The CQRS/ES pattern

Domain-driven Design

Let’s start at the beginning

Bounded contexts

Large applications need to have multiple models at the same time. They are organised in bounded contexts, as in we define their boundaries in order to avoid mixing models together.

Aggregate

“Cluster of associated objects that we treat as unit”. In general, the aggregate is ***invariant***. At the *root* of a *bounded context* is a single aggregate.

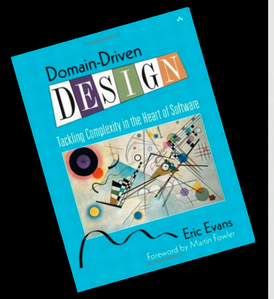
Entities & value objects

the building blocks of the aggregates. Entities are defined by their identity, value objects don’t.

Domain modelling tips

Buy Eric Evans' book

"Domain-Driven Design – Tackling complexity in the heart of software"

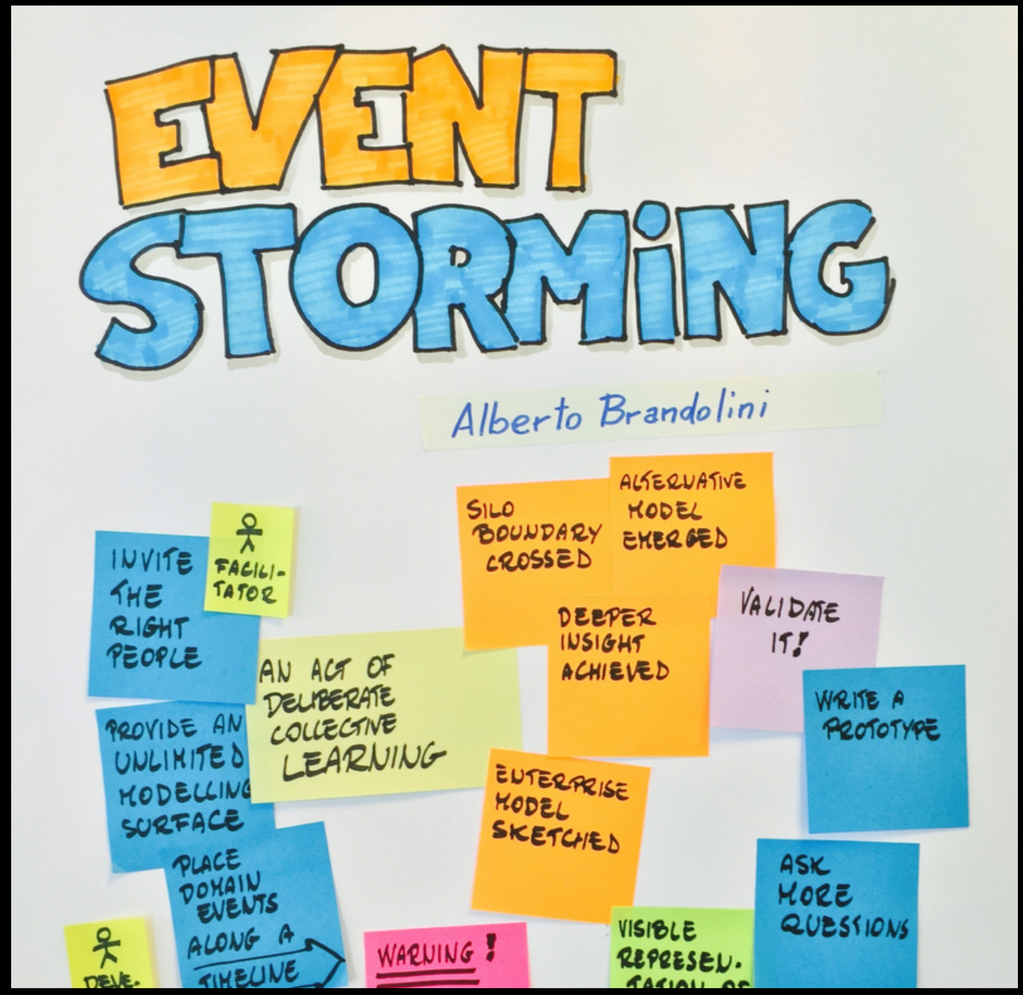


Involve EVERYONE

People with questions, People with answers. From IT, Marketing, Sales, Support, etc.

