Oxc

The JavaScript Oxidation Compiler

A deep dive into Oxc, Rust and Performance.

Who am I

- My name is Boshen
- Frontend configuration engineer
  - grunt gulp webpack
- Worked on super large monorepos and web apps
- Started working closely with JavaScript tools written in Rust
  - Rolldown Rspack Biome swc napi-rs
- Initiated and lead the Oxc project
- Member of the Rolldown project
- Member of VoidZero Inc.

What is Oxc?

JavaScript Oxidation Compiler

A place to learn Rust and JavaScript by contribution

oxc.rs

Oxc Search ⌘K Guide Learn Contribute Playground Resources

The JavaScript Oxidation Compiler

A collection of JavaScript tools written in Rust

Get Started View on GitHub

Parser ✅
3x faster than swc
Usage guide →

Linter ✅
50~100x faster than ESLint
400+ rules and counting
Usage guide →

Resolver ✅
28x faster than enhanced-resolve
Usage guide →

Transformer 🚧
Babel compatible
Isolated Declarations Dts Emit

Formatter
Prettier compatible

Minifier
Faster and better at compression

Rolldown Bundler 🚧
Rollup compatible
Designed for Vite

Nova JavaScript Engine 🚧
ECMAScript specification with data-oriented design

Who’s using Oxc?

- Rolldown uses the oxc crate for parsing and transformation
- Nova engine uses the oxc crate for parsing
- Rolldown, Biome and swc-node uses the `oxc_resolver` crate for module resolution
- Projects and companies like Preact, Shopify, ByteDance and Shopee uses oxlint for linting

How did it get started?

- Learning project
  - Rust
  - How to write a parser
  - Performance Engineering
- Wrote a JavaScript / Typescript parser from scratch
- And then a linter as an experiment
- … Kept going

What is Performance Engineering?

Massachusetts Institute of Technology

6.172, Fall 2018

Performance Engineering of Software Systems

Charles Leiserson

Lecture 1: Introduction and Matrix Multiplication

Software Properties

What software properties are more important than performance?

- Compatibility
- Correctness
- Clarity
- Debuggability
- Functionality
- Maintainability
- Modularity
- Portability
- Reliability
- Robustness
- Testability
- Usability

… and more.

