Raku++

A from-scratch Raku, hand-written in C++17.

- **Interpreter** + native compiler
- runs in the **browser** via WebAssembly
- runs **zef-installed** modules
- **Zero** third-party dependencies
- v2.0.0

NOT A RAKUDO FORK — SHARES NO CODE WITH IT

One implementation of the language, from lexer to native binary.

A hand-written lexer, parser, and tree-walking evaluator for real **Raku** — classes, roles, grammars, regexes, multi-dispatch, junctions, lazy sequences, a bignum tower, Unicode-correct strings, concurrency. It can also **compile** a program to a standalone native executable, and — as **Raku.js** — run client-side in a browser.

- **C++17, no deps**
  Hand-rolled bignum, Unicode tables, and regex engine. Builds anywhere CMake and a C++17 compiler run.
- **Interpret *or* compile**
  The same binary tree-walks a script or transpiles it to C++ and compiles native.
- **Measured, not claimed**
  Graded test-by-test against Roast, the official Raku specification suite.

GRADED AGAINST ROAST · MEASURED ON V2.0.0

How much Raku actually runs.

90%

197,090 / 218,675 declared tests pass — 97% of the tests that actually run.

594 / 1,462

files pass on the strict **all-or-nothing** bar — every assertion in the file must pass (~41%). v2.0.0 made Pair-form subtests actually run, which removed ~2,340 vacuous passes: a stricter bar than the 43% before it.

100%

of **S15** — Unicode, strings, NFG — **91,752 / 91,752** assertions, closed completely for v1.1.

Every release is gated on the full suite with **zero fully-passing-file regressions** — and, since v1.5.0, on **performance** too. Numbers are measured on each tag, never projected; the whole suite runs in ~3½ minutes through a harness written in Raku and run by Raku++ itself.

THE BREADTH OF WHAT'S IMPLEMENTED

Big-language Raku, most corners lit.

NUMBERS
Exact by default
Arbitrary-precision `Int`, exact `Rat` — `0.1+0.2-0.3` is exactly `0` — `Num`, `Complex`, allomorphs.

OBJECTS
class · role · grammar
Multi-dispatch, `BUILD` / `TWEAK`, mixins, `augment`, full `.^` introspection, `.wrap`.

REGEX
Grammars that parse
Backtracking engine, Unicode classes, `token` / `rule`, `.parse` with action classes & `make`.

SIGNATURES
multi & proto
Type / literal / `where` constraints, destructuring, coercion types, `callsame`.

FUNCTIONAL
Lazy & higher-order
Closures, `*` WhateverCode, junctions, `gather` / `take`, feeds, `1,2,*+*...Inf`.

METAPROG
Your own operators
User ops in all six categories with working precedence traits and meta-operators over them.

CONCURRENCY
Real OS threads
`start` / `await`, `Supply` / `react` / `whenever`, `Channel`, atomics; opt-in true parallelism.

UNICODE
Grapheme-correct
UAX #29 (emoji ZWJ), NFC/NFD/NFKC/NFKD, UCA collation — generated UCD/UCA 17.0 tables.

ONE SOURCE, FIVE TARGETS

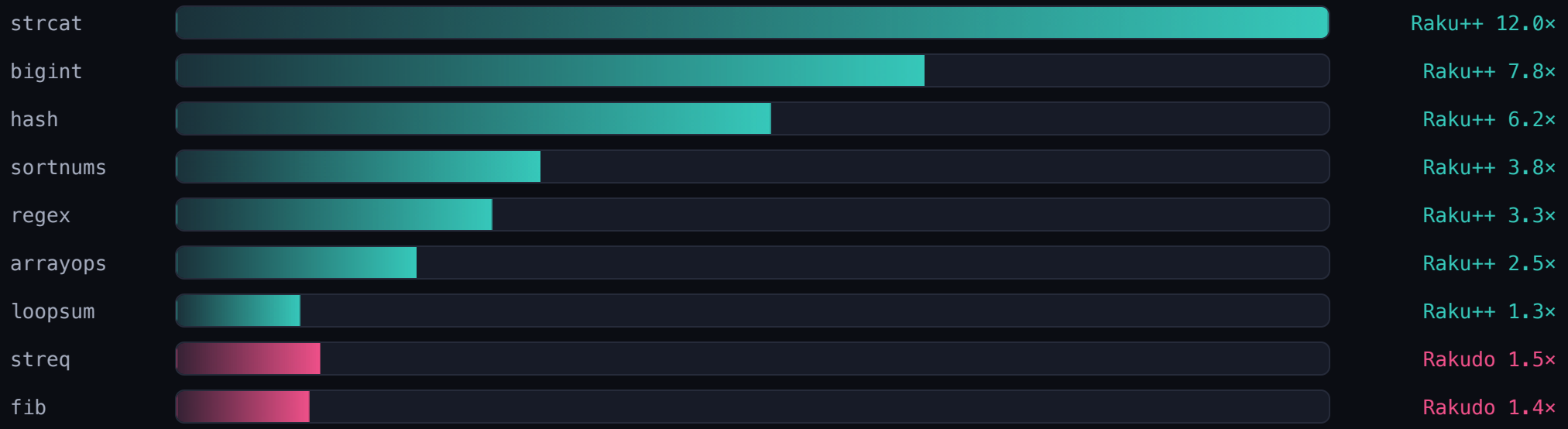
Four ways to run — and a fifth in the browser.

