CS 423
Operating System Design: File Systems - II

Ram Alagappan

Acks: Prof. Tianyin Xu and
Prof. Shivaram Venkataraman (Wisconsin) for the slides.

Recap

File names: FD, inode, path names

Directories: name → inode mapping

FD table – per process

Open file table – OS level

Fork, dup – sharing of OFT entries

Fsync, rename, unlink

Hard links vs. soft link (symlinks)

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

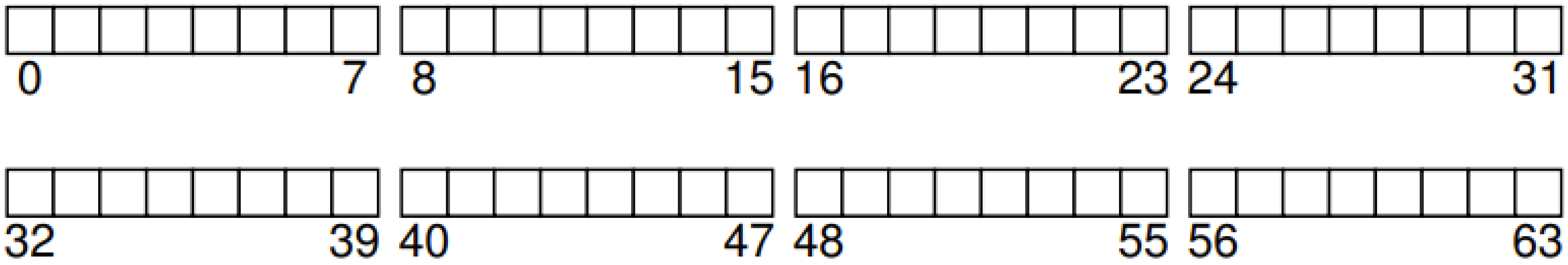
Two aspects:

