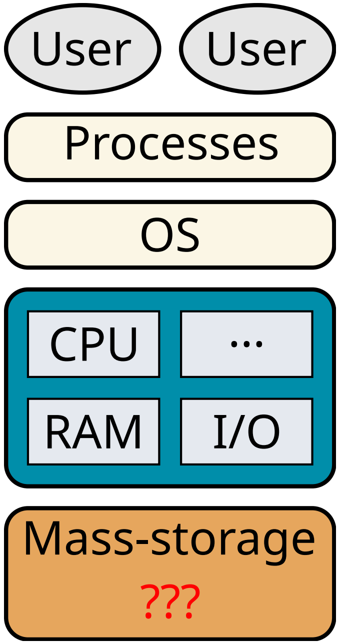
Filesystem Abstraction

Introduction

Long-term data storage requirements

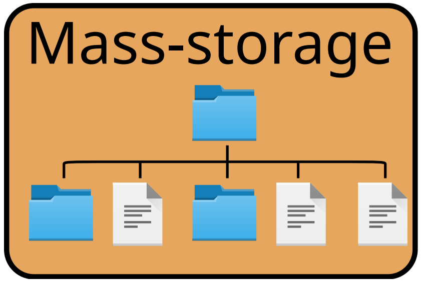


- (Very) large amounts of non-volatile data
- Easy way to find data
- Concurrent access from processes
- Controlled sharing between users
- Performance
- Reliability
  - Survive power-off, OS crash, process termination

Filesystems (FS)

Abstractions for maximizing the usage of non-volatile storage

- Persistent, named data (files)
- Hierarchical organization (directories, subdirectories)
- Performance
- Access control
- Crash and storage error tolerance



Filesystem concepts

File

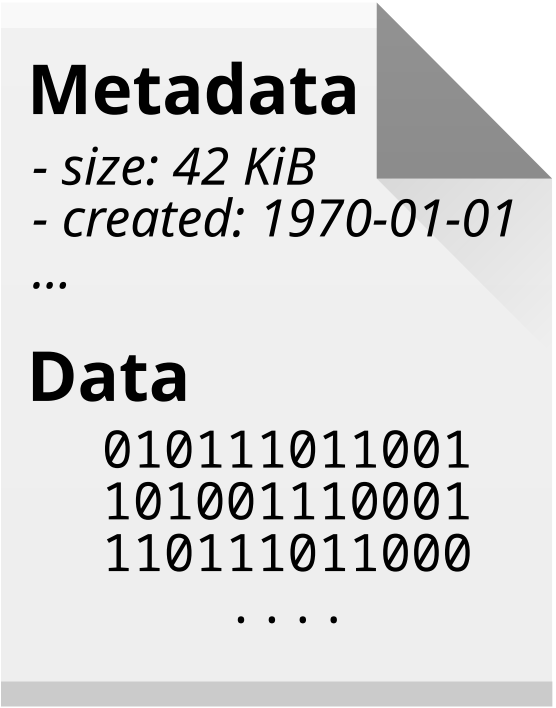
- Abstraction that provides persistent, described data
- Logical storage unit

Metadata

- File attributes added and managed by the OS
- Size, creation/modification time, owner, permissions, etc.
- File's content location (index structure)

Data

- What a user puts in the file
- Array of untyped bytes



Metadata

- size: 42 KiB
- created: 1970-01-01
...

Data

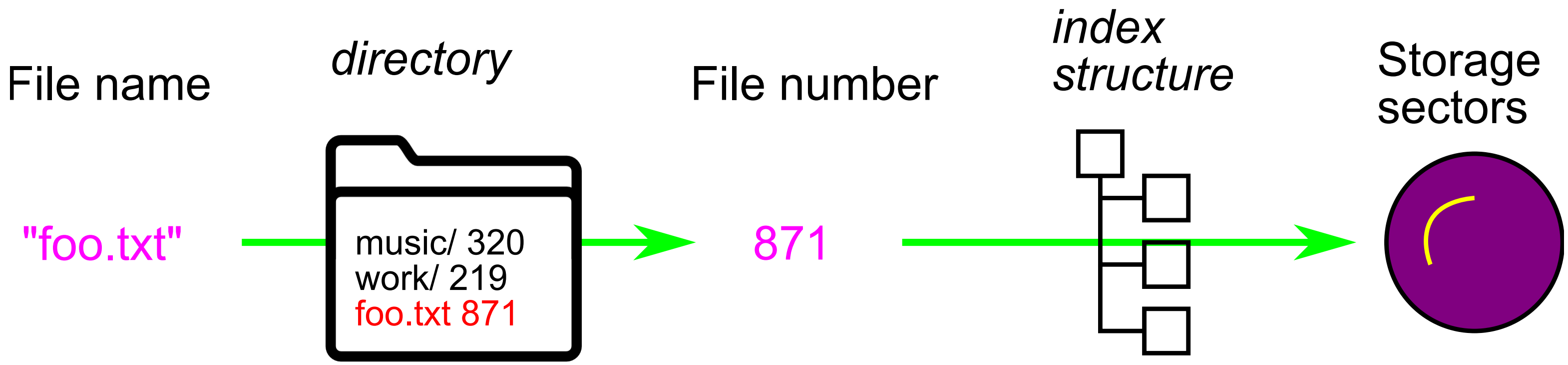
```
010111011001
101001110001
110111011000
    ....
```

