OPERATING SYSTEMS: FILE SYSTEMS

File Systems and File Server

Goals

- To know the concepts of file and directory as well as their characteristics.
- To use the file management services offered by the operating system.
- To understand the structure of a file system.
- To understand the mechanisms supporting a file server and to apply them in simple exercises.

Contents

- **File System structure.**
- File server.
- Associated data structures.
- Performance increase.

File systems and partitions

- The file system allows to organize information within secondary storage devices in an intelligible format for the operating system.
- Prior to file system install procedure it is necessary to physically (or logically) dividing disks into **partitions** and **volumes**.
- A **partition** is a disk portion with its own identity that can be manipulated by the OS as an independent logical entity.
- Once partitions are created, OS must create file system structures in those partitions.
  - Using commands **format** and **mkfs**.

#mkswap –c /dev/hda2 20800

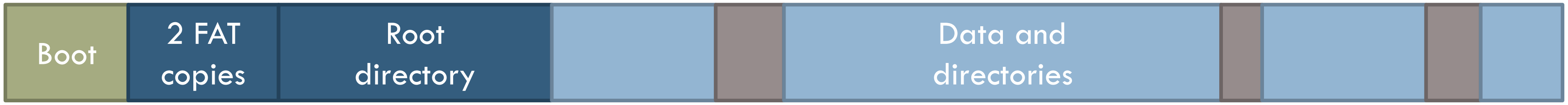
#mkfs -c /dev/hda3 –b 8196 123100

File System and partitions

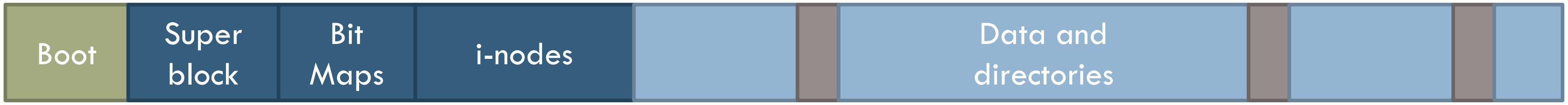
- File system: Coherent set of meta-information and data.

- File systems examples:

