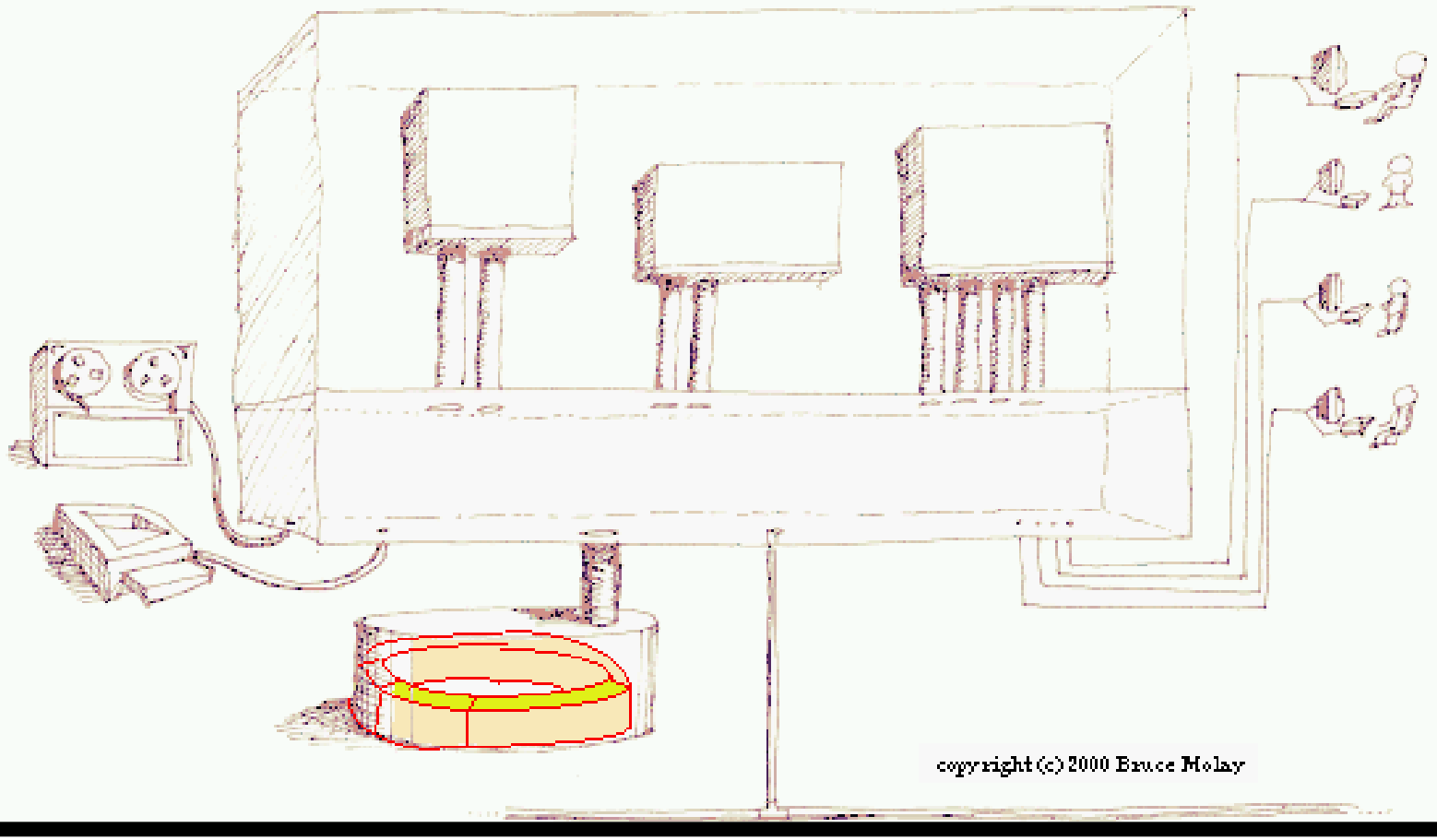
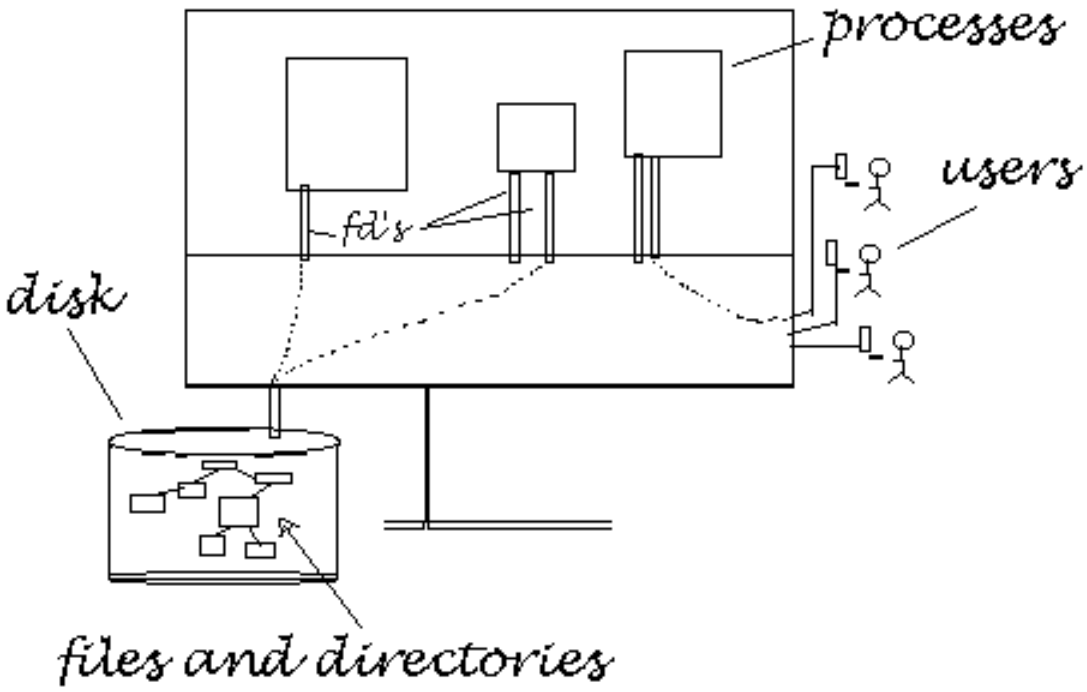
Class 4: **Focus on File Systems**



