R and Rust

like a match made in heaven

Andrés Felipe Quintero

Who am I

- Andrés F Quintero
- Online: **andyquinterom**
- From Cali, Colombia
- Currently @ ixpantia
- Working with R daily for over 6 years
- Working with Rust daily for over 2 years
- Deployed **big data pipeline** in production using both languages
- Love learning and teaching

Table of contents

Before getting started…

How are R and Rust similar?

Mutability (default behavior)

```r
try_mutate <- function(x) {
  x <- 2
}

x <- 1

try_mutate(x)

print(x)
# [1] 1
```

```rust
fn try_mutate(x: i32) {
    let x = 2;
}

let x = 1;

try_mutate(x);

println!("{x}”);
// 1
```

Functional programming patterns

```r
0:10 |>
  purrr::map_int(~ .x + 1) |>
  sum()
```

```rust
(0..=10)
    .map(|x| x + 1)
    .sum::<i32>();
```

Functional programming patterns

```r
0:10 ▷
  purrr::map_int(~ .x + 1) ▷
  sum()
```

```rust
(0..=10)
    .map(|x| x + 1)
    .sum::<i32>();
```

Polymorphism (In a non typical OO way)

```r
my_poly_function ← function(x) {
  UseMethod("my_poly_function")
}

my_poly_function.double ← function(x) {
  x^2
}


my_poly_function.integer ← function(x) {
  x^3
}


my_poly_function.character ← function(x) {
  paste(x, x, sep = "")
}
```

```rust
trait MyPolyTrait {
    fn my_poly_fn(&self) → Self;
}

impl MyPolyTrait for f64 {
    fn my_poly_fn(&self) → f64 {
        self.powi(2)
    }
}

impl MyPolyTrait for i32 {
    fn my_poly_fn(&self) → i32 {
        self.pow(3)
    }
}

impl MyPolyTrait for String {
    fn my_poly_fn(&self) → String {
        format!("{self}{self}")
    }
}
```

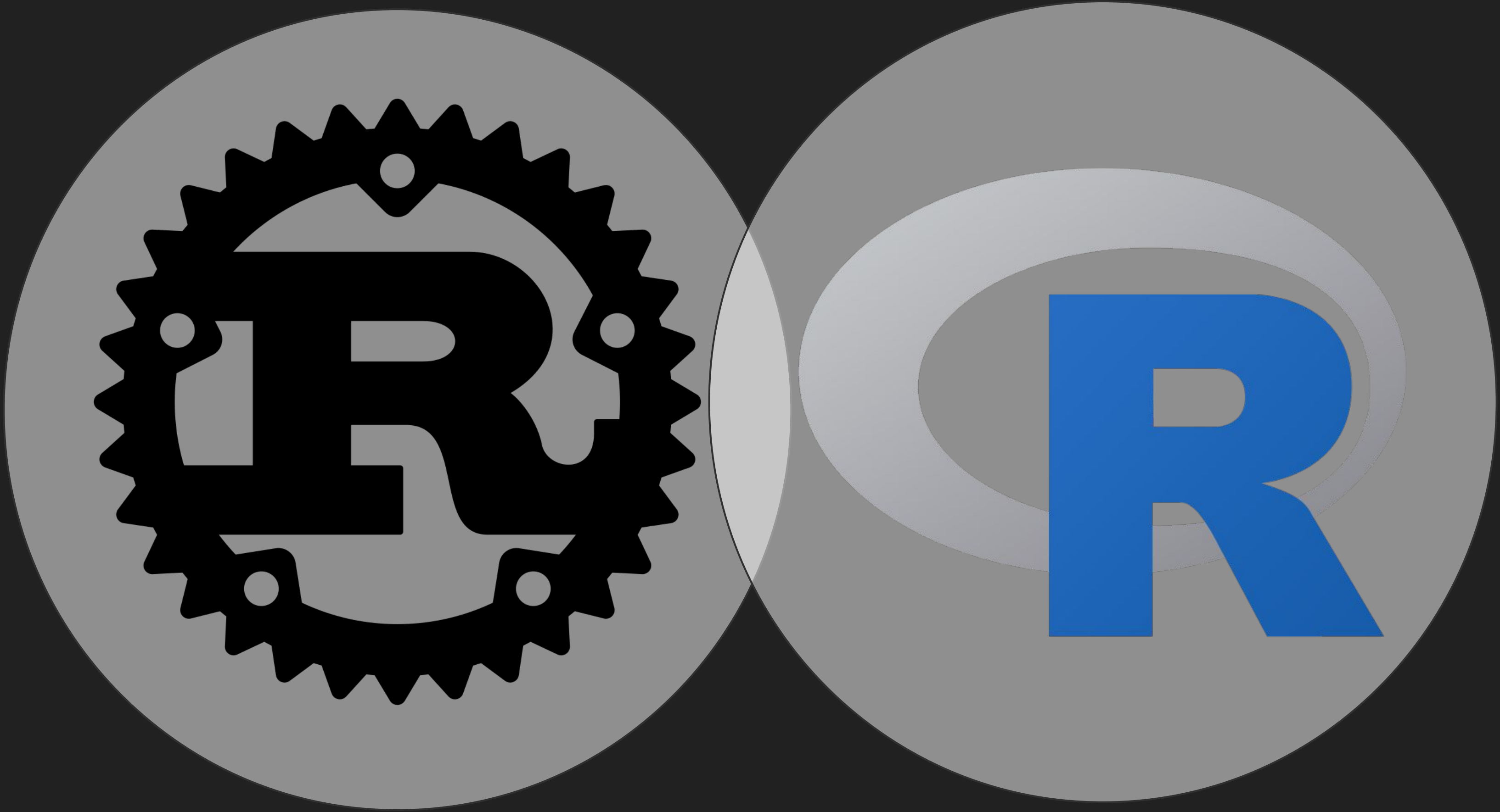
Functions are first class citizens

```r
my_funcs ← c(
  function(x) x + 1,
  function(x) x + 2,
  function(x) x + 3
)

res ← my_funcs ▷
  purrr::map_dbl(~ .x(1)) ▷
  purrr::reduce(~ .x + .y)


print(res)
# [1] 9
```

```rust
let my_funcs = [
    |x: f64| x + 1.0,
    |x: f64| x + 2.0,
    |x: f64| x + 3.0
];

let res = my_funcs
    .iter()
    .map(|f| f(1.0))
    .reduce(|x, y| x + y);

println!("{:?}", res);
// Some(9.0)
```

Many… many more

How R and Rust compliment each other



- Structure
- “Enforced contracts”
- High (really high) performance
- General purpose