FAT



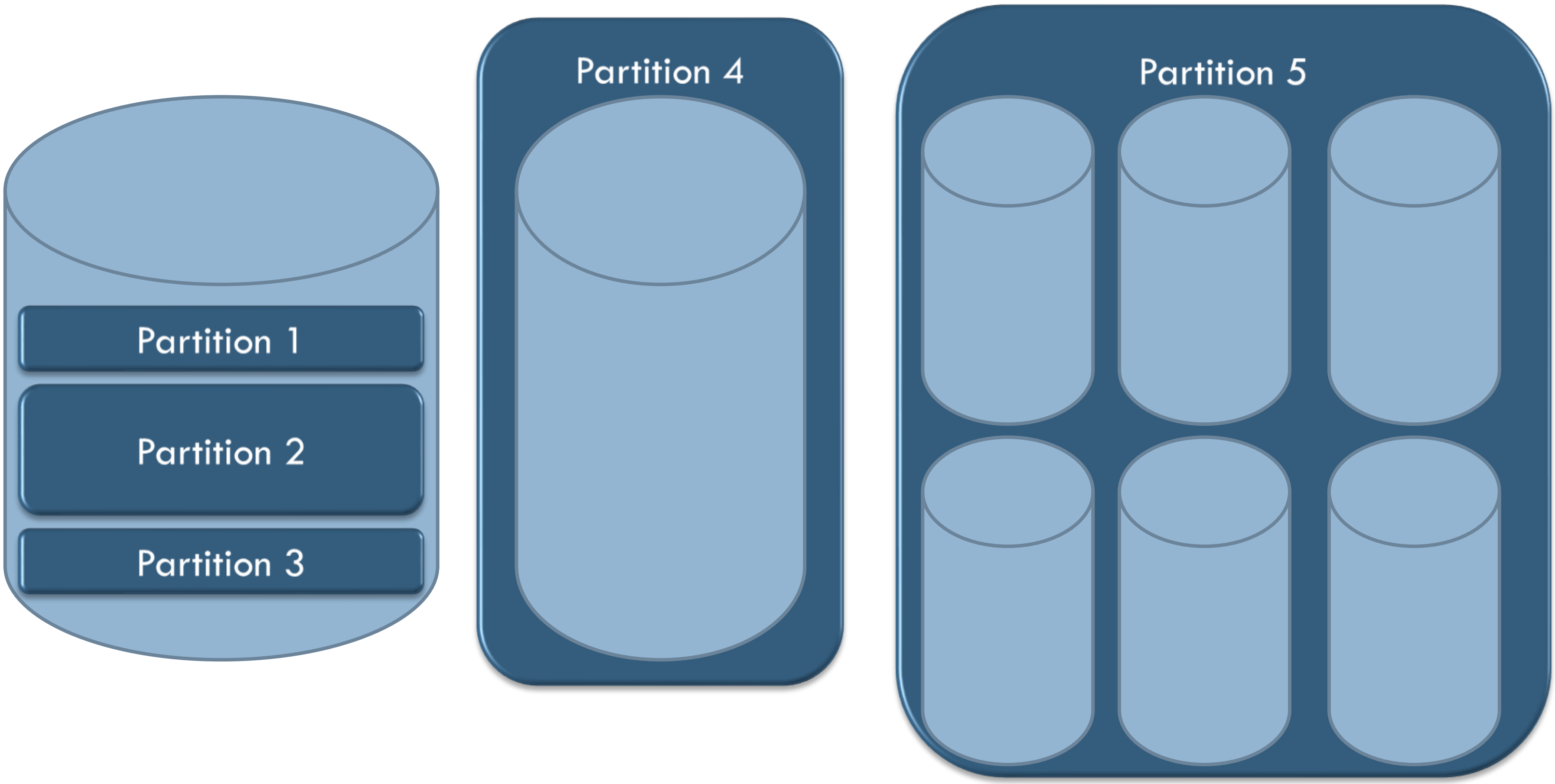
UNIX



File systems and partitions

- File system description:
  - Boot sector in MS-DOS.
  - Superblock in UNIX.
- Relationship file system to device:
  - Typical: 1 device to N file systems (partitions).
  - Large files: N devices to 1 file system.
- Typically, each device is divided in one or more partitions
  - One file system in each partition.
- Partition table contains start, size and type of partition.

Partition types



Blocks and clusters

- **Block**: logical grouping of disk sectors.
  - Minimal transfer unit used in the file system.
  - To optimize efficiency in I/O to secondary storage devices.
  - All OS provide a default block size.
  - Users may define block size to be used in a file system through **mkfs**.

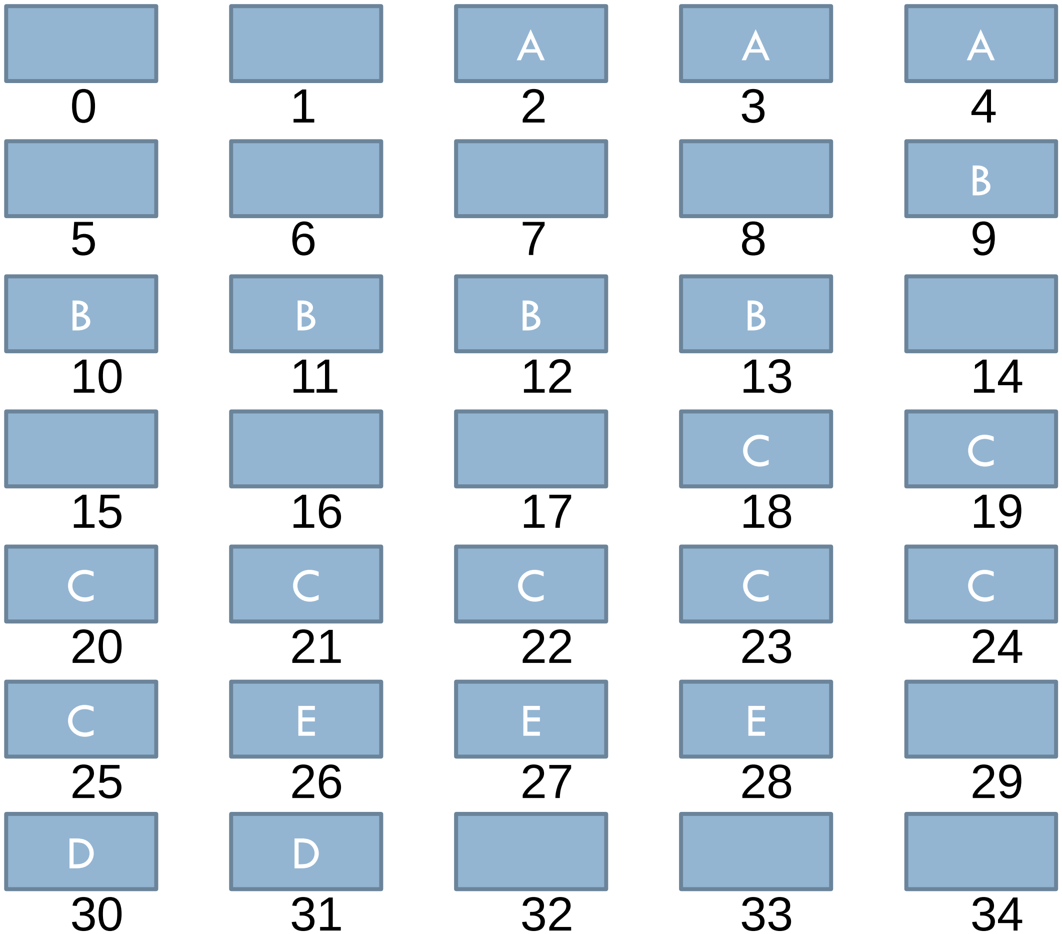
- **Cluster**: set of blocks managed as a logical storage management unit.
  - Problem introduced by large clusters and blocks: Internal fragmentation.

ISO-9660 file system

- Used for read-only devices → contiguous block allocation.

- Storage space divided in 5 areas:
  - **System area** (16 blocks): Used by Rock-Ridge y Joliet extensions.
  - **Volume description** (1 block): Information about CD image.
  - **Path tables**: Precompiled list of all directories → Accelerates lookups.
  - **Directories**: List of directory entries.
  - **Files**: File blocks.

