CQRS на практике. В поиске точки масштабирования и новых метафор

Александр Бындю
ByndyuSoft

Обо мне

1. Владелец компании ByndyuSoft
   http://byndyusoft.com
2. Консультант по вопросам разработки приложений и организации работы IT компаний
3. Внештатный сотрудник Академии АйТи
4. Технический блог http://blog.byndyu.ru
5. Преподаю в ЮУрГУ и ЧелГУ
6. Тренер на AgileCamp
7. Организую конференции .NET-разработчиков
   http://dotnetconf.ru
8. Веду группу по проблемам разработки приложений
   https://groups.google.com/forum/?hl=ru&fromgroups#!forum/dotnetconf

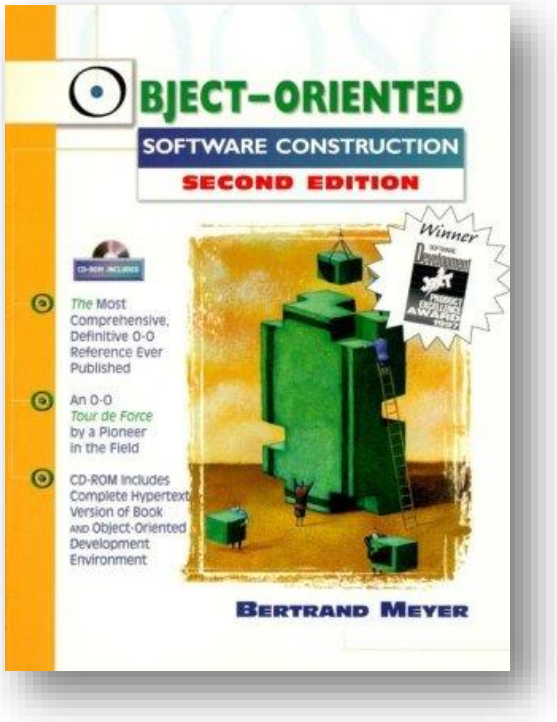
План

1. Основная теория CQRS
2. Эволюция кода
3. Эволюция архитектуры
4. Event Sourcing
5. Ограничения
6. Примеры реализации и подходы

1. Основная теория CQRS

Bertrand Meyer

«Object Oriented Software Construction»
1994 г.



Command-query separation (CQS)

Методы объекта нужно разделить на:

1. **Queries**: Возвращают результат, не изменяя состояние объекта

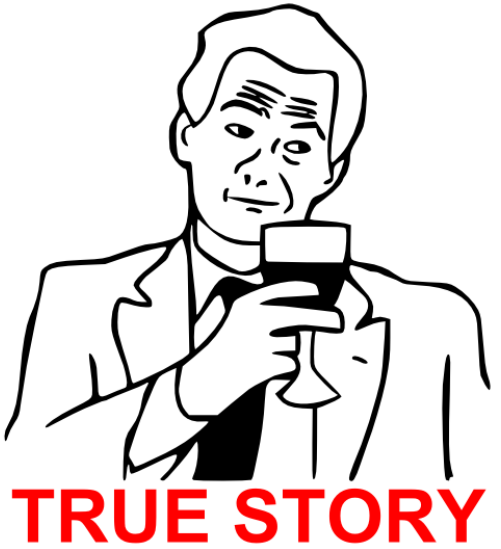
free of side effects

2. **Commands**: Изменяют состояние, не возвращая значение.

Что значит Command?

Более корректно называть:

1. Modifiers
2. Mutators

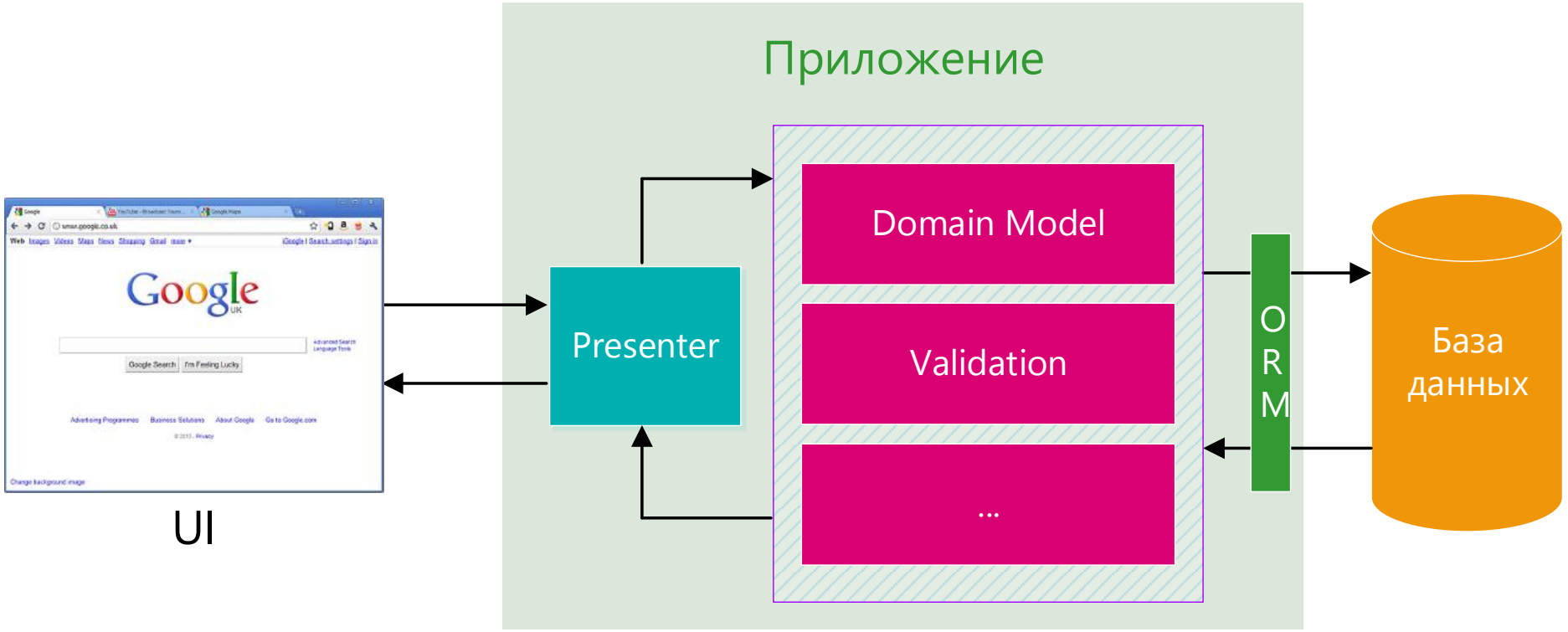


```
public class User
{
    public string Email { get; private set; }

    public bool IsEmailValid(string email)
    {
        bool isMatch = Regex.IsMatch("email pattern", email);

        if (isMatch)
        {
            Email = email;
        }

        return isMatch;
    }
}
```

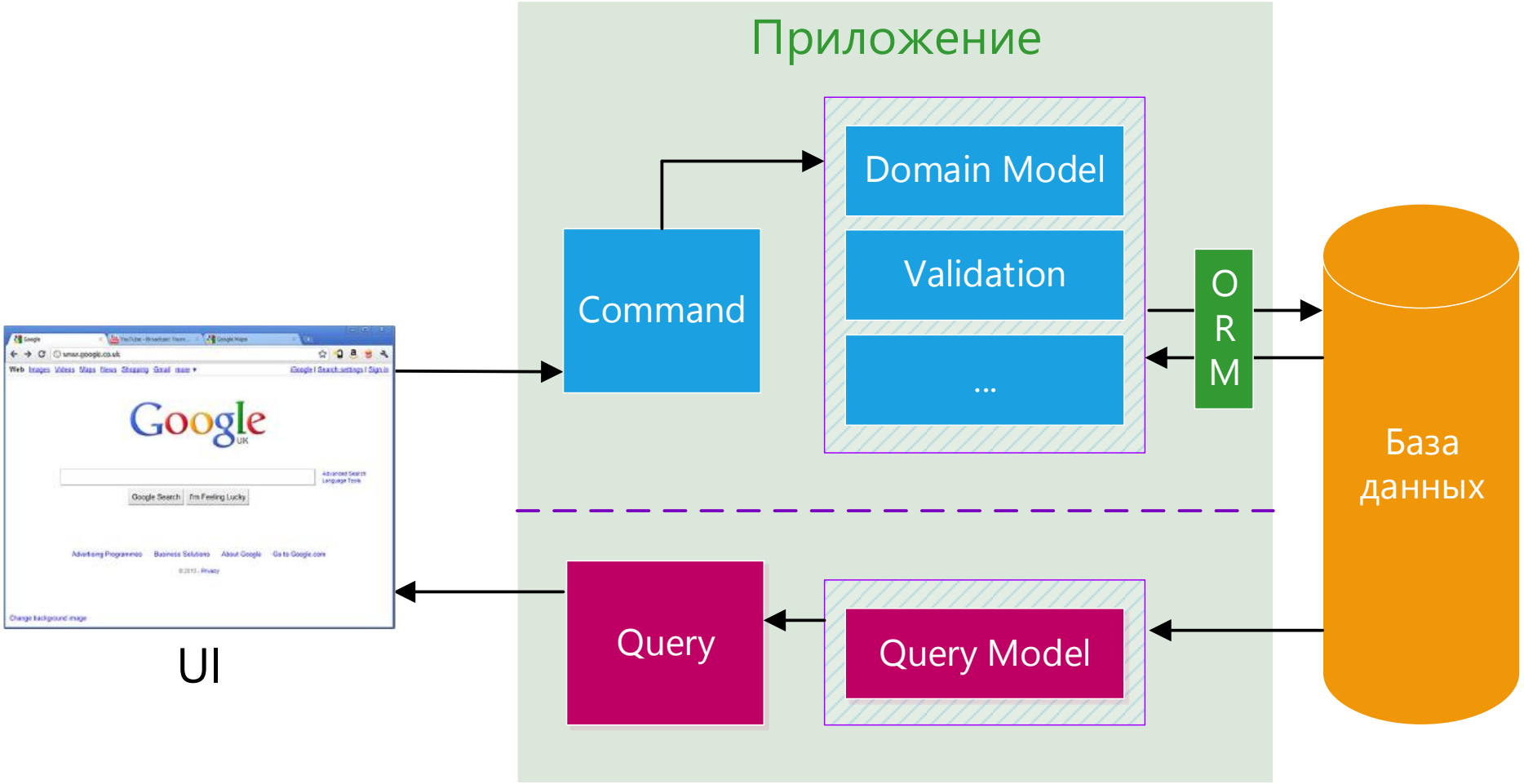
```
public class User
{
    public string Email { get; private set; }

    public bool IsEmailValid(string email)
    {
        return Regex.IsMatch("email pattern", email);
    }

    public void ChangeEmail(string email)
    {
        if (IsEmailValid(email) == false)
            throw new ArgumentOutOfRangeException(email);
        Email = email;
    }
}
```

