VAXINE

Building a rich-CRDT database on AntidoteDB.

RainbowFS Workshop, Monday 28 March 2022
Sorbonne-Université–LIP6, Paris, France

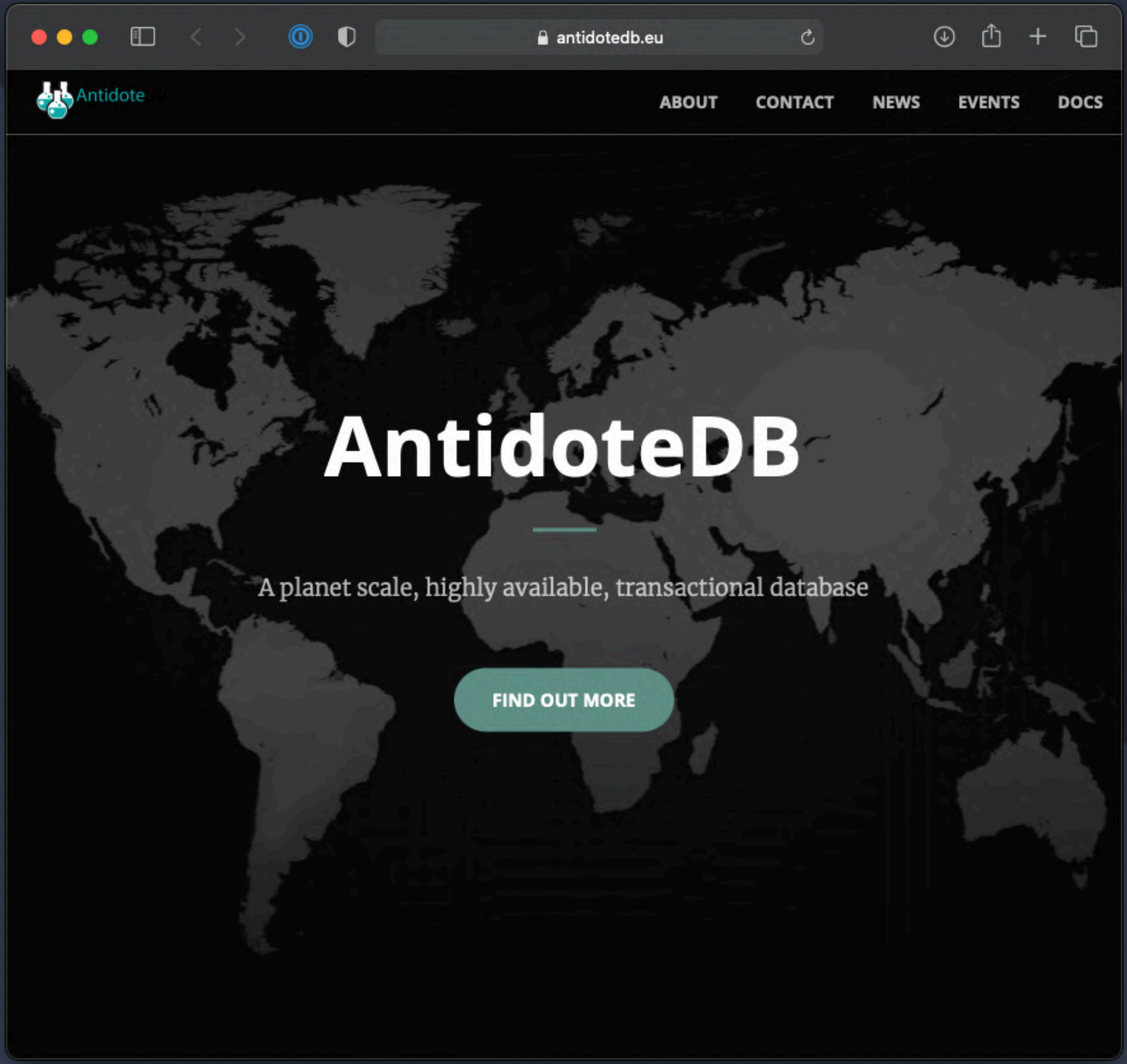
James Arthur, CEO

https://vaxine.io

Antidote

TCC+:

- Highly Available Transactions
- Sticky Availability
- Causal Consistency
- CRDTs



AntidoteDB

A planet scale, highly available, transactional database

“Vaxine”

- TCC+ is a Cure for consistency under partition
- Antidote implements the Cure protocol
- Vaxine is a delivery mechanism for the Antidote

