POLYGLOT PERSISTENCE

VIA

CQRS

@vladikk

vladikk.com

Internovus

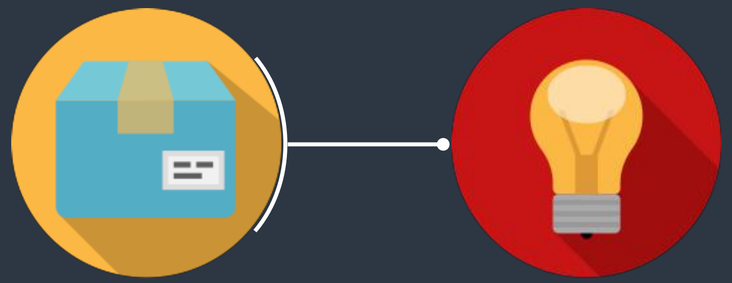
AGENDA

Problem Definition

Proposed Solution: CQRS

Experience report of applying CQRS at Internovus

Your Product



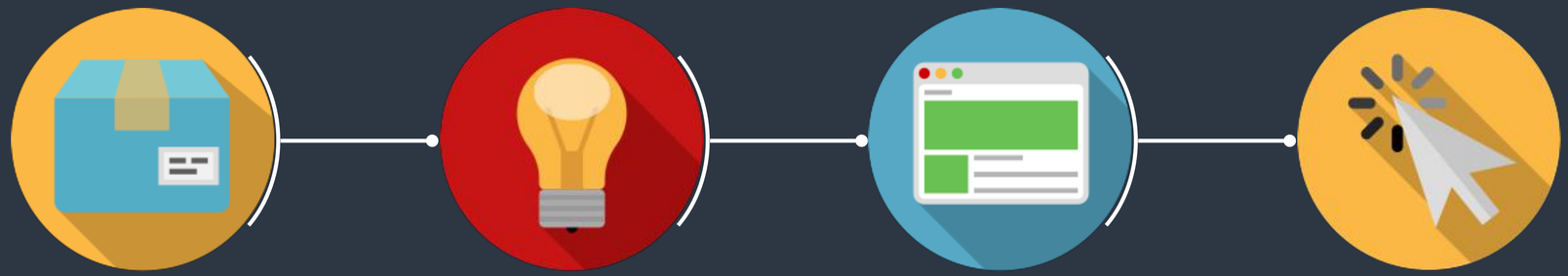
Your Product

Marketing
Strategy

Your Product

Marketing Strategy

Creatives



Your Product

Marketing Strategy

Creatives

Campaigns

Your Product

Marketing Strategy

Creatives

Campaigns

Sales Agents

Your Product

Marketing Strategy

Creatives

Campaigns

Sales Agents

Optimization

vladikk

Your Product

Marketing Strategy

Creatives

Campaigns

Sales Agents

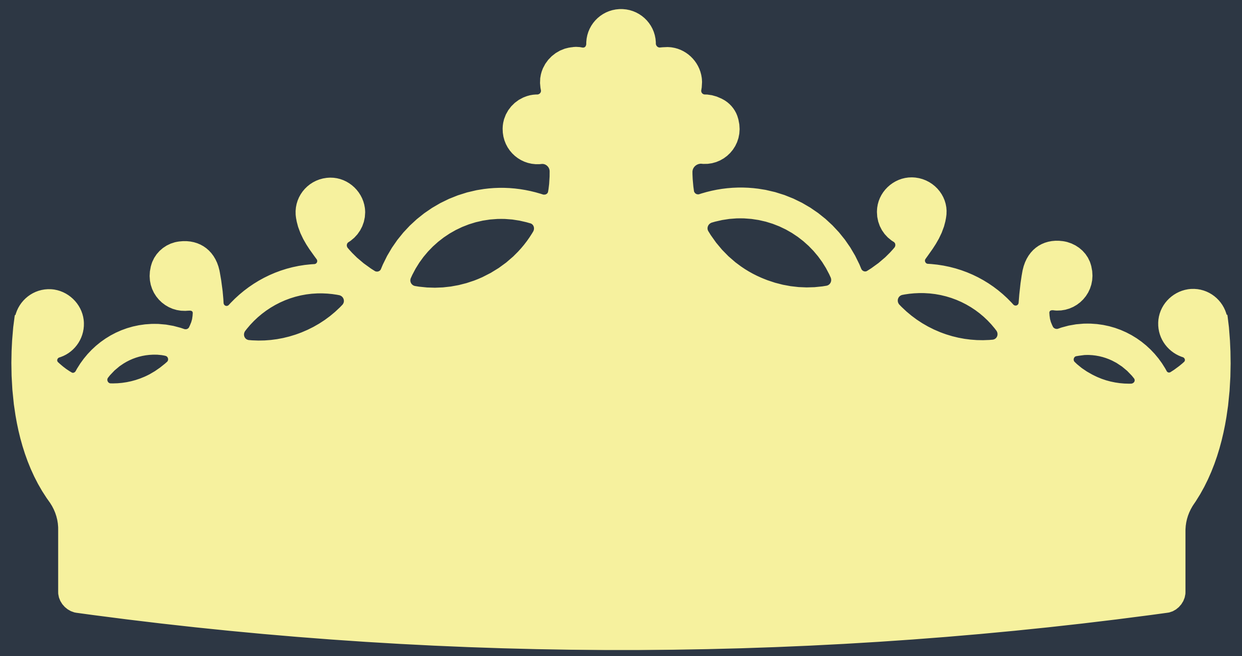
Profits

Optimization

PROBLEM

01

Back in the good ol’ days…



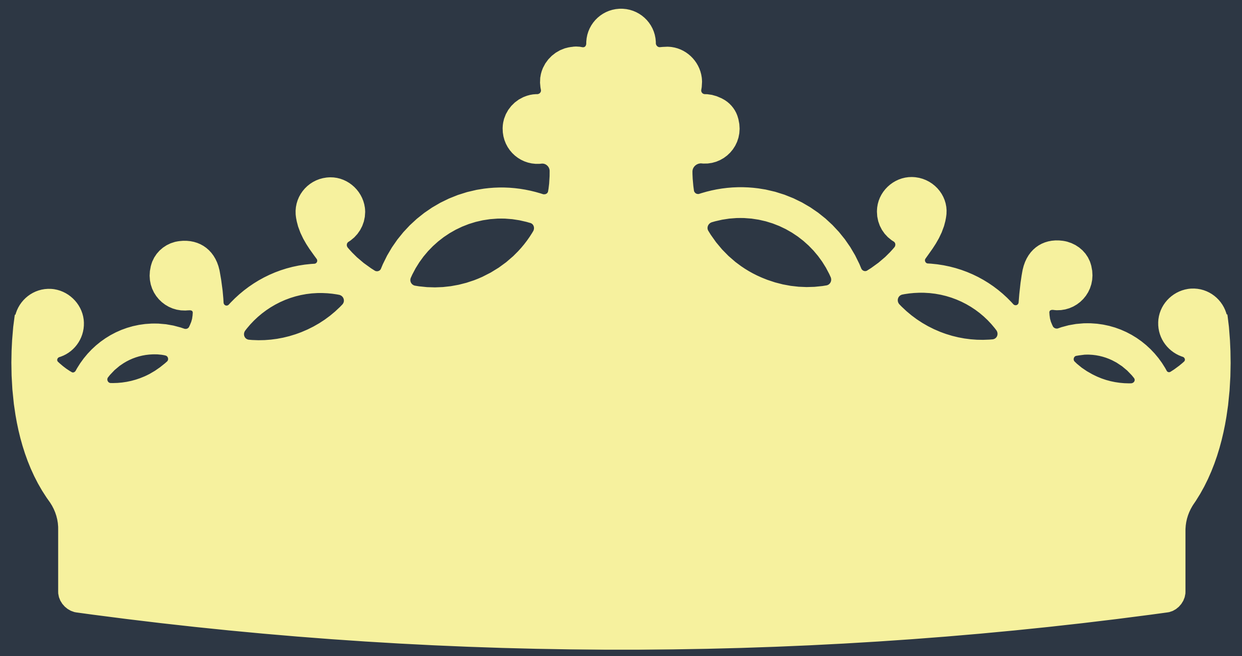
Relational Databases (RDBMS)

MySQL

Microsoft SQL Server

Oracle

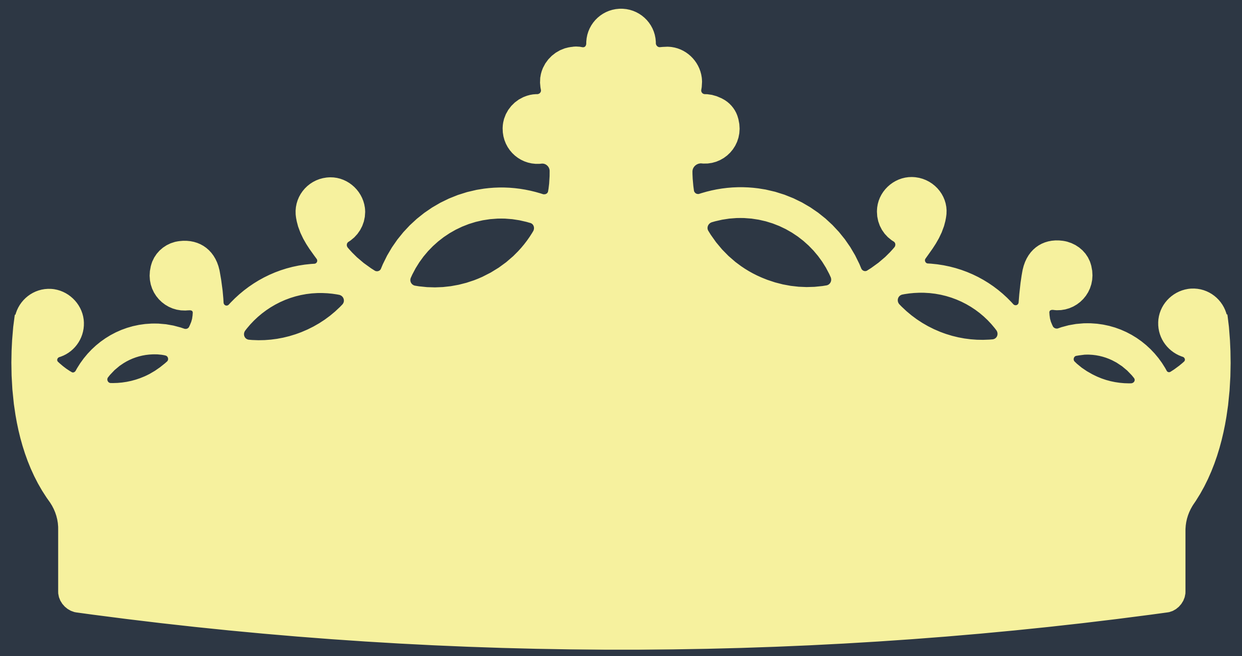
etc



Relational Databases (RDBMS)

Relational Databases (RDBMS)


Web 2.0
Aggregators
Folksonomy
Wikis
Blogs
User Centered
Joy of Use
Participation
Six Degrees
Usability
Pagerank
XFN
Widgets
Recommendation
Social Software
FOAF
Browser
Simplicity
Sharing
Collaboration
Perpetual Beta
Videocasting
Podcasting
AJAX
Audio
IM
Video
Design
Convergence
CSS
Pay Per Click
UMTS
Mobility
Atom
XHTML
SVG
Ruby on Rails
VC
Trust
Affiliation
OpenAPIs
RSS
Semantic
Web Standards
SEO
Economy
OpenID
Remixability
REST
Standardization
The Long Tail
DataDriven
Accessibility
XML
Microformats
Syndication
Modularity
SOAP



Relational Databases (RDBMS)

NoSQL

Consistency

Availability

Partition Tolerance

Document stores

Key/Value stores

Columnar stores

Graph databases

Search indexes

Document stores: MongoDB, CouchDB, RavenDB

Key/Value stores: DynamoDB, Redis, Memcached

Columnar stores: Cassandra, HBase, Vertica

Graph databases: Neo4j, HyperGraphDB, InfiniteGraph

Search indexes: Lucene, Elasticsearch

CHOOSING THE RIGHT DB

Many different options, each having its pros and cons

High risk architectural decision

Very hard to change in the future!

ABSTRACT REPOSITORIES?

```
interface IRepository { ... }

class SqlServerRepository : IRepository { ... }

class MongoRepository : IRepository { ... }

class DynamoRepository : IRepository { ... }
```

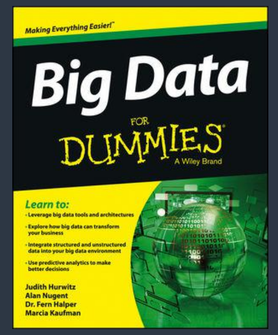
ABSTRACT REPOSITORIES?

