Rust Plugins with WASM

Mike Seddon

Opinions Inside

About Me

- **Scala** mainly due to Apache Spark.
    - JVM with GC
    - Opinions

- **Go** as an antidote to type-theorists.
    - Tooling
    - Stdlib

- **Rust** as the sensible middle-ground.

The Problem

- VisualCortex deploys Computer Vision.

- Main platform are Rust Gstreamer pipelines where plugins are written with Rust.

- Need an arbitrary code execution capability to allow dynamic pipelines.

WebAssembly

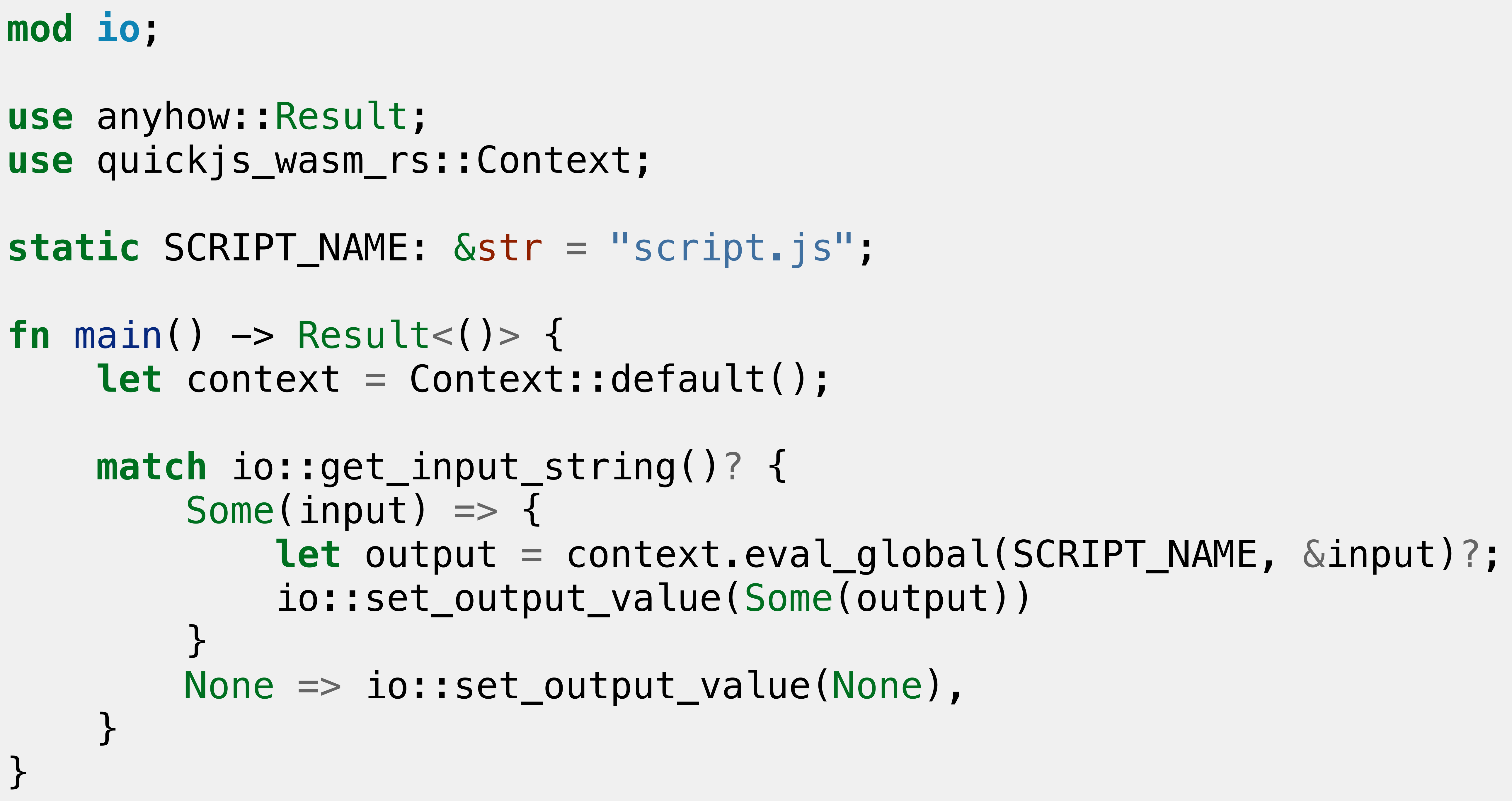
- WebAssembly is a portable standard Assembly specification.

