Principles of Operating Systems

Lecture 6 - Main Memory

Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from previous slides by Prof. Nalini Venkatasubramanian, and course text slides © Silberschatz*]

Virtualizing Resources

- Physical Reality: Processes/Threads share the same hardware
  - Need to multiplex CPU (CPU Scheduling)
  - Need to multiplex use of memory (Topic of these slides)
- Why worry about memory multiplexing?
  - The complete working state of a process and/or kernel is defined by its data in memory (and registers).
  - Consequently, cannot just let different processes use the same memory
  - Also, don’t want different processes to even have access to each other’s memory (protection)

Important aspects of memory multiplexing

- Controlled overlap:
  - Processes should not collide in physical memory
  - Conversely, would like the ability to share memory when desired (for inter-process communication)
- Protection:
  - Prevent access to private memory of other processes
  - Kernel data protected from user programs
- Translation:
  - Ability to translate accesses from one address space (logical/virtual) to a different one (physical)
  - Process uses logical/virtual addresses, physical memory uses physical addresses

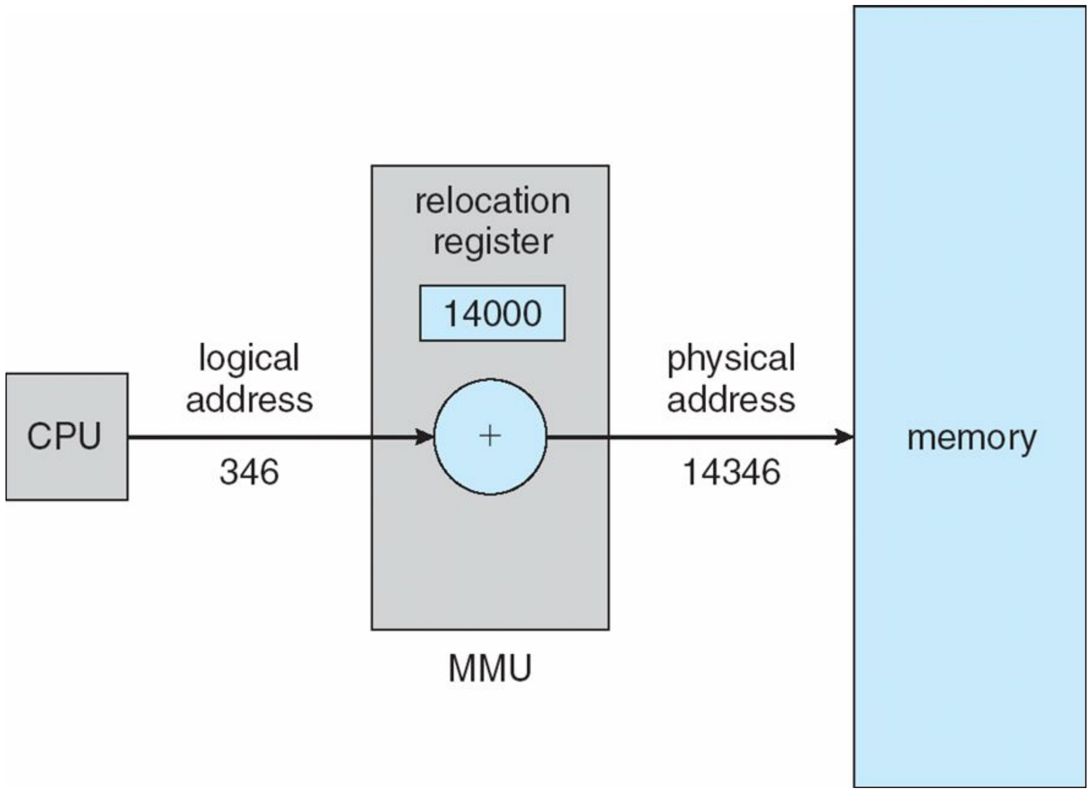
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - Logical Address: or virtual address - generated by CPU
  - Physical Address: address seen by memory unit.

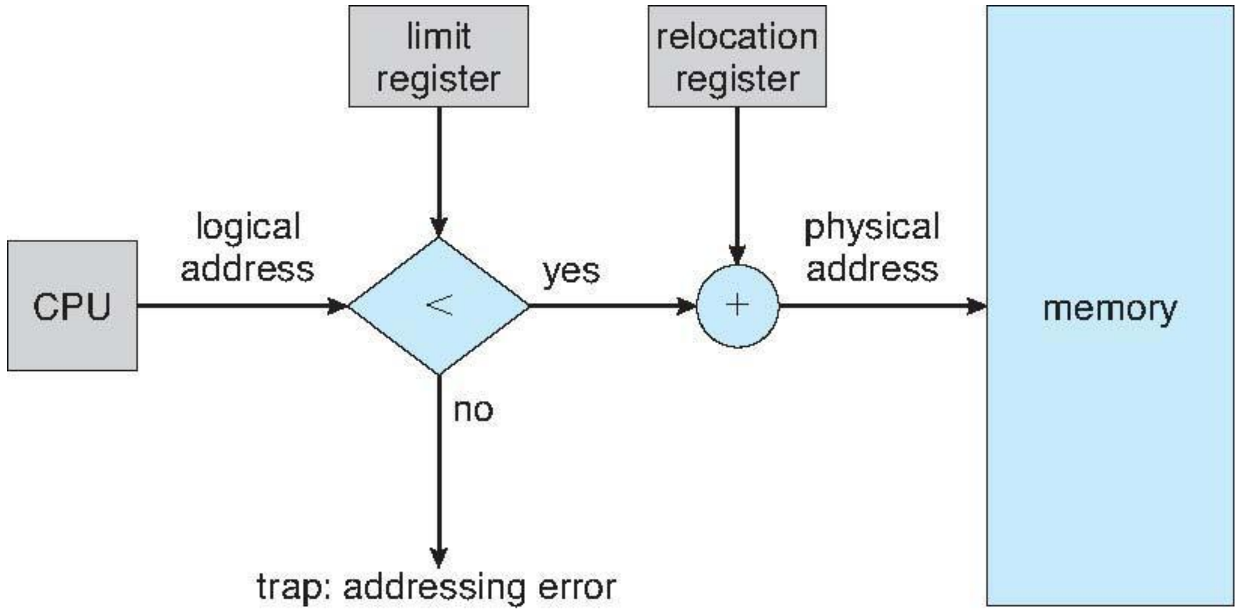
Contiguous Allocation

- Memory allocated in contiguous partitions for processes
  - ❑ Relocation and limit registers used to determine the partition at runtime
  - ❑ Relocation register contains value of smallest physical address; limit register contains range of logical addresses - each logical address must be less than the limit register.
  - ❑ Operating system has its own partition