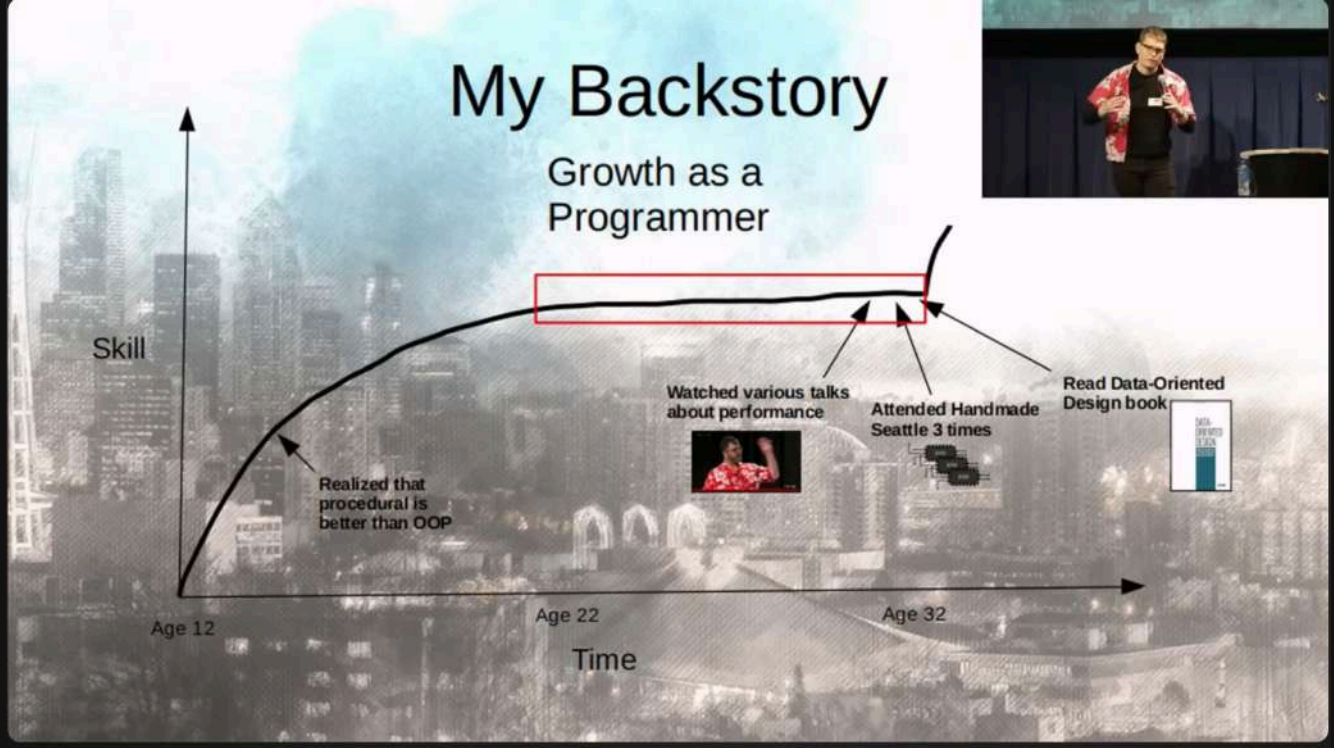
If programmers are willing to sacrifice performance for these properties, why study performance?

Performance is the currency of computing. You can often "buy" needed properties with performance.

Performance is used as a budget for buying more features.

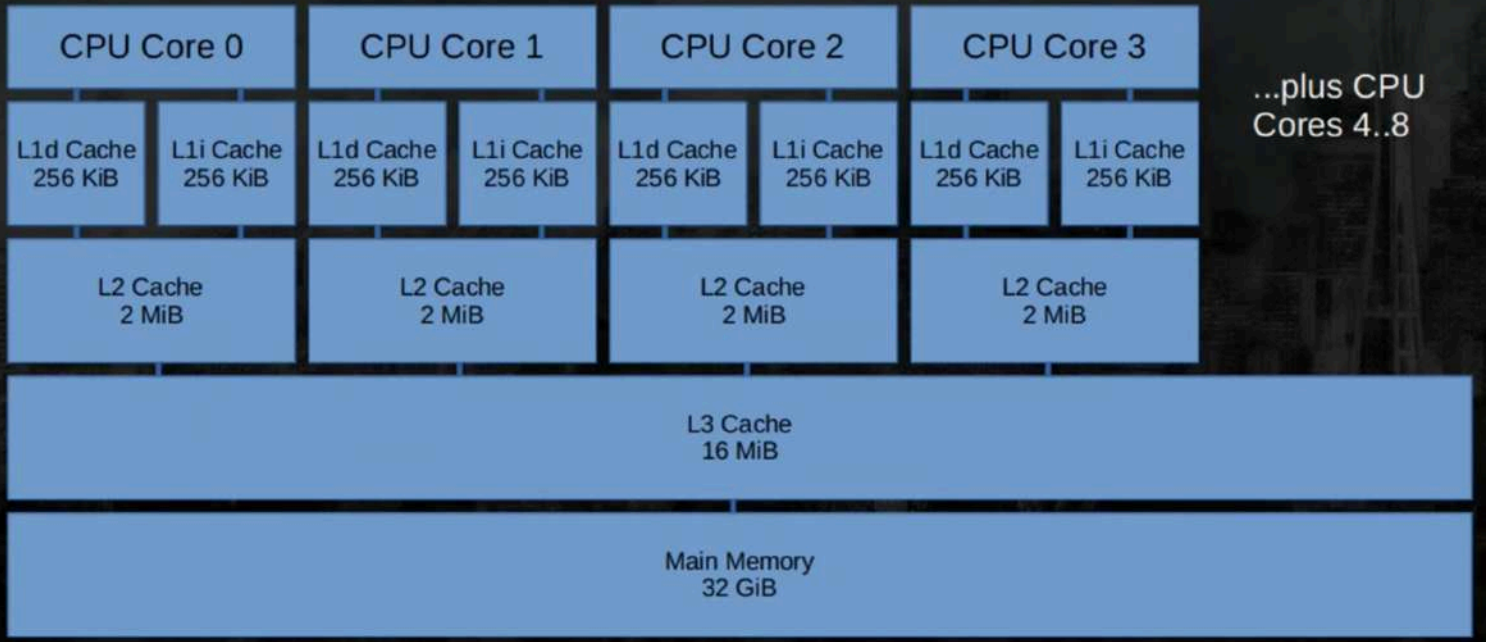
All performance issues are considered as bugs in this project.

