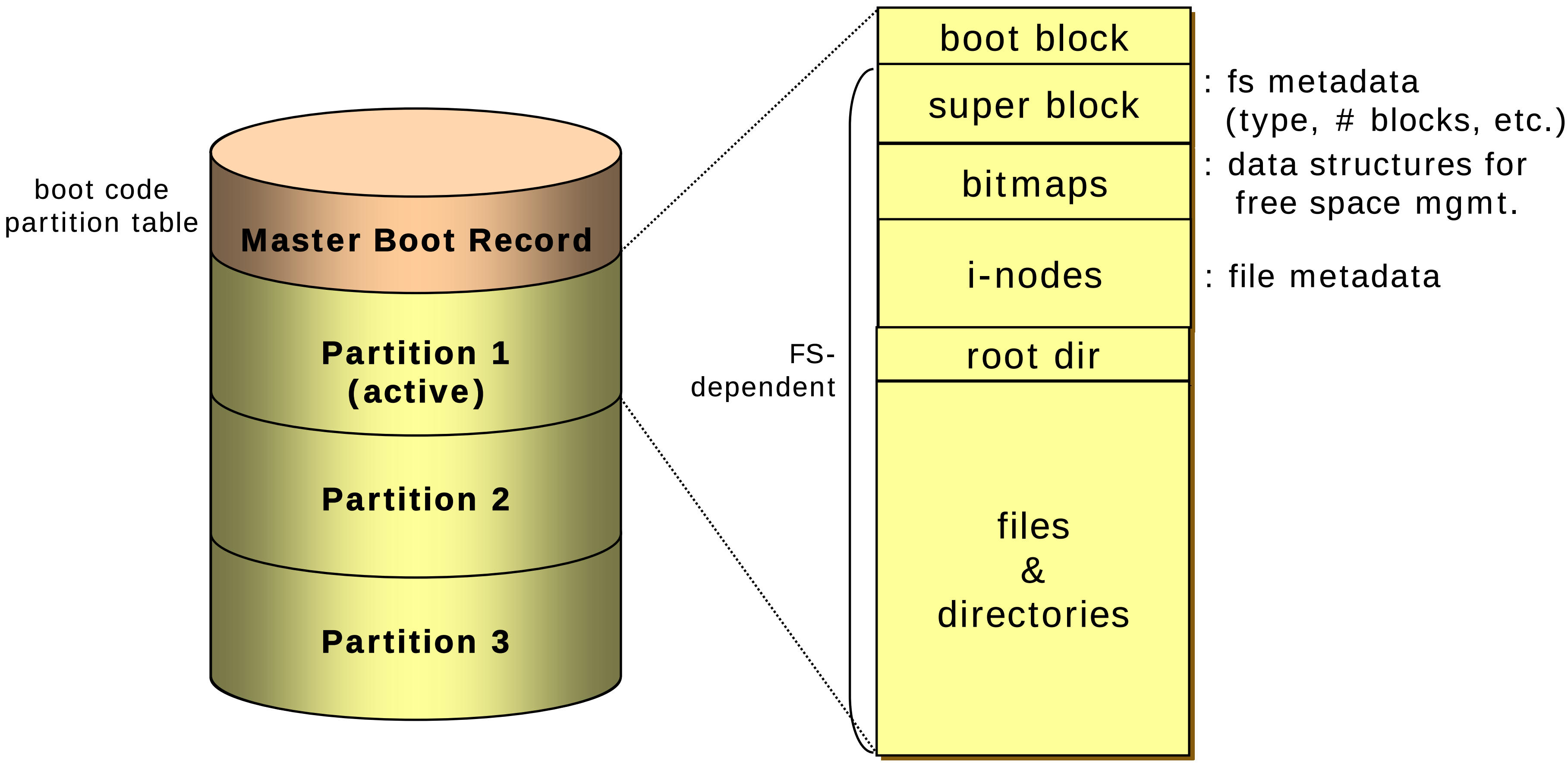
File System Implementation

Overview

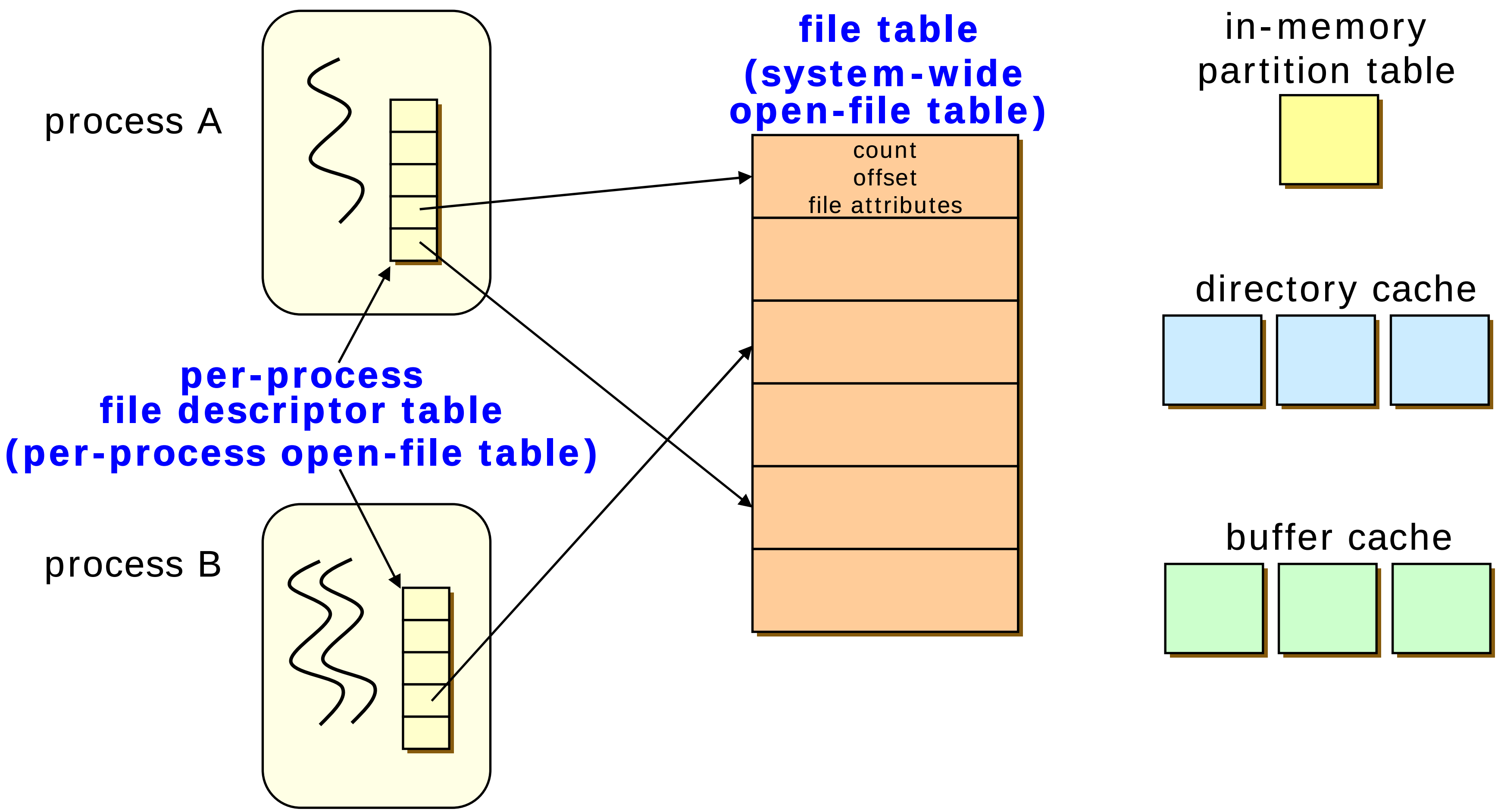
- User's view on file systems:
  - How files are named?
  - What operations are allowed on them?
  - What the directory tree looks like?