Filesystem concepts

Directory

Directories provide names for files

- Groups of named files or subdirectories
- Mapping from human-readable file names to file metadata locations



Filesystem concepts

Naming conventions

- Depends on filesystem

Case

- Windows traditionally case-insensitive
- UNIX traditionally case-sensitive

Length

- Before, could be quite limited (11 with MS-FAT)
- Today, usually up to 255 characters

Extension

- Before, separated from filename, and meaningful
- Today, usually part of the filename, and just a hint

```
$ file innocent_text_file.txt
innocent_text_file: ELF 64-bit LSB executable, x86-64,
version 1 (SYSV), dynamically linked

$ file scary_executable_file.x
scary_executable_file.x: ASCII text
```

Filesystem concepts

Digression about executables

- How does the OS recognize executable files?

Binary executables

- *Magic number* at very beginning of file, to identify its format
- Example with an `ELF` executable

```
$ xxd /usr/bin/firefox | head -1
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
```

Scripts

- *Shebang* as first line of script (`#!`)
  - Tells OS which *interpreter* should be used to run the script
  - Line ignored by interpreter because it's a comment
- Example with a shell script

```
$ cat test_script.sh
#!/bin/sh

echo Hello
```

```
$ ./test_script.sh
Hello
$ /bin/sh test_script.sh
Hello
```

Filesystem concepts

Path

Absolute

- Path of file from the root directory
- e.g., `/home/jporquet/project1/solution`

Relative

- Path from the current working directory (`CWD`)
- `CWD` stored in process' PCB

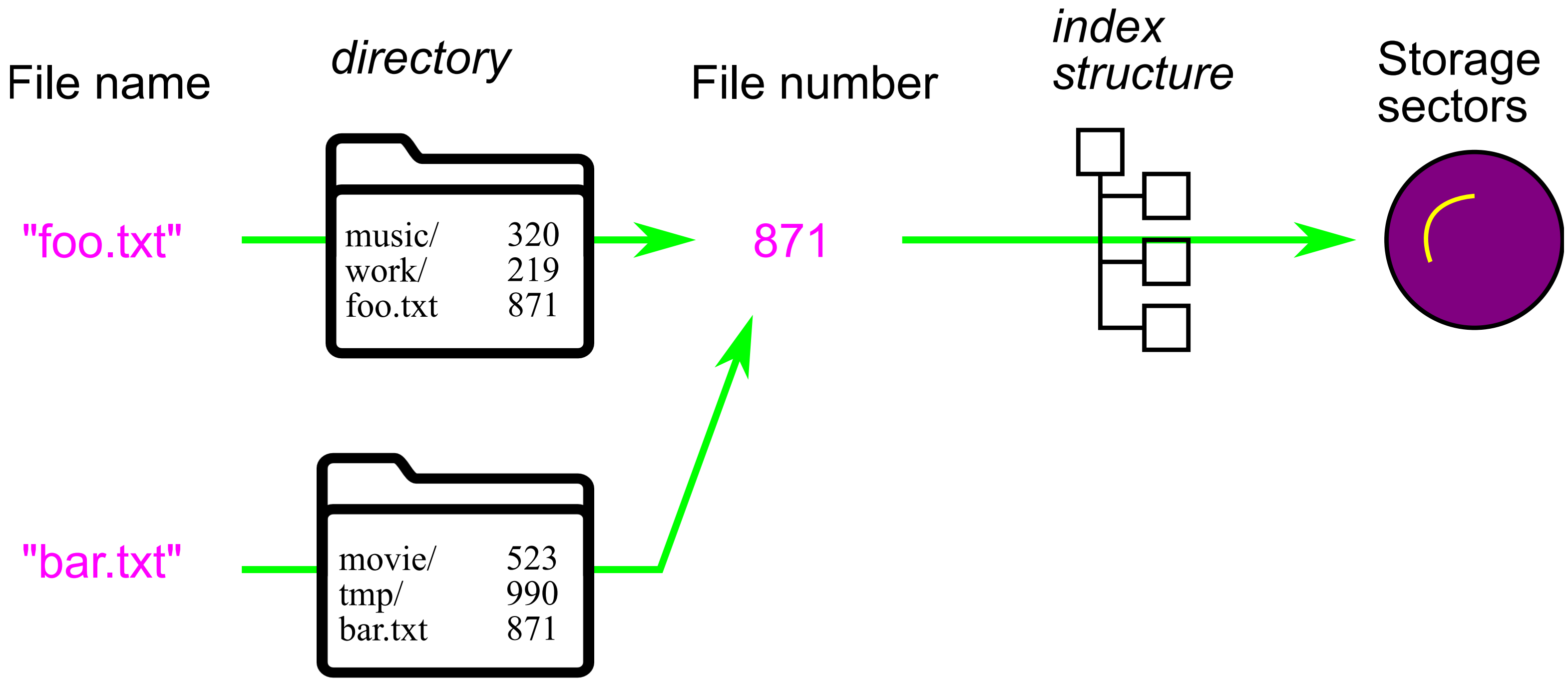
Special entries

- 2 special entries in each UNIX directory
  - `.`: current directory
  - `..`: parent directory

