Data Integrity

- disks can fail
- failure modes
  - whole disk failure
    - either the disk works or it doesn't
    - addressed by RAID
  - single-block failure
    - latent sector errors (LSEs)
    - block corruption
  - write problems
    - misdirected writes
    - lost writes

	Cheap	Costly
LSEs	9.40%	1.40%
Corruption	0.50%	0.05%

% drives exhibiting at least one error, from a study of 1.5 million disk drives over 3 years

RAID

- *redundant array of inexpensive disks* (RAID)
- combines multiple disks into a single logical unit to improve performance, capacity, and/or reliability
- *fail-stop* fault model
  - either the disk works or it has failed
  - assumes a failed disk is easy to detect
- key RAID levels
  - RAID 0 – *striping* (only)
    - data is split across disks – improves performance, no redundancy
  - RAID 1 – *mirroring*
    - duplicate data on another disk – protects against single disk failure
  - RAID 4/5 – parity-based
    - stores *parity* (XOR of data blocks) to reconstruct lost data
    - protects against (single) disk failure, not silent corruption – if data is corrupted before parity is computed, the bad data is preserved

Single Block Failures

- *latent sector error* (LSE)
  - a bad or unreadable sector not detected until a read is attempted
    - *error correcting codes* (ECC) allow for detection (and sometimes correction)
  - typically shows up as a read failure
- causes
  - e.g. from a *head crash* (disk head touches a platter surface)
  - e.g. from a *single event upset* (cosmic rays flipping bits)
    - more common in airplanes and space, but not unheard of in everyday systems
      - 4096 extra votes in an election
        - » https://www.independent.co.uk/news/science/subatomic-particles-cosmic-rays-computers-change-elections-planes-autopilot-a7584616.html
      - Super Mario speedrunner – a rare case of a benefit
        - » https://www.thegamer.com/how-ionizing-particle-outer-space-helped-super-mario-64-speedrunner-save-time/

Handling Latent Sector Errors

- when an error is detected, use the disk's redundancy mechanism to recover
  - e.g. RAID 1 (mirrored), RAID 4/5 (parity based)
  - e.g. RAID-DP, WAFL – extra redundancy to allow recovery with both a full-disk fault and LSEs

Single Block Failures

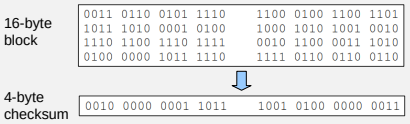
- *block corruption*
  - block is readable but contains bad data
  - may or may not be detected (*silent corruption*)
- causes
  - e.g. buggy disk firmware results in a bad write
  - e.g. a faulty bus when data is transferred to disk

Detecting Block Corruption

- utilize a *checksum*
  - checksum = small summary of a chunk of data
  - compute and store with original write
  - re-compute and compare with stored value on read
- checksum functions
  - there's typically a tradeoff between speed and strength
  - important to minimize the chance of *collisions*
    - collision = two different blocks having the same checksum

Simple Checksums

- XOR-based (parity bits)
  - bitwise XOR chunks of the blocks



- strength
  - can detect one (odd number) flipped bit per column
  - can't detect two (even number) flipped bits in the same column

Simple Checksums

- twos complement sum
  - add the bytes of the block, discarding the MSB carry bit
  - invert final result
    - ensures non-zero checksum for all-0 data

```
  10110101
  11000000
  00001101
+ 11100011
----------
  01100101
```

  - strength
    - not sensitive to the order of bytes – differently-ordered content results in collisions
    - can detect errors as long as the lowest order column with an error has only one (odd number) flipped bit
- ones complement sum
  - add the bytes of the block, wrapping the MSB carry bit to the LSB
  - strength
    - similar to twos complement, but can also detect even-bit errors in MSB

Better Checksums

- Fletcher checksum
  - block *D* consists of bytes *d1 d2 … dn*
  - involves two check bytes
    - check byte *s1* is a sum-of-bytes checksum *s1* = (*s1*+*di*) % 255
    - check byte *s2* is the sum of successive *s1* values *s2* = (*s2*+*s1*) % 255
  - strength
    - incorporates order sensitivity
    - detects all single-bit and double-bit errors
    - detects many burst errors (sequences of flipped bits)

Better Checksums

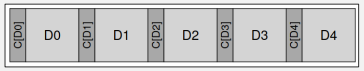
- cyclic redundancy check (CRC)
  - compute *D* % *k*

```
11010011101100 000 <--- input padded by 3 bits from the right
1011               <--- divisor
01100011101100 000 <--- result (the first four bits are the XOR with the divisor beneath, the rest of the
bits are unchanged)
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000 <--- the divisor moves over to align with the next 1 in the dividend (since quotient for
that step was zero)
       1011             (in other words, it doesn't necessarily move one bit per iteration)
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <--- remainder (3 bits).  Division algorithm stops here as dividend is equal to zero.
```

 https://en.wikipedia.org/wiki/Cyclic_redundancy_check

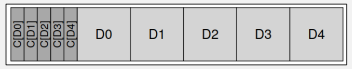
Checksum Layout

- store a checksum with each data block



  - wrinkle: checksums are small (8 bytes) and disks can only handle writing whole sections (512 bytes)
  - solution: format drive with 520-byte sectors (requires drive support)

- group checksums together in block-sized chunks



  - works on all disks
  - less efficient
    - writing a data block adds read and write of the checksum block

 https://pages.cs.wisc.edu/~remzi/OSTEP/file-integrity.pdf

Using Checksums

- when a block is read, also read its *stored checksum*
- compute a checksum for the read block
- compare the stored and computed checksums
- can be done by the file system or storage controller
- checksums only support error *detection*
- error can only be fixed if there is a redundant copy available