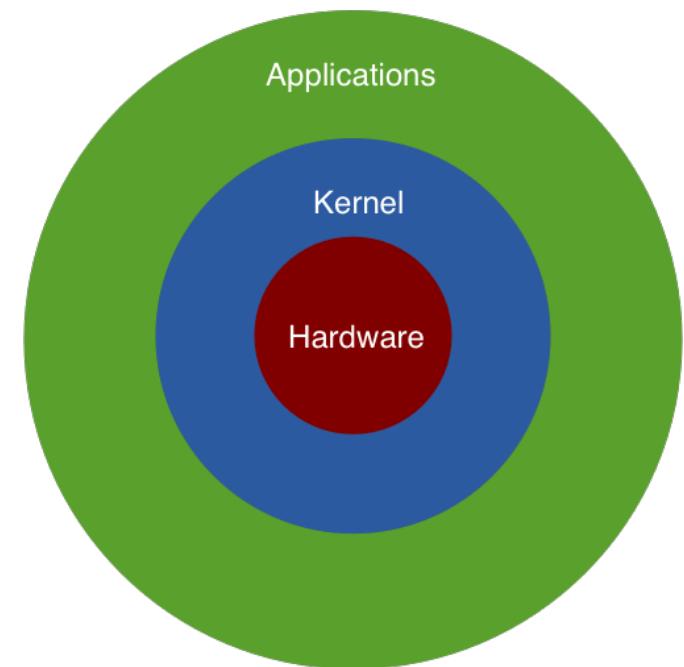


Final Review Day 1

Fall 2018

Kernel

Kernel - The part of the OS that implements basic functionality and is always resident in memory.



The Operating System's Role

- » Provide the user with a cleaner model of the computer
 - An abstracted interface to the resources
- » Manage the resources

Interrupts

- Mechanism used by the OS to signal the system that a high-priority event has occurred that requires immediate attention.
 - I/O drives a lot of interrupts. Mouse movements, disk reads, etc
- The controller causing the interrupt places the interrupt number in an interrupt register. The OS must then take action

Interrupt Vector

- Normal technique for handling interrupts is a data structure called the interrupt vector.
- One entry for each interrupt.
- Each entry contains the address for the interrupt service routine.
- Some small hardware devices don't provide an interrupt system and instead uses an event loop. This is known as a status-driven system.

I/O Devices

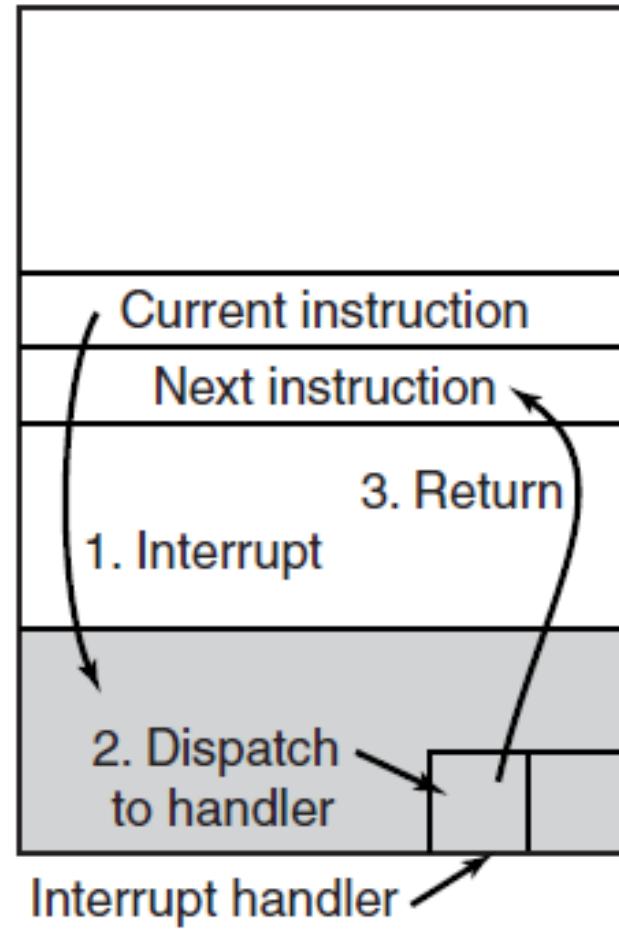
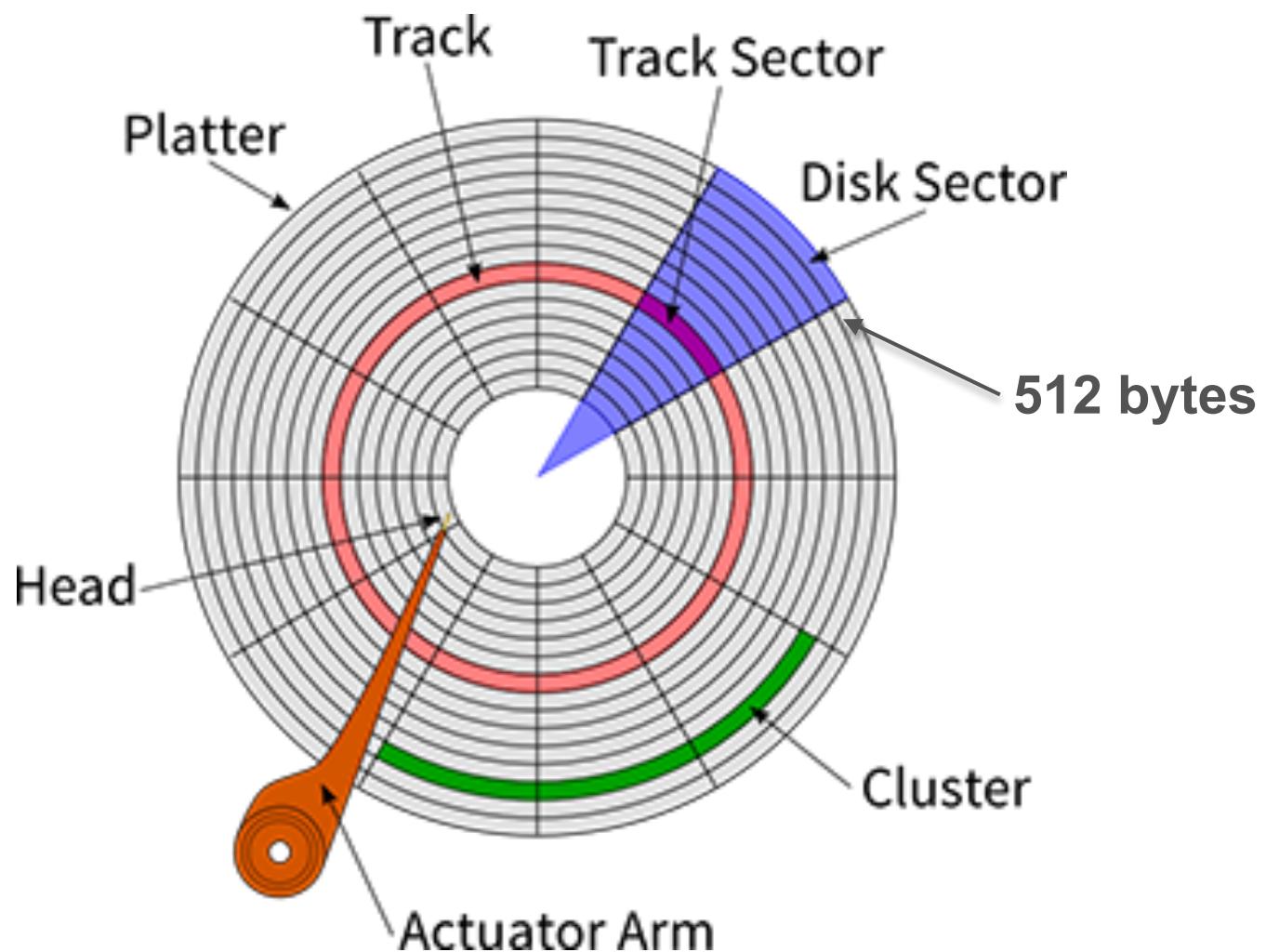
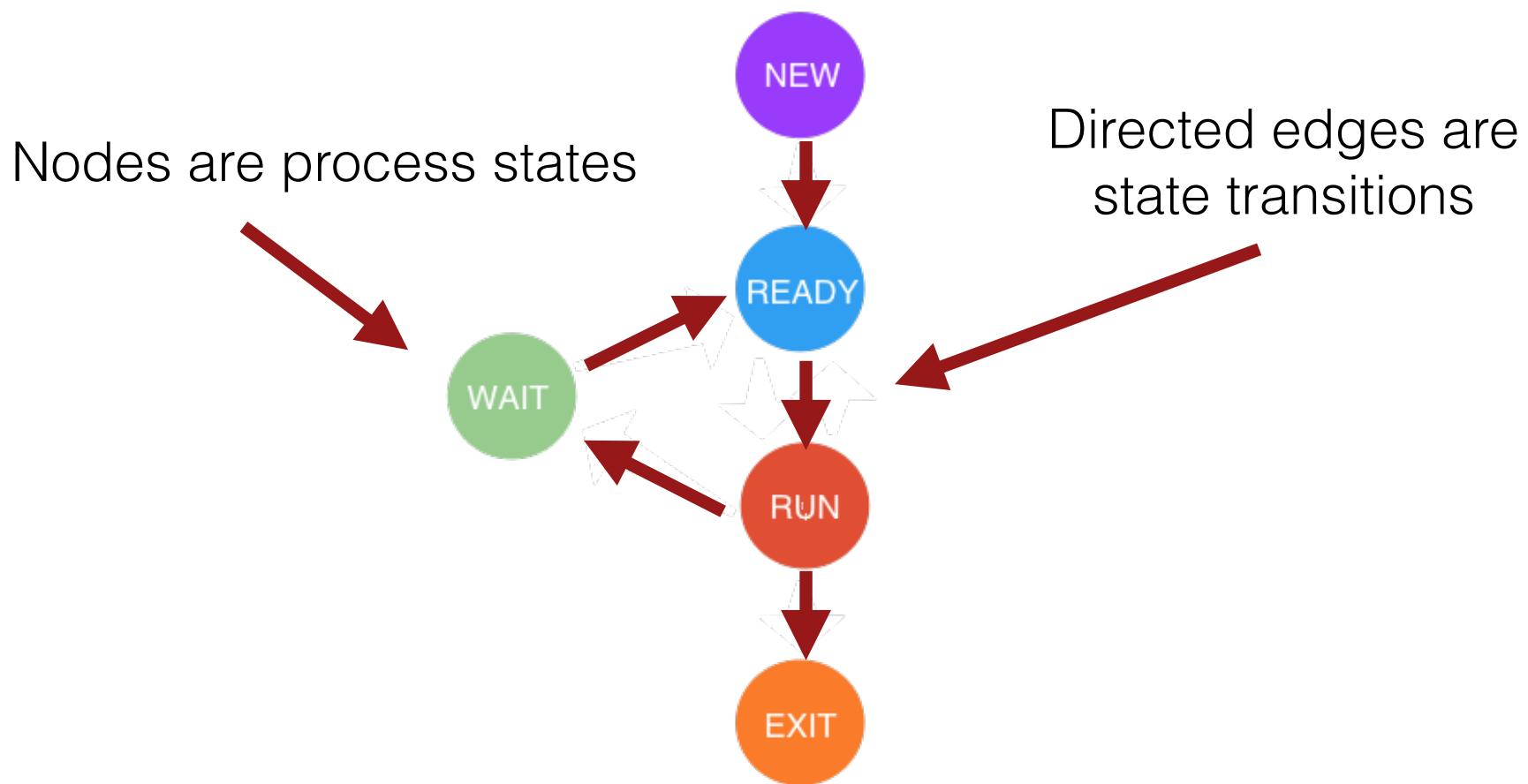


Figure 1-11. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Disks



Process State Diagram



Process Creation

- Processes are created when an existing process calls the `fork()` function
- New process created by `fork` is called the *child process*
- `fork()` is a function that is called once but returns twice
 - Only difference in the return value. Returns 0 in the child process and the child's PID in the parent process

Parent and children PIDs

- Why does fork return the child's PID to parent processes but it returns 0 to the children?
- Provides a way to determine which process you are in.
- Processes can only have one parent. To determine the parent PID you can call getppid().
- Processes can have multiple children so there is no way for a function to give a parent its child PID

Process Creation

- After fork() both the parent and child continue executing with the instruction that follows the call to fork().
- The child process is a copy of the parent process. The child gets:
 - copy of the parent's data space
 - copy of the parent's heap
 - copy of the parent's stack
 - text segment if it's read-only

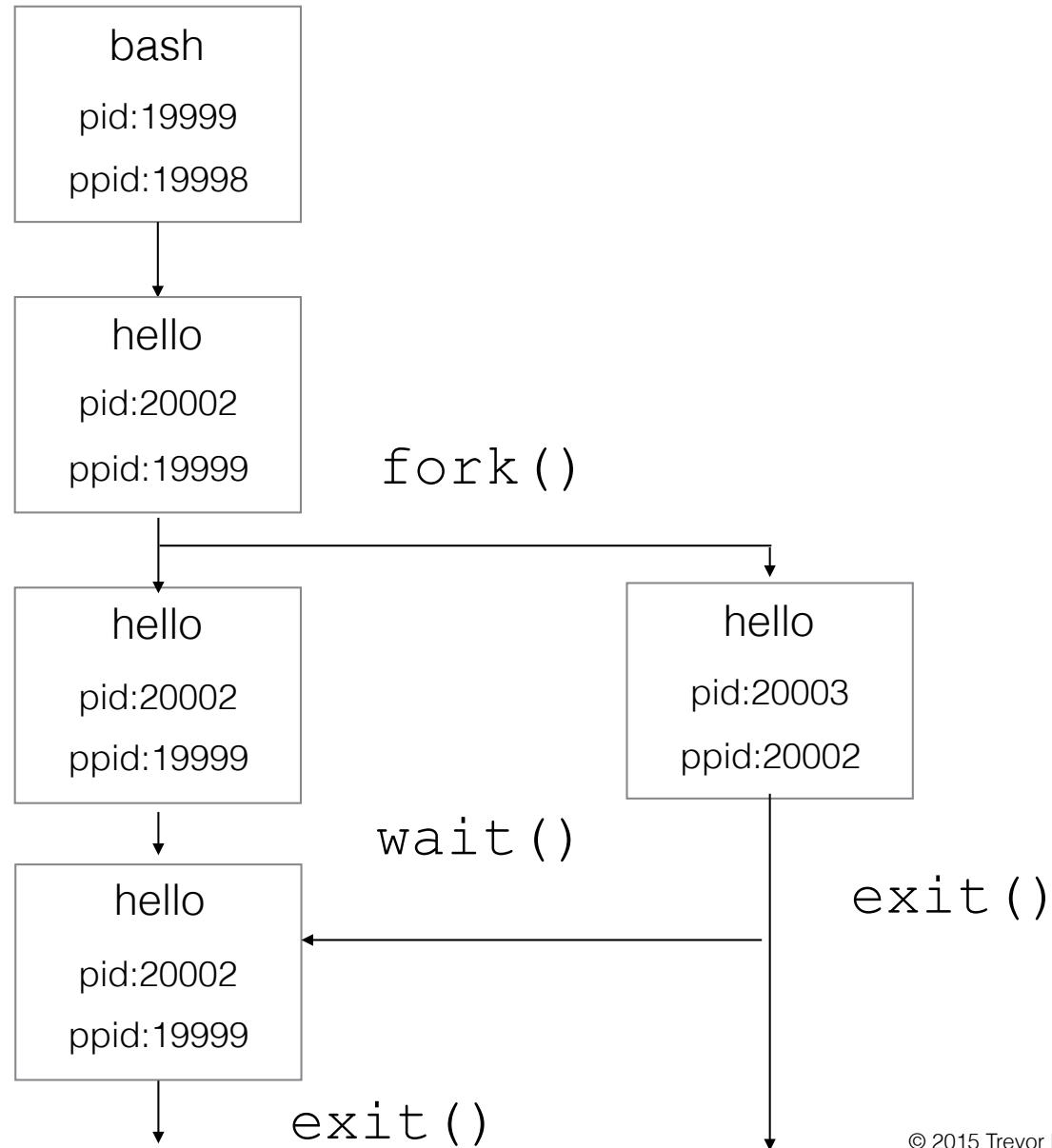
fork() code example

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t pid = fork();

    if (pid == -1) {
        // When fork() returns -1, an error happened.
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) {
        // When fork() returns 0, we are in the child process.
        printf("Hello from the child process!\n");
        fflush(NULL);
        exit(EXIT_SUCCESS);
    }
    else {
        // When fork() returns a positive number, we are in the parent process
        // and the return value is the PID of the newly created child process.
        int status;
        (void)waitpid(pid, &status, 0);
        printf("Hello form the parent process!");
        fflush(NULL);
    }
    return EXIT_SUCCESS;
}
```

Hello World



fork() code example

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t pid = fork();

    if (pid == -1) {
        // When fork() returns -1, an error happened.
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) {
        // When fork() returns 0, we are in the child process.
        printf("Hello from the child process!\n");
        fflush(NULL); ←
        exit(EXIT_SUCCESS);
    }
    else {
        // When fork() returns a positive number, we are in the parent process
        // and the return value is the PID of the newly created child process.
        int status;
        (void)waitpid(pid, &status, 0);
        printf("Hello form the parent process!");
        fflush(NULL); ←
    }
    return EXIT_SUCCESS;
}
```

I/O is buffered.
Make sure to
use flush()

`wait()` and `waitpid()`

- When a process terminates the parent is notified by the operating system sending a SIGCHLD signal.
 - Default handling is to ignore the signal
- Child termination is asynchronous
- POSIX provides two system calls wait and waitpid

exec family

```
int execl(char const *path, char const *arg0, ...);  
int execle(char const *path, char const *arg0, ..., char const *envp[]);  
int execlp(char const *file, char const *arg0, ...);  
int execv(char const *path, char const *argv[]);  
int execve(char const *path, char const *argv[], char const *envp[]);  
int execvp(char const *file, char const *argv[]);
```

Meaning of the letters after exec:

- I – A list of command-line arguments are passed to the function.
- e – An array of pointers to environment variables is passed to the new process image.
- p – The PATH environment variable is used to find the file named in the path argument.
- v – Command-line arguments are passed to the function as an array of pointers.

exec code example

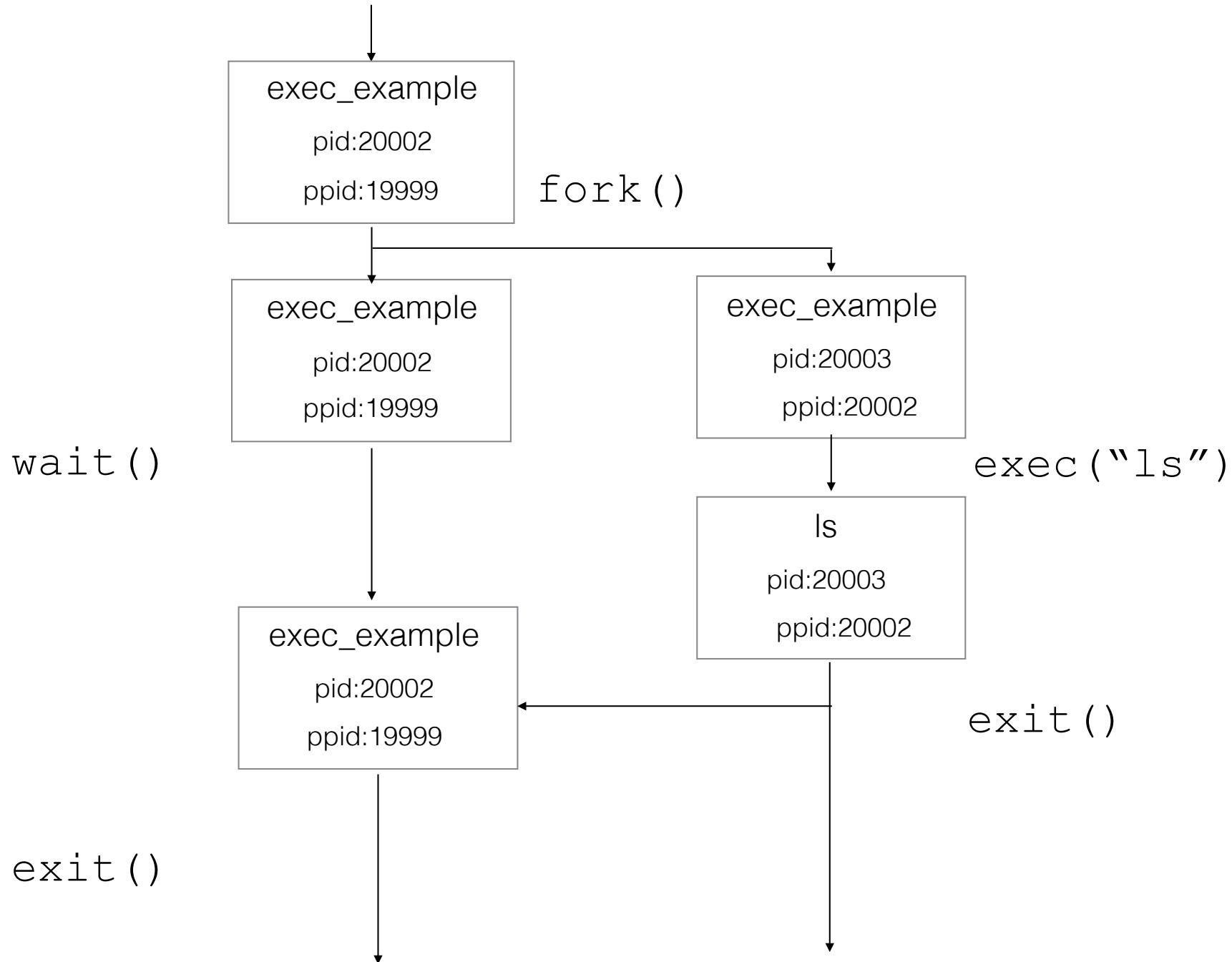
```
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>

int main(void)
{
    pid_t child_pid = fork();
    int status;

    if (child_pid == 0)
    {
        execl("/bin/ls", "ls", NULL );
        exit(0);
    }

    // Wait for the child to exit
    waitpid( child_pid, &status, 0 );

    return 0;
}
```



waitpid() code example

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    pid_t child_pid = fork();
    int status;

    if (child_pid == 0)
    {
        // Sleep for a second
        sleep(1);

        // Intentionally SEGFAULT the child process
        int *p = NULL;
        *p = 1;

        exit(0);
    }

    // Wait for the child to exit
    waitpid( child_pid, &status, 0 );

    // See if the child was terminated by a signal
    if( WIFSIGNALED( status ) )
    {
        // Print the signal that the child terminated with
        printf("Child returned with status %d\n", WTERMSIG( status ) );
    }

