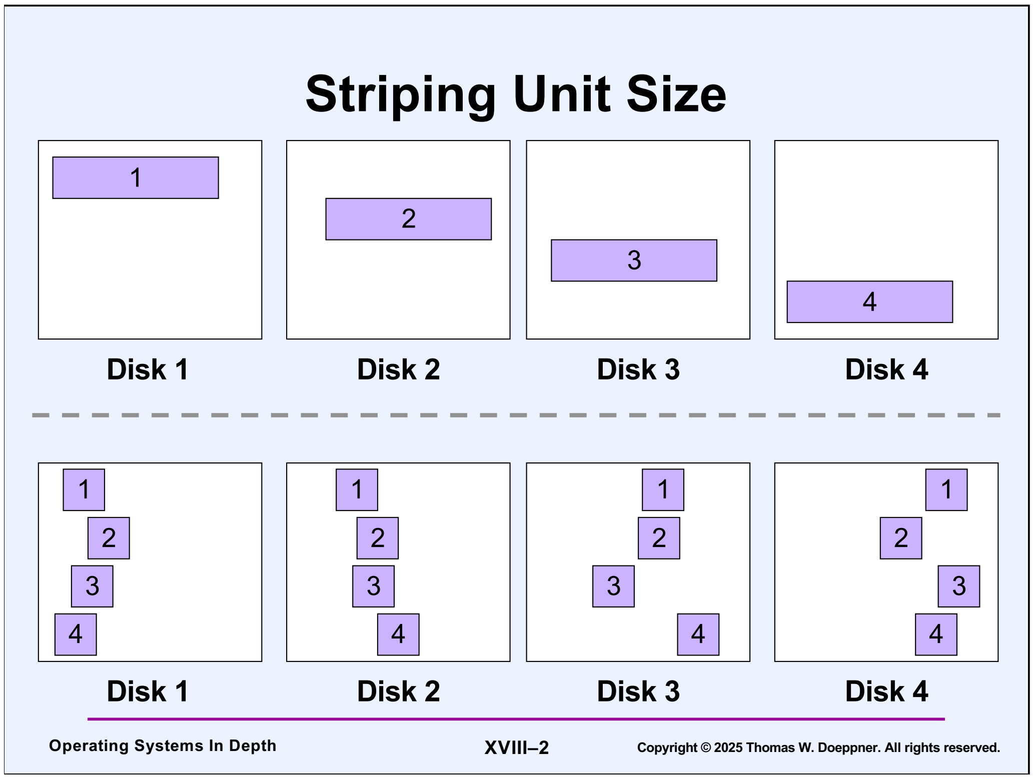
File Systems Part 5



Striping Unit Size

If we have only one waiting disk request at a time (a **concurrency factor** of 1), then a smaller striping unit is typically better than a larger one, since we can spread one request across lots of disks. But with more than one waiting disk request (a larger **concurrency factor**), it may make sense to have larger striping units, as the slide illustrates.

The top half of the slide shows four disks with a four-sector striping unit. Suppose we have concurrent requests for the four data areas shown, each four sectors in length. The four requests can be handled in roughly the time it takes to handle one, since the positioning for each of the requests can be done simultaneously as can the data transfer.

The bottom half of the slide shows four disks with a one-sector striping unit. We have the four concurrent requests for the same data as before, but in this case each item is spread across all four disks, one sector per disk. Handling each request requires first positioning the heads on all four disks for the first, then positioning the heads on all four disks for the second, and so forth. Thus the total positioning delays are four times that of the top half of the figure, which has a larger striping unit.

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

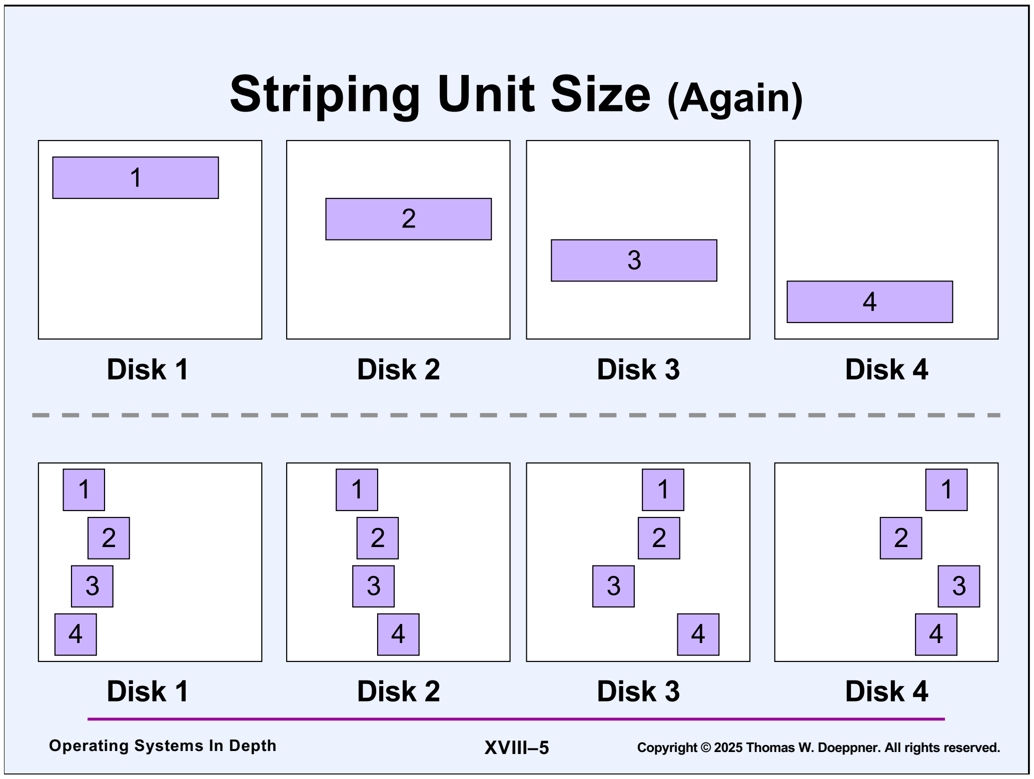
Quiz 1

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

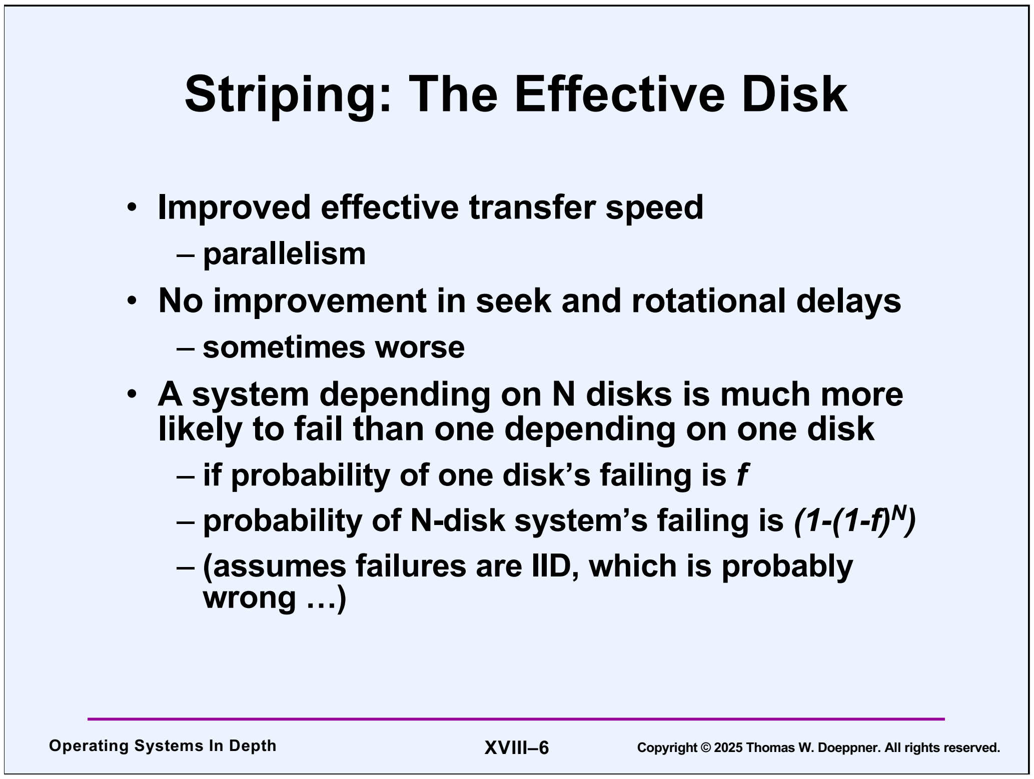
c) roughly equal



Striping Unit Size (Again)

Getting back to the case of just one waiting disk request (a concurrency factor of 1), suppose we have one thread that's first reading the data labelled one, then two, etc. In the layout of the top part of the slide, it would, for each disk in turn, wait for positioning delays and then transfer four sectors. Thus the total time required would be, roughly, four times the typical seek and rotational delays plus the transfer time for 16 total sectors.

