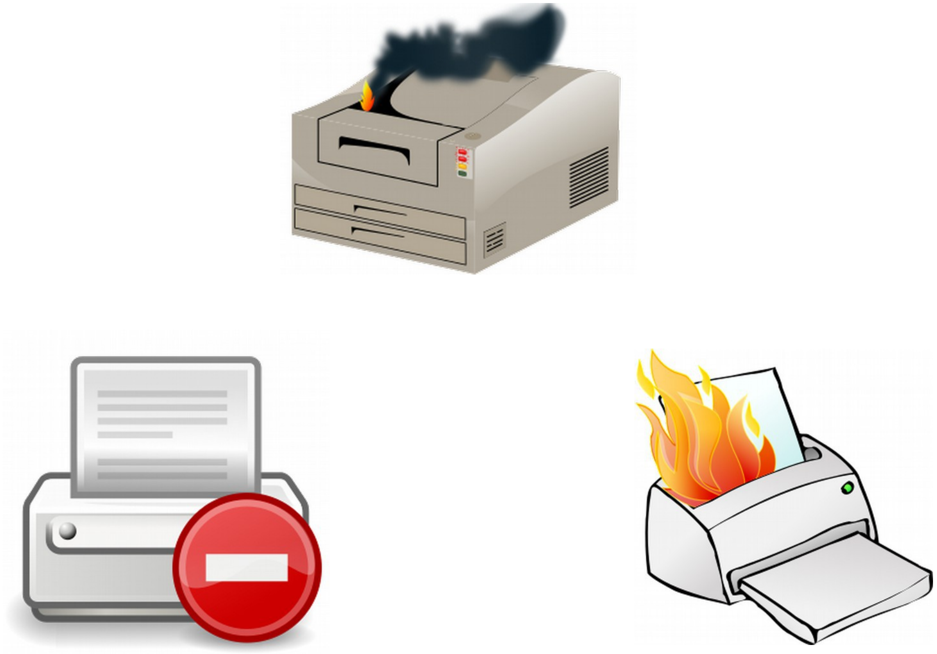
CS 470
Spring 2017

Mike Lam, Professor



# Fault Tolerance

Content taken from the following:

*"Distributed Systems: Principles and Paradigms"* by Andrew S. Tanenbaum and Maarten Van Steen (Chapter 8)
Various online sources, including `github.com/donnemartin/system-design-primer`

# Desirable system properties

- We want dependable systems
  - Available: ready for use at any given time
  - Reliable: runs continuously without failure
  - Safe: nothing catastrophic happens upon failure
  - Maintainable: easy to repair
  - Similar to definitions for dependable software (CS 345)

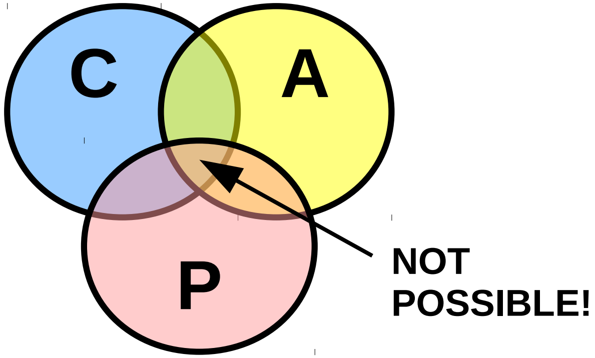
# Problem

- Inherent tension between:
  - **C**onsistency: reads see previous writes ("safety")
  - **A**vailability: operations finish ("liveness")
  - **P**artition tolerance: failures don't affect correctness

*Can we “have it all?”*

# CAP Theorem

- A system cannot be simultaneously consistent (C), available (A), and partition-tolerant (P)
  - We can only have two of three
  - In a non-distributed system, P isn't needed
    - Tradeoff: latency vs. consistency ("PACELC Theorem")
  - In a distributed system, P isn't optional
    - Thus, we must choose: CP or AP
    - I.e., consistency or availability