    return 0;
}
```

exec functions

- When exec is called the new program, specified by exec, completely replaces the running process.
 - text, data, heap and stack are all replaced
 - PID stays the same since it's not a new process

Kernel Mode v. User Mode

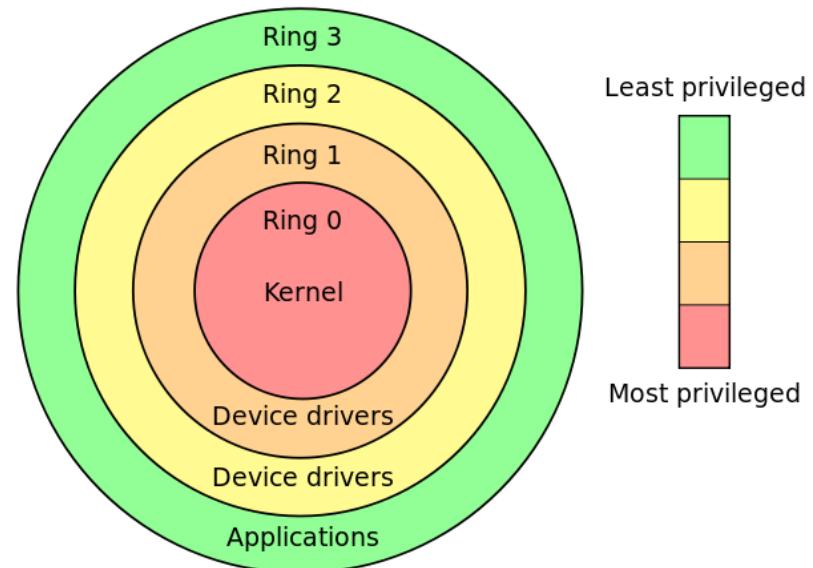
- » A process is executing either in user mode, or in kernel mode. Depending on which privileges, address space a process is executing in, we say that it is either in user space, or kernel space.
- » When executing in user mode, a process has normal privileges and can and can't do certain things. When executing in kernel mode, a process has every privilege, and can do anything.
- » Processes switch between user space and kernel space using system calls.

Kernel Mode v. User Mode

- » These two modes aren't just labels; they're enforced by the CPU hardware.
- » If code executing in User mode attempts to do something outside its purview such as accessing a privileged CPU instruction or modifying memory that it has no access to:
 - Trappable exception is thrown. Instead of your entire system crashing, only that particular application crashes.

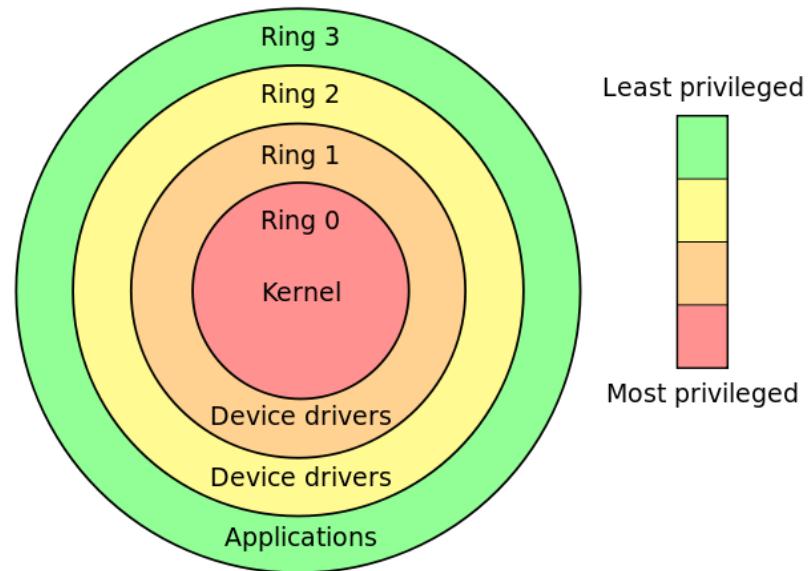
x86 Protection Rings

- » Four privilege levels or rings, numbered from 0 to 3, with ring 0 being the most privileged and 3 being the least.
- » Rings 1 and 2 weren't used in practice.
- » VM's changed that



x86 Protection Rings

- » Programs that run in Ring 0 can do anything with the system.
- » Code that runs in Ring 3 should be able to fail at any time without impact to the rest of the computer system.



CPU Rings and Privilege

- CPU privilege level has nothing to do with operating system users.
- Whether you’re root, Administrator, guest, or a regular user, it does not matter.
- All user code runs in ring 3 and all kernel code runs in ring 0, regardless of the OS user on whose behalf the code operates.

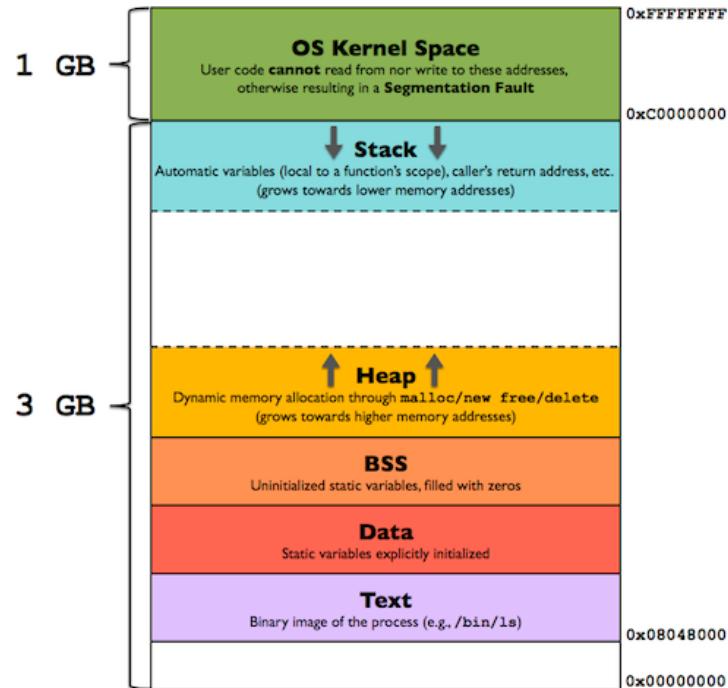
CPU Rings and Privilege

- » Due to restricted access to memory and I/O ports, user mode can do almost nothing to the outside world without calling on the kernel.
 - It can't open files, send network packets, print to the screen, or allocate memory.
 - User processes run in a severely limited sandbox set up by ring zero.

Address Spaces

- An address space is the set of addresses in RAM that a process can use.
- Multiple programs in memory need OS to partition the available memory
 - Keep programs from interfering with each other and OS.

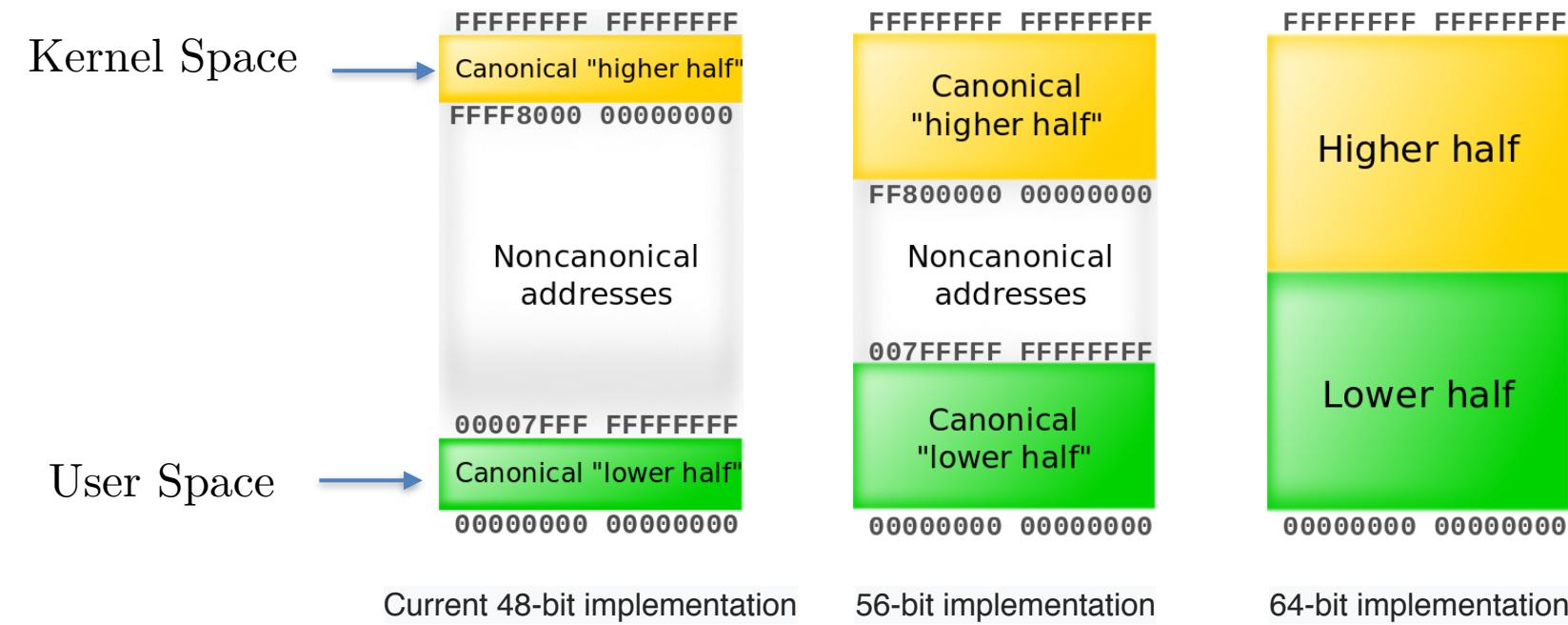
Address Spaces



32-bit address = 2^{32} = 4GB of address space

- Kernel gets upper 1 GB

Address Spaces



Only 48-bits implemented in current 64-bit architecture

Address Spaces

- » What happens if the process has more address space than free main memory ?
 - Virtual memory
 - OS abstracts address space
 - Decouples programs address space from physical memory

System Calls

- How do our programs utilize the resources controlled by the OS or communicate with other process?
- Because user mode software can not access hardware devices directly, they must notify the operating system in order to complete system tasks. This includes displaying text, obtaining input from user, printing a document, etc.

System Calls

- Instead of directly calling a section of code the system call instruction issues an interrupt.
- By not allowing the application to execute code freely the operating system can verify that the application has appropriate privileges to call the function.
 - Only system calls enter the kernel. Procedure calls do not.

System Calls (1)

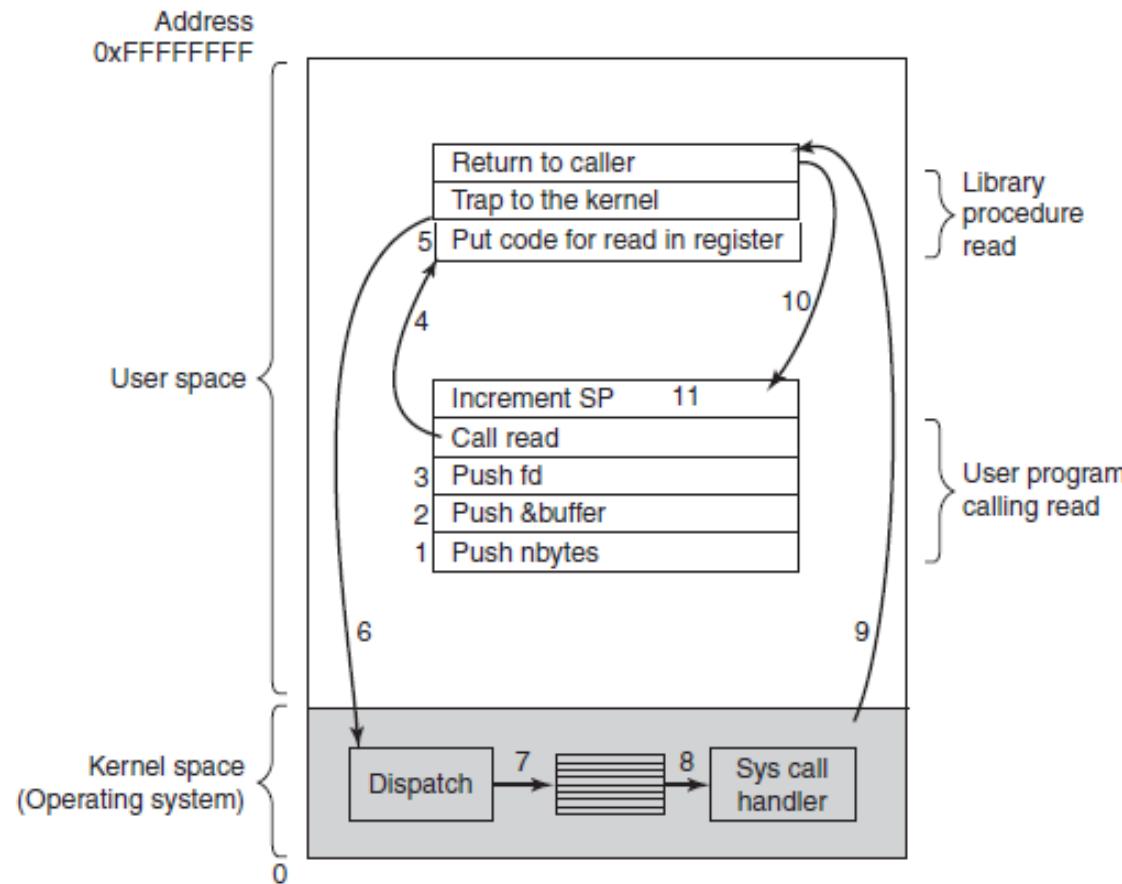
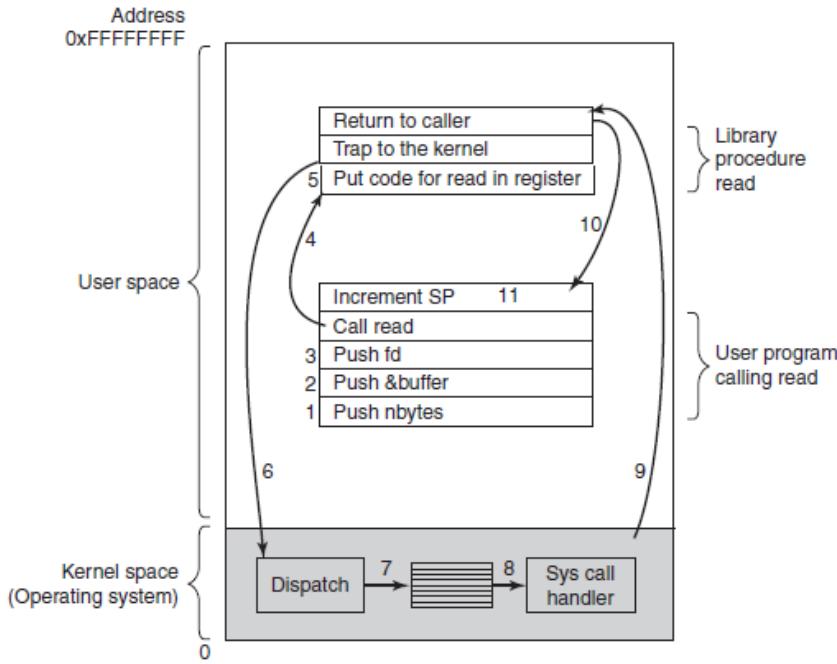


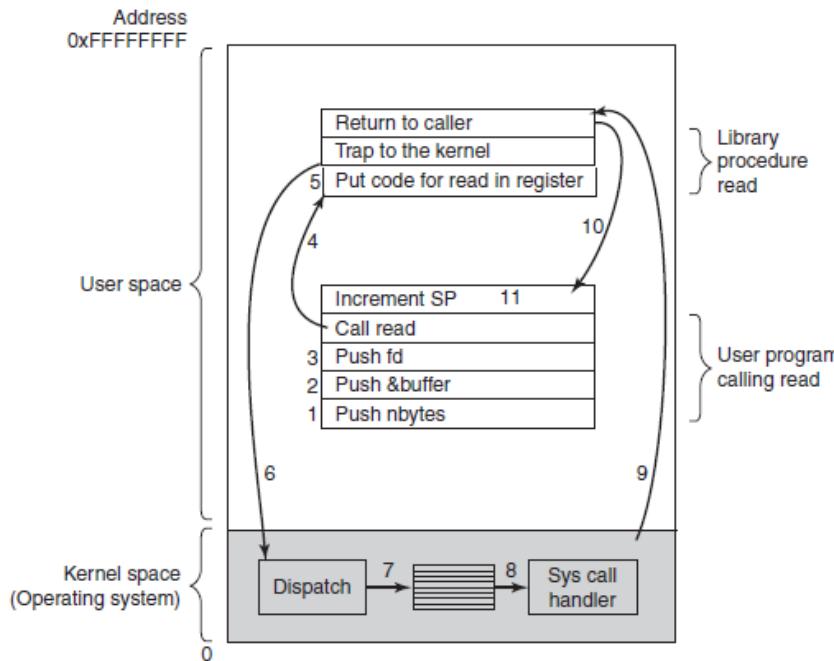
Figure 1-17. The 11 steps in making the system call `read(fd, buffer, nbytes)`.

System Calls (1)



- 1-3. Push parameters on the stack
4. Call system function.
 - Same as a procedure call
5. Place system call code in register.
6. TRAP instruction to switch to kernel mode.
7. Dispatch to signal call handler.

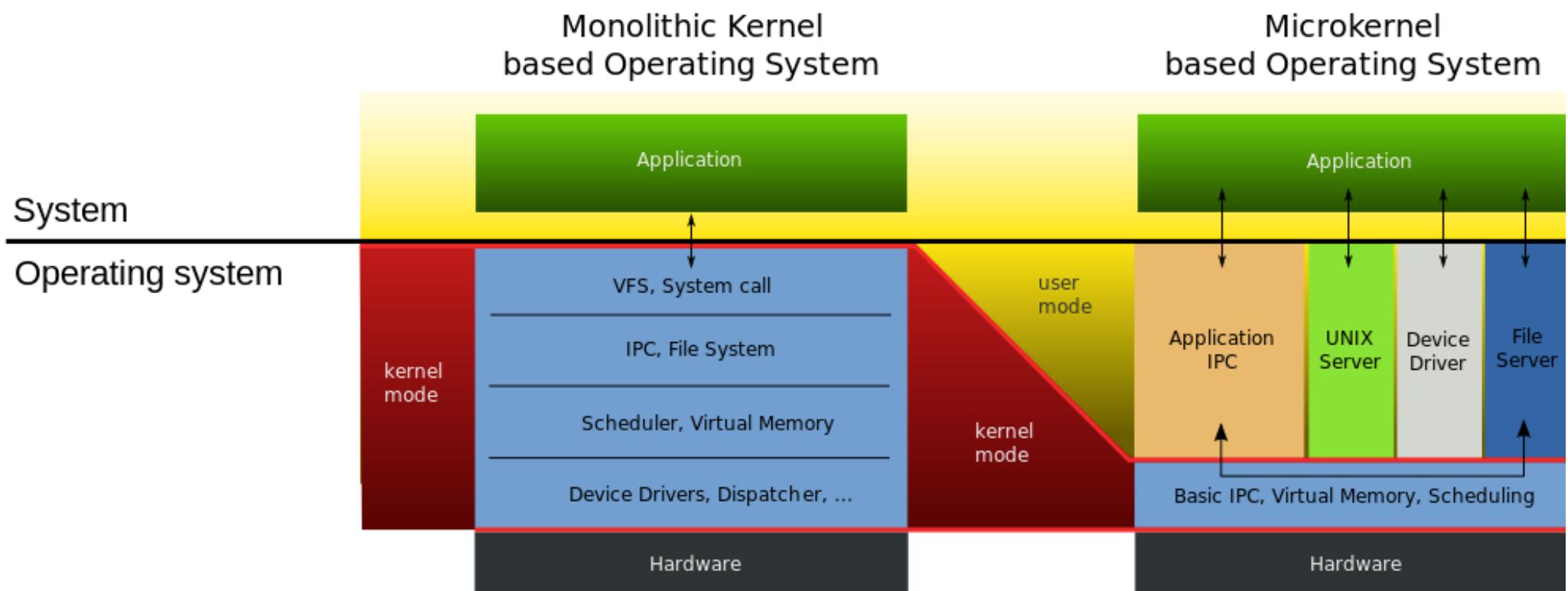
System Calls (1)



8. Signal call handler runs
9. Return from TRAP to user space of system call
10. Return to user program
11. Increment stack pointer to clear stack, as with all procedure calls returned.

https://github.com/CSE3320/kernel-code/blob/master/linux-3.16.36/arch/x86/syscalls/syscall_64.tbl

Micro v. Monolithic



"OS-structure2" by Golftheimer - <http://en.wikipedia.org/wiki/Image:OS-structure.svg>. Licensed under Public domain via Wikimedia Commons - <http://commons.wikimedia.org/wiki/File:OS-structure2.svg#mediaviewer/File:OS-structure2.svg>

Race Conditions

- » Processes working together may share common storage.
- » A race condition is where two or more processes are reading or writing shared data and the final result depends on who runs precisely when.
- » Very common as multiprogramming with more and more cores

Critical Regions

- » Key is to prevent more than one process from reading and writing from a shared area at the same time
- » Mutual exclusion
- » pthread mutexes

Priority Inversion and Starvation

- Indefinite blocking or starvation
 - process is not deadlocked
 - but is never removed from the semaphore queue
- Priority inversion
 - lower-priority process holds a lock needed by higher-priority process !
- Assume three processes L, M, and H
 - Priorities in the order $L < M < H$
 - L holds shared resource R, needed by H
 - M preempts L, H needs to wait for both L and M !!

Priority Inversion and Starvation

- Solutions
 - Only support at most two priorities
 - Priority inheritance protocol – lower priority process accessing shared resource inherits higher priority

Scheduling

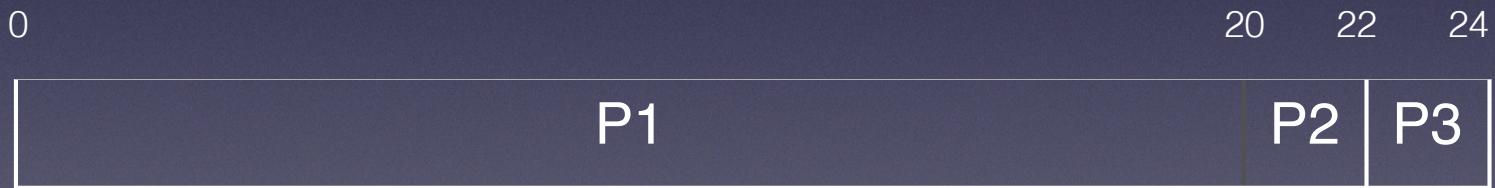
- I/O interrupt
 - Hardware clock provides 50 or 60 hz interrupts
- Scheduling algorithms can be divided into 2 categories based on how they deal with clock interrupts
 - Non-preemptive (cooperative)
 - Preemptive

FCFS Scheduling

- First come, first served algorithm (FCFS).
- Easy to implement
- Well understood by anyone
- The fairest algorithm. No process is favored over another.

FCFS

Process ID	Arrival Time	Runtime
1	0	20
2	2	2
3	2	2



$$\text{Average Waiting Time: } (0 + 18 + 20) / 3 = 12.67$$

FCFS

Process ID	Arrival Time	Runtime
1	0	2
2	2	2
3	2	20

0 2 4

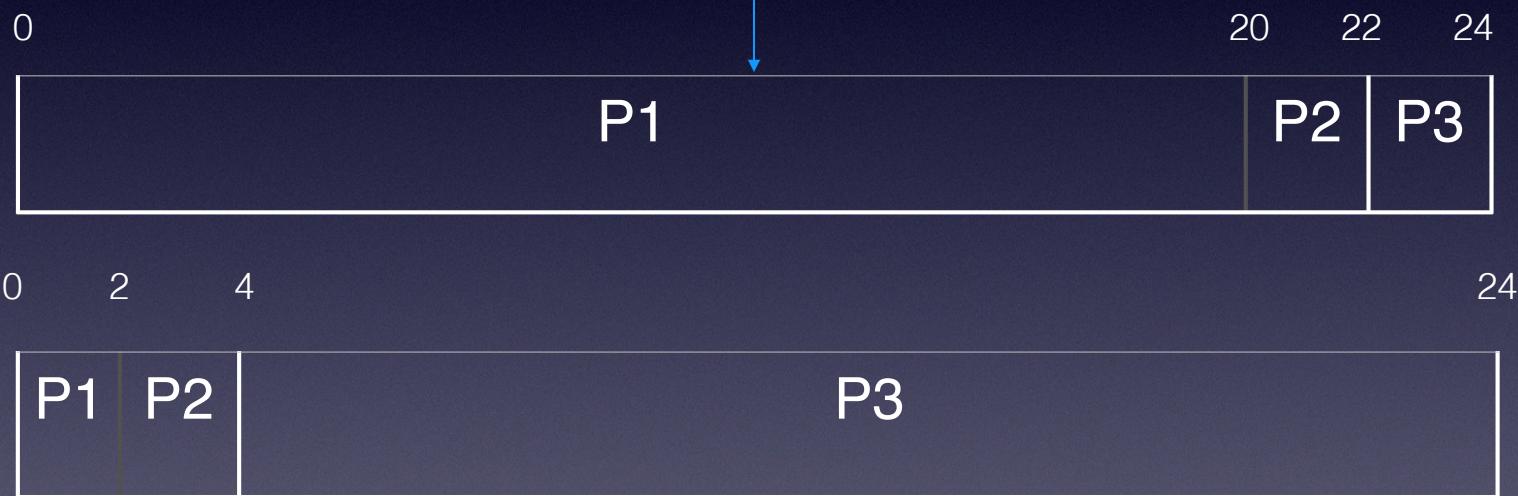
24



$$\text{Average Waiting Time: } (0 + 0 + 2) / 3 = 0.67$$

FCFS

Convoy Effect



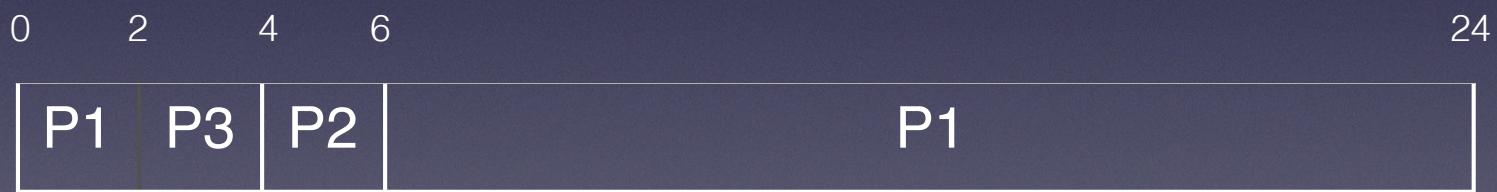
Very Dependent On Job Arrival Time

Preemption

Without Preemption	With Preemption
FCFS	Round Robin
SRTF	Shortest Remaining Time Next
Priority	Priority With Preemption

FCFS w/ Priority (Round Robin)

Process ID	Arrival Time	Runtime	Priority
1	0	20	4
2	2	2	2
3	2	2	1



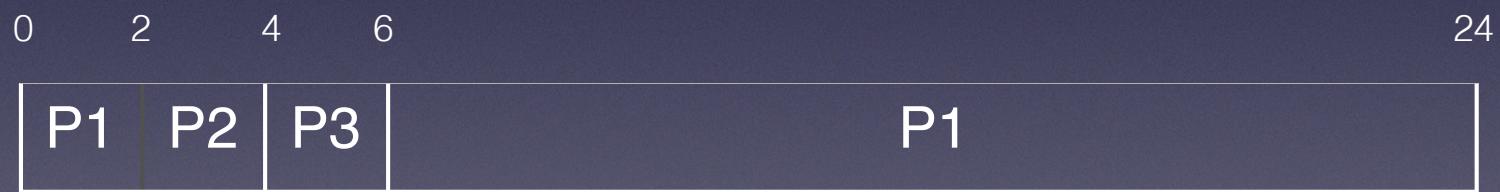
$$\text{Average Waiting Time: } (4 + 2 + 0) / 3 = 2$$

Time Quantum

- Choosing length is a problem
 - Context switching is expensive
 - Still need responsiveness

SJN with Preemption

Process ID	Arrival Time	Runtime
1	0	20
2	2	2
3	2	2



$$\text{Average Waiting Time: } (4 + 0 + 2) / 3 = 2$$

SJN

Process ID	Arrival Time	Runtime
1	0	12
2	2	4
3	3	1
4	4	2

P1 runs first since it's the only process

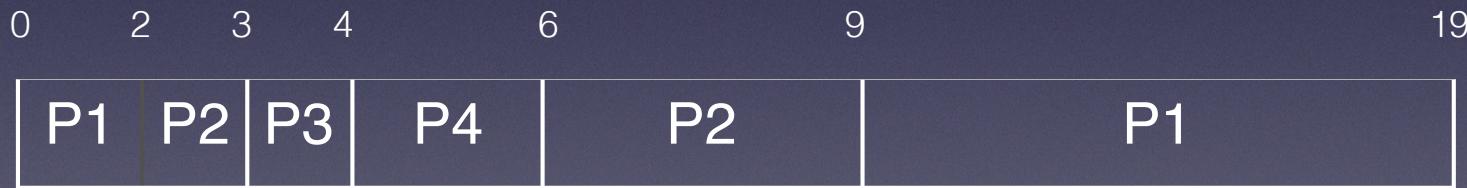
No preemption so P3, P4, and P2 wait. Then sorted by least runtime



$$\text{Average Waiting Time: } (0+13+9+9)/4 = 7.75$$

SJN With Preemption

Process ID	Arrival Time	Runtime
1	0	12
2	2	4
3	3	1
4	4	2



$$\text{Average Waiting Time: } (7+3+0+0)/4 = 2.5$$

SRTF Without and With Preemption

3 Context Switches



5 Context Switches

Priority Scheduling

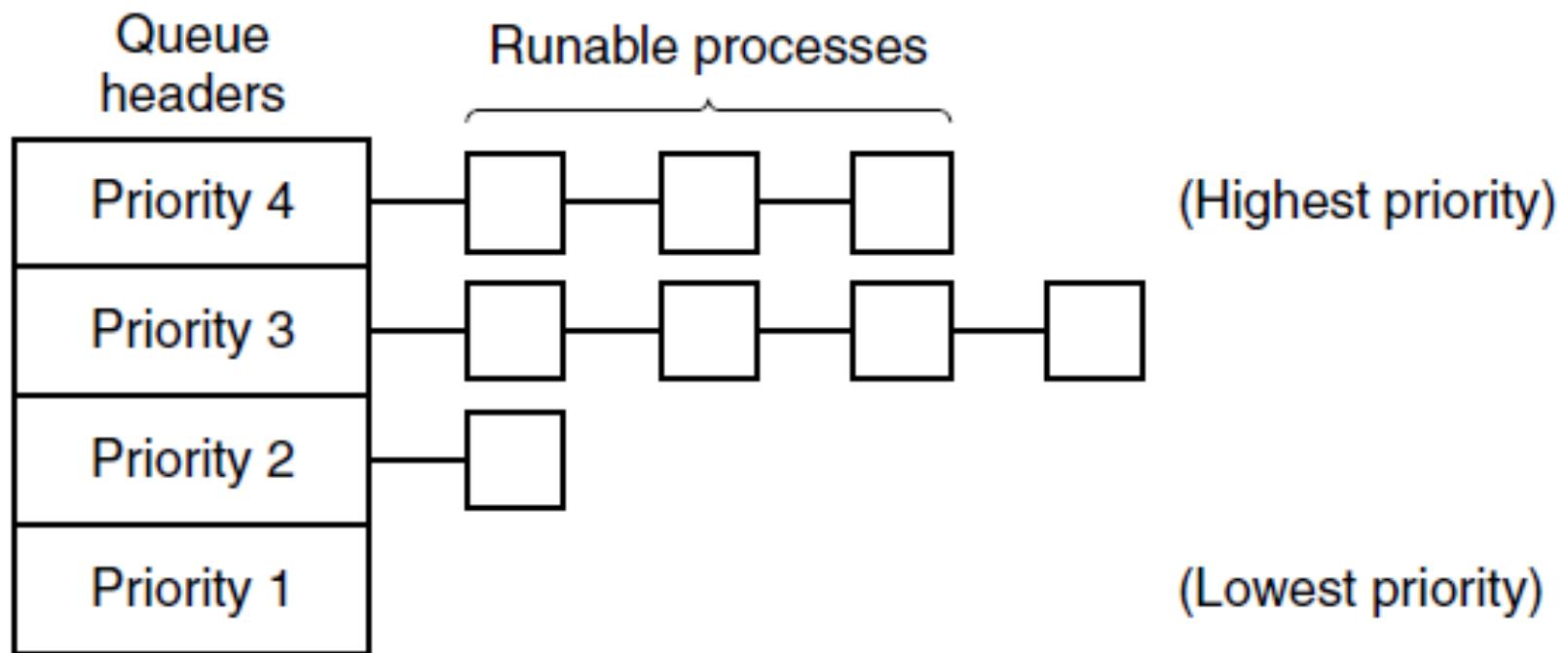


Figure 2-43. A scheduling algorithm with four priority classes.

Memory Management

- » Paraphrase of Parkinson's Law, "Programs expand to fill the memory available to hold them."
- » Average home computer nowadays has 10,000 times more memory than the IBM 7094, the largest computer in the world in the early 1960s

Memory Hierarchy

- » How does the operating system create abstractions from memory, and how does it manage them?
- » Memory hierarchy:
 - » Few megabytes of very fast, very expensive, volatile cache memory
 - » A few gigabytes of medium-speed, medium priced, volatile main memory
 - » A few terabytes of slow, cheap, nonvolatile magnetic or solid state disk storage.

Memory Hierarchy

- » Memory Manager manages it
 - » Tracks which parts of memory are in use
 - » Allocates memory to processes
 - » Deallocates memory when processes are done.

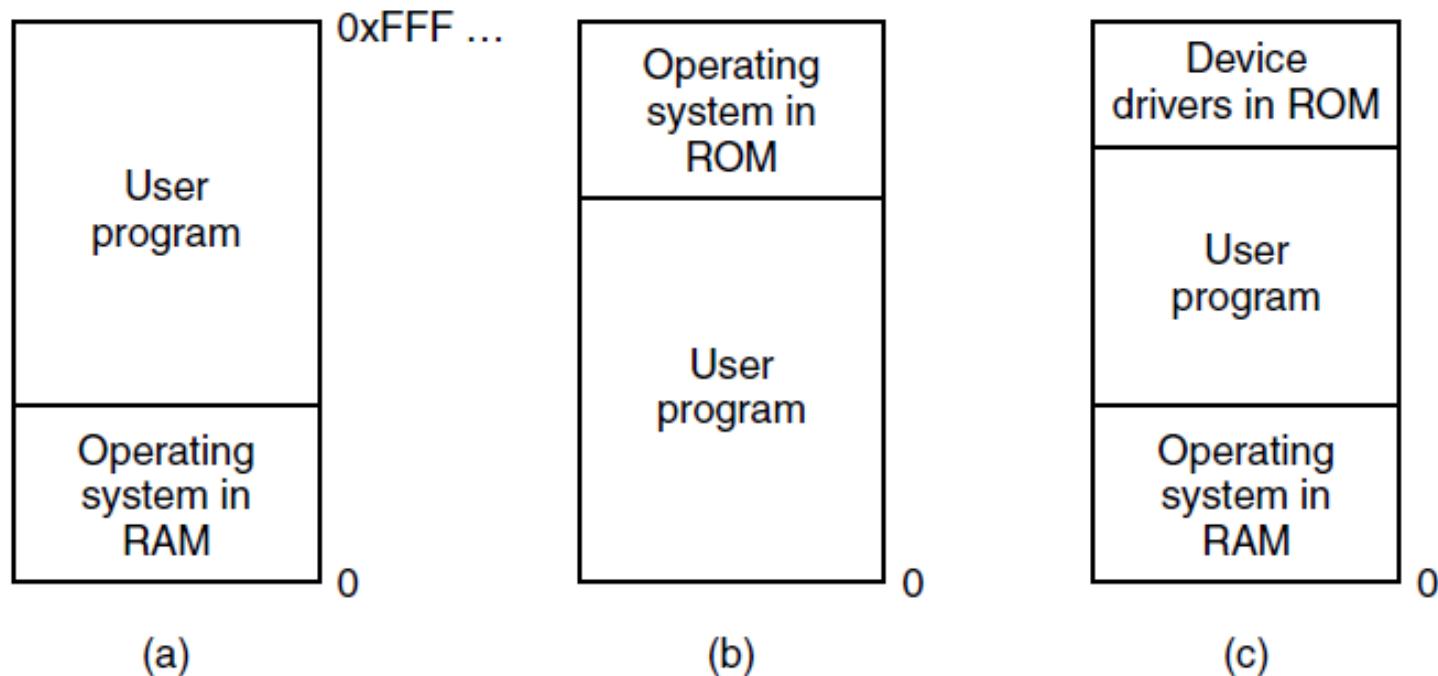
No Memory Abstraction

- » Early mainframes (before 1960), early minicomputers (before 1970), early personal computers (before 1980) had no memory abstraction
- » Every program saw physical memory.
- » Programmers saw a set of addresses from 0 to some maximum.

No Memory Abstraction

- » Two programs could not be resident in memory at the same time.
- » Address conflicts
- » Could run multi-threaded since threads share address space.
- » Limited use. Users want unrelated programs to be running at once.

Memory Layout



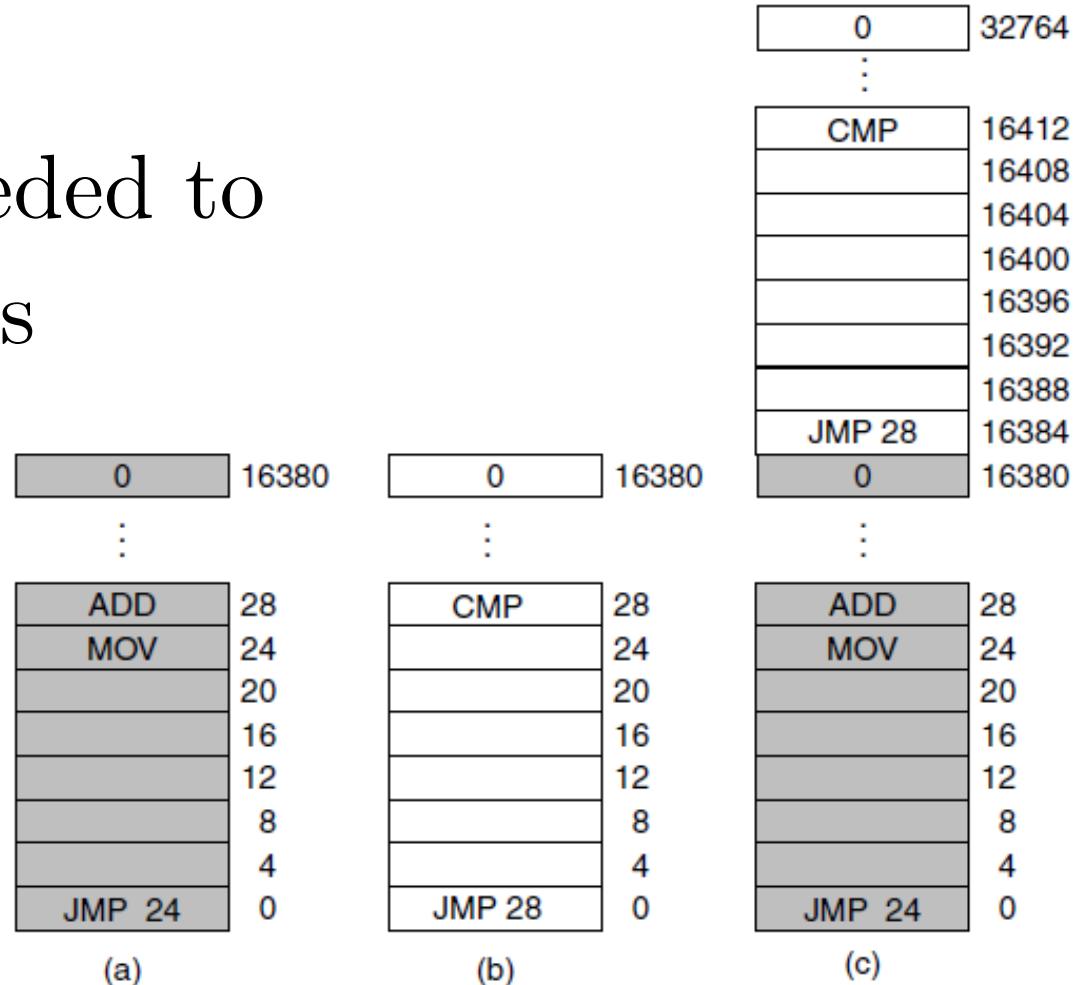
No Memory Abstraction

- » Multiple programs possible
- » Save entire contents of memory to a disk file
- » Bring in new and run it
- » Swapping.
- » Additional hardware will allow concurrency without swapping.

No Memory Abstraction

- » IBM 360 solution: static relocation

- » Extra info needed to specify what is relocatable



Address Spaces

- » Exposing memory to processes have several drawbacks
- » If user programs can address every byte of memory, the OS can be trashed intentionally or by accident.
- » Difficult to have multiple programs running at once

Address Spaces

- » Abstract memory
 - » Process creates a virtual CPU abstraction
 - » Address space is abstract memory for a process to live in
- » Address space: Set of all addresses that a process can see to address memory
- » Each process has an independent address space

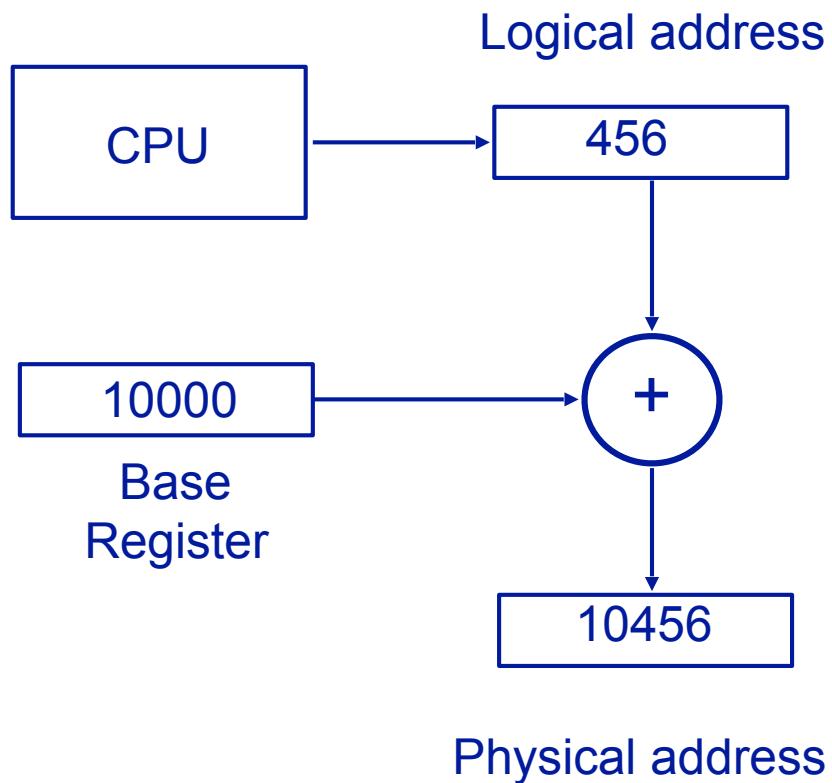
Address Spaces

- » Need to map address 28 in one process and address 28 in another process to a separate physical address.
- » Dynamic relocation
- » Older solution: two special registers
 - » Base register
 - » Limit register

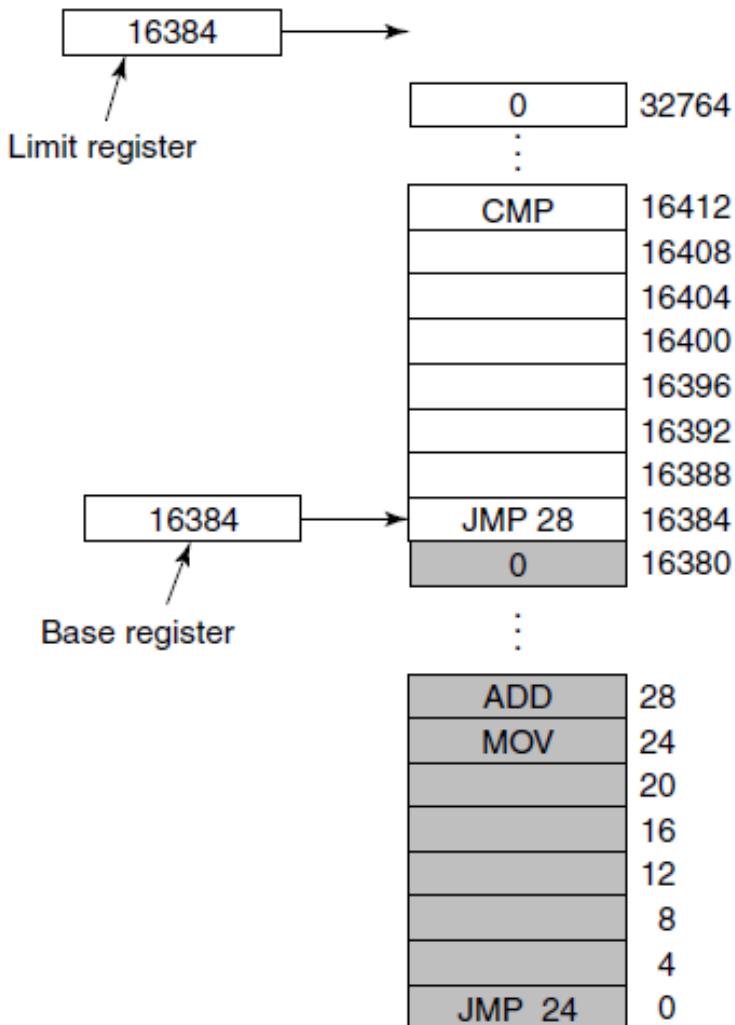
Physical v. Logical Addresses

- » Originally programs were compiled to reference addresses that corresponded one to one with physical memory addresses.
- » Unknowingly using concept of logical and physical memory
 - Logical addresses - Set of addresses the CPU generates as the program is executed.
 - Physical addresses - Set of addresses used to reference physical memory.

Dynamic Relocation



Dynamic Relocation

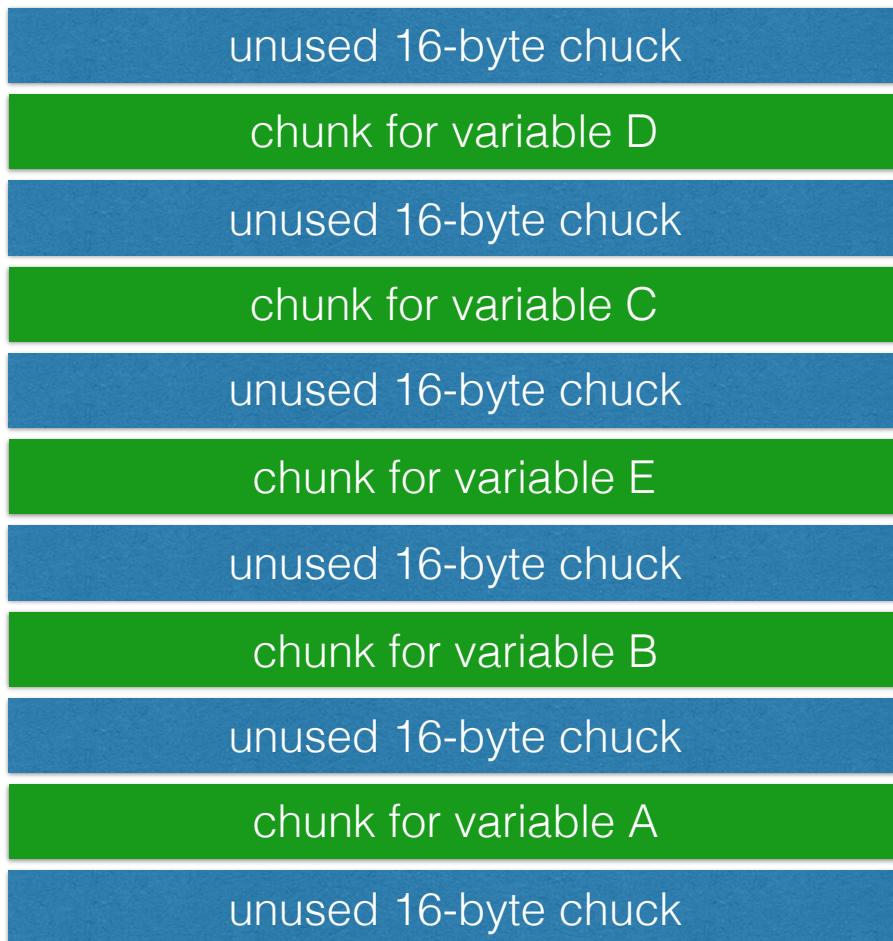


Swapping

- » If physical memory is large enough to hold all the processes, the previous schemes will do
- » In practice, total amount of RAM is more than can fit in memory.
- » Two approaches:
 - » Swapping: Bring in each process in its entirety, run it for a while, then put it back onto disk
 - » Virtual Memory: Run processes when they are partially in memory

Heap Fragmentation

Heap Space



96 bytes of free
memory but we
can't allocate any
chuck larger than
16 bytes

External Fragmentation

Compaction

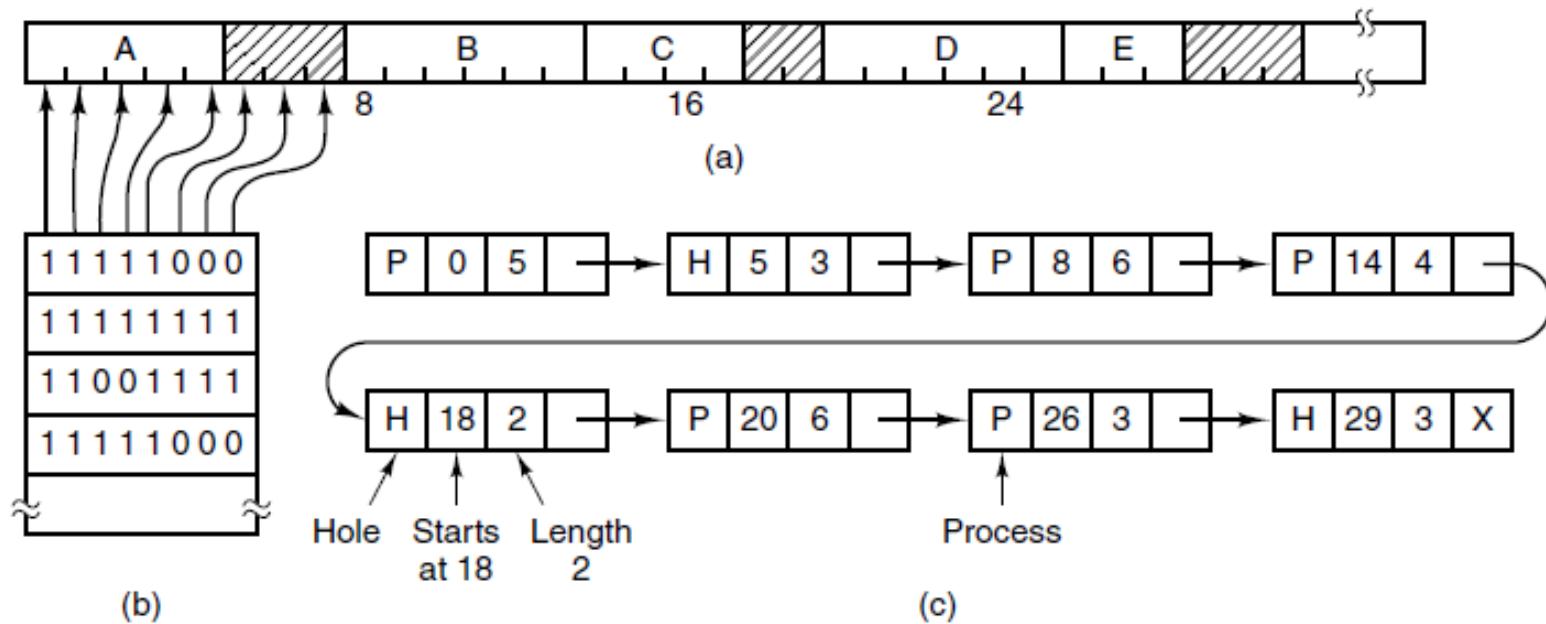
- » The operating system can reorganize the fragmented space so that the free space is contiguous.
- » Expensive

On a 16-GB machine that can copy 8 bytes in 9 sec, it would take 16 sec to compact all of the memory

Managing Free Memory

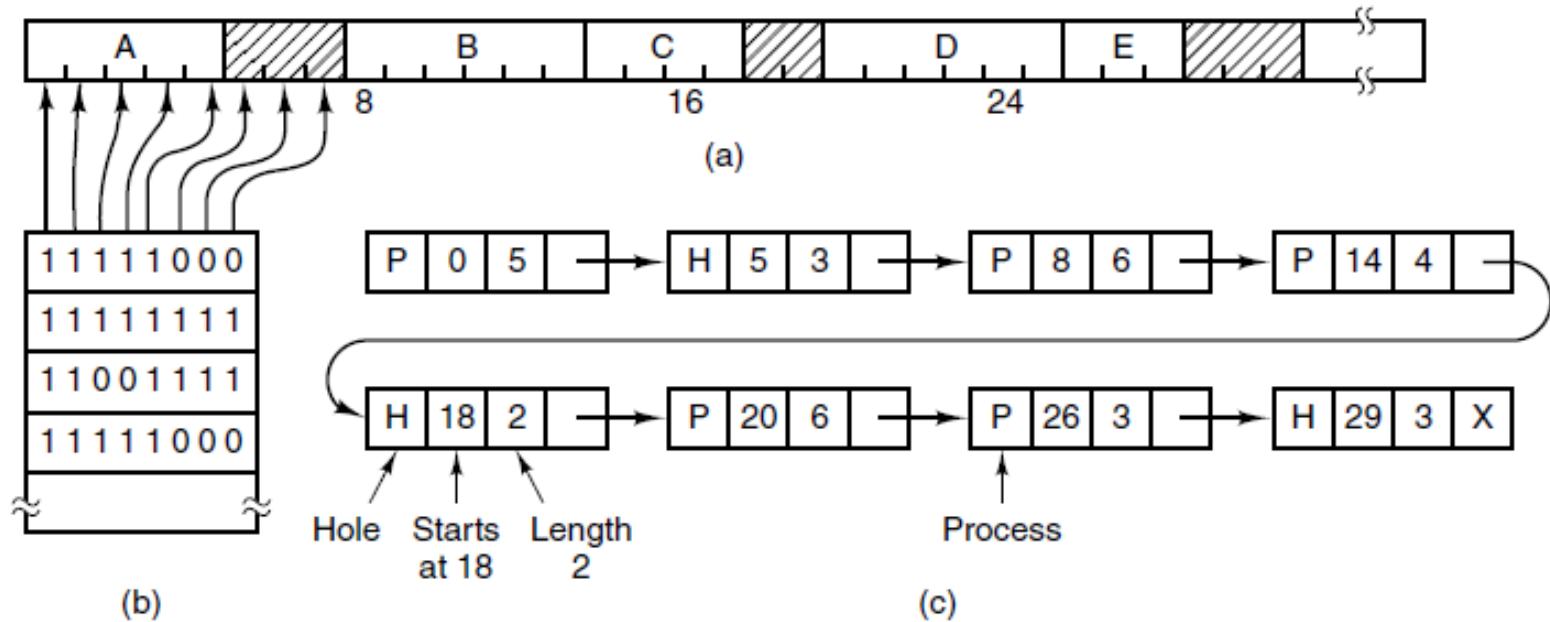
- » Two ways to track memory usage
 - » Bit maps
 - » Free lists
- » Chapter 10: Linux memory allocators such as buddy and slab.

Bitmap



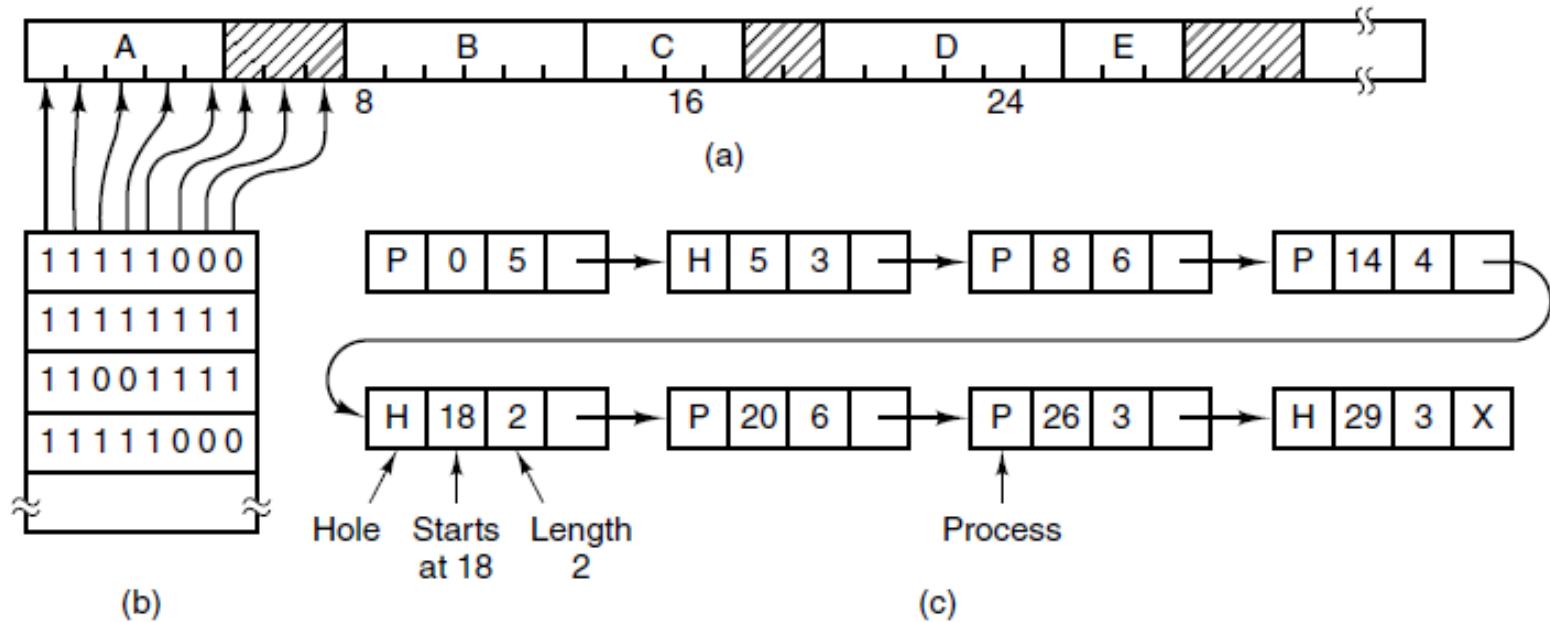
- » With bitmap, memory divided into allocation units as small as a few words to a couple kilobytes.
- » Each allocation unit corresponds to a bit

Bitmap



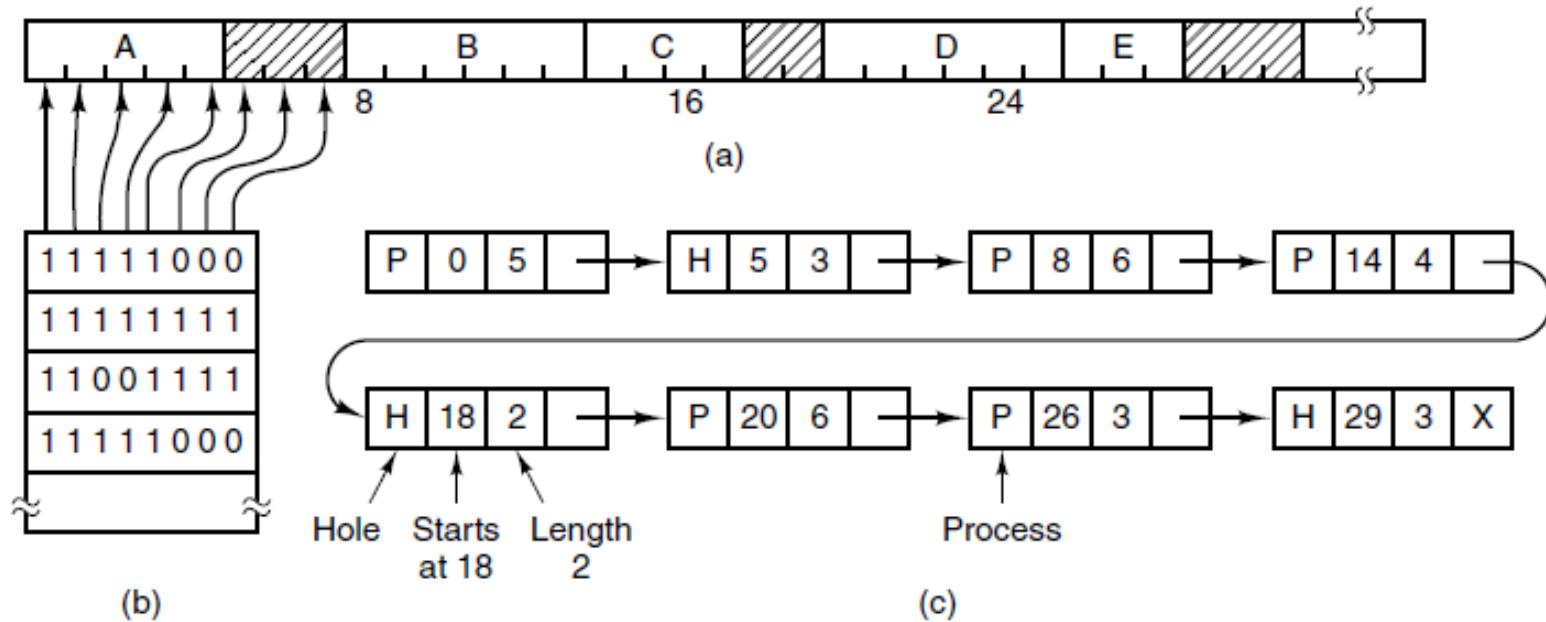
- » Problem with bitmap: When bringing in K-units, the memory manager must search the bitmap for a run of k consecutive 0's

Bitmap



- » Problem with bitmap: When bringing in K-units, the memory manager must search the bitmap for a run of k consecutive 0's

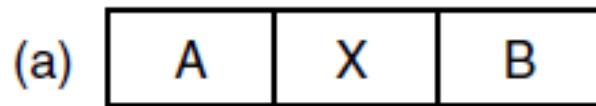
Linked List



- » Linked list: list of allocated and free memory segments.

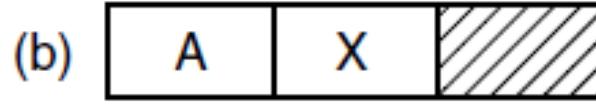
Linked List

Before X terminates

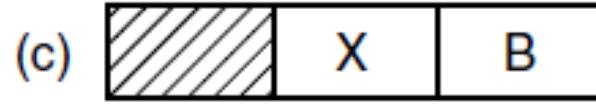


becomes

After X terminates



becomes



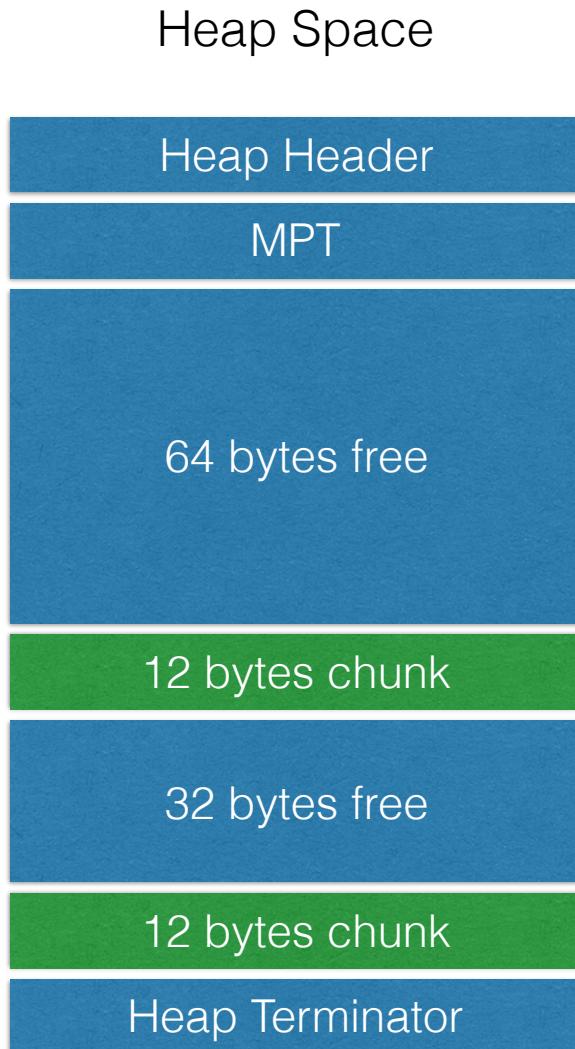
becomes



becomes



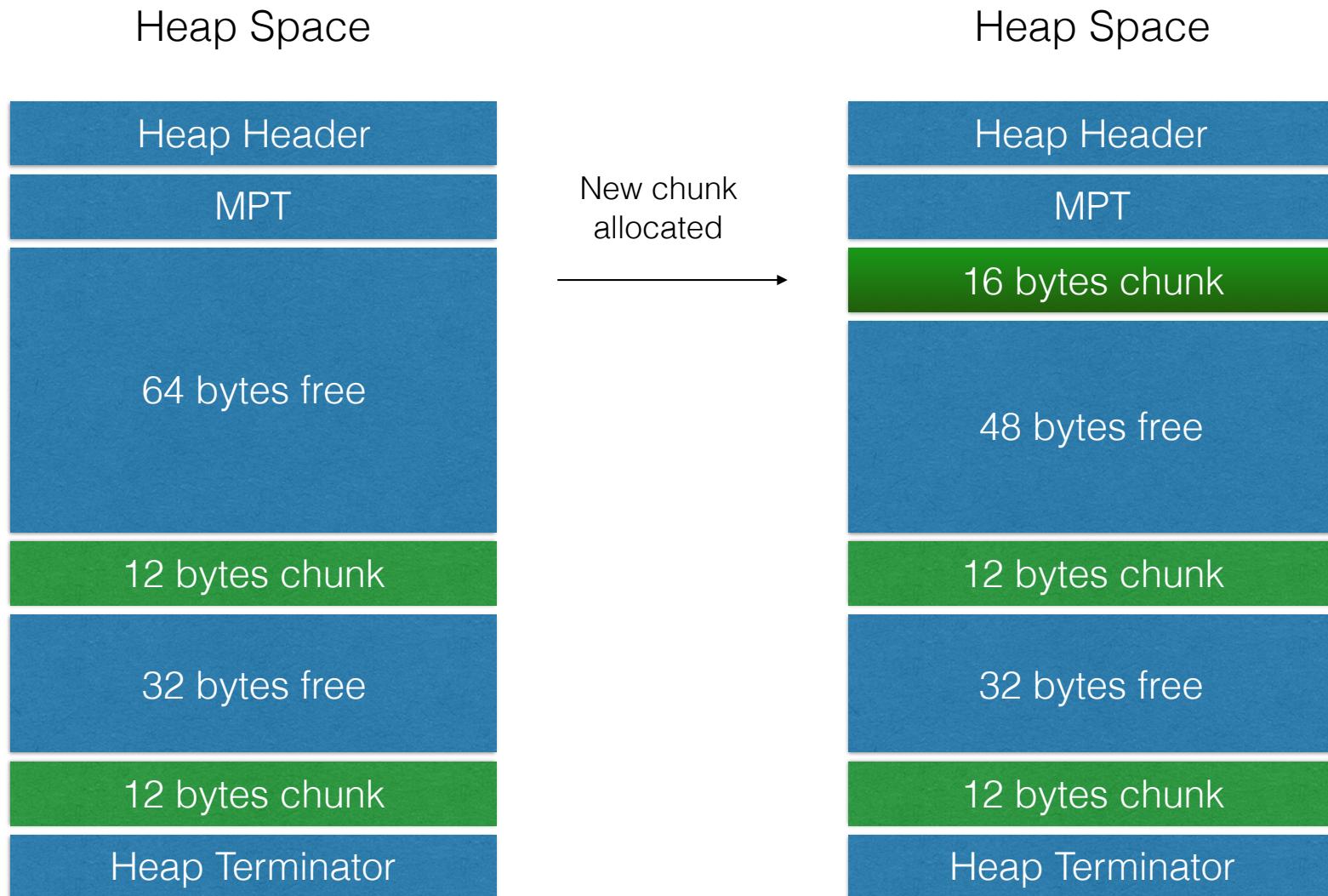
Allocating Memory



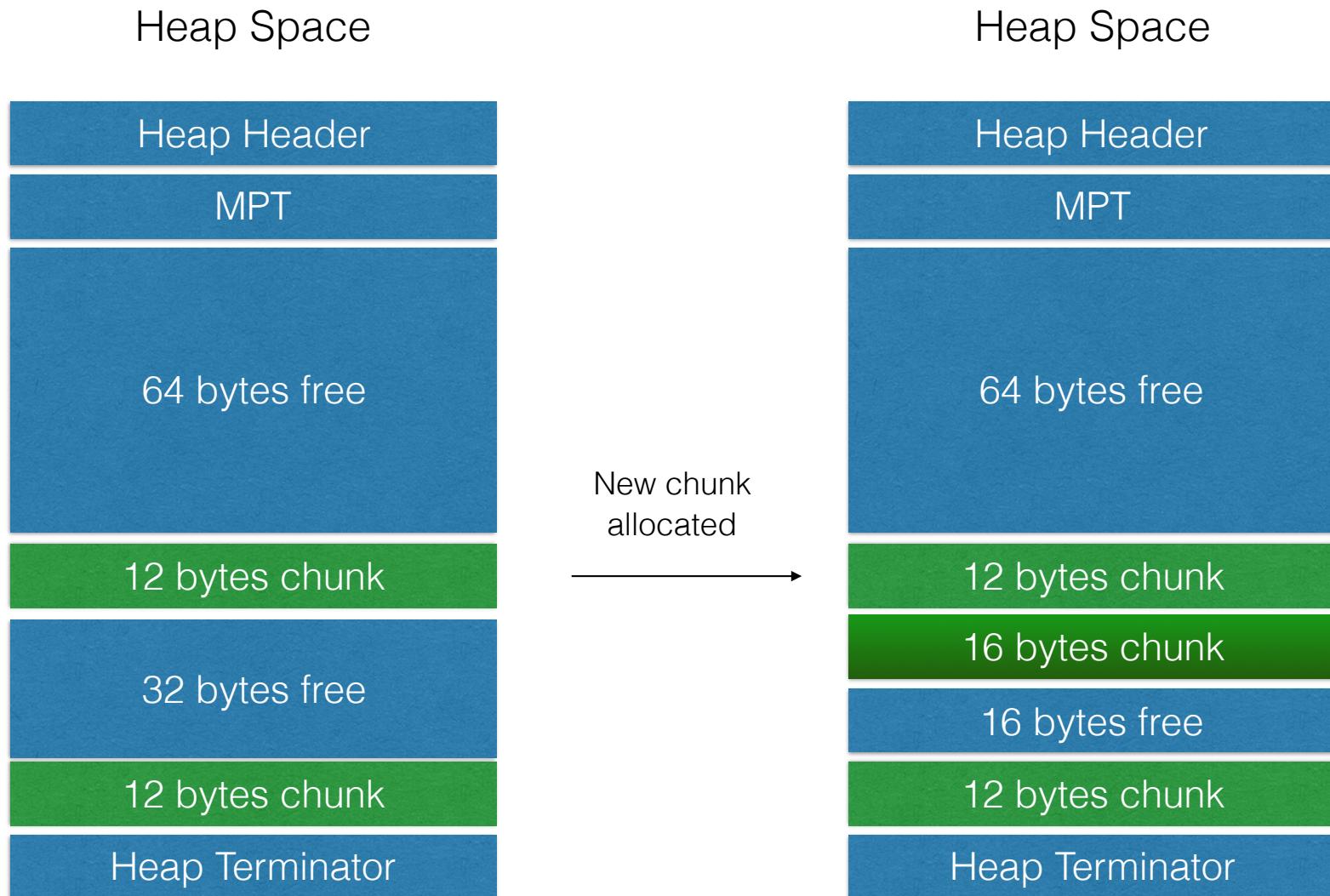
Example: We want to allocate a new 16 byte block.

Which free block does the OS choose?

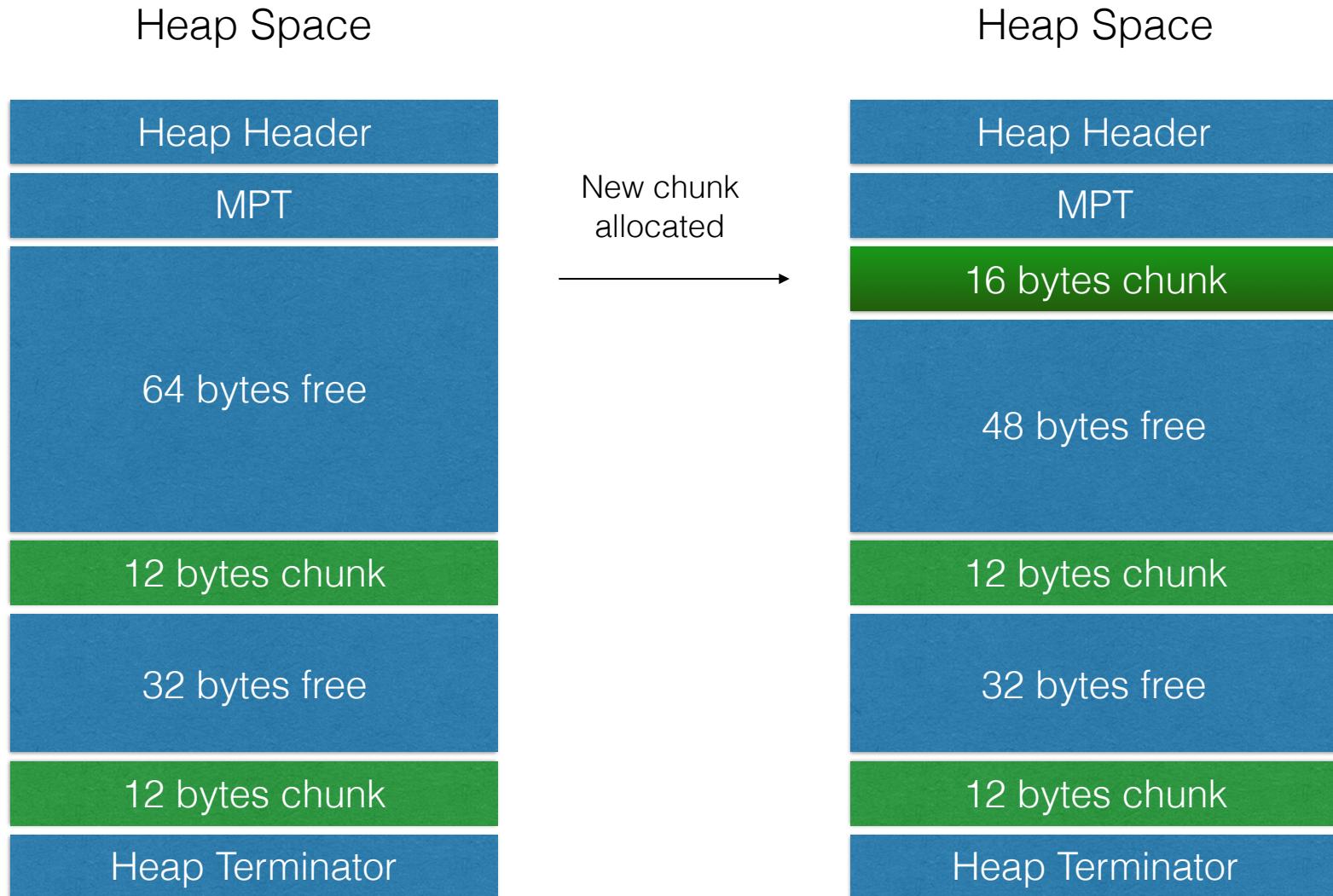
First Fit



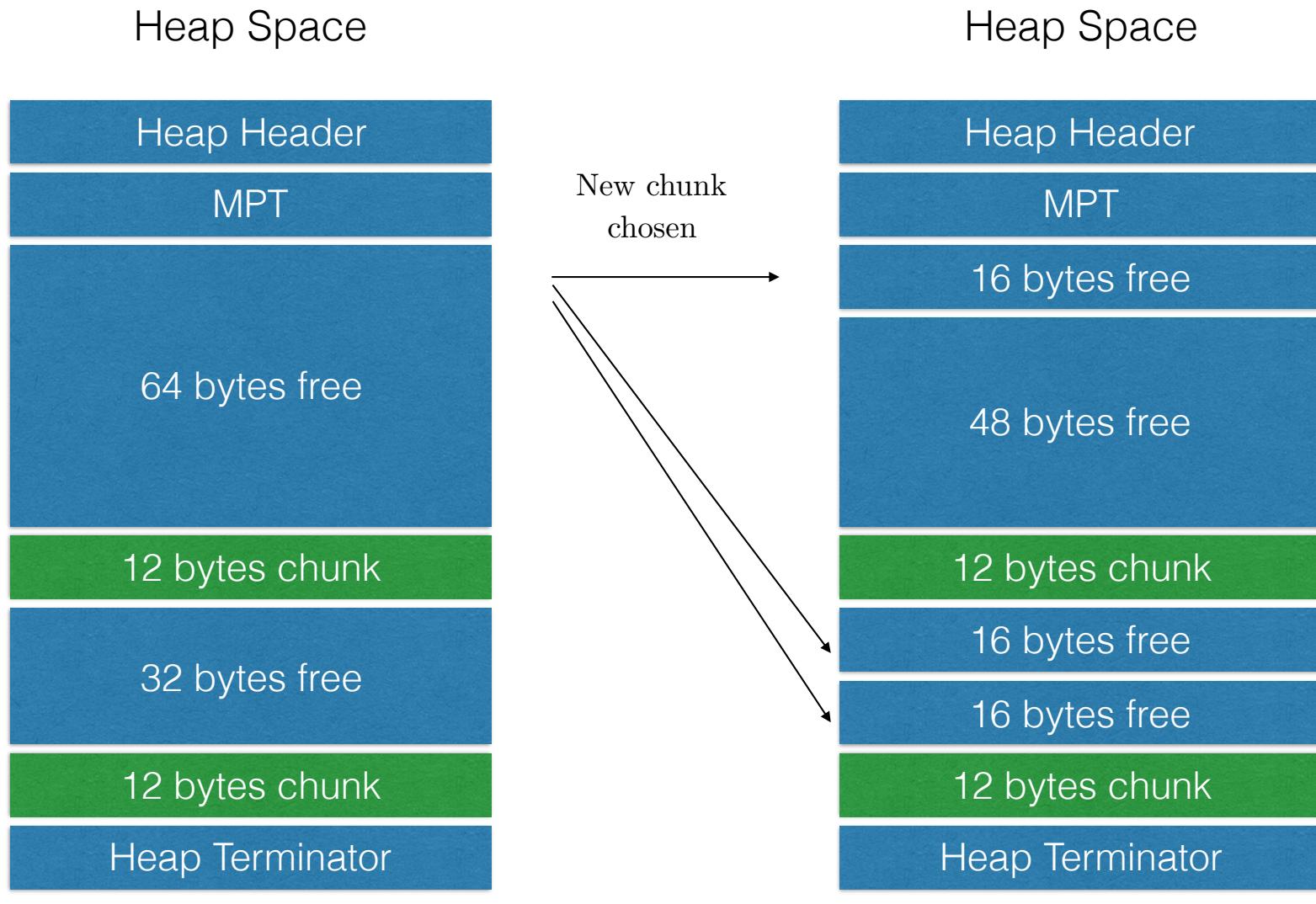
Best Fit



Worst Fit



Quick Fit



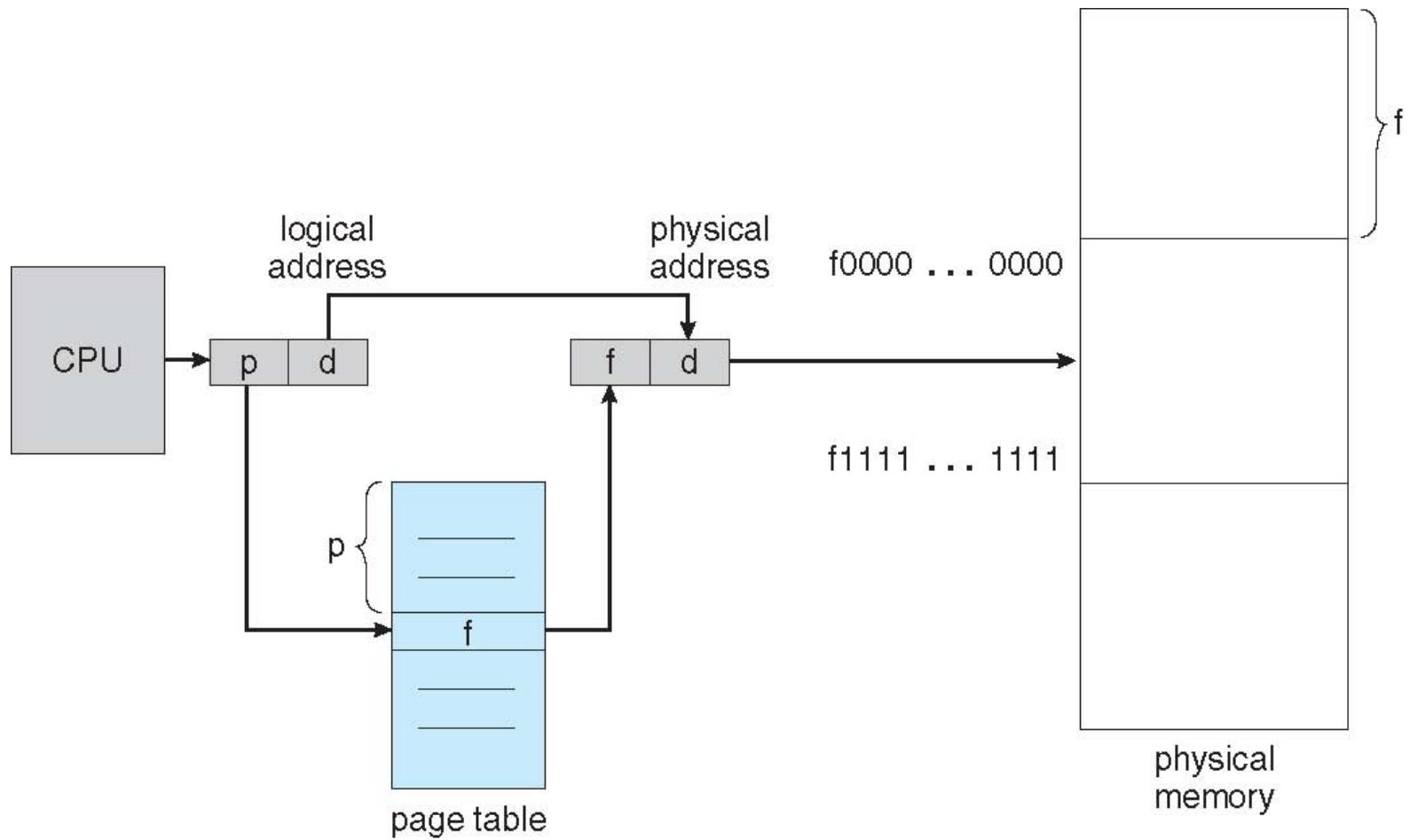
We need help, but why?

- » Multiprocessing causes external fragmentation
- » Compaction wastes resources
- » Solution – break RAM into fixed parts
- » Do not need to be contiguous
- » Map from logical to physical spaces
- » Need hardware help

Paging

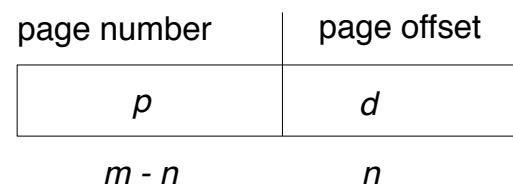
- » Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- » Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8,192 bytes)
 - Commonly 4 KB
- » Divide logical memory into blocks **of same size** called pages
- » Keep track of all free frames
- » To run a program of size n pages, need to find n free frames and load program
- » Set up a page table to translate logical to physical addresses
 - Holds the map to the physical address
- » Internal fragmentation

Paging Hardware



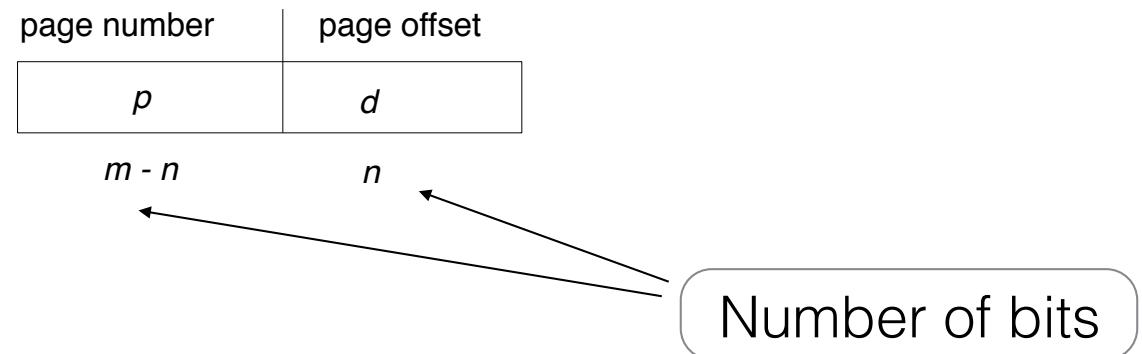
Address Translation Scheme

- » Address generated by CPU is divided into:
 - **Page number (p)** – used as an index into a *page table* which contains base address of each page in physical memory
 - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit
 - For given logical address space 2^m and page size 2^n



Address Translation Scheme

- For given logical address space 2^m and page size 2^n



Address Translation Scheme

- For a system with 4 GB of addressable logical address space and a page size of 4 KB, the logical address is:

page number	page offset
p	d
$m - n$	n

$$\text{Page size: } 4096 = 2^{12}$$

$$\text{Address space: } 4294967296 = 2^{32}$$

page number	page offset
20	12

Address Translation Scheme

- Number of pages:

$$4294967296 / 4096 = 1048576$$

» And to verify our numbers:

$$1048576 = 2^{20}$$

page number	page offset
20	12

Address Translation Scheme

For a system with 12 bit addressing:

1. What is the maximum size of addressable real memory?

If the addresses are split into a page (6 bits) and an offset (displacement, 6 bits)

1. How large is a memory frame?
2. How many entries (maximum) is the page table?

Address Translation Scheme

For a system with 12 bit addressing:

1. What is the maximum size of addressable real memory?

$$2^{12} = 4096 \text{ bytes}$$

Address Translation Scheme

If the addresses are split into a page (6 bits) and an offset (6 bits)

1. How large is a memory frame?

$$2^6 = 64 \text{ bytes}$$

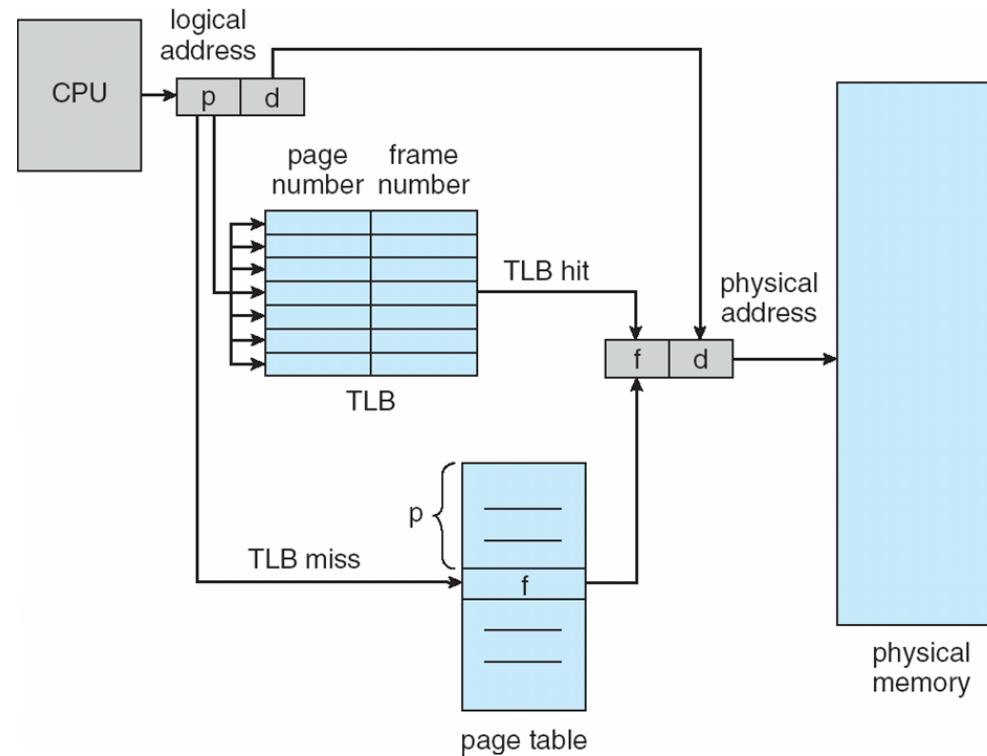
2. How many entries (maximum) is the page table?

$$2^6 = 64 \text{ pages}$$

Implementation of Page Tables

- » In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
 - Half speed
- » The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs)

Translation Lookaside Buffer



- All entries search in parallel
- Complex circuit so it's small
- Rarely over 1000 entries

Effective Access Time

- » Associative Lookup = ε time unit
- » Assume memory cycle time = M
- » Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- » Hit ratio = α

Effective Access Time (EAT)

$$EAT = (M + \varepsilon) \alpha + (2M)(1 - \alpha)$$



2M because a miss requires us to get the frame number out of the page table (M cost) and get the normal memory reference (M cost)

Effective Access Time Example

- »Associative Lookup = ϵ = 5 nanoseconds
- »Assume memory cycle time is 100 nanoseconds
- »Hit ratio = a = 80%

Effective Access Time (EAT)

$$\begin{aligned} \text{EAT} &= (100 + 5) .8 + (200)(1 - .8) \\ &= 124 \text{ nanoseconds} \end{aligned}$$

Effective Access Time

- » First equation assumed TLB and Page Table lookups happened in parallel.
- » If not:

Effective Access Time (EAT)

$$EAT = (M + \varepsilon) \alpha + (2M + \varepsilon)(1 - \alpha)$$

Context Switches

- » Most systems TLB doesn't concern itself with which process is running.
 - Context switch means TLB must be flushed.
 - First few memory accesses of the new process will not get any TLB hits so the process will run at half speed for memory references.
 - Threads vs. processes
- » Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process

Demand Paging

- » Our previous methods had one requirement:
 - The instructions being executed must be in physical memory.
 - The first approach to meeting this requirement is to place the entire logical address space in physical memory.
 - Dynamic loading can help to ease this restriction
 - Requires special precautions and extra work by the programmer.

Demand Paging

- » Up until now we have assumed all programs fit fully into memory before they execute.
- » Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- » One major advantage of this scheme is that programs can be larger than physical memory.

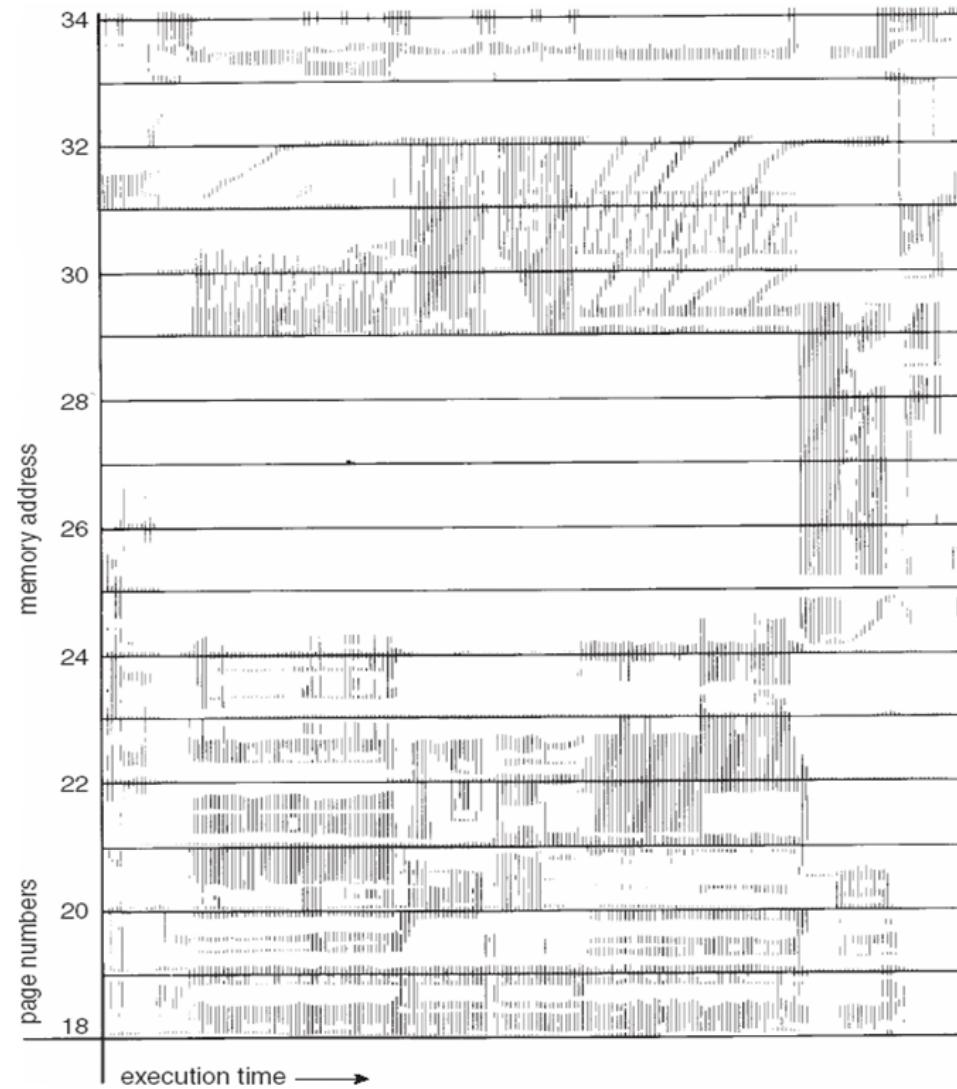
Demand Paging

- » The requirement that instructions must be in physical memory to be executed seems both necessary and reasonable;
 - Also unfortunate, since it limits the size of a program to the size of physical memory.
- » In fact, an examination of real programs shows us that, in many cases, the entire program is not needed. For instance, consider the following:
 - Programs often have code to handle unusual error conditions. Since these errors seldom, if ever, occur in practice, this code is almost never executed.
 - Arrays, lists, and tables are often allocated more memory than they actually need. An array may be declared 100 by 100 elements, even though it is seldom larger than 10 by 10 elements. An assembler symbol table may have room for 3,000 symbols, although the average program has less than 200 symbols.
 - Certain options and features of a program may be used rarely

Demand Paging

- » Even in those cases where the entire program is needed, it may not all be needed at the same time.
 - Locality of Reference - Only a few pages of memory are accessed by the process in any given time slot.

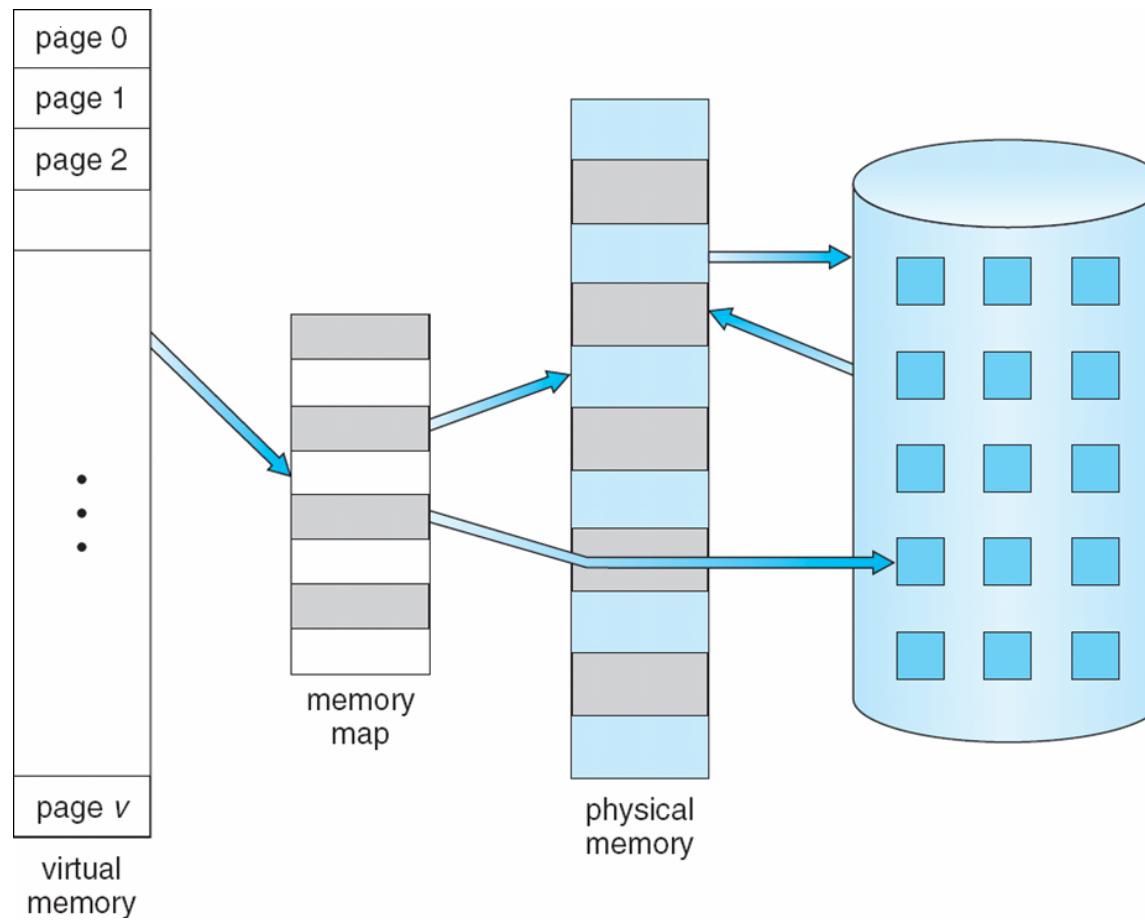
Locality In A Memory-Reference Pattern



Demand Paging

- » The ability to execute a program that is only partially in memory would confer many benefits:
 - A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space, simplifying the programming task.
 - Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput but with no increase in response time or turnaround time.
 - Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.

Virtual Memory



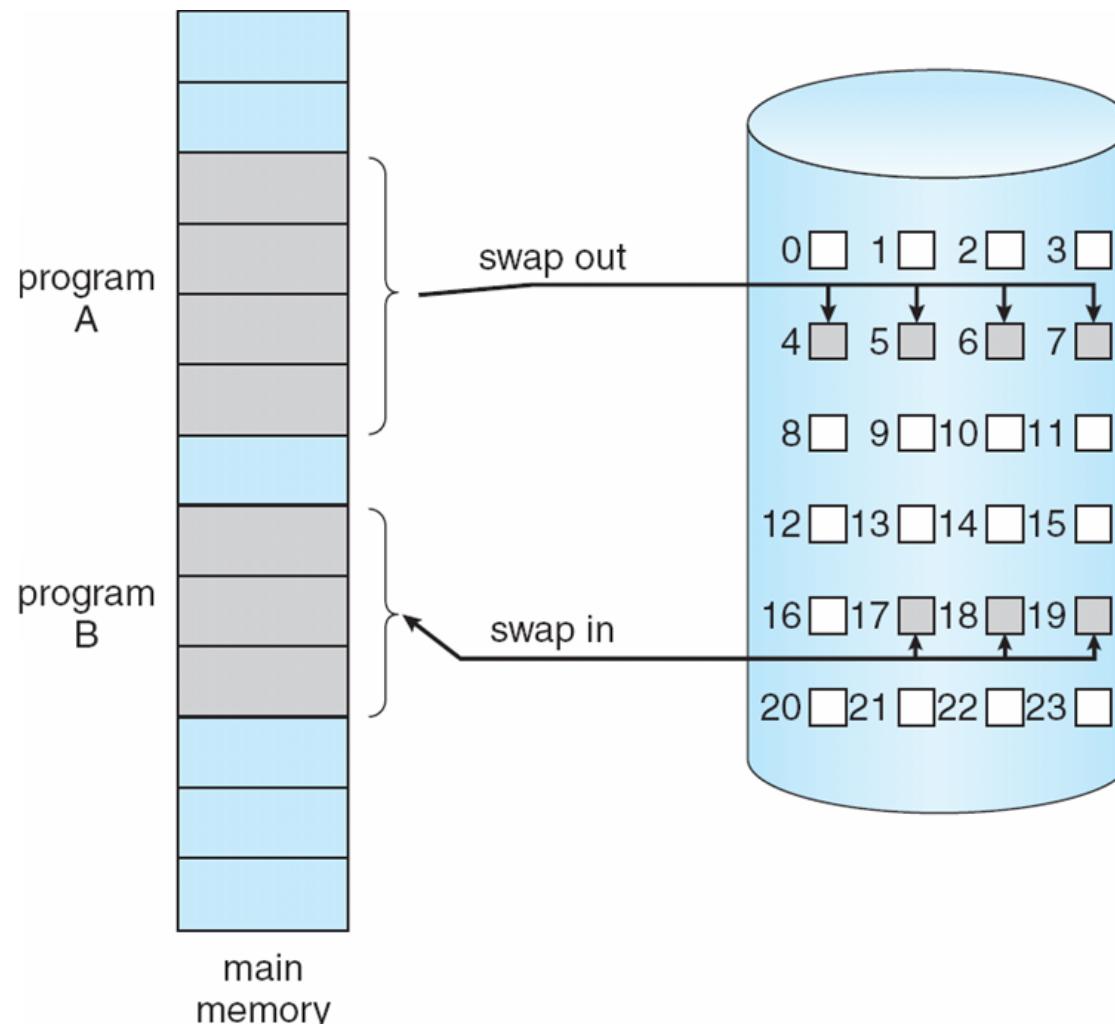
Demand Paging

- » Bring a page into memory only when it is needed
 - Less I/O needed
 - Less memory needed
 - Faster response
 - More users

Demand Paging

- » Page is needed \Rightarrow reference to it
- » invalid reference \Rightarrow abort
- » not-in-memory \Rightarrow bring to memory
- » *Lazy loading* – never swaps a page into memory unless page will be needed
- » Swapper that deals with pages is a pager

Demand Paging



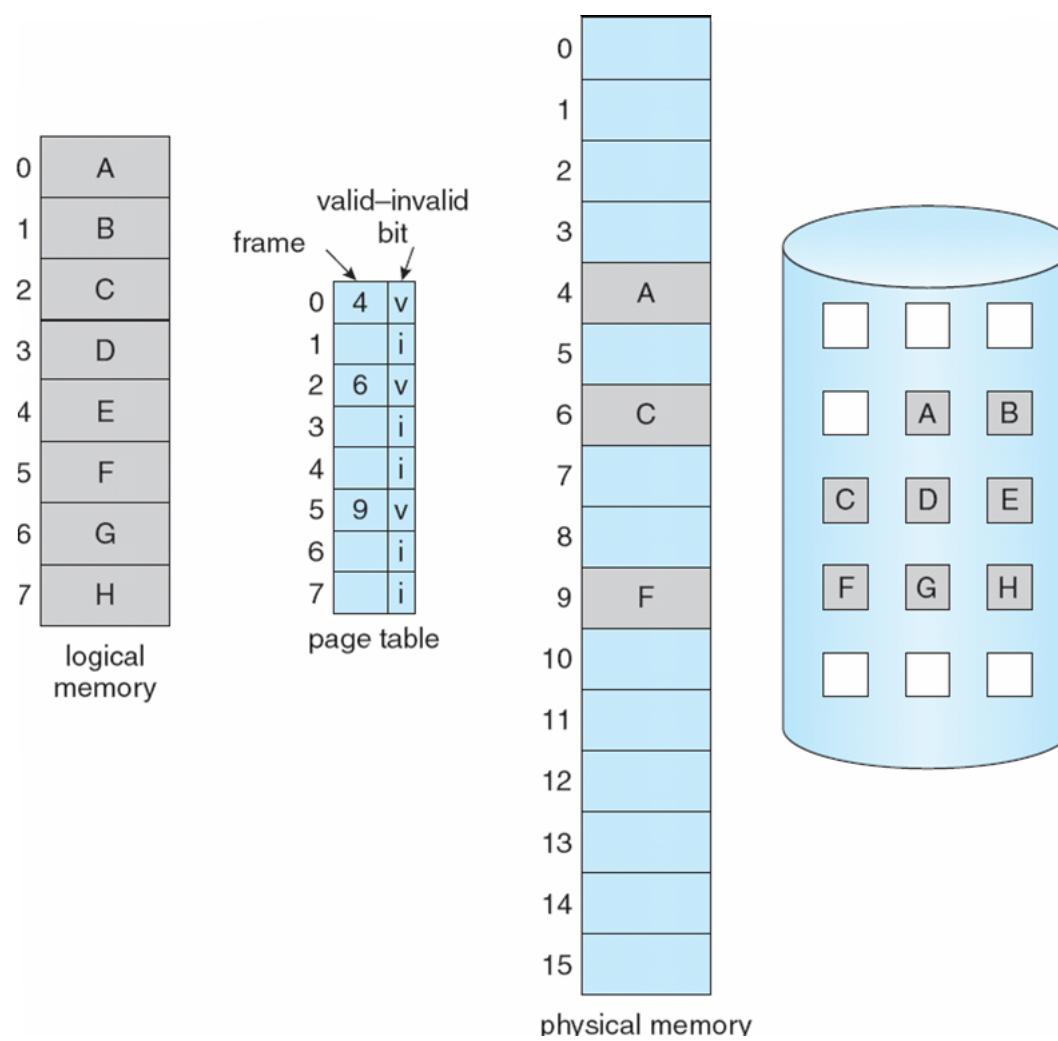
Demand Paging

- » With each page table entry a valid–invalid bit is associated ($v \Rightarrow$ in-memory, $i \Rightarrow$ not-in-memory)
 - » Present / Absent in book parlance
- » Initially valid–invalid bit is set to i on all entries
- » Example of a page table snapshot:

Frame #	valid-invalid bit
	v
	v
	v
	v
	i
....	
	i
	i

page table

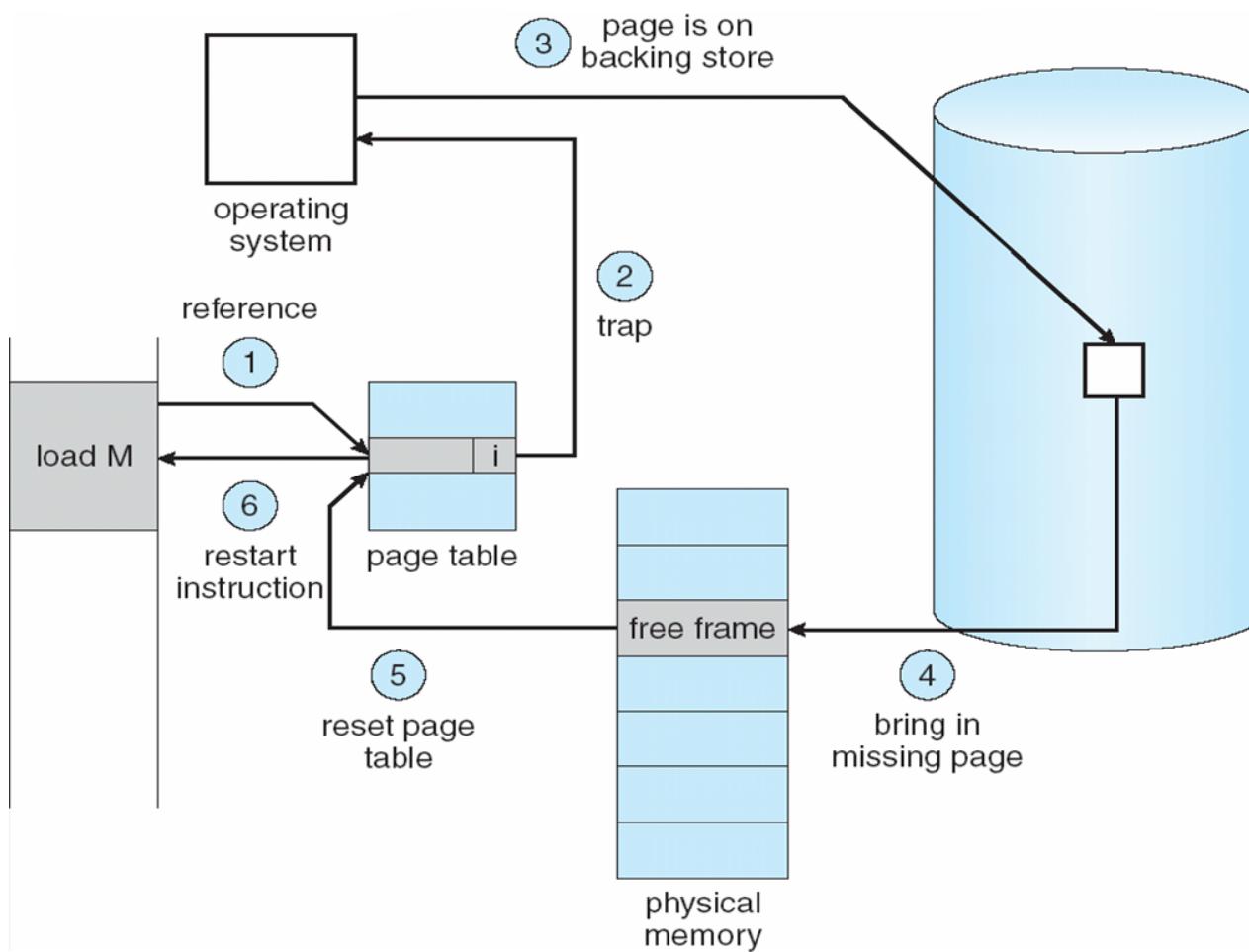
Demand Paging



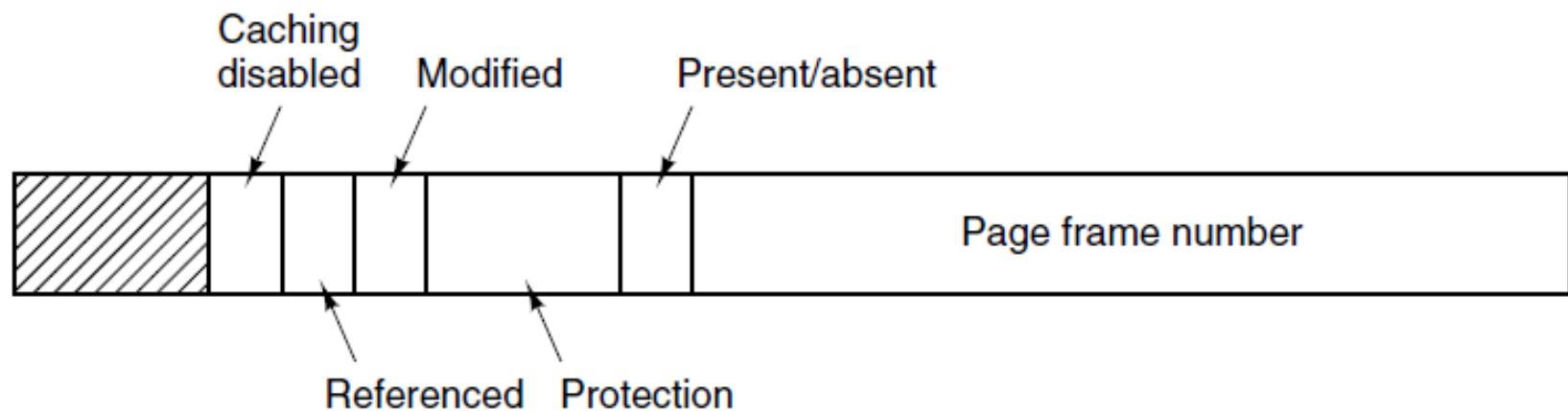
Demand Paging

- » If there is a reference to a page, first reference to that page will trap to operating system:
 - » page fault
- 1. Operating system looks at another table to decide:
 1. Invalid reference \Rightarrow abort
 2. Just not in memory
- 2. Get empty frame
- 3. Swap page into frame
- 4. Reset tables
- 5. Set validation bit = v
- 6. Restart the instruction that caused the page fault

Demand Paging



Page Table Entry



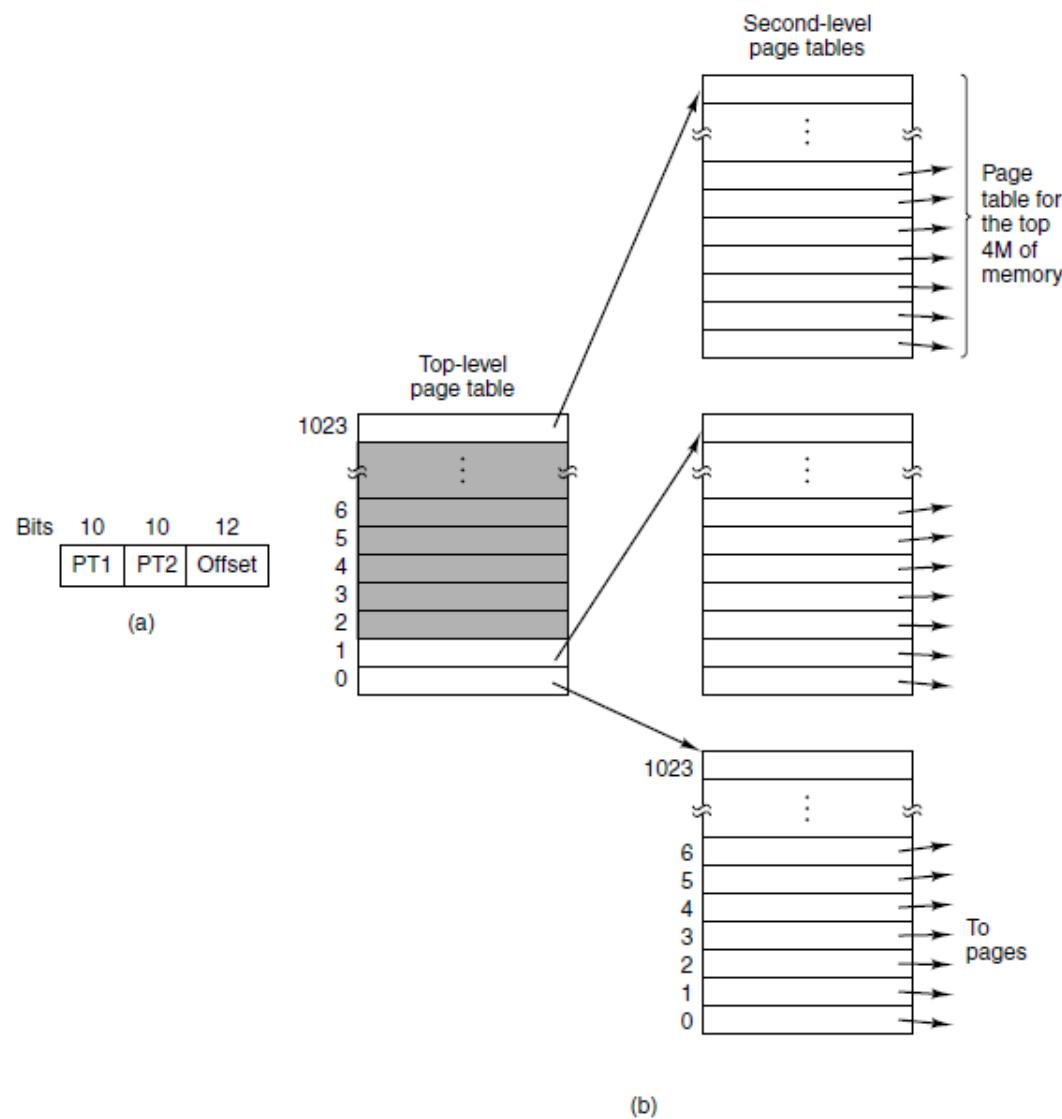
Types of Page Faults

- » Soft miss: 10-20 instructions
- » Hard miss: several milliseconds
- » Minor page fault
- » Major page fault
- » Segmentation fault

Multilevel Page Tables

- » Single page table contains one entry per virtual page per process
 - » 4 MB per process. $2^{32} / 4096 * 4$
- » Multilevel page tables avoid keeping all page tables in memory at a time.

Multilevel Page Tables



Multilevel Page Tables

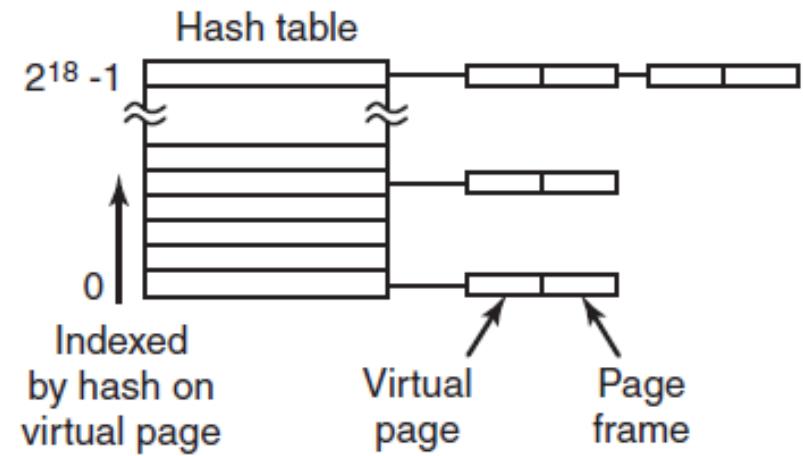
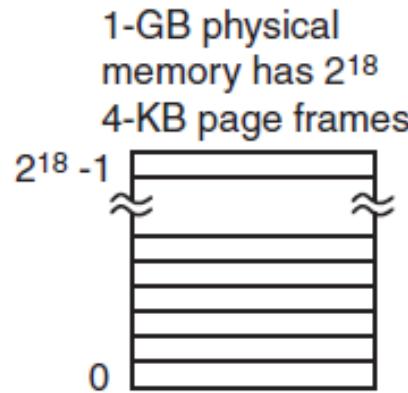
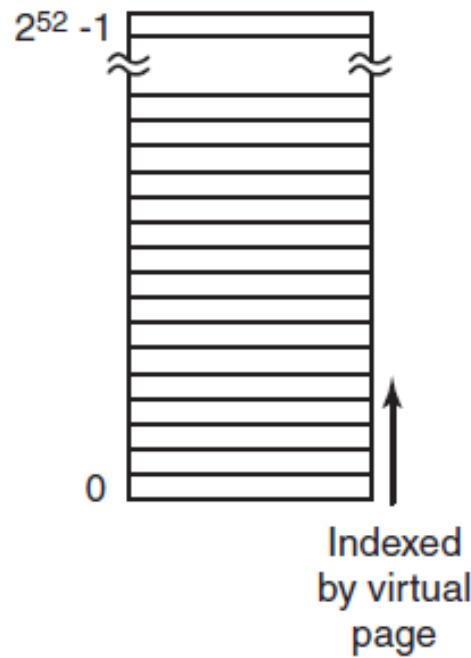
- » 80386 addressed 4 GB of memory using a two-level page table.
 - » Page directory -> page tables -> frames
- » Pentium Pro
 - » Page directory pointer table
 - » 64 bit so it could address over 4GB
 - » Page Map Level 4
 - » $2^{48} = 256$ TB

Inverted Page Tables

- » First used on PowerPC, UltraSPARC and Itanium
- » 64bit page tables are large
 - » 4MB page and 64 bit virtual addresses mean 2^{42} page table entries
- » Instead of an entry per page of virtual address space, one entry per page frame of real memory

Inverted Page Tables

Traditional page table with an entry for each of the 2^{52} pages



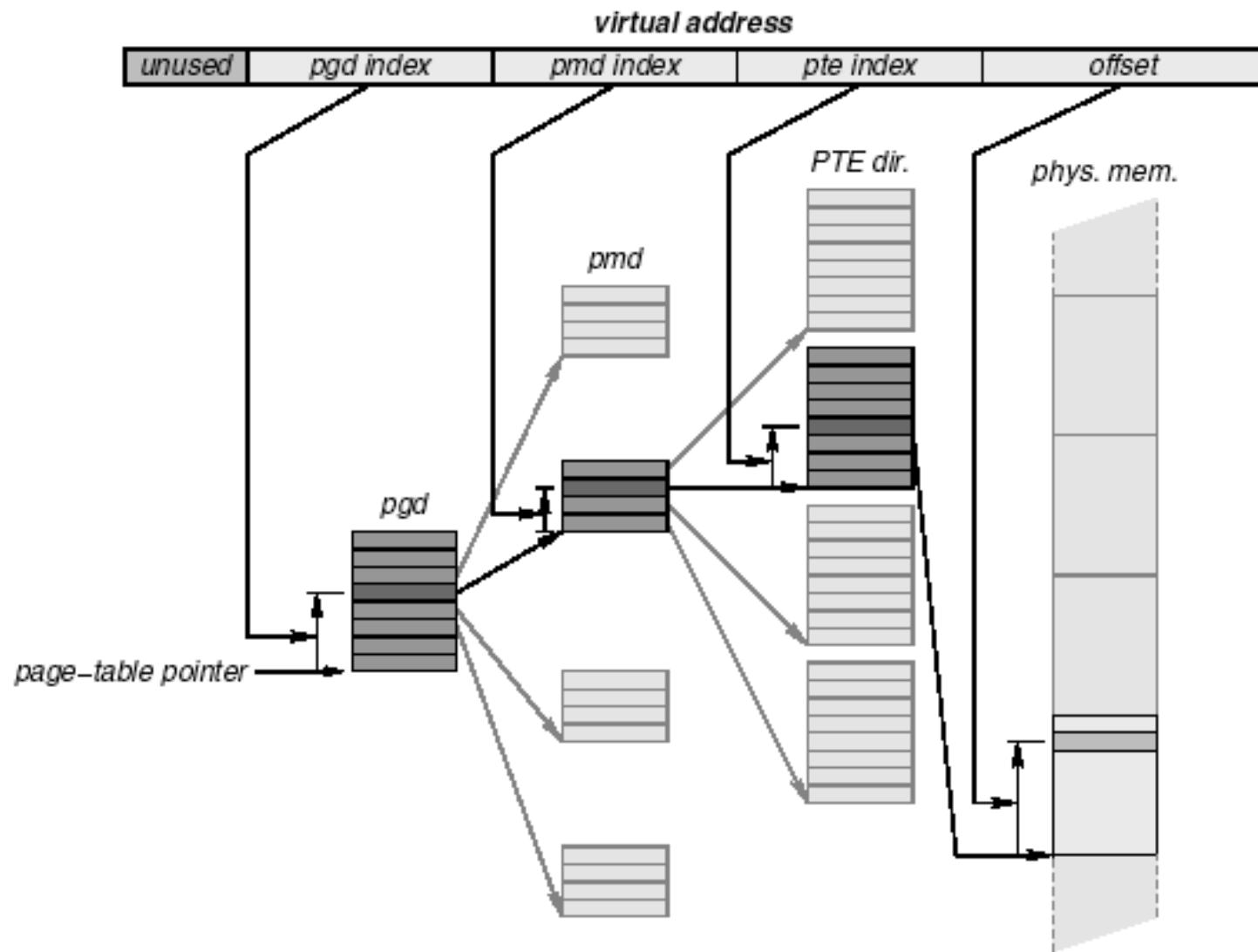
Inverted Page Tables

- » 64-bit virtual addresses and 4 KB page size with 4 GB of RAM, inverted page table requires on 1,048,476 entries.
- » Each entry tracks (virtual address, process) pair
- » Saves a lots of space, but big down side
 - » Virtual -> Physical mapping much harder
 - » Every memory reference must search the page table not just on page faults

Inverted Page Tables

- » TLB can help by holding heavily used pages.
- » On TLB miss, still have to search the inverted table
 - » Feasible way to accomplish is to hash on the virtual address.
 - » If the hash table has as many slots as the machine has physical pages, the bucket will hold a single frame

Linux Page Table



Page Replacement

- What if we are out of frames?
 - Have to discard one
 - May have pages no longer used
- How to identify unneeded?

Working Set

- Can't assume all the pages we need will be in memory when we need them.
- **Working set** - pages a process is referencing in a short period, e.g the set of pages a process is currently using
- **Sliding window** - fixed interval over which the working set is measured.
- **Page replacement** - removing a page we may not need anymore.

Working Set

Page reference string:

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

We never reference
page 5 again after
step 4

Working set	1	1	Event Number
12	12	2	3
