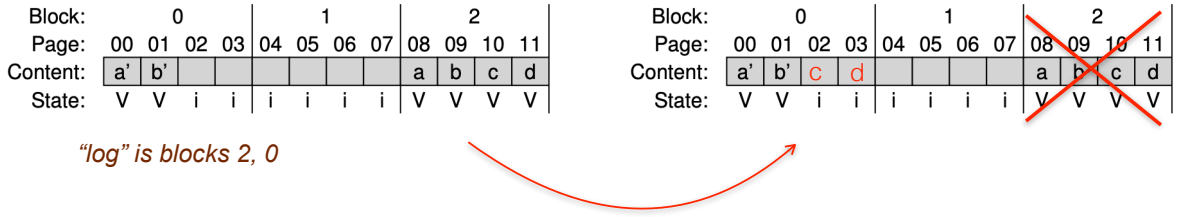
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency and Journaling
- 43 - Log-structured (and other) File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

SSDs *hybrid mapping*



- This is a *partial merge:*
  - could clean up by copying c, d to end of log (block 0)
  - i.e. the rest of a single block
- Might need to copy from many blocks i.e, a *full merge:*
  - assume blocks 0, 4, 8, 12 written
  - would need to write *0,1,2,3* and *4,5,6,7* and….
  - expensive
- Another approach is to only cache part of mapping in memory
  - the rest are stored on flash

SSDs *conclusion*

- Other issues
  - FTL can be expensive
  - wear leveling:
    - erase/program cycles quickly wear out blocks
    - FTL tries to distribute evenly
    - long-lived data does not get write share:
      - periodically read live data and write elsewhere (??)
  - cost
- But:

Device	Random Reads (MB/s)	Random Writes (MB/s)	Sequential Reads (MB/s)	Sequential Writes (MB/s)
Samsung 840 Pro SSD	103	287	421	384
Seagate 600 SSD	84	252	424	374
Intel SSD 335 SSD	39	222	344	354
Seagate Savvio 15K.3 HDD	2	2	223	223

Data Integrity *how to ensure our data is safe?*

- RAID
  - good, but assumes *fail-stop* failures
  - also need to worry about:
    - latent-sector errors (LSEs)
    - block corruption

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

  - over 3 years, 1.5 million drives

Data Integrity *latent sector errors*

Latent sector errors:

- causes:
  - head crashes
  - cosmic rays
- hardware for the win….
  - in-disk error-correcting codes (ECC)
  - ECC fails lead to *disk returning an error* while reading
  - depending on the failure, and the type of ECC, disk might even be able to correct bit errors
- recover using RAID
  - but what if full-disk failure while attempting to recover a sector?
  - *use two parity blocks…*

Data Integrity *block corruption*

- disk might become corrupt in way not detectable by disk itself:
  - disk might have incorrect block
  - block corrupted on way to (or from) disk
- causes
  - buggy firmware might write block to wrong location
  - buggy hardware
- detection
  - file systems use checksums w/ various speeds and strengths:
    - XOR of all words
    - addition of all words
    - cyclic redundancy check (CRC)
  - but where to store checksums?

Data Integrity *misdirected blocks*

- Where to store checksums?
  - manufacturer can format drive w/ 520-byte sectors

  - consolidate checksums on another sector

- How do we use them?
  - compare checksums when reading, hope for a backup
- What if block b_x stored to sector *y* instead of *x*?
  - checksum would be valid
  - *include x in the checksum*

Distributed Systems

Distributed Systems

- 48 - Communication Basics
- 49 - NFS
- 50 - AFS
- GFS

Communication Basics

- Building distributed systems
  - all components fail
  - communication fails
  - how to build systems that *rarely* fail from components that do?
- Issues:
  - communication
    - what are the right primitives?
    - what are the right types of applications?
  - performance
    - especially with interconnects much slower than buses
  - security
    - systems span users, domains
    - the Internet is scary

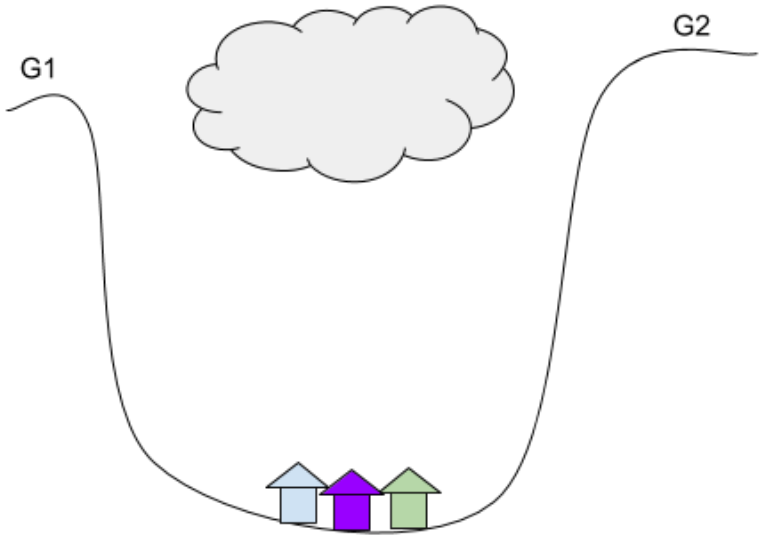
Communication

"progress and correctness of distributed consensus algorithms is impossible to prove in asynchronous environments" - FLP theorem

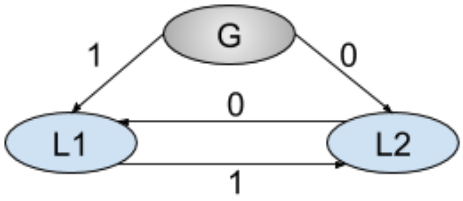
- communication is fundamentally unreliable
  - packet loss
  - packet corruption
  - packet delays
- maybe don't rely on reliability
  - maybe add encryption to the link!
  - but….

Two Generals *brief segue…*

"progress and correctness of distributed consensus algorithms is impossible to prove in asynchronous environments" - FLP theorem

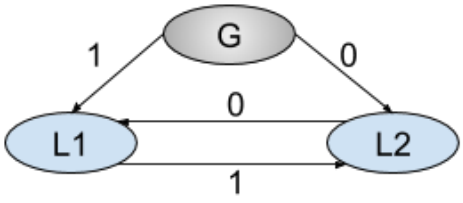


Impossibility of consensus w/ 3

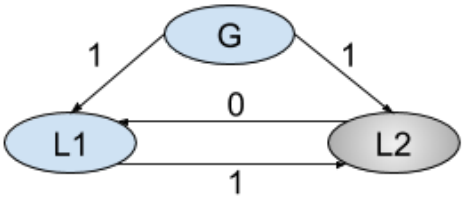


Impossibility of consensus w/ 3

Can L1 tell who is faulty?



Can L1 tell who is faulty?



- *all local lieutenants do the same thing*
- *any local lieutenant does what the general says*

more about consensus later, if we have time….

End-to-End Argument *crypto is always good, right?*

- *end-to-end argument* says:
  - application layers at each end of comm are the *only* appropriate places where some functionality should be implemented
- examples:
  - provided encryption might not be good enough
    - 

    - 3DES is ancient, maybe want to use *AES*, *blowfish*
  - provided encryption might be too expensive
    - might not need encryption at all, just adds overhead
  - app semantics might be needed
    - different app messages might have different needs
- but strong semantics in underlying layers *do* help