Operating Systems II

Filesystems

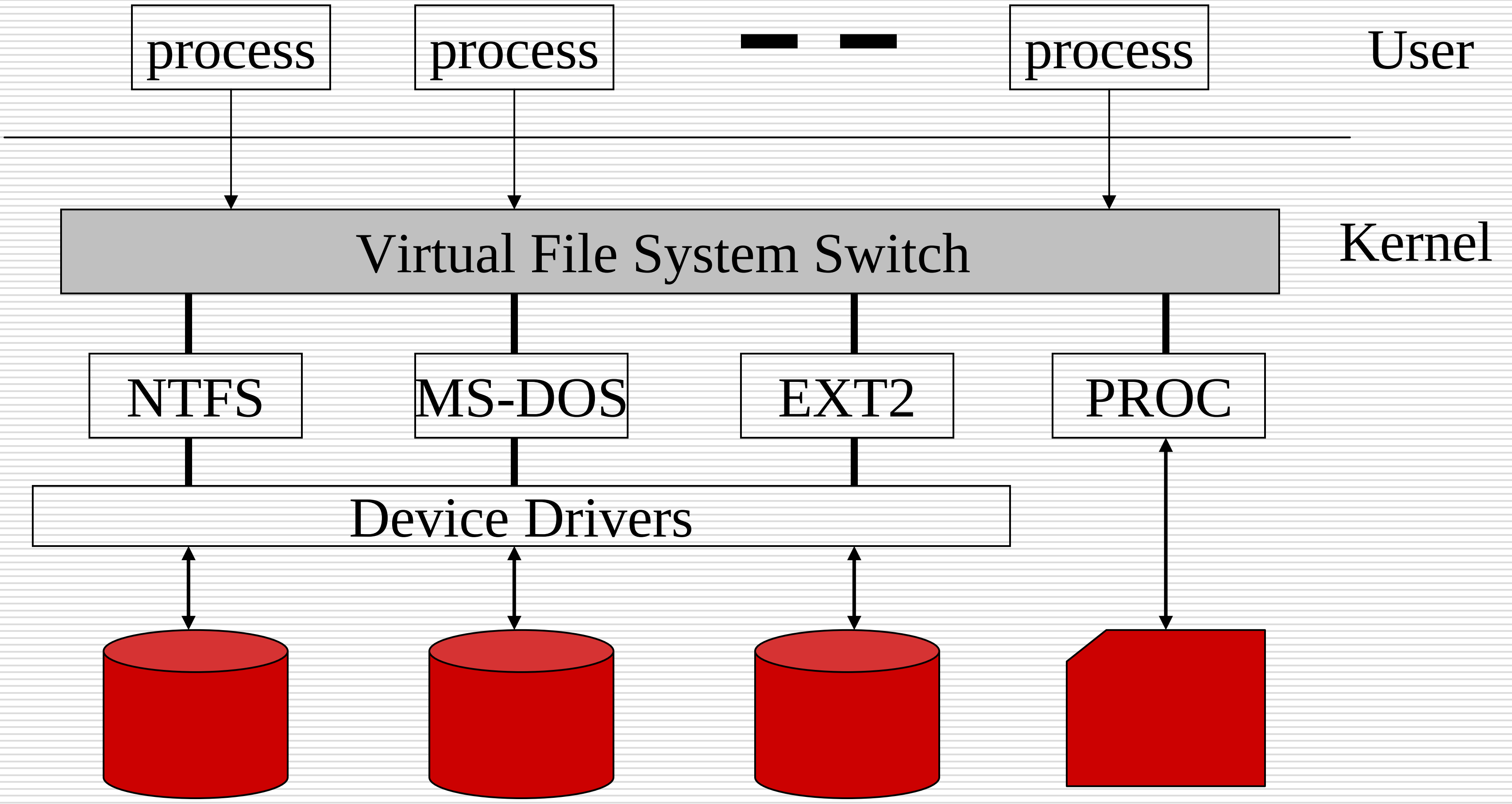
Filesystems

- Many different filesystems exist.
  - Each one has its benefits and penalties in performance, space, security, …
- One of the popular demands on operating systems is the support for different filesystems.
- The technique to achieve this is through the use of a Standard Filesystem Interface (SFI).
  - Linux uses the **Virtual File System Switch** (VFS) to accomplish this.