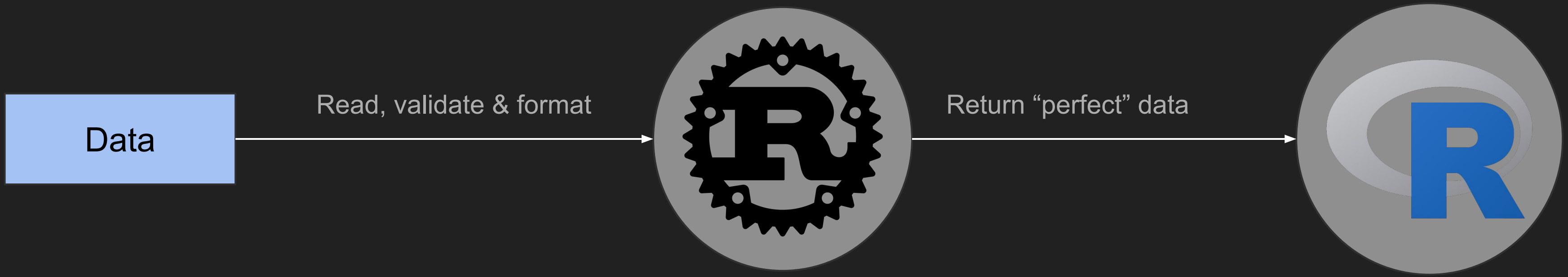
- Unstructured (dynamic)
- Hard to truly enforce contracts
- Low performance
- Statistical programming language

- Structure
- “Enforced contracts”
- High (really high) performance
- General purpose

- Unstructured (dynamic)
- Hard to truly enforce contracts
- Low performance
- Statistical programming language

- Data Validation
- Complex data-structures
- I/O (Database, File System, Etc)
- Orchestration

- Exploration
- Dashboards / Apps (Shiny)
- Modelling
- Almost everything you already do



What I refer to “perfect data” is data that complies with a defined contract.

```
my_data |>
  filter(age == 10) |>
  mutate(new_column = paste(“prefix”, other_column) |>
  group_by(new_column) |>
  count()
```

```
my_data |>
  filter(age == 10) |>
  mutate(new_column = paste(“prefix”, other_column) |>
  group_by(new_column) |>
  count()
```

What if `other_column` is not present on the data?

```
my_data ▷
  filter(age == 10) ▷
  mutate(new_column = paste(“prefix”, other_column) ▷
  group_by(new_column) ▷
  count()
```

Is this even a data.frame?

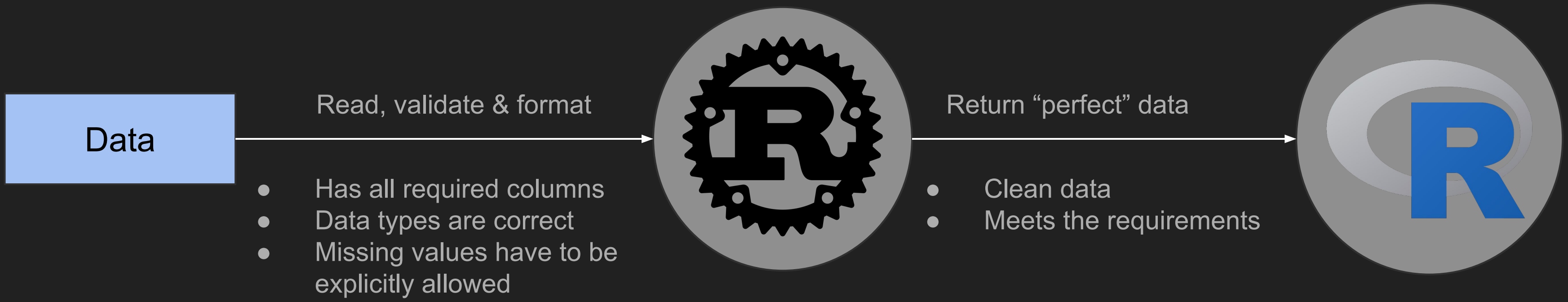
What if `other_column` is not present on the data?

```
my_data |>
  filter(age == 10) |>
  mutate(new_column = paste(“prefix”, other_column) |>
  group_by(new_column) |>
  count()
```

Is age comparable to a numeric?

Is this even a data.frame?

What if `other_column` is not present on the data?



Read, validate & format

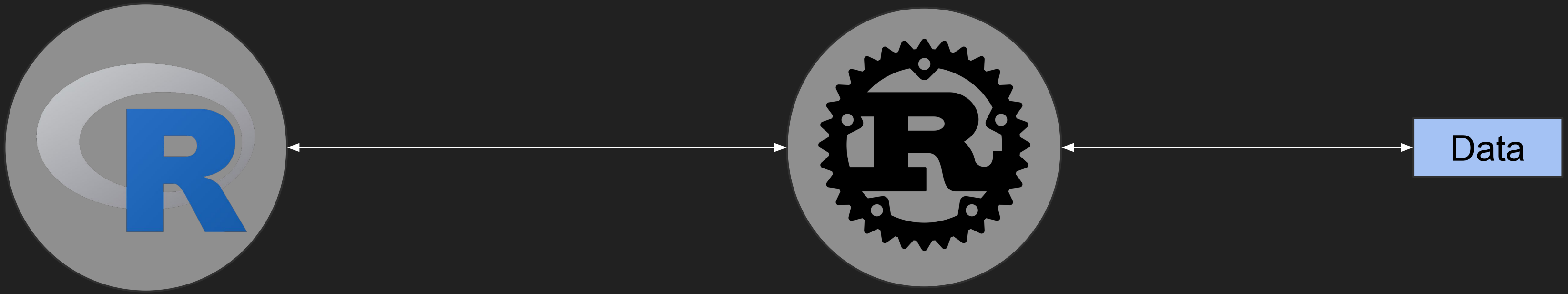
- Has all required columns
- Data types are correct
- Missing values have to be explicitly allowed

