What Swift Concurrency Repairs Actually Guarantee

Two local regression studies and a targeted review of 29 production defect families

Ehsan Azish | Independent developer
Technical report, version 1.0 | 6 November 2026

TECHNICAL REPORT - NOT PEER REVIEWED

Abstract

Compiler concurrency checking, runtime race detection and application lifecycle correctness answer different questions. This report examines that distinction through two completed local regression studies and a targeted source review of 29 defect families in public Swift projects. A Kingfisher cancellation component lost cancellation before task attachment in 100/100 selected trials on the faulty revision and 0/100 on the repair, including an explicit Swift 5 complete-checking build. A Nuke pipeline lost the terminal event for an immediate memory-cache subscription in 20/20 trials on the faulty revision and 0/20 on the repair under Swift 6. These are bounded fixture outcomes, not rates in deployed applications or proofs over all schedules.

The practical lesson is to review separate obligations: who owns completion, what state survives before registration, which waiters receive the outcome, when work becomes obsolete, and how cleanup and progress occur. Related Kingfisher and Vapor patches address different obligations within a lifecycle. That observation does not establish that earlier repairs introduced later defects. The sample is deliberately targeted, its labels are provisional assistant coding, and no independent human agreement score is available.

Read for	Pages
Scope, sampling and claims	2
Kingfisher and Nuke validation	3-6
Design discussion, checking and review guidance	7-9
Limits and contrary evidence	10
Complete source-case ledger	11-12
Revision and artifact provenance	13
References	14

Key result: matched repairs passed the selected fixtures. The report does not claim ecosystem prevalence, a novel concurrency mechanism, universal repair correctness, or a measured 29-case tool-detection comparison.

1. Evidence and sample boundaries

The unit of analysis is a deduplicated defect family, not a PR, a symptom report or every changed line. The current ledger contains 29 qualifying production families across 13 repositories. One NIO candidate is disputed and excluded from that denominator; one runtime family is tracked separately. Compound lifecycle repairs remain conservatively grouped. An accepted upstream patch is source evidence, not proof that every attributed root or outcome is correct.

The initial retrieval was purposive: fixed concurrency-related records from selected applications and libraries, followed by server-side configuration-first retrieval and bounded continuation/cancellation enrichment. The working record contains 44 reviewed PR records. Searches, caps, linked discussions and enrichment were not a random sample of all Swift code. Cancellation and continuation terms actively influenced selection. The counts below describe this sample only.

Recorded checking context	Families	Interpretation
A: unconditional source-declared Swift 6	10	Target/manifest evidence; not whole-program checking proof
A if compiler >= 6.2; otherwise C	1	Keep compiler condition explicit
B via Xcode; C via SwiftPM	3	Consumer/build route matters
C: other Swift 5 settings	6	Not a checked-path survivor claim
C: unknown settings	9	Unknown is retained, not inferred from dates

Two bridge definitions, two different questions

A legacy bridge causally connects callback, Combine, GCD or platform-event code to native Swift Tasks, continuations, cancellation handlers or async sequences. The broader handoff definition also includes native actor-to-stream/continuation interactions. Legacy-only callback races are not automatically native-concurrency bridges. Imports, MainActor annotations and the presence of an async facade are insufficient causal evidence.

Definition	All 29: yes / no / uncertain	A only: yes / no / uncertain
Legacy-to-native bridge	5 / 19 / 5	2 / 5 / 3
Broad native/legacy handoff	9 / 17 / 3	6 / 3 / 1

The broader A majority is exploratory and definition-sensitive. Two Nuke and two Vapor families supply four of its six positives. Different eras, domains and retrieval terms confound A-versus-other comparisons. No code-exposure denominator supports a prevalence or concentration estimate.

2. Kingfisher: cancellation before attachment

Kingfisher PR #2526 is a follow-up within family P07, not an additional family. The completed local test exercised the real internal CancellationDownloadTask and DownloadTask types, with a local Swift Task as the wrapped target. The fixture used the old and new production-facing actions through a small test adapter. Production source files were unchanged; a research test target was added. [K1]

The selected ordering was explicit: create the bridge without an attached task, perform its cancellation action, attach a task, then inspect the target task cancellation bit. The old action consulted the optional task too early and had no persistent memory of cancellation. The repaired component stores cancellation state and applies it when attachment occurs. The fixture is deliberately a component test; it does not run the full image manager, view integration, URLSession or an application.

