Operating System Design

Dr. Jerry Shiao, Silicon Valley University

[Start 09 June]

Overview

- For Multiprogramming to improve CPU utilization and user response time, the Operating System must keep several processes in memory
- Memory Management Strategies
  - Hardware Support: Bind Logical Address to Physical Address
    - Memory Management Unit (MMU) : Bind/translate Logical addr → Physical addr.
  - Paging: Non-Contiguous Address Space / Fixed Size Frames
  - Segmentation: Non-Contiguous Address Space / Var Size Frms

storing group of functions of the program

- Virtual Memory: Process Memory Larger than Physical Memory. .
  - More programs run concurrently – multiprogramming.
  - More programs in memory – Increase CPU usage, less I/O for load or swap.
- Page Replacement and Frame Allocation Algorithms

Page replacement

  - FIFO, Optimal, LRU, LRU-Approximation (Reference Bit), Second-Chance, Counting-Based (LFU, MFU), Page-Buffering Page Replacement Algorithms.

Frame allocation

  - Equal, Proportional, Global vs Local Frame Allocation Algorithms.
- Thrashing mem swap in & out cont'
  - Not enough free frames for processes for executing press.
  - Limit effects of Thrashing using Locality window and Working Set Model.
- Allocating Kernel Memory
  - Slabs: Nonpaged contiguous frames.

Copyright @ 2009 John Wiley & Sons Inc.

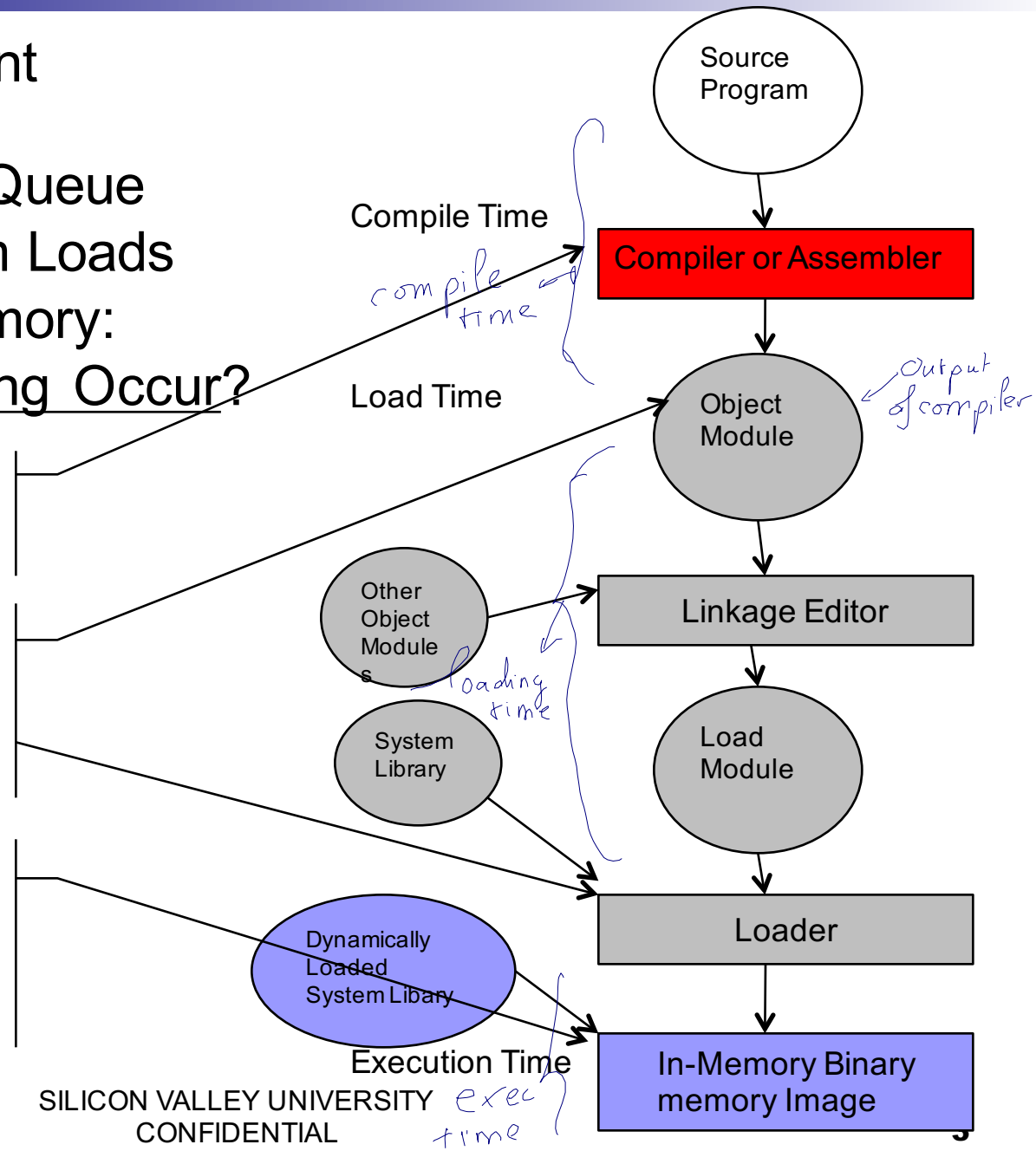
Memory Management

- Address Binding
- Process in Input Queue
- Operating System Loads Process into Memory: When does binding Occur?

Compile Time: physical memory location known. Absolute code (MS-DOS)

Load Time: physical memory location NOT known until Loader. Relocatable code. Loader: Final binding.

Execution Time: physical memory location not known until run time. Common method. NOTE: Require hardware assist.



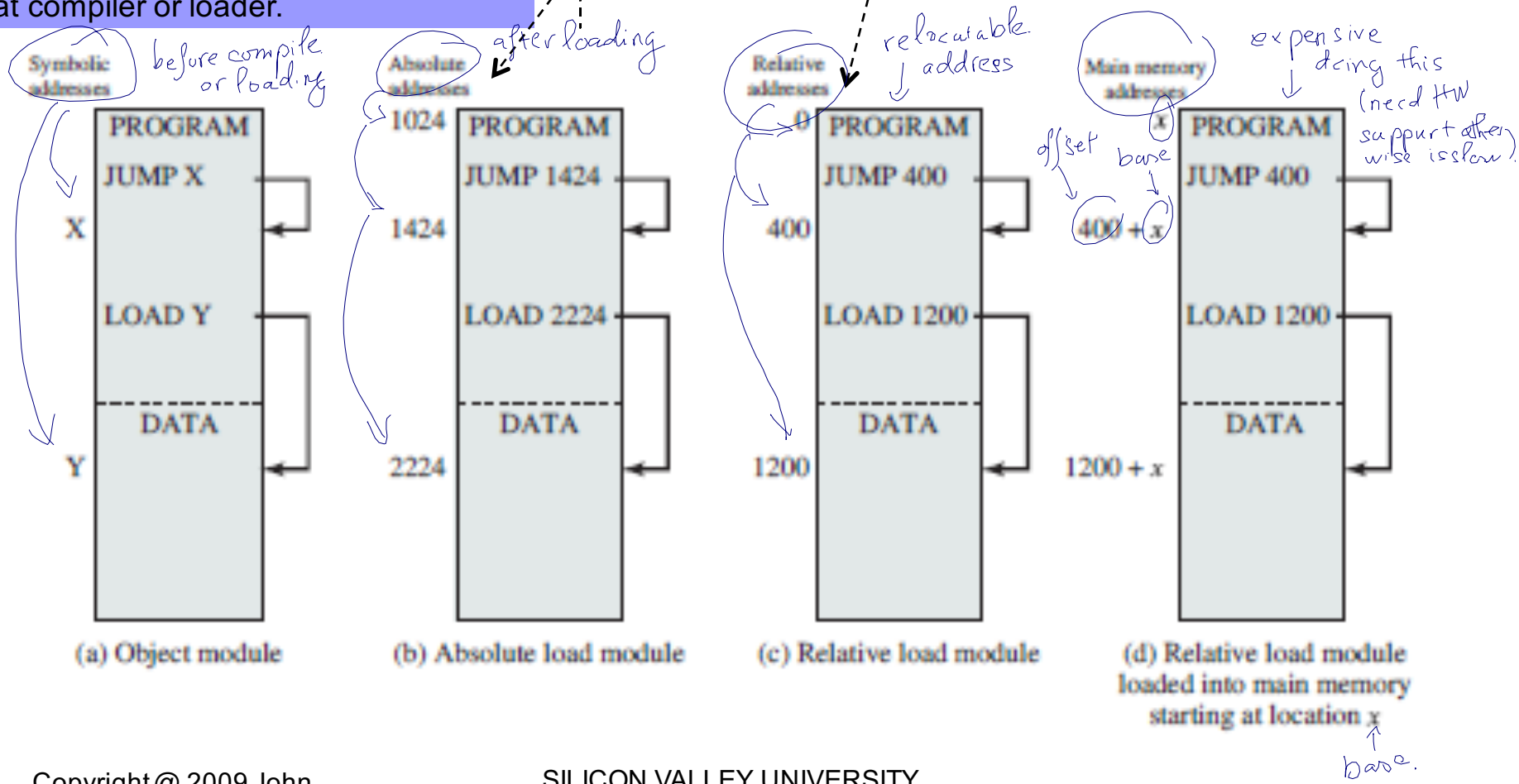
Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Load Modules

Initial program has symbolic references to functions or data items. Absolute addresses can be converted at compiler or loader.

All memory references within load module expressed relative to the beginning of the module. Load module can be located anywhere in main memory.



(a) Object module (b) Absolute load module (c) Relative load module (d) Relative load module loaded into main memory starting at location x

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

Memory Management Techniques

- Principle Operation of Memory Management is to bring processes into main memory for execution by the processor.

program is loaded contiguously in to the memmory.

no longer requires the program to load contiguosly into mem

program is divied into group of funcs, each group is stored in a segment.

only load into mem the nesessary funcs, otherwise just leave them on disk

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
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.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
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.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
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.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Copyright @ 2009 Pearson Education, Inc.

Table 7.1 Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
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.	No internal fragmentation.	Improved memory utilization and reduced overhead compared to dynamic partitioning.
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.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
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.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

V.S

weakness: external fragment

Table 7.2 Address Binding

(a) Loader

Binding Time	Function
Programming time	All actual physical addresses are directly specified by the programmer in the program itself.
Compile or assembly time	The program contains symbolic address references, and these are converted to actual physical addresses by the compiler or assembler.
Load time	The compiler or assembler produces relative addresses. The loader translates these to absolute addresses at the time of program loading.
Run time	The loaded program retains relative addresses. These are converted dynamically to absolute addresses by processor hardware.

(b) Linker

Linkage Time	Function
Programming time	No external program or data references are allowed. The programmer must place into the program the source code for all subprograms that are referenced.
Compile or assembly time	The assembler must fetch the source code of every subroutine that is referenced and assemble them as a unit.
Load module creation	All object modules have been assembled using relative addresses. These modules are linked together and all references are restated relative to the origin of the final load module.
Load time	External references are not resolved until the load module is to be loaded into main memory. At that time, referenced dynamic link modules are appended to the load module, and the entire package is loaded into main or virtual memory.
Run time	External references are not resolved until the external call is executed by the processor. At that time, the process is interrupted and the desired module is linked to the calling program.

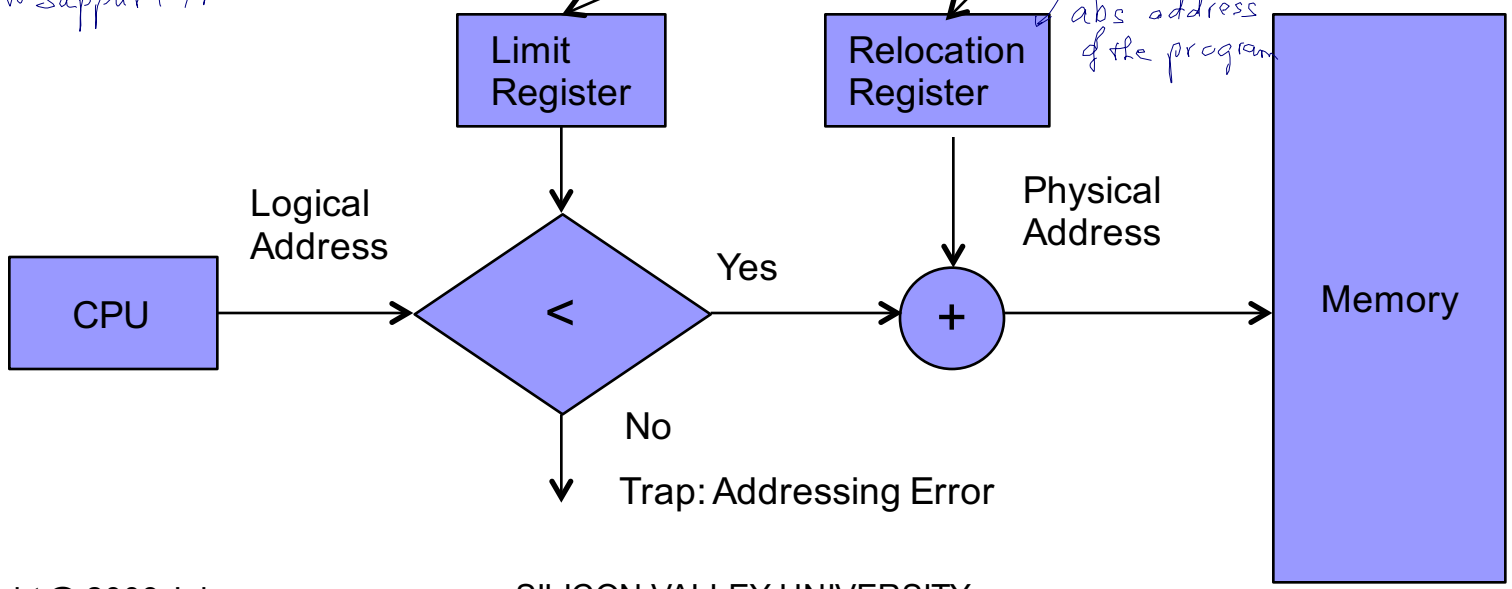
Memory Management

- User Process in Main Memory: Contiguous Memory Allocation
- Memory Shared by Operating System and User Process
- Memory Divided into Two Partitions:
  - Resident Operating System
    - Low Memory (Reside with Interrupt Vector).
  - User Processes
    - Each process in single contiguous section of memory.
  - Memory Mapping and Protection

Memory Mgt Unit

MMU maps logical address dynamically using relocation register and validates address range with limit registers.

Need HW support / Must be done in HW



Copyright @ 2009 John Wiley & Sons Inc.

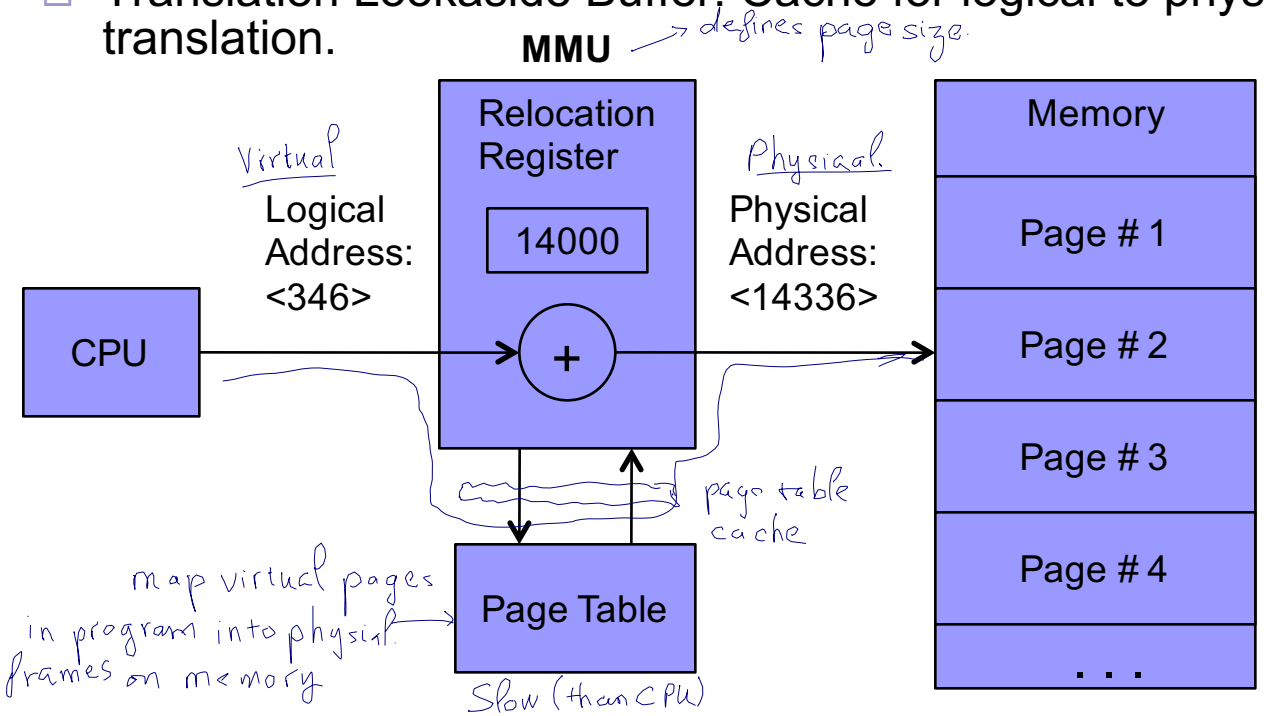
Memory Management

- Swapping
  - Process executing in memory does not always stay in memory until it completes execution.
- Swap Considerations:
  - Address Binding Method
    - Load time or compile time: swap process into same memory location.
    - Execution time: swap process into different memory location.
  - Backing Store (Disk) → where things stored when swapped out
    - Copies of all memory images in Swap Partition.
    - Operating System Ready Queue: Processes with memory images in Backing Store or in memory.
  - Context-Switch Time
    - Transfer time latency, proportional to amount of memory swapped.
    - Reduce swap time, dynamic memory requirements only request memory needed and release used memory.
  - Process with pending I/O Operations → swapped out until I/O completes
    - Use Operating System buffers and transfer to process memory when process swapped in.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

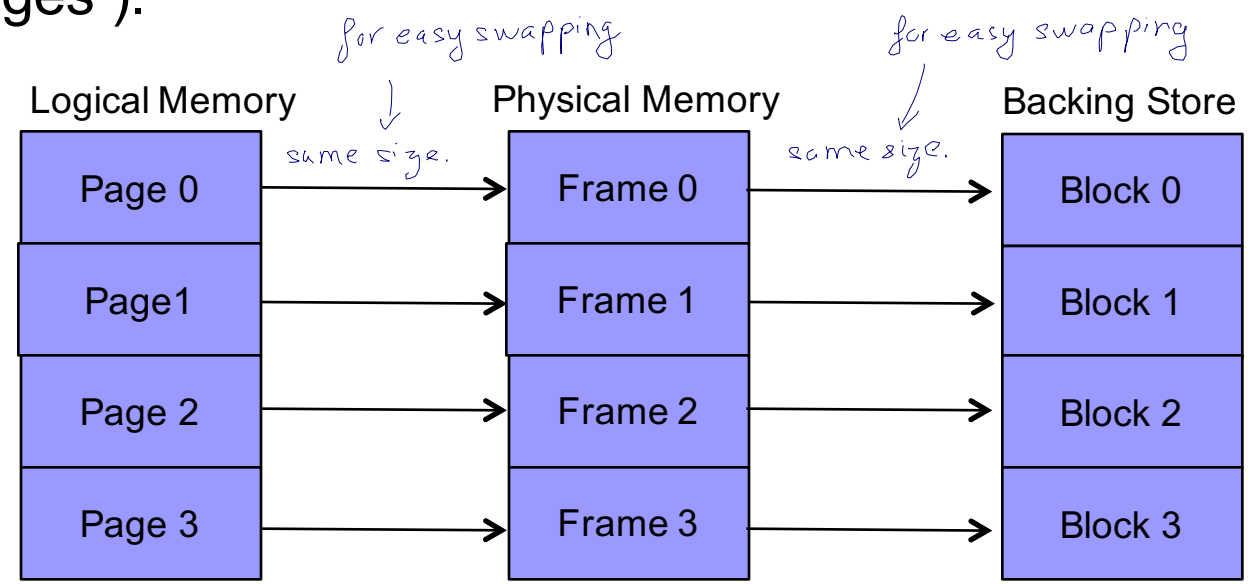
- Run-time mapping Virtual to Physical addresses done by:
  - Memory Management Unit (MMU)
- Divides Virtual Address Space into Pages (512Bytes to 16 Mbytes)
  - Paging: Memory Management scheme permits physical address space to be noncontiguous.
  - Translation Lookaside Buffer: Cache for logical to physical address translation.



Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

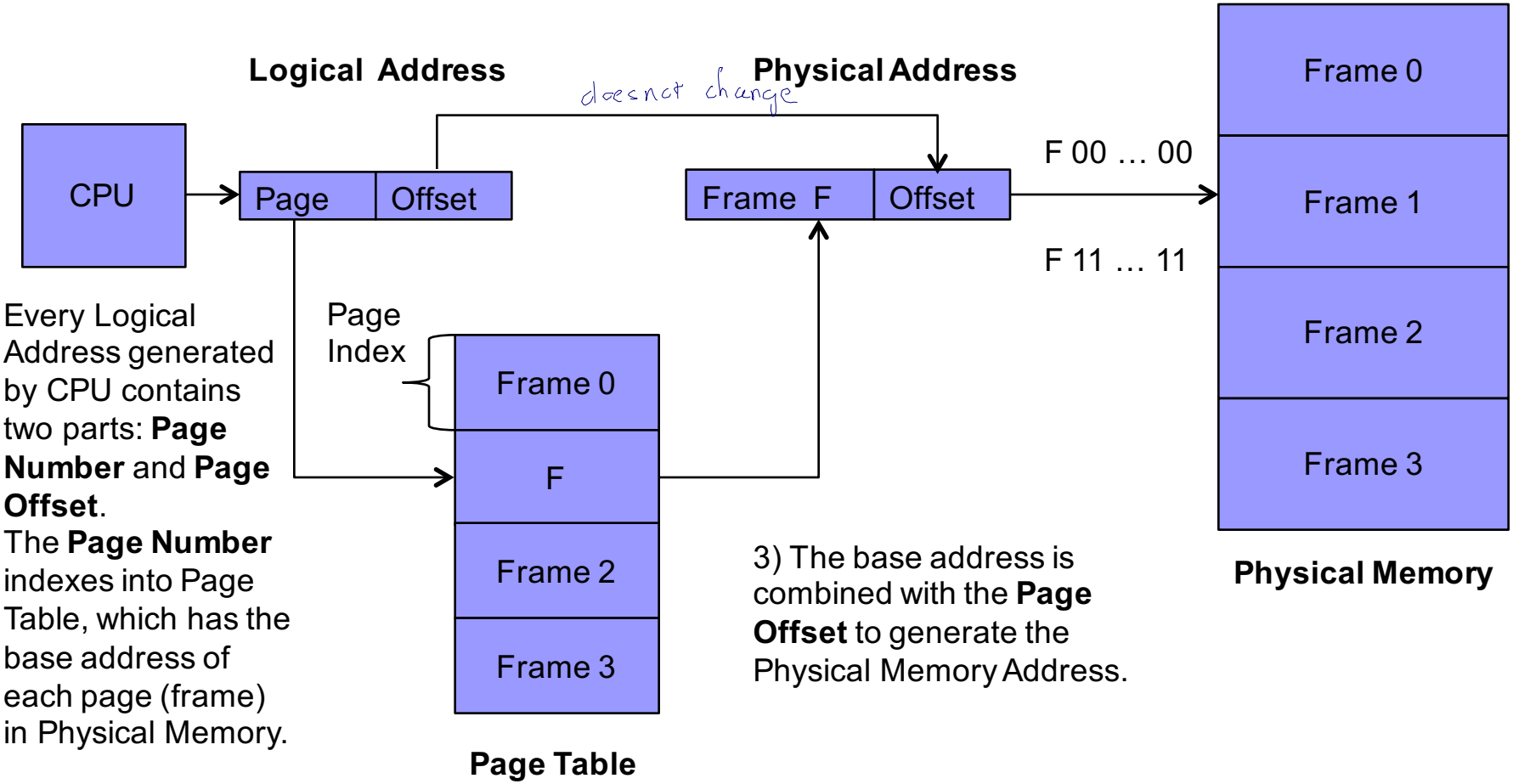
- Paging
- Permits the Physical Address Space of Process to be Non-contiguous.
- Avoids Memory External Fragmentation and Compaction.
- Avoids Backing Store Fragmentation.
- Frames: Physical Memory partitioned into fixed-size blocks (Frames).
- Page: Logical Memory partitioned into fixed-size blocks (Pages).



Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Paging
- Paging Hardware



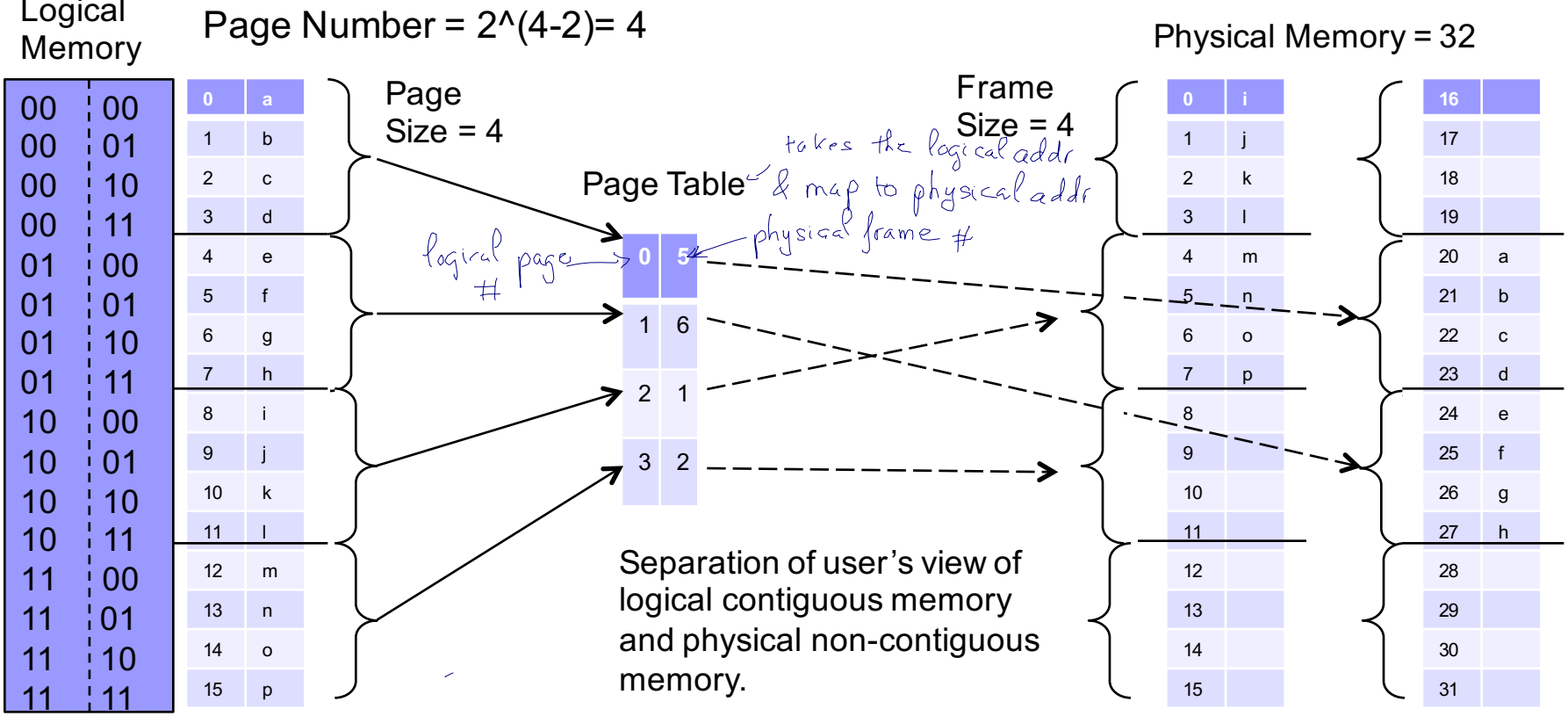
1) Every Logical Address generated by CPU contains two parts: **Page Number** and **Page Offset**.
2) The **Page Number** indexes into Page Table, which has the base address of each page (frame) in Physical Memory.

3) The base address is combined with the **Page Offset** to generate the Physical Memory Address.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Paging
- Page Size Defined by Hardware
  - Power of 2: Typically 4K Bytes and 8 Kbytes
  - Logical Address = 2^m, Page Size = 2^n, Page Number = 2^{m-n} $= \frac{2^m}{2^n}$
    - M = 4, Logical Address = 2^M=2^4=16, n =2 Page Size = 2^N=2^2=4, Page Number = 2^(4-2)= 4



Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Paging
- Hardware Support
- Page Table must be fast, every access to memory goes through Page Table.

  must be fast otherwise be bottleneck.
- Page Table can be implemented as set of registers.
  - Small Page Table (256 entries).
- Page Table implemented in memory with Page-Table Base Register (PTBR), starting address of Page Table.
  - Large Page Table: PTBR in Memory is slow.
- Page Table in Fast-Lookup Hardware Cache.
  - Translation Look-Aside Cache (TLB).
    - High-Speed Memory, contains few Page Table entries (64 to 1024 Entries).
    - TLB Miss: Access Page Table in memory.
    - TLB Replacement policy (LRU, random).
    - Some TLB entries are “Wired Down” (Kernel Code).
    - Each process Address Space Identifies (ASIDs) stored in TLB entry.
      - Allows multiple processes to use TLB.
    - Hit Ratio: Percentage of times that a page is found in the TLB.

Copyright @ 2009 John Wiley & Sons Inc.

