Event sourcing in practice

Using Elixir to build event-driven applications

by Ben Smith

Photo by Amanda Jones on Unsplash

Talk outline

- Practical example of event sourcing in Elixir
- Where event sourcing can help, or hinder

Commanded

Use Commanded to build Elixir CQRS/ES applications

UK http://commanded.io/

Repositories 9 People 1 Teams 1 Projects 0 Settings

commanded

Use Commanded to build Elixir CQRS/ES applications

Elixir ★ 733 99

eventstore

Event store using PostgreSQL for persistence

Elixir ★ 475 62

commanded-ecto-projections

Read model projections for Commanded using Ecto

Elixir ★ 32 17

commanded-audit-middleware

Command auditing middleware for Commanded CQRS/ES applications

Elixir ★ 14 6

commanded-swarm-registry

Distributed process registry using Swarm for Commanded

Elixir ★ 5 3

commanded-scheduler

Schedule one-off and recurring commands for Commanded CQRS/ES applications

Elixir ★ 18 5

Let's start by introducing the application

STRAVA

Log In

The #1 app for runners and cyclists

Sign up with Facebook

Sign up with Google

or

Use my email

By signing up for Strava, you agree to the Terms of Service. View our Privacy Policy.

Already a Member? Log In

STRAVA

Ride 2,019 miles in 2019

Hosted by Segment Challenge

Join challenge

January 1, 2019 — December 31, 2019

GOAL	TIME REMAINING	ATTEMPTS	BY COMPETITORS
2019 miles	293 days	353	7

Can you ride 2,019 miles in the year 2019?

Join this challenge to track the total distance you've ridden towards the 2,019 mile target.

Challenge goal

This challenge has a total activity target of 2019 miles.

Athletes who complete this challenge by achieving the goal will receive a digital finisher's badge in their Trophy Case.

Challenge information

Activities included in the challenge are based on each athlete's local time zone.

Only **Ride** activities are allowed for this challenge. Manual entries, e-bike rides, and trainer rides are not eligible.

All activities logged during the challenge period must be uploaded to Strava no later than three days after a stage ends.

This challenge is organised by Segment Challenge but **anyone** can join.

Ride 2,019 miles in 2019

Hosted by Segment Challenge

Join challenge

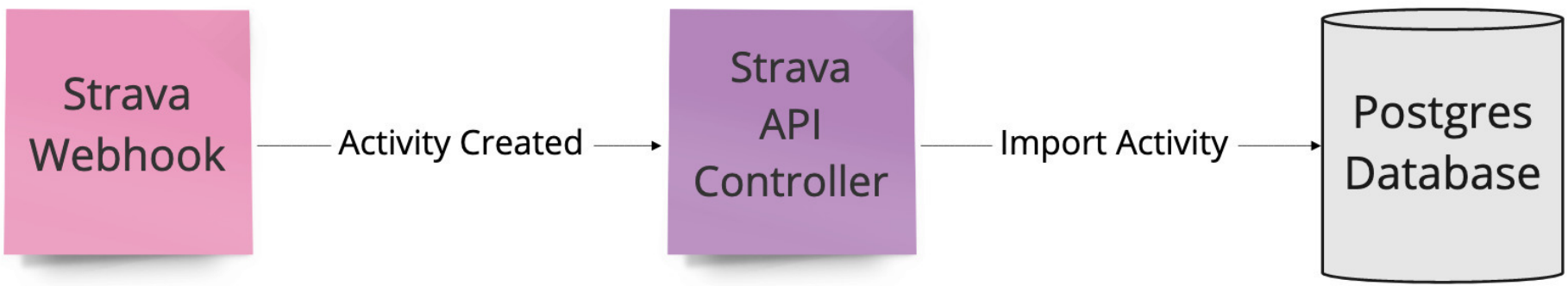
January 1, 2019 — December 31, 2019

About Activity Leaderboards

Overall Men | Overall Women

Rank	Name	Distance	Duration	Elevation	Progress		Activities
1	Tathagat Chatterjee CNG	**2009.6**	6d 4h 37m 28s	68577.6		100%	98
2	Utkarsh Verma	**1787.3**	6d 7h 19m 40s	60196.1		89%	130
3	Aaran Daniells	**431.1**	1d 4h 25m 5s	20676.6		21%	85
4	Robby Norwood	**407.4**	23h 13m 8s	17409.9		20%	12
5	Darran Cowell CCR	**398.8**	1d 4h 50m 51s	16063.1		20%	20
6	Ben Smith (VCV)	**94.4**	6h 13m 18s	6922.5		5%	5
7	Zenek Grychtoł	**72.9**	6h 46m 4s	3004.2		4%	3

```elixir
defmodule SegmentChallenge.Challenge do
  use SegmentChallenge.Web, :model

  schema "challenges" do
    field(:name, :string)
    field(:description, :string)
    field(:start_date, Ecto.DateTime)
    field(:start_date_local, Ecto.DateTime)
    field(:end_date, Ecto.DateTime)
    field(:end_date_local, Ecto.DateTime)
    field(:active, :boolean)
    field(:competitor_count, :integer, default: 0)
    field(:slug, :string)

    belongs_to(:club, Club)

    has_many(:challenge_participations, ChallengeParticipation)
    has_many(:competitors, through: [:challenge_participations, :athlete])

    timestamps()
  end
end
```


Strava Webhook
Activity Created
Strava API Controller
Import Activity
Postgres Database

```elixir
defmodule SegmentChallengeWeb.API.StravaController do
  use SegmentChallengeWeb, :controller

  def webhook(conn, %{"object_type" => "activity", "aspect_type" => "create"} = params) do
    %{"object_id" => strava_activity_id} = params

    SegmentChallenge.ActivityImport.execute(strava_activity_id)

    json(conn, "")
  end

  def webhook(conn, _params) do
    json(conn, "")
  end
end
```

```elixir
defmodule SegmentChallenge.ActivityImport do
  def execute(strava_activity_id) do
    {:ok, %Strava.Activity{} = activity} = get_strava_act

    for challenge <- athlete_active_challenges(activity)
      Repo.transaction(fn ->
        :ok = Challenge.import_activity(activity)
        :ok = Leaderboard.rank_leaderboard(challenge)
      end)
    end
  end
end
```

- Readable
- Extensible

But …

- Testable

  — domain logic mixed with data access
- Concurrency

  — requires Ecto's optimistic lock

SEGMENT CHALLENGE

Sorry, you've lost 8th place

You just lost 1 place on Stage 2 VCV Sleepers Hill.

You are now in 9th place.

There are **32 hours left** to attempt the stage before it ends. So get out there, ride hard and try again.

— Segment Challenge

Ben Smith (VCV)

Recorded an attempt at stage VCV Sleepers Hill of **3:19**

February 28, 2019 at 9:53am

```elixir
defmodule SegmentChallenge.ActivityImport do
  def execute(strava_activity_id) do
    {:ok, %Strava.Activity{} = activity} = get_strava_act

    for challenge <- athlete_active_challenges(activity)
      Repo.transaction(fn ->
        :ok = Challenge.import_activity(challenge, activi
        :ok = Leaderboard.rank_leaderboard(challenge)
        :ok = ActivityFeed.record_activity(activity)
        :ok = Email.send_lost_place_notification(activity
      end)
    end
  end
end
```

- All succeed, or nothing
- Latency

  — delegate to job queue
- Third party calls not transactional

  — use *outbox* pattern

```elixir
defmodule SegmentChallenge.ActivityImport do
  def execute(strava_activity_id) do
    {:ok, %Strava.Activity{} = activity} = get_strava_activity(strava_acti

    challenges = athlete_active_challenges(activity)

    multi =
      Enum.reduce(challenges, Ecto.Multi.new(), fn challenge, multi ->
        multi
        |> import_challenge_activity(challenge, activity)
        |> rank_challenge_leaderboard(challenge)
        |> record_activity_in_feed(activity)
        |> send_lost_place_email_notification(challenge)
      end)

    case Repo.transaction(multi) do
      {:ok, _changes} -> :ok
      {:error, _operation, _failure, _changes} -> {:error, :activity_impor
    end
  end
```