12	125	3	4
125	1257	4	5
1257	1257	5	6
1257	1567	6	7
1567	1367	7	8
1367	1367	8	9
1367	1367	9	10
1367	1367	10	11
1367	1467	11	12
1467	1246	12	13
1246	2467	13	14

How about removing 5 after it falls out of the 4 step sliding window?

Working Set

Page reference string:	Working set	1	1	Event Number
1	12		2	
2	12		3	
1	125		4	
2	1257		5	
1	1257		6	
5	1567		7	
1	1367		8	
6	1367		9	
7	1367		10	
1	1367		11	
4	1467		12	
6	1246		13	
7	2467		14	

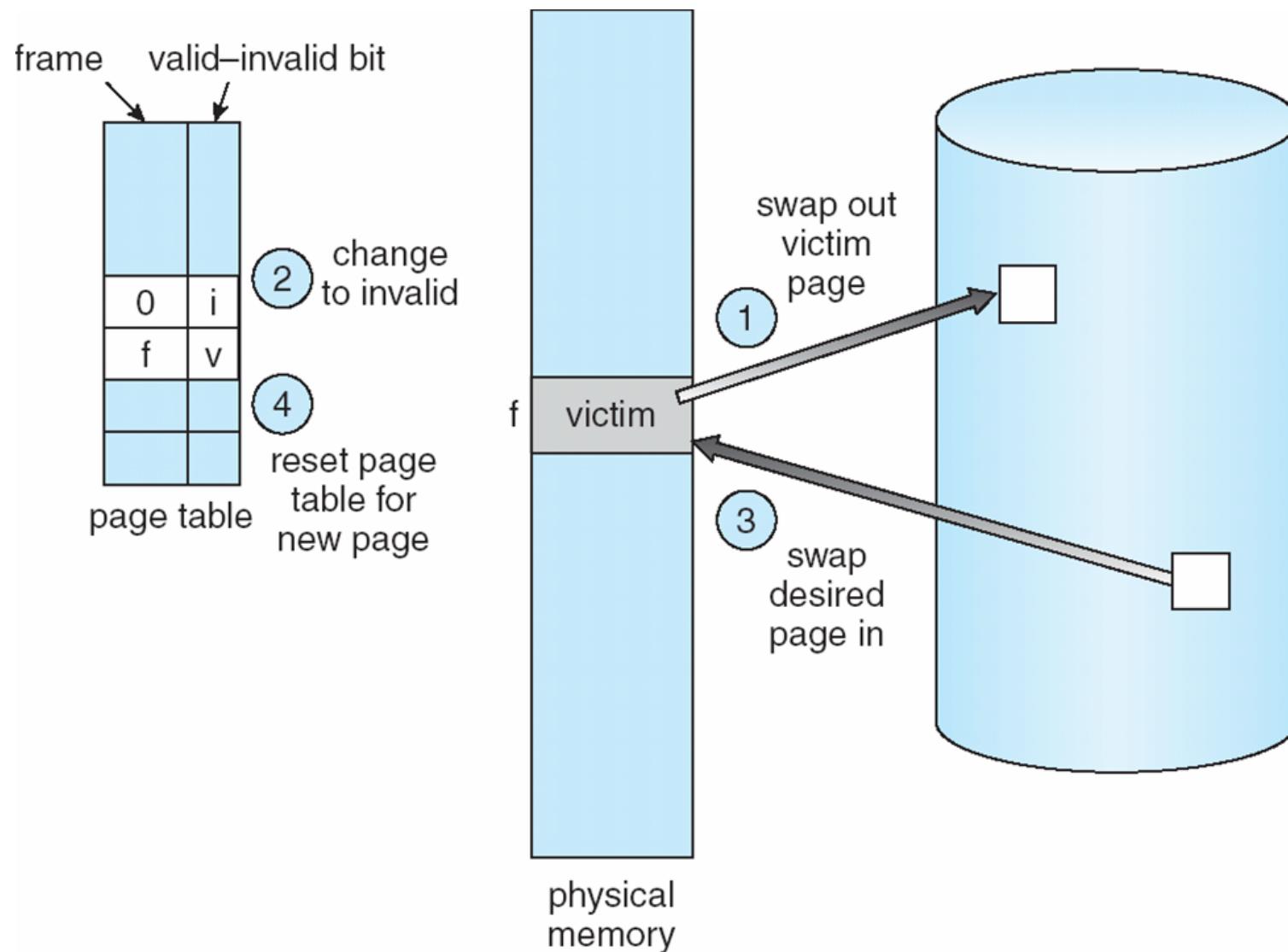
2 is not referenced
between steps 2 and 13

How about removing 5 after it falls out of the 4 step sliding window?

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
 - If there is a free frame, use it
 - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process

Page Replacement



Page Replacement Algorithms

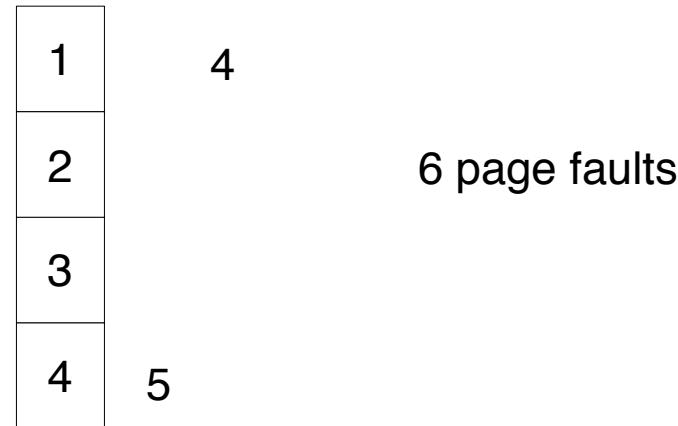
- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

Optimal Algorithm

Replace page that will not be used for longest period of time

4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



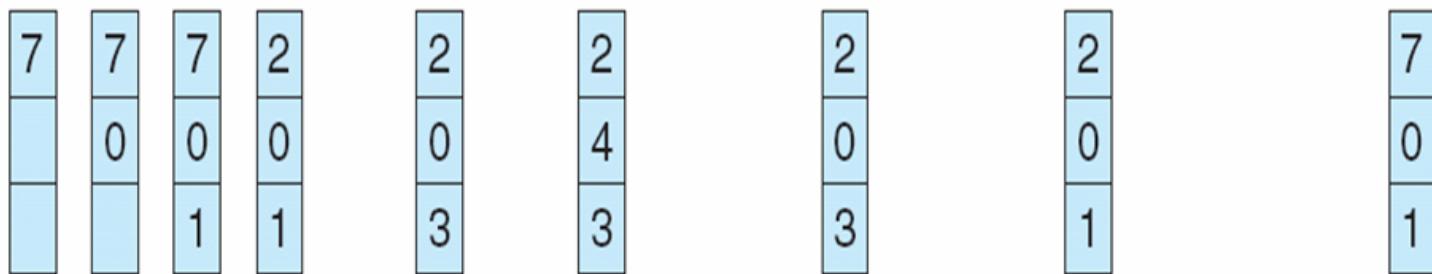
How do you know this?

Used for measuring how well your algorithm performs against a best-case

Optimal Page Replacement

reference string

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



page frames

Optimal

- Unrealizable
- Have to be able to read the future
- Provides a best-case

Not Recently Used

- Each page has two status bit, R (read) and M (modified)
- On startup all bits are set to 0.
- Every page reference the R bit is set, every dirty page the M bit is set
- Every clock tick R bits are cleared

Not Recently Used

- On a page fault, OS inspects all pages and puts them into 4 categories
- Class 0: not referenced, not modified
- Class 1: not referenced, modified
- Class 2: referenced, not modified
- Class 3: referenced, modified
- NRU removes a page at random from the lowest class

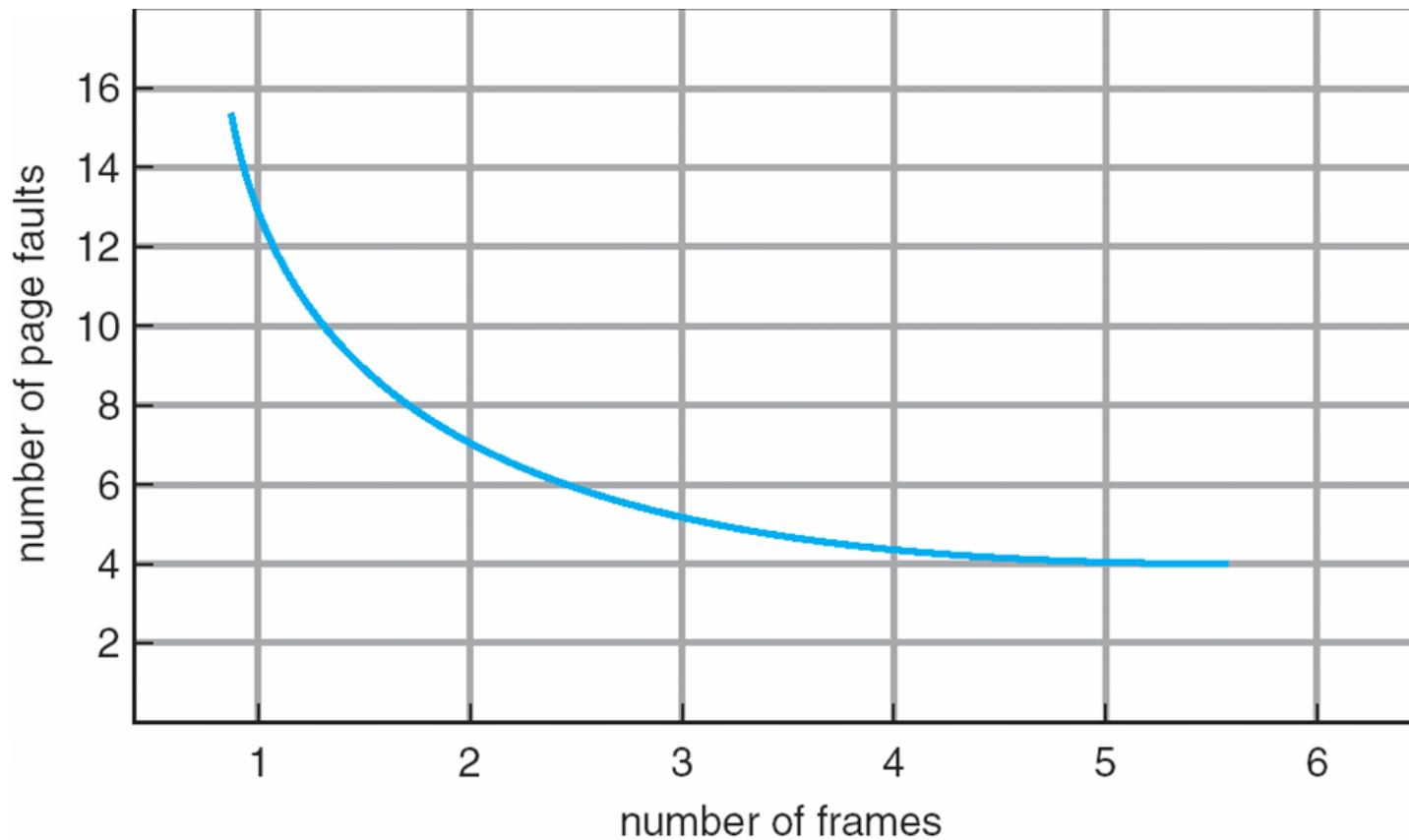
Not Recently Used

- Better to removed a modified page than a clean page frequently used.
- Easy to understand
- Moderately efficient to implement
- Not optimal, but adequate.

First In, First Out

- Eject the oldest page.
- Very low overhead
- Easy to implement
- Poor performance and erratic behavior.
 - Can end up evicting the most frequently used page
- Subject to Bélády's anomaly

Graph of Page Faults Versus The Number of Frames



Bélády's anomaly

- Increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	3	2	4	4	4	1	0	0
	3	2	1	0	3	2	2	2	4	1	1	
Oldest Page	3	2	1	0	3	3	3	2	4	4		

3 page frames
9 faults (red)

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	0	0	4	3	2	1	0	4
	3	2	1	1	1	0	4	3	2	1	0	
	3	2	2	2	1	0	4	3	2	1	0	
Oldest Page	3	3	3	2	1	0	4	3	2	1	0	

4 page frames
10 faults (red)

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

- 4 frames

1	1	4	5
2	2	1	3
3	3	2	4

9 page faults

- Belady's Anomaly: more frames = more page faults

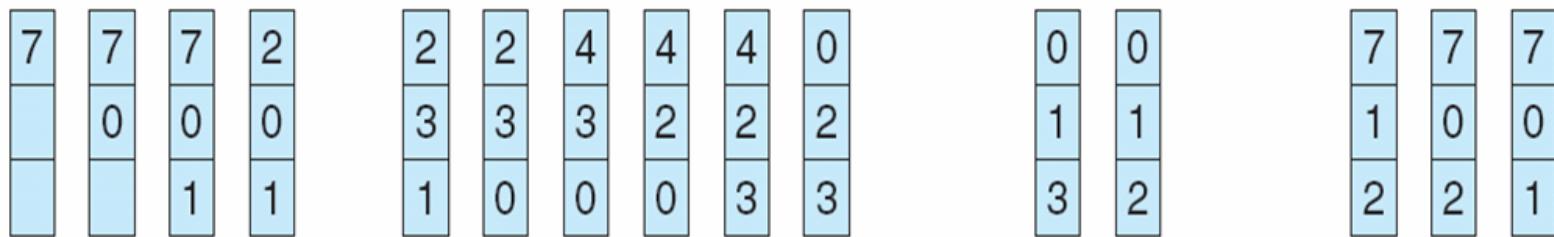
1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

FIFO Page Replacement

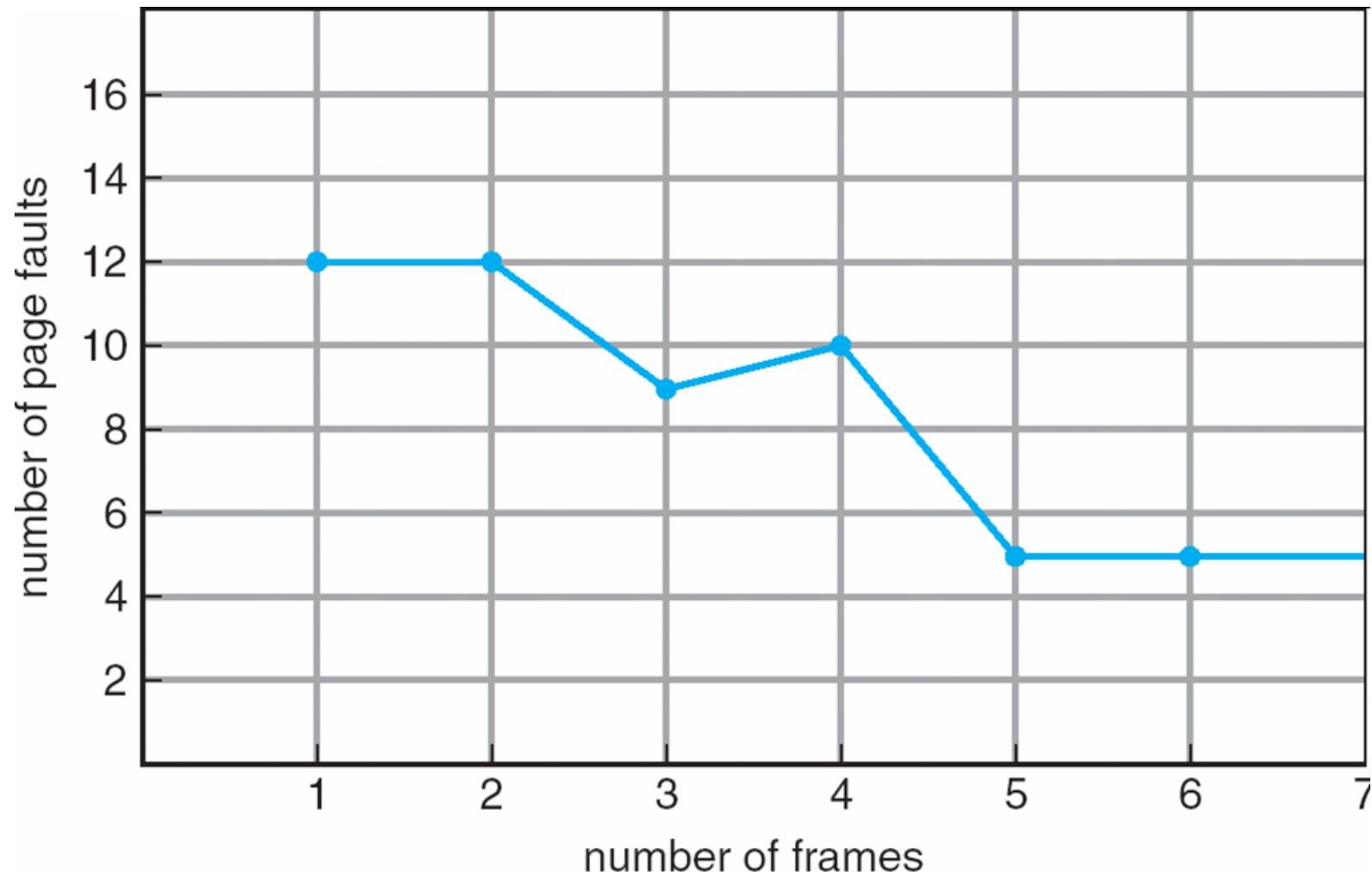
reference string

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

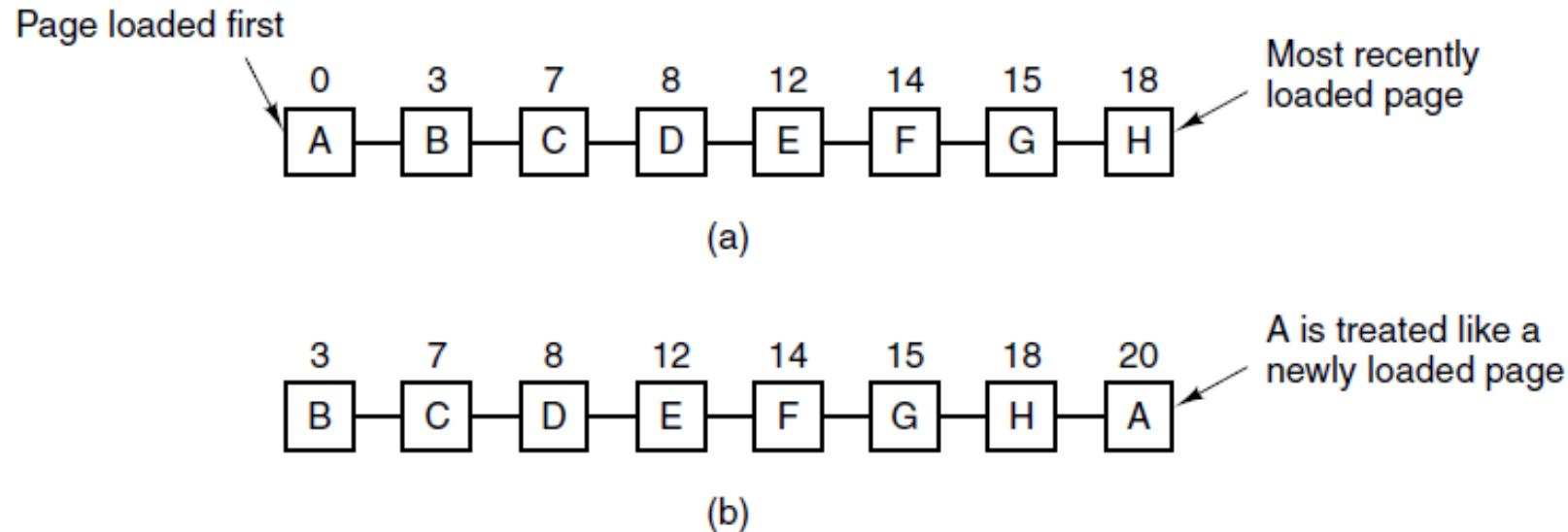


page frames

FIFO Illustrating Belady's Anomaly

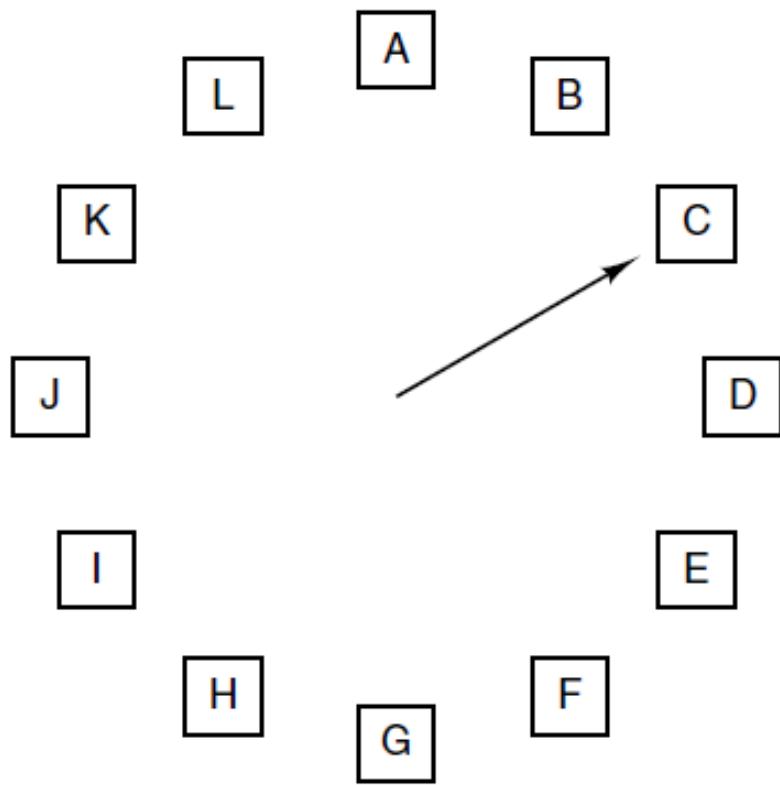


Second Chance Algorithm



- Sorted in reference time order.
- When page referenced, set R bit
- When faulted check oldest page, if R bit set then set page time to current time, clear R bit and move to the end.
- Repeat until page found with no R bit set.

Clock Page Algorithm



When a page fault occurs,
the page the hand is
pointing to is inspected.
The action taken depends
on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

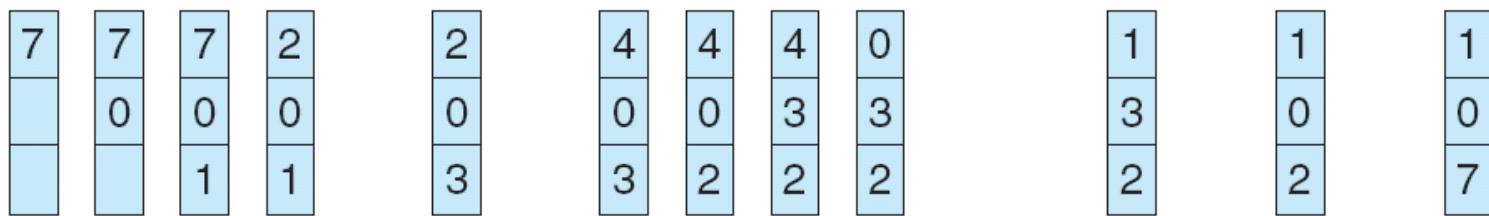
1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

- Impractical to track.
 - Would need one additional memory access for each memory access or
 - Linked list of each page with most recent in front.
- Can approximate, similar to NRU
 - Need hardware assistance
 - Page reference bit in page table

LRU Page Replacement

reference string

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



page frames

LRU Algorithm

- Stack implementation – keep a stack of page numbers in a double link form
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - No search for replacement

LRU Algorithm

- Stack implementation – keep a stack of page numbers in a double link form
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - No search for replacement

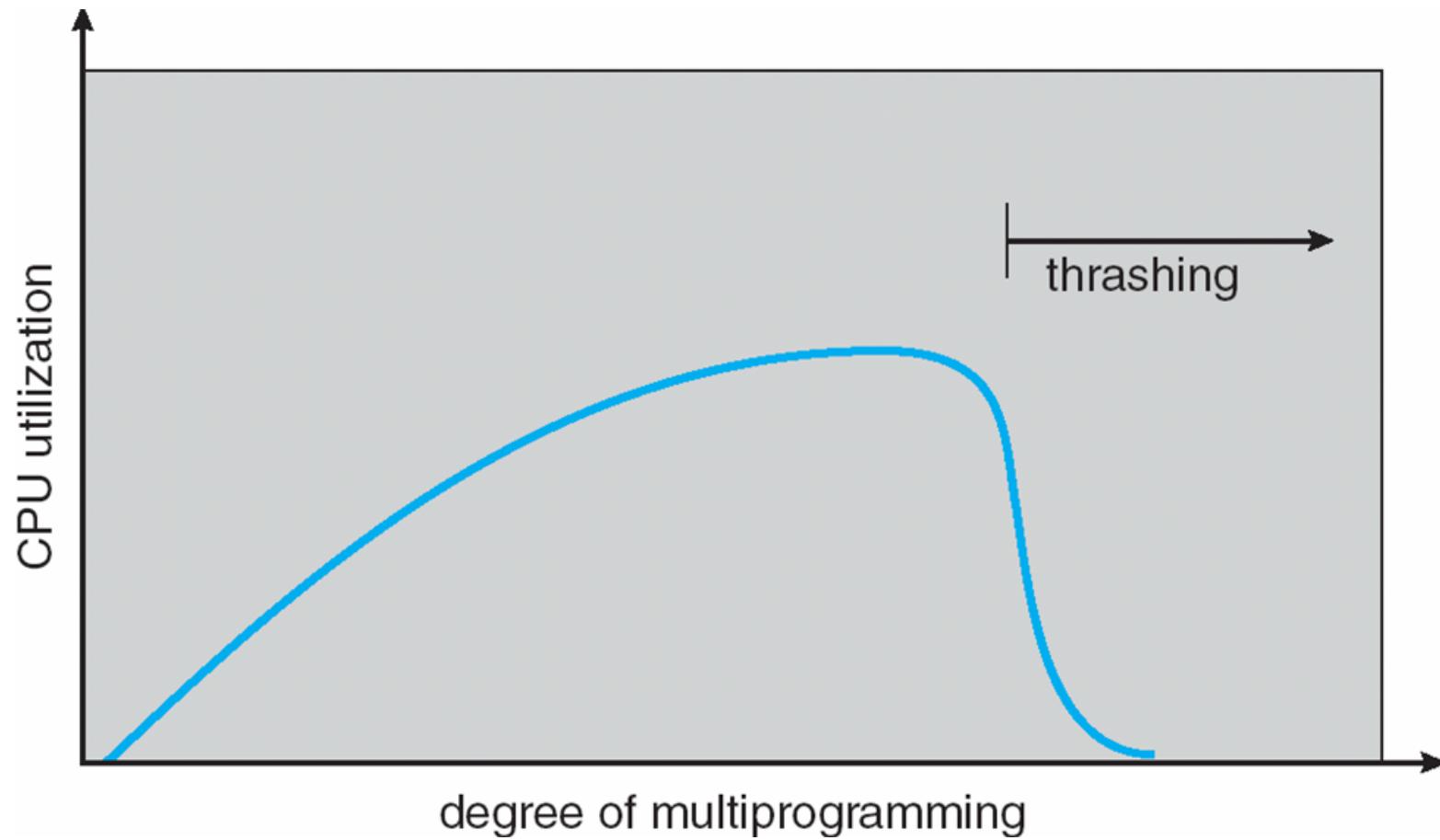
Reference Counter Variation

- Also known as [Aging](#)
- As reference bits are cleared, shift into a counter
 - one for each page
- Smallest counter referenced is least recently

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
 - low CPU utilization
 - operating system thinks that it needs to increase the degree of multiprogramming
 - another process added to the system
- **Thrashing** – a process is busy swapping pages in and out

Thrashing (Cont.)



Demand Paging and Thrashing

Why does demand paging work?
Locality model

Process migrates from one locality to another

Localities may overlap

Why does thrashing occur?
 Σ size of locality > total memory size

Why file systems?

- » Storage needs exceed virtual address space.
- » Virtual address space storage is lost when a process terminates
- » Multiple processes may need the same data
 - » Can't access other process' address space

Files

- » Files are logical units of information created by processes
- » O/S contains millions
- » An address space
- » Persistent: not be affected by process creation and termination.
- » Deleted only when explicitly removed.

Files

- » How they are structured, named, accessed, used, protected, implemented, and managed are major topics in operating system design.
- » File System: part of the operating system dealing with files

File System

- » User's perspective, the most important:
 - » How it appears
 - » How files are named and protected
 - » Operations are allowed on files
- » File system designer perspective:
 - » Linked lists or bitmaps to track free storage
 - » How many sectors there are in a logical disk block

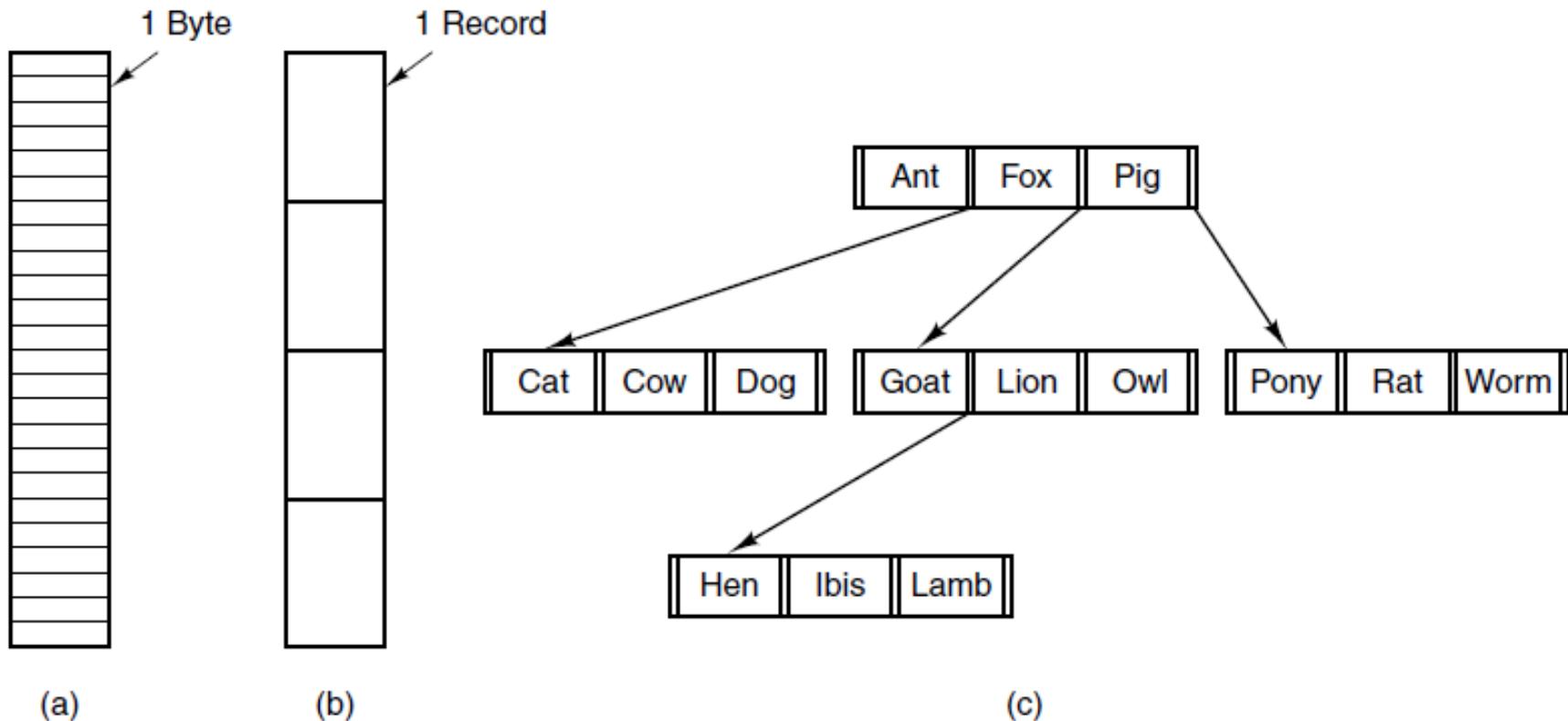
Files

- » File is an abstract mechanism that provides a way to store information on the disk and read it back later.
- » Shields the user from where the bits are stored and how it's retrieved.

File Naming

- » The exact rules for file naming vary
 - » Most modern OS allow 255+ characters
 - » Case sensitive / Case insensitive
 - » Unicode
 - » File extensions (8 + 3)
 - » Unix: convenience
 - » Windows: cares

File Structure



Three kinds of files. (a) Byte sequence.
(b) Record sequence. (c) Tree.

Three Essential Requirements

1. It must be possible to store a very large amount of information.
2. The information must survive the termination of the process using it.
3. Multiple processes must be able to access the information at once.



File Types

- » Regular files are the ones that contain user information
- » Directories are system files for maintaining the structure of the file system
- » Character special files are related to input/output and used to model serial I/O devices
- » Block special files are used to model disks.
- »

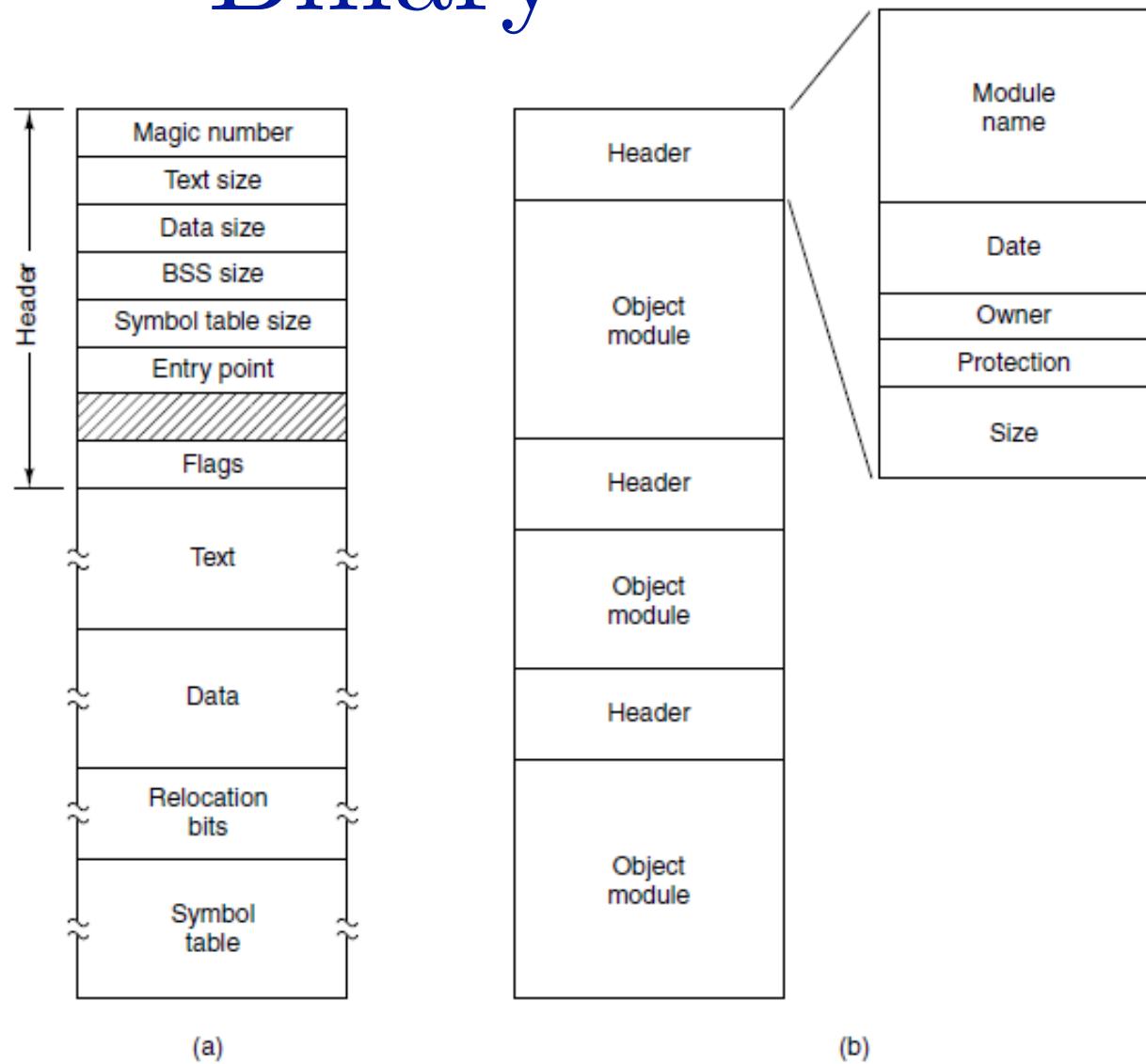
Regular Files

- » Regular files are generally either ASCII files or binary files.
- » Some systems each line is terminated by a carriage return character.
- » Others, the line feed character is used.
- » Windows uses both

ASCII

- » Big advantage of ASCII files
 - » Can be displayed and printed as is
 - » Edited with any text editor.
- » Easy to connect the output of one program to the input of another, as in shell pipelines.

Binary



(a) An executable file. (b) An archive

Binary Files

- » Although technically the file is just a sequence of bytes, the operating system will execute a file only if it has the proper format.
- » Five sections: header, text, data, relocation bits, and symbol table.
- » Unix uses the ELF format

Header

- » Header
 - » Magic Number - Identifies the file as executable*
 - » Magic numbers are common in programs across many operating systems.
 - » Magic numbers implement strongly typed data
 - » Form of in-band signaling to the controlling program.

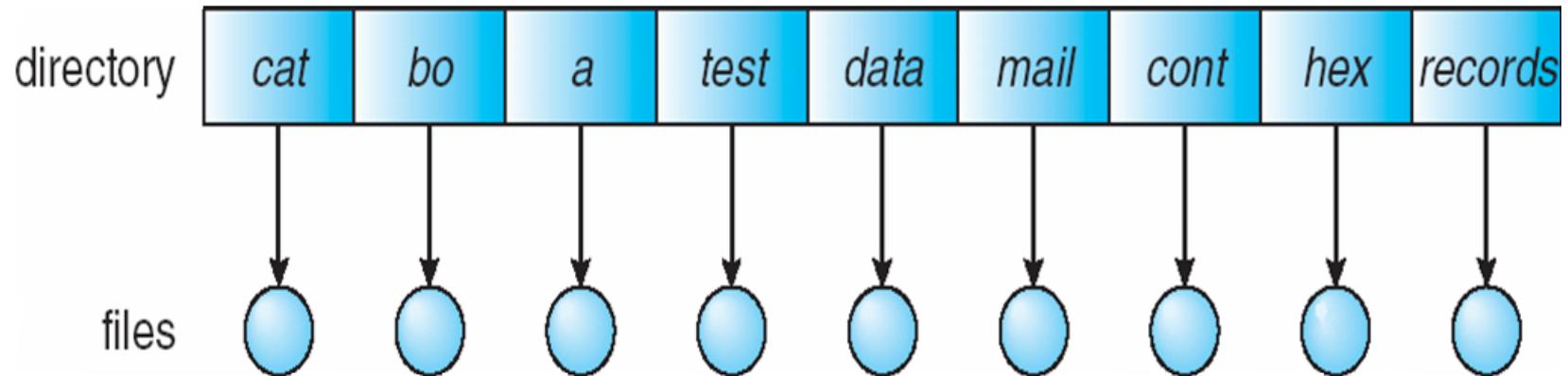
Header (cont)

- » Sizes of the various pieces of the file
- » Address at which execution starts
- » Flag bits
- » Following the header
 - » Text segment
 - » Data segment

Strongly Typed Files

- » Having strongly typed files causes problems whenever the user does anything that the system designers did not expect.
- » user friendliness may help novices
- » drives experienced users up the wall

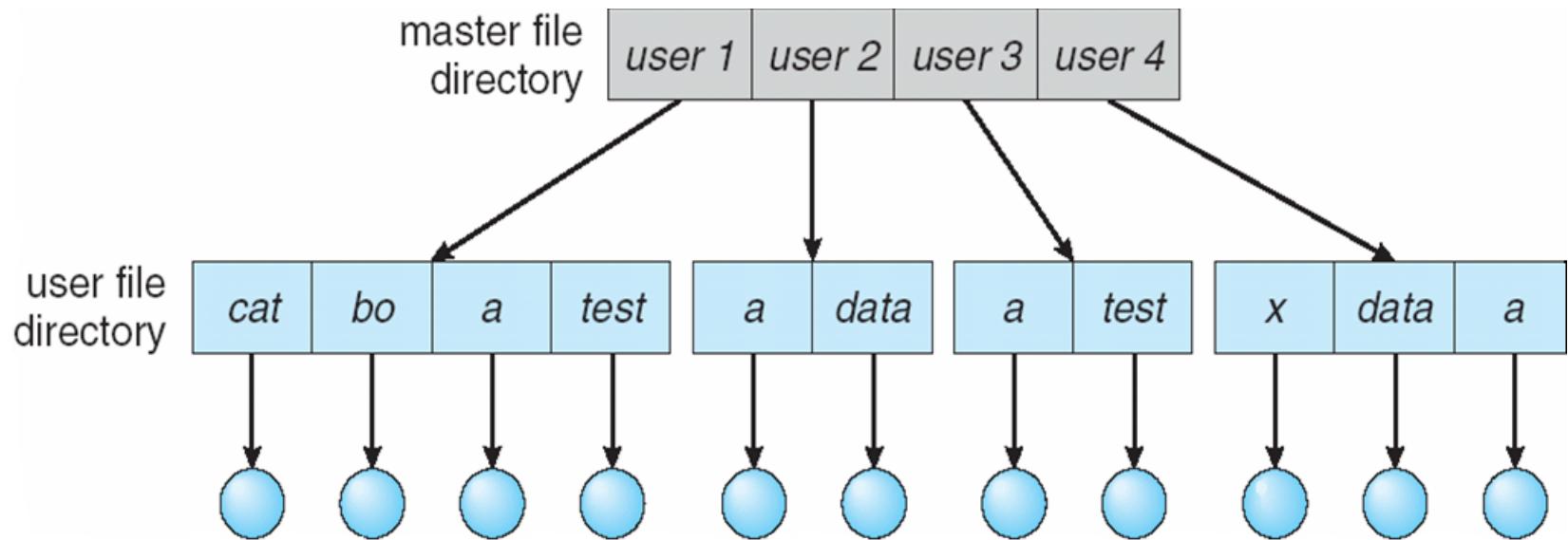
Directories



- » The simplest form of directory system is having one directory containing all the files.
- » Naming problem
- » Grouping problem

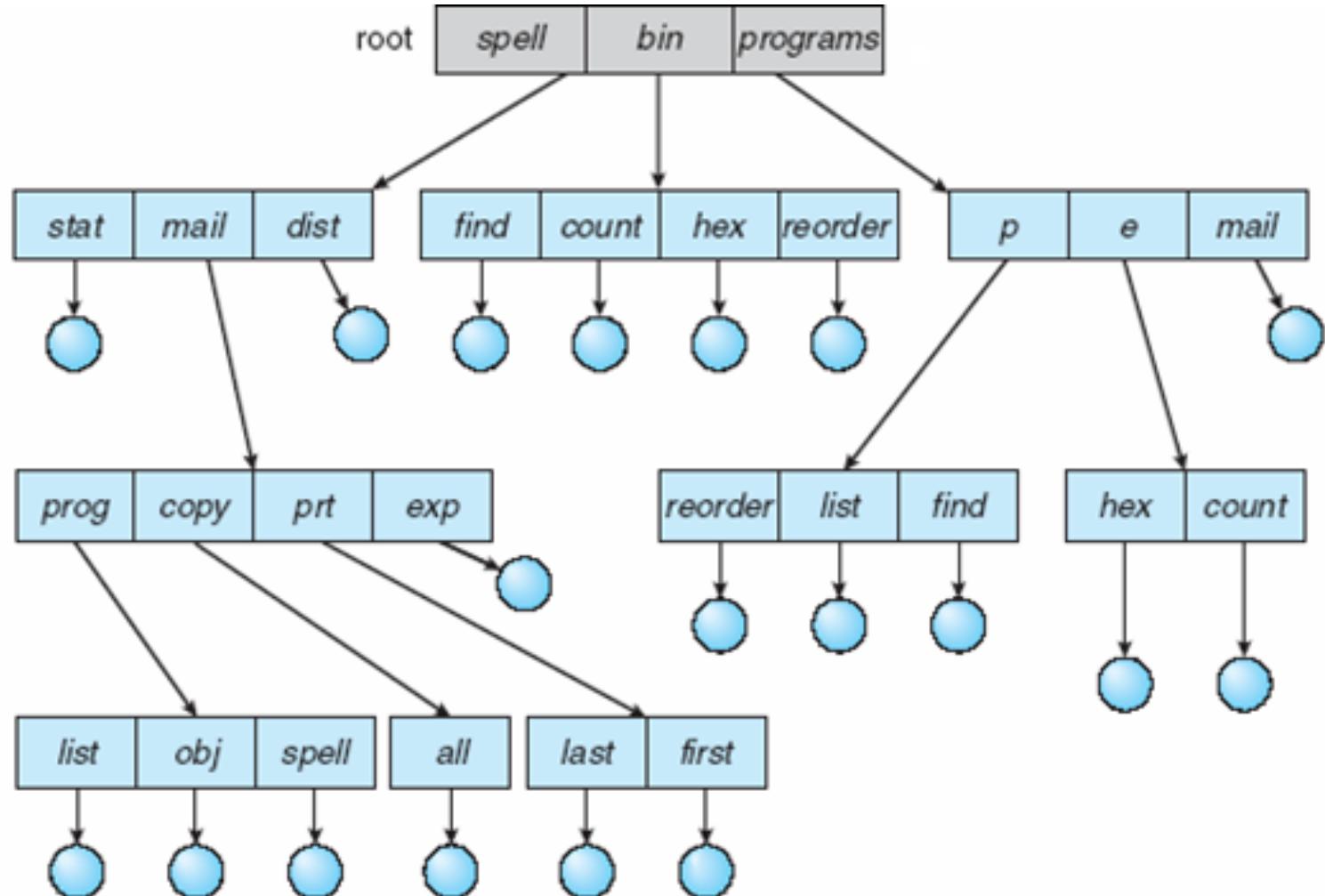
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
 - `cd /spell/mail/prog`
 - `type list`

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

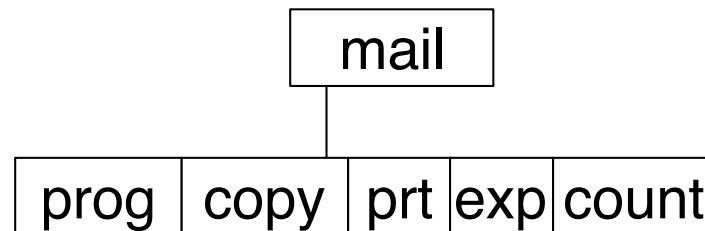
rm <file-name>

- Creating a new subdirectory is done in current directory

mkdir <dir-name>

Example: if in current directory **/mail**

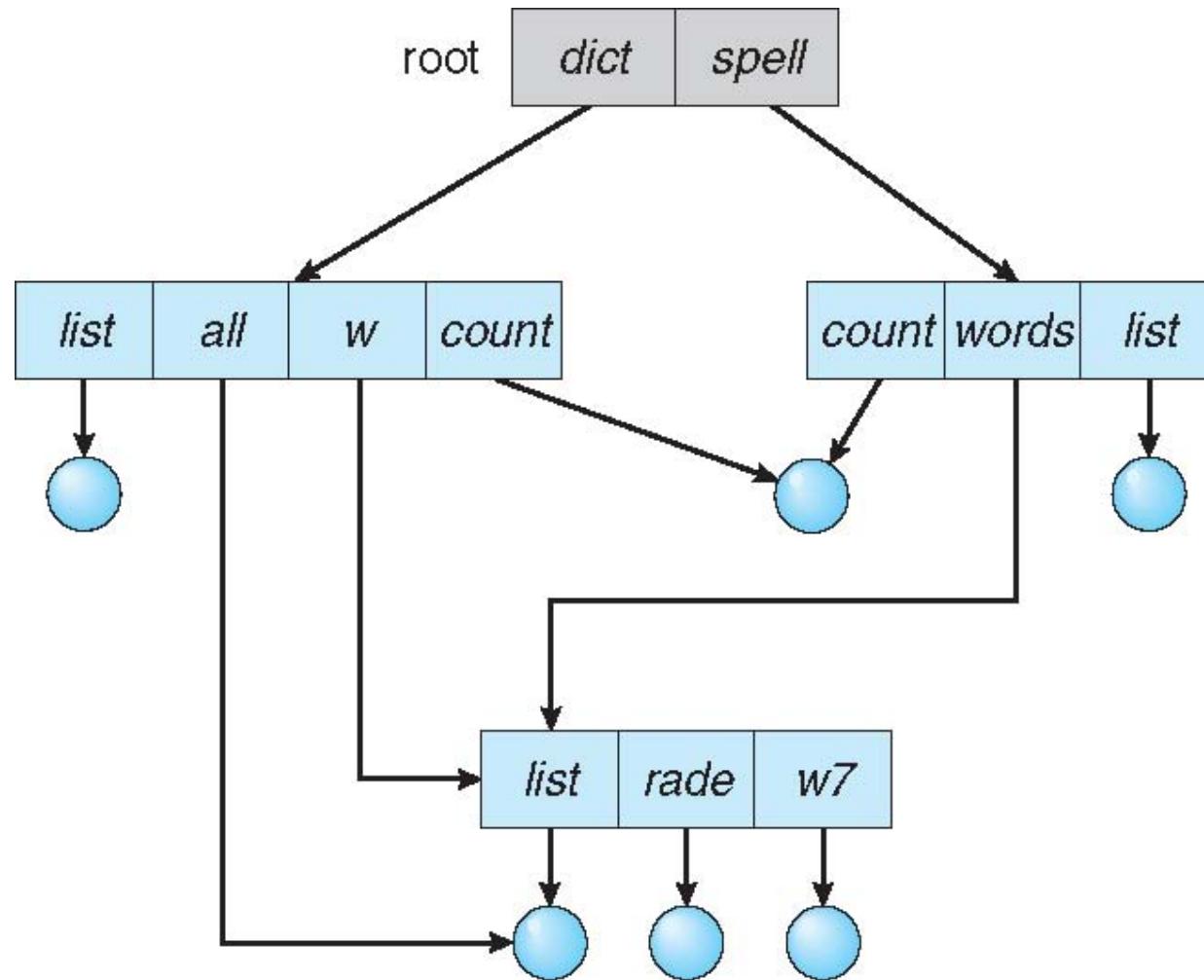
mkdir count



Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Directed Acyclic-Graph Directories

- Have shared subdirectories and files



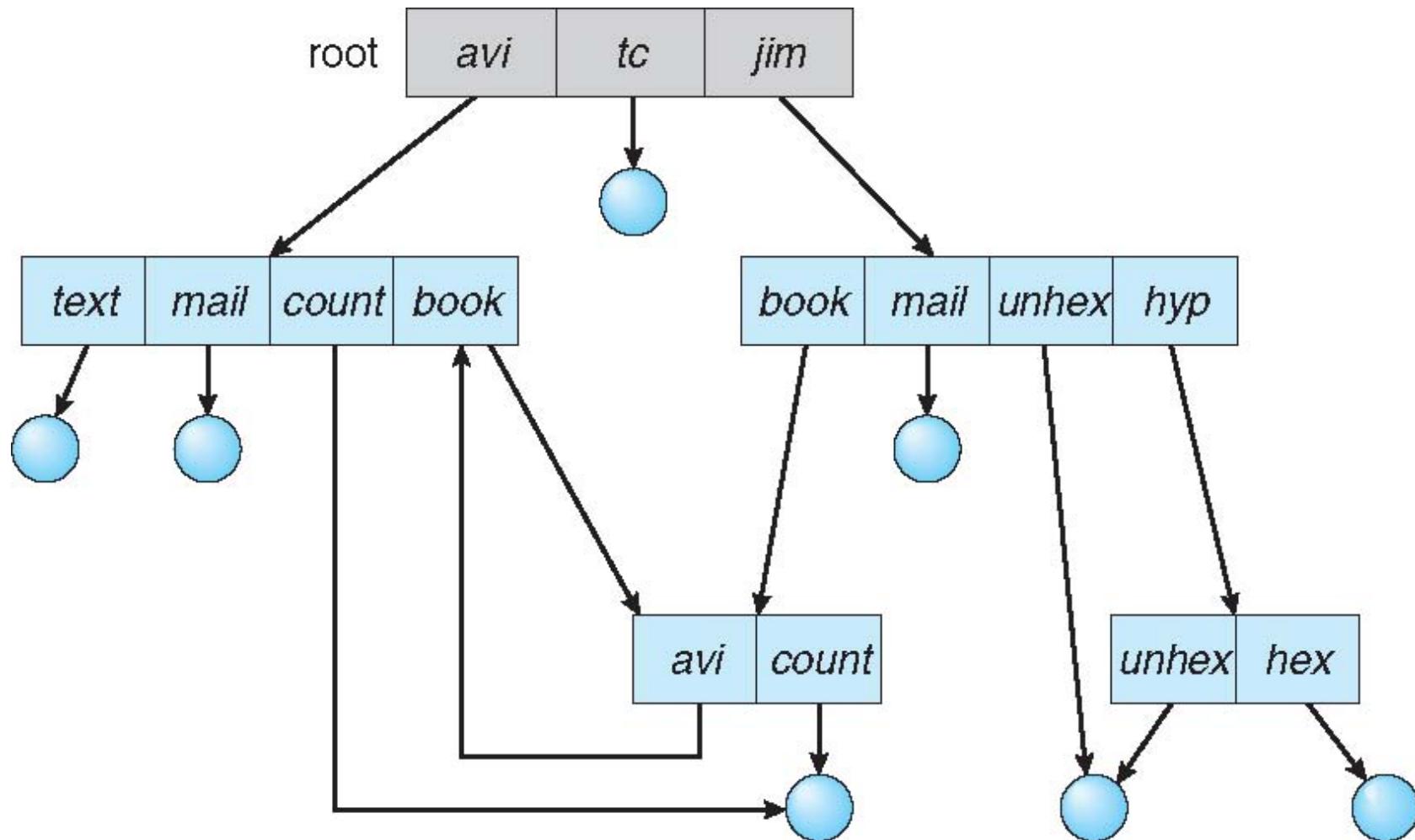
Acyclic-Graph Directories (Cont.)

- ❑ Two different names (aliasing)
- ❑ If *dict* deletes *list* ⇒ dangling pointer

Solutions:

- ❑ Backpointers, so we can delete all pointers
Variable size records a problem
- ❑ Backpointers using a daisy chain organization
- ❑ Entry-hold-count solution
- ❑ New directory entry type
 - ❑ **Link** – another name (pointer) to an existing file
 - ❑ **Resolve the link** – follow pointer to locate the file

General Graph Directory



General Graph Directory (Cont.)

- How do we guarantee no cycles?
 - Allow only links to file not subdirectories
 - Garbage collection
 - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

Disk-Space Management

- » Management of disk space is a major concern to file system designers
- » Two general strategies for storing an n-byte file
 - » n consecutive blocks of disk are allocated
 - » split the file into a number of not always contiguous blocks

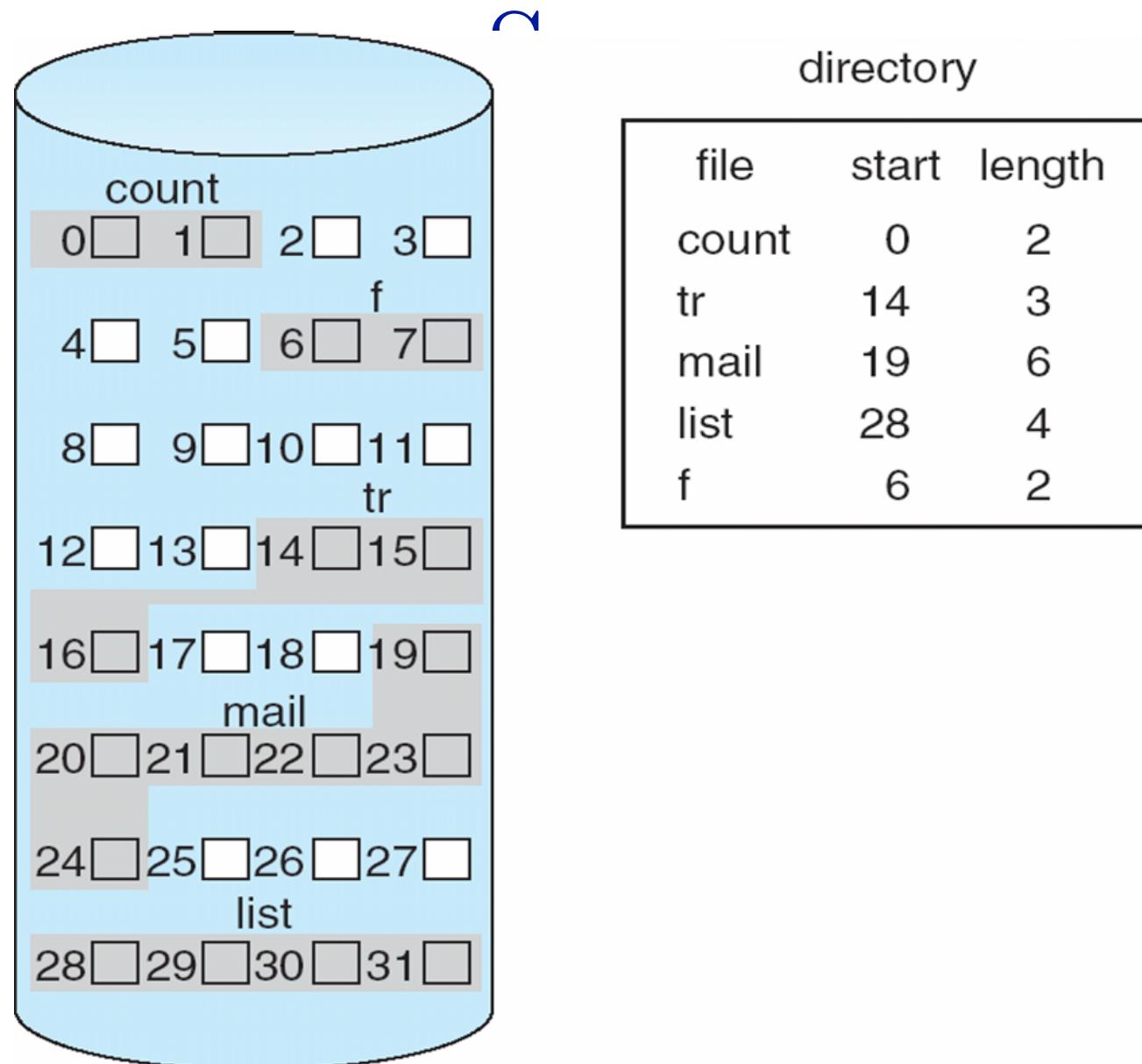
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

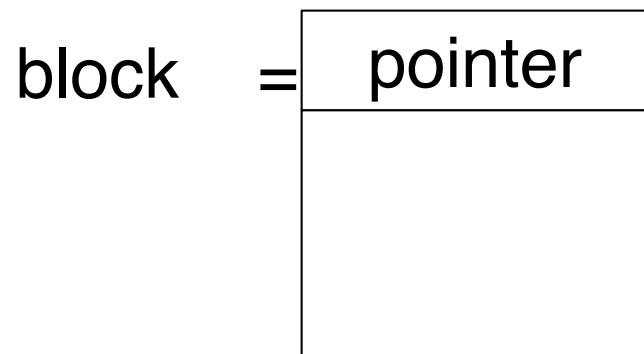
- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Contiguous Allocation of Disk



Linked Allocation

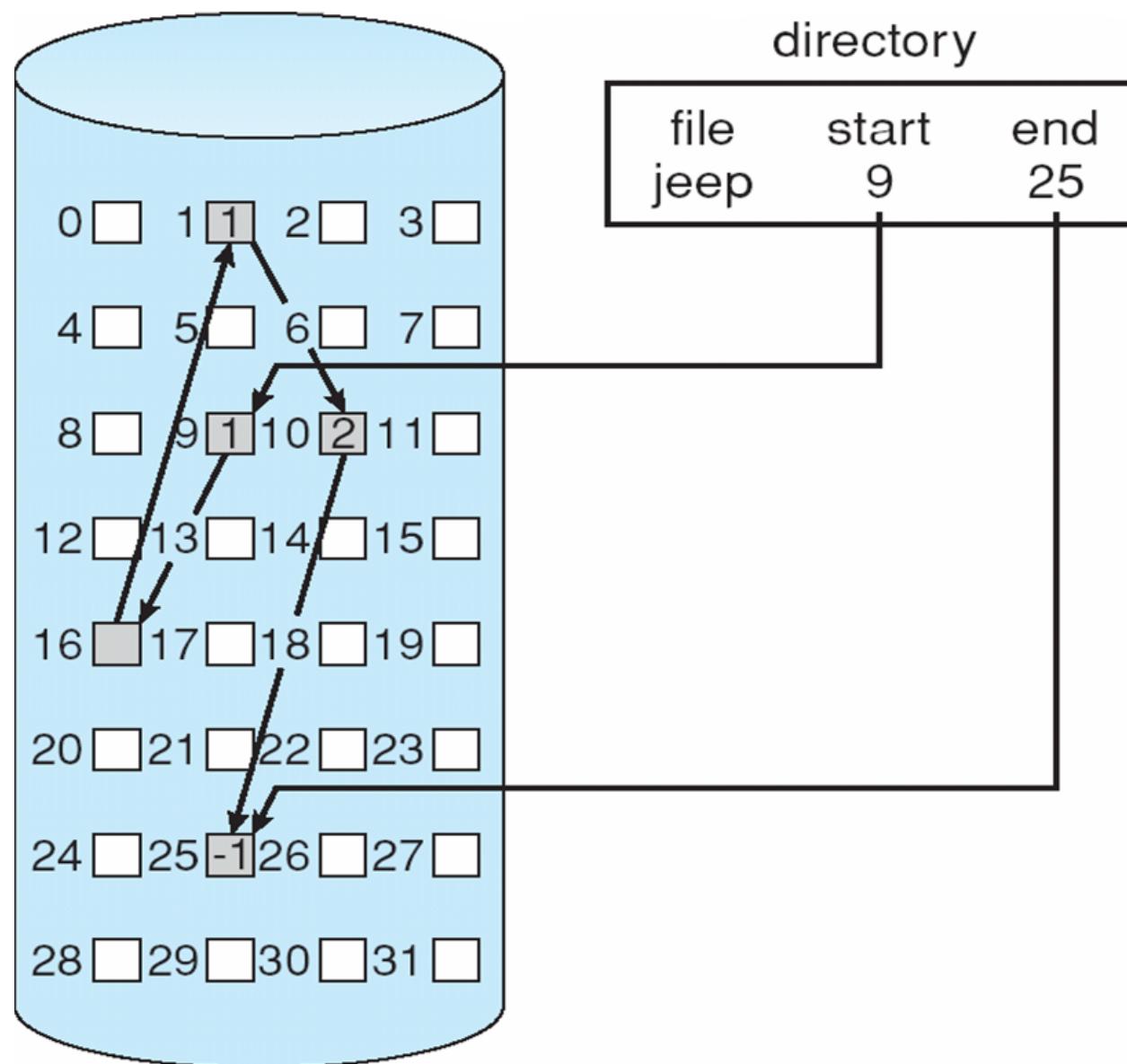
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



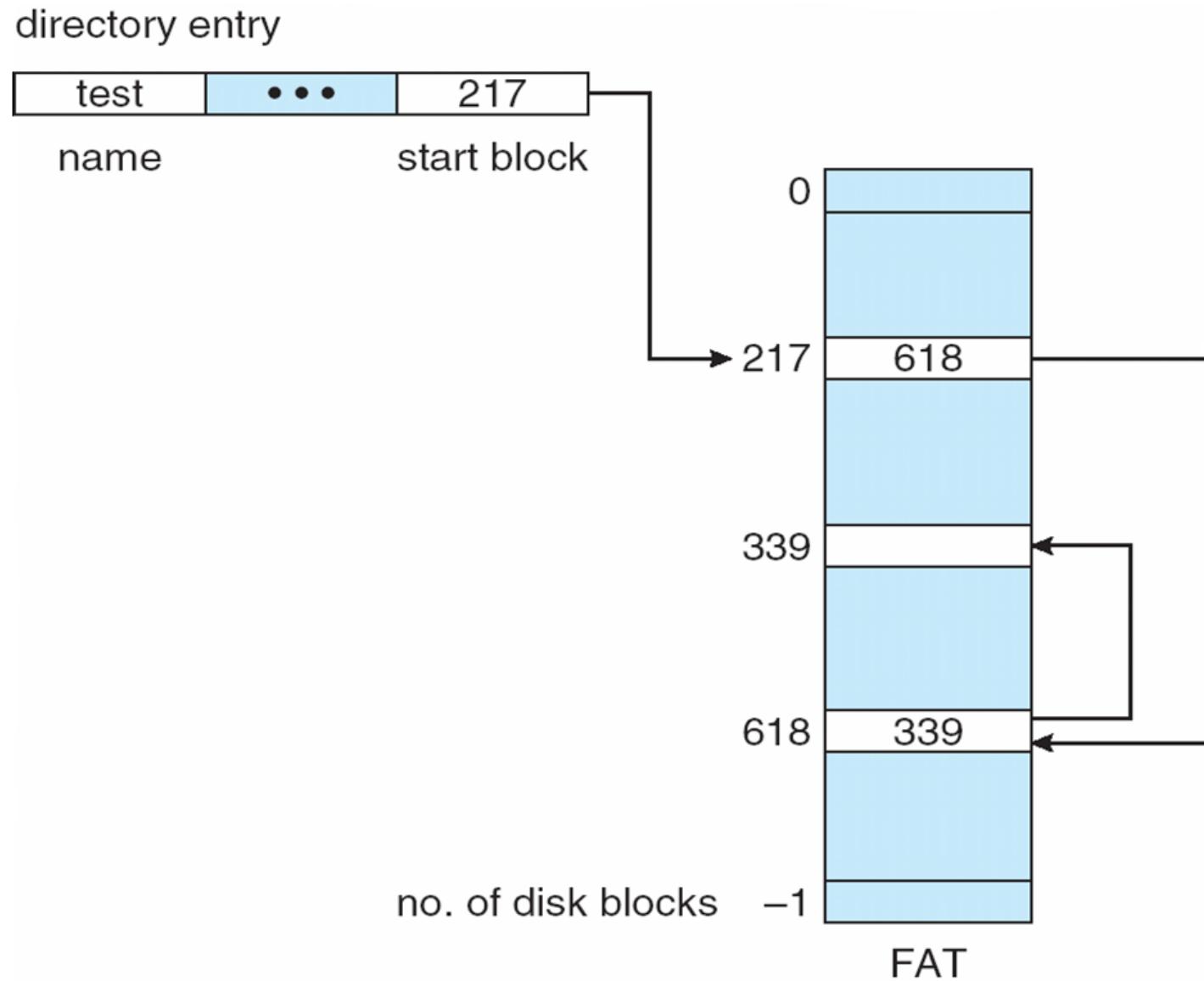
Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access

Linked Allocation

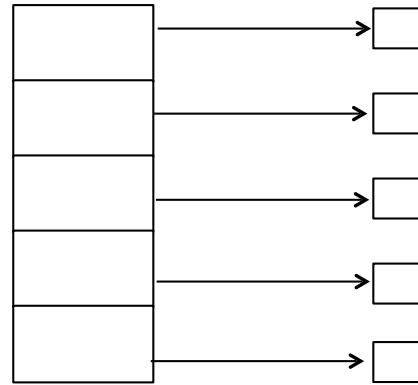


File-Allocation Table



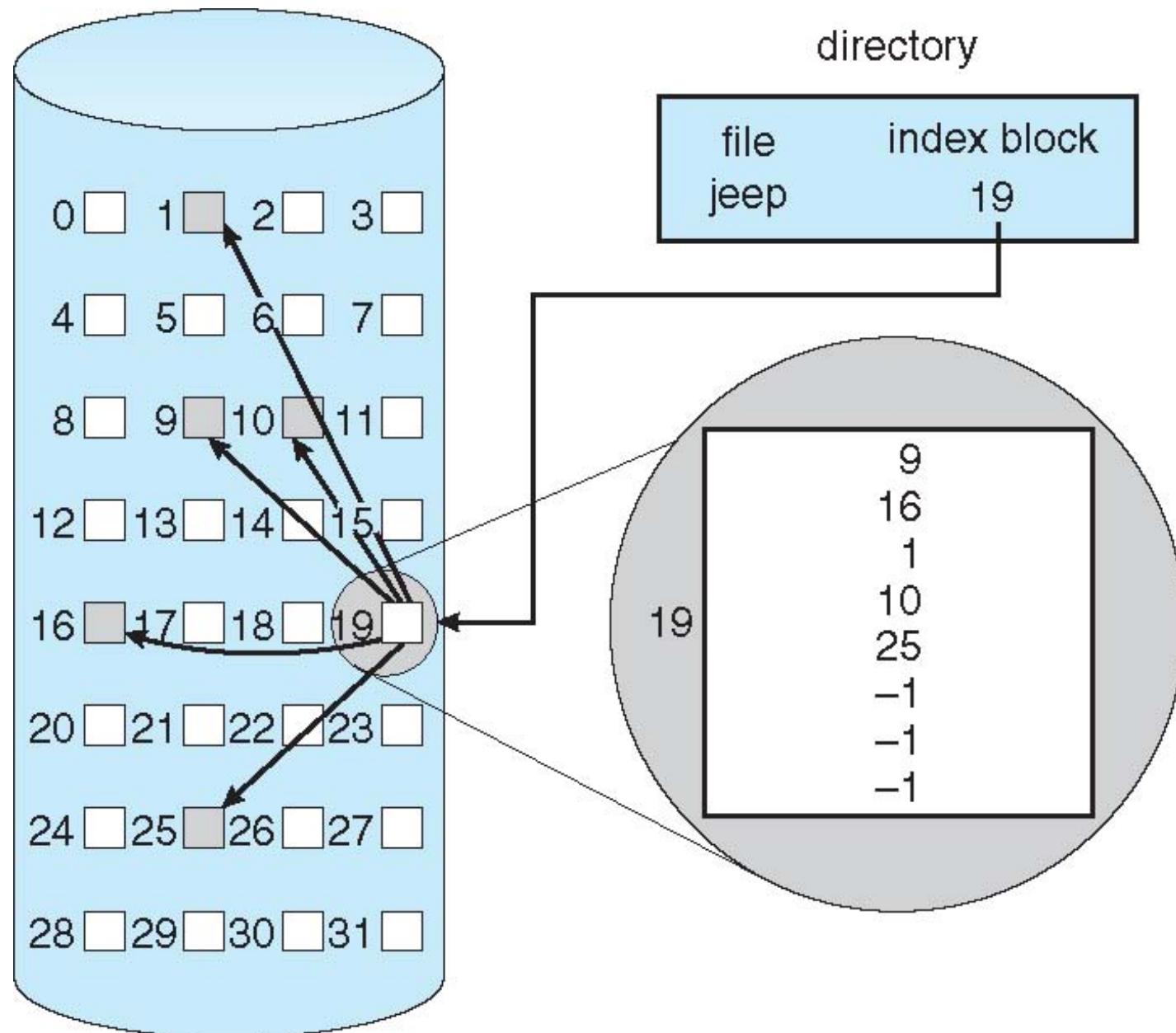
Indexed Allocation

- ② Brings all pointers together into the **index block**
- ② Logical view



index table

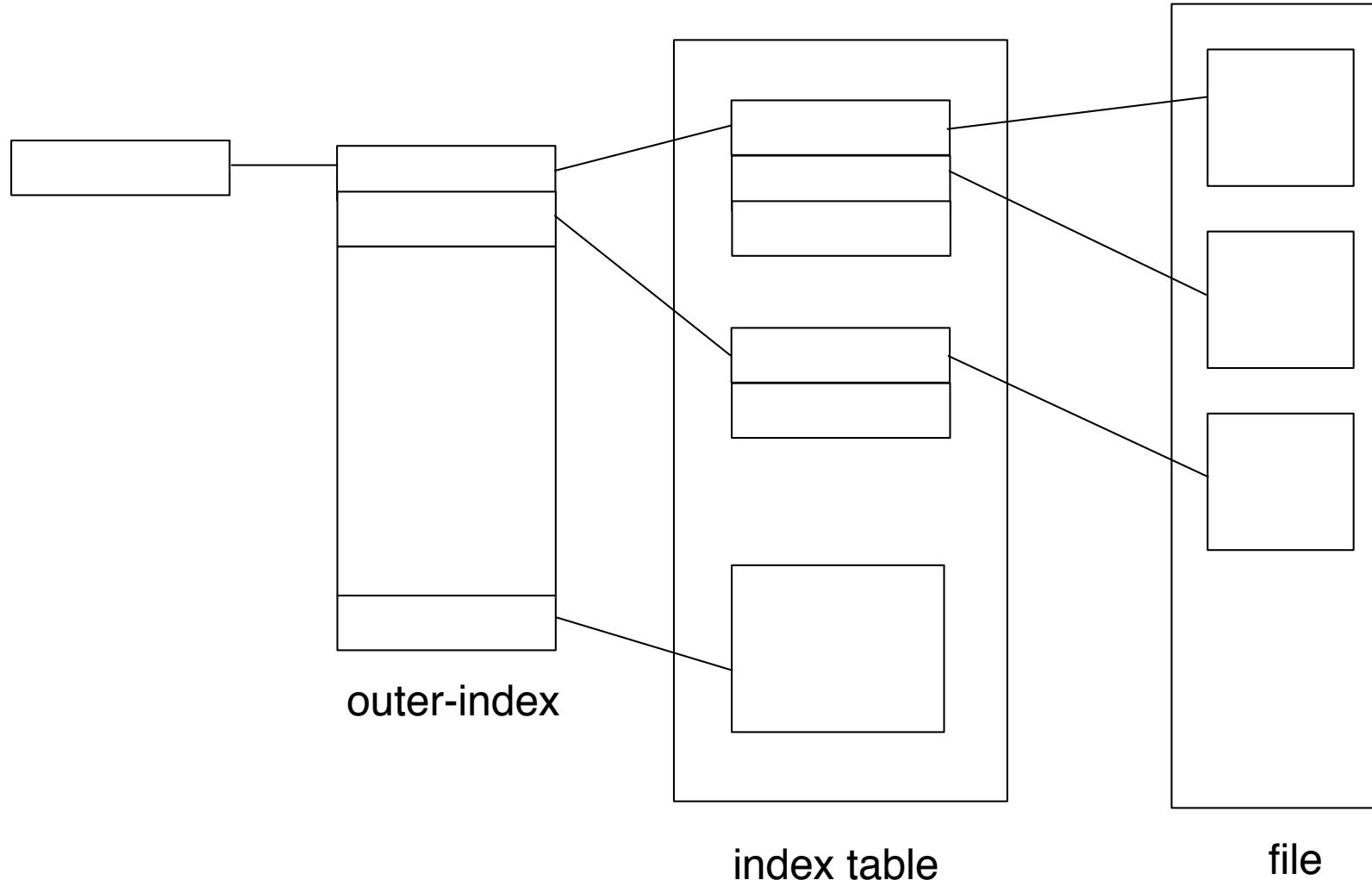
Example of Indexed Allocation



Indexed Allocation (Cont.)

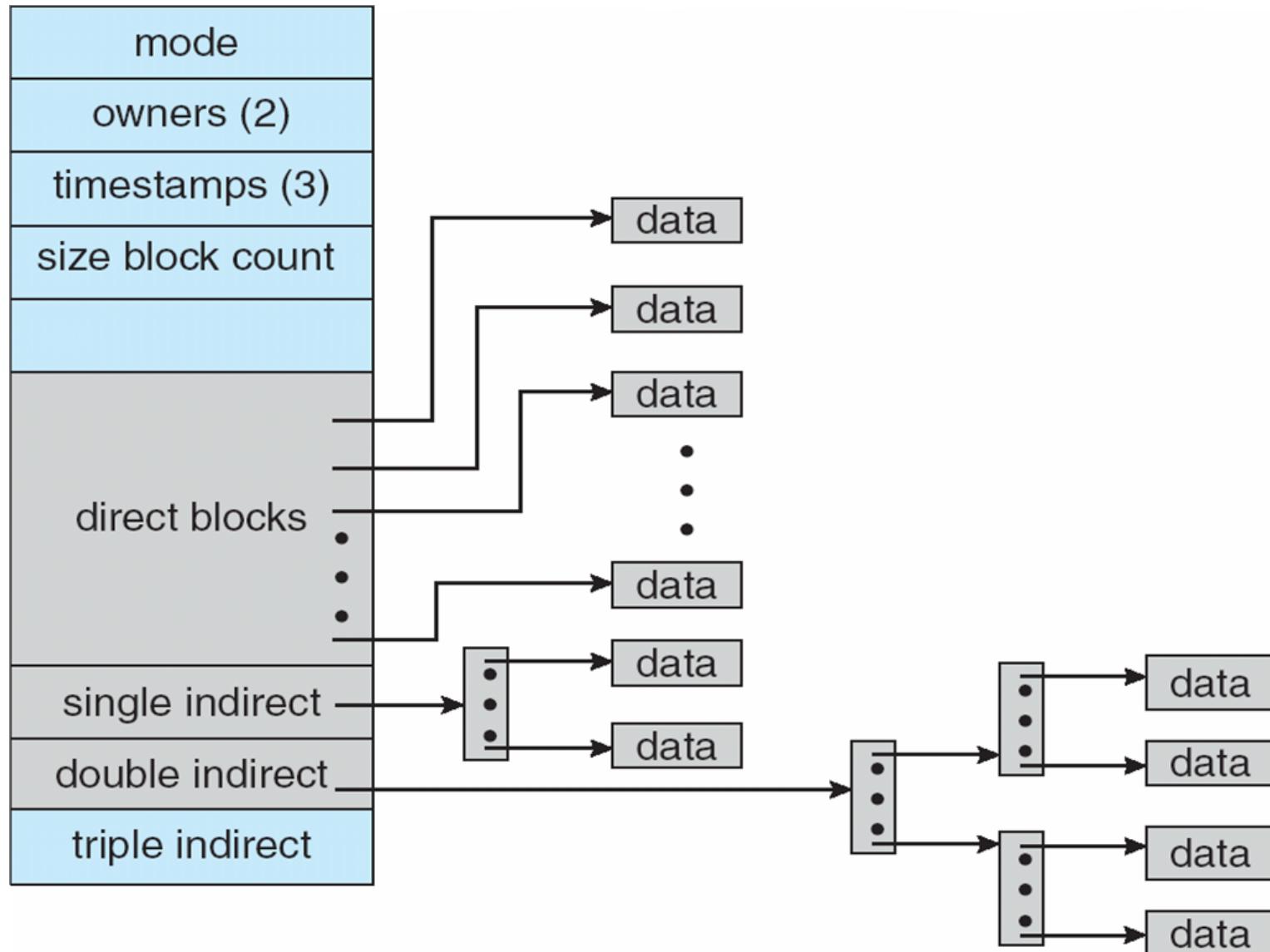
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX UFS

(4K bytes per block)



Block Size

- » How big should the blocks be?
- » Sector, tracks and cylinders are good candidates
- » Large block size means even 1 byte files tie up a whole cylinder/sector
- » Small block size means file span multiple blocks
 - » Multiple seeks and rotational delays

Block Size

- » Consider a disk with 1MB per track, a rotation time of 8.33 ms and a seek time of 5 ms.
- » The time in ms to read a block of k bytes is the sum of the seek, rotational delay and transfer times:

$$5 + 4.165 + (k / 1000000) * 8.33$$

File System Backups

- » File system destruction is far greater disaster than computer destruction.
- » PCs can be replaced within the hour*
- » If the file system is lost restoring the missing information may be time consuming or impossible
- » Backups not viewed as important until it's too late

File System Backups

- » Backups are generally made to handle one of two potential problems:
 - » Recover from disaster
 - » Fire, Flood, Disk crash
 - » Rare, so people don't backup
 - » Recover from stupidity.
 - » Accidental file deletion

File System Backups

- » Backups take time and occupy a lot of space
- » Do you backup the entire file system or part?
- » Applications can be reinstalled from media
- » Unix system /dev backup would be bad
- » Wasteful to backup files that haven't changed since last backup

File System Backups

- » Incremental dump
 - » Complete backup weekly or monthly
 - » Daily backup of modified files daily.
 - » Recovery is more complicated.
- » Do you encrypt your backup?
 - » Corrupted byte may make entire backup unreadable.

File System Backups

- » Backing up a live filesystem is difficult.
- » Snapshotting a moving target
- » Algorithms devised to snapshot critical state by copying critical data structures
- » Frozen at moment of capture

File System Backups

- » Security
 - » Can't leave the tapes lying around.
 - » Backups must be kept off site.
 - » Daily backups do no good if a fire destroys them all.
 - » Now two sites must be secured.

File System Backups

- » Two strategies for dumping a disk to backup:
 - » Physical and Logical
- » Physical Dump
 - » Starts at block 0 and writes all disk blocks in order until it reaches the end of the disk.
 - » Very simple to write. Very fast
 - » Wasteful. Backing up unused blocks.
 - » Skip but have to tap with block id.

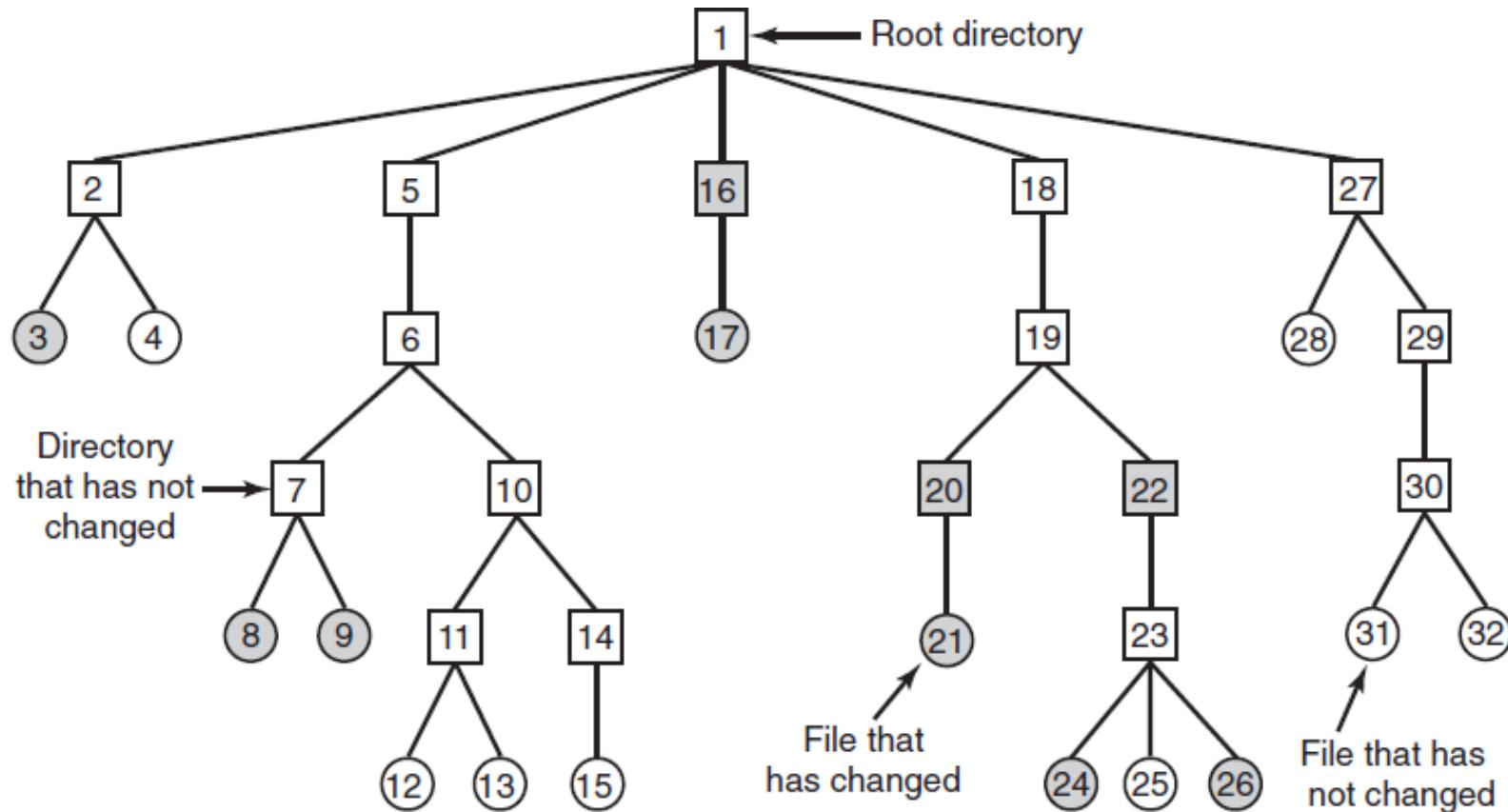
File System Backups

- » Physical Dump cont.
 - » Bad blocks
 - » Always present.
 - » Low level format occasionally detects, mark them and replace with spare blocks at the end of tracks. OS has no idea.
 - » OS detects others after format and stores in unreadable file.
 - » Backup program must have access to this so they don't corrupt when read back.

File System Backups

- » Logical Dump
- » Starts at a directory and recursively dumps all the files that have changed since the last dump.

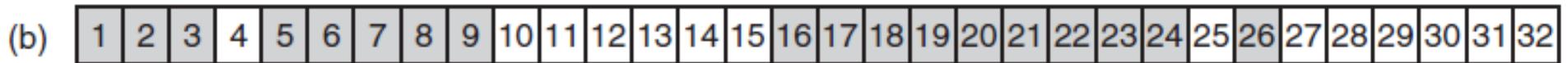
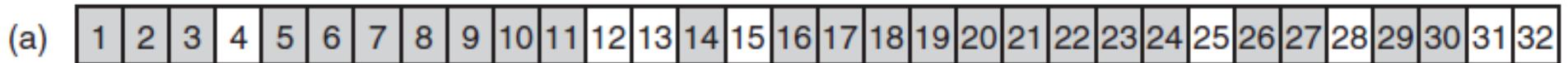
File-System Backups



A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

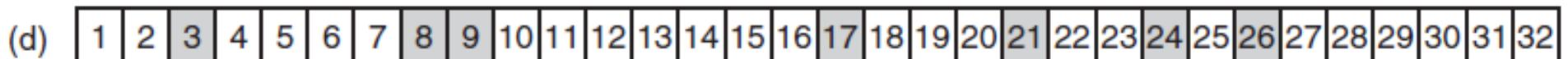
File-System Backups

- » Dump algo maintains a bitmap indexed by i-node number with several bits per inode.
- » bits are set and cleared as also works in 4 phases



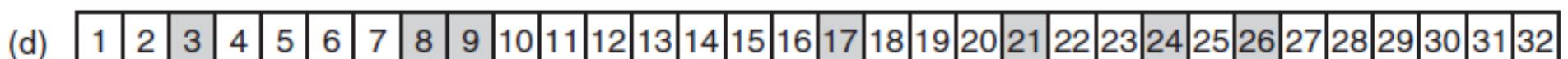
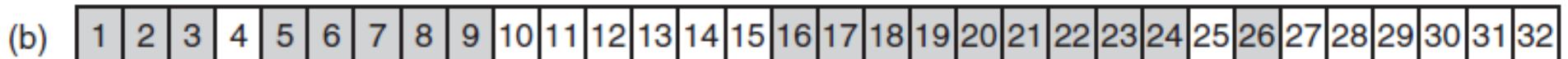
File-System Backups

- » Phase 1: At the starting directory, examine all entries.
 - » For each modified file, mark its inode in the bitmap.
 - » Mark every directory



File-System Backups

- » Phase 2: Recursively walk the tree
- » Unmark directories with no unchanged files or directories



File-System Backups

» Phase 3: Scan inodes in numeric order and dump all the directories marked for dumping, preceded by their metadata

- (a)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
- (b)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
- (c)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----
- (d)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

File-System Backups

- » Phase 4: Dump the files, preceded by their metadata.

(a)	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32		
(b)	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32		
(c)	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32		
(d)	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32		

File-System Backups

- » Restoring.
 - » Create an empty file system on the disk
 - » Restore the most recent full dump
 - » Since directories come first they give a skeleton filesystem for the files.
 - » Then restore the files
 - » Restore all the subsequent incremental dumps

File-System Backups

- » Logical Dumping Issues

- » Since free block list is not a file, it is not dumped and must be reconstructed from scratch after all the dumps have been restored.
- » Links: If a file is linked in two directories, only restore it once.
- » Unix files can have holes.
 - » Holes are not part of the file and should not be dumped
 - » Special files such as named pipes and /dev can not be stored.

File-System Consistency

- » If a system crashes before all modified blocks are written the file system can be left in an inconsistent state.
- » Especially a problem if the blocks not written are the inodes, directory blocks or the free block list.

File-System Consistency

- » Two consistency checks
- » Block
 - » Build two tables each containing a counter for each block, initially set to zero.
 - » First table tracks how many times each block is present in a file.
 - » Second records how often each block is in the free list

File-System Consistency

- » Read all the inodes using a raw device (ignore the file system structure)
- » Build list of all blocks in the corresponding file and increment counter in table one
- » Read free block list and oil out table two.

File-System Consistency

Block number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(a)

Block number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	1	0
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1

(b)

Block number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	1	0
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1

(c)

Block number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	2	1	1	1	0	0	1	1	1	1	0
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1

(d)

Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

File-System Consistency

- » Missing block (b) - No real harm but wasted space.
- » Twice in the free list (c) - Can only occur if the free block list is a list, not a bitmap. Solution:
Rebuild the free list
- » Block in two or more files (d): Bad. If either file is removed the block will be both free and used.
» To fix: Allocate a block. Copy the data and update one of the inodes.

File-System Consistency

- » Directory checks
 - » Use per file table rather than per block
 - » Recurse through the tree
 - » For every inode in every directory increment a counter for the files usage.
 - » Hard links mean multiple counts
 - » Symbolic links don't count

File-System Consistency

- » When done the list is a list indexed by inode telling how many directories contain the file.
- » Compare against the nodes individual link count.
- » Starts at 1 at file creation. Incremented each hard link.

File-System Consistency

- » Two errors can occur
 - » Link count can be too high or too low
 - » If count is too high even removing all the files would result in some inodes not being removed since count should still be greater than 0.
 - » Wasted space but not serious

File-System Consistency

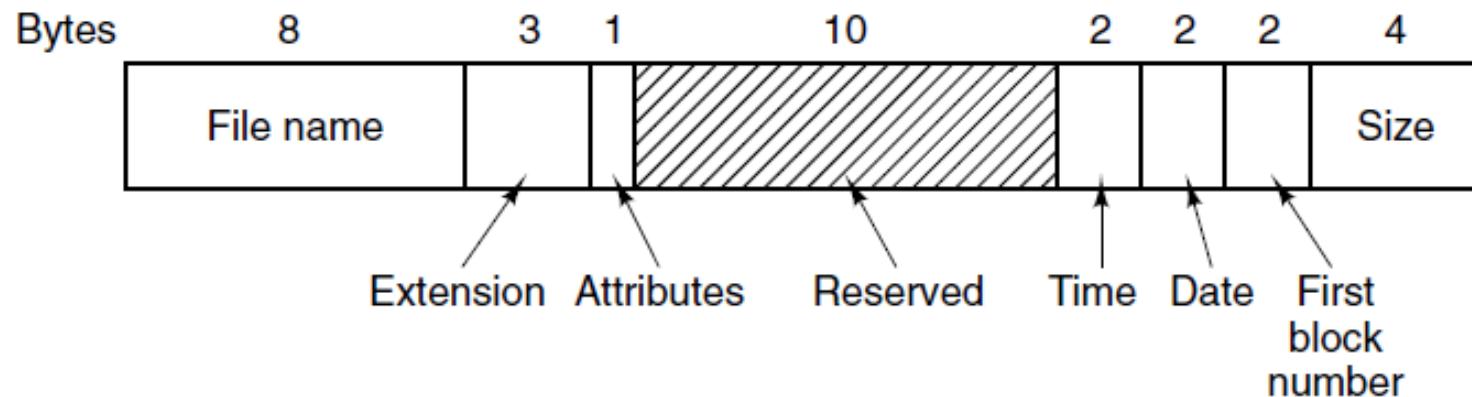
- » If the link count is too low, a file may be deleted from a directory and the inode removed even though it is still referenced.
- » Force the link count in the inode to match the directory entries.

MS-DOS File System

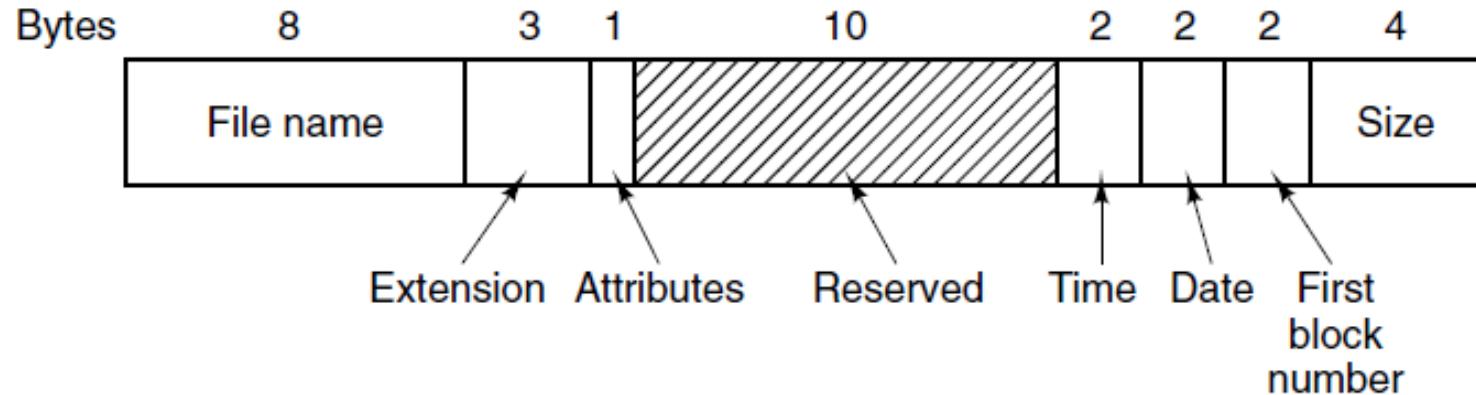
- » First file system for IBM PCs.
- » Main file system up through Windows 98 and Windows ME
 - » Still supported in Windows 2000, Windows XP, Windows Vista.
- » Extension (FAT-32) still widely used in digital cameras, mp3 players, thumb drives

MS-DOS File System

- » More devices use FAT-32 than NTFS
- » MS-DOS directories are variable sized.
- » Each directory entry is fixed 32 bytes.

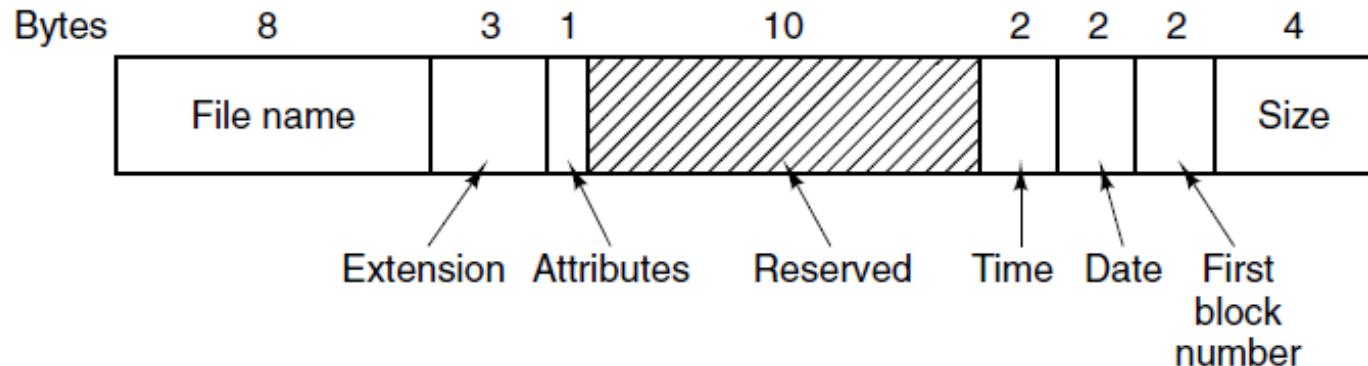


MS-DOS File System



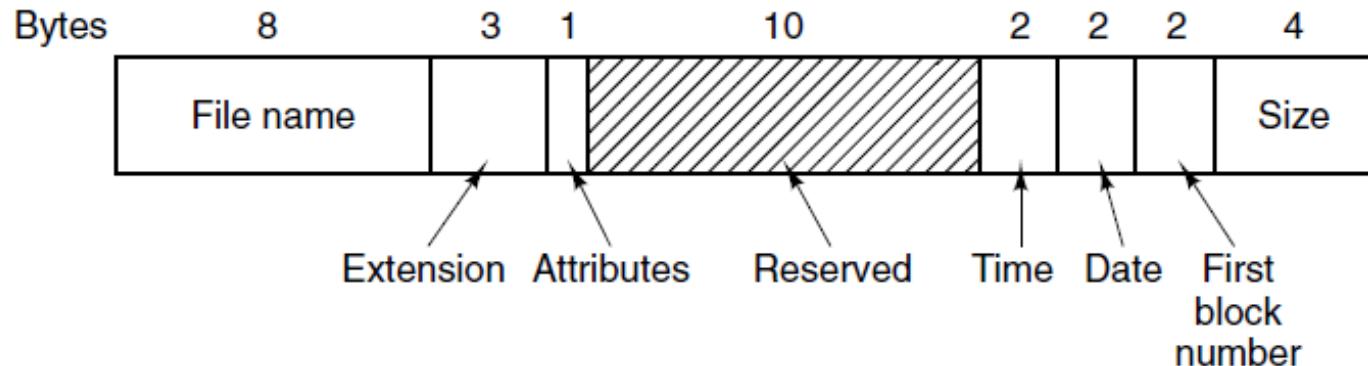
- » File names max size is 8+3, left justified and padded with 0 on the right if smaller.
- » Attributes for read/write/hidden/archive/system.

MS-DOS File System



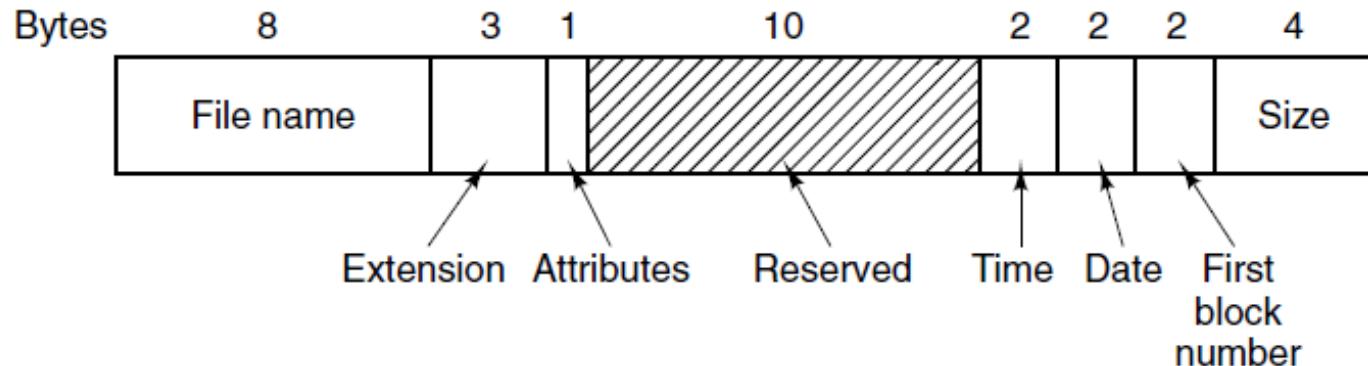
- » Time stores creation time.
- » Accurate to +/- 2 seconds since it's stored in a 2 byte field.
- » Only 65,535 unique values.
- » A day has 86,400 seconds
- » seconds (5bits), minutes (6bits) hours (5bits)

MS-DOS File System



- » Date stores the date in days.
- » Day (5bits) , Month (4bits), Year (7bits)
- » Year expressed in years since 1980.
- » Max year is 2107.
- » Y2108 problem.

MS-DOS File System

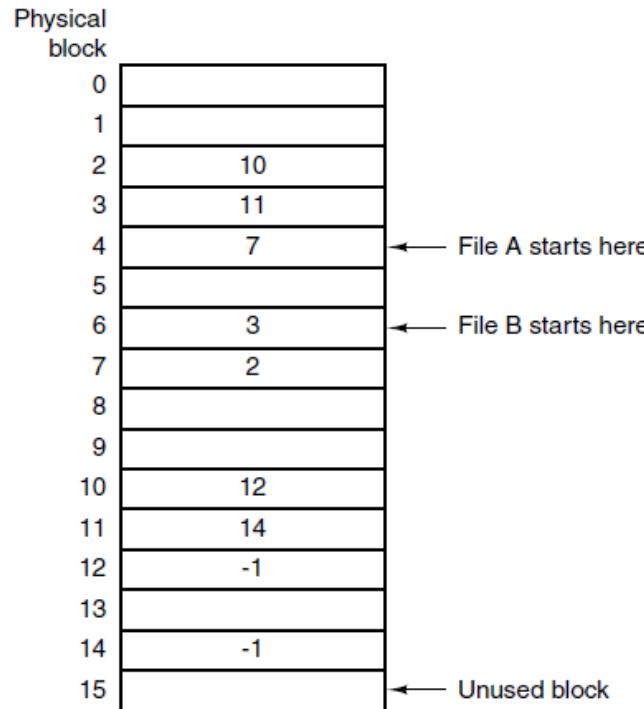


- » Size is stored as 32-bit number. Theoretical max file size of 4 GB
- » Other limitations limit this to 2 GB.
- » 10 reserved bits unused.

MS-DOS File System

- » MS-DOS tracks blocks via File Allocation Table (FAT)
- » The directory entry contains the number of the first block.
- » Used as an index into the 64K FAT in main memory.

MS-DOS File System



Linked list allocation using a file allocation table in main memory.

No bitmap or free list required.

MS-DOS File System

- » Three versions of FAT.
 - » FAT-12, FAT-16, FAT-32
- » FAT-32 misnomer. Only 28 bits are actually used.
- » exFAT - Microsoft introduced for large removable devices.
 - » Apple licensed so Windows to OS X transfers of exFAT can occur
 - » Spec not public

MS-DOS File System

- » FAT disk blocks are multiples of 512 bytes.
 - » Block size called cluster size
- » Can vary per partition

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

MS-DOS File System

- » When hard disks arrived 2MB partitions were too small.
- » MS allowed additional blocks sizes of 1KB, 2KB, and 4KB
 - » Preserved 12 bit FAT table
 - » Allowed disk partitions up to 16 MB
- » 4 disk partitions per drive so could support up to 64 MB disks

MS-DOS File System

- » FAT-16 with 16 bit pointers introduced
 - » Additional block sizes of 8KB, 16KB, 32 KB.
- » FAT-16 table occupied 128KB of main memory.
- » 2GB largest partition.
 - » 64K entries of 32KB each.
- » 8GB disk max. (4 partitions * 2 GB)

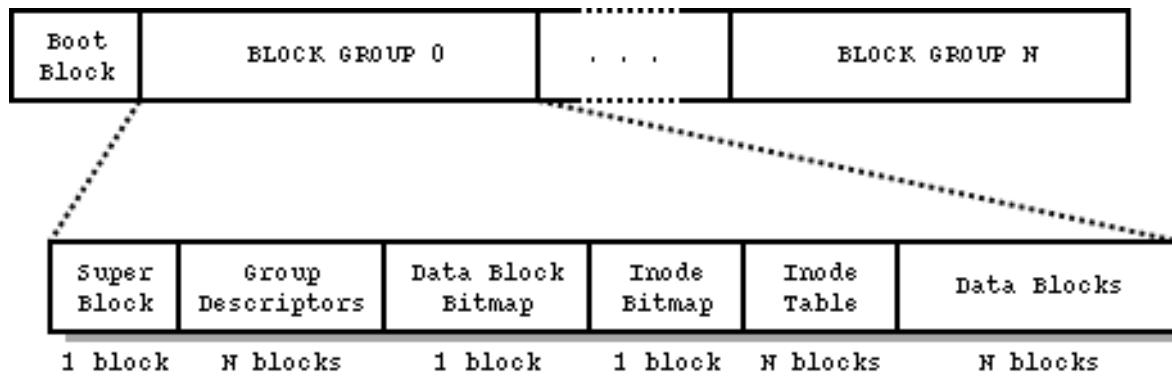
MS-DOS File System

- » Second release of Windows 95 introduced the FAT-32 file system with 28-bit disk addresses.
- » Theoretically the partitions could be $2^{28} \times 2^{15}$ bytes
- » Actual limit is 2TB (2048 GB)
 - » Internally tracks partition sizes with a 32 bit number. $2^9 \times 2^{32} = 2 \text{ TB}$
 - » Max 8GB in single partition
 - » Can use 4KB blocks for large partitions.

MS-DOS File System

- » Second release of Windows 95 introduced the FAT-32 file system with 28-bit disk addresses.
- » Theoretically the partitions could be $2^{28} \times 2^{15}$ bytes
- » Actual limit is 2TB (2048 GB)
 - » Internally tracks partition sizes with a 32 bit number. $2^9 \times 2^{32} = 2 \text{ TB}$
 - » Max 8GB in single partition
 - » Can use 4KB blocks for large partitions.

Structure of an Ext2 Filesystem



- The first 1024 bytes of the disk, the "boot block", are reserved for the partition boot sectors and are unused by the Ext2 filesystem. The rest of the partition is split into block groups,

The SuperBlock

It contains information such as the total number of blocks on disk, the size of a block (usually 1024 bytes), the number of free blocks, etc.

Part of this structure:

```
struct ext2_super_block {
    __u32 s_inodes_count;      /* Inodes count */
    __u32 s_blocks_count;     /* Blocks count */
    ...
    __u32 s_free_blocks_count; /* Free blocks count */
    __u32 s_free_inodes_count; /* Free inodes count */
    __u32 s_first_data_block; /* First Data Block */
    __u32 s_log_block_size;   /* Block size */
    ...
    __u32 s_blocks_per_group; /* # Blocks per group */
    ...
    __u16 s_magic;           /* Magic signature */
    ...
}
```

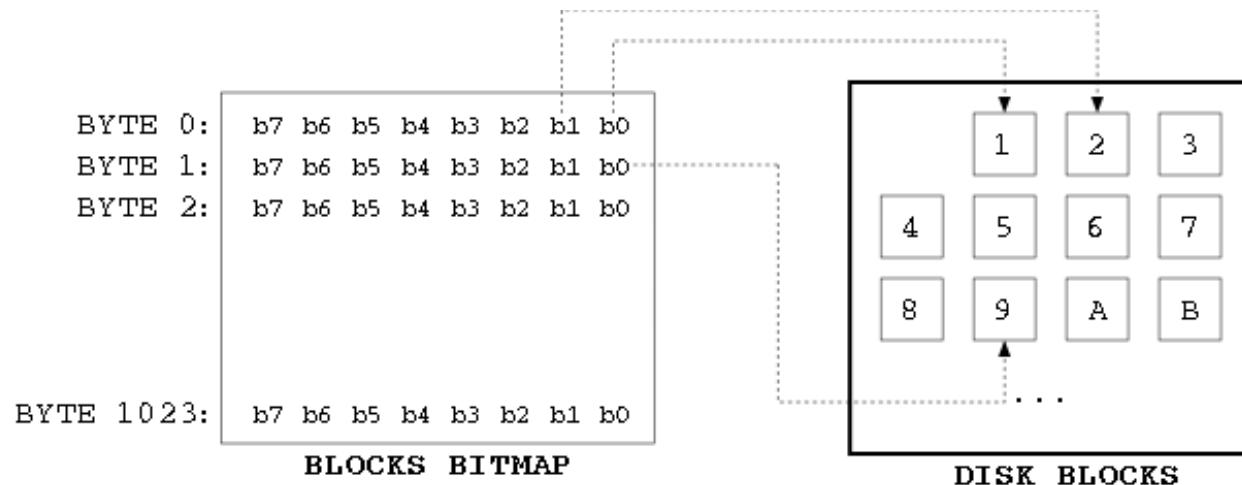
Group Descriptors

- In the blocks immediately following the super-block reside the list of block-group descriptors. This list contains a descriptor for each block group on the disk. In the case of a floppy, there is only one block group and therefore one group descriptor.