Relocation Register



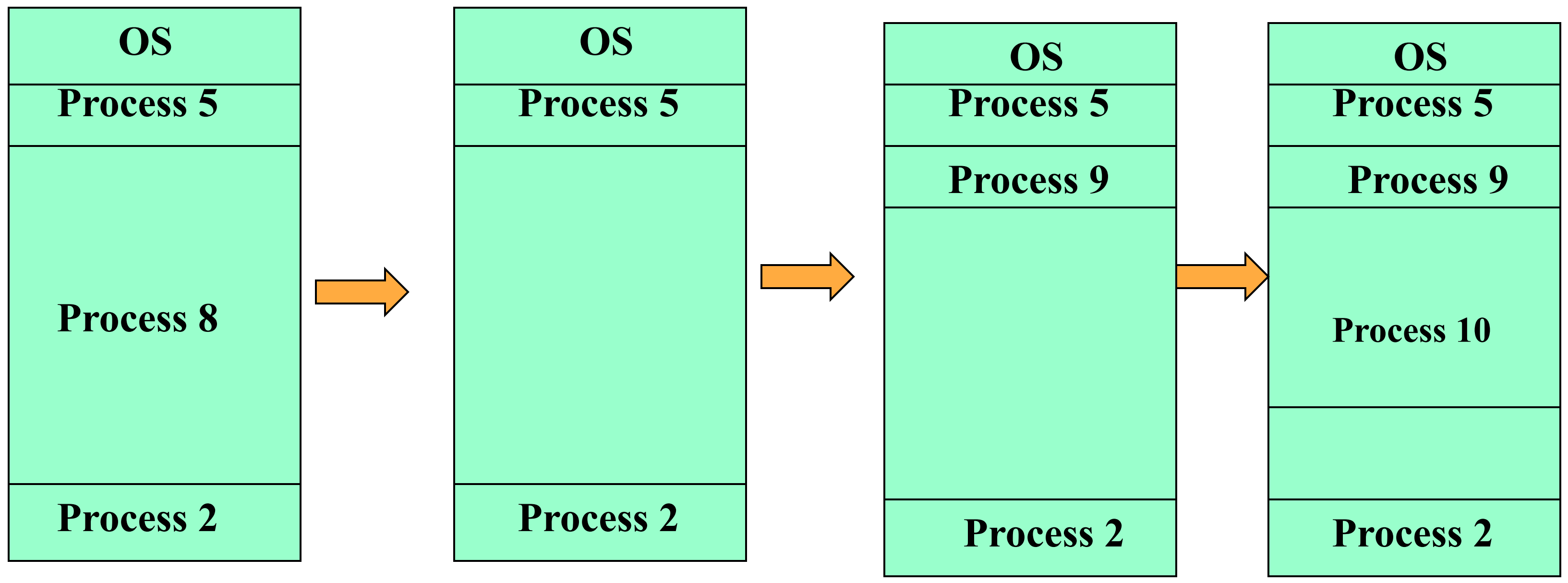
Relocation and Limit Registers



Contiguous Allocation (cont.)

- Multiple partition Allocation
  - Hole - block of available memory; holes of various sizes are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about
    - allocated partitions
    - free partitions (holes)

Contiguous allocation example



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, unless ordered by size. Produces the largest leftover hole.

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, unless ordered by size. Produces the largest leftover hole.
- ??? is the best in terms of execution speed.

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, unless ordered by size. Produces the largest leftover hole.
- First-fit is the best in terms of execution speed.
- ???? are better than ???? in terms of storage utilization.

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, unless ordered by size. Produces the largest leftover hole.
- First-fit is the best in terms of execution speed.
- First-fit and best-fit are better than worst-fit in terms of storage utilization.

Fragmentation

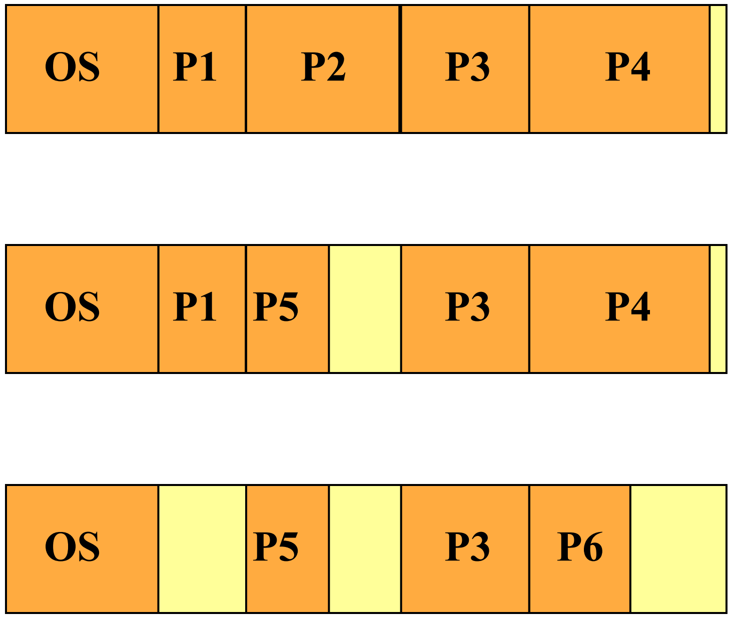
- External fragmentation
  - total memory space exists to satisfy a request, but it is not contiguous.
- Internal fragmentation
  - allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

Fragmentation

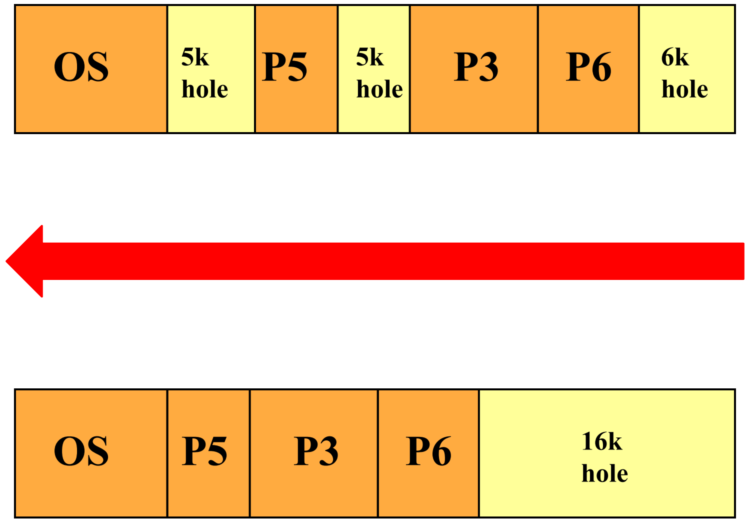
Contiguous allocation suffers mainly from external fragmentation (but also from internal fragmentation)

- We can reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block

Fragmentation example



Compaction



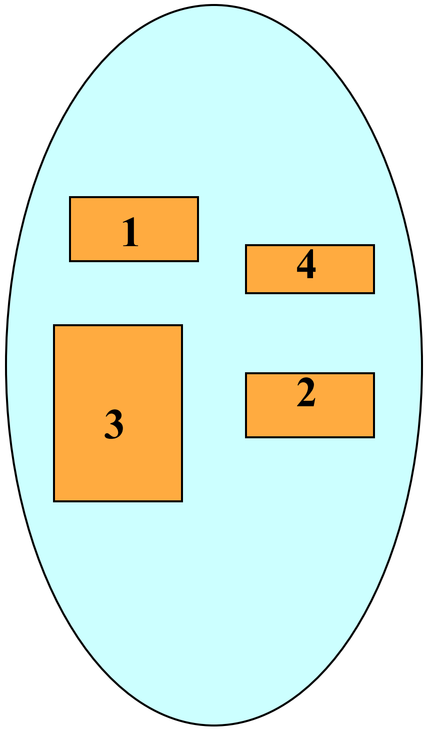
Compaction

- Compaction might cause problems for I/O due to inflight DMA
  - Solutions
    - (1) Pin process memory used by I/O devices
    - (2) Do I/O only into kernel buffers (which should also be pinned). -> Results in an additional data copy!