For the layout of the bottom half of the slide, The thread would read the areas labelled one from all four disks, then the areas labelled two from all four disks, etc. Since the thread is reading the areas sequentially (first all of one, then all of two, etc.), it would, for each area, have to wait the maximum of the seek and rotational delays for the four disks, then the time to transfer one sector (since the four sectors for each area are transferred in parallel). This results in a total time of four positioning delays, plus the time required to transfer four sectors (totaling 16 sectors since four are transferred at a time). This will generally be considerably faster than for the non-striped case, under the assumption that the positioning delays don't completely dominate the transfer times.



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 …)

IID = independent and identically distributed.

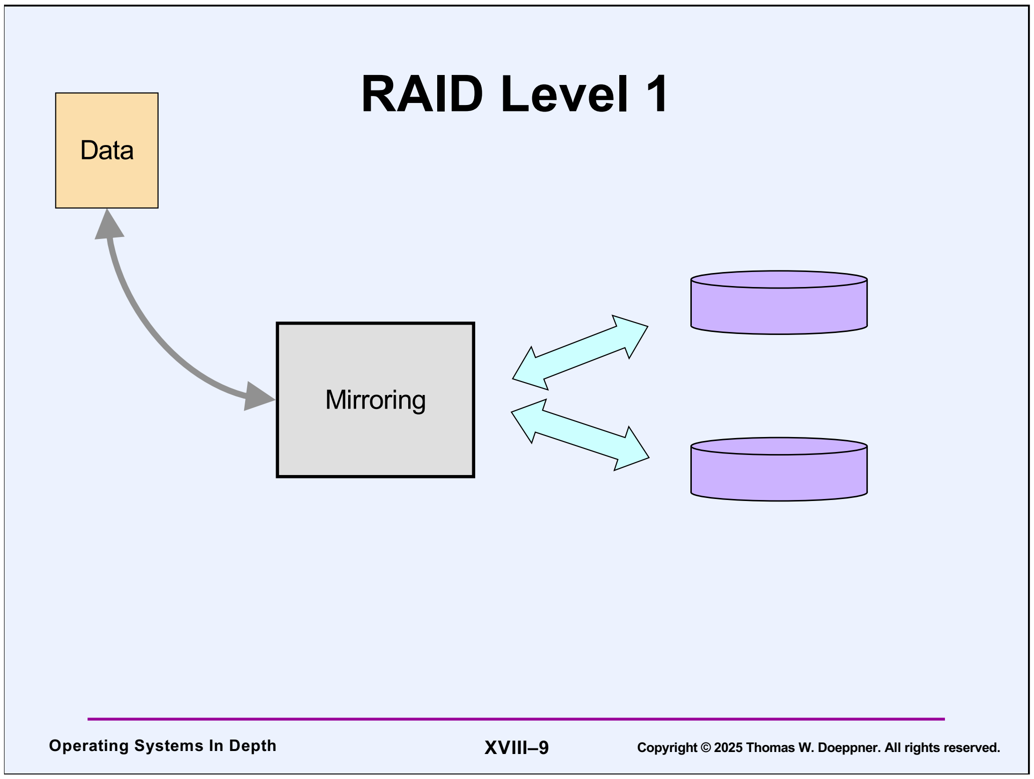
If the probability of one disk's failing is f, the probability of its not failing is 1-f. If we have two disks, the probability of their both not failing is $(1-f)^2$. Thus the probability that none of N disks fail is $(1-f)^N$. And thus the probability that at least one of them fails is 1-$(1-f)^N$.

