

File Systems

CS110 Discussion Section 1

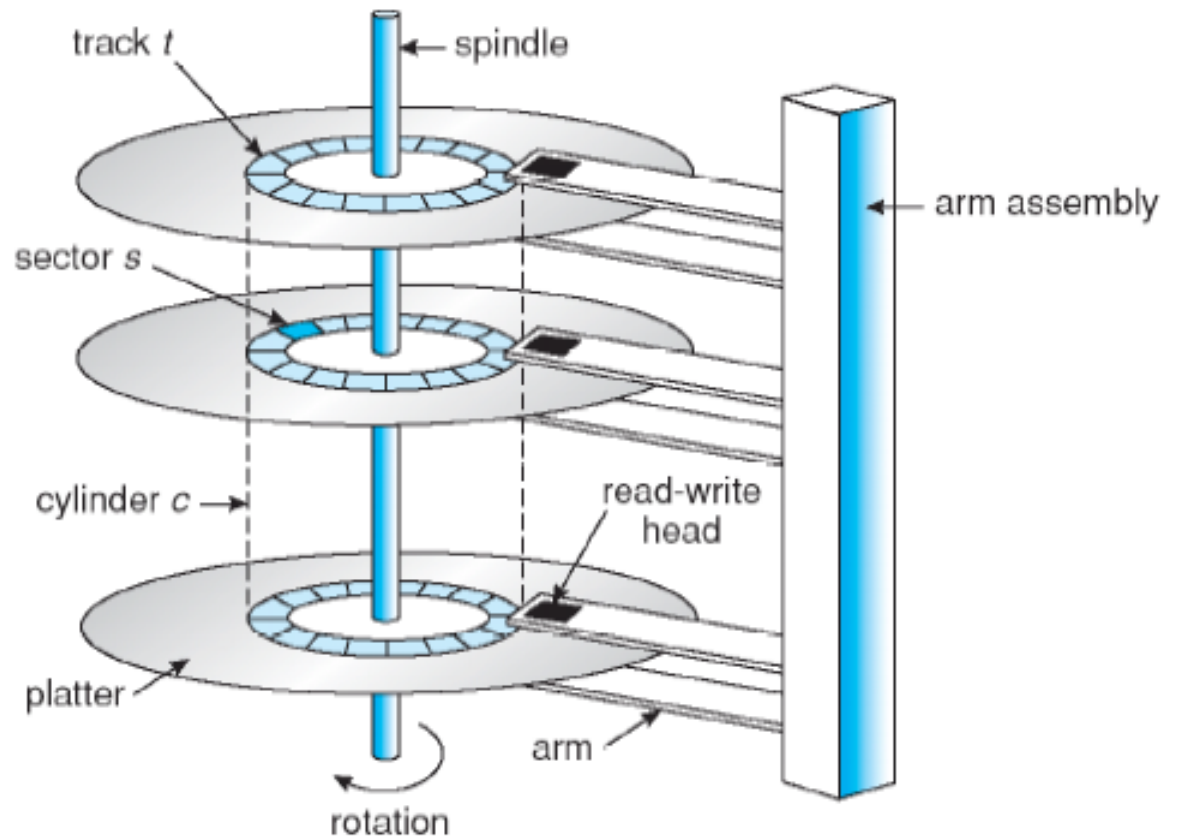
Agenda

- Disks
- Files
- File system abstraction
- Unix Version 6 file system
- Assignment 1

Magnetic Disks

Access method:

- 1) Seek - position disk arm over track
- 2) Wait until desired sector rotates under disk head
- 3) Read one or more sectors from track

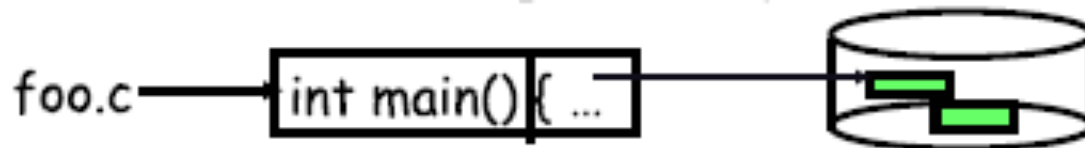


Disk Sector Name Space

- 1970s disks:
 - Specify cylinder, head, sector
 - Example: cylinder = 20, head = 4, sector = 4
 - Unix Version 6 – int16 sector number
- 2000s disks: (SCSI)
 - Specify sector number (0 ... #sectors on disk-1)
 - Example: sector 1

Files

- File: Collection of zero or more bytes
- Stored in hierarchical space
 - Example: /a/test/foo.c
- Operations like:
 - read() - Read byte range from file
 - write() - Write byte range to file



