pragmatic CQRS

Артём Акуляков

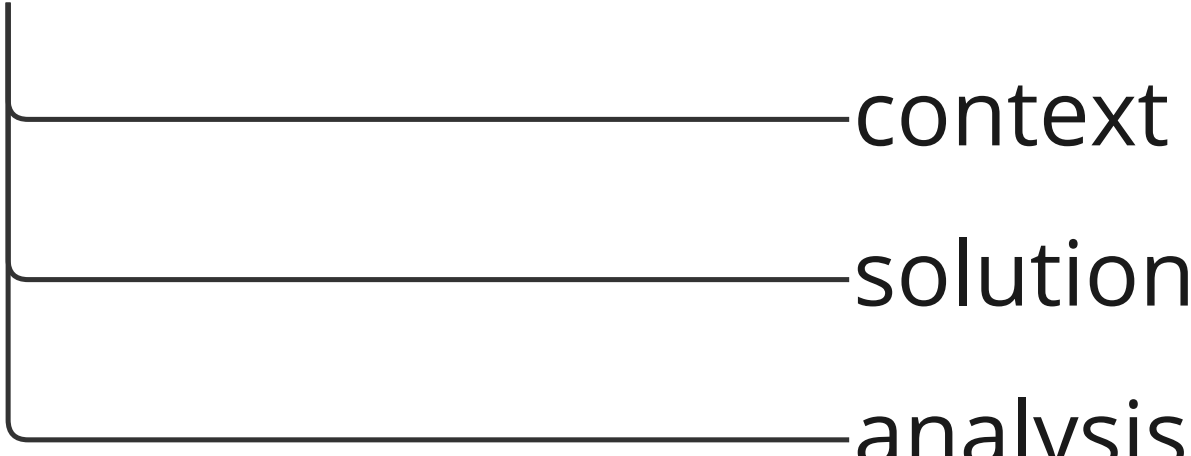
- architect в CallCTO
- 2й сезон член пк .NEXT
- .net с 2й версии (~15 лет)
- f(ƛ) & f#, arch (DDD, **CQRS**, ES, ...)
- > 10 проектов с **CQRS** irl

pragmatic CQRS

CQRS ≠ best practice

CQRS — это pattern
(архитектурный стиль)

