RUFF

An extremely fast Python linter, written in Rust.

VERSION: v0.0.292

ASTRAL

CHARLIE MARSH

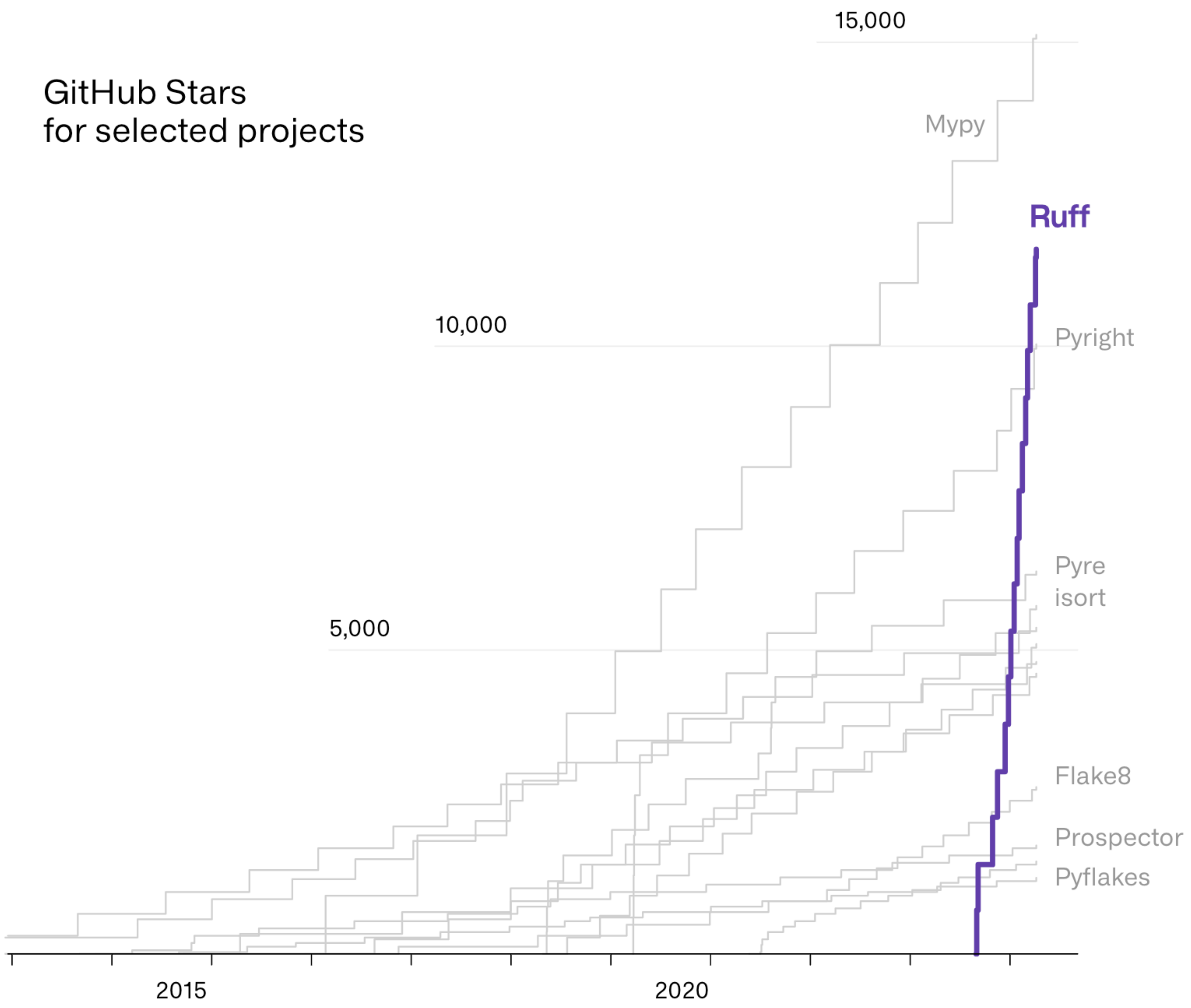
PYTEXAS

What is Ruff?

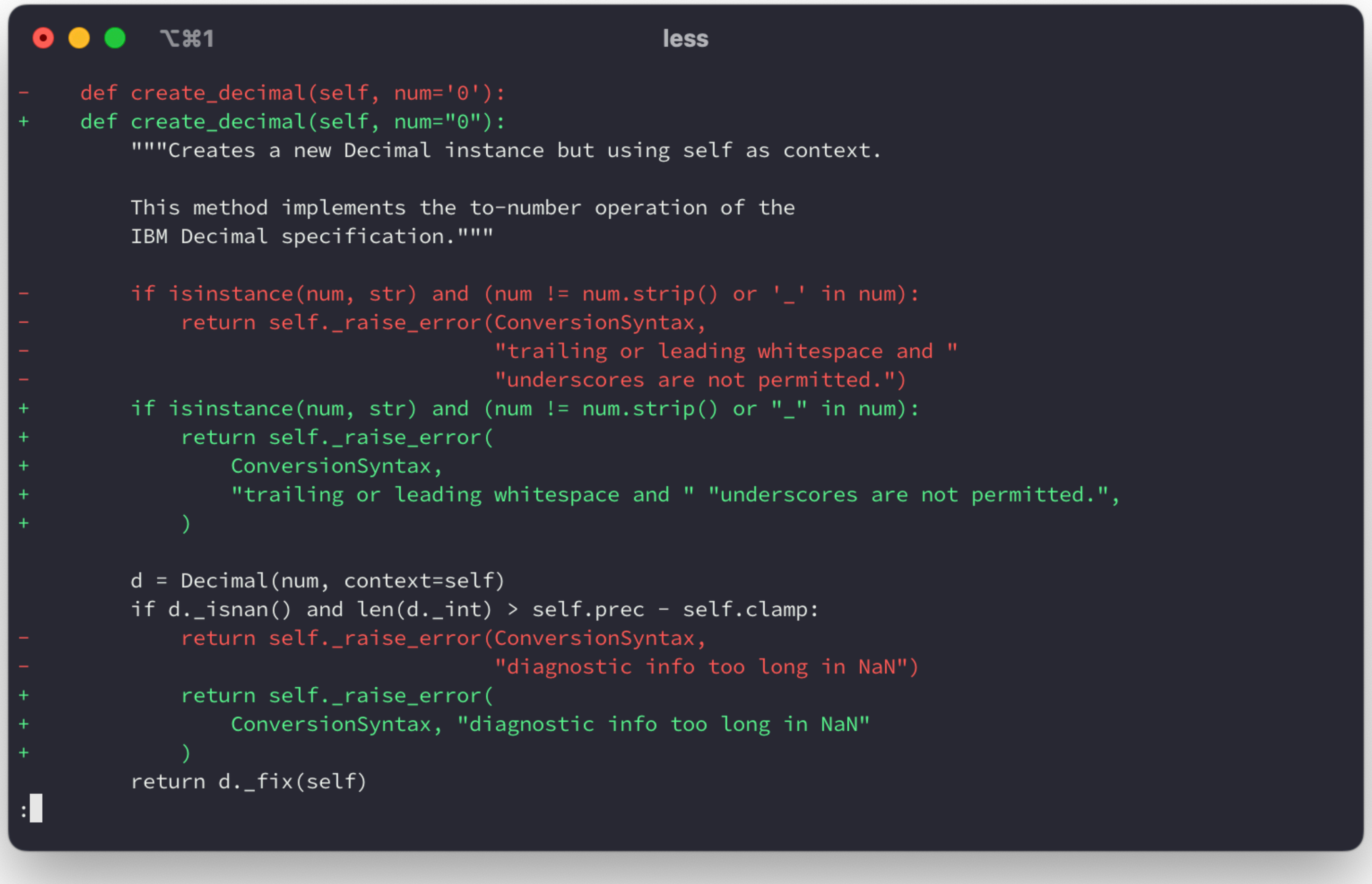
- “An extremely fast Python linter written in Rust”