Data Oriented Design



Andrew Kelley Practical Data Oriented Design (DoD)

The Big Picture



...plus CPU Cores 4..8

Not all CPU operations are created equal

ithare.com

Operation Cost in CPU Cycles	10^0	10^1	10^2	10^3	10^4	10^5	10^6
"Simple" register-register op (ADD,OR,etc.)	<1						
Memory write	~1						
Bypass delay: switch between integer and floating-point units	0-3						
"Right" branch of "if"	1-2						
Floating-point/vector addition	1-3						
Multiplication (integer/float/vector)	1-7						
Return error and check	1-7						
L1 read	3-4						
L2 read		10-12					
"Wrong" branch of "if" (branch misprediction)		10-20					
Floating-point division		10-40					
128-bit vector division		10-70					
C function direct call		15-30					
Integer division		15-40					
C function indirect call		20-50					
C++ virtual function call		30-60					
L3 read		30-70					
Main RAM read			100-150				
NUMA: different-socket L3 read			100-300				
Allocation+deallocation pair (small objects)			200-500				
NUMA: different-socket main RAM read			300-500				
Kernel call				1000-1500			
Thread context switch (direct costs)				2000			
C++ Exception thrown+caught				5000-10000			
Thread context switch (total costs, including cache invalidation)					10000 - 1 million		

Memory Allocation (sys call)

Distance which light travels while the operation is performed

CPU is fast

Memory is order of magnitude slower

Do more in CPU

Do less in memory

Data Oriented Design Strategy:

Find the most used data, make it smaller.

The Rust Programming Language

- Lots of unique features
  - Memory Safety
  - No garbage collector
  - "borrow checker"
  - Node.js N-API (NAPI)
- We are going to focus on:
  - Controlled memory management
  - CPU instructions optimized with LLVM
    - Rust Language -> IR (Intermediate Representation) -> LLVM -> CPU instructions

JavaScript

- Much harder to find data that are more frequently used
- Example:
  - https://github.com/microsoft/TypeScript/pull/58928
  - monomorphic node/type/signature
  - monomorphic - all object for a type has the same shape
  - a total time win of 16.0%
  - a checker time win of 20.8%

Rest of the talk

- Parser & AST
- Linter
- Transformer
- Minifier
- Bundler (Rolldown)

Parser

- 3x - 5x faster than swc
- 20x - 50x faster than Babel
- Passes all parser tests from Test262 and 99% from Babel and TypeScript

Parser - Most allocated object: Token

```
pub enum TokenKind {
    Number(f64), // 8 bytes
    String(String), // 24 bytes
} // 24 bytes * 1000 tokens = 24,000 bytes
```

to:

```
pub enum TokenKind {
    Number,
    String,
} // 1 byte * 1000 tokens = 1000 bytes
```

