ECE391
Computer System Engineering

Lecture 16

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

Fall 2022

Lecture Topics

- File system
  - Linux file structure
  - file operations
  - ext2 filesystem (on disk)

- MP3 file system

Aministrivia

- **MP3 Checkpoint 1**
  - **Due by 6:00pm Monday, 17 October**

Virtual File System

FAT32
(File Allocation Table32)

EXT4
(Fourth Extended File System)

F2FS
(Flash-Friendly File System)

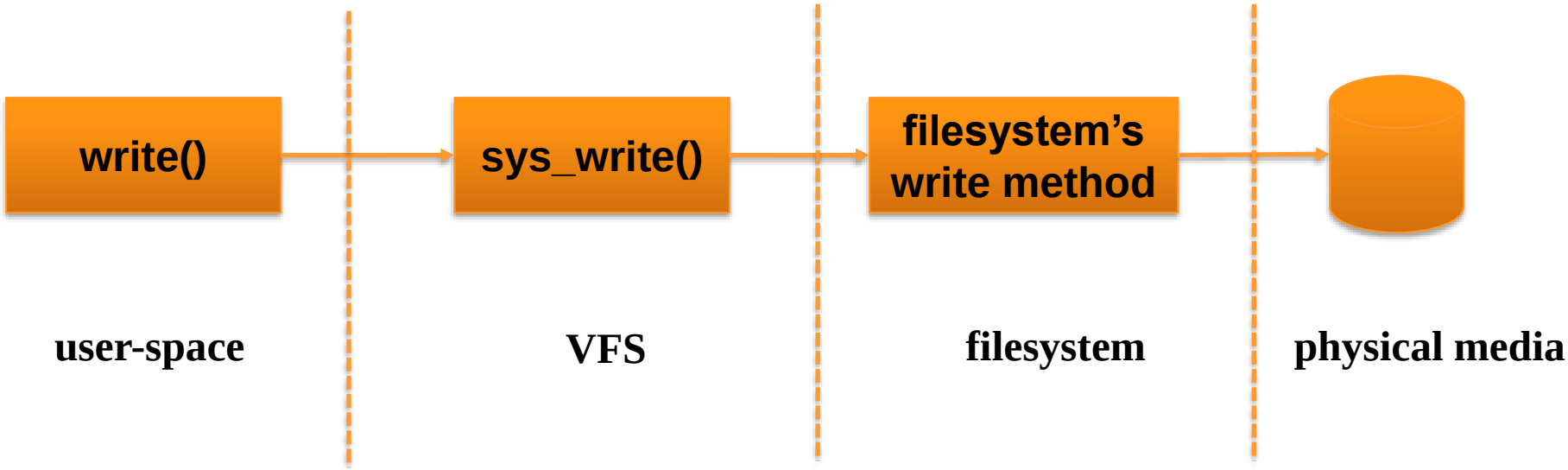
PMFS
(Persistent Memory File System)

File System Abstraction Layer

- VFS provides common file system interfaces
  - create, open, read, write, etc.
- Each file system implements the real functionalities on top of the device drivers

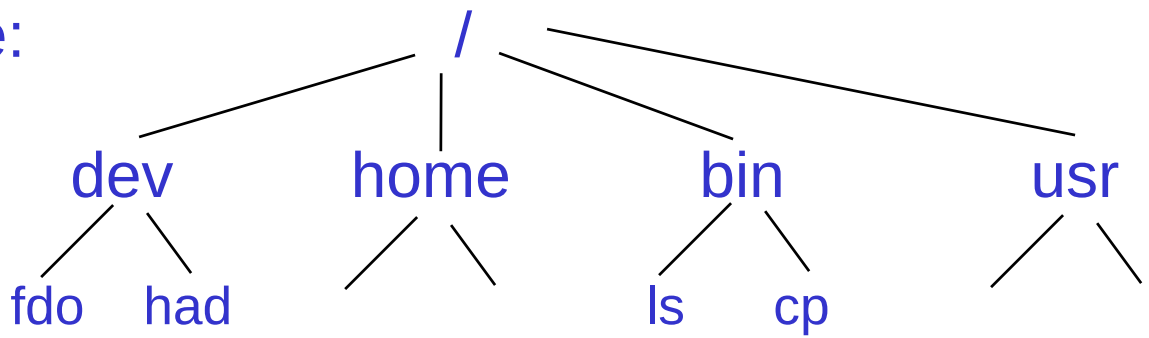
An Example

- **write (fd, &buf, len)**



File System

- A Unix-like file system is an information container structured as a sequence of bytes
- Files are organize in a tree structured namespace
- Example:



