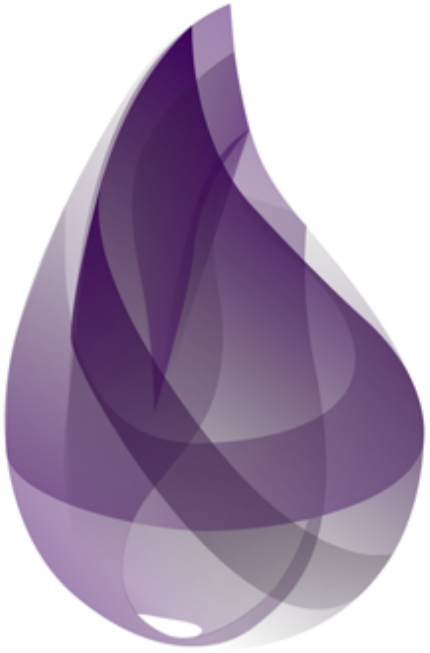
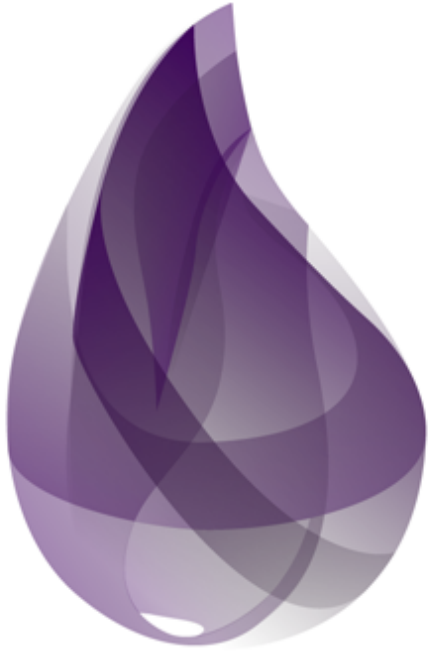
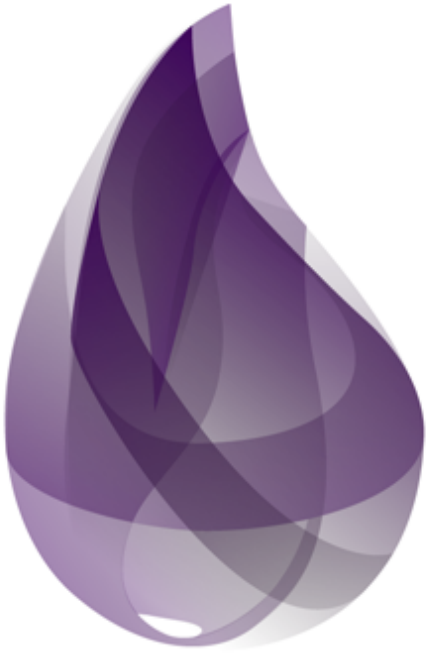


elixir

functional



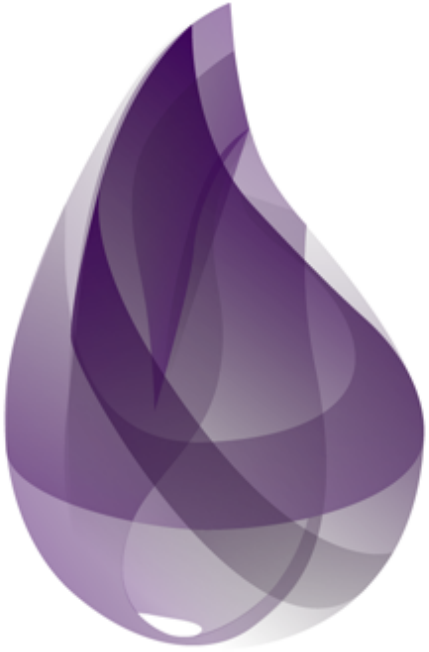
elixir



elixir

functional

ErlangVM



elixir

functional

ErlangVM

Parallelism

Who wants to go and
build a system in Elixir?

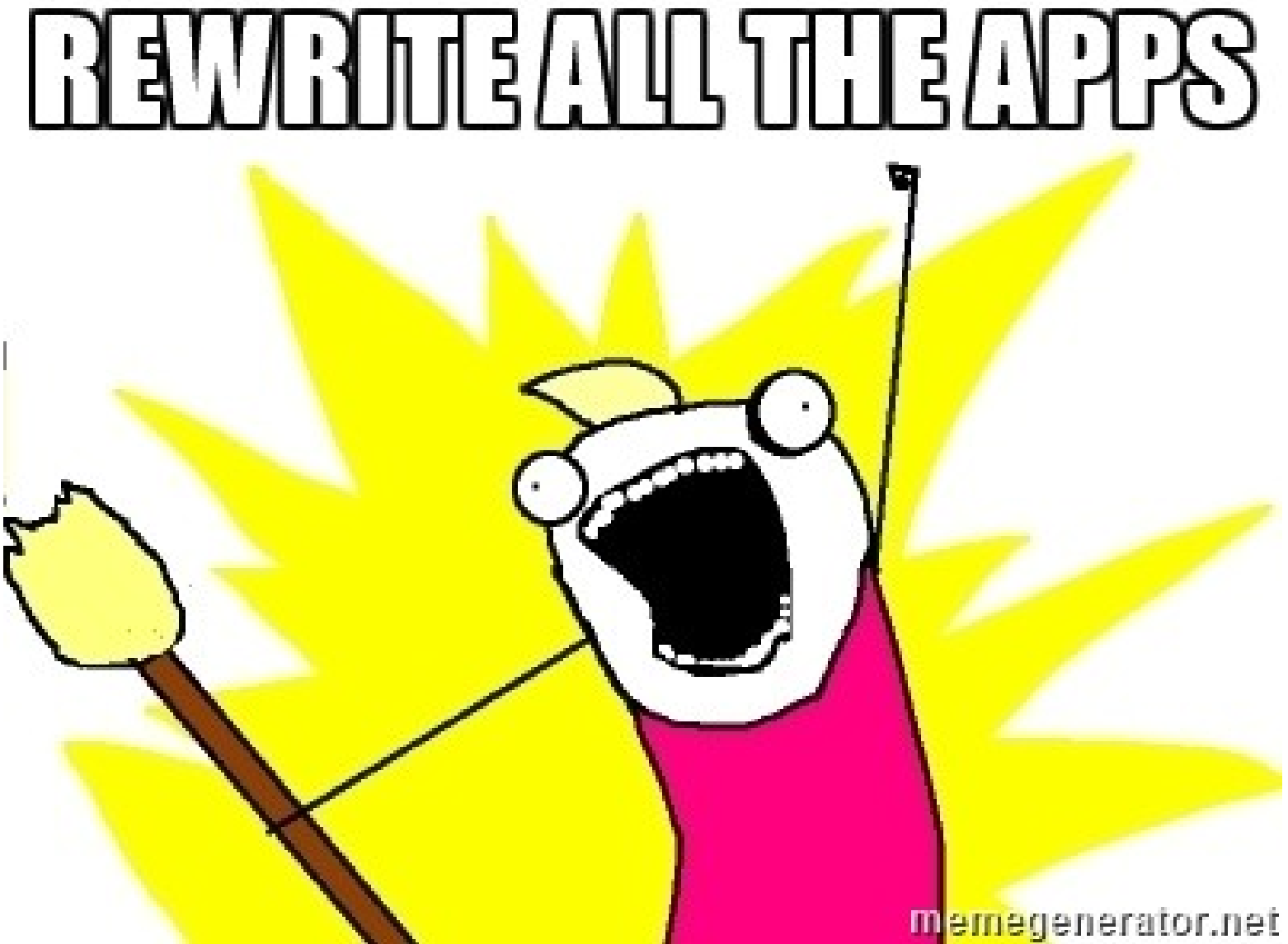
Dave Thomas
@pragdave

Following

Reading patterns books is like watching drug ads on TV. By the end, you're convinced you have all the symptoms and need all the remedies.

RETWEETS	LIKES
93	128

11:47 PM - 25 Oct 2016


REWRITE ALL THE APPS
memegenerator.net

YOU CAN'T

YOU SHOULDN'T

You shouldn’t

Elixir, Your Monolith and You

Tobias Pfeiffer
@PragTob
pragtob.info

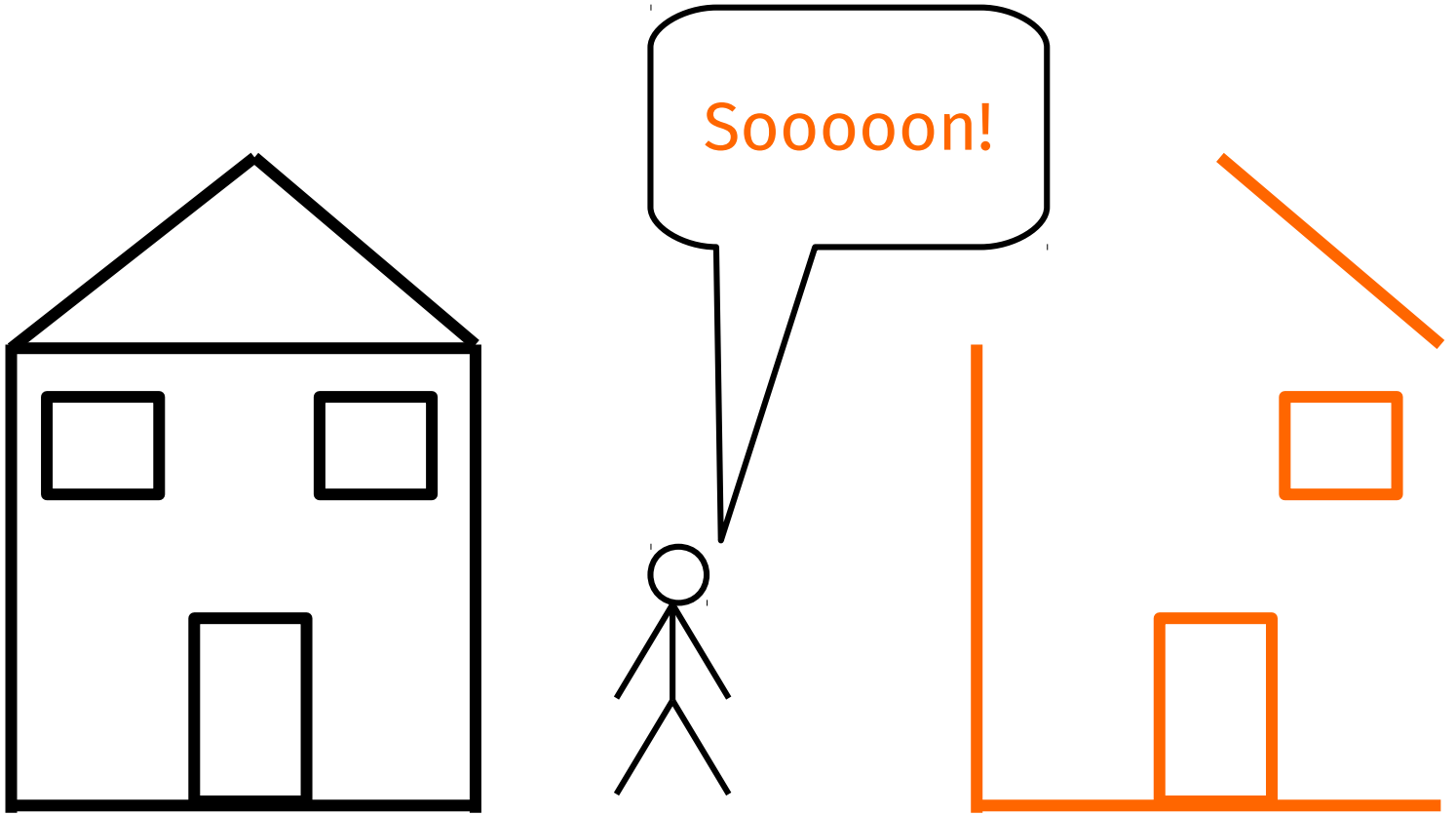
Your Monolith and You


Your Monolith now?


Your Monolith?

Tear it all down and
rewrite it


SENORGIE.COM


Sooooon!

"We rewrote it all in Language x, now it's 10 times faster and only 50% of the code!"