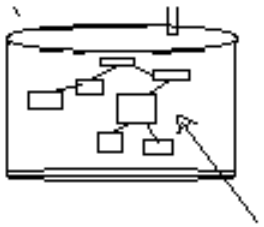
4) Focus on FileSystems

The story so far



files have content AND properties

Tonight: focus on file systems



A disk stores files, file info, and dirs in a tree-structure.

How can a stack of metal platters appear to be a tree of files and dirs?

Q: What is the internal structure of the file system?
Prof: Write `pwd`

2) But first, a brief interruption to discuss error handling

errno and perror(3)

Fact: system calls return -1 on error
But: what *is* the specific error?
e.g.: open() can fail for many reasons

Q: How can a program determine which error it is?
A: The kernel sets a global variable: errno to the error code in /usr/include/errno.h

Q2: How can a program inform the user of the cause of the error?
A2: perror(str) prints the description

Q3: What to do? A: it depends

perror_demo.c

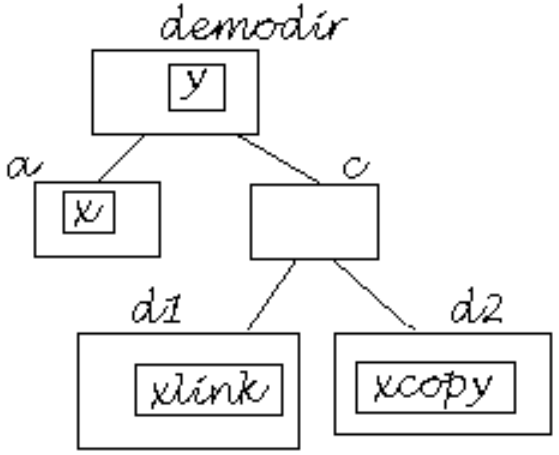
3) User view and Commands for file system

first, let's become familiar with the major fs-related commands by building a demo directory tree

commands

mkdir	cat
rmdir	cp
cd	rm
pwd	mv
	ln

du
ls -la
ls -R



4) Almost no limits to tree structure

- directories can contain lots of files
- directory depths can exceed the capacity of most fs commands

