File System

part I

CS170

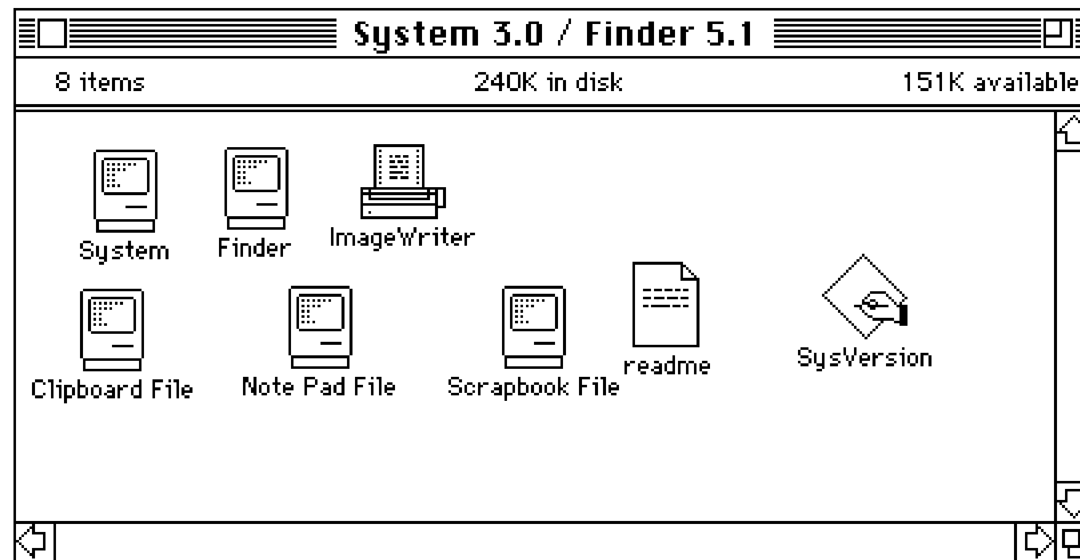
File System

Interface to the *secondary storage*

(Among other things)

1. User's view (abstractions)
2. Bottom up view (implementation)
3. Performance optimizations

User's view



Data & metadata

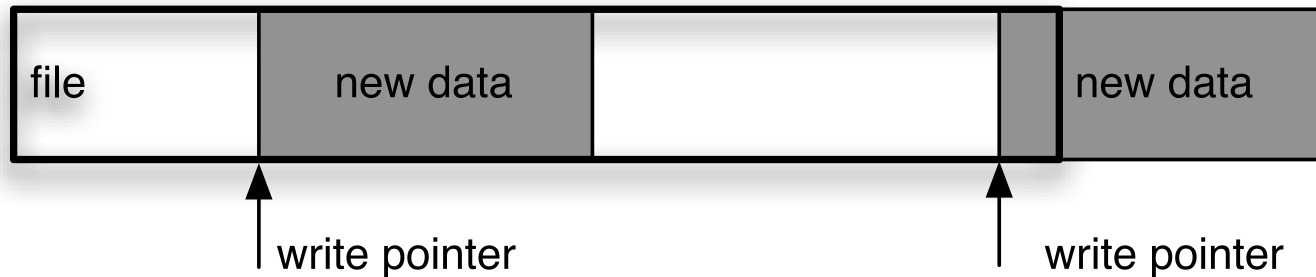
- **File:** logical storage unit (for data)
- **Attributes** (metadata)
  - name
  - size
  - location (on disk)
  - protection (including ownership)
  - time: of creation, modification, access

Operations

- **Create:** allocate space for data & metadata
  - Check proposed name and permissions
- **Delete:** release data & metadata
  - Typically does not destroy the data
- **Truncate:** shrink data to 0, keep metadata

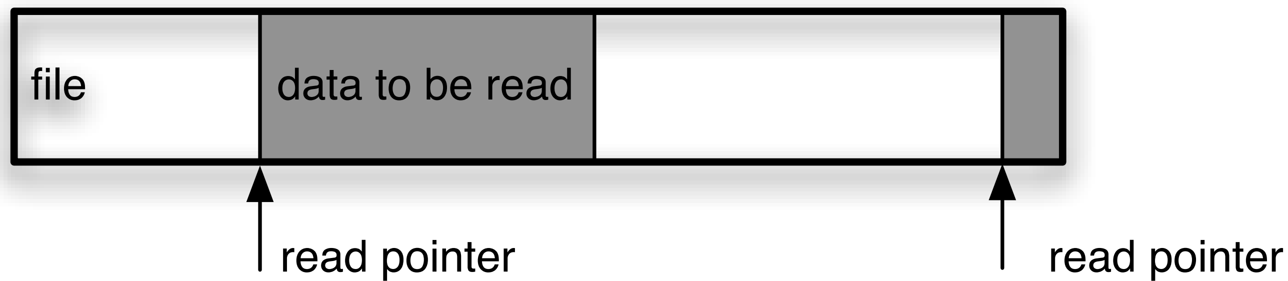
Operations

Write: *which* file, *what* to write, and *where*



Operations

Read: *which* file, *how much* to read, and *where*



“Conveniences”

