File Systems Part 6

UBI/UBIFS

- UBI (unsorted block images)
  - supports multiple logical volumes on one flash device
  - performs wear-leveling across entire device
  - handles bad blocks
- UBIFS
  - file system layered on UBI
  - it really has a journal (originally called JFFS3)
  - page index kept in flash
    - no need to scan entire file system when mounted
  - compresses files as an option

Wednesday's Quiz

Suppose one used UBI/UBIFS on a disk rather than on a flash drive. Would it still be a usable file system?

a) no
b) yes, but some of what it does would be unnecessary and thus a waste of time
c) yes, and everything it does would be useful, even on a disk

Quiz 1

We’ve discussed the following HDD-based file-system topics:

1) seek and rotational delays
2) mapping file-location to disk-location
3) directory implementations
4) transactions
5) RAID

Which are not relevant for file systems on SSDs?

a) all
b) none
c) just one
d) just two

NTFS

- Journaled
  - normally redo
  - can do redo and undo simultaneously
- "Volume aggregation" options
  - spanned volumes
  - RAID 0 (striping)
  - RAID 1 (mirroring)
  - RAID 5
  - snapshots

Quiz 2

What is the benefit of doing redo and undo journaling simultaneously?

a) It makes the file system twice as reliable as it would be with just one

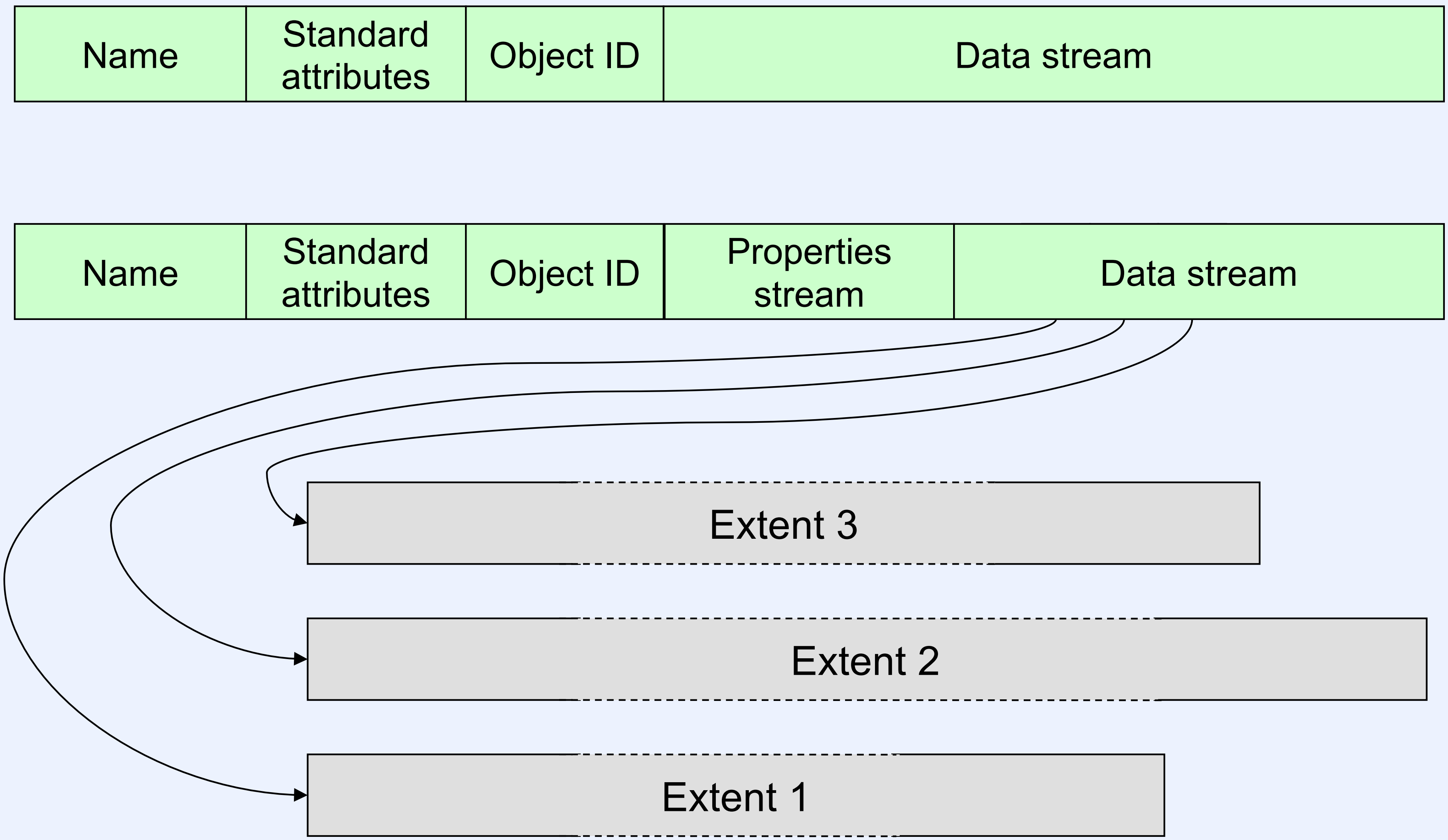
b) Since updates are committed before checkpointing, updates are less likely to be lost in the event of a crash

c) If the OS is running low on RAM, undo journaling makes it possible to reclaim RAM from cached blocks quickly

