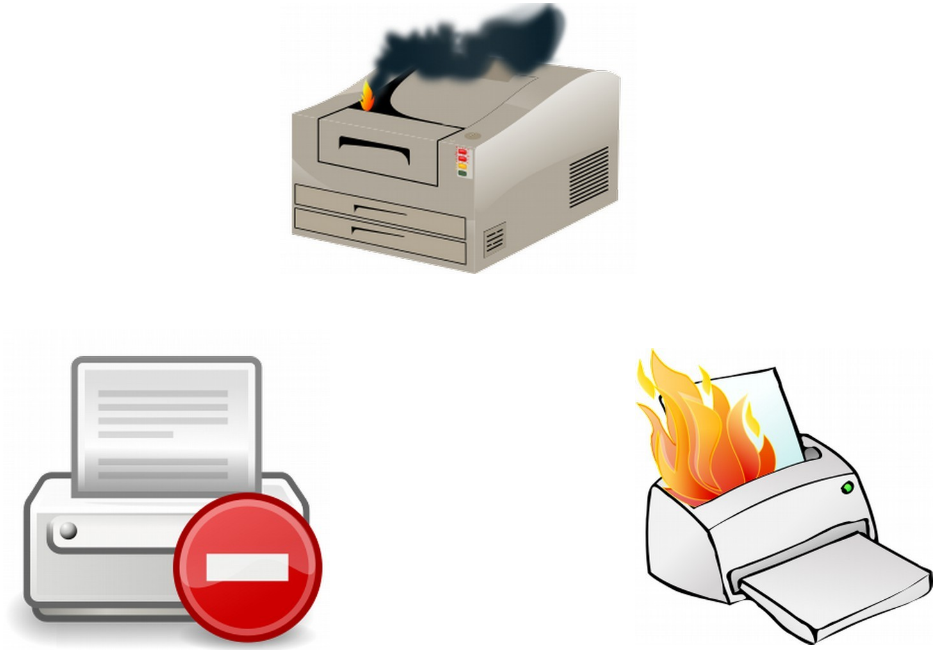
CS 470
Spring 2025

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`

Fault tolerance

- Desirable system properties
- Failure models
- Dealing with failure

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:
  - Consistency: reads see previous writes ("safety")
  - Availability: operations finish *relatively quickly* ("liveness")
  - Partition tolerance: failures don't affect correctness

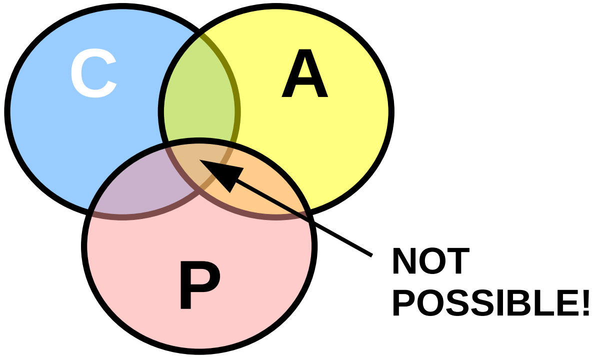
Systems design involves tradeoffs

Problem

- Which of the following is **most** important in a distributed system?
  - A. Consistency
  - B. Availability
  - C. Partition tolerance

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`

Problem

- Which of the following is **least** important in a distributed system?
  - A. Consistency
  - B. Availability
  - C. Partition tolerance

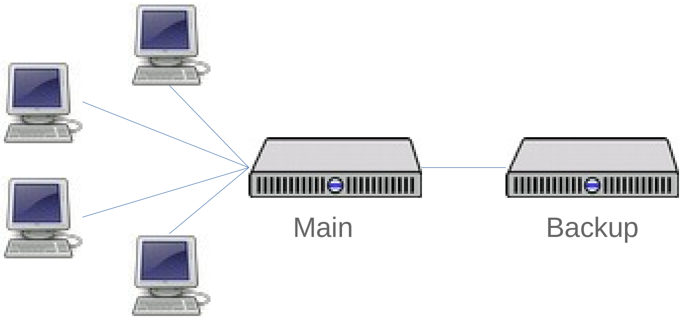
Consistency

- 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"

