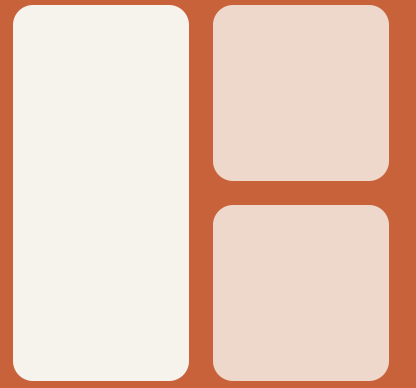


Tiler

one desktop at a time.

How a 4,000-line Swift menu bar app tiles macOS windows with Accessibility, SkyLight and Carbon – and touches nothing it was not asked to.



Five parts, from the problem to the release.

- 01 **Why and what.** The scope, the audience, and what was left out of HyprMac.
- 02 **Architecture.** Components, threads, and how a desktop is identified.
- 03 **Events and layout.** Reconcile, frame writes, drags, the BSP tree.
- 04 **Input, focus and UI.** macOS 14 focus, Carbon hotkeys, move/resize mode, focus follows mouse, desktop moves, config, Settings.
- 05 **Quality and shipping.** Performance, tests, bugs, limits, release pipeline.

AT A GLANCE

Language	Swift 5.10
Build	SwiftPM
Runs on	macOS 14+
Binary	universal
Dependencies	none

Tile one desktop. Touch nothing else.

Tiler does HyprMac's core job – Hyprland-style BSP “dwindle” tiling with keyboard focus, swap and resize – but only on the desktops the user turns on, one at a time.

A desktop is one Space on one display. Space ids change at login; the Space uuid does not, so that is the key that persists.

```
// Desktop.swift
struct DesktopKey: Codable, Hashable {
    let displayUUID: String
    let spaceUUID: String
}
```

The hard rules

- Never touch a window outside a tiled desktop's display **and** Space – except one window the user explicitly acts on (move/resize mode, move to another desktop, ⌘-drag).
- Never call Space-mutation APIs; moving a window to another desktop drives the system's own Mission Control shortcut.
- No polling, never a keyboard tap: react to AX and NSWorkspace events. Focus follows mouse uses a listen-only mouse tap; ⌘-drag one active tap for left-button events only, on its own thread.
- Every AX call for an app runs on that app's own thread.
- Act on moved/resized notifications only while the mouse is down.
- Every config key is required; no silent defaults.

Who Tiler is for – and who it is not for.

01

Mac power users

Several displays, several Spaces, a keyboard-first habit. They want tiling on the coding desktop and free windows on the mail desktop.

02

Engineers

People who want to read, audit or extend a window manager. 4,000 lines, no dependencies, one design document, 30 unit tests.

03

Not the target

Anyone who wants virtual workspaces, per-app rules engines or a scripting language. yabai and AeroSpace cover those.

This deck is for engineers. A separate deck walks users through installing and using Tiler.

What was left out of HyprMac, and why.

HYPRMAC FEATURE	WHY TILER DROPS IT	TILER INSTEAD
Ten virtual workspaces, windows hidden in a screen corner	Conflicts with “touch only this desktop”. The hide-corner, parking and ghost-window logic was the largest bug source.	Native Spaces; each desktop keeps its own tree.
Caps Lock “Hypr” key via an active CGEventTap	An active tap delays every keystroke system-wide when its thread stalls.	Carbon RegisterEventHotKey : no permission, only its own keys.
1 Hz polling and frame verification loops	Constant work and fights with apps that refuse a size.	AX notifications and a ~1 ms reconcile.
Borders, dimming, scratchpad, overview	Outside the core job; each costs WindowServer time.	Not built.

Small enough to read in an afternoon.

4,000

lines of Swift, Settings, the shortcuts overlay and the move/resize mode included – against about **35,000** in HyprMac.

Third-party dependencies	0
CPU while idle	0 %
One reconcile pass	≈1 ms
Unit tests (XCTest)	30
Private symbols used	9

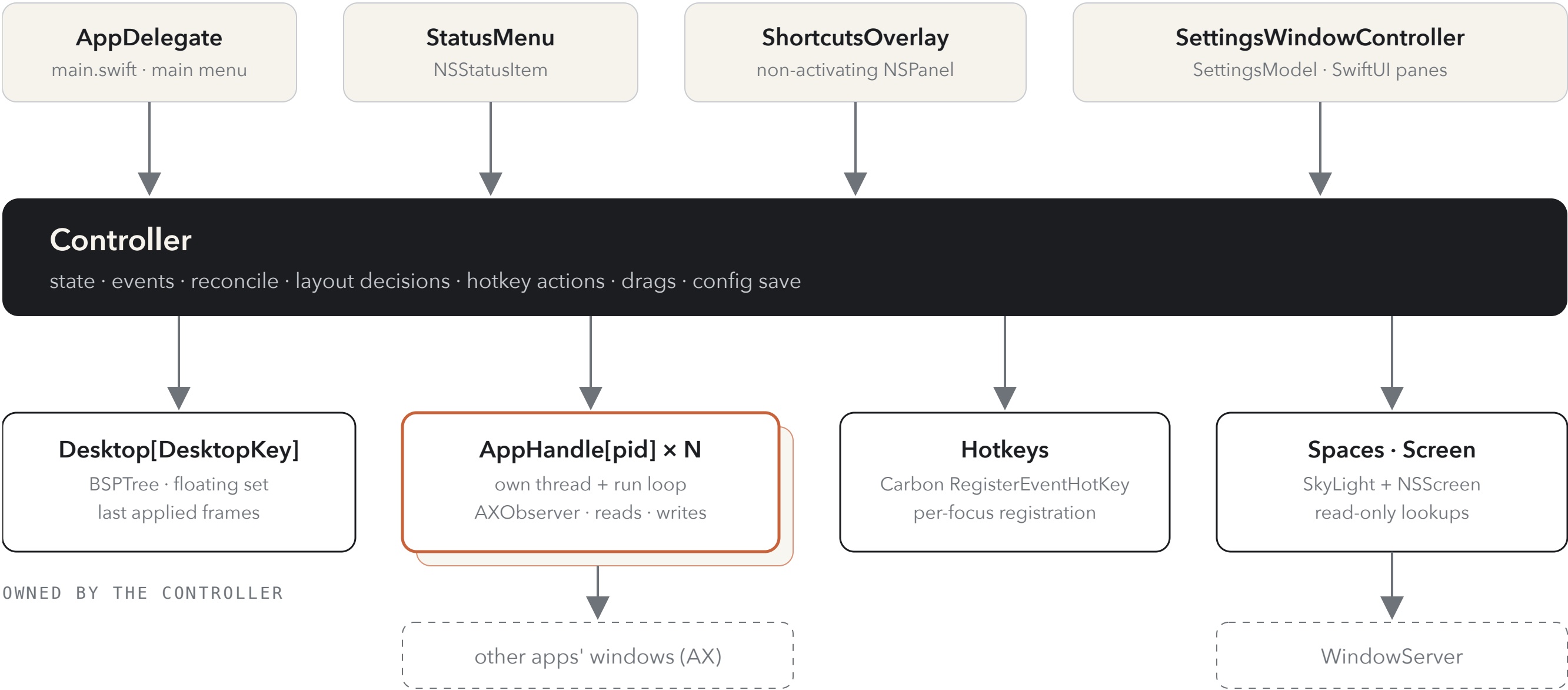
02

Architecture.

