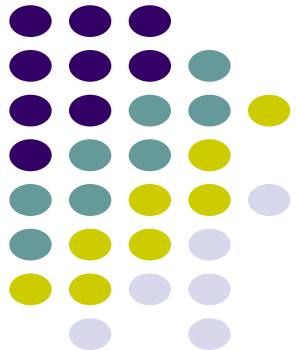
File System Abstraction

ECE 469, April 10th

Aravind Machiry

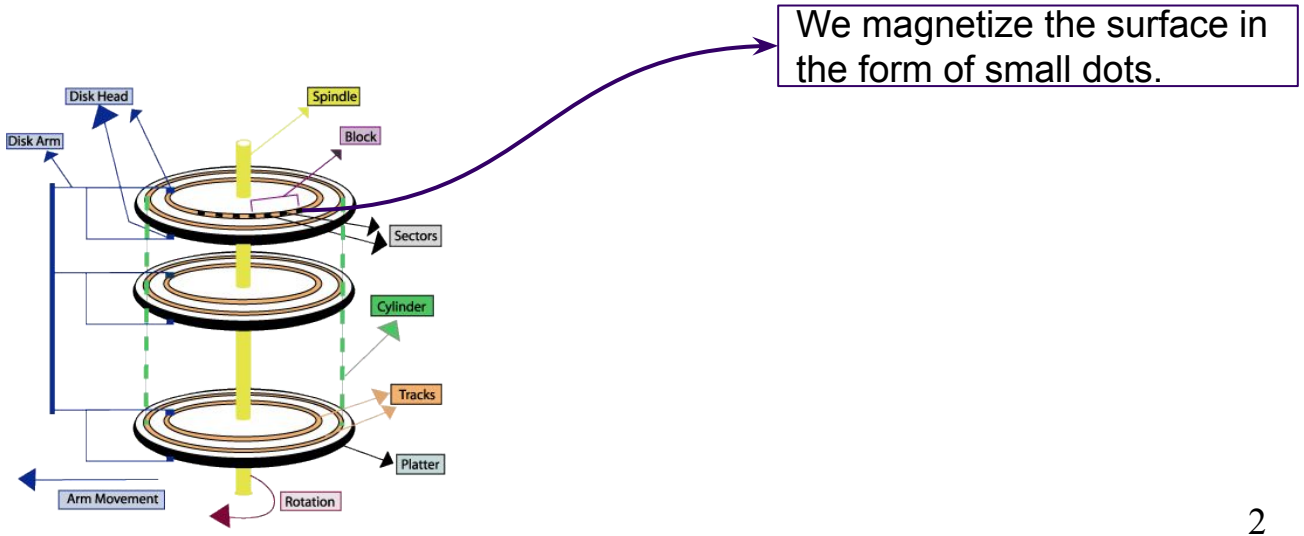


Storage Technologies

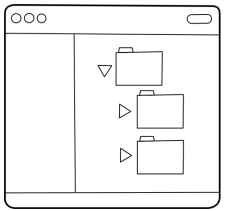
- Tapes
- Magnetic Disks
- Flash Memory

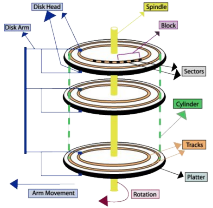
How to store data on Disks?

- Writing magnetized dots onto disk platter:
  - Each dot: 0/1
  - Unit of access: Block (512 or 4K)

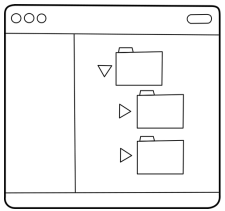


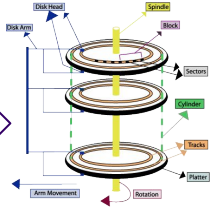
How do we access storage?





File System Abstraction





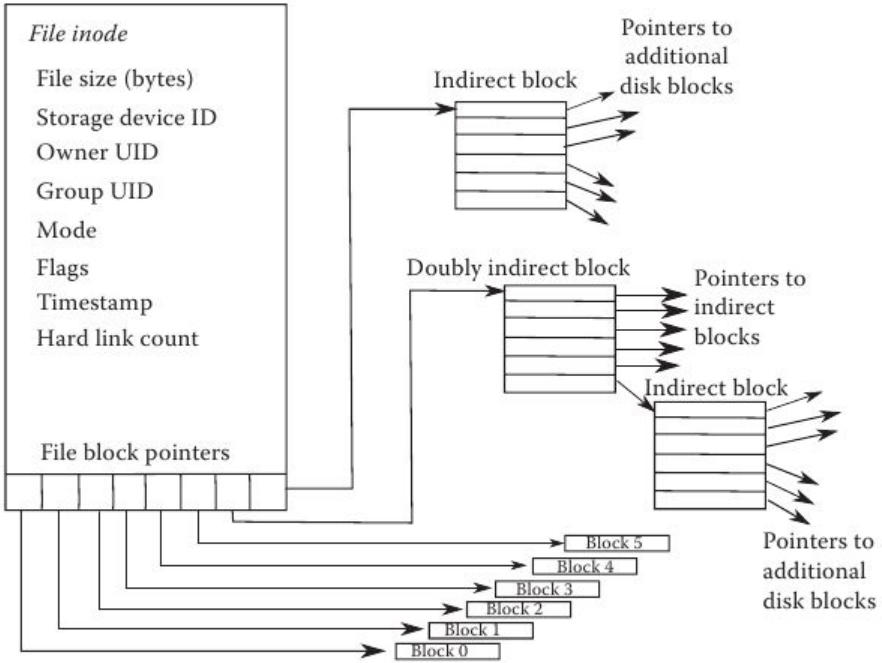
What does a File System Store about a file?