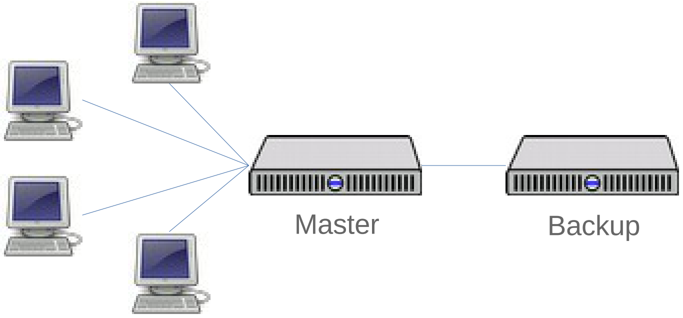
Original conjecture by Eric Brewer: `http://dl.acm.org/citation.cfm?id=822436`
Formal theorem: `http://dl.acm.org/citation.cfm?id=564601`

# Consistency

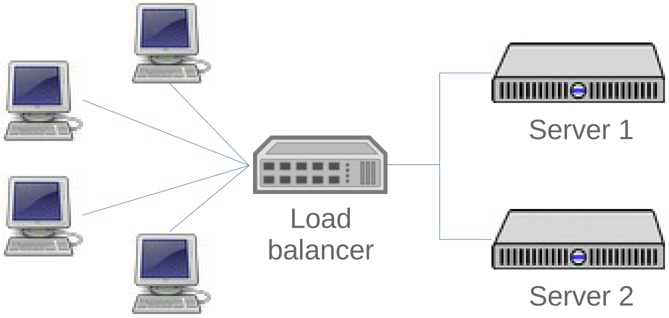
- Usual choice: compromise on consistency
  - Strong consistency: reads see all previous writes (sequential consistency)
    - Alternatively, continuous w/ short interval
    - Causal consistency: reads see all causally-related writes
  - Eventual consistency: reads eventually see all previous writes (continuous w/ long interval)
    - E.g., "guaranteed convergence"
  - Weak consistency: reads may not see previous writes
    - E.g., "best effort"

# Availability

- Active-passive / master-slave (asymmetric)
  - Master server handles all requests
  - Backup/failover server takes over if master fails
- Active-active / master-master (symmetric)
  - Multiple master servers share request load
  - Load re-balances if one fails



**Active-passive**



**Active-active**

# Fault tolerance

- Sometimes, consistency/availability tradeoff decisions depend on the failure model:
  - What kinds of failures happen?
  - How often do they happen?
  - What are the effects of a failure?

# Fault tolerance

- Soft vs hard failures
  - Soft failure: data is corrupted (often corrected by hardware)
  - Hard failure: a component of a system stops working
- Hard failures in a non-distributed system are usually fatal
  - The entire system must be restarted
- Hard failures in a distributed system can be non-fatal
  - Partial failure: a failure of a subset of the components of a distributed system
  - If the system is well-designed, it may be able to recover and continue after a partial failure

# Measuring failure

- Failure rate (λ): failures per unit of time
- Mean Time Between Failures (MTBF) = 1 / λ
  - Assumes constant failure rate
- Failures In Time (FIT) = failures expected in one billion device-hours
  - MTBF = 1e9 x 1/FIT

# Measuring failure

- Failure rate (λ): failures per unit of time
- Mean Time Between Failures (MTBF) = 1 / λ
  - Assumes constant failure rate
- Failures In Time (FIT) = failures expected in one billion device-hours
  - MTBF = 1e9 x 1/FIT

On a 10 million core machine, 1 FIT means once every 100 hours
or **once every ~4.2 days**!

