PERSISTENCE: FILE API

Shivaram Venkataraman

CS 537, Spring 2019

ADMINISTRIVIA

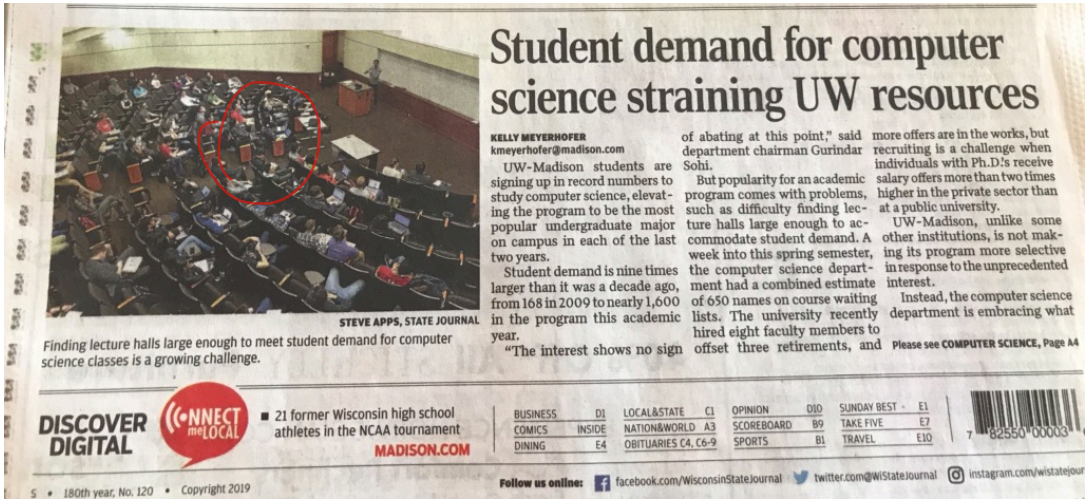
Mid-semester grades: Regrades in-progress

Today is last day to request regrades! Piazza

Project 4a is due tonight!

Project 4b: More details in discussion section

IN OTHER NEWS…



STEVE APPS, STATE JOURNAL

Finding lecture halls large enough to meet student demand for computer science classes is a growing challenge.

Student demand for computer science straining UW resources

KELLY MEYERHOFER
kmeyerhofer@madison.com

UW-Madison students are signing up in record numbers to study computer science, elevating the program to be the most popular undergraduate major on campus in each of the last two years.

Student demand is nine times larger than it was a decade ago, from 168 in 2009 to nearly 1,600 in the program this academic year.

"The interest shows no sign of abating at this point," said department chairman Gurindar Sohi.

But popularity for an academic program comes with problems, such as difficulty finding lecture halls large enough to accommodate student demand. A week into this spring semester, the computer science department had a combined estimate of 650 names on course waiting lists. The university recently hired eight faculty members to offset three retirements, and more offers are in the works, but recruiting is a challenge when individuals with Ph.D.'s receive salary offers more than two times higher in the private sector than at a public university.

UW-Madison, unlike some other institutions, is not making its program more selective in response to the unprecedented interest.

Instead, the computer science department is embracing what

Please see COMPUTER SCIENCE, Page A4

■ 21 former Wisconsin high school athletes in the NCAA tournament

S • 180th year, No. 120 •

Follow us online: facebook.com/WisconsinStateJournal twitter.com/WiStateJournal instagram.com/wistatejour

https://madison.com/news/local/education/university/student-demand-for-computer-science-straining-uw-madison-department-resources/article_344acba7-e337-5c3c-81a7-4376172f0d74.html

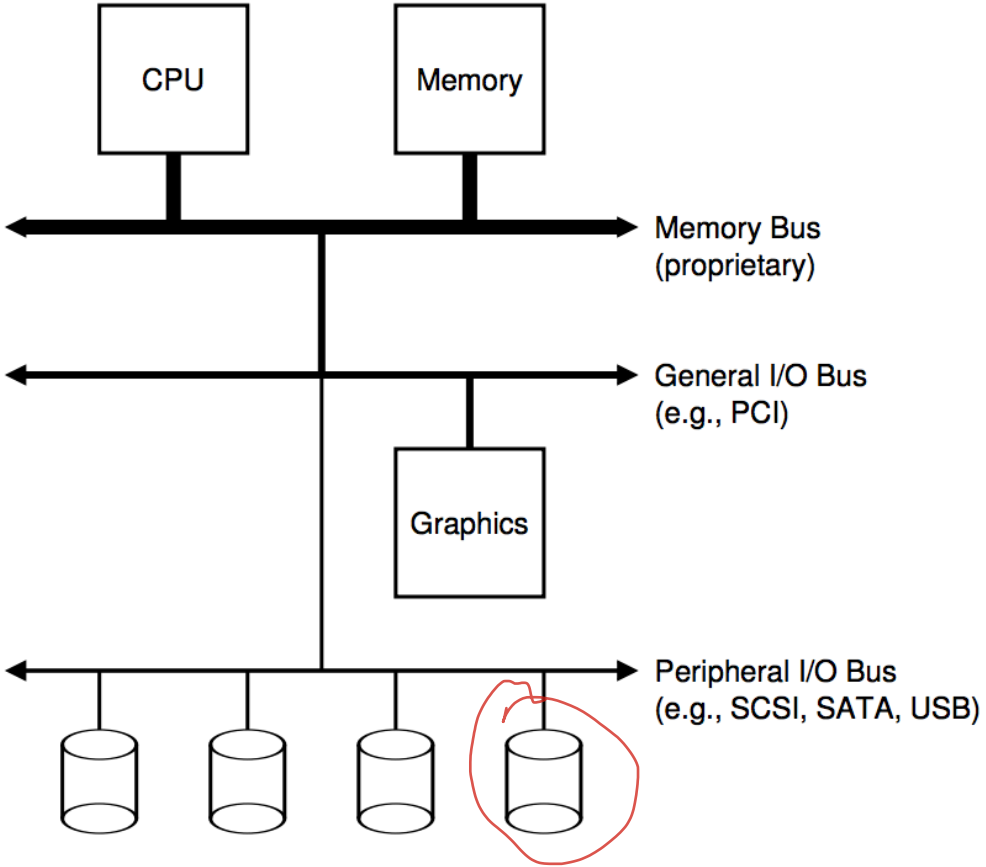
AGENDA / LEARNING OUTCOMES

How to name and organize data on a disk?