Query

Command


Приложение
Domain Model
Validation
...
Presenter
O
R
M
База данных
UI

Command-query responsibility segregation (CQRS)



Command

- Изменяет состояние системы
- Контекст команды хранит нужные для ее выполнения данные
- Ничего не возвращает
- Хорошо описывает предметную область

```
public class DeleteUserHandler : ICommandHandler<DeleteUser>
{
    private readonly ISession session;

    public DeleteUserHandler(ISession session)
    {
        this.session = session;
    }

    public void Execute(DeleteUser context)
    {
        session.Delete<User>(context.UserId);
    }
}
```

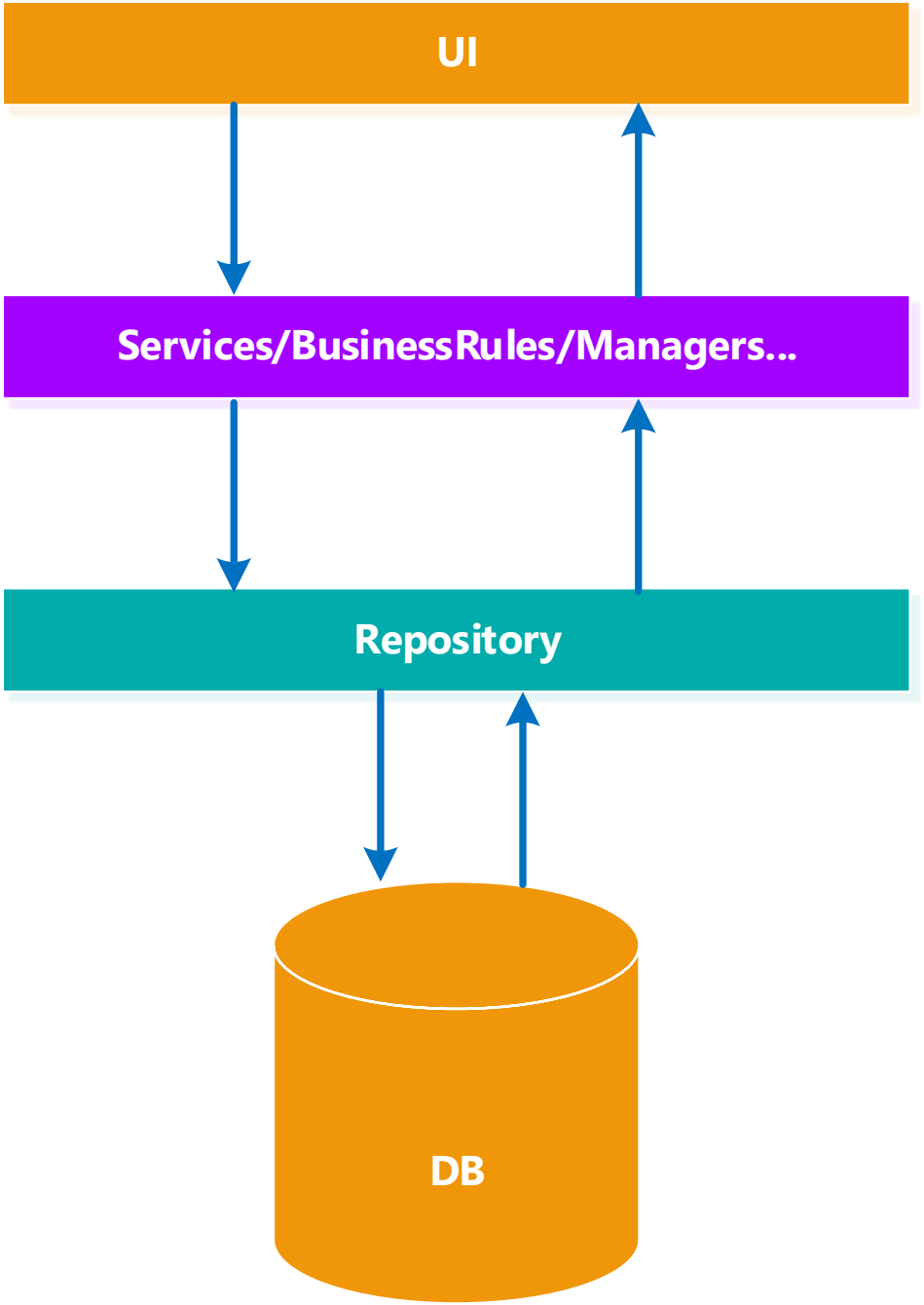
Query

free of side effects

1. Не изменяет состояние системы
2. Контекст запроса хранит нужные для ее выполнения данные (пейджинг, фильтры и т.п.)
3. Возвращает результат

```
public class FindUserByIdQuery : IQuery<FindById, User>
{
    private readonly ISession session;

    public FindUserByIdQuery(ISession session)
    {
        this.session = session;
    }

    public User Ask(FindById context)
    {
        return session.Query<User>()
            .SingleOrDefault(x => x.Id == context.Id);
    }
}
```

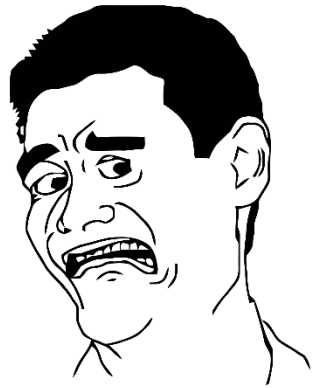
2. Эволюция кода


UI
Services/BusinessRules/Managers...
Repository
DB

Repository v1.0

```
public interface IRepository<TEntity>
{
    void Create(TEntity entity);

    TEntity Get(int id);

    void Update(TEntity entity);

    void Delete(TEntity entity);
}
```

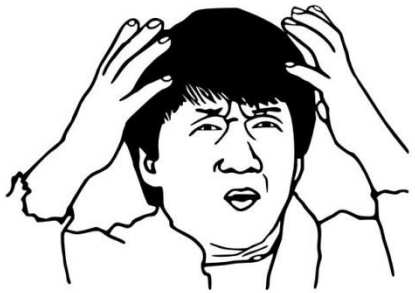
Repository v2.0
Нужно больше методов



```
public class AccountRepository : IRepository<Account>
{
    public IEnumerable<Account> GetActiveAccounts()
    {
        // ...
    }

    public void ChangeAccountAddress(int id, string newAddress)
    {
        // ...
    }

    public IEnumerable<Account> GetPremiumAccountsByManager()
    {
        // ...
```

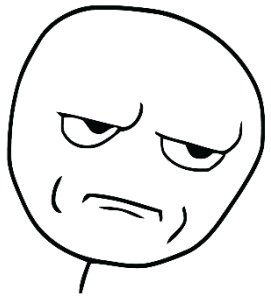
Repository v2.0
Нужно больше зависимостей

```
public class AccountRepository : IRepository<Account>
{
    public AccountRepository(
        IPriceCalculator priceCalculator,
        IMessageDispatcher messageDispatcher,
        IEmailSender emailSender,
        IDataContext dataContext,
        IAwsProvider awsProvider,
        ISphinxProvider sphinxProvider,
        IMongoDbProvider mongoDbProvider
        /* ... */)
    {
```



Repository v3.0
Daeшь IQueryable!

```
public class AccountRepository : IRepository<Account>
{
    public IQueryable<Account> GetActiveAccounts()
    {
        // ...
    }

    public IQueryable<Account> GetPremiumAccountsByManager()
    {
        // ...
    }

    public Account GetAccountWithRoleInformation(int id)
    {
        // ...
    }
```



Дублирование условий

```
1. session.Query<User>()
        .Where(x => x.Activated);

2. session.Query<User>()
        .Where(x => x.Activated &&
                    x.Balance > 0);

3. session.Query<User>()
        .Where(x => x.Activated &&
                    x.Balance > 0 &&
                    ...);
```

Дублирование подгрузок

```
1. session.Query<User>()
       .Fetch(x => x.Bills);

2. session.Query<User>()
       .Fetch(x => x.Bills)
       .Fetch(x => x.Roles);

3. session.Query<User>()
       .Fetch(x => x.Bills)
       .Fetch(x => x.Roles)
       .Fetch(...);
```

Repository v4.0
Предикаты условий выборки

```
public class ActiveAccountSpecification : ISpecification<Account>
{
    public Func<Account, bool> IsSatisfiedBy()
    {
        return x => x.IsActive && x.Credit > 0;
    }
}
```

Repository v4.0
Стратегии подгрузки в отдельные классы

```
public class AccountCommentFetchStrategy : IFetchStrategy<Account>
{
    public Action<Account> Apply()
    {
        return x => x.Posts.Select(p => p.Comments);
    }
}
```

Repository v4.0

```
public class AccountRepository : IRepository<Account>
{
    public IEnumerable<Account> GetAccounts(
                        IFetchStrategy<Account>[] fetchStrategies,
                        ISpecification<Account>[] specifications)
    {
        // ...
    }
```

Еще круче собирать их через IoC-контейнер по конвенции

Repository v5.0

Для изменения состояния системы работаем с корнями агрегации.