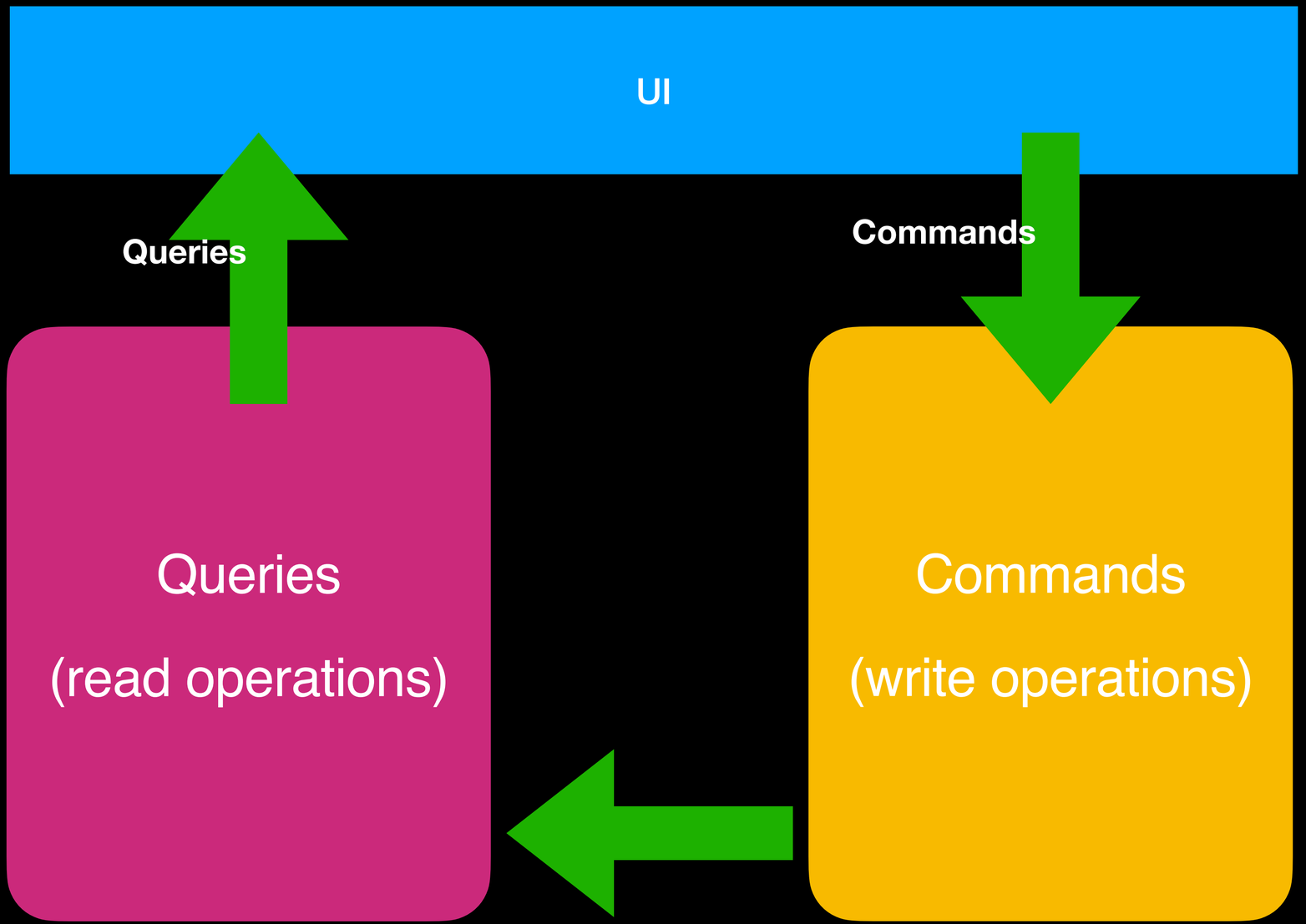
Event storming can be a powerful tool

Get post-its. Then get more post-its. Put relevant business events over a timeline, then isolate your commands, read models, and your domain model appears in front of your very eyes.