- All nodes (except leaves) denote directories
- The directory corresponding to the root is called the root directory [slash (/)]

File Types

- Regular files
- Directory
- Symbolic Link
- Block-oriented device file
- Character-oriented device file
- Pipe and named pipes
- Socket

File Descriptor and Inode

- There is clear distinction between the contents of a file and the information about a file
  - With the exception of device files and files of special file systems, each file contains a sequence of bytes

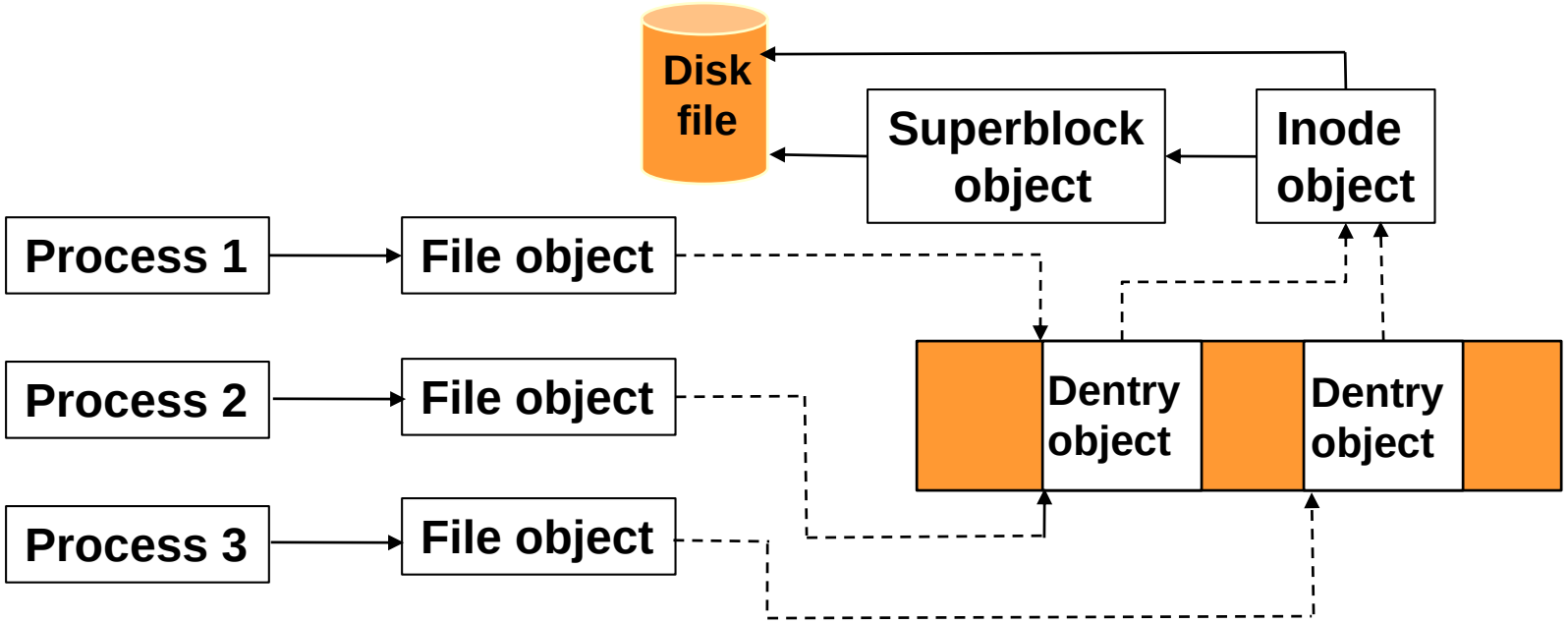
- *Inode* contains Information needed by the file system to handle a file (access permissions, size, owner..)
  - each file has its own *inode*
- File descriptor is an index to a data structure (at the kernel level) containing the details of all open files
  - created by a process when a file is opened

File Descriptor and Inode

- superblock object – represents a specific mounted file system
- inode object – represents a specific file
- dentry object – represents a directory entry, a single component of a path
- file object – represents an open file as associated with a process (exits only in the kernel memory)

  - All objects are stored in a suitable data structure
    - includes object attributes and a pointer to a table of object methods

