



Biome

TOOLCHAIN OF THE WEB


Victorien Elvinger

@Conaclos

Biome lead maintainer

What is Biome?

A code **linter**

- JavaScript, **TypeScript**, **JSX**, TSX
 - no extra dependencies
- **Helpful diagnostics**
- **200 lint rules**
 - some unique to Biome
 - ESLint, ESLint plugins
 -  Tailwind class sorting

➤ `npx @biomejs/biome lint main.ts`

`main.ts:4:18 lint/noAccumulatingSpread`


✗ Avoid the use of spread `...` syntax on accumulators.

```
3 | array.reduce(  
> 4 |   (acc, v) => [...acc, v],  
   |                      ^^^^^^^  
5 |   []  
6 | )
```

i Spread syntax should be avoided on accumulators because it causes a time complexity of `O(n2)`.

i Consider methods such as `.splice` or `.push` instead.

A code formatter

- JavaScript, TypeScript, JSX, TSX
- JSON, JSONC
-  CSS
- **format invalid code**

```
TS main.ts 2 ●
TS main.ts > Person > constructor
1  export class Person {
2      #name: string
3
4      constructor() {
5          this.#name =
6      }
7
8  get name() { return this.#name }
9  }
10
```



vjeux

@Vjeux



There's lot of excitement around faster pretty printers using Rust. The main issue is that none of them match the long tail of formatting logic of prettier.

I'm putting up a \$10k bounty for any project written in Rust that passes > 95% of the prettier JavaScript tests.

\$10,000 Bounty

10:50 PM · Nov 9, 2023



1K



Reply

Write a pretty printer in Rust

Win \$25,000

Grand Prize

\$22,500

Pass > 95% of the prettier JavaScript tests

[READ THE ANNOUNCEMENT →](#)

WASIX Prize

\$2,500

Compile to WASIX and publish (via CI) to Wasmer

[READ THE ANNOUNCEMENT →](#)





Biome

@biomejs



Biome formatter has reached 97% compatibility with Prettier in JavaScript formatting. 🚀

5:38 PM · Dec 22, 2023



850



Reply

Is Biome fast?



fforres

@fforres



Holy shit.

Just did a quick parsing test of [@biomejs](#) in one of our [@OpenAI](#) codebases

- 3 eslint + prettier = 58.81 s
- 2 eslint (w/ cache) + prettier = 12.82 s
- 1 🤯 biomeJS check (first run) = 1.78 s

(Tried tweaking the config, and was able to get a 95% config parity in ~20m) 🤯

9:45 PM · Dec 11, 2023



550



Reply

A community

- 📦 170k weekly downloads
- ★ 8.4k GitHub Stars
- 🐦 4.6k followers
- 💬 1.2k Discord members