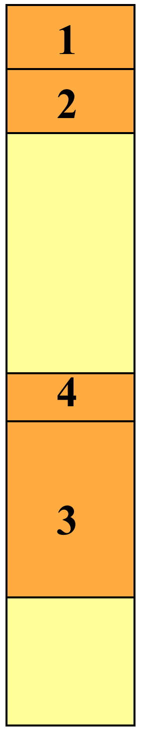
Segmentation

- A program is a collection of segments.
- A segment is a logical unit such as
  - Program code, stack, heap
- Protect each entity independently
- Allow each segment to grow independently
- Share each segment independently

Logical view of segmentation



Logical view of memory for a process



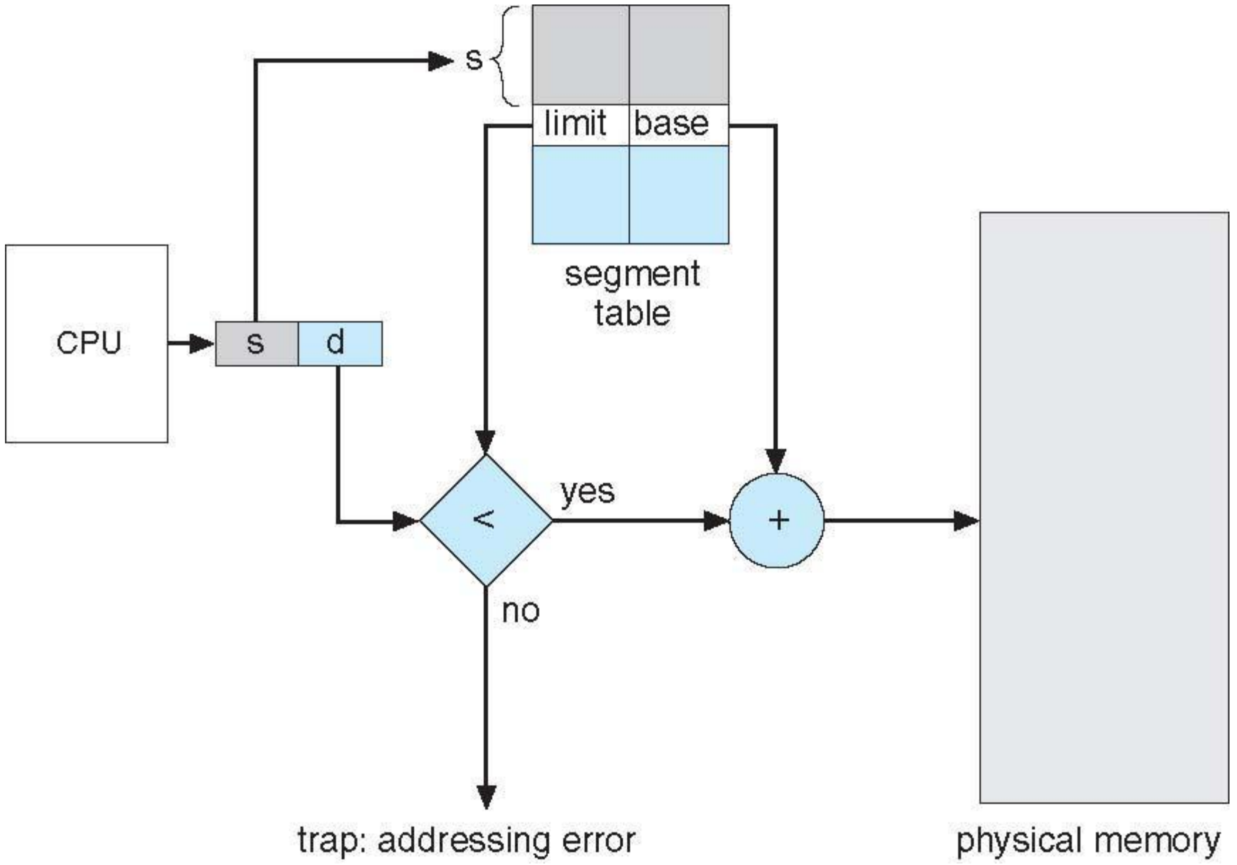
Physical Memory

Segmentation Architecture

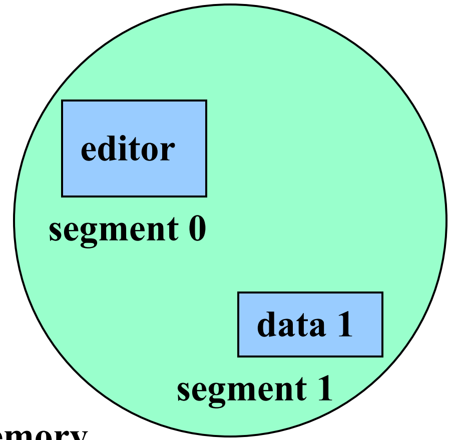
- Logical address consists of a two tuple

  <segment-number, offset>
- Segment Table
  - Maps two-dimensional user-defined addresses into one-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 segment table's location in memory.
  - *Segment-table length register* (STLR) indicates the number of segments used by a program; segment number is illegal if segment-number >= STLR (assuming segment-number starting at 0).