Delivery mechanism

Goals

✓ cloud database

✗ not edge / p2p / byzantine tolerant

✓ online system

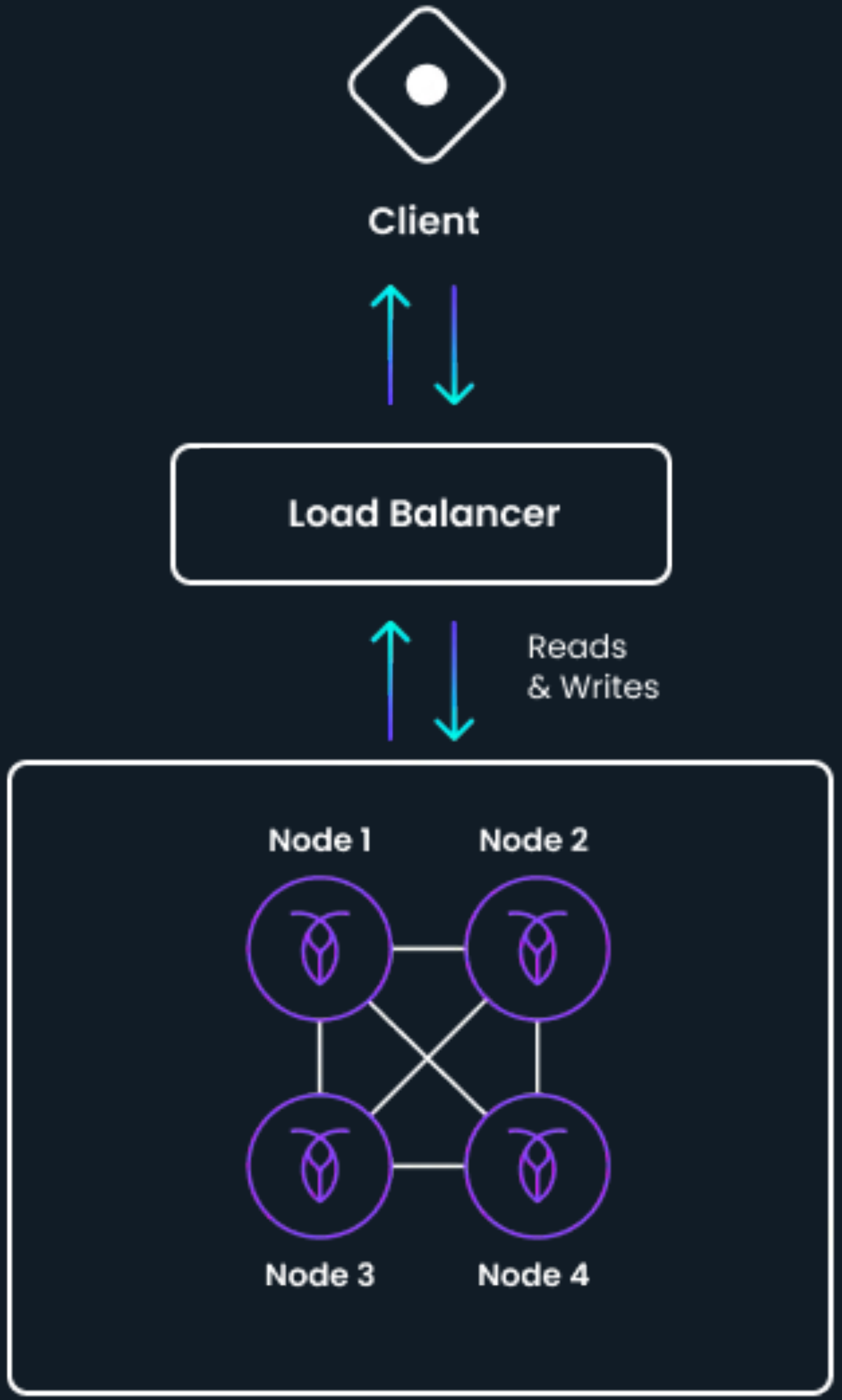
✗ not offline / local first

✓ 5 – 15 data centres

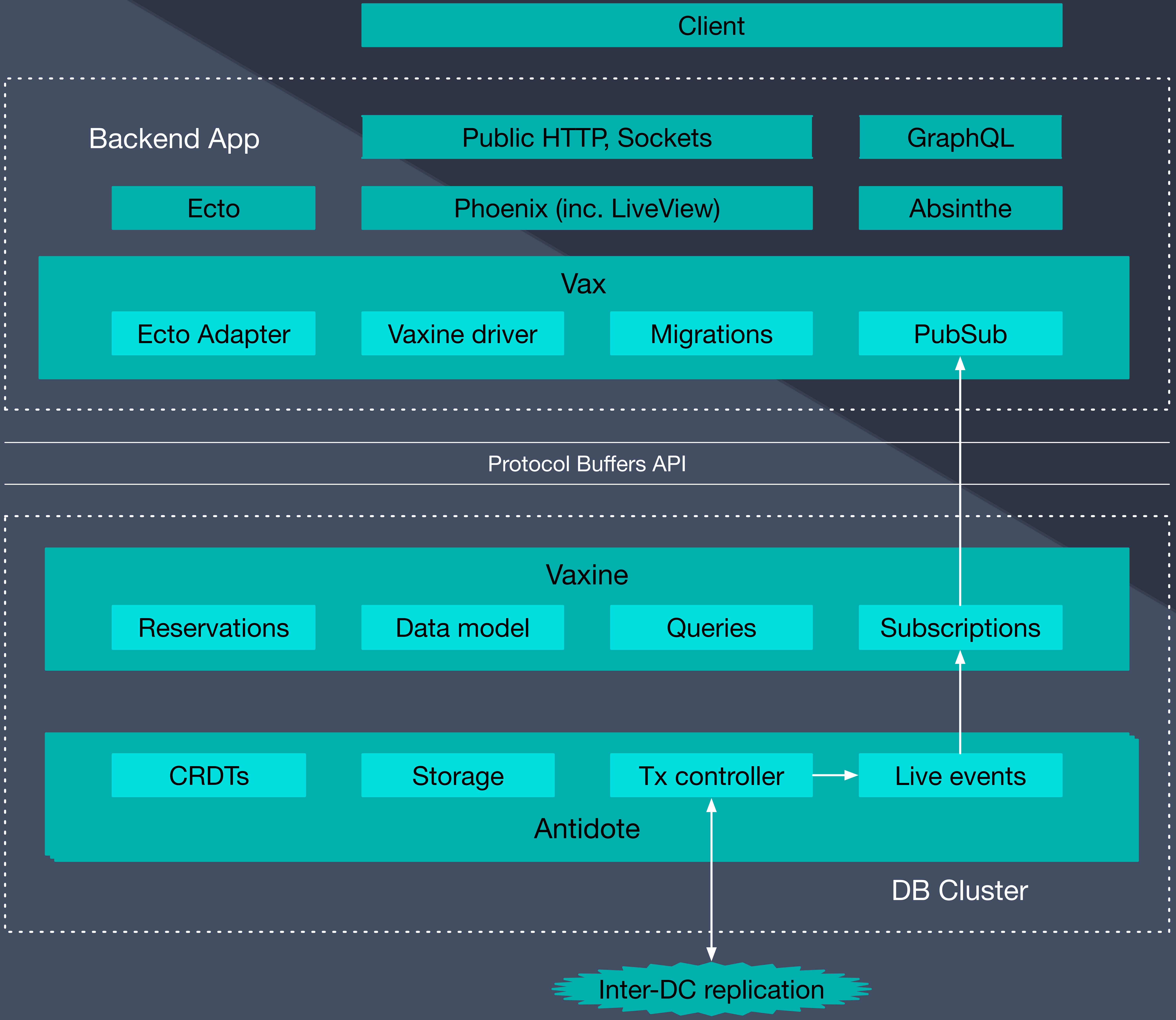
✗ not hundreds or thousands

✓ optimised for latency + integrity

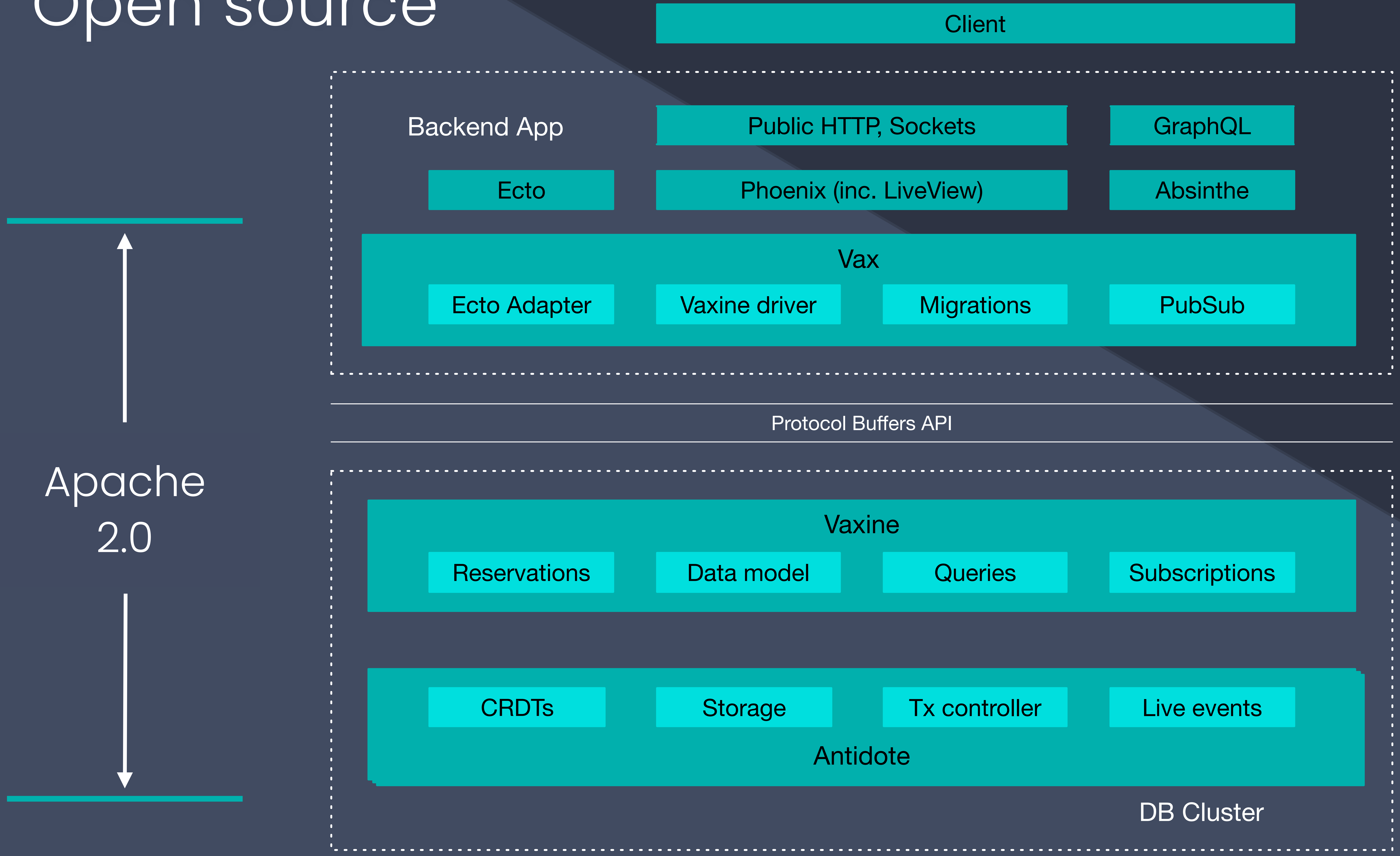
✗ not throughput or storage efficiency



Layers


Client
Backend App
Public HTTP, Sockets
GraphQL
Ecto
Phoenix (inc. LiveView)
Absinthe
Vax
Ecto Adapter
Vaxine driver
Migrations
PubSub
Protocol Buffers API
Vaxine
Reservations
Data model
Queries
Subscriptions
CRDTs
Storage
Tx controller
Live events
Antidote
DB Cluster
Inter-DC replication

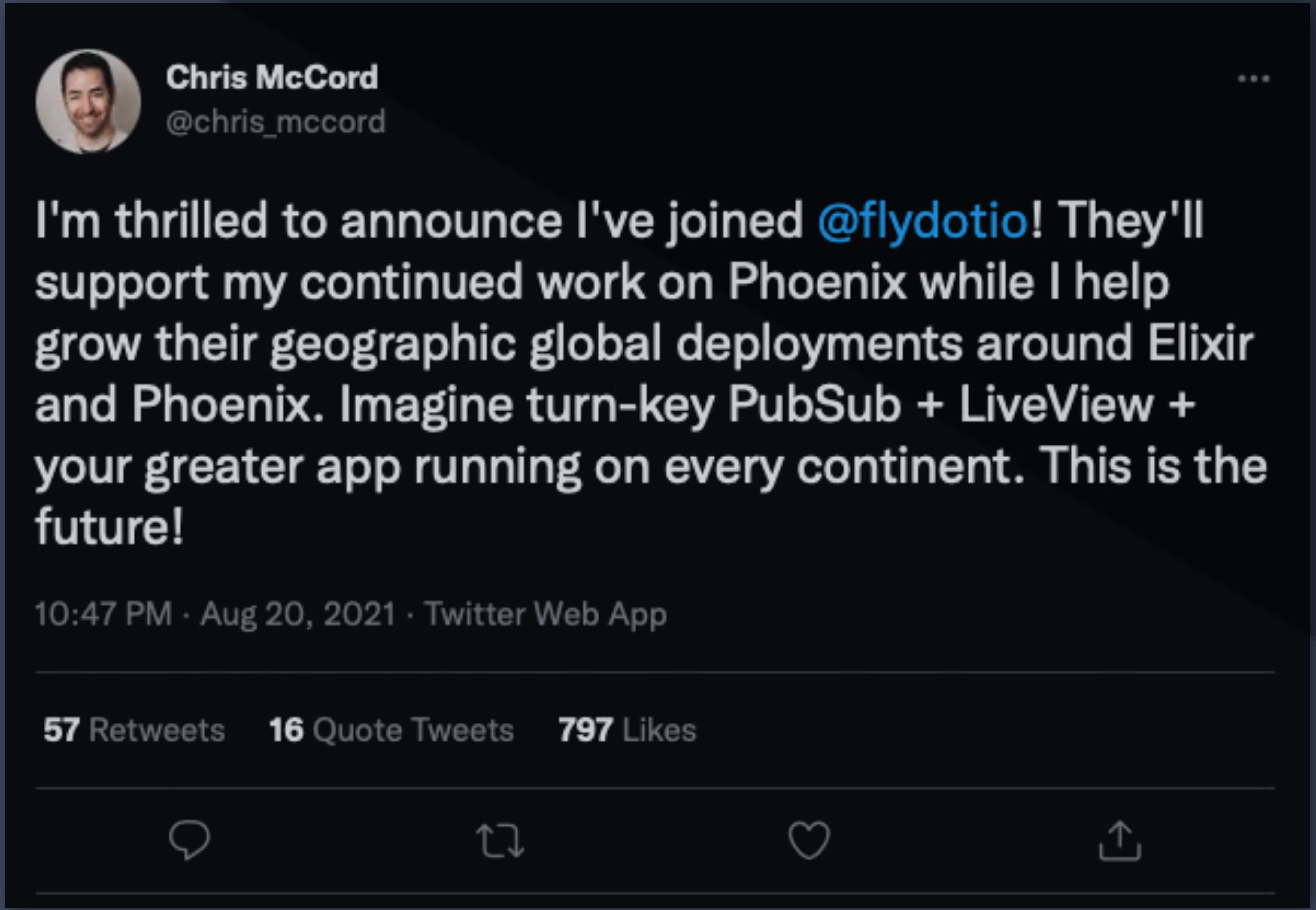
Open source


Client
Backend App
Public HTTP, Sockets
GraphQL
Ecto
Phoenix (inc. LiveView)
Absinthe
Vax
Ecto Adapter
Vaxine driver
Migrations
PubSub
Protocol Buffers API
Vaxine
Reservations
Data model
Queries
Subscriptions
CRDTs
Storage
Tx controller
Live events
Antidote
DB Cluster
Apache
2.0

Elixir

Natural fit:

- we're building on a BEAM-based system
- overlap between Erlang/ Elixir and distributed systems communities
- Phoenix LiveView is driving demand for geo-distributed deployment



Chris McCord
@chris_mccord

I'm thrilled to announce I've joined @flydotio! They'll support my continued work on Phoenix while I help grow their geographic global deployments around Elixir and Phoenix. Imagine turn-key PubSub + LiveView + your greater app running on every continent. This is the future!

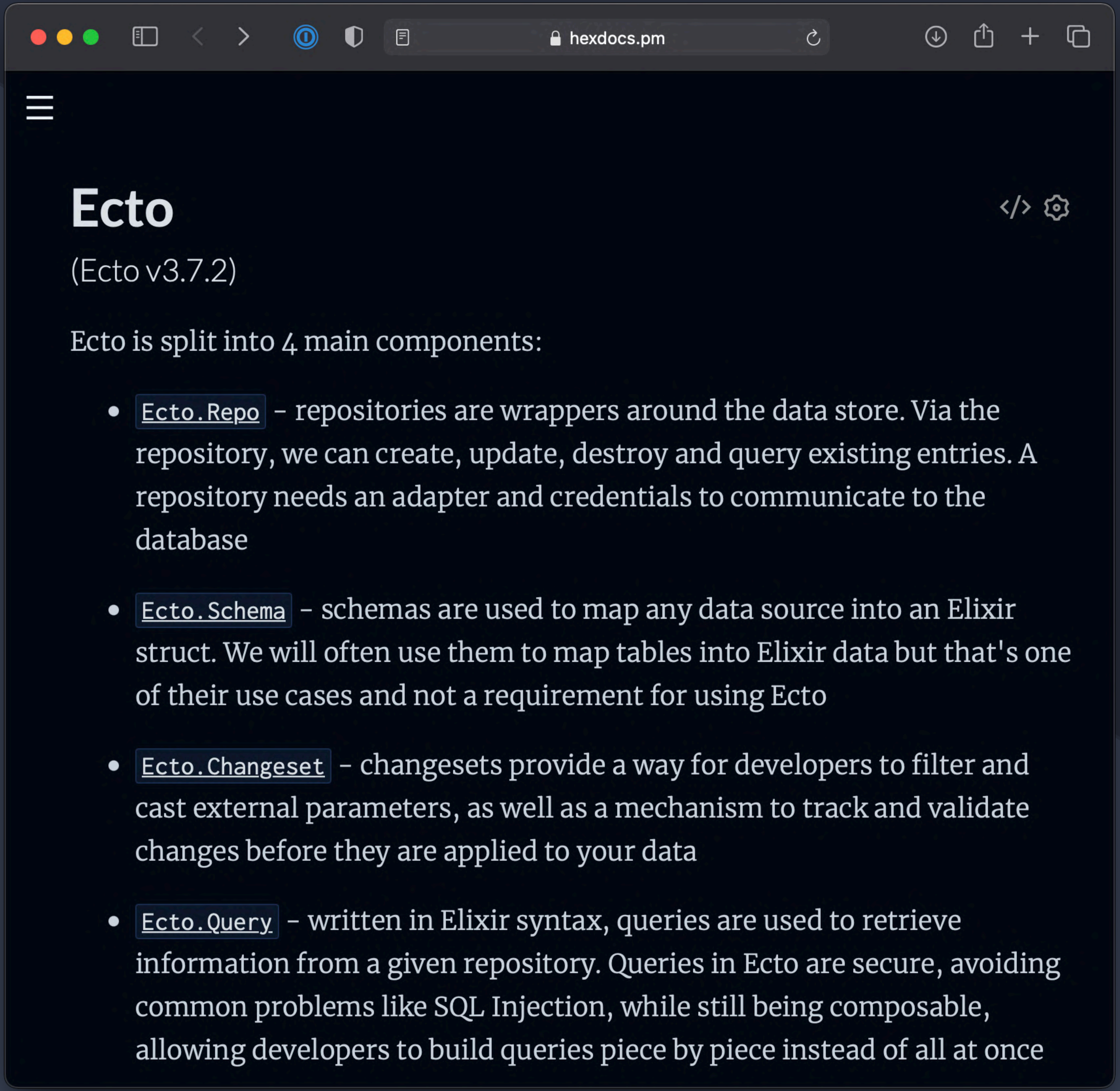
10:47 PM · Aug 20, 2021 · Twitter Web App

