CS370 Operating Systems

Colorado State University
Yashwant K Malaiya
Fall 2016 Lecture 42

Virtualization Review

Slides based on

- **Various sources**

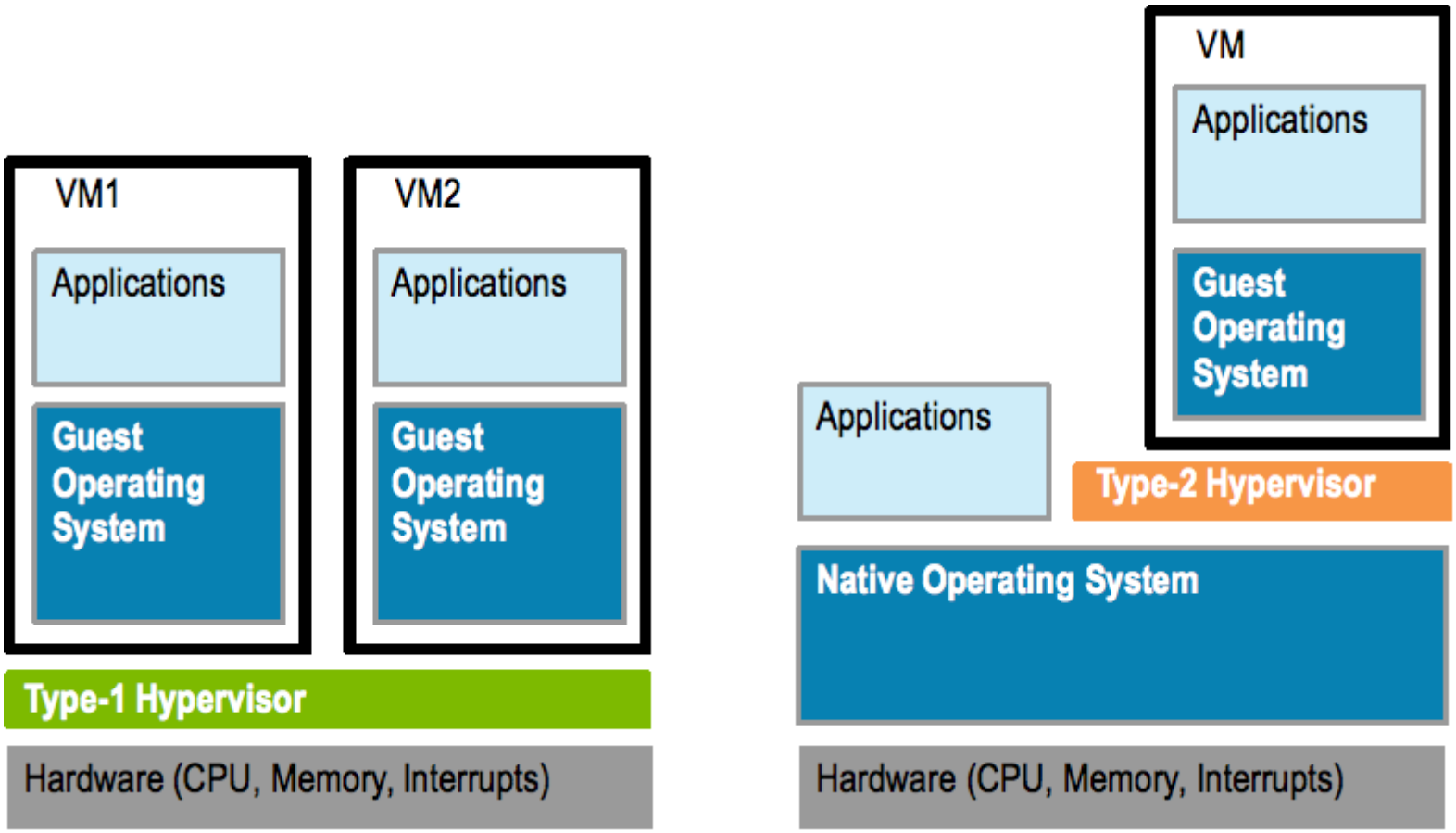
FAQ

- For VMM, is the CPU scheduling just like a single CPU managing resources?

Notes

- Poster Session Dec 9 10-11:30
  - See Assignments page, Canvas
  - You need to put it up and take it down later
  - At least one person should be available to provide an explanation
  - Prepare a 1 minute presentation
- Look at other posters. Need to review 3 posters, including one specified.
- Submit poster file: Dec 9, 5 PM, Reviews and Final paper Dec 13.
- Final Friday Dec 16 7:30AM.

Implementation of VMMs



Type 1: Bare metal Type 2: Hosted

https://microkerneldude.files.wordpress.com/2012/01/type1-vs-2.png

Colorado State University

Virtual Machine (VM) as a set of files

- Configuration file describes the attributes of the virtual machine containing
  - server definition,
  - how many virtual processors (vCPUs)
  - how much RAM is allocated,
  - which I/O devices the VM has access to,
  - how many network interface cards (NICs) are in the virtual server
  - the storage that the VM can access
- When a virtual machine is instantiated, additional files are created for logging, for memory paging etc.
- Copying a VM produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

CPU Scheduling

- One or more virtual CPUs (vCPUs) per guest
  - Can be adjusted throughout life of VM
- When enough CPUs for all guests
  - VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs (CPU overcommitment)
  - VMM can use scheduling algorithms to allocate vCPUs
  - Some add fairness aspect

Memory Management

Memory mapping:

- On a bare metal machine: Each process has its own virtual address space. OS uses page table/TLB to map Virtual page number (VPN) to Physical page number (PPN) (physical memory is shared). Each process has its own page table/TLB.

  VPN -> PPN

Memory Management

Memory mapping:

- On a bare metal machine:
  - VPN -> PPN
- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN ->MPN
- Where is this done?
  - In Full virtualization?
  - In Para virtualization?

