S08E04 Recap: We move from the macroscopic chaos of cross-server clock skew to the microscopic hazards of single-service interleaved execution.



Hard Problems in Computing: S08E05

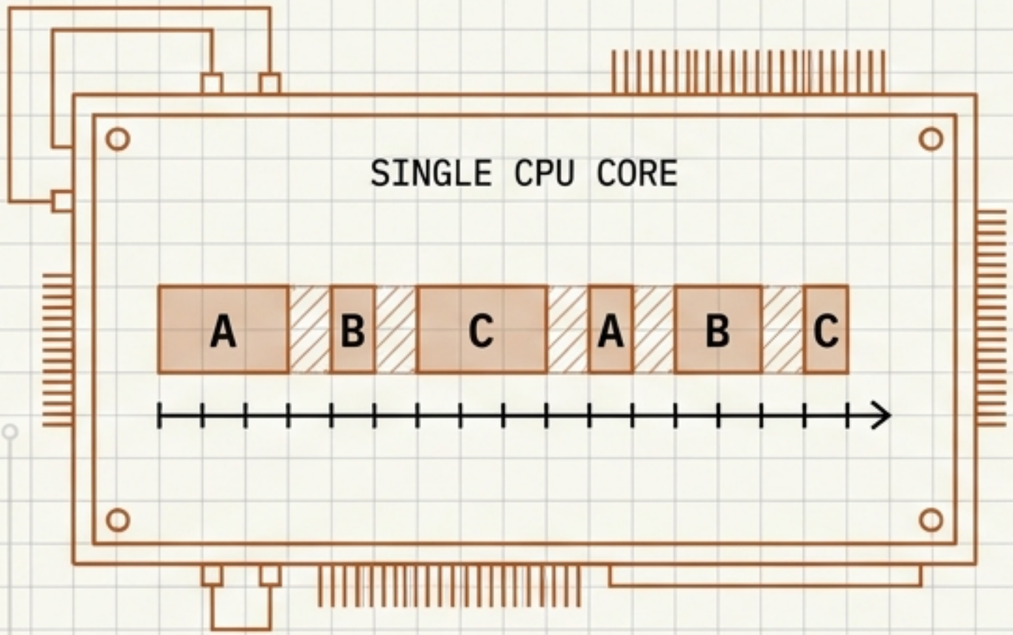
Concurrency vs. Parallelism

From Hardware Atomics to Deterministic Simulation

Gemini Notebook

1. STRUCTURE VS. EXECUTION

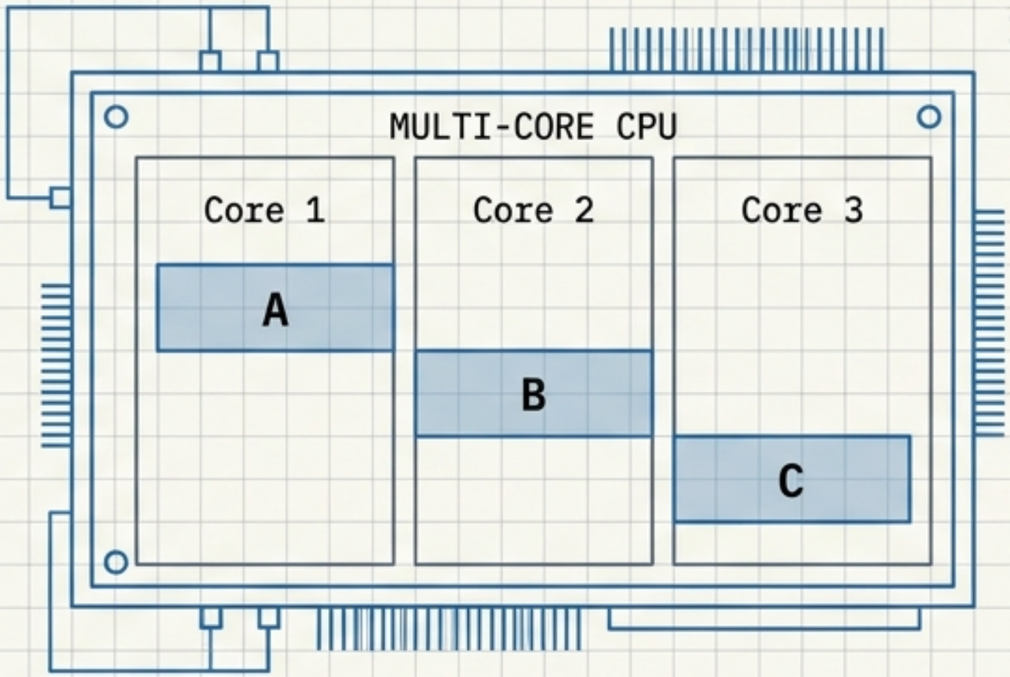
CONCURRENCY



The composition of independently executing processes.

Structure. Dealing with lots of things at once.