Fixture condition	Faulty	Repaired
Cancellation before attachment	100/100 failures	0/100 failures
Cancellation after attachment control	100/100 pass	100/100 pass
No-cancellation control	100/100 pass	100/100 pass

The table applies to each of the two saved configurations: the selected SwiftPM Swift 5 build and Swift 5 with complete concurrency checking explicitly enabled. These repetitions exercise a constructed ordering; they are not independent estimates of a production probability. The wrapper forwards cancellation to a local Task even if that Task has already returned. Observing its cancellation bit establishes forwarding state, not interruption of active work.

What the checking result establishes

Production compile commands record Swift 5 language mode and the explicit complete-checking flag in the added context. That is a cohort-B validation context. It is not Swift 6 language mode, proof of the original reporter configuration, or an original PR #2398 reproduction. Build warnings exist in some logs; this report does not claim a warning-free project. The relevant observation is that compilation succeeded while the selected faulty component behavior remained observable.

Faulty: fcf02a9ab324e2a346aa5f82480d60e1c6a04448
Repair: 77173fa007500c701392d8376a5fa43a4aee9990

3. Kingfisher: what the repair changes

1	**Cancel before attachment** Old component sees no task; cancellation history is absent.
2	**Attach the task** A later assignment cannot recover a request that was forgotten.
3	**Retain cancellation state** The repair records cancellation and checks it when attaching.

State memory and mutual exclusion do different jobs

An actor can serialize a read of nil and a later write of a task without knowing that the read represented a cancellation request that must survive. The repair changes the represented state as well as the coordination mechanism. Its lock protects the cancellation flag and task reference. The chosen task cancellation call is made after releasing the lock. The component result supports that specific attachment obligation; it does not show that replacing every actor with a lock is desirable.

```
Illustrative state contract, not copied Swift code:
  cancel(): remember cancelled; capture current target
  attach(target): store target; observe remembered cancellation
  invoke target cancellation outside the protected state update
```

A production design must also specify task replacement, repeated cancellation, reentrant callbacks, terminal cleanup and ownership of retained references. The test here did not establish universal correctness for all those questions. Nor does cancellation forwarding prove immediate physical I/O termination. Cooperative tasks and external APIs can have their own cancellation behavior.

Completion claims are a separate obligation

The earlier Kingfisher #2398 patch introduced actor-backed once-only continuation handling and setup cancellation checks. The later #2526 attachment change addresses remembering and delivering cancellation. The later PR bundles other callback bookkeeping and integration changes. Source history and the component comparison do not prove that #2398 introduced the later failure, that it persisted unchanged through every revision, or that the latch alone explains all later improvements. [K0, K1]

Review implication: write both properties down. At-most-once continuation resumption is different from eventual cancellation delivery to a subsequently installed target. A patch can improve one without establishing the other. This is a familiar concurrency distinction made concrete by a Swift repair history, not a new algorithm or a measured incomplete-fix rate.

4. Nuke: an immediate subscriber misses the result

Nuke PR #916 changes ImageTask event subscription behavior. The completed local fixture used the production ImagePipeline, ImageTask and ImageCache, with a tiny NSImage already placed in memory. It created an image task and immediately iterated task.events, counting terminal events. The fixture ran on ImagePipelineActor. No image was fetched from the example URL; it served only as a cache key. [N1]

The internal registration hop can reach the actor after work has already completed. On the faulty parent, the stream can end without delivering its terminal event even though the task has a completed result. The repair retains and replays the terminal outcome when a subscription registers after completion. This is native actor-to-stream coordination, not demonstrated legacy-to-async conversion.

Saved test condition	Faulty	Repaired
Immediate memory-cache subscription	20/20 missing terminal event	0/20 missing terminal event
Subscription after awaiting response	20/20 missing terminal event	0/20 missing terminal event

Both production-module builds used Swift 6. The late-subscriber row checks the new replay behavior, but its interpretation differs from the first row: the patch also changes the documented contract for deliberately late subscription. The counted defect is the immediate subscription losing an outcome because of an unavoidable internal registration hop. Deliberately demanding replay from an API that documented no replay would not independently establish the same bug.

What passed, and what remains untested

The repaired revision passed the same selected fixtures. This validates those paths on the installed compiler/runtime. It does not cover all cancellation orders, multiple subscribers, every cache layer, URLSession behavior, platform variants or a full application. The tests are newly written research fixtures against unmodified production source, not an execution of the entire upstream test suite. The unchecked Sendable boundary remains part of the source context; a Swift 6 flag is not a proof that every participating component is checked.