- File Metadata:
  - File Name.
  - Permissions (read or write).
  - owner/group.
  - Type of file.
  - List of disk blocks that contain data of the file.
  - Several others...

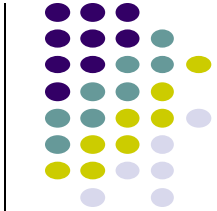
inode: Structure of metadata

- Indirect Node.
- Each file has one or more inodes corresponding to it.
- Where are inodes stored?
  - Disk
- A portion of disk is dedicated to inodes.

File inode


File inode
File size (bytes)
Storage device ID
Owner UID
Group UID
Mode
Flags
Timestamp
Hard link count
File block pointers
Indirect block
Pointers to additional disk blocks
Doubly indirect block
Pointers to indirect blocks
Indirect block
Pointers to additional disk blocks
Block 5
Block 4
Block 3
Block 2
Block 1
Block 0

File Permissions



- Type of accesses:
  - Read (R)
  - Write (W)
  - Execute (X)

- Who can access?
  - Owner : A user who owns the file.
  - Group : A group to which the file belongs.
  - Others: All other users.

Permission Bits

- How many bits are needed to stored file permissions?
  - 3 Types of accesses X 3 Entities = 9.
  - `ls -l`.
    - rw-rw-r-- …
    - Bit Pattern (664): 110110100
    - User, Group, Others
  - Change Permissions:
    - e.g., chmod 775 <filePath>

Accessing File System From User Space

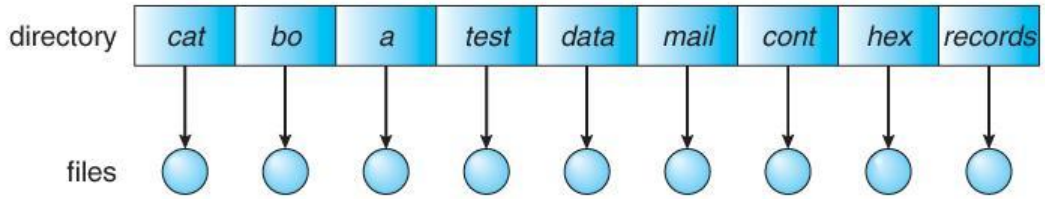
- System calls:
  - create/open/read/write/stat/etc.

Directories

- Files namespace:
  - Helps in organizing files.

- How are they stored?
  - Same as files, where the data block contains: **List of (filename, inode) pairs**.

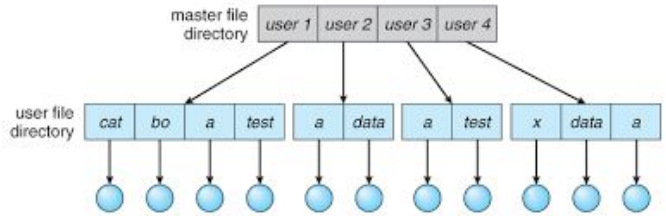
Single Level Directories



- Single level grouping.
- All users see the same directories.

- Easy to search.
- No grouping.
- Inconvenient naming.

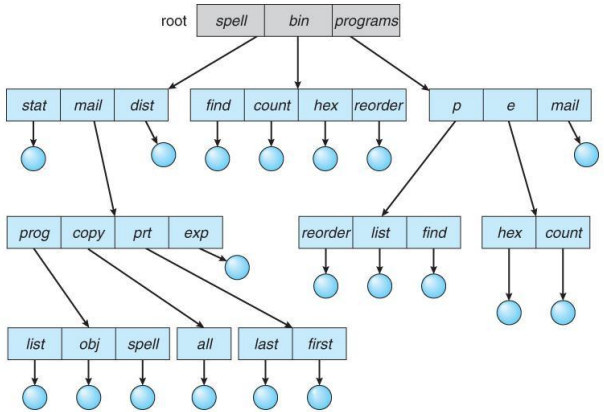
Two Level Directories



- Each user can create grouping.
- Path names.

- Efficient search.
- Not enough grouping.

Tree Structured Directories



- Directories can now contain files or subdirectories.
- Efficient search and arbitrary grouping!

Directory Permissions

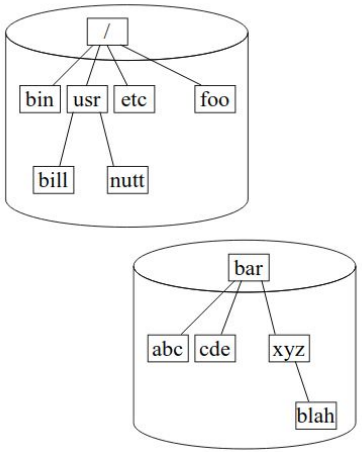
- Reading a directory:
  - Being able to list all files.
- Writing a directory:
  - Write contents into directory: Create a delete inside the directory.
- Executing a directory:
  - Able to access individual files in a directory.
- Difference between executing and read?