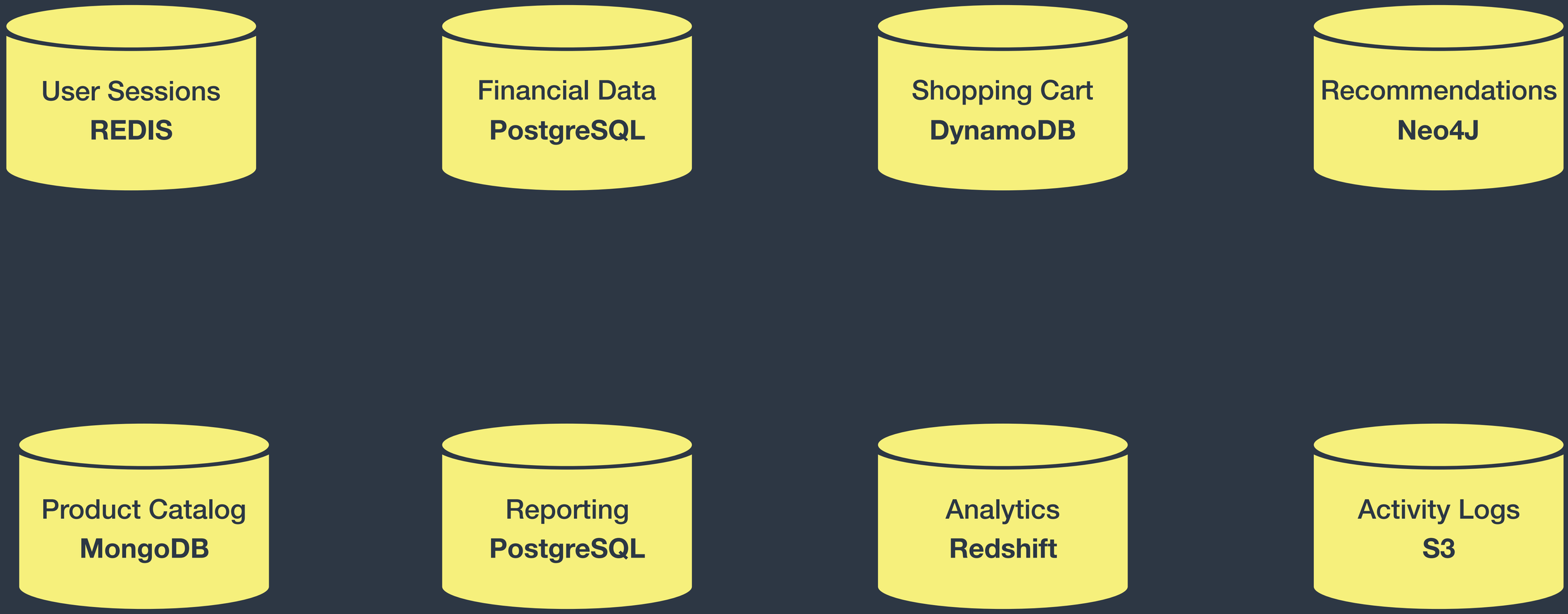
```
interface IRepository { ... }

class SqlServerRepository : IRepository { ... }

class MongoRepository : IRepository { ... }

class DynamoRepository : IRepository { ... }
```

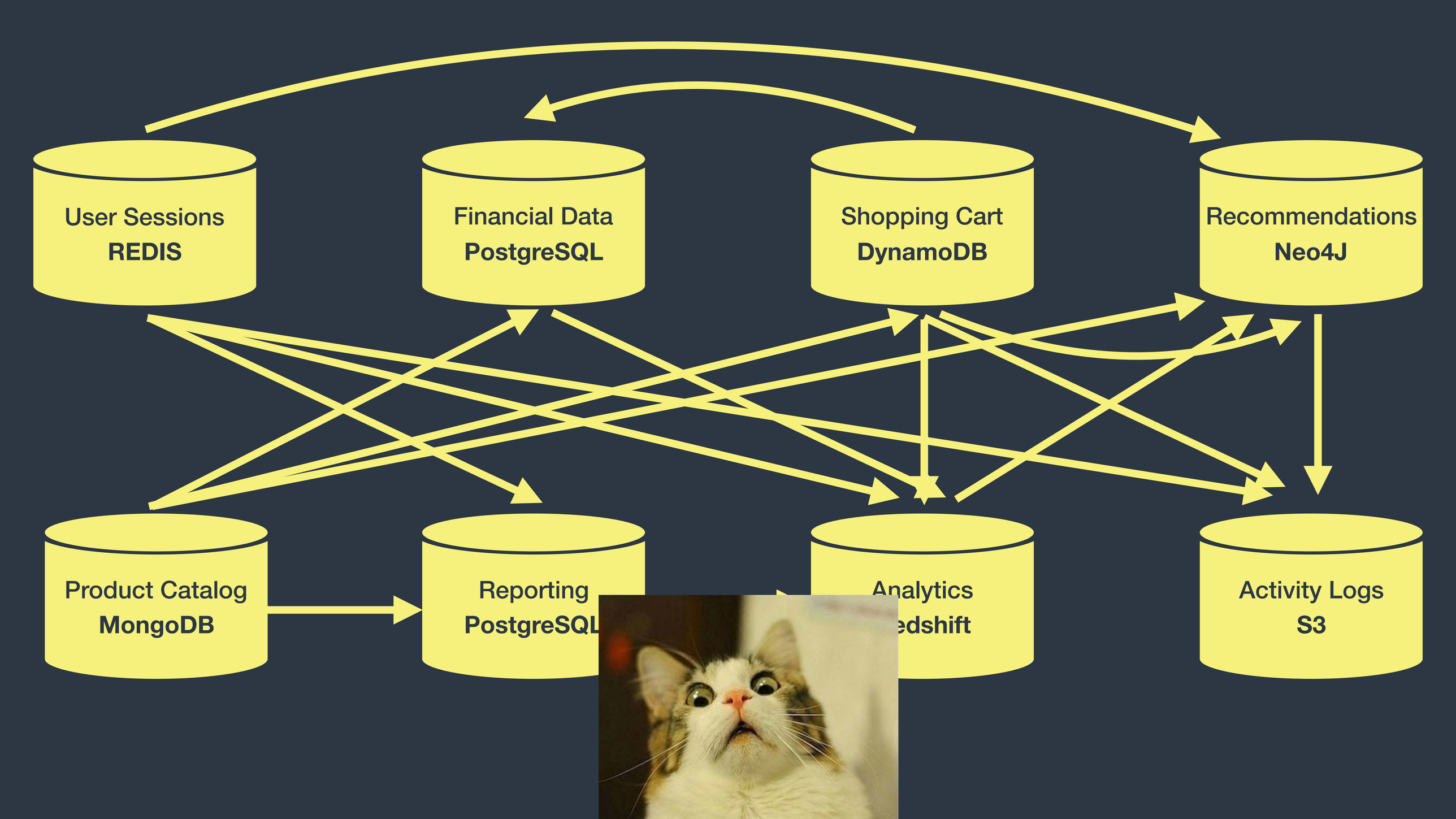
POLYGLOT PERSISTENCE:
USE THE RIGHT TOOL FOR THE JOB

“You are already using **polyglot persistence** in your existing workplace… You are probably using multiple *RDBMSs, data warehouses, flat files, content management servers,* and so on.”

“The most ideal of environments, where you have **only one persistence technology**, is probably **not suited to big data problem solving**”




User Sessions
REDIS
Financial Data
PostgreSQL
Shopping Cart
DynamoDB
Recommendations
Neo4J
Product Catalog
MongoDB
Reporting
PostgreSQL
Analytics
Redshift
Activity Logs
S3


User Sessions
REDIS
Financial Data
PostgreSQL
Shopping Cart
DynamoDB
Recommendations
Neo4J
Product Catalog
MongoDB
Reporting
PostgreSQL
Analytics
edshift
Activity Logs
S3


INTERNOVUS
The Ultimate Acquisition Solution

Your Product

Marketing Strategy

Creatives

Campaigns

Sales Agents

Profits

Optimization

CHALLENGES WE FACED

Database - high risk decision

Data flow management

Multiple sources of truth

CHALLENGES WE FACED

Database - high risk decision

Data flow management

Multiple sources of truth

.... and Microservices

CQRS

02

CQRS IS ARCHITECTURAL PATTERN FOR REPRESENTING DATA IN **DIFFERENT PERSISTENT MODELS**

ARCHITECTURAL PATTERN THAT ENABLES REPRESENTATION OF DATA IN MULTIPLE PERSISTENT MODELS

PERSISTENT MODELS

Relational

Document

Search Index

Key / Value

Graph

Event Log

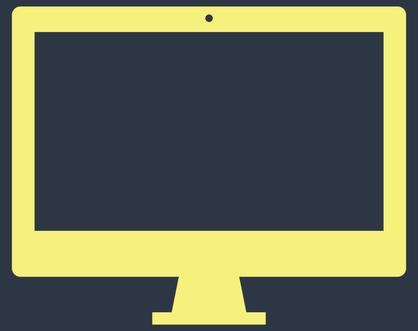
etc

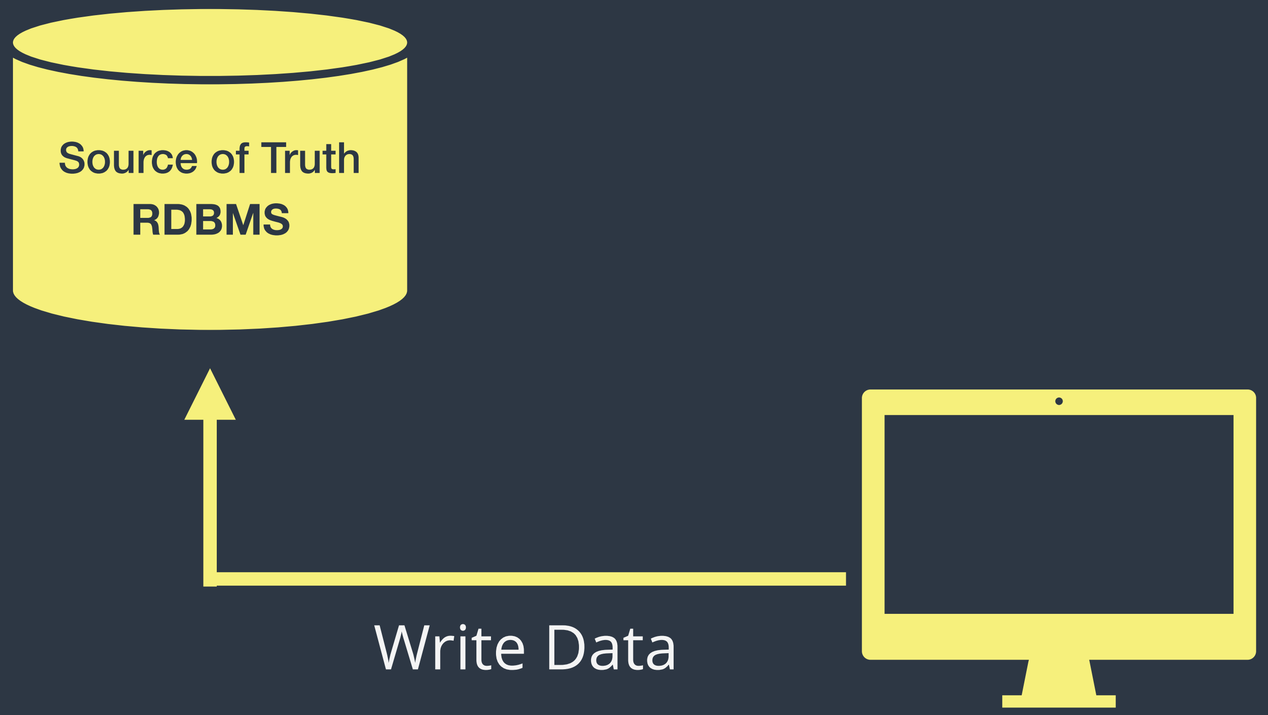
COMMAND QUERY RESPONSIBILITY SEGREGATION

