Today

- File Systems
  - Roles
  - Structures

Review

- What is the role of the file system?
- What information do file systems keep track about files?
  - What data structures do they use to keep track of them?
  - What does Classical Unix use?
- Why does a file system keep track of a file id as well as a name?

File System as Illusionist: Hide Limitations of Physical Storage

- Persistence of data stored in file system:
  - Even if crash happens during an update
  - Even if disk block becomes corrupted
- Naming:
  - Named data instead of disk block numbers
    - Files, directories
  - Directories instead of flat storage
  - Byte addressable data even though devices are block-oriented

File System as Illusionist: Hide Limitations of Physical Storage

- Performance:
  - Achieve close to the hardware limit in the average case
  - Cached data
  - Data placement and data structure organization
- Controlled access to shared data

Defragmenting

https://en.wikipedia.org/wiki/Defragmentation

File System Design Constraints

- For small files:
  - Small blocks for storage efficiency
  - Files used together should be stored together
- For large files:
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation whether file will become small or large

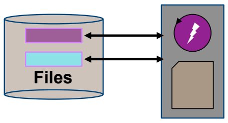
File System Design

- Data structures
  - Directories: file name -> file metadata
    - Store directories as files
  - File metadata: how to find file data blocks
  - Free map: list of free disk blocks
- How do we organize these data structures?
  - Device has non-uniform performance

Design Challenges

- ✓ Index structure
  - How do we locate the blocks of a file?
- ✓ Index granularity
  - What block size do we use?
- Free space
  - How do we find unused blocks on disk?
- Locality
  - How do we preserve spatial locality?
- Reliability
  - What if machine crashes in middle of a file system op?

Files



Unix file syscalls

```
fd = open(name, <options>);
write(fd, “abcdefg”, 7);
read(fd, buf, 7);
lseek(fd, offset, SEEK_SET);
close(fd);

creat(name, mode);
fd = open(name, mode, O_CREAT);
mkdir (name, mode);
rmdir (name);
unlink(name);
```

A **file** is a named, variable-length sequence of data bytes that is **persistent**: it exists across system restarts, and lives until it is removed.

An **offset** is a byte index in a file. Programs may read and write files **sequentially** or **seek** to a particular offset and read/write there.

- called a "logical seek" because it seeks to a particular location in the *file*, independent of where that data actually resides on *storage* (it could be anywhere).

File Access

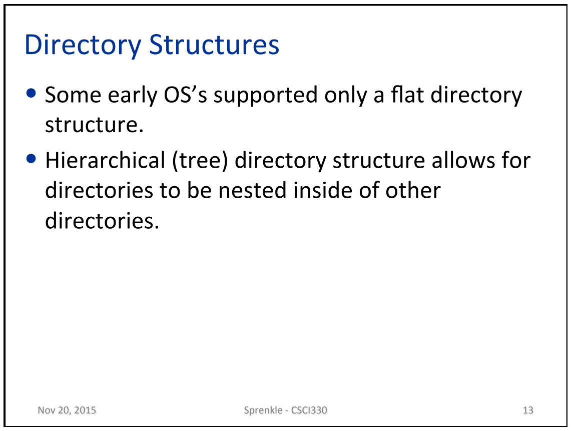
- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was magnetic tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker
- *Keyed* (or indexed) access – usually DBs

File System Data Structures

- File Descriptor
  - One for every file or directory on the disk is maintained on the disk.
- File Structure
  - One for each file or directory that is opened by a process.
- Open File ID Table
  - An array of pointers to all of the Open File Structures.
  - One for the **entire system**.
- Open File Structure:
  - One for each file or directory that is open.