Выборка данных для отображения собирает DTO из разных частей данных.

Repository v4.0

```
public class AccountRepository : IRepository<Account>
{
    public IEnumerable<Account> GetActiveAccounts()
    {
        // ...
    }

    public void ChangeAccountAddress(int id, string newAddress)
    {
        // ...
    }

    public IEnumerable<Account> GetPremiumAccountsByManager()
    {
        // ...
    }
```

GetActiveAccountsQuery

ChangeAccountAddressCommand

GetPremiumAccountsByManagerQuery

Отдельный класс из метода

```
public class FindPremiumAccountsByManagerQuery :
                IQuery<FindPremiumAccountsByManager, User>
{
    private readonly ISession session;

    public FindPremiumAccountsByManagerQuery (ISession session)
    {
        this.session = session;
    }

    public User Ask(FindPremiumAccountsByManager context)
    {
        return session.Query<User>()…;
    }
}
```

Services/Managers/BusinessRules

Такая же история:

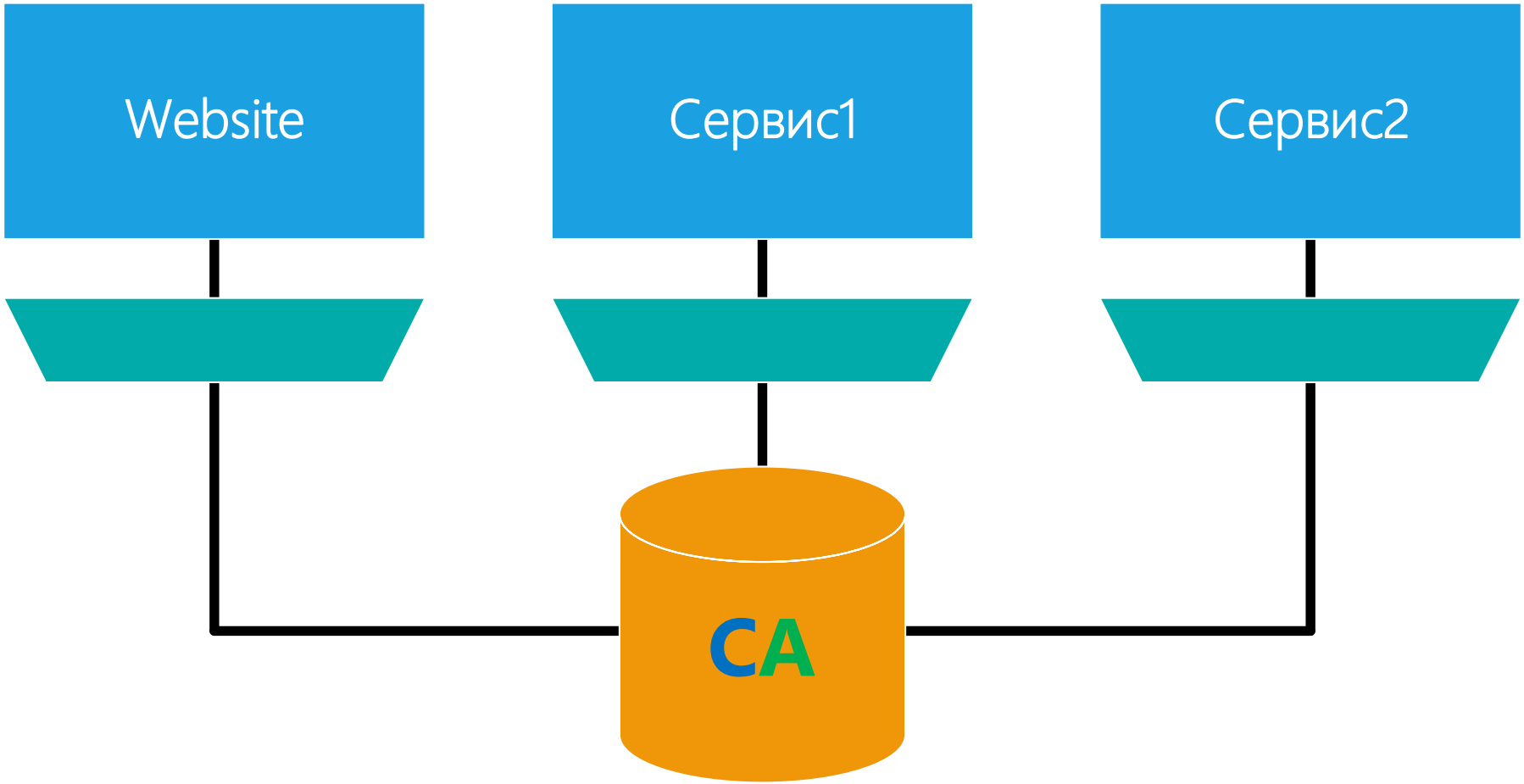
1. Растет количество классов такого типа
2. Растет количество методов
3. Растет количество зависимостей каждого класса
4. Разбиваем сервисы на Command и Query

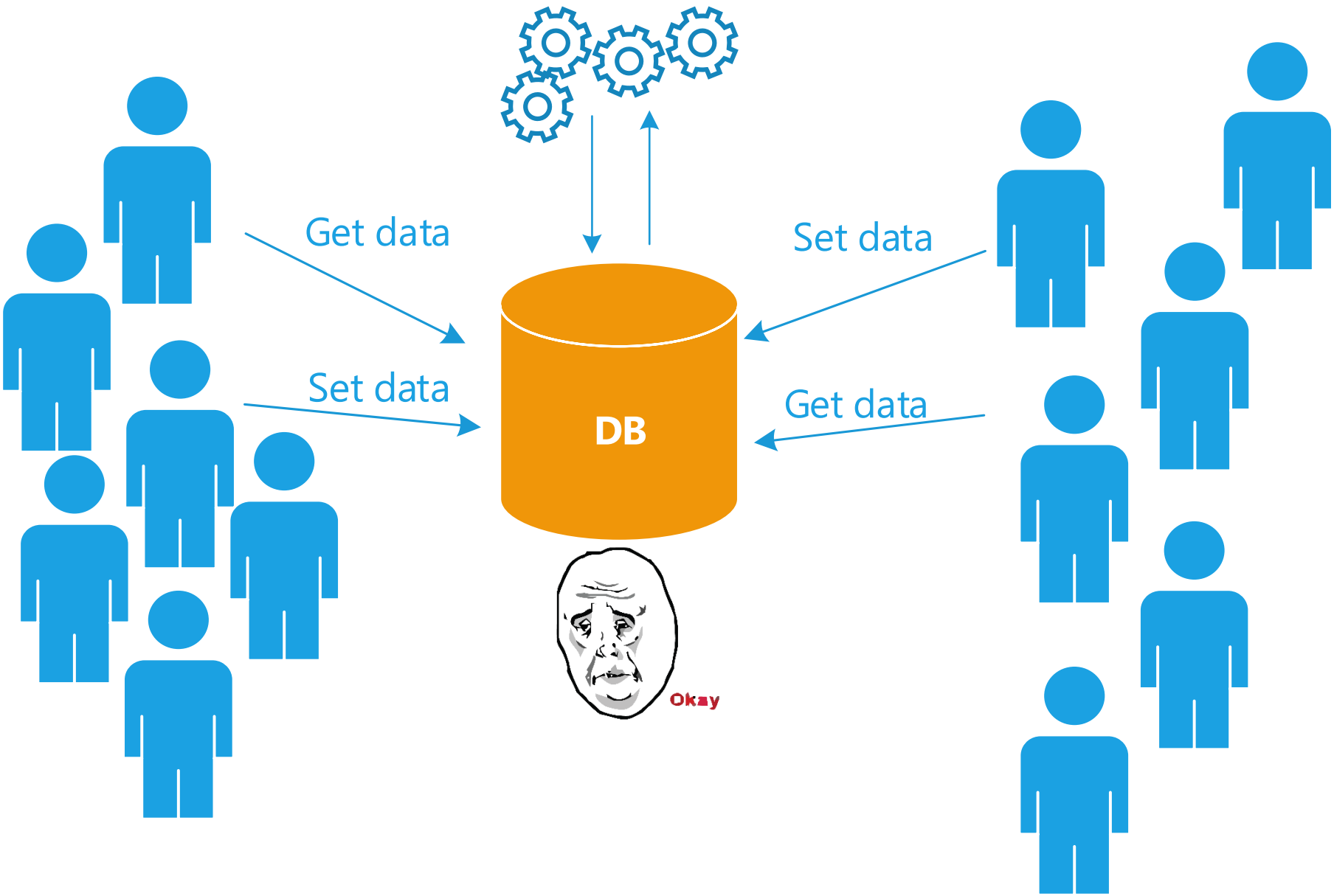
Маленькие объекты

1. Меньше зависимостей в каждом классе
2. SRP
3. Проще заменить
4. Проще тестировать
5. Делают дизайн кода однотипным
6. При расширении функциональности системы сложность увеличивается (почти) линейно

3. Эволюция архитектуры

Shared DB




Get data
Set data
Set data
Get data
DB
Okay


А ЧТО ЕСЛИ ВАША
АРХИТЕКТУРА
НЕ МАСШТАБИРУЕТСЯ?
troll-face.ru

Что делать?

