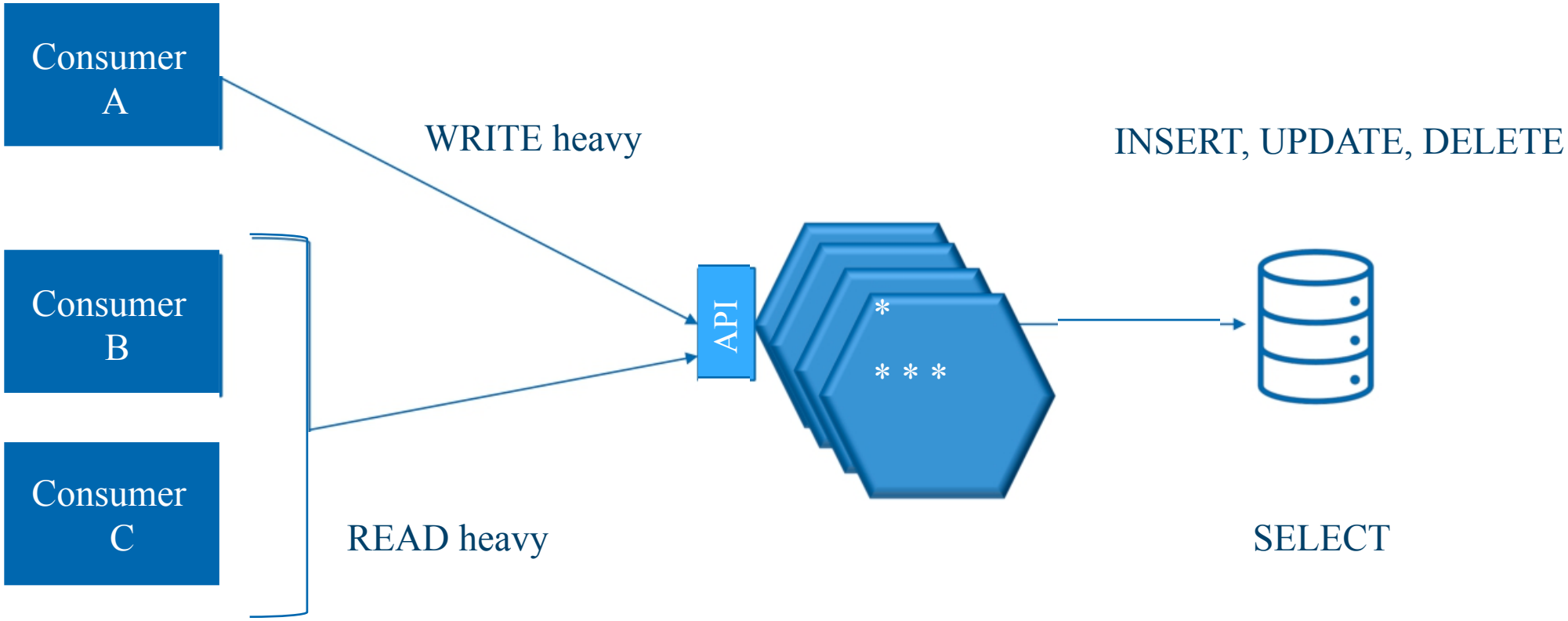
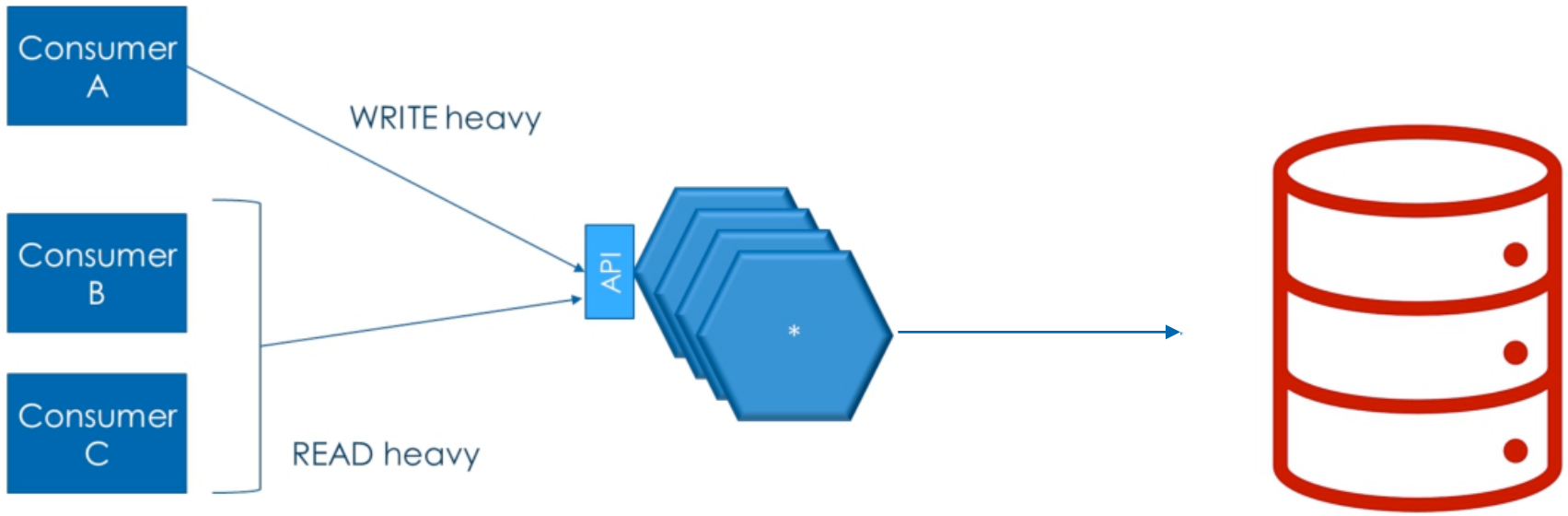
Performance & Data Integrity

Microservice Performance



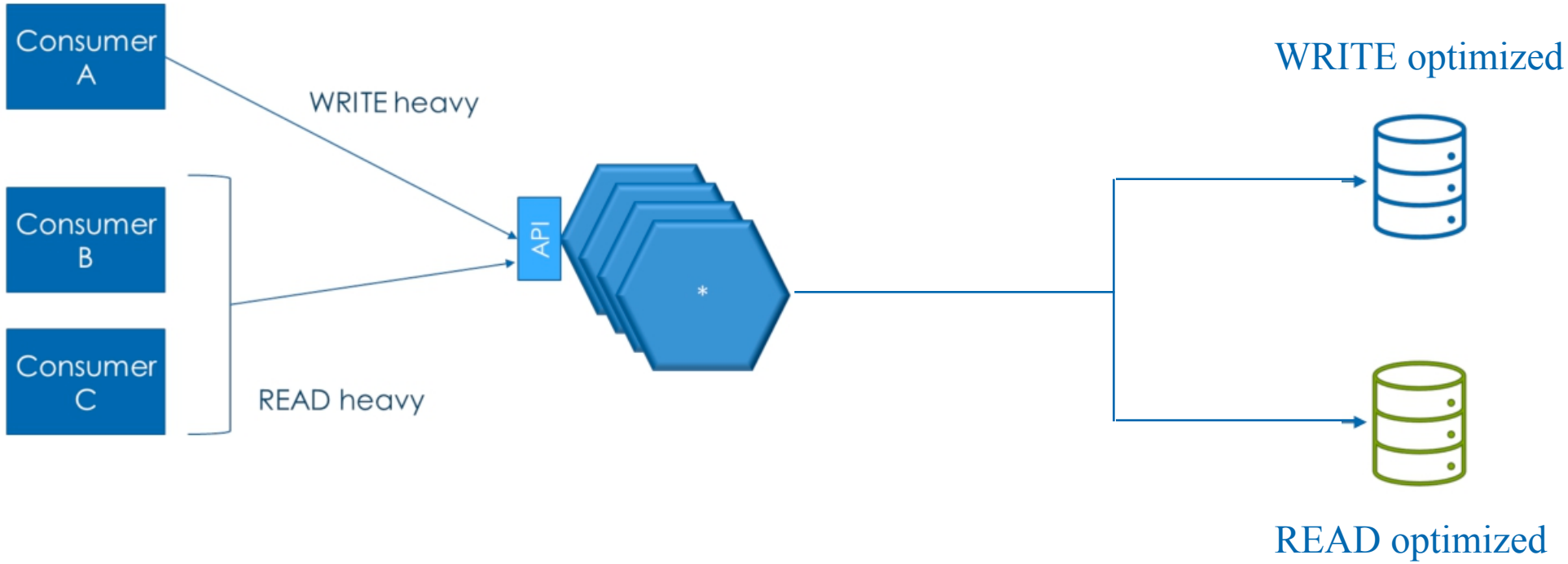
Performance & Data Integrity

Database will become the Bottleneck



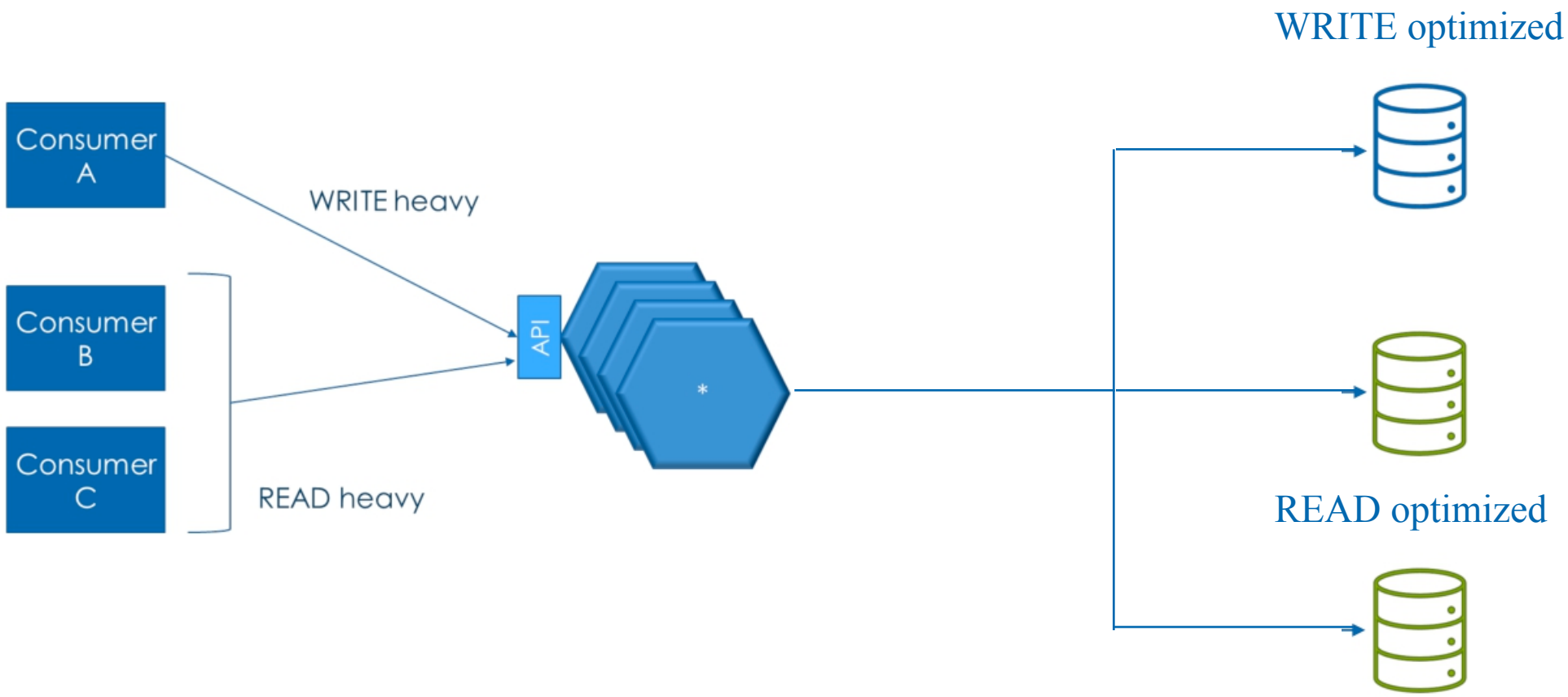
Performance & Data Integrity

Separate the Databases !!!



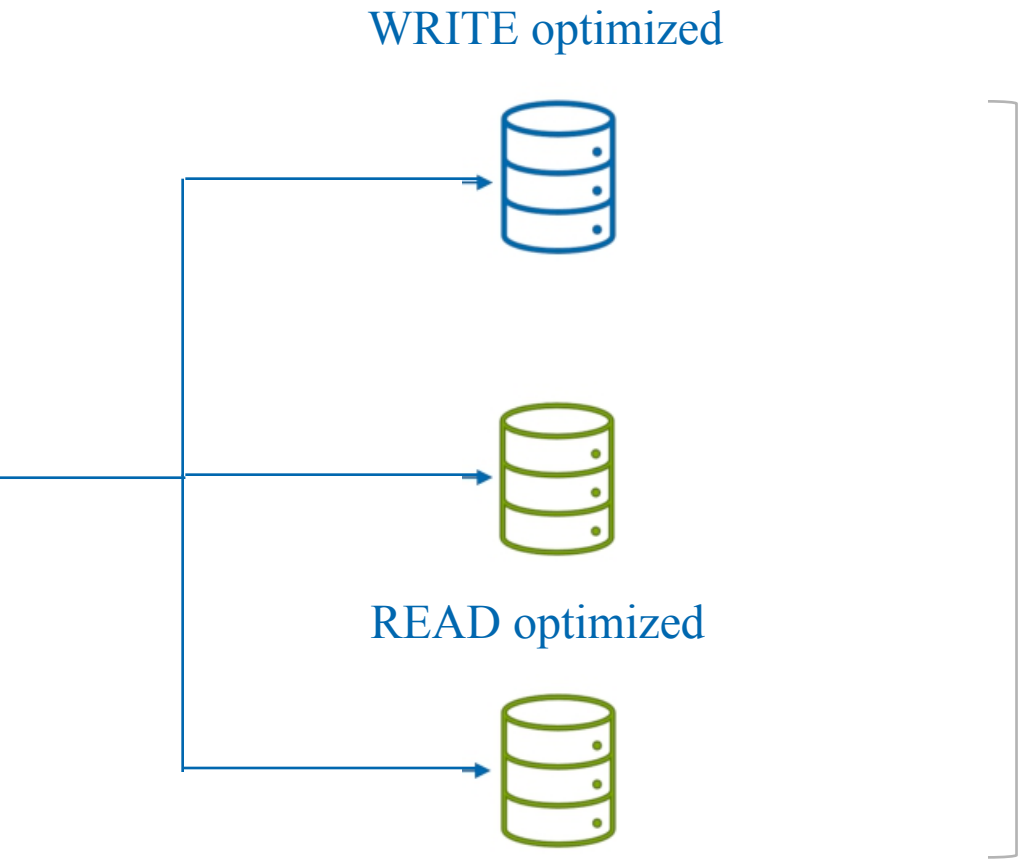
Performance & Data Integrity

Separate the Databases !!!



Performance & Data Integrity

Data Integrity MUST be maintained across Databases



Need for additional patterns ☺

Section objective

1. Command Query Separation pattern (CQS)
2. Command Query Responsibility Segregation (CQRS)
3. Event Sourcing pattern
4. Case Study : Working CQRS for ACME Sales proposal capability (UML, JAVA, RabbitMQ, PostgreSQL, MongoDB)

Commands & Queries

Command Query Separation Principle

1. Commands & Queries
2. Command Query Separation (CQS) Principle
3. Realization of CQS

Domain Object Operation

An Operation is either a command or a query

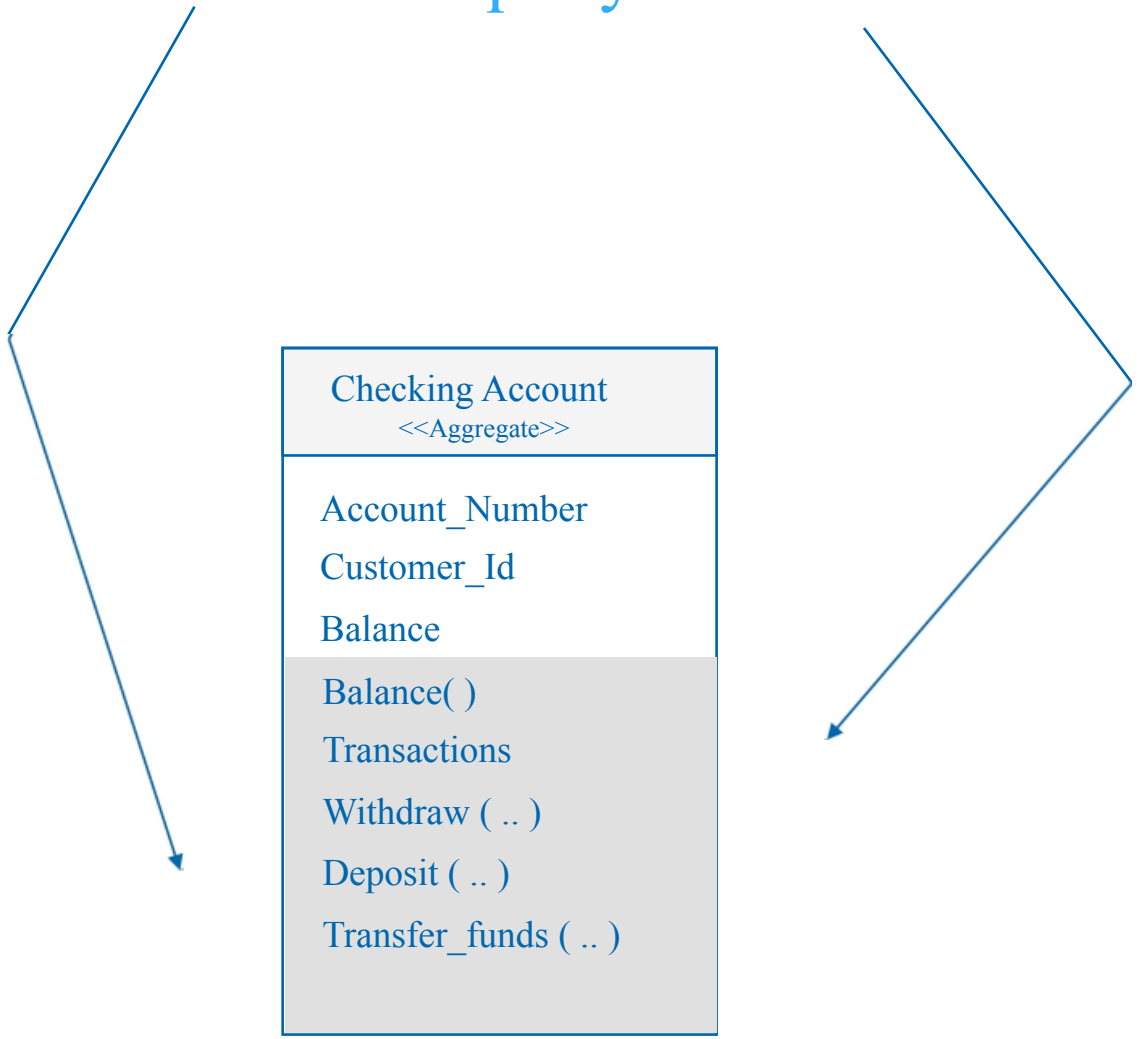
Serve state of aggregate

Change the state of aggregate

A message that tells application | system to do something

Domain Object Operation

An Operation is either a command or a query



Command Query Separation principle a.k.a. CQS pattern

“

Clear demarcation between methods that change the state & those that don't change the state of domain

Commands = Changes the state of object
Queries = Does NOT change the state

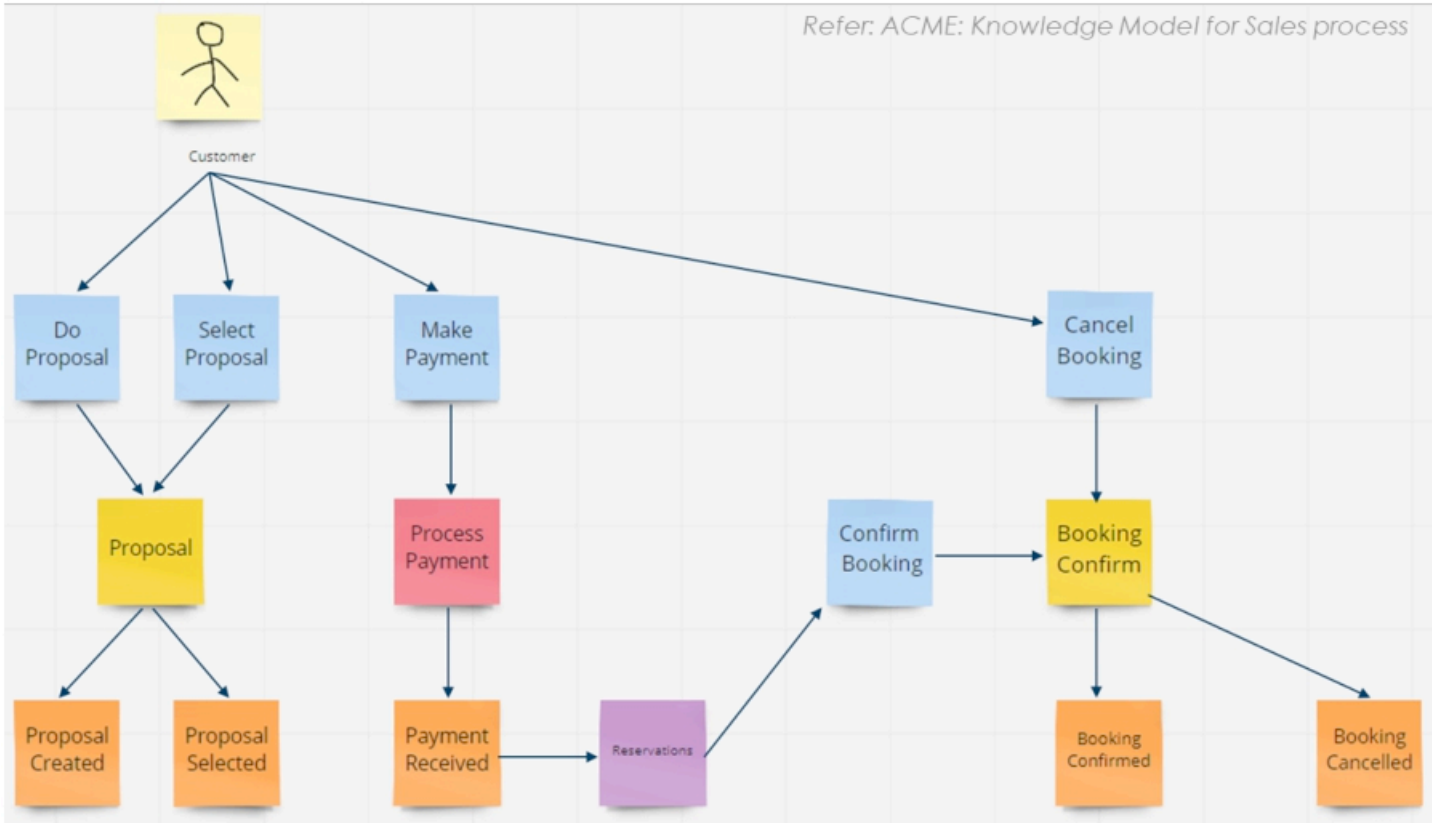
Suggested By: Bertrand Myers

https://en.wikipedia.org/wiki/Command - query_separation

Commands

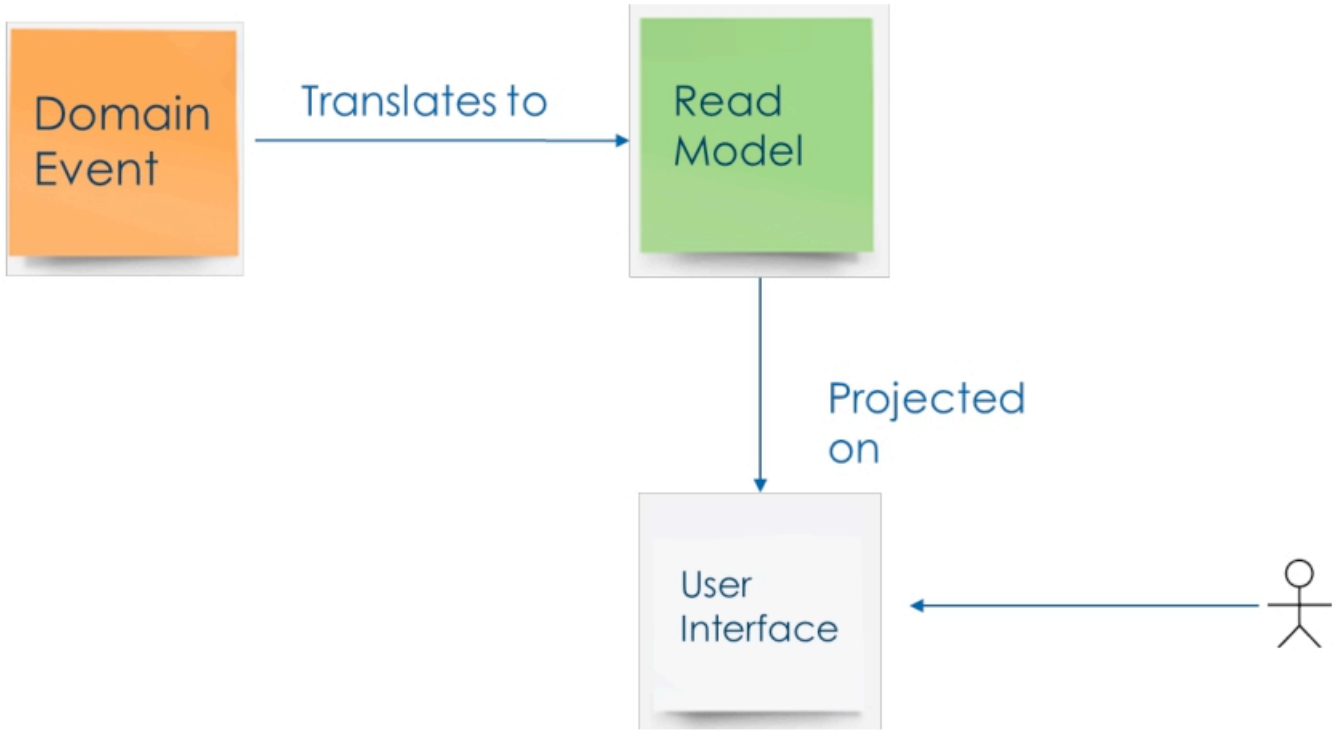
Command

- An Action initiated by user/system/service/time
- This action MUST be carried out
- Named as verbs