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

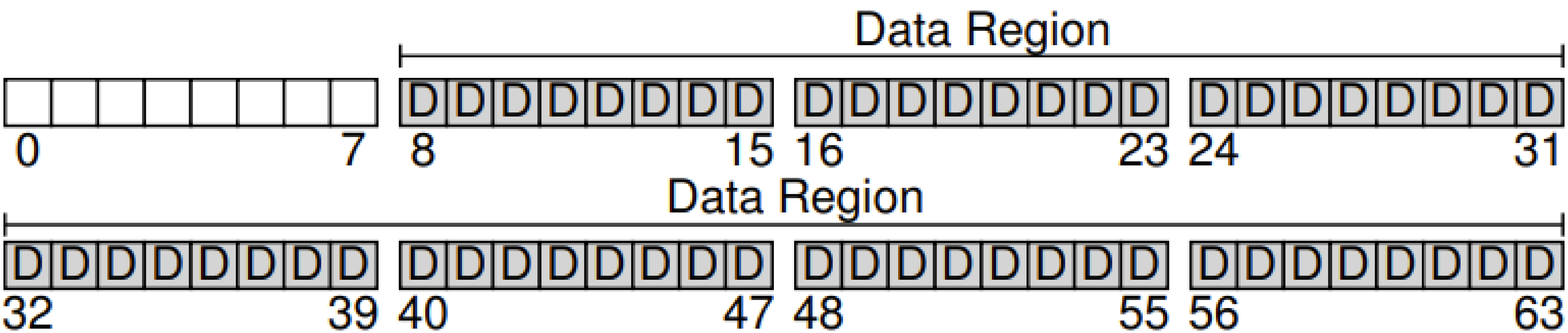
VSFS

Assume a small disk partition with 64 blocks

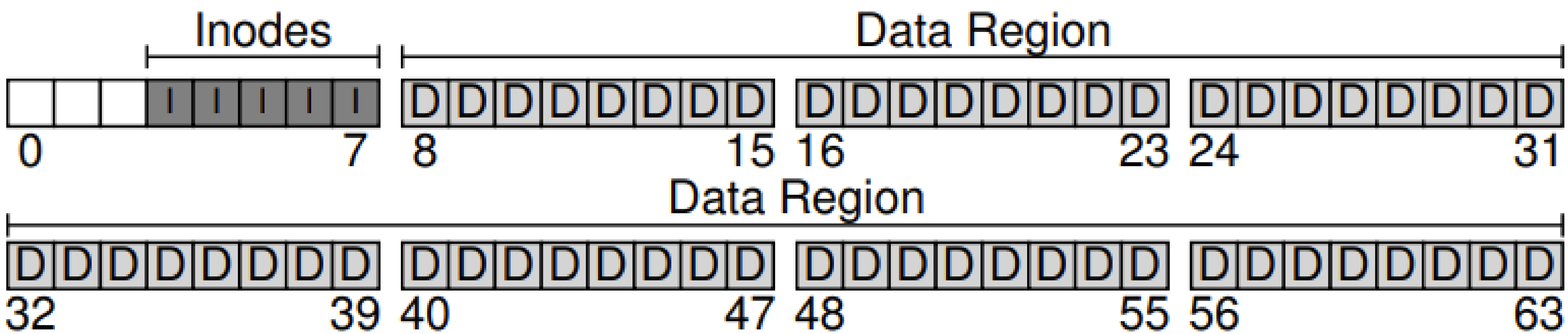


Data and metadata – most space must go for data blocks

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

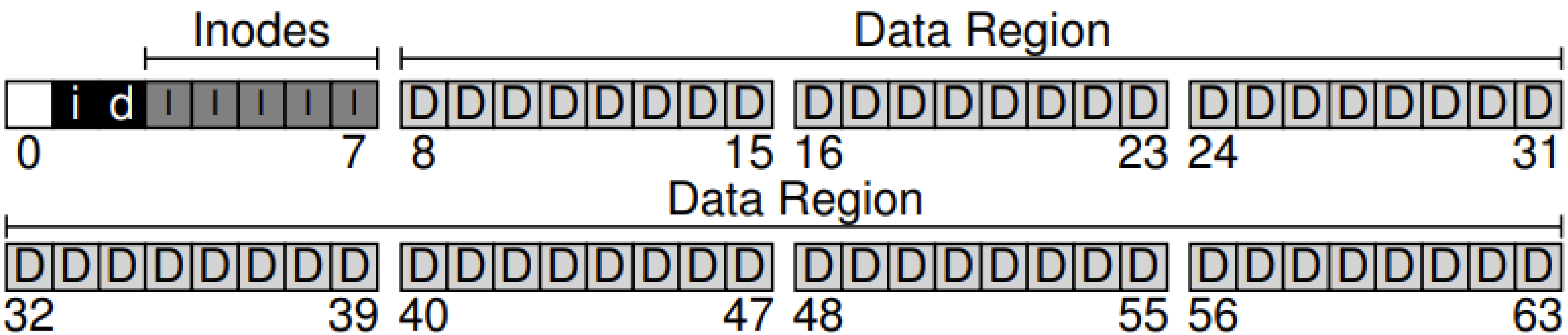
But we can simply scale VSFS to a larger disk

VSFS – Bitmaps (metadata)

Need allocation structures

Free lists – linked list is an option

Most commonly used: bitmap (ib, db)

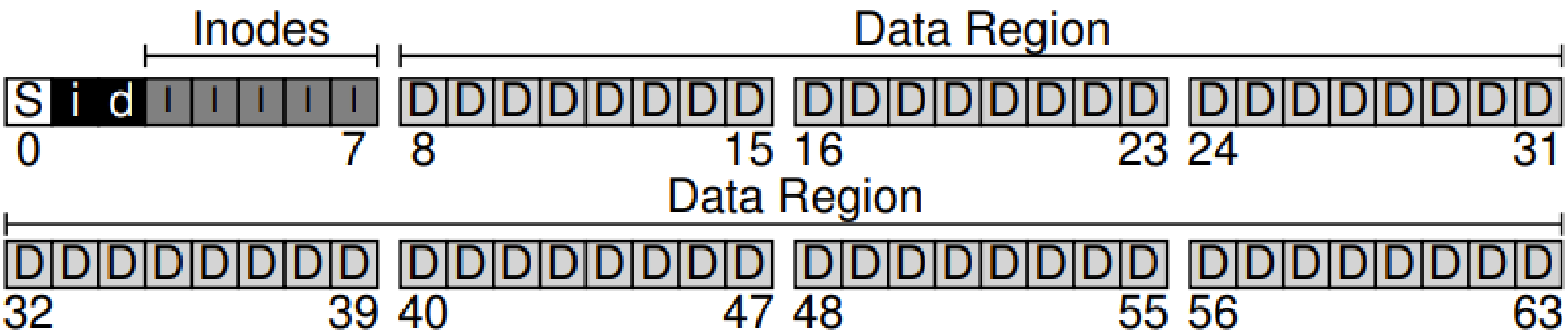


We actually don't need a block for bitmap

VSFS – (metadata)

What’s stored in the first block?

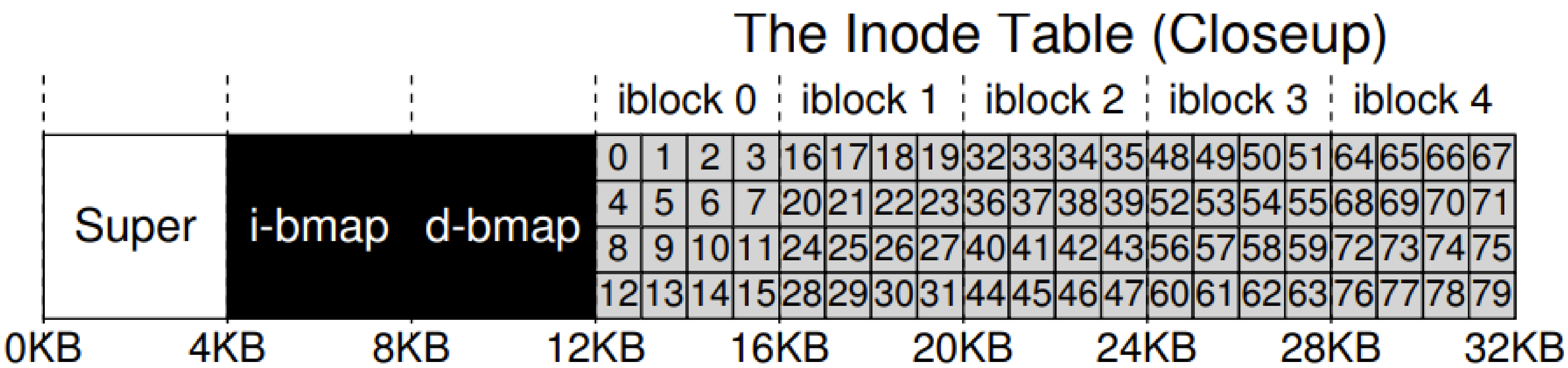
VSFS – Superblock (metadata)