- Implementor's view on file systems:
  - How files and directories are stored?
  - How disk space is managed?
  - How to make everything work efficiently and reliably?

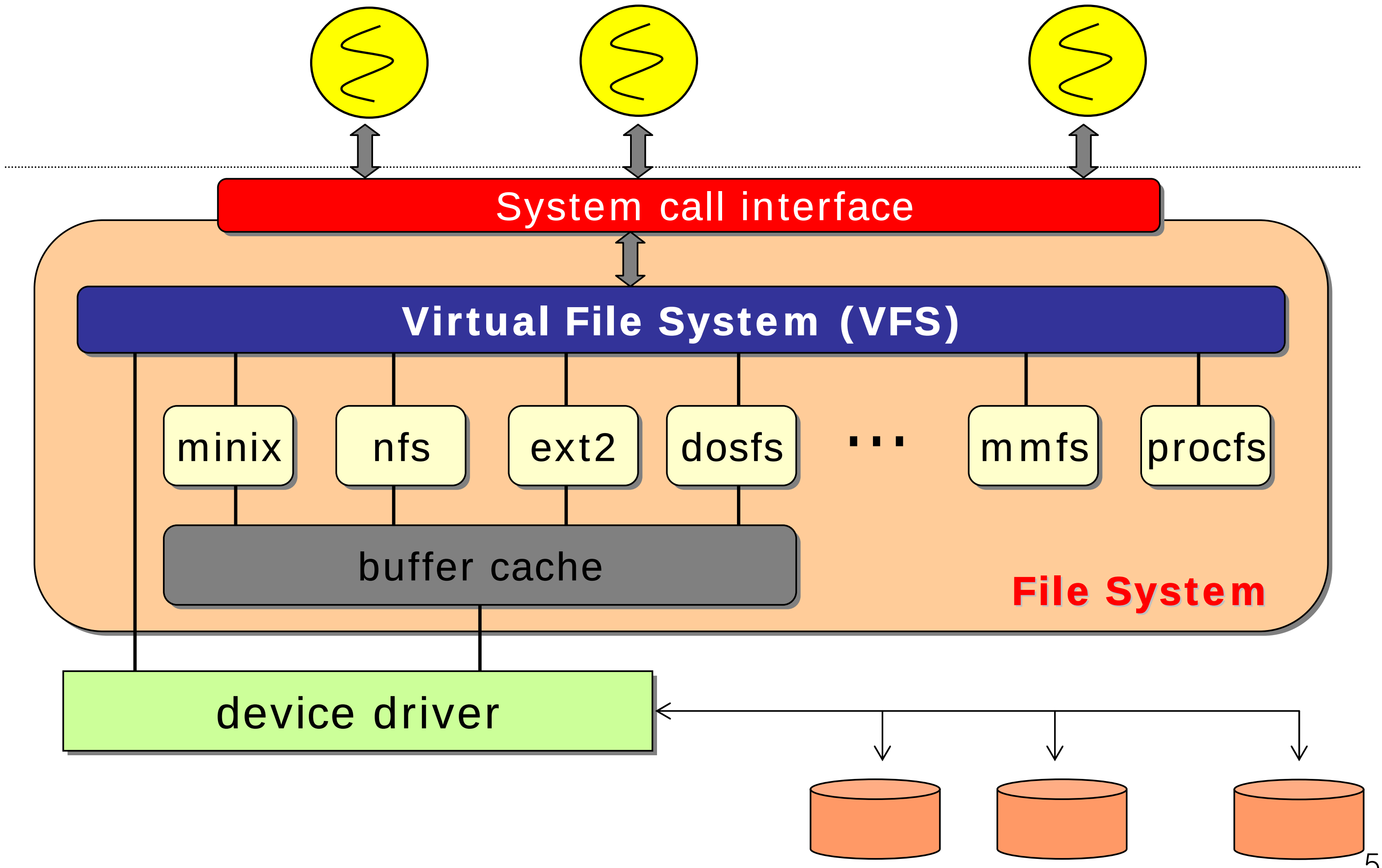
Disk Layout


boot code
partition table
Master Boot Record
Partition 1
(active)
Partition 2
Partition 3
FS-
dependent
boot block
super block
bitmaps
i-nodes
root dir
files
&
directories
: fs metadata
(type, # blocks, etc.)
: data structures for
free space mgmt.
: file metadata

In-memory Structures



File System Internals



VFS (1)

- Virtual File System
  - Manages kernel-level file abstractions in one format for all file systems.
  - Receives system call requests from user-level (e.g., open, write, stat, etc.)
  - Interacts with a specific file system based on mount point traversal.
  - Receives requests from other parts of the kernel, mostly from memory management.
  - Translates file descriptors to VFS data structures (such as vnode).

VFS (2)

- Linux: VFS common file model
  - The superblock object
    - stores information concerning a mounted file system.
  - The inode object
    - stores general information about a specific file.
  - The file object
    - stores information about the interaction between an open file and a process.
  - The dentry object
    - stores information about the linking of a directory entry with the corresponding file.
  - In order to stick to the VFS common file model, in-kernel structures may be constructed on the fly.