ex. burrow.sh

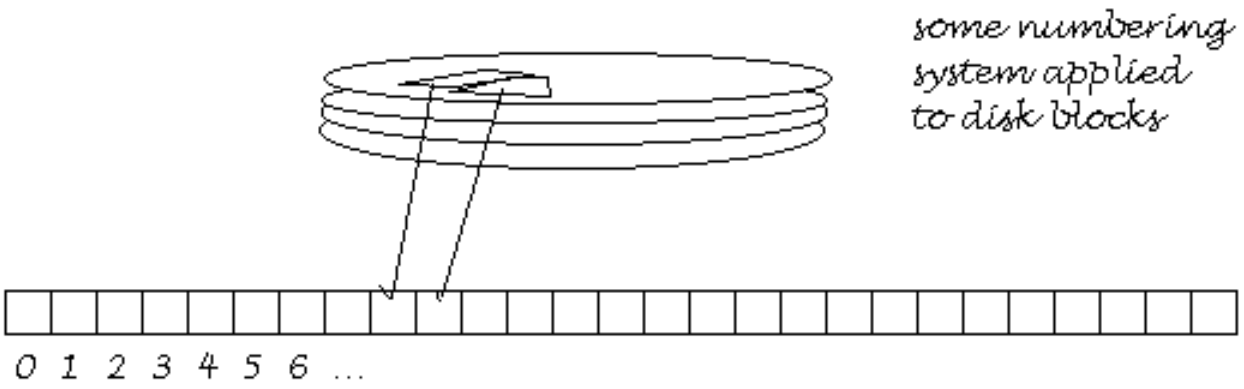
```
while true; do
  mkdir tedwilliams
  cd tedwilliams
done
```

what does du do? how about find, ls -R? How can you fill this hole?

warning: your sysadmin will not be amused by this trick

5) Face Reality: a disk is not a tree
we are not `in' a directory

6) A disk is a stack of magnetic platters, disks, tracks, sectors, viewable as a sequence of DISK BLOCKS

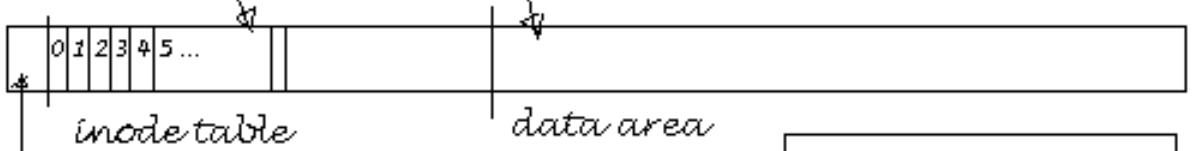


7) How can a numbered sequence of blocks store files, file info, and directories ?

Ans: divide the disk into three regions, structure them sp:

B) file info stored in structs called inodes: Fixed size, several per block. Contains allocation info.

A) file contents are stored in data blocks, as many as are needed



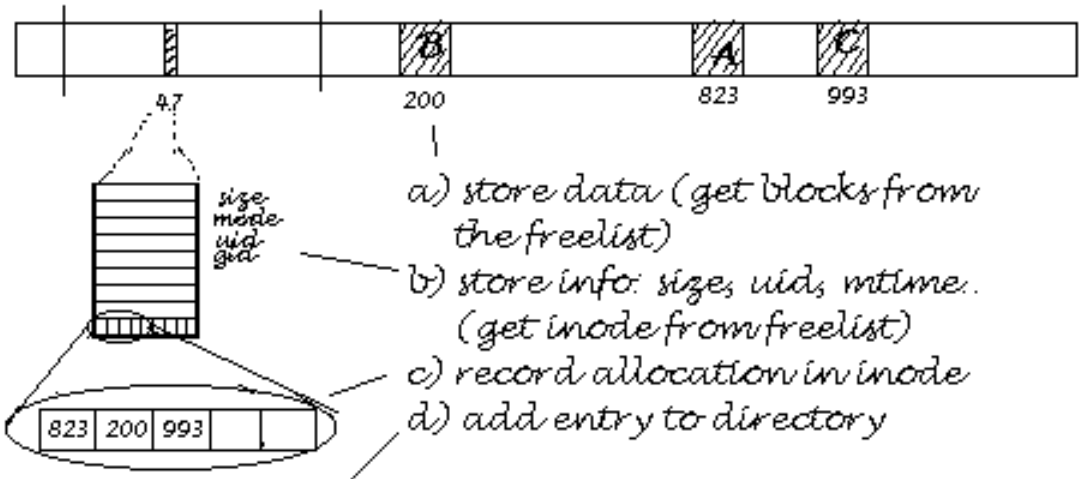
C) info about the disk size and regions is stored in superblock

an inode is identified by its position in the inode table. an inum = 1 file

D) a directory is a table of form: inum filename

8) Using the UFS model to understand creating a file: e.g. who > userlist

say the file requires three diskblocks of storage..