Parser - Most allocated object: Token

```
pub struct Token {
    start: u64, // 8 bytes
    end: u64, // 8 bytes
} // 16 bytes
```

- Max of u64 of is 18446744073709551615 = 18446744073 GB file
- Max of u32 of is 4294967295 = 4.2 GB file

```
pub struct Token {
    start: u32, // 4 bytes
    end: u32, // 4 bytes
} // 8 bytes
```

Parser

String

- `String` = heap allocation = slow
- `String` has 24 bytes (pointer, length, capacity 8 bytes each)
- uses stack memory - no heap allocation
- inline for strings that are smaller than 24 bytes
- https://crates.io/crates/compact_str

Parser

SIMD - Single instruction, multiple data

- A single instruction to perform 8 / 16 / 24 bytes at a time
- For JavaScript, only available in web assembly
- For finding the end of a multi-line comment
- `/* comment */`
  - 1 CPU instruction - `/* comem`
  - 1 CPU instruction - `ent */`

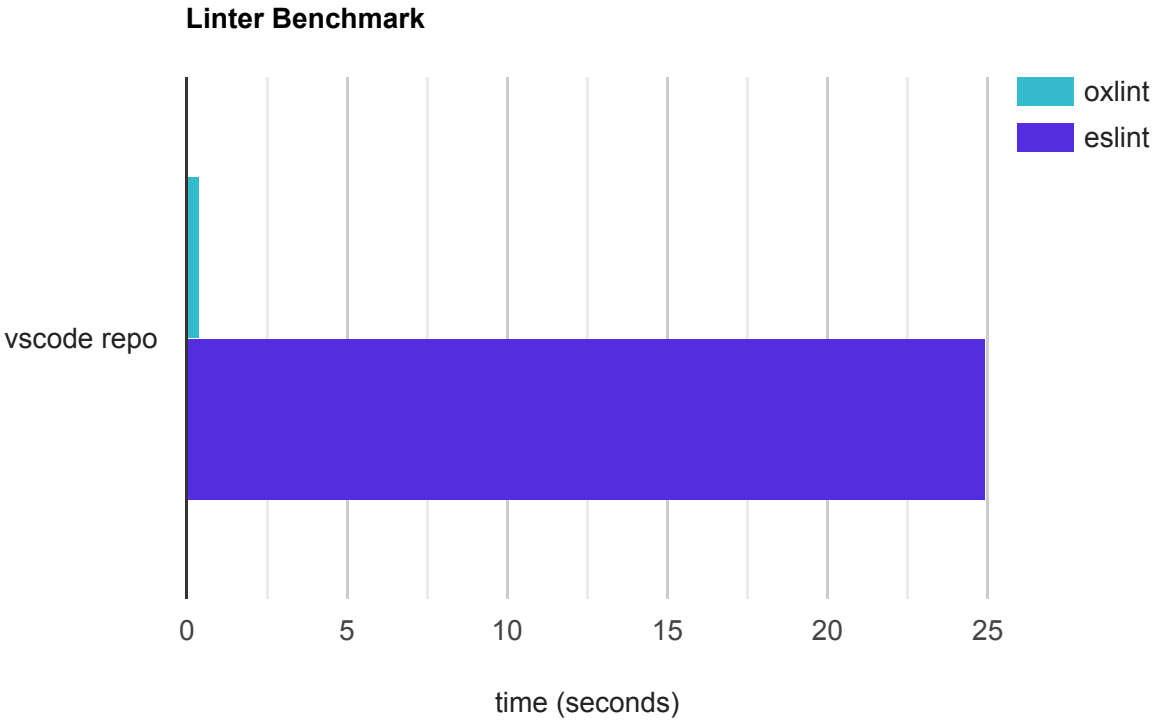
Parser

Memory Arena for AST

- In Rust default memory management, millions of AST nodes are allocated randomly on the heap
- Problem: Rust does not have garbage collection.
  - When releasing the memory, each AST node has to be released individually
- Solution: allocate in a continuous memory region, release together
- Oxc uses https://crates.io/crates/bumpalo
- AST visit becomes faster too, due to CPU cache lines
- 20% performance improvement