# Failure types

- Crash: the system halts
- Omission: the system fails to respond to requests
- Timing: the system responds too slowly
- Response: the system responds incorrectly
- Arbitrary failure: anything else (unpredictable!)
  - Sometimes called "Byzantine" failures if they can manifest in such a way that prevents future consensus

# Failures

- Some distinguish between failure levels:
  - A failure occurs when a system cannot meet its specification
  - An error is the part of a system's state that leads to a failure
  - A fault is the low-level cause of an error
  - Most common source of faults: memory or disk storage
- If a system can provide dependable services even in the presence of faults, that system is fault-tolerant

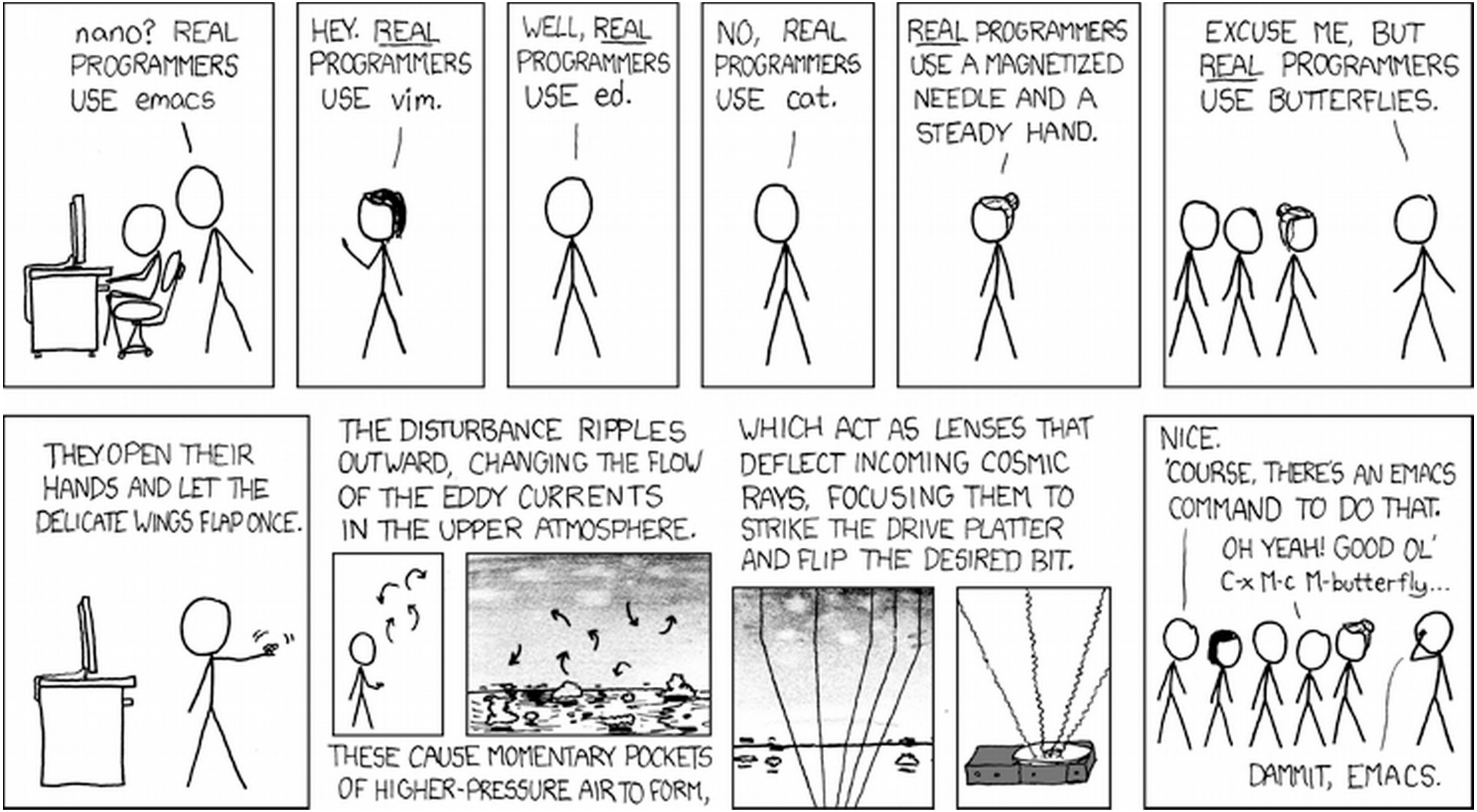
# Faults

- Permanent faults reproduce deterministically
  - These are usually the easiest to fix
- Intermittent faults recur but do not always reproduce deterministically
  - Unfortunately common in distributed systems
  - Heisenbug: a software defect that seems to change or disappear during debugging
- Transient faults occur only once
  - Often the result of physical phenomena

