Virtual Memory I

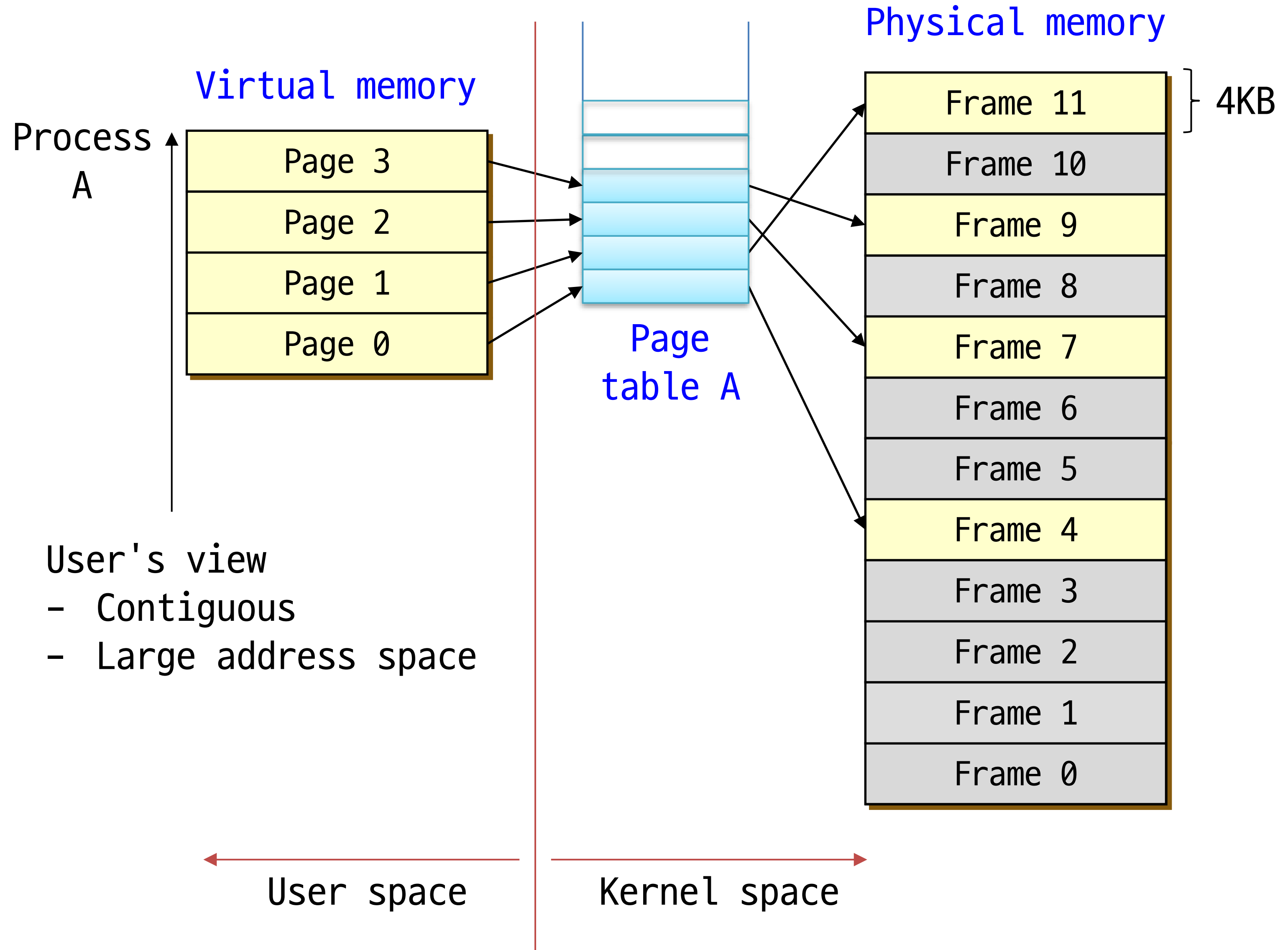
Jo, Heeseung

Today's Topics

Virtual memory implementation

- Paging
- Segmentation

Paging Introduction

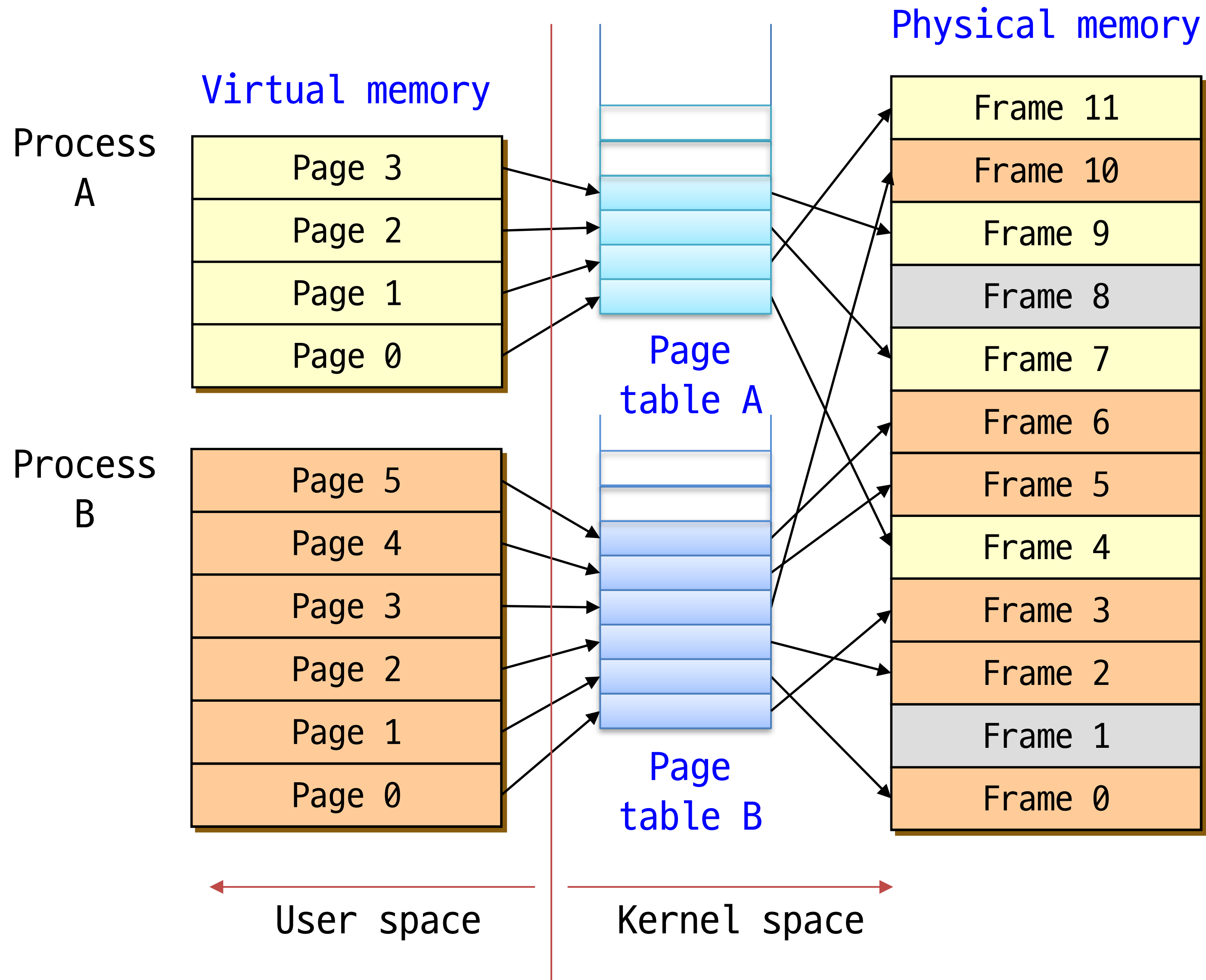


Paging (1)

Paging

- Permits the physical address space of a process to be noncontiguous
- Divide physical memory into fixed-sized blocks called frames
- Divide logical memory into blocks of same size called pages
  - Page (or frame) size is power of 2 (typically, 512B – 8KB)
  - Mostly use 4K in modern OS
- To run a program of size n pages, need to find n free frames and load program
- OS keeps track of all free frames
- Set up a page table to translate virtual to physical addresses

Paging (2)



Paging (3)

Natural extension of fixed partitions

- Fixed partitions: 1 partition / process
- Paging: many small partitions / process

User's perspective