How many data blocks, how many inode blocks

Inode table starting block #

INODE



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

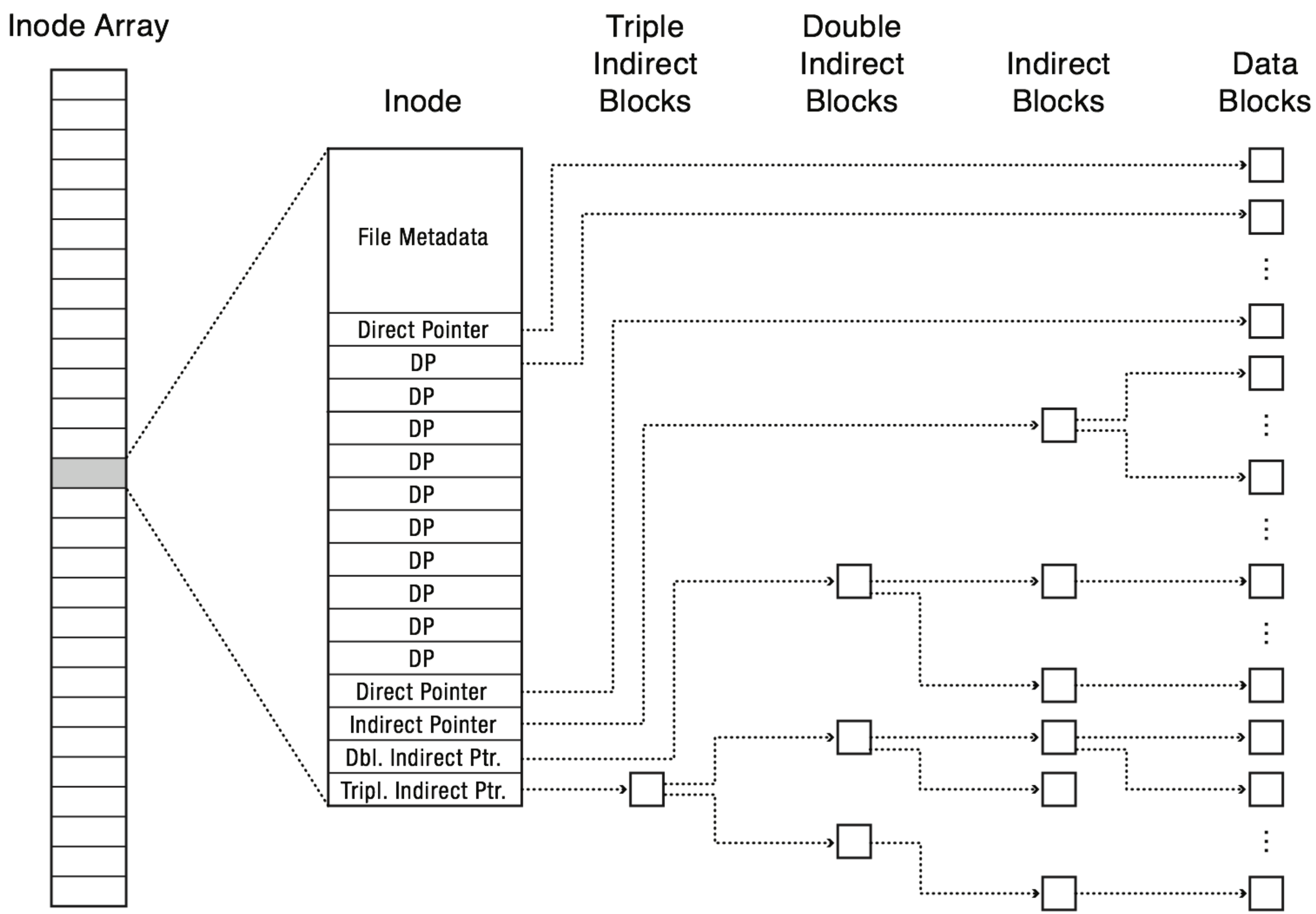
What is the max file size?

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

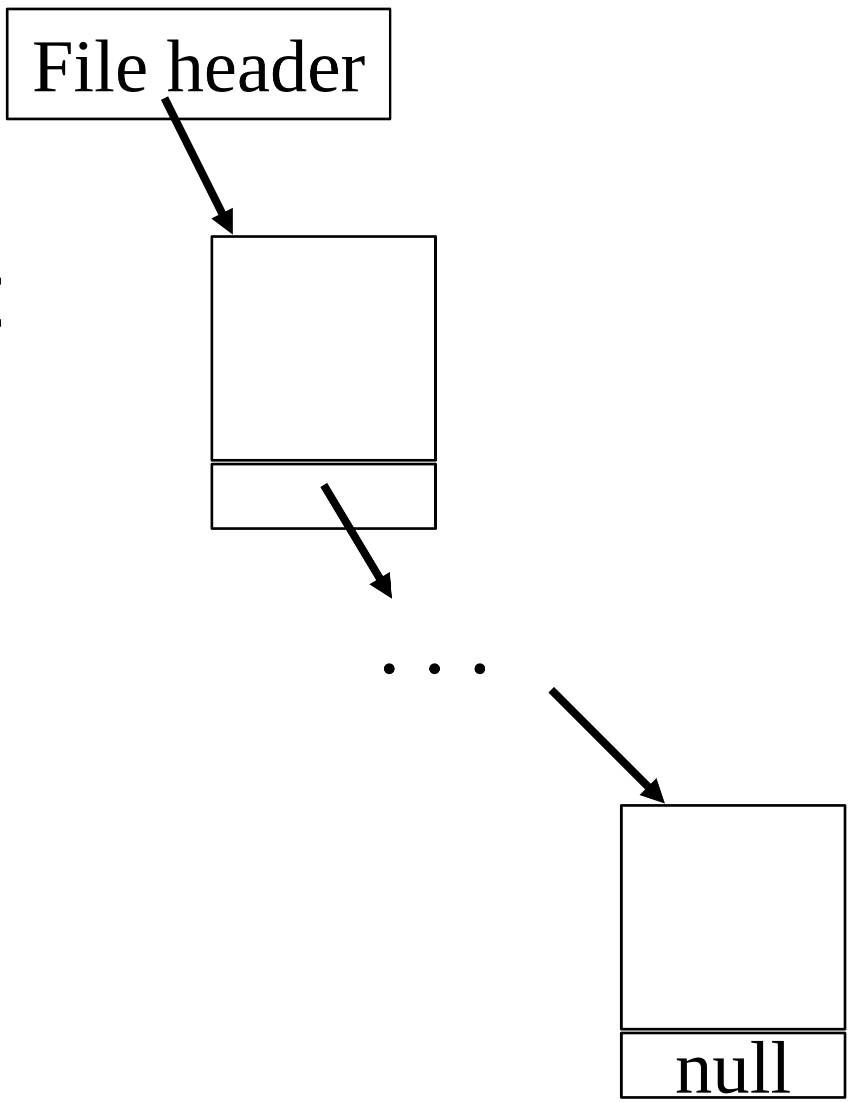
<Starting block, num blocks>

Adv compared to pointer approach?