時雨堂

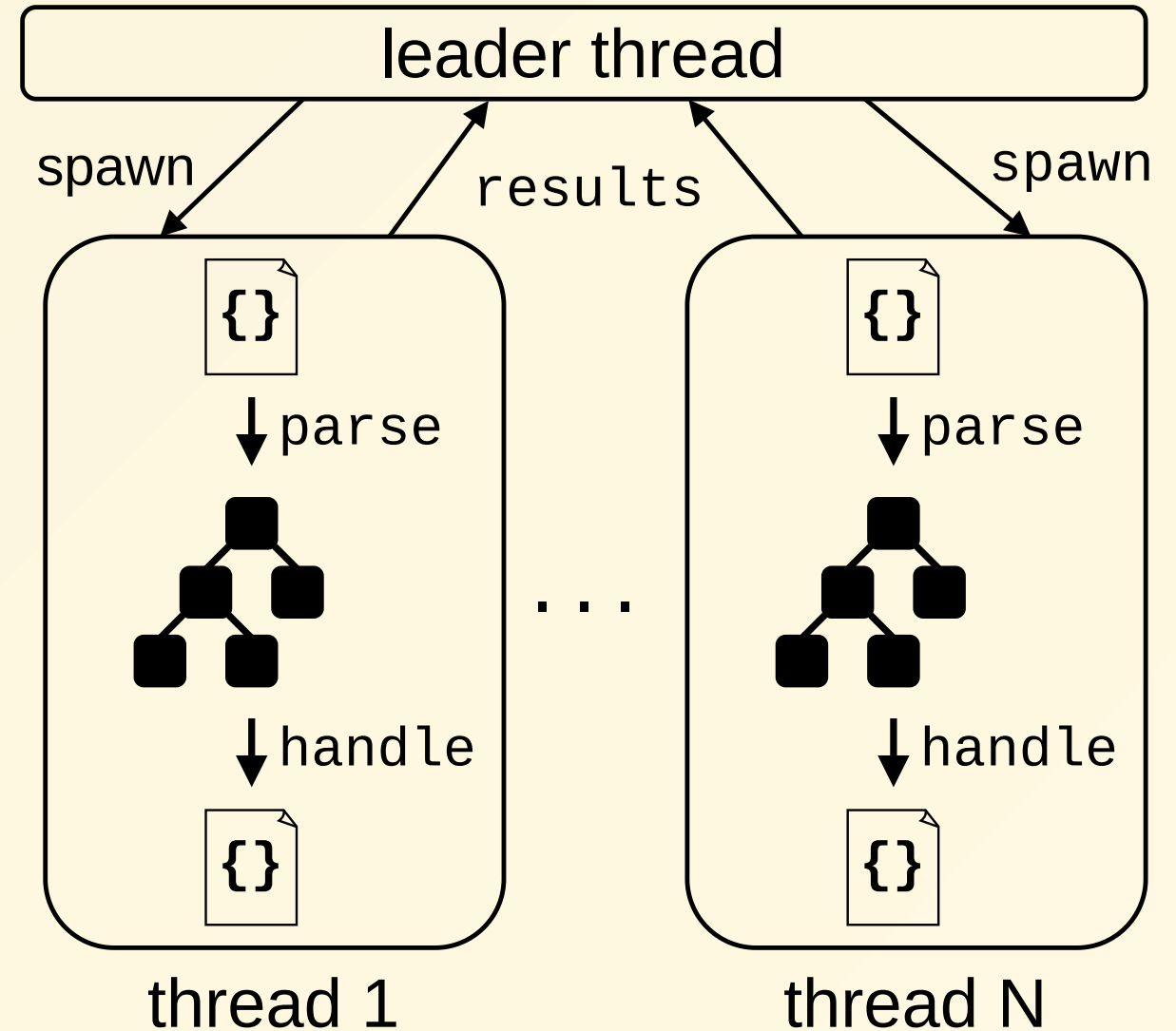
Shiguredo



How Biome works?

Architecture

- **leader-follower**
- the leader thread
 - spawn a thread per file
 - collect results
- a follower thread
 - parse the given file
 - handle (format, lint)

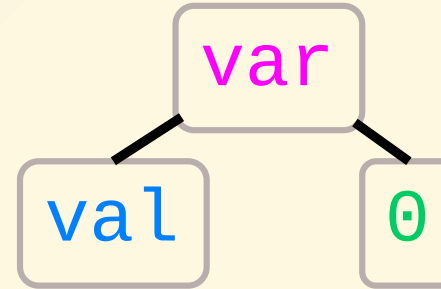


Regular parser

1. parse to an Abstract Syntax Tree
2. handle (format, lint)

```
· · var · val · = · 0
```

↓ parse



Abstract
Syntax
Tree

↓ lint/noVar

✗ Don't use var

```
> 1 | · · var · val · = · 0
    |   ^^^
```

i . . .

Regular parser

- doesn't handle invalid code
 - **emit syntax error**

· · var · val · = ·

↓ parse

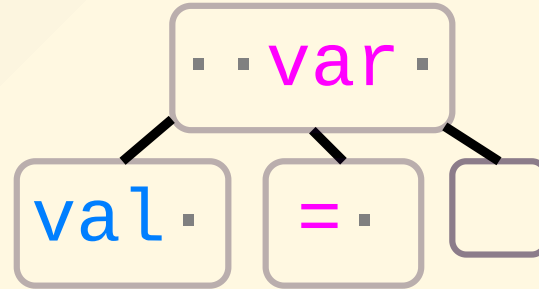
✗ syntax error

Biome parser

- **accept invalid code**
 - bogus tree nodes
 - holes in the tree
- **lossless** parsing using CST
 - preserve whitespace

```
· · var · val · = ·
```