What is the API programs use to communicate with OS?

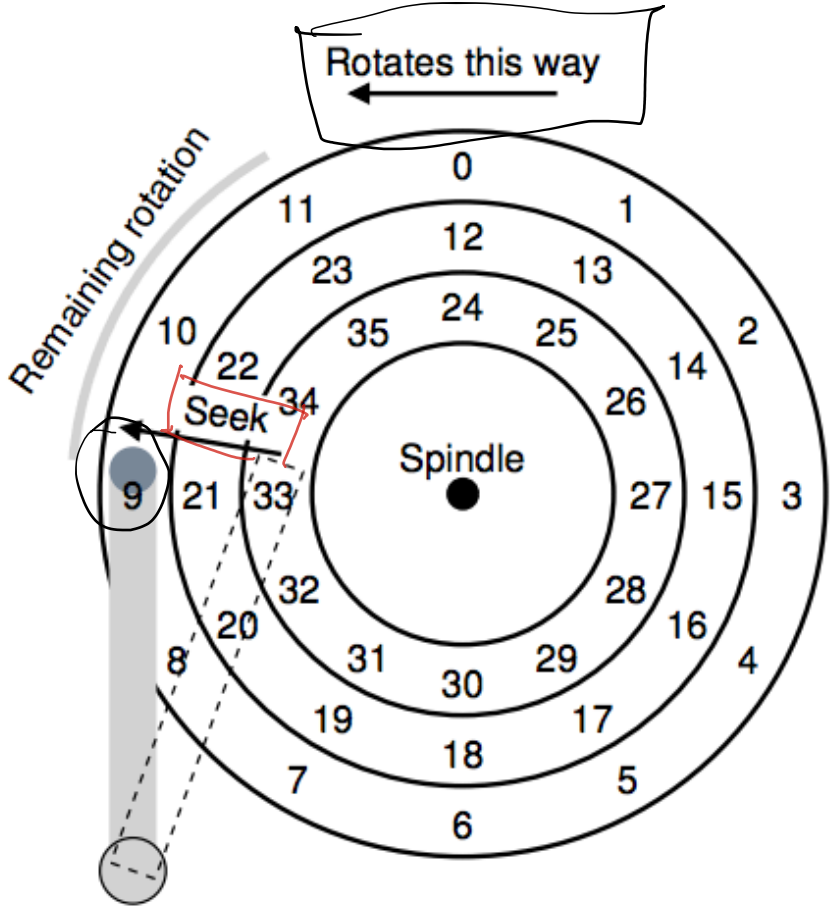
RECAP

HARDWARE SUPPORT FOR I/O



Storage devices → Hard drives disk

READING DATA FROM DISK



Seek Time

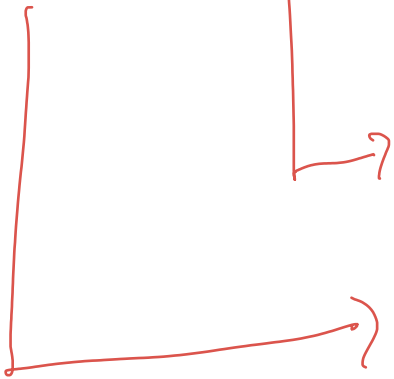
Rotational delay

RAID: GOALS

Capacity → 1TB of data ↓ 10TB? → Redundant Array of Inexpensive Disks

Fault tolerance → Disk failure

Performance → Disks are slow!



RAID LEVELS

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

RAID 0: Striping

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID 1: Mirroring
or RAID-10 or (RAID 1+0)
mirrored pairs and then stripes

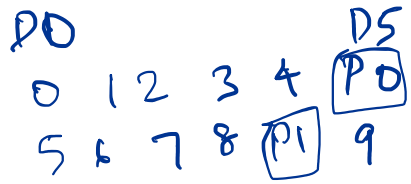
Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

RAID 4: Parity!

COMPARISON

	RAID-0	RAID-1	RAID-4
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1
Throughput			
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$
Latency			
Read	T	T	T
Write	T	T	$2T$

R - random b/w of 1 disk



RAID-5: ROTATED PARITY

Capacity: (N-1) * B
Reliability: one
Latency
Read T, Write 2T

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

sequential reads? (N-1) * S
sequential writes? (N-1) * S
random reads? (N) * R
random writes? N/4 * R

Read old block, old parity
Write new block, new parity

Block 0 : Disk 0, Disk 4
Block 7 : Disk 2, Disk 3

COMPARISON

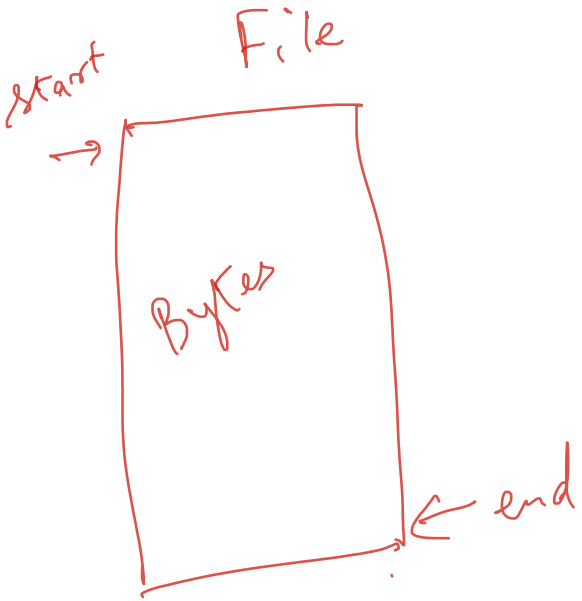
	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

Abstraction - Files
Directories
Filesystem

Implementation -

FILES

WHAT IS A FILE?