- Users (and processes) view memory as one contiguous address space from 0 through N
  - Virtual address space (VAS)
- In reality, pages are scattered throughout the physical memory
  - Virtual-to-physical mapping
  - This mapping is invisible to the program
- Protection is provided because a program cannot reference memory outside of its VAS
  - The virtual address *0xdeadcafe* maps to different physical addresses for different processes

Virtual Memory (1)

Example

```
#include <stdio.h>

int n = 0;

int main ()
{
     printf ("&n = 0x%08x\n", &n);
}

% ./a.out
&n = 0x08049508
```

What happens if two users simultaneously run this application?

Paging (4)

Translating addresses

- A virtual address has two parts:

  <virtual page number (VPN)::offset>

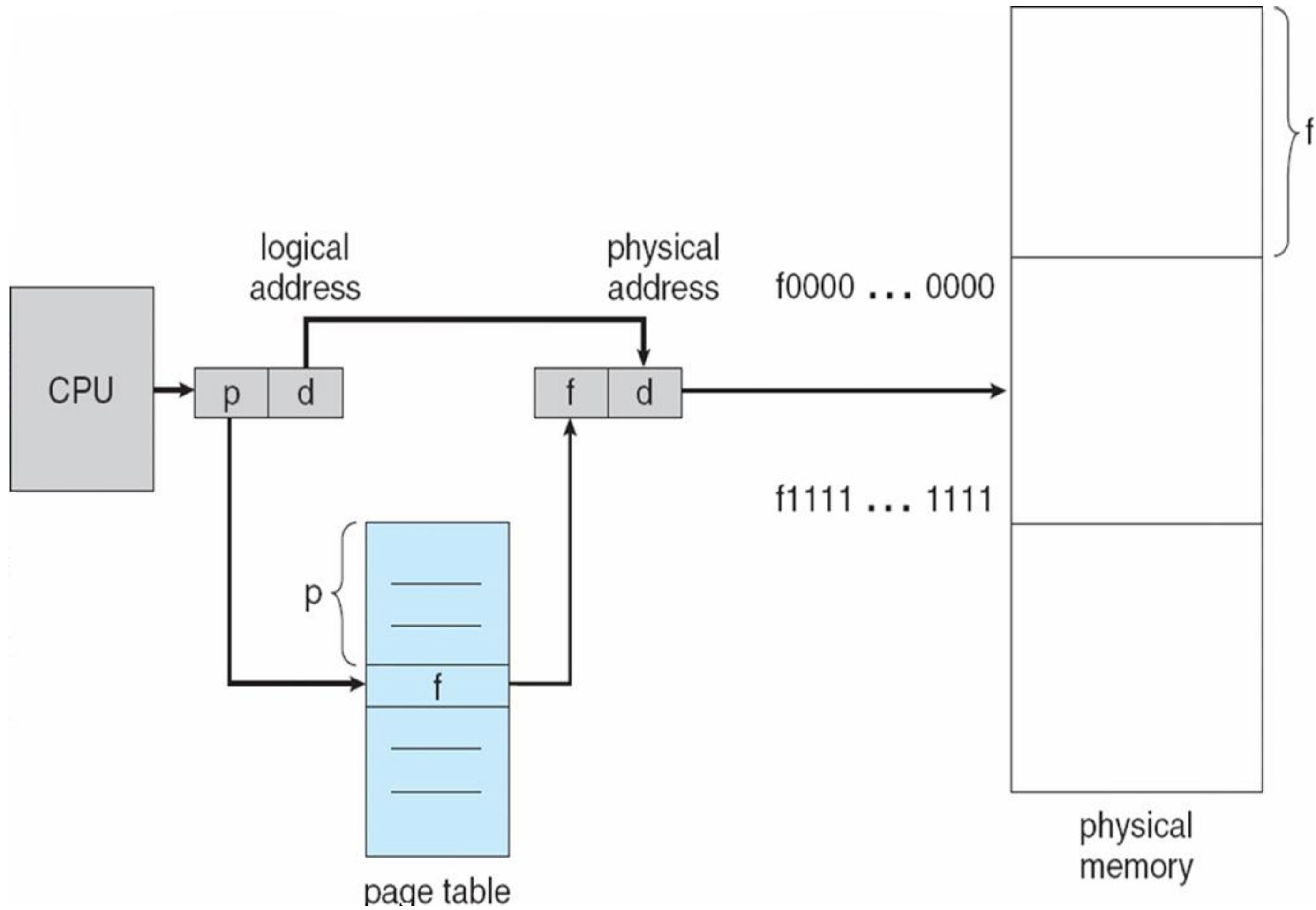
- VPN is an index in a page table
- Page table determines page frame number (PFN)
- Physical address is <PFN::offset>

