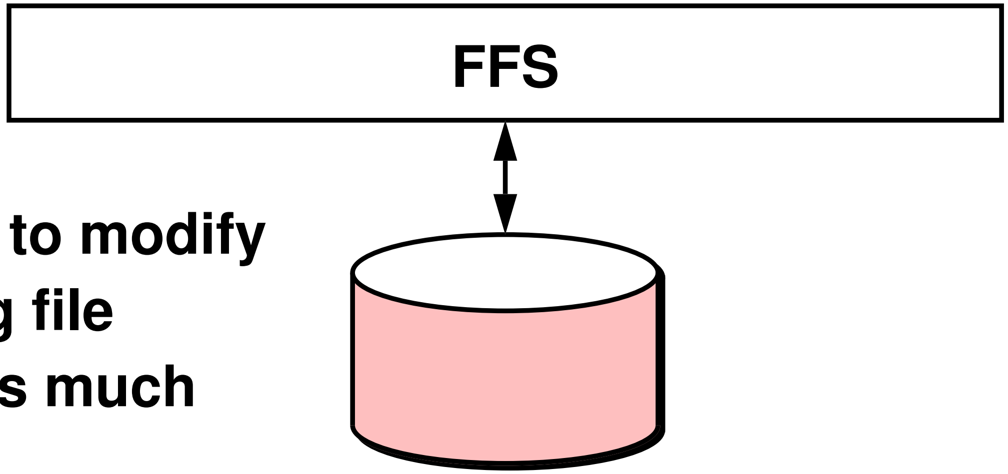
6.4 Multiple Disks

- RAID
  - what can be done if you lose an entire disk?

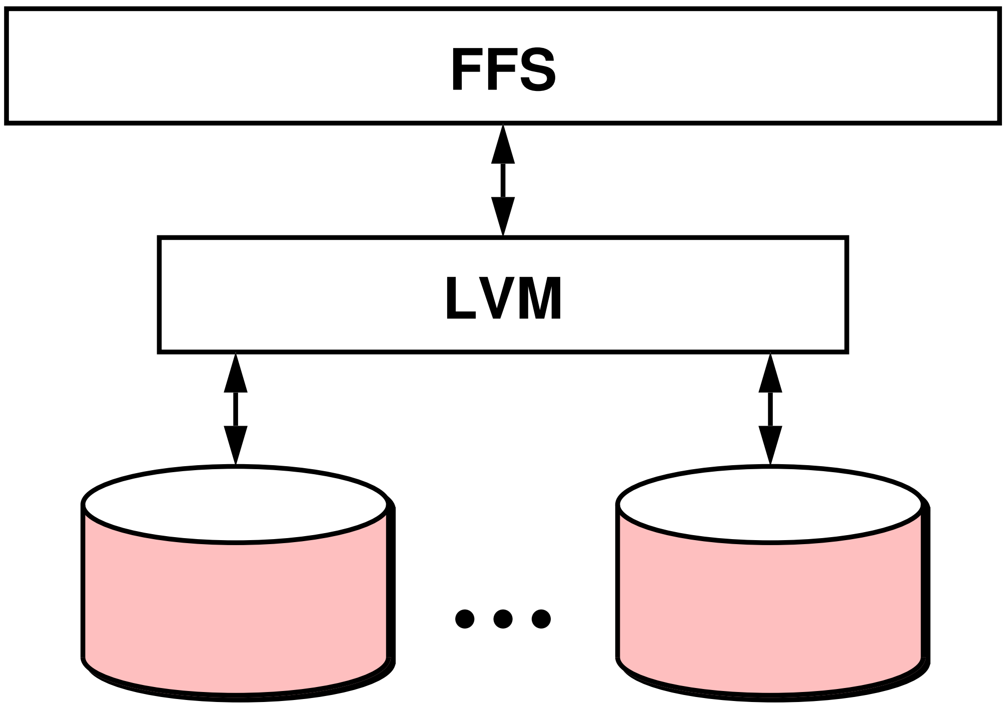
Benefits of Multiple Disks

- They hold more data than one disk does
- Data can be stored *redundantly* so that if one disk fails, they can be found on another
  - increase reliability
  - increase availability
- Data can be spread across multiple drives, allowing *parallel* access
  - increase effective access time

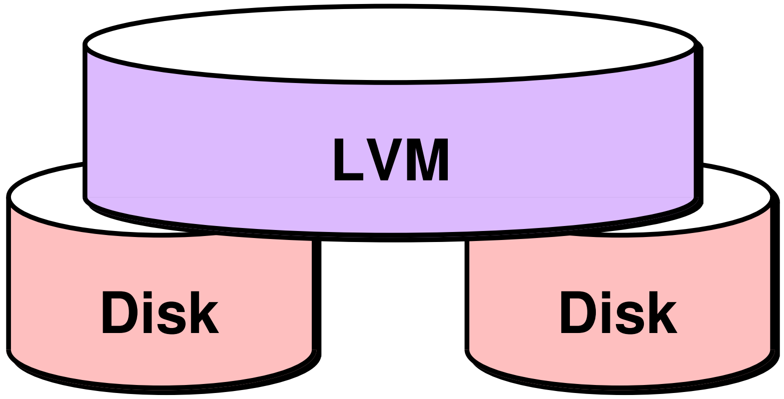
Logical Volume Manager



- Try not to modify existing file systems much



Logical Volume Manager



- **Spanning**
  - two real disks appear to file system as one large disk
- **Mirroring**
  - file system writes redundantly to both disks
  - reads from one

Striping

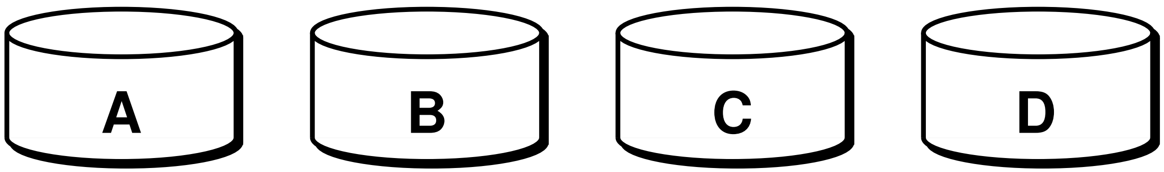
- Ex: ***stripe width*** = 4

	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
Stripe 1	Unit 1	Unit 2	Unit 3	Unit 4	
Stripe 2	Unit 5	Unit 6	Unit 7	Unit 8	
Stripe 3	Unit 9	Unit 10	Unit 11	Unit 12	
Stripe 4	Unit 13	Unit 14	Unit 15	Unit 16	
Stripe 5	Unit 17	Unit 18	Unit 19	Unit 20	

- theoretically, a ***striping unit*** can be a bit *(i.e., **bit-interleaving**)*
  - pack these bits into disk blocks and store on disk

Parallel Disks

- Advantages
  - increase parallelism
    - can retrieve blocks belonging to multiple files simultaneously if they are on different disks

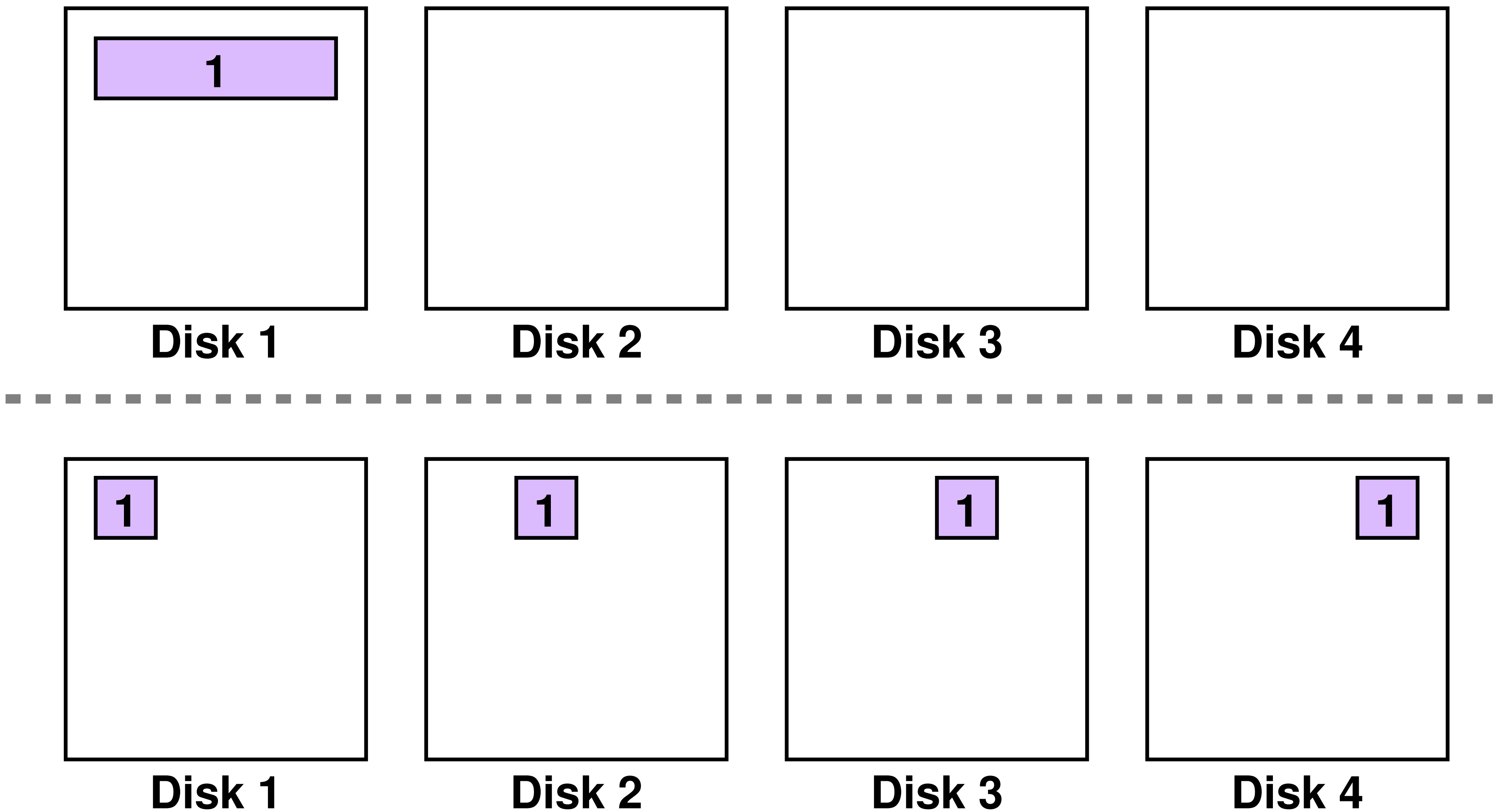


    - reduced access time if a block is spread over multiple disks
      - seek in parallel, same rotational latency on all disks
- Disadvantages
  - higher variance
    - average is just part of the story
  - worse reliability
  - heterogenious disks

How To Stripe?

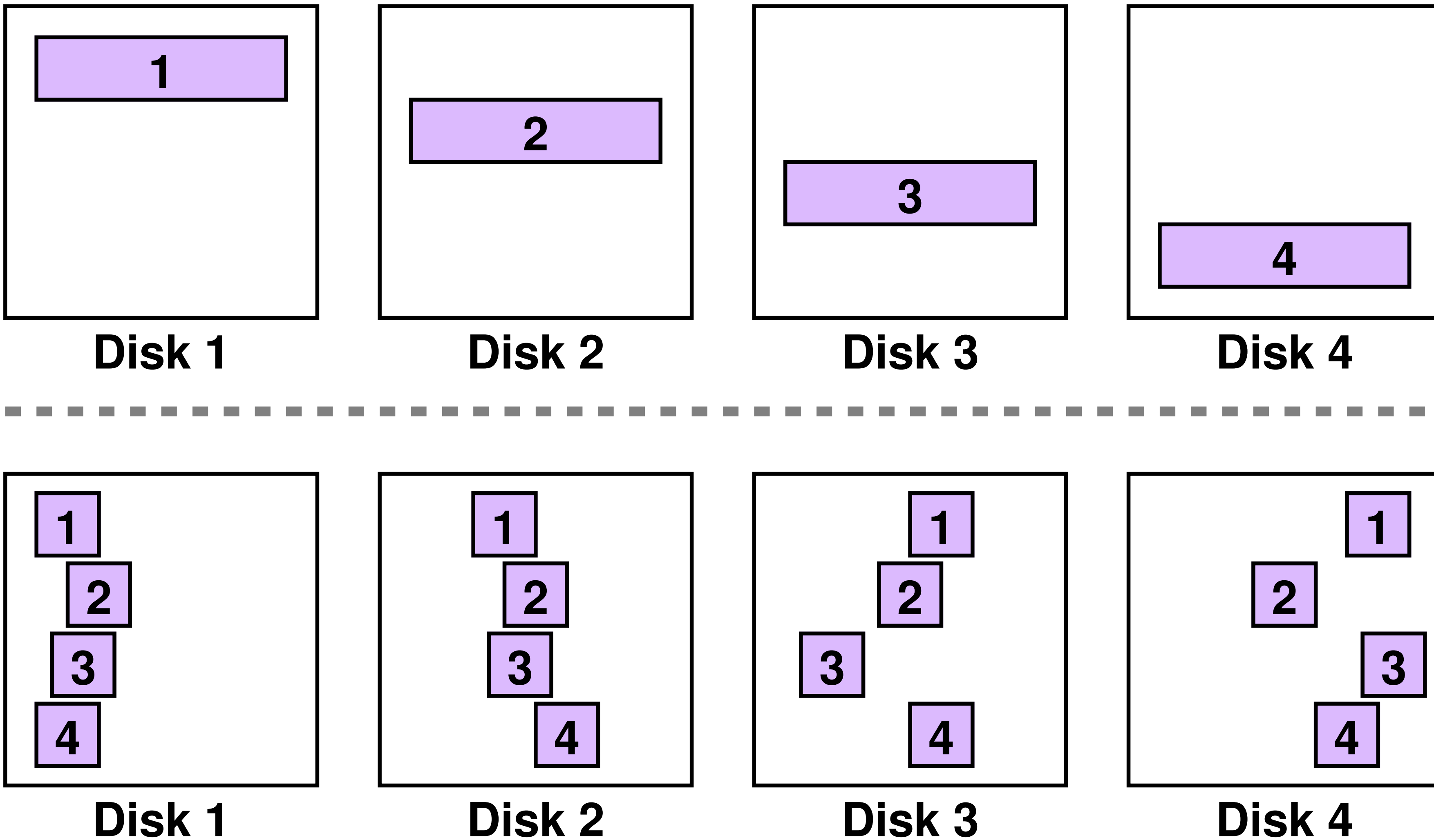
- How to stripe?
  - what's the best *stripe width* (i.e., across how many disks)?
  - how large should be the *striping unit* (i.e., how much to put on one disk before moving to the next disk)?
- Concurrency Factor: how many requests are available to be executed at once?
  - one request in queue at a time
    - concurrency factor = 1
    - e.g., one single-threaded application placing one request at a time
  - many requests in queue
    - concurrency factor > 1
    - e.g., multiple threads placing file-system requests
  - the larger the concurrency factor, the less important striping is
    - in general, performance is better with *larger striping unit*

Striping Unit Size



- **Bottom solution seems better**
  - **data transfer time is 1/4 of the solution on the top**

Striping Unit Size



- The above two cases have *same transfer time*
  - can *reduce seek time* by using a *larger striping unit*

