Memory management

Dr. Naser Al Madi

Learning objective

- Finish page replacement algorithms
- Understand Thrashing
- Segmentation scheme
- Mass storage Systems
- Disk scheduling

Optimal Algorithm

Now you try:

0	2	1	2	3	2	3	0	1	0	3	1

Page faults = ?

Optimal Algorithm

Now you try:

0	2	1	2	3	2	3	0	1	0	3	1
0	0	1	1	3	3	3	3	1	1	1	1
	2	2	2	2	2	2	0	0	0	3	3
*	*	*		*			*	*		*	

Page faults = 7

FIFO Algorithm

Now you try:

0	2	1	2	3	2	3	0	1	0	3	1

Page faults = ?

FIFO Algorithm

Now you try:

0	2	1	2	3	2	3	0	1	0	3	1
0	0	1	1	1	2	2	2	1	1	1	1
	2	2	2	3	3	3	0	0	0	3	3
*	*	*		*	*		*	*		*	

Page faults = 8

LRU Algorithm

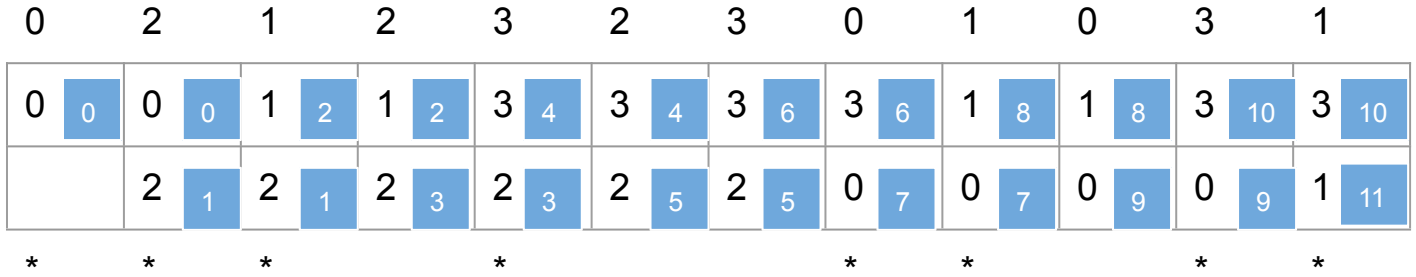
Now you try:

0	2	1	2	3	2	3	0	1	0	3	1

Page faults = ?

LRU Algorithm

Now you try:



Page faults = 8

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- Not common
- **Least Frequently Used (LFU)** Algorithm: replaces page with smallest count
- **Most Frequently Used (MFU)** Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Global vs. Local Allocation

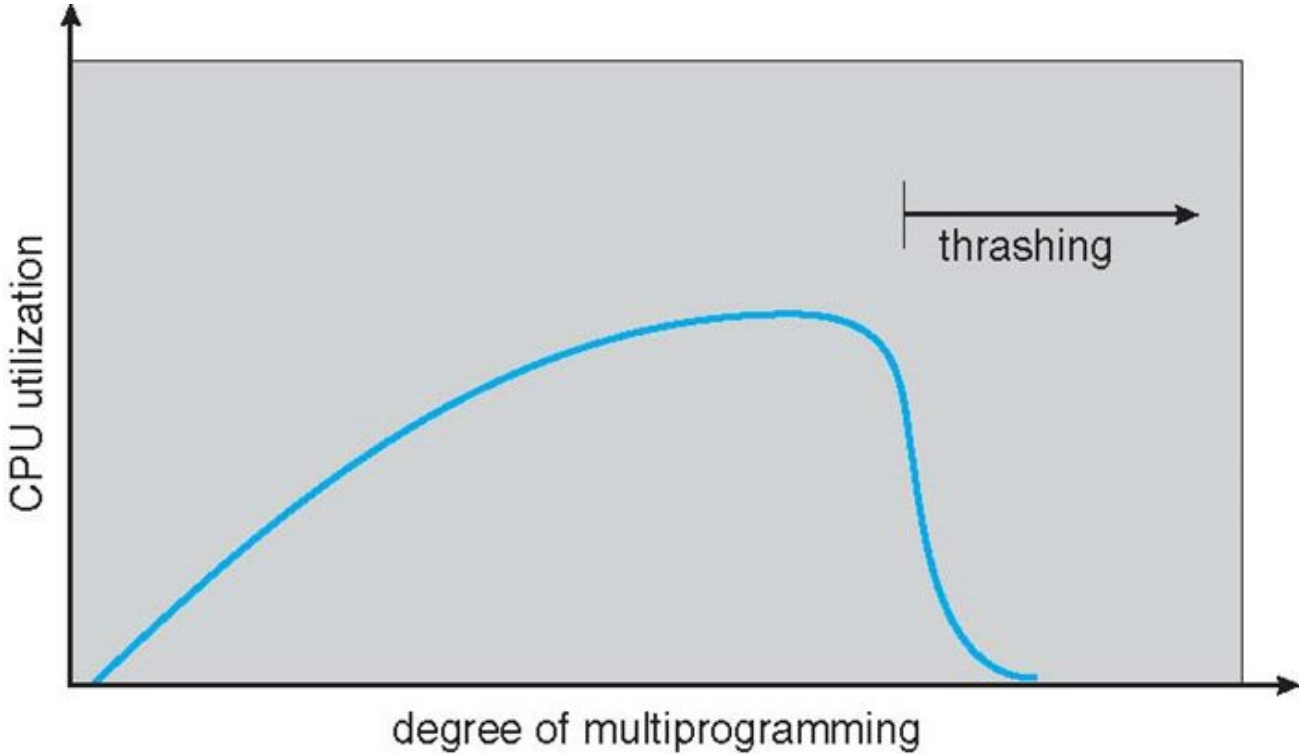
- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- But then process execution time can vary greatly
- But greater throughput so more common

- **Local replacement** – each process selects from only its own set of allocated frames
- More consistent per-process performance
- But possibly underutilized memory

Thrashing

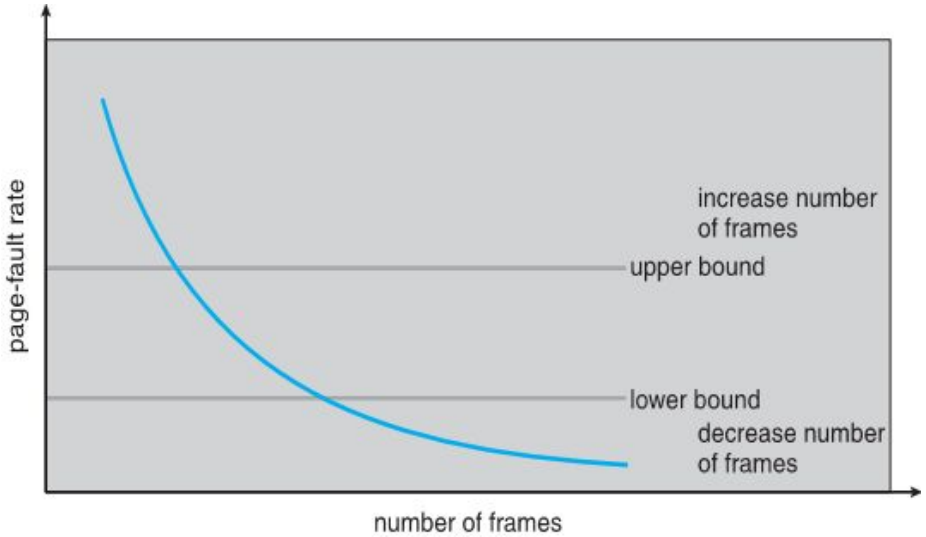
- If a process does not have “enough” pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
- This leads to:
  - Low CPU utilization
  - Operating system thinking that it needs to increase the degree of multiprogramming
  - Another process added to the system
- Thrashing = a process is busy swapping pages in and out