PARALLELISM



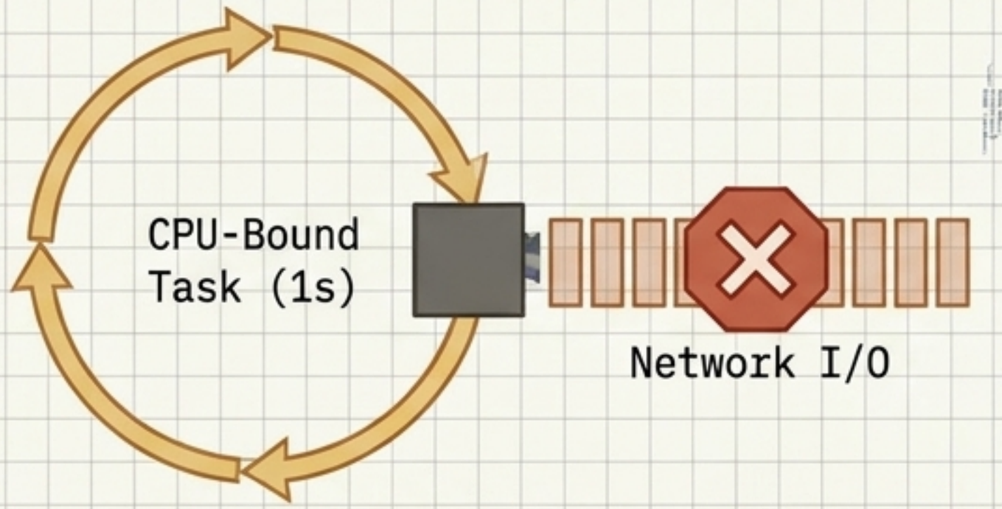
The simultaneous execution of (possibly related) computations.

Execution. Doing lots of things at once.

Gemini Notebook

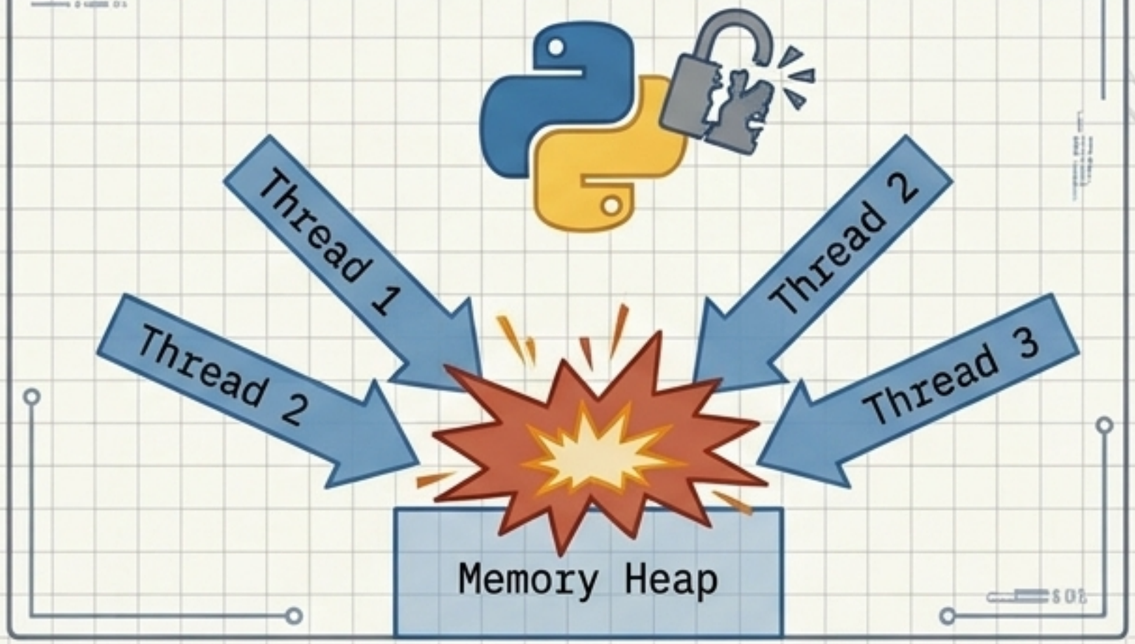
1. STRUCTURE VS. EXECUTION

Cooperative Starvation (Node.js / asyncio)



Node.js runs on a single thread. A 1-second synchronous calculation delays all concurrent await IO tasks by exactly 1 second. The loop is starved.

True Multi-Core Hazards (Python PEP 703)



Removing the GIL exposes true parallel data races. Global mutual exclusion is replaced by Biased Reference Counting (BRC) and deferred collection.

Gemini Notebook

2. CONCURRENCY MODELS ARCHITECTURE MATRIX

Model	Memory Overhead	Preemption Points	Data-Race Safety	CPU-Bound Behavior	Failure Isolation
Cooperative Event Loops (Node.js/asyncio)	Low	Yield (await)	Safe (1 core)	Starves Loop	Process crashes
Preemptive OS Threads (Java/C++)	High (MBs)	OS Scheduler	Unsafe (Locks req.)	Scales well	Thread/Process crashes
M:N Green Threads (Go goroutines)	Very Low (KBs)	Func calls / I/O	Unsafe (Channels/Mutex)	Work-stealing	Panics crash process
Actor Processes (Erlang/BEAM)	Low	Reductions (per op)	Safe (Share-nothing)	Preempts fairly	Isolated (Let it crash)
Compile-Time Ownership (Rust)	Zero-cost	Await/Yield	Safe (Send/Sync)	Thread-dependent	Panic caught/isolated

Gemini Notebook

2. CONCURRENCY MODELS ARCHITECTURE MATRIX

Runtime Detection (Go)

```
WARNING: DATA RACE
```

```
go func() { m[key] = 1 }()
go func() { m[key] = 2 }()
```

M:N Work-Stealing is incredibly cheap, but compiler allows data races. Safety requires runtime instrumentation (go test -race) or strict sync.Mutex / channel discipline.