↓ parse



**Concrete
Syntax
Tree**

↓ lint/noVar

✘ Don't use var

```
> 1 | · · var · val · = ·  
    |   ^^^
```

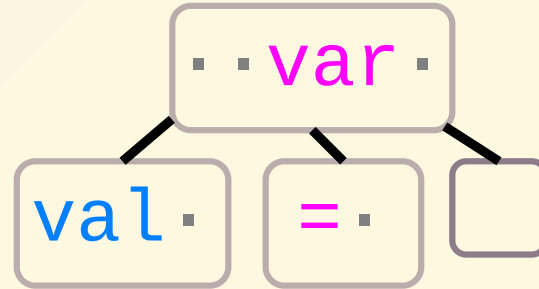
i . . .

Biome parser

- **accept invalid code**
 - bogus tree nodes
 - holes in the tree
- **lossless** parsing using CST
 - preserve whitespace

```
· · var · val · = ·
```

↓ parse



**Concrete
Syntax
Tree**

↓ format & fix

```
let · val · =
```

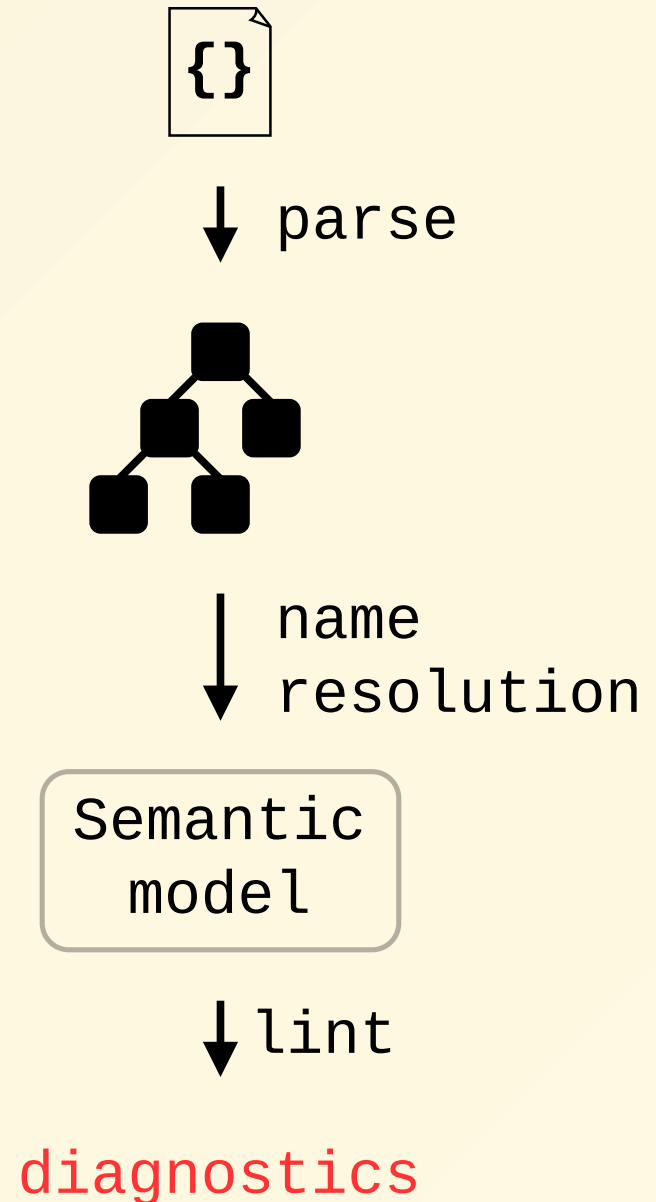

Lint rules

- many rules query the tree
 - noVar
 - noDoubleEquals
 - noAccumulatingSpread
- others need more complex data
 - **noUnusedVariables**
 - noUnusedImports
 - useImportType
 - useExportType

```
function Person (name ) {  
    if (name == "") {  
        let name  
        name = "anonymous"  
    }  
    return { name }  
}  
  
export { Person }
```