- Used by… Amazon, Apache Airflow, Databricks, FastAPI, Hugging Face, Jupyter, Microsoft, Mozilla, Mypy, Netflix, Pandas, Poetry, Polars, PyTorch, Pydantic, Snowflake, SciPy, Zulip, pip, etc.

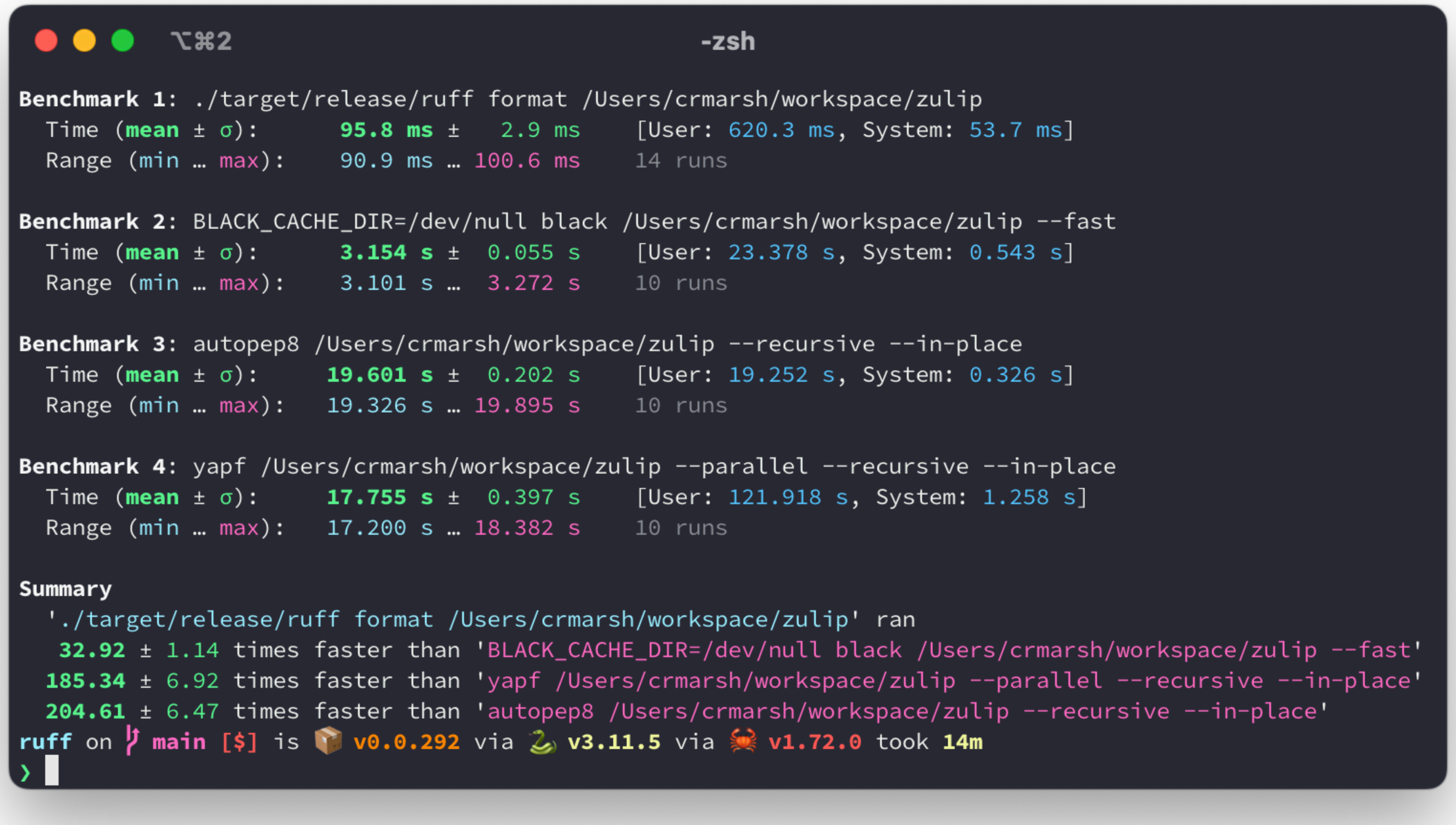
- The first tool in a **toolchain**



Ruff is (also) a formatter



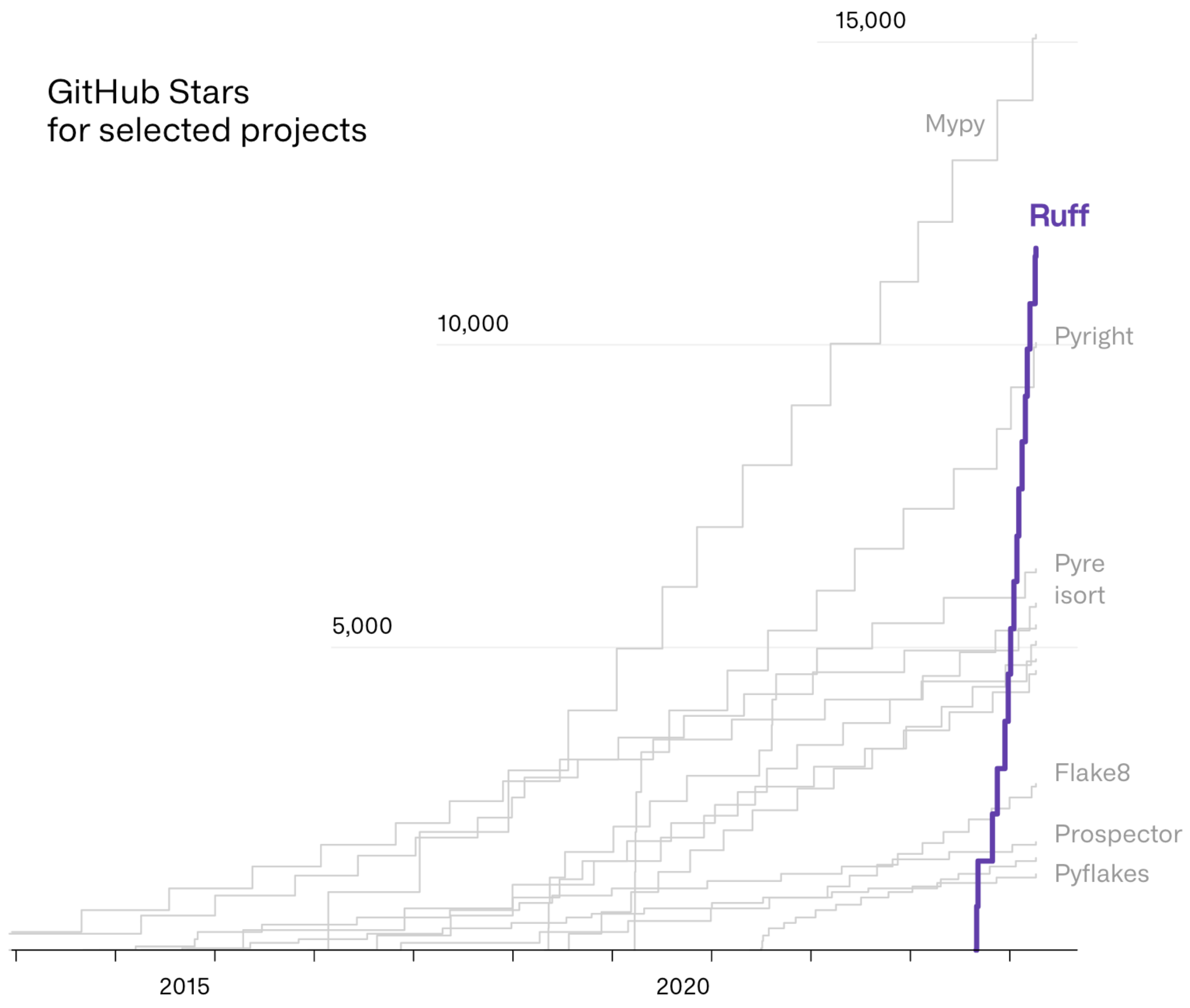
```
-    def create_decimal(self, num='0'):
+    def create_decimal(self, num="0"):
         """Creates a new Decimal instance but using self as context.

         This method implements the to-number operation of the
         IBM Decimal specification."""

-        if isinstance(num, str) and (num != num.strip() or '_' in num):
-            return self._raise_error(ConversionSyntax,
-                                     "trailing or leading whitespace and "
-                                     "underscores are not permitted.")
+        if isinstance(num, str) and (num != num.strip() or "_" in num):
+            return self._raise_error(
+                ConversionSyntax,
+                "trailing or leading whitespace and " "underscores are not permitted.",
+            )

         d = Decimal(num, context=self)
         if d._isnan() and len(d._int) > self.prec - self.clamp:
-            return self._raise_error(ConversionSyntax,
-                                     "diagnostic info too long in NaN")
+            return self._raise_error(
+                ConversionSyntax, "diagnostic info too long in NaN"
+            )
         return d._fix(self)
:
```



```
Benchmark 1: ./target/release/ruff format /Users/crmarsh/workspace/zulip
