cqrs and events sourcing

who am I

gottfried szing

- doing IT stuff for 30+ years
- freelanceer
- architect/requirements engineer
- co-host of meetups
  - microservices, reactive and distributed systems
  - ddd vienna

what to expect…

- some patterns & concepts
- no code, json snippets for illustration
- no silver bullet for solving problems

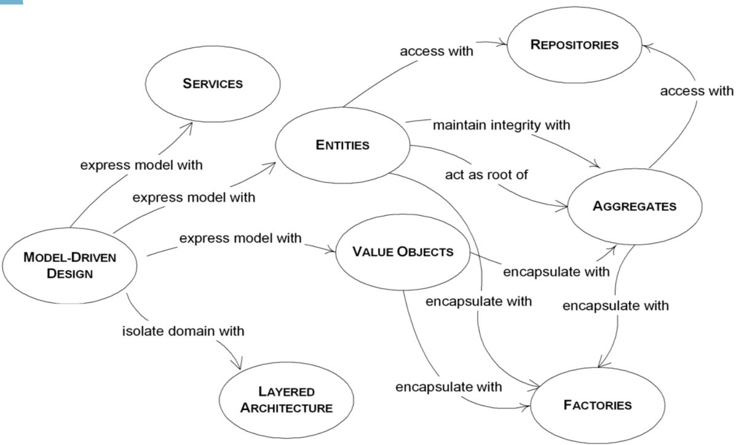
a possible path to a scalable, maintainable, fault tolerant, fault resilient application

Patterns & Concepts in DDD

Patterns & concepts in DDD

- ubiquitous language
- domain & subdomain
- bounded context
- context map

- entity
- value object
- aggregate
- service
- repository
- factory
- domain event



“

Captures the memory of something interesting which affects the domain.

”

Martin Fowler, 2005

“The essence of a Domain Event is that you use it to **capture** things that can **trigger a change** to the **state of the application** you are developing. ”

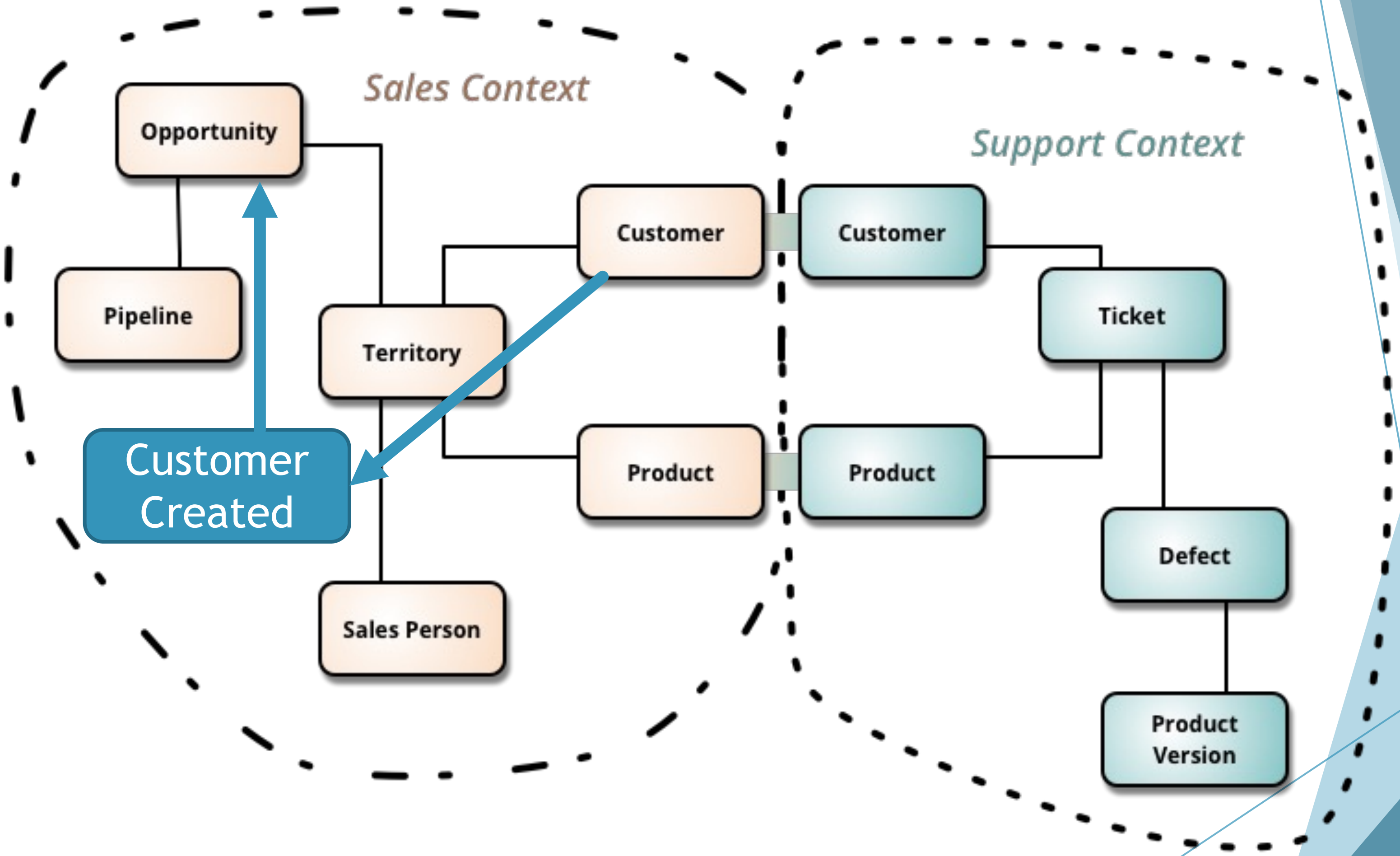
Martin Fowler, 2005

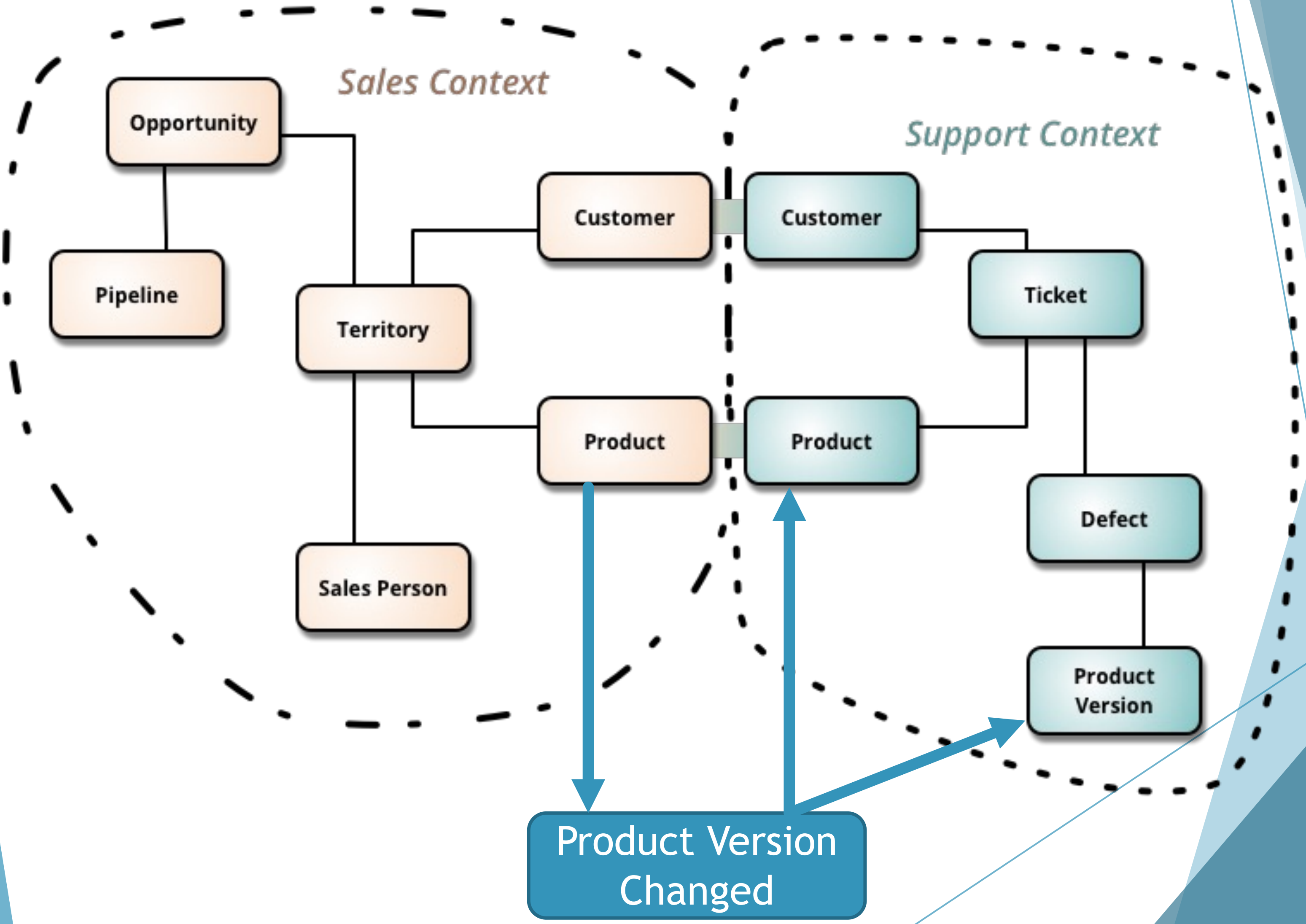
“

A Domain Event is a **record** of some **business-significant** occurrence in a **Bounded Context.**

