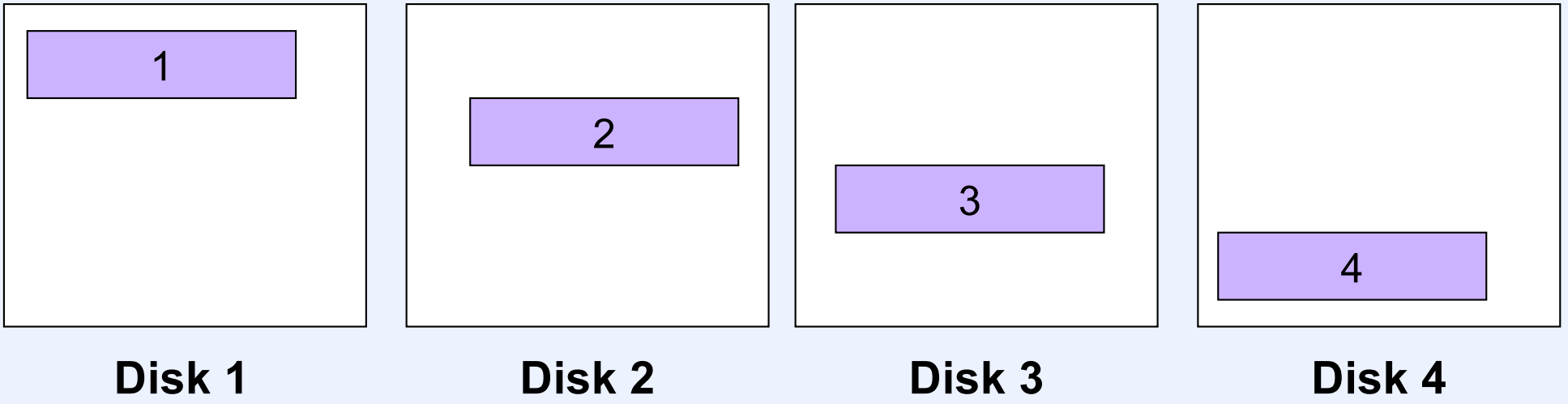
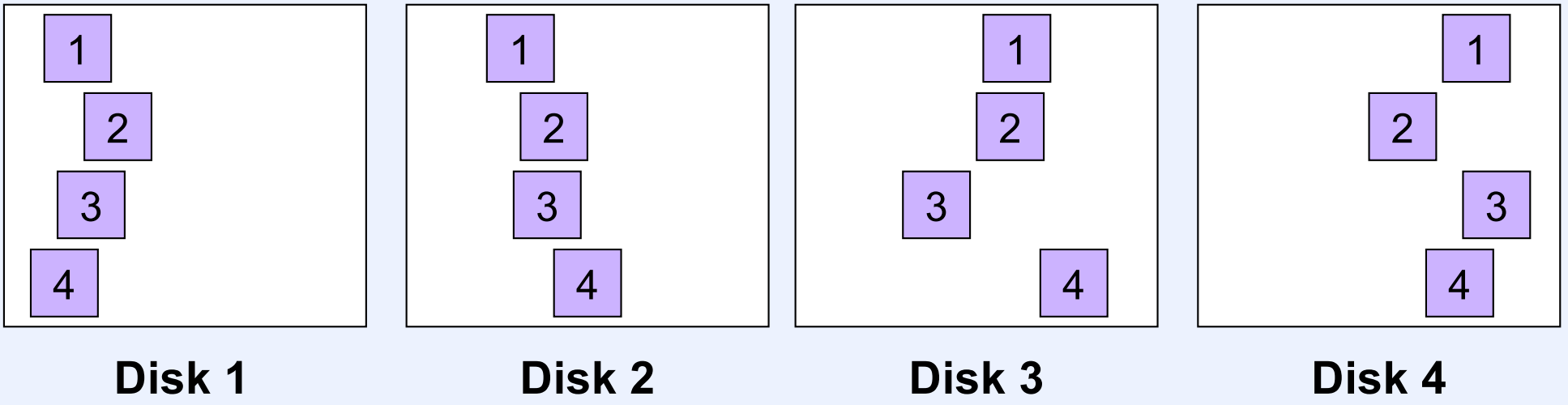
File Systems Part 5

Striping Unit Size





Monday's Quiz

In the previous slide, suppose we have four threads concurrently reading from the disks. The first is reading all the sectors labeled one, the second is reading all the sectors labeled two, the third three, and the fourth four. With which layout would they complete all the transfers the quickest?

a) top

b) bottom

c) roughly equal

Quiz 1

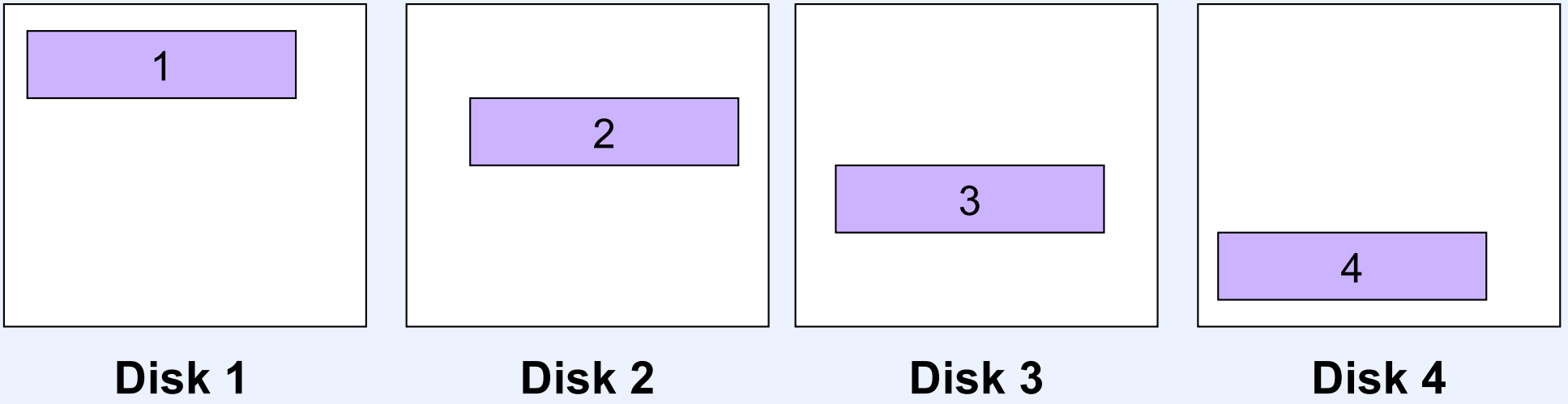
Now suppose we have one thread that first reads the sectors labeled one, then those labeled two, then three, then four. With which layout would it complete the transfers the quickest?