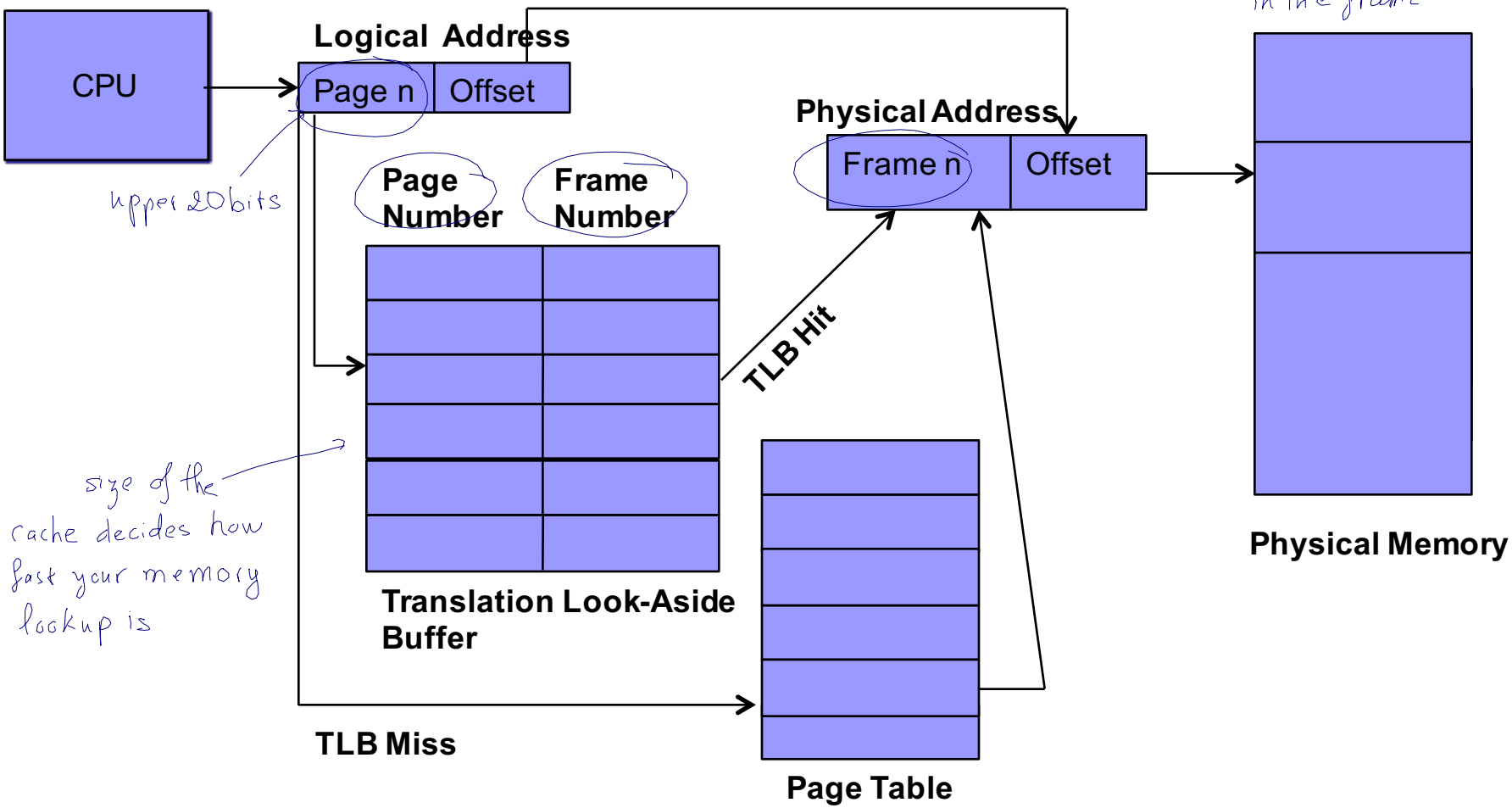
Memory Management

- Paging
- TLB Hardware Support

Logical address = page no. + offset

used for lookup frame in physical memory

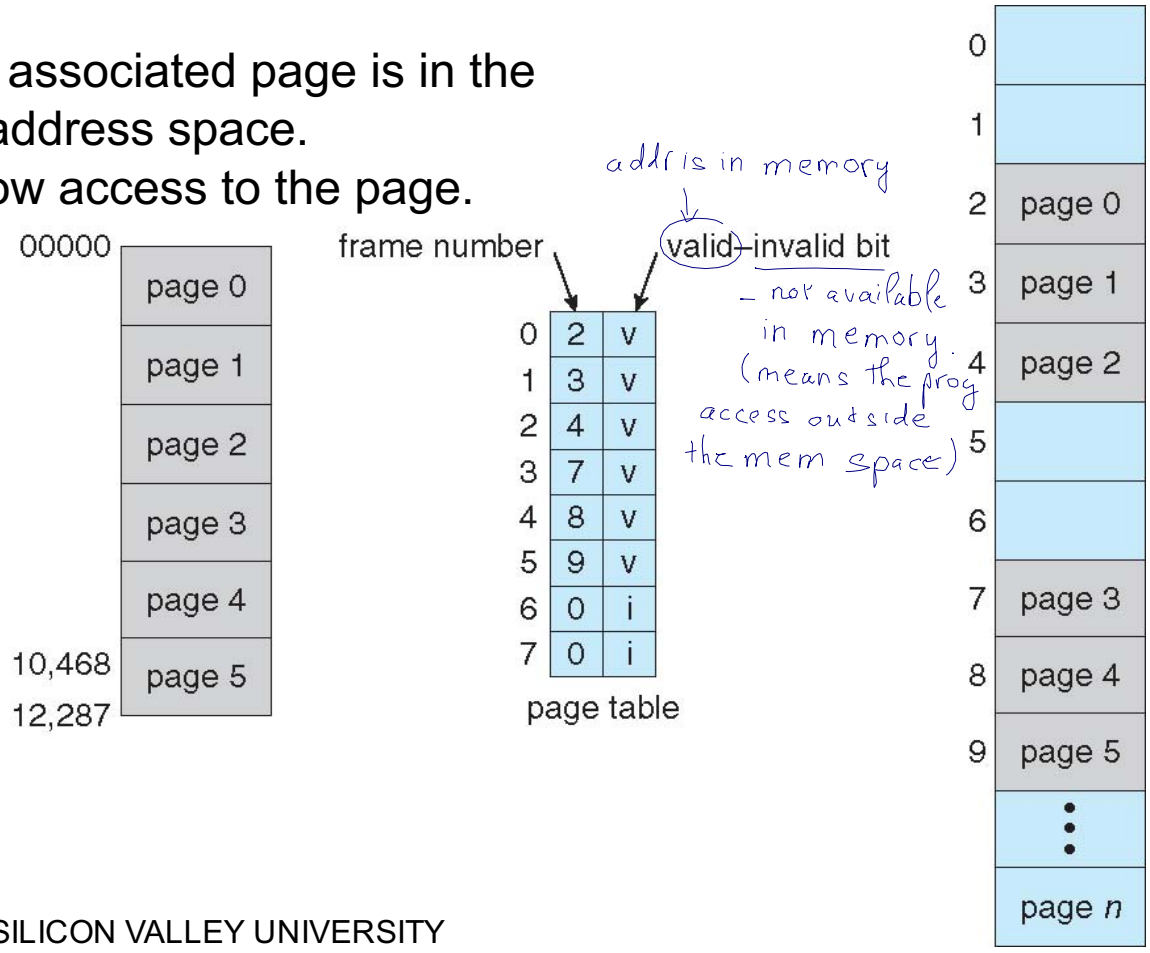
Relative location of data in the frame



Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Paging
- Memory Protection with Protection Bits
  - Read-Write, Read-Only, Execute-Only
  - Valid-Invalid:
    - Tests whether the associated page is in the process's logical address space.
    - OS allow or disallow access to the page.
  - Hardware Trap to Operating System

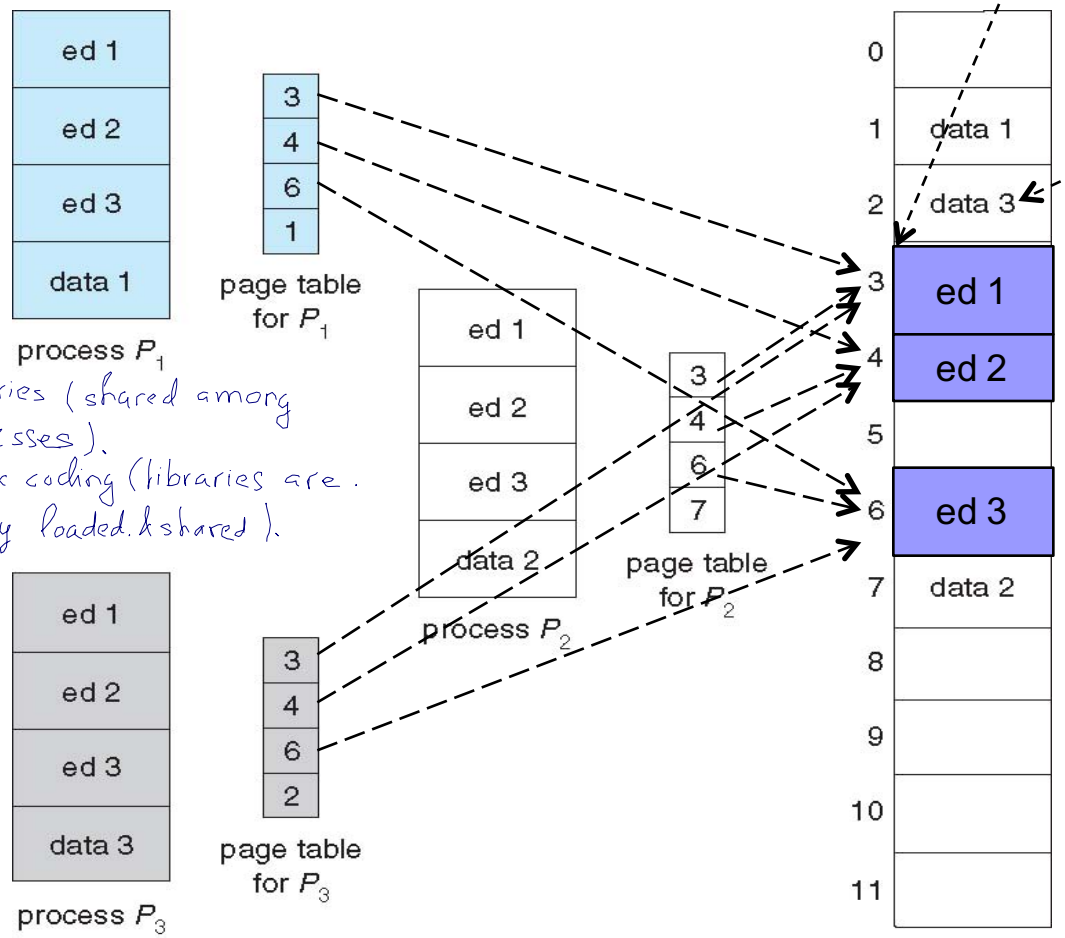


Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

relocatable file = L file.

- Paging
- Shared Pages: Sharing Common Code



- for system libraries (shared among multiple processes).
- usually for dynamic coding (libraries are already loaded & shared).

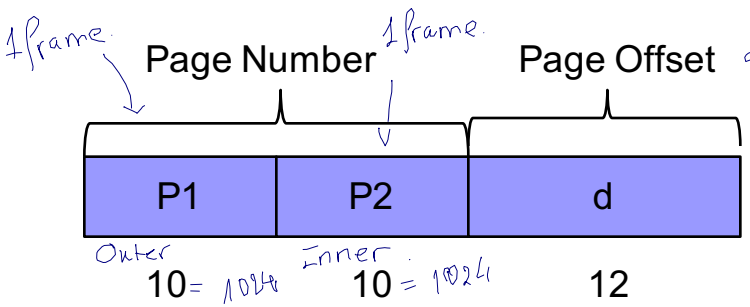
Shared Pages:
-Read-Only (reentrant) code (vi, cc, run-time library).
-Same logical address space in all processes.

Private Code / Data:
-Process has own copy of registers.
-Process has separate code and data.
-Pages for private memory appear anywhere in logical address space.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

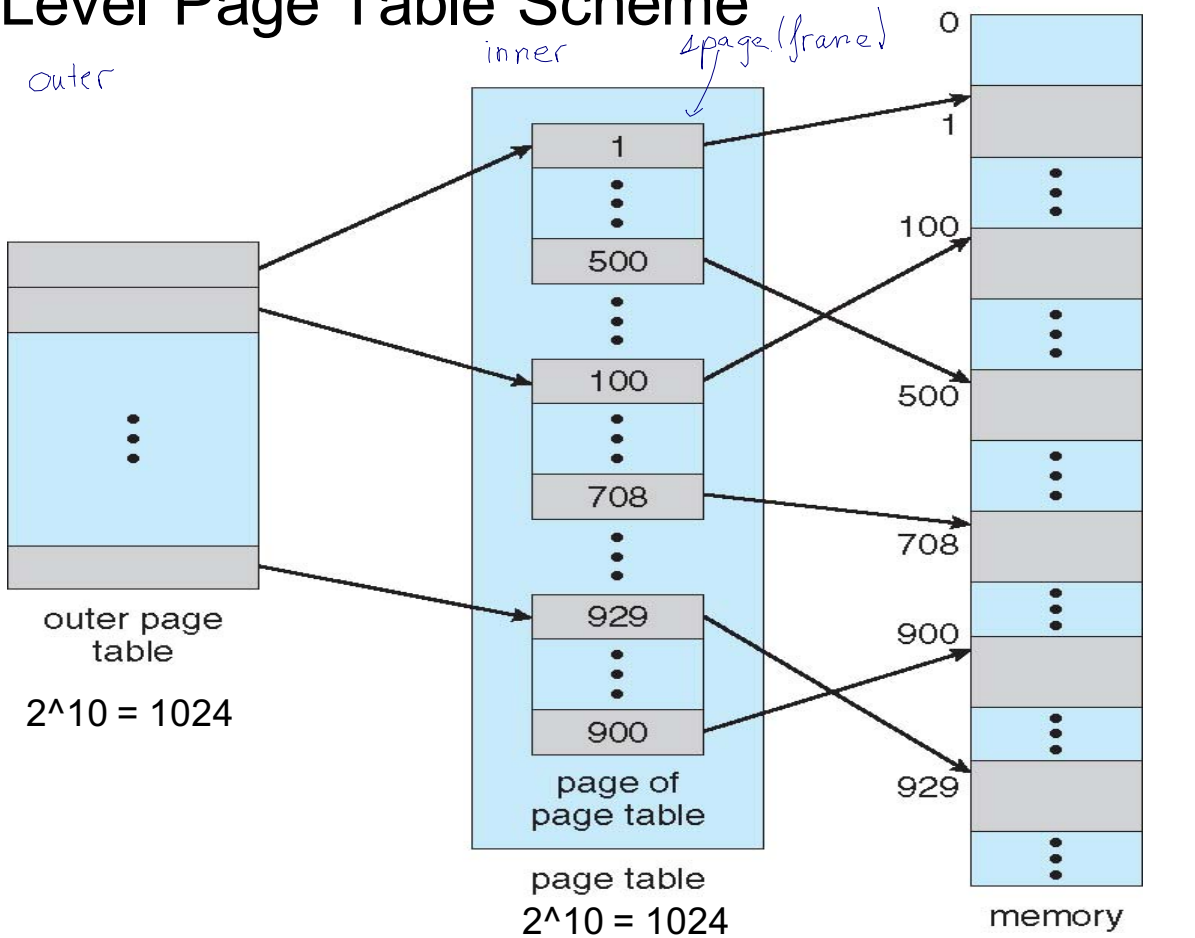
- Structure of the Page Table
- Hierarchical Page Tables
  - Problem:
    - Logical Address: 32 Bits
    - Page Size: 4KBytes (2^12 = 4096)
    - Page Table Entries: 2^(32 – 12) = 2^20 = 1M Entries
    - Page Table Size: 1M Entries X 4 Bytes/Entry = 4MBytes Page Table per Process
  - Two-Level Page Table Scheme



  - P1 = Index into the Outer Page Table
  - P2 = Displacement within the page of the outer Page Table
  - d = Physical Page offset

Memory Management

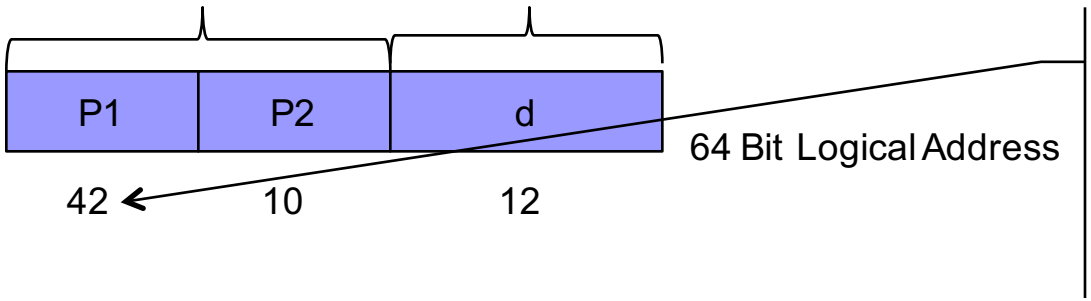
- Structure of the Page Table
- Hierarchical Page Tables
- Two-Level Page Table Scheme



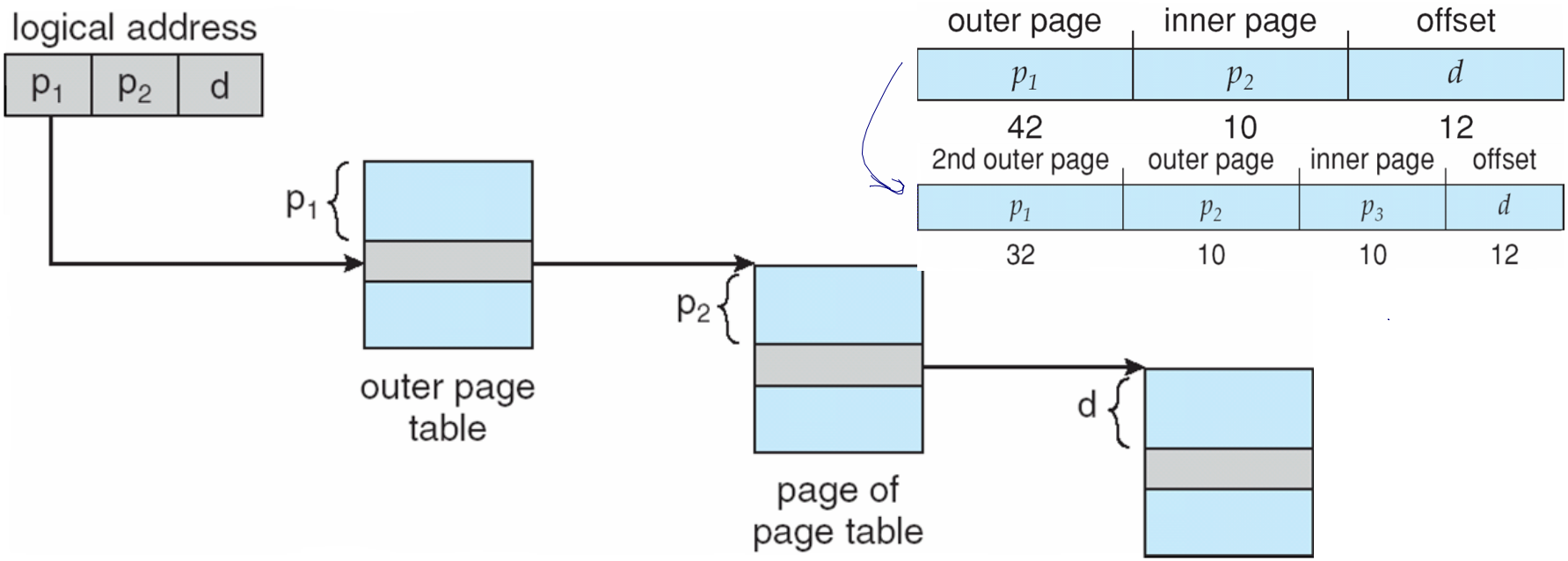
Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Hierarchical Page Tables
- N-Level Page Tables



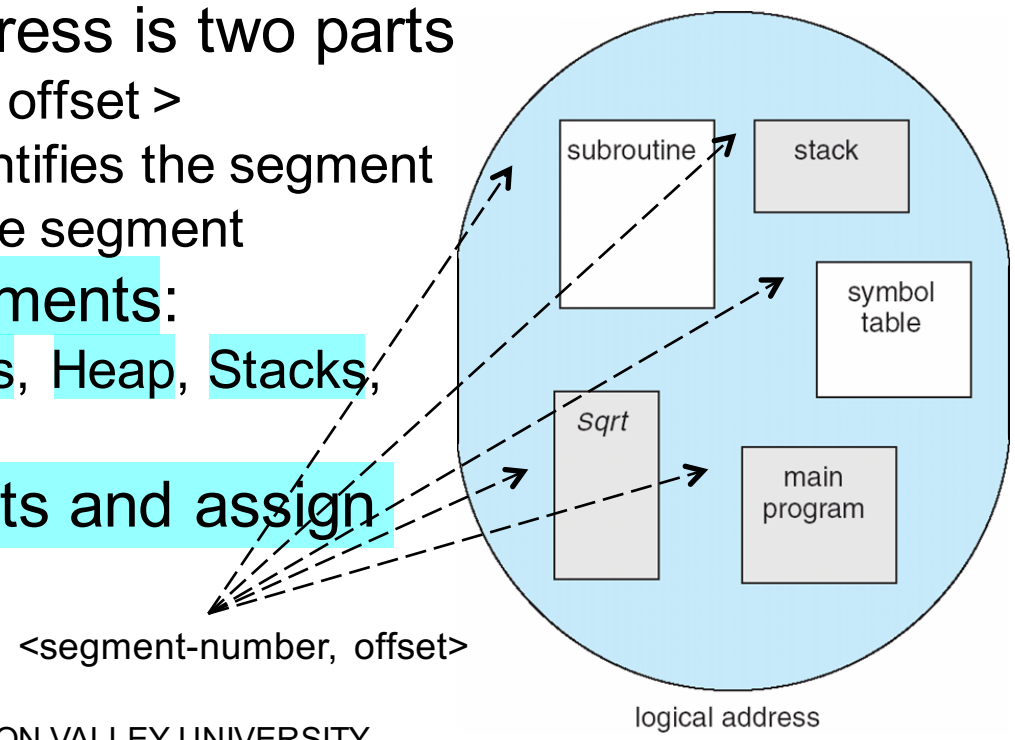
64 Bit Logical Address will use 2^42. CANNOT use Two-Level Page Table.
Outer Page Table would have to be partitioned further.



Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Segmentation ← not always the same size. size can be defined by the programmer (var sized).
- Memory viewed as collection of variable-sized segment, with no ordering among segments.
  - Each segment has specific purpose:
    - Main program, subroutine, stack, symbol table, library function.
- Logical Address Space: Collection of Segments
- Segment Logical Address is two parts
  - < segment – number, offset >
  - Segment-number: Identifies the segment
  - Offset: Offset within the segment
- Compiler creates segments:
  - Code, Global Variables, Heap, Stacks, Standard C Library
- Loader takes segments and assign segment numbers


subroutine
stack
symbol table
Sqrt
main program
<segment-number, offset>
logical address

Copyright @ 2009 John Wiley & Sons Inc.

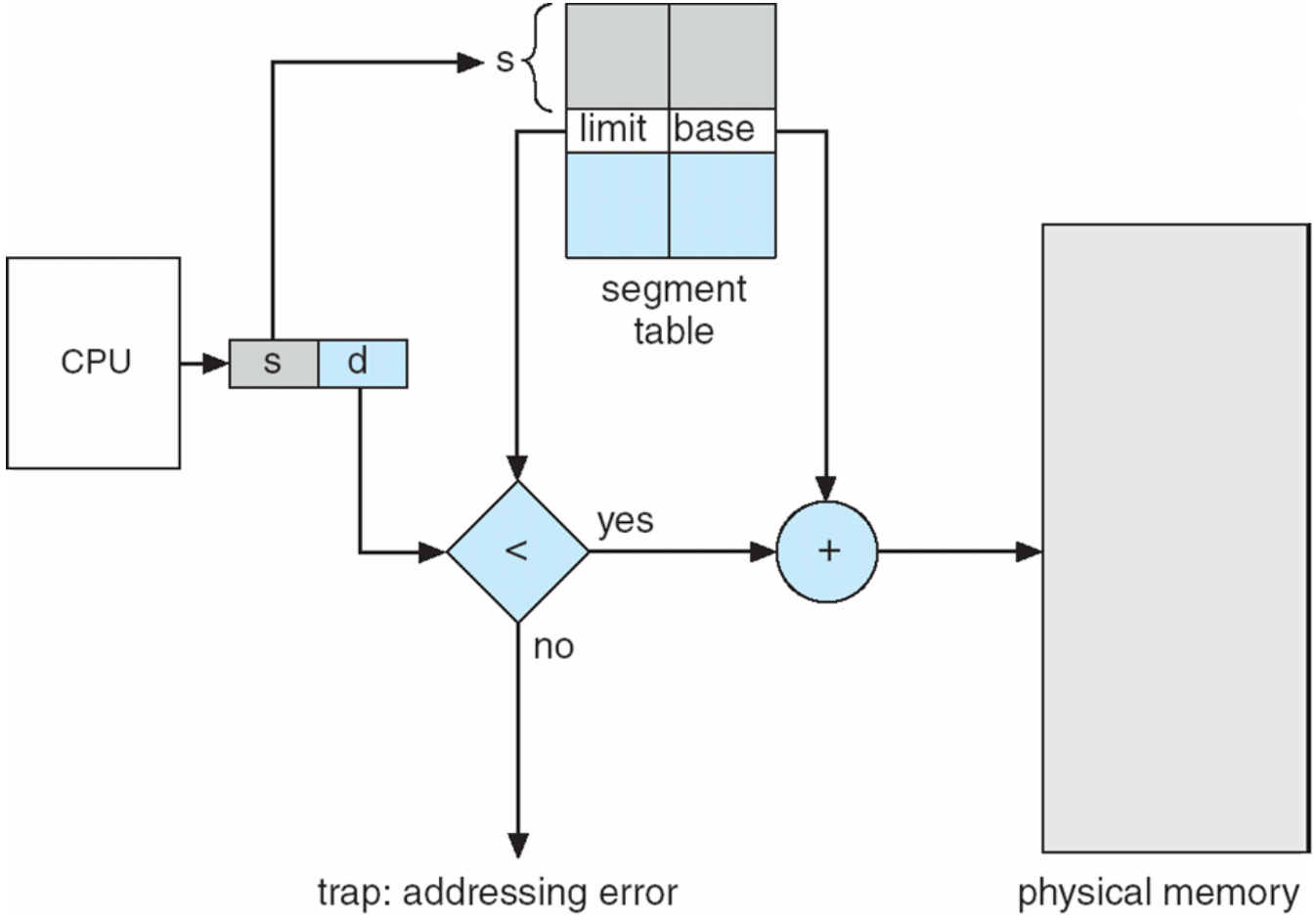
Memory Management

- Segmentation
- Segment Table
  - Segment Base: Starting physical address where the segment resides in memory.
  - Segment Limit: Length of the segment.
  - < segment – number, offset > ← logical address of a segment
    - Segment-number is the index into the segment table.
    - Offset is the offset into the segment.
    - Offset between 0 and the Segment Limit.
    - Offset added to the Segment Base to produce the physical address in memory.
  - Segment Table Base Register (STBR)
    - Segment Table's location in physical memory. Points to STBR saved in Process Control Block.
  - Segment Table Length Register (STLR)
    - Number of segments used by a program.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Segmentation

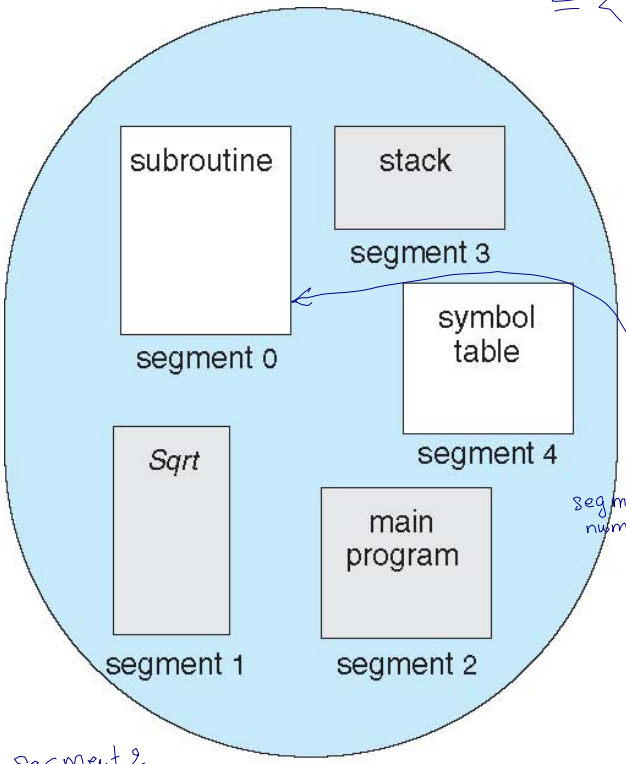


Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Segmentation

logical address of segment
= < segment-number , offset >



logical address space

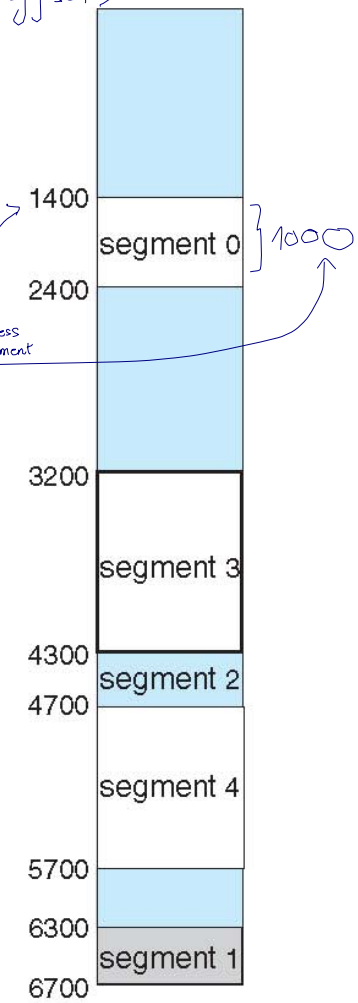
size of each segment

starting address of each segment

segment number

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

segment table



physical memory

segment 2

offset

base

offset

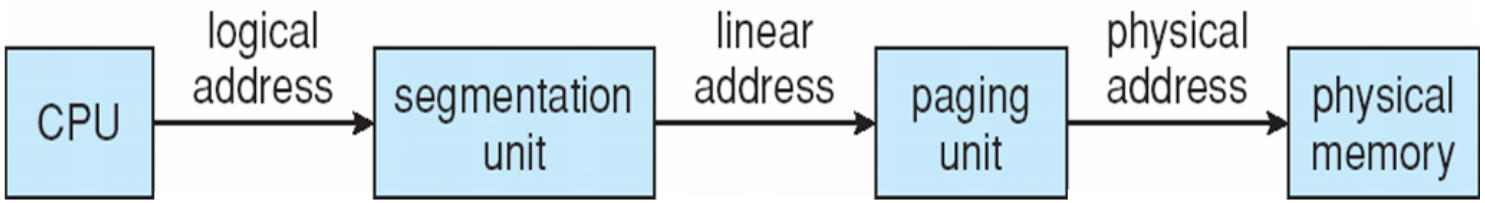
Logical Address = < 2, 53 >
Base = 4300: Physical Address = 4300 + 53 = 4353
Logical Address = < 3, 852 >
Base = 3200: Physical Address = 3200 + 852 = 4052

