Example File Systems

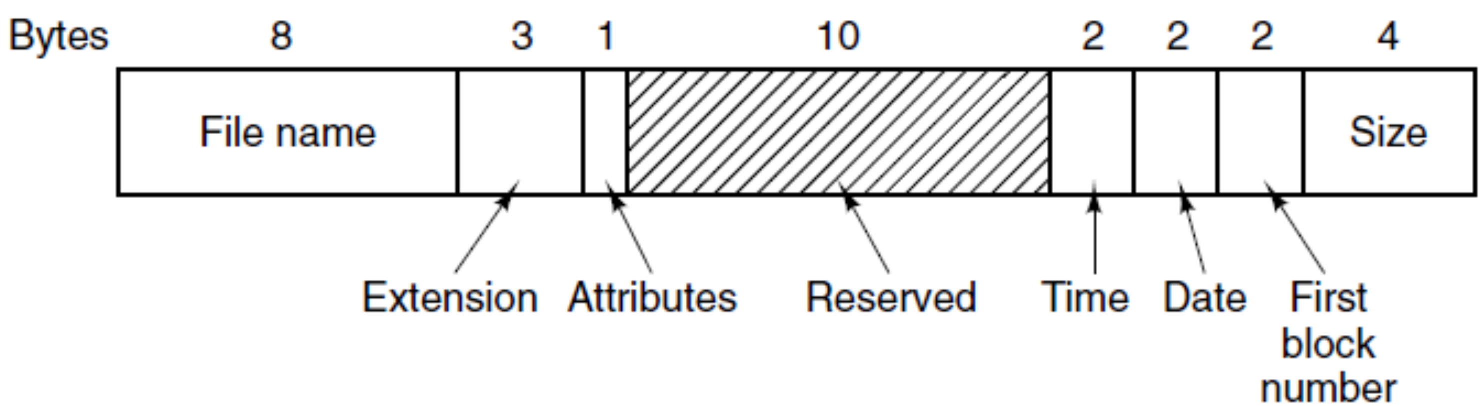
Chapter 13

MS-DOS File System

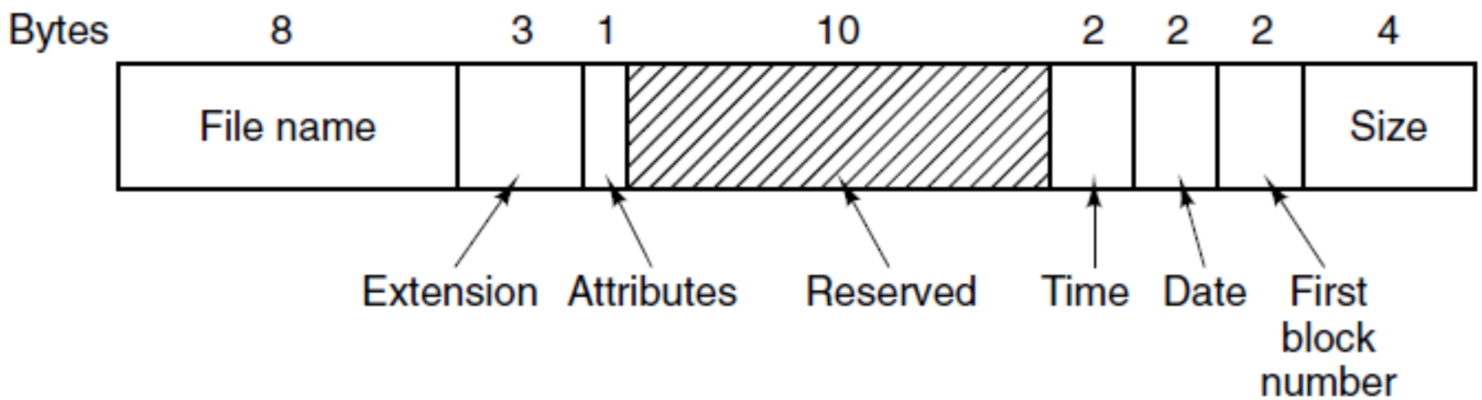
- First file system for IBM PCs.
- Main file system up through Windows 98 and Windows ME
  - Still supported in Windows 2000, Windows XP, Windows Vista.
- Extension (FAT-32) still widely used in digital cameras, mp3 players, thumb drives

MS-DOS File System

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

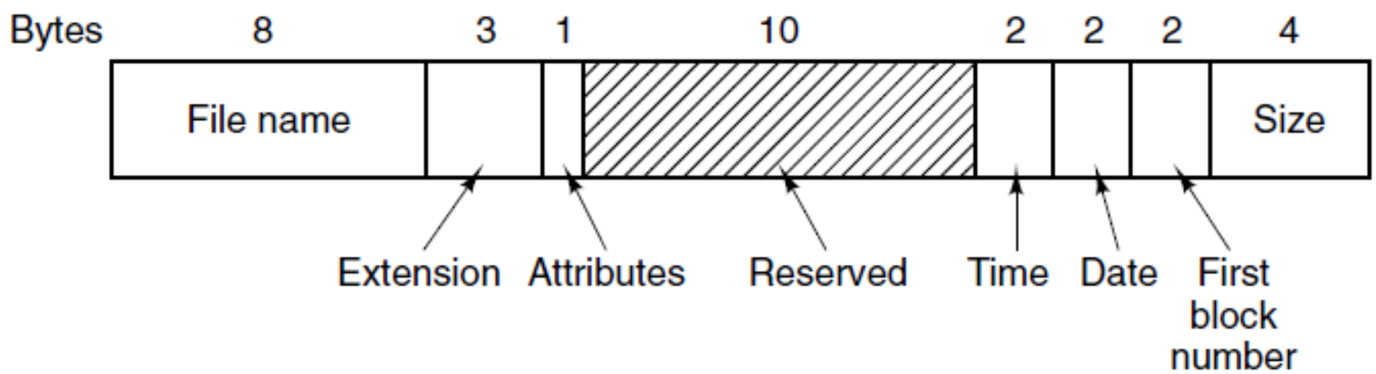


MS-DOS File System



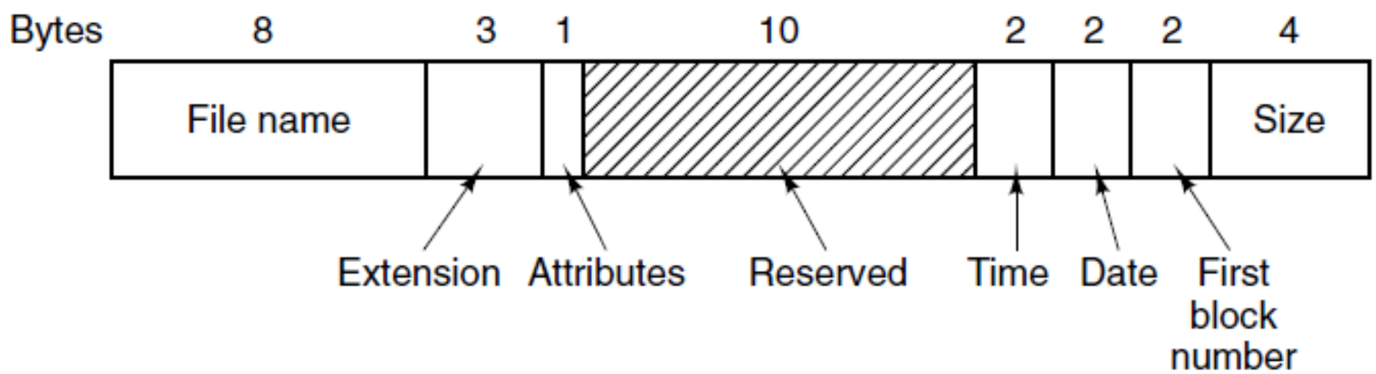
- » File names max size is 8+3, left justified and padded with 0 on the right if smaller.
- » Attributes for read/write/hidden/archive/ system.