Array of persistent bytes that can be read/written

File system consists of many files

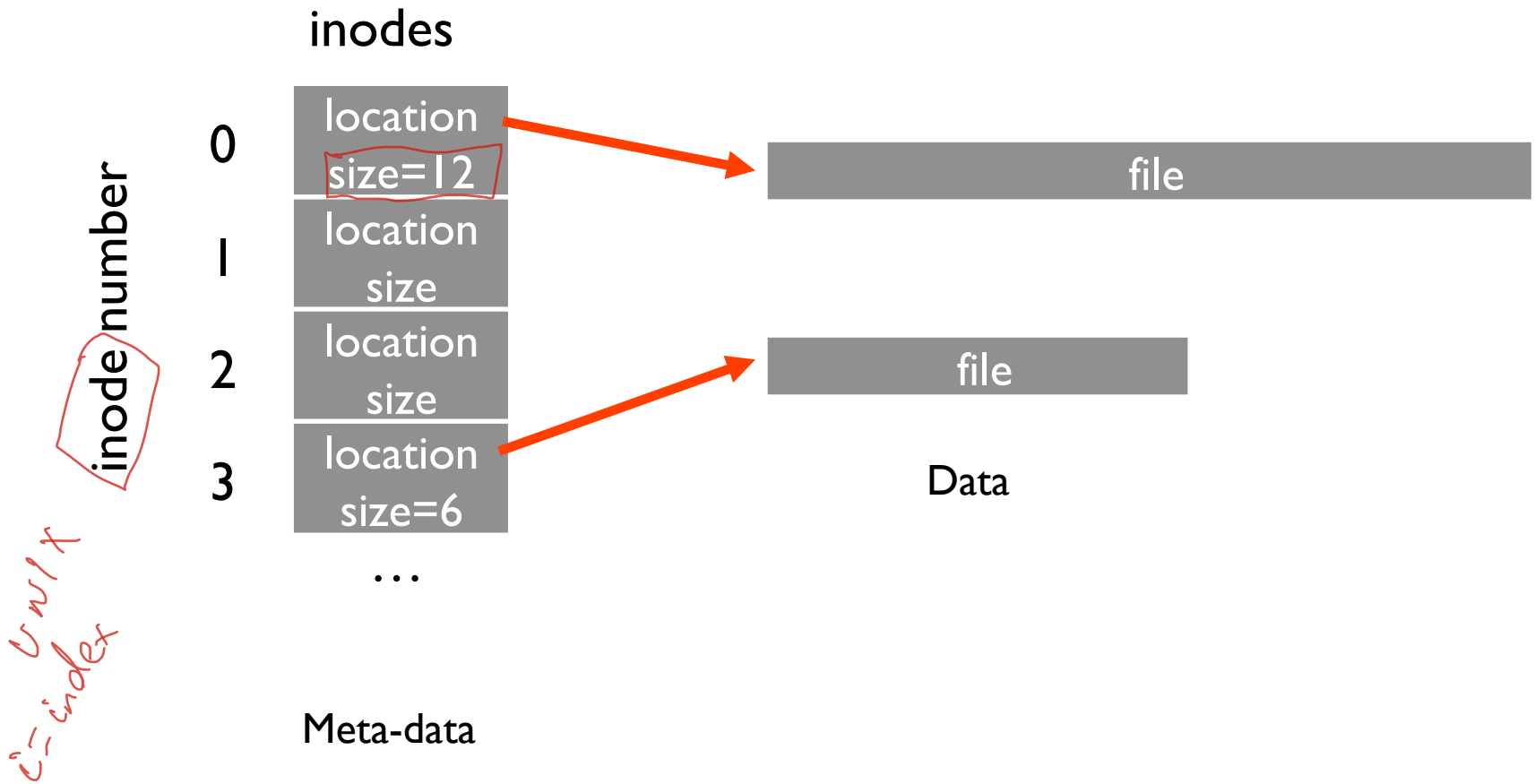
- Refers to collection of files
- Also refers to part of OS that manages those files

Files need names to access correct one

FILE NAMES

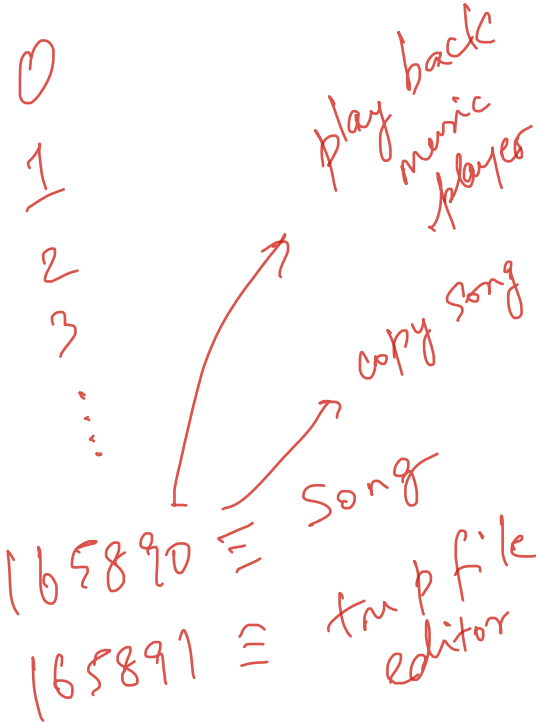
Three types of names

- Unique id: inode numbers
- Path
- File descriptor


inodes
inode number
0
1
2
3
location
size=12
location
size
location
size
location
size=6
...
file
file
Data
Meta-data
i = index
unix

FILE API (ATTEMPT 1)

```
read(int inode, void *buf, size_t nbyte)
write(int inode, void *buf, size_t nbyte)
seek(int inode, off_t offset)
```



Disadvantages?