Directory Permissions

dir permissions	Octal	del rename create files	dir list	read file contents	write file contents	cd dir	cd subdir	subdir list	access subdir files
---	0								
-W-	2								
R--	4		only file names (*)						
RW-	6		only file names (*)						
--X	1			X	X	X	X	X	X
-WX	3	X		X	X	X	X	X	X
R-X	5		X	X	X	X	X	X	X
RWX	7	X	X	X	X	X	X	X	X

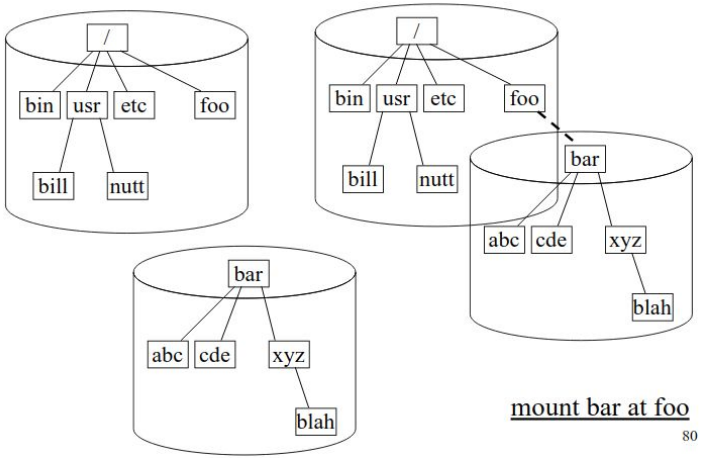
Mounting File System

- Mapping from a name in a filesystem to root of another filesystem
- Allows users to manage multiple filesystems through a single filesystem.



Mounting File System

- Mapping from a name in a filesystem to root of another filesystem
- Allows users to manage multiple filesystems through a single filesystem.



Links: File Aliases

- Multiple files representing the same data.

- How to create file aliases?
  - Files having the same inode number.
  - A file whose data is the name of the original file.

Hard Link

- Files which point to the **same inode**.

- Hard links should belong to the same filesystem:
  - Inode numbers are specific to a file system.

File Name	Type	Inode	Content
foo	Regular	4357891	….
….	….	….	….
baz	Regular	4357891	….

Hard Link : File Deletion

- What happens when a file gets deleted?
- What happens to the inode?
  - Reference counting

Soft Link

- A file whose **data is the path of the original file**.

- Soft links can be created across file systems:
  - Referenced by file path.

File Name	Type	Inode	Content
foo	Regular	4357891	….
….	….	….	….
bar	Soft Link	4768929	**“...foo”**

Soft Link : File Deletion

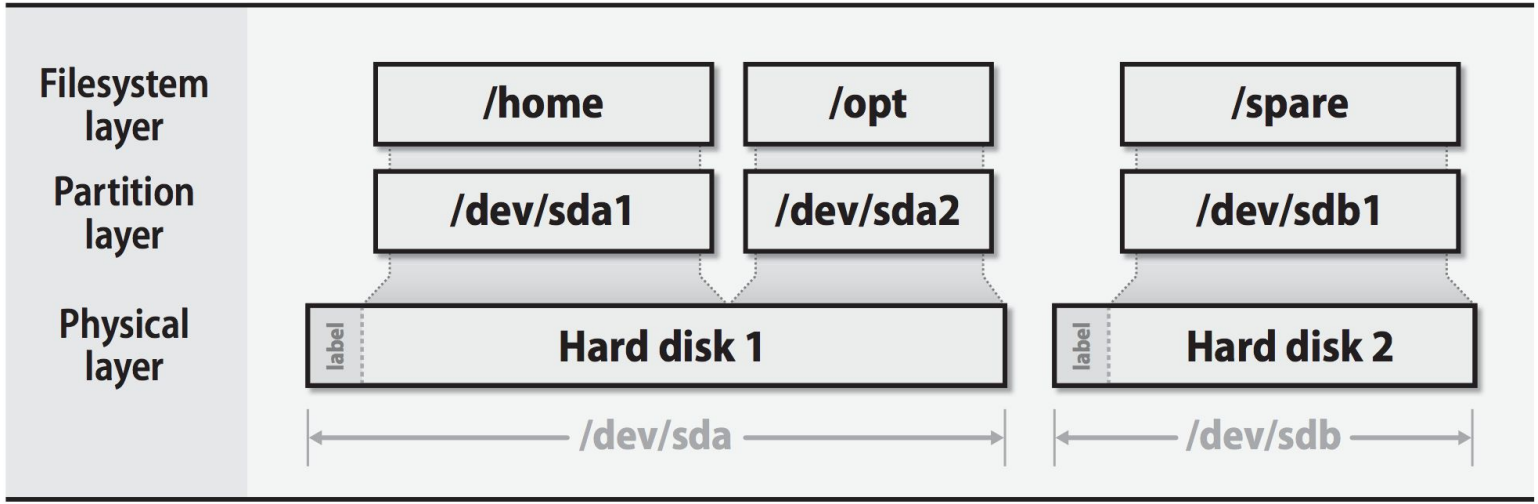
- What happens when a file gets deleted?
  - target file v/s soft link file

Other Permissions

- setuid bit:
  - For files with execute permissions.
  - The file will always be executed by using owner user id.
  - e.g., Network daemon.

- sticky bit
  - Users can read/write but only owners can delete.
  - e.g., “/tmp” folder.

Big Picture


Filesystem layer
Partition layer
Physical layer
/home
/opt
/spare
/dev/sda1
/dev/sda2
/dev/sdb1
label
Hard disk 1
label
Hard disk 2
/dev/sda
/dev/sdb

Looking into Simple File Systems

- File System Design Considerations.
- Simple File systems: FAT/EXT2.