```
struct ext2_group_desc
{
    __u32 bg_block_bitmap;      /* Blocks bitmap block */
    __u32 bg_inode_bitmap;     /* Inodes bitmap block */
    __u32 bg_inode_table;      /* Inodes table block */
    __u16 bg_free_blocks_count; /* Free blocks count */
    __u16 bg_free_inodes_count; /* Free inodes count */
    __u16 bg_used_dirs_count;  /* Directories count */
    __u16 bg_pad;
    __u32 bg_reserved[3];
};
```

The blocks and inodes bitmaps

- Each bit represents a specific block (blocks bitmap) or inode (inode bitmap) in the block group. A bit value of 0 indicates that the block/inode is free, while a value of 1 indicates that the block/inode is being used. A bitmap always refers to the block-group it belongs to, and its size must fit in one block.



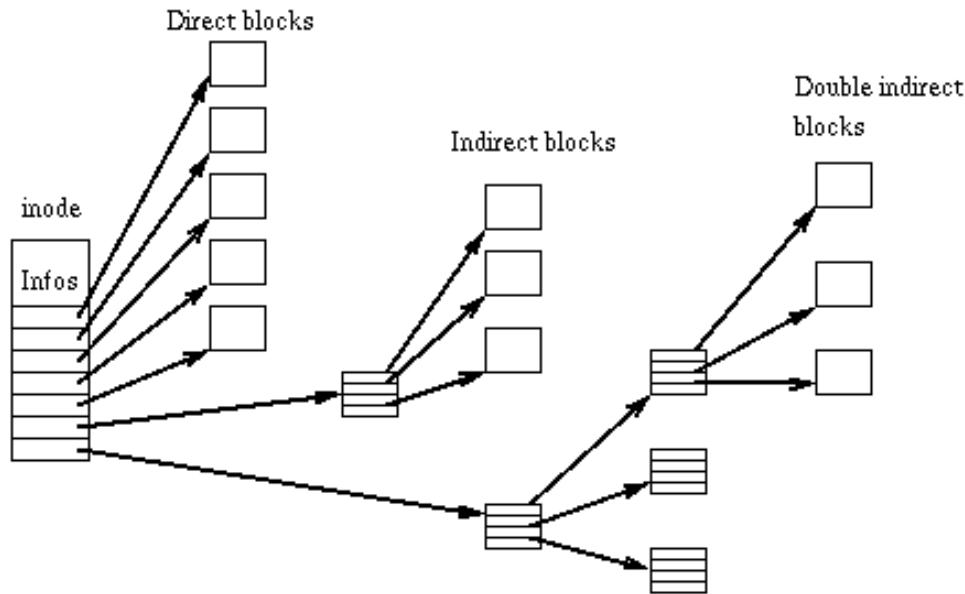
The inode table

- The inode table consists of a series of consecutive blocks, each of which contains a predefined number of inodes.
- The inode table contains everything the operating system needs to know about a file, including the type of file, permissions, owner, and, most important, where its data blocks are located on disk. It is no surprise therefore that this table needs to be accessed very frequently and its read access time should be minimized as much as possible.

The inode table