Filesystem concepts

Hard link

Link from name to metadata location



music/	320
work/	219
foo.txt	871

movie/	523
tmp/	990
bar.txt	871

Filesystem concepts

Hard links and cycles

- No difference between an "original" file name and a hard-link to it

```
$ touch test
$ ls -li test
21892004 -rw-r--r-- 1 joel joel 0 2017-03-02 17:15 test

$ ln test test2
$ ls -li test*
21892004 -rw-r--r-- 2 joel joel    0 2017-03-02 17:15 test
21892004 -rw-r--r-- 2 joel joel    0 2017-03-02 17:15 test2
```

- Creation of a directory loop would make any file tree walker error-prone (e.g. `du` or `fsck`)
    - Unless keeping track of all the inodes that are being traversed
- Hard-links to directory are forbidden

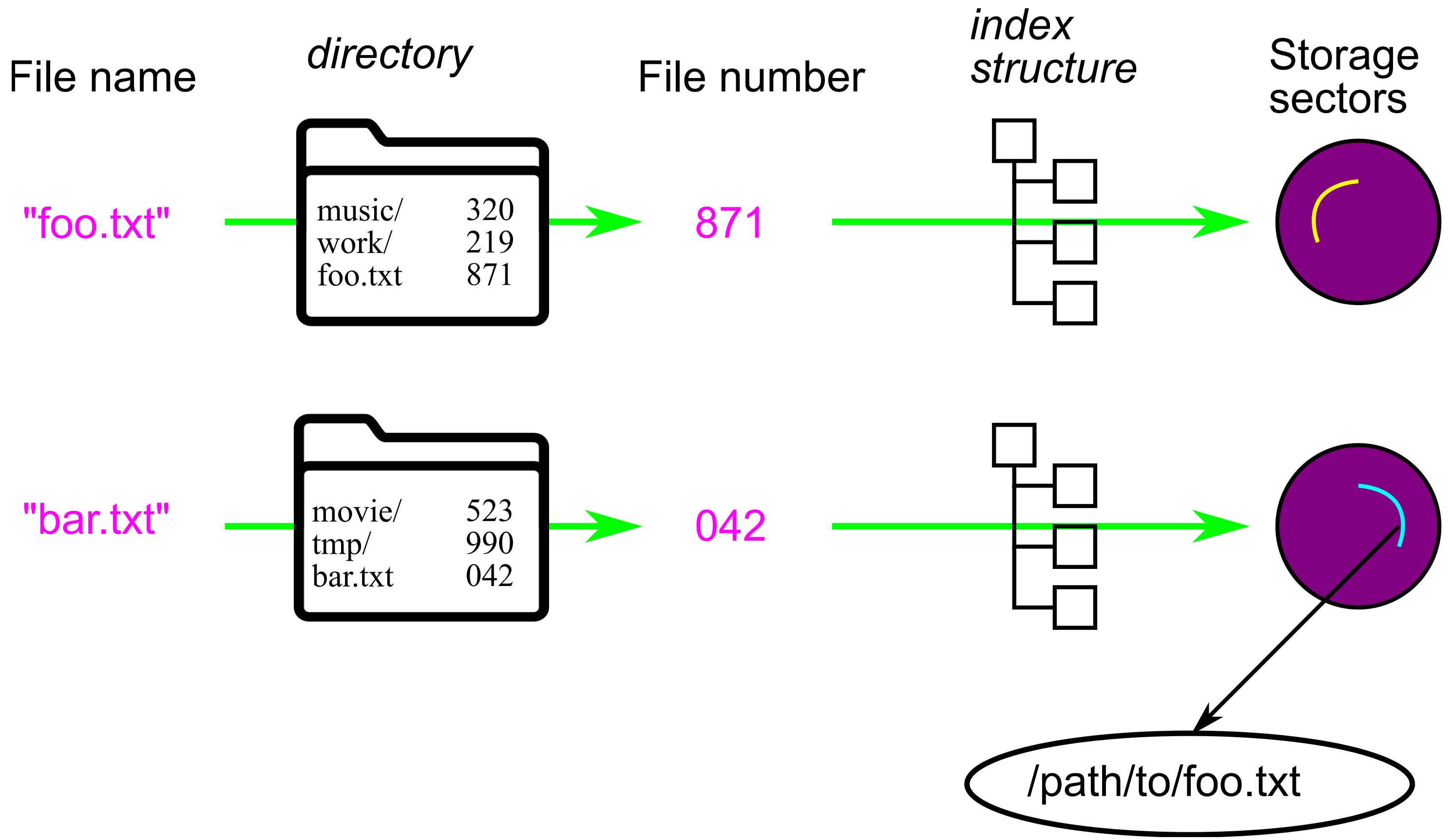
```
$ mkdir mydir
$ ln mydir mydirlink
ln: mydir: hard link not allowed for directory
```

Solution: special type of files dedicated for links.