Memory Management

- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN ->MPN

- Where is this done?
  - In Full virtualization? Has to be done by hypervisor. Guest OS knows nothing about MPN.
  - In Para virtualization? May be done by guest OS. It knows about hardware. Commands to VMM are "hypercalls"
- Full virtualization: PT/TLB updates are trapped to VMM. It needs to do VPN->PPN ->MPN. It can do VPN->MPN directly (VMware ESX)

Handling memory oversubscription

Oversubscription solutions:

- Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests
- Double-paging, the guest page table indicates a page is in a physical frame but the VMM moves some of those to disk.
- Install a pseudo-device driver in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
  - Balloon memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available.

Live Migration

Running guest can be moved between systems, without interrupting user access to the guest or its apps

- for resource management,
- maintenance downtime windows, etc

- Migration from source VMM to target VMM
  - Needs to migrate all pages gradually, without interrupting execution (details in next slide)
  - Eventually source VMM freezes guest, sends vCPU's final state, sends other state details, and tells target to start running the guest
  - Once target acknowledges that guest running, source terminates guest

Live Migration

- Migration from source VMM to target VMM
  - Source VMM establishes a connection with the target VMM
  - Target creates a new guest by creating a new VCPU, etc
  - Source sends all read-only memory pages to target
  - Source starts sending all read-write pages to the target, marking them as clean
    - repeats, as during that step some pages were modified by the guest and are now dirty.
  - Source VMM freezes guest, sends VCPU's final state, other state details, final dirty pages, and tells target to start running the guest
    - Once target acknowledges that guest running, source terminates guest

VIRTUAL APPLIANCES: “shrink-wrapped” virtual machines

- Developer can construct a virtual machine with
  - required OS, compiler, libraries, and application code
  - Freeze them as a unit … ready to run
- Customers get a complete working package
- Virtual appliances: “shrink-wrapped” virtual machines
- Amazon’s EC2 cloud offers many pre-packaged virtual appliances examples of *Software as a service*

CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 41

HW2

Slides based on

- Various sources

HW2

- Use solutions.

CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 41

Review

Slides based on

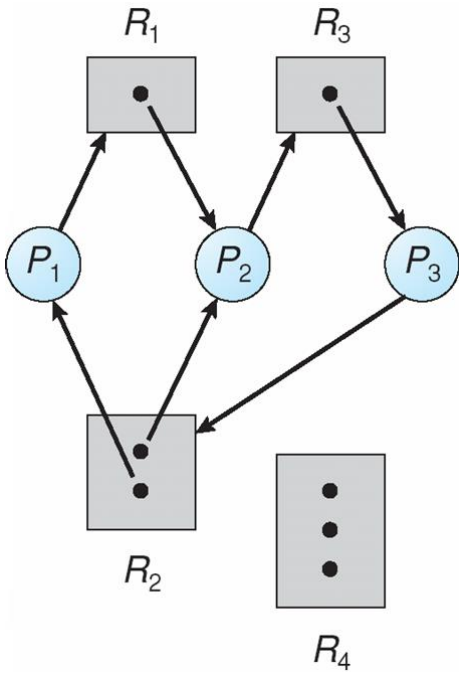
- Various sources

Classical Problems of Synchronization

- Bounded Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

Deadlocks

- System Model
  - Resource allocation graph, claim graph (for avoidance)
- Deadlock Characterization
  - Conditions for deadlock - mutual exclusion, hold and wait, no preemption, circular wait.
- Methods for handling deadlocks
  - Deadlock Prevention
  - Deadlock Avoidance
  - Deadlock Detection
  - Recovery from Deadlock
  - Combined Approach to Deadlock Handling



At this point, two minimal cycles exist in the system:
P1→ R1→ P2→ R3→ P3→ R2→ P1
P2→ R3→ P3→ R2→ P2
Processes *P1, P2,* and *P3* are deadlocked.

Deadlock Prevention

- If any one of the conditions for deadlock (with reusable resources) is denied, deadlock is impossible.
- Restrain ways in which requests can be made
  - Mutual Exclusion - cannot deny (important)
  - Hold and Wait - guarantee that when a process requests a resource, it does not hold other resources.
  - No Preemption
    - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, the process releases the resources currently being held.
  - Circular Wait
    - Impose a total ordering of all resource types.

Deadlock Avoidance

- Requires that the system has some additional apriori information available.
  - Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- Computation of Safe State
  - When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. Sequence <P1, P2, …Pn> is safe, if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by Pj with j<i.
  - Safe state - no deadlocks, unsafe state - possibility of deadlocks
  - Avoidance - system will never reach unsafe state.

Example: 12 Tape drives available in the system

	Max need	Current need
P0	10	5
P1	4	2
P2	9	2

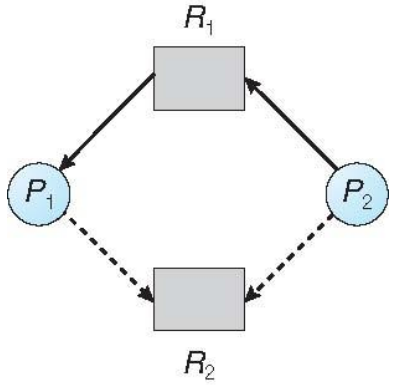
At T0:
3 drives available

Safe sequence
<P1, P0 , P2>

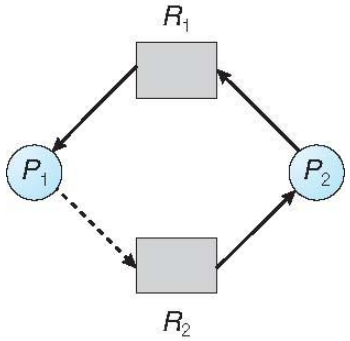
- At time **T0** the system is in a safe state
  - P1 can be given 2 tape drives
  - When P1 releases its resources; there are 5 drives
  - P0 uses 5 and subsequently releases them (# 10 now)
  - P2 can then proceed.