Thrashing



Page-Fault Frequency

- Establish “acceptable” page-fault frequency (PFF) rate and use local replacement policy
- If actual rate too low, process loses frame
- If actual rate too high, process gains frame

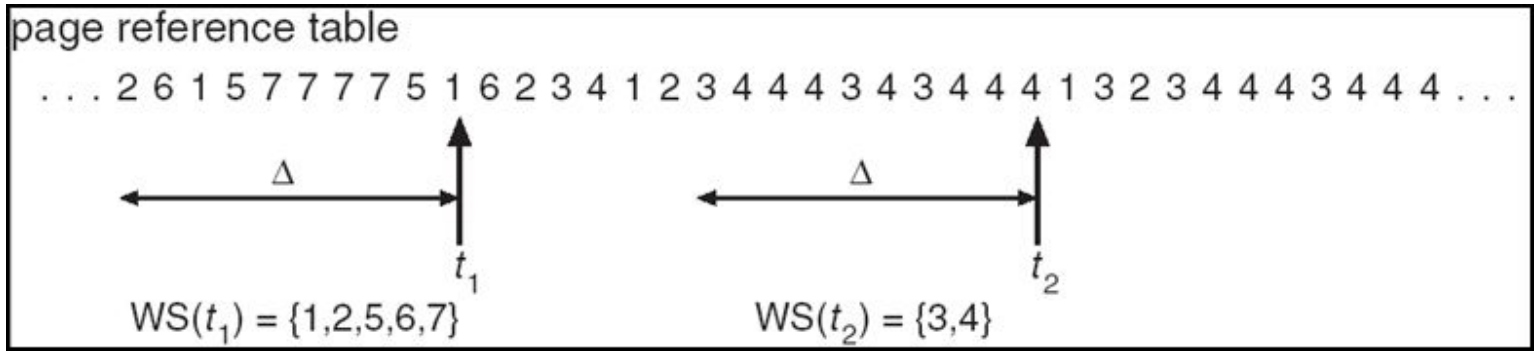


Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory were wasted

Working Set

- working set— the collection of pages that a process is working with, and which must be resident in main memory, to avoid thrashing.
- implementation: choose time T, pages that were accessed during time T constitute a working set, the rest can be discarded, scan periodically to update working set.
  - unix: T - about one second; scans - every several milliseconds
  - Example below: if T = 10, notice the different working sets WS(t_1) and WS(t_2)



Operating System Example: Windows

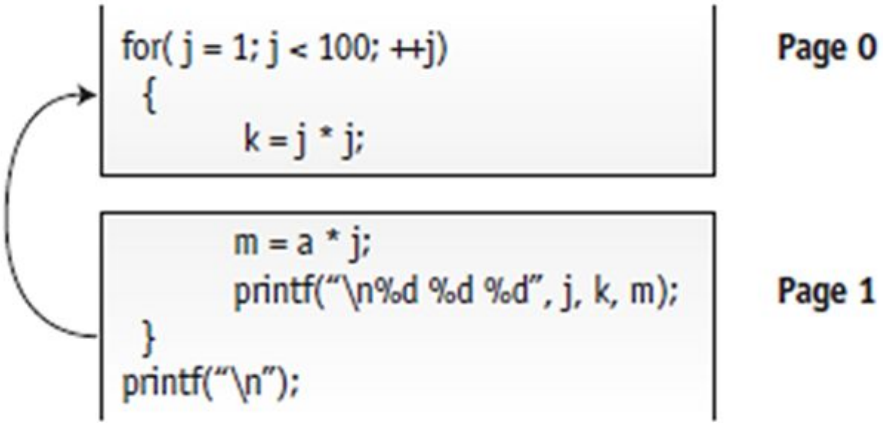
- Uses demand paging with clustering. Clustering brings in pages surrounding the faulting page
- **Working set** minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory
- Working set **trimming** removes pages from processes that have pages in excess of their working set minimum

Thrashing

- Excessive page swapping: inefficient operation
- Main memory pages: removed frequently; called back soon thereafter
- Occurs across jobs
  - Large number of jobs: limited free pages
- Occurs within a job
  - Loops crossing page boundaries

Thrashing

If only a single page frame is available, this program will have one page fault each time the loop of this C program is executed.



Segmentation

Segmentation

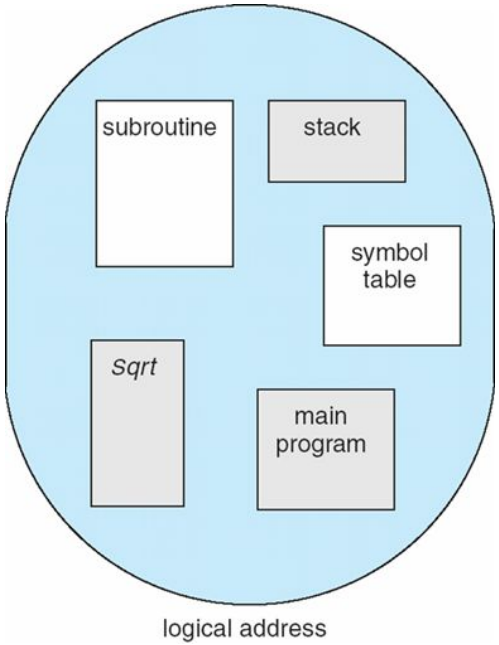
- Memory-management scheme that supports user view of memory
- A program is a collection of segments
- A segment is a logical unit such as:
  - Stack
  - Heap
  - Data
  - Text

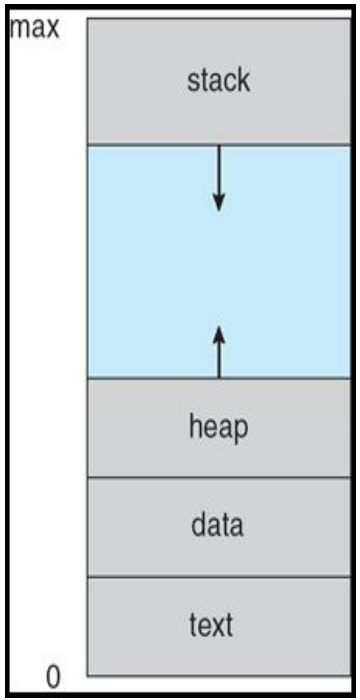
  Or
  - Functions
  - Procedures
  - Modules