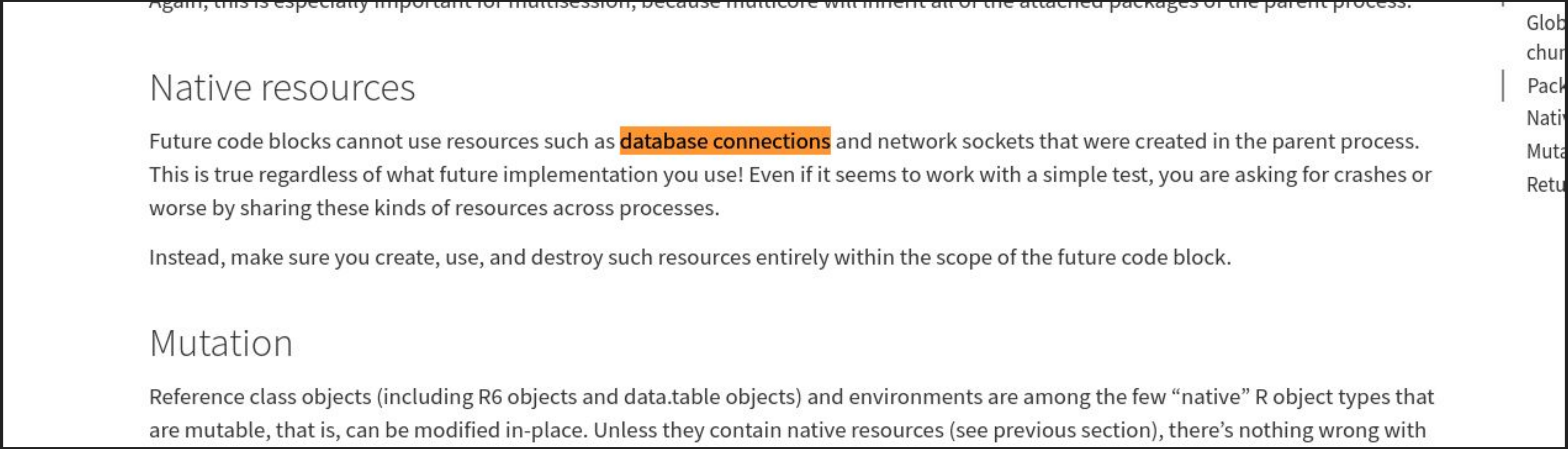
Return "perfect" data

- Clean data
- Meets the requirements

What I refer to "perfect data" is data that complies with a defined contract.

- Heavy computation on data
- Fetch data from different sources





Again, this is especially important for multisession, because multicore will inherit all of the attached packages of the parent process.

Native resources

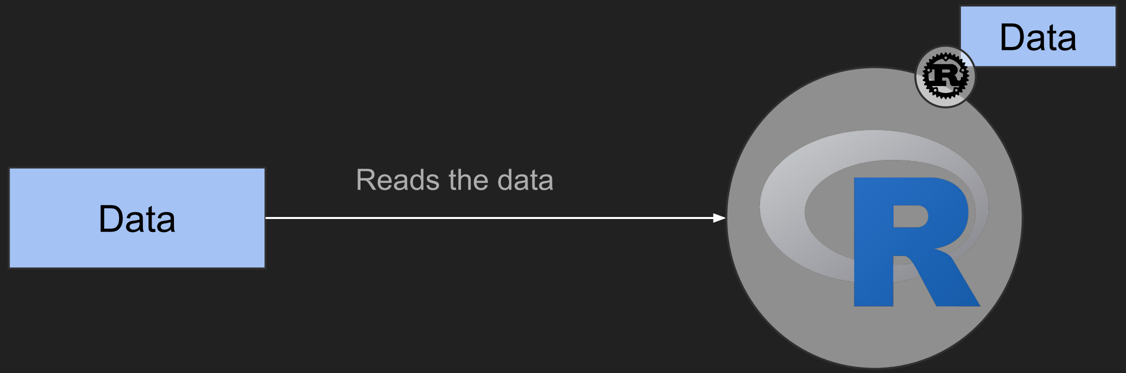
Future code blocks cannot use resources such as database connections and network sockets that were created in the parent process. This is true regardless of what future implementation you use! Even if it seems to work with a simple test, you are asking for crashes or worse by sharing these kinds of resources across processes.

Instead, make sure you create, use, and destroy such resources entirely within the scope of the future code block.