Page tables

- Managed by OS
- Each process has its own page table
  - For 100 process, there are 100 page tables
- Map VPN to PFN
  - VPN is the index into the table that determines PFN
- One page table entry (PTE) per page in virtual address space
  - i.e. one PTE per VPN

Paging (5)

Address translation architecture



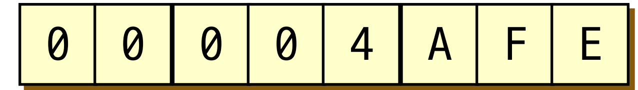
Paging (6)

Paging example

- Virtual address: 32 bits (4G)
- Physical address: 20 bits (1M)
- Page size: 4KB

- Offset: 12 bits
- VPN: 20 bits
- Page table entries: 2^{20}

Virtual address (32bits)



VPN

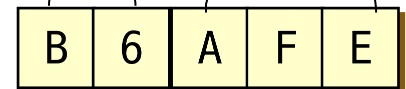
Offset

Page tables

2	7
D	0
3	1
F	E
B	6
A	4

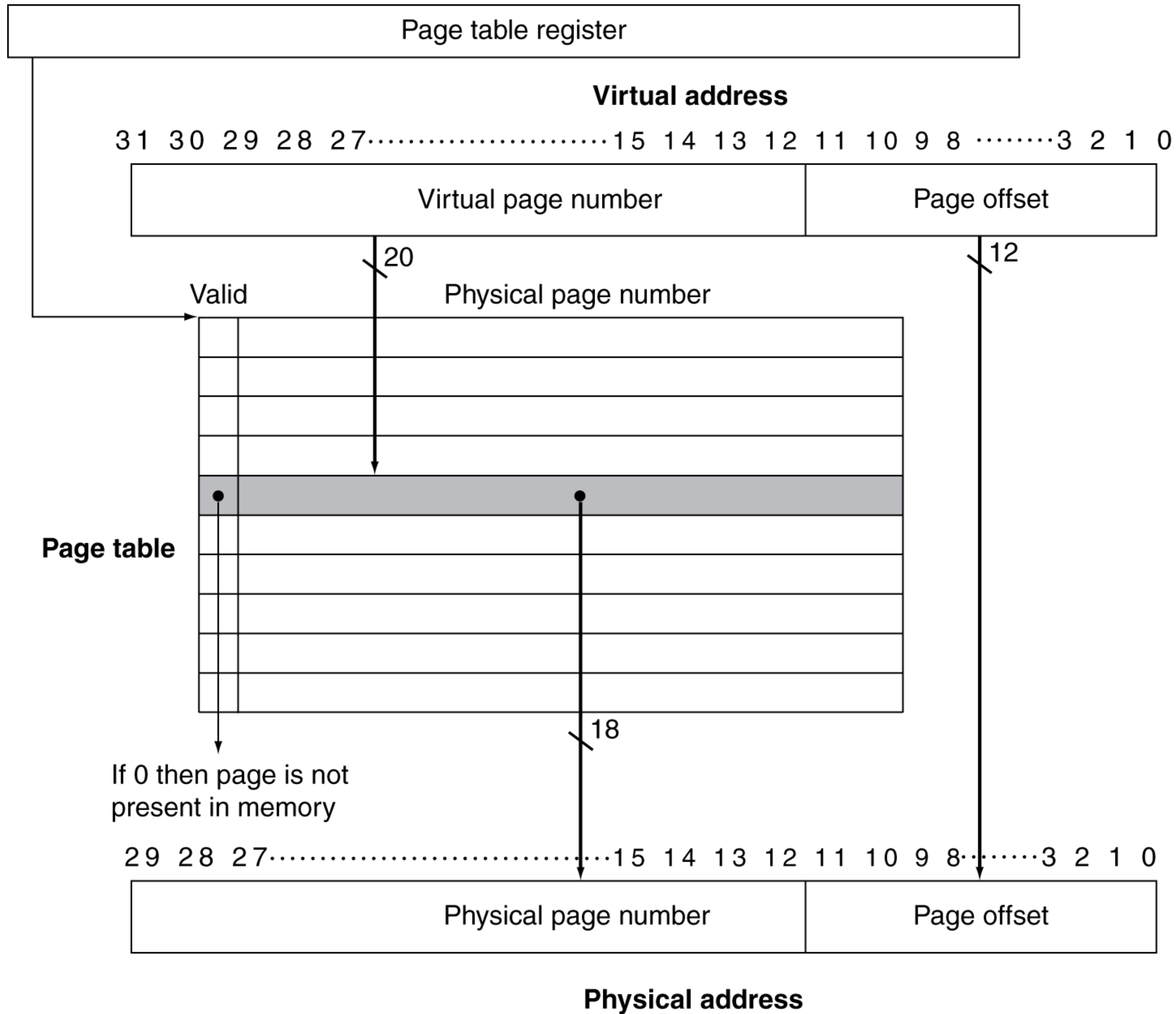
⋮

PFN



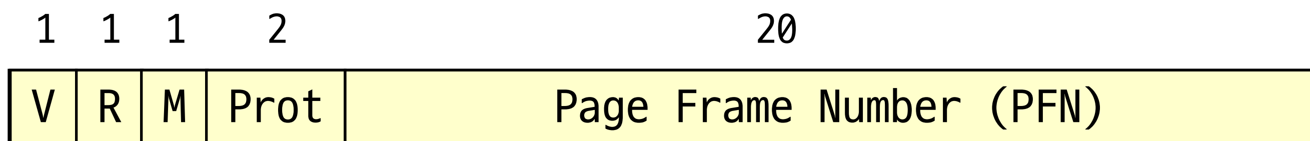
Physical address (20bits)

Translation Using a Page Table



Paging (7)

Page Table Entries (PTEs)



- Valid bit (V) says whether or not the PTE can be used
  - It is checked each time a virtual address is used
- Reference bit (R) says whether the page has been accessed
  - It is set when a read or write to the page occurs
- Modify bit (M) says whether or not the page is dirty
  - It is set when a write to the page occurs
- Protection bits (Prot) control which operations are allowed on the page
  - Read-only, Read-write, Execute-only, etc.
- Page frame number (PFN) determines physical page