”

Vernon, Vaughn. Domain-Driven Design Distilled, 2016

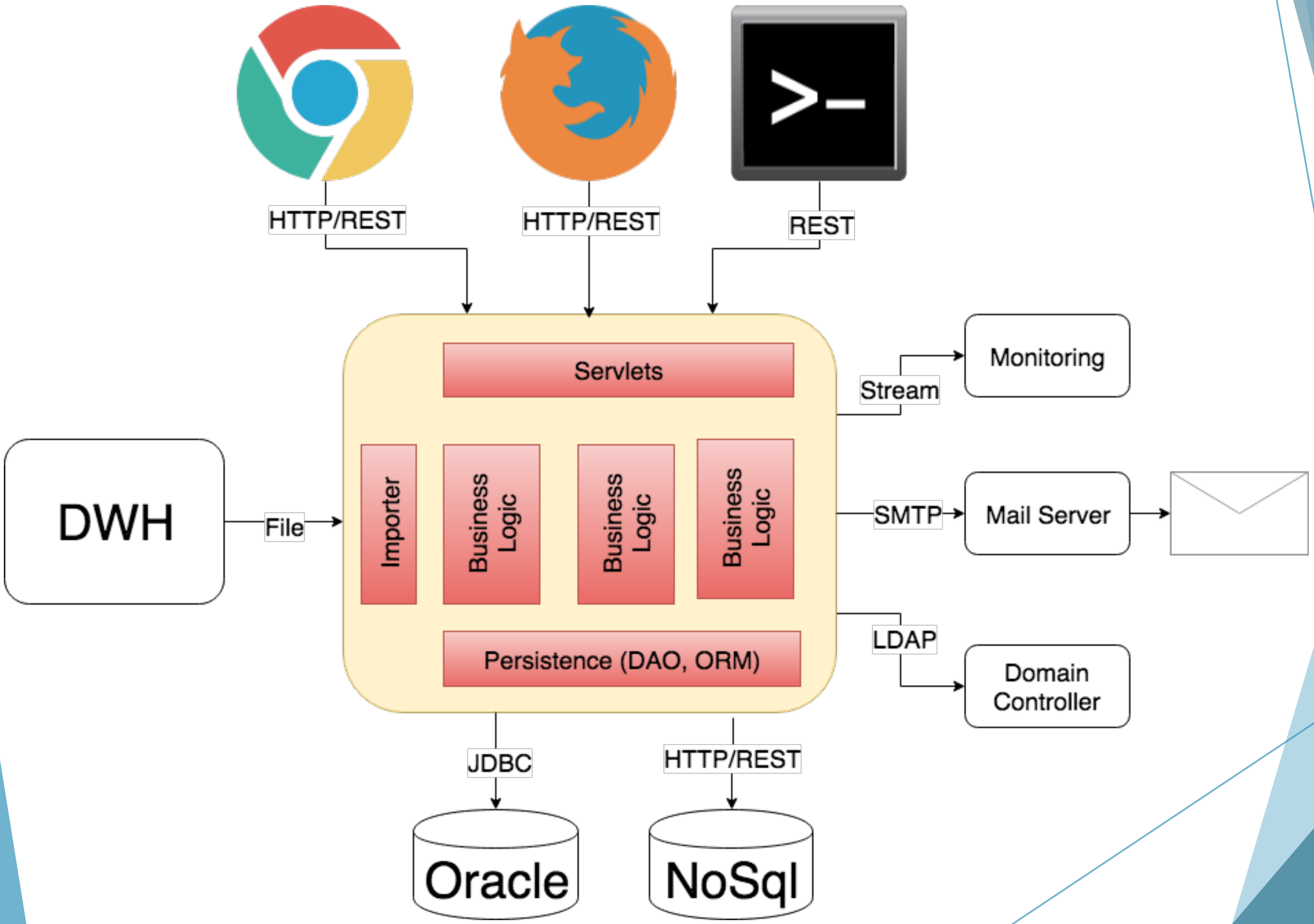

Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Product
Sales Person
Customer Created
Customer
Ticket
Product
Defect
Product Version


Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Customer
Ticket
Product
Product
Defect
Sales Person
Product Version
Product Version Changed

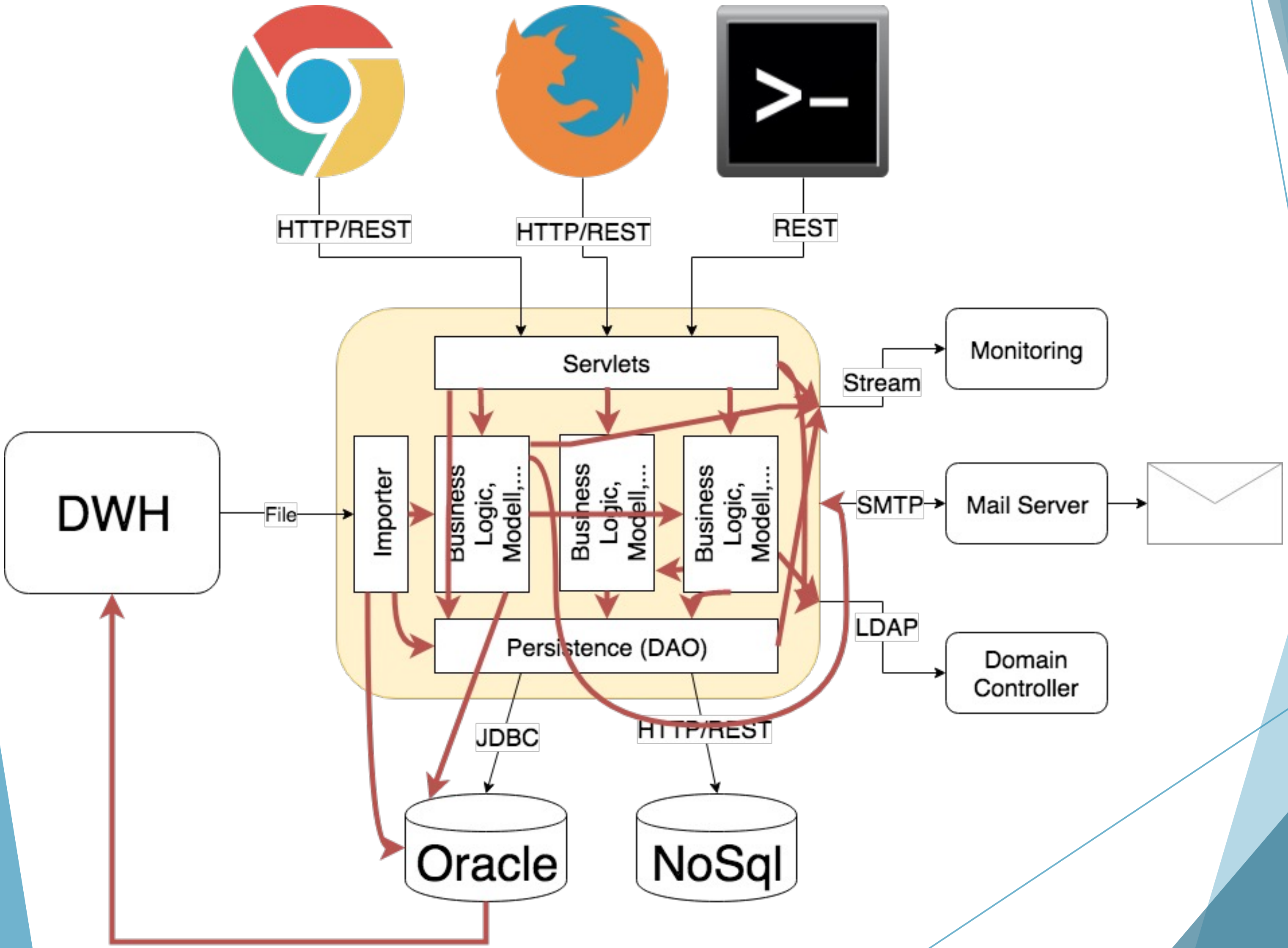
Characteristics of Domain Events

- result of a state change
- events happened in the **past**
- events are **facts**
- events can never be **rejected**
- events can never be changed, they are **immutable**
- part of the ubiquitous language
  - nameChanged
  - invoicePrinted
  - orderConfirmed
  - callRecorded
  - softwareCrashed

A typical application



Zoomed in...