Overview

- First, the SFI is not a true filesystem! It does not directly manage files and directories.
- It is an interface that provides access between the processes and the different filesystems that exist.
- Specifically this includes:
  - system calls for file management
  - maintain internal structures
  - buffer between different filesystem types.

Example: VFS


process
process
process
User
Virtual File System Switch
Kernel
NTFS
MS-DOS
EXT2
PROC
Device Drivers

Duties of a Filesystem

- A filesystem is responsible for the "purposeful structuring" of data in the computer.
- What is *purposeful structuring*?
  - Speed to access (both sequential and random)
  - Organization (directories and sub-directories)
  - Utilization of the storage space available.
  - Other factors (i.e. security, data sharing, ...)

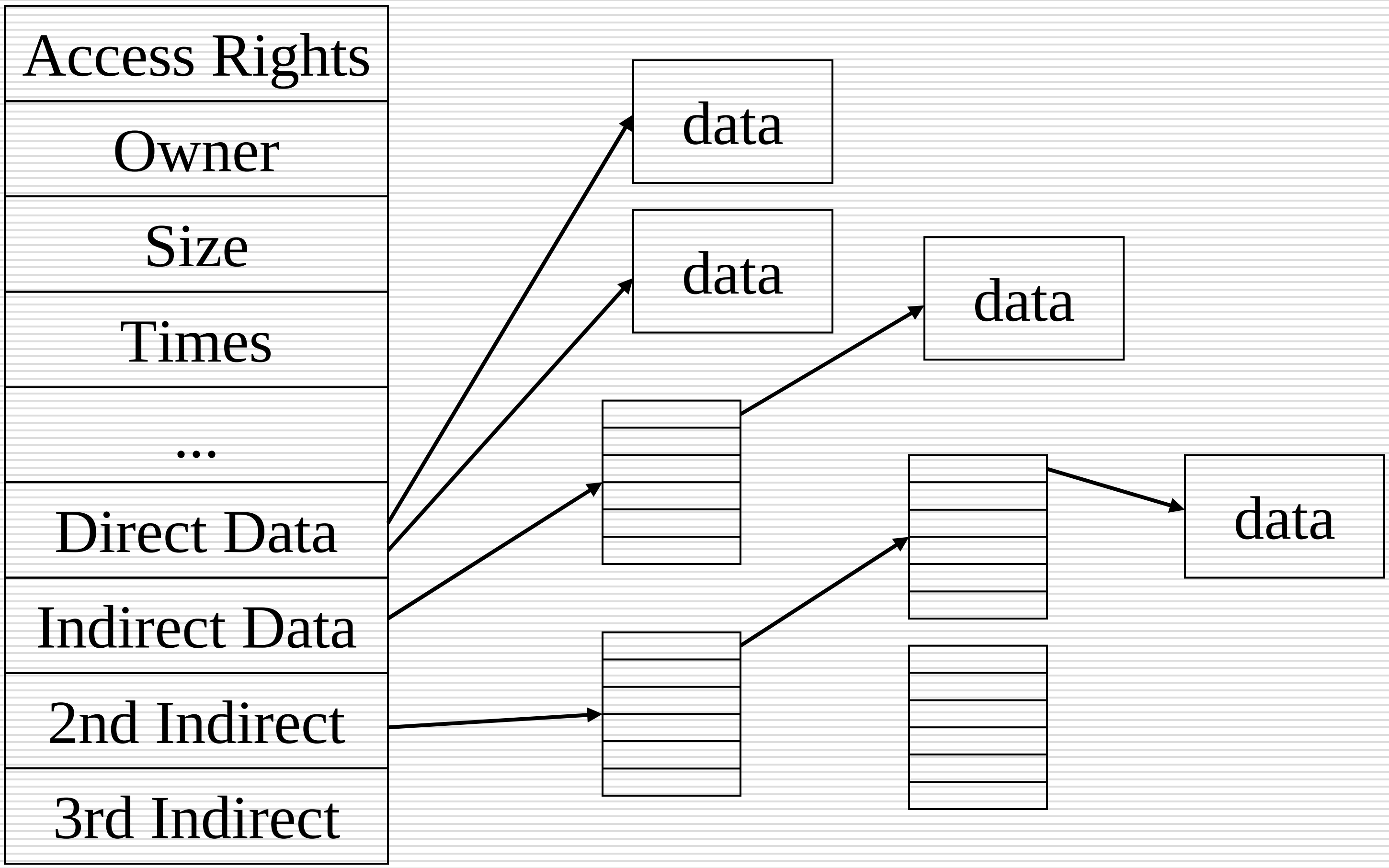
The Breakdown

- A file (or directory) can be looked upon as containing two parts.
  - First: the contents of the file that is a simple data flow.
  - Second: the control specification of the file (naming, security, length, ...)
- Each of these are split into sizes that can fit within blocks on the disk.

General Principle

- In general, filesystems have 4 types of blocks
  - boot block - first block of the device that contains the boot image.
  - superblock - holds all information vital for the management of the filesystem.
  - inodes - contains control information for a particular file.
  - data blocks - contains the actual data that is inside a file.

Example inode



Device Hierarchical Tree

- In some OS's, storage devices are organized by some device identifier number (drive letter or number).
- UNIX however organizes storage devices by a hierarchical directory tree.
- Devices are added to the tree to complete the filesystem.
- This process is known as **mounting** and **unmounting** filesystems.

Access through the SFI

- ANY manipulation of a filesystem goes through the SFI.
- On the highest level, this access is provided through different system calls.
  - `mount, umount, sysfs, statfs, fstatfs, ustat, chroot, pivot_root, chdir, fchdir, getcwd, mkdir, rmdir, getdents, readdir, link, unlink, rename, readlink, symlink, chown, fchown, lchown, chmod, fchmod, utime, stat, fstat, lstat, access, open, close, creat, umask, dup, dup2, fcntl, select, poll, truncate, ftruncate, lseek, _llseek, read, write, readv, writev, sendfile, readahead, pread, pwrite, mmap, munmap, madvise, mincore, fdatasync, sync, msync, flock`