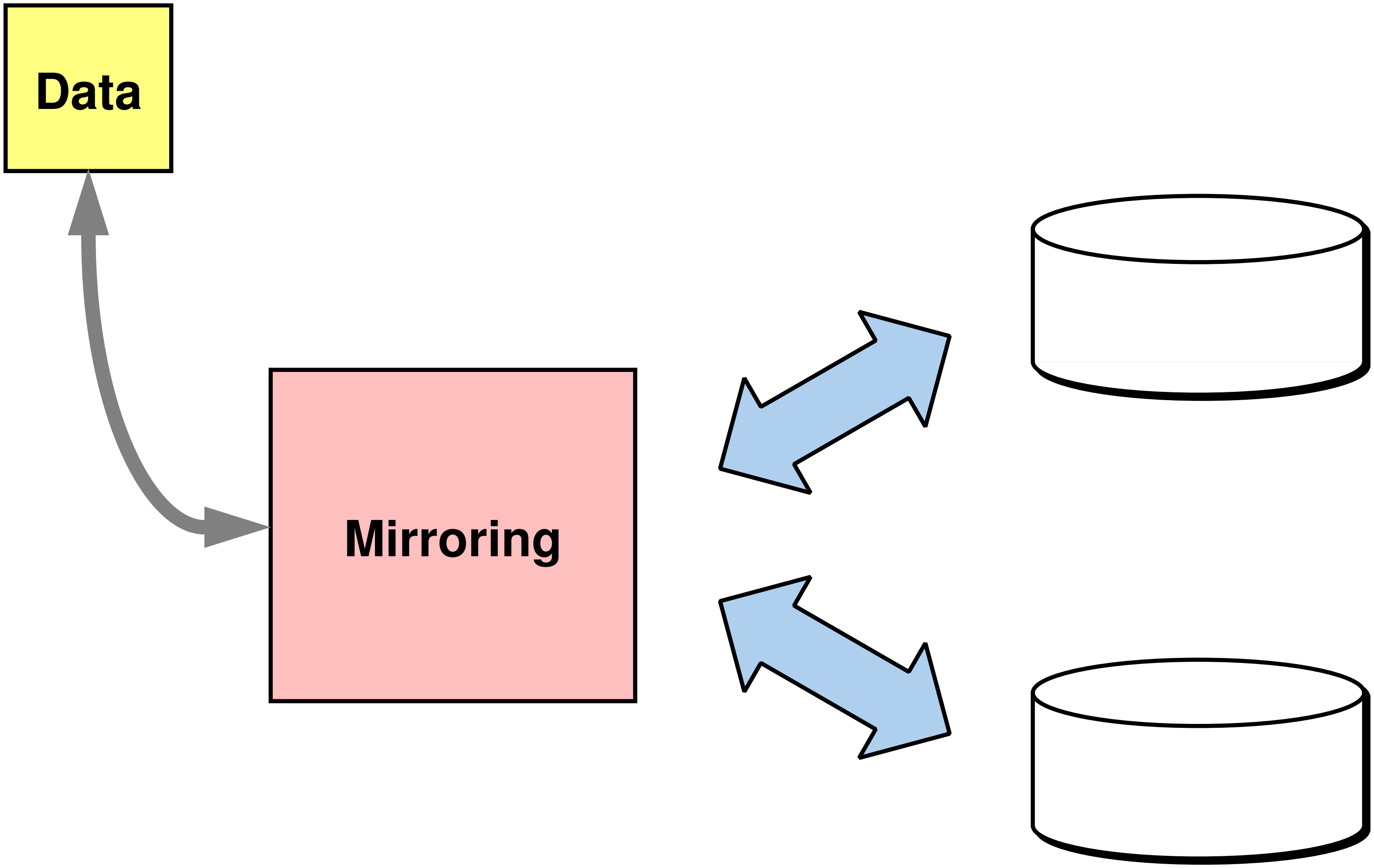
Striping: The Effective Disk

- Improved effective transfer speed
  - parallelism
- No improvement in seek and rotational delays
  - sometimes worse
- A system depending on N disks is much more likely to fail than one depending on one disk
  - if probability of one disk failing is f
  - probability of N-disk system failing is $(1 - (1 - f)^N)$
    - assumes failures are i.i.d., which is probably wrong ...
      - *i.i.d.:* independent and identically distributed

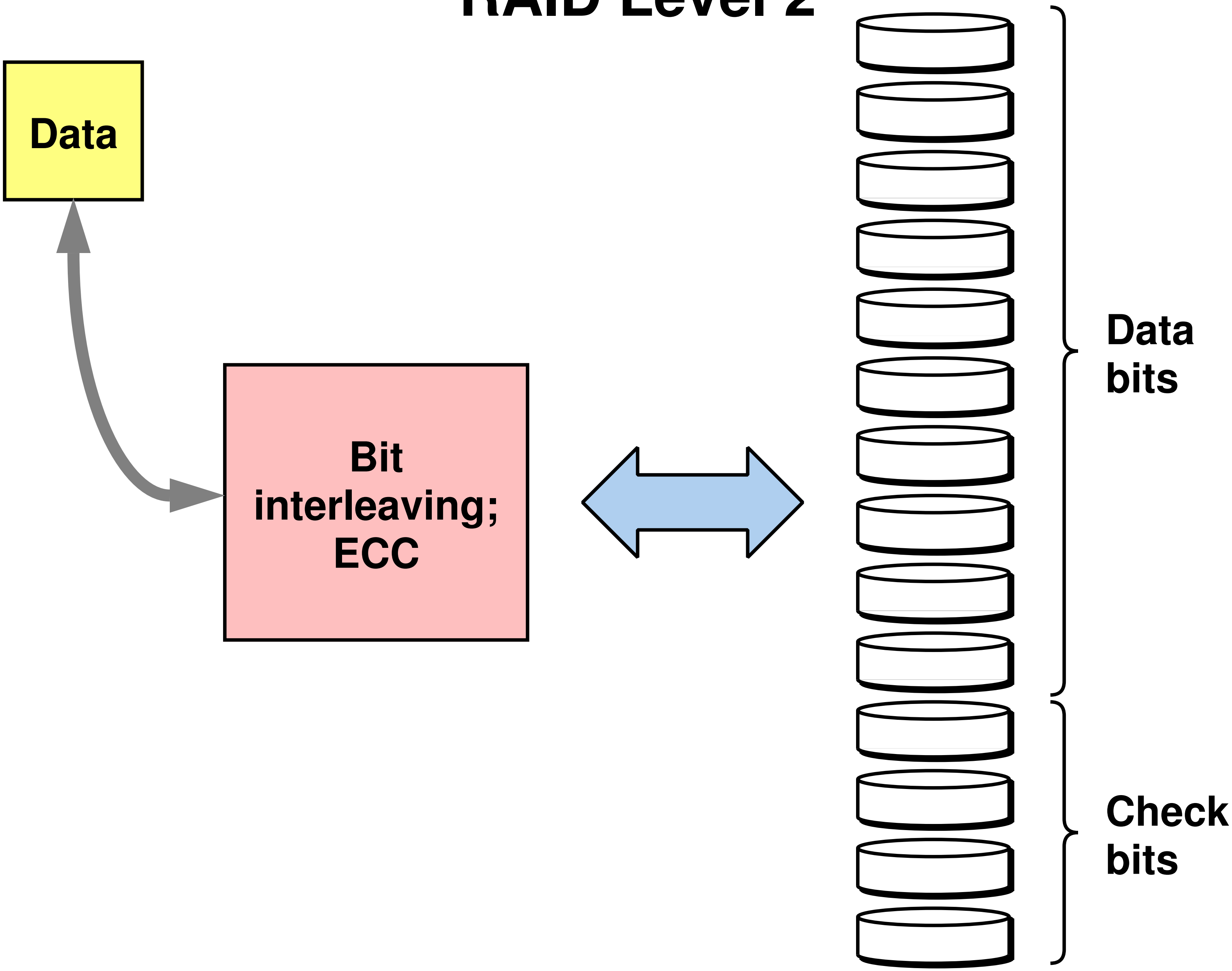
RAID to the Rescue

- ***RAID:*** **Redundant Array of Inexpensive Disks**
  - (as opposed to Single Large Expensive Disk: SLED)
  - combine *striping* with *mirroring*
  - 5 different variations originally defined
    - RAID level 1 through RAID level 4 developed by IBM
    - RAID level 5 developed by UC Berkeley
      - RAID level 0: pure striping (numbering extended later)
    - RAID level 1: pure mirroring

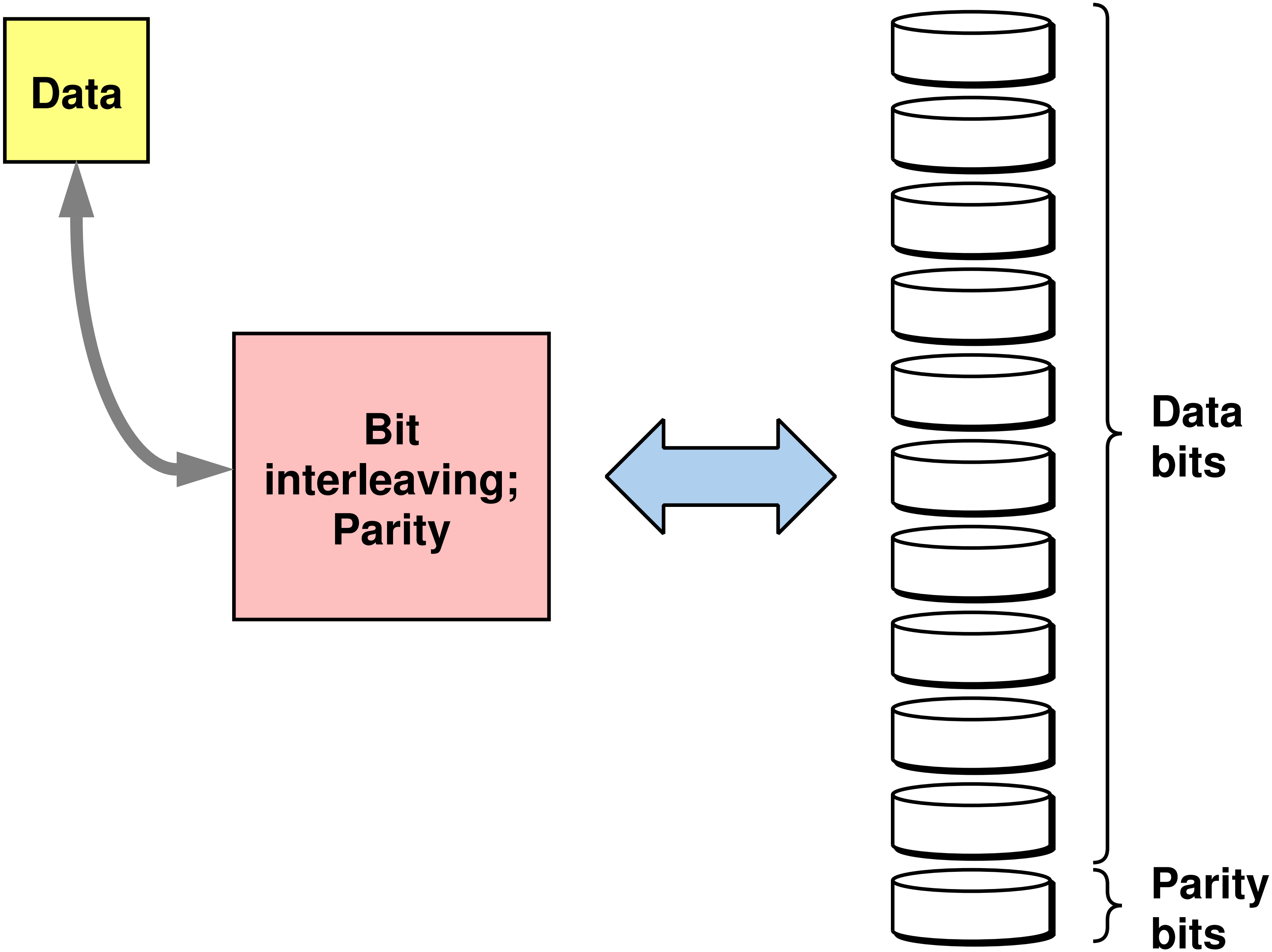
RAID Level 1



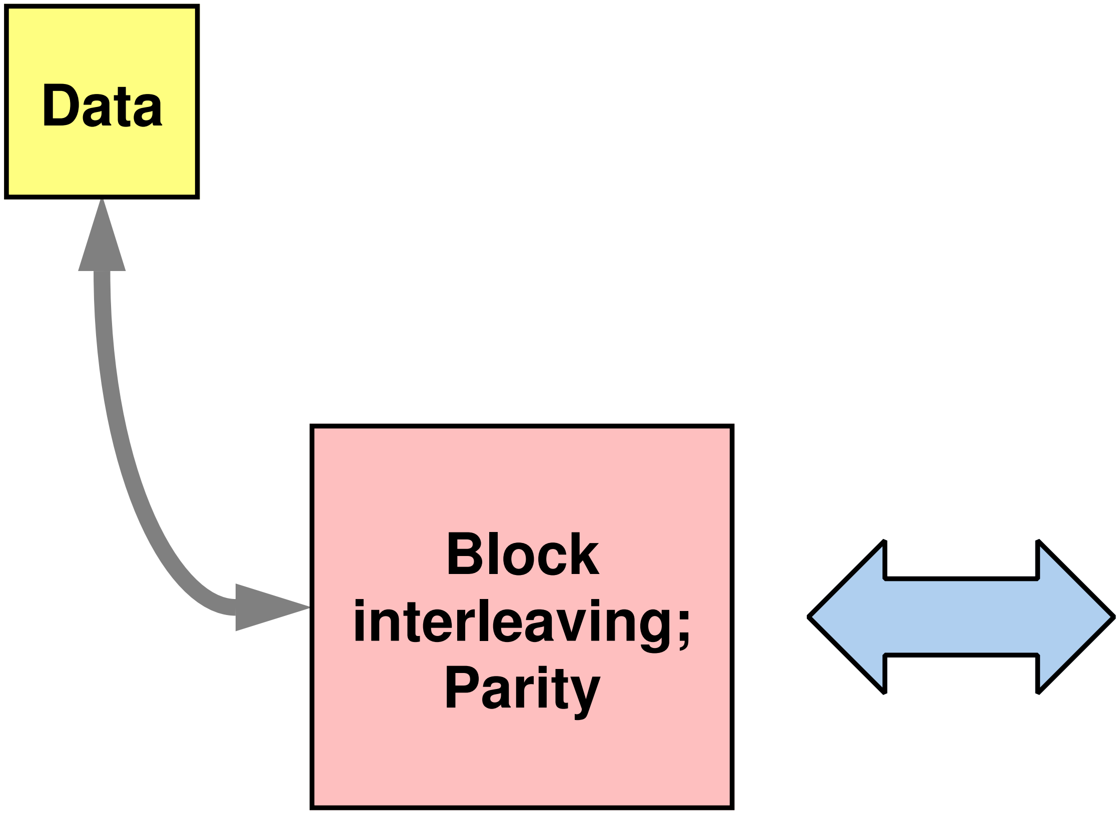
RAID Level 2

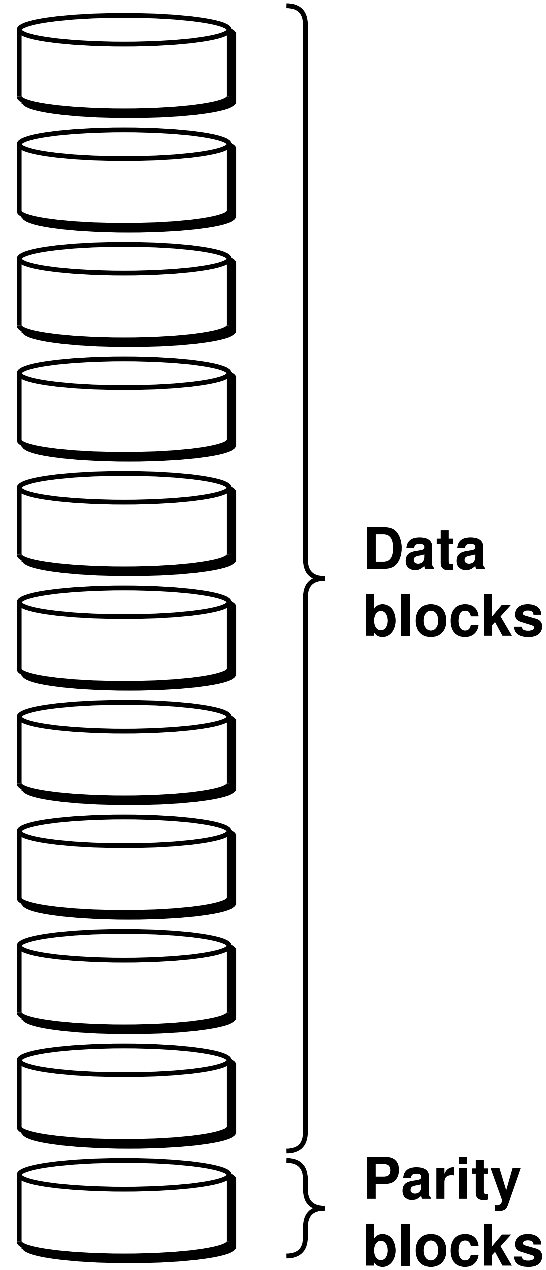


RAID Level 3



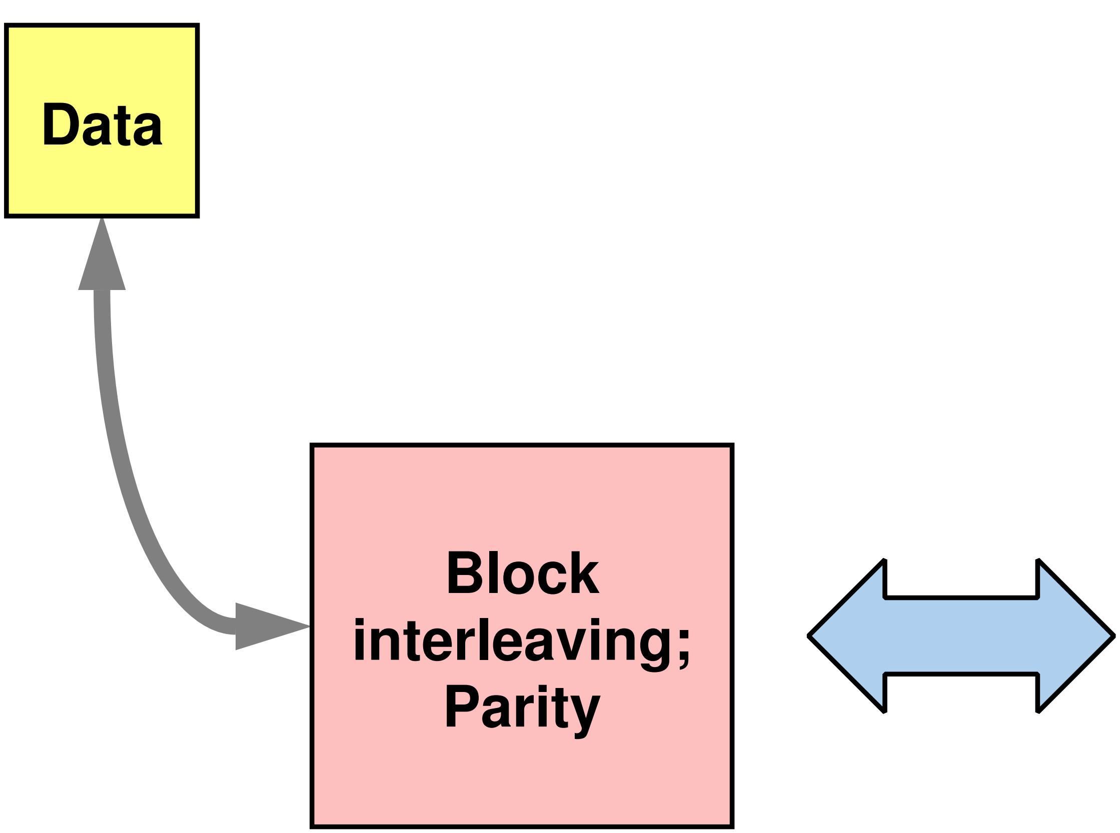
RAID Level 4

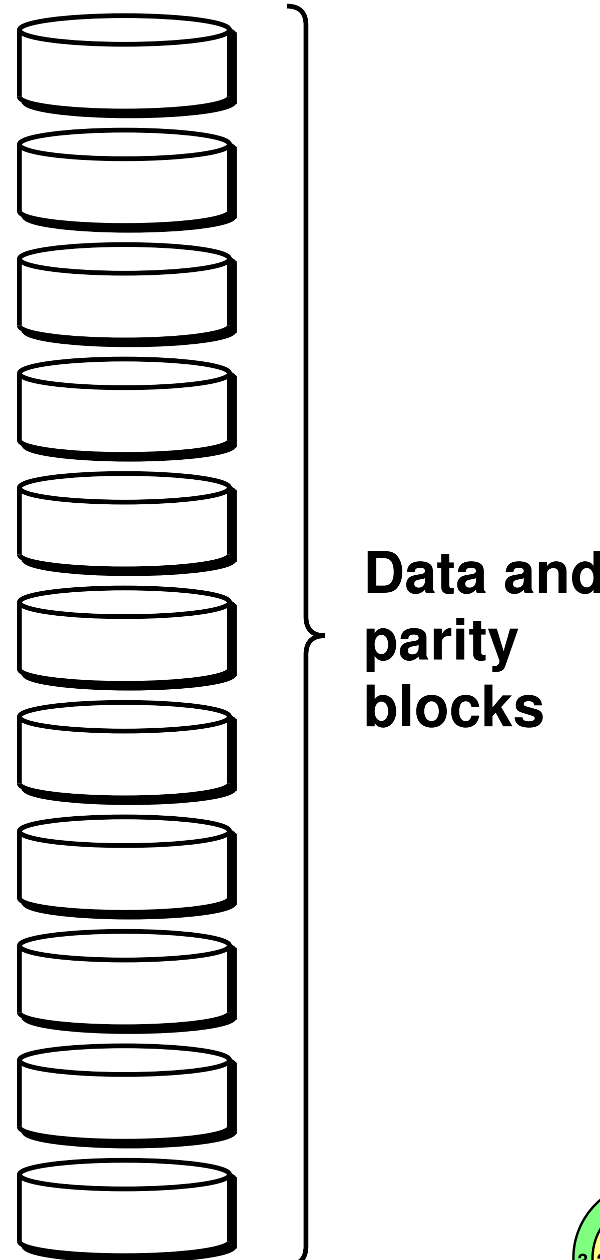




- Write only to 2 disks if only one block is modified
  - but *write performance bottleneck* at the *parity disk*

RAID Level 5





- Parity *blocks* are *spread* among *all* the disks in a systematic way
  - *stripe width < number of disks*

RAID 4 vs. RAID 5

- Lots of small writes
  - RAID 5 is best
- Mostly large writes
  - multiples of stripes
  - either is fine
- Expansion
  - add an additional disk or two
  - RAID 4: add them and recompute parity
  - RAID 5: add them, recompute parity, shuffle data blocks among all disks to reestablish check-block pattern
- Write performance
  - RAID 4: parity disk have workload multiple of other disks
  - RAID 5: same workload on all disks on the average
- One disk failure
  - RAID 4: parity disk have workload multiple of other disks
  - RAID 5: work load spread out more evenly

Beyond RAID 5

- RAID 6
  - like RAID 5, but additional parity
  - handles two failures
- Cascaded RAID
  - RAID 1+0 (RAID 10)
    - striping across mirrored drives
  - RAID 0+1
    - two striped sets, mirroring each other

6.5 Flash Memory

- Flash Technology
- Flash-Aware File Systems
- Augmenting Disk Storage

Beyond Disks: Flash

- Pro
  - Flash block ≈ file-system block
  - Random access
    - no seek, no rotational latency
  - Low power
  - Vibration-resistant
- Con
  - Limited lifetime
  - Write is expensive
  - Cost more than disks
    - 128GB SSD: ~$300
    - 1TB disk: ~$60

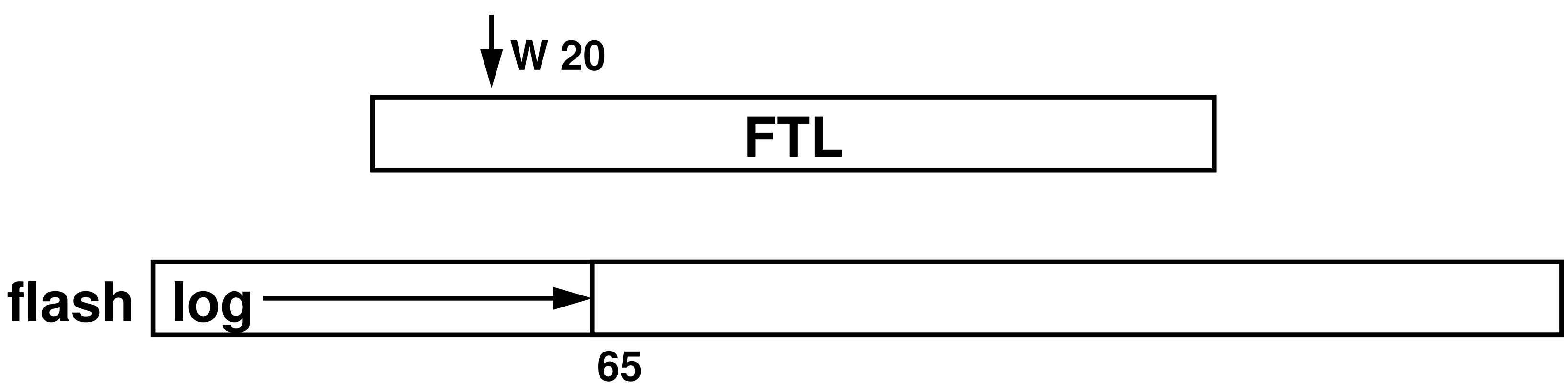
Flash Memory

- Two technologies
  - NOR
    - byte addressable
  - NAND
    - page addressable (about 1-4KB per page and 512KB per block)
    - cheaper
      - suitable for file systems use
    - limit on P/E (program/erase) cycle, about 10,000

- Writing
  - newly "erased" block is all ones
  - "programming" changes some ones to zeroes
    - per byte in NOR; per page in NAND (multiple pages/block)
    - to change zeroes to ones, must erase entire block
    - can erase no more than ~100k times/block

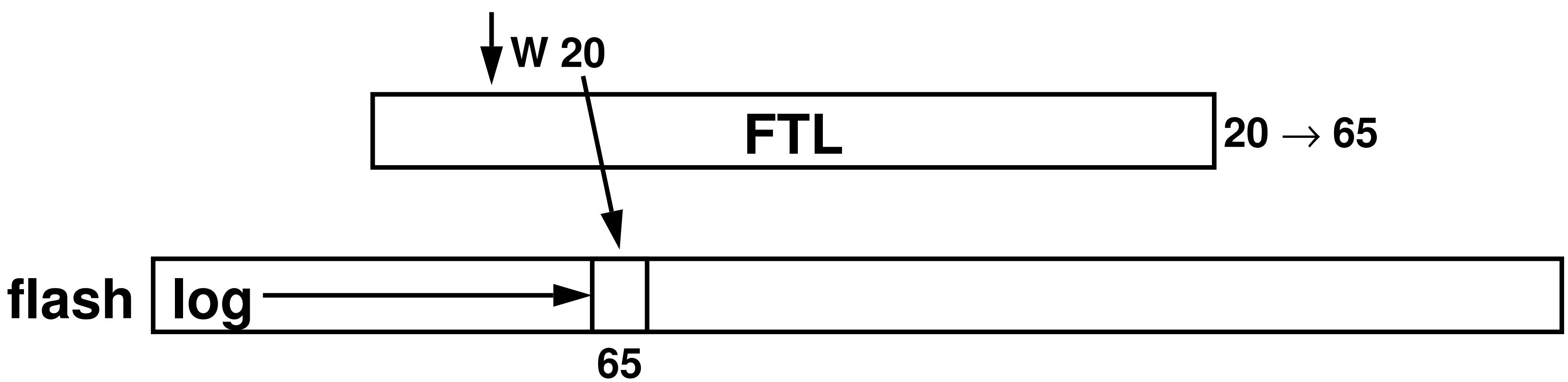
Coping

- *Wear leveling*
  - spread writes (erasures) across entire drive
  - approache:
    - *flash translation layer (FTL)*
    - *log-structured file system*
      - blocks on the flash drive are used sequentially
- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

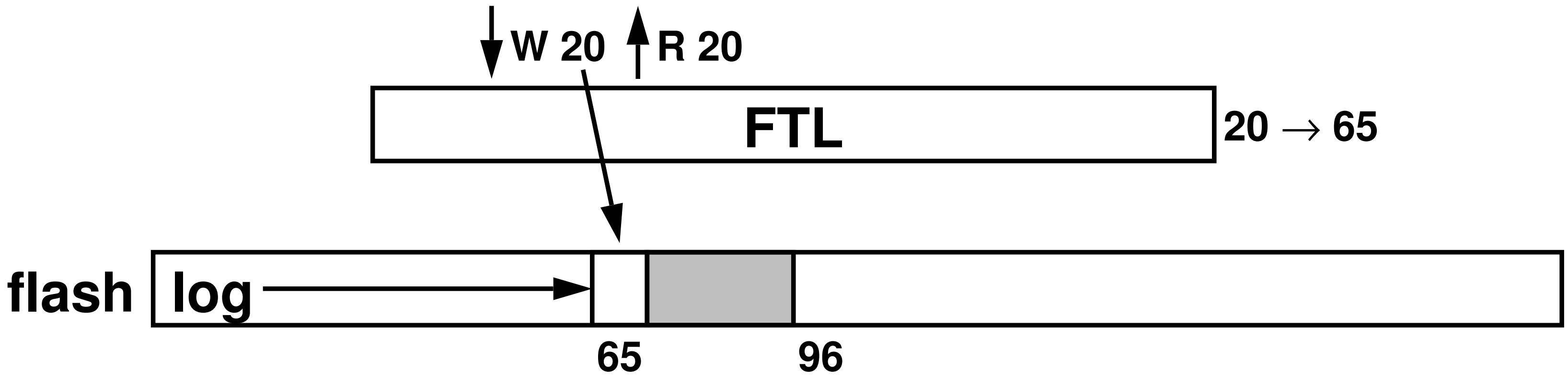
- *Wear leveling*
  - spread writes (erasures) across entire drive
  - approaches:
    - *flash translation layer (FTL)*
    - *log-structured file system*
      - blocks on the flash drive are used sequentially
- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

- ***Wear leveling***
  - spread writes (erasures) across entire drive
  - approaches:
    - ***flash translation layer (FTL)***
    - ***log-structured file system***
      - blocks on the flash drive are used sequentially

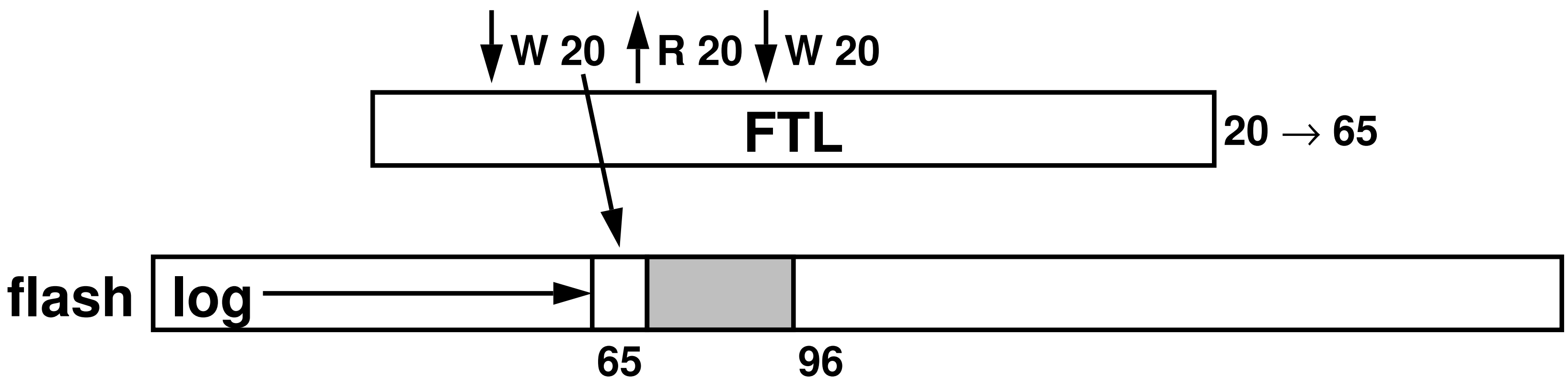
- **FTL: Flash translation layer (often on a separate device)**
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

- *Wear leveling*
  - spread writes (erasures) across entire drive
  - approaches:
    - *flash translation layer (FTL)*
    - *log-structured file system*
      - blocks on the flash drive are used sequentially

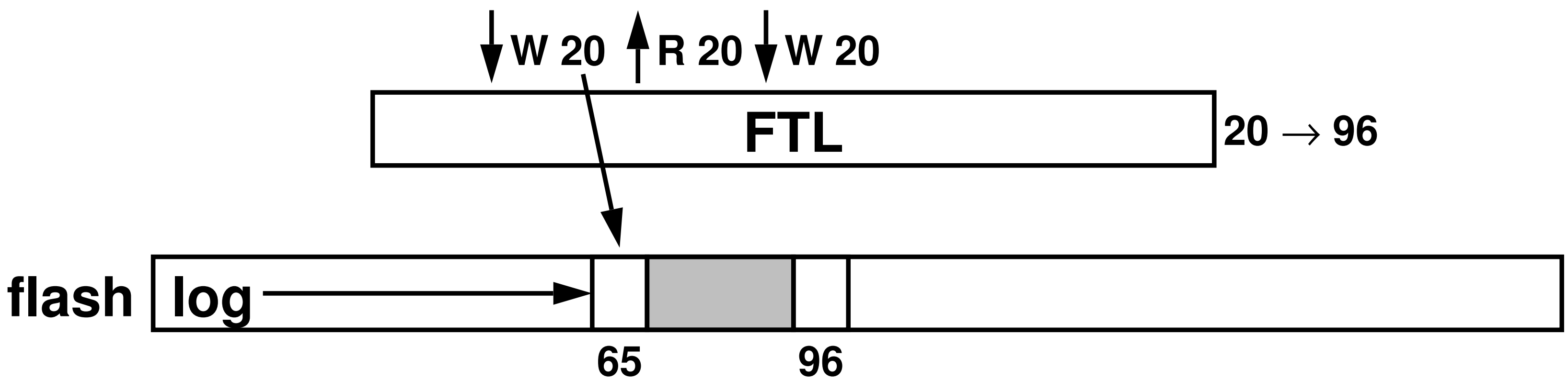
- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

- ***Wear leveling***
  - spread writes (erasures) across entire drive
  - approaches:
    - ***flash translation layer (FTL)***
    - ***log-structured file system***
      - blocks on the flash drive are used sequentially

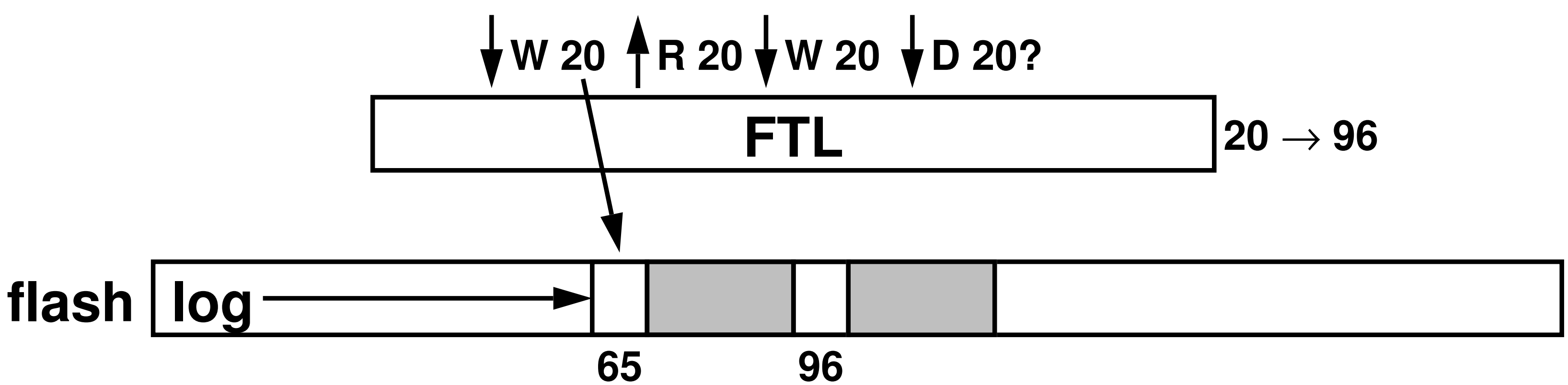
- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

- *Wear leveling*
  - spread writes (erasures) across entire drive
  - approaches:
    - *flash translation layer (FTL)*
    - *log-structured file system*
      - blocks on the flash drive are used sequentially

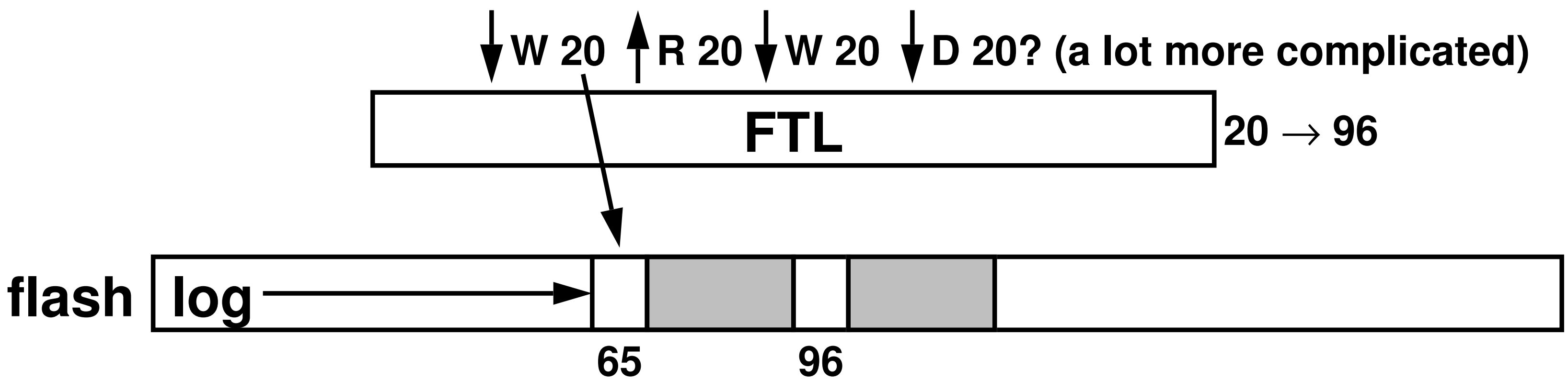
- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Coping

- *Wear leveling*
  - spread writes (erasures) across entire drive
  - approaches:
    - *flash translation layer (FTL)*
    - *log-structured file system*
      - blocks on the flash drive are used sequentially

- FTL: Flash translation layer (often on a separate device)
  - specification from 1994
  - provides disk-like block interface (firmware on device controller)
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling



Flash with FTL

- Which file system?
  - FAT32 (sort of like S5FS, but from Microsoft)
  - NTFS
  - FFS
  - Ext3
- All were designed to exploit disks
  - much of what they do are irrelevant for flash

Flash without FTL

- Known as memory technology device (MTD)
  - software wear-leveling
  - perhaps other tricks

JFFS and JFFS2

- Journaling flash file system
  - log-based: no journal!
    - each log entry contains inode info and some data
    - garbage collection copies info out of partially obsoleted blocks, allowing block to be erased
    - complete index of inodes (i.e., meta-data) kept in RAM
      - entire file system must be read when mounted

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)
  - file map kept in flash as B+ tree
  - no need to scan entire file system when mounted

Flash as Part of the Hierarchy

- Flash as log device
  - aggregate write throughput sufficient, but latency is bad
  - augment with DRAM and a "super-capacitor"
- Flash as cache
  - large level-2 cache
    - integrated into ZFS
    - can use cheaper (slower) disks with no loss of performance
      - reduced power consumption

6.6 Case Studies

- FFS
- Ext3
- Reiser FS
- NTFS
- WAFL
- ZFS

Linux

- Linux had 57 file systems built for it to date!
- FFS
  - ext2
  - ext3 (journaling - crash resiliency)
    - ReiserFS (B-tree everywhere)
  - ext4
    - extents (optimize read/write)
    - LVM
    - hash trees for directories
  - BtrFS (Oracle)

Windows NT

- NTFS
  - extents (optimize read/write)
  - B-trees (optimize directory lookup)
  - journaling (crash resiliency)

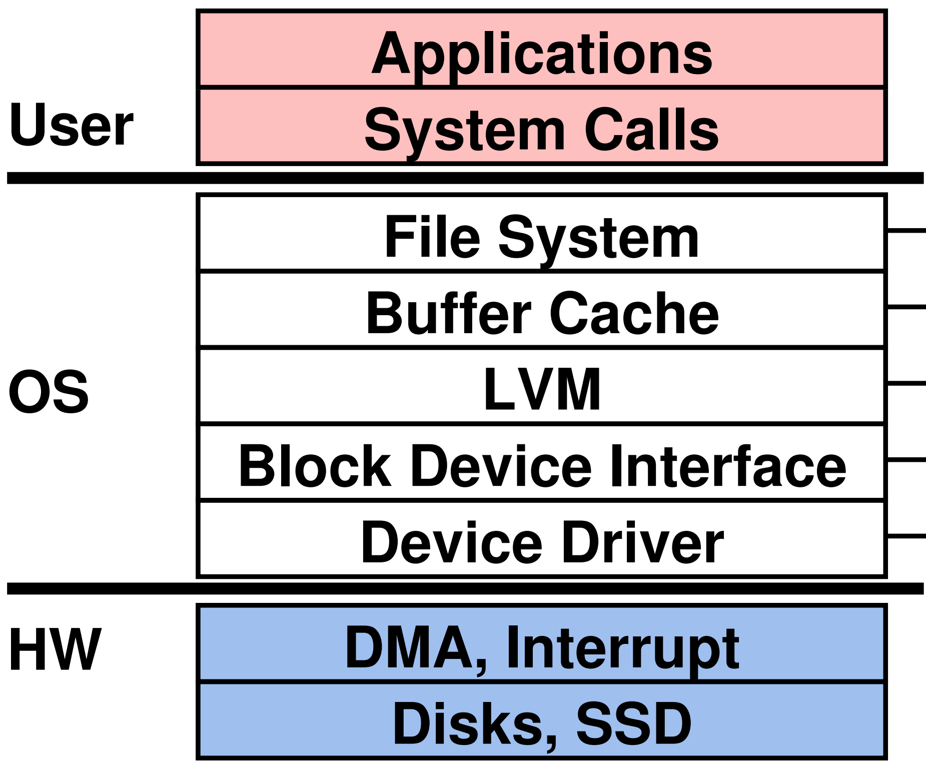
Mac OS X

- Mac OS X
  - HFS+ (planned to use ZFS but dropped the idea)
    - extents (optimize read/write)
    - B*-trees (optimize directory lookup)
    - journaling (crash resiliency)

Journaling

- Why did everyone choose journaling and not shadow pages?
  - journaling can be added to any existing file system

File System Summary



- File System: abstraction of files, directories, etc.
- Buffer Cache: abstraction that *the device is very fast*
- LVM: presents multiple devices as *one large array of blocks*
- Block Device Interface: presents the abstraction that *the device is simply an array of blocks*
- Device Driver: *maps blocks to device addresses*

Extra Slides

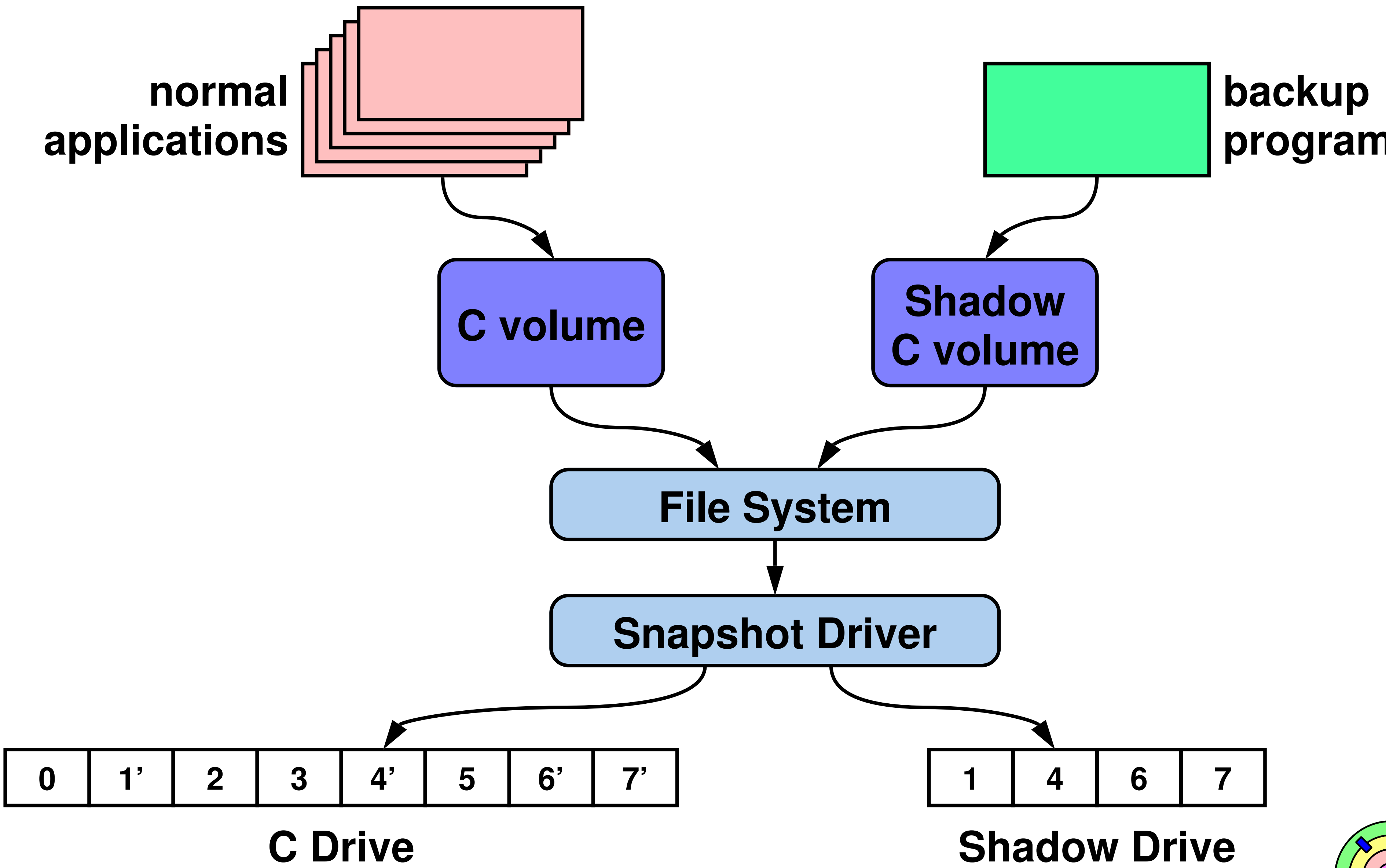
NTFS

- "Volume aggregation" options
  - spanned volumes
  - RAID 0 (striping)
  - RAID 1 (mirroring)
  - RAID 5
  - snapshots

