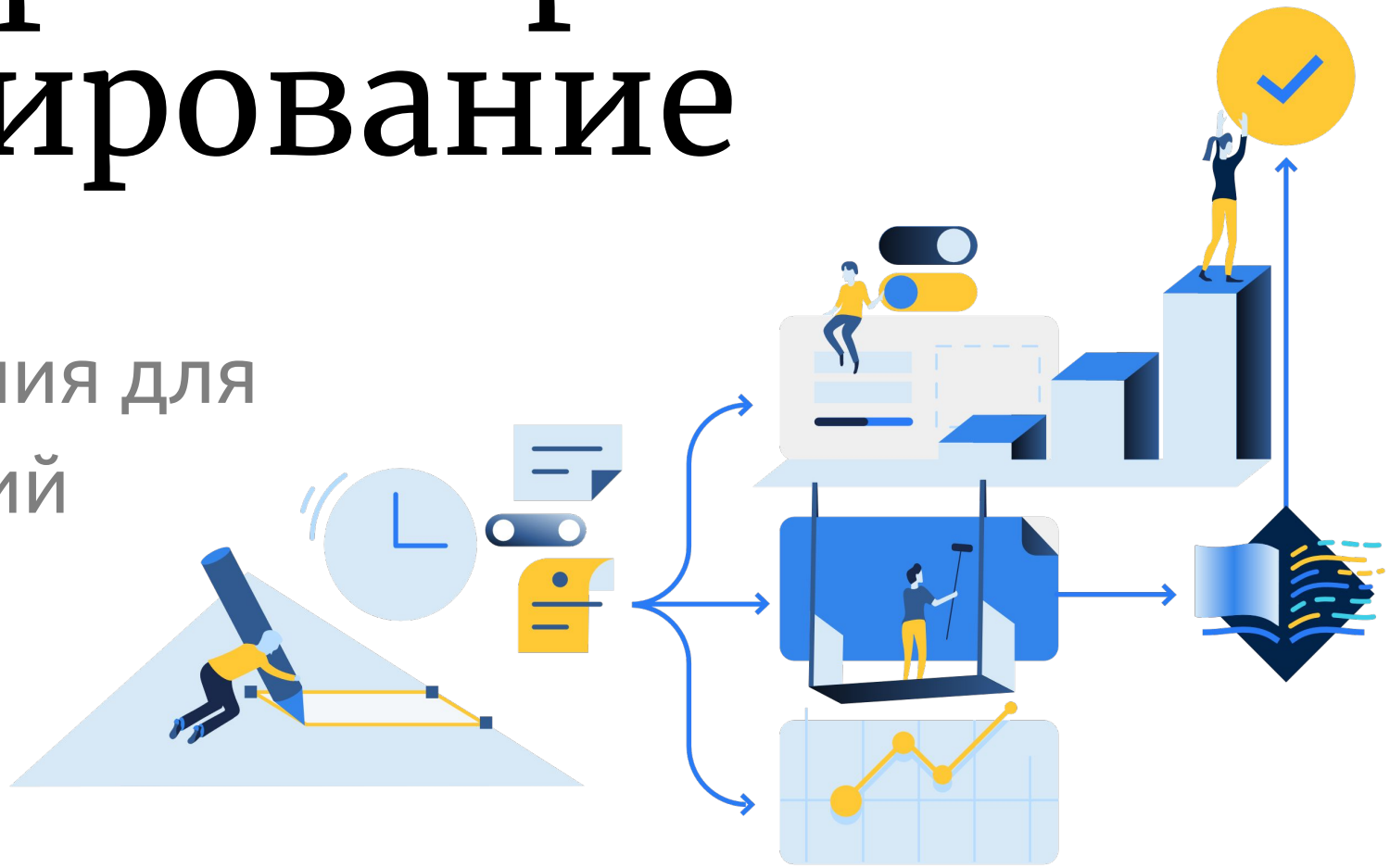
Быстрорастворимое проектирование

Типовые решения для web-приложений



Максим Аршинов

предприниматель, блогер, преподаватель

max@hightech.group
https://habr.com/users/marshinov

О чем этот доклад

О чем этот доклад

- Критерии

О чем этот доклад

- Критерии
- Краткая история развития архитектурной мысли

О чем этот доклад

- Критерии
- Краткая история развития архитектурной мысли
- Обзор недостатков классической слоеной архитектуры

О чем этот доклад

- Критерии
- Краткая история развития архитектурной мысли
- Обзор недостатков классической слоеной архитектуры
- Решение

О чем этот доклад

- Критерии
- Краткая история развития архитектурной мысли
- Обзор недостатков классической слоеной архитектуры
- Решение
- Пошаговый разбор реализации без погружения в детали

О чем этот доклад

- Критерии
- Краткая история развития архитектурной мысли
- Обзор недостатков классической слоеной архитектуры
- Решение
- Пошаговый разбор реализации без погружения в детали
- Все вместе

Критерии

- Количество фич в единицу времени

Критерии

- Количество фич в единицу времени
- Количество возвратов на этапе код-ревью

Критерии

- Количество фич в единицу времени
- Количество возвратов на этапе код-ревью
- Количество возвратов на этапах тестирования и сдачи-приемки

Критерии

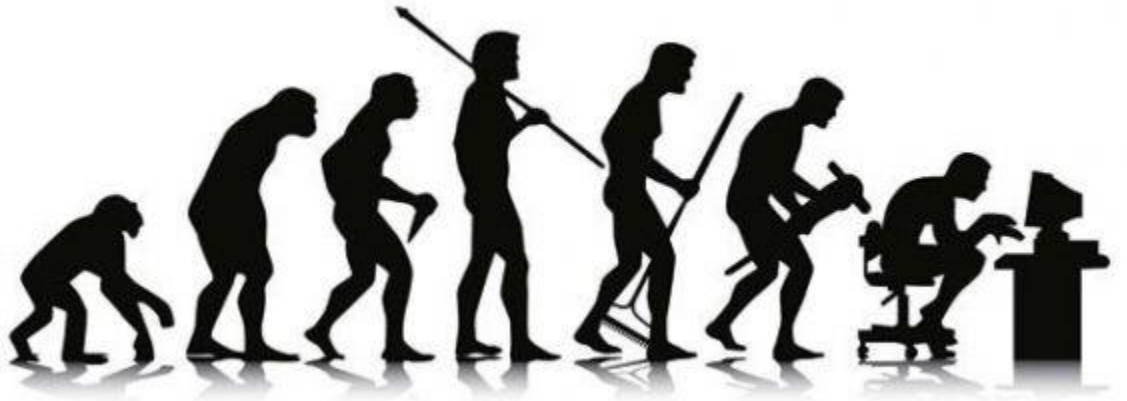
- Количество фич в единицу времени
- Количество возвратов на этапе код-ревью
- Количество возвратов на этапах тестирования и сдачи-приемки
- Регрессия и количество багов, дошедших до продакшна

Критерии

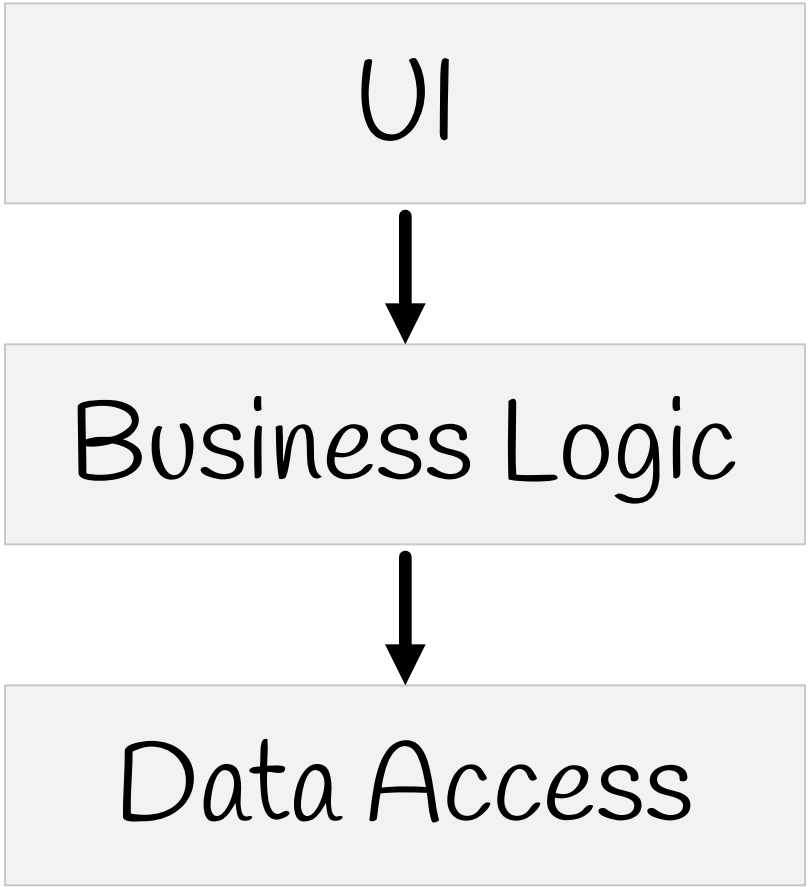
- Количество фич в единицу времени
- Количество возвратов на этапе код-ревью
- Количество возвратов на этапах тестирования и сдачи-приемки
- Регрессия и количество багов, дошедших до продакшна

