CS 423
Operating System Design: Persistence: FS Implementation
04/23

Ram Alagappan

RECAP

RAID LEVEL COMPARISONS

	Reliability	Capacity	Read latency	Write Latency	Seq Read	Seq Write	Rand Read	Rand Write
RAID-0	0	C*N	D	D	N * S	N * S	N * R	N * R
RAID-1	1	C*N/2	D	D	N/2 * S	N/2 * S	N * R	N/2 * R
RAID-4	1	(N-1) * C	D	2D	(N-1)*S	(N-1)*S	(N-1)*R	R/2
RAID-5	1	(N-1) * C	D	2D	(N-1)*S	(N-1)*S	N * R	N/4 * R

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

Three types of names

- Unique id: inode numbers
- Path
- File descriptor

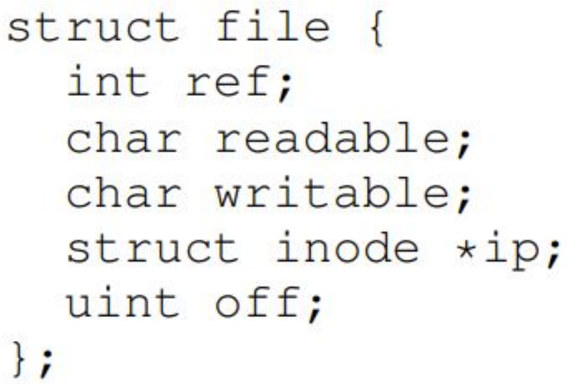
File API (attempt 3)

```
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)
```

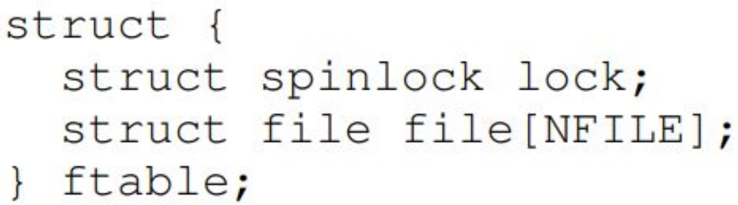
advantages:

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

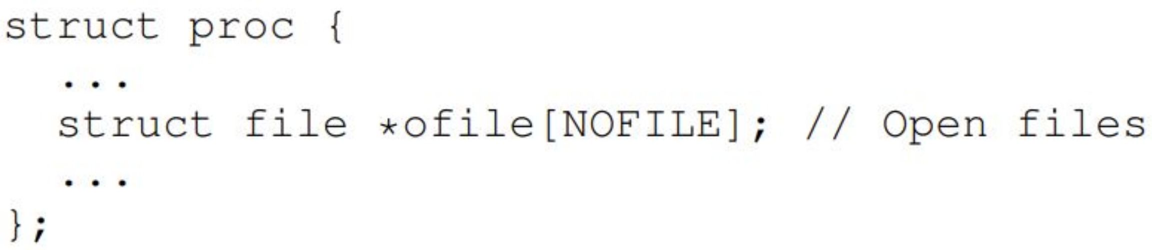
FD Table (xv6)



```
struct file {
  int ref;
  char readable;
  char writable;
  struct inode *ip;
  uint off;
};
```