Compile-Time Prevention (Rust)

```
error[E0277]: `Rc<T>` cannot be
  sent between threads safely
```

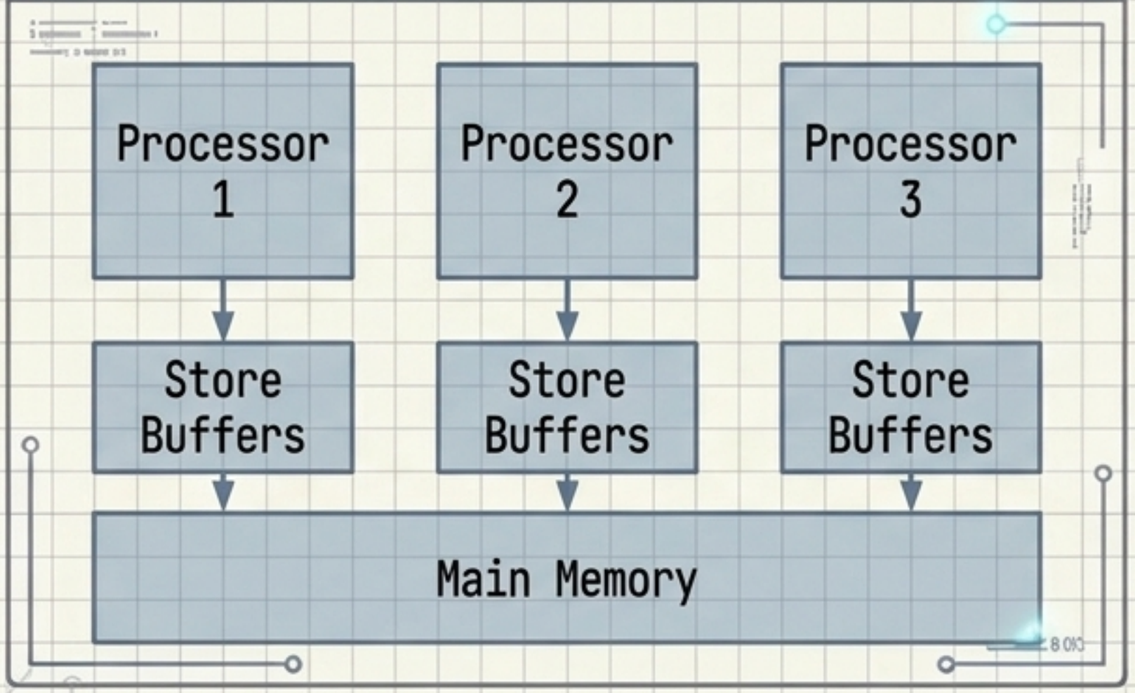
```
thread::spawn(move || {
    data.push(1);
});
```

Compile-Time Ownership. Send (safe to move) and Sync (safe to share). The compiler proves the absence of data races before binary generation.

Gemini Notebook

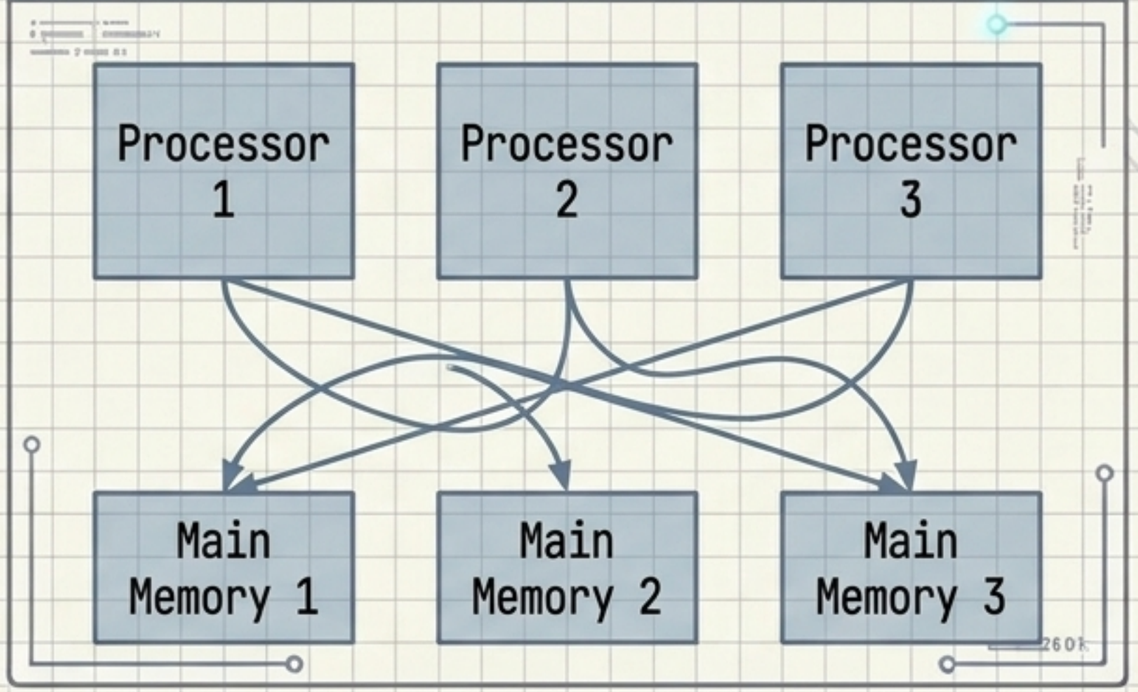
3. HARDWARE MEMORY MODELS

x86 TSO (Total Store Order)



Processors see their own writes first. However, all processors agree on the **total** order of writes reaching main memory.