Read Models

- Also known as Query Model
- Projected on the UI



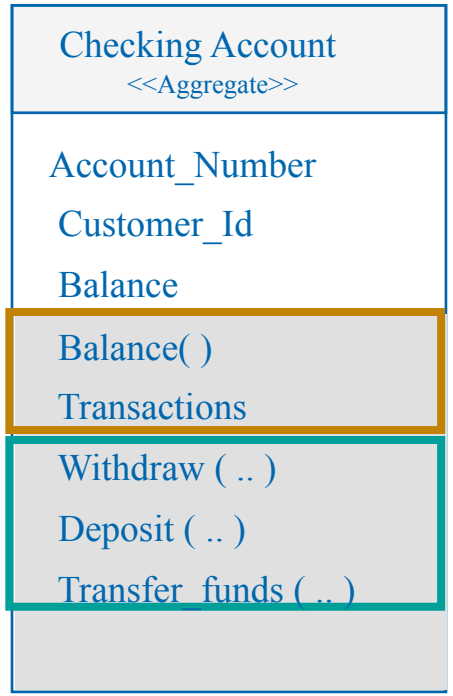
Separation of concerns

Command & Query realized across multiple classes

Complex Logic

Unit of Work

Scalability



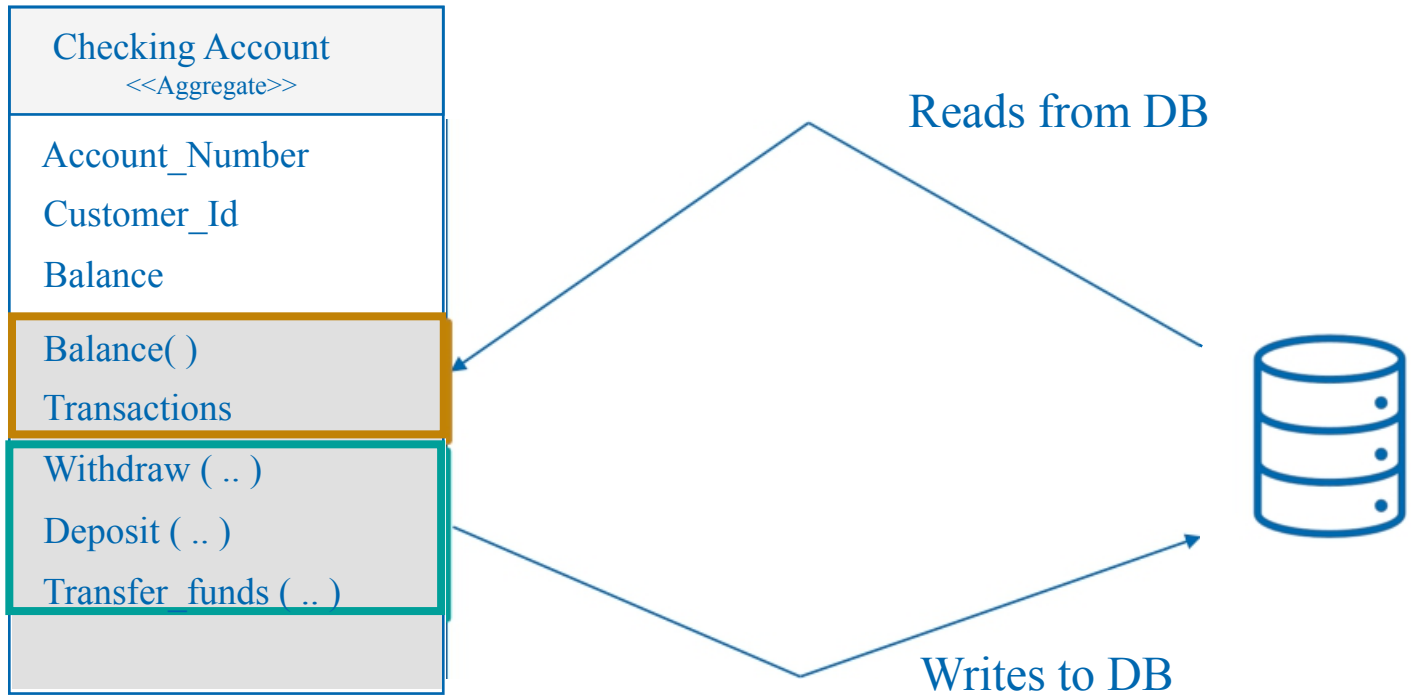
Performance

Flexibility

Shared Datastore

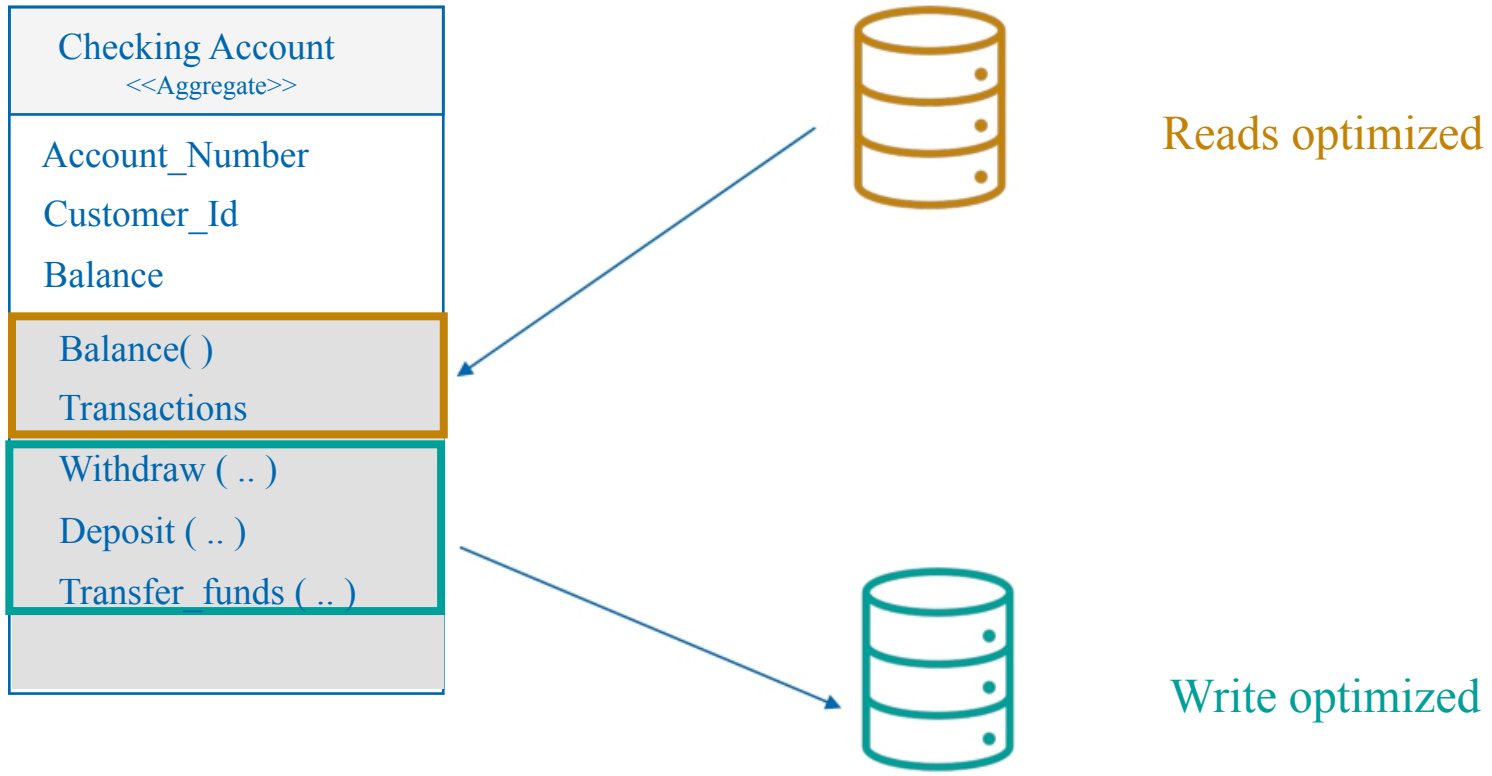
Commands - Writes

Query - Reads



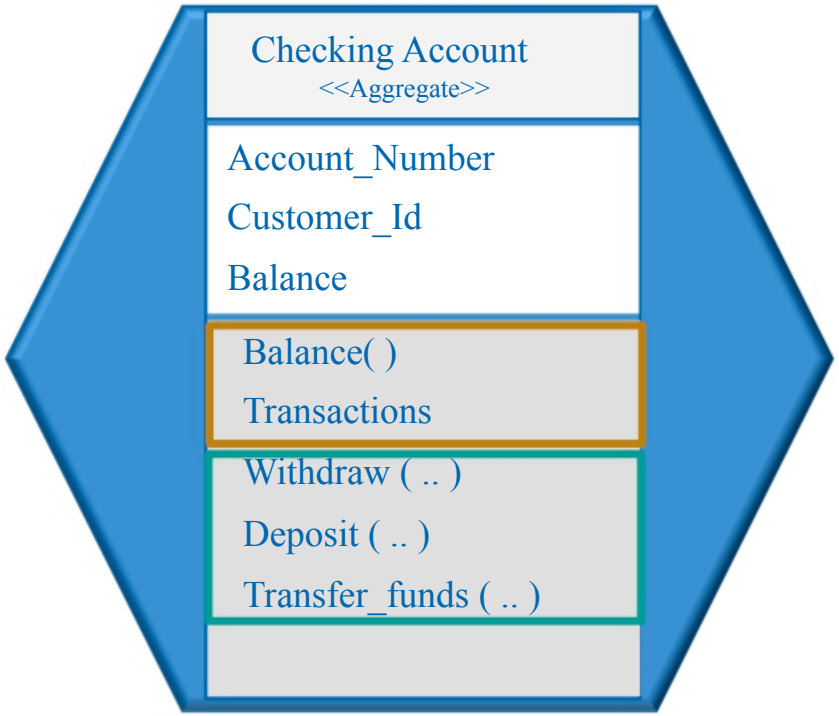
Separate Datastores

May use Datastores optimized for Reads & Writes



Realization of CQS

Commands & Queries in CQS are part of the same component



Quick Review

Commands => Actions or Intent

Query => Serves the domain data

CQS => Separate operations for Command & Query

Use Shared | Separate Datastores depending on needs

Realization : Command & Query

READ & WRITE performance considerations

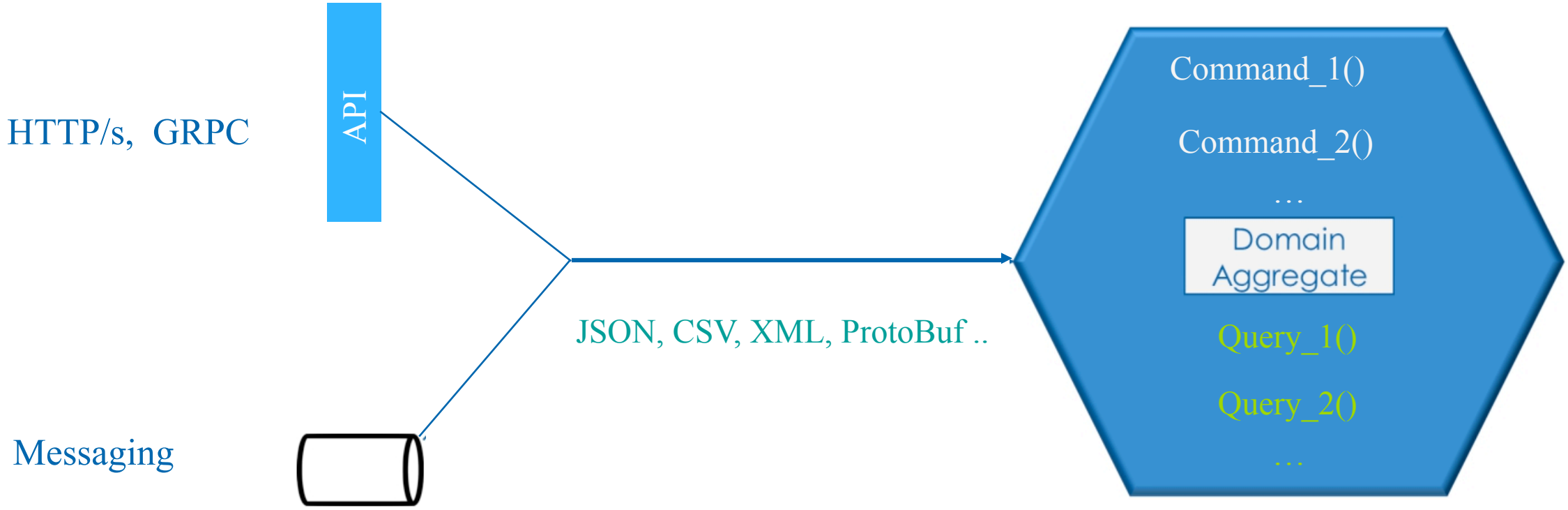
Command | Query Realization

WRITE & READ Performance

Collaborative Domains

Command & Query Interface

Commands & Queries exposed via Synchronous & Asynchronous



DDD : Command & Query models

Data from/to DB translated to/from the common Model

- Commands => WRITEs to the DB

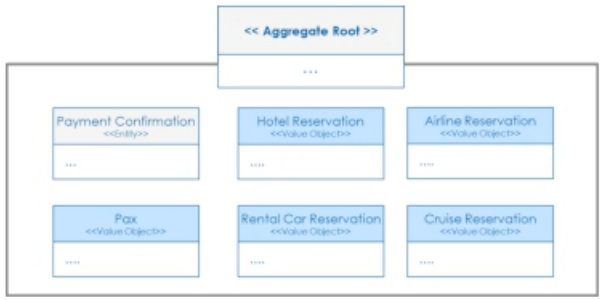
  Referred: WRITE Model

  WRITE Side

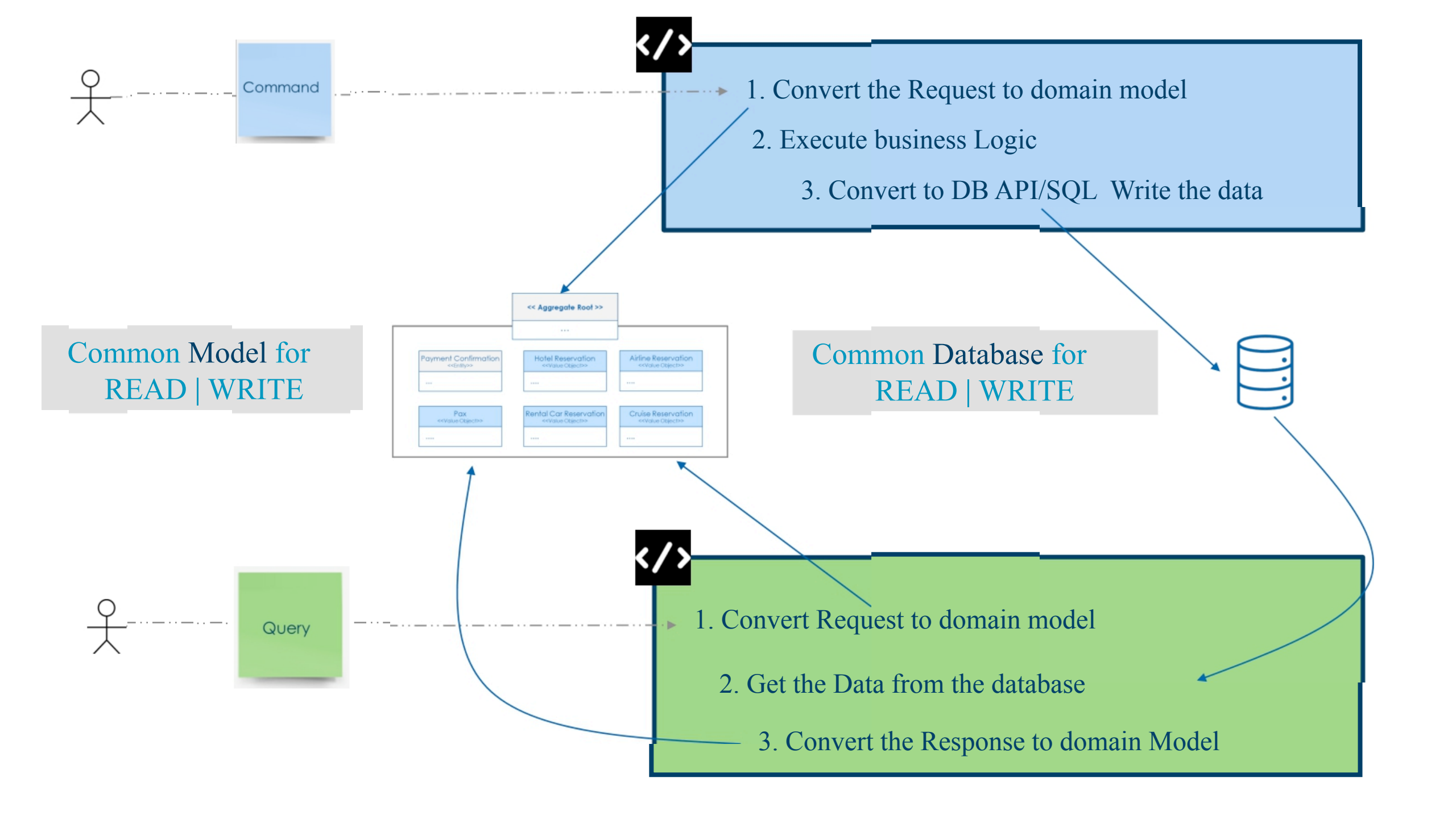
- Queries => READs from the DB

  Referred: READ Model

  READ Side

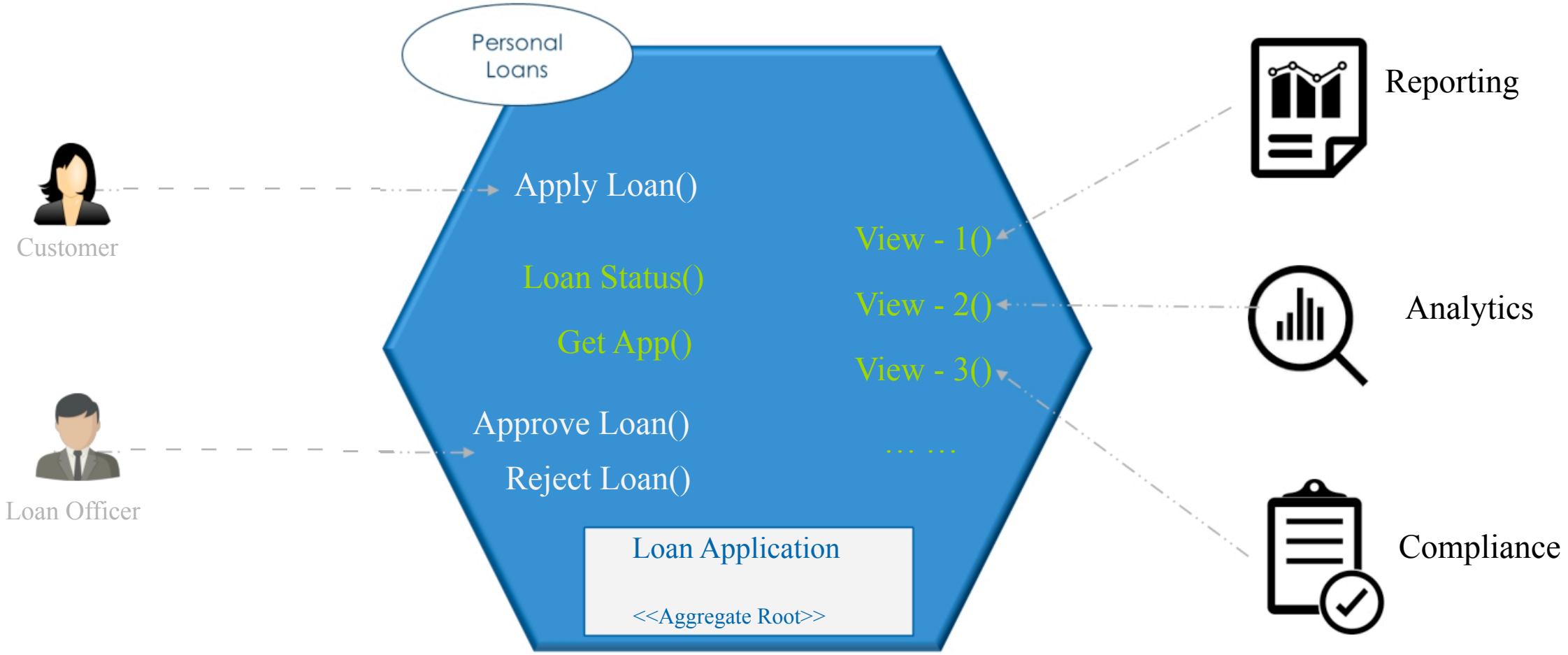


Common Domain Model


Command
1. Convert the Request to domain model
2. Execute business Logic
3. Convert to DB API/SQL Write the data
Common Model for READ | WRITE
<< Aggregate Root >>
Payment Confirmation
Hotel Reservation
Airline Reservation
Pax
Rental Car Reservation
Cruise Reservation
Common Database for READ | WRITE
Query
1. Convert Request to domain model
2. Get the Data from the database
3. Convert the Response to domain Model

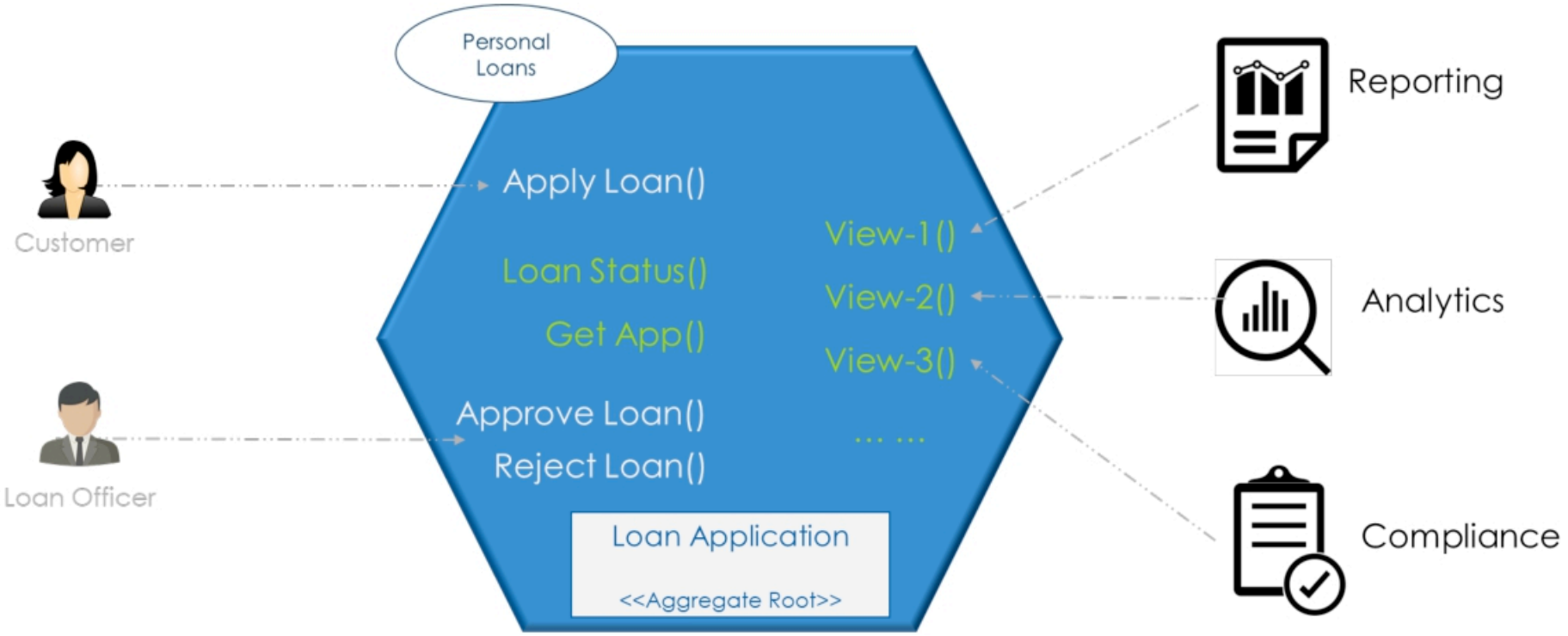
READ model flexibility

Query consumers requests different view/model of the same data



READ model flexibility

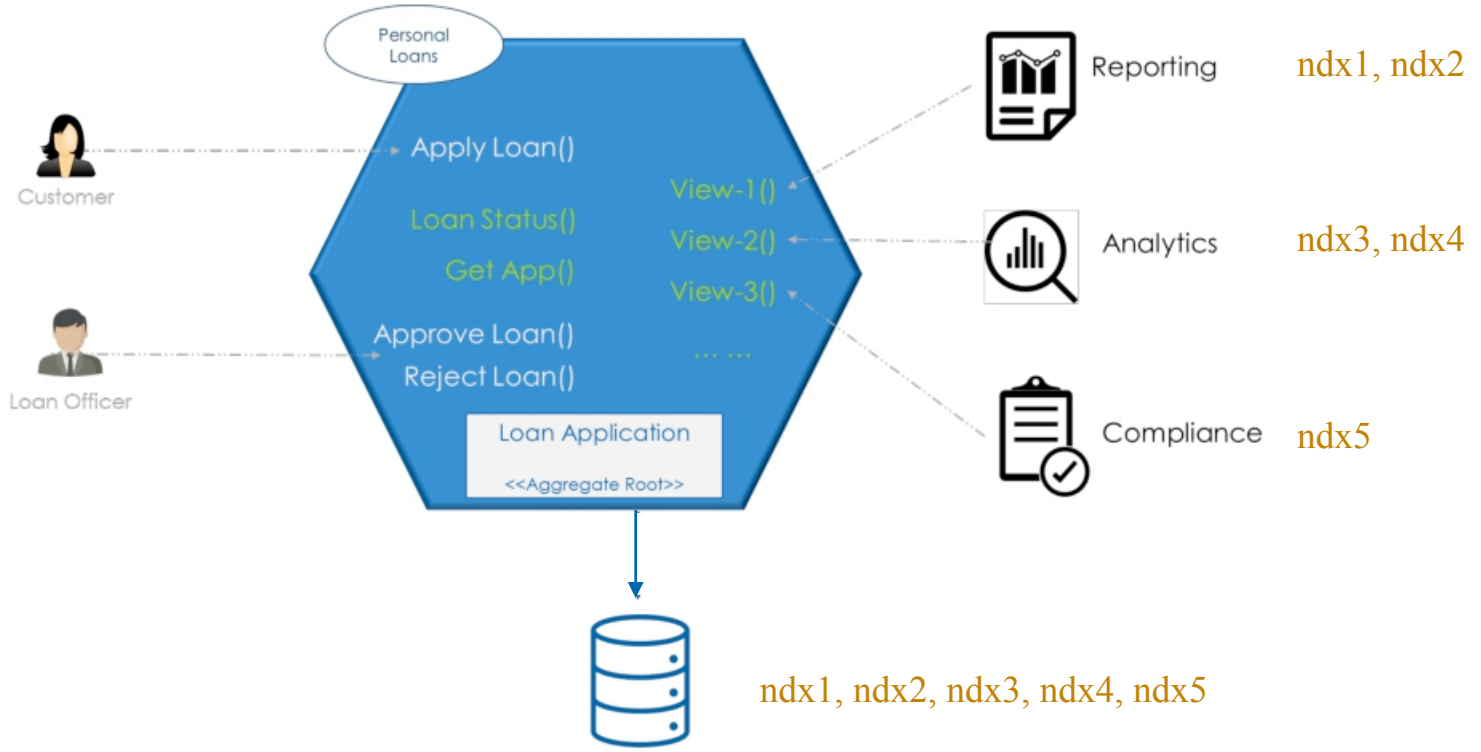
Query consumers requests different view/model of the same data



Requires changes to Microservices code

READ performance

Multiple DB Indexes needed to achieve desired performance



Indexes negatively impacts the performance of WRITEs !!!