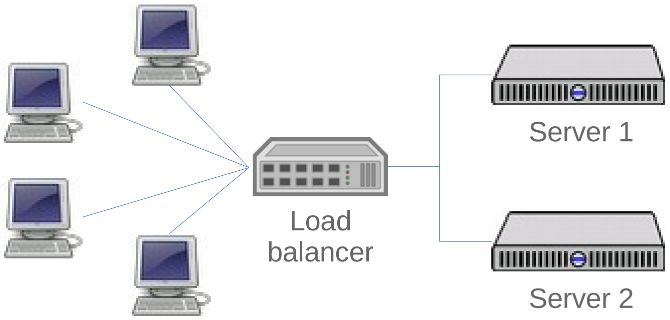
We’ll cover these models in more detail next week

Availability

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



Active-passive



Active-active

Availability

- The JMU CS software mirror consists of two servers `mirror3` and `mirror4`. At any given point, one is designated “primary” and handles all incoming traffic. If it fails, the other server will take over as primary. Which availability model is this closest to?
  - A. Active-passive
  - B. Active-active
  - C. Passive-passive

Failure models

- 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?
  - At what level of abstraction does the failure take place?
  - How hard is it to debug a failure?

Kinds of failures

- Soft vs hard failures
  - Soft failure: a.k.a. silent data corruption (SDC)
    - 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

Kinds of failures

- A buffer overflow bug causes inadvertent data corruption. What is this an example of? (select all that apply)
  - A. Soft failure
  - B. Hard failure
  - C. Partial failure

Kinds of failures

- One of the JMU cluster nodes goes offline due to a faulty power supply. What is this an example of? (select all that apply)
  - A. Soft failure
  - B. Hard failure
  - C. 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

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

Measuring failure

- If a JMU cluster hard drive dies on average every 5 years, what is the failure rate?
  - A. 0.05 failures/yr
  - B. 0.2 failures/yr
  - C. 0.5 failures/yr
  - D. 2.0 failures/yr
  - E. 5.0 failures/yr

Effects of failure

- 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

Levels of failure

- Some systems 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

Debugging 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
  - Single-event upset (SEU): caused by ions

https://www.thegamer.com/how-ionizing-particle-outer-space-helped-super-mario-64-speedrunner-save-time/

Debugging faults

- Suppose there is a bug in one of your CS 361 projects that is a result of improper synchronization, causing you to fail one of the automated tests. However, it does not reproduce in gdb. What kind of fault is this?
  - A. Permanent
  - B. Intermittent
  - C. Transient

Debugging faults

- Suppose your roommate trips and falls, accidentally hitting the switch on your surge protector and causing your desktop to lose power. What kind of fault is this?
  - A. Permanent
  - B. Intermittent
  - C. Transient

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
  - Electrical charge 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
- Ionizing radiation and cosmic rays
  - E.g., single-event upsets

