PERSISTENCE: REVIEW

Andrea Arpaci-Dusseau

CS 537, Fall 2019

ADMINISTRIVIA

Project 7: xv6 File systems: Improvements + Checker

- Due tomorrow 5pm/midnight
- Two directories: p7/PartA and p7/PartB – see Canvas specification

Final Exam

- Friday, December 13th 7:25-9:25 pm
- Two Rooms: Last name A-H in SOC SCI 5206, I-Z in Humanities 3650
- Slightly cumulative (some T/F from Virtualization and Concurrency – 25%)

Exam Review

- Today: Cover questions you asked
- Tomorrow: discussions

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization

2. Concurrency

3. Persistence

VIRTUALIZATION

Make each application believe it has each resource to itself

CPU and Memory

Abstraction: Process API, Address spaces

Mechanism:

- Limited direct execution, CPU scheduling
- Address translation (segmentation, paging, TLB)

Policy: MLFQ, LRU etc.

CONCURRENCY

Events occur simultaneously and may interact with one another

Need to

- Hide concurrency from independent processes
- Manage concurrency with interacting processes

Provide abstractions (locks, semaphores, condition variables etc.)

Correctness: mutual exclusion, ordering

Difficult to write multi-threaded applications!

PERSISTENCE

How to ensure data is available across reboots

- even after power outages, hardware failure, system crashes?

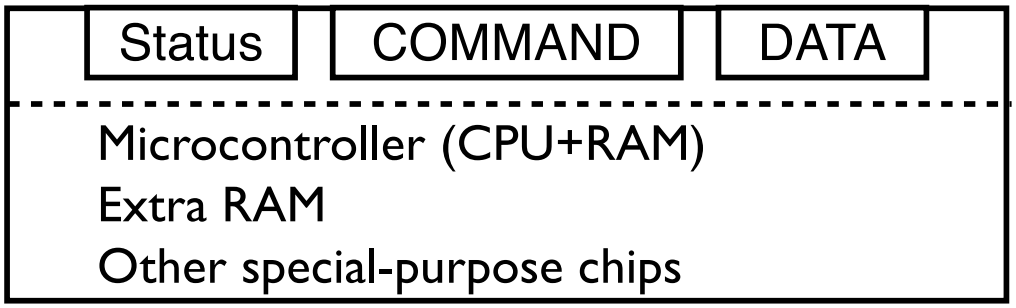
Topics:

- Persistent storage devices: HDDs, RAID, SSDs
- File API for processes
- FS implementation: meta-data structures, allocation policies in FFS
- Crash recovery: journaling (ext3) and copy-on-write(LFS)
- Distributed file systems: NFS and AFS

I/O DEVICES

- For canonical device, why do we have to write data first then write command, is there a way to reverse these?

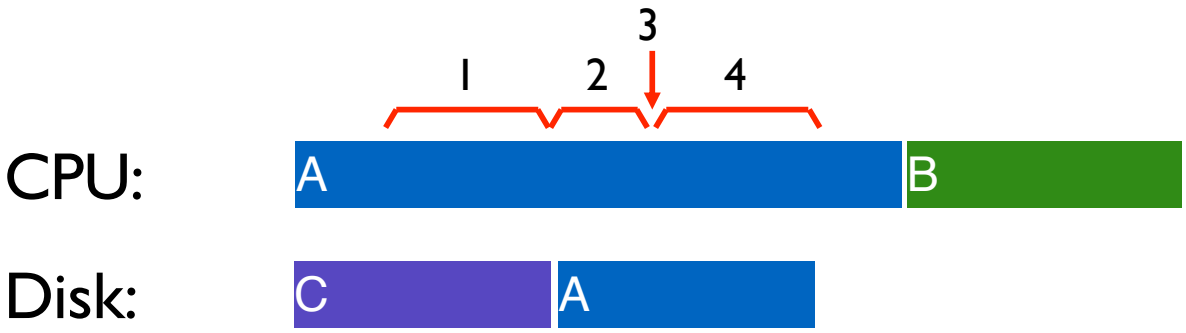
EXAMPLE WRITE PROTOCOL: STARTING POINT




```
while (STATUS == BUSY)
   ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
   ; // spin
```

IO DEVICES

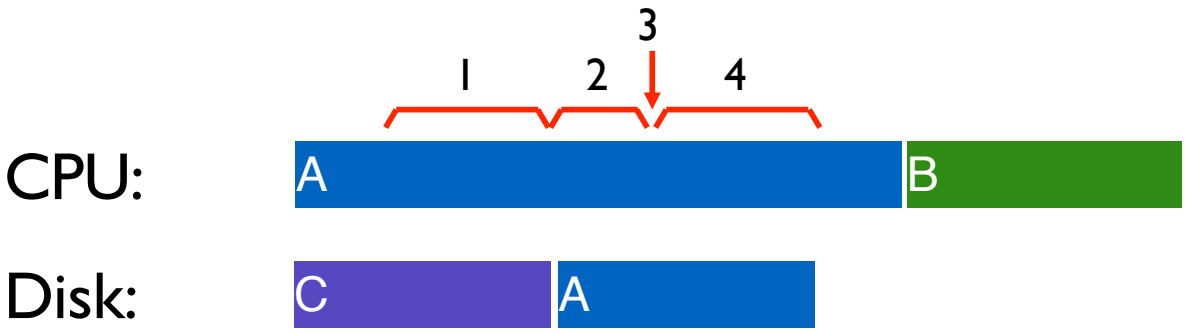
After process A finished writing command, there is still A in CPU and Disk, I assume it has finished I/O, should it give up disk and only stays in CPU?



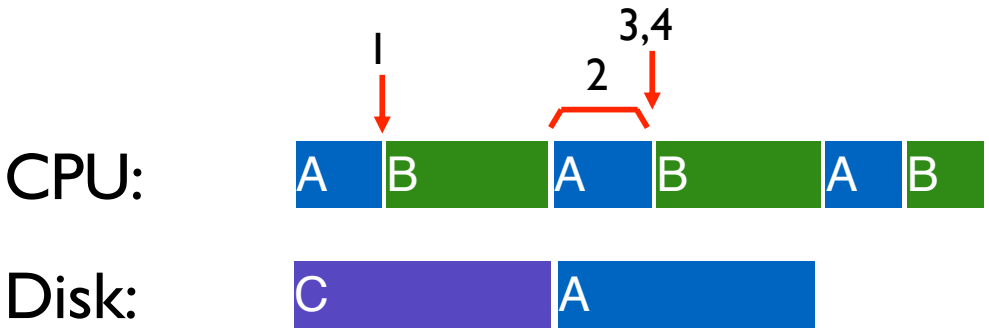

```
while (STATUS == BUSY)                  // 1
  ;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
  ;
```

how to avoid spinning?

interrupts!




```
while (STATUS == BUSY)                  // 1
      wait for interrupt;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
      wait for interrupt;
```




```
while (STATUS == BUSY)                  // 1
      wait for interrupt;
Write data to DATA register             // 2
Write command to COMMAND register       // 3
while (STATUS == BUSY)                  // 4
      wait for interrupt;
```

EXAMPLE WRITE PROTOCOL: INTERRUPTS

INTERRUPTS VS. POLLING

Are interrupts always better than polling?

Fast device: Better to spin than take interrupt overhead

- Device time unknown? Hybrid approach (spin then use interrupts)

Flood of interrupts arrive

- Can lead to livelock (always handling interrupts)
- Better to ignore interrupts while make some progress handling them

Other improvement

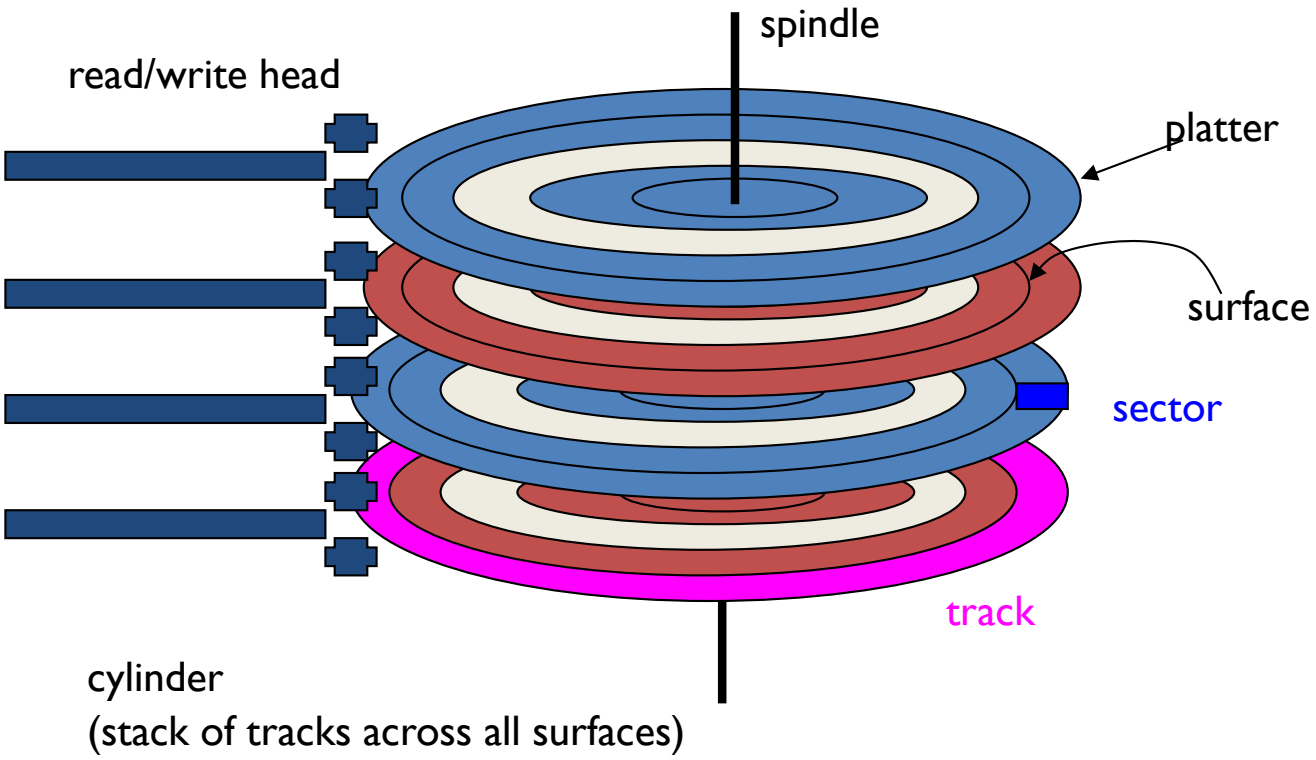
- Interrupt coalescing (batch together several interrupts)

HARD DISKS

IO DEVICES

- If a disk is "double sided", does that mean that the number of heads = 2 & the number of sector/cylinder is doubled as well? Ie if it has 10 sectors/track, 2 double-sided platters, it would be 10 * 2 * 2 sectors/cylinder?

DISK TERMINOLOGY

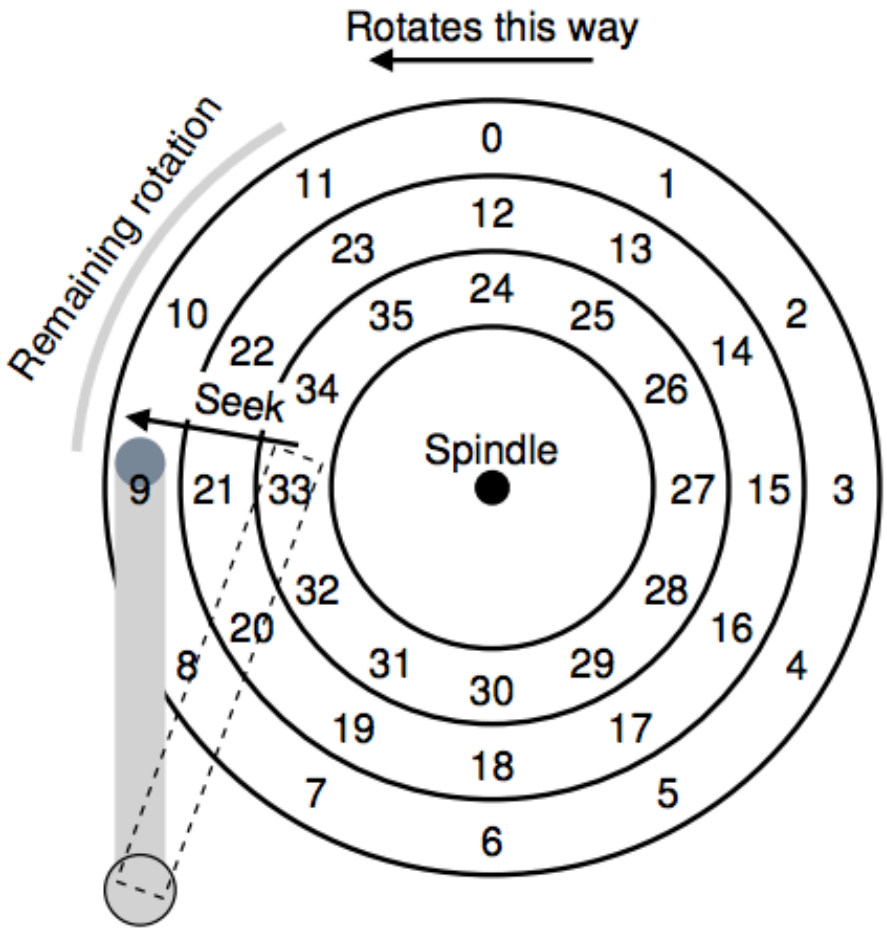


TIME TO READ/WRITE

Three components:

Time = seek + rotation + transfer time

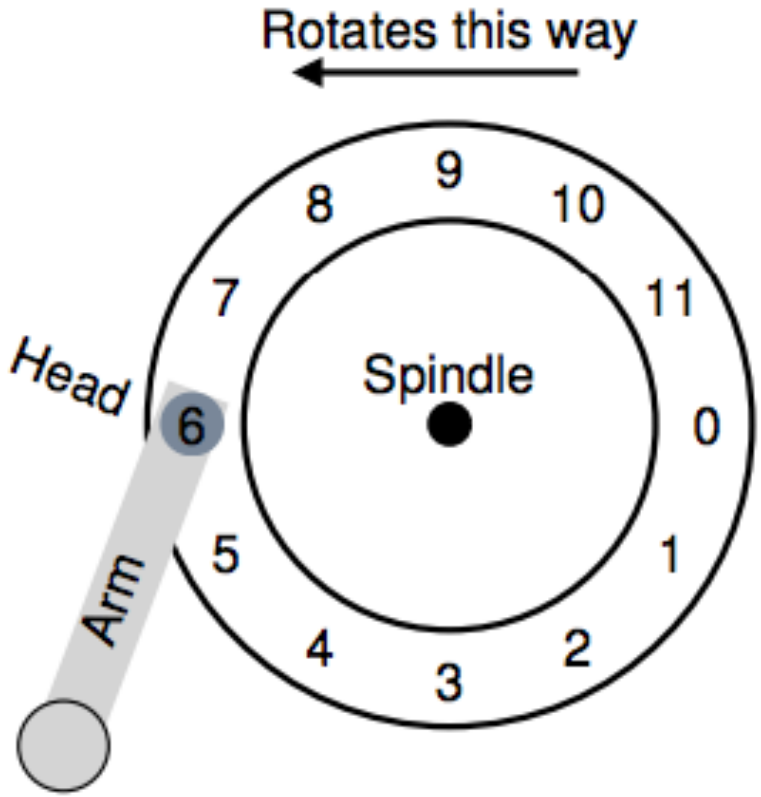
READING DATA FROM DISK



Example: Read sector 1

1) Seek

READING DATA FROM DISK



Example: Read sector 1

2) Rotational delay

3) Transfer time

IO DEVICES

- What's the difference between scan and c-scan? Does c-scan go from outside to inside and then inside to outside or just one direction? How can we know he time cost/distance of c-scan bouncing back to outmost track from inner?

I/O SCHEDULERS

I/O SCHEDULERS

Given stream of I/O requests, in what order should they be served?

Goal: Minimize seek + rotation time

Much different than CPU scheduling

Position of disk head relative to request position matters more than length of job

AVOID STARVATION: SCAN

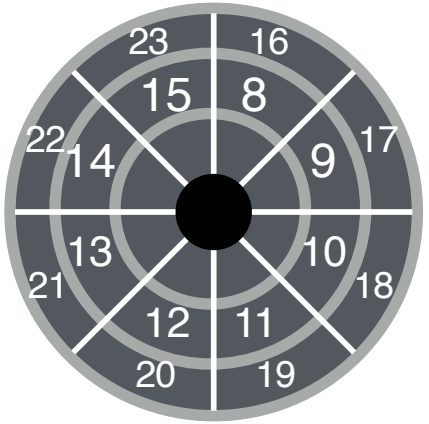
Elevator Algorithm (or Scan)

- Sweep back and forth, from one end of disk to other, and back
- Serve requests as pass that cylinder in each direction
- Sorts by cylinder number; ignores rotation delays

Disadvantage?

Better: C-SCAN (circular scan)

- Only sweep in one direction



IO DEVICES

- By "seek distance", do you just mean the total # of cylinders covered? ie if you move 12 -> 10 -> 15, it would be 2 + 5 = 7?

- I am confused between "Work conserving scheduler" and "non-Work conserving scheduler" and how can this affect the disk scheduler

WORK CONSERVATION

Work conserving schedulers always do work if work exists

- Principle applies to any type of scheduler (CPU too)

Could be better to wait if can **anticipate** another request will arrive

Such **non-work-conserving schedulers** are called **anticipatory** schedulers

- Keeps resource idle while waiting for future requests

Better stream of requests for OS to give disk: AAAAAABBBBBBAAAAAA

RAIDS

- What exactly is throughput? So like for RAID, is that the amount of data that can be read/written concurrently? Or is there another definition?

RAID-0: ANALYSIS

What is capacity?	**N * C**
How many disks can fail (no loss)?	**0**
Latency?	**D**
Throughput (sequential, random)?	**N*S , N*R**

Buying more disks improves throughput, but not latency!

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk 0	Disk 1	Disk 2	Disk3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

RAIDS

- How many disks can actually fail in RAID-1? I think it's 1 but the slides say N/2 is also allowed, but that doesn't make sense. If you have four disks w/ 0 and 1 containing the same info and they both fail, then you've lost data haven't you?

RAID-1: MIRRORING

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

How many disks can fail without losing any data?

Assume disks are **fail-stop**
- each disk works or it doesn't
- system knows when disk fails

Tougher Errors:
- latent sector errors
- silent data corruption

Always handle 1 disk failure
May handle N/2 if to different replicas

RAIDS

- How do you calculate offsets for RAID-5 left-symmetric?
- For the RAID simulator, what calculation do you need to do to find out which disk is the parity disk in RAID 5 during a write operation?
- how do you calculate left-symmetric on RAID-5 offsets?

LEFT-SYMMETRIC RAID-5

D0	D1	D2	D3	D4
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

Pattern repeats…

RAIDS

- please go through the calculation of random write of raid4 and raid5 again.

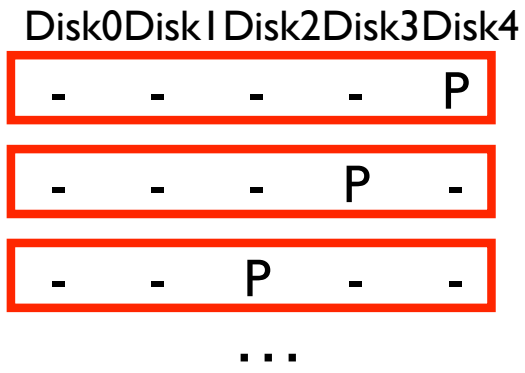
RAID-5: THROUGHPUT

Steady-state throughput for RAID-4

- sequential reads? **(N-1) * S**
- sequential writes? **(N-1) * S (parity calculated for full stripe)**
- random reads? **(N-1) * R**
- random writes? **R/2 (read and write parity disk)**

What is steady-state throughput for RAID-5?

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



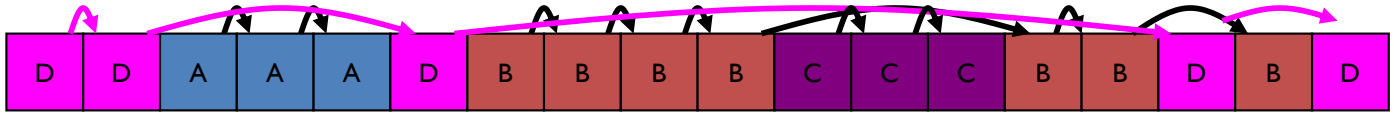
FS IMPLEMENTATION

- FS linked allocation. Trade-off: Block size (does not need to equal sector size) What is the trade-off exactly compared to contiguous allocation?

LINKED ALLOCATION

Allocate linked-list of **fixed-sized** blocks (multiple sectors)

- Meta-data: Location of first block of file
  Each block also contains pointer to next block
- Examples: TOPS-10, Alto



Fragmentation (internal and external)?	+ No external frag (use any block); internal?
Ability to grow file over time?	+ Can grow easily
Seek cost for sequential accesses?	+/- Depends on data layout
Speed to calculate random accesses?	- Ridiculously poor
Wasted space for meta-data?	- Waste pointer per block

Trade-off: Block size (does not need to equal sector size)

FILE SYSTEM IMPLEMENTATIONS

- What is the difference between an indirect block and a directory?

FILE SYSTEM IMPLEMENTATION

- 1- In creating a new file: we allocate inode for that file, then we read and write that inode, Why not to write it directly without reading?

ONE INODE BLOCK

Each inode is typically 256 bytes
(depends on the FS, maybe 128 bytes)

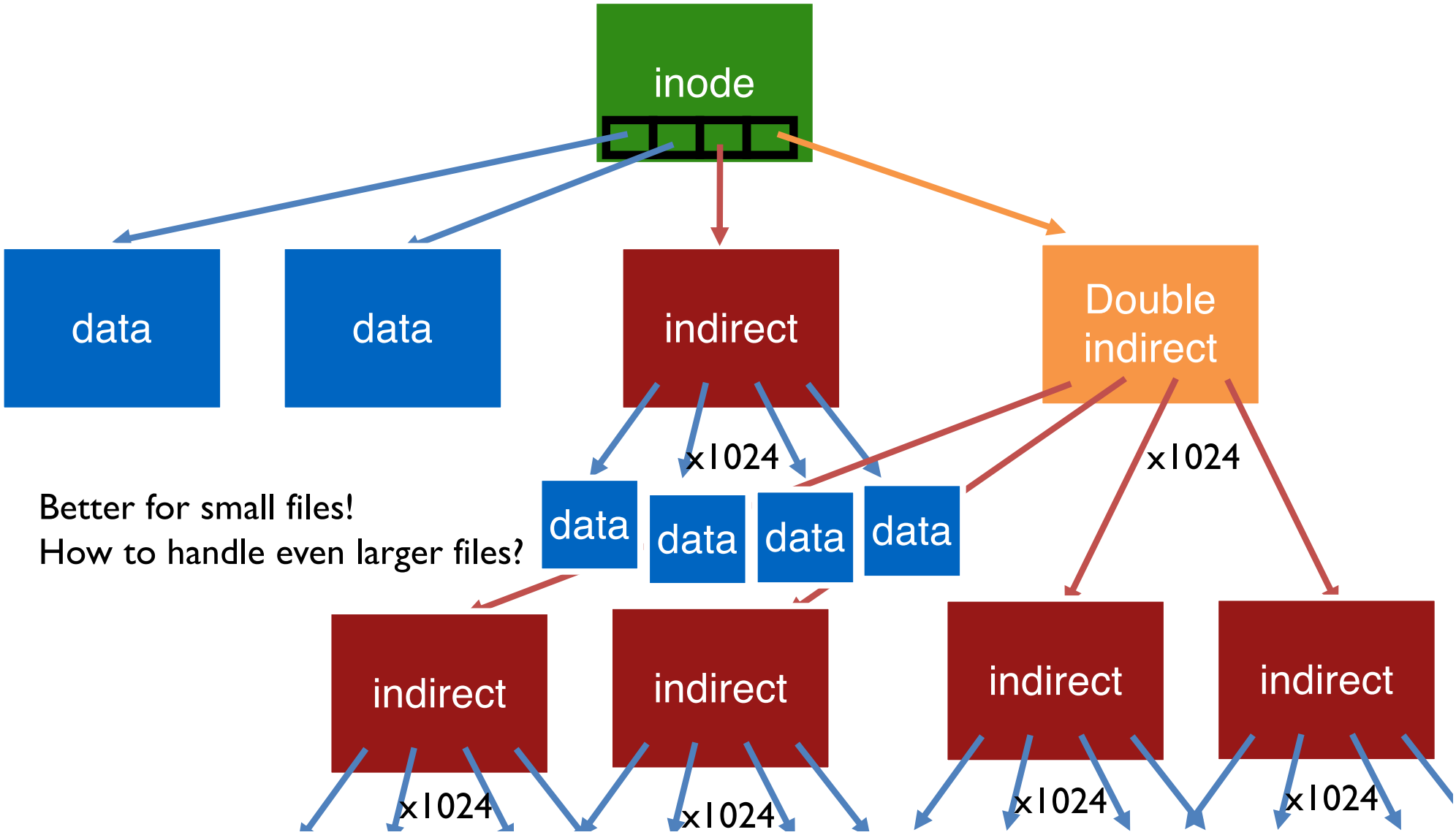
- 4KB disk block
- 16 inodes per inode block

How to modify 1 inode?

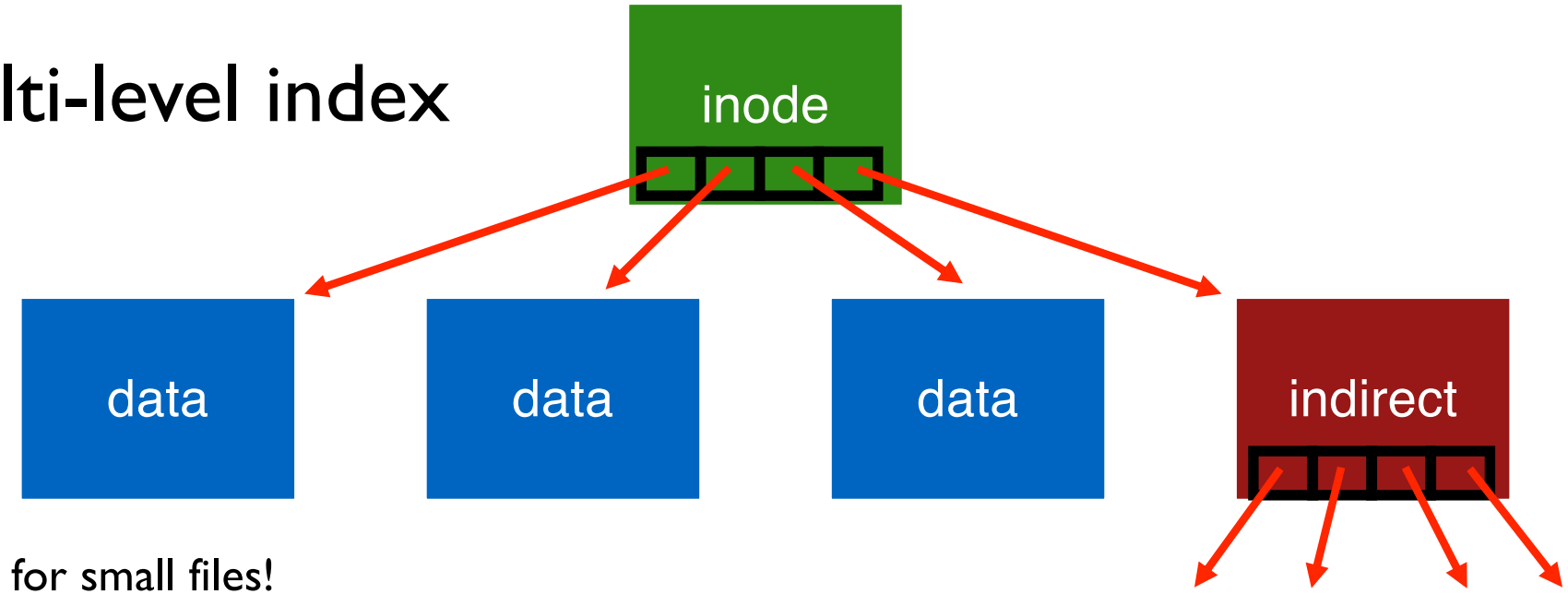
Static calculation to determine where particular inode resides on disk

FFS

- How do you calculate largest file sizes in FFS?


inode
data
data
indirect
Double indirect
x1024
x1024
Better for small files!
How to handle even larger files?
data
data
data
data
indirect
indirect
indirect
indirect
x1024
x1024
x1024
x1024

Multi-level index



Better for small files!
How to handle even larger files?

Double indirect blocks
Triple indirect blocks

Example:
12 direct pointers + single + double indirect block.
Block size of 4 KB and 4-byte pointers

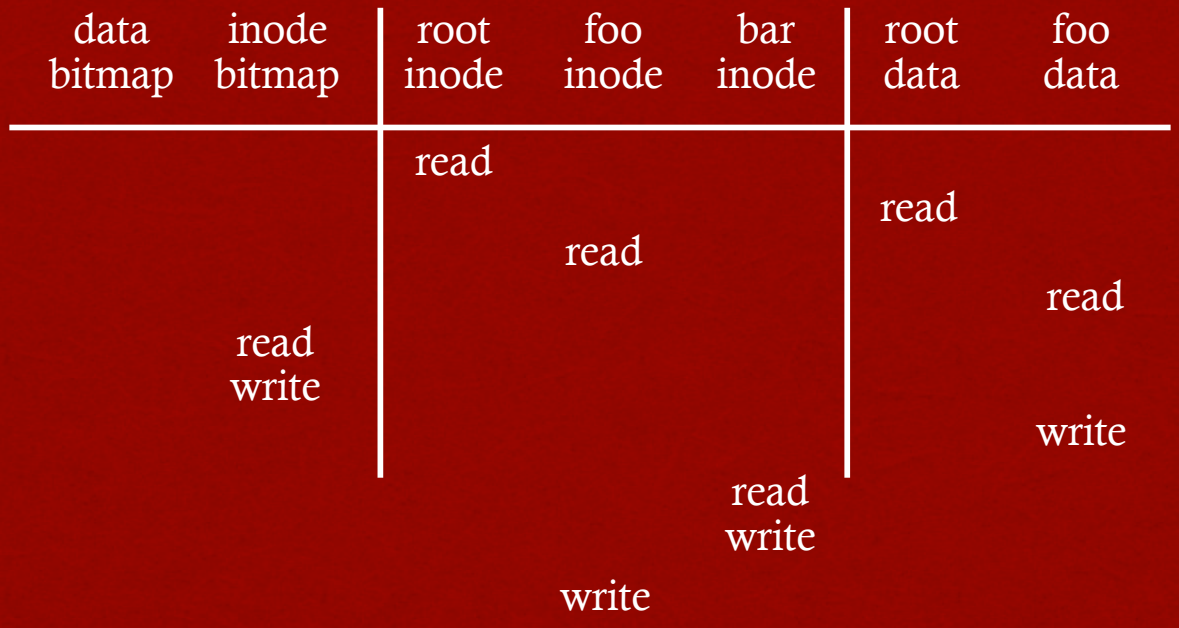
(12 + 1024 + 1024*1024) × 4 KB) = 4 GB

FILE SYSTEM IMPLEMENTATION

- Could you go over the operations needed to create a new file?

- Why does the book vs. slides sometimes switch the order of some of the writes?

create /foo/bar



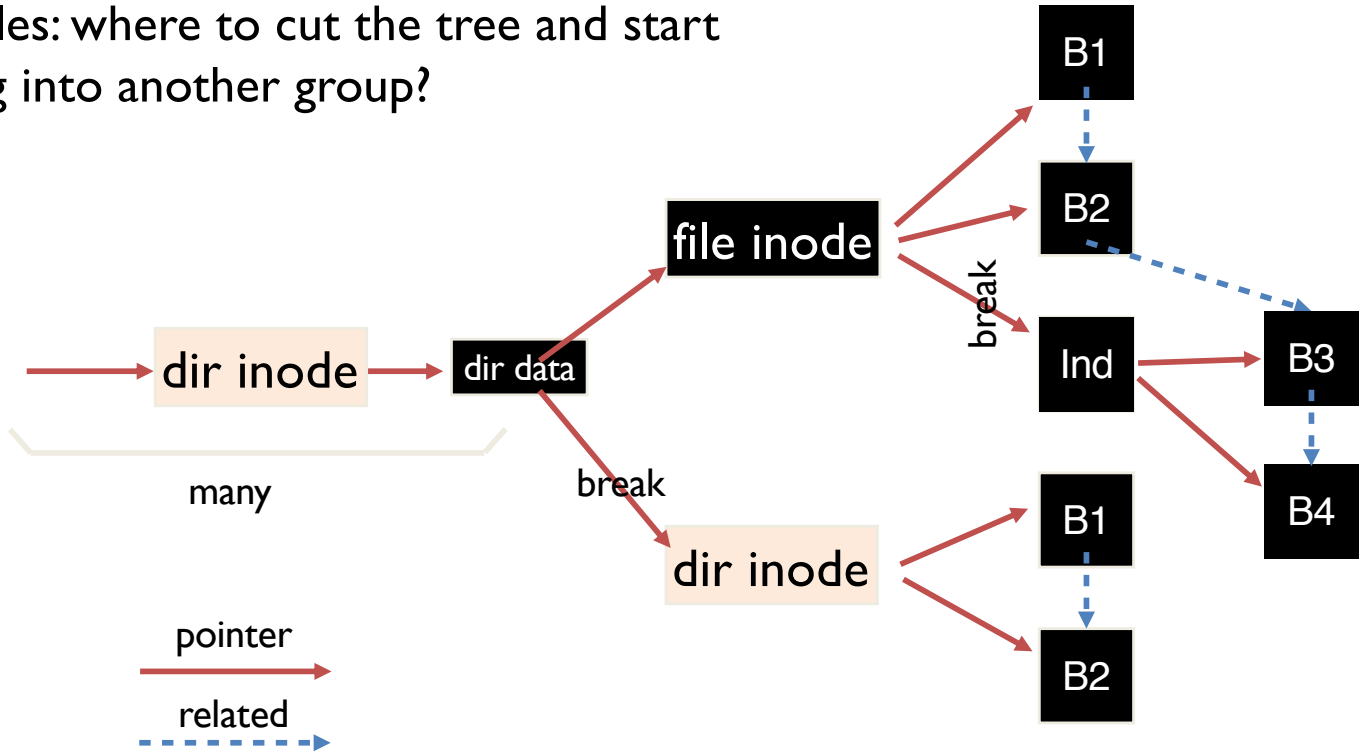
What needs to be read and written?

FFS

- Preferences for building FFS especially for Large File.

  "after 48KB, go to new group. Move to another group (w/ fewer than avg blocks) every subsequent 1MB." –
- - How to determine avg blocks and how to come up to 48KB then move to new group?

Large files: where to cut the tree and start growing into another group?



Define "large" as requiring an indirect block

Starting at indirect (e.g., after 48 KB) put blocks in a new block group

POLICY SUMMARY

File inodes: allocate in same group with dir

Dir inodes: allocate in new group with fewer used inodes than average group

First data block: allocate near inode

Other data blocks: allocate near previous block

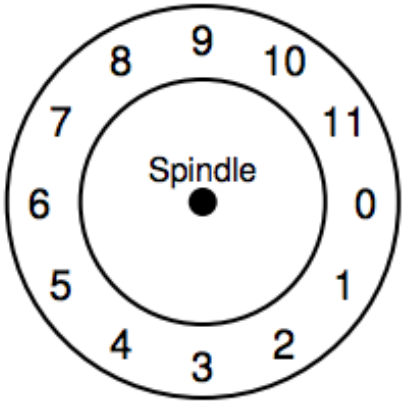
Large file data blocks: after 48KB, go to new group.

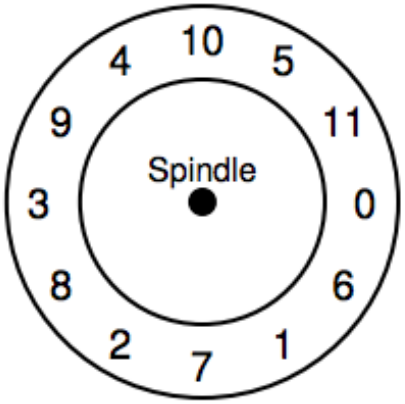
Move to another group (w/ fewer used blocks than avg) every subsequent chunk (e.g., 1MB or 4MB).

FFS SECTORS

- Why do we need sector placement?

FFS: SECTOR PLACEMENT





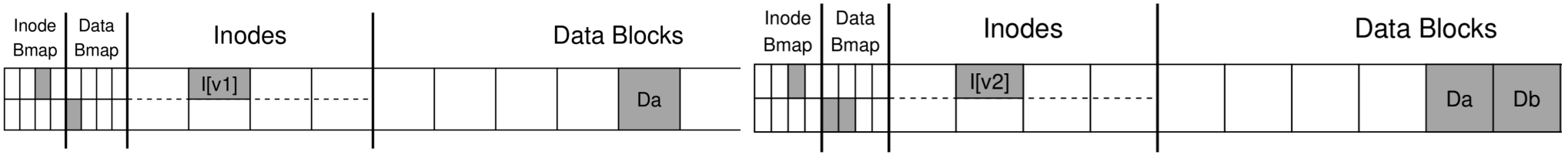
Similar to track skew in disks chapter

Modern disks: Disk cache

JOURNALING

- If both bitmaps and inodes are written (but not data), would that count as a data leak as well? Or just pointing to garbage?

FILE APPEND EXAMPLE

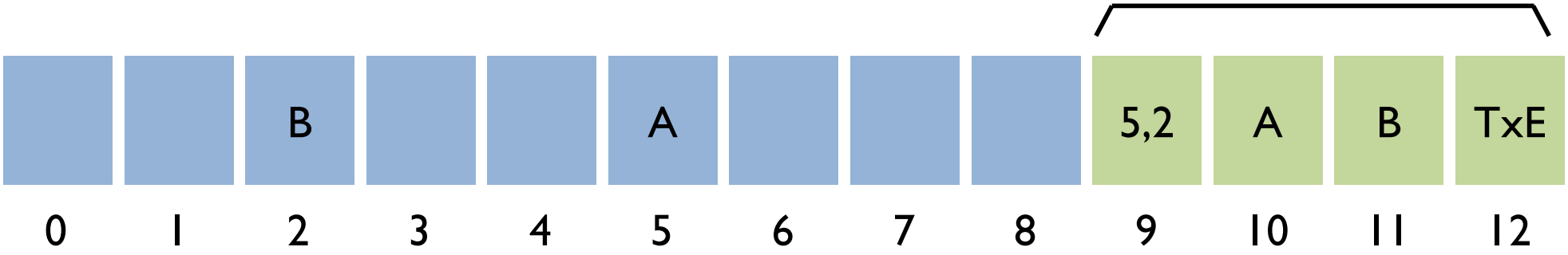


Written to disk	Result
Db	Lost data (nothing bad)
I[v2]	Point to garbage; another file could use data block
B[v2]	Space leak
I[v2] + B[v2]	Point to garbage data
I[v2] + Db	Another file could use same datablock
B[v2] + Db	Space leak

JOURNALING

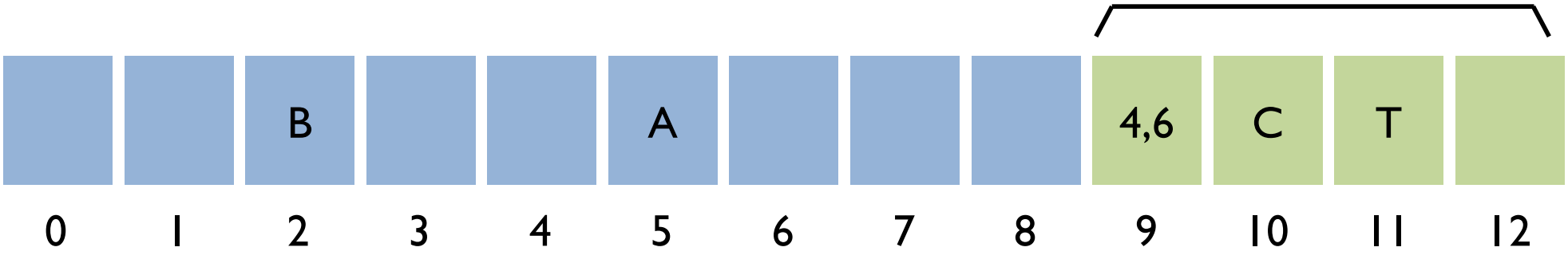
- In part 8b on the exam, isn't recovery ALWAYS run because it's explicitly stated in the question? Sure it might differ between old and new states, but why is there an option for "no recovery is run" in the first place?

JOURNAL REUSE AND CHECKPOINTS



After first transaction checkpoints, can begin next one
Next transaction: write C to block 4; write T to block 6

JOURNAL REUSE AND CHECKPOINTS

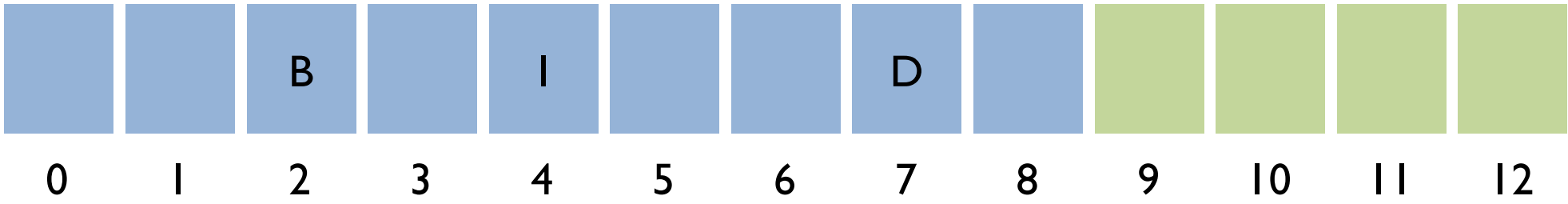


transaction: write C to block 4; write T to block 6

JOURNALING

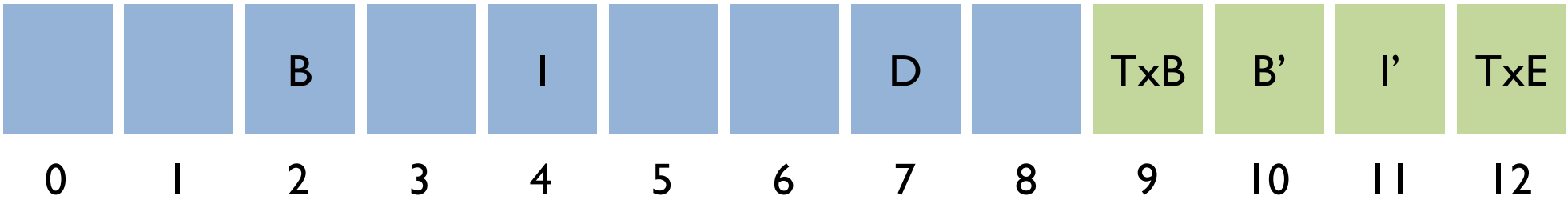
- In metadata journaling, the "data write" is done before "journal metadata write" in order to make sure that the inode will not be pointing to the garbage. However, what if the system crashes right after the "data write" but before the "journal metadata write"? In this case, has the data-bitmap allocated the new data block? (If yes, then this block will be lost? If no, how was the new data able to find its write location?) What does "checkpoint metadata" specifically update, inode, inode-bitmap, data-bitmap?

ORDERED JOURNAL



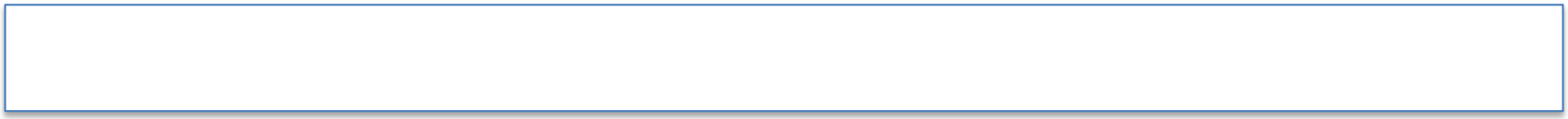
What happens if crash now? (before update B and I)
B indicates D currently free, I does not point to D;
Lose D, but that might be acceptable

ORDERED JOURNALING

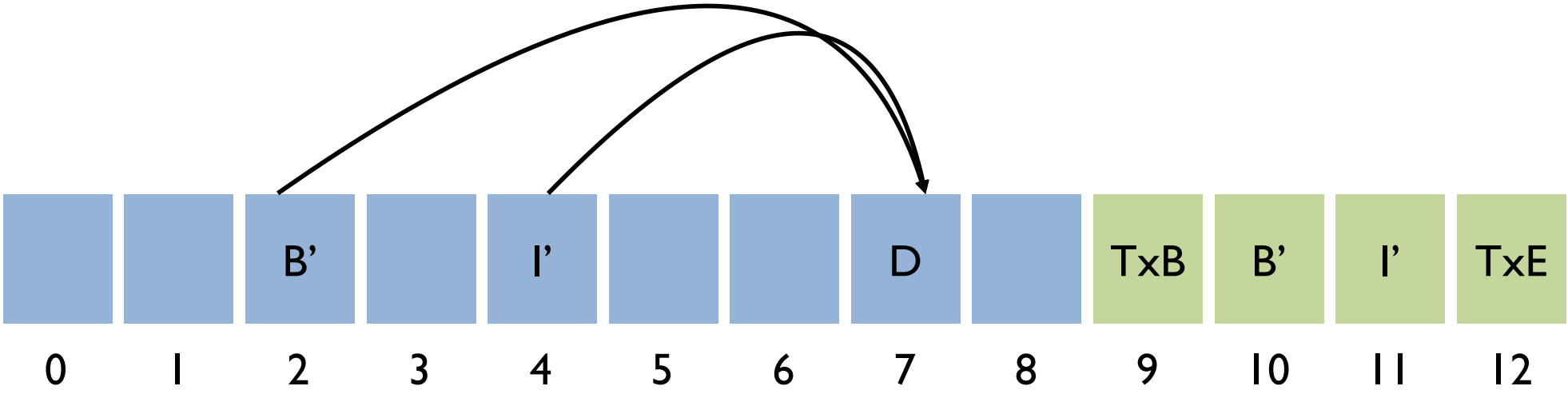


transaction: append to inode I

Crash !?!



ORDERED JOURNALING



transaction: append to inode I
Ensures B’ and I’ are updated atomically AFTER D is on disk

Crash !?!

Everything is all good; if didn’t clear TxE, will replay transaction, extra work but no problems

FLASH

- Would FFS or LFS be better on flash?

WRITE AMPLIFICATION

One write is amplified to many additional writes

Writing one 2KB page may cause:
- `read`, `erase`, and `program` of 256KB block.

Random writes are extremely expensive!
(Sequential could modify all pages in one block at one time…)

Implication for File System:
Would FFS or LFS be better on flash?

Copy-On-Write FS may prevent some expensive random writes

DIST FS

- What messages are sent between client and server in NFS and AFS?

NFS PROTOCOL

Time	Client A	Client B	Server Action?
0	fd = open("file A");		lookup()
10	read(fd, block1);		read
20	read(fd, block2);		read
30	read(fd, block1);	check cache: attr expired getattr(), okay, use local	getattr
31	read(fd, block2);	attr not expired, use local	
40		fd = open("file A");	lookup
50		write(fd, block1); keep local	
60	read(fd, block1);	attr. expired use local data	getattr()
70		close(fd); write bl to server!	write to disk
80	read(fd, block1);	attr expired; get attr. CHANGED FILE - kick out	read()
81	read(fd, block2);	not in cache → read	read()
90	close(fd);		
100	fd = open("fileA");		lookup
110	read(fd, block1);	attr expire; get new attr local ok	getattr
120	close(fd);		

AFS PROTOCOL

Time	Client A	Client B	Server Action?
0	fd = open("file A");		setup callback for A
10	read(fd, block1);		send all of file A
20	read(fd, block2);		
30	read(fd, block1);		
31	read(fd, block2);		
40		fd = open("file A");	setup callback
50		write(fd, block1);	send all of A
60	read(fd, block1);		
70		close(fd);	send back changes of A, break callbacks
80	read(fd, block1);		
81	read(fd, block2);		
90	close(fd);		
100	fd = open("fileA");		
110	read(fd, block1);		
120	close(fd);		