d) both b and c

e) none of the above

NTFS File Records



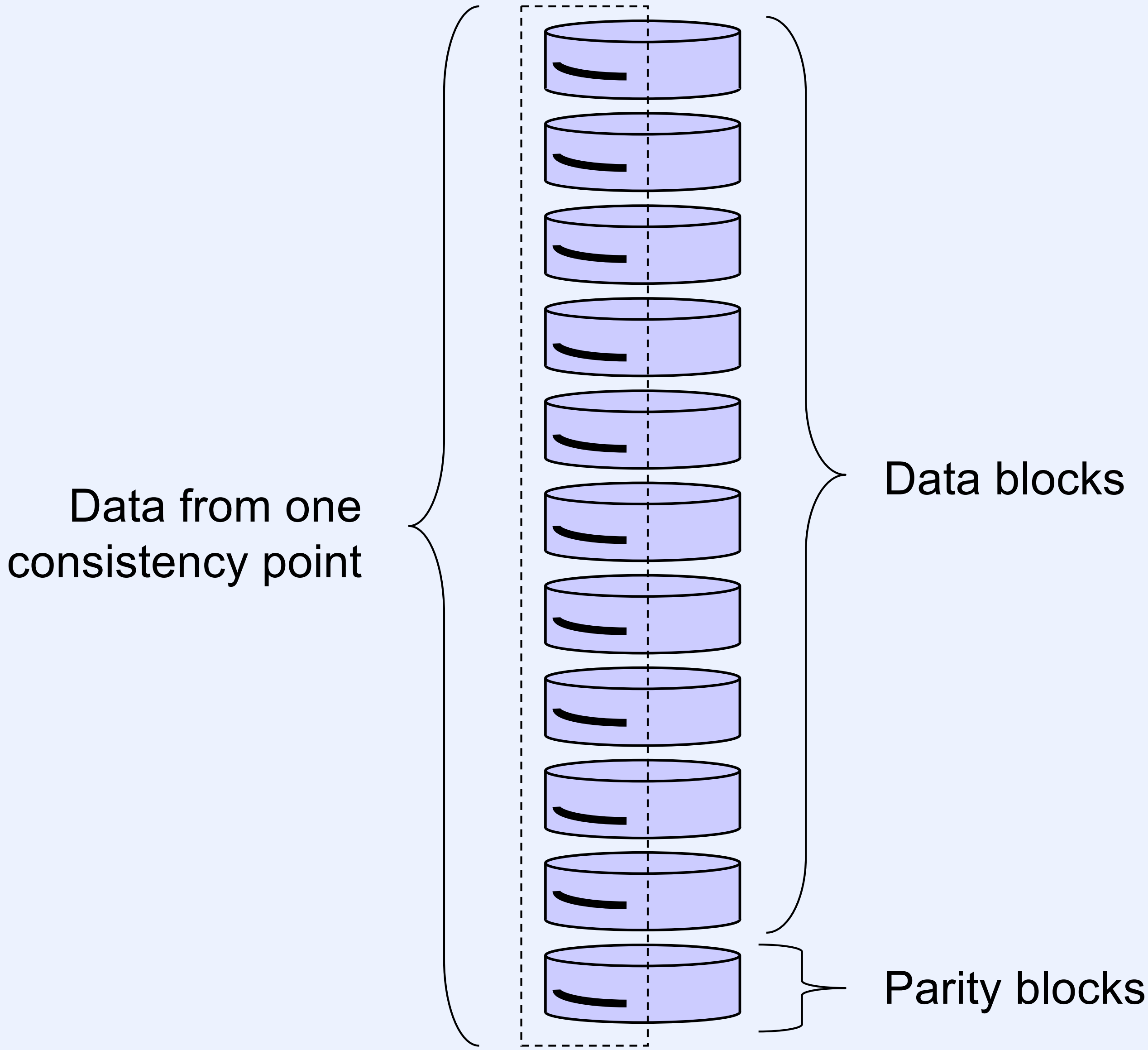
Additional NTFS Features

- Data compression
  - run-length encoding of zeroes
  - compressed blocks
- Encrypted files

WAFL

- Runs on special-purpose OS
  - machine is dedicated to being a *filer*
  - handles both NFS and SMB requests
- Utilizes shadow paging and log-structured writes
- Provides snapshots

WAFL and RAID



Consistency Points … and Beyond

- Consistency points taken every ~10 seconds
  - too relaxed for many applications
    - NFS
    - databases
- Solution …

(battery-backed-up RAM)

(a.k.a. non-volatile RAM (NVRAM))

Quiz 2

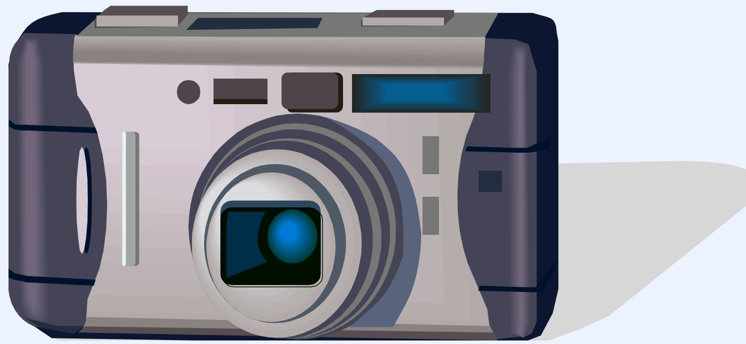
We have an 8-disk RAID 4 system (with a single parity disk). One of the disks goes bad – its entire contents are lost. Can we recover its contents using nothing but the contents of all the other disks?