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Segmentation
- Intel Pentium
  - Supports Pure Segmentation and Segmentation with Paging

each segment has 1 table where it stores the segment



- Logical Address to Physical Address Translation
  - CPU generates Logical Address.
    - Logical Address Passed to Segmentation Unit.
  - Segmentation Unit generates Linear Address.
    - Linear Address Passed to Paging Unit.
  - Paging Unit generates Physical Address in Memory.

logical address
→ linear address
→ physical address

- One Segment Table Per Process
- One Page Table Per Segment

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

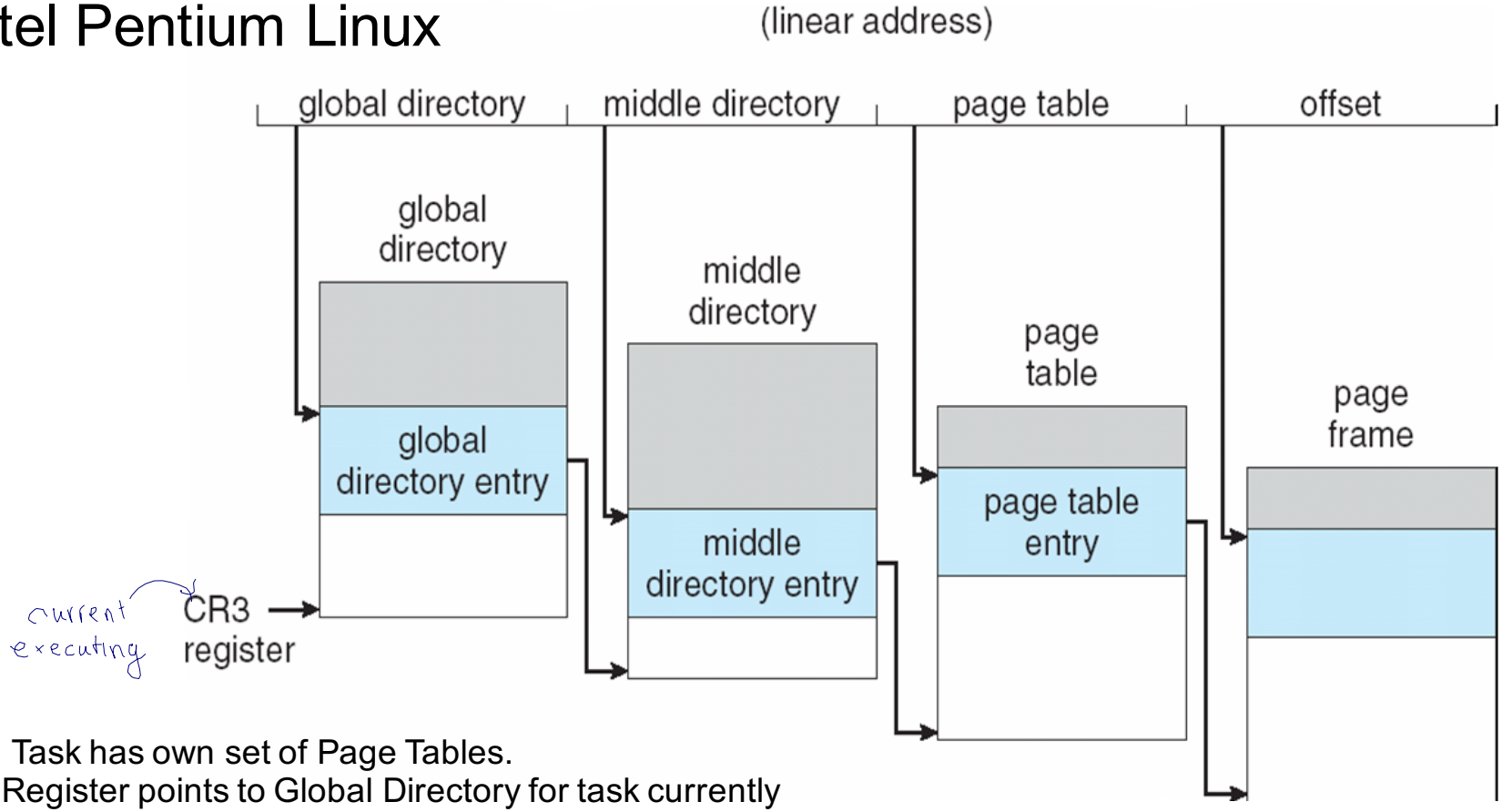
- Segmentation
- Intel Pentium Linux
- Linux Uses 6 Segments:
  - Kernel Code Segment
  - Kernel Data Segment
  - User Code Segment and User Data Segment
    - Shared by all processes in user mode.
    - All processes uses same logical address space.
    - Segment Descriptors in Global Descriptor Table (GDT).
  - Task-State Segment (TSS)
    - Store the hardware context of each process during context switches.
  - Default Local Descriptor Table(LDT)
- Linux Uses Three-Level Paging for 32-Bit or 64-Bit Architectures.

global directory	middle directory	page table	offset

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Segmentation
- Intel Pentium Linux



- Each Task has own set of Page Tables.
- CR3 Register points to Global Directory for task currently executing.
- CR3 Register saved in TSS Segments of the task during context switch.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Benefits from Paging and Segmentation:
  - All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time.
  - Process swapped in and out at different regions of main memory.
  - Process are broken up into number of pieces (pages or segments) and are not contiguously located in main memory during execution.
  - Page Table or Segment Table with dynamic run-time address translation permits supports this.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- What about Virtual Memory?

only working set needs to be in memory

- Entire Process Image Not Needed in Memory
  - Code for Error Conditions.
  - Arrays, Lists, and Tables are over subscribed.
  - Functions to handle options or features of process that is rarely used.
  - Resident Set of the process: Portion of a process that is actually in main memory at any time.
- Ability to Have Partial Process in Memory has benefits:
  - Process not constrainted by Physical Memory.
  - Large virtual address space.
  - More processes can be loaded in Physical Memory.
  - Increase CPU Utilization and Throughput.
  - Less overhead with Swapping processes in/out of memory.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

Characteristics of Paging and Segmentation

- Improved System Utilization from:
  More processes may be maintained in main memory and process may be larger than all of main memory.

important

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

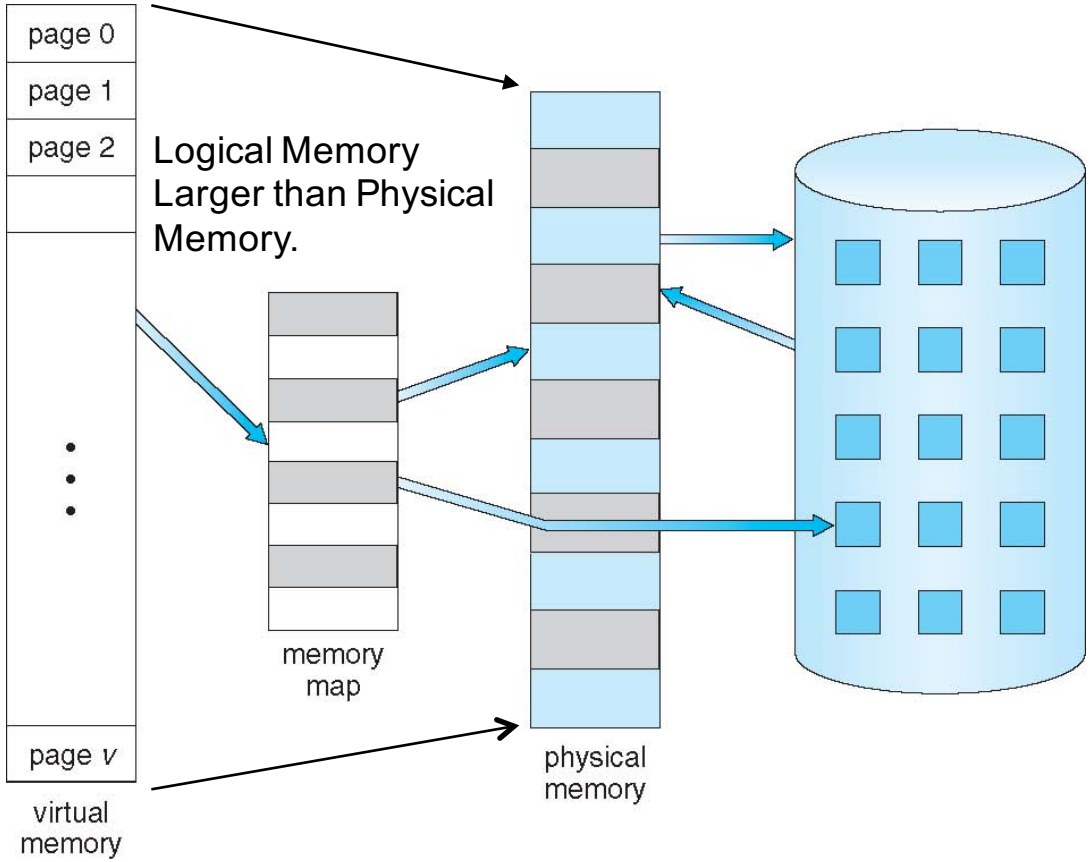
Copyright @ 2009 Pearson Education, Inc.

Table 8.1 Characteristics of Paging and Segmentation

	Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Physical Structure	Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Logical Structure	Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmented?	Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
External fragmented?	No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Required info per process?	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Unused Space?	Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Calculate abs address	Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All page/segment required in memory?	All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory frames for the process to run. Segments may be read in as needed
Swapping required?		Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Virtual Memory

- Virtual Memory
  - Separation of Logical Memory (as seen by users) from Physical Memory.
  - Large Virtual Memory maps into smaller physical memory.
  - Program does not worry about amount of physical memory available.



Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Virtual Memory
  - Shared Library Using Virtual Memory

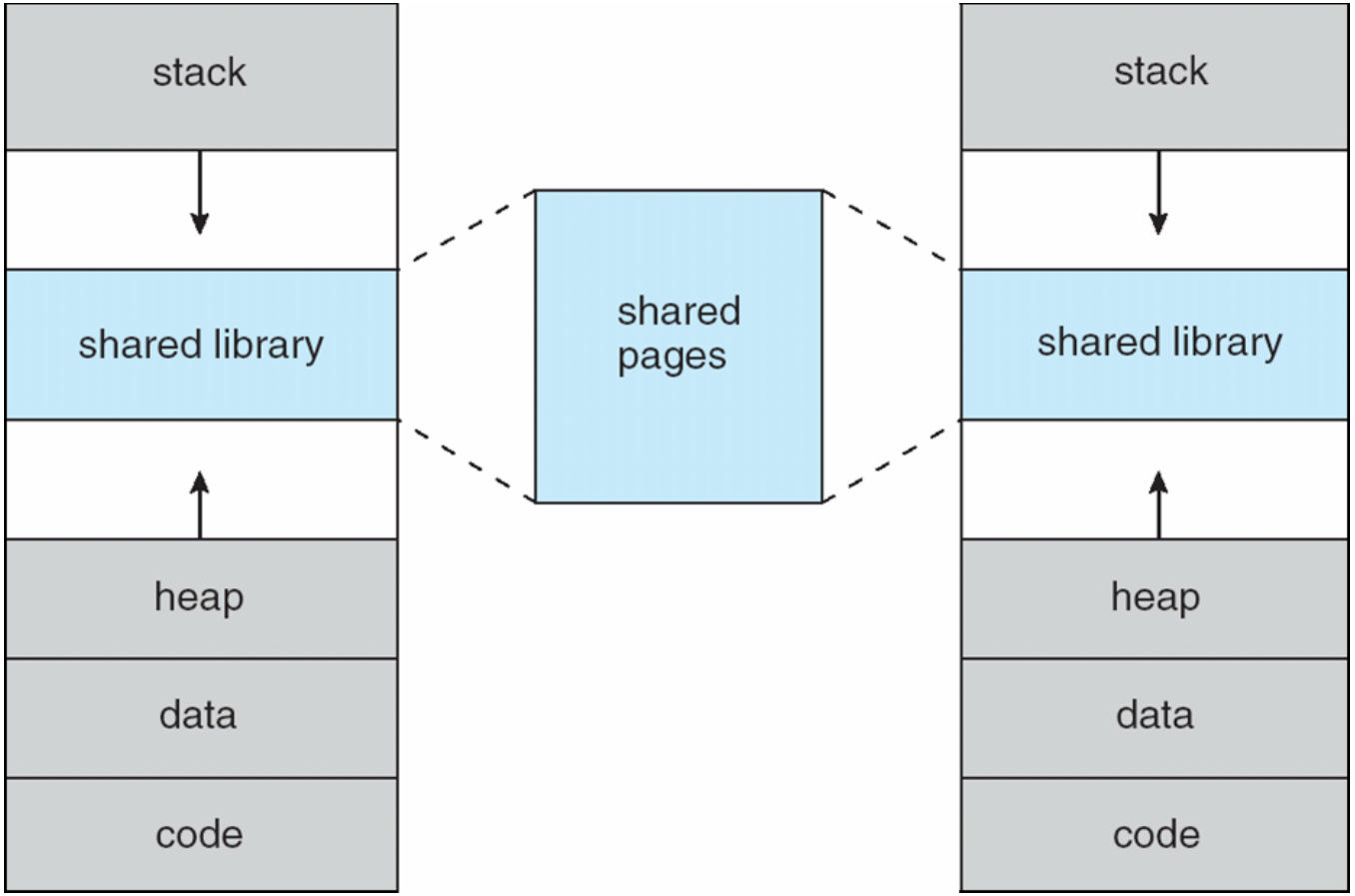
Mapping shared object into a Virtual Address Space of multiple processes.

System Libraries shared:

1. Actual physical pages of the library shared through the virtual address space of processes.
2. Library mapped read-only into space of each process.

Processes share memory:

1) One process creates region of memory shared with other processes.



Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Demand Paging

do not load until the program is executed.

  - Without Virtual Memory (Limitation):
    - All pages or segments pages of a process must be in Main Memory.
  - Demand Paging
    - Entire Program Not Needed in Memory, only load pages as needed.
    - Decreases swap time.
    - Decreases amount of memory needed (more processes in memory).
    - Faster response time.
  - Pager: Swaps In/Out individual pages of a process.
    - Lazy Swapper: Swaps page into memory when needed.
    - Start NO page in memory, load each page on demand (Page Fault).
    - Many page faults (during) initialization.
    - Locality Of Reference: Reasonable performance from demand paging.
      - Knuth estimated that 90% of a program spends time in 10% of the code
  - Hardware Support
    - Page Table Valid-Invalid Bit.
    - Valid: Page is in memory.
    - In-Valid: Page currently on swap device. Page Fault.

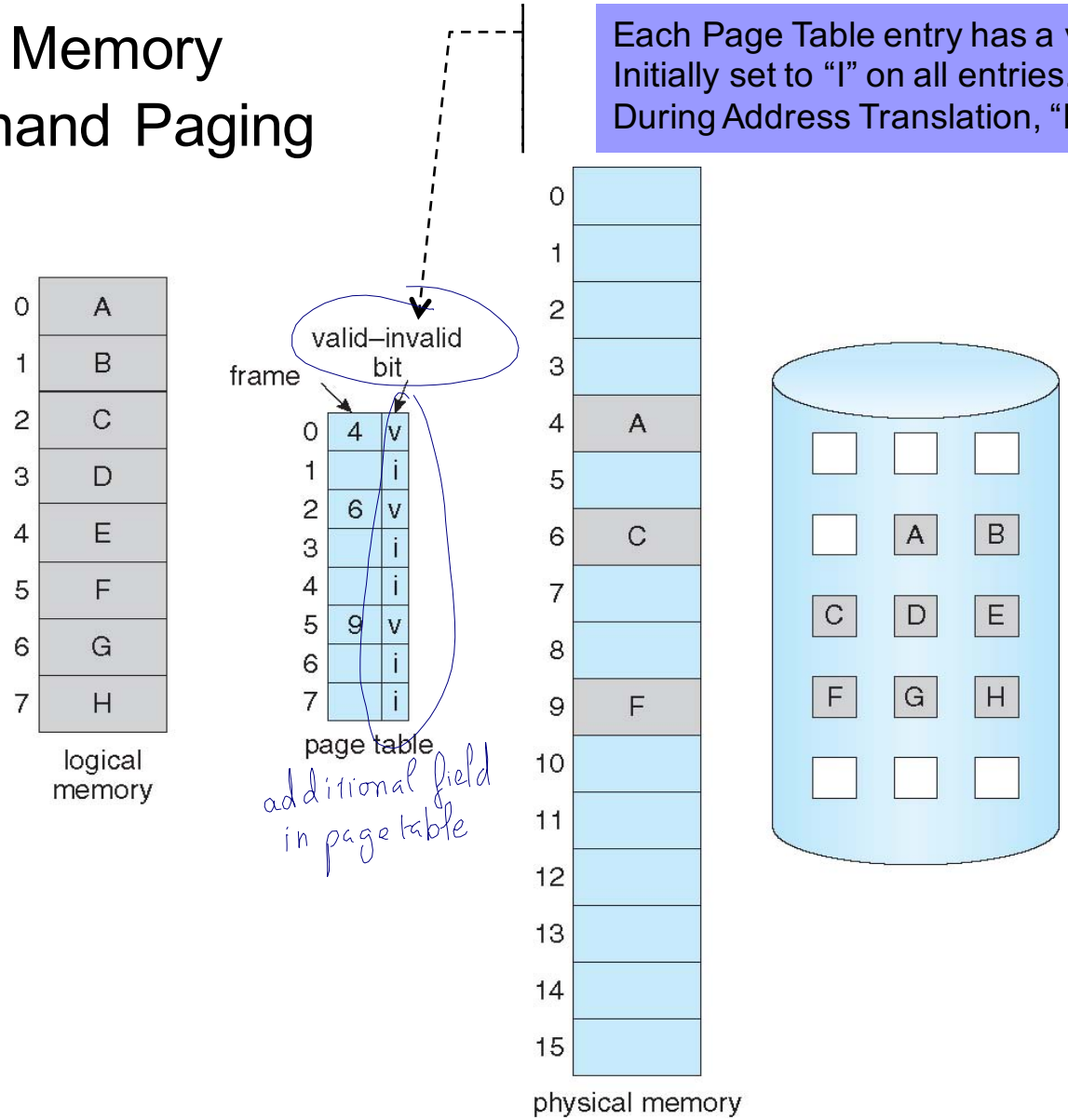
must have HW support feature. page table valid/invalid bit in page table

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Demand Paging

Each Page Table entry has a valid-invalid bit.
Initially set to "I" on all entries.
During Address Translation, "I" bit means Page Fault.



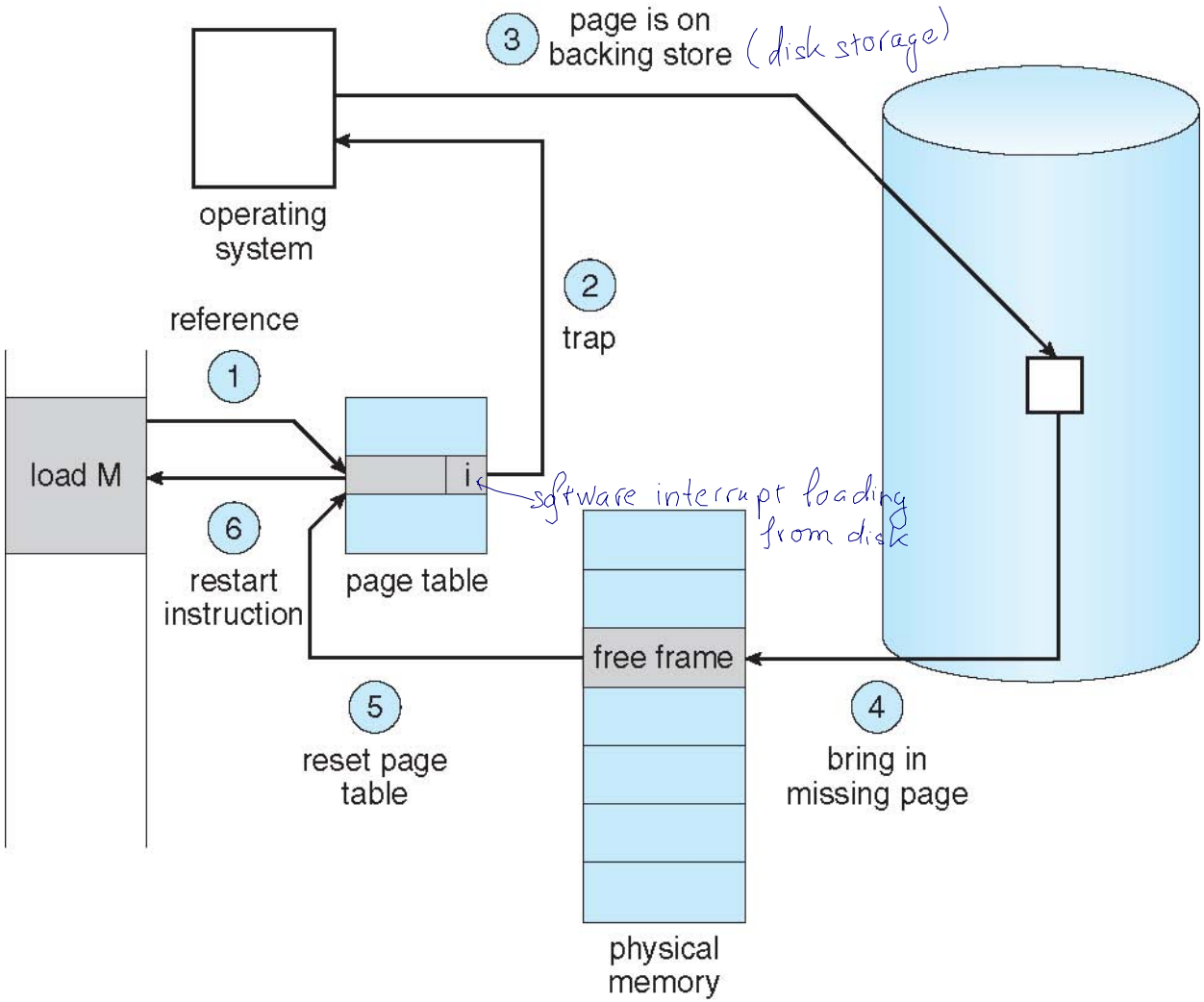
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Demand Paging

Handling Page Fault:

1) Check Page Table in PCB, "invalid" bit.
2) Reference swap in page.
3) Operating System Frame Table has free frame entry.
4) Schedule disk operation to swap in desired page into frame entry.
5) Modify Page Table in PCB, "valid" bit.
6) Restart instruction interrupted by the Page Fault trap. Process access the page.



Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

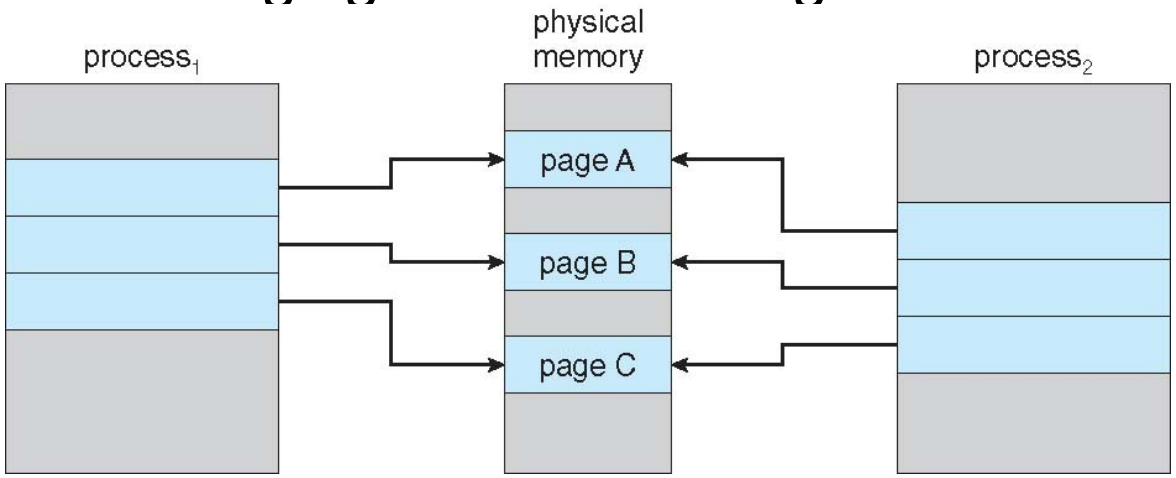
- Process Start: Demand Paging Instruction Page.
- Copy on Write

donot create new pages

<u>Fork( ) bypass Demand Paging with Page Sharing Technique.</u>

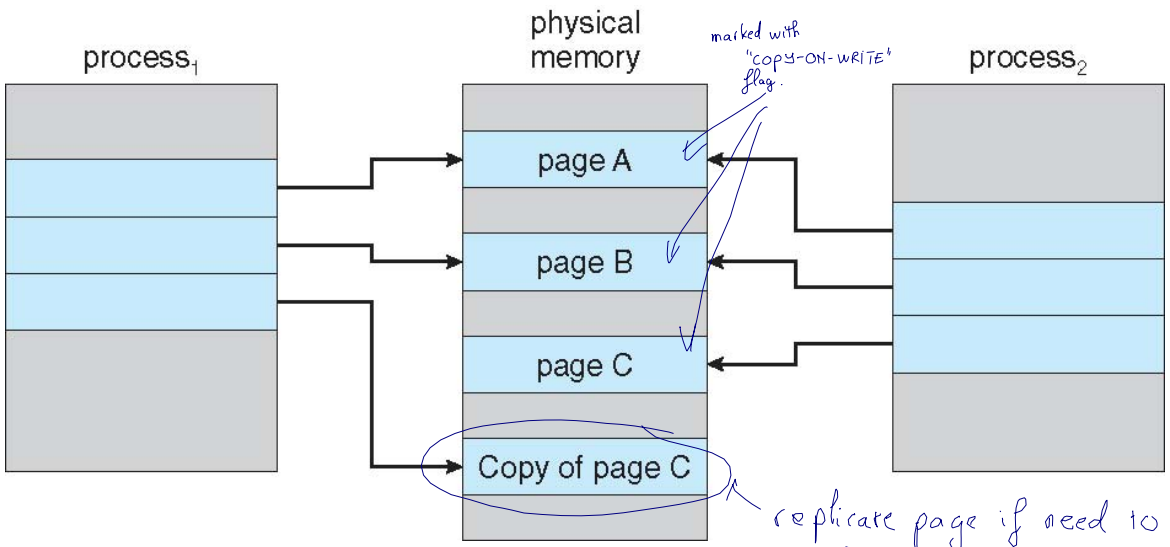
1) Parent and Child Process share same pages.

2) Shared Pages marked as Copy-On-Write in processes Page Table.



3) Process1 writes to Page C, copy of Page C created.

4) Process 1 and Process 2 does not modify each other's data.



replicate page if need to modify page content.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

the algorithm is to choose which page to be swapped out

- Page Replacement Algorithm
- Increase Multiprogramming by Over-Allocating Memory.
- Over-allocated Memory: Running processes with more pages than physical frames.
- How to recover from "No Free Frames"?
  - Swap out process: Reduces level of multiprogramming.
  - Page Replacement: Find frame not currently used and free it.
    - Write frame contents to swap space.
    - Change Page Table and Frame Table.
    - Use free frame for process with Page Fault.
    - Read desired page into free frame.
    - Change Page Table and Frame Table.
    - Requires two page transfers, doubles Page-Fault Service Time.
- Demand Paging Require Page-Replacement Algorithm and Frame-Allocation Algorithm

  to choose which free frame to use for storing new swapped in content

  to choose which pages to be swapped out
  - Completes separation of Logical Memory and Physical Memory.
  - Large Virtual Memory on a Smaller Physical Memory.

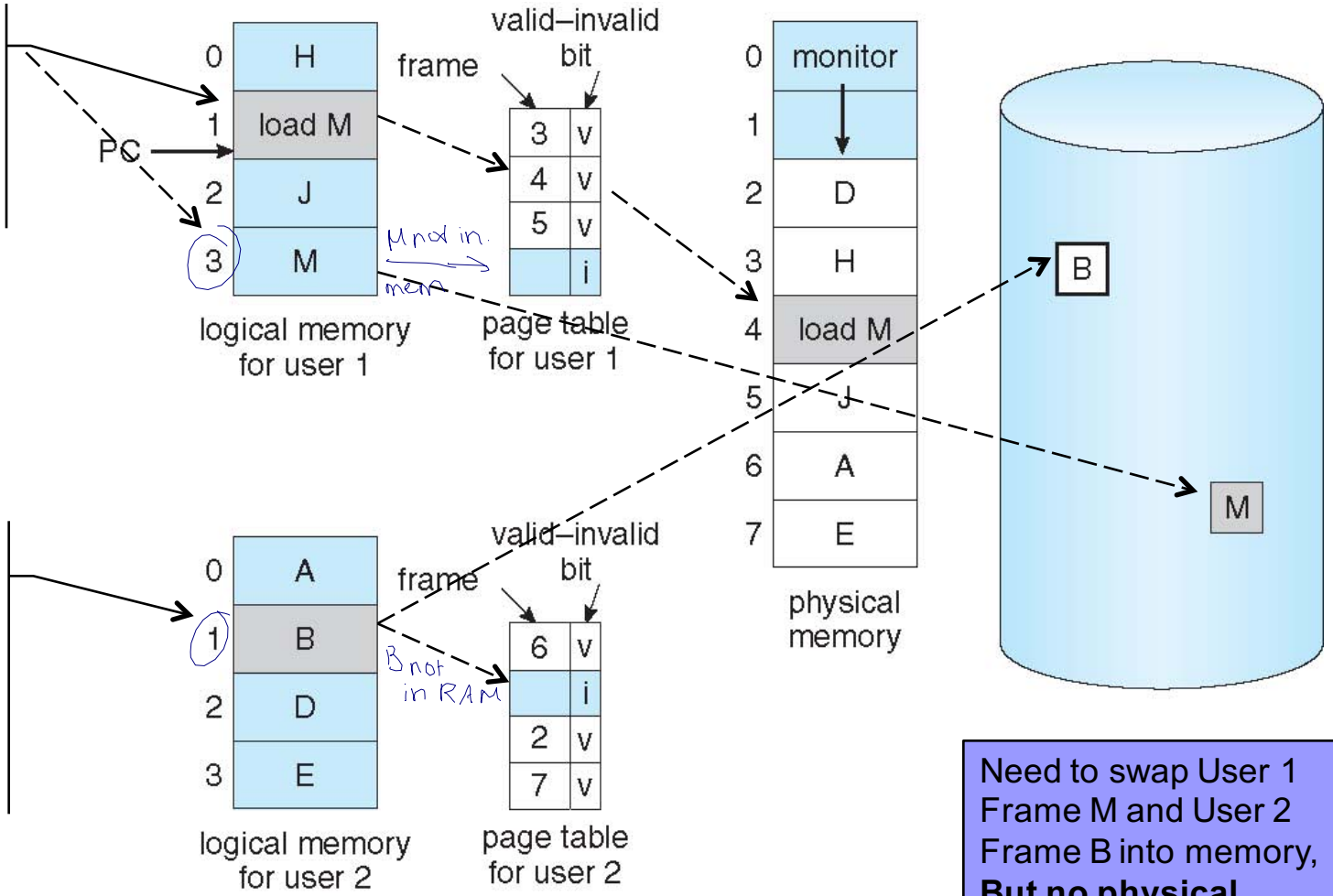
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

0

- Page Replacement Detection

User 1 executing "load M" requires Page 3 to be loaded. There is no free frames.