Filesystem concepts

Soft link (aka symbolic link)

Link from name to alternate name



music/	320
work/	219
foo.txt	871

movie/	523
tmp/	990
bar.txt	042

- Symlinks can be well-identified
- File tree walkers can safely ignore them
- POSIX limit: `#define _POSIX_SYMLOOP_MAX 8`

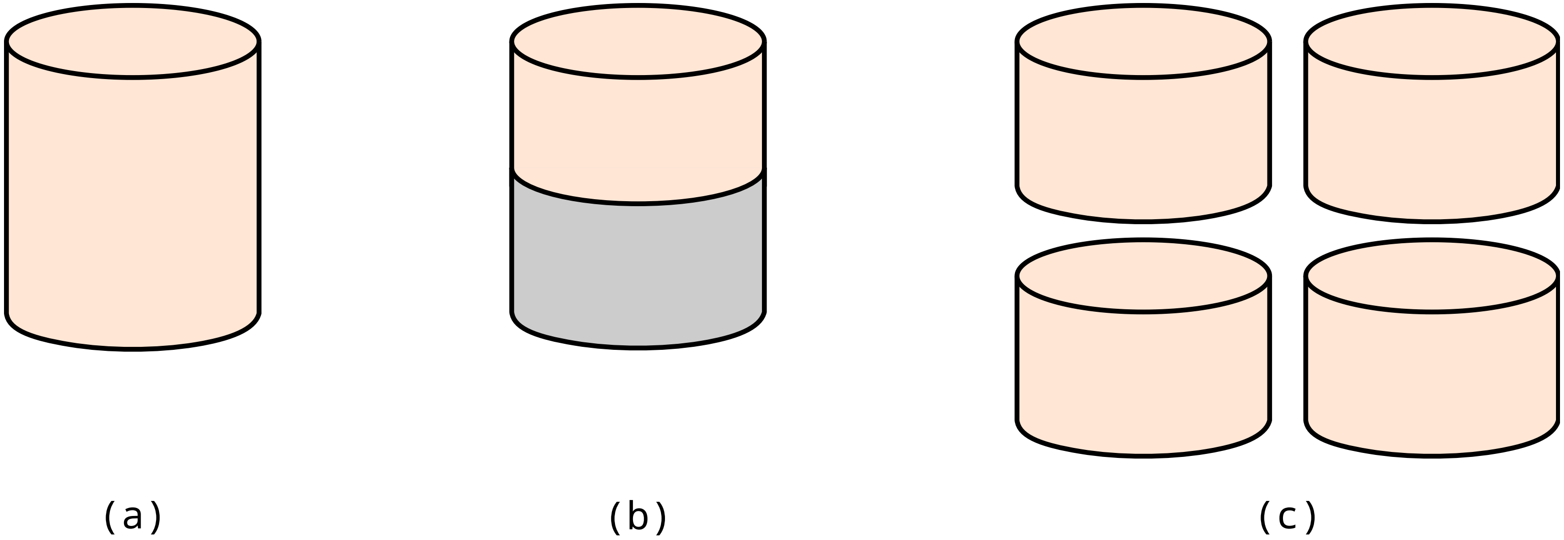
Filesystem concepts

Volume

A volume is a collection of physical storage resources that form a logical storage device containing a file-system.

A volume can be:

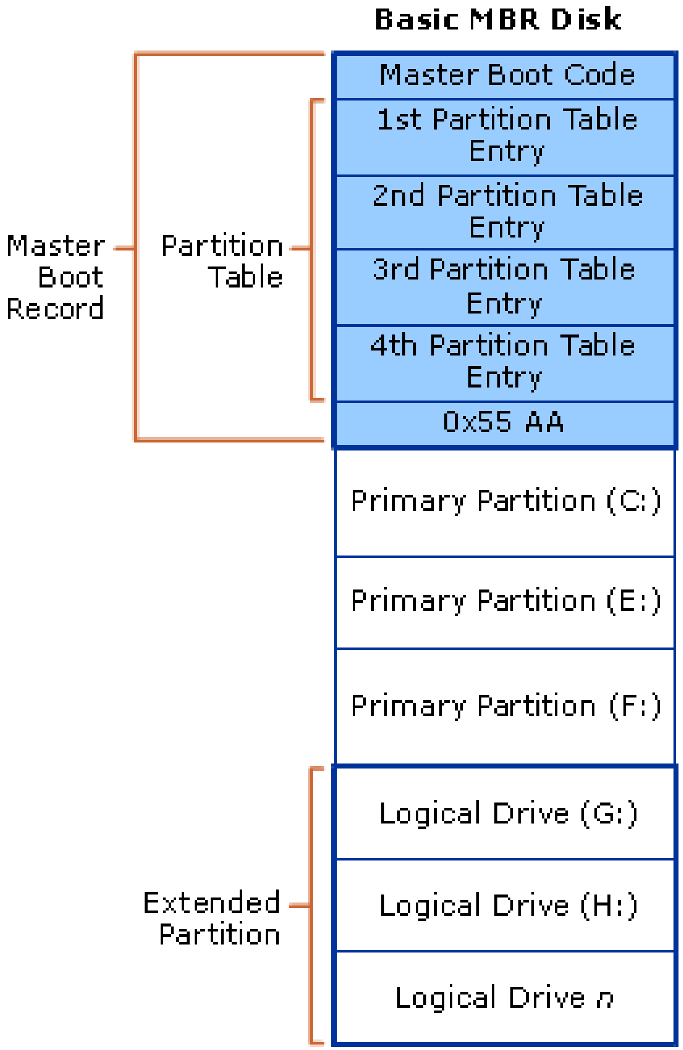
- (a) A whole disk
- (b) A partition on a disk
- (c) Multiple disks