Algorithms for Deadlock Avoidance

- Resource allocation graph algorithm
  - only one instance of each resource type



- Banker's algorithm
  - Used for multiple instances of each resource type.
  - Data structures required
    - Available, Max, Allocation, Need
  - Safety algorithm
  - resource request algorithm for a process.

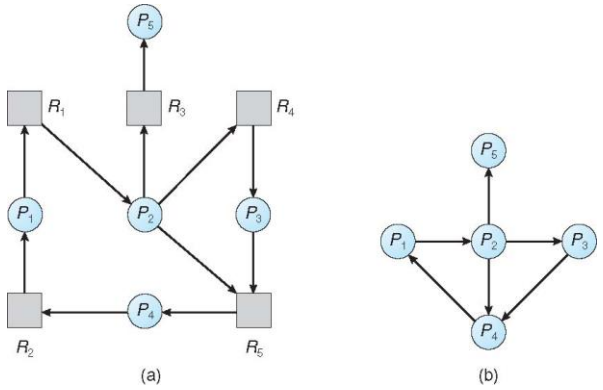


Unsafe state

Suppose *P2* requests *R2.* Although *R2* is currently free, we cannot allocate it to *P2,* since this action will create a cycle getting system is in an unsafe state. If *P1* requests *R2,* and *P2* requests *R1,* then a deadlock will occur.

Deadlock Detection

- Allow system to enter deadlock state
- Detection Algorithm
  - Single instance of each resource type
    - use wait-for graph
  - Multiple instances of each resource type
    - variation of banker's algorithm
- Recovery Scheme
  - Process Termination
  - Resource Preemption



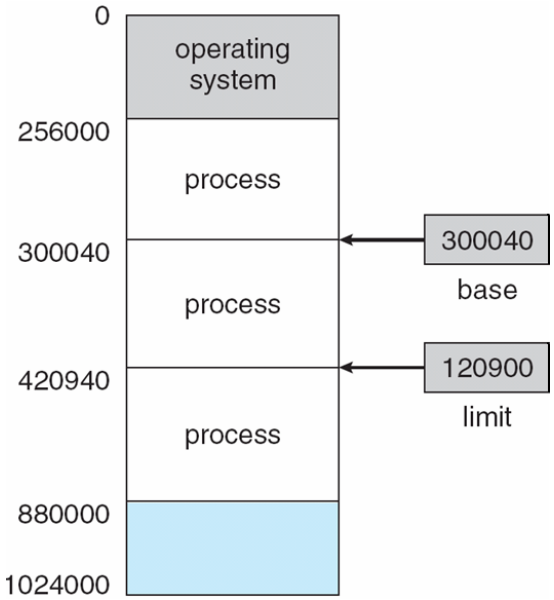
Resource-Allocation Graph

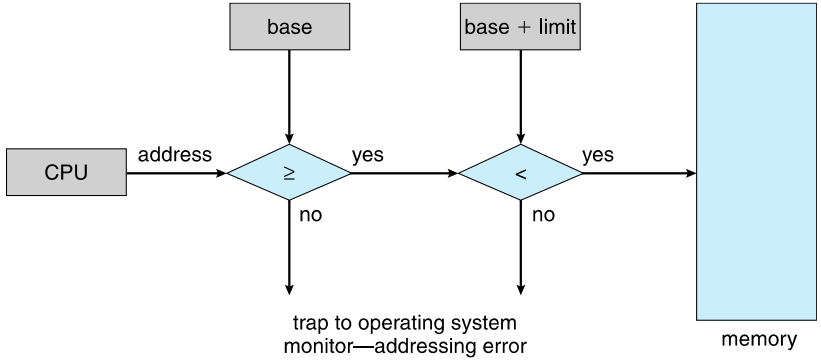
Corresponding wait-for graph

3 cycles. Deadlock.

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space for a process
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user
- Base: **Smallest** legal physical address
- Limit: Size of the **range** of physical address
- Eg: Base = 300040 and limit = 120900
- Legal: 300040 to (300040 + 120900 -1) = 420939





Colorado State University

Binding of instructions and data to memory

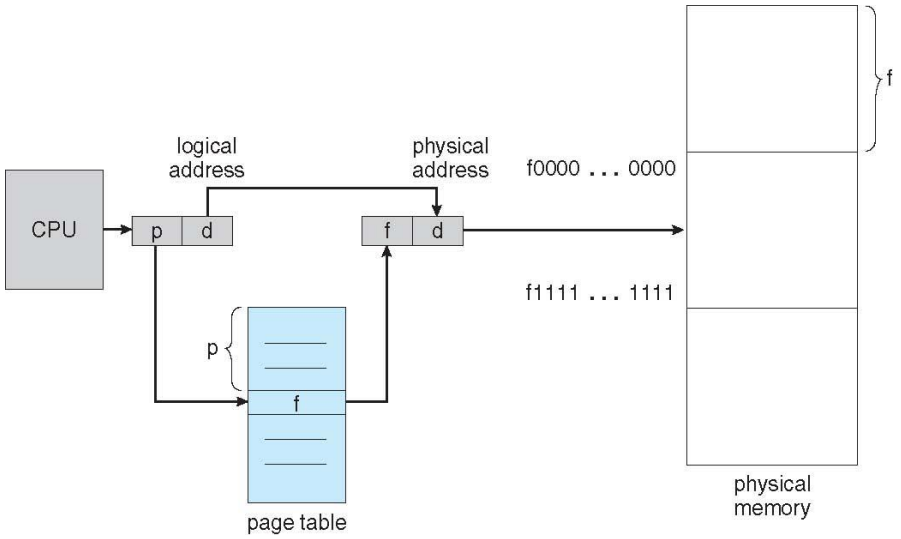
- Address binding of instructions and data to memory addresses can happen at three different stages.
  - Compile time, Load time, Execution time
- Other techniques for better memory utilization
  - Dynamic Loading - Routine is not loaded until it is called.
  - Dynamic Linking - Linking postponed until execution time
  - Overlays - Keep in memory only those instructions and data that are needed at any given time
  - Swapping - A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution
- MMU - Memory Management Unit
  - Hardware device that maps virtual to physical address.