Contiguous allocation

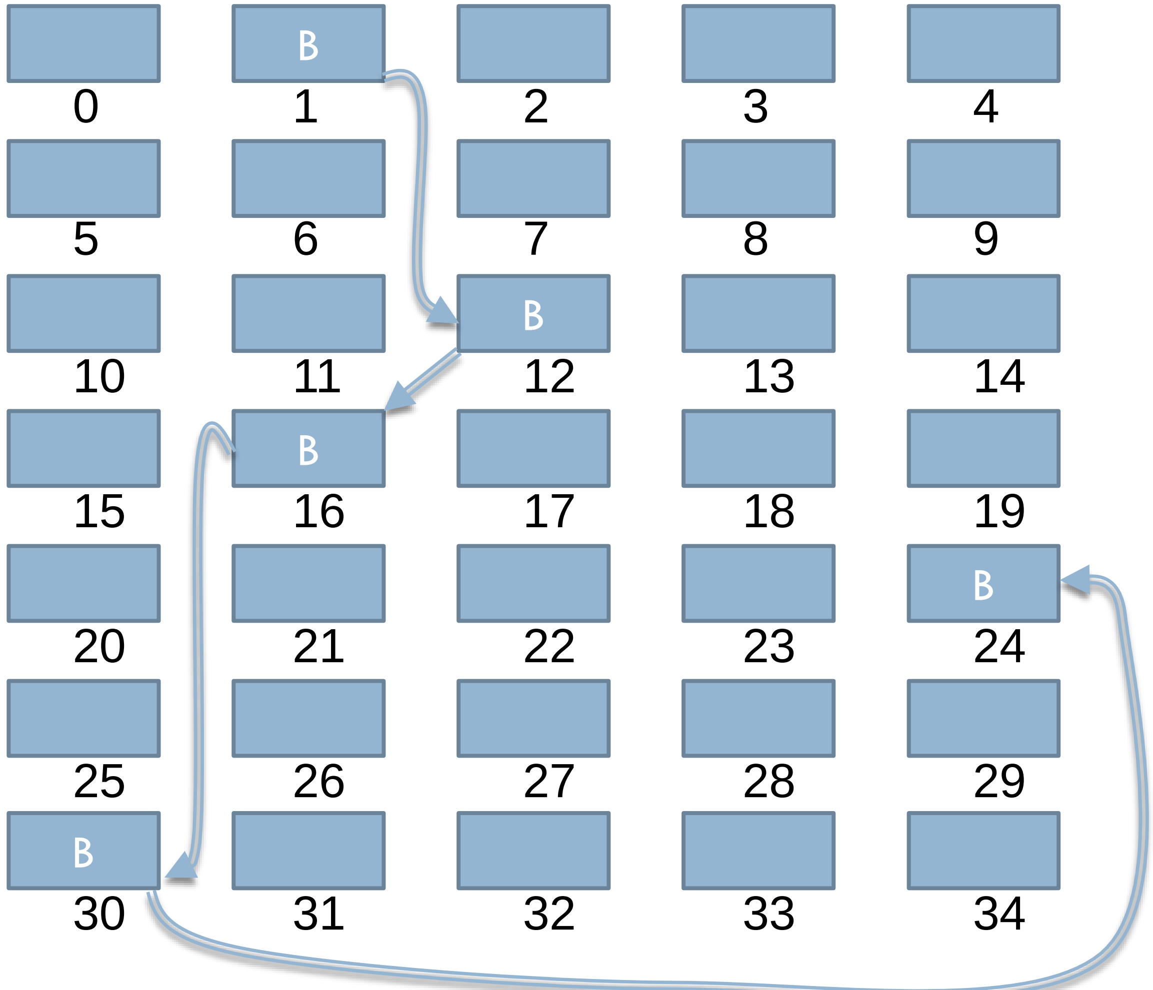


File	Start	Length
A	2	3
B	9	5
C	18	8
D	30	2
E	26	3

FAT file system

- Used in DOS and some old versions of Windows.
- Still used for portable storage devices.
- Space divided into:
  - **Boot**: OS boot info.
  - **FAT**: File allocation table.
  - **FAT copy**: Backup of FAT for reliability.
  - **Root directory**: Main directory in volume.
  - **Directories and files area**.

Linked allocation



File	Start	Length
B	1	5

FAT12

- Table with in entry per disk block.
- Table with 12-bit addresses.
- Maximum number of blocks: 4096.
- Block size: 512 bytes to 8 KB.
- Maximum size: 32 MB.
- Used for floppy disks.

FAT16

- Table with one entry per disk block.
- 16 bit addresses.
- Maximum number of blocks: 65,535.
- Block size: 512 byts to 64 KB.
- Used in old disks.

Block size	Max size
512 B	32 MB
1 KB	64 MB
2 KB	128 MB
4 KB	256 MB

Block size	Max size
8 KB	512 MB
16 KB	1 GB
32 KB	2 GB
64 KB	4 GB

Max size of FAT table: 128 KB

FAT32

- Table with on entry per disk block.
- 32 bit addresses (uses only 28).
- Maximum number of blocks: 256 Mblocks.
- Block size: 4kB to 32 KB.
- Windows limited to devices up to 32 GB.
- Used in portable storage devices.
- FAT may occupy an important amount of space.
  - Cannot be permanently in memory and must be read from disk.
- Maximum file size: 4 GB.