Segmented Memory Allocation

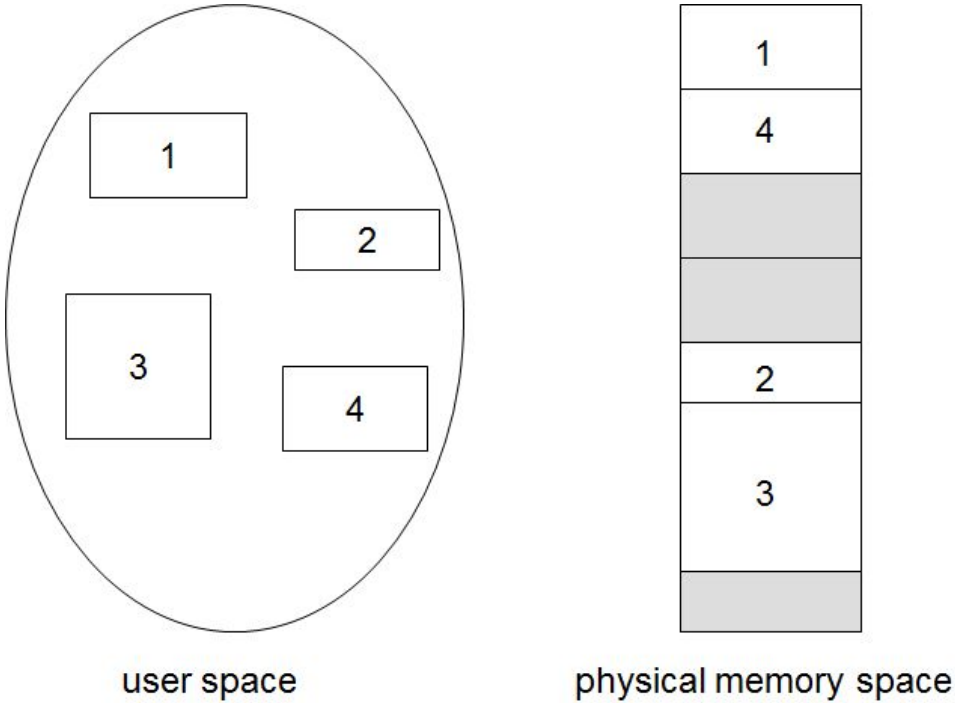
- Each job divided into several segments: different sizes
  - One segment for each module: related functions
- Reduces faults
  - Loops: not split over two or more “pages”
- Main memory: allocated dynamically
- Program’s structural modules: determine segments

User's View of a Program





Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a tuple:

  <segment-number, offset>

- 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

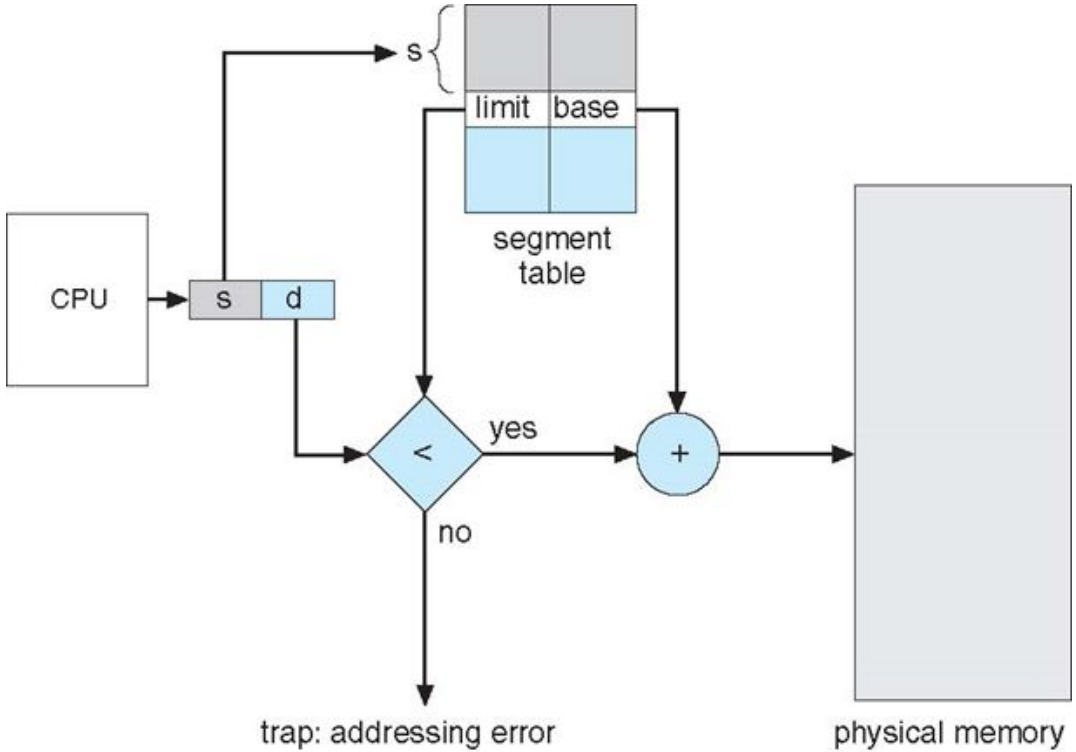
Segmentation Architecture

- Segment-table base register (STBR) points to the segment table's location in memory

- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number s is legal if s < STLR