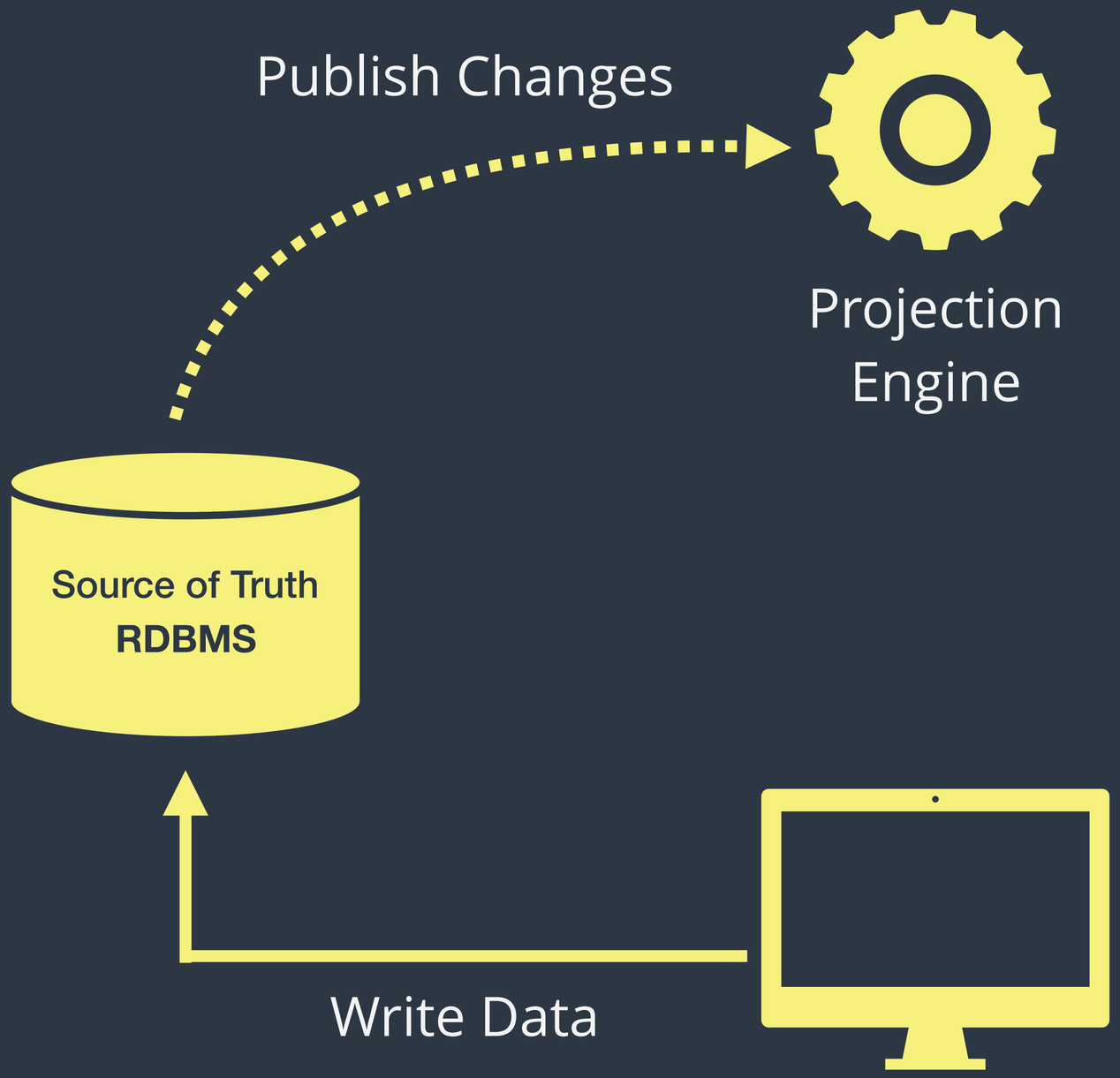
1. Segregate the responsibilities of writing and reading data

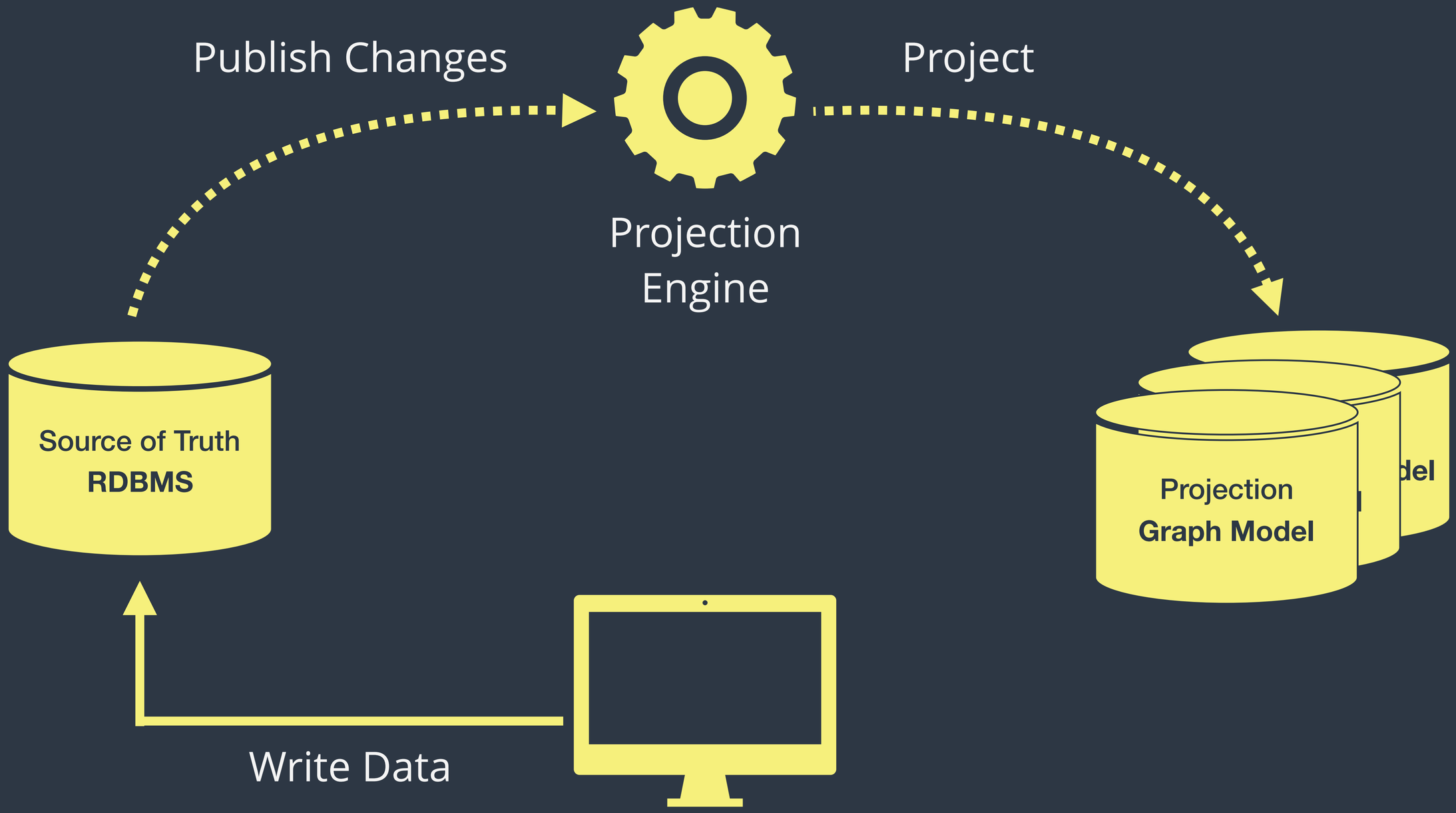
2. One model dedicated for writing data (executing commands)

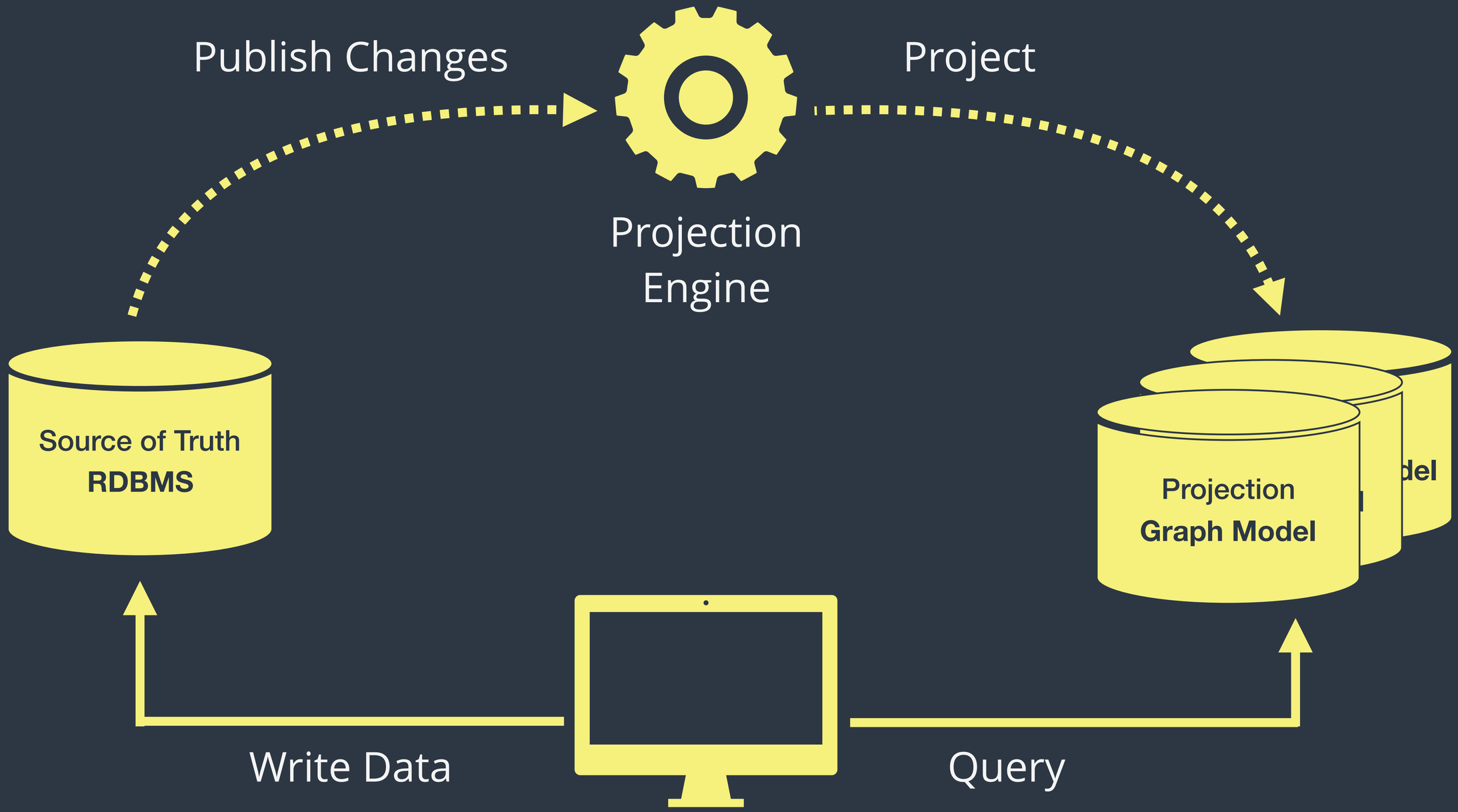
3. Multiple models for reading data (querying)

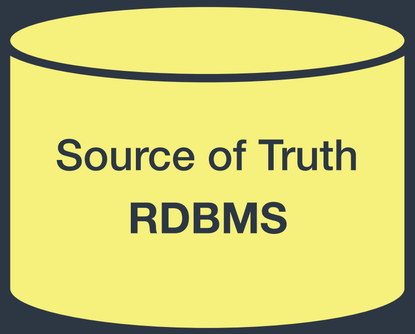



Source of Truth
RDBMS
Write Data


Publish Changes
Projection
Engine
Source of Truth
RDBMS
Write Data


Publish Changes
Project
Projection
Engine
Source of Truth
RDBMS
Projection
Graph Model
Write Data

