UPDATE your VIEW on DELETE

The benefits of Event Sourcing

Sebastian von Conrad - @envato - @vonconrad

🇸🇪 🇺🇸 🇦🇺

envato

themeforest activeden

videohive photodune

graphicriver 3docean

codecanyon audiojungle

envato studio envato tuts+

In the top 300 of the world's largest websites.

themeforest

activeden

videohive

photodune

graphicriver

3docean

codecanyon

audiojungle

envato studio

envato tuts+

Event Sourcing.

Sebastian von Conrad - @envato - @vonconrad

(With an appearance of CQRS architecture.)

What is Event Sourcing?

What is an Event?

This poor bugger is horrendously overloaded.

An event is a business fact…

…that happened at a particular time.

- Appointment Scheduled
- Appointment Rescheduled
- Appointment Location Moved
- Appointment Cancelled
- Invitation Extended
- Invitation Notification Sent
- Invitation Accepted
- Invitation Declined

- Item Version Submitted
- Item Version Approved
- Item Added To Cart
- Item Removed From Cart
- Item Licence Purchased
- Item Support Purchased
- Withdrawal Request Submitted
- Withdrawal Completed

Event Sourcing, in this context.

Standard practice is store the current state of an object in a database using an ORM.

When state changes, the DB representation changes.

John wants to take his partner’s last name.

ORM

```
#<User id: 216, first_name: “John”,
last_name: “Reed” ...>

+ --- + ---------- + --------- + --- +
| id  | first_name | last_name | ... |
| --- | ---------- | --------- | --- |
| 216 | John       | Reed      | ... |
| ... | ...        | ...       | ... |
+ --- + ---------- + --------- + --- +
```

ORM

```
@user.update(last_name: “Hill”)
```

```
+ --- + ---------- + --------- + --- +
| id  | first_name | last_name | ... |
| --- | ---------- | --------- | --- |
| 216 | John       | Hill      | ... |
| ... | ...        | ...       | ... |
+ --- + ---------- + --------- + --- +
```

Event Sourcing doesn’t do that.

An append-only set of immutable events as source of truth.

Derive everything else from the events.

Source the current state by replaying events.

Event Sourcing

```
{
  user_id: 216,
  event_type: “signed_up”,
  body:
    { first_name: “John”, last_name: “Reed” }
}

#<User id: 216, first_name: “John”,
last_name: “Reed”, ...>
```

Event Sourcing

```
@user.change_name(last_name: “Hill”)

{
  user_id: 216,
  event_type: “name_changed”,
  body:
    { last_name: “Hill” }
}
```

Event Sourcing

```
{
  event_type: “sign_up”,
  body: { first_name: “John”, last_name: “Reed” }
},
{
  event_type: “name_changed”,
  body: { last_name: “Hill” }
}

#<User id: 216, first_name: “John”,
last_name: “Hill”, ...>
```

Make everything else completely disposable.

Including
current state.

Language agnostic.

Why Event Sourcing?

DELETE is evil.

Every UPDATE is a DELETE.

…so UPDATE is evil too.

Business concepts at the heart of the system.

It’s tried and tested. (For centuries.)

You're using it already and would refuse to do it any other way.

Current state is hard to distribute.

Append-only streams of events are easy to distribute.

