Paweł Rajba
pawel@cs.uni.wroc.pl
http://pawel.ii.uni.wroc.pl/

CQRS and Event Sourcing

Agenda

- CQRS
  - A common approach
  - Nature of business applications
  - CQS and CQRS
  - Why to create separate models?
  - A few more things about commands
  - Levels of segregation
  - Database synchronization
  - Eventual consistency
  - Consideration
  - Benefits
- Event sourcing

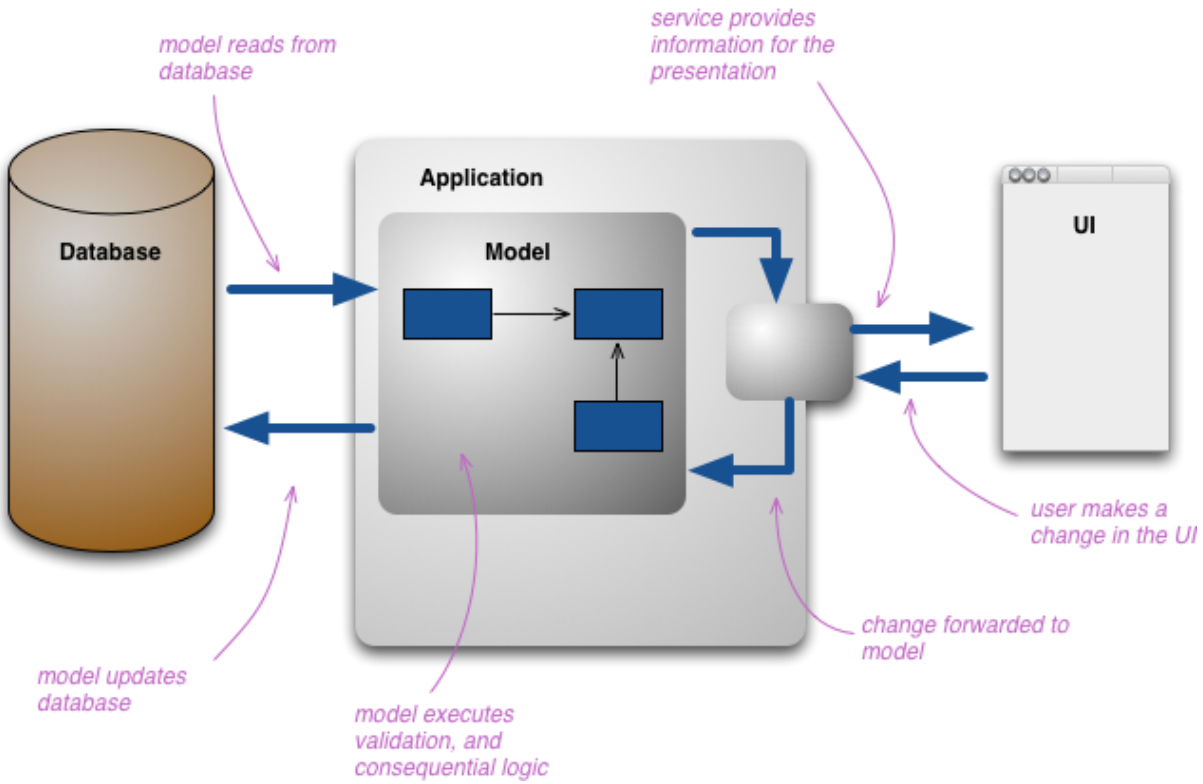
CQRS

A common approach

- What is specific in that approach? If we focus a specific bounded context, then we have…
  One model for all types of operations
- If we focus a specific part of UI, then again we have…
  One model for all types of operations
- Although in most cases it is ok, sometimes it doesn't…

The common approach

- From a flow perspective



Query of despair

Nature of business applications

- Business requirements usually can be represented by use cases
- Uses cases can be generally split into:
  - Ones in which user want to **modify** data
  - Ones in which user want to **search** and **read** data

Let’s introduce some definitions…

CQS

- First pattern which applies that observation is CQS Command-Query Separation

> *It states that every method should either be a command that performs an action, or a query that returns data to the caller, but not both. In other words, asking a question should not change the answer. (Wikipedia, CQS)*

- Godfather: Bertrand Meyer
- To put it simply, it separates methods for

Commands Queries

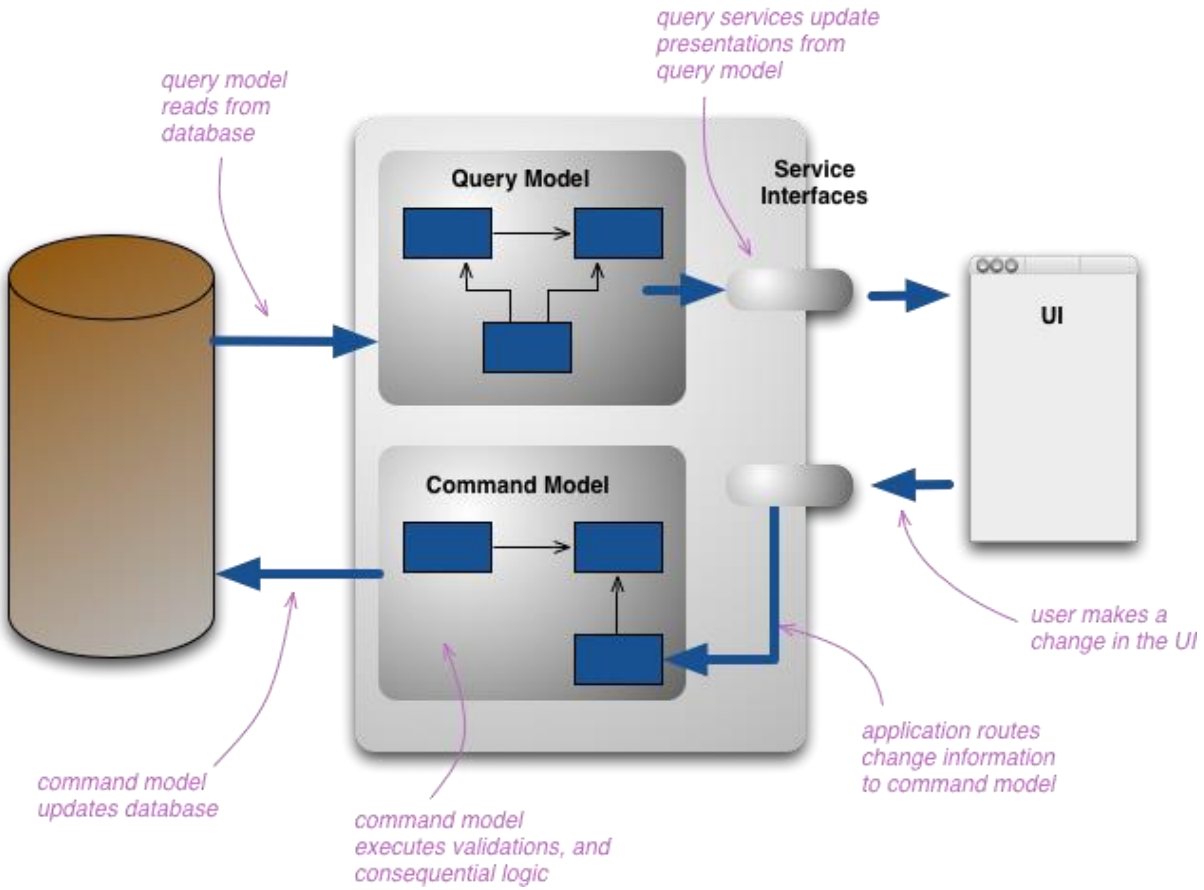
CQRS

- CQRS: Command-Query Responsiblity Segregation
- How to put it in a shorter way?

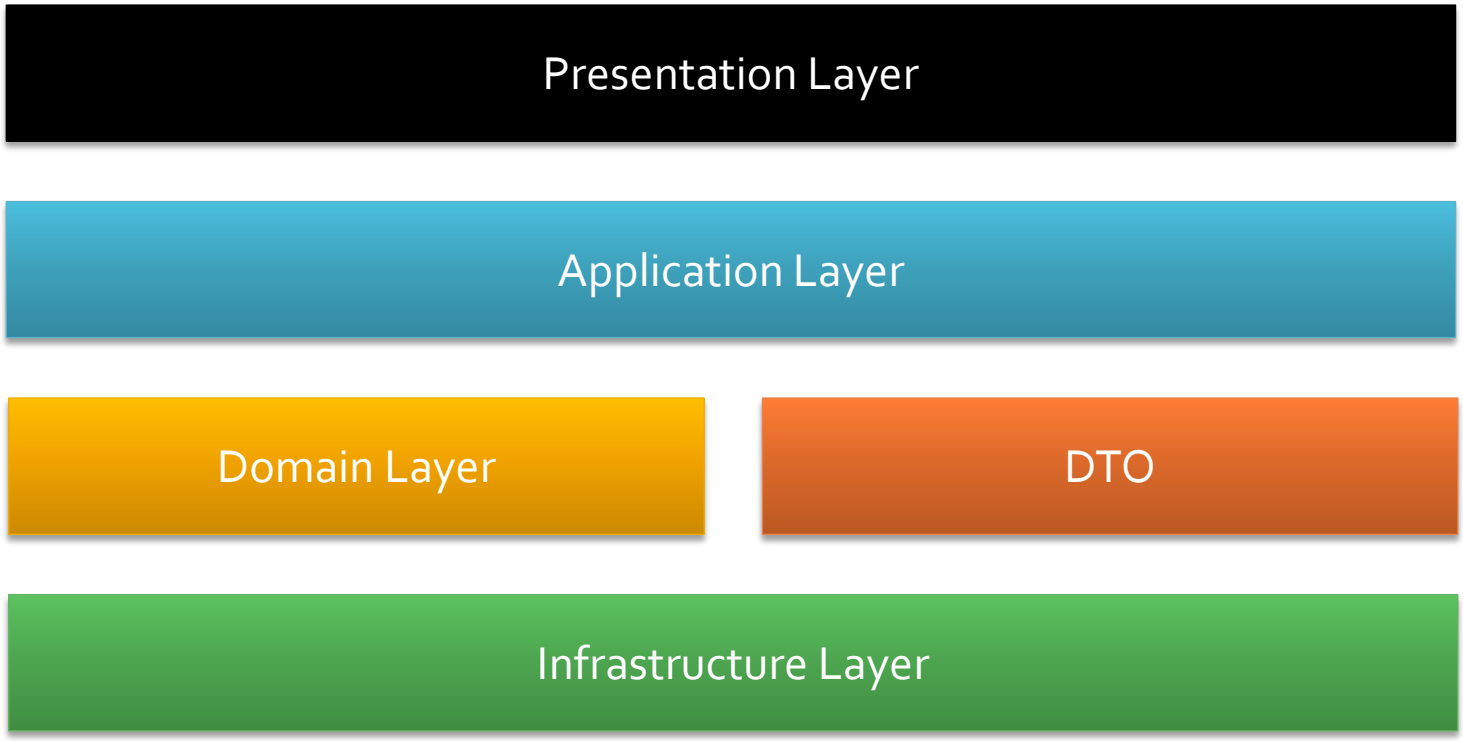
> *CQRS is simply the creation of two objects where there was previously only one*
>
> *CQRS by Greg Young*

- Godfathers: Greg Young and Udi Dahan

CQRS flows


query services update
presentations from
query model
query model
reads from
database
Query Model
Service
Interfaces
UI
Command Model
user makes a
change in the UI
application routes
change information
to command model
command model
updates database
command model
executes validations, and
consequential logic

CQRS architecture



Why to create separate models?

Command

- Targets a single Aggregate
- Validation rules
- Examples:
  - CreatePost, AddComment, SubmitOrder, LockUser
- Optimized for update

Query

- Collapsing multiple records into one
- Forming virtual records by combining information for different places
- No validation needed
- Optimized for search

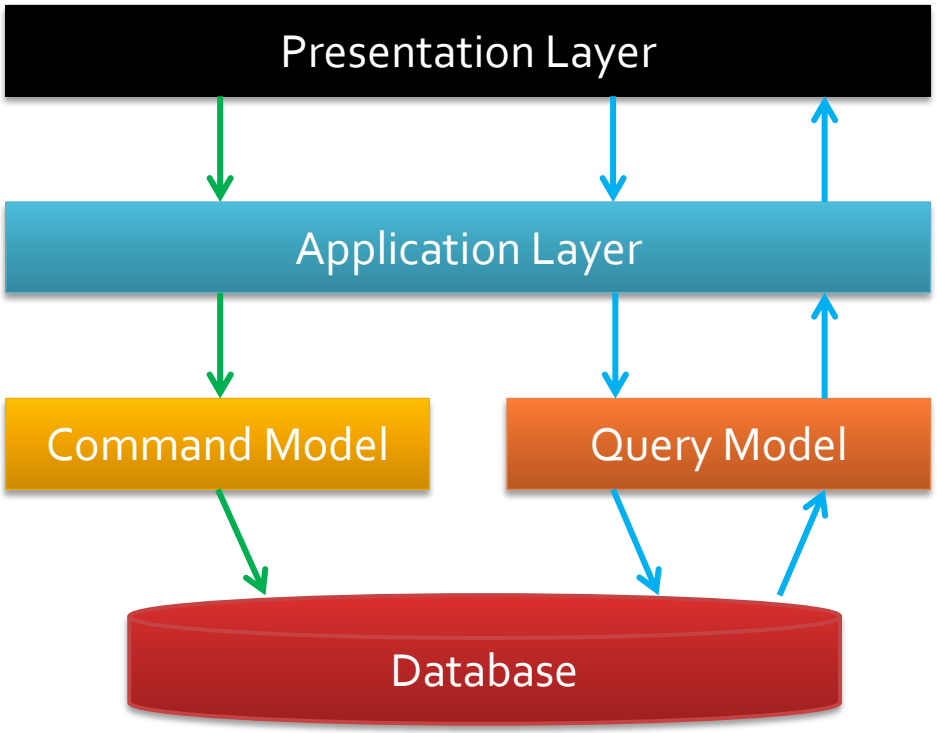
A few more things about commands

- What is command?
  - Actually is a request for change, a simple object
- Sometimes we expect that commands should be queued
- Another thing is that command should be able to process without quering for a data

But this is not only about models
– a new mindset is needed in creating UI

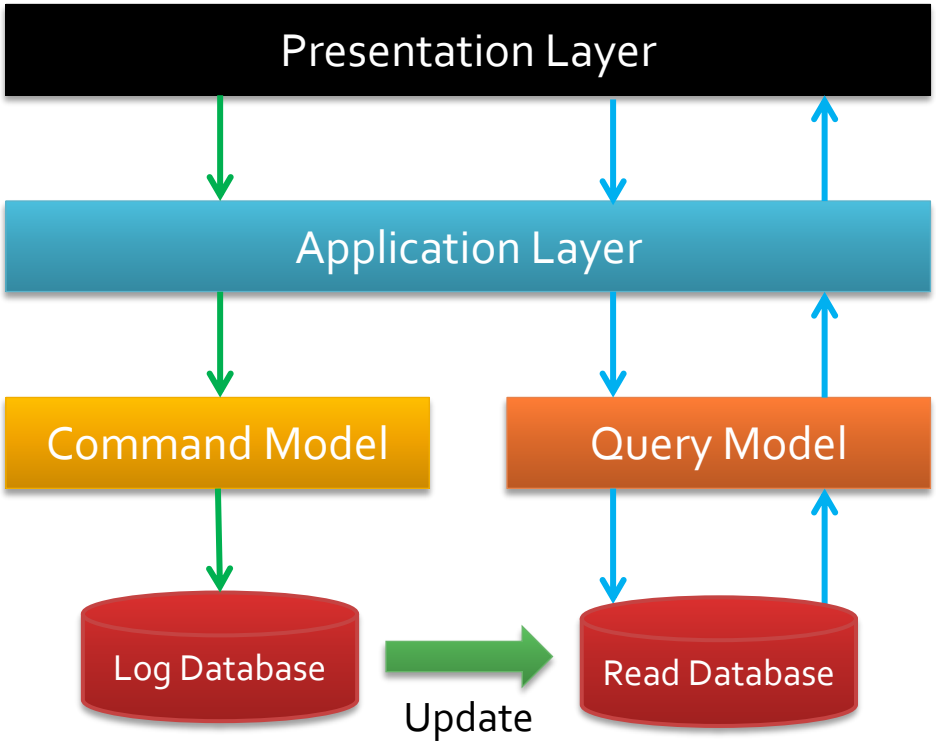
Levels of segregation

- **Model segregation**



Levels of segregation

- **Model & database segregation**



Database synchronization

Type	Consistency	Delay
Synchronous	Transactional	N/A
Asynchronous	Eventual	Short
Scheduled	Eventual	Long
On-demand	Eventual	Depends on event

Eventual consistency

- An oposite to the transactional consistency
- It guaranties that model become consistent

  at some point in time…

  … but not immediately.
- So, what are the delayes?
  - It depends: seconds, minutes, days…
- Very often encountered in aggregates considerations, but…
  - Can be found in other considerations, e.g. databases
- What if something go wrong?
  - Well, then sometimes we may have a big problem

Consideration

- Decision on seperation depends on the scenario and on **WHY** we need CQRS
- Example for 2 DBs
  - Write DB: Visits on a web site
  - Read DB: Aggregated statistics
- Synchronization between databases is usually based on events
- To avoid problems with concurrency as a part of architecture we can introduce
  - Command Handler/Bus
  - Event Bus

Benefits

- Different optimization strategies for command and queries
- Scalability
- In a collaborative environment, we can reduce concurrency problems by applying CQRS

Event Sourcing

Event Sourcing

- The way of thinking…
  - When you meet someone, most likely you are going to describe last 1 month by events, not by a state you are now
- Losing important knowledge
  - By design you assume what it is going to be kept
  - So, if you miss something in a design, you can't recover that
  - Having events allows you to conclude anything from the history

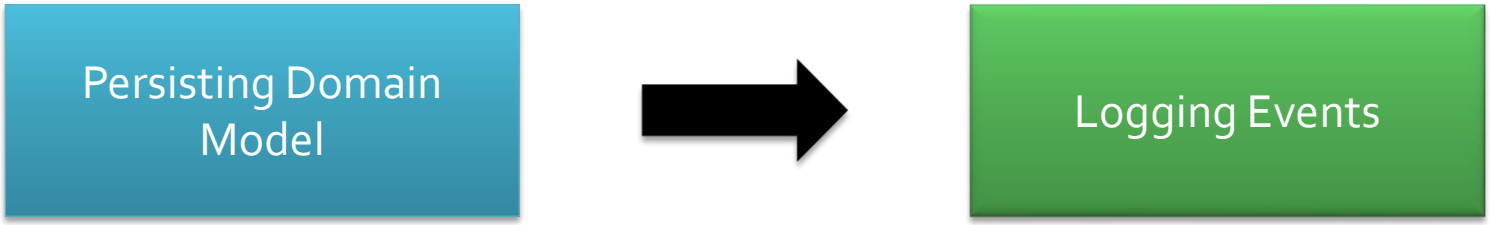
Events vs. Models

Event Sourcing

- Event describe something that happened in past
  - So, they are immutable
  - We can only append events, no delete, no update
  - E.g. order created, status changed
- Event sourcing
  - ...is a way of persisting your application's state by storing the history that determines the current state of your application.
  - In short: Events as a Data Store

Event Sourcing

- **Mindset change**



Event Sourcing

How to store data?

- Keep the current state
- Log events as a history

Concept familiar from the database world

- Keep events
- Build knowledge by reviewing relevant events

Common example: shopping cart

- Let's consider the following sequence of events
  1. Shopping Cart Created (ID=1)
  2. Item (ID=22) Added to Cart (ID=1)
  3. Item (ID=53) Added to Cart (ID=1)
  4. Item (ID=22) Removed from Cart (ID=1)
  5. Shopping Cart (ID=1) Checked-Out
- As we can observe, every event relates to some data, specifically entityID (Cart ID)

Showing data to a user

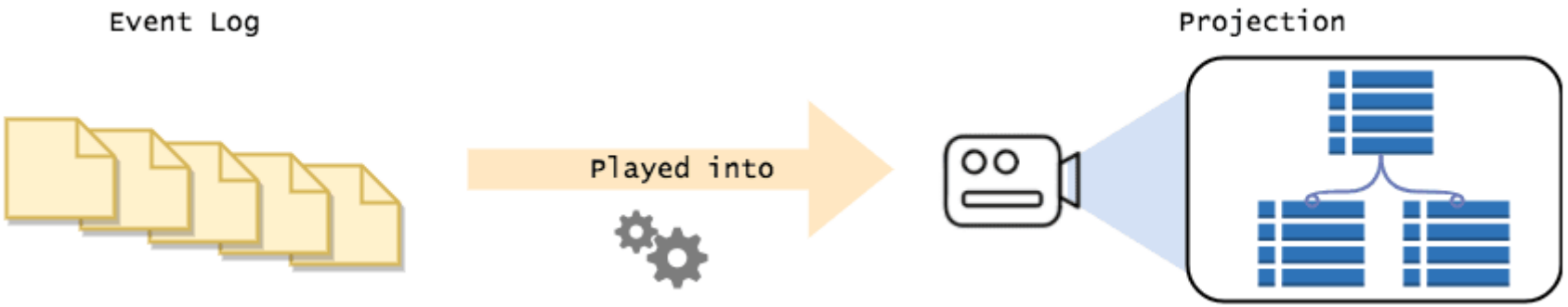
- Quering is based on replaying events
  - Get all events related to a specific aggregate (EntityID) by the method GetEventStream(ID)
  - Apply all events to the agregate instance
  - In the end, we get the current state
- We can replay for other reasons, e.g.
  - Statistical
  - Executing business rules (e.g. if an item is in the card, so you can remove the item)
- Events can be replicated for the scalability
  - They are immutable, so we can do that easily

Showing data to a user

- Log of events can only grow and reviewing all of them every time may affect performance
- To address that we can create a snapshot
  - Which serialize the state of aggregate at some point in time
  - A snapshot is (of course) attached to the event stream
- Afterwards, you replay the events from the snapshot, not from the beginning

Showing data to a user

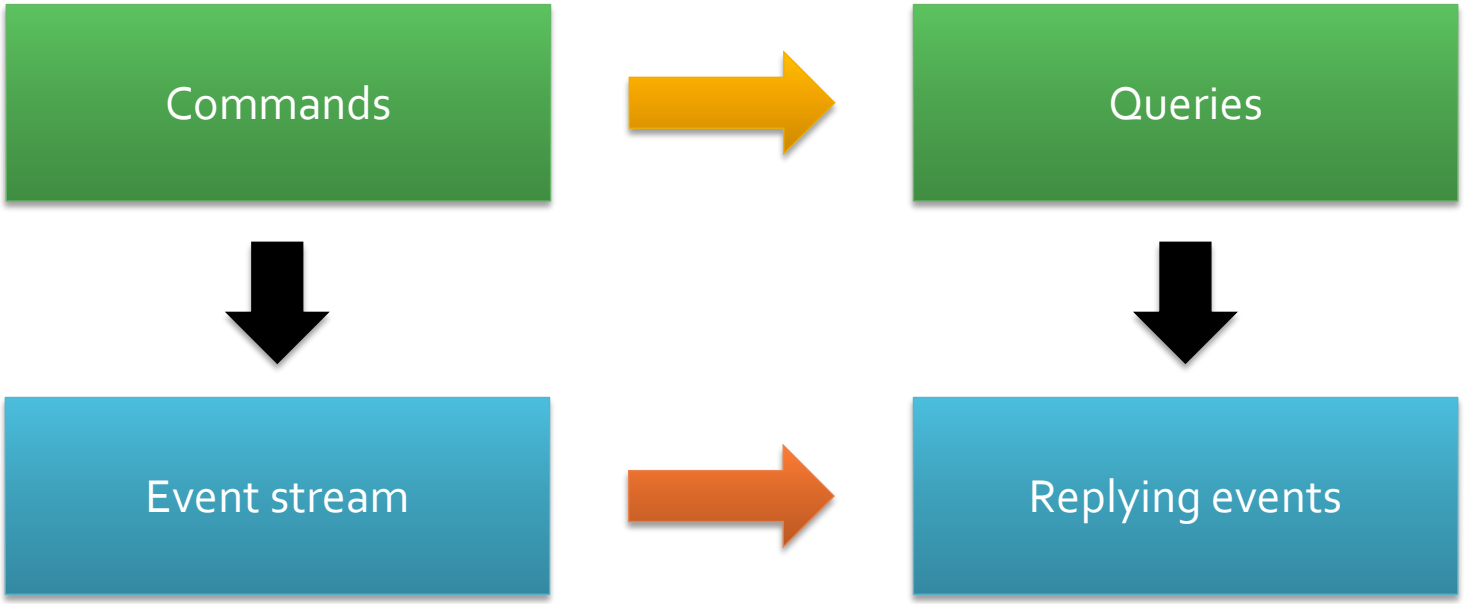
- To speed up, you can cache the results
  - A set of tables where the state is based on events
  - The structure reflects the current business need
    - If the need is changed, it can be reflected in the structure



https://dev.to/barryosull/event-sourcing-what-it-is-and-why-its-awesome

Connection to CQRS

- Now it is easy to the connection to CQRS when we have a model & database separation



Use cases

- Ensure the correct state of the object
  - E.g. caused by transaction isolation problem or problem with eventual consistency
  - For efficiency usually combined with snapshots
- Audit trails for analyzing current state
  - Debugging (although classic logging may work)
  - Audit for sensitive activities (e.g. account history)
- Multiple concurrent update problem
  - There is „append-only" and „read" operations
  - There is no „update" operation

References

- Article by Martin Fowler
  http://martinfowler.com/bliki/CQRS.html
- Great introduction by Udi Dahan
  http://www.udidahan.com/2009/12/09/clarified-cqrs/
- Good presentations
  http://www.slideshare.net/jeppec/cqrs-why-what-how
  http://www.slideshare.net/brianritchie1/cqrs-command-query-responsibility-segregation
- Nice introduction
  http://cqrs.nu/

References

- CQRS document by Greg Young
  http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
- Article by Microsoft
  http://msdn.microsoft.com/en-us/library/dn568103.aspx
- DDD/CQRS sample application
  http://cqrssample.codeplex.com/
- REST API and CQRS
  http://www.infoq.com/articles/rest-api-on-cqrs

References

- Event sourcing
  - https://dev.to/barryosull/event-sourcing-what-it-is-and-why-its-awesome
  - https://docs.microsoft.com/en-us/previous-versions/msp-n-p/jj591559(v=pandp.10)
  - https://martinfowler.com/eaaDev/EventSourcing.html
  - http://cqrs.wikidot.com/doc:event-sourcing
  - https://bulldogjob.pl/articles/122-cqrs-i-event-sourcing-czyli-latwa-droga-do-skalowalnosci-naszych-systemow_
  - https://medium.com/swlh/simple-event-souring-with-c-ec1eff55ee9d
  - http://www.andreavallotti.tech/en/2018/01/event-sourcing-and-cqrs-in-c/
  - https://dzone.com/articles/introduction-to-event-sourcing
  - https://www.eventstore.com/blog/what-is-event-sourcing