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

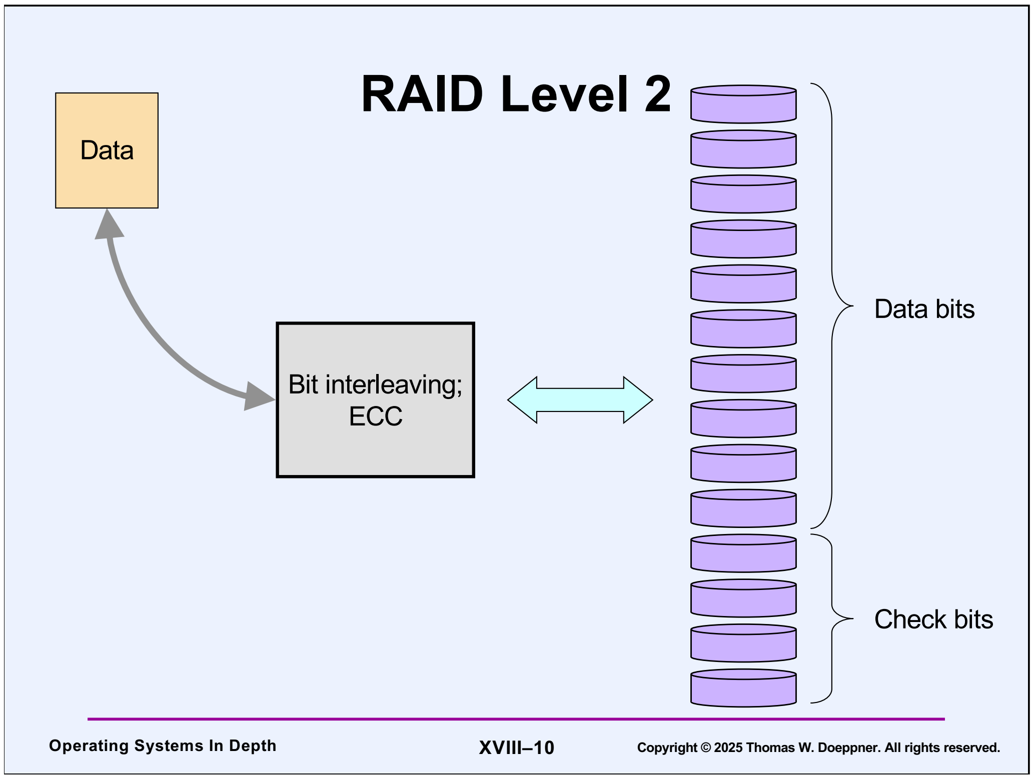
RAID to the Rescue

- 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



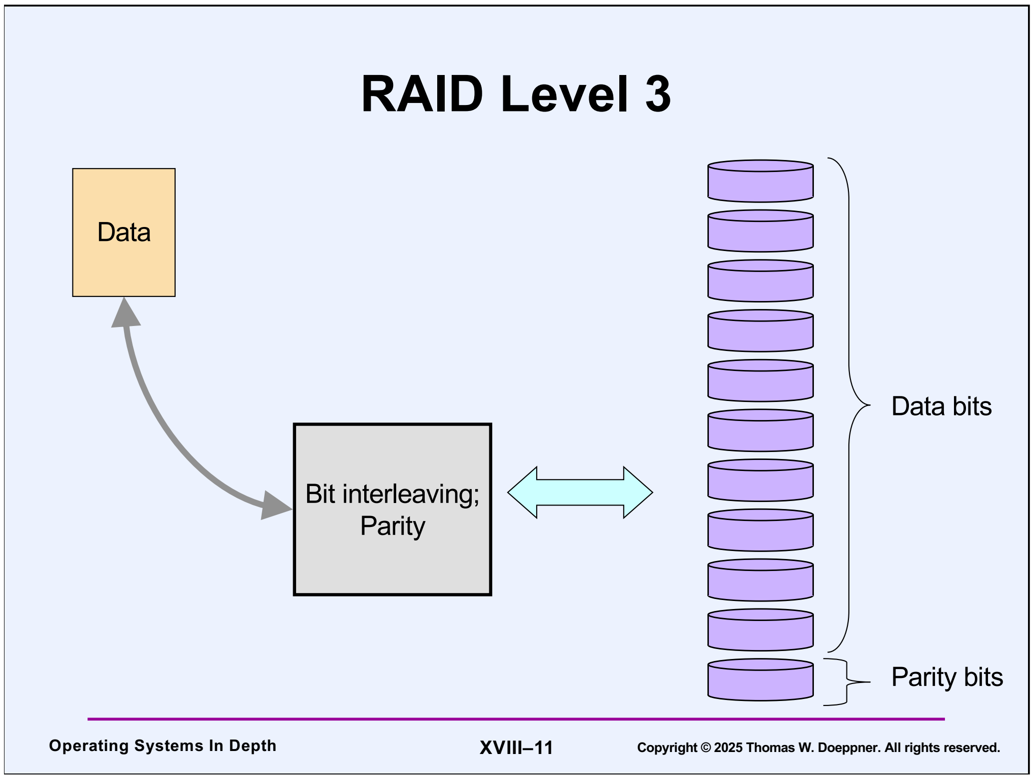
RAID Level 1

RAID level 1: mirroring.



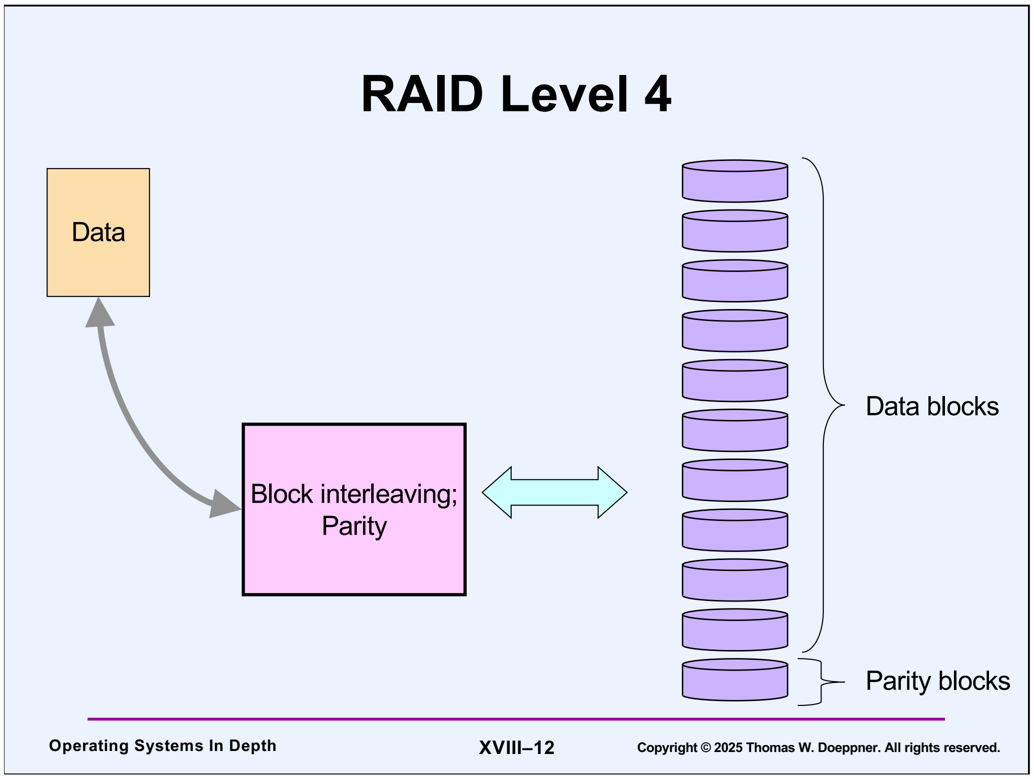
RAID Level 2

RAID level 2: bit interleaving with an error-correcting code.



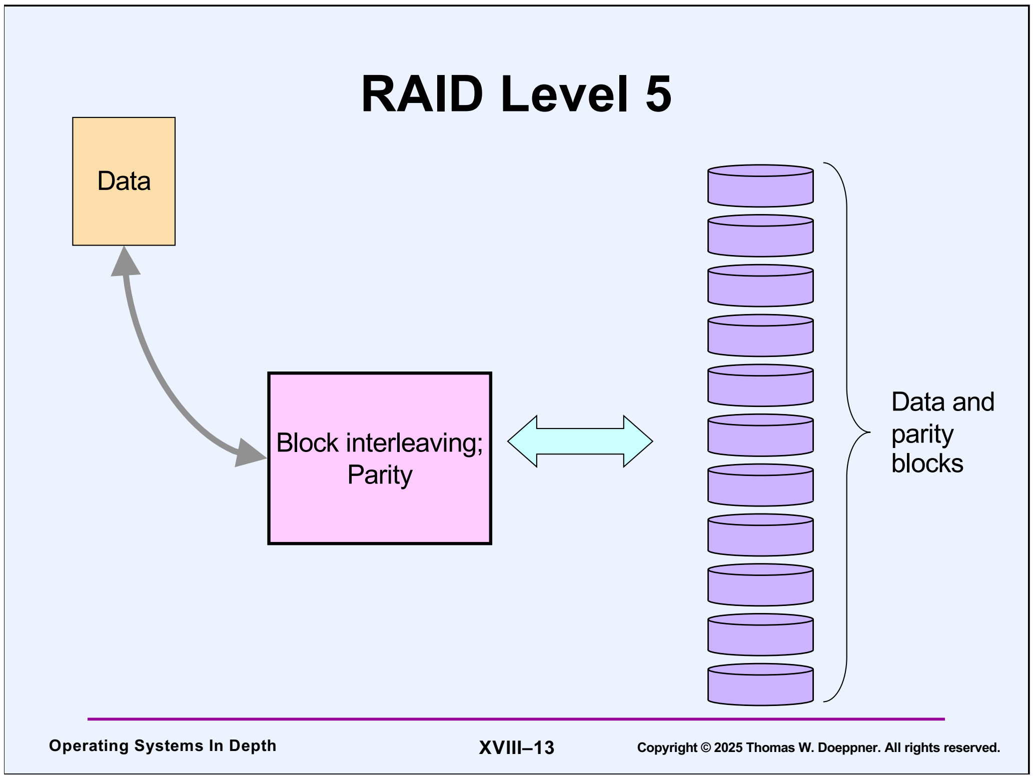
RAID Level 3

RAID level 3: bit interleaving with parity bits.



RAID Level 4

RAID level 4: block interleaving with parity blocks. Note that an update to any of the disks holding data blocks requires an update to the disk holding the parity blocks, causing it to be a bottleneck.



RAID Level 5

RAID level 5: block interleaving with parity blocks. Rather than dedicating one disk to hold all the parity blocks, the parity blocks are distributed among all the disks. For stripe 1, the parity block might be on disk 1; for stripe 2 it would be on disk 2, and so forth. If we have eleven disks, then for stripe 12 the parity block would be back on disk 1.

