

SAILSCONF 2026 · OPEN SOURCE

Rebuilding Panda's Engine

Rust, WebAssembly, and 37× faster extraction.



Adebesin Tolulope · Lope

Maintainer · Chakra UI, Ark UI & Panda CSS

WHERE THIS STARTED

Before any code, we asked one question.



New majors across the ecosystem, from the bottom up. Each layer has to be ready before the one above it can move. So Panda went first, and we asked: *what do you want from v2?*



GROUND FLOOR

Panda is a styling engine. You define, it generates.

```
const styles = css({ color: 'primary', p: '4' })  
//      ↓ the engine turns your call into  
.text_primary { color: var(--colors-primary) }  
.p_4 { padding: var(--spacing-4) }
```

Tokens, recipes, conditions — you write the intent, the compiler emits the CSS. Everything today is about that compiler.



THE ANSWER, IN THEIR WORDS

Not one complaint. Years of them.

Dec 2023 "How to troubleshoot *very* slow extraction times?"

Jan 2024 "Vite HMR unusably slow"

Nov 2024 "Slow TypeScript Language Server Performance with Panda CSS"

Jan 2025 "Slow performance during build (> 30 seconds, nothing cached??)"

Apr 2026 "Panda CSS v2: feature requests from a large-scale adopter"

github.com/chakra-ui/panda · #1866 · #1970 · #2977 · #3106 · #3522



THE WALL

Thirty seconds to build. Worse in watch mode.

Teams building large design systems on Panda kept hitting it. They weren't wrong.

our bar: half a second

past that, dev tools feel slow



WHERE THE REWRITE CAME FROM

One reply went further. It came with a prototype, in Rust.



"I prototyped a Rust port that built a medium-sized app in under 100ms, so the approach is viable."

One of our biggest adopters, answering our call for v2 ideas. The rewrite was pulled out of us by the people depending on us.



SO WE DID IT

We rewrote the engine.

Rust on Oxc, instead of TypeScript on Node.

Seven months. Two maintainers who learned Rust by shipping it.

WHAT CHANGED FOR YOU

Nothing. Same `css()`, same recipes, same CSS out.

```
// v1 and v2 - byte for byte identical
const styles = css({
  color: 'primary',
  fontSize: 'lg',
  padding: '4',
  _hover: { color: 'accent' },
})
```

The machine underneath is new. Your code is not.



WHY THE OLD ENGINE HIT ITS CEILING

Whole-program graph

ts-morph held the entire TypeScript program in memory to answer one style call.

Interpreter on the hot path

A JS evaluator folded values like `sizes.lg * 2` on every build.

Node-only

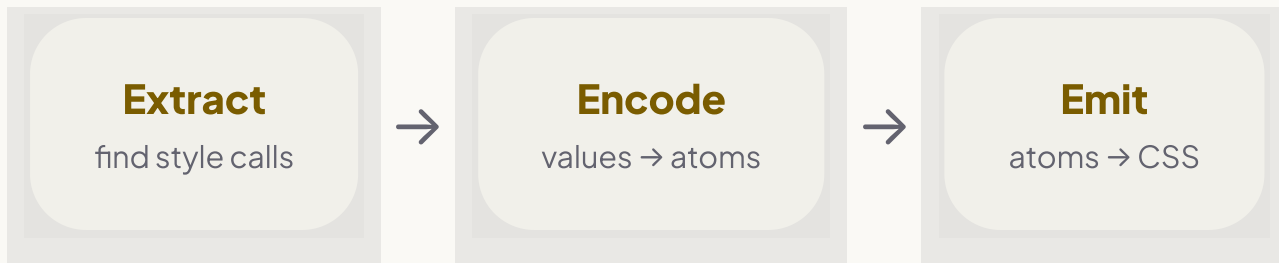
Playground, editor and diagnostics each re-implemented extraction, and drifted.

Panda's real question is narrow: what value does this one style call get? The old engine paid for the general answer anyway.



THE NEW PIPELINE

Three passes, one direction.



Seventeen small Rust crates, no cycles. Parsing runs on Oxc — the same Rust toolchain behind a new wave of JS tooling.

Answer the narrow question. Skip everything else.