Collaborative domain

Multiple actors invoke commands in parallel on the same data

- Online bidding
- Concert tickets reservations
- Trading Systems

Characteristics of Collaborative domains

Low tolerance for performance hit | degradation

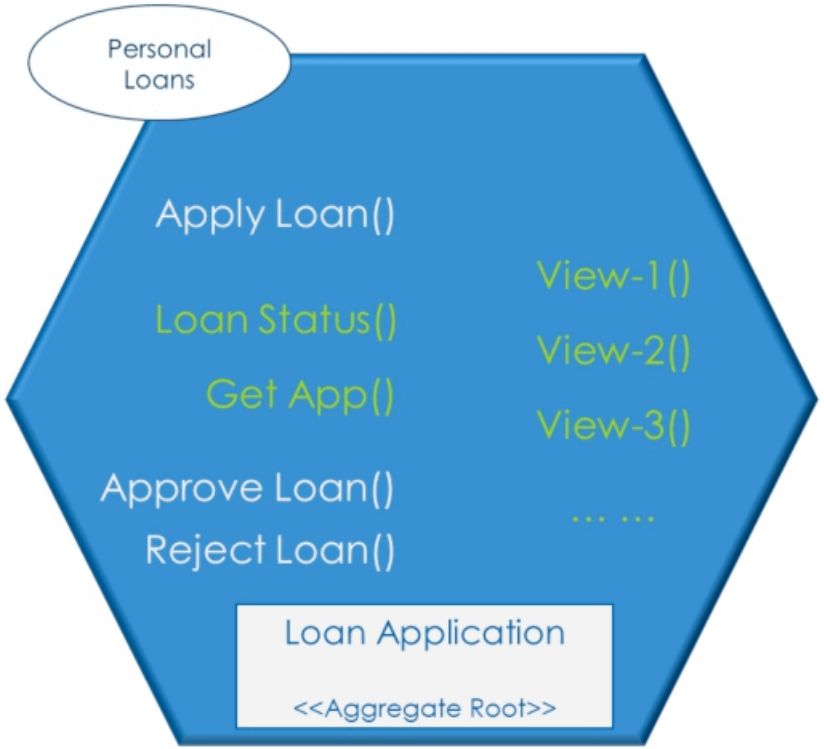
- Commands in such systems have complex business logic
- Commands are implemented with finer levels of granularity
- Business rules | logic may change frequently

WRITE & READ models need to meet these demands !!

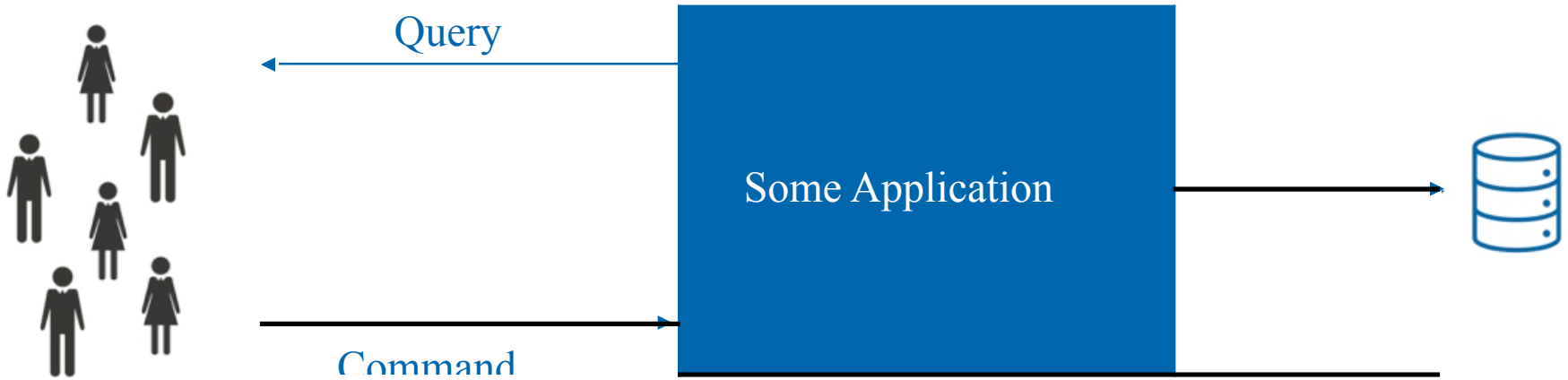
READ vs WRITE

Typical applications | systems have more READ (s) than WRITE (s)

~15% Writes

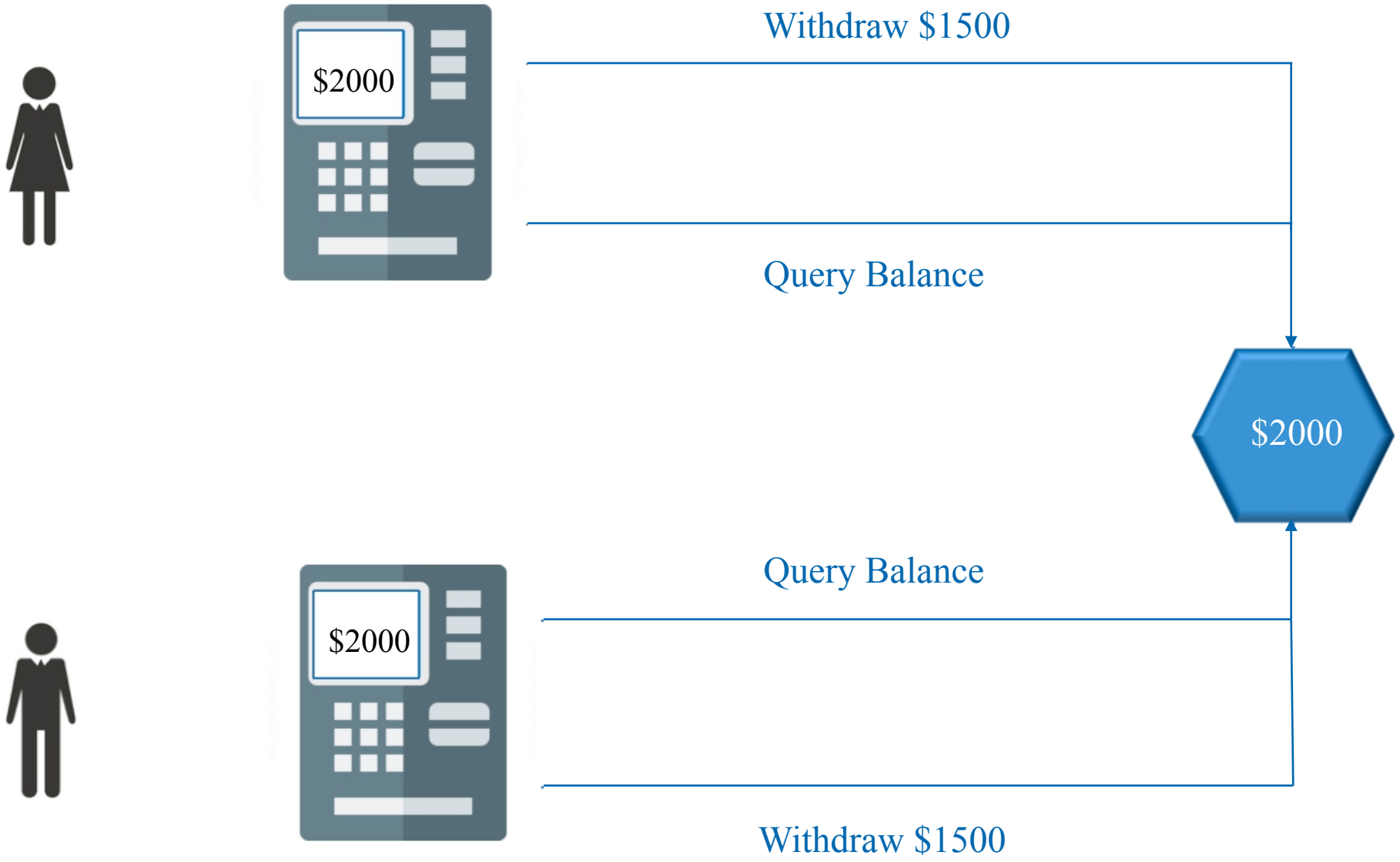


~85% Reads



Can your app deal with Stale query data?

MOST apps can !!!



Query dat is ALREADY stale when it left the microservice service !!!

Quick Review

In DDD common model is used for WRITE/Command & READ/Query

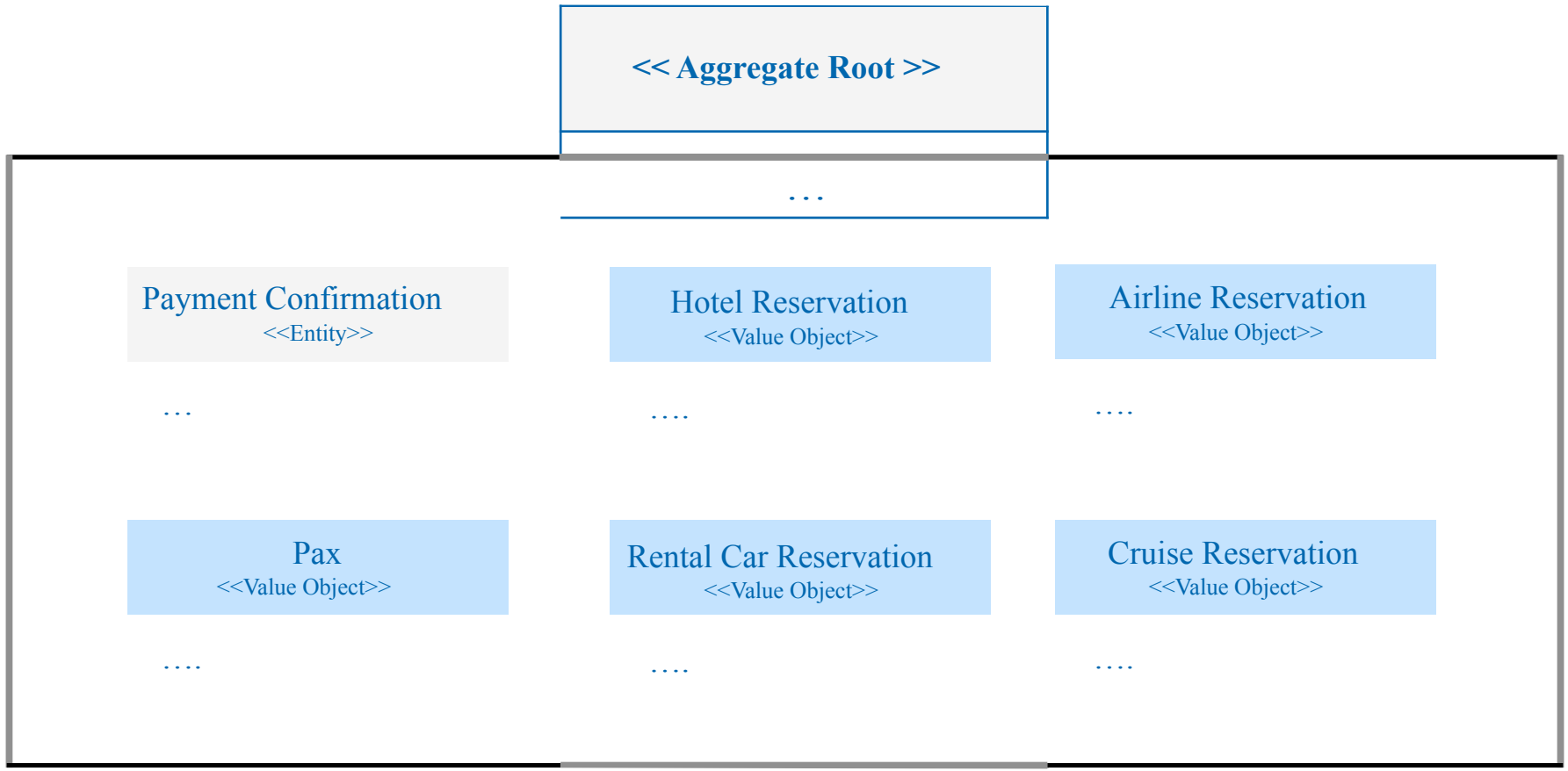
Common to see MULTIPLE Query requirements

- Requires changes to the common model
- Use of indexes leads to negative impact on WRITE performance

Alternate design options needed for Collaborative domains

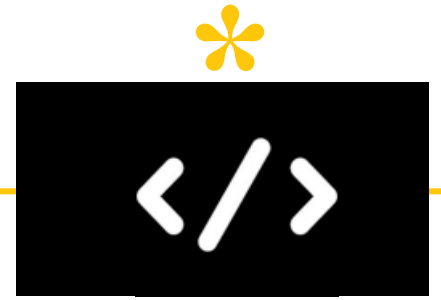
Booking Confirmation

Changes made only via exposed aggregate functions



Repository for JDBC source

Managing the Data in PostgreSQL

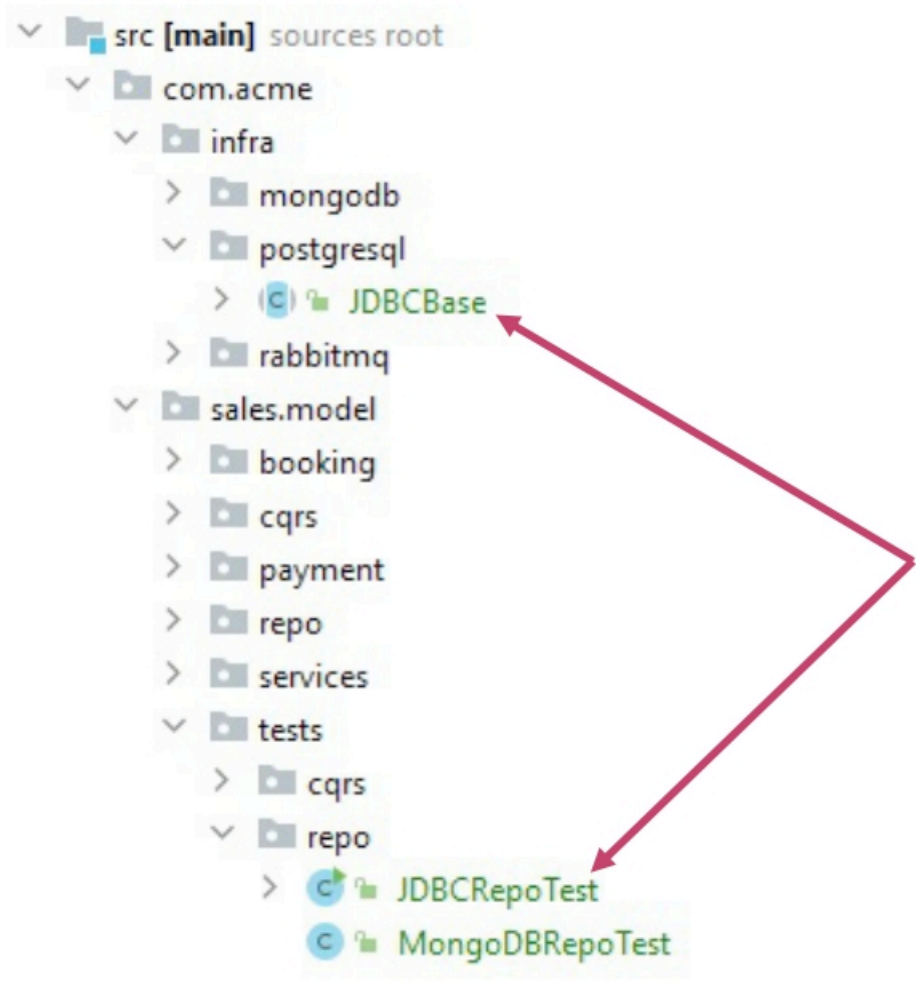


1. Create an instance of PostgreSQL
2. JDBCBase , JDBCRepoTest
3. Testing

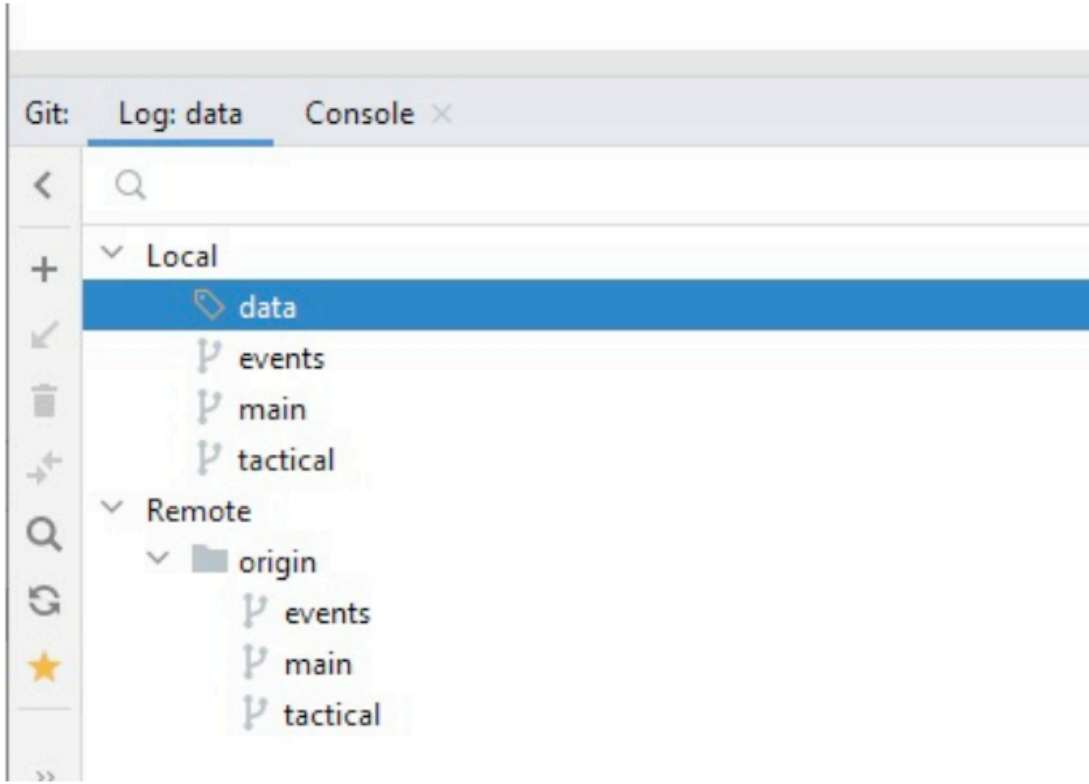
PS: Intent is NOT to teach JDBC

Code

Checkout the branch : data



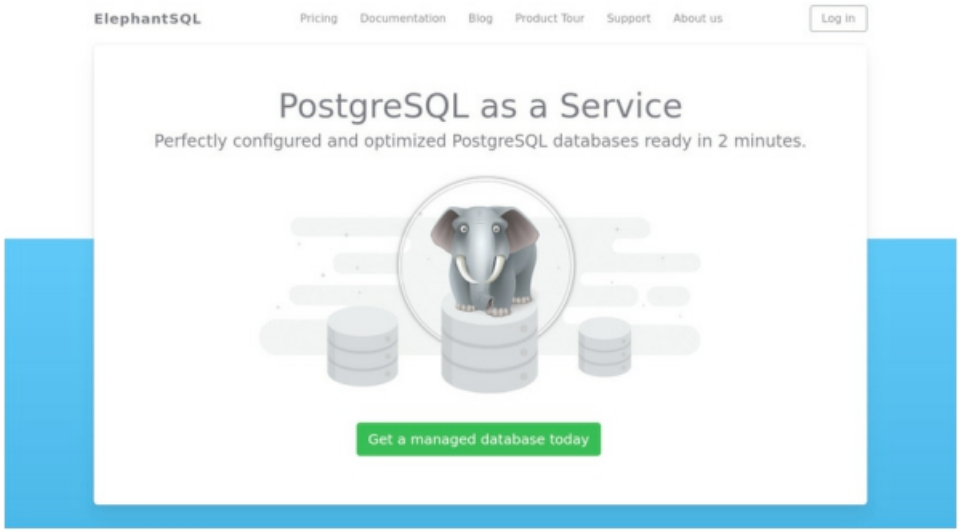
- Pre - Requisites: JDBC & JSON
- Instructions in README.md