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

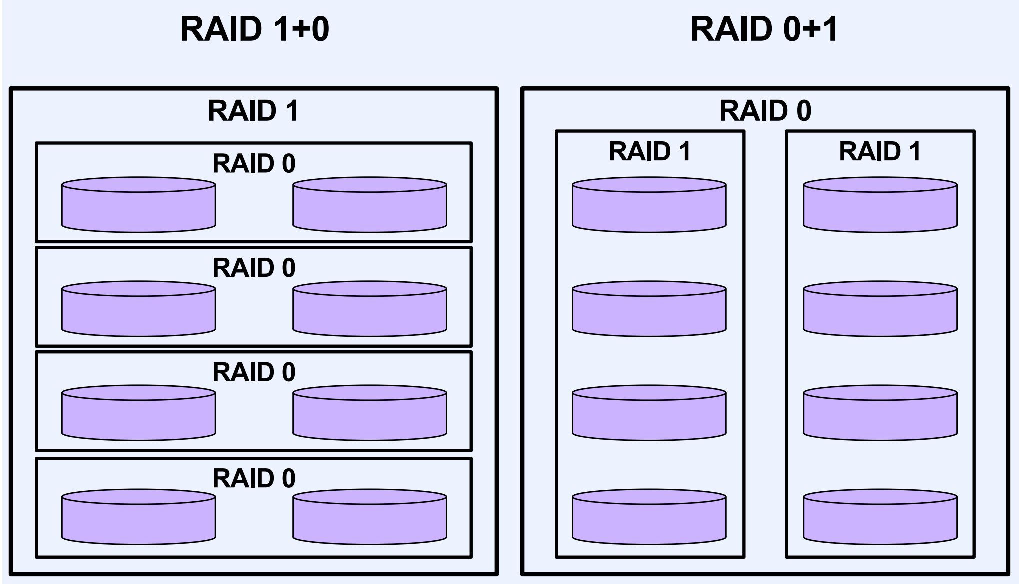
b) RAID 4

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

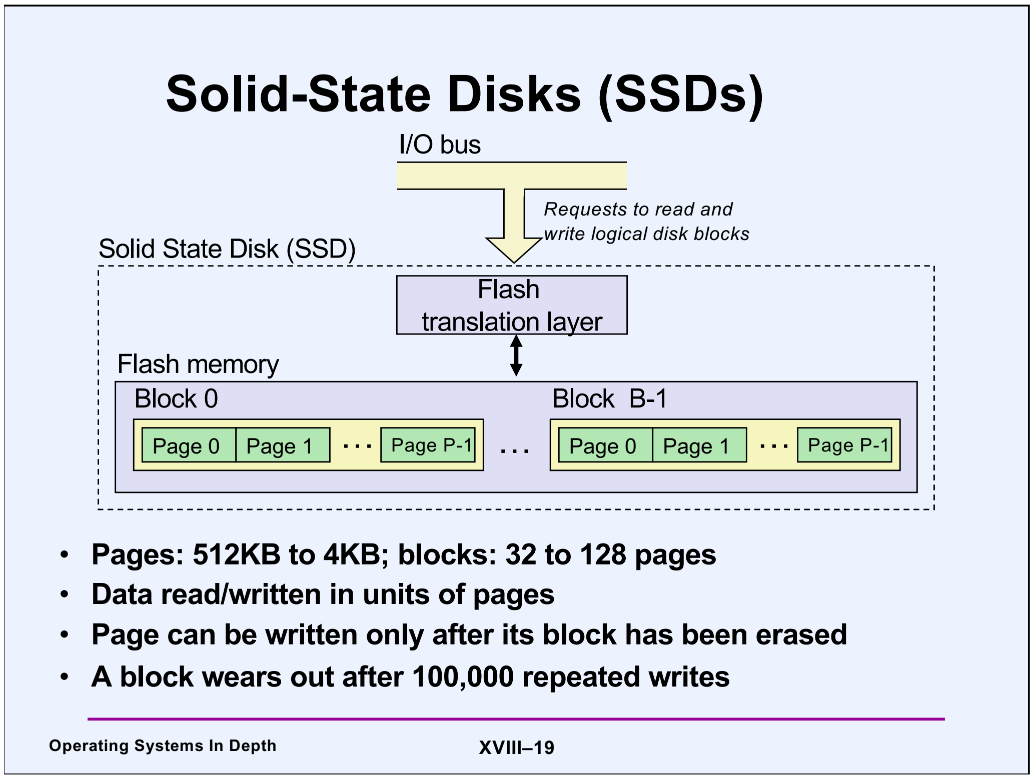


Here is a depiction of how RAID 1+0 is organized (striping on mirrored pairs) and how RAID 0+1 is organized (mirrored stripes).

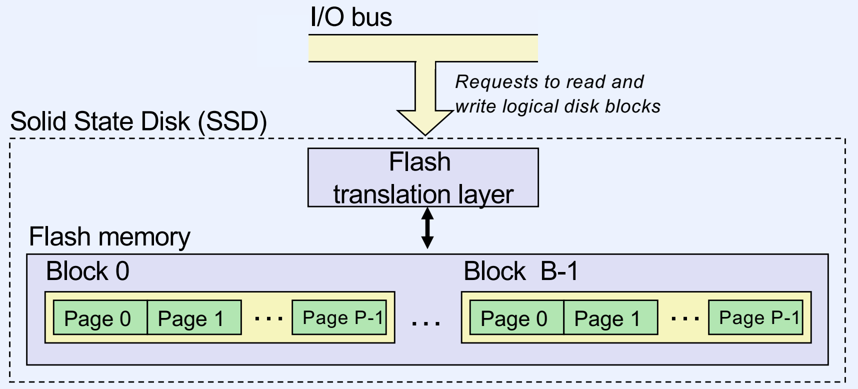
Quiz 3

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

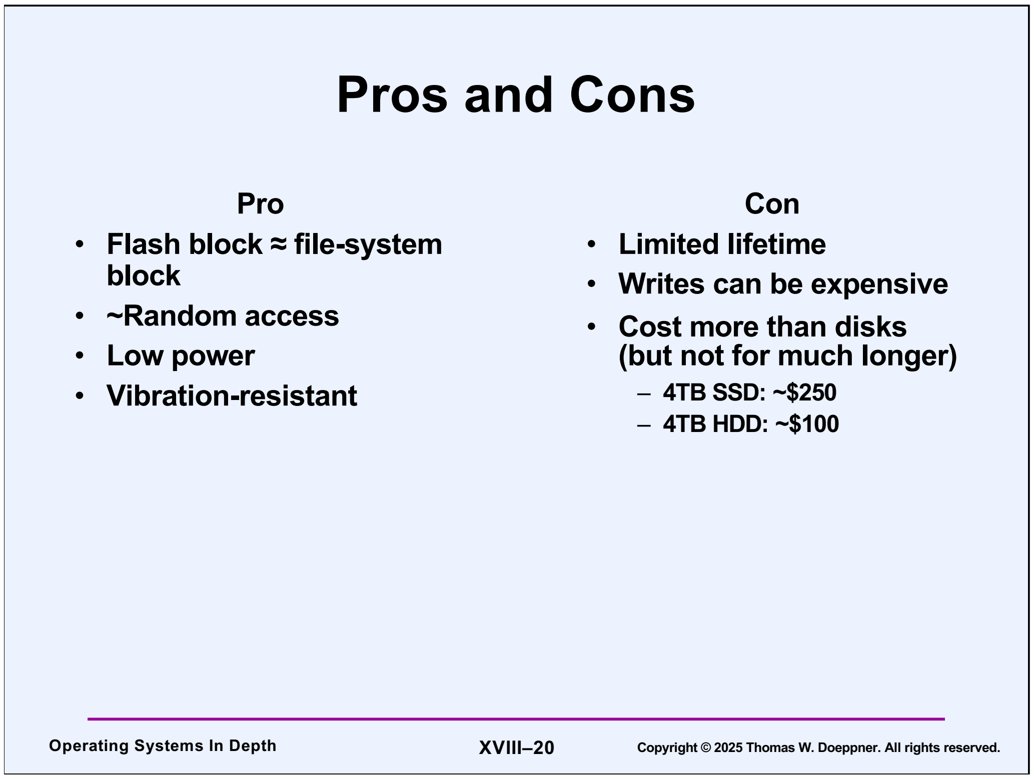


Solid-State Disks (SSDs)



- **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**

This slide was produced by the authors of the textbook “Computer Systems: A Programmer’s Perspective,” 2nd Edition and was provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O’Hallaron in Fall 2010.