- names hard to remember
- no organization or meaning to inode numbers
- semantics of offset across multiple processes?

Isolation

PATHS

String names are friendlier than number names → "Cool Song.mp3" is 168590

File system still interacts with inode numbers

Store *path-to-inode* mappings in a special file or rather a Directory!

Directory

(Name, Inode)

Song1.mp3

Image.jpg

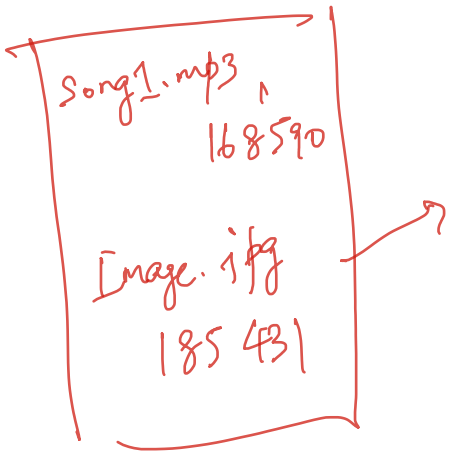
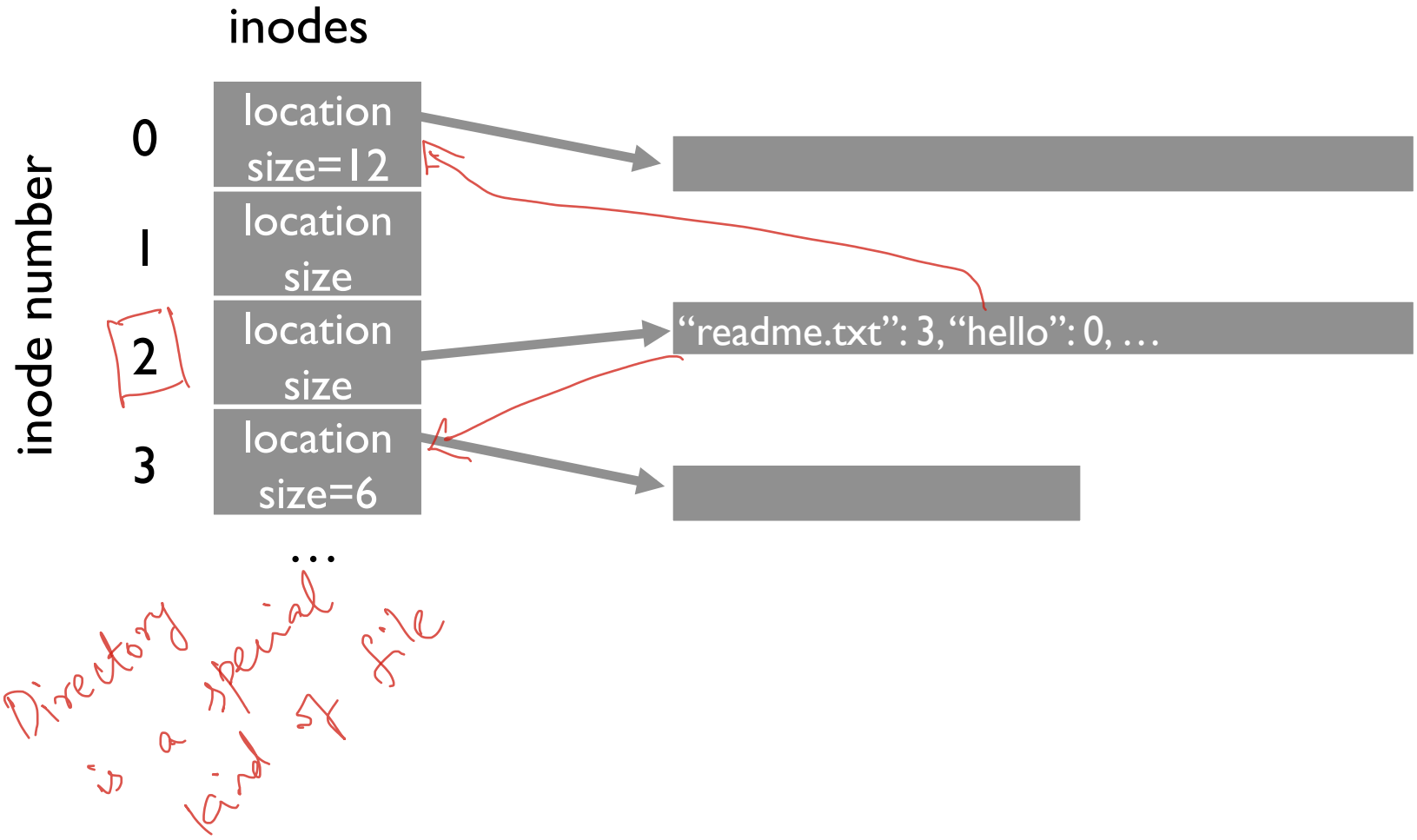