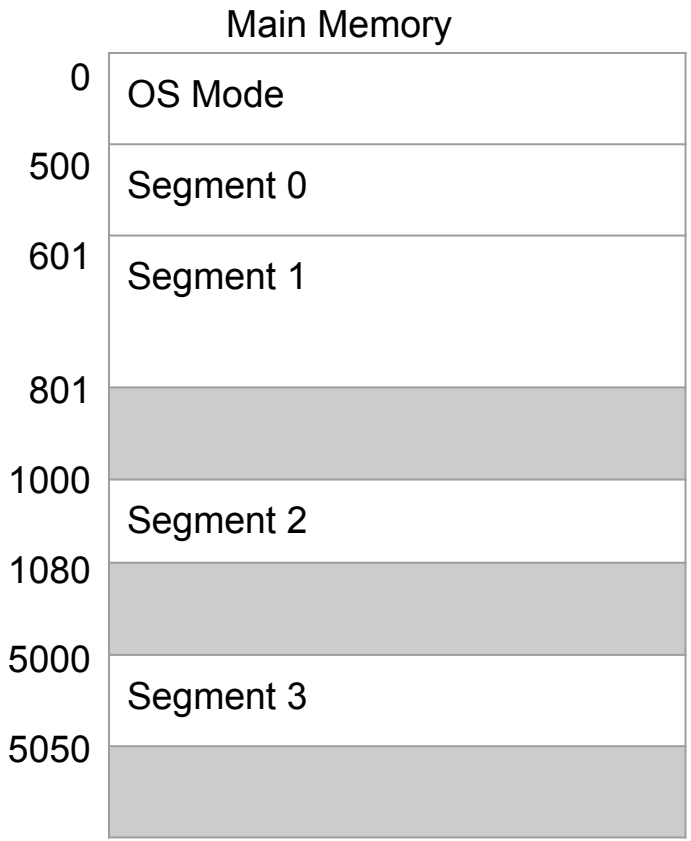
Segmentation Hardware



Segmentation Example

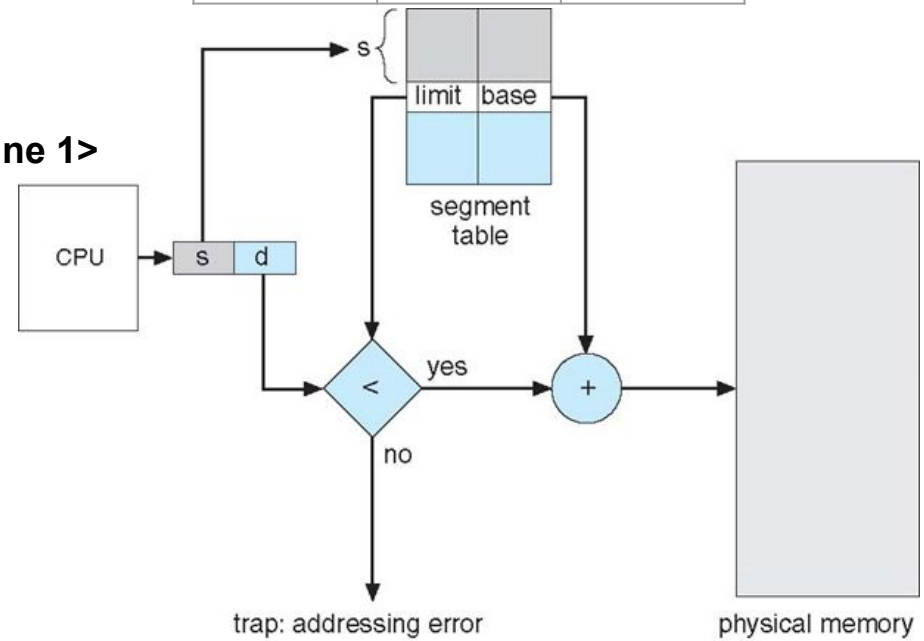
Segmentation table

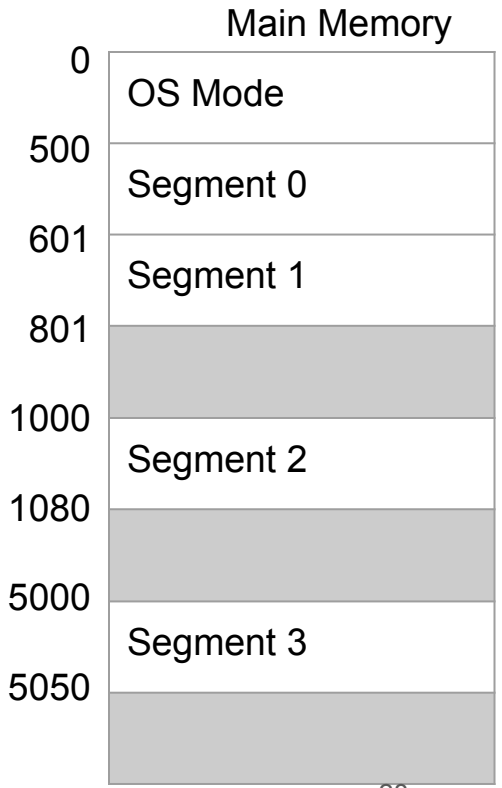
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000



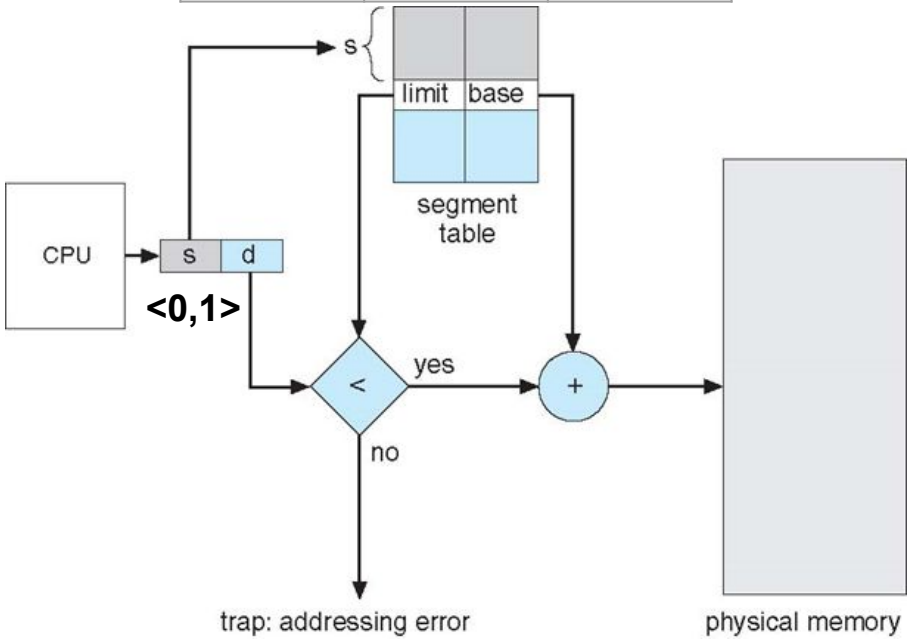
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000