One Oxc parse per file

Shared by imports, `css()` calls, JSX and identifier lookup. Files that never import Panda are skipped, including most of `node_modules`.

Values fold in Rust

Imports, operators, ternaries, enums, `token()`. No JS interpreter. Values from other files are kept as small descriptors, not whole files.

CSS emitted natively

No PostCSS pass. Media-range syntax, merged queries, and `@property` variables only when used.

v1 did each of these in the general case. v2 only does what Panda needs.



v1's TypeScript toolchain weighed about as much as all of v2.

95.3 MB → 38.2 MB

install size · -60%

141 → 41

dependencies

37 MB

typescript + ts-morph, in v1 alone

Oxc, compiled into Panda's native binary, replaced the TypeScript compiler you used to download.

37x

faster extraction

cold parse + style extraction, 100 files

184 ms → 5 ms. And I'll show you exactly what that measures.



AGAINST WHICH BASELINE

A speedup you can't inspect isn't worth much.

OLD ENGINE	ITS COLD PARSE	PANDA 2.0 · RUST	SPEEDUP
Panda 1.11.3	184 ms	5 ms	37x
Panda 1.12.0	105 ms	5 ms	21x

Same v2 either way — its cold parse is a stable ~5 ms. The multiplier only moves because the old engine got faster between 1.11 and 1.12. So I name the baseline.



EXTRACTION WASN'T THE ONLY WIN

85×

faster staticCss

25.7 s → 0.3 s · 29k rules

~360×

watch re-parse

per changed file

~99%

fewer TS type instantiations

40–60% less type-check time

Faster to build, faster to save, and lighter on the tools around it.



NOT OUR BENCHMARK — THEIRS

A 107-project monorepo, in production.

64 s → 3.8 s

wall-clock build · ~17x

76 s → 1.5 s

CPU time · ~50x

4 GB → 157 MB

peak memory · ~25x

Output CSS essentially identical. One config line changed. An adopter validating the rewrite for us, in their own words.



One engine. Two thin bindings.

Native · NAPI

@pandacss/compiler

The CLI, Vite and PostCSS. This is your build.

WebAssembly

@pandacss/compiler-wasm · ~490 KB gz

The browser and the playground. Same engine, same CSS.

One extractor, not three that drift. The bindings only pass data in and CSS back out to JavaScript.



KEEPING THE PROMISE

"Identical" is a claim you have to prove.

15 of 20

emitter cases byte-identical

5 of 20

different on purpose

Modern `@media (width ≥ ...)` range syntax
and inline `color-mix()`. Choices, not
regressions.

A test suite is half the proof. The other half is real people upgrading real apps.



MIGRATE ON YOUR OWN CLOCK

No forced march.

```
npm i @pandacss/dev@latest
```

→ v2, Rust

```
npm i @pandacss/dev@1
```

→ stay on v1

What you'll actually touch

ESM only, Node 22+: `"type": "module"`

Presets are explicit in your config, not injected

`createStyleContext` → `createRecipeContext`

`createSlotRecipeContext`

Thousands of production apps depend on Panda's primitives. A rewrite that breaks them on a Tuesday is not a rewrite worth shipping.



THE PART MOST REWRITE TALKS SKIP

**Deciding what not to build
was most of the work.**

THE WISHLIST, TRIAGED

SHIPPED

A shared, publishable styled-system

Tree-shaking a design system's unused CSS

A version guard between packages

DECLINED, ON PURPOSE

Variants through arbitrary wrapper components

Plural design systems

A package registry

Each declined one opened questions we couldn't answer cleanly yet. Saying no kept Panda something you can reason about.



ONE "NO", IN DETAIL

Tracking a recipe through a wrapper you wrote.

```
const MyButton = (props) => <Button {...props} />  
// <MyButton size="lg" /> - which recipe variant is that?
```

A real feature, with its own design. We shipped a limited version and wrote the rest down as "not yet", in the open.

**Escape hatch
today**

tell the recipe about your wrappers: `jsx: [/Button$/]` matches `Button`, `MyButton`, `IconButton`...



DEMO

The compiler, in a browser tab.

Type Panda styles. Watch it emit CSS. No server.