Paging (8)

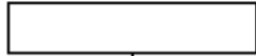
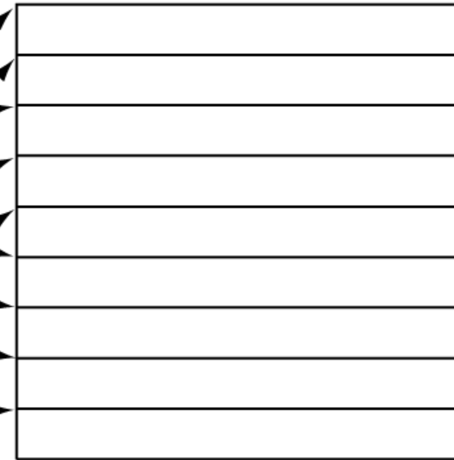
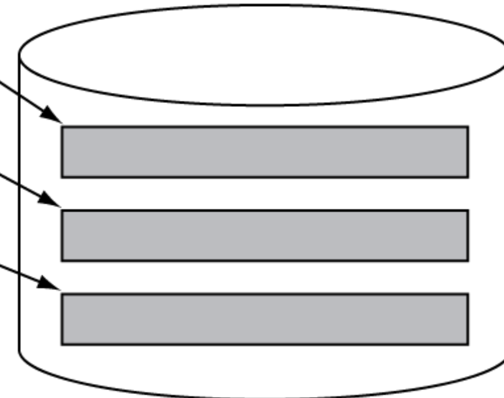
Valid / Invalid bit

- "Valid" indicates that the associated page is in the process' virtual address space, and is thus a legal page
- "Invalid" indicates that the page is not in the process' virtual address space
  - Never allocated or page out to disk

Protection

- Memory protection is implemented by protection bit for each frame
- Finer level of protection is possible for valid pages
  - Read-only
  - Read-write
  - Execute-only

Mapping Pages to Storage

Paging (9)

Advantages

- Easy to allocate physical memory
  - Physical memory is allocated from free list of frames
  - To allocate a frame, just use a free frame from its free list
- No external fragmentation
- Easy to "page out" chunks of a program
  - All chunks are the same size (page size)
  - Use valid bit to detect reference to "paged-out" pages
  - Pages sizes are usually chosen to be convenient multiple of disk block sizes
- Easy to protect pages from illegal accesses
- Easy to share pages

Paging (10)

Disadvantages

- Can still have internal fragmentation
  - Process may not use memory in exact multiple of pages
- Memory reference overhead (Performance overhead)
  - 2 references per address lookup (page table, then memory)
  - Solution: get a hardware support (TLB)
- Memory required to hold page tables can be large (Space overhead)
  - Need one PTE per page in virtual address space
  - 32-bit address space with 4KB pages = 2^{20} PTEs
  - 4 bytes/PTE = 4MB per page table
  - OS's typically have separate page tables per process (25 processes = 100MB of page tables)
  - Solution: page the page tables, multi-level page tables, inverted page tables, etc.

Paging Summary

Virtual memory

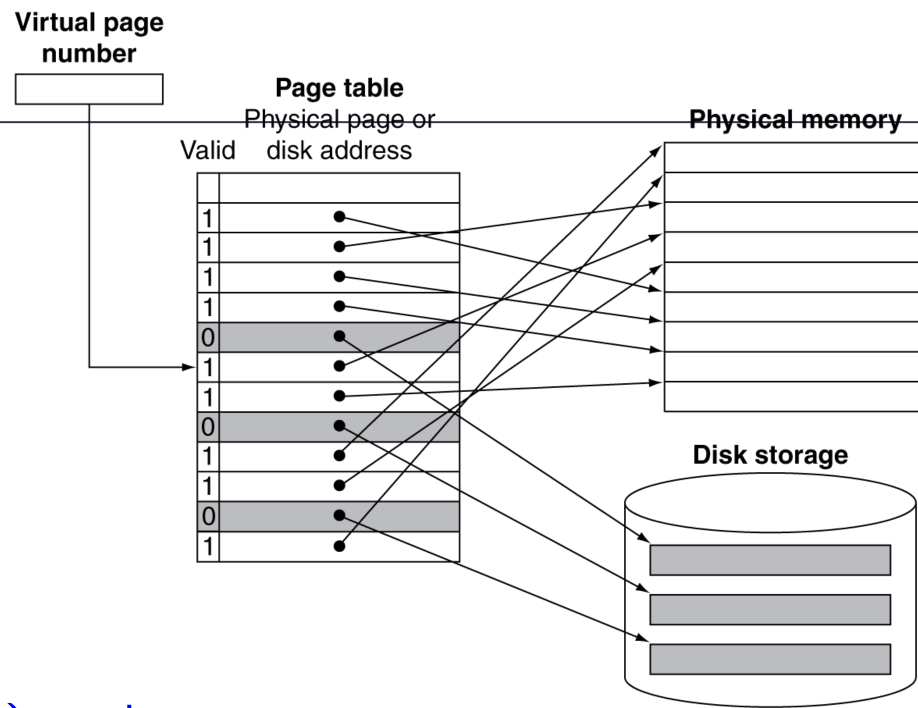
2. Addr 0x13000	Page 3
	Page 2
3. Addr 0x7000	Page 1
1. Addr 0x4000	Page 0

V	R	M	Prot	Page Frame Number (PFN)
V	R	M	Prot	Page Frame Number (PFN)

Physical memory

Frame 11
Frame 10
Frame 9
Frame 8
Frame 7
Frame 6
Frame 5
Frame 4
Frame 3
Frame 2
Frame 1
Frame 0

