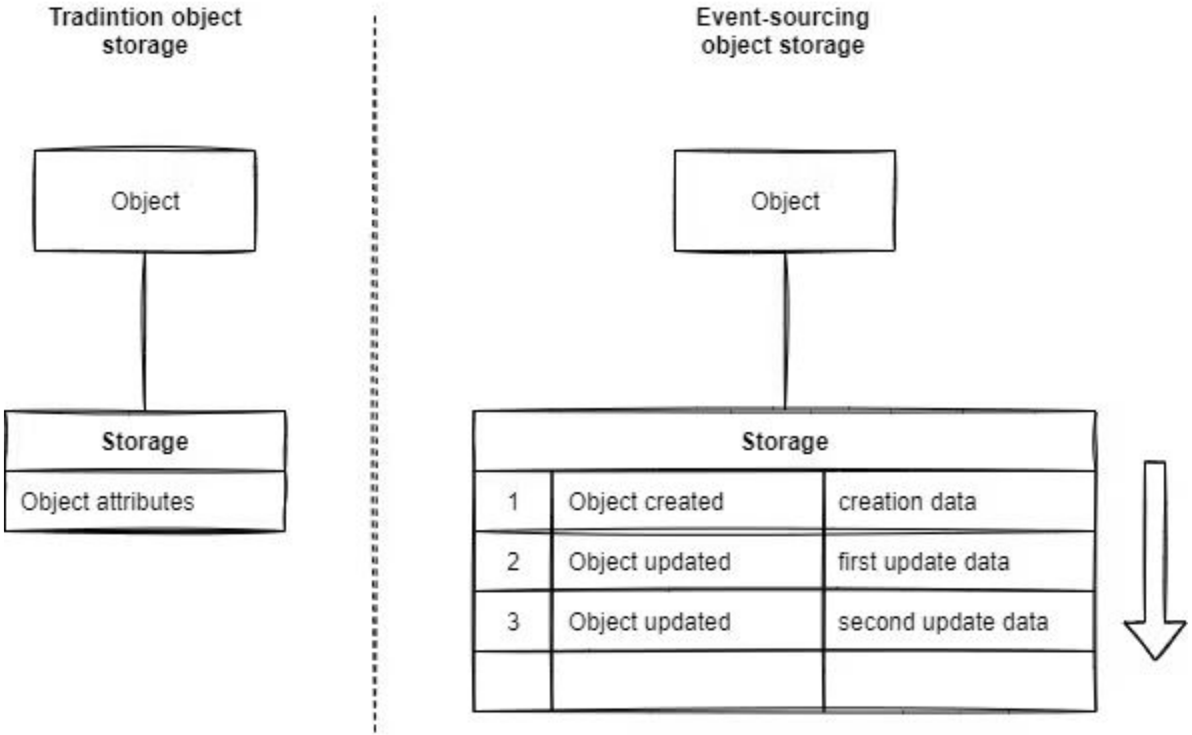
Event Sourcing using Akka Persistence

Agenda

- What is Event Sourcing (ES)?
- What is CQRS?
- Features of ES
- Challenges with ES
- ES with Akka Persistence

What is Event Sourcing?

- ★ ES is a design pattern for storing the **internal state** of a system as a **sequence of events**
- ★ In ES we don’t persist the current state, but **changes** captured in events that lead to this state



