Why a Browser Timer Needs Anchor-Based Math, Not a Running Counter

> A timer that just counts down from a starting number drifts or stalls the moment its browser tab is backgrounded, while one anchored to a real timestamp does not.

A naive countdown or stopwatch implementation that decrements or increments an in-memory counter once per tick will drift away from real elapsed time whenever the browser throttles or pauses that tick, which happens routinely when a tab is backgrounded, the device sleeps, or the system is under load. A timer anchored to a real timestamp instead recomputes the elapsed or remaining time from the current clock on every redraw, so it self-corrects to the true elapsed time regardless of how irregularly the redraws themselves actually fire.



A tick-counter timer loses whatever ticks a backgrounded tab skips; an anchor-based timer recomputes from a real timestamp and stays correct regardless.

What actually happens when a tab is backgrounded

Browsers deliberately throttle or suspend timing callbacks in a backgrounded tab to save power and CPU, which is expected and correct browser behavior, not a bug to work around by fighting the browser. A `requestAnimationFrame` callback can stop firing entirely while a tab is hidden, while a `setInterval` callback typically keeps firing but at a reduced, throttled rate rather than its nominal interval. Either way, the number of ticks that actually fired is not a reliable proxy for how much real time passed.

Why anchoring to a timestamp fixes this without fighting the throttle

Recording the real start time (or target end time) once, then computing elapsed or remaining time as a fresh subtraction against the current time on every redraw, makes the displayed value correct regardless of how many ticks were actually skipped in between redraws. The redraw scheduler only controls how often the display refreshes, not what value it computes, which decouples visual smoothness from timing accuracy entirely.

What WhenMath's timer engines do

All of WhenMath's active-time engines (Timer, Stopwatch, Interval Timer, Pomodoro / Focus Cycle Timer) compute their displayed value from an anchored `Date.now()` comparison on every redraw rather than from an incrementing counter, confirmed directly in live-browser testing. A bare `requestAnimationFrame` call was found to stop entirely on a backgrounded tab, which is exactly why the redraw scheduler uses `setInterval` (which keeps firing, throttled) while the underlying time math stays anchor-based regardless of which scheduler drives the redraw.

Key takeaway. Never trust a tick or frame count as a proxy for real elapsed time in a browser timer. Anchor to a real timestamp and recompute the elapsed or remaining value fresh on every redraw so backgrounded-tab throttling cannot cause drift.

Frequently asked

Does this mean setInterval is always the right choice over requestAnimationFrame?

For a timer that must keep counting while backgrounded, yes, since rAF can stop entirely when hidden while setInterval keeps firing (throttled). rAF remains the better choice for purely visual animation that should pause anyway when unseen.

Can a backgrounded tab still lose a countdown's final "time's up" moment?

No, as long as the completion check is also anchor-based (comparing the current timestamp against the target end time) rather than waiting for an exact tick to land on zero. The engine detects that the target has already passed on the next redraw, even a delayed one.

Does this affect the accuracy of a completed timer's recorded duration?

No, the final recorded duration is itself an anchor-based calculation (end timestamp minus start timestamp), which was never dependent on tick counting in the first place.

WhenMath is a browser-local calculation tool, not a legal, payroll, or scheduling authority. This guide explains a calculation convention; it is educational, not a determination that any specific contract, employer, or jurisdiction follows it.