Directory Implementation (1)

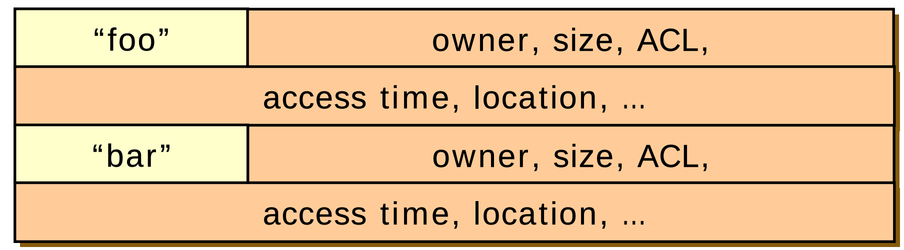
- Directory structure
  - Table (fixed length entries)
  - Linear list
    - Simple to program, but time-consuming.
    - Requires a linear search to find an entry.
    - Entries may be sorted to decrease the average search time and to produce a sorted directory listing easily (e.g., using B-tree).
  - Hash table
    - Decreases the directory search time.
    - A hash table is generally fixed size and the hash function depends on that size. (need mechanisms for collisions)
    - The number of files can be large:

      (1) enlarge the hash table and remap.

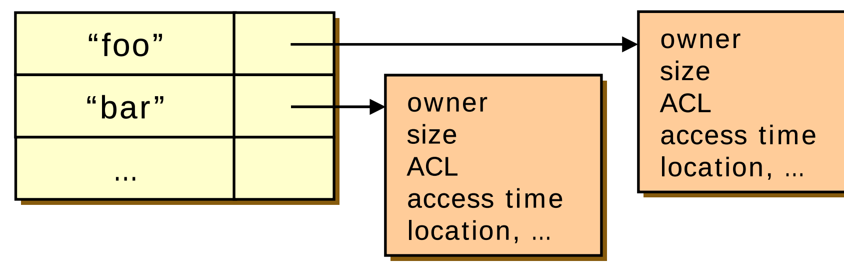
      (2) use a chained-overflow hash table.

Directory Implementation (2)

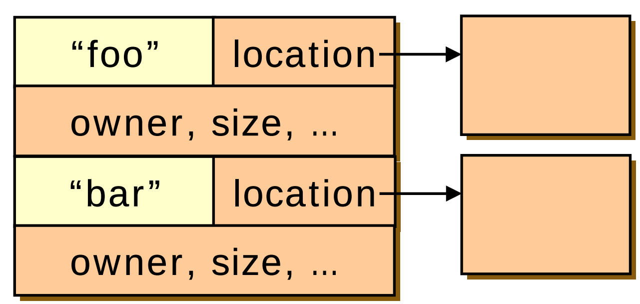
- The location of metadata
  - In the directory entry



  - In the separate data structure (e.g., i-node)