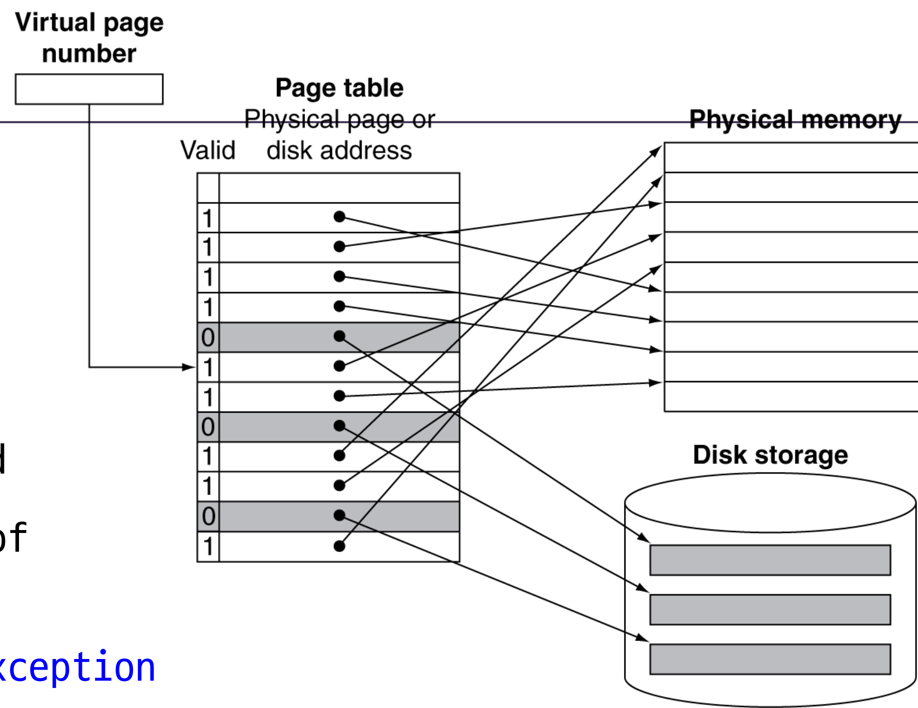
Demand Paging (1)



Demand paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- OS uses main memory as a (page) cache
  - Cache of all of the data allocated by processes in the system
  - When physical memory fills up, replacing (eviction and load)
- Evicted pages go to disk
  - Only need to write if they are dirty
  - Evict to a swap file on disk
  - Movement of pages between memory/disks is done by the OS (page fault)
  - Transparent to the application

Demand Paging (2)



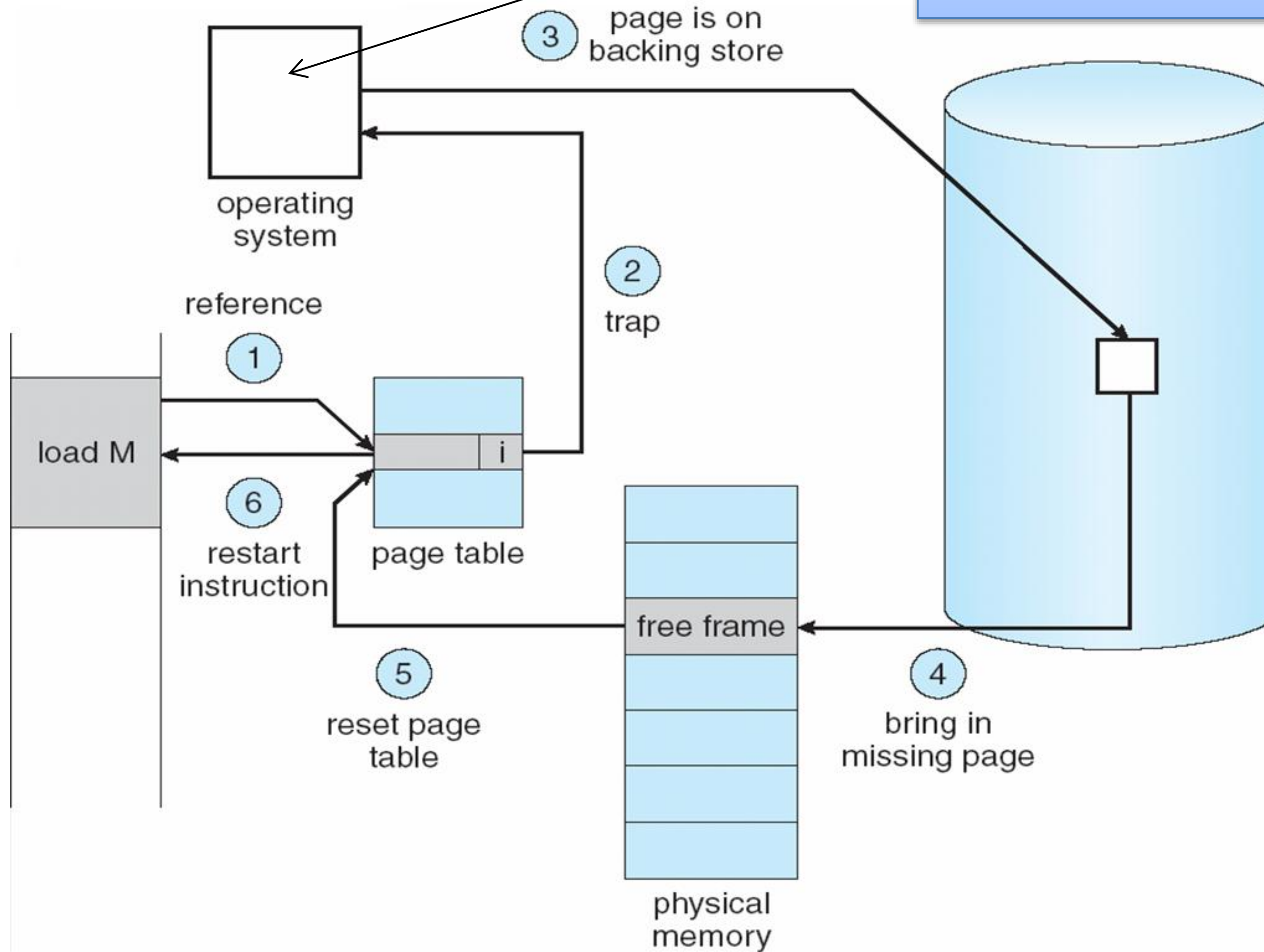
Page faults

- Referencing a virtual address in an evicted page
  - When the page was evicted, the OS sets the PTE as invalid
  - Stores (in PTE) the location of the page in the swap file
  - Accessing the page cause an exception
- The OS will run the page fault handler in response
  - Locating the page in swap file via invalid PTE
  - Handler reads page into a physical frame
  - Updates PTE to point to it and to be valid
  - Handler restarts the faulted process
- Where does the page that's read in go?
  - Must evict something else -> Which one? -> page replacement algorithm
  - OS typically tries to keep a pool of free pages around so that allocations don't inevitably cause evictions

Demand Paging (3)

Handling a page fault

```
page_fault_handler()
{
    ...
    ...
}
```



Demand Paging (4)

Why does this work?

