Ted M. Young

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

Event Sourcing

The End of DDD Tactical Patterns?

Me: https://**ted.dev**/about

Source Code? Slides?

https://ted.dev/talks

Ted M. Young

ted@tedmyoung.com

I Can Help Your Team…

Write more Testable code
with more Effective tests

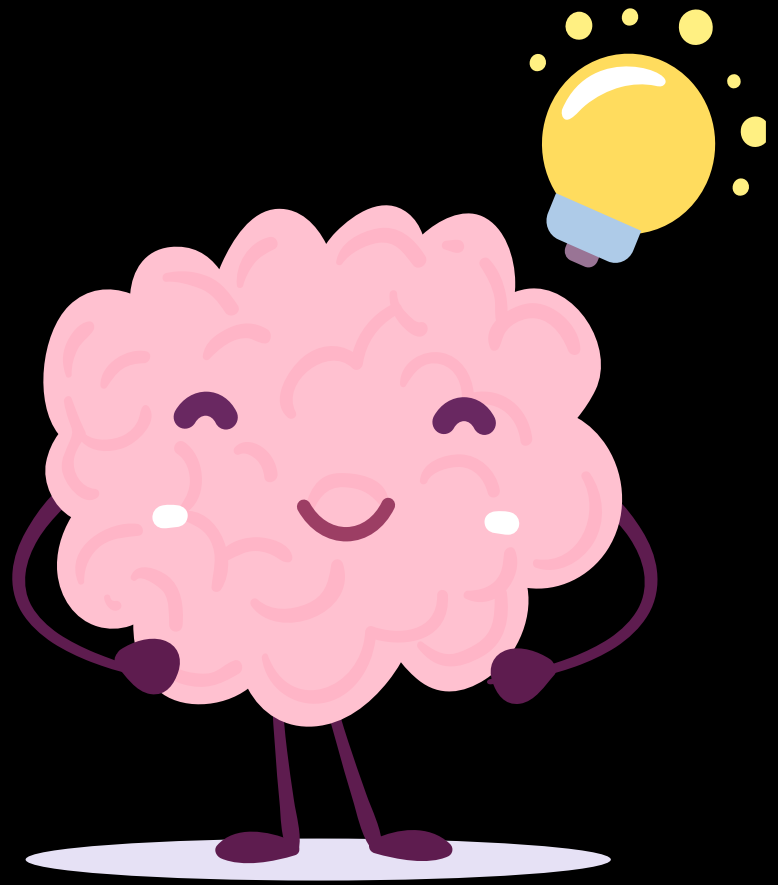
Be more productive in
Java & Spring

Effectively use
TDD

Refactor Messy Code

https://ted.dev/about

Ask Questions As You Need



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

What's in this TED Talk…

1. How (and why) do we persist data?
2. JitterTicket: Concert Ticketing domain
3. Define terms, where to make decisions?
4. Dig into some code & tests
5. Related topics briefly mentioned:
   - CQRS, Event Modeling
   - Versioning & Schema Migration
   - Performance & Snapshotting
   - GDPR

https://ted.dev/about

What is APPLICATION STATE?

DATA IN + Outcome of DOMAIN DECISIONS

TIME

JitterTicket

Concert Ticket Sales and Scheduling System

JitterTicket Domain

CUSTOMER

CONCERT

TICKET

https://ted.dev/about

Domain Decisions

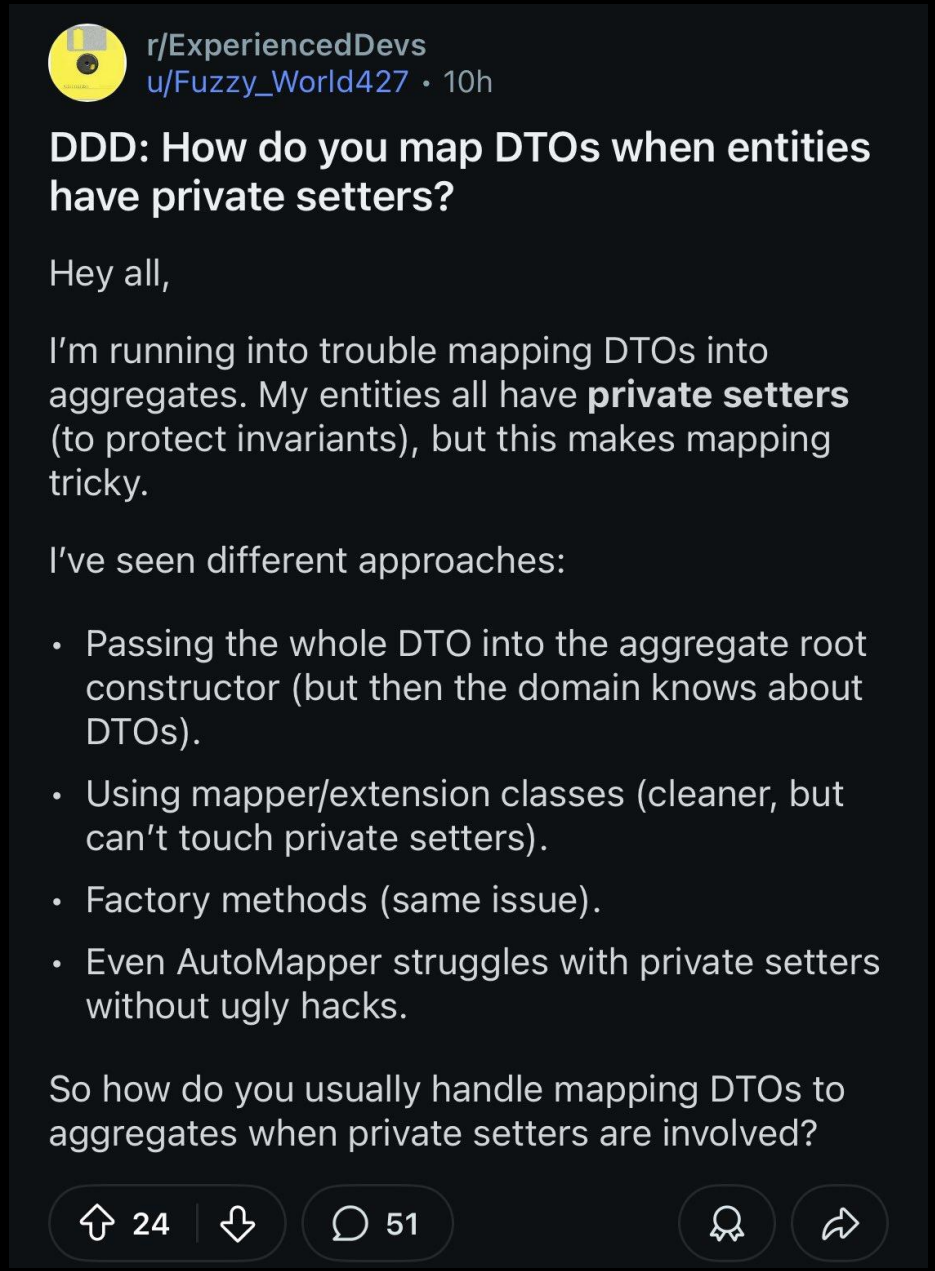
- Purchase Ticket
  - Decision (Domain Rule): enough tickets?
    - Yes -> State Change: Ticket Purchased
    - No -> No State Change (inform buyer)

- Reschedule Concert
  - Rules: not canceled; in the future; no schedule conflicts
    - 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.

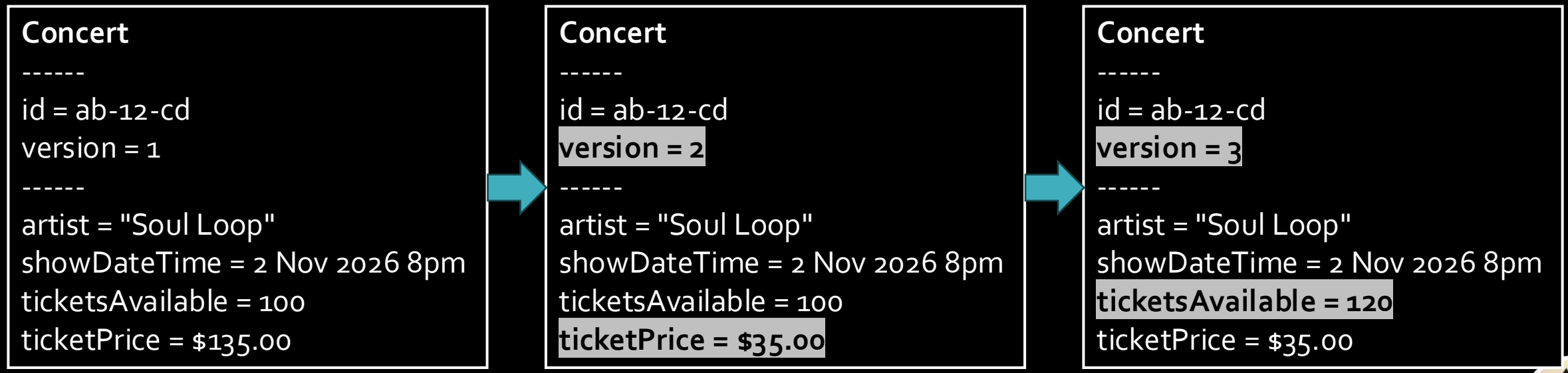
So how do you usually handle mapping DTOs to aggregates when private setters are involved?