Someone

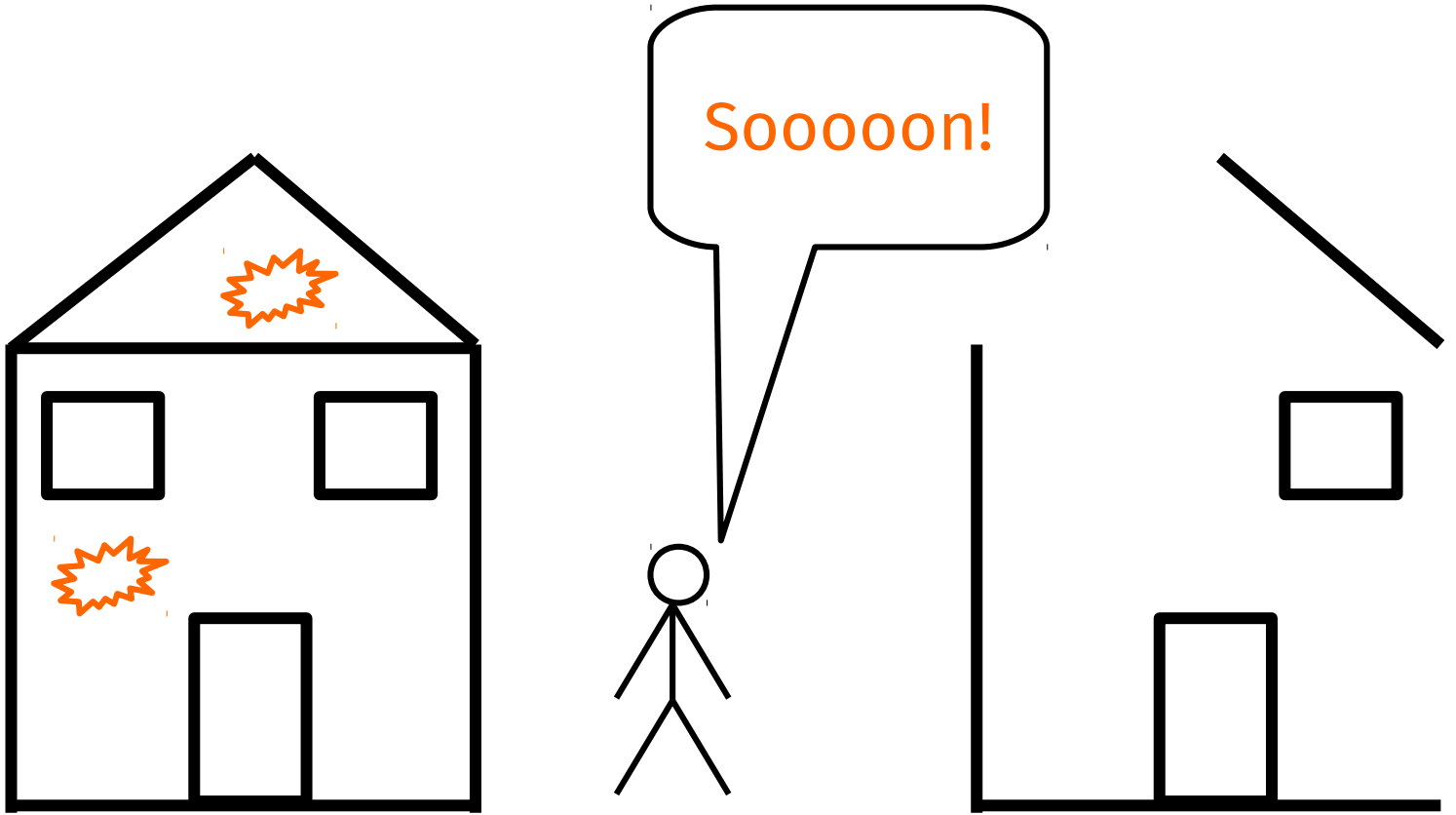
Often implies...

"Language X, is 10 times faster and requires 50% of the code of language Y!"

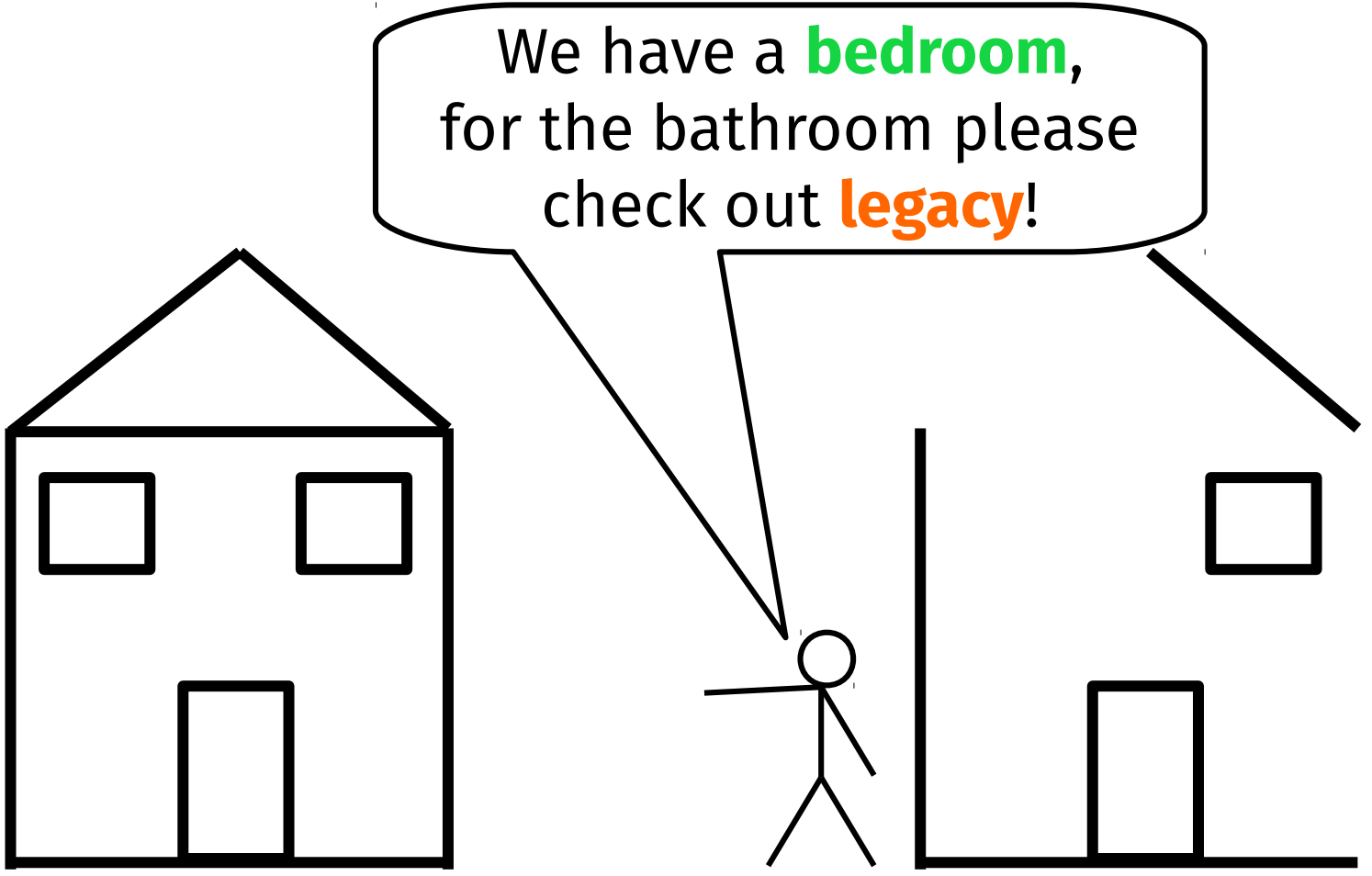
Someone

What nobody tells you…


Business Value?


Sooooon!

Replace it step by step


We have a bedroom,
for the bathroom please
check out legacy!

poteto/terraform

Terraform allows you to incrementally transform an older API into one powered by Phoenix - one endpoint at a time.

poteto/terraform

allows you to

incre *ansform an*

older API in *ered by*

Phoenix - one e *g*

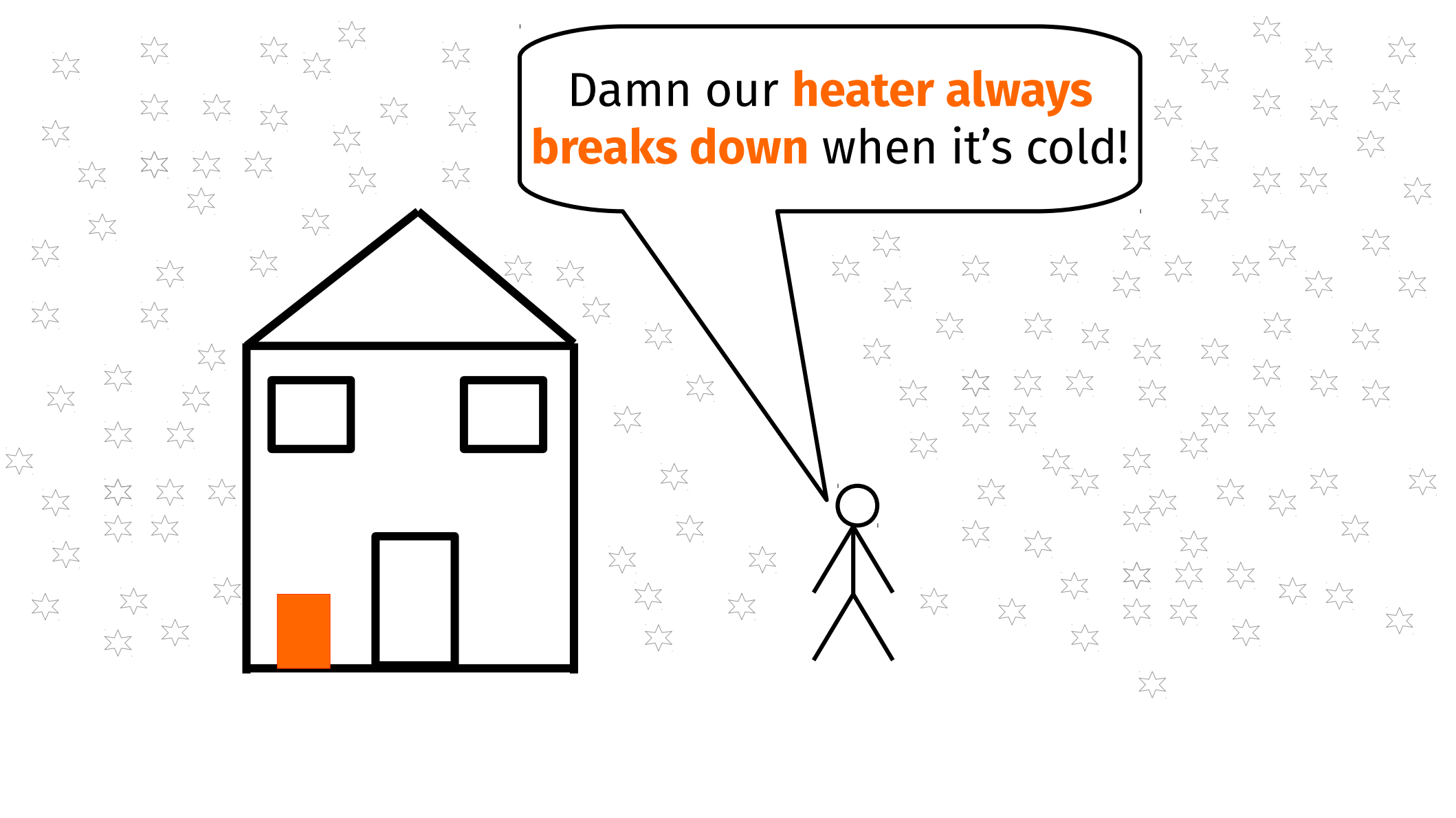
time.