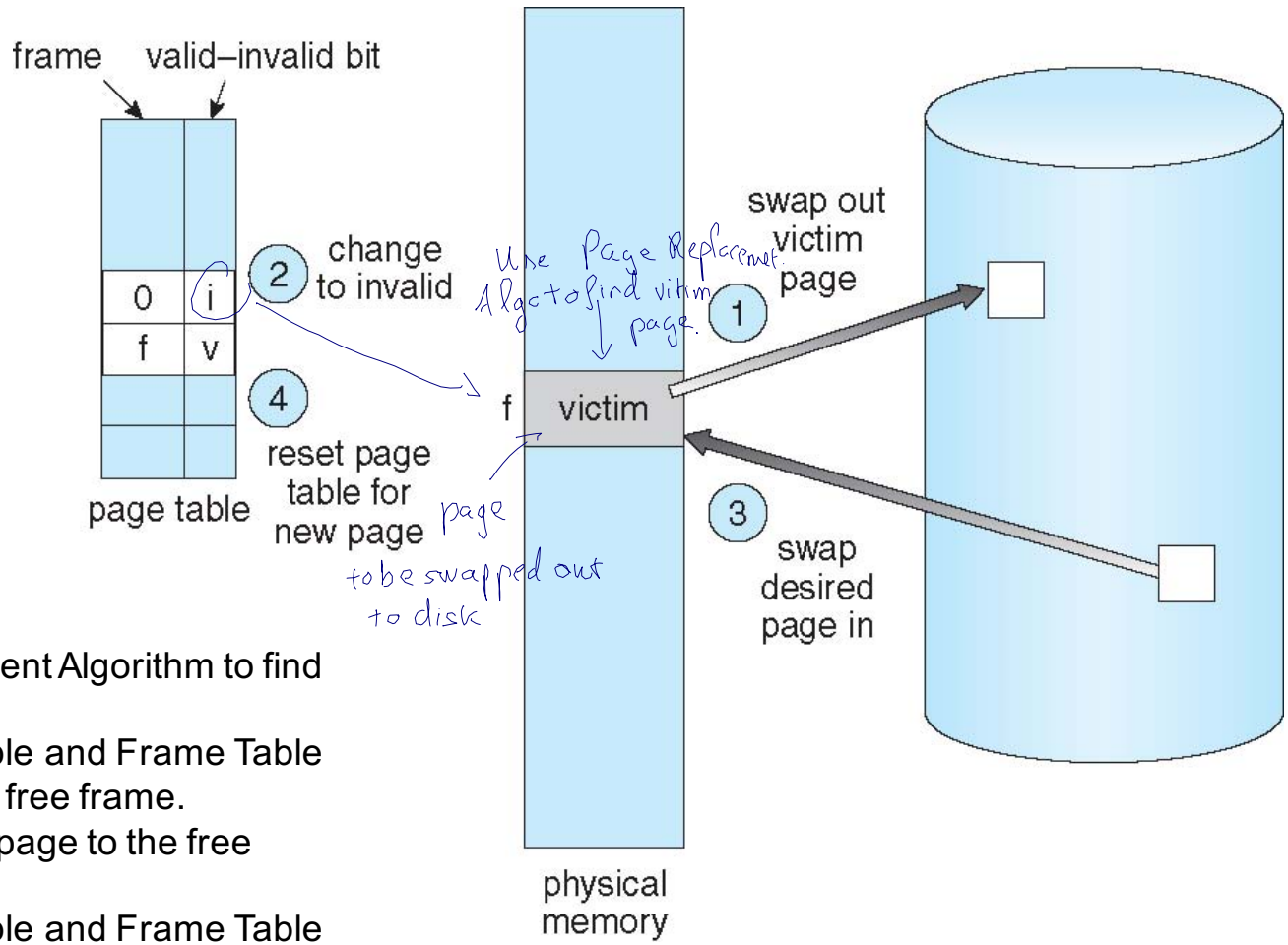
User 2 executing at Page 1. **<u>Page 1 is "Invalid" and needs to be loaded.</u>** There is no free frames.

Need to swap User 1 Frame M and User 2 Frame B into memory, **<u>But no physical memory available.</u>**

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Page Replacement



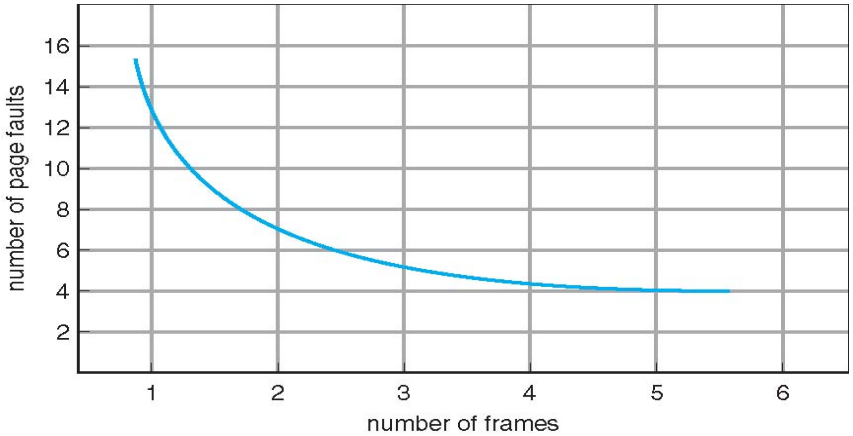
1) Use Page Replacement Algorithm to find a "victim" frame.
2) Update the Page Table and Frame Table to "Invalid". Create a free frame.
3) Swap in the desired page to the free frame.
4) Update the Page Table and Frame Table for the swapped in page.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

End on 12-June

- Page Replacement Algorithm
  - Disk I/O Expensive.
  - Page Reference String: Sequence of Page Requests.
  - More Page Frames (increase Physical Memory), Page Faults decreases.



  - FiFO Page Replacement
    - When a Page is replaced, the oldest Page is chosen.
    - Belady's Anomaly: On certain Page Reference String, Page-Fault rate may increase as the number of allocated Page Frames increase.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Page Replacement Algorithm

usually the worst case scenarior

FIFO Page-Replacement Algorithm

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
F	F	F	F		F	F	F	F	F	F			F	F			F	F	F

page fault rate = 15

3 Frames FIFO Queue

Fault occurs when page referenced.

Page Reference String.

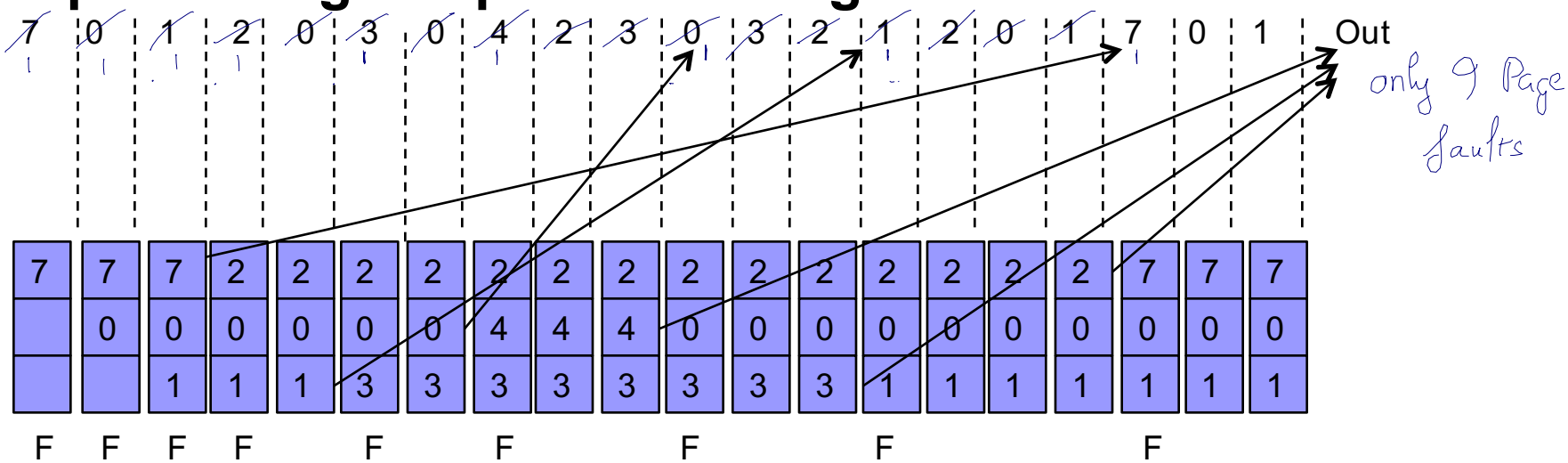
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

best case scenario (but not practical)

- Optimal Page Replacement Algorithm
  - **Of the frames in memory**, replace page that will not be used for the longest period of time.
  - Pages not referenced will be removed from the frames.
  - Lowest possible Page-Fault Rate for fixed number of frames.
  - Difficult to implement because future knowledge of Reference String required.
  - Mainly used for comparision studies.

Optimal Page-Replacement Algorithm



only 9 Page faults

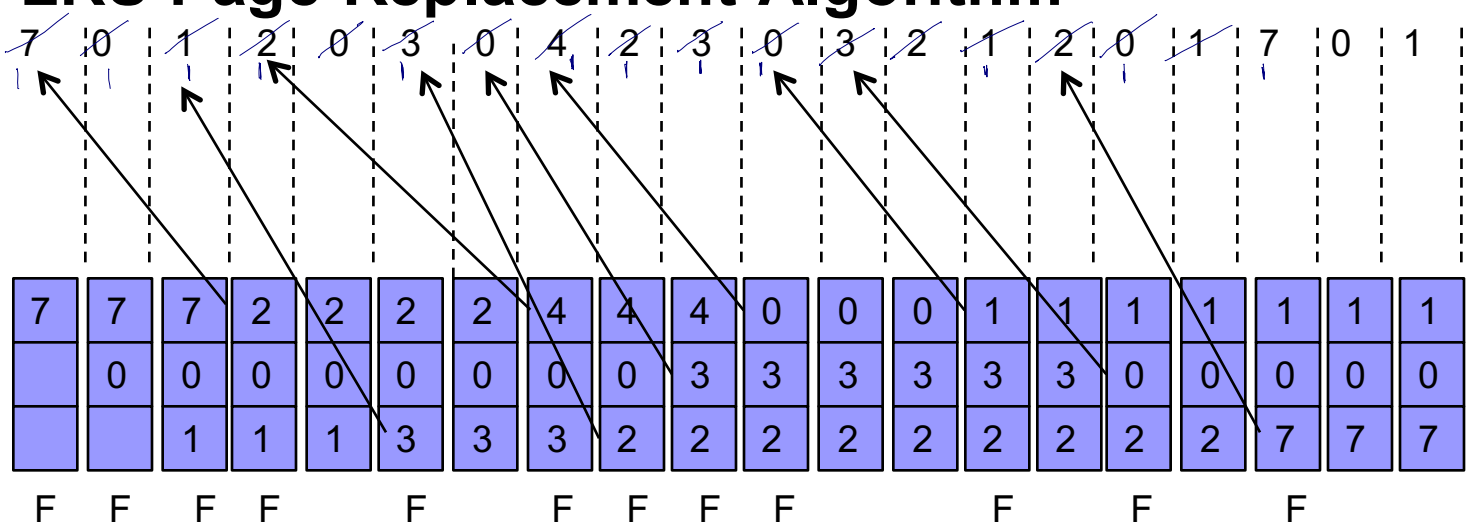
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- LRU Replacement Algorithm
- Use recent past as an approximation of the future use.
- **Of the frames in memory**, replace the page that has not been used for the longest period of time. ← similar to FIFO, except recently used page is re-prioritized & push to end of the Q (so it comes out last)
  - Time-of-Use field and counter for each Page Table entry.
  - Stack with most recently used on top and least recently used on the bottom.

LRU Page-Replacement Algorithm

page fault = 12



Copyright @ 2009 John Wiley & Sons Inc.

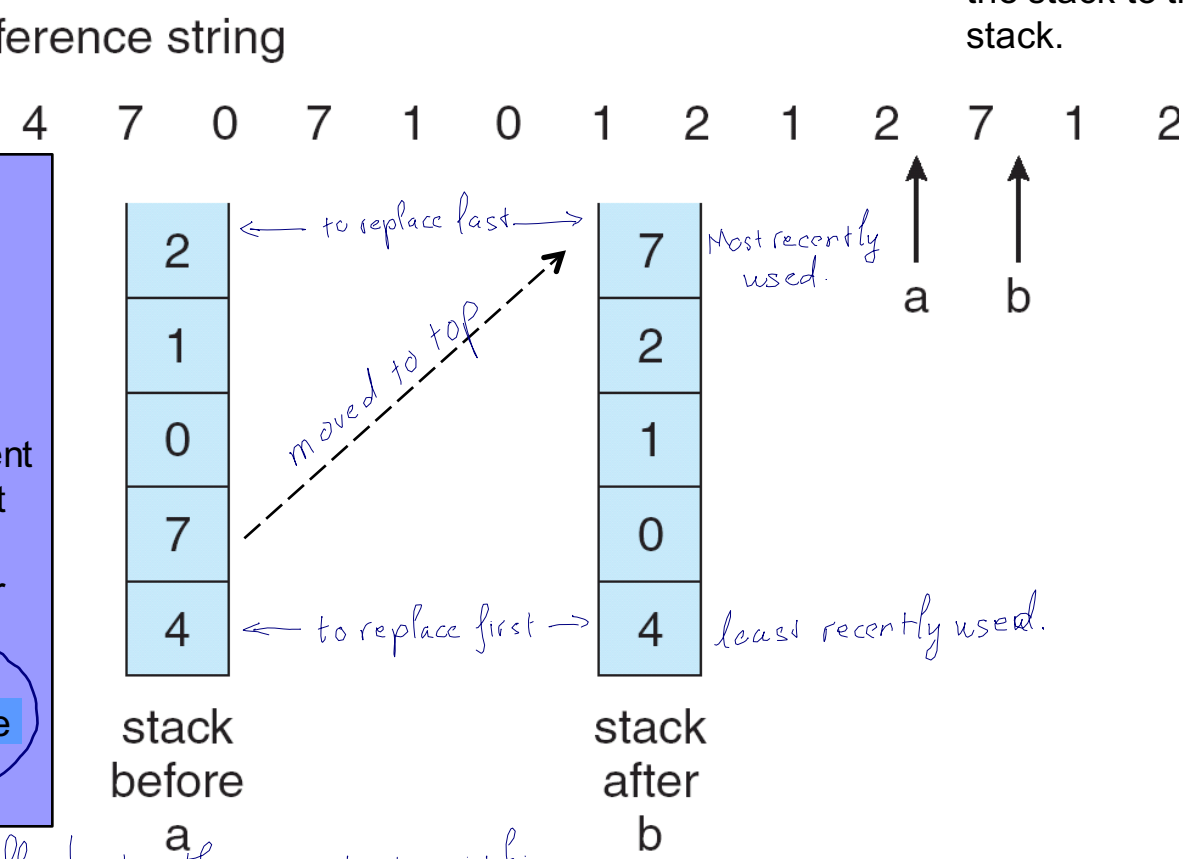
Virtual Memory

- LRU Replacement Algorithm
- Stack Implementation

Move 7 from its position in the stack to the top of the stack.

Stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on the top. Most recent is on top and least recent on bottom. -- Update done for every memory reference. Slow memory reference considerably.

but still better than context switching or swapping cost



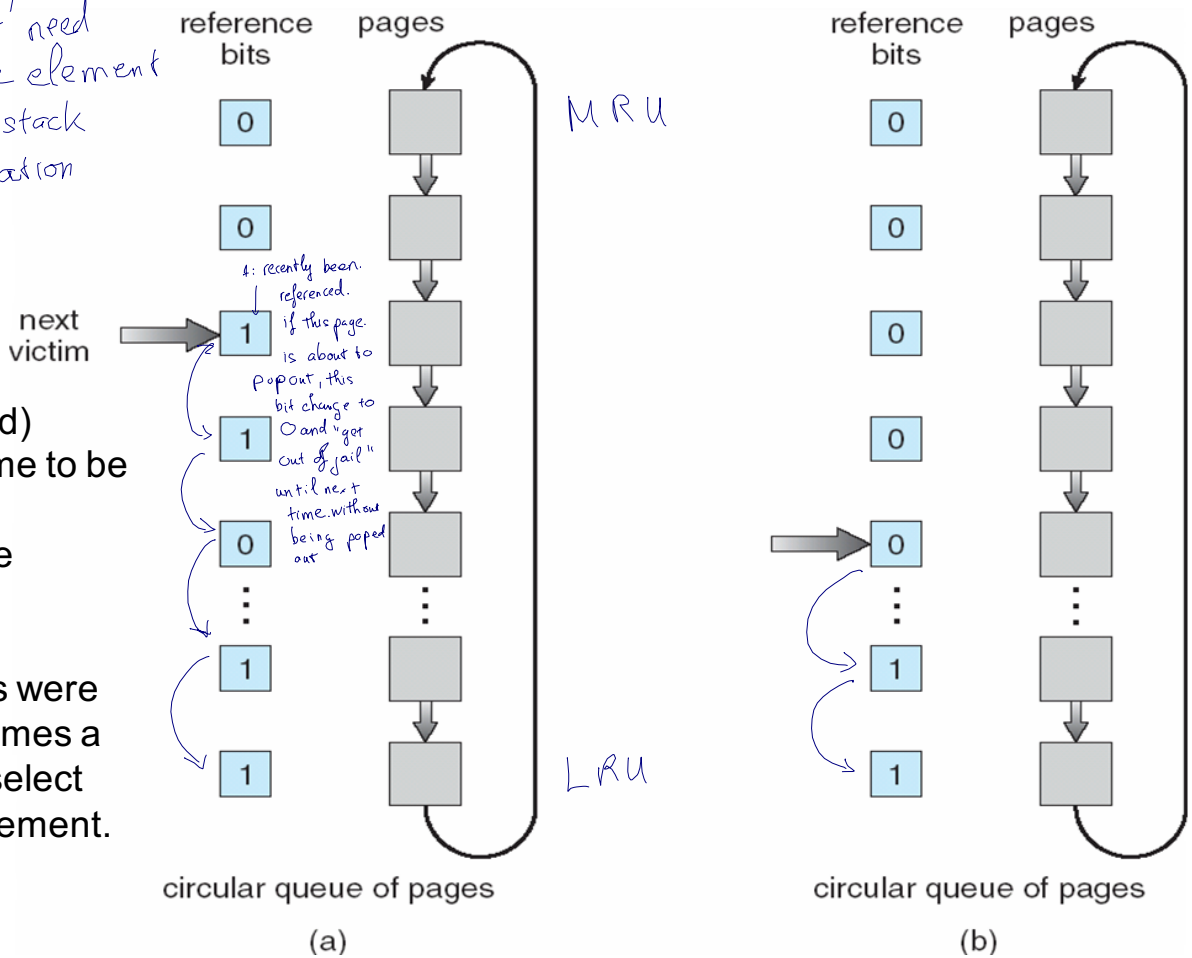
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

improvement of LRU

- LRU-Approximation Page Replacement Algorithm
  - Second-Chance (Clock) Page Replacement Algorithm

do not need to move element down as stack implementation



1) Pointer (Clock Hand) points to which frame to be replaced next.
2) Algorithm clears the Reference Bit after inspection.
3) If all Reference Bits were set, algorithm becomes a FIFO algorithm to select the page for replacement.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- LRU-Approximation Page Replacement Algorithm
- Enhanced Second-Chance Algorithm
- Reference Bit and Modified Bit Ordered Pair

consider "Mod" bit to make better decision if the frame has been used AND modified (highest priority).

  - Most likely to be replaced 1st → (0, 0): Neither recently used or modified. Replace
  - (0, 1): Not recently used, modified. If replace, has to write the page first.
  - (1, 0): Recently used, Not modified. Likely used again soon.
  - To be replace last → (1, 1): Recently used, modified: Likely used again soon, has to write the page first.
  - Circular queue scanned several times before select page.
  - Gives preference to page that has been modified, try to avoid replacing page that needs to be written.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- LRU-Approximation Page Replacement Algorithm
- Counting-Based Page Replacement Algorithm
  - Reference count for each page.
  - LFU Algorithm: Least Frequently Used.
    - Replace page with smallest count.
    - Use timer interrupt to right shift count.
  - MFU Algorithm: Most Frequently Used.
    - Replace page with largest count.
    - Small Reference count indicates page has just been loaded and will be used.

Comparision LRU v/s LFU:
Given a request of page ex: A, B, C, A, A, A, A, A, A, A, A, A, A, B, C, D

LRU:
[A]
[A, B]
[A, B, C]
[B, C, A] (for the next 11 As)
[C, A, B]
[A, B, C]
[B, C, D] ← A is evicted!

LFU:
A - 12
B - 2
C - 2
D - 1
} the eviction algo uses a counting table to identify lowest counting as the next victim for eviction.

- Page-Buffering Algorithm
  - Pool of free frames.
    - Page is reloaded from the free frame pool, if frame has not been modified.
    - VAX/VMS uses in combination with FIFO Replace Algorithm.
    - Some versions of UNIX system uses in conjunction with Second-Chance Replacement Algorithm.

add 1 or 2 bits & a counter to the page replacement mechanism to identify next page to be replaced.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

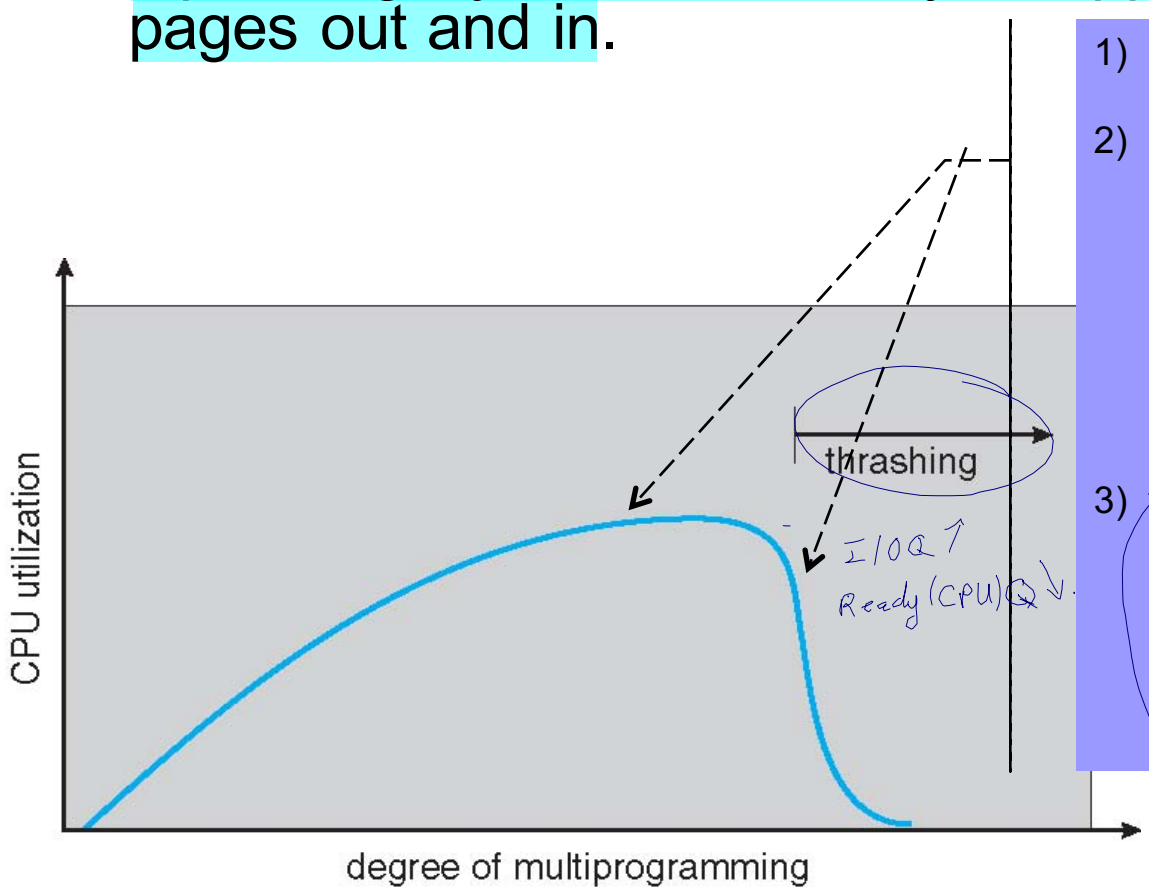
- Allocation of Frames
- Global Versus Local Frame Allocation
- Global Page Replacement Selection
  - Process select from the set of ALL frames, including other processes.
  - Allocation scheme allow high-priority process to select frames from low-priority processes.
- Local Page Replacement Selection ← prevent thrashing.
  - Process select ONLY from its own set of allocated frames.
  - Tend to hinder throughput, since other processes cannot use any unused frames of processes. ← does not retrieve frames cross processes.
  - Equal Allocation: Divide frames equally between processes.
  - Proportional Allocation: According to size of process priority.
- Memory Locality Issue:
  - Non-Uniform Memory Access

    there can be 2 or more memory components.
    - Memory access times vary because memory subsystem located on different boards.
    - Operating System tries to allocate frames close ~~the~~ to the CPU where the process is scheduled (improves cache hits and memory access times).

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Thrashing ← by product of Global Page Replacement.
- Excessive Paging Activity in a Process, causing by the Operating System continually swapping the Process' pages out and in.



1) As degree of multiprogramming increases, the CPU utilization increases.
2) At some point, the thrashing sets in, CPU utilization drops sharply. There are not enough physical frames to satisfy all processes paging request. The Operating System starts to swap pages out from running systems to allow processes to swap pages in to continue to execute.
3) Operating System scheduler attempts to increase the degree of multiprogramming, by scheduling more processes, which causes more page faults and longer queue for the paging device.

Processes gets placed in I/O queue & the Ready Q is empty. Everything is slow down waiting for I/O.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Thrashing
- Limit effects of thrashing:
  - Use Local Replacement Algorithm or Priority Replacement Algorithm.
    - Process can only access pages from its own set of allocated frames.
  - Use process execution Locality model.
    - Locality is the set of pages process actively use together.
    - Memory reference structure made of instructions of function calls, its local data structure, local variables, and subset of global variables.
    - This patterned data access is principle behind caching.
    - Process composed of several different localities (each function call).
    - Allocate enough frames to a process to accommodate its current Locality.
  - Process Working-Set Window
    - The set of pages (delta) in the most recent page reference is the working set.
    - Delta is the approximation of the program's locality.
    - Delta is too small, it will not encompass the entire locality.
    - Delta is too large, it may overlap several localities.
    - Once Delta is selected, process is allocated enough frames for its WS.
    - Approximate Working-Set model: Use fixed-interval timer and Page Table Reference Bit.
    - Suspend process:
      - Thrashing Occurs when: **Σ working sets > total memory size**
      - Frames allocated to other processes, optimizing CPU usage.

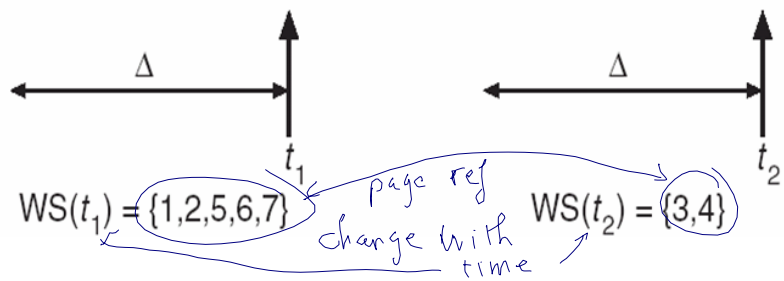
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Thrashing
- Working Set Model

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .



Δ = 10 Memory References

WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)

$D = \Sigma\ WSS_i$ ≡ total demand frames ← sum of all working set.

M = total number of available frames

with working set, thrashing can happen locally first with minimal effect (only to its process). The OS will consider to add more free frames to that working set to reduce thrashing w/o impacting other W.S.

If D > M → Thrashing

Policy: If D > M, then suspend process.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

Locality v/s Working Set ? Equivalent

- Working Set Model
  - Working Set of a Process changes over time as references to data and code section moves from one locality to another.

1) Peak in Page Fault Rate begin Demand-Paging in new Locality.