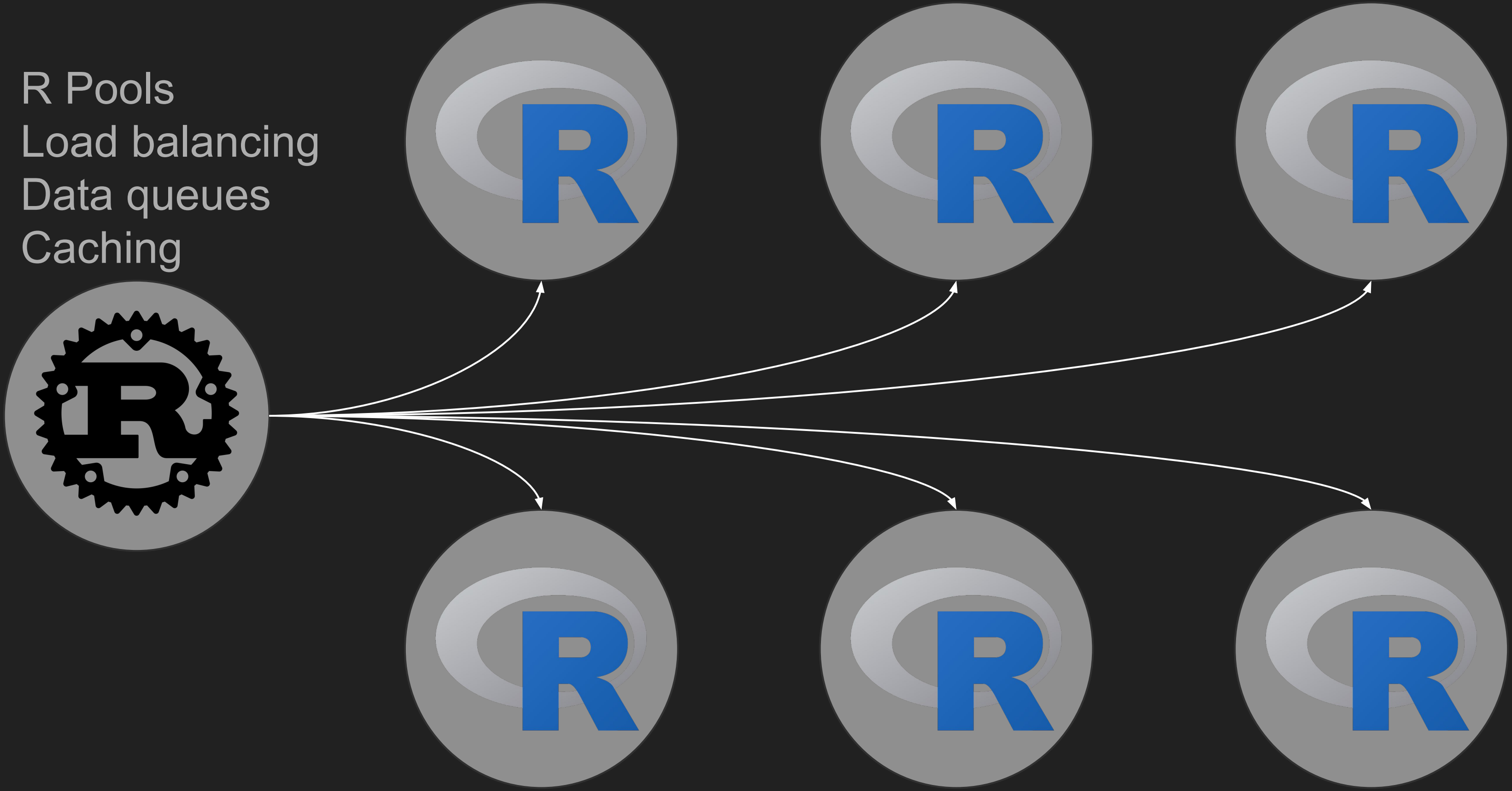
Mutation

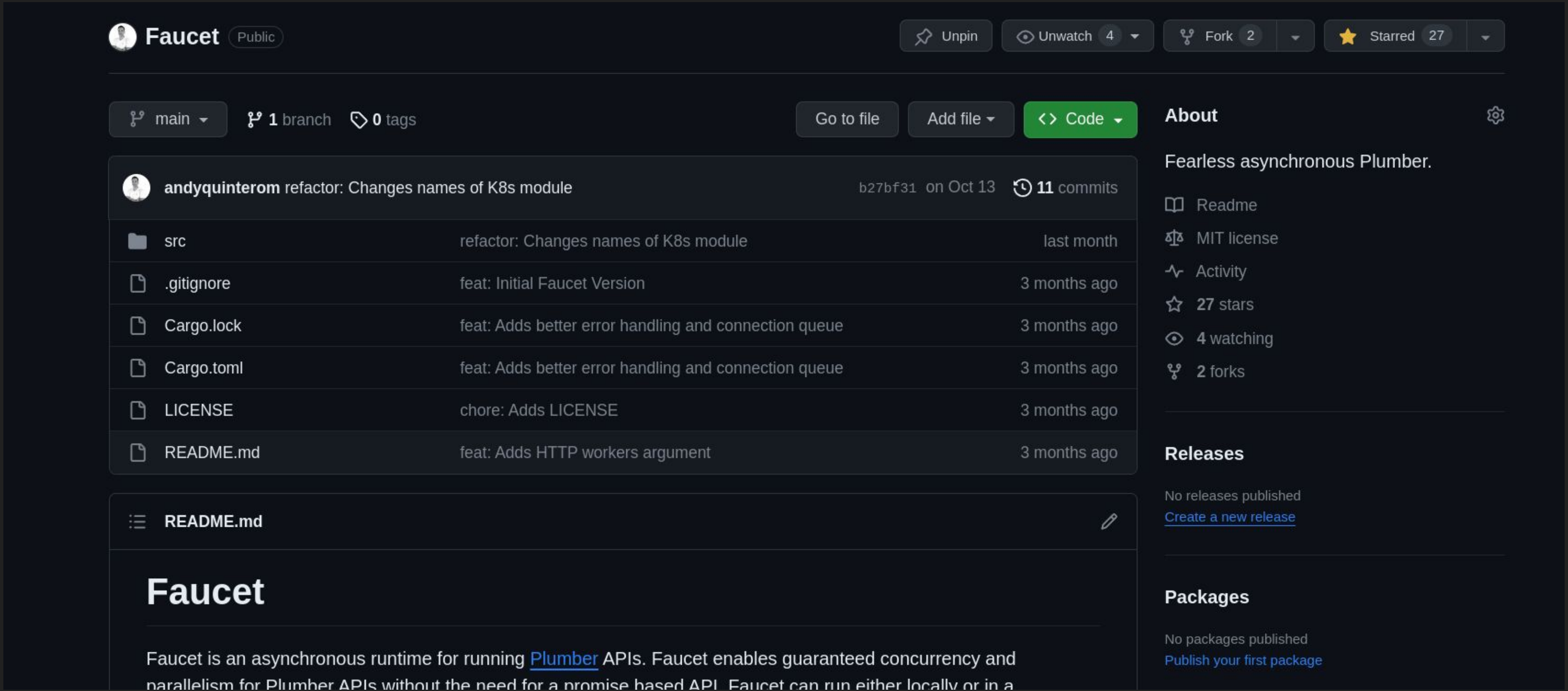
Reference class objects (including R6 objects and data.table objects) and environments are among the few “native” R object types that are mutable, that is, can be modified in-place. Unless they contain native resources (see previous section), there’s nothing wrong with

- Rust holds data in memory behind a **black box**
- Rust performs computation on said data and returns it to R



- R Pools
- Load balancing
- Data queues
- Caching




Faucet Public

Unpin | Unwatch 4 | Fork 2 | Starred 27

main | 1 branch | 0 tags | Go to file | Add file | Code

andyquinterom refactor: Changes names of K8s module b27bf31 on Oct 13 11 commits

src	refactor: Changes names of K8s module	last month
.gitignore	feat: Initial Faucet Version	3 months ago
Cargo.lock	feat: Adds better error handling and connection queue	3 months ago
Cargo.toml	feat: Adds better error handling and connection queue	3 months ago
LICENSE	chore: Adds LICENSE	3 months ago
README.md	feat: Adds HTTP workers argument	3 months ago

README.md

Faucet

Faucet is an asynchronous runtime for running Plumber APIs. Faucet enables guaranteed concurrency and parallelism for Plumber APIs without the need for a promise based API. Faucet can run either locally or in a

About

Fearless asynchronous Plumber.

Readme
MIT license
Activity
27 stars
4 watching
2 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

TLDR

- Rust is high-level enough where you don’t need a full low-level knowledge.
- Rust provides structure when R needs it
- Rust provides high performance when there is no alternative.
  - Most popular libraries will not need a boost from something like Rust since they are already written in C or C++
- Rust can help you make the most out of R by smartly orchestrating R processes / load balancing different APIs.

Building high performance R packages with Rust

We need more performance!

Let’s say we need to scan the file system, read and parse many JSON files.

We need more performance!

First we create 100.000 json files for our simulation.

```bash
#!/bin/bash

mkdir json_files

for i in {1..100000}
do
    JSON_VALUE="{\"value\": $i}"
    echo $JSON_VALUE > json_files/$i.json
done
```

Approach #1

```
library(jsonlite)
library(purrr)

json_files ← list.files(path = "json_files")

system.time({
  values ← json_files ▷
    map(~ fromJSON(file.path("json_files", .x))) ▷
    map_dbl(~ .x$value)
})

#   user  system elapsed
#  7.703   2.064   9.792
```

Approach #2

```
library(jsonlite)
library(furrr)
library(future)

plan(multicore)

json_files ← list.files(path = "json_files")

system.time({
  values ← json_files ▷
    future_map(~ fromJSON(file.path("json_files", .x))) ▷
    future_map_dbl(~ .x$value)
})

#   user  system elapsed
# 15.098   4.588   3.641
```

Approach #3

```
usethis::create_package("fastjson")
—
cd fastjson
—
rextendr::use_extendr()
```

Approach #3

```
.
├── DESCRIPTION
├── NAMESPACE
├── R
│   └── extendr-wrappers.R
└── src
    ├── entrypoint.c
    ├── fastjson-win.def
    ├── Makevars
    ├── Makevars.ucrt
    ├── Makevars.win
    └── rust
        ├── Cargo.toml
        └── src
            └── lib.rs
```