  Time (mean ± σ):      95.8 ms ±   2.9 ms    [User: 620.3 ms, System: 53.7 ms]
  Range (min … max):    90.9 ms … 100.6 ms    14 runs

Benchmark 2: BLACK_CACHE_DIR=/dev/null black /Users/crmarsh/workspace/zulip --fast
  Time (mean ± σ):      3.154 s ±  0.055 s    [User: 23.378 s, System: 0.543 s]
  Range (min … max):    3.101 s …  3.272 s    10 runs

Benchmark 3: autopep8 /Users/crmarsh/workspace/zulip --recursive --in-place
  Time (mean ± σ):     19.601 s ±  0.202 s    [User: 19.252 s, System: 0.326 s]
  Range (min … max):   19.326 s … 19.895 s    10 runs

Benchmark 4: yapf /Users/crmarsh/workspace/zulip --parallel --recursive --in-place
  Time (mean ± σ):     17.755 s ±  0.397 s    [User: 121.918 s, System: 1.258 s]
  Range (min … max):   17.200 s … 18.382 s    10 runs

Summary
  './target/release/ruff format /Users/crmarsh/workspace/zulip' ran
   32.92 ± 1.14 times faster than 'BLACK_CACHE_DIR=/dev/null black /Users/crmarsh/workspace/zulip --fast'
  185.34 ± 6.92 times faster than 'yapf /Users/crmarsh/workspace/zulip --parallel --recursive --in-place'
  204.61 ± 6.47 times faster than 'autopep8 /Users/crmarsh/workspace/zulip --recursive --in-place'
ruff on  main [$] is 📦 v0.0.292 via 🐍 v3.11.5 via 🦀 v1.72.0 took 14m
❯
```

What is Ruff?

- “An extremely fast Python linter written in Rust”
- Used by… Amazon, Apache Airflow, Databricks, FastAPI, Hugging Face, Jupyter, Microsoft, Mozilla, Mypy, Netflix, Pandas, Poetry, Polars, PyTorch, Pydantic, Snowflake, SciPy, Zulip, pip, etc.
- The first tool in a **toolchain**



Where did Ruff come from?