Tradintion object storage

Object

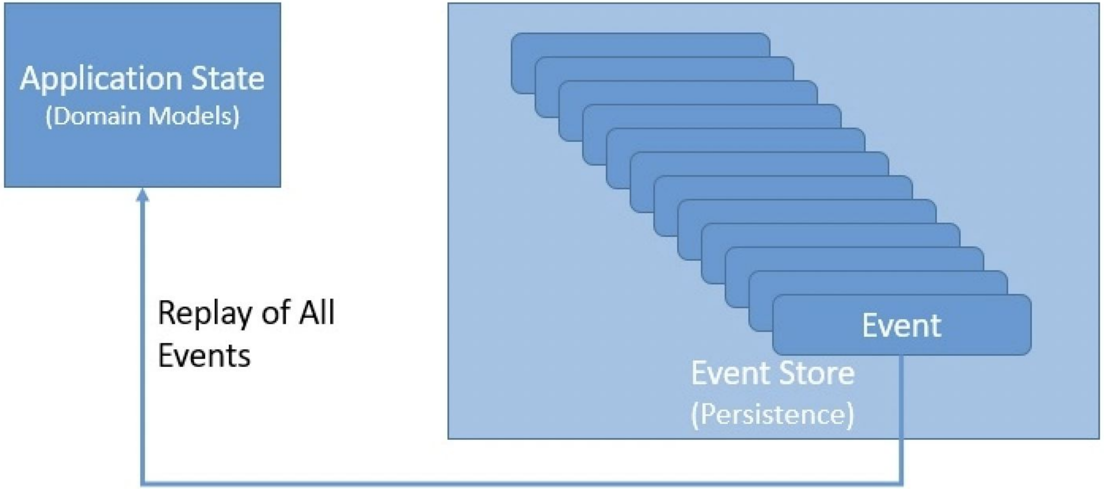
Storage
Object attributes

Event-sourcing object storage

Object

Storage		
1	Object created	creation data
2	Object updated	first update data
3	Object updated	second update data

★ **The state** of the system is derived by **replaying these events** in order in which these events were produced



ES Journal

★ Journal is a sequence of events, known as event log or event store

★ Each event is identified by a unique ID, sequence number and it represents a change that was made to the system state over time

★ Journal events are always **immutable and append-only**

★ Journal is the **source of truth**

Commands vs Events vs Aggregate

- ★ **Command** is a **change request** to the system
- ★ **Event** is the actual **fact** that has happened in the past
- ★ An **aggregate** is a collection of related domain objects
  - ○ Used to enforce consistency and integrity within the system
  - ○ Used to handle commands and generate events
- ★ When a new command is handled, the system can produce
  - ○ … one or more events
  - ○ … or an error

When a command is received, the system validates the command, enforcing business and logical constraints to ensure that only **valid and authorized** events are generated and written to the journal

Current application state

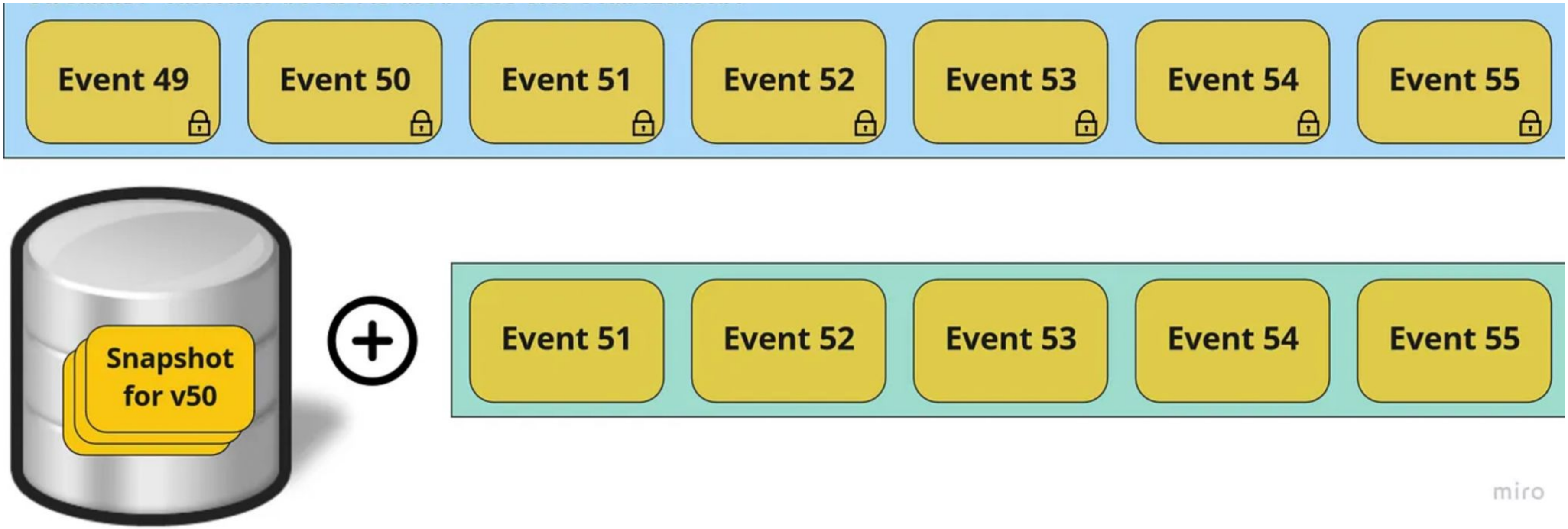
- ★ Is built based on event journal
- ★ can be discarded completely and rebuilt at any time
- ★ can be used to validate commands prior generating events because events should not lead to inconsistent state