File Access

File name ---------->

offset

```c
int main(int argc, char *argv[])
{
  int fd;
  char buffer[4096];
  struct stat_buf;
  DIR *dir;
  struct dirent *entry;

  /* 1. Path name -> inode mapping */
  fd = open("/home/machiry/hello.c" , O_RDONLY);

  /* 2. File offset -> disk block address mapping */
  pread(fd, buffer, sizeof(buffer), 0);

  /* 3. File meta data operation */
  fstat(fd, &stat_buf);
  printf("file size = %d\n", stat_buf.st_size);

  /* 4. Directory operation */
  dir = opendir("/home");
  entry = readdir(dir);
  printf("dir = %s\n", entry->d_name);
  return 0;
}
```

File Access

File name ---------- *Directory* ----------> File Number

offset offset

```c
int main(int argc, char *argv[])
{
  int fd;
  char buffer[4096];
  struct stat_buf;
  DIR *dir;
  struct dirent *entry;

  /* 1. Path name -> inode mapping */
  fd = open("/home/machiry/hello.c" , O_RDONLY);

  /* 2. File offset -> disk block address mapping */
  pread(fd, buffer, sizeof(buffer), 0);

  /* 3. File meta data operation */
  fstat(fd, &stat_buf);
  printf("file size = %d\n", stat_buf.st_size);

  /* 4. Directory operation */
  dir = opendir("/home");
  entry = readdir(dir);
  printf("dir = %s\n", entry->d_name);
  return 0;
}
```

File Access

File name offset ---Directory---> File Number offset ---inode---> Storage block

```c
int main(int argc, char *argv[])
{
  int fd;
  char buffer[4096];
  struct stat_buf;
  DIR *dir;
  struct dirent *entry;

  /* 1. Path name -> inode mapping */
  fd = open("/home/machiry/hello.c" , O_RDONLY);

  /* 2. File offset -> disk block address mapping */
  pread(fd, buffer, sizeof(buffer), 0);

  /* 3. File meta data operation */
  fstat(fd, &stat_buf);
  printf("file size = %d\n", stat_buf.st_size);

  /* 4. Directory operation */
  dir = opendir("/home");
  entry = readdir(dir);
  printf("dir = %s\n", entry->d_name);
  return 0;
}
```

Key Data Structures of a FS

- inodes: File metadata and storage blocks
- Directories: Special files that contain:
  - File name -> inode
- Freemap: which disk blocks are free/allocated?

File System Workload

- File Sizes:
  - Are most files small/large?
    - Small
  - which accounts for total storage: small/large?
    - Large

File System Workload

- File Sizes:
  - Are most files small/large?
    - Small
  - which accounts for total storage: small/large?
    - Large

- File Access:
  - Are most file accesses to small/large files?
    - Small
  - which accounts for more total I/O bytes: small/large?
    - Large

File System Design

- For Small files:
  - Small blocks for storage efficiency.
  - File used together should be stored together.

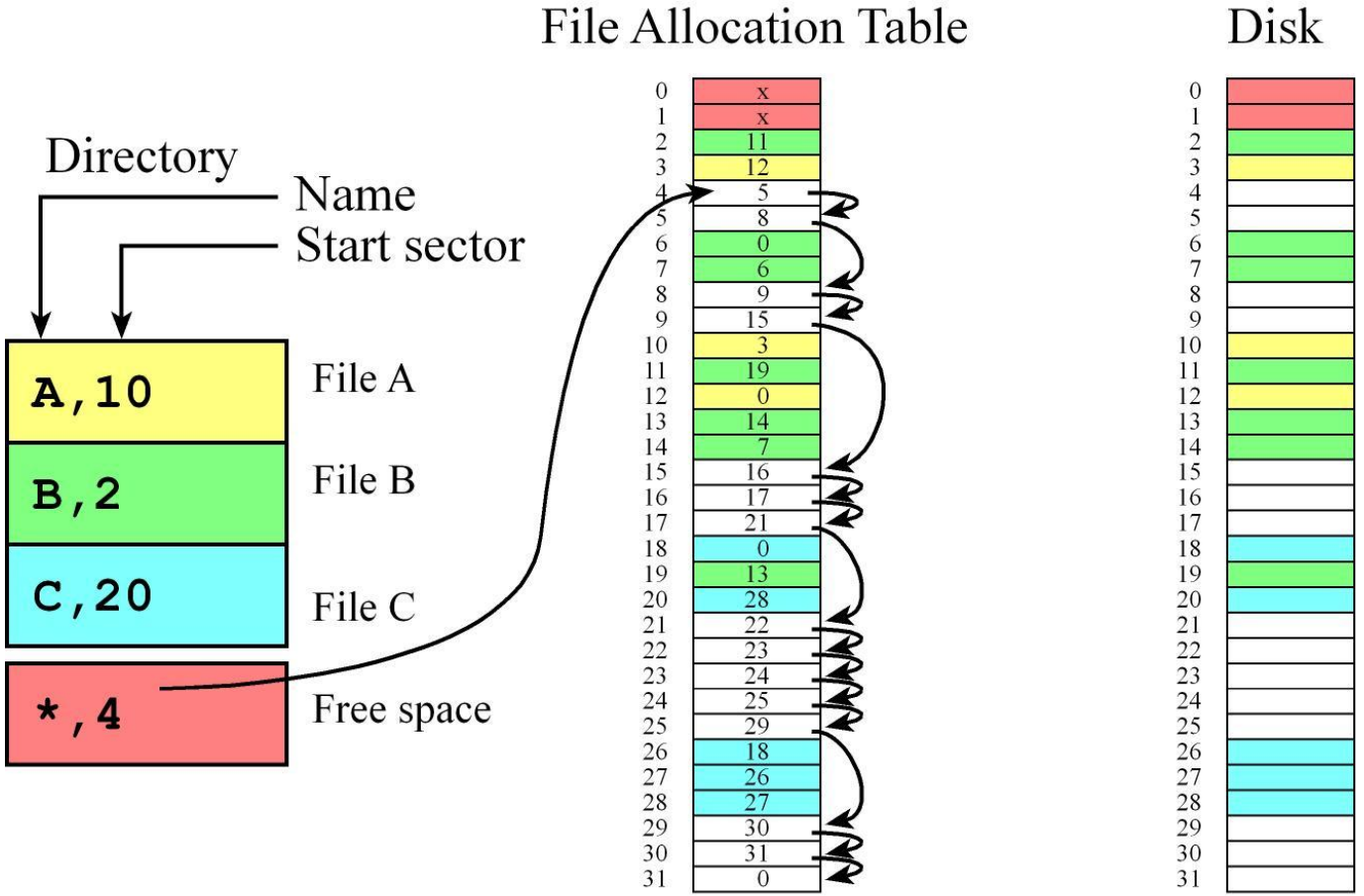
- For Large files:
  - Large blocks for storage efficiency.
  - Contiguous allocation for fast sequential lookup.
  - Efficient lookup for random access.