Approach #3

```
cd src/rust
cargo add serde -F derive
cargo add serde_json
```

Approach #3

```rust
// src/lib.rs
use extendr_api::prelude::*;
use std::fs::read_to_string;

#[derive(serde::Deserialize)]
struct JsonData {
    value: f64,
}

/// @export
#[extendr]
fn read_json_and_extract_values(files: Vec<String>) -> Vec<f64> {
    files
        .iter()
        .map(|file_path| read_to_string(file_path).expect("Unable to read file"))
        .map(|json_string| {
            // Equivalent to `jsonlite::fromJSON`
            serde_json::from_str::<JsonData>(&json_string).expect("Unable to parse json")
        })
        .map(|json_data| json_data.value)
        .collect::<Vec<f64>>()
}

extendr_module! {
    mod fastjson;
    fn read_json_and_extract_values;
}
```

Approach #3

```
rextendr::document()
devtools::install()
```

Approach #3

```
library(fastjson)

json_files ← list.files(path = "json_files")

system.time({
  values ← file.path("json_files", json_files) ▷
    read_json_and_extract_values()
})

#   user  system elapsed
#  0.073   0.527   0.609
```

Approach #4

```
cd src/rust
cargo add rayon
```

Approach #4

```rust
// src/lib.rs
use extendr_api::prelude::*;
use std::fs::read_to_string;

#[derive(serde::Deserialize)]
struct JsonData {
    value: f64,
}

/// @export
#[extendr]
fn read_json_and_extract_values(files: Vec<String>) -> Vec<f64> {
    files
        .iter()
        .map(|file_path| read_to_string(file_path).expect("Unable to read file"))
        .map(|json_string| {
            // Equivalent to `jsonlite::fromJSON`
            serde_json::from_str::<JsonData>(&json_string).expect("Unable to parse
json")
        })
        .map(|json_data| json_data.value)
        .collect::<Vec<f64>>()
}

extendr_module! {
    mod fastjson;
    fn read_json_and_extract_values;
}
```

```rust
// src/lib.rs
use extendr_api::prelude::*;
use rayon::prelude::*;
use serde::Deserialize;
use std::fs::read_to_string;

#[derive(Deserialize)]
struct JsonData {
    value: f64,
}

/// @export
#[extendr]
fn read_json_and_extract_values(files: Vec<String>) -> Vec<f64> {
    files
        .par_iter()
        .map(|file_path| read_to_string(file_path).expect("Unable to read file"))
        .map(|json_string| {
            // Equivalent to `jsonlite::fromJSON`
            serde_json::from_str::<JsonData>(&json_string).expect("Unable to parse
json")
        })
        .map(|json_data| json_data.value)
        .collect::<Vec<f64>>()
}

extendr_module! {
    mod fastjson;
    fn read_json_and_extract_values;
}
```

Approach #4 (Our R code did not change)

```
library(fastjson)

json_files ← list.files(path = "json_files")

system.time({
  values ← file.path("json_files", json_files) ▷
    read_json_and_extract_values()
})

#   user  system elapsed
#  0.118   0.544   0.077
```

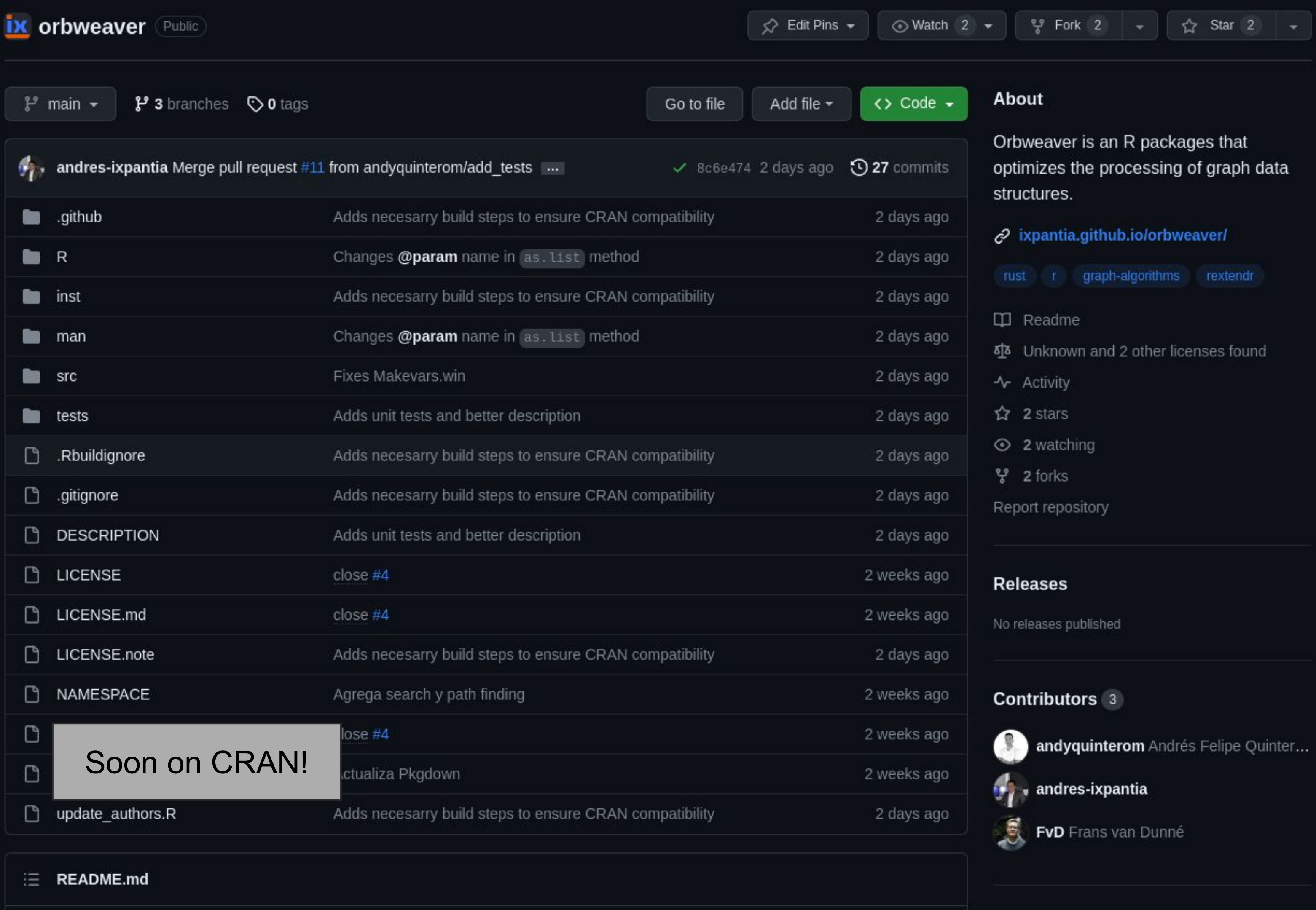
Results

Base R with Purrr and Jsonlite: `9.792`

Base R with Furrr and Jsonlite: `3.641`

R + Rust (Serde): `0.609`

R + Rust (Serde) + Rayon: `0.077`



orbweaver Public

Edit Pins | Watch 2 | Fork 2 | Star 2

main | 3 branches | 0 tags | Go to file | Add file | Code

andres-ixpantia Merge pull request #11 from andyquinterom/add_tests ... ✓ 8c6e474 2 days ago 27 commits

