

**Ted M. Young**

Java Trainer, Coach, Live Coder, and Creator of JitterTed's TDD Game



# Stop Fighting Your ORM

Simpler, More Flexible Systems with Event-Sourcing

Me: <https://ted.dev/about>

Source Code? Slides? Recording?

<https://ted.dev/talks>



**Ted M. Young**  
**ted@tedmyoung.com**

**I Can Help Your Team...**

Write more Testable code  
with more Effective tests

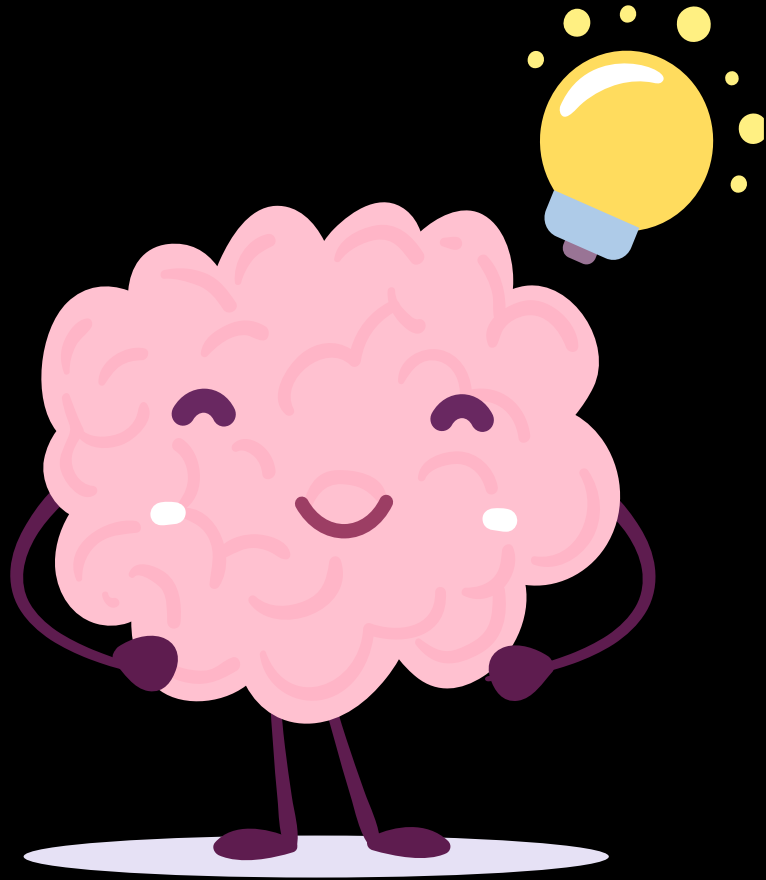
Be more productive in  
**Java & Spring**

Effectively use  
**TDD**

**Refactor  
Messy  
Code**



# Ask Questions As You Need



I may defer the  
answer if I'm going  
to cover it later!



# Event Sourcing Frameworks/Libraries

**.NET:**  
**Marten**

**PHP:**  
**EventSauce**  
**Ecotone**

**Java:**  
**AxonIQ**



# What's in this TED Talk...

1. How (and why) do we persist data?
2. Define terms, clear up misconceptions
3. JitterTicket: Concert Ticketing domain
4. Dig into code (warning: generics!) & tests
5. Related topics to explore on your own:
  - CQRS, Event Modeling
  - Versioning & Schema Migration
  - Performance & Snapshotting
  - Dynamic Consistency Boundary & Decider & GDPR



# What is **APPLICATION STATE?**

Outcome of DOMAIN DECISIONS made over TIME

# JitterTicket Domain

CUSTOMER

CONCERT

TICKET



# Domain Decisions

- **Purchase Ticket**

- **Decision (Domain Rule): enough tickets?**
  - Yes -> State Change: Ticket Purchased
  - No -> No State Change

- **Reschedule Concert**

- **Rules: not canceled; in the future**
  - Yes -> State Change: Concert Rescheduled
  - No -> No State Change (report why)



# How to Persist Decisions?

Often: Map Objects to Database Tables (ORM)

# Mapping Objects from/to Database



r/ExperiencedDevs  
u/Fuzzy\_World427 · 10h

## DDD: How do you map DTOs when entities have private setters?

Hey all,

I'm running into trouble mapping DTOs into aggregates. My entities all have **private setters** (to protect invariants), but this makes mapping tricky.

I've seen different approaches:

- Passing the whole DTO into the aggregate root constructor (but then the domain knows about DTOs).
- Using mapper/extension classes (cleaner, but can't touch private setters).
- Factory methods (same issue).
- Even AutoMapper struggles with private setters without ugly hacks.

So how do you usually handle mapping DTOs to aggregates when private setters are involved?