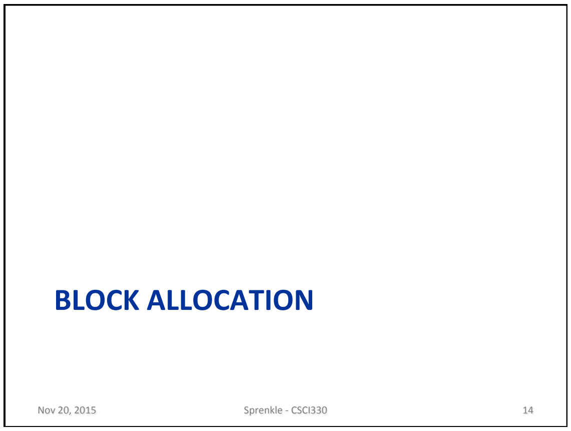
File organization

- How are files typically organized?

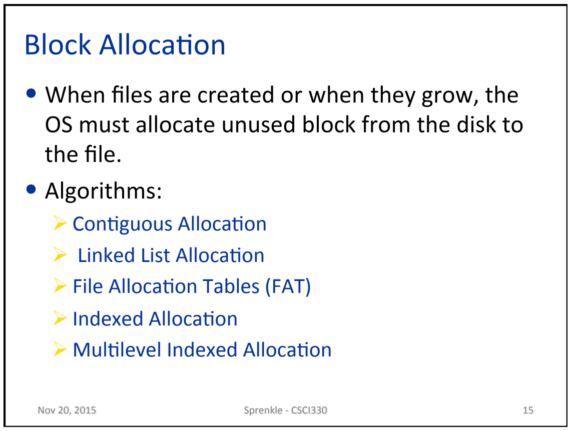


Directory Structures

- Some early OS’s supported only a flat directory structure.
- Hierarchical (tree) directory structure allows for directories to be nested inside of other directories.

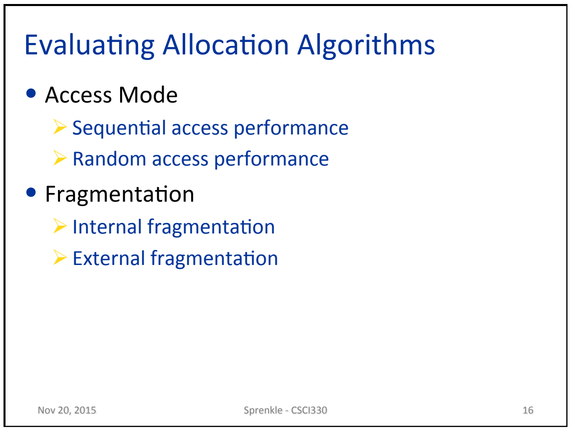


BLOCK ALLOCATION



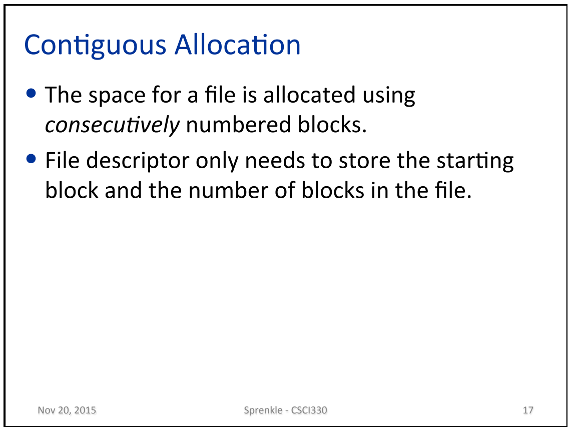
Block Allocation

- When files are created or when they grow, the OS must allocate unused block from the disk to the file.
- Algorithms:
  - ➢ Contiguous Allocation
  - ➢ Linked List Allocation
  - ➢ File Allocation Tables (FAT)
  - ➢ Indexed Allocation
  - ➢ Multilevel Indexed Allocation