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



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

LSEEK and READ

```
off_t lseek(int filedesc, 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.

Assume head is on track 1

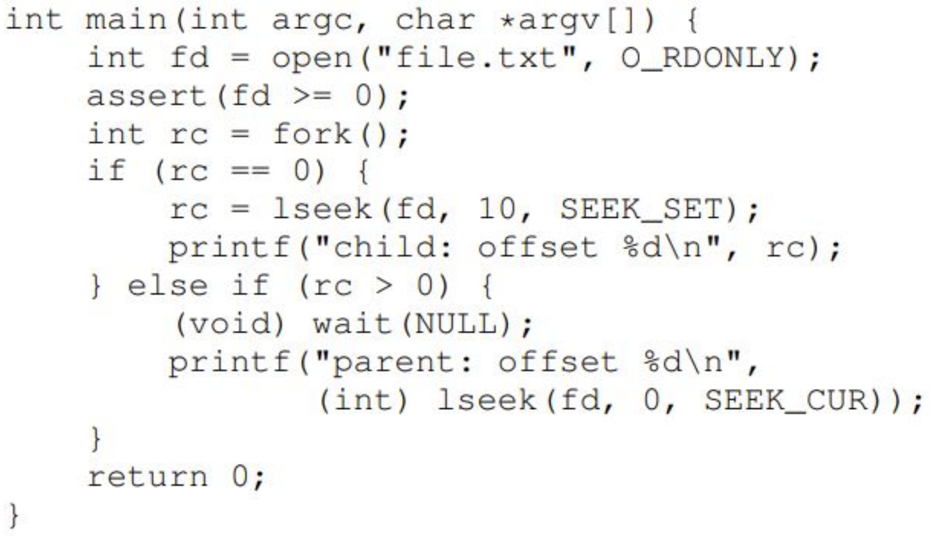
Suppose we do lseek to X and the sector for X is on track 4

Where is head immediately after lseek?

END RECAP

Shared Entries in OFT

Fork:



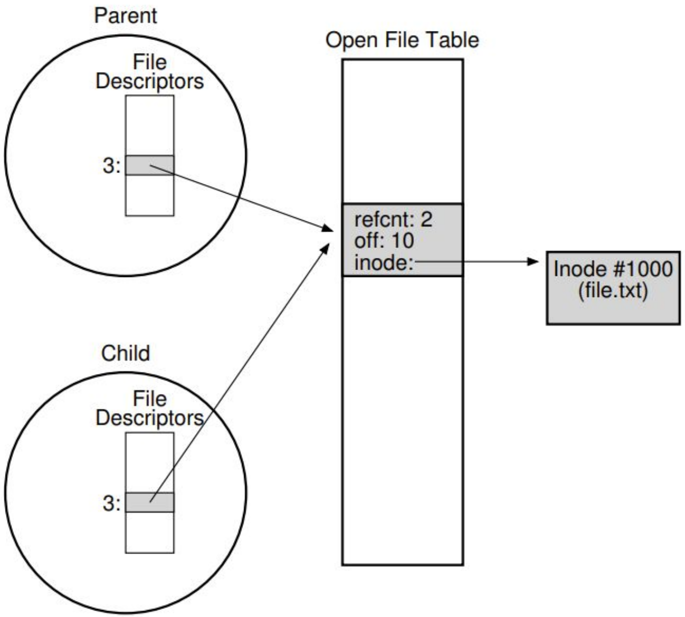
```
int main(int argc, char *argv[]) {
    int fd = open("file.txt", O_RDONLY);
    assert(fd >= 0);
    int rc = fork();
    if (rc == 0) {
        rc = lseek(fd, 10, SEEK_SET);
        printf("child: offset %d\n", rc);
    } else if (rc > 0) {
        (void) wait(NULL);
        printf("parent: offset %d\n",
               (int) lseek(fd, 0, SEEK_CUR));
    }
    return 0;
}
```

What is the parent trying to print?

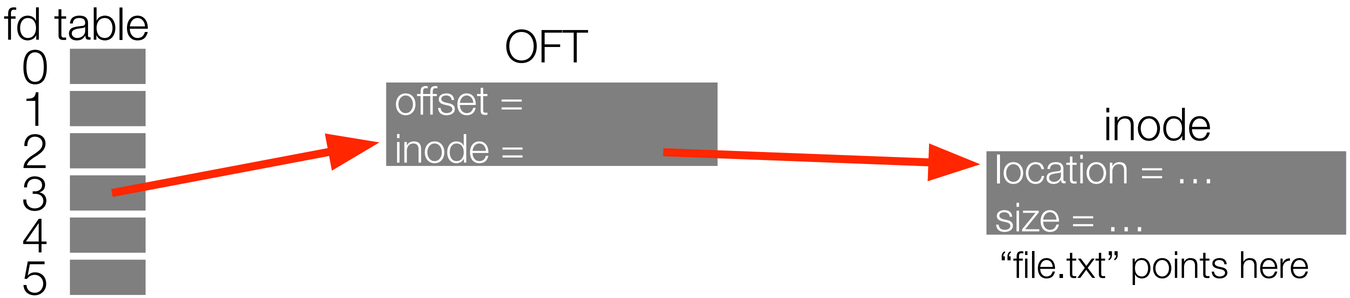
What value will be printed?

Shared Entries in OFT

What's happening here?

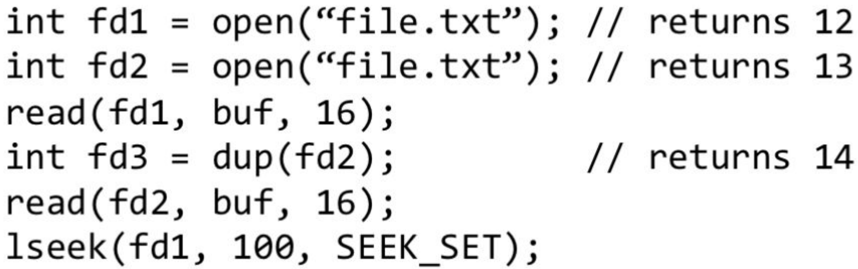


DUP



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

DUP