ARM/POWER Relaxed Reordering



No total store order. Reads can be delayed; threads may observe different writes in different orders. Requires explicit **Data-Race-Free Sequential-Consistency (DRF-SC) barriers.**

Gemini Notebook

The Atomic Primitive

```
int CompareAndSwap(int *addr,
    int expected, int new) {
  if (*addr == expected) {
    *addr = new; return 1;
  }
}
```

Atomicity at the silicon level. The foundation of lock-free data structures.

The ABA Problem Timeline



Thread 1 reads value ‘A’. (Preempted).

Thread 2 changes ‘A’ to ‘B’.

Thread 2 changes ‘B’ back to ‘A’.

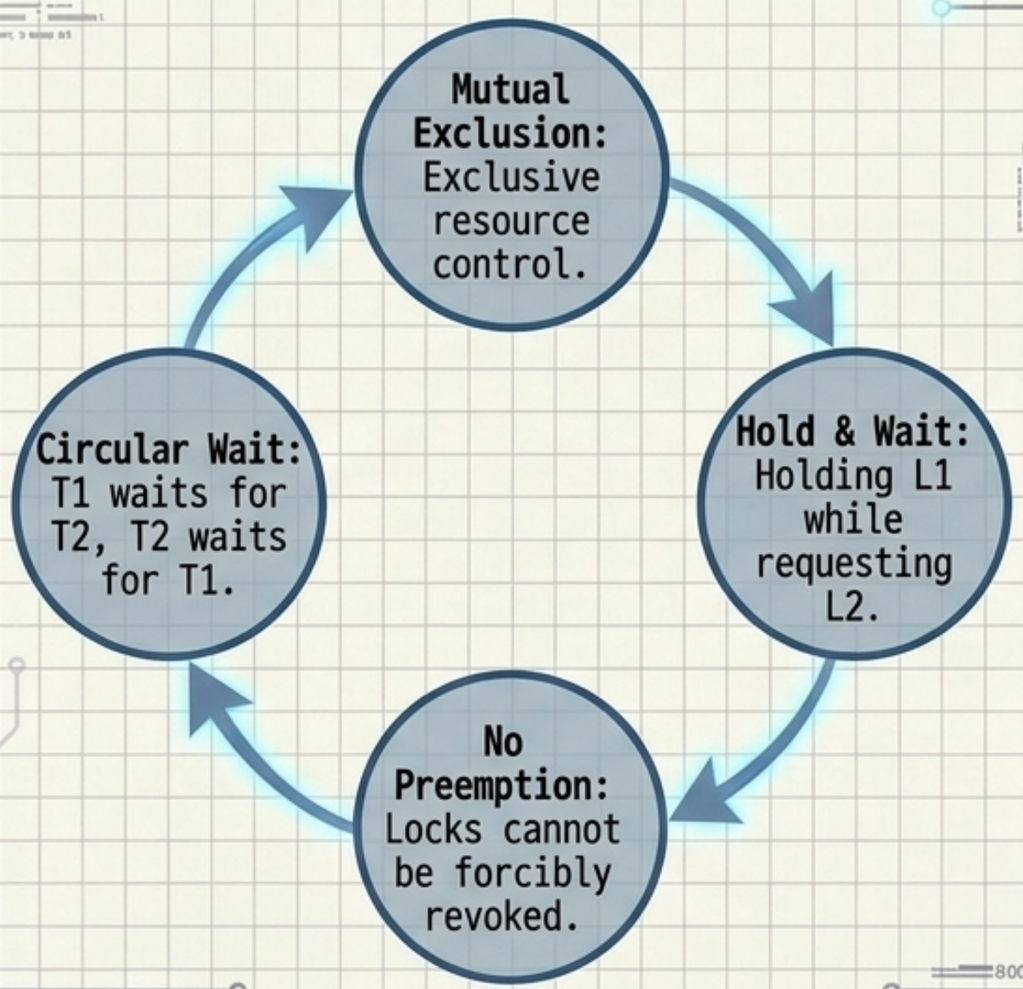
Thread 1 resumes, executes CAS(‘A’, ‘new’). It succeeds.

The illusion of untouched state. The data structure is corrupted. Solved strictly by hazard pointers or version tagging.

Gemini Notebook

3. LOCKS & DEADLOCKS UNDER THE HOOD

Coffman's 4 Conditions (The Cycle)



Dijkstra's Total Lock-Ordering (The Fix)



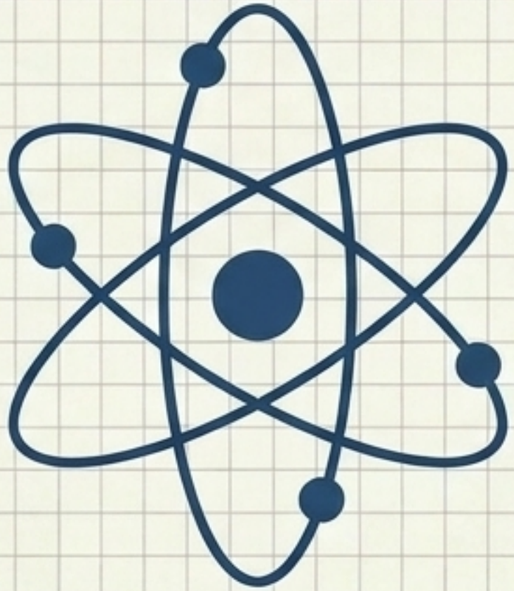


Always acquire lower-numbered locks first.

Contrast with **Livelock** (continuous state change, no progress) and **Starvation** (permanently denied access).