Oxlint

- `npx oxlint` in your project
- Designed to catch erroneous or useless code without requiring any configurations by default
- compatible with ESLint
- 50 - 100 times faster than ESLint, and scales with the number of CPU cores
- Over 430 rules with a growing list from popular ESLint plugins


Linter Benchmark
oxlint
eslint
vscode repo
0
5
10
15
20
25
time (seconds)

Boshen
@boshen_c

I went to the largest internal codebase in person, ran oxlint, and it completed 122,630 files in 3.4 seconds.

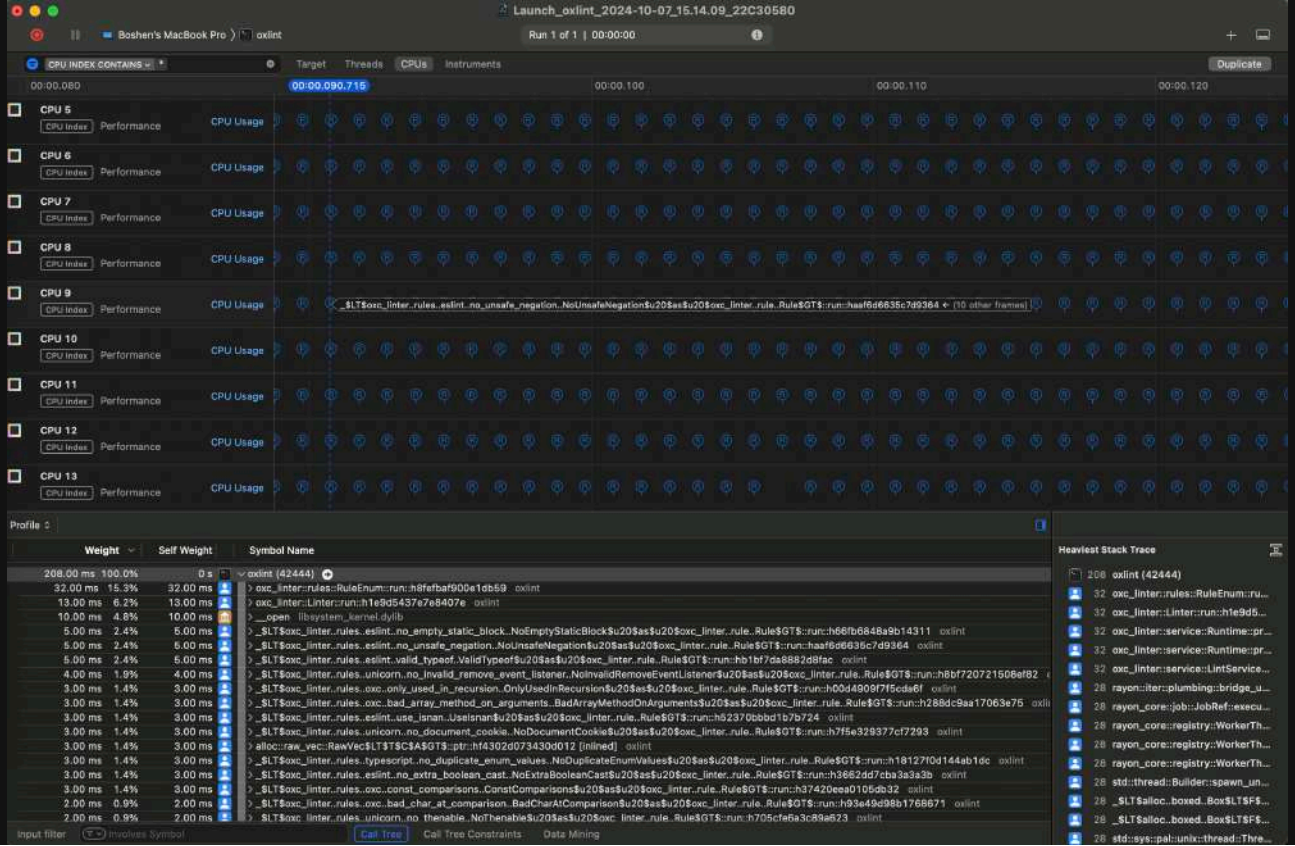
> Finished in 3.4s on 122,630 files with 92 rules using 24 threads.
> Found 23095 warnings and 57 errors.