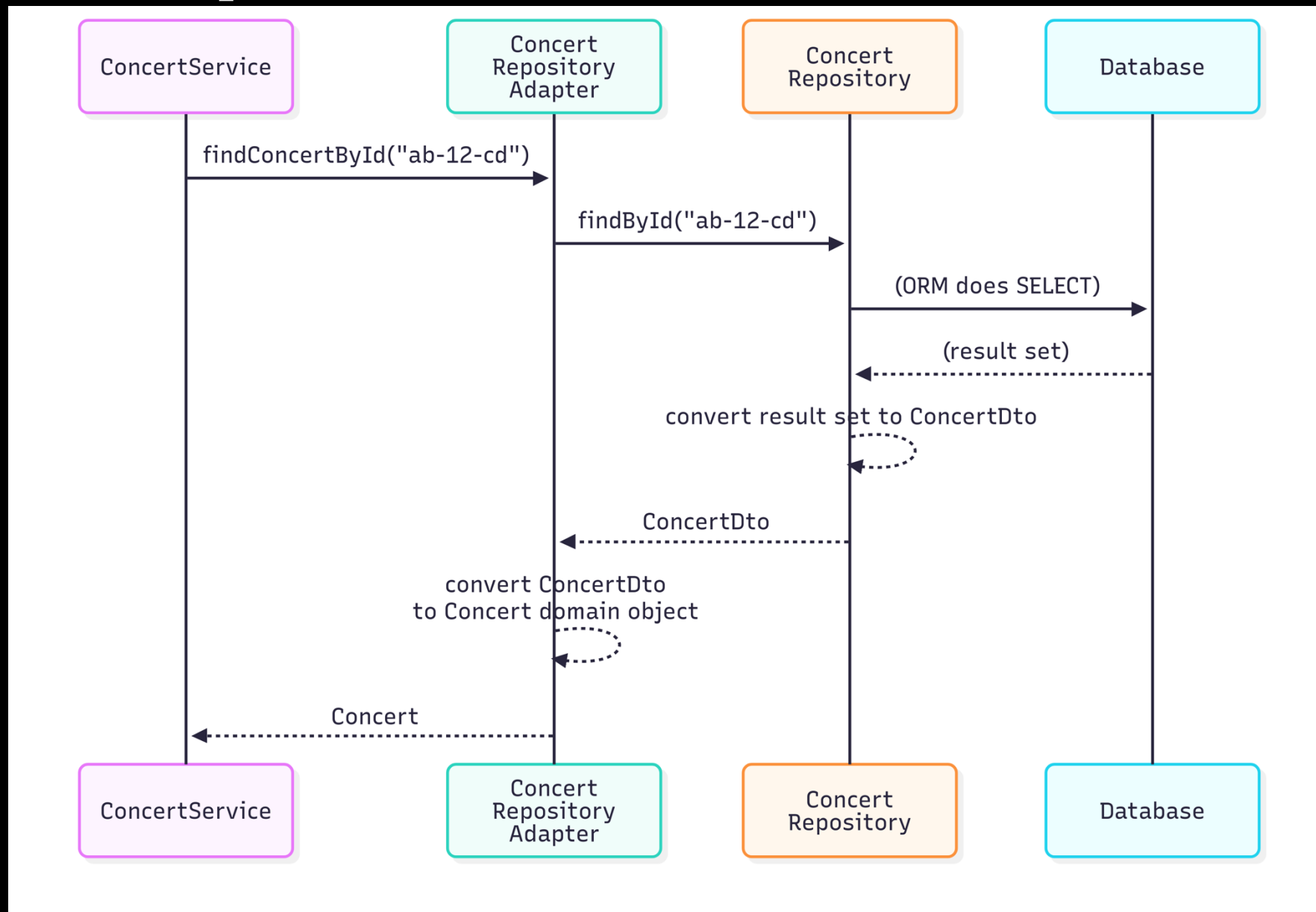
↑ 24



○ 51

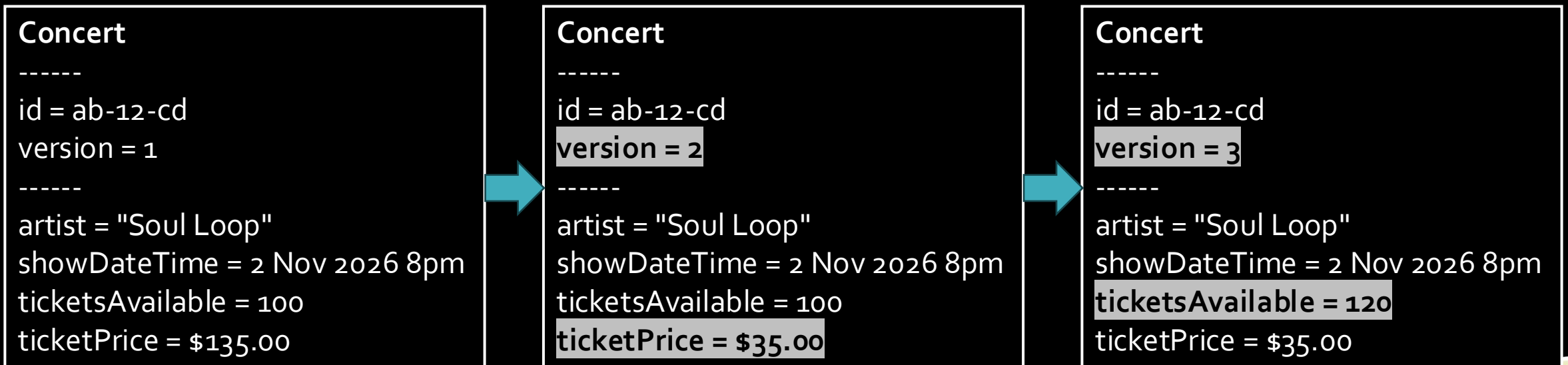


# Adapter-Driven Persistence



# Storing Data as "Latest State"

- "State-Sourced"
- No history, throws away old information
- No knowledge of how/why state changed