24 51

https://ted.dev/about

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

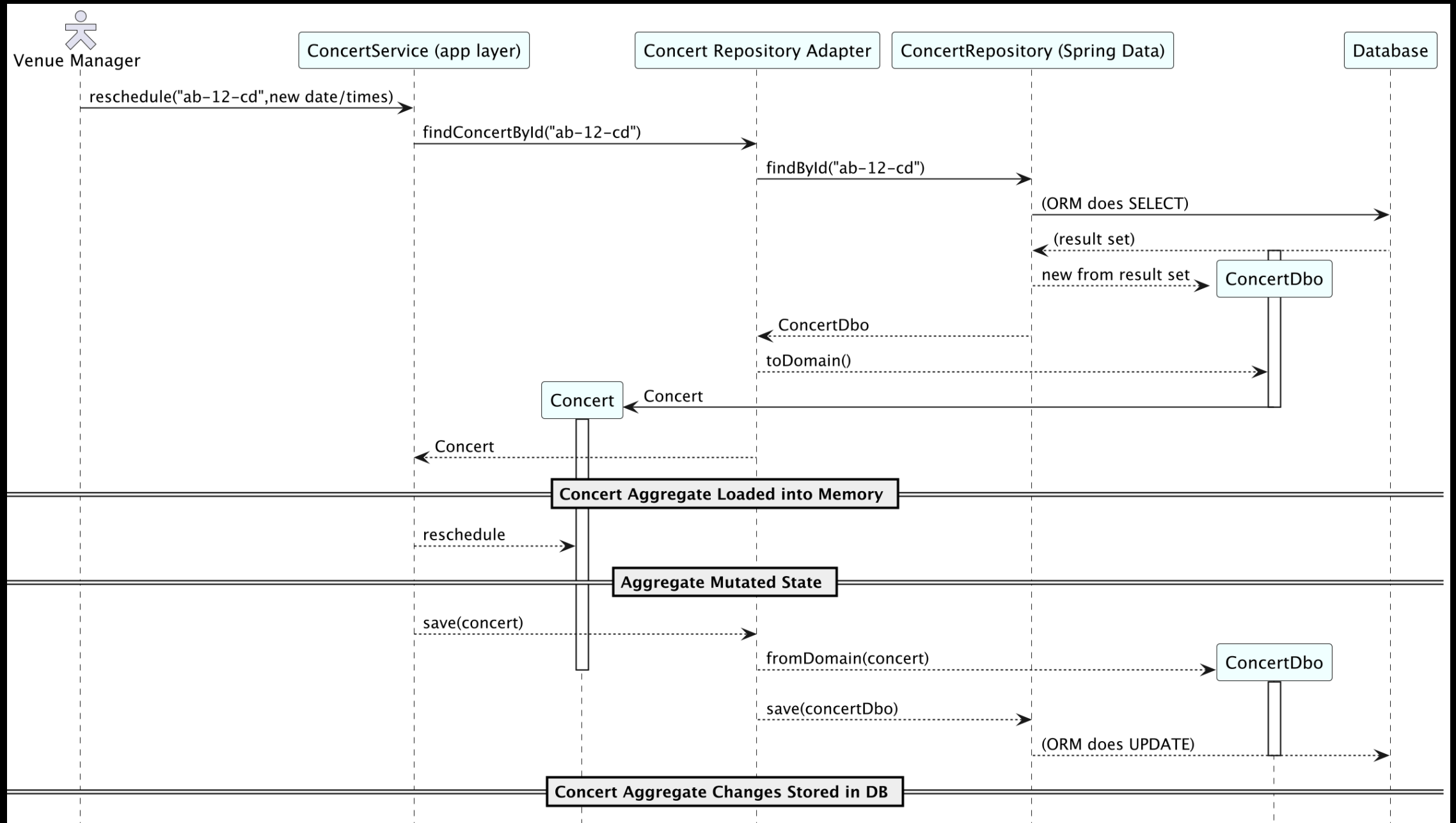
Concert Scheduled: 23, ab-12-cd, "Soul Loop", 2 Nov 2026, 20:00, 100, 135.00

Ticket Price Changed: 41, ab-12-cd, 35.00

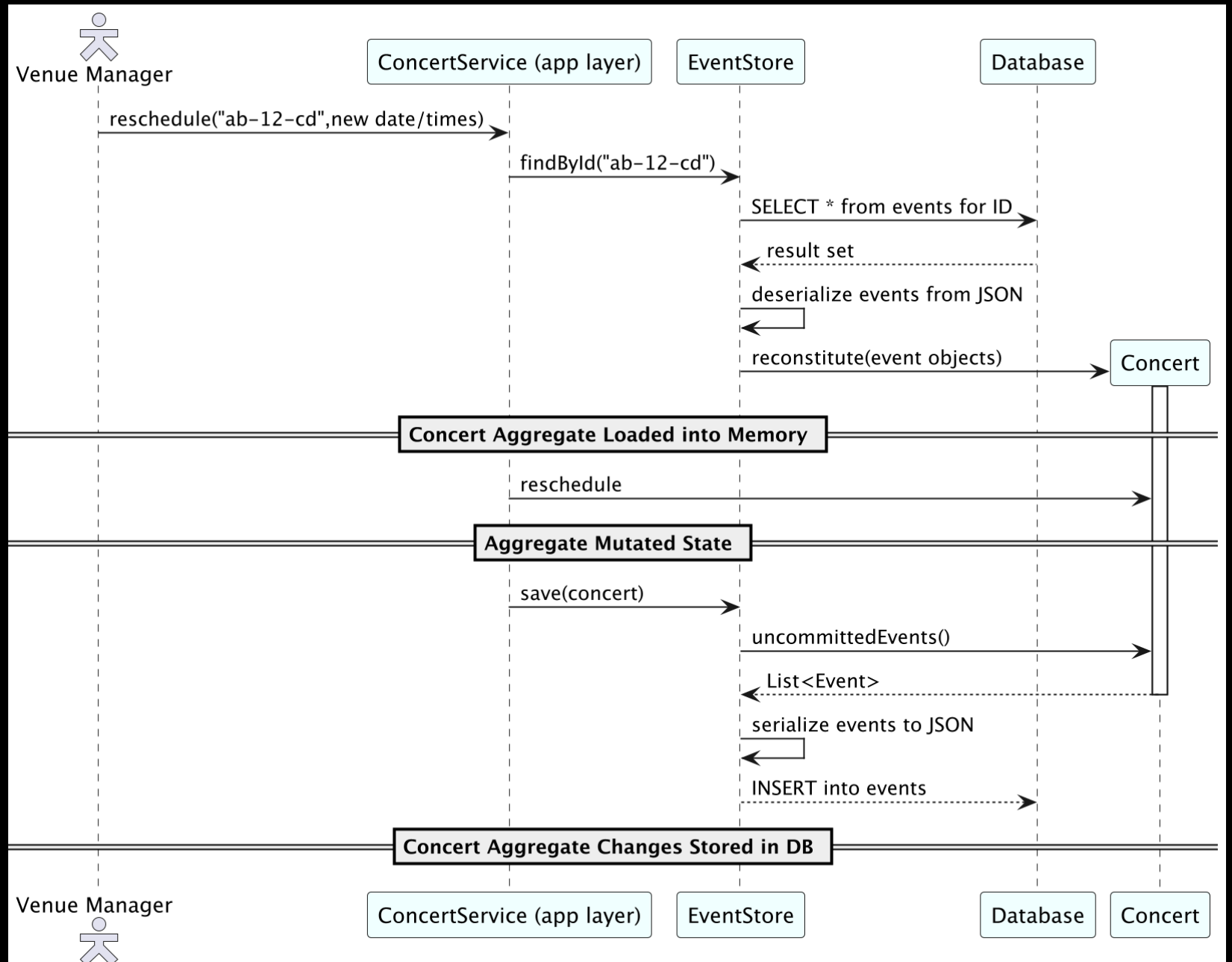
Capacity Increased: 73, ab-12-cd, 120

Concert
---
id = ab-12-cd
---
artist = "Soul Loop"
showDateTime = 2 Nov 2026 8pm
ticketsAvailable = 120
ticketPrice = $35.00