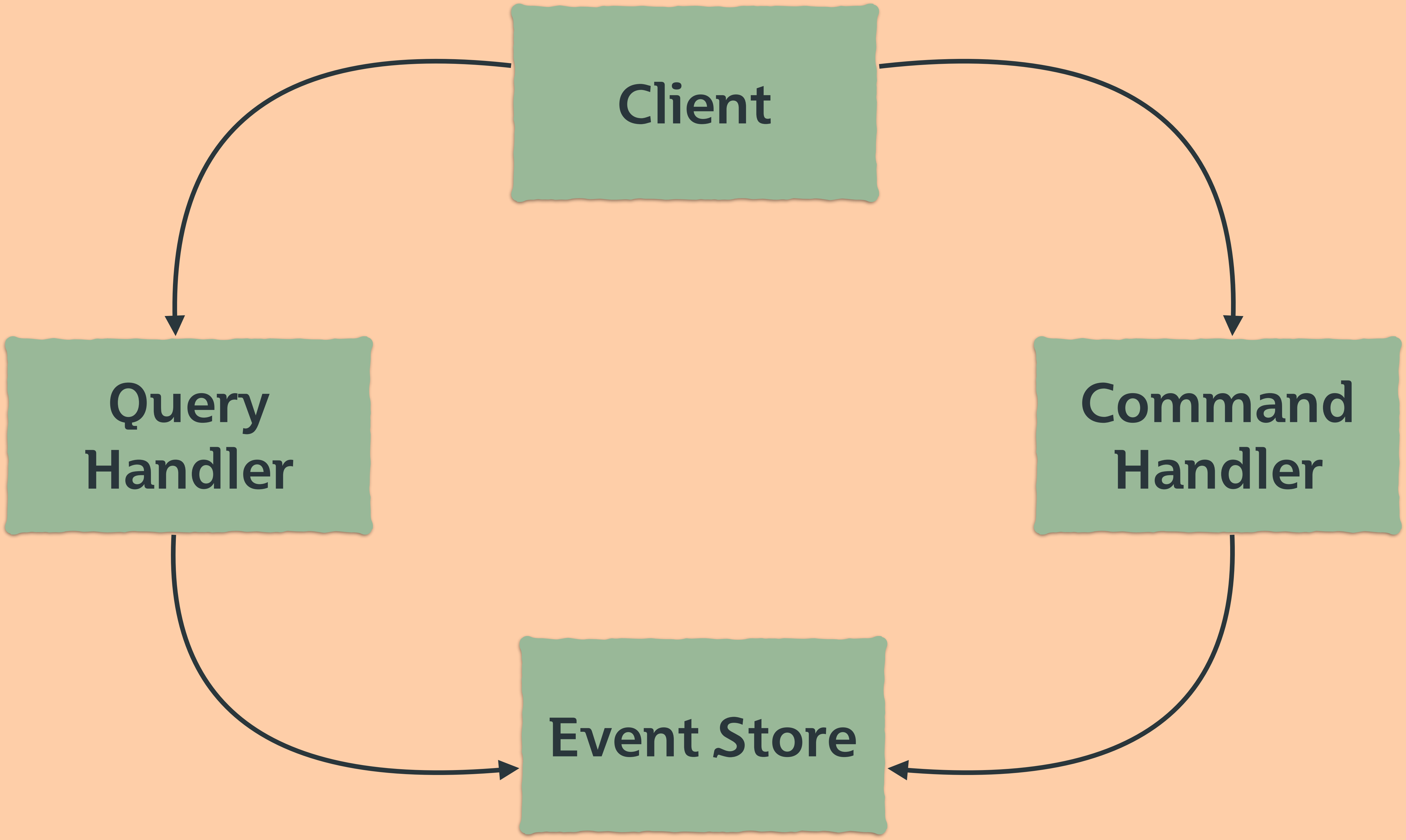
Last but not least: free time machine!

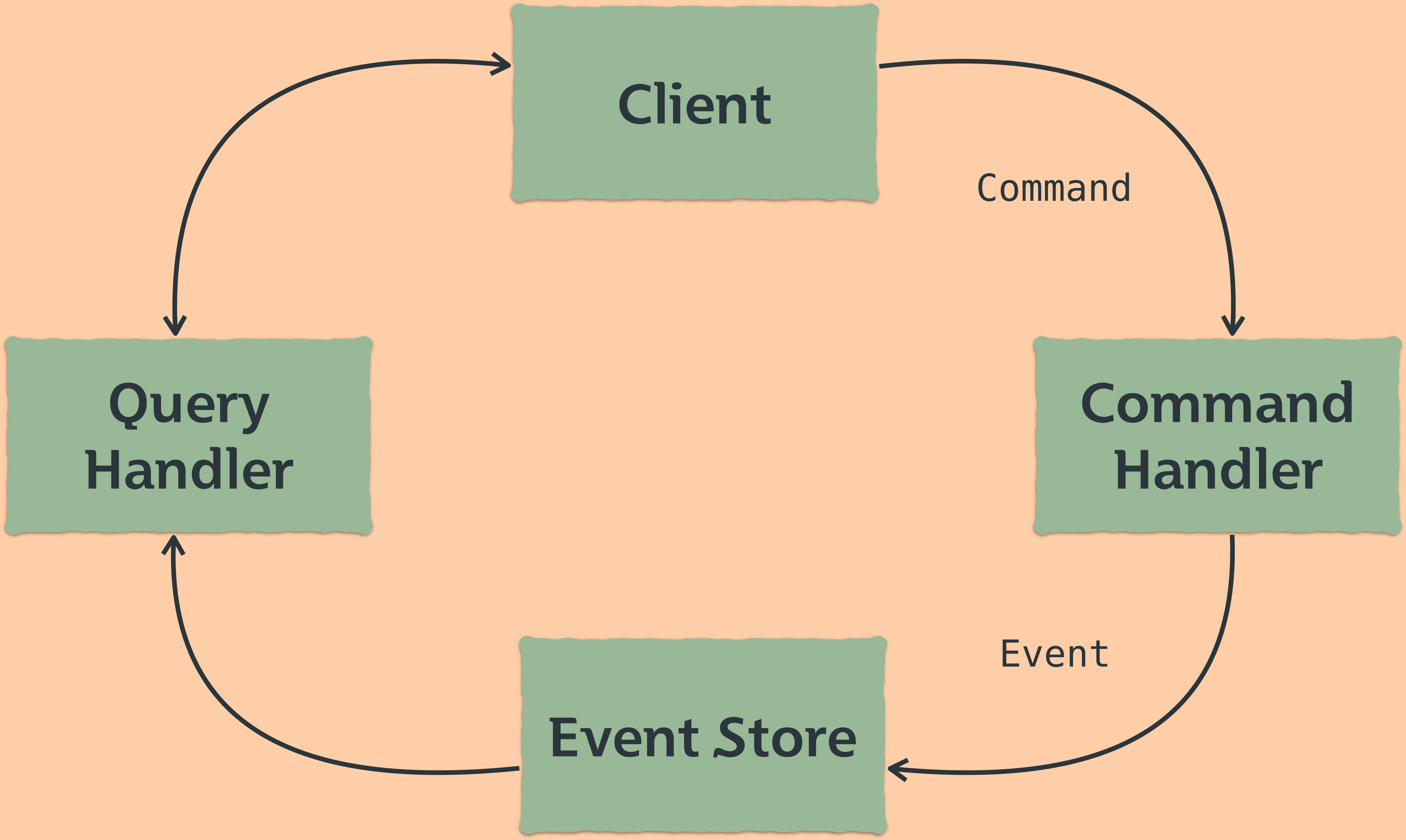
CQRS

CQS: methods can read (queries) or write (commands) but not both.