Gemini Notebook

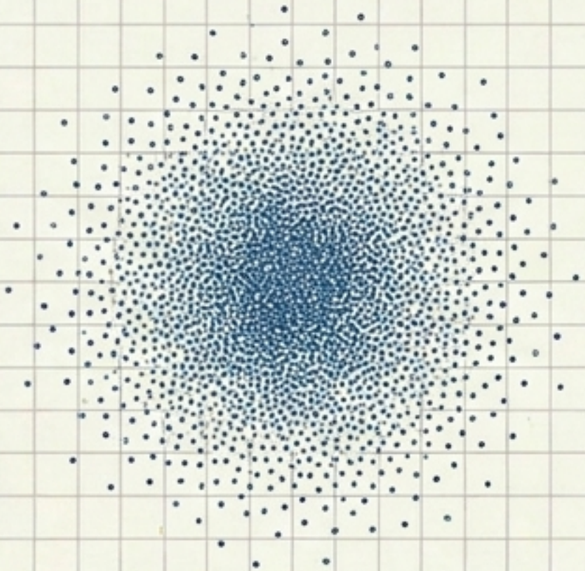
Bohrbugs (Jim Gray, 1985)



Characteristics: Deterministic. Easily reproducle. If you run the code, it crashes the exact same way every time.

Mitigation: Handled by standard sequential unit tests.

Heisenbugs



Characteristics: The bug alters its behavior or vanishes entirely when observed. Adding printf or attaching a debugger alters memory timings, hiding the race condition. Rooted strictly in concurrency interleavings.

Gemini Notebook

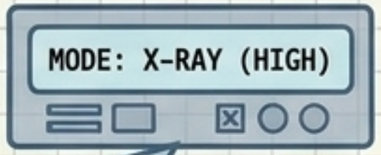
Therac-25 — A Fatal Concurrency Disaster (Nancy Leveson Investigation)

STEP 1:



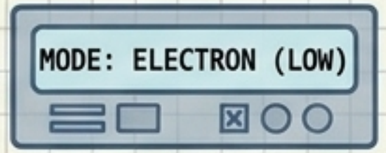
Operator enters 'X' (high power)

MODE: X-RAY (HIGH)



Makes a typo. Quickly edits to 'E' (low power)

MODE: ELECTRON (LOW)

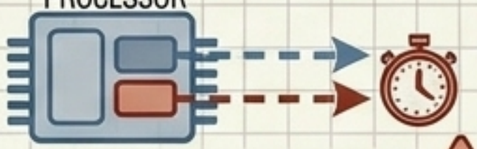


Rapid Edit Command

USER INTERFACE (UI THREAD)



PROCESSOR



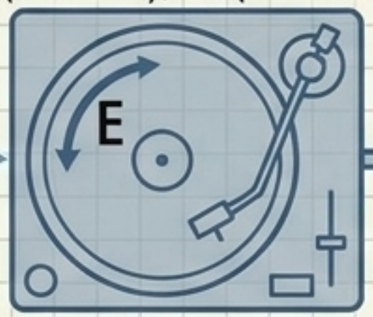
DESYNCHRONIZATION

INTERLEAVING & RACE CONDITION

Concurrent UI thread processes the rapid edit, updating the display but not properly synchronizing with the hardware command queue.

'X' (shielded), 'E' (unshielded)

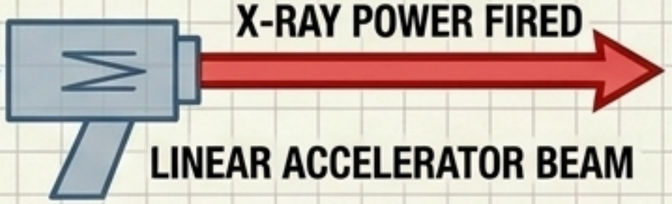
3. HARDWARE STATE CHANGES



MOTOR



Turntable moves to 'E' position (unshielded), but the beam fires at 'X' power.