No real improvement

```elixir
defmodule SegmentChallenge.ActivityImport do
  def execute(strava_activity_id) do
    {:ok, %Strava.Activity{} = activity} = get_strava_activity(strava_activity_id)

    for challenge <- athlete_active_challenges(activity do
      with {:ok, activity} <- Challenge.import_activity(challenge, activity) do
        Registry.dispatch(SegmentChallenge.PubSub, :challenge, fn entries ->
          for {pid, _} <- entries do
            send(pid, {:activity_recorded, challenge, activity})
          end
        end)
      end
    end
  end
end
```

```elixir
defmodule SegmentChallenge.ChallengeActivityHandler do
  use GenServer

  def init(state) do
    {:ok, _} = Registry.register(SegmentChallenge.PubSub, :challenge

    {:ok, state}
  end

  @doc """
  Handle challenge activity recorded message.
  """
  def handle_info({:activity_recorded, challenge, activity}, state)
    :ok = Leaderboard.rank_leaderboard(challenge)

    {:noreply, state}
  end
end
```

- Loosely coupled components
- Extensible
- Isolation
- Testable

But …

- No message delivery guarantee

  — use a persistent message queue

Using domain events as the source of truth

Command function

```
execute(state, command) :: {:ok, [event]}
  | {:error, term}
```

```elixir
defmodule SegmentChallenge.Challenge do
  # Command functions
  def execute(%Challenge{}, %CreateChallenge{}), do: # ...
  def execute(%Challenge{}, %JoinChallenge{}), do: # ...
  def execute(%Challenge{}, %ImportActivity{}), do: # ...
end
```

```elixir
defmodule SegmentChallenge.Challenge do
  defstruct [:id, :start_date_local, :end_date_local, competitors: MapSet.new(), activities: []]

  def execute(%Challenge{id: id} = challenge, %ImportActivity{} = command) do
    %ImportActivity{
      activity_id: activity_id,
      athlete_id: athlete_id,
      start_date_local: start_date_local
    } = command

    with :ok <- validate_new_activity(challenge, activity_id),
         :ok <- validate_is_competitor(challenge, athlete_id),
         :ok <- validate_within_challenge_period(challenge, start_date_local) do
      event = struct(ActivityRecorded, Map.from_struct(command))

      {:ok, [event]}
    else
      {:error, error} -> {:error, error}
    end
  end
```

State mutator function

```
apply(state, event) :: state
```

```elixir
defmodule SegmentChallenge.Challenge do
  # State mutators
  def apply(%Challenge{}, %ChallengeCreated{}), do: # ...
  def apply(%Challenge{}, %CompetitorJoinedChallenge{}, do: # ...
  def apply(%Challenge{}, %ActivityRecorded{}), do: # ...
end
```

```elixir
defmodule SegmentChallenge.Challenge do
  defstruct [:id, :start_date_local, :end_date_local, competitors: MapSet.new(), activities: []]

  def apply(%Challenge{} = challenge, %CompetitorJoinedChallenge{} = event) do
    %Challenge{competitors: competitors} = challenge
    %CompetitorJoinedChallenge{athlete_id: athlete_id} = event

    %Challenge{challenge | competitors: MapSet.put(competitors, athlete_id)}
  end

  def apply(%Challenge{} = challenge, %ActivityRecorded{} = event) do
    %Challenge{activities: activities} = challenge
    %ActivityRecorded{athlete_id: athlete_id} = event

    %Challenge{challenge | activities: [event | activities]}
  end
end
```

Example usage

```
challenge = %Challenge{}

{:ok, events} = Challenge.execute(challenge, command)

challenge = Enum.reduce(events, challenge, &Challenge
```

- Hosted in a `GenServer` process
- Requests serialised
- Initial state reduced from existing events
- Events persisted to an event store

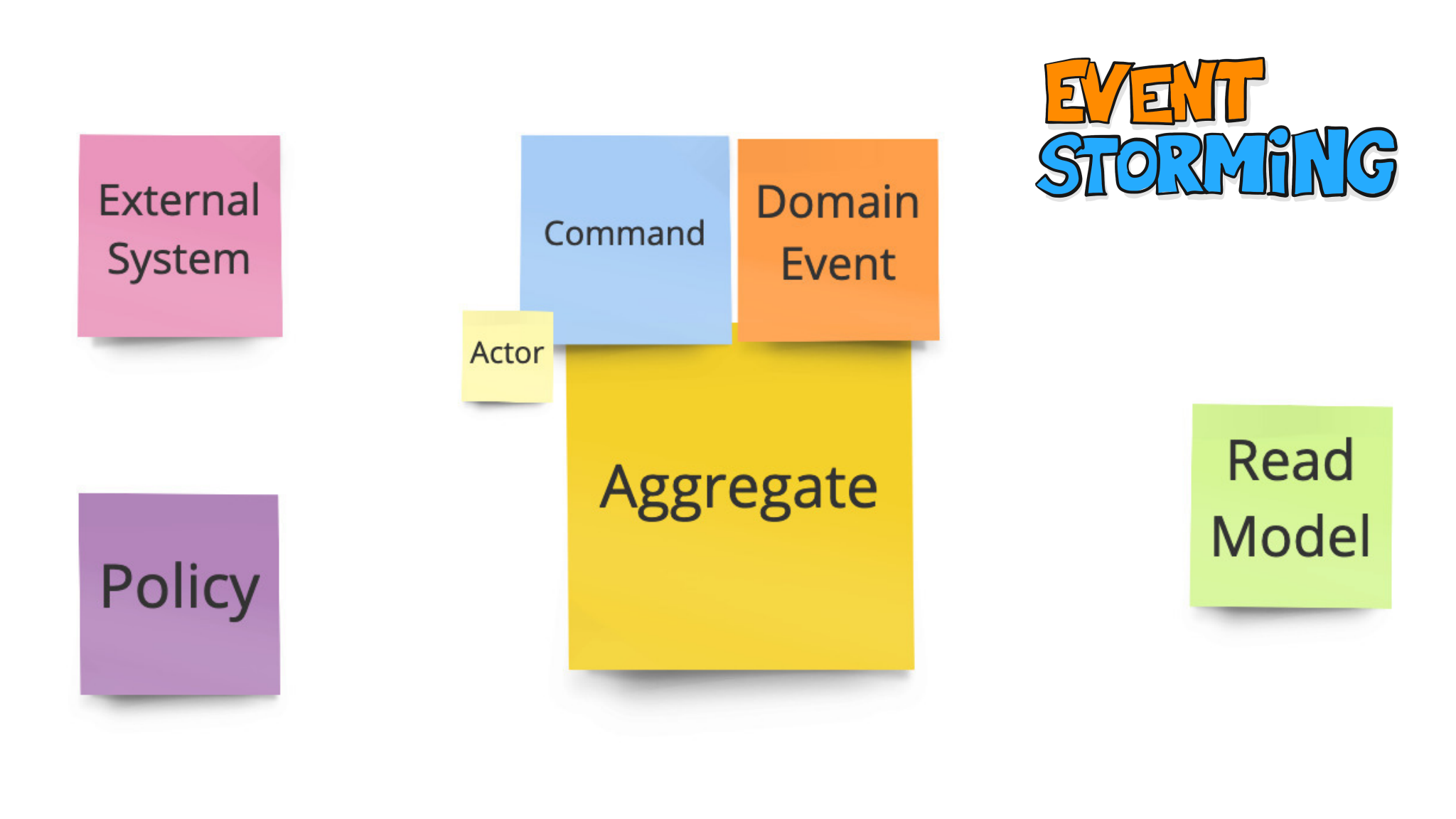
```elixir
defmodule SegmentChallenge.ChallengeTest do
  use SegmentChallenge.AggregateCase, aggregate: SegmentChallenge.Challenge

  describe "challenge aggregate" do
    test "should record activity for competitor within challenge period" do
      assert_events(
        [
          %ChallengeCreated{
            id: "1",
            start_date_local: ~N[2019-03-01 00:00:00],
            end_date_local: ~N[2019-03-31 23:59:59]
          },
          %CompetitorJoinedChallenge{id: "1", athlete_id: "2"}
        ],
        %ImportActivity{athlete_id: "2", start_date_local: ~N[2019-03-28 10:27:15]},
        [
          %ActivityRecorded{athlete_id: "2", start_date_local: ~N[2019-03-28 10:27:15]}
        ]
      )
    end
  end
end
```

```elixir
defmodule SegmentChallenge.ChallengeTest do
  use SegmentChallenge.AggregateCase, aggregate: SegmentChallenge.Challenge

  describe "challenge aggregate" do
    test "should exclude activity for non-comptitor" do
      assert_error(
        [
          %ChallengeCreated{
            id: "1",
            start_date_local: ~N[2019-03-01 00:00:00],
            end_date_local: ~N[2019-03-31 23:59:59]
          }
        ],
        %ImportActivity{athlete_id: "2", start_date_local: ~N[2019-03-28 1(
        {:error, :not_a_competitor}
      )
    end
  end
end
```