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.

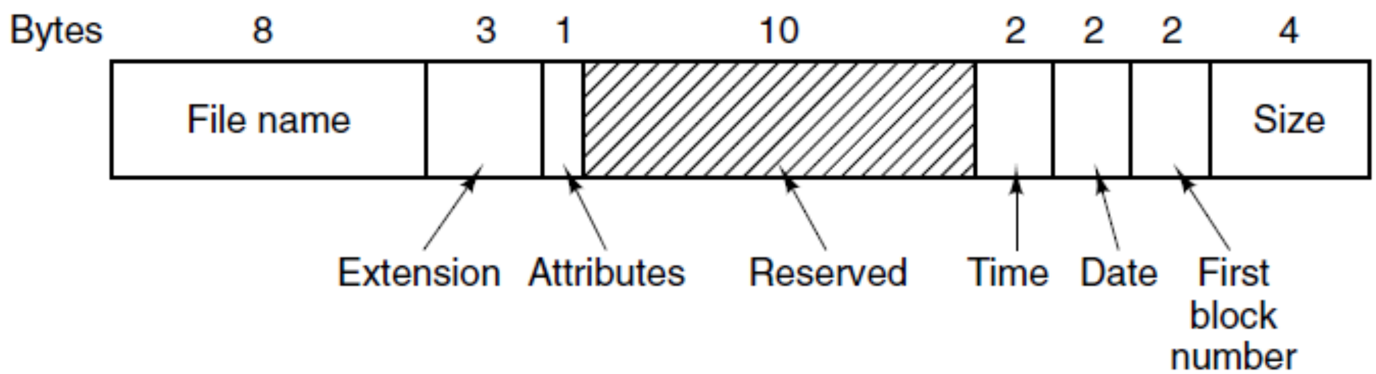
» Day (5bits) , Month (4bits), Year (7bits)

» Year expressed in years since 1980.

» Max year is 2107.

» Y2108 problem.

MS-DOS File System

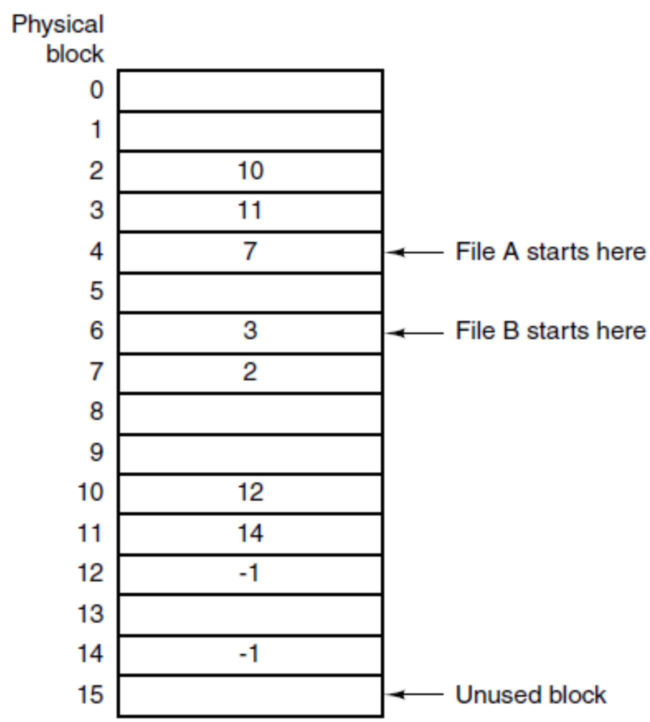


- Size is stored as 32-bit number. Theoretical max file size of 4 GB
  - Other limitations limit this to 2 GB.
  - 10 reserved bits unused.

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)