```
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);
```

How many entries in the OFT (assume no other process)?

Offset for fd1?

Offset for fd2?

Offset for fd3

Fsync

File system keeps newly written data in memory for a while

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):

Do we need to copy/move data?

How does the FS implement this?

Does it matter whether the old and new names are in the same directory or different directories?

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?

Paths and Links

(Hard) Link

Inode has a field called "nlinks"

When is it incremented?

When is it decremented?

Deleting Files

What is the system call for deleting files?

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

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

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

Symbolic or soft links

A different type of link

Hard links don’t work with directory and cannot be cross-FS

touch foo; echo hello > foo;

Hardlink: ln foo foo2

Stat foo; what will be the size and inode?

Stat foo2; what will be the size and inode?

Softlink: ln –s foo bar

Stat bar; what will be the size and inode?

Using vs. Implementing

So far, focus on interface of FS

how apps view FS

Today, more about how to implement the FS itself

Then, crash consistency

FILE SYSTEM IMPLEMENTATION

VSFS

Very Simple File System

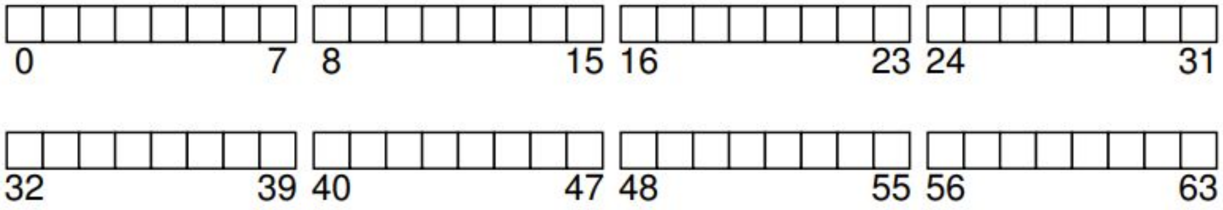
Two aspects:

Data structures – how are files, directories, etc stored on disk

Access methods – how are high-level operations like open, read, write mapped to these DS operations

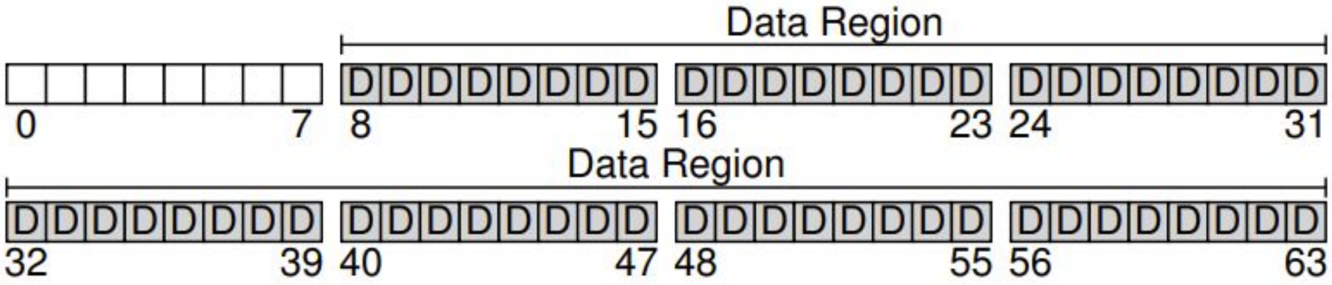
VSFS

Assume a small disk partition with 64 blocks

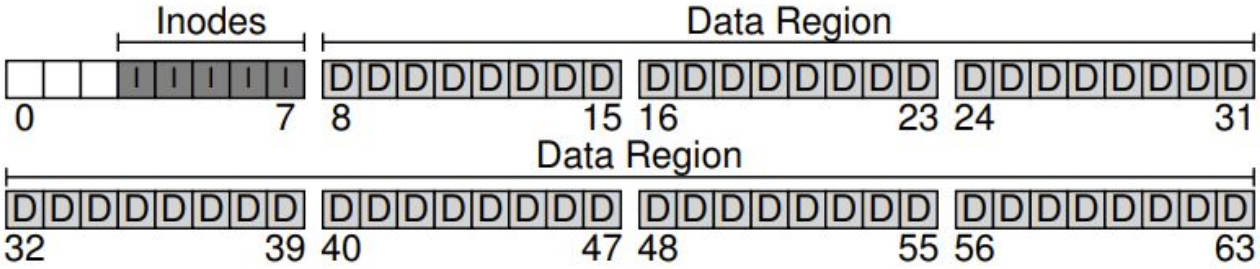


Data and metadata – most space must go for data blocks

VSFS – Data blocks



VSFS – Inodes (metadata)



Called the inode table

With 256-byte inodes, we can store 16 inodes in a block, so totally 80 files can be stored in VSFS

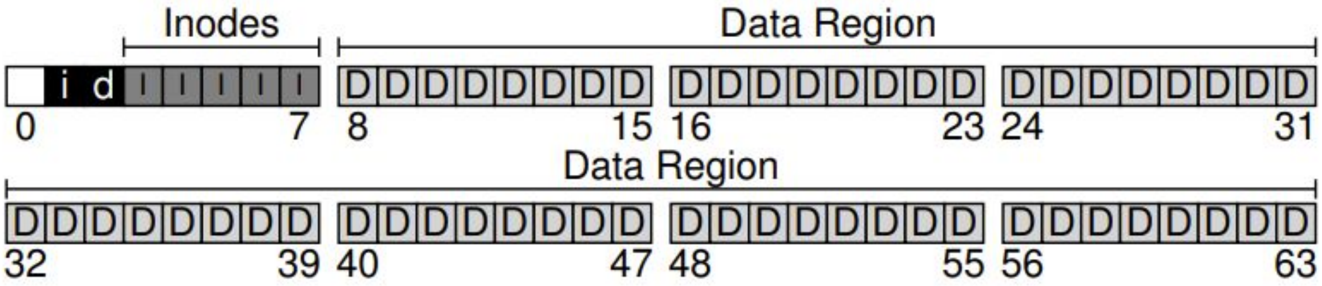
But we can simply scale VSFS to a larger disk

VSFS – Bitmaps (metadata)

Need allocation structures

Free lists – linked list is an option

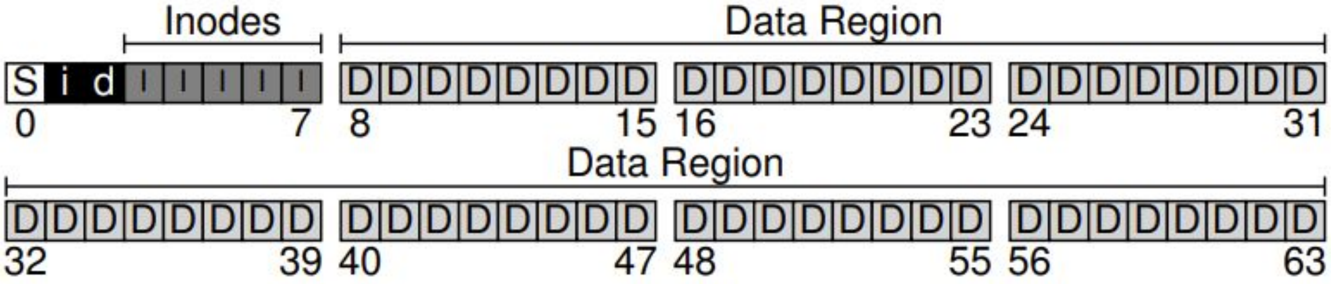
Most commonly used: bitmap (ib, db)



VSFS – (metadata)

What's stored in the first block?

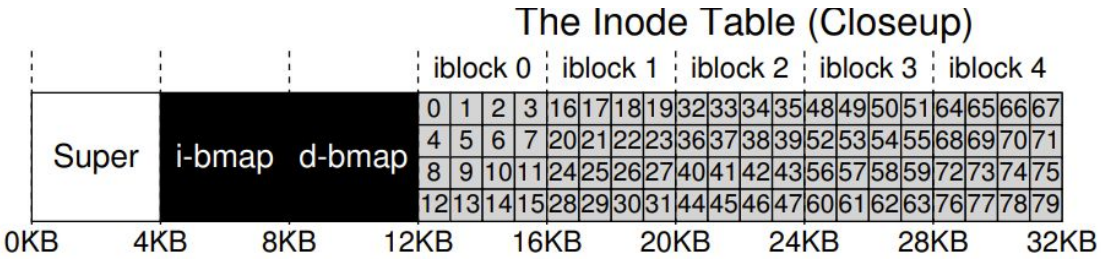
VSFS – Superblock (metadata)



How many data blocks, how many inode blocks