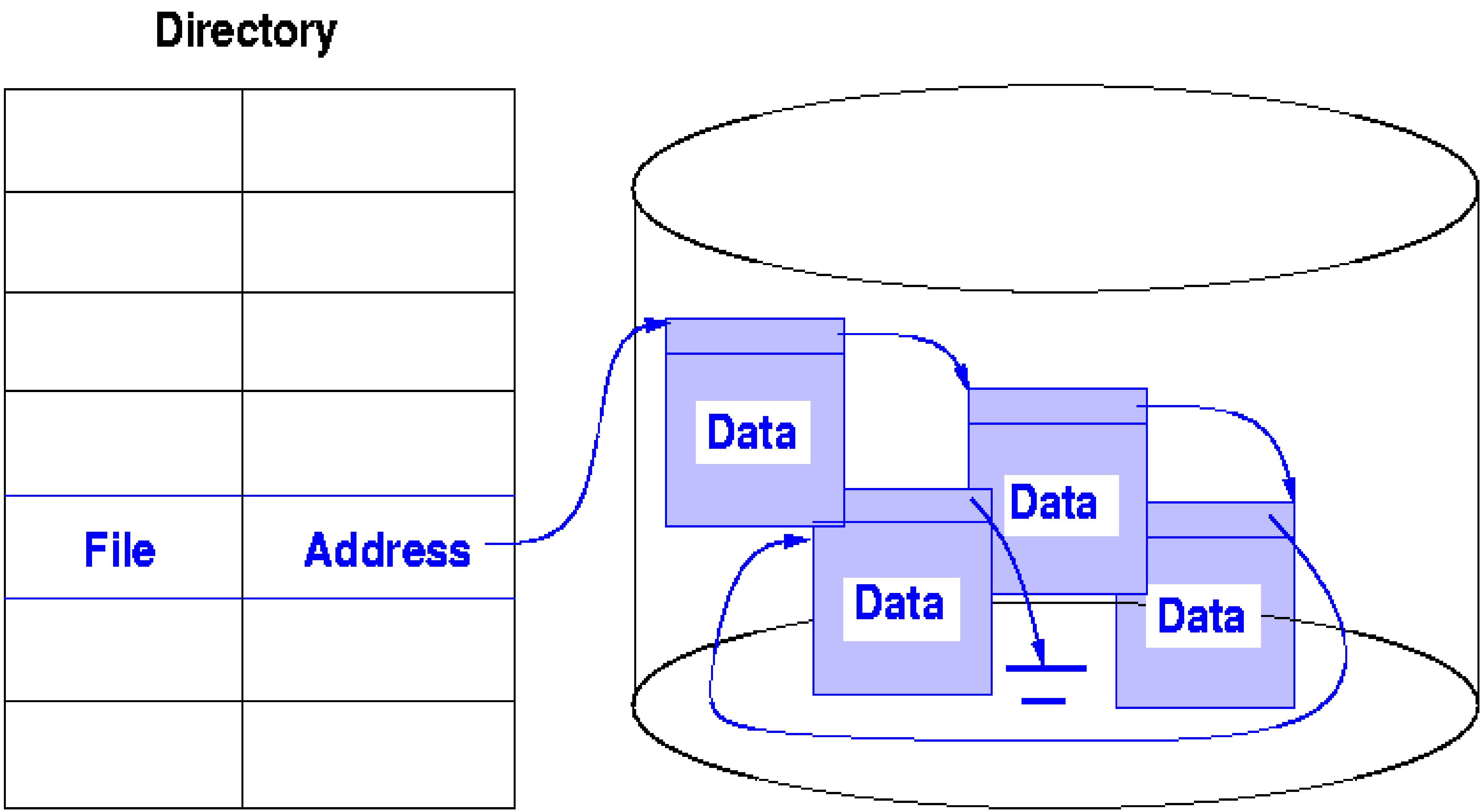
Cons?

Linked Files

- File header points to 1st block on disk
- Each block points to next
- Pros
  - Can grow files dynamically
  - Free list is similar to a file
- Cons
  - random access: horrible
  - unreliable: losing a block means losing the rest