Faulty: be75e5ff30d50ed6b7ce3fff6839d9b490678d20
Repair: 47bc90620d18f48e54668c26a34d0d14d554777a

5. Three handoff patterns in the source evidence

Retain an outcome, not just a waiting handle

Nuke P29 needs a terminal result after registration arrives late. Hummingbird P02 preserves an intervening produceMore notification using a generation counter around yield and stopProducing. Vapor P27 retains startup failure so both existing and late address waiters can receive it. All three concern information that a snapshot of current waiting state can fail to preserve. Only Nuke has a completed local reproduction in this report. [N1, H1, V1]

A	**Producer finishes or notifies** A consumer may not yet be registered.
B	**Receiver registers later** Current-state protection alone cannot reconstruct forgotten history.
C	**Retain the required evidence** Terminal result, error, cancellation flag or generation token.

Keep the right identity and the right number of waiters

Vapor P28 addresses two cooperating assumptions: repeated access to app.server must return the same server for running and waiting, and concurrent address waiters must not overwrite one another in a single continuation slot. The patch caches server identity and stores a collection of waiters. These are source-supported properties, not locally reproduced outcomes here. They remain one compound family to avoid inflating counts. [V2]

The earlier P27 terminal-error repair and later P28 identity/cardinality repair are complementary. A stored error does not connect two different server objects; a single protected slot does not preserve several waiting tasks. But a later repair is not evidence that the earlier patch caused the other problem. Introduction and root boundaries must be traced separately.

Revalidate when the work becomes useful

Kingfisher P30 propagates a cancellation/staleness token through queued disk-cache work and checks it at later stages. A network download cancellation handle does not necessarily cover cache reads, decoding or promotion to memory. GRDB P10 revalidates observation cancellation near the protected fetch/use point. These cases motivate lifecycle checkpoints; they do not establish that every proposed checkpoint is sufficient or that expensive work stops instantaneously. [K2]

6. Relation to split continuations

The split-continuation pitch explicitly identifies cancellation occurring before operation registration, the need for a shared identity, and the burden of remembering cancellation until registration. Its proposed wait phase installs handlers after registration can be completed. The pitch is primary design discussion, not a peer-reviewed result or evidence of a shipping guarantee. The underlying mechanism is the pitch author's motivation, not a discovery claimed by this report. [S1]

Where these cases are relevant

Kingfisher P07 gives a concrete cancellation-before-attachment example at an internal component boundary. A redesigned API might establish attachment before a cancellation handler can act. That conclusion requires checking the actual registration path: starting another unawaited Task to attach a target is not the same as having finished registration. TCA P09 provides a source example of capturing an optional cancellation handle before its subsequent installation; sharing a reference fixes stale capture, but does not by itself demonstrate every cancellation ordering. Neither family has been adapted to and tested against the proposed split API.

Obligation	What this report can say
Cancellation before registration	Relevant field evidence; no proposal-based repair validated
Exactly-once completion ownership	Must be established separately for competing producers
Retained terminal success/failure	Not automatically supplied for arbitrary event streams or omitted branches
Multiple waiters and shared object identity	Still application/API responsibilities
Queued cache staleness and resource cleanup	Require their own lifetime and propagation contracts

Questions useful to the design discussion

An API example should make completion of registration explicit before waiting begins, show how cancellation reaches an already-completed operation, and state who owns cleanup. It should also distinguish one operation's continuation from a broadcast stream or a collection of waiters. These questions come from the repair obligations above; they are review guidance, not a claim that the proposed API fails those requirements.

This report offers cases and limits for discussion. It does not claim that the pitch resolves Nuke's AsyncStream replay behavior, Vapor's server identity, or every cancellation-related source family. A language mechanism and an application's complete lifecycle protocol are different levels of the design.

7. Checking and detection: keep the claims separate

Complete concurrency checking diagnoses potential data-race hazards within its language model and available annotations. Unsafe opt-outs and foreign or manually synchronized components require separate reasoning. Even correctly synchronized access can preserve the wrong protocol, forget an outcome or select the wrong owner. This report's source-declared A/B/C tags are therefore context labels, not proof that all defective paths lie inside a fully checked boundary. [S2]

The completed Kingfisher component comparison supplies one Swift 5 complete-checking survival context. Nuke supplies one Swift 6 production-pipeline context. These establish bounded existence, not a compiler false-negative rate. No local Thread Sanitizer comparison was run. TSan evidence for IINA P17 and P19 came from reporter traces in saved discussions, not from this project's execution. [S3]