- Difficult to predict what a file will turn out to be.

File Allocation Table (FAT)

- Simple.
- Easy to implement.
- Still used in Phones and Thumb drives.

- Key data structure: File Allocation Table
  - List of all disk blocks.
  - File: Linked list of blocks.

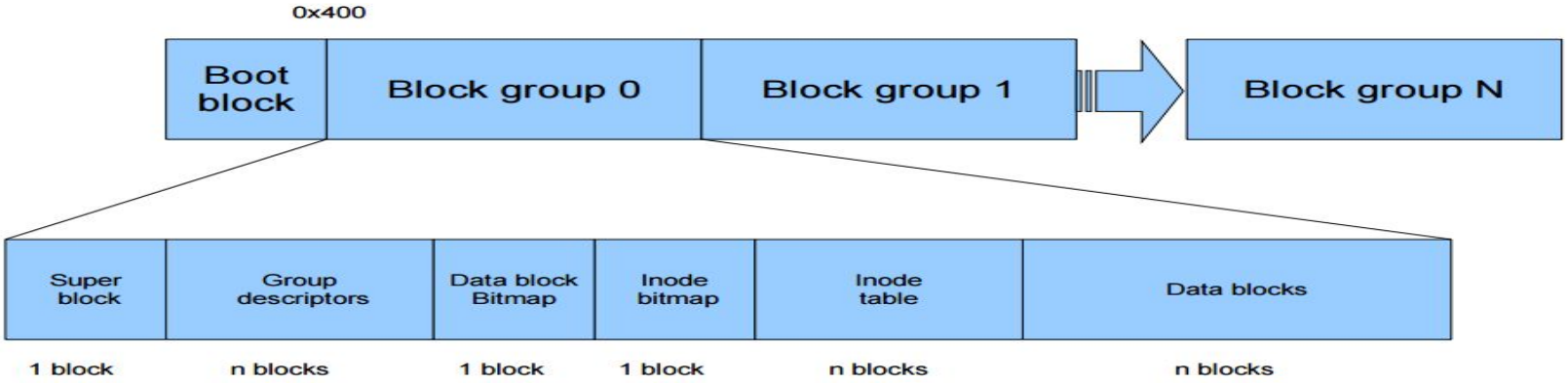

File Allocation Table
Disk
Directory
Name
Start sector
A,10
B,2
C,20
*,4
File A
File B
File C
Free space
0 x
1 x
2 11
3 12
4 5
5 8
6 0
7 6
8 9
9 15
10 3
11 19
12 0
13 14
14 7
15 16
16 17
17 21
18 0
19 13
20 28
21 22
22 23
23 24
24 25
25 29
26 18
27 26
28 27
29 30
30 31
31 0

FAT

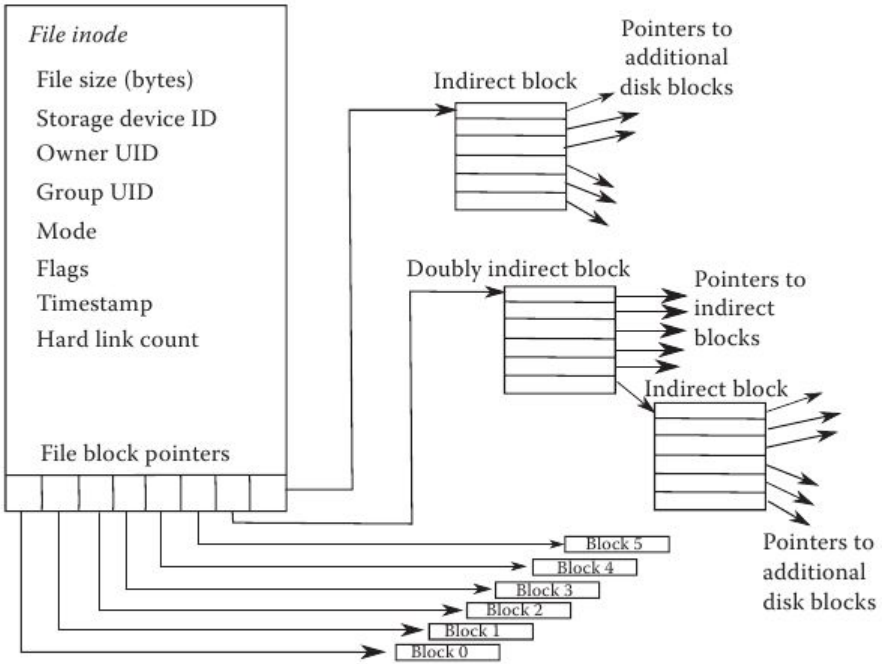
- Pros: Simple
  - Easy to find free block.
  - Easy to append file.
  - Easy to delete a file.