1. Оптимизировать скрипты выборки
2. Убираем ORM для лучшей оптимизации
3. Убираем весь код выборки в хранимки
4. Оптимизируем индексы
5. Денормализуем данные

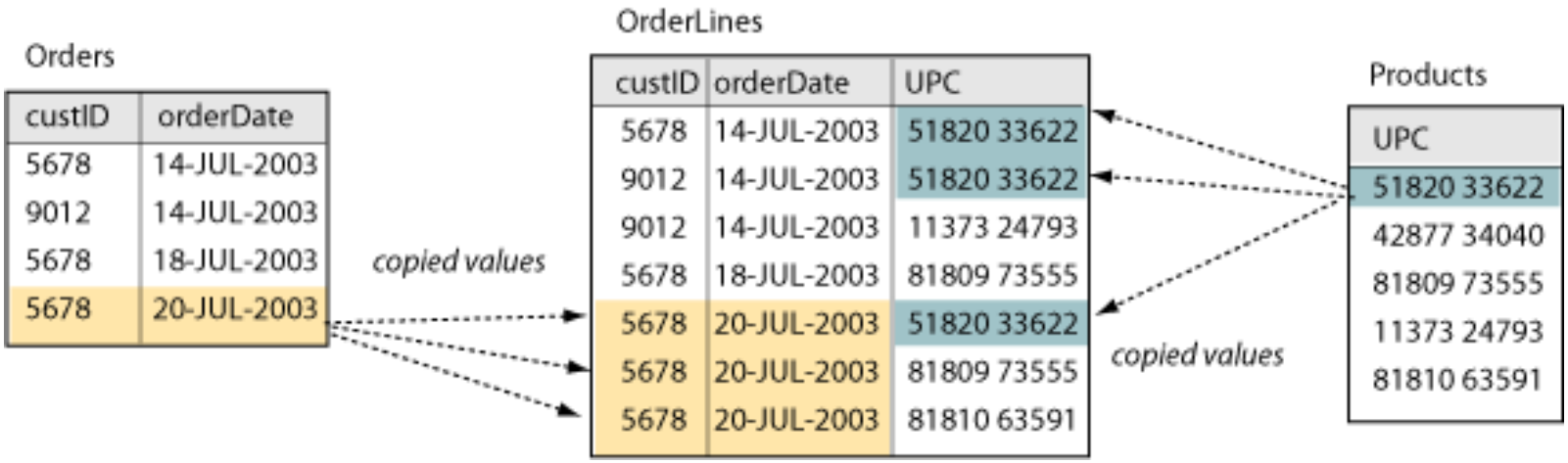
Денормализация v1.0

Создать дополнительные колонки в текущих таблицах

```
SELECT pt.Code
FROM Products p
INNER JOIN ProductType pt ON
p.ProductTypeID = pt.ProductTypeID
WHERE p.ProductID = 20
```

Денормализация v2.0

Создать отдельные таблицы/view для денормализованных данных



Денормализация v3.0

Создать еще одну БД (хранилище) с «плоскими» данными для чтения

1. Отдельная реляционная БД с «плоскими» данными без связей
2. Различные NoSQL
3. Поисковые системы