Inode table starting block #

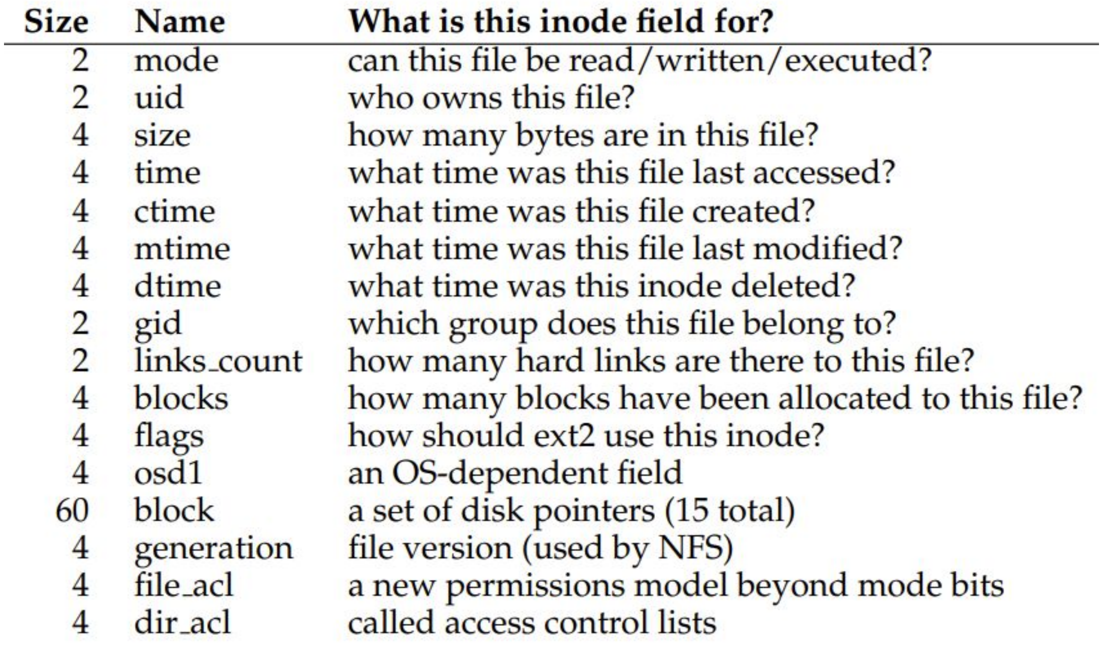
INODE



0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51	64	65	66	67
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55	68	69	70	71
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59	72	73	74	75
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63	76	77	78	79

Implicitly know the block/sector number

INODE



Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

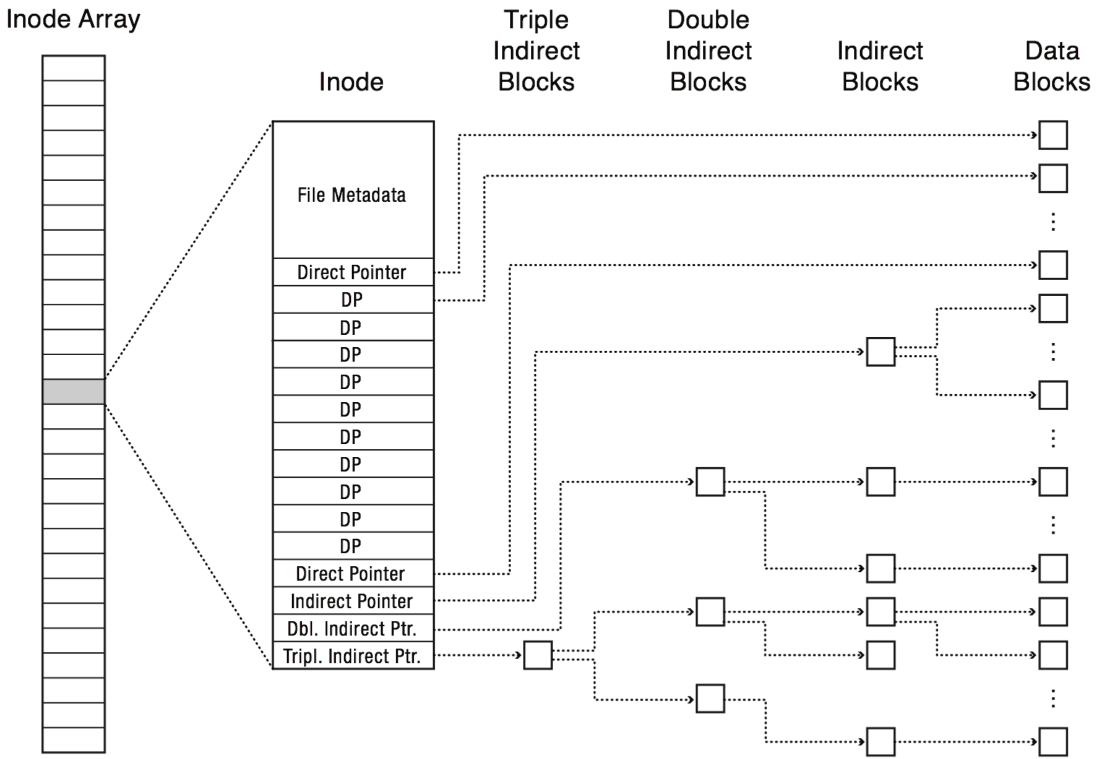
What is the max file size?

We have 15 block pointers

What is the max file size?

How can we support larger files?

Direct and Indirect Pointers



File Size

File size with one indirect pointer + 12 direct:

1024 * 4K + 12*4K – roughly 4MB

File size with 1 double ID + 1 ID + 12 direct:

1024 * 1024 * 4K + 1024 * 4K + 12*4K – roughly 4GB

Extent based approach

No pointer for every block

<Starting block, num blocks>

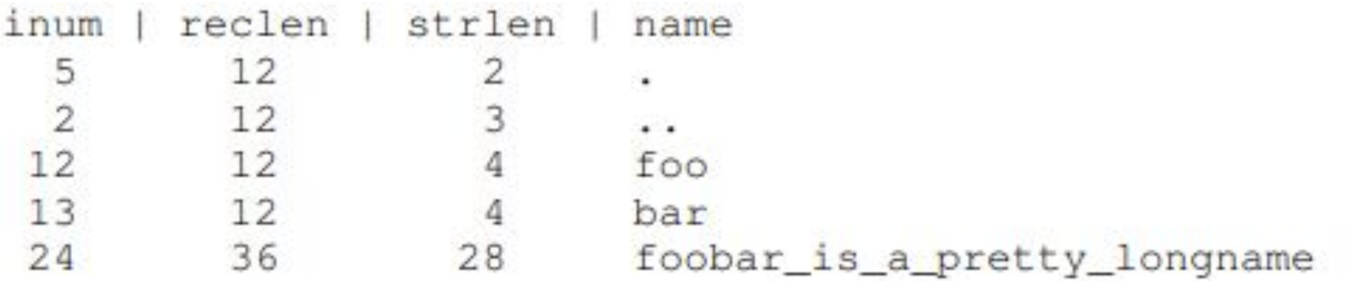
Adv compared to pointer approach?

Cons?

Small files: Inlined

- Really small files
- No need to have a separate data block
- Inline them into the inode – can access with fewer disk accesses

Directory Organization

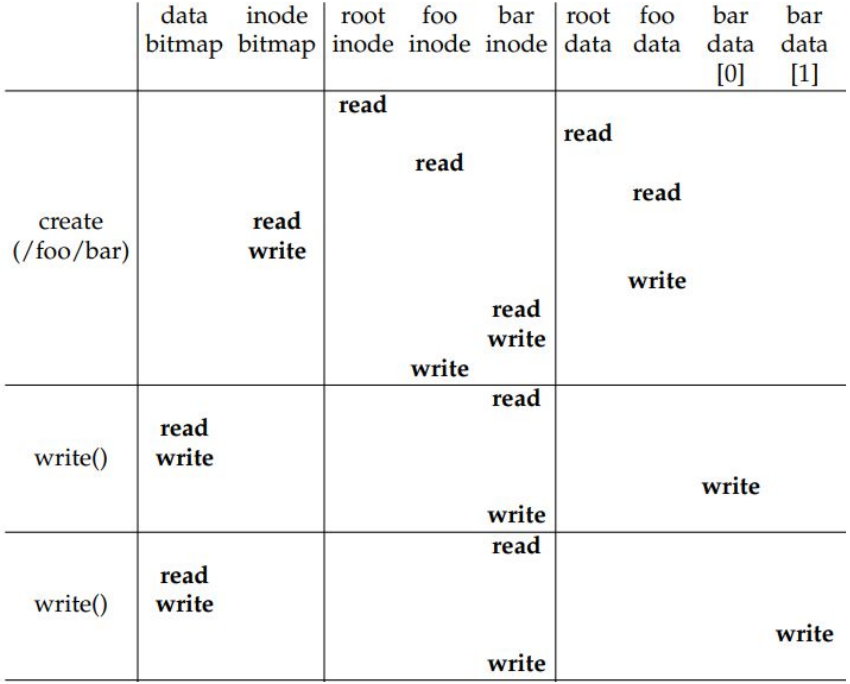