- Cons:
  - Small file access is slow.
  - Random file access is very slow.
  - Fragmentation:
    - Blocks of a small file could be heavily scattered.
    - Problem becomes worse as the usage increases.

EXT2 File System



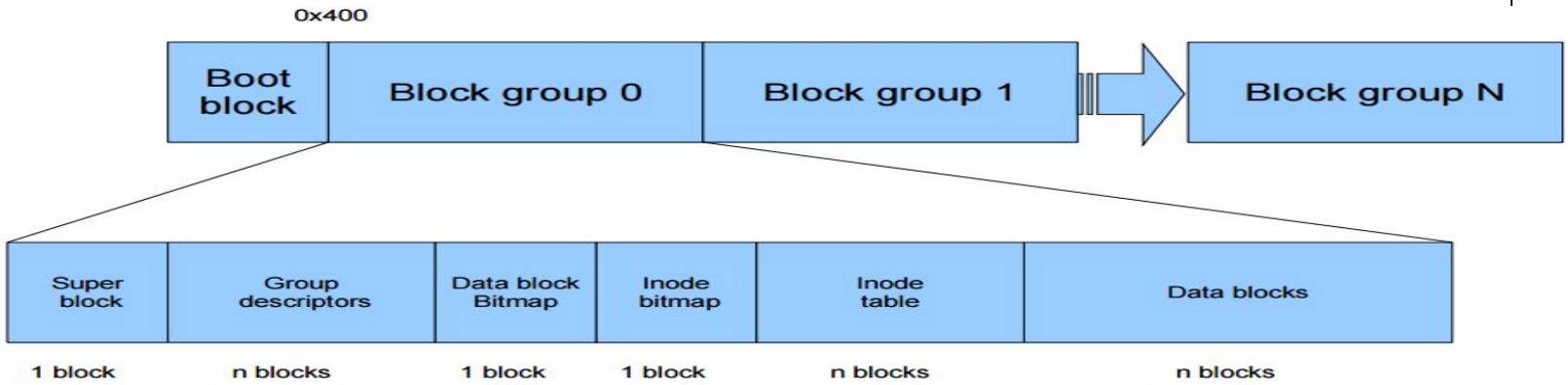
EXT2 File System : inode



EXT2 File Size (Block Size: 4K)

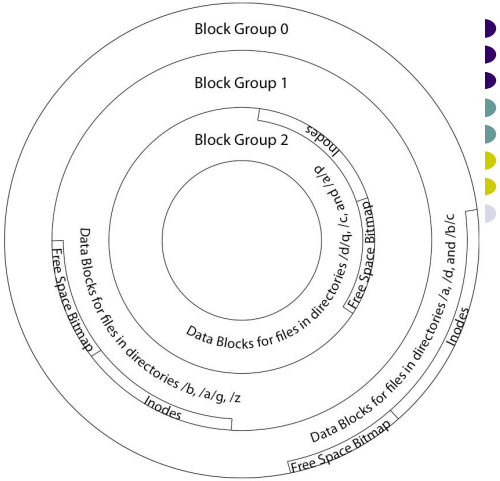
- 12 File Block Pointers = 12*4 = 48K
- 1 Indirect block pointer (4K) = 1K direct block pointers = 1K*4K = 4MB
- 1 doubly indirect block pointer (4K) = 1K Indirect block pointers = 1K*4MB = 4 GB
- 1 triply indirect block pointer (4K) = 1K doubly Indirect block pointers = 1K*4GB = 4 TB
- Total Size = 48K + 4MB + 4GB + 4TB

EXT2 Locality



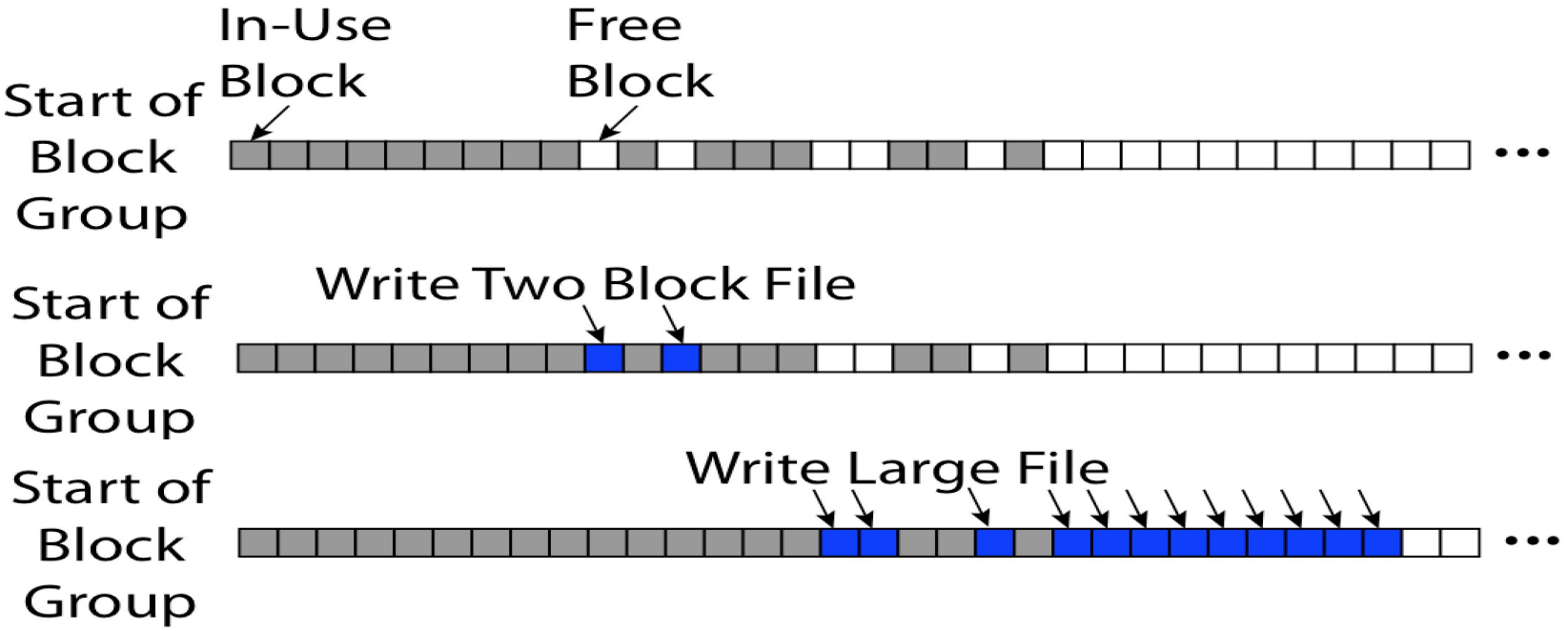
- Disk divided into block groups.
- Inode table spread throughout the disk.