```
struct ext2_inode {  
    __u16  i_mode;          /* File type and access rights */  
    __u16  i_uid;           /* Low 16 bits of Owner Uid */  
    __u32  i_size;          /* Size in bytes */  
    __u32  i_atime;         /* Access time */  
    __u32  i_ctime;         /* Creation time */  
    __u32  i_mtime;         /* Modification time */  
    __u32  i_dtime;         /* Deletion Time */  
    __u16  i_gid;           /* Low 16 bits of Group Id */  
    __u16  i_links_count;   /* Links count */  
    __u32  i_blocks;        /* Blocks count */  
    __u32  i_flags;         /* File flags */  
    ...  
    __u32  i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */  
    ...  
};
```

The inode structure



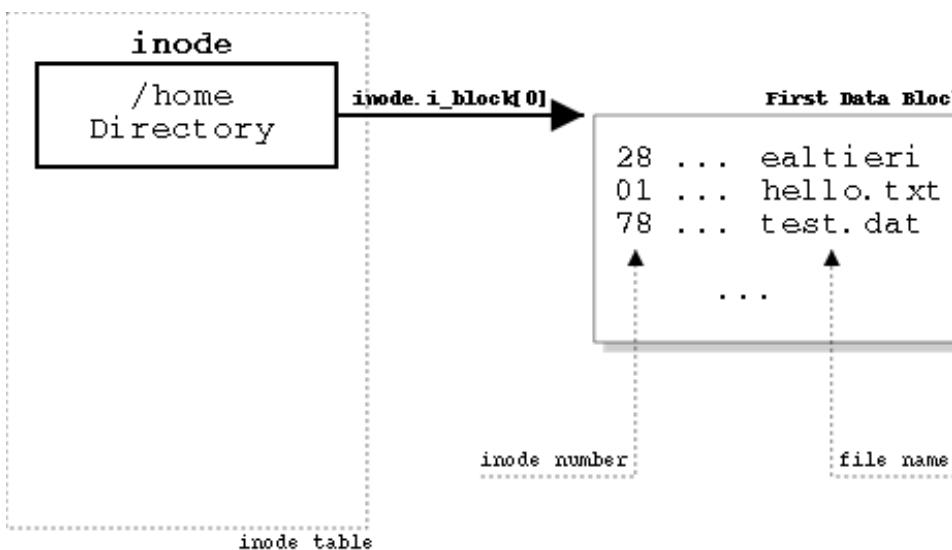
Quote from the Linux kernel documentation for ext2:

"There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect block."

So, there is a structure in ext2 that has 15 pointers. Pointers 1 to 12 point to direct blocks, pointer 13 points to an indirect block, pointer 14 points to a doubly indirect block, and pointer 15 points to a trebly indirect block.

Directory entries in the inode table

- In the case of directory entries, the data blocks pointed by `i_block[]` contain a list of the files in the directory and their respective inode numbers.
- Finding: /home/ealtieri/hello.txt