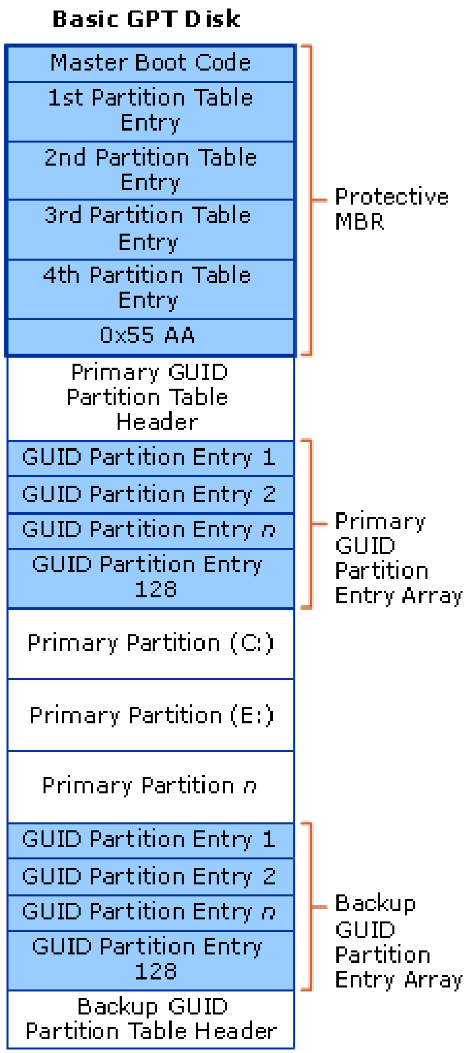
Filesystem concepts

Disk partitions

MBR (old)



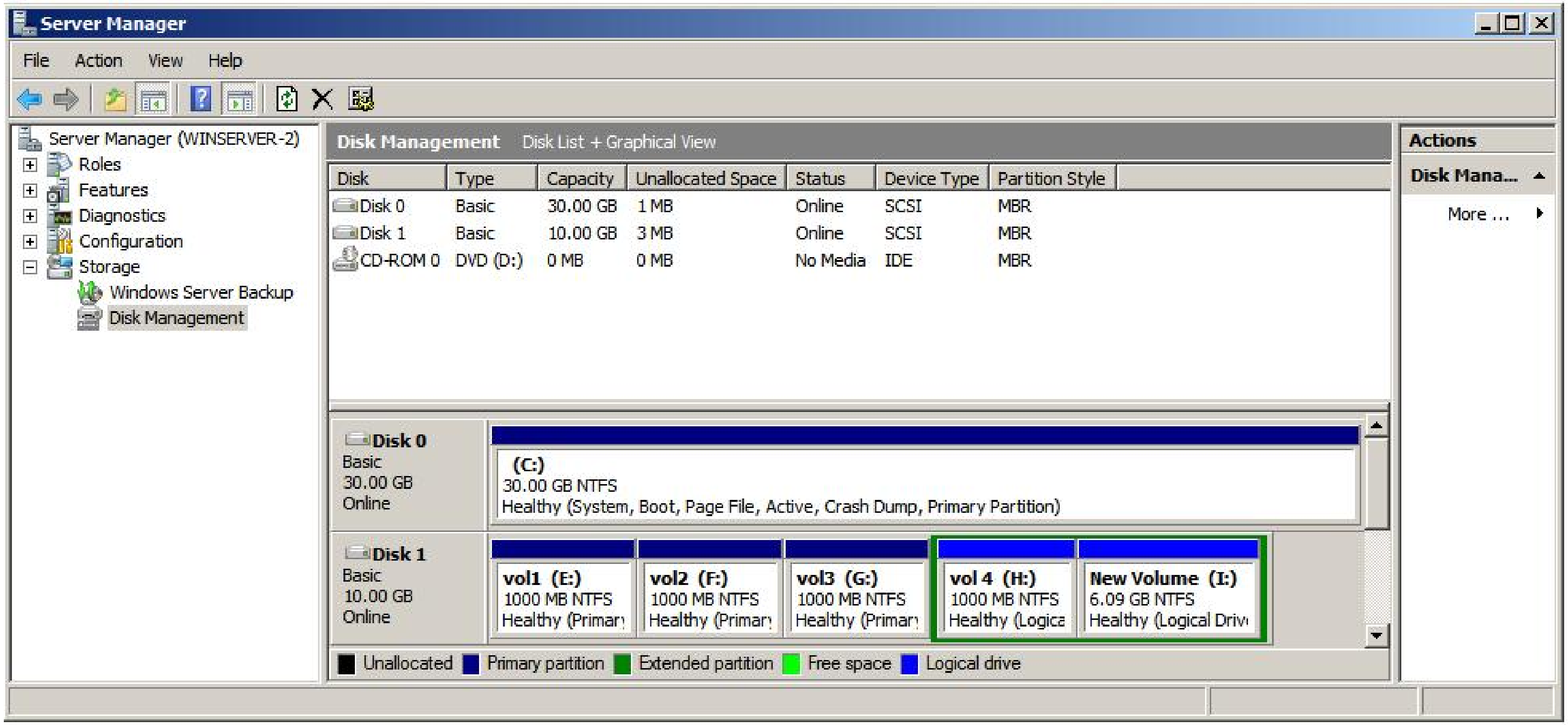
GPT (new)