HTTP/REST
HTTP/REST
REST
Servlets
Stream
Monitoring
DWH
File
Importer
Business Logic, Modell,...
Business Logic, Modell,...
Business Logic, Modell,...
SMTP
Mail Server
LDAP
Domain Controller
Persistence (DAO)
JDBC
HTTP/REST
Oracle
NoSql

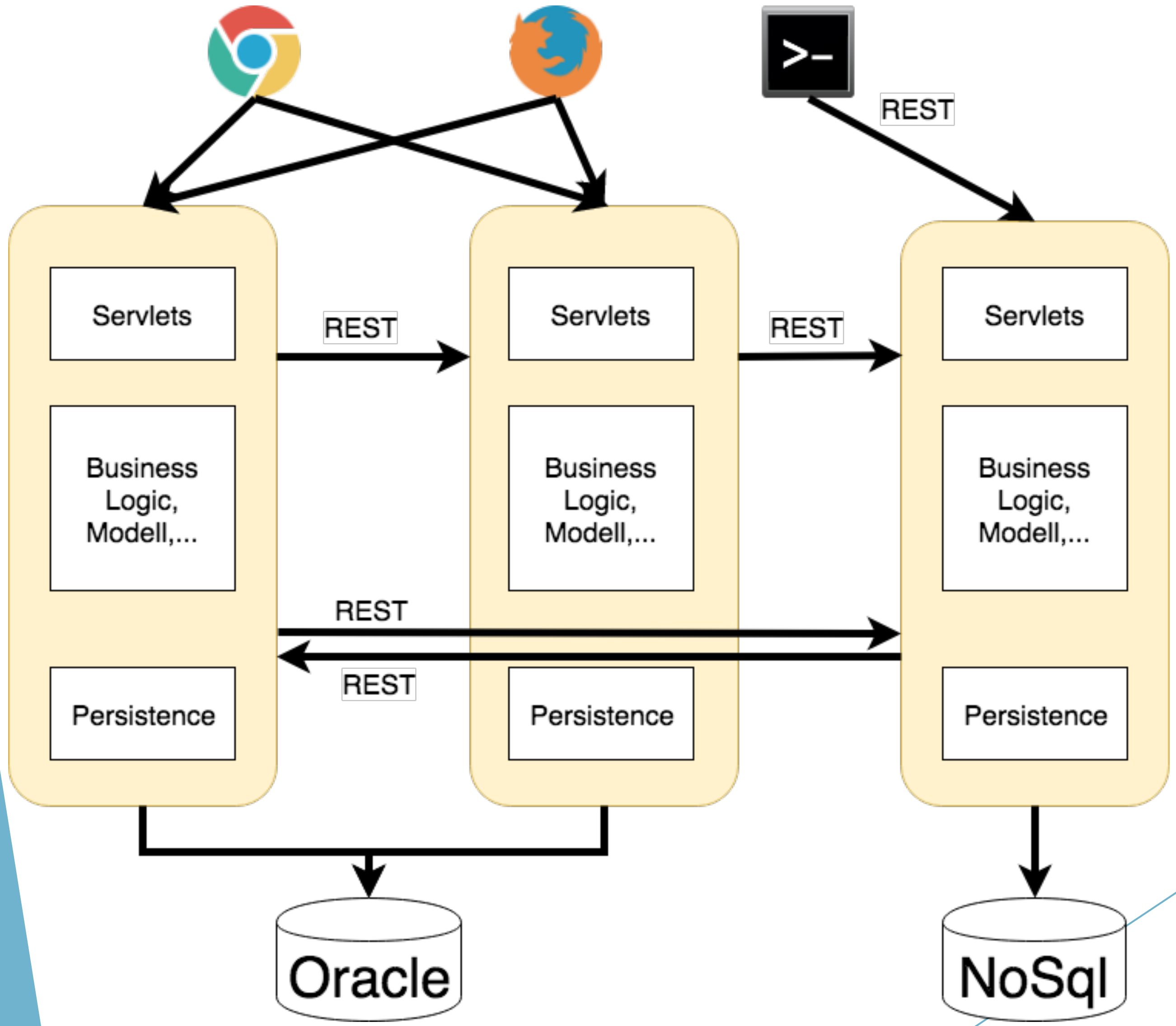
Evolving a monolith

Microservices to the rescue!?

What are microservices?

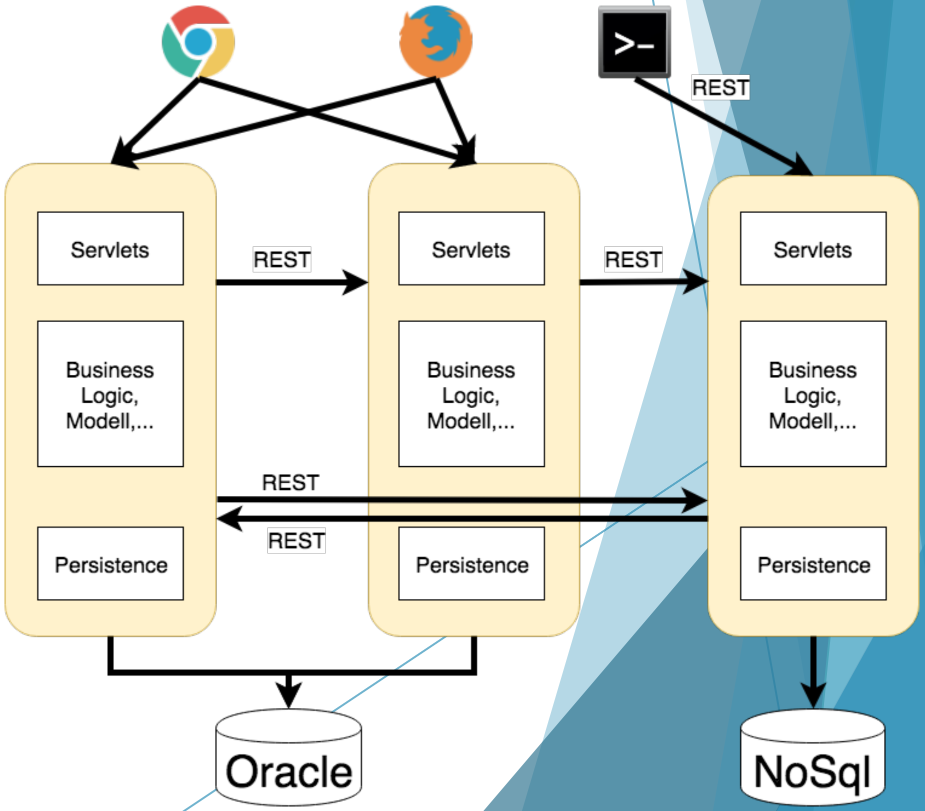
- Small,
- independent applications,
- loosely coupled,
- communicating with each other in an asynchronous manner (across system boundaries), and
- can fail independently, and if one or some fails, the system as a whole still works, though the functionality might be downgraded when there are failures.

First approach



First result

- A distributed monolith 😱
  - Tight Coupling
  - Synchronous Communication
  - Shared Datastores
  - Deployment Dependency
  - Lack of Autonomy
  - Complexity



Let’s try it again…

CQS: command-query separation

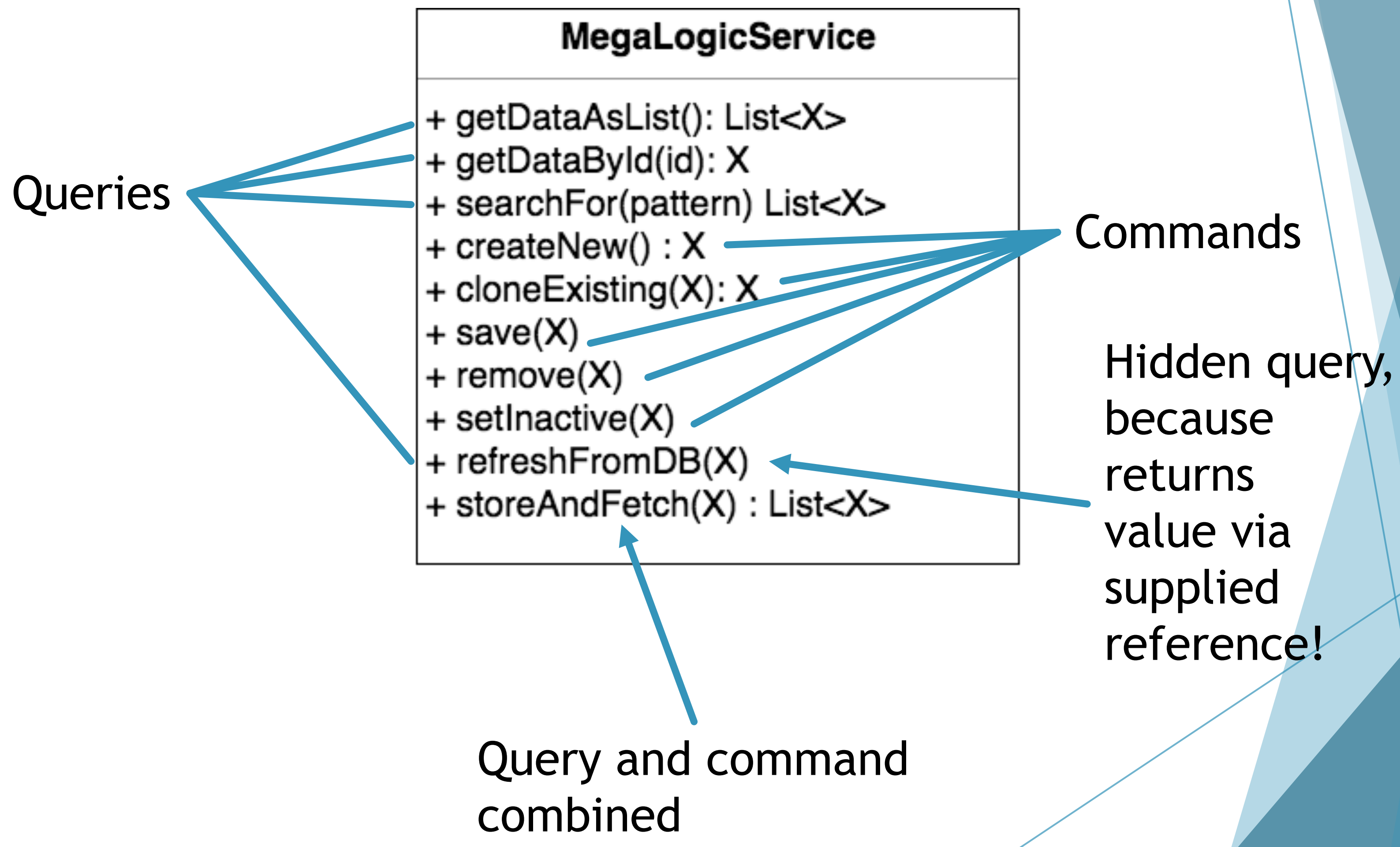
- pattern on component level

Methods either query or command but never both:

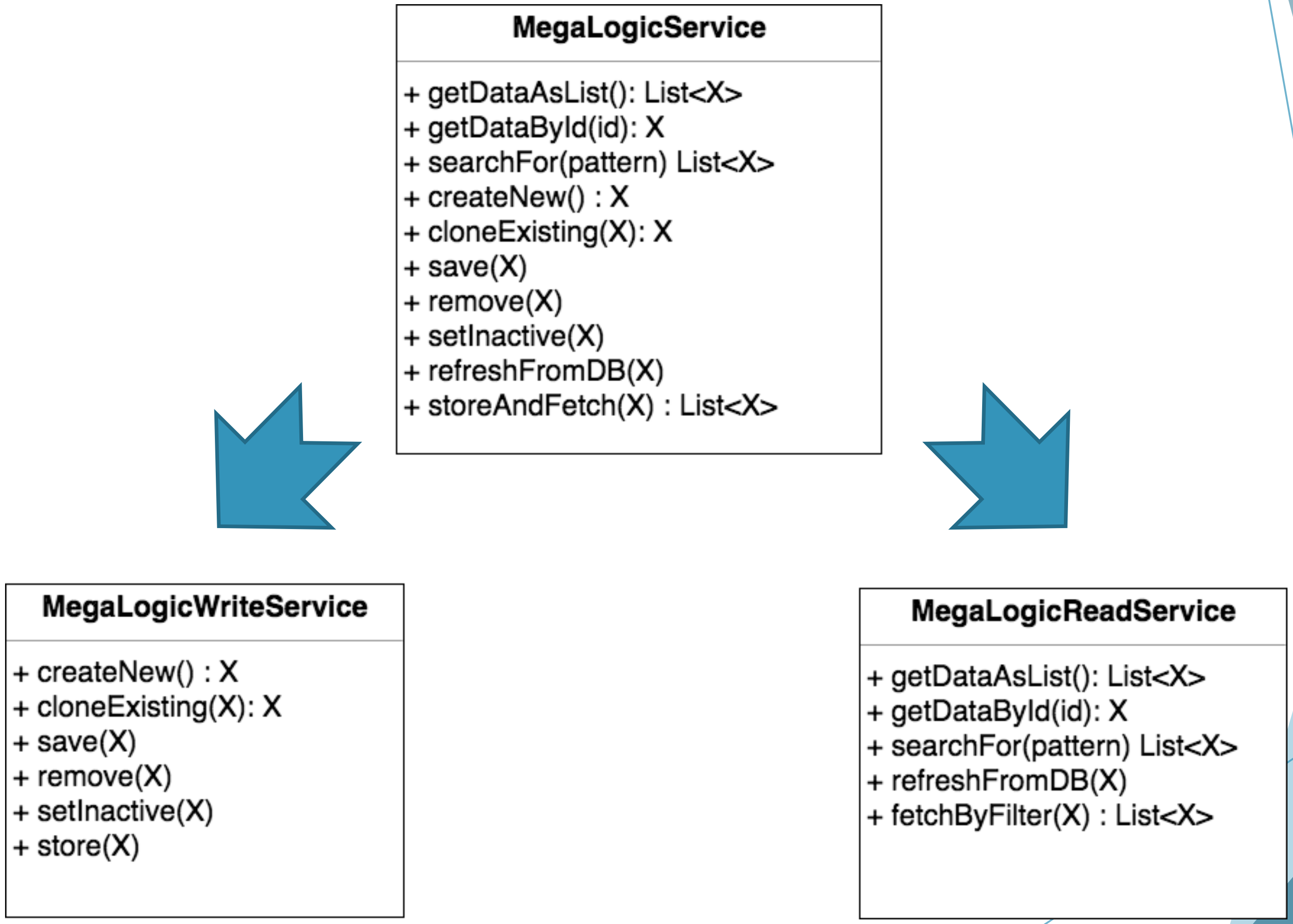
- **query** does not change state
- **command** does not return value, only changes the state

- read/write can have a separated model
- reads can be run multiple times without side effects
- reads are automatically idempotent

Example


MegaLogicService
+ getDataAsList(): List<X>
+ getDataById(id): X
+ searchFor(pattern) List<X>
+ createNew() : X
+ cloneExisting(X): X
+ save(X)
+ remove(X)
+ setInactive(X)
+ refreshFromDB(X)
+ storeAndFetch(X) : List<X>
Queries
Commands
Hidden query, because returns value via supplied reference!
Query and command combined

CQS applied

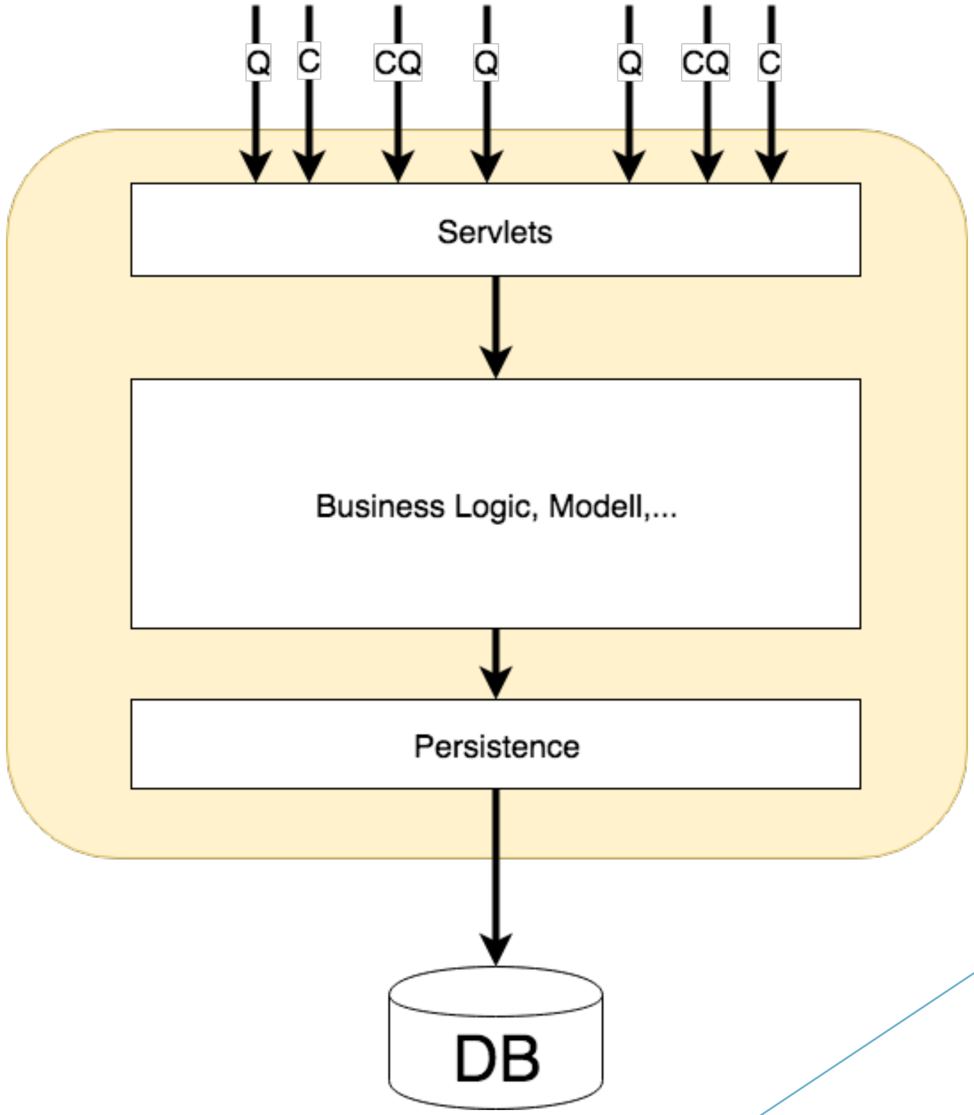


```
MegaLogicService
+ getDataAsList(): List<X>
+ getDataById(id): X
+ searchFor(pattern) List<X>
+ createNew() : X
+ cloneExisting(X): X
+ save(X)
+ remove(X)
+ setInactive(X)
+ refreshFromDB(X)
+ storeAndFetch(X) : List<X>
```