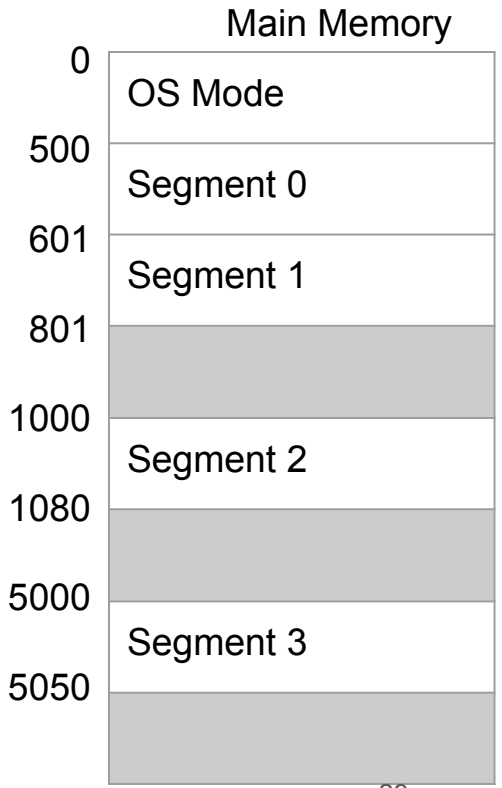
<segment 0, line 1>



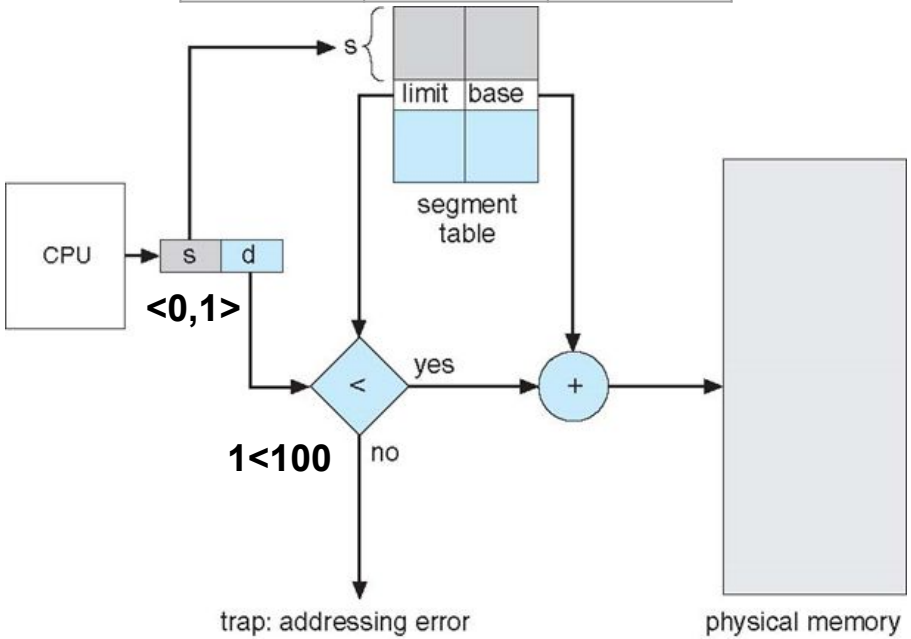


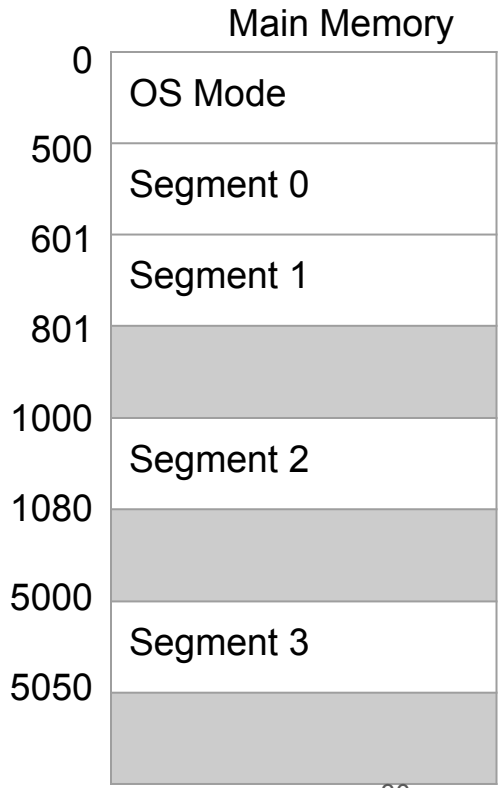
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000



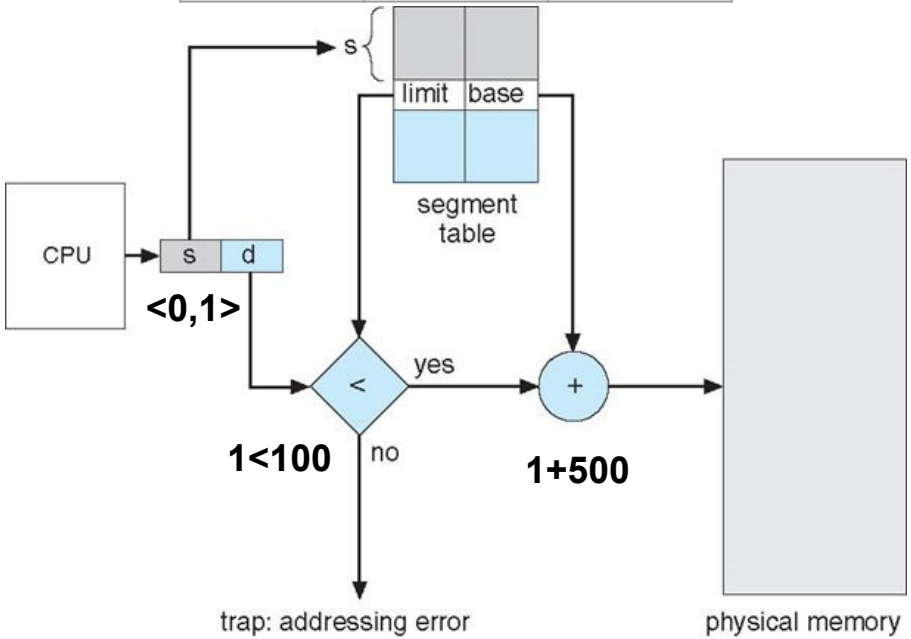


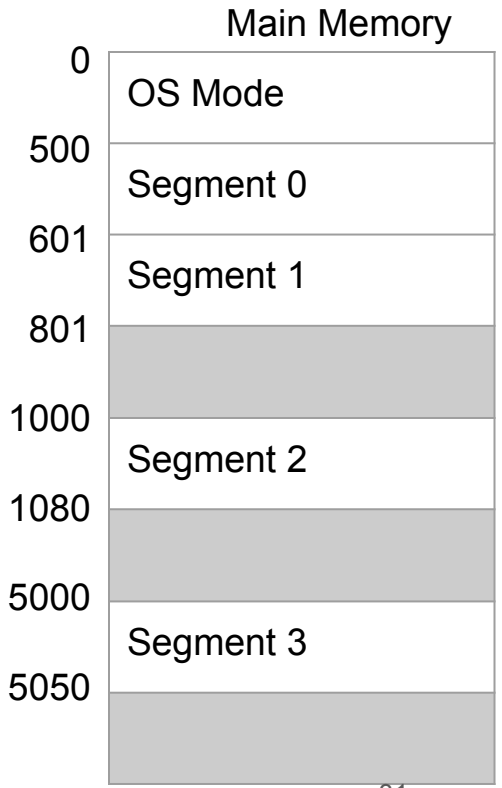
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000