```
inum | reclen | strlen | name
  5      12        2      .
  2      12        3      ..
 12      12        4      foo
 13      12        4      bar
 24      36       28      foobar_is_a_pretty_longname
```

What is the inode of this directory?

Where is the directory’s content stored?

Creating and Writing File



	data	inode	root	foo	bar	root	foo	bar	bar
	bitmap	bitmap	inode	inode	inode	data	data	data	data
								[0]	[1]
create (/foo/bar)			read						
						read			
				read					
							read		
		read							
		write							
							write		
					read				
					write				
				write					
write()					read				
	read								
	write								
								write	
					write				
write()					read				
	read								
	write								
									write
					write				

Why read foo data?

What is written in foo data?

What is written in foo inode?

why is bar inode written upon data write?

Page Cache

Disk access is expensive

Can cache blocks in memory – all FS do this

Integrated with virtual memory

- can balance fs cache vs. vm

Also helps write buffering (need to fsync for persistence)

Flushing deamon

Crash Consistency

Basic problem:

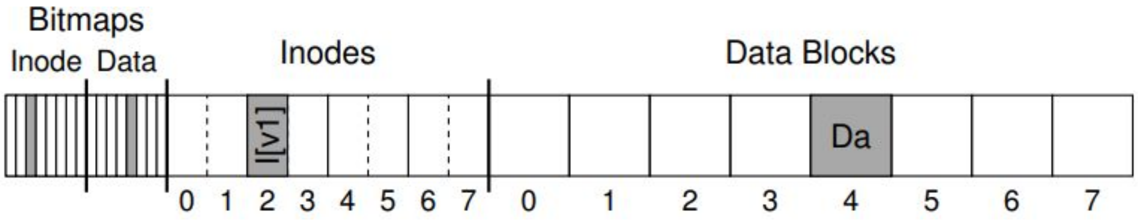
Must update many data structure on disk as a unit

What if failure happens in the middle

Types of failure:

- kernel panic
- power failures

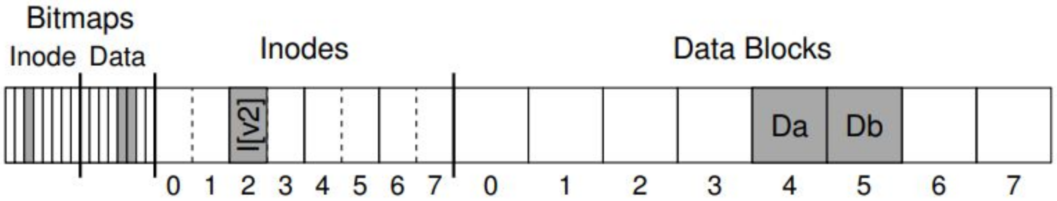
Append a Block Example



How many blocks do we need to write to accomplish the append?

Which ones?

Problems



What if only Db is written?

Only i[V2] is written to disk? (2 problems)

Data bitmap is alone written to disk?

Bitmap and data are written:

Data and inode are written:

Bitmap and inode are written:

What’s special about the last case?

Metadata vs. Data

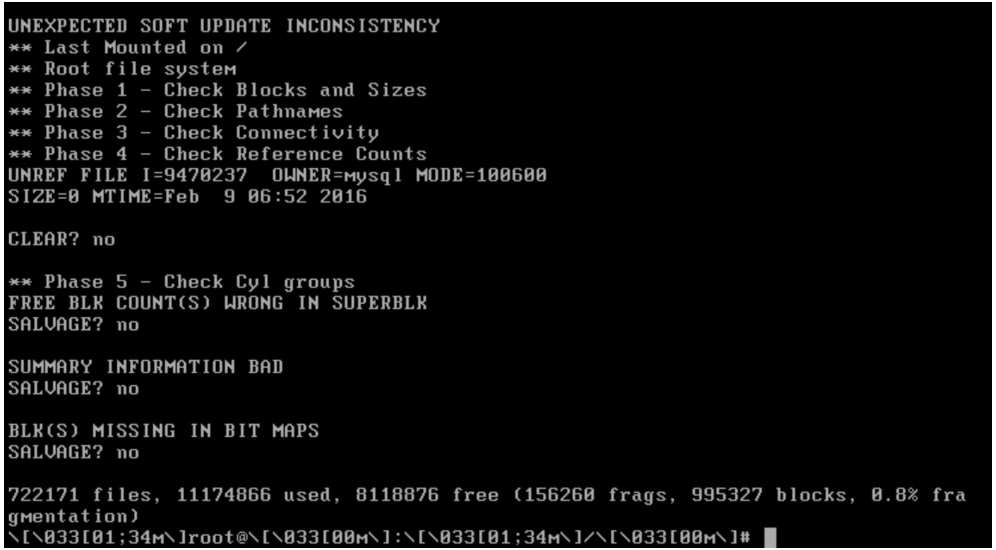
FS Metadata consistency vs. Data consistency

FS metadata consistency: internal structures agree with each other

Data consistency: additionally, the data must “make sense” to applications and users

FSCK

Let inconsistencies happen and take care during reboot

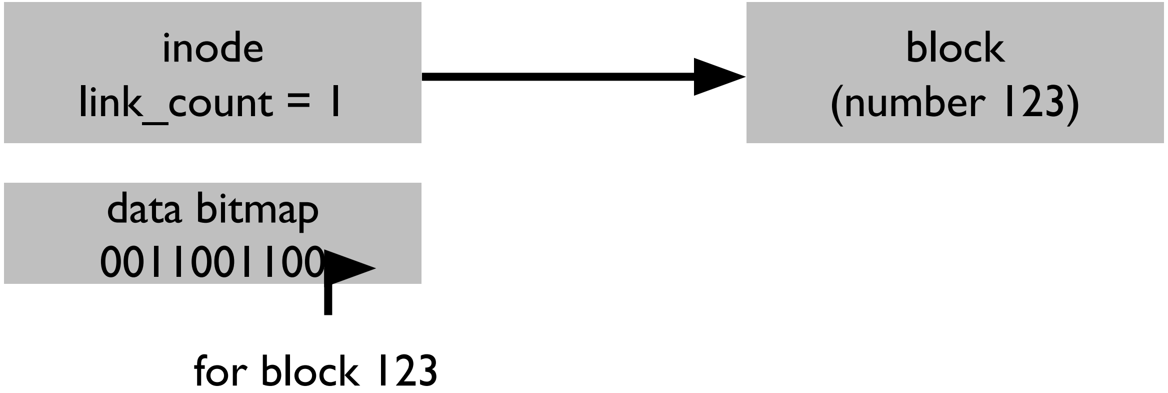