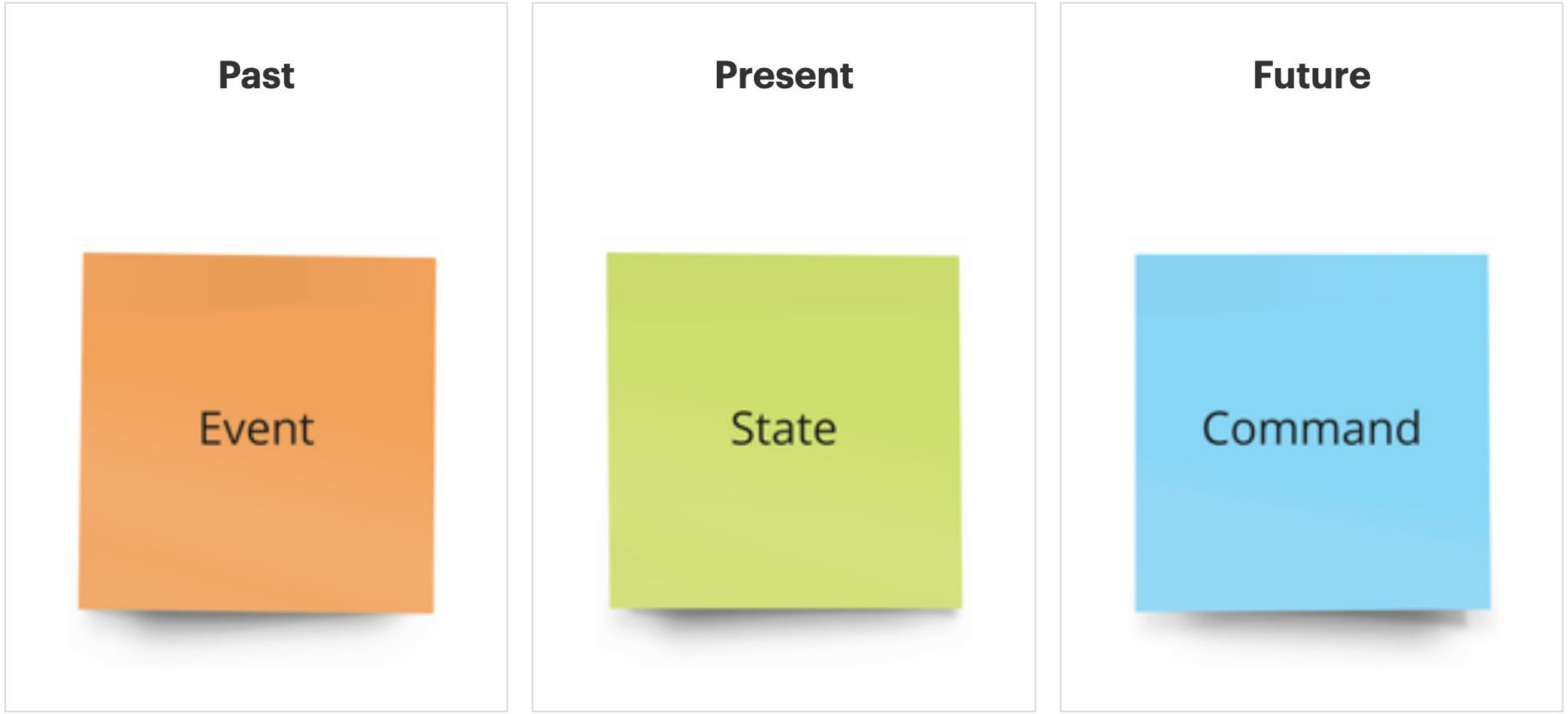
Adapter-Driven ORM Persistence



Event-Sourced Persistence (Aggregate-based)



Fundamentals



https://ted.dev/about

Definitions: EVENT (the past)

a *fact,* something that *happened*

Customer Registered

Concert Scheduled

Ticket Price Changed

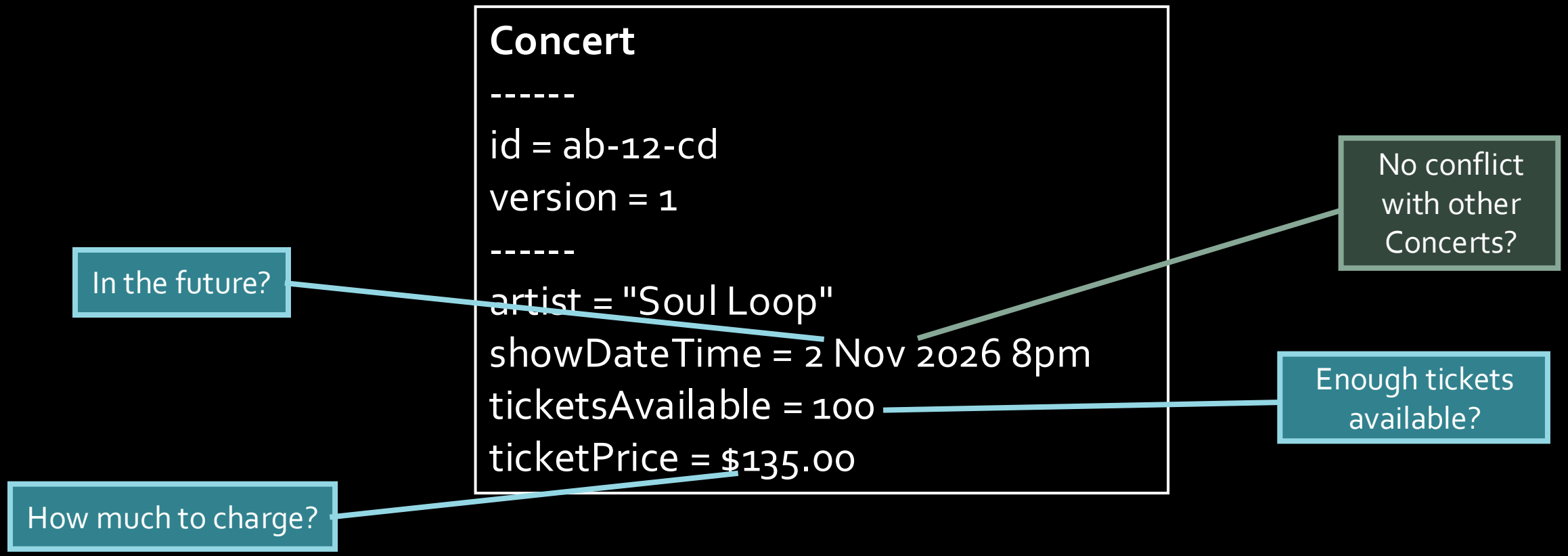
Capacity Increased

Tickets Sold

https://ted.dev/about

Definition: STATE (now)

Info used by application to *DECIDE* and record outcome of the decision

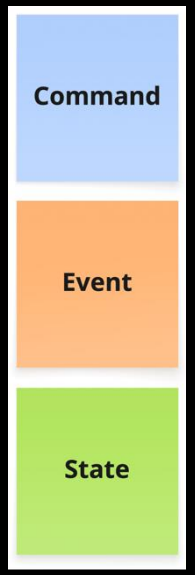


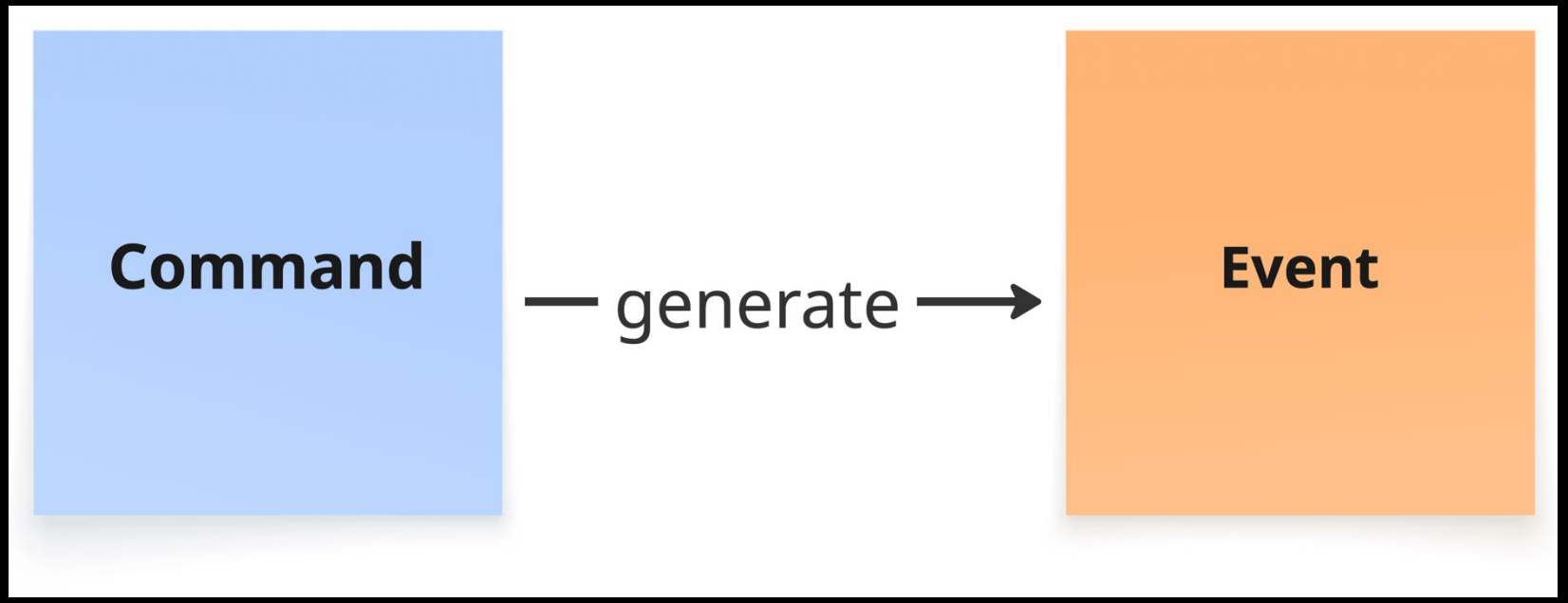
Definition: COMMAND (future)