cRud

1. «Плоский» SQL
2. NoSQL
3. Поисковые системы
4. Кэши
5. …

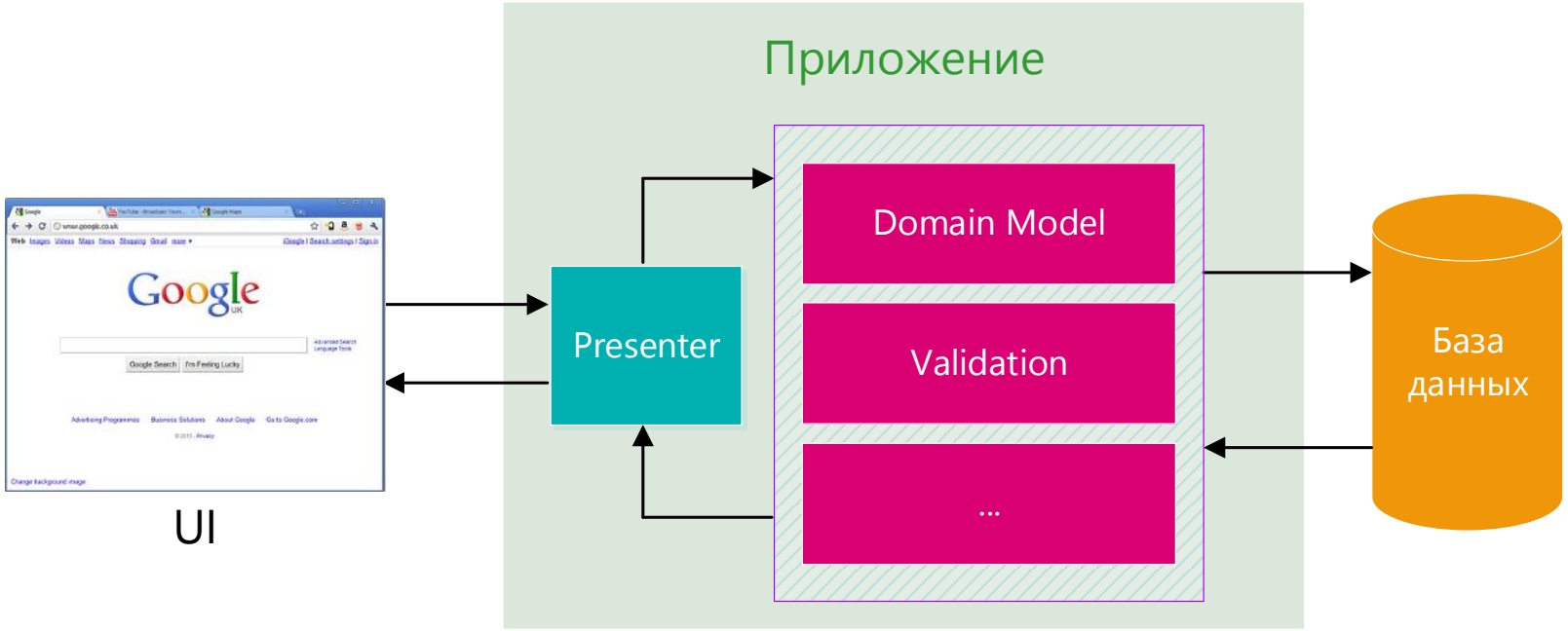
UI

Только выборка

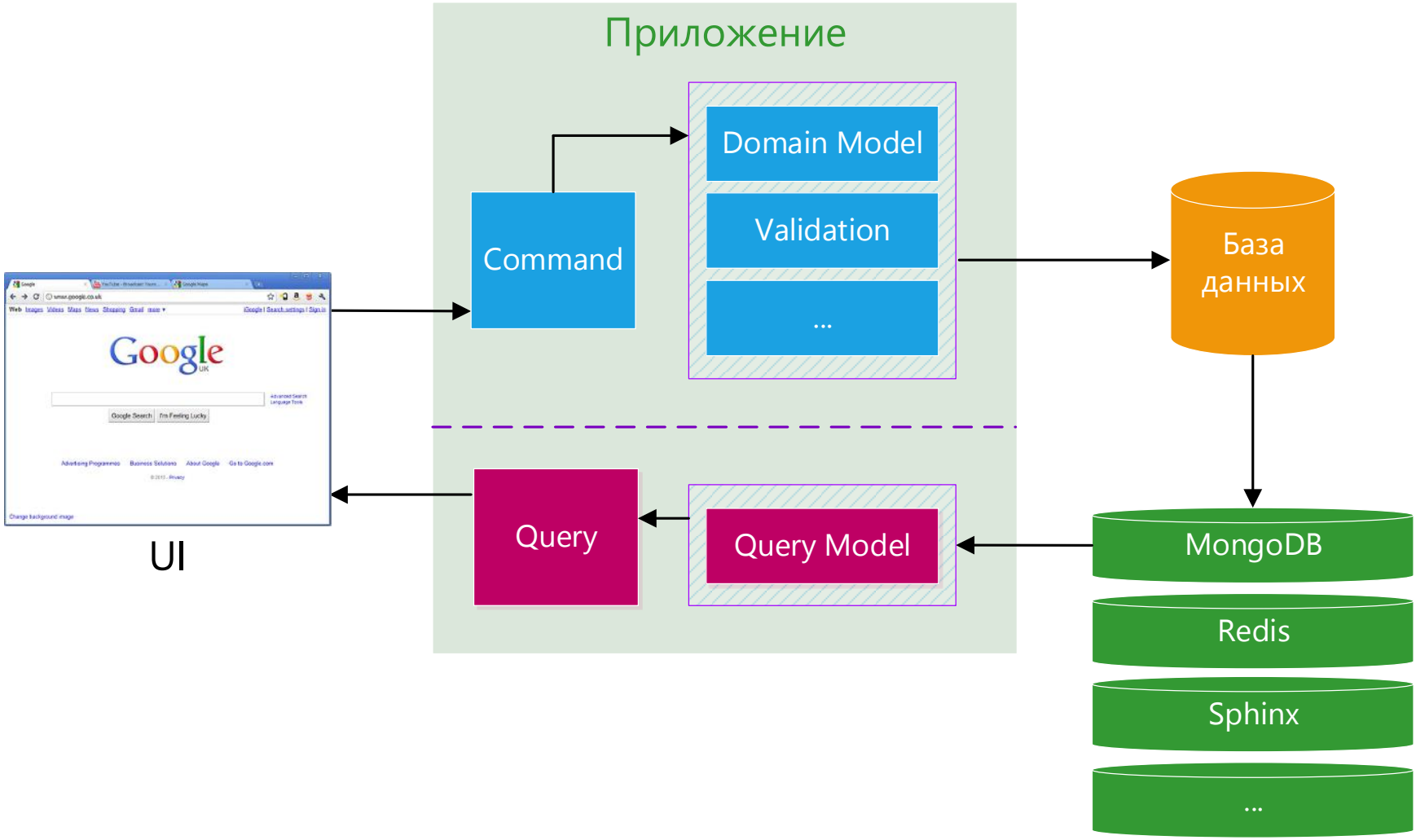
«Плоские» данные

CrUD

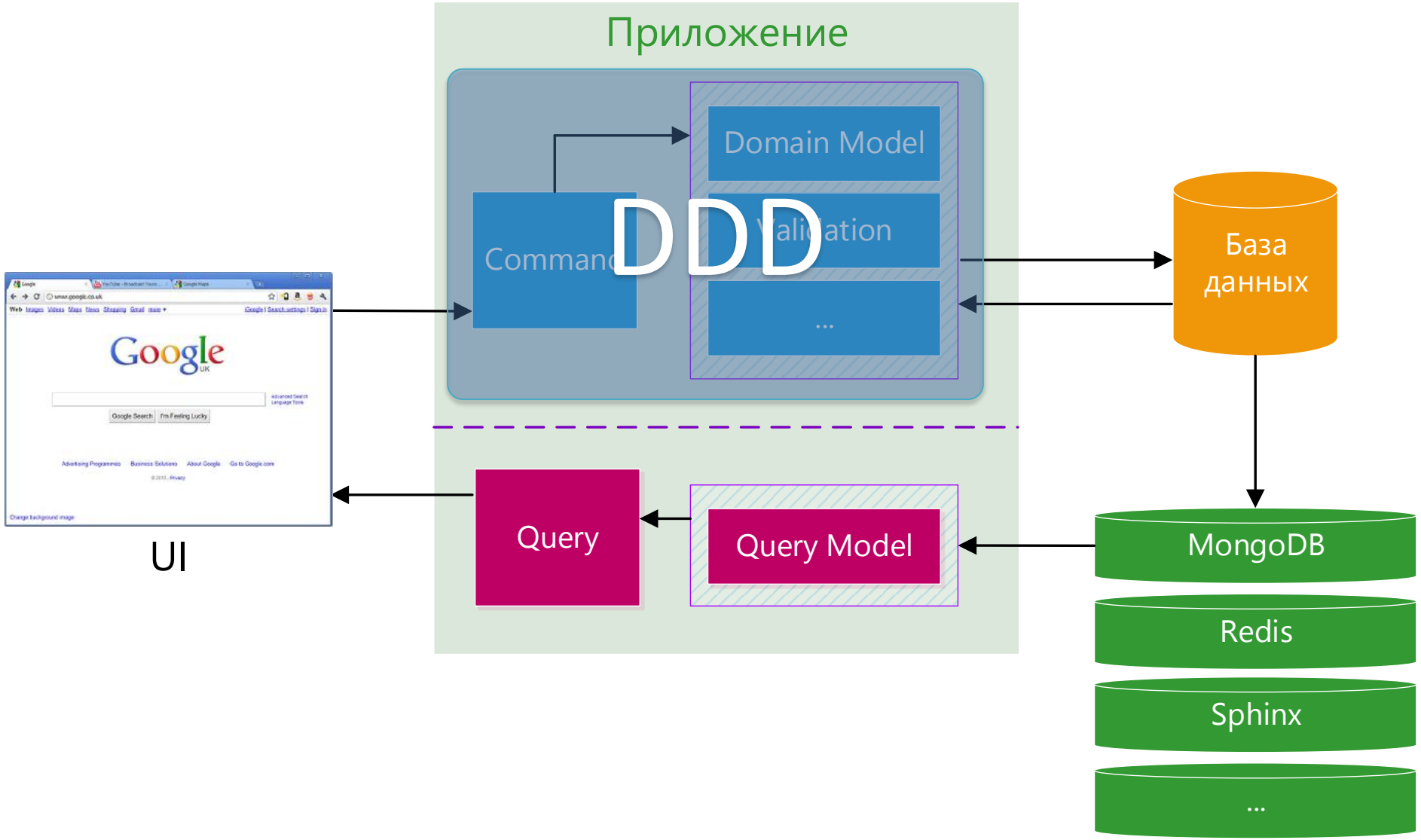
1. Domain-driven design (DDD)
2. N- tier, onion,… architecture
3. ORM (NHibernate, Entity Framework,…)



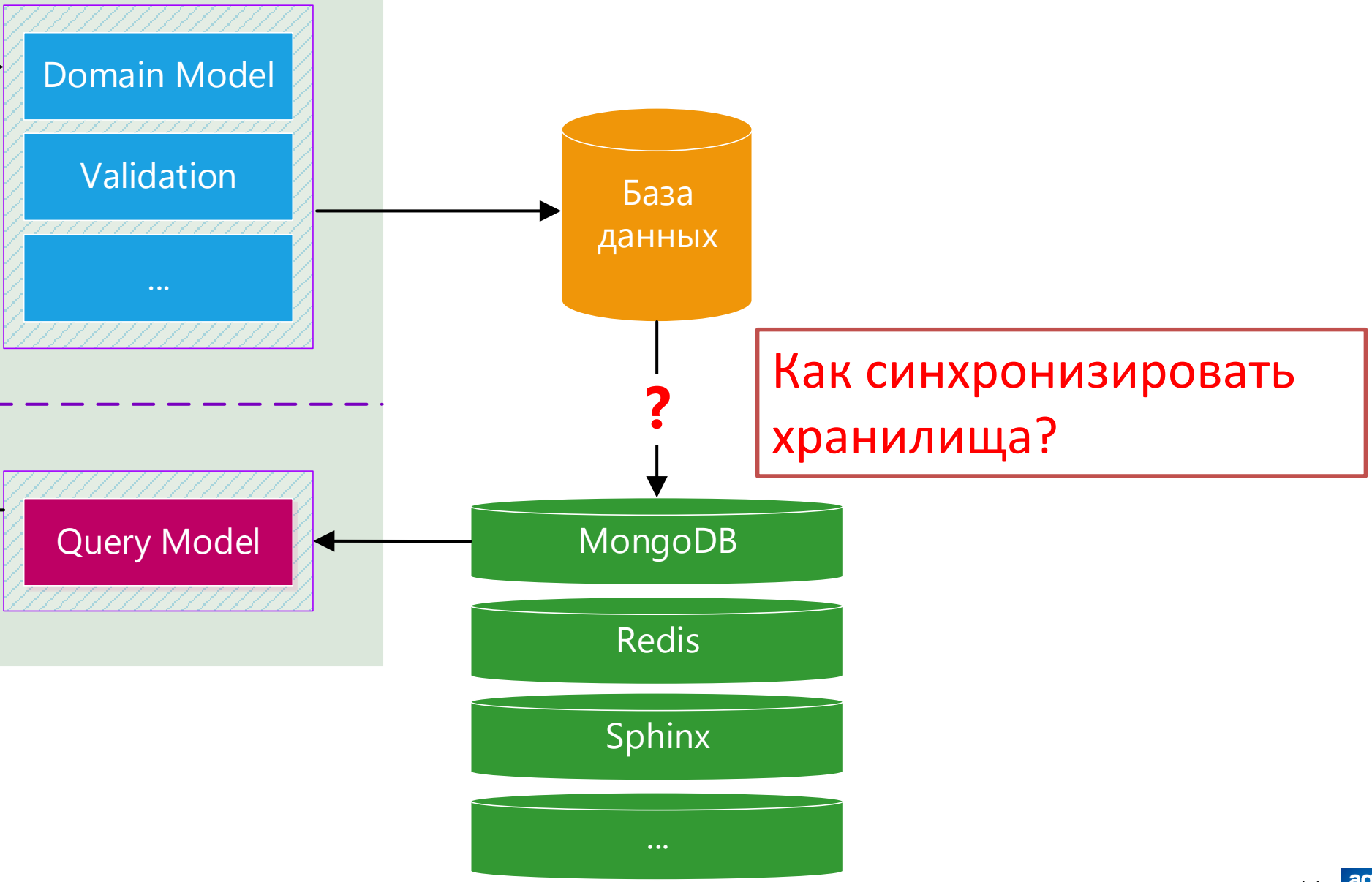
Отделяем чтение от записи



Где живет DDD?