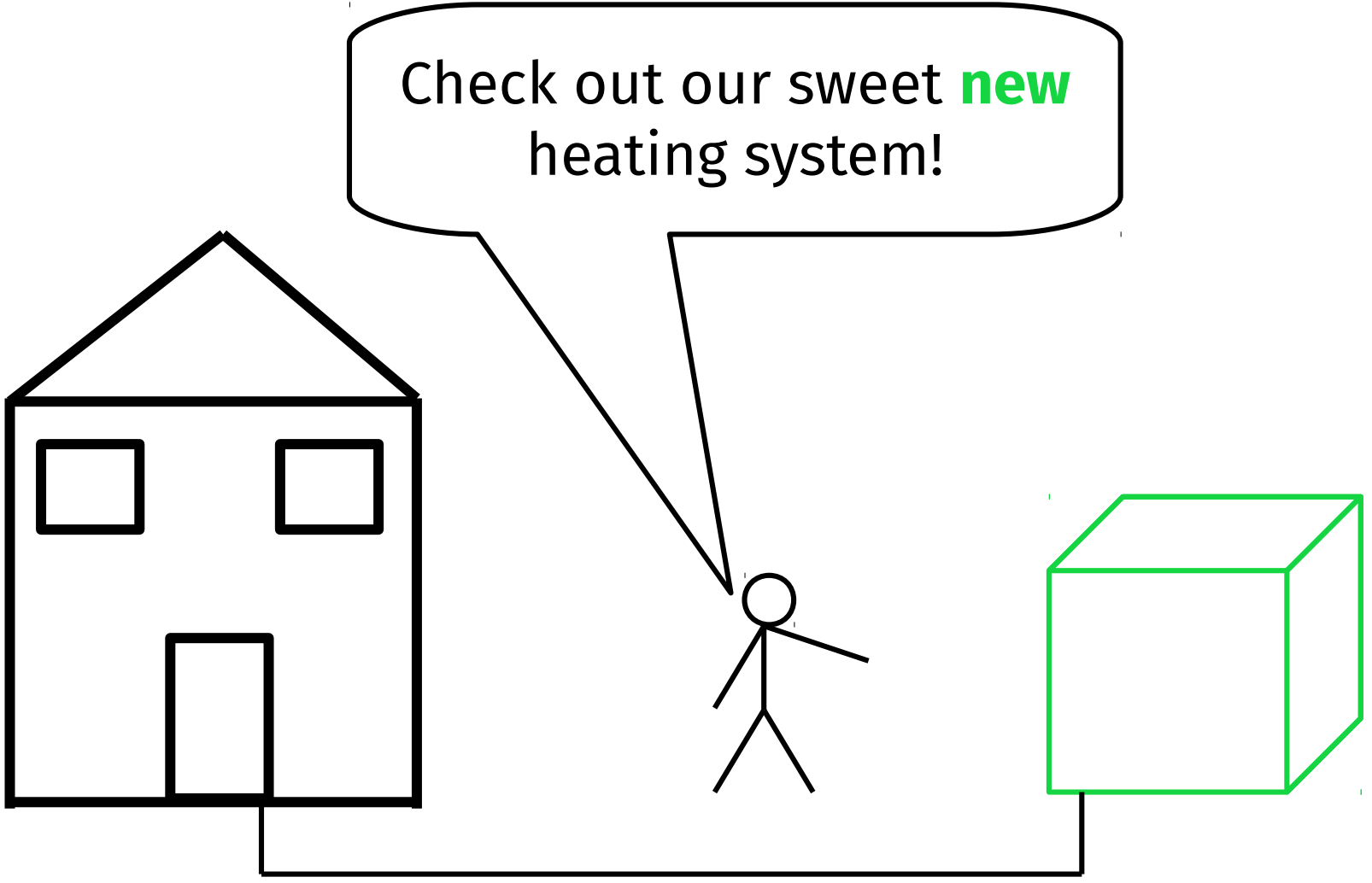
Not the Infrastructure As Code

poteto/terraform

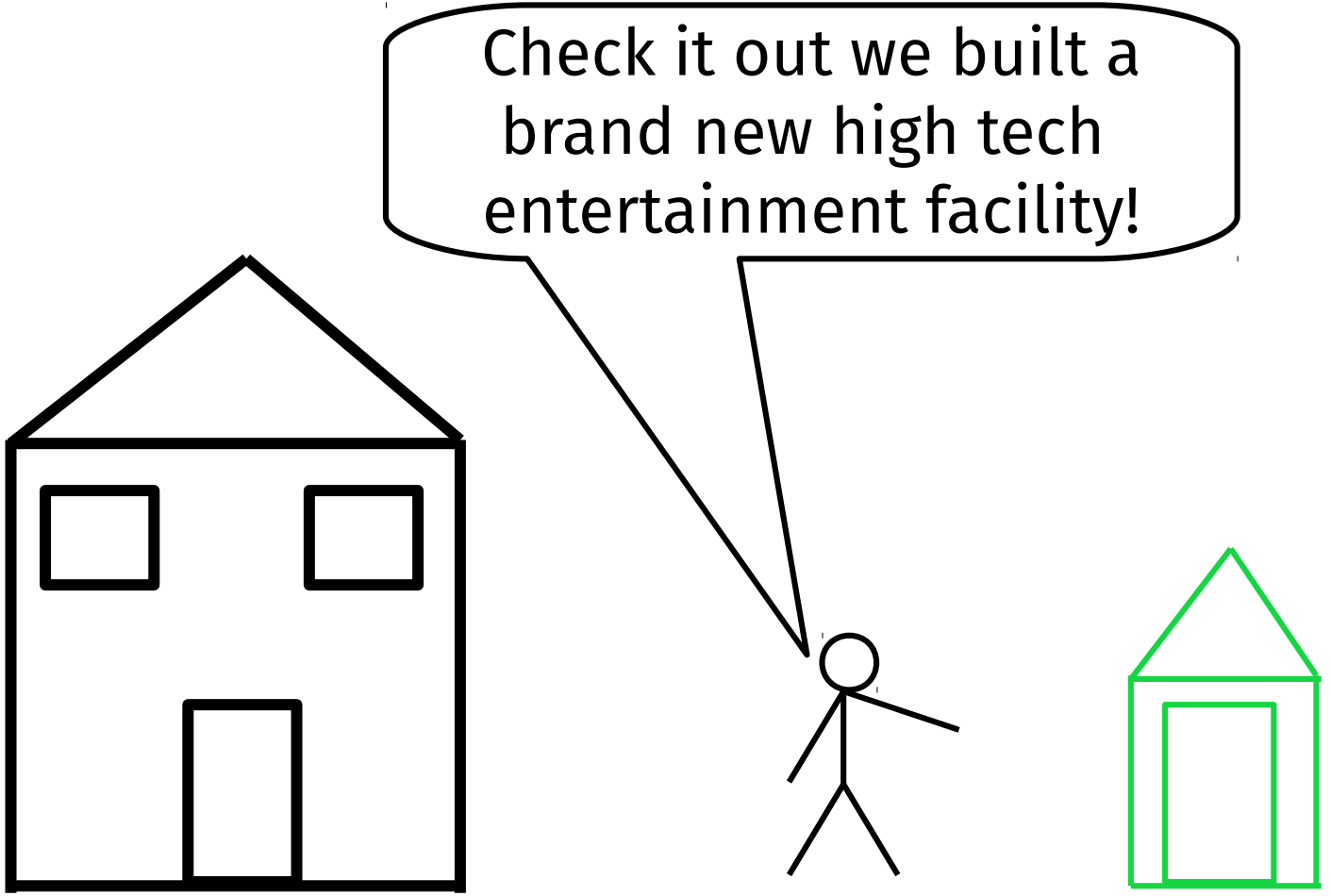
```
defmodule MyApp.Terraformers.Foo do
  alias MyApp.Clients.Foo
  use Plug.Router

  plug :match
  plug :dispatch

  get "/v1/hello-world", do: send_resp(conn, 200, "Hi")

  get _ do
    res = Foo.get!(conn)
    send_response({:ok, conn, res})
  end
end
```

Replace a critical component


Damn our heater always breaks down when it's cold!


Check out our sweet new heating system!

Write a new component


Check it out we built a brand new high tech entertainment facility!

Obvious in microservices

Write a new component

*"Is the **technology** I'm choosing the **right fit** for the **problem** I'm trying to solve?"*

Everyone, hopefully

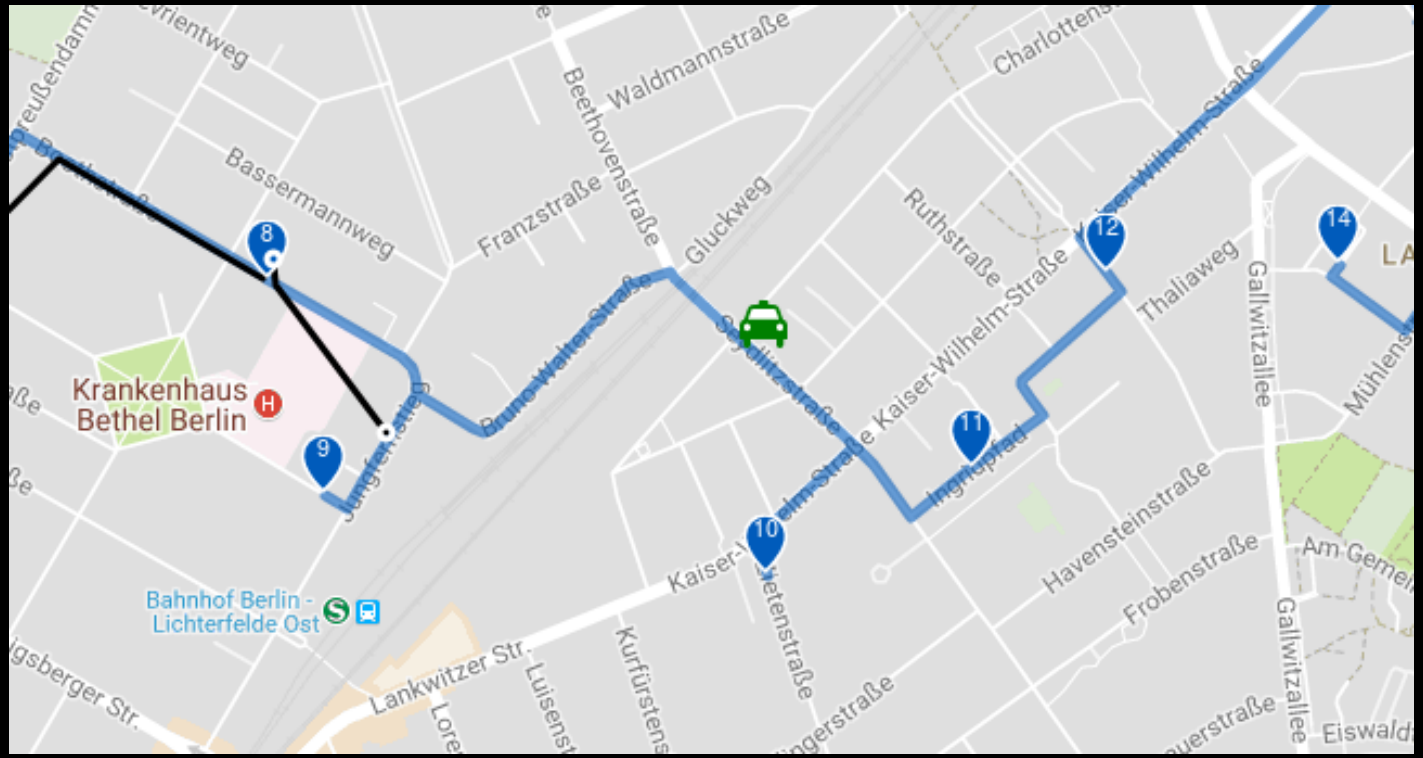
Reasonably separate and limited problem

Requirement that poses a problem for the current stack

Spark of Enthusiasm

Courier Tracking

Courier Tracking



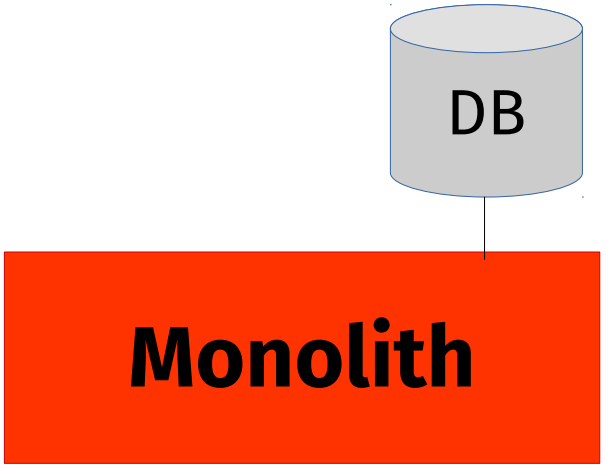
Courier Tracking

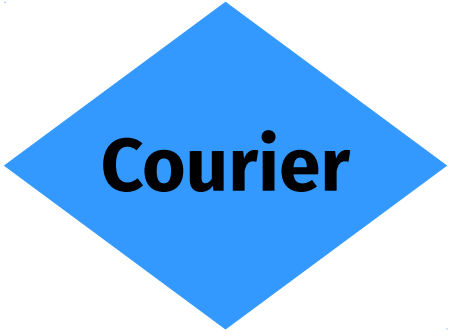
Courier Tracking

Websockets

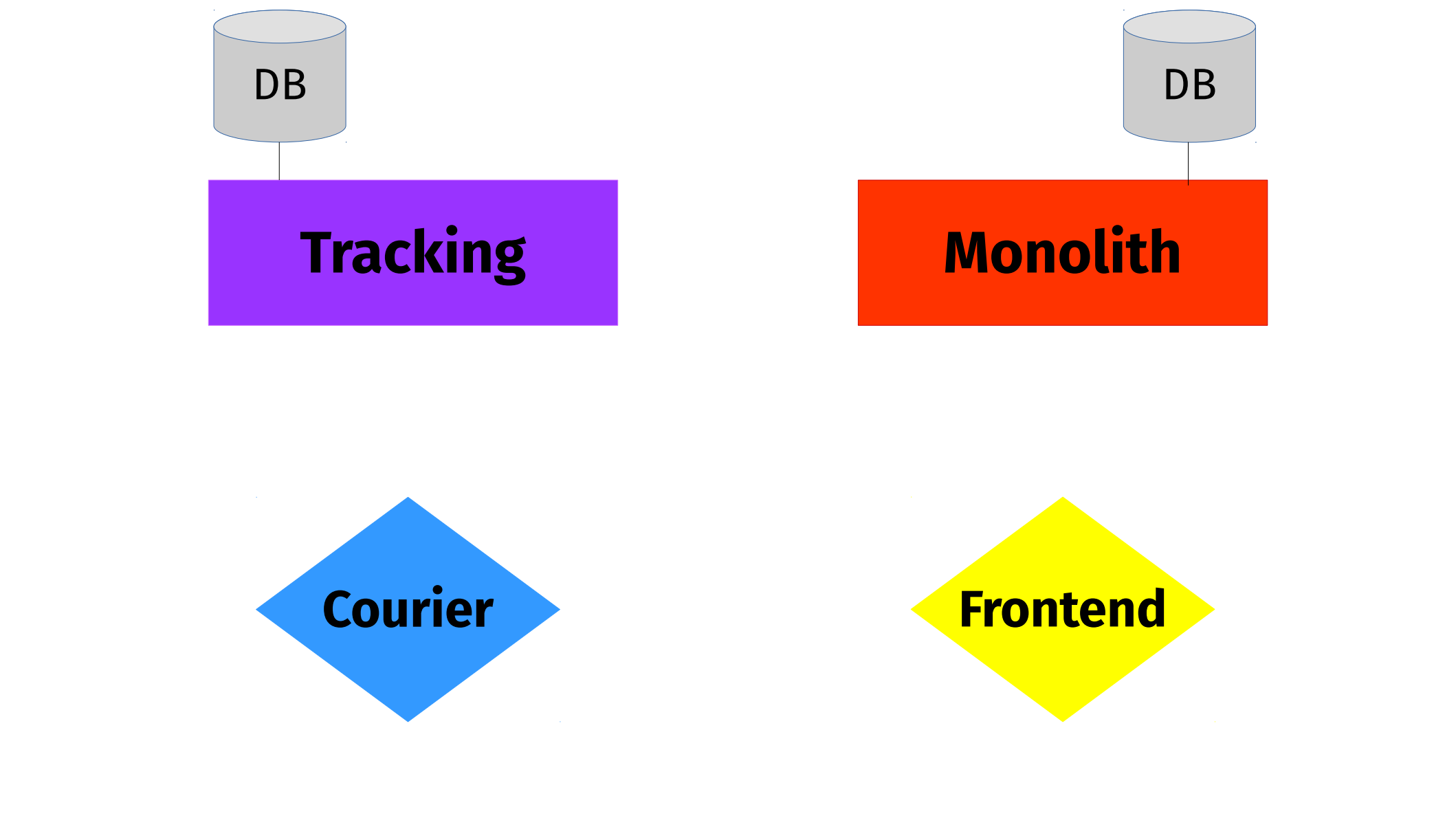

Courier Tracking
Websockets
Basic Knowledge
LIEFERY
KURIERDIENST
www.liefery.com
HH LF 6009

How to make them talk?