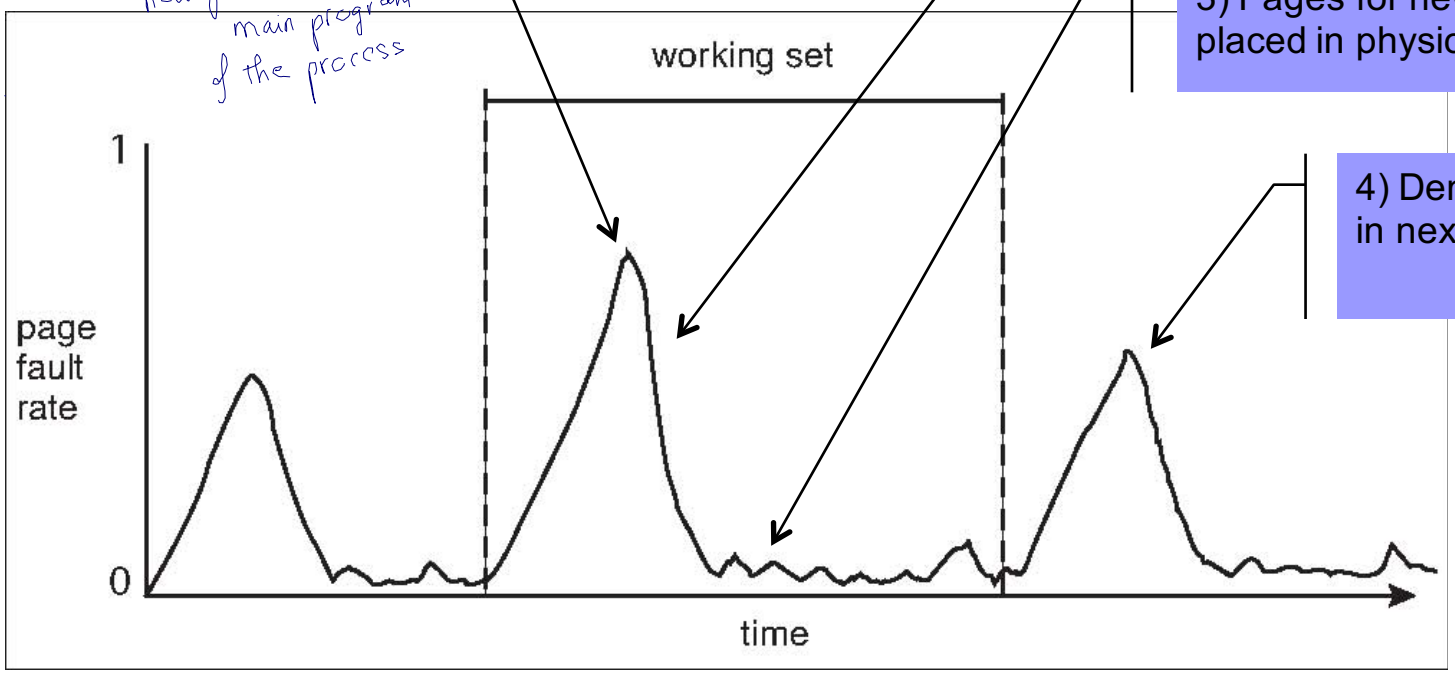
loading data from disk to memory

new function, var, main program of the process

2) Once Working Set of new Locality is in memory, Page Fault Rate falls.

3) Pages for new Locality placed in physical frames.

4) Demand Paging in next Locality.



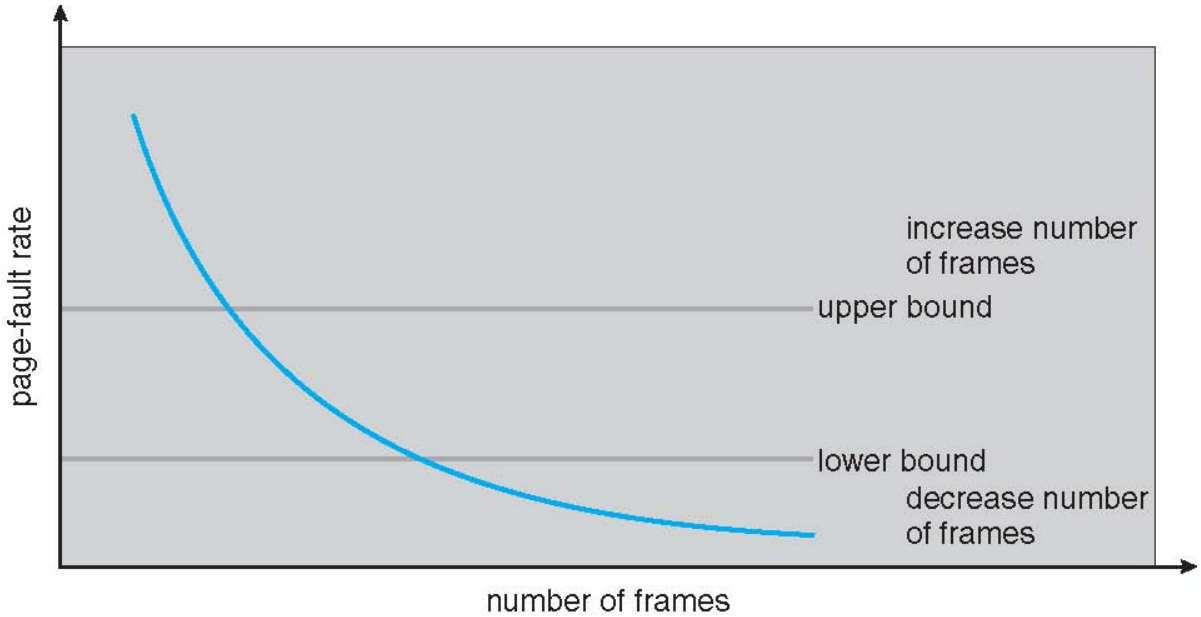
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Thrashing
- Page-Fault Frequency (PFF)
- Control Page-Fault Rate:
  - If Process Page-Fault Rate High, allocate more frames to process.
  - If Process Page-Fault Rate Low, remove allocated frames.
  - Suspend process when Page-Fault Rate increases and no frames are available.

provide more memory

reduce memory allocated to the process



Copyright @ 2009 John Wiley & Sons Inc.

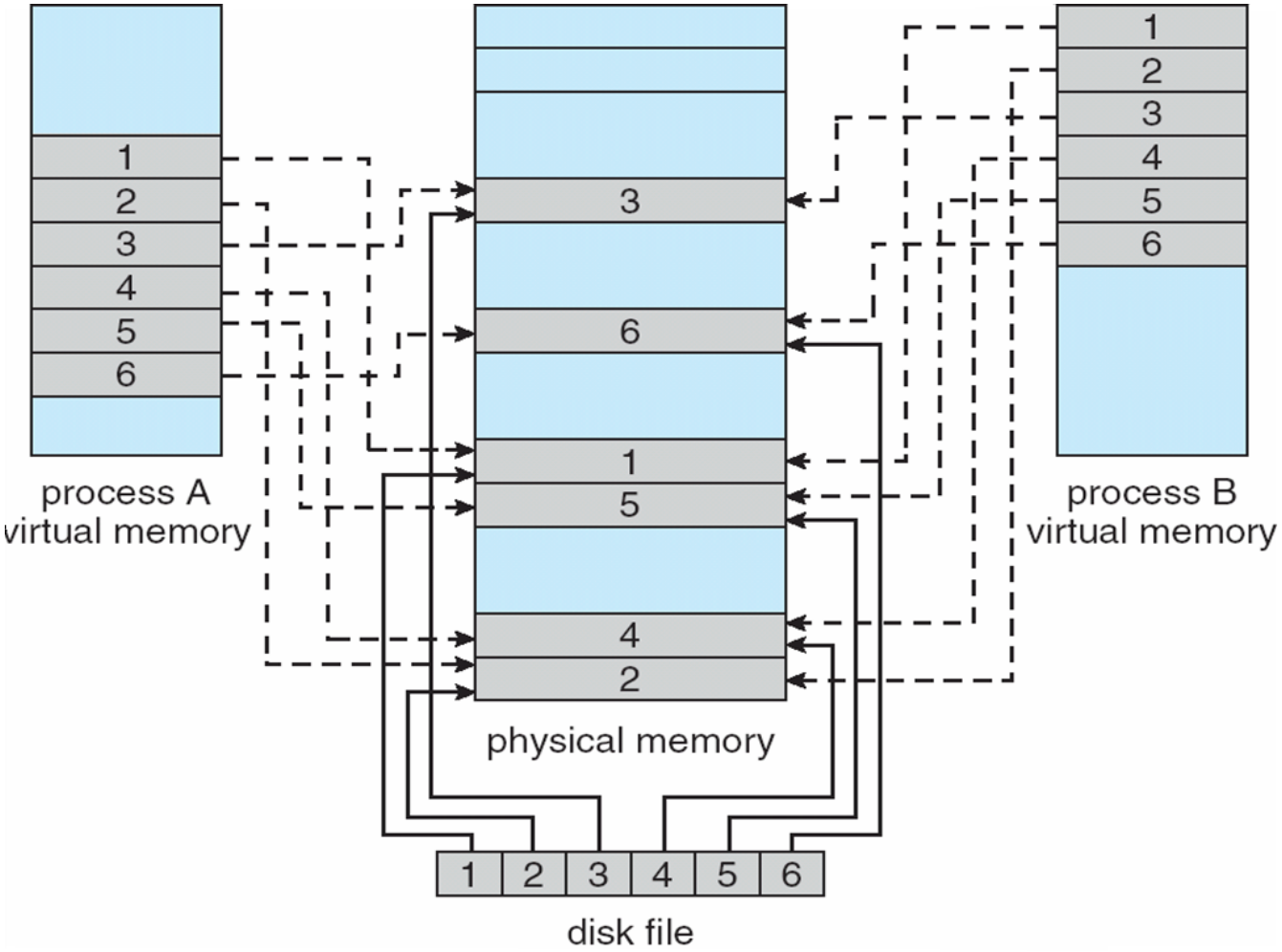
Virtual Memory

- Memory-Mapped Files
- File I/O Treated as Routine Memory Access.
  - Virtual address space logically associated with the file.
  - Map disk block to a page in memory.
  - Page sized portion of file is read from file system into physical page, using ordinary memory access to access file.
  - Multiple processes can map the same file concurrently, allowing sharing of data.
  - Memory Mapping UNIX and Linux:
    - mmap ( ): system call.
  - Memory Mapping Win32 API:
    - CreateFile( ): Opens file to be mapped.
    - CreateFileMapping ( ): Shared-memory object.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Memory-Mapped Files



Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Memory-Mapped Files
- Memory-Mapped I/O
- I/O Controller contains registers to hold commands and data being transferred.
  - Special I/O instructions transfers data between registers and system memory.
  - Range of memory addresses reserved and mapped to the device registers.
- Serial and Parallel Ports
  - CPU transfers data by read/write device data/control registers (I/O ports).
  - Device Ready:
    - Programmed I/O: CPU poll control register for device Ready.
    - Interrupt driven: Interrupt handler receive interrupt when device is Ready.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Allocating Kernel Memory
- User mode requests for free page frames allocated by list maintained by Kernel (FIFO, Optimal, LRU, LRU-Approximation Page Replacement).
- Kernel memory allocated from different list:
  - Minimize internal fragmentation. Data structures less than page.
  - Hardware devices interact directly with physical memory, require memory residing in contiguous physical pages.
- Buddy System for managing Kernel Memory.
  - Allocates memory from fixed-size segment of physically contiguous pages.
  - Power-of-2 Allocator: Segments allocated in units sized as power of 2 (4 KB, 8 KB, 16 KB, 32 KB, …).
  - Each segment split into two "buddies" of equal sizes, until segment size (round up to next highest power of 2) can hold the memory request.
  - Coalescing: When segment released, coalesced with adjacent "buddy" (if also free) into larger segment.

have to be contiguous.
- is not managed by MMU
- fixed memory.

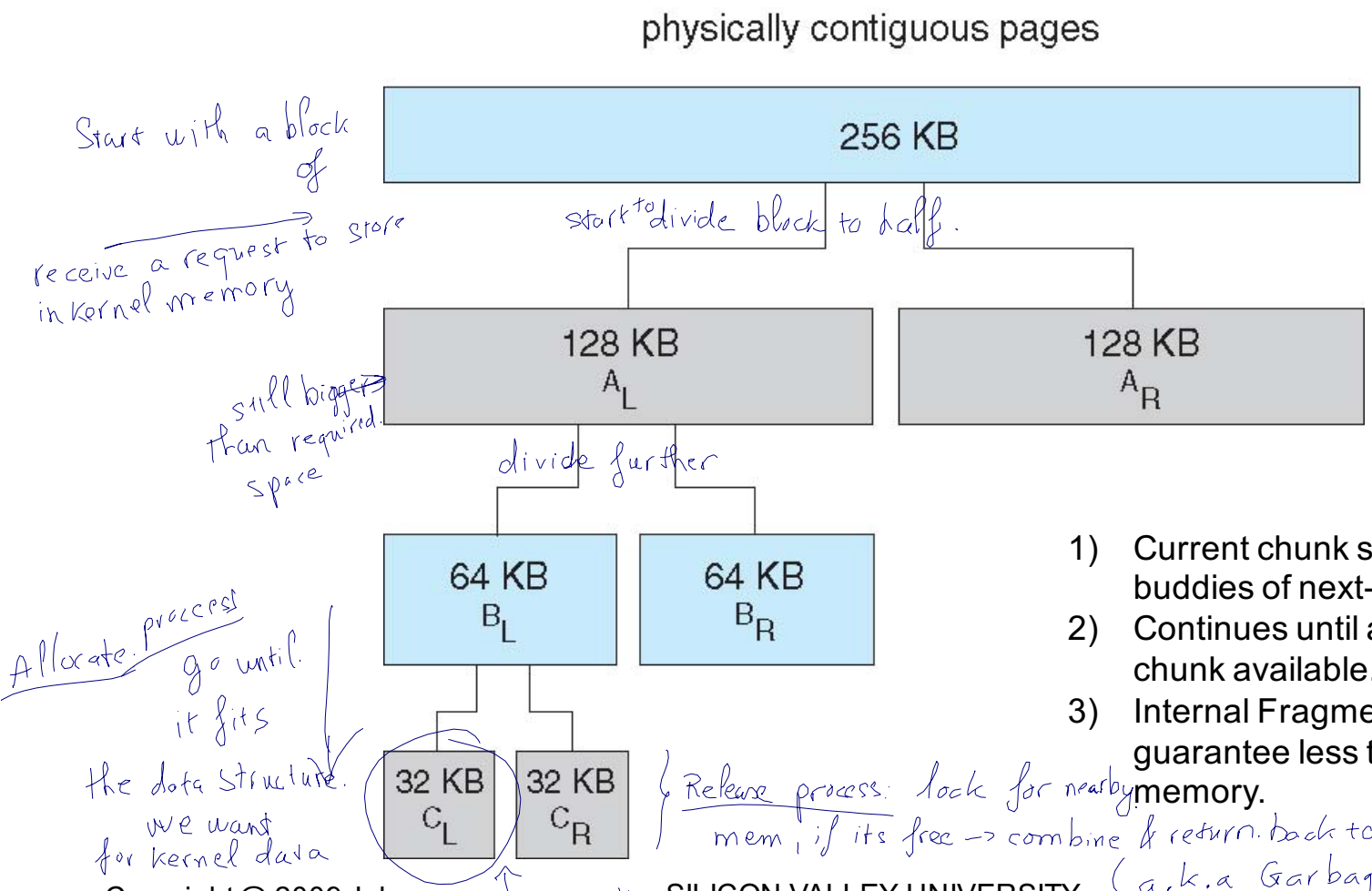
first way to organize the kernel mem hierarchy.

[merge]

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Allocating Kernel Memory
- Buddy System ← divide memory into smaller chunks next to each other



1) Current chunk split into two buddies of next-lower power of 2.
2) Continues until appropriate size chunk available.
3) Internal Fragmentation cannot guarantee less than 50% unused memory.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Allocating Kernel Memory
- Slab Allocator System for managing Kernel Memroy

2nd technique for managing kernel mem

Memory

  - Slab is one or more physically contiguous pages.
  - Cache is assigned to one or more slabs.

Pros: Data structure has been pre-defined.
=> no complicated allocation
=> faster processing.

    - Single cache for each unique kernel data structure.
    - Cache is filled with instantiations of the kernel data structure.
    - A cache can be filled with process descriptor data structures or semaphore data structures or file object data structures.
    - Cache objects does not have to be contiguous, but taken from assigned slab.
    - Cache objects marked as FREE when cache is created and object released.
    - Cache object marked as USED when cache object allocated from the slab for the requested data structure.
    - If slab full of USED objects, object allocated ~~from~~ into empty slab.
    - If no empty slabs are available, new slab is allocated from continguous physical pages and the slab is assigned to the cache. The Object is allocated from the slab.

- Slab States
  - Full: All objects in the slab marked as USED.
  - Empty: All objects in the slab marked as FREE.
  - Partial: Both USED and FREE objects in the slab.

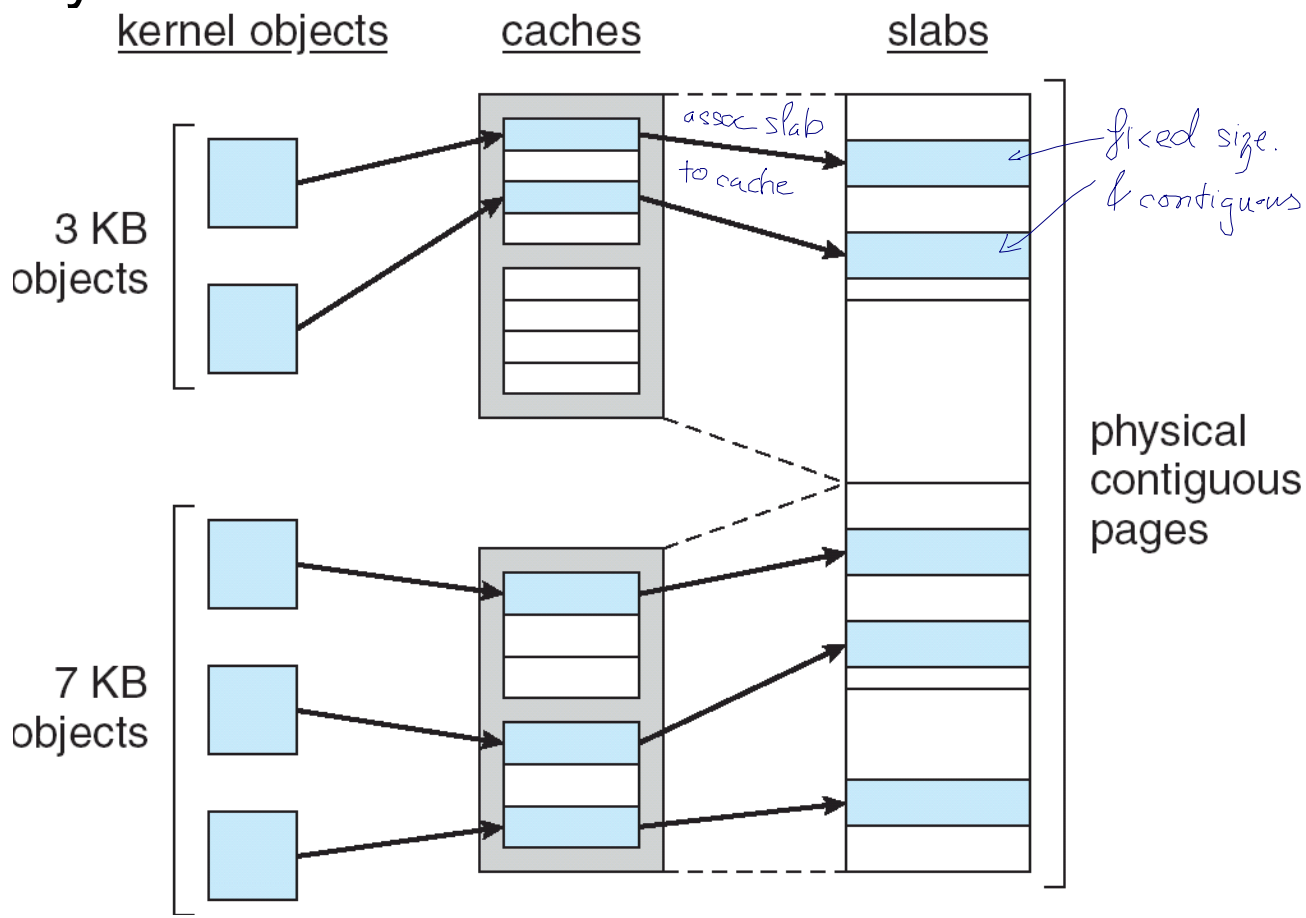
Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Allocating Kernel Memory
- Slab Allocator System

Benefits:

1) No memory wasted due to fragmentation. Cache ~~assigned to slab~~ slab ~~that~~ is divided up into chunks the size of the cache object.
2) Memory request satisfied quickly. Efficient when objects frequently allocated and deallocated. Since **objects created in advance**, objects can be quickly allocated.



Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

Locality ≈ working set

- Other Paging System Issues
- Prepaging ← prevent slow down when process initializes: preallocate some free frame b4 the process starts.
  - Large number of page faults occur during process initialization and when process swapped-in.
  - Using the Working Set model (Locality), list of pages in the Working Set can be saved and when process swapped-in, the Working Set pages are preloaded. ← preloading locality b4 the process awakes
  - Risk of preloaded pages not used (I/O and memory wasted).
- Page Size (small v/s large).
  - Page Table increases with smaller Page size.
  - Internal Fragmentation greater with larger Page size.
    - On average, half of the process final page is unused.
  - I/O Overhead (Seek, Latency, and Transfer times). ← large page size has lower overhead than multiple smaller page size.
    - Seek and Latency overhead dwarfs Transfer time.
    - Large Page size needs less I/O.
  - Locality best matched with smaller Page size.
    - Smaller Page size will match program locality more accurately, resulting in less total allocated memory.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Other Paging System Issues
- Program Structure
  - Selection of data structures and programming structures can increase Locality and reduce Page-Fault Rate and reduce number of pages needed in the Working Set.
    - Stack has good Locality since access made at the top.
    - Hash Table has bad Locality since references are scattered.
      - Compensate with better search speed, and less memory references.
    - C and C++ uses pointers and randomize memory references, causing bad Locality.
      - C++ or object-oriented programs tend to have bad Locality.
    - Compiler and Loader can have affect on paging.
      - Separating code and data can have code in read-only pages.
      - Clean pages (not modified) does not have to be written before paged out.
      - Routines with interactions can be placed in one page.
      - Loader avoid placing routines across page boundaries.

the method or data structure. can be outside of the Locality => bad for optimizing WS as it needs to read pages from outside of its Locality

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Operating System Examples
- Windows XP
- Virtual Memory

win x P Virtual mem uses Demand Paging + Clustering (load pages surrounding the faulting page)

  - Demand Paging with Clustering
    - Clustering loads pages surrounding the faulting page.
  - Process assigned Working Set Minimum and Working Set Maximum. ← each process has min & max no. of working set pages
    - Working Set Minimum: Guaranteed minimum number of pages. per process
    - Working Set Maximum: Max assigned pages. per process
  - Virtual Memory Manager maintains list of free page frames.
    - Processes are allocated page frames from the list, until the process has reached its Working Set Maximum.
    - The process will select a page for replacement using Local Page Replacement Algorithm.
    - Uses variation of FIFO algorithm.
  - Virtual Memory Manager manages free memory threshold: Uses Automatic Working Set Trimming to removes pages from processes until the process is at its Working Set Minimum.

Copyright @ 2009 John Wiley & Sons Inc.

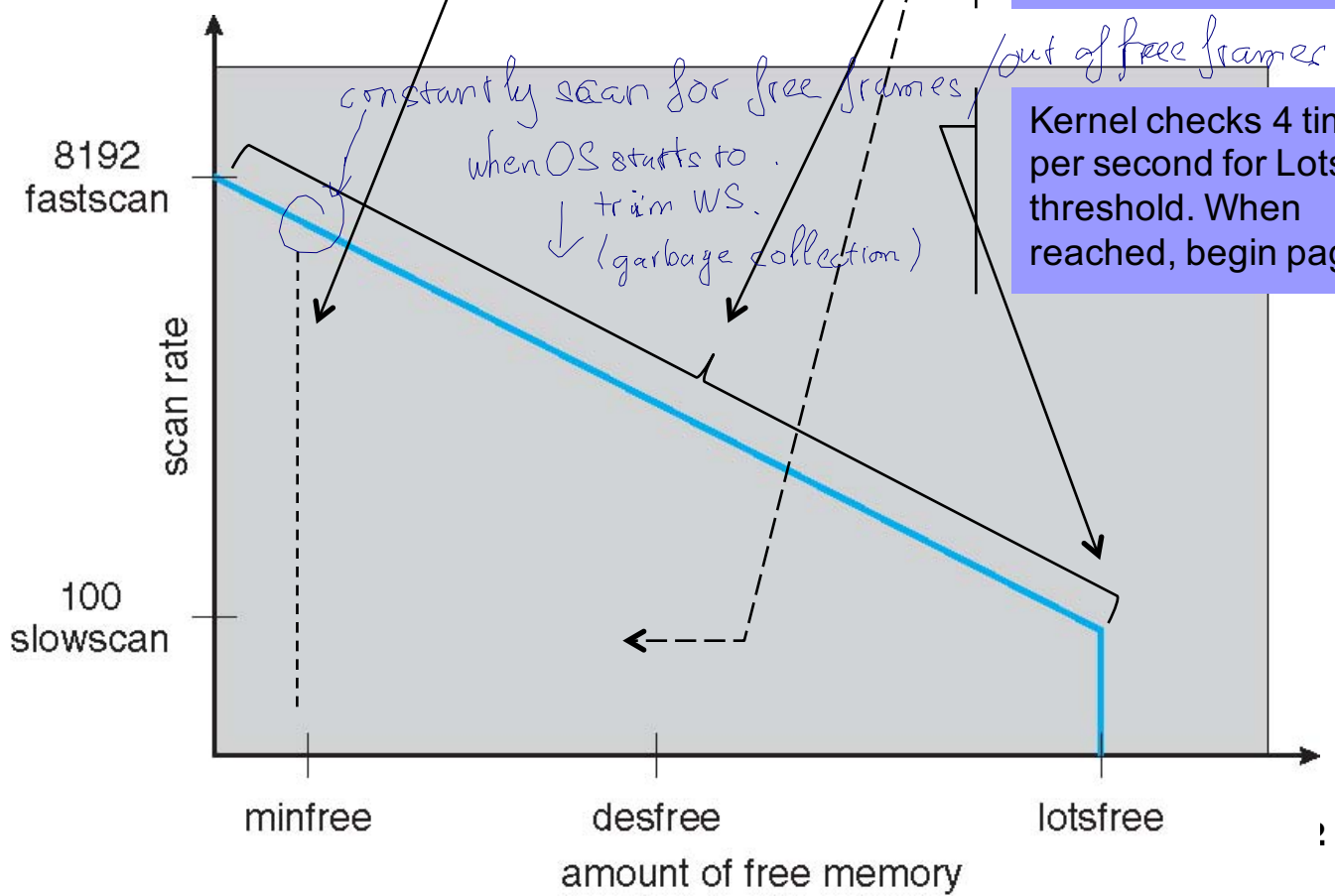
Virtual Memory

- Operating System Examples
- Solaris
- Maintains list of free pages.

If system unable to maintain amount of free memory at minfree, the pageout process is called for every request for a new page.

As the amount of free memory decreases, the scan rate increases.

Lotsfree: Pageout process starts using Second-change Page Replacement Algorithm. Scan rate increases as amount of free memory decreases.
Cache list of pages that have been paged out, but can be reclaimed.



Kernel checks 4 times per second for Lotsfree threshold. When reached, begin paging.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Operating System Examples
- Solaris
- Maintains list of free pages to assign faulting processes.
  - Kernel checks 4 times per second for lotsfree threshold.
  - Lotsfree: Threshold parameter (1/64 physical memory) to begin paging.

    if free memory below "lotsfree", → start paging out

    - Pageout process starts: Second-chance Page Replacement algorithm.
    - Pageout process ~~scans~~ starts at "slowscan", 100 pages per second, and scan rate increases as amount of free memory decreases.
    - Cache list of pages that have been paged out, but can be reclaimed.
  - Destfree: Threshold parameter

    page out rates

    - Pageout process runs 100 times per second.
    - Kernel swapping processes, freeing all pages allocated to swapped processes.
  - Minfree: Threshold parameter

    free memory (pages) reach super low, ⇒ switch scan rate to max at 8192 pages/sec

    - Pageout process called for every request for a new page.
    - Pageout process scans max at "fastscan" or 8192 pages per second.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

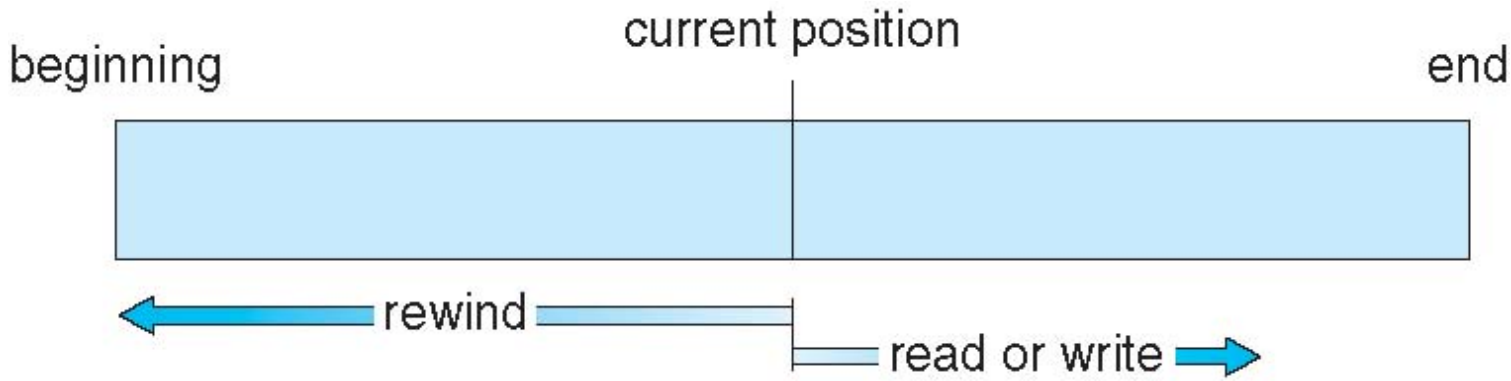
- File Concept
  - Operating System abstracts from physical properties of its storage device to define a logical storage unit, the file File System:
    - User's Perspective: File is the smallest allotment of secondary storage (data cannot be written to storage unless its in a file).
    - Collection of Files storing related data.
      - Programs: source, object forms.
      - Data: alphanumeric, binary, video, audio.
      - Free formed (text file) or structured (records).
    - File Type defines its structure:
      - Text file is a sequence of characters.
      - Source programming language file is sequence of subroutines and functions.
      - Object file is sequence of bytes in blocks understandable by the linker.
      - Executable file is series of code sections used by loader to execute.
    - Directory Structure organizing and providing information about all files in the system.
  - Operating System provides a uniform logical view of information storage.

the OS only sees disk as a sequence of bytes, only user defines the files

Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Access Methods
  - Files store information: The information in the file must be accessed and read into memory.
  - Simplest access method: Sequential Access ← Method 1.
    - Information in the file is processed in order (one record after another).
    - Read Next: Reads next portion of the file.
    - Write Next: Appends to the end of the file.
    - Skip forward or backward "n" records.
    - Automatic adjustment of File Pointer.



Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Access Methods
  - Direct Access (or Relative Access) ← Method 2 :
  - Fixed length Logical Records
    - Read/write records in no particular order.
    - Numbered sequence of blocks or records with no restrictions for Read/Write.
    - Databases: Query retrieves information by blocks.
      - Airline Reservation: All available seats stored in block for Flight 123.
    - Read N: Read block number “N”.
    - Write N: Write block number “N”.
    - Block number relative to beginning of the file (Relative Access).

sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

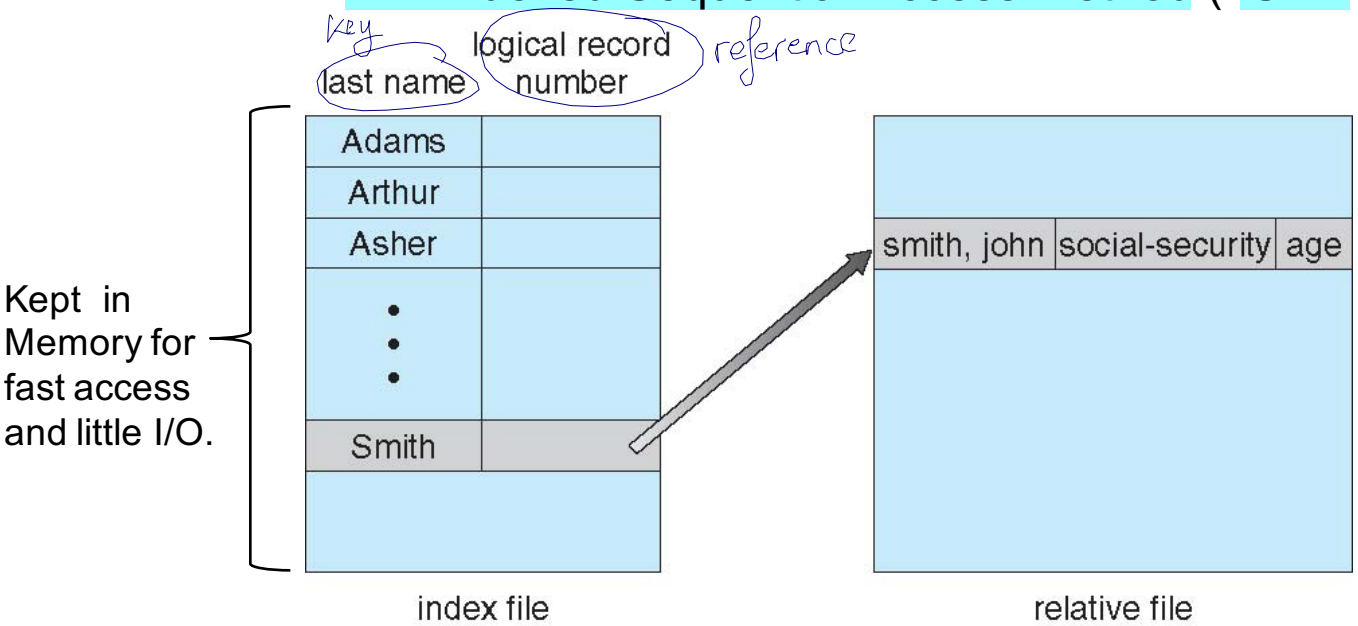
Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Access Methods
  - Index and Relative Access Method (Direct Access Method)
  - Index created for a file kept in Memory: Pointers to the blocks.
  - Search using index and using pointer to access the block.
    - Minimizes I/O.
  - Large index placed in Index File containing pointers to the Relative File containing actual file block.
    - IBM Indexed Sequential-Access Method (ISAM).

Method 3.

MySQL.

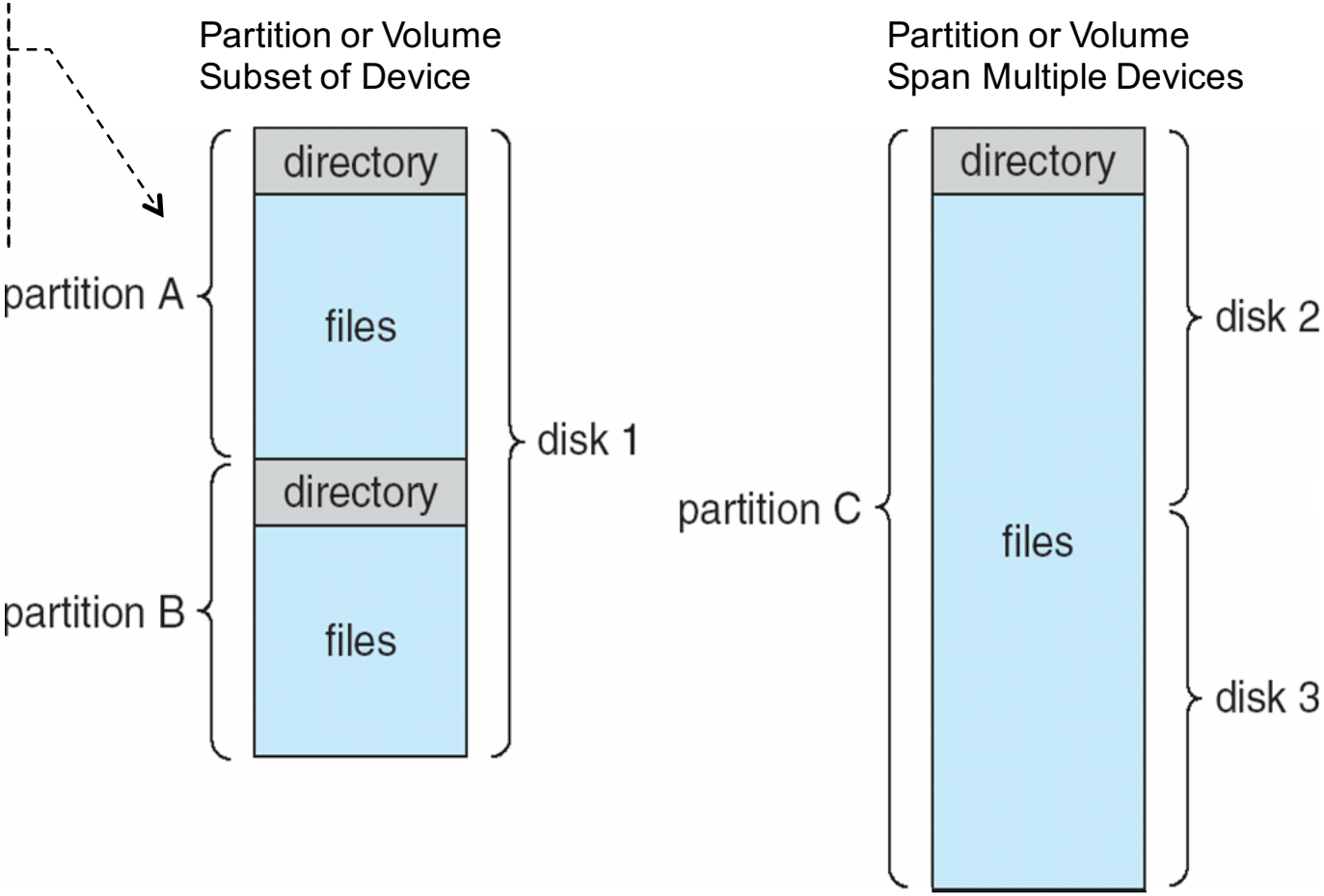


Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Directory and Disk Structure

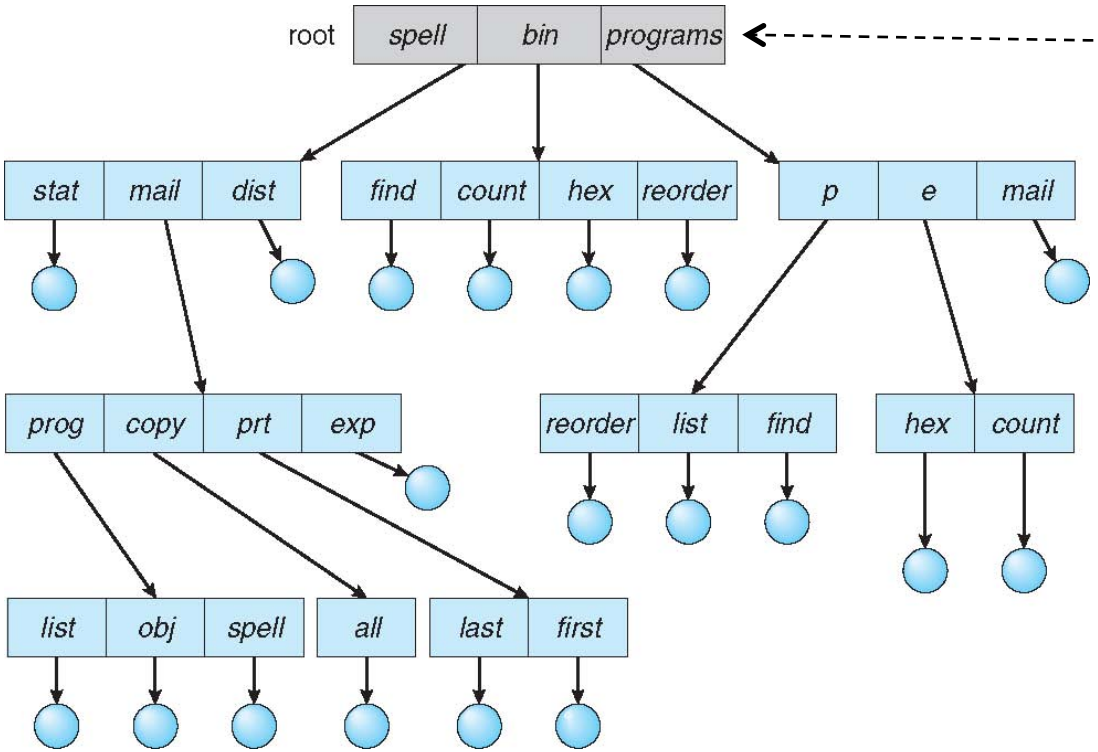
How to Store Files?
Partioning: Subdivide a Storage Device (File System, Operating System, Swap Space).



Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Directory and Disk Structure
- Tree-Structured Storage
- Tree with arbitrary levels has a root directory and every file in the system has unique path name.



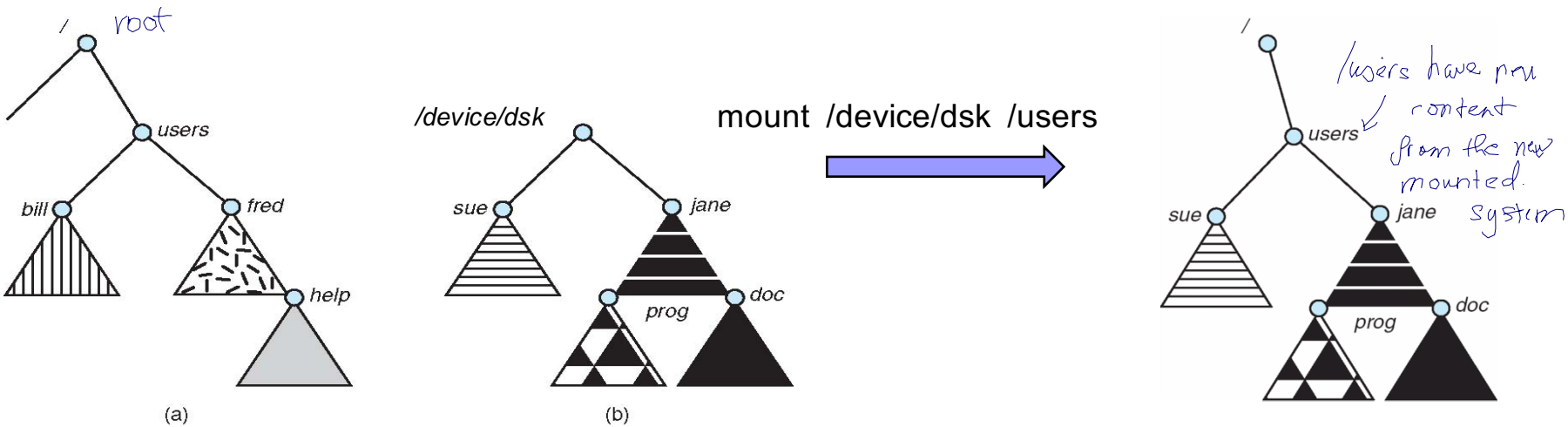
Current Directory (Working Directory): Relative path is derived.
Absolute Path: From the root to the specified file.
Relative Path: Begins at Current Directory.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

directory is the mount point.

- File-System Mounting
- Before File system is used, directory structure must be made available to processes on the system, by mounting.
- Mount Point:
  - Empty Directory which is the location where the file system is attached.
  - File System Type can be provided, else Operating System will determine the type of file system.
  - Device Driver reads the device directory to verify the device directory has expected format.
  - Mount file system in existing directory will obscure the existing directory until the file system is unmounted.



Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface (does not mention)

- File Sharing
- Multiprogramming Operating System must allow user processes to interact with each other through file sharing.
- Operating System must mediate the file sharing: providing access control and protection.
  - File and Directory Attributes expanded:
    - Owner (User) can change attributes and grant access.
    - Group is a subset of users who can share access to the file.
    - Group access permissions (Read/Write/Execute) can be different from "other" access permissions.
    - Owner and Group IDs are stored with the file or directory attributes.
- Remote File Systems
  - Networking allowed communications among remote computers and sharing among remote computers.
  - FTP and TFTP protocol: File transfers between computer systems.
  - Distributed File Systems: Remote directories are visible from local computer system.
  - World Wide Web: Interface to remote computer systems through WEB Browser and FTP, TFTP is used to transfer files.
    - Uses Anonymous FTP

Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

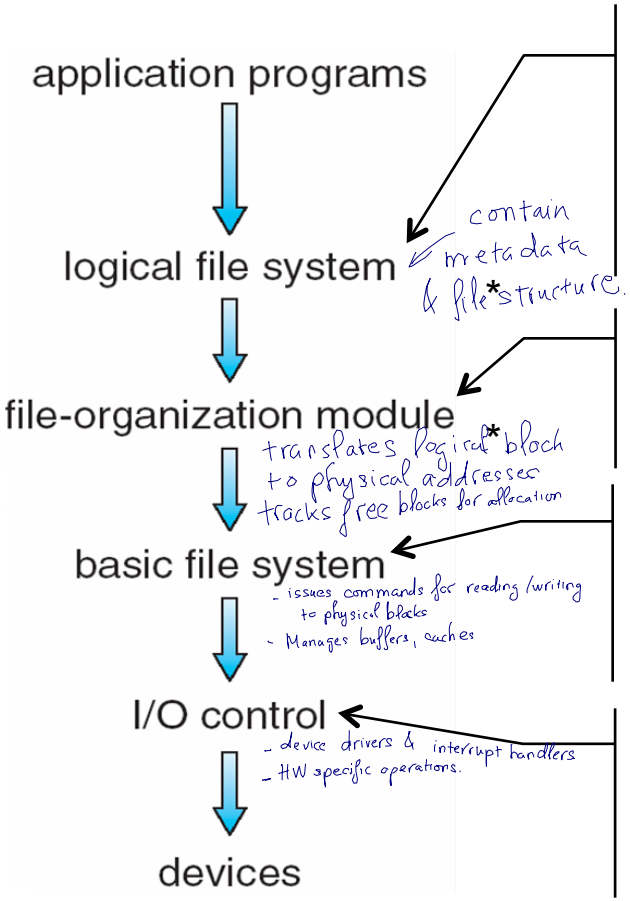
(does not mention)

- File Sharing
- Remote File Systems
- The Client-Server Model
  - Allows a computer system to mount file systems from remote machines.
  - Server: Remote machine containing the files.
    - Declares resource is available and specifies which files and exactly which clients.
    - Files on a volume or directory level.
  - Client: Local machine seeking access to the files.
    - Specified by network name or other identifies (i.e. IP Address).
    - Client Networking Information.
      - User ID must be same on Client as Server: Allows access rights based on the user ID.
  - Mounted File System is sent file operation requests via the NFS Protocol.
  - **NFS** (Network File System) is standard UNIX client-server file sharing protocol.
  - **CIFS** (Common Internet File System) is standard Windows protocol.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- File-System Structure
  - File System composed of many different levels.
  - Layer concept can be reused by multiple file systems.
    - Each file system has own Logical File System and File Organization Module.



Logical File System manages metadata information (file system structures except actual data).
<u>FCB or File Control Block (Linux Inode) has information on file ownership, permissions, location</u>.
Manages the directory structure used by the File Organization Module.

File Organization Module handles files and <u>logical block addresses to physical block addresses translation</u>.
Free Space Manager tracks unallocated blocks.

Basic File System issues commands to device driver to read/write physical block on the disk.
Manages memory buffer, cache for file system directories and I/O blocks.

Lowest level: I/O control consists of device drivers and interrupt handlers. Transfers I/O blocks from main memory to disk.
Hardware-specific instructions used by the hardware controller to interface with I/O device.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- File-System Implementation
  - Structures In-Memory for Implementing File System
    - Mount Table: Information on each mounted volume.
    - Directory Structure Cache: Directory information of recently accessed directories.
    - Open-File Table (system wide): Copy of the FCB of each open file.
    - Open-File Table (Per Process): Pointer to the FCB in system-wide Open-File Table.
    - Buffers holding File-System blocks being read/written from/to disk.

defines the physical structure of file system

File Control Block (FCB)

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

FCB:

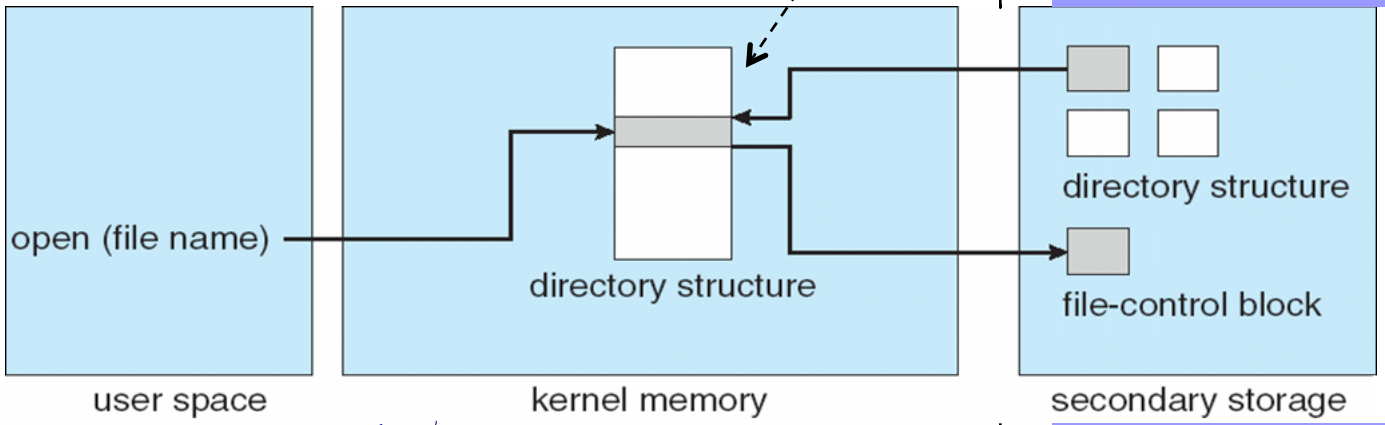
1) Open () system call passes file name to Logical File System and looks up file name in Directory Structure.
2) Search system-wide Open-File Table.
3) Already exist (used by another process): Per-Process Open-File Table created pointing to the system-wide Open-File entry.
4) Not exist, Directory Structure searched, the FCB is copied into system-wide Open-File Table. Open-File Table keeps track of processes using the file.
5) Per-Process Open-File Table created pointing to the system-wide Open-File entry.
6) UNIX: File Descriptor is index into Per-Process Open-File Table. Windows: File Handle.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- File-System Implementation

Open() file name NOT in system-wide open-file table. The directory structure is searched for file name. Parts of the directory structure is cached in memory. The entry contains the pointer to the FCB.

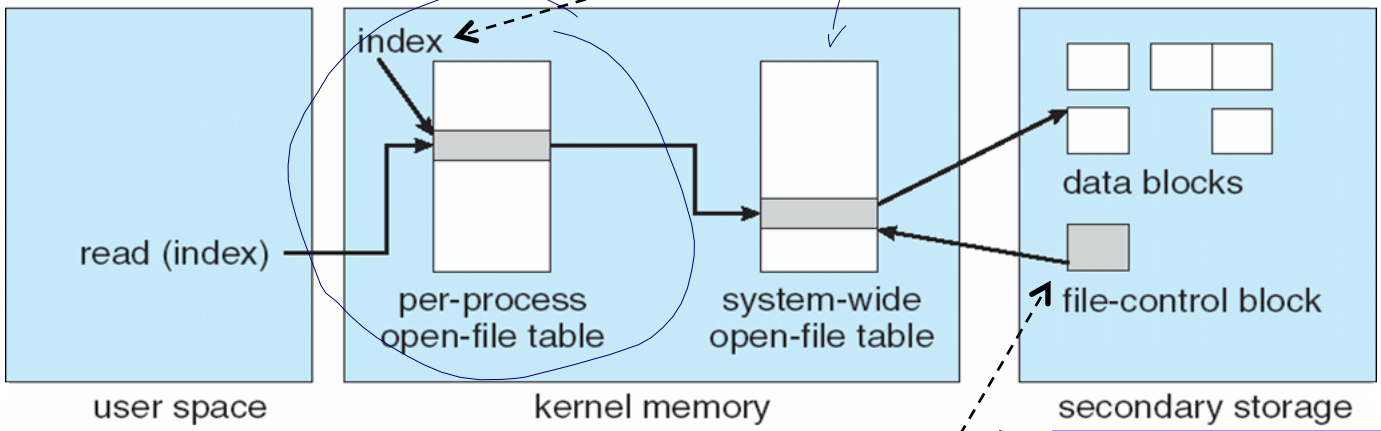


(a)

file descriptor table (per process)

user do not see this

Application uses Index (File Descriptor) to access file data.



(b)

FCB is placed in the system-wide open-file table during the open().

Copyright @ 2009 John Wiley & Sons Inc.

SILICON VALLEY UNIVERSITY CONFIDENTIAL

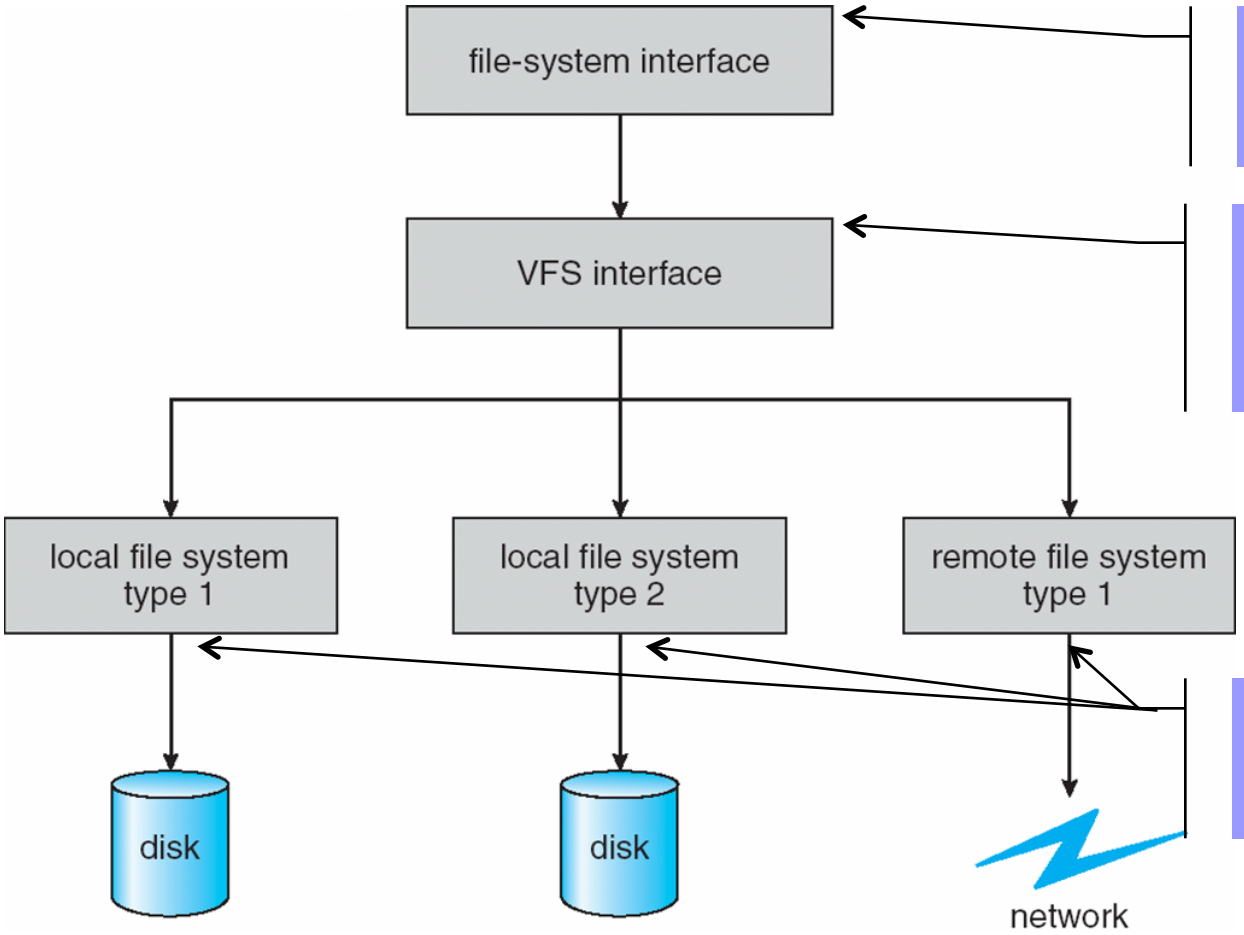
File-System Implementation

- File-System Implementation
- Virtual File System
- Integrating Multiple File Systems into Directory Structure
  - Object-oriented technique to modularize implementation.
  - Data structures and procedures used to isolate System Calls from implementation details.
- VFS API uses same System Call API to be used for different types of File Systems.
  - Clean VFS interface: Separates file-system-generic operations.
  - Vnode, file-representaiton structure to uniquely representing a file in a file system locally and throughout a network.
    - Network File System support.
    - Vnode structure for each active file or directory node.
    - UNIX: Inode unique within a single File System.
  - File-System types
    - File-System-Specific operations are called through the inode structure.
    - VFS does not know whether an inode represents a disk file, a directory file or a remote file (through NFS).

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- File-System Implementation
- Virtual File System Major Layers



1) User interface with File Descriptors : open(), read(), write(), close().

2) VFS API, Inode for active node (file or directory). Inodes are unique within a single File System.

3) The File System type or the Remote File System.

Copyright @ 2009 John Wiley & Sons Inc.

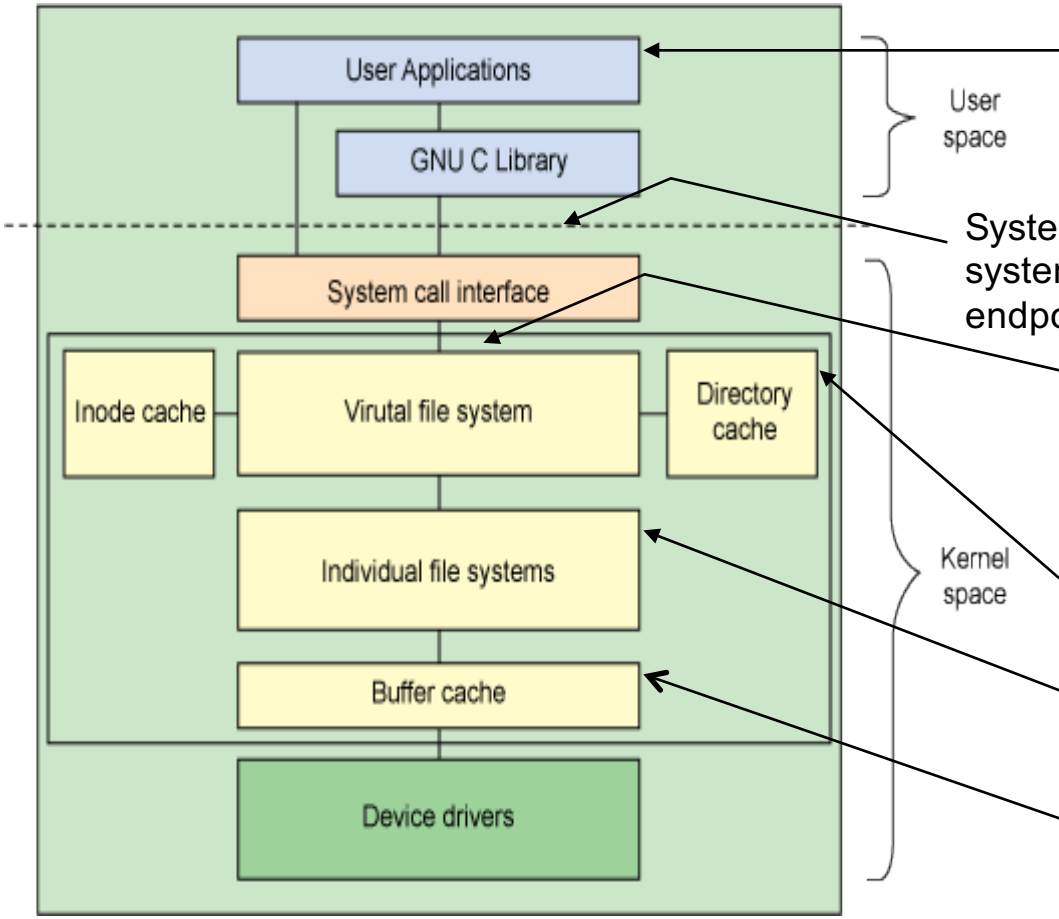
File-System Implementation

- File-System Implementation
- Virtual File System
- Main Object Types defined by Linux VFS:
  - Inode Object: Represents an individual file.
  - File Object: Represents an Open File.
  - SuperBlock Object: Represents as entire File System.
  - Dentry Object: Represents an individual Directory entry.
- Set of Operations defined for each Object Type.
- VFS performs operation on the Object Types by indirectly calling the functions registered for the Object Types.
  - Does not need to know what kind of object (i.e. Inode Object could represent a disk file, directory file, or a remote file).

