Files: Organization

Each file has a name

Each file has a parent directory

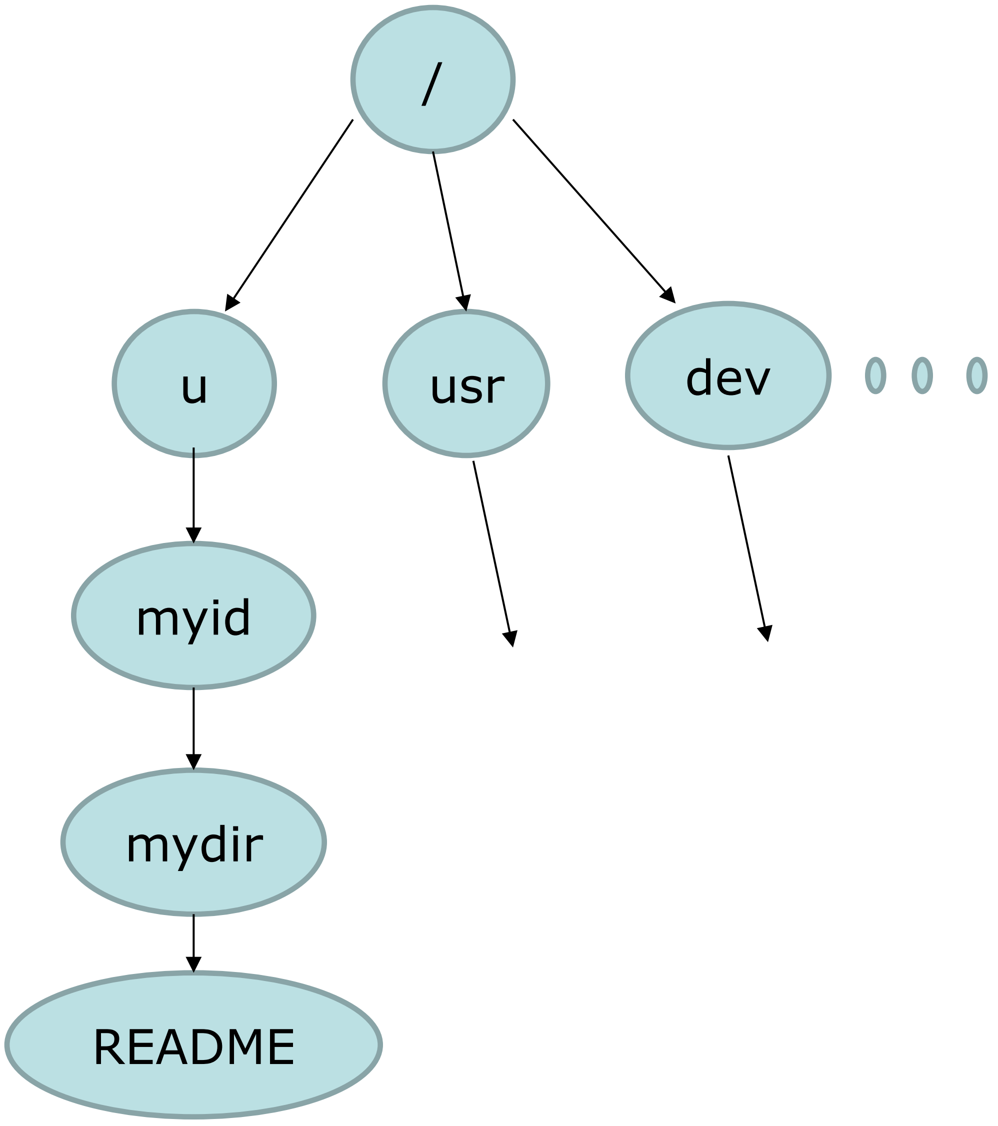
Each directory has a parent directory

/ is the ”root” or top-most ancestor

Files: key abstractions

Directory

- Functions much like a file
  - Has a name, metadata
  - is contained within a parent Directory
- But contains specific contents
  - List of contained files and directories


/
u
usr
dev
myid
mydir
README

Files: Naming

Absolute: **`/u/myid/mydir/README`**

Relative: **`README`**

Special directories: `‘.’, ‘..’, ‘~’`

Illegal characters: `‘/’, NULL`

Avoid these: `‘>’, ‘<‘, ‘|’, ’&’, etc.`

Length: 255 characters (bytes?)

Filename must be unique within directory

File System API **(POSIX)**

```
read(fd,…), write(fd,…)
lseek(fd,…)  // for random access


struct file {  // xv6 file structure
  int ref;
  char readable;
  char writable;
  sruct inode *ip;
  uint off;  // offset into the file
};
```

Directory APIs

Back in the olden days (pre 1990) Unix provided only one type of file system.