pattern

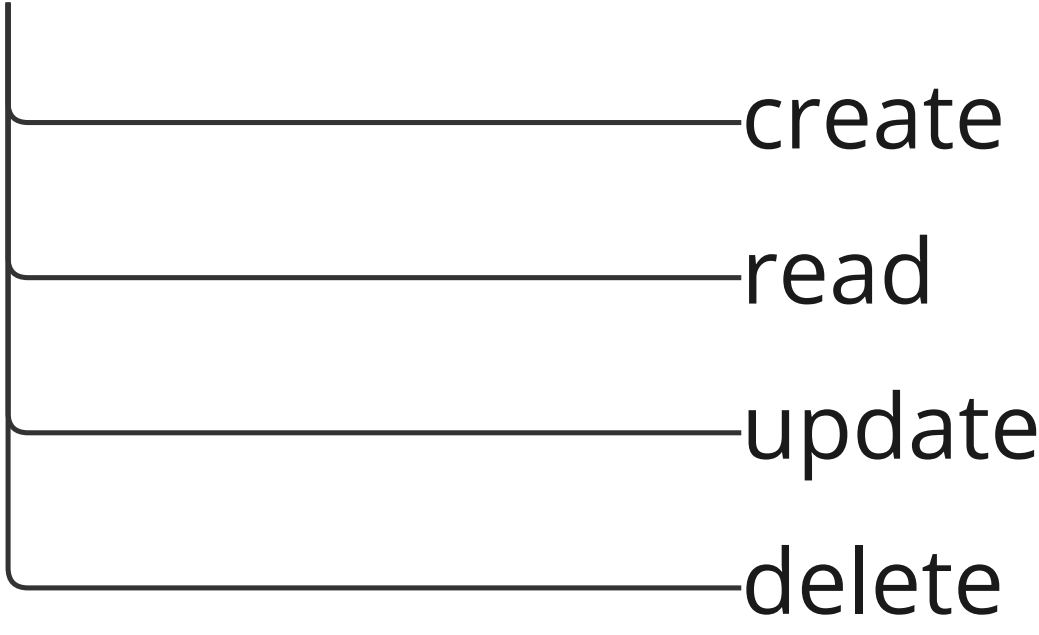


context

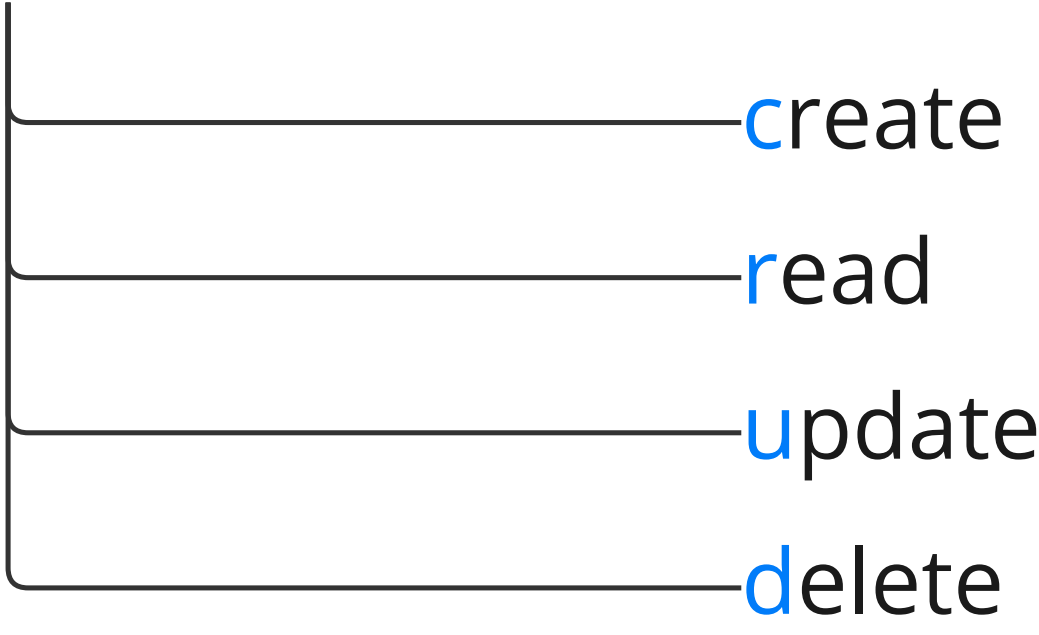
software =

data + operations

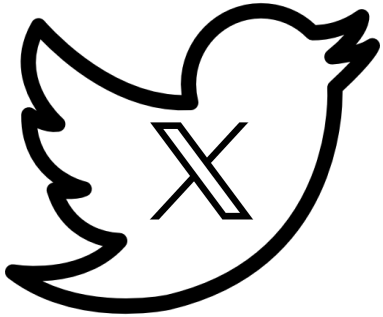
operations



operations



CRUD → c**R**ud



~49% акаунтов публикуют < 5 твитов в месяц

~30.4 минуты в день на пользователя

eCom

READ

полистали каталог

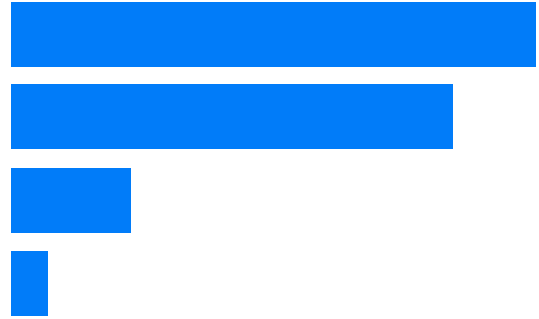
посмотрели товар

READ

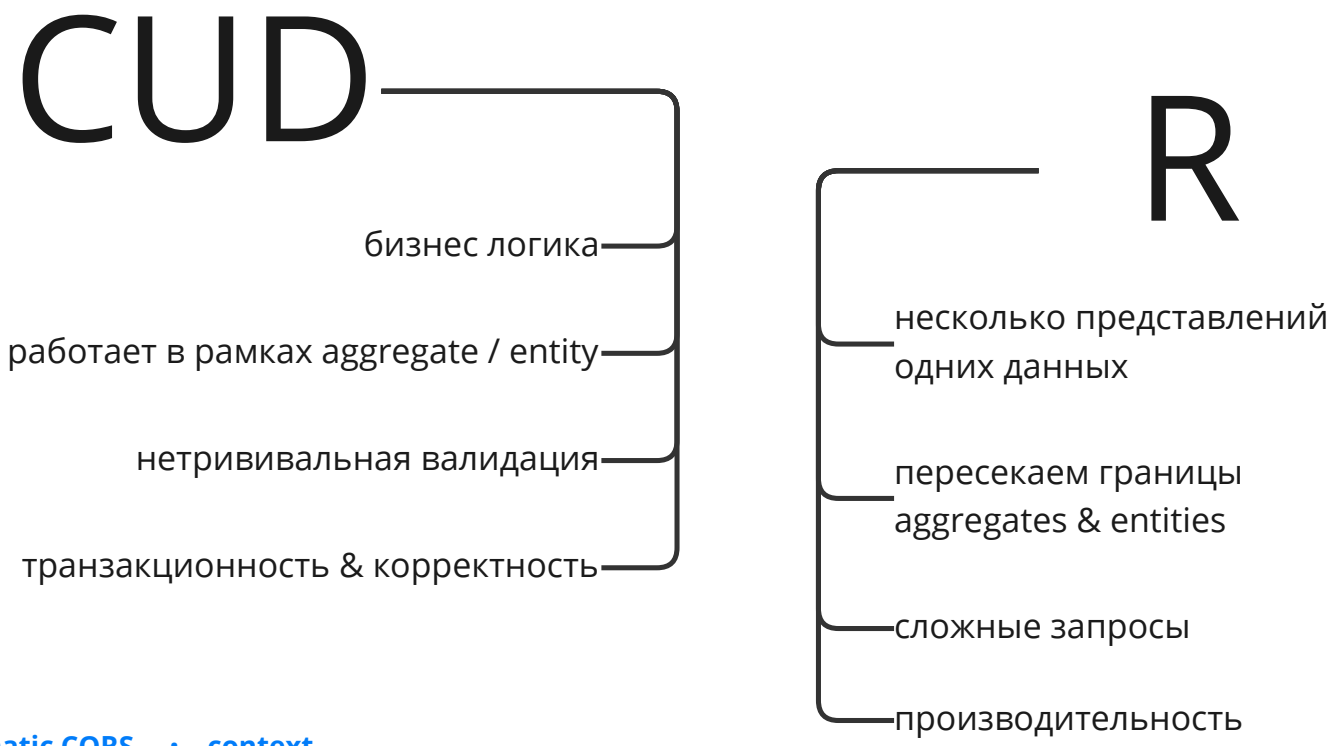
добавили в корзину

WRITE

наконец купили



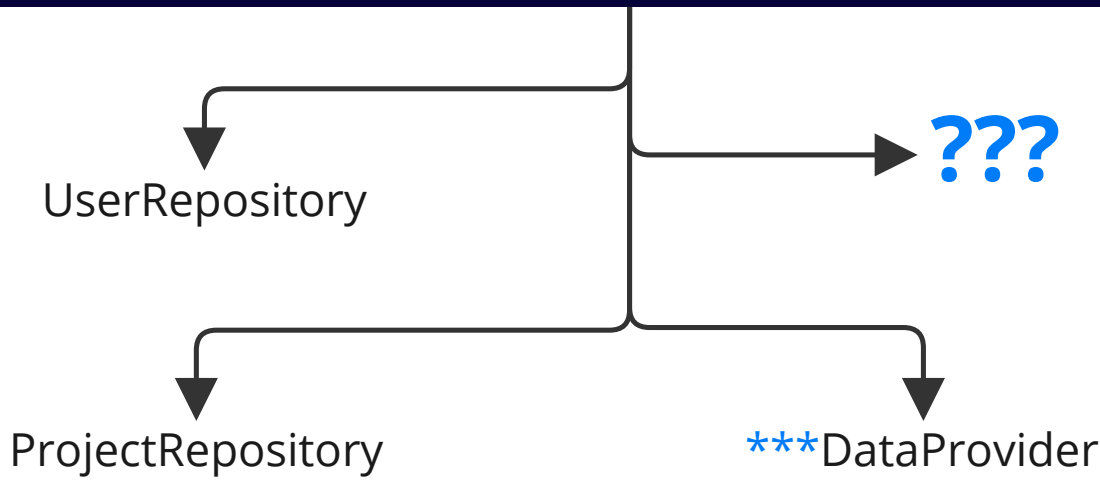
CRUD → c**R**ud


CUD
бизнес логика
работает в рамках aggregate / entity
нетрививальная валидация
транзакционность & корректность
R
несколько представлений одних данных
пересекаем границы aggregates & entities
сложные запросы
производительность

CUD *vs* R

```
public class UserRepository: IUserRepository {
    ...

    public void Create(User user) {...}
    public void Update(User user) {...}
    public void Delete(User user) {...}

    public User GetById(Guid id) {...}
    public User GetRecentActive() {...}
    public User? TryFind(string searchString) {...}
    public User[] GetAllOnFreePlan() {...}
    public User[] GetAllWithoutBillingInformation() {...}
    ...
    // 100500 methods later
    public User[] GetWhoCommentedArticleInProject(Guid projectId) {...}
}
```

```
public User[] GetWhoCommentedArticleInProject(Guid projectId)
{
    ...
}
```

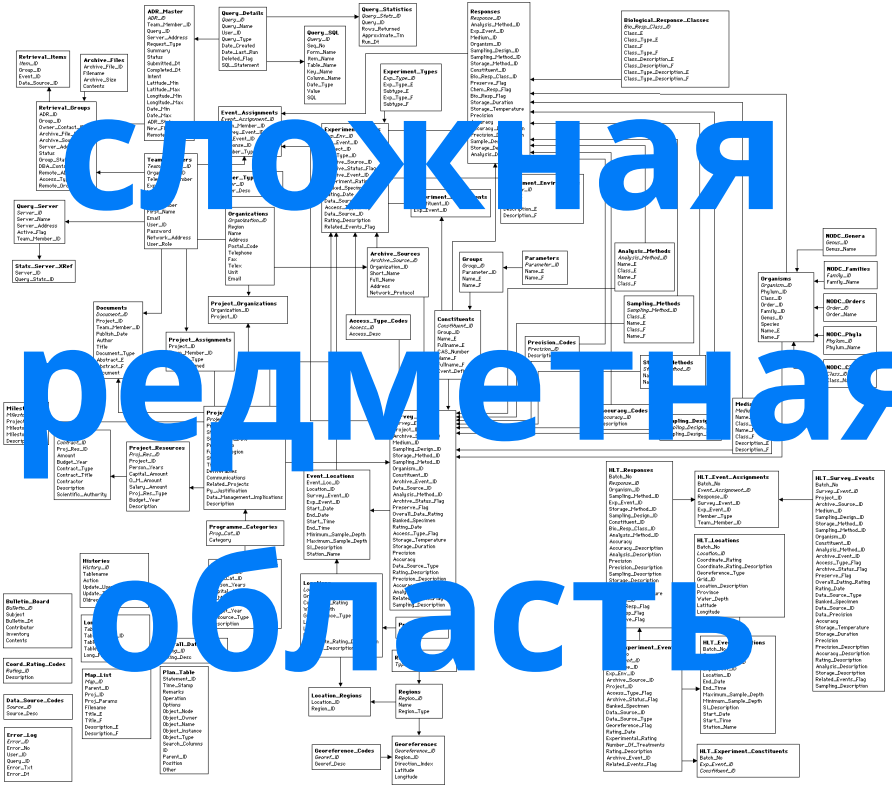


mapping

```
public class UserToShortViewMapper: IMapper<User, UserShortView>
{
    public UserShortView Map(User obj) { ... }
}
```

UserShortView, UserView, UserDetailView, ..., ***View

complexity



сложная предметная область

```
SELECT t.line_id AS LineId, ...
       sn.item_transaction_id AS SerialItemTransactionId, sn.serial_number AS SerialNumber,
       sl.id AS SaleShipmentNoteLineId, sl.item_id SaleShipmentNoteItemId,
       sh.id AS SaleShipmentNoteHeaderId, ...
       ol.id AS SaleOrderLineId, ...
       sc.lines AS ConfigurationsLines
FROM unnest(@line_id, @link_id, @item_transaction_id, @link_serial_number,
            @link_item_id, @serial_number_item_transaction_id)
AS t(line_id, link_id, item_transaction_id, link_serial_number,
     link_item_id, serial_number_item_transaction_id)
LEFT JOIN read.serial_numbers AS sn
    ON (t.serial_number_item_transaction_id IS NOT NULL AND ... )
    OR (serial_number_item_transaction_id IS NULL AND ...)
LEFT JOIN read.shipment_note_lines AS sl
    ON (t.serial_number_item_transaction_id IS NOT NULL AND ...)
    OR (t.serial_number_item_transaction_id IS NULL
        AND sn.item_transaction_id = sl.item_transaction_id
        AND t.link_item_id = sl.item_id)
LEFT JOIN read.shipment_note_headers AS sh ON sl.header_id = sh.id
LEFT JOIN read.order_lines AS ol ON sl.item_transaction_id = ol.item_transaction_id
LEFT JOIN read.serial_number_configurations AS sc ON sl.item_transaction_id = sc.item_transaction_id
LEFT JOIN ...
```

```sql
CREATE TABLE orders (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   ...
);

CREATE TABLE order_lines (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   order_id UUID NOT NULL,
   ...
   CONSTRAINT fk_order
      FOREIGN KEY(order_id)
        REFERENCES orders(id)
);
```

денормализация

cache hell

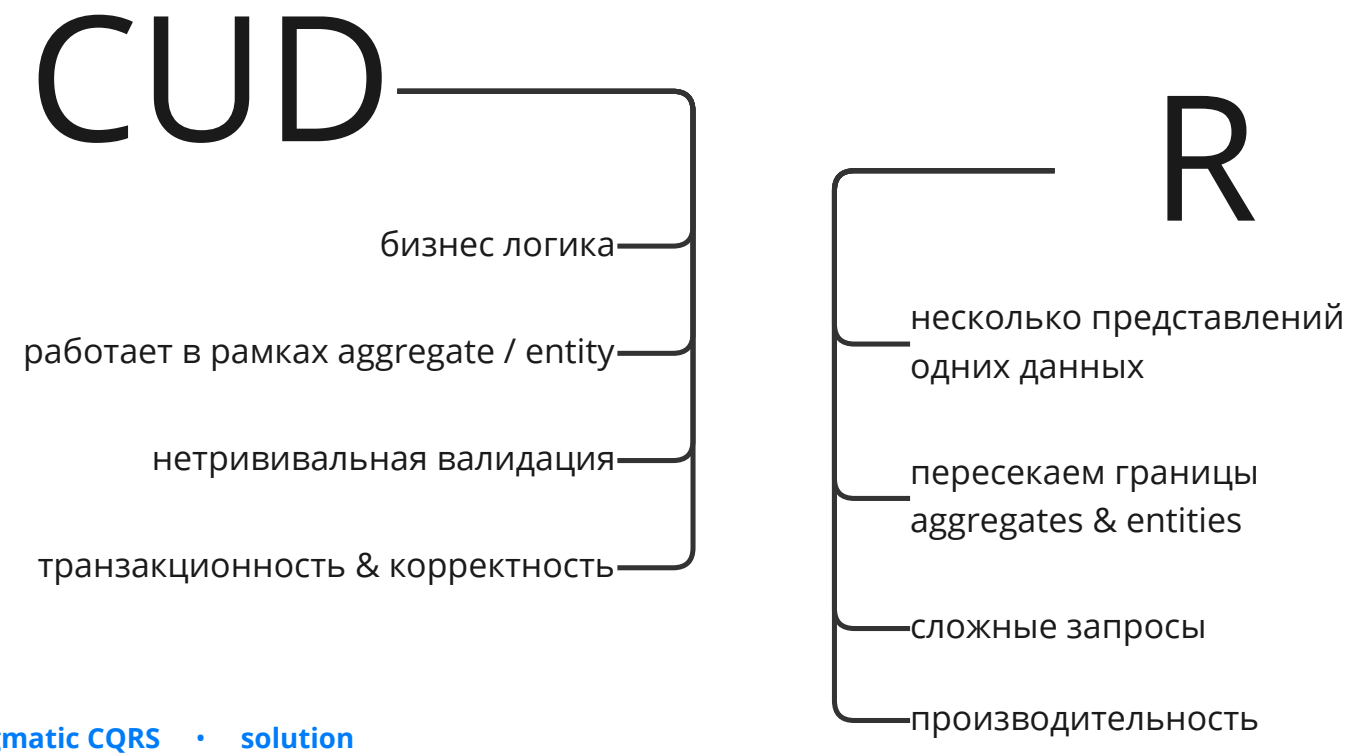
```sql
CREATE TABLE orders (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   lines JSONB,
   ...
);
```