Example: GPU fault study

The Titan supercomputer
has 18,688 GPUs

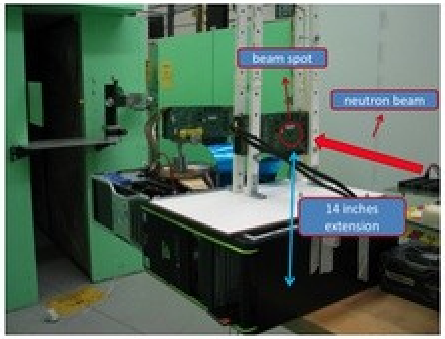
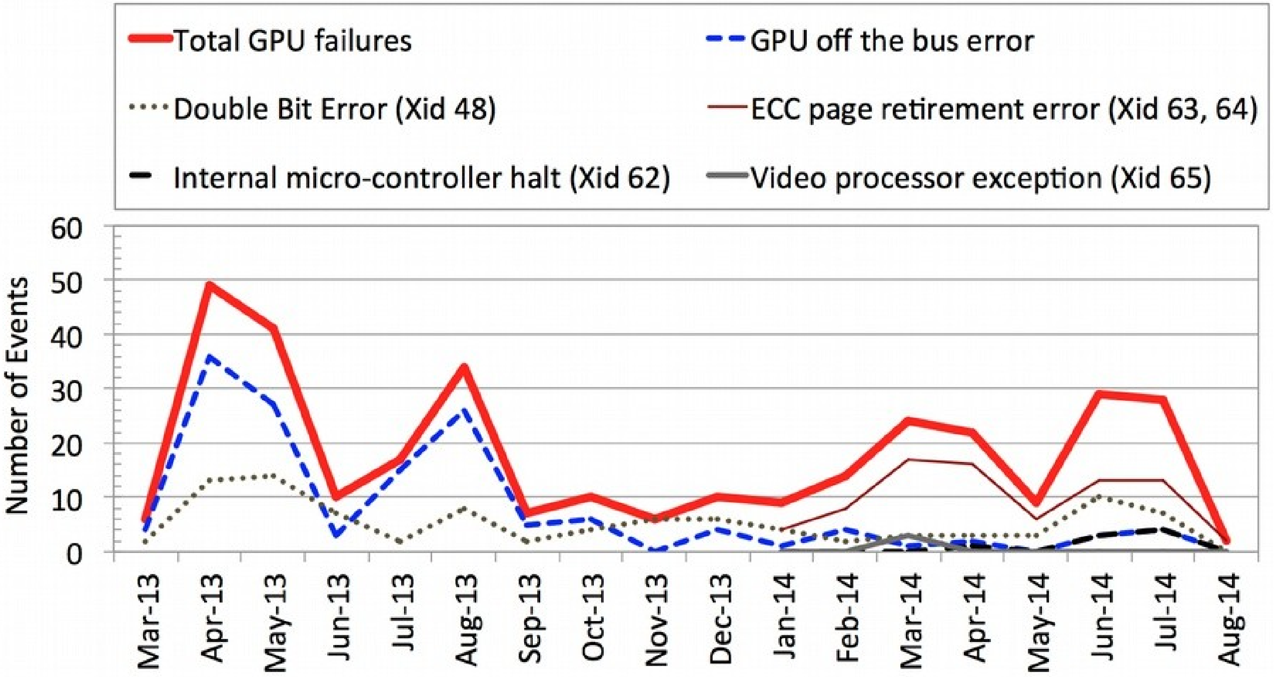
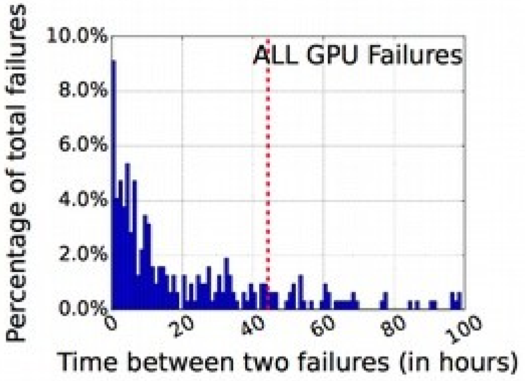
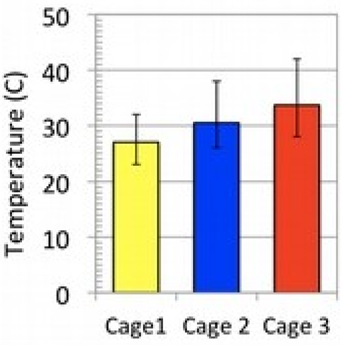
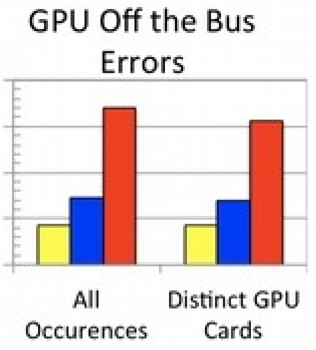


Figure 3: Radiation test setup inside the ICE House II, Los Alamos Neutron Science Center (LANSC), LANL. A similar setup was used at ISIS, Didcot, UK.