```
struct ext2_dir_entry_2 {  
    __u32    inode;          /* Inode number */  
    __u16    rec_len;        /* Directory entry length */  
    __u8     name_len;       /* Name length */  
    __u8     file_type;  
    char   name[EXT2_NAME_LEN]; /* File name */  
};
```

CD-ROM File System

- Simple since designed for write-once media
- No provision for tracking free blocks since blocks can't be freed or added to once manufactured.
- CD-R (CD Recordable) introduced the possibility to add files.
 - Appended to the end of the file system.
 - All free space is one contiguous chunk at the end of the file system.

ISO 9660 File System

- Adopted as a standard in 1998.
- Virtually every CD on the market is compatible.
- Goal was make every CD ROM readable on every computer independent of byte ordering and OS used.
 - Caused limitations to be placed on the system.

ISO 9660 File System

- CD-ROM do not have concentric cylinders.
 - Single continuous spiral containing the bits in a linear sequence.
 - Logical seeks across the spiral are still possible..
 - Bits along the spiral are divided into logical blocks (also called logical sectors) of 2352 bytes.
 - Some are preamble, ECC and other overhead.
 - Payload of each block is 2048 bytes

ISO 9660 File System

- When used for music CDs have leadins, leadoffs and inter track gaps. Not used in data formats.
- Block position sometimes quoted in minutes and seconds
 - 1 sec = 75 blocks.
- Each CD begins with 16 blocks that are undermined by the standard and can be used to hold a bootstrap program.
-

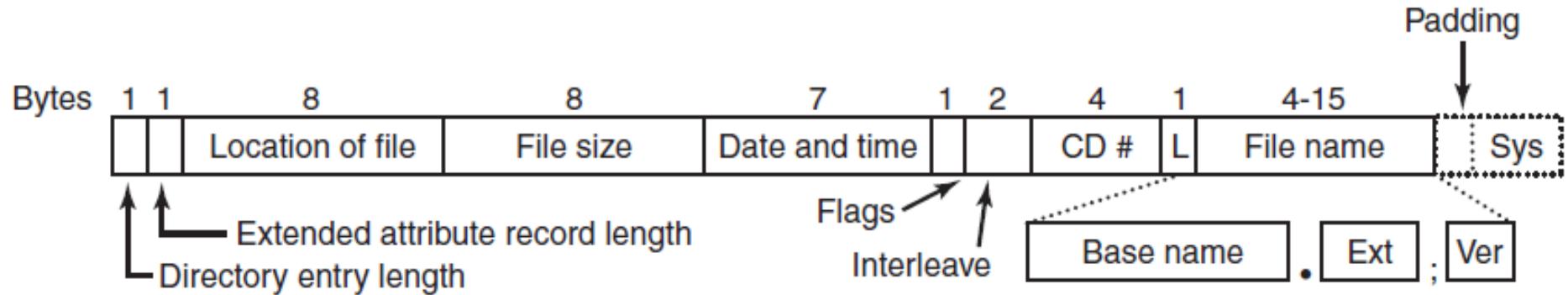
ISO 9660 File System

- Following is the primary volume descriptor
 - System Identifier (32 bytes)
 - Volume Identifier (32 bytes)
 - Publisher Identifier (128 bytes)
 - Data Preparer Identifier (128 bytes)
- All must be uppercase and only a few punctuation supported.

ISO 9660 File System

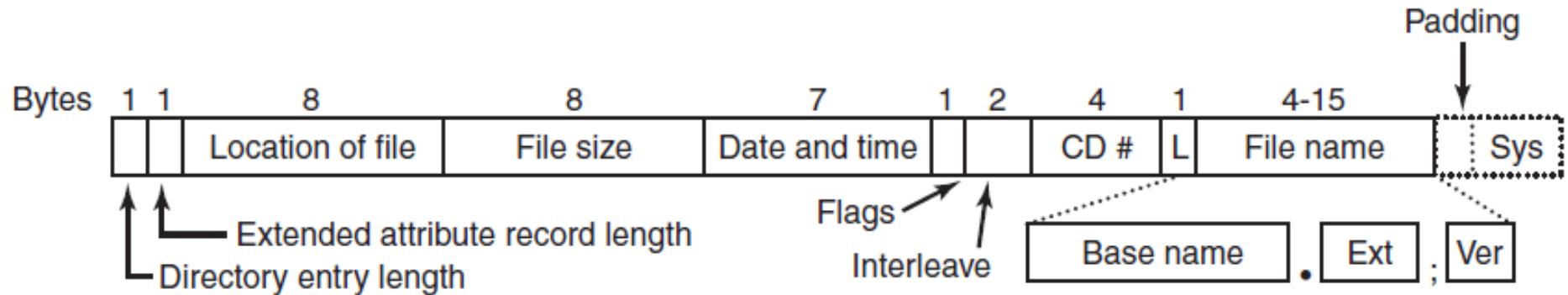
- Also contains the name of three files which contain:
 - Abstract
 - Copyright Notice
 - Bibliographic Information
- Also holds block size (normally 2048 but 4KB, 8KB and higher powers of 2 can be supported.)
- Contains directory entry for the root directory.

ISO 9660 File System



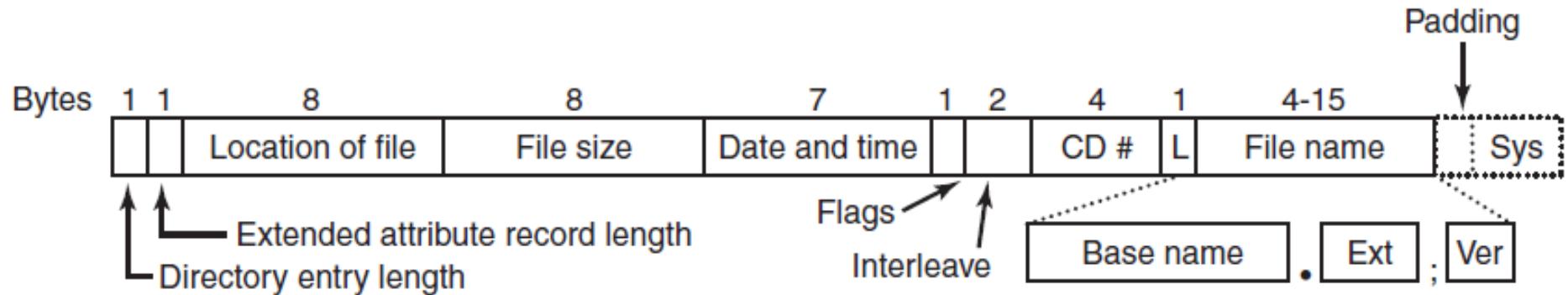
- Variable number of entries
 - Last contains marker signifying end
 - Each entry encoded in ASCII or binary.
 - Binary encoded twice. Little endian and big endian

ISO 9660 File System



- Directory entries may have extended attributes
- Date and time stored in bytes each for day, month, year
 - Years started at 1900. Y2156 problem.

ISO 9660 File System



- No limit to number of entries in a directory.
- There is a limit of eight to nesting.
 - Arbitrarily set to make some implementations easier.

ISO 9660 File System

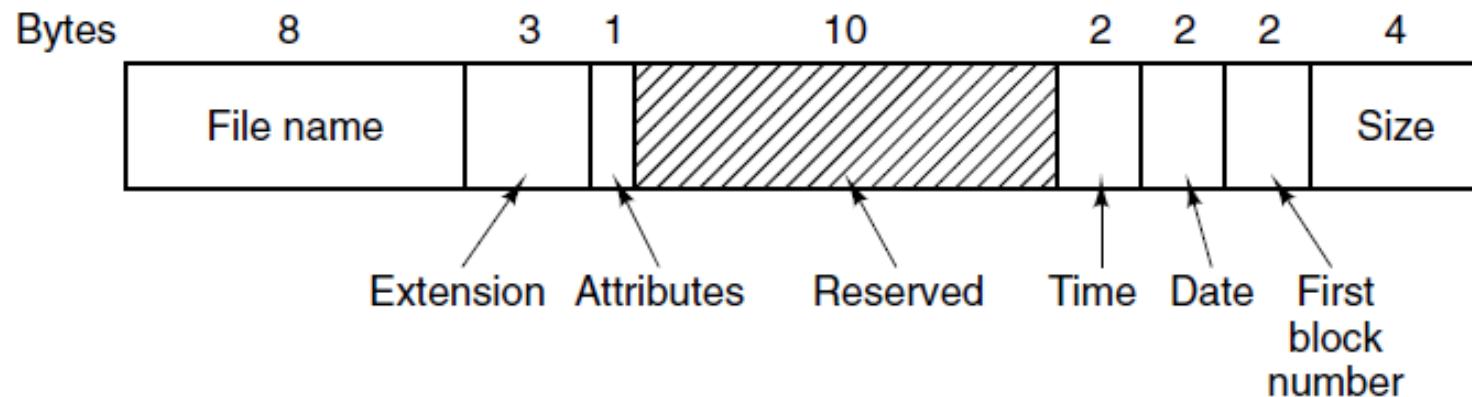
- Three levels
 - Level 1:
 - 8+3 file names
 - All files contiguous
 - 8 character file names
 - Level 2:
 - Up to 31 character file and directory names
 - Level 3:
 - Same limits as level 2 but files do not have to be contiguous.

MS-DOS File System

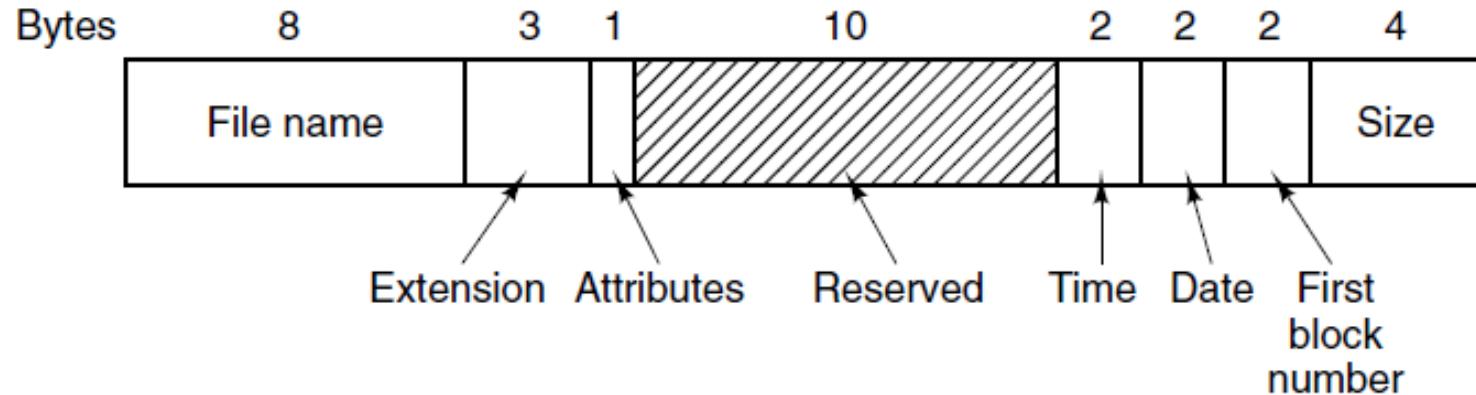
- » First file system for IBM PCs.
- » Main file system up through Windows 98 and Windows ME
 - » Still supported in Windows 2000, Windows XP, Windows Vista.
- » Extension (FAT-32) still widely used in digital cameras, mp3 players, thumb drives

MS-DOS File System

- » More devices use FAT-32 than NTFS
- » MS-DOS directories are variable sized.
- » Each directory entry is fixed 32 bytes.

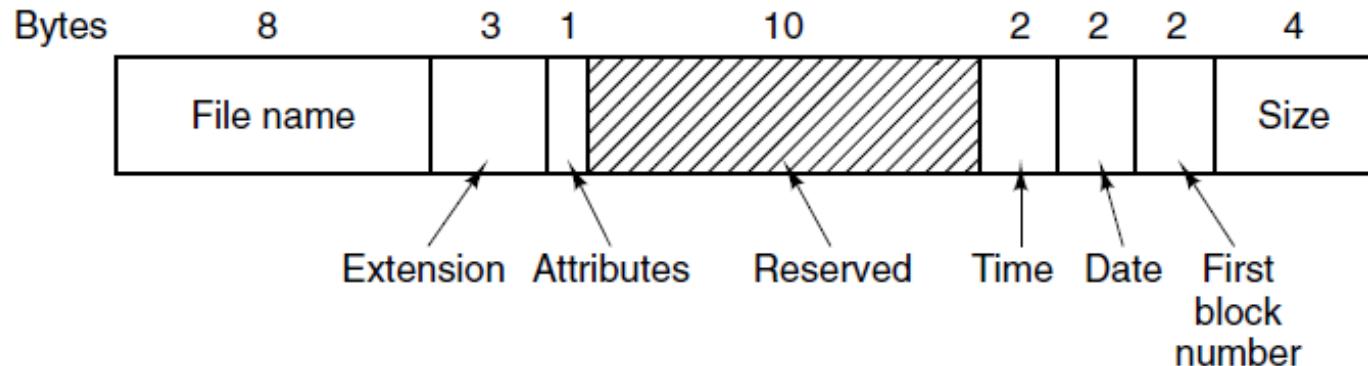


MS-DOS File System



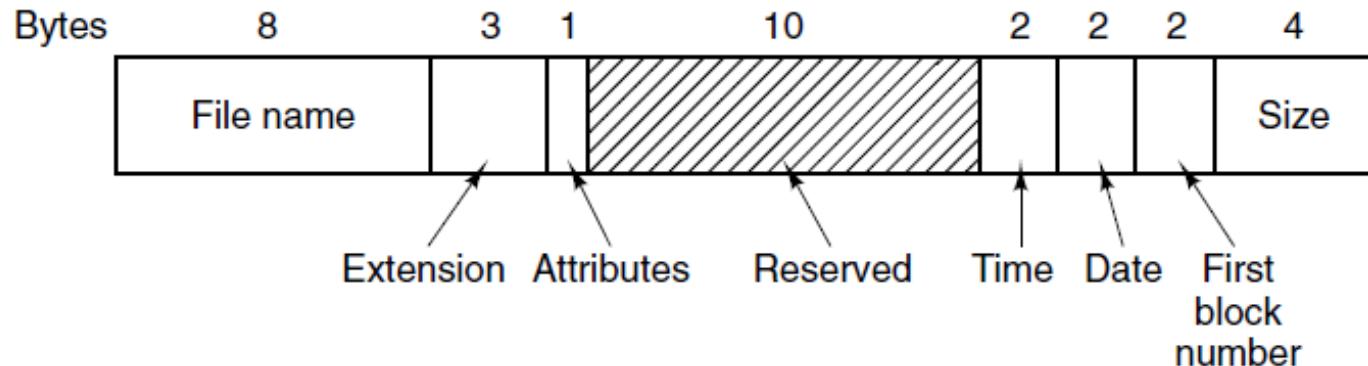
- » File names max size is 8+3, left justified and padded with 0 on the right if smaller.
- » Attributes for read/write/hidden/archive/system.

MS-DOS File System



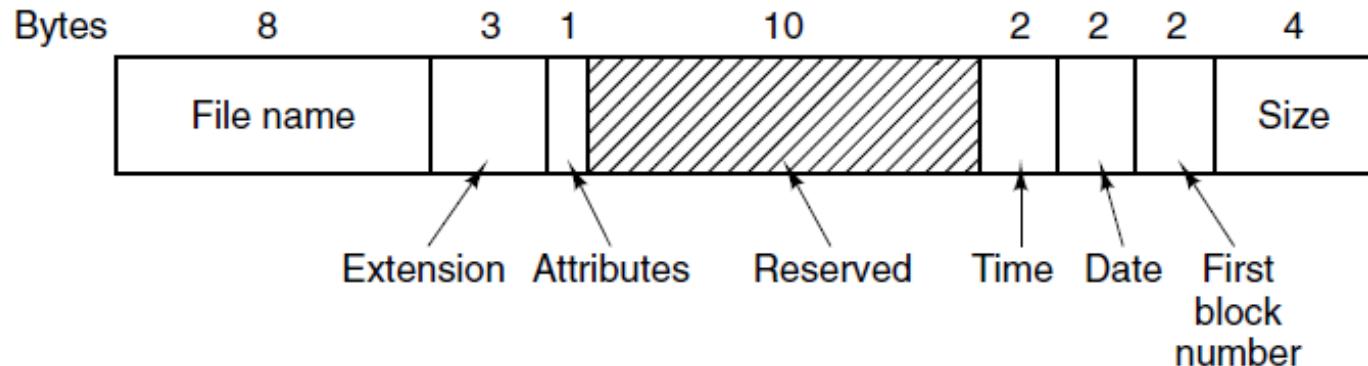
- » Time stores creation time.
- » Accurate to +/- 2 seconds since it's stored in a 2 byte field.
- » Only 65,535 unique values.
- » A day has 86,400 seconds
- » seconds (5bits), minutes (6bits) hours (5bits)

MS-DOS File System



- » Date stores the date in days.
- » Day (5bits) , Month (4bits), Year (7bits)
- » Year expressed in years since 1980.
- » Max year is 2107.
- » Y2108 problem.

MS-DOS File System

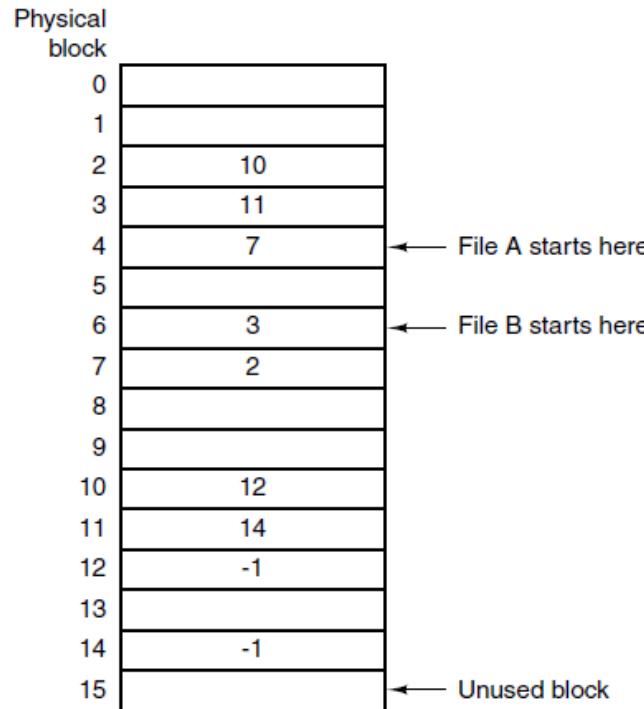


- » Size is stored as 32-bit number. Theoretical max file size of 4 GB
- » Other limitations limit this to 2 GB.
- » 10 reserved bits unused.

MS-DOS File System

- » MS-DOS tracks blocks via File Allocation Table (FAT)
- » The directory entry contains the number of the first block.
- » Used as an index into the 64K FAT in main memory.

MS-DOS File System



Linked list allocation using a file allocation table in main memory.

No bitmap or free list required.

MS-DOS File System

- » Three versions of FAT.
 - » FAT-12, FAT-16, FAT-32
- » FAT-32 misnomer. Only 28 bits are actually used.
- » exFAT - Microsoft introduced for large removable devices.
 - » Apple licensed so Windows to OS X transfers of exFAT can occur
 - » Spec not public

MS-DOS File System

- » FAT disk blocks are multiples of 512 bytes.
 - » Block size called cluster size
- » Can vary per partition

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

MS-DOS File System

- » When hard disks arrived 2MB partitions were too small.
- » MS allowed additional blocks sizes of 1KB, 2KB, and 4KB
 - » Preserved 12 bit FAT table
 - » Allowed disk partitions up to 16 MB
- » 4 disk partitions per drive so could support up to 64 MB disks

MS-DOS File System

- » FAT-16 with 16 bit pointers introduced
 - » Additional block sizes of 8KB, 16KB, 32 KB.
- » FAT-16 table occupied 128KB of main memory.
- » 2GB largest partition.
 - » 64K entries of 32KB each.
- » 8GB disk max. (4 partitions * 2 GB)

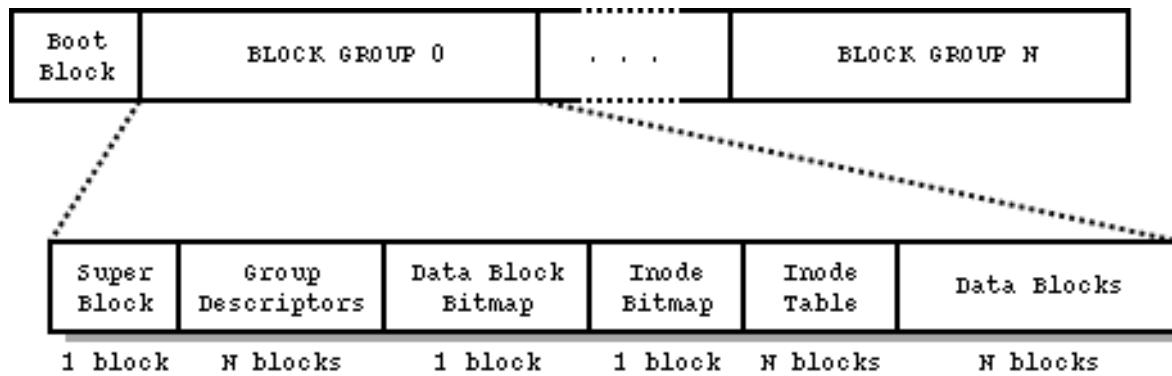
MS-DOS File System

- » Second release of Windows 95 introduced the FAT-32 file system with 28-bit disk addresses.
- » Theoretically the partitions could be $2^{28} \times 2^{15}$ bytes
- » Actual limit is 2TB (2048 GB)
 - » Internally tracks partition sizes with a 32 bit number. $2^9 \times 2^{32} = 2 \text{ TB}$
 - » Max 8GB in single partition
 - » Can use 4KB blocks for large partitions.

MS-DOS File System

- » Second release of Windows 95 introduced the FAT-32 file system with 28-bit disk addresses.
- » Theoretically the partitions could be $2^{28} \times 2^{15}$ bytes
- » Actual limit is 2TB (2048 GB)
 - » Internally tracks partition sizes with a 32 bit number. $2^9 \times 2^{32} = 2 \text{ TB}$
 - » Max 8GB in single partition
 - » Can use 4KB blocks for large partitions.

Structure of an Ext2 Filesystem



- The first 1024 bytes of the disk, the "boot block", are reserved for the partition boot sectors and are unused by the Ext2 filesystem. The rest of the partition is split into block groups,

The SuperBlock

It contains information such as the total number of blocks on disk, the size of a block (usually 1024 bytes), the number of free blocks, etc.

Part of this structure:

```
struct ext2_super_block {
    __u32 s_inodes_count;      /* Inodes count */
    __u32 s_blocks_count;     /* Blocks count */
    ...
    __u32 s_free_blocks_count; /* Free blocks count */
    __u32 s_free_inodes_count; /* Free inodes count */
    __u32 s_first_data_block; /* First Data Block */
    __u32 s_log_block_size;   /* Block size */
    ...
    __u32 s_blocks_per_group; /* # Blocks per group */
    ...
    __u16 s_magic;           /* Magic signature */
    ...
}
```

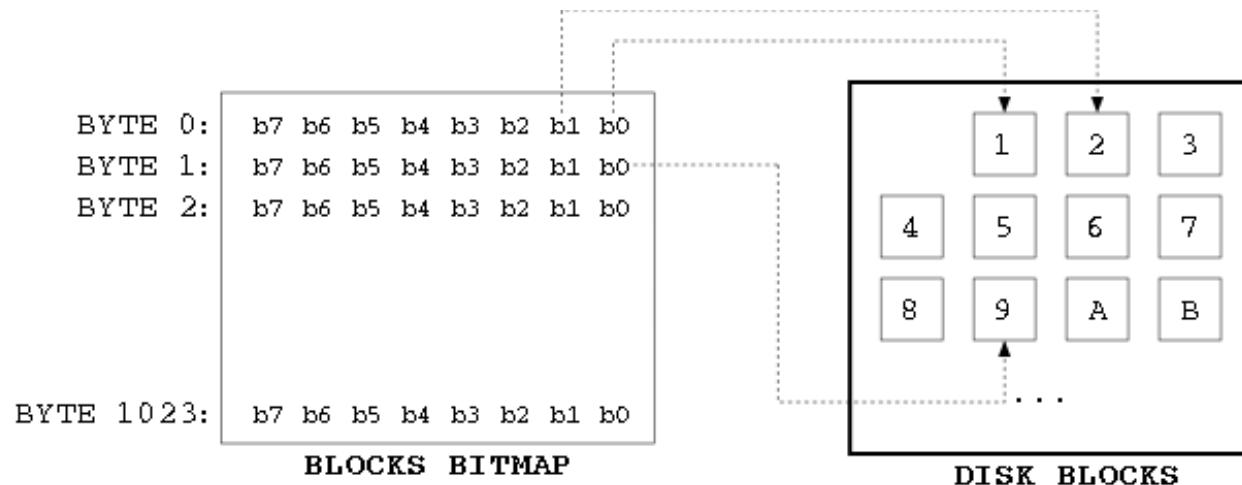
Group Descriptors

- In the blocks immediately following the super-block reside the list of block-group descriptors. This list contains a descriptor for each block group on the disk. In the case of a floppy, there is only one block group and therefore one group descriptor.

```
struct ext2_group_desc
{
    __u32 bg_block_bitmap;      /* Blocks bitmap block */
    __u32 bg_inode_bitmap;     /* Inodes bitmap block */
    __u32 bg_inode_table;      /* Inodes table block */
    __u16 bg_free_blocks_count; /* Free blocks count */
    __u16 bg_free_inodes_count; /* Free inodes count */
    __u16 bg_used_dirs_count;  /* Directories count */
    __u16 bg_pad;
    __u32 bg_reserved[3];
};
```

The blocks and inodes bitmaps

- Each bit represents a specific block (blocks bitmap) or inode (inode bitmap) in the block group. A bit value of 0 indicates that the block/inode is free, while a value of 1 indicates that the block/inode is being used. A bitmap always refers to the block-group it belongs to, and its size must fit in one block.



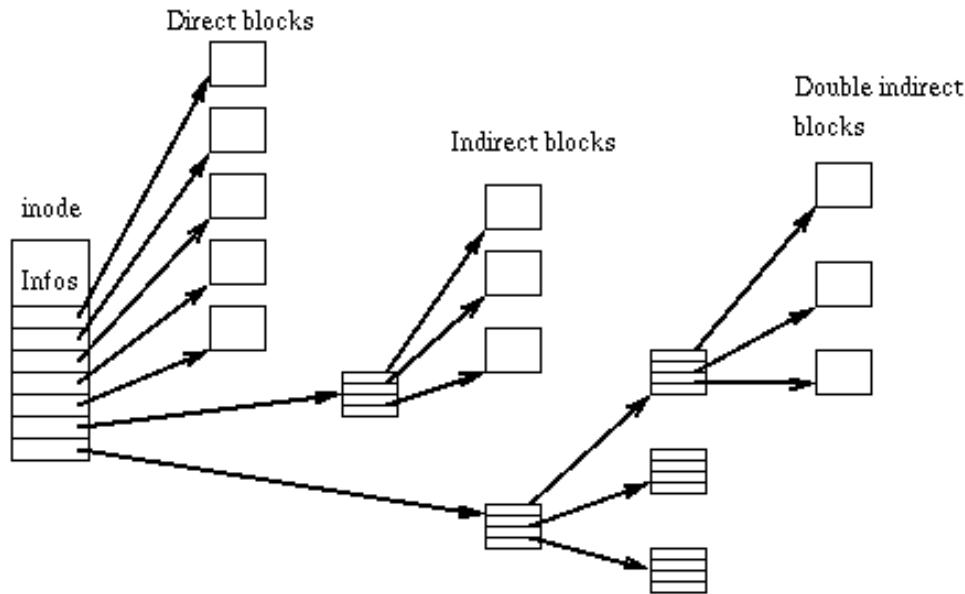
The inode table

- The inode table consists of a series of consecutive blocks, each of which contains a predefined number of inodes.
- The inode table contains everything the operating system needs to know about a file, including the type of file, permissions, owner, and, most important, where its data blocks are located on disk. It is no surprise therefore that this table needs to be accessed very frequently and its read access time should be minimized as much as possible.

The inode table

```
struct ext2_inode {  
    __u16  i_mode;          /* File type and access rights */  
    __u16  i_uid;           /* Low 16 bits of Owner Uid */  
    __u32  i_size;          /* Size in bytes */  
    __u32  i_atime;         /* Access time */  
    __u32  i_ctime;         /* Creation time */  
    __u32  i_mtime;         /* Modification time */  
    __u32  i_dtime;         /* Deletion Time */  
    __u16  i_gid;           /* Low 16 bits of Group Id */  
    __u16  i_links_count;   /* Links count */  
    __u32  i_blocks;        /* Blocks count */  
    __u32  i_flags;         /* File flags */  
    ...  
    __u32  i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */  
    ...  
};
```

The inode structure



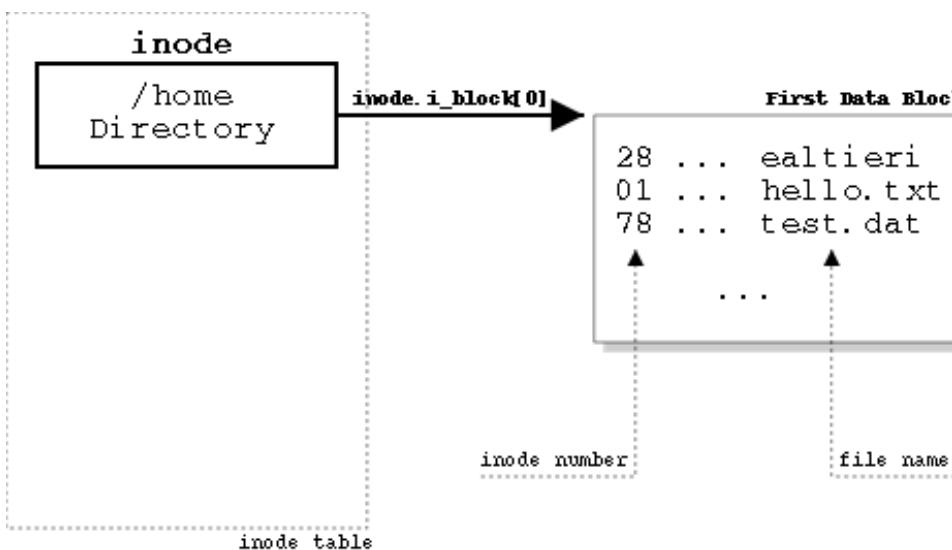
Quote from the Linux kernel documentation for ext2:

"There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect block."

So, there is a structure in ext2 that has 15 pointers. Pointers 1 to 12 point to direct blocks, pointer 13 points to an indirect block, pointer 14 points to a doubly indirect block, and pointer 15 points to a trebly indirect block.

Directory entries in the inode table

- In the case of directory entries, the data blocks pointed by `i_block[]` contain a list of the files in the directory and their respective inode numbers.
- Finding: /home/ealtieri/hello.txt