- Current file position
  - **Seek**: moves pointer (no I/O)
- Open file table
  - **Open**: returns an index into it (fd)
  - **Close**: frees up the entry
  - Obviates lookups upon every operation

Access Modes

- **Sequential** (think tape)
  - Read next
  - Write next
  - Rewind
- **Direct** (think disk/memory)
  - Read next *or* Read at position
  - Write next *or* Write at position
  - Move to position

Inside files

- Sequence of logical *records*
  - bytes, words, structures
- Special format for executables
- Resource fork, file creator on Mac OS
- Some names (e.g. command.com) are special

Organizing files: single-level directory

- aka "flat namespace"
- Have you seen this? Napster!

Organizing files: tree-based directories

- Directory as a special file
- System calls for creating and deleting
  - What if not empty?
- Current directory
  - Chdir() system call
  - Absolute/relative paths
  - Search path

Organizing files: beyond trees

- DAG allows file and directory sharing, but:
  - Absolute paths are not unique
  - Deletion semantics
  - Cycles are a danger
- Implementations:
  - Symbolic (soft) links
  - True (hard) links

Protection

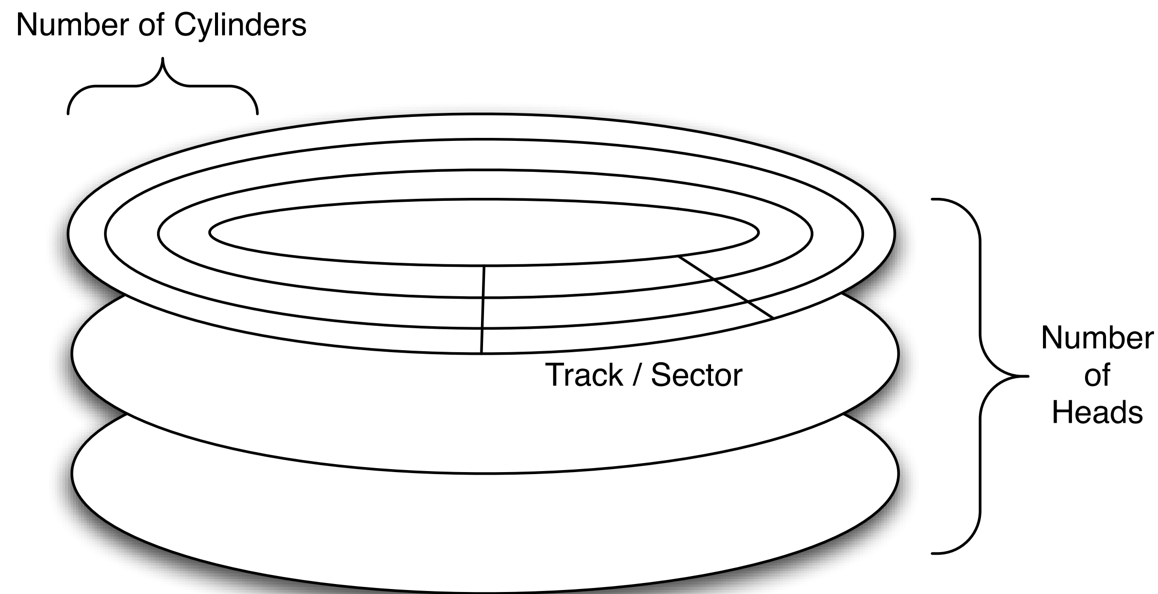
- Protect users and system components from each other through file access restrictions
- Operations:
  - Low-level: read, write, execute, delete, list
  - High-level: rename, copy, edit, print
- Policy: which ops, which users, which files

Access lists

- Alternative approaches:
  - For each file/dir, list valid ops for each user
  - For each user, list files that they can op
- Either list can be large, so:
  - Group users: *owner, group, universe*
  - Simplify operations: *rwx* (for files & dirs)