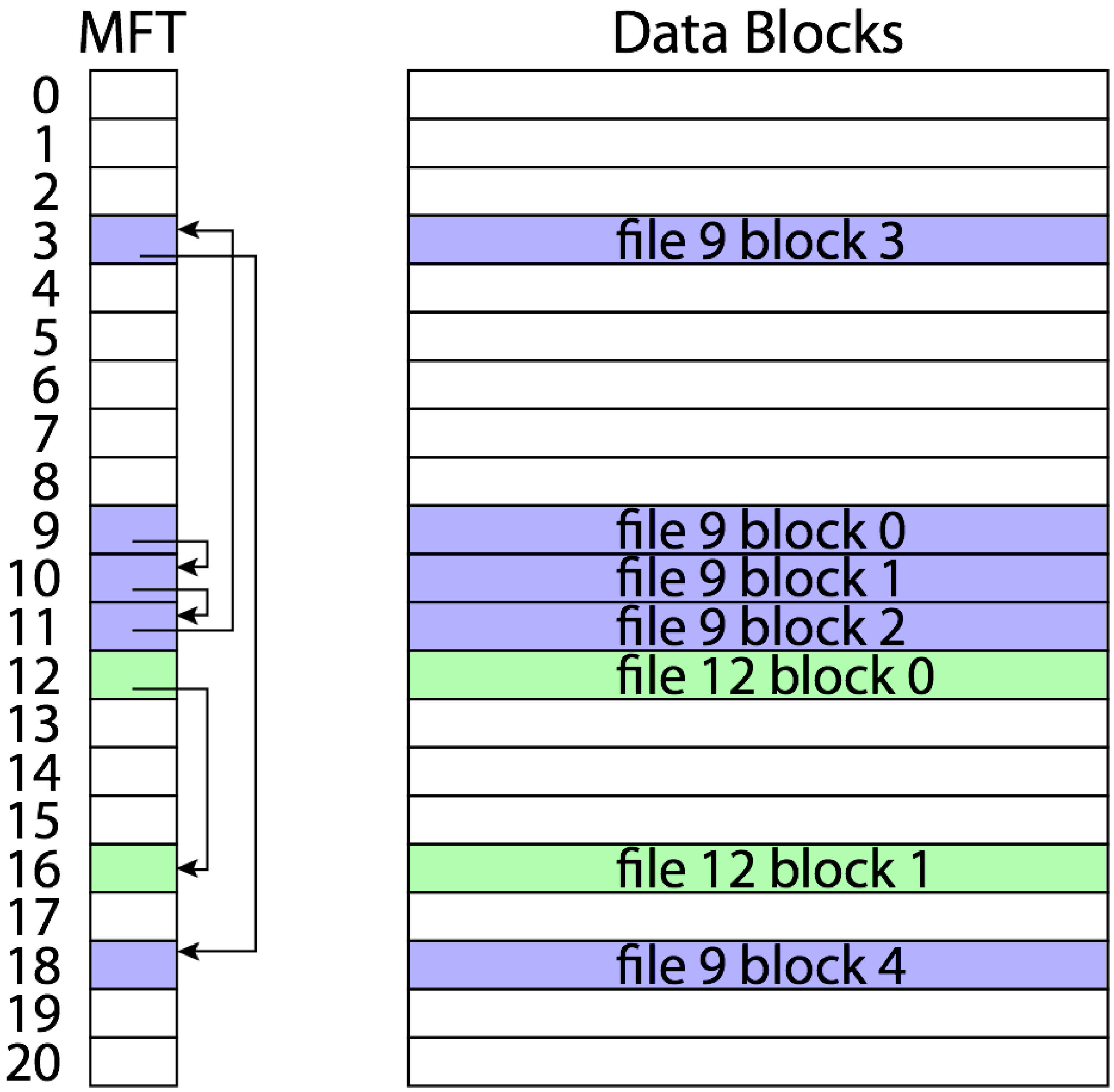
Linked Allocation



MS File Allocation Table (FAT)

- Linked list index structure
  - Simple, easy to implement
  - Still widely used (e.g., thumb drives)
- File table:
  - Linear map of all blocks on disk
  - Each file a linked list of blocks

MS File Allocation Table (FAT)



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 bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data [0]	bar data [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 bar inode?

Will you ever need to write data bitmap on file create?

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:

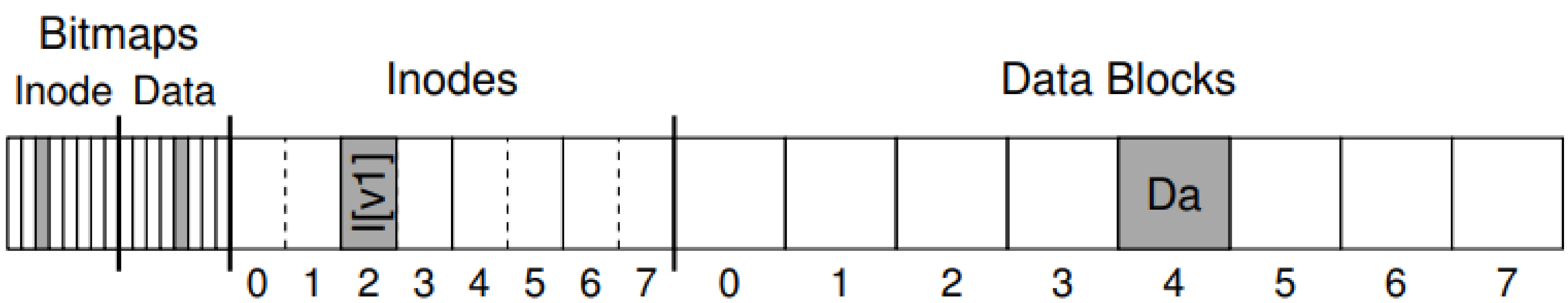
Must update many data structure on disk as a unit

What if failure happens in the middle

Types of failure:

- kernel panic
- power failures

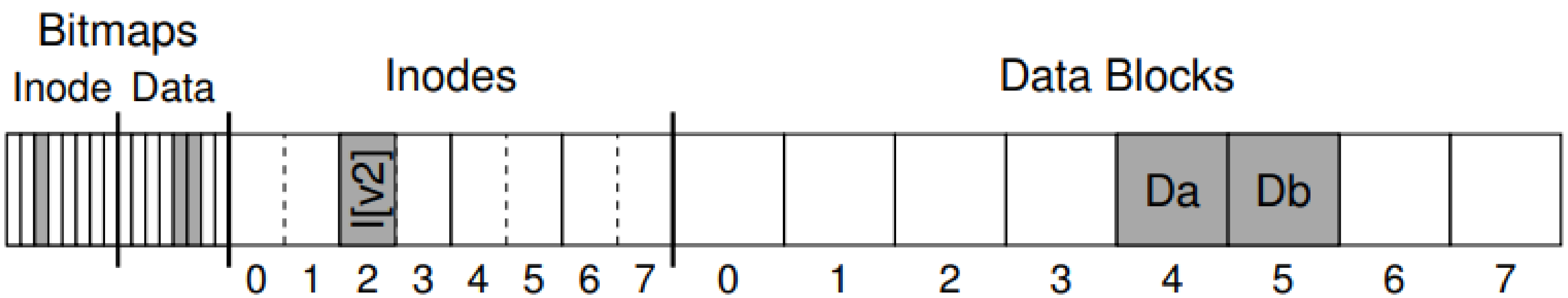
Append a Block Example



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?