One main-thread controller decides; one thread per app talks to that app. A pure library holds everything that can be unit-tested.

The whole app on one page.

MAIN THREAD · UI



TilerCore
pure Swift · no AppKit · no AX

BSPTree
dwindle layout, edits

Geometry
neighbours, hit tests

Config
strict parse, json()

KeyCombo · Action
symbols, titles, groups

30 XCTest tests

The main thread decides. App threads talk to apps.

Main thread

Controller, menu, Settings, overlay, Carbon hotkey events, NSWorkspace notifications. Owns all state; never makes an AX call.

One `RunLoopThread` per app

AXObserver source, window reads and writes. A hung app stalls only its own thread – and at most 1 s per call.

Nothing tiled, nothing running

Handles exist only while a desktop is tiled. Turning the last one off took the process from 20 threads to 8.

```
// AppHandle.swift – work hops to the app's thread
func perform(_ block: @escaping () -> Void) {
    guard let runLoop else { return }
    CFRunLoopPerformBlock(runLoop,
        CFRunLoopMode.defaultMode.rawValue, block)
    CFRunLoopWakeUp(runLoop)
}

// results come back on main
func resolve(_ ids: Set<CGWindowID>,
    completion: @escaping (Facts) -> Void) {
    thread.perform { [self] in
        refreshElements()
        let facts = ... isTileable(el) ...
        DispatchQueue.main.async { completion(facts) }
    }
}
```

Which Space is showing, and which windows are on it.

SKYLIGHT CALL	USED FOR
CGSCopyManagedDisplaySpaces	Every Space with id, uuid and type; desktop numbers. Run at launch, on toggle and when Spaces change (~1-15 ms).
SLSManagedDisplayGetCurrentSpace	The Space a display shows now. Tens of μ s, so never cached.
SLSCopySpacesForWindows	The Spaces a window belongs to (mask <code>0x7</code>).
SLSManagedDisplayIsAnimating	Skip reconcile mid-switch, when the list mixes two desktops.

```
// Spaces.swift
static func current(onDisplay uuid: String)
    -> SpaceID {
    let id = SLSManagedDisplayGetCurrentSpace(
        connection, uuid as CFString)
    // "Displays have separate Spaces" off:
    // every display shows the "Main" Space
    return id != 0 ? id
        : SLSManagedDisplayGetCurrentSpace(
            connection, "Main" as CFString)
}
```

A desktop is **visible** when its display's current Space equals its stored Space id. Ids are re-resolved from the uuid at launch and on display changes; a desktop deleted in Mission Control is forgotten.

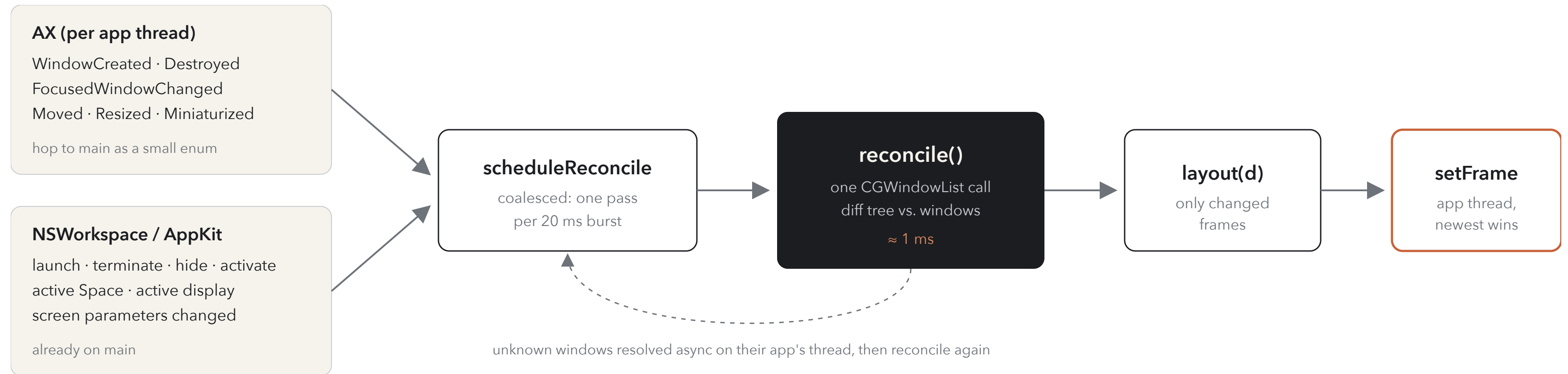
`desktops.json` keeps floating windows as id + owner pid + frame (read 0.3 s after a move, saved $\leq 1/s$). A restore keeps an id only if that pid still owns it, then puts the frame back once the desktop is on screen. Re-floating reuses the last frame if $\geq 50\%$ is on the display.

03

Events and layout.

No polling: every change starts from a notification, is folded into one reconcile pass, and ends in the fewest possible frame writes.

From a notification to a written frame.



Created windows trigger extra passes at +0.15 s and +0.5 s (some apps show the window late); a window moved by a script gets one at +0.4 s (its Space updates late); an untileable window is re-checked once after 0.6 s.

Six steps, about a millisecond.

- 1 Skip unless a tiled desktop is visible; wait while its display is animating a Space switch.
- 2 One `CGWindowListCopyWindowInfo(onScreenOnly)` ; keep layer-0 windows of other apps with alpha > 0.
- 3 Per visible desktop: centre on its display **and** its Space in `SLSCopySpacesForWindows` .
- 4 Re-check the Space did not change during the read; if it did, the next event redoes it.
- 5 Remove vanished and always-float windows; insert new tileable ones next to the last focused tile.
- 6 Lay out only if something changed, and write only frames that differ from the last one written.

When is a window tileable?

- role `AXWindow` , subrole `AXStandardWindow`
- not minimised, not full screen
- an **enabled** green button (dialogs and settings panes have none)
- `AXSize` is settable
- app not in `floatingApps`

Everything else floats and is never moved.

Write the newest frame only, in an order apps accept.