# Storing Data as Changes in State

Concert Scheduled: ab-12-cd, "Soul Loop", 2 Nov 2026, 20:00, 100, 135.00

Ticket Price Changed: ab-12-cd, 35.00

Capacity Increased: ab-12-cd, 120



## Concert

-----

id = ab-12-cd

version = 3

-----

artist = "Soul Loop"

showDateTime = 2 Nov 2026 8pm

ticketsAvailable = 120

ticketPrice = \$35.00



# Fundamentals

**Past**

Event

**Present**

State

**Future**

Command



# Definitions: EVENT (the past)

*a fact, something that happened*

Customer Registered

Concert Scheduled

Ticket Price Changed

Ticket Purchased



# Definition: STATE (now)

Context needed by the application to *DECIDE*

## Concert

-----

id = ab-12-cd

version = 1

-----

artist = "Soul Loop"

showDateTime = 2 Nov 2025 8pm

ticketsAvailable = 100

ticketPrice = \$135.00



# Definition: COMMAND (future)

*Request to change state*

Register New Customer

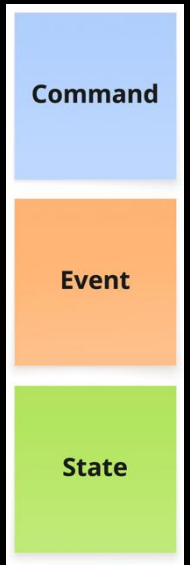
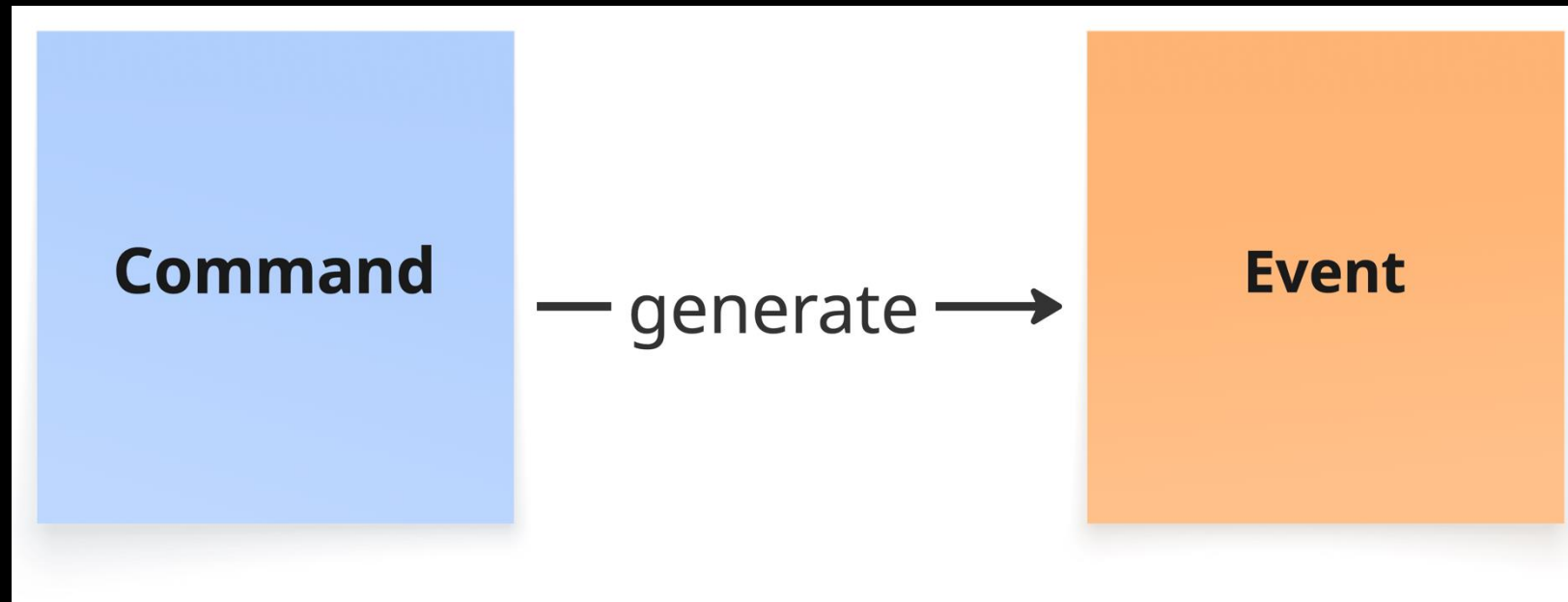
Schedule Concert