complexity

performance

solution

CUD *vs* R

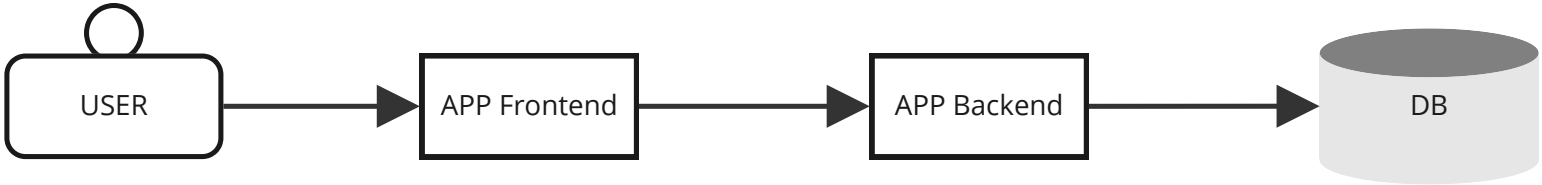

CUD
бизнес логика
работает в рамках aggregate / entity
нетрививальная валидация
транзакционность & корректность
R
несколько представлений одних данных
пересекаем границы aggregates & entities
сложные запросы
производительность

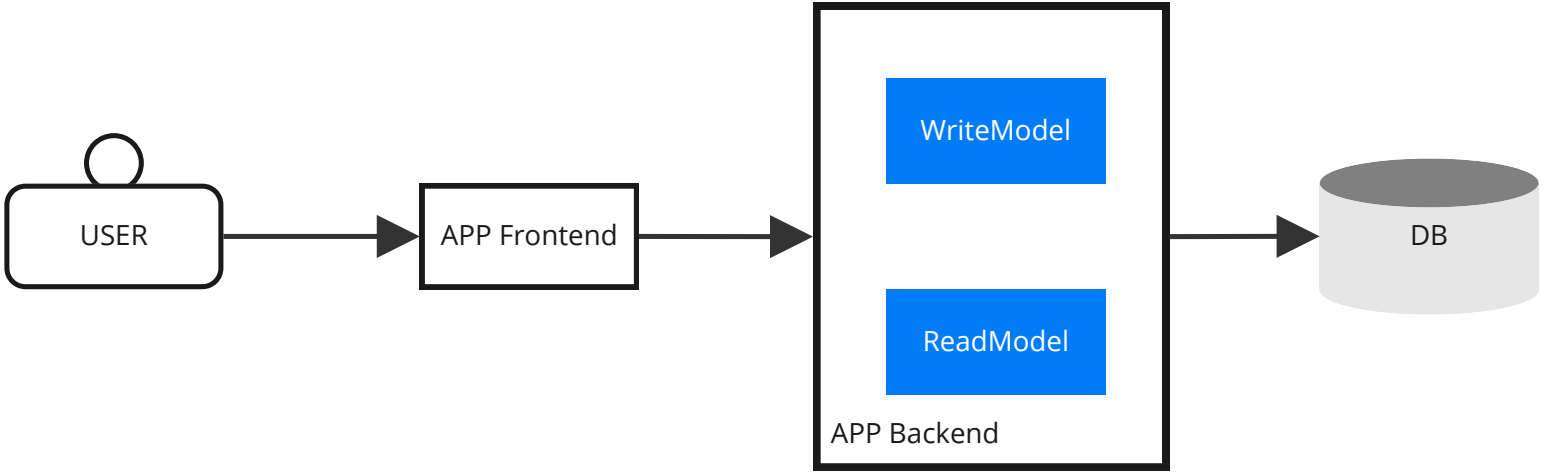
CUD *vs* R

CUD **&** R

RW Model →

R Model ∪ W Model


USER
APP Frontend
APP Backend
DB


USER
APP Frontend
WriteModel
ReadModel
APP Backend
DB

CQRS ~~v1.0~~ #1

Write Model

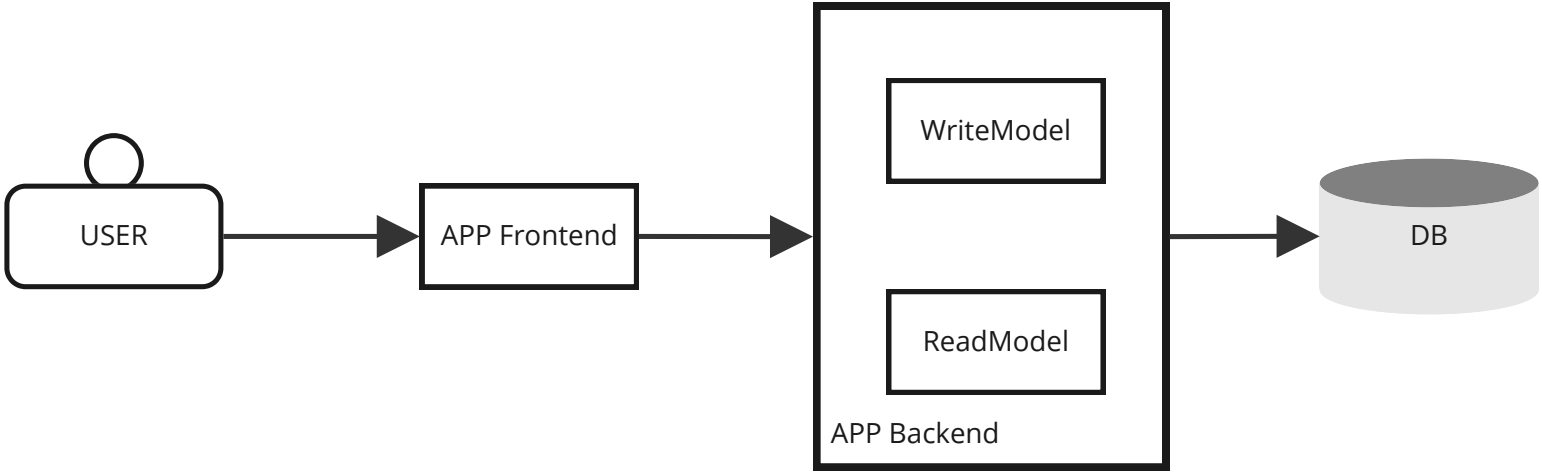
- FluentValidator+MediatR+EF+AutoMapper
  (не является рекомендацией)
- clean architecture, DDD, rich domain, ...
- DAL: repository & co

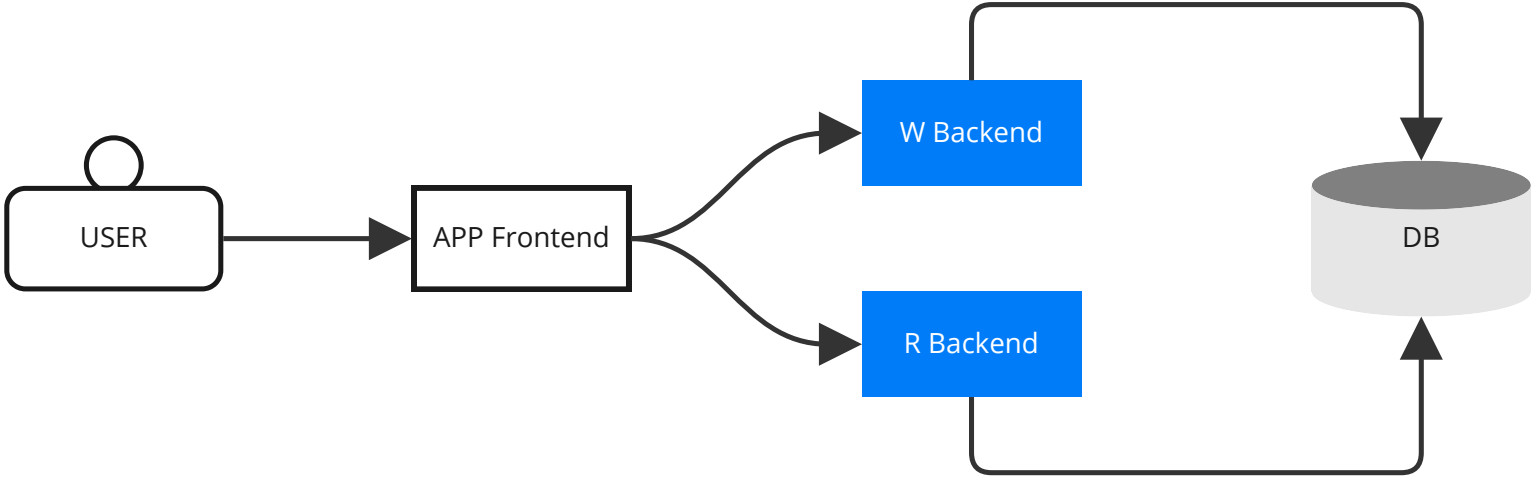
Read Model

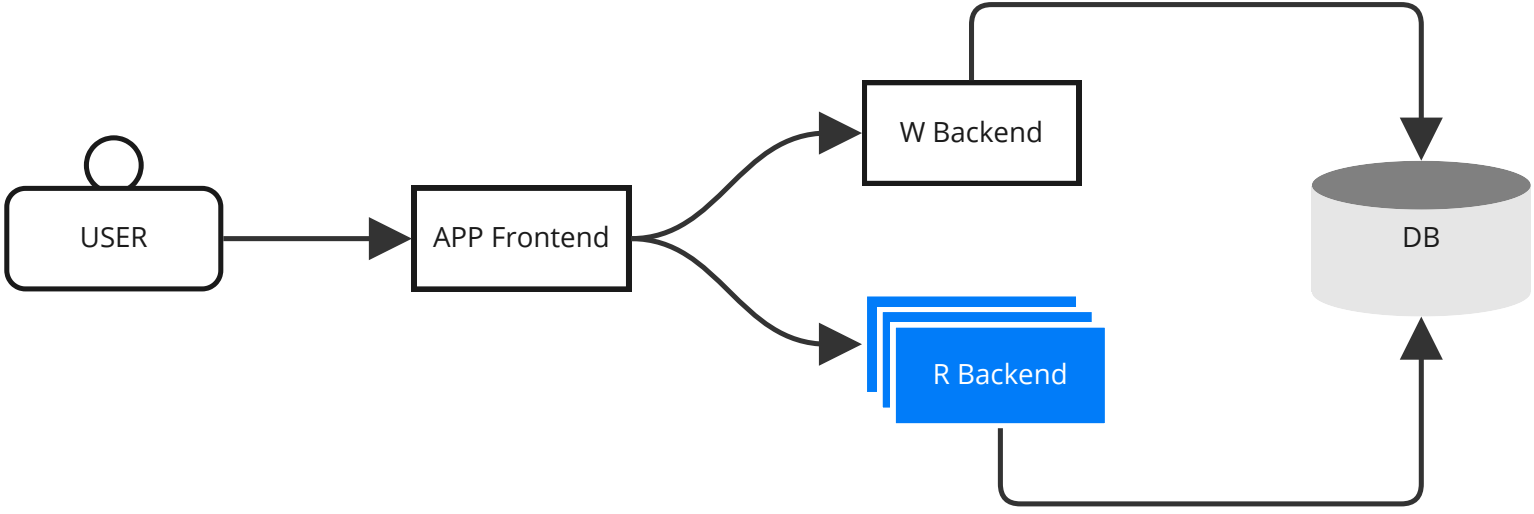
- Dapper + SqlKata | Linq2DB
- minimum abstraction, anemic model, no DAL, ...
- data driven architecture