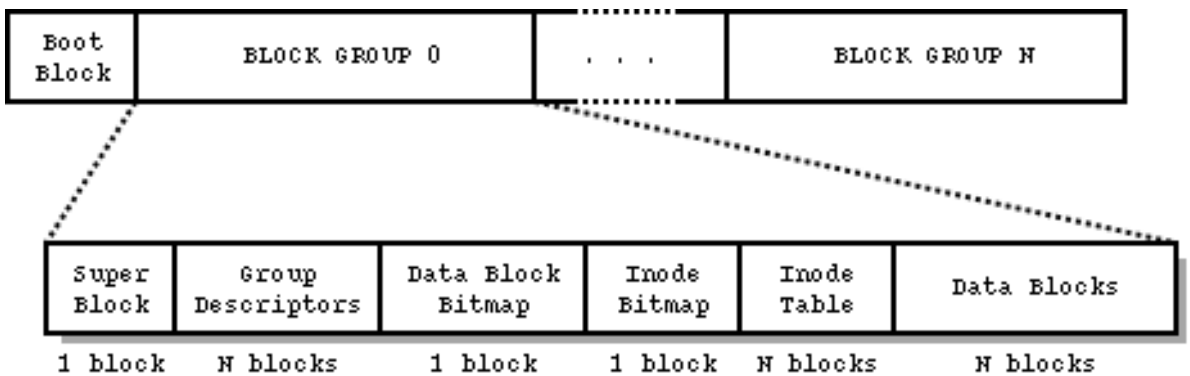
MS-DOS File System

- Second release of Windows 95 introduced the FAT-32 file system with 28-bit disk addresses.
- Theoretical the partitions could be 2^{28} x 2^{15} bytes
- Actual limit is 2TB (2048 GB)
  - Internally tracks partition sizes with a 32 bit number. 2^{9} x $2^{32} = 2$ 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.
- Theoretical the partitions could be 2^{28} x 2^{15} bytes
- Actual limit is 2TB (2048 GB)
  - Internally tracks partition sizes with a 32 bit number. 2^9 x $2^{32} = 2$ 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 */
   ...
```

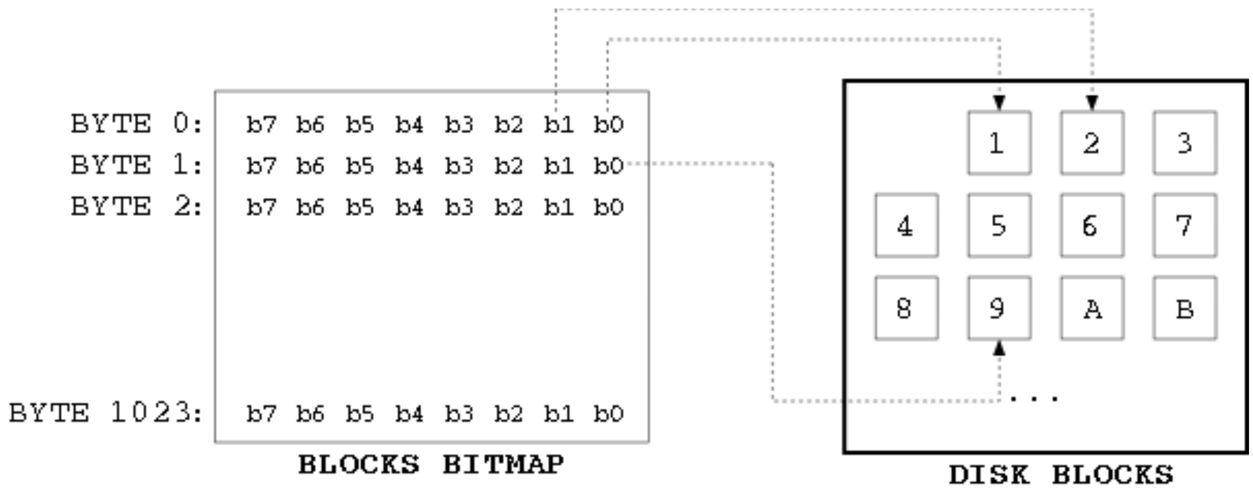
Group Descriptors

- 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

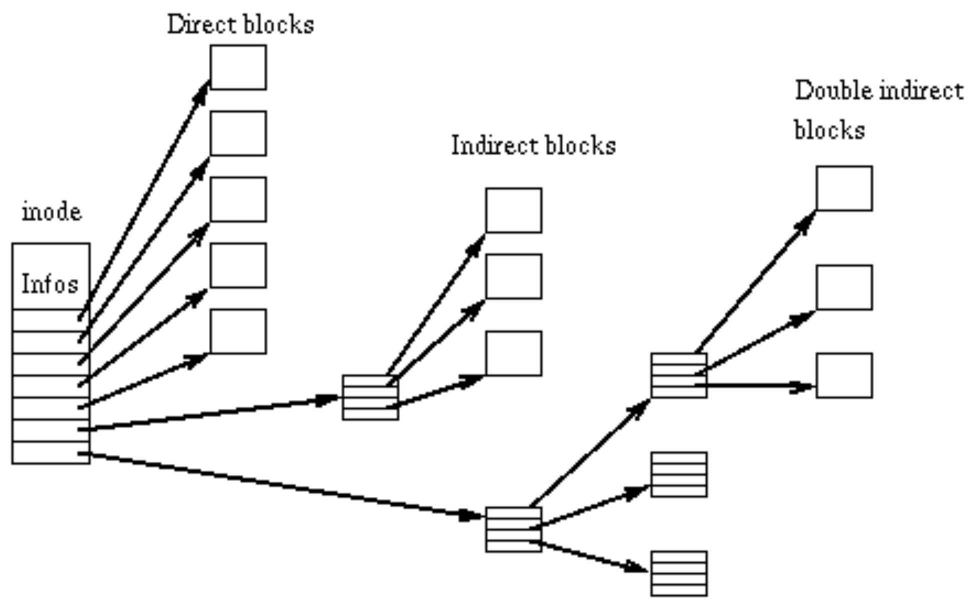
- The inode table consists of a series of consecutive blocks, each of which contains a predefined number of inodes.

- 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

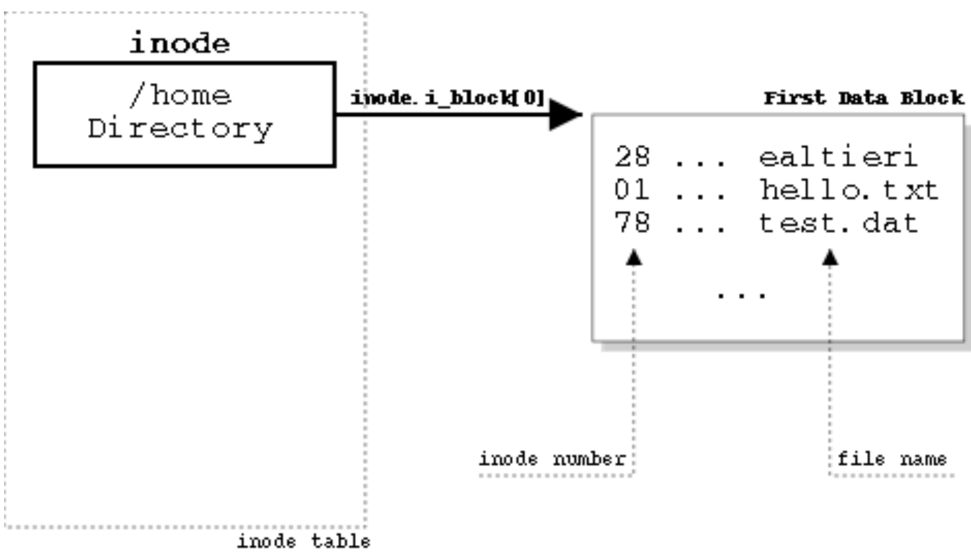


Quote from the Linux kernel documentation for ext2:
"There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect 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.

-

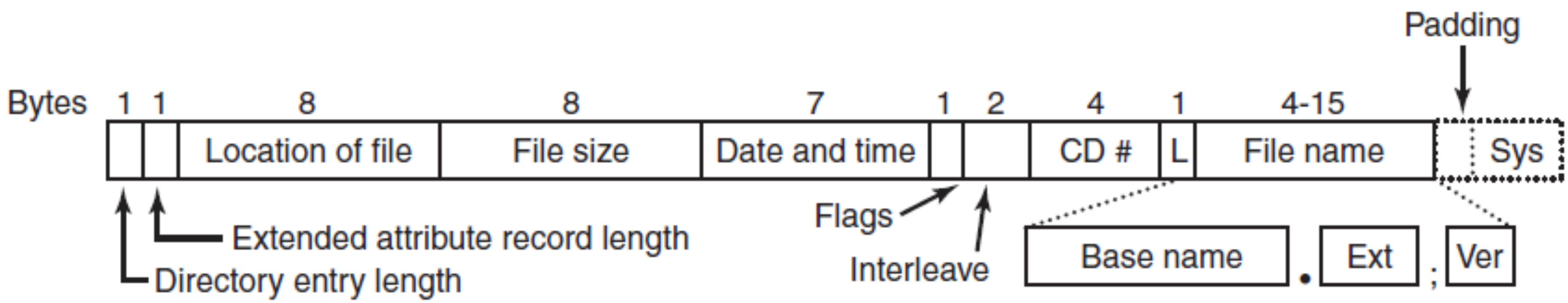
ISO 9660 File System

- 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

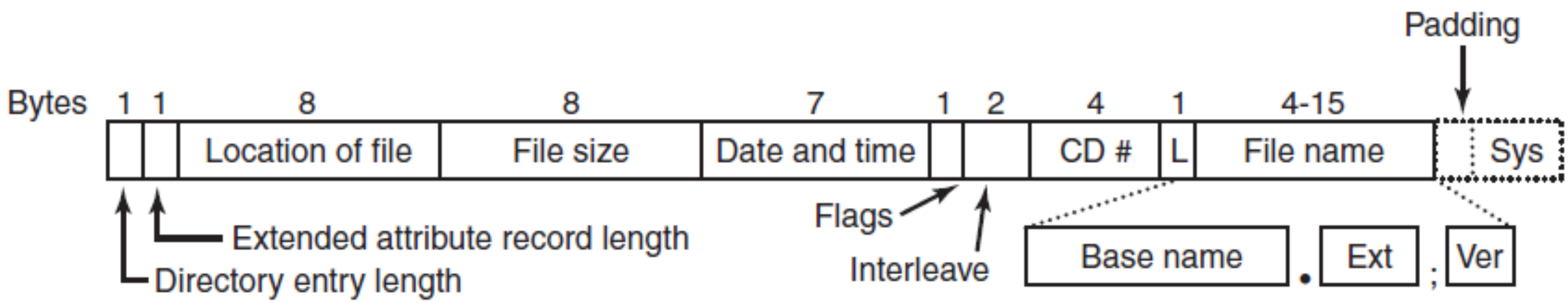
- Also contains the name of three files which contain:
  - Abstract
  - Copyright Notice
  - Bibliographic Information
- Also holds block size (normally 2048 but 4KB, 8KB and higher powers of 2 can be supported.)
- Contains directory entry for the root directory.

ISO 9660 File System



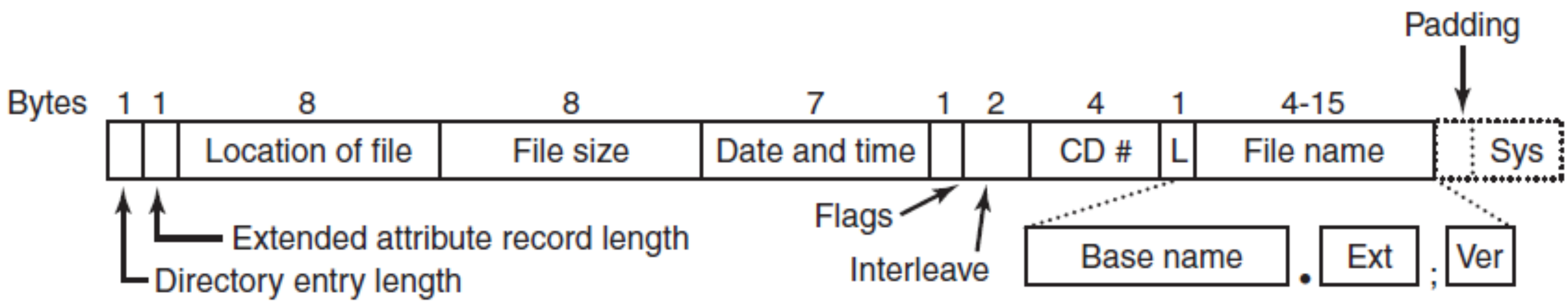
- Variable number of entries
  - Last contains marker signifying end
  - Each entry encoded in ASCII or binary.
    - Binary encoded twice. Little endien and big endien

ISO 9660 File System



- Directory entires 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 bt files do not have to be contiguous.

File-System Management and Optimization

Disk-Space Management

- Management of disk space if 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