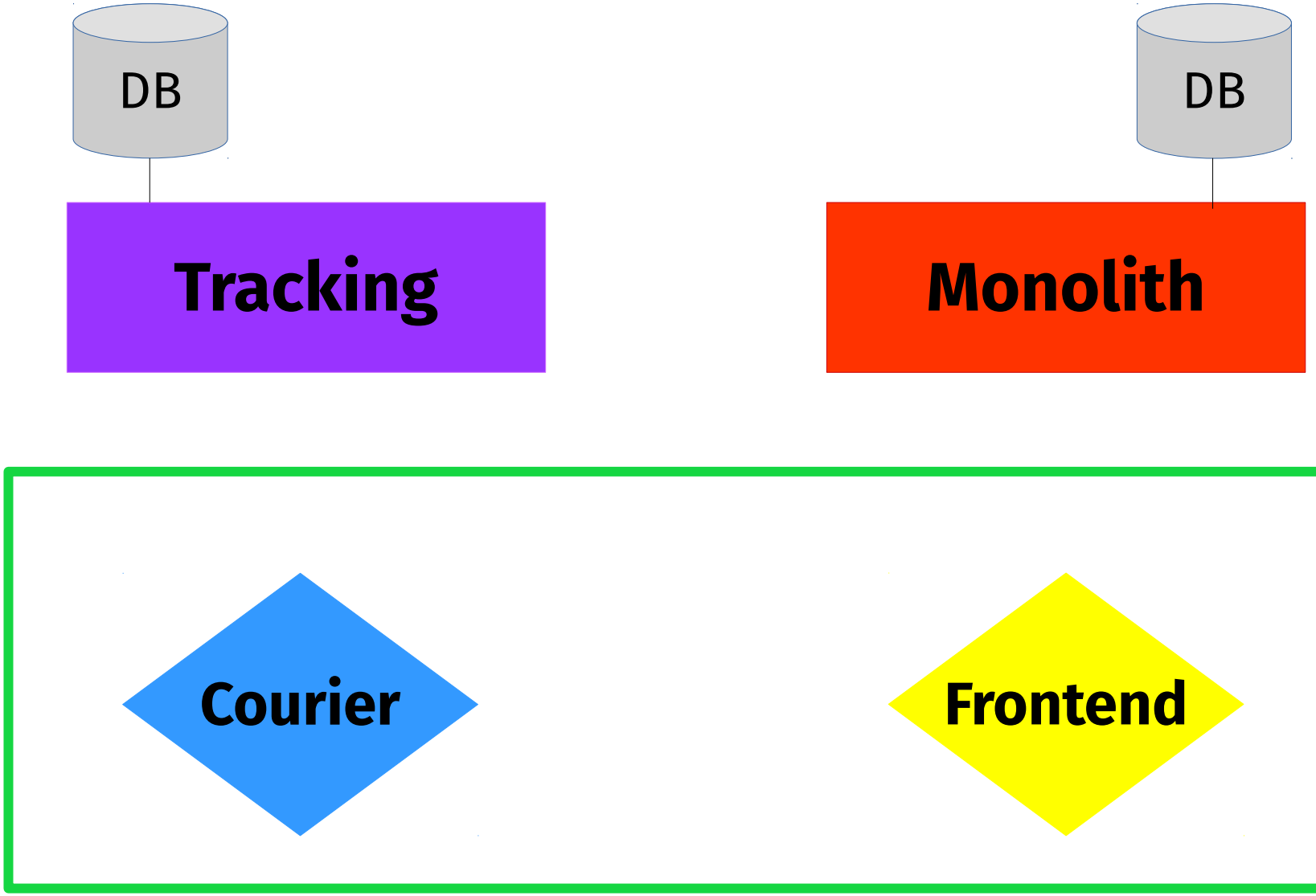

DB
Monolith


Courier


Frontend


DB
Tracking
DB
Monolith
Courier
Frontend

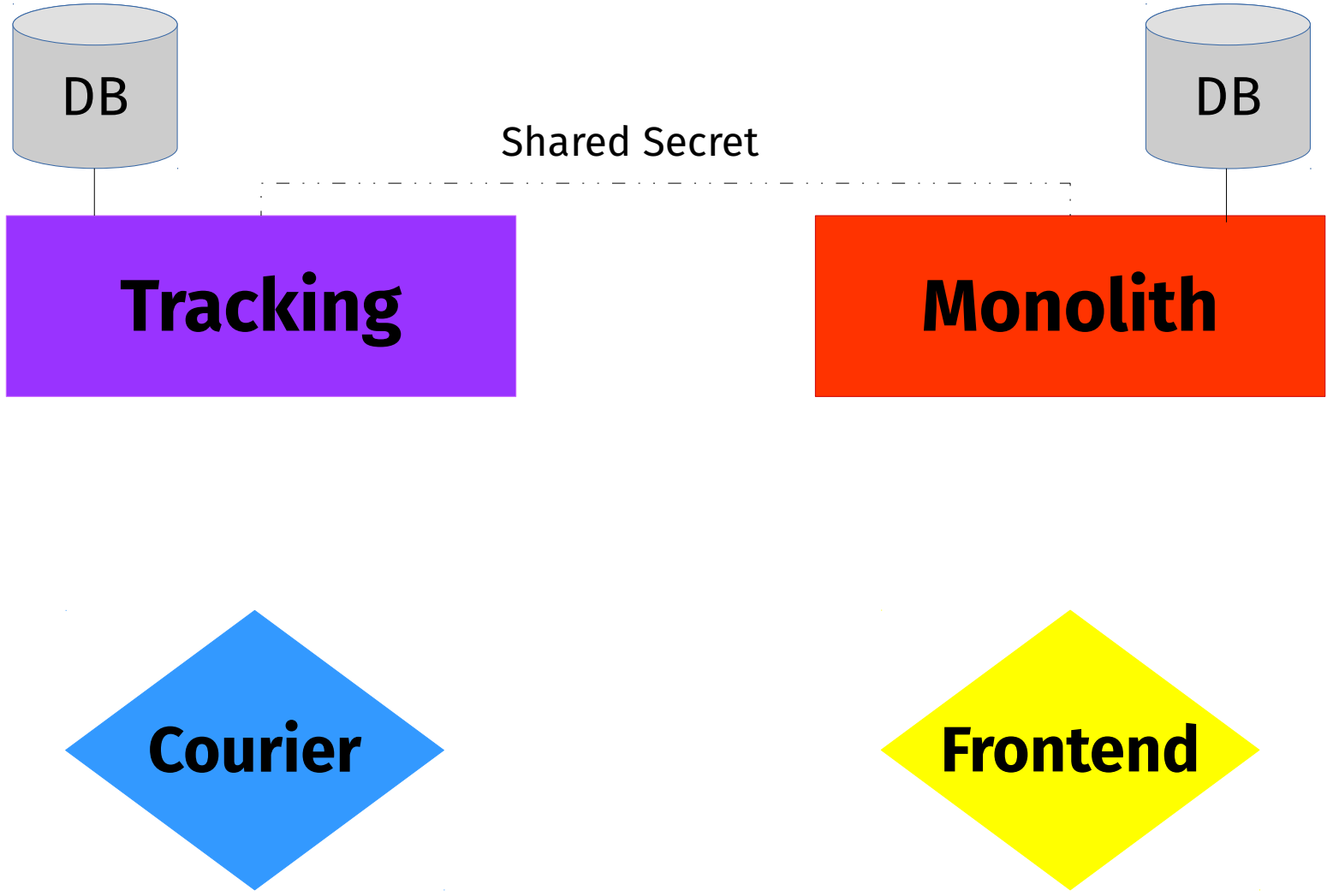
Connect them?