Filesystems Everywhere

- The beauty of the SFI is that different filesystem types can be added.
- Just as with device drivers this is accomplished through a registration function.
  - `register_filesystem(struct file_system_type *fs)`
- A global variable `file_systems` gives the list of the registered filesystems.

file_system_type struct

- `char *name` - name of filesystem (e.g. "ext2")
- `int fs_flags` - flags of special features of filesystem. (e.g. no mount point, requires physical disk, ...)
- `struct super_block* (*read_super) (struct super_block *, void *, int)` - method for reading the superblock of filesystem.

file_system_type struct (cont.)

- struct module *owner - reference to module that implements the filesystem.
- struct file_system_type *next - pointer to next filesystem in the list.

Mounting

- There are two different ways of mounting a filesystem.
  - `mount_root()` function - This function is primarily used to mount the root filesystem on booting.
  - `mount()` system call - Used when the system is in a more booted state and filesystems are added onto the root.

Mounting Process

- Every mounted filesystem is represented by a superblock that is stored in a global list `super_blocks`.
- The list of superblocks is limited by the maximum `max_super_blocks`.
  - So there is a limit to how many filesystems one can have mounted.
- The `super_read` function of the SFI creates, adds, and initializes the `super_block` for a requested mount.

Mount Options

- Different mounting options can be specified for the filesystem.
  - MS_RDONLY - read only.
  - MS_NOSUID - no setuid.
  - MS_NODEV - no access to device files.
  - MS_NOEXEC - no execution access to files.
  - MS_SYNCHRONOUS - no caching of writes.
  - …

The Superblock

- The superblock holds information that is specific for a filesystem instance.
- This gives the capability to modify or change attributes that are specific to a filesystem.

super_block struct

- list_head – list of super_block structs (doubly linked).
- s_dev – device identifier (major and minor)
- s_blocksize – block size in bytes.
- s_blocksize_bits – block size in bits.
- s_dirt – modified or dirty flag.
- s_maxbytes – maximum size of the files.