CQRS: objects have queries or commands but not both.

In our world we take it one step further.

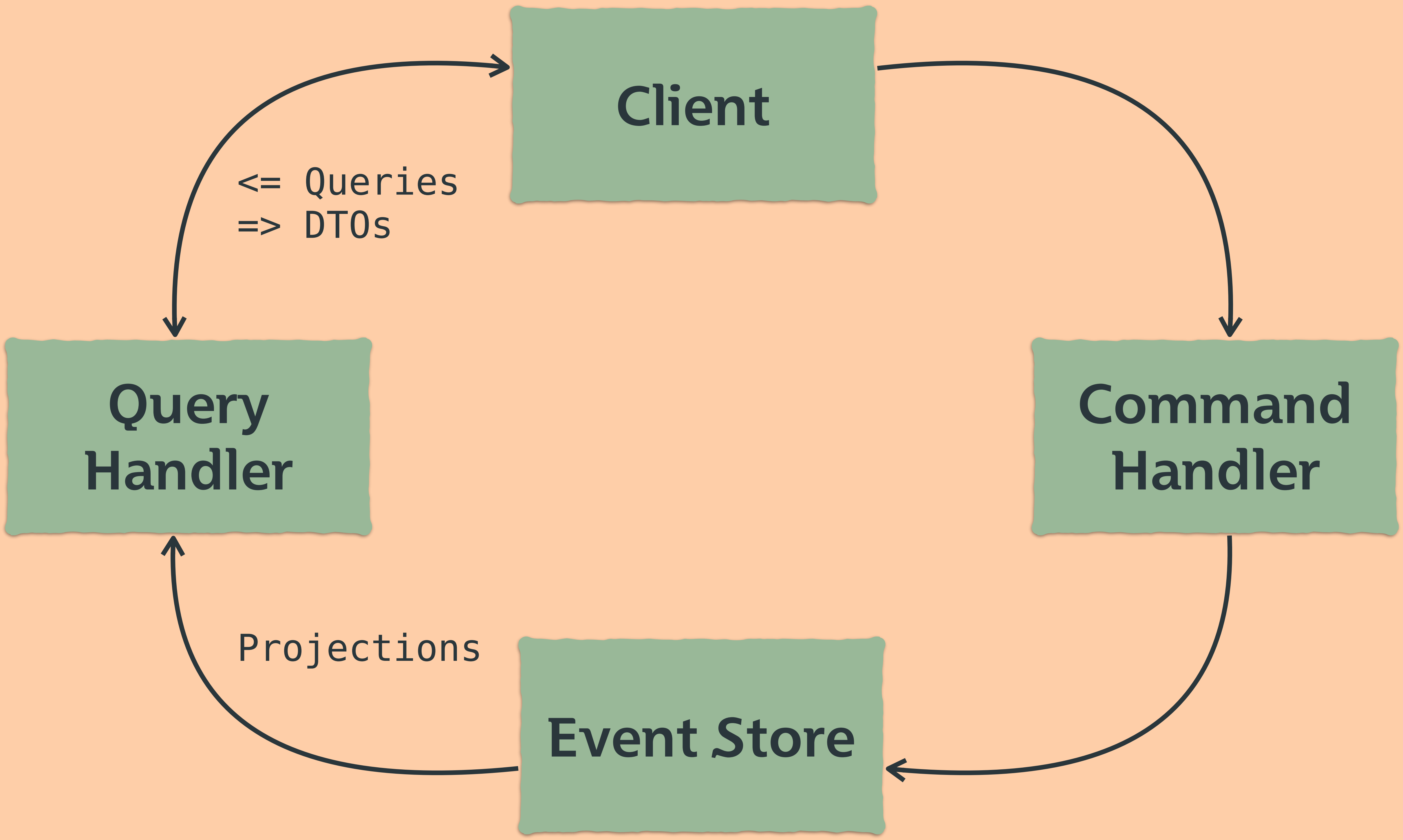

Client
Query Handler
Command Handler
Event Store


Client
Command
Command Handler
Event
Event Store
Query Handler

Commands represent user intent.

Commands can be rejected.

Events are created when commands are accepted.


Client
<= Queries
=> DTOs
Query Handler
Command Handler
Projections
Event Store

Projectors process Events in order.

