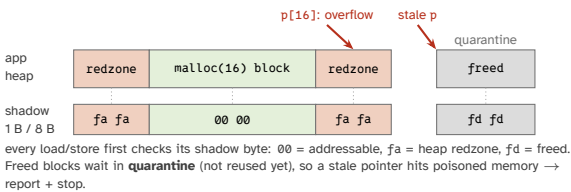
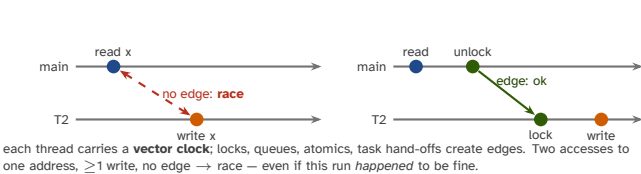


In one line: Memory and threading bugs corrupt silently and crash *later, elsewhere*. Sanitizers are **compiler instrumentation + a runtime** that check every access as it happens (ASan: is this byte addressable? TSan: is there a happens-before edge?) and stop at the **first bad access** with the stacks that explain it. Runtime checkers (Main Thread Checker, Thread Performance Checker, exclusivity) need no special build. Price: slowdown — so they live in Debug schemes, test plans and CI, never in the shipped app.

ASan: shadow memory + redzones + quarantine



TSan: is there a happens-before edge?



Tool → bug class → cost → combine?

tool	catches	cost	combine
ASan	overflow (heap/stack/global), use-after-free/return/scope, double free. C + Swift	2-5 × CPU, 2-3 × mem	not TSan ; + UBSan; device ok
TSan	data races, Swift access races, bad mutexes, thread leaks. C + Swift	2-20 × CPU, 5-10 × mem	not ASan ; Simulator/ macOS only + ASan or TSan (<i>unverified</i>)
UBSan	C UB: overflow, shifts, null, misaligned, bad enum/bool. Not Swift	≈20 %	
MTC	UIKit/AppKit API off main	1-2 %, ≤100 ms launch	default on, no rebuild
TPC	priority inversion; sync I/O, networking on main	low	on for Run, no rebuild
Malloc Scribble	free → 0x55, alloc → 0xAA	cheap	checkbox
Guard Edges	guard pages round ≥4 KB blocks	memory	checkbox
Guard Malloc	page per alloc: fault at once	huge	Mac + Sim only
Zombies	message to a freed ObjC object	never frees	scheme / Allocations
Exclusivity	overlapping inout/mutating (Swift)	small	always, Release too

MTC = Main Thread Checker · TPC = Thread Performance Checker.

How they work

- **ASan**: the compiler inserts a shadow check before every load/store; the allocator adds poisoned *redzones* around each block and *quarantines* freed memory. Apple: does *not* find leaks, uninitialised reads or integer overflow.
- **TSan**: instruments accesses *and* synchronisation, keeps the last few accesses per 8 bytes in shadow cells, compares vector clocks (*unverified*). Only paths you *executed* are checked — drive concurrency in tests. Swift 6 checks isolation at compile time; TSan still owns locks, @unchecked Sendable, C/ObjC and unsafe code.
- **Main Thread Checker / Thread Performance Checker**: runtime interposition, no recompile, show as purple runtime issues; can *pause on issue*. TPC flags a user-interactive thread waiting on lower-QoS work (e.g. a semaphore — no priority donation).
- **Zombies**: Xcode 26 removed the *Zombies template* — use Allocations + *Enable NSZombie detection*, or the scheme's *Zombie Objects*. Freed objects become zombies that log “message sent to deallocated instance”.
- **Exclusivity**: statically where it can, dynamically for class properties, globals, escaping captures; traps with “Simultaneous accesses to ..., but modification requires exclusive access”.

Turning them on

Scheme → Run/Test → **Diagnostics**; test plans: one configuration per sanitizer, *Runtime API Checking* “On (as Failure)”. CI: xcodebuild test -enableAddressSanitizer YES, then a second run with -enableThreadSanitizer YES; -enableUndefinedBehaviorSanitizer YES; swiftc -sanitize=address. Tune with ASAN_OPTIONS (Xcode 26: C++ container-overflow checks on by default — detect_container_overflow=0 to silence).

Reading an ASan report (abridged)

```
==4242==ERROR: AddressSanitizer: heap-use-after-free
on address 0x60200000eff0 ...
READ of size 8 at 0x60200000eff0 thread T0
#0 parse_header buf.c:42 // 1 WHERE: bad access
0x60200000eff0 is located 0 bytes inside of 16-byte region
freed by thread T0 here:
#1 conn_close conn.c:88 // 2 WHO freed it
previously allocated by thread T0 here:
#1 conn_open conn.c:31 // 3 WHO made it
SUMMARY: AddressSanitizer: heap-use-after-free buf.c:42
```

Read the **bug type** (heap-buffer-overflow, stack-use-after-return...), then three stacks: the access, the free, the allocation. The fix is almost always in #2 or #3, not #1.

Reading a TSan report (abridged)

```
WARNING: ThreadSanitizer: data race (pid=4242)
Write of size 8 at 0x7b0800001234 by thread T3:
#0 Cache.store(...) Cache.swift:18
Previous read of size 8 at 0x7b0800001234 by main thread:
#0 Cache.lookup(...) Cache.swift:11
Location is heap block of size 48 ... allocated by main thread
Thread T3 (running) created by main thread at: ...
SUMMARY: ThreadSanitizer: data race Cache.swift:18
```

Two stacks, two threads, one address, no edge. Fix the *ownership* (actor, serial queue, Mutex), not the timing.

Example — exclusivity trap at run time

```
final class Counter { var n = 0 }
func bump(_ x: inout Int, by c: Counter) { x += c.n }
let c = Counter()
bump(&c.n, by: c) // write access to c.n open, then read c.n
// traps: "Simultaneous accesses to ..." (see above)
```

Interview traps

- ! “Run ASan and TSan together” — impossible; two test-plan configs.
- ! TSan on a physical iPhone — not supported; use the Simulator.
- ! UBSan for Swift — Swift traps on overflow/bounds already; UBSan is C-only.
- ! Clean TSan run ≠ race-free: only executed interleavings count.

Remember

ASan = where memory, TSan = when threads, UBSan = C rules, MTC/TPC = the main thread. Crash here, not later.

Likely questions

1. How does ASan find an overflow? — shadow byte check hits a redzone.
2. Why not ship with ASan? — 2-5 × slower, 2-3 × memory.
3. EXC_BAD_ACCESS in ObjC? — Zombies, then ASan.
4. Priority inversion? — TPC; replace the semaphore wait with async/await.