A Directory was just a File

With a very specific, well-known file format

So you could just use standard Unix file I/O syscalls: open(), read(), write(), close()

But then more file system implementations

ext*, ZFS, NTFS, XFS, LFS, jfs, Btrfs, etc.

Linux Directory APIs

The new fs implementations use complex directory file formats.

So in the slightly less olden days Linux introduced opendir(), readdir(), writedir(), etc.

But for large systems this was inefficient. Readdir() retrieved one directory entry at a time. Many OS context switches for large directories.

No batching → inefficient

Linux Directory APIs

`getents()` – retrieve many entries from a given directory

Then they rewrote **`readdir()`**, etc. as Clib functions on top of **`getents()`**.

So existing software works, just better

Lessons: batching good. well-defined APIs good

File and Directory Permissions

Each file/directory has a list of permission bits:

```
-rw-r--r--
```

The first bit indicates the type of the entity: file, directory, link, special file, named pipe, something else

The other bits indicate permissions: octal: **644**

Permissions are in three groups: owner, group, other

Each bit indicates “read”, “write” or “execute” permission

Directory needs execute bit?

The original unix file system introduced several hacks to save space and time, and one of them was to overload the meaning of “execute”

For a Directory, the execute permission actually means “can chdir() to this directory” permission.

There are more hacks, see **`chmod()`** man page for more information about sticky bits and setuid bit. setuid bits and sticky bits)

File System ACLs

ACL: Access Control List

- A listing of which users have which permissions
- Much flexible than simple unix permissions
- More complex to implement
- Existed in some 1960s research systems
- Later introduced by HP into HPUX (90s)
- Eventually standardized in POSIX (1997)
- Now supported in most OS distributions
- Essential for Role-Based Administration

Mounting File Systems

Goal: utilize multiple storage devices

Goal: maintain a coherent, abstract naming system

Use **`mkfs()`** to make a new file system on a device

Then use **`mount()`** to insert it into a specific place in the file system name hierarchy

And **`unmount()`** when done.

Quick Quiz

What is a File? Directory?

What command do you use to list files in a directory?

What system call retrieves entries from an open directory?

What’s the difference between

`drwx------` and **`-rwx------`** ?

What command is used to remove a file?

What is a file descriptor?

When might **`fsync()`** not be good enough?

What does the **`unlink()`** system call do?

What Is a File?