CQRS

A natural evolution of DDD


UI
Queries
Commands
Queries
(read operations)
Commands
(write operations)

Scalability

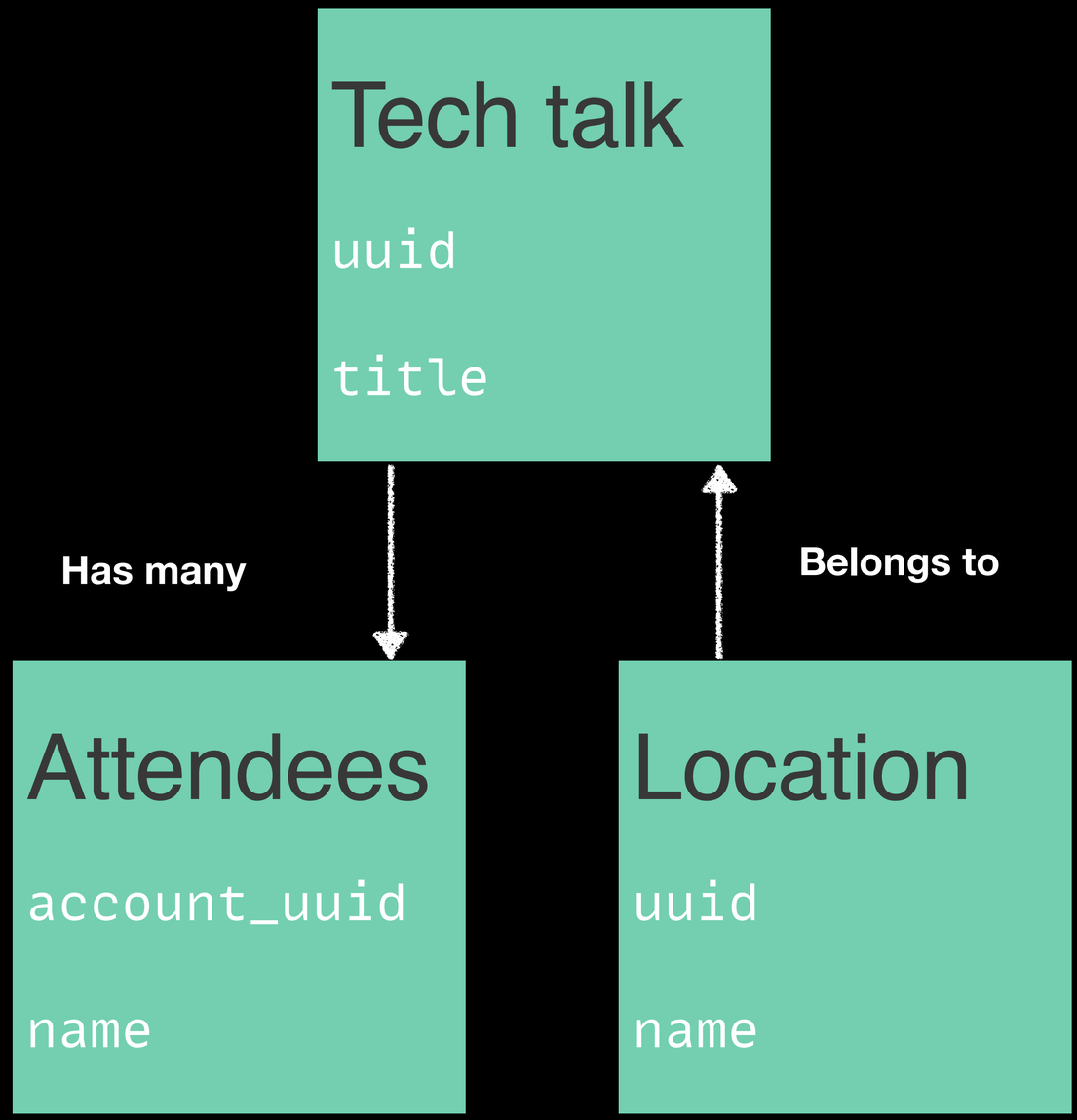
Scale the writes and the reads independently.