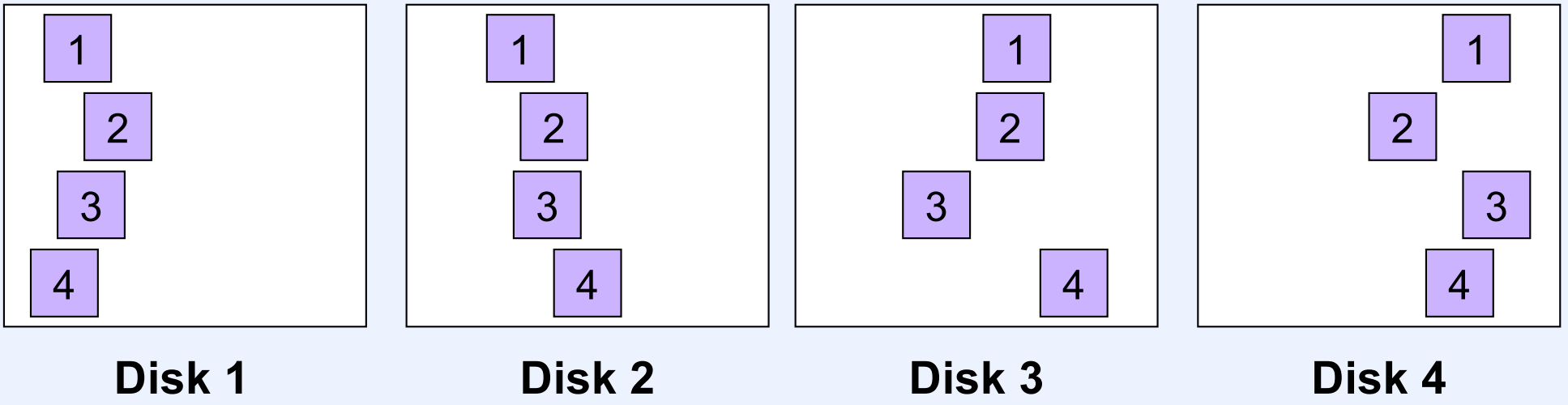
a) top

b) bottom

c) roughly equal

Striping Unit Size (Again)





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's failing is *f*
  - probability of N-disk system's failing is $(1-(1-f)^N)$
  - (assumes failures are IID, which is probably wrong …)

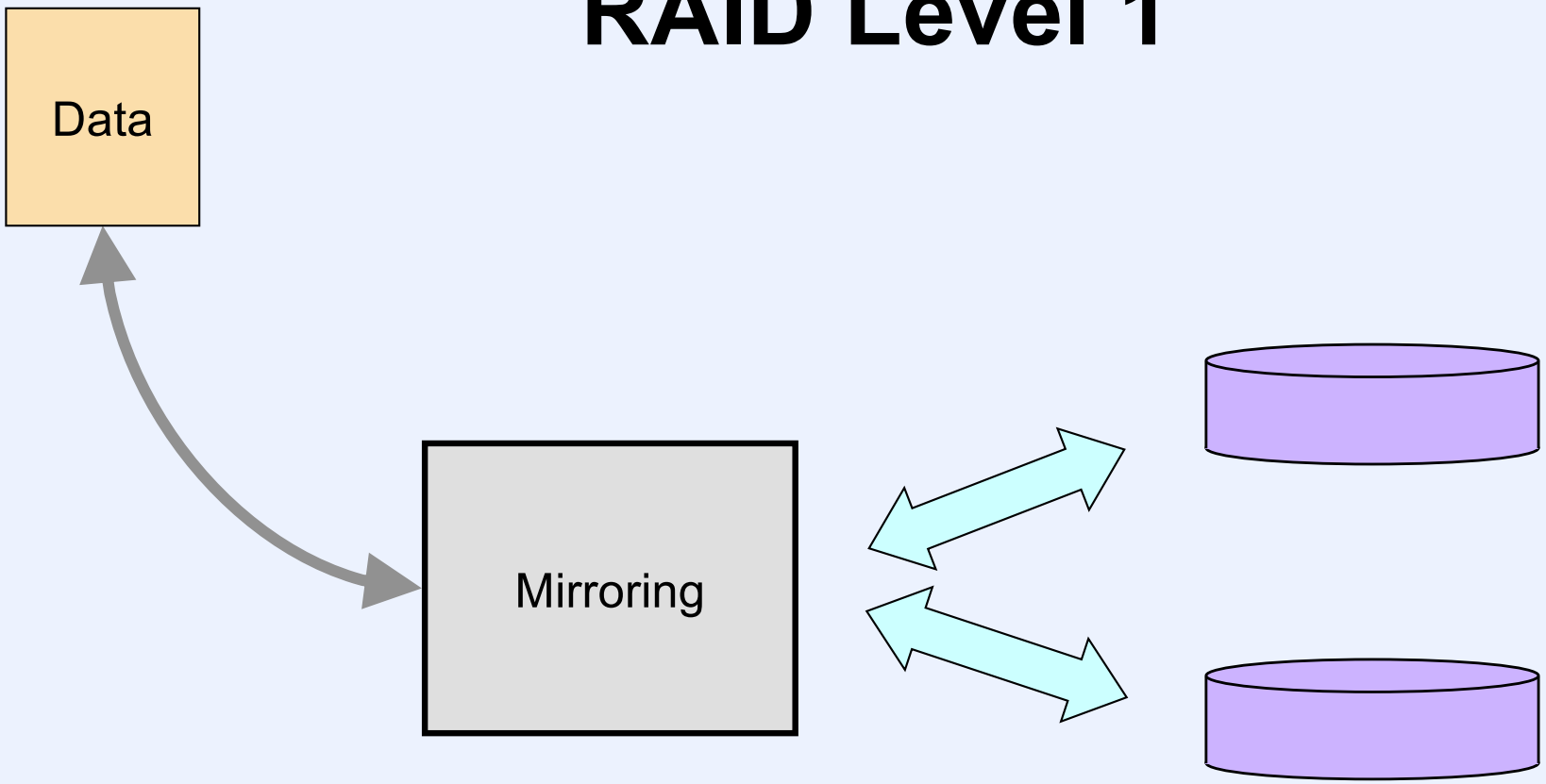
RAID to the Rescue

- 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 5
      - RAID level 0: pure striping
        - numbering extended later
      - RAID level 1: pure mirroring