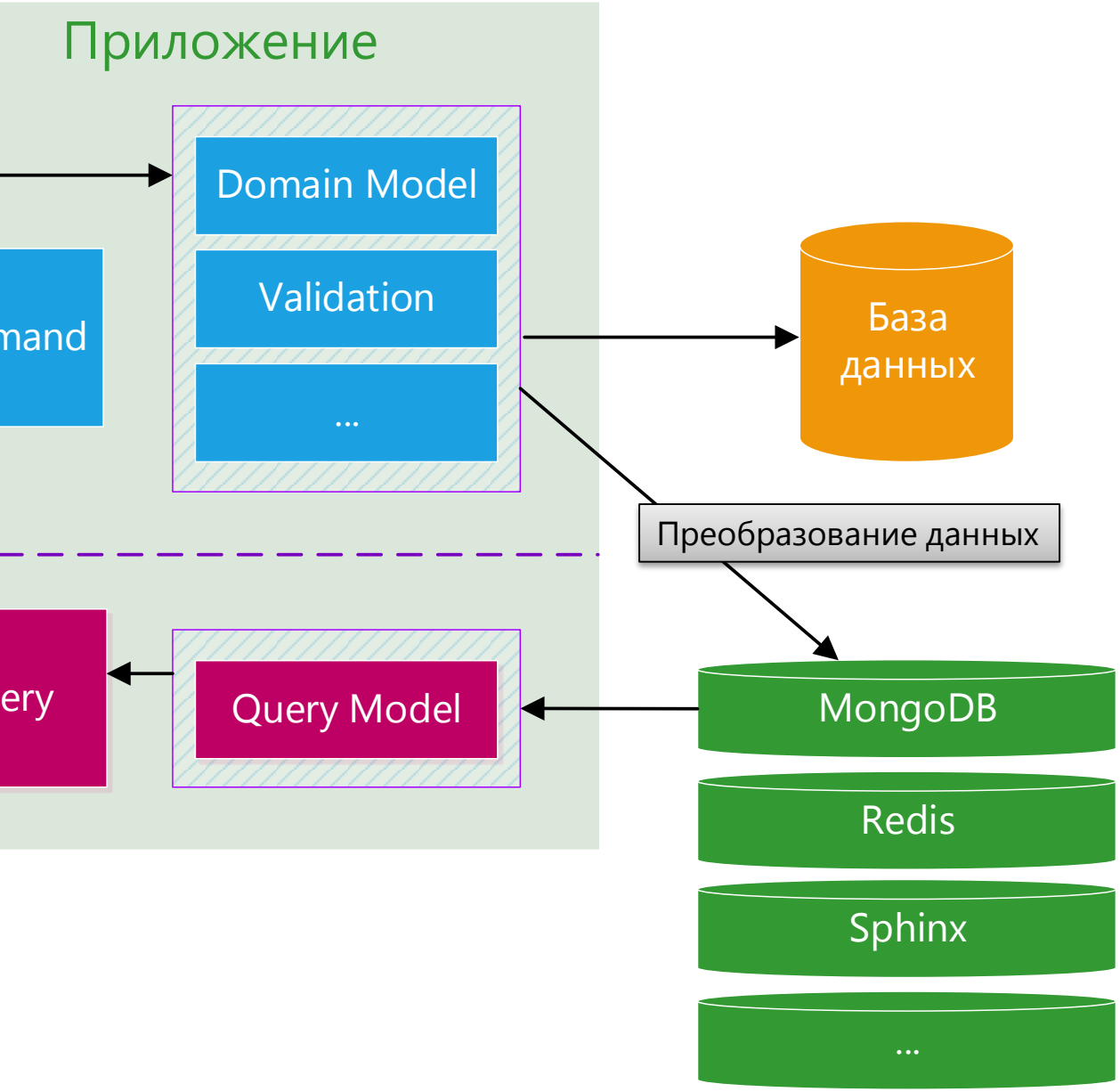


ложение

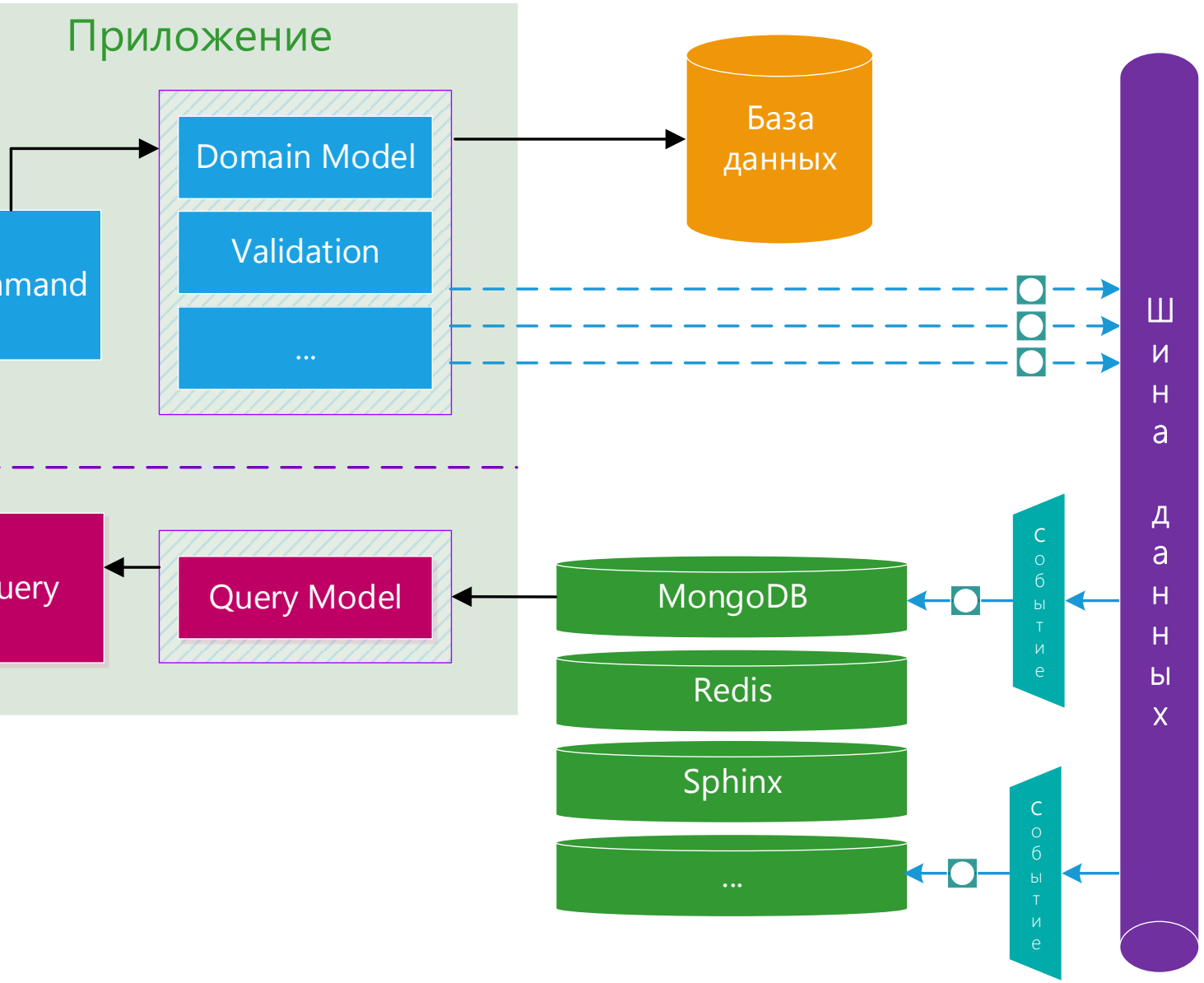


Как синхронизировать хранилища?