Filesystem concepts

Drive letter assignment (Windows)

- Each volume holds a fully independent tree, in its own namespace
- Letter assignment typical order:
  - `A:`, `B:`: first and second floppy disk drives
  - `C:`: first active primary partition of the first physical hard drive
  - Then, first active primary partition of subsequent physical hard drives, [etc.](#)



Disk	Type	Capacity	Unallocated Space	Status	Device Type	Partition Style
Disk 0	Basic	30.00 GB	1 MB	Online	SCSI	MBR
Disk 1	Basic	10.00 GB	3 MB	Online	SCSI	MBR
CD-ROM 0	DVD (D:)	0 MB	0 MB	No Media	IDE	MBR

Disk 0
Basic
30.00 GB
Online

(C:)
30.00 GB NTFS
Healthy (System, Boot, Page File, Active, Crash Dump, Primary Partition)

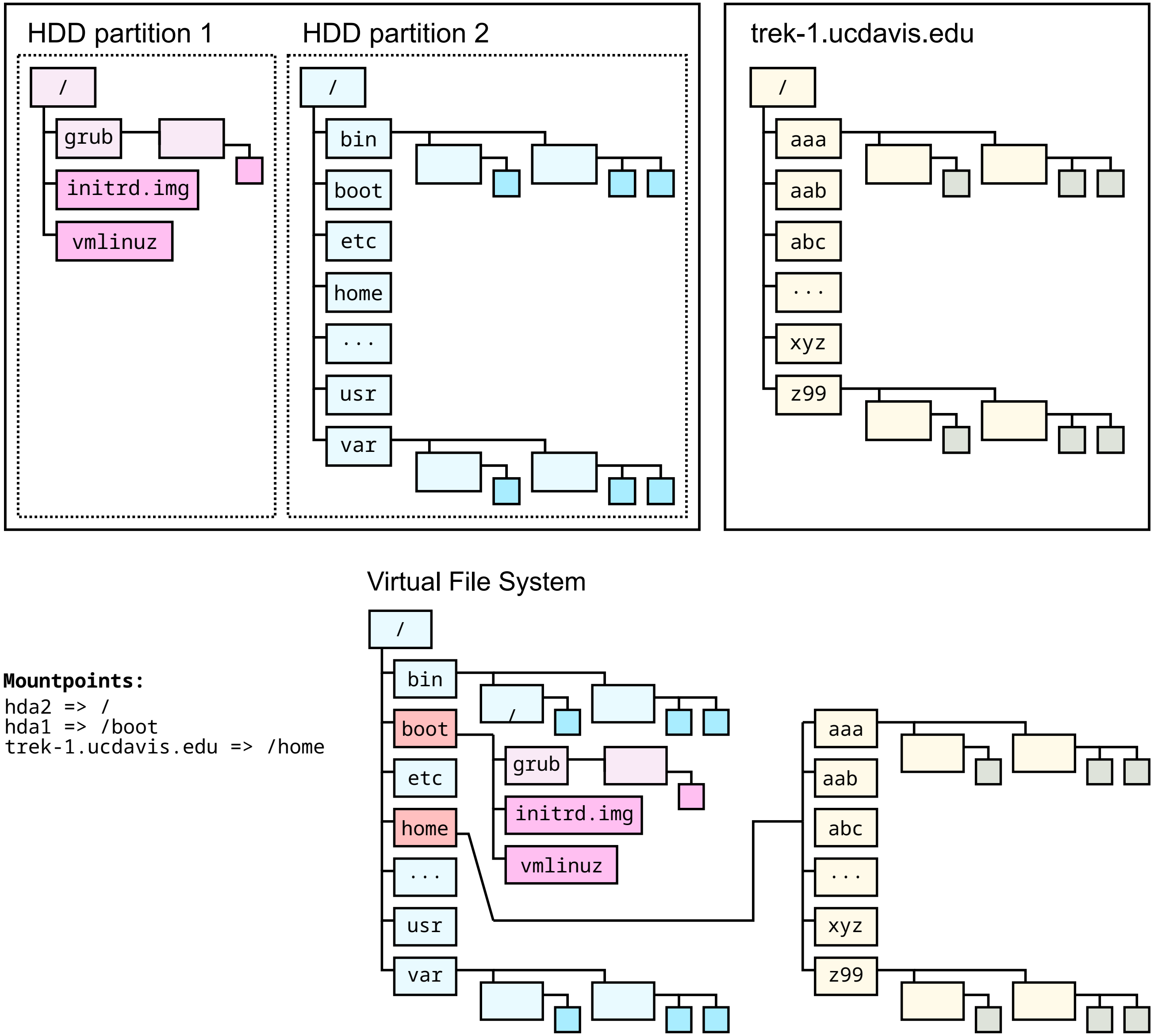
Disk 1
Basic
10.00 GB
Online

vol1 (E:)	vol2 (F:)	vol3 (G:)	vol 4 (H:)	New Volume (I:)
1000 MB NTFS	1000 MB NTFS	1000 MB NTFS	1000 MB NTFS	6.09 GB NTFS
Healthy (Primar	Healthy (Primar	Healthy (Primar	Healthy (Logica	Healthy (Logical Driv