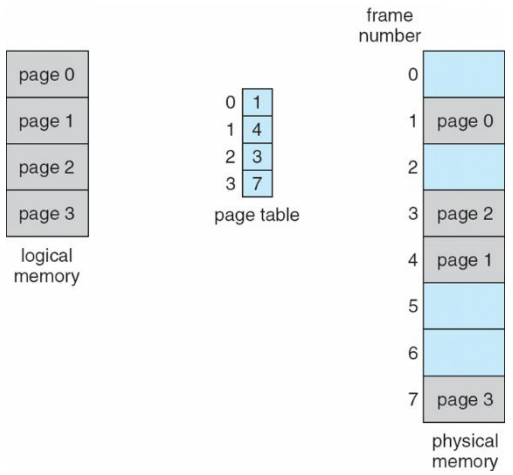
Dynamic Storage Allocation Problem

- How to satisfy a request of size n from a list of free holes.
  - First-fit
  - Best-fit
  - Worst-fit
- Fragmentation
  - External fragmentation
    - total memory space exists to satisfy a request, but it is not contiguous.
  - Internal fragmentation
    - allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
  - Reduce external fragmentation by compaction

Page Table Implementation

- Page table is kept in main memory
  - Page-table base register (PTBR) points to the page table.
  - Page-table length register (PTLR) indicates the size of page table.
  - Every data/instruction access requires 2 memory accesses.
    - One for page table, one for data/instruction
    - Two-memory access problem solved by use of special fast-lookup hardware cache (i.e. cache page table in registers)
      - associative registers or translation look-aside buffers (TLBs)

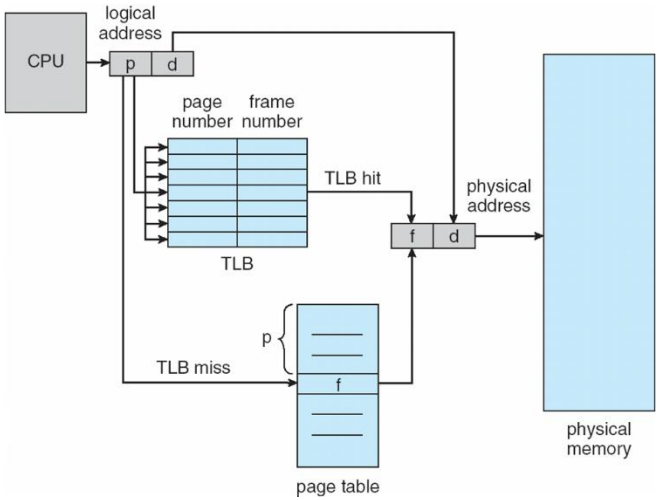




Effective Access Time

- Associative Lookup = ε time unit
  - Can be < 10% of memory access time
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access
- **Effective Access Time** (**EAT**)

  EAT = (100 + ε) α + (200 + ε)(1 – α)



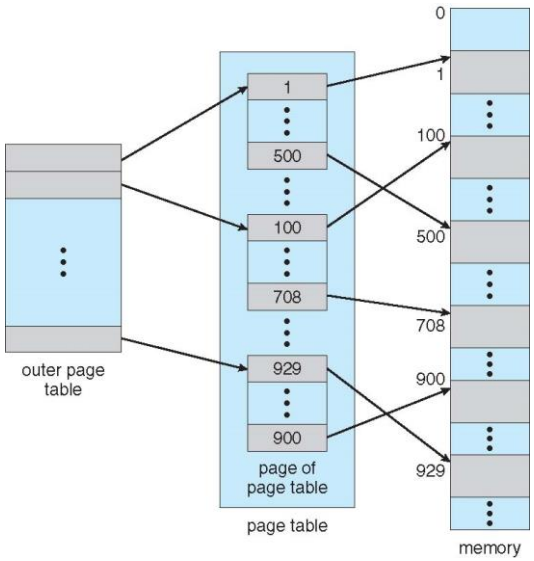
Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access

- EAT = 0.80 x 120 + 0.20 x 220 = 140ns

- Consider more realistic hit ratio -> α = 99%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.99 x 120 + 0.01 x 220 = 121ns

Paging Methods

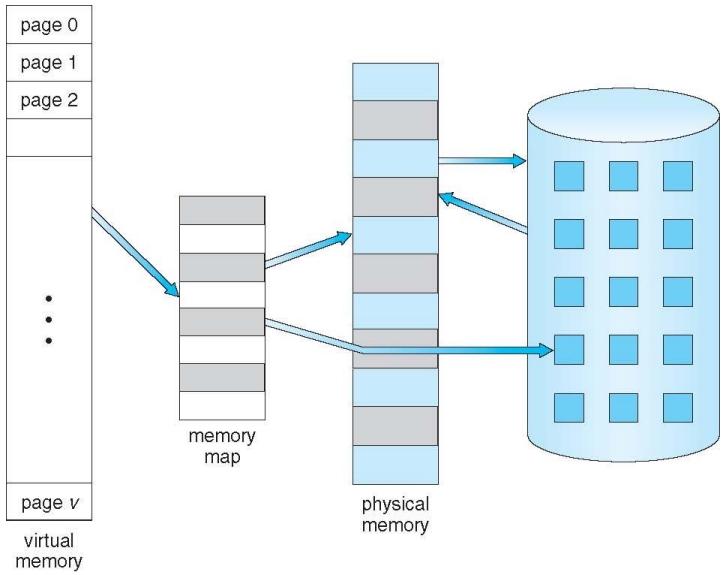
- Multilevel Paging
  - Each level is a separate table in memory
  - converting a logical address to a physical one may take 4 or more memory accesses.
  - Caching can help performance remain reasonable.
- Hashed page table
- Inverted Page Tables
  - One entry for each real page of memory. Entry consists of virtual address of page in real memory with information about process that owns page.
- Shared Pages
  - Code and data can be shared among processes. Reentrant (non self-modifying) code can be shared. Map them into pages with common page frame mappings



page number		page offset
p_1	p_2	d
12	10	10

Virtual Memory

- Virtual Memory
  - Separation of user logical memory from physical memory.
  - Only *PART* of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Need to allow pages to be swapped in and out.
- Virtual Memory can be implemented via
  - Paging
  - Segmentation