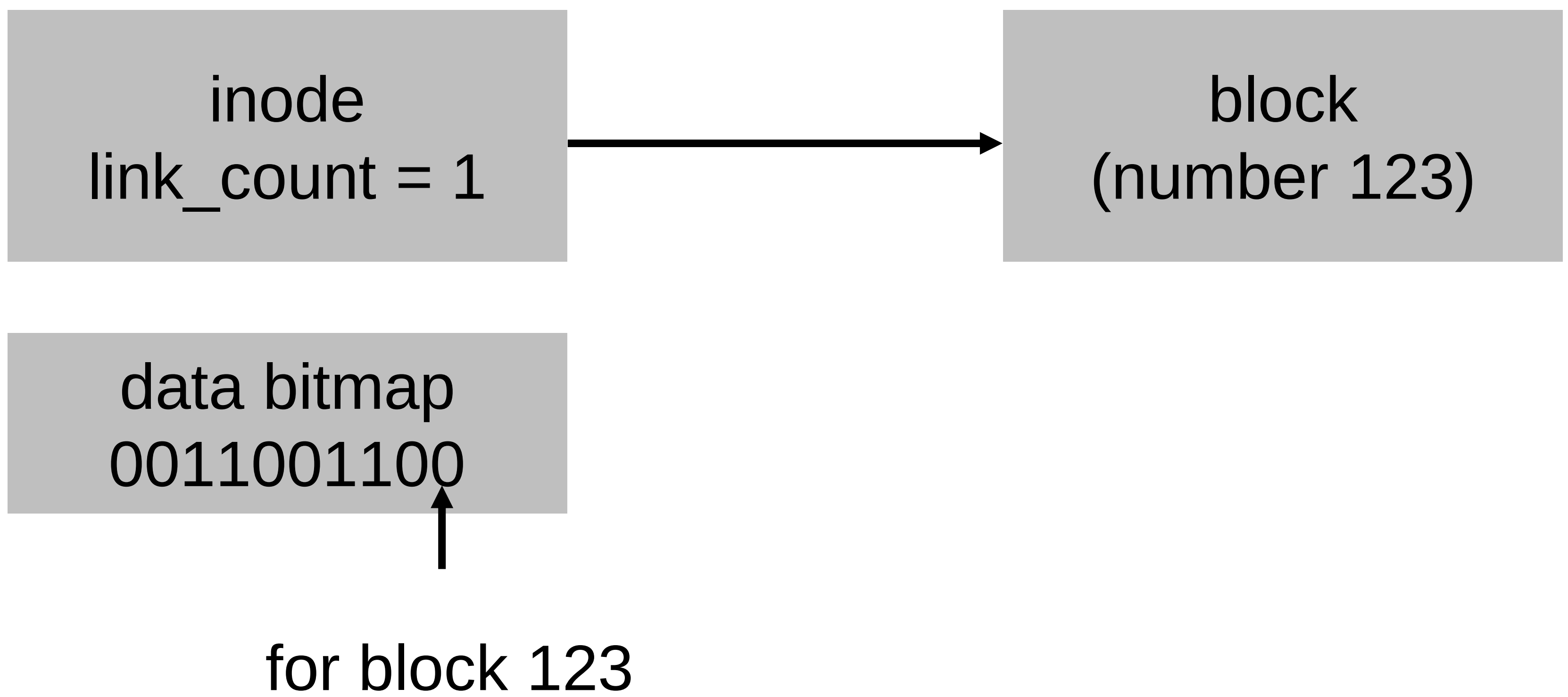
Do different inodes ever point to same block?

Are there any bad block pointers?

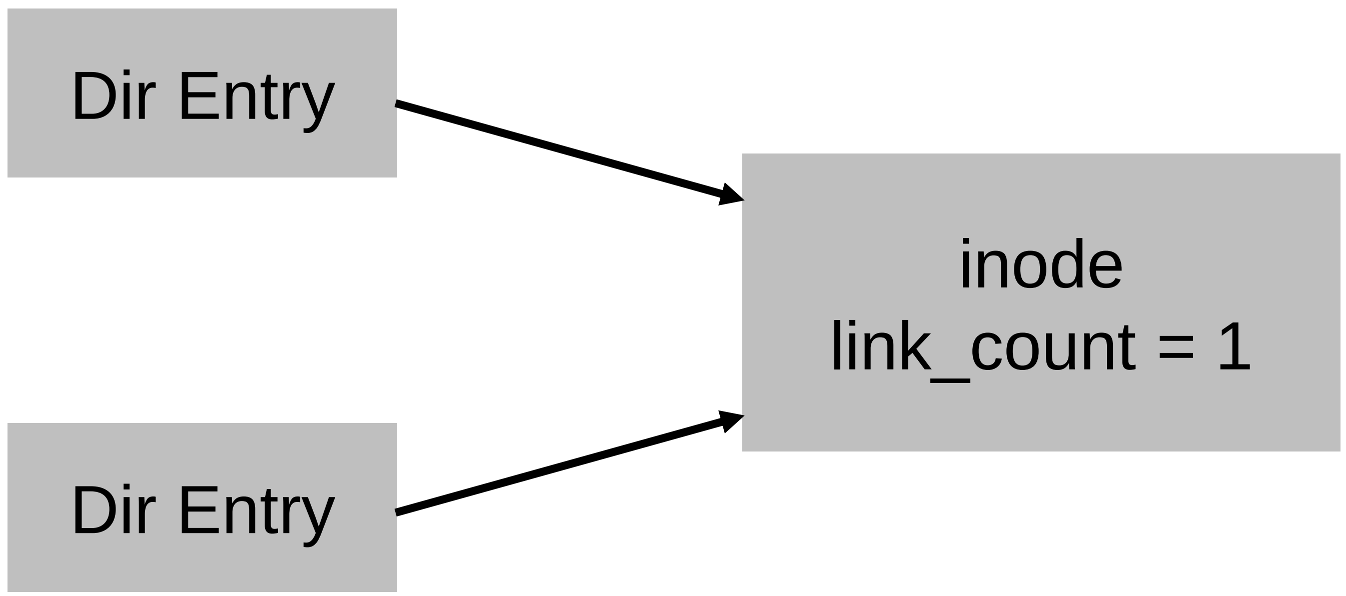
Do directories contain “.” and “..”?