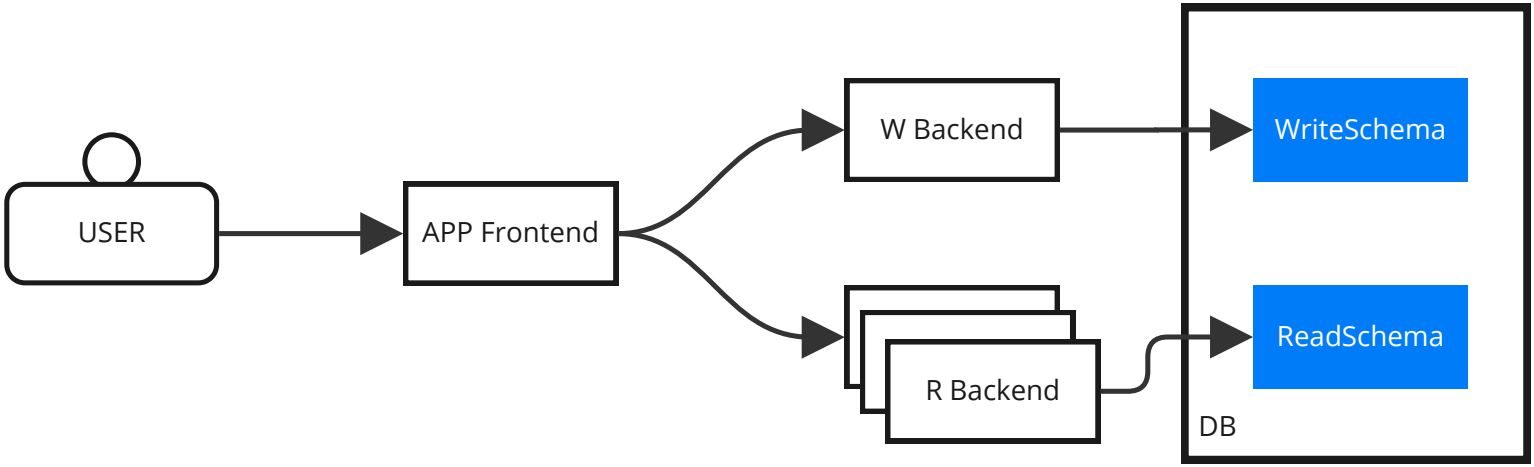
~~complexity~~

performance


USER
APP Frontend
WriteModel
ReadModel
APP Backend
DB

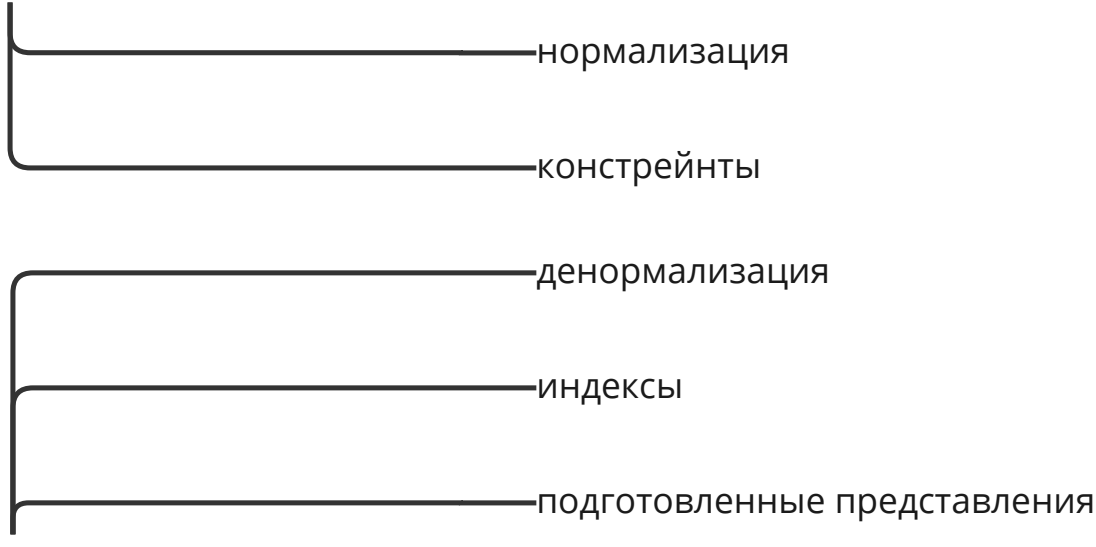

USER
APP Frontend
W Backend
R Backend
DB


USER
APP Frontend
W Backend
R Backend
DB

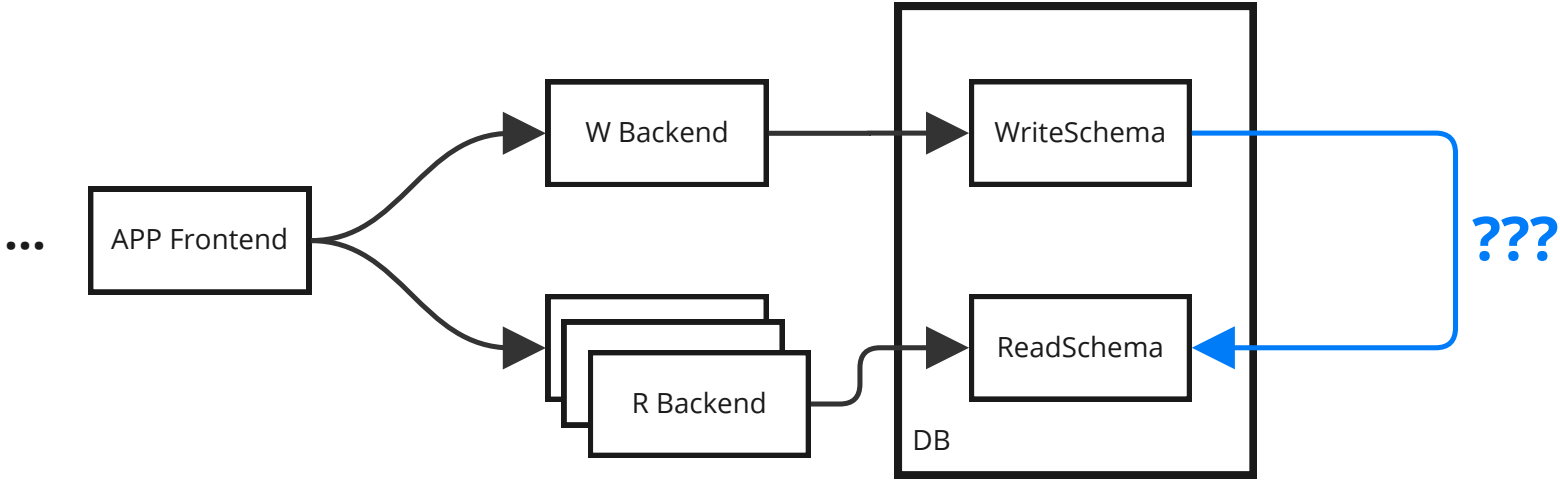

USER
APP Frontend
W Backend
R Backend
WriteSchema
ReadSchema
DB

CQRS #2

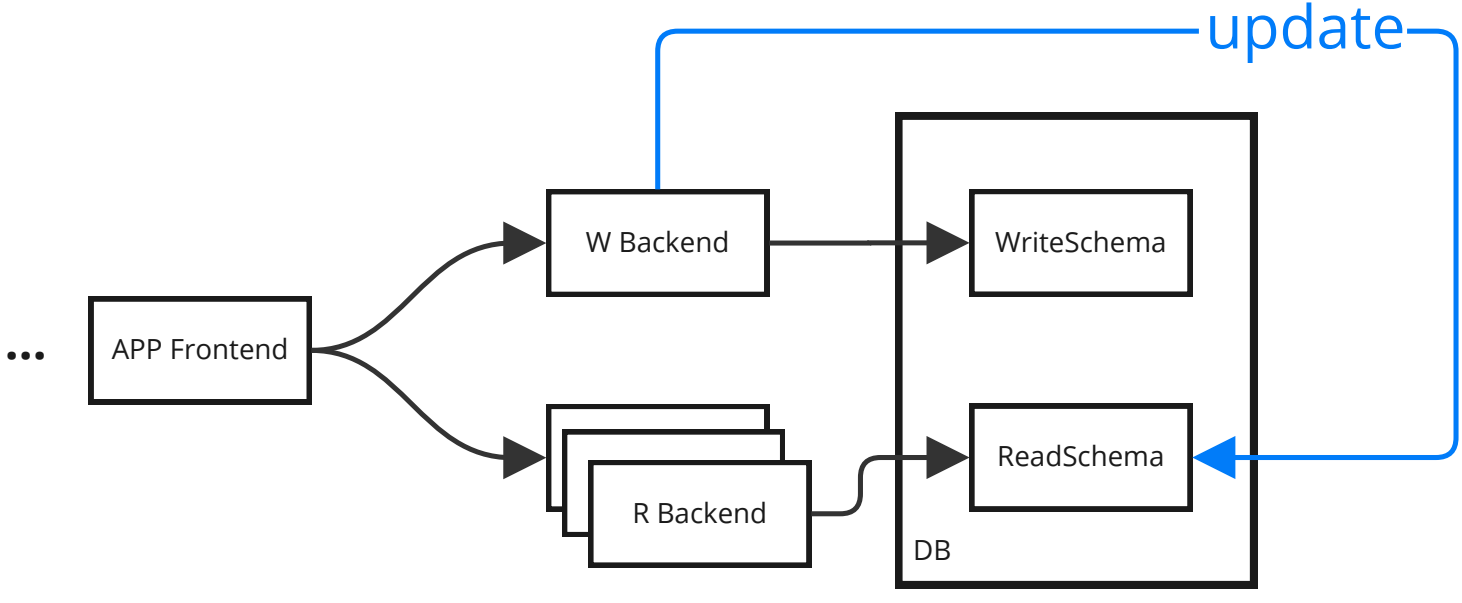
Write Schema



Read Schema


...
APP Frontend
W Backend
R Backend
WriteSchema
ReadSchema
DB
???