Audit column	Recorded evidence across 29 families
Strict checking	2 saved survival contexts; 21 source-predicted scope mismatches; 6 unknown
Thread Sanitizer	2 reporter detections; 2 candidates not run; 23 source-predicted scope mismatches; 2 unknown
Split-continuation pitch	2 redesign-relevant families, unvalidated; 27 roots not addressed by registration ordering alone
Nothing catches it	Unknown for every family; no exhaustive negative result

The columns are not interchangeable outcomes. A scope prediction means that the established root does not require the property the approach normally checks. It is not a demonstrated tool miss. A proposed API is also not a detector: applying it changes code and requires a separate equivalence and outcome argument. Reproducing one failure without running a sanitizer does not establish sanitizer blindness.

Why the distribution is not a prevalence result

The broader handoff reading gives 6/10 A families and 3/19 among other or conditional contexts. The narrower legacy reading gives only 2/10 A positives. Those observations are useful to describe the selected cases. They cannot isolate the effects of Swift 6 from era, repository mix, keyword targeting, unchecked boundaries or changes in the amount of async code. A majority of ten selected cases is not a reliable ecosystem forecast or a measured detection gap.

8. Practical review and regression guidance

For each asynchronous operation, write down the required lifecycle properties before deciding that a repair is complete. The questions below are derived from these cases and established concurrency practice. They have not been evaluated as a superior review method or a replacement for compiler checking, sanitizers or upstream tests.

Review question	Concrete check to add to the design/test plan
Who owns completion?	Make the claim/remove operation atomic; identify competing callback and cancellation paths.
What survives before registration?	Choose explicit memory for cancellation, terminal result or notification generation.
Which identity receives the result?	Capture stable operation/server identity and define reuse or replacement semantics.
How many receivers are allowed?	Specify single waiter, collection or broadcast; reject or retain additional registrations deliberately.
When is work obsolete?	Revalidate after queue hops and before publishing decoded/cached/UI results.
How does teardown make progress?	Coordinate callbacks with release; avoid circular waits and account for platform activation/wakeup rules.

Cover both sides of the handoff

A useful regression plan includes cancellation before attachment, cancellation after attachment and a no-cancellation control. For result delivery, include a fast synchronous/cache hit, an immediate subscriber whose internal registration is asynchronous, an ordinary delayed completion and the documented late-subscription contract. Add multiple-waiter and replacement-identity scenarios only where the API permits them. Distinguish cancellation delivery from cessation of physical work.

Make the ordering explainable

Prefer an explicit, valid API-level ordering over a loop whose only argument is that it ran often. Preserve production lock boundaries. Test adapters should identify which action is being modeled and whether they bypass application integration. A reduction that calls two state-machine methods in an order impossible under the production lock cannot validate the claimed bug. Save the faulty and repaired revision, language mode, flags, fixture, result and controls together.

These are proposed future review/test practices. No new executable tests were run while preparing this report. Finite passes do not prove all schedules safe, and no amount of repetition repairs an invalid reduction. The present tests are retained as bounded regression observations.

9. Negative evidence, limits and prior work

NIO remains disputed

NIO P01 (#3453) did not reproduce in the completed isolated state-machine/public-writer fixtures. Both revisions passed 100 selected guarded-gap sequences and 100 public cancellation trials. Source tracing found a second cancellation recheck under the same lock as continuation registration. A schedule inserted between that recheck and registration would violate the production critical section. The candidate is disputed rather than declared universally safe. A later instrumented executor experiment was interrupted and contains barrier-arrival failures; it supplies no completed paired result and is excluded. [D1]

What this report cannot establish

The 29-family classification is exploratory assistant first-pass coding. No investigator-ratified independent human ratings or kappa are recorded. Some families bundle obligations and could be split or recoded by a future reviewer. Targeted discovery, capped windows and repository clustering limit external validity. Historical source was tested with the installed modern toolchain/runtime, not the source-era environment. Internal components, production pipelines and full applications are different validation levels. No full upstream suite or TSan campaign is claimed.

An accepted patch and passing controls strengthen a local diagnosis; they do not prove universal correctness. Later related changes can concern a new property rather than a regression. Pre-merge IINA lock-order mistakes and bundled workaround reversions are not counted as independent shipped-fix failures. Lack of a later issue is not evidence that a repair is complete.