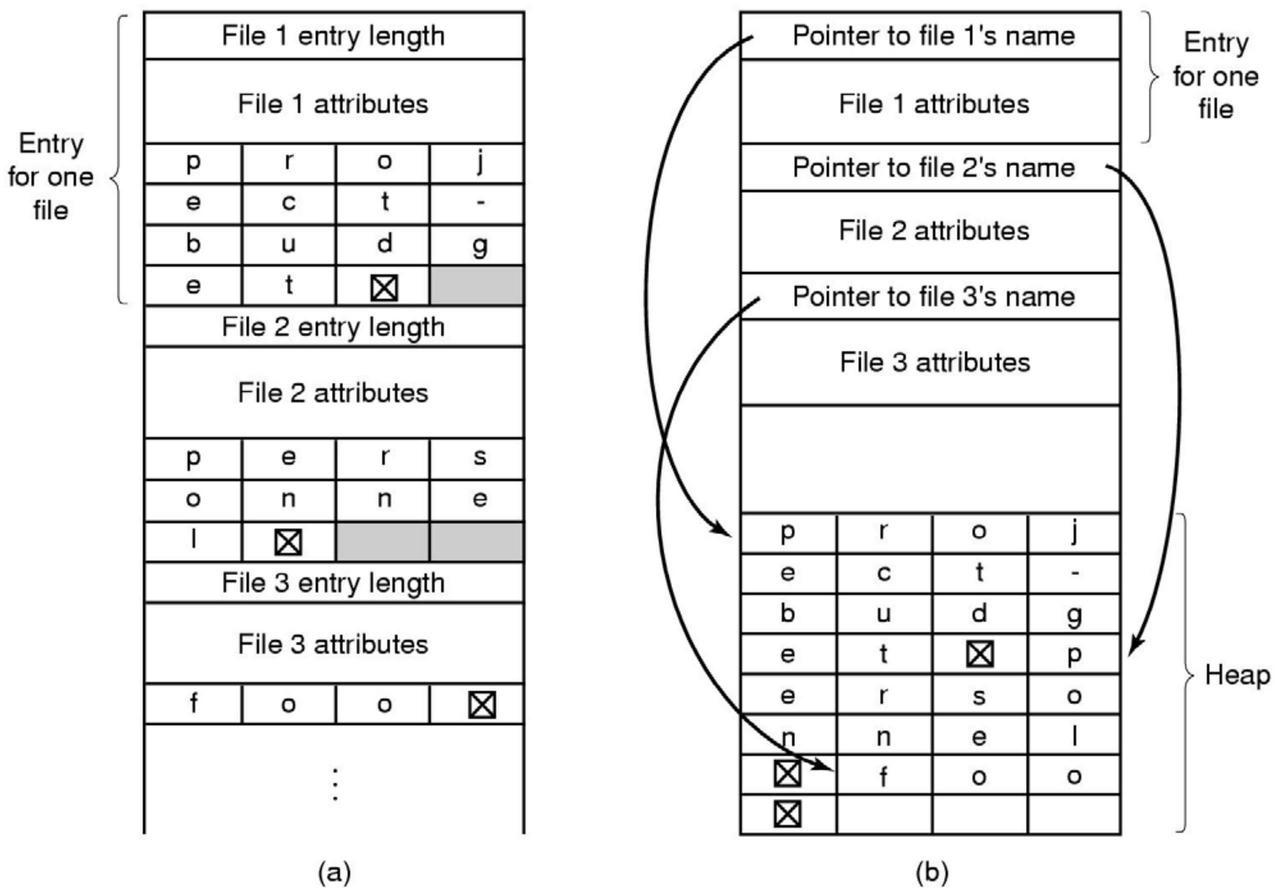


  - A hybrid approach



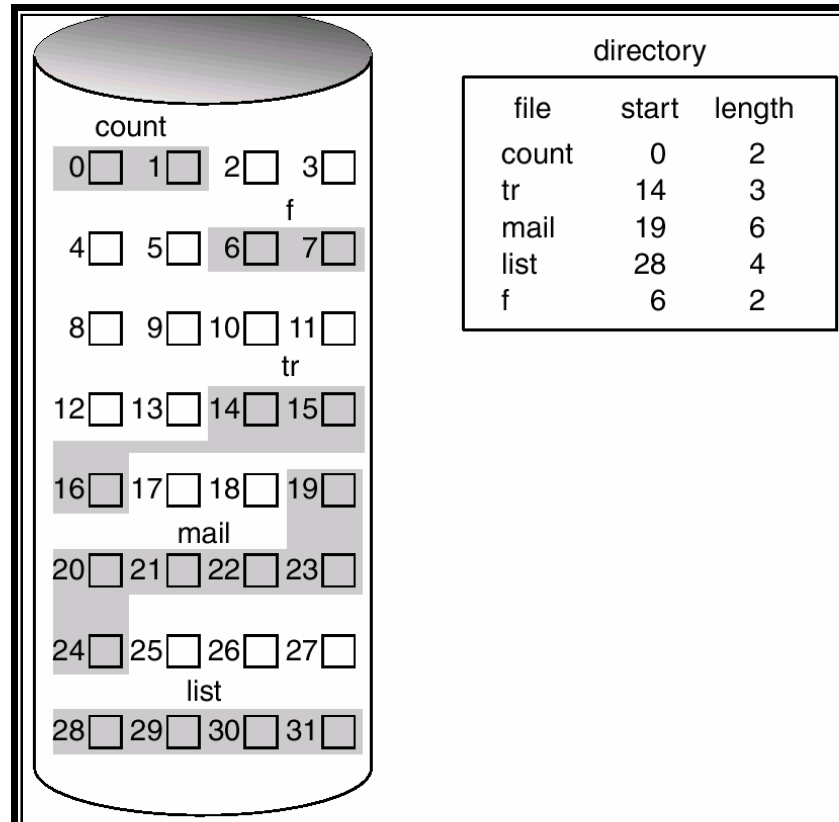
Directory Implementation (3)

- Supporting long file names



Allocation (1)

- Contiguous allocation
  - A file occupies a set of contiguous blocks on the disk.
  - Used by IBM VM/CMS



Allocation (2)

- Contiguous allocation (cont'd)
  - Advantages
    - The number of disk seeks is minimal.
    - Directory entries can be simple:
      <file name, starting disk block, length, etc.>
  - Disadvantages
    - Requires a dynamic storage allocation: First / best fit.
    - External fragmentation: may require a compaction.
    - The file size is hard to predict and varying over time.
  - Feasible and widely used for CD-ROMS
    - All the file sizes are known in advance.
    - Files will never change during subsequent use.

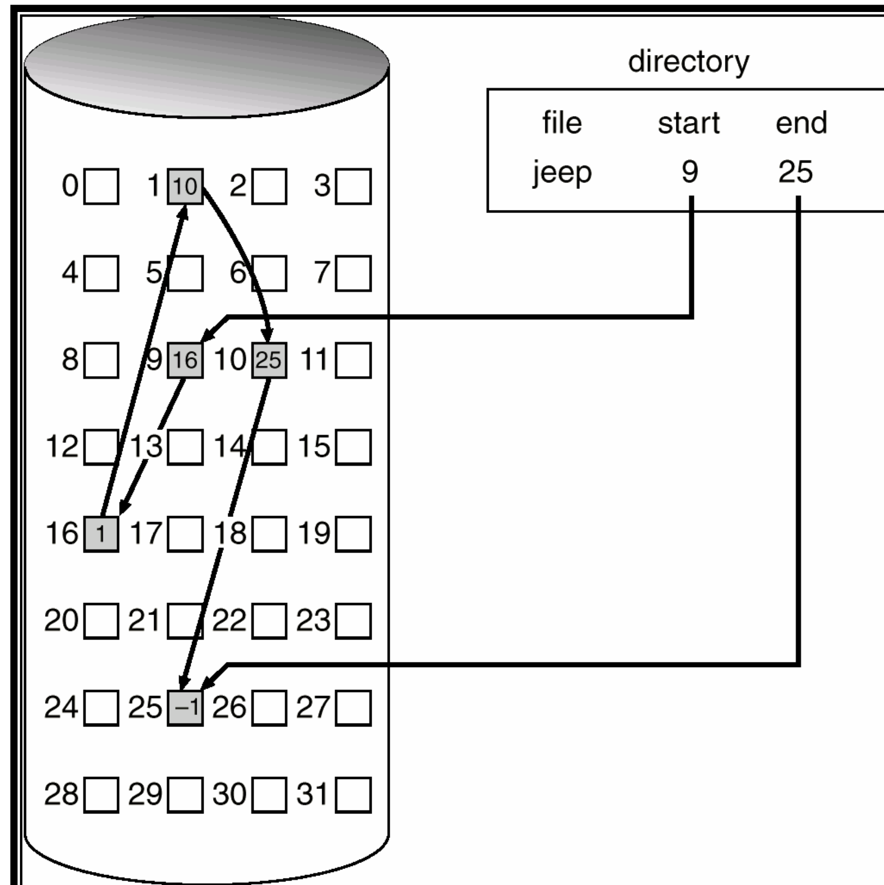
Allocation (3)

- Modified contiguous allocation
  - A contiguous chunk of space is allocated initially.
    - When the amount is not large enough, another chunk of a contiguous space (an extent) is added.
  - Advantages
    - Still the directory entry can be simple.

      <name, starting disk block, length, link to the extent>
  - Disadvantages
    - Internal fragmentation: if the extents are too large.
    - External fragmentation: if we allow varying-sized extents.
  - Used by Veritas File System (VxFS).

Allocation (4)

- Linked allocation
  - Each file is a linked list of disk blocks.



Allocation (5)

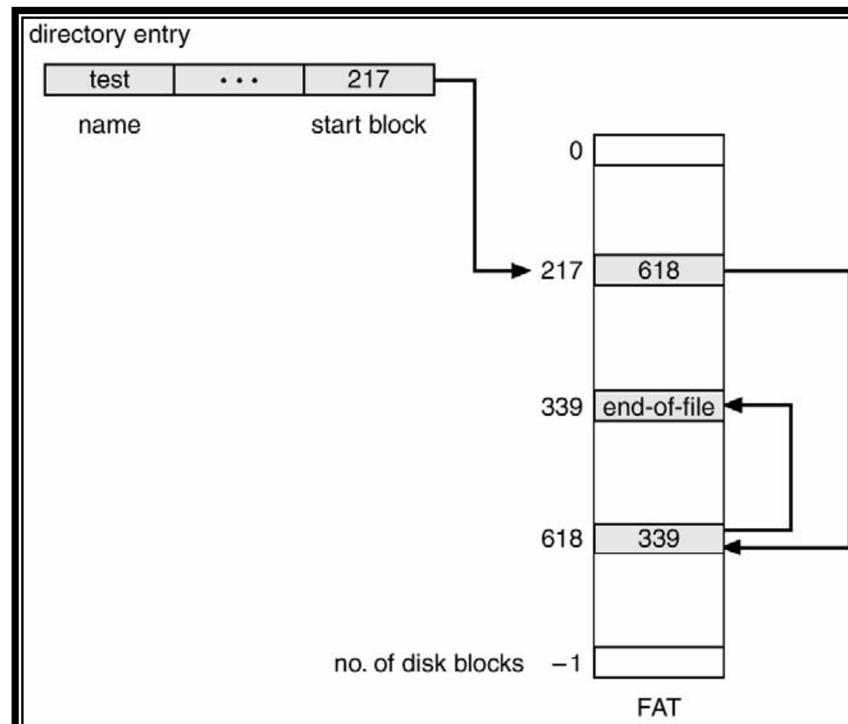
- Linked allocation (cont'd)
  - Advantages
    - Directory entries are simple:
      <file name, starting block, ending block, etc.>
    - No external fragmentation: the disk blocks may be scattered anywhere on the disk.
    - A file can continue to grow as long as free blocks are available.
  - Disadvantages
    - It can be used only for sequentially accessed files.
    - Space overhead for maintaining pointers to the next disk block.
    - The amount of data storage in a block is no longer a power of two because the pointer takes up a few bytes.
    - Fragile: a pointer can be lost or damaged.