super_block struct (cont)

- s_type – filesystem type.
- s_op – superblock methods.
- dq_op – disk quota methods.
- s_flags – mount flags (read-only, ...).
- s_magic – filesystem magic number.
- s_root – dentry object of root directory.
- s_umount – semaphore used for unounting.

super_block struct (cont)

- s_lock – superblock semaphore.
- s_wait – wait queue for s_lock semaphore.
- s_count – reference count to usage.
- s_active – secondary reference counter.
- s_dirty – list of modified inodes.
- s_locked_inodes – list of inodes in use (involved with I/O).

super_block struct (cont)

- s_dquot – options for disk quota.
- s_instances – list of superblocks that are the same filesystem type as this one.
- s_bdev – reference to block device driver descriptor.
- Other stuff ...

Locking the super_block

- There are functions provided for locking and unlocking the super_block before performing any modifications.
  - `lock_super(struct super_block *arg)`
  - `unlock_super(struct super_block *arg)`

super_block Control

- The general idea for the super_block is to maintain information of the mounted filesystem AND to control the next level of data - the inodes!
- inodes contain the information that is specific for files in the filesystem.
- Within the super_block, the s_op (struct super_operations) contains references to the functions that can be used to manipulate a super_block and manage the inodes.

super_operations

- read_inode(struct inode*) – reads a specific inode from the disk.
- write_inode(struct inode*) – fills in information into the inode specified.
- release_inode(struct inode*) – releases the inode object (does not delete).
- delete_inode(struct inode*) – deletes the inode and data blocks of the file associated.
- notify_change(struct inode*, struct iattr*) – grab attributes of the inode (??)

super_operations (cont)

- release_super(struct super_block*) – releases the super_block object since the filesystem is unmounted.
- write_super(struct super_block *) – update a filesystem super_block with contents of parameter.
- statfs(struct super_block*, statfsbuf, int) - return information of the status of the superblock.

super_operations (cont)

- remount_fs(struct super_block*, flags, options) – reset or restore the mounting of filesystem to some status.
- clear_inode(struct inode*) – clears the information from an inode (also updates quota).
- umount_begin(struct super_block *) – will begin to unmount a filesystem by making it inaccessible to others. Forces the unmounting to take place.

inodes

- As already discussed, inodes are used to contain the control information associated with each file/directory.
- As a result, there is a lot of information that is stored in the inode structure.

inode

- i_ino – inode number
- i_count – usage counter
- i_dev – device identifier
- i_mode – file type and access rights
- i_nlink – the number of hard links
- i_uid – owner identifier
- i_gid – group identifier
- i_rdev – real device identifier

inode (cont)

- `i_size` – flie length in bytes
- `i_atime` – time of last file access
- `i_mtime` – time of last file modification
- `i_ctime` – time of last inode change
- `i_blkbits` – block size in number of bits
- `i_blksize` – block size in number of bytes
- `i_blocks` – number of blocks of the file
- `i_version` – version number, automatically incremented after each use.

inode (cont)

- i_sem – inode semaphore
- i_zombie – secondary semaphore used when removing or renaming inode
- i_op – inode operations structure
- i_fop – default file operations
- i_sb – reference to super_block object
- i_wait – inode wait queue for semaphore

inode (cont)

- i_flock – reference to file lock list
- i_mapping – reference to an address_space object
- i_dquot – inode disk quotas
- i_pipe – used if the file is a pipe
- i_bdev – reference to the block device driver
- i_cdev – reference to the character device driver

inode (cont)

- i_state – inode state flags
- i_flags – filesystem mount flags
- i_sock – nonzero if the file is a socket
- i_writecount – usage counter for writing processes
- i_attr_flags – file creation flags
- i_generation – inode version number (used by some filesystems)
- u – specific filesystem information

inode (cont)

- Also lots of information for quick access between files...
  - `i_hash` – pointers for the hash list
  - `i_list` – pointers for the inode list
  - `i_dentry` – pointers for the directory list
  - `i_dirty_buffers` – pointers for the modified buffers list
  - `i_dirty_data_buffers` – pointers for the modified data buffers list

inode locations

- Within the kernel, an inode structure is located in one of 3 places.
  - The list of in-use inodes that reflect some inode on the disk that is being used by a process.
  - The list of dirty inodes that need to be written to disk.
  - The list of valid unused inodes typically reflecting some inode on disk, but not being used by a process. Kept as a caching mechanism.