File System Abstraction

- Implement the file abstraction on top of the disk abstraction
 - Pack files onto disk
 - Include file system “metadata” on disk to find files
- Translate name & offset to disk blocks

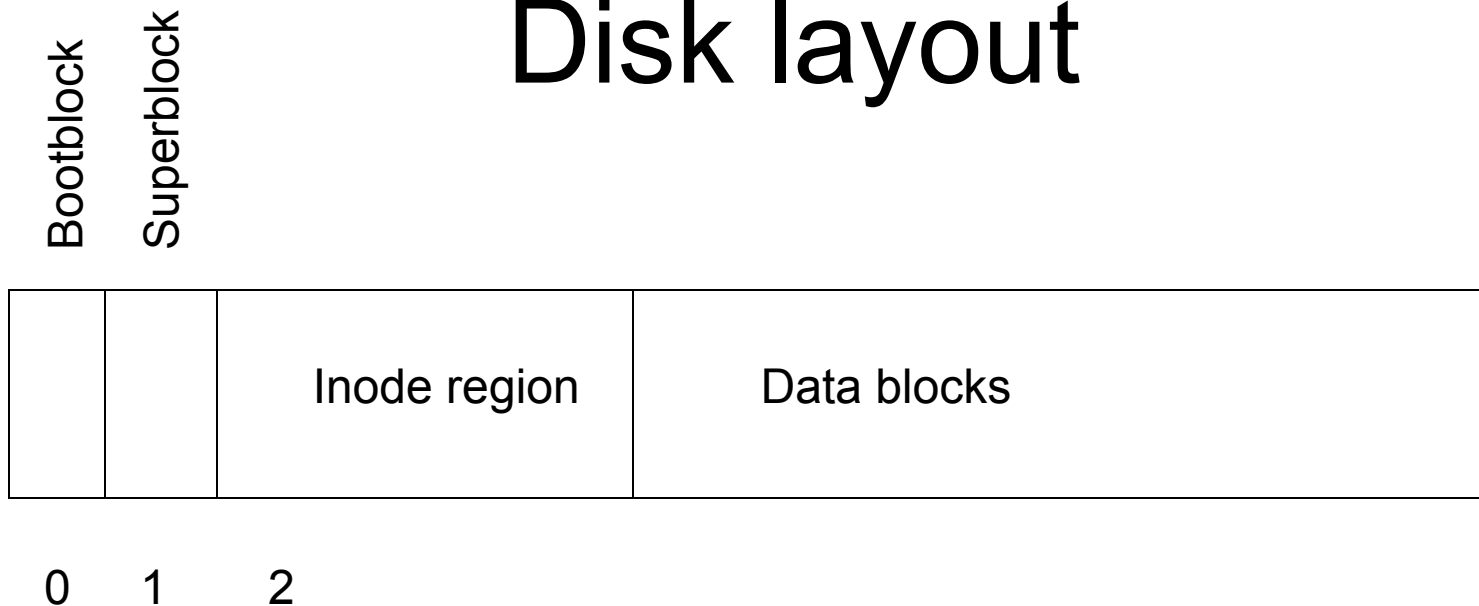


Unix Version 6 File System

Metadata

- Data structures to track files on disk and map files into disk blocks
 - Superblock
 - Describes layout of file system: size, free space, etc
 - Located in Sector 1 (Unix Version 6)
 - Inodes
 - Contains file attributes (size, access times, etc.), block numbers storing the file