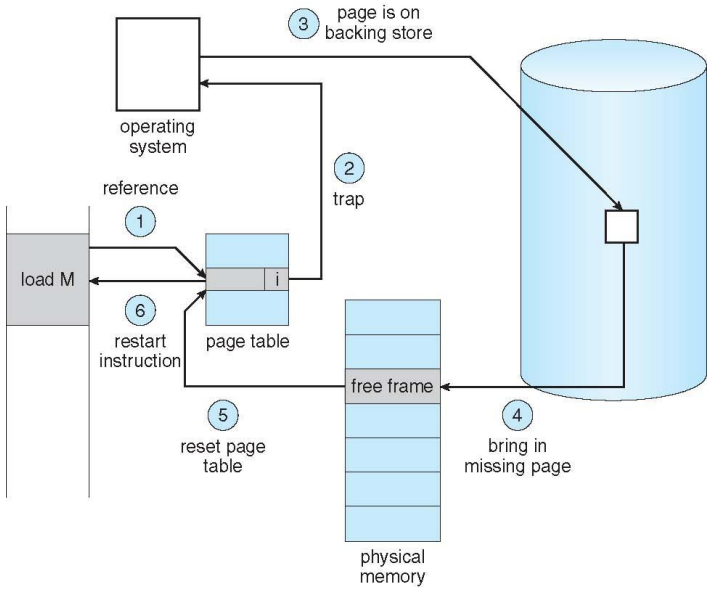


Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less Memory needed
  - Faster response
  - More users
- The first reference to a page will trap to OS with a page fault.
- OS looks at another table to decide
  - Invalid reference - abort
  - Just not in memory.

Page fault:

1. Find free frame
2. Get page into frame via scheduled disk operation
3. Reset tables to indicate page now in memory
   Set validation bit = **v**
4. Restart the instruction that caused the page fault

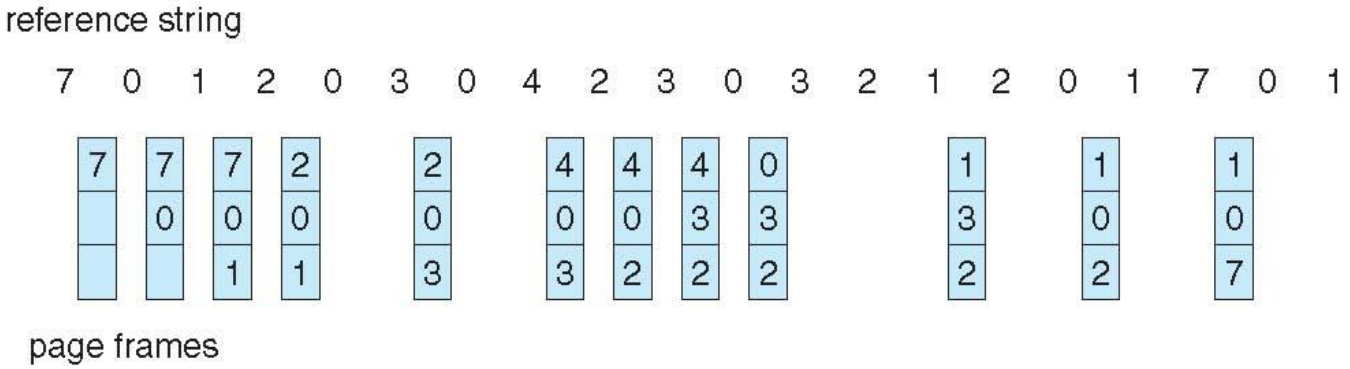


Page Replacement Strategies

- The Principle of Optimality
  - Replace the page that will not be used again the farthest time into the future.
- Random Page Replacement
  - Choose a page randomly
- FIFO - First in First Out
  - Replace the page that has been in memory the longest.
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time.
  - LRU Approximation Algorithms - reference bit, second-chance etc.
- LFU - Least Frequently Used
  - Replace the page that is used least often.
- NUR - Not Used Recently
  - An approximation to LRU
- Working Set
  - Keep in memory those pages that the process is actively using

Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



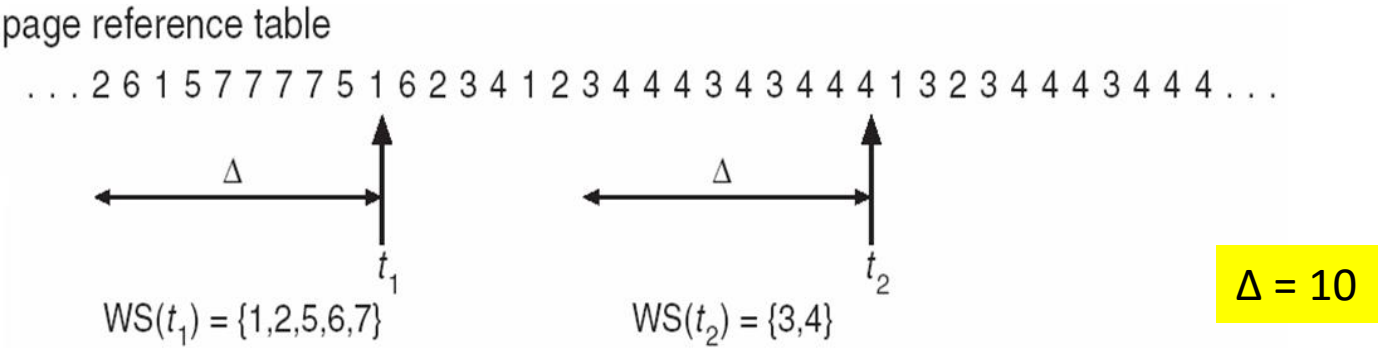
- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- Approximate Implementations:
  - Counter implementation time of use field
  - Stack implementation
  - Reference bit
  - Second chance