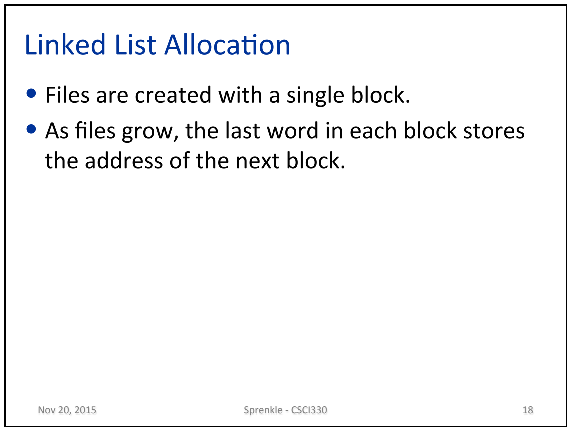
Evaluating Allocation Algorithms

- Access Mode
  - ➢ Sequential access performance
  - ➢ Random access performance
- Fragmentation
  - ➢ Internal fragmentation
  - ➢ External fragmentation



Contiguous Allocation

- The space for a file is allocated using *consecutively* numbered blocks.
- File descriptor only needs to store the starting block and the number of blocks in the file.



Linked List Allocation

- Files are created with a single block.
- As files grow, the last word in each block stores the address of the next block.

File Allocation Tables

- OS maintains a File Allocation Table (FAT) for each disk.
- The FAT has one entry for every disk block.
- File descriptor stores only the starting block of the file.
  - This is also an index into the FAT.
  - Entries in the FAT are used as a linked list to find the remaining blocks of the file.

Indexed Allocation

- With indexed allocation each file descriptor contains a list of the blocks making up the file.
- Multi-level indexed allocation for larger files
- (discussed previously)

Tradeoffs?

FREE SPACE MANAGEMENT

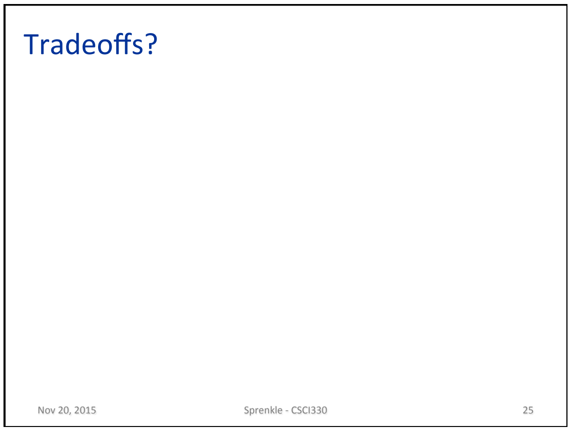
Free Space Management

- The OS must also keep track of which blocks on the disk are not yet allocated to files or used used for directory files (i.e. *free blocks*).
- Several approaches:
  - Bit vector
  - Linked list
  - Indexed

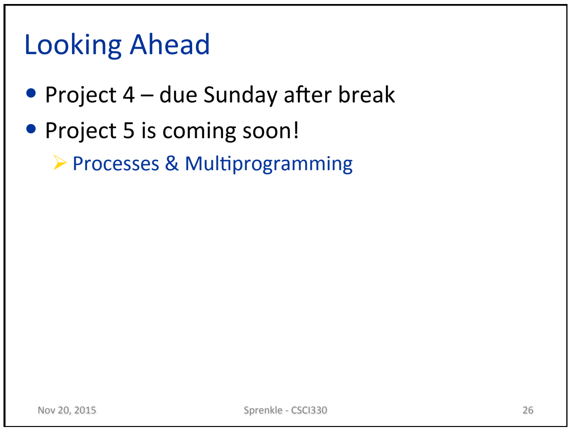
What are these?
What are their tradeoffs?

Approaches to Free Space Management

- Bit Vector:
  - Free disk blocks can be tracked using a bit vector.
    - Each 0 indicates an allocated block.
    - Each 1 indicates a free block.
- Linked List
  - Free space can also be managed using a linked list scheme.
    - Keep track of each free sector
    - Can be modified to be a linked list of *holes*
- Indexed
  - Unix has used inodes 0 and 1 to track the free blocks on a disk.



Tradeoffs?



Looking Ahead

- Project 4 – due Sunday after break
- Project 5 is coming soon!
    - Processes & Multiprogramming