Example of “Bank Account” aggregate

Command in order	Event(s) or Error(s)	State
1: Increase on 100	Event: Increased on 100	Balance is 100
2: Decrease by 30	Event: Decreased on 30	Balance is 70
3: Decrease by 100	Error: Balance falls below 0	unchanged
4: Increase on 2000	Error: Balance limit of 1000 is reached	unchanged

Snapshots

- If there’re a lot of events, replaying events to build state can be slow
- Creating snapshots each N-events is a nice optimization
- In case of using snapshots, current state is calculated by
  - Restoring the state to the latest snapshot
  - And then replaying remaining events, that were generated after the snapshot creation, on top

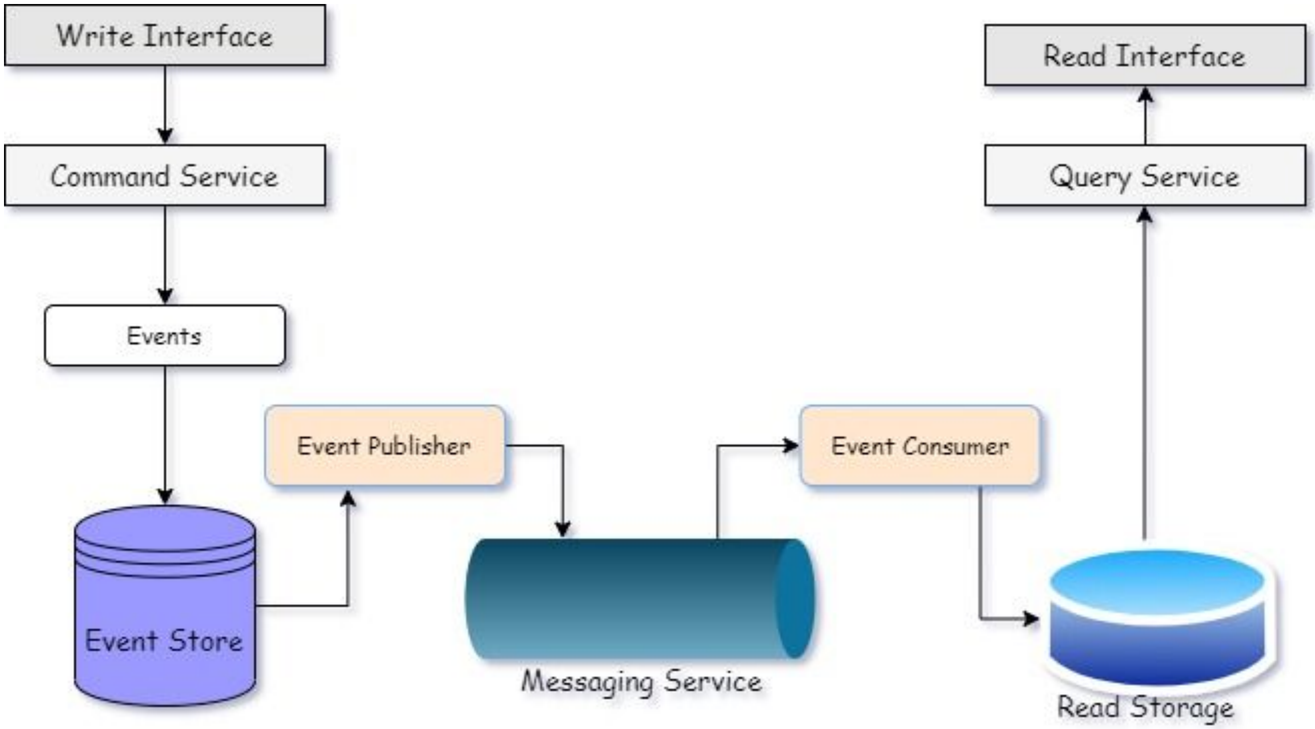

Event 49
Event 50
Event 51
Event 52
Event 53
Event 54
Event 55
Snapshot for v50
Event 51
Event 52
Event 53
Event 54
Event 55
miro

What is CQRS?