Pros and Cons

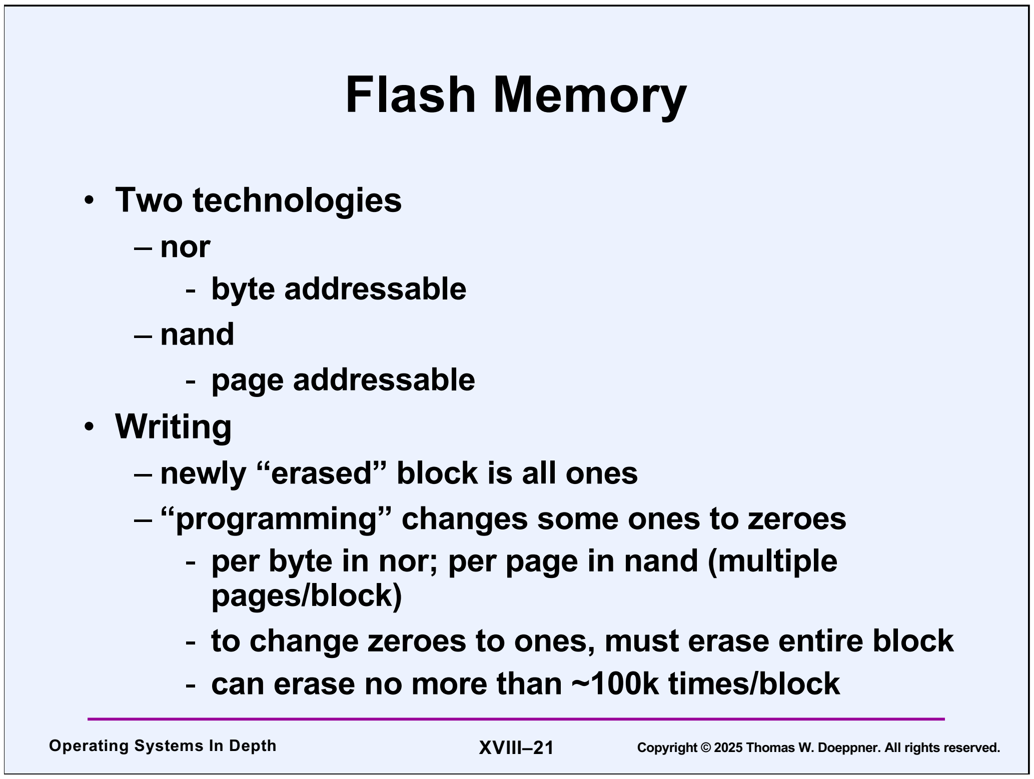
Pro

- 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

Prices are from Amazon.com; current as of 3/4/2025. 2-day free shipping included for Amazon Prime members.



Flash Memory

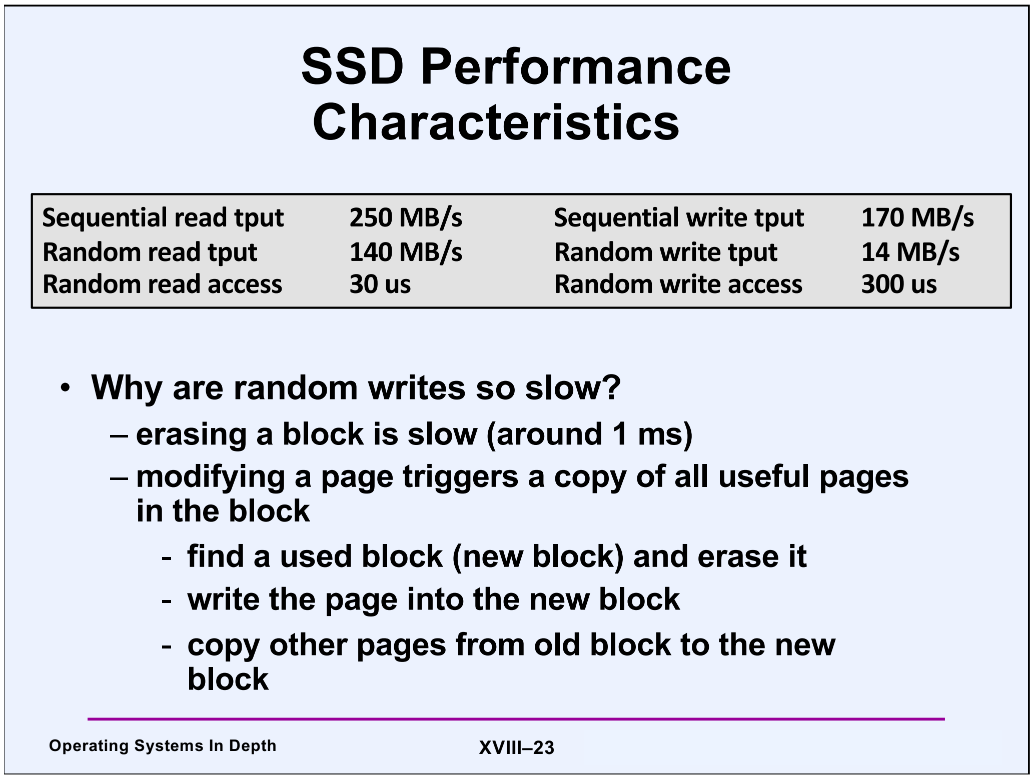
- Two technologies
  - nor
    - byte addressable
  - nand
    - page addressable
- Writing
  - newly "erased" block is all ones
  - "programming" changes some ones to zeroes
    - per byte in nor; per page in nand (multiple pages/block)
    - to change zeroes to ones, must erase entire block
    - can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

It's the nand technology that's used for SSDs.

Pages sizes are typically 512, 2k, or 4k bytes in length. Block sizes range from 16k bytes to 512k bytes.

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

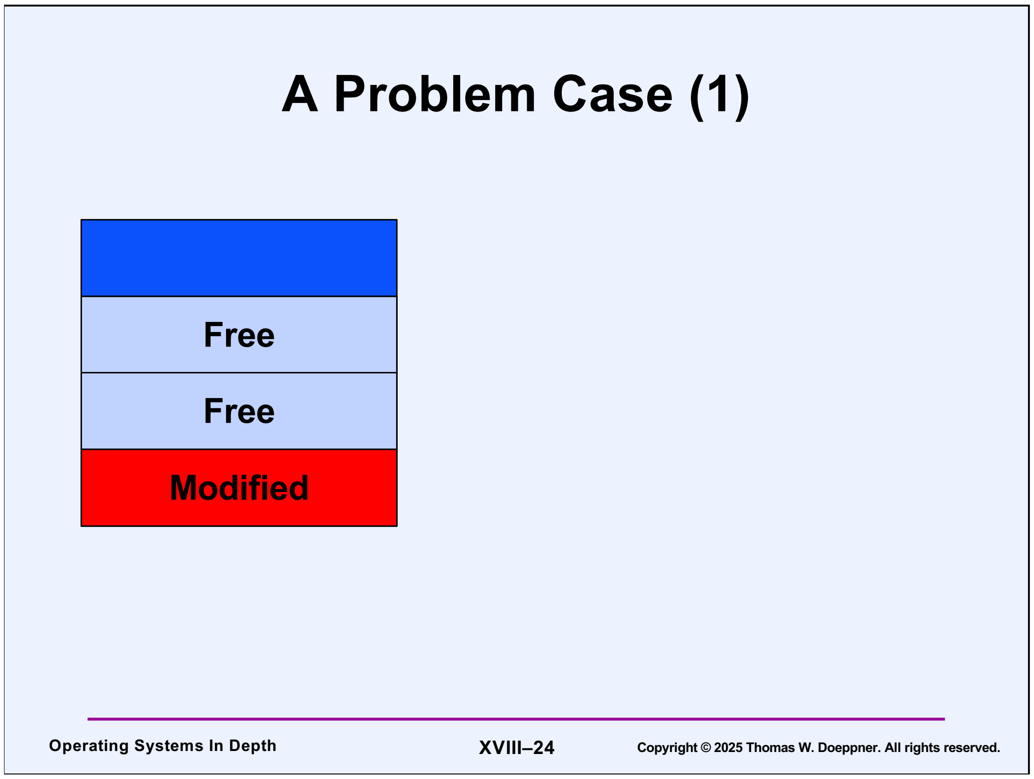


SSD Performance Characteristics

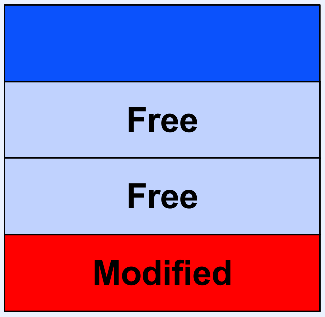
Sequential read tput	250 MB/s	Sequential write tput	170 MB/s
Random read tput	140 MB/s	Random write tput	14 MB/s
Random read access	30 us	Random write access	300 us