57 Retweets 16 Quote Tweets 797 Likes

Ecto

Key integration target:

- relational-oriented data access library
- easy to use and familiar for generalist web developers



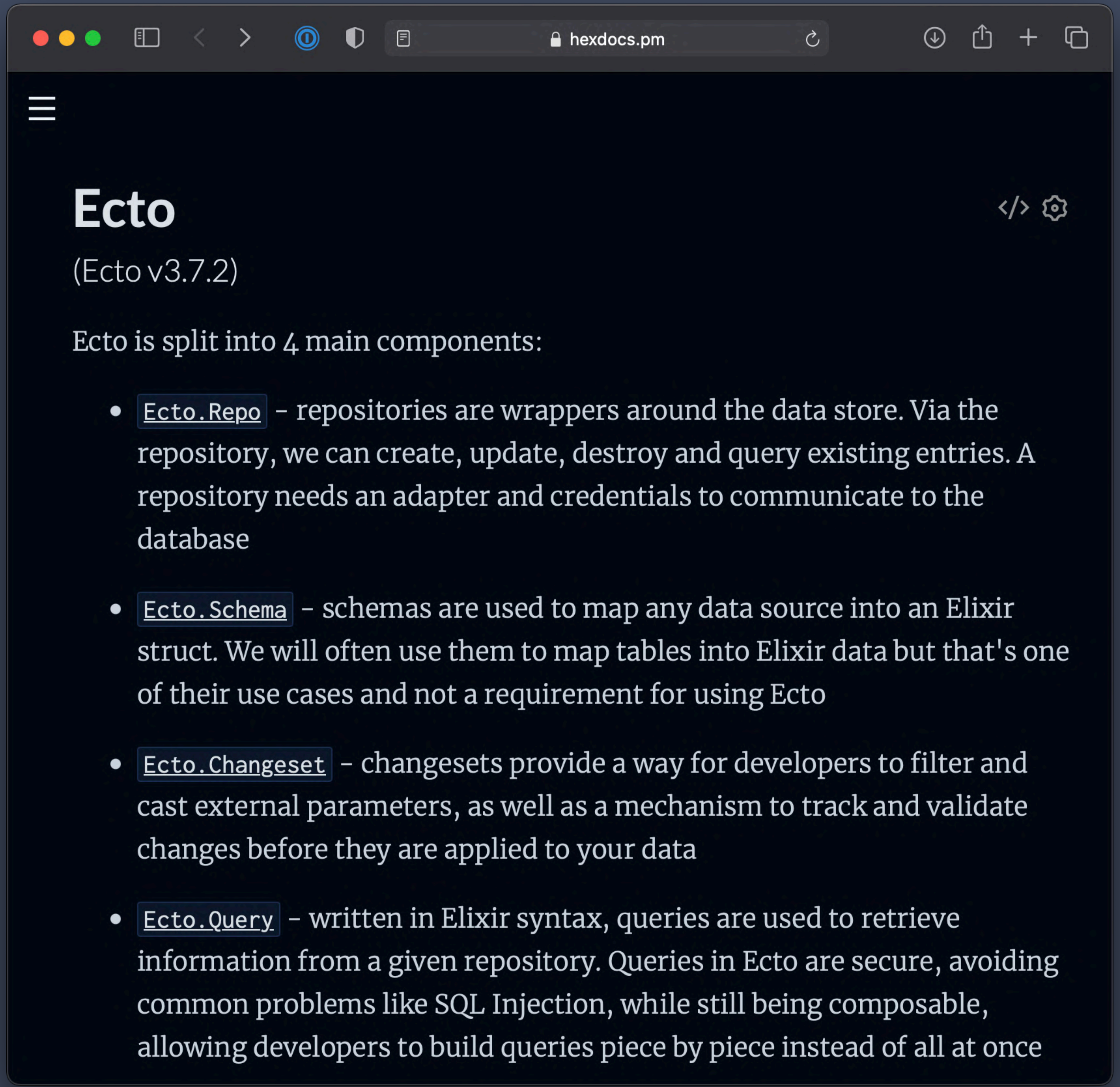
Ecto

(Ecto v3.7.2)

Ecto is split into 4 main components:

- `Ecto.Repo` - repositories are wrappers around the data store. Via the repository, we can create, update, destroy and query existing entries. A repository needs an adapter and credentials to communicate to the database
- `Ecto.Schema` - schemas are used to map any data source into an Elixir struct. We will often use them to map tables into Elixir data but that's one of their use cases and not a requirement for using Ecto
- `Ecto.Changeset` - changesets provide a way for developers to filter and cast external parameters, as well as a mechanism to track and validate changes before they are applied to your data
- `Ecto.Query` - written in Elixir syntax, queries are used to retrieve information from a given repository. Queries in Ecto are secure, avoiding common problems like SQL Injection, while still being composable, allowing developers to build queries piece by piece instead of all at once

The world

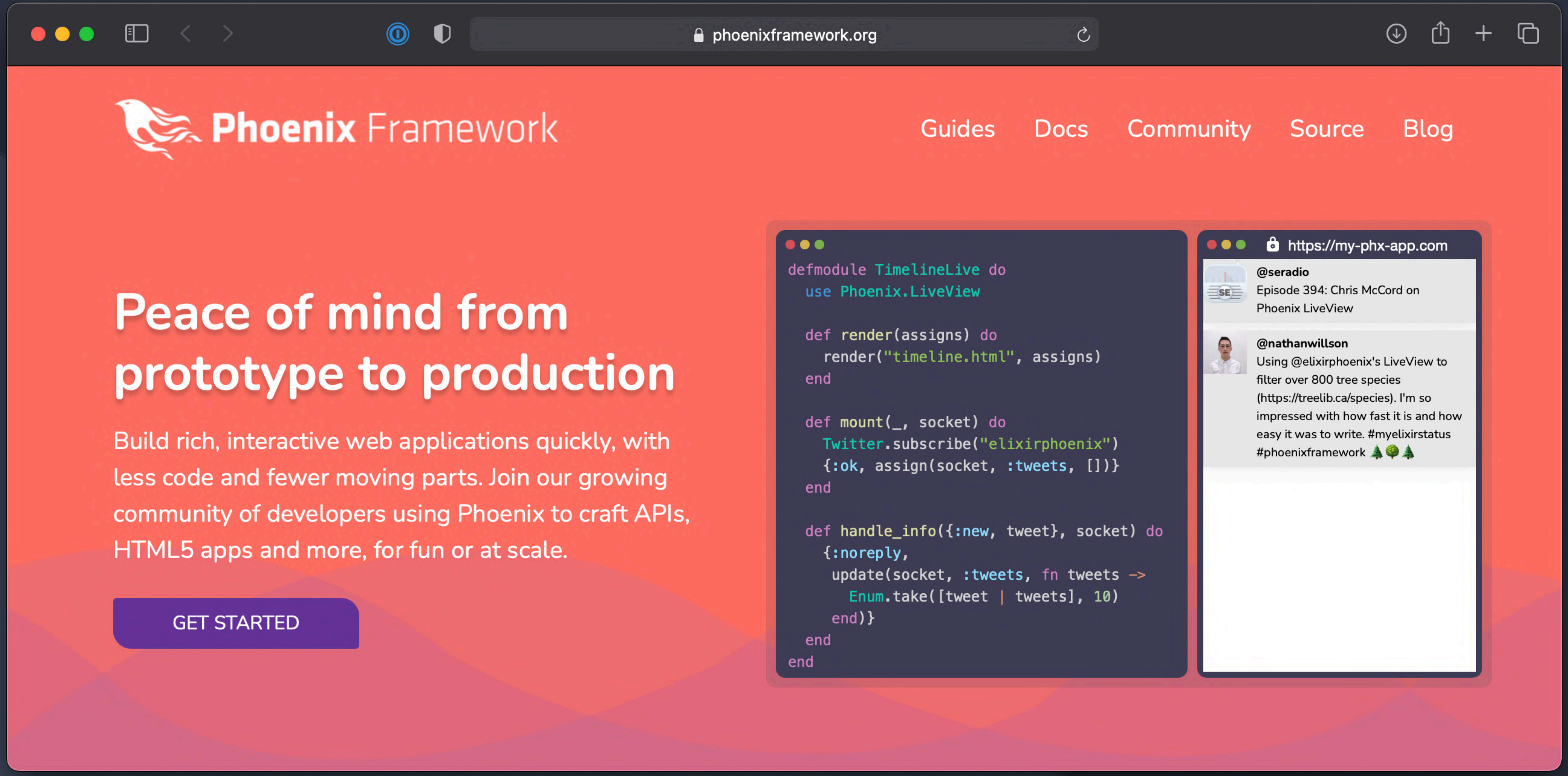


Ecto

(Ecto v3.7.2)

Ecto is split into 4 main components:

- `Ecto.Repo` - repositories are wrappers around the data store. Via the repository, we can create, update, destroy and query existing entries. A repository needs an adapter and credentials to communicate to the database
- `Ecto.Schema` - schemas are used to map any data source into an Elixir struct. We will often use them to map tables into Elixir data but that's one of their use cases and not a requirement for using Ecto
- `Ecto.Changeset` - changesets provide a way for developers to filter and cast external parameters, as well as a mechanism to track and validate changes before they are applied to your data
- `Ecto.Query` - written in Elixir syntax, queries are used to retrieve information from a given repository. Queries in Ecto are secure, avoiding common problems like SQL Injection, while still being composable, allowing developers to build queries piece by piece instead of all at once



Peace of mind from prototype to production

Build rich, interactive web applications quickly, with less code and fewer moving parts. Join our growing community of developers using Phoenix to craft APIs, HTML5 apps and more, for fun or at scale.