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.
`https://pdfs.semanticscholar.org/3b2c/8bb9471bd52a40b72a61bfede076f4d414b5.pdf`

Dealing with failure

- Prevention: eliminate the possibility of failure
  - Often impossible in a distributed system
- Detection: discovering failures
  - Active (pinging) vs. passive (wait for messages)
  - Issue: unreliability of timeouts
- Avoidance: divert around failure possibilities
  - Only possible in particular circumstances
- Recovery: restore valid system state after a failure
  - Forward error correction includes additional info for recovery

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)

Detection and avoidance

- How many total bits must be transmitted to **detect** a single-bit error?
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5

Detection and avoidance

- How many total bits must be transmitted to detect a **double-bit** error?
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5

Recovery

- How many total bits must be transmitted to **correct** a single-bit error?
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5

Recovery in 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:

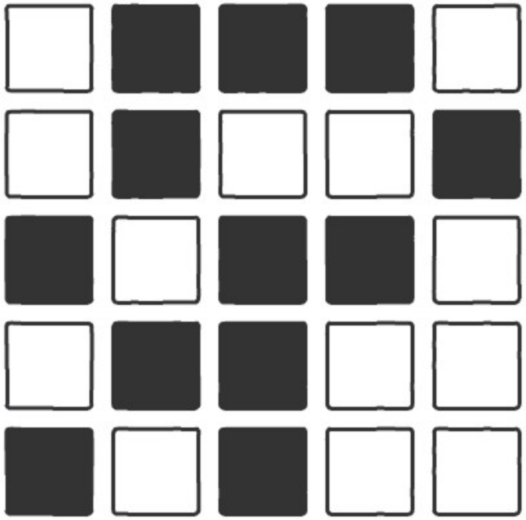
```
0 0 0  ok (value = 0)
0 0 1  SBE (value = 0) or DBE
0 1 0  SBE (value = 0) or DBE
0 1 1  SBE (value = 1) or DBE
1 0 0  SBE (value = 0) or DBE
1 0 1  SBE (value = 1) or DBE
1 1 0  SBE (value = 1) or DBE
1 1 1  ok (value = 1)
```

Parity

- Which of the following bytes has been corrupted during transmission, assuming 7-bit even parity?
  - A. `01010101`
  - B. `10100101`
  - C. `00001111`
  - D. `01101000`
  - E. `11111111`

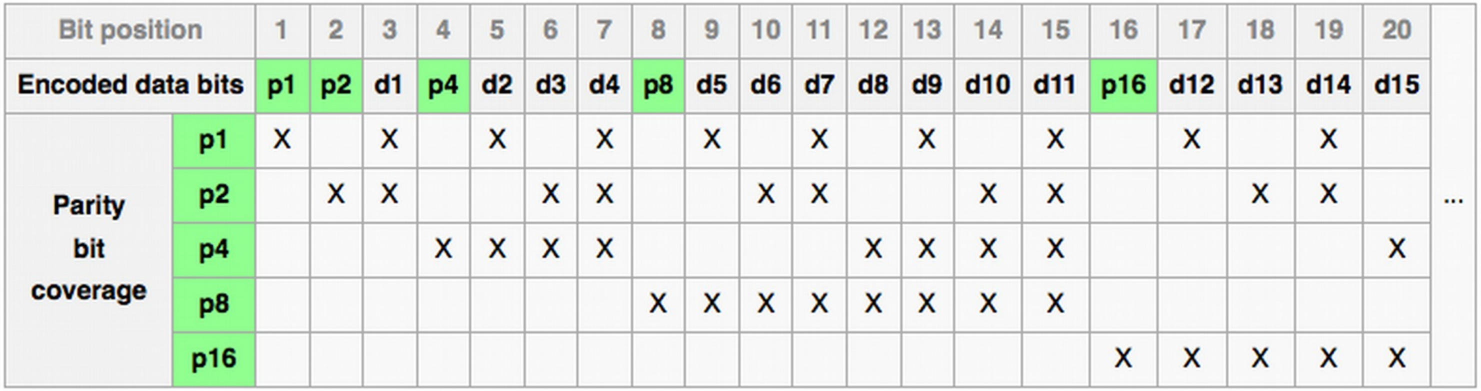
Fun activity/demo

- CS Unplugged activity “Parity Magic”
  - https://www.csunplugged.org/en/topics/error-detection-and-correction/parity-magic-junior/