- Khan Academy (2015 - 2017)
    - Web frontend (JavaScript)
    - Web backend (Python)
    - Android (Java)
    - iOS (Objective-C, Swift)
- Spring Discovery (2018 - 2022)
    - Machine learning infrastructure (Python)
    - Data infrastructure (Python, Rust)
    - Web frontend (TypeScript)
- Ruff (August 2022)
- Astral (March 2023)

What is Ruff?

~~What is Ruff?~~

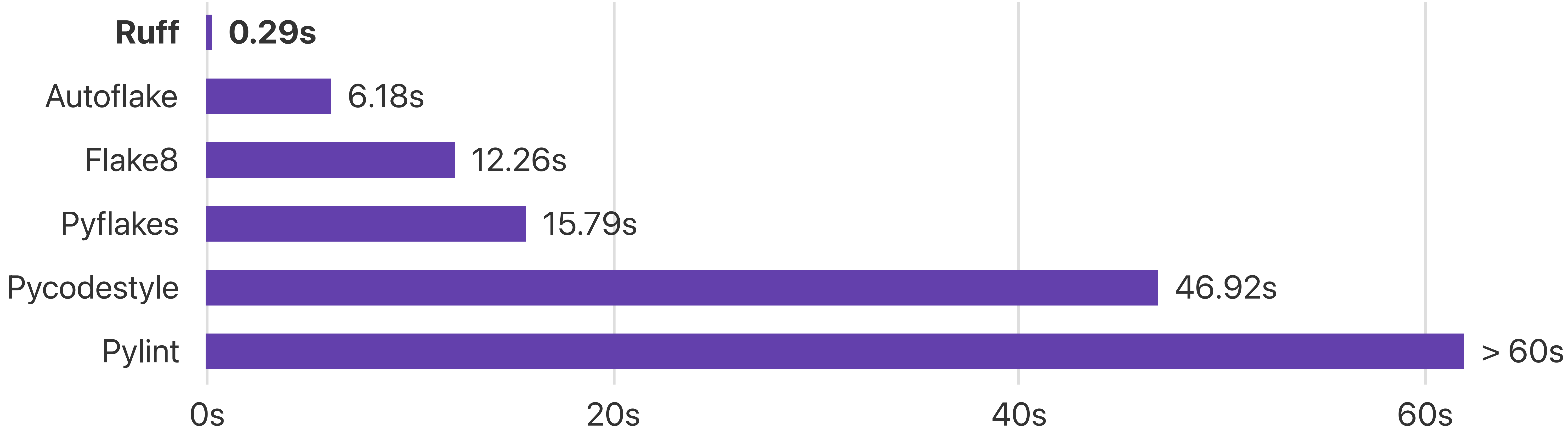
Why do people like Ruff?

Why do people like Ruff?

1. **Performant: 10-1000x faster than existing Python linters**

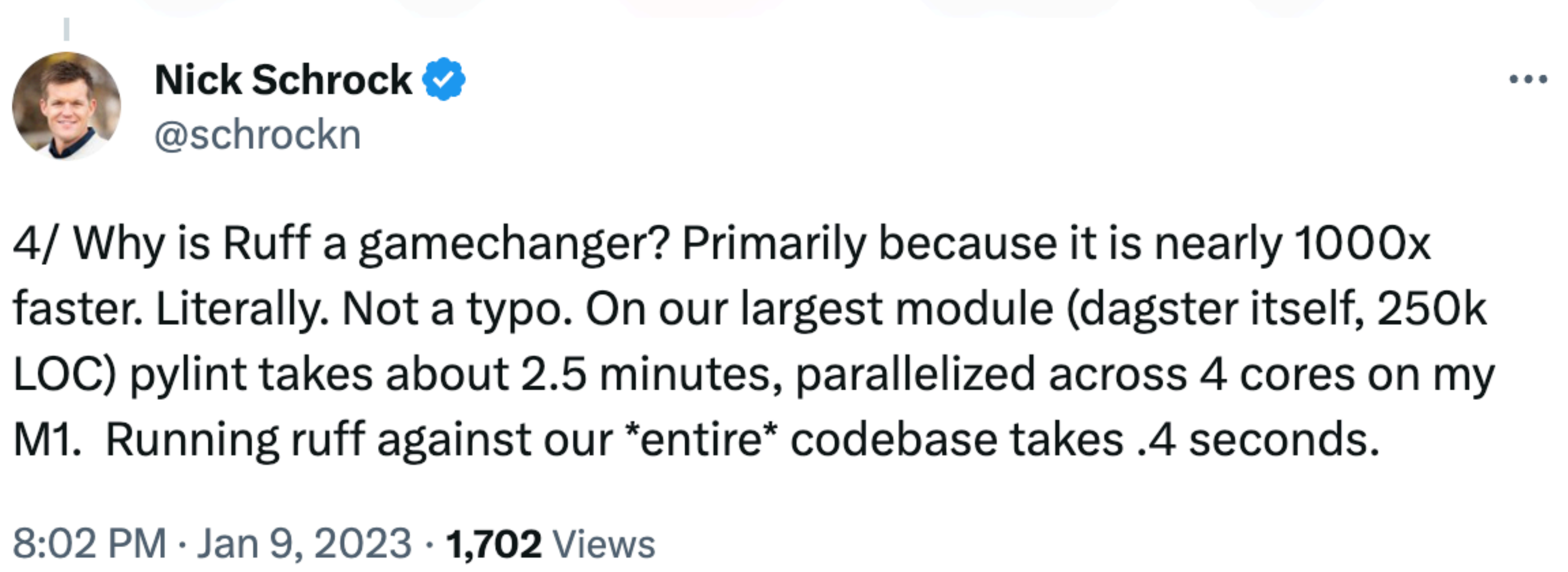
Why do people like Ruff?

1. **Performant: 10-1000x faster than existing Python linters**



Why do people like Ruff?

1. **Performant: 10-1000x faster than existing Python linters**



Nick Schrock
@schrockn

4/ Why is Ruff a gamechanger? Primarily because it is nearly 1000x faster. Literally. Not a typo. On our largest module (dagster itself, 250k LOC) pylint takes about 2.5 minutes, parallelized across 4 cores on my M1. Running ruff against our *entire* codebase takes .4 seconds.

8:02 PM · Jan 9, 2023 · **1,702** Views

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

Why do people like Ruff?