a) no

b) yes

c) it depends upon whether the bad disk was the parity disk or was a data disk

Snapshots

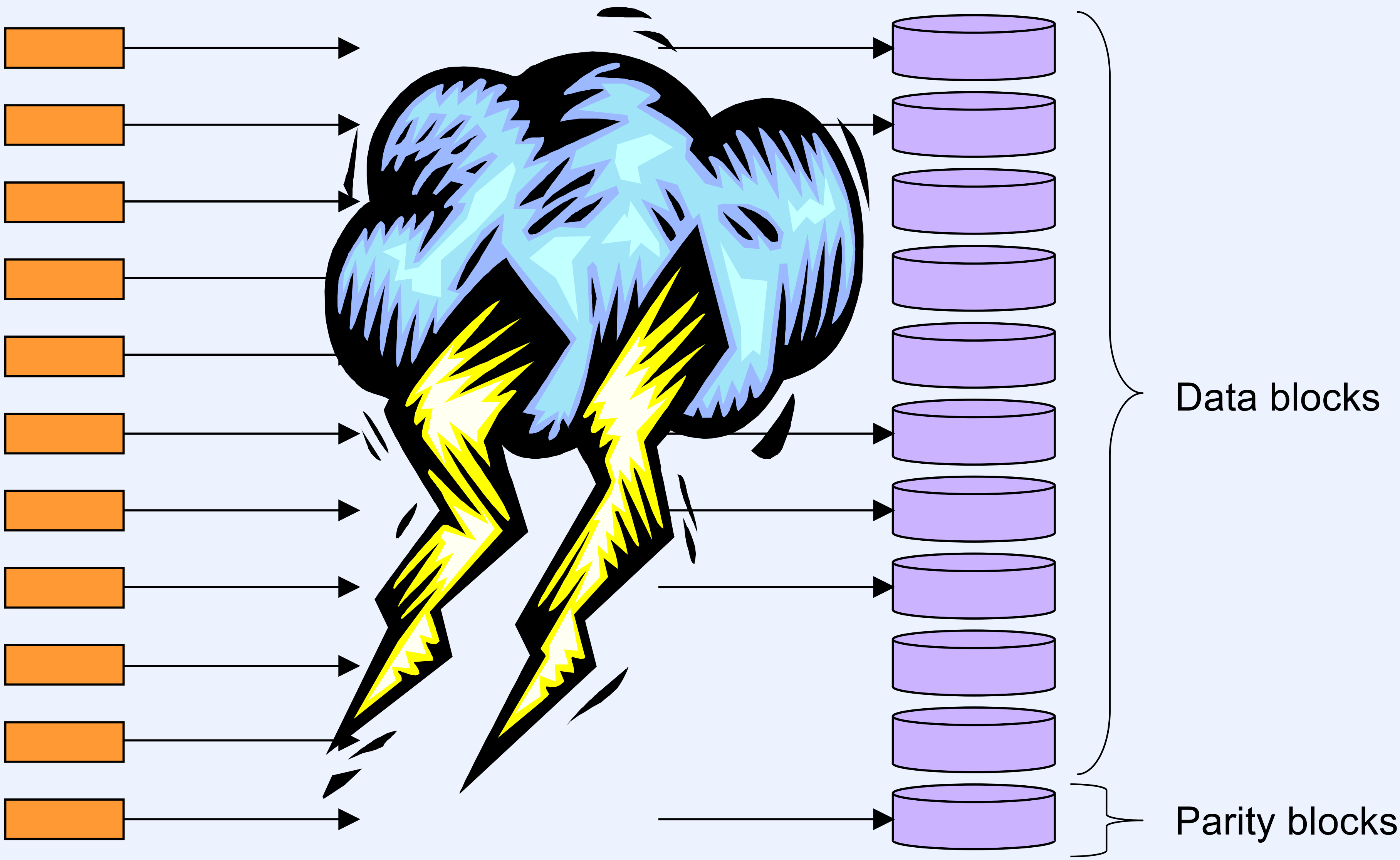


- Periodic snapshots kept of file system
  – made easy with shadow paging

Paranoia

- You think your files are safe simply because they're on a RAID-4 or RAID-5 system …
  - power failure at inopportune moment
    - parity is irreparably wrong
  - obscure bug in controller firmware or OS
    - data is garbage (but with correct parity!)
  - sysadmin accidentally scribbled on one drive
    - (profuse apologies …)
  - out of disk space
    - must restructure 16TB file system
  - out of address space
    - 2^{64} isn't as big as it used to be

Partial Writes



Quiz 3

NTFS employs journaling. Is it susceptible to the problem described in the previous slide? (Hint: it's possible that parts of a transaction might be written in a single stripe.)