Последний критерий сложно измерить

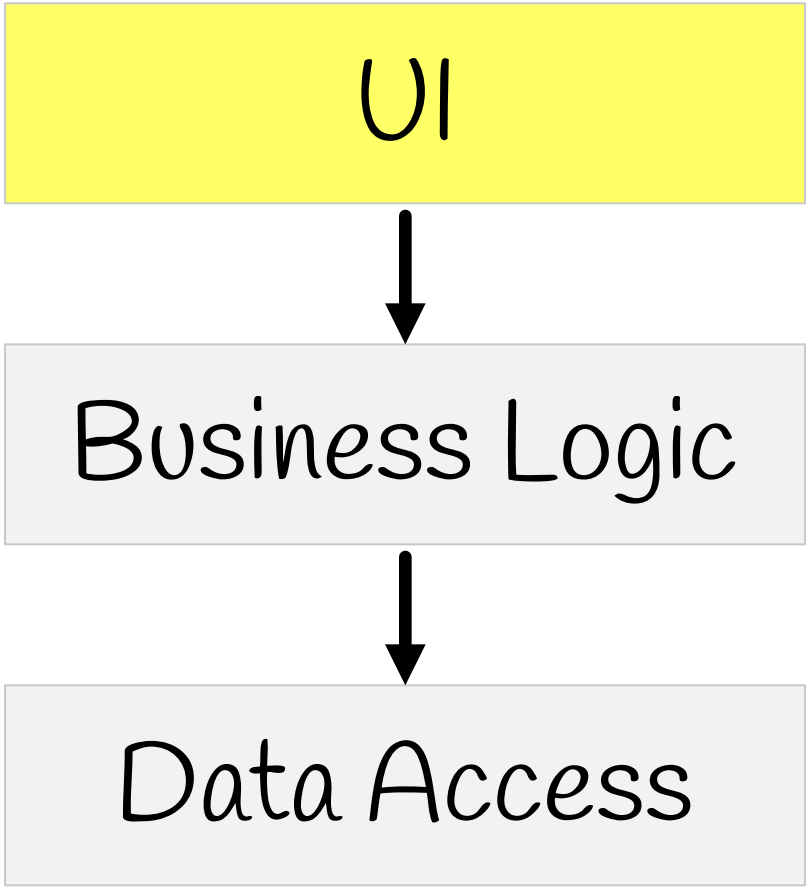
Развитие архитектуры



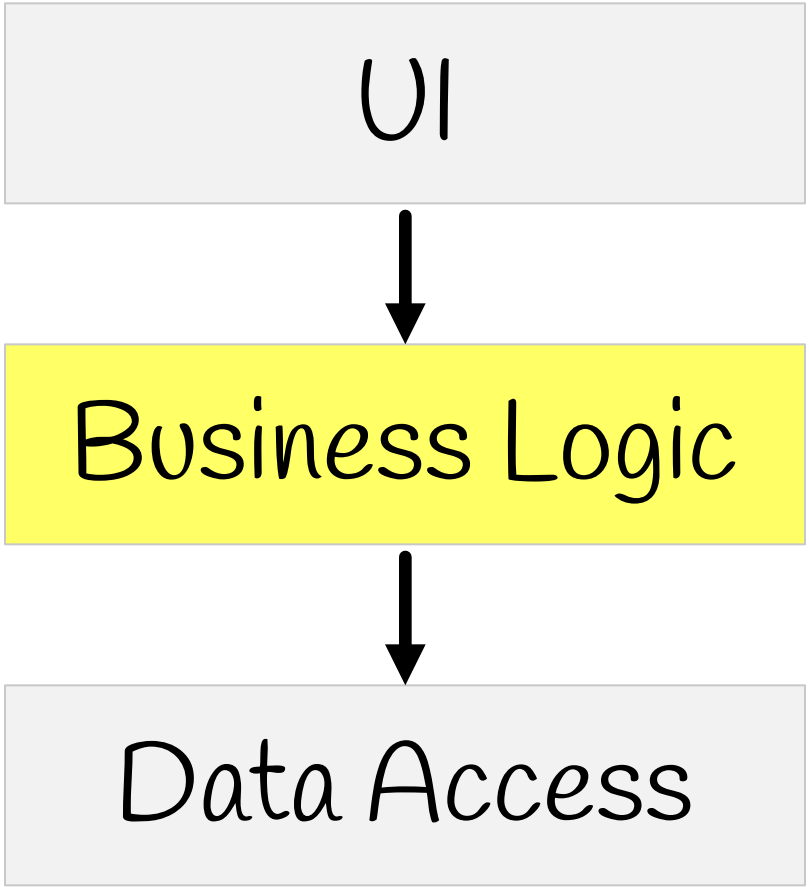
Традиционная слоеная архитектура



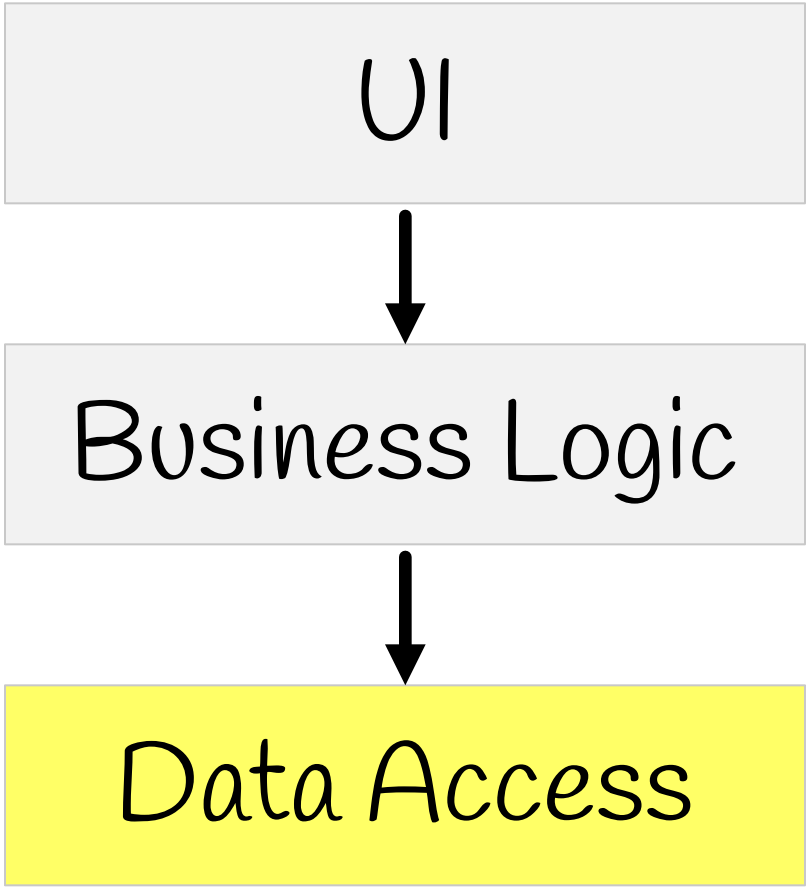
Традиционная слоеная архитектура

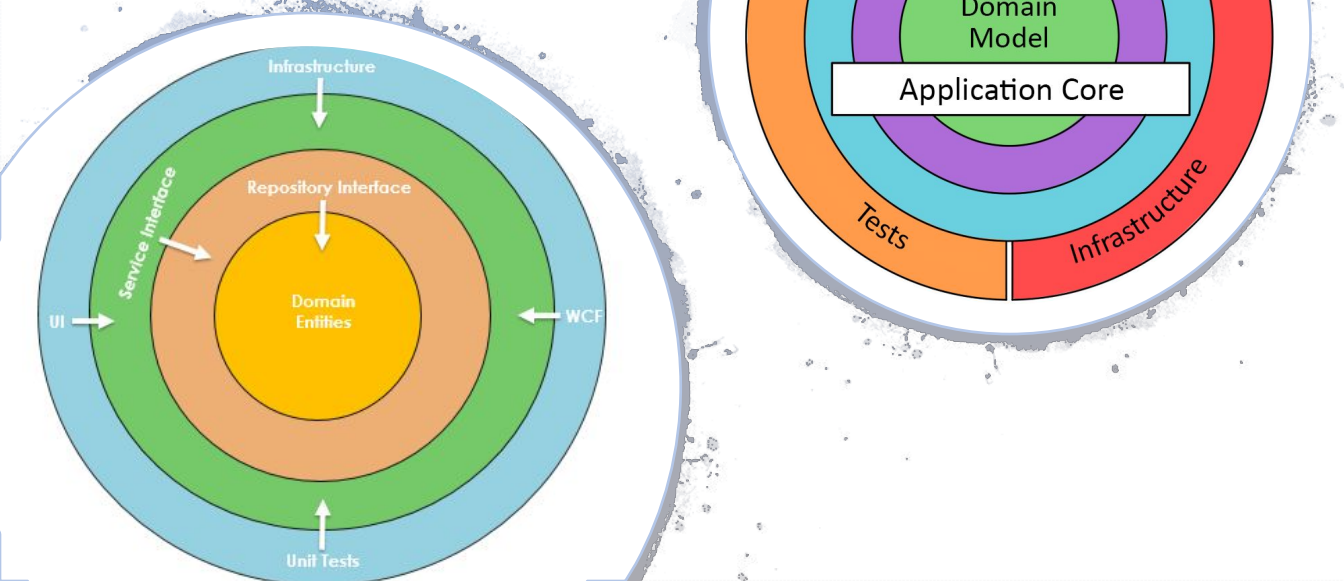


Традиционная слоеная архитектура



Традиционная слоеная архитектура