- Locality
  - Temporal locality: locations referenced recently tend to be referenced again soon
  - Spatial locality: locations near recently referenced locations are likely to be referenced soon
- Locality means paging can be infrequent
  - Once you've paged something in, it will be used many times
  - On average, you use things that are paged in
- But this depends on many things:
  - Degree of locality in application
  - Page replacement policy
  - Amount of physical memory
  - Application's reference pattern and memory footprint

Demand Paging (5)

Why is this "demand" paging?

- When a process first starts up, it has a brand new page table, with all PTE valid bits "false"
  - All pages are empty
  - No pages are yet mapped to physical memory
- When the process starts executing:
  - Instructions immediately fault on both code and data pages (Cold miss / Cold page fault)
  - Only the code/data needed (demanded!!) by process will to be loaded
  - Faults stop when all necessary code/data pages are in memory

2. Addr 0x13000
3. Addr 0x7000
1. Addr 0x4000

Page 3
Page 2
Page 1
Page 0

Frame 11
Frame 10
Frame 9
Frame 8
Frame 7
Frame 6
Frame 5

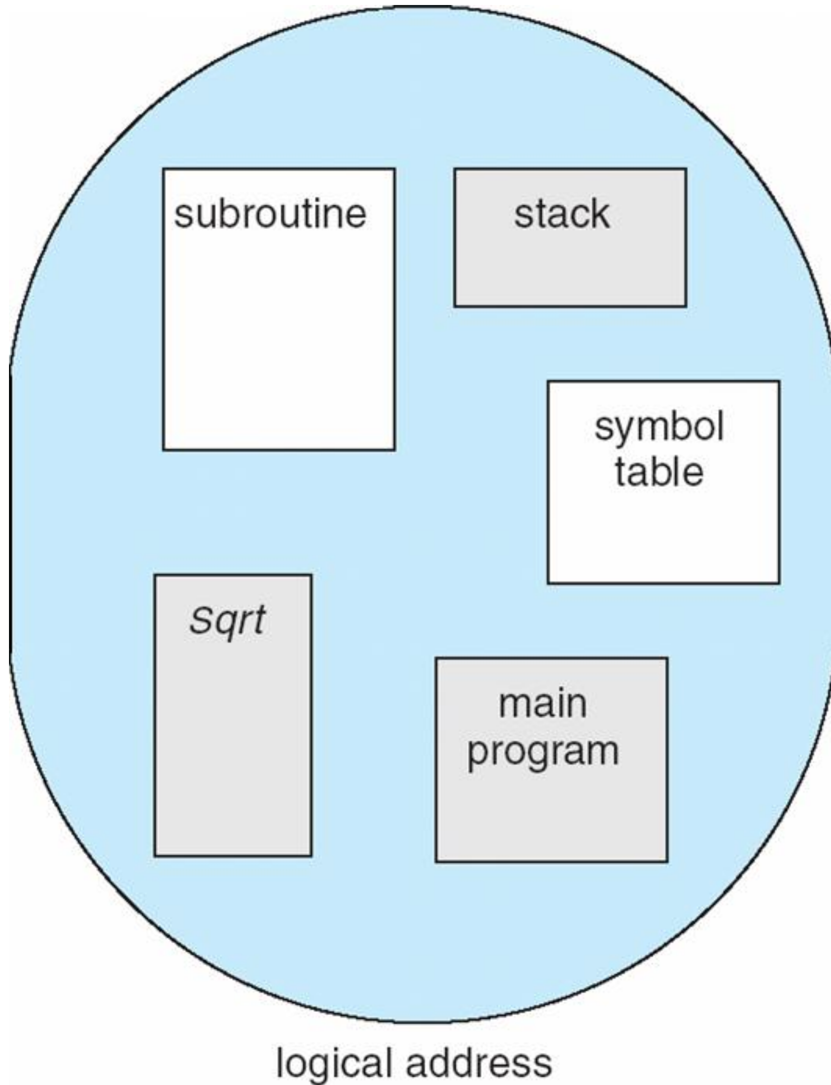
Segmentation (1)

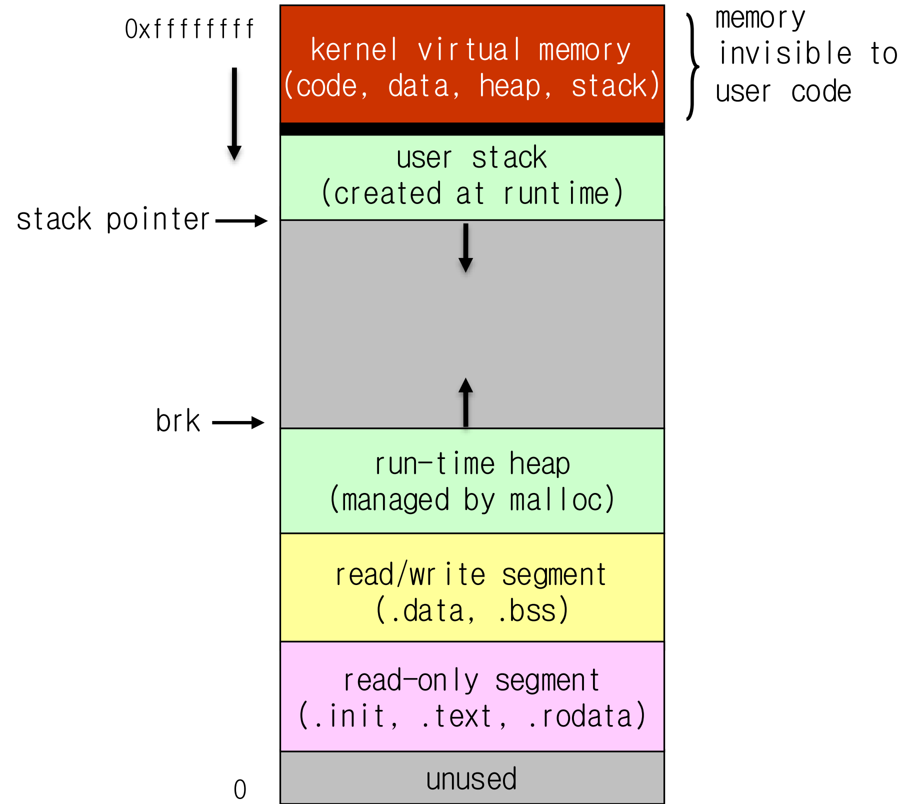
Segmentation

- Partitions memory into logically related data units
  - Code, stack, heap, etc.
- Users view memory as a collection of variable-sized segments
  - With no necessary ordering among them
  - Virtual address: <Segment #::Offset>
- Different segments can grow or shrink independently
  - Without affecting each other
- Natural extension of variable-sized partitions
  - Variable-size partitions: 1 segment / process
  - Segmentation: many segments / process