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:

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

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

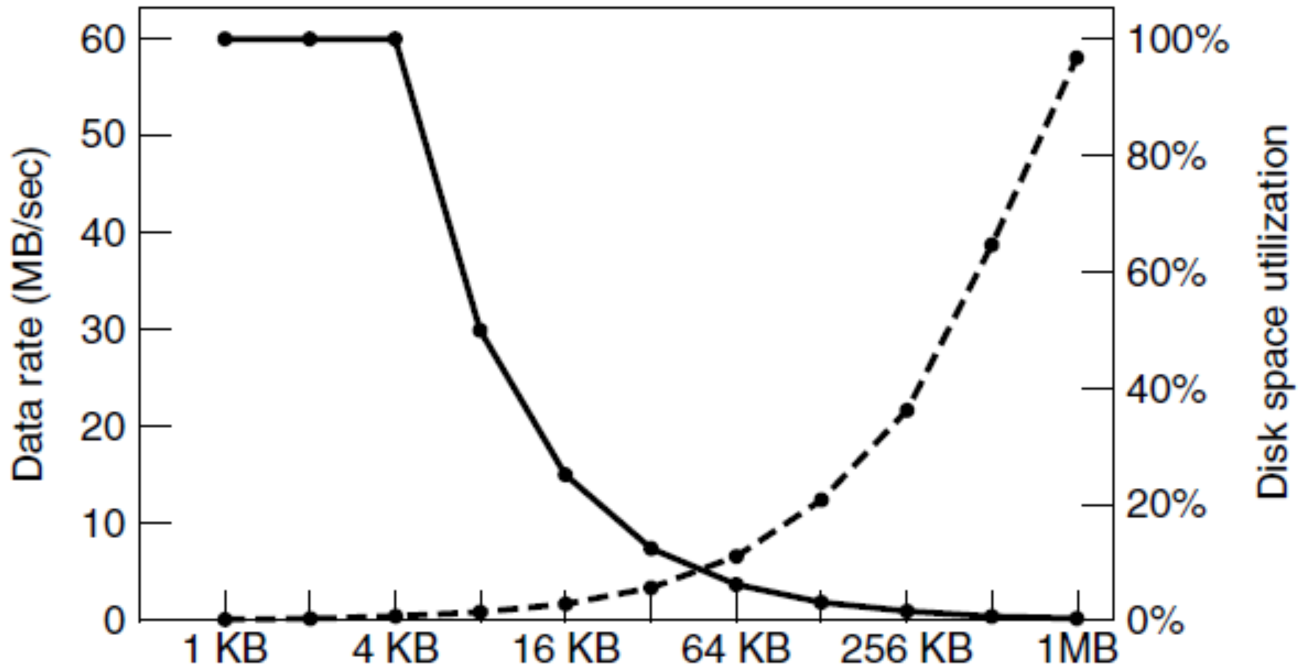
Block Size

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

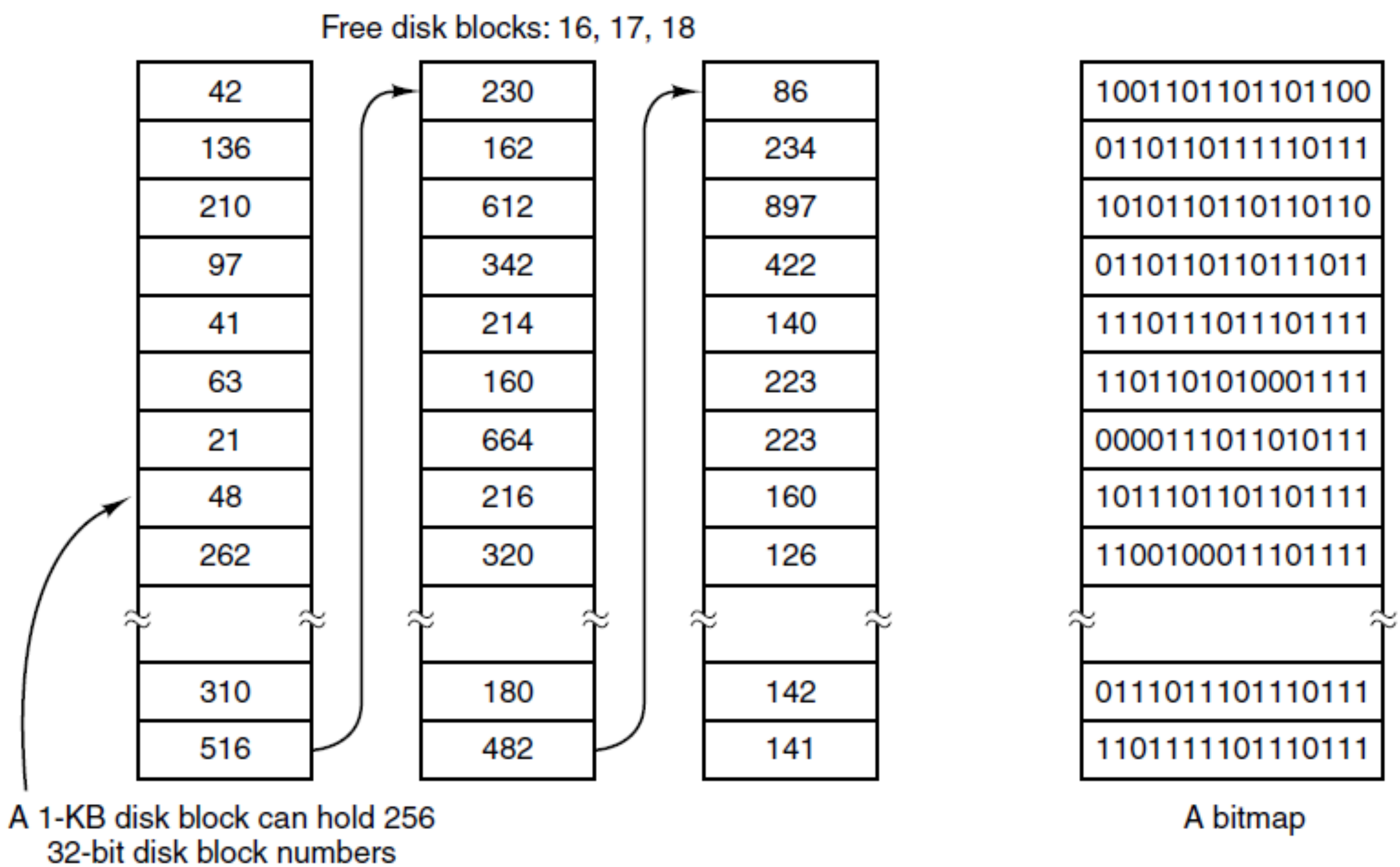
Percentage of files smaller than a given size (in bytes).

Block Size



The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Free Blocks



(a)

(b)

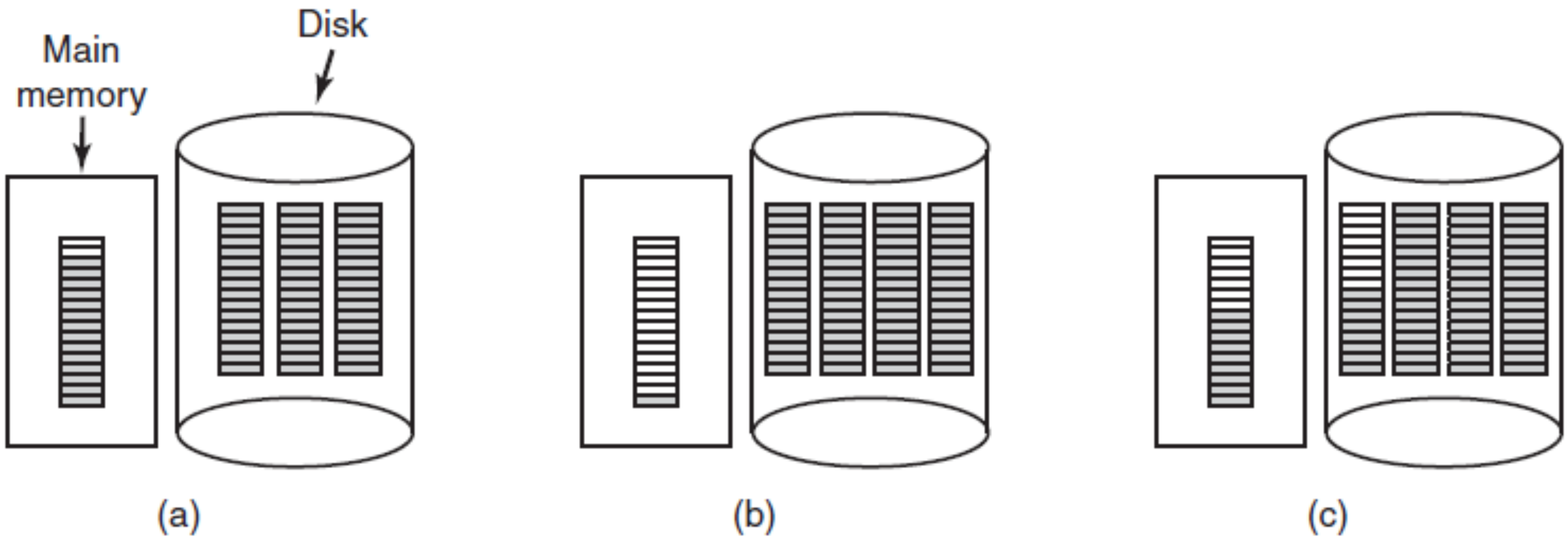
Free Blocks

- Two methods used
  - Linked list of disk blocks
  - Like inode, use a disk block to store it
    - 1-KB block, 32 bit block numbers means 255 blocks per list
    - 1 TB (1 billion disks blocks) needs 4 million blocks to track
    - Since free blocks used to store free block list: essentially free storage.

Free Blocks

- Only a single block of pointers needs to be kept in main memory.
- When a file is created the needed blocks are taken from a block of pointers.