Change Ticket Price

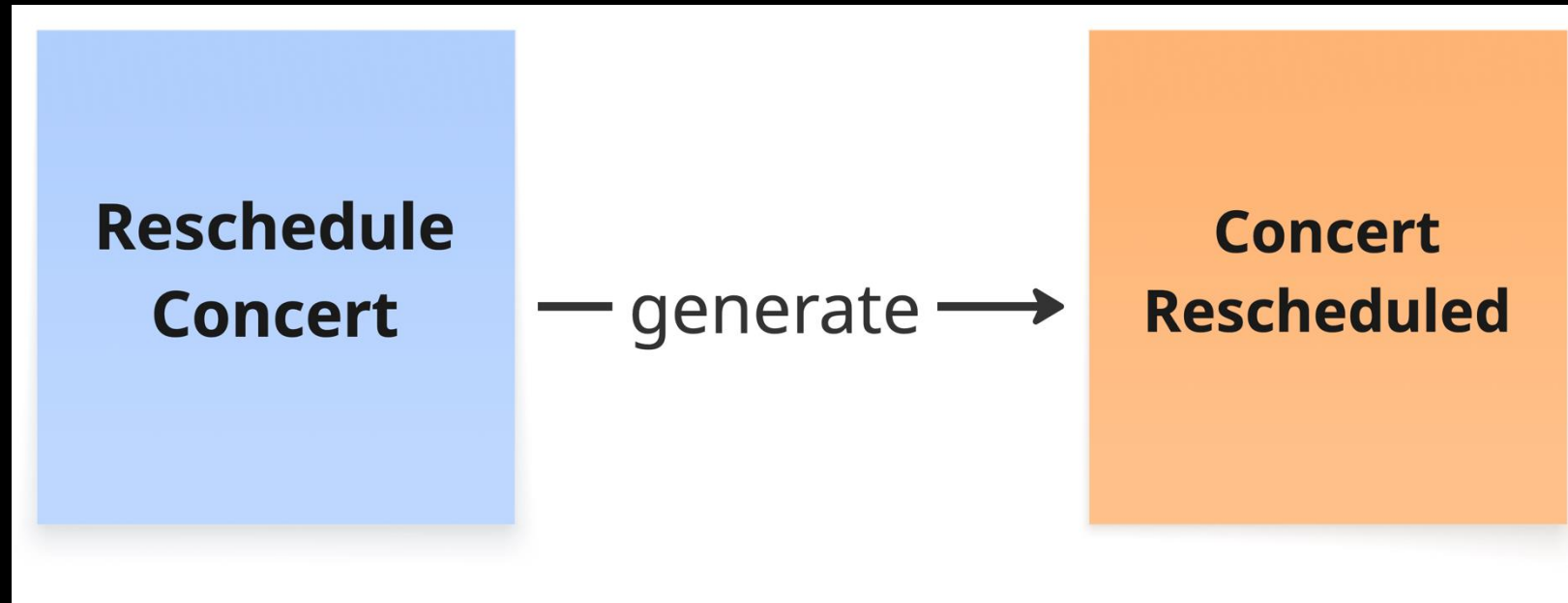
Purchase Tickets



# Commands Generate Events



# Commands Generate Events



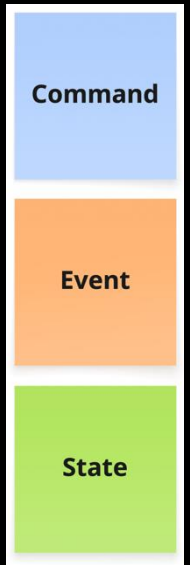
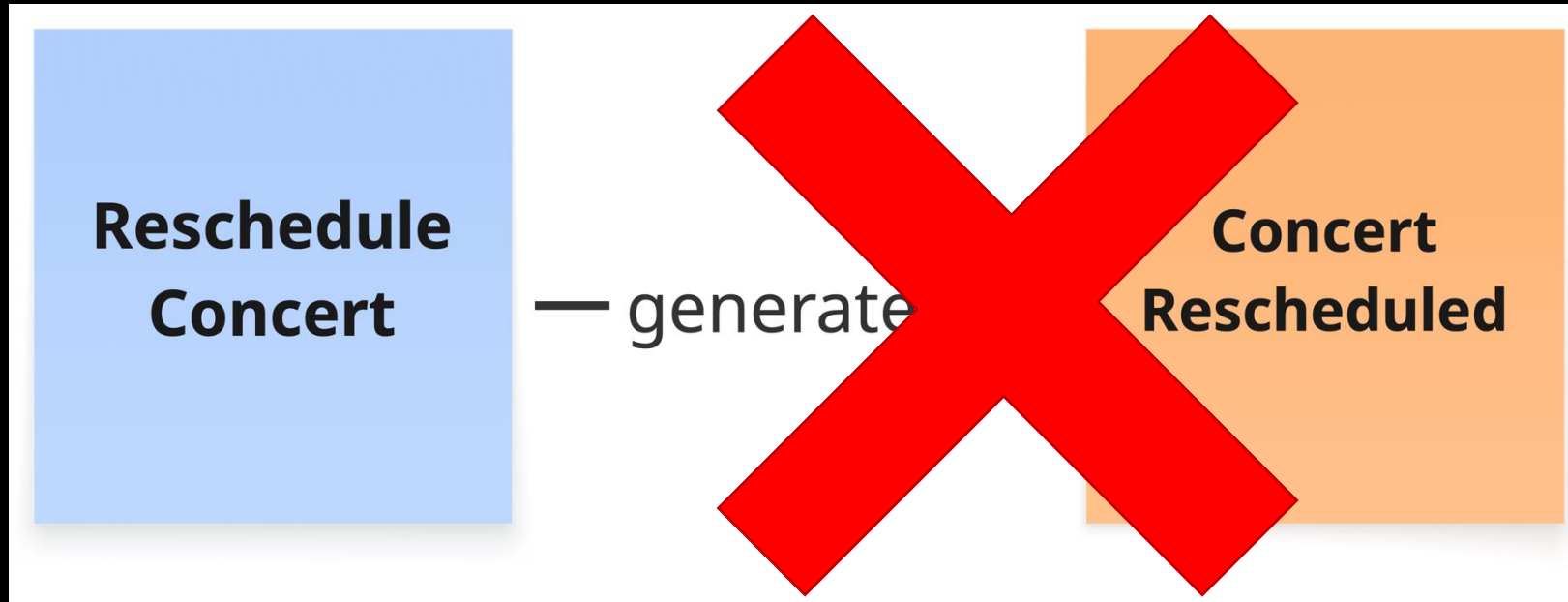
Command