Bottom up view

Head, cylinder, sector

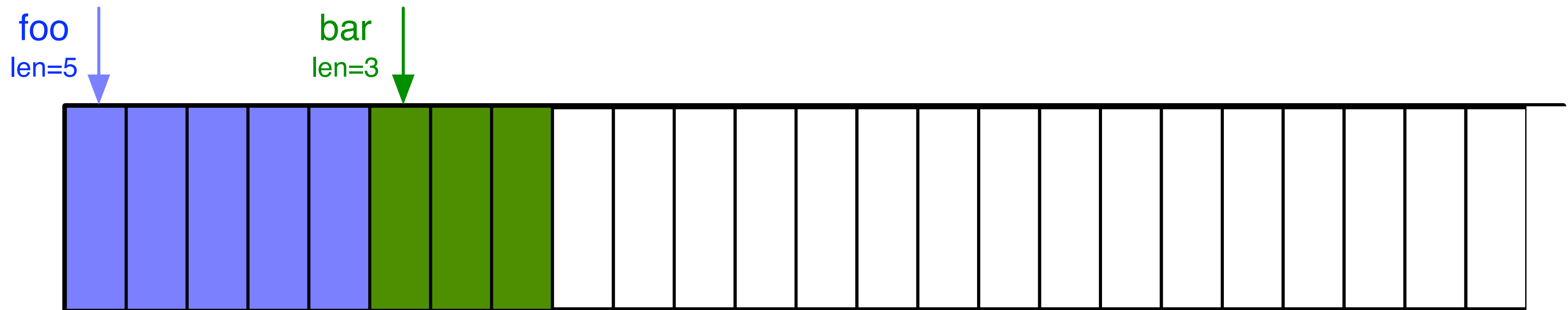


Block = Sector + Sectors-per-track *
(Head + Cylinder * Tracks-per-Cylinder)

Disk Allocation

- Where to put blocks of file data?
  - View disk as a contiguous array of sectors
  - Ignore the hierarchy for now
- Files can be mapped as:
  - Variable-sized, contiguous “portions”
  - Fixed-sized, scattered “blocks”

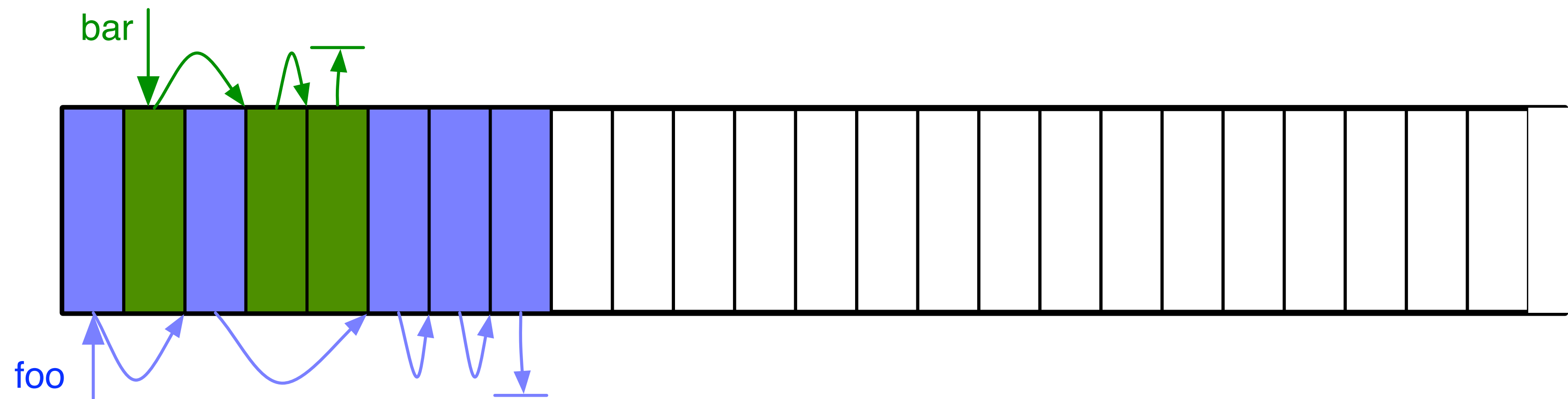
Contiguous Allocation



Contiguous allocation

- Pros:
  - Efficient direct and sequential access
  - Minimal overhead for metadata
- Cons:
  - Finding open slots
    - External fragmentation ⇒ compacting
  - "Outgrowing" the block
- Use "extent" to extend the file?