.github	Adds necesarry build steps to ensure CRAN compatibility	2 days ago
R	Changes @param name in `as.list` method	2 days ago
inst	Adds necesarry build steps to ensure CRAN compatibility	2 days ago
man	Changes @param name in `as.list` method	2 days ago
src	Fixes Makevars.win	2 days ago
tests	Adds unit tests and better description	2 days ago
.Rbuildignore	Adds necesarry build steps to ensure CRAN compatibility	2 days ago
.gitignore	Adds necesarry build steps to ensure CRAN compatibility	2 days ago
DESCRIPTION	Adds unit tests and better description	2 days ago
LICENSE	close #4	2 weeks ago
LICENSE.md	close #4	2 weeks ago
LICENSE.note	Adds necesarry build steps to ensure CRAN compatibility	2 days ago
NAMESPACE	Agrega search y path finding	2 weeks ago
	lose #4	2 weeks ago
	ctualiza Pkgdown	2 weeks ago
update_authors.R	Adds necesarry build steps to ensure CRAN compatibility	2 days ago

README.md

Soon on CRAN!

About

Orbweaver is an R packages that optimizes the processing of graph data structures.

ixpantia.github.io/orbweaver/

rust | r | graph-algorithms | rextendr

Readme
Unknown and 2 other licenses found
Activity
2 stars
2 watching
2 forks
Report repository

Releases

No releases published

Contributors 3

andyquinterom Andrés Felipe Quinter...
andres-ixpantia
FvD Frans van Dunné

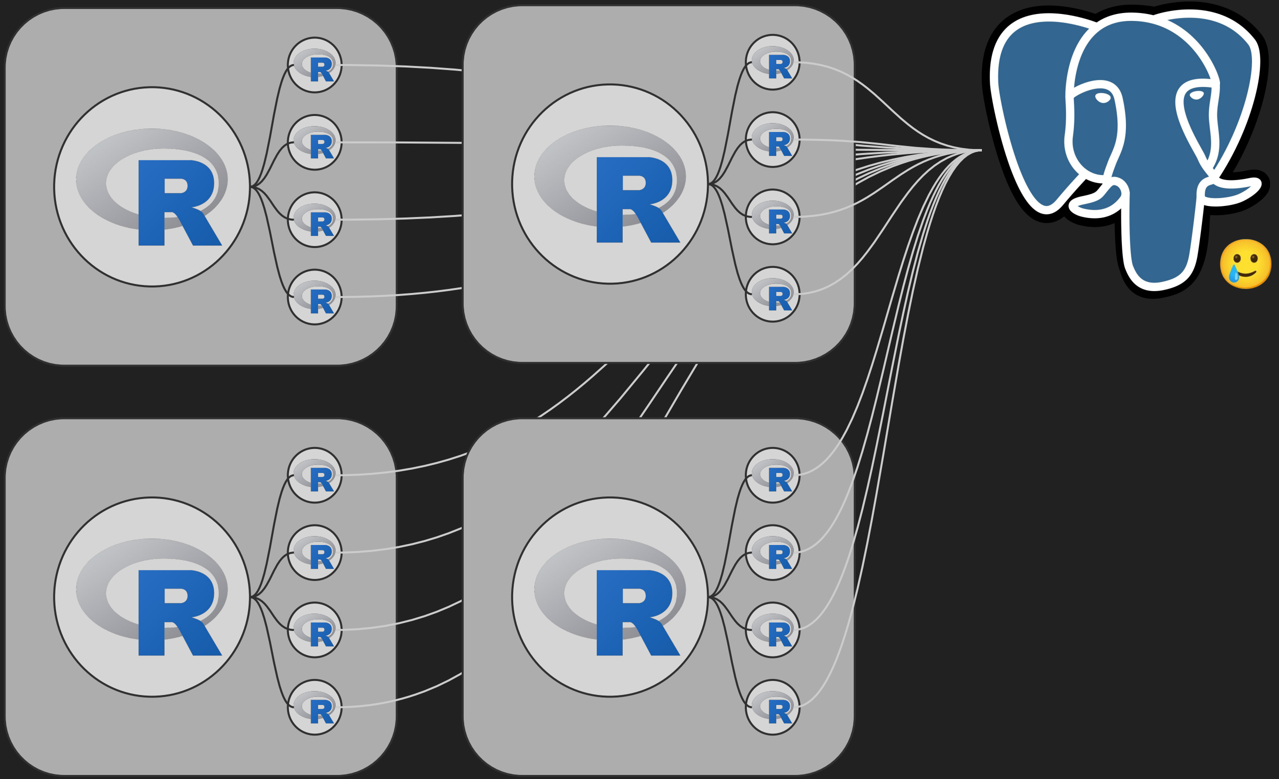
```r
library(orbweaver)

tree <- new_graph(type = "acyclic") |>
  # Node A has children B and C
  add_child("A", "B") |>
  add_child("A", "C") |>
  # Node B has children D and E
  add_child("B", "D") |>
  add_child("B", "E") |>
  # Node C has child F
  add_child("C", "F")

tree |>
  find_roots()
# [1] "A"

tree |>
  find_leaves("A")
# [1] "F" "E" "D"

tree |>
  find_least_common_parents(c("B", "D", "E"))
# [1] "B"
```