  - Read a new block when the in memory pointers run low
- File deleted? Return pointers to memory.
- When the block of pointers is full, write it to disk

Free Blocks



(a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk.

(b) Result of freeing a three-block file.

(c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

Free Blocks

- Other method is a bitmap
  - Disk with n block represented by a bitmap with n bits
  - 1-TB disk needs 1 billions bits, around 130,000 1-KB blocks
- Less space than linked list until disk nearly full.

Free Blocks

- Linked list optimization
  - Track runs of blocks rather than individual blocks
    - Best case an empty disk represented by two numbers: first free block and size.

Disk Size / Free Blocks

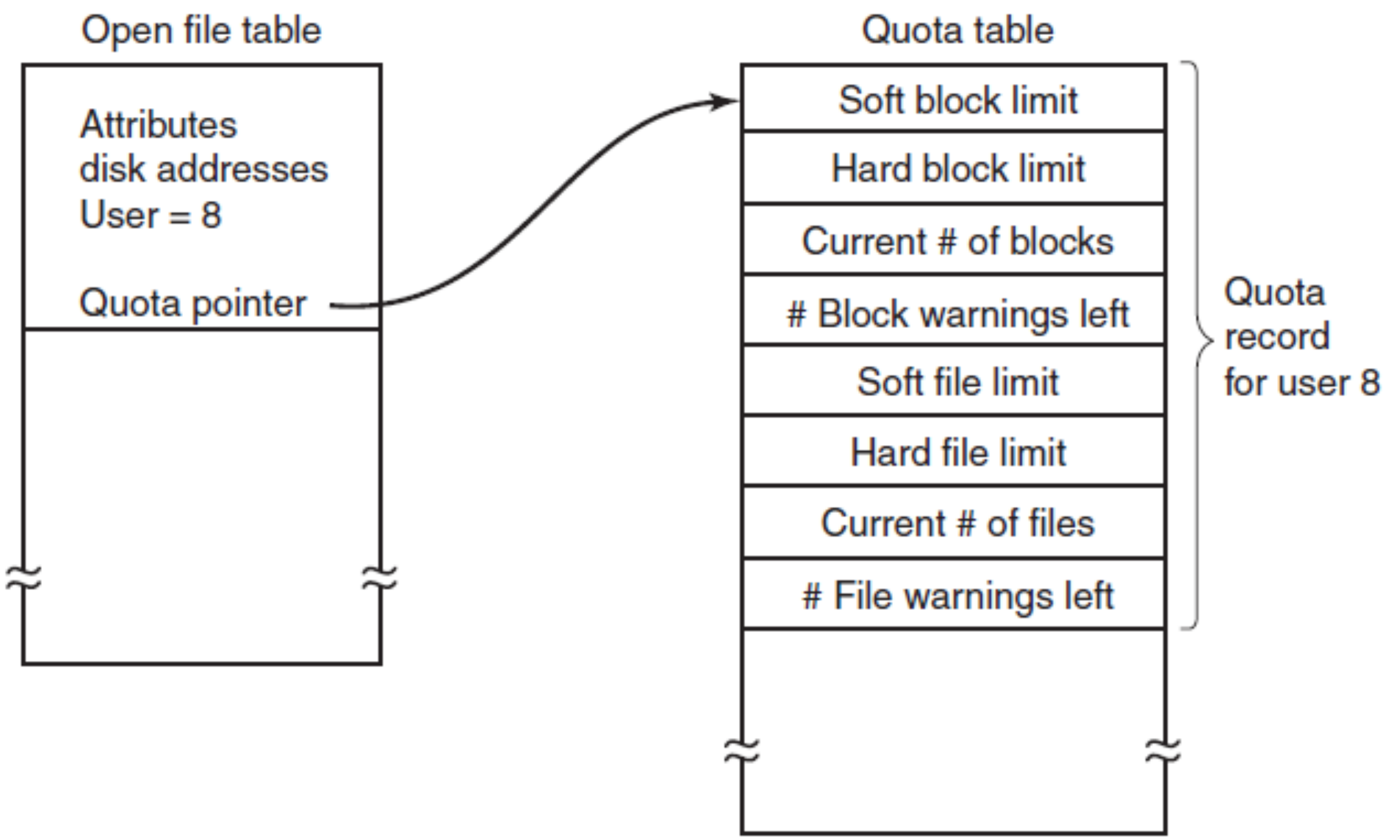
- Deciding optimal solution requires data the OS designers won't have until the system is deployed and heavily used.
- Extrapolating from even a large set of computers would not give a reasonable answer.

Disk Quotas

» Each user gets a maximum allotment of files and blocks.