- 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

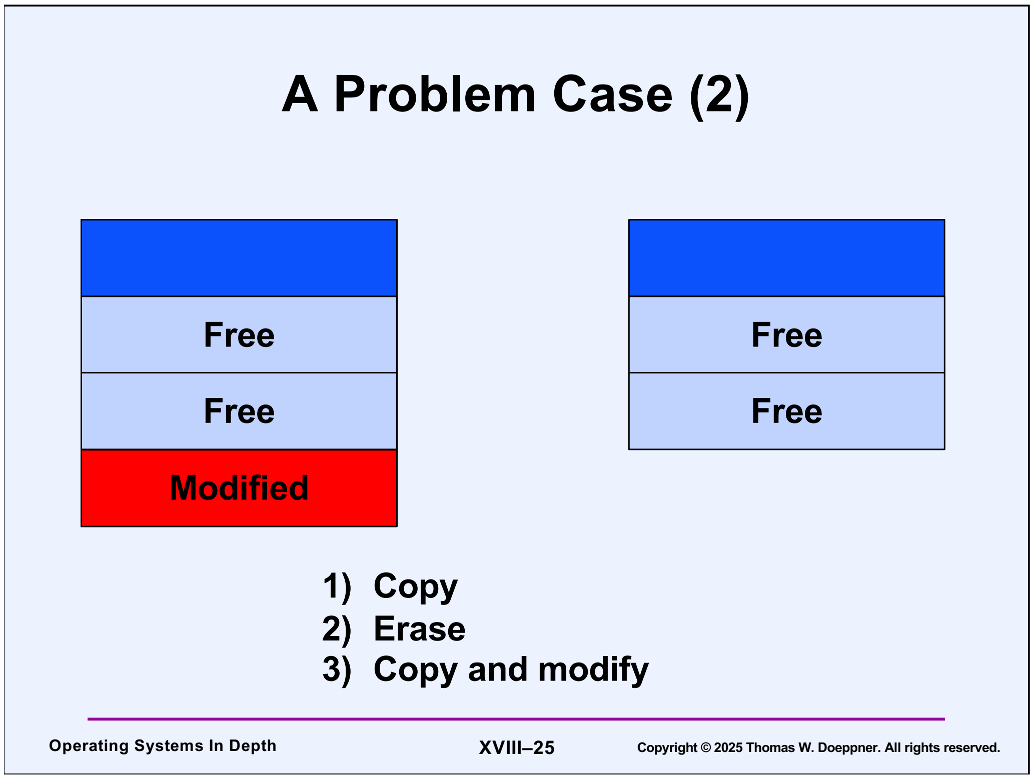
This slide was produced by the authors of the textbook "Computer Systems: A Programmer's Perspective," 2nd Edition and was provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O'Hallaron in Fall 2010.



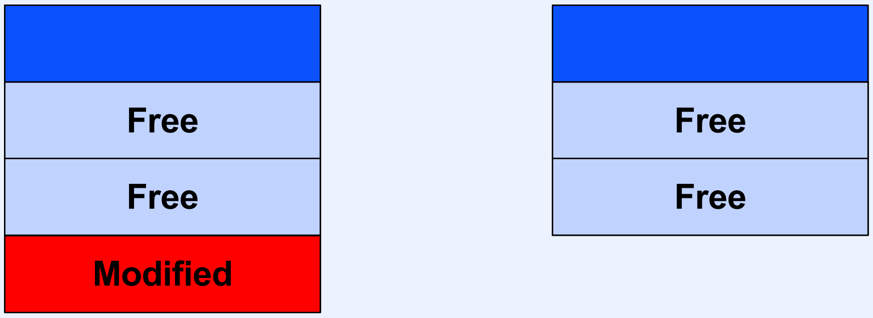
A Problem Case (1)



Here we have an SSD block whose pages are initially allocated. Two pages are then freed, then a page is modified.



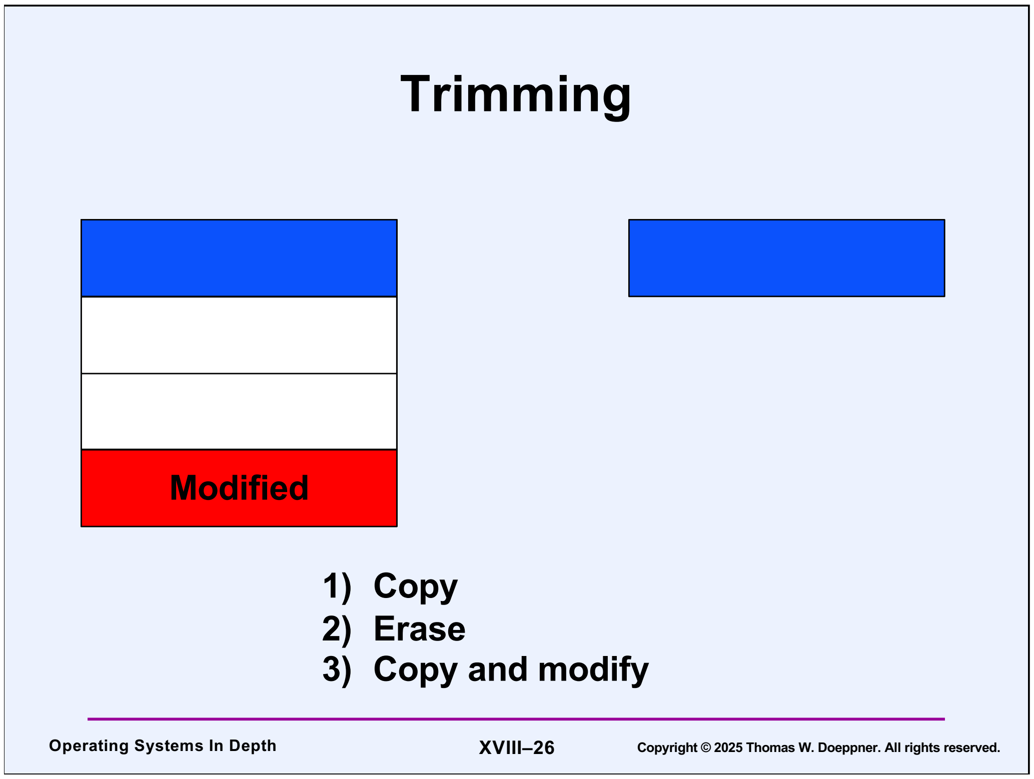
A Problem Case (2)



1) Copy
2) Erase
3) Copy and modify

What must happen, as implemented by the SSD firmware, is the following. Since the block must first be erased before any page can be modified, a copy of the entire block must be made. Then the block is erased. Then the original contents along with the modifications to the page are copied back.

Note that the fact that two of the pages are free, and thus their contents are not important and don't need to be copied, is not known to the SSD – the notion of free pages is known only at the file-system level.



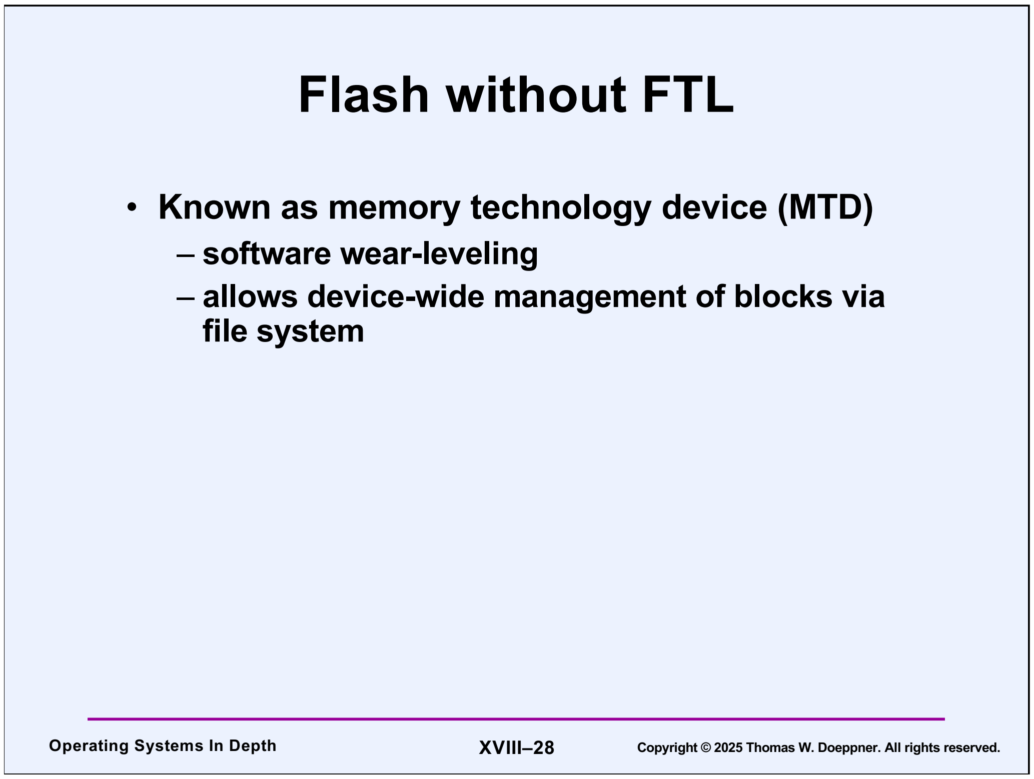
Trimming

1) Copy
2) Erase
3) Copy and modify

To get around the problem of the previous slide, SSD provides a "trim" command by which the file system can tell it that certain pages are free and thus their contents are not important. Thus the two freed pages don't need to be twice copied as in the previous slide.