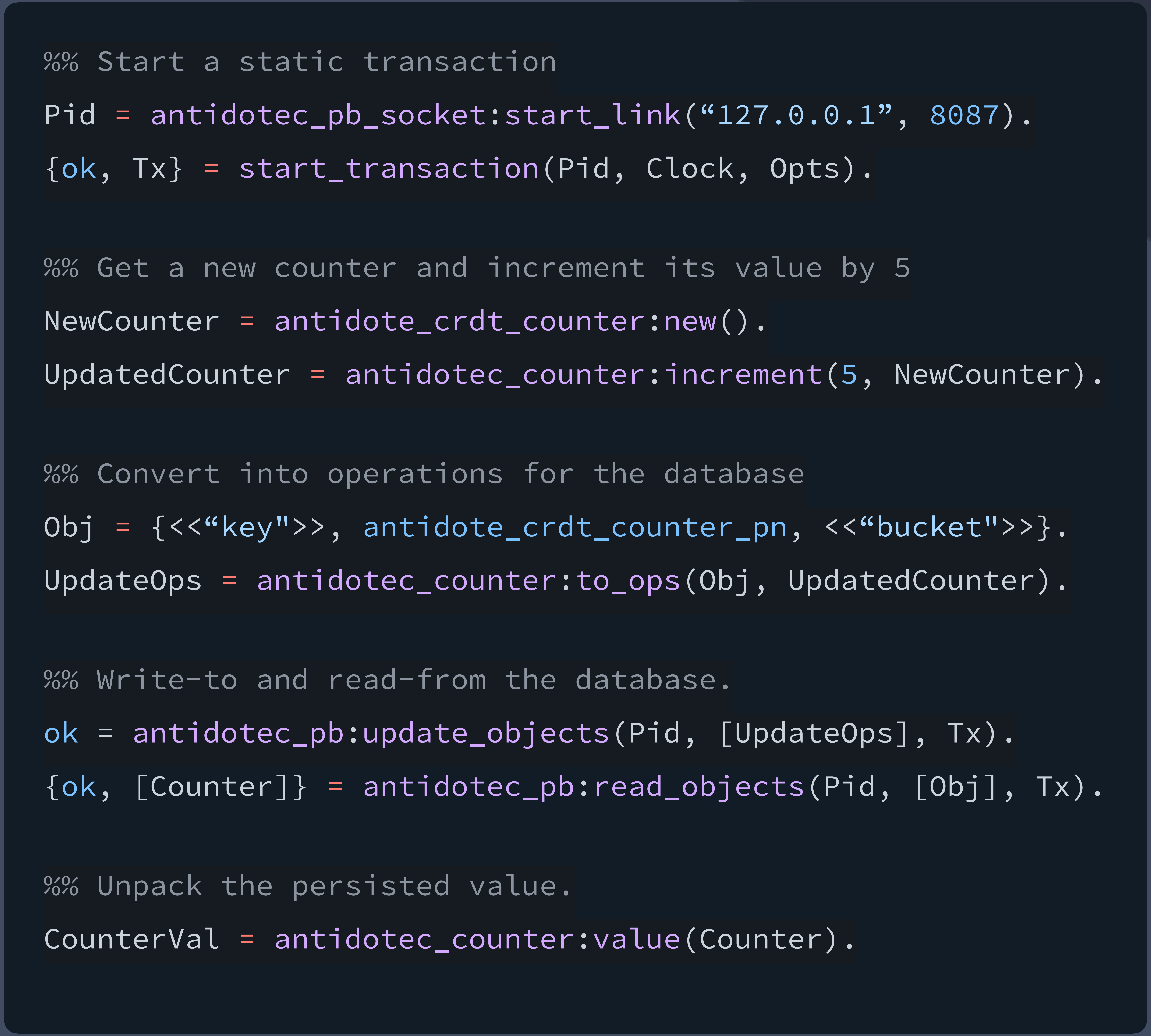
GET STARTED

```elixir
defmodule TimelineLive do
  use Phoenix.LiveView

  def render(assigns) do
    render("timeline.html", assigns)
  end

  def mount(_, socket) do
    Twitter.subscribe("elixirphoenix")
    {:ok, assign(socket, :tweets, [])}
  end

  def handle_info({:new, tweet}, socket) do
    {:noreply,
     update(socket, :tweets, fn tweets ->
       Enum.take([tweet | tweets], 10)
     end)}
  end
end
```

@seradio
Episode 394: Chris McCord on Phoenix LiveView

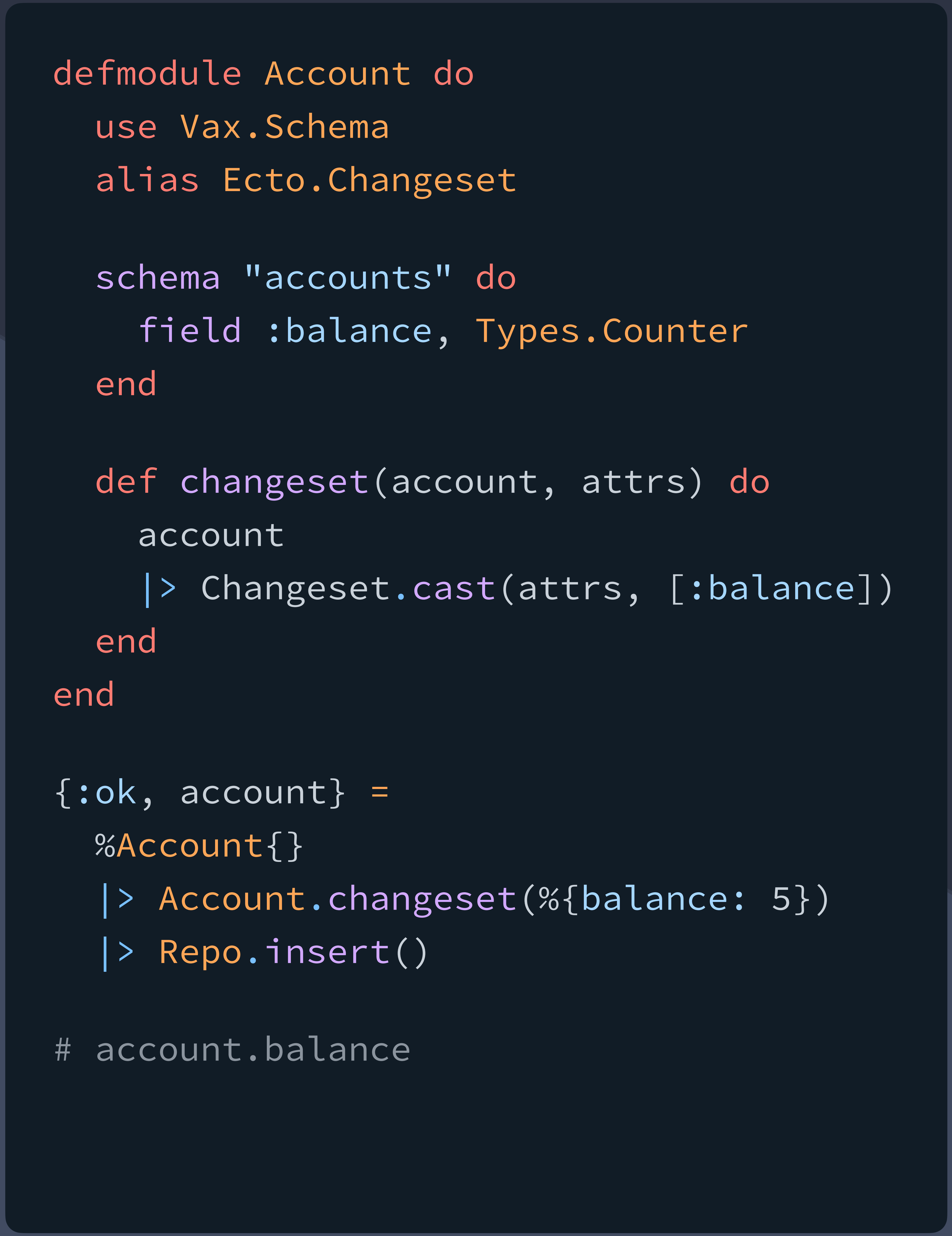
@nathanwillson
Using @elixirphoenix's LiveView to filter over 800 tree species (https://treelib.ca/species). I'm so impressed with how fast it is and how easy it was to write. #myelixirstatus #phoenixframework 🌲🌳🌲

The GraphQL toolkit for Elixir



```
%% Start a static transaction
Pid = antidotec_pb_socket:start_link(“127.0.0.1”, 8087).
{ok, Tx} = start_transaction(Pid, Clock, Opts).

%% Get a new counter and increment its value by 5
NewCounter = antidote_crdt_counter:new().
UpdatedCounter = antidotec_counter:increment(5, NewCounter).

%% Convert into operations for the database
Obj = {<<“key">>, antidote_crdt_counter_pn, <<“bucket">>}.
UpdateOps = antidotec_counter:to_ops(Obj, UpdatedCounter).

%% Write-to and read-from the database.
ok = antidotec_pb:update_objects(Pid, [UpdateOps], Tx).
{ok, [Counter]} = antidotec_pb:read_objects(Pid, [Obj], Tx).

%% Unpack the persisted value.
CounterVal = antidotec_counter:value(Counter).
```

→



```
defmodule Account do
  use Vax.Schema
  alias Ecto.Changeset

  schema "accounts" do
    field :balance, Types.Counter
  end

  def changeset(account, attrs) do
    account
    |> Changeset.cast(attrs, [:balance])
  end
end

{:ok, account} =
  %Account{}
  |> Account.changeset(%{balance: 5})
  |> Repo.insert()

# account.balance
```

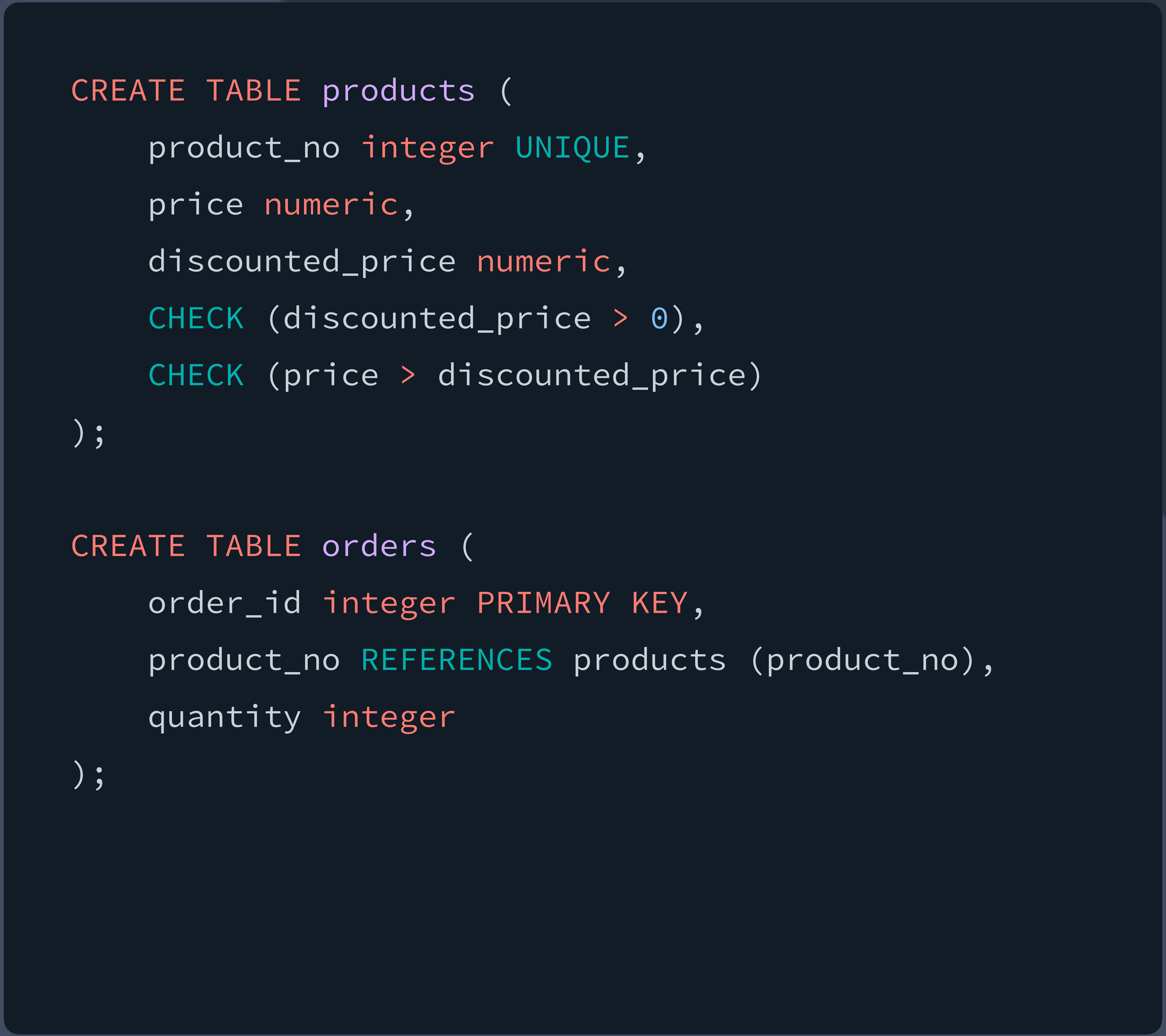
“Surgical precision”