sync *vs* async

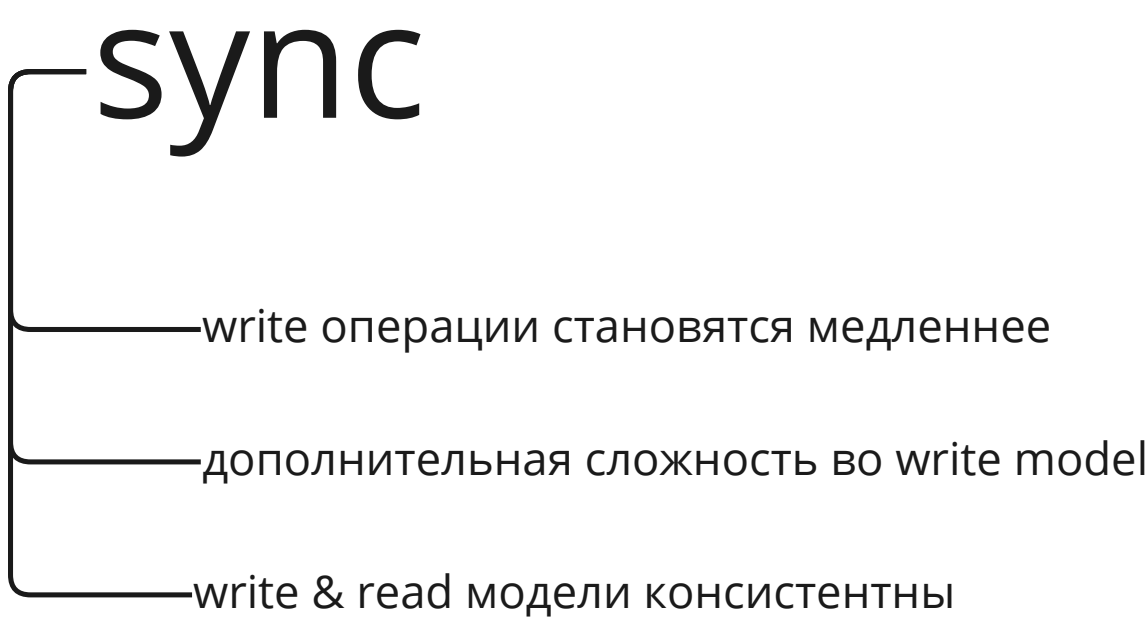

update
W Backend
WriteSchema
...
APP Frontend
R Backend
ReadSchema
DB

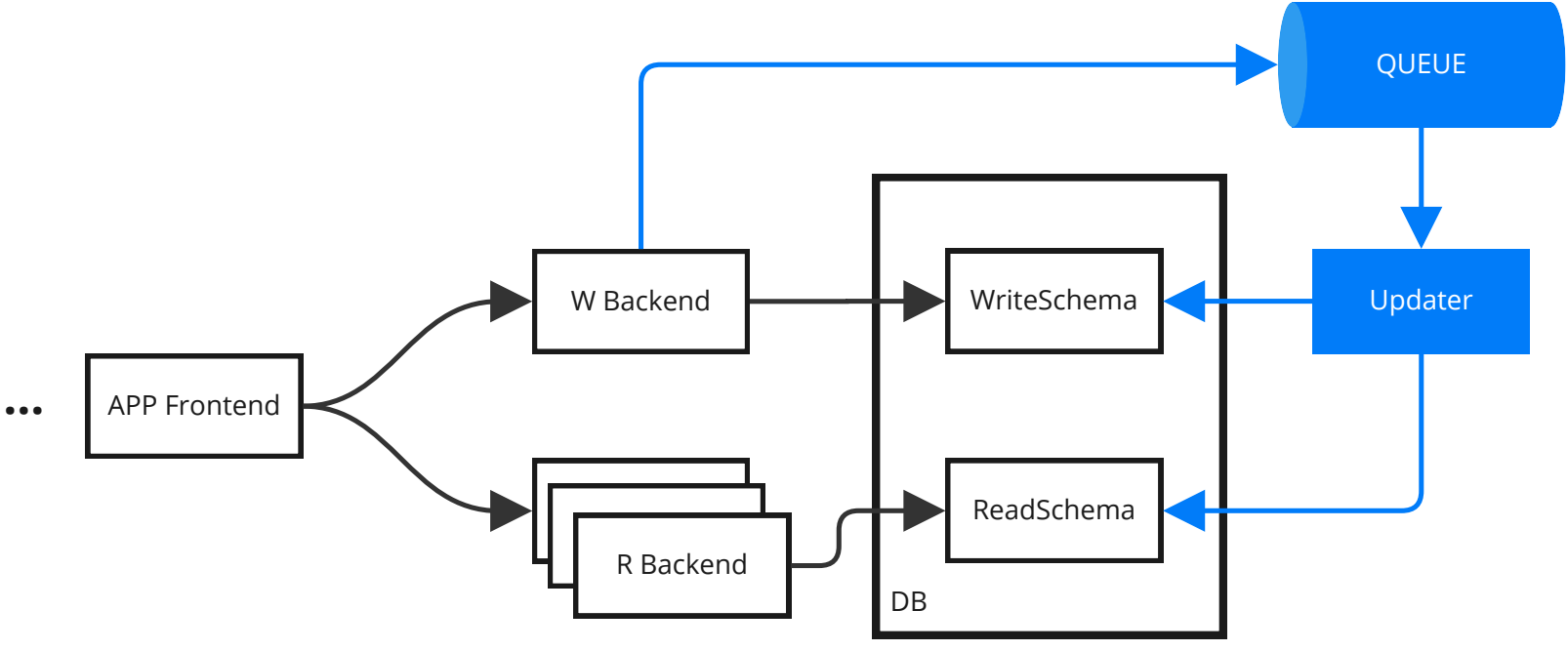
```
public sealed class ChangeUserNameCommandHandler: ICommandHandler<ChangeUserNameCommand, Result>
{
    private readonly IUserRepository _userRepository;
    private readonly IChangeNotifier _changeNotifier;

    ...

    public Result Handle(ChangeUserNameCommand command)
    {
        var user = _userRepository.GetById(command.UserId);

        if (!user.CanChangeName() || !IsValidNewName(command.NewName))
        {
            return Result.Error(...);
        }

        user.Name = command.NewName;
        _changeNotifier.Notify(new UserChangeNotification(command.UserId));

        return Result.Ok();
    }
}
```

```
public sealed class ReadModelUpdateDecorator<TCommand, TResult>:  ICommandHandler<TCommand, TResult>
{
    private readonly ICommandHandler<TCommand, TResult> _decoratedHandler;
    private readonly IChangeNotificationsProvider _notificationsProvider;
    private readonly IChangeNotificationDispatcher _dispatcher;

    ...

    public TResult Handle(TCommand command)
    {
        using var transaction = new TransactionScope();

        var result = _decoratedHandler.Handle(command);
        foreach (var notification in _notificationsProvider.GetNotifications())
        {
            _dispatcher.Handle(notification);
        }

        transaction.Complete();
        return result;
    }
}
```

```
public sealed class UserShortViewUpdater : IChangeNotificationHandler<UserChangeNotification>
{
    ...

    public void Handle(UserChangeNotification notification)
    {
        // update read model here
        ...
    }
}
```

```
_dbContext
    .ChangeTracker
    .Entries()
    .Where(e => e is IAggregate)
    .Select(e => (e as IAggregate).ToChangedNotification())
    .ToList()
    .ForEach(cn => _dispatcher.Handle(cn));
```


sync
write операции становятся медленнее
дополнительная сложность во write model
write & read модели консистентны


QUEUE
W Backend
WriteSchema
Updater
APP Frontend
R Backend
ReadSchema
DB

async

- write операции не теряют в производительности значительно
- дополнительная сложность во write model
- дополнительная архитектурная сложность и новые точки отказа
- eventually consistency

ДОКЛАД Architecture 10.09 / 12:00 – 13:00 (UTC+3)

Outbox: сложно о, казалось бы, простом

RU

Outbox — известный архитектурный паттерн, о котором написано множество статей, рассказано в докладах и даже написаны книги.

В этот раз обсудим, с какими неожиданными проблемами сталкиваются разработчики при больших нагрузках и как не выстрелить себе в ногу.