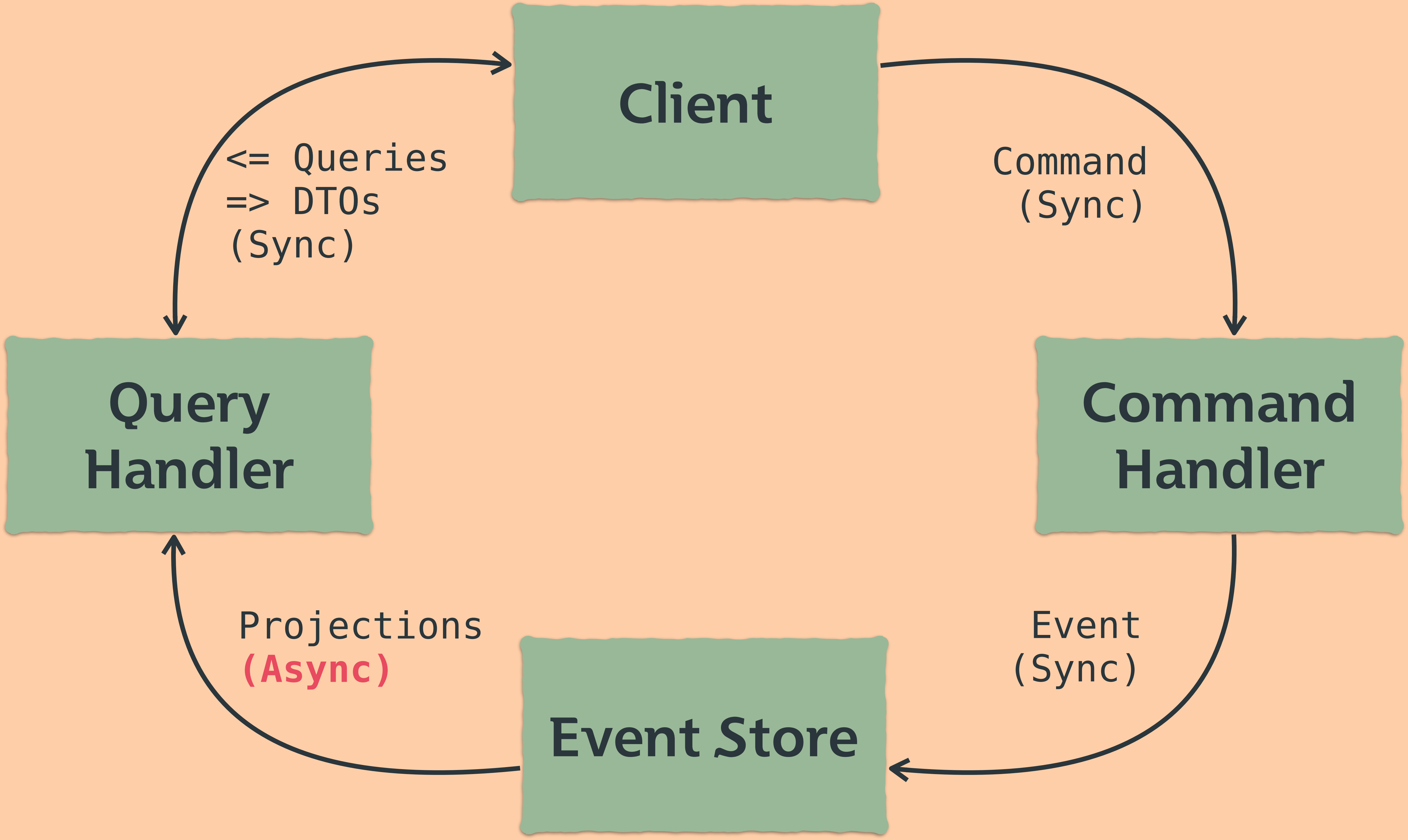
Projectors maintain denormalised Projections.

Projectors and Projections are 1:1.

One Projection per screen/endpoint.

Query Handlers query projections.

Query Handlers return DTOs to clients.


Client
<= Queries
=> DTOs
(Sync)
Command
(Sync)
Query Handler
Command Handler
Projections
(Async)
Event
(Sync)
Event Store

Eventual Consistency.

(That thing we’re not supposed to talk about.)