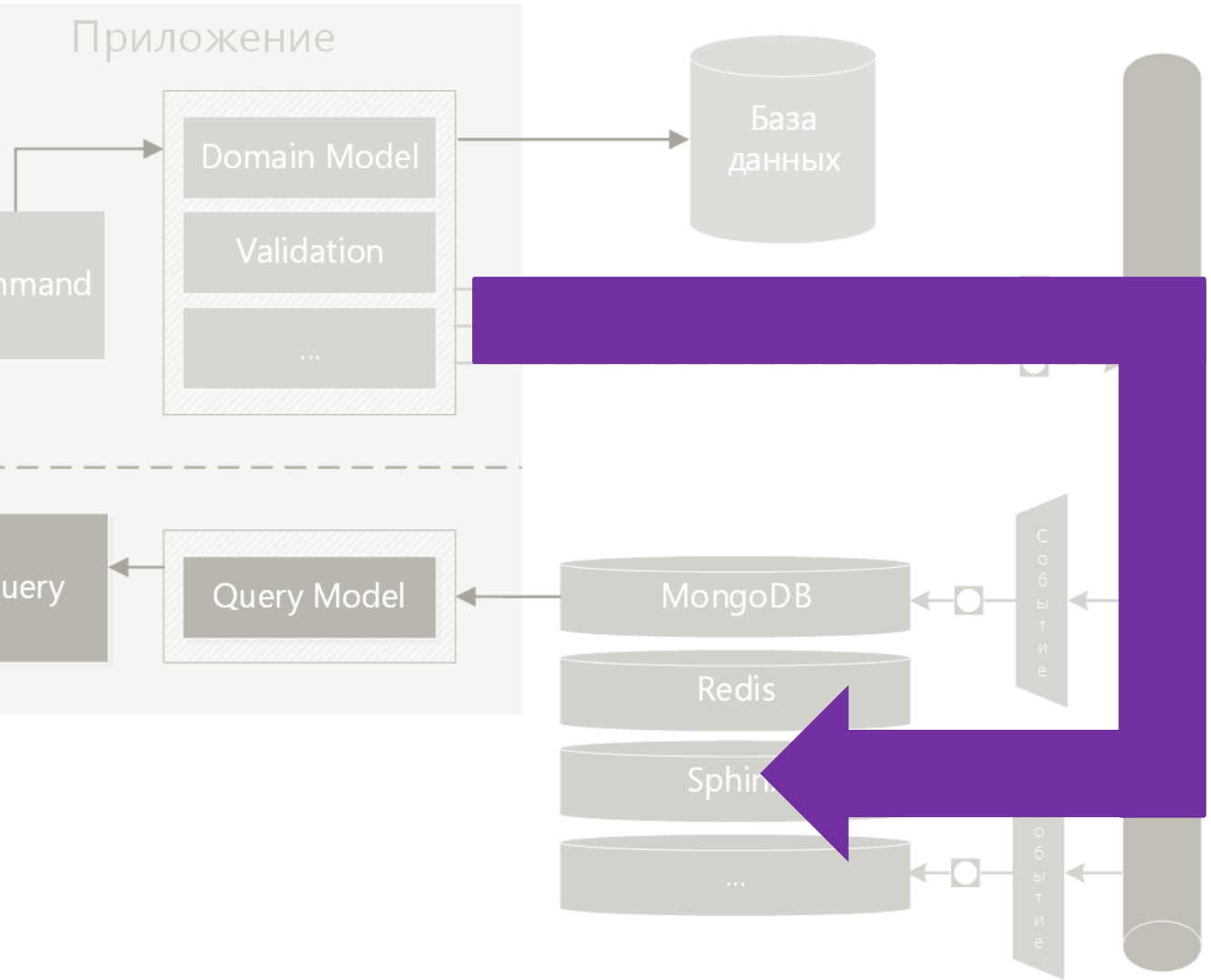
Обновляем синхронно



Обновляем асинхронно



Eventually consistent



Какое время уйдет на синхронизацию?

Вам это не нужно

1. Усложнение архитектуры
2. Обоснование Eventually Consistent

4. Event Sourcing

Event Sourcing

CQRS

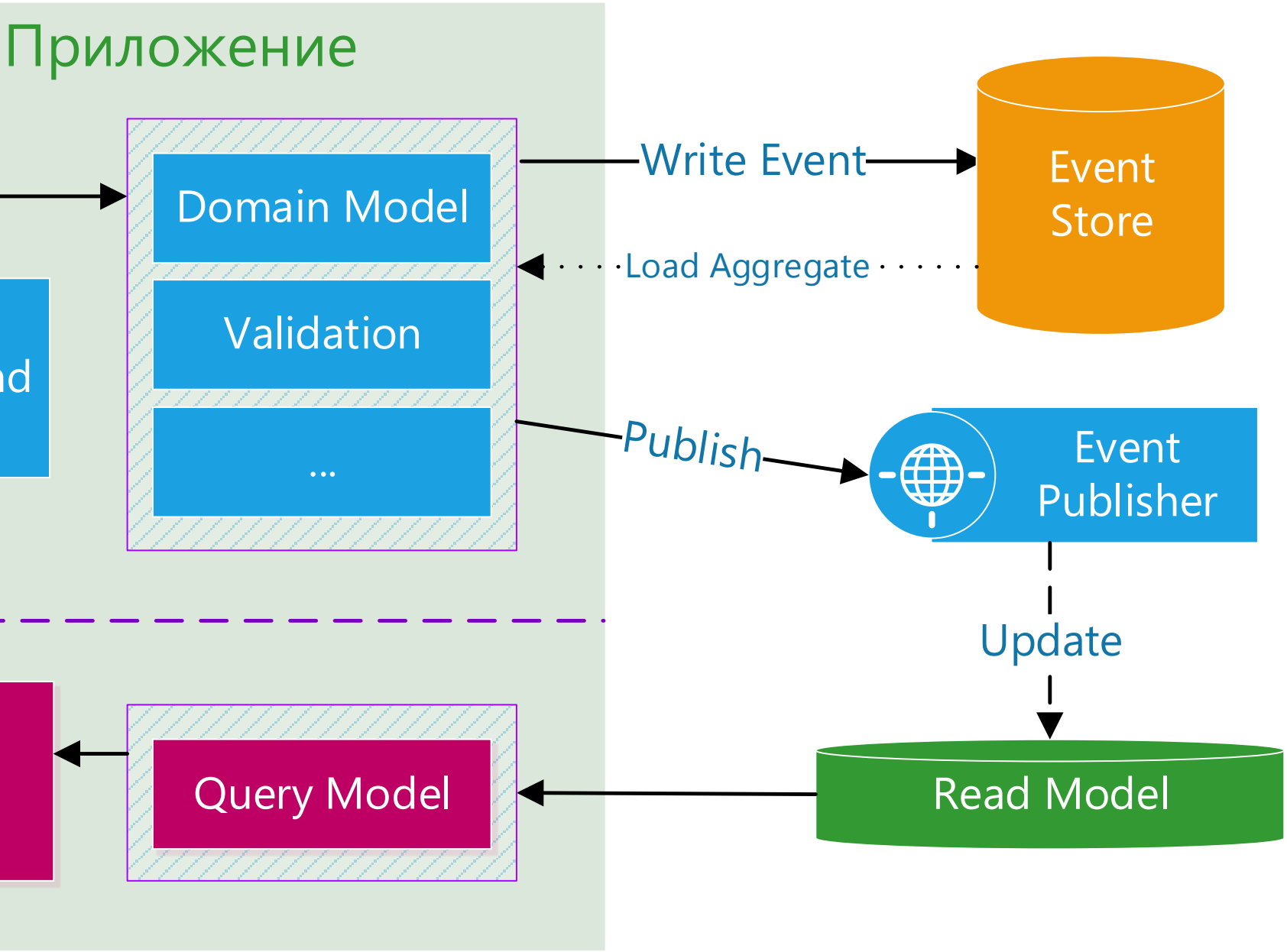
Предпосылки к Event Sourcing

1. Каким было состояние системы 2 недели назад?
2. Имеете ли вы право затереть данные в ячейке новыми?
3. Переходы между состояниями являются частью бизнеса

Сохраняем историю изменений

Event Sourcing

1. Все изменения записывать в виде дельты
2. Текущее состояние домена – это проигрывание «журнала транзакций»
3. Построение проекций для выборок
4. Snapshot как оптимизация


Приложение
Domain Model
Validation
...
Write Event
Load Aggregate
Event Store
Publish
Event Publisher
Update
Query Model
Read Model

Надо ли мне Event Sourcing?

- Есть проблемы, которые не просто решить
  - Как рефакторить агрегаты?
  - Как изменять уже произошедшие события?
  - Как накатываем события, которые зависели от данных стороннего сервиса?
- Заказчики из разных прототипов не выбирали ES
- Бизнесу не надо хранить всю историю

5. Ограничения

1. Нужна подготовка, возможен bus factor
2. Кто-то не любит много классов
3. Дублирование в маленьких классах
4. Сложно целиком придерживаться CQS
5. Не всегда Eventually persisted подходит UX в системе

6. Примеры реализации и подходы

Пример вызова Query

```
public class FindUserById
{
    public int Id { get; set; }
}

public class UserController : Controller
{
    [HttpGet]
    public ActionResult UserDetails(FindUserById context)
    {
        var dto = queryBuilder
            .For<UserForEditDto>()
            .With(context);

        return View(dto);
    }
```

Пример вызова Command

```
public class EditUser
{
    public int UserId { get; set; }
    public string Name { get; set; }
}

public class UserController : Controller
{
    [HttpPost]
    public ActionResult Edit(EditUser context)
    {
        commandHander.Execute(context);

        return this.RedirectToAction(x => x.List());
    }
```

Command Handler

```
public interface ICommandHandler<T> where T : ICommand
{
    void Handle(T command);
}

public class EditUserCommandHandler : ICommandHandler<EditUser>
{
    public void Execute(EditUser context)
    {
            // обновление данных
    }
}
```

IoC-контейнер с абстрактной фабрикой

```
container.AddFacility<TypedFactoryFacility>();

var queries = AllTypes.FromAssemblyNamed("Infrastructure")
    .BasedOn(typeof (IQuery<,>))
    .WithService.AllInterfaces()
    .Configure(x => x.LifeStyle.Transient);

container.Register(
    queries,
    Component.For<IQueryBuilder>()
             .AsFactory().LifeStyle.Transient,
    Component.For(typeof (IQueryFor<>))
             .ImplementedBy(typeof (QueryFor<>)));
```