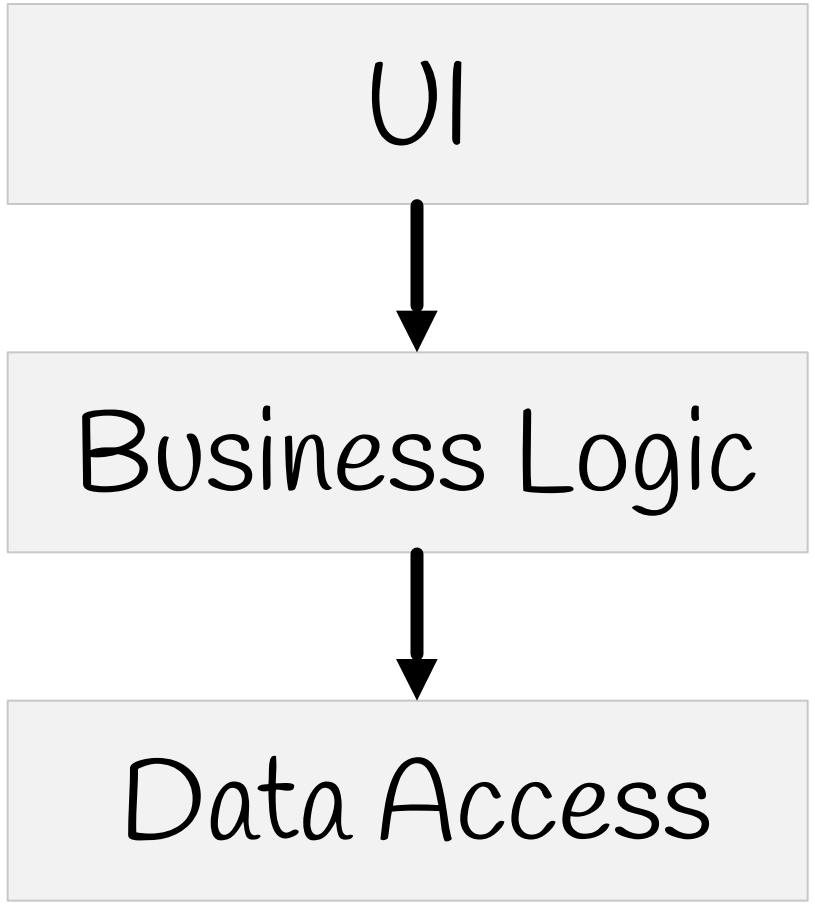




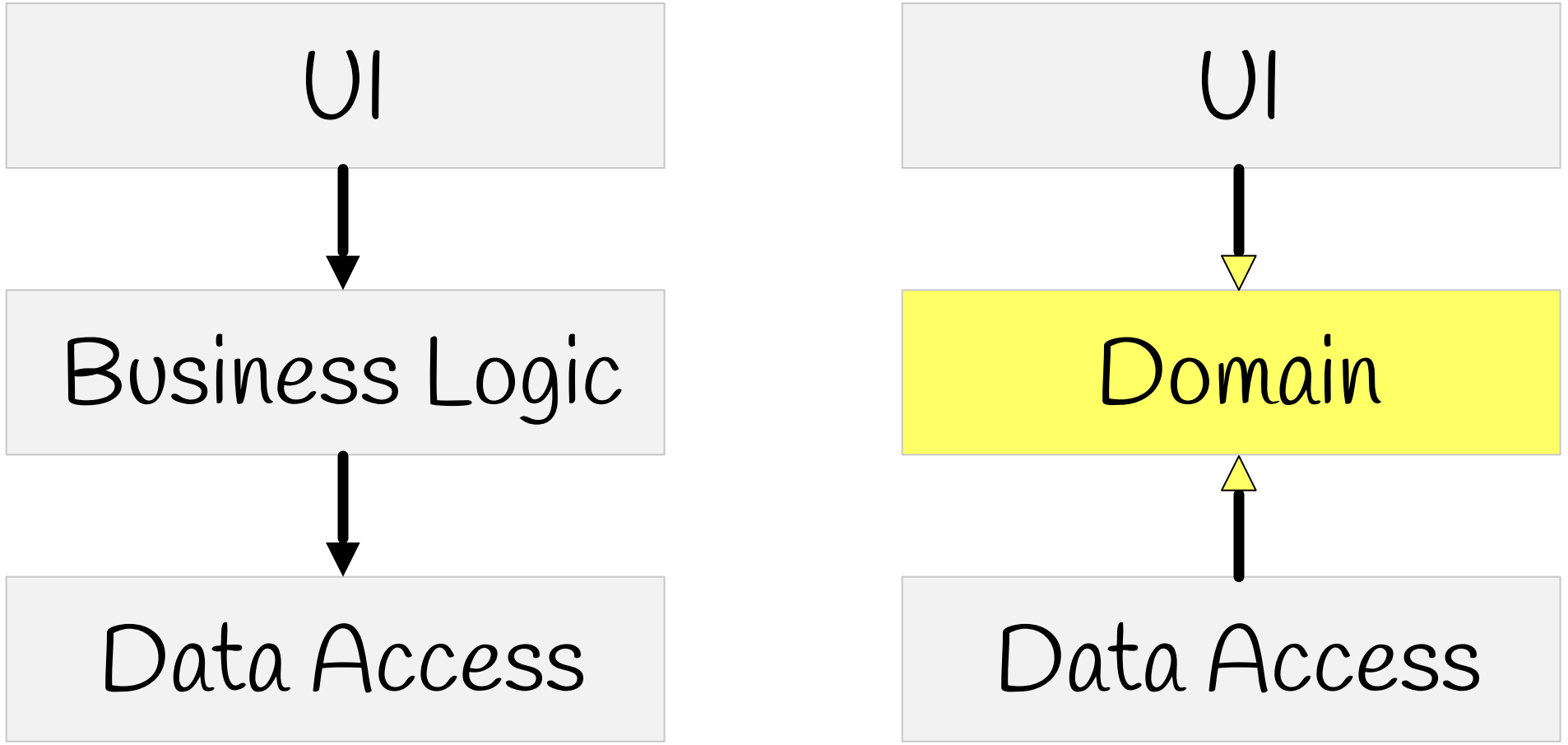
Луковая

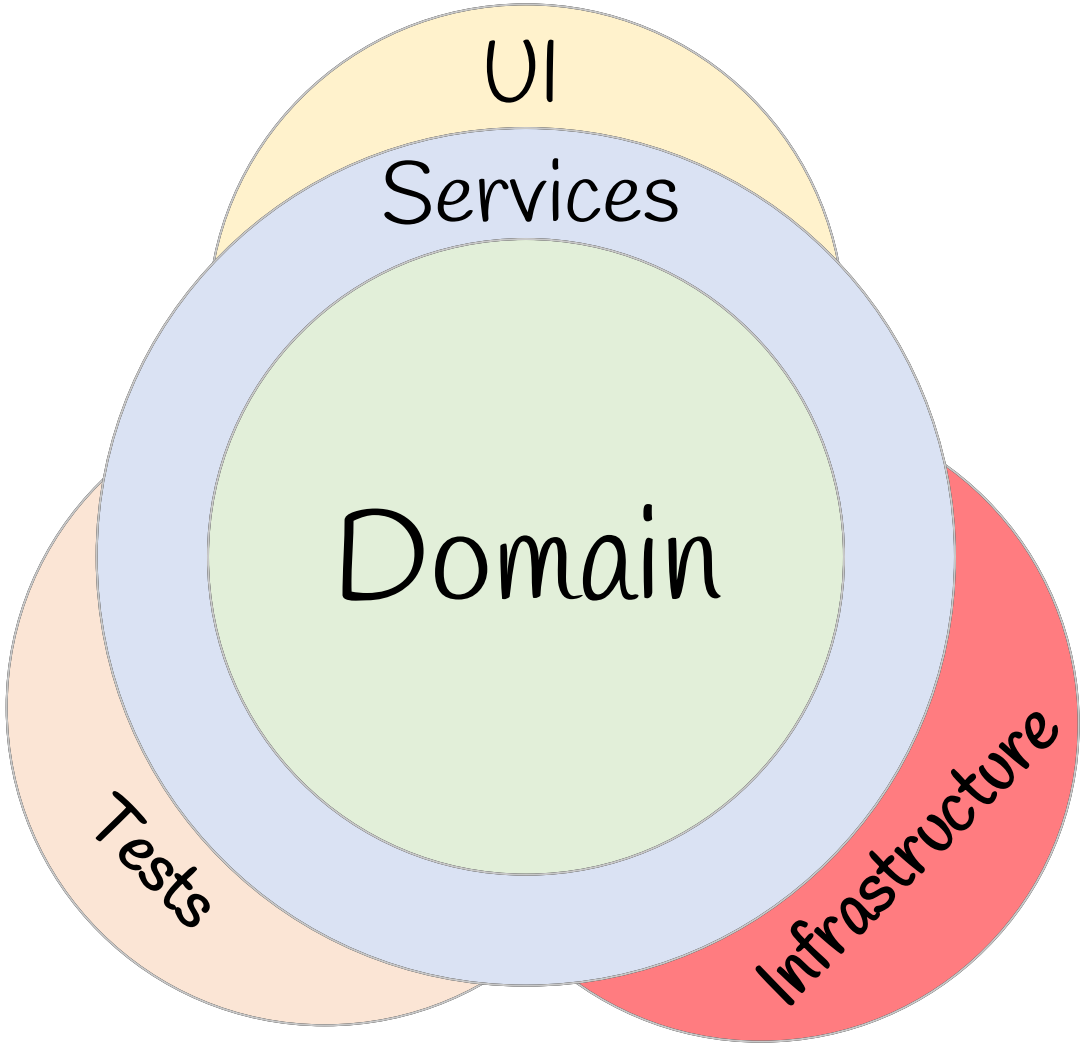
Инверсия зависимостей

Слоеная VS Луковая



Слоеная VS Луковая

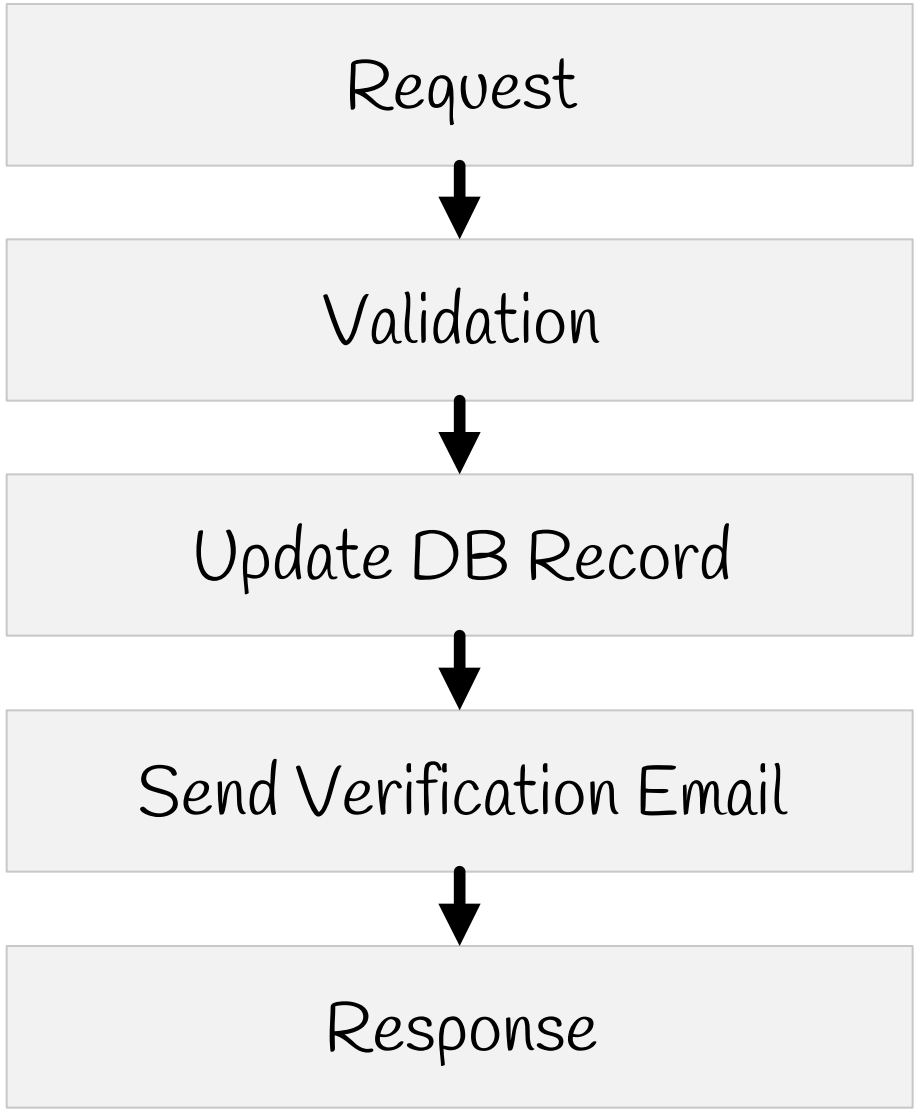


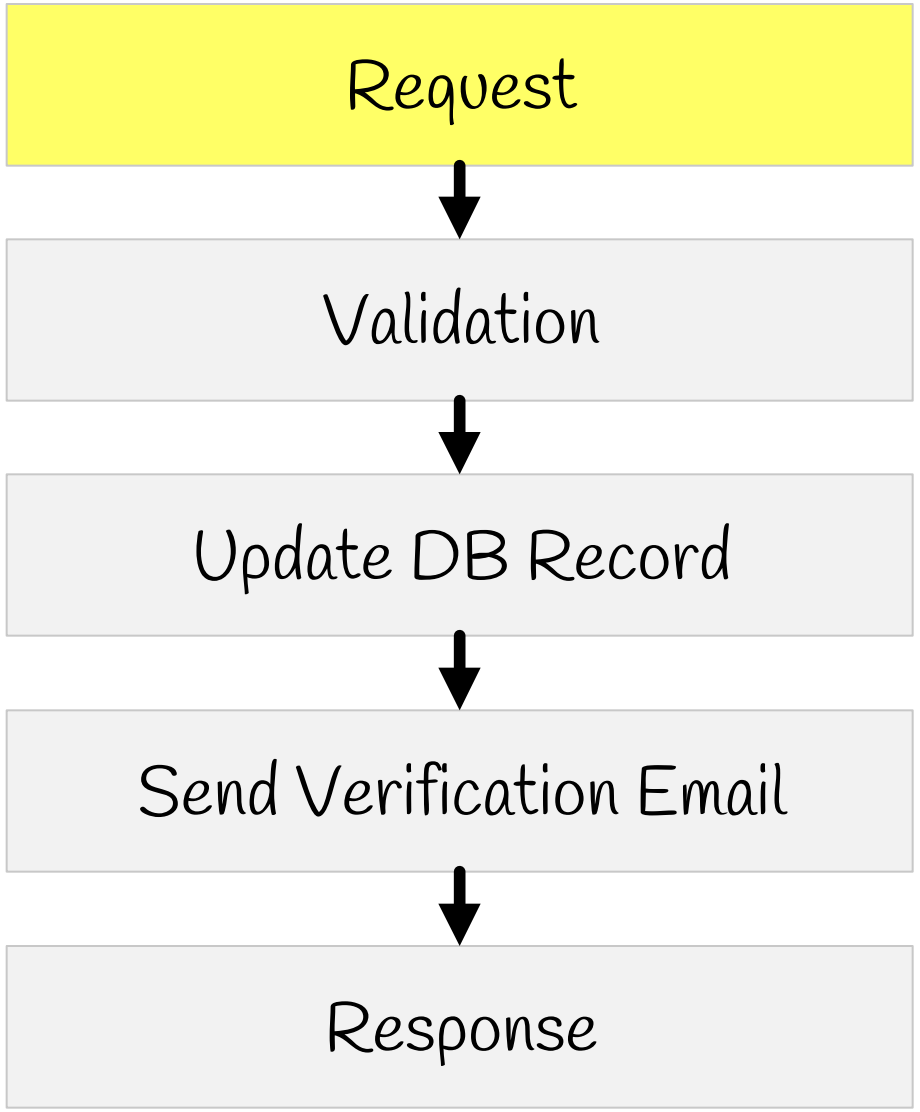


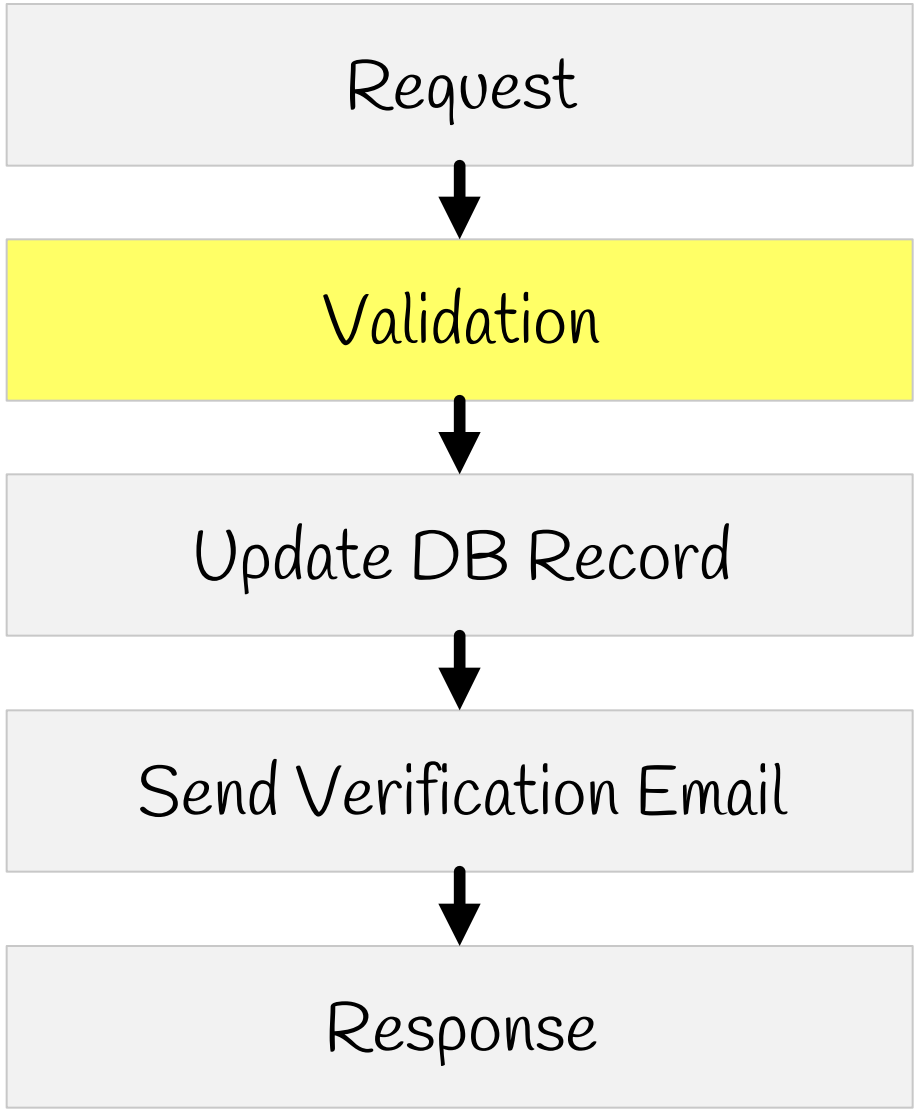
- Луковая
- Гексагональная
- Порты и адаптеры