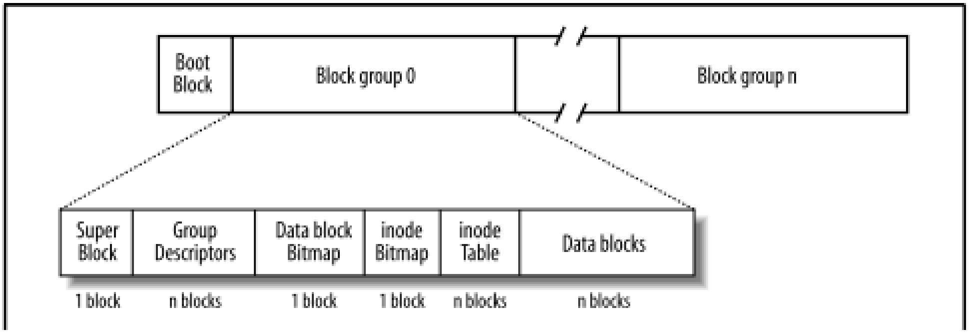
UNIX-like file systems

- Structure:
  - **Boot**: Operating system boot info.
  - **Superblock**: Descriptive information for file system structure.
    - **Virtual superblock**: Generic information.
    - **Specific superblock**: OS dependent information.
  - **Block bitmaps**: One bit per block to signal free/used.
  - **i-node bitmaps**: One bit per i-node (in i-node) area to signal free/used.
  - **i-nodes**: As many i-nodes as number of files that can be stored on the file system.
    - Linux creates one i-node for every to data blocks.
  - **Data blocks**.

BSD-like file systems

- Problems with UNIX-like file systems:
  - Metadata grouped at the beginning of disk.
  - Single copy of metadata → what about corruption of FS?
  - Block dispersion → Long seek time.

- Solutions is BSD (FFS) and ext2.
  - Partition divided into multiple areas: Cylinder groups (block groups)
  - Superblock replicated in each group.
  - In each group bitmaps and i-nodes for that group.



Contents

- File System structure.
- **File server.**
- Associated data structures.
- Performance increase.

File server

- Provides efficient and simple access to storage devices.
- Functions: store, find, and read data easily.

- Design problems:
  - Define **user view** of I/O system including services, files, directories, file systems, etc …
  - Define **algorithms and data structures** to use for mapping the user view to the physical system for secondary storage.

Already seen

File system layers

- **Virtual file system**: Provides I/O call interface
  - Independent from a particular file system.
- **File system organization module**: Transforms logical requests into physical ones.
  - Different for every particular file system
- **Block server**: Manages requests for block operations on devices.
  - Keeps a block cache and/or page cache.
- **Device driver**: Transforms block requests into device requests.
  - I/O scheduling policy.