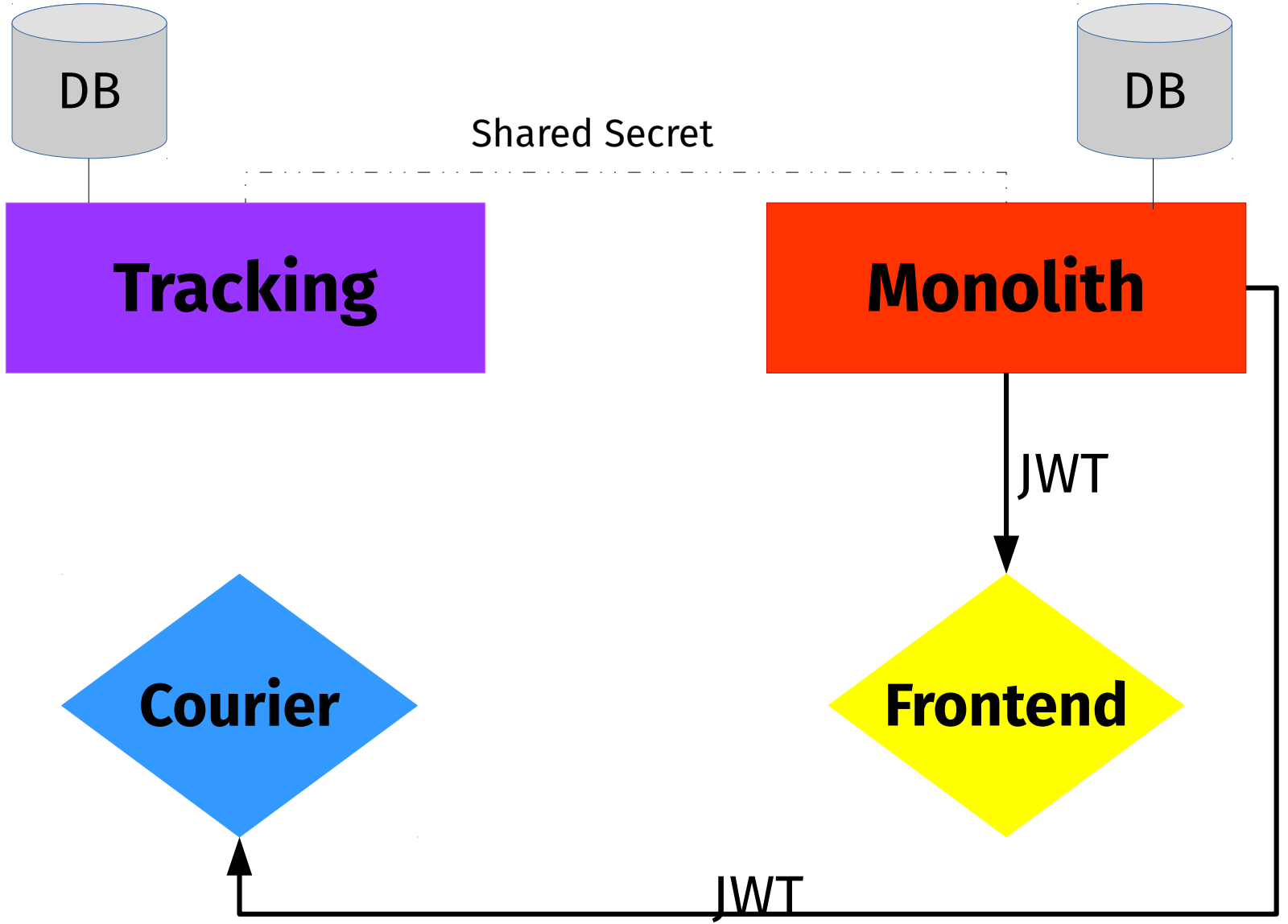

DB
DB
Tracking
Monolith
Courier
Frontend

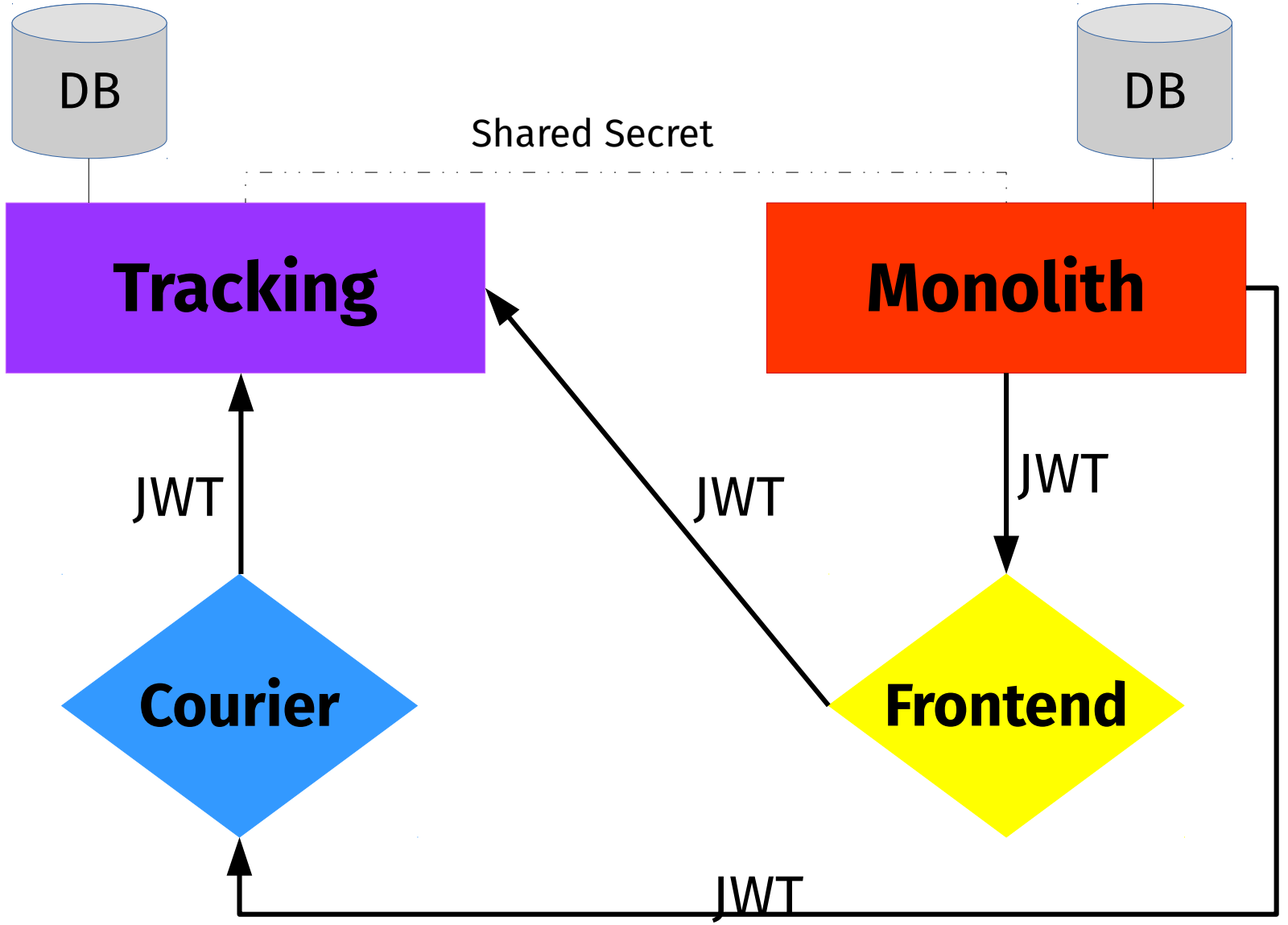
JSON
Web
Token

Header
Payload
Signature

```
{
  "role":        "admin",
  "courier_ids": [1337, 55],
  "id":          42,
  "exp":         1520469475
}
```


DB
DB
Shared Secret
Tracking
Monolith
Courier
Frontend


DB
DB
Shared Secret
Tracking
Monolith
JWT
Courier
Frontend
JWT


DB
DB
Shared Secret
Tracking
Monolith
JWT
JWT
JWT
Courier
Frontend
JWT

Socket

```
def connect(%{"token" => token}, socket) do
  token
  |> JWT.verify_token
  |> respond_based_on_token_contents(socket)
end
```

Pattern Match

```
def connect(%{"token" => token}, socket) do
  token
  |> JWT.verify_token
  |> respond_based_on_token_contents(socket)
end
```

Pipe

```
def connect(%{"token" => token}, socket) do
  token
  |> JWT.verify_token
  |> respond_based_on_token_contents(socket)
end
```

Socket

```
defp login_user(%{"role" => role, ...}, socket) do
  {:ok, assign(socket, :user, %{role: role, ...})}
end
```

Channel

```
def join("courier:" <> courier_id, _,
         %{assigns: %{user: user}}) do
  if authorized?(courier_id, user) do
    # login
  else
    # unauthorizd
  end
end
```

Channel

```
defp authorized?("", _), do: false
defp authorized?(courier_id, %{courier_ids: ids}) do
  Enum.member?(ids, String.to_integer(courier_id))
end
defp authorized?(_, _), do: false
```

Channel

```
defp authorized?("", _), do: false
defp authorized?(courier_id, %{courier_ids: ids}) do
  Enum.member?(ids, String.to_integer(courier_id))
end
defp authorized?(_, _), do: false
```

Channel

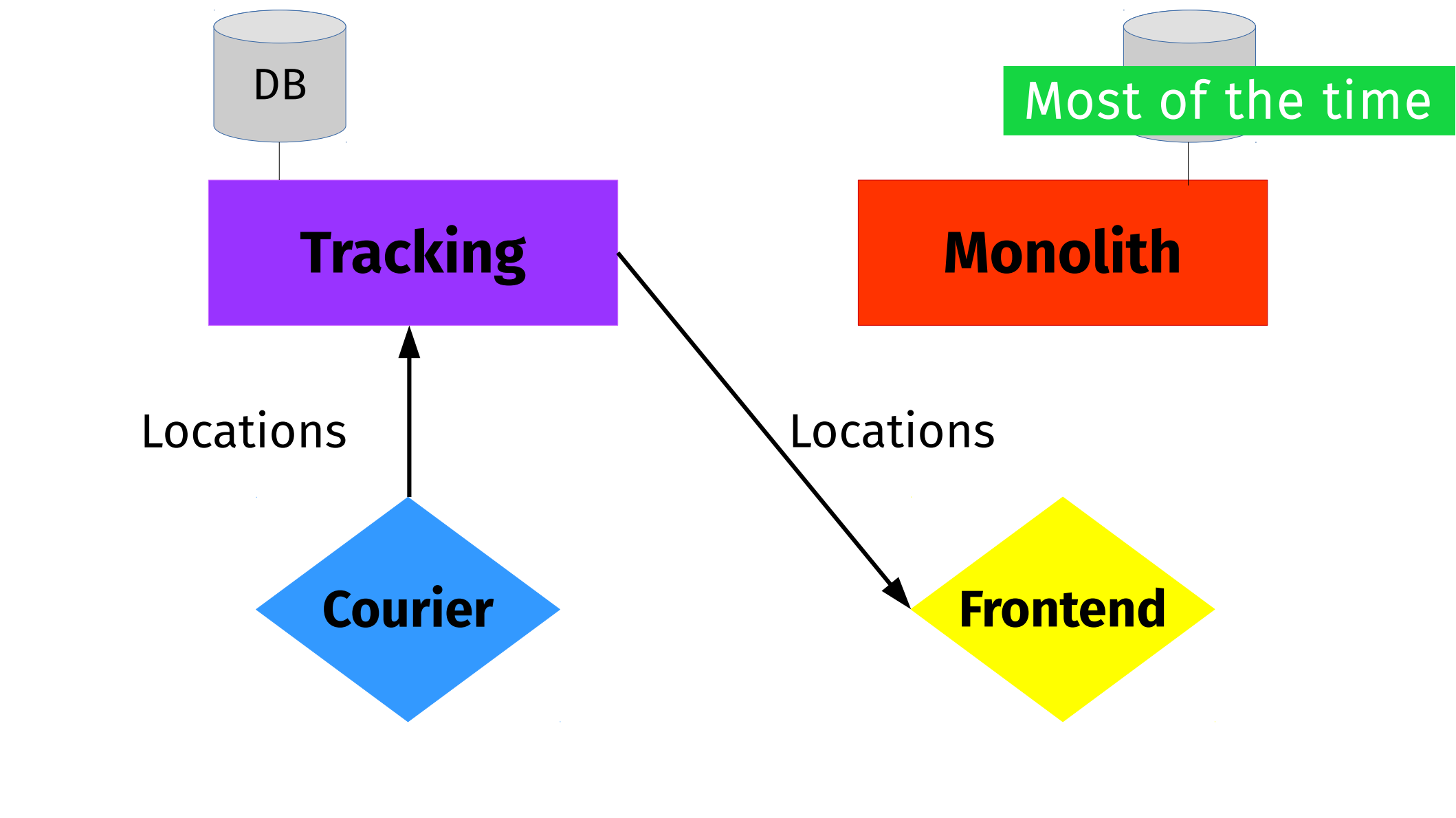
```
defp authorized?("", _), do: false
defp authorized?(courier_id, %{courier_ids: ids}) do
  Enum.member?(ids, String.to_integer(courier_id))
end
defp authorized?(_, _), do: false
```