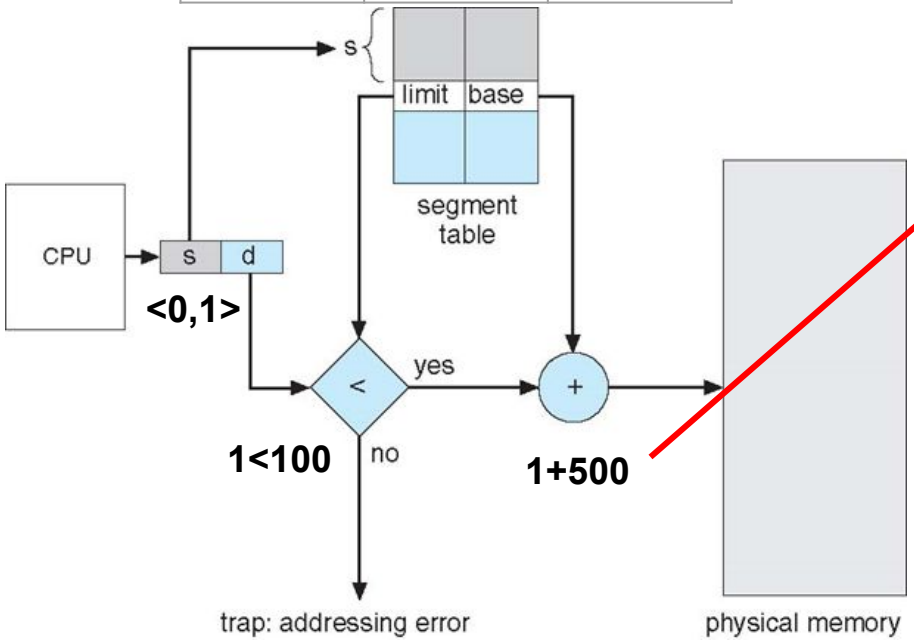


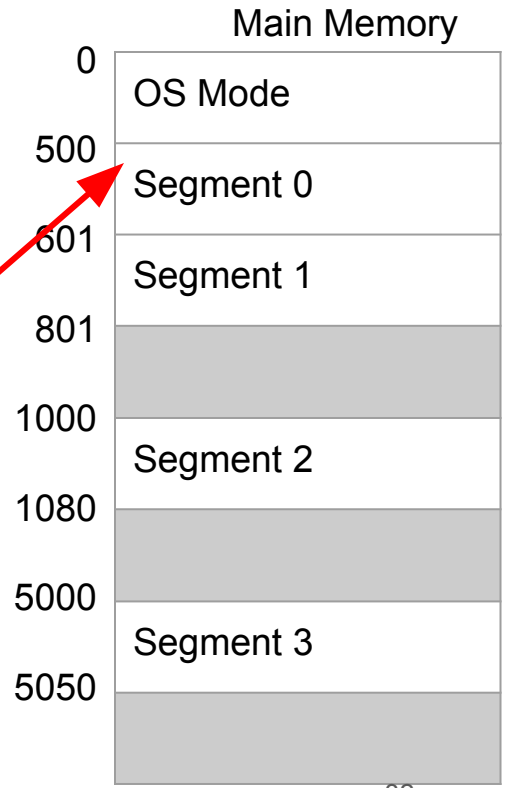
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000





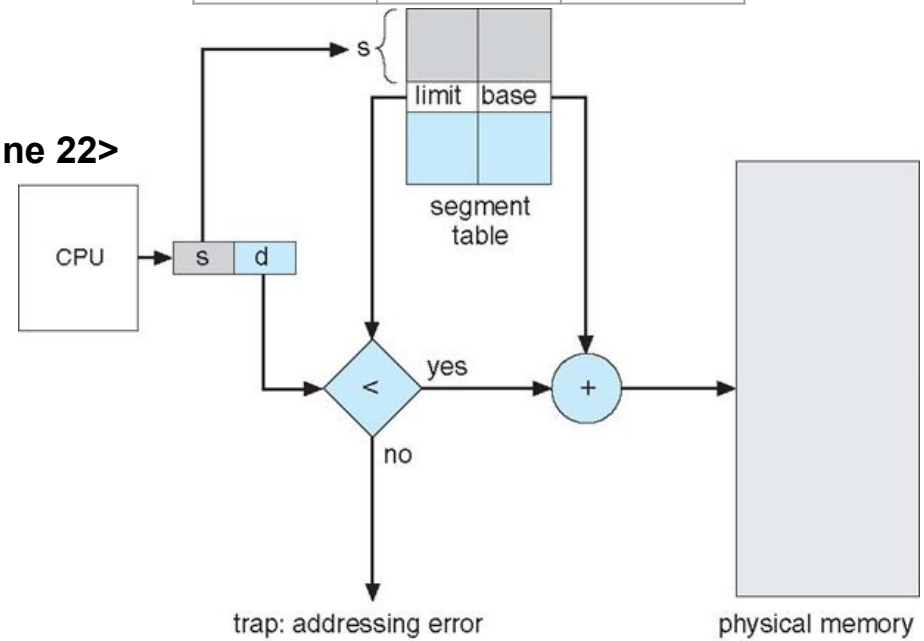
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000

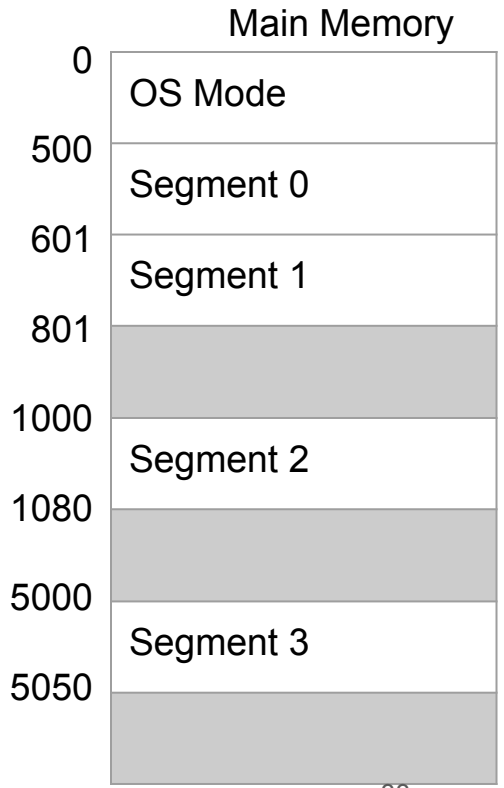




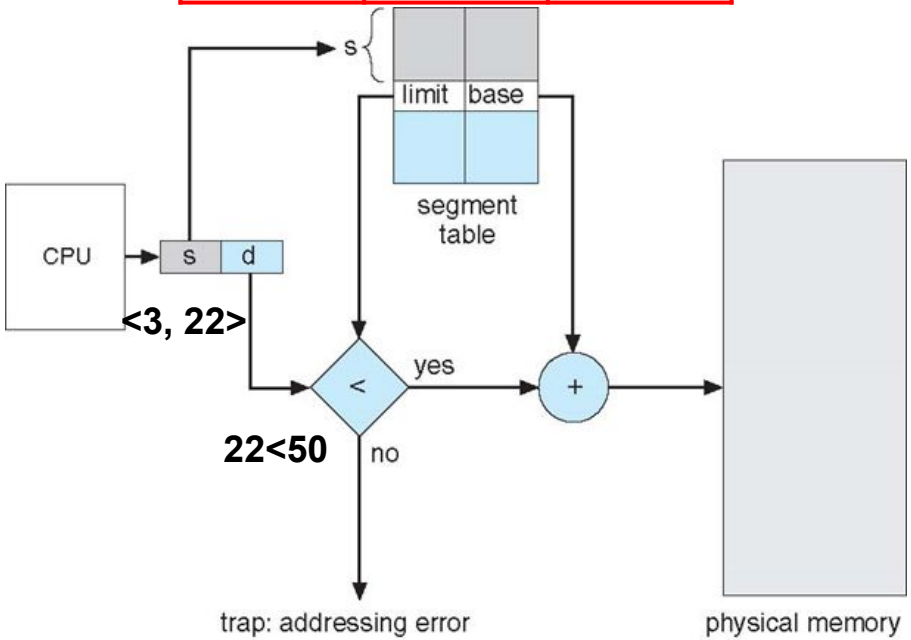
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000