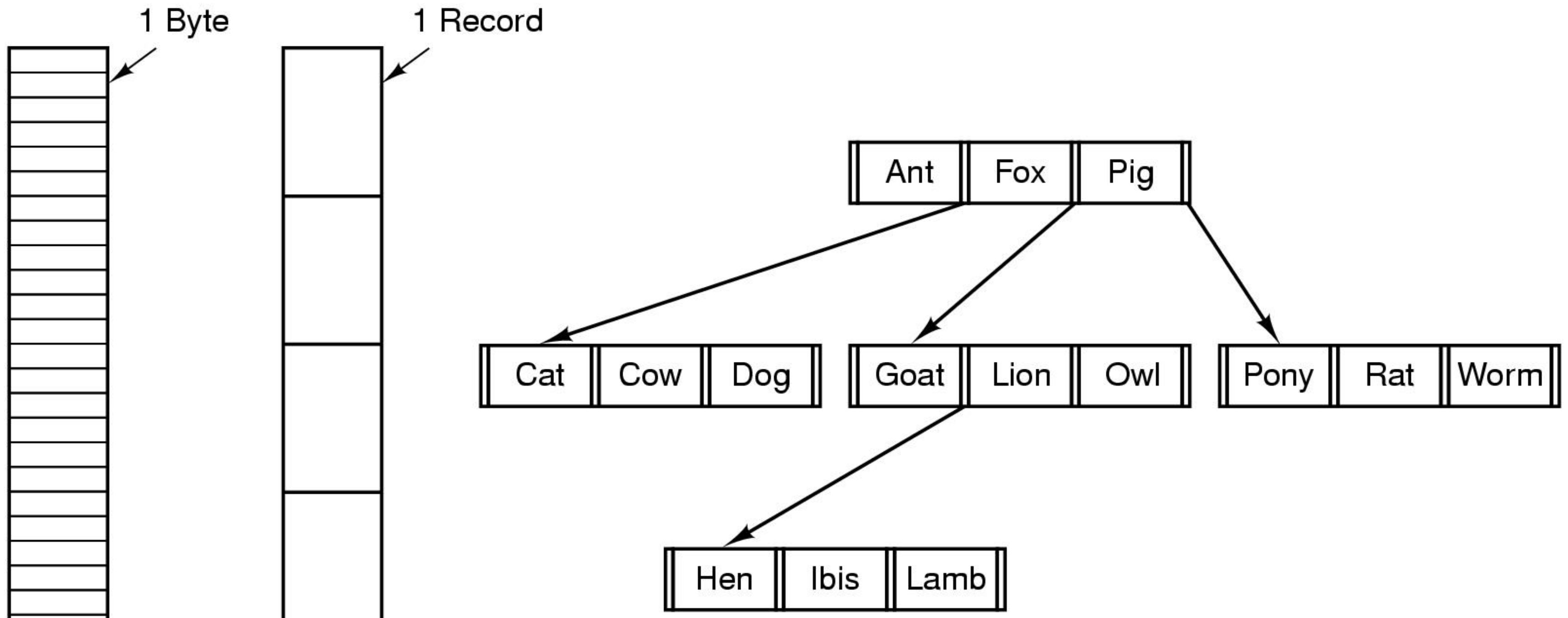
Files can be structured or unstructured

- Unstructured: just a sequence of bytes
- Structured: a sequence or tree of typed records

In Unix-based operating systems a file is an unstructured sequence of bytes

- similar to memory

File Structure



Sequence of bytes

Sequence of records

Tree of records

Unix/Linux File Extensions

Even though files are just a sequence of bytes, programs can impose structure on them, by convention

- Files with a certain standard structure imposed can be identified using an extension to their name
- Application programs may look for specific file extensions to indicate the file's type
- But as far as Unix/Linux is concerned its just a sequence of bytes

Typical File Extensions

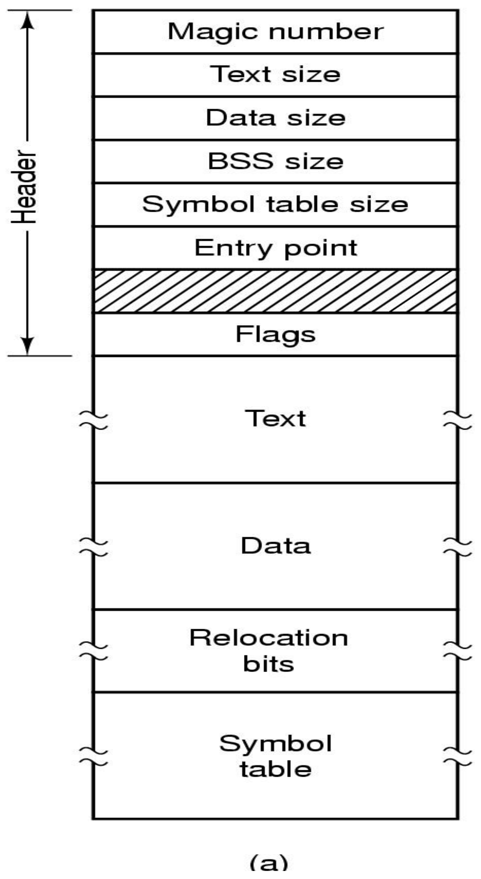
Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Executable Files

Executable files are special

- The OS must understand the format of executable files in order to execute programs
- The exec system call needs this information
  - Exec puts program and data in process address space

Executable File Format



An executable file

File Attributes

Various meta-data needs to be associated with files

- Owner
- Creation time
- Access permissions / protection
- Size etc

This meta-data is called the file attributes

- Maintained in file system data structures for each file
- Stored in the I-node in Unix file systems

Memory-Mapped Files

Conventional file I/O

- Use system calls (e.g., open, read, write, ...) to move data from disk to memory

Observation

- Data gets moved between disk and memory all the time without system calls
    - Pages moved to/from PAGEFILE by VM system
- Do we really need to incur system call overhead for file I/O?

Memory-Mapped Files

Why not “map” files into the virtual address space

- Place the file in the “virtual” address space
- Each byte in a file has a virtual address

To read the value of a byte in the file:

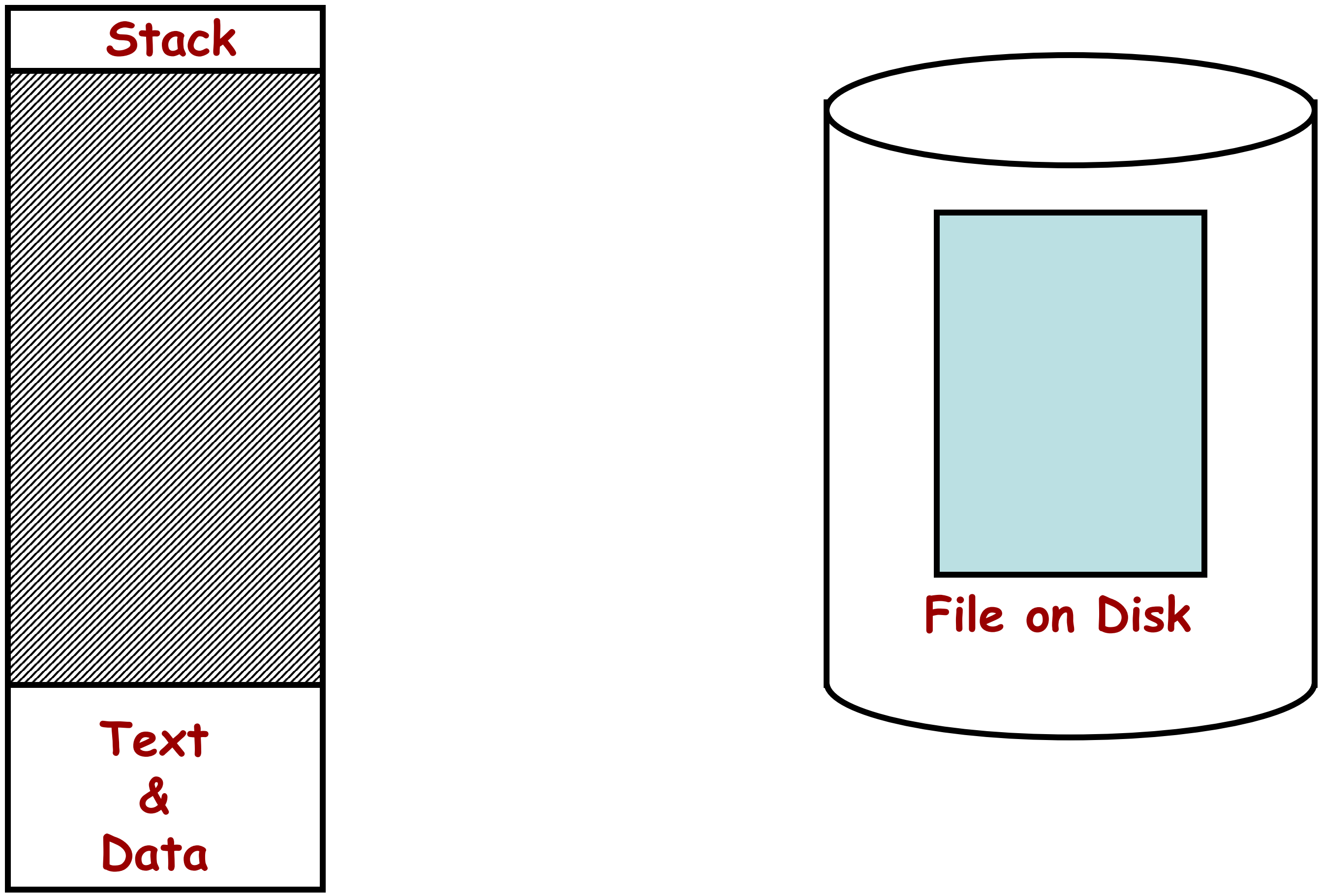
- Just load that byte’s virtual address
    - Calculated from the starting virtual address of the file and the offset of the byte in the file
- Kernel will fault in pages from disk when needed

To write values to the file:

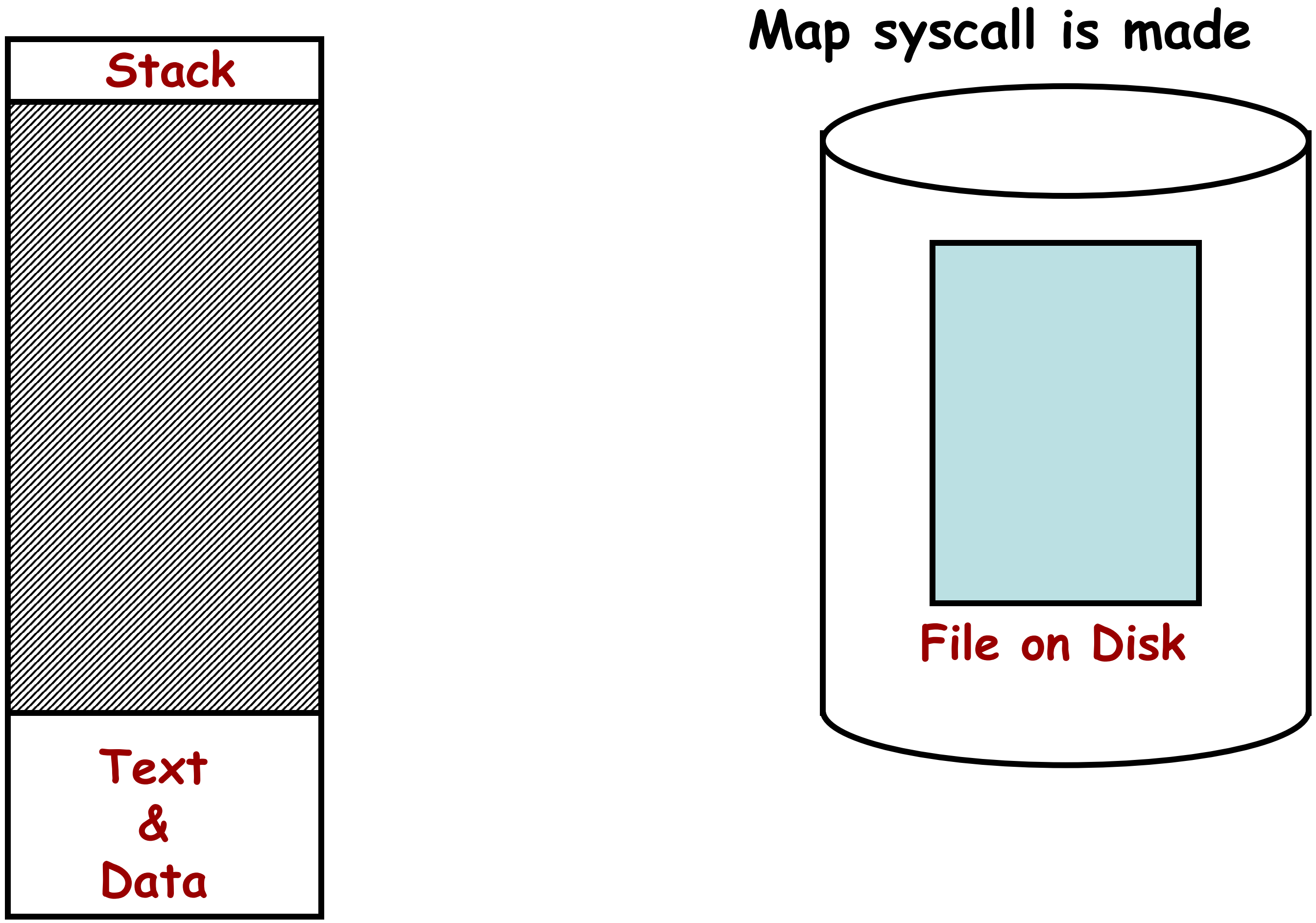
- Just store bytes to the right memory locations