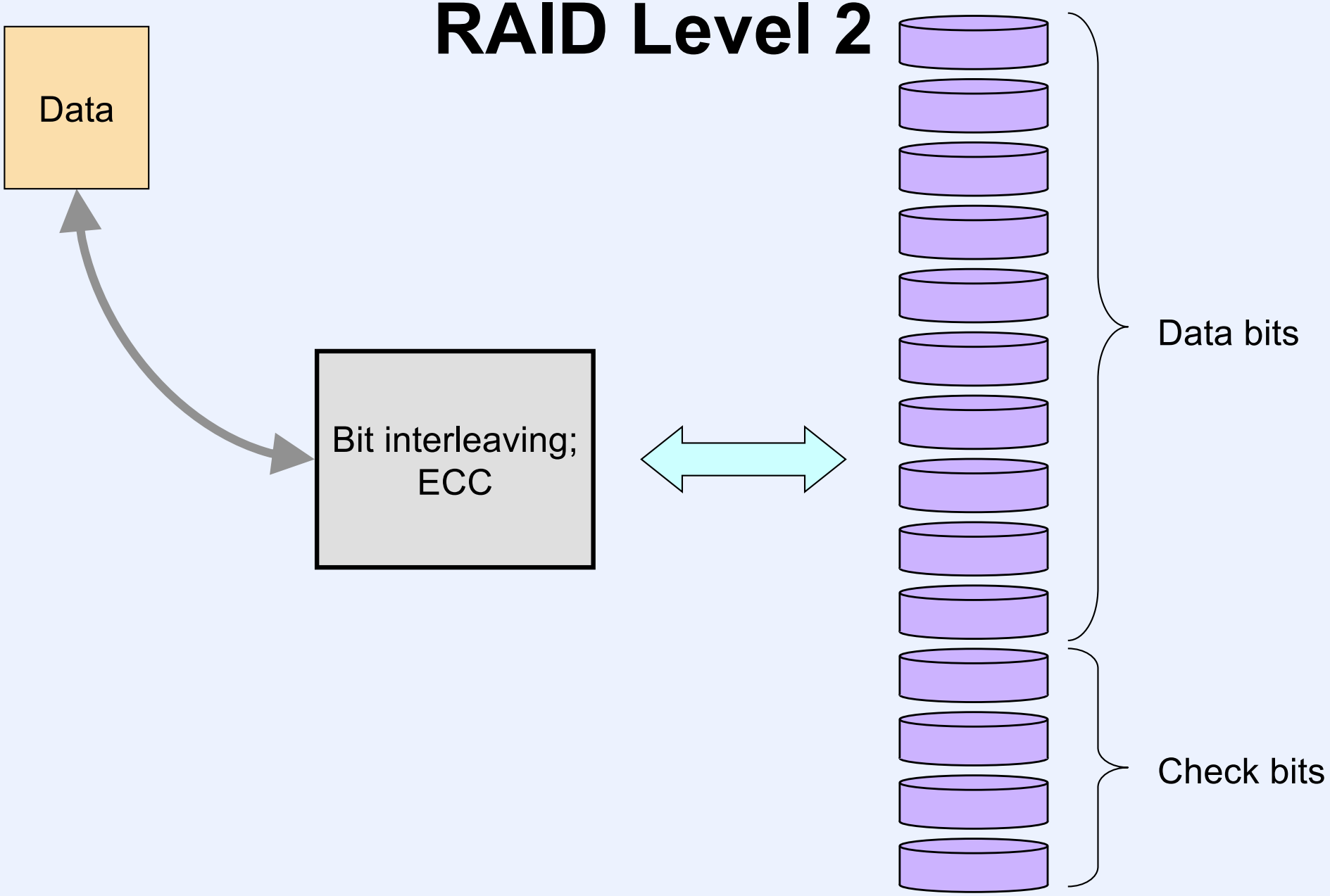
RAID to the Rescue

- 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 5
      - RAID level 0: pure striping
        - numbering extended later
      - RAID level 1: pure mirroring