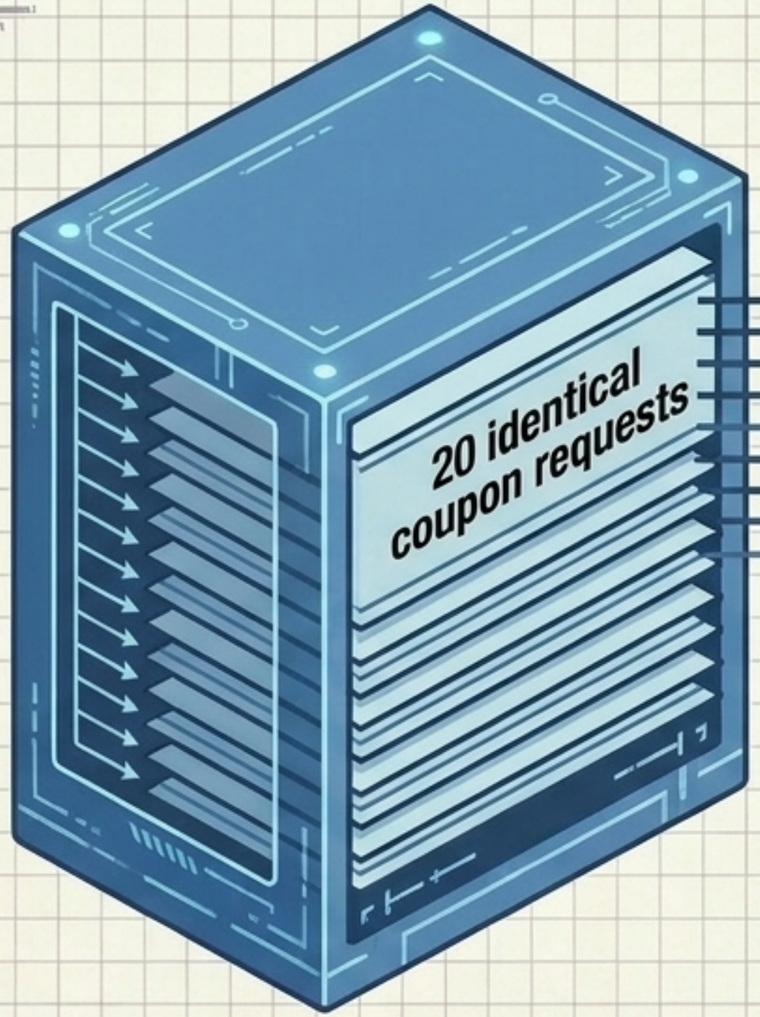
X-RAY POWER FIRED

LINEAR ACCELERATOR BEAM

Software locks were removed in favor of software checks without proper synchronization, directly causing fatal radiation overdoses.

Gemini Notebook

HTTP/2 Single-Packet Races & CWE-367 (TOCTOU)



Exploit Logic

```
1. if (valid(coupon)) { // Time-of-Check
2.     await database.update();
3.     applyDiscount();} // Time-of-Use
```

MICROSECOND WINDOW

TELEMETRY: By packing 20 requests into a single TCP packet, attackers bypass network jitter.
All 20 requests hit the 'Check' (Step 1) simultaneously before any single thread reaches the 'Use' (Step 3).
Result: 20x discounts applied.

Gemini Notebook

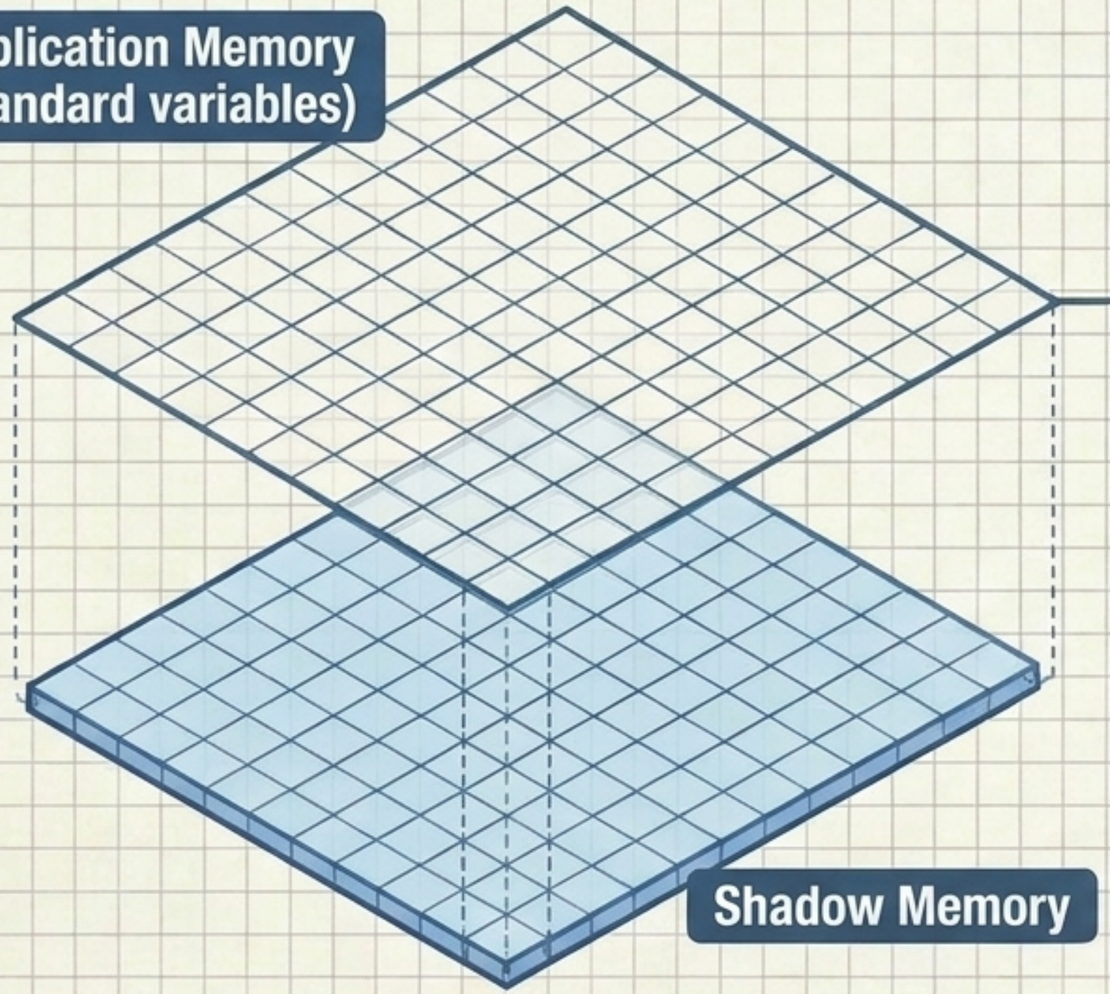
- [x] Unit Tests Passing
- [x]
- [x]
- [x]
- [x]

184,000+ Interleavings.

TELEMETRY: Why sequential unit tests never trigger interleaving bugs. If Thread A has 10 instructions and Thread B has 10 instructions, there are over 184,000 possible execution interleavings. Sequential tests only validate one idealized, isolated path.

