

File Systems (Part 4)

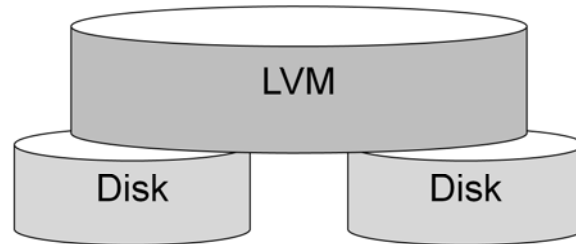
Outline

- Utilizing multiple disks
 - LVM
 - RAID
- Utilizing no disks
 - Flash

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**
- **Data can be spread across multiple drives, allowing parallel access**

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

	Disk 1	Disk 2	Disk 3	Disk 4
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

Disk striping: each stripe is written across all disks at once. The size of a “unit” may be anywhere from a bit to multiple tracks. If it’s less than a sector in size, then multiple stripes are transferred at once so that the amount of data per transfer per disk is an integer multiple of a sector.

Concurrency Factor

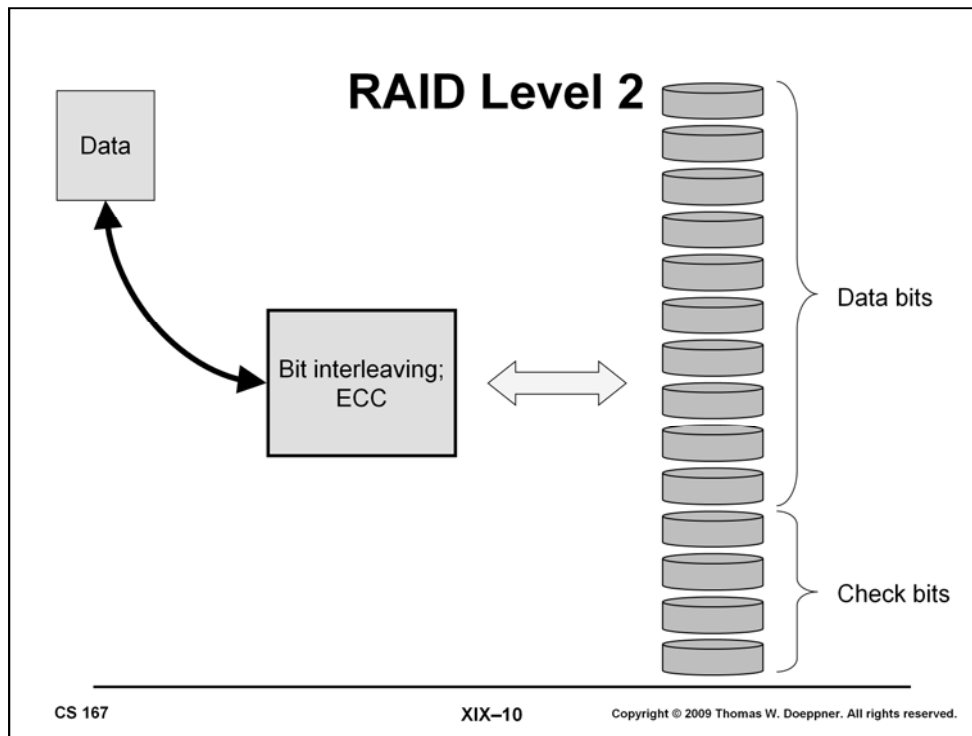
- **How many requests are available to be executed at once?**
 - **one request in queue at a time**
 - concurrency factor = 1
 - e.g., one single-threaded application placing one request at a time
 - **many requests in queue**
 - concurrency factor > 1
 - e.g., multiple threads placing file-system requests