Image.jpg


inodes
inode number
0
1
2
3
location
size=12
location
size
location
size
location
size=6
...
"readme.txt": 3, "hello": 0, …
Directory is a special kind of file

PATHS

/ → root of the tree

/ → separator across dirs

Directory Tree instead of single root directory

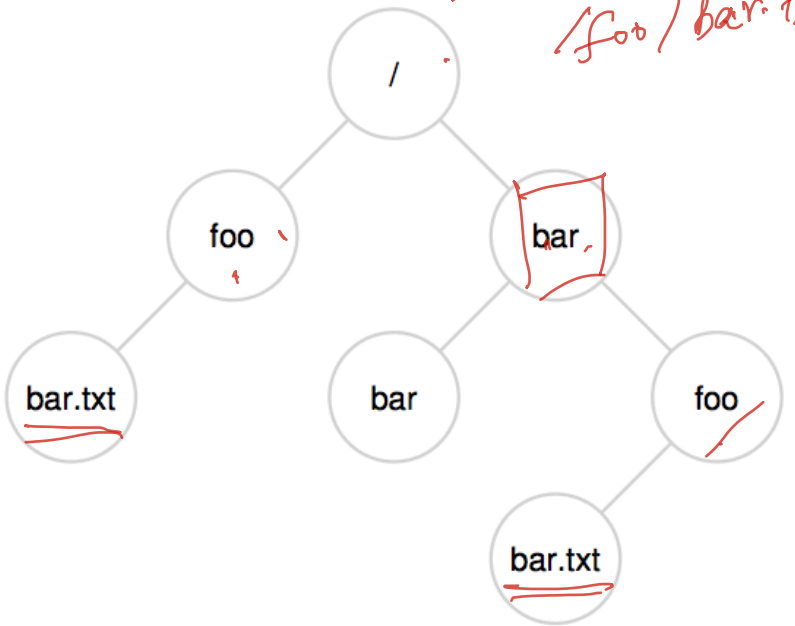
File name needs to be unique within a directory

/usr/lib/file.so

/tmp/file.so

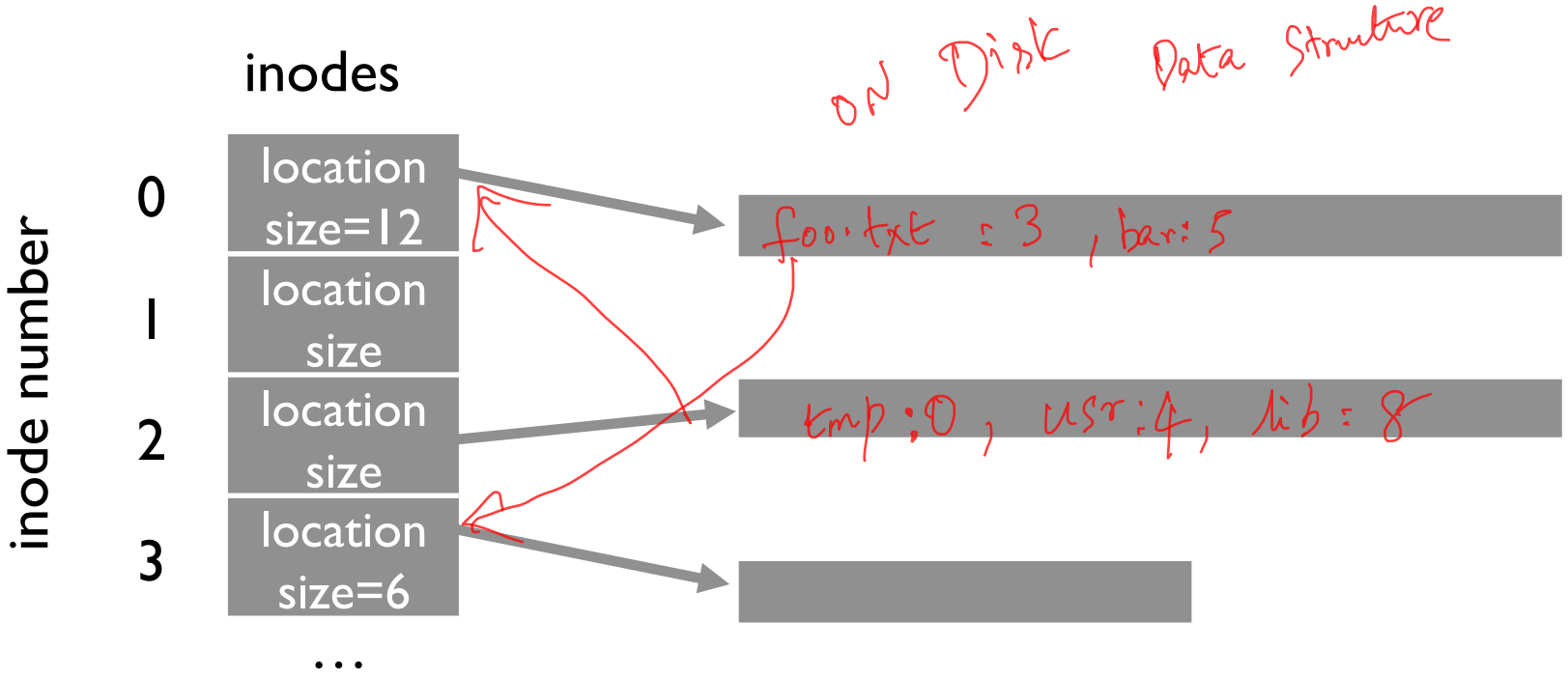
Store file-to-inode mapping in each directory

Sub directory inode is stored in parent directory



UNIX

/foo/bar.txt



Reads for getting final inode called "traversal"

Example: read /tmp/foo.txt

"/tmp/bar/foo.txt"

FILE API (ATTEMPT 2)

```
read(char *path, void *buf, off_t offset, size_t nbyte)
write(char *path, void *buf, off_t offset, size_t nbyte)
```

Disadvantages?