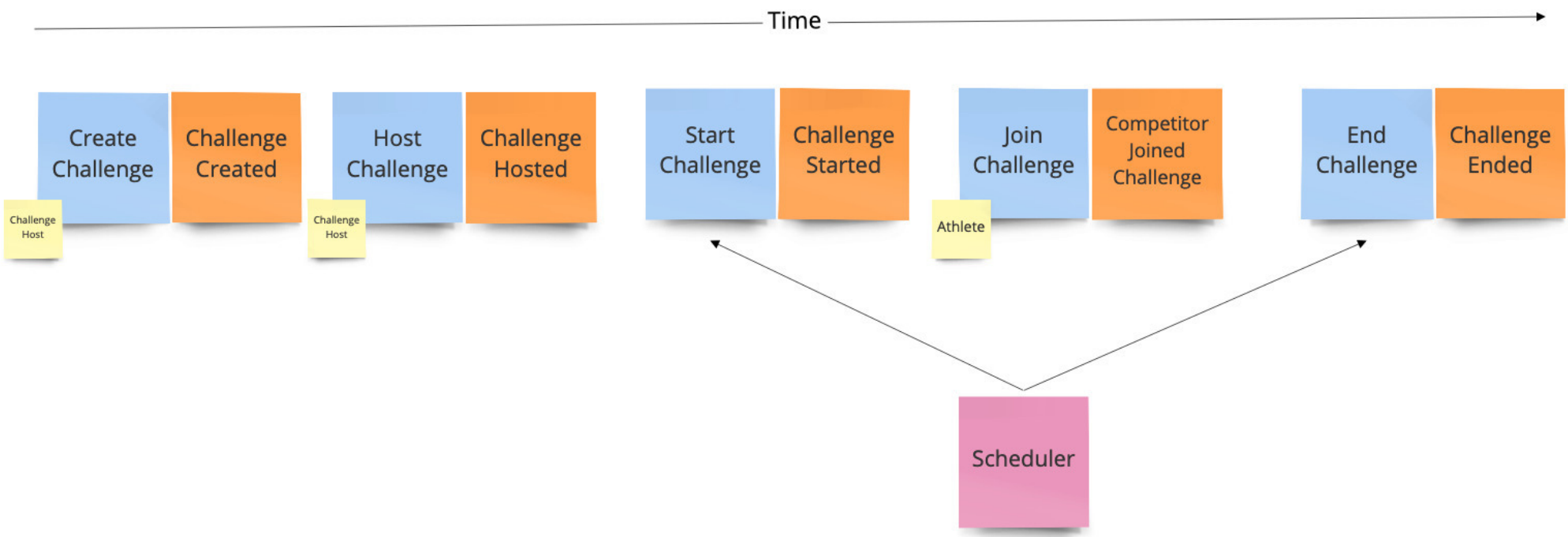
- Domain specific language
- Support `async: true`
- Fast tests

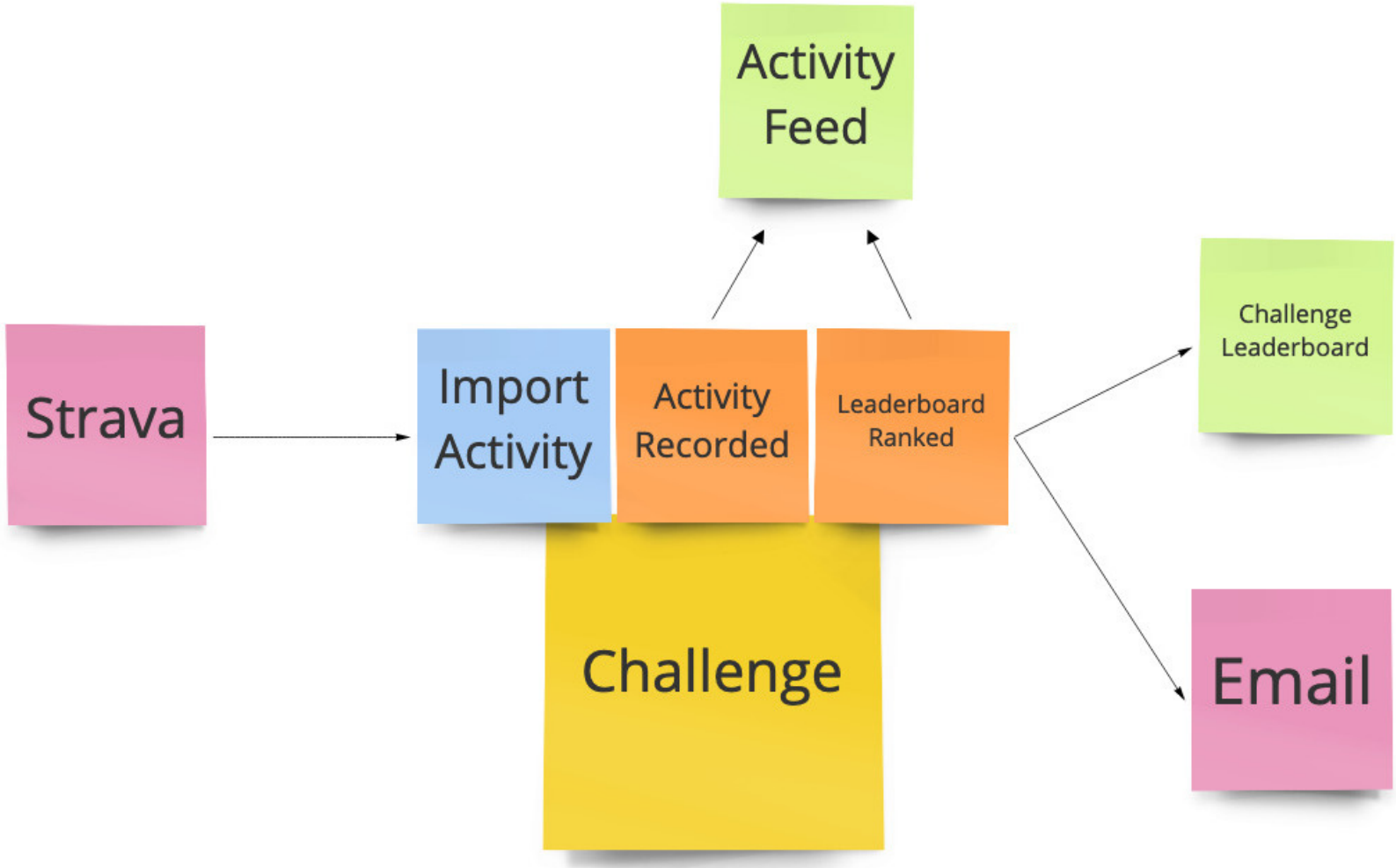
Command and state mutation functions are pure — IO and side-affect free

How do I design using events?