Disk layout

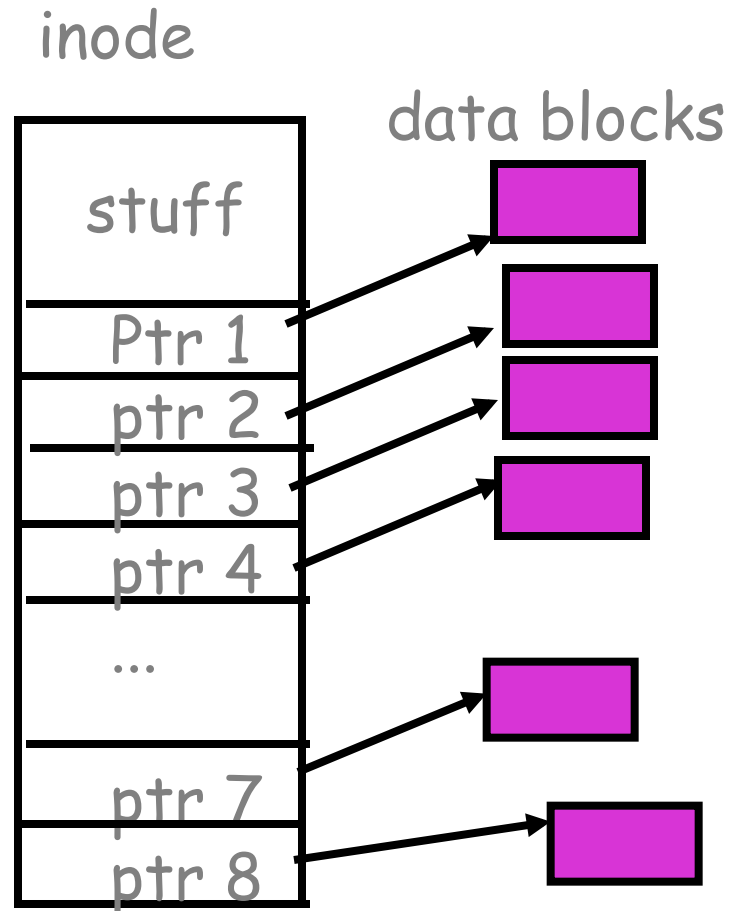


- Superblock describes layout of file system
- Inode region starts at 2 and its size is specified in Superblock
- Data blocks start after Inode region

inode

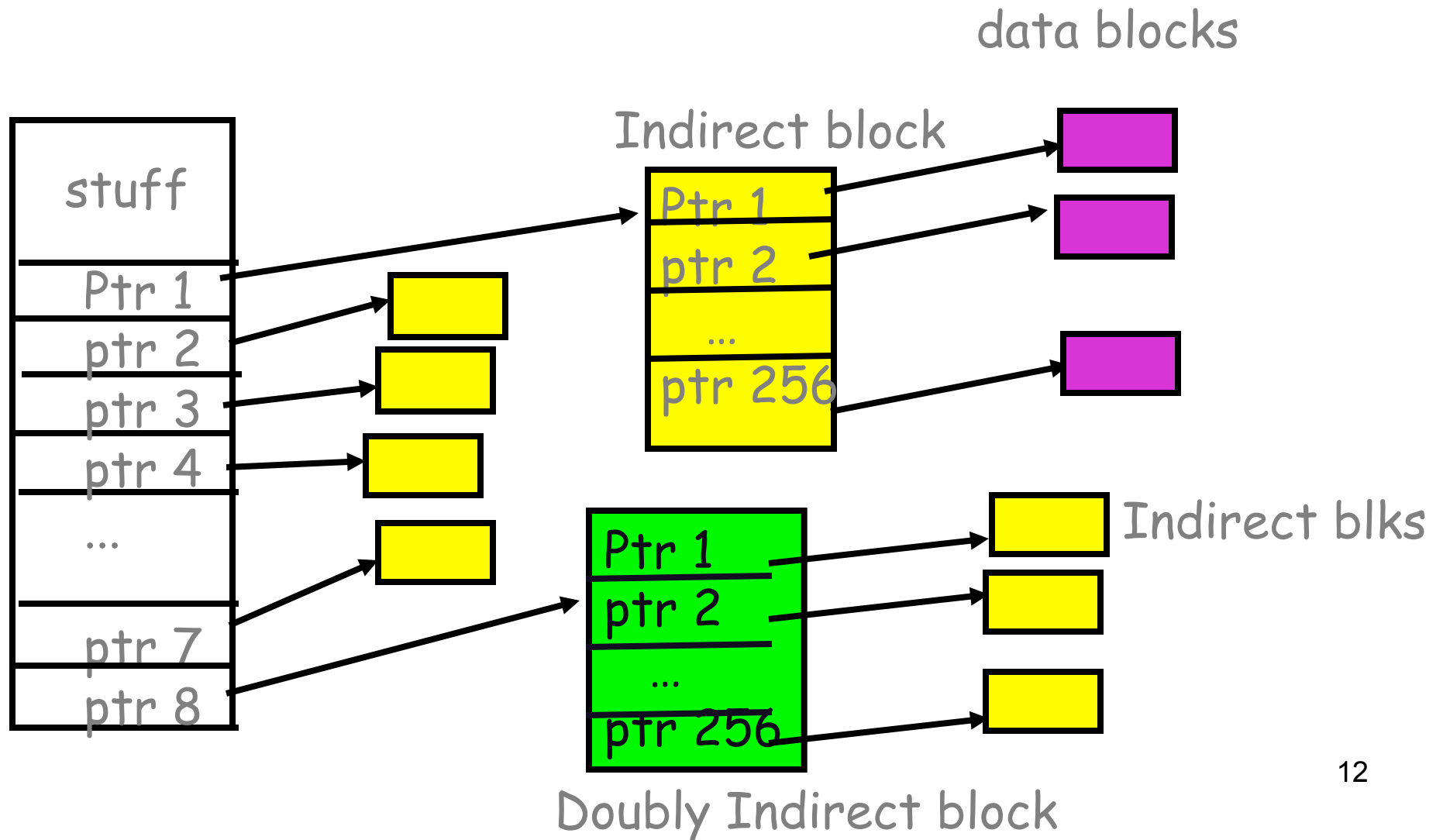
```
struct inode {
    uint16_t    i_mode;
    uint8_t     i_nlink;           /* directory entries */
    uint8_t     i_uid;            /* owner */
    uint8_t     i_gid;           /* group of owner */
    uint8_t     i_size0;         /* most significant of size */
    uint16_t    i_size1;         /* least sig */
    uint16_t    i_addr[8];       /* device addresses constituting file */
    uint16_t    i_atime[2];      /* access time */
    uint16_t    i_mtime[2];      /* modify time */
};
#define IALLOC    0100000        /* file is used */
#define IFMT      060000        /* type of file */
#define IFDIR     040000        /* directory */
#define IFCHR     020000        /* character special */
#define IFBLK     060000        /* block special, 0 is regular */
#define ILARG     010000        /* large addressing algorithm */
```