Request **to change state**

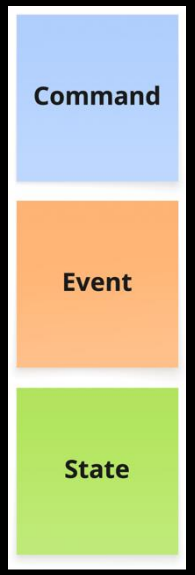
- **Register New Customer**
- **Schedule Concert**
- **Change Ticket Price**
- **Increase Capacity**
- **Purchase Tickets**

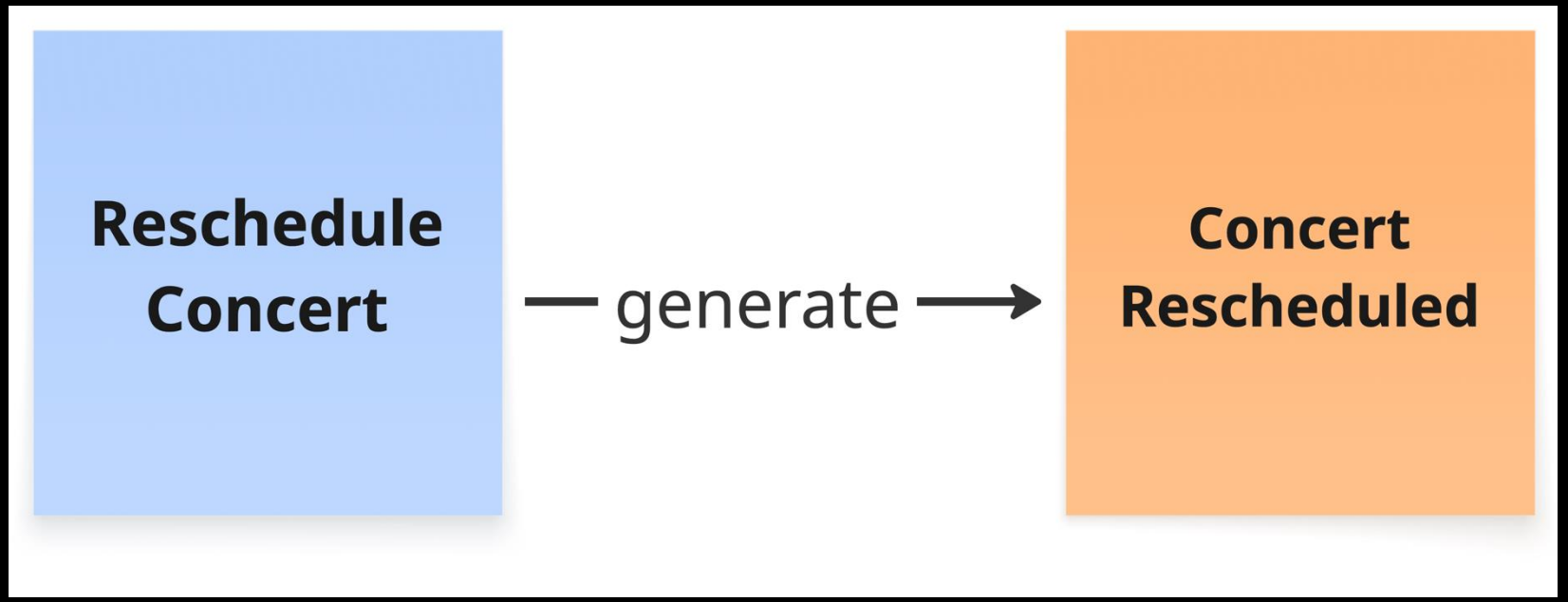
Commands Generate Events



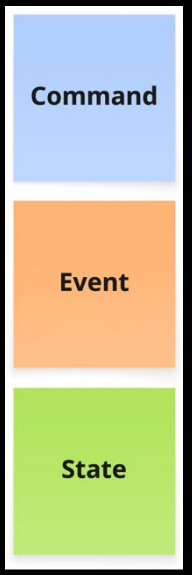


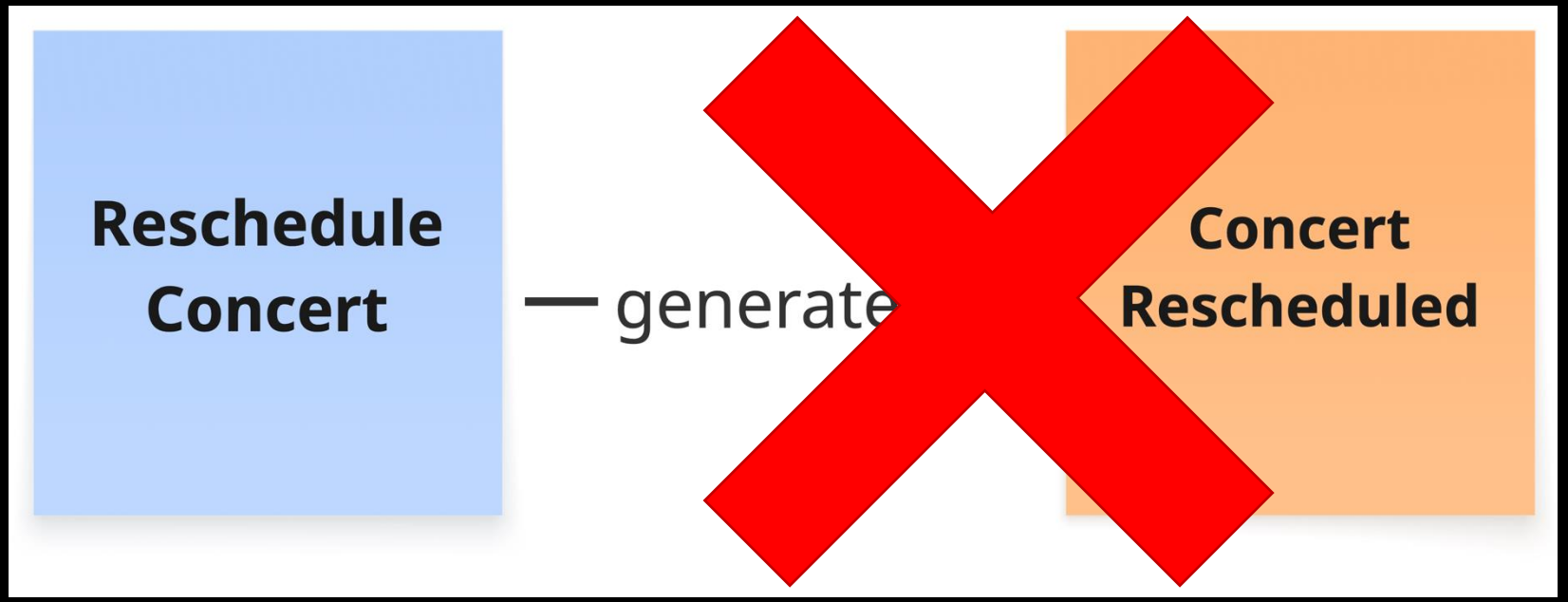
Commands Generate Events





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



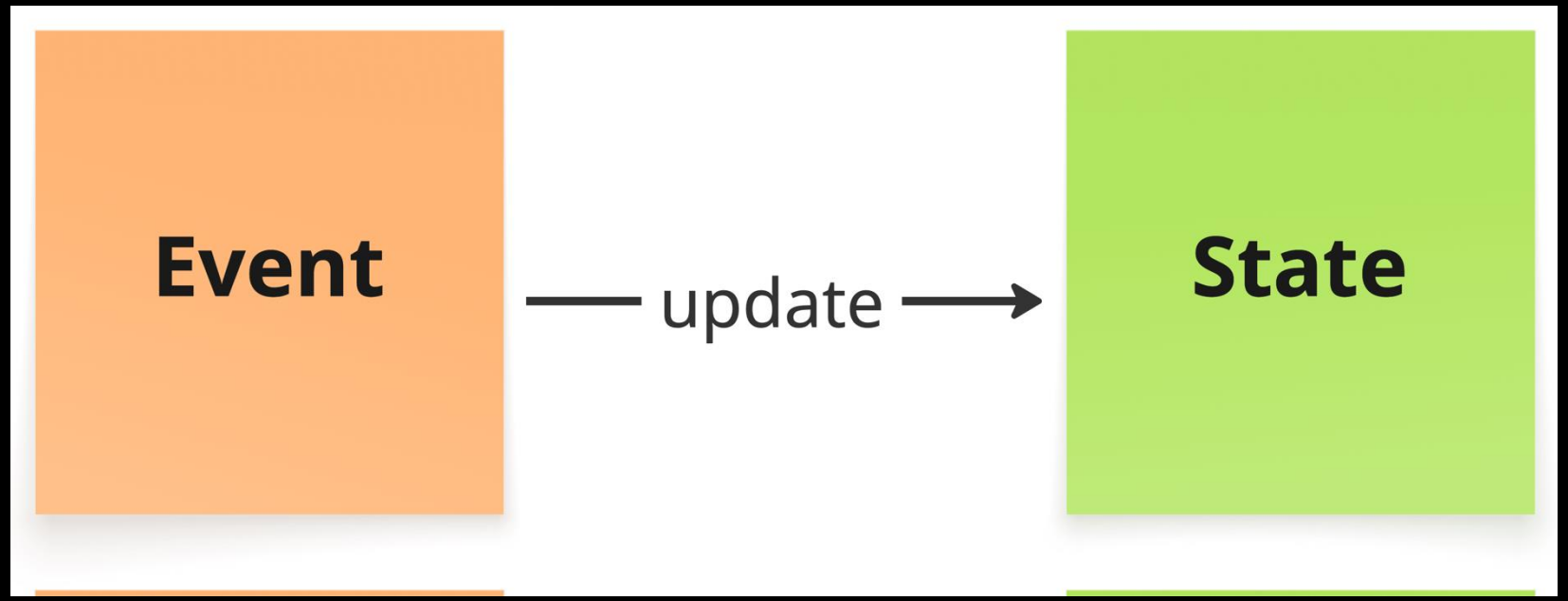