Flash with FTL

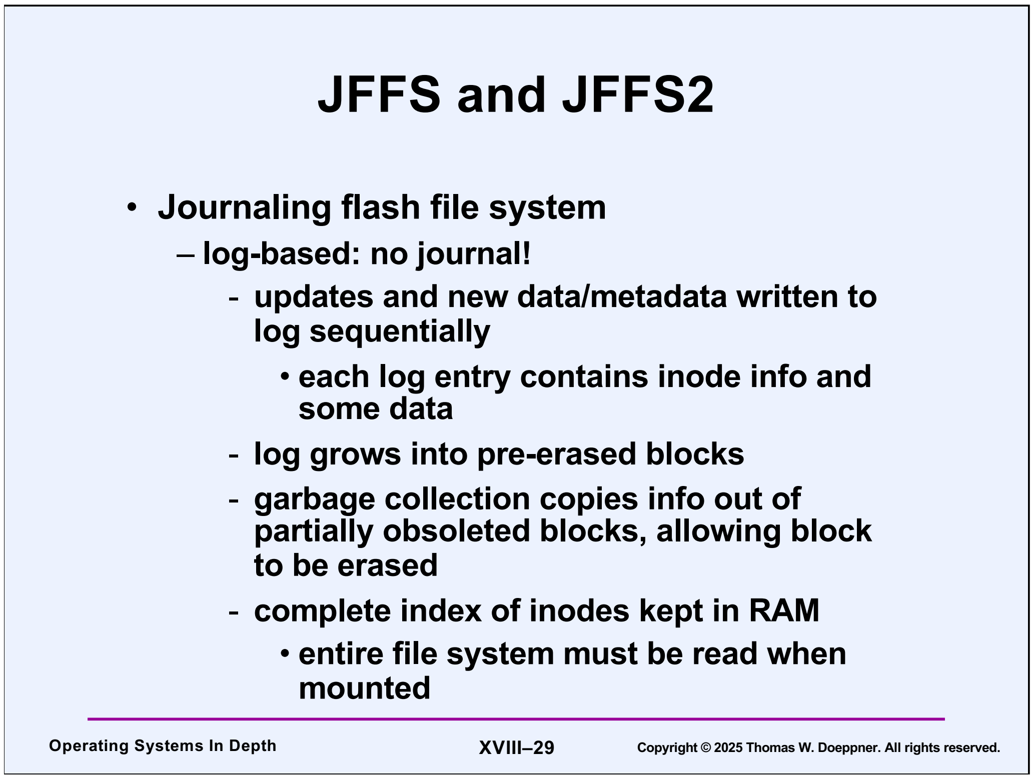
- Which file system?
  - FAT32 (sort of like S5FS, but from Microsoft)
  - NTFS
  - FFS
  - Ext3
  - Ext4
- All were designed to exploit disks
  - much of what they do is irrelevant for flash



Flash without FTL

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

With FTL, modifying a block involves finding a free block, erasing it, then copying unmodified pages to the new block, then writing the modified page. Perhaps we can do better by doing away with FTL and have the file system manage all blocks.



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

See http://sourceware.org/jffs2/jffs2-html/jffs2-html.html for descriptions of JFFS and JFFS2.

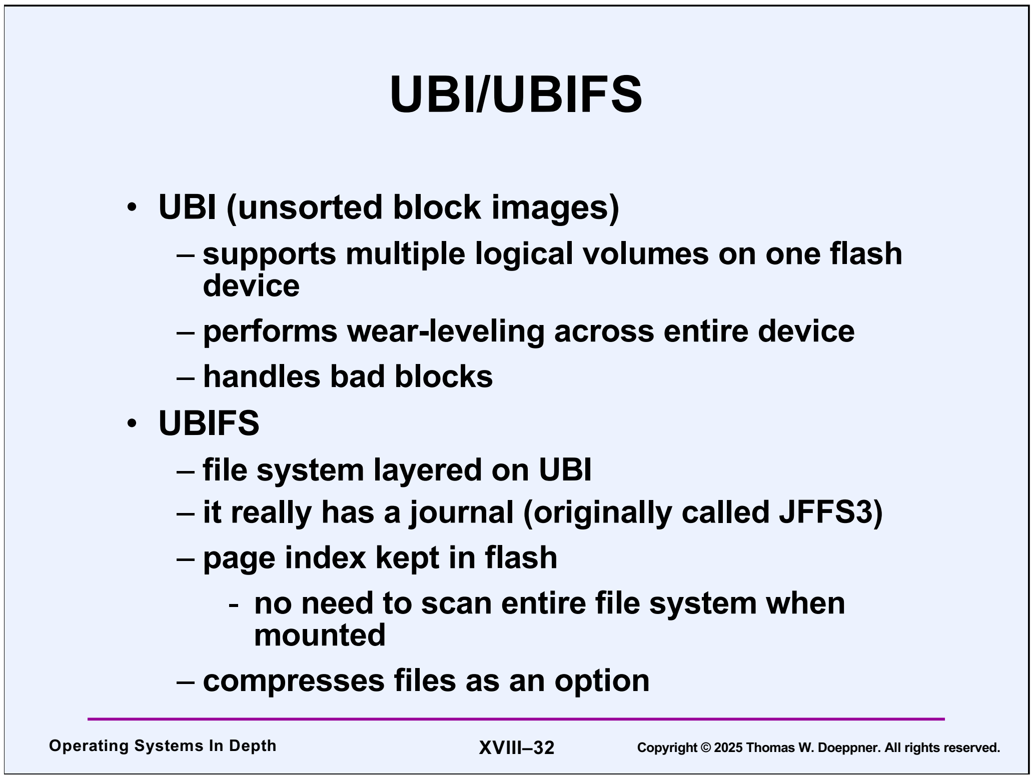
To modify a page, rather than finding a new block, then erasing it and updating it, there's a supply of erased blocks. Updates involve copying the old block to the new one, then modifying the desired page. These updates are done sequentially within the pre-erased blocks, forming a log.

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

Some information about how UBIFS works can be found at http://www.linux-mtd.infradead.org/doc/ubifs_whitepaper.pdf. (UBI stands for Unsorted Block Images.)

Updates to the page index are done in a transactional fashion: they are stored in a journal, then updated in flash after being committed. The goal is not so much to have atomic updates, but to reduce the frequency of writes to flash. Thus a number of updates can be done via a single write.