EXT2 Locality



- All files with in a directory in same block group.
- Sub-directories in different block groups.
- First-fit algorithm for assigning data-blocks:
  - Fragmented small files but contiguous large files.

EXT2 First Fit



EXT2

- Pros: Simple
  - Efficient storage for both small and large files.
  - Locality for data and inodes.

- Cons:
  - Inefficient for tiny files.
  - Inefficient encoding for very large files.
  - Needs 10-20% space to avoid fragmentation.

File Traversal Review

- Let's say you want to open "/one/two/three"
  fd = open("/one/two/three", O_RDWR);
- What goes on inside the file system?
  - open directory "/" (well known, can always find)
  - search the directory for "one", get location of "one"
  - open directory "one", search for "two", get location of "two"
  - open directory "two", search for "three", get loc. of "three"
  - open file "three"
  - (of course, permissions are checked at each step)
- Another example: mkdir /a/b/c
  - Read inode 2 (root), look for "a": find <"a", 5>
  - Read inode 5, look for "b": find <"b", 9>
  - Read inode 9, verify no "c" exists; allocate c and add "c" to directory
- FS spends lots of time walking down directory paths
  - this is why open is separate from read/write
  - OS will cache prefix lookups to enhance performance
    - /a/b, /a/bb, /a/bbb all share the "/a" prefix

Exploring EXT2

gzip -d disk.img.gz

$ ls -lh disk.img
-rw-rw-r-- 1 machiry machiry 2.0M Apr 6 2020 disk.img

mkdir /tmp/myfs

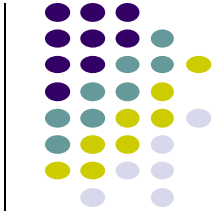
# Mount the file system
$ sudo mount disk.img /tmp/myfs

$ mount | grep myfs
/home/machiry/Desktop/lec22/disk.img on /tmp/myfs type ext2 (rw,relatime)

Directory Structure

```
$ ls -lh /tmp/myfs/
total 14K
-rw-r--r-- 1 root root   10 Apr  6  2020 aa
lrwxrwxrwx 1 root root    2 Apr  6  2020 aa-symlink -> aa
drwx------ 2 root root  12K Apr  2  2020 lost+found
drwxr-xr-x 2 root root 1.0K Apr  6  2020 testdir
```

Checking Super Block



```
$ df -h | grep myfs
/dev/loop6                      2.0M   23K  1.9M   2% /tmp/myfs

$ sudo dumpe2fs /dev/loop6
dumpe2fs 1.45.5 (07-Jan-2020)
Filesystem volume name:   <none>
Last mounted on:          /tmp/myfs
Filesystem UUID:          6430eccd-11d0-4ea9-bd26-ce6e946dc02b
Filesystem magic number:  0xEF53
…
Inode count:              256
Block count:              2048
Reserved block count:     102
Free blocks:              1988
Free inodes:              242
First block:              1
Block size:               1024
Fragment size:            1024
Reserved GDT blocks:      7
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         256
Inode blocks per group:   32

Group 0: (Blocks 1-2047)
  Primary superblock at 1, Group descriptors at 2-2
  Reserved GDT blocks at 3-9
  Block bitmap at 10 (+9)
  Inode bitmap at 11 (+10)
  Inode table at 12-43 (+11)
  1988 free blocks, 242 free inodes, 3 directories
  Free blocks: 59-512, 514-2047
  Free inodes: 15-256
```

Only one block group

Exploring inode map and block map

```
# Inode bitmap, starts at block 11, few occupied (as indicated by 1s)
$ sudo dd if=/dev/loop6 bs=1024 skip=11 count=1 status=none | xxd -b -l 32
00000000: 11111111 00111111 00000000 00000000 00000000 00000000  .?....
00000006: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000000c: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 00000000 00000000 00000000 00000000 00000000 00000000  ......

0000001e: 00000000 00000000
```

```
# Block map: starting at block 10.
$ sudo dd if=/dev/loop6 bs=1024 skip=10 count=1 status=none | xxd -b -l 256
00000000: 11111111 11111111 11111111 11111111 11111111 11111111  ......
00000006: 11111111 00000011 00000000 00000000 00000000 00000000  ......
0000000c: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000024: 00000000 00000000 00000000 00000000 00000000 00000000  ......
```