Semantic model

- find references of a declaration
 - write references
 - read references



Name resolver v1

- bind declarations to references
 - unique id for each declaration
 - a reference refers to a single declaration
- take scopes into account
 - variable shadowing

```
function Person1(name2) {  
  if (name2 == "") {  
    let name3  
    name3 = "anonymous"  
  }  
  return { name2 }  
}  
  
export { Person1 }
```

TypeScript 🤯

- **type & variable** with **same name**

```
interface Person0 {  
  name: string  
}  
  
function Person1(name): Person1 {  
  return { name2 }  
}  
  
export { Person1 }
```

The diagram illustrates a TypeScript error where a type and a variable share the same name. The code defines an interface `Person0` with a `name` property of type `string`. It then defines a function `Person1` that takes a `name` parameter and returns an object with a `name2` property. The function is exported as `Person1`. The red 'X' indicates the conflict between the interface name and the function name. The arrows show the relationships between the names in the code.

TypeScript 🤯

- type & variable with same name
- **a reference** refers to **multiple declarations**

```
interface Person {  
  name: string  
}  
  
function Person (name): Person {  
  return { name }  
}  
  
export { Person }
```

The diagram illustrates how a single reference 'Person' can point to multiple declarations in TypeScript. Three curved arrows originate from the 'Person' identifier in different contexts: 1) The arrow from the 'Person' in the function signature 'function Person' points to the 'Person' in the 'interface Person' declaration. 2) The arrow from the 'Person' in the return type ': Person' points to the 'Person' in the 'interface Person' declaration. 3) The arrow from the 'Person' in the export 'export { Person }' points to the 'Person' in the 'interface Person' declaration. This demonstrates that the 'Person' reference is resolved to the 'interface Person' declaration, despite the presence of a function with the same name.

TypeScript 🤯🤯

- type & variable with same name
- **a reference** refers to **multiple declarations**

```
type Element<D> =  
  D extends  
    Array<infer E>  
    | Set<infer E>  
    | infer E  
  ? E : never
```

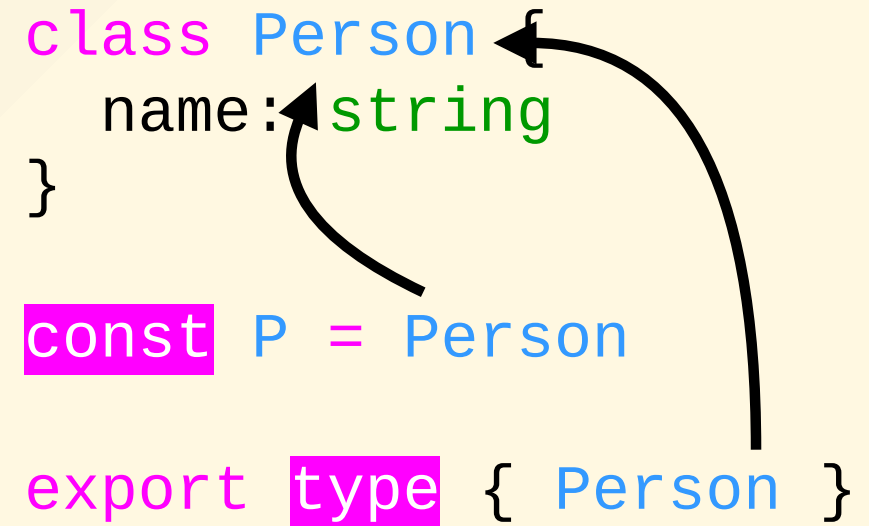
The diagram illustrates the recursive inference of the type `E` within the `Element` type definition. It features three curved arrows that represent the flow of inference:

- A top arrow originates from the `infer E` inside the `Array` type and points to the `E` in the `? E` declaration.
- A middle arrow originates from the `infer E` inside the `Set` type and points to the `E` in the `? E` declaration.
- A bottom arrow originates from the `infer E` inside the `infer E` type and points to the `E` in the `? E` declaration.