★ Command Query Responsibility Segregation

★ CQRS is a design pattern that separates handling read and write operations

★ CQRS separates reads and writes into different models, using **commands** to update data, and **queries** to read data


Write Interface
Command Service
Events
Event Store
Event Publisher
Messaging Service
Event Consumer
Read Storage
Query Service
Read Interface

Write Model

- ★ Is responsible for handling commands to the system
- ★ Command and Aggregate Handlers:
  - ○ the command handler is responsible for receiving commands and passing them to the appropriate aggregate
  - ○ the aggregate handler is responsible for processing commands and generating events on valid commands

Read Model

★ Is responsible for handling read requests to the system and generating views or projections

★ Is optimized for fast queries and can be updated asynchronously and eventually in the background

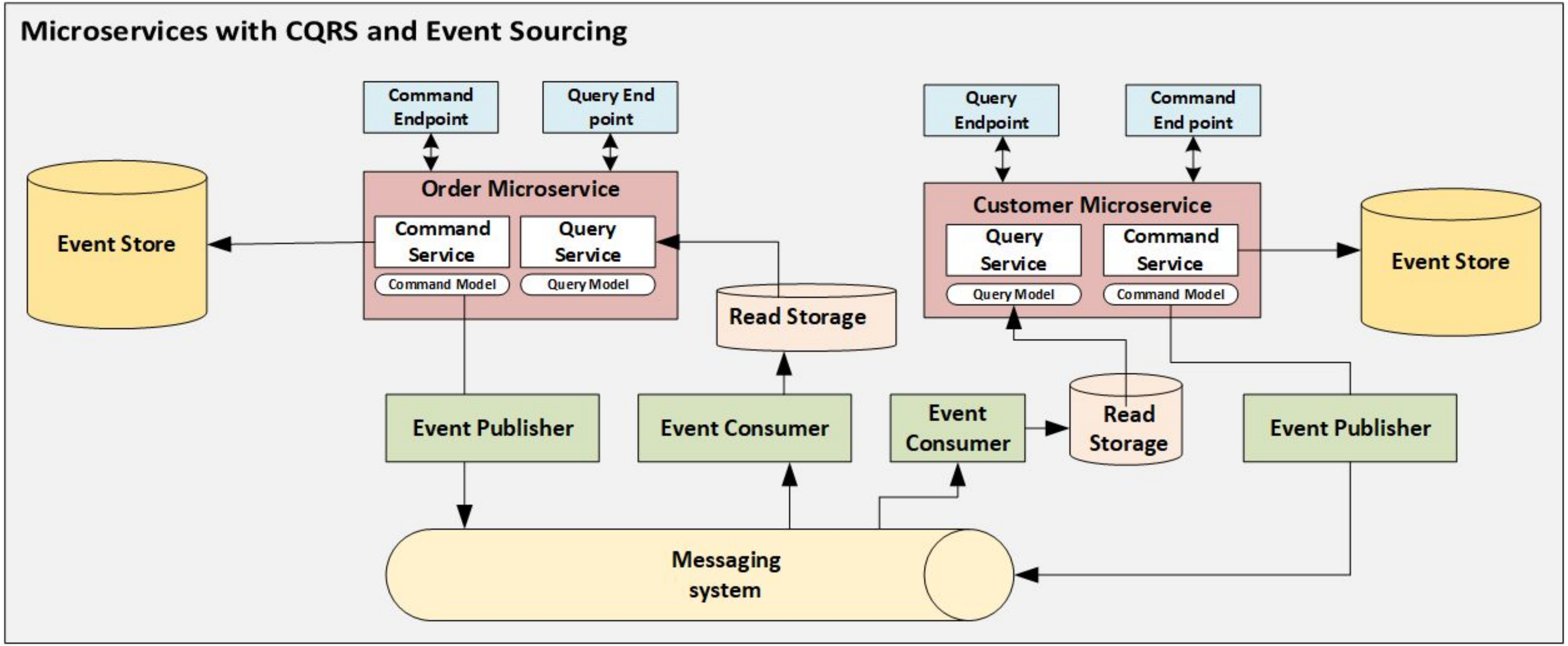
Views

★ A view is a representation of the state that is optimized for specific use cases