External System
Command
Domain Event
Actor
Aggregate
Policy
Read Model
EVENT STORMING

Challenge lifecycle

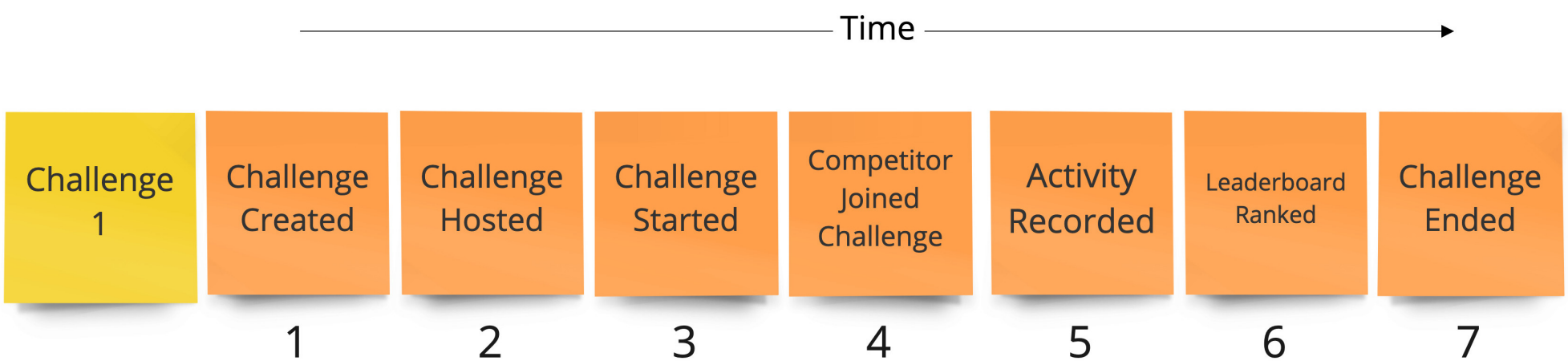



Activity Feed
Strava
Import Activity
Activity Recorded
Leaderboard Ranked
Challenge Leaderboard
Challenge
Email

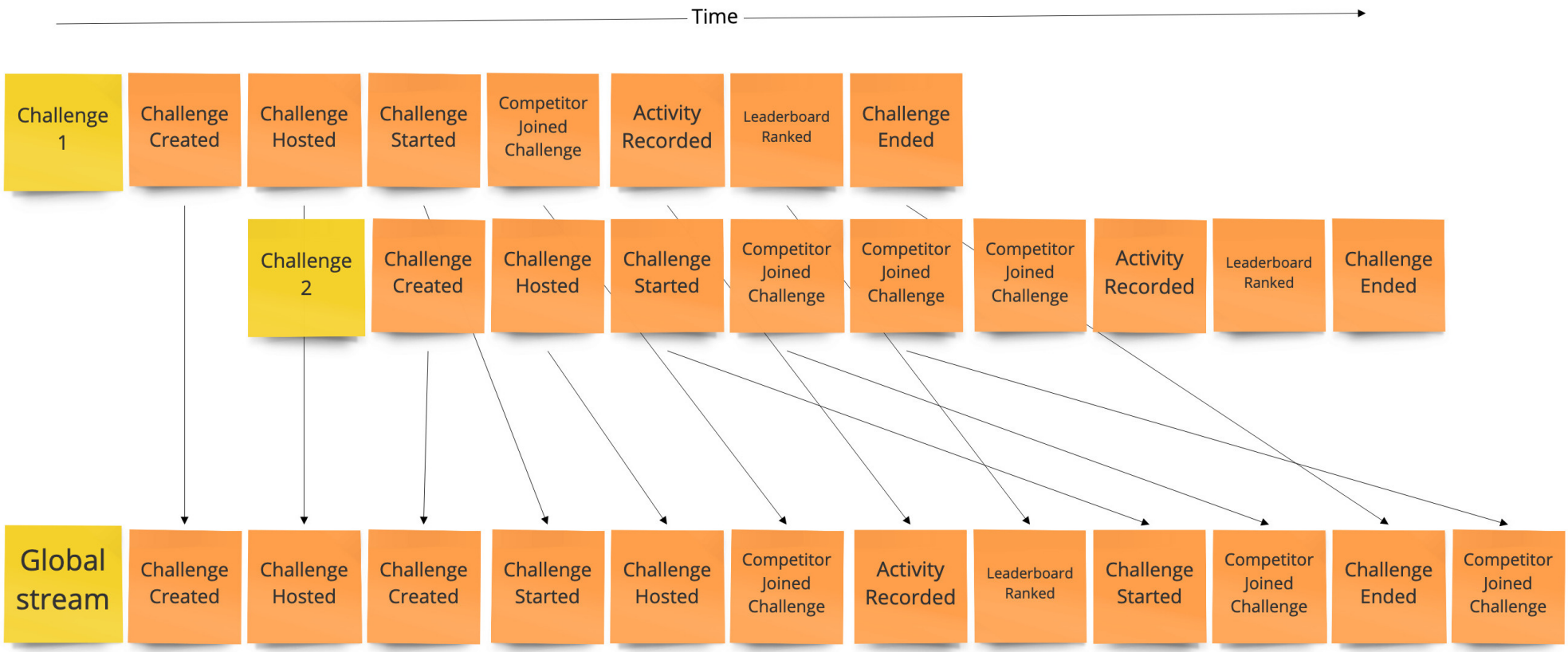
```
%ChallengeCreated{id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b", name: "Ride 2,0189 miles in 2019"},
%ChallengeHosted{id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b"},
%ChallengeStarted{id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b"},
%CompetitorJoinedChallenge{id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b", competitor: "athlete-1234"},
%ActivityRecorded{
  id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b",
  athlete: "athlete-1234",
  start_date_local: ~N[2019-01-17 10:25:16],
  distance_in_metres: 42417.8,
  moving_time_in_seconds: 6078
},
%LeaderboardRanked{
  id: "42abbe22-a93b-411b-b0b1-c11e3ccad77b",
  rankings: [
    %LeaderboardRanked.Ranking{
      rank: 1,
      athlete_uuid: "athlete-1234",
      total_distance_in_metres: 42417.8,
      total_moving_time_in_seconds: 6078
    }
  ],
  new_positions: [%LeaderboardRanked.Position{rank: 1, athlete_uuid: "athlete-1234"}],
  positions_gained: [],
  positions_lost: []
}
```

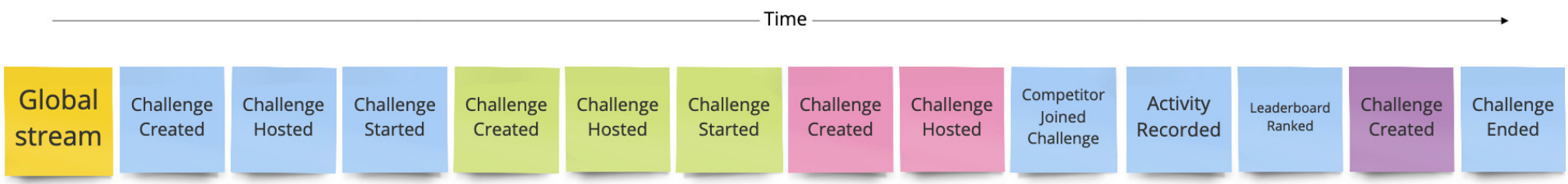
```elixir
defmodule SegmentChallenge.LostPlaceEmailNotification do
  use Commanded.Event.Handler, name: __MODULE__

  @doc """
  Handle leaderboard ranked events where an athlete has lost a position.
  """
  def handle(%LeaderboardRanked{} = event, _metadata) do
    %LeaderboardRanked{positions_lost: positions_lost} = event

    for position <- positions_lost do
      :ok = Email.send_lost_place_notification(position)
    end

    :ok
  end
end
```