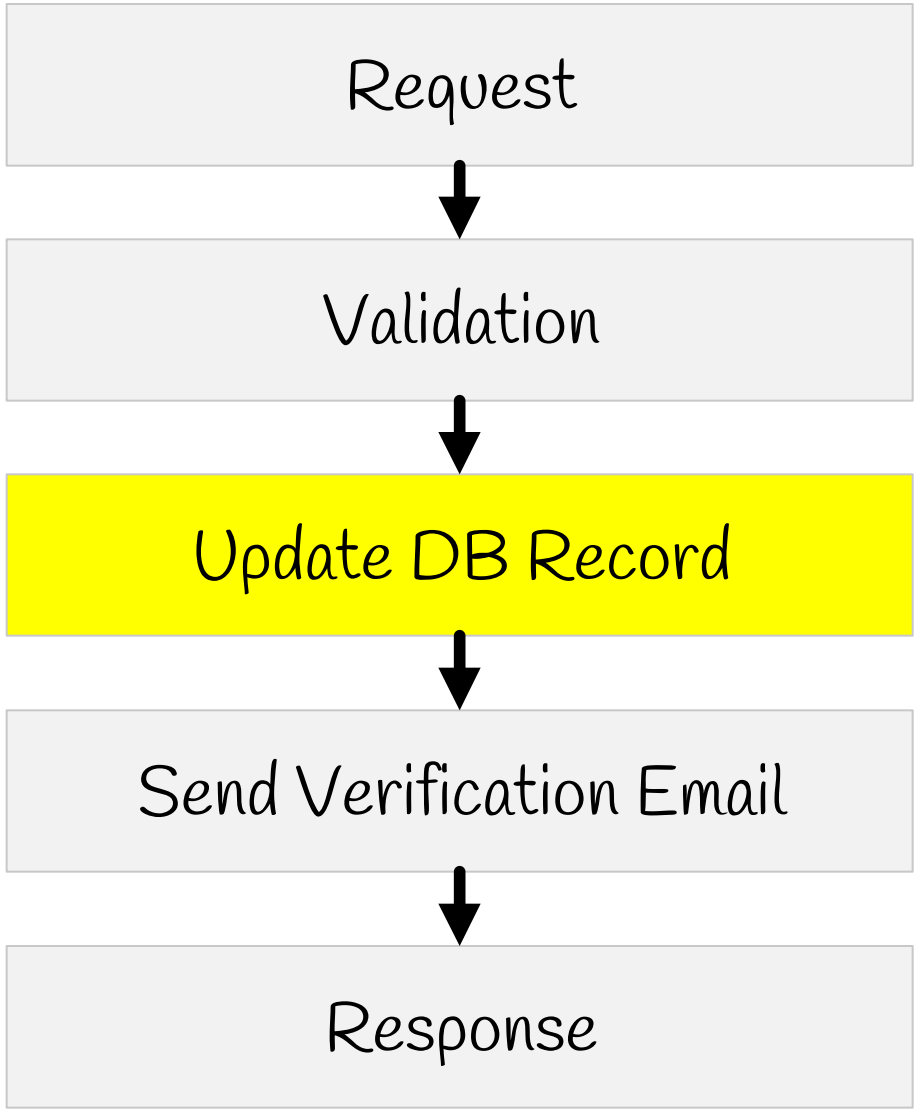
Простой пример

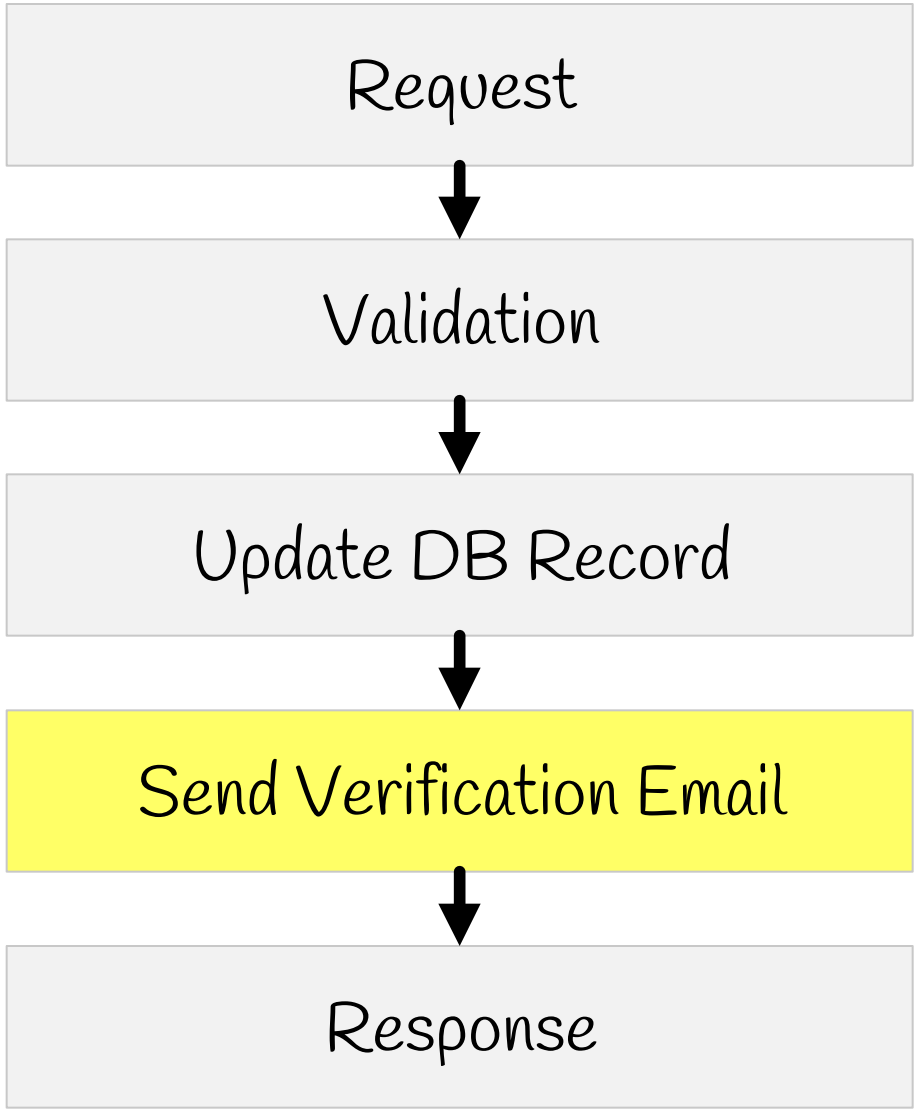
Обновление email

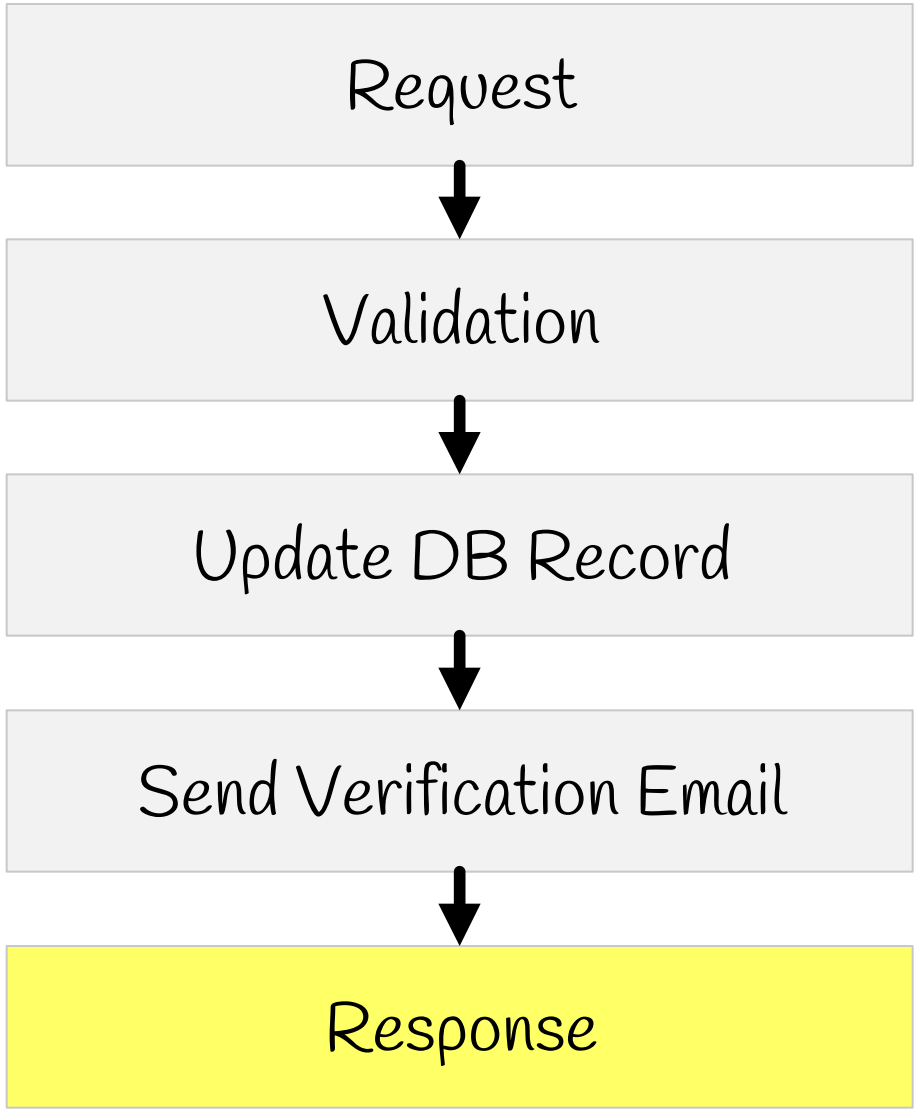

Request
Validation
Update DB Record
Send Verification Email
Response


Request
Validation
Update DB Record
Send Verification Email
Response


Request
Validation
Update DB Record
Send Verification Email
Response


Request
Validation
Update DB Record
Send Verification Email
Response


Request
Validation
Update DB Record
Send Verification Email
Response


Request
Validation
Update DB Record
Send Verification Email
Response

```
public IActionResult UpdateEmail(int id, string email)
{
    if (!ModelState.IsValid) return BadRequest();
    var previousEmail = user.Email;

    var user = _dbContext.Set<User>()
        .FirstOrDefault(x => x.Id == id);
    user.Email = email;
    _dbContext.SaveChanges();

    _emailSender.SendEmail(email,
        "Ваш email изменен", $"с {previousEmail} на {email}");
    return Ok();
}
```

```
public IActionResult UpdateEmail(int id, string email)
{
    if (!ModelState.IsValid) return BadRequest();
    var previousEmail = user.Email;

    var user = _dbContext.Set<User>()
        .FirstOrDefault(x => x.Id == id);
    user.Email = email;
    _dbContext.SaveChanges();

    _emailSender.SendEmail(email,
        "Ваш email изменен", $"с {previousEmail} на {email}");
    return Ok();
}
```

Validate

```
public IActionResult UpdateEmail(int id, string email)
{
    if (!ModelState.IsValid) return BadRequest();
    var previousEmail = user.Email;

    var user = _dbContext.Set<User>()
        .FirstOrDefault(x => x.Id == id);
    user.Email = email;
    _dbContext.SaveChanges();

    _emailSender.SendEmail(email,
        "Ваш email изменен", $"с {previousEmail} на {email}");
    return Ok();
}
```

Update DB record

```
public IActionResult UpdateEmail(int id, string email)
{
    if (!ModelState.IsValid) return BadRequest();
    var previousEmail = user.Email;

    var user = _dbContext.Set<User>()
        .FirstOrDefault(x => x.Id == id);
    user.Email = email;
    _dbContext.SaveChanges();

    _emailSender.SendEmail(email,
        "Ваш email изменен", $"с {previousEmail} на {email}");
    return Ok();
}
```

Send verification email

```
public IActionResult UpdateEmail(int id, string email)
{
    if (!ModelState.IsValid) return BadRequest();
    var previousEmail = user.Email;

    var user = _dbContext.Set<User>()
        .FirstOrDefault(x => x.Id == id);
    user.Email = email;
    _dbContext.SaveChanges();

    _emailSender.SendEmail(email,
        "Ваш email изменен", $"с {previousEmail} на {email}");
    return Ok();
}
```

Return response

```
[Authorize]
public IActionResult UpdateEmailRealLife(int id, [EmailAddress, Required] string email)
{
    try
    {
        string previousEmail = null;
        using (MiniProfiler.Current.Step("Update email"))
        {
            if (!ModelState.IsValid) return BadRequest();
            email = email.ToLower();
            var user = _dbContext.Set<User>().FirstOrDefault(x => x.Id == id);
            if (user == null)
            {
                return NotFound();
            }

            if (user.Id != id)
            {
                _logger.LogWarning($"Попытка изменить чужие данные {user.Id}/{id}");
                return Forbid();
            }

            previousEmail = user.Email;
            user.Email = email;
            user.LastUpdated = DateTime.UtcNow;
            _dbContext.SaveChanges();
        }

        _emailSender.SendEmail(email, "Ваш email изменен", $"с {previousEmail} на {email}");
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        return new ObjectResult(new
        {
            message = "Произошла ошибка при обновлении данных в БД"
        })
        {
            StatusCode = 500
        };
    }
    catch (SmtpException e)
    {
        _logger.LogError(e.Message);
    }

    return Ok();
}
```

Суровая реальность

```
[Authorize]
public IActionResult UpdateEmailRealLife(int id, [EmailAddress, Required] string email)
{
    try
    {
        string previousEmail = null;
        using (MiniProfiler.Current.Step("Update email"))
        {
            if (!ModelState.IsValid) return BadRequest();
            email = email.ToLower();
            var user = _dbContext.Set<User>().FirstOrDefault(x => x.Id == id);
            if (user == null)
            {
                return NotFound();
            }

            if (user.Id != id)
            {
                _logger.LogWarning($"Попытка изменить чужие данные {user.Id}/{id}");
                return Forbid();
            }

            previousEmail = user.Email;
            user.Email = email;
            user.LastUpdated = DateTime.UtcNow;
            _dbContext.SaveChanges();
        }

        _emailSender.SendEmail(email, "Ваш email изменен", $"с {previousEmail} на {email}");
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        return new ObjectResult(new
        {
            message = "Произошла ошибка при обновлении данных в БД"
        })
        {
            StatusCode = 500
        };
    }
    catch (SmtpException e)
    {
        _logger.LogError(e.Message);
    }

    return Ok();
}
```

- Авторизация
- Обработка ошибок
- Форматирование
- Профилирование
- Аудит-лог
- Логирование

Точно, мы же забыли сервисы и репозитории!