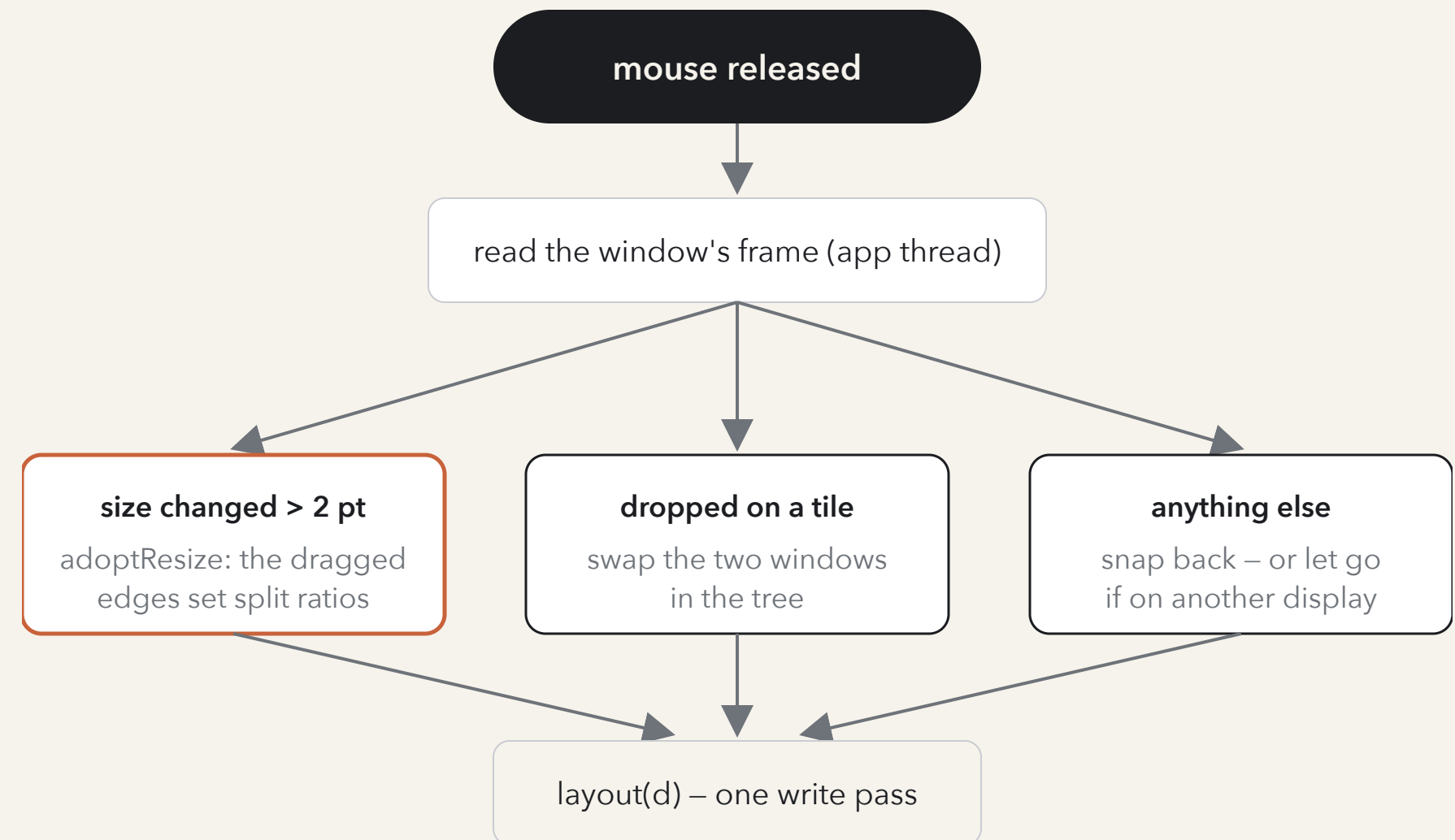
- **Coalesced.** `setFrame` stores the newest frame per window and schedules one flush. A slow app (iTerm2 took up to 700 ms) never replays stale layouts.
- **Size** → **position** → **size**. The first resize cannot clamp the move at a screen edge; the second fixes a resize clamped by the old position.
- **AXEnhancedUserInterface** is switched off for the writes, or the app animates every resize.
- **Bounded.** `AXUIElementSetMessagingTimeout` = 1 s, system-wide and per element.

```
// AppHandle.swift – on the app's thread
private func flushFrames() {
    lock.lock()
    let frames = pendingFrames
    pendingFrames.removeAll()
    flushScheduled = false
    lock.unlock()
    let enhanced = bool(app,
        "AXEnhancedUserInterface") == true
    if enhanced { set(app, ..., false) }
    defer { if enhanced { set(app, ..., true) } }
    for (id, frame) in frames {
        guard let el = elements[id] else { continue }
        setSize(el, frame.size)
        setPosition(el, frame.origin)
        setSize(el, frame.size)
    }
}
```

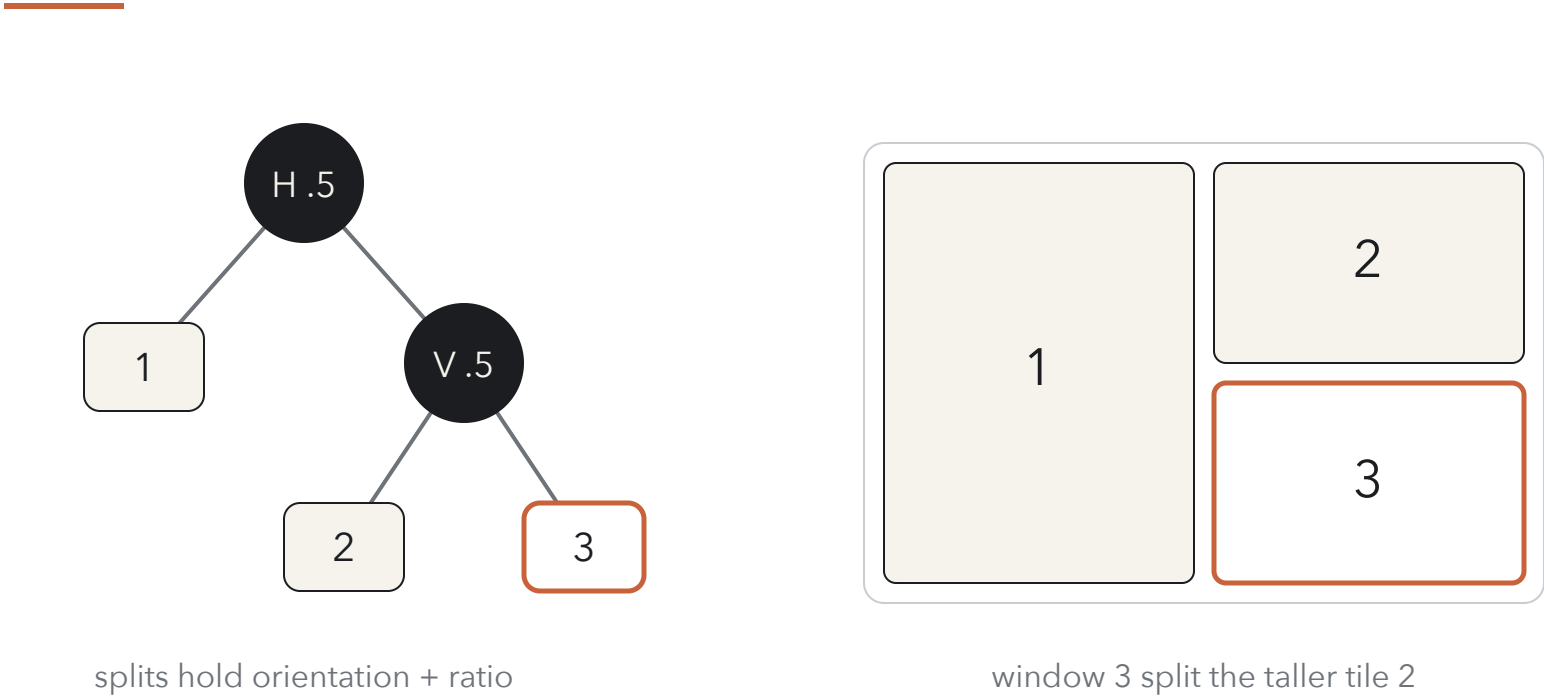
React to window moves only while the mouse button is down.