Key design decision:

A. vertical integration with a single language client / data access library; vs

B. language agnostic network protocol

```
%Account{}
|> Account.changeset(%{})
|> IO.inspect()

=> #Ecto.Changeset<
  action: nil,
  changes: %{balance: 0},
  errors: [],
  data: #Account<>,
  valid?: true
>

|> Repo.insert()
```



```
CREATE TABLE products (
    product_no integer UNIQUE,
    price numeric,
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no REFERENCES products (product_no),
    quantity integer
);
```

Complexity

Compensating with failure code:

- write application code to handle failure/edge cases
- you want to put that complexity back in the database

```
results = db.query(...)

# work around null bugs in your app code!
valid_results = [
    x for x in results if x.parent
]
```

Rich-CRDTs

We’re using rich-CRDTs to build in “standard” database guarantees.

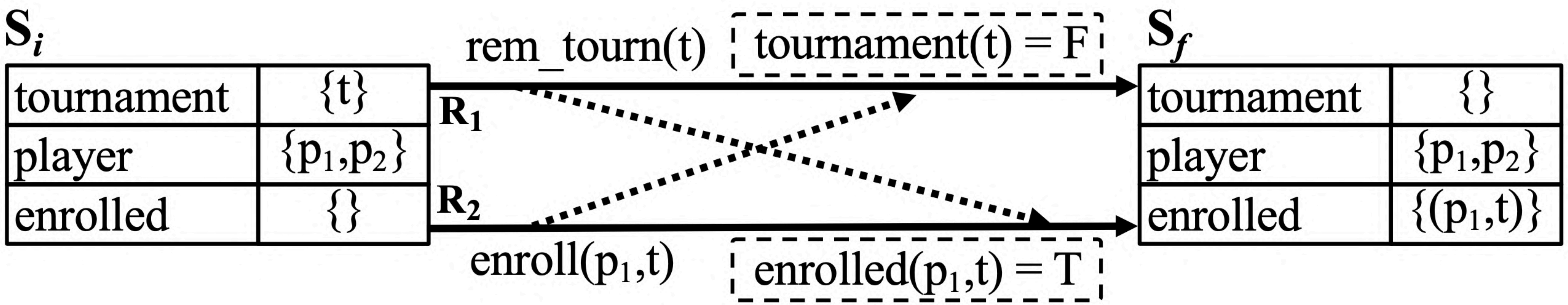
Three techniques:

- ✓ conflict-free concurrency semantics
- ✓ runtime coordination using reservations
- ✗ static analysis

Standard database guarantees

- Referential integrity
- Unique constraint
- Check constraints
- Prefixed uuid (autogenerated uuid)
- Auto-incremented sequential ID (unique sequence)
- Auto-incremented identifier (ordered unique value)

Conflict-free concurrency semantics



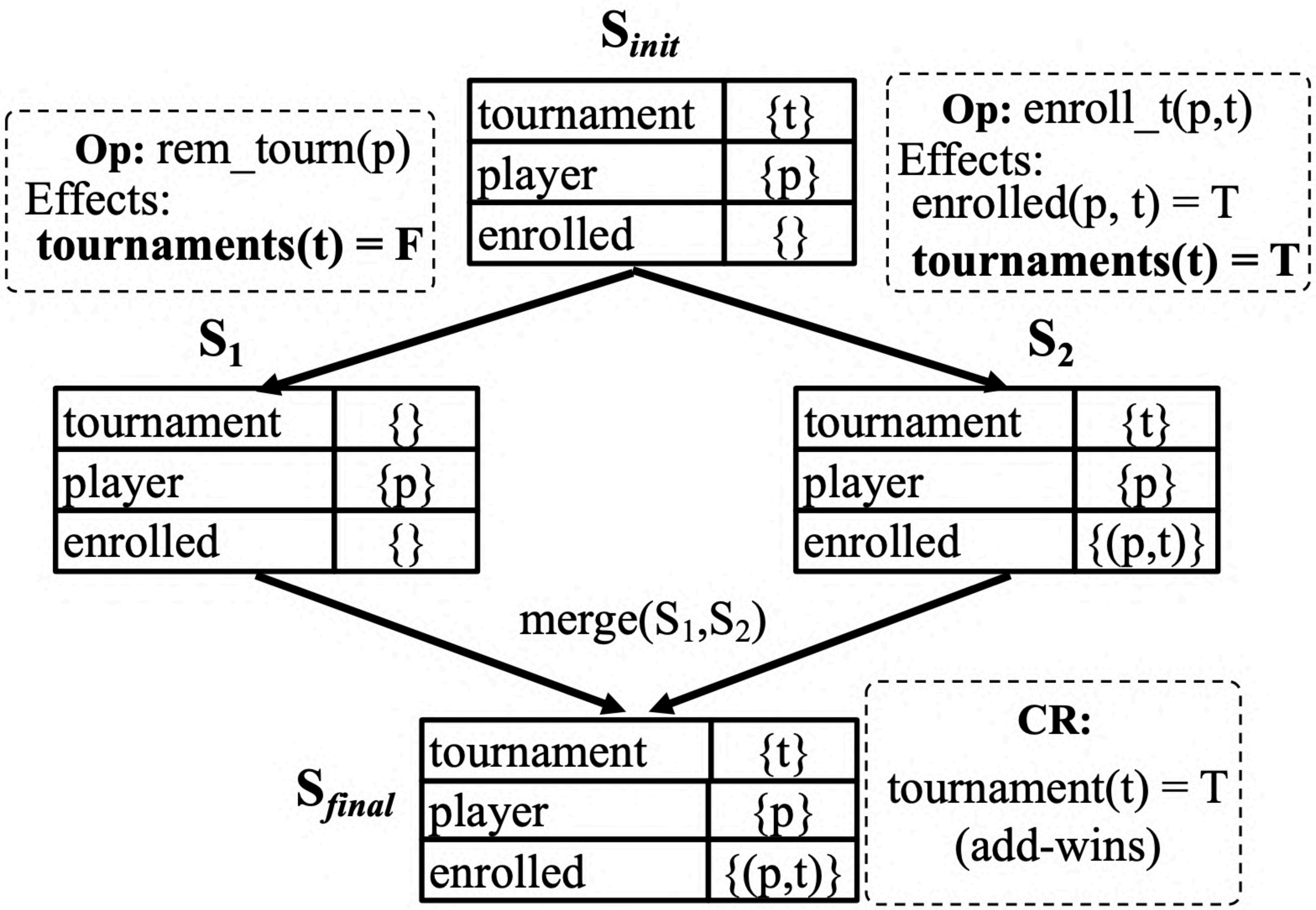
S_i

tournament	{t}
player	{p_1,p_2}
enrolled	{}

S_f

tournament	{}
player	{p_1,p_2}
enrolled	{(p_1,t)}

Figure 1: Concurrent execution of $enroll(p, t)$ and $rem\_tourn(t)$ leads to an invariant violation.



S_{init}

tournament	{t}
player	{p}
enrolled	{}

Op: rem_tourn(p)
Effects:
tournaments(t) = F

Op: enroll_t(p,t)
Effects:
enrolled(p, t) = T
tournaments(t) = T

S_1

tournament	{}
player	{p}
enrolled	{}

S_2

tournament	{t}
player	{p}
enrolled	{(p,t)}

merge(S_1,S_2)

S_{final}

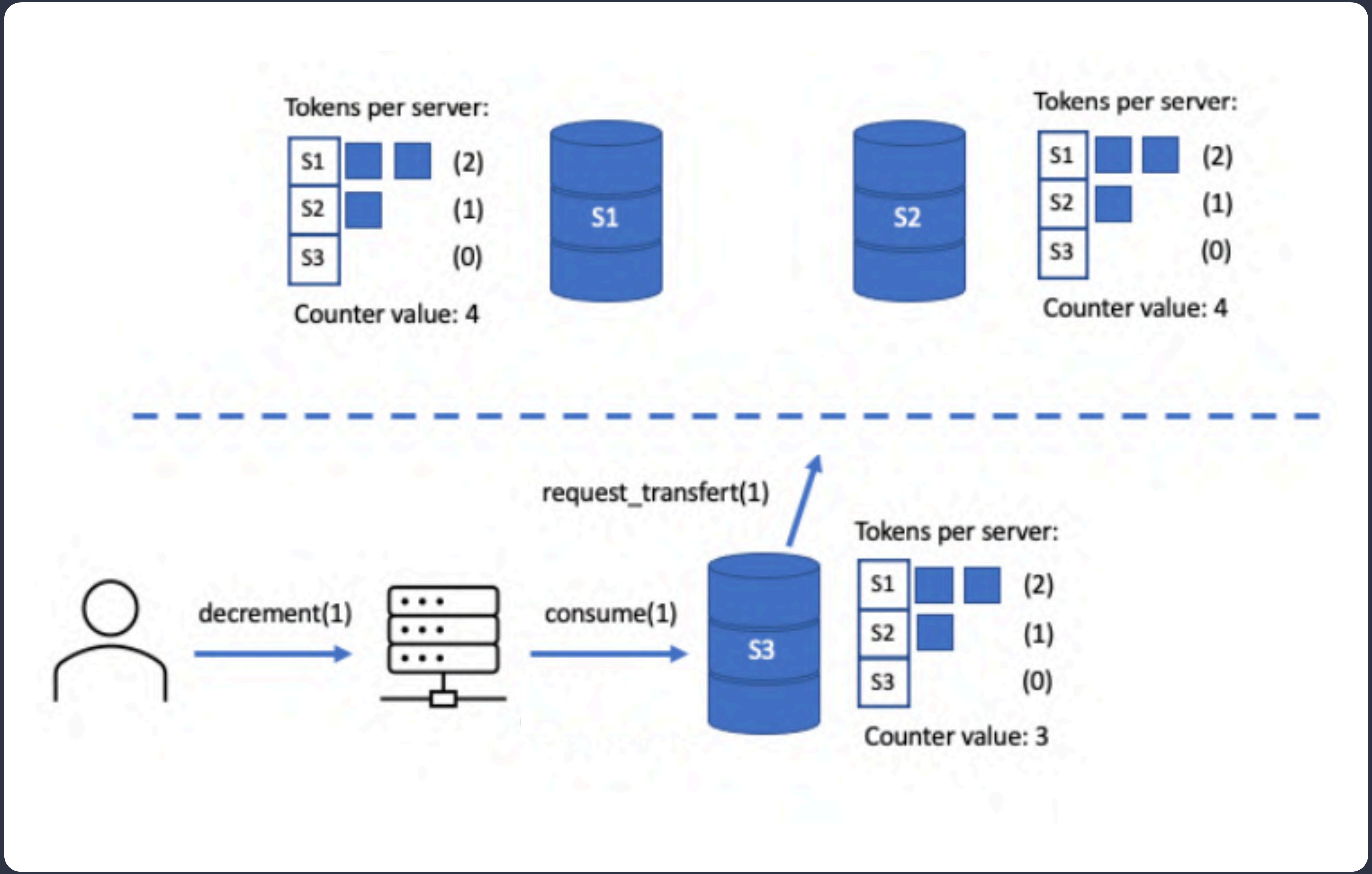
tournament	{t}
player	{p}
enrolled	{(p,t)}

CR:
tournament(t) = T
(add-wins)

(b) Recreates tournament *t*.