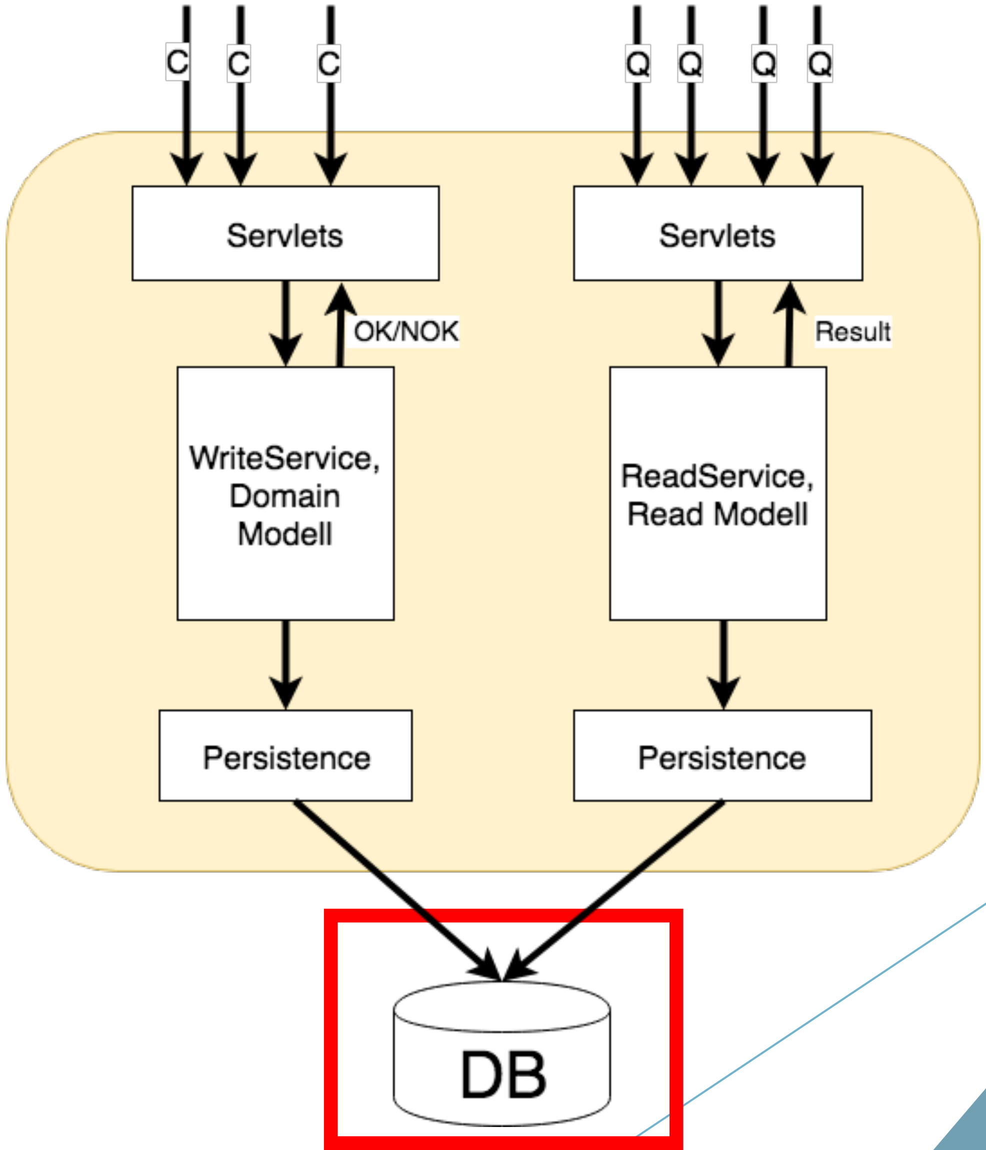
```
MegaLogicWriteService
+ createNew() : X
+ cloneExisting(X): X
+ save(X)
+ remove(X)
+ setInactive(X)
+ store(X)
```

```
MegaLogicReadService
+ getDataAsList(): List<X>
+ getDataById(id): X
+ searchFor(pattern) List<X>
+ refreshFromDB(X)
+ fetchByFilter(X) : List<X>
```