Existing work supplies the mechanisms

Earlier concurrency studies already analyze ordering, callback readiness and repairs beyond locks [R1, R2]. Cancellation propagation and cleanup have a substantial empirical treatment [R3]. Incorrect-fix research distinguishes incomplete fixes from newly introduced problems [R4]. Actor and Rust studies already show lifecycle/identity failures and semantic concurrency bugs beyond basic safety guarantees [R5, R6]. This report contributes concrete Swift case explanations and bounded regression evidence, not a new mechanism, taxonomy or first-study claim.

Authorship and assistance

AI assistance was used for source retrieval, extraction, provisional classification, research-fixture construction and drafting. Independent human classification reliability is absent. This technical report presents provisional source classifications; no external reviewer endorsement or affiliation is implied. A future academic study would require a distinct research question and appropriate validation. No academic schedule is active.

Appendix A. Case ledger (1/2)

Each row links to its upstream PR. Descriptions state the source-supported repair focus, not locally demonstrated success unless explicitly identified as Nuke P29 or the Kingfisher P07 follow-up. A*=compiler-conditional A/C; B*=Xcode B / SwiftPM C. Full historical revision pairs are retained in the companion JSON ledger.

Family / context	Project (PR link)	Source repair focus
P02 / A	hummingbird-project/hummingbird	count notifications and validate generation at handoff
P03 / A	swift-server/async-http-client	accept completion after early response and pending pause
P04 / A*	swift-server/swift-service-lifecycle	retain cleanup ownership; cancel sources and activate cleanup
P05 / A	awslabs/swift-aws-lambda-runtime	resume all pending states on connection close
P06 / B*	onevcat/Kingfisher	atomically remove and return callback batch
P07 / B*	onevcat/Kingfisher	early cancellation checks and actor-guarded once-only resumption
P08 / A	kean/Nuke	cancel orphan work and guard disposed state before early return
P09 / C	pointfreeco/swift-composable-architecture	share reference to subsequently installed subscription
P10 / C	groue/GRDB.swift	revalidate under fetch lock and return cancelled result
P11 / C	Alamofire/Alamofire	guard cancellation before invoking retry flow
P12 / C	kean/Nuke	derive completion from transition of task set to empty
P13 / C	pointfreeco/swift-composable-architecture	recursive lock around all throttle dictionary accesses; scheduler delivery
P14 / C	groue/GRDB.swift	asynchronous coalesced fetch without blocking writer for reader
P15 / C	mozilla-mobile/firefox-ios	rebuild header constraints only at deliberate setup/location changes
P16 / C	mozilla-mobile/firefox-ios	single-owner polling; extension only signals pending command

P07 links the original #2398 family; its completed local observation is the later #2526 component. P28 remains one compound identity/cardinality family. P01 is disputed and excluded; R01 is a separate runtime family. The ledger is targeted evidence, not a list of bugs newly discovered by this report.

Appendix A. Case ledger (2/2)

Each row links to its upstream PR. Descriptions state the source-supported repair focus, not locally demonstrated success unless explicitly identified as Nuke P29 or the Kingfisher P07 follow-up. A*=compiler-conditional A/C; B*=Xcode B / SwiftPM C. Full historical revision pairs are retained in the companion JSON ledger.

Family / context	Project (PR link)	Source repair focus
P17 / C	iina/iina	serialize all draw work on private mpvGLQueue
P18 / C	iina/iina	guard all affected window-state and resize entry points with closing state
P19 / C	iina/iina	lock compound subsystem lookup/insertion and UI iteration
P20 / C	iina/iina	shared lifecycle lock across uninit, display, draw and swap reporting
P21 / A	apple/swift-nio	close full channel when writeEOF observes input already closed
P22 / A	apple/swift-nio	socket-pair wakeup integrated with WSAPoll
P23 / C	onevcat/Kingfisher	move expensive scan off state-coordination queue; publish snapshot briefly
P24 / C	mozilla-mobile/firefox-ios	remove global-queue refresh and execute on caller main-thread path
P25 / C	iina/iina	retain renderer context; lock and set it on each calling thread
P26 / A	awslabs/swift-aws-lambda-runtime	atomic compare-exchange run gate released with defer
P27 / A	vapor/vapor	retain startup error and resolve both existing and late waiters
P28 / A	vapor/vapor	cache one server instance and retain all address waiters
P29 / A	kean/Nuke	retain/replay terminal result when registering late event stream
P30 / B*	onevcat/Kingfisher	thread-safe cancellation token, checkpoint revalidation and stale outcome propagation

P07 links the original #2398 family; its completed local observation is the later #2526 component. P28 remains one compound identity/cardinality family. P01 is disputed and excluded; R01 is a separate runtime family. The ledger is targeted evidence, not a list of bugs newly discovered by this report.