PostgreSQL Database

Database for ACME Sales management

✓ Online FREE instance of the Database

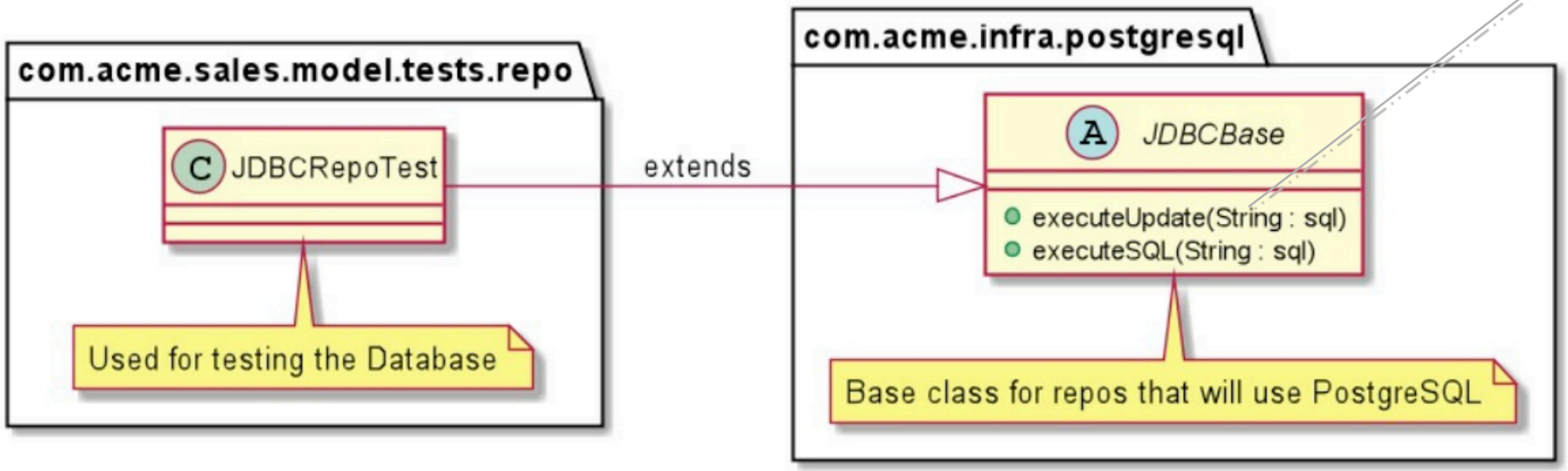


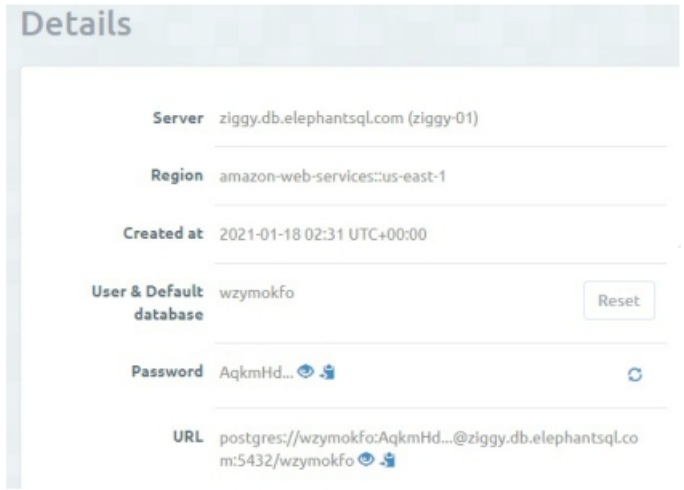
- Install PostgreSQL locally | cloud

Testing from Java

uml / db/ jdbcbase.class.puml

Execute SQL functions return result in JSON format



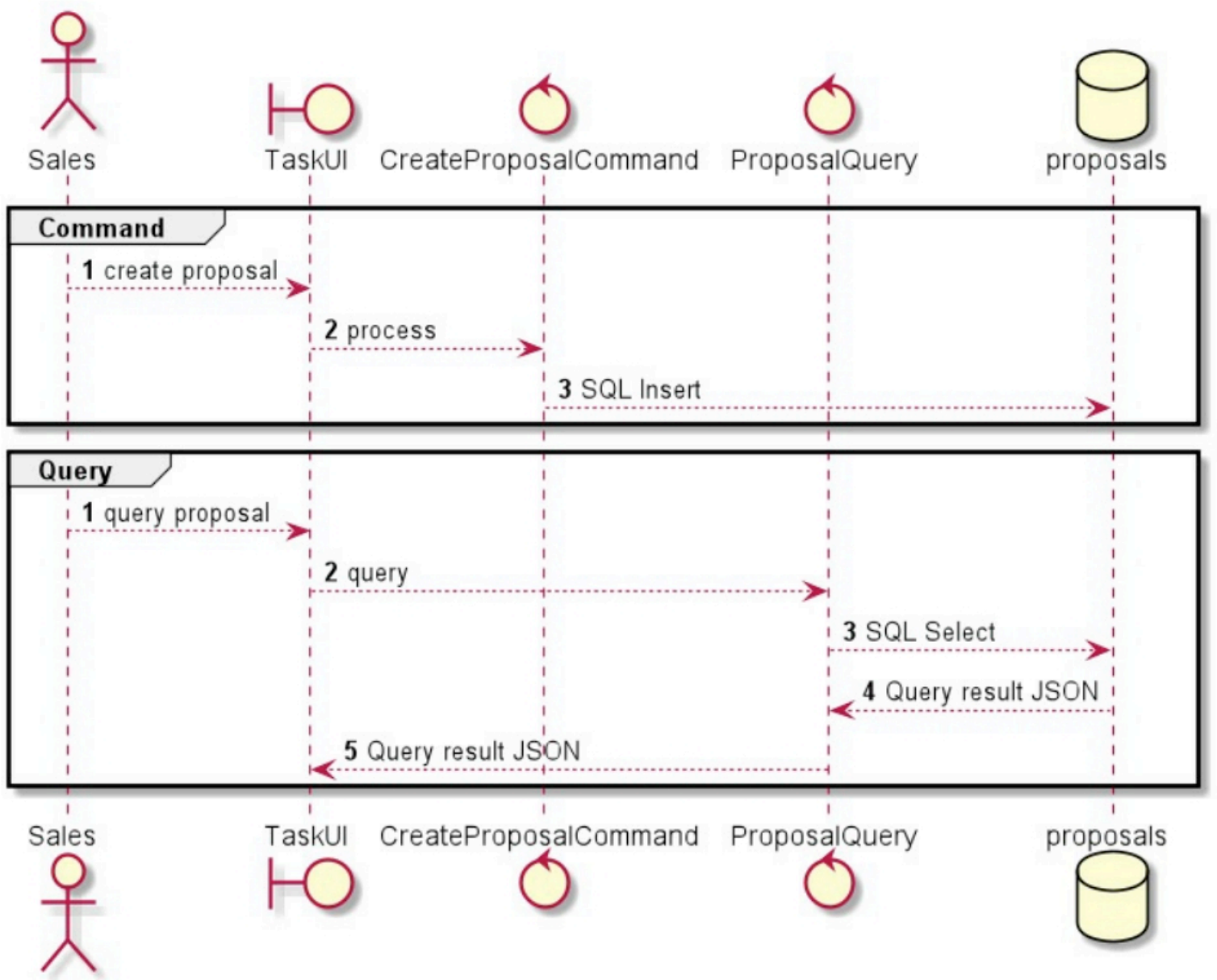


MUST setup the PostgreSQL connection parameters and the credentials in JDBCBase class

OTHERWISE you would get Errors/ SQLException

Command - Query Sequence

uml / cqrs /commandquery.v1.sequence.puml



Build & Test Command and Query

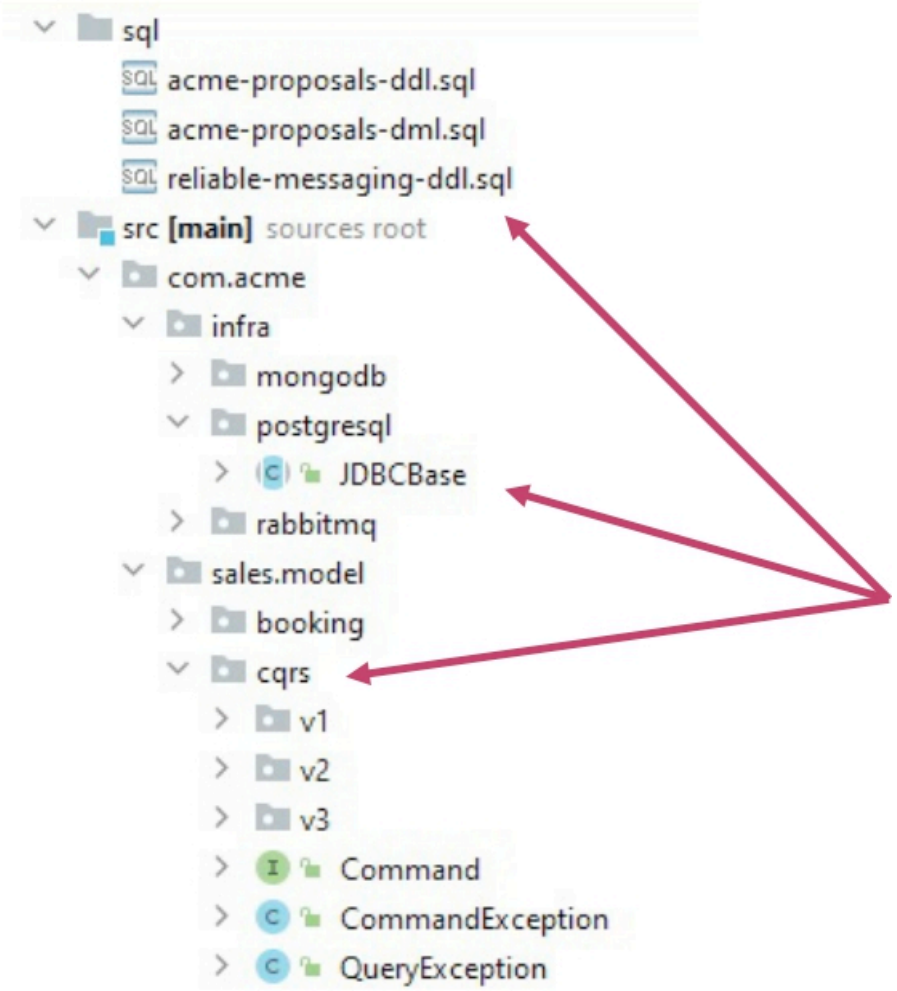
Command will write to DB; Query will read from DB

1. Need for a PoC
2. Classes for implementing the CQS
3. Test the Command & Query

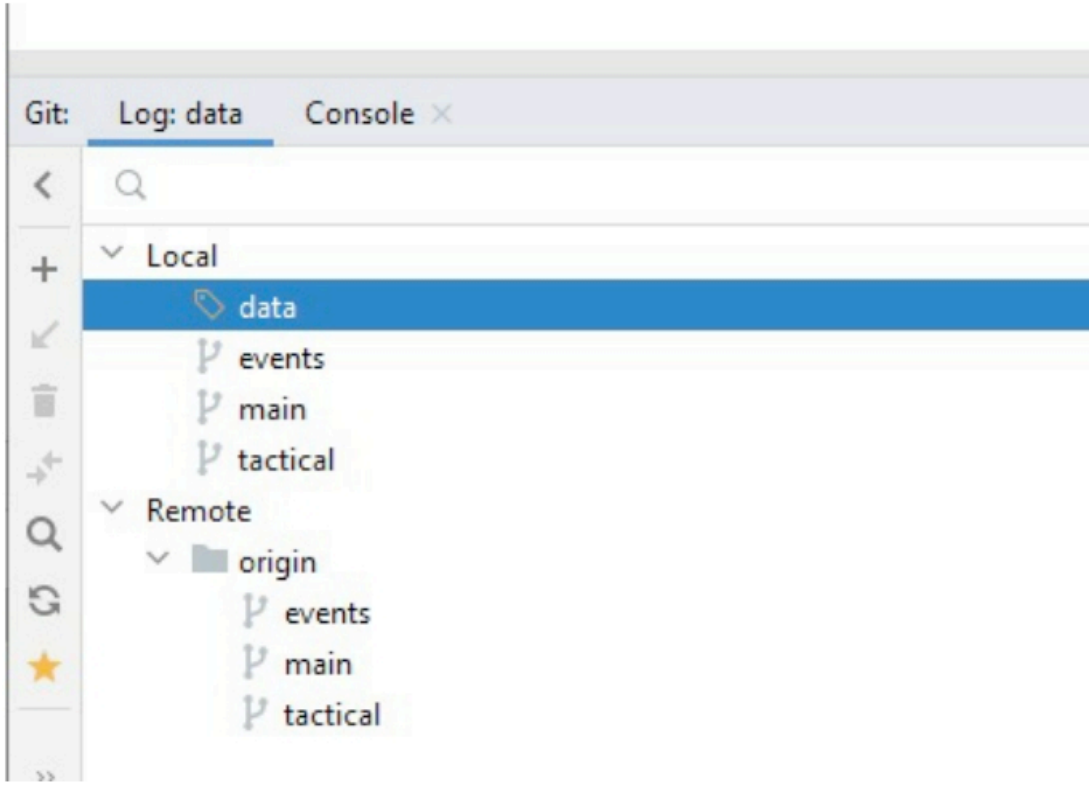
Note: Use the PostgreSQL Database setup in previous lecture

Code

Checkout the branch : data



- Pre - Requisites: JDBC & JSON
- Instructions in README.md



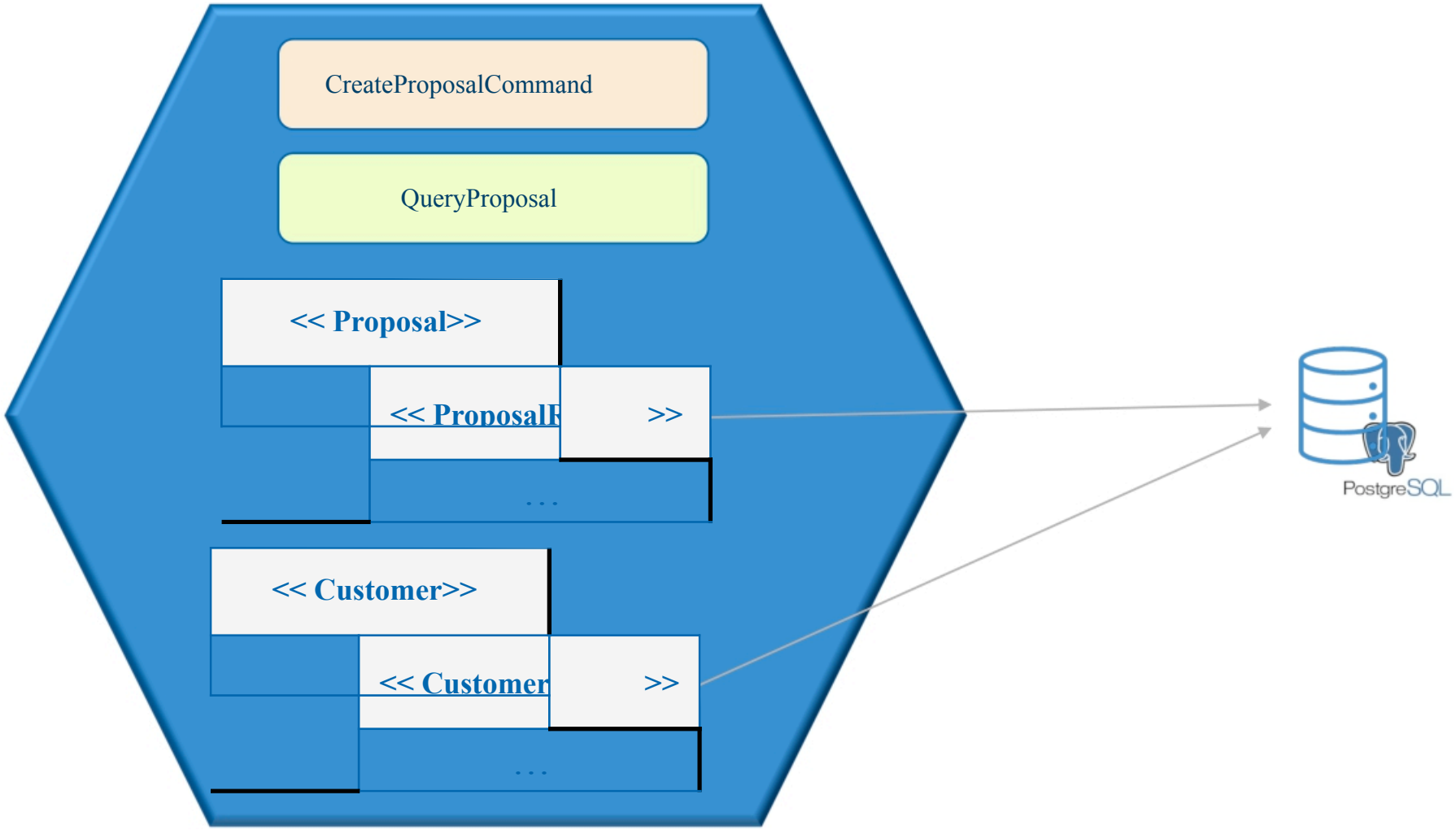
Need a working model to understand CQS

Create the command & query for proposals

Create the proposal for customer in Database

Start with some common queries but make it flexible for future queries

Implementation components



Setup the DB tables : customer table

sql/acme-proposals-ddl.sql

```
CREATE TABLE customers (
    customer_id     INT  generated by default as identity,
    email_address   varchar(255 ) not null,
    fname           varchar(64 ) not null,
    mname           varchar(64 ) not null,
    lname           varchar(64 ) not null,
    phone           varchar(10) not null,
    PRIMARY   KEY(customer_id)
);
```

Setup the DB tables : proposals Table

sql/acme-proposals-ddl.sql

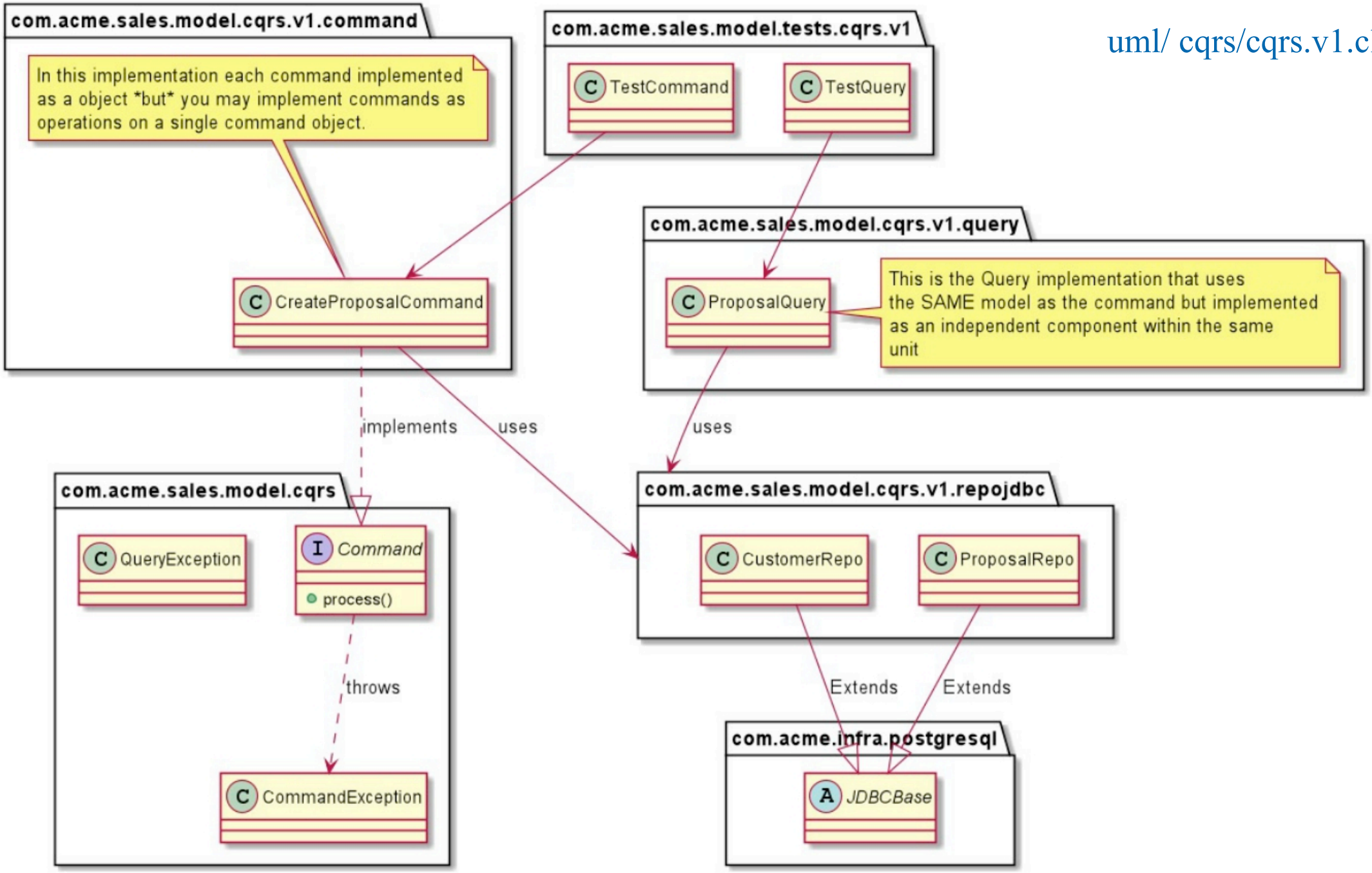
```
CREATE TABLE proposals (
    proposal_id     INT  generated by default as identity,
    customer_id     INT not null,
    package_id      varchar(64) not null,
    pax_0           varchar(64) not null,
    pax_age_0       INT  not null,
    pax_1           varchar(64),
    pax_age_1       INT,
    pax_2           varchar(64),
    pax_age_2       INT,
    pax_3           varchar(64),
    pax_age_3       INT,
    pax_4           varchar(64),
    pax_age_4       INT,
    PRIMARY   KEY(proposal_id) ,
    CONSTRAINT fk_customer
        FOREIGN KEY(customer_id)
            REFERENCES customers(customer_id)
);
```

Create & Populate the tables

1. Create tables — sql/acme-proposals-ddl.sql

2. Populate tables — sql/acme-proposals-dml.sql

uml/ cqrs/cqrs.v1.class.puml



CQRS

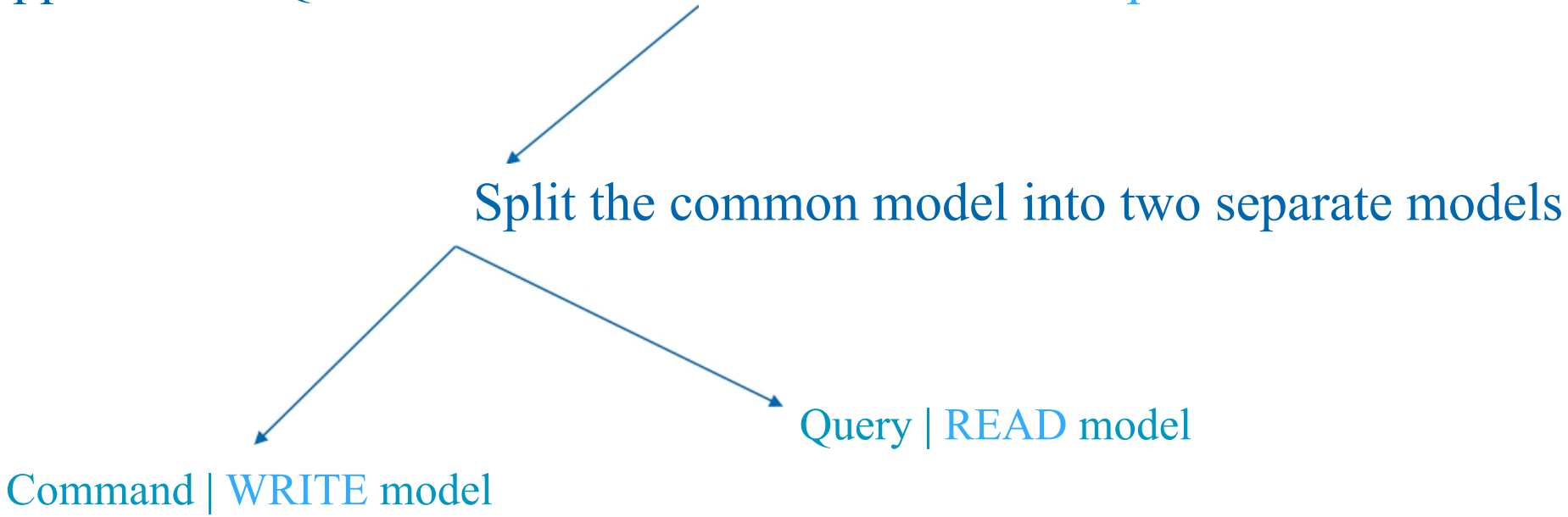
Command Query Responsibility Segregation

1. Define CQRS
2. Benefits
3. Realization

CQRS

Command Query Responsibility Segregation

- Applies the CQS idea to Domain model rather than operations



Suggested By: Greg Young

https://martinfowler.com/bliki/CQRS.html

CQRS

Suggests the use of separate READ and WRITE Models for the realization of Collaborative domains

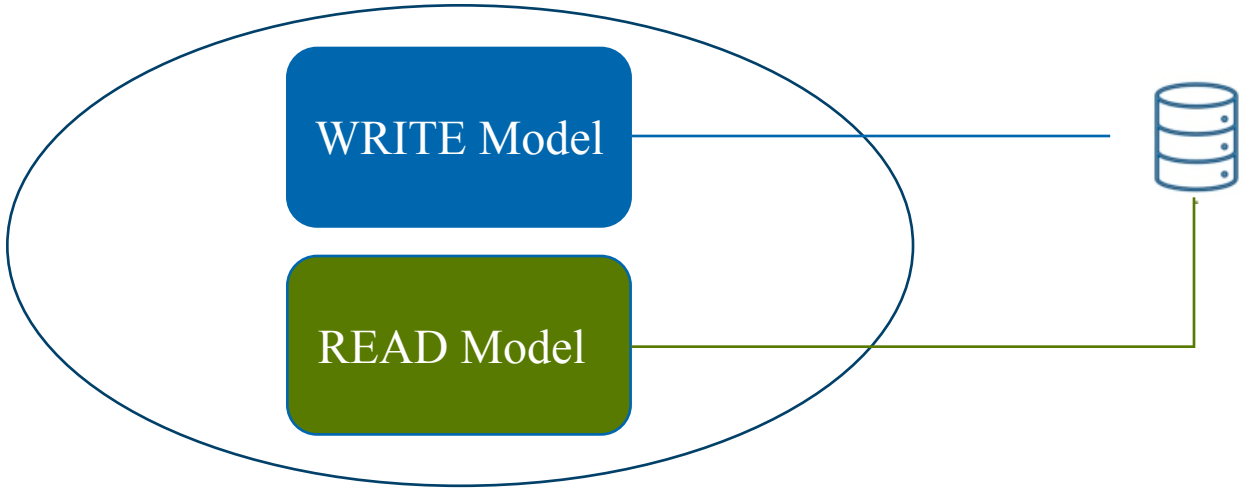
- Finer Granularity
- Frequent changes to Business Rules