Tiler's own writes produce moved/resized notifications too. So do apps that refuse a size or snap themselves. Acting on those would start a fight.

The rule: with no mouse button down, a move of a tiled window is ignored. With the button down it is a user drag, handled once on release.



A binary space partition, dwindle style.



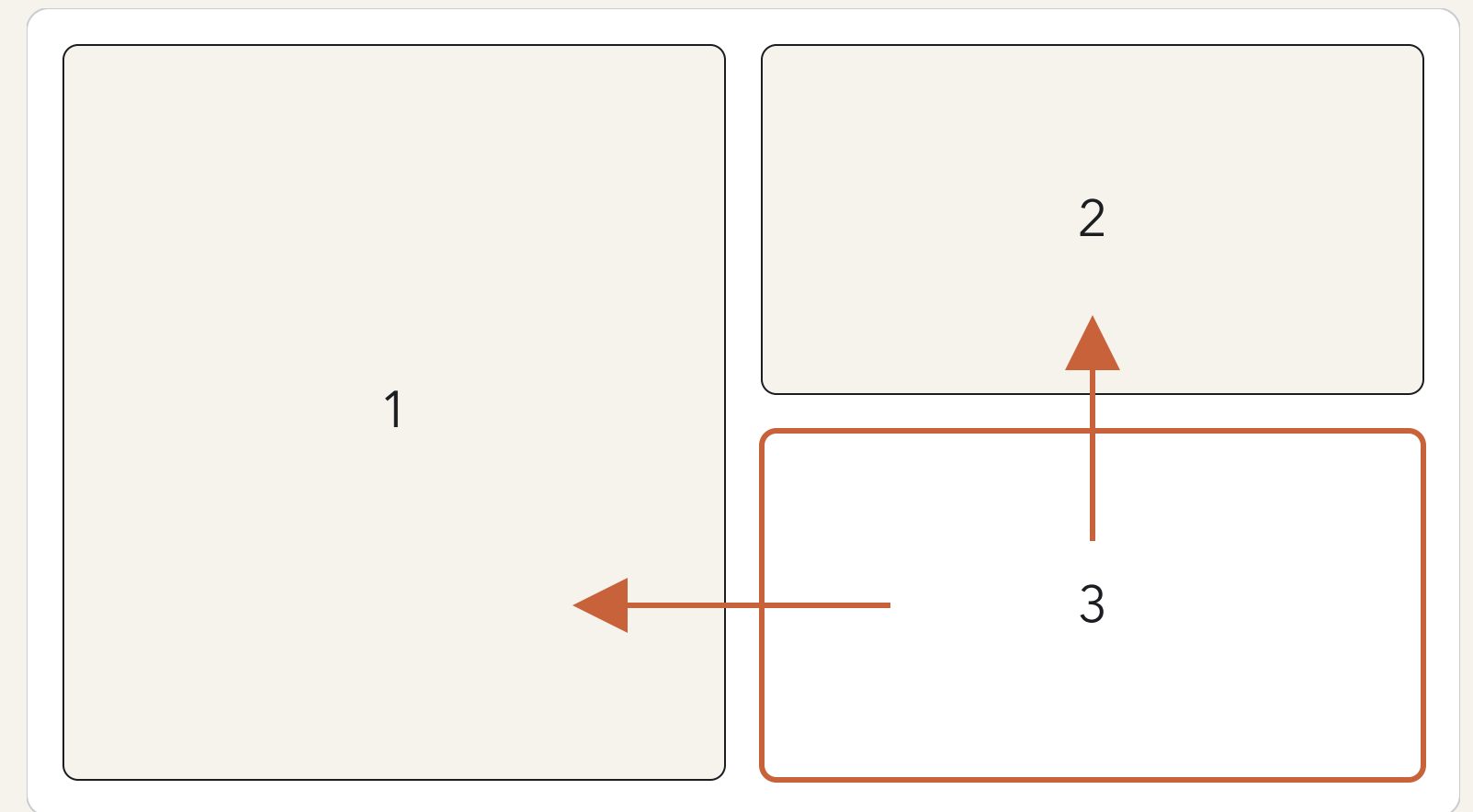
A new window splits the last focused tile along its **longer** side and takes the second half. Ratios are clamped to 0.1-0.9; frames are rounded to whole points so neighbours never overlap.

OPERATION	WHAT IT DOES
insert / remove	Split a leaf; on removal the sibling takes the parent's place.
swap	Exchange two leaves' windows.
moveDivider	Prefer the divider on the arrow's side, else the opposite one.
adoptResize	Turn dragged edges into ratios of the splits behind them.
toggleOrientation	Side by side ↔ stacked for one split.
mirror · equalize	Flip left ↔ right; set every ratio to 0.5.

Which window is "to the left"?

Focus and swap move by geometry, not by tree position, so the keys do what the eye expects. `Geometry.neighbor` ranks every frame on the arrow's side (4 pt tolerance):

- 1 overlaps the source across the movement axis
- 2 then the smaller gap between the facing edges
- 3 then the smaller offset between centres
- 4 then the lower window id, so ties are stable



from 3: left → 1, up → 2, down → none (unit-tested)

04

Input, focus and UI.

macOS 14 changed how a background app may focus windows. Carbon hotkeys, a strict config file and native Settings complete the picture.