Allocation of Frames

- Single user case is simple
  - User is allocated any free frame
- Problem: Demand paging + multiprogramming
  - Each process needs minimum number of pages based on instruction set architecture.
  - Two major allocation schemes:
    - Fixed allocation - (1) equal allocation (2) Proportional allocation.
    - Priority allocation - May want to give high priority process more memory than low priority process.
  - Global vs local allocation

Working-Set Model

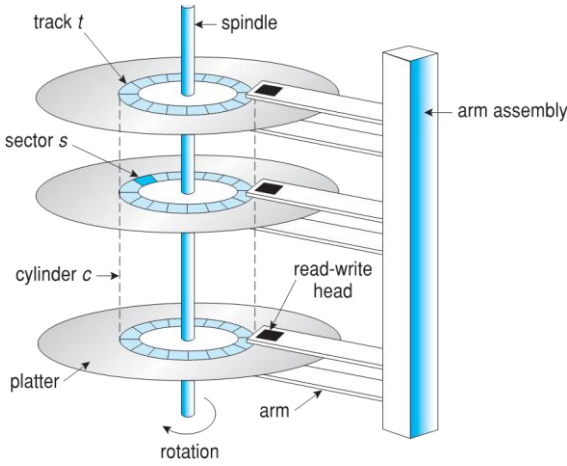
- $\Delta \equiv$ **working-set window** $\equiv$ a fixed number of page references
  Example: 10,000 instructions



- ***WSS*$_i$ (working set of Process *P*$_i$)** =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- ***D* = Σ *WSS*$_i$ $\equiv$ total demand frames**
  - Approximation of locality
- **if *D* > *m* $\Rightarrow$ Thrashing**
- **Policy** if *D* > m, then suspend or swap out one of the processes

Hard Disk Performance

- **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time
  - Transfer time = 4KB / 1Gb/s = 4x8K/G = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms



Disk Scheduling

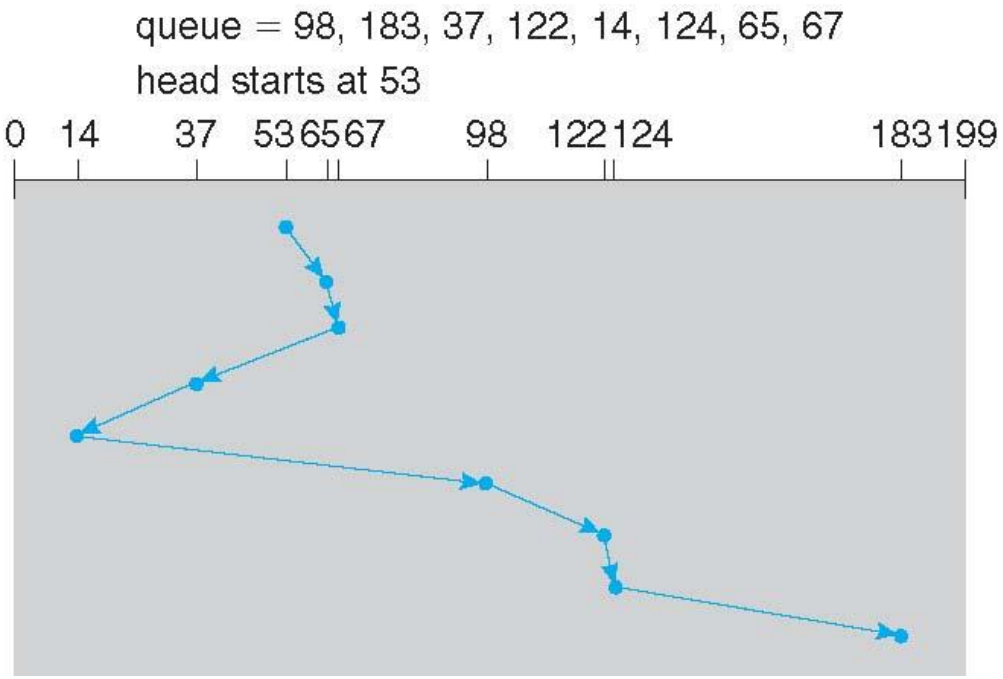
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (cylinders 0-199)

  98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53 (head is at cylinder 53)
- FCFS (First come first served)
- SSTF Shortest Seek Time First
- SCAN, C-SCAN, C-LOOK,

SSTF **Shortest Seek Time First**

- **Shortest Seek Time First** selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- total head movement of **236** cylinders