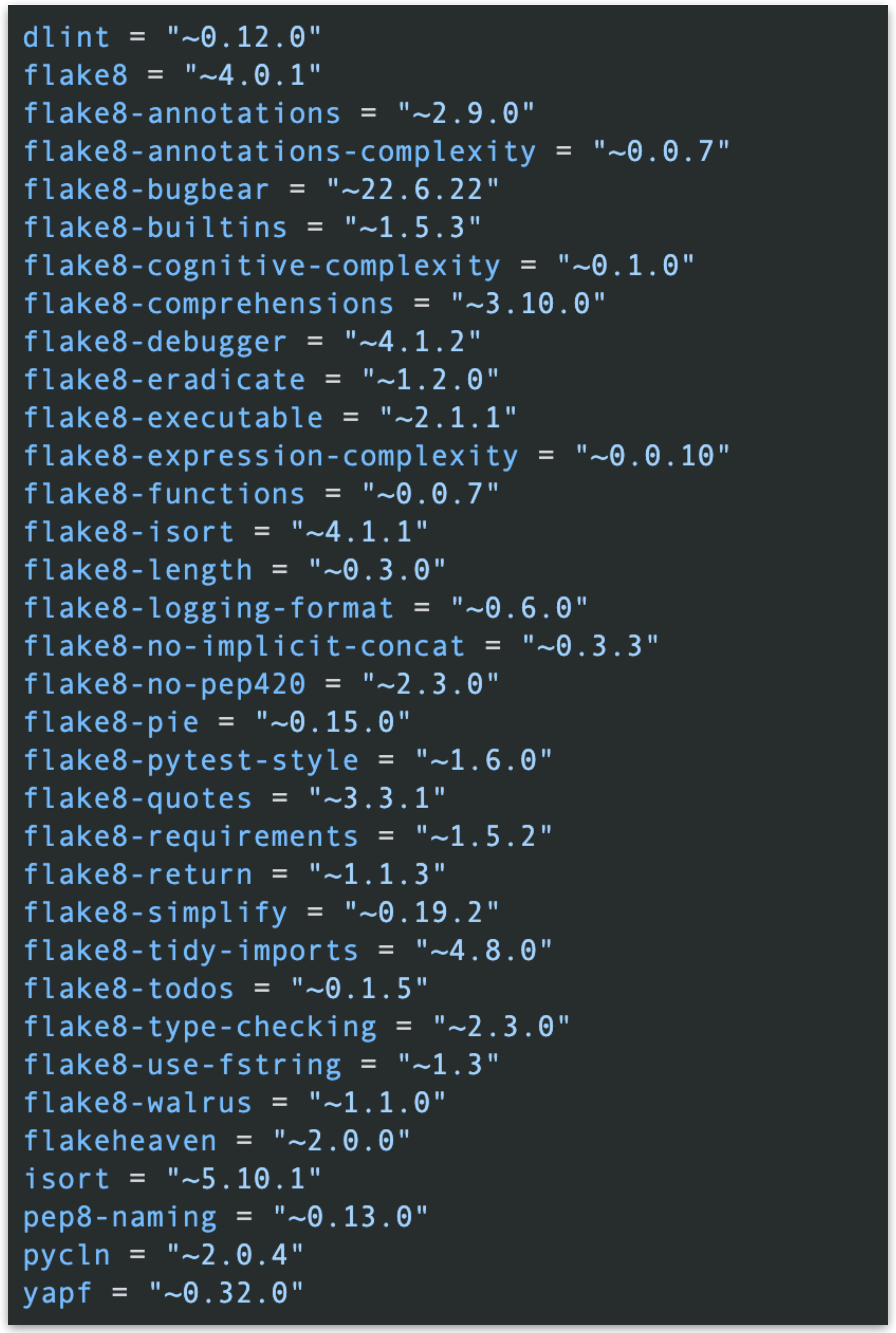
1. Performant: 10-1000x faster than existing Python linters

2. **Unified: replace dozens of tools with a single interface**

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

2. **Unified: replace dozens of tools with a single interface**



```
dlint = "~0.12.0"
flake8 = "~4.0.1"
flake8-annotations = "~2.9.0"
flake8-annotations-complexity = "~0.0.7"
flake8-bugbear = "~22.6.22"
flake8-builtins = "~1.5.3"
flake8-cognitive-complexity = "~0.1.0"
flake8-comprehensions = "~3.10.0"
flake8-debugger = "~4.1.2"
flake8-eradicate = "~1.2.0"
flake8-executable = "~2.1.1"
flake8-expression-complexity = "~0.0.10"
flake8-functions = "~0.0.7"
flake8-isort = "~4.1.1"
flake8-length = "~0.3.0"
flake8-logging-format = "~0.6.0"
flake8-no-implicit-concat = "~0.3.3"
flake8-no-pep420 = "~2.3.0"
flake8-pie = "~0.15.0"
flake8-pytest-style = "~1.6.0"
flake8-quotes = "~3.3.1"
flake8-requirements = "~1.5.2"
flake8-return = "~1.1.3"
flake8-simplify = "~0.19.2"
flake8-tidy-imports = "~4.8.0"
flake8-todos = "~0.1.5"
flake8-type-checking = "~2.3.0"
flake8-use-fstring = "~1.3"
flake8-walrus = "~1.1.0"
flakeheaven = "~2.0.0"
isort = "~5.10.1"
pep8-naming = "~0.13.0"
pycln = "~2.0.4"
yapf = "~0.32.0"
```