Inode – Small file (≤ 8 blocks)



“ptr” are not memory pointers, but rather names in the disk name space (i.e. sector numbers)

Inode – large file (> 8 blocks)



Directories

- Layered on top of file abstraction
- Same as files but containing an array of 16 byte directory entries:

File name	Inode Number
program	10
paper	12

- Directory entry:
 - 2-byte i-number
 - 14-byte zero-terminate string name

```
struct direntv6 {  
    uint16_t d_inumber;  
    char    d_name[14];  
};
```

Assignment 1: Read Files from a Version 6 Unix Disk

File System Implementation

Layer	Implements	Name Space
Pathname layer	Hierarchical namespace	Pathname
Directory layer	Directories	Filename
File layer	Files	Inumber
Disk Layer	Disk access	Sector number

Layering is important. Respect the layers (unless you have a good reason not to)

Inode

```
int inode_iget(  
    struct unixfilesystem *fs,           ← environment  
    int inumber,                         ← input  
    struct inode *inp);                 ← output
```

C idiom: output parameters are often pointers to structures.

(Benefit: caller can choose to allocate it on the stack or on the heap)

C boot camp: Bitwise Operations

```
struct inode {
    /* Hidden here */
    uint8_t    i_size0;      /* most significant of size */
    uint16_t   i_size1;     /* least sig */
    /* Hidden here */
};
int
inode_getsize(struct inode *inp) {
    return ((inp->i_size0 << 16) |inp->i_size1);
}
```

- `inode_getsize` should make sense now
 - Shift the 8 most significant bits over by 16
 - Bitwise OR in the 16 low bits to find the size

Inode : i_mode

- Single bit fields:
 - Unallocated: $(i_mode \& IALLOC) == 0$
 - Large file: $(i_mode \& ILARG) != 0$
 - Small file: $(i_mode \& ILARG) == 0$
- Two-bit fields
 - $(i_mode \& IFMT) == IFDIR$

File layer

- `int` ← bytesRead
`file_getblock(
struct unixfilesystem *fs, ← environment
int inumber, ← input
int blockNo, ← input
void *buf); ← output`

Valid assertion:

(bytesRead = -1 || bytesRead <= DISK_SECTOR_SIZE)

Verify assumptions on disk layout

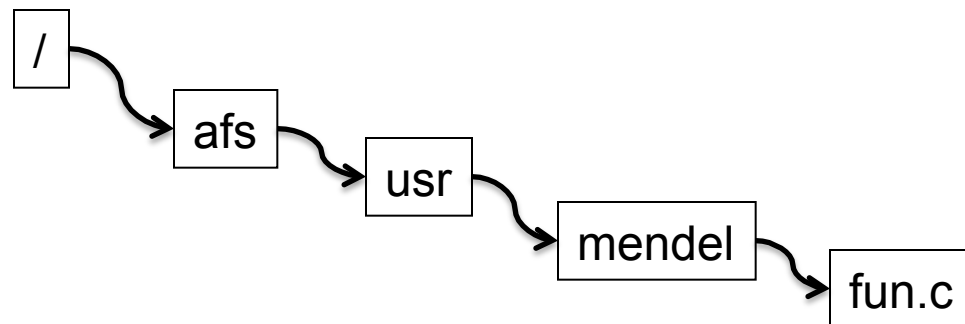
- Otherwise: risk of GIGO (garbage-in, garbage-out)
- Example: directory layer

```
struct direntv6 {  
    uint16_t d_inumber;  
    char    d_name[14];  
};
```

- Can verify that
 - d_inumber is a valid inumber
 - d_name is a null-terminated string

Pathname layer

- `/afs/usr/mendel/fun.c`
- Semantically equivalent to:



- In your code, don't create a link list

Questions?
Thank you!