a) store data (get blocks from the freelist)
b) store info: size, uids, mtime.. (get inode from freelist)
c) record allocation in inode
d) add entry to directory

123	.
833	..
4004	hello.c
47	userlist

therefore: in cat userlist

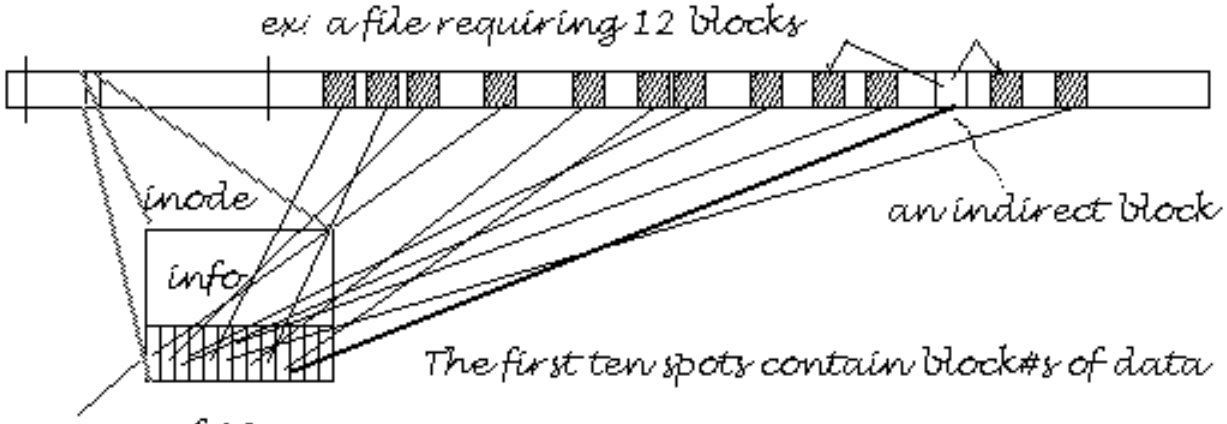
the filename specifies an inum
the inum locates the inode
the inode lists the data blocks

Details of recording disk allocation

fact: a large file is stored in many blocks

fact: the inode holds the info about which blocks contain the data

problem: how can a fixed-sized inode store a *long* block list?



The first ten spots contain block#s of data

block 11 contains the block # of a block of block#s of data

block 12 contains the block # of a block of block#s that contain block#'s of data

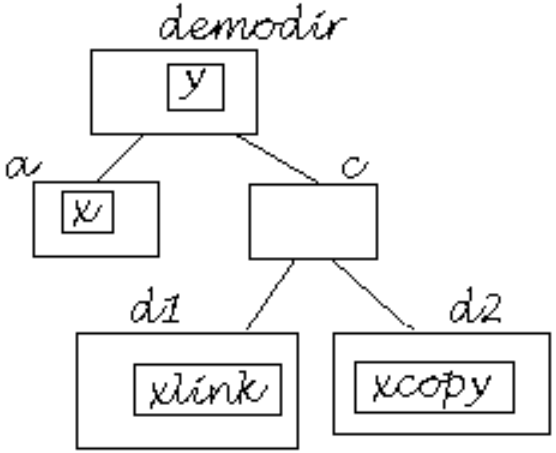
block 13 contains the block # of a block of block#s that contain block#s of block#s that contain data

note: a 'large' file requires more blocks than size/1024

9) Using this model to revisit demodir

now that we know the internal structure of dirs, we can see what is *really* going on in our demo dir structure

user view



system view

	.
	..
	y
	a
	c

	.
	..
	x

	.
	..
	d1
	d2

	.
	..
	xlink

	.
	..
	xcopy

10) The System Calls for standard file ops

command	syscall	action
rm	unlink()	removes a link if links == 0, deallocate
rmdir	rmdir()	delinks a directory
ln	link()	creates a new link
mv	link() then unlink() now rename()	
mkdir	mkdir()	creates new directory

11) writing pwd

pwd prints the path to the current directory. But, the current directory only knows itself as "." How can its location in the tree be determined?