…

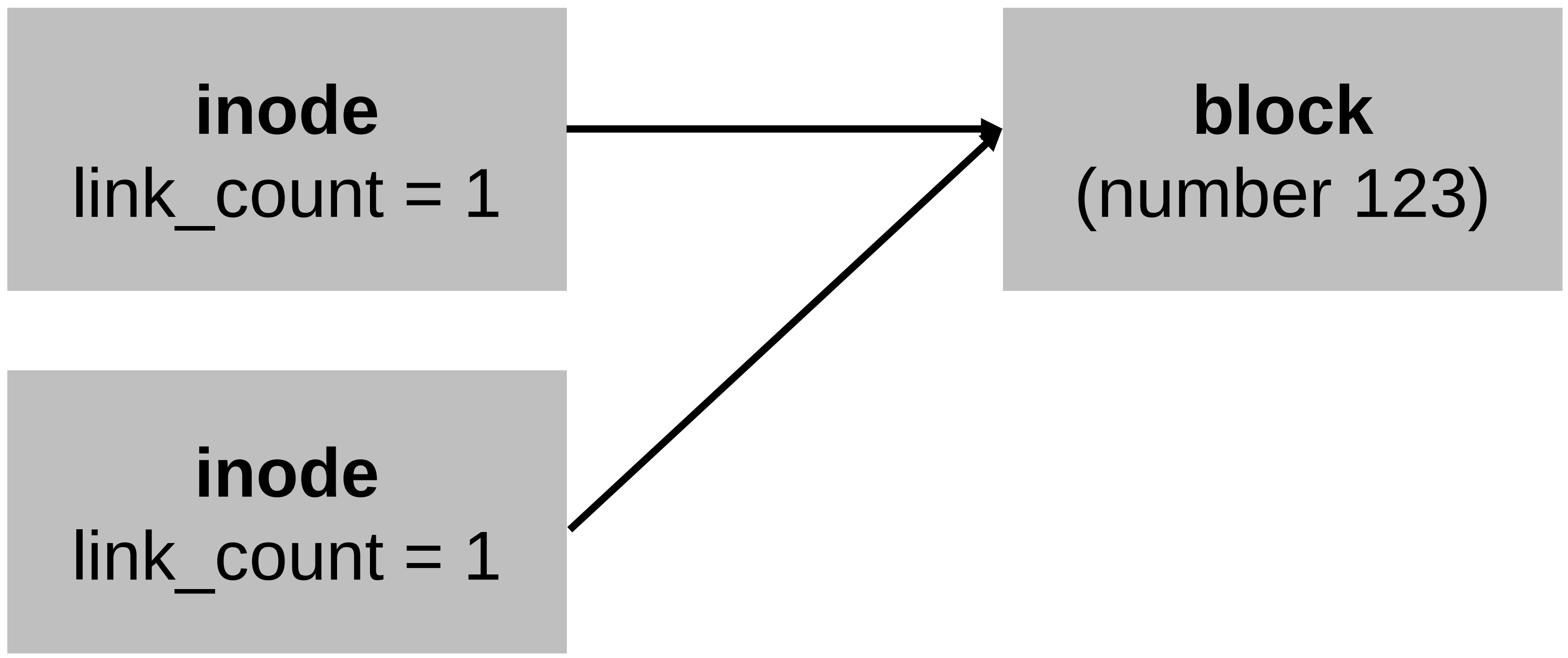
Free Blocks Example



Link Count Example



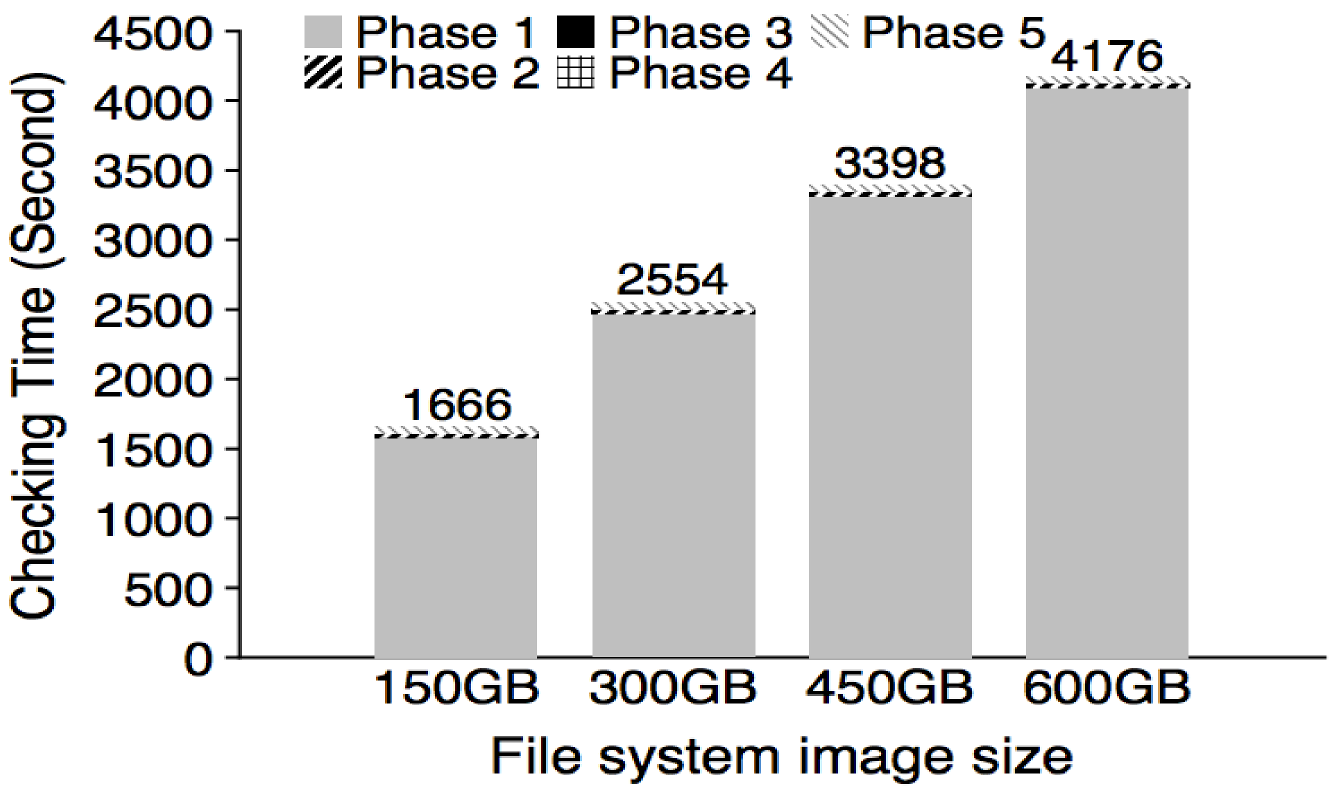
DUPLICATE POINTERS



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

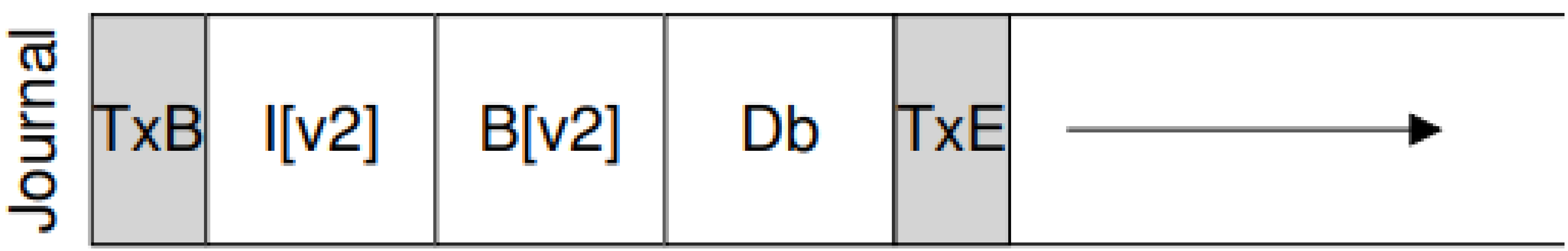
Main idea: write a “note” to a well-known location before actually writing the blocks