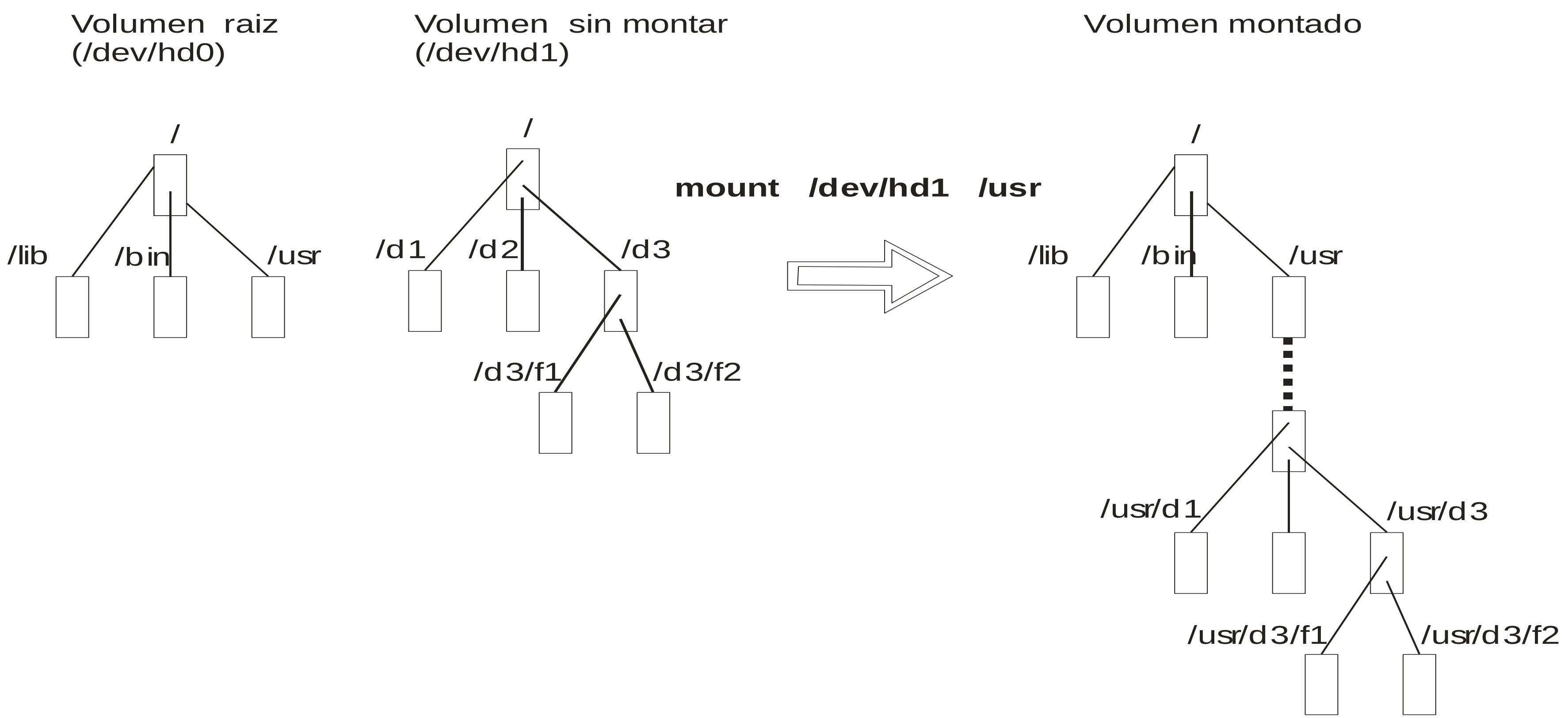
Virtual file system

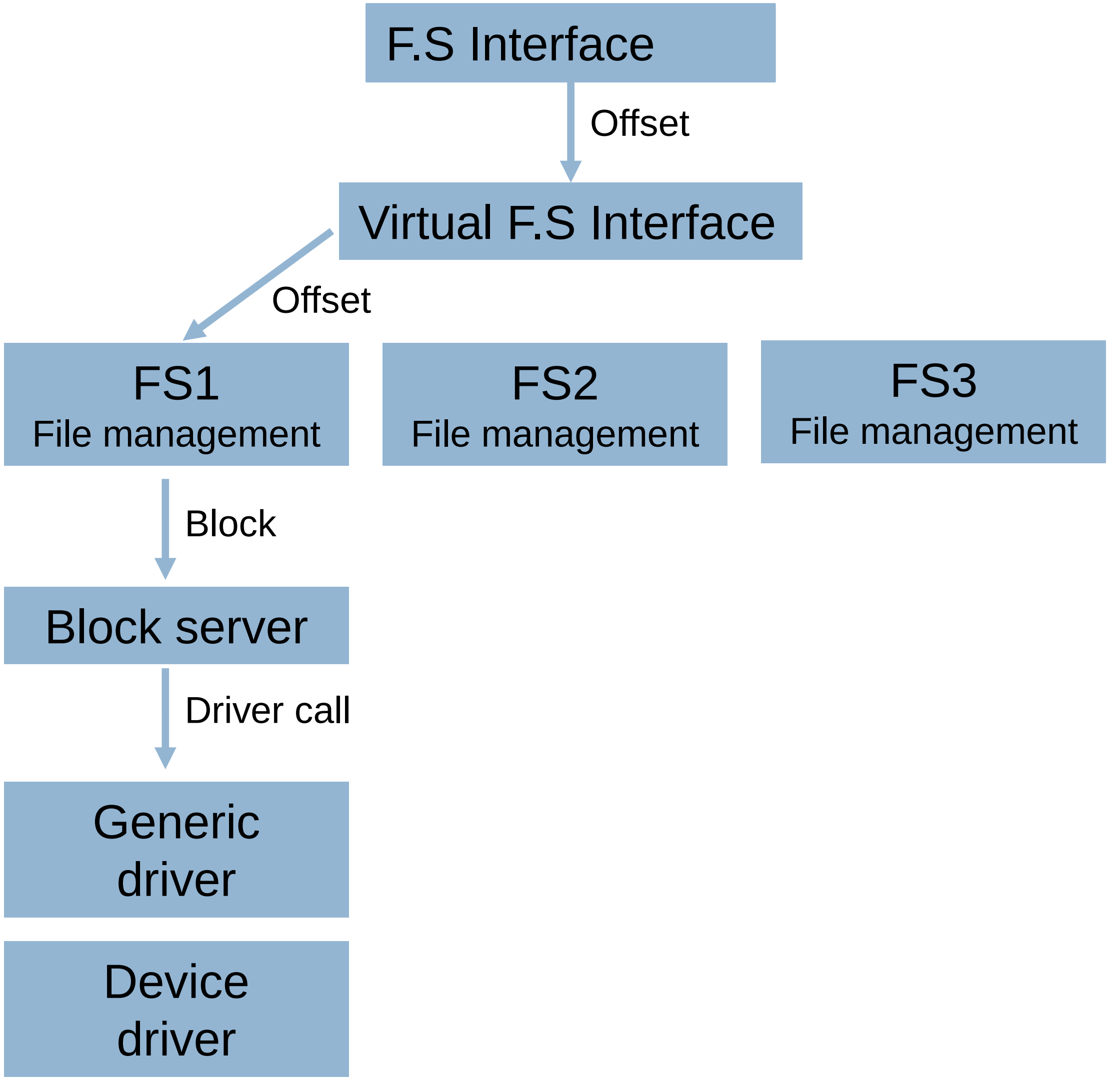
- Provides file system management calls interface.

- **Services**:
  - **Directory management.**
  - **Name management.**
  - **Security services.**
  - **Generic services on files and directories.**

- Data structure: v-node.
  - A virtual node (v-node) contains generic information on a file.

Example: mounting partitions




F.S Interface
Offset
Virtual F.S Interface
Offset
FS1
File management
FS2
File management
FS3
File management
Block
Block server
Driver call
Generic
driver
Device
driver

File management module

- Maps logical file image to physical file image.
  - Algorithms for mapping logical block addresses into physical addresses.
- Manages:
  - File system storage space.
  - Block allocation for files.
  - File descriptors (i-nodes) management.

- Highlights:
  - A file management module per supported file system (UNIX, AFS, NTFS, EFS, …)
  - Also files for pseudo files (e.g. files in /proc).

- This module resolves system calls specific to a file system.
  - Uses existing information in i-node.