Events Update State

Command

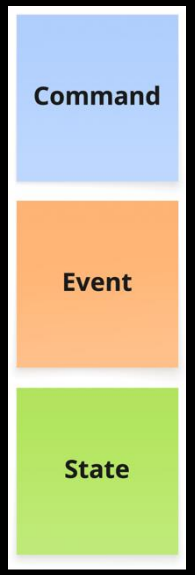
Event

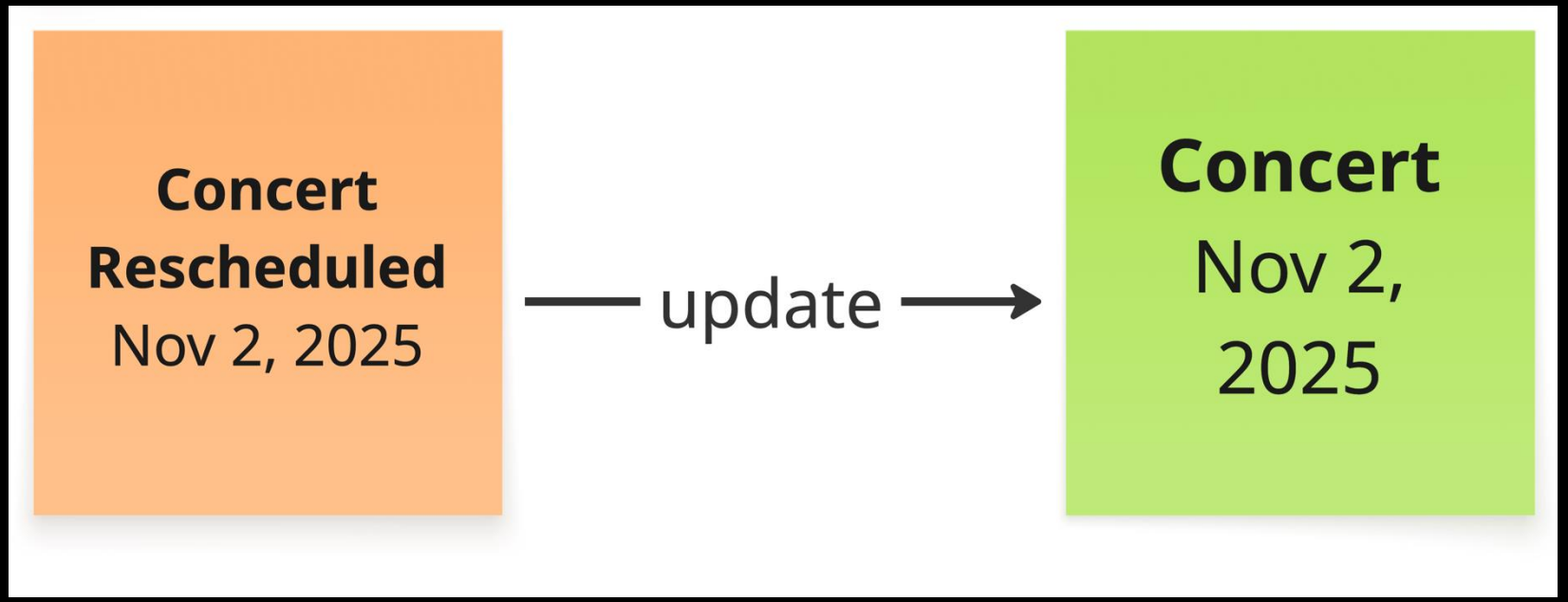
State



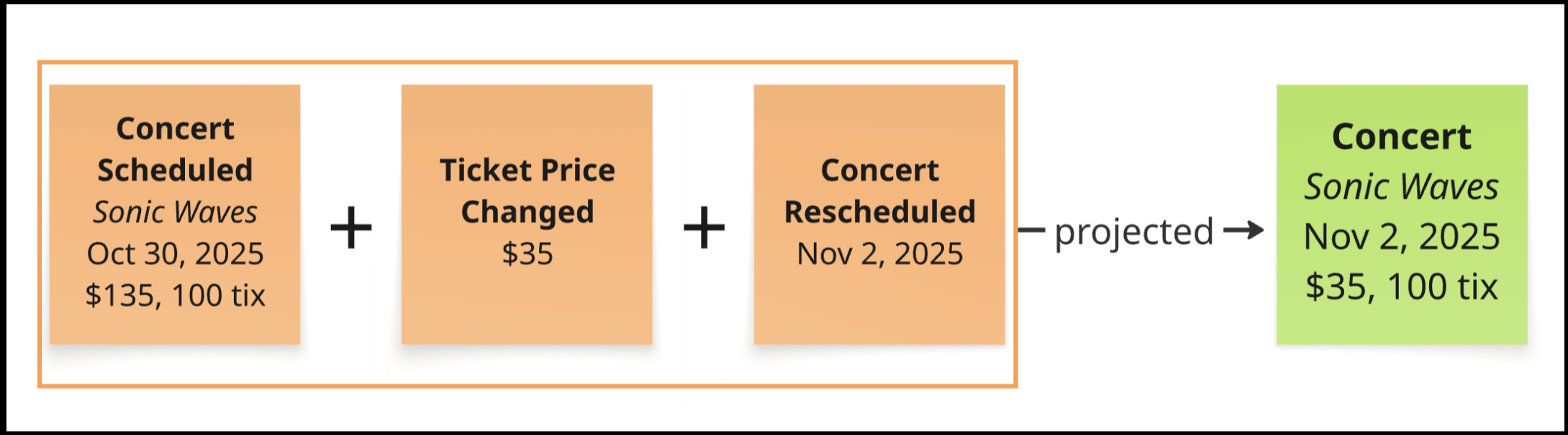
https://ted.dev/about

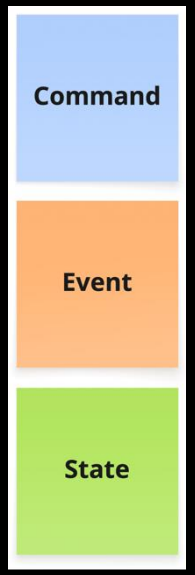
Events Update State (always!)





Events *Project* to State





Event-Sourcing PROJECTOR

A function that constructs a *Data Model* (state) by sequentially *replaying events* (fold/reduce) loaded by the *Event Store*.

