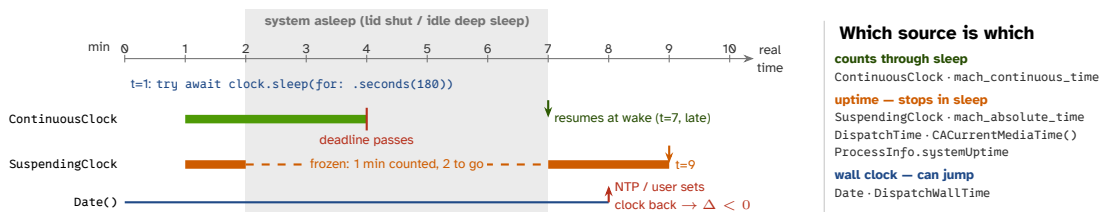


In one line: Clock (Swift 5.7, iOS 16) abstracts a time source: `now`, `minimumResolution`, `sleep(until:tolerance:)`. **Continuous-Clock** keeps counting while the system sleeps, **SuspendingClock** stops; both are **monotonic**, unlike `Date`, which is wall-clock and can jump. Measure elapsed time with a clock, store absolute moments as `Date`, and **inject** the clock so tests never really wait.



How it works

- protocol `Clock<Duration>`: `Sendable` — associated type `Instant`: `InstantProtocol`; requires `now`, `minimumResolution`, `sleep(until:tolerance:)` `async` throws. Extensions give `sleep(for:)` and `measure {}` (`sync` + `async`).
- `InstantProtocol`: `Comparable`, `Hashable`; `advanced(by:)`, `duration(to:)`; `instant + duration`, `instant - instant` → `Duration`. Each clock has its own `Instant` type — mixing clocks does not compile.
- `Duration`: clock-independent signed span, 128-bit (components: `Int64` seconds + attoseconds). `.seconds(1.5)`, `.milliseconds(250)`, `.microseconds`, `.nanoseconds`; `+` `-` `*` `/`. Display: `.formatted(.time(pattern: .hourMinuteSecond))` or `.units(allowed:)` — no hand-rolled modulo.
- `Task.sleep(for:tolerance:clock:)` — clock defaults to `.continuous`. **Suspends** the task (frees the thread); throws `CancellationError` early on cancel. `tolerance` lets the OS coalesce wake-ups (energy). `Task.sleep(nanoseconds:)` = legacy raw integer, no clock.
- `Date` = seconds since 2001-01-01 (reference date), an absolute moment: right for timestamps, calendars, persistence; wrong for elapsed time. Time zones and DST change its *display*, not its value.
- An `Instant` means nothing after a reboot — never persist it.

Example — inject, measure, test

```
struct Poller<C: Clock<Duration>> {
    let clock: C // injected
    func run(_ tick: () async -> Void) async throws {
        while true {
            await tick() // sleep throws on cancel
            try await clock.sleep(for: .seconds(30))
        }
    }
}

let d = ContinuousClock().measure { parse(data) } // Duration
print(d.formatted(.units(allowed: [.milliseconds])))
// test (Point-Free swift-clocks):
let clock = TestClock()
let poll = Poller(clock: clock)
let task = Task { try await poll.run { await hits.inc() } }
await clock.advance(by: .seconds(60)) // ticks 0/30/60 at once
task.cancel()
```

Test clocks (Point-Free swift-clocks)

`TestClock`: virtual `now`, sleepers resume only on `await advance(by:)` / `run()` → assert “nothing fired before 30 s”. `ImmediateClock`: every sleep returns at once (previews, happy-path tests). `UnimplementedClock`: fails the test if touched. Production takes some/any `Clock<Duration>`; a hard-coded `Task.sleep(for:)` cannot be redirected.

Timers

API	Mechanism · trap
<code>Timer</code>	run-loop source; <code>scheduledTimer</code> adds it in <code>.default</code> mode → stops while scrolling (<code>.tracking</code>); fix <code>RunLoop.main.add(t, forMode: .common)</code> . Needs a <i>running</i> run loop (a GCD thread has none → never fires). Retains its target until <code>invalidate()</code> .
<code>DispatchSource</code> <code>TimerSource</code>	<code>schedule(deadline:repeating:leeway:)</code> + <code>setEventHandler + resume()</code> ; no run loop. Keep a strong ref; never release it while suspended (crash).
<code>AsyncTimer</code> <code>Sequence</code>	swift-async-algorithms: for <code>await _ in .repeating(every: .seconds(1))</code> ; ends with the task.
<code>CADisplayLink</code>	fires per screen refresh — animation, not scheduling.

Interview traps

- ! `Date().timeIntervalSince(start)` for a timeout/benchmark: NTP or the user moves the wall clock → negative or huge.
- ! `SuspendingClock` stops when the **system** sleeps — not when your app is backgrounded or suspended.
- ! A test that waits 30 real seconds: the code hard-codes `Task.sleep` / `ContinuousClock()` instead of taking a clock.
- ! `Thread.sleep` / `usleep` in `async` code blocks a cooperative-pool thread.
- ! `TestClock` is from `swift-clocks`, not `swift-async-algorithms`.

Remember

Date = when, Clock = how long. `Continuous` counts the night, `Suspending` sleeps with the machine; inject the clock, advance it in tests.

Likely questions

- Continuous vs Suspending? — counts through system sleep vs pauses in it.
- Why not `Date` for elapsed time? — wall clock jumps; use a monotonic clock.
- Timer stops while scrolling? — `.default` mode; add it to `.common`.
- Test a 30 s retry fast? — inject `Clock`; `TestClock.advance(by:)`.