```
struct ext2_dir_entry_2 {  
    __u32    inode;          /* Inode number */  
    __u16    rec_len;        /* Directory entry length */  
    __u8     name_len;       /* Name length */  
    __u8     file_type;  
    char   name[EXT2_NAME_LEN]; /* File name */  
};
```

CD-ROM File System

- Simple since designed for write-once media
- No provision for tracking free blocks since blocks can't be freed or added to once manufactured.
- CD-R (CD Recordable) introduced the possibility to add files.
 - Appended to the end of the file system.
 - All free space is one contiguous chunk at the end of the file system.

ISO 9660 File System

- Adopted as a standard in 1998.
- Virtually every CD on the market is compatible.
- Goal was make every CD ROM readable on every computer independent of byte ordering and OS used.
 - Caused limitations to be placed on the system.

ISO 9660 File System

- CD-ROM do not have concentric cylinders.
 - Single continuous spiral containing the bits in a linear sequence.
 - Logical seeks across the spiral are still possible..
 - Bits along the spiral are divided into logical blocks (also called logical sectors) of 2352 bytes.
 - Some are preamble, ECC and other overhead.
 - Payload of each block is 2048 bytes

ISO 9660 File System

- When used for music CDs have leadins, leadoffs and inter track gaps. Not used in data formats.
- Block position sometimes quoted in minutes and seconds
 - 1 sec = 75 blocks.
- Each CD begins with 16 blocks that are undermined by the standard and can be used to hold a bootstrap program.
-

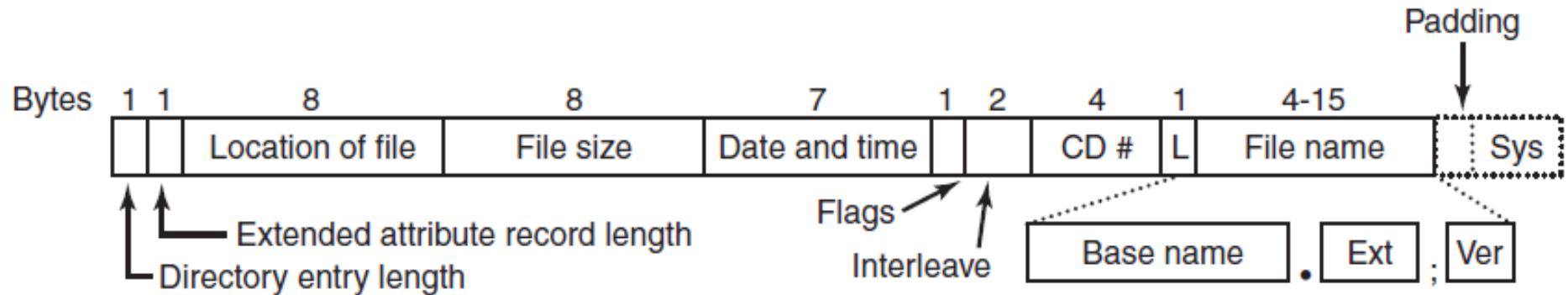
ISO 9660 File System

- Following is the primary volume descriptor
 - System Identifier (32 bytes)
 - Volume Identifier (32 bytes)
 - Publisher Identifier (128 bytes)
 - Data Preparer Identifier (128 bytes)
- All must be uppercase and only a few punctuation supported.

ISO 9660 File System

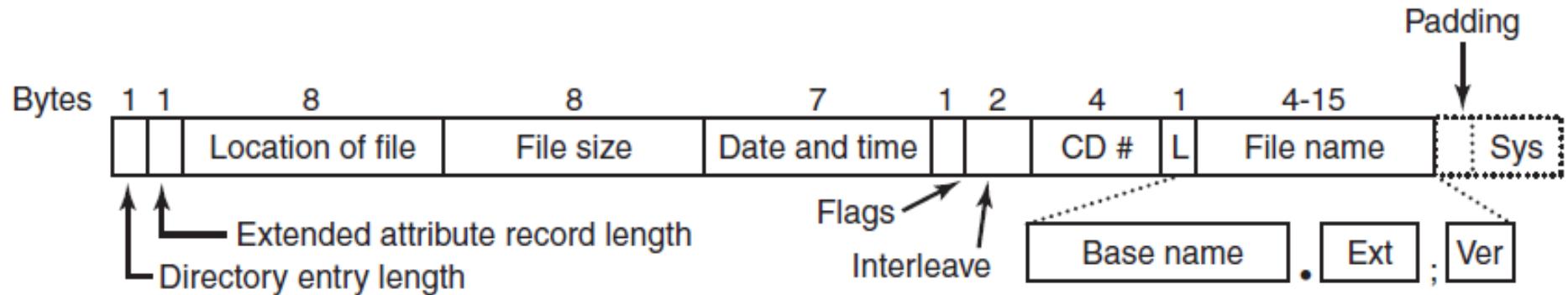
- Also contains the name of three files which contain:
 - Abstract
 - Copyright Notice
 - Bibliographic Information
- Also holds block size (normally 2048 but 4KB, 8KB and higher powers of 2 can be supported.)
- Contains directory entry for the root directory.

ISO 9660 File System



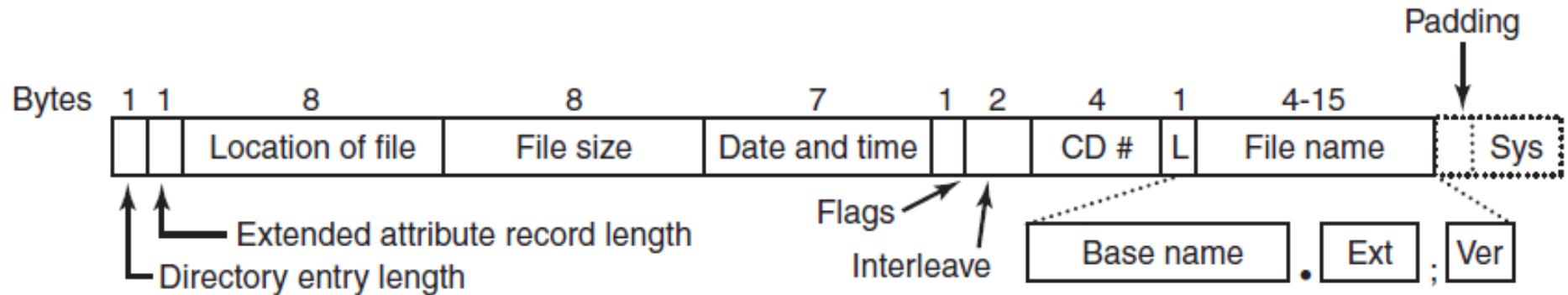
- Variable number of entries
 - Last contains marker signifying end
 - Each entry encoded in ASCII or binary.
 - Binary encoded twice. Little endian and big endian

ISO 9660 File System



- Directory entries may have extended attributes
- Date and time stored in byte each for day, month, year
 - Years started at 1900. Y2156 problem.

ISO 9660 File System



- No limit to number of entries in a directory.
- There is a limit of eight to nesting.
 - Arbitrarily set to make some implementations easier.

ISO 9660 File System

- Three levels
 - Level 1:
 - 8+3 file names
 - All files contiguous
 - 8 character file names
 - Level 2:
 - Up to 31 character file and directory names
 - Level 3:
 - Same limits as level 2 but files do not have to be contiguous.

Deadlock Problem

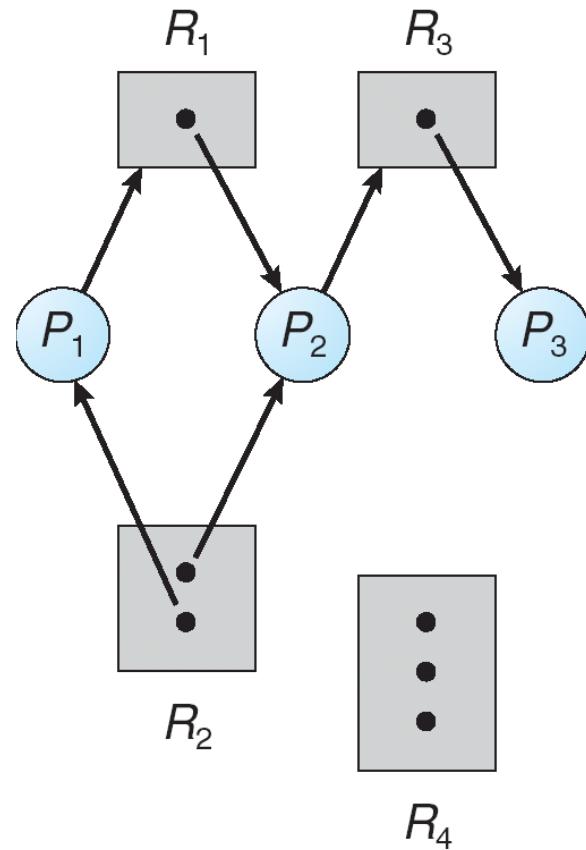
- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set
- Example
 - System has 2 disk drives
 - P1 and P2 each hold one disk drive and each needs another one
- Example
 - semaphores A and B, initialized to 1

P0	P1
wait (A);	wait(B)
wait (B);	wait(A)

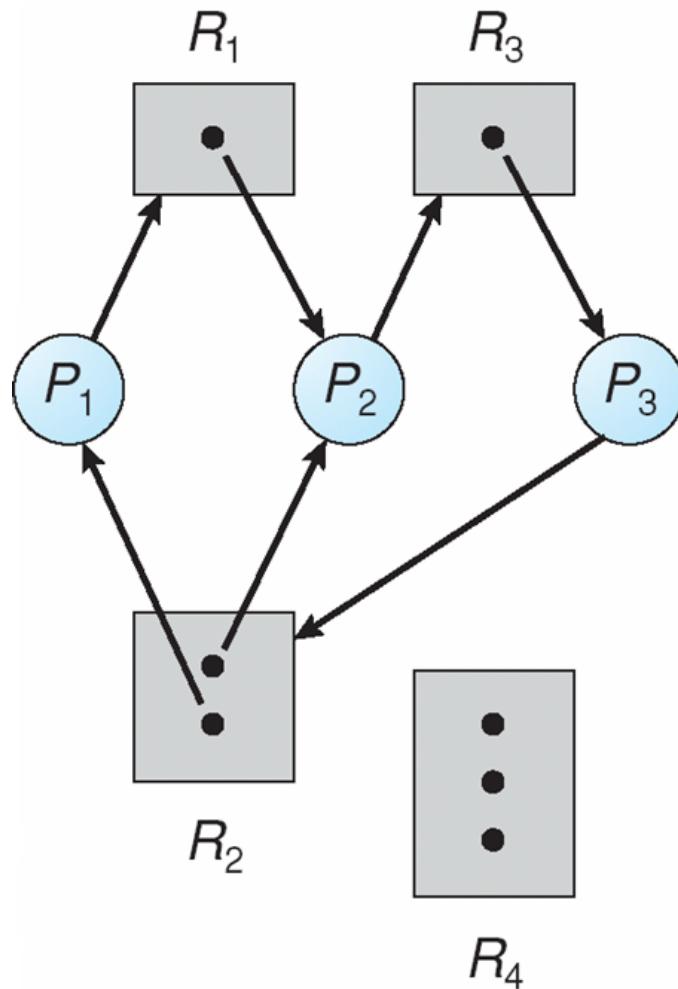
Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously.
 1. Mutual exclusion: only one process at a time can use a resource
 2. Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes
 3. No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task
 4. Circular wait: there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

Example of a Resource Allocation Graph



Resource Allocation Graph with a Deadlock



Basic Rules

- If graph contains no cycles \Rightarrow no deadlock
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then deadlock
 - if several instances per resource type, possibility of deadlock

Handling Deadlocks

- Ensure that the system will **never** enter a deadlock state
- Allow the system to enter a deadlock state and then recover
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Race Condition

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

Access Methods

- Sequential Access

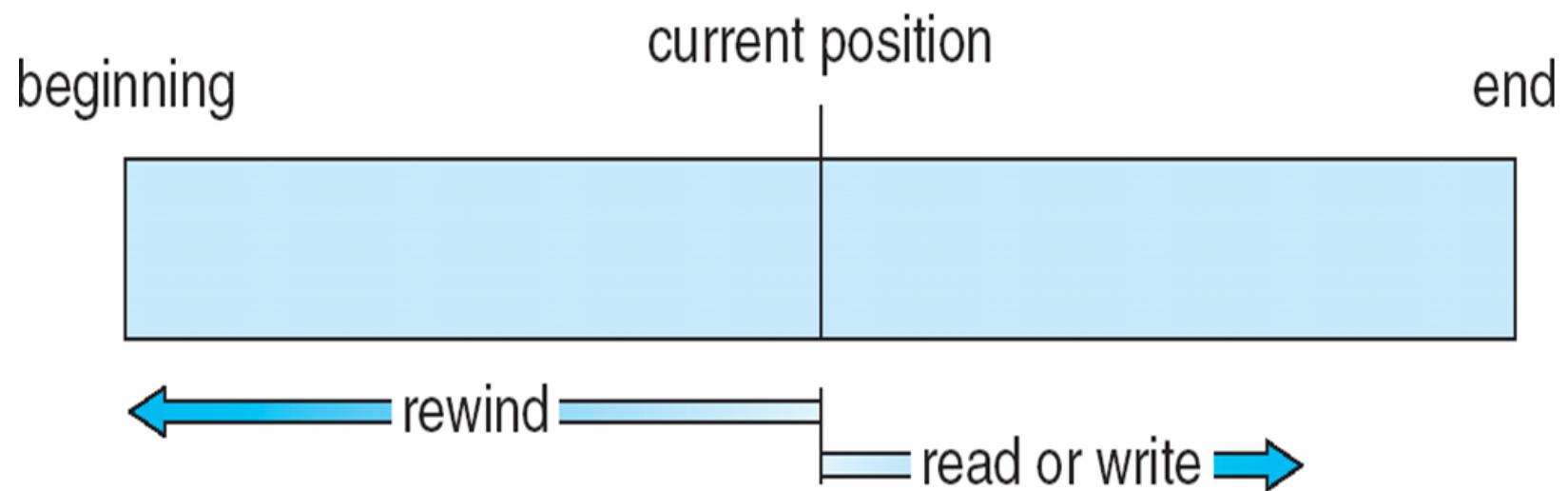
- read next
 - write next
 - reset
 - no read after last write
(rewrite)

- Direct Access

- read n
 - write n
 - position to n
 - read next
 - write next
 - rewrite n

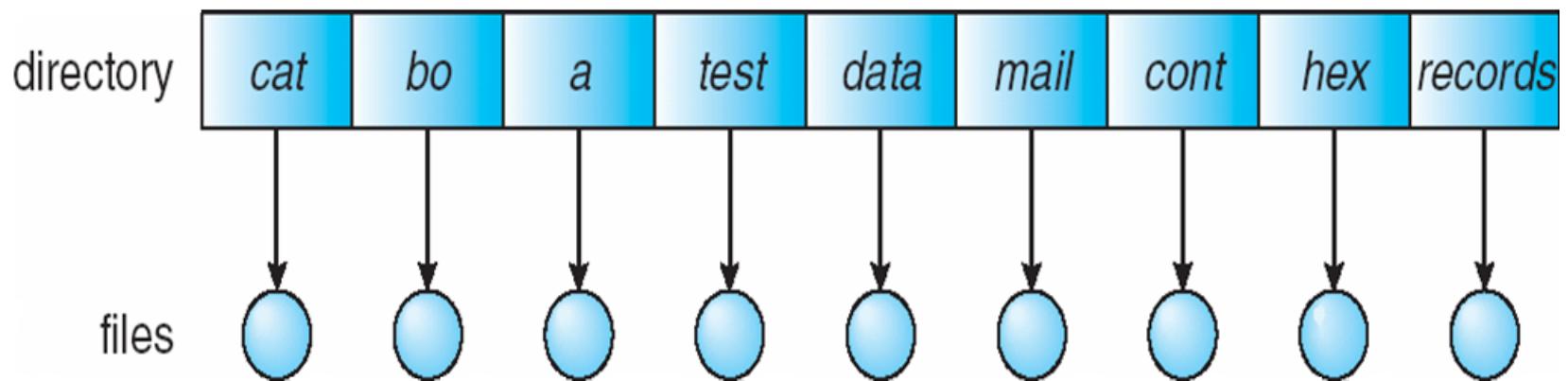
n = relative block number

Sequential-access File



Single-Level Directory

- A single directory for all users

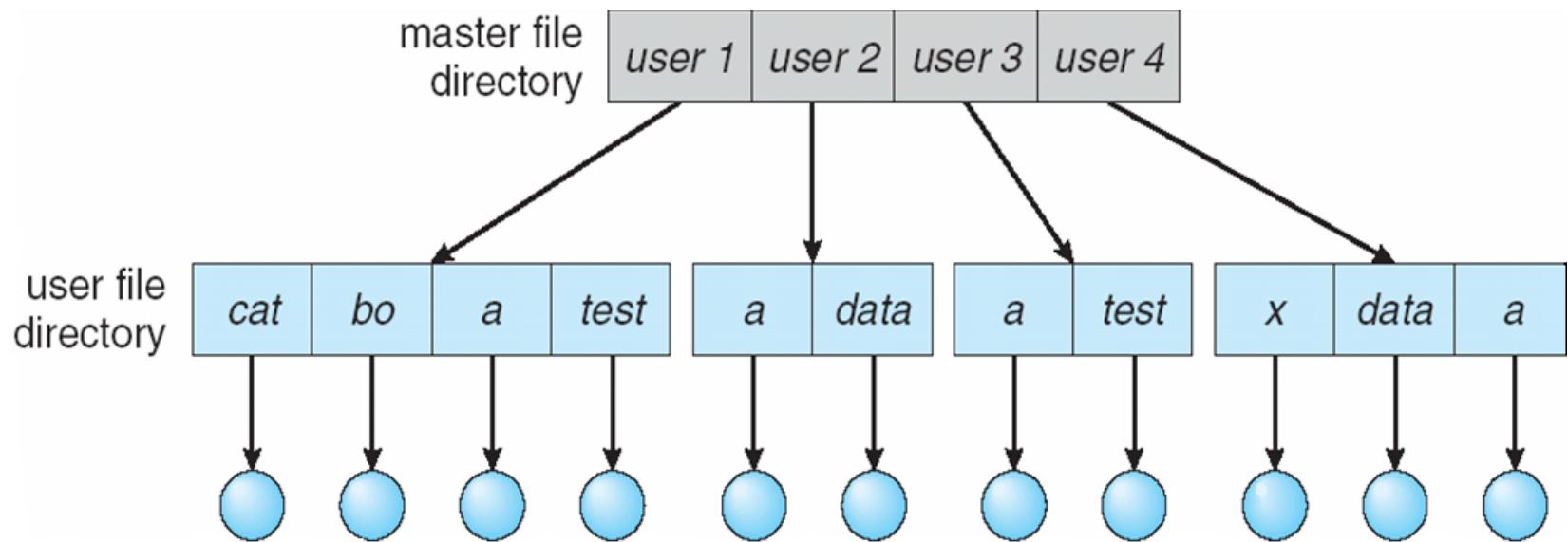


Naming problem

Grouping problem

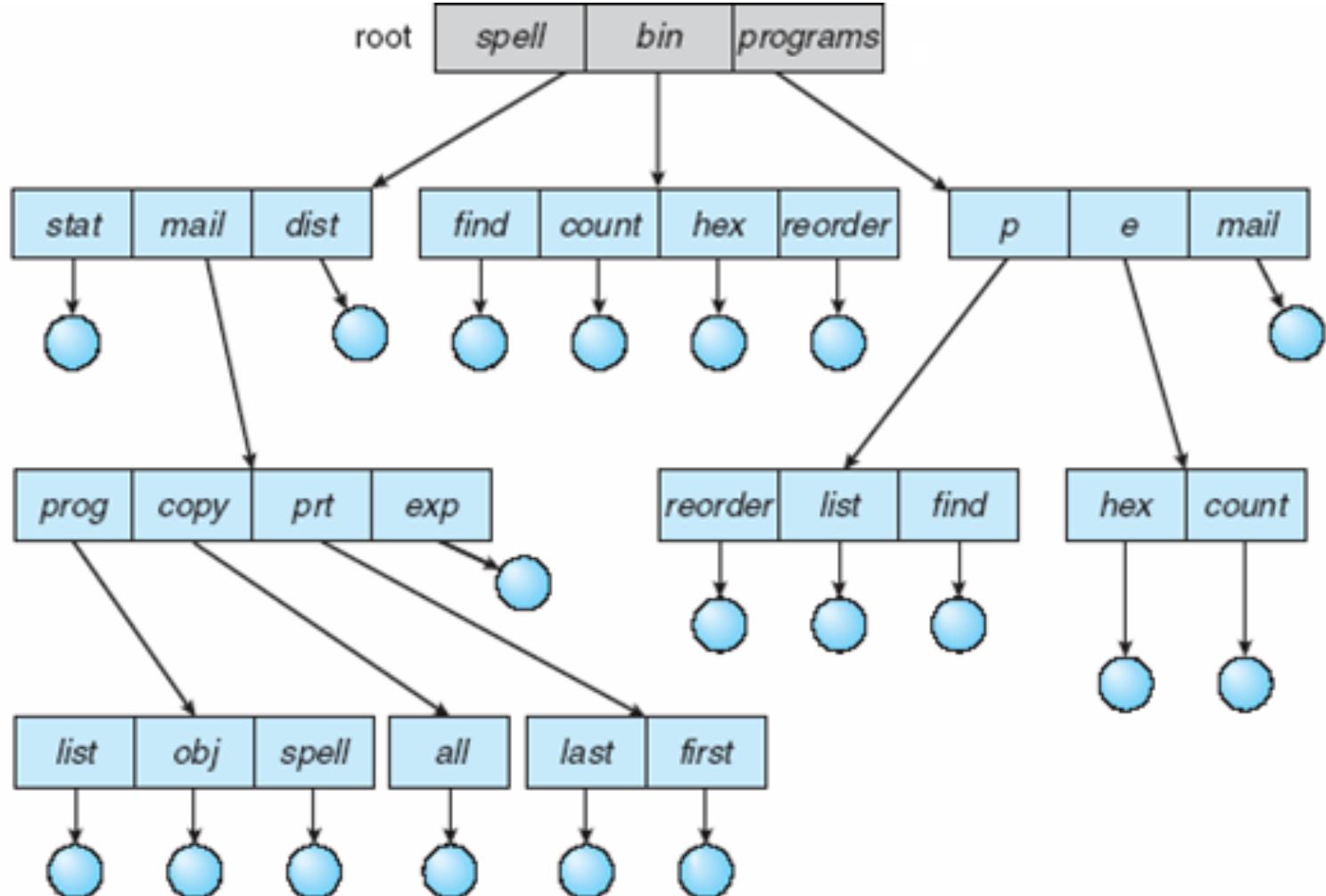
Two-Level Directory

- Separate directory for each user



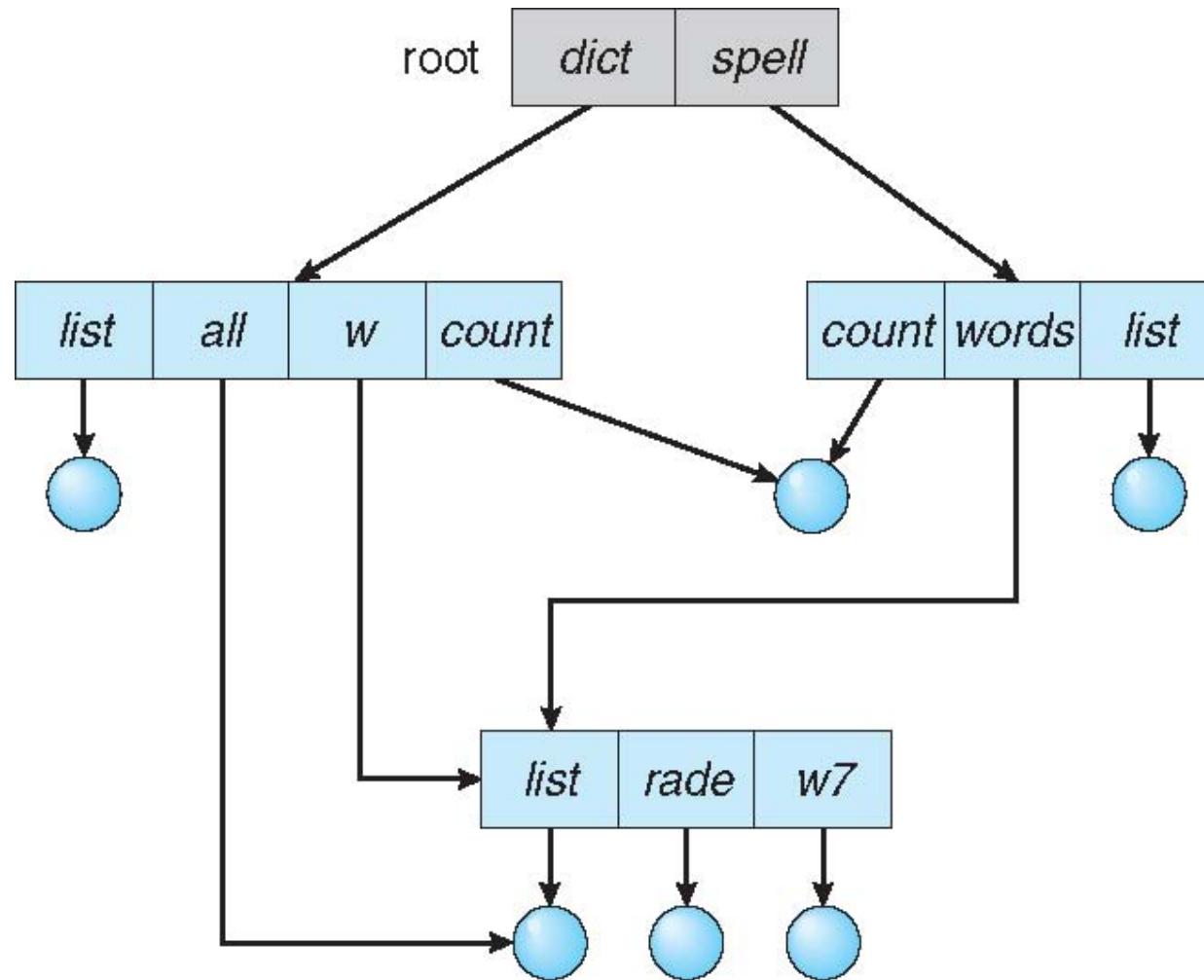
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

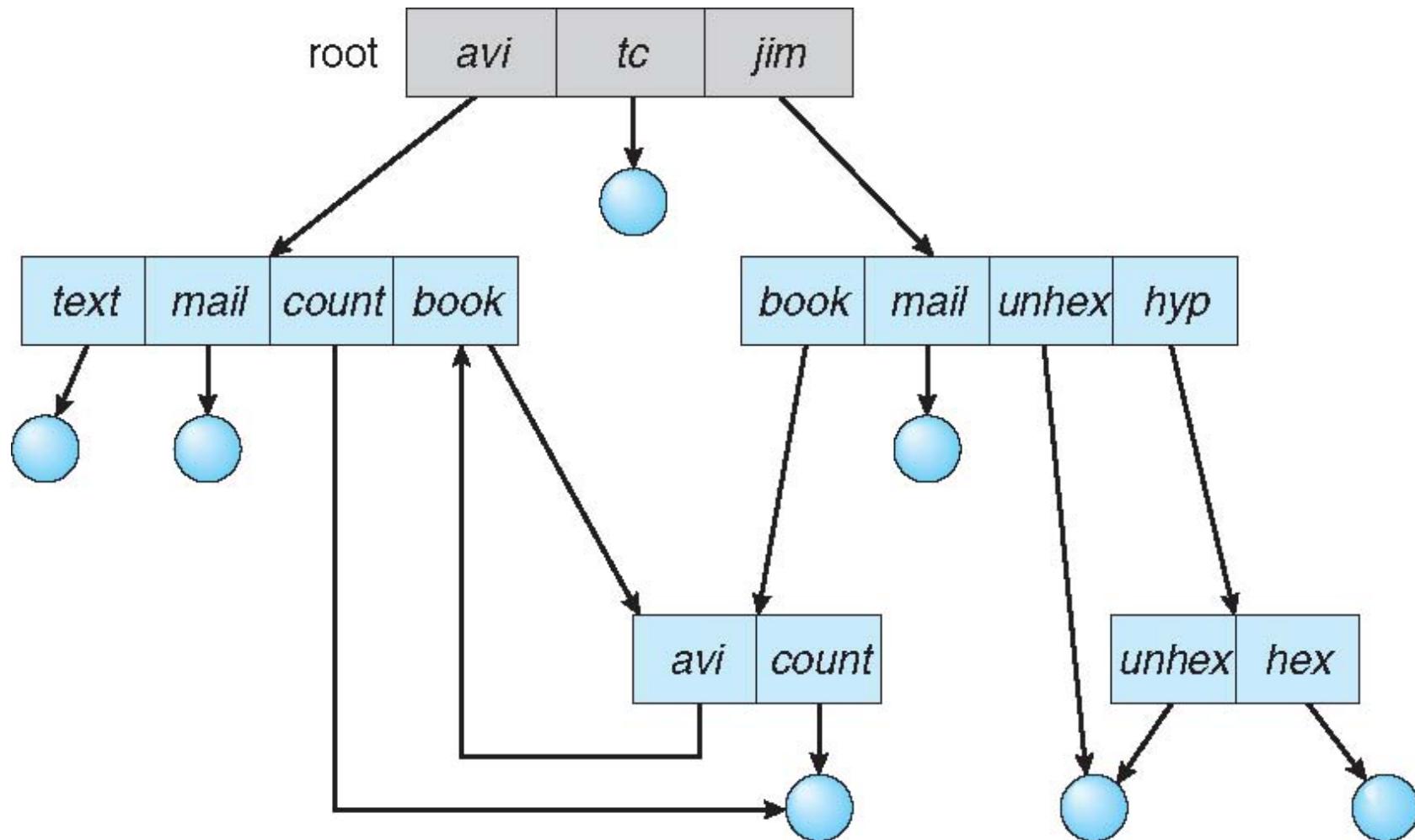


Directed Acyclic-Graph Directories

- Have shared subdirectories and files



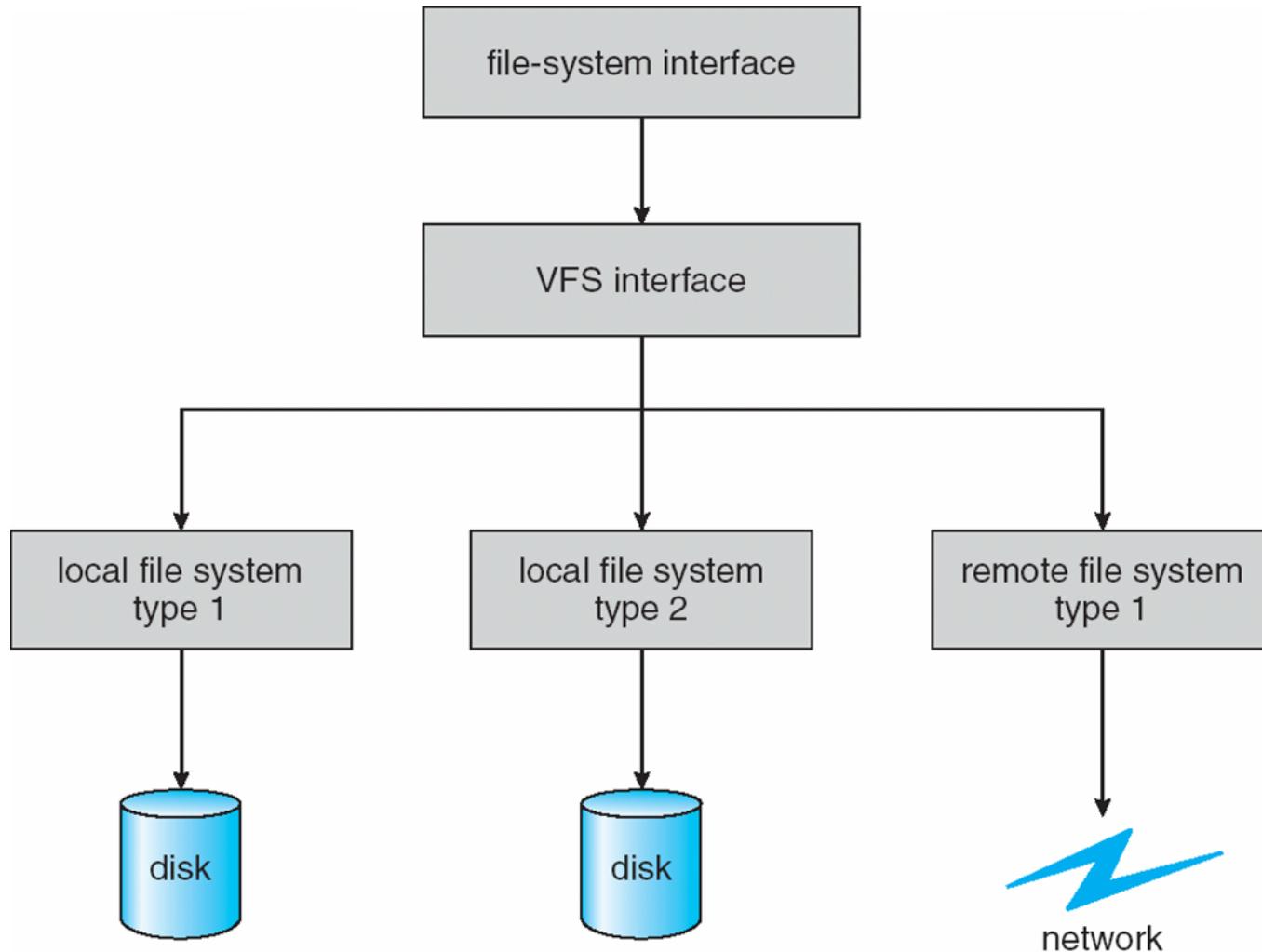
General Graph Directory



Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Directory Implementation

- Linear list of file names with pointer to the data blocks.
 - simple to program
 - time-consuming to execute
- Hash Table – linear list with hash data structure.
 - decreases directory search time
 - **collisions** – situations where two file names hash to the same location
 - fixed size

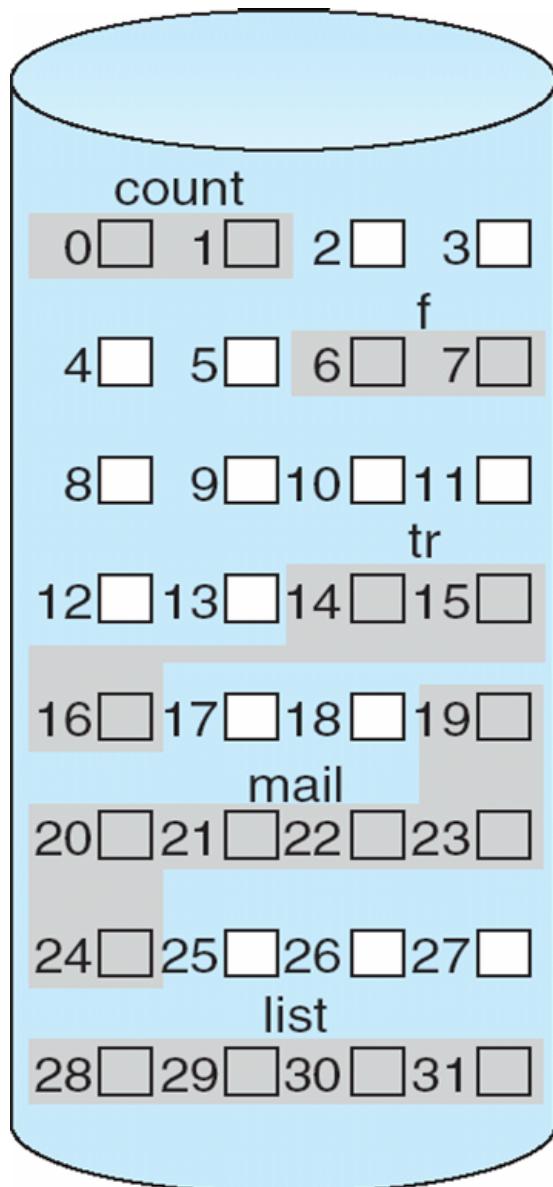
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

Contiguous Allocation of Disk Space



directory

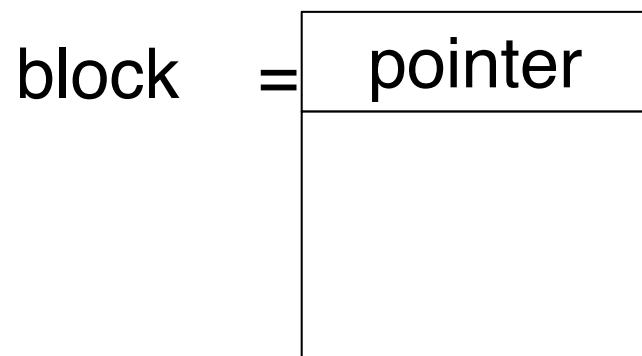
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Extent-Based Systems

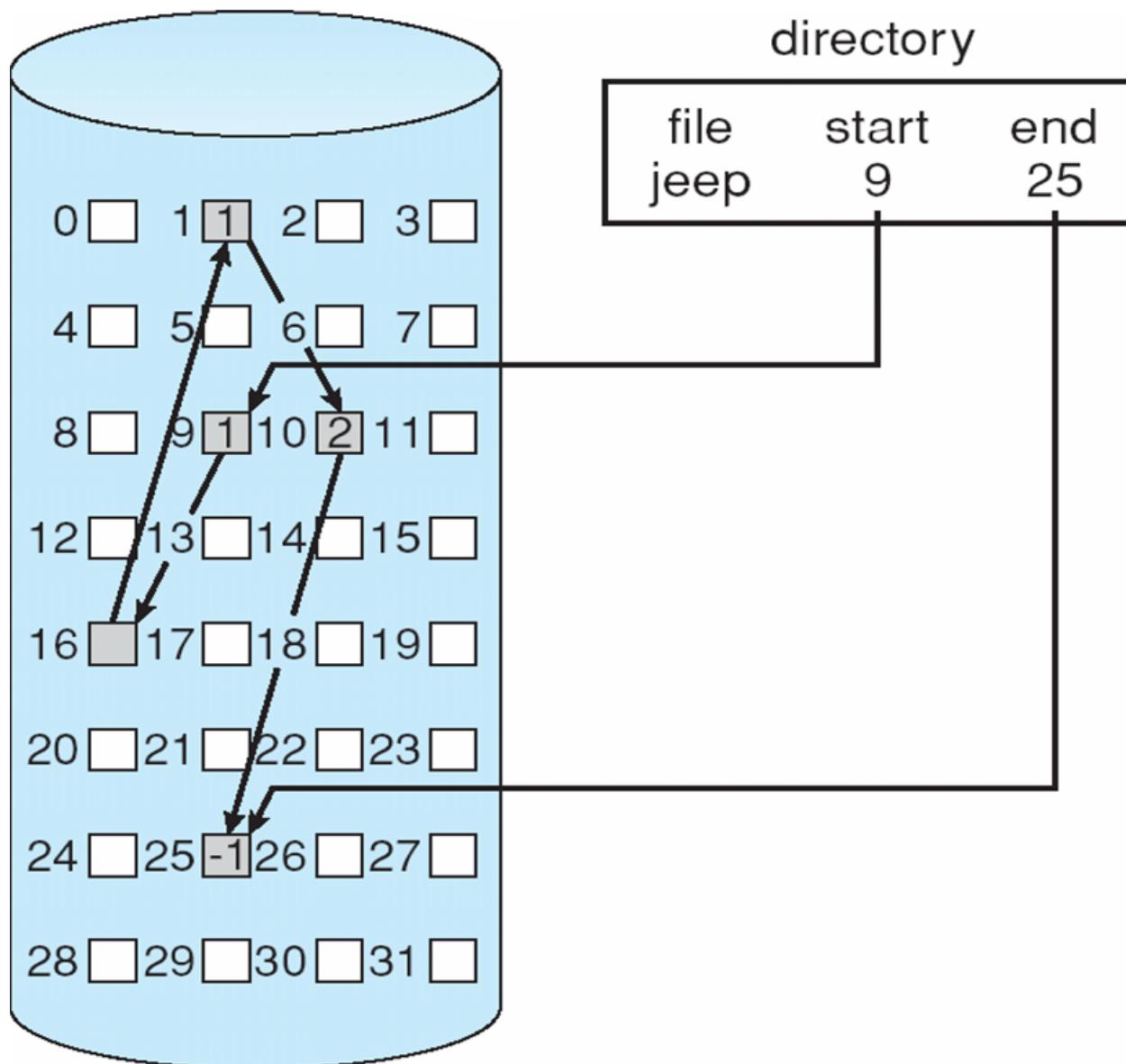
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
 - Extents are allocated for file allocation
 - A file consists of one or more extents

Linked Allocation

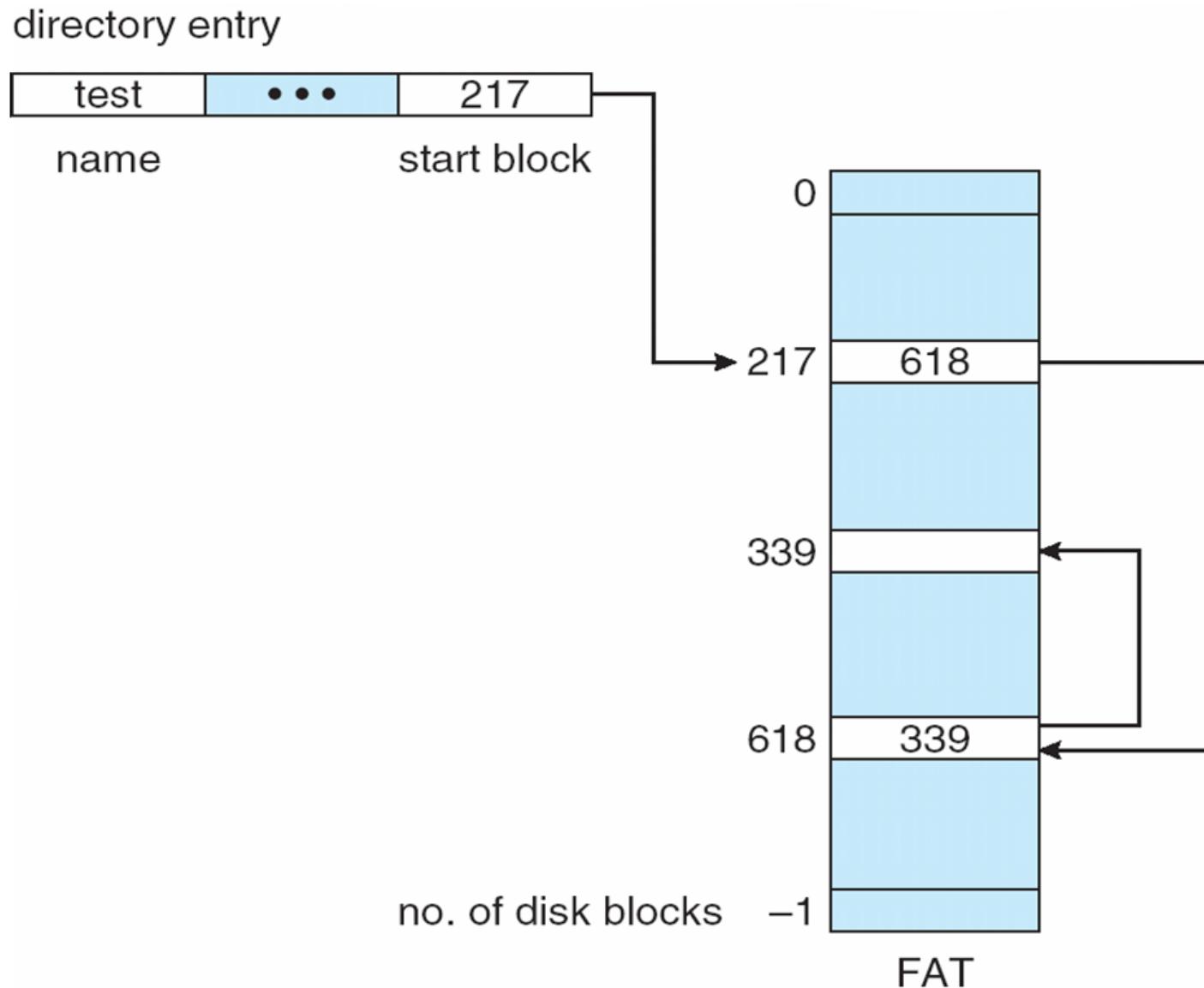
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



Linked Allocation

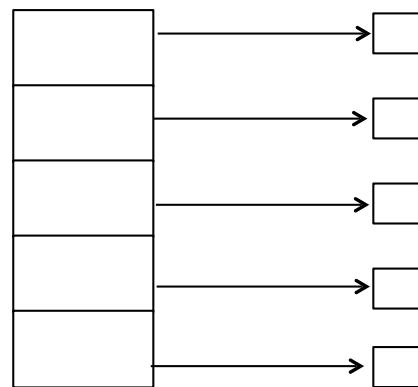


File-Allocation Table



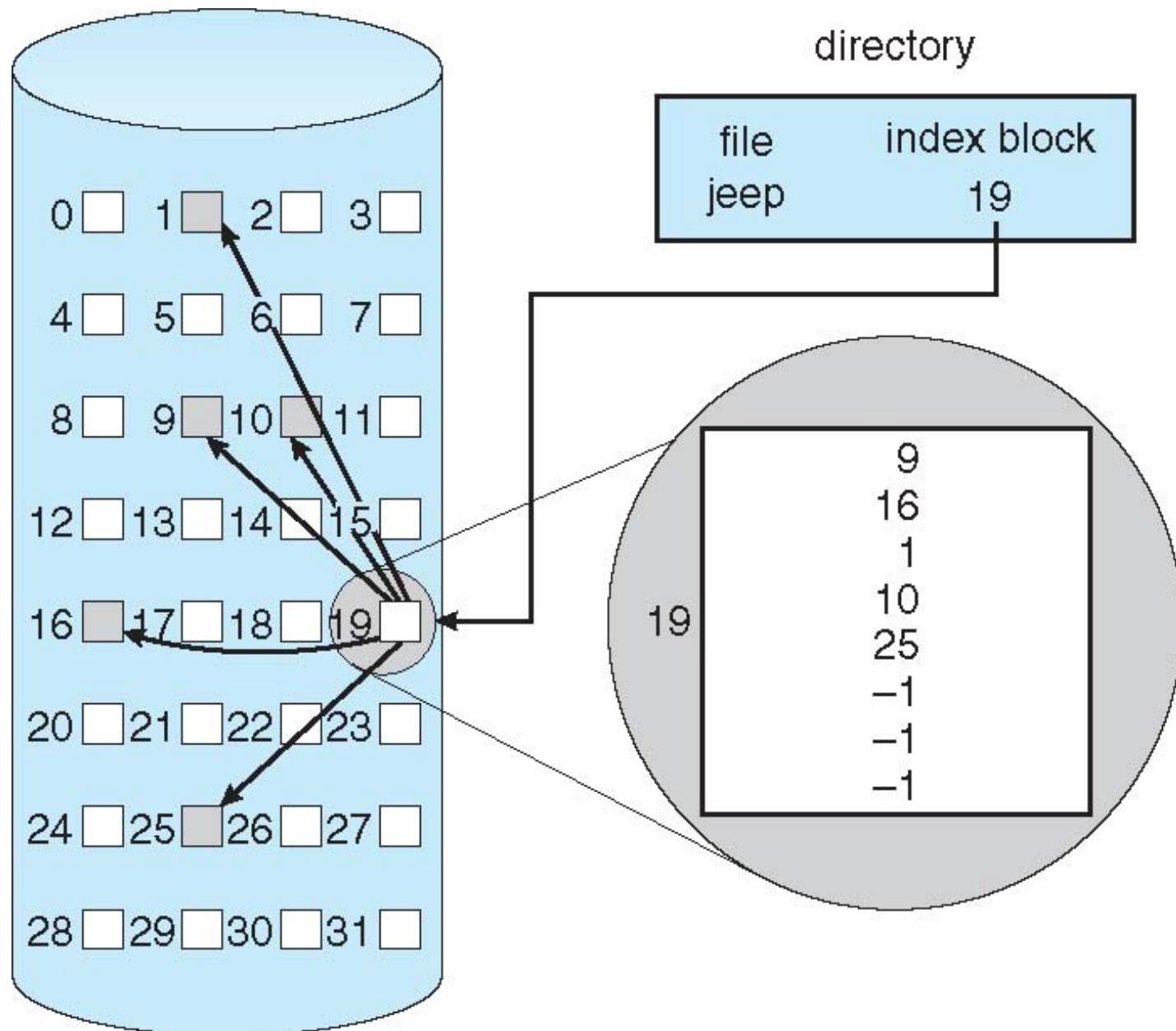
Indexed Allocation

- Brings all pointers together into the **index block**
- Logical view

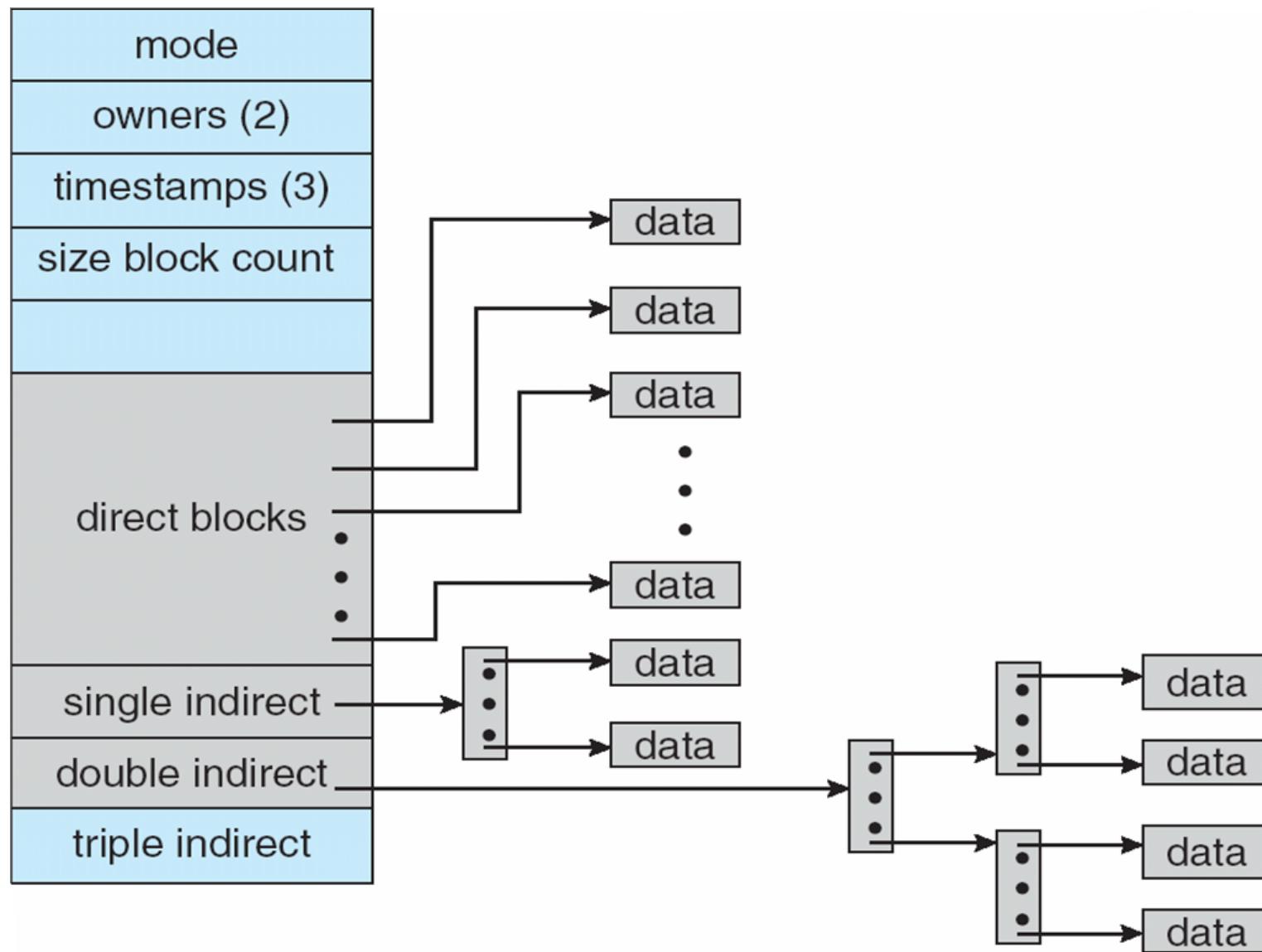


index table

Example of Indexed Allocation



Combined Scheme: UNIX UFS
(4K bytes per block)



Free-Space Management (Cont.)

- Bit map requires extra space
 - Example:
 - block size = 2^{12} bytes
 - disk size = 2^{30} bytes (1 gigabyte)
 - $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
 - Easy to get contiguous files
 - Linked list (free list)
 - Cannot get contiguous space easily
 - No waste of space
 - Grouping
 - Counting

Free-Space Management (Cont.)

- Need to protect:
 - Pointer to free list
 - Bit map
 - Must be kept on disk
 - Copy in memory and disk may differ
 - Cannot allow for $\text{block}[i]$ to have a situation where $\text{bit}[i] = 1$ in memory and $\text{bit}[i] = 0$ on disk
- Solution:
 - Set $\text{bit}[i] = 1$ in disk
 - Allocate $\text{block}[i]$
 - Set $\text{bit}[i] = 1$ in memory

Efficiency and Performance

- Efficiency dependent on:
 - disk allocation and directory algorithms
 - types of data kept in file's directory entry
- Performance
 - disk cache – separate section of main memory for frequently used blocks
 - free-behind and read-ahead – techniques to optimize sequential access
 - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup (full vs. incremental backups)

Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each update to the file system as a **transaction**
- All transactions are written to a log
 - A transaction is considered committed once it is written to the log
 - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
 - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

Principles of I/O Hardware

- » I/O can be roughly divided into two categories
- » Block devices - Stores information in a fixed size block each with its own address. Transfers are in units of blocks
- » Character devices - Delivers or accepts a stream of characters without regard to a block structure.

Principles of I/O Hardware

- » Classification scheme is not perfect.
- » Clocks, not block addressable, don't generate or accept streams.
- » Memory mapped screens
- » Touch screens

Device Controllers

- » I/O often consists of a mechanical component and an electronic component.
- » Generally split into two modular portions
- » Electric component is the device controller or device adapter.
- » Mechanical component is the device itself

Device Controllers

- » If the interface between the controller and device is a standard interface companies can make controllers or devices that fit that interface.
- » ANSI, IEEE, ISO or de facto.

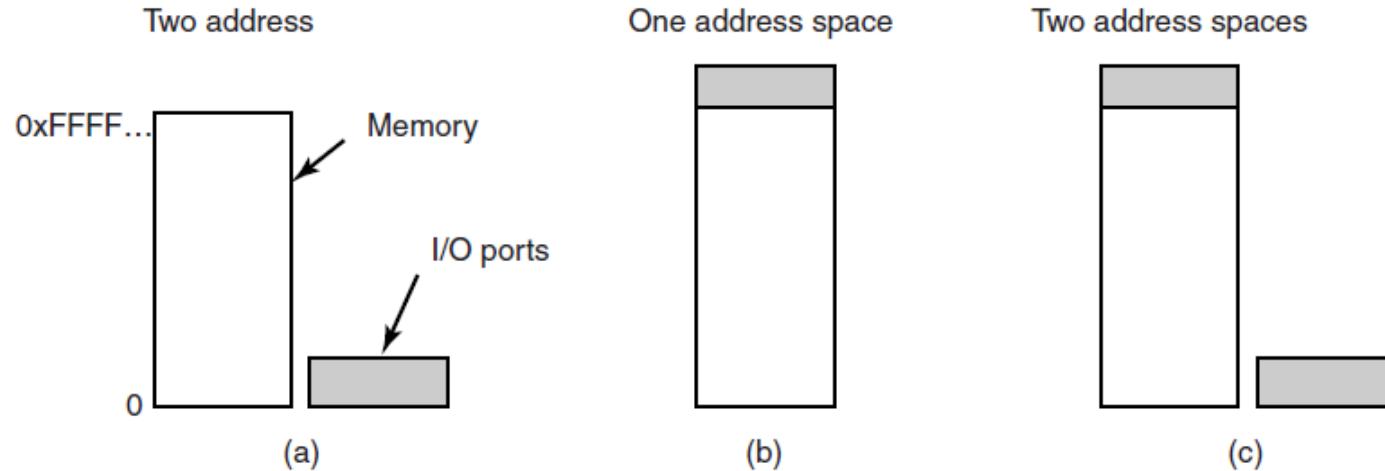
Standards

- » de facto standard - One vendor comes up with a good idea and other vendors follow the lead. Or some organizations come together and agree on a standard.
 - QWERTY keyboard layout
 - Microsoft Word DOC
- » de jure standard - Technically have the force of law behind them. Set by professional, national, or international organizations such as IEEE, ANSI and ISO.
 - PDF and HTML (started de facto eventually made de jury)
 - 802.11

Memory-Mapped I/O

- » Each controller has a couple registers used for communicating with the CPU.
- » Command the device to deliver data
- » Switch it on/off
- » Other actions
- » Device also maintains data buffer.
- » Video RAM is effectively a data buffer

Device Controllers



- » Two methods for CPU to communicate
 - » Each control register is assigned an I/O port number (8 or 16 bit number)
 - » Set of all ports is the I/O port space.
 - » Ordinary user programs can not access
 - » IN REG PORT or OUT PORT REG

Device Controllers

- » Different strengths and weaknesses
- » Memory mapped I/O
 - » If special I/O instructions needed then need to access the control registers with assembly
 - » Memory mapped allow devices drivers to be written in C.
 - » No memory protection needed to keep user processes from performing I/O.
 - » O/S just doesn't map I/O address range in process address space

Device Controllers

- » Different strengths and weaknesses
- » Memory mapped I/O
 - » Multiple devices can be given their own page of memory. Access can be given to users over some devices and not others
 - » Different address spaces keep drivers from conflicting with each other.

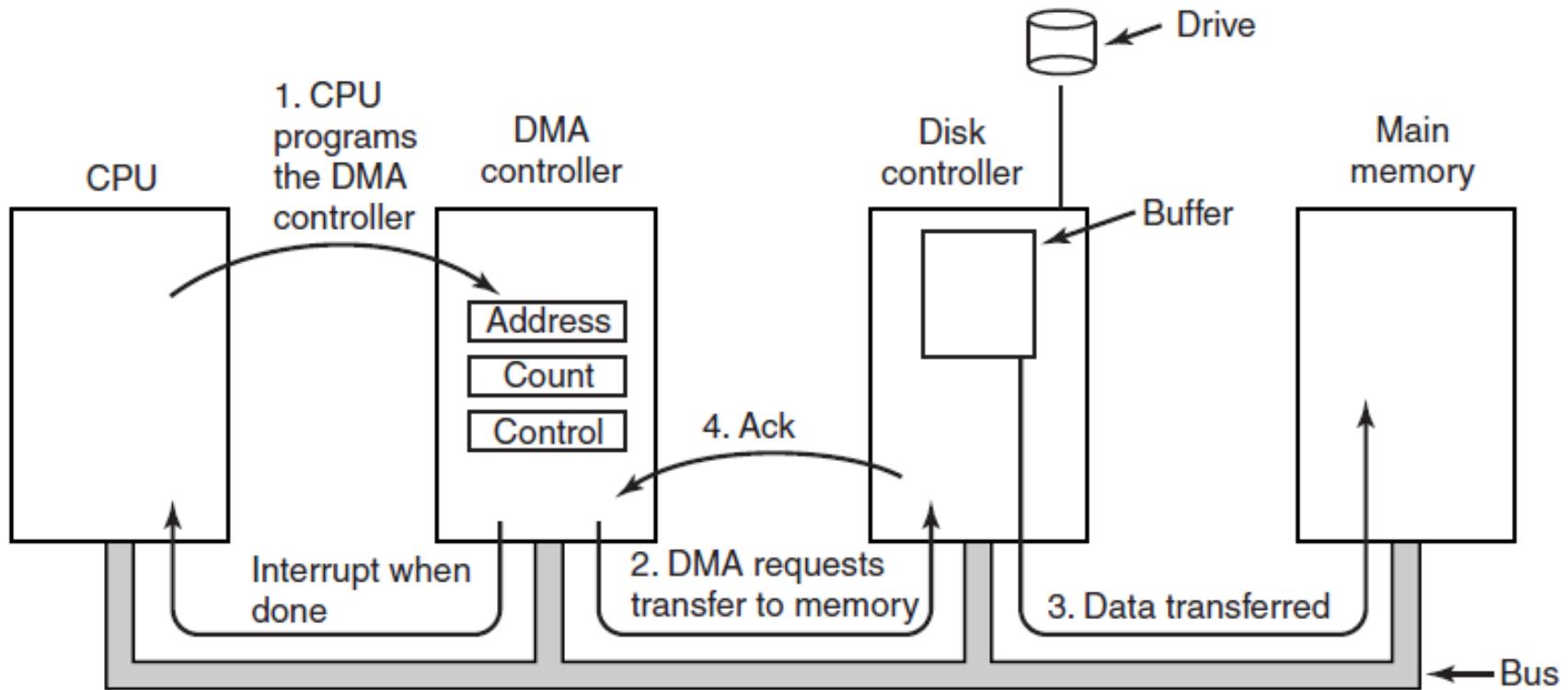
Device Controllers

- » Different strengths and weaknesses
- » Memory mapped I/O weaknesses
 - » Caching a control register would be disastrous.
 - » Hardware has to disable caching on a per page basis.
 - » If there is only one address space then all memory modules and all I/O devices must examine all memory references to see which to respond to.

Direct Memory Access

- » CPU can request data from the controller one byte at a time, but wastes the CPUs time.
- » Direct Memory Access (DMA)
 - » Must have a DMA controller
 - » DMA controller has access to the bus independent of the PCU.
 - » CPU can read/write to the DMA controller control registers.
 - » Address, Count, Control

Direct Memory Access



Direct Memory Access

- » DMA:
 - » More sophisticated DMA can handle multiple transfers at a time.
 - » Round robin the requests or priority requests
 - » System busses can operate in two modes: word-at-a-time and block mode.
 - » Word-at-a-time mode allows the device controller to sneak in and steal an occasional bus cycle from the CPU
 - » Cycle Stealing

Direct Memory Access

- » DMA:
 - » Block mode - the DMA controller tells the device to acquire the bus, issue the transfers then release the bus.
 - » Burst mode
 - » More efficient than cycle stealing
 - » Blocks the CPU and other devices

Direct Memory Access

- » DMA:
 - » Fly-by mode - DMA tells the device controller to write directly to main memory
 - » Alternate mode allows the device to send to the DMA controller which then issues a second bus request to write to a destination
 - » Allows device to device DMA
 - » Some computers don't use DMA
 - » Main CPU is often faster than DMA controller
 - » Idling CPU waiting for slower DMA is pointless
 - » Less hardware means cheaper cost

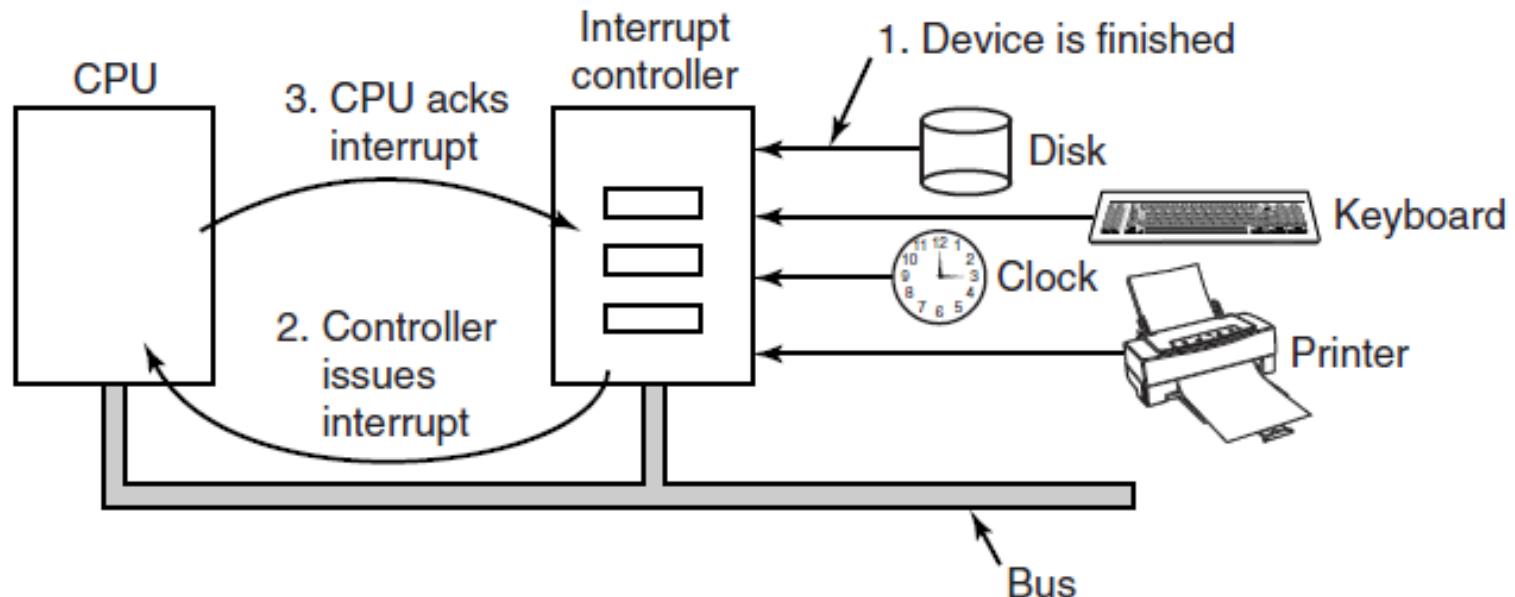
Interrupts Revisited

- » Refresher:
 - » When an I/O device is done with its work it issues an interrupt.
 - » Asserts a signal on a bus line that it has been assigned.
 - » Interrupt controlled chip on motherboard decides what to do
 - » If no interrupt pending, the interrupt controller handles the interrupt immediately.
 - » If other interrupts in progress it is ignored until the interrupt controller is free.
 - » Interrupt handler puts a number on the address line specifying which device needs attention and asserts a signal to interrupt the CPU.

Interrupts Revisited

» Refresher:

- » The number on the address line is used as an index into the interrupt vector table that allows the CPU to fetch a new program counter.



Interrupts Revisited

- » When executing the interrupt, what does the CPU save?
 - » Program counter is bare minimum
 - » All visible registers and most internal registers at the other end
- » Where do you put the data?
 - » Internal registers, but then can't acknowledge interrupt handled until all registers read back out
 - » Takes time, leaves dead space
 - » Stack
 - » Can't be user stack. Stack pointer may not be legal
 - » Might be on the end of a page. If you page fault where do you save the state to handle the fault?

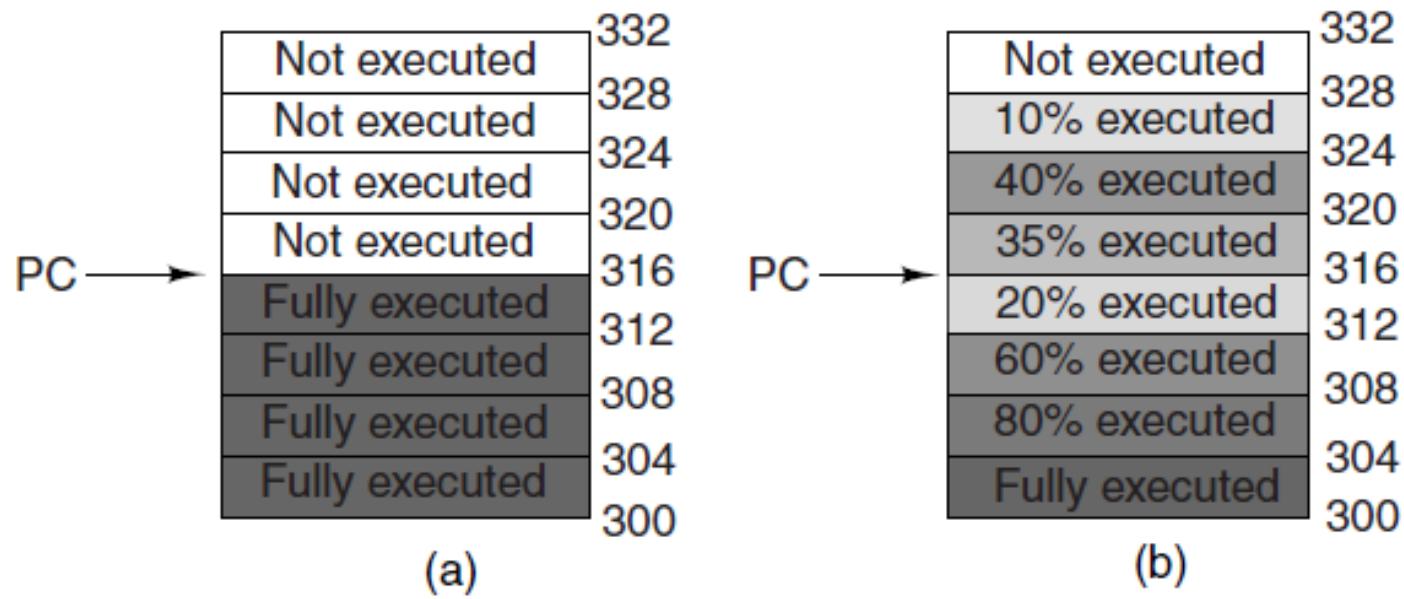
Interrupts Revisited

- » Kernel stack?
- » May require switching into kernel mode
 - » Change MMU context
 - » Invalidate the cache and TLB

Interrupts Revisited

- » Kernel stack?
 - » May require switching into kernel mode
 - » Change MMU context
 - » Invalidate the cache and TLB
 - » Modern CPUs are heavily pipelined, often superscalar.
 - » Can't assume if an interrupt occurs after an instruction that all instructions leading up to and including that instruction have been executed completely.
 - » Many partially executed instructions.

Interrupts Revisited



(a) A precise interrupt. (b) An imprecise interrupt.

Interrupts Revisited

- » Precise Interrupt - Interrupt that leaves the machine in a well-defined state.
- » Four properties of a precise interrupt:
 1. The PC saved in a known place.
 2. All instructions before that pointed to by PC have fully executed.
 3. No instruction beyond that pointed to by PC has been executed.
 4. Execution state of instruction pointed to by PC is known.

Interrupts Revisited

- » Imprecise Interrupt - Interrupt that does not meet the four requirements
 - » Unpleasant for OS developer.
 - » Machines with imprecise interrupts usually vomit a large amount of internal state onto the stack to give the operating system the possibility of figuring out what was going on.
 - » Large amount of work to restart
 - » Can make very fast superscalar CPUs unsuitable for real-time work due to slow interrupts
- » x86 has very complex logic in the CPU to have precise interrupts

Principle of I/O software

- » Key concepts:
 - » Device independence - Write a program that can access any I/O without having to specify the device in advance
 - » open should work on any device: drive, flash, cd
- » Uniform Naming - should not depend on device in any way.

Principle of I/O software

- » Key concepts:
 - » Error handling: errors should be handled as close to the hardware as possible.
 - » If the controller discovers a read error it should correct it .
- » Synchronous / Asynchronous
 - » Most physical I/O is asynchronous
 - » Blocking programs easier to write

Principle of I/O software

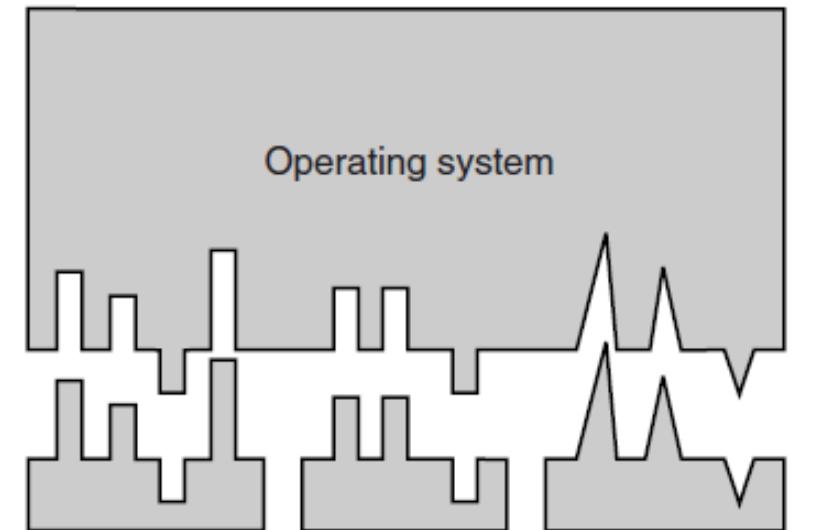
- » Key concepts:
 - » Buffering: Usually can't store data directly off the device into its final destination
 - » Sometime must decouple rate at which the data is filled from the rate at which it is emptied.
- » Shareable / Dedicated devices
 - » Drives can have multiple users sharing
 - » Printers can only have one user at a time

Programmed I/O

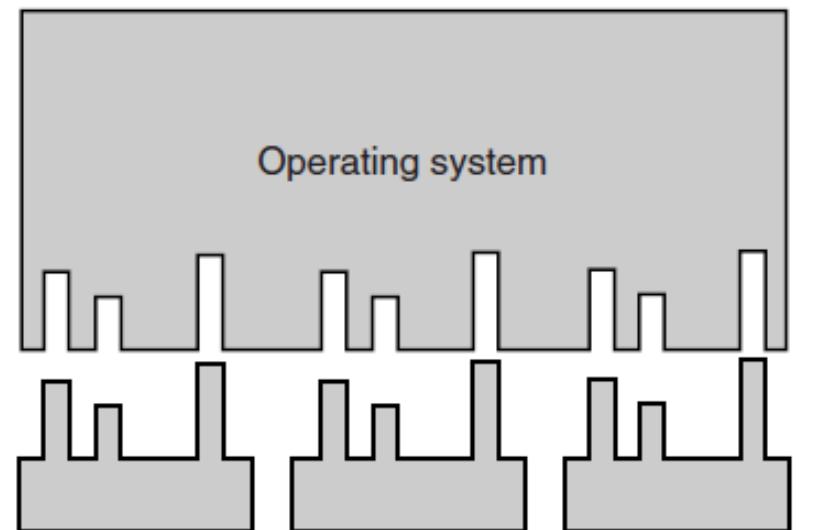
- » Three fundamentally different ways I/O can be performed.
 - » Programmed I/O
 - » Interrupt Driven I/O
 - » I/O using DMA
- » Simplest is programmed I/O, also known as letting the CPU do all the work.

Uniform Interfacing

» How do we make all the drivers and I/O devices look the same?



(a)



(b)

- (a) Without a standard driver interface.
- (b) With a standard driver interface.

Uniform Interfacing

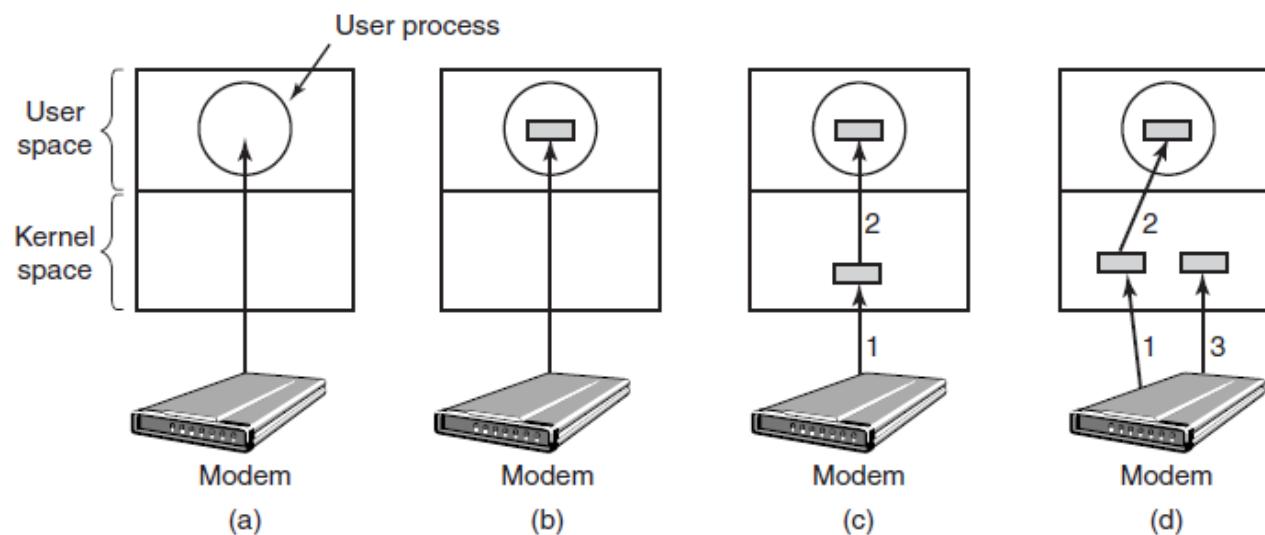
- » Device driver writers know what is expected in their drivers
- » While all I/O devices are not identical usually there are a small number of device types
- » Each class defines a set of operations a driver must supply

Uniform Interfacing

- » Uniform naming.
 - » /dev/sda
 - » /dev/hda
- » Uniform protections and permissions

Buffering

- » Both block and character devices must deal with buffering.



- (a) Unbuffered input. (b) Buffering in user space. (c) Buffering in the kernel followed by copying to user space. (d) Double buffering in the kernel.

Buffering

- » (a) Unbuffered input.
 - » read, block, interrupt, repeat
 - » inefficient
- » (b) Buffering in user space.
 - » Process provides n-byte buffer
 - » When buffer full the process is awakened
 - » What if buffer is paged out?

Buffering

- » (c) Buffering in the kernel followed by copying to user space.
- » Copy in one operation
- » (d) Double buffering in the kernel.
 - » Copy one while accumulating in another
 - » Circular buffer

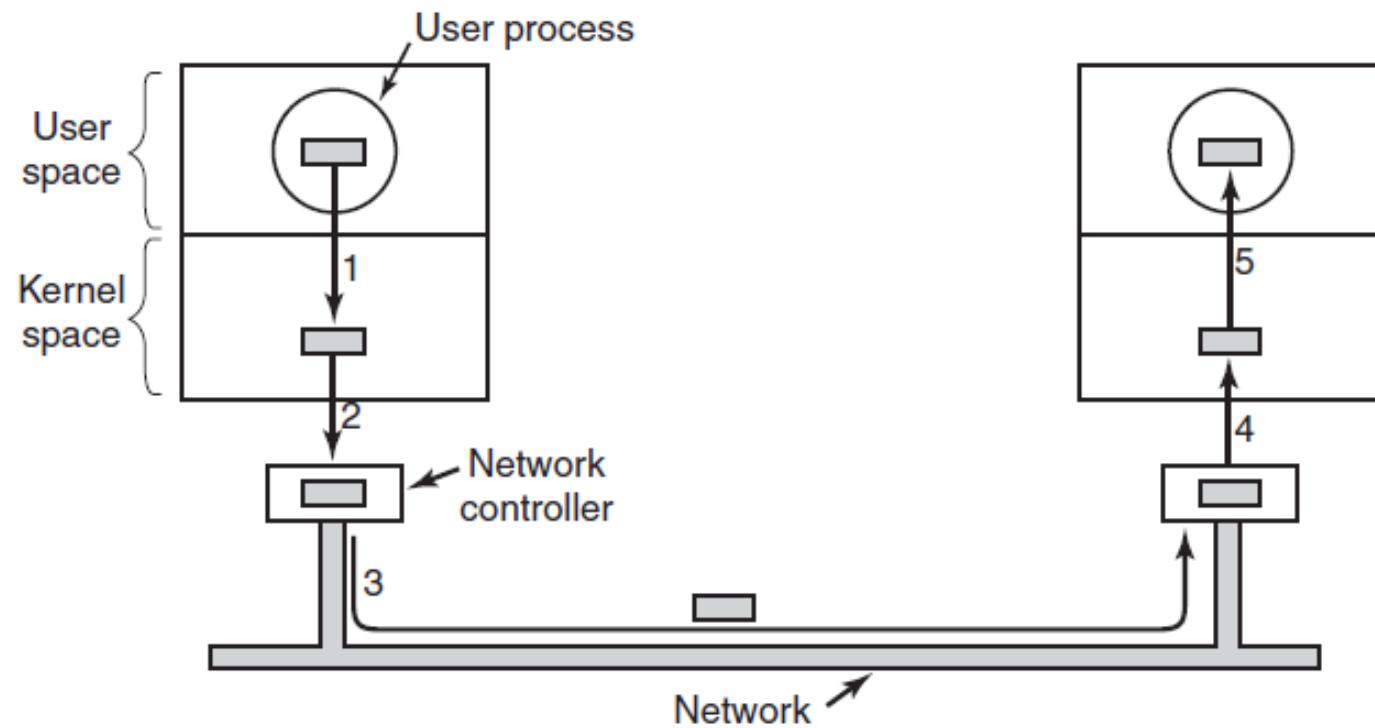
Buffering

- » Also important on output
- » On write:
 - » Block the user until all characters written.
 - » Slow
- » Release the user and process I/O while user process continues
 - » How do you know when it's finished?

Buffering

- » System could generate signal or interrupts
 - » Prone to race conditions
- » Better to copy to kernel buffer and in (c)
- » Too many buffers on the path cause performance problems.

Buffering



Error Reporting

- » Errors are far more common in I/O context than in any other.
- » Device specific and handled by driver
- » Framework for error handling is device independent

Classes of Errors

1. Programmer error

- Writing to an input
- Just report error

2. I/O error

- Writing to bad disk block
- Driver needs to determine what to do
- May pass error up to device independent software

Dedicated Devices

- » Some device, such as printers, can only be used by a single process at a time
 - Operating system adjudicates requests
 - Require processes to call `open()`
 - Special mechanism for requesting and releasing devices
 - Processes block and wait in queue

Device Independent Block Size

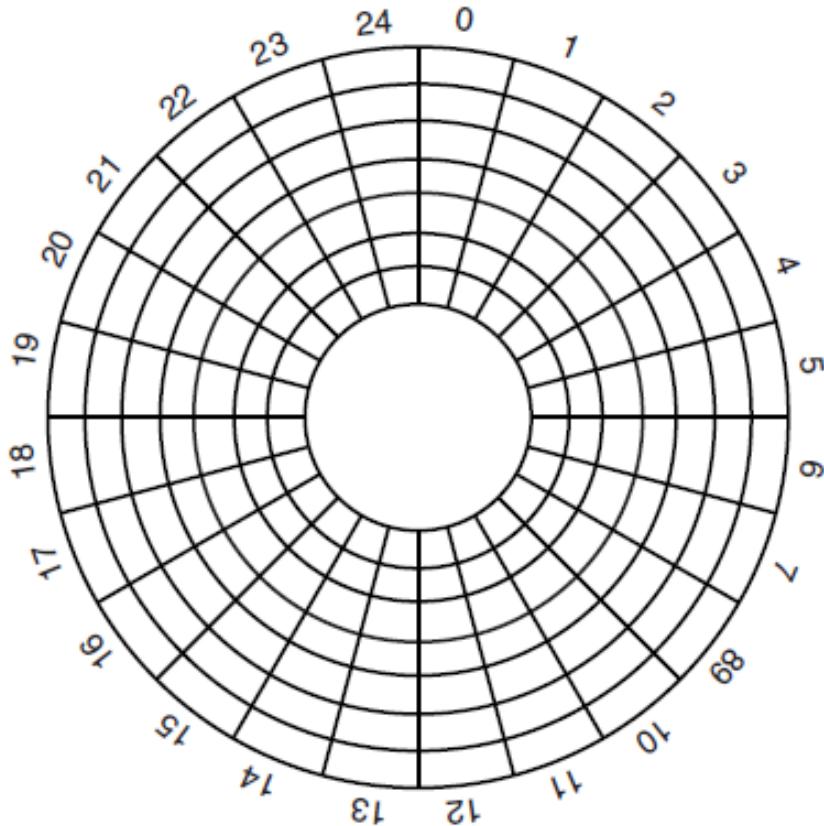
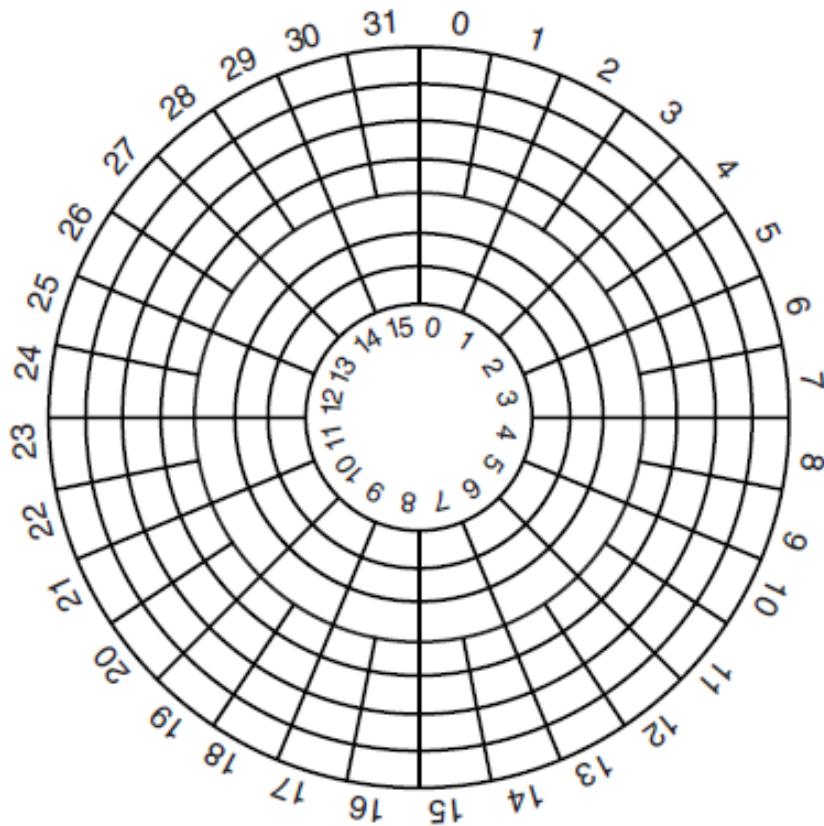
- » Different disks may have different block sizes.
 - Device independent software hides this and provides a uniform block size to higher layers.
 - Treat several sectors as a single logical block.

Disks

- » Magnetic disks are organized into cylinders, each having as many tracks as there are heads stack vertically.
 - Tracks are divided into sectors
- » Early disks delivered serial bit stream
- » IDE and SATA have a microcontroller that does considerable work and allows the real controller to issue higher commands.
 - Overlapped seeks

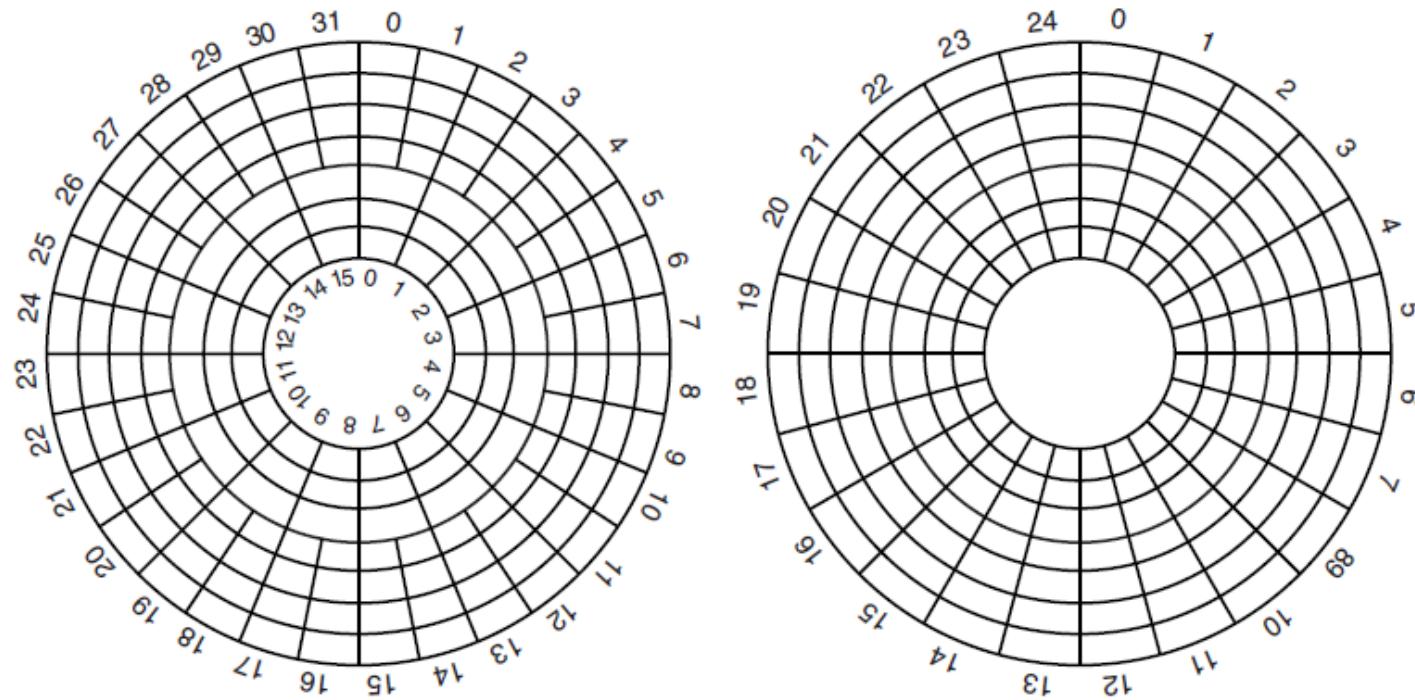
Disks

- » Geometry specified and used by the driver software is almost always different from the physical format.



Disks

- » Logical Block Addressing - disk sectors are numbered consecutively starting at 0 with no regard for the disk geometry,



RAID Structure

- Reliability measured in mean time between failure
- RAID - redundant array of inexpensive disks
- RAID – multiple disk drives provides reliability via redundancy
- Frequently combined with NVRAM to improve write performance
- RAID is arranged into six different levels

RAID (Cont.)

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk [striping](#) uses a group of disks as one storage unit
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
 - [Mirroring or shadowing \(RAID 1\)](#) keeps duplicate of each disk
 - Striped mirrors ([RAID 1+0](#)) or mirrored stripes ([RAID 0+1](#)) provides high performance and high reliability
 - [Block interleaved parity \(RAID 4, 5, 6\)](#) uses much less redundancy

RAID (Cont.)

- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

RAID Levels



(a) RAID 0: non-redundant striping.



(b) RAID 1: mirrored disks.



(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



(f) RAID 5: block-interleaved distributed parity.



(g) RAID 6: P + Q redundancy.

RAID 0

- RAID 0 - Striped disk array without parity.
 - Data spread across multiple disk drives but no data redundancy
 - Improves performance because multiple reads and writes can be carried out at the same time.
 - Works best with large requests.
 - Works worst with OS that ask for data one sector at a time.
 - No parallelism

RAID 0

- RAID 0 - Striped disk array without parity.
 - Does not increase fault tolerance.
 - Actually decreases reliability since if one drive fails the all data is lost since the OS treats the array as a single drive.
 - N drives means configuration is N times as likely to fail.

RAID 1

- RAID 1 - Mirroring
 - Duplicate set of drives
 - When data is written to one it's also written to its duplicate.
 - When one fails, swap in new and copy the data over.

RAID 1

- RAID 1 - Mirroring
 - Assuming drive and its mirror can be read at the same time it provides twice the read transaction rate
 - Write transaction is unaffected.
 - No performance gain
 - Most expensive raid configuration

RAID 2 and RAID 3

- RAID 2 - Error-correcting coding
- RAID 3 - Bit interleaved parity
- Prohibitively expensive and inferior to other RAID levels.
- Drive spinning must be synchronized

RAID 4

- RAID 4 - Dedicated parity drive
- Block level striping like RAID 0 with a parity disk
- If a data disk fails the parity data is used to create a replacement disk.
- Every time a block is written the parity block must also be read, recalculated and rewritten. I/O bottleneck.

RAID 4

- Same reliability as RAID 1 but if a drive fails the performance hit is worse.
- Heavy load on the parity drive
 - Bottleneck

RAID 5

- RAID 5 - Block interleaved distribution parity
- Like RAID 4 but instead of parity on the same drive the parity block is assigned to the drives in a round robin fashion.
- Removes excessive use of the parity drive.

RAID 6

- RAID 6 - Independent data disks with double parity
- Block-level striping with parity across the disks like RAID 5 but instead of a simple parity scheme it calculates parity using two different algorithms.
- Requires an extra disk drive of RAID 5 but will tolerate the loss of two drives at the same time.

Disk Formatting

- » Hard disk consists of a stack of aluminum or glass platters.
- » On each is deposited a thin magnetizable metal oxide.
- » No information
- » Each platter must receive a low-level format.
- » Series of sectors with short gaps between

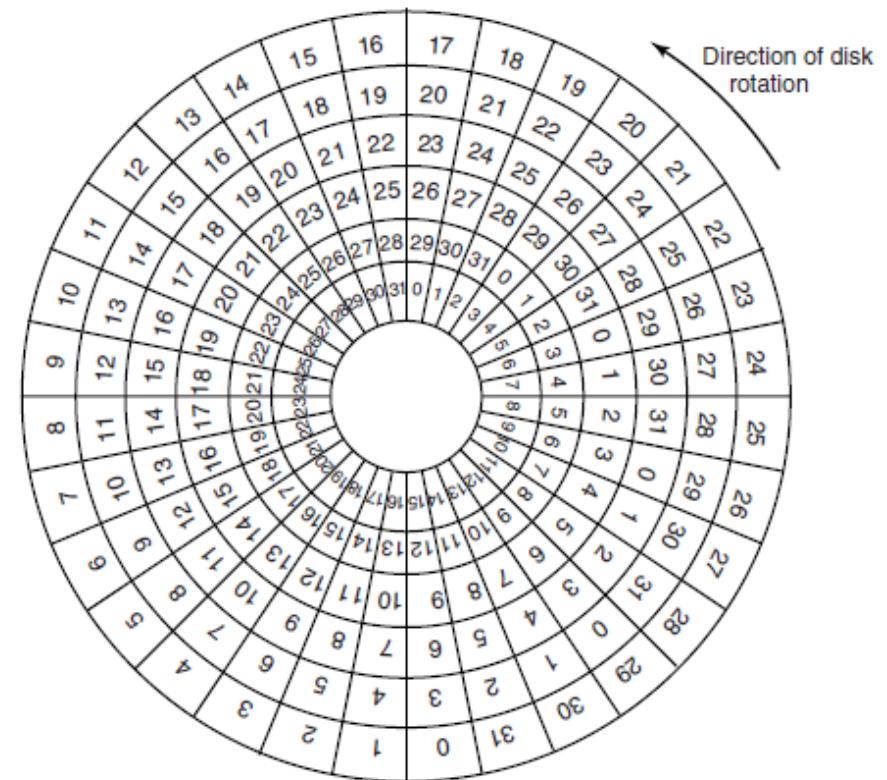
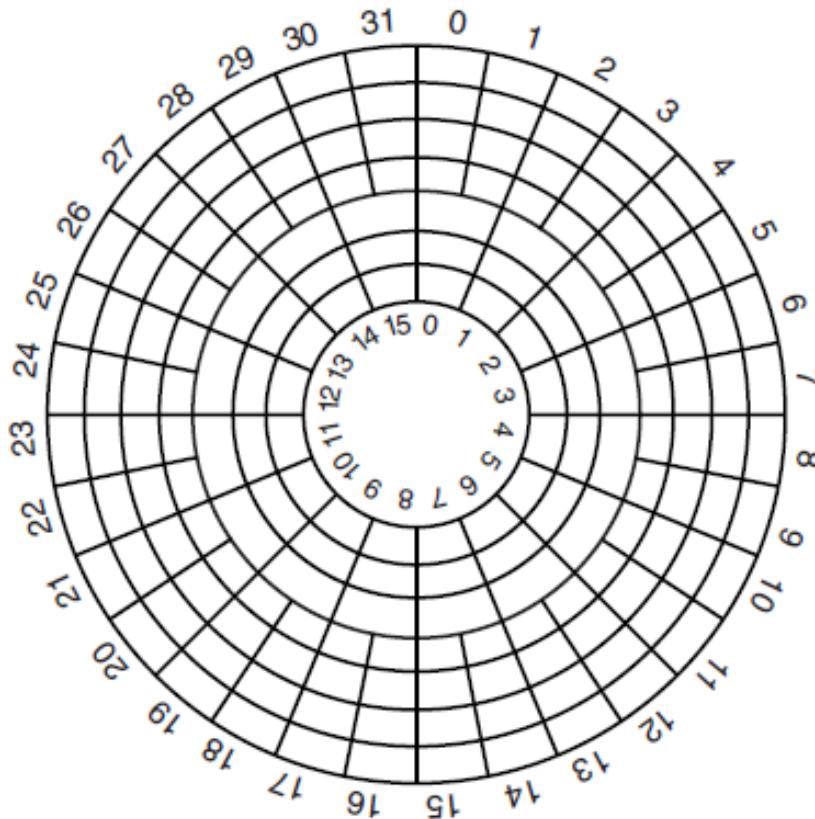
Preamble	Data	ECC
----------	------	-----

Disk Formatting

- » Preamble contains bit pattern hardware can recognize the sector.
- » Cylinder number
- » Sector number
- » Usually 512 -byte sectors.
- » ECC size varies by manufacturers but usually 16-bytes.

Disk Formatting

- » Cylinder skew - position of sector 0 is offset from previous track

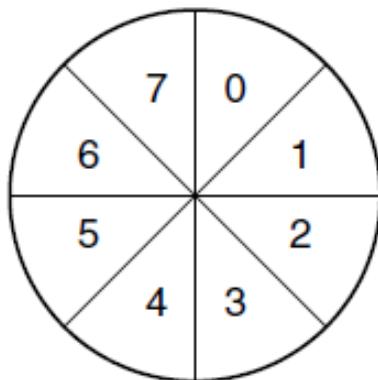


Disk Formatting

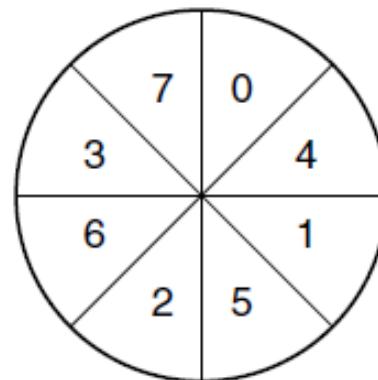
- » 10,000 RPM drive rotates in 5 msec. If a track has 300 sectors a new sector passes under the head every 20 usec.
- » If the track-to-track seek time is 800 usec, 40 sectors will pass by during the seek so skew must be at least 40 sectors.
- » Head skew - much smaller than cylinder skew, less than one sector of time.
- »

Disk Formatting

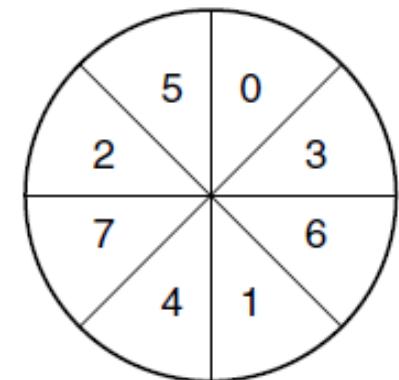
- » Transferring from controller to memory also causes delays that must be accounted for:
 - Single interleaving
 - Double interleaving



(a)



(b)



(c)

Disk Scheduling

- ❑ The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- ❑ Access time has two major components
 - **Seek time** is the time for the disk are to move the heads to the cylinder containing the desired sector
 - **Rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head
- ❑ Minimize seek time
- ❑ Seek time \approx seek distance
- ❑ Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

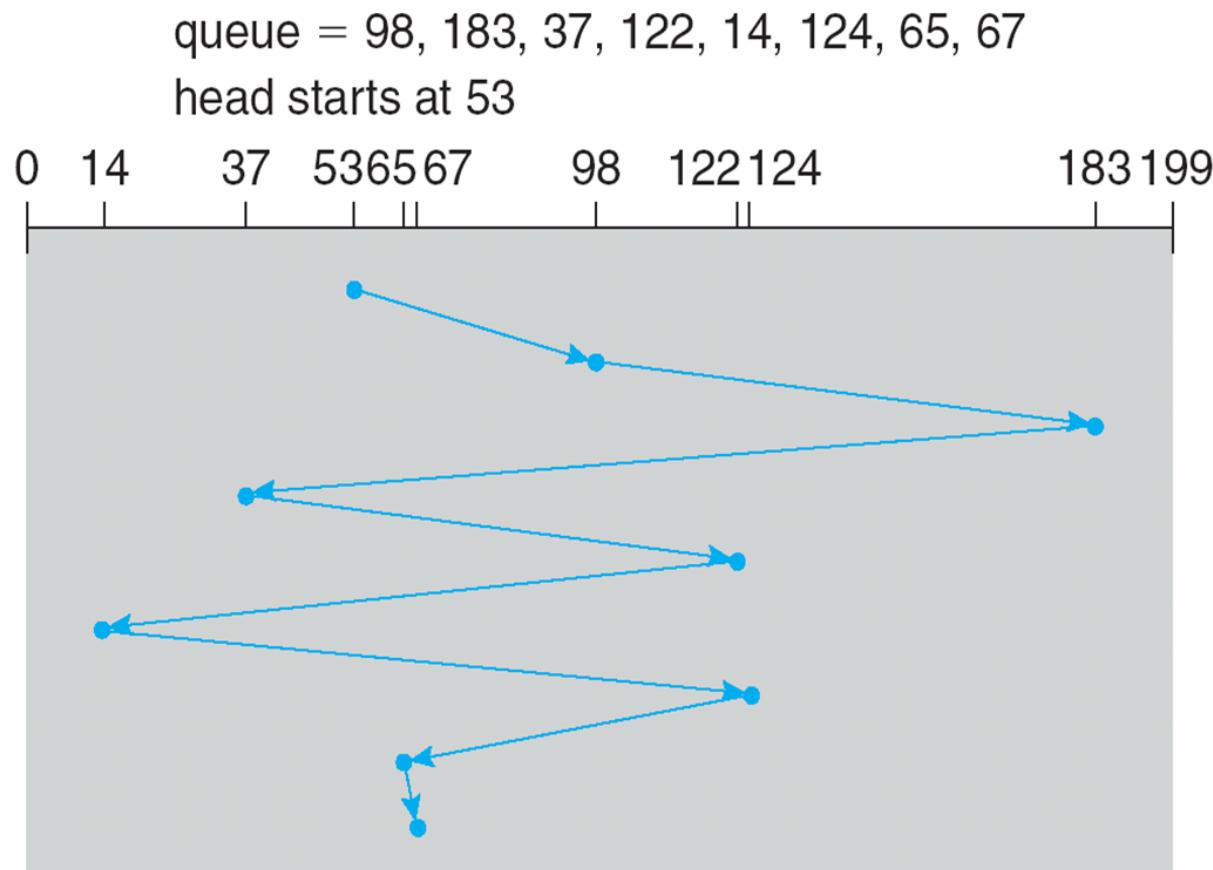
- ❑ Several algorithms exist to schedule the servicing of disk I/O requests
- ❑ We illustrate them with a request queue (0-199)

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

Head pointer 53

FCFS

Illustration shows total head movement of 640 cylinders



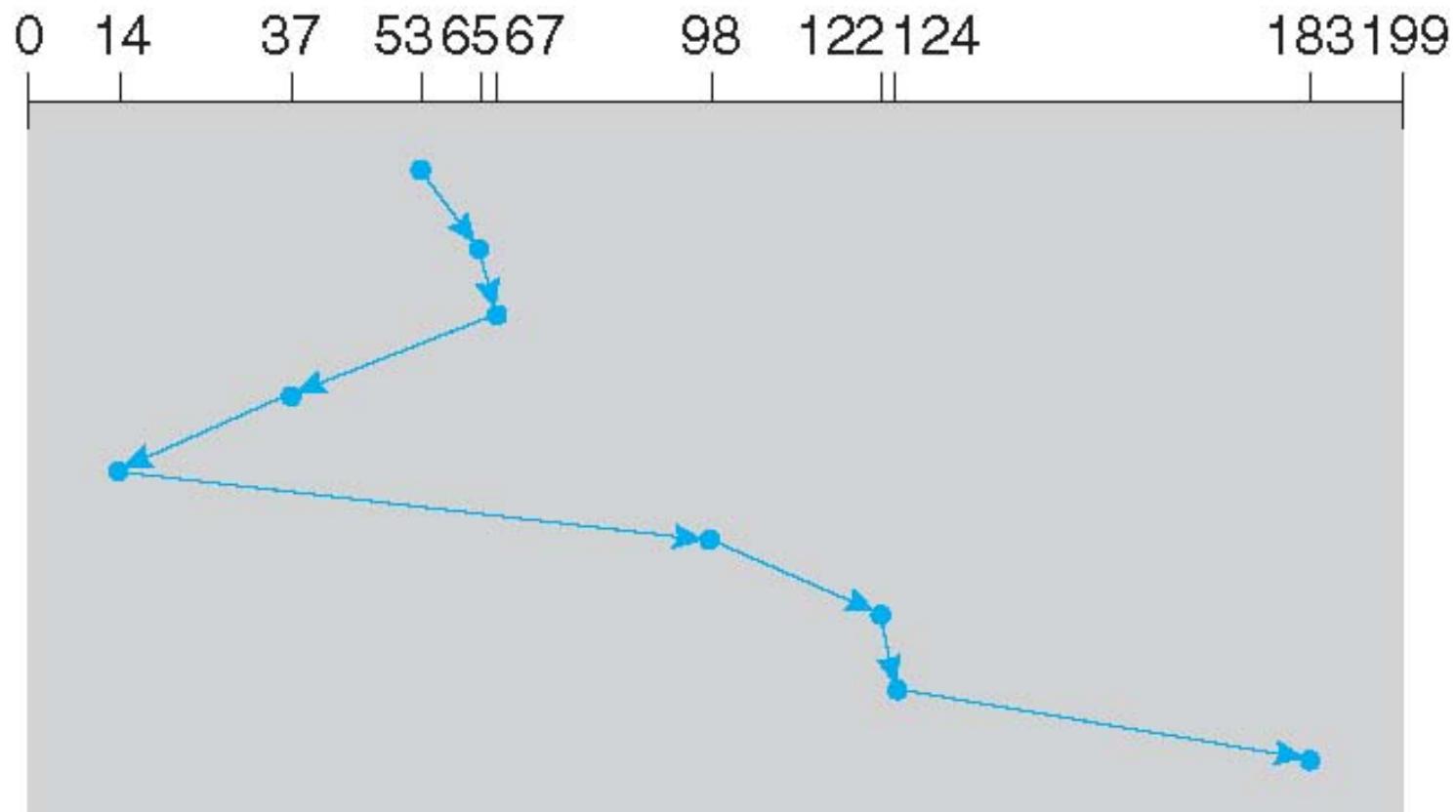
SSF

- ❑ Selects the request with the minimum seek time from the current head position
- ❑ SSF scheduling is a form of SJF scheduling; may cause starvation of some requests
- ❑ Illustration shows total head movement of 236 cylinders

SSF (Cont.)

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

head starts at 53



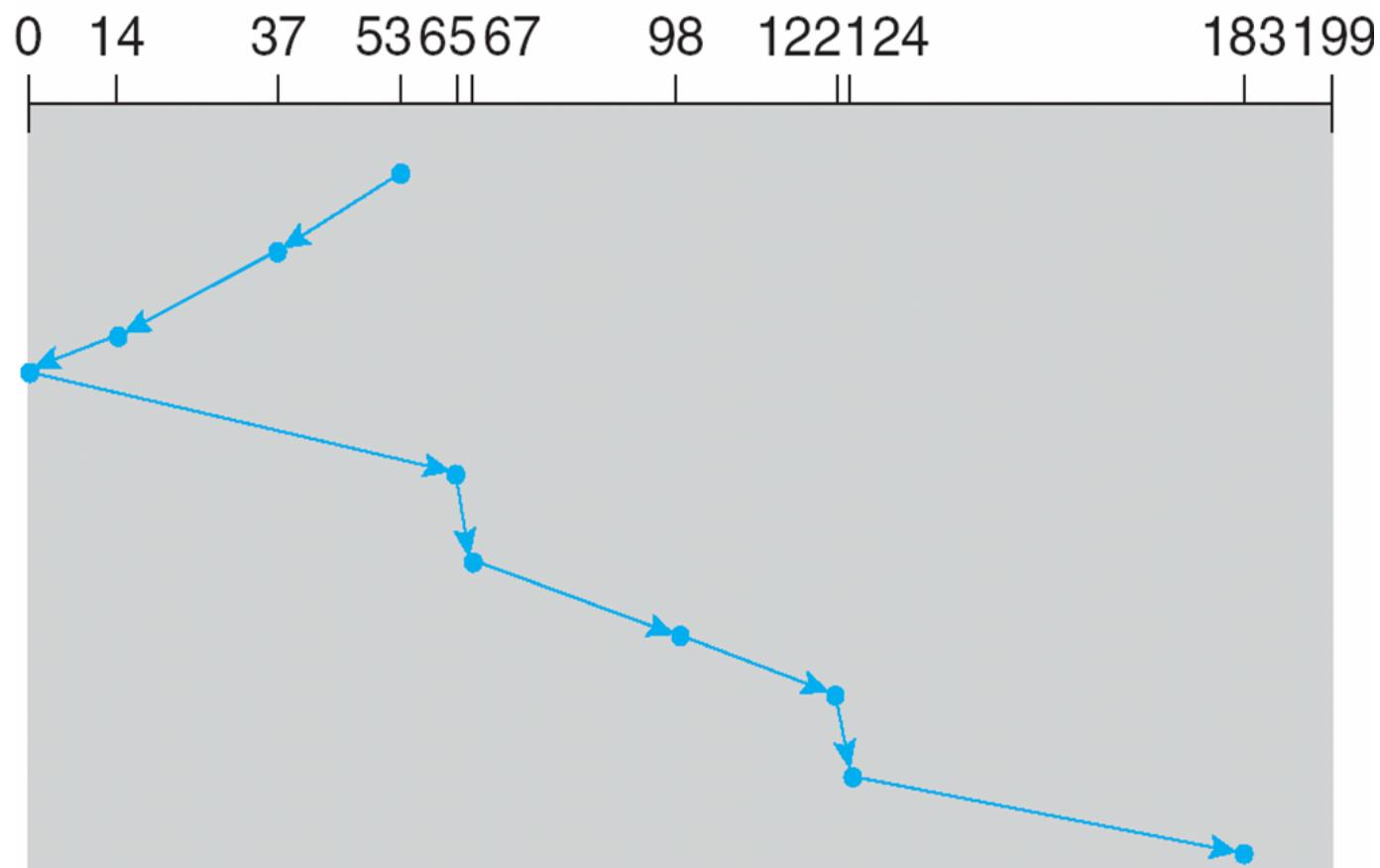
SCAN/Elevator

- ❑ The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- ❑ **SCAN algorithm** Sometimes called the **elevator algorithm**
- ❑ Illustration shows total head movement of 236 cylinders

SCAN (Cont.)

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

head starts at 53



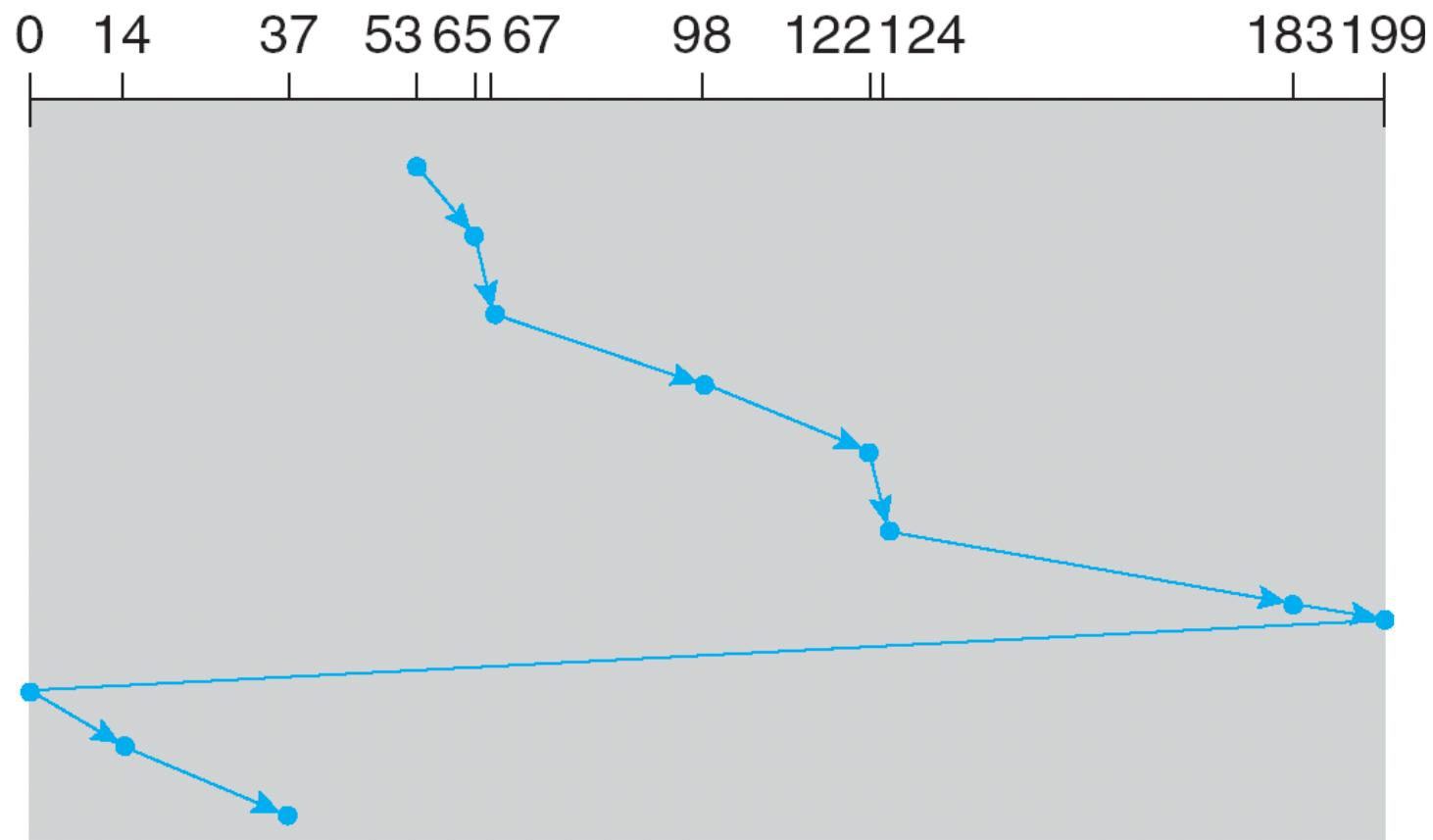
C-SCAN

- ❑ Provides a more uniform wait time than SCAN
- ❑ The head moves from one end of the disk to the other, servicing requests as it goes
 - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- ❑ Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN (Cont.)

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

head starts at 53



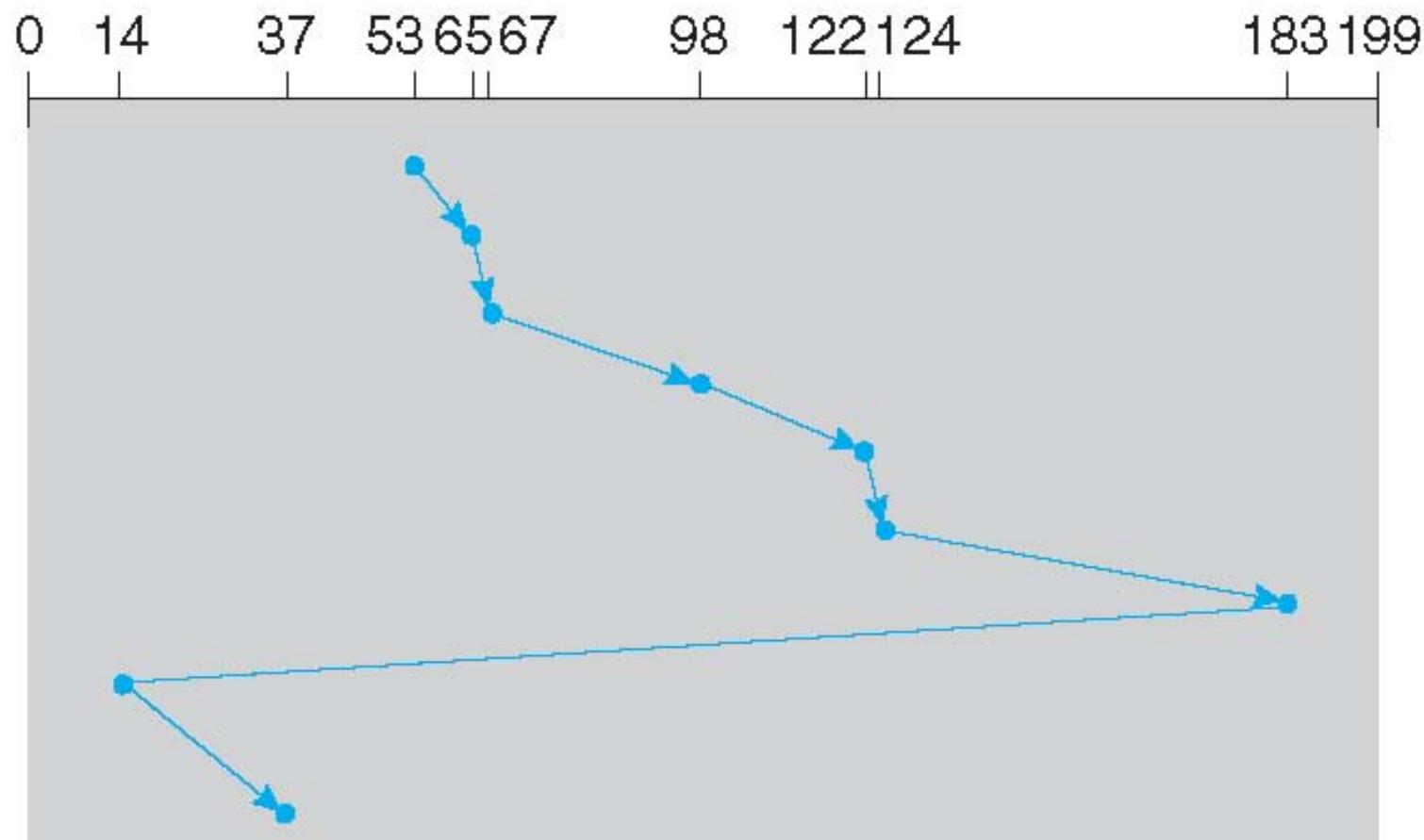
C-LOOK

- ❑ Version of C-SCAN
- ❑ Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

C-LOOK (Cont.)

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

head starts at 53



Selecting a Disk-Scheduling Algorithm

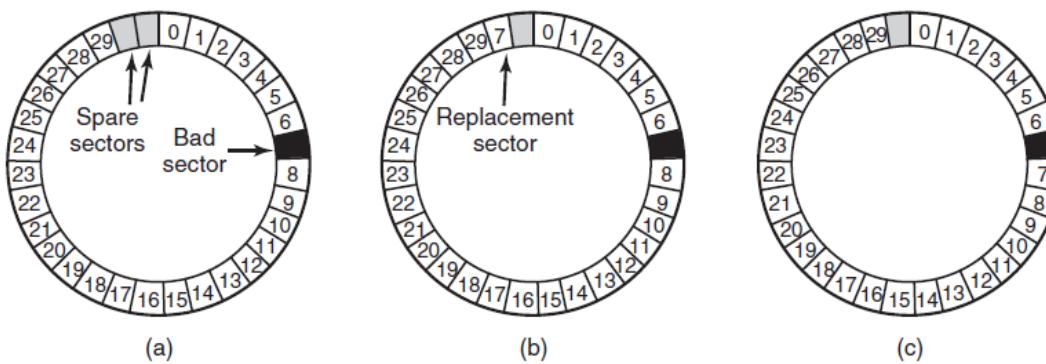
- ❑ SSTF is common and has a natural appeal
- ❑ SCAN and C-SCAN perform better for systems that place a heavy load on the disk
- ❑ Performance depends on the number and types of requests
- ❑ Requests for disk service can be influenced by the file-allocation method
- ❑ The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- ❑ Either SSTF or LOOK is a reasonable choice for the default algorithm

Disk Error Handling

- » Increasing linear bit densities constant drive by manufacturers
 - Defects introduced
- » Bad sector - sector that does not correctly read back the value written to them.
 - A few bits can be corrected by ECC
 - Larger defects can not be masked.
 - Deal with them in the controller or in the OS

Disk Error Handling

- » Before a drive is shipped it is tested and bad sectors are written onto the disk
 - Spares substituted
 - Substitute or shift



(a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.

Disk Error Handling

- » What about errors during runtime?
 - The first line of defense if ECC fails, try reading it again
 - Some read errors are transient: dust
 - Controller can switch to a spare before the sector dies completely

Disk Error Handling

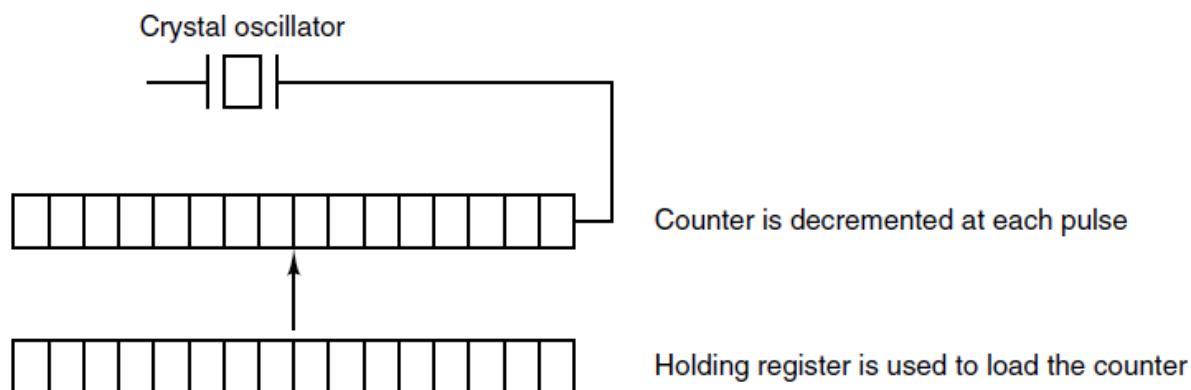
- » Option 2: Let the OS handle it
 - OS must acquire a list of bad sectors
 - Read them from the disk
 - Test entire disk
 - Can't let bad sectors reside in free block list or bitmap
 - Create a secret file
 - Backups can't backup bad block file
 - Sector by sector backups an issue

Disk Error Handling

- » Seek errors also an issue.
 - Mechanical problem with the arm
 - Modern controllers handle arm issues automatically
 - 80's and 90's floppy drives set error bit and let the driver handle it
 - Driver issued a recalibrate command to move arm as far out as it would go and reset the current cylinder to 0

Clocks

- » Essential to the operation of a system
- » Original clocks tied to 110 or 220 volt power and caused an interrupt every voltage cycle at 50 or 60 Hz
- » More recent built out of crystal oscillator, counter and holding register.



Clocks

- » Two modes
 - One shot
 - square-wave mode
- » Interrupts called clock ticks
- » 500 Mhz crystal will pulse every 2 nsec
 - 32 bit register means interval from 2 sec to 8.6 sec
- » Battery backup
- » UNIX / Windows Epoch

Clocks

- » Typical duties of a clock driver:
- » Maintaining the time of day.
- » Preventing processes from running longer than allowed.
- » Accounting for CPU usage.
- » Handling alarm system call from user processes.
- » Providing watchdog timers for parts of system itself.
- » Profiling, monitoring, statistics gathering.

Clocks

- » 32 bit at 60Hz will overflow in about over 2 years
 - How can we store ticks since Jan 1, 1970?
 - 1. 64 bit counter
 - 2. Store time in seconds rather than ticks
 - Use secondary counter for ticks this second
 - 136 years
 - 3. Calculate ticks since boot
 - On boot store current time in memory
 - Use ticks as offset.

Clocks

- » Processes can request clock notifications
- » If driver has enough clocks it can assign one to each process
 - Never enough clocks
 - Driver instead maintains a table with pending events
 - Store events as linked list indexed by time
- » Watchdog timers
 - Timer and procedure must be in the same address

Soft Timers

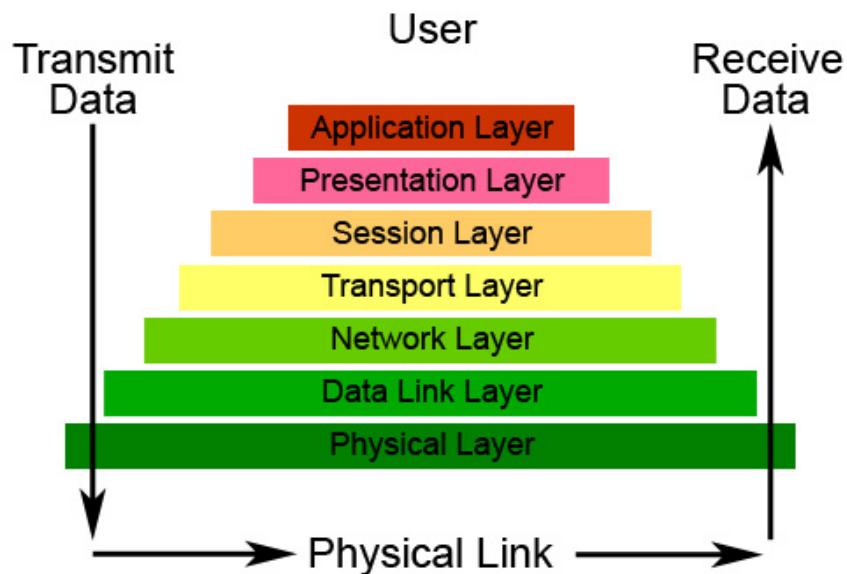
- » Interrupts have significant overhead.
 - Context switches
- » Polling has high latency
- » Soft timers avoid interrupts
 - Check real time mode every time before leaving kernel mode.
 - If timer expires then perform the event.
 - Already in kernel mode, no overhead

Soft Timers

- » Soft timers stand or fall with the rate at which kernel entries are made for other reasons. These reasons include:
 - System calls.
 - TLB misses.
 - Page faults.
 - I/O interrupts.
 - The CPU going idle.
- » Average varies from 2 to 18 usec. 10 usec

OSI Model

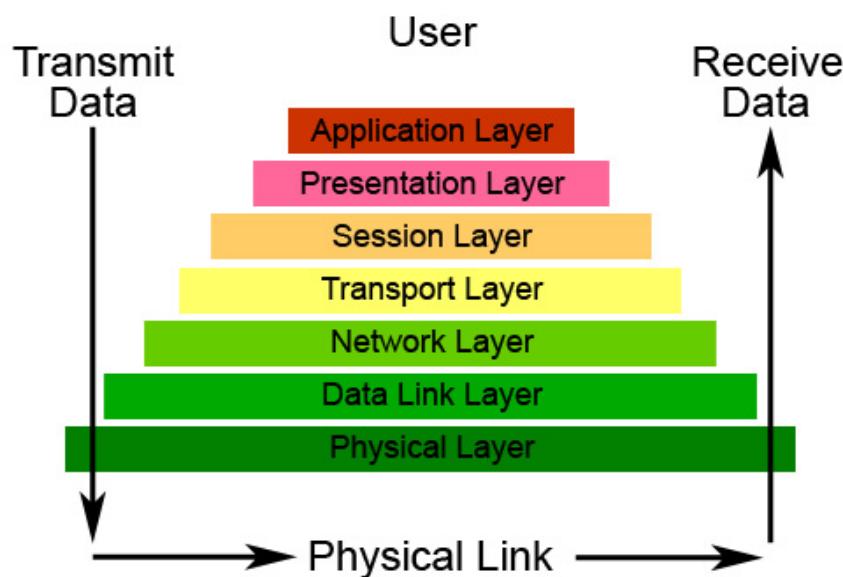
The Seven Layers of OSI



- Most widely known network layer model.
- Developed by the International Standards Organization (ISO).
- Abstract design that described no existing model

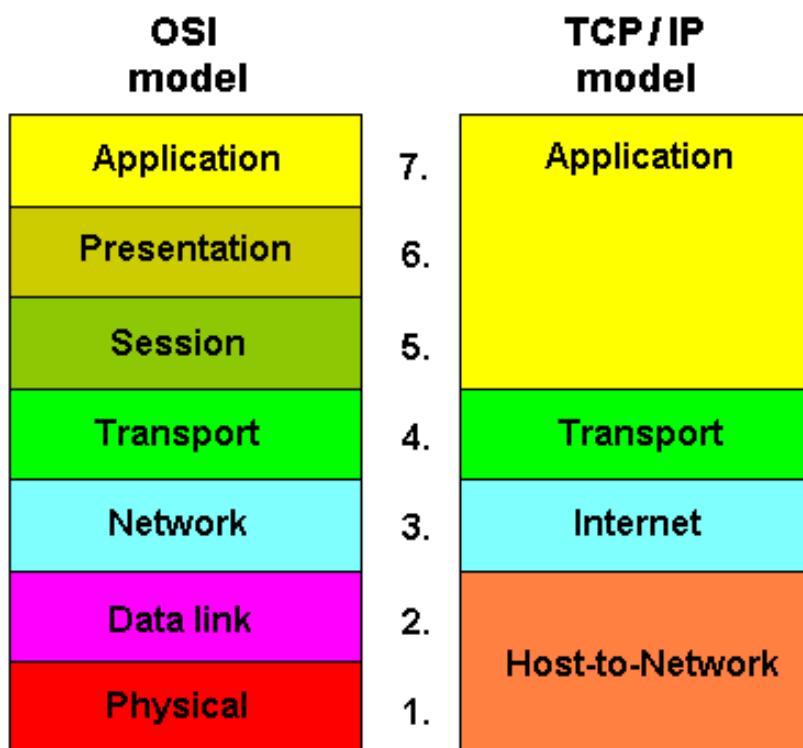
OSI Model

The Seven Layers of OSI



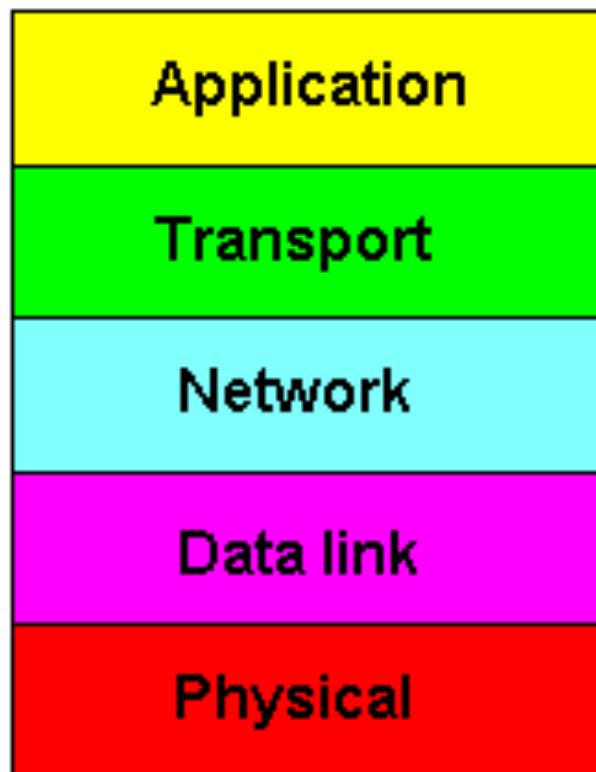
- At one point US government mandated all computer systems purchased by the government implemented this protocol.
- Never successful and finally abandoned.
- Problems: Presentation and Session layers are never implemented.

TCP/IP Model



- Another model was constructed that described a protocol already in existence, TCP/IP.
- Focused heavily on the upper layers(TCP and IP).
- Ignored the lower layers.
- Assumes hardware and drivers are commodities to be purchased.

Hybrid Model



- Our book uses a hybrid approach.
- Bottom two layers of the OSI
- Top three of the TCP

Physical Layer

- Defines:
 - The actual physical medium used for communicating.
 - The methods and techniques for getting information on and off the medium.
 - Medium may be wire, cable, optical fiber, electromagnetic signal.

Network Topologies

- The pattern of connections between the individual machines.
- First broad division is between local area networks (LAN) and wide area networks (WAN).
- WANs
 - Generally speaking are point to point.
 - No addressing needed since there is only one device to read them
 - Since no address no broadcast packet or multicast packet.
 - Frequently WAN links are full duplex so both of the hosts can transmit at the same time.

Network Topologies

- LANs
 - Typically broadcast communications
 - When two hosts are communicating with one another it's across a shared medium.
 - Devices communicating on the LAN have to share the medium.
 - Packets need addresses so broadcast and multicast can be used.

Network Topologies

- Switching blurs the distinction between LANs and WANs.
 - Individual devices are connected directly to ports on the switch.
 - Switch reads the destination from the header and forwards the data to the appropriate port
 - No sharing needed
 - Can run full duplex.

Network Topologies

- Linear
- Hierarchical
- Star
- Ring
- Partly connected mesh
- Fully connected mesh

Datalink Layer

- Responsible for accessing the shared medium.
- Concerned with packaging information into discrete packets and arbitrating access to the medium.
- Datalink layer devices such as switches or bridges eliminated the need for arbitration in modern networks.

Network layer

- Responsible for routing the information through complex multiple networks with differing physical layer techniques.

Transport Layers

- Creates a reliable connection between two network entities, though not all applications require a connection or a guarantee of reliability.

Transport Layer

- Previous to the Internet there were many different sets of network protocols.
 - Phenomenal success of the Internet changed things.
 - TCP/IP came to dominate the networking landscape.

Transport Layer

- In TCP/IP, IP is the network protocol
- TCP, [transmission control protocol](#) is one of the two principle transport layer protocols.
- UDP, [user datagram protocol](#), is the other.
 - Unreliable datagram
 - TCP provides “[connection-oriented, reliable](#)”, communication.

Application Layer

- The entity on one system interacting with the entity on another system.
- Each application will use a specific protocol.
 - Many widely used and have been assigned port numbers.
 - port numbers used by the transport layer to determine which application should receive the incoming data.
 - [well-known port number](#) - assigned by the IETF to only be used by that application. Valid range 0 - 1023
 - Non well known are 1024-49150
 - 49151 - 65535 - Used for dynamic purposes.

Layers Implementation

- In a networked device there is an entity at each layer that is responsible for the functions of that layer.
- At the physical it's hardware.
- Datalink may be hardware as well.
- Most devices have software for all the other layers.

Layers: Pros and Cons

- Pro:
 - Small modules are easy to understand, develop, and debug.
 - Can be replaced with newer improved modules.
 - Organizations can specialize in developing different layers.
- Extremely important we have very good definitions of the interfaces, because:
- Con:
 - Many standards

IP Addressing and Routing

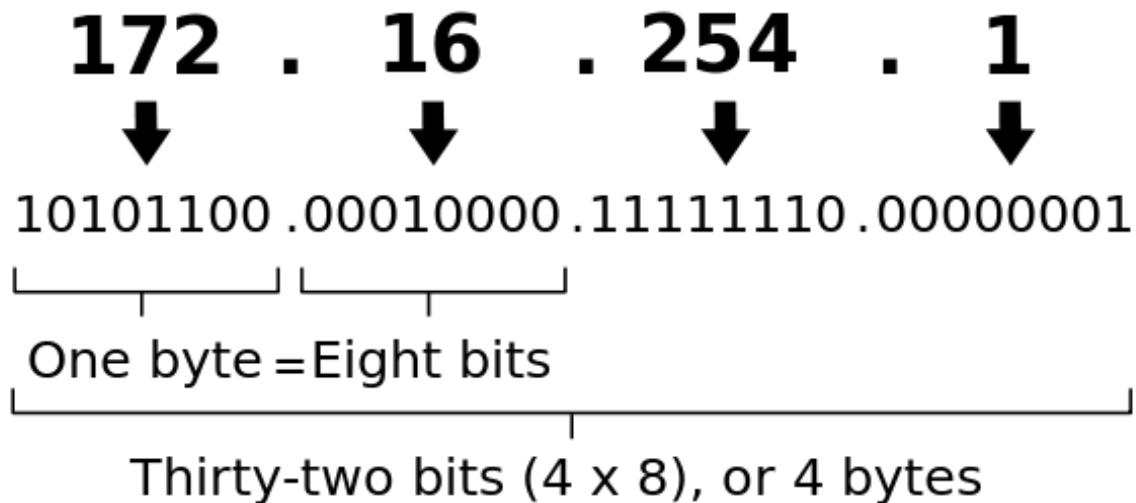
- IP Address - a numerical label assigned to each device
- An IP address serves two principal functions: host or network interface identification and location addressing.
- The designers of the Internet Protocol defined an IP address as a 32-bit number
 - Internet Protocol Version 4 (IPv4),
 - Due to the enormous growth of the Internet a new version of IP (IPv6), using 128 bits for the address, was developed in 1995.

IP Address

- IPv4 address consists of 32 bits which limits the address space to 4294967296 (2^{32}) possible unique addresses.
 - IPv4 reserves some addresses for special purposes such as private networks (~18 million addresses) or multicast addresses (~270 million addresses).
- IPv4 addresses are represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 192.168.20.1. Each part represents a group of 8 bits (octet) of the address.

IP Address

An IPv4 address (dotted-decimal notation)



Decomposition of an IPv4 address from dot-decimal notation to its binary value.

IPv6 Addresses

- The rapid exhaustion of IPv4 address space, prompted the Internet Engineering Task Force (IETF) to expand the addressing capability in the Internet.
 - Address size was increased from 32 to 128 bits or 16 octets.
 - Provides the potential for a maximum of 2^{128} , or about 3.403×10^{38} addresses.

IPv6 Addresses

An IPv6 address (in hexadecimal)

2001:0DB8:AC10:FE01:0000:0000:0000:0000

2001:0DB8:AC10:FE01:: Zeroes can be omitted

10000000000001:0000110110111000:1010110000010000:1111111000000001:
0000000000000000:0000000000000000:0000000000000000:0000000000000000

Routing

- Routers are responsible for delivering IP packets from the source device to the destination device.
 - Routing protocol used between two routers depends on their administrative relationship.
 - Different routing groups: distance vector and link state.

Routing

- Distance Vector
 - RIP (routing information protocol)
 - IGRP (interior gateway routing protocol)
 - RIP2 (routing information protocol version 2)
 - EIGRP (enhanced interior gateway routing protocol)
 - BGP (border gateway protocol)

Routing

- Link State
 - OSPF (open shortest path first)

Routing

- Routers connected in a **partial mesh topology**.
 - Loss of a link will not **partition** the network into pieces that can't communicate.
 - Earlier days of the internet the term **gateway** referred to the class of device we now call a router.
 - default gateway = default router

DHCP

- Devices can be configured with a predetermined IP address, **static**
- DHCP (dynamic host configuration protocol)
 - DHCP servers configured with a range of IP addresses.
 - Host that is turned on will broadcast a message looking for the DHCP server.
 - DHCP server will tell the host which IP address to use.
 - Address is **leased** for some period of time after which it must be renewed.

Name Resolution

- Remembering IP addresses is hard.
- DNS (domain name service) is a protocol that translates from a user-friendly name to IP address
- Fully qualified name - www.uta.edu
 - Parts between the periods known as domains.

Domains

- Domains are organized into a tree structure.
- Top Level Domain (TLD) such as .com, delegated to specific organizations by the Internet Corporation for Assigned Names and Numbers (ICANN), which operates the Internet Assigned Numbers Authority (IANA),

Domains

- Originally, the top-level domain space was organized into three main groups: Countries, Categories, and Multiorganizations.
- IANA today distinguishes the following groups of top-level domains:
- Country-code top-level domains: Two letter domains established for countries or territories
- Internationalized country code top-level domains: ccTLDs in non-Latin character sets (e.g., Arabic or Chinese).
- Generic top-level domains (gTLD): Top-level domains with three or more characters

Datalink Layer

- LANs originally had a unique characteristic.
 - Data transmitted in such a way that all the hosts connected to the same link will see every transmission.
 - “multiaccess network”
- Need mechanism to allow hosts to share access. Only read what they should.
 - Media Access Control (MAC)

MAC address

- The original IEEE 802 MAC address comes from the original Xerox Ethernet addressing scheme. This 48-bit address space contains potentially 2^{48} or 281,474,976,710,656 possible MAC addresses.
- Every host is connected to the LAN via [network interface card \(NIC\)](#). Each NIC has a 6 byte identifier
 - Upper three bytes identify the manufacturer
 - Lower three bytes identify the card, uniquely

ARP

- Address resolution protocol (ARP) used to map from IP address to MAC.
 - Host looking for a server will make a broadcast request.
 - All hosts will receive and look into ARP table

Ethernet

- Ethernet relies on the probability that most of the time the network is not busy.
 - If busy, then the sender would wait until it was free and then transmit.
 - If two transmitted at the same time a **collision** would occur.