`rakupp prog.raku`	**Interpret** — the default; covers the whole language.
`--bundle -o prog`	Standalone binary that re-parses the source at startup.
`--aot -o prog`	Standalone binary embedding the already-parsed AST.
`--exe -o prog`	**Transpile to C++ and compile native** — anything unsupported falls back to `--bundle` automatically.
`Raku.js → WASM`	**In the browser** — same semantics as native, client-side, no server, with an embeddable playground.

Opt-in, off by default: `--precomp-modules=on` caches the parsed form of everything a program uses — use XML (10 files, 1,110 lines) goes 16.0 ms → 5.7 ms cold-to-warm. Nothing is written to disk until you ask.

INTERPRETER · × VS RAKUDO V2026.06 · BYTE-IDENTICAL OUTPUT GATE

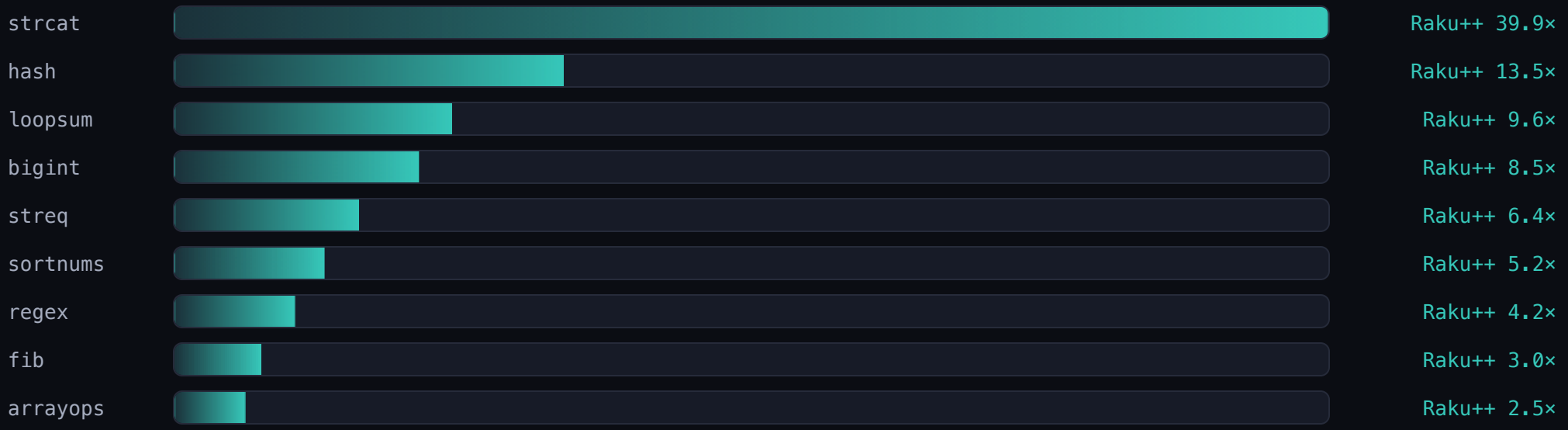
Fast to start, fast to run.



~2 ms cold start (best 1.8 ms of a 200-spawn loop) — a tiny native binary, no VM to spin up. Even interpreted, Raku++ wins 7 of 9 kernels; the two it trails on — streq 1.5×, fib 1.4×, both narrowed by node specialization in v1.7.0 — the next slide compiles away.

RAKUPP --EXE · TRANSPILED TO C++, COMPILED NATIVE · × VS RAKUDO

Compiled, it wins every kernel.



--exe strips the tree-walk: every benchmark lands 2.5× to 39.9× ahead of Rakudo — streq and fib, the interpreter's only two losses, now included. Anything the compiler can't lower falls back to bundling automatically.

ONE INTERPRETER BEHIND EVERY SURFACE

A small constellation, one source of truth.

`src/` — the C++ interpreter

native
The binary + `--exe` compiler.
`brew · releases · CI action`

Raku.js
src/ compiled to WebAssembly (Emscripten).
`rakujs.{js,wasm}`

raku.online
Public playground — editor, share links, widget.
`embeds the wasm`

the spec
Behavioural spec; examples verified live.
`raku.online/spec`

the tour
18 lessons, every example a live editor.
`raku.online/tour`

raku-corpus
Real programs as a beyond-Roast test target.
`differential run`

Everything downstream ships the *same* interpreter — the wasm, the sub-sites, the Homebrew tap, and the release binaries all trace back to `src/`. Neither the spec nor the tour hosts an engine of its own, so both inherit a new one for free the moment the playground redeploys.