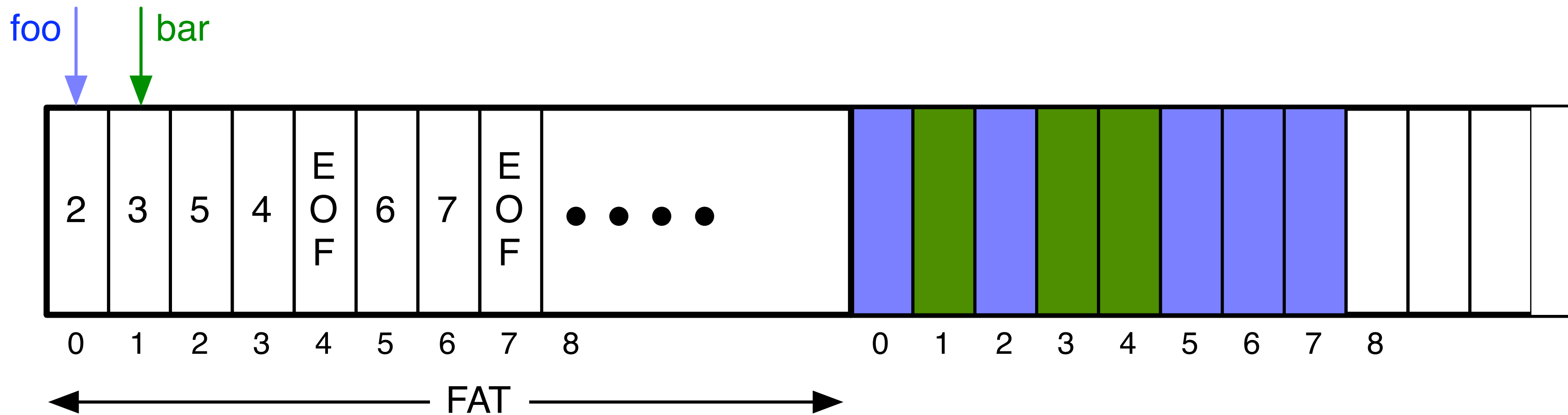
Linked allocation



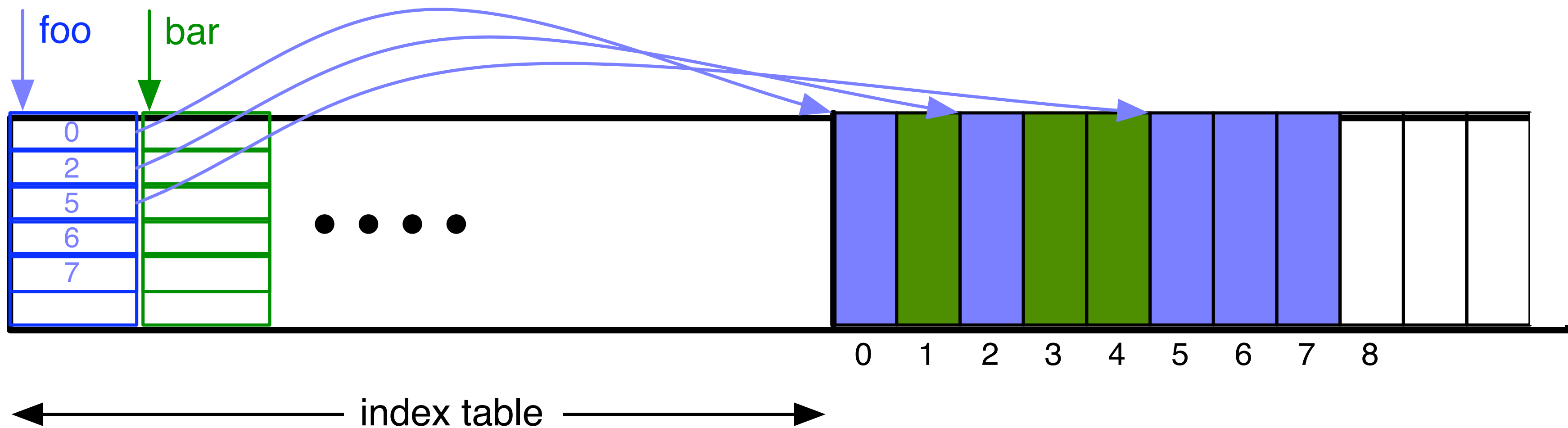
Linked allocation

- Pros:
  - No external fragmentation
  - Unlimited growth
- Cons:
  - Slow random access
  - Overhead for pointers
- Keep the links close together: FAT

Linked allocation: FAT



Indexed allocation



Indexed allocation:

- Pros:
  - No external fragmentation
  - Efficient random access
- Cons:
  - Link overhead (more than with linked)
- Outgrowing the index block:
  - Chain, multiple levels, both