Focusing another app's window from the background.

Since macOS 14, `NSRunningApplication.activate()` from an inactive app is ignored ("cooperative activation"). The first focus hotkeys did nothing.

Tiler uses what yabai and AeroSpace use: bring the process forward with the target window, post a synthetic make-key-window record, then `AXRaise`.

Which display is "active"?

The display of the focused window, else the one under the mouse.

`SLSCopyActiveMenuBarDisplayIdentifier` reported the wrong display on macOS 27 and swallowed keys.

```
// PrivateAPI.swift
func focusWindow(pid: pid_t, wid: CGWindowID) {
    var psn = ProcessSerialNumber()
    guard GetProcessForPID(pid, &psn) == noErr
        else { return }
    _ = _SLPSSetFrontProcessWithOptions(
        &psn, wid, 0x200) // user generated
    // make-key-window record (layout: yabai)
    var bytes = [UInt8](repeating: 0, count: 0xf8)
    bytes[0x04] = 0xf8; bytes[0x3a] = 0x10
    // wid at 0x3c, 0xff over 0x20..<0x30
    bytes[0x08] = 0x01
    _ = SLPSPostEventRecordTo(&psn, &bytes)
    bytes[0x08] = 0x02
    _ = SLPSPostEventRecordTo(&psn, &bytes)
}
```

Carbon hotkeys, registered only where they apply.

	CGEVENTTAP	REGISTEREVENTHOTKEY
Permission	Input Monitoring / Accessibility	none
Sees	every keystroke	only its own combos
If its thread stalls	all typing lags	nothing else is affected
Conflicts	can swallow anything	first owner wins

- **Per focus.** `toggleDesktop` , `showHotkeys` , `moveResize` and the two desktop moves are always registered; the tiling keys only while the active display shows a tiled desktop. Elsewhere apps keep those keys.
- **Suspended while recording.** Settings sets `hotkeysSuspended` , so a new combination reaches the recorder.
- **Keys borrowed.** `borrow` / `release` : Esc while the overlay is up; arrows, C, M, Return and Esc while the move/resize mode is on (ids from `0x1000` , pressed and released events), given back when it ends.
- **Signature** `"TLRR"` ; the id indexes `Action.allCases` .

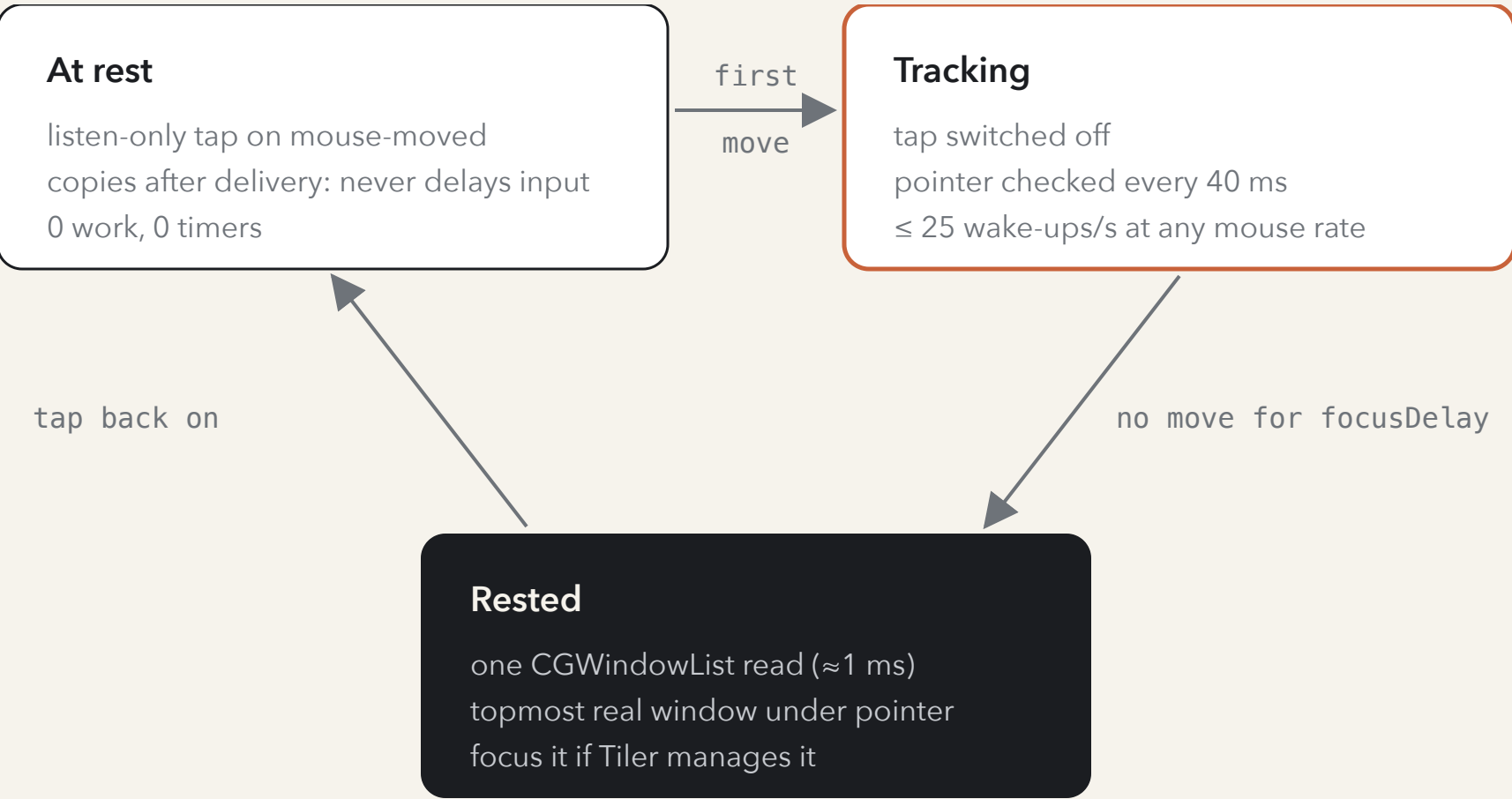
Keyboard moves for untiled windows, borrowed keys.

- **Scope.** `⌘M` acts on the focused window only, and only when no tiled desktop holds it (`isTiled`). A tiled window gets a note instead.
- **Keys exist only in the mode.** Arrows, `⇧/⌥` arrows, C, M, Return, Esc are borrowed Carbon hotkeys; pressed/released events drive a repeat timer (0.3 s, then 45 ms). App switch, Space change or a click ends it.
- **Own AX thread.** `WindowMover` reads the focused window and writes frames on a `RunLoopThread` ; only the newest frame is written (size → position → size), then the frame the app accepted is read back.
- **Pure arithmetic.** `WindowNudge` in `TilerCore`: 40 pt or 8 pt steps, top-left anchored resize, min 160×100, clamped to the usable area minus the outer gap.

```
// WindowNudge.swift – unit-tested
public enum Operation {
    case move(Direction)    // arrows
    case resize(Direction)  // ↑ arrows
    case center              // C
    case maximize            // M
}

static func apply(_ op: Operation,
                 to frame: CGRect,
                 in area: CGRect,
                 step: CGFloat) -> CGRect
```

Focus follows the pointer, at almost no cost.