Striping: The Effective Disk

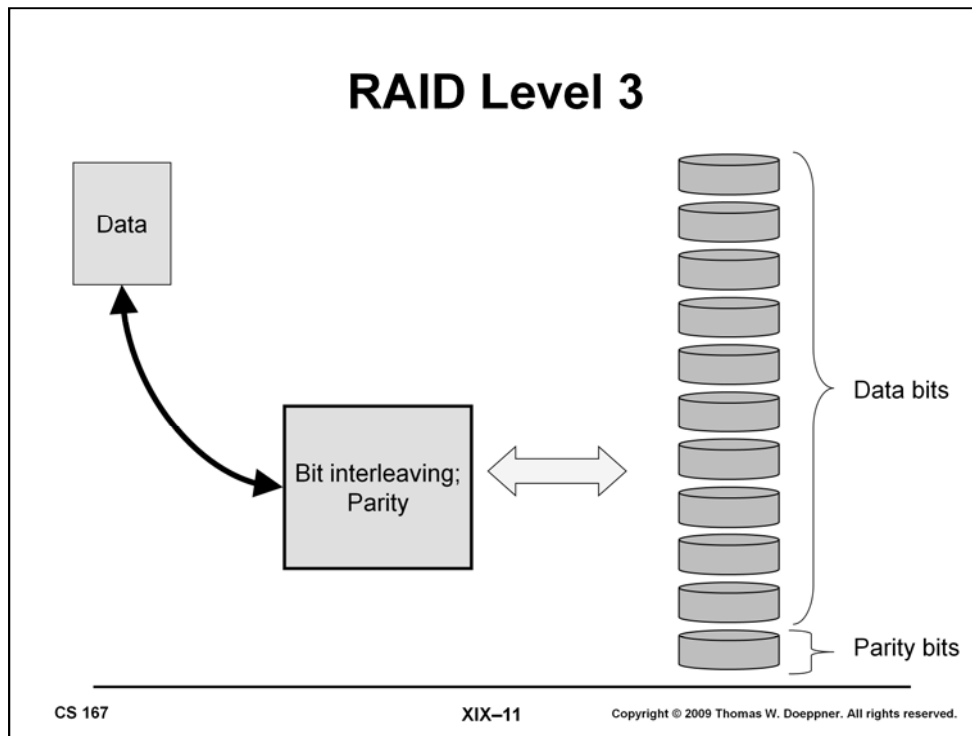
- Improved effective transfer speed
 - parallelism
- No improvement in seek and rotational delays
 - sometimes worse
- A system depending on N disks is much more likely to fail than one depending on one disk
 - if probability of one disk's failing is f
 - probability of N -disk system's failing is $(1-(1-f)^N)$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

RAID to the Rescue

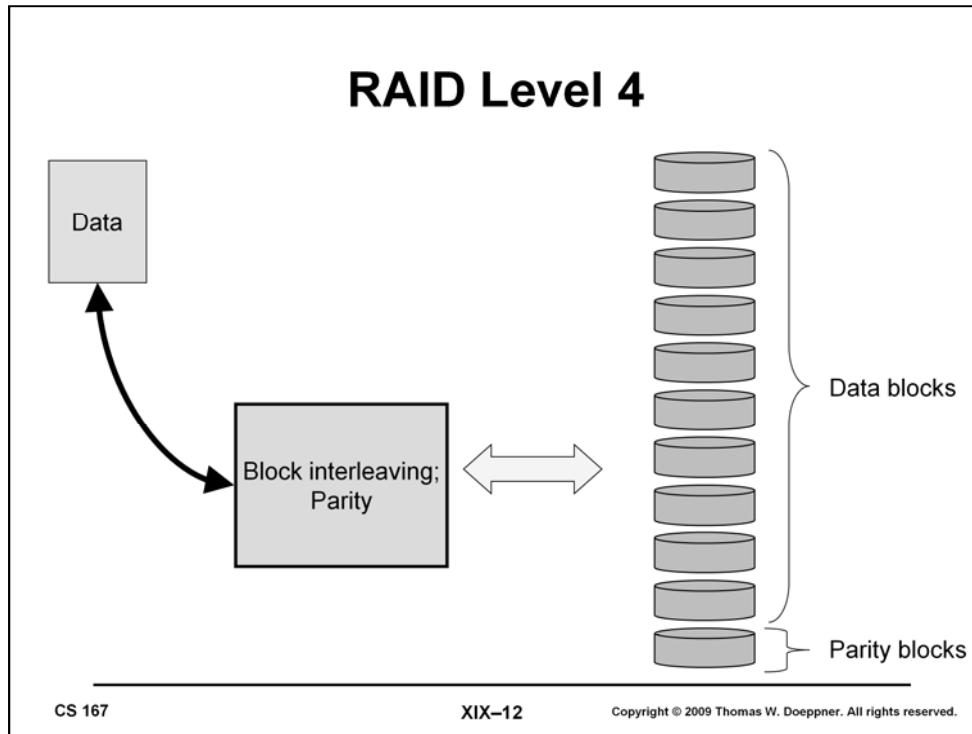
- **Redundant Array of Inexpensive Disks**
 - (as opposed to Single Large Expensive Disk: SLED)
 - combine striping with mirroring
 - 5 different variations originally defined
 - RAID level 1 through RAID level 5
 - RAID level 0: pure striping
 - numbering extended later
 - RAID level 1: pure mirroring