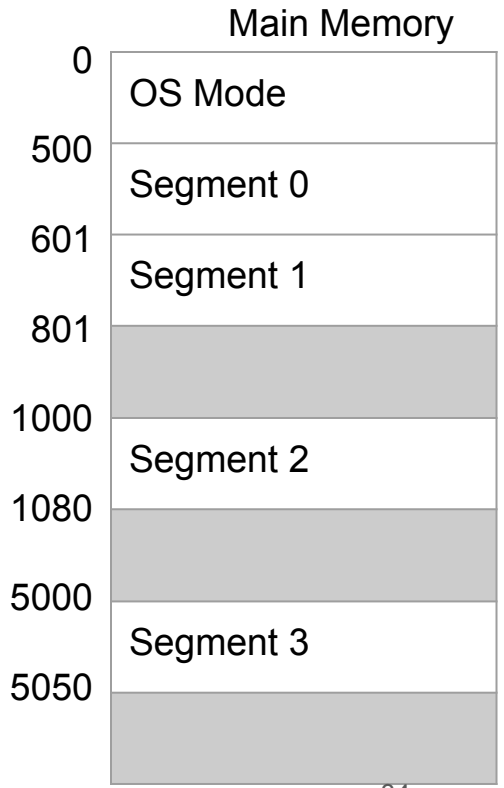
<segment 3, line 22>



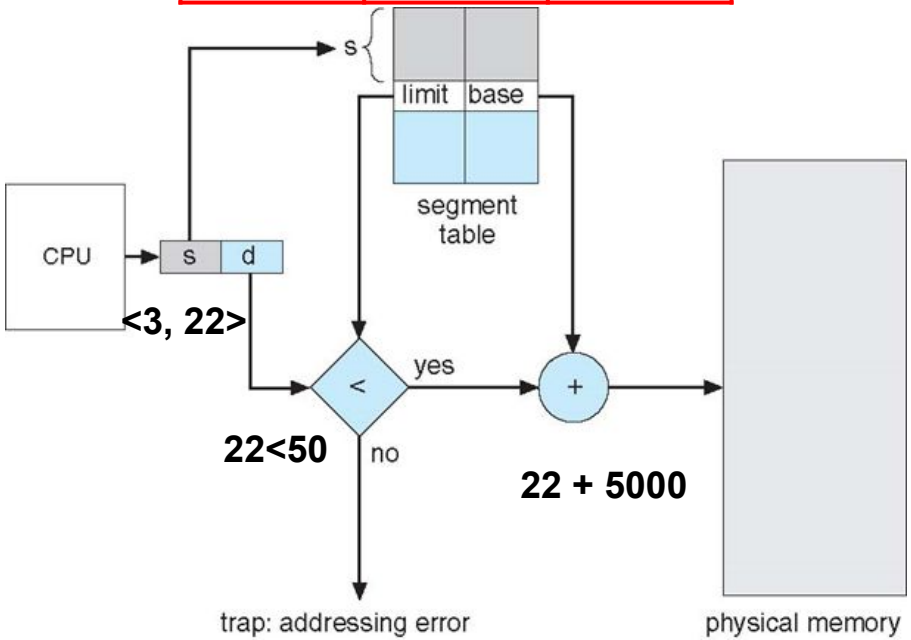


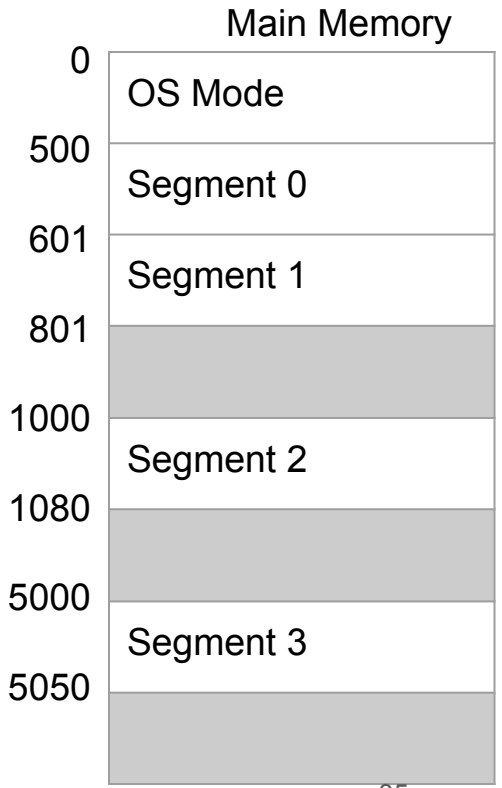
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000



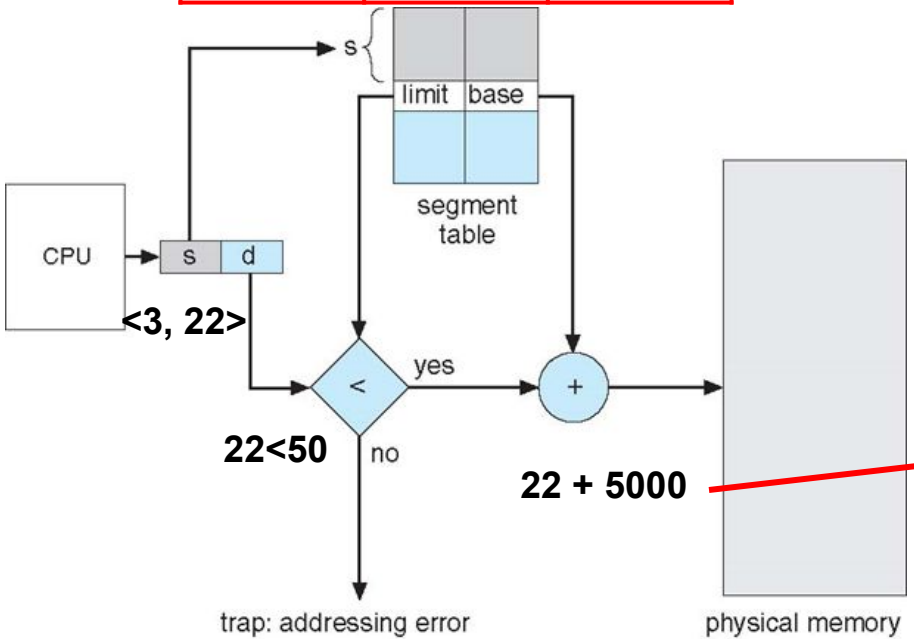


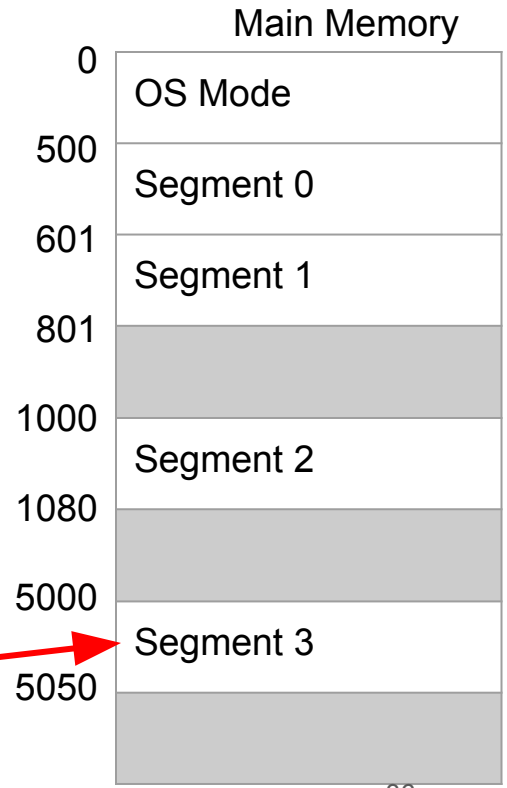
segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000





segment	limit	base
0	100	500
1	200	601
2	80	1000
3	50	5000





Segmented Memory Allocation

- Two-dimensional addressing scheme
  - Segment number and displacement
- Disadvantage
  - External fragmentation