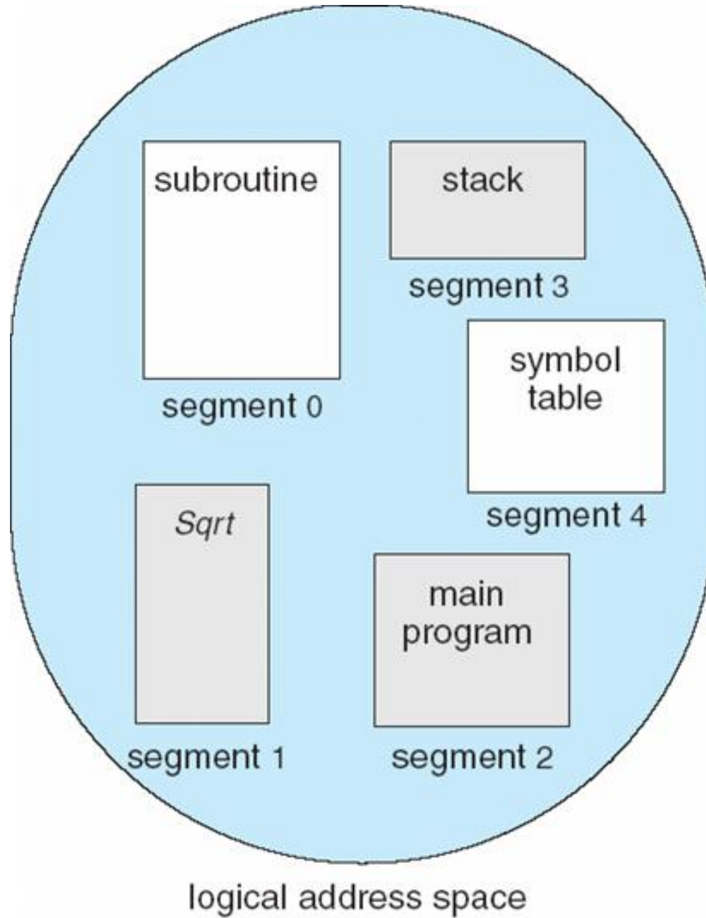
Segmentation (2)

User's view of a program



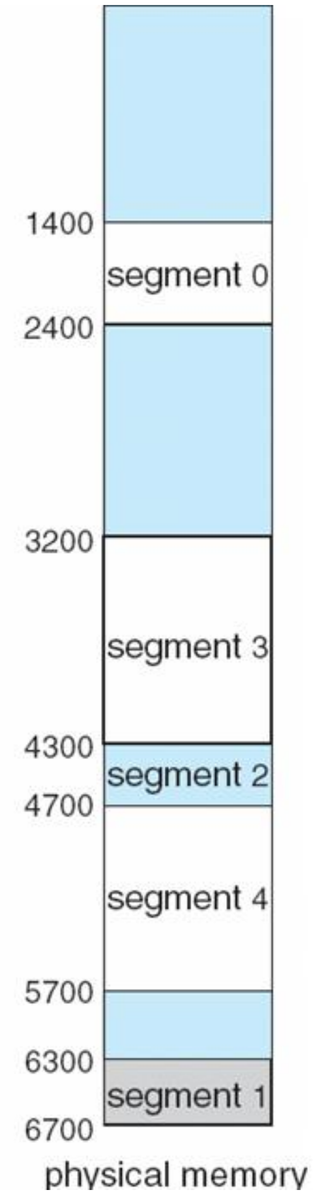


Segmentation (3)