- Meet different READ requirements
- High performance

Common Datastore for READ & WRITE

READ | WRITE models may use the same instance of Datastore

Creation of multiple indexes will impact WRITE



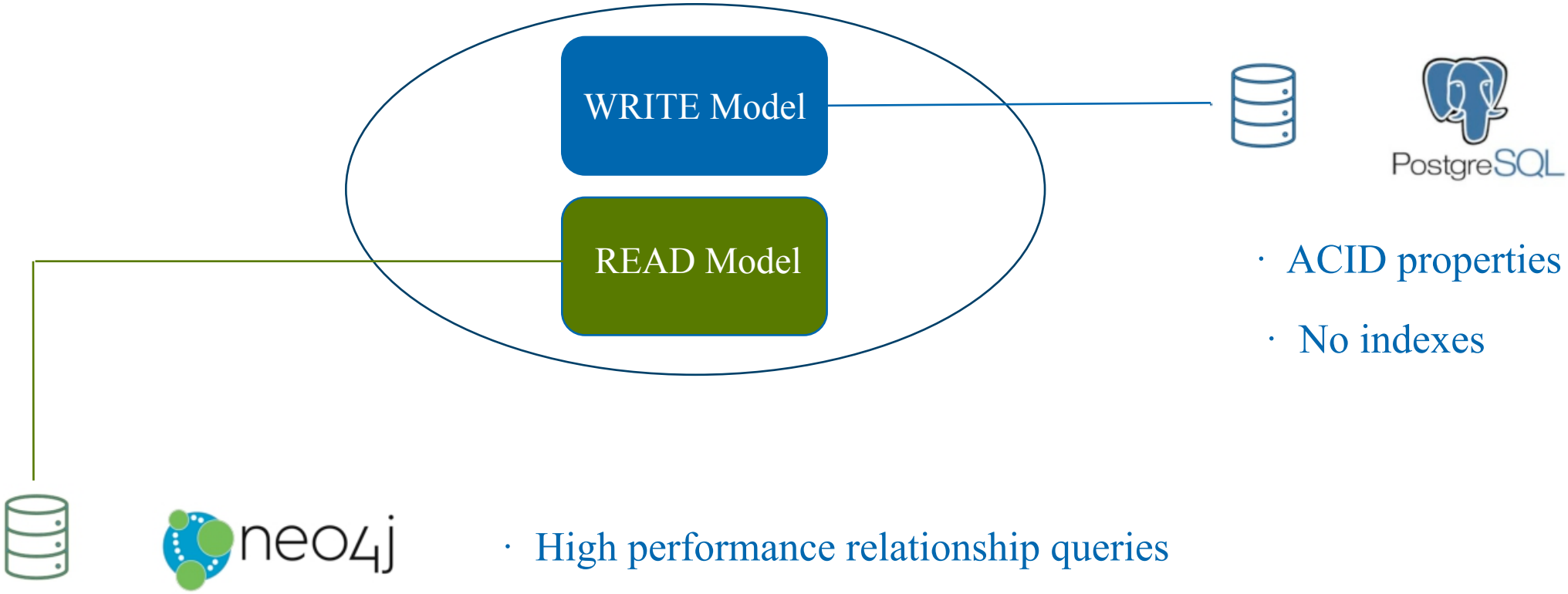
Removal of indexes will impact QUERY performance

Loss of ability to optimize the DB for READ | WRITE

Independent Datastores & Polyglot persistence

READ | WRITE models may use independent Datastores

- May use different Database technologies

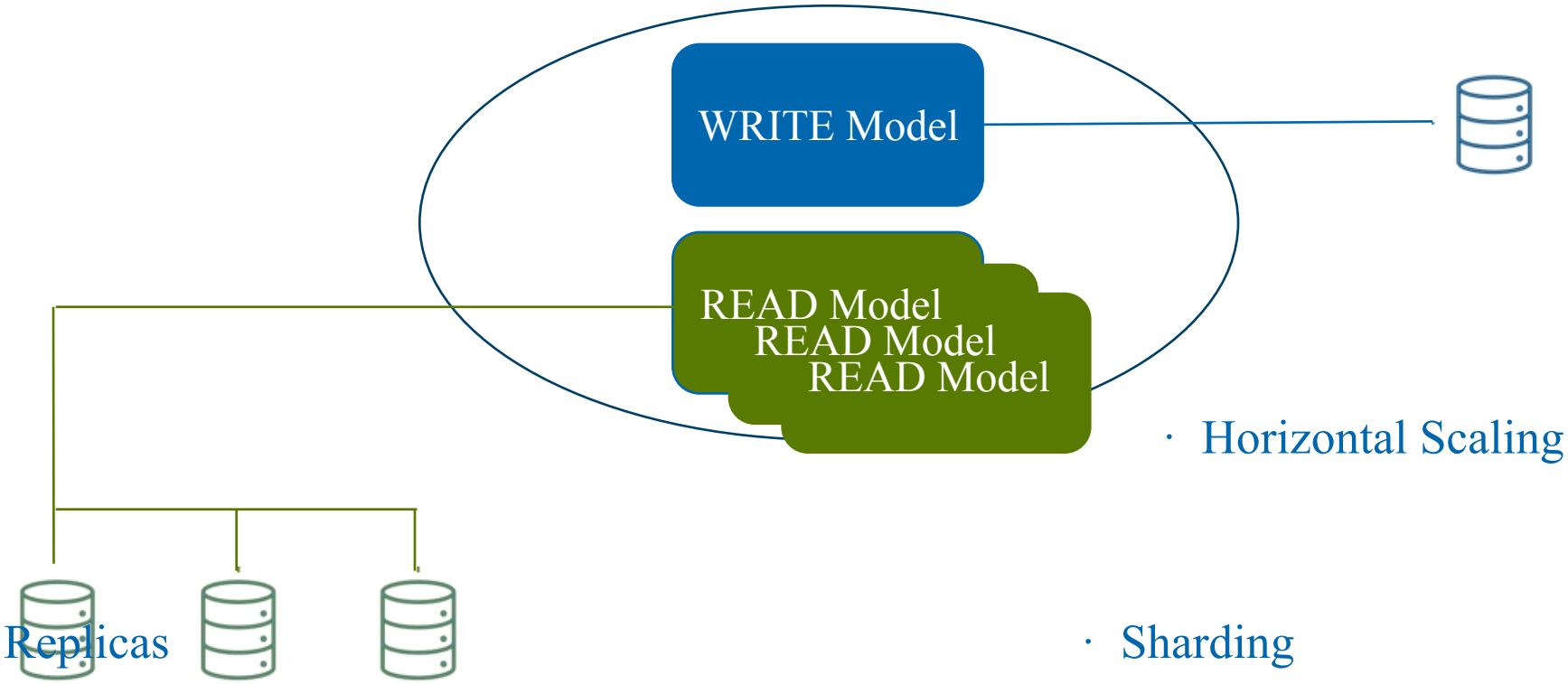


- ACID properties
- No indexes
- High performance relationship queries

Control on Scaling

READ | WRITE side may scale independently

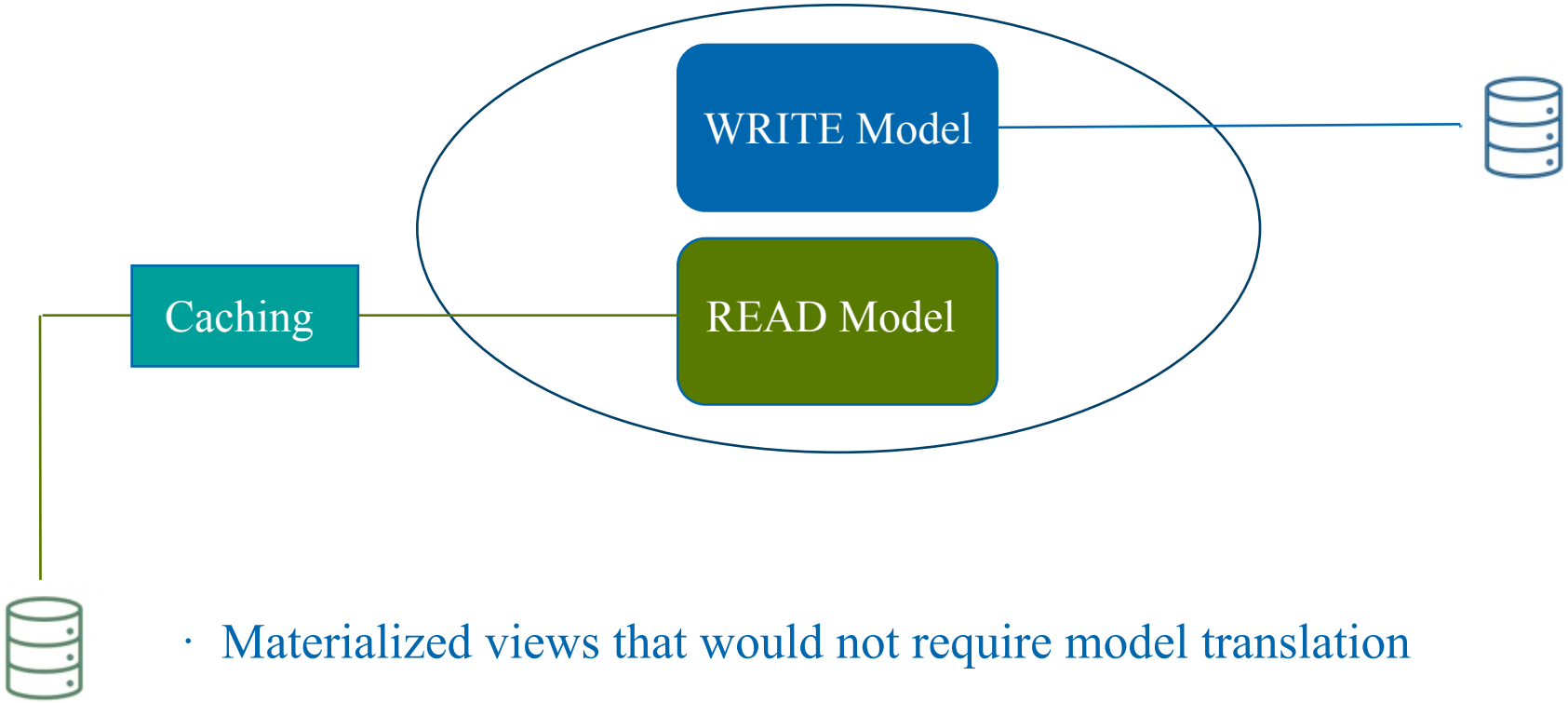
- Example: Read heavy application



- Horizontal Scaling
- Read Replicas
- Sharding

READ Performance considerations

Typically, there are more READ than WRITE (~ 85% Reads)

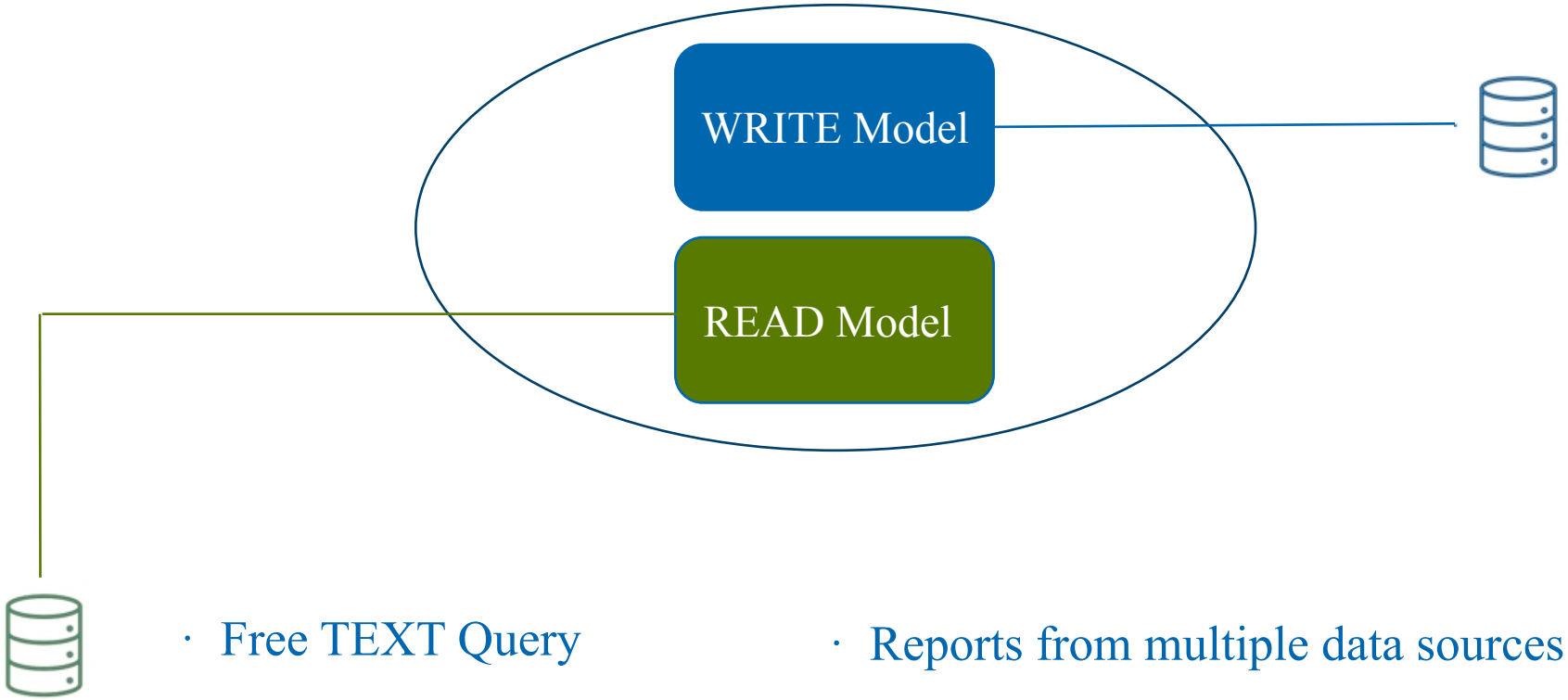


- Materialized views that would not require model translation

READ Performance considerations

Typically, there are more READ than WRITE (~ 85% Reads)

- Example: Multiple query requirements

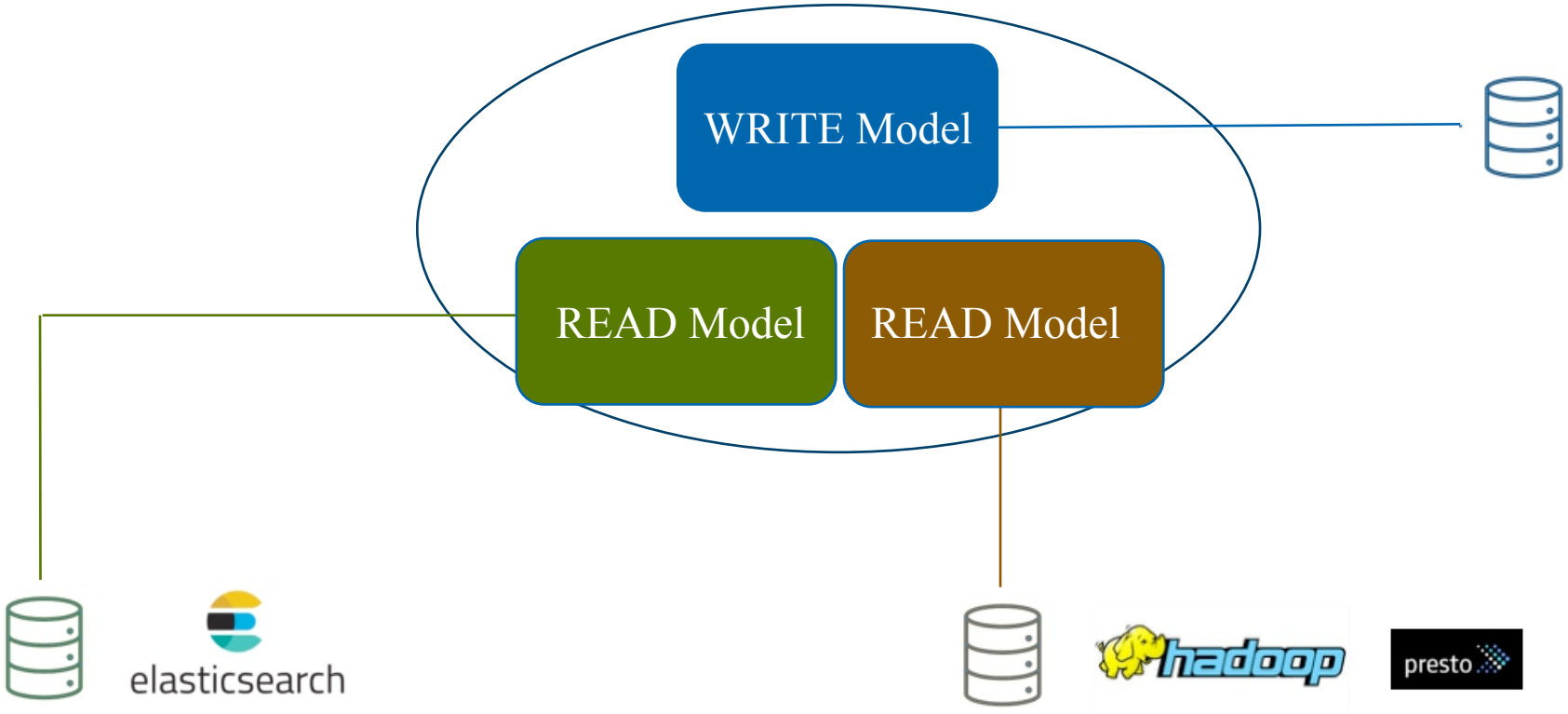


- Free TEXT Query
- Reports from multiple data sources

READ Performance considerations

Typically, there are more READ than WRITE (~ 85% Reads)

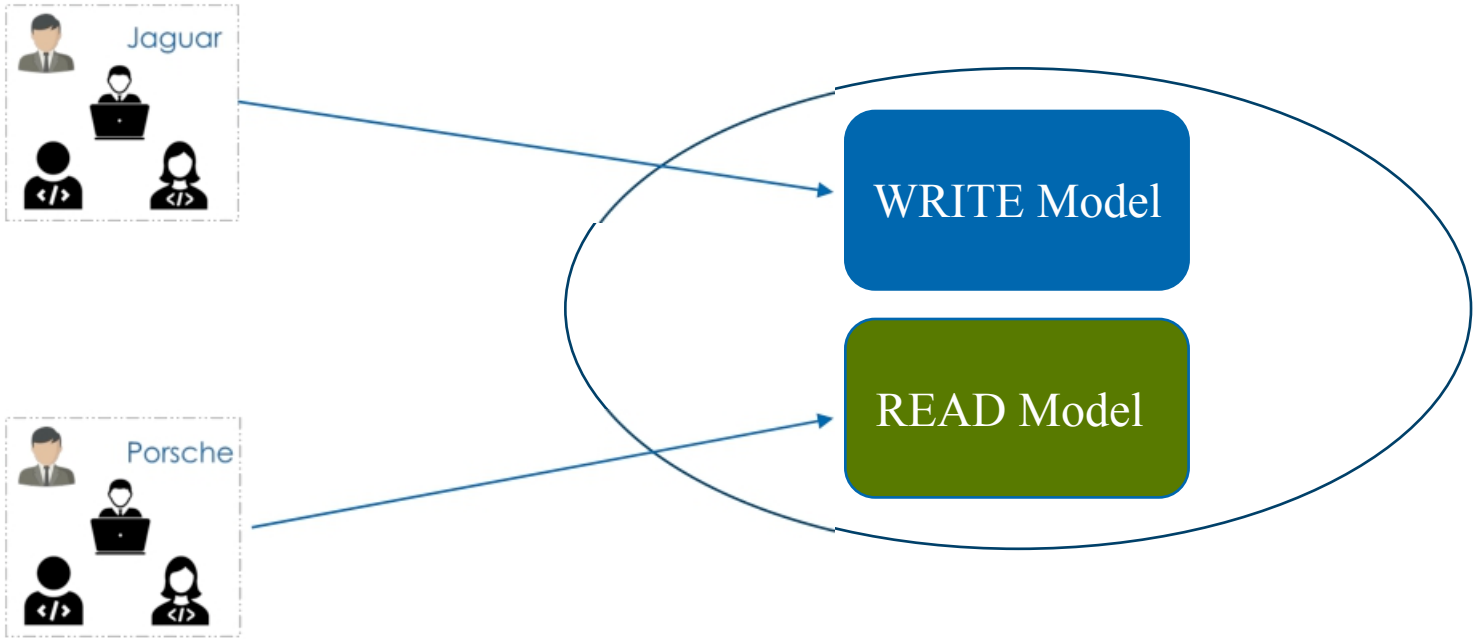
- Example: Multiple query requirements



- Free TEXT Query
- Reports from multiple data sources

Independent Change Management

READ | WRITE models may be managed by different teams



- Independent code bases
- Independent changes
- Independent deployments

Collaborative | Dynamic apps require frequent WRITE changes

Misconception

In DDD you MUST always use CQRS

- Considerations
  - What is the advantage
  - HIGH Cost of solution
  - More moving parts | components

Depends on the Requirements | Use Case | Objective

Quick Review

Consider using CQRS for collaborative domains

- READ | WRITE independently control:
  - Performance
  - Scalability
  - Changes

CQRS : Data Replication

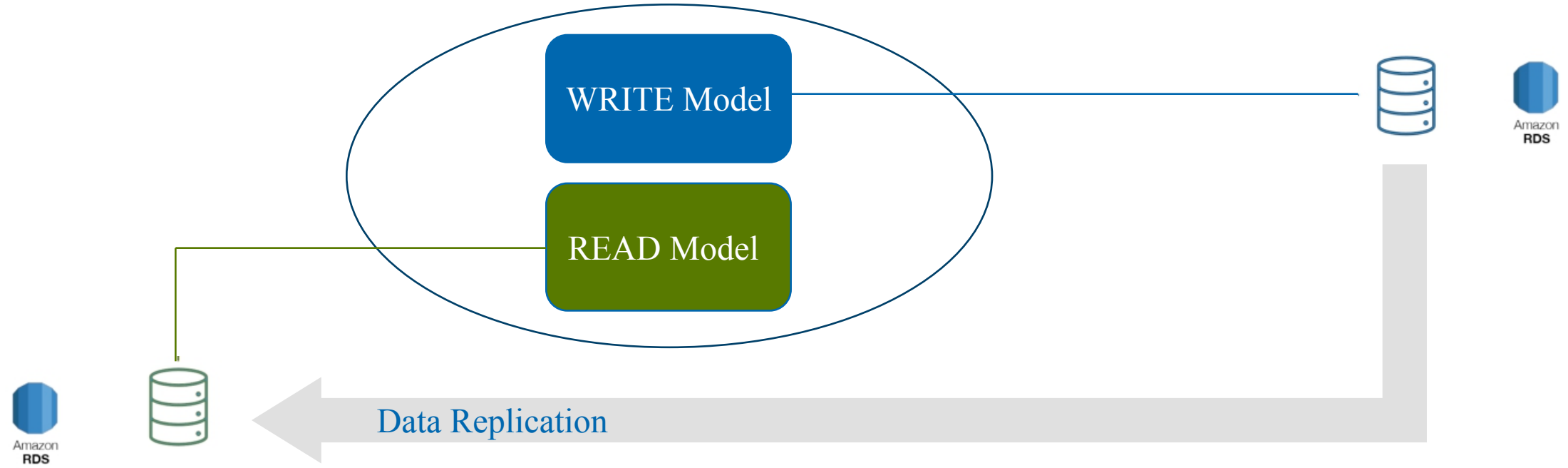
Data needs to be replicated between WRITE & READ side

1. Need for Replication
2. Synchronous | Asynchronous Replication
3. Data Replication options

Data Replication Considerations

NEEDED for independent READ | WRITE Data stores

Synchronous Replication

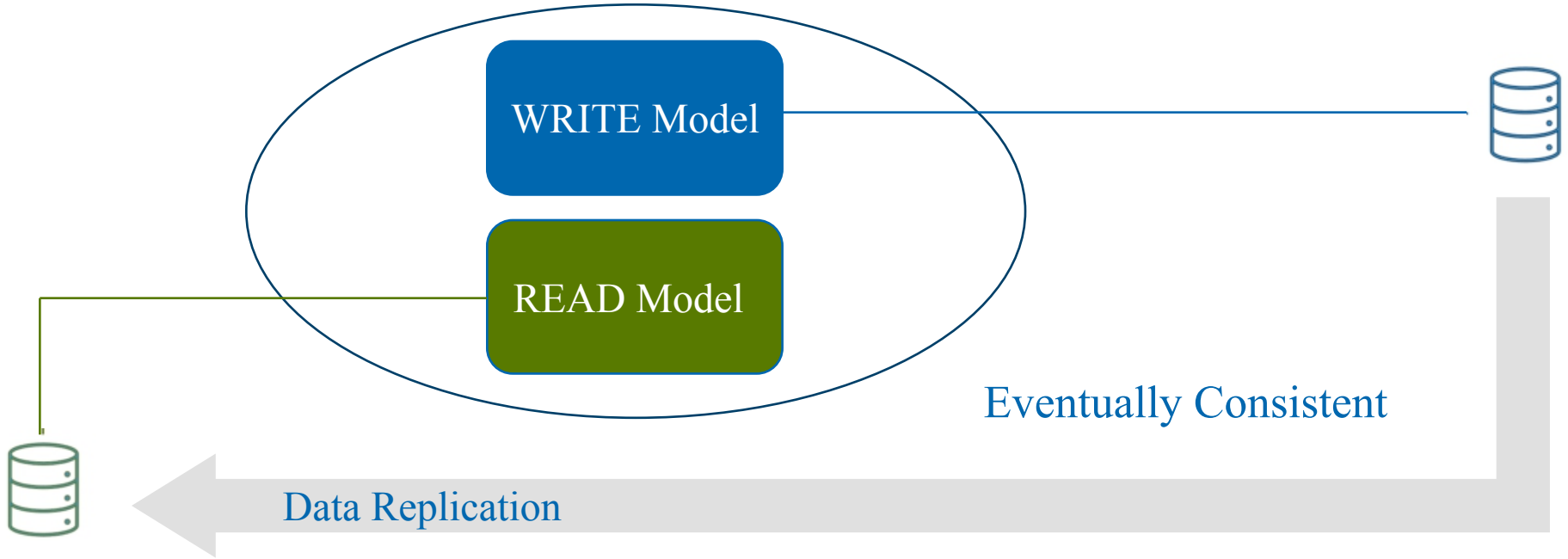


Technical feasibility | May impact WRITE Performance

Data Replication Considerations

NEEDED for independent READ | WRITE Data stores

Asynchronous Replication



READ side may NOT reflect the current state !!!