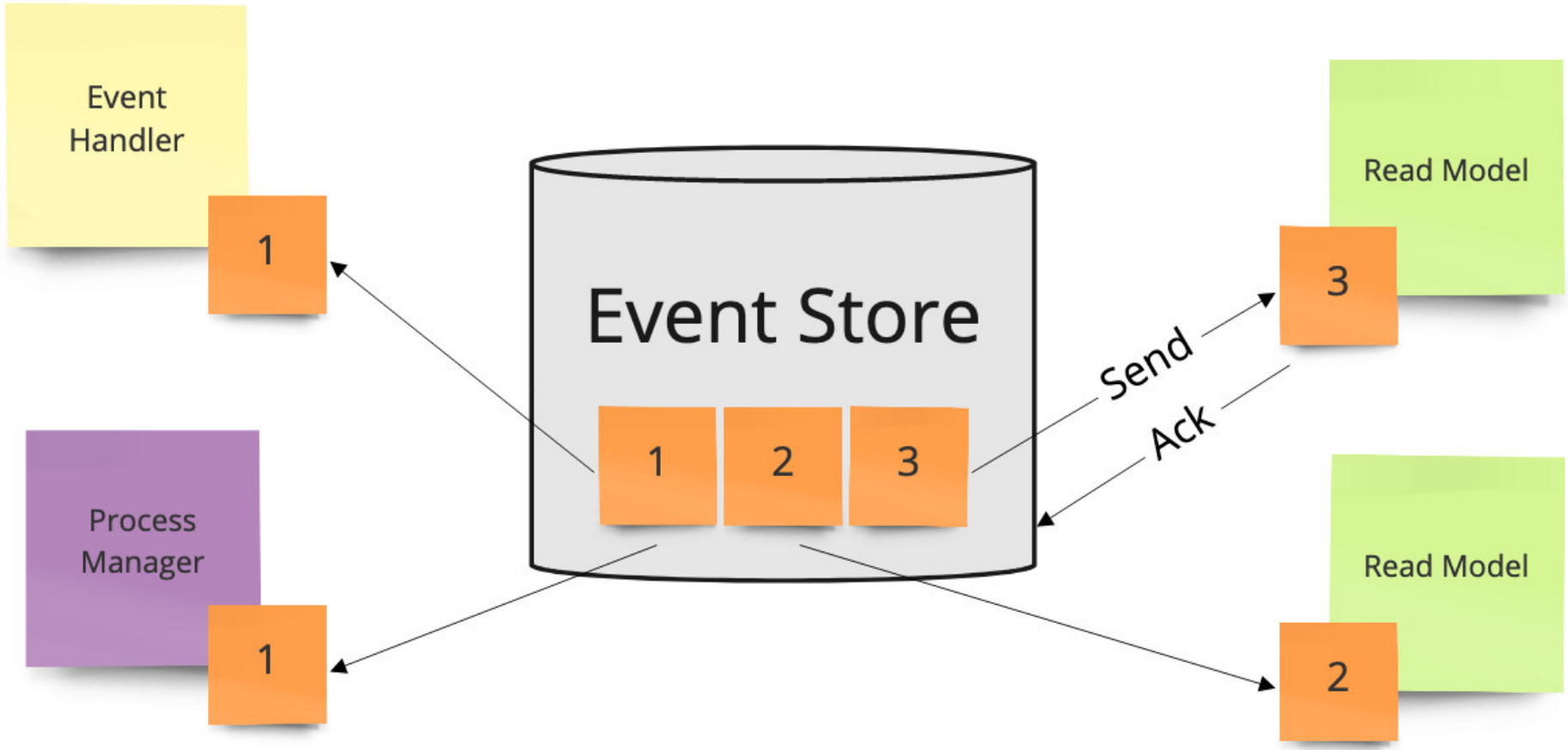
```elixir
defmodule SegmentChallenge.ChallengeProjector do
  use Commanded.Projections.Ecto, name: __MODULE__

  project %ActivityRecorded{} = event, fn multi ->
    projection = to_projection(Activity, event)

    Ecto.Multi.insert(multi, :activity, projection)
  end

  project %LeaderboardRanked{} = event, fn multi ->
    %LeaderboardRanked{rankings: rankings} = event

    Enum.reduce(rankings, multi, fn ranking, multi ->
      projection = to_projection(LeaderboardEntry, ranking)

      Ecto.Multi.insert(multi, :leaderboard_entry, projection)
    end)
  end
```

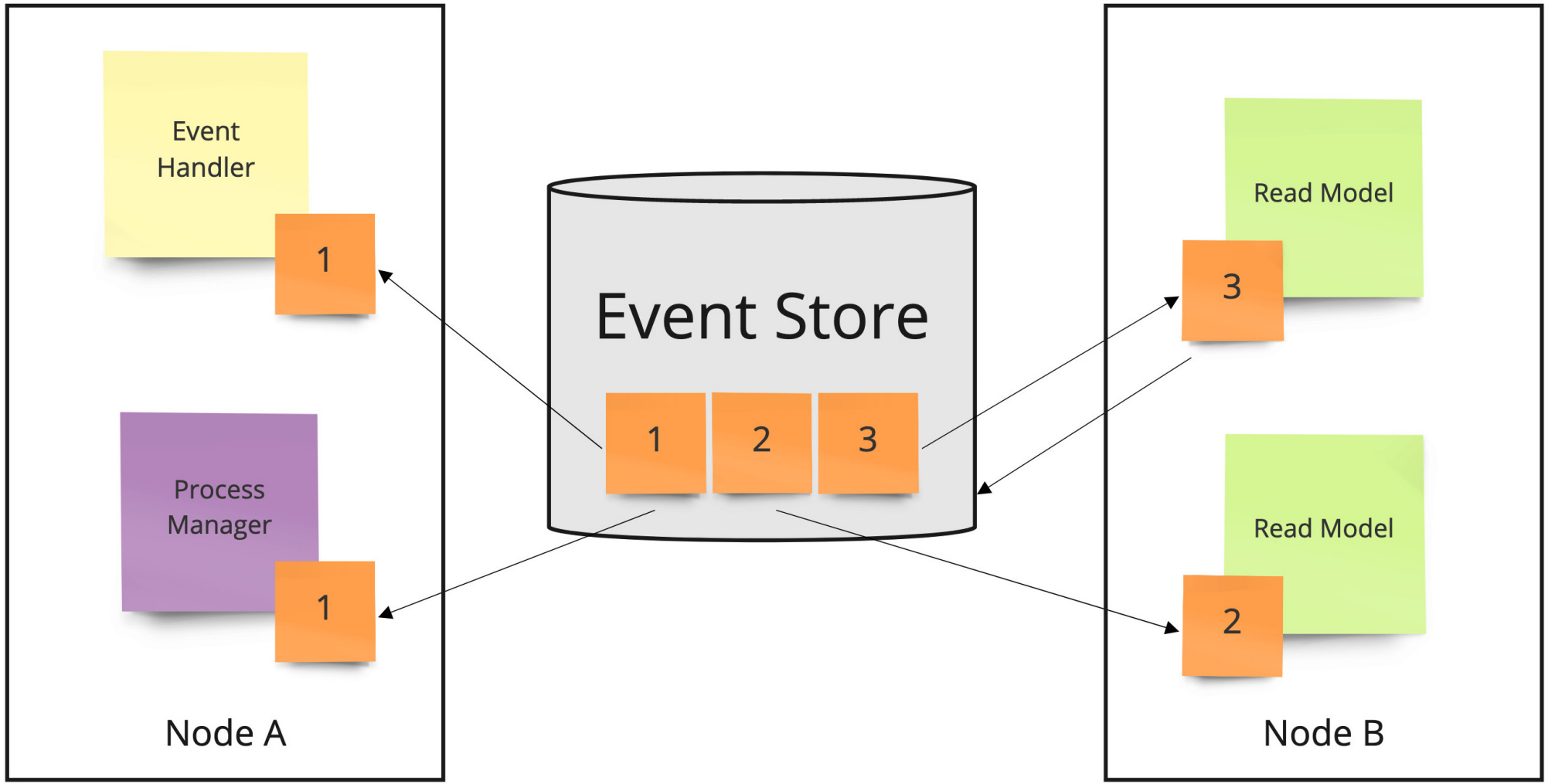

Time
Challenge 1
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Ended
1
2
3
4
5
6
7