Сейчас добавим и все встанет на свои места

```
public IActionResult UpdateEmailWithService(int id, string email)
{
    try
    {
        if (!ModelState.IsValid) return BadRequest();
        _userService.UpdateEmail(id, email);
        var previousEmail = _dbContext
            .Set<User>().Where(x => x.Id == id)
            .Select(x => x.Email).First();

        _emailSender.SendEmail(email,
            "Ваш email изменен",
             $"с {previousEmail} на {email}");
    }
    //...
}
```

```
public IActionResult UpdateEmailWithService(int id, string email)
{
    try
    {
        if (!Mod                                                    ;
        _userService.UpdateEmail(id, email);
        var previousEmail = _dbContext
            .Set<User>().Where(x => x.Id == id)
            .Select(x => x.Email).First();

        _emailSender.SendEmail(email,
            "Ваш email изменен",
             $"с {previousEmail} на {email}");
    }
    //...
}
```

- Стало лучше?

```
public IActionResult UpdateEmailWithService(int id, string email)
{
    try
    {
        if (!Mod                                                    ;
        _userSer
        var prev
            .Set
            .Sel

        _emailSe
            "Bal
             $"c
    }
    //...
}
```

- Стало лучше?
- Метод UpdateEmail можно использовать повторно

```
public IActionResult UpdateEmailWithService(int id, string email)
{
    try
    {
        if (!Mod                                                    ;
        _userSer
        var prev
            .Set
            .Sel

        _emailSe
            "Bas
             $"c
    }
    //...
}
```

- Стало лучше?
- Метод UpdateEmail можно использовать повторно
- Заглянем внутрь

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update email"))
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update email"))
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update ema
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

- Валидация

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update ema
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

- Валидация
- Обработка ошибок

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update ema
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чуж
                _logger.LogWarning(message);
                throw new SecurityException(message)
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

- Валидация
- Обработка ошибок
- Здесь или в контроллере?

```
public void UpdateEmail(int id, string email)
{
    try
    {
        using (MiniProfiler.Current.Step("Update ema
        {
            email = email.ToLower();
            var user = _userRepository.ById(id);
            if (user.Id != id)
            {
                var message = $"Попытка изменить чуж
                _logger.LogWarning(message);
                throw new SecurityException(message)
            }
            user.Email = email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

- Валидация
- Обработка ошибок
- Здесь или в контроллере?
- Стоит ли вызывать SaveChanges?


Что если я скажу тебе...
слои не делают код лучше

Альтернатива

Варианты использования вместо слоев

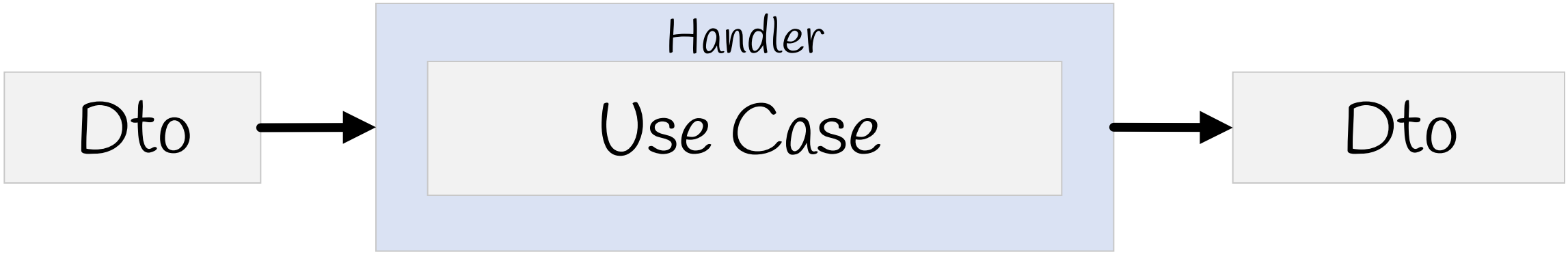
Вариант использования

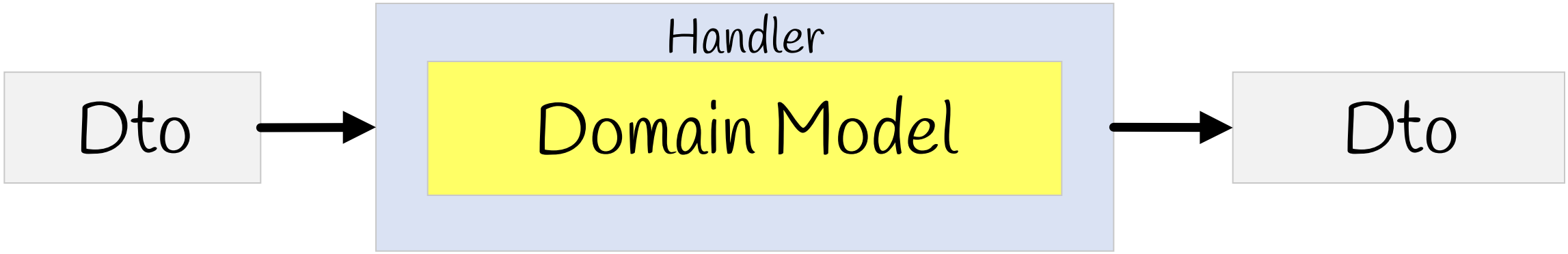
```
public interface IHandler<in TIn, out TOut>
{
    TOut Handle(TIn input);
}
```

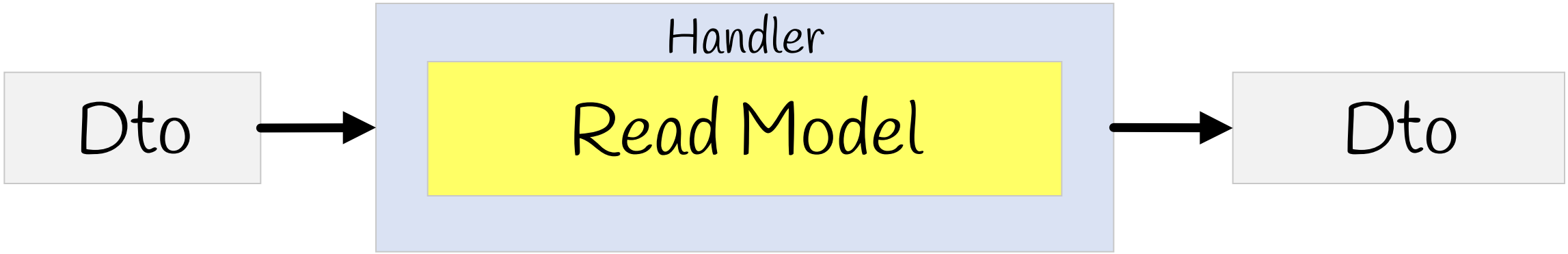
```
public interface IHandler<in TIn, out TOut>
{
    TOut Handle(TIn input);
}
```

```
public interface IHandler<in TIn, out TOut>
{
    TOut Handle(TIn input);
}
```

```
public interface IHandler<in TIn, out TOut>
{
    TOut Handle(TIn input);
}
```

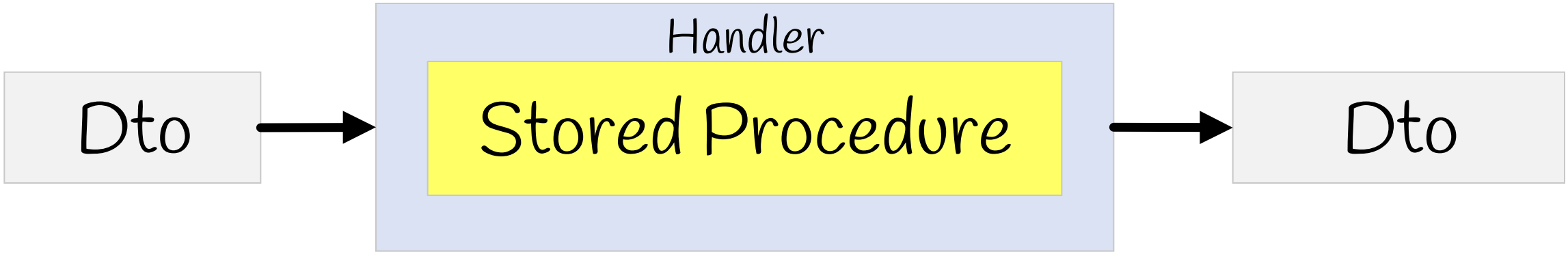

Handler
Dto
Use Case
Dto


Handler
Dto
Domain Model
Dto

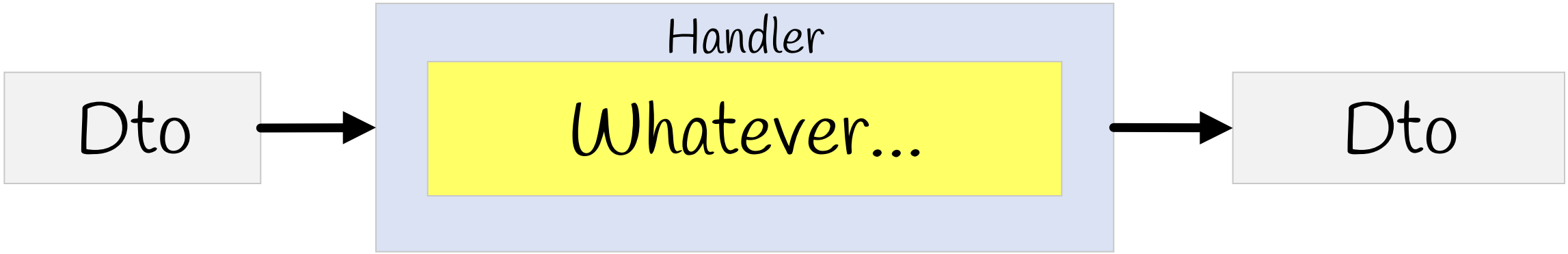

Handler
Dto
Read Model
Dto


Handler
Dto
Dapper
Dto


Handler
Dto
Elastic Search
Dto


Handler
Dto
Stored Procedure
Dto


Handler
Dto
Http Client
Dto


Handler
Dto
Whatever...
Dto


Слоев нет

UserService

User GetUser (int id)

void UpdateEmail (int id)

void BanUser (int id)

UserService

void UpdateEmail (int id)

void BanUser (int id)

class GetUser

UserService

void BanUser (int id)

class GetUser

class UpdateEmail

UserService

class GetUser

class UpdateEmail

class BanUser

UserService

class GetUser

class UpdateEmail

class BanUser

class GetUser

class UpdateEmail

class BanUser

Отлично,
сервисов нет…
Что дальше?

class GetUser

class UpdateEmail

class BanUser

Возвращает данные

class GetUser

class UpdateEmail

class BanUser

Возвращают результат операции и изменяют состояние

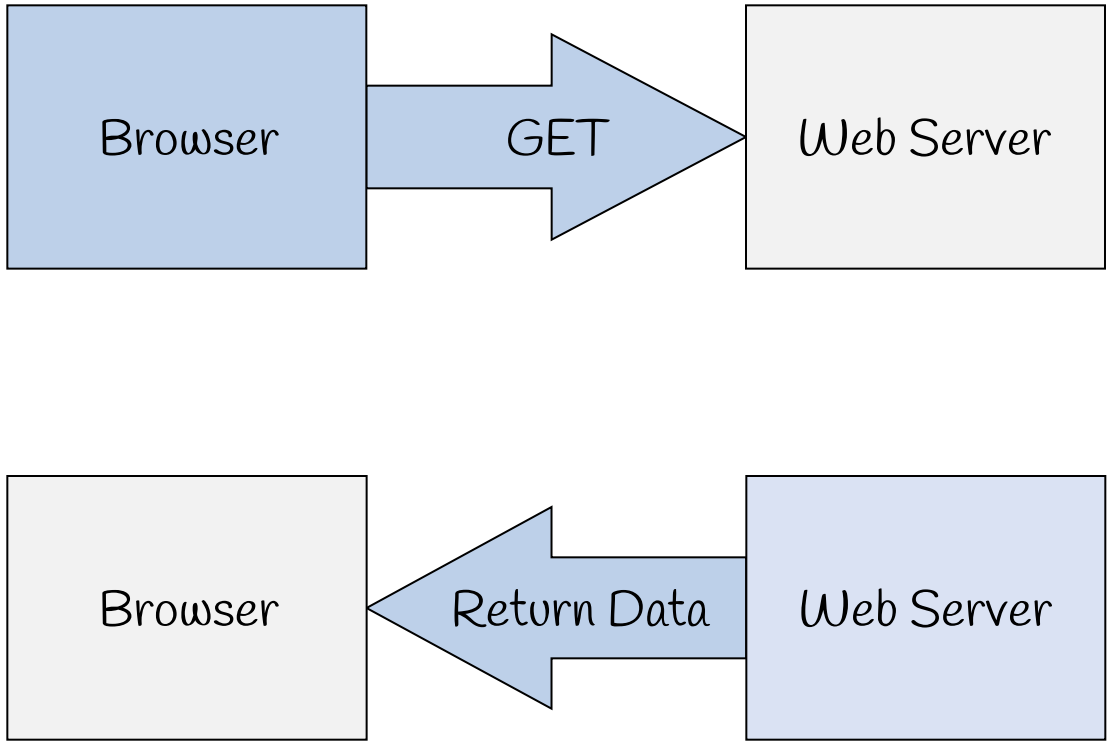
class GetUser

class UpdateEmail

class BanUser

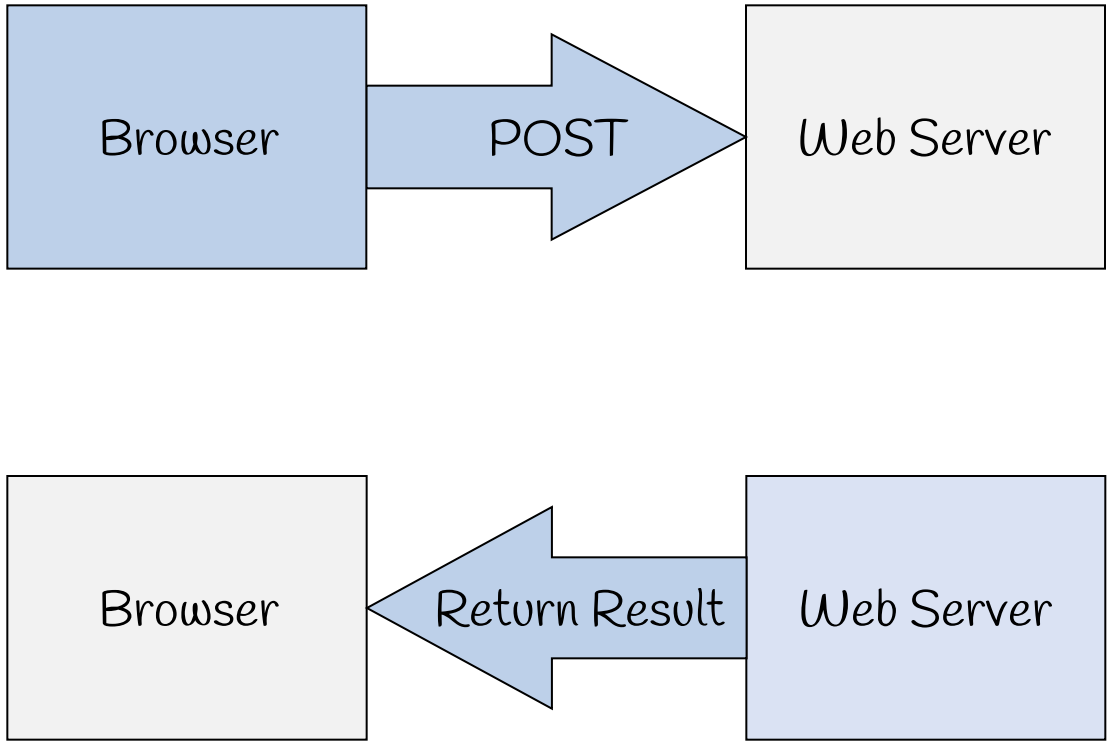
GET

- Возвращает данные
- Не меняет состояние сервера



POST, PUT, DELETE

- Возвращает результат операции
- Меняет состояние сервера



CQRS + HTTP = ♥

CQRS + HTTP = ♥

Не пиши где читаешь