Runtime coordination using reservations

- like dynamic locks
- distribute rights to perform operations across regions
- proactively rebalance to minimise coordination



Static analysis

Formal verification (ahead of time) of explicit consistency specifications.

- ✗ harder for general developers to reason about
- ✗ brittle when exposed to real world deployment and usage patterns

```
// “normal” application code
def getUser(id: UserId): getUserResult {
  atomic {
    if (user_exists(id)) {
      return found(user_name_get(id), user_mail_get(id))
    } else {
      return notFound()
    }
  }
}

// explicit consistency specifications defining invariants
// that must be preserved.
invariant (forall r: invocationId, g: invocationId, u: UserId ::
     r.info == removeUser(u)
  && g.info == getUser(u)
  && r happened before g
  ==> g.result == getUser_res(notFound()))
```

James Arthur

CEO

Software developer and entrepreneur. Co-Founder of Hazy, LGN, Opendesk and Post Urban.

Valter Balegas

Principal Engineer

Distributed systems researcher and engineer. Rich-CRDTs. Just-right consistency. MySQL at Oracle.

Annette Bieniusa

Chief Architect

Lead developer of AntidoteDB at TUK. Concurrent and distributed software. Geo-replication.

Purva Gujar

Growth & Community

Founder and CEO at Inceptive. Investment at Rainbow Capital and Swig. South Park Commons. MIT.

Dave Cottlehuber

Founding Engineer & Chief People Officer

FreeBSD. CouchDB. Distributed systems in Erlang & Elixir. Values-driven person & leader.

Vasilii Demidenok

Founding Engineer

Senior Engineer & Tech Lead at Cisco, Klarna, Exante. Distributed systems & formal methods.

Ilia Borovitinov

Founding Engineer

Senior full-stack developer. Elixir, Javascript, databases, orchestration, web app development.

Felipe Stival

Founding Engineer

Software engineer, focused on functional programming and distributed systems. Elixir. Core Ecto team.

Marc Shapiro

Scientific Advisor

Co-inventor of CRDTs. Inventor of the proxy. Chief Scientist at Concordant. Inria & Sorbonne Université.

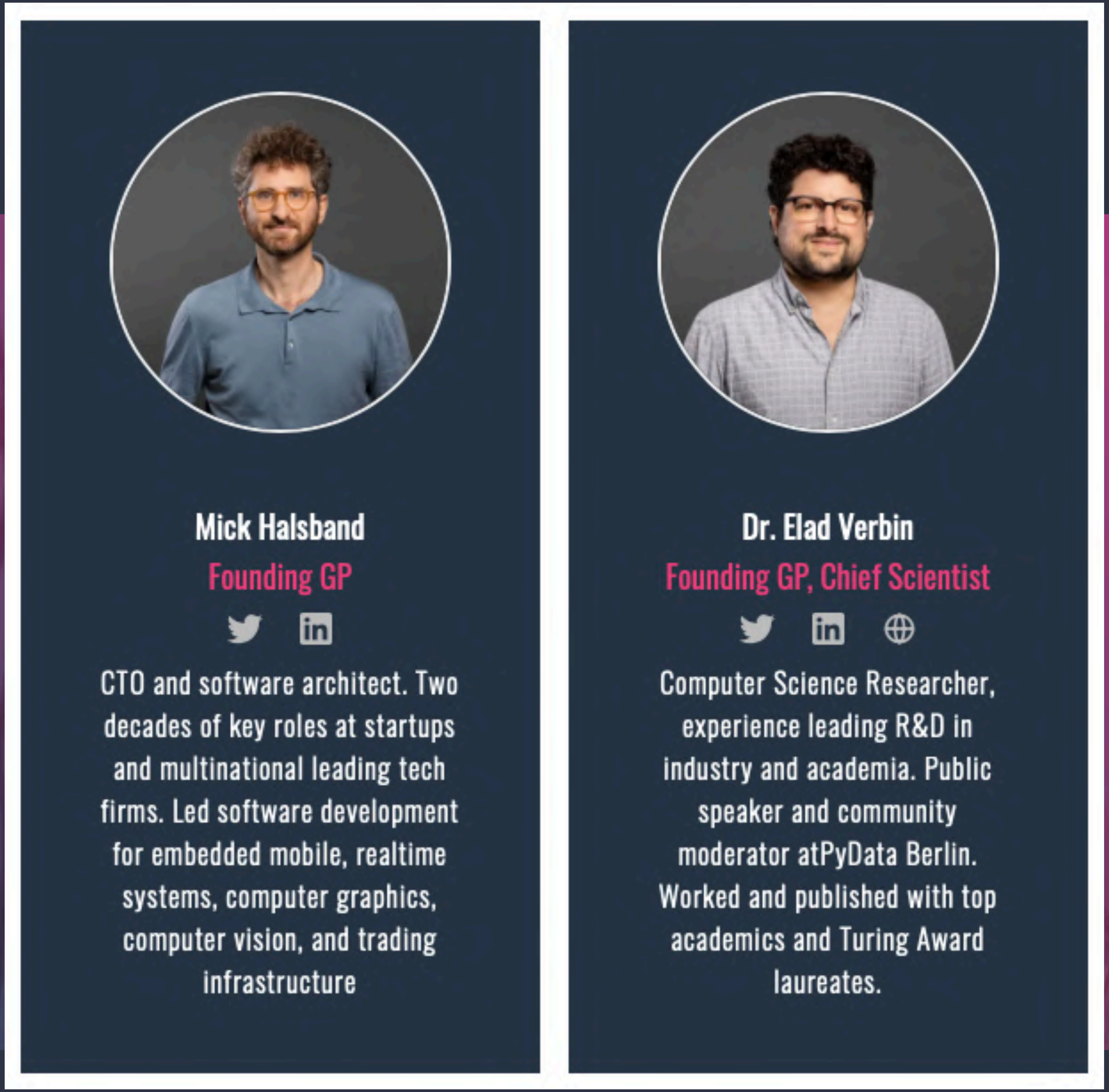
COMPANY

Investors

LUNAR VENTURES

Backing European Deep Tech Founders

If you're a technical founder who's struggled to explain your tech and vision to investors, we set up a fund for you. We invest pre-revenue to help turn your science fiction into reality.



Mick Halsband

Founding GP

CTO and software architect. Two decades of key roles at startups and multinational leading tech firms. Led software development for embedded mobile, realtime systems, computer graphics, computer vision, and trading infrastructure

Dr. Elad Verbin

Founding GP, Chief Scientist

Computer Science Researcher, experience leading R&D in industry and academia. Public speaker and community moderator atPyData Berlin. Worked and published with top academics and Turing Award laureates.

High-level proposition hypothesis

Low latency

- solve the global write-path latency problem
- help mainstream apps use low-latency CRDT tech
- snappy UX without failure code

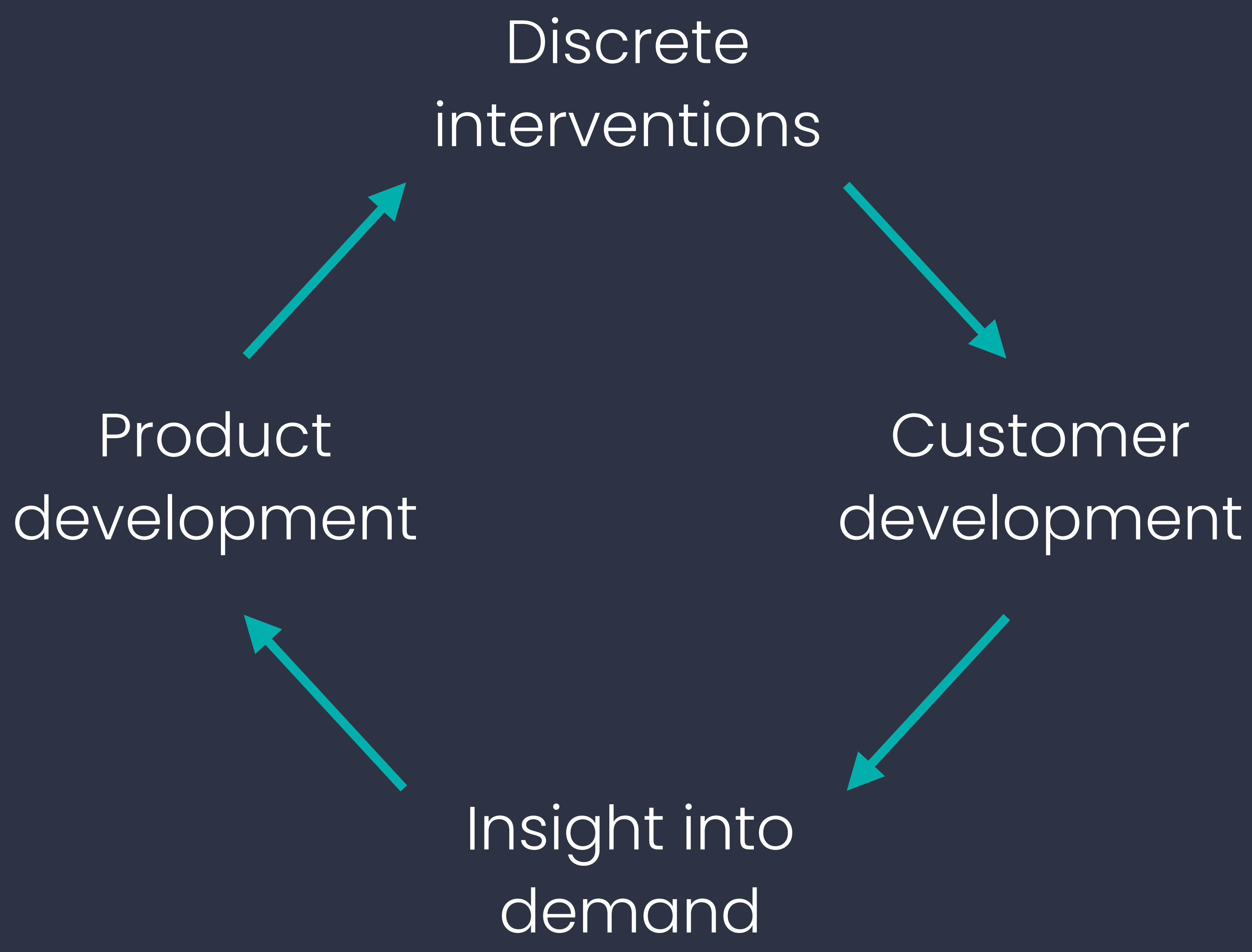
Collaboration

- real time, multi-player apps and collaboration tools
- immersive web, virtual worlds
- unify structured and collaborative data model