- Rust is able to be compiled to that Assembly format. So are many other languages.

- Rust has the excellent **Wasmtime** WebAssembly runtime.

Scripting Language

- **QuickJS** A portable (1.2MB) JavaScript runtime.
- Excellent bindings from Shopify.



```rust
mod io;

use anyhow::Result;
use quickjs_wasm_rs::Context;

static SCRIPT_NAME: &str = "script.js";

fn main() -> Result<()> {
    let context = Context::default();

    match io::get_input_string()? {
        Some(input) => {
            let output = context.eval_global(SCRIPT_NAME, &input)?;
            io::set_output_value(Some(output))
        }
        None => io::set_output_value(None),
    }
}
```

An imaginary problem

260 Feature objects, ~110KB string

```
{
  "type": "FeatureCollection",
  "name": "track_points",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
    }
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "track_fid": 0,
        "track_seg_id": 0,
        "track_seg_point_id": 1,
        "ele": 60,
        "time": "2023-01-22 00:00:00+00"
      },
      "geometry": { "type": "Point", "coordinates": [151.22989, -33.89279] }
    }
  ]
}
```

An imaginary problem

Some Arbitrary Code

```
// simple approximate distance function returning distance
// between two points in meters
function distance(lat0, lon0, lat1, lon1) {
  if (lat0 == lat1 && lon0 == lon1) {
    return 0;
  } else {
    const radlat0 = (Math.PI * lat0) / 180;
    const radlat1 = (Math.PI * lat1) / 180;
    const theta = lon0 - lon1;
    const radtheta = (Math.PI * theta) / 180;
    let dist =
      Math.sin(radlat0) * Math.sin(radlat1) +
      Math.cos(radlat0) * Math.cos(radlat1) * Math.cos(radtheta);
    if (dist > 1) {
      dist = 1;
    }
    dist = Math.acos(dist);
    dist = (dist * 180) / Math.PI;
    return dist * 60 * 1853.159;
  }
}
```

```
// calculate the total length of a set of input features
function calculate(data) {
  return data.features.reduce(
    (accumulator, currentValue, currentIndex, array) => {
      if (currentIndex == 0) {
        return 0;
      } else {
        const previousValue = array[currentIndex - 1];
        const dist = distance(
          currentValue.geometry.coordinates[1],
          currentValue.geometry.coordinates[0],
          previousValue.geometry.coordinates[1],
          previousValue.geometry.coordinates[0]
        );
        return accumulator + dist;
      }
    },
    0
  );
}
```

Some Benchmarks: Baseline

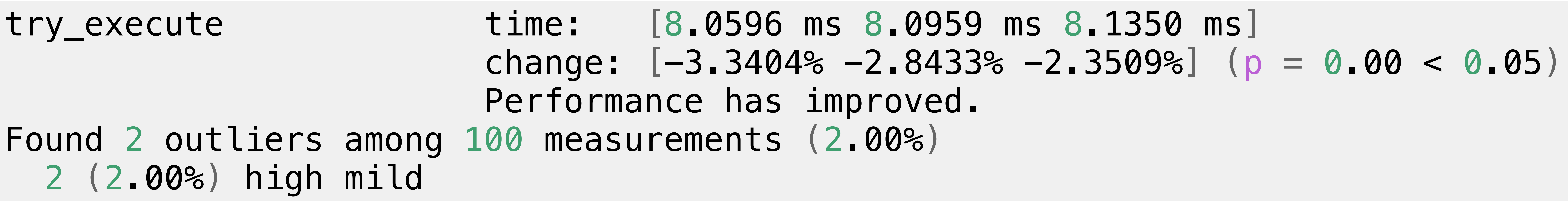
- Just pass in everything and run it.

```
try_execute               time:   [8.3125 ms 8.3329 ms 8.3544 ms]
Found 5 outliers among 100 measurements (5.00%)
  1 (1.00%) low mild
  3 (3.00%) high mild
  1 (1.00%) high severe
```

… terrible …

Some Benchmarks: Wizer

- Cache the QuickJS instantiation cost.