■ Unallocated ■ Primary partition ■ Extended partition ■ Free space ■ Logical drive

Filesystem concepts

Mounting (UNIX)



```
Mountpoints:
hda2 => /
hda1 => /boot
trek-1.ucdavis.edu => /home
```

Filesystem concepts

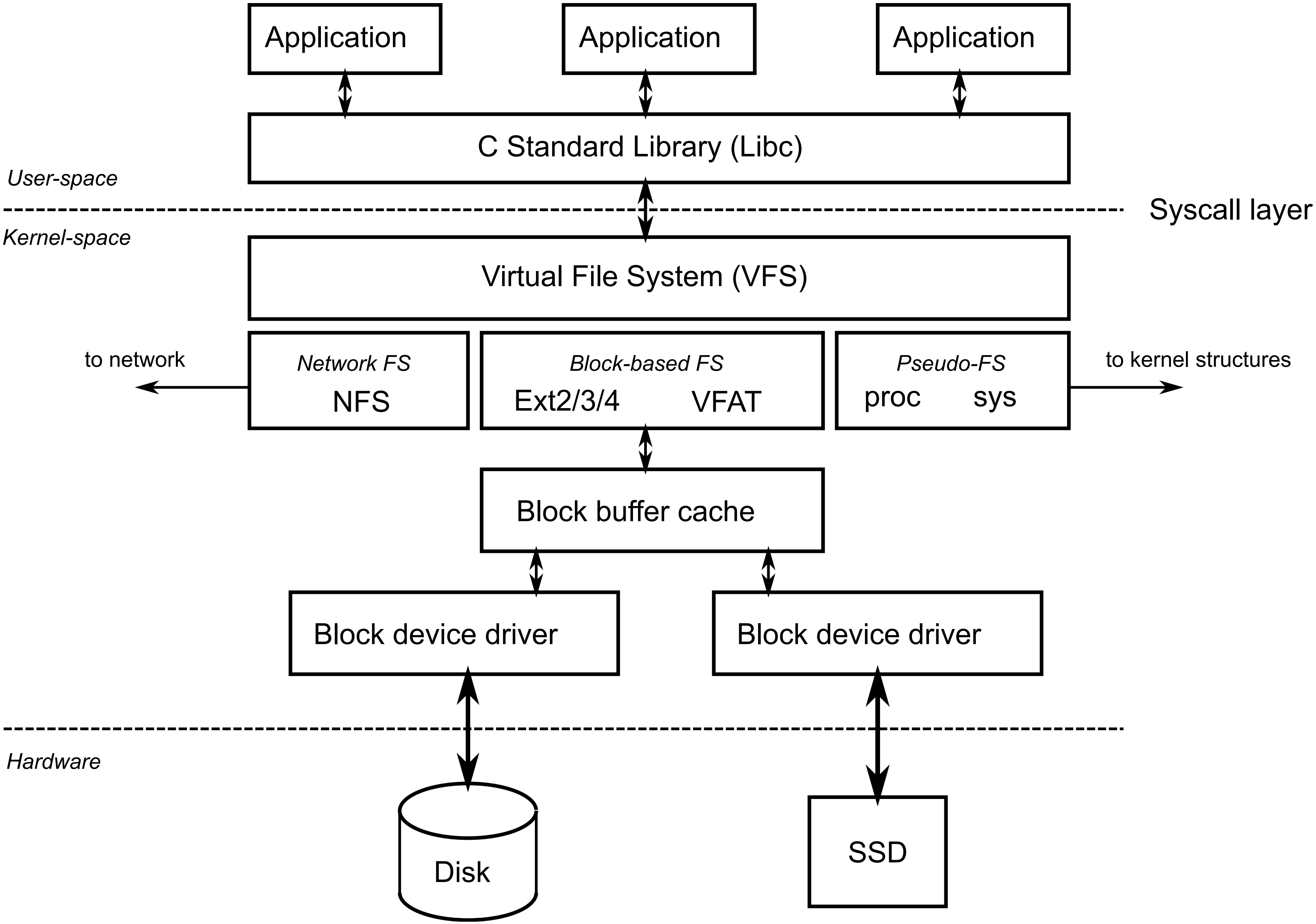
Mounting (UNIX)

Mounting multiple volumes arbitrarily in a single logical hierarchy

```
$ cat /etc/fstab
/dev/sda1            /                    ext3
/dev/sda2            /boot                ext3
/dev/sdb1            /home                ext3
/dev/cdrom           /mnt/cdrom           iso9660
/dev/sdc1            /mnt/usbkey          vfat
10.0.0.1:/shared     /srv/shared          nfs

# the following entry would not make much sense (duplicate) but is possible
#/dev/sdb1            /mnt/home                ext3
```