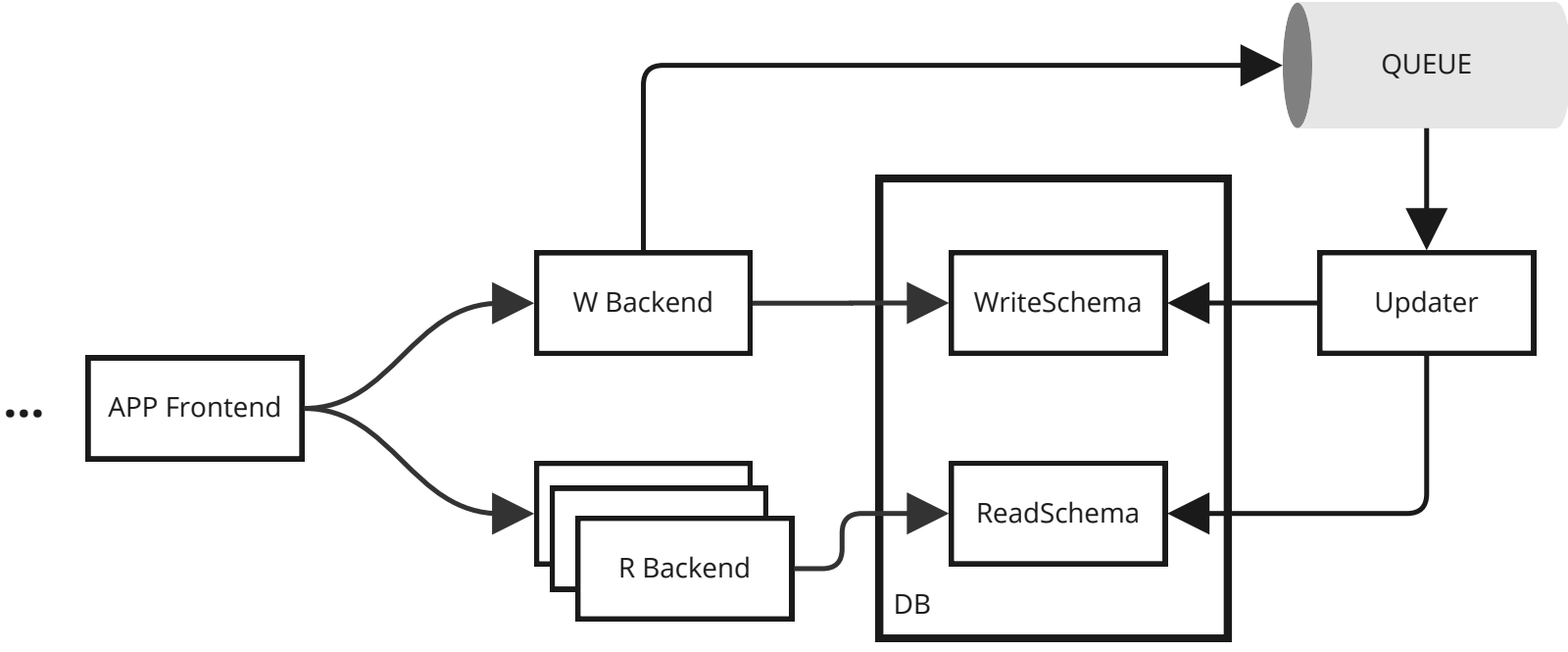
Спикеры

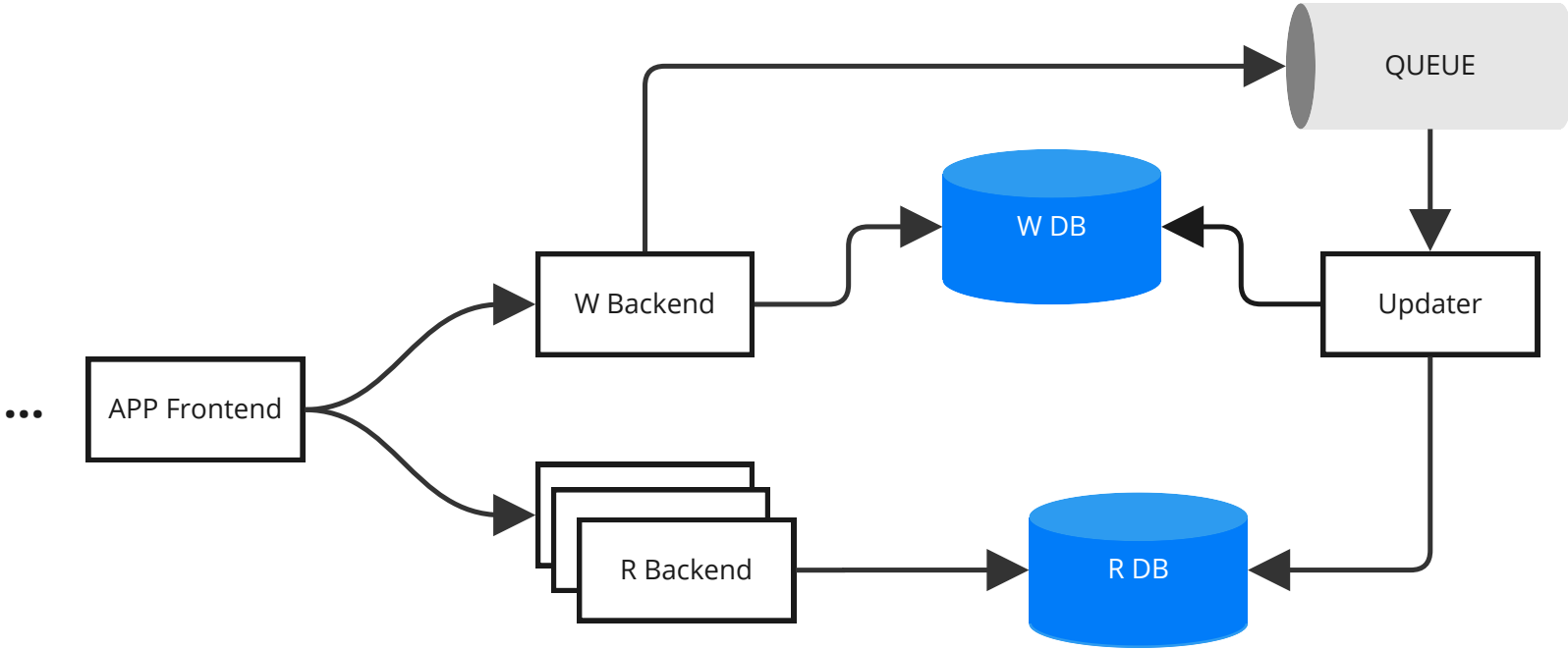
Борис Кузоваткин
Ex-Ozon

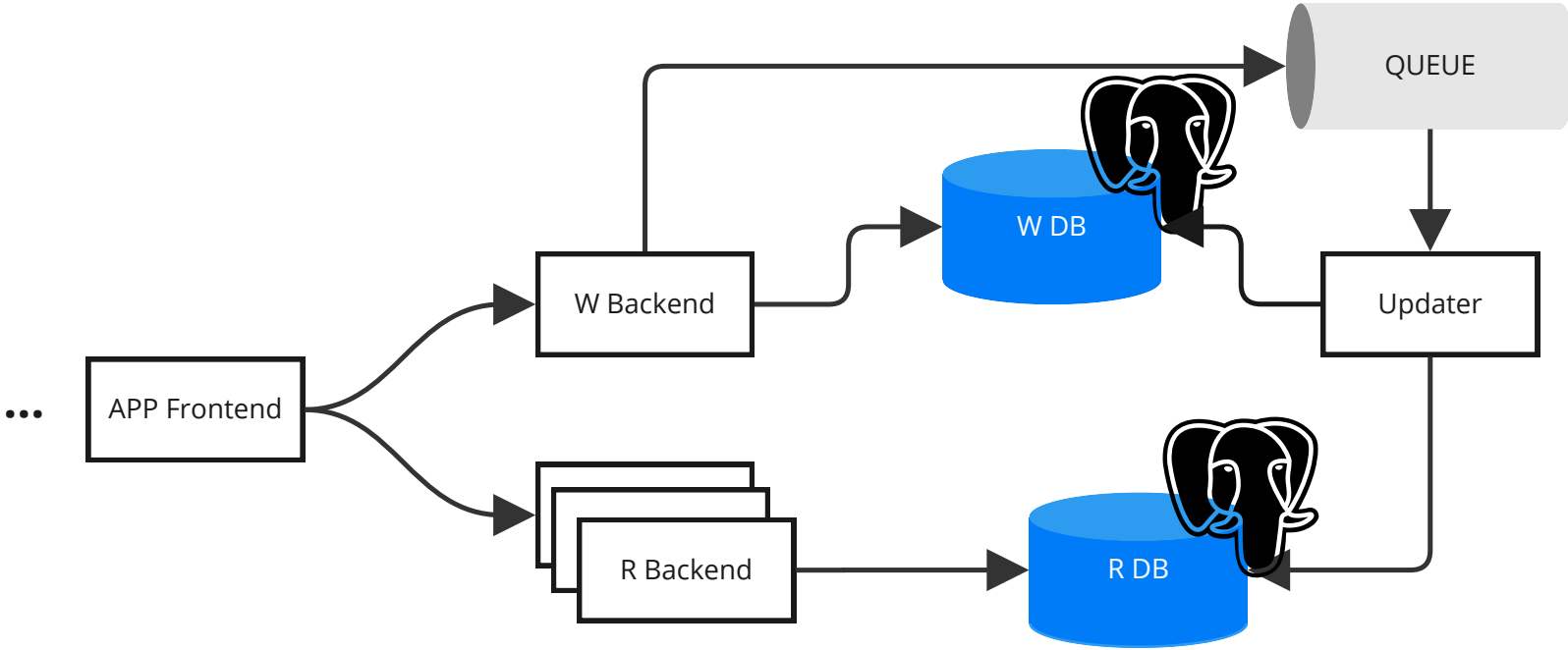
~~complexity~~

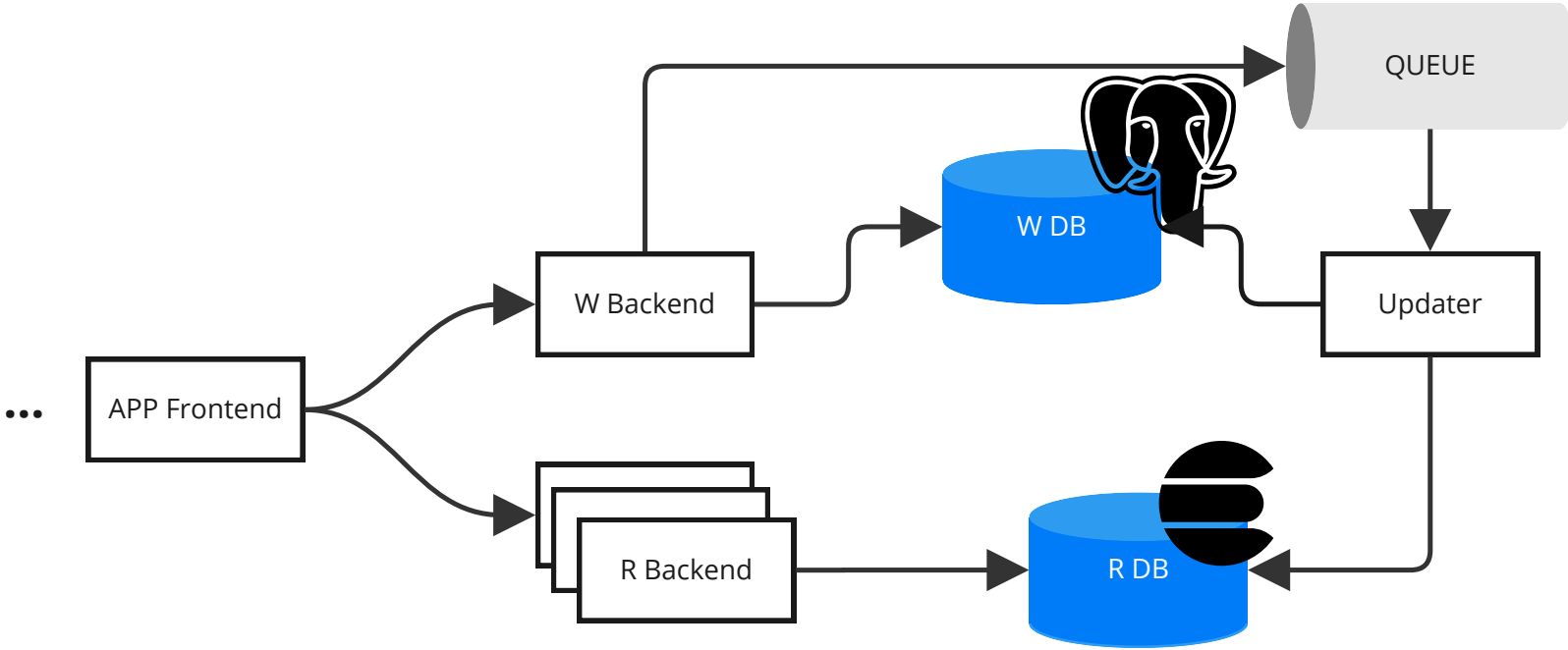
~~performance~~

нужно больше performance


QUEUE
W Backend
WriteSchema
Updater
APP Frontend
ReadSchema
R Backend
DB


QUEUE
W DB
W Backend
Updater
APP Frontend
R Backend
R DB


QUEUE
W DB
W Backend
Updater
APP Frontend
R Backend
R DB


QUEUE
W DB
W Backend
Updater
APP Frontend
R Backend
R DB

CQRS #3

sync - фантастический вариант

eventually consistency - данность

~~complexity~~

~~performance~~

command

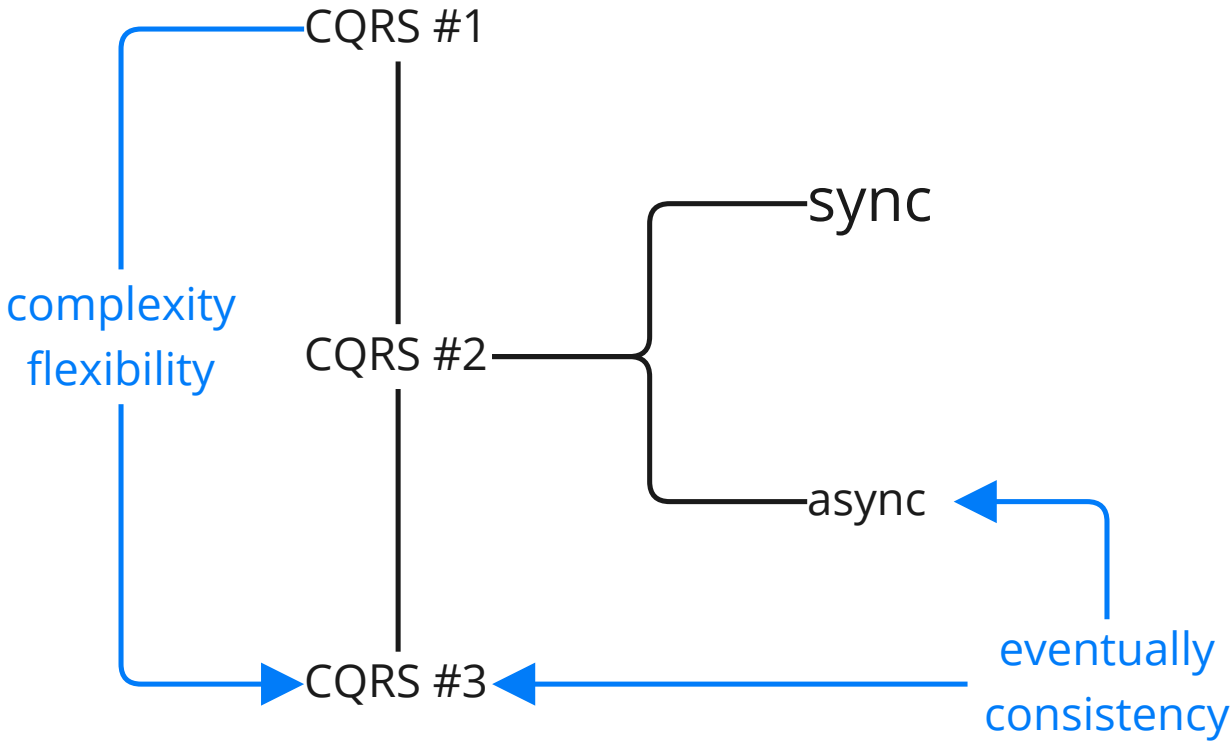
& query?

```
public sealed record ChangeUserNameCommand(Guid UserId, string NewName) : ICommand;

...

public sealed class ChangeUserNameCommandHandler: ICommandHandler<ChangeUserNameCommand, Result>
{
    ...

    public Result Handle(ChangeUserNameCommand command)
    {
        var user = _userRepository.GetById(command.UserId);

        if (!user.CanChangeName() || !IsValidNewName(command.NewName))
        {
            return Result.Error(...);
        }

        user.Name = command.NewName;