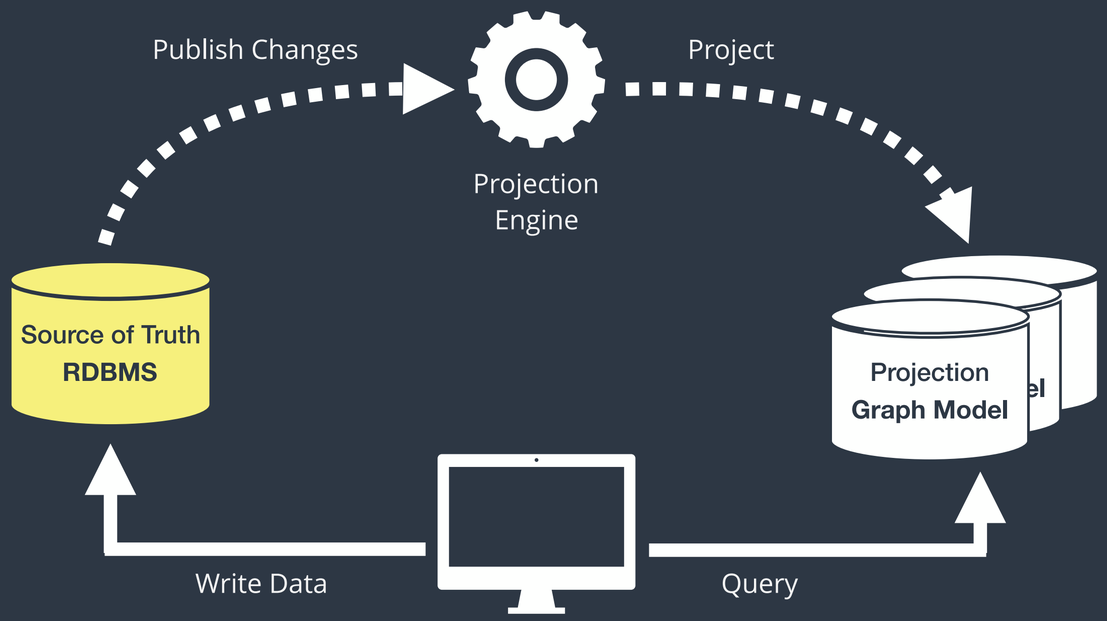

Publish Changes
Project
Projection
Engine
Source of Truth
RDBMS
Projection
Graph Model
Write Data
Query



Scale Requirements

Concurrency Control

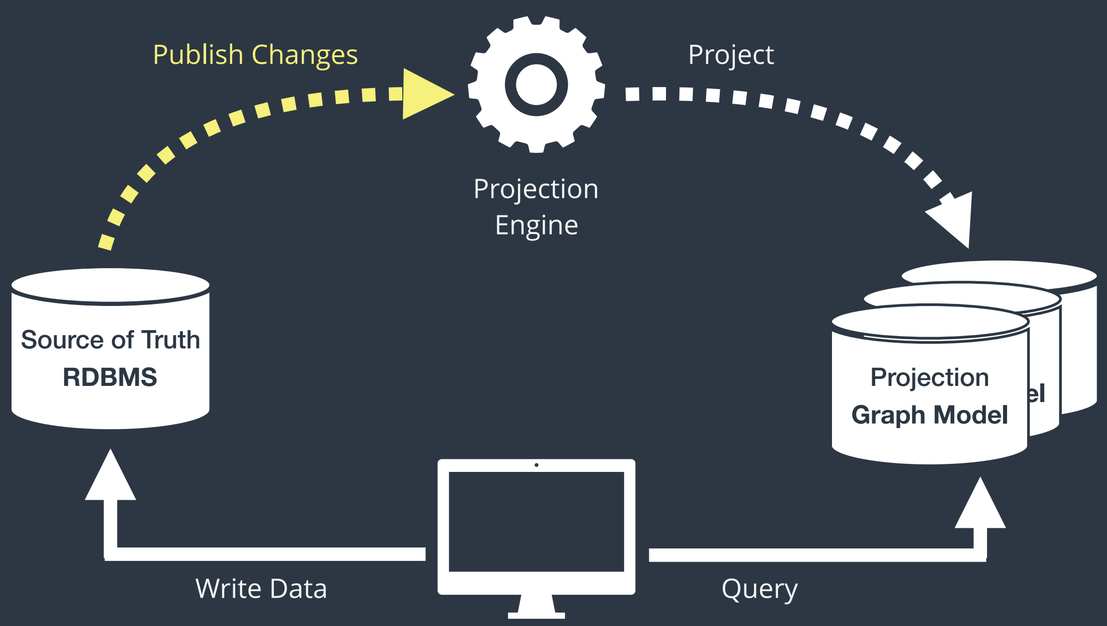
Synchronous Enumeration of Records



Supports Scale Requirements

Concurrency Control

Synchronous Enumeration of Records

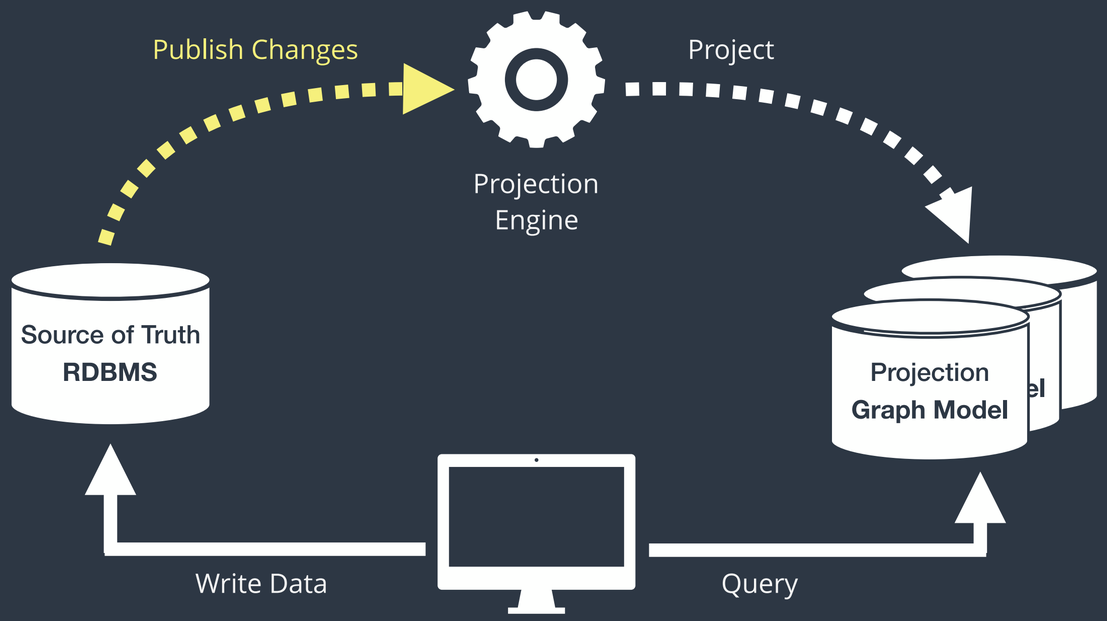


Publishing

- Synchronous
- Asynchronous

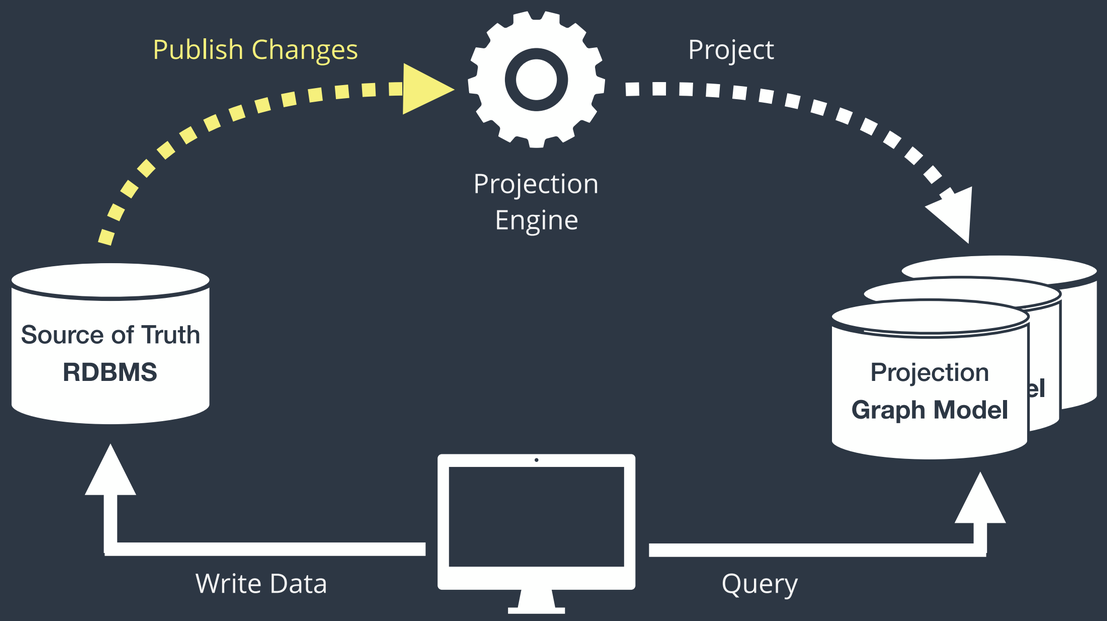
Changes

- Last snapshot
- Full



Synchronous

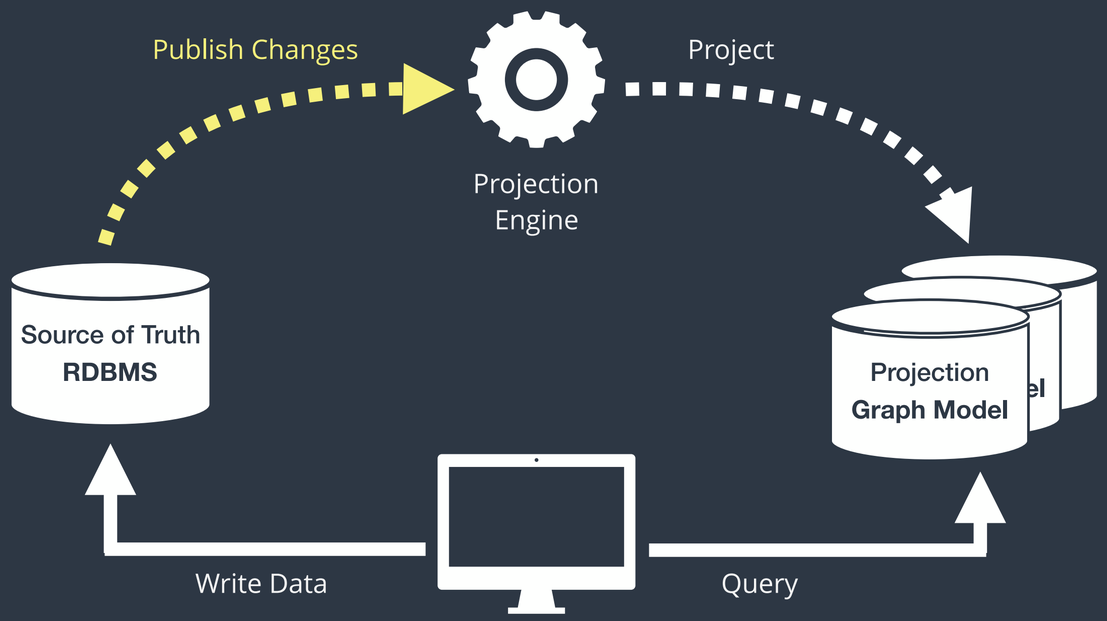
- Timestamp based enumeration
- Also known as *“Catch-up Subscription”*
- *Required*



Synchronous

- Timestamp based enumeration
- Also known as *“Catch-up Subscription”*
- *Required*

/GetChangesSince?timestamp=1087

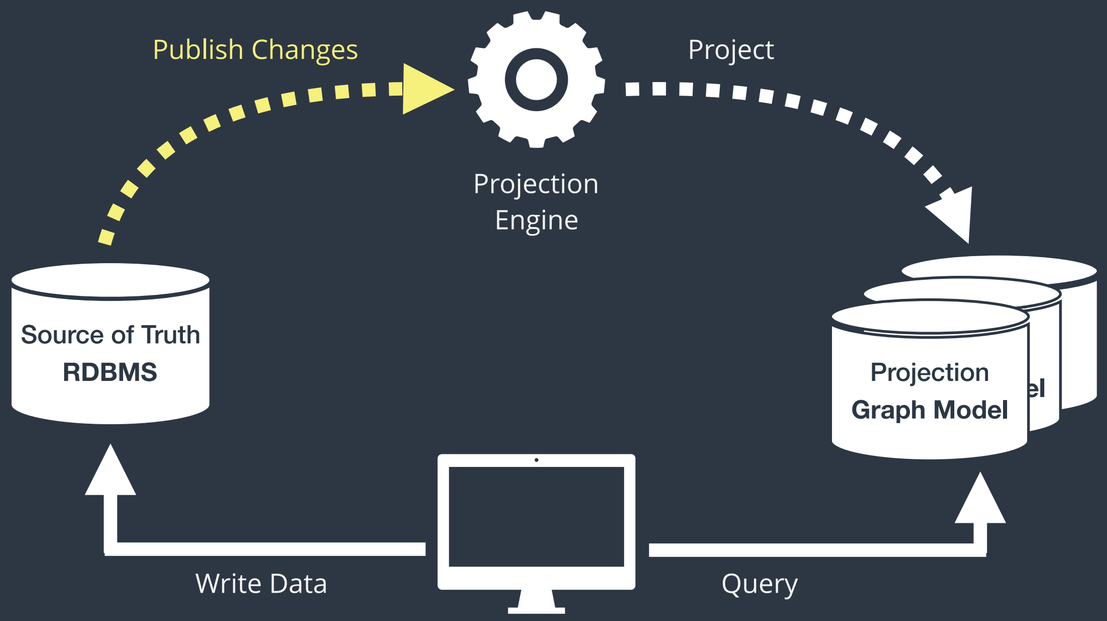


Synchronous

- Timestamp based enumeration
- Also known as *“Catch-up Subscription”*
- *Required*