inode operations

- create - acquires a free inode and intializes it with a specific file/dir information.
- link - used to create a hard link for an inode between two dentries.
- unlink - removes an inode from a dentry.
- symlink - establishes a symbolic link between an inode and a dentry.

inode operations (cont)

- mkdir - creates a new directory given the inode of the parent directory.
- rmdir -remove a directory.
- mknod - sets up a new inode for a special file.
- rename - moves an inode from one dentry to another with a name change.
- readlink - reads the absolute filename of a link.

inode operations (cont)

- follow_link - resolves a symbolic link to appropriate inode.
- truncate - modifies the size of the file associated with the inode.
- permission - checks if permissions allow access to inode.

inode operations (cont)

- revalidate - updates the disk with cached attributes of a file (useful for network filesystems).
- setattr - notifies a change event after touching the inode attributes.
- getattr - gets the attributes of an inode that needs to be refreshed (useful for network filesystems)

`file` and `dentry`

- The kernel views the filesystem in terms of inodes, blocks, and superblocks.
- However, this is not how users view filesystems. They view the filesystem as a group of `file` and `dentry` types.
  - `dentry` represents a directory entry to the user.

file Structure

- f_op – (file_operations) structure
- f_count – file objects usage counter
- f_flags – flags specified when opening the file
- f_mode – process access mode
- f_pos – current position in file
- f_uid – file owner's userid
- f_gid – file owner's group id
- f_error – error code for network write operation
- …

`file` Operations

- Remember this from device drivers?
  - Same structure with the same operations!
- Operations include:
  - `llseek, read, write, readdir, poll, ioctl, mmap, open, flush, release, fsync, fasync, lock, readv, writev, sendpage, get_unmapped_area`

dentry Structure

- d_count – usage counter
- d_flags – directory flags
- d_inode – inode associated with the dentry
- d_parent – the parent directory of this directory
- d_child – list of dentrys in the parent (siblings)
- d_subdirs – list of sub directories (dentries)

dentry Structure

- d_mounted – flag to indicate if root of mounted filesystem
- d_name – directory name
- d_op – dentry methods
- d_sb – superblock of the dentry
- d_fsdata – filesystem dependent data
- d_alias – list of associated inodes (files)
- d_hash – pointers for list in hash table entry (files)
- …

`dentry` State

- Each dentry that exists in the system can be in 1 of 4 different states.
  - **Free** – dentry object is free containing no valid information and is managed by a slab allocator.
  - **Unused** – It sits in memory refering to a valid inode, but no process is using it. Can be freed if necessary.
  - **In use** – currently being used by a process and refers to an inode and contains valid info.
  - **Negative** – The inode with the dentry no longer exists. Either the inode was blown away OR the dentry was created with reference to an existing inode. Substate of unused.

dentry Operations

- d_revalidate – verifies the dentry is still valid before use. Useful for network filesystems.
- d_hash – creates a hash value for the dentry hash table.
- d_compare – compares two filenames. (ie MSDOS does not distinguish between upper/lower case, others do!)

dentry Operations

- d_release – called when the dentry object is going to be frred back to the slab allocator.
- d_delete – called when the last reference to the dentry is released. Default VFS does nothing.
- d_iput – called when a dentry object loses its inode – releases the inode object.

Case Study

- Originally Linux used the minix filesystem
- It was quickly replaced because of its restrictions:
  - Only supported partitions up to 64 Mb.
  - Filenames could only be up to 14 characters in length.
- This was replaced by the ext filesystem which provided much better features, though poor performance.

ext2 filesystem

- The ext filesystem was criticized for its poor performance and quickly replaced by ext2.
- Features of the ext2 filesystem include:
  - Administrator choice of block size on partition creation.
  - Choice of how many inodes to allow for a partition of a given size.