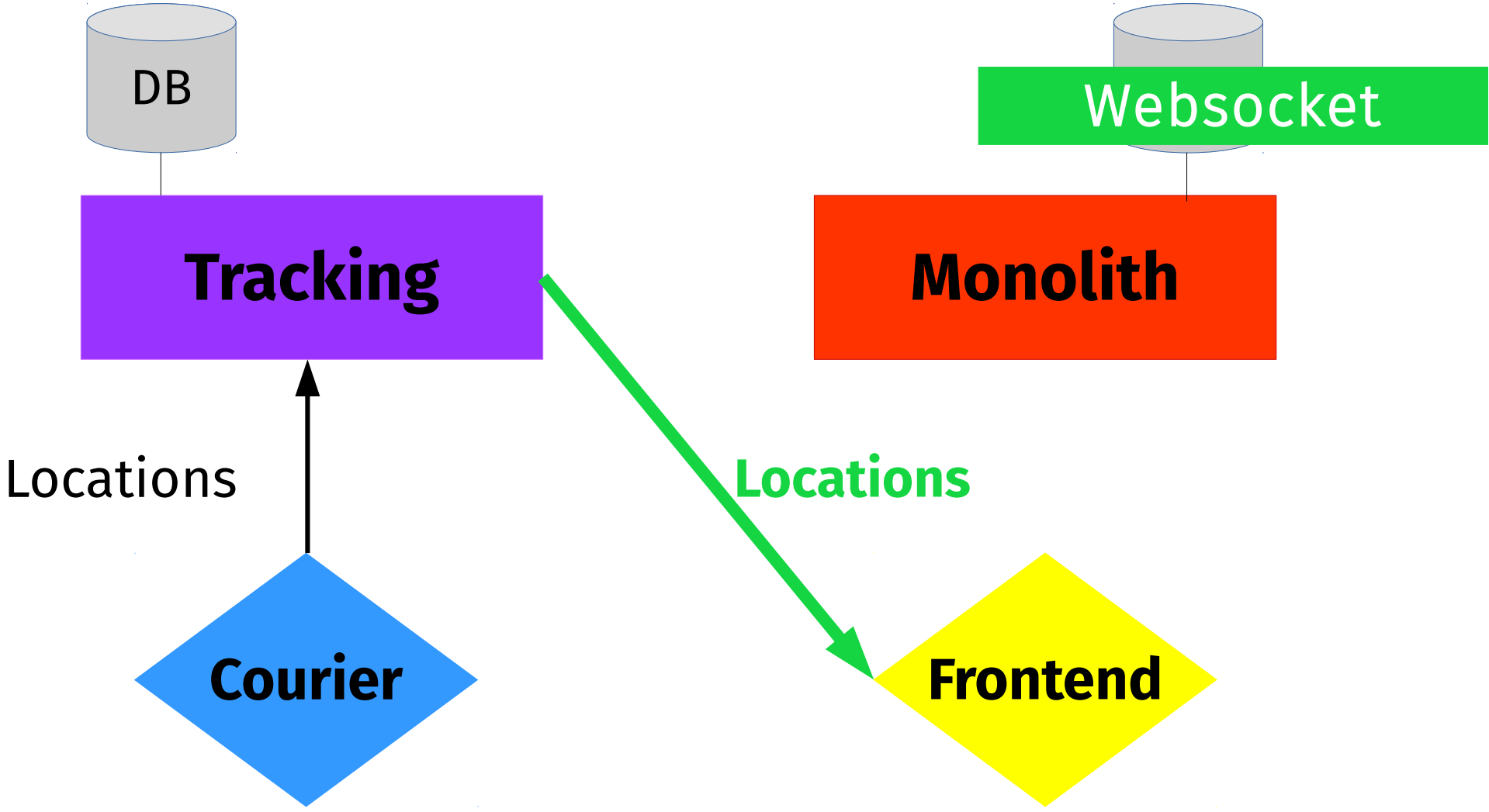
Channel

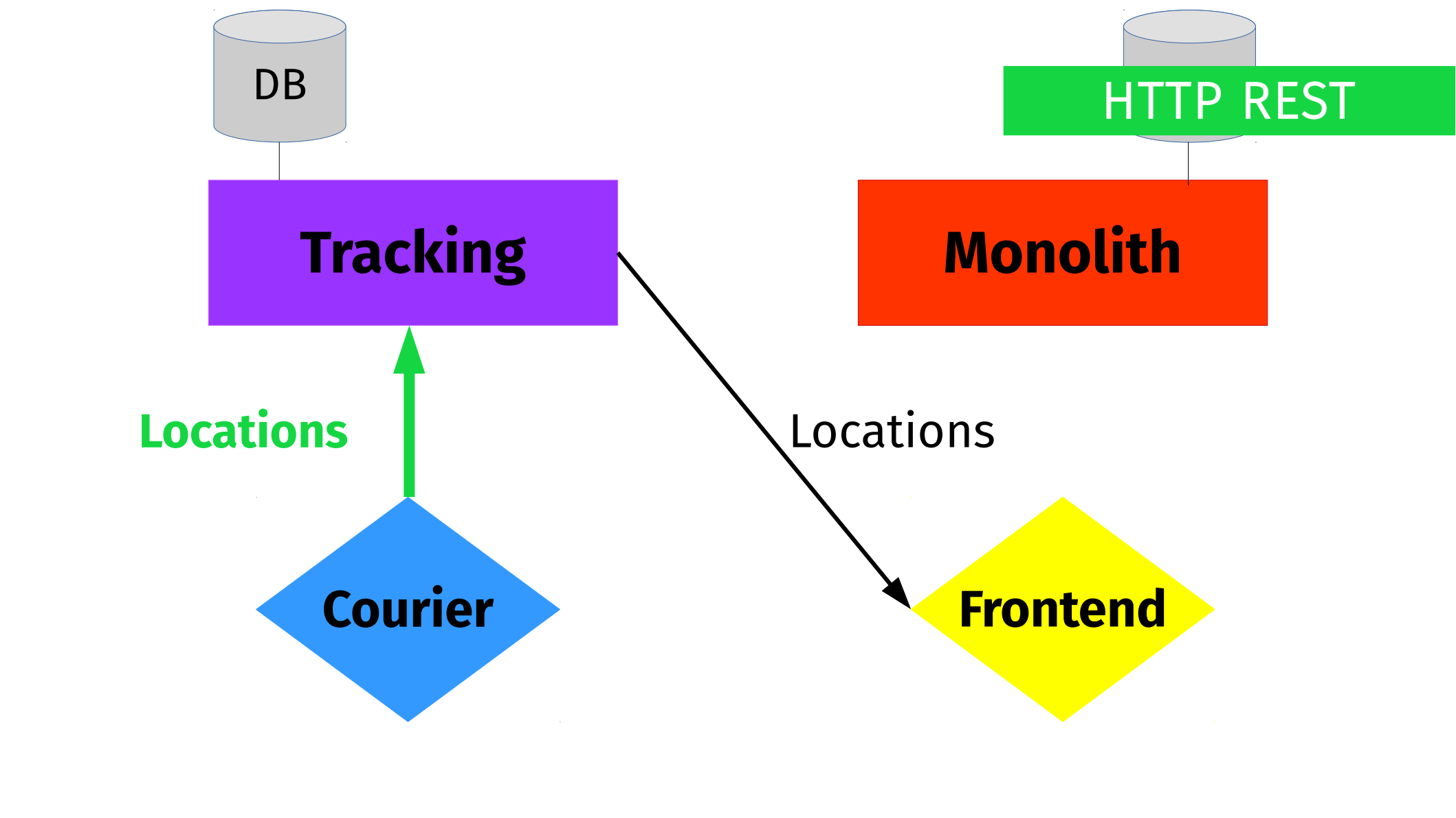
```
defp authorized?("", _), do: false
defp authorized?(courier_id, %{courier_ids: ids}) do
  Enum.member?(ids, String.to_integer(courier_id))
end
defp authorized?(_, _), do: false
```

Channel

```
defp authorized?("", _), do: false
defp authorized?(courier_id, %{courier_ids: ids}) do
  Enum.member?(ids, String.to_integer(courier_id))
end
defp authorized?(_, _), do: false
```