Event

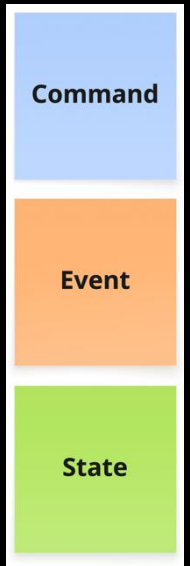
State



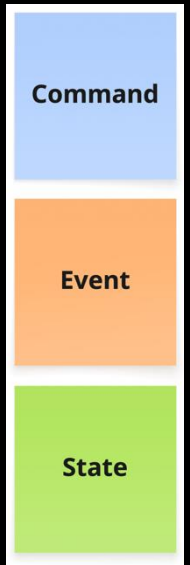
# ...unless It's Not Allowed (Rule)



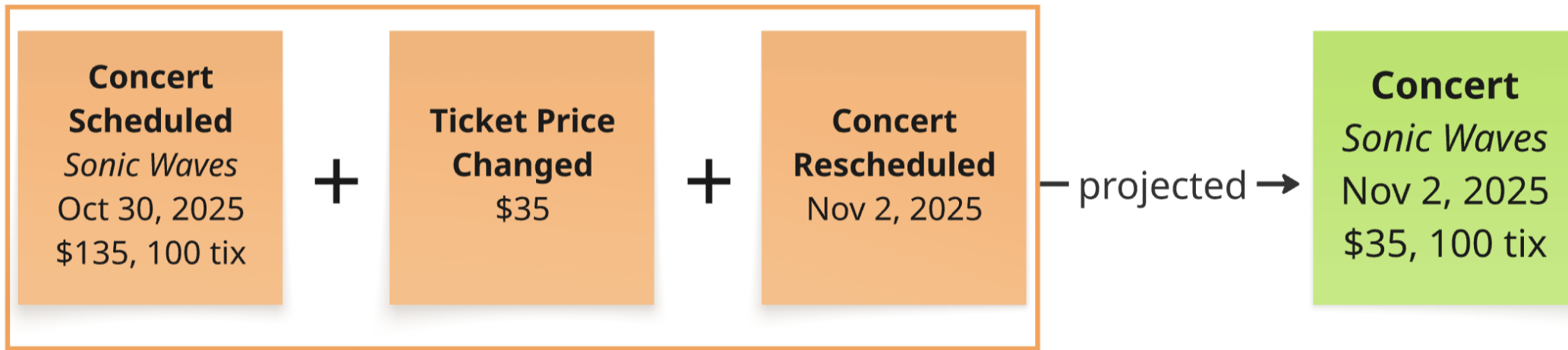
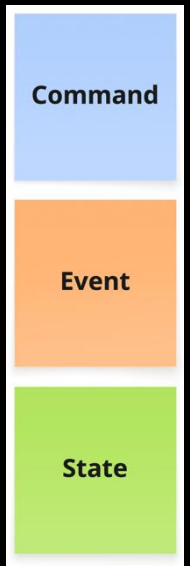
# Events Update State



# Events Update State (always!)



# Events *Project to State*



# Event-Sourcing PROJECTOR

A function that constructs a *data model* by sequentially *replaying events* from the *event store*.

Produces a PROJECTION



# Projections Can Provide

- Current **STATE** (context) to **DECIDE**
- Customized Views for UI
- Database tables for integration

**One Concept: Many Uses**



# Event Store

An append-only "database" that records all events generated by the application.



# Benefits of Event-Sourcing

# Ask Questions You Didn't Know You Had

How many customers transfer tickets?

How often are we rescheduling concerts?