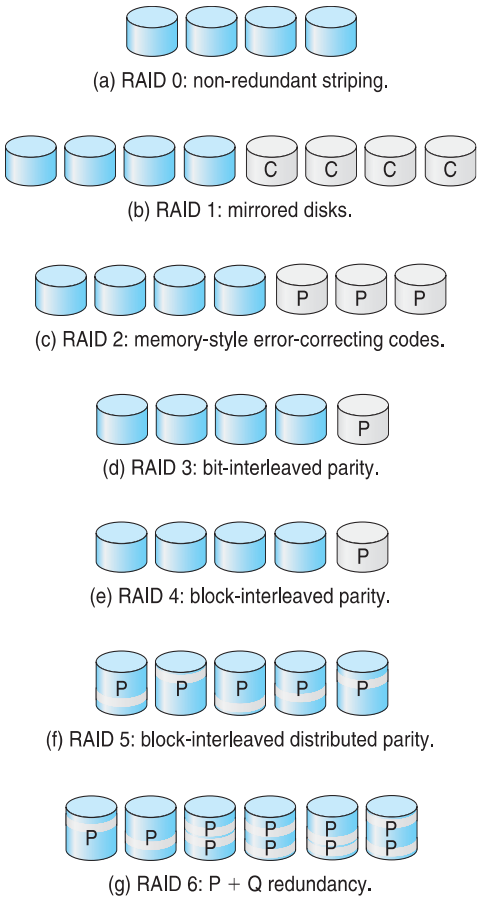


RAID Techniques

- **Striping** uses multiple disks in parallel by splitting data: higher performance, no redundancy (ex. RAID 0)
- **Mirroring** keeps duplicate of each disk: higher reliability (ex. RAID 1)
- **Block parity:** **One Disk hold** parity block for other disks. A failed disk can be rebuilt using parity. Wear leveling if interleaved (RAID 5, double parity RAID 6).
- Ideas that did not work: Bit or byte level level striping (RAID 2, 3) Bit level Coding theory (RAID 2), dedicated parity disk (RAID 4).
- Nested Combinations:
  - RAID 01: Mirror RAID 0
  - RAID 10: Multiple RAID 1, striping
  - RAID 50: Multiple RAID 5, striping
  - others

Not common:
RAID 2, 3,4

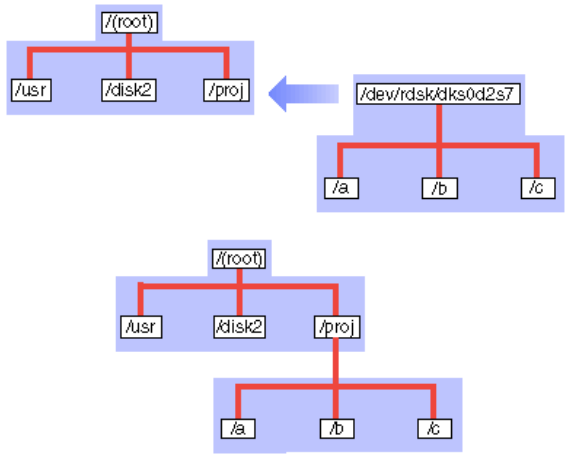
Most common
RAID 5



Colorado State University

File-System Implementation

- File System Structure
  - File System resides on secondary storage (disks).To improve I/O efficiency, I/O transfers between memory and disk are performed in blocks. Read/Write/Modify/Access each block on disk.
  - File System Mounting - File System must be mounted before it can be available to process on the system. The OS is given the name of the device and the mount point.
- Allocation Methods
- Free-Space Management
- Directory Implementation
- Efficiency and Performance, Recovery



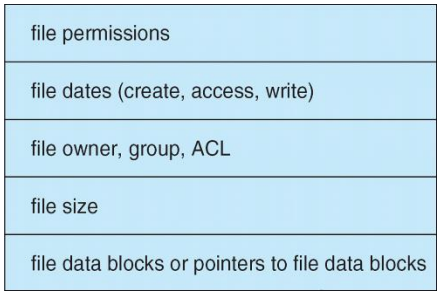
File Systems

- Many file systems, sometimes several within an operating system
  - Each with its own format
    - Windows has FAT (1977), FAT32 (1996), NTFS (1993)
    - Linux has more than 40 types, with **extended file system** (1992) ext2 (1993), ext3 (2001), ext4 (2008);
    - plus distributed file systems
    - floppy, CD, DVD Blu-ray
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE, xFAT

On-disk File-System Structures

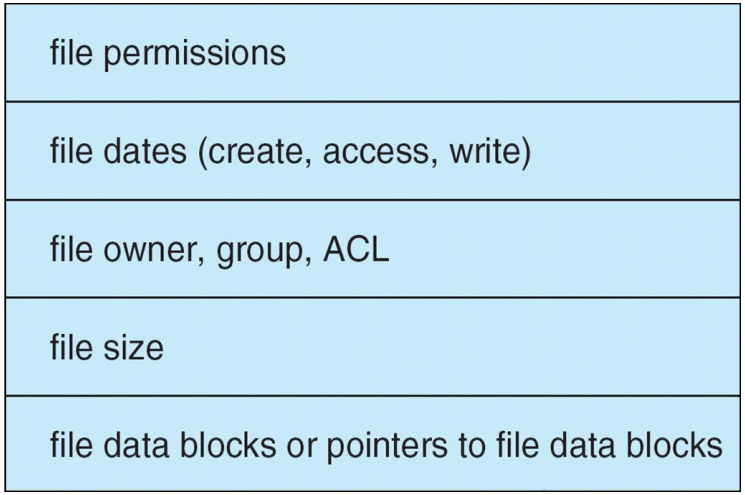
1. **Boot control block** contains info needed by system to boot OS from that volume
   - Needed if volume contains OS, usually first block of volume
2. **Volume control block (superblock UFS or master file tableNTFS)** contains volume details
   - Total # of blocks, # of free blocks, block size, free block pointers or array
3. Directory structure organizes the files
   - File Names and inode numbers UFS, master file table NTFS
4. Per-file **File Control Block (FCB or "inode")** contains many details about the file
   - Indexed using inode number; permissions, size, dates UFS
   - master file table using relational DB structures NTFS

Volume: logical disk drive, perhaps a partition



File-System Implementation (Cont.)