Produces a PROJECTION

Projections Can Provide

- Current STATE (context) to DECIDE (command)
- Customized Views for UI
- Data Models for access via API
- Database tables for integration

One Concept: Many Uses

Event Store

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

Domain-Driven Design

Tactical Patterns

Entity

Unique (Has Identity), History, and Attributes

Value Object

Identified Only by its Attributes (immutable)

Quantify, describe, and measure

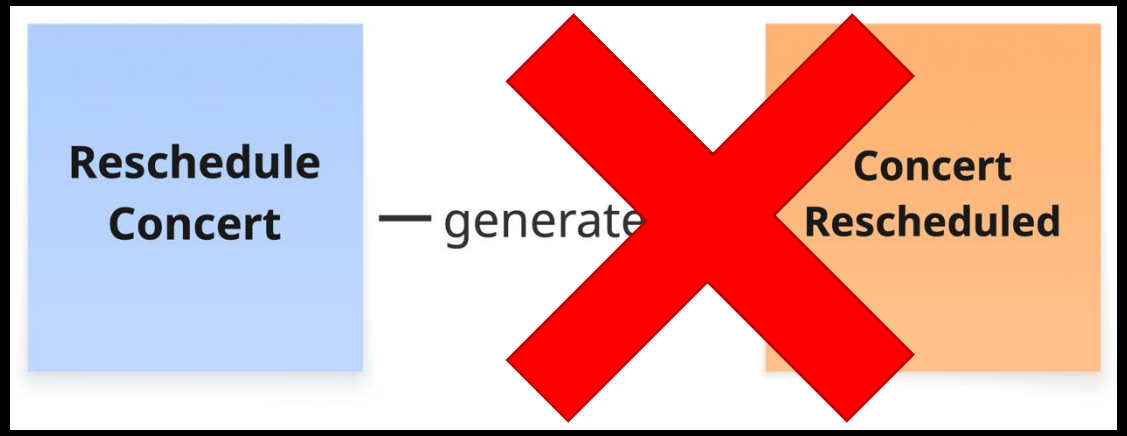
Aggregate

Consistency [transactional] Boundary

(Enforced by "Aggregate Root" Entity)

Repository

Used for Storing and Retrieving Aggregates



Aggregate & Projections

Aggregates in Event-Sourcing combine:

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

Used for Storing and Retrieving Aggregates

Where Do Decisions Live?

State + [Context] + Command Request -> Decider -> Event

Aggregate as Decision Container

e.g., Concert Aggregate holds logic for making Concert-related decisions

Aggregates Bound Consistency

With Repositories, enforce transactions

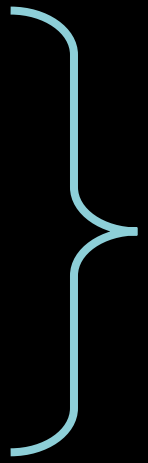
Commands Must Be Pure

Pure Domain, No I/O (No fetching, UUID, Clock)

Event Store -> fold to STATE

Event Store and/or I/O -> additional CONTEXT

triggering information -> command REQUEST



Domain Command (aka DECIDER)

A Functional Decider Approach

```
class AggregateRoot {
  static Result<Event> decide(State state,
                              Context ctx, CommandRequest req) {
    if (okToExecute(state, ctx, req)) {
      return Result.ok(new Event(...data...));
    }
    return Result.fail(reasonForFailure);
  }
}
```

A Partial OO Approach

```
// project State from List<Event> loaded from Event Store
record AggregateRoot(State state) {
  Result<Event> decide(Context ctx, CommandRequest req) {
    if (okToExecute(ctx, req)) {
      return Result.ok(new Event(…data…));
    }
    return Result.fail(reasonForFailure);
  }
}
```

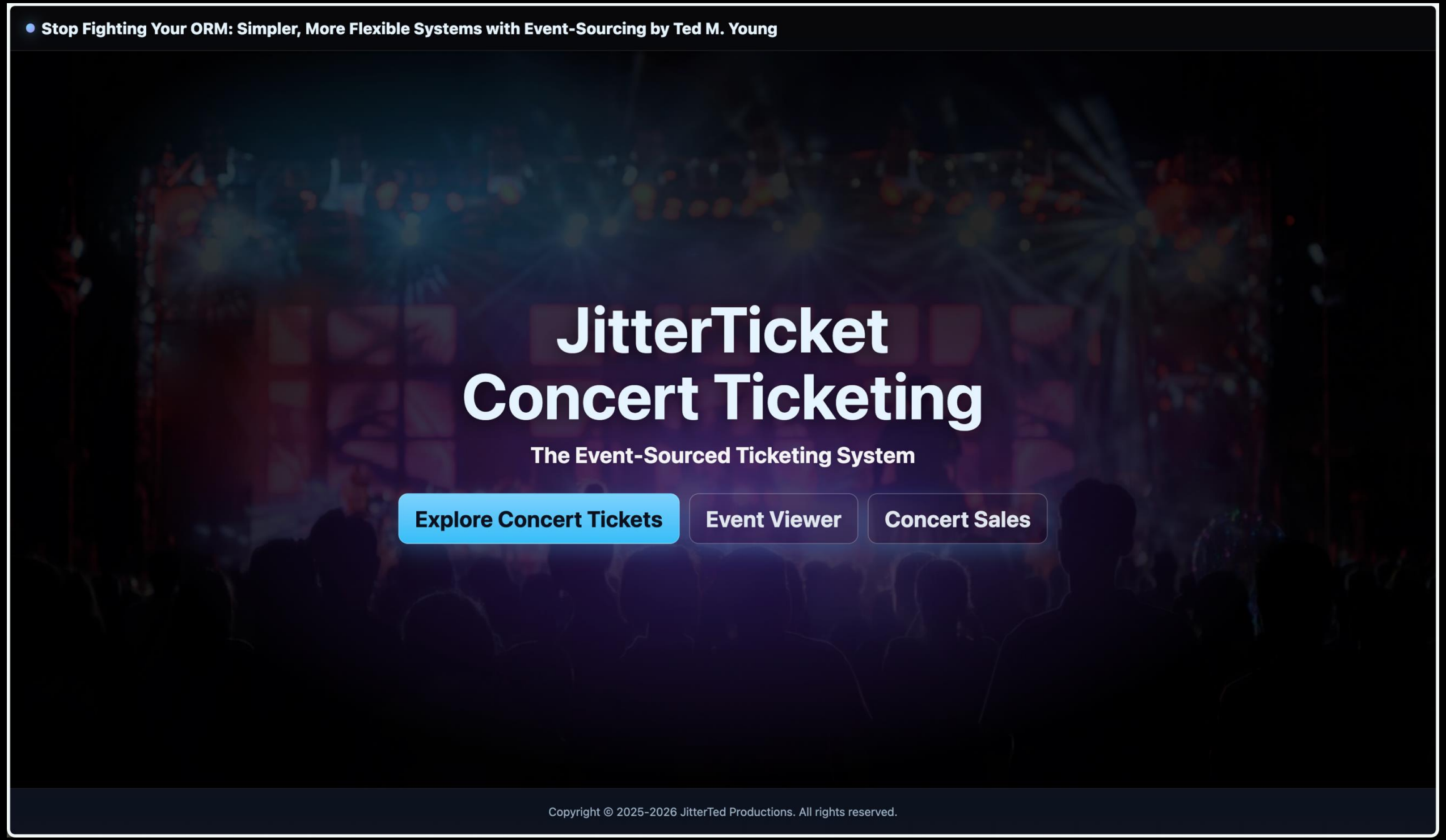
A More Traditional Approach

```
class AggregateRoot {
  private final State state;
  private final List<Event> uncommittedEvents = new ArrayList<>();

  public AggregateRoot(State state) { this.state = state; }

  void decide(Context ctx, CommandRequest req) {
    if (okToExecute(ctx, req)) {
      uncommittedEvents.add(new Event(…data…));
    }
  }

  Stream<Event> uncommittedEvents() { return uncommittedEvents.stream(); }
}
```

Events = State + Command + Decider

```
private String artist;
private int ticketPrice;
private LocalDateTime showDateTime;
private LocalTime doorsTime;
private int capacity;
private int maxTicketsPerPurchase;
private int availableTicketCount;
private boolean ticketsOnSale = true;
```

```
public void rescheduleTo(LocalDateTime newShowDateTime, LocalTime newDoorsTime) {
    if (newShowDateTime.isAfter(showDateTime) && capacity - availableTicketCount == 0) {
        ConcertRescheduled concertRescheduled =
                new ConcertRescheduled(getId(), null, newShowDateTime, newDoorsTime);
        enqueue(concertRescheduled);
    }
}
```



https://ted.dev/about

JitterTicket Flows

tldraw

JitterTicket: Concert Ticketing System

Diving Into the Code

Benefits of Event-Sourcing

Ask Questions You Didn't Know You Had

How many Customers buy maximum # of Tickets?

How often are we Rescheduling Concerts?

Do we ever reduce Maximum Capacity?

Data Shape Matches Questions Precisely

Relationships between customers = Graph projection

Music recommendations = Vector projection

API for external usage = JSON in-memory projection

Consistent Test Structure (Events→Cmd→Event)

- All Command-oriented tests are:
  - GIVEN: Event(s)
  - WHEN: execute Command
  - THEN: expect Event

(ConcertTest.rescheduleConcertGeneratesConcertRescheduled)

Consistent Test Structure (Events→Projection)

- All State-oriented (projection) tests are:
  - GIVEN: Event(s)
  - THEN: expected State

(ConcertSalesProjectorTest.multipleConcerts_MultipleTicketsSoldPerConcert_ProjectsRowPerConcert)

Easier Troubleshooting

Discover a Bug? See the trail of decision outcomes.