Filesystem software layers



Application

Application

Application

C Standard Library (Libc)

User-space

Syscall layer

Kernel-space

Virtual File System (VFS)

to network

Network FS

NFS

Block-based FS

Ext2/3/4 VFAT

Pseudo-FS

proc sys

to kernel structures

Block buffer cache

Block device driver

Block device driver

Hardware

Disk

SSD

Filesystem typical API

Create and delete files

- `create(filename, mode)`
  - create new (empty) file, including metadata and name in directory
- `link(existing_filename, new_filename)`
  - create new name for same underlying file as existing filename
- `unlink(filename)`
  - remove name for a file from its directory

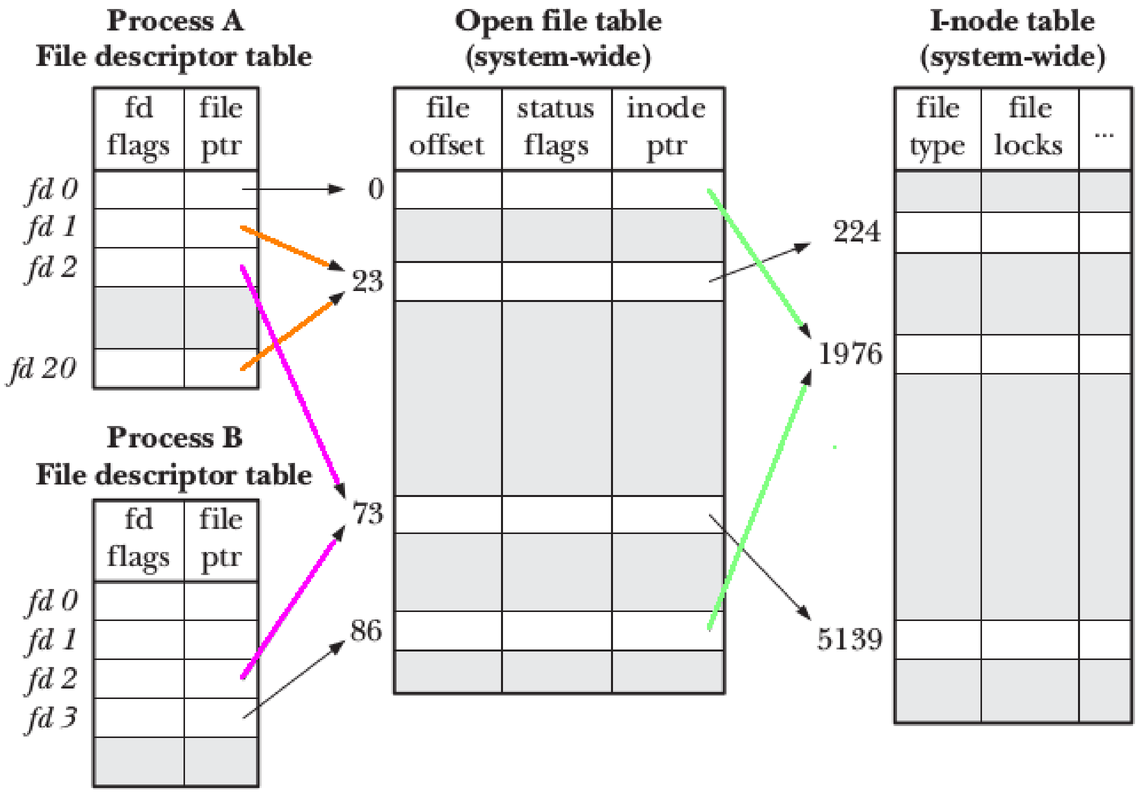
```
int fd = open("test", O_CREAT, 0644);   // also creat() but deprecated
close(fd);
link("test", "test2");
unlink("test");
```

```
$ ls -l         # After open()+link()
-rw-r--r-- 2 joel joel    0 2018-05-22 11:28 test
-rw-r--r-- 2 joel joel    0 2018-05-22 11:28 test2
$ ls -l         # After end of program
-rw-r--r-- 1 joel joel    0 2018-05-22 11:44 test2
```

Filesystem typical API

Open and close

- `open(filename)`
  - prepare to access specified file and return file descriptor
- `close(filename)`:
  - release resources associated with open file



- dup()

- fork()

- open()

*Image from "The Linux Programming Interface" by Michael Kerrisk

Filesystem typical API

File access

- `read()`, `write()`: sequential reading/writing
- `seek()`: change file's current position for random access

```
fd = open(...)
lseek(fd, 42, SEEK_CUR)
/* Write in open file at offset 42 */
char c = 'a';
write(fd, &c, 1);
close(fd);
```

- `mmap()`: create mapping between file's content and memory
- `munmap()`: destroy mapping

```
fd = open(...)
char *address = mmap(0, len, PROT_WRITE, MAP_SHARED, fd, 0)
address[42] = 'a';
munmap(address, len);
close(fd);
```