```
UNEXPECTED SOFT UPDATE INCONSISTENCY
** Last Mounted on /
** Root file system
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
UNREF FILE I=9470237  OWNER=mysql MODE=100600
SIZE=0 MTIME=Feb  9 06:52 2016

CLEAR? no

** Phase 5 - Check Cyl groups
FREE BLK COUNT(S) WRONG IN SUPERBLK
SALVAGE? no

SUMMARY INFORMATION BAD
SALVAGE? no

BLK(S) MISSING IN BIT MAPS
SALVAGE? no

722171 files, 11174866 used, 8118876 free (156260 frags, 995327 blocks, 0.8% fra
gmentation)
\[\033[01;34m\]root@\[\033[00m\]:\[\033[01;34m\]/\[\033[00m\]#
```

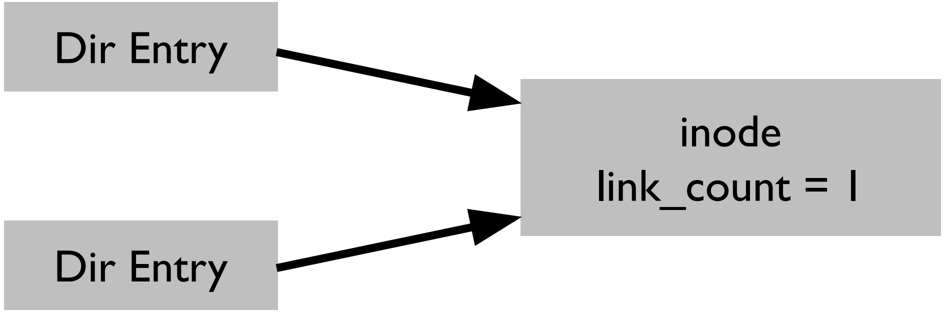
FSCK CHECKS

Do superblocks match?
Is the list of free blocks correct?
Do number of dir entries equal inode link counts?
Do different inodes ever point to same block?
Are there any bad block pointers?
Do directories contain "." and ".."?
…

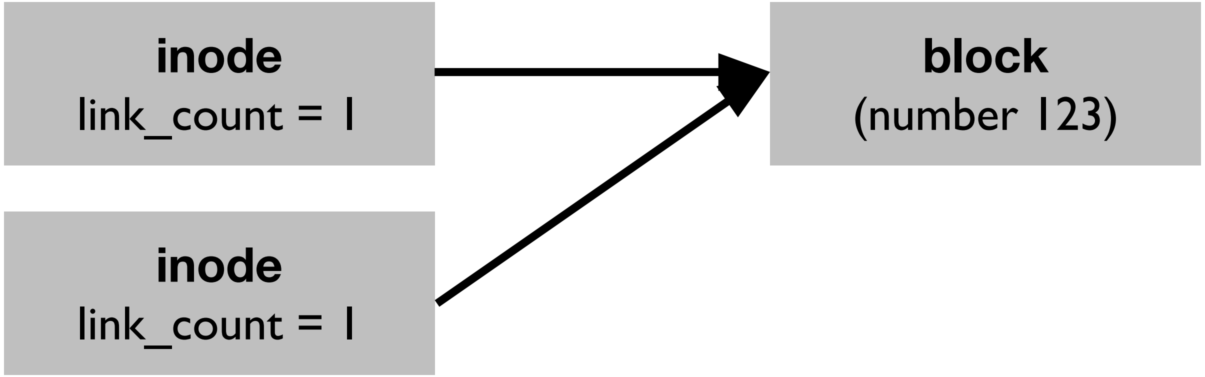
Free Blocks Example



Link Count Example



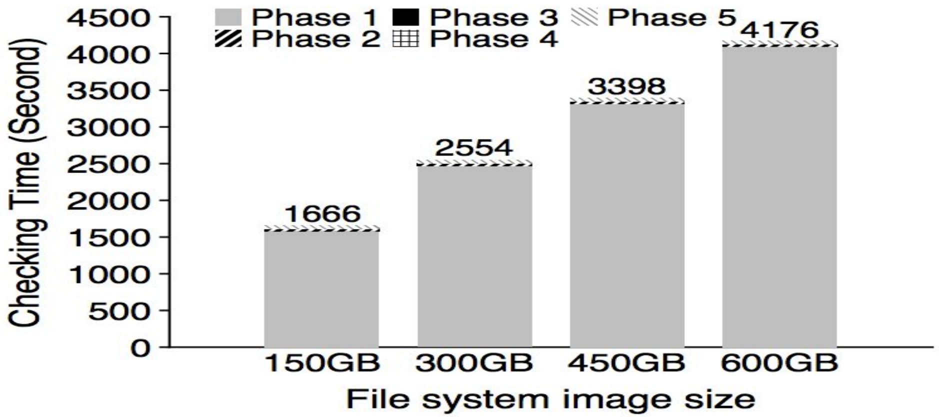
DUPLICATE POINTERS



FSCK PROBLEMS

Not always obvious how to fix file system image - don't know "correct" state, just consistent one

Simply too slow!



Checking a 600GB disk takes ~70 minutes

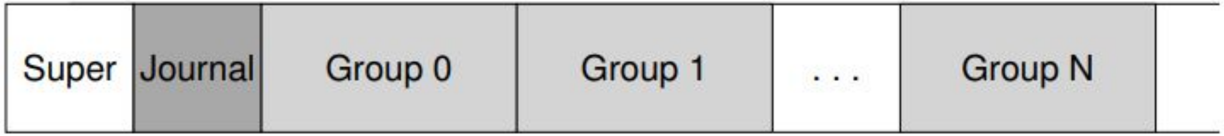
ffsck: The Fast File System Checker