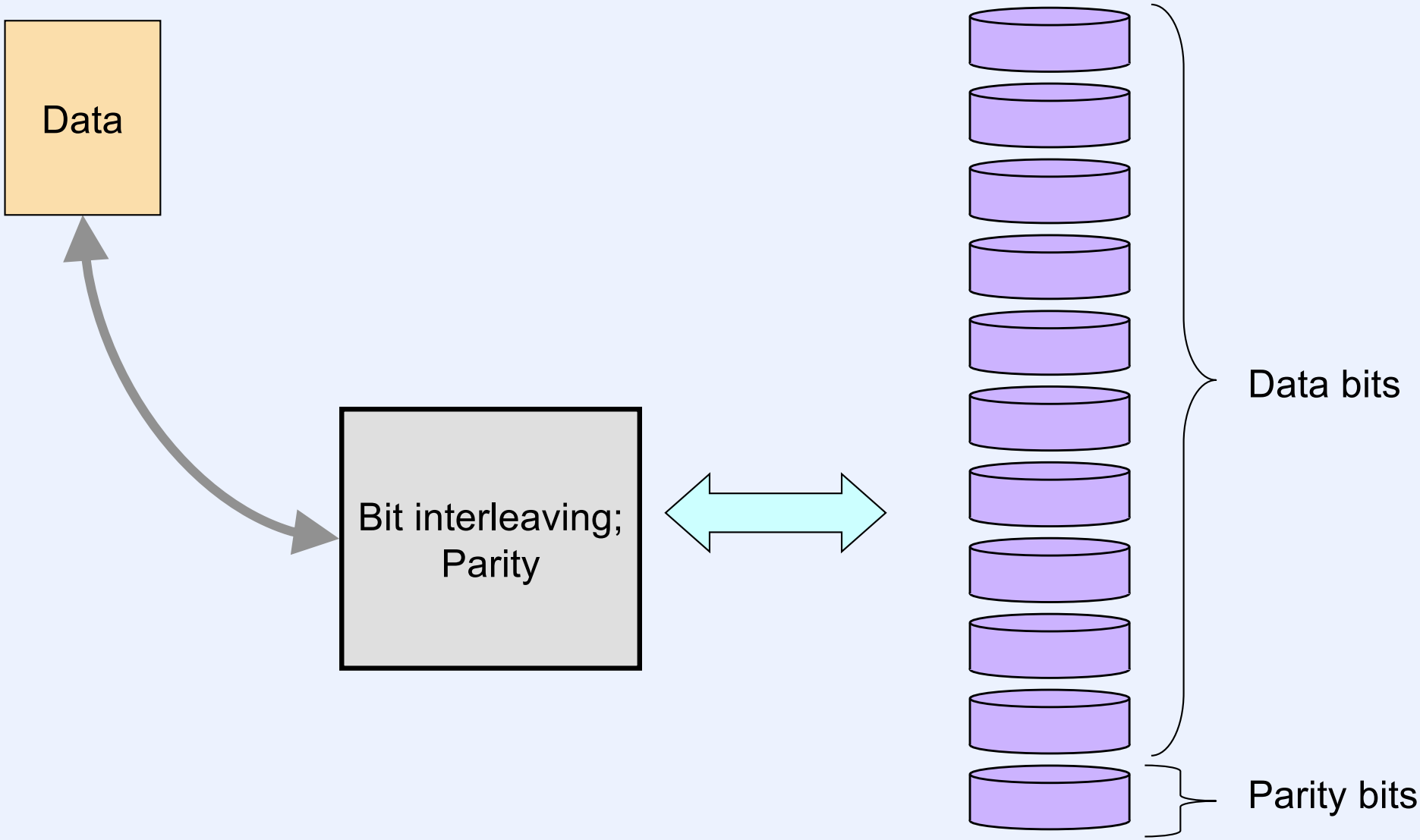
RAID Level 1



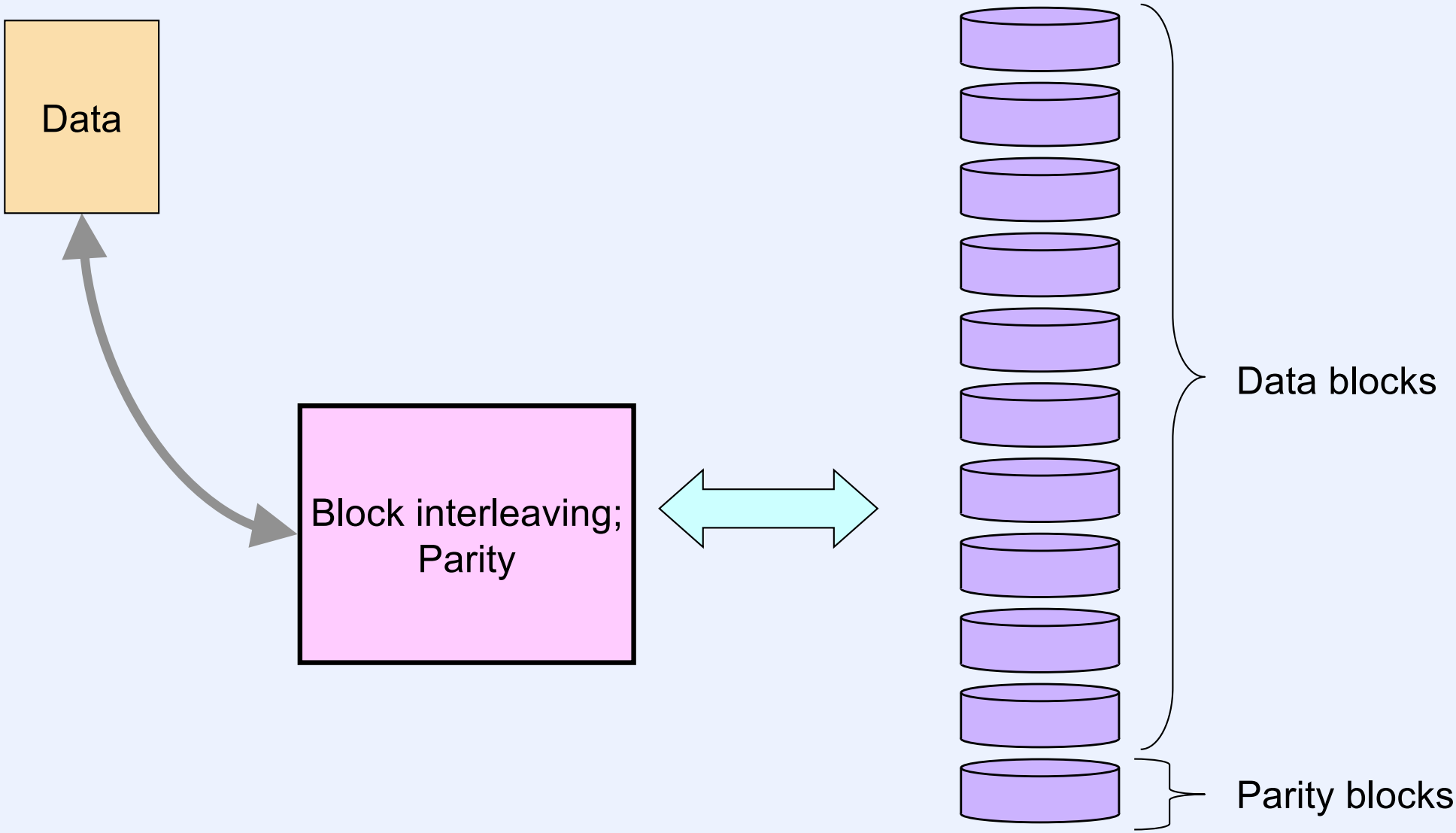
RAID Level 2



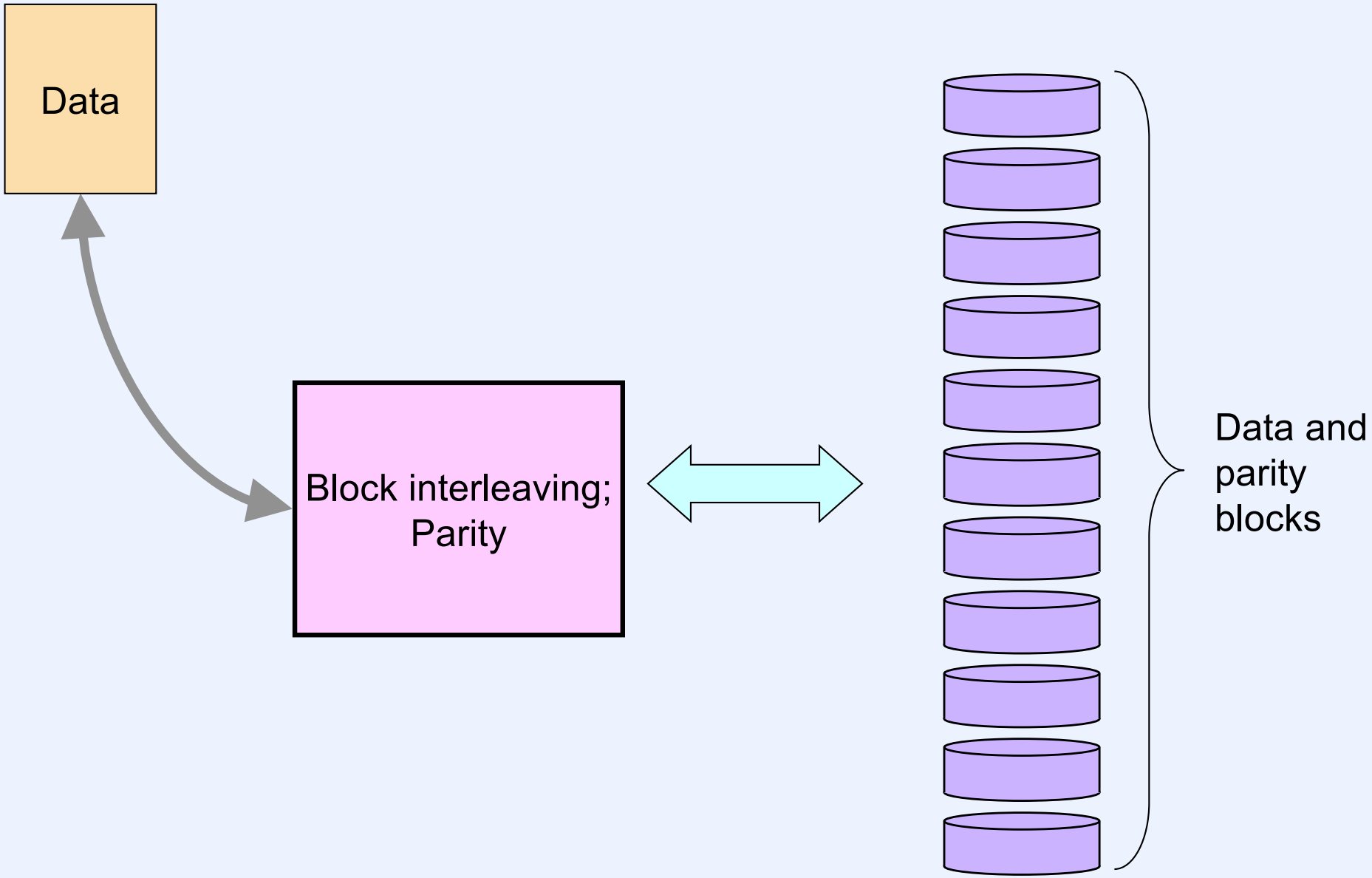
RAID Level 3



RAID Level 4



RAID Level 5



RAID 4 vs. RAID 5

- Lots of small writes
  - RAID 5 is best
- Mostly large writes
  - multiples of stripes
  - either is fine

Quiz 2

We've run out of memory in our RAID system and decide to add a new disk (going from, say, 6 disks to 7 disks). On which system is this easiest to do?

a) Can't be done on either

b) RAID 4

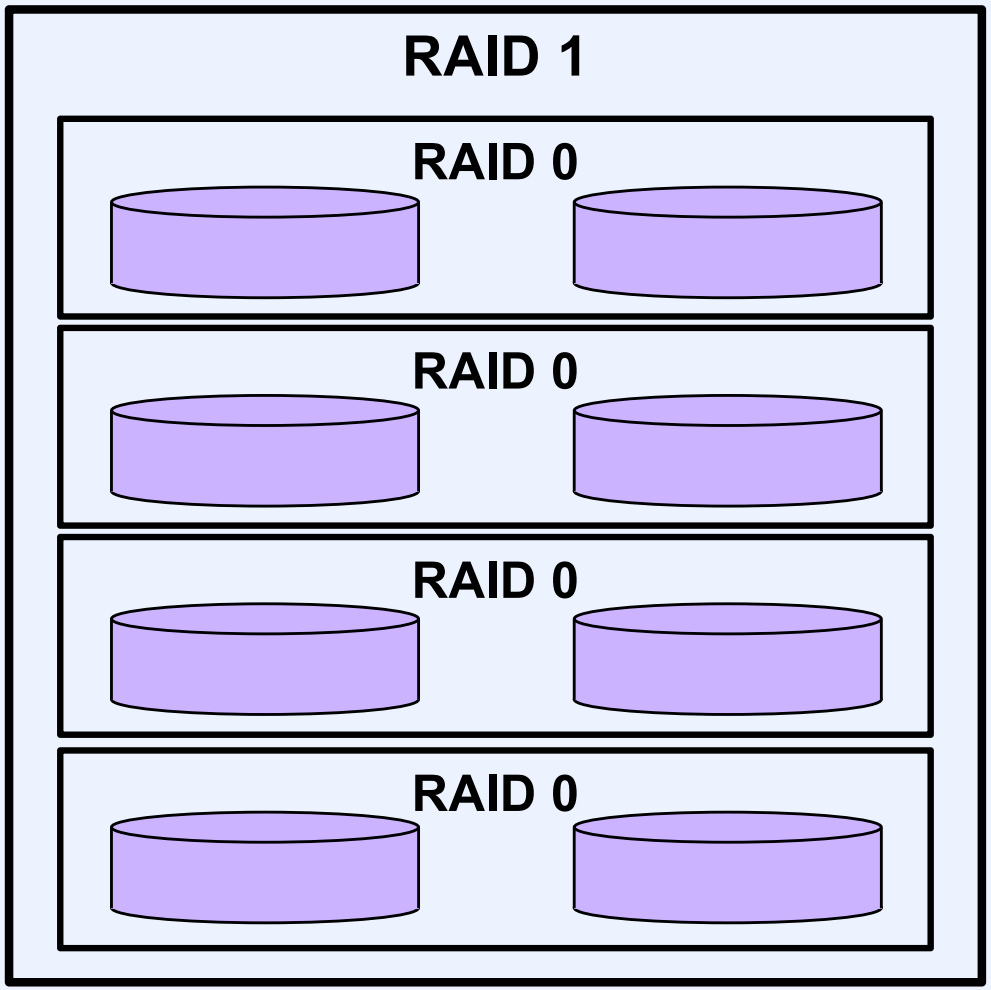
c) RAID 5

d) It's about the same on both

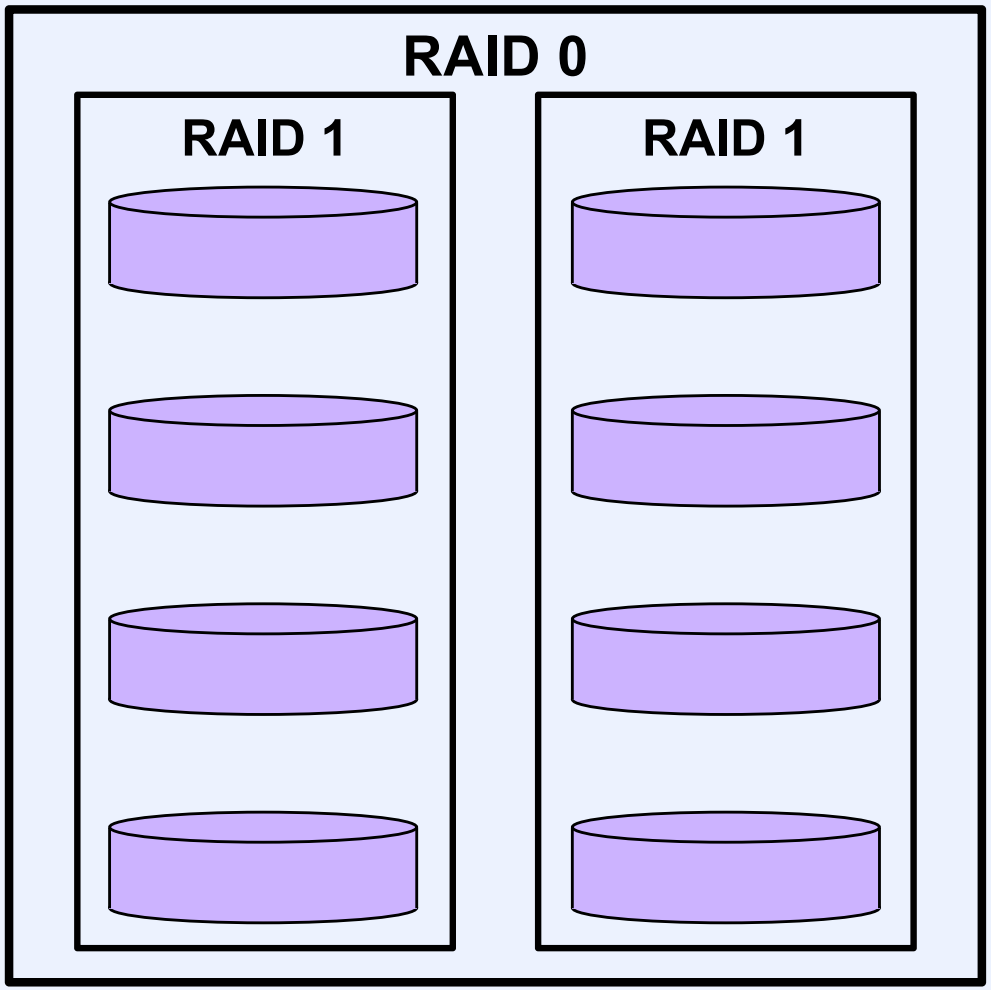
Beyond RAID 5

- 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

RAID 1+0



RAID 0+1



Quiz 3

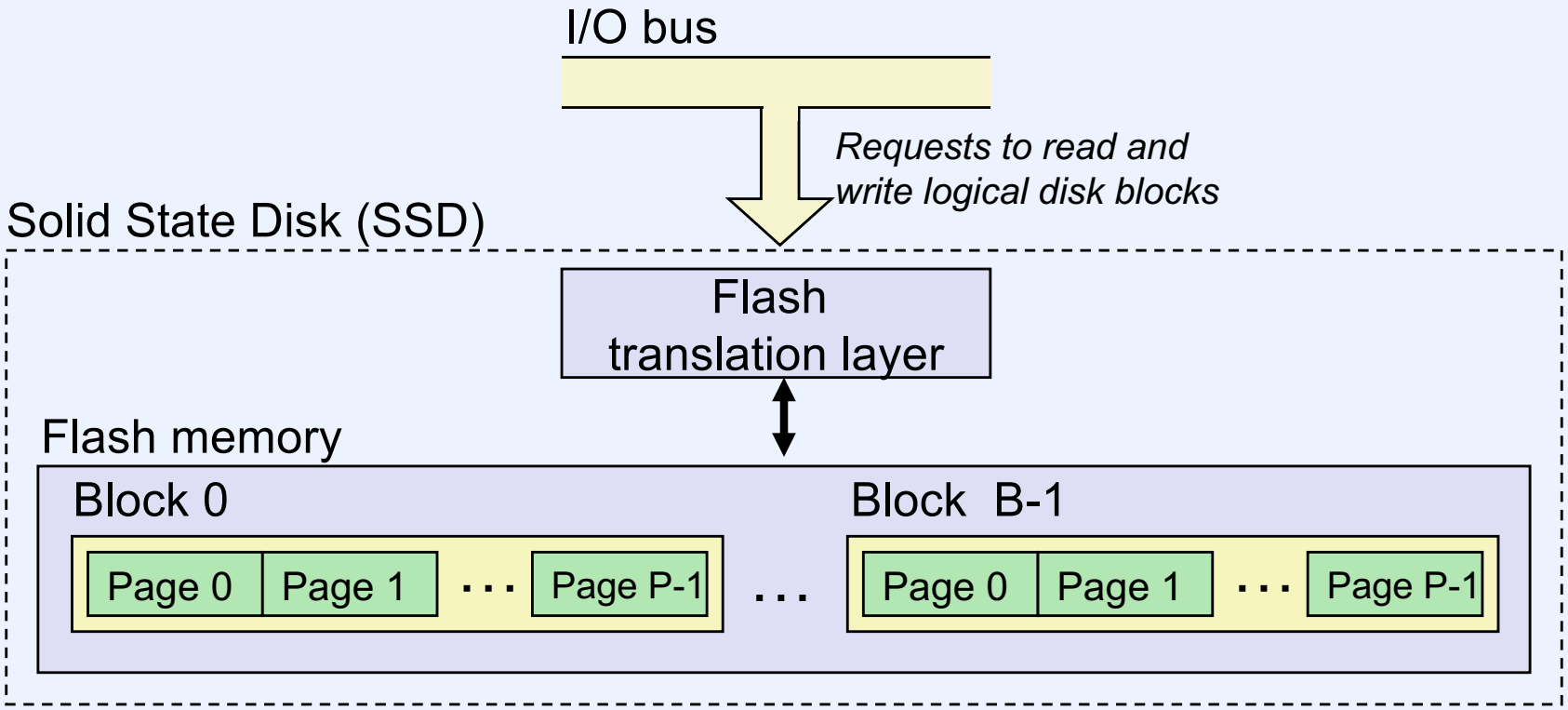
Which of RAID 1+0 and 0+1 is more resilient to failure? (Hint: consider the situation after one disk has failed. How likely is it that a failure of a second disk would make the system unusable?)

a) They have similar resilience

b) RAID 1+0

c) RAID 0+1

Solid-State Disks (SSDs)



- **Pages: 512KB to 4KB; blocks: 32 to 128 pages**
- **Data read/written in units of pages**
- **Page can be written only after its block has been erased**
- **A block wears out after 100,000 repeated writes**

Pros and Cons

Pro

- Flash block ≈ file-system block
- ~Random access
- Low power
- Vibration-resistant

Con

- Limited lifetime
- Writes can be expensive
- Cost more than disks (but not for much longer)
  - 4TB SSD: ~$250
  - 4TB HDD: ~$100

Flash Memory

- Two technologies
  - nor
    - byte addressable
  - nand
    - page addressable
- 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

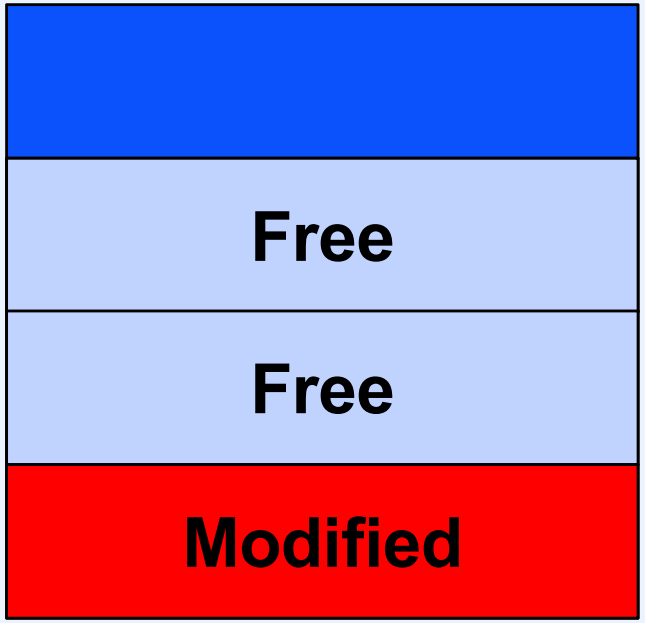
- Wear leveling
  - spread writes (erasures) across entire drive
- Flash translation layer (FTL)
  - specification from 1994
  - provides disk-like block interface
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling

SSD Performance Characteristics

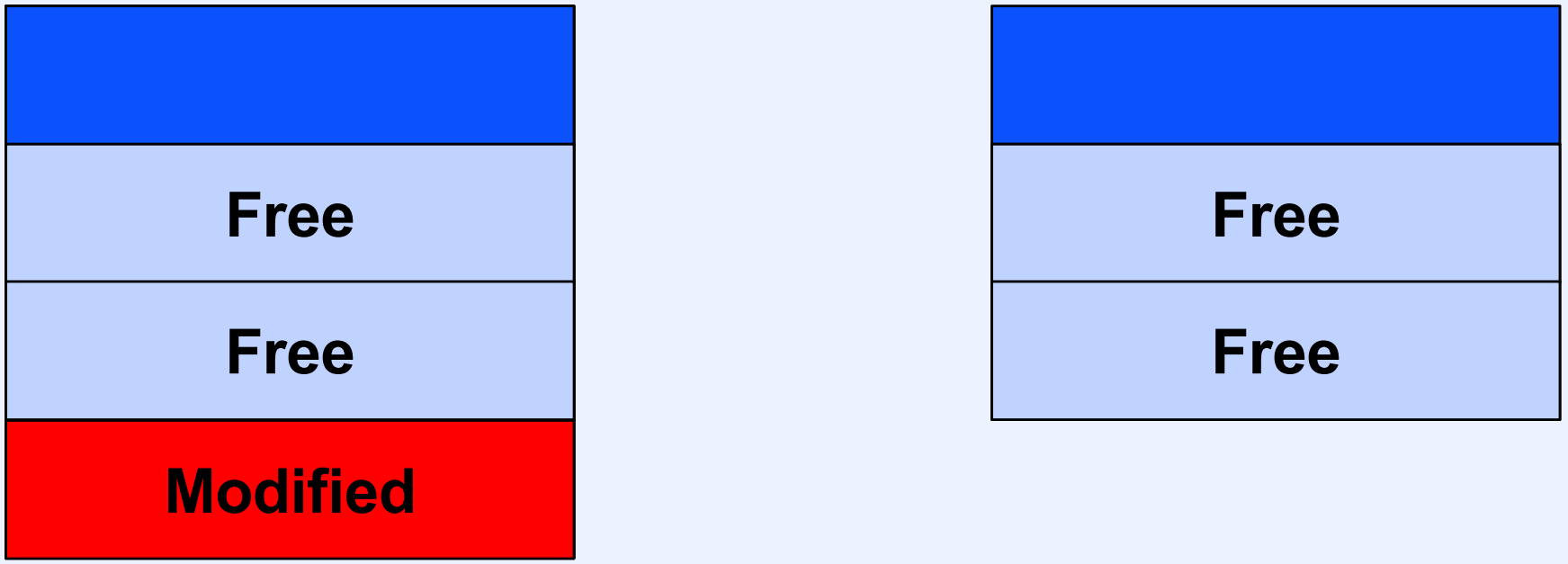
Sequential read tput	250 MB/s	Sequential write tput	170 MB/s
Random read tput	140 MB/s	Random write tput	14 MB/s
Random read access	30 us	Random write access	300 us