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

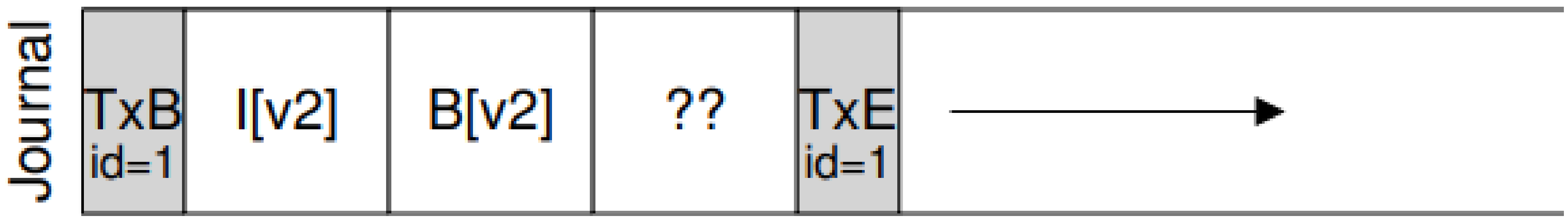
First write the txn to journal

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

What if crash happens during journal write?

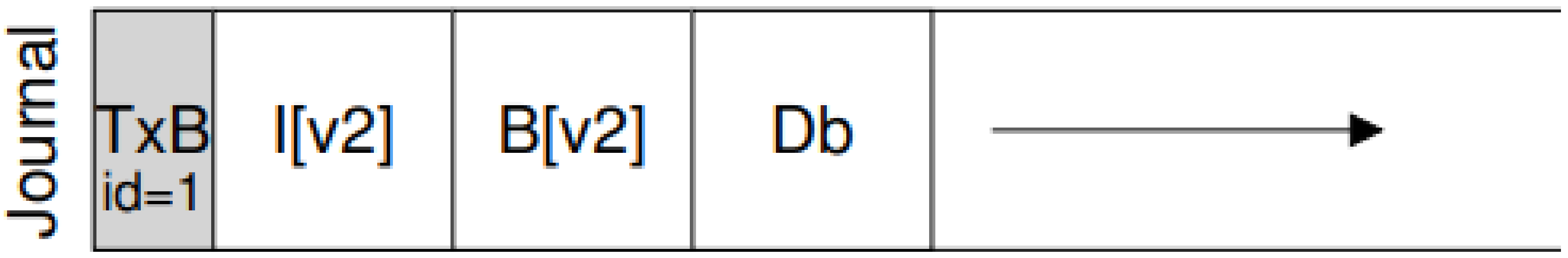
Journal Writes

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