Gemini Notebook

Weaponizing Shadow Memory (TSan & go test -race)

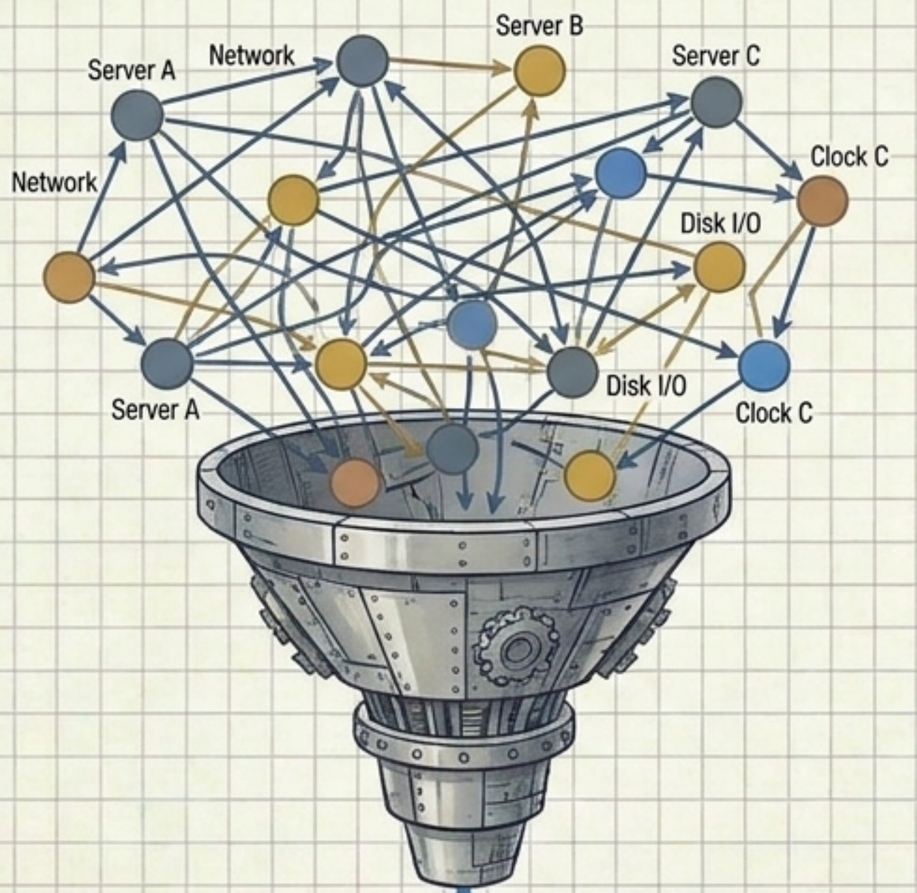


Mechanism:

1. Tracks every memory access using a vector clock.
2. Records [Thread ID | Timestamp | Access Type (R/W)] in the hidden shadow memory array.
3. Evaluation: If a Read and Write occur to the exact same location from different threads without a mathematical ‘Happens-Before’ synchronization edge...
4. Result: **RACE DETECTED** alert.

Gemini Notebook

Deterministic Simulation Testing (FoundationDB)



Telemetry: Catching 1-in-a-billion races in CI.

- Mock out the network, disk, and clock.
- Multiplex the entire cluster's concurrent actors onto one single deterministic thread.
- Seed the random number generator.

Result: If a Heisenbug occurs, replay the exact seed to reproduce it 100% of the time.

Deterministic Thread

100% Reproducibility

Gemini Notebook

Trap 1: The await RMW (Read-Modify-Write)

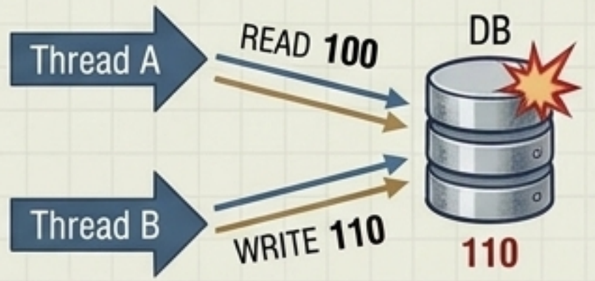
AI Code

```
bal = await db.get();
await db.set(bal + 10)
```

System Reality

Massive data loss under load (PostgreSQL Read Committed).

The classical TOCTOU failure.



Trap 2: The Unbounded Fan-Out

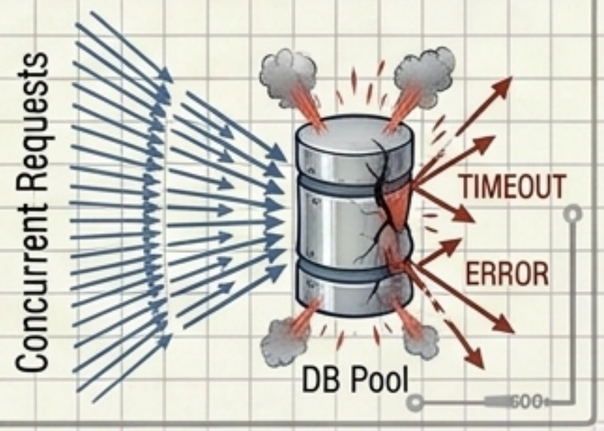
AI Code

```
await Promise.all(
hugeArray.map(insertDb))
```

System Reality

Instantly exhausts the DB connection pool, causing cascading timeouts and DoS.