Block server

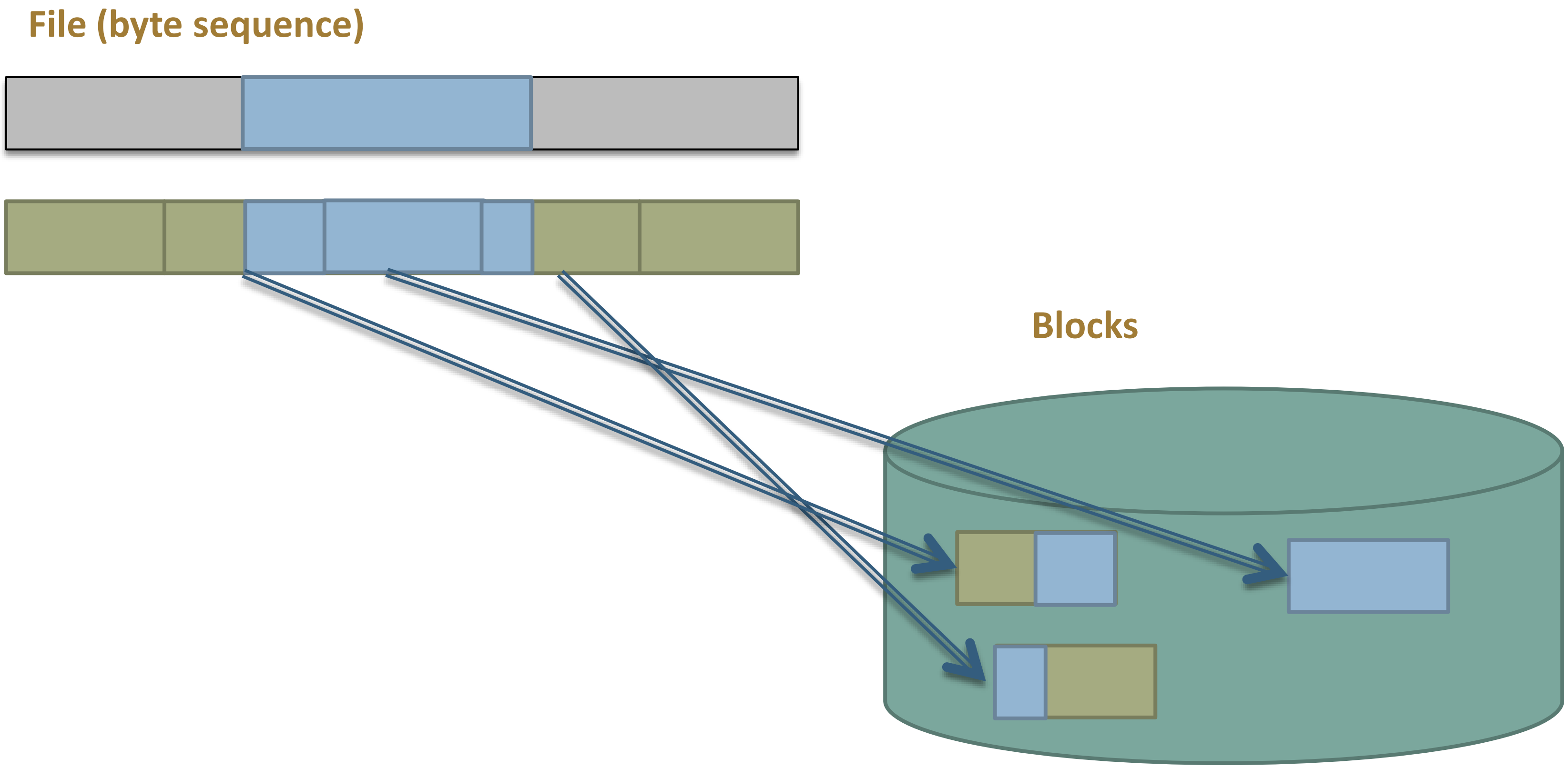
- Sends requests to device driver.
  - Read block.
  - Write block.

- Optimizations:
  - Block cache.
  - May be integrated with virtual memory page manager.

- Operations translated into calls to drivers for each specific devices and passed to lower level in file system.
- This layer hides device differences, using logical names.
  - For example, **/dev/hda3** is a hard disk (**hd**) device, whose main name is **a** and working on partition **3**.