4. Per-file **File Control Block (FCB or "inode")** contains many details about the file
   - Indexed using inode number; permissions, size, dates UFS
   - master file table using relational DB structures NTFS



In-Memory File System Structures

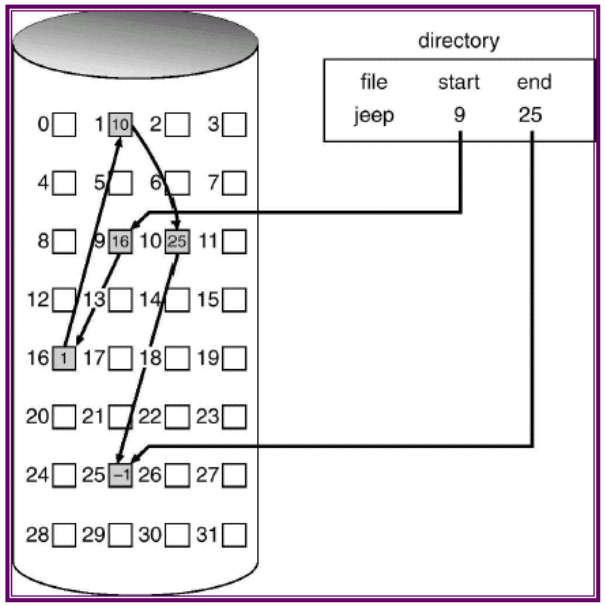
- An in-memory **mount table** contains information about each mounted volume.
- An in-memory directory-structure cache holds the directory information of recently accessed directories.
- The system-wide open-file table contains a copy of the FCB of each open file, as well as other information.
- The per-process open file table contains a pointer to the appropriate entry in the system-wide open-file table
- Plus buffers hold data blocks from secondary storage

Open returns a file handle (file descriptor) for subsequent use

- Data from read eventually copied to specified user process memory address

Allocation of Disk Space

- Low level access methods depend upon the disk allocation scheme used to store file data
  - Contiguous Allocation
    - Each file occupies a set of contiguous blocks on the disk. Dynamic storage allocation problem. Files cannot grow.
  - Linked List Allocation
    - Each file is a linked list of disk blocks. Blocks may be scattered anywhere on the disk. Not suited for random access.
    - Variation - FILE ALLOCATION TABLE (FAT) mechanisms
  - Indexed Allocation
    - Brings all pointers together into the index block. Need index table. Can link blocks of indexes to form multilevel indexes.

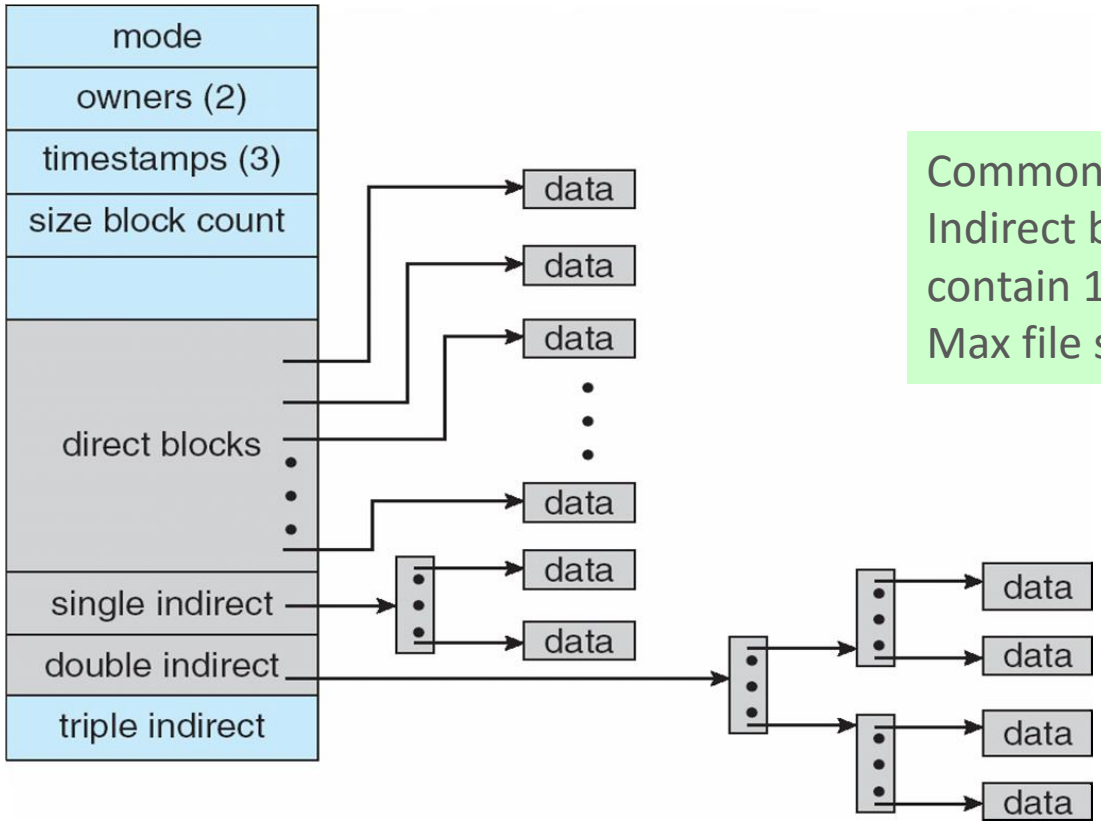


Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses

Volume block: Table with file names Points to this

Inode (file control block)

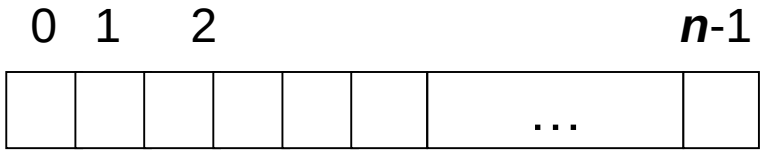


Common: 12+3 Indirect block could contain 1024 pointers. Max file size: k.k.k.4k+

More index blocks than can be addressed with 32-bit file pointer

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Approaches: i. Bit vector ii. Linked list iii. Grouping iv. Counting**
- **Bit vector** or **bit map** (***n*** blocks)



$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \text{ free} \\ 0 \Rightarrow block[i] \text{ occupied} \end{cases}$$

Block number calculation

(number of bits per word) *(number of 0-value words) + offset of first 1 bit

```
00000000
00000000
00111110
..
```

CPUs have instructions to return offset within word of first "1" bit