MID-SIZE PROGRAMS, EACH LEANING ON A DIFFERENT PART OF THE LANGUAGE

Real programs, written in Raku.

Scheme
A Lisp interpreter on a Raku grammar.
GRAMMAR · EVAL

Forth
Stack machine, also grammar-parsed.
GRAMMAR

Markdown → HTML
Converter — also runs in the browser.
REGEX · WASM

JSON
Parser + formatter, browser-ready too.
GRAMMAR · WASM

Pastebin
HTTP service on raw sockets.
SOCKETS

Chat server
Concurrent, many-client.
THREADS

Key-value store
Its own wire protocol.
SOCKETS

rakus
A static-file HTTP server.
SOCKETS

Plus three interpreters for *other* languages, each written in Raku: **JavaScript/TypeScript**, **Perl 5** (Raku parsing its own ancestor), and **Python 3** — off-side rule included, via a CPython-style INDENT/DEDENT tokenizer — with examples byte-identical to `node`, `perl` and `python3`. Every one of the 24 bundled `examples/` also compiles natively with `--exe`, byte-for-byte identical.

THE V2.0 CAMPAIGN — WHAT PEOPLE ACTUALLY USE

Modules from the ecosystem, as installed.

Raku++ ships no package manager. It reads the **same store zef already populates** — install a module the normal way, then `use` it from Raku++ with no flags and no reinstall.

THE BAR

50 / 59 distributions

pass *their own* test suites — the files zef runs at install time, not a one-line API probe. 32 → 50 in one release.

THE SET

Top 50 by reverse-deps

JSON::Fast, XML, YAMLish, URI, File::Temp, File::Find, Terminal::ANSIColor, Digest, Encode — ranked over the ecosystem archive, version-pinned.

THE METHOD

Every fix is general

No module accommodations: `is raw` write-back, `CALL-ME` by name, `method FALLBACK`, `END` at process exit — all landed as interpreter fixes.

Far end of the same road: a live **`HTTP/1.1 200 OK` over TLS** — Cro's client on `IO::Socket::Async::SSL`, talking to the system OpenSSL through Raku++'s own NativeCall. Getting there was a chain of ~13 bugs, and nearly every one was a general correctness bug.

BEYOND WHAT ROAST'S UNIT GRANULARITY EXERCISES

Real applications run unmodified.

RAKU-COURSE

~1,500 pages, byte-identical

The full course static-site generator regenerates the entire site byte-for-byte identically to Rakudo — including the real zef-installed `YAMLish` module.

COVID.OBSERVER

End-to-end build

The site builder runs start to finish on Raku++ — a genuine data-processing application, not a curated snippet.

```
# exact arithmetic, lazy infinite sequence, grapheme strings —
# the things that make real programs match, not just tests
say 0.1 + 0.2 - 0.3;              # 0  (exact Rat)
say (1, 2, *+* ... *)[^10];  # 1 1 2 3 5 8 13 21 34 55
say "café".chars;               # 4  (graphemes, not bytes)
```

HONEST ABOUT THE EDGES

Not there yet.

- **Macros / RakuAST / slangs** — compile-time metaprogramming and pluggable syntax. Deliberately deferred, with the scope measured first.
- **C structs passed by value**, and callbacks fired from a thread the C library owns. The rest of NativeCall landed: `libffi` found at run time with nothing to install, plus structs, unions, pointers, callbacks and variadics.
- **Regex code blocks under backtracking** — `/(\d) { say $0 }/` re-fires its side effects; the fix has to defer them to the accepted path.
- Some **IO** and **POD** corners; **true parallelism by default** — the primitives (`atomicint`, `Channel`, `Supplier`) are done, the thread pool and the gates are not.
- The last **9 of 59** distributions — `NativeHelpers::Blob` is MoarVM-bound by design, `AttrX::Mooish` wants deep metamodel emulation.

The cheap Roast wins are spent; moving the count now takes whole features — parse, runtime and dispatch together. Every release stays gated on no regressions, in conformance *and* in speed.

MACOS · LINUX · WINDOWS · THE BROWSER

Try Raku++ today.

```
$ brew tap ash/rakupp   # macOS
```

```
$ brew install rakupp
```

```
↗ raku.online   # run it in the browser, no install
```

github.com/ash/rakupp · raku.online · raku.online/spec · raku.online/tour

Prebuilt archives on every GitHub Release — macOS universal, Linux static, Windows static-CRT — plus the WebAssembly bundle. No runtime dependencies, anywhere.