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

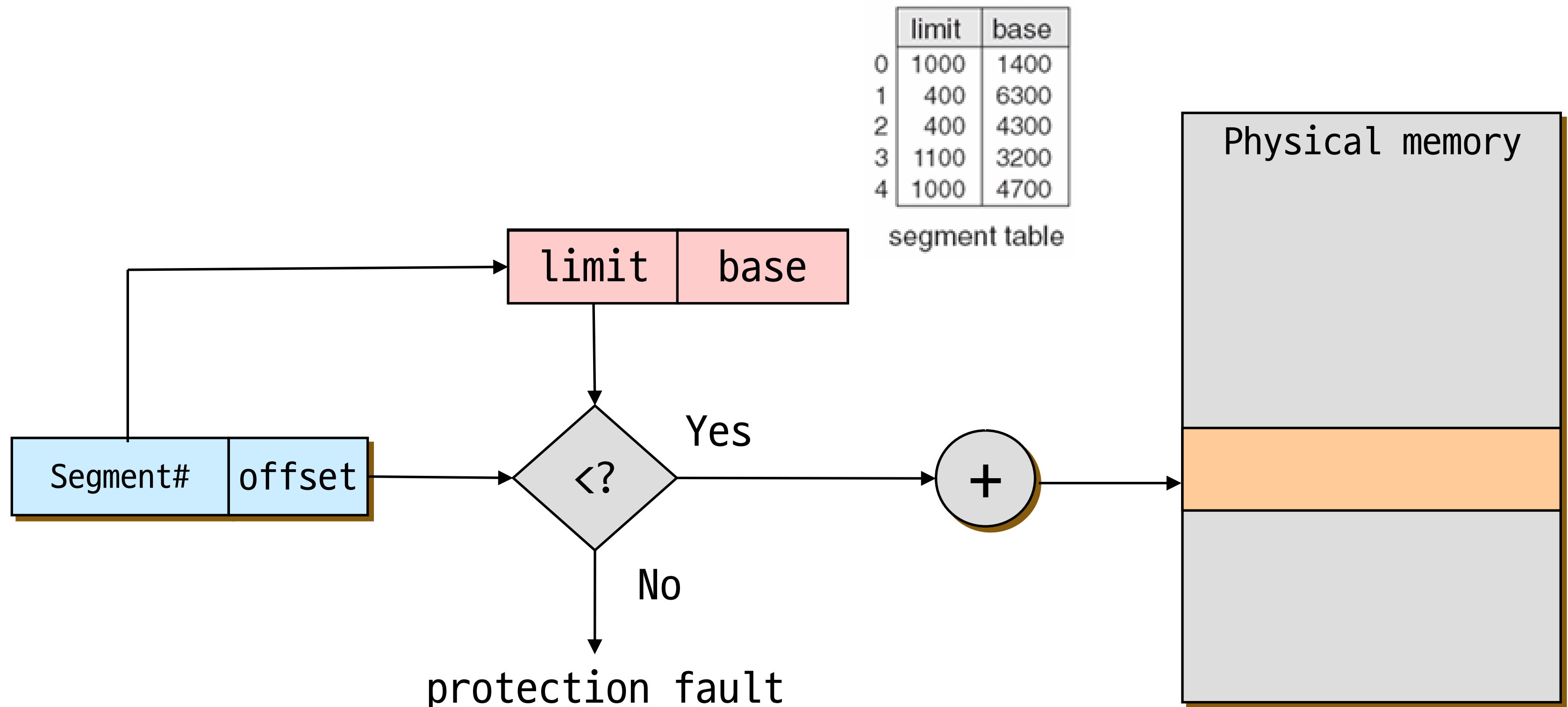
segment table



Segmentation (4)

Hardware support

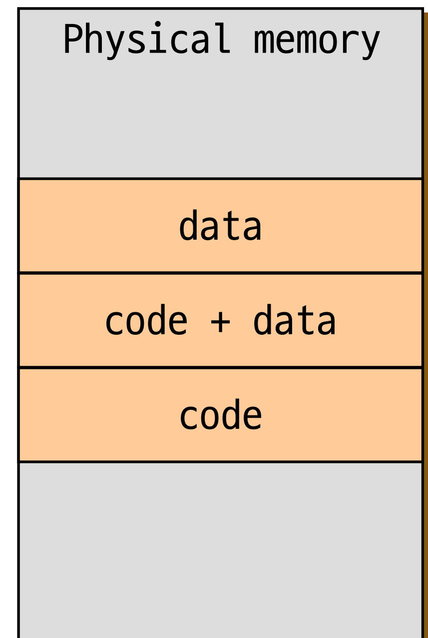
- Multiple base/limit pairs, one per segment (segment table)
- Segments are named by segment #, used to index into table



Segmentation (5)

Advantages

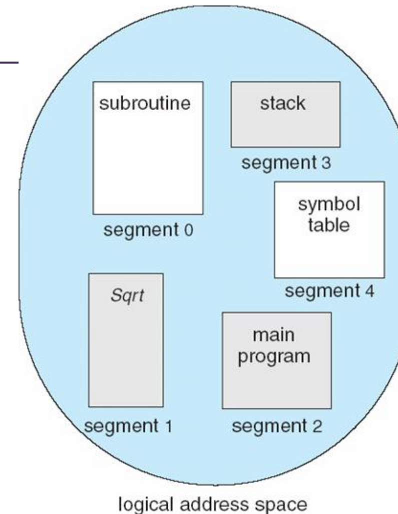
- Simplifies the handling of data structures that are growing or shrinking
- Easy to protect segments
  - With each entry in segment table, associate a valid bit
  - Protection bits (read/write/execute) are also associated with each segment table entry
- Easy to share segments
  - Put same translation into base/limit pair
  - Code/data sharing occurs at segment level
  - e.g. shared libraries



Segmentation (6)

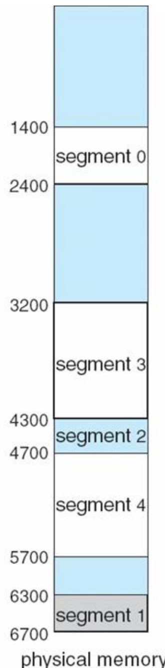
Disadvantages

- Large segment tables
  - Keep in main memory,
    use hardware cache for speed
- External fragmentation
  - Since segments vary in length,
    memory allocation is a dynamic
    storage-allocation problem



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

segment table



Paging vs. Segmentation (1)

	Paging	Segmentation
Block size	Fixed (4KB to 64KB)	Variable
Linear address space	1	Many
Memory addressing	page number + offset	segment + offset
Replacement	Easy (all same size)	Difficult (find where segment fits)
Fragmentation	Internal	External
Disk traffic	Efficient (optimized for page size)	Inefficient (may have small or large transfers)
Transparent to the programmers?	Yes	No

Paging vs. Segmentation (2)

	Paging	Segmentation
Can the total address space exceed the size of physical memory?	Yes	Yes
Can codes and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of codes easy?	No	Yes
Why was this technique invented?	large linear address space	logically independent address spaces (sharing and protection)

Paging vs. Segmentation (3)

Hybrid approaches

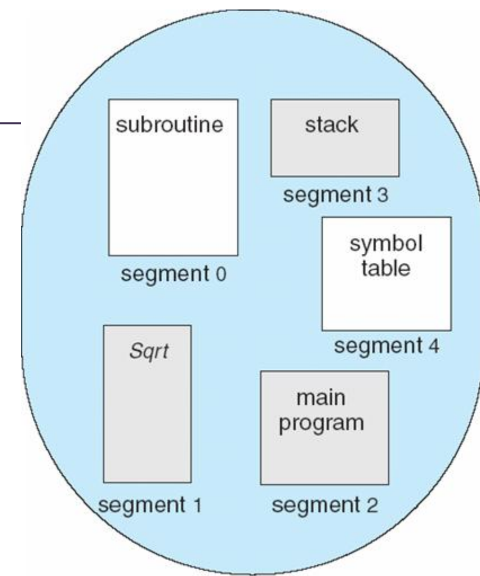
- Paged segments
  - Segmentation with Paging
  - Segments are a multiple of a page size
- Multiple page sizes
  - 4KB, 2MB, and 4MB page sizes are supported in IA32
  - 8KB, 16KB, 32KB or 64KB in Alpha AXP Architecture
    (43, 47, 51, or 55 bits virtual address)

Segmentation with Paging (1)

Combine segmentation and paging

- Use segments to manage logically related units
  - Code, data, heap, etc.
  - Segments vary in size, but usually large (multiple pages)
- Use pages to partition segments into fixed size chunks
  - Makes segments easier to manage within physical memory
  - Segments become "pageable"
    - rather than moving segments into and out of memory, just move page portions of segments
  - No external fragmentation
- The IA-32 supports segments and paging

Segmentation with Paging (2)



IA-32

logical address
selector
offset
descriptor table
segment table
segment descriptor
+
linear address
directory
page
offset
page frame
physical address
2 level page table
page directory
directory entry
page table
page table entry
page directory base register