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

2. Unified: replace dozens of tools with a single interface

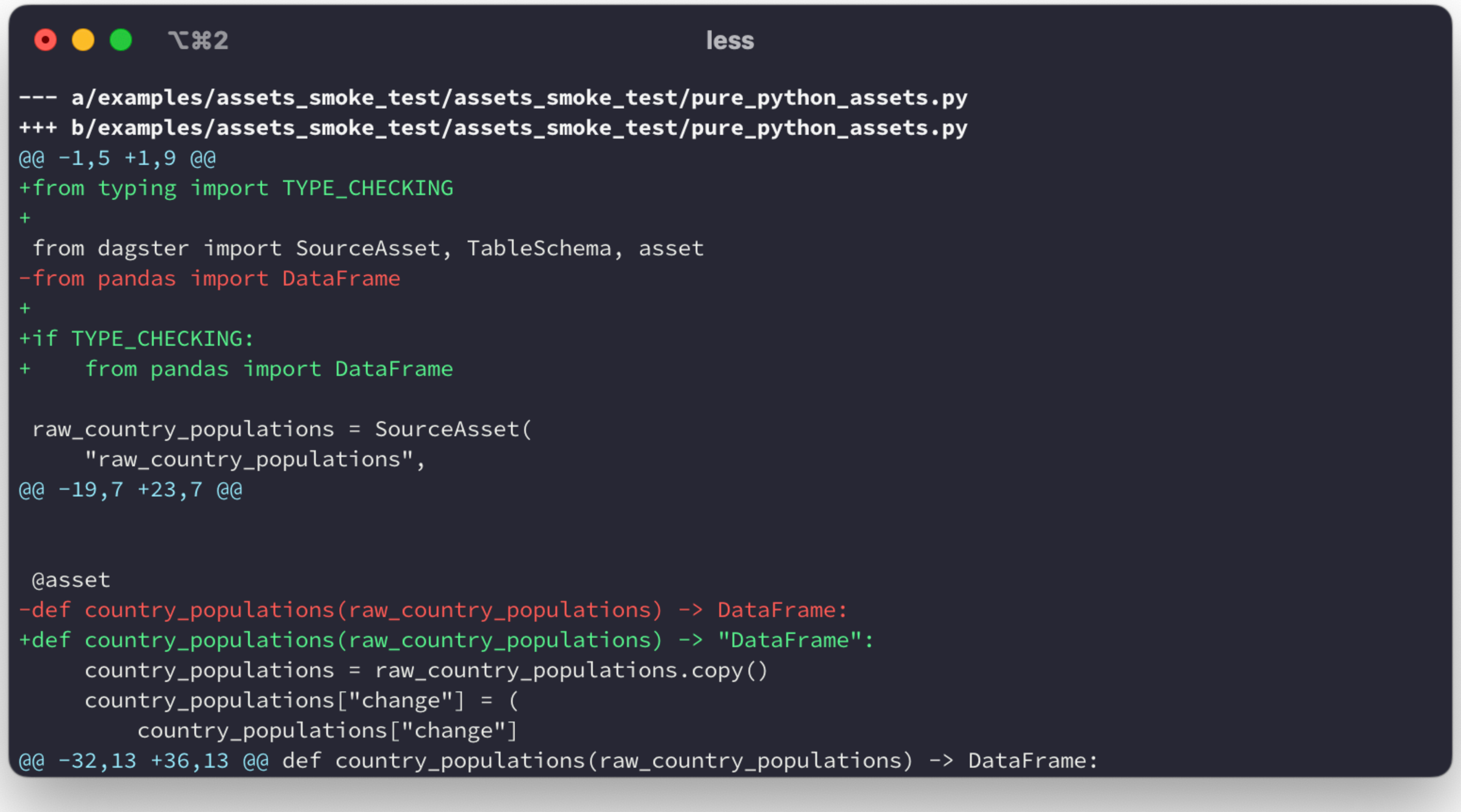
3. **Automated: a linter with code transformation capabilities**

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

2. Unified: replace dozens of tools with a single interface

3. **Automated: a linter with code transformation capabilities**



```
--- a/examples/assets_smoke_test/assets_smoke_test/pure_python_assets.py
+++ b/examples/assets_smoke_test/assets_smoke_test/pure_python_assets.py
@@ -1,5 +1,9 @@
+from typing import TYPE_CHECKING
+
 from dagster import SourceAsset, TableSchema, asset
-from pandas import DataFrame
+
+if TYPE_CHECKING:
+    from pandas import DataFrame

 raw_country_populations = SourceAsset(
     "raw_country_populations",
@@ -19,7 +23,7 @@


 @asset
-def country_populations(raw_country_populations) -> DataFrame:
+def country_populations(raw_country_populations) -> "DataFrame":
     country_populations = raw_country_populations.copy()
     country_populations["change"] = (
         country_populations["change"]
@@ -32,13 +36,13 @@ def country_populations(raw_country_populations) -> DataFrame:
```

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

2. Unified: replace dozens of tools with a single interface

3. Automated: a linter with code transformation capabilities

Why do people like Ruff?

1. Performant: 10-1000x faster than existing Python linters

2. Unified: replace dozens of tools with a single interface

3. Automated: a linter with code transformation capabilities

4. **Adoptable: drop-in replacement for existing tools**

Why do people like Ruff?

1. **Performant**: 10-1000x faster than existing Python linters

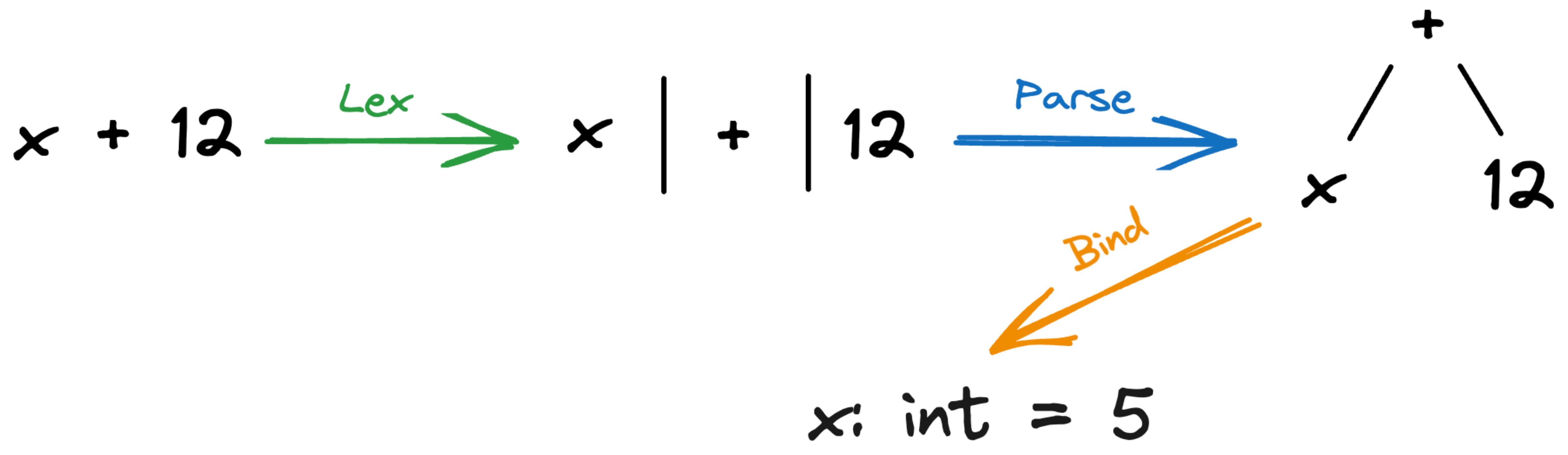
2. **Unified**: replace dozens of tools with a single interface

3. **Automated**: a linter with code transformation capabilities

4. **Adoptable**: drop-in replacement for existing tools

How does Ruff work?