Fix requires p-limit or strict Semaphore(N) bounding.



Gemini Notebook

Trap 3: FastAPI Module-Global Caches

AI Code

```
global_cache = {};

@app.get('/')
async def handler(user):
  global_cache[user] = data
```

System Reality

Concurrent requests mutate the same dictionary. In Python 3.13+ (GIL removed), this becomes a hard crash/data corruption vector.



Trap 4: React useEffect Missing Cleanup

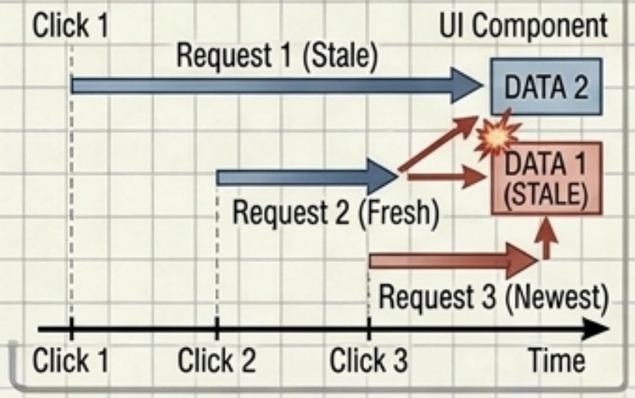
AI Code

```
Fetching data on component
mount without cleanup.
```

System Reality

User clicks fast → multiple inflight requests → older request resolves after newer request → UI displays stale data.

Fix: Mandatory `AbortController` / `AbortSignal` implementation.



Gemini Notebook

CLAUDE.md / AGENTS.md

[INVARIANT 1] Use single-statement SQL (UPDATE...RETURNING or SELECT...FOR UPDATE). NEVER hold transactional logic across external HTTP await calls.

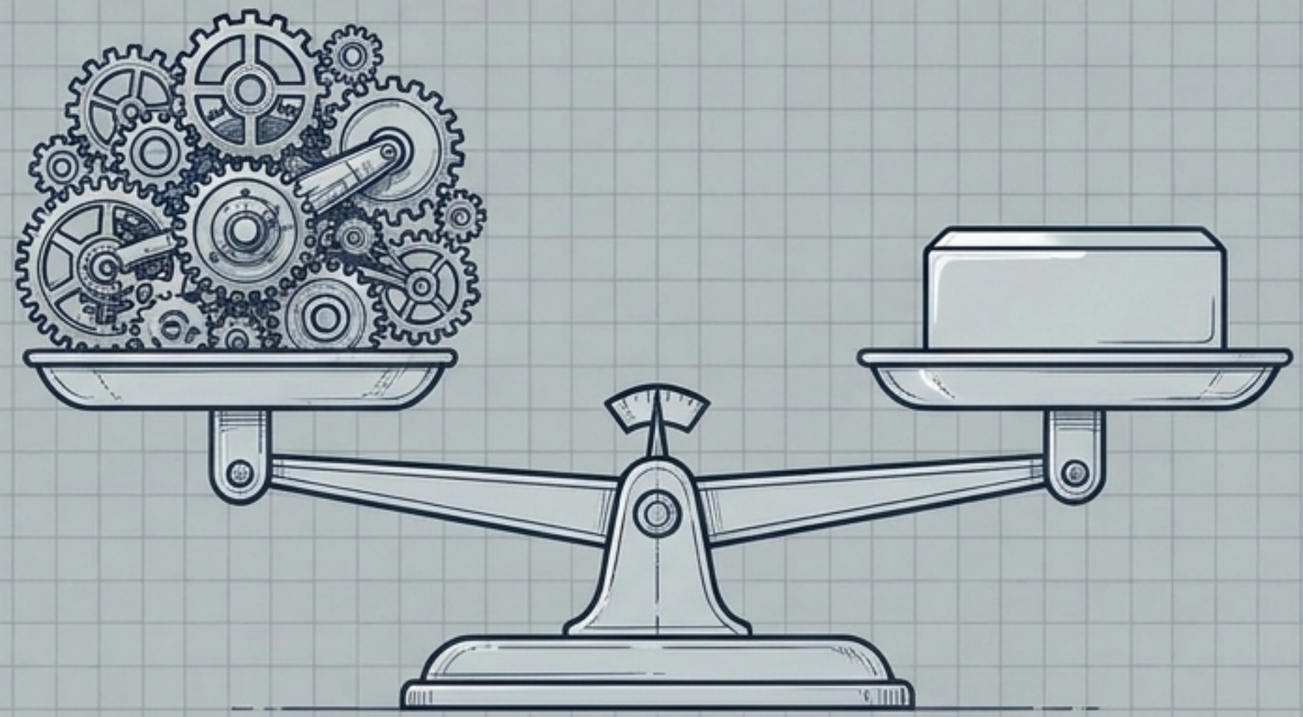
[INVARIANT 2] Always bound concurrent fan-out using p-limit or Semaphore(N).

[INVARIANT 3] All React async effects MUST implement AbortController cleanup logic.

[INVARIANT 4] Write parallel barrier tests for all critical state mutations.

Gemini Notebook

7. YAGNI RUBRIC & NEXT UP



When to say NO (Complexity)

- Lock-free CAS gymnastics
- Serializable transaction retry loops
- custom in-memory distributed mutexes

When to say YES (Simplicity)

- Single SQL predicates (let the database lock it)
- stateless worker processes
- deterministic test seeding

Next Up in S08E06: Distributed Consensus & Split-Brain: When clocks lie and networks partition.

Gemini Notebook