DB
Most of the time
Tracking
Monolith
Locations
Locations
Courier
Frontend

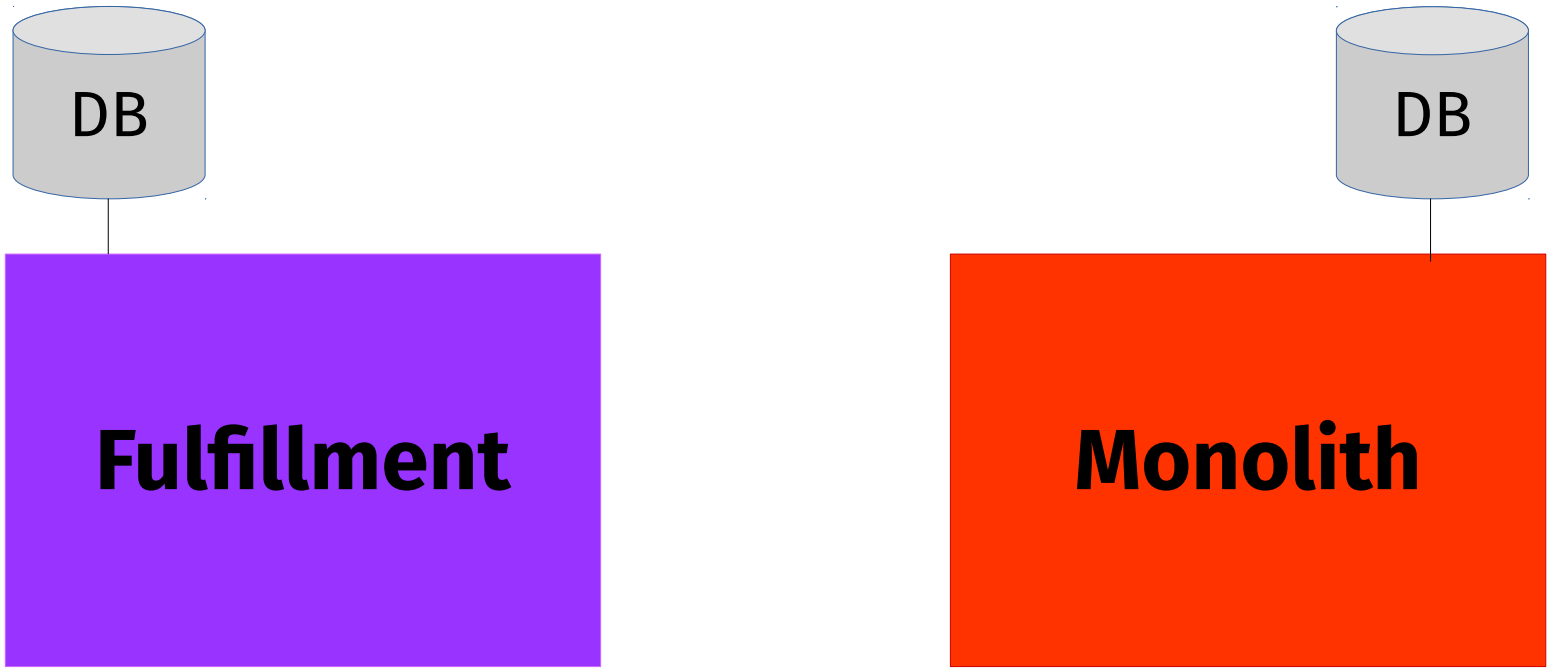

DB
Websocket
Tracking
Monolith
Locations
Locations
Courier
Frontend


DB
HTTP REST
Tracking
Monolith
Locations
Locations
Courier
Frontend

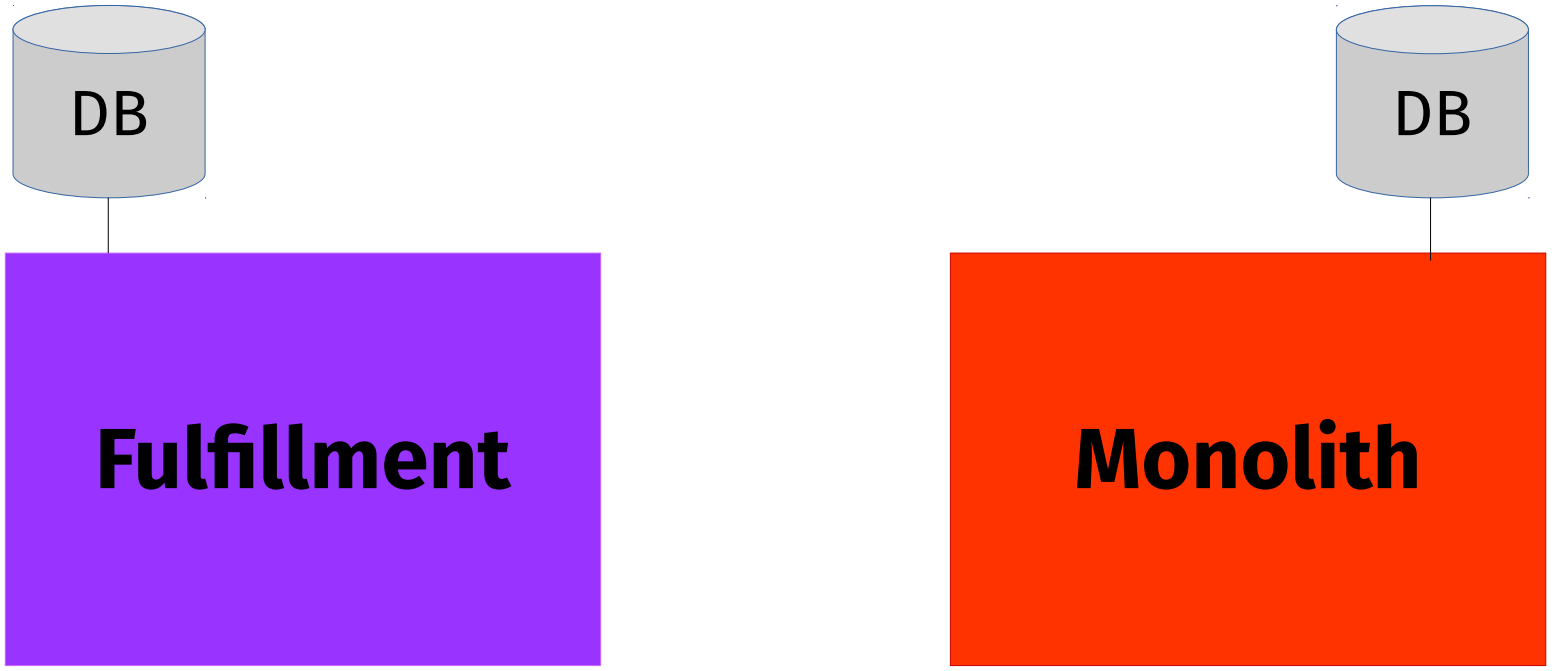

Fulfillment
2
aisle
pasillo
STAND
1
45


Fulfillment
CRUD
2 aisle
pasillo

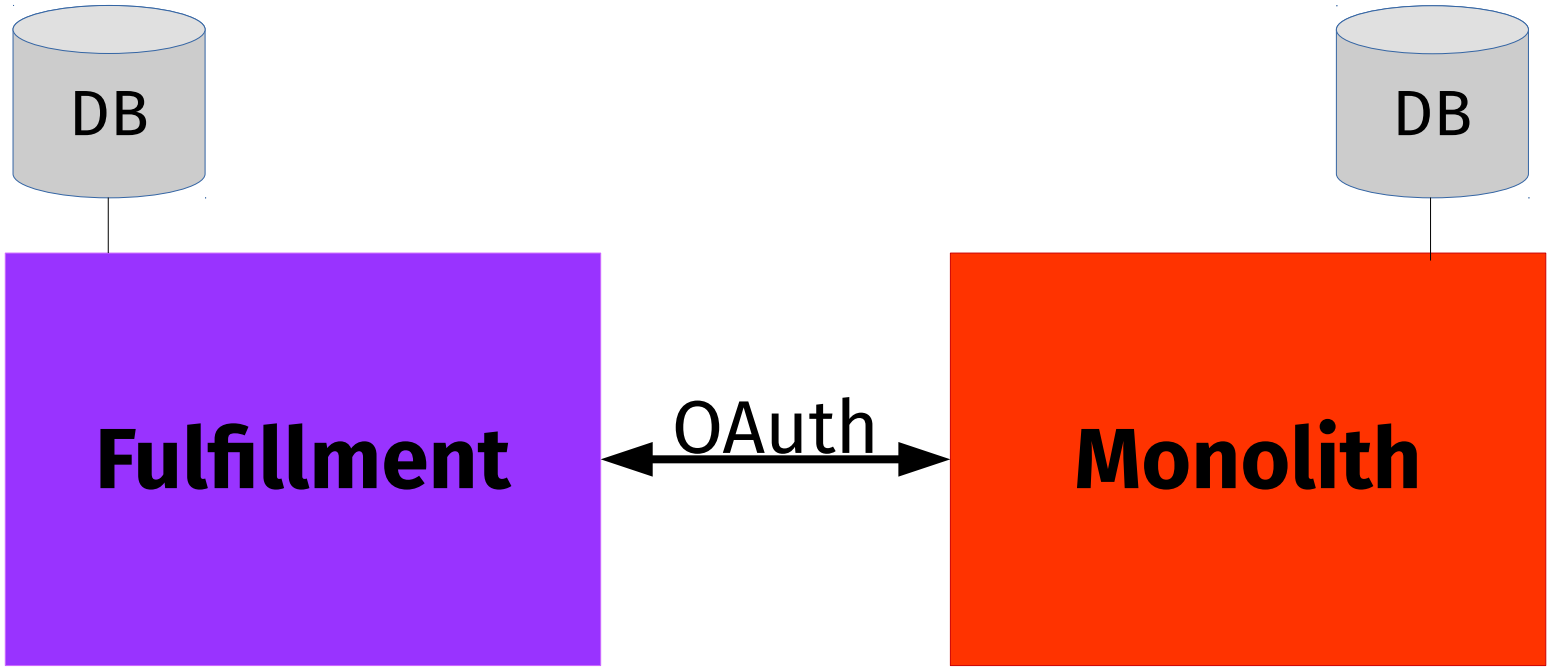

Fulfillment
CRUD
Experience
2 aisle pasillo


DB
Fulfillment
DB
Monolith

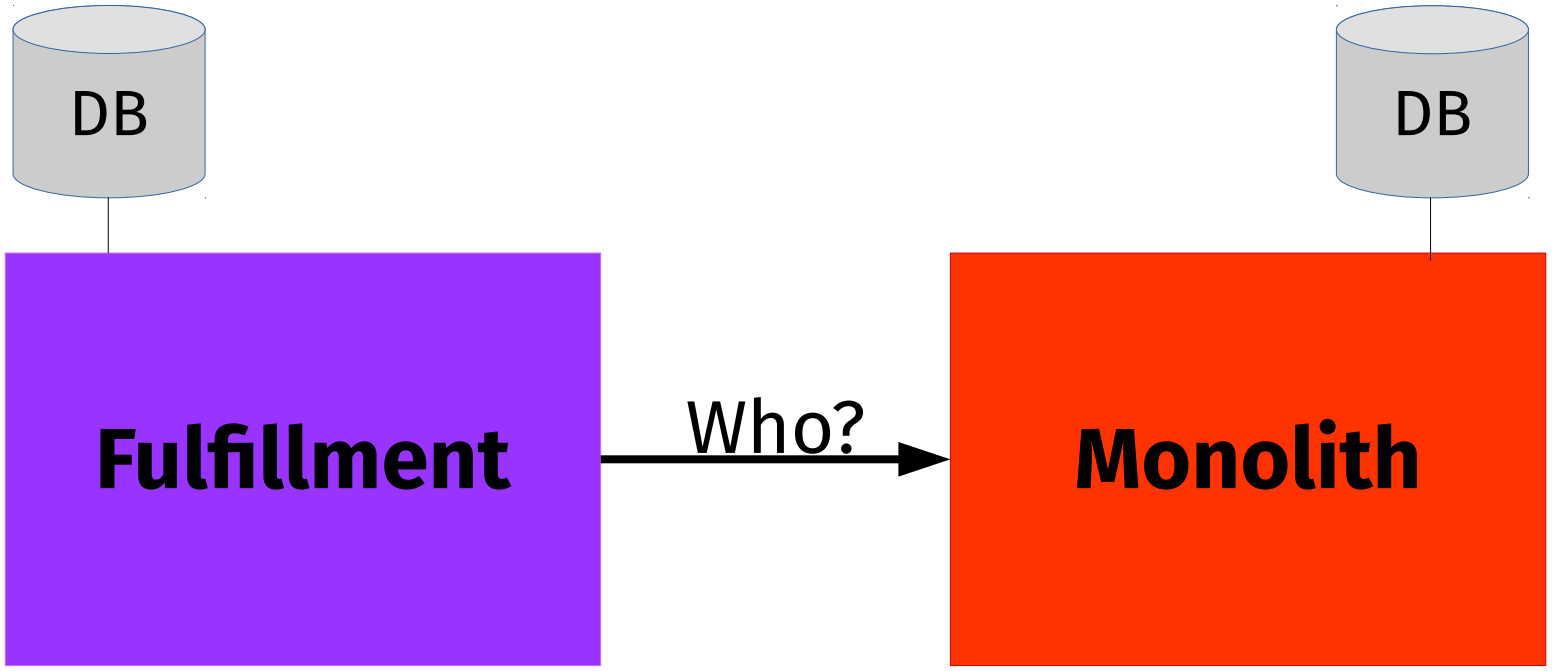
Own UI



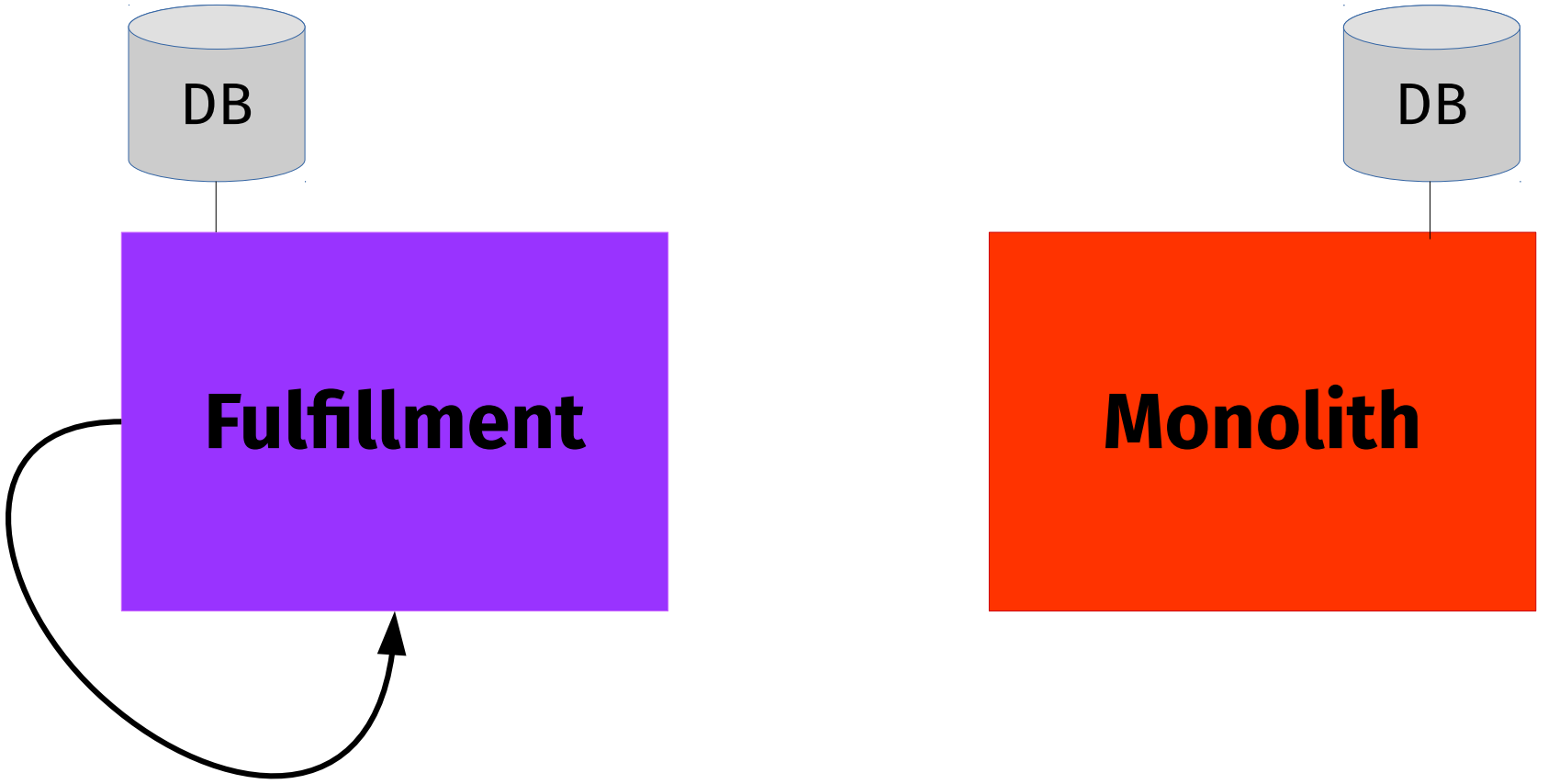
OAuth



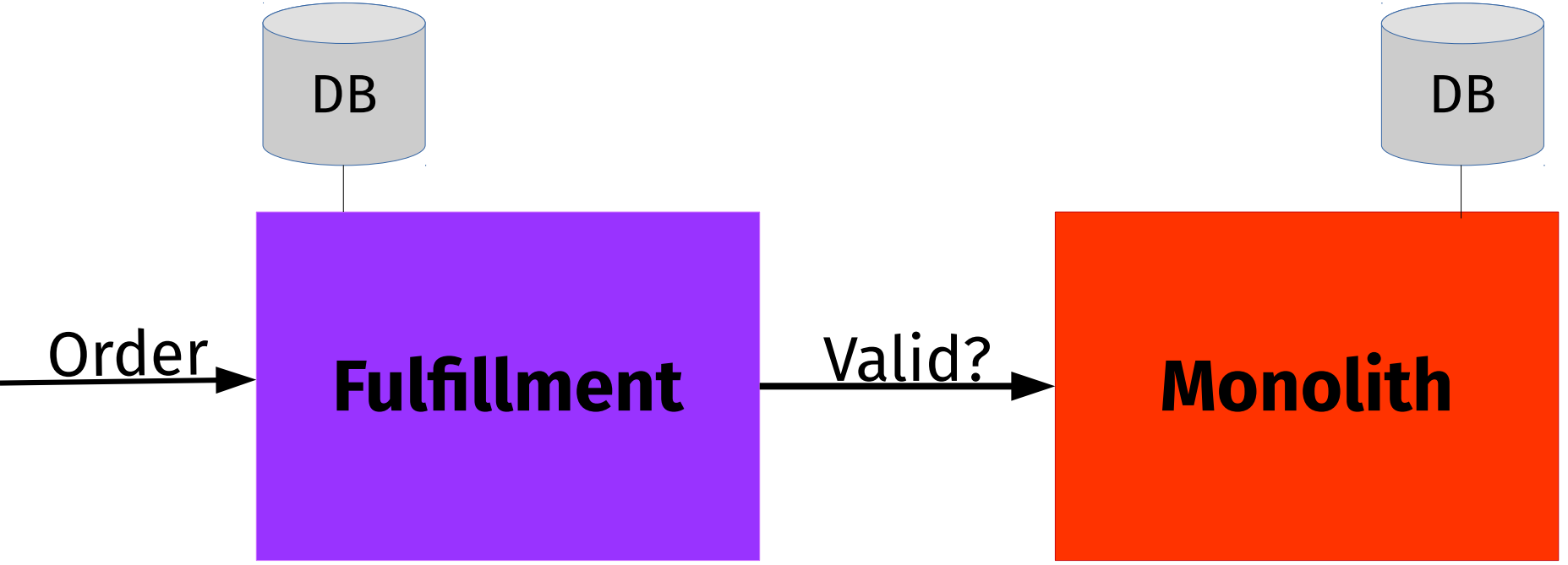
/current_user



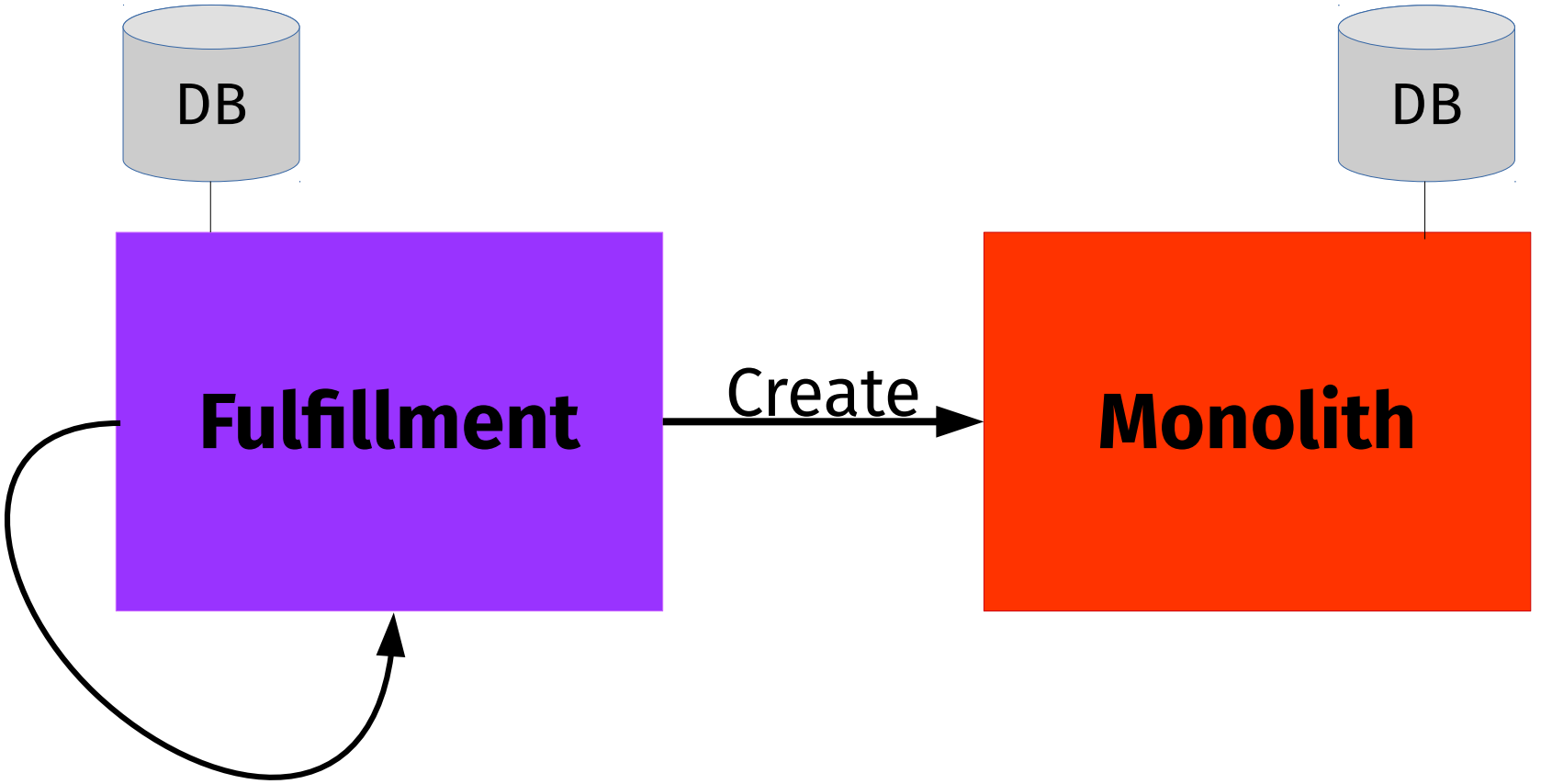
Products



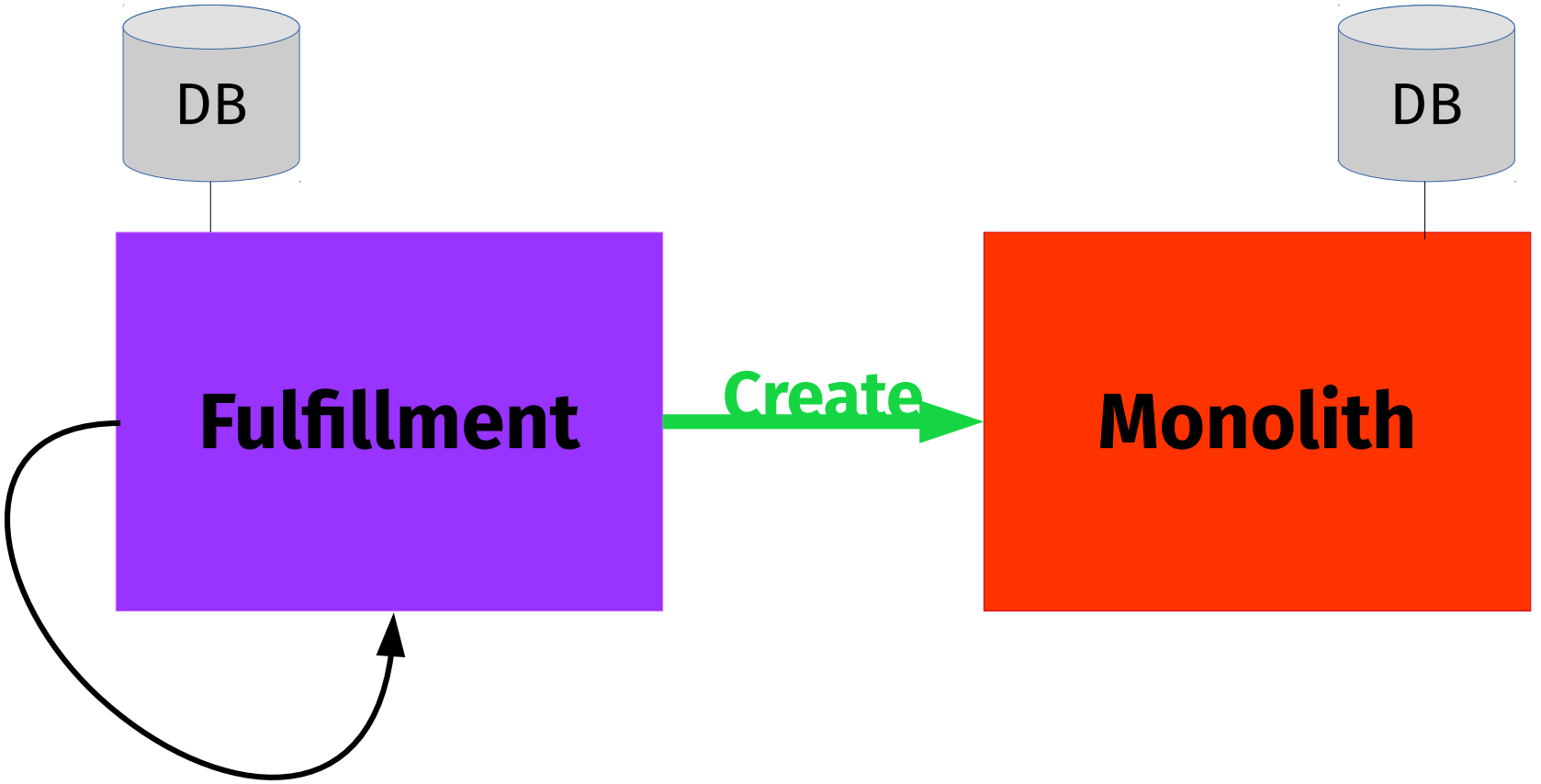
Order



Packaging



HTTP REST




Railway Oriented Programming

```elixir
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

“Success” Rail

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

“Error” Rail

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

Validate internally

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

Validate externally

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

Save

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

Done

```
defmodule Fulfillment.OrderCreation do
  def create(params, customer, auth) do