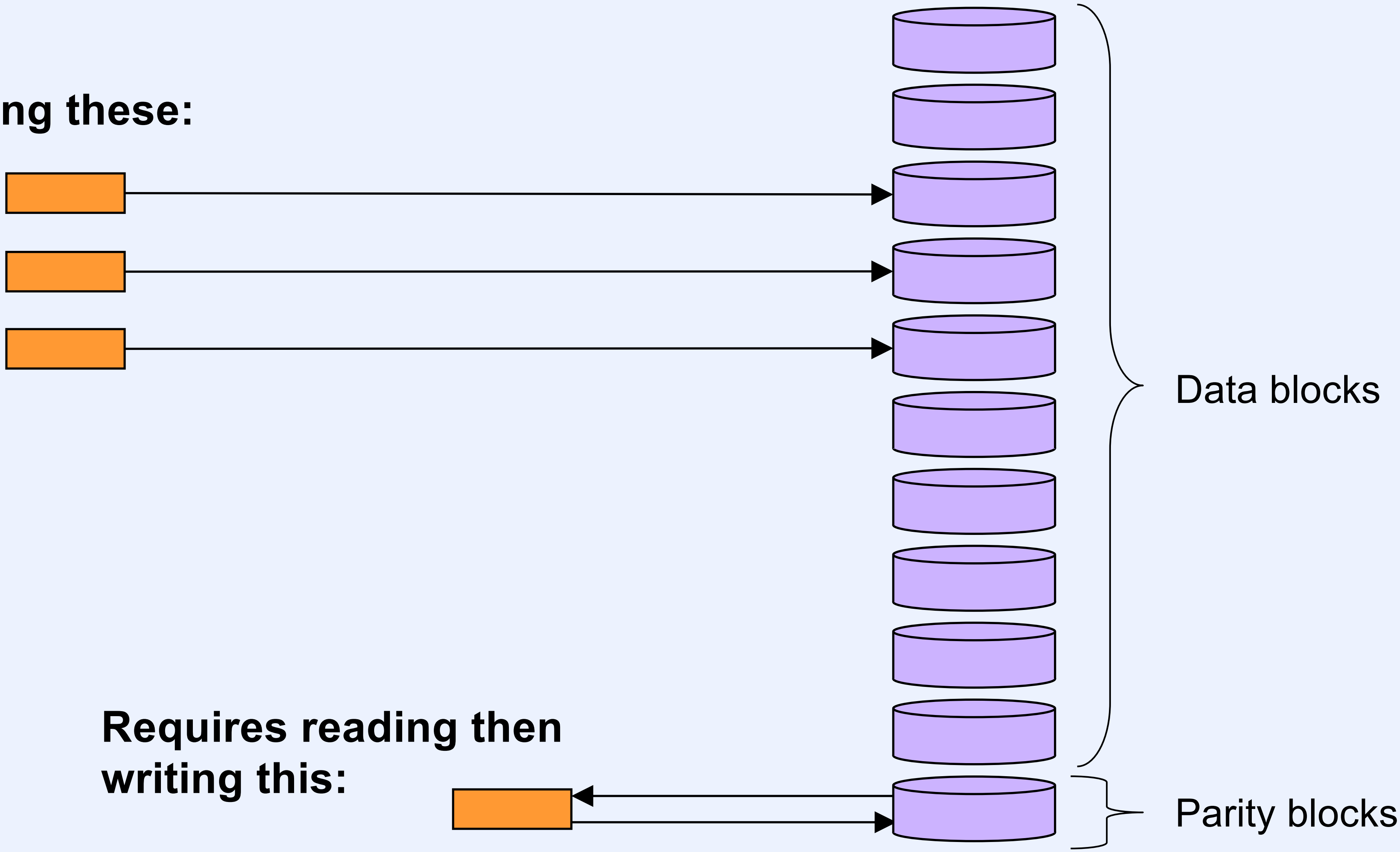
a) no

b) yes

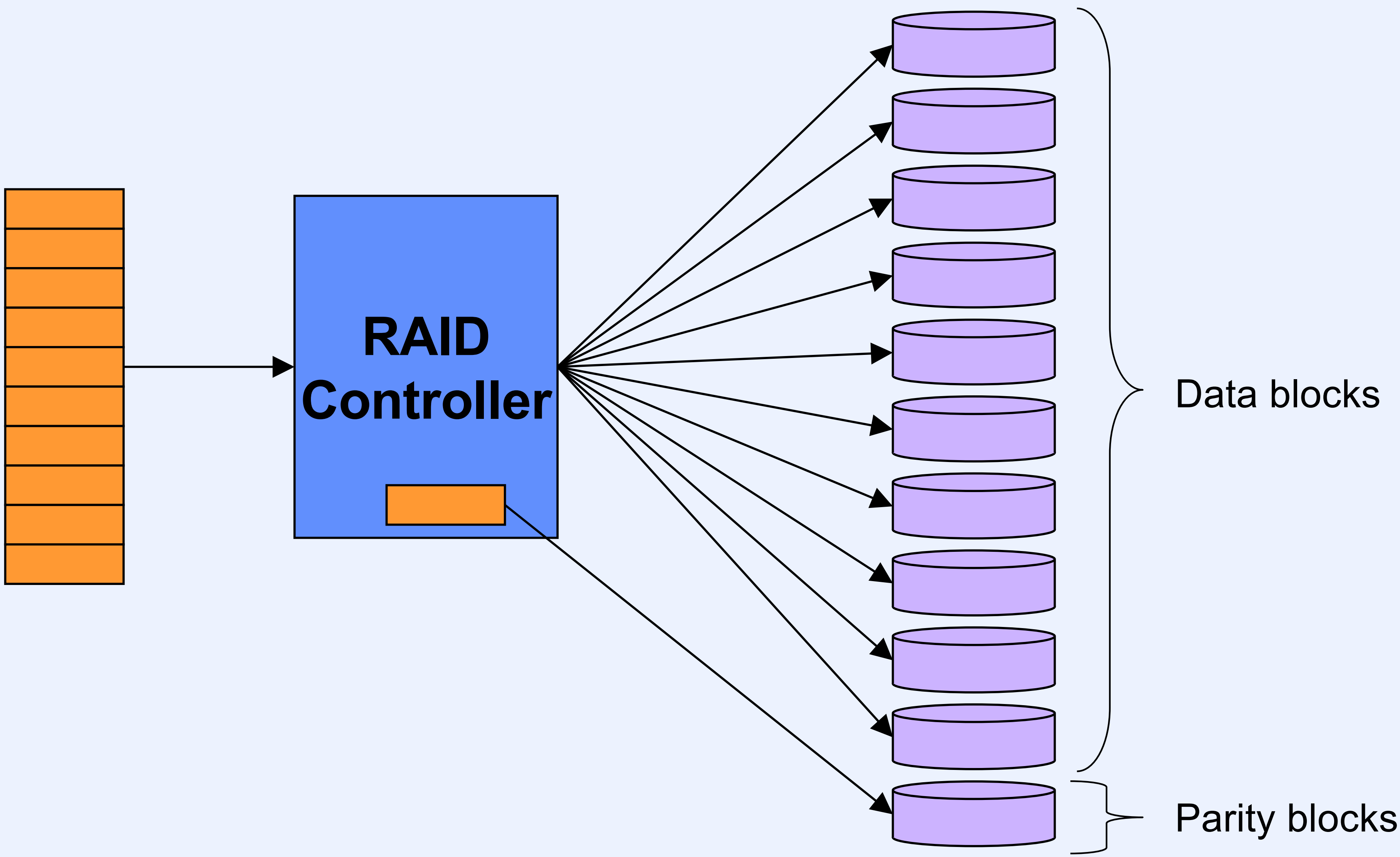
Small Writes

Writing these:

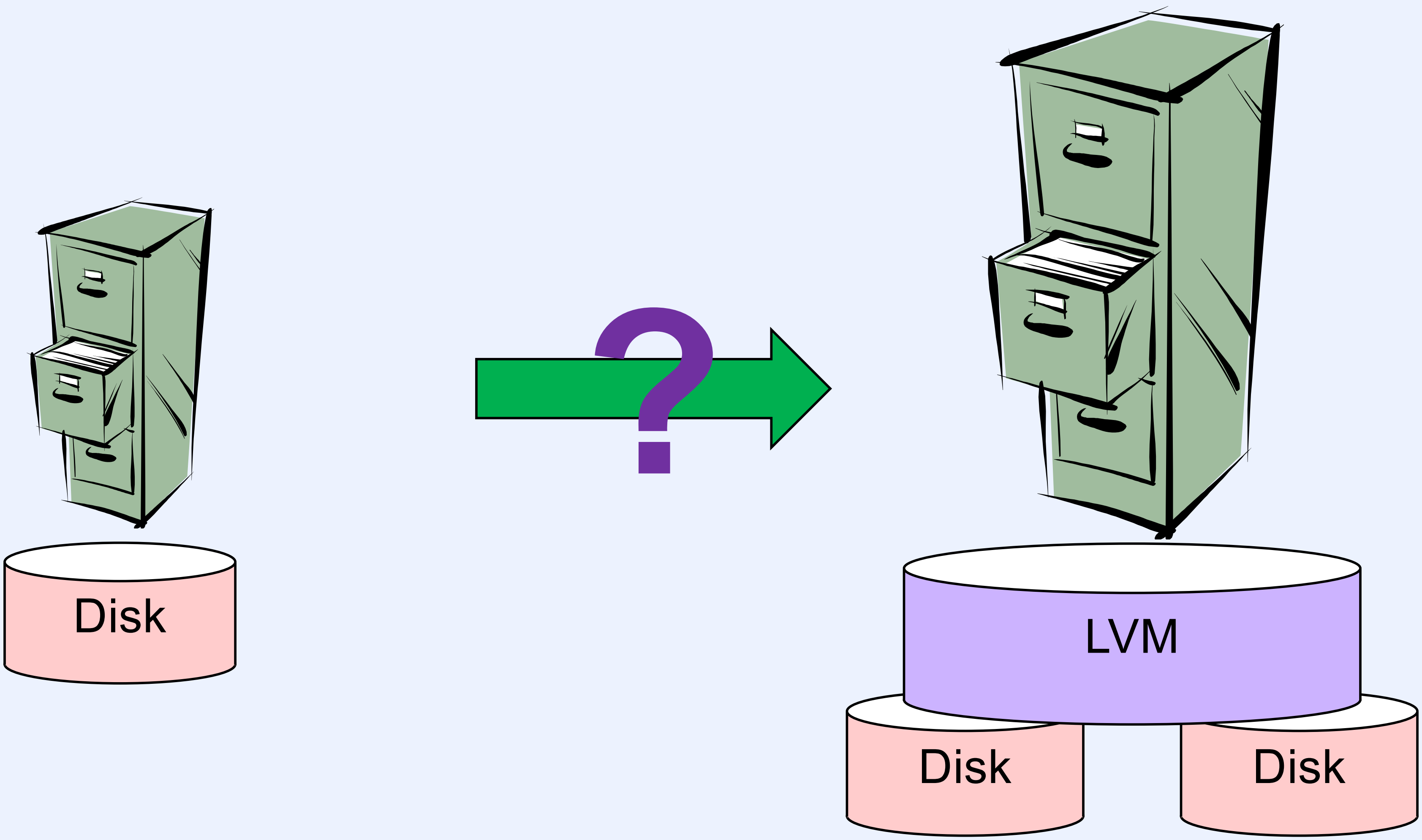
Requires reading then writing this:



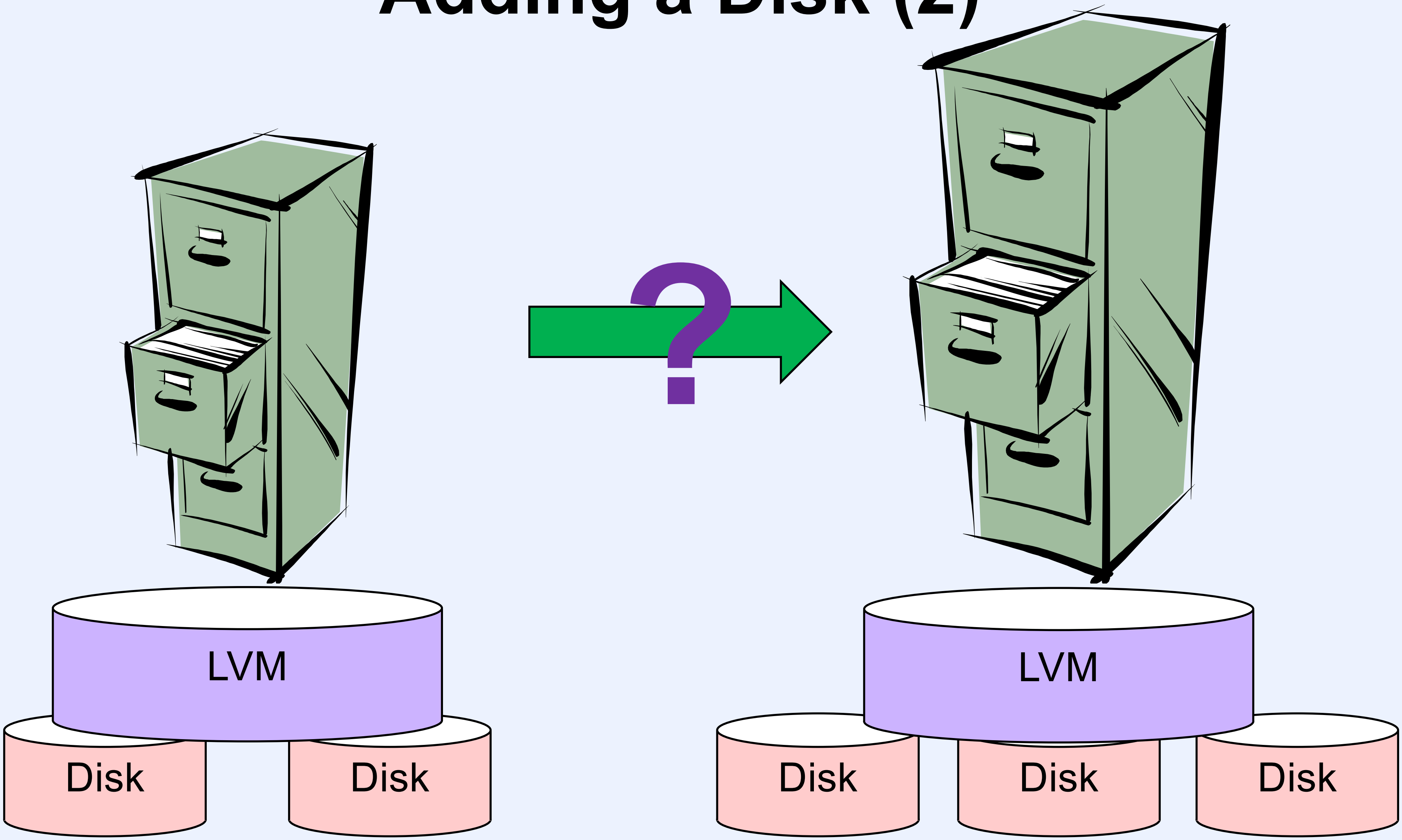
Hardware RAID



Adding a Disk (1)



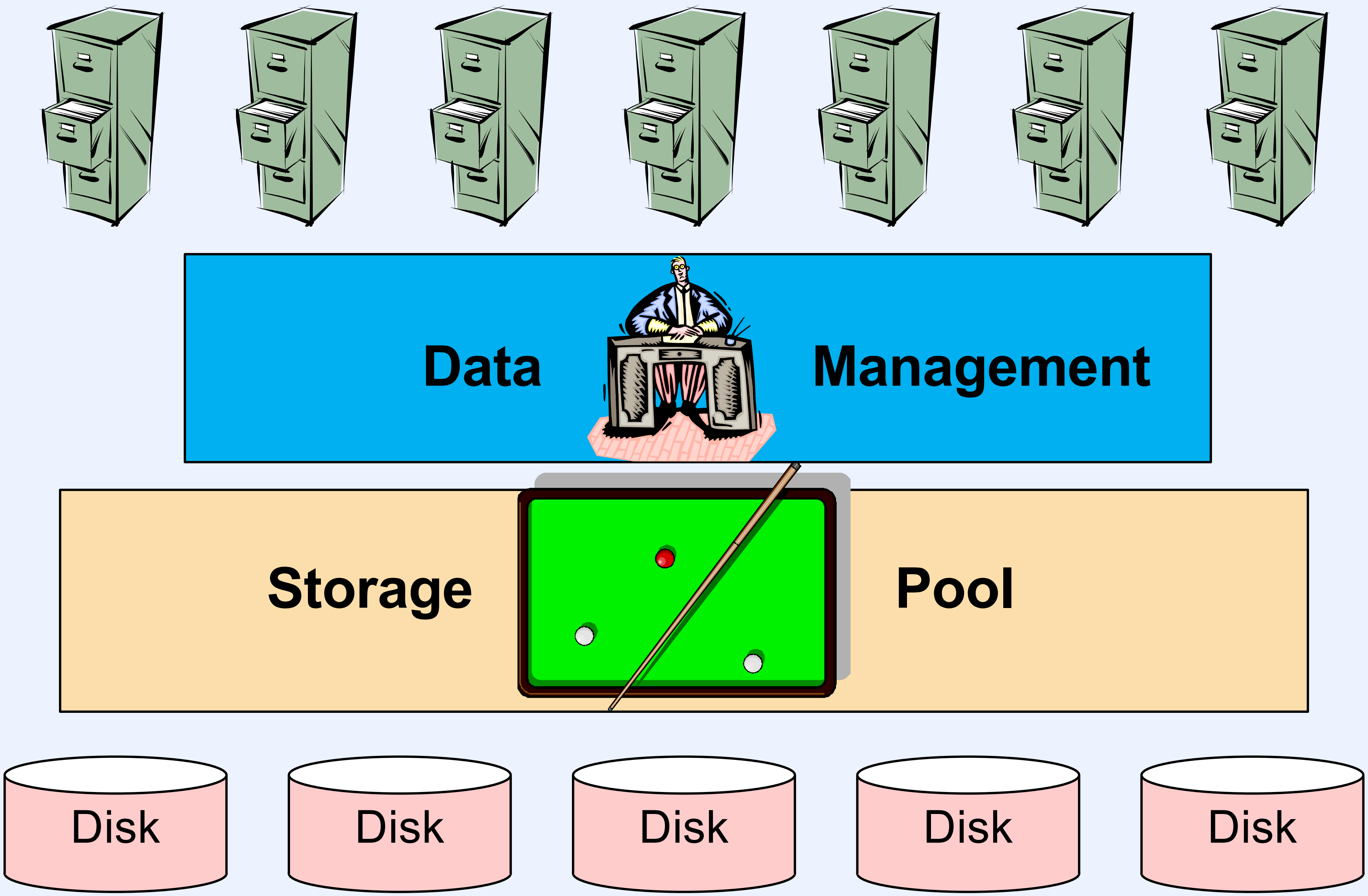
Adding a Disk (2)



ZFS

The Last (?!) Word in File Systems

ZFS Layers



Enter ZFS

VFS

Vnode operations

ZFS POSIX Layer (ZPL)

<dataset, object, offset>

2^{64} objects; each up to 2^{64} bytes

Provides transactions on objects

Data Management Unit (DMU)

<data virtual address>

128-bit addresses!

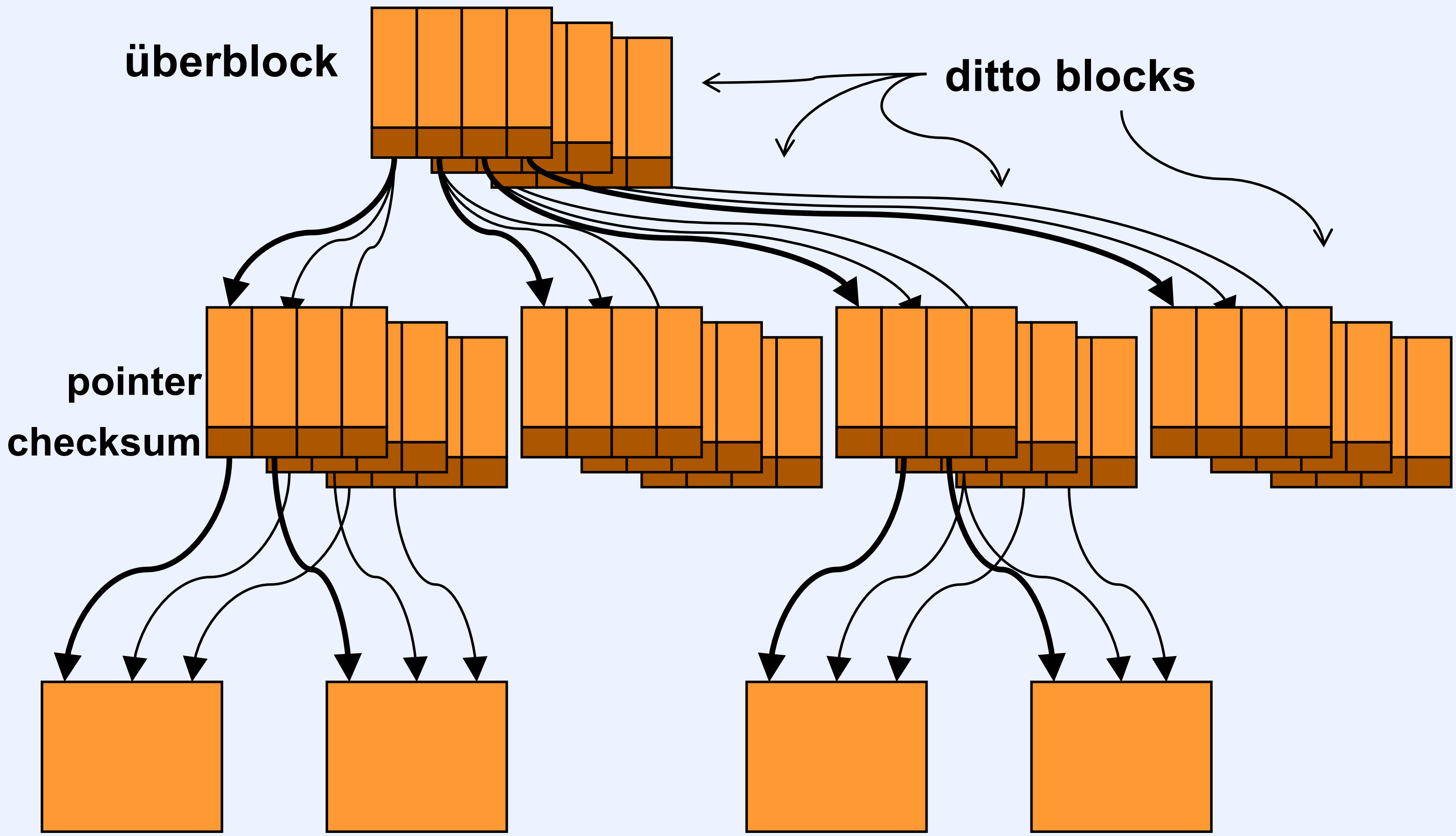
Maps virtual blocks to disks and physical blocks

Storage Pool Allocator (SPA)

<physical device, offset>

Device Driver

Shadow-Page Tree
(with a twist …)



Storage Pool Allocator

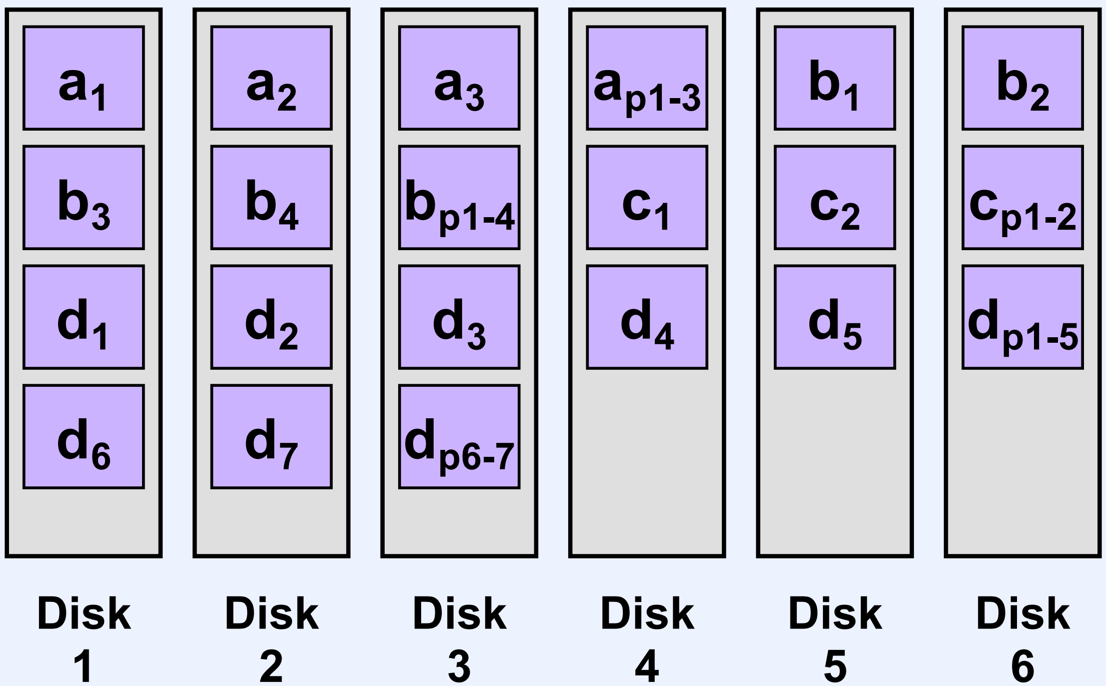
Data Management Unit (DMU)

Storage Pool Allocator

Mirroring, spanning, or RAID

RAID-Z

Software Dynamic Striping



Quiz 4

Compared with RAID 4, which of the following would be more time-consuming with RAID-Z?