```
public interface IQueryHandler<in TIn, out TOut>
    : IHandler<TIn, TOut>
    where TIn: IQuery<TOut>
```

```
public interface ICommandHandler<in TIn, out TOut>
    : IHandler<TIn, TOut>
    where TIn: ICommand<TOut>
```

```
public interface ICommandHandler<in TIn, out TOut>
    : IHandler<TIn, TOut>
    where TIn: ICommand<TOut>
```

Это пригодится чуть позже

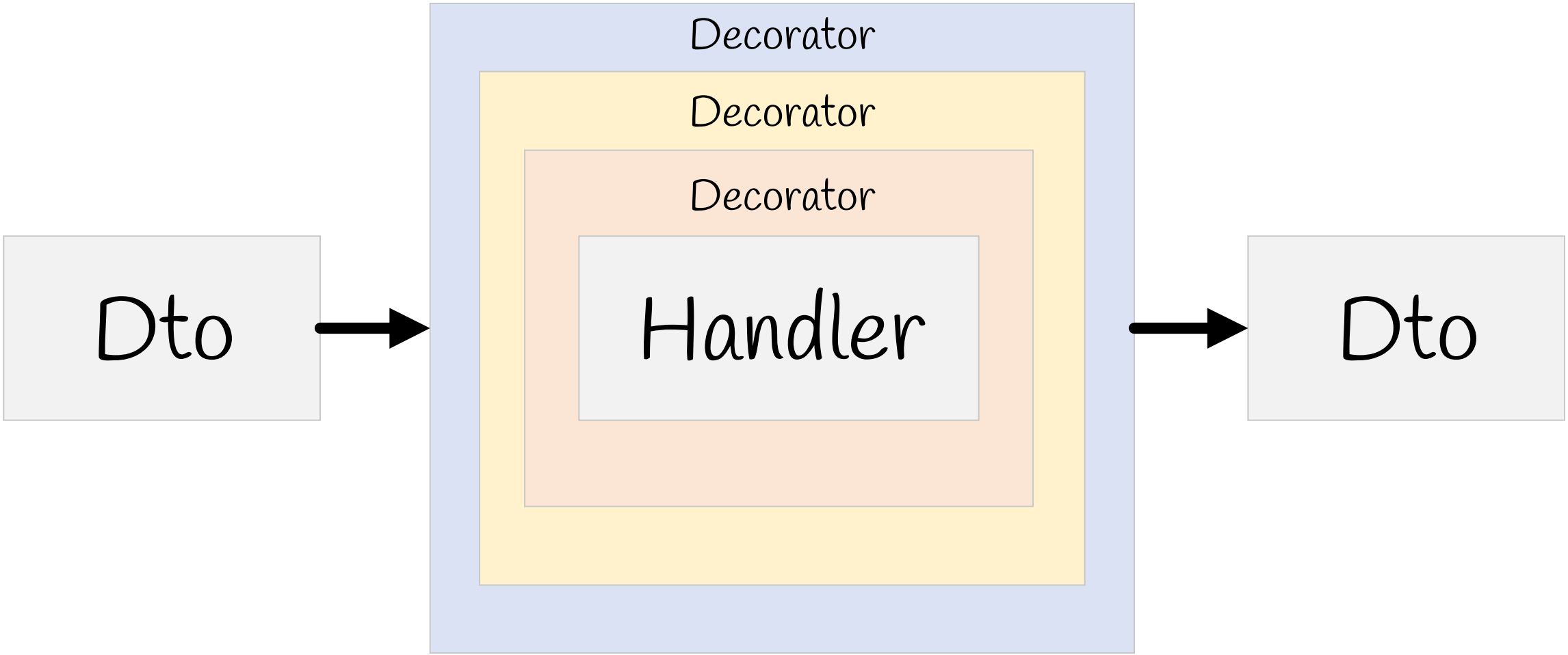
Реализация Handler

```
public void Handle(UpdateEmail command)
{
    try
    {
        using (MiniProfiler.Current.Step("Update email"))
        {
            command.Email = command.Email.ToLower();
            var user = _userRepository.ById(command.Id);
            if (user.Id != command.Id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{command.Id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = command.Email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

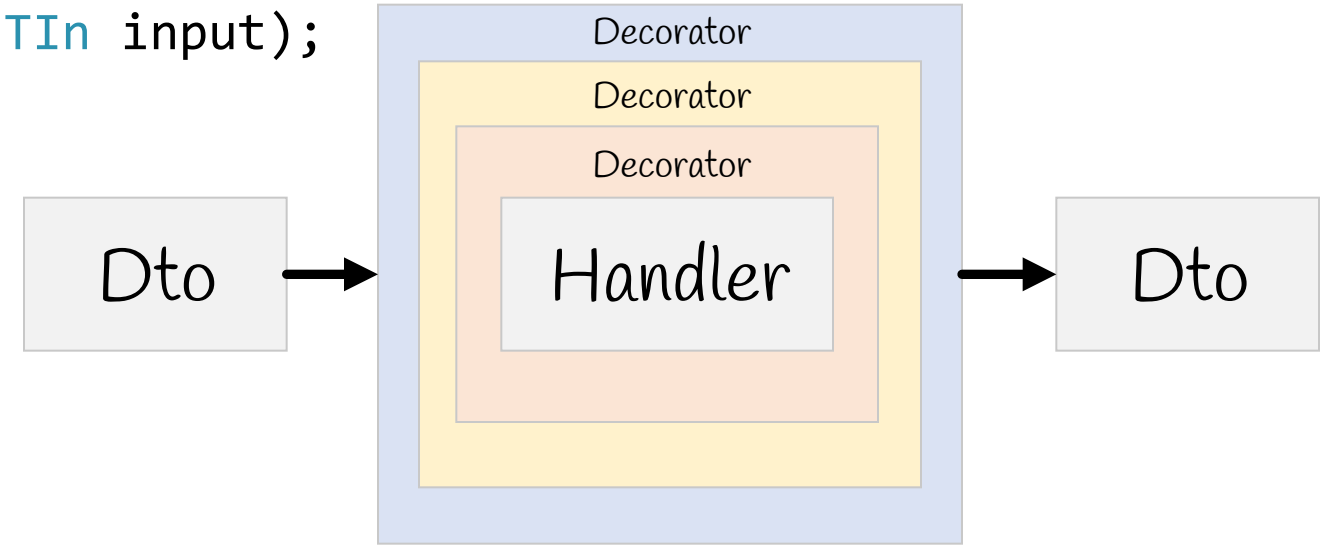
```
public void Handle(UpdateEmail command)
{
    try
    {
        using (MiniProfiler.Current.Step("Update email"))
        {
            command.Email = command.Email.ToLower();
            var user = _userRepository.ById(command.Id);
            if (user.Id != command.Id)
            {
                var message = $"Попытка изменить чужие данные {user.Id}/{command.Id}";
                _logger.LogWarning(message);
                throw new SecurityException(message);
            }
            user.Email = command.Email;
            _dbContext.SaveChanges();
        }
    }
    catch (DbUpdateException e)
    {
        _logger.LogCritical(e.Message);
        throw;
    }
}
```

Декораторы

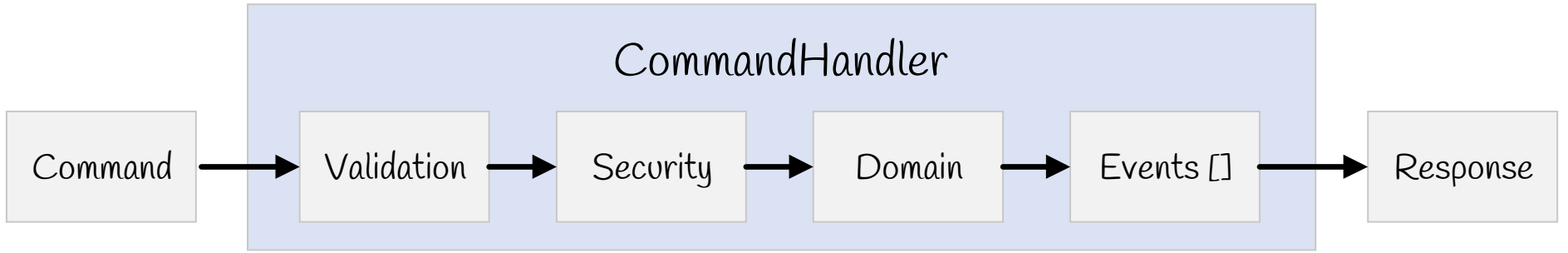
Спешат на помощь


Decorator
Decorator
Decorator
Dto
Handler
Dto

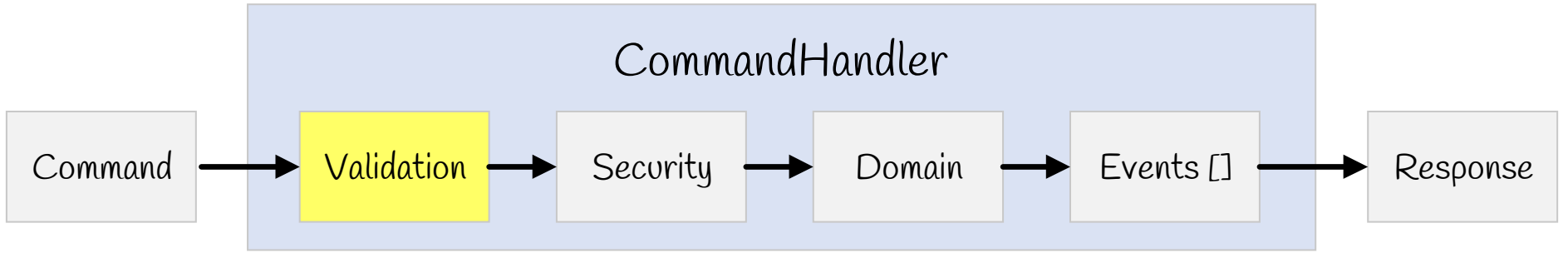
```
public abstract class HandlerDecoratorBase<TIn, TOut>: IHandler<TIn, TOut>
{
    protected readonly IHandler<TIn, TOut> Decorated;
    protected HandlerDecoratorBase(IHandler<TIn, TOut> decorated)
    {
        Decorated = decorated;
    }

    public abstract TOut Handle(TIn input);
}
```