Segmentation hardware



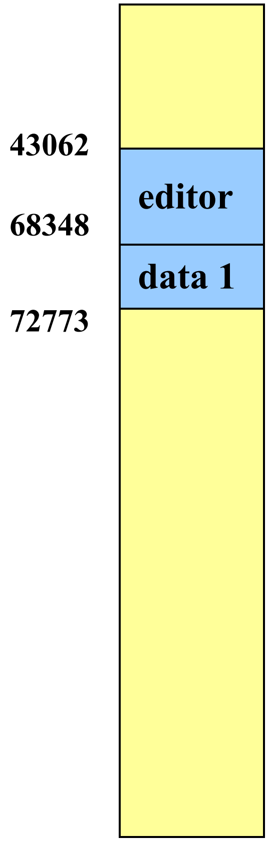
Segmentation example



Logical Memory
process P1

	Limit	Base
0	25286	43602
1	4425	68348

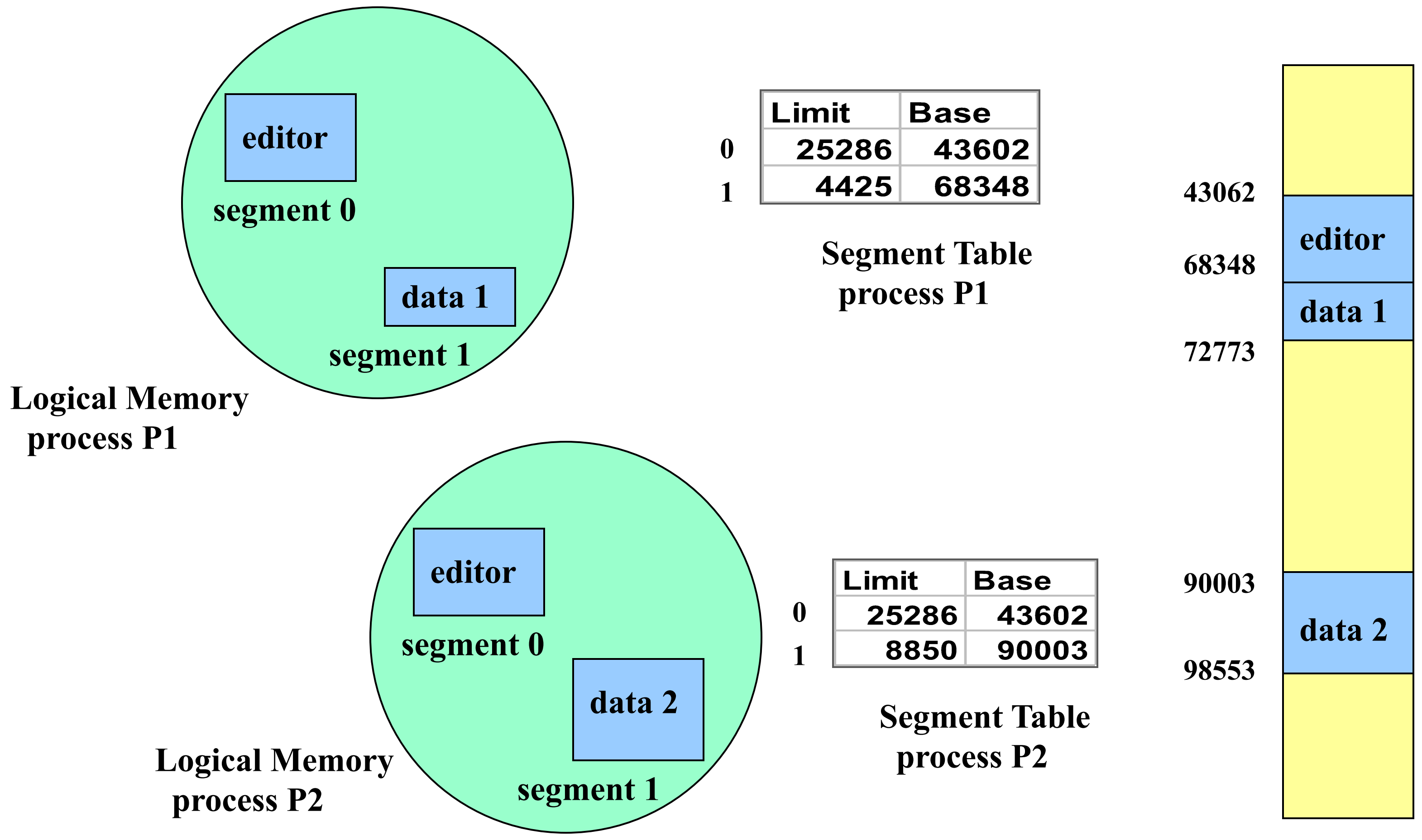
Segment Table
process P1



Segmentation Architecture (cont.)

- Sharing
  - Code sharing occurs at the segment level.
  - Shared segments must not necessarily have same segment number for different processes.

Shared segments



	Limit	Base
0	25286	43602
1	4425	68348

Segment Table
process P1

	Limit	Base
0	25286	43602
1	8850	90003

Segment Table
process P2

Segmentation Architecture (cont.)

- Allocation of segments - dynamic storage allocation problem
  - use best fit/first fit, may cause external fragmentation.
- Protection
  - protection bits associated with segments
    - read/write/execute privileges
    - Example use case: array in a separate segment - hardware can check for illegal array indexes.

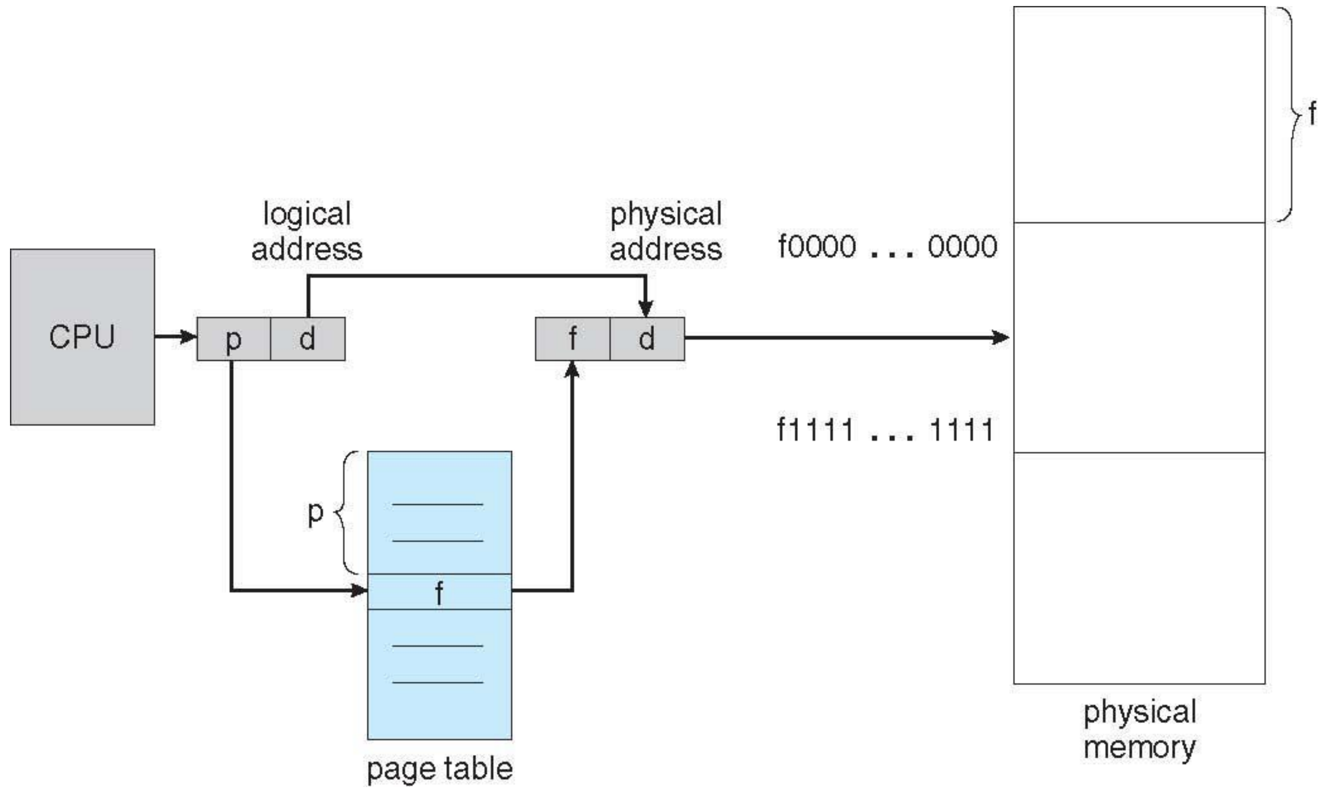
Paging

- A solution that solves external fragmentation
  - Divide physical memory into fixed size blocks called ***frames***
    - size is power of 2: 4 kbytes, 1 Mbytes, etc.
  - Divide logical memory into same size blocks called ***pages.***
    - Keep track of all free frames.
    - To run a program of size n pages, find n free frames and load program.
  - Set up a page table to translate logical to physical addresses.
  - Note:: Internal Fragmentation possible!!

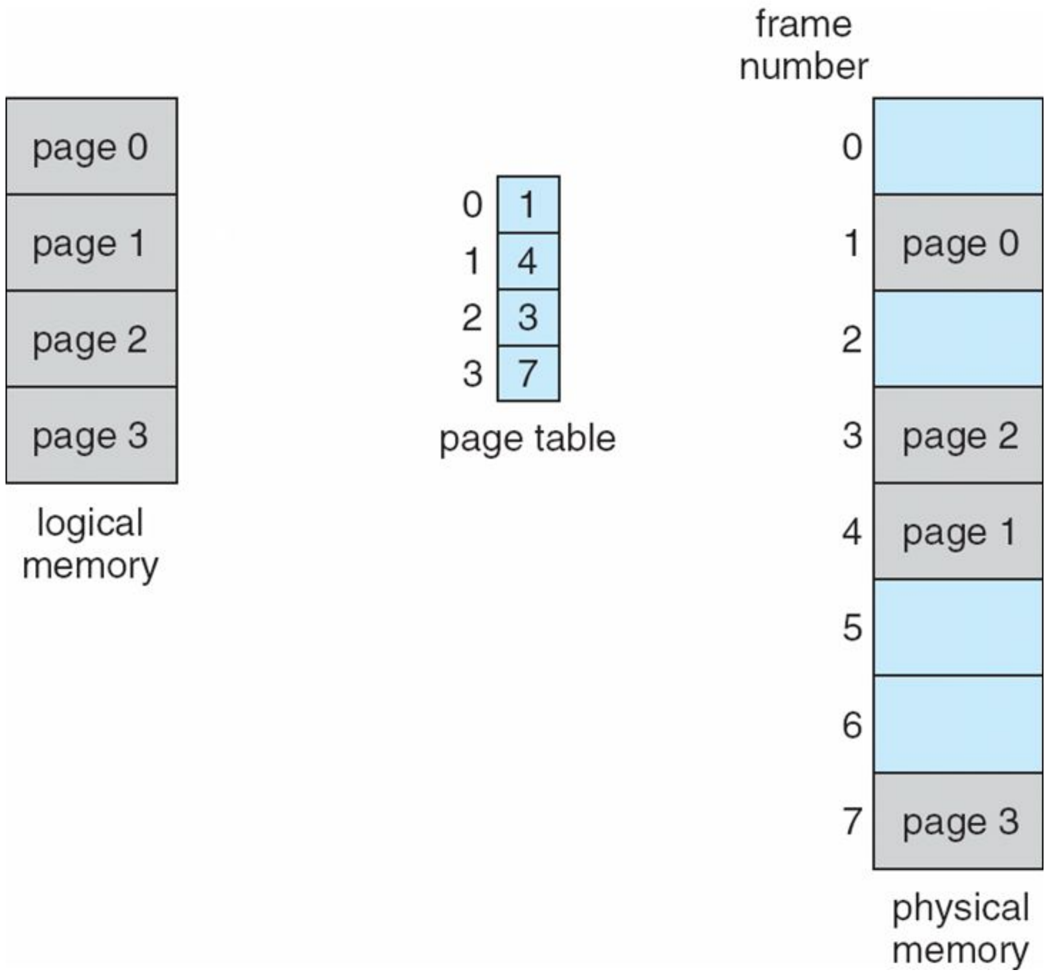
Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number(p)
    - used as an index into 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.