# Bit errors

- Bit error: low-level fault where a bit is read/written incorrectly
- Single-bit vs. double-bit vs. multi-bit
  - Single-Bit Error (SBE), Double-Bit Error (DBE)
  - Hamming distance: # of bits different
- Potential DRAM source: "weak bits" in hardware
  - Electrons are stored in a memory cell capacitor
  - Critical charge ($Q_{crit}$) is the threshold between 0 and 1 values
  - Refreshed often, but sometimes still read incorrectly
- Radiation and cosmic rays

# Cosmic rays


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

# Example: GPU fault study

The Titan supercomputer
has 18,688 GPUs

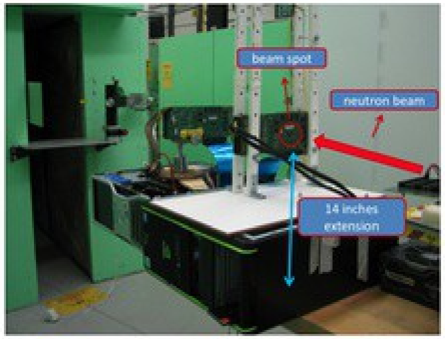
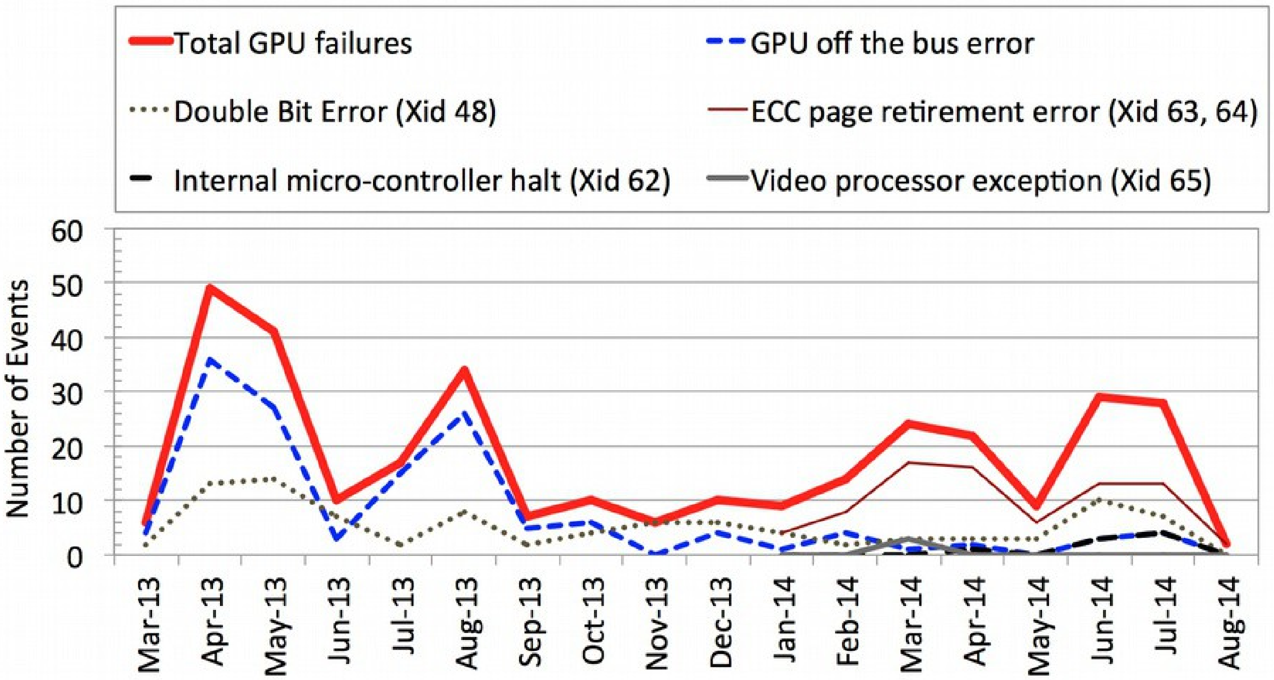
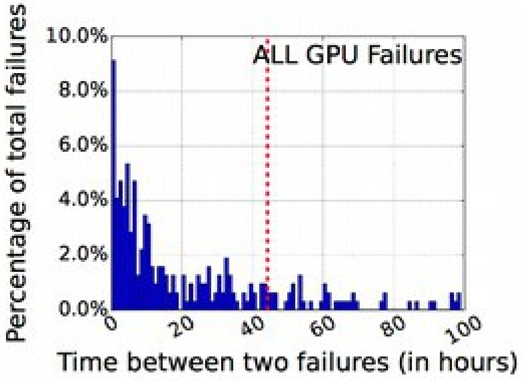
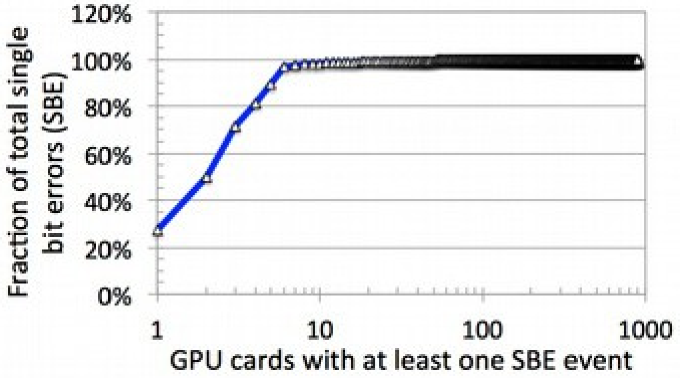