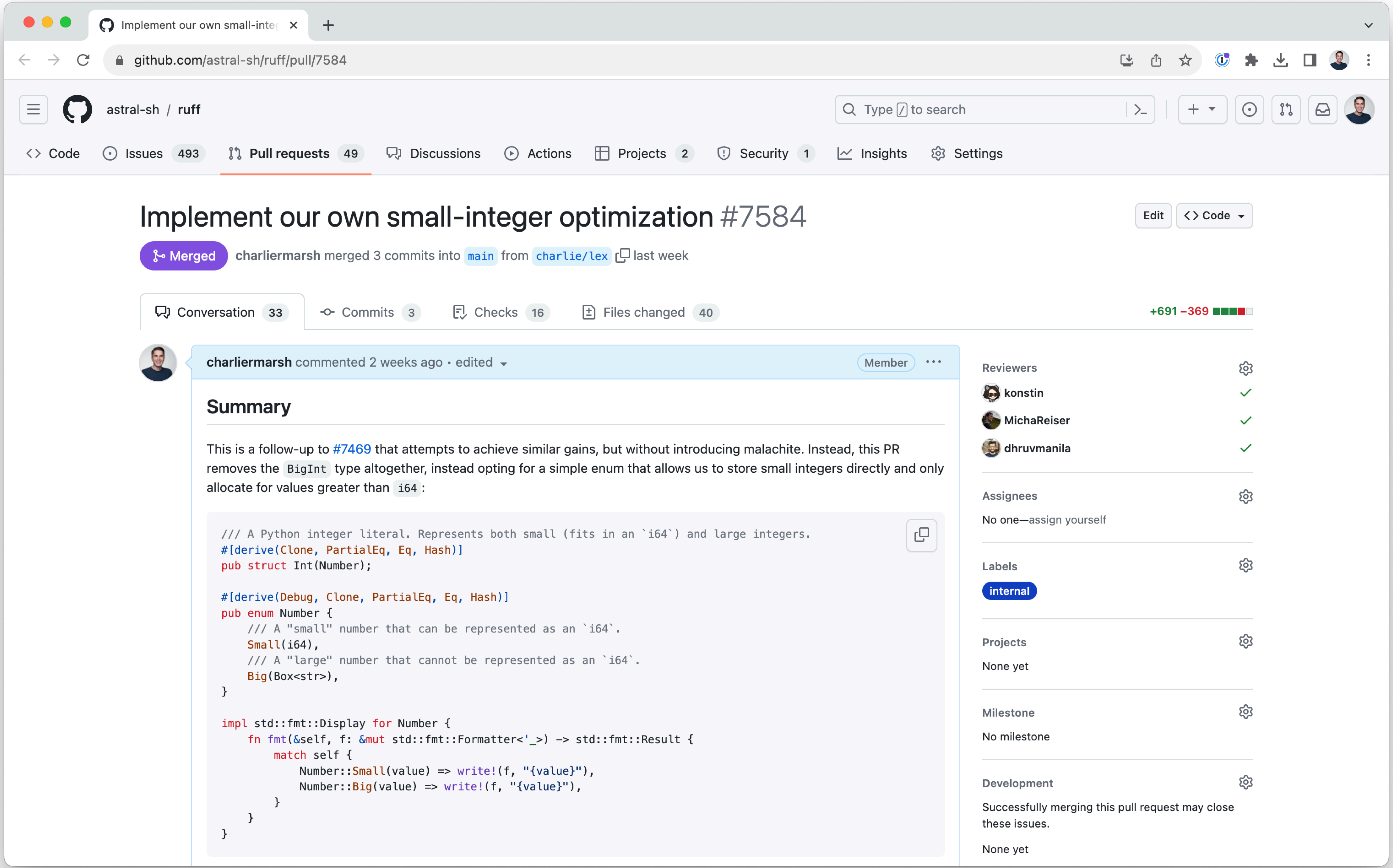
- **Compiler:** Python files in, diagnostics out
  - Discover all Python files
  - For every file, in parallel:
    - Read from disk
    - **Lex**: turn source code into tokens
    - **Parse**: turn tokens into syntax nodes
    - **Bind**: turn syntax nodes into semantic bindings
    - **Analyze**: run lint rules
    - Apply automatic fixes
    - Re-run until convergence
  - Report diagnostics



What makes Ruff fast?

- *Rust*
  - Rust is fast, but writing your program in Rust doesn't *guarantee* that it will be fast
  - Writing performant Rust is its own skillset
- *Parse once*
  - Unified tooling means significantly less repeated work
- *Fearless concurrency*
  - Embarrassingly parallel compilation model
- *A constant focus on performance*

ASTRAL



Implement our own small-integer optimization #7584

Merged charliermarsh merged 3 commits into `main` from `charlie/lex` last week

charliermarsh commented 2 weeks ago • edited

Summary

This is a follow-up to #7469 that attempts to achieve similar gains, but without introducing malachite. Instead, this PR removes the `BigInt` type altogether, instead opting for a simple enum that allows us to store small integers directly and only allocate for values greater than `i64`:

```
/// A Python integer literal. Represents both small (fits in an `i64`) and large integers.
#[derive(Clone, PartialEq, Eq, Hash)]
pub struct Int(Number);

#[derive(Debug, Clone, PartialEq, Eq, Hash)]
pub enum Number {
    /// A "small" number that can be represented as an `i64`.
    Small(i64),
    /// A "large" number that cannot be represented as an `i64`.
    Big(Box<str>),
}

impl std::fmt::Display for Number {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Number::Small(value) => write!(f, "{value}"),
            Number::Big(value) => write!(f, "{value}"),
        }
    }
}
```