Customized Projections

Disposable: easy to create, change, and recreate

Immutable Events*

Never lose information: there is no DELETE

* Events might change as part of Event Migration

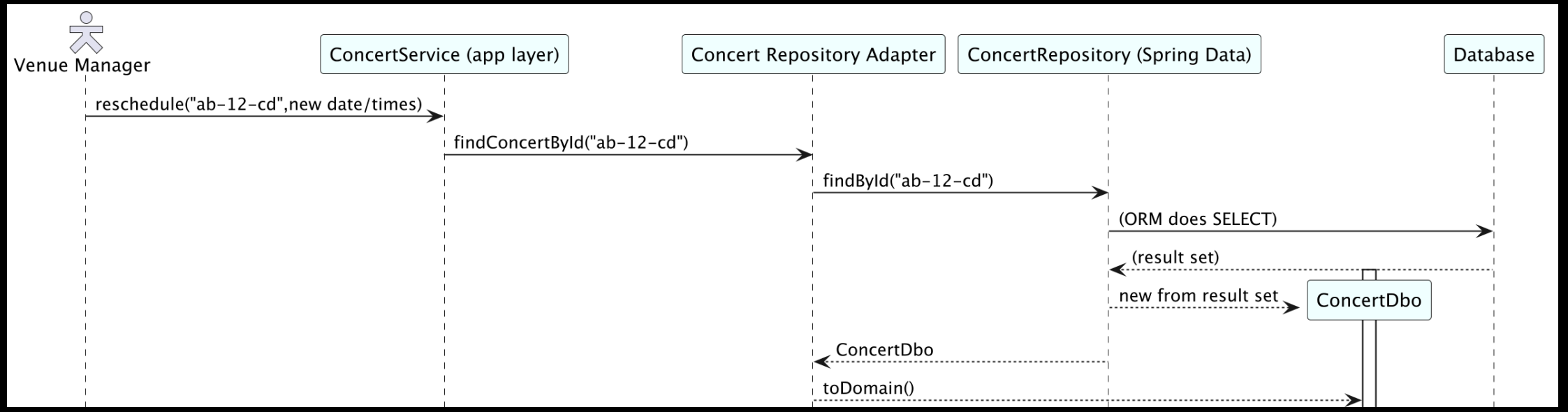
Fewer Concepts

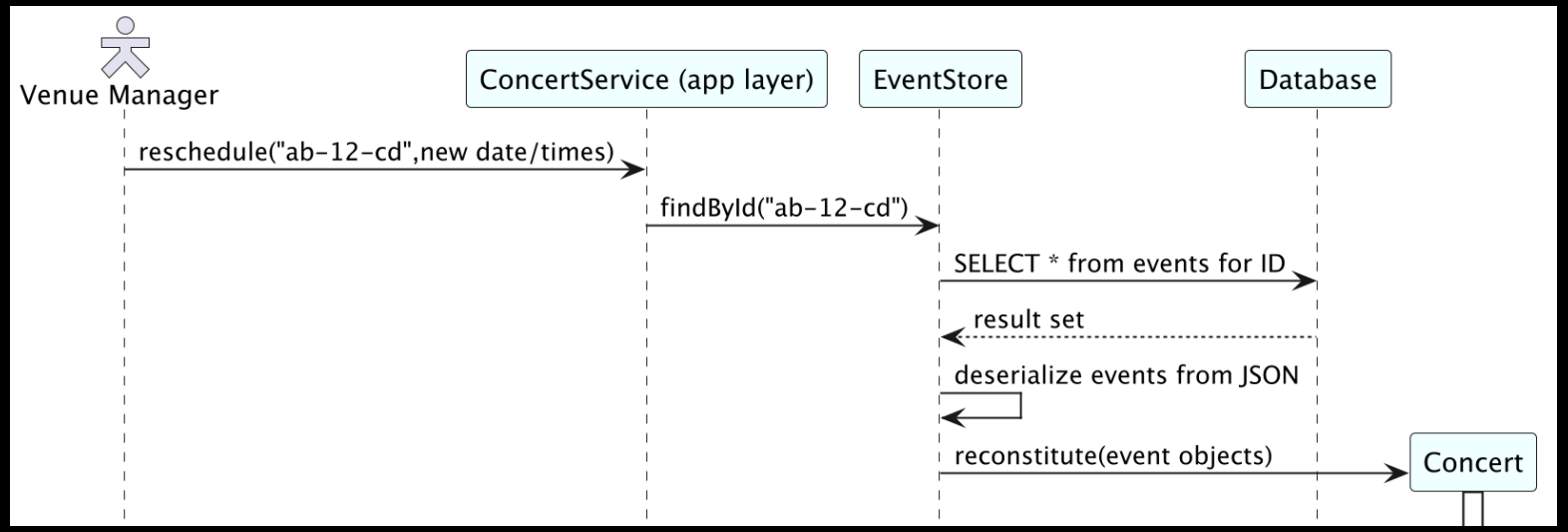
Commands, Events, Projections

Less complexity than ORMs

No awkward mapping, lazy loading, or dirty-checking

Fewer Concepts





https://ted.dev/about

Replayability

Time Travel: see state of system in the past

Still Need Aggregates?

Aggregates + Entities can still be useful

Soften the Decision Boundaries

Value Objects still useful

Let's talk about Performance

Similar, if not BETTER than ORM

Startup New Projections Creation

1 Million (1,000,100) Events:

```
11:41:16.848 ProjectionCoordinator    : Catching up for AvailableConcertsProjector
11:41:16.870 JdbcEventStore           : Fetching [ConcertRescheduled, TicketSalesStopped, ConcertScheduled, TicketsSold], after event sequence: C
11:41:39.447 JdbcEventStore           : Fetched 1,000,100 events, mapping to Domain events
11:41:39.451 ProjectionCoordinator    : Fetched events, now updating projection...
11:41:43.562 ProjectionCoordinator    : Subscribing to event store for all events
11:41:43.581 ProjectionCoordinator    : Catching up for ScheduledConcertsProjector
11:41:43.593 JdbcEventStore           : Fetching [ConcertRescheduled, TicketSalesStopped, ConcertScheduled, TicketsSold], after event sequence: C
11:42:06.159 JdbcEventStore           : Fetched 1,000,100 events, mapping to Domain events
11:42:06.159 ProjectionCoordinator    : Fetched events, now updating projection...
11:42:10.406 NewProjectionCoordinator : Catching up on events for event types: [class CustomerRegistered]
11:42:10.406 JdbcEventStore           : Fetching [CustomerRegistered], after event sequence: Checkpoint(0/INITIAL)
11:42:10.641 JdbcEventStore           : Fetched 10,000 events, mapping to Domain events
11:42:10.641 NewProjectionCoordinator : Fetched events, now handling them...
11:42:10.700 ProjectionCoordinator    : Catching up for AllConcertsProjector
11:42:10.719 JdbcEventStore           : Fetching [ConcertRescheduled, TicketSalesStopped, ConcertScheduled, TicketsSold], after event sequence: C
11:42:33.915 JdbcEventStore           : Fetched 1,000,100 events, mapping to Domain events
11:42:33.915 ProjectionCoordinator    : Fetched events, now updating projection...
11:42:38.036 ProcessorConfiguration   : Concert Started Processor: catching up on all events
11:42:38.036 JdbcEventStore           : Fetching [TicketSalesStopped, ConcertScheduled, ConcertRescheduled], after event sequence: Checkpoint(0/
11:42:38.042 JdbcEventStore           : Fetched 100 events, mapping to Domain events
```

https://ted.dev/about

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!