- Why are random writes so slow?
  - erasing a block is slow (around 1 ms)
  - modifying a page triggers a copy of all useful pages in the block
    - find a used block (new block) and erase it
    - write the page into the new block
    - copy other pages from old block to the new block

A Problem Case (1)

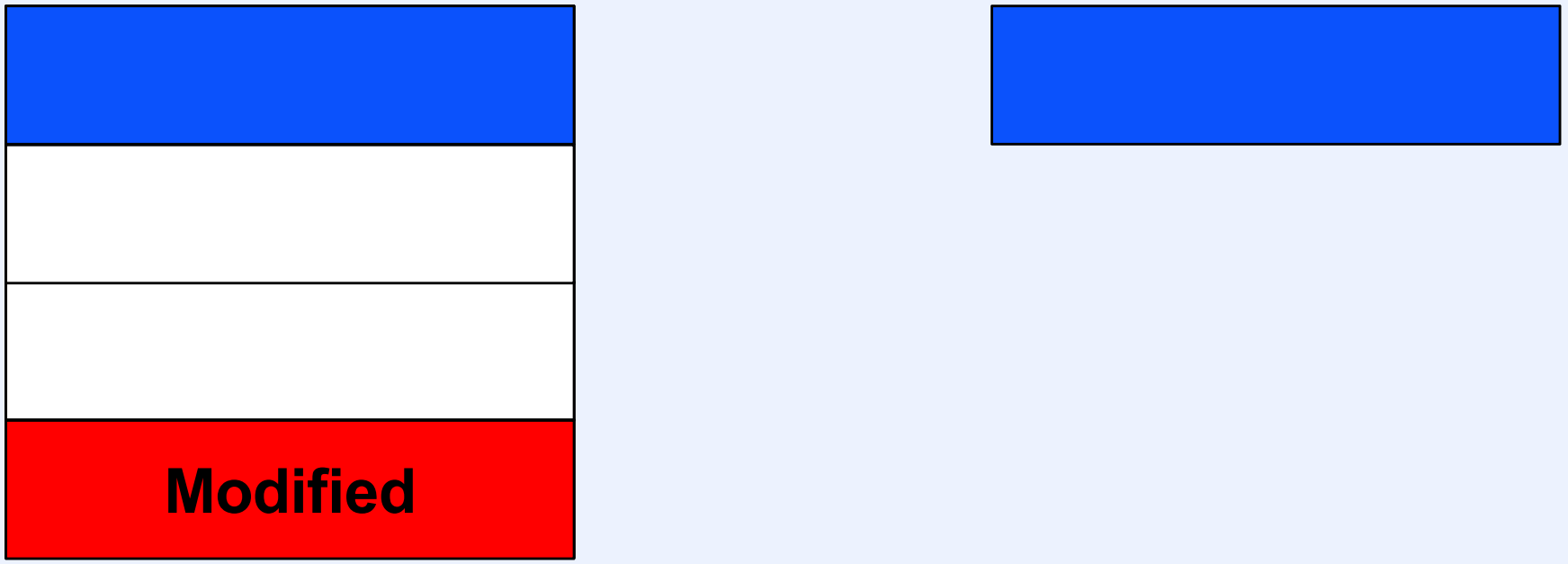


A Problem Case (2)



1) Copy
2) Erase
3) Copy and modify

Trimming



1) Copy
2) Erase
3) Copy and modify

Flash with FTL

- Which file system?
  - FAT32 (sort of like S5FS, but from Microsoft)
  - NTFS
  - FFS
  - Ext3
  - Ext4
- All were designed to exploit disks
  - much of what they do is irrelevant for flash

Flash without FTL

- Known as memory technology device (MTD)
  - software wear-leveling
  - allows device-wide management of blocks via file system

JFFS and JFFS2

- Journaling flash file system
  - log-based: no journal!
    - updates and new data/metadata written to log sequentially
      - each log entry contains inode info and some data
    - log grows into pre-erased blocks
    - garbage collection copies info out of partially obsoleted blocks, allowing block to be erased
    - complete index of inodes kept in RAM
      - entire file system must be read when mounted

Space Allocation and Garbage Collection

- Space allocated in pages
  - allocation in blocks results in too much external fragmentation
- "Garbage-collect" partially full blocks, coalescing used pages into full blocks
  - causes blocks to move, and thus references to blocks must be modified (resulting in further erasures, etc.)

Garbage Collection

- Background thread scans through all blocks, looking for partial allocations
- Goal is to produce a combination of completely full and completely empty blocks
- For each page, must determine what file it belongs to and where in that file
  - stored with the page as additional metadata
- Need a page index to determine for each piece of a file where it is located
  - changes as pages move
- B+ tree maintained to do this

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

Quiz 3

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 4

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

Apple Fusion Drives (FD)

- SSD used along with hard drive
- Implemented in the LVM
  - total capacity is sum of disk and SSD capacities
    - works with both HFS+ and ZFS
  - SSD used to buffer all incoming writes
  - data is moved from disk to SSD if used sufficiently often
    - migration happens in background

FD Observations

- Implementation is in the LVM, i.e., below the file system
  - all decisions based on block access
- 4GB available on SSD for writes
  - all writes go to SSD while there's space
  - otherwise go to HDD
- Frequent reads trigger promotion to SSD
- Data transferred between SSD and HDD in units of 128KB

FD Write

```
if !onSSD(block) {
        if SSDfreeSpace > 0 {
                remap(block)
                writeOnSSD(block)
        } else {
                writeOnHDD(block)
        }
} else {
        writeOnSSD(block)
}
```

FD Read

```
Update_Usage(block)
if onSSD(block)
    readFromSSD(block)
else
    readFromHDD(block)
```

FD Background Activities

```
when (accessThresholdReached(block)) {
    if SSDfreeSpace > 0 {
        remap(block)
        writeOnSSD(block)
    }
}

when(SSDFreeSpace < 4GB) {
    for each infrequent block {
        remap(block)
        writeOnHDD(block)
    }
}
```