Reviewers
konstin ✓
MichaReiser ✓
dhruvmanila ✓

Assignees
No one—assign yourself

Labels
internal

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

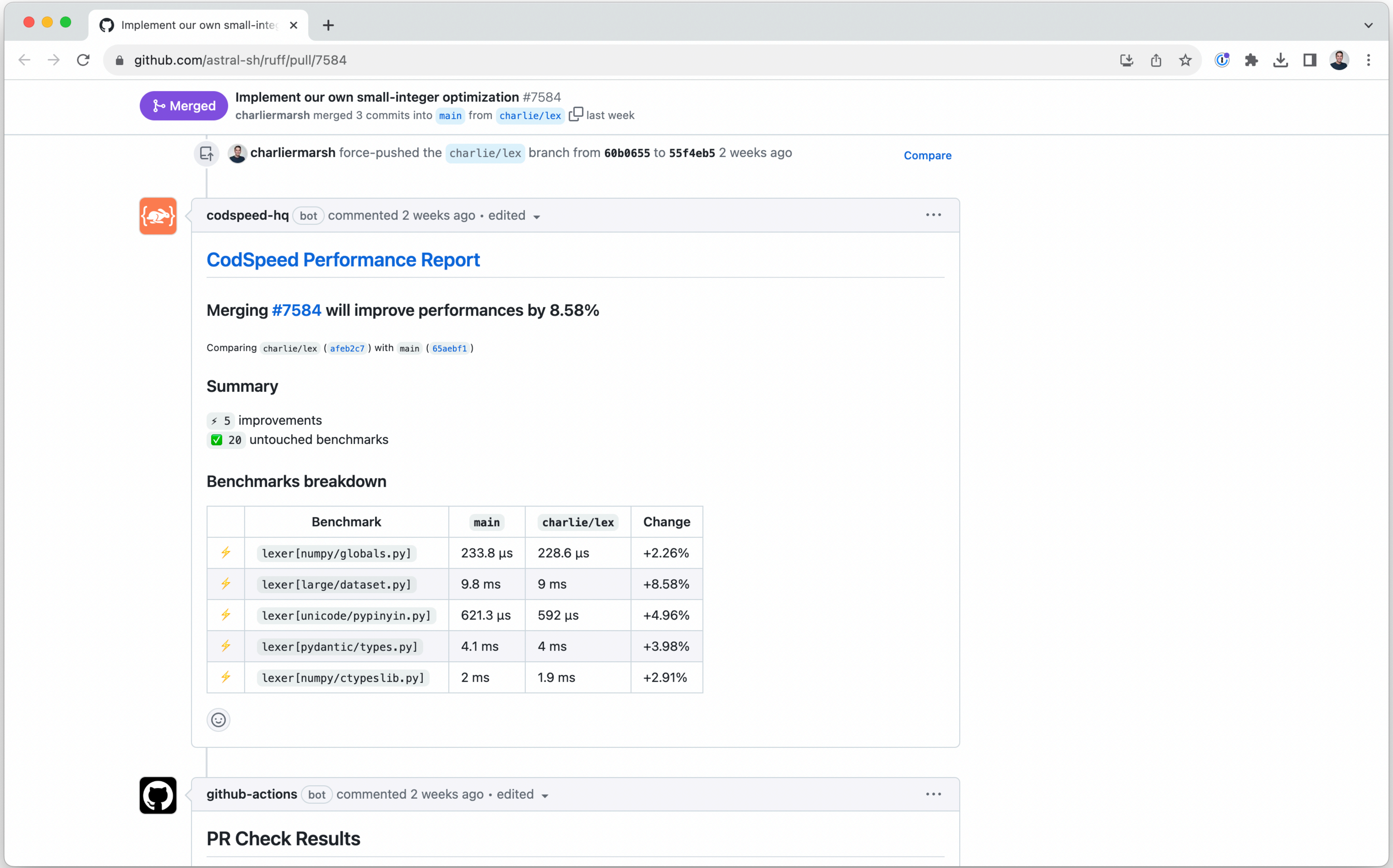
None yet

9_223_372_036_854_775_807

"12" ⟶ [1, 2]

"12" $\xrightarrow{\checkmark}$ i64

"9223372036854775808" $\xrightarrow{\times}$ str


Implement our own small-integ
github.com/astral-sh/ruff/pull/7584
Merged
Compare

Implement our own small-integer optimization #7584
charliermarsh merged 3 commits into `main` from `charlie/lex` last week

charliermarsh force-pushed the `charlie/lex` branch from **60b0655** to **55f4eb5** 2 weeks ago

codspeed-hq bot commented 2 weeks ago • edited

CodSpeed Performance Report

Merging #7584 will improve performances by 8.58%

Comparing `charlie/lex` (`afeb2c7`) with `main` (`65aebf1`)

Summary

⚡ `5` improvements
✅ `20` untouched benchmarks

Benchmarks breakdown

	Benchmark	`main`	`charlie/lex`	Change
⚡	`lexer[numpy/globals.py]`	233.8 µs	228.6 µs	+2.26%
⚡	`lexer[large/dataset.py]`	9.8 ms	9 ms	+8.58%
⚡	`lexer[unicode/pypinyin.py]`	621.3 µs	592 µs	+4.96%
⚡	`lexer[pydantic/types.py]`	4.1 ms	4 ms	+3.98%
⚡	`lexer[numpy/ctypeslib.py]`	2 ms	1.9 ms	+2.91%

github-actions bot commented 2 weeks ago • edited

PR Check Results

What makes Ruff fast?

- **Compiler:** Python files in, diagnostics out
    - Discover all Python files
    - For every file, in parallel:
        - Read from disk
        - **Lex**: turn source code into tokens
        - **Parse**: turn tokens into syntax nodes
        - **Bind**: turn syntax nodes into semantic bindings
        - **Analyze**: run lint rules
        - Apply automatic fixes
        - Re-run until convergence
    - Report diagnostics

What makes Ruff fast?

- **Compiler:** Python files in, diagnostics out
  - Discover all Python files **(~5%)**
  - For every file, in parallel:
    - Read from disk **(~5%)**
    - **Lex**: turn source code into tokens **(~20%)**
    - **Parse**: turn tokens into syntax nodes **(~30%)**
    - **Bind**: turn syntax nodes into semantic bindings **(~10%)**
    - **Analyze**: run lint rules **(~10%)**
    - Apply automatic fixes
    - Re-run until convergence
  - Report diagnostics

Ruff could be much faster

- **Compiler:** Python files in, diagnostics out
    - Discover all Python files **(~5%)**
    - For every file, in parallel:
        - Read from disk **(~5%)**
        - **Lex**: turn source code into tokens **(~20%)**
        - **Parse**: turn tokens into syntax nodes **(~30%)**
        - **Bind**: turn syntax nodes into semantic bindings **(~10%)**
        - **Analyze**: run lint rules **(~10%)**
        - Apply automatic fixes
        - Re-run until convergence
    - Report diagnostics

Rust, Python

NEXT-GEN PYTHON TOOLING

PYTEXAS

OCTOBER 3, 2023