**Figure 3: Radiation test setup inside the ICE House II, Los Alamos Neutron Science Center (LANSC), LANL.** A similar setup was used at ISIS, Didcot, UK.







Tiwari, Devesh, et al. "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation."
High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. IEEE, 2015.
`http://www4.ncsu.edu/~dtiwari2/Papers/2015_HPCA_Tiwari_GPU_Reliability.pdf`

# Dealing with failure

- Detection: discovering failures
  - Active (pinging) vs. passive (wait for messages)
  - Issue: unreliability of timeouts
- Prevention: eliminate the possibility of failure
  - Not possible in a distributed system
- Avoidance: divert around failure possibilities
  - Only possible in particular circumstances
- Recovery: restore valid system state after a failure

# Detection and avoidance

- Data-centric
  - Redundancy, diversity, and replication
    - E.g., dual modular redundancy (DMR), TMR
  - Parity bits, checksums, and hashes
    - E.g., cyclic redundancy check (CRC), MD5, SHA
- Computation-centric
  - Acknowledgement (ACK)-based protocols
  - Consensus and voting protocols
    - One-phase vs. two-phase (e.g., Paxos)

# Recovery (hardware)

- Hardware (general space vs. safety tradeoff)
  - Dual modular redundancy (DMR) can **detect** a single-bit error
  - Triple modular redundancy (TMR) can **recover** one corrupted bit
    - Or detect a double-bit error
  - Parity bits
    - *Even* parity bits are 0 if the # of 1s is even; 1 otherwise
      - Special case of CRC (polynomial is x+1)
    - *Odd* parity bits are 1 if the # of 1s is even; 0 otherwise