Address Translation Architecture



Example of Paging

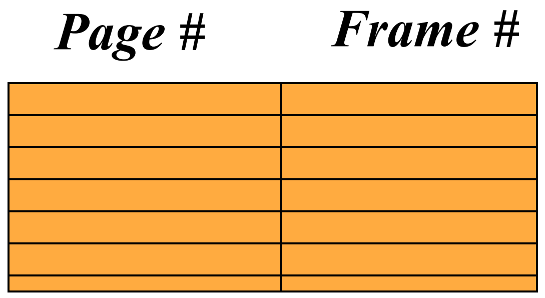


Page Table Implementation

- Page table is kept in main memory
    - Page-table base register (PTBR) points to the page table.

- Every data/instruction access requires 2 memory accesses.
    - One for page table, one for data/instruction
    - Two-memory access problem solved by use of special fast-lookup hardware cache (i.e. cache page table in registers)
        - Associative Registers or Translation Look-aside Buffers (TLBs)

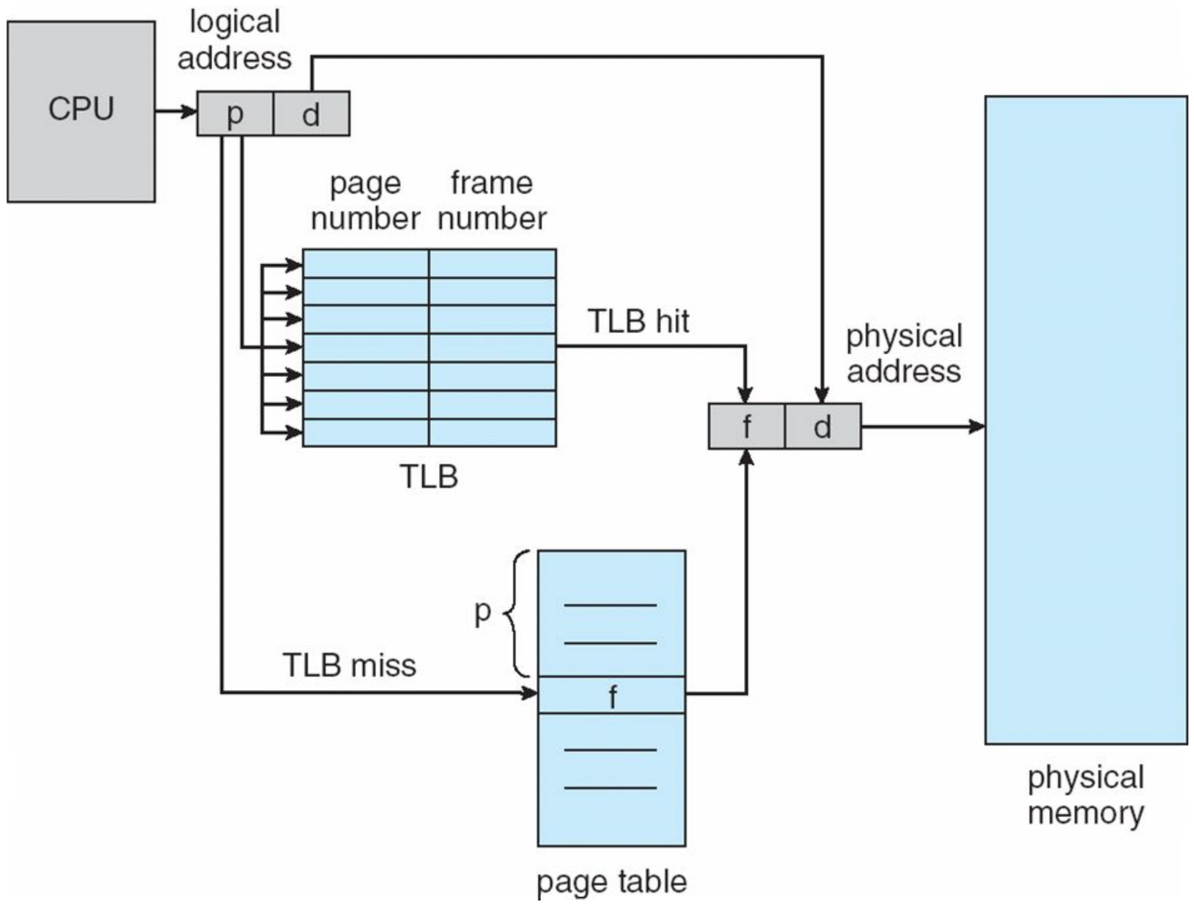
Translation Lookaside Buffer (TLB) (aka Associative Registers)



Address Translation
(A, A')

- If A is in TLB, get frame #
- Otherwise, need to go to page table for frame#
  - requires additional memory reference
- TLB Hit ratio - percentage of time page is found in TLB.

Paging hardware with TLB



Effective Access time

- TLB lookup time = ε time units
- Assume Memory access time = m time units
- TLB Hit ratio = α
- Effective access time (EAT) with TLB
  - EAT = (m + ε) α + ((2 * m) + ε) (1-α)
- Effective access time (EAT) without TLB
  - EAT = 2 * m