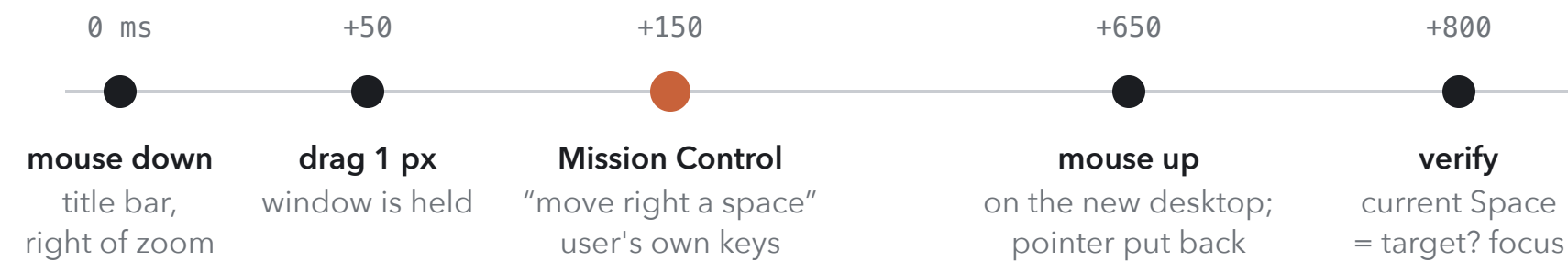


Idle	0 CPU
10 s of motion	≈0.015 s
Same, NSEvent monitor	≈0.13 s

- The cursor and the Dock's full-screen window are skipped in the hit test.
- A menu, popover or panel on top blocks; any open menu or the app switcher leaves focus alone.
- A quick pass never focuses: only a rest does.

Tap exists only while the setting is on and a desktop is tiled.

Moving a window between Spaces, the way a person would.



Synthetic mouse events carry no modifier flags: the shortcut's $\text{^}\text{⌘}$ are still held.

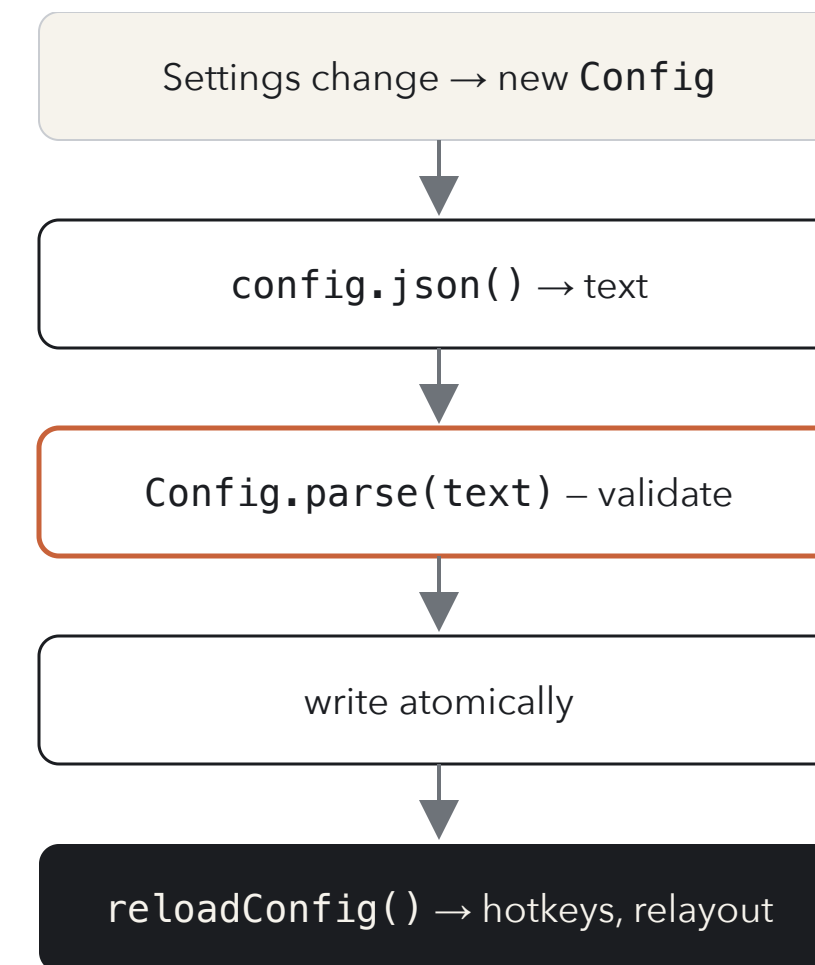
Since macOS 14.5 the private move-window-to-Space calls fail; Tiler never calls them.

- $\text{^}\text{⌘}, / \text{^}\text{⌘}$. work on every desktop. `DesktopMover` reads the focused window on its own AX thread: frame, zoom-button position, subrole.
- **No phantom drag.** `beginDesktopMove` sets `movingWindow`: its moved events are ignored and reconcile waits until it lands.
- **State hand-over.** The window leaves the old desktop's tree (that desktop lays out when next shown); a tiled target tiles it; a floating window stays floating there, with its frame.
- **Refusals, with a note:** no desktop on that side; a full-screen Space next; a dialog or panel; the Mission Control shortcut turned off.

```
// com.apple.symbolichotkeys: 79 = left, 81 = right
"81": { enabled = 1; parameters = (65535, 124, 8650752) }
// parameters: (unused, key code = →, modifiers = ^ + fn)
```

One file, strictly parsed, written back by Settings.