- Expensive traversal!
- Goal: traverse once

every read or write will incur traversal

FILE NAMES

Three types of names:

- inode
- path
- file descriptor

FILE DESCRIPTOR (FD)

Idea:

Do expensive traversal once (open file)
Store inode in descriptor object (kept in memory).
Do reads/writes via descriptor, which tracks offset

Each process:

File-descriptor table contains pointers to open file descriptors

Integers used for file I/O are indexes into this table

stdin: 0, stdout: 1, stderr: 2

FILE API (ATTEMPT 3)

```
int fd = open(char *path, int flag, mode_t mode)
read(int fd, void *buf, size_t nbyte)
write(int fd, void *buf, size_t nbyte)
close(int fd)
```

advantages:

- string names
- hierarchical
- traverse once
- offsets precisely defined

FD TABLE (XV6)

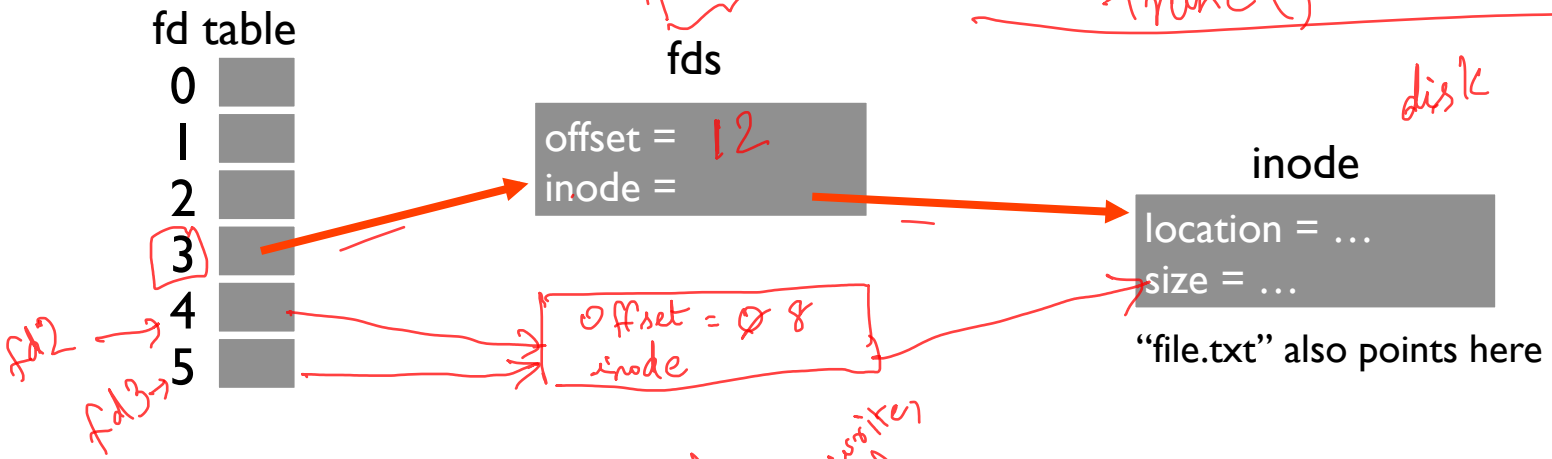
```
struct file {
  ...
  struct inode *ip;
  uint off;
};

// Per-process state
struct proc {
  ...
  struct file *ofile[NOFILE];  // Open files
  ...
}
```

tracking where in the file you are

```
struct {
    struct spinlock lock;
    struct file file[NFILE];
} ftable;
```

global in the OS

Maximum number of files



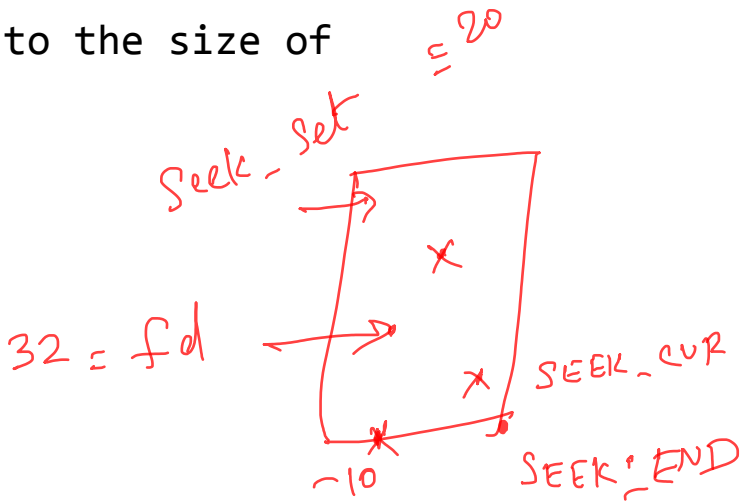

```
int fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
int fd2 = open("file.txt"); // returns 4
int fd3 = dup(fd2);         // returns 5
```

READ NOT SEQUENTIALLY

```
off_t lseek(int fildes, off_t offset, int whence)
```

If whence is SEEK_SET, the offset is set to offset bytes.
If whence is SEEK_CUR, the offset is set to its current location plus offset bytes.
If whence is SEEK_END, the offset is set to the size of the file plus offset bytes.

Seek does not generate I/O operations



BUNNY 13

https://tinyurl.com/cs537-sp19-bunny13

BUNNY 13

https://tinyurl.com/cs537-sp19-bunny13

```
int fd1 = open("file.txt"); // returns 12
int fd2 = open("file.txt"); // returns 13
read(fd1, buf, 16);
int fd3 = dup(fd2);         // returns 14
read(fd2, buf, 16);
lseek(fd1, 100, SEEK_SET);
```

What are the value of offsets in fd1, fd2, fd3 after the above code sequence?