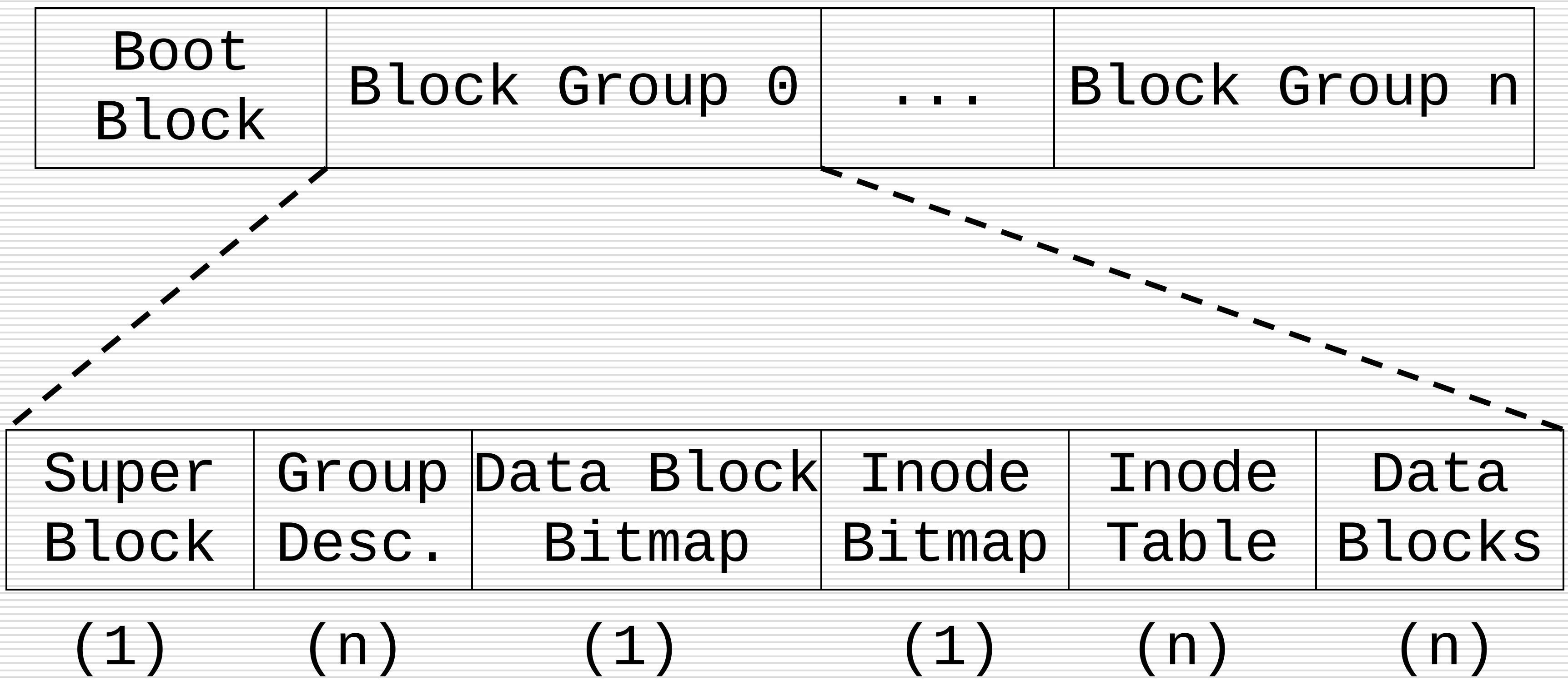
ext2 filesystem (cont.)

- Partitions disk blocks into groups. Each group includes data blocks and inodes stored in adjacent tracks.
- Preallocates disk blocks to regular files before they are actually used. Thus, when files increase, the adjacent blocks are readily available. Reduces disk fragmentation.
- Fast symbolic links. If 60 bytes or less can be resolved without reading a data block.

ext2 filesystem (cont.)