Directory Structure

machiry@/tmp/myfs$ ls -li

total 14

12 -rw-r--r-- 1 root root 10 Apr 6 2020 aa

14 lrwxrwxrwx 1 root root 2 Apr 6 2020 aa-symlink -> aa

11 drwx------ 2 root root 12288 Apr 2 2020 lost+found

13 drwxr-xr-x 2 root root 1024 Apr 6 2020 testdir

$ cat aa

mycontent

Dumping Disk Block

$ sudo debugfs -R "stat <12>" /dev/loop6

Inode: 12 Type: regular Mode: 0644 Flags: 0x0

Generation: 4138714773 Version: 0x00000001

User: 0 Group: 0 Size: 10

File ACL: 0

Links: 1 Blockcount: 2

Fragment: Address: 0 Number: 0 Size: 0

ctime: 0x5e8b4e5f -- Mon Apr 6 11:44:31 2020

atime: 0x606c6806 -- Tue Apr 6 09:54:14 2021

mtime: 0x5e8b4e5f -- Mon Apr 6 11:44:31 2020

BLOCKS:

(0):513

TOTAL: 1

Dumping Disk Block

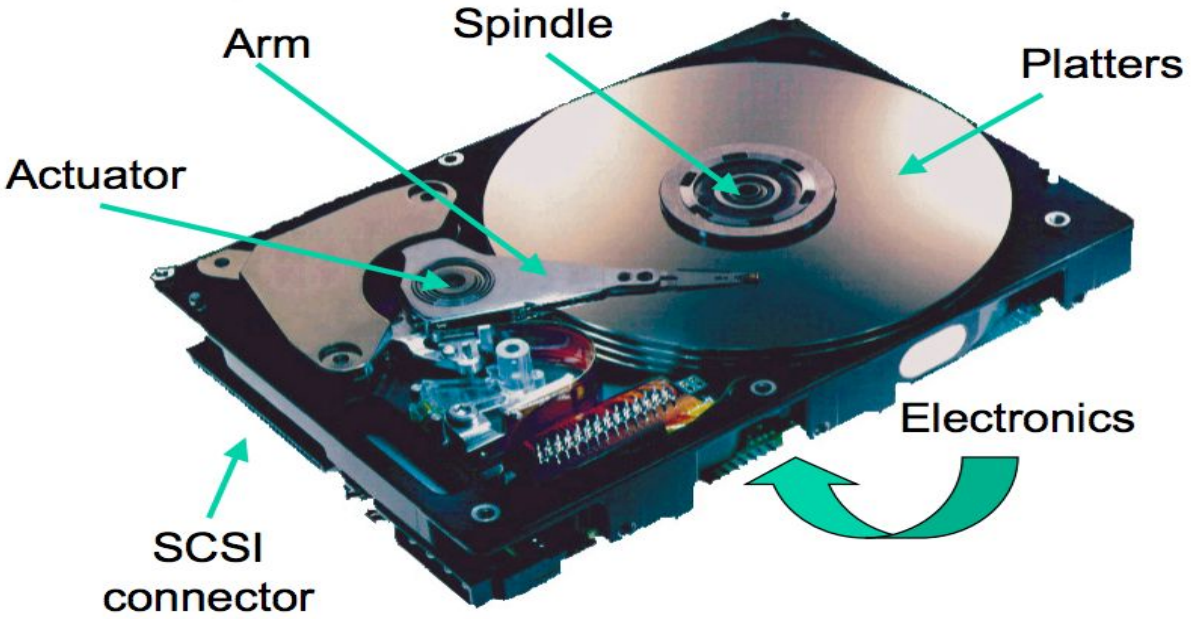
$ sudo dd if=/dev/loop6 bs=1024 **skip=513** count=1 status=none | hexdump -C

```
00000000  6d 79 63 6f 6e 74 65 6e  74 0a 00 00 00 00 00 00  |mycontent.......|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

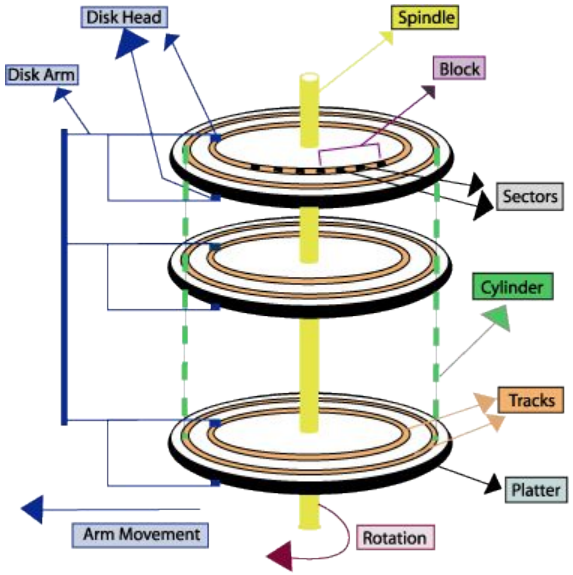
BACKUP SLIDES

BACKUP STARTS

What is inside a disk drive?



More Deeper!


Disk Head
Spindle
Disk Arm
Block
Sectors
Cylinder
Tracks
Platter
Arm Movement
Rotation