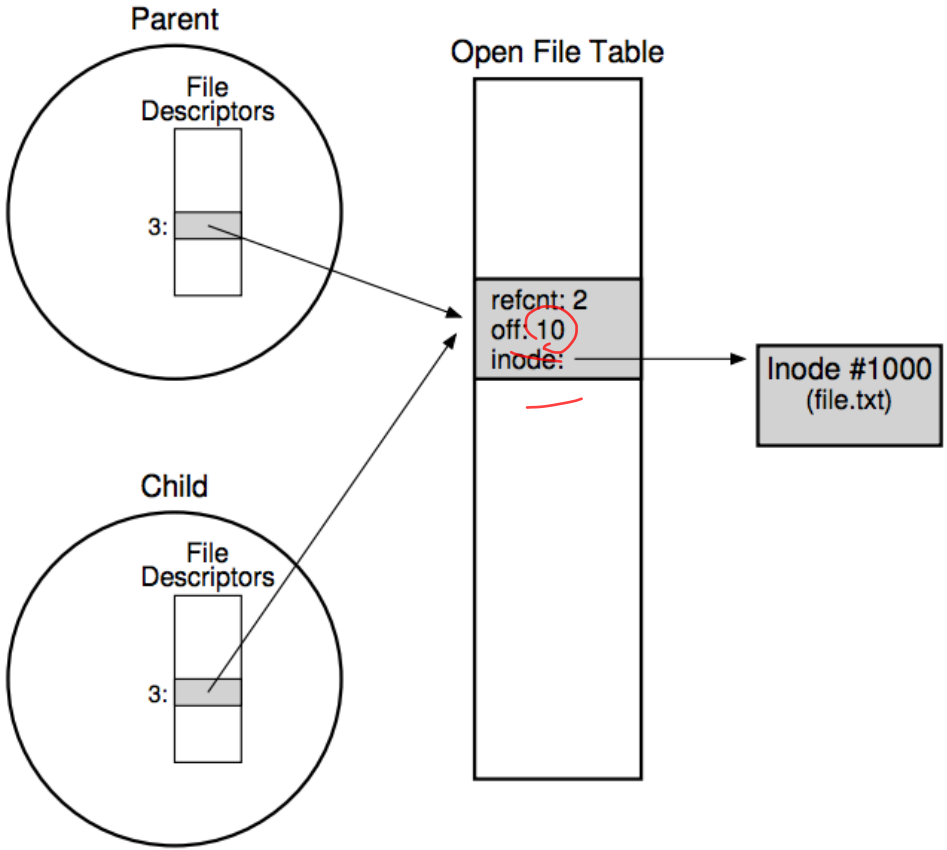
FD1: 100

FD2: 16

FD3: 16

WHAT HAPPENS ON FORK?

set of open file* from parent to child process



child
lseek(3, 100, SEEK_CUR)

parent
= 10 + 100
= 110

DELETING FILES

"rm" → not a syscall

There is no system call for deleting files!

Inode (and associated file) is **garbage collected** when there are no references

Paths are deleted when: `unlink()` is called

/tmp /foo.txt

FDs are deleted when: `close()` or process quits

unlink /tmp/foo.txt

Go to /tmp remove foo.txt

COMMUNICATING REQUIREMENTS: FSYNC

File system keeps newly written data in memory for awhile

Write buffering improves performance (why?)

But what if system crashes before buffers are flushed?

fsync(int fd) forces buffers to flush to disk, tells disk to flush its write cache

Makes data durable

RENAME

rename(char *old, char *new):

- deletes an old link to a file
- creates a new link to a file

Just changes name of file, does not move data

Even when renaming to new directory

What can go wrong if system crashes at wrong time?

ATOMIC FILE UPDATE

Say application wants to update file.txt atomically

If crash, should see only old contents or only new contents

1. write new data to file.txt.tmp file
2. fsync file.txt.tmp
3. rename file.txt.tmp over file.txt, replacing it

DIRECTORY FUNCTIONS, LINKS

DIRECTORY CALLS

mkdir: create new directory

readdir: read/parse directory entries

Why no writedir?

SPECIAL DIRECTORY ENTRIES

```
➜  xv6-sp19 ls -la .
total 5547
drwxrwxr-x  7 shivaram shivaram    2048 Mar 10 22:59 .
drwxr-xr-x 47 shivaram shivaram    6144 Apr  4 11:27 ..
-rwxrwxr-x  1 shivaram shivaram     106 Mar  6 15:23 bootother
-rw-r-----  1 shivaram shivaram     223 Feb 28 17:37 FILES
drwxrwxr-x  2 shivaram shivaram    2048 Mar  6 15:23 fs
-rw-rw-r--  1 shivaram shivaram  524288 Mar  6 15:23 fs.img
drwxr-x---  2 shivaram shivaram    2048 Mar 13 13:34 include
-rwxrwxr-x  1 shivaram shivaram      44 Mar  6 15:23 initcode
drwxr-x---  2 shivaram shivaram    6144 Apr  3 22:22 kernel
-rw-------  1 shivaram shivaram    4816 Feb 28 17:37 Makefile
-rw-r-----  1 shivaram shivaram    1793 Feb 28 17:37 README
drwxr-x---  2 shivaram shivaram    2048 Mar  6 15:23 tools
drwxr-x---  3 shivaram shivaram    4096 Apr  4 11:26 user
-rw-r-----  1 shivaram shivaram      22 Feb 28 17:37 version
-rw-rw-r--  1 shivaram shivaram 5120000 Mar  6 15:28 xv6.img
```

LINKS

Hard links: Both path names use same inode number

File does not disappear until all removed; cannot link directories

```
echo “Beginning…” > file1
ln file1 link
cat link
ls –li
echo “More info” >> file1
mv file1 file2
rm file2
```

SOFT LINKS

Soft or symbolic links: Point to second path name; can softlink to dirs

```
ln -s oldfile softlink
```

Confusing behavior: “file does not exist”!