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

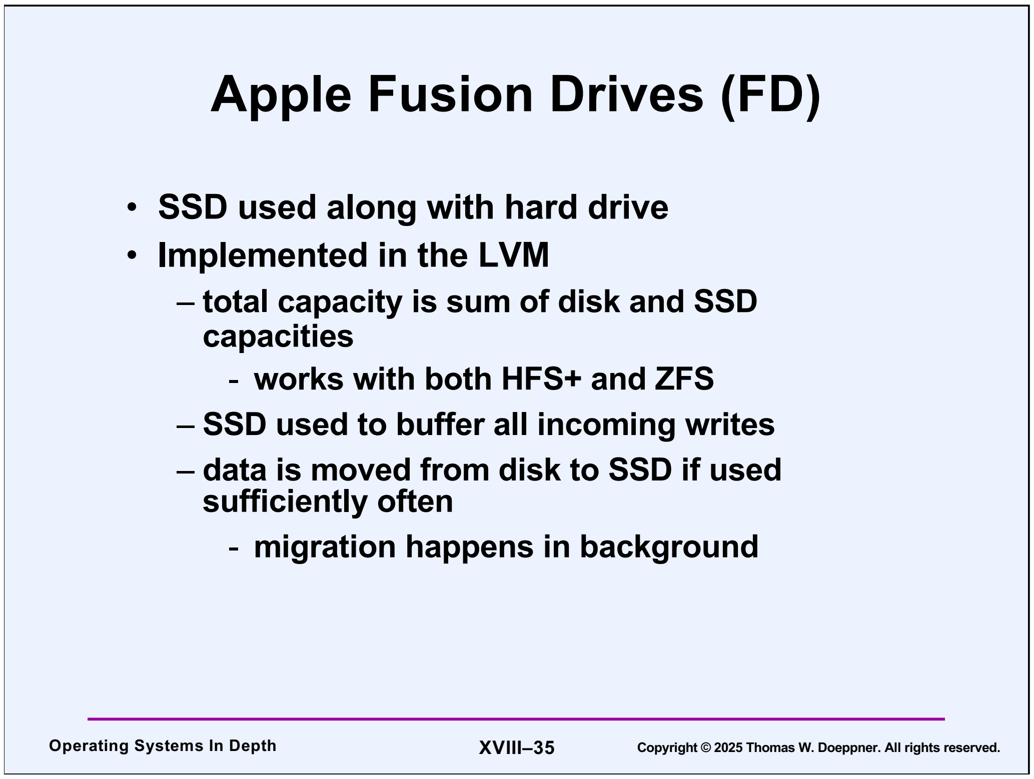
Quiz 4

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

The information in this slide is not confirmed by Apple (who considers the technology proprietary) but comes from https://en.wikipedia.org/wiki/Fusion_Drive.

Apple appears to be no longer using this technology, supplying only SSD storage on recent computers.

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

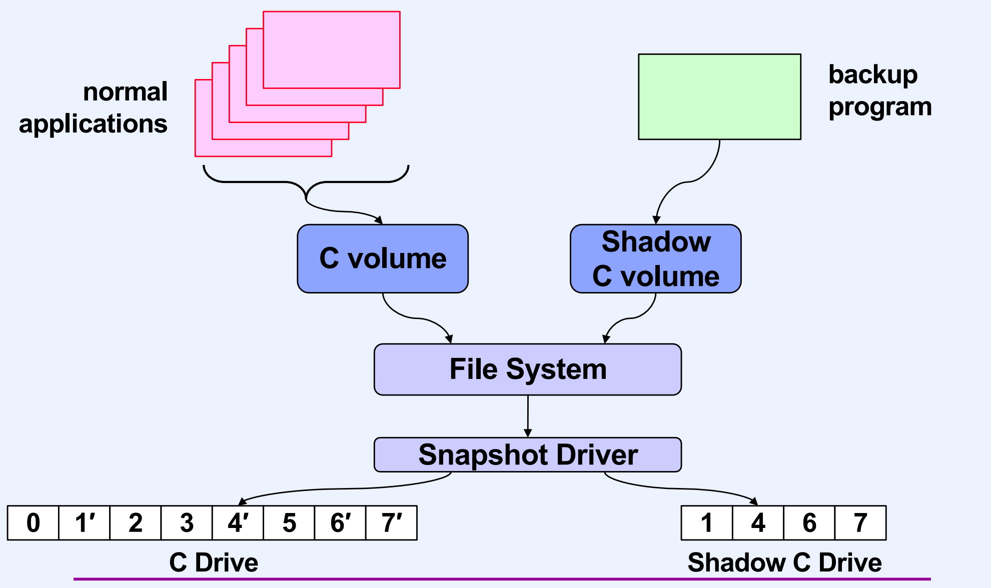
NTFS

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

Backups

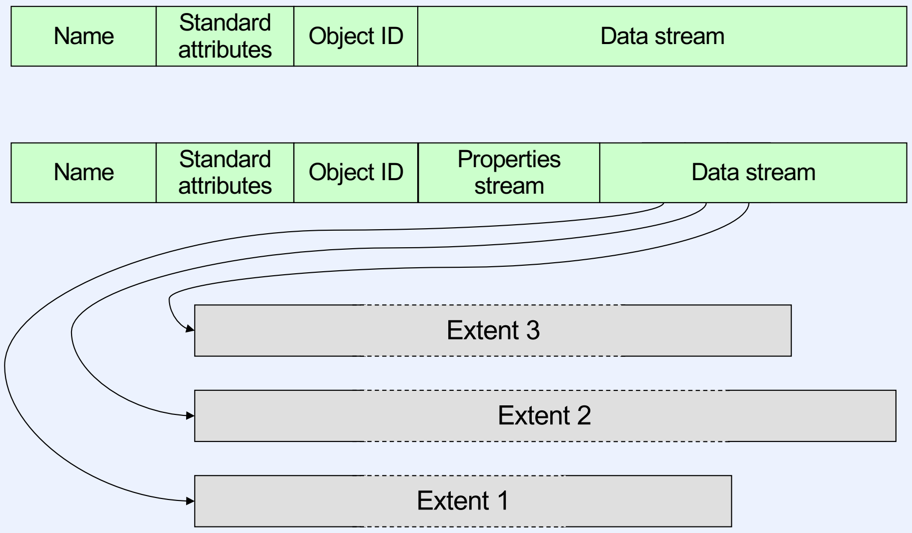
- Want to back up a file system
  - while still using it
    - files are being modified while the backup takes place
    - applications may be in progress — files in inconsistent states
- Solution
  - have critical applications quickly reach a safe point and pause
  - snapshot the file system
  - resume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

Windows Snapshots



Windows does snapshotting at the “volume” level rather than at the file-system level. For example, normal applications might be accessing and modifying files on the C drive. The backup program asks for a snapshot, which creates a new logical volume called the shadow C drive. This actually causes a special “snapshot driver” to be inserted between the file system code and the (logical) disk drives. Initially, the snapshot driver sends requests for both the C drive and the shadow C drive to the C drive. However, when a normal application attempts to modify a block, first the snapshot driver copies the block to the shadow C drive, then modifies the block on the C drive. Future attempts by the backup program to read the block are directed to the shadow C drive, so that it gets the original version.

NTFS File Records



Two NTFS file records. The top one is for a small file fitting completely within the record. The bottom one is for a file containing two streams: one for some application-specific properties and one for normal data. The latter is too big to fit in the file record and is held in three extents.

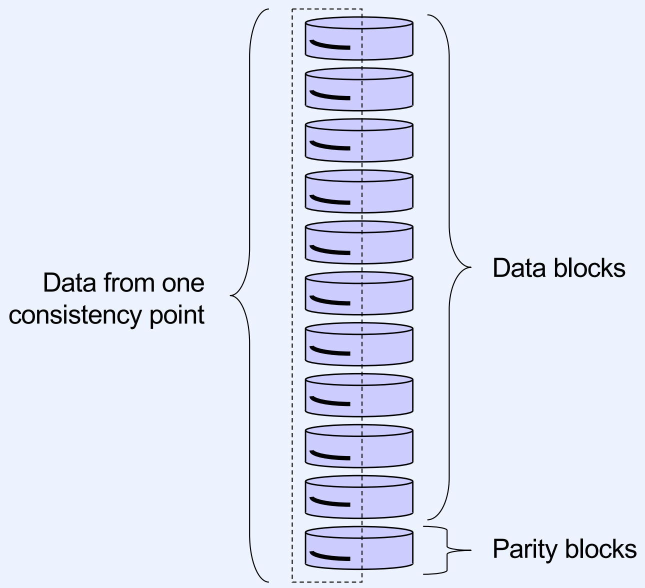
Additional NTFS Features

- 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



WAFL uses RAID level 4 and writes out changes in batches called consistency points. Each batch occupies some integral number of stripes — thus the parity disk is written to no more often than the data disks. Though blocks in a consistency point are from any number of files, they're written (using log-structured techniques) to the next contiguous locations on each disk.