Time
Challenge 1
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Ended
Challenge 2
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Competitor Joined Challenge
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Ended
Challenge 3
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Challenge 4
Challenge Created

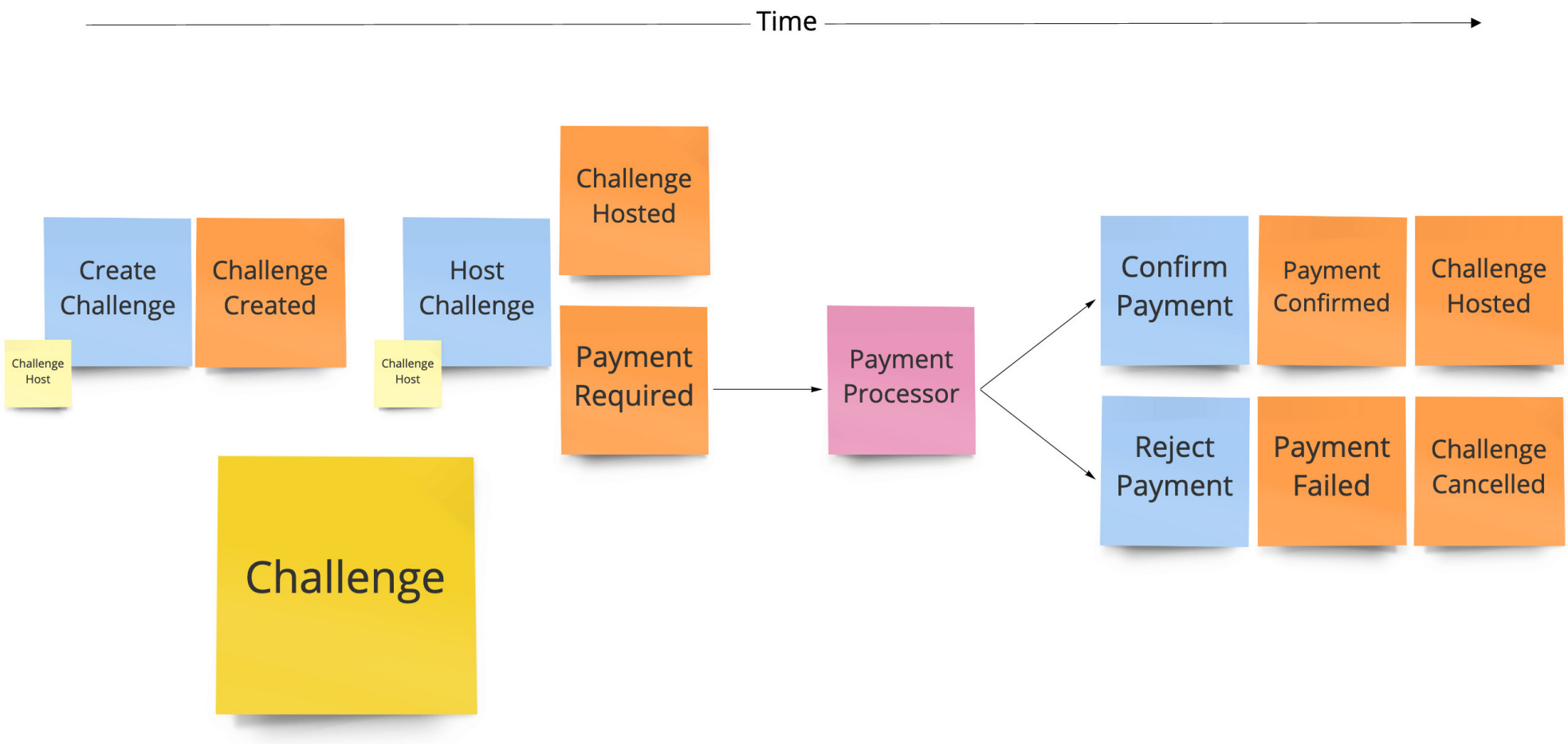

Time
Challenge 1
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Ended
Challenge 2
Challenge Created
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Competitor Joined Challenge
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Ended
Global stream
Challenge Created
Challenge Hosted
Challenge Created
Challenge Started
Challenge Hosted
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Started
Competitor Joined Challenge
Challenge Ended
Competitor Joined Challenge


Time
Global stream
Challenge Created
Challenge Hosted
Challenge Started
Challenge Created
Challenge Hosted
Challenge Started
Challenge Created
Challenge Hosted
Competitor Joined Challenge
Activity Recorded
Leaderboard Ranked
Challenge Created
Challenge Ended


Event Handler
1
Event Store
1
2
3
Read Model
3
Send
Ack
Process Manager
1
Read Model
2

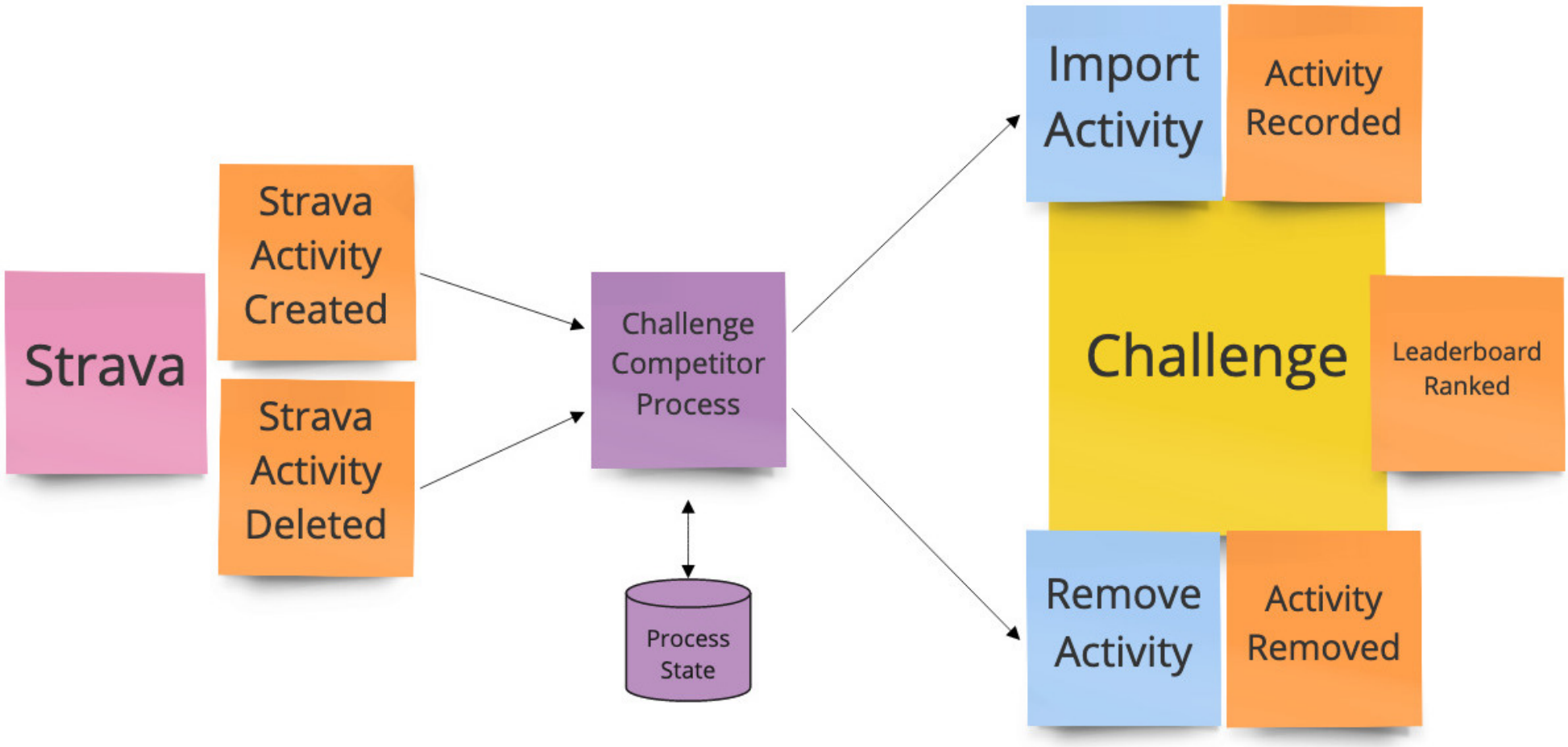

Event
Handler
1
Process
Manager
1
Node A
Event Store
1
2
3
Read Model
3
Read Model
2
Node B