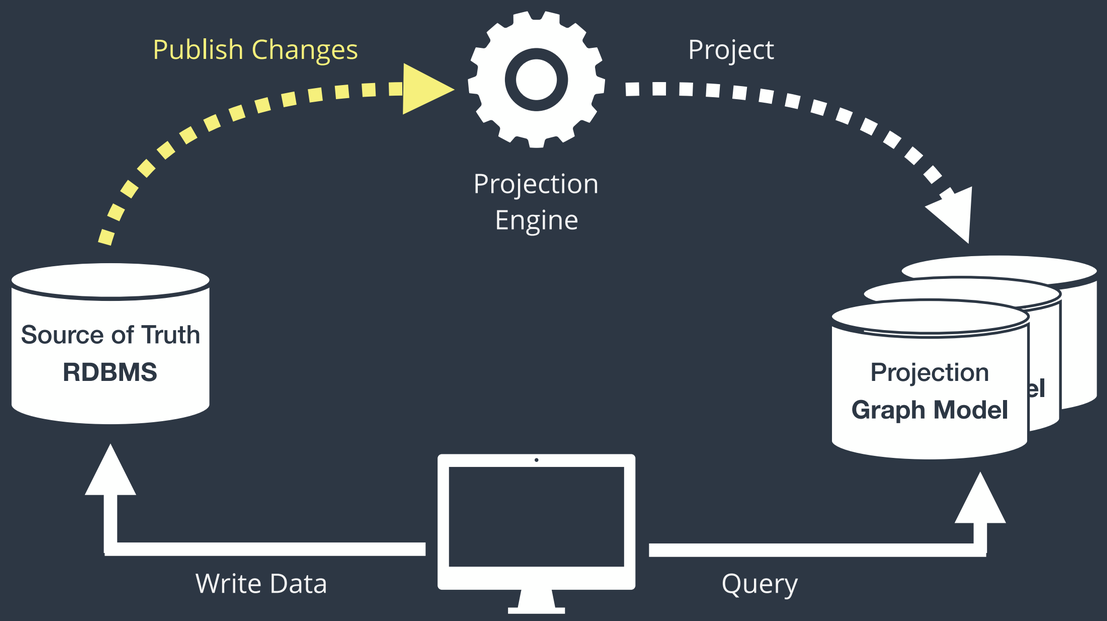
/GetChangesSince?timestamp=1087

employee_id	name	manager_id	**last_timestamp**
842	Daniel Cormier	510	**1088**
357	Nate Diaz	305	**1089**
408	Jake Shields	305	**1090**
512	Jon Jones	208	**1091**



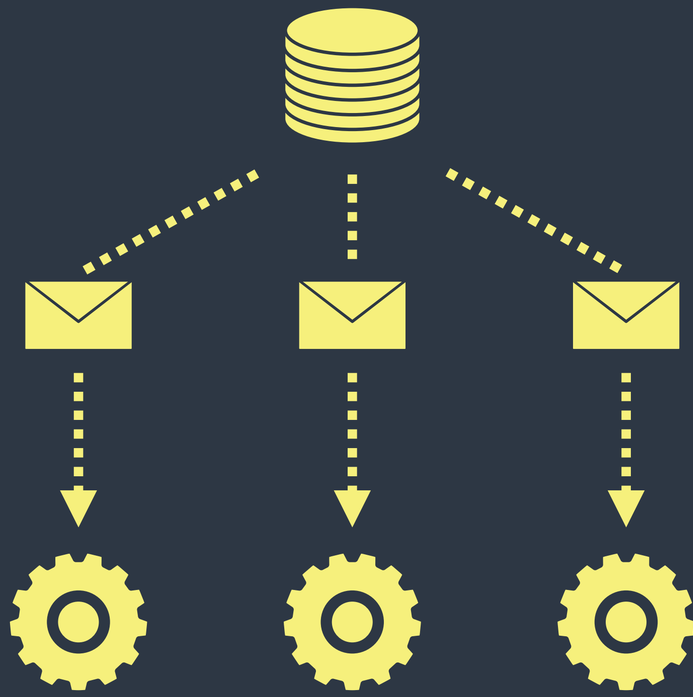
Asynchronous

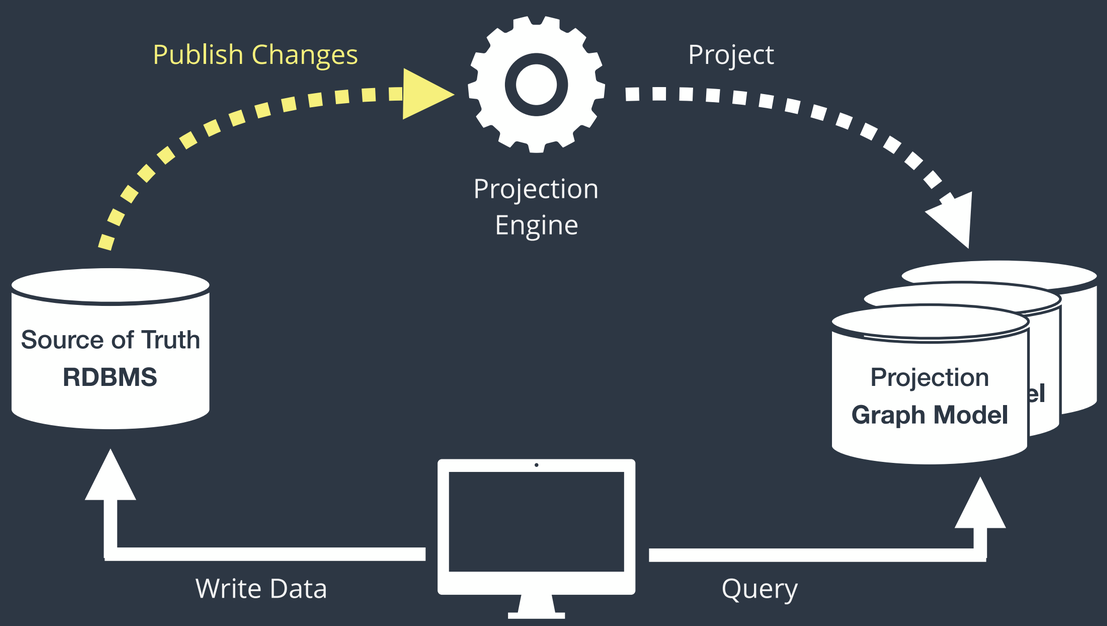
- Publishing to a messaging mechanism (stream / queue / topic)
- *Optional*



Asynchronous

- Publishing to a messaging mechanism (stream / queue / topic)
- *Optional*





Publishing

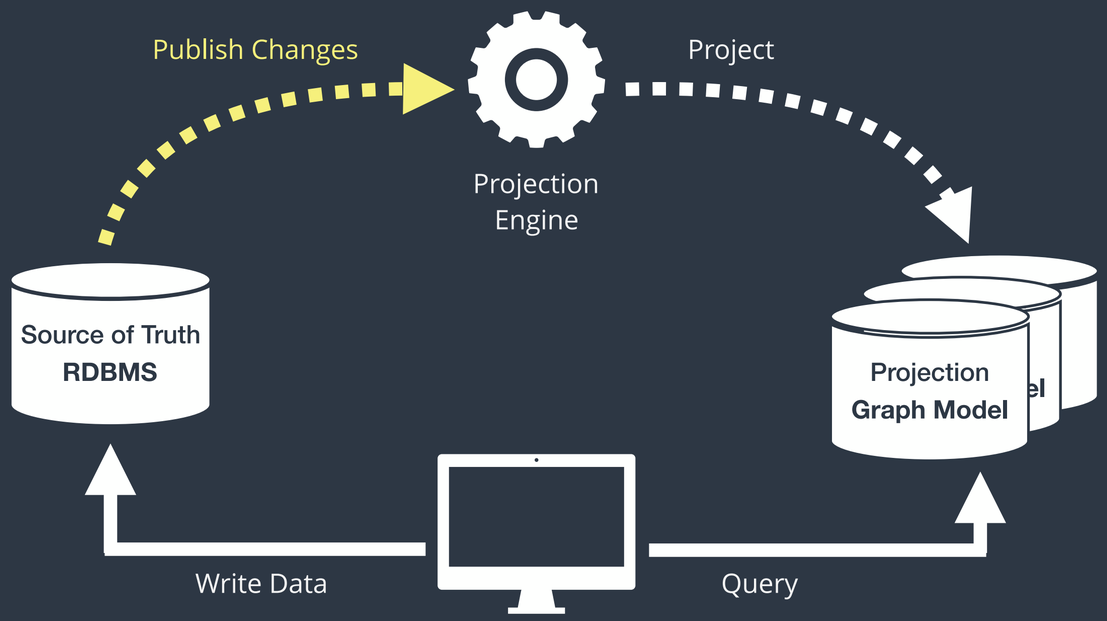
- Synchronous
- Asynchronous

Changes

- Last snapshot
- Full

Publish Last Snapshot

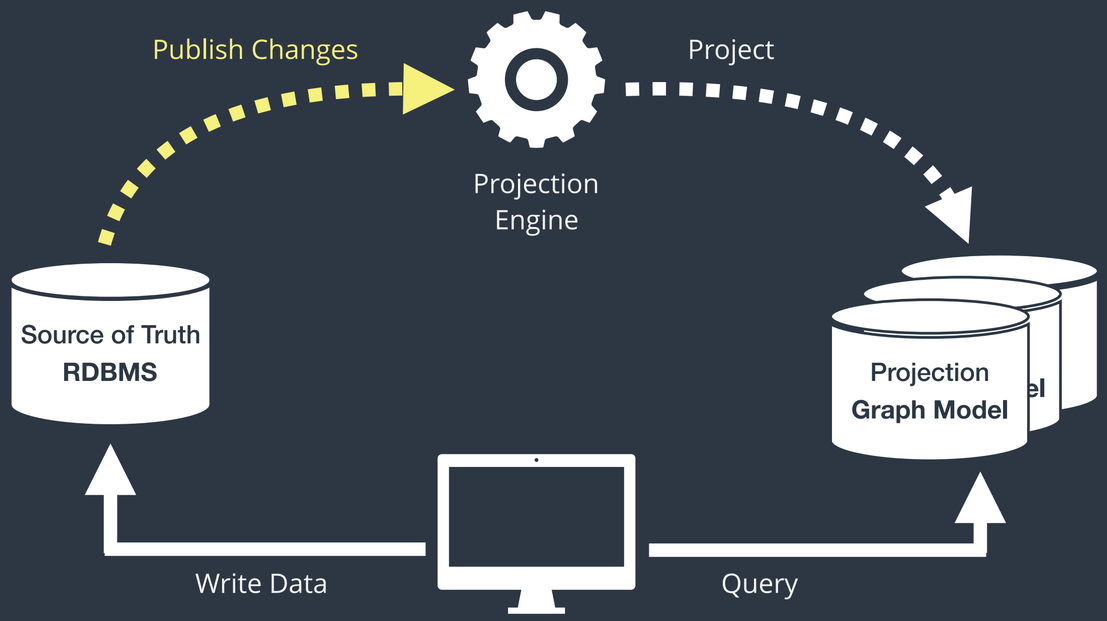
- Only the last state of each record is retrievable by catch-up subscription



