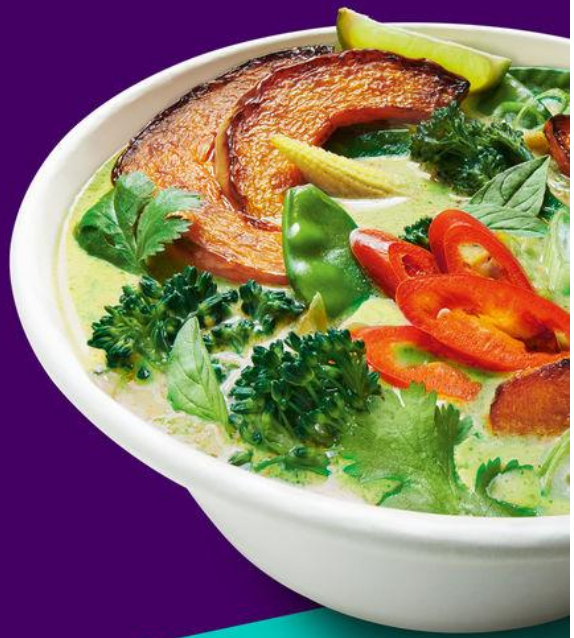
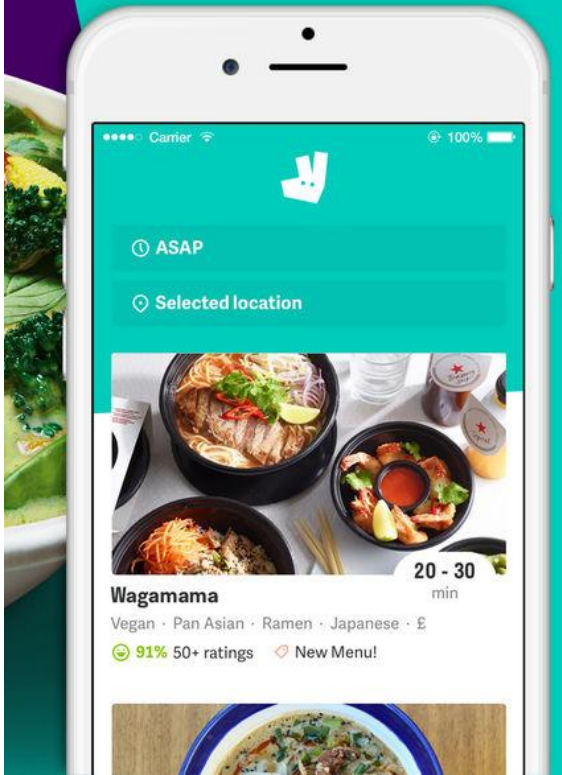
Deliveroo: Ruby to Rust

Michael Killough

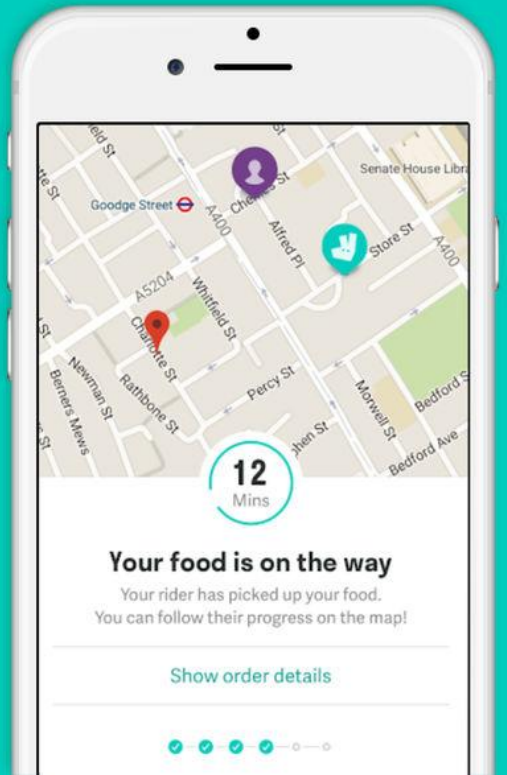
restaurant food, delivered



quickly browse great restaurants near you



track your rider right to your doorstep



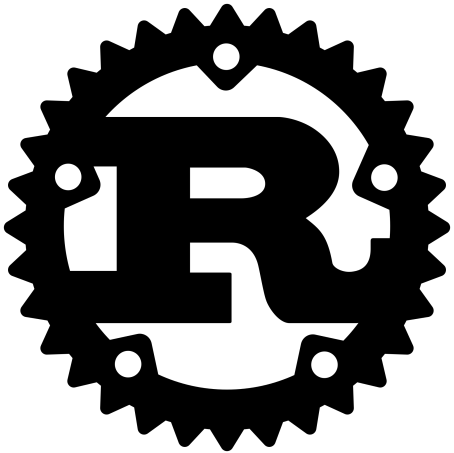
- Deliveroo is 5 years old
- Deliveroo originally a Ruby/Rails monolith
- Now multiple services in a distributed system
- Ruby/Rails is great

When will the food be ready?
How long will it take a rider to get from A to B?
Which order should we deliver first?
Which rider should we offer each order to?
Should one rider take two orders?