Appendix B. Reproduction and artifact provenance

The completed observations were recorded on a Mac Studio (Mac13,1), Apple M1 Max, 32 GB, macOS 27.2 build 26B123, Apple Swift 6.4.2 (swiftlang-6.4.2.1.9), Xcode 27.2 build 27B52 and the macOS 27.2 SDK. These identifiers are copied from the frozen execution record; historical sources were not tested on historical runtimes. No rerun was performed for this report.

Kingfisher #2526

Compared parent: fcf02a9ab324e2a346aa5f82480d60e1c6a04448
Repair: 77173fa007500c701392d8376a5fa43a4aee9990

Nuke #916

Compared parent: be75e5ff30d50ed6b7ce3fff6839d9b490678d20
Repair: 47bc90620d18f48e54668c26a34d0d14d554777a

NIO #3453 (disputed)

Compared parent: 5bf841dde54db41048f17a394061753d91fc889a
Repair: d2feeaa3631561117aa5276a5166505fe5dcfab7

Research changes and execution scope

Nuke and Kingfisher add research test targets; NIO isolates the research target after an unrelated integration-test compile failure on the newer SDK. No production-source instrumentation is present in the completed paired artifact. Kingfisher's adapter selects the old actor action versus repaired component calls. Package dependency setup used network access, while the fixtures themselves were offline. Compiler command records establish production language mode and the extra complete-checking context. Initial setup failures, warnings, timeouts and partial experiments are not runtime successes.

What the retained record contains

The private working artifact retains fixtures, test-only manifest diffs, selected dependency resolution, raw build/test logs, commands, exit codes and available timestamps. The report companion contains a derived 29-case ledger, sanitized reproduction summary and SHA-256 provenance. It does not republish full third-party patch payloads or raw machine-specific logs. No public archive URL or DOI has yet been assigned to this companion. Source PR and commit links remain independently inspectable.

Some initial invocation wall-clock intervals were not captured; test durations are preserved in their logs. Later runs have command timing records. Build time depends strongly on cache state and is not human adjudication time. Per-case repair-coding times were unmeasured, and no human-cost estimate is inferred. The full 29-case diagnostic comparison is a source-based triage matrix, not an executed benchmark.

References and source links

[R1] Lu, Park, Seo and Zhou. Learning from Mistakes: A Comprehensive Study on Real World Concurrency Bug Characteristics. ASPLOS, 2008. Sections 5.1-5.2. Source.

[R2] Wang et al. A Comprehensive Study on Real World Concurrency Bugs in Node.js. 2017. Section VII. Source.

[R3] Sethi, Pan, Lu, Musuvathi and Nath. Cancellation in Systems: An Empirical Study of Task Cancellation Patterns and Failures. OSDI, 2022. Section 5.2.1. Source.

[R4] Yin, Yuan, Zhou, Pasupathy and Bairavasundaram. How Do Fixes Become Bugs? ESEC/FSE, 2011. Section 4.2. Source.

[R5] Bagherzadeh, Fireman, Shawesh and Khatchadourian. Actor Concurrency Bugs: A Comprehensive Study on Symptoms, Root Causes, API Usages, and Differences. OOPSLA, 2020. Source.

[R6] Qin, Chen, Yu, Song and Zhang. Understanding Memory and Thread Safety Practices and Issues in Real-World Rust Programs. PLDI, 2020. Source.

[S1] Franz Busch. [Pitch] Split continuation. Swift Forums, October 15, 2026. Primary design discussion, accessed November 6, 2026. Source.

[S2] Swift project. Swift 5.10 Released. Complete concurrency checking and unsafe opt-outs. Source.

[S3] LLVM project. ThreadSanitizer documentation. Instrumentation and runtime detection. Accessed November 6, 2026. Source.

[K0] Kingfisher #2398: continuation handling and setup cancellation. Source.

[K1] Kingfisher #2526: cancellation attachment and related repairs. Source.

[K2] Kingfisher #2497: stale disk-cache retrieval. Source.

[N1] Nuke #916: terminal event replay and immediate subscription ordering. Source.

[H1] Hummingbird #908: lost produceMore notification. Source.

[V1] Vapor #3506: startup failure propagation. Source.

[V2] Vapor #3536: waiter storage and server identity. Source.

[D1] SwiftNIO #3453: disputed candidate in this review. Source.

All case PR links in Appendix A are additional primary source references. Reference metadata and inspected sections were checked in the preceding literature review. No claim of exhaustive literature coverage is made.