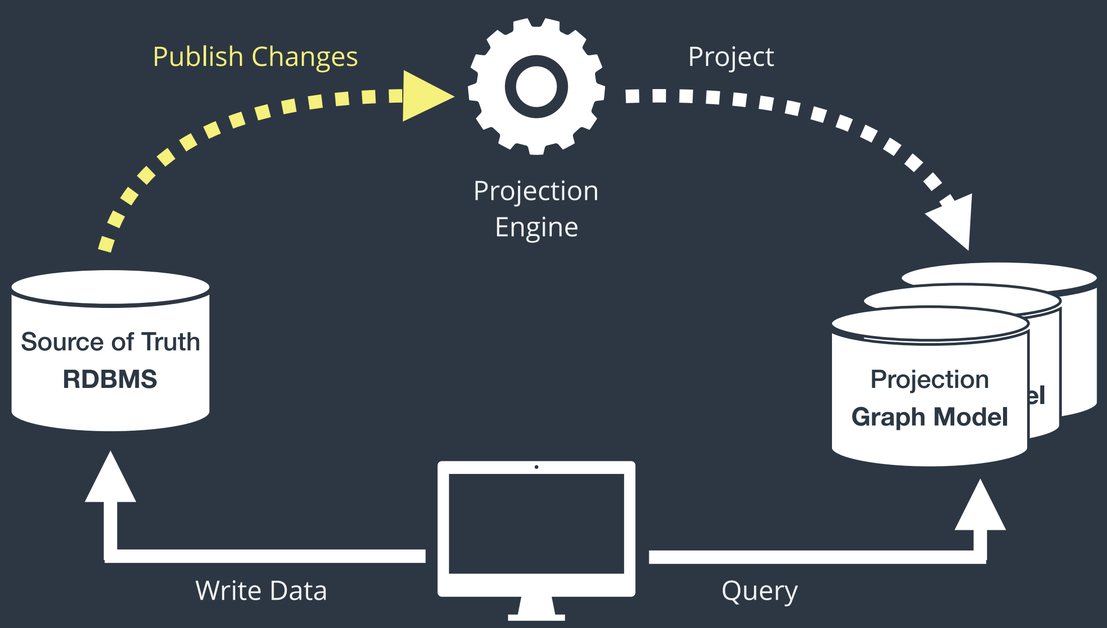
employee_id	name	manager_id	**last_timestamp**
842	Daniel Cormier	510	**1088**
357	Nate Diaz	305	**1089**
408	Jake Shields	305	**1090**
512	Jon Jones	208	**1091**

Publish Full History of Changes

- Full record of changes in each record's state



employee_id	name	manager_id	**revision**	**last_timestamp**
842	***Danel Cormer***	***510***	***1***	***1088***
842	***Daniel Cormier***	***510***	***2***	***1089***
512	Jon Jones	208	**1**	**1090**
842	***Daniel Cormier***	***305***	***3***	***1091***

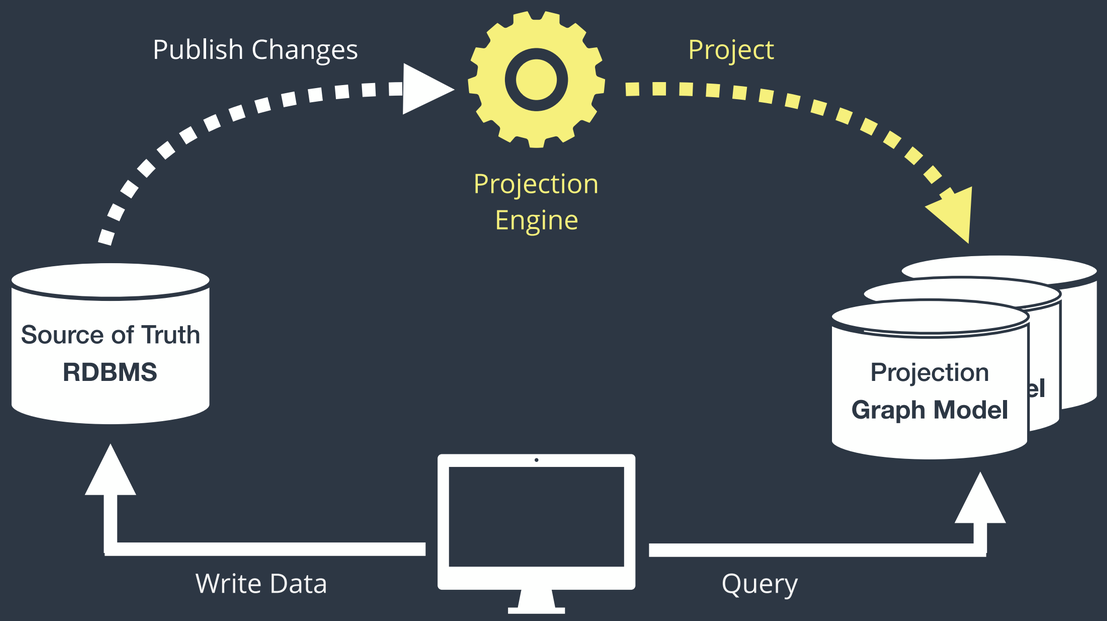


Synchronous

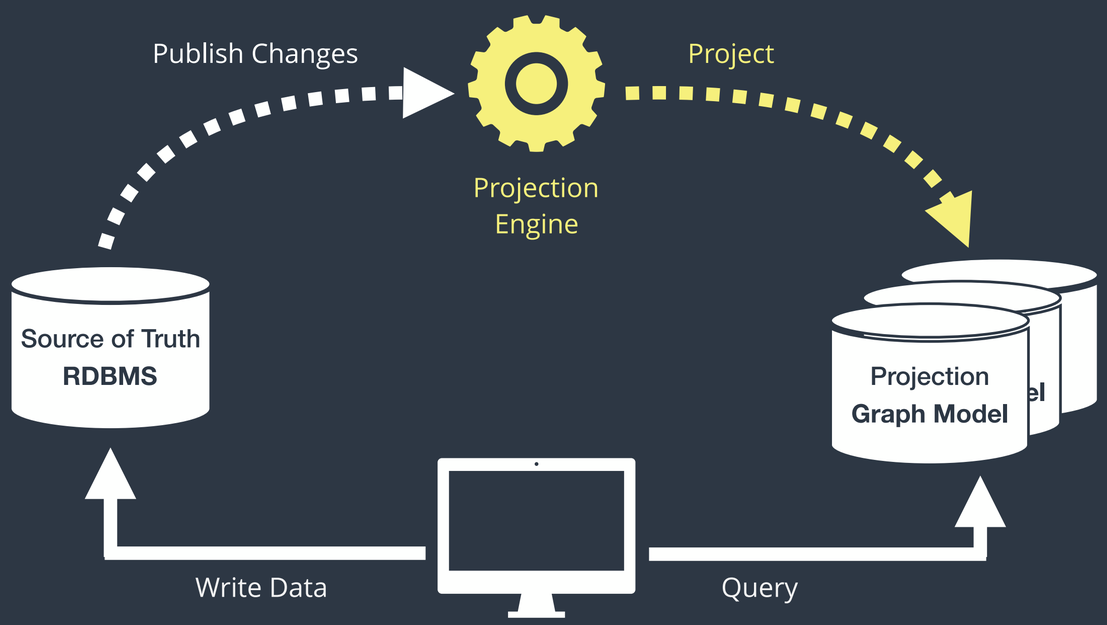
- Timestamp based enumeration
- *Required*

Asynchronous

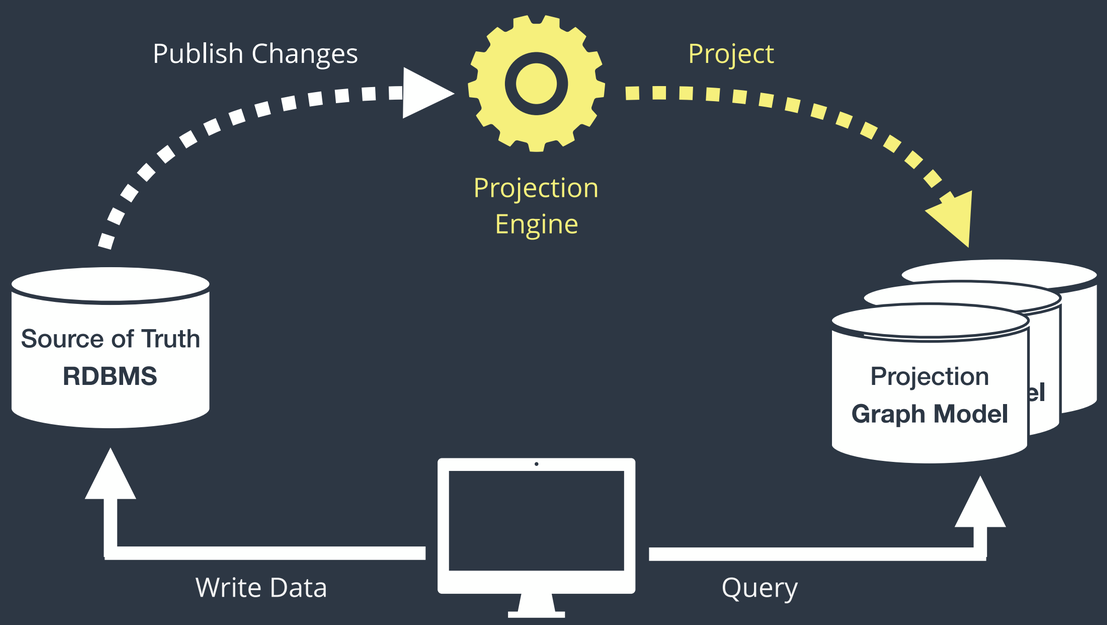
- Publishing to a messaging mechanism (stream / queue / topic)
- *Optional*