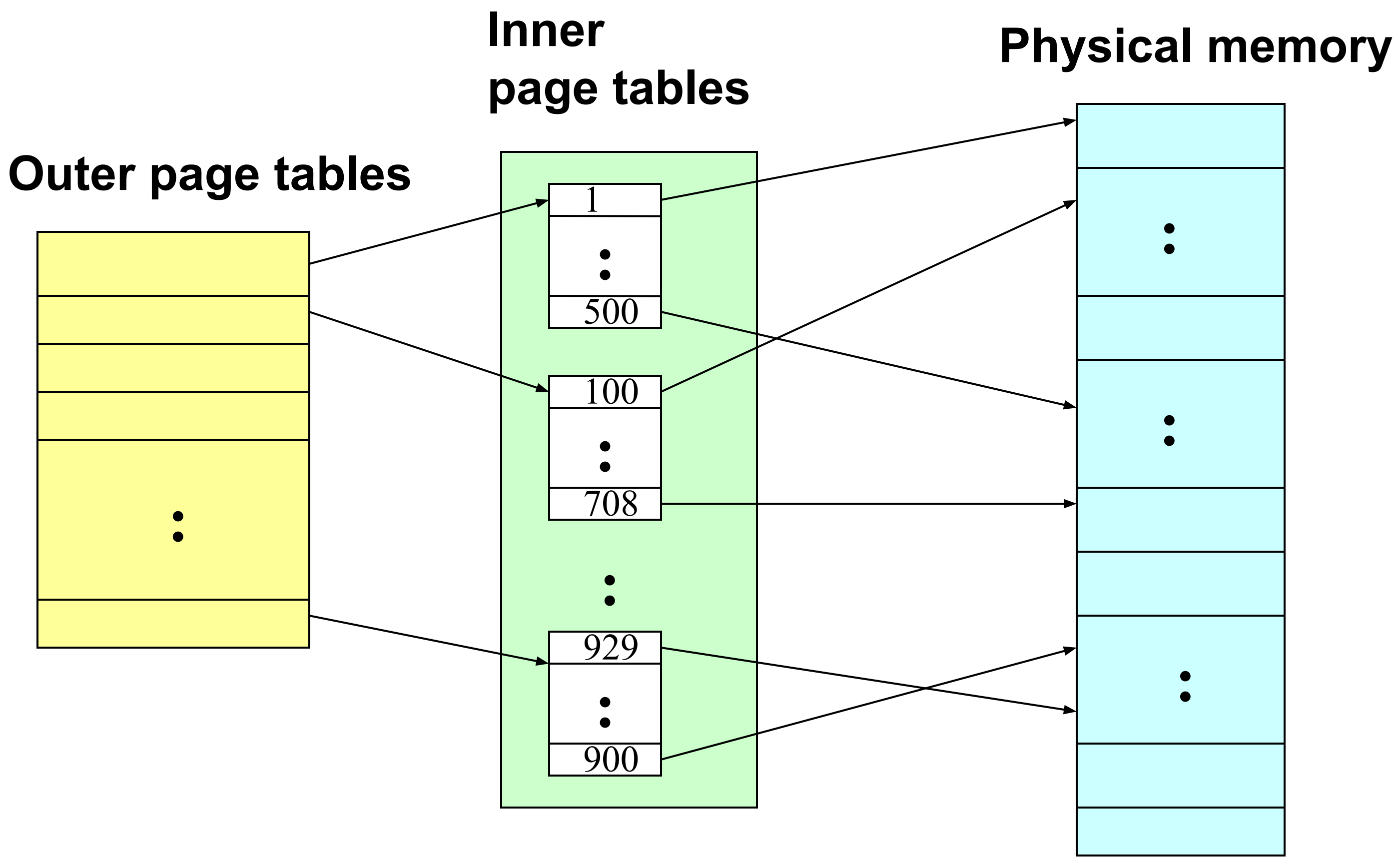
Memory Protection

- Implemented by associating protection bits with each page.
- Valid/invalid bit, read/write bit, and execute bit attached to each entry in page table.
  - Valid/invalid bit: indicates that the associated page is (bit = 1) or is not (bit = 0) in the process' logical address space.
  - Read: indicates that the page can (bit = 1) or cannot (bit = 0) be read
  - Write: indicates that the page can (bit = 1) or cannot (bit = 0) be written to
  - Execute: indicates that page content can (bit = 1) or cannot (bit = 0) be executed

Copy-on-Write using page tables

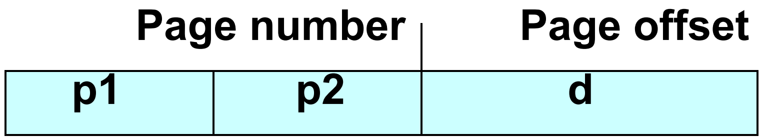
- Both parent and child processes share the address space at first (i.e., when child is created), but only in read-only mode.
- If any of them tries to write to some part of the address space, they’ll get a separate writable instance of that part of the address space, e.g., a separate writable page.

Two Level Page Table Scheme

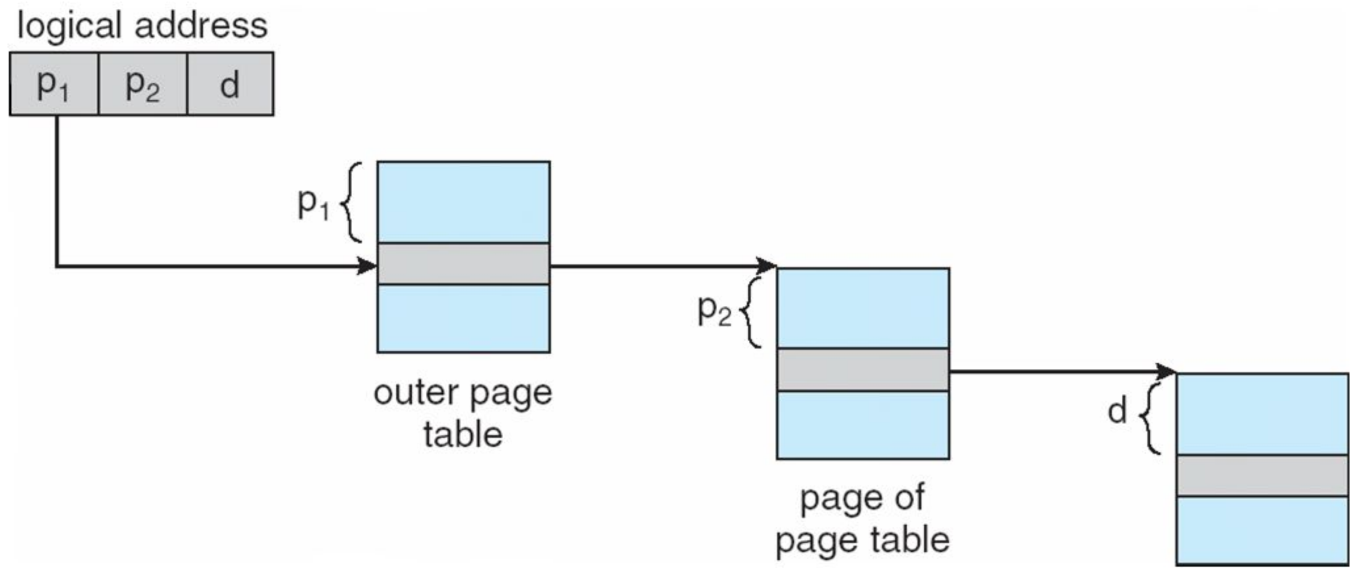


Two Level Paging Example

- A logical address (32bit machine, 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 consists of
  - a 10-bit page number, a 10-bit page offset
- Thus, a logical address is organized as (p1,p2,d) where
  - p1 is an index into the outer page table
  - p2 is the displacement within the page of the outer page table



Two Level Paging Example



Multilevel paging

- Each level is a separate table in memory
    - converting a logical address to a physical one may take multiple memory accesses.
  - TLB can help keep performance reasonable.
    - Assume a four-level page table
    - Assume TLB hit rate is 98%, memory access time is 100 nanoseconds, TLB lookup time is 20 nanoseconds
    - Effective Access time with TLB =

Multilevel paging

- Each level is a separate table in memory
    - converting a logical address to a physical one may take multiple memory accesses.
  - TLB can help keep performance reasonable.
    - Assume a four-level page table
    - Assume TLB hit rate is 98%, memory access time is 100 nanoseconds, TLB lookup time is 20 nanoseconds
    - Effective Access time with TLB = 0.98 * 120 + .02 * 520 = 128 ns
      - This is only a 28% slowdown in memory access time.
    - Effective Access time without TLB =