Recovery codes

- 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 SBEs (can be extended to detect DBEs)
- Reed-Solomon codes are more complex (but widely used)
  - Function values or coefficients of a polynomial



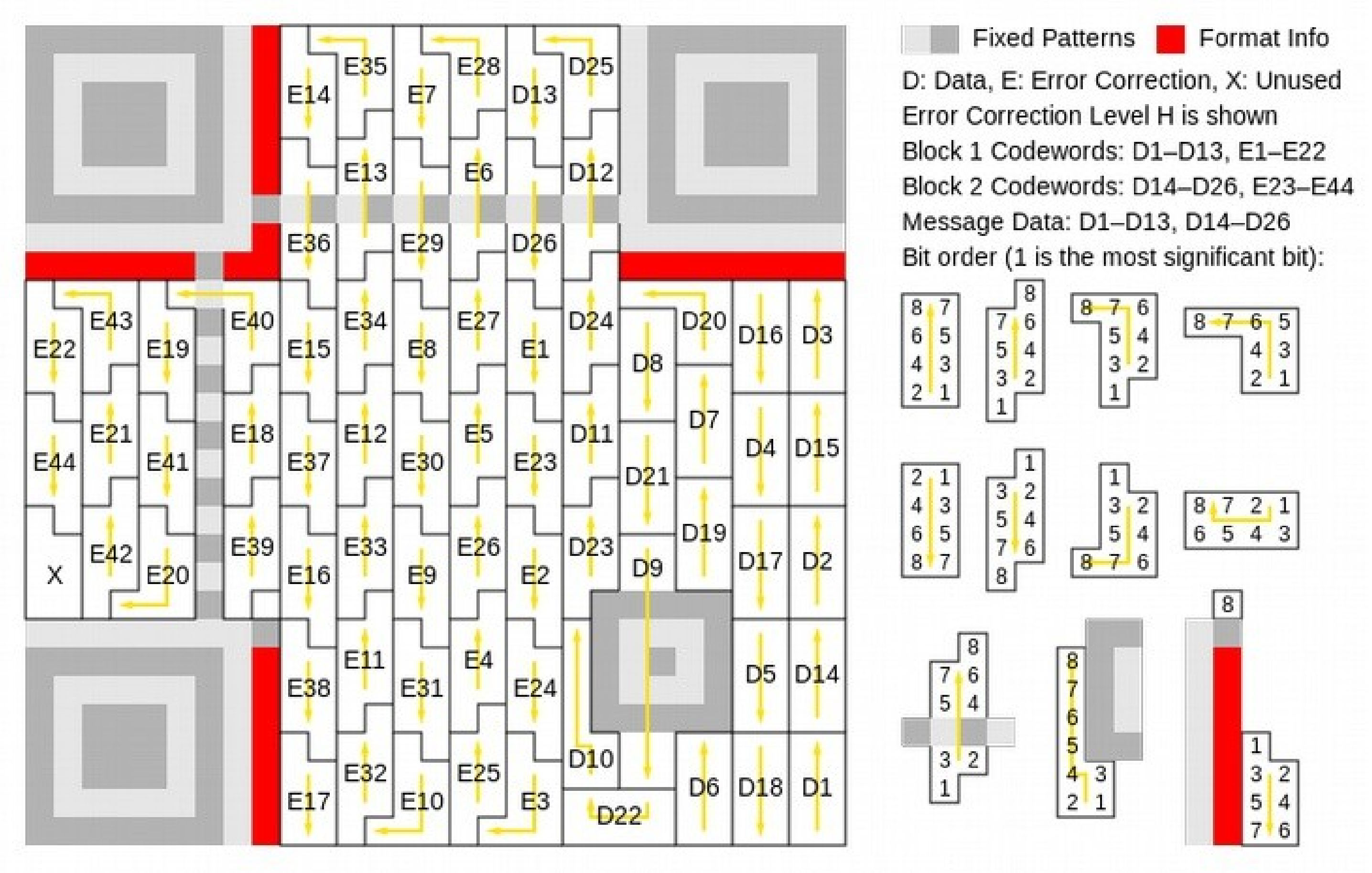
Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X		
	p2		X	X			X	X			X	X			X	X			X	X		...
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X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

- 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 in software

- 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?