# Immutable Events

Never lose information: there is no DELETE

# Fewer Concepts

Commands, Events, Projections

# Less complexity than ORMs

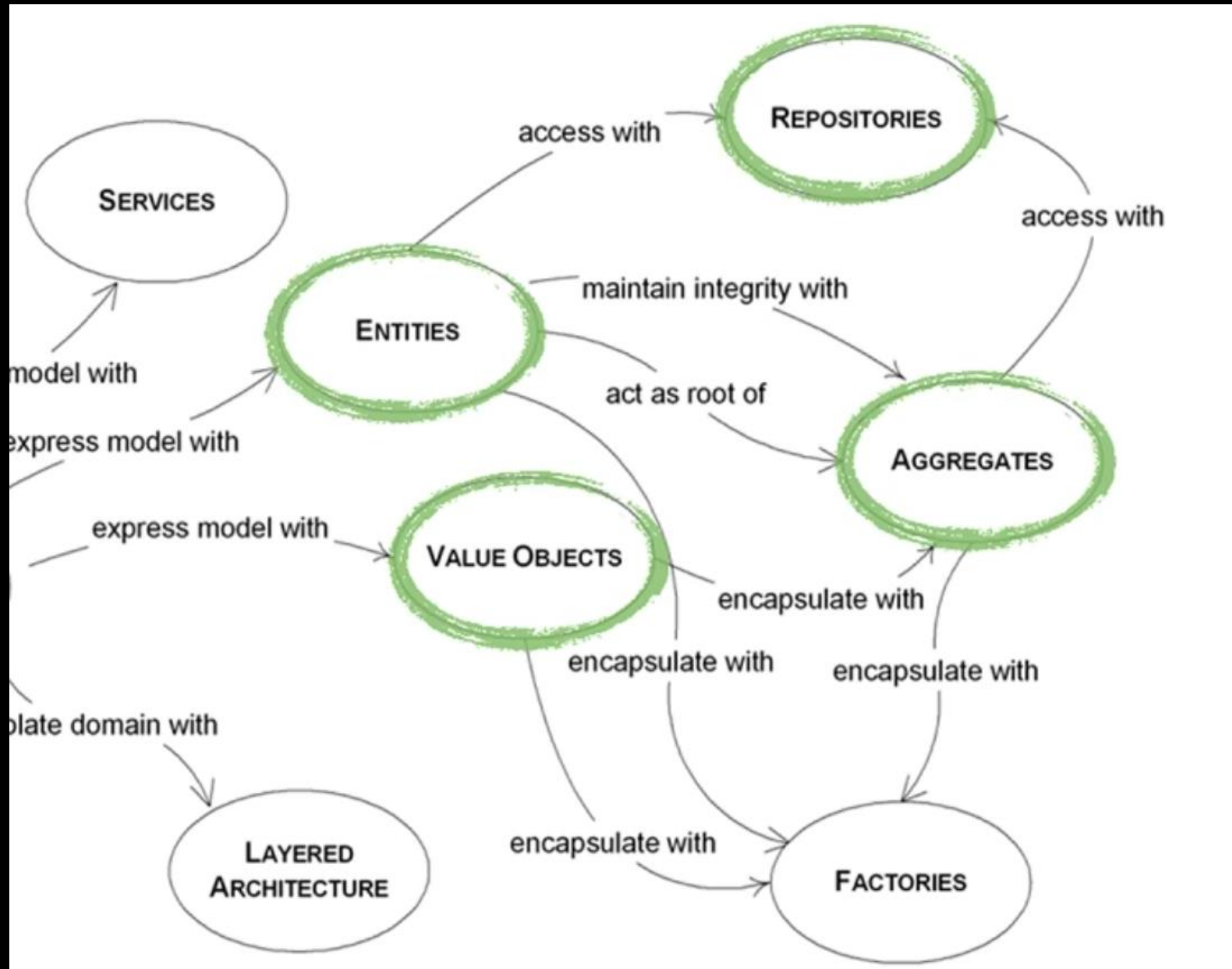
No awkward mapping, lazy loading, or dirty-checking

# Replayability

Time Travel: see state of system in the past

# Domain-Driven Design

Tactical Patterns



# Entity

Unique (Has Identity), History, and Attributes

# Value Object

Identified Only by its Attributes (immutable)

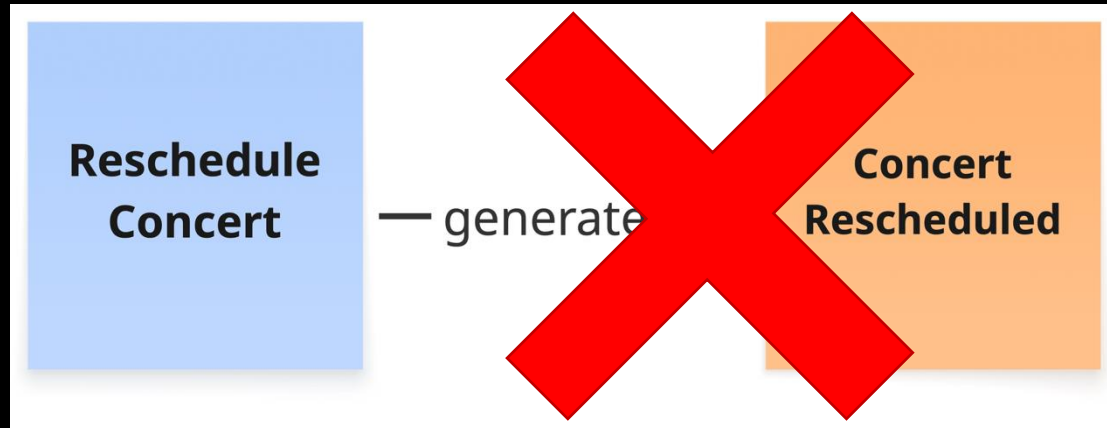
Quantify, describe, and measure

# Aggregate

Consistency [transactional] Boundary  
(Enforced by "Aggregate Root" Entity)

# Repository

Used for Storing and Retrieving Aggregates



# Aggregate & Projections

Aggregates in Event-Sourcing combines:

***PROJECTION*** (state) + ***DECIDER*** (rule-applier)



**Repository**

Used for Storing and Retrieving Aggregates

# Misconceptions

# Event-Driven Architecture

Often Integrating across Context Boundaries

# Event Streaming

High Volume and/or Rate of Events

# CQRS

Command-Query Responsibility Segregation

*Does Not Require Event-Sourcing!*

# JitterTicket Concert Ticketing

The Event-Sourced Ticketing System

Explore Concert Tickets

Event Viewer

Concert Sales



# JitterTicket Flows

tldraw

JitterTicket: Concert Ticketing System

# Diving Into the Code

# Let's talk about Performance

Similar, if not BETTER than ORM

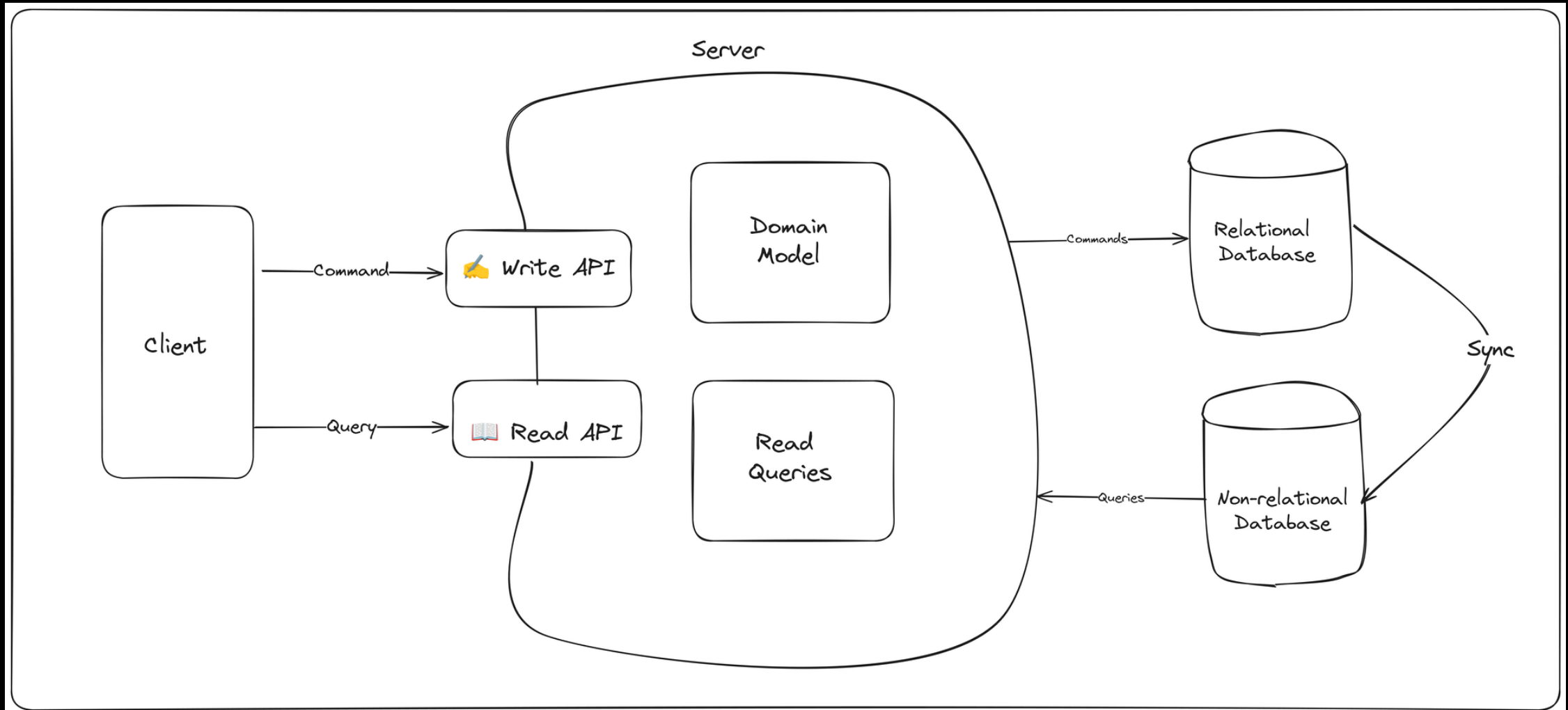
# Versioning Event Logs (Streams)

(see Greg Young's Versioning in an Event-Sourced System)