TypeScript 🧨

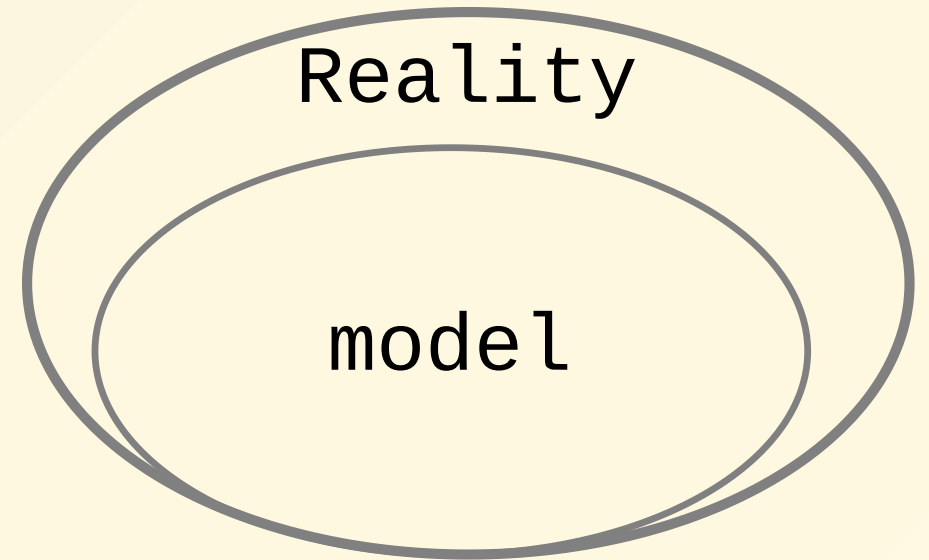
- type & variable with same name
- a reference refers multiple declarations
- **partially referenced declarations**
 - type / variable duality

```
class Person {  
  name: string  
}  
  
const P = Person  
  
export type { Person }
```



Simplification

- a reference refers to a **single declaration**
 - handle differently edge cases (*export, infer*)
- type and value with same names
- type / variable duality



Name resolver v2

- A declaration is either
 - a type
 - a variable
 - both
- A reference refers either
 - a type
 - a variable

```
interface Persont0 {  
    name: string  
}
```

```
function Personv1(namev2): Persont0 {  
    return { namev2 }  
}
```

```
export type { Persont0 }
```

Name resolver v2

- A declaration is either
 - a type
 - a variable
 - both
- A reference refers either
 - a type
 - a variable
- type/value duality not exposed

```
interface Person0 {  
    name: string  
}
```


```
function Person1(name2): Person0 {  
    return { name2 }  
}
```

```
export type { Person0 }
```






Conclusion

- Biome is both a **formatter** and a **linter**
 - and more: JavaScript **import sorting**
- Biome is **fast**
- Biome is **editor-ready**
 - error-resilient parsers
 - Concrete Syntax Tree
- Biome **supports TypeScript**
 - type-aware semantic model

2024 and beyond

- extend to **more languages**
 - CSS, HTML, Markdown
 - Vue, Angular, Svelte, Astro
- **improve linter capabilities**
 - multi-file analysis
 - simplified type system
-  **plugins**

Want to help?

-  try Biome
 -  report issues
 -  feedbacks
- contribute to Biome
 - [GitHub good first issues](#)
 -  [How to create a lint rule in Biome](#)
(youtube.com/@Biomejs)
-  sponsor us!
 - [Biome Open Collective](#)



biomejs.dev

format code

`npx @biomejs/biome format --write src`

lint code, apply safe fixes

`npx @biomejs/biome lint --apply src`

all at once

`npx @biomejs/biome check --apply src`

Backup slides

A toolchain

-  toolchain for **web dev**
 - code formatter
 - code linter
- written in **Rust** 
- supports main web language
 - **JavaScript, TypeScript, JSX, TSX**
 - JSON, JSONC
 - CSS 
- **community successor** of Rome Tools

A fast formatter

- scales with available threads



400 ms

35x



Biome



14 s



Prettier

Formatting **170k lines** of code in **2.1k files** with an **Intel Core i7 1270P**

A governance

- leads (2) 🔑 owners
 - 🛡️ access to sensible data
 - ⚔️ act as tiebreakers
- core contributors (5)
 - 👍 project directions
- maintainers (5)
 - 👍 project decisions
 - 📶 write access to the repo