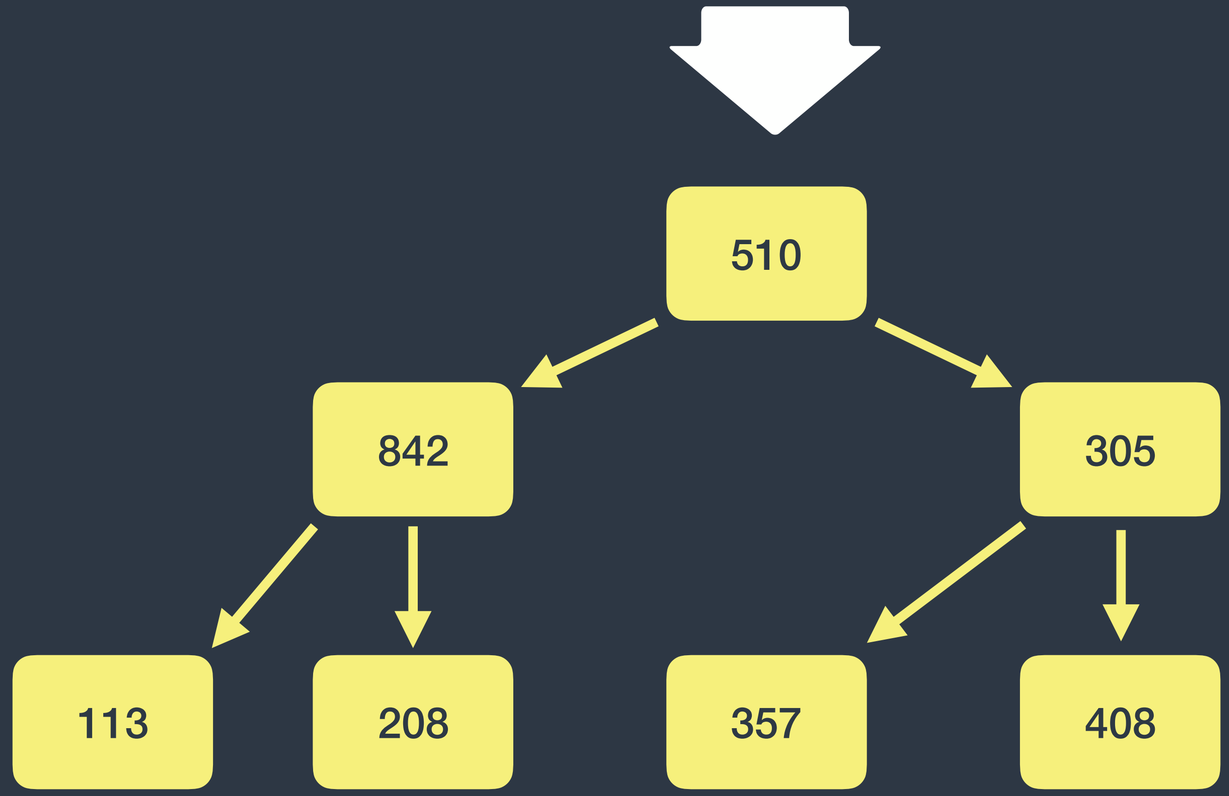
Transforms data to other models

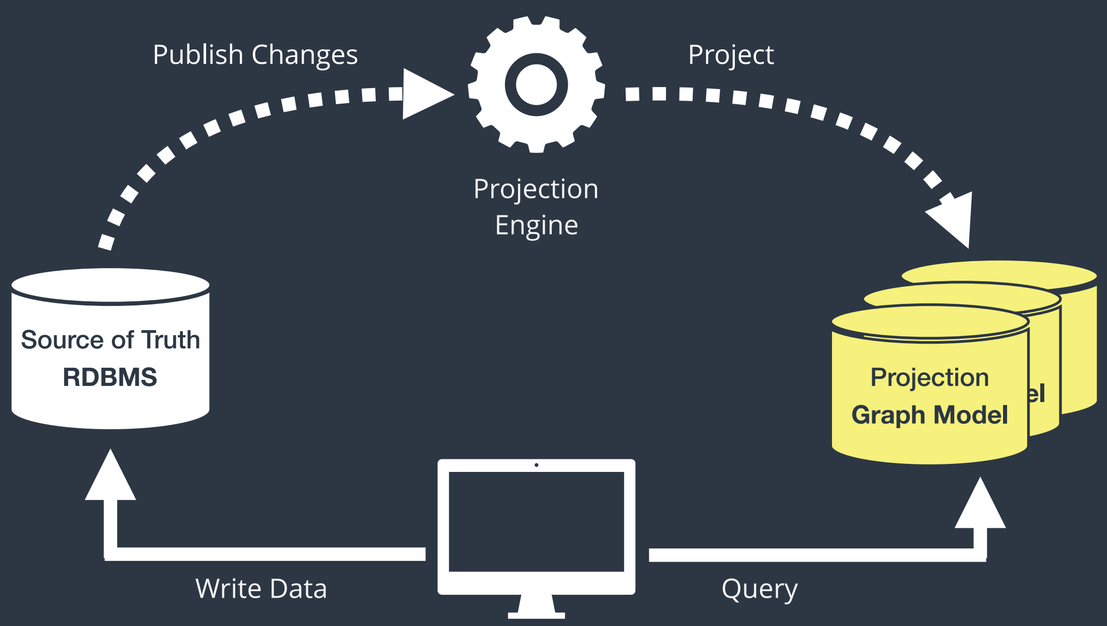


employee_id	name	manager_id	**last_timestamp**
842	Daniel Cormier	510	**1088**
357	Nate Diaz	305	**1089**
408	Jake Shields	305	**1090**
512	Jon Jones	208	**1091**



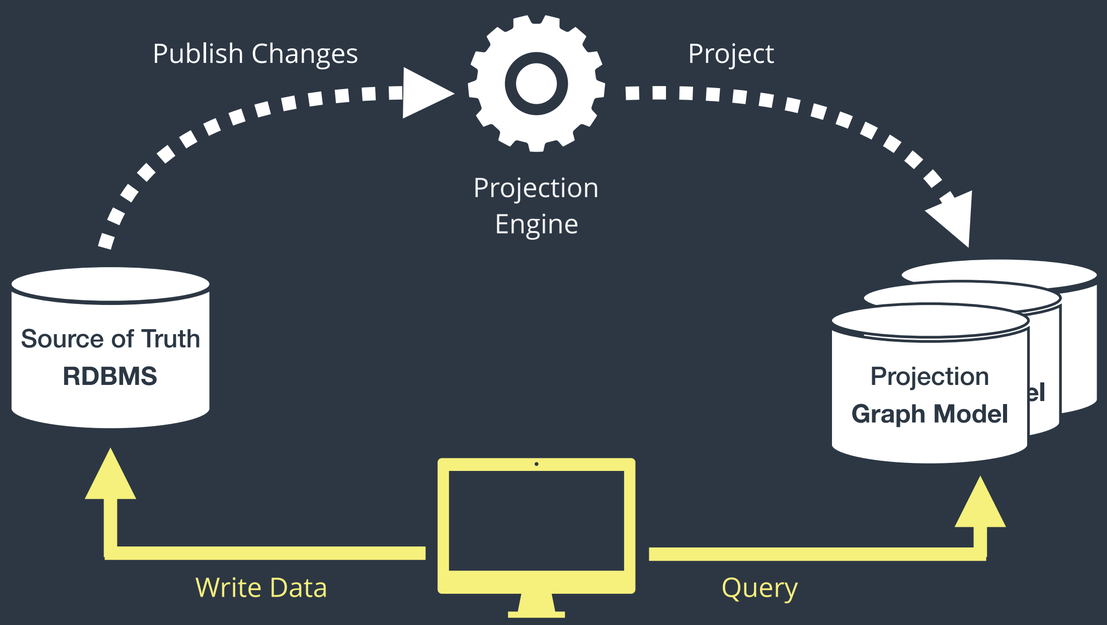
employee_id	name	manager_id	**last_timestamp**
842	Daniel Cormier	510	**1088**
357	Nate Diaz	305	**1089**
408	Jake Shields	305	**1090**
512	Jon Jones	208	**1091**





Projected Models

- Also known as *“Projections”*
- Eventually Consistent
- Naturally scalable
- **Cache**
- Can be deleted and rebuilt
- Flexible querying
- New projections can be added

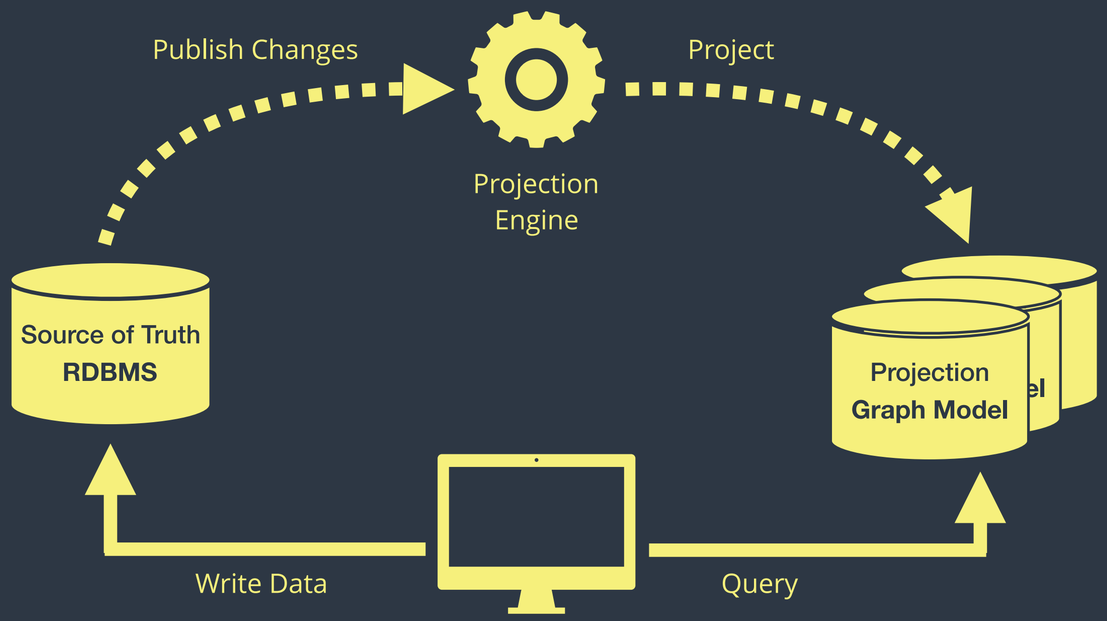


Source of Truth

- Used for writing data (executing *commands*)
- Strongly consistent

Projections

- Read only data
- Eventually consistent



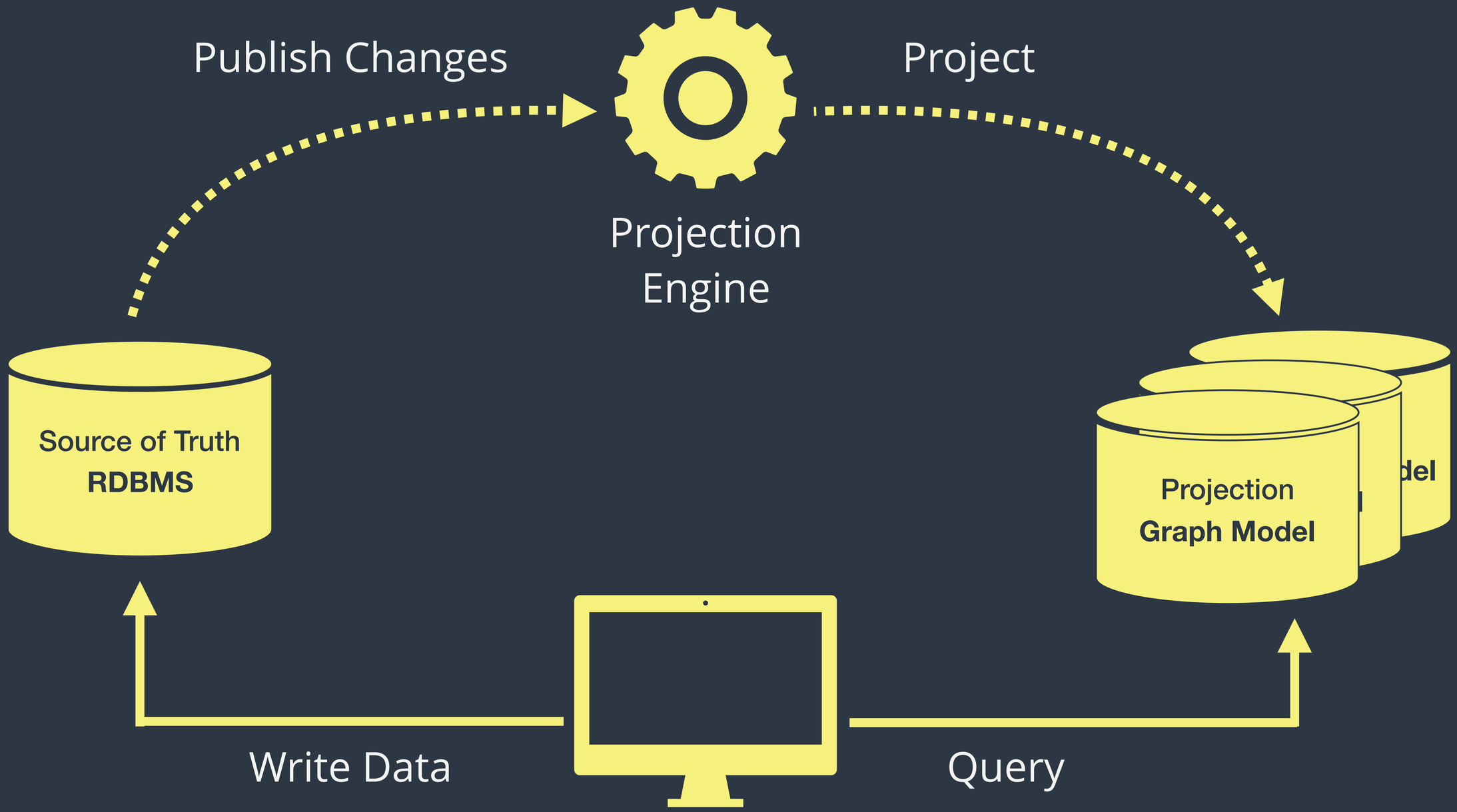
CQRS

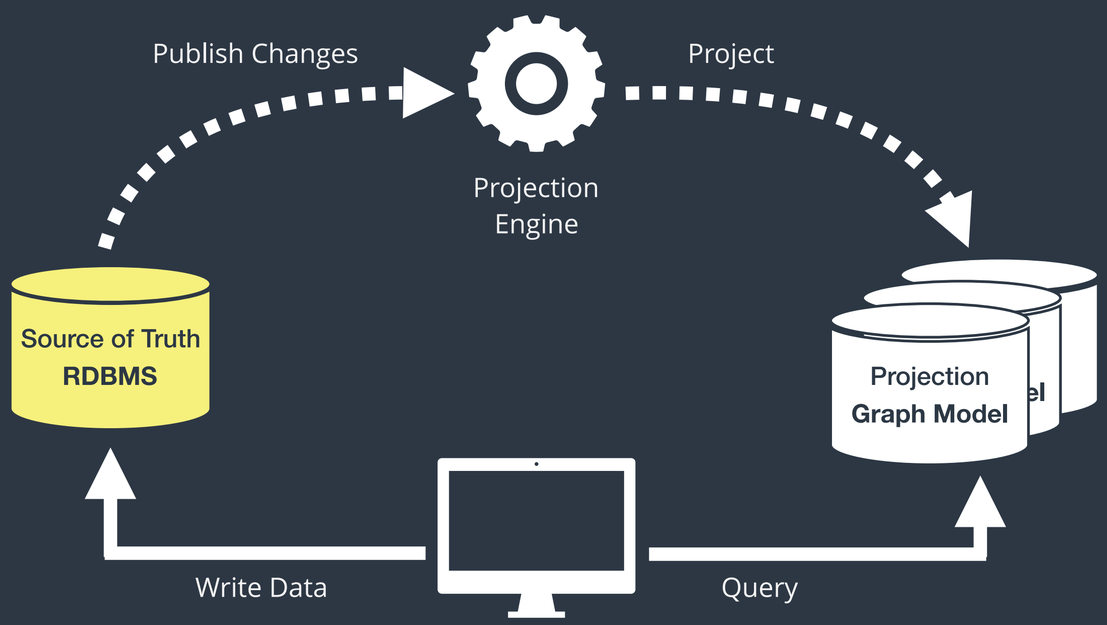
Command / Query
Responsibility Segregation