- **Copy-Replace**
- **Split-Stream**
- **Join-Stream**
- **Copy-Transform**



# CQRS – Separated Data Models

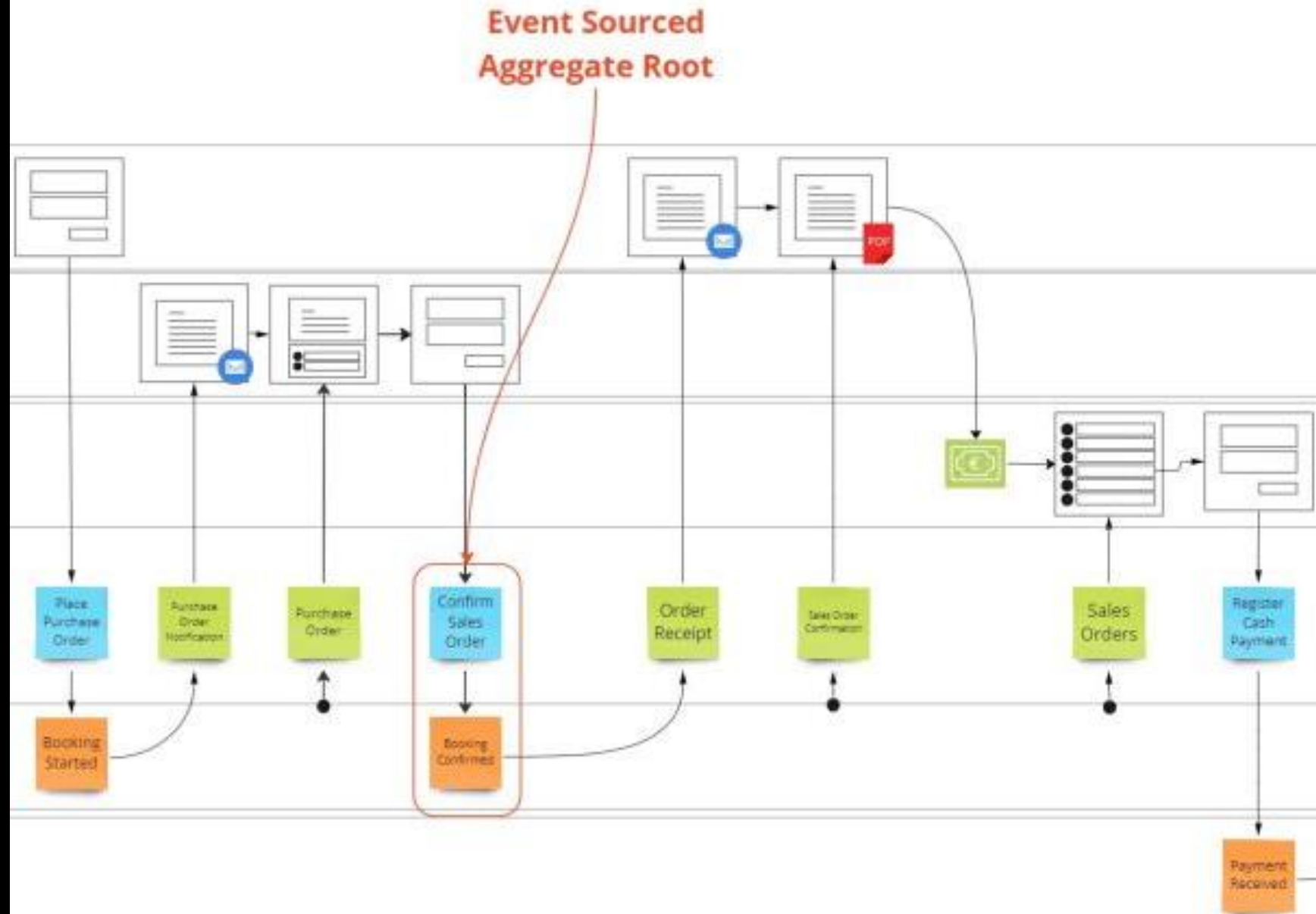


# Event Modeling

Command

Event

State



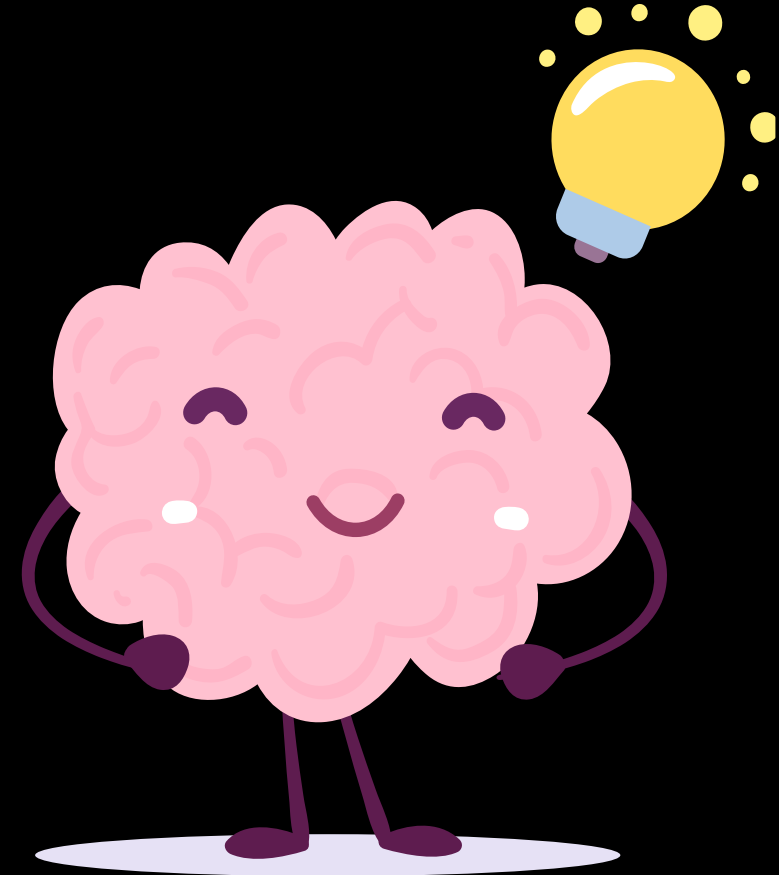
**Ted M. Young**

Java Trainer, Coach, Live Coder,  
and Creator of JitterTed's TDD Game

Get in touch: [ted@tedmyoung.com](mailto:ted@tedmyoung.com)

About me: <https://ted.dev/about>

# What other questions do you have?



**Ted M. Young**

Java Trainer, Coach, Live Coder,  
and Creator of JitterTed's TDD Game

Get in touch: [ted@tedmyoung.com](mailto:ted@tedmyoung.com)

About me: <https://ted.dev/about>

**Thank You...**

Source Code? Slides?

<https://ted.dev/talks/>