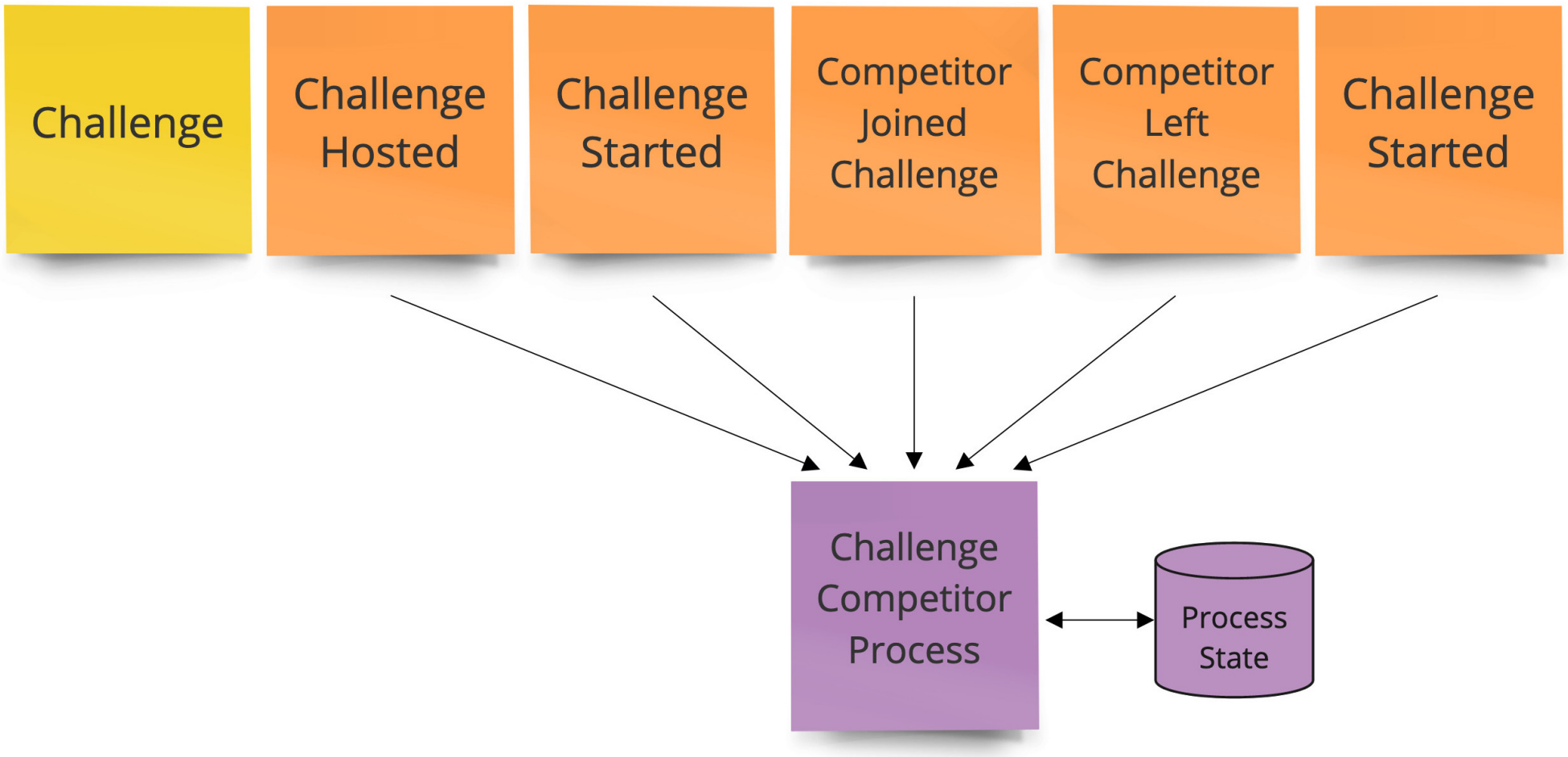
What about changing requirements?

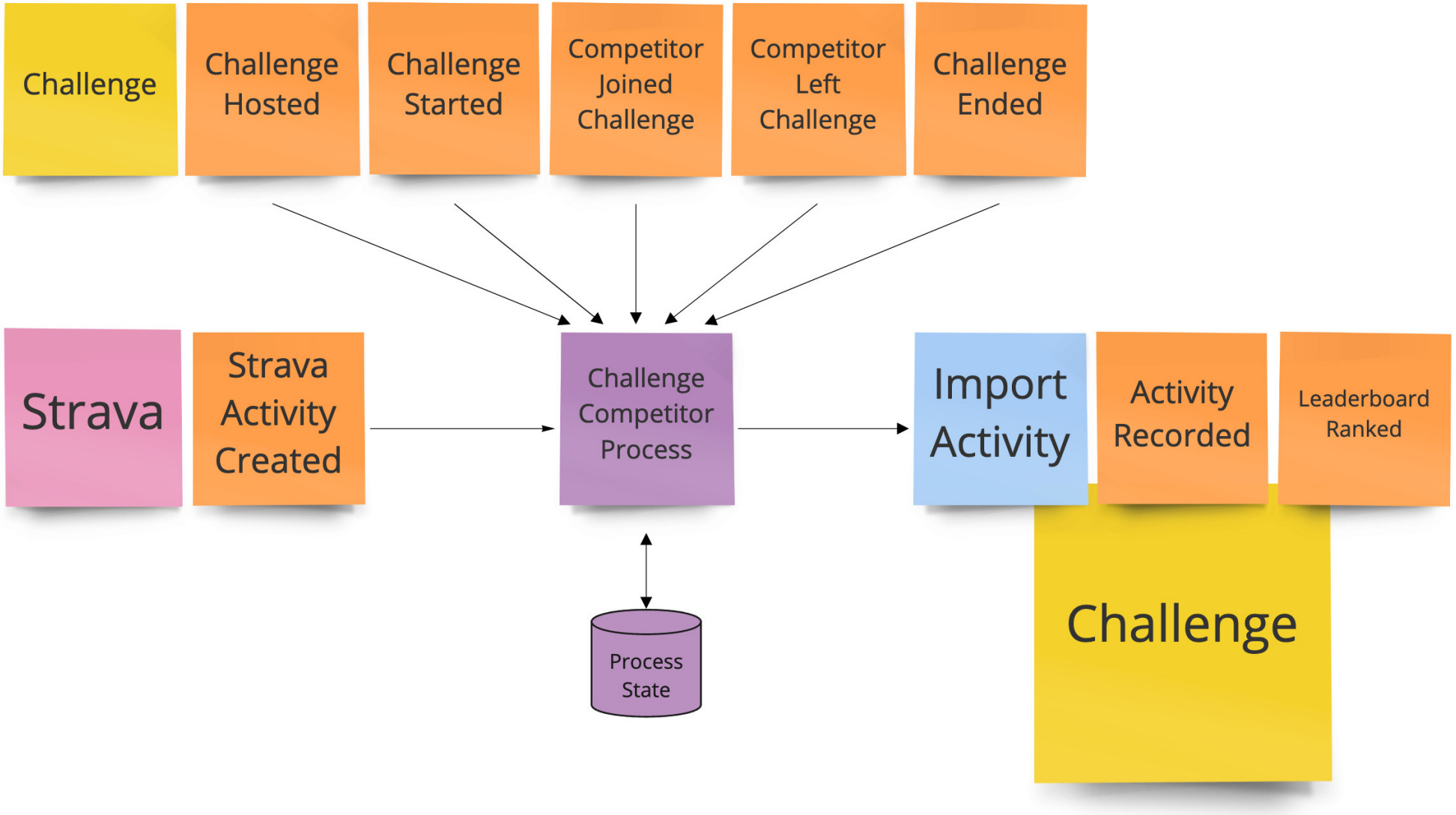

Time
Create Challenge
Challenge Created
Challenge Host
Host Challenge
Challenge Host
Challenge Hosted
Payment Required
Payment Processor
Confirm Payment
Payment Confirmed
Challenge Hosted
Reject Payment
Payment Failed
Challenge Cancelled
Challenge