Ao Ma, Chris Dragga, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

Journaling or WAL

Main idea: write a “note” to a well-known location before actually writing the blocks

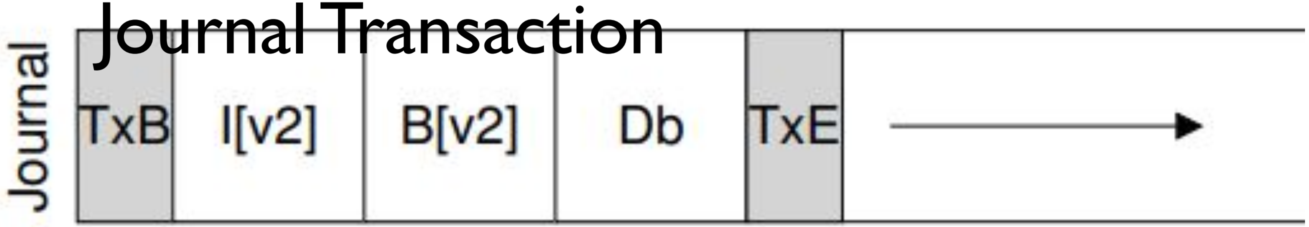
If crash, know what to fix and how to do so from the note (instead of scanning the entire disk)

Journaling in Linux ext3



Append a block to an existing file example

Journal Transaction



Data journaling vs. metadata journaling

Journaling or WAL

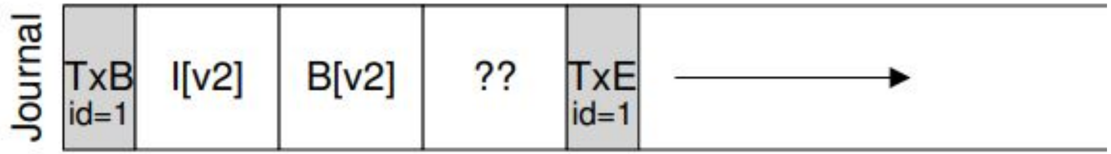
First write the txn to journal

Once that is safe, write the actual blocks (this is called checkpointing)

What if crash happens during journal write?

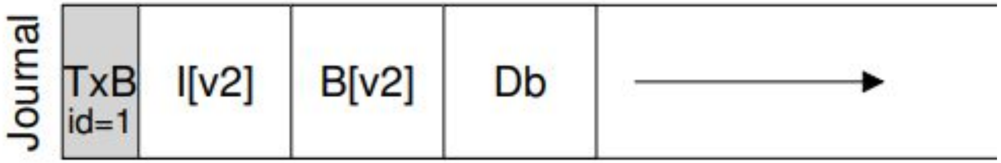
Journal Writes

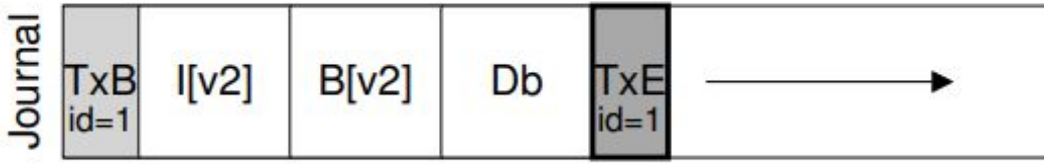
Can issue one write at a time but is too slow
Must maximize how many writes can be concurrently sent
But sending all 5 blocks together is problematic



How to solve this?

One solution





Incurs a wait or flush between TxB + Data and TxE…

How to do without waiting?

Solution without Wait

What is the problem with DJ?

Next Lecture

Continue CC (more journaling + LFS)

Then:

Advanced storage-1: RAID, NFS

Advanced storage-2: AFS, GFS