**DMR**:

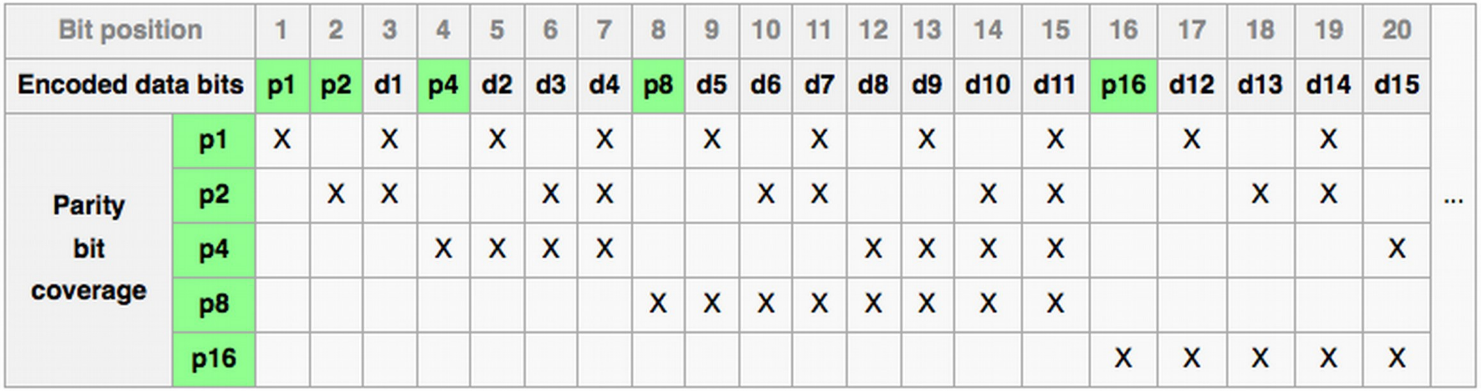
```
0 0  ok (value = 0)
0 1  SBE
1 0  SBE
1 1  ok (value = 1)
```

**TMR**:

```
0 0 0  ok (value = 0)
0 0 1  SBE (value = 0) or DBE
0 1 0  SBE (value = 0) or DBE
0 1 1  SBE (value = 1) or DBE
1 0 0  SBE (value = 0) or DBE
1 0 1  SBE (value = 1) or DBE
1 1 0  SBE (value = 1) or DBE
1 1 1  ok (value = 1)
```

# Recovery

- Hamming codes (often used in ECC memory) use parity bits
  - Bit position $2^i$ is a parity covering all bits with the (*i*+1)th least significant bit set
  - Each bit is covered by a unique set of parity bits
  - Error locations are identified by summing the positions of the faulty parity bits
  - Can detect & recover single-bit errors (can be extended to detect double-bit errors)
- Reed-Solomon codes are more complex (but widely used)
  - Function values or coefficients of a polynomial