Dealing with external events

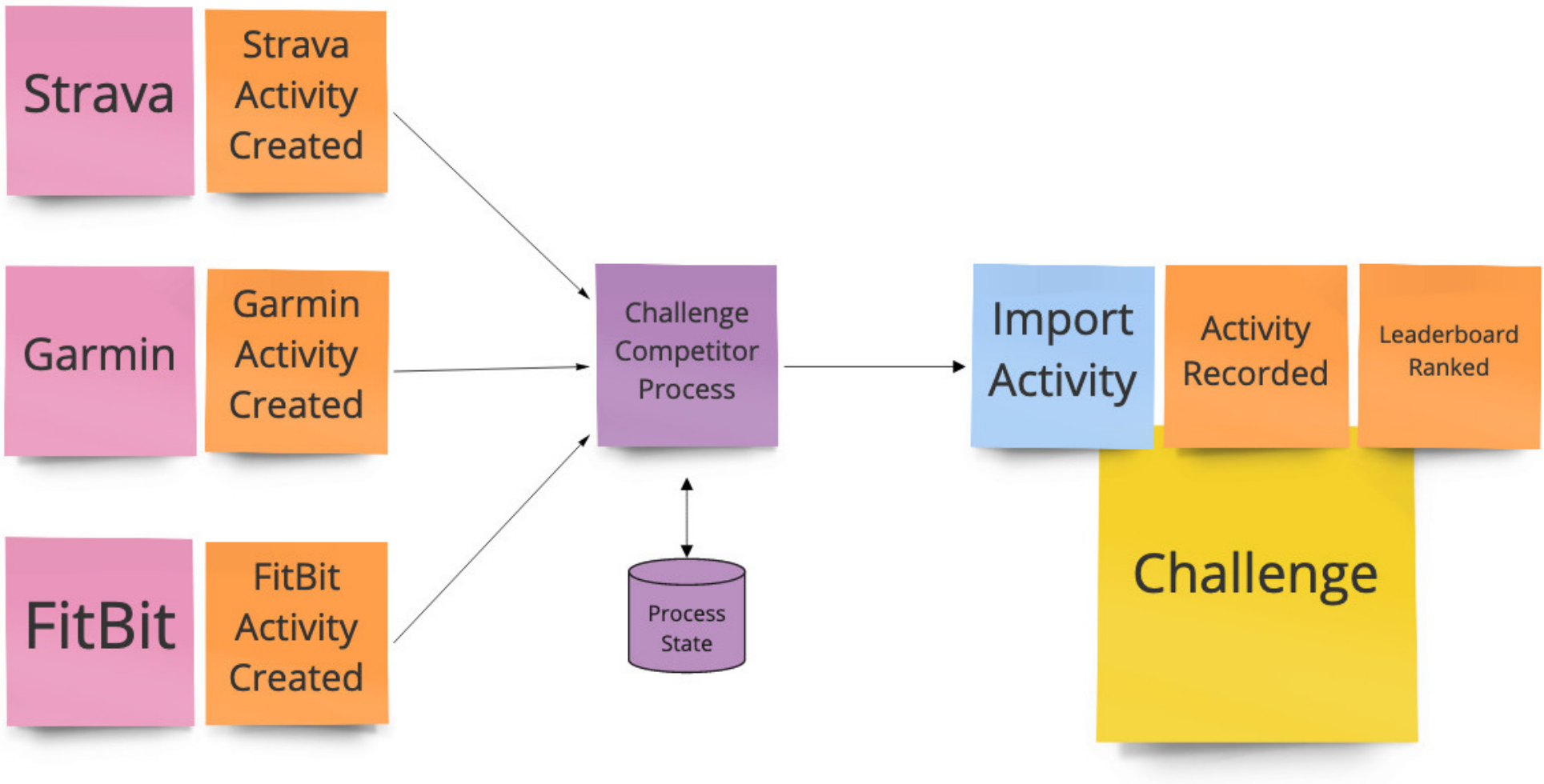
```elixir
defmodule SegmentChallenge.StravaActivityRecorder do
  def execute(strava_activity_id) do
    {:ok, %Strava.Activity{} = activity} = get_strava_activity(strava_activity_id)

    events = [
      struct(StravaActivityCreated, Map.from_struct(activity))
    ]

    :ok = EventStore.append_to_stream("strava", :any_version, events)
  end
end
```


Strava
Strava Activity Created
Strava Activity Deleted
Challenge Competitor Process
Process State
Import Activity
Activity Recorded
Challenge
Leaderboard Ranked
Remove Activity
Activity Removed


Challenge
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Competitor Left Challenge
Challenge Started
Challenge Competitor Process
Process State


Challenge
Challenge Hosted
Challenge Started
Competitor Joined Challenge
Competitor Left Challenge
Challenge Ended
Strava
Strava Activity Created
Challenge Competitor Process
Import Activity
Activity Recorded
Leaderboard Ranked
Process State
Challenge

Extending third party providers


Strava
Strava Activity Created
Garmin
Garmin Activity Created
FitBit
FitBit Activity Created
Challenge Competitor Process
Process State
Import Activity
Activity Recorded
Leaderboard Ranked
Challenge

Questions?

Want to learn more?

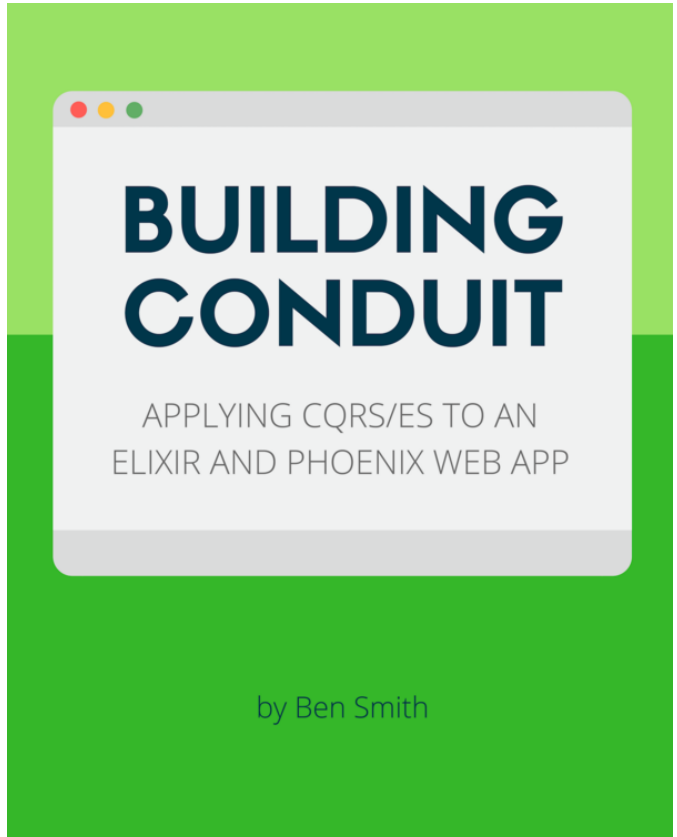
- **Commanded**, **EventStore** and related open source projects

  github.com/commanded

- **Building Conduit**

  — *Applying CQRS/ES to an Elixir and Phoenix web app*

  http://bit.ly/buildingconduitbook



Get in touch

- **Email** ben@10consulting.com
- **Web** 10consulting.com
- **Slides** 10consulting.com/presentations/event-sourcing-in-practice/

I'm available to help your company become event-driven.