WHAT YOU'RE SEEING

The same Rust engine, running as WebAssembly.

```
// left: you type this
const styles = css({ color: 'red.500', padding: '4' })
```

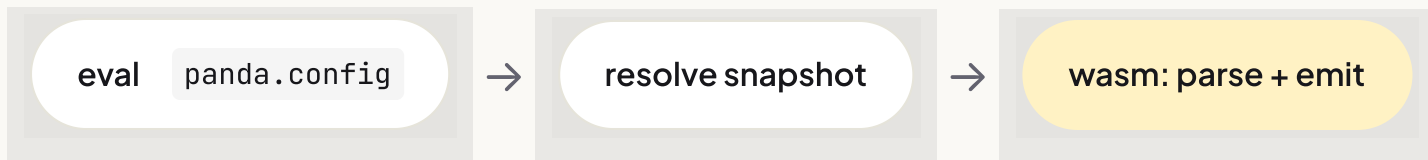
```
/* right: the wasm engine emits this, live */
@layer utilities {
  .p_4 { padding: var(--spacing-4); }
  .c_red\.500 { color: var(--colors-red-500); }
}
```

No install, no server. The compiler that runs your build is the one running in the tab.



HOW IT WORKS

Your config, evaluated in the browser, fed to wasm.



Change a token in the config and every style recompiles against it, right there in the tab. Same engine, same CSS as your build.



DEMO

A design system you can publish.

Author it once. Consume it in one line.

PUBLISH, THEN CONSUME

```
// in the design system package  
"scripts": { "lib": "panda lib" }
```

```
// in every app that uses it  
export default defineConfig({  
  designSystem: '@acme/ds',  
})
```

One line pulls in the tokens, recipes, and the CSS the design system already generated.



WHY THAT'S NEW

The app only builds what it adds.

