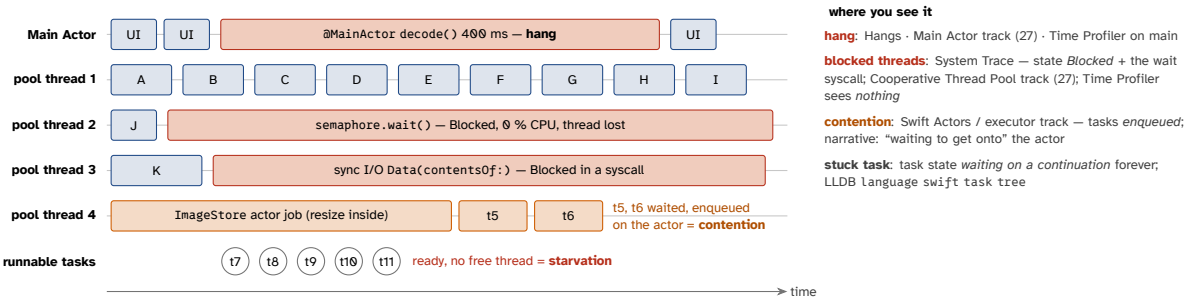


In one line: Tasks run on the **cooperative thread pool** (~one thread per core, **never grows**) or on the **main actor's** main thread. The runtime's contract is **forward progress**: *await* suspends and frees the thread; a *blocked* thread is simply lost. Three failures follow — a **blocked main actor** (hang), **actor contention** (parallel code turned serial), **pool starvation** (threads parked in waits). Concurrency instruments show task/actor/executor *state*; System Trace shows *why* a thread is not running.



How it works

- **Pool width** ~CPU cores, fixed: no thread explosion, but no replacement for a blocked thread either. Unsafe in async code: DispatchSemaphore, NSCondition, pthread_cond, group.wait(), sync I/O.Mutex / os_unfair_lock: only short critical sections, never across an await.
- **Main actor** = the main thread's serial executor. Xcode 26 new projects default to MainActor isolation (unverified), and with *Approachable Concurrency* a nonisolated async func runs on the caller's actor (SE-0461) — mark CPU work @concurrent to leave main.
- **Actor** = serial executor: one job at a time, callers *enqueue*. Keep only state mutation inside; do the heavy part nonisolated / @concurrent.
- **Swift Concurrency** template: *Swift Tasks* + *Swift Actors*. Running / Alive / Total Tasks; per-task state track; Task Forest (parent/child); creation backtrace; narrative ("waiting on Task X"). 26: shows Task(name:) names (Swift 6.2).
- **Xcode 27: Swift Executors** instrument — tracks for the *Cooperative Thread Pool*, the *Main Actor* and each TaskExecutor / SerialExecutor (OS 27; older: "Unknown executor"). **Swift Task Collection** tracks group tasks by name or creation site (lifetimes or states). *Profile* detail = call tree of samples taken while a Task / Collection / Actor / Executor was *Running* (record with Time or CPU Profiler).
- **System Trace**: thread states, syscalls, VM faults — one plot in 27; ←/→ follows the scheduling chain (who made this thread runnable); 27 graphs **thread priority over time** (starvation). Thread Activity (27): effective QoS per thread.
- **Thread Performance Checker** (on for Run by default): priority inversions + non-UI work on main (sync I/O, networking) → Issue navigator.
- **TSan**: same address, ~same time, one a write (+ thread leaks, uninitialised mutexes). macOS or **Simulator only**; 5-10× memory, 2-20× slower. Swift 6 errors cover checked code — TSan catches @unchecked Sendable, unsafe pointers, C/ObjC.
- **LLDB**: 26 steps follow a Task across threads; 27 language swift task tree prints every task the debugger knows. LIBDISPATCH_COOPERATIVE_POOL_STRICT=1 (scheme env): pool width 1, so a blocking wait deadlocks now, in dev.

Example — suspend, don't block

```
// BAD: parks a pool thread until the callback fires
func thumb(_ u: URL) async -> UIImage? {
    let sem = DispatchSemaphore(value: 0); var img: UIImage?
    legacyLoad(u) { img = $0; sem.signal() }
    sem.wait() // Blocked, thread lost
    return img
}

// GOOD: the task suspends; resume exactly once
func thumb(_ u: URL) async -> UIImage? {
    await withCheckedContinuation { c in
        legacyLoad(u) { c.resume(returning: $0) } }
}

// CPU work off the caller's actor (6.2); a named task
@concurrent func decode(_ d: Data) async -> UIImage? {
    UIImage(data: d)
}

Task(name: "thumb") { await show(thumb(url)) }
```

Symptom → evidence → fix

symptom	evidence	fix
UI frozen	one long Main Actor job	@concurrent
slow, CPU idle	pool threads <i>Blocked</i> (System Trace)	await, continuation
parallel = serial	tasks <i>enqueued</i> on one actor	shrink / shard actor
never finishes	stuck waiting on a continuation	resume on every path
inversion	TPC issue; priority graph (27)	no wait on lower QoS

Interview traps

- ! "Spawn more tasks" — the pool never grows; a blocked thread stays lost.
- ! nonisolated async ≠ background under SE-0461: it inherits the caller's actor. @concurrent means "off it".
- ! Task.detached to "get off main" drops priority and task-locals.
- ! TSan: not on a device. Swift 6: does not check @unchecked.

Remember

Suspend, never block — the pool never grows. Main actor = main thread; one actor = one lane.

Likely questions

1. Semaphore in async code? — parks a pool thread: starvation.
2. Actor contention? — actor/executor track: time *enqueued*.
3. Priority inversion? — high QoS waits on low QoS; TPC flags it.
4. Stuck task? — a continuation never resumed.