Block operations

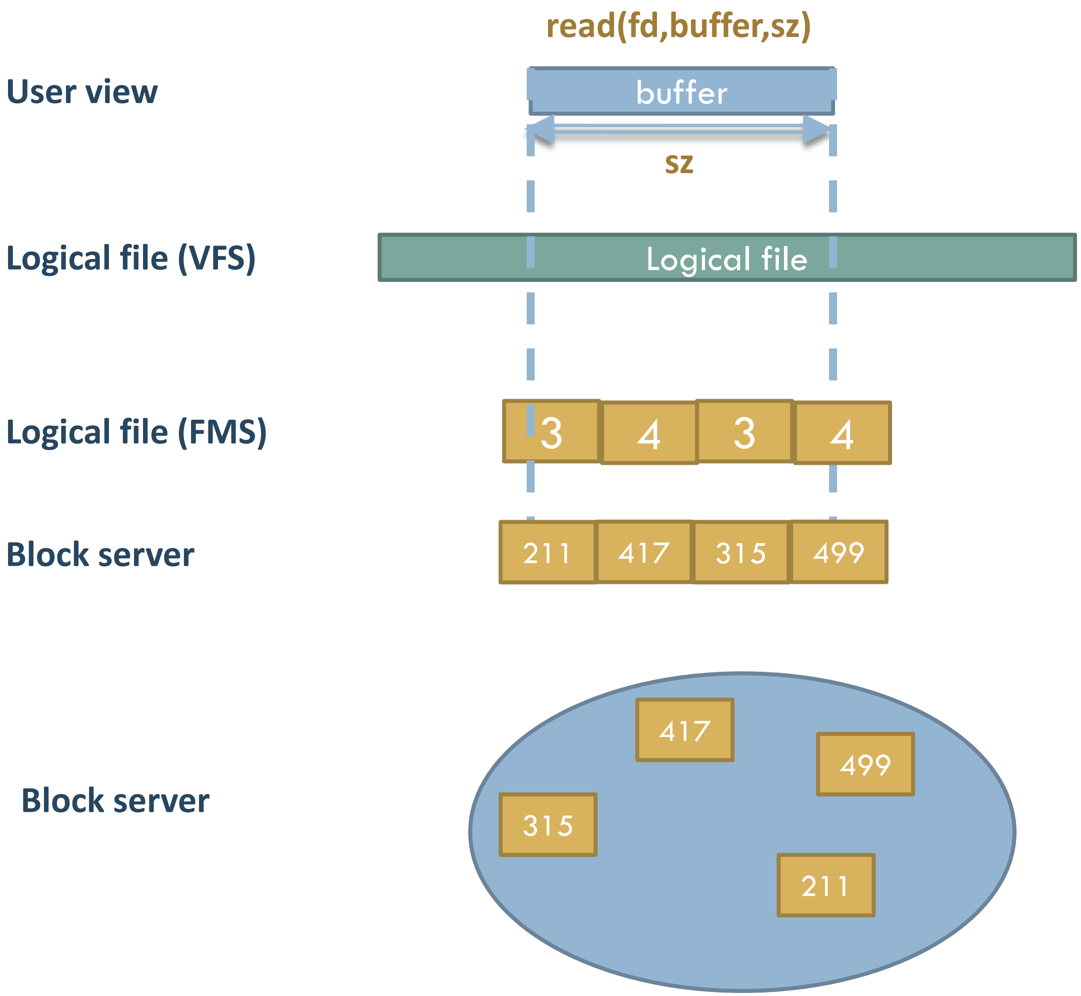
- Byte structure files



Device drivers

- On driver per device.
  - Consisting of generic driver and device driver.
- Main function is to receive high level I/O requests.
  - **read_block 234** translated into a format that can be understood by device controller.
- Usually, each device has a requests queue.
  - A driver may attend multiple devices at the same time.
- Key function: I/O queues management.
  - Scheduling algorithm for I/O.
  - Queue request merging.

Dataflow in file system



Contents

- File System structure.
- File server.
- **Associated data structures.**
- Performance increase.

Associated data structures

- v-nodes table:
  - Single table with all v-nodes for all opened files.
- Open files table:
  - Per process table with one entry per opened file.
- Positions in open files table.
  - Single table with pointer positions in opened files.
- i-node tables.
  - Single table with i-nodes from all opened files.

V-nodes table

- Keeps table with all v-nodes for open files.

- It establishes a limit on maximum number of simultaneously open files.

- In each entry:
  - V-node information in disk.
  - Additional only in-memory information.

Open files table

- Table with one entry per open file.
- Table size limits maximum number of open files per process.
- Included in process PCB.
- Each entry keeps a pointer to a position in unique table of positions in open files.
- Table filled in order.
- Standard descriptors: 0, 1, and 2.
- Operations:
- Operaciones:
  - **Open** → Find first empty entry in table.
  - **Close** → Marks as empty an entry in table.
  - **Dup** → Copies value from an entry to the first empty entry.
  - **Fork** → Copy all table entries to a new process.

Positions in open files table

- Single table with information for each open file.
  - Pointer position in open file.
  - Pointer to file v-node.
  - Open mode for file.

Open i-nodes table

- Table with all i-nodes of open files.

- Establishes limit on maximum number of files simultaneously opened.

- In each entry:
  - Information of i-node on disk.
  - Additional only in-memory information.

Contents

- File System structure.
- File server.
- Associated data structures.
- **Performance increase.**

Performance improvement

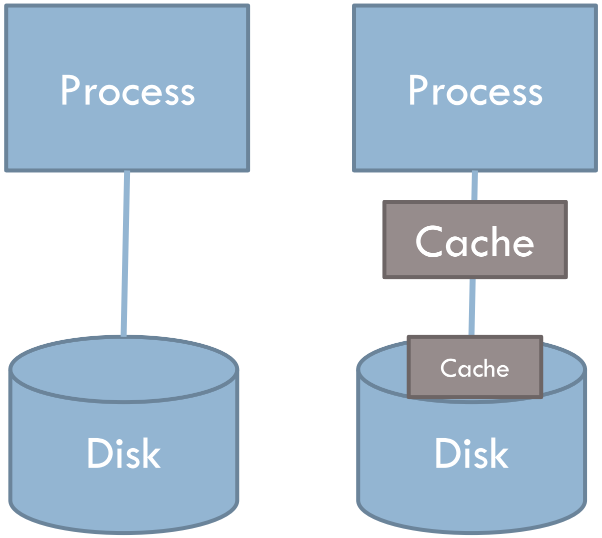
- Based in use of **intermediate storage** for I/O data in main memory.
- Two mechanisms:
  - **RAM disks**, data is stored only in memory.
    - Accept all operations from any other file system and are managed by user.
    - Pseudo devices for temporal storage of for auxiliary operations.
    - Content is volatile.
  - **Data cache**, data stored in sections of main memory under OS control.
    - Optimize later accesses.
    - Based in existence of **spatial and temporal locality** for I/O data.
    - Two important caches: **name cache** and **block cache**.