- A careful implementation of the file updating strategy that minimizes the impact of system crashes.
- Support for automatic consistency checks on the filesystem status at boot time.
- Support for immutable files (they cannot be modified, deleted, or renamed)
- Compatibility for use with different unix systems. (e.g. group id inheritance)

ext2 Disk Structure



The ext2 Superblock

<table>
<tr><td colspan="2">Number of inodes</td><td colspan="2">Number of blocks</td></tr>
<tr><td colspan="2"># of reserved blks</td><td colspan="2"># of free blocks</td></tr>
<tr><td colspan="2"># of free inodes</td><td colspan="2">First data block</td></tr>
<tr><td colspan="2">Block size</td><td colspan="2">fragment size</td></tr>
<tr><td colspan="2">Blocks per group</td><td colspan="2">Fragments per group</td></tr>
<tr><td colspan="2">Inodes per group</td><td colspan="2">Mounting time</td></tr>
<tr><td colspan="2">Time of last write</td><td>Mount Count</td><td>Max # mnt</td></tr>
<tr><td>Ext2 sign</td><td>status</td><td>Err handling</td><td>Minor rev</td></tr>
<tr><td colspan="2">Time of last test</td><td colspan="2">Max test time interval</td></tr>
<tr><td colspan="2">Operating system</td><td colspan="2">File system revision</td></tr>
<tr><td>RESUID</td><td>RESGID</td><td colspan="2"></td></tr>
</table>

The ext2 Block Group Descriptors

<table>
<tr><td colspan="2">Block Bitmap</td><td colspan="2">Inode Bitmap</td></tr>
<tr><td colspan="2">Inode Table</td><td># free blocks</td><td># free nodes</td></tr>
<tr><td># directories</td><td></td><td></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td></tr>
</table>

The remaining space is left for future development and additional features.

The ext2 inode

<table>
<tr><td>Type/rights</td><td>User (uid)</td><td colspan="2">File size</td></tr>
<tr><td colspan="2">Access time</td><td colspan="2">Time of creation</td></tr>
<tr><td colspan="2">Time of modification</td><td colspan="2">Time of deletion</td></tr>
<tr><td>Group (GID)</td><td>Link Count</td><td colspan="2"># of blocks</td></tr>
<tr><td colspan="2">File Attributes</td><td colspan="2">Reserved (OS)</td></tr>
<tr><td colspan="4">12 direct blocks</td></tr>
<tr><td colspan="2">Single indirect</td><td colspan="2">Duplicate indirect</td></tr>
<tr><td colspan="2">Triplicate indirect</td><td colspan="2">File version</td></tr>
<tr><td colspan="2">File ACL</td><td colspan="2">Directory ACL</td></tr>
<tr><td colspan="2">Fragment Address</td><td colspan="2" rowspan="2">Reserved (OS)</td></tr>
<tr><td colspan="2">Reserved (OS)</td></tr>
</table>

Replication

- The ext2 filesystem replicates data in each of its groups that is used for the full partition
  - superblock and group descriptor.
- The purpose of this is for disk recovery due to inconsistencies from system crashes.
- The system utility e2fsck can be used to reset the main superblock from the replicas.

How many groups?

- The number of block groups that exist is dependent on both the partition and block size.
- The main constraint is the block bitmap (each bit tells if the data block is free or in use)
- This gives the number of block groups to be roughly (partition blocks / 8 * bytes in a block)
- For example: 8Gb partition with 4K block size will require roughly   blocks.

Data Block Allocation

- Data blocks as previously mentioned are allocated in groups and pre-allocated at that to provide quick access to files.
- When another data block is needed, the search is not for just one more block, but instead for a set of blocks.
  - Typically this is 8 blocks.

Data Block Allocation (cont)

- When a file needs more data blocks, the search for data blocks proceeds in the following order:
  - Check to see if any unused pre-allocated blocks are available.
  - Checks for any free blocks adjacent to the currently used blocks for the file.
  - Checks for any free blocks in the current block group.
  - Checks for any free blocks in other block groups.

Extensions for ext2

- Block Fragmentation - Files tend to be on either extreme really small or really large in comparison to the actual block size. Desire to place several small files into the one disk block.
- Access Control Lists - Greater control over file permissions than the owner, group, world that is now used.

Extensions for ext2

- Logical Deletion - provision of an undelete option.
- Journaling - avoids the time consuming check that is automatically performed on filesystems when they are abruptly unmounted.
- File features - Handling transparently compressed and encrypted files.