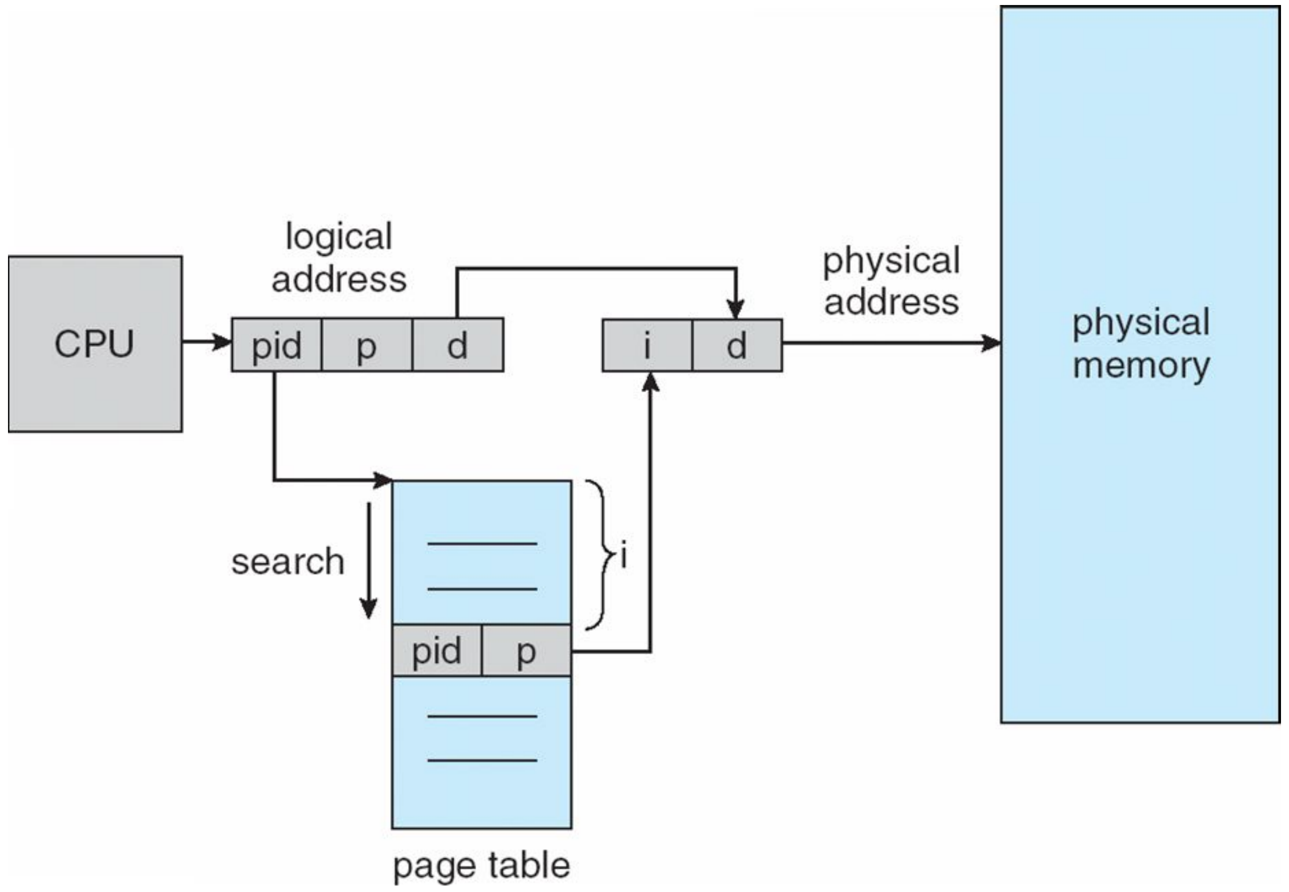
Multilevel paging

- Each level is a separate table in memory
    - converting a logical address to a physical one may take multiple memory accesses.
  - TLB can help keep performance reasonable.
    - Assume a four-level page table
    - Assume TLB hit rate is 98%, memory access time is 100 nanoseconds, TLB lookup time is 20 nanoseconds
    - Effective Access time with TLB = 0.98 * 120 + .02 * 520 = 128 ns
      - This is only a 28% slowdown in memory access time.
    - Effective Access time without TLB = 500 ns
      - This is a 400% slowdown in memory access time incurred just for paging.

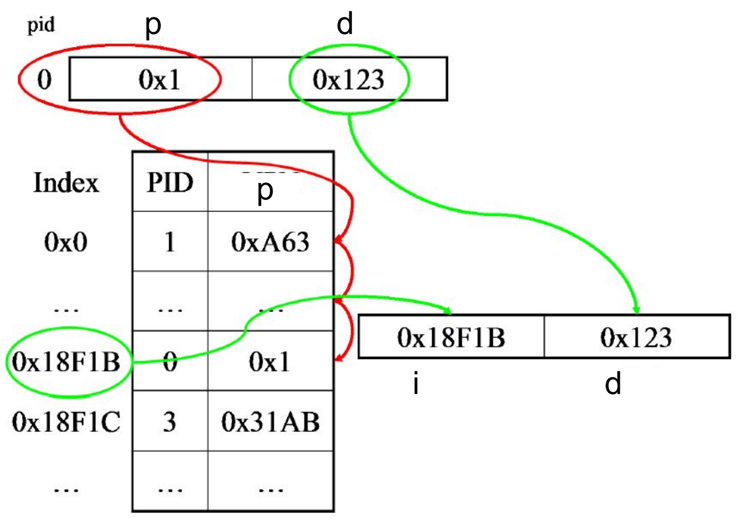
Inverted Page Table

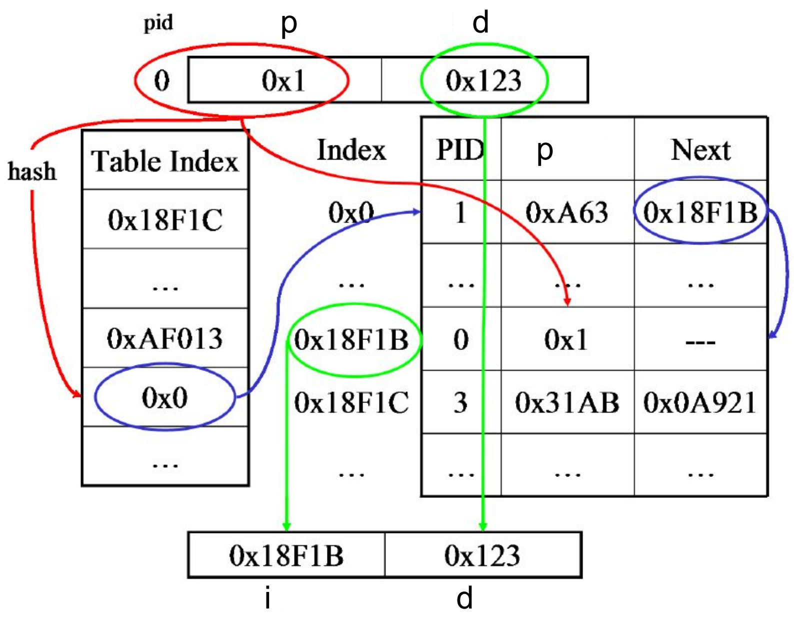
- One entry for each frame of memory
    - Entry consists of virtual address of frame in physical memory with information about process that owns page.
- ❑ Decreases memory needed to store page tables for all processes
- ❑ Increases time to search table when a page reference occurs
    - table sorted by physical address, lookup by virtual address
- ❑ Use hash table to limit search to one (maybe few) page-table entries.