FD Block Map

- Vital data structure
- Kept up to date on SSD
- Perhaps backed up on HDD

Case Studies

- NTFS
- WAFL
- ZFS
- APFS

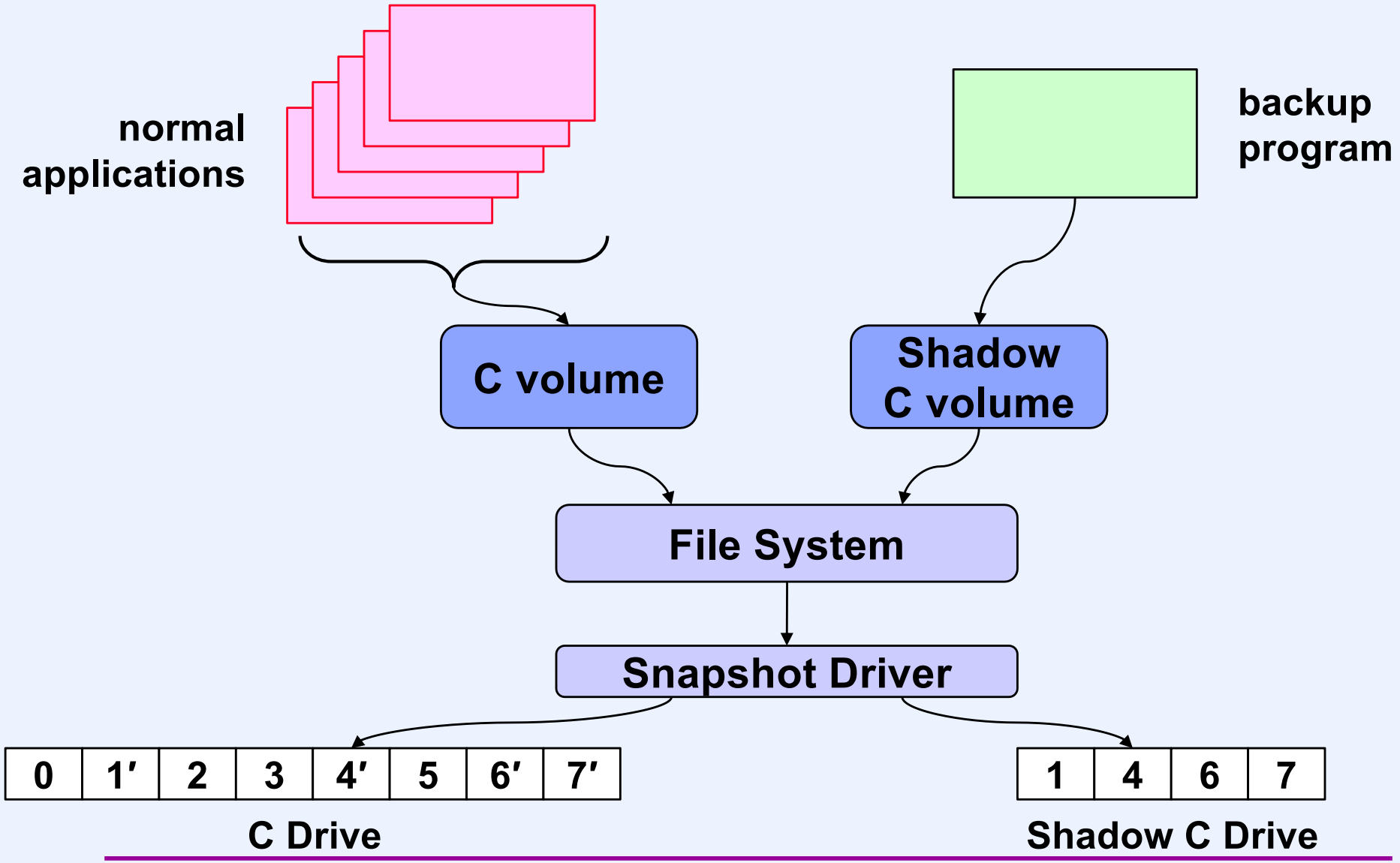
NTFS

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

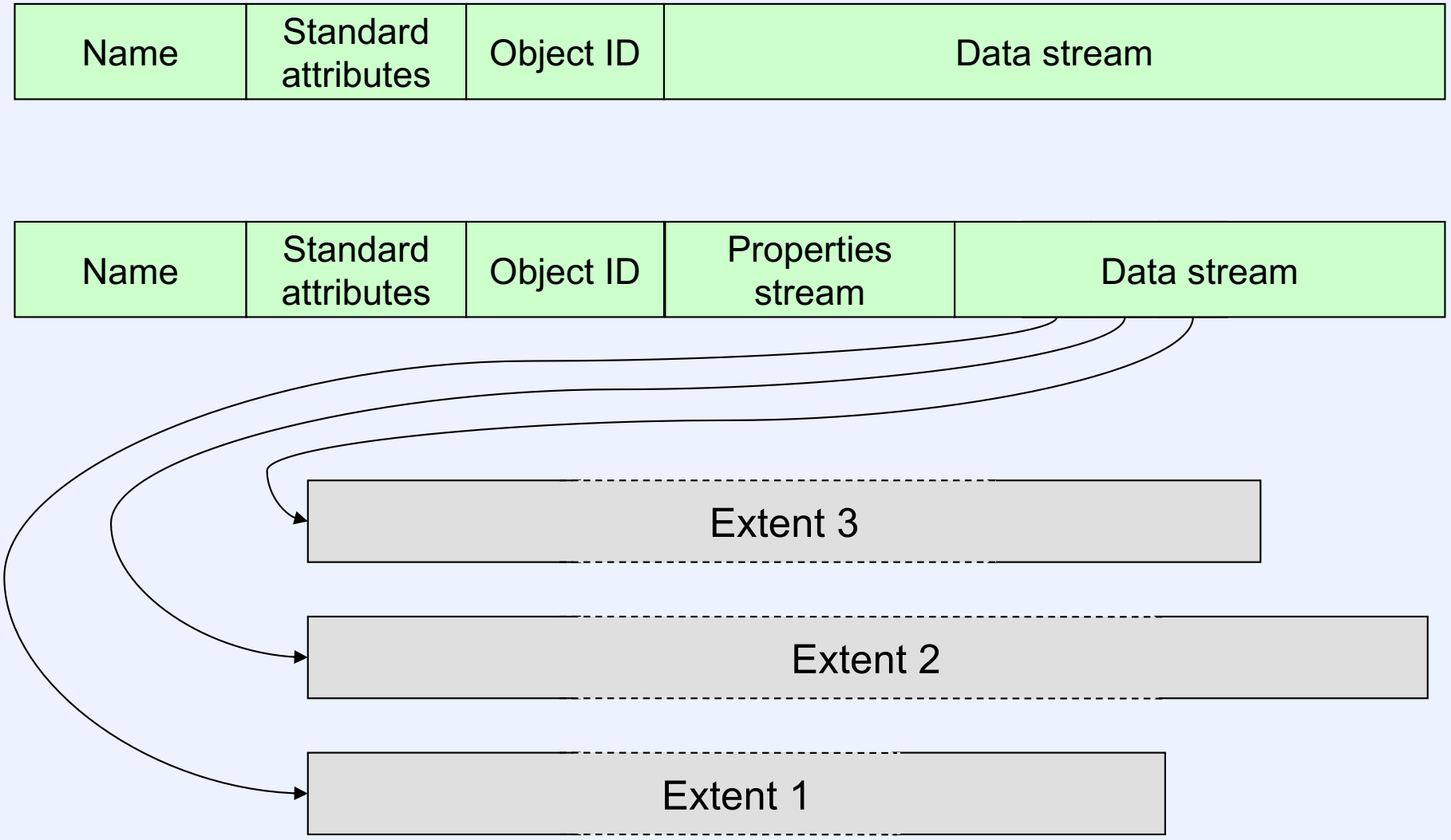
Backups

- 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 SMB requests
- Utilizes shadow paging and log-structured writes
- Provides snapshots

WAFL and RAID