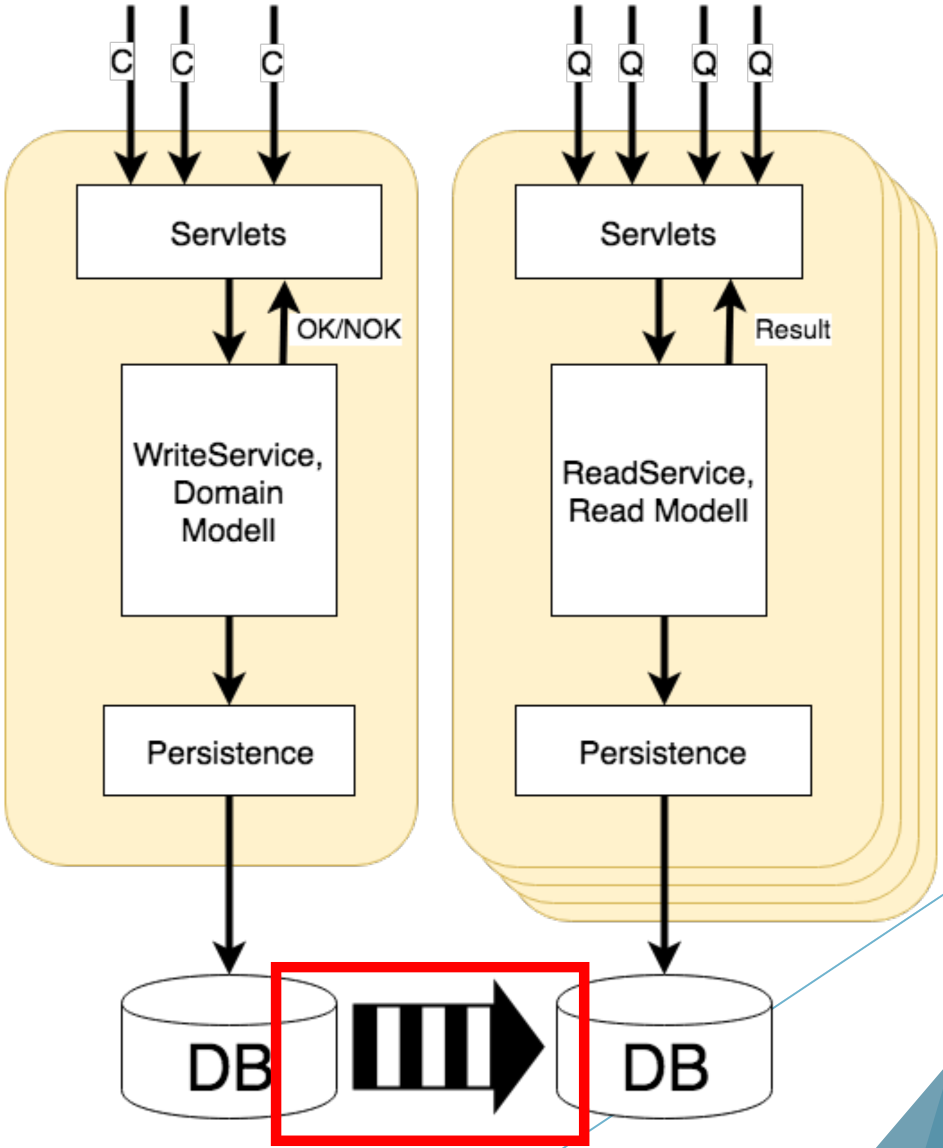
Monolith



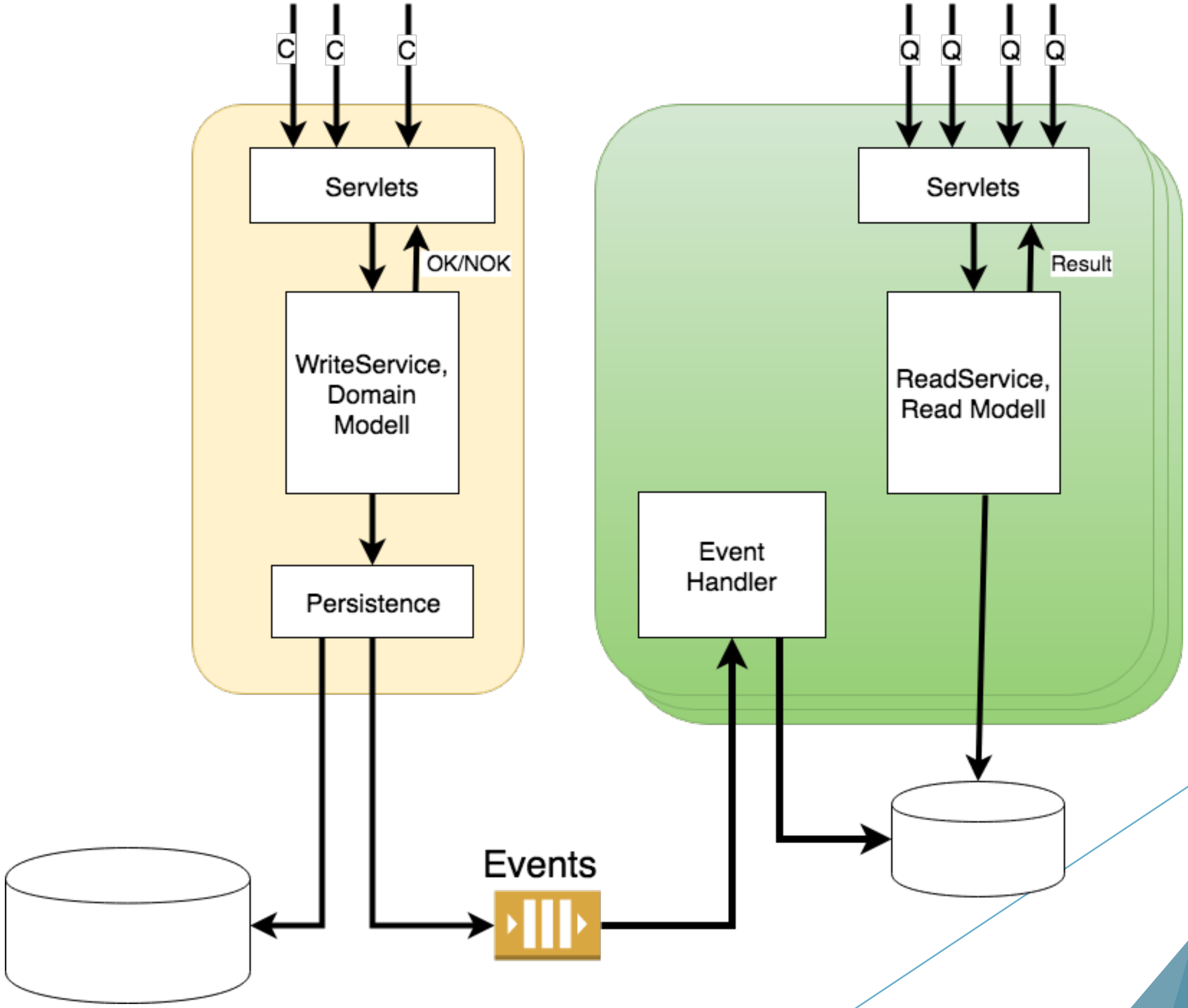
CQS for services



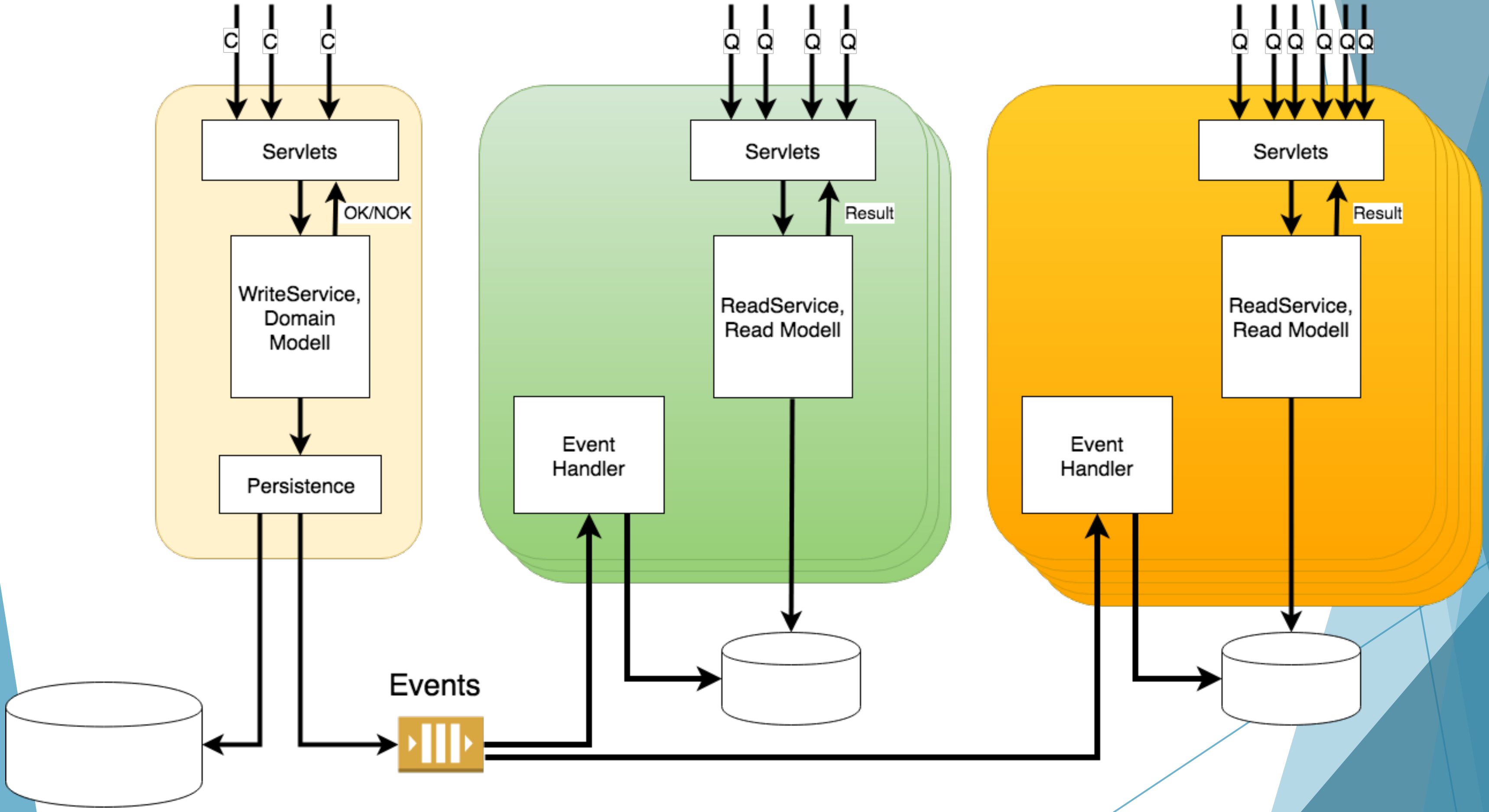
CQS with datapump



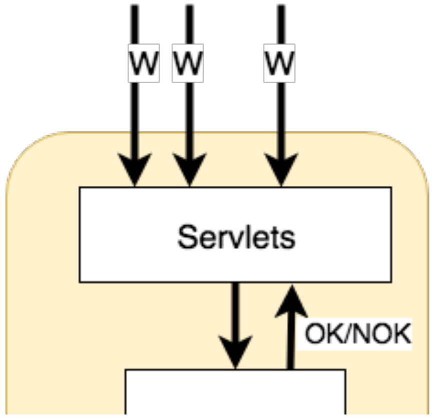
Event driven CQRS



CQRS for multiple read models



Concurrency of commands



commands should be handled concurrently…

… **but** this could lead to conflicts!

Conflicts with command

```
changeUsernameCommand {
    "userId": 123456
    "username": "toor"
}
```

```
changeUsernameCommand {
    "userId": 123456
    "username": ”admin"
}
```

```
person {
    "userId": 123456
    "username": “root”
}
```

```
usernameChangedEvent {
    "userId": 123456
    "username": ???
}
```

Conflict resolution/prevention

- optimistic locking
  - first one wins
  - last one wins
- pessimistic locking
- serialization of commands
  - single-writer principle
- versioning of the aggregate

Evolution of events

Reason for changes

- aggregate boundaries are wrong
- new attributes
- refactoring
- typos
- type changes
- …

Non-breaking changes

- adding attribute: up-casting of old events to new ones by using a new default by converting old events to new
  - leads to cascades of up-casts and costs performance ☹
- assumes a tolerant consumer

```
event {
    "id": 123
}
```