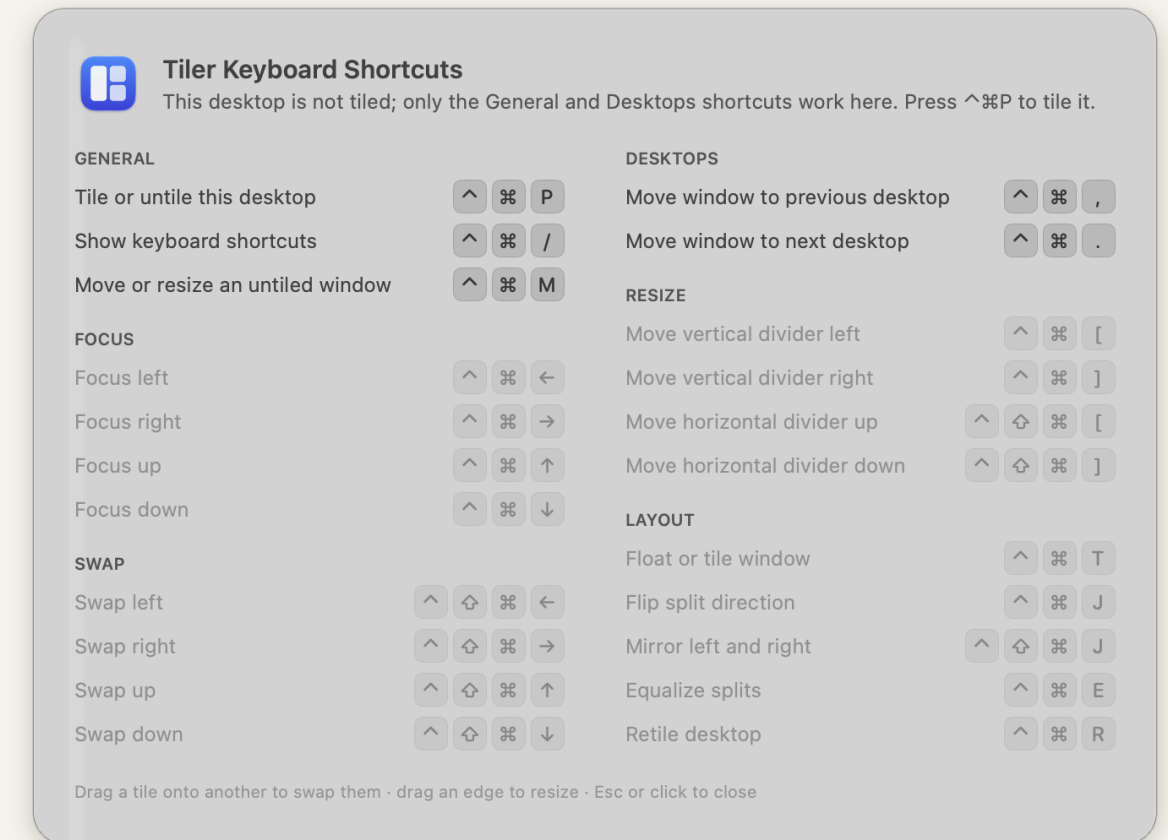
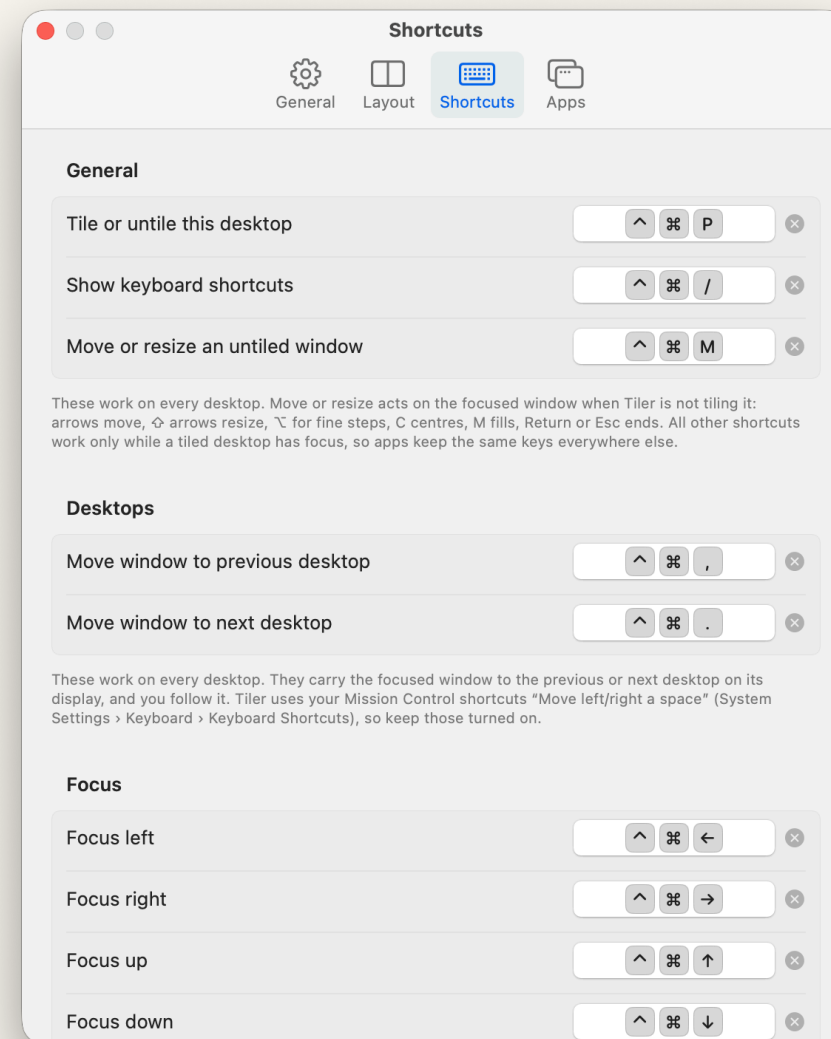
- `~/Library/Application Support/Tiler/config.json`, written from a template on first launch.
- **Every key required.** Missing, unknown, out-of-range or duplicate keys are errors, never defaults. The last good config stays in force; the menu and Settings show the error.
- `Config.json()` writes every key in a fixed order. 0.4 added `focusFollowsMouse` (bool) and `focusDelay` (0.05–1 s).
- While the file has an error, Settings is read-only: it never overwrites a file being fixed by hand.
- **Upgrades.** A new setting is a missing key, so an error. Settings offers **Add Missing Settings**: `Config.addingMissingSettings` adds only the missing keys from the template, by explicit click.



Controller.save – the file stays the single source of truth

AppKit where macOS expects it, SwiftUI inside.

- **Settings:** `NSTabViewController` in toolbar style, one `NSHostingController` per pane with a grouped `Form`.
- **Overlay:** borderless non-activating `NSPanel`, `.hudWindow` material. It never becomes key, so the app underneath keeps the keyboard.
- **Menu bar glyph:** a template image drawn in code, filled when the desktop is tiled.
- **App icon:** rendered by `scripts/make-icon.swift`; no image assets in the repo.



Nine private symbols, none of them for moving windows.