Interaction – Processes and Filesystem Objects



- Processes open the same file
- Two of them use the same link (i.e., same dentry)
- Each uses its own file object

Linux File Structure *(file object)*

```
struct file {
        /*
         * fu_list becomes invalid after file_free is called and queued via
         * fu_rcuhead for RCU freeing
         */
        union {
                struct list_head        fu_list;
                struct rcu_head         fu_rcuhead;
        } f_u;

        struct path             f_path;
#define f_dentry        f_path.dentry
#define f_vfsmnt        f_path.mnt
        const struct file_operations    *f_op;
        atomic_t                f_count;
        unsigned int            f_flags;
        mode_t                  f_mode;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        unsigned int            f_uid, f_gid;
        struct file_ra_state    f_ra;

        unsigned long           f_version;

        void                    *f_security;

        /* needed for tty driver, and maybe others */
        void                    *private_data;

        /* Used by fs/eventpoll.c to link all the hooks to this file */
        struct list_head        f_ep_links;
        spinlock_t              f_ep_lock;

        struct address_space    *f_mapping;
};
```

directory entry

virtual file system (VFS) mount point

file operations structure

asynchronous I/O and other flags

file position

data allocated and released by associated driver

- The file structure (file object) is created when the file is opened
- No corresponding image on the disk

Comments on Linux's internal file structure

f_u	list of files / read-copy-update cleanup list
(these two are macros; part of struct path f_path)	
f_dentry	directory entry
f_vfsmnt	virtual filesystem (VFS) mount point
f_op	file operations structure (ops explained later)
f_count	# of openers and uses (system calls) in progress
f_flags	asynchronous I/O and other flags
f_mode	read/write modes (user, group, other)

Comments on Linux's internal file structure

`f_pos` file position

`f_owner` for FASYNC signaling (async. I/O signals to user processes)

`f_uid, f_gid` user and group ID of file's opener

`f_ra` file read ahead control fields

`f_version` used for some types of files (FIFO, pipes, etc.)

`f_security` generic data block for more advanced security model support (e.g., capabilities); see **`fs/security.h`**

Comments on Linux's internal file structure

`f_private_data` data allocated and released by associated driver

`f_ep_links` support for Linux-specific variant of poll: see **`epoll(7)`**

`f_ep_lock`

`f_mapping` address space into which file is memory-mapped

File Operations

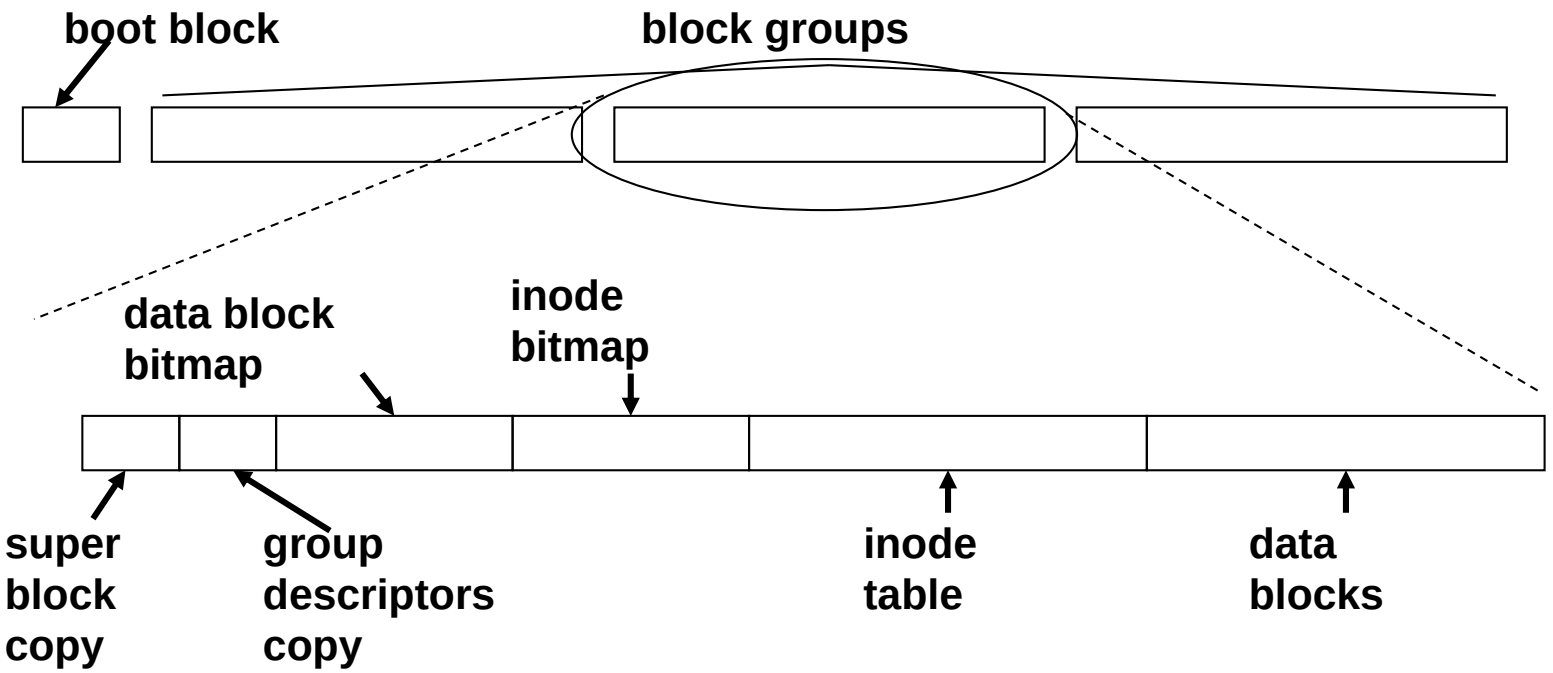
- File operations structure
  - jump table of file operations / character driver operations
  - generic instance for files on disk
  - distinct instances for sockets, etc.
  - one instance per device type