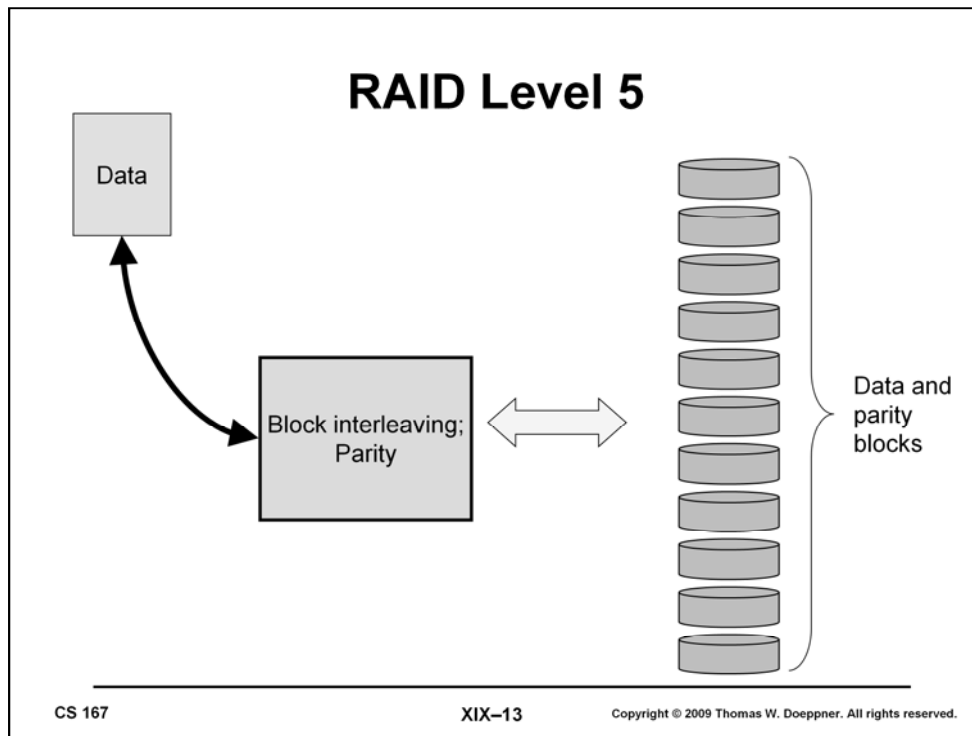
RAID level 2: bit interleaving with an error-correcting code.



RAID level 3: bit interleaving with parity bits.



RAID level 4: block interleaving with parity blocks.



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 11 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
- **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

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

Beyond Disks: Flash

Pro

- Flash block \approx file-system block
- Random access
- Low power
- Vibration-resistant

Con

- Limited lifetime
- Write is expensive
- Cost more than disks
 - 128GB SSD: ~\$400
 - x5 more last year
 - 1TB disk: ~\$100
 - x10 more last year

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

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 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.

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

See “Flash Storage Memory” by Adam Leventhal (Brown '01 and former 167/169 HTA) in Communications of the ACM, July 2008, vol. 51, no. 7.