Geo-distribution

- orchestrate geo-distributed deployment topologies
- simplify engineering challenges
- data plane for edge/fass

Drill down on specific use-cases


Discrete interventions

Customer development

Insight into demand

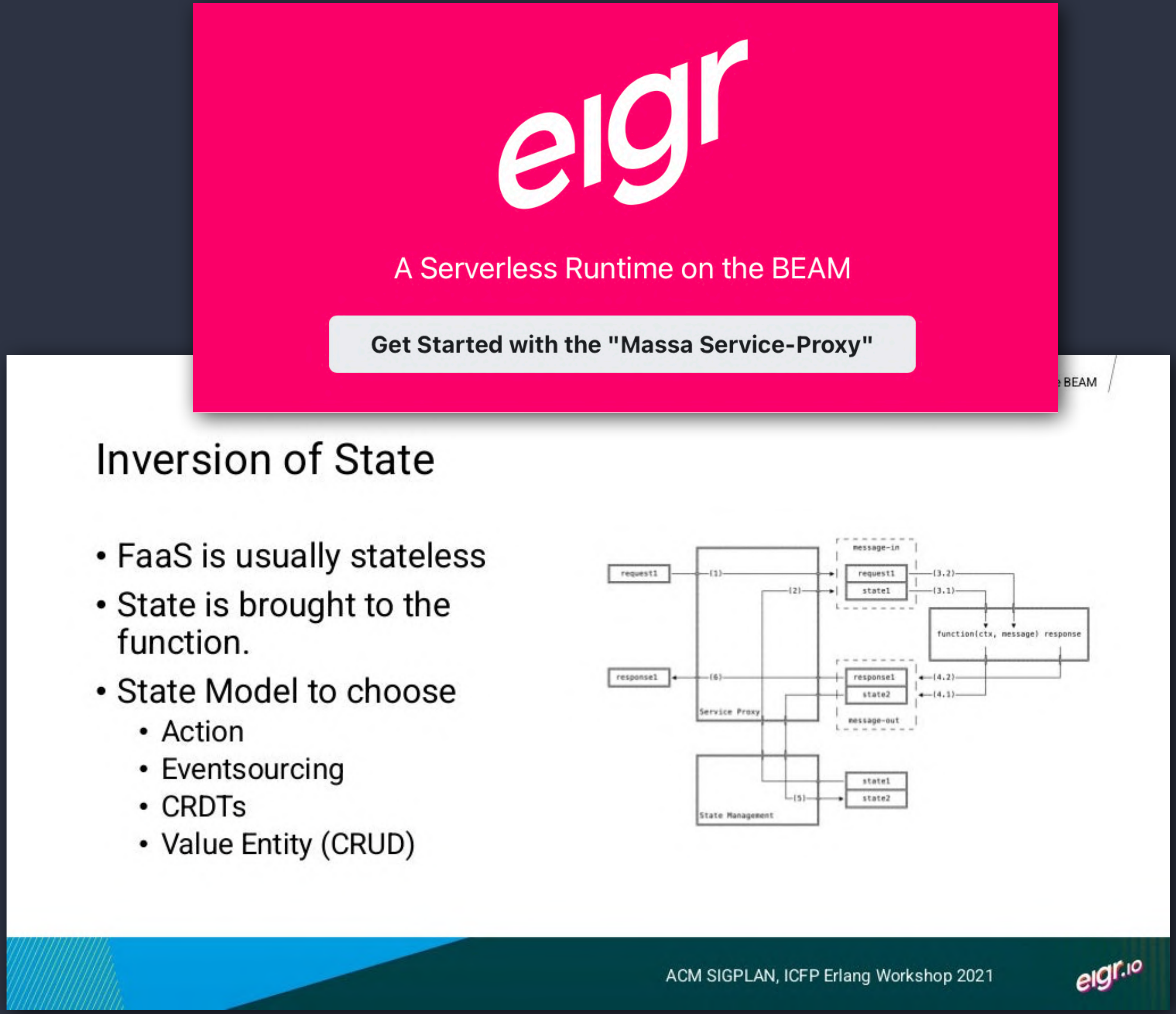
Product development

Customer segmentation	how tight is your ideal customer definition? can you identify common pain and buying characteristics?	0 - 5
Value proposition	do you have a consistent value proposition with strong evidence of willingness to pay?	0 - 5
Pricing	have you validated your pricing assumptions?	0 - 5
Impact	how much business value have you delivered?	0 - 5
TOTAL	(out of 20)	…

Desire paths / self-selection

Low latency geo-distribution	Engineering edge data plane	Vaxine?	
Snappy UX	Optimistic writes with failure code	Vaxine?	
Realtime collaboration	Custom multiplayer system	Vaxine?	
Genesis	Custom build	Product	Commodity

Edge data plane



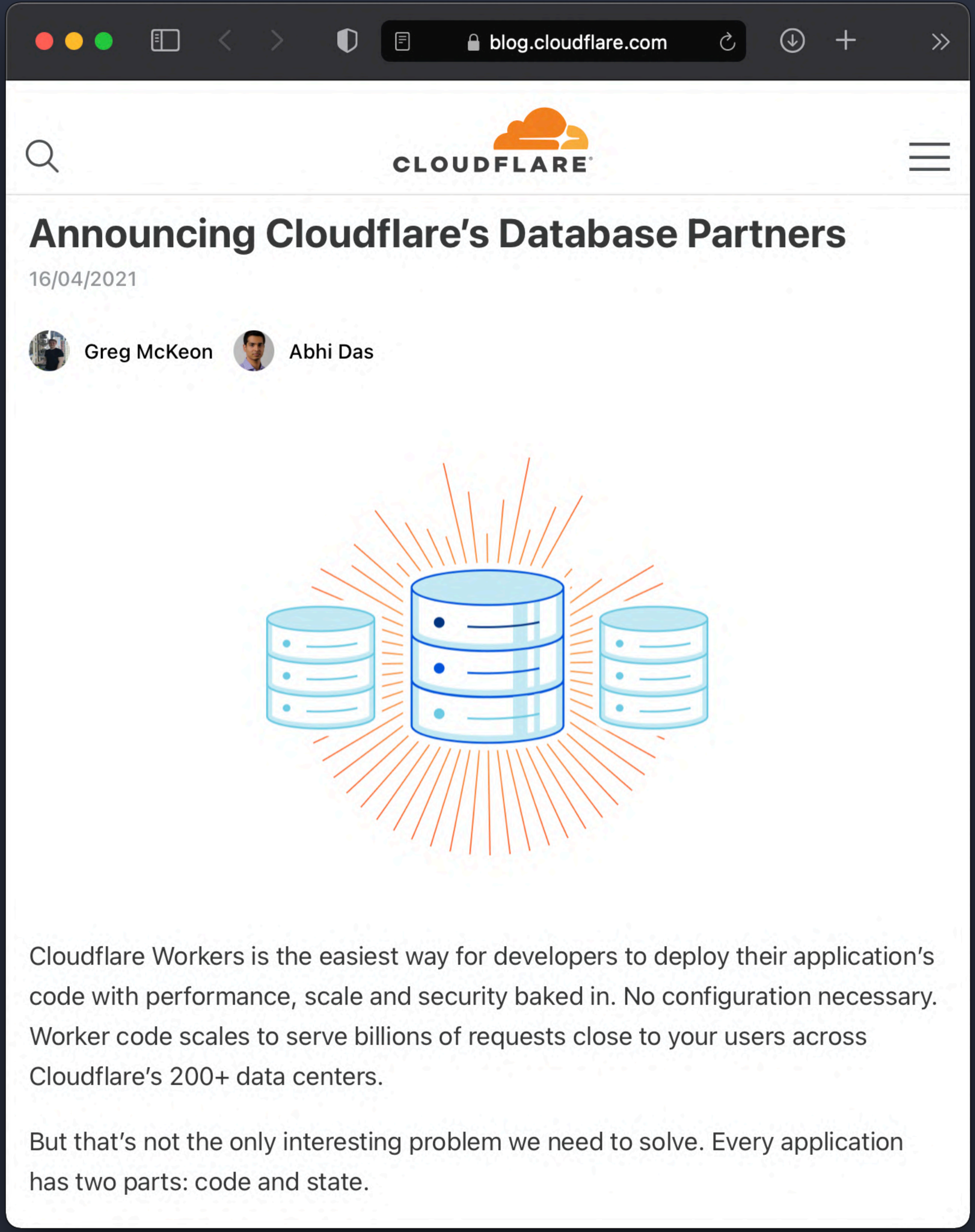
eigr

A Serverless Runtime on the BEAM

Inversion of State

- FaaS is usually stateless
- State is brought to the function.
- State Model to choose
  - Action
  - Eventsourcing
  - CRDTs
  - Value Entity (CRUD)

ACM SIGPLAN, ICFP Erlang Workshop 2021



Announcing Cloudflare's Database Partners

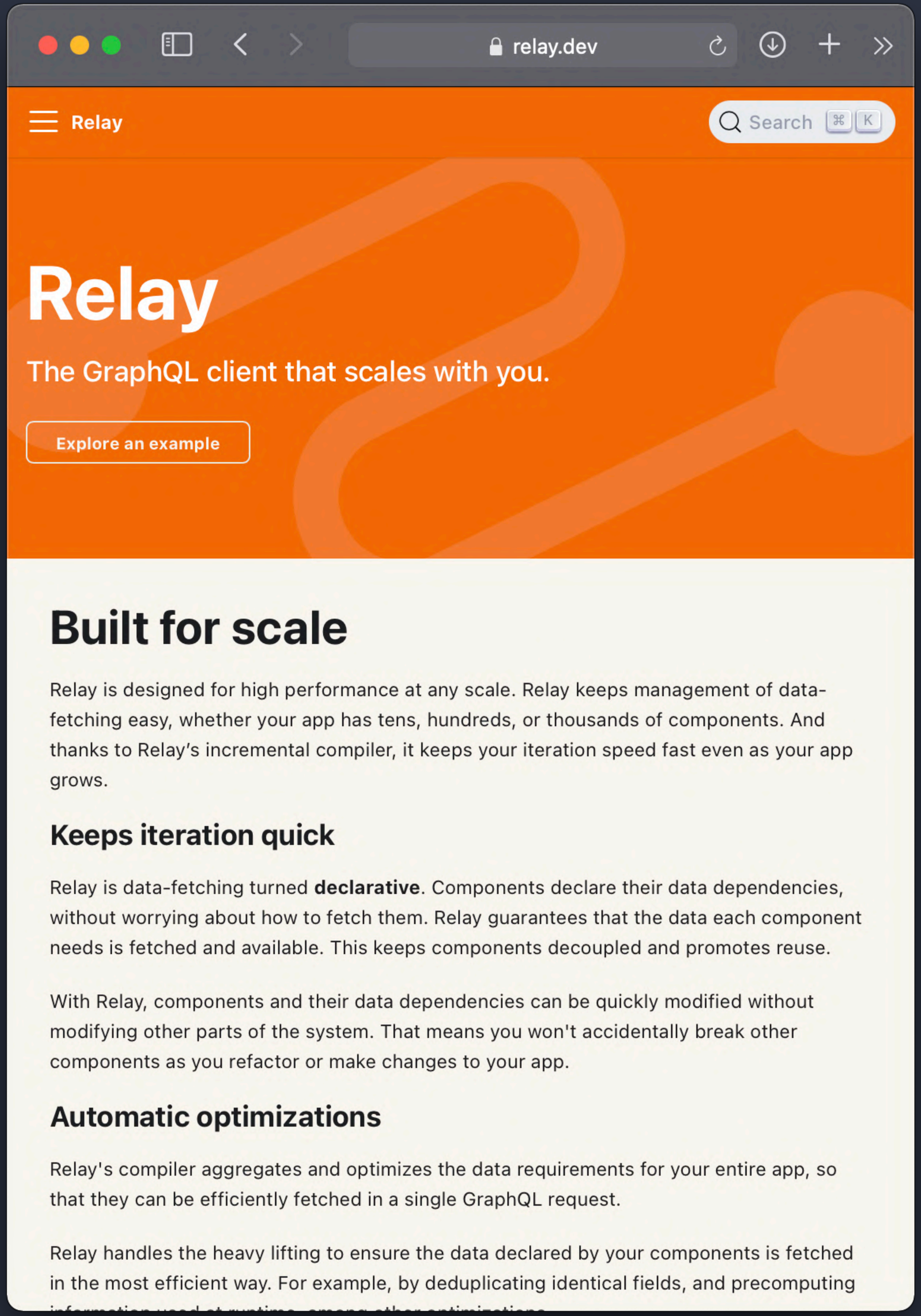
16/04/2021

Greg McKeon Abhi Das

Cloudflare Workers is the easiest way for developers to deploy their application's code with performance, scale and security baked in. No configuration necessary. Worker code scales to serve billions of requests close to your users across Cloudflare's 200+ data centers.