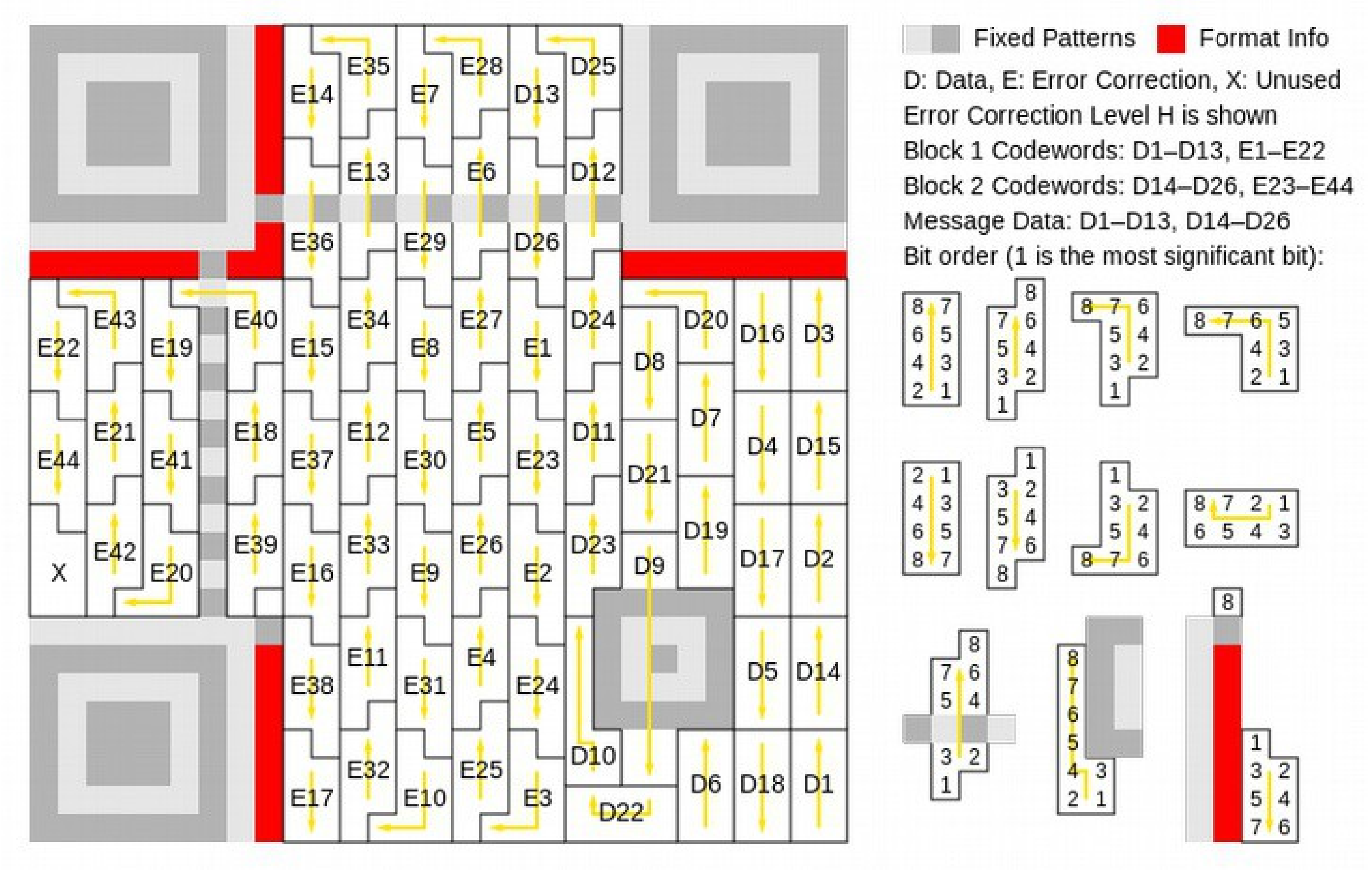
| Bit position | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Encoded data bits | | p1 | p2 | d1 | p4 | d2 | d3 | d4 | p8 | d5 | d6 | d7 | d8 | d9 | d10 | d11 | p16 | d12 | d13 | d14 | d15 | |
| Parity bit coverage | p1 | X | | X | | X | | X | | X | | X | | X | | X | | X | | X | | |
| | p2 | | X | X | | | X | X | | | X | X | | | X | X | | | X | X | | ... |
| | p4 | | | | X | X | X | X | | | | | X | X | X | X | | | | | X | |
| | p8 | | | | | | | | X | X | X | X | X | X | X | X | | | | | | |
| | p16 | | | | | | | | | | | | | | | | X | X | X | X | X | |

**Hamming code**: parity bits and corresponding data bits

from `https://en.wikipedia.org/wiki/Hamming_code`

# Recovery

- QR codes provide multiple recovery % options
  - Four levels: L (7%), M (15%), Q (25%), H (30%)



▨ Fixed Patterns 🟥 Format Info

D: Data, E: Error Correction, X: Unused

Error Correction Level H is shown

Block 1 Codewords: D1–D13, E1–E22

Block 2 Codewords: D14–D26, E23–E44

Message Data: D1–D13, D14–D26

Bit order (1 is the most significant bit):

# Recovery

- Software level
  - Log: record of operations (can enable recovery)
  - Checkpoint: snapshot of current state
    - Independent vs. coordinated checkpointing
    - Standalone vs. incremental checkpointing
    - Tradeoff: space vs. time (how much to save?)
  - Restore: revert system state to a checkpoint
    - May require replaying some calculations
    - Can a checkpoint be restored on a different system?
      - If so, how?