Allocation (6)

- Linked allocation using clusters
  - Collect blocks into multiples (clusters) and allocate the clusters to files.
    - e.g., 4 blocks / 1 cluster
  - Advantages
    - The logical-to-physical block mapping remains simple.
    - Improves disk throughput (fewer disk seeks)
    - Reduced space overhead for pointers.
  - Disadvantages
    - Internal fragmentation

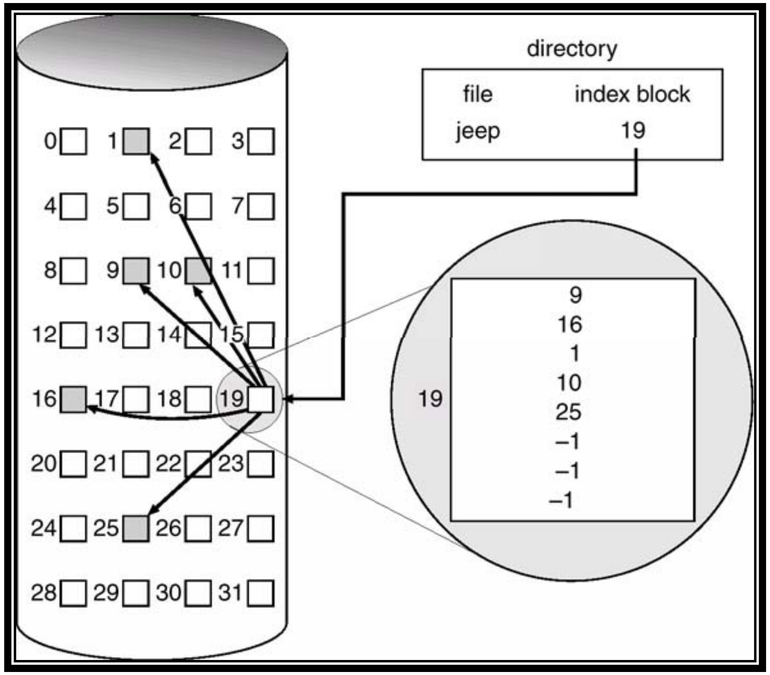
Allocation (7)

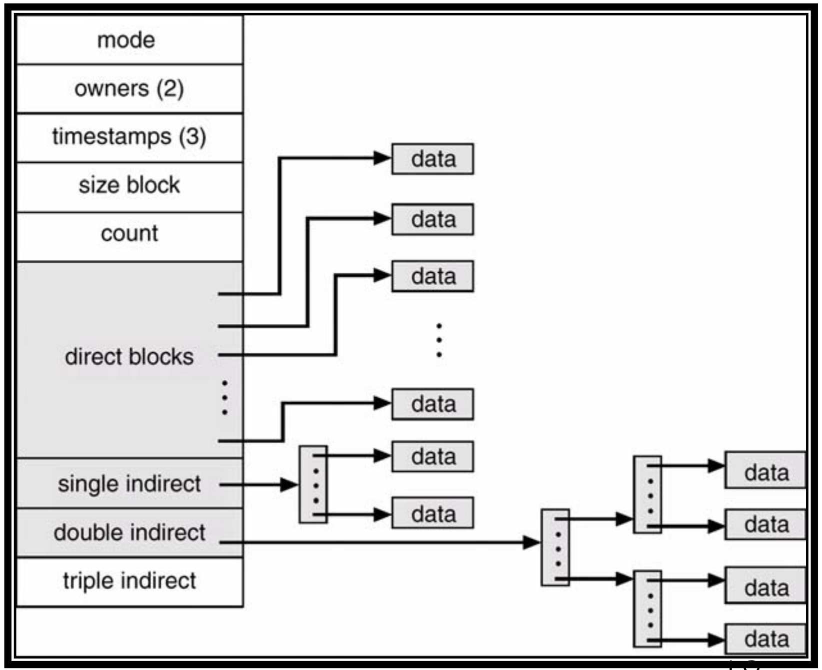
- Linked allocation using a FAT
  - A section of disk at the beginning of each partition is set aside to contain a file allocation table (FAT).
  - FAT should be cached to minimize disk seeks.
    - Space overhead can be substantial.
  - Random access time is improved.
  - Used by MS-DOS, OS/2
    - cf. FAT-16: 2GB limitation with 32KB block size



Allocation (8)

- Indexed allocation
  - Bring all the pointers together into one location (index block or i-node)
  - Each file has its own index block.





Allocation (9)

- Indexed allocation (cont'd)
  - Advantages
    - Supports direct access, without suffering from external fragmentation.
    - I-node need only be in memory when the corresponding file is open.
  - Disadvantages
    - Space overhead for indexes:

      (1) Linked scheme: link several index blocks

      (2) Multilevel index blocks

      (3) Combined scheme: UNIX

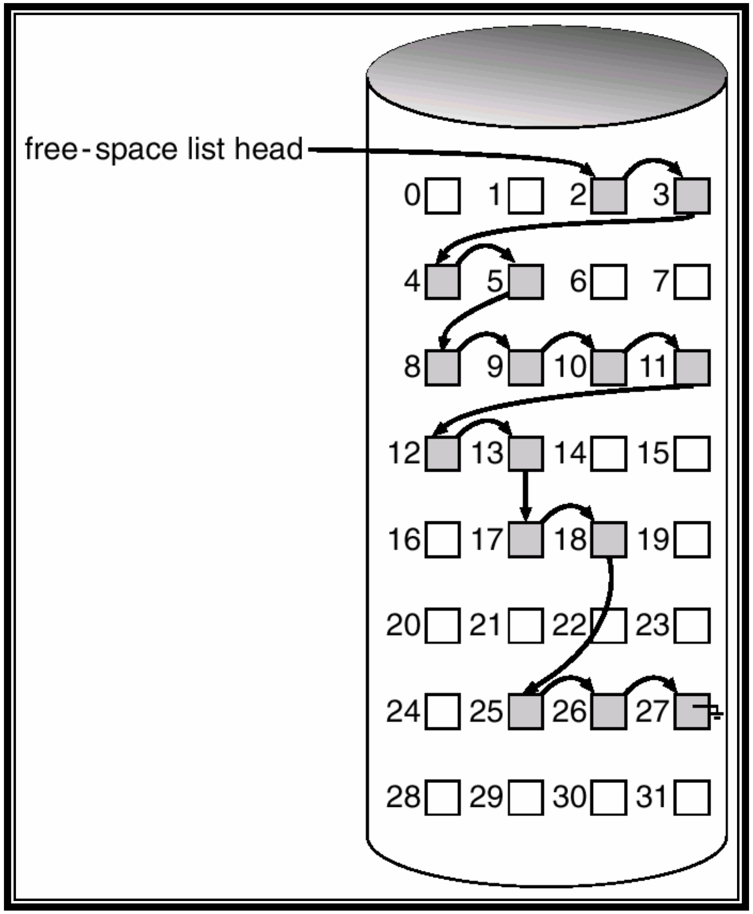
      - 12 direct blocks, single indirect block, double indirect block, triple indirect block

Free Space Management (1)

- Bitmap or bit vector
  - Each block is represented by 1 bit.
    - 1 = free, 0 = allocated
  - Simple and efficient in finding the first free block.
    - May be accelerated by CPU's bit-manipulation instructions.
  - Inefficient unless the entire vector is kept in main memory.
    - Clustering reduces the size of bitmaps.

Free Space Management (2)

- Linked list
  - Link together all the free disk blocks, keeping a pointer to the first free blocks.
  - To traverse the list, we must read each block, but it's not a frequent action.
  - The FAT method incorporates free-block accounting into the allocation data structure.



Free Space Management (3)

- Grouping
  - Store the addresses of *n* free blocks in the first free block.
  - The addresses of a large number of free blocks can be found quickly.
- Counting
  - Keep the address of the free block and the number of free contiguous blocks.
  - The length of the list becomes shorter and the count is generally greater than 1.
    - Several contiguous blocks may be allocated or freed simultaneously.

Reliability (1)

- File system consistency
  - File system can be left in an inconsistent state if cached blocks are not written out due to the system crash.
  - It is especially critical if some of those blocks are i-node blocks, directory blocks, or blocks containing the free list.
  - Most systems have a utility program that checks file system consistency
    - Windows: scandisk
    - UNIX: fsck

Reliability (2)

- fsck: checking blocks
  - Reads all the i-nodes and mark used blocks.
  - Examines the free list and mark free blocks.

Consistent

	0	1	2	3	4	5	6	7
Blocks in use	1	1	0	1	0	1	1	1
Free blocks	0	0	1	0	1	0	0	0

Missing block

-- add it to the free list

	0	1	2	3	4	5	6	7
Blocks in use	1	1	0	1	0	1	1	1
Free blocks	0	0	0	0	1	0	0	0

Duplicated free block

-- rebuild the free list

	0	1	2	3	4	5	6	7
Blocks in use	1	1	0	1	0	1	1	1
Free blocks	0	0	1	0	2	0	0	0

Duplicated data block

-- allocate a new block and copy

	0	1	2	3	4	5	6	7
Blocks in use	1	1	0	1	0	2	1	1
Free blocks	0	0	1	0	1	0	0	0

Reliability (3)

- fsck: checking directories
  - Recursively descends the tree from the root directory, counting the number of links for each file.
  - Compare these numbers with the link counts stored in the i-nodes.
  - Force the link count in the i-node to the actual number of directory entries.

i-node	count
1	1
5	2
12	4
...	...

i-node #5
...
count=3
...

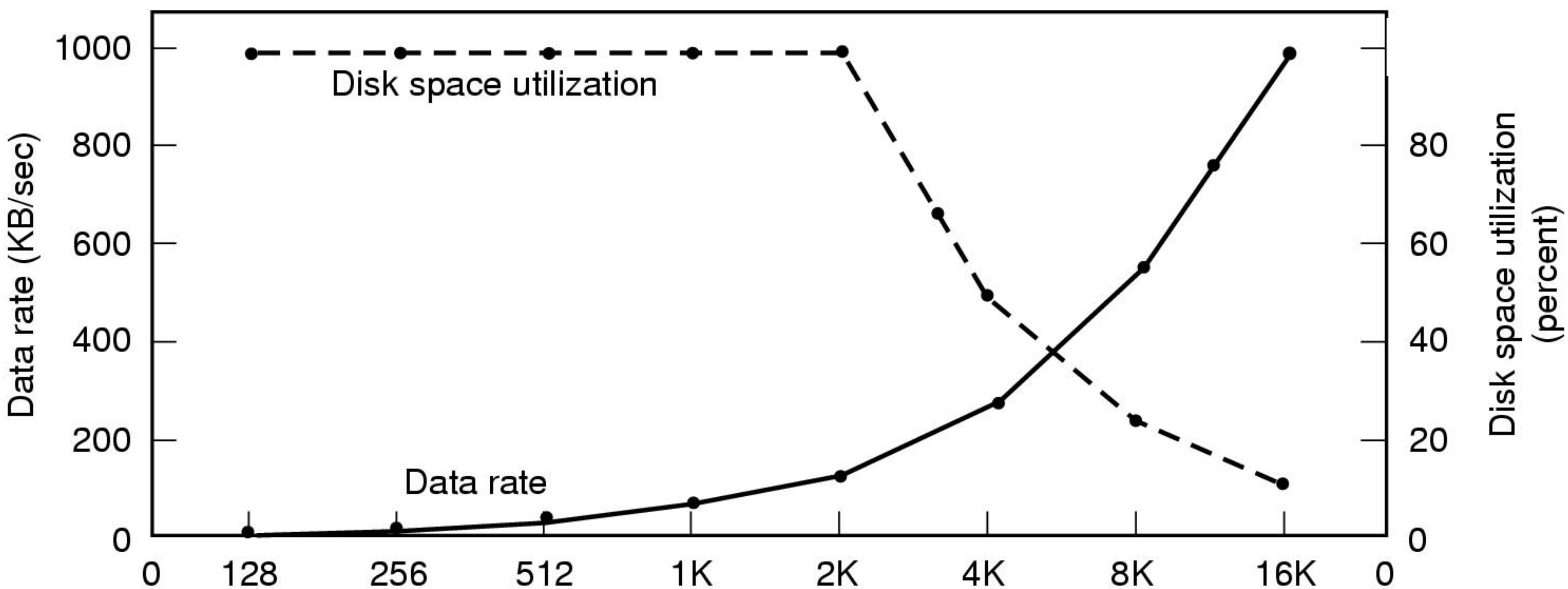
i-node #12
...
count=2
...