Block cache

- Fundamentals:
  - Spatial locality.
  - Temporal locality.
  - Two kinds of I/O flow:
    - A block used only once.
    - A block used repeatedly.
  - Disk access much slower that memory access.

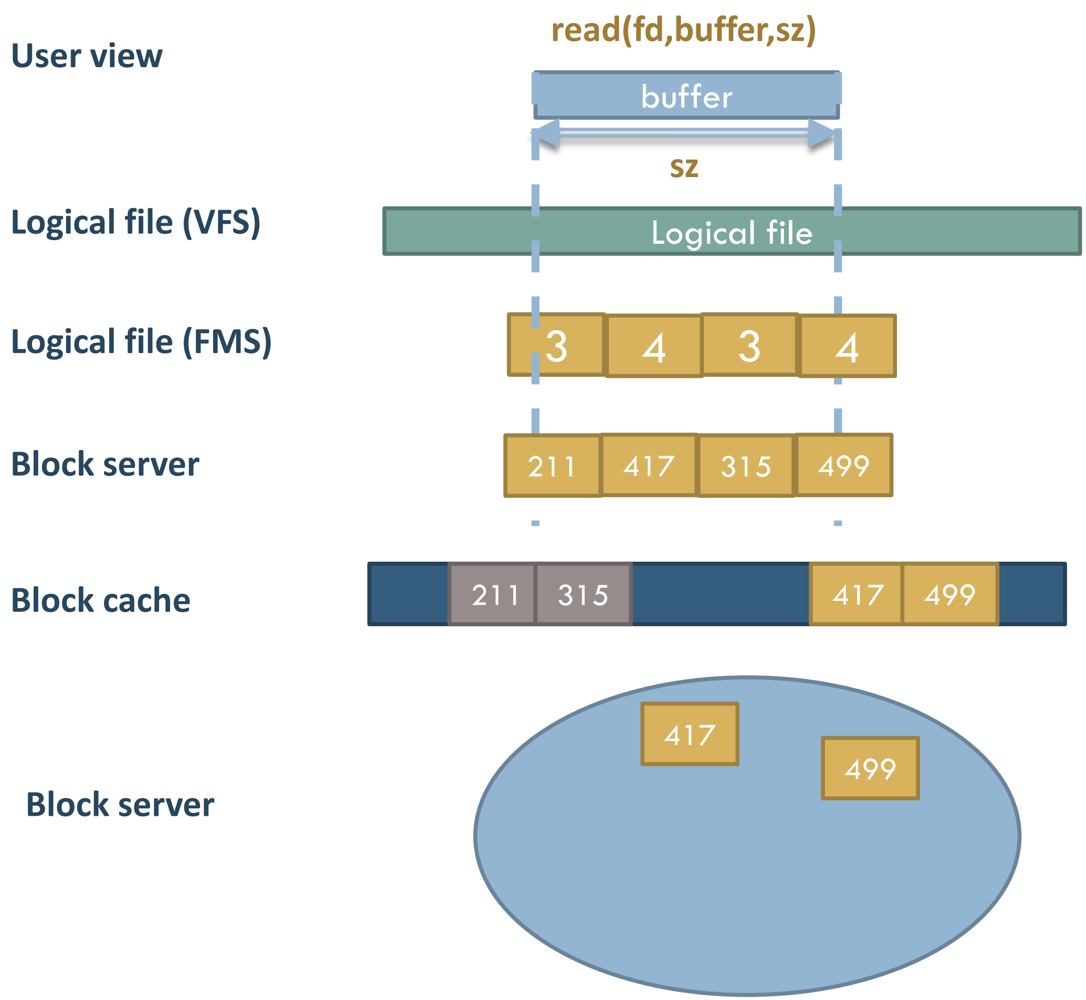
Block cache

- In-memory data structure with most frequently used blocks.



  - Prefetching.
  - Cache cleanup (sync)
- Main problem: File system reliability.

Data flow with block cache



Replacement policies

- Algorithm:
  - Check if block to be read is in cache.
    - If not found, read from device and copy to cache.
    - If cache is full, need to make room for new block replacing an existing one (**replacement policy**).
    - If block has been modified (dirty): **write policy**.

- Replacement policy: FIFO (*First in First Out*), second chance, MRU (*Most Recently Used*), LRU (*Least Recently Used*), etc.
  - Most commonly used is LRU.
  - Replaces block with more time since last use, assuming it will not be used in near future.
  - Most used blocks tend to be always in cache. Could lead to reliability problems on failure.

Write politics

- **Write-through**: Write each time block is used.
  - No reliability problem, but performance penalty.
- **Write-back**: Only writes to disk when block is replaced from cache.
  - Optimizes performance, but generates reliability problems.
- **Delayed write**: Write periodically modified blocks in cache (e.g. 30 seconds in some UNIX)
  - Performance/Reliability trade-off.
  - Reduces impact from possible damages from data loss.
  - Special blocks immediately written to disk.
  - Cannot un-mount disk without dumping cache data.
- **Write on close**: When file is closed, all blocks are dumped to disk