Open & Close syscalls $\rightarrow$ Map & Unmap syscalls

Memory-Mapped Files

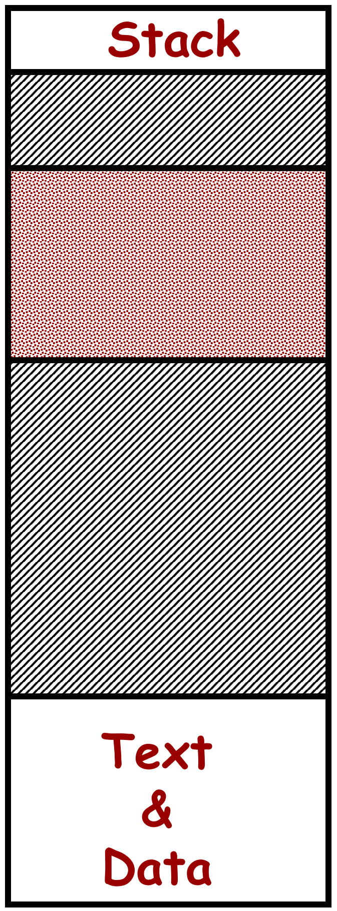


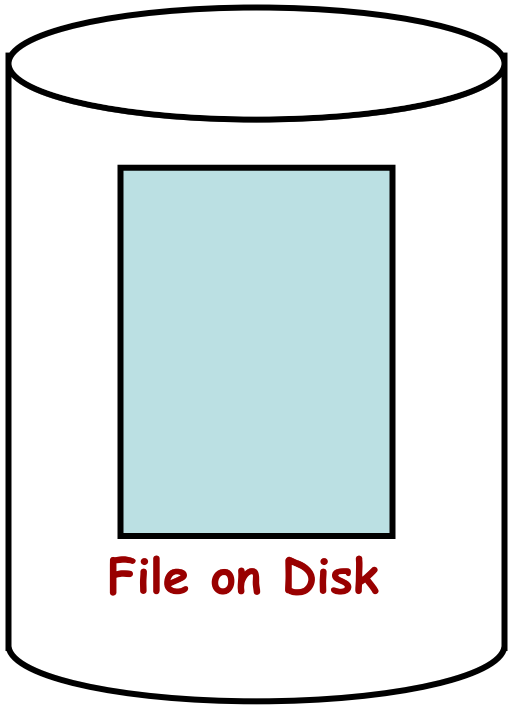
Memory-Mapped Files



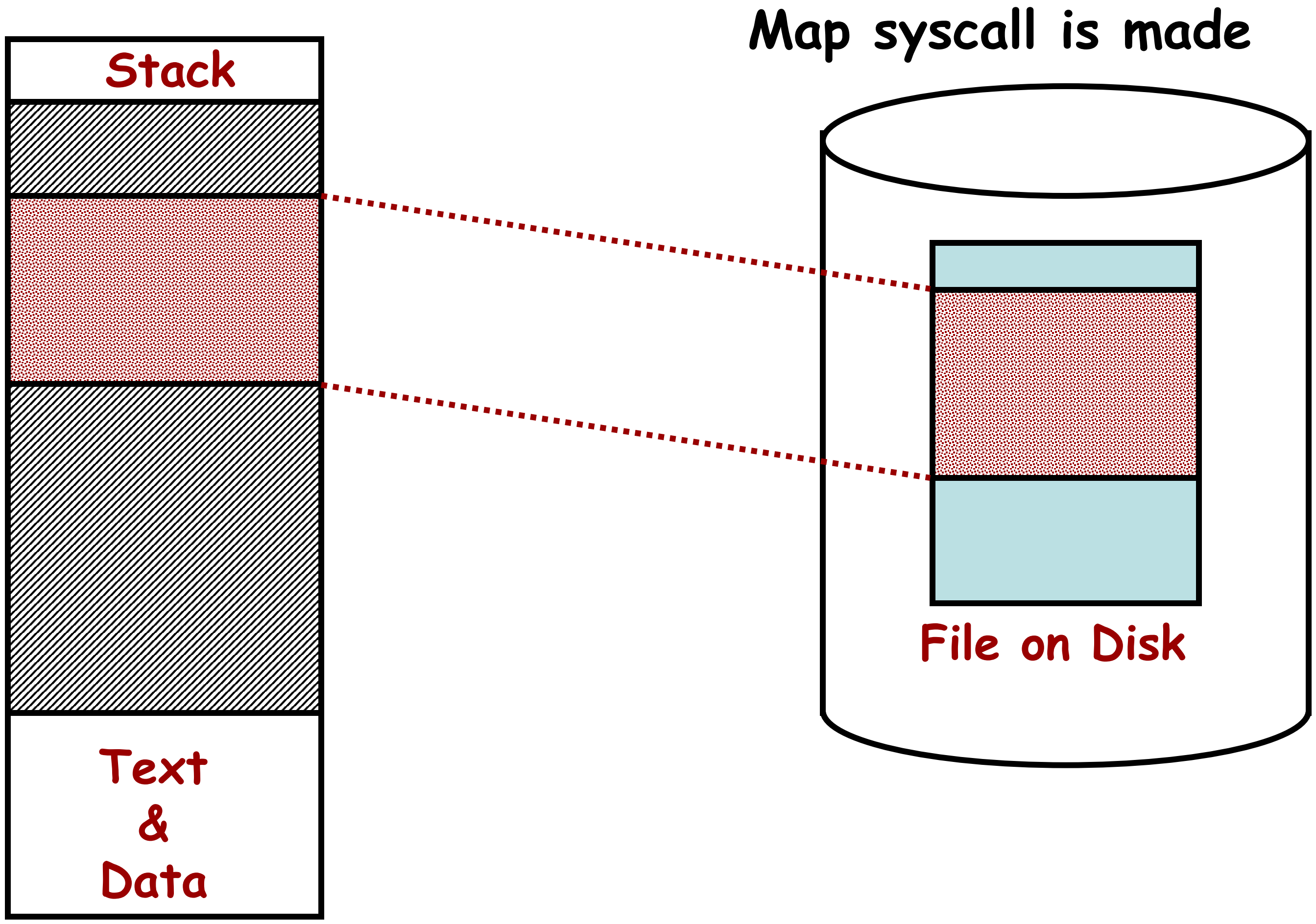
Memory-Mapped Files

Map syscall is made

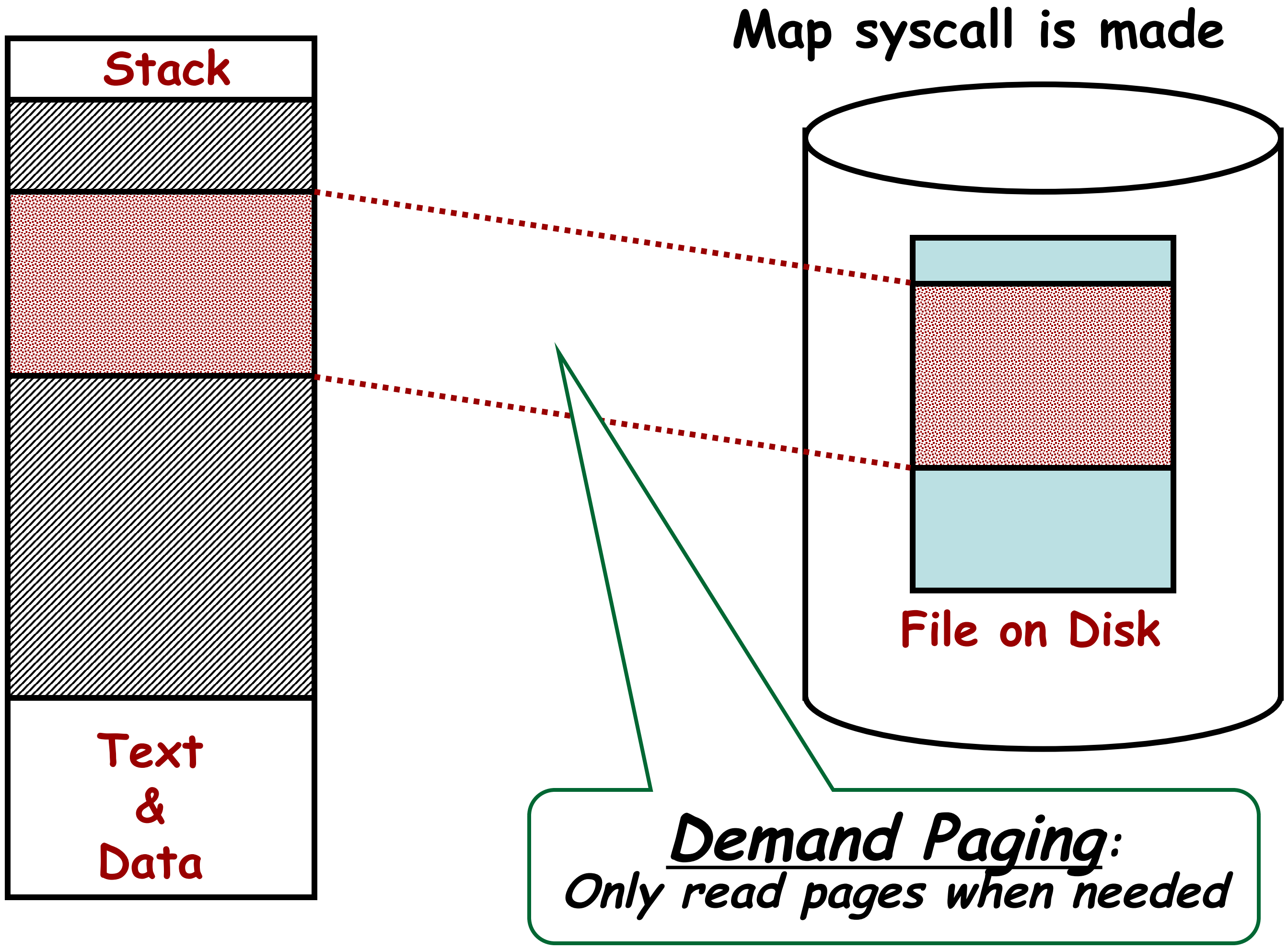




Memory-Mapped Files



Memory-Mapped Files



File System Consistency

Invariant:

  Each disk block must be

    in a file (or directory), or

    on the free list

What inconsistent states can arise? Why?

Old solution: fsck (file system checker)

Newer solution: Journaling

File System Consistency

Inconsistent States:

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list (“missing block”)

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list (“missing block”)

- Some block is on free list and is in some file

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list (“missing block”)
- Some block is on free list and is in some file
- Some block is on the free list more than once

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list (“missing block”)
- Some block is on free list and is in some file
- Some block is on the free list more than once
- Some block is in more than one file

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list (“missing block”)

  Add it to the free list.
- Some block is on free list and is in some file
- Some block is on the free list more than once
- Some block is in more than one file

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list ("missing block")

  Add it to the free list.
- Some block is on free list and is in some file

  Remove it from the free list.
- Some block is on the free list more than once
- Some block is in more than one file

File System Consistency

Inconsistent States:

- Some block is not in a file or on free list ("missing block")

  Add it to the free list.
- Some block is on free list and is in some file

  Remove it from the free list.
- Some block is on the free list more than once

  (Can't happen when using a bitmap for free blocks.)