    with {:ok, order} <- validate_order(params, customer),
         {:ok, order} <- validate_in_backend(order, auth),
         {:ok, order} <- Repo.insert(order) do
      {:ok, order}
    else
      {:error, changeset} -> {:error, changeset}
    end
  end
end
```

No Kafka/
Message Queue/
GraphQL?

Be careful with new technologies!

Microservices!

~~Microservices!~~

”Macroapplications”

All challenges are
clearly technical

Stakeholders

Your team

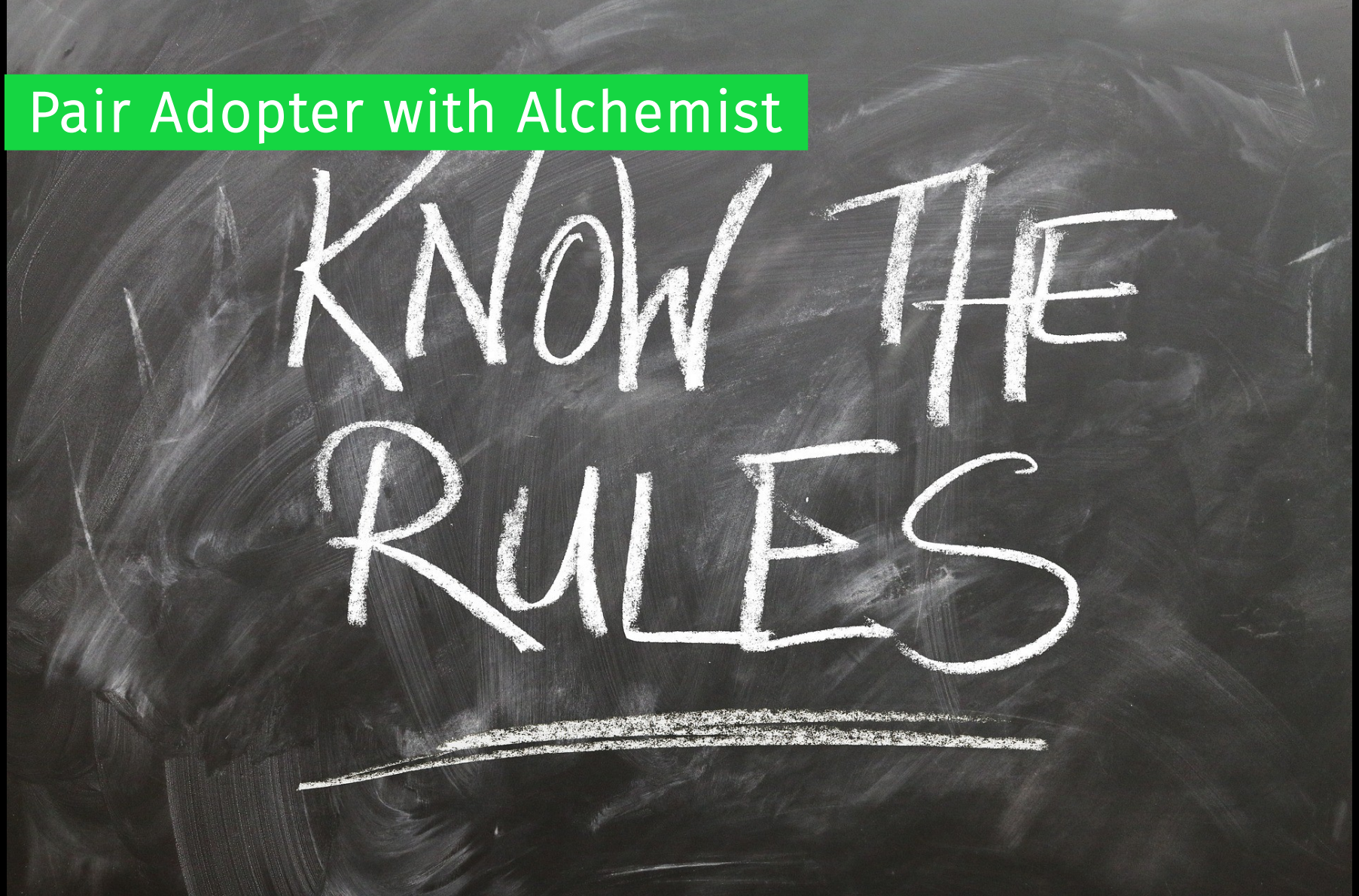

First Major Version

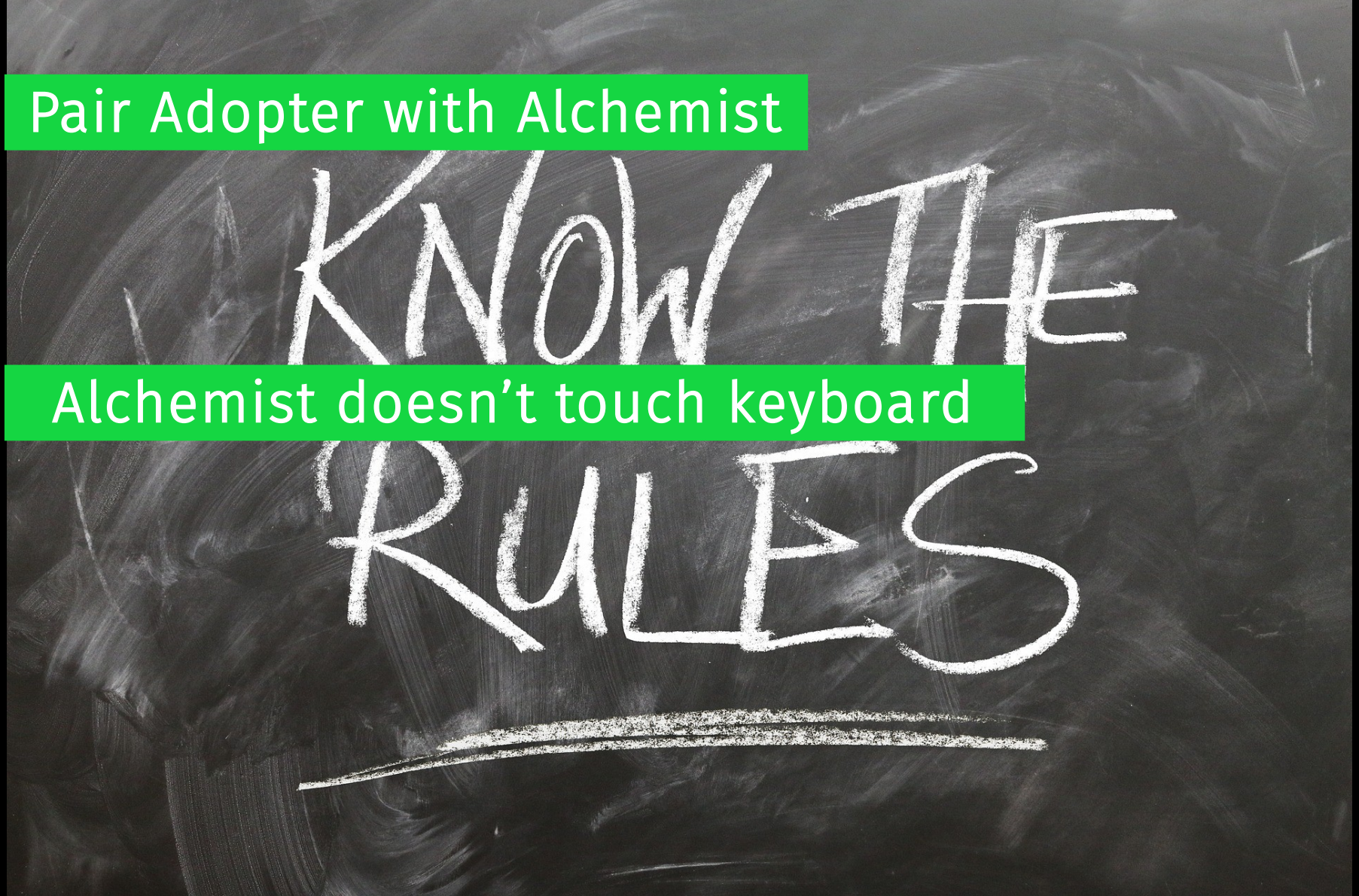
Get the others on board


Workshops

To know it,
you got to build it


KNOW THE
RULES


Pair Adopter with Alchemist
KNOW THE
RULES


Pair Adopter with Alchemist
KNOW THE
Alchemist doesn't touch keyboard
RULES

The Pragmatic Programmers

Your Elixir Source

Knowledge

Programming Phoenix

Productive |> Reliable |> Fast

Chris McCord,
Bruce Tate,
and José Valim

edited by Jacquelyn Carter

The Pragmatic Programmers

Your Elixir Source

Adopting Elixir

From Concept to Production

Ben Marx, José Valim, Bruce Tate

edited by Jacquelyn Carter

Knowledge

“We are still using this like Rails – we should use GenServers!”

Someone on your team

You don’t need to kill your Monolith, Complement it


Your Apps

Enjoy Using Elixir effectively to Help Your Company

Tobias Pfeiffer
@PragTob
pragtob.info

LIEFERY