Data Replication Options

Select based on use case | requirements

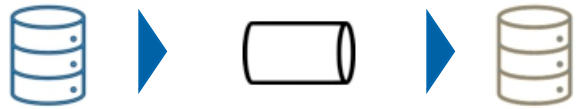
- Change Data Capture (CDC)

- Native replication

Replicas

- 3rd party tools

- Messaging | Streaming | Events



Quick Review

Replication needed between READ | WRITE data stores

- Synchronous | Asynchronous
- Option depends on requirements & supporting technologies

CQRS

Command Query Responsibility Segregation

1. Define CQRS
2. Benefits & Considerations
3. Data Replication

CQRS Frameworks

AxonIQ

eventuate

https://github.com/topics/cqrs - framework

CQRS : Proposal Management

Implementation of CQRS for Proposals

1. Proposal Write/Read Requirements
2. Component Diagram
3. Sequence Diagram

IT Lead

Proposal Management Design

Should it be implemented with CQRS

IT Lead

John, Travel Advisor

We are in a highly competitive industry; performance matters

Business rules change almost every week

Travis, Business Intelligence

We need access to ALL the proposals data - new & historical

Management dashboards need to be real time and performant

Jack, Marketing & Sales

We use current & historical data to understand market trends

Fast access to data is essential to be able to send offers

Implement Proposal Management using CQRS

Dynamic Business Requirements

Multiple types of READ/Query requirements

Expectation for High performance

Opportunity to achieve competitive advantage

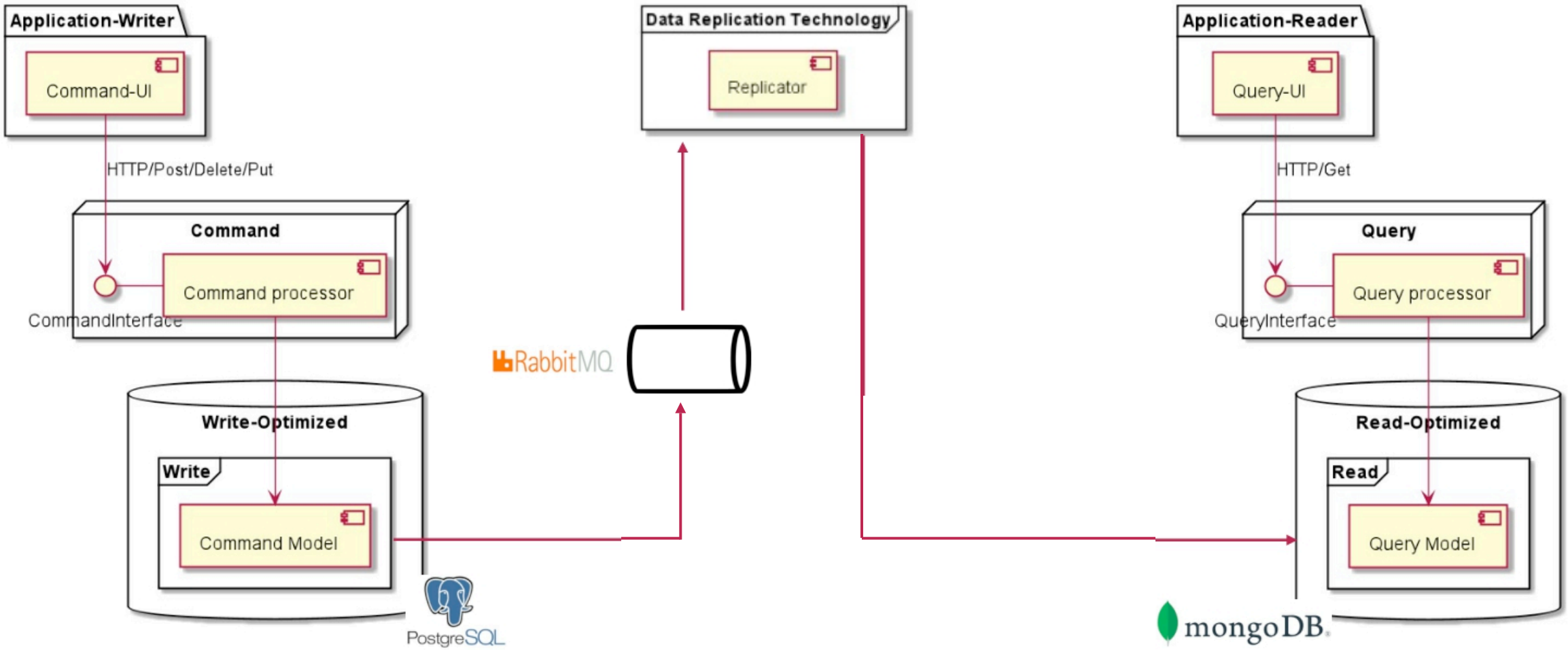
Proposal CQRS Deployment

uml / cqrs /cqrs.v2.component.puml

Command

Event Subscriber

Query

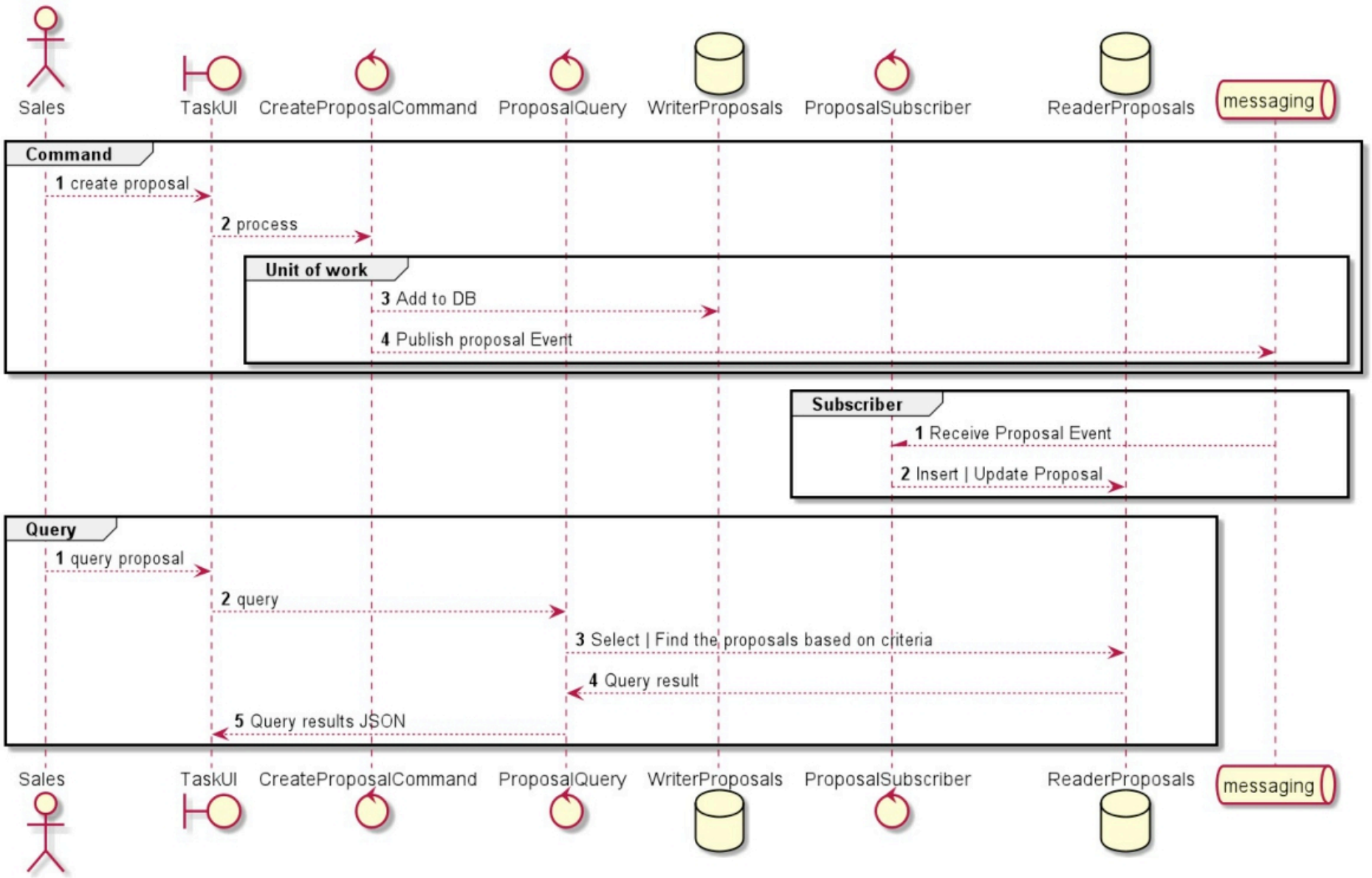


Proposal CQRS : Replication with MQ

uml / cqrs /commandquery.v2.sequence.puml

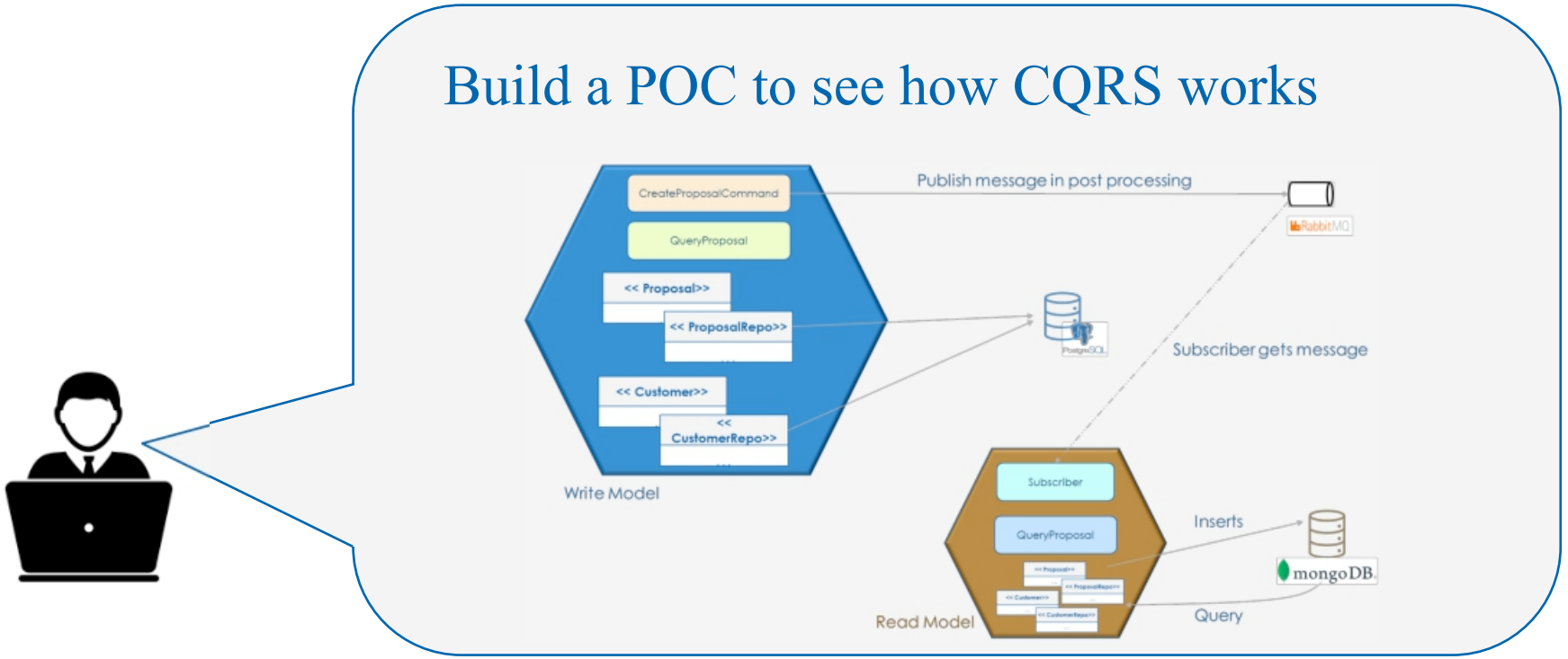
WRITE Model

READ Model



Decision to implement CQRS

- Depends on Use Case
- READ | WRITE characteristics

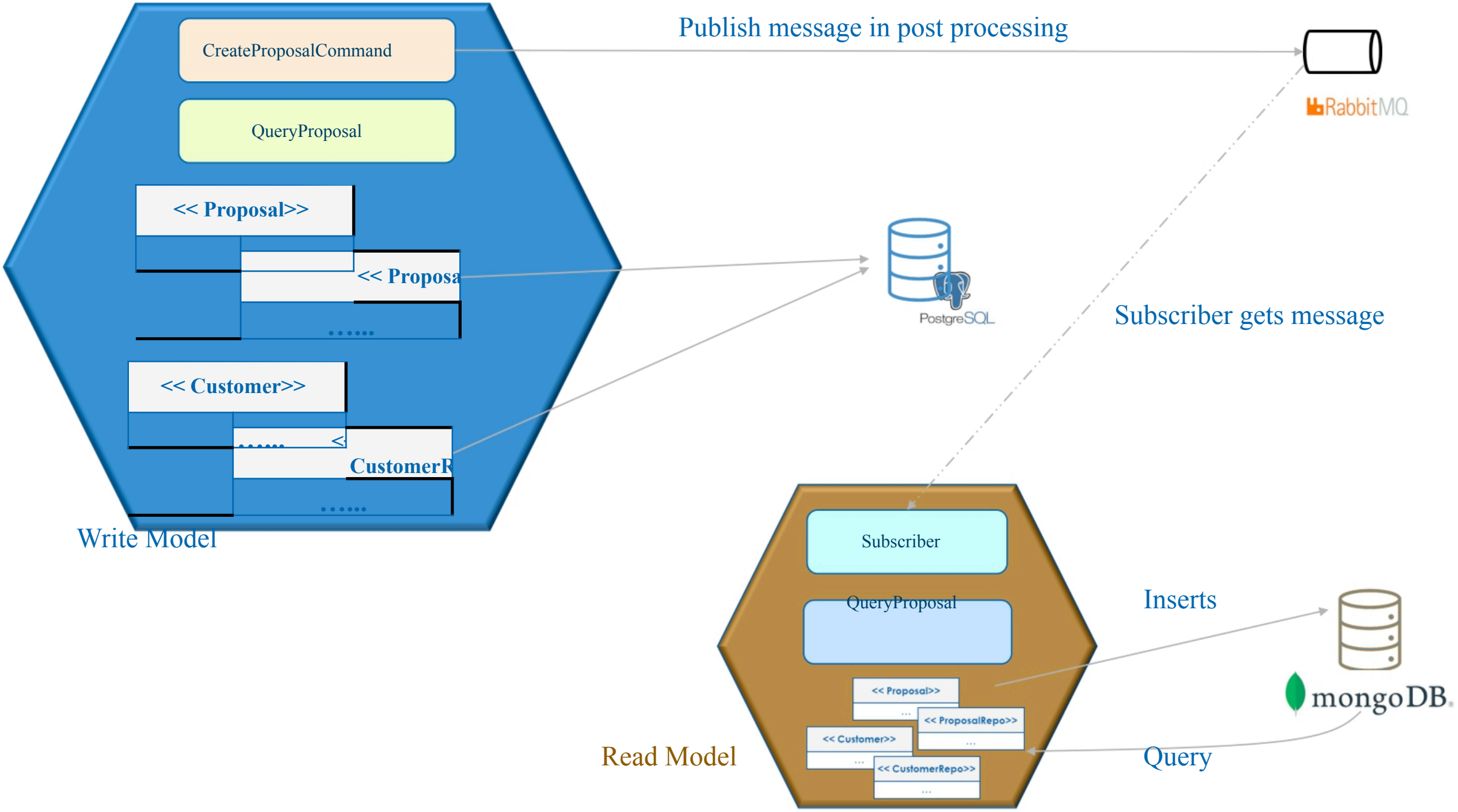


CQRS: Build & Test Command

Command will write out an event to the queue

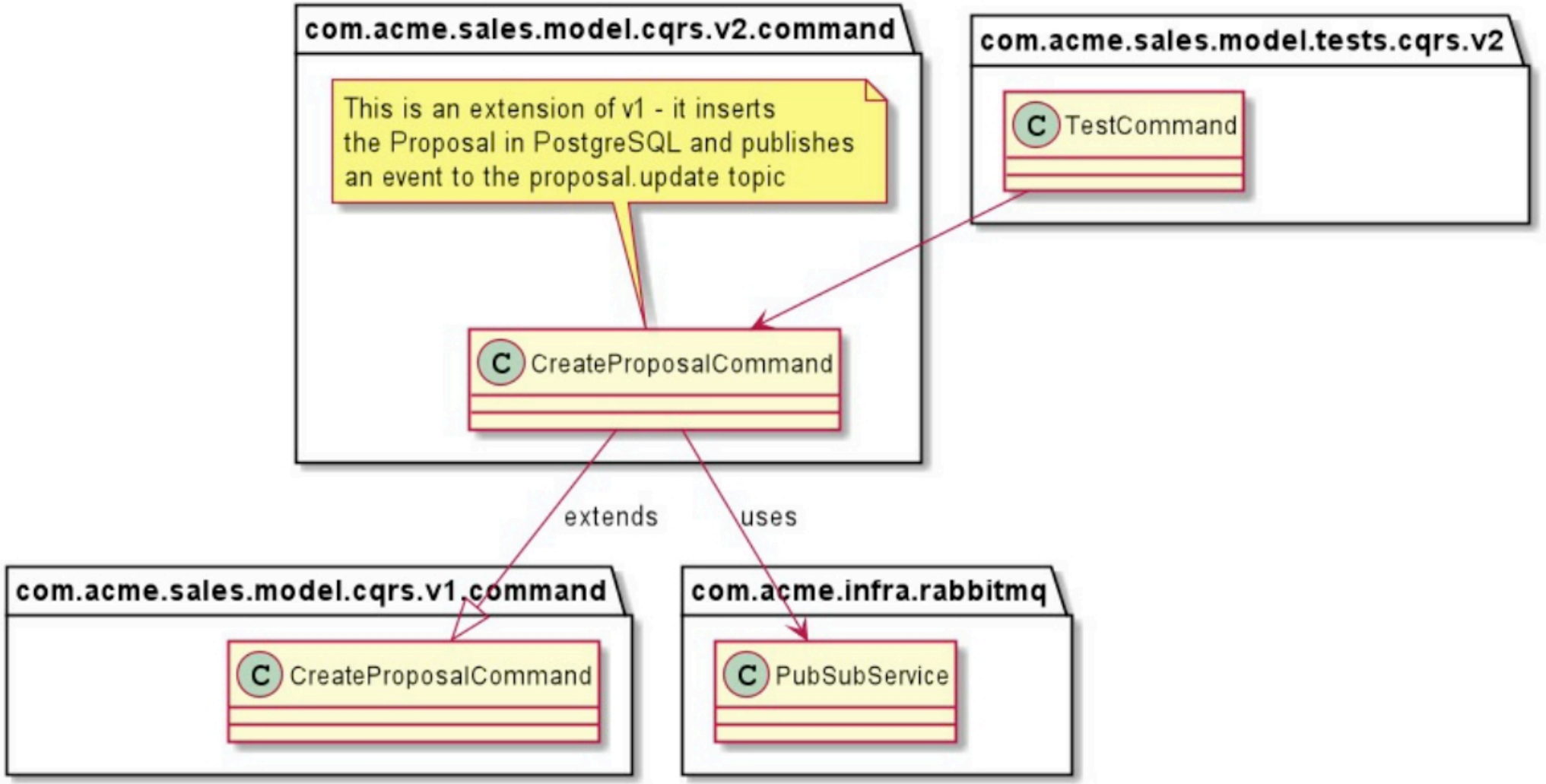
1. Setup Rabbit MQ for event messaging
2. Walkthrough of code for Command v2
3. Test the Command v2

Note: Uses the PostgreSQL Database & Rabbit MQ


CreateProposalCommand
QueryProposal
<< Proposal>>
<< Proposa
<< Customer>>
CustomerR
Write Model
Publish message in post processing
RabbitMQ
PostgreSQL
Subscriber gets message
Subscriber
QueryProposal
<< Proposal>>
<< ProposalRepo>>
<< Customer>>
<< CustomerRepo>>
Read Model
Inserts
mongoDB.
Query

Command Classes

uml/ cqrs/cqrs.v2.class.puml



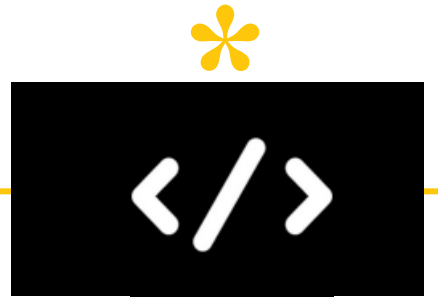
RabbitMQ setup

Queue	proposal.reader.queue
Topic Exchange	acme.sales.topic
Queue Binding	proposal.update

Refer: Lectures in section "Domain Events"