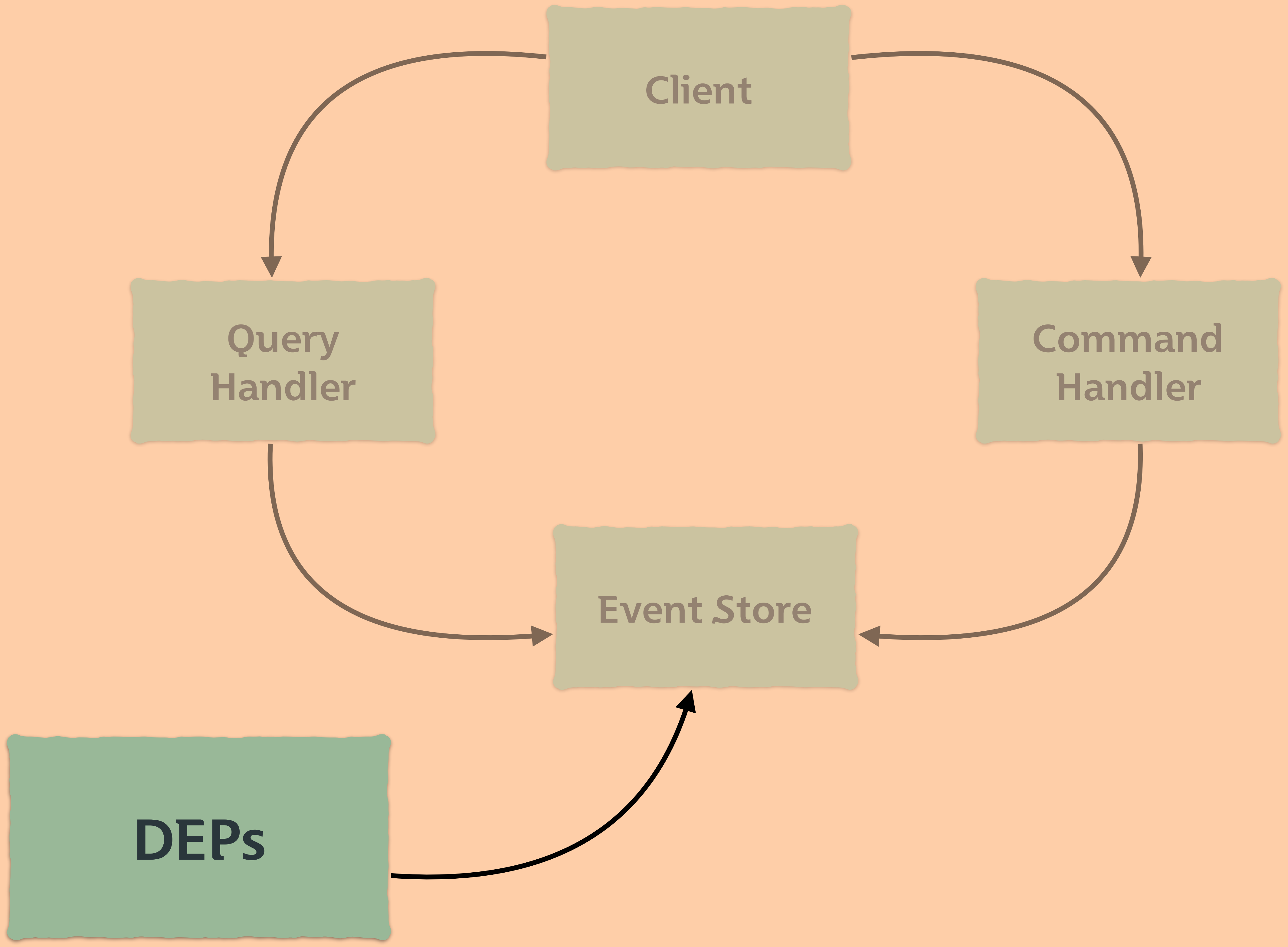
The world is eventually consistent.

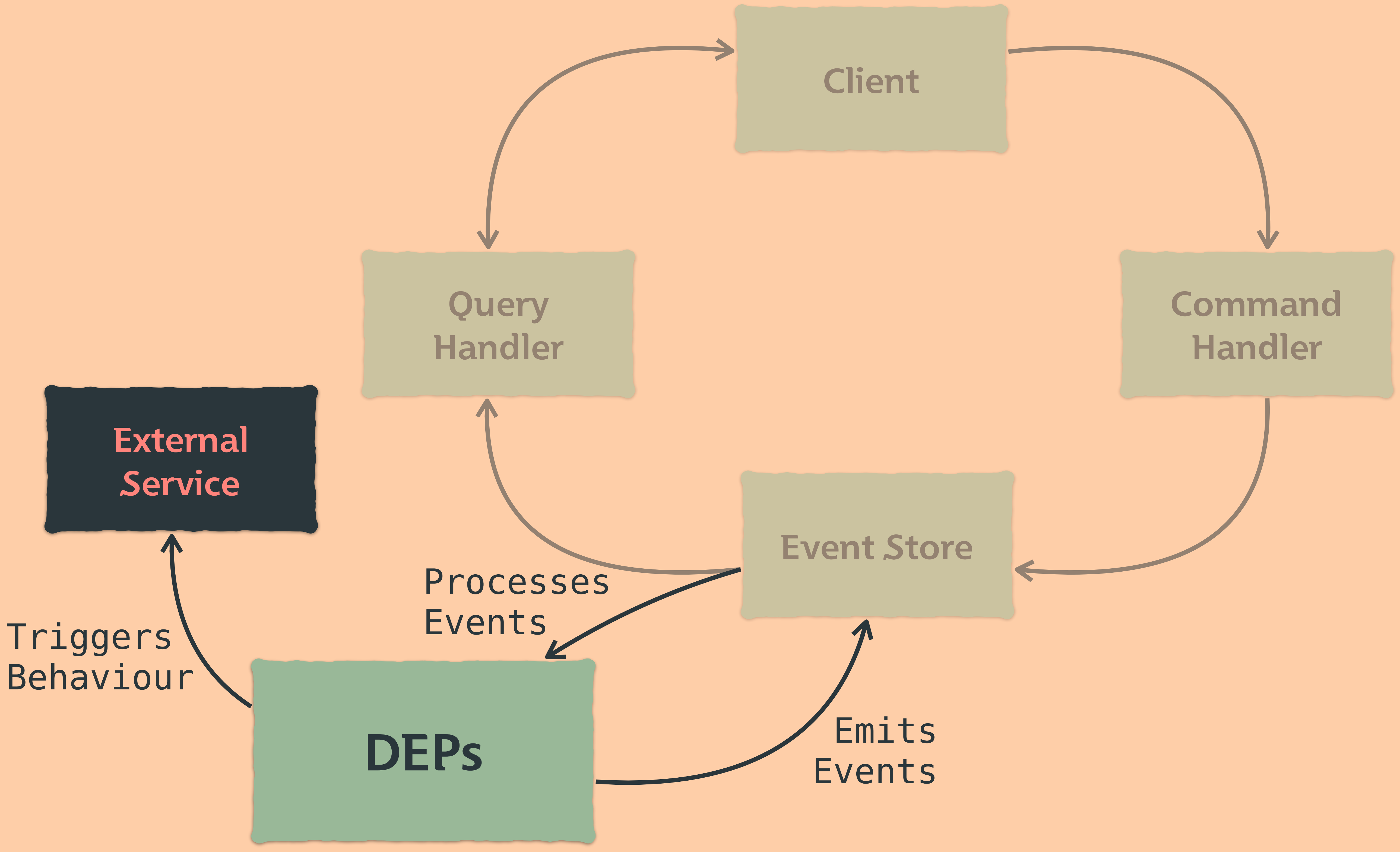
Is a nanosecond okay?