Confusing behavior: “cd linked_dir; cd ..; in different parent!

PERMISSIONS, ACCESS CONTROL

```
➜  xv6-sp19 ls -la .
total 5547
drwxrwxr-x  7 shivaram shivaram    2048 Mar 10 22:59 .
drwxr-xr-x 47 shivaram shivaram    6144 Apr  4 11:27 ..
-rwxrwxr-x  1 shivaram shivaram     106 Mar  6 15:23 bootother
-rw-r-----  1 shivaram shivaram     223 Feb 28 17:37 FILES
drwxrwxr-x  2 shivaram shivaram    2048 Mar  6 15:23 fs
-rw-rw-r--  1 shivaram shivaram  524288 Mar  6 15:23 fs.img
```

```
➜  xv6-sp19 fs la .
Access list for . is
Normal rights:
  system:administrators rlidwka
  system:anyuser l
  shivaram rlidwka
```

MANY FILE SYSTEMS

Users often want to use many file systems

For example:

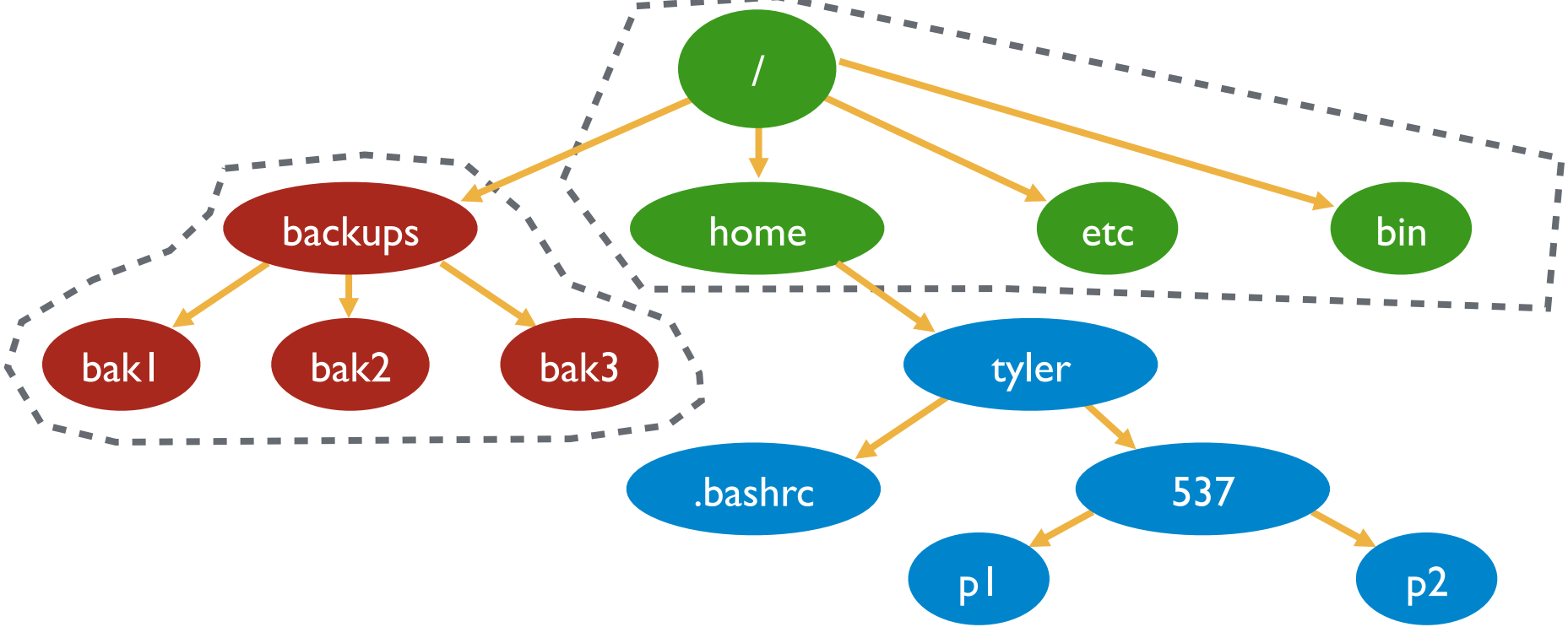
- main disk
- backup disk
- AFS
- thumb drives

What is the most elegant way to support this?

MANY FILE SYSTEMS

Idea: stitch all the file systems together into a super file system!

```
sh> mount
/dev/sda1 on / type ext4 (rw)
/dev/sdb1 on /backups type ext4 (rw)
AFS on /home type afs (rw)
```



/dev/sda1 **on** /

/dev/sdb1 **on** /backups

AFS **on** /home

BUNNY 14

https://tinyurl.com/cs537-sp19-bunny14

BUNNY 14

Consider the following code snippet:

https://tinyurl.com/cs537-sp19-bunny14

```
echo “hello” > oldfile
ln –s oldfile link1
ln oldfile link2
rm oldfile
```

What will be the output of `cat link1`

What will be the output of `cat link2`

What is the file permission to only give current user read, write, execute access?

SUMMARY

Using multiple types of name provides convenience and efficiency

Mount and link features provide flexibility.

Special calls (fsync, rename) let developers communicate requirements to file system

NEXT STEPS

Next class: How to implement file systems

Project 4b in Discussion today