Setup MongoDB for READ side

Read Repository will use the MongoDB as datastore

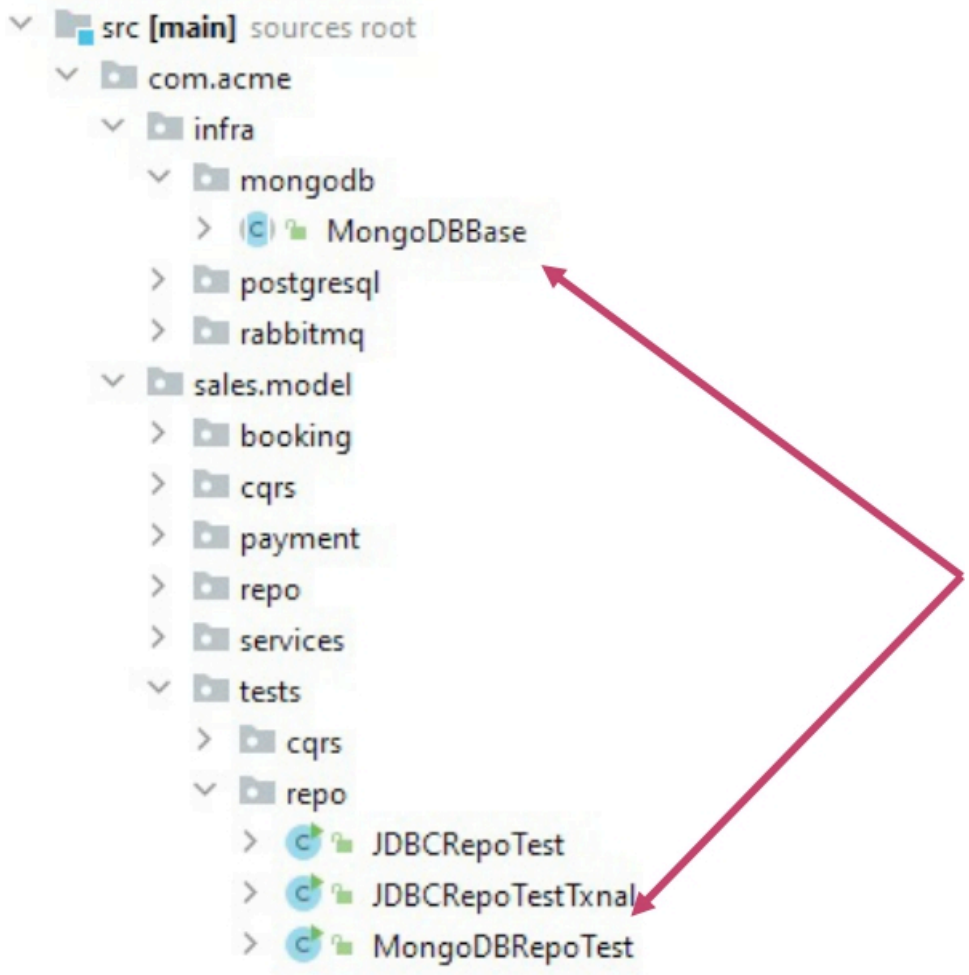


1. Setup MongoDB on cloud.mongodb.com
2. Walkthrough of MongoDBBase class
3. Test the MongoDB setup

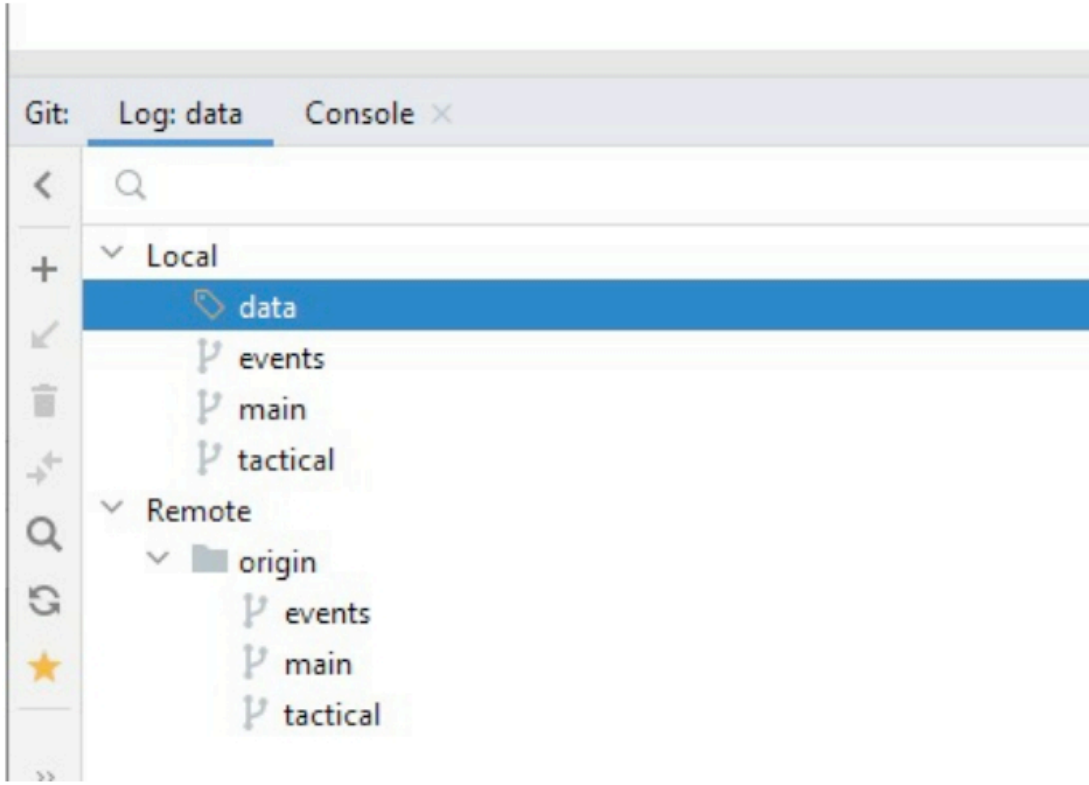
Note: Intent is NOT to teach MongoDB

Code

Checkout the branch : data

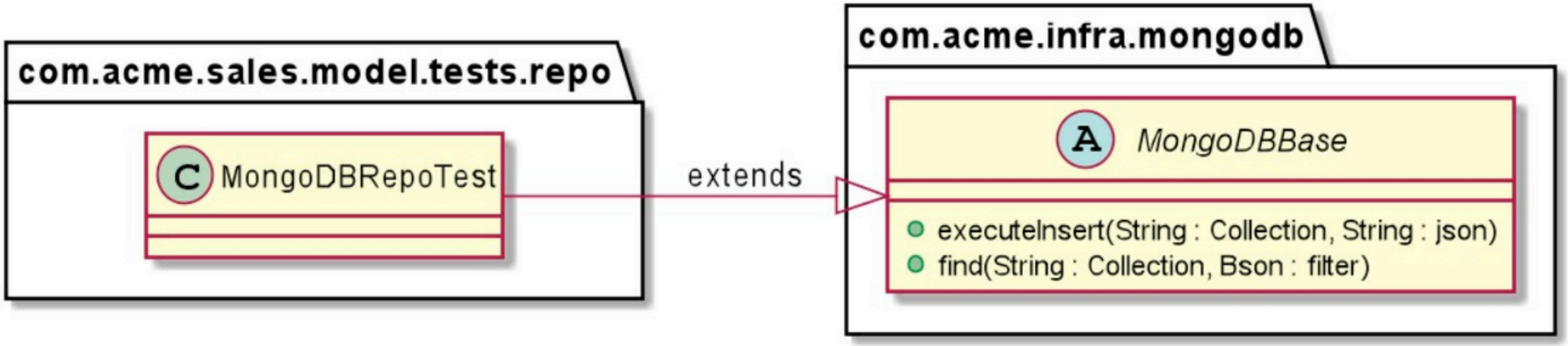


- Basic understanding of NoSQL Databases
- Instructions in README.md



MongoDBBase Class

uml/ cqrs/cqrs.v2.mongodb.class.puml



MongoDB

Create an instance of DB on cloud.mongodb.com

Cluster Name AcmeTravel

PS: You may create a local instance of MongoDB

Setup the database instance

Database user
repouser

Database pasword
repouser

Event Sourcing

Persisting the events in a data store

1. State - Oriented versus Event- Sourced persistence
2. Event Stores
3. Benefits of Event Sourcing

State Management

State -Oriented persistence system maintains only the current state

Example:

- Number of "Item x" in the inventory
- Item Added => Count increased
- Item Sold => Count decreased

Event -Sourced persistence systems persist all state changes

- Domain events are stored as they are received

Event Sourcing

Event sourcing suggests persisting the domain events in an Event store and using it for recreating the state of the domain objects at any point in time

Event store is an append only event log used for persisting the received events

Example : Bank Transactions

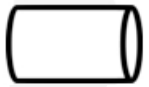
READ side persists events emitted by WRITE side in an Event Store

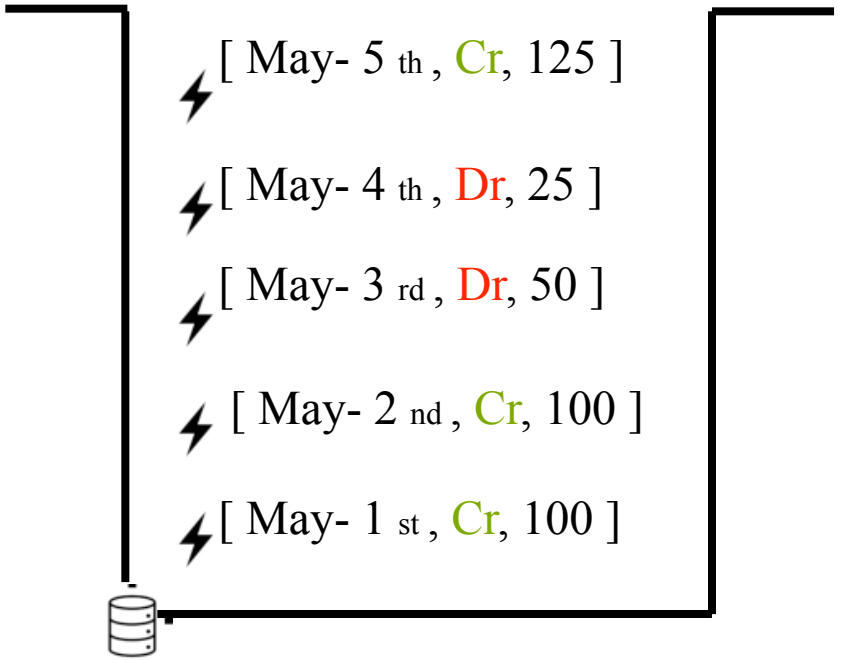
WRITE

READ

John's Bank Account

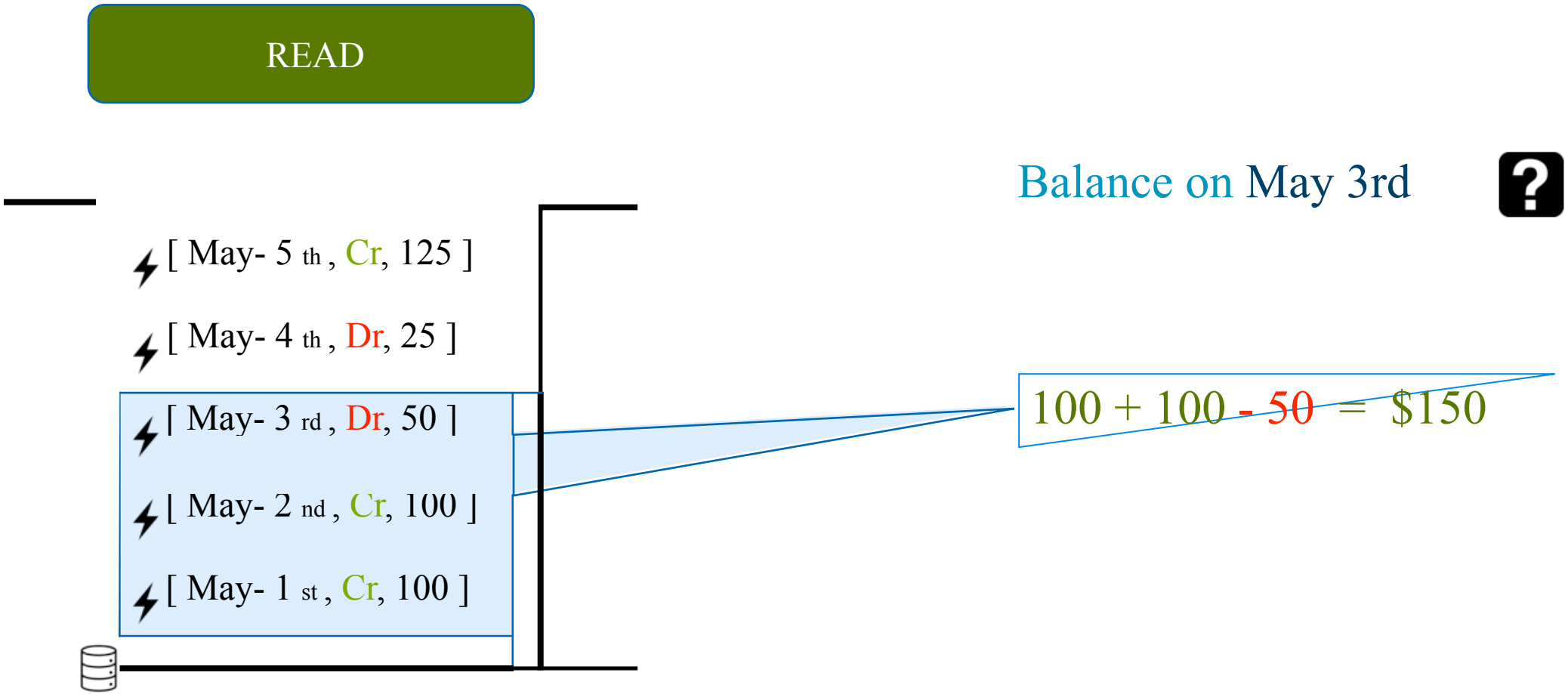
100	May - 1 st	Deposit	$100
101	May - 2 nd	Deposit	$100
102	May - 3 rd	Withdraw	$50
103	May - 4 th	Withdraw	$25
104	May - 5th	Deposit	$125





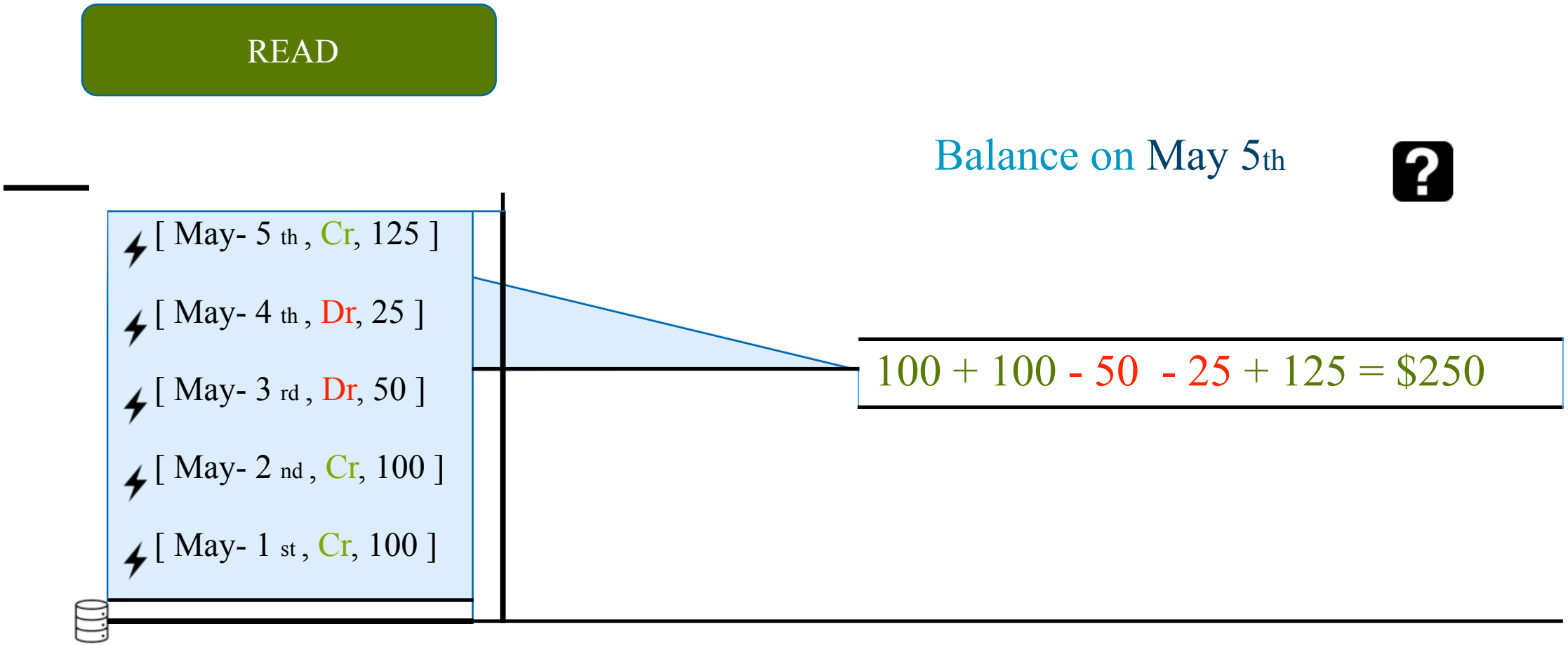
Example : Bank Transactions

READ side can create the state for any date



Example : Bank Transactions

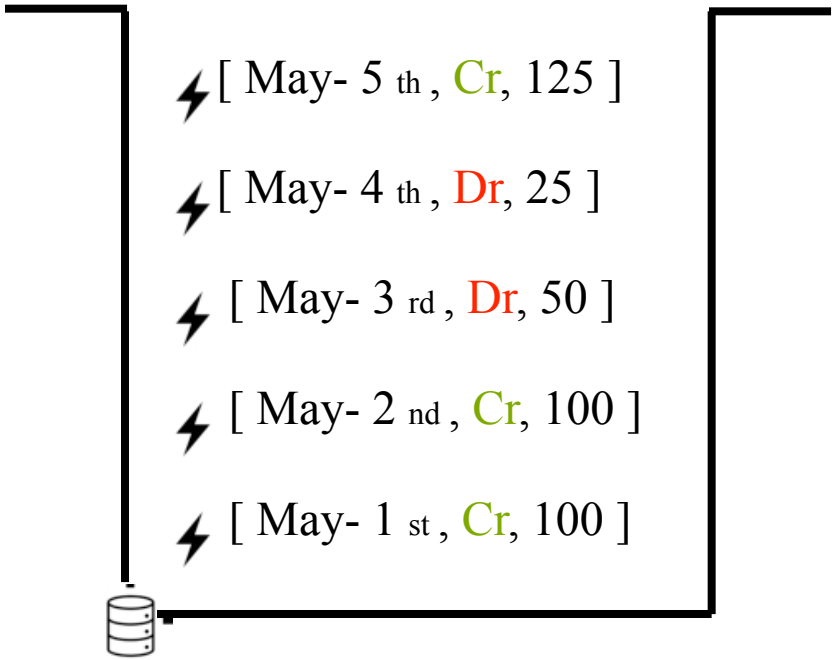
READ side can create the state for any date



Performance consideration

Recreating the state from events will lead to BAD performance

READ



Current state in a separate Datastore

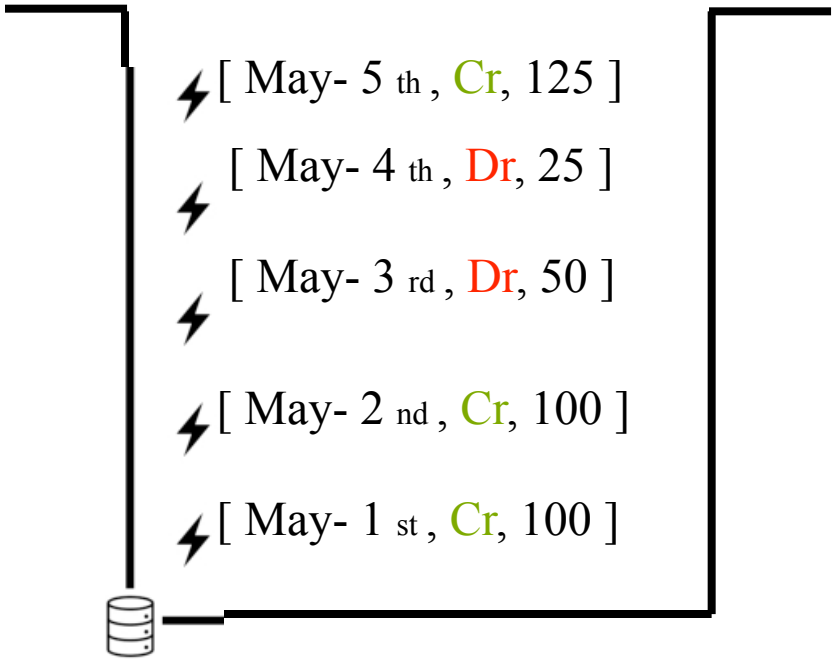
Lookup on Query

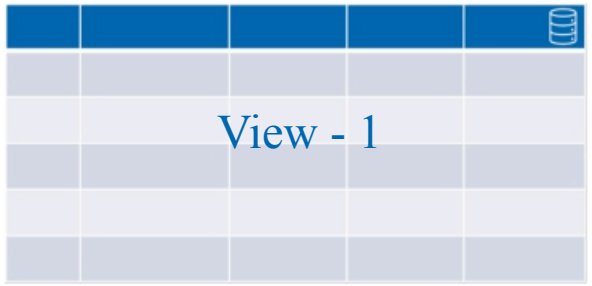
100 May 1 st	CR	100	100
101 May 2 nd	CR	100	200
102 May 3 rd	DR	50	150
103 May 4 th	DR	25	125
104 May 5 th	CR	125	250

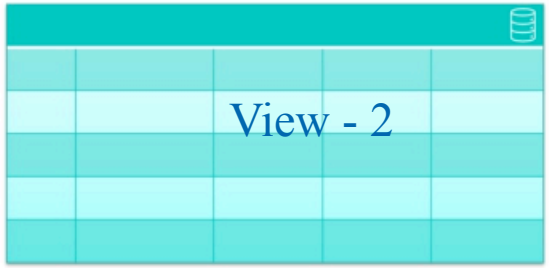
Performance consideration

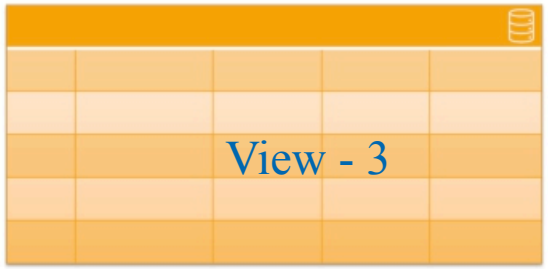
Multiple data views may be created for optimizing queries

READ









Benefits

1. Event replay to create *"point in time state"*
2. Multiple read models | views
3. Accurate out of the box auditing
4. Simplified Reconciliation
5. Temporal & Complex historical queries

Realization of Event Store

Traditional databases may be used

- RDBMS

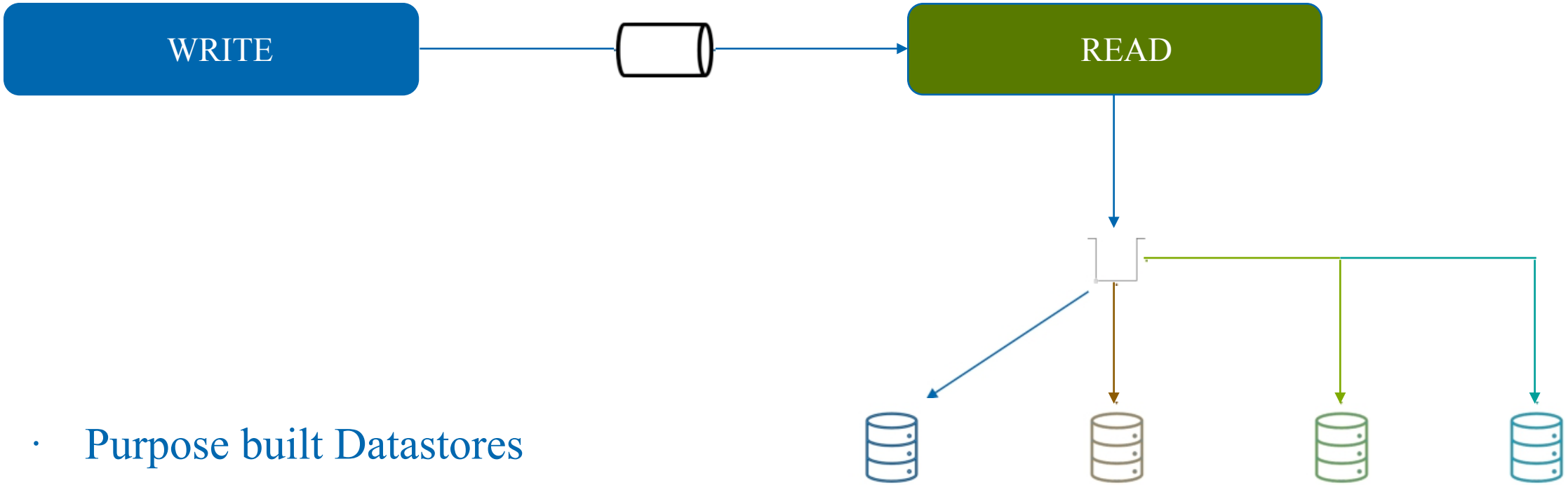
- NoSQL Database

- Specialized databases

https://www.eventstore.com

CQRS & Event Sourcing

Commonly used together



- Purpose built Datastores

When to use Event Sourcing?