What about a month?

Risk is always a function of time.

Downstream Event Processors (aka DEPs).


Client
Query Handler
Command Handler
Event Store
DEPs



Sebastian von Conrad - @envato - @vonconrad

DEPs process events like projectors.

Can react by emitting events back to the Event stream.

Can react by triggering external behaviour.

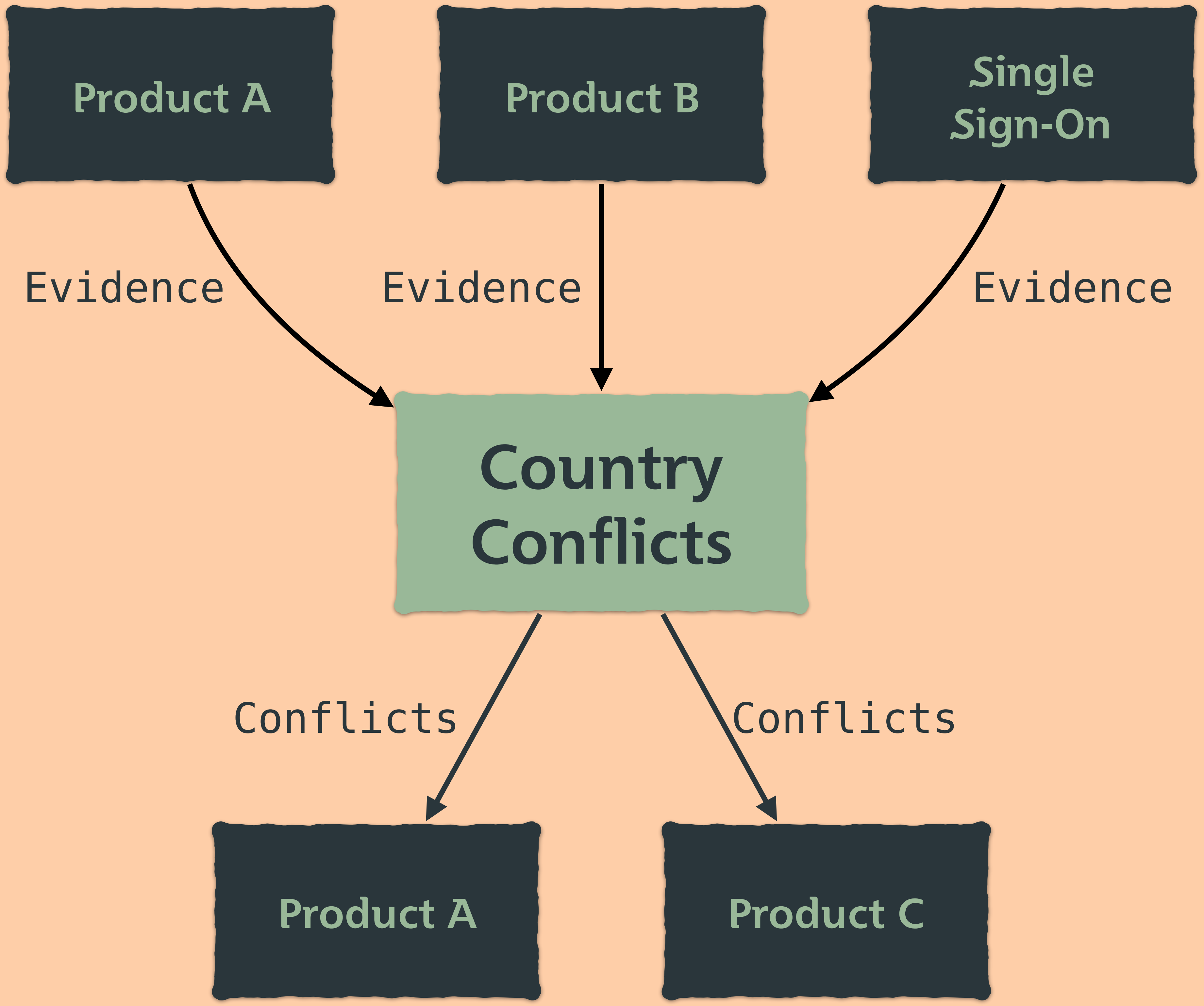
Encourages clean separation of concerns.