$\Rightarrow$

```
event {
    "id": 123
    "state": "cool"
}
```

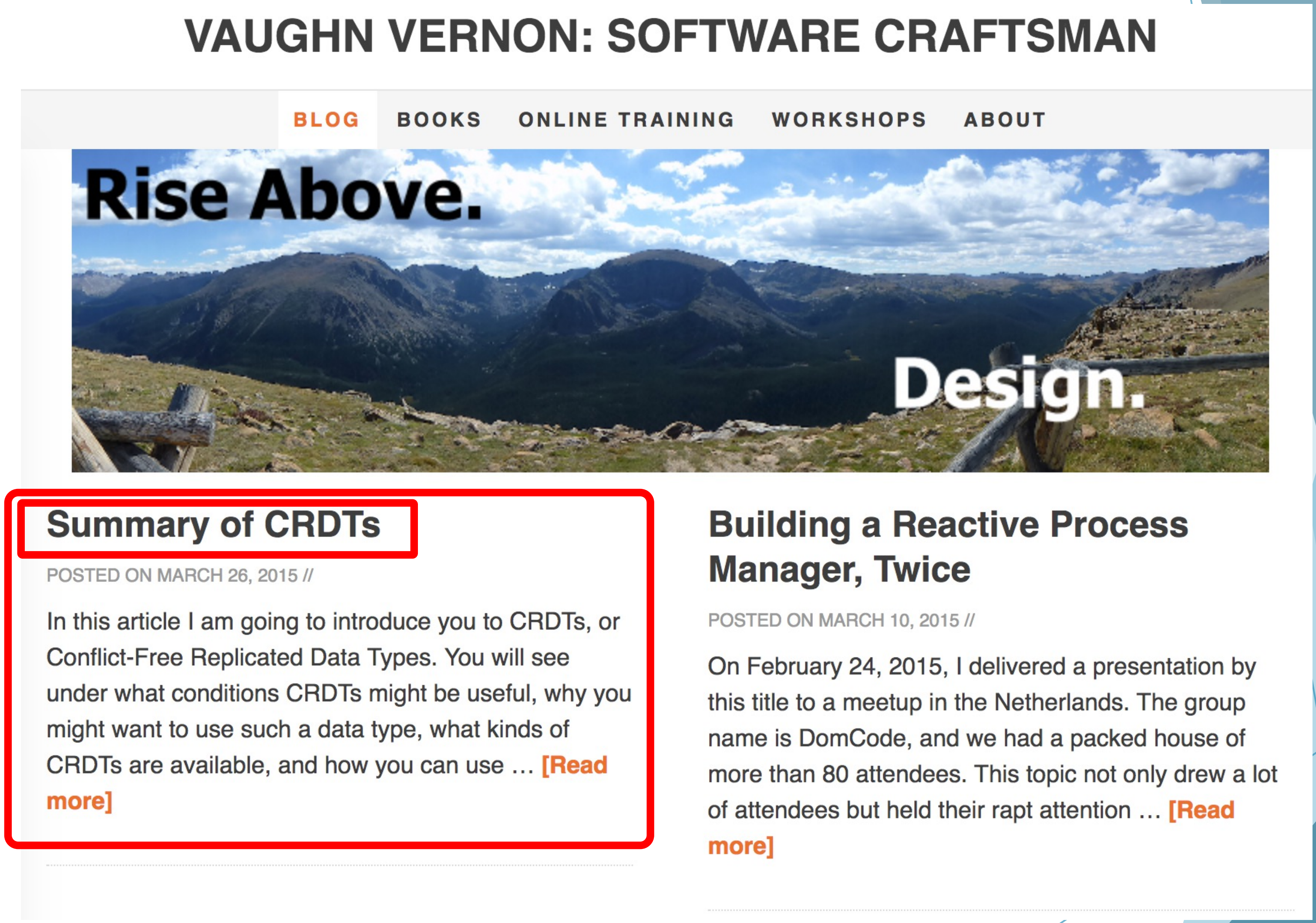
Breaking changes

- removing attribute
  - leave it in
- renaming of attribute
  - add new attribute or live with it
- changes of semantic meaning
  - e.g. discount of price in percentage or euro?
- type changes
  - e.g. flag to array
  - better to add an additional array to the existing flag

```
event {
     "id": 123
     ”discont": ”10”
     "flag": "boolean”
     ”newFlag": ["boolean”]
}
```

Event sourcing

What is event sourcing?



VAUGHN VERNON: SOFTWARE CRAFTSMAN

Summary of CRDTs

POSTED ON MARCH 26, 2015 //

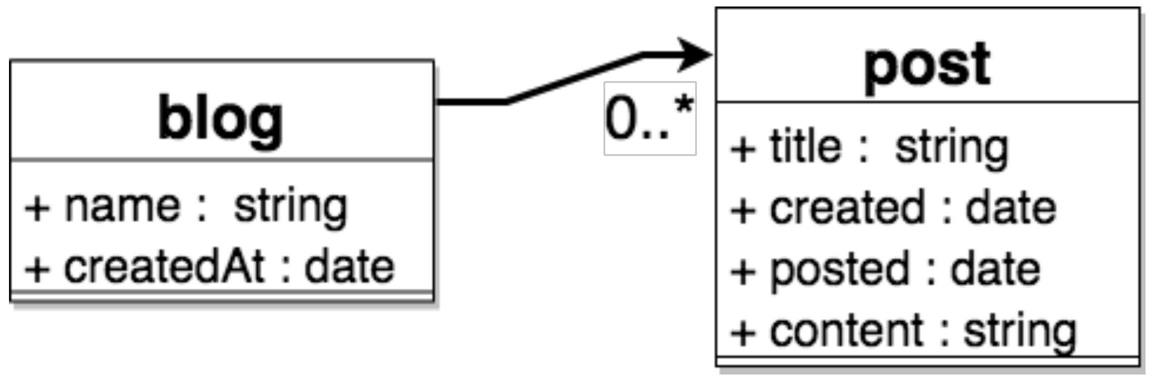
In this article I am going to introduce you to CRDTs, or Conflict-Free Replicated Data Types. You will see under what conditions CRDTs might be useful, why you might want to use such a data type, what kinds of CRDTs are available, and how you can use … [Read more]

Building a Reactive Process Manager, Twice

POSTED ON MARCH 10, 2015 //

On February 24, 2015, I delivered a presentation by this title to a meetup in the Netherlands. The group name is DomCode, and we had a packed house of more than 80 attendees. This topic not only drew a lot of attendees but held their rapt attention … [Read more]

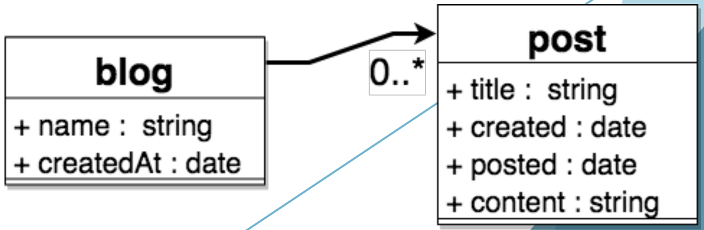
Domain modell



Persistence

id	name	created_at
33	Vernons Blog	NOVEMBER 1, 2008

id	blog	title	created_at	posted	content
12	33	Summary of CRDTs	MARCH 01, 2015	MARCH 26, 2015	In this article I am going to introduce you to CRDTs....



Events for persistence

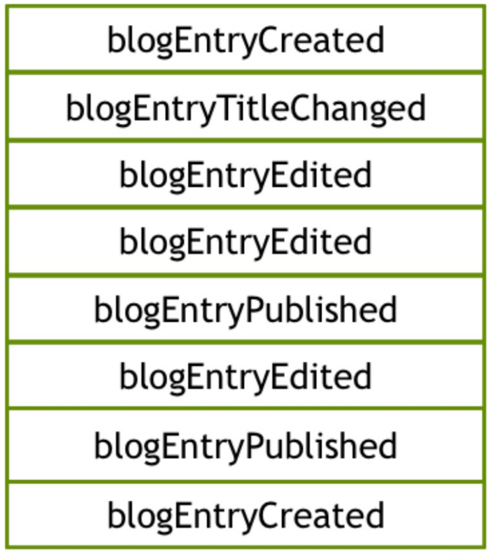
Time

blogEntryCreated	12;33;MARCH 01, 2015
blogEntryTitleChanged	12;“Summary of CRDTs”
blogEntryEdited	12;“...CDRTs...”
blogEntryEdited	12;“...CRDTs...”
blogEntryPublished	12;Vernon, MARCH 25, 2015
blogEntryEdited	12;“Yeah, But…”
blogEntryPublished	12;Vernon, MARCH 26, 2015
blogEntryCreated	13;33;MAY 14, 2015

id	blog	title	created_at	posted	content
12	33	Summary of CRDTs	MARCH 01, 2015	MARCH 26, 2015	...CRDTs...
13	33		MAY 14, 2015		

Event sourcing

- events are the source of truth
  - audit logging
  - traceability
- easy temporal queries
- ability to put the system in prior state
  - for debugging
- recreating state with new business logic
  - replay of events
  - bug fixing
- evolutionary architectures



Event stores

- strict order
- append the event
- full sequential read
- read all events for a given entity
- atomicity and durability
- scalable

