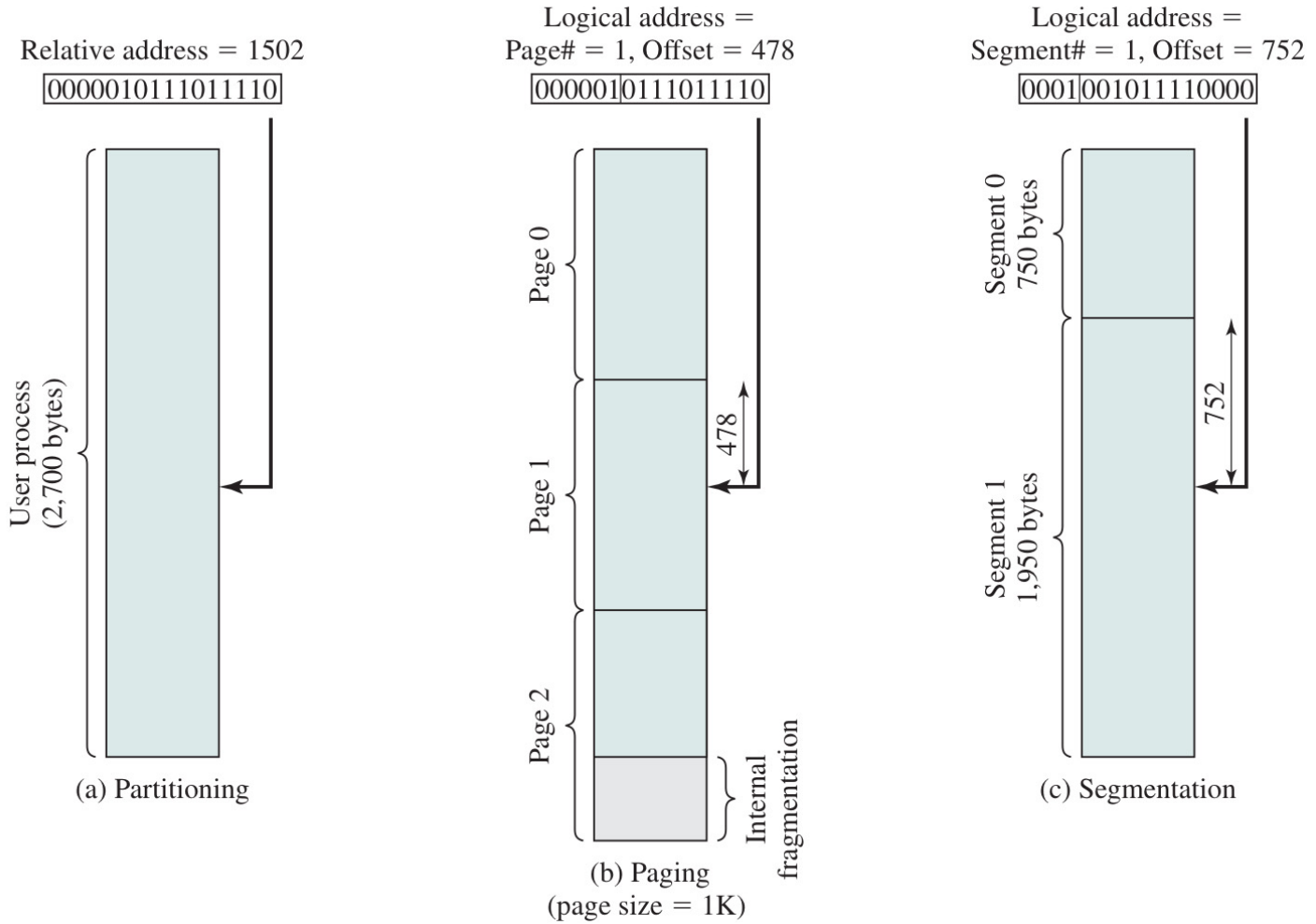


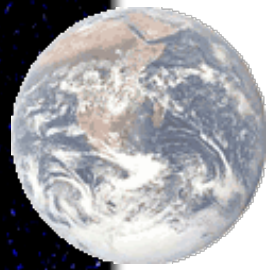
2.2. Simple Segmentation

- Each process is divided into a number of segments.
- A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.
- Strength:
  - No internal fragmentation
  - Improve memory utilization and reduced overhead campared to fixed partitioning.
- Weakness:
  - External fragmentation

Logical Address

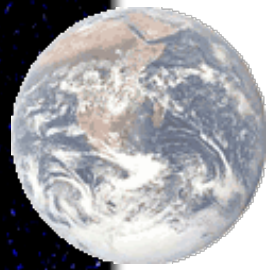
Illustrate the same 16-bit logical address, which is 0000 0101 1101 1110_2, of a user process of size 2,700 bytes under three different memory management schemes.





2.3. Virtual Memory Paging

- As with simple paging, except that it is not necessary to load all of the pages of a process.
- Nonresident pages that are needed are brought in later automatically.
- Strength:
  - No external fragmentation
  - Higher degree of multiprogramming
  - Large virtual address space
- Weakness:
  - Overhead of complex memory management

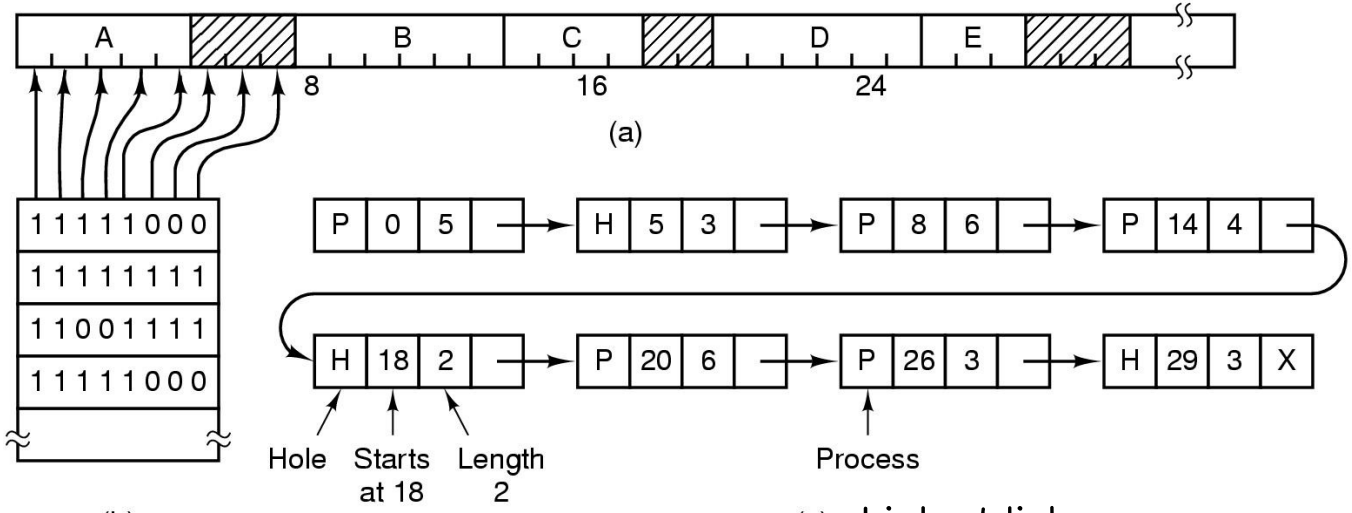


2.4. Virtual Memory Segmentation

- As with simple segmentation, except that it is not necessary to load all of the segments of a process.
- Nonresident segments that are needed are brought in later automatically.
- Strength:
  - No internal fragmentation
  - Higher degree of multiprogramming
  - Large virtual address space
  - Protection and sharing support
- Weakness:
  - Overhead of complex memory management

Memory Space Management

Part of memory with 5 processes, 3 holes; Shaded regions are free

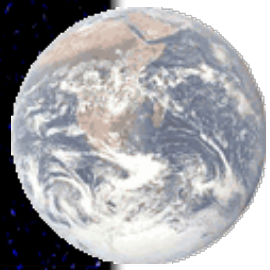


Bit map (b)

(c) Linked lists

- The size of a bit map depends on the memory size and the allocation unit size.
- Slow operation of a sequential search to locate for holes of size k units to be occupied by a process.

- Nodes in linked list are sorted by memory address.
- Node structure includes node type (process/hole), start address, length.



References

1. Modern Operating Systems, 4th edition, Andrew S. Tanenbaum, Herbert Bos
2. Operating Systems, 3rd edition, H.M.Deitel, Pearson Education Limited: Longman.
3. Operating System Concepts, 10th edition, Abraham Silberschatz, Yale University, Peter Baer Galvin, Pluribus Networks, Greg Gagne, Westminster College, 10th edition. Wiley.
4. Operating Systems: Internals and Design Principles, 7th edition, William Stallings.