Reliability (4)

- Journaling file systems
  - Fsck'ing takes a long time, which makes the file system restart slow in the event of system crash.
  - Record a log, or journal, of changes made to files and directories to a separate location. (preferably a separate disk).
  - If a crash occurs, the journal can be used to undo any partially completed tasks that would leave the file system in an inconsistent state.
  - IBM JFS for AIX, Linux
    Veritas VxFS for Solaris, HP-UX, Unixware, etc.
    SGI XFS for IRIX, Linux
    Reiserfs, ext3 for Linux

Performance (1)

- Block size
  - Disk block size vs. file system block size
  - The median file size in UNIX is about 1KB.



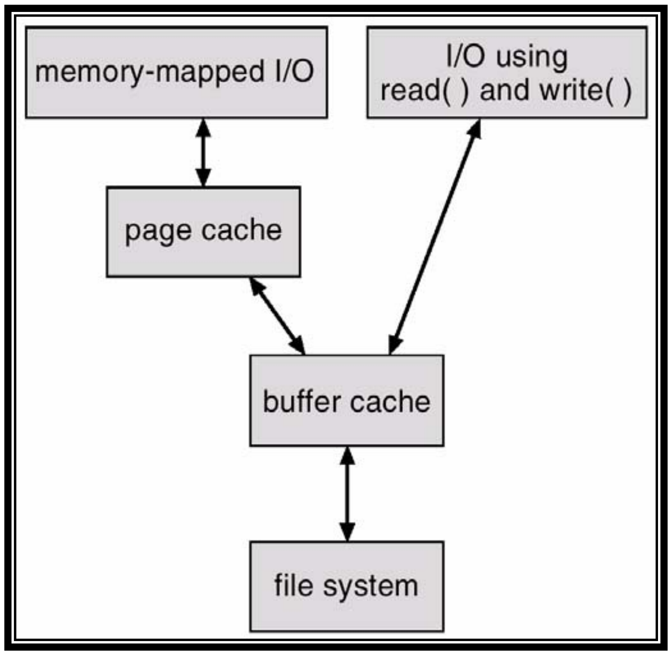
Performance (2)

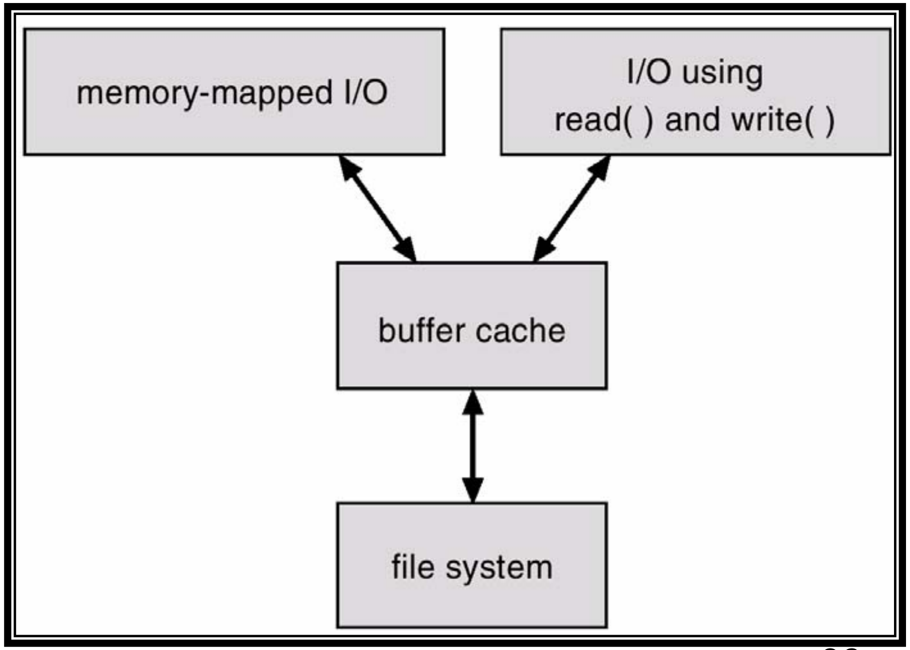
- Buffer cache
  - Applications exhibit significant locality for reading and writing files.
  - Idea: cache file blocks in memory to capture locality in buffer cache (or buffer cache).
    - Cache is system wide, used and shared by all processes.
    - Reading from the cache makes a disk perform like memory.
    - Even a 4MB cache can be very effective.
  - Issues
    - The buffer cache competes with VM.
    - Live VM, it has limited size.
    - Need replacement algorithms again.

      (References are relatively infrequent, so it is feasible to keep all the blocks in exact LRU order)

Performance (3)

- Unified buffer cache
  - Page caches are used to cache virtual memory pages and mmap()'ed file data.
  - Unified buffer cache avoids double caching and the possibility of inconsistencies between page caches and buffer caches.





Performance (4)

- Caching writes
  - Synchronous writes are very slow.

  (1) write-behind (or asynchronous writes)
    - Maintain a queue of uncommitted blocks.
    - Periodically flush the queue to disk.
    - Unreliable: metadata requires synchronous writes. (with small files, most writes are to metadata)

  (2) Battery backed-up RAM (NVRAM)
    - Maintain a queue in NVRAM
    - Expensive

  (3) Log-structured file system
    - Always write next block after last block written
    - Complicated

Performance (5)

- Read ahead
  - File system predicts that the process will request next block.
    - File system goes ahead and requests it from the disk.
    - This can happen while the process is computing on previous block, overlapping I/O with execution.
    - When the process requests block, it will be in cache.
  - Compliments the disk cache, which also is doing read ahead.
  - Very effective for sequentially accessed files.
  - File systems try to prevent blocks from being scattered across the disk during allocation or by restructuring periodically.

FFS (1)

- Fast file system (FFS)
  - The original Unix file system (70’s) was very simple and straightforwardly implemented:
    - Easy to implement and understand.
    - But very poor utilization of disk bandwidth (lots of seeking).
  - BSD Unix folks redesigned file system called FFS.
    - McKusick, Joy, Fabry, and Leffler (mid 80’s)
    - Now it is the file system from which all other UNIX file systems have been compared.
  - The basic idea is aware of disk structure.
    - Place related things on nearby cylinders to reduce seeks.
    - Improved disk utilization, decreased response time.

FFS (2)

- Data and i-node placement
  - Original Unix FS had two major problems:

  (1) Data blocks are allocated randomly in aging file systems.
    - Blocks for the same file allocated sequentially when FS is new.
    - As FS "ages" and fills, need to allocate blocks freed up when other files are deleted.
    - Problem: Deleted files essentially randomly placed.
    - So, blocks for new files become scattered across the disk.