EVENT STORE

kafka

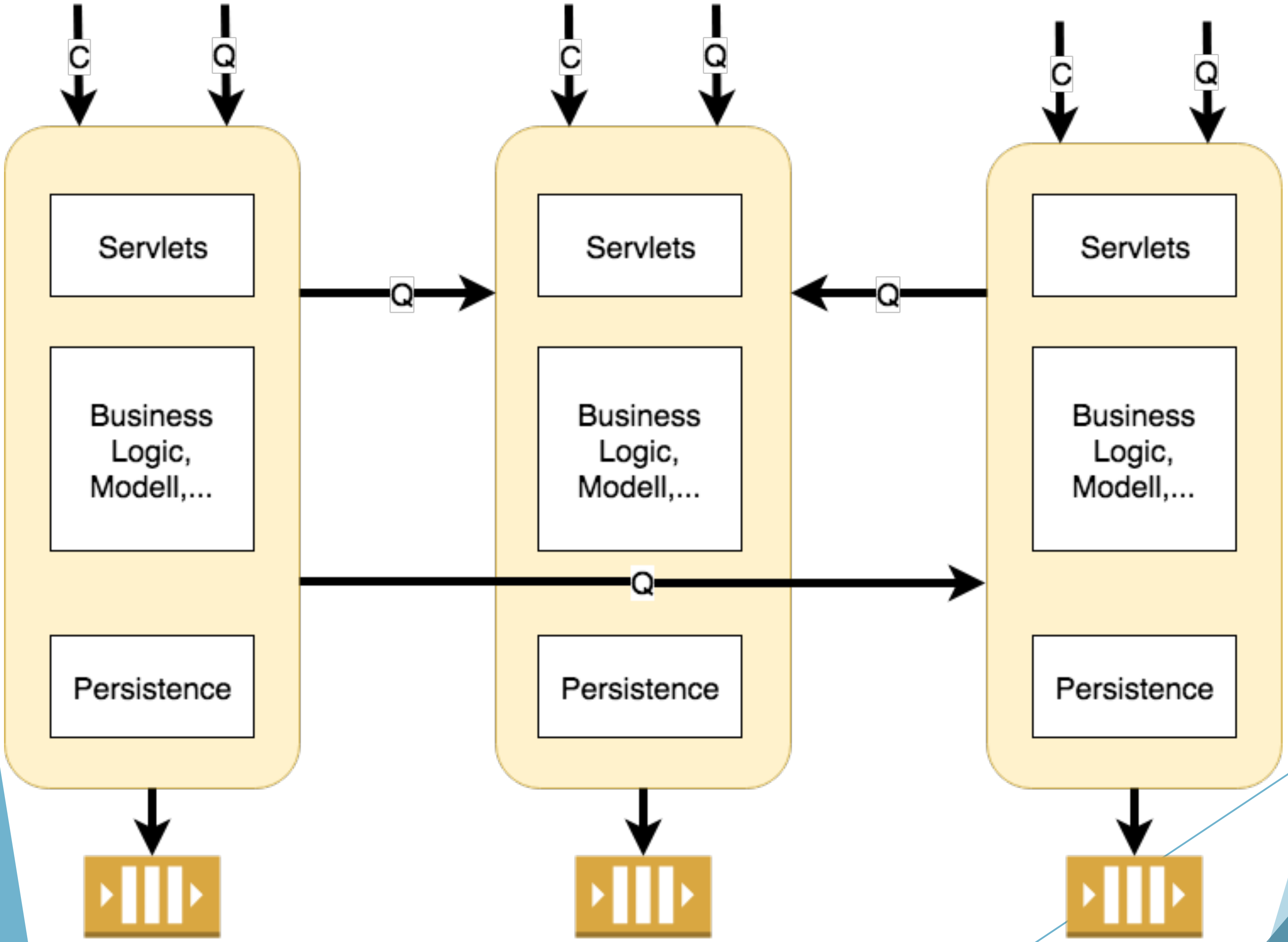
AxonIQ

PostgreSQL

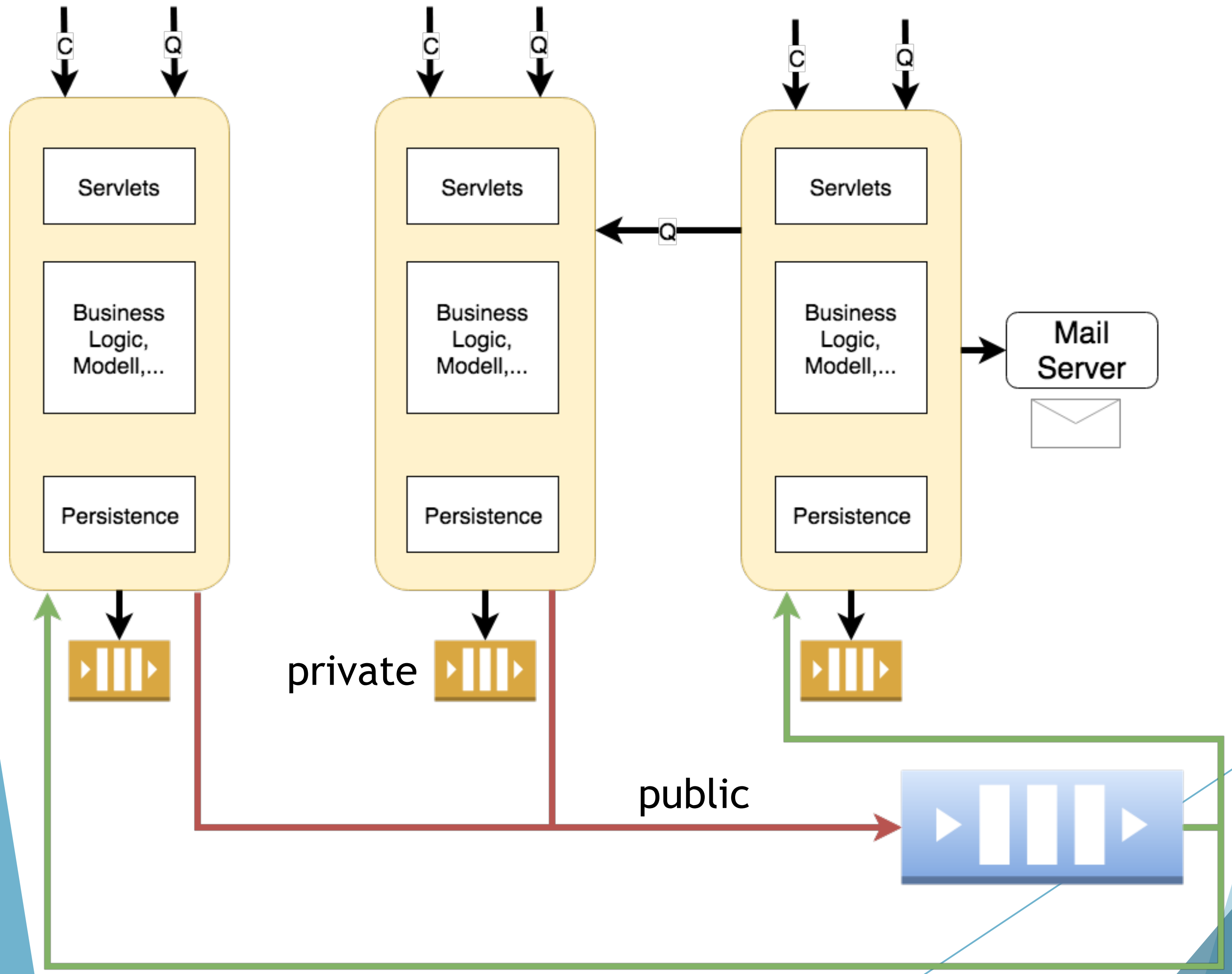
blogEntryCreated
blogEntryTitleChanged
blogEntryEdited
blogEntryEdited
blogEntryPublished
blogEntryEdited
blogEntryPublished
blogEntryCreated

Events for integration

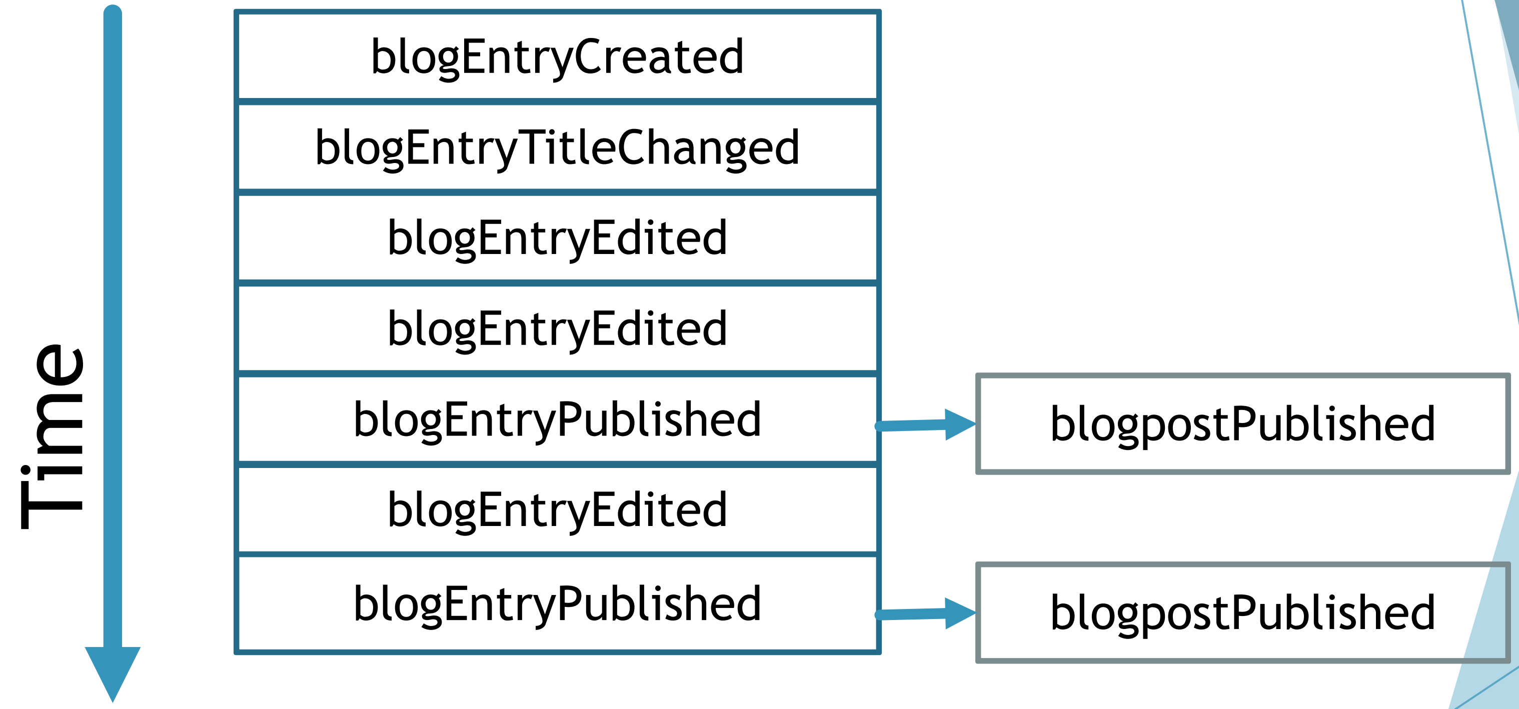
Integration of microservices


C
Q
C
Q
C
Q
Servlets
Business Logic, Modell,...
Persistence
Q
Servlets
Business Logic, Modell,...
Persistence
Q
Servlets
Business Logic, Modell,...
Persistence
Q

Integration Events



Events sourcing & integration



internal / private

external / public

Further topics and pitfalls

- versioning of events
- performance in event sourcing
- long running processes: orchestration vs. choreography
- sagas for long lived/distributed transactions
- how to identify domain events
- data protection, authorization, GDPR,...
- transactional boundaries between DB & event store
- conway’s law

Questions…

gottfriedszing

gottfried@szing.eu