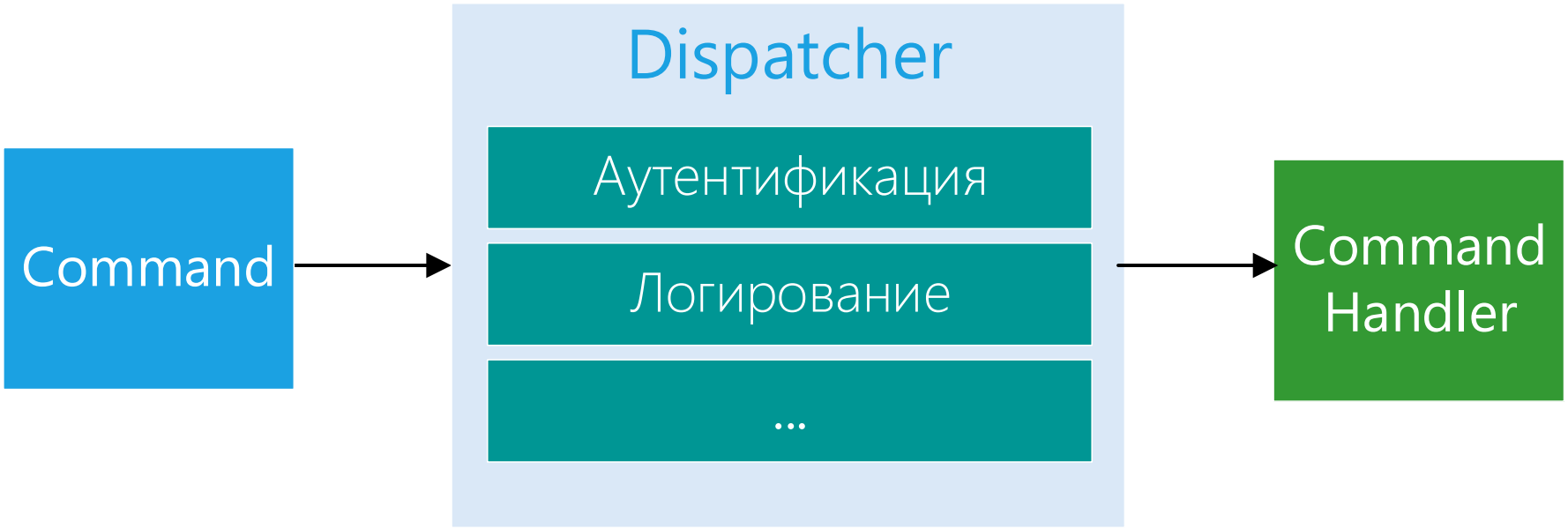
Диспетчеризация через шину

```
public class HomeController : Controller
{
    [HttpPost]
    public ActionResult ChangeName(Guid id, string name, int version)
    {
        var command = new RenameInventoryItem(id, name, version);
        bus.Send(command);

        return RedirectToAction("Index");
    }
```

Диспетчеризация через шину

```
public class FakeBus : ICommandSender, IEventPublisher
{
    public void Send<T>(T command) where T : Command
    {
        List<Action<Message>> handlers;
        if (_routes.TryGetValue(typeof(T), out handlers))
        {
            if (handlers.Count != 1)
                throw new InvalidOperationException();
            handlers[0](command);
        }
        else
        {
            throw new InvalidOperationException("no handler registered");
        }
    }
```

Промежуточный Dispatcher



Готовая инфраструктура CQRS на примере .NET приложений

1. https://github.com/gnschenker/cqrs-introduction
2. https://github.com/gregoryyoung/m-r
3. http://lokad.github.io/lokad-cqrs/
4. https://github.com/AlexanderByndyu/ByndyuSoft.Infrastructure/tree/master/samples/aspnetmvc
5. http://msdn.microsoft.com/en-us/library/jj554200
6. https://github.com/ncqrs/ncqrs

Полезные ссылки

1. http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
2. http://www.udidahan.com/2009/12/09/clarified-cqrs
3. http://martinfowler.com/bliki/CQRS.html
4. http://martinfowler.com/bliki/CommandQuerySeparation.html
5. http://abdullin.com/cqrs
6. http://richarddingwall.name/2010/06/15/brownfield-cqrs-part-1-commands
7. http://msdn.microsoft.com/en-us/library/dn568103.aspx

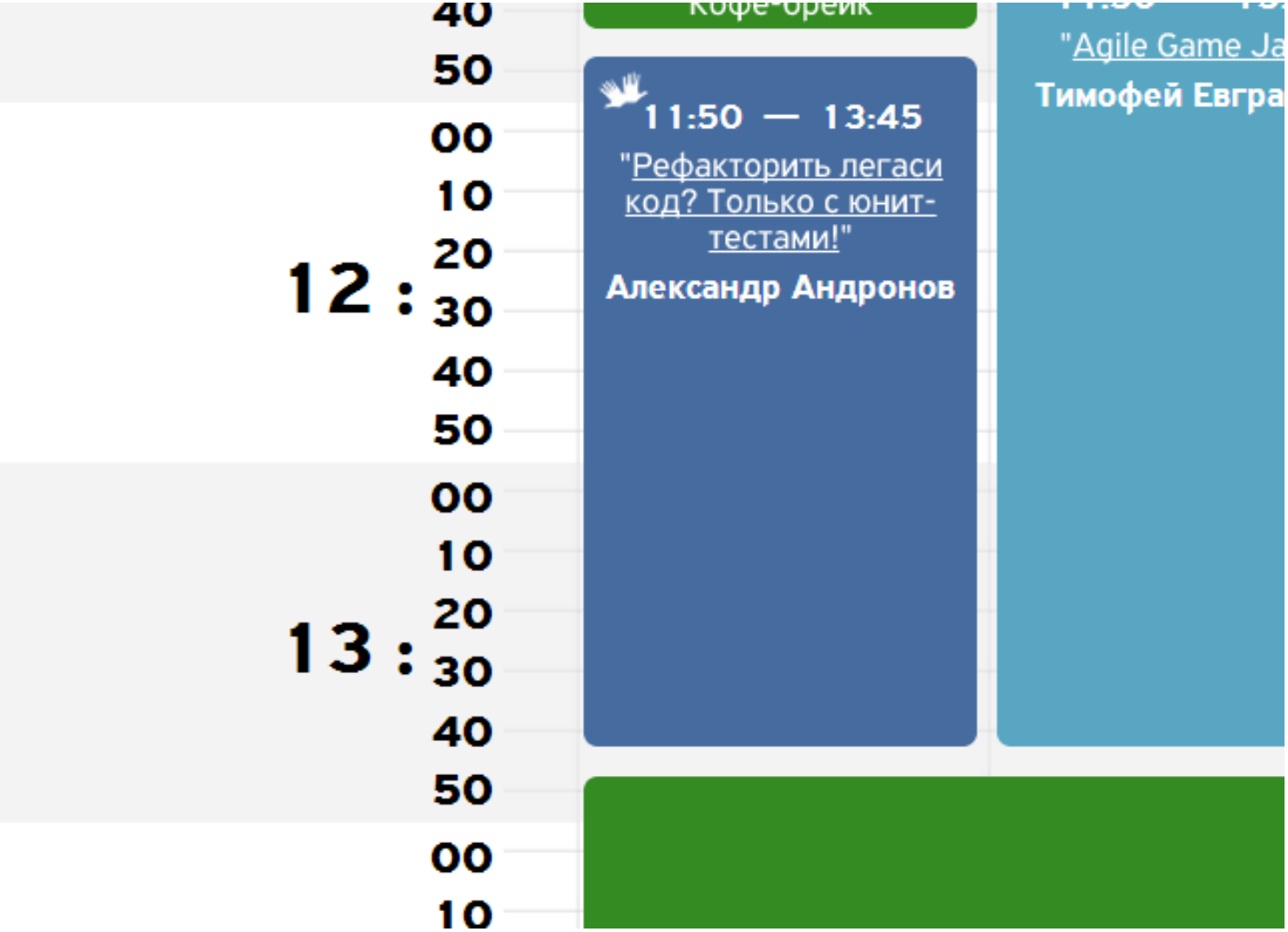
Спасибо за внимание!

Буду рад ответить на ваши вопросы лично или через:

 blog.byndyu.ru

 alexanderbyndyu

 alexander.byndyu@gmail.com


40
50
00
10
12 : 20
30
40
50
00
10
13 : 20
30
40
50
00
10
Кофе-брейк
11:50 — 13:45
"Рефакторить легаси код? Только с юнит-тестами!"
Александр Андронов
"Agile Game Ja
Тимофей Евгра

Ниже углубление

- Не факт, что войдет в выступление

CQRS + DDD

- Выбрать Aggregation Root
- Какие зависимости вытянуть?
- Какие-то куски нужно сложить перемножить с другими корнями
- Уходим от UoW в сторону оптимизированных запросов

CQRS и облака

- http://abdullin.com/xlim/

Слишком много обвязки

- Как вариант Automapper и конвенции

UoW и CQ

- Чем выше открыта бизнес-транзакция тем лучше
- Вариант делать это внутри каждой CQ

Bounded Contexts

- While CQRS touches on many pieces of software architecture, it is still not at the top of the food chain. CQRS if used is employed within a bounded context (DDD) or a business component (SOA) – a cohesive piece of the problem domain. The events published by one BC are subscribed to by other BCs, each updating their query and command data stores as needed.
- UI's from the CQRS found in each BC can be "mashed up" in a single application, providing users a single composite view on all parts of the problem domain. Composite UI frameworks are very useful for these cases.

- Рисунок, много подсистем внутки каждой из которых CQRS

- Also, CQRS should not be your top-level architectural pattern – that would be SOA, микросервисы
  CQRS, if used at all, would be used inside a service boundary only.

Пример теста

Пример теста на Query

Одна модель для C и Q?

- Это контексты выполнения, а не просто DTO
- Тоже самое относится и к parameter object

Пример структуры солюшена

Объединяем запрос и команду

- Команда, которая вернет ID
- Пример со стеком от Фаулера

Запрос на чтение сделает +1 к просмотрам блога

С orm и без orm

- Простой переход
- Часть уходит в без орм, часть остается

Пример CommandHanlder

- Что делать с ошибками?
- Что делать при асинхронной обработке?

Query реализация