File System on Disk

- Disks are slow
  - fast ones rotate at ~10,000 rpm
  - average of (60 sec/10000)/2 = 3 ms to wait for disk to turn
  - another few milliseconds to move read head side to side
  - 5-10 millisecond access time

- Disks use blocks (and prefetching) to alleviate problems
  - filesystems can use several adjacent disk blocks to further improve
  - ext2 allows 1- 4kB blocks (chosen when filesystem is created)

ext2 (Second Extended Filesystem) on Disk



ext2 on Disk

- ext2 blocks are fixed size (1kB to 4kB) chosen at creation time
- Groups are fixed size, also chosen at creation time
- Two blocks replicated in each block group for reliability
  - super block describes filesystem (e.g., choice of block size)
  - group descriptors describes block groups
- bitmaps used to represent free blocks for index nodes (metadata) and data

Superblock Image on Disk

- Sized at 1kB, so fits within any selected block size
- Sizes selected for filesystem
- Filesystem check information
  - # mounts between checks, time between checks, errors found
  - also state of filesystem
    - 0 when mounted/uncleanly dismounted
    - 1 when cleanly dismounted
    - 2 if errors have been found
- Reserved blocks & authentication data
- Volume name
- Performance specs (e.g., preallocation)

Group Descriptor Image on Disk

- Sized at 32B, so 32 of them fit within any selected block size
- Shortcuts to block/inode bitmaps and inode table/data blocks
- Free block counts