SYMBOL	PURPOSE
SLSMaInConnectionID	window-server connection
CGSCopyManagedDisplaySpaces · SLSManagedDisplayGetCurrentSpace · SLSManagedDisplayIsAnimating · SLSCopySpacesForWindows	Spaces, read-only
_AXUIElementGetWindow	AX element → window id
GetProcessForPID · _SLPSSetFrontProcessWithOptions · SLSPPostEventRecordTo	focus

Risk, managed

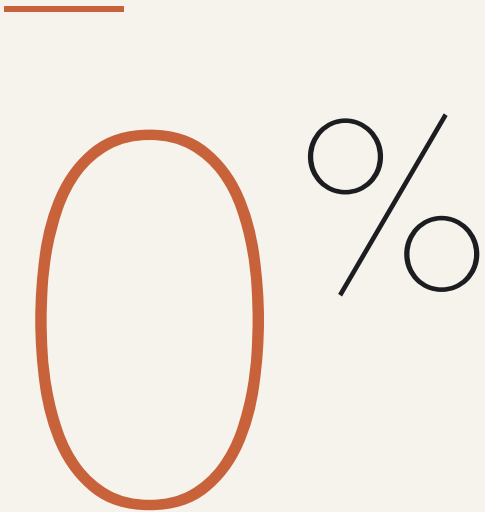
- All work with SIP on; checked on macOS 27.0.1.
- The same calls yabai, AeroSpace and HyprMac rely on.
- Frames are written only through the public AX API.
- No moving windows between Spaces – that path broke on macOS 26/27.

05

Quality and shipping.

What it costs at runtime, how it was tested, what broke along the way, what is still open – and how a release reaches Homebrew.

Idle means idle.



CPU while nothing changes. No timers at rest, no polling: Tiler wakes only when the window server or an app tells it something happened.

Reconcile pass	≈1 ms
Coalescing window	20 ms
Memory (debug build)	≈52 MB
Threads, last desktop off	20 → 8
Worst AX call	1 s cap
Mouse moving, focus follows on	≈0.15 %

Unit tests for the logic, live checks for the rest.

30 XCTest tests on TilerCore

BSPTree · 11	dwindle split, gaps, remove, swap, divider moves, clamp, mirror, equalize, adoptResize
Config · 14	template parses, missing / unknown / duplicate keys, "none", json() round trip, key events, conflicts, adding missing settings, booleans, desktop-move defaults
WindowNudge · 3	move stays inside, resize anchors top-left and clamps, centre and fill
Geometry · 2	neighbour ranking, hit testing

Checked live on macOS 27

- tile on activation; persistence across restarts
- new and closed windows, swap, resize, float, equalize
- focus left/right; drag-to-swap; gap reload
- hotkeys released on untiled desktops; other display never touched
- overlay Esc and toggle with Finder in front; Settings recorder and conflict check
- move/resize: tiled window refused; arrows, ⬆, ⬇, hold-repeat, fill; keys released on exit
- focus follows mouse: rest focuses, a fast pass doesn't, an open menu blocks; floating survives a restart; Add Missing Settings
- desktop moves: untiled → next and back; into a tiled desktop (tiled) and out; full-screen neighbour refused

What live testing caught.

SYMPTOM	CAUSE	FIX
Focus hotkeys did nothing	macOS 14 ignores <code>activate()</code> from a background app	SLPS front-process + key-window record + <code>AXRaise</code>
Keys swallowed on an untiled desktop	Active-menu-bar-display call answered wrong on macOS 27	Active display = focused window's, else the mouse's
Overlay ignored Esc	A non-activating panel gets no keyboard while another app is in front	Never take focus; borrow Esc as a hotkey while visible
Settings opened behind other apps	Same cooperative-activation rule	<code>focusWindow</code> on our own window
Default keys moved windows away	Rectangle and Hammerspoon own <code>^v</code> and Hyper combos	Defaults moved to the <code>^⌘</code> family

Still open.

Minimum sizes

Apps that refuse a width overlap their neighbour (Outlook stops at 770 pt). A fix must read back sizes without oscillating.

Shared shortcuts

Carbon accepts a combination another tool already owns, and that tool wins. Settings can only see clashes inside Tiler.

Other window tools

Rectangle, Hammerspoon or HyprMac on the same desktop act on the same windows. Use one manager per desktop.

Drag-resize

Unit-tested through `adoptResize`; drag-swap was checked live, drag-resize not yet.

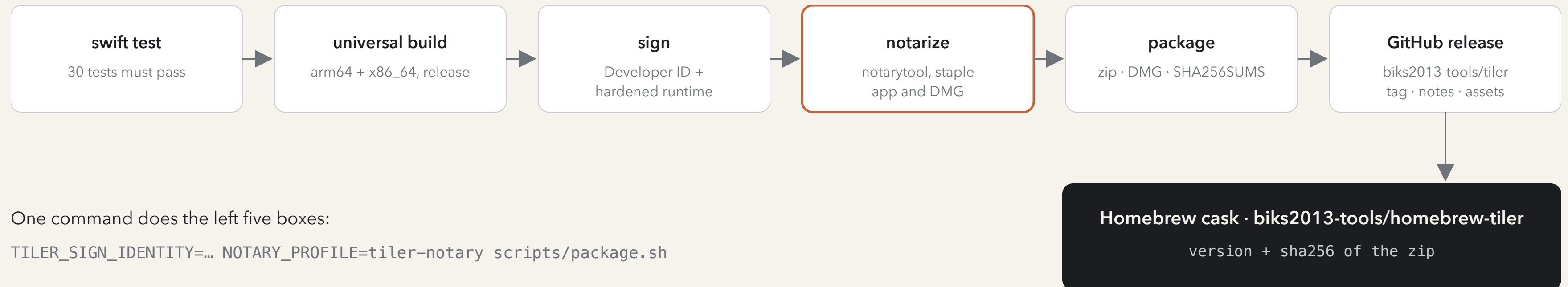
Tahoe corners

Large, varying window corner radii on macOS 26+ make equal gaps look uneven. Cosmetic.

Shared Spaces

“Displays have separate Spaces” off is handled (the “Main” Space) but not checked live.

From **swift test** to **brew install**.



A stable Developer ID signature keeps the Accessibility grant valid across updates. Gatekeeper reports “Notarized Developer ID” for the app and the disk image. Steps and checks: [docs/design/release-runbook.md](#).

Install it, then open the source.

```
# install
brew tap biks2013-tools/tiler
brew install --cask tiler

# build from source
git clone https://github.com/biks2013-tools/tiler
cd tiler && swift test && swift build

# drive a running instance
scripts/tiler-ctl.sh toggle active
scripts/tiler-ctl.sh retile | moveresize | dump
```

Source	github.com/biks2013-tools/tiler
Homebrew tap	biks2013-tools/homebrew-tiler
Decks	biks2013-tools.github.io/tiler
Design	docs/design/project-design.md
Logs	com.giorgosmarinos.tiler

First launch asks for Accessibility. That is the only permission Tiler needs.

THANK YOU

Do one job,
on one desktop, well.

Questions, issues and pull requests: github.com/biks2013-tools/tiler