Case study.

Country Conflicts.

Business problem?

Detect and investigate conflicting information regarding a user's physical location.

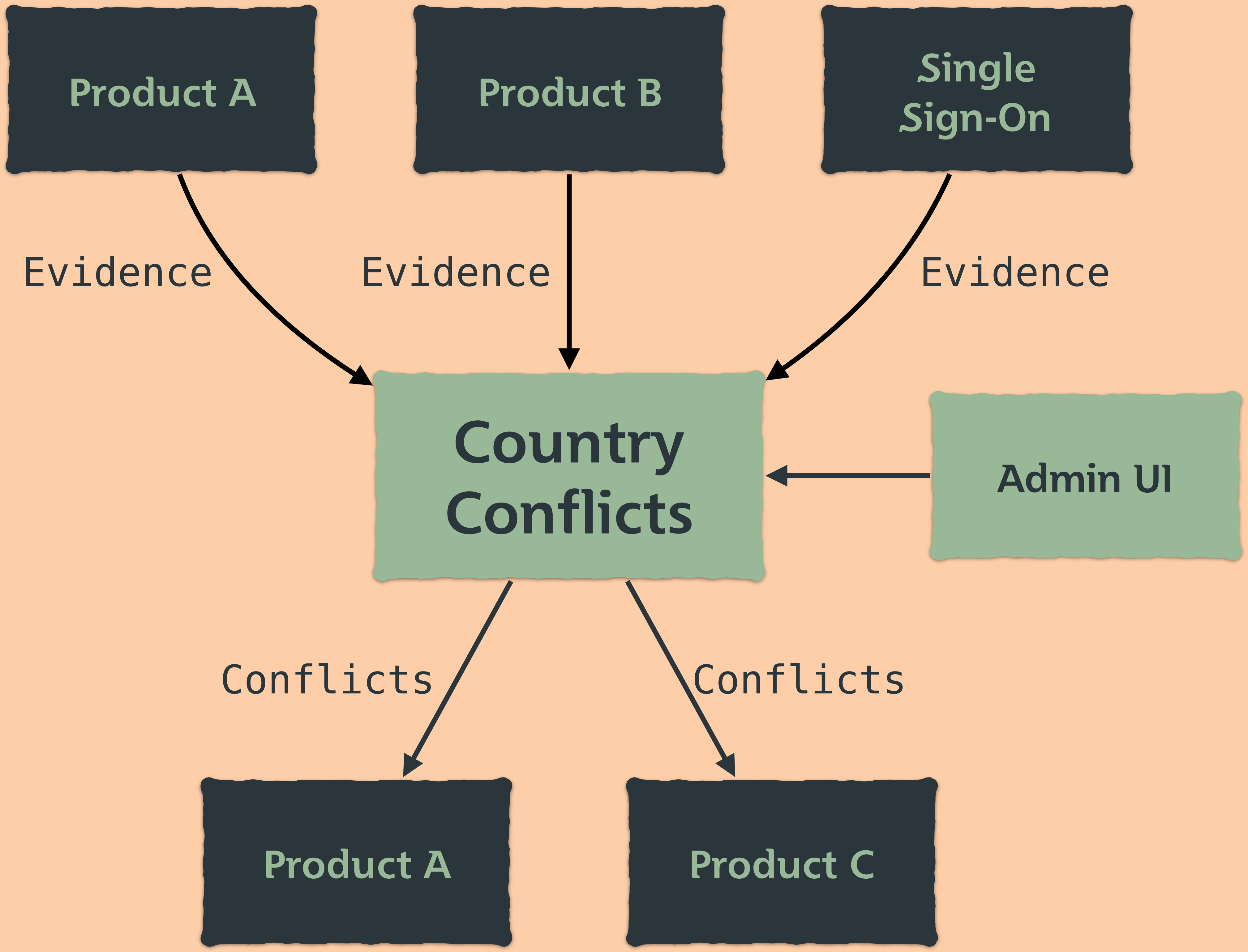

Product A
Product B
Single Sign-On
Evidence
Evidence
Evidence
Country Conflicts
Conflicts
Conflicts
Product A
Product C

Gather evidence and store as events.

DEP looks at evidence and raises ConflictDetected.

(Another) DEP sees conflict event and emails a notification.

(Another) DEP might auto-resolve the conflict based on more evidence.