Vehicle Routing Problem

Dispatch

When CPU bound, Ruby is Slow



- Fast, safe, fearless concurrency
- No GC, no runtime
- Good library support
- We’re already using it in the Dispatch algorithm!

Files changed 100

+20,000 −20,000

Incremental!

extern crate ruru;

README:

> Have you ever considered rewriting some parts of your
> ~~slow~~ Ruby application?
>
> **Just replace your Ruby application with Rust, method**
> **by method, class by class.** It does not require you to
> change the interface of your classes or to change any
> other Ruby code.
>
> As simple as Ruby, as efficient as Rust.

```ruby
class Greeter
  def friendly_greeting(name)
    name = 'Anonymous' unless name.is_a?(String)
    "Hello #{name}!"
  end
End

# Expected: Hello MadRust!
Greeter.friendly_greeting("MadRust")
```

```rust
use ruru::RString;

class!(Greeter);

methods!(Greeter, itself,
    fn friendly_greeting(name: RString) -> RString {
        let name = name
            .map(|name| name.to_string())
            .unwrap_or("Anonymous".to_string());
        let greeting = format!("Hello {}!", name);
        RString::new(&greeting)
    }
);
```

```
use ruru::RString;

class!(Greeter);

methods!(Greeter, itself,
    fn friendly_greeting(name: RString) -> RString {
        let name = name
            .map(|name| name.to_string())
            .unwrap_or("Anonymous".to_string());
        let greeting = format!("Hello {}!", name);
        RString::new(&greeting)
    }
);
```

Needs to `impl ruru::Object`

```
use ruru::RString;

class!(Greeter);

methods!(Greeter, itself,
    fn friendly_greeting(name: RString) -> RString {
        let name = name
            .map(|name| name.to_string())
            .unwrap_or("Anonymous".to_string());
        let greeting = format!("Hello {}!", name);
        RString::new(&greeting)
    }
);
```

Actually a `Result<RString, _>`

```
use ruru::RString;

class!(Greeter);

methods!(Greeter, itself,
    fn friendly_greeting(name: RString) -> RString {
        let name = name
            .map(|name| name.to_string())
            .unwrap_or("Anonymous".to_string());
        let greeting = format!("Hello {}!", name);
        RString::new(&greeting)
    }
);
```

Returning complex types (like errors) is hard

```ruby
class Order
    attr_reader :id, :contains_alocohol?

    def delivery_location
        {latitude: 0.0, longitude: 0.0}
    end

    # ...
end
```

```
methods!(Dispatcher, itself,
    fn generate_plan(orders: Array, /* ... */) -> AnyObject {
        let orders = orders
            .expect("not actually an Array")
            .into_iter()
            .map(|element| {
                Order {
                    // `element.id` in Ruby
                    id: element.send("id", vec![]),
                    // ...
                }
            });
        /* generate plan */
    }
);
```

Serde!

```rust
#[derive(Deserialize, Serialize)]
struct Order {
    id: u64,
    delivery_location: Location,

    #[serde(rename = "contains_alcohol?")]
    contains_alcohol: bool,

    // ...
}
```

```rust
methods!(Dispatcher, itself,
    fn generate_plan(orders: Vec<Order>, /* ... */) -> Plan {
        // orders is actually a Vec<Order>!
        let plan = /* generate plan */;
        plan
    }
);
```

`extern crate ruru_serde;`

- Uses Ruby's `rb_protect` to safely call back into Ruby
  - Requires unreleased `ruru` / `ruby-sys` crates
- Serde to convert arguments/return types
  - Re-raises errors as Ruby exceptions with extra debug context
- Catch Rust panics and safely re-raise as Ruby exceptions
- Allows returning `Result<_, Into<Exception>>`

```
# Exceptions as seen from Ruby:

Dispatcher::RustError:
    undefined method ‘contains_alcohol?' for nil:NilClass
    Context from Rust:
        - While deserializing "contains_alcohol"
        - While deserializing "orders"
```

Piece by Piece

- Shipped changes in ~10 stages
- Everything was feature-flagged
  - Initially enabled for only a small % of users
- Monitored for crashes, automatically switch back to Ruby

Result

- No downtime!
- >12.5x faster
  - What was taking 10s of seconds now <1 second
  - Probably much bigger improvement if we compare against pure Ruby
  - Without profiling/optimisation
- We've been able to extend the algorithm in ways we couldn't in Ruby
  - Better matching of rider/order, and better experiences for riders and customers

deliveroo

Thanks!