★ Views can be stored as materialized views

★ A materialized view is a precomputed view that is stored in a separate datastore for faster access

★ Materialized views can be updated in real-time or on a scheduled basis


Microservices with CQRS and Event Sourcing
Command Endpoint
Query End point
Order Microservice
Command Service
Query Service
Command Model
Query Model
Event Store
Read Storage
Event Publisher
Event Consumer
Query Endpoint
Command End point
Customer Microservice
Query Service
Command Service
Query Model
Command Model
Event Store
Event Consumer
Read Storage
Event Publisher
Messaging system

Features of ES

- ★ Rebuild state at any point of time
- ★ Avoids relational structure of the data
- ★ Compatibility with CQRS & Domain Driven Design

Immutable event log

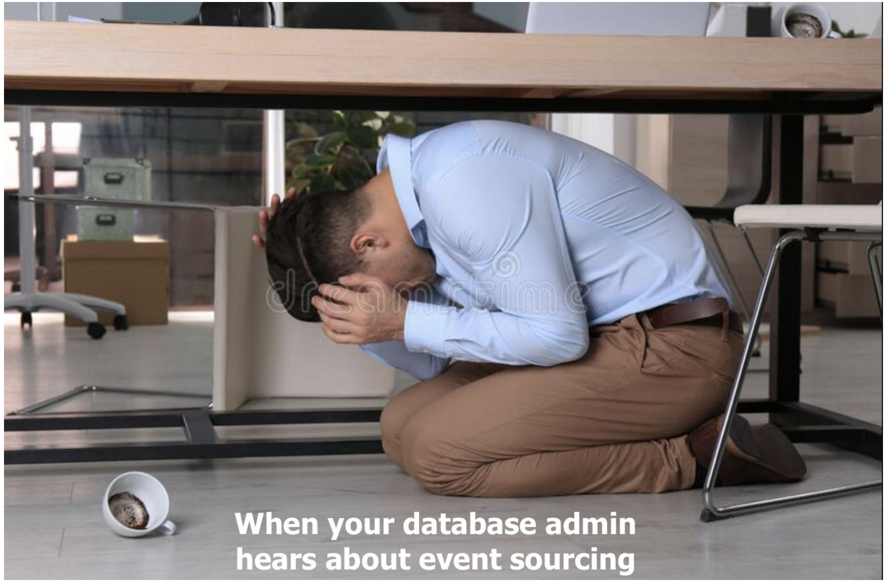
★ provides a complete history of all changes to the domain model over time, easy to trace history of changes (who did, what and when)

★ easy to build views for audit log and demonstrate compliance with regulatory requirements

Challenges of ES

- ★ Increased development and operational complexity
- ★ Evolution of system, events and data models
- ★ Choice of an aggregate root

Operational and development overhead



Operational and development overhead

- These approaches require additional infrastructure to handle
  - event stores for a journal
  - message brokers
  - materialized views
- This can require additional resources and management overhead to maintain

Operational and development overhead

Many tips to optimize solution

- Build snapshots of the state
- Configure TTL for operational data (events, snapshots)
- Move historical data to cheaper storages with cheaper disks
- Use cloud-based infrastructure
- Use event sourcing frameworks

Evolution of events and data models

★ Systems evolve over time without having to change the underlying data model, but event and snapshot formats can change, need to understand how to

  ○ Choice of data formats of events

  ○ deal with event evolution

  ○ and how long you keep backwards compatibility

  ○ Personal comment: exposing journal events is a bad idea

Choice of aggregate root

- ★ An aggregate is a collection of related domain objects that are used to enforce **consistency** and **integrity** within the system
- ★ **Too small** aggregate lead to need for cross-aggregate transactions
- ★ **Too large** aggregate lead to large state, complicated events, system **scales** only to some limit

- *Why did the event sourcing engineer refuse to go to the party?*
- *Because he was busy replaying all the events of the day!*

@ chat gpt

Evolution Engineering | ACCELERATING EVOLUTION