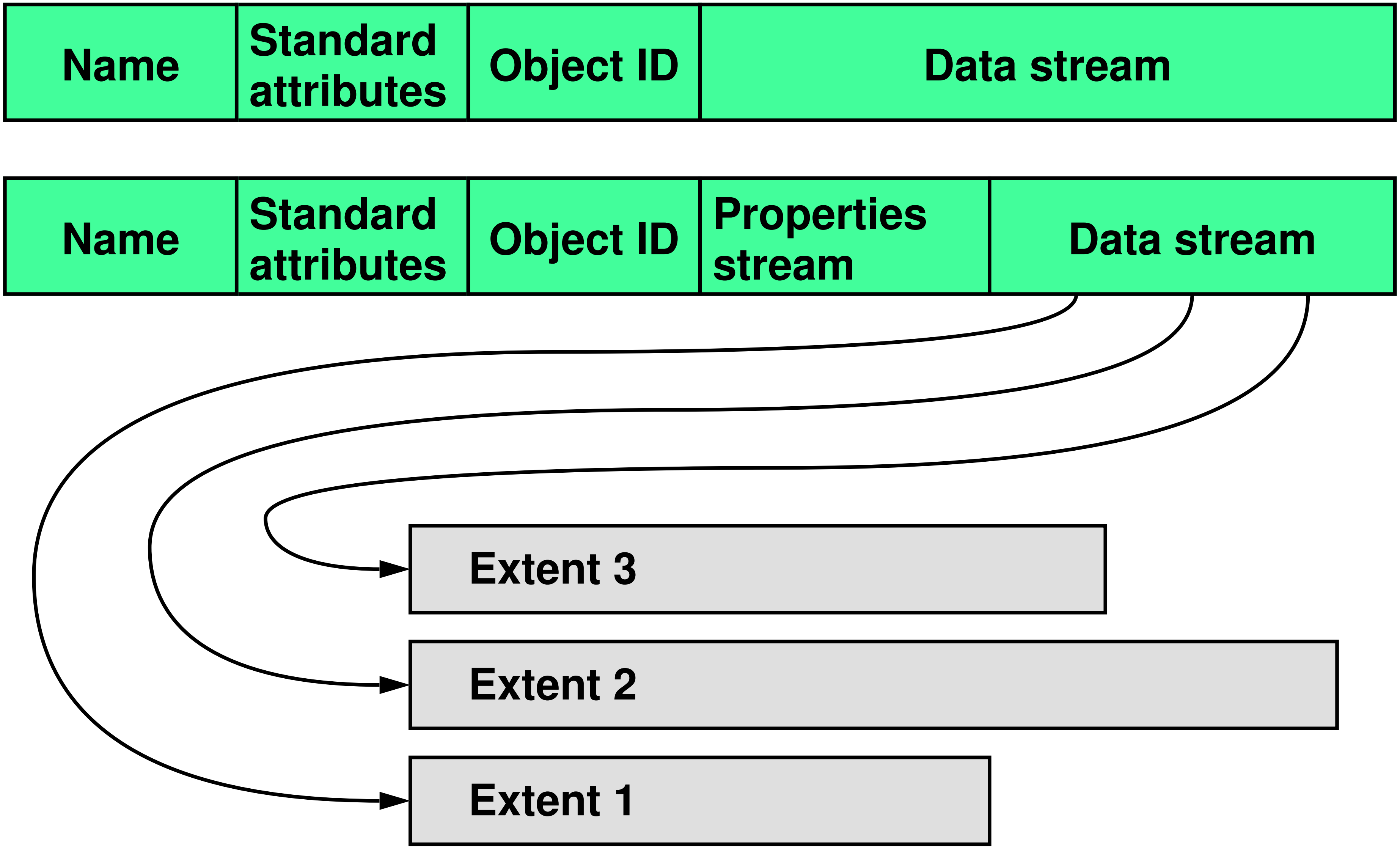
Backups

- Want to back up a file system
  - while still using it
  - files are being modified while the backup takes place
  - applications may be in progress - files in inconsistent states
- Solution
  - have critical applications quickly reach a safe point and pause
  - snapshot the file system
  - resume applications
  - back up the snapshot

Windows Snapshots



NTFS File Records



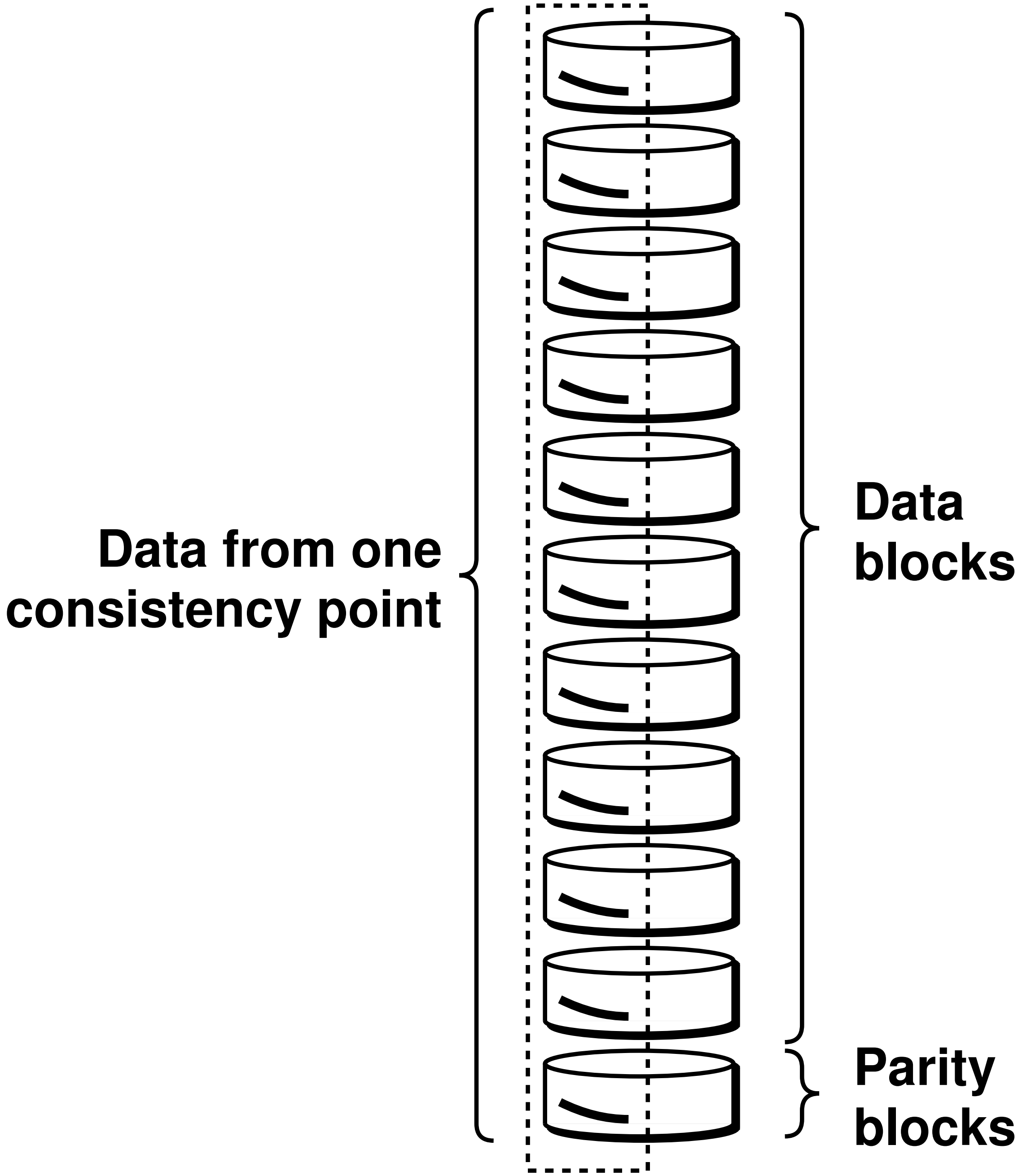
Additional NTFS Features

- 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 CIFS 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)

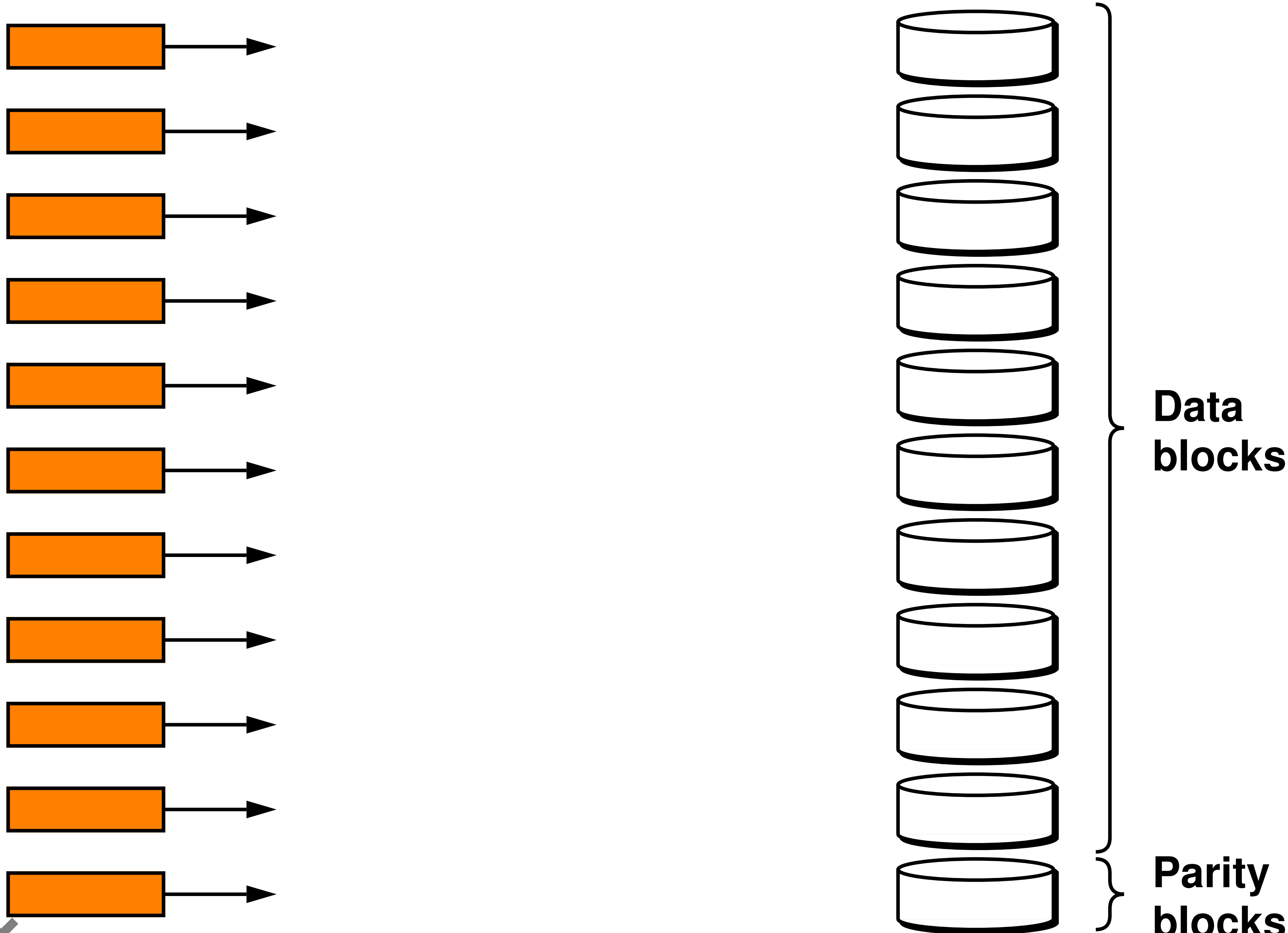
Snapshots

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

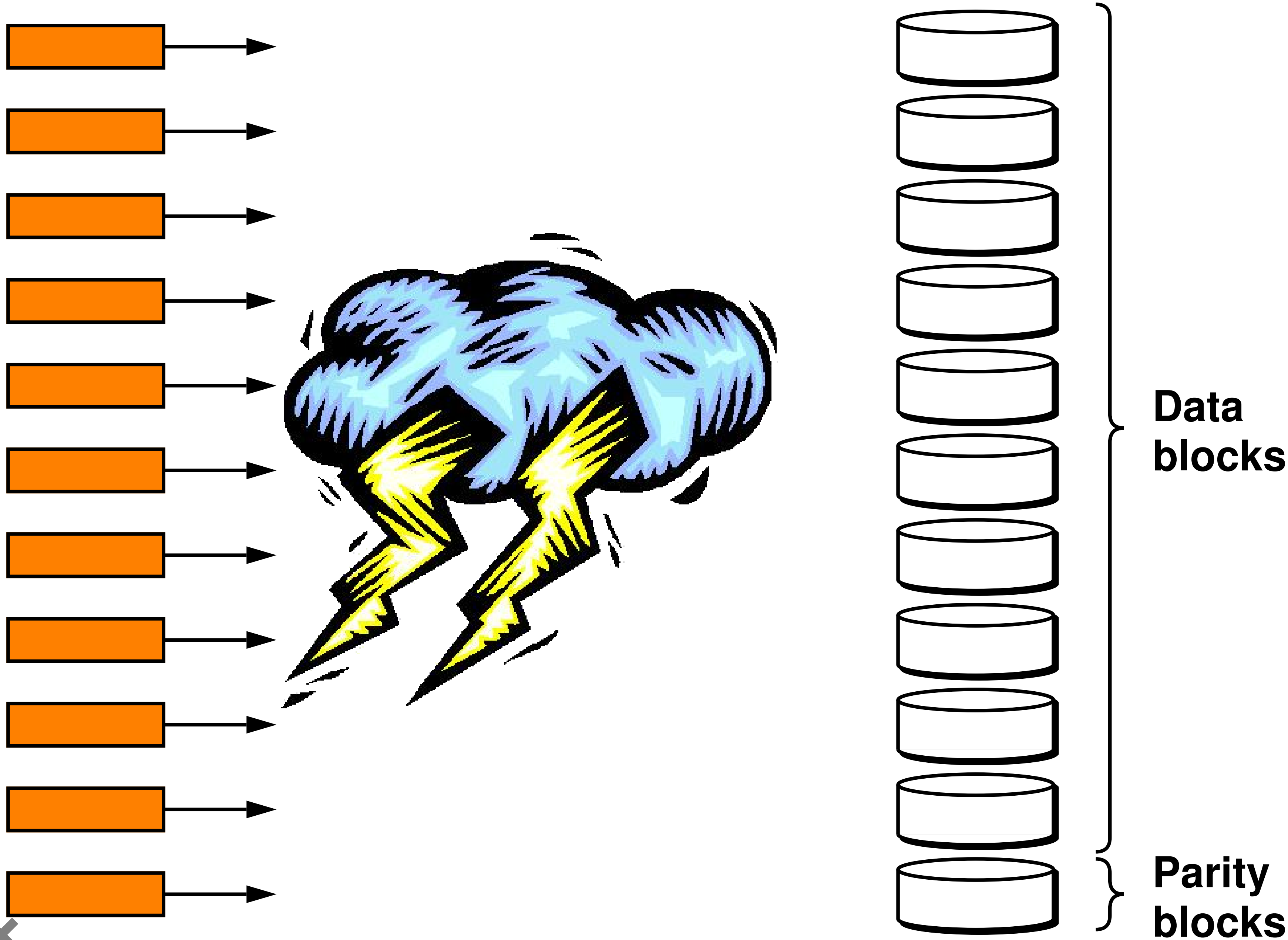
Paranoia

- 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 4TB file system
  - out of address space
  - 264 isn't as big as it used to be

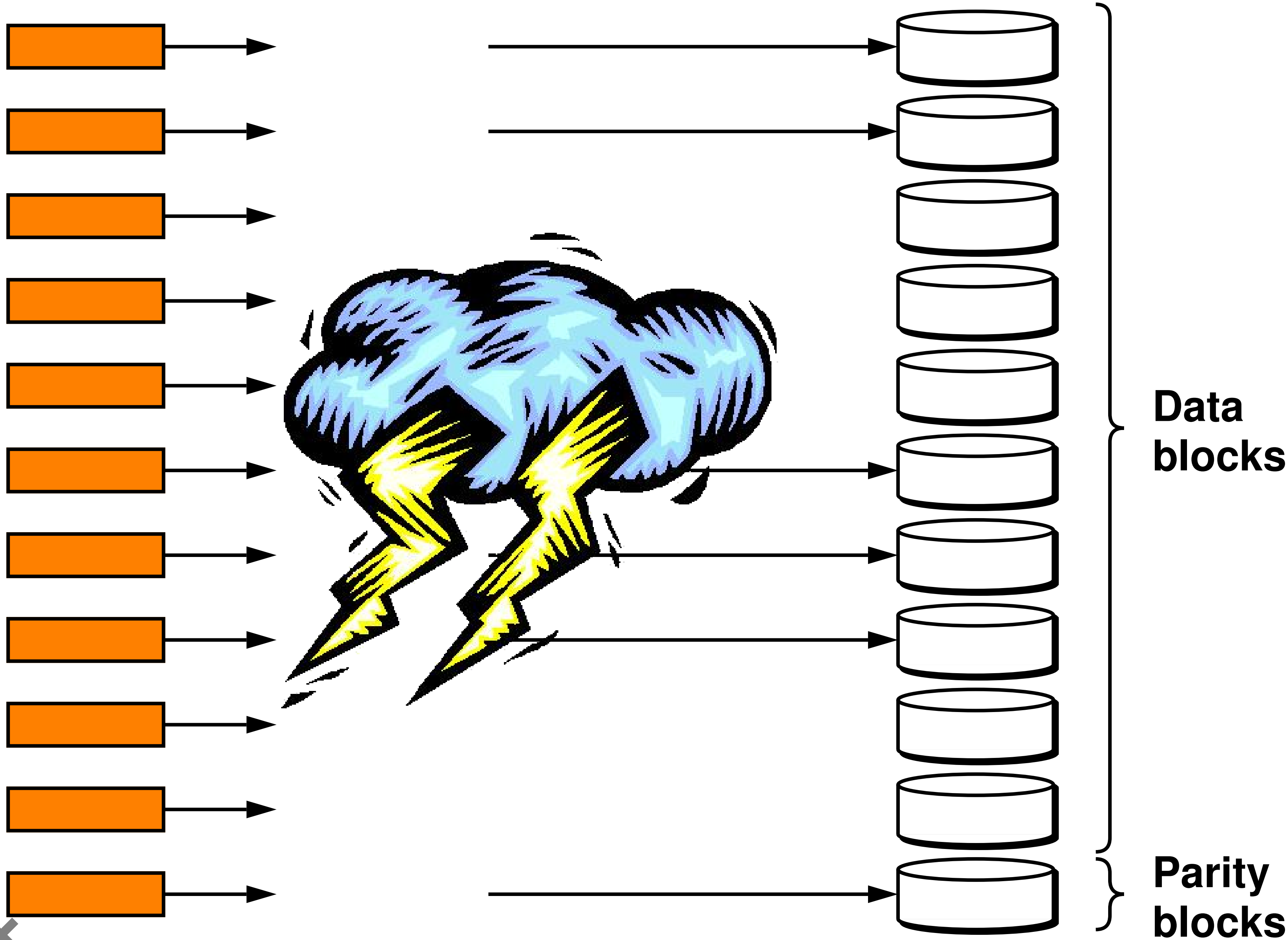
Partial Writes



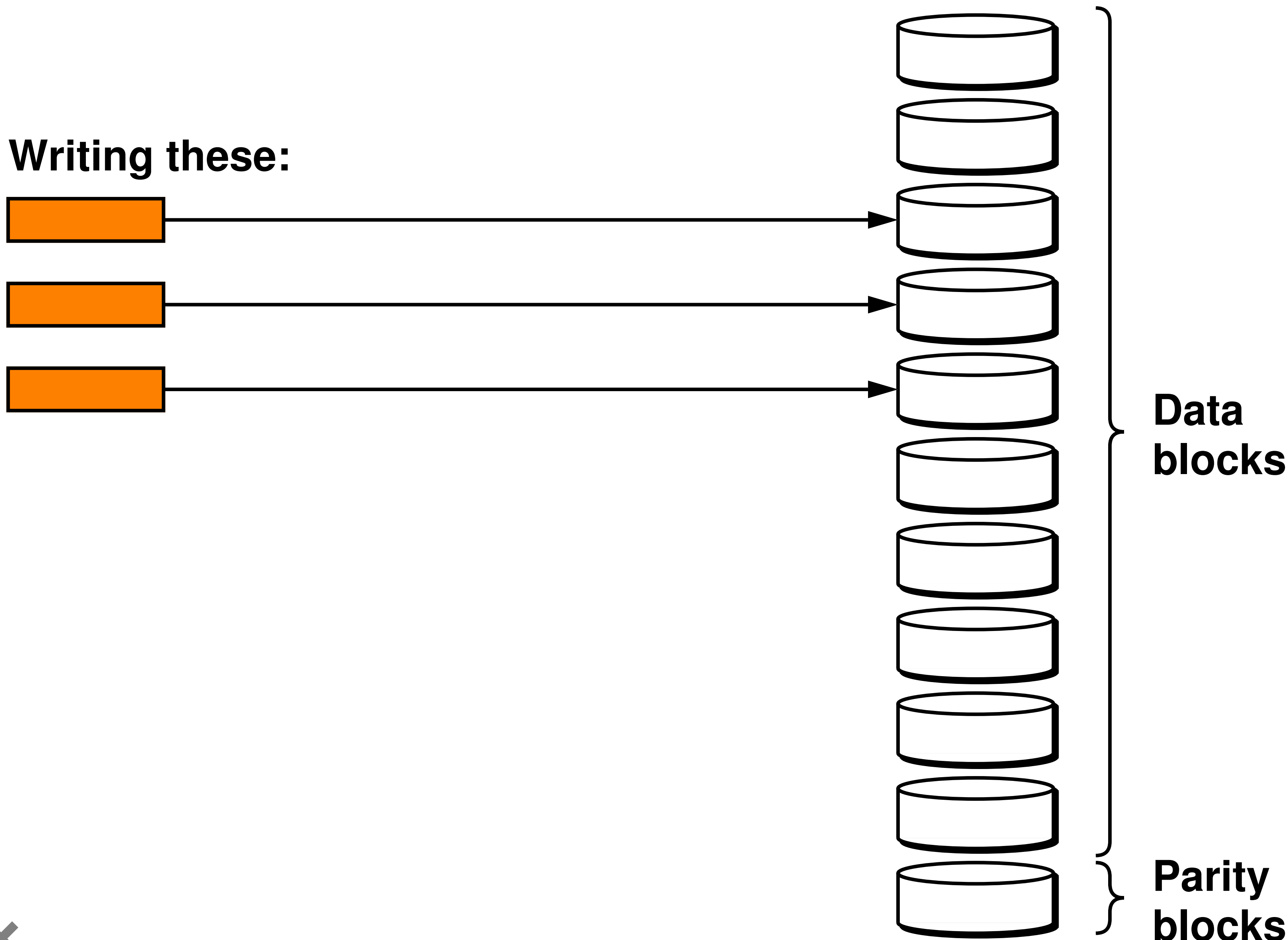
Partial Writes



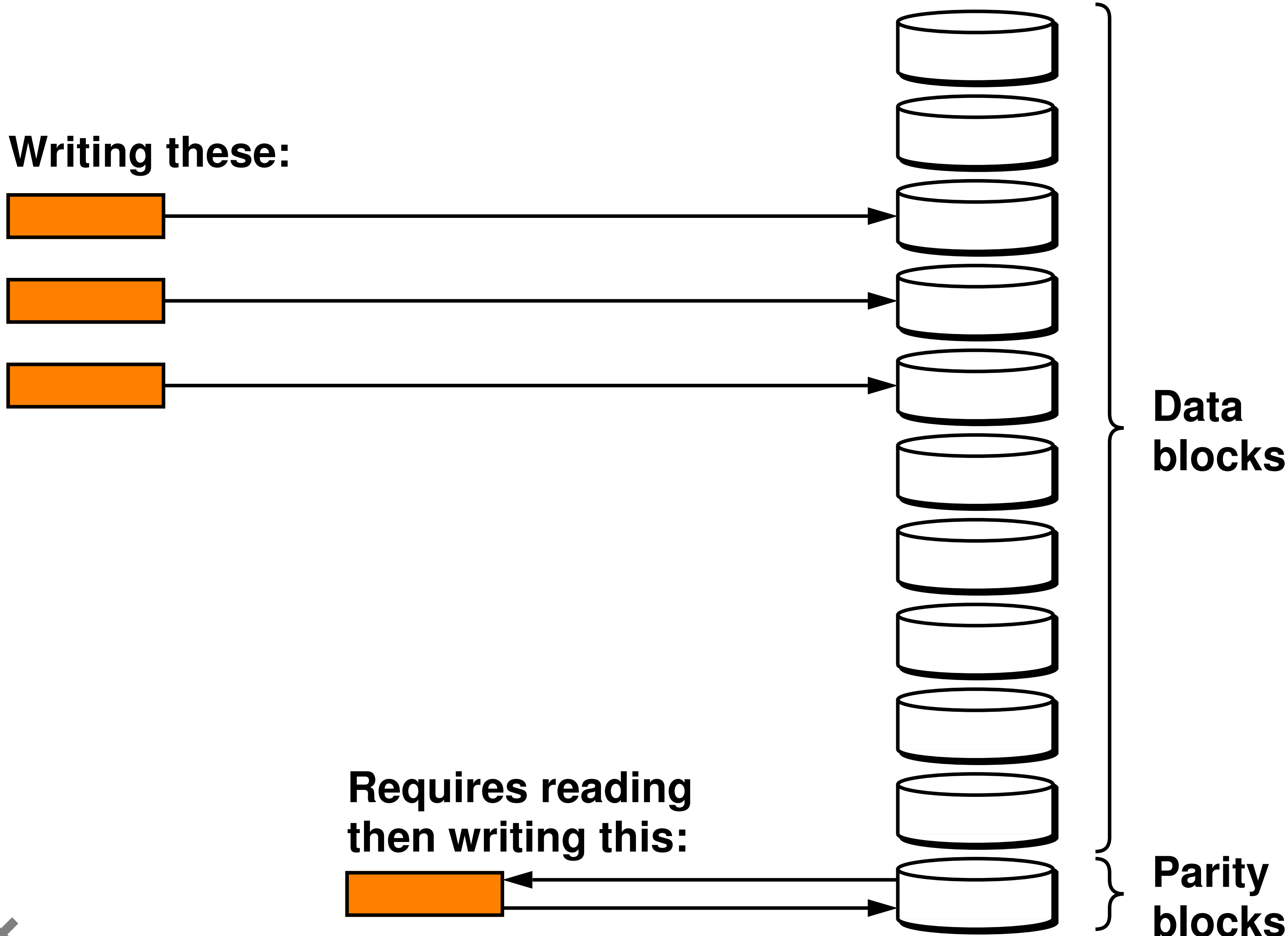
Partial Writes



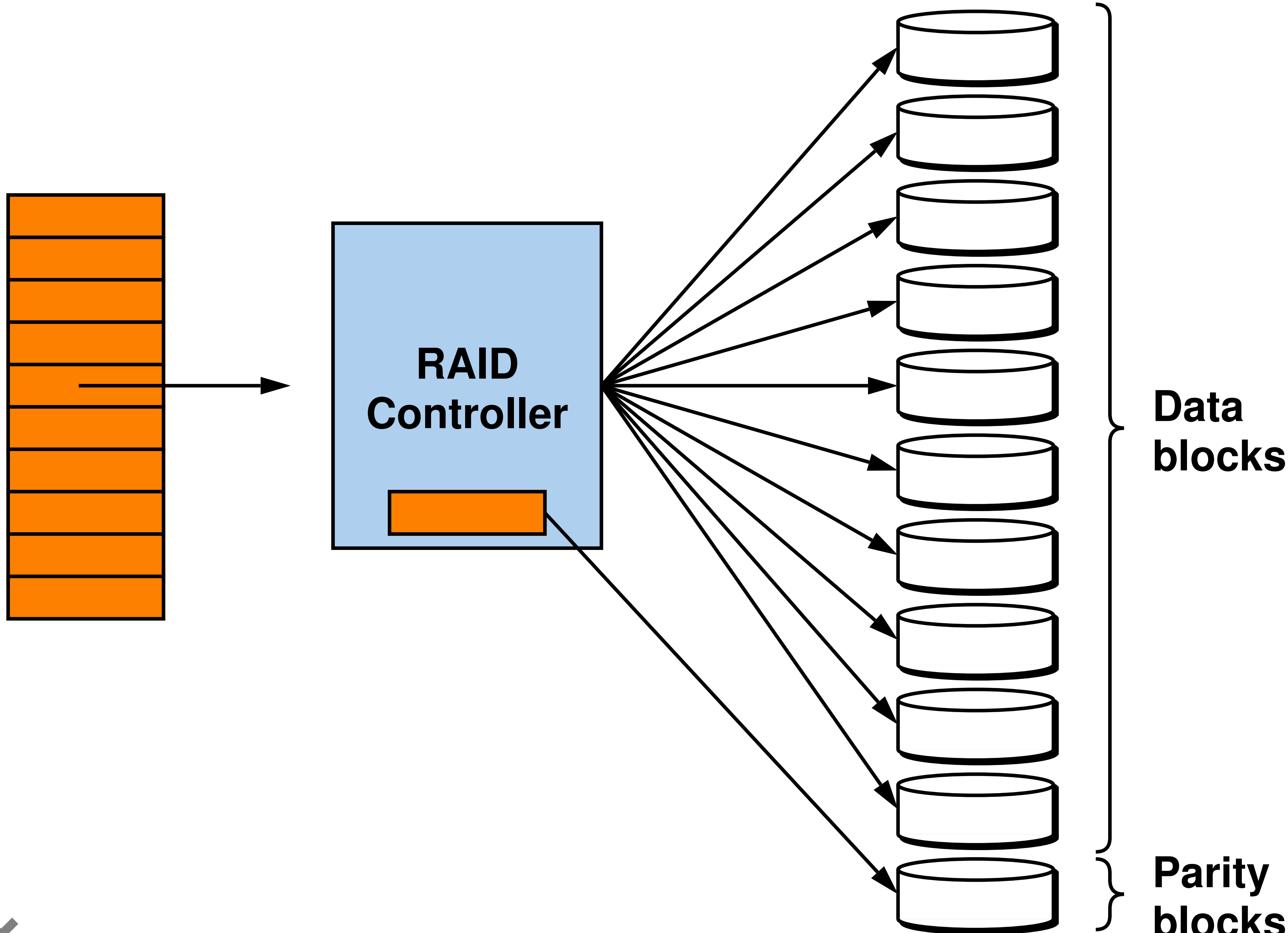
Small Writes



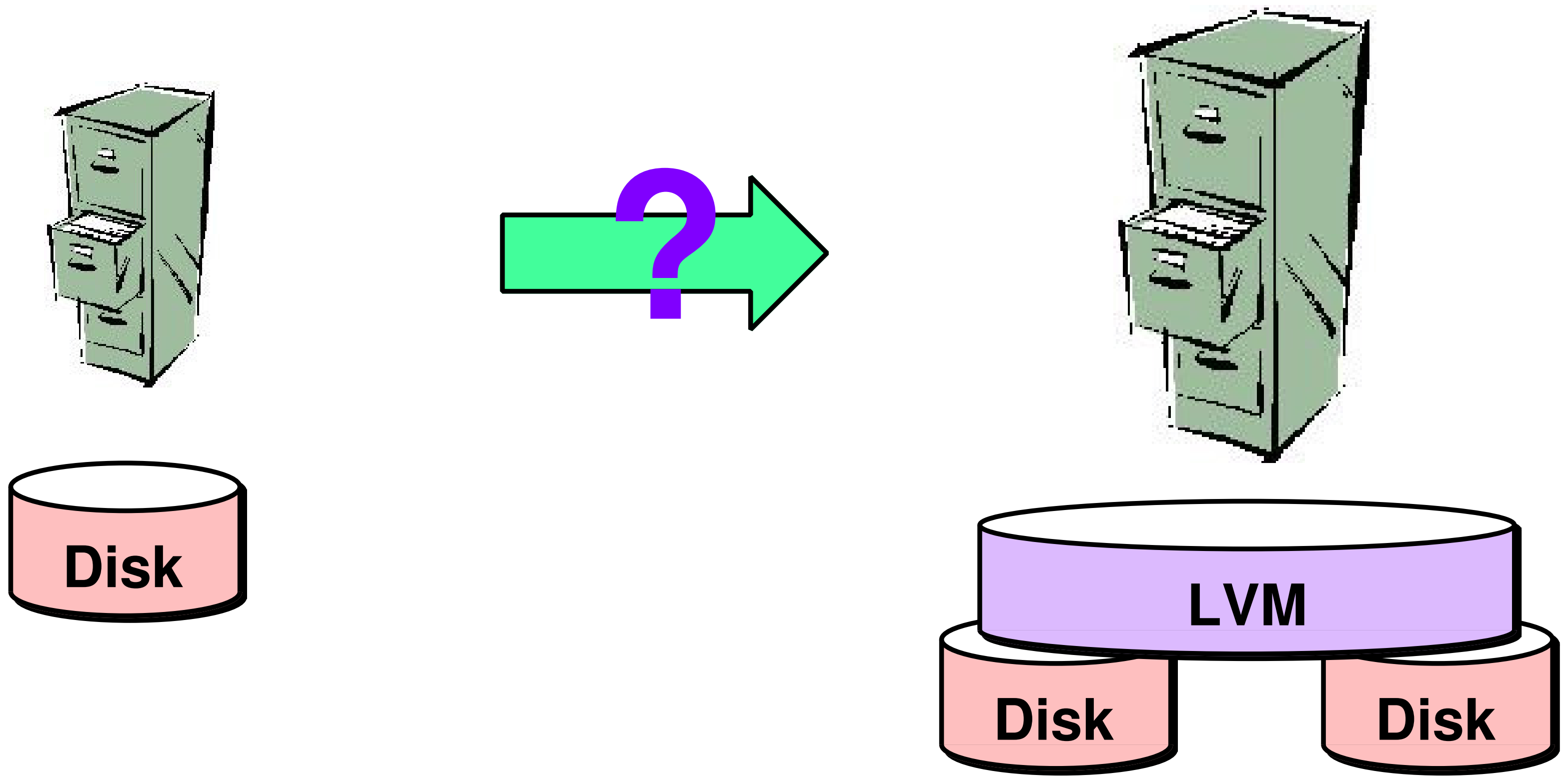
Small Writes



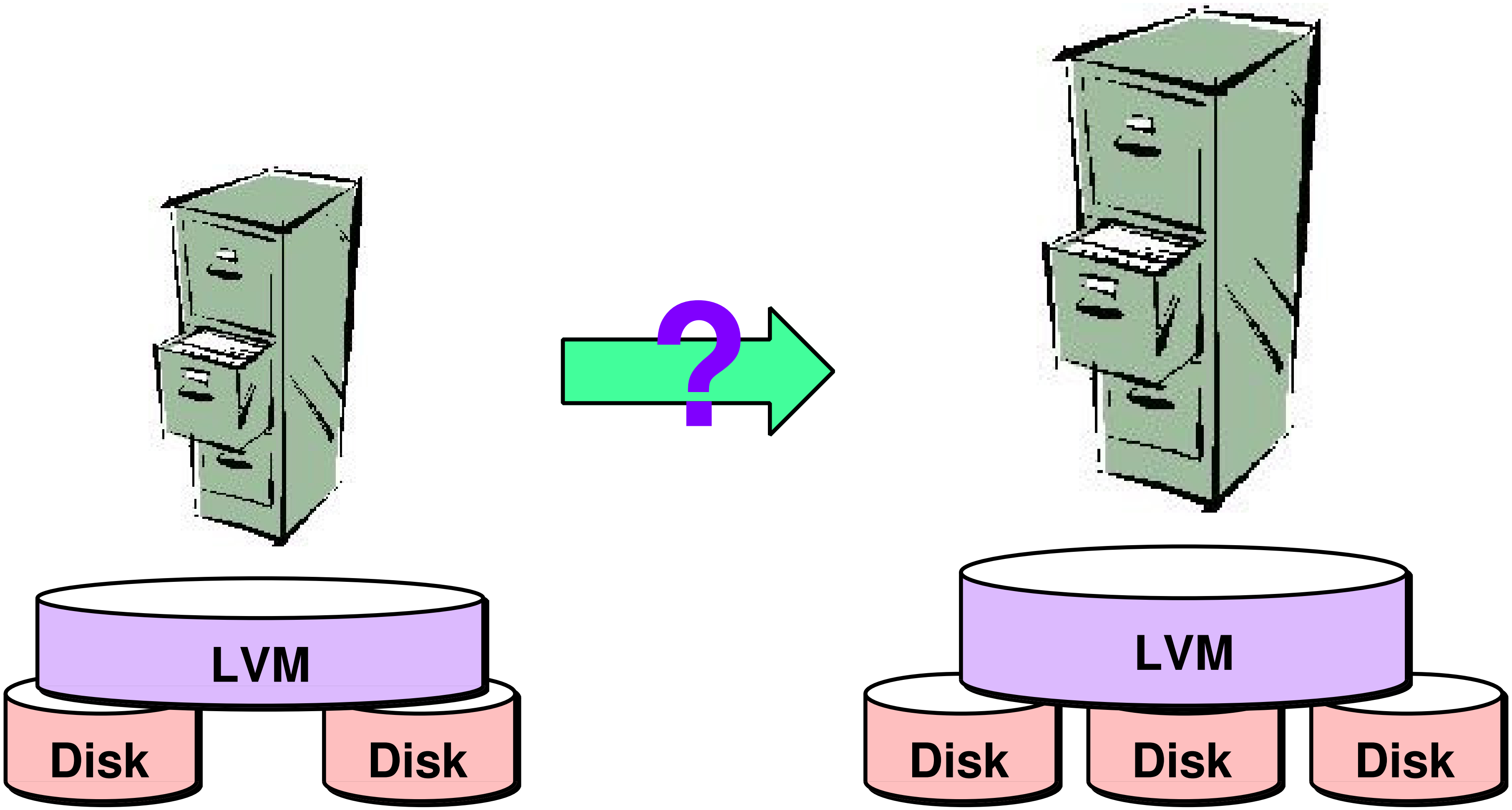
Hardware RAID



Adding a Disk (1)



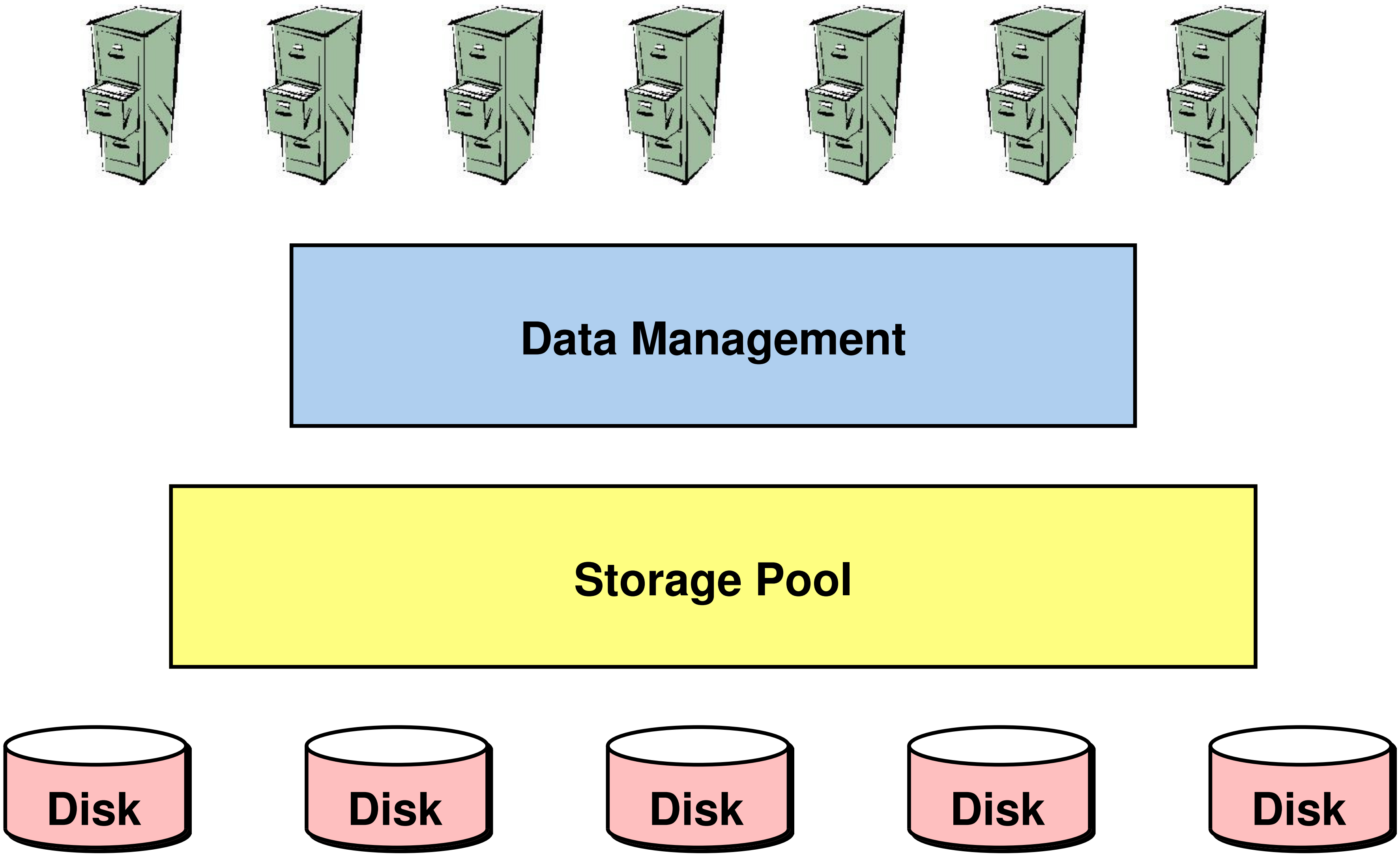
Adding a Disk (2)



ZFS

- **The Last (?!) Word in File Systems**

ZFS Layers



Enter ZFS

VFS

Vnode operations

ZFS POSIX Layer (ZPL)

<dataset, object, offset> — 264 objects; each up to 264 bytes

Data Management Unit (DMU) — Provides transactions on object

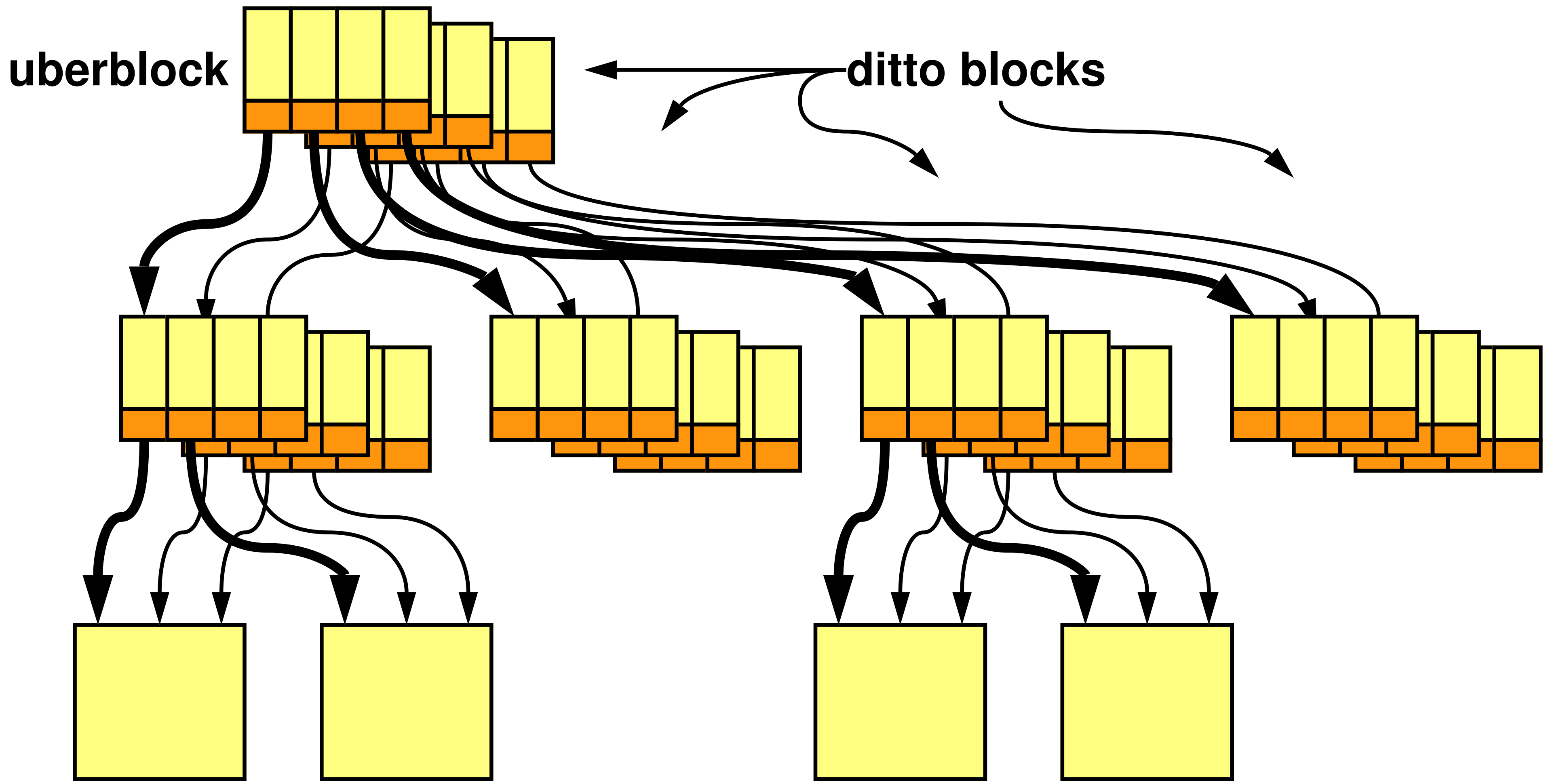
<data virtual address> — 128-bit addresses!

Storage Pool Allocator (SPA) — Maps virtual blocks to disks and physical blocks

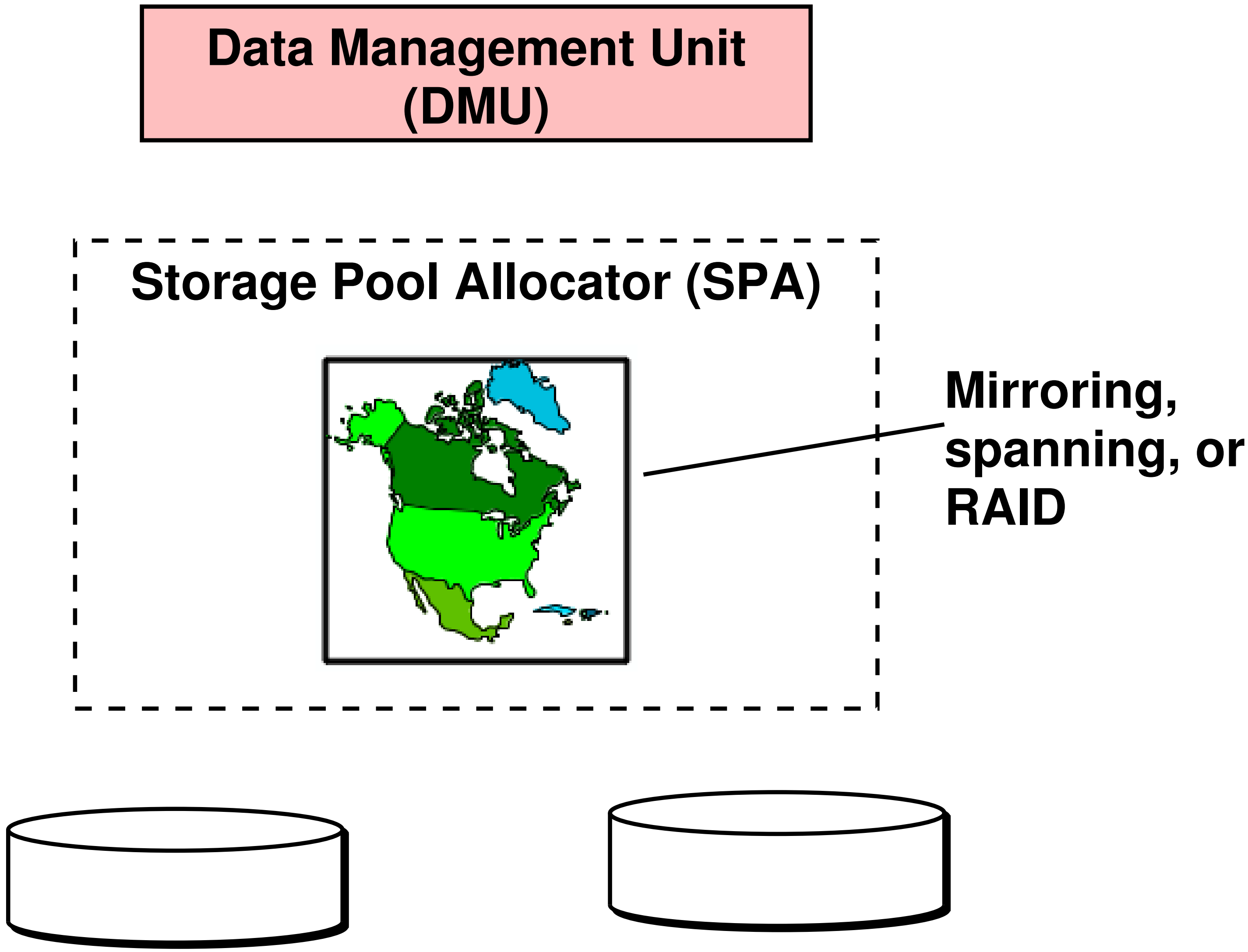
<physical device, offset>

Device Driver

Shadow-Page Tree (with a twist ...)

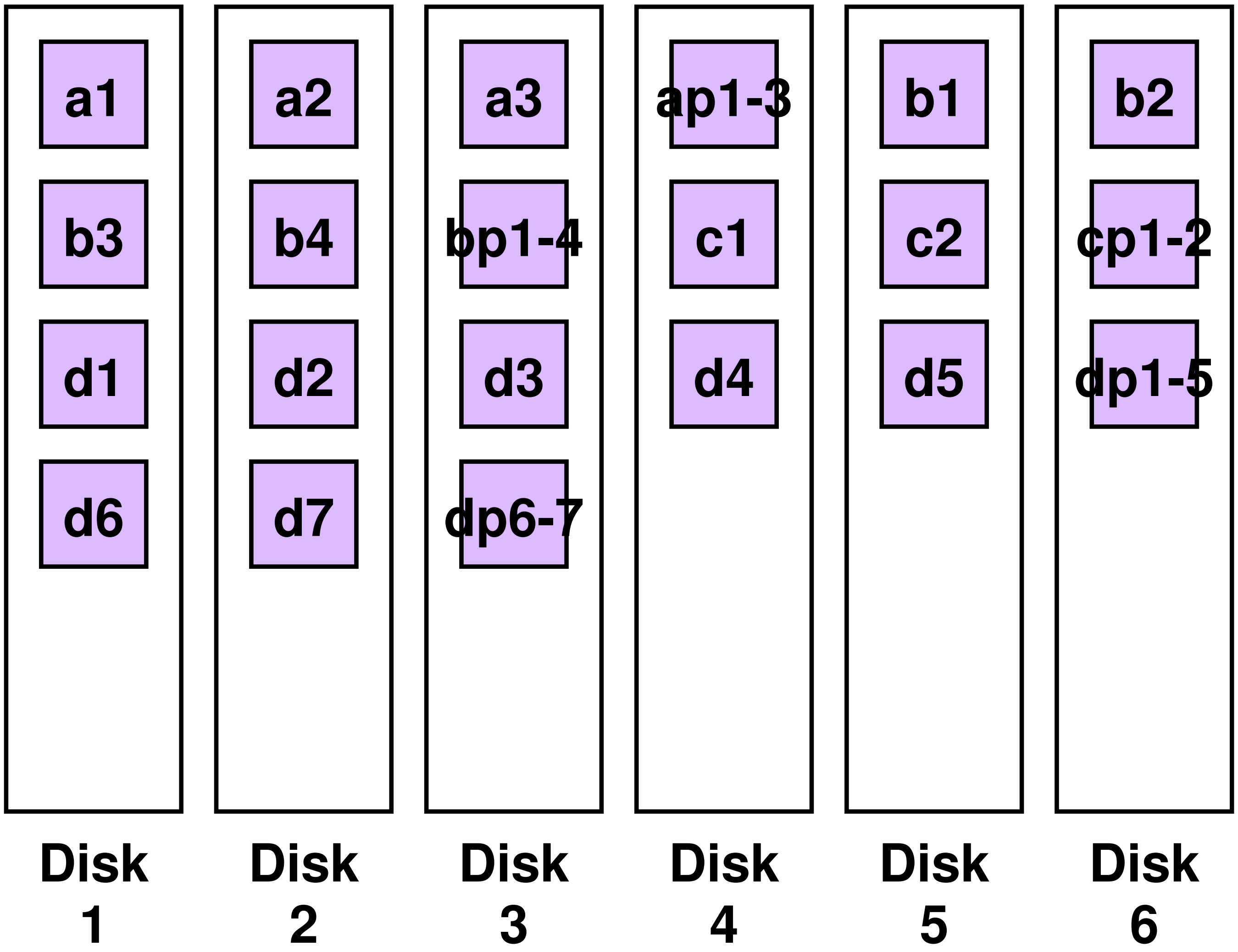


Storage Pool Allocator



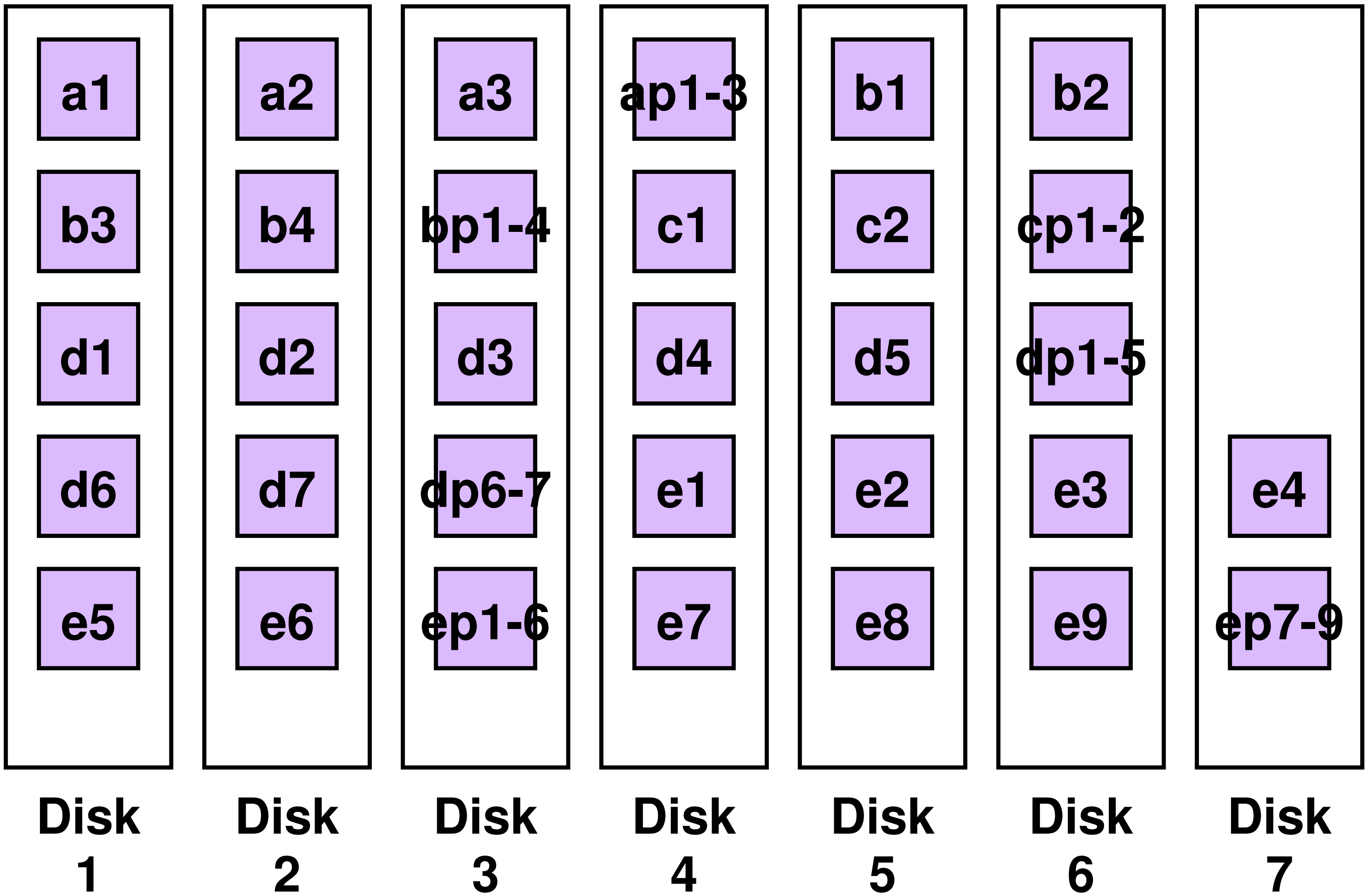
RAID-Z

- Software Dynamic Striping



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
  - 2128 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

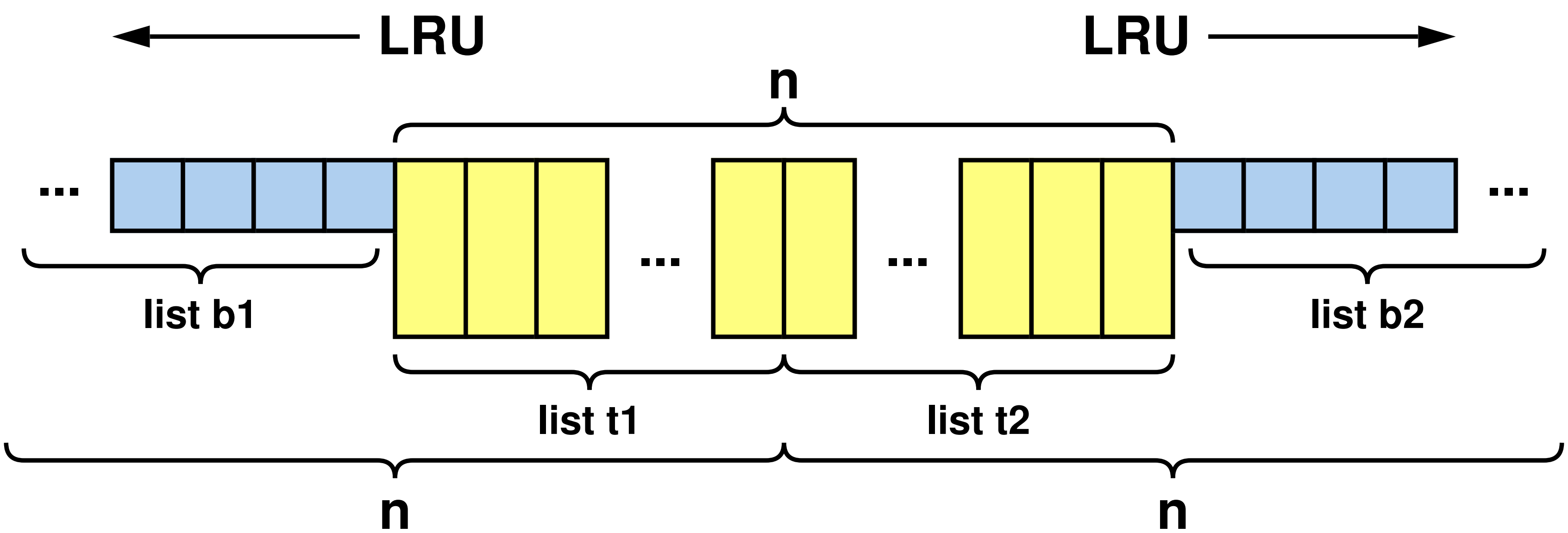
LRU Caching

- 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



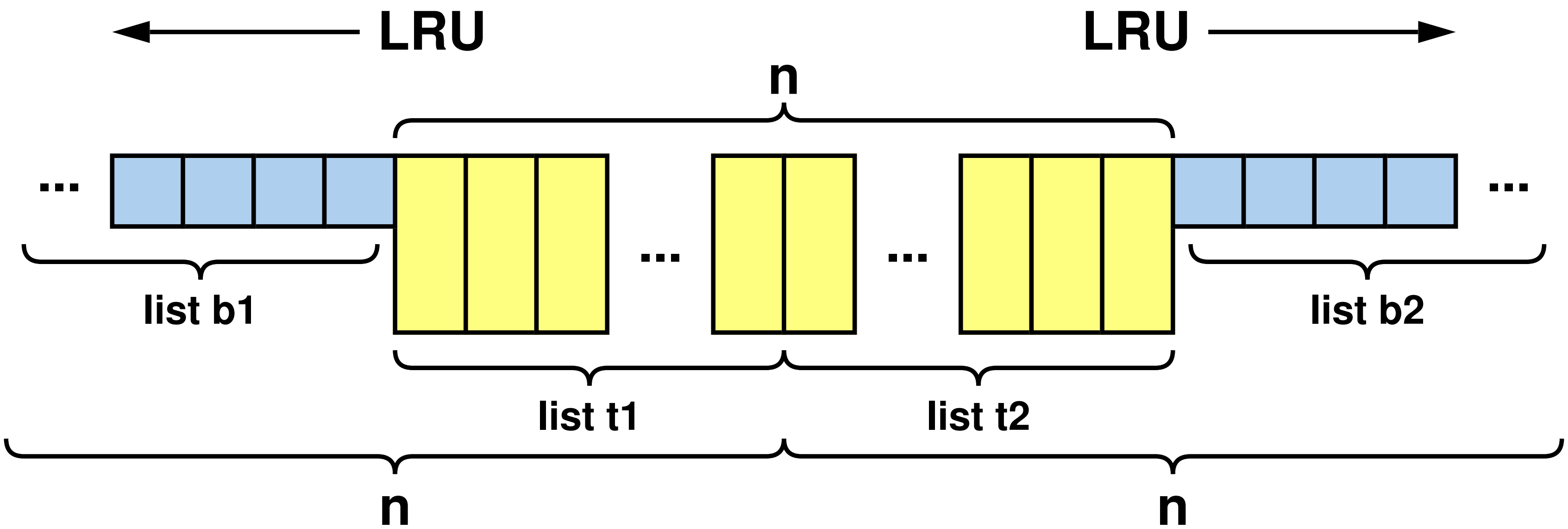
t1 ; b1:

- LRU list of blocks referenced once
- t1 list (most recently used) contain contents
- b1 list (least recently used) contain just references

t2 ; b2:

- LRU list of blocks referenced more than once
- t2 list (most recently used) contain contents
- b2 list (least recently used) contain just references

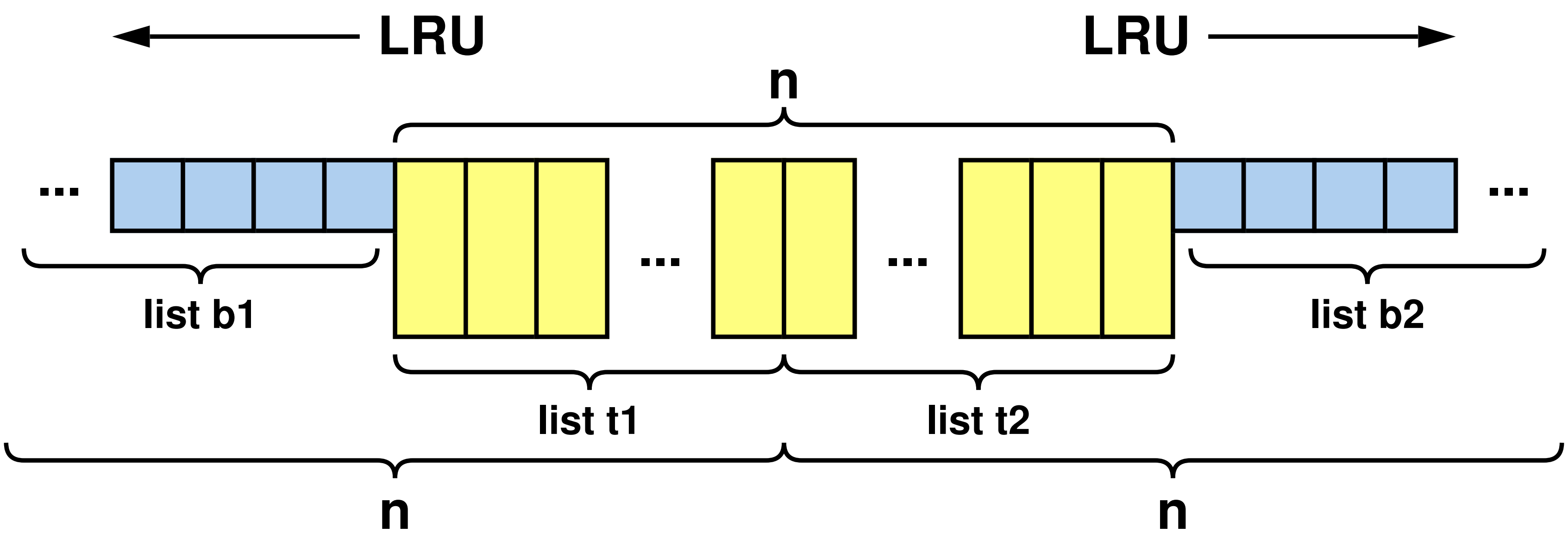
Adaptive Replacement Cache



cache miss:

- if t1 is full
  - evict LRU(t1) and make it MRU(b1)
- referenced block becomes MRU(t1)

Adaptive Replacement Cache



cache hit:

- if in t1 or t2, block becomes MRU(t2)
- otherwise
  - if block is referred to by b1, increase t1 space at expense of t2
  - otherwise
    increase t2 space at expense of t1
  - if t1 is full, evict LRU(t1) and make it MRU(b1)
  - if t2 is full, evict LRU(t2) and make it MRU(b2)
  - insert block as MRU(t2)

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