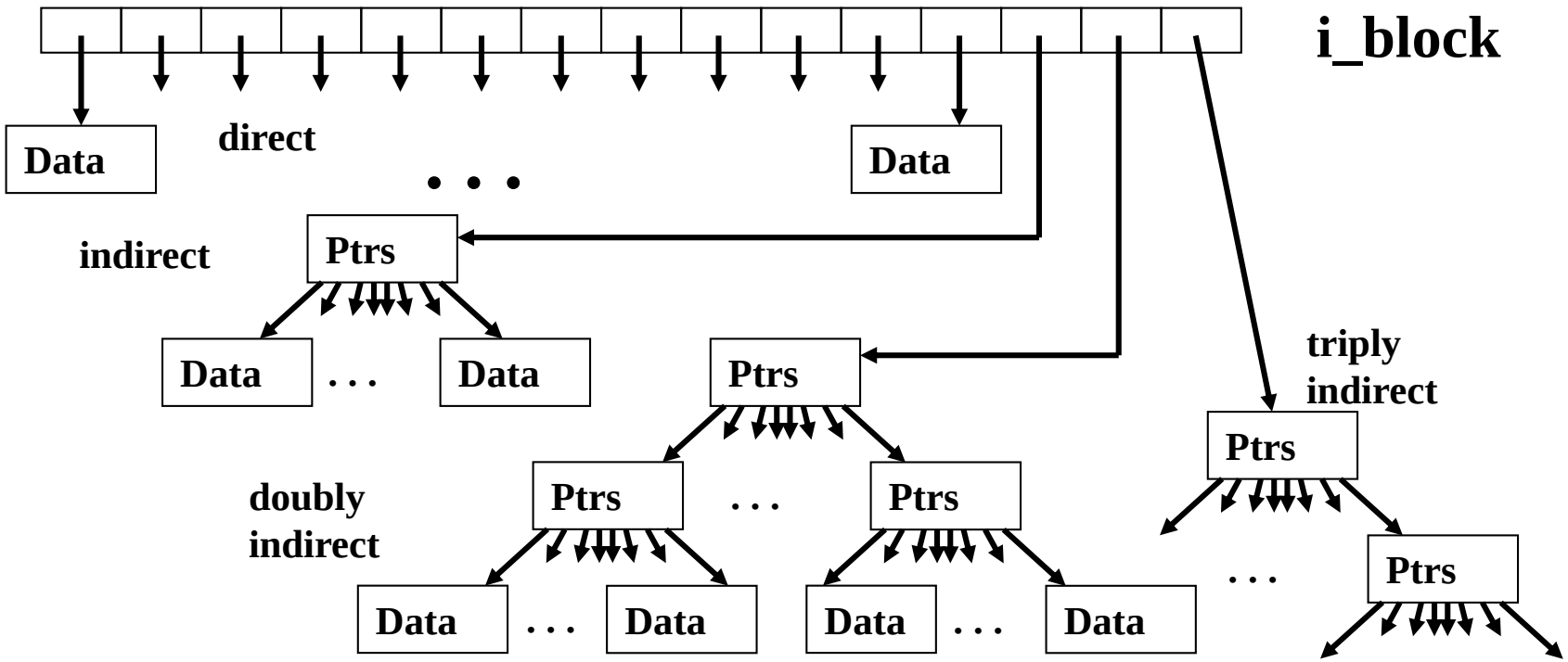
Index Node on Disk (128B total)

`i_mode`	file type & access rights (e.g., readable by group)
`i_uid`	owner
`i_size`	length in bytes
`i_atime`	time of last access
`i_ctime`	time of last creation
`i_mtime`	time of last modification
`i_dtime`	time of last deletion
`i_gid`	group id
`i_links_count`	# of hard links
`i_blocks`	blocks in file (disk blocks, in 512B units)
`i_flags`	e.g., immutable, append-only
`i_block[15]`	data block #'s
`i_generation`	generation (incremented on each modification)