Fixing 23,095 warnings is going to be hard, but they don't need to tinker with config files and wait forever,
they can start picking more important rules and fix things one by one, for example `oxlint -D no-constant-binary-expression`.

1:59 PM · Apr 24, 2024 · **6,655** Views

Oxlint - Parallelization

- run on multiple cores
- super easy in Rust
- use the `rayon` crate: `list.par_iter()`


Launch_oxlint_2024-10-07_15.14.09_22C30580
Boshen's MacBook Pro
oxlint
Run 1 of 1 | 00:00:00
CPU INDEX CONTAINS
Target
Threads
CPUs
Instruments
Duplicate
00:00.080
00:00.090.715
00:00.100
00:00.110
00:00.120
CPU 5
CPU 6
CPU 7
CPU 8
CPU 9
CPU 10
CPU 11
CPU 12
CPU 13
CPU Index
Performance
CPU Usage
_LToxc_linter..rules..eslint..no_unsafe_negation..NoUnsafeNegation$u20$as$u20$oxc_linter..rule..RuleGT::run::haaf6d6635c7d9364 ← (10 other frames)
Profile
Weight
Self Weight
Symbol Name
208.00 ms 100.0% 0 s oxlint (42444)
32.00 ms 15.3% 32.00 ms oxc_linter::rules::RuleEnum::run::h8fafbaf900e1db59 oxlint
13.00 ms 6.2% 13.00 ms oxc_linter::Linter::run::h1e9d5437e7e8407e oxlint
10.00 ms 4.8% 10.00 ms __open libsystem_kernel.dylib
5.00 ms 2.4% 5.00 ms _LToxc_linter..rules..eslint..no_empty_static_block..NoEmptyStaticBlock$u20$as$u20$oxc_linter..rule..RuleGT::run::h66fb684ba9b14311 oxlint
5.00 ms 2.4% 5.00 ms _LToxc_linter..rules..eslint..no_unsafe_negation..NoUnsafeNegation$u20$as$u20$oxc_linter..rule..RuleGT::run::haaf6d6635c7d9364 oxlint
5.00 ms 2.4% 5.00 ms _LToxc_linter..rules..eslint..valid_typeof..ValidTypeof$u20$as$u20$oxc_linter..rule..RuleGT::run::hb1bf7da8882d8fac oxlint
4.00 ms 1.9% 4.00 ms _LToxc_linter..rules..unicorn..no_invalid_remove_event_listener..NoInvalidRemoveEventListener$u20$as$u20$oxc_linter..rule..RuleGT::run::h8bf720721508ef82
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..oxc..only_used_in_recursion..OnlyUsedInRecursion$u20$as$u20$oxc_linter..rule..RuleGT::run::h00d4909f7f5cda6f oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..oxc..bad_array_method_on_arguments..BadArrayMethodOnArguments$u20$as$u20$oxc_linter..rule..RuleGT::run::h288dc9aa17063e75 oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..eslint..use_isnan..UseIsnan$u20$as$u20$oxc_linter..rule..RuleGT::run::h52370bbbd1b7b724 oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..unicorn..no_document_cookie..NoDocumentCookie$u20$as$u20$oxc_linter..rule..RuleGT::run::h7f5e329377cf7293 oxlint
3.00 ms 1.4% 3.00 ms alloc::raw_vec::RawVecLTTCAGT::ptr::hf4302d073430d012 [inlined] oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..typescript..no_duplicate_enum_values..NoDuplicateEnumValues$u20$as$u20$oxc_linter..rule..RuleGT::run::h18127f0d144ab1dc oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..eslint..no_extra_boolean_cast..NoExtraBooleanCast$u20$as$u20$oxc_linter..rule..RuleGT::run::h3662dd7cba3a3a3b oxlint
3.00 ms 1.4% 3.00 ms _LToxc_linter..rules..oxc..const_comparisons..ConstComparisons$u20$as$u20$oxc_linter..rule..RuleGT::run::h37420eea0105db32 oxlint
2.00 ms 0.9% 2.00 ms _LToxc_linter..rules..oxc..bad_char_at_comparison..BadCharAtComparison$u20$as$u20$oxc_linter..rule..RuleGT::run::h93e49d98b1768671 oxlint
2.00 ms 0.9% 2.00 ms _LToxc_linter..rules..unicorn..no_thenable..NoThenable$u20$as$u20$oxc_linter..rule..RuleGT::run::h705cfe6a3c89a823 oxlint
Input filter
Involves Symbol
Call Tree
Call Tree Constraints
Data Mining
Heaviest Stack Trace
208 oxlint (42444)
32 oxc_linter::rules::RuleEnum::ru...
32 oxc_linter::Linter::run::h1e9d5...
32 oxc_linter::service::Runtime::pr...
32 oxc_linter::service::Runtime::pr...
32 oxc_linter::service::LintService...
28 rayon::iter::plumbing::bridge_u...
28 rayon_core::job::JobRef::execu...
28 rayon_core::registry::WorkerTh...
28 rayon_core::registry::WorkerTh...
28 rayon_core::registry::WorkerTh...
28 std::thread::Builder::spawn_un...
28 _LTalloc..boxed..BoxLTF$...
28 _LTalloc..boxed..BoxLTF$...
28 std::sys::pal::unix::thread::Thre...