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Virtual File System (VFS)



User space has the applications and glibc (provides API file system calls: open, read, write, close).

System call interface acts as a switch, funneling system calls from user space to the appropriate endpoints in kernel space.

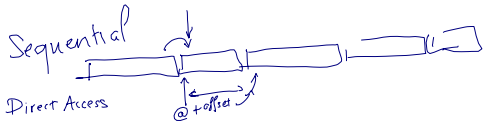
VFS exports APIs and abstracts them to individual file systems.
Inode cache and Directory cache provides a pool of recently-used file system objects.
Directory cache speeds up translation from file pathname to inode.
Individual file systems exports APIs that is used by the VFS.
Buffer cache buffers the requests between the file systems and the block devices that they manipulate (for faster access).

File-System Implementation

- Directory Implementation
- Directory Allocation and Management Algorithms
  - Affects the efficiency, performance, and reliability of the File System.
- Linear List
  - Linear list of File names and pointers to the data blocks.
  - File search, creation and deletion, needs a linear search.
    - Improve on file access with:
      - Software cache of most recent directory information.
      - Sorted linked list.
      - Binary tree of directory entries.
- Hash Table
  - Directory search time greatly decreased.
    - Hash index using the File Name to compute the index.
  - Hash function can make Hash Table very large.
  - Handle collisions using linked list of collided entries.
    - Slower, but easier to implement.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

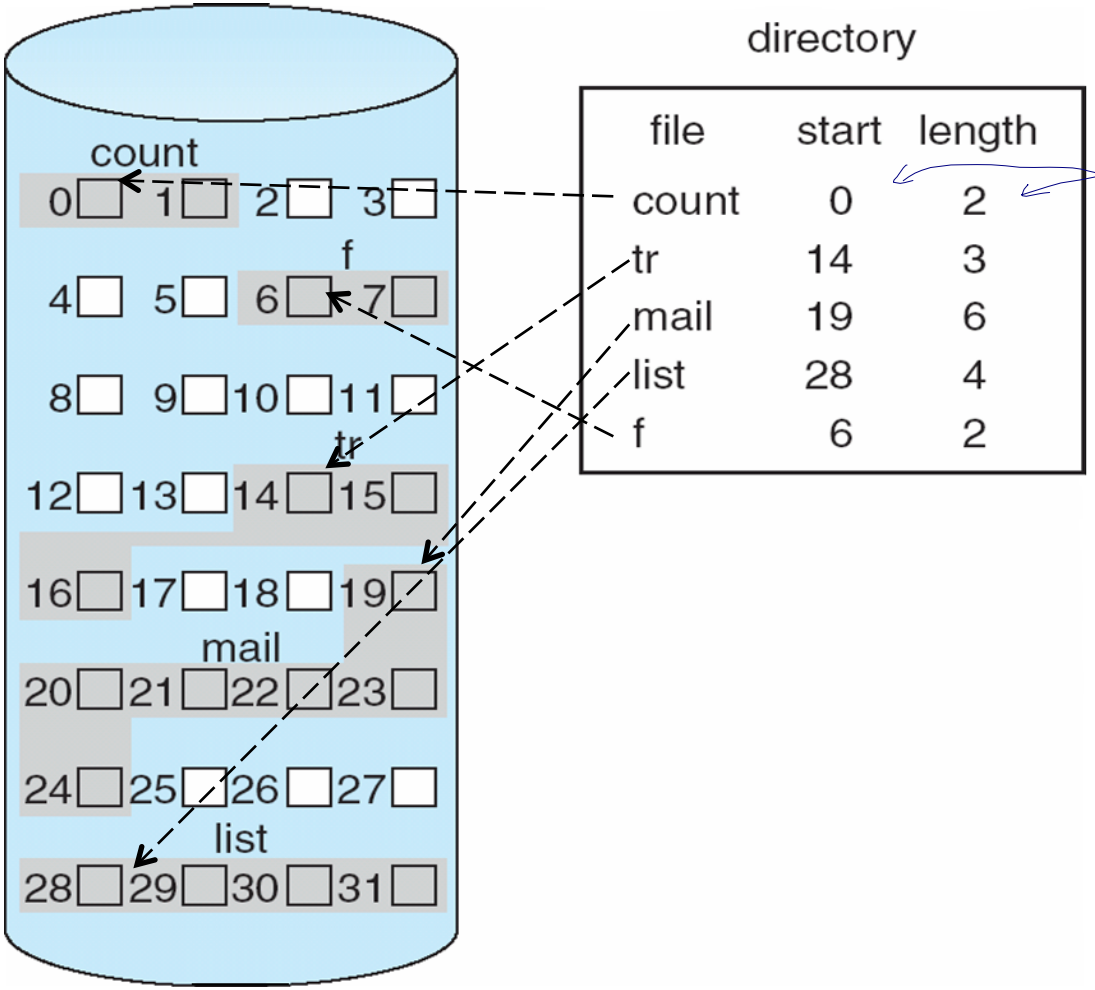


- Allocation Methods
- Disk Space Utilized Effectively
- Three Methods of Allocating Disk Space:
  - Contiguous, Linked, Indexed.
- Contiguous Allocation ← direct Access
  - File occupy set of continguous blocks: Disk address define linear ordering on the disk.
    - Sequential Access: Disk address of last block plus next block.
    - Direct Access: Disk address of first block plus block offset.
  - Minimal head movement, seek minimal.
    - IBM VM/CMS Operating System uses Contiguous Allocation for performance.
  - Limitations: Finding contiguous space for new file.
    - First Fit and Best Fit dynamic storage –allocation.
    - File extension: Difficult estimating size of an output file (over-estimate).
      - Over-estimitation and internal fragmentation.
    - External fragmentation.
    - Off-line compaction (copying to another disk or tape and then compacting).
    - Hybrid contiguous-allocation scheme.
      - Extend to another contiguous block (Internal fragmentation when extend too large).

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

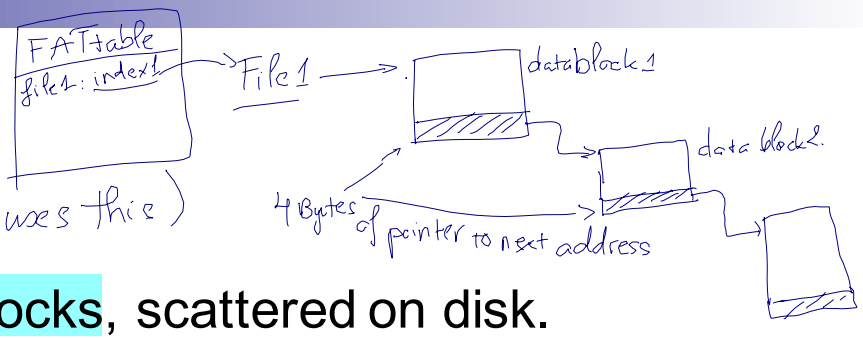
- Allocation Methods
- Contiguous Allocation



direct access

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

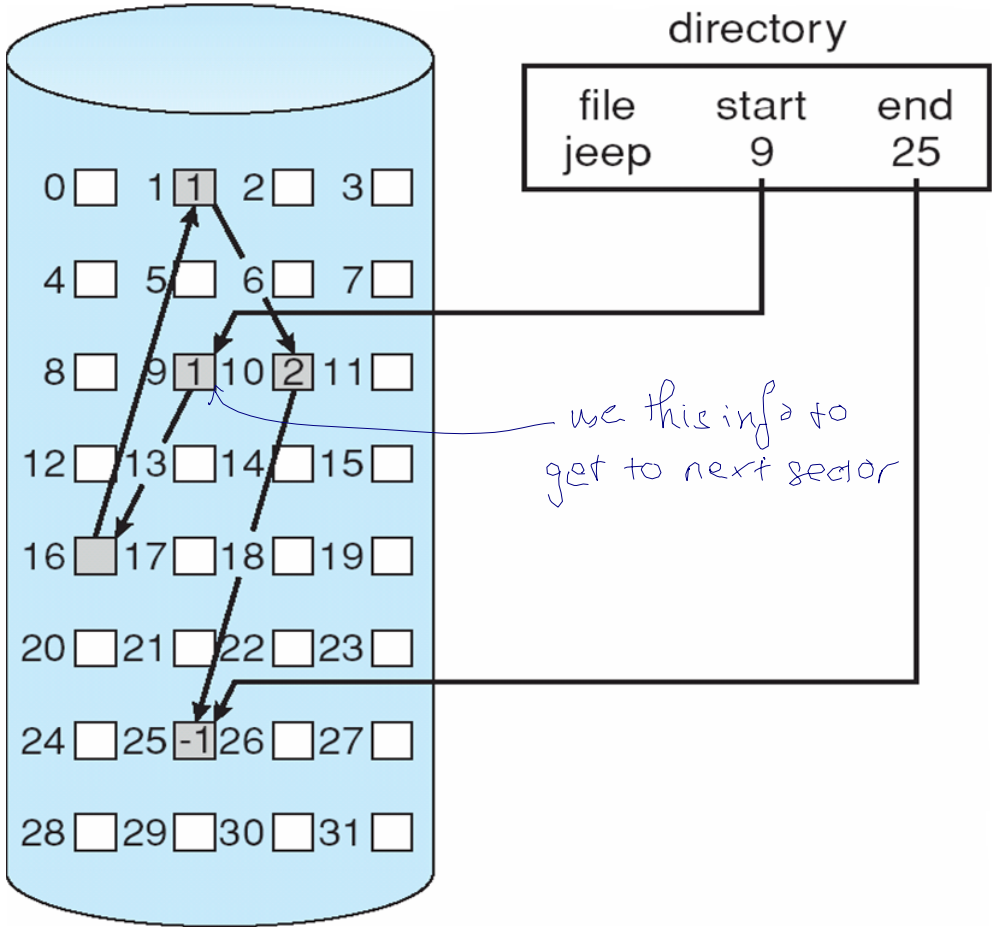


- Allocation Methods
- Linked Allocation ← most popular (FAT uses this)
  - Each file is a linked list of disk blocks, scattered on disk.
  - Each block contains 4 Byte pointer to the next block.
    - For 512 Byte block, has 508 Bytes of usable space.
  - Size of file does not have to be known.
    - A file can grow as long as free blocks are available.
  - Directory contains pointer to the first and last block of the file.
  - Limitations:
    - Effective only for Sequential-Access files.
    - Inefficient for Direct-Access files
  - Allocate clusters of blocks, instead of one block at a time.
    - Internal fragmentation problem.
    - Reliability: Pointers can be damaged in the block.
  - File Allocation Table (FAT): MS-DOS and OS/2.
    - Section of disk at the beginning of each volume contains the FAT.
    - Table has one entry for each block and indexed by block number
    - Access similar to Link List.
    - FAT cached to minimize disk overhead (disk head seeks).

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

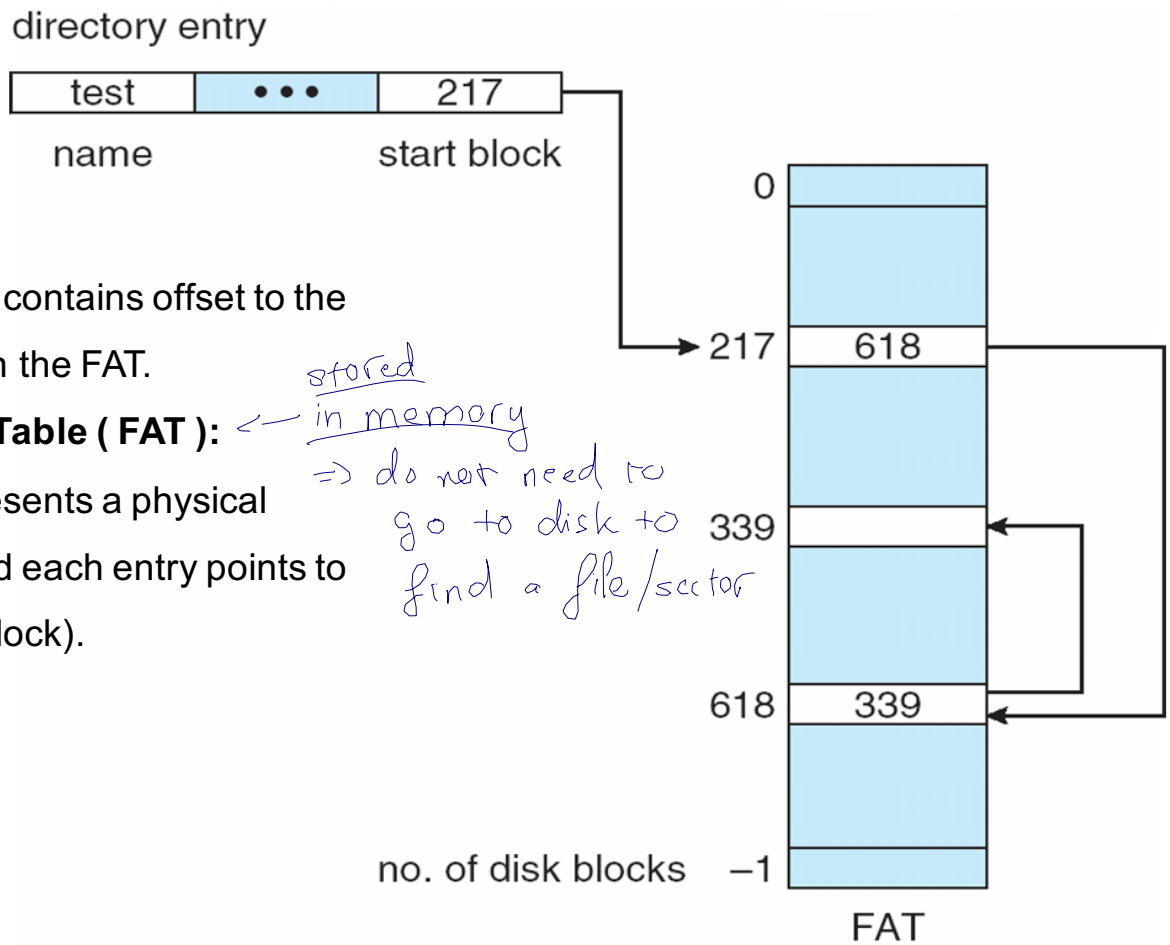
- Allocation Methods
- Linked Allocation



Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Allocation Methods
- Linked Allocation



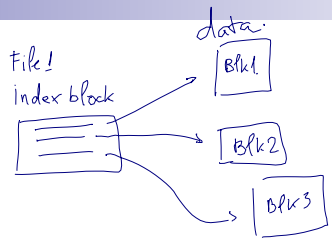
Directory Entry contains offset to the start of the File in the FAT.

File Allocation Table (FAT): ← stored in memory ⇒ do not need to go to disk to find a file/sector

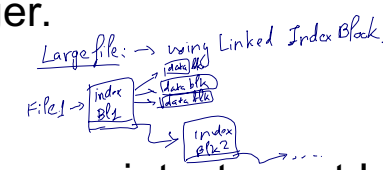
Each entry represents a physical block on disk and each entry points to the next entry (block).

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation



- Allocation Methods
- Indexed Allocations
  - Index Block: Bring all pointers into one location.
- Each file has index block: Pointers to each Data Block.
  - Similar to Page Table for Virtual Memory.
  - Index block allocated disk block from free-space manager.
  - Supports direct access without external fragmentation.



- Linked Index Block:
  - Large files, link several Index Blocks. The Index Block has pointer to next Index Block.
  - Pointer overhead of Index Allocation more overhead than Linked Allocation.
- Multilevel Index Block:
  - First –level Index Block points to Second-level Index blocks, etc.
  - 4096 Byte blocks → 1024 4 Byte pointers in Index Block → 2 levels Index Blocks → 1024 x 1024 → 1048576 data blocks x 4096 blocks → 4 Gbytes file.
- Combined Scheme:
  - First 15 pointers of the Index Block in the Inode.
  - First 12 pointers are pointing to direct blocks.
  - Next 3 pointers are single, double, triple indirect blocks.
  - Number of Data Blocks to a file exceeds 2^32 or 4 GB.

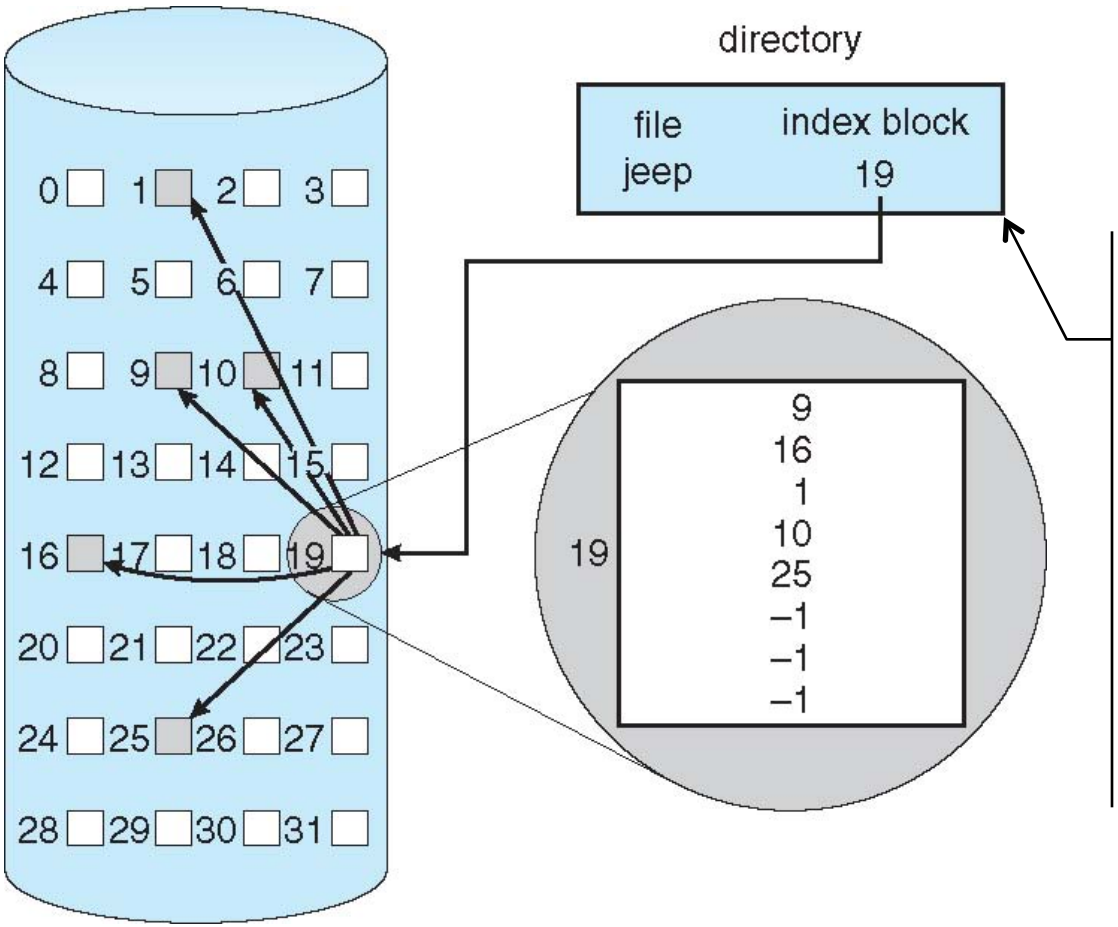
?

- Index Blocks can be cached in memory for performance.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Allocation Methods
- Indexed Allocation



1) Directory entry contains the address of the Index Block.
2) Index Block is an array of disk-block addresses.
3) “ith” Data Block, use the disk address in the “ith” Index Block entry.

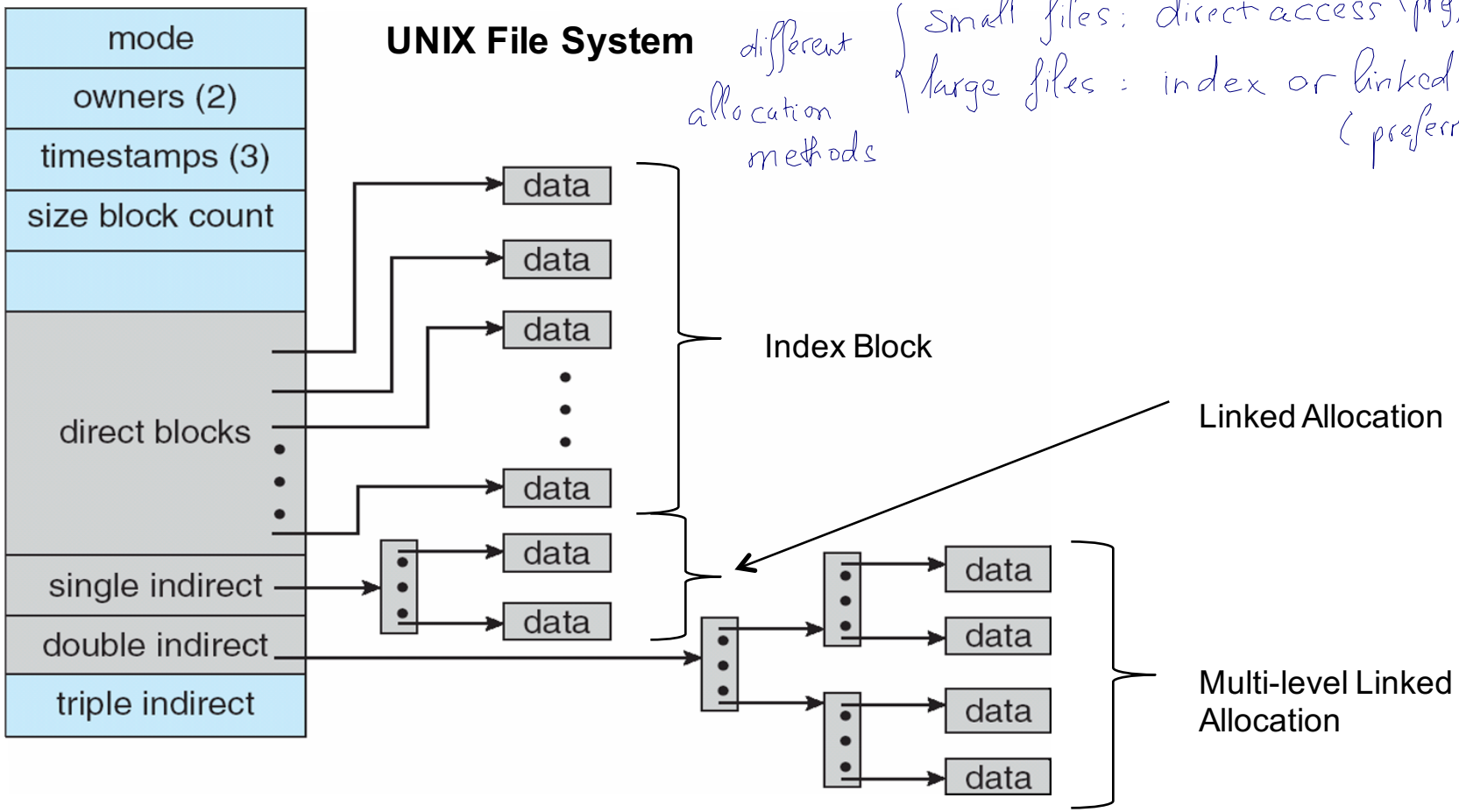
Index block per file similar to Paging scheme with Page Table per process.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Allocation Methods
- Combined Scheme (Linked and Multi-level) Allocation

different allocation methods { small files: direct access (preferred) / large files: index or linked list (preferred).



Copyright @ 2009 John Wiley & Sons Inc.

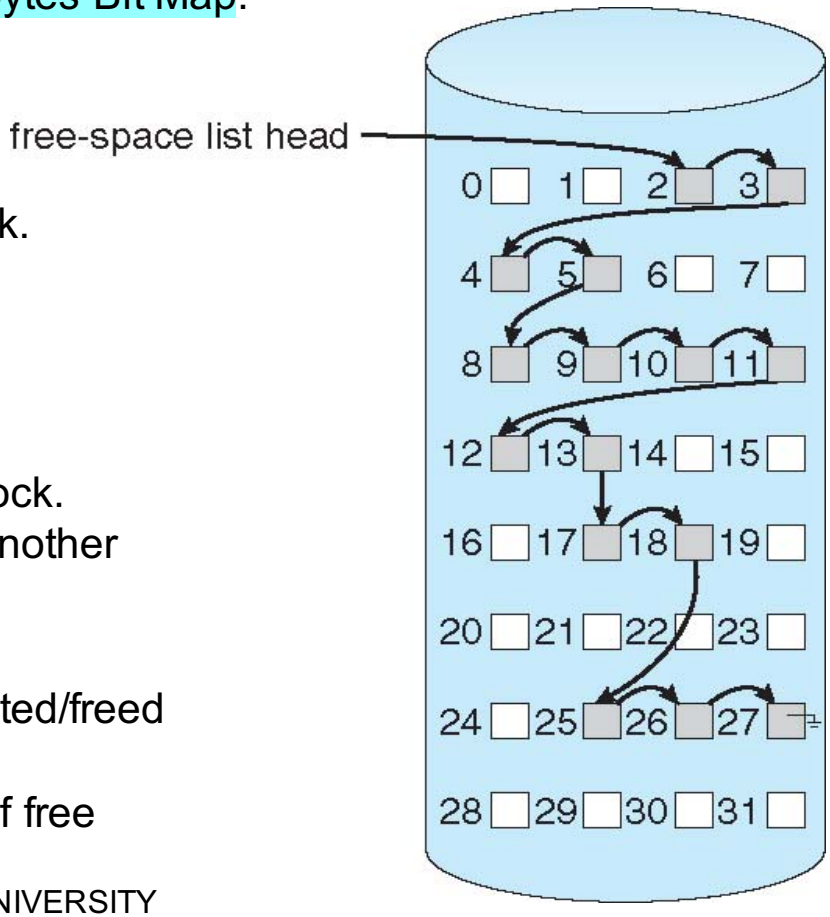
File-System Implementation

- Allocation Methods
- Performance
  - Allocation Methods Measured by:
    - Storage efficiency and data-block access time.
  - Criterias: How a system is used
    - Sequential Access
      - Linked Allocation (address of next block in memory).
    - Random Access
      - Direct Access.
      - Contiguous Allocation (start of block plus offset).
    - Define how a file is used and create either contiguous (Direct Access with maximum length) or Linked Allocation (Sequential Access).
    - File converted from one type to another by creating new file of desired type.
      - Operating System uses contiguous allocation for small files (3 or 4 blocks) and Index Allocation for larger.
      - Average performance is good, since most files are small.
  - UNIX allocates space in clusters of 56 Kbytes (DMA transfer size).
    - Minimize external fragmentation, seek and latency times.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Free-Space Management
- Bit Map or Bit Vector: 1=Free, 0=Allocated
  - Disk size constantly increasing bit vector continually increasing.
  - 1TByte disk with 4KByte blocks = 32 Mbytes Bit Map.
- Free-Space Linked List
- Linked List
  - Link together free disk blocks:
    First free block in special location on disk.
  - Operating System usually requests first block in the Linked List.
  - FAT uses Free-Block Linked List.
- Grouping Linked List
  - Address of “n” free blocks in first free block.
  - “(n-1)” block address is the address of another “n” free blocks.
- Counting
  - Expect several contiguous blocks allocated/freed simultaneously.
  - Address of first free block and number of free contiguous block that follow.



Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Recovery
- System crash can cause inconsistencies among on-disk File-System data structures.
  - Typical File Operation causes:
    - Directory Structures are modified.
    - FCBs are allocated.
    - Data Blocks are allocated.
    - Free counts for FCB pointers, Free-Block pointers are decremented.
    - Operating System caches data structures.
  - Operating System can interrupt these changes.
- Consistency Checking
  - File System must detect and correct problems.
    - Metadata scan to confirm or deny the Operating System consistency when system boots.
    - File System can record its state within File System metadata.
  - Consistency Checker: fsck ( ) in UNIX and chkdsk in Windows.
    - Allocation algorithm and Free-Space Management algorithm dictates what type of recovery is possible.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Recovery
- Log-Based Transaction-Oriented File Systems (Journaling)
  - All metadata changes (Transactions) are written sequentially to a circular buffer (Log).
  - Once written to the Log, transactions are considered committed and system call returns to the User Process.
  - Log entries are replayed across File System and removed from the Log when entry completes.
    - Operating System crashes before Log entry completed,
    - Changes from incomplete transaction must be undone when Operating System rebooted.
    - All transactions in Log will execute to modify the File System.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Magnetic Disks: Secondary Storage of Computer Systems
  - Multiple Disk Platters Rotates at 60 – 200 times per second.
  - Transfer Rate: Rate of data flow between Disk Drive and Computer System.
    - Typically several MegaBytes per second.