```
[tbakker@omega ~]$ quota
Disk quotas for user tbakker (uid 419371): none
```

Disk Quotas



» Quota record table for every user.

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 ti 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.

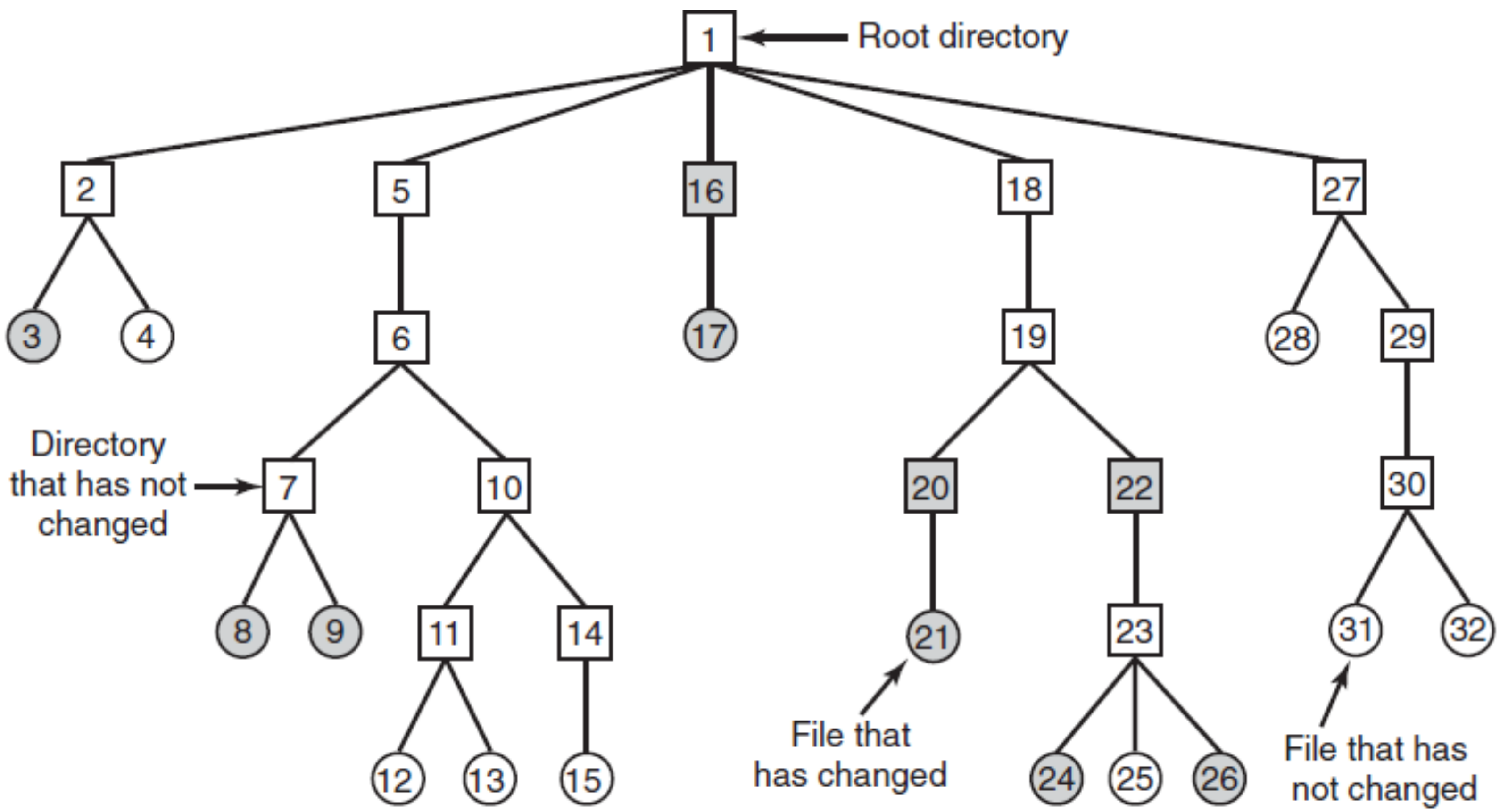
File System Backups

- 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.

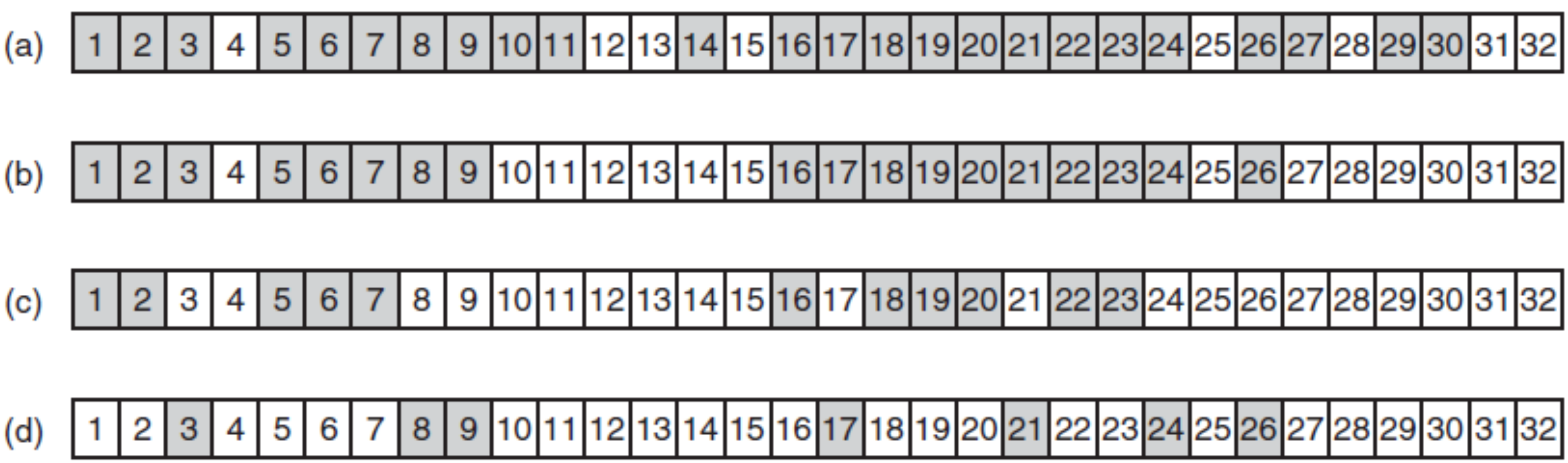
File-System Backups



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.

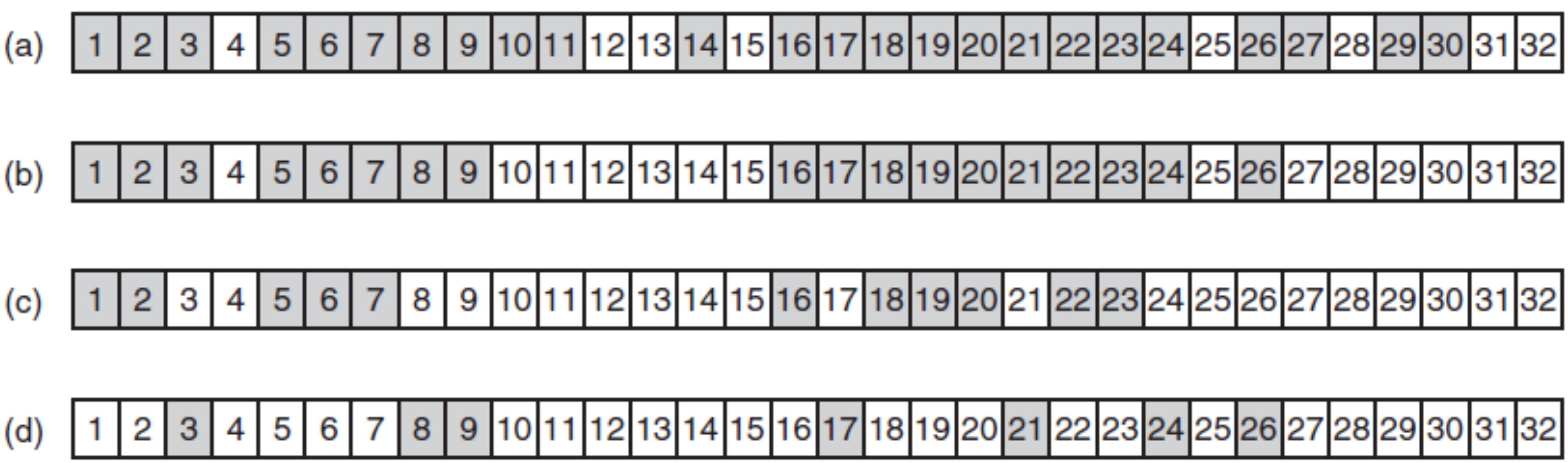
File-System Backups

- 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



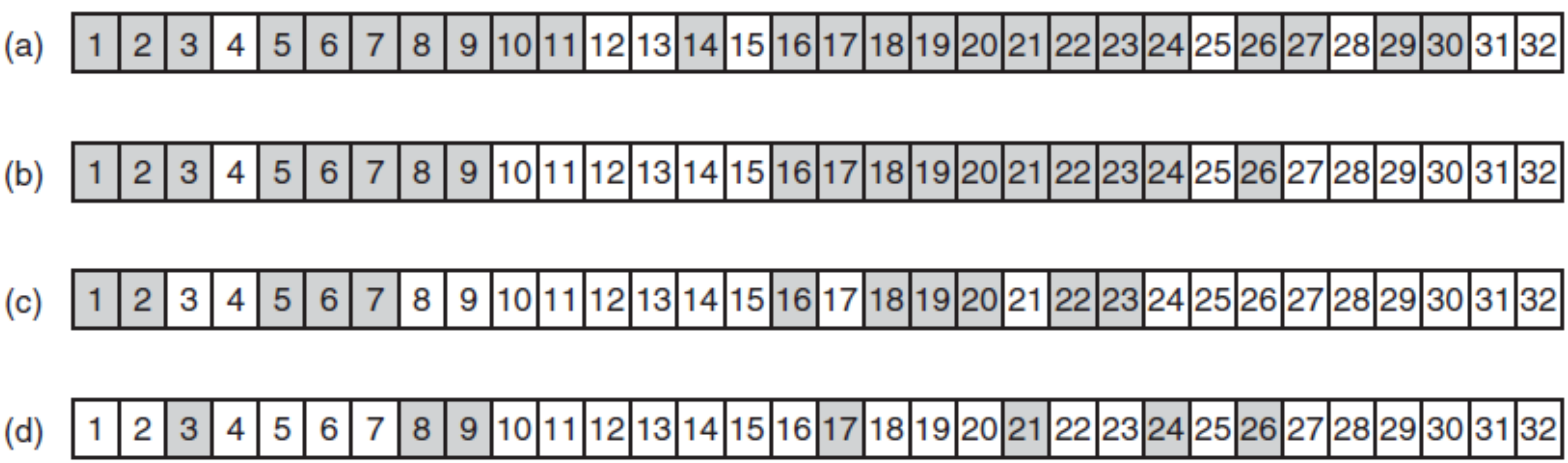
File-System Backups

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



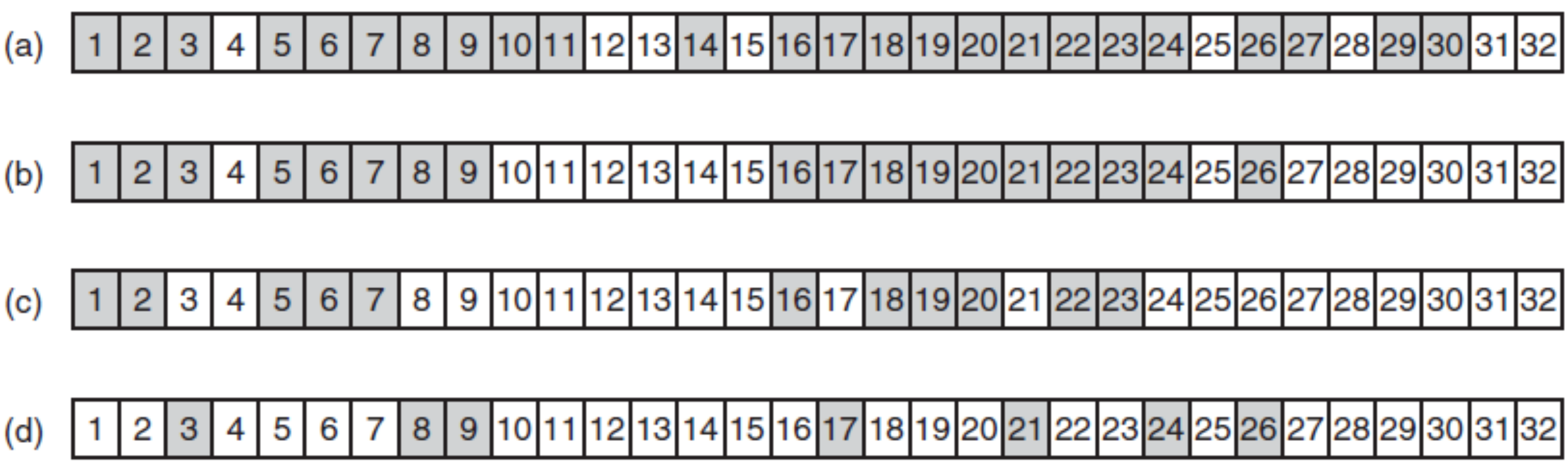
File-System Backups

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



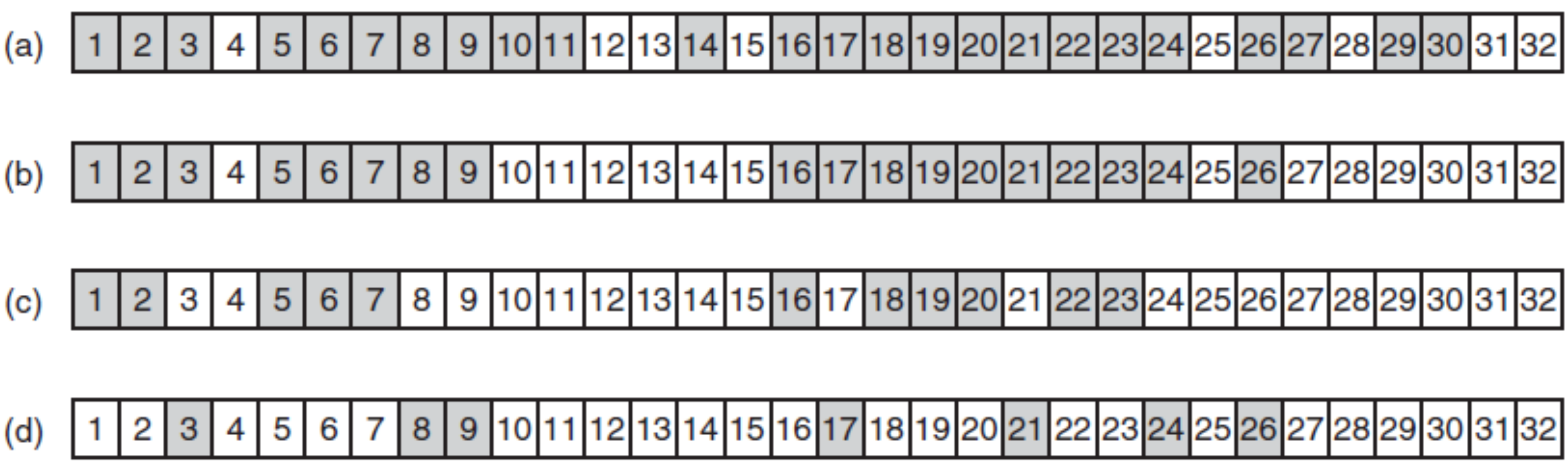
File-System Backups

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



File-System Backups

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



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 form scratch after all the dumps has 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
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	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
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1

(b)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Free blocks	0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1

(c)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Blocks in use	1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
Free blocks	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

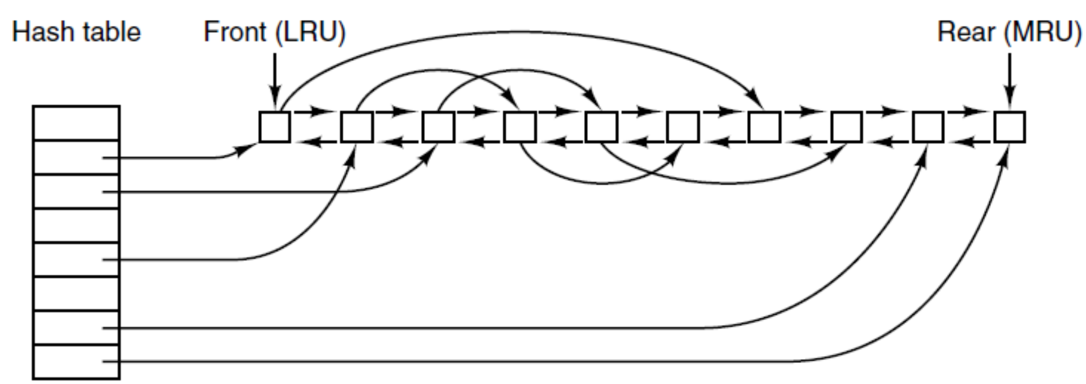
- If the link count is too low, a file may be deleted from a directory and the inode removed even though it is still referenced.
  - Force the link count in the inode to match the directory entries.

File-System Performance

- Reading a single word memory access is a million times faster than disk access.
- Three techniques we will cover to improve performance:
  - Caching
  - Block Read Ahead
  - Reducing Disk-Arm Motion

File-System Performance

- Caching
  - Block cache or Buffer cache.
  - Upon request check cache, if in cache return otherwise hit the disk.
  - Thousands of blocks in the cache.
    - Hash the device and disk address and lookup if it's in the cache.



File-System Performance

- If you read a block into cache you have to evict a block.
- Block references are relatively infrequent so an LRU implementation using a linked list is feasible.
  - Except: LRU is nor desirable

File-System Performance

- If a critical block, such as an inode block is read into cache and modified, but not rewritten to disk a crash will leave the FS in an inconsistent state.
  - Placed at the end of the LRU chain, it may be a while before it's evicted and written to disk.

File-System Performance

- Modified LRU
  - Is the block likely to be needed again soon?
    - End of list so they hang around in cache longer
  - Is the block essential to the consistency of the file system?
    - Front of list so they get removed and written to disk sooner.

File-System Performance

- Still undesirable to keep blocks in cache for too long.
  - Unix system call sync forces all modified blocks to disk.
    - *update* (pdflush on centos) runs in the background in an endless loop calling sync every 30s.
    - Windows system call FlushFileBuffers
      - Used to use a **write-through-cache**

File-System Performance

- Block Read Ahead
  - Get the blocks in cache before they are needed.
    - When request for block k is made, also check for k+1 in cache. If not there, read it in.
  - Does not work for random access and hurts performance by tying up resource

File-System Performance

- Reduce disk arm motion
- Keep disk storage in terms of groups of block rather than blocks
- File accesses, even for small files require two access: inode and data.
  - Allocate each cylinder with its own inodes.
  - Try to keep blocks in the same cylinder.

File-System Performance

» Reduce disk arm motion