But that's not the only interesting problem we need to solve. Every application has two parts: code and state.

Snappy UX



Relay

The GraphQL client that scales with you.

Built for scale

Relay is designed for high performance at any scale. Relay keeps management of data-fetching easy, whether your app has tens, hundreds, or thousands of components. And thanks to Relay's incremental compiler, it keeps your iteration speed fast even as your app grows.

Keeps iteration quick

Relay is data-fetching turned **declarative**. Components declare their data dependencies, without worrying about how to fetch them. Relay guarantees that the data each component needs is fetched and available. This keeps components decoupled and promotes reuse.

With Relay, components and their data dependencies can be quickly modified without modifying other parts of the system. That means you won't accidentally break other components as you refactor or make changes to your app.

Automatic optimizations

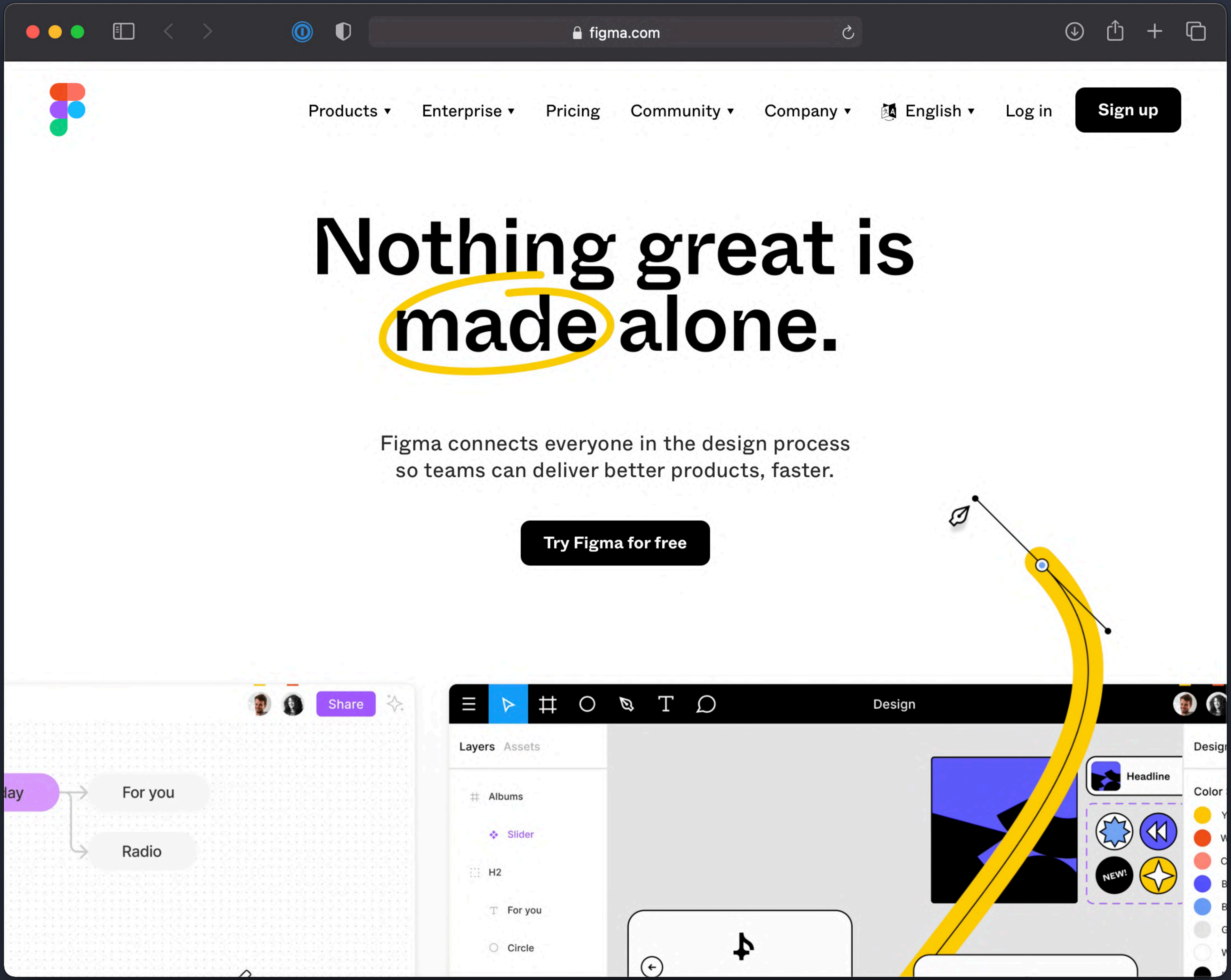
Relay's compiler aggregates and optimizes the data requirements for your entire app, so that they can be efficiently fetched in a single GraphQL request.

Relay handles the heavy lifting to ensure the data declared by your components is fetched in the most efficient way. For example, by deduplicating identical fields, and precomputing information used at runtime, among other optimizations.

```
import { commitMutation, graphql } from ‘react-relay';

const mutation = graphql`
  mutation ReadMessageMutation($input:
ReadMessageMutationInput!) {
    ReadMessage(input: $input) {
      message {
     status
      }
    }
  }
`;

commitMutation(
  env,
  {
    mutation,
    variables,
    optimisticResponse: {
      ReadMessage: {
        message: {status: ‘READ’}
      }
    }
    onCompleted: () => {} /* Mutation completed */,
    onError: error => {} /* Mutation errored */
  }
)
```

Multi-user

“When we first started building multiplayer functionality in Figma four years ago, we decided to develop our own solution.”



figma.com

Products ▾ Enterprise ▾ Pricing Community ▾ Company ▾ English ▾ Log in Sign up

Nothing great is made alone.

Figma connects everyone in the design process so teams can deliver better products, faster.

Try Figma for free

COMPANY

Join us!

Website

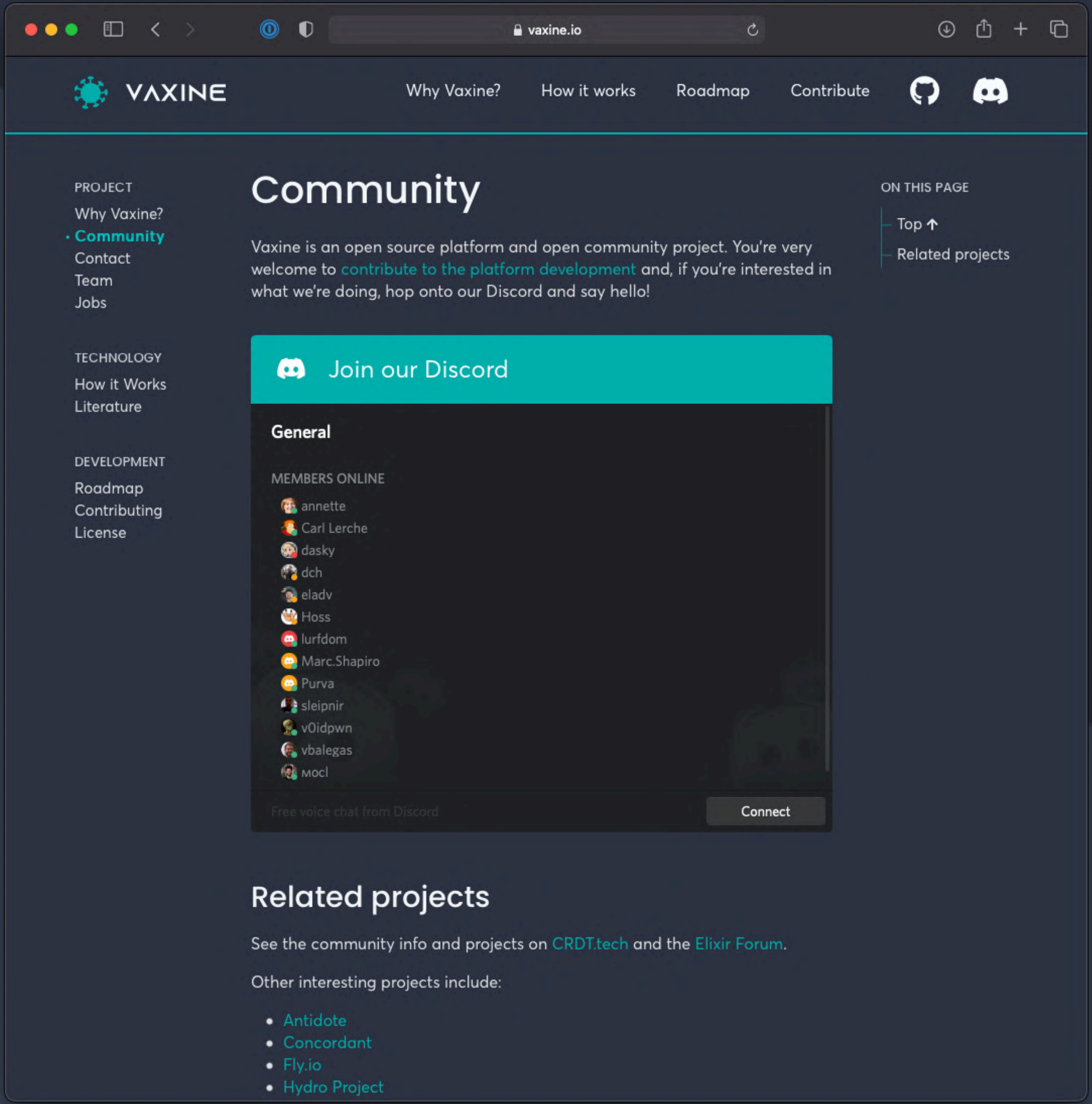
➡ https://vaxine.io

GitHub

➡ github.com/vaxine-io

Twitter

➡ @VaxineIO



Community

Vaxine is an open source platform and open community project. You're very welcome to contribute to the platform development and, if you're interested in what we're doing, hop onto our Discord and say hello!

Join our Discord

General

MEMBERS ONLINE

annette
Carl Lerche
dasky
dch
eladv
Hoss
lurfdom
Marc.Shapiro
Purva
sleipnir
v0idpwn
vbalegas
mocl

Free voice chat from Discord

Connect

Related projects

See the community info and projects on CRDT.tech and the Elixir Forum.

Other interesting projects include:

- Antidote
- Concordant
- Fly.io
- Hydro Project