File Data Blocks Stored Hierarchically

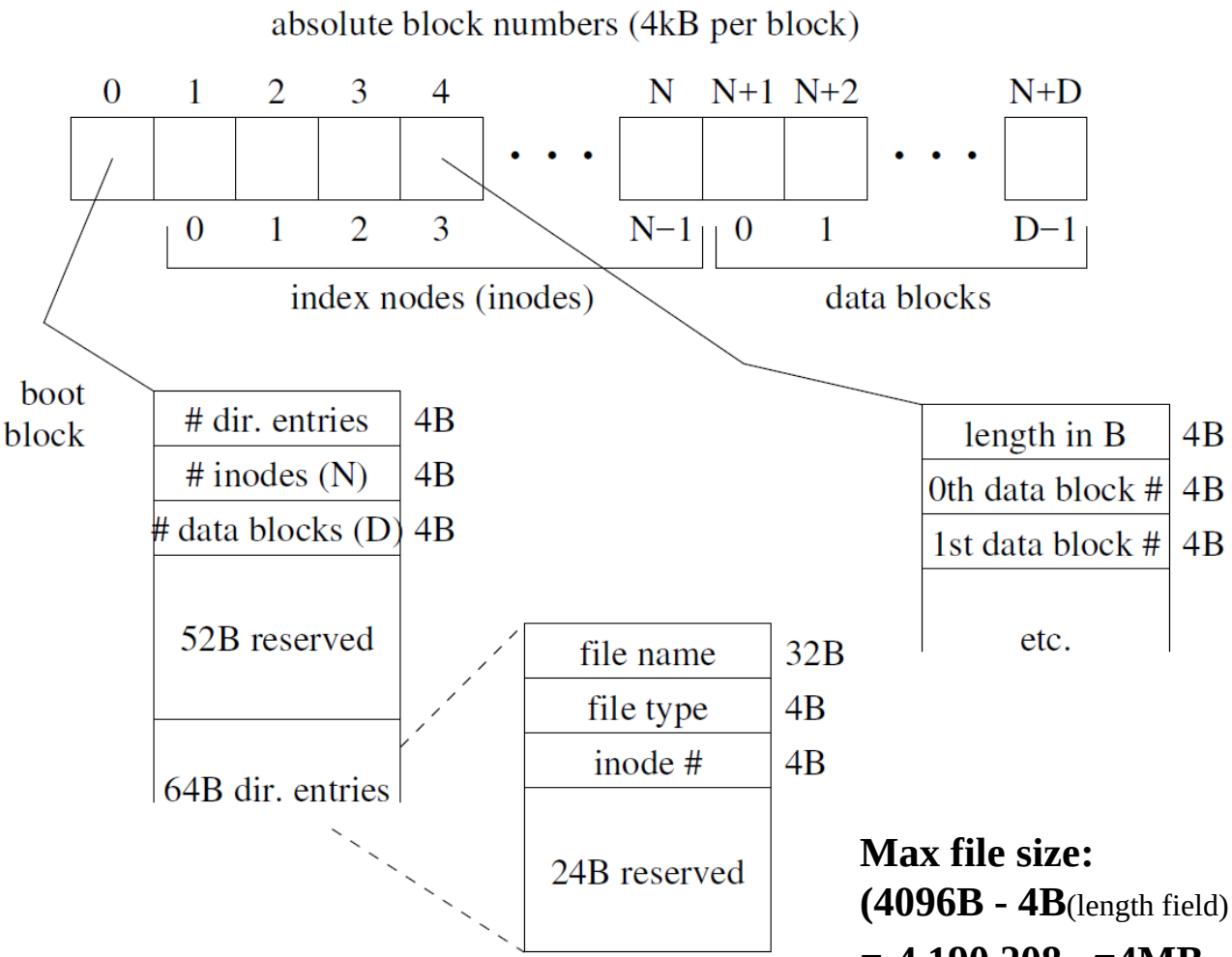
- let ***b*** denote block size
- first 12 block pointers are direct: block #'s
  (first 12***b*** bytes of file)
- 13th block pointer is indirect: block # of a block of block #'s
  (***b***/4 pointers, so the next $b^2/4$ data bytes in file)
- 14th block pointer is doubly indirect (next $b^3/16$ data bytes in file)
- 15th block pointer is triply indirect (last $b^4/64$ data bytes in file)
- small files only need direct blocks

File Data Blocks



block size b	4 kB	2 kB	1 kB
direct	48 kB	24 kB	12 kB
indirect	4 MB	1 MB	256 kB
doubly ind.	4 GB	512 MB	64 MB
triply ind.	4 TB	256 GB	16 GB

MP3 File System



Max file size:
(4096B - 4B(length field)**) / 4B) * 4096B =**
= 4,190,208 ~=4MB

Max number of files:
(4096B / 64B) – 1(statistics)**– 1**(first directory entry)**= 62**

MP3 File System: example declaration of data structures

- Memory file system -> all data kept in memory

- boot block:
  - int32_t dir_count;
  - int32_t inode_count;
  - int32_t data_count;
  - int8_t reserved[52];
  - <dentry> direntries[63];

- dentry:
  - int8_t filename[FILENAME_LEN];
  - int32_t filetype;
  - int32_t inode_num;
  - int8_t reserved[24];

- inode:
  - int32_t length;
  - int32_t data_block_num [1023];

MP3 File System Utilities (used by the kernel)

```
int32_t read_dentry_by_name (const uint8_t* fname, dentry_t* dentry);
int32_t read_dentry_by_index (uint32_t index, dentry_t* dentry);
int32_t read_data (uint32_t inode, uint32_t offset, uint8_t* buf, uint32_t length);
```

read_dentry_by_name () {

< Scans through the directory entries in the “boot block” to find the file name >

Call *read_dentry_by_index() {*

< Populates the dentry parameter -> file name, file type, inode number >

}

}

You call *read_dentry_by_name ()* in *sys_open()* *system call*

< open a file; set up the file object -> inode, fops, flags, position >

MP3 File System Abstractions

- **Each task can have up to 8 open files**
- **Open files are represented with a file array (in a Process Control Block; PCB)**
- **File array is indexed by *a file descriptor***