  Fix the free list so the block appears only once.
- Some block is in more than one file

File System Consistency

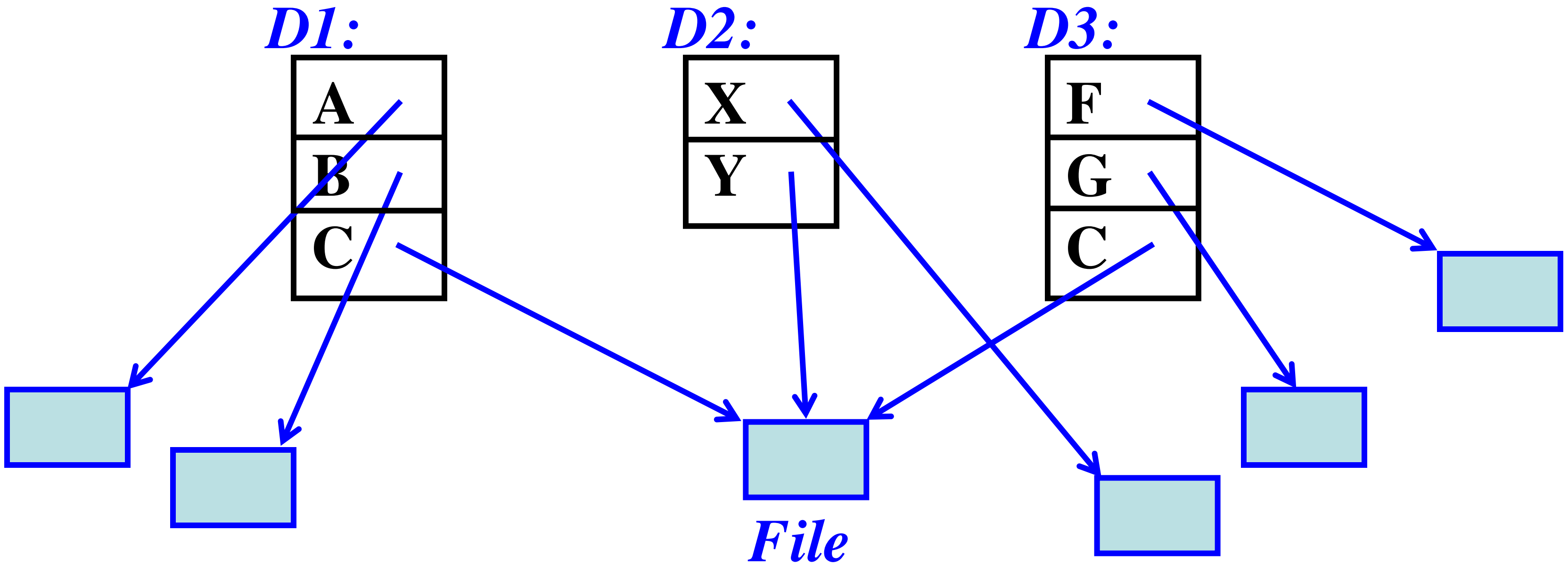
Inconsistent States:

- Some block is not in a file or on free list ("missing block")
  Add it to the free list.
- Some block is on free list and is in some file
  Remove it from the free list.
- Some block is on the free list more than once
  (Can't happen when using a bitmap for free blocks.)
  Fix the free list so the block appears only once.
- Some block is in more than one file
  Allocate another block.
  Copy the block.
  Put each block in each file.
  Notify the user that one file may contain data from another file.

File System Consistency

Invariant (for Unix):

"The reference count in each i-node must be equal to the number of hard links to the file."



File System Consistency

Problems:

- Reference count is too large

- Reference count is too small

Portland State UNIVERSITY

File System Consistency

Problems:

- Reference count is too large
  - The “rm” command will delete a hard link
  - When the count becomes zero, the blocks are freed
  - Permanently allocated; blocks can never be reused
- Reference count is too small

File System Consistency

Problems:

- Reference count is too large
  - The “rm” command will delete a hard link
  - When the count becomes zero, the blocks are freed
  - Permanently allocated; blocks can never be reused
- Reference count is too small
  - When links are removed, the count will go to zero too soon!
  - The blocks will be added to the free list, even though the file is still in some directory!

File System Consistency

Problems:

- Reference count is too large
  - The “rm” command will delete a hard link
  - When the count becomes zero, the blocks are freed
  - Permanently allocated; blocks can never be reused
- Reference count is too small
  - When links are removed, the count will go to zero too soon!
  - The blocks will be added to the free list, even though the file is still in some directory!

Solution:

- Correct the reference count!