        _changeNotifier.Notify(new UserChangeNotification(command.UserId));

        return Result.Ok();
    }
}
```

- один сценарий = один handler
- единство ответственности
- удобное управление зависимостями
- удобно реализовывать сквозную логику
- управление сложностью
- это не главное в CQRS


DOTNEXT 2018 Moscow
Максим Аршинов
Хайтек Груп
Быстрорастворимое проектирование

analysis

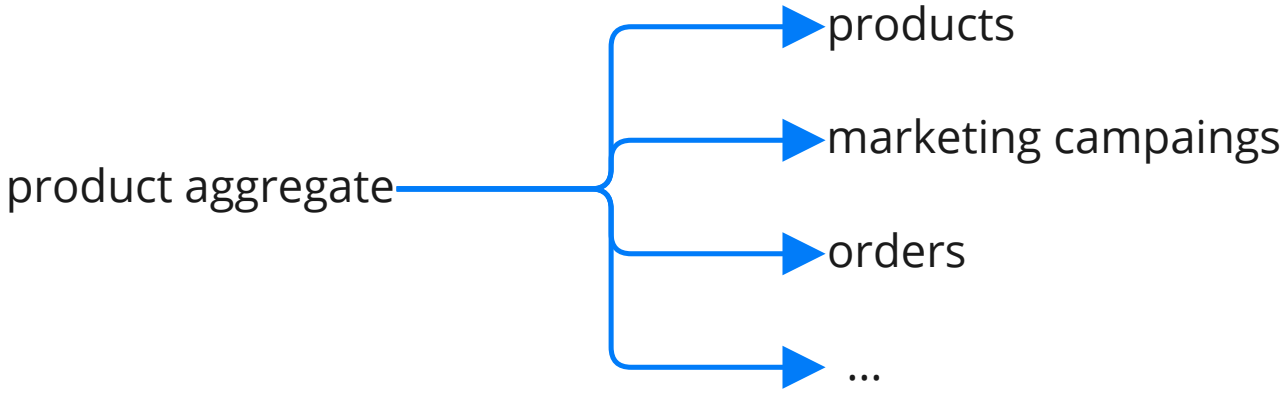

CQRS #1
sync
complexity
flexibility
CQRS #2
async
eventually
consistency
CQRS #3

fcmp(rwmodel) < fcmp(rmodel) ∪ fcmp(wmodel)
fcmp(rwmodel) > fcmp(rmodel)
fcmp(rwmodel) > fcmp(wmodel)

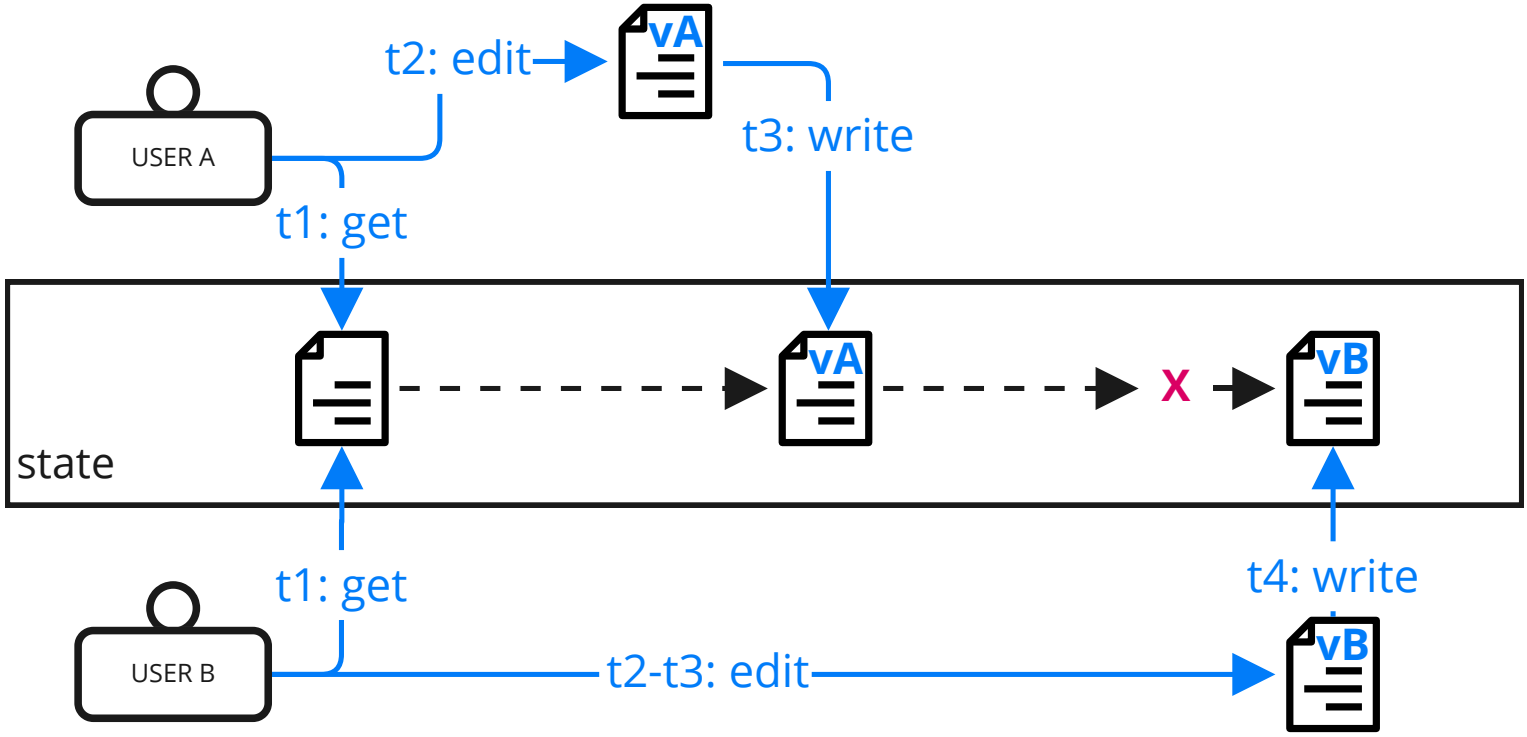
fcmp - complexity

```
public class UserRepository: IUserRepository {
    ...

    public void Create(User user) {...}
    public void Update(User user) {...}
    public void Delete(User user) {...}

    public User GetById(Guid id) {...}
    public User GetRecentActive() {...}
    public User? TryFind(string searchString) {...}
    public User[] GetAllOnFreePlan() {...}
    public User[] GetAllWithoutBillingInformation() {...}
    ...
    // 100500 methods later
    public User[] GetWhoCommentedArticleInProject(Guid projectId) {...}
}
```