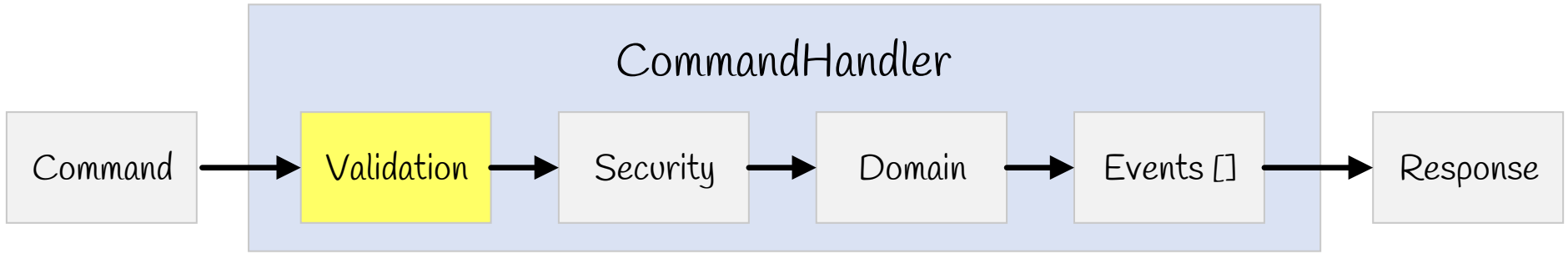
Pipeline


CommandHandler
Command
Validation
Security
Domain
Events []
Response

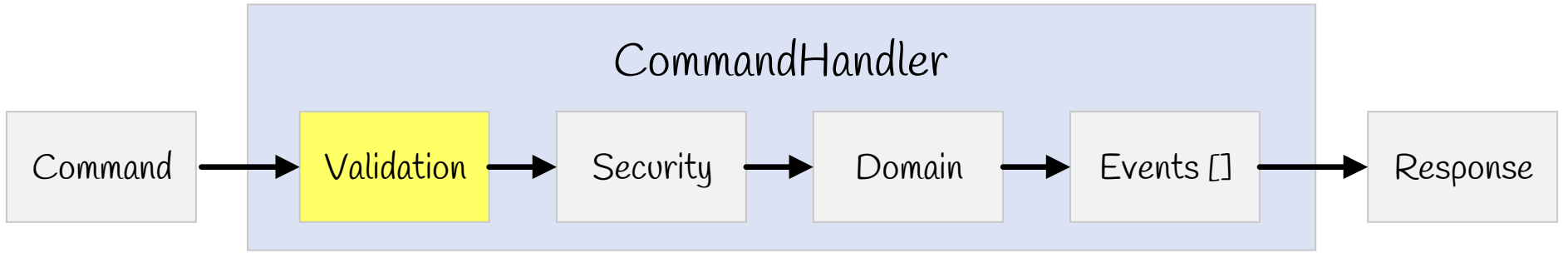
Validation


CommandHandler
Command
Validation
Security
Domain
Events []
Response

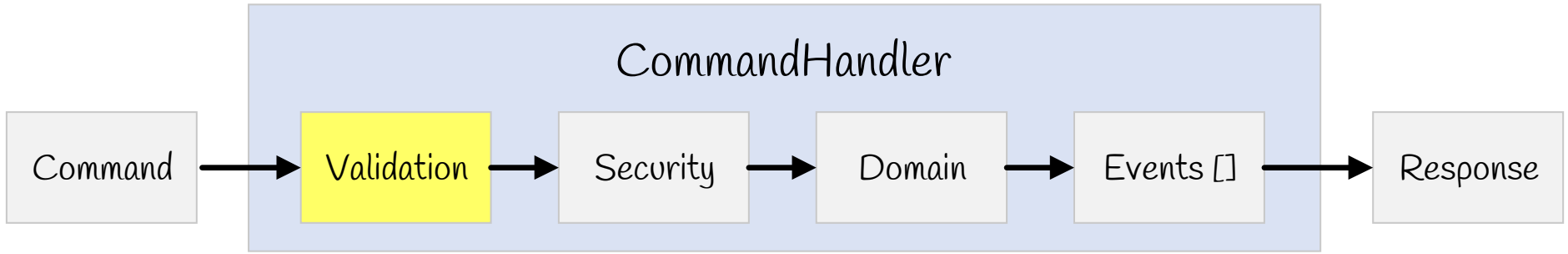
```
public override TOut Handle(TIn input)
{
    IEnumerable<ValidationResult> res = _validators.Validate(input);
    if (!res.IsValid())
    {
        if (typeof(TOut) == typeof(IEnumerable<ValidationResult>))
            return (TOut) res;

        throw new ValidationException(message);
    }

    return Decorated.Handle(input);
}
```



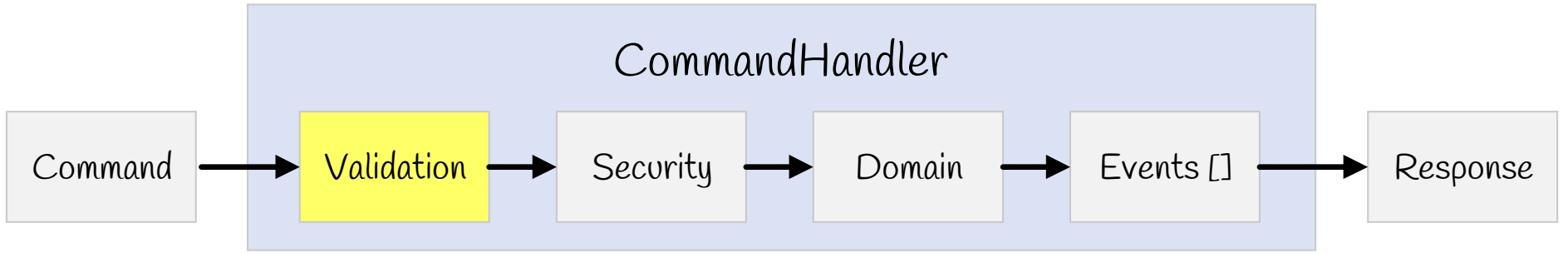
```
public override TOut Handle(TIn input)
{
    IEnumerable<ValidationResult> res = _validators.Validate(input);
    if (!res.IsValid())
    {
        if (typeof(TOut) == typeof(IEnumerable<ValidationResult>))
            return (TOut) res;

        throw new ValidationException(message);
    }

    return Decorated.Handle(input);
}
```



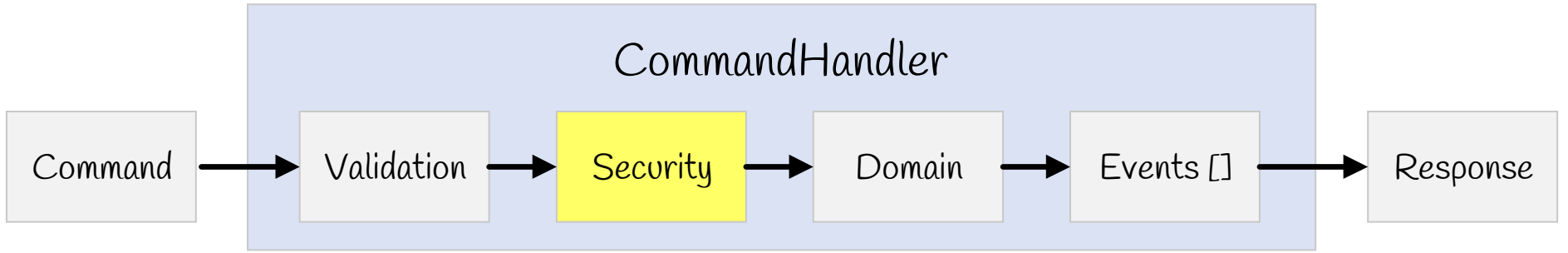
```
public override TOut Handle(TIn input)
{
    IEnumerable<ValidationResult> res = _validators.Validate(input);
    if (!res.IsValid())
    {
        if (typeof(TOut) == typeof(IEnumerable<ValidationResult>))
            return (TOut) res;

        throw new ValidationException(message);
    }

    return Decorated.Handle(input);
}
```



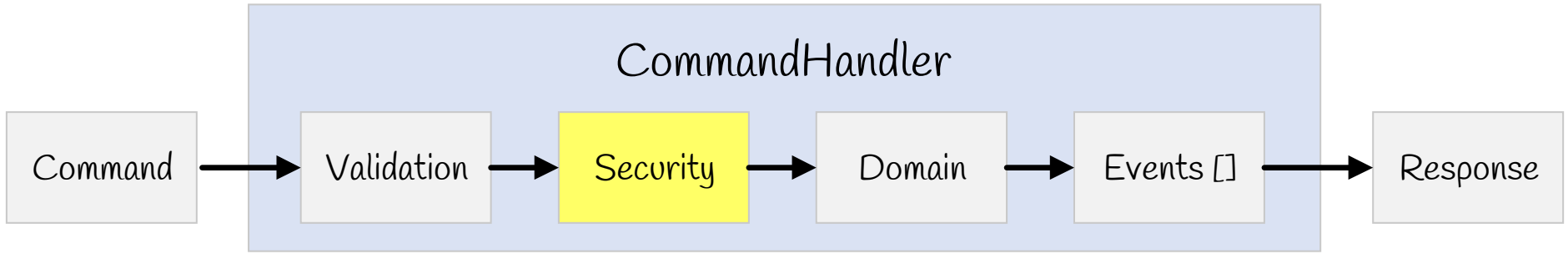
```
public override TOut Handle(TIn input)
{
    IEnumerable<ValidationResult> res = _validators.Validate(input);
    if (!res.IsValid())
    {
        if (typeof(TOut) == typeof(IEnumerable<ValidationResult>))
            return (TOut) res;

        throw new ValidationException(message);
    }

    return Decorated.Handle(input);
}
```



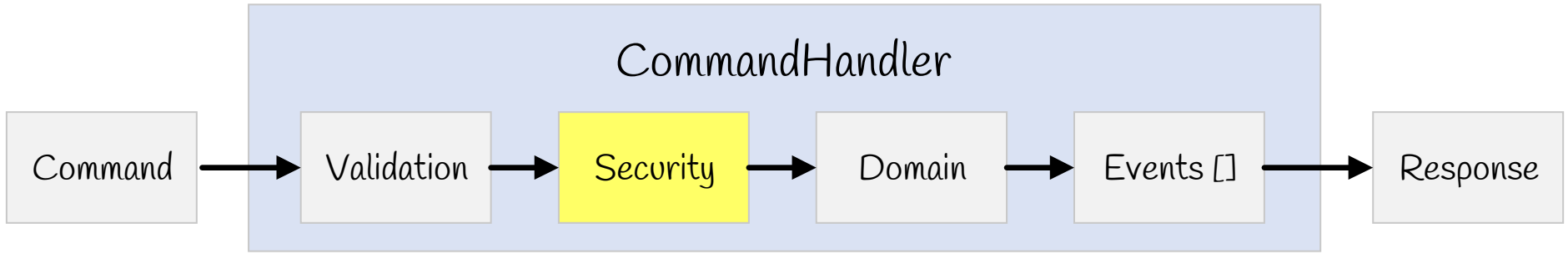
Security


CommandHandler
Command
Validation
Security
Domain
Events []
Response

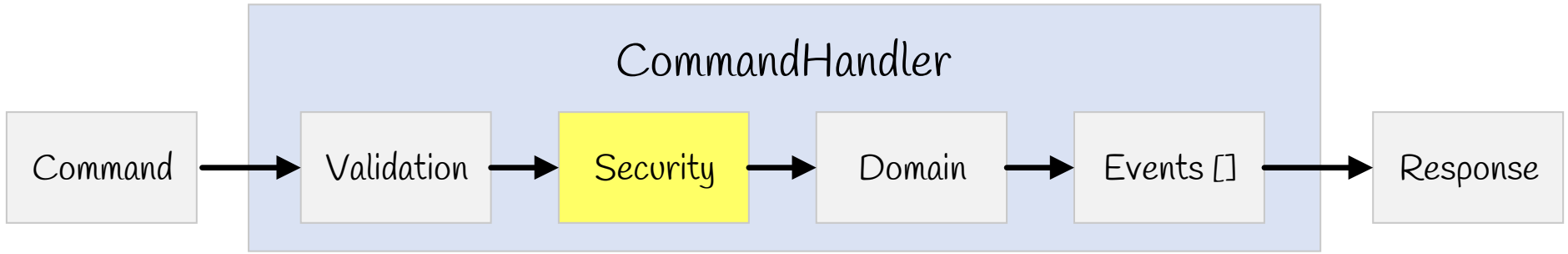
```
public override TOut Handle(TIn input)
{
        if (!CheckPermissions(input))
        {
            throw new SecurityException();
        }

        return Decorated.Handle(input);
}
```



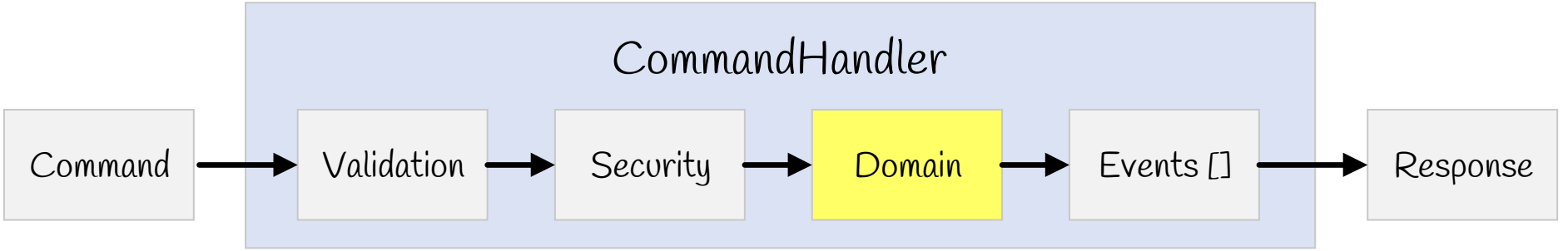
```
public override TOut Handle(TIn input)
{
        if (!CheckPermissions(input))
        {
            throw new SecurityException();
        }

        return Decorated.Handle(input);
}
```



```
public override TOut Handle(TIn input)
{
        if (!CheckPermissions(input))
        {
            throw new SecurityException();
        }

        return Decorated.Handle(input);
}
```



Domain


CommandHandler
Command
Validation
Security
Domain
Events []
Response

Одержимость примитивами

```
public class UpdateUserEmail: IValidatableCommand
{
    public int UserId { get; set; }

    public string Email { get; set; }
}
```

```
public class UpdateUserEmail: IValidatableCommand
{
    [Required, Id]
    public int UserId { get; set; }

    [Required, EmailAddress]
    public string Email { get; set; }
}
```

```
public class UpdateUserEmail: IValidatableCommand
{
    [Required, Id]
    public int UserId { get; set; }

    [Required, EmailAddress]
    public string Email { get; set; }
}
```

- Как быть с проверкой значений в БД?

```
public class UpdateUserEmail: IValidatableCommand
{
    [Required, Id]
    public int UserId { get; set; }

    [Required, EmailAddress]
    public string Email { get; set; }
}
```