  (2) i-nodes are allocated far from blocks.
    - All i-nodes at the beginning of disk, far from data.
    - Traversing file name paths, manipulating files and directories require going back and forth from i-nodes to data blocks.
  - Both of these problems generate many long seeks!

FFS (3)

- Cylinder groups
  - BSD FFS addressed these problems using the notion of a cylinder group.
  - Disk partitioned into groups of cylinders.
  - Data blocks from a file all placed in the same cylinder group.
  - Files in same directory placed in the same cylinder group.
  - i-nodes for files allocated in the same cylinder group as file's data blocks.

FFS (4)

- Free space reserve
  - The disk must have free space scattered across all cylinders.
  - If the number of free blocks falls to zero, the file system throughput tends to be cut in half, because of the inability to localize blocks in a file.
  - A parameter, called free space reserve, gives the minimum acceptable percentage of file system blocks that should be free.
  - If the number of free blocks drops below this level, only the system administrator can continue to allocate blocks.
  - Normally 10%; this is why df may report > 100%.

FFS (5)

- Fragments
  - Small blocks (1KB) caused two problems:
    - low bandwidth utilization
    - small max file size (function of block size)
  - FFS fixes by using a larger block (4KB)
    - Allows for very large files .
      (1MB only uses 2 level indirect)
    - But introduces internal fragmentation: there are many small files (i.e., < 4KB)
  - FFS introduces "fragments" to fix internal fragmentation.
    - allows the division of a block into one or more fragments (1K pieces of a block).

FFS (6)

- Media failures
  - Replicate master block (superblock)

- File system parameterization
  - Parameterize according to disk and CPU characteristics.
    - Maximum blocks per file in a cylinder group
    - Minimum percentage of free space
    - Sectors per track
    - Rotational delay between contiguous blocks
    - Tracks per cylinder, etc.
  - Skip according to rotational rate and CPU latency.

Ext2 FS (1)

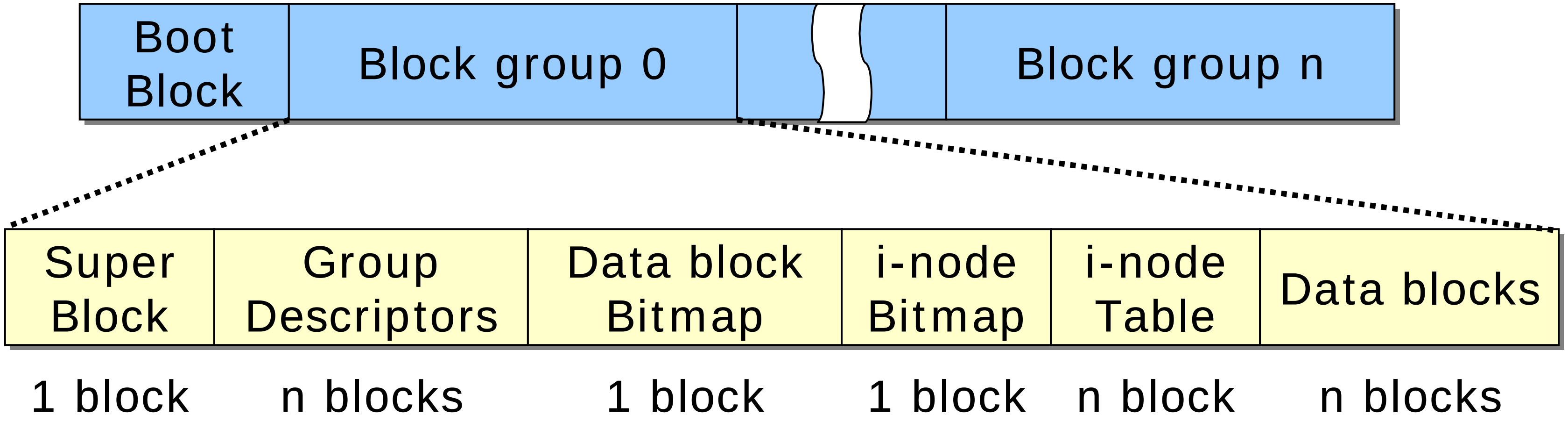
- History
  - Evolved from Minix filesystem.
    - Block addresses are stored in 16bit integers – maximal file system size is restricted to 64MB.
    - Directories contain fixed-size entries and the maximal file name was 14 characters.
  - Virtual File System (VFS) is added.
  - Extended Filesystem (Ext FS), 1992.
    - Added to Linux 0.96c
    - Maximum file system size was 2GB, and the maximal file name size was 255 characters.
  - Second Extended Filesystem (Ext2 FS), 1994.
  - Evolved to Ext3 File system (with journaling)

Ext2 FS (2)

- Ext2 Features
  - Configurable block sizes (from 1KB to 4KB)
    - depending on the expected average file size.
  - Configurable number of i-nodes
    - depending on the expected number of files
  - Partitions disk blocks into groups.
    - lower average disk seek time
  - Preallocates disk data blocks to regular files.
    - reduces file fragmentation
  - Fast symbolic links
    - If the pathname of the symbolic link has 60 bytes or less, it is stored in the i-node.
  - Automatic consistency check at boot time.

Ext2 FS (3)

- Disk layout
  - Boot block
    - reserved for the partition boot sector
  - Block group
    - Similar to the cylinder group in FFS.
    - All the block groups have the same size and are stored sequentially.



Ext2 FS (4)

- Block group
  - Superblock: stores file system metadata
    - Total number of i-nodes,
    - File system size in blocks
    - Free blocks / i-nodes counter
    - Number of blocks / i-nodes per group
    - Block size, etc.
  - Group descriptor
    - Number of free blocks / i-nodes / directories in the group
    - Block number of block / i-node bitmap, etc.
  - Both the superblock and the group descriptors are duplicated in each block group.
    - Only those in block group 0 are used by the kernel.
    - fsck copies them into all other block groups.
    - When data corruption occurs, fsck uses old copies to bring the file system back to a consistent state.

Ext2 FS (5)

- Block group size
  - The block bitmap must be stored in a single block.
    - In each block group, there can be at most 8xb blocks, where b is the block size in bytes.
  - The smaller the block size, the larger the number of block groups.
  - Example: 8GB Ext2 partition with 4KB block size
    - Each 4KB block bitmap describes 32K data blocks = 32K * 4KB = 128MB
    - At most 64 block groups are needed.

Ext2 FS (6)

- Directory structure

	inode	record length	name length	file type	name							
0	21	12	1	2	.	\0	\0	\0				
12	22	12	2	2	.	.	\0	\0				
24	53	16	5	2	h	o	m	e	1	\0	\0	\0
40	67	28	3	2	u	s	r	\0				
62	0	16	7	1	o	l	d	f	i	l	e	\0
68	34	12	3	2	b	i	n	\0				

<deleted file>