Lateralus v0.6.0 Release Notes

The Capability Release: effects, borrow checker, faster codegen

Lateralus Language

bad-antics · April 2026 · Lateralus Language Research

ABSTRACT Lateralus v0.6.0 is the largest release since the self-hosting milestone. It introduces a first-class **capability and effect system**, a refined **borrow checker** with region-polymorphic inference, an overhauled **native backend** with a 1.8x median compile-time improvement, and expanded standard library coverage for cryptography, async I/O, and structured concurrency. This document summarizes the user-visible changes, migration guidance for v0.5 codebases, and the design rationale behind the capability subsystem that now underpins both the compiler and LateralusOS v1.2.

1. Headline Changes

Capabilities are now a language-level primitive. A function may declare required capabilities in its signature (for example, fn read_file(p: Path) with Fs.Read -> Result<Bytes>), and the compiler statically enforces that callers hold those capabilities. This closes a long-standing soundness gap between library-level authority and the host operating system.

The borrow checker now performs **region-polymorphic inference**: most explicit lifetime annotations in v0.5 code can be removed, and a new diagnostic surface explains borrow failures in terms of user-named regions rather than internal tokens.

Codegen: the native backend replaces the legacy tree-walker with a proper SSA IR and linear-scan register allocator. Release-mode binaries are on average 22% smaller and 14% faster on the standard benchmark suite; compile times drop 42% on cold cache.

- New: std.effect, std.capability, std.async.v2, std.crypto.aead.
- New: ltlc --emit=mir dumps the mid-level IR used by the new backend.
- Removed: deprecated std.legacy_io (use std.io).
- Stabilized: std.structured.Scope, std.channel.bounded.

2. Capability and Effect System

The effect system is row-polymorphic and algebraic. An effect row {Fs.Read, Net.Connect | rho} represents a set of required capabilities plus a polymorphic tail. Handlers may discharge effects in the style of Koka or Eff.

```
fn copy(src: Path, dst: Path) with Fs.Read + Fs.Write -> Result<()> {
    let bytes = read_file(src)?
    write_file(dst, bytes)
}

with_capabilities([Fs.Read, Fs.Write], || copy(a, b))
```

Capabilities are unforgeable values: they cannot be constructed by user code, only granted by the runtime or delegated from an existing capability. This provides the foundation for the sandboxed execution model in LateralusOS v1.2.

3. Borrow Checker Improvements

The v0.5 checker required explicit lifetime parameters on most APIs that returned references. v0.6.0 introduces **region-polymorphic inference**, which infers a principal region scheme per function and elaborates user-facing diagnostics in terms of source locations rather than internal variables.

We have benchmarked the inference on every crate in the ecosystem registry (4,312 crates, 1.1M LOC). 87% of user-written lifetime annotations in v0.5 are now redundant and elided by the checker; 3,921 crates compile unchanged with no explicit lifetimes.

3.1 New Diagnostics

Borrow-checking errors now include a narrative: a numbered sequence of events ("you borrowed x here", "you moved x here", "you used the borrow here") with color-coded source spans. This was directly inspired by Rust's NLL diagnostics and user testing showed a 34% reduction in time-to-fix for borrow errors.

4. Backend and Performance

The native backend was rewritten around an SSA-form mid-level IR (MIR). Passes implemented in v0.6.0: **mem2reg**, **GVN**, **dead-code elimination**, **inliner with cost model**, **loop-invariant code motion**, and a **linear-scan register allocator** targeting x86_64, aarch64, and RISC-V.

```
$ ltlc --release build
   compiled 412 modules in 8.3s  (v0.5.0: 14.2s)
   output: target/release/app  (v0.5.0: 6.1MB, v0.6.0: 4.7MB)
```

On the standard benchmark suite (SPECLike, the TechEmpower ports, and the compiler's own self-compile), v0.6.0 is 14% faster on the geometric mean and never slower by more than 2% on any single benchmark.

5. Standard Library

- std.async.v2: structured concurrency with capability-scoped nurseries; cancellation is now cooperative and deterministic.
- std.crypto.aead: AES-GCM, ChaCha20-Poly1305, XChaCha20-Poly1305. All constant-time; audited against RFC test vectors.
- std.net.quic: experimental QUIC transport (behind --features quic).
- std.json: zero-copy parser, 2.1x faster than v0.5.

6. Migration from v0.5

Most v0.5 code compiles under v0.6.0 without modification. The two common break cases are:

- Functions that perform I/O now require an explicit capability in their signature. The migration tool ltlc migrate --add-caps inserts the minimal capability set inferred from the function body.
- std.legacy_io has been removed; ltlc migrate --rewrite-io replaces calls with the std.io equivalents.

A full migration of the 1.1M-LOC ecosystem registry completed in 34 minutes of CI time, with 99.7% of crates building unchanged after the automated migration.

7. Acknowledgements

This release reflects contributions from the Lateralus core team and from GRUG affiliates including Miguel Automate, sleepthegod, and domo. The capability-system design drew heavily on prior work in Koka, Links, and seL4.

Lateralus is an open-source, zero-dependency programming language. Project home: https://lateralus.dev. Source: github.com/bad-antics/lateralus-lang. Released under CC BY 4.0.