a) adding a disk

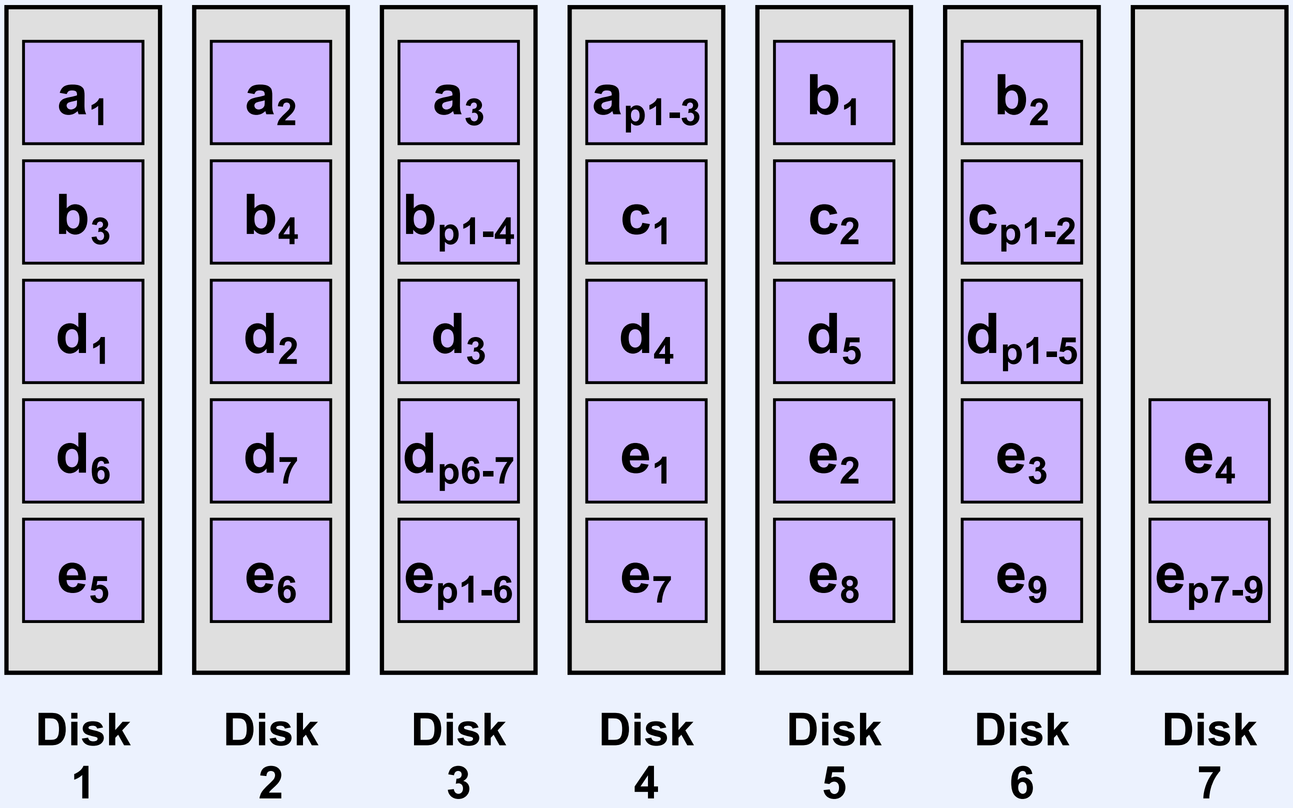
b) replacing a crashed disk

c) both

d) neither

RAID-Z

Adding a Disk



Scenarios

- Power failure at inopportune moment
  - "live data" is not modified
  - single lost write can be recovered
- Obscure bug in controller firmware or OS
  - detected by checksum in pointer
- Sysadmin accidentally scribbled on one drive
  - detected and repaired
- Out of disk space
  - add to the pool; SPA will cope
- Out of address space
  - 2^{128} is big
    - 1 address per cubic yard of a sphere bounded by the orbit of Neptune

And There’s More …

- Adaptive replacement cache
- Advanced prefetching

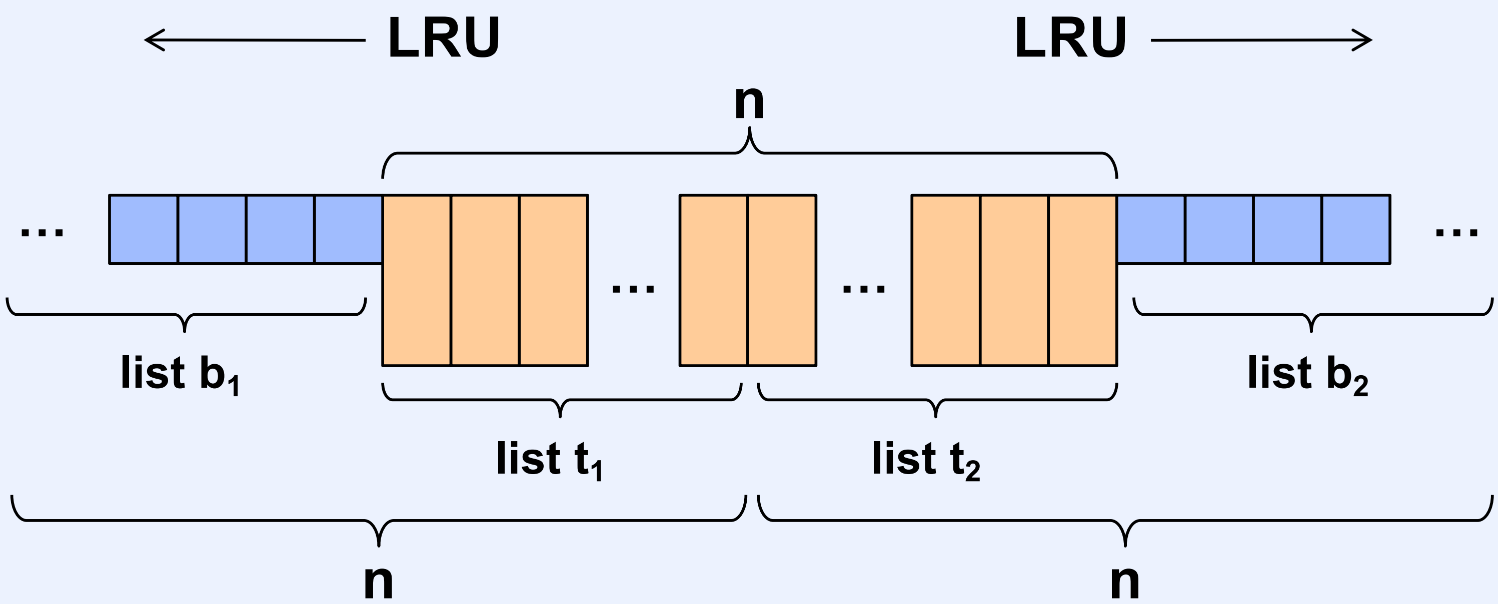
LRU Caching

- LRU cache holds n least-recently-used disk blocks
  - working sets of current processes
- New process reads n-block file sequentially
  - cache fills with this file’s blocks
  - old contents flushed
  - new cache contents never accessed again

(Non-Adaptive) Solution

- Split cache in two
  - half of it is for blocks that have been referenced exactly once
  - half of it is for blocks that have been referenced more than once
- Is 50/50 split the right thing to do?