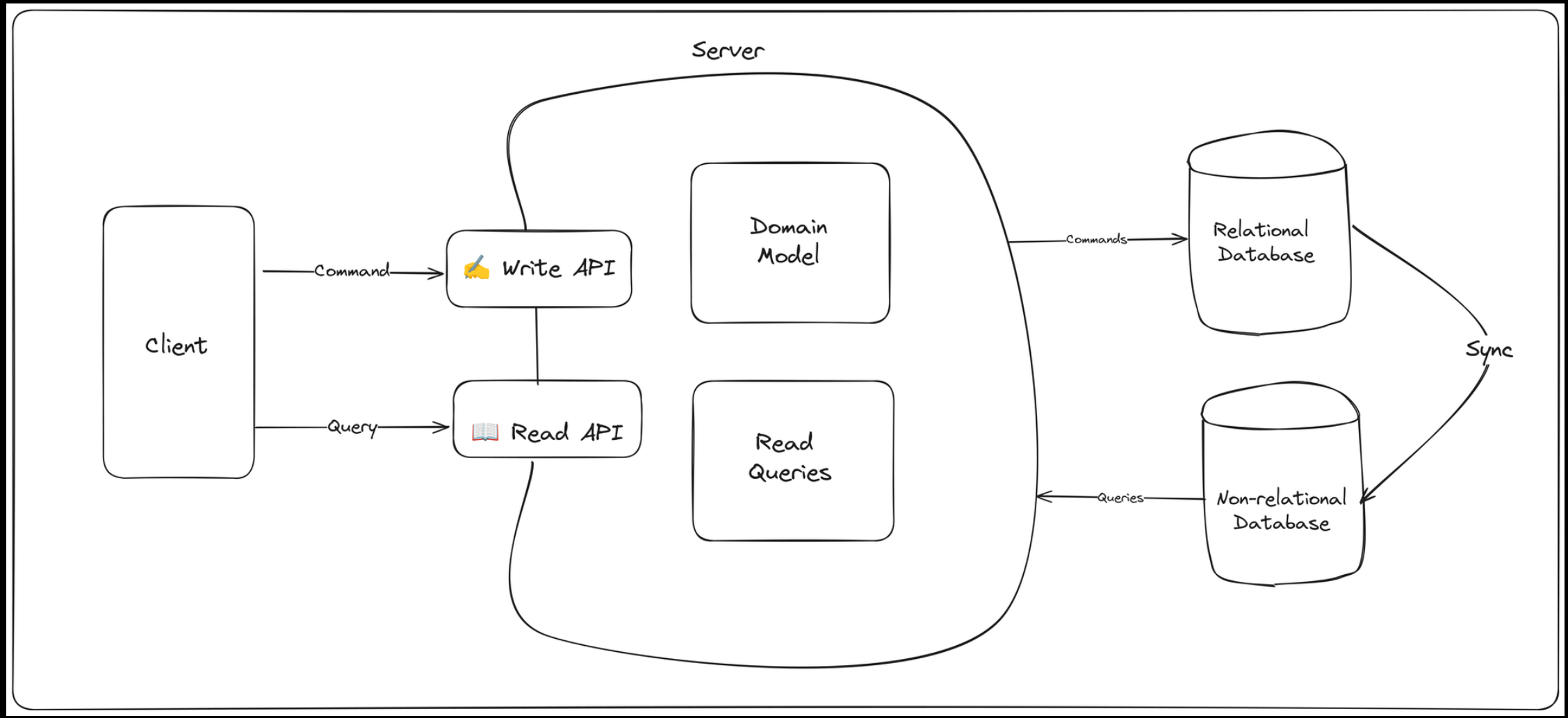
Other Topics…

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

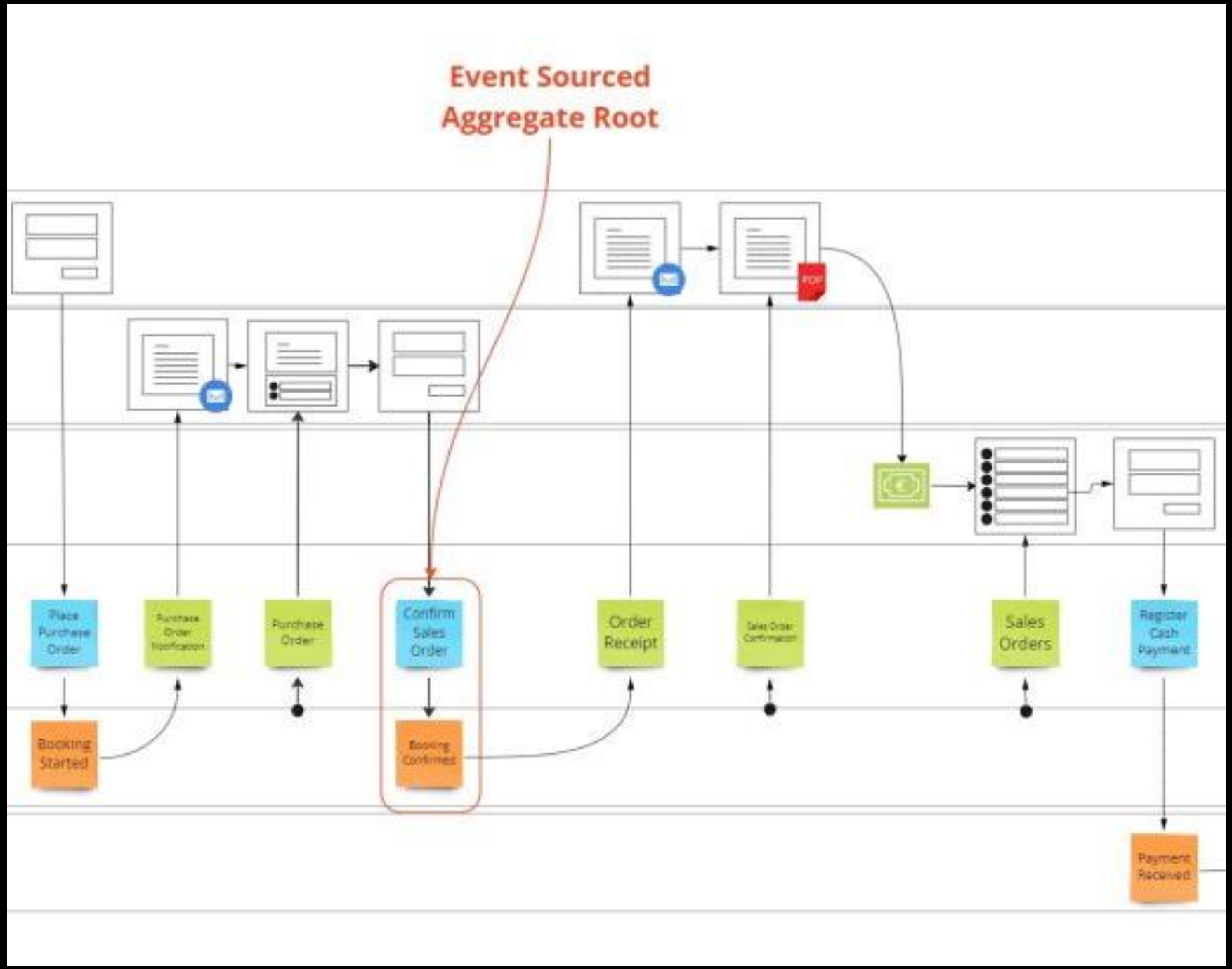


Event Modeling

Command

Event

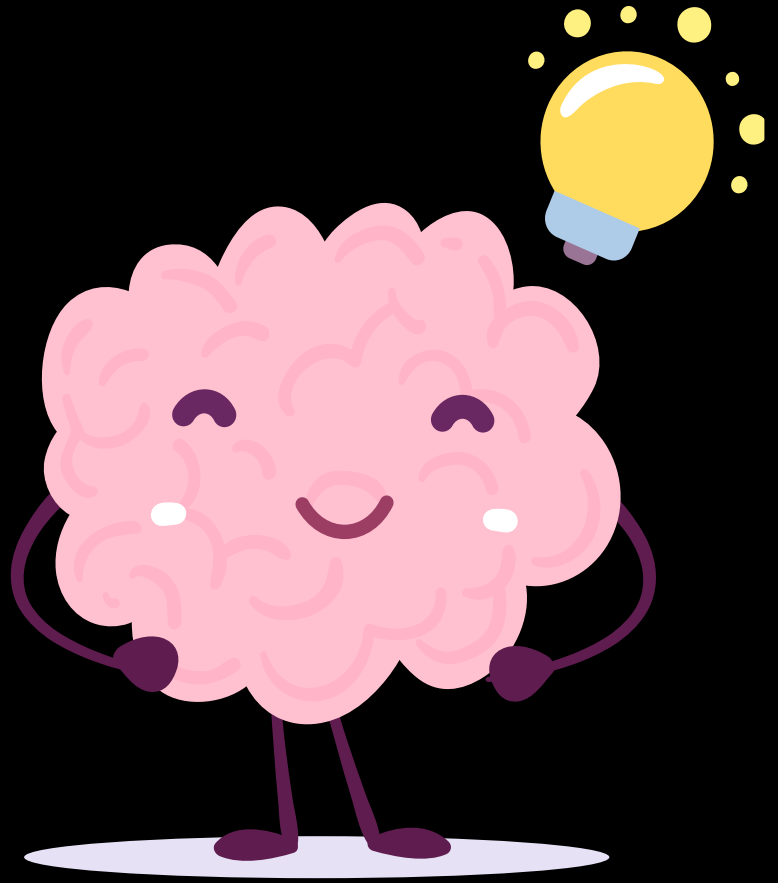
State



Ted M. Young
Java Trainer, Coach, Live Coder,
and Creator of JitterTed's TDD Game

Get in touch: ted@tedmyoung.com
About me: https://ted.dev/about

Questions?



Ted M. Young
Java Trainer, Coach, Live Coder,
and Creator of JitterTed's TDD Game

Get in touch: ted@tedmyoung.com
About me: https://ted.dev/about

Thank You...

Source Code? Slides?
https://ted.dev/talks/

Please provide feedback here: