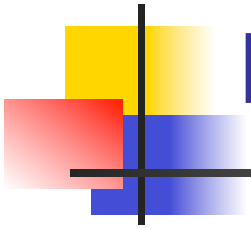
Operating Systems Design (CS 423)

Elsa L Gunter

2112 SC, UIUC

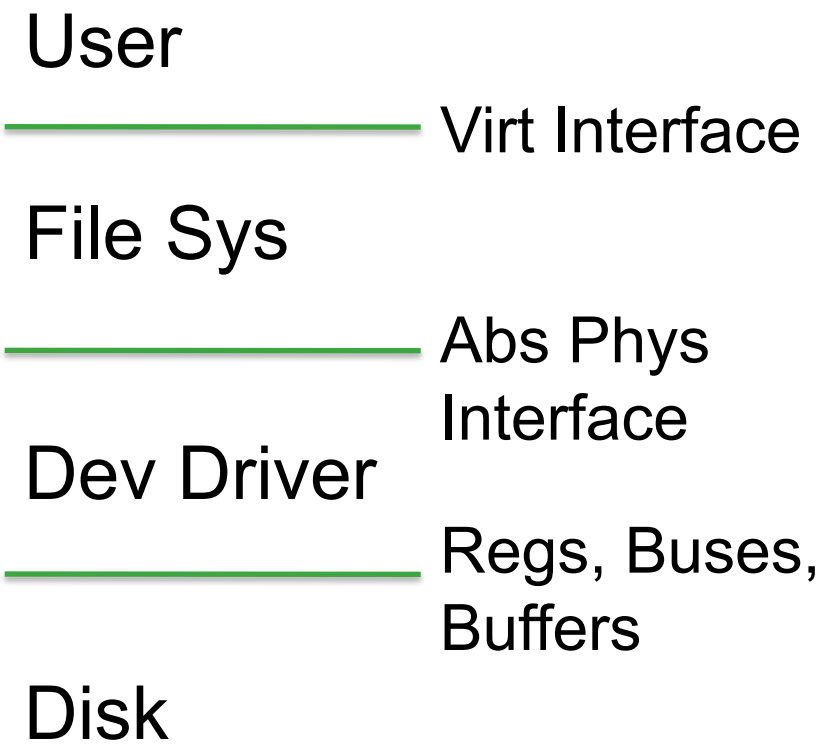
http://www.cs.illinois.edu/class/cs423/

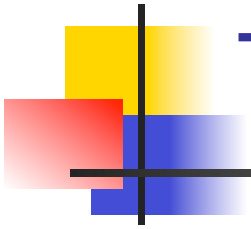
Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum



File Systems

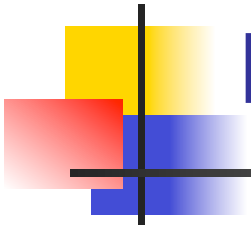
- File system: OS abstraction to make disk easier to use
- Physical reality
  - Slow access to disk blocks
- Illusion provided
  - Fast access to byte oriented files, indexed using symbolic (user-spec) names





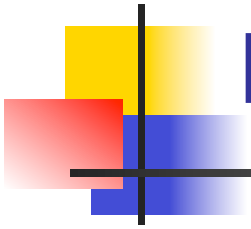
The File Illusion: How to Implement?

- How to map file space onto disk space?
  - File system structure on disk; disk allocation
  - Very similar to memory management
- How to use symbolic names instead of disk sectors?
  - Naming; directories
  - Not similar to memory management where virtual and physical both use same name (i.e.address)



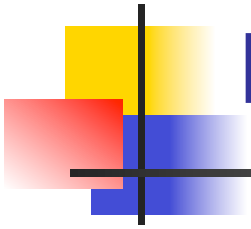
File System Structure

- Overall question: how to organize files on disk
  - What data structure is right one to use?
  - Side note: many things in OS (and CS in general) boil down to data structures and algorithms



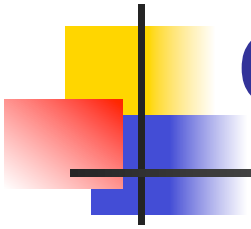
File System Structure

- Need internal structure to describe object
  - Called "file header" in this class
    - Inode in Unix
  - File header also contains miscellaneous information about file, e.g., file size, modification date, permissions
    - Also called file meta-data
- Many ways to organize data on disk



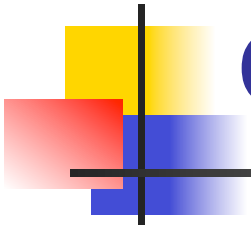
File System Usage Patterns

- 80% of file accesses are reads
- Most programs that access file sequentially access the entire file
  - Alternative is random access
    - Examples?
- Most files are small; most bytes on disk are from large files



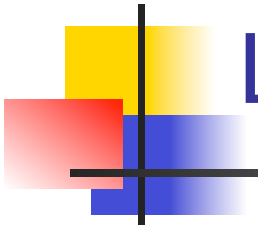
Contiguous Allocation

- Store file in one contiguous segment on disk (sometimes called an extent)
  - User must declare size of file in advance
  - File system will pre-allocate this memory on disk
- What do you do if file grows larger?
- File header is simple: starting block num & size
- Similar to base & bounds for mem mngt



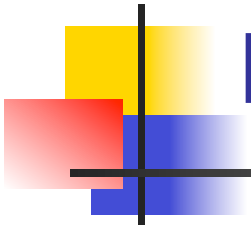
Contiguous Allocation

- Pros
  - No seeks between blocks
  - Easy random access
  - Easy and fast to calculate any block in file
- Cons
  - External fragmentation
  - Hard to grow files
  - Wastes space



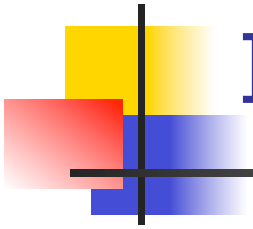
Linked List

- Each block contains pointer to next block of file (along with data)
  - Used by Alto (first personal computer)
- File header contains pointer to first disk block



Linked List

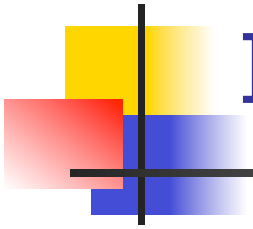
- Pros
  - Grow easily (i.e. append) files
  - No external fragmentation (pick any free block)
- Cons
  - Sequential access quite slow
  - Lots of seeks between blocks
  - Random access is really slow



Indexed Files

- User (or system) declares max # of blocks in file
- System allocates file header with array of pointers big enough to point to that number of blocks
- Extra level of indirection, like a page table

File Block #	Disk Block #
0	18
1	50
2	3
3	22



Indexed Files

```
#define FS_BLOCKSIZE 1024
#define FS_MAXFILEBLOCKS 253
#define FS_MAXUSERNAME 7
typedef struct {
 char owner[FS_MAXUSERNAME + 1];
 int size; // size of the file in bytes
 int blocks[FS_MAXFILEBLOCKS]; // array of file blocks
} fs_inode; (note sizeof(fs_inode) == FS_BLOCKSIZE)
Disk_readblock(int diskBlockNo, void *buf);
Disk_lookupinode(char *fileName, fs_inode *inode);
```

Write code for reading a file block for a given file name

```
Fs_readblock(char *fileName, int fileBlockNo, void *buf)
```