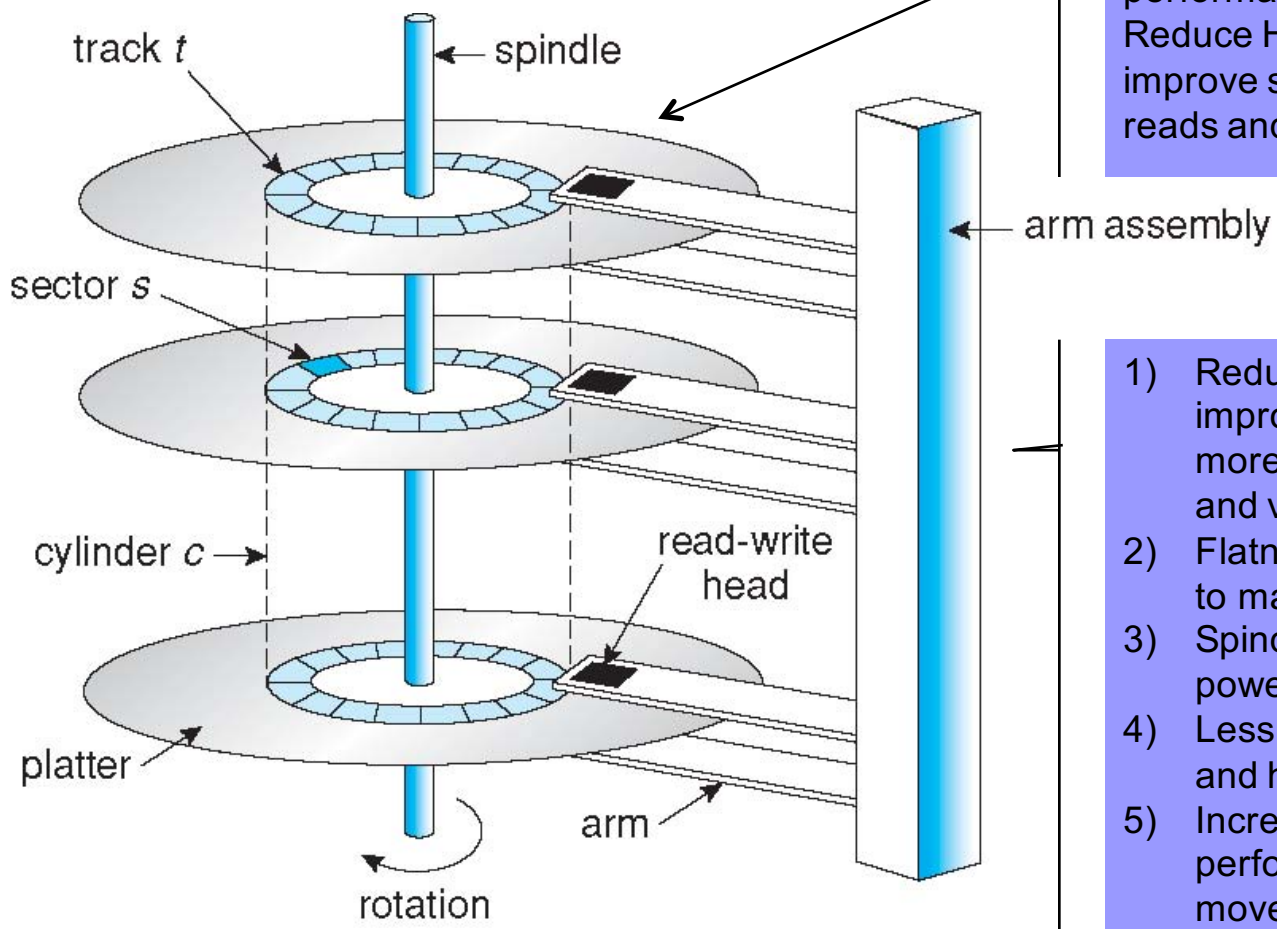
  - Positioning Time: Time move Disk Arm to Cylinder (Seek Time) and time rotate desired sector to disk head (Rotational Latency).

    - Typicallly Seek Time and Rotational Latency several milliseconds.
  - Disk Head travels over disk, separated by microns.
    - Head crash when Disk Head contacts the Disk.
  - Removable Disk: Consists of one Disk Platter.
  - Disk Controller Connected to Host Controller with I/O Bus.
    - I/O Busses: EIDE (Enhanced Integrated Drive Electronics), ATA (Advanced Technology Attachment), SATA (Serial ATA), USB (Universal Serial Bus), FC (Fibre Channel), SCSI (Small Computer Systems Interface).

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Magnetic Disks: Secondary Storage of Computer Systems



Platter Size related to performance.
Reduce Head movements and improve seek times (Faster reads and writes).

1) Reduce platter size, improve stiffness and more resistant to shock and vibrations.
2) Flatness of surface eaiser to manufacture.
3) Spindle spins faster , less-powerful motors.
4) Less power, reduces noise and heat.
5) Increases seek performance, less head movements.

Copyright @ 2009 John Wiley & Sons Inc.

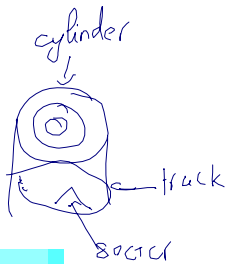
Mass-Storage Structure

- Magnetic Tapes: Early Secondary-Storage Medium.
- Large Quantities of Data (20 GBytes to 200 Gbytes).
- Limitations:
  - Access time slow (1000 times slower than Hard Disk).
    - Backup File System
  - Mainly used for backup, storage of infrequently used data.
  - Tape in spool and moves across Disk Head.
  - Built-in Compression to increase capacity.
- Magnetic Tapes categorized by width
  - 4, 8, and 19 Millimeters
  - ¼ and ½ Inches.
  - LTO-2 (Linear Tape-Open) Tape Cartridge
  - SDLT (Super Digital Linear Tape) Tape Cartridge

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Structure
- Mapped as one-dimensional Array of Logical Blocks
  - Logical Block: Typically 512 Bytes.
  - Sector 0: First sector of first track on outermost cylinder.
    - Mapped outermost cylinder to innermost cylinder.
  - Disk Address: Cylinder number, Track number, Sector number
    - Difficult to translate logical block to physical sector.
      - Number of sectors per track not constant in some drives.
    - Further from center, greater number of sectors per track.
    - CLV (Constant Linear Velocity) specifies uniform density of bits per track.
      - Drive increases rotational speed when head moves from outer to inner track.
      - Used in CDROM and DVDROM.
    - CAV (Constant Angular Velocity) specifies bit density of inner track less to outer track.
      - Drive rotational speed constant.
      - Used in hard disk.



Copyright @ 2009 John Wiley & Sons Inc.

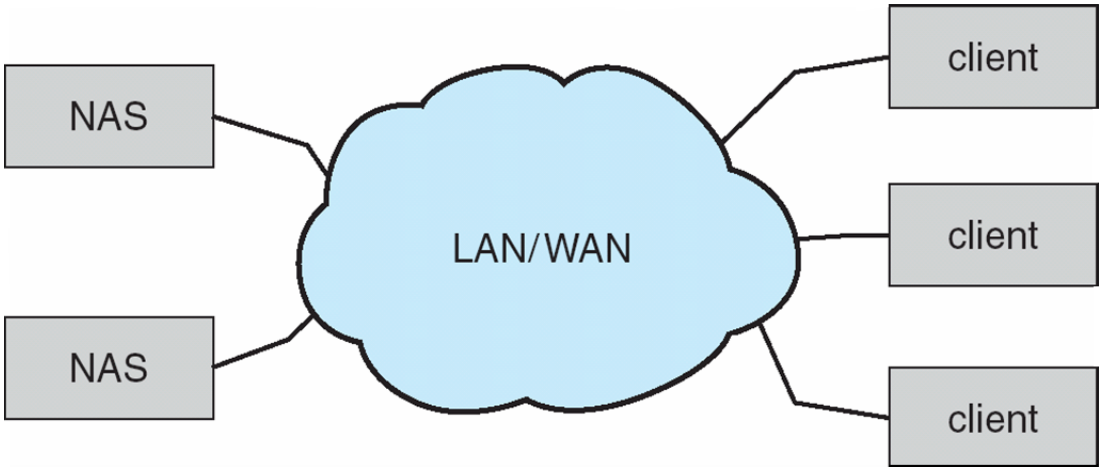
Mass-Storage Structure

- Disk Attachment
- Computers Access Disk: Host-Attached Storage (I/O Ports) or Network-Attached Storage (remote host).
- Host-Attached Storage
  - Access through I/O ports.
  - IDE (Integrated Drive Electronics) or ATA (Advanced Technology Attachment)
    - Supports 2 drives per I/O Bus.
  - SCSI (Small Computer Systems Interface)
    - Supports 16 drives or SCSI targets per I/O Bus.
    - Host Controller Card
    - Each SCSI target address up to 8 Logical Units
  - FC (Fiber Channel)
    - 24 bit address Switch Fabric.
      - Multiple hosts and Storage Devices can attach to the fabric.
    - FC-AL (Arbitrated Loop)
      - Supports 126 devices (drives and controllers).

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Attachment → Host Attached Storage. → Network Attached Storage
- Network-Attached Storage (NAS)
  - Accessed remotely over an IP network using TCP or UDP.
  - Clients uses RPC (Remote Procedure Calls)
    - NFS for UNIX
    - CIFS for Windows
  - NAS usually a RAID array with RPC software.
  - Lower performance than direct-attached storage devices.
  - iSCSI: Uses IP network protocol to carry SCSI protocol.

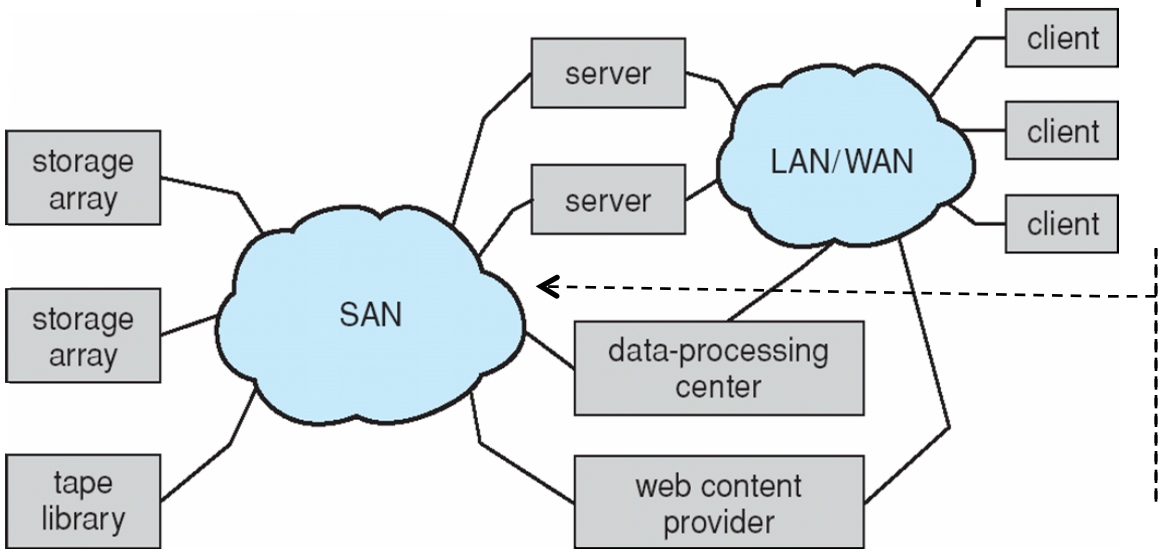


Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Attachment
- Storage-Area Network (SAN)
- Private Network Connecting Servers and Storage Units.
- Uses Storage Protocols rather than Network Protocols
- Multiple Hosts (Servers) and Multiple Storage Arrays are attached.
- SAN Switch controls access to the Storage Units.
- Fiber Channel Interconnects SAN components.

SAN v/s NAS ?



Separate storage I/O operations from data network traffic. Remove possibility of congestion on LAN/WAN from SAN operations.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- Operating System must provide fast access time and transfer rate to mass storage.
  - Access Time Components:
    - Seek Time: Disk arm to move the heads to the cylinder of the sector.
    - Rotational Latency: Disk to rotate to the sector.
  - Disk Bandwidth: Total Bytes transferred, divided by total time between first request and completion of transfer.
  - Improve Access Time (Seek Time and Rotational Latency) by managing order of disk I/O Request.
  - Disk Queue: Requests made when drive or controller is busy.
  - Disk Scheduling: Operating System algorithm to choose the next pending request to service.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

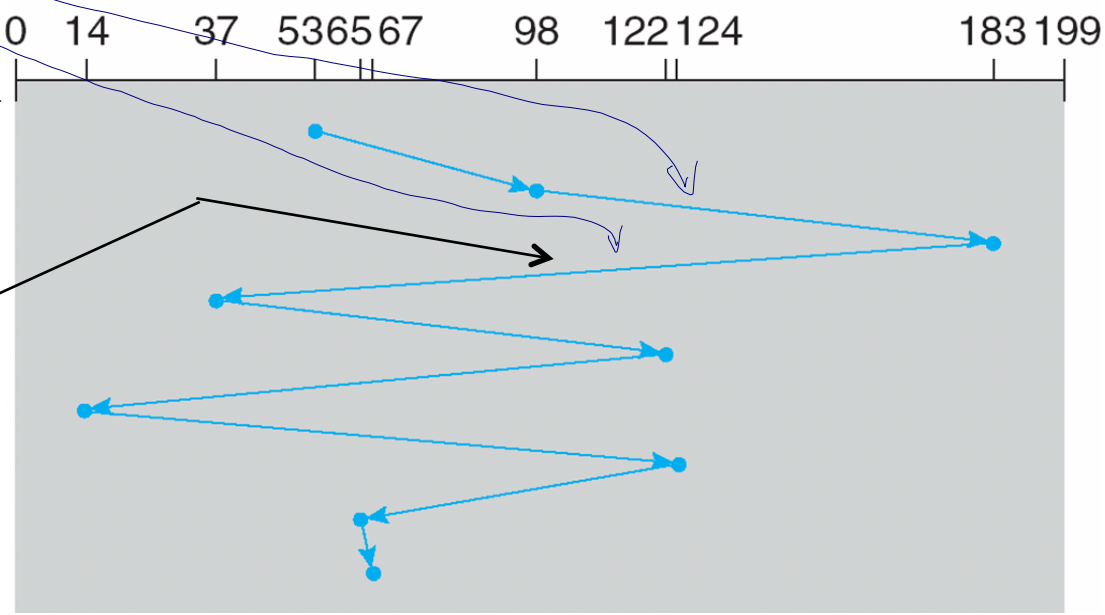
- Disk Scheduling
- FCFS Scheduling: First Come First Serve

based on the current head

  - Fair, but does not provide the fastest service.
  - Large swings of the disk head possible, causing lengthy Access Time.

the slower access the large swing

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



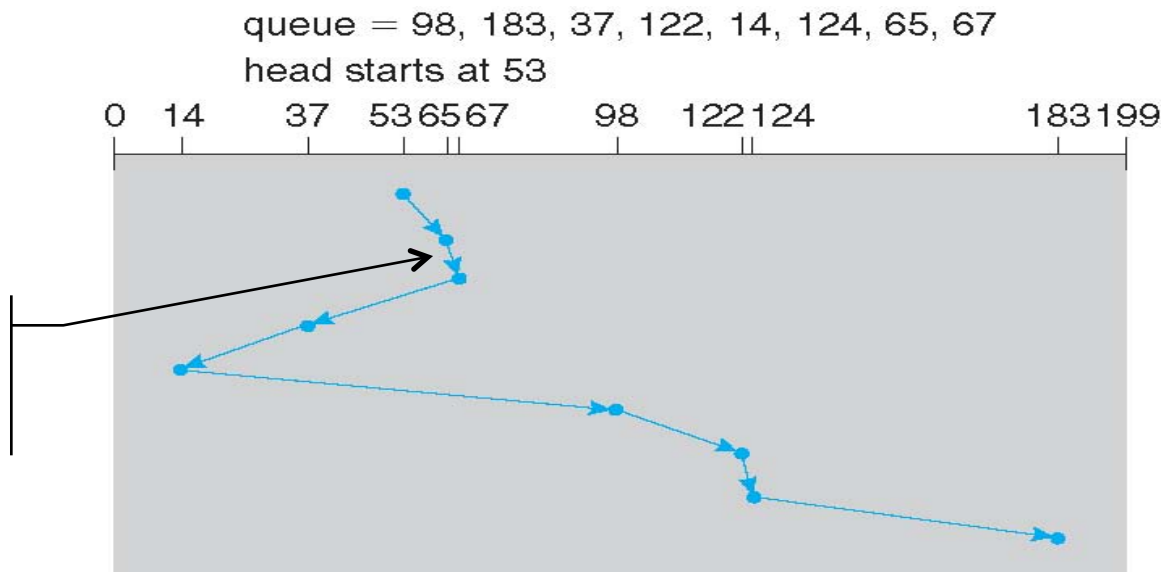
Cylinders: Seek time is the Head Movement to a Cylinder.

Large jumps between Cylinders.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- SSTF Scheduling: Shortest Seek Time First
  - Service requests closes to current Head Position before moving Head further away (minimize Seek Time).
  - Similar to Shortest Job First Scheduling.
  - Starvation of Job furthest from the current request queue.

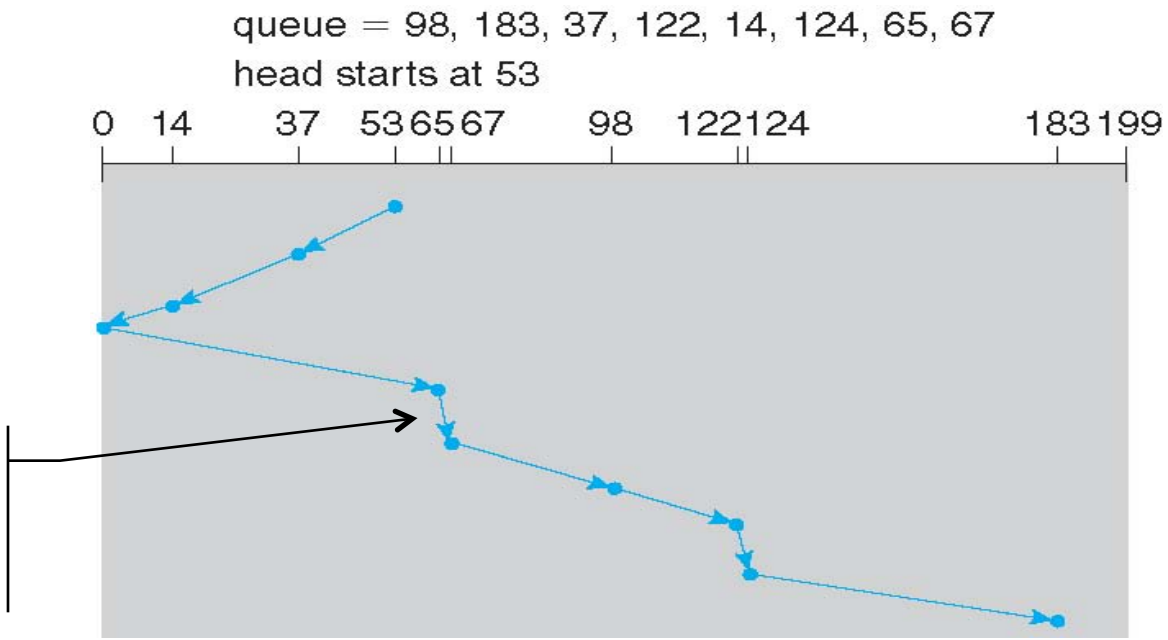


Cylinders:
Minimize Cylinder access (seek) time.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- SCAN Algorithm
  - Elevator Algorithm: Disk arm starts at one end of the disk and moves towards the other end, servicing requests at each cylinder. When the end is reached, head movement is reversed.
  - Requests collect at the cylinders that was just passed.



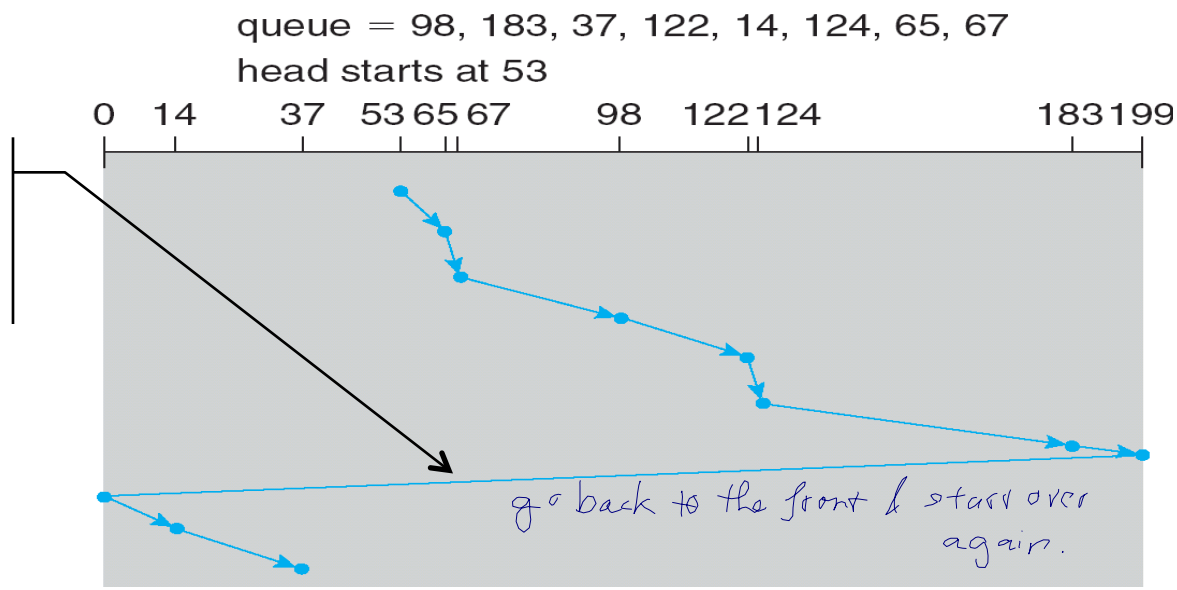
Cylinders:
Access chart similar to SSTF.
Minimize Cylinder access (seek) time.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- C-SCAN Algorithm
  - Variant of SCAN with a uniform wait time.
    - Head moves from one end to the other end servicing requests.
    - When other end reached, immediately returns to beginning before servicing requests.
    - Treats cylinders as circular list that wraps from last to first cylinder.

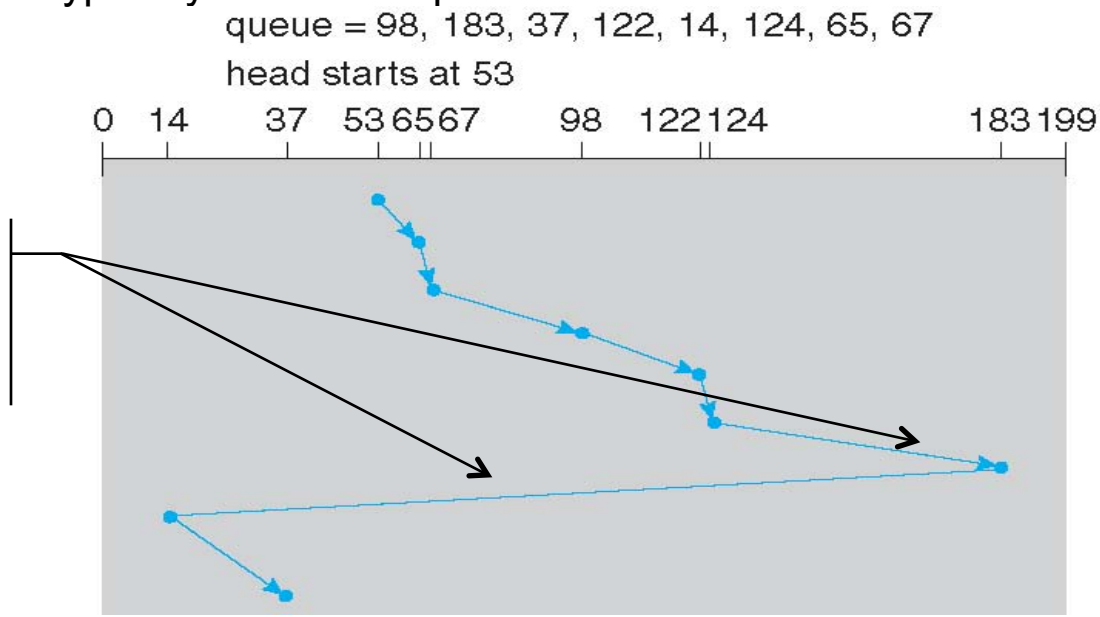
Cylinders:
Wraps from last cylinder to first cylinder before servicing requests again.



Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- C-LOOK or LOOK Algorithm
  - Variant of C-SCAN
    - Head moves from one end to the other end servicing requests.
    - Head only goes as far as final request in one direction before reversing directions.
    - SCAN and C-SCAN typically follows this pattern.



Cylinders:
Wraps as soon as last request is reached.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Disk Scheduling
- Selection of Disk-Scheduling Algorithm
  - SSTF simple, performs better than FCFS.
  - SCAN, C-SCAN perform better for systems with heavy disk usage.
    - Require retrieval algorithm.
  - File Allocation Algorithm important.
    - Contiguous blocks will have minimum head movement.
    - Linked or Indexed File can have blocks scattered on the disk.
  - Caches for directories and index blocks required, otherwise frequent directory access causes excess head movements.
  - SSTF or C-LOOK (LOOK) typically used as default algorithm.

easiest to use

- Disk Scheduling Algorithm written as separate module, allowing it to be replaced with a different algorithm.

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Swap-Space Management
  - Operating Systems merged Swapping with Virtual Memory Paging Techniques.
    - In Paging Systems, only pages of a process are swapped out.
  - Virtual Memory uses disk space as extension of main memory.
    - Disk access is slower than memory access: how swap space is used and how swap space is managed is critical to performance of Operating System.
    - Swap Space in raw partition:
      - Swap-Space Storage Manager used to allocate and deallocate blocks from the raw partition.'
      - Swap Space partition is fixed.
      - Internal Fragmentation can occur, but process life is short.
      - Increasing Swap Space require repartitioning.
    - Swap Space in File-System Space:
      - Large file within the File System: File-System APIs to create/allocate.
      - Straightforward to implement, but inefficient to use File-System data structures.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

Base on the content → define the allocation methods: DA, index, ...

- Summary
- Different Memory-Management Algorithms:
  - Direct Access → Contiguous Allocation: Each process in single contiguous section of memory.
  - Paging ⟷ frame, fixed size
  - Segmentation → code/function section, dynamic sizing
  - Segmentation with Paging → combined solution.
- Comparison Considerations:
  - Hardware Support: Base-Limit Register pair. MMU for Paging and Segmentation.
  - Performance: More complex scheme, mapping logical address to physical address takes longer. Translation Look-Aside Buffer (TLB) fast lookup hardware cache to map page to frame number.
  - Fragmentation: Fixed size allocation (Paging) has internal fragmentation. Variable size allocation (Segmentation) has external fragmentaiton.
  - Relocation: Compaction for external fragmentation.

Copyright @ 2009 John Wiley & Sons Inc.

Memory Management

- Summary
- Comparison Considerations:
  - Swapping: Essential for allowing more processes to run than can fit into memory at one time.
  - Sharing: Sharing code and data among different users through shared segments or shared pages.
  - Protection: For paging or segmentation, different sections of a user program can be declared execute-only, read-only, or read-write.
  - Linux does not rely on segmentation and uses it minimally.
    - Segment for Kernel Code, Kernel Data.
    - Segment for User Code, User Data.
    - Task-State Segment (TSS).
    - Default LDT Segment.
    - Linux mainly uses paging:
      - User Code and User Data Segment shared by all processes.
      - All Segment Descriptors are in the Global Descriptor Table.
      - TSS stores hardware context of each process during context switch.
      - Default LDT Segment not used.

Copyright @ 2009 John Wiley & Sons Inc.

Virtual Memory

- Summary
- Virtual Memory enables Operating System to map a large logical address space onto a smaller physical memory.
  - Run extremely large processes.
  - Raise the degree of multiprogramming.
  - Increases CPU utilization.
- Allow processes to share system libraries and memory.
- Demand Paging: Brings page in when page fault.
- Page Replacement Algorithms: Total Memory requirements exceed physical memory.
  - FIFO, Optimal Page Replacement, LRU (approximation of Optimal), Second-Chance (approximation of LRU).
- Frame Replacement Policy Needed:
  - Local Page Replacement: Allocation fixed to local process frames.
  - Global Page Replacement: Dynamic selection from set of all frames.
  - Working-Set Model: Pages in the current locality.
- Memory Mapping Files: File I/O treated as memory access.
- Kernel Memory Allocation:
  - Buddy System: Allocates memory to Kernel processes in power of 2 unit sizes.
  - Slab Allocators: Kernel data structures to caches associated with slabs.

Copyright @ 2009 John Wiley & Sons Inc.

File-System Interface

- Summary
- Operating System defines a file is an abstract data type consisting of a sequence of logical records.
- The major task for Operating System is to map logical file concept onto physical storage device.
- The storage device is a volume that holds a file system that has device directory listing the location of the files on the device.
  - The file system must be mounted to make the file system available for use.
  - Distributed File System allow hosts to mount file systems on remote servers throught the internet (Client-Server Model).
- The directory is generalized as a tree-structure, allowing a user to create sub-directories to organize files.
- The tree-structured directory can be seen as an acyclic-graph directory structure, allowing users to share subdirectories and files.
  - In the acyclic-graph directory, cycles has to be avoided, otherwise loops can form in searching and non-zero reference count would prevent a file or directory from being deleted.
- Information in a file system must be protected from improper access using Access Control.
  - Access Control (Read/Write/Execute).
  - File Protection with Access Lists,

Copyright @ 2009 John Wiley & Sons Inc.

File-System Implementation

- Summary
  - File System Structure is used to abstract the physical properties of secondary storage (disk) into the logical storage unit, the file, and the organization of the files, the directory.
  - The physical disk can be a partition with one File System or segmented into multiple partitions or multiple File Systems.
  - File Systems are implemented in a layered or modular structure.
    - Logical File System (Directory Structure and File Control Block (FCB).
    - File-organization Module (Logical to Physical Block Translation).
    - Basic File System (Commands to Device Driver and manages the memory buffers and caches).
    - I/O Control (Device Drivers and Interrupt handlers).
  - Allocation Space on Disk can be done in three ways: Through Contiguous, Linked or Index Allocation.
  - Free Space Allocation Methods:
    - Bit Vectors and Linked Lists.
    - Optimized through Grouping, and Counting.
  - Directory using Linear List or Chained-Overflow Hash Table.
  - Directory Implemented as Network File System (NFS).
    - Using Mount and RPC protocol to execute File System operations on the Remote Server.
  -

Copyright @ 2009 John Wiley & Sons Inc.

Mass-Storage Structure

- Summary

[2 more weeks until end on June 16 midterm : 30 June]

- Disk Drives are the major Secondary-Storage I/O Devices.
  - Structured as large one-dimensional arrays of logical 512 disk blocks.
  - Attached to Computer Systems through I/O Ports or through Network Connections.
- Disk-Scheduling Algorithms improves the effective bandwidth, average response time and variance in response time.
  - SSTF, C-SCAN, LOOK, and C-LOOK.
- Operating System manages the disk blocks.
  - Formats the sectors on raw hardware and partitions the disk.
  - Boot Blocks is created.
  - File System created.
  - Raw disk partition or File System for Swap space .
- Reliability via Redundancy using Redundant Array of Independent Disks (RAID): Different levels of RAID.
- Tertiary Storage of Removable Disk and Tape Drives.
  - Operating System Support Removable Disk with File System interface and Tapes with specialilzed Device Driver.

Copyright @ 2009 John Wiley & Sons Inc.