- Major difference between paging and segmentation
  - Pages: physical units; invisible to the program
  - Segments: logical units; visible to the program; variable sizes

Segmented/Demand Paged Memory Allocation

- Subdivides segments: equal-sized pages
  - Smaller than most segments
  - More easily manipulated than whole segments
  - Segmentation’s logical benefits
  - Paging’s physical benefits
  - Working set = page cluster
- Segmentation problems removed
  - Compaction, external fragmentation
- Three-dimensional addressing scheme
  - Segment number, page number (within segment), and displacement (within page)

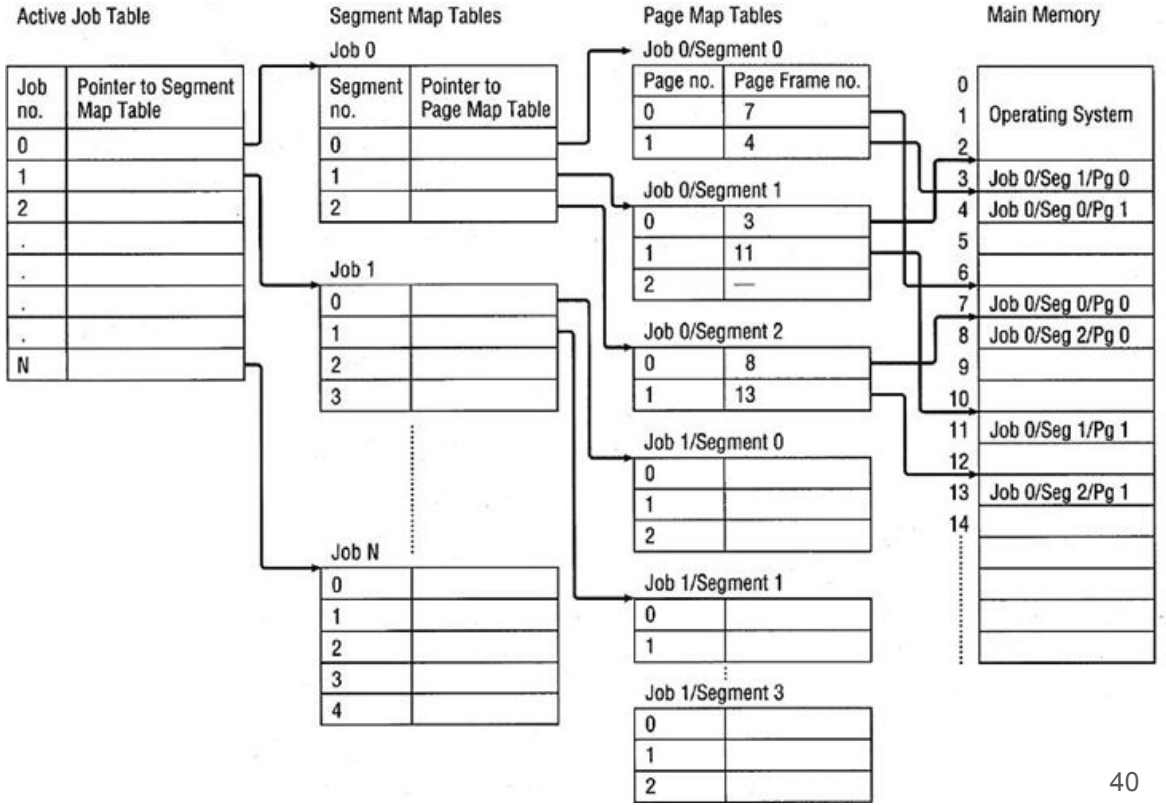
Segmented/Demand Paged Memory Allocation

Scheme requires four tables:

- Job Table: one for the whole system
  - Every job in process
- Segment Map Table: one for each job
  - Details about each segment
- Page Map Table: one for each segment
  - Details about every page
- Memory Map Table: one for the whole system
  - Monitors main memory allocation: page frames

Segmented/Demand Paged Memory Allocation

How the Job Table, Segment Map Table, Page Map Table, and main memory interact in a segment/paging scheme.



Segmented/Demand Paged Memory Allocation

- Disadvantages
  - Overhead: managing the tables
  - Time required: referencing tables
- Associative memory
  - Several registers allocated to each job
  - Segment and page numbers: associated with main memory
- Page request: initiates two simultaneous searches
  - Associative registers
  - SMT and PMT

Virtual Memory

- Made possible by swapping pages in/out of memory
- Program execution: only a portion of the program in memory at any given moment
- Requires cooperation between:
  - Memory Manager: tracks each page or segment
  - Processor hardware: issues the interrupt and resolves the virtual address

Virtual Memory

Virtual Memory with Paging	Virtual Memory with Segmentation
Allows internal fragmentation within page frames	Doesn't allow internal fragmentation
Doesn't allow external fragmentation	Allows external fragmentation
Programs are divided into equal-sized pages	Programs are divided into unequal-sized segments that contain logical groupings of code
The absolute address is calculates using the page number and displacement	The absolute address is calculated using the segment number and displacement
Requires Page Map Table (PMT)	Requires Segment Map Table (SMT)

Virtual Memory

Advantages:

- Job size: not restricted to size of main memory
- More efficient memory use
- Unlimited amount of multiprogramming possible
- Code and data sharing allowed
- Dynamic linking of program segments facilitated

Disadvantages:

- Higher processor hardware costs
- More overhead: handling paging interrupts
- Increased software complexity: prevent thrashing

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
    - Total physical memory space of processes can exceed physical memory

- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images

Swapping

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

- System maintains a ready queue of ready-to-run processes which have memory images on disk

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
    - Started if more than threshold amount of memory allocated
    - Disabled again once memory demand reduced below threshold

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 then be very high

Context Switch Time including Swapping

- Ex: 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)

Context Switch Time including Swapping

- Other constraints as well on swapping
  - Pending I/O – can’t swap out as I/O would occur to wrong process
- Or always transfer I/O to kernel space, then to I/O device
  - Known as double buffering, adds overhead
- standard swapping not used in modern operating systems
  - But modified version common
  - Swap only when free memory extremely low

Questions?