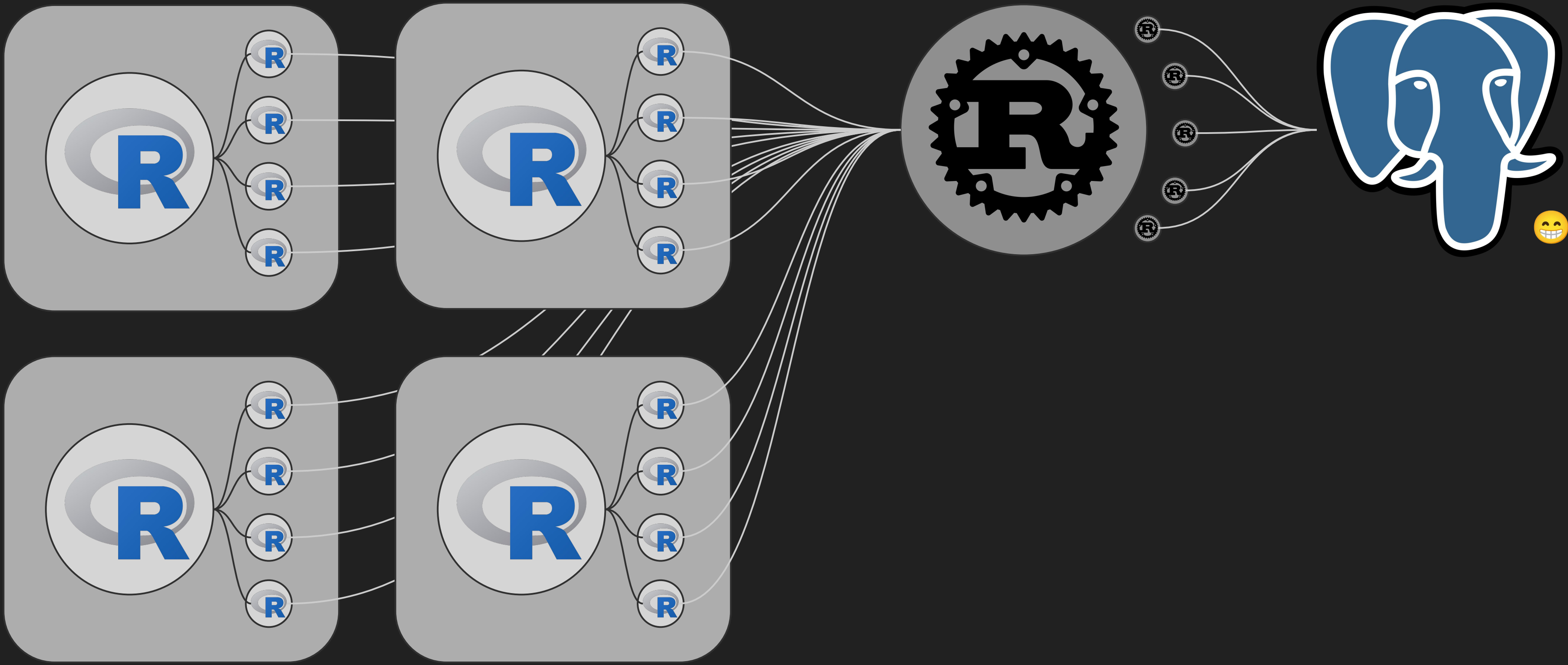
R and Rust in microservice architectures