Flexibility

Write and re-write the read models based on your business needs

Event Sourcing

Not quite a new concept


Tech talk
uuid
title
Has many
Belongs to
Attendees
account_uuid
name
Location
uuid
name


Attendee Added
Talk title updated
Talk created

Event Sourcing ensures that all changes to application state are stored as a sequence of events

Martin Fowler - https://www.martinfowler.com/eaaDev/EventSourcing.html

Immutable

Thy shall never delete thing from the event store. It is append-only

Single source of truth

$$current\_state = \sum events$$

Be careful with snapshots

Fine-tune your snapshotting logic (race conditions)

Time travel

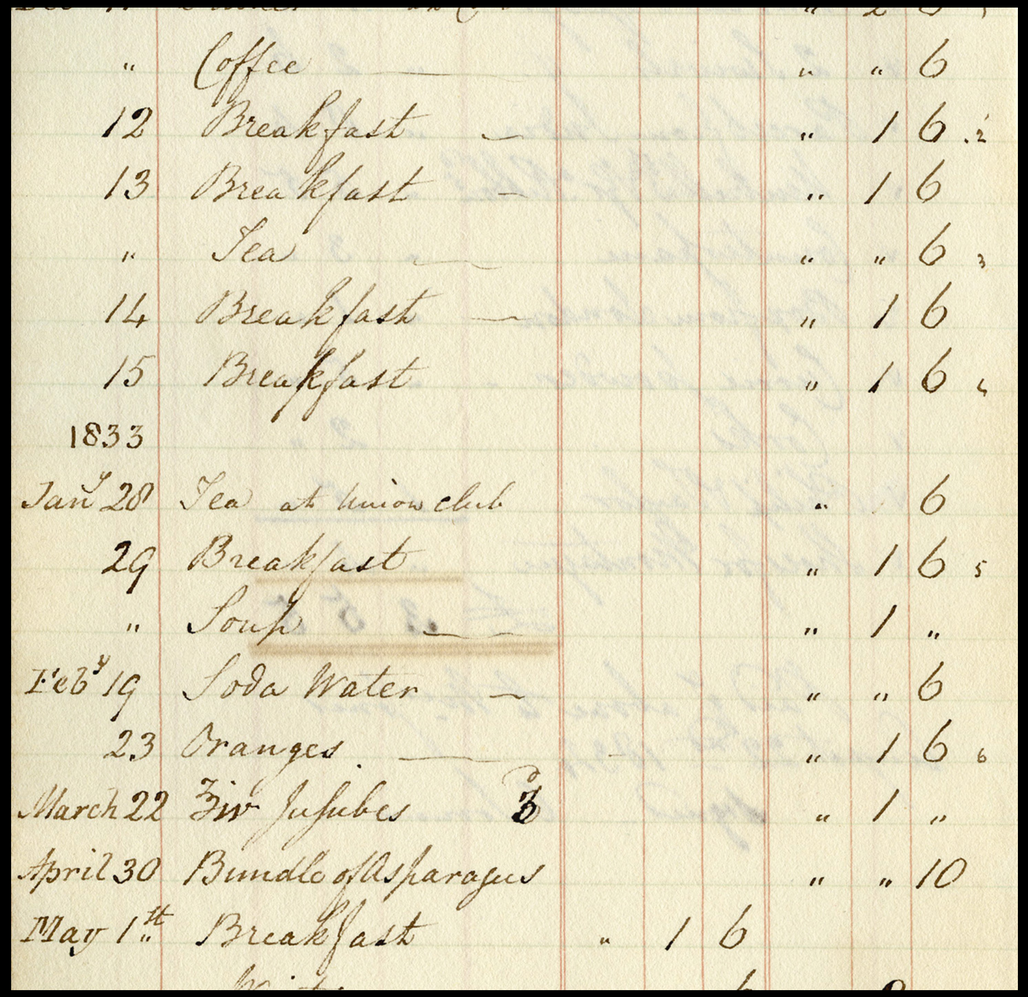
Project data from the event store to any point in time in the history of the system. Business will love you for it

Audit log

The event store is an immutable, append-only datastore

CQRS/ES: when is it a good idea?

- Reads > Writes, Writes > Reads
- When the amount of reads you need to perform is far greater than the amount of writes. Or vice-versa.
- When you have a complex domain to model
- It’s easier to reason about a series of events when modelling a complex domain (insurance policy changes, e-commerce, banking transactions…)
- When your system can’t afford to lose data
- Your industry is heavily regulated, or you’re dealing with critical data for your customers



Sources:

eventstore.org

martinfowler.com

eventstorming.com

Thank you!