Transformer

- Transform TypeScript and JSX to ESNext
- TypeScript Isolated Declarations `.d.ts` Emit without using the TypeScript compiler `tsc --isolatedDeclarations`

Transformer Performance

- 4x faster than swc, 40x than Babel
- Babel and swc: multiple AST pass
- Challenge for Oxc to make it fast - Single AST Pass

Minifier (Prototype)

- Three components
  - White space remover
  - Symbol mangler
  - Compressor
- Aim to have the best compression size
- Similar Google Closure Compiler https://github.com/google/closure-compiler
  - Lots of AST passes, run in a loop
  - Written in Java
- The only time where we are not going to aim for extreme performance

Minification Benchmark

Shout to @privatenumber!

Artifact	Original size	Gzip size
typescript v4.9.5 (Source)	10.95 MB	1.88 MB

Minifier	Minified size	Minzipped size	Time
1. @swc/core	🏆 -70% 3.31 MB	🏆 -55% 851.73 kB	5x 1,067 ms
2. terser	-69% 3.35 MB	-55% 854.26 kB	26x 5,251 ms
3. esbuild	-68% 3.49 MB	-51% 915.50 kB	1x 263 ms

Oxc Test Infrastructure

Layers of test infrastructure for correctness and reliability

- Conformance suite on Test262, Babel, TypeScript
- Lots of fuzzing
- Linter snapshot diagnostics
- oxlint ecosystem ci
- Idempotency testing
- Code coverage
- End to end 3000 top npm packages
- https://oxc.rs/docs/learn/architecture/test.html

Bundler (Rolldown)

- Rolldown + Oxc is going to serve as the future JavaScript bundler for Vite
- Rust core but allow JavaScript plugins
- Aim to remove the dev-prod inconsistency issue
- Aim to align with Rollup's API as much as possible
- A lot more performance budget for new features