Evaluate use case from the event sourcing benefits perspective

1. Event replay to create "*point in time state*"
2. Multiple read models | views
3. Accurate out of the box auditing
4. Simplified Reconciliation
5. Temporal & Complex historical queries

Quick Review

Event -Sourced persistence systems persist all state changes

- Current state managed in a separate data store for performance
- Out of the box Audit, Reconciliation, Temporal queries

Events are stored in Event Stores that may be traditional DBs

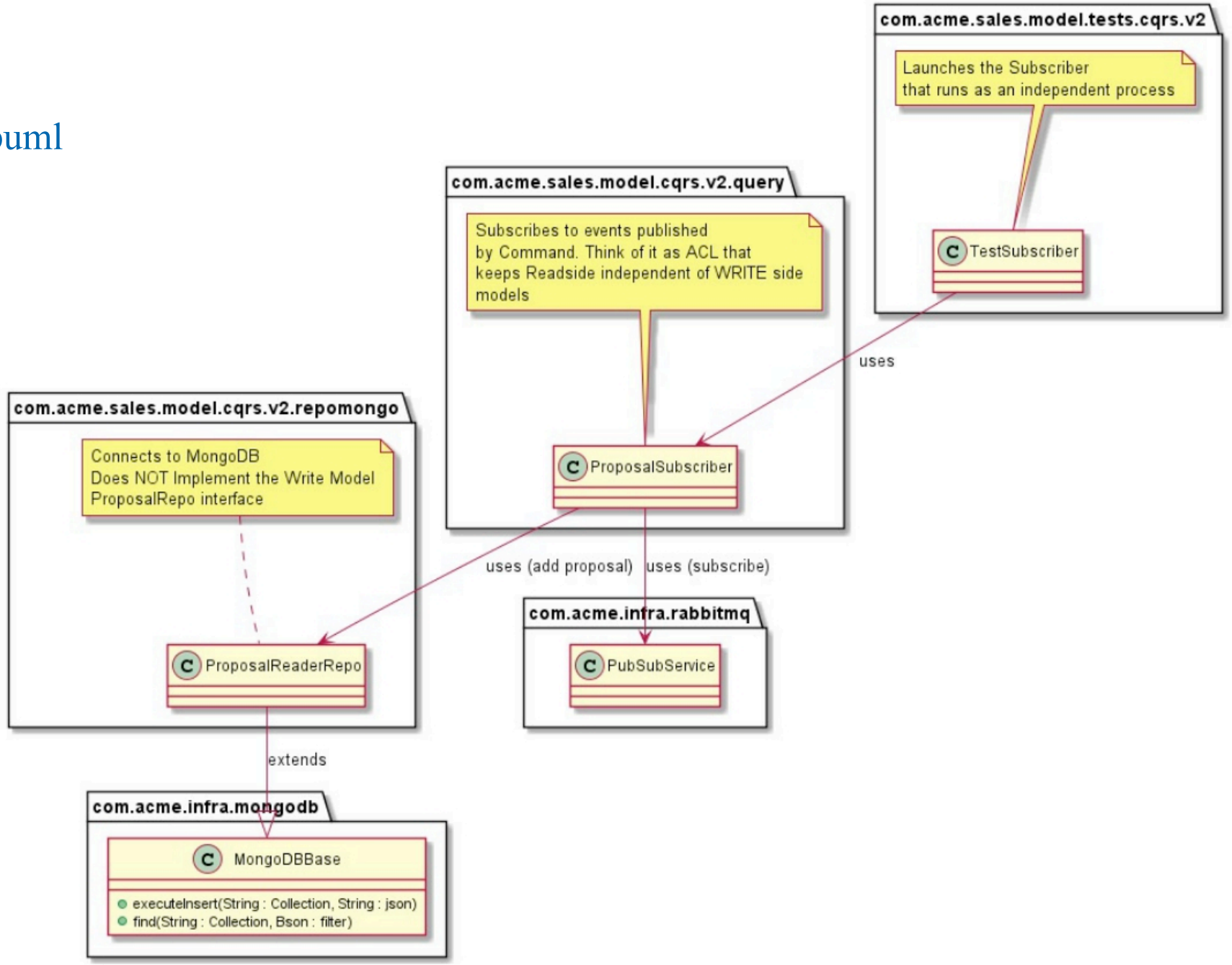
CQRS: Build ACL Subscriber

Stand alone Subscriber protects the READ side from corruption

1. Walkthrough of ProposalReaderRepo
2. Walkthrough of ProposalSubscriber
3. Test the Proposal Subscriber

READ Side Classes

uml/ cqrs/cqrs.v2.readside.class.puml



Proposal Event

Testing

1. Launch TestSubscriber

2. Execute the TestCommand

3. Check data in MongoDB

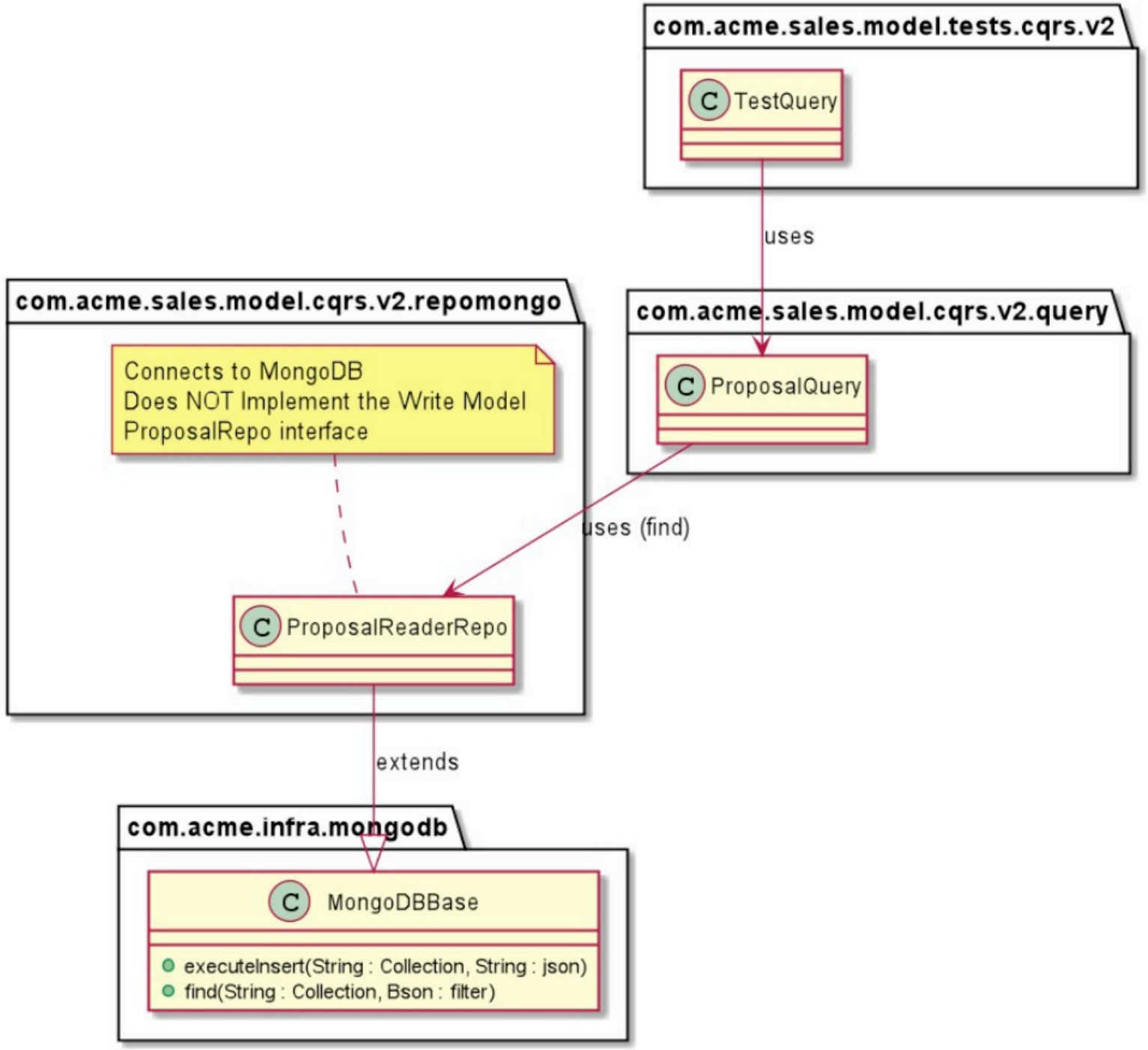
CQRS: READ Side Query

READ Side Query will use the MongoDB Data Store

1. Walkthrough of ProposalQuery
2. Test the Proposal Query

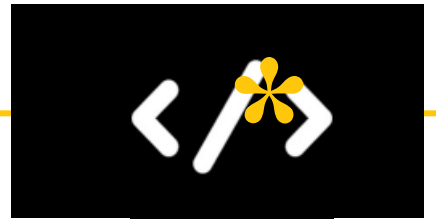
READ side Query class

uml/ cqrs/cqrs.v2.readside.class.puml



Reliable Messaging | Dist Txn

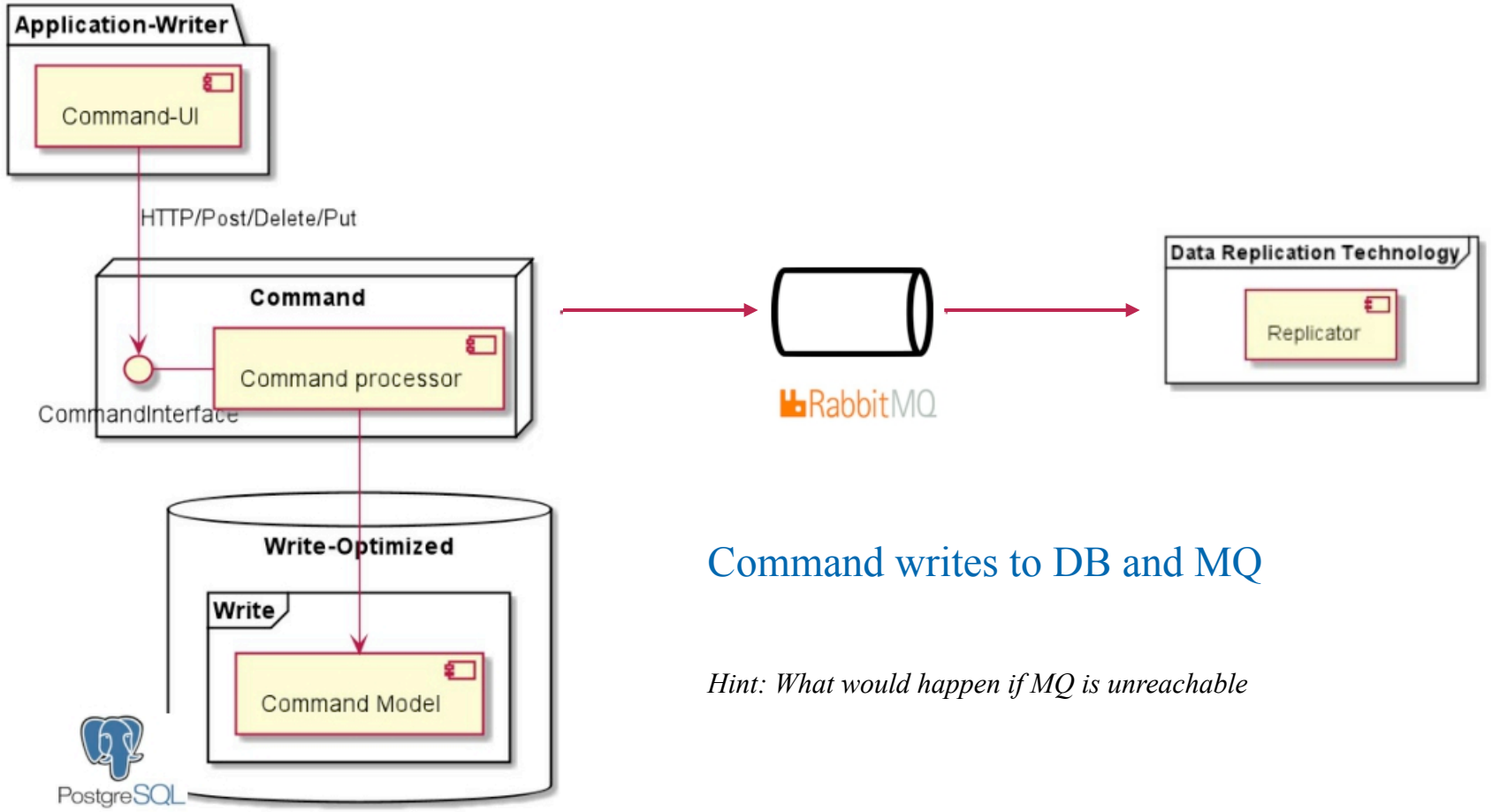
What are they?



https://vimeo.com/111998645

1. Challenge
2. Publishing the messaging
3. Receiving the messaging

Command Writes to DB & MQ



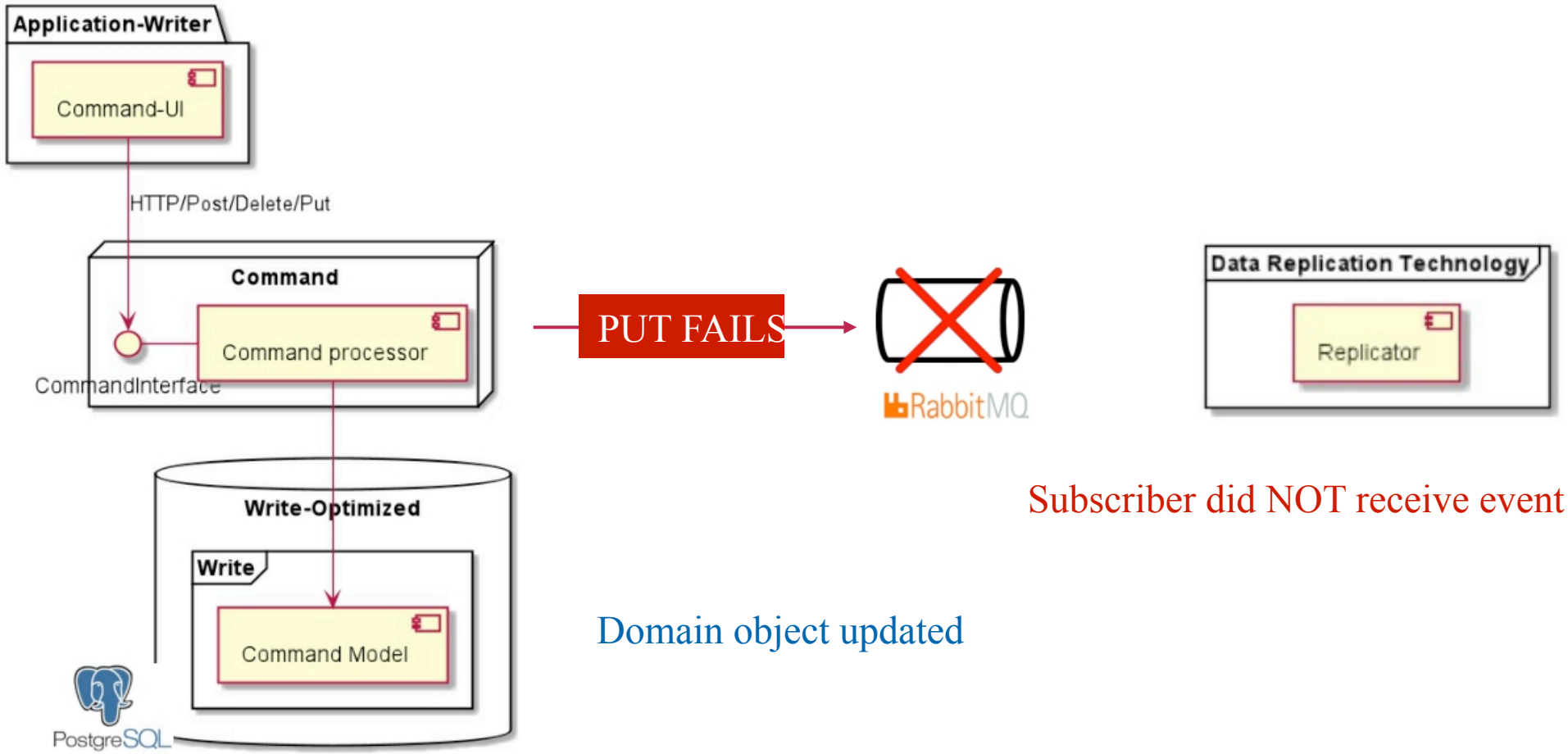
Command writes to DB and MQ

Hint: What would happen if MQ is unreachable

Do you see a problem here?

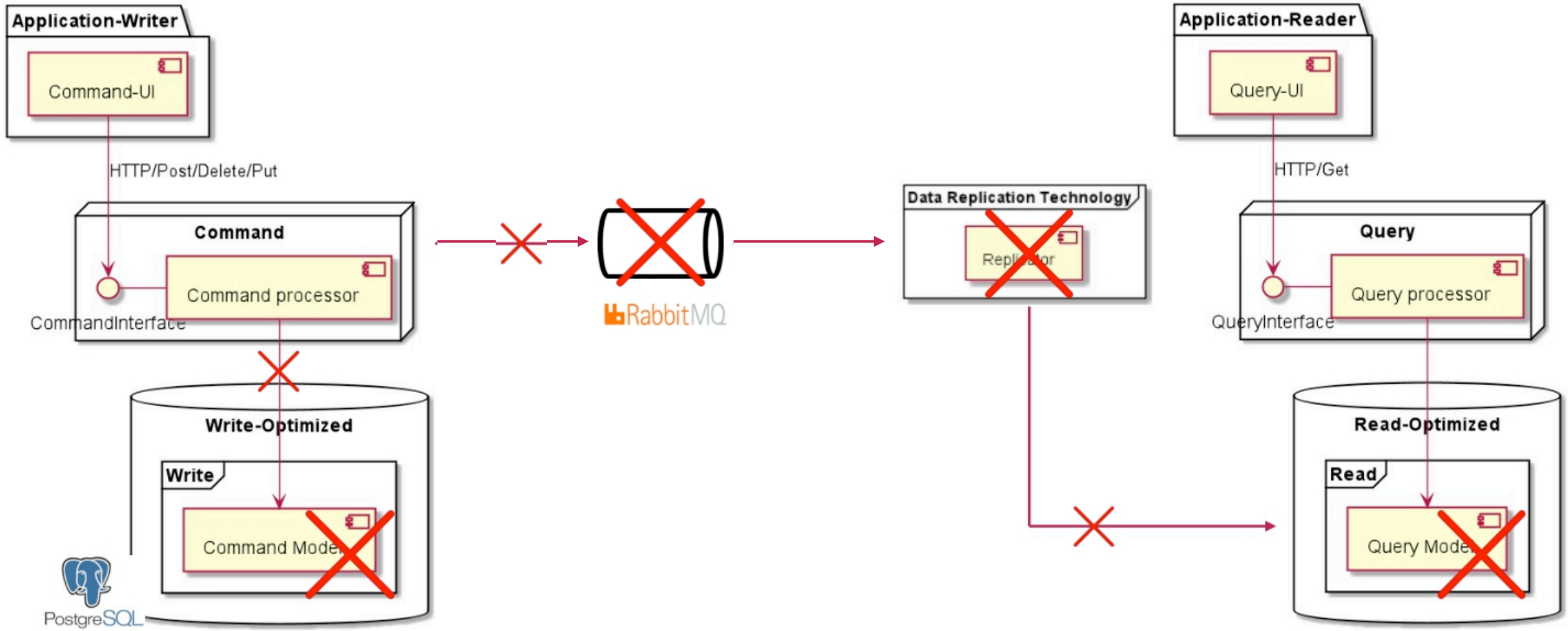
Events will be LOST !!!!

UNLESS DB Write and MQ PUT are carried out in a single Transaction



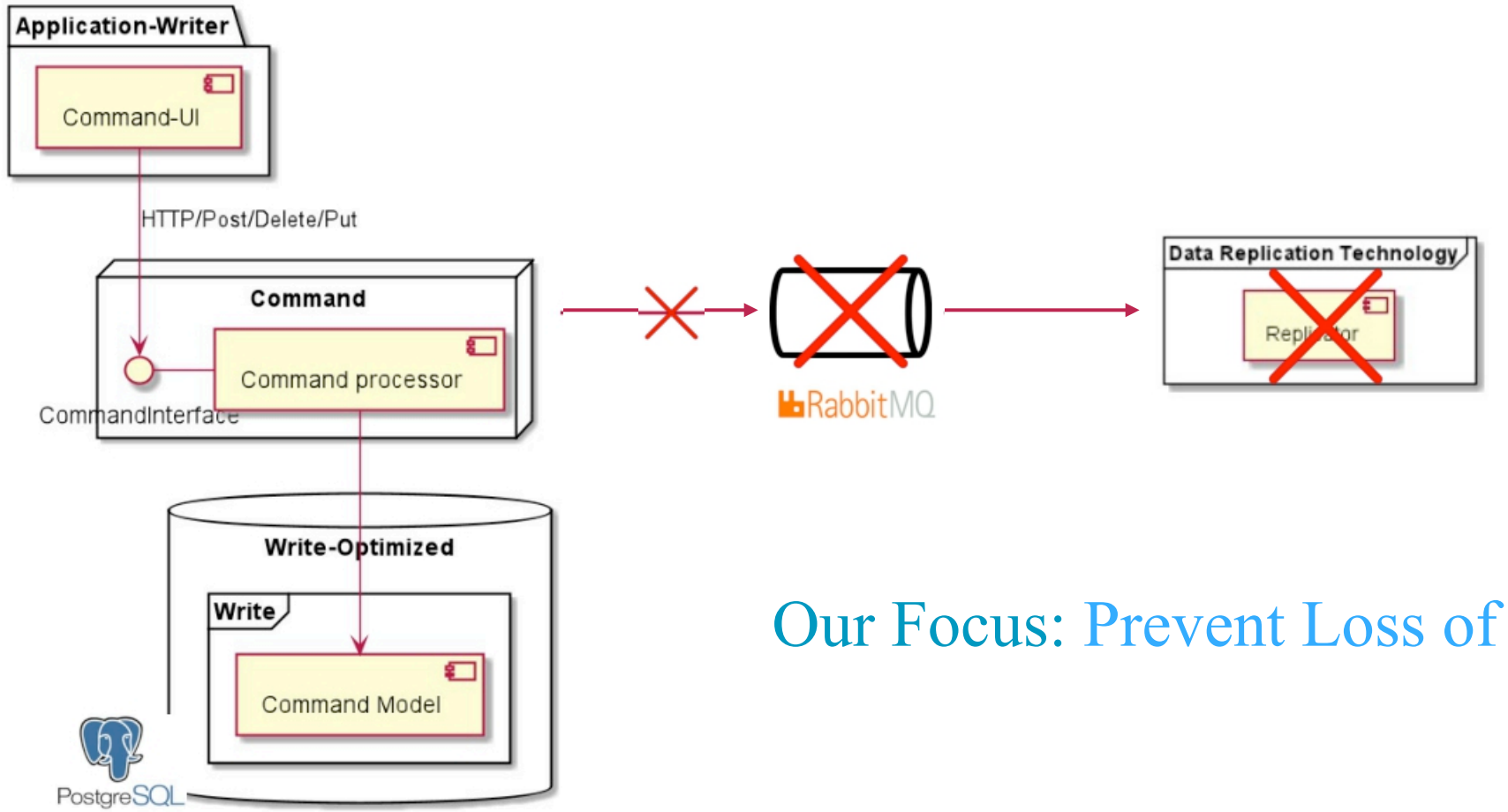
Failures in Distributed systems

Just ASSUME that there will be failures & prepare



Failures in Distributed systems

Just ASSUME that there will be failures & prepare



Our Focus: Prevent Loss of Events

Solutions

DB Write & MQ Put in a single unit of work (transaction)

2 Phase Commit

Distributed algorithm that coordinates all the processes involved in distributed transaction

- Also known as eXtended Architecture or XA
- Transaction manager coordinates with resources

https://en.wikipedia.org/wiki/Two - phase_commit_protocol

Solutions

DB Write & MQ Put in a single unit of work (transaction)

2 Phase Commit

Distributed algorithm that coordinates all the processes involved in distributed transaction

- Many commonly used distributed technologies do NOT support it !!!!

https://en.wikipedia.org/wiki/Two - phase_commit_protocol

Solutions

DB Write & MQ Put in a single unit of work (transaction)

Reliable Messaging

Write aggregate data & event data in the database with a local transaction and replay the events against queueing system in a subsequent step.

Reliable Messaging in action

An implementation that prevents loss of event messages

1. Walkthrough of ProposalReaderRepo
2. Walkthrough of ProposalSubscriber
3. Test the Proposal Subscriber

IT Lead

Proposal Events CQRS Testing

Events are LOST when Rabbit MQ is down!!

Build Reliable Messaging to fix it !!!!

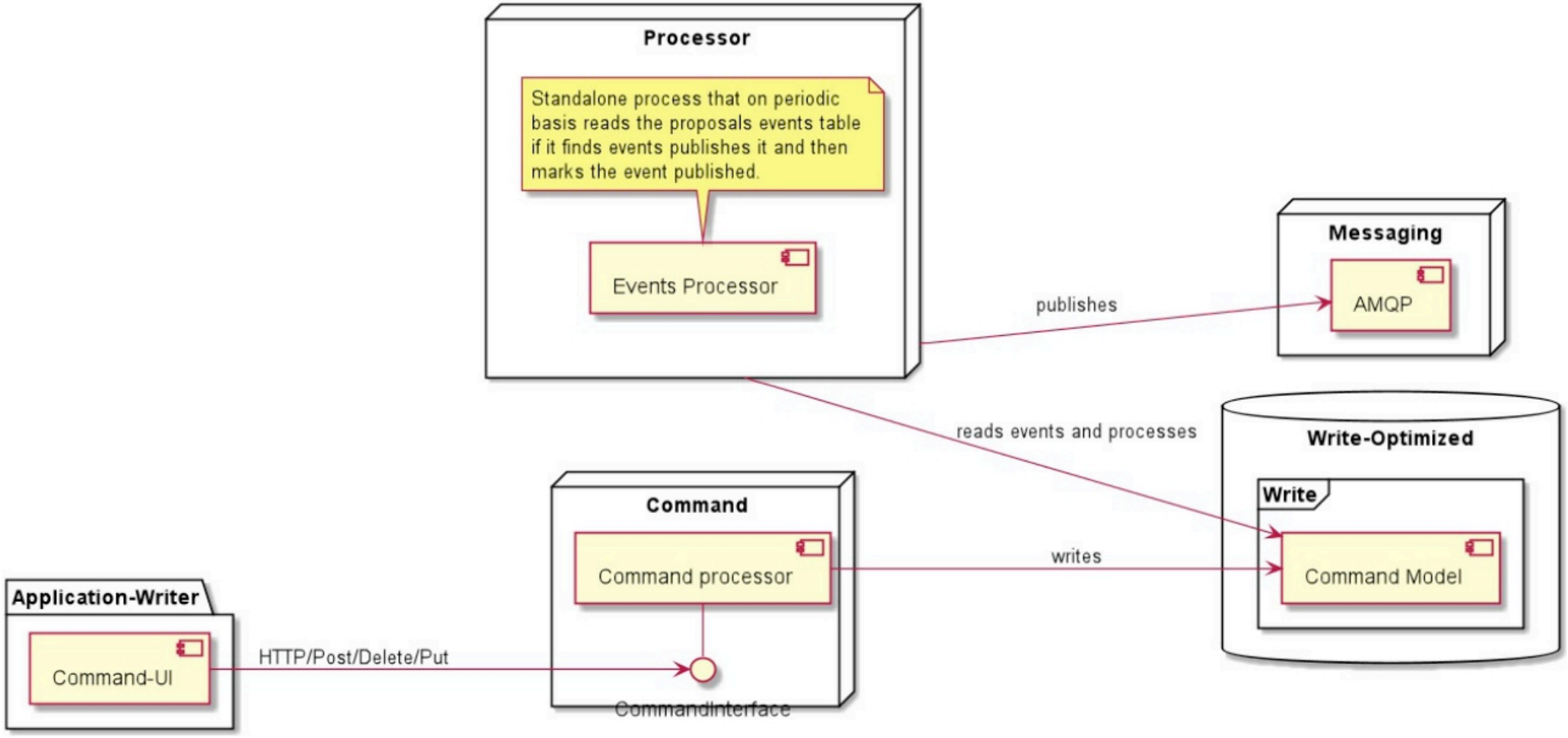
Setup the DB tables : proposal events table

sql/reliable -messaging -ddl.sql

```
CREATE TABLE proposalevents (
    event_id     INT  generated by default as identity,
    proposal_id  INT,

    event_guid   VARCHAR(36) not null,

    payload      VARCHAR(1024) not null,

    processed    BOOLEAN DEFAULT FALSE,

    created_date  TIMESTAMP DEFAULT now(),
    processed_date  TIMESTAMP,

    PRIMARY   KEY(event_id),

    CONSTRAINT fk_customer
        FOREIGN KEY(proposal_id)
            REFERENCES proposals(proposal_id)
);
```

Write side Components

uml/ cqrs/cqrs.writeside.v3.component.puml



Event processing classes

uml/ cqrs/cqrs.writeside.v3.class.puml