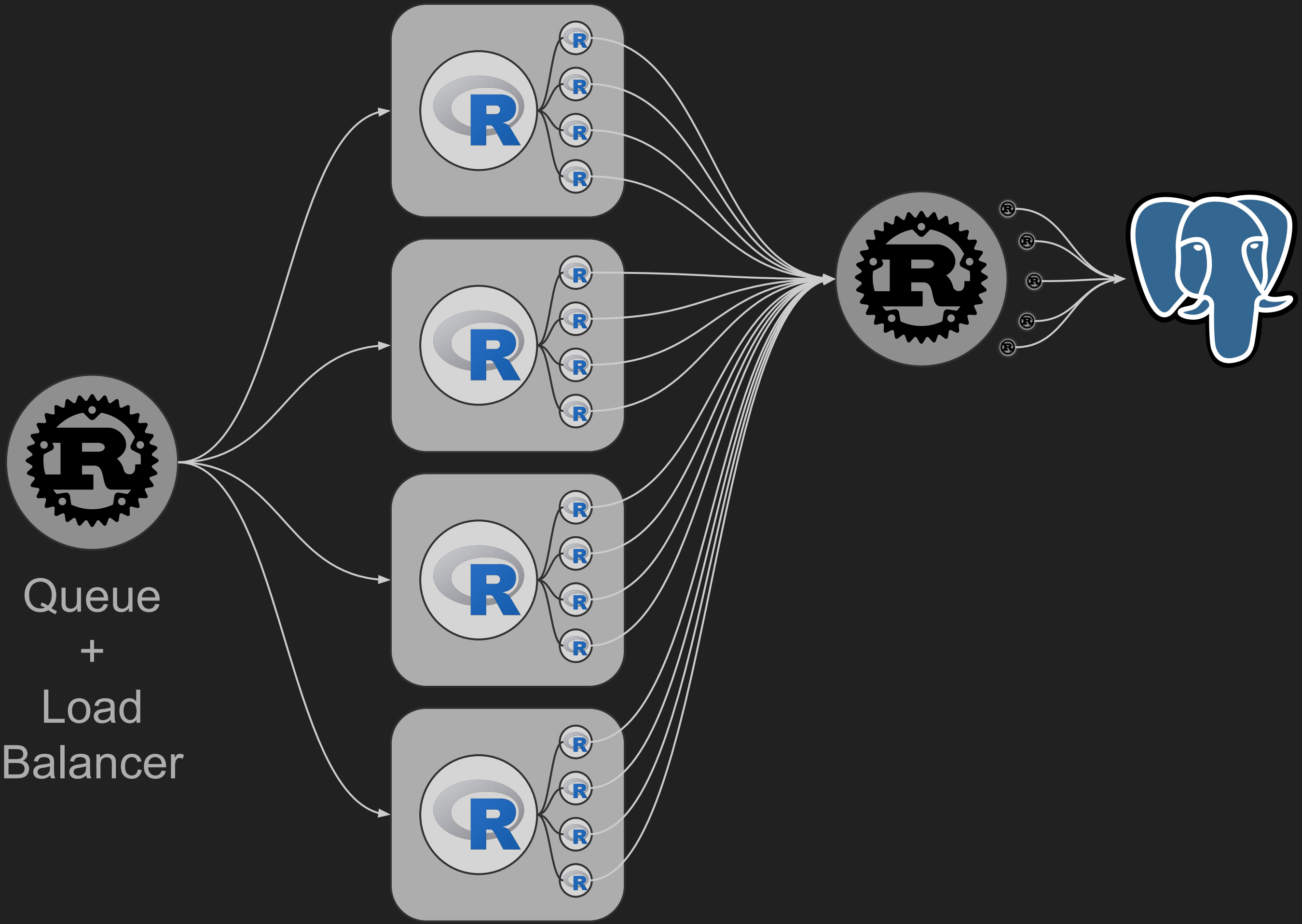
Using Futures/Promises

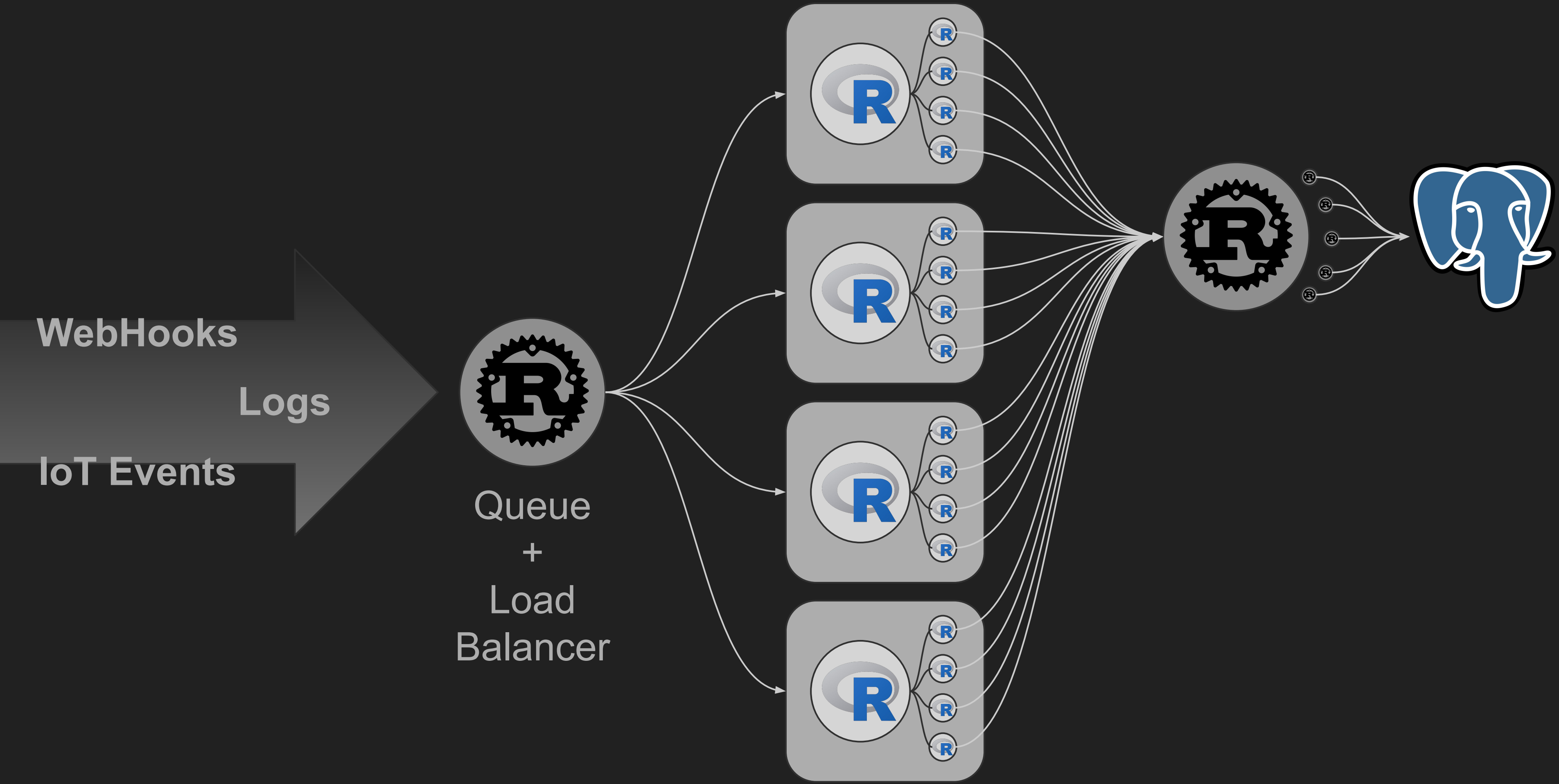
R cannot manage a single connection pool across threads.

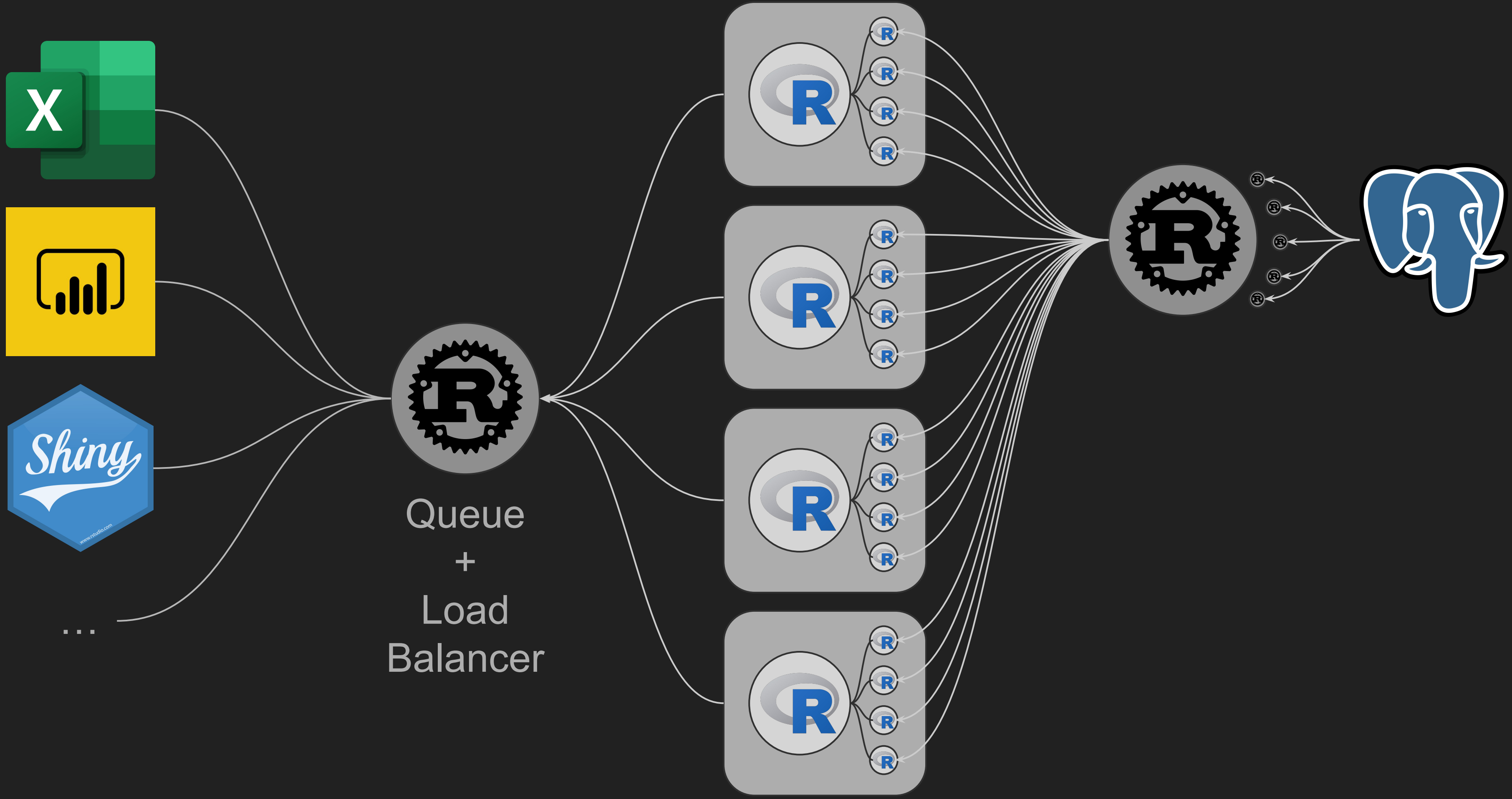
This means that on every sub-process (on every call to the future) we need to establish a new connection.






Queue
+
Load
Balancer


WebHooks
Logs
IoT Events
Queue
+
Load
Balancer


Shiny
...
Queue
+
Load
Balancer

Conclusions

- Rust is an amazing tool for making **fast**, **reliable** and **structured** R libraries.
- Rust can dramatically speed up a **Big Data** pipeline.
- Rust is not that hard.

Give Rust a try!