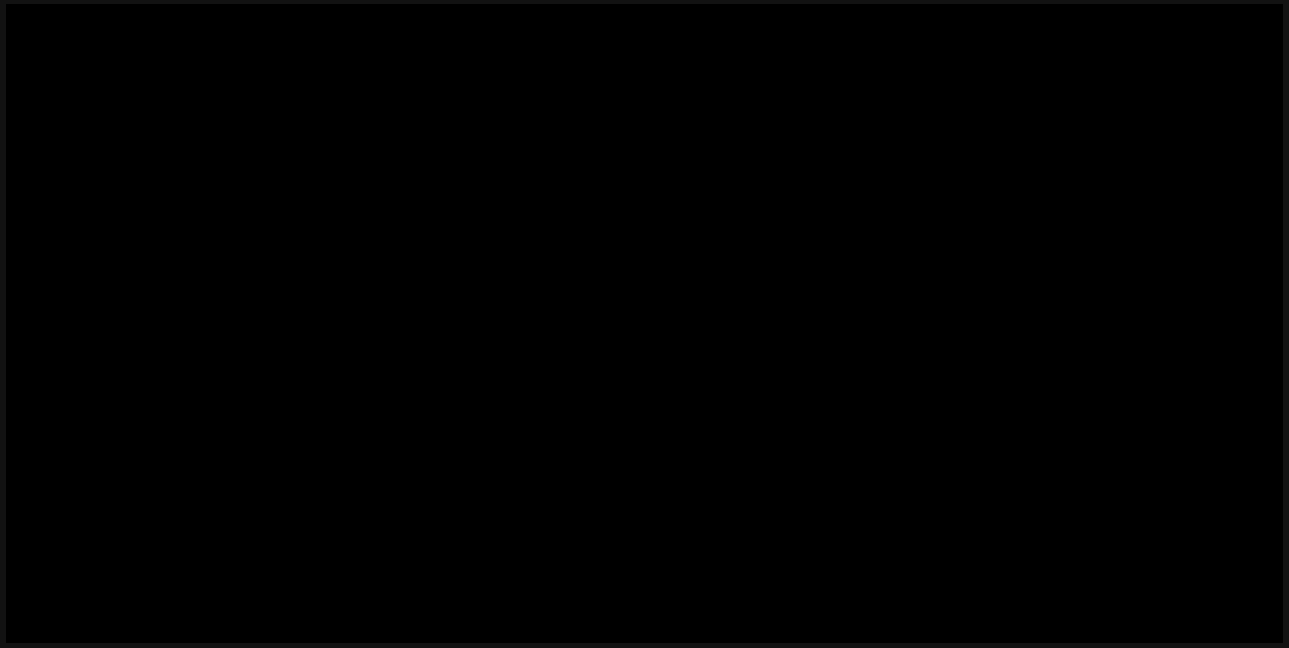
Rolldown Performance

- github.com/oxc-project/monitor-oxc
- 3000 most popular npm packages, from npm-high-impact:
  - download count of 1 million or more per week
  - depended on by 500 or more other packages

Rolldown Demo

All 3000 packages statically imported into a single file

- Single chunk with lots of code
- Processed 46k files
- Generated 809 files, 200MB entry file
- 3.26s
- 7GB Max RSS (Resident set size)
  - No more `max-old-space-size=8196`



Rust

Should I learn Rust? Yes if

- you like performance engineering, want things fast, hate to wait
- you need the strictness offered by Rust
- your infrastructure has a scaling problem
- your tooling does not work with so much code
- your node.js app has a component that can be sped up by Rust

Rust for tooling is already here.

We expect most tooling infrastructure written in Rust by the end of next year.

JavaScript for web development will stay.

Learn more

- oxc.rs
- rolldown.rs
- We appreciate any kind of contribution ❤️

Thank You.