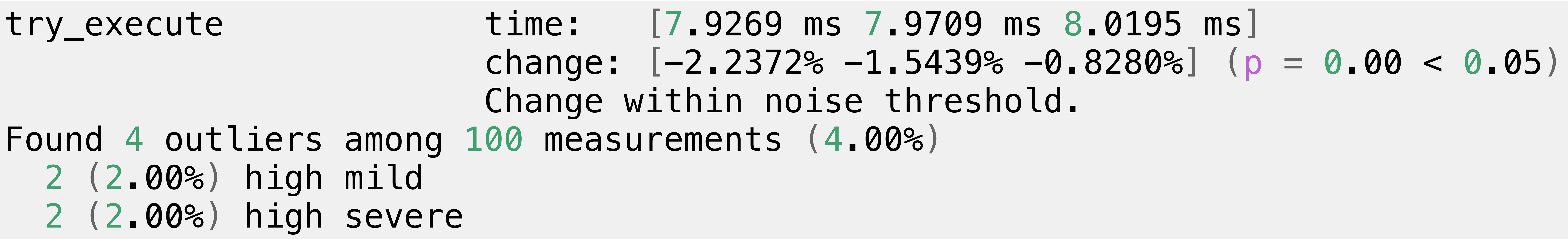


```
try_execute             time:   [8.0596 ms 8.0959 ms 8.1350 ms]
                        change: [-3.3404% -2.8433% -2.3509%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 2 outliers among 100 measurements (2.00%)
  2 (2.00%) high mild
```

… slightly less terrible …

Some Benchmarks: Wizer+ aka Javy

- Cache the QuickJS instantiation cost and the function parsing cost.



```
try_execute             time:   [7.9269 ms 7.9709 ms 8.0195 ms]
                        change: [-2.2372% -1.5439% -0.8280%] (p = 0.00 < 0.05)
                        Change within noise threshold.
Found 4 outliers among 100 measurements (4.00%)
  2 (2.00%) high mild
  2 (2.00%) high severe
```

… still slow and now you need to compile a WASM binary for each unique set of functions …

Some Benchmarks: Digging Deeper

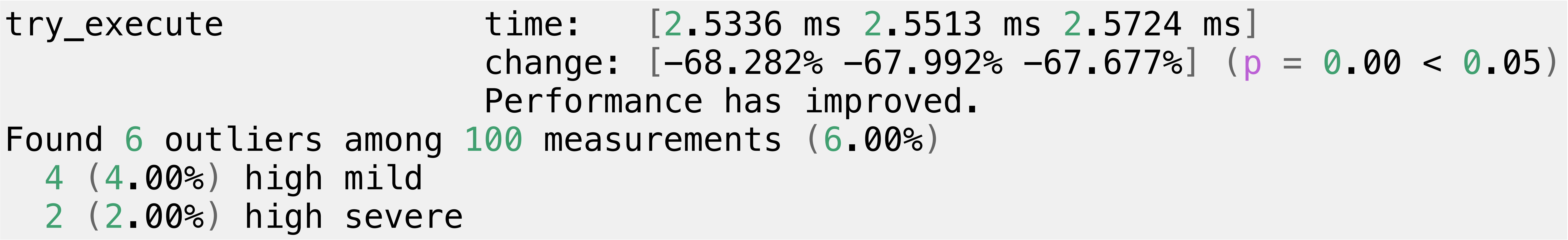
- Cache everything up front to isolate the cause of the performance.

```
try_execute             time:   [890.44 µs 898.09 µs 906.87 µs]
                        change: [−88.793% −88.647% −88.501%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 6 outliers among 100 measurements (6.00%)
  4 (4.00%) high mild
  2 (2.00%) high severe
```

… QuickJS is not slow, its parser is …

Some Benchmarks: Javy

- Use Javy's parser to pass data between Rust and QuickJS.



```
try_execute             time:   [2.5336 ms 2.5513 ms 2.5724 ms]
                        change: [-68.282% -67.992% -67.677%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 6 outliers among 100 measurements (6.00%)
  4 (4.00%) high mild
  2 (2.00%) high severe
```

… no longer the slowest component of the pipeline

Some Benchmarks: Parallel

- Rayon to the rescue

```
cargo run --release --example par_iter

elapsed: 475.869146ms
iteration: 475.869µs
```

… we have a lot of threads, use them.

Links

Blog:

https://reorchestrate.com/posts/plugins-for-rust/

Plumbing Code:

https://github.com/seddonm1/quickjs