Product A
Product B
Single Sign-On
Evidence
Evidence
Evidence
Country Conflicts
Admin UI
Conflicts
Conflicts
Product A
Product C

Admins issue Commands to manually resolve conflicts.

After 30 days, GracePeriodExpired is raised.

DEP checks whether it is. If not, raises ConflictActivated.

- Evidence Provided

- Conflict Detected
- Conflict Notification Email Sent
- Conflict Automatically Resolved
- Conflict Manually Resolved
- Conflict Grace Period Expired
- Conflict Activated

Other systems are querying conflicts through projections.

Small, well-defined, and simple.

Other systems are substantially larger.

So why this CQRS architecture?

Encourages Single Responsibilities.

Command and Query Handlers can scale independently.

Writes are fast.

Reads are faster.

Projections can be thrown away when no longer needed.

Separating recording from interpreting what happened.

Limit blast radius of changes.

Reduces fear and enables rapid change.

Keep the
cost of change
lower for longer.

The heart of a system is far more stable than the edges.

Should you use it?

Well, maybe.

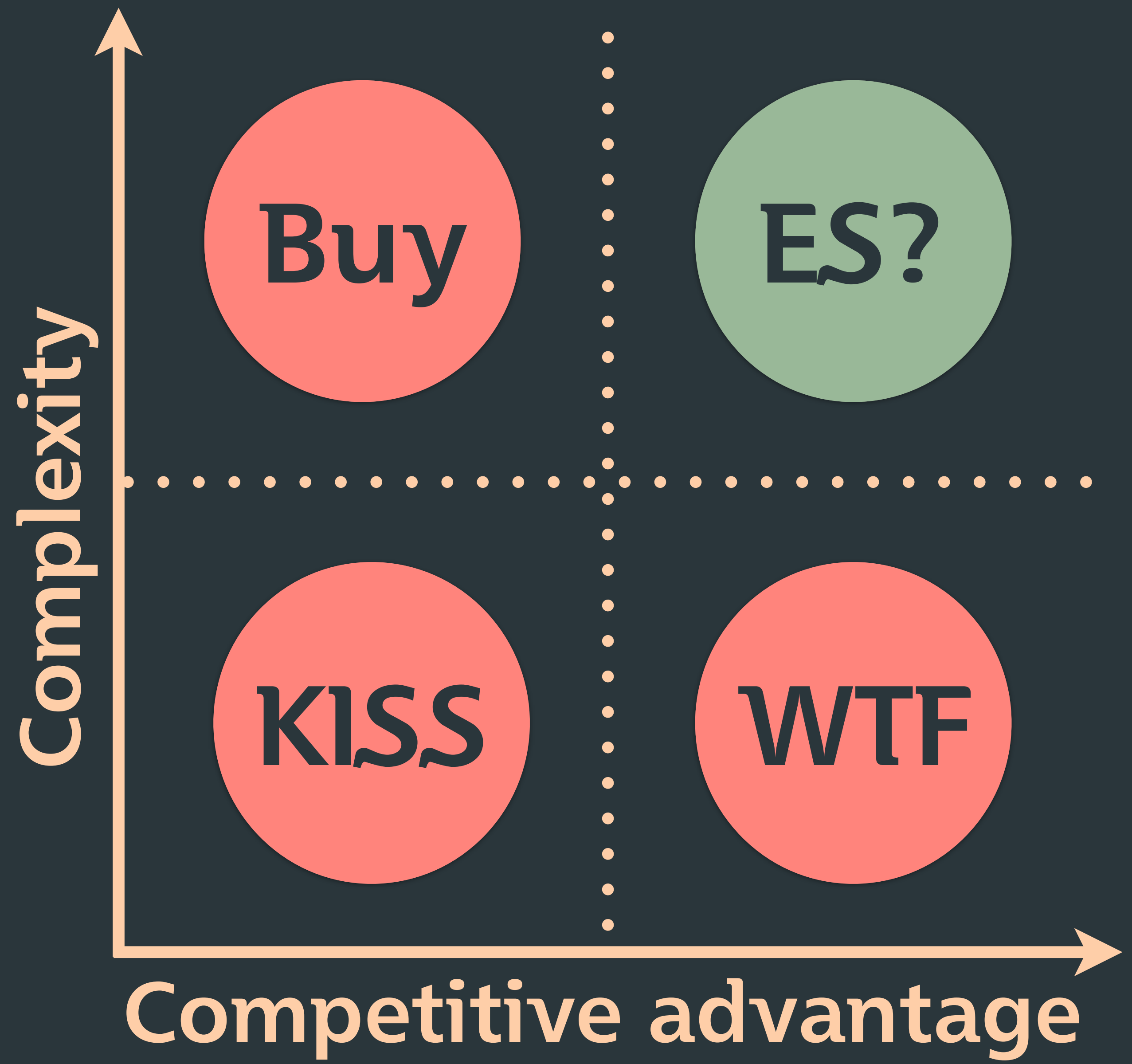
Not appropriate for **every problem.**

Works better for read-heavy systems.

Well suited for commerce-oriented domains, for example.

Or anywhere where history is important.

Strategic Design*



It’s a great tool to have in your toolbox.

Thank you.
Questions?

Sebastian von Conrad - @envato - @vonconrad