- Как быть с проверкой значений в БД?
- Добавить типы!

```
public class Id<T>: Id<int, T>
    where T : class, IHasId<int>
{
    public Id(T entity) : base(entity) {}

    public Id(int value, Func<int, T> loader)
        : base(value, loader) {}
}
```

```
public class Id<T>: Id<int, T>
    where T : class, IHasId<int>
{
    public Id(T entity) : base(entity) {}

    public Id(int value, Func<int, T> loader)
        : base(value, loader) {}
}
```

```
public class Id<T>: Id<int, T>
    where T : class, IHasId<int>
{
    public Id(T entity) : base(entity) {}

    public Id(int value, Func<int, T> loader)
        : base(value, loader) {}
}
```

```
public class Id<T>: Id<int, T>
    where T : class, IHasId<int>
{
    public Id(T entity) : base(entity) {}

    public Id(int value, Func<int, T> loader)
        : base(value, loader) {}
}
```

```
public class Id<T>: Id<int, T>
    where T : class, IHasId<int>
{
    public Id(T entity) : base(entity) {}

    public Id(int value, Func<int, T> loader)
        : base(value, loader) {}
}
```

```
[JsonConverter(typeof(ValueTypeConverter))]
[ModelBinder(typeof(EmailModelBinder))]
public class Email: StringValueObject
{
    private static readonly EmailAddressAttribute Attr
        = new EmailAddressAttribute();

    public Email(string email): base(email?.ToLowerInvariant())
    {
        if (!Attr.IsValid(email))
        {
            throw new ArgumentException(nameof(email),
                $"\"{email}\" is not valid email");
        }
    }
}
```

```
[JsonConverter(typeof(ValueTypeConverter))]
[ModelBinder(typeof(EmailModelBinder))]
public class Email: StringValueObject
{
    private static readonly EmailAddressAttribute Attr
        = new EmailAddressAttribute();

    public Email(string email): base(email?.ToLowerInvariant())
    {
        if (!Attr.IsValid(email))
        {
            throw new ArgumentException(nameof(email),
                $"\"{email}\" is not valid email");
        }
    }
}
```

```
[JsonConverter(typeof(ValueTypeConverter))]
[ModelBinder(typeof(EmailModelBinder))]
public class Email: StringValueObject
{
    private static readonly EmailAddressAttribute Attr
        = new EmailAddressAttribute();

    public Email(string email): base(email?.ToLowerInvariant())
    {
        if (!Attr.IsValid(email))
        {
            throw new ArgumentException(nameof(email),
                $"\"{email}\" is not valid email");
        }
    }
}
```

```
[JsonConverter(typeof(ValueTypeConverter))]
[ModelBinder(typeof(EmailModelBinder))]
public class Email: StringValueObject
{
    private static readonly EmailAddressAttribute Attr
        = new EmailAddressAttribute();

    public Email(string email): base(email?.ToLowerInvariant())
    {
        if (!Attr.IsValid(email))
        {
            throw new ArgumentException(nameof(email),
                $"\"{email}\" is not valid email");
        }
    }
}
```

```
[JsonConverter(typeof(ValueTypeConverter))]
[ModelBinder(typeof(EmailModelBinder))]
public class Email: StringValueObject
{
    private static readonly EmailAddressAttribute Attr
        = new EmailAddressAttribute();

    public Email(string email): base(email?.ToLowerInvariant())
    {
        if (!Attr.IsValid(email))
        {
            throw new ArgumentException(nameof(email),
                $"\"{email}\" is not valid email");
        }
    }
}
```

```
public class UpdateUserEmail: IValidatableCommand
{
    [Required]
    public Id<User> UserId { get; set; }

    [Required]
    public Email Email { get; set; }
}
```

```
public class UpdateUserEmail: IValidatableCommand
{
    [Required]
    public Id<User> UserId { get; set; }

    [Required]
    public Email Email { get; set; }
}
```

Эти значения точно корректны

Инварианты

```
public class User: EntityBase, IHasDomainEvents
{
    protected User() {}
    public User(Email email, string firstName, string lastName)
    {
        Created = DateTime.UtcNow;
        Email = email
            ?? throw new ArgumentNullException(nameof(email));
        ChangeName(firstName, lastName);
    }
}
```

```
public class User: EntityBase, IHasDomainEvents
{
    protected User() {}
    public User(Email email, string firstName, string lastName)
    {
        Created = DateTime.UtcNow;
        Email = email
            ?? throw new ArgumentNullException(nameof(email));
        ChangeName(firstName, lastName);
    }
}
```

- Ждем nullable reference type в C#8

```
public class User: EntityBase, IHasDomainEvents
{
    protected User() {}
    public User(Email email, string firstName, string lastName)
    {
        Created = DateTime.UtcNow;
        Email = email
            ?? throw new ArgumentNullException(nameof(email));
        ChangeName(firstName, lastName);
    }
}
```

- Ждем nullable reference type в C#8
- Имя и фамилию меняем вместе

```
private static DefaultStringLengthAttribute _attr
    = new DefaultStringLengthAttribute();

public void ChangeName(string firstName, string lastName)
{
    if (!_defaultStringLength.IsValid(firstName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(firstName));

    if (!_defaultStringLength.IsValid(lastName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(lastName));

    FirstName = firstName;
    LastName = lastName;
}
```

```
private static DefaultStringLengthAttribute _attr
    = new DefaultStringLengthAttribute();

public void ChangeName(string firstName, string lastName)
{
    if (!_defaultStringLength.IsValid(firstName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(firstName));

    if (!_defaultStringLength.IsValid(lastName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(lastName));

    FirstName = firstName;
    LastName = lastName;
}
```

```
private static DefaultStringLengthAttribute _attr
    = new DefaultStringLengthAttribute();

public void ChangeName(string firstName, string lastName)
{
    if (!_defaultStringLength.IsValid(firstName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(firstName));

    if (!_defaultStringLength.IsValid(lastName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(lastName));

    FirstName = firstName;
    LastName = lastName;
}
```

```
private static DefaultStringLengthAttribute _attr
    = new DefaultStringLengthAttribute();

public void ChangeName(string firstName, string lastName)
{
    if (!_defaultStringLength.IsValid(firstName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(firstName));

    if (!_defaultStringLength.IsValid(lastName))
        throw new ArgumentException(_ attr.ErrorMessage, nameof(lastName));

    FirstName = firstName;
    LastName = lastName;
}
```

Можно повторно использовать в DTO

```
public class UpdateUserInfo: IValidatableCommand
{
    [Required, DefaultStringLength]
    public string FirstName { get; set;}

    [Required, DefaultStringLength]
    public string LastName { get; set; }
}
```

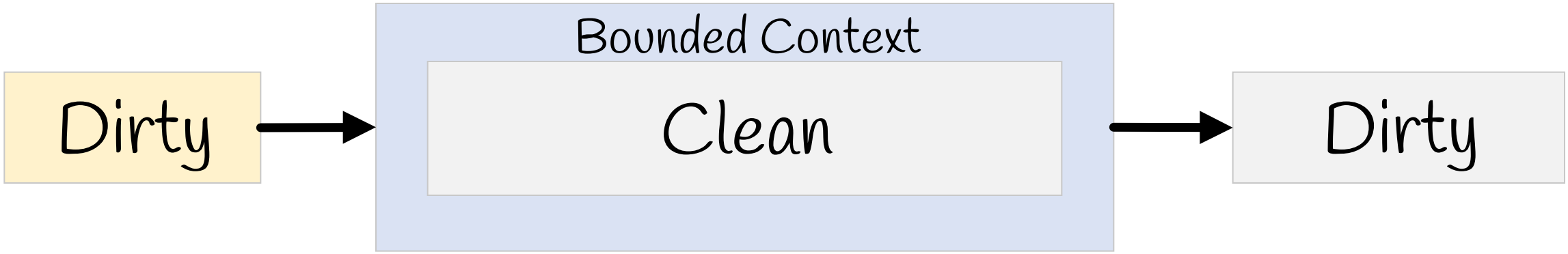
```
public class UpdateUserInfo
{
    [Required, DefaultStringLength]
    public string FirstName { get; set;}

    [Required, DefaultStringLength]
    public string LastName { get; set; }
}
```

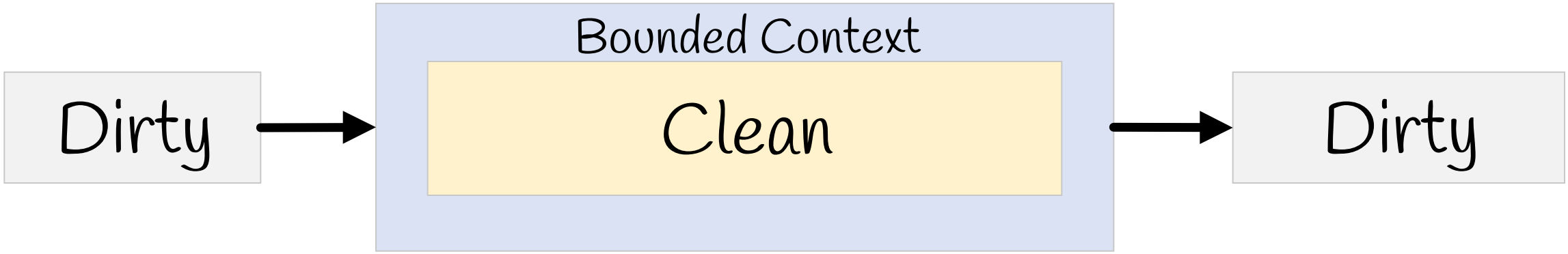
А как же конструктор?

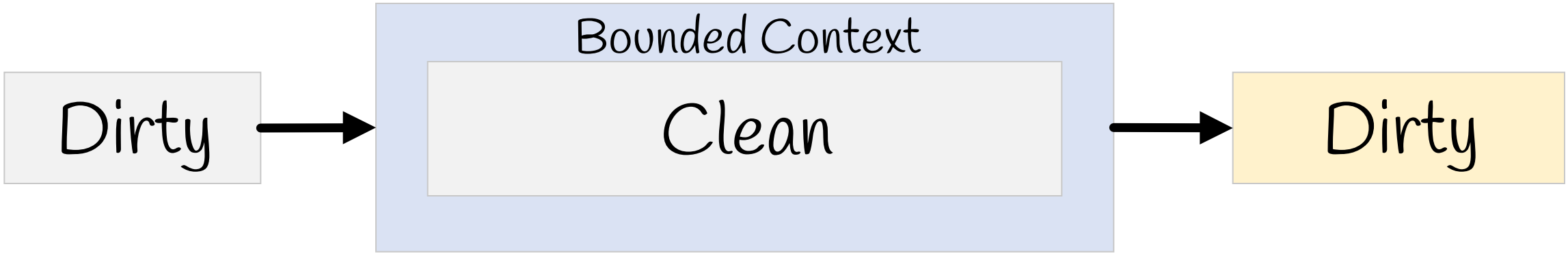
```
public class UpdateUserInfo
{
    [Required, DefaultStringLength]
    public string FirstName { get; set;}

    [Required, DefaultStringLength]
    public string LastName { get; set; }
}
```

А как же конструктор?

На границах программы **не объектно-ориентированы**


Bounded Context
Dirty
Clean
Dirty


Bounded Context
Dirty
Clean
Dirty


Bounded Context
Dirty
Clean
Dirty

Есть вопросы про реализацию DDD?

Сегодня 14:30 – 16:00 на стенде DotNetRu

Handler

```
public void Handle(UpdateUserEmail command)
{
    var user = command.UserId.Entity;
    user.Email = command.Email;
}
```

```
public void Handle(UpdateUserEmail command)
{
    var user = command.UserId.Entity;
    user.Email = command.Email;
}
```

- Данные корректны, email не занят

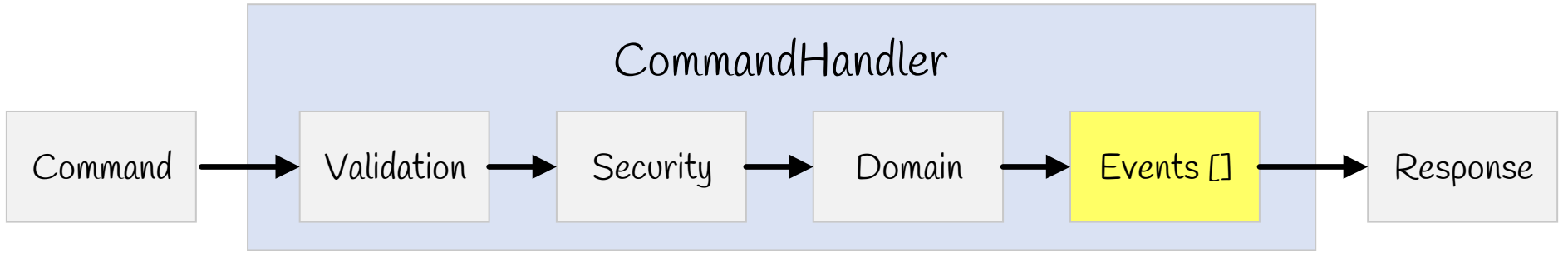
```
public void Handle(UpdateUserEmail command)
{
    var user = command.UserId.Entity;
    user.Email = command.Email;
}
```

- Данные корректны, email не занят
- Такой пользователь есть и мы имеем право менять ему email

```
public void Handle(UpdateUserEmail command)
{
    var user = command.UserId.Entity;
    user.Email = command.Email;
}
```

- Данные корректны, email не занят
- Такой пользователь есть и мы имеем право менять ему email
- Где SaveChanges(), нотификации, логи и профайлер?

Events


CommandHandler
Command
Validation
Security
Domain
Events []
Response

```
DomainEvents.Raise
(new Event())
```

```
DomainEvents.Raise
(new Event())
```

У меня есть
кое-что получше!

```
public Email Email
{
    get => _email;
    set
    {
        if (value != _email) _domainEventStore.Raise(
            new UserEmailChanged(this, _email, value));
        _email = value;
    }
}
```

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
    _dbContext.ChangeTracker
        .Entries()
        .Where(x => x is IHasDomainEvents)
        .SelectMany(x => ((IHasDomainEvents)x)
            .GetDomainEvents())
        .ToList()
        .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
    _dbContext.ChangeTracker
        .Entries()
        .Where(x => x is IHasDomainEvents)
        .SelectMany(x => ((IHasDomainEvents)x)
            .GetDomainEvents())
        .ToList()
        .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
    _dbContext.ChangeTracker
        .Entries()