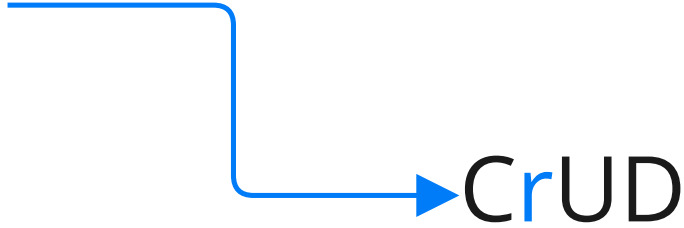
обновление read model может быть далеко не простым



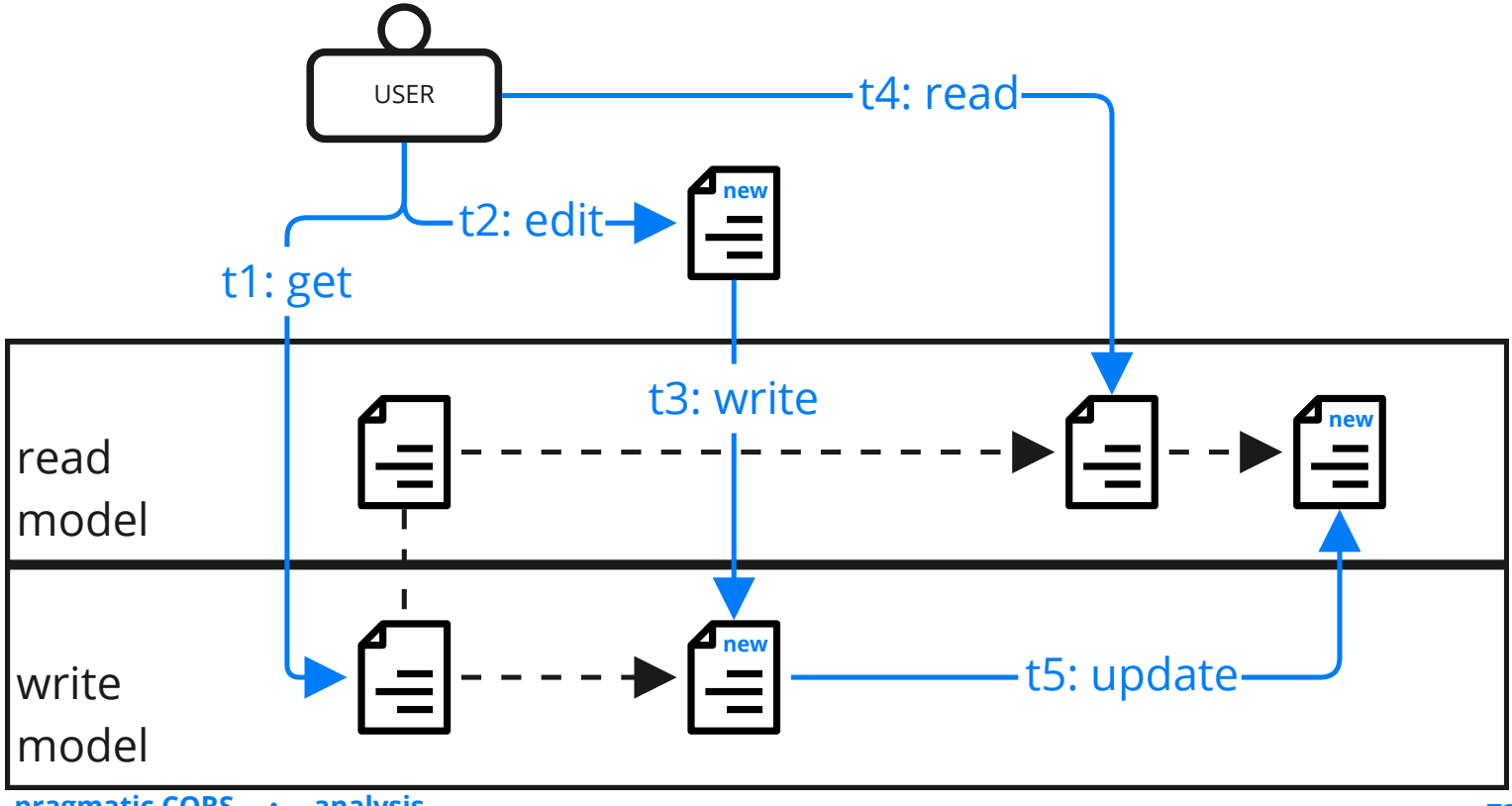
eventually consistency

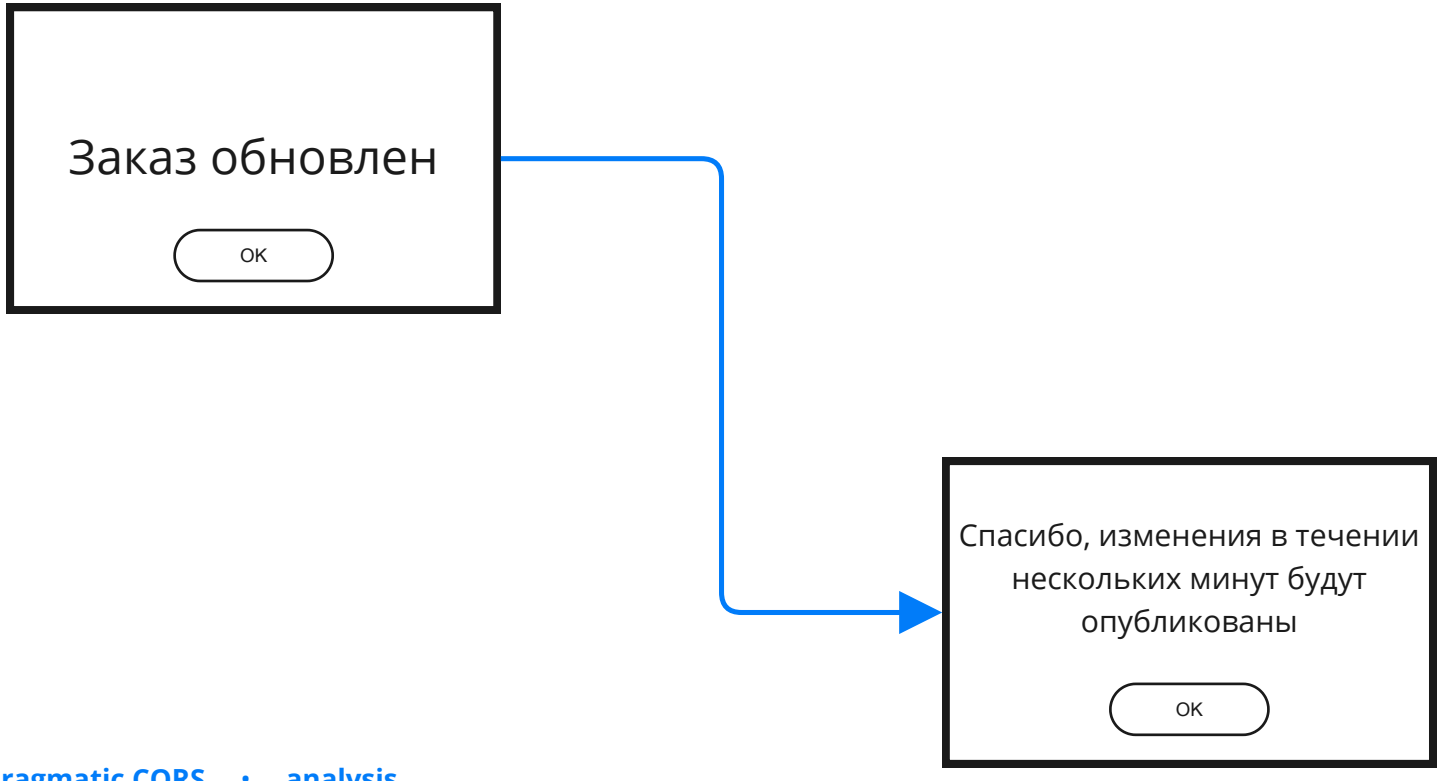

t2: edit
vA
t3: write
USER A
t1: get
vA
X
vB
state
t1: get
t4: write
USER B
t2-t3: edit
vB

write model: CUD



versioning check & (merge or conflict solving)


USER
t4: read
t2: edit
new
t1: get
t3: write
read
model
new
write
model
new
t5: update


Заказ обновлен
OK
Спасибо, изменения в течении нескольких минут будут опубликованы
OK

bussines solution

CQRS полезно, но осторожно

QA

akulyakov_tech