

Event sourcing strikes back

Building resilient, compliant backends
with immutable, append-only data structures

Dr. Bertrand Caron
Chief Technology Officer and Co-founder, Bloom Impact Investing



Show Me The Code (Brisbane)

v3.0.0

May the 4th, 2023

About me

- CTO of Bloom Impact Investing
 - Mobile app to invest to fight climate change (from \$100)
- Full-stack software engineer (8+ years)
- 5+ years building back-ends for high-compliance industries
 - Geospatial surveying (Fugro)
 - FinTech (Bloom)

BLOOM



Using mutable data models

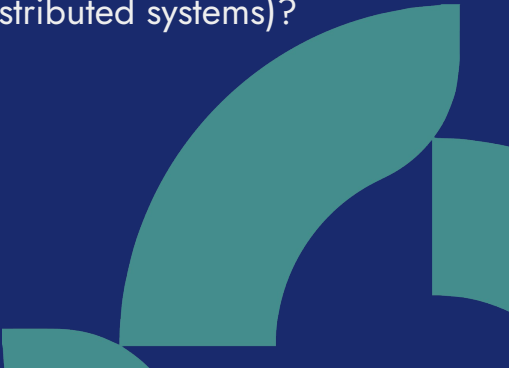
- Product Owner: "We need to store users in a database."
- Dev team: "Sure thing!"

```
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL UNIQUE,  
  password_hash VARCHAR(255) NOT NULL,  
  created_at TIMESTAMP DEFAULT NOW()  
);
```

SIX MONTHS LATER ...

- Product Owner: "I need a list of all users that have changed their name in the past six months."
- Dev team: "..."

Limitations of mutable data models

- Need to know upfront what data and access pattern you might need
 - Which fields are mutable?
 - Which fields need to be auditable?
 - Chasing (state) bugs gets very difficult
 - What state was the data in when it happened?
 - How did we get into that state (“How did we get there”?)
 - How to describe the state of the data at a given time (for instance, in distributed systems)?
 - Fundamentally, a lot of data is thrown away (i.e. mutated)!
 - You might only find that out when you most need it!
- 

Solution: using immutable, append-only data structures

“Event Sourcing is a pattern for storing data as events in an append-only log.”

“Every change is represented as an event, and appended to the event log.”

“An entity’s current state can be created by replaying all events in order of occurrence.”

eventstore.com

Time



Create User


Set Name

Set Password

Set Name

Set Password

Event sourcing vernacular

- **Entity:** An object of the domain e.g. a user
 - **Event:** A fact that took place in the domain
 - **Event stream:** Ordered set of events for a given entity (domain object)
 - **Projection:** A view on an underlying event-based data model
- 

Advantages

- **Observability**
 - Anything can be measured at any time
 - New metrics (projections) can be applied to old data!
- **Time-travel**
 - An entity can be represented at any time, by folding the event stream up to that point
 - Helps tremendously with debugging!
- **Subscriptions and event-driven architectures**
 - Entities are versioned by default (e.g. by their last event)!
 - Systems/services can subscribe to entity updates (push vs pull model)




Live(ish) demo time!

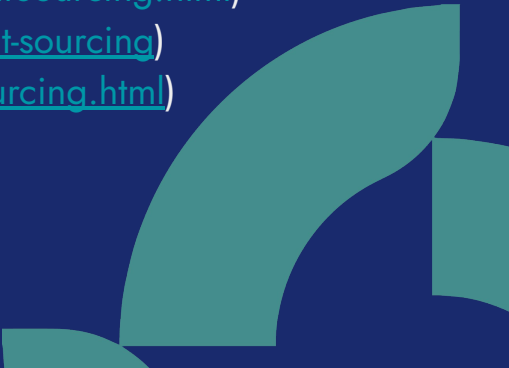
- Let's build a serverless REST API (in Node/TypeScript) using event sourcing!
- Follow along: <https://github.com/bertrand-caron/event-sourcing-serverless-demo>



Challenges of event sourcing

- Adds non-zero complexity
 - Requires more storage and compute than mutable data models
 - Migrations are hard and discouraged i.e. you need to get it right the first time
 - Caching projected entities can be challenging
- 

Want to know more?

- Look at dedicated event sourcing database / adaptors:
 - Datomic (free since May 2023)
 - Patchyderm (for data science)
 - EventStoreDB (free tier)
 - Plenty of good articles, books and tutorials on event sourcing
 - Event sourcing by Martin Fowler (<https://martinfowler.com/eaDev/EventSourcing.html>)
 - A beginner's guide to event sourcing (<https://www.eventstore.com/event-sourcing>)
 - Pattern: Event sourcing (<https://microservices.io/patterns/data/event-sourcing.html>)
- 

Thanks for listening!

Comments?

Questions?