Inverted Page Table



Inverted Page Table vs. Hashed Inverted Page Table





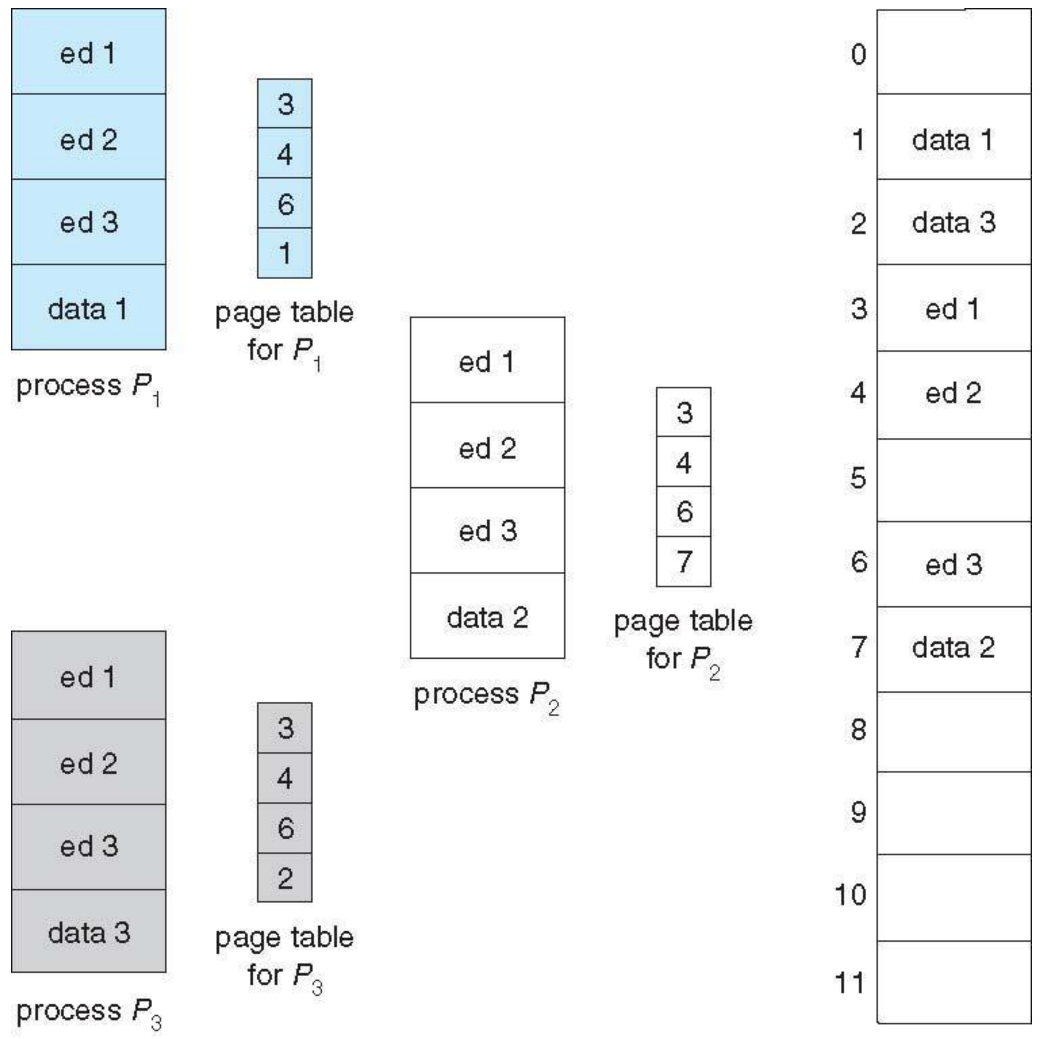
Figures adopted and modified from: CS162, UC Berkeley, Spring 2004, Discussion #10, by Amir Kamil, Topics: Inverted Page Tables, TLBs (https://web.eecs.umich.edu/~akamil/teaching/sp04/040104.pdf)

Shared pages

- Code and data can be shared among processes
  - Non-self-modifying code can be shared.
  - Pages for shared code and data can appear anywhere in logical address space.

- Shared code must not necessarily appear in the same location in the logical address space of all processes
- Private code and data
  - Each process keeps separate private code and data
  - Pages for private code and data can appear anywhere in the logical address space.

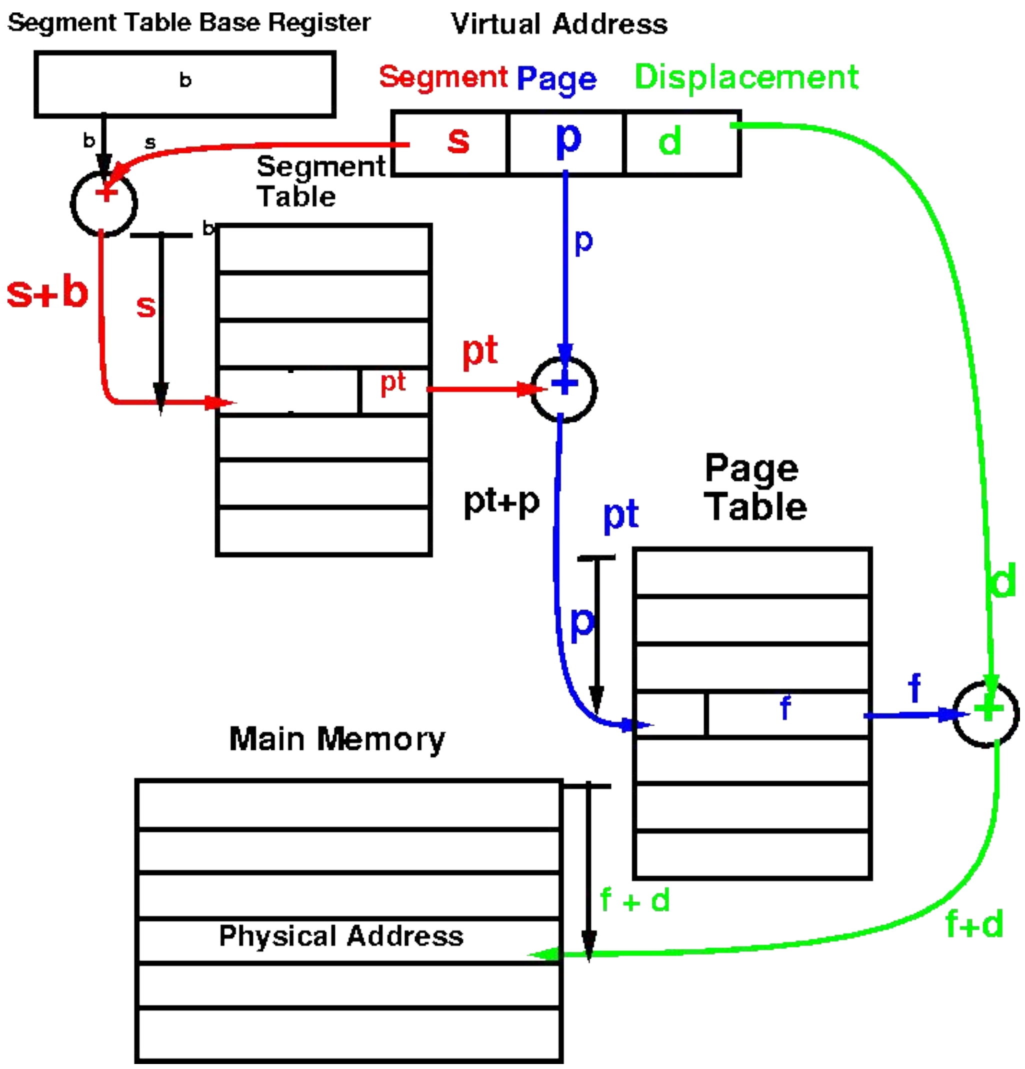
Shared Pages



Segmented Paged Memory

- Segment-table entry contains not the base address of the segment, but the base address of a page table for this segment.
  - Overcomes external fragmentation problem of segmented memory.
  - Paging also makes allocation simpler; time to search for a suitable segment (using best-fit etc.) reduced.
  - Enables the use of segments, e.g., for easier control of permissions of a memory region
  - Introduces some internal fragmentation and table space overhead.

Example: single-level page tables



Segment Table Base Register
b
Virtual Address
Segment
Page
Displacement
s
p
d
b
s
Segment Table
b
s+b
s
pt
pt
p
pt+p
pt
Page Table
p
d
f
f
Main Memory
Physical Address
f + d
f+d