0

re-extraction in the consumer

only used

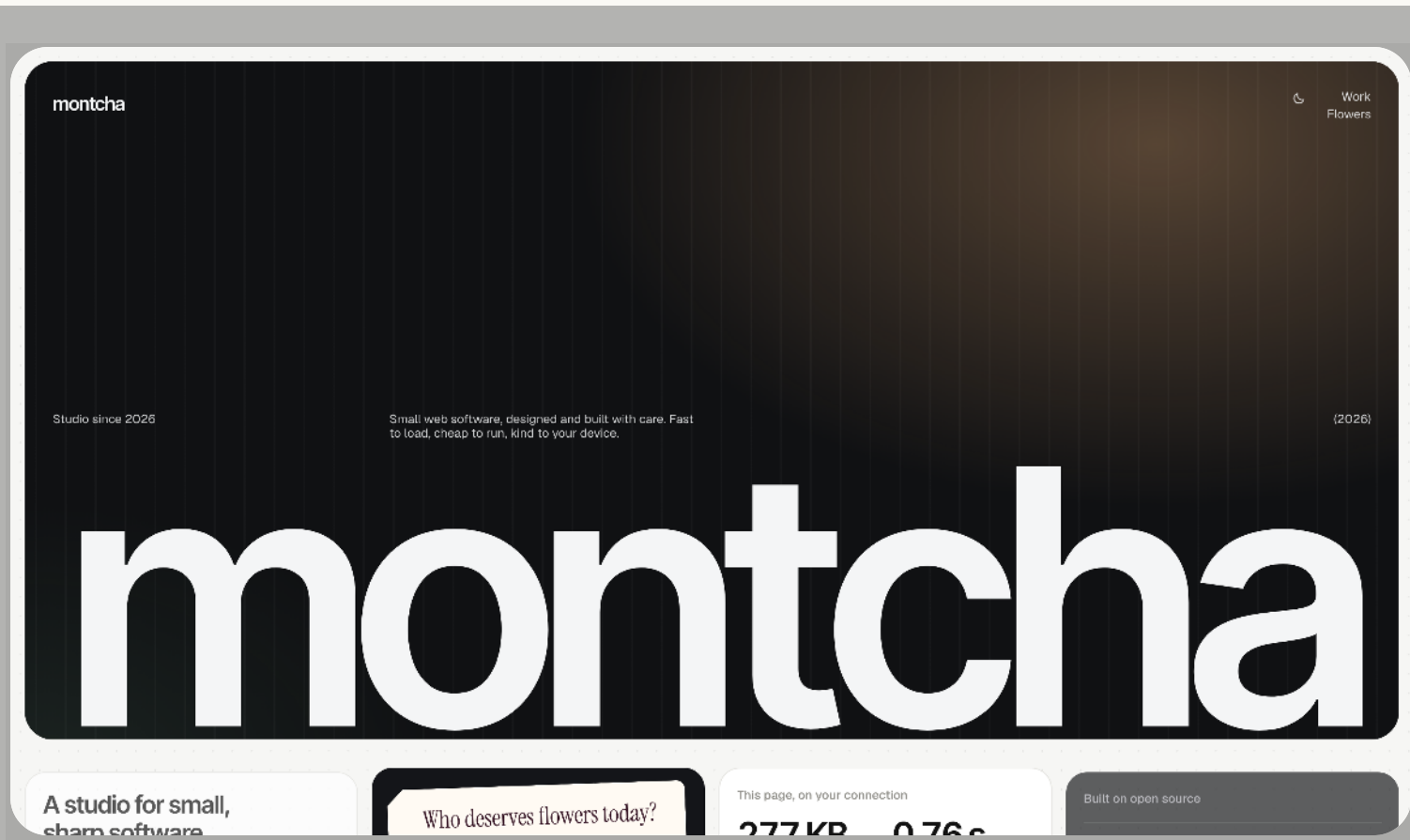
component CSS ships

Overlay codegen reuses the library's styled-system instead of copying it. Import ten of a hundred components, ship the CSS for ten.



I built Montcha on it.

studio.adebesin.com



Flowers, a Montcha app.

flowers.studio.adebesin.com



Every app is one config.

```
// apps/web/panda.config.ts — the whole thing
export default defineConfig({
  designSystem: '@montcha/ui',
  include: ['./app/**/*.vue,ts'],
  outdir: 'styled-system',
  preflight: true,
  strictTokens: true,
})
```

4 apps · 1 system

Nuxt + Ark UI, all on `@montcha/ui`

0.21s

`panda lib`, the whole design system

0.08s

app codegen, under the half-second bar

Plain `.vue` files, no Vue plugin. The goal: a thousand apps on one design system.



THE ONE THE REWRITE UNLOCKED

Now the styling runtime leaves your bundle too.

```
// you write
export const title = css({ fontSize: '2xl', color: 'gray.900' })

// your bundle ships — no call, no import, nothing to run
export const title = 'fs_2xl c_gray.900'
```

transform: true

across

vite

webpack

rollup

bun

"Zero runtime" used to mean no injected styles. Source transforms rewrite every call site to the class string Panda already computed. One flag — the bundler was always the one place that could.



Small factories for things you used to hand-roll.

```
import { css, firstThatWorks, keyframes } from 'styled-system/css'

const fade = keyframes({ from: { opacity: 0 }, to: { opacity: 1 } })

css({
  color: firstThatWorks('oklch(55% 0.18 250)', '#0057b8'),
  minHeight: firstThatWorks('100dvh', '100vh'),
})
```

viewTransition()

firstThatWorks()

keyframes()

positionTry()



And a lot around it.

ESM-native

Node 22+, CommonJS only where
PostCSS and Next need it

`doctor` · `analyze` · `debug`

`--profile` includes Rust engine
time

First-party ESLint plugin

same extractor, runs under oxlint
too

Vue · Svelte · Astro, natively

no framework plugins needed

`@property` **variables**

emitted only when used

`npx @pandacss/mcp`

your design system, readable by
agents

Half of this is legibility: see what Panda is doing, don't guess.



THE ARGUMENT

Speed at this layer isn't a nice-to-have. It's the whole competition.

And a rewrite under a live ecosystem is mostly about the people you can't afford to break.



IF YOU TAKE THREE THINGS

- 01** A rewrite can be invisible. Same API, same CSS, new machine.
- 02** Measure so the number means something. Name the baseline.
- 03** Deciding what not to build is the work. Say no in the open.





Thank you.

Panda CSS 2.0 — the engine is Rust now, and your code didn't change.



Adebesin Tolulope · Lope

panda-css.com · play.panda-css.com · studio.adebesin.com

Questions? I'm in the chat.