CQRS

INTERNOVUS EXPERIENCE REPORT

03

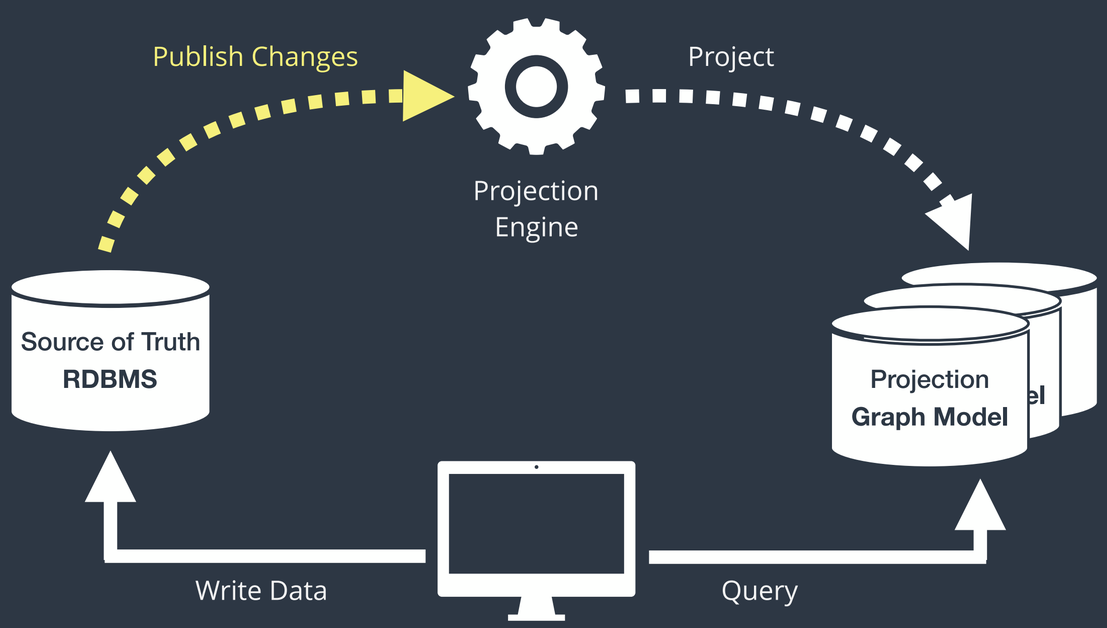

Publish Changes
Project
Projection
Engine
Source of Truth
RDBMS
Projection
Graph Model
Write Data
Query



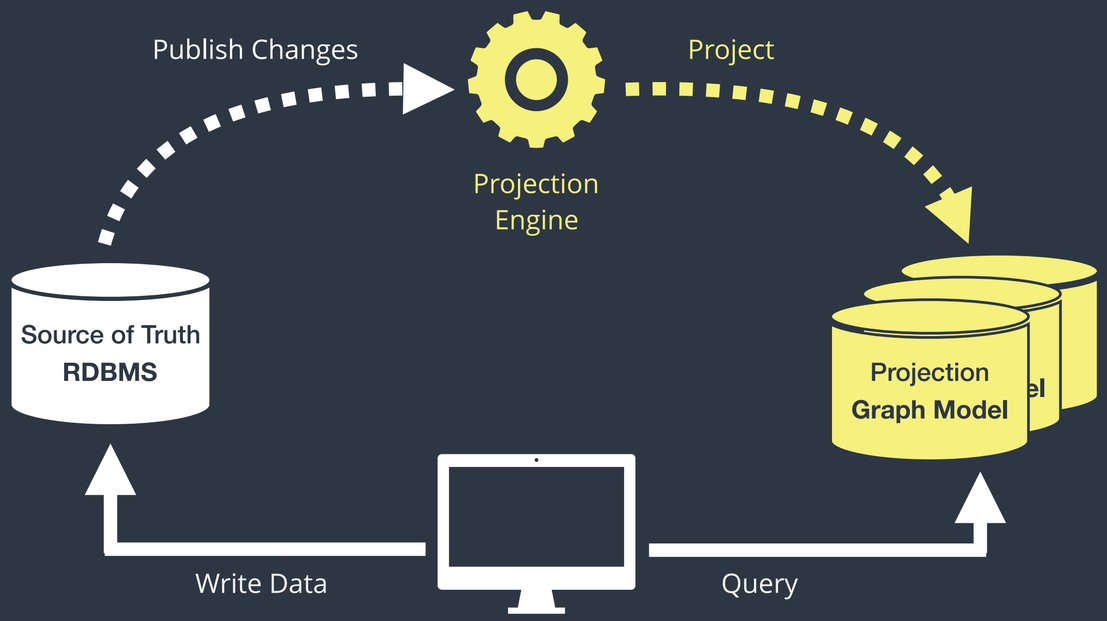
Implementing catch-up subscriptions is not always trivial.
Only logical deletes.

We used:

- DynamoDB
- S3 (in *very* special cases only)
- PostgreSQL / MSSQL / MySQL



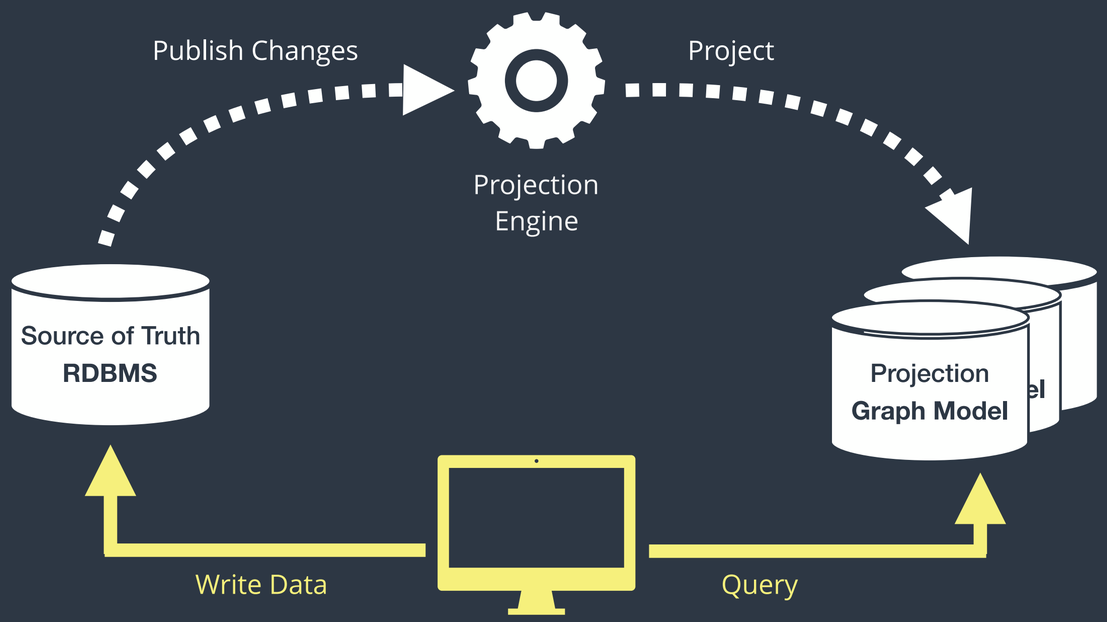
- Asynchronous publishing is not trivial
- Messages go out of order, get duplicated, etc’
- Stream processing systems (Kaffka, Kinesis) are useful
- Should always work in parallel w/ catch-up subscription (synchronous)
- Catch-up subscription is required for rebuilding projections from scratch



We projected into:

- RDBMS (MySQL, PostgreSQL, etc)
- Columnar Stores (Redshift)
- Flat files (S3)
- Search indexes (Elasticsearch)

Idempotency is crucial!



- Void commands make no sense!
- Command can return its status - whether it succeeded or failed
- Command can return reasons for failure
- Command can return data from the writing model
- Querying a projection cannot modify any data

POLYGLOT PERSISTENCE CHALLENGES

Database - high risk decision

Data flow management

Multiple sources of truth

.... and Microservices

POLYGLOT PERSISTENCE CHALLENGES

Database - high risk decision

Data flow management

Multiple sources of truth

.... and Microservices

Projected Models

- Freedom to experiment
- New models can be added anytime

Source of Truth Model

- More options - lower requirements
- Easier to refactor if needed

POLYGLOT PERSISTENCE CHALLENGES

Database - high risk decision

Data flow management

Multiple sources of truth

…. and Microservices

- Projections are easy to manage
- Easier refactoring
- Ubiquitous integration architecture

POLYGLOT PERSISTENCE CHALLENGES

Database - high risk decision

Data flow management

Multiple sources of truth

.... and Microservices

- One source of truth for all data
- Data can be extended in different contexts
- Read-only projections serve as model boundaries

POLYGLOT PERSISTENCE CHALLENGES

Database - high risk decision

Data flow management

Multiple sources of truth

.... and Microservices

- Boundaries are crucial
- Projections “absorb” changes in models
- Reduce coupling between microservices

BUT...
EVENT SOURCING?

04

EVENT SOURCING IS NOT CQRS

CQRS IS NOT EVENT SOURCING

CQRS IS ARCHITECTURAL PATTERN FOR REPRESENTING DATA IN DIFFERENT PERSISTENT MODELS

EVENT SOURCING IS A BUSINESS LOGIC MODELING PATTERN THAT ALLOWS TO MODEL THE DIMENSION OF TIME

STATE-BASED MODEL

employee_id	name	manager_id	**last_timestamp**
842	Daniel Cormier	510	**1088**
357	Nate Diaz	305	**1089**
408	Jake Shields	305	**1090**
512	Jon Jones	208	**1091**

EVENT SOURCING

All changes to the application’s state should be represented as a series of domain events

EVENT SOURCING

1. { “employee-id”: 842, “event”: “employee-initialized”, “manager-id”: 302 }
2. { “employee-id”: 842, “event”: “name-changed”, “name”: “Danel Cormer” }
3. { “employee-id”: 842, “event”: “manager-changed”, “id”: 510 }
4. { “employee-id”: 842, “event”: “name-changed”, “name”: “Daniel Cormier” }

EVENT SOURCING

1. { “employee-id”: 842, “event”: “employee-initialized”, “manager-id”: 302 }
2. { “employee-id”: 842, “event”: “name-changed”, “name”: “Danel Cormer” }
3. { “employee-id”: 842, “event”: “manager-changed”, “id”: 510 }
4. { “employee-id”: 842, “event”: “name-changed”, “name”: “Daniel Cormier” }

EVENT SOURCING

Time

1. { “employee-id”: 842, “event”: “employee-initialized”, “manager-id”: 302 }
2. { “employee-id”: 842, “event”: “name-changed”, “name”: “Danel Cormer” }
3. { “employee-id”: 842, “event”: “manager-changed”, “id”: 510 }
4. { “employee-id”: 842, “event”: “name-changed”, “name”: “Daniel Cormier” }

employee_id	842
name	Daniel Cormier
manager_id	510
last_timestamp	1088

EVENT SOURCING → CQRS

EVENT SOURCING → CQRS

CQRS ≠ EVENT SOURCING

SUMMARY

05

- Polyglot persistence is essential for solving Big Data problems
- CQRS alleviates polyglot persistence
  - By dedicating a model for writing data
  - Variety of different models can be projected out of the writing model
  - Segregating writing and reading models reduces architectural risks and allows implementing additional models
- CQRS is NOT Event Sourcing

THANK YOU!

QUESTIONS?

THANK YOU!

 @vladikk

 vladikk.com