Adaptive Replacement Cache



t_1 ; b_1:

LRU list of blocks referenced once

t_1 list (most recently used) contain contents

b_1 list (least recently used) contain just references

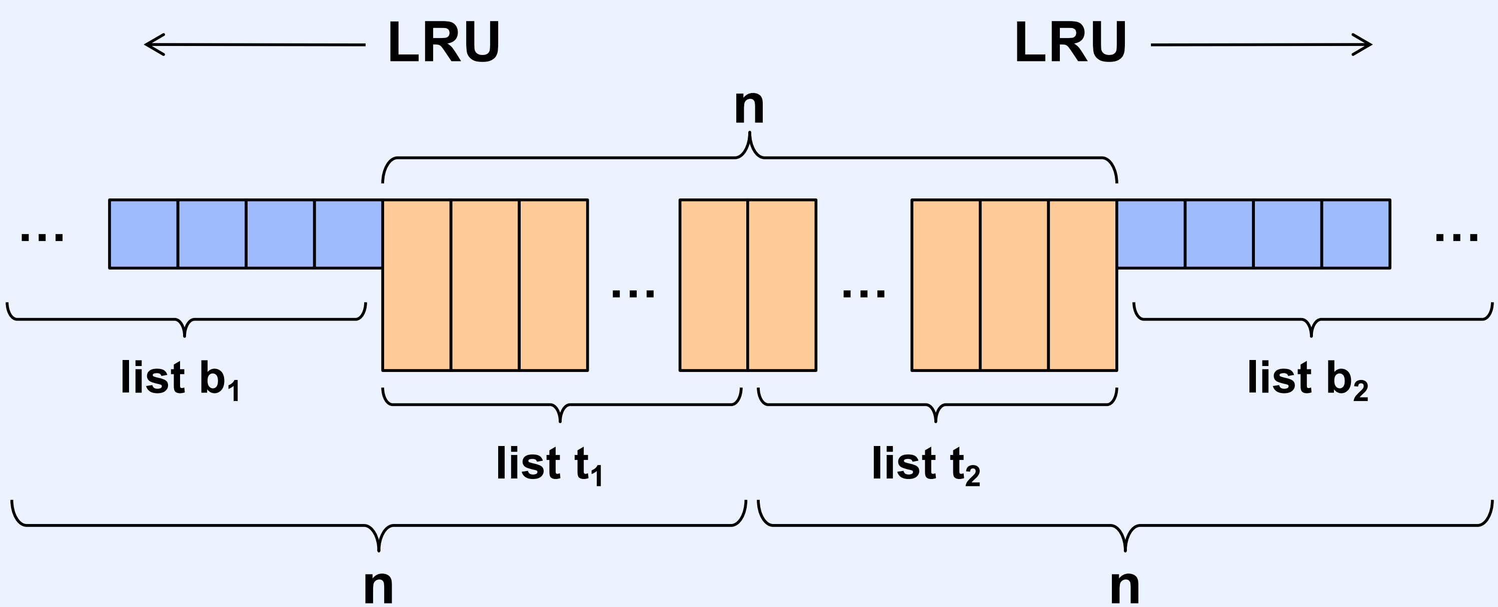
t_2 ; b_2:

LRU list of blocks referenced more than once

t_2 list (most recently used) contain contents

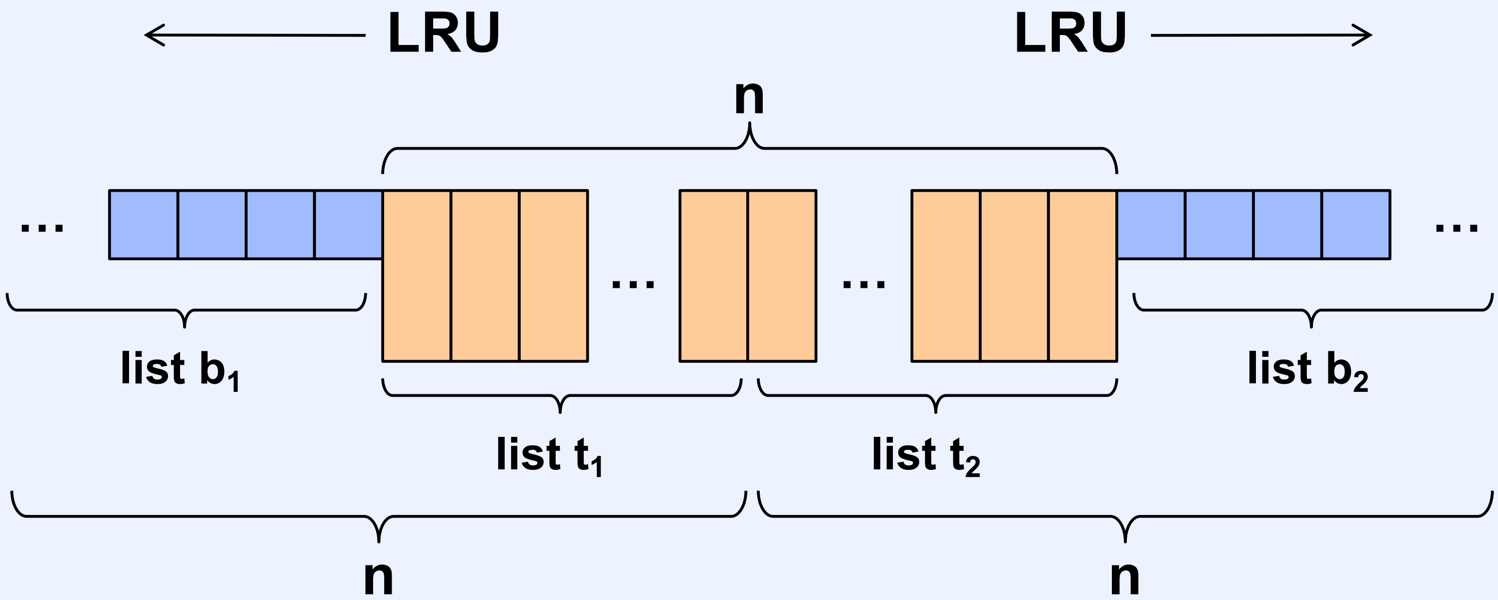
b_2 list (least recently used) contain just references

Adaptive Replacement Cache



cache miss:
   if t_1 is full
         evict LRU(t_1) and make it MRU(b_1)
   referenced block becomes MRU(t_1)

Adaptive Replacement Cache



cache hit:

- if in t_1 or t_2, block becomes MRU(t_2)
- otherwise
  - if block is referred to by b_1, increase t_1 space at expense of t_2
  - otherwise (referred to by b_1)
    - increase t_2 space at expense of t_1
  - if t_1 is full, evict LRU(t_1) and make it MRU(b_1)
  - if t_2 is full, evict LRU(t_2) and make it MRU(b_2)
  - insert block as MRU(t_2)

Quiz 5

Lists b_1 and b_2 do not contain cached blocks, but just their addresses. Why are they needed?

a) So that one can determine how much better things would be if the cache were twice as large

b) As placeholders so that when these blocks are read in, it's known where in the cache they would go

c) So that we would know, if the addressed block is referenced, whether it would have been in the cache if the corresponding t list were larger

d) They are used by the file system to help determine block reference patterns

Prefetch

- FFS prefetch
  – keeps track of last block read by each process
  – fetches block i+1 if current block is i and previous was i-1
  – chokes on
    - diff file1 file2

zfetch

- Tracks multiple prefetch streams
- Handles four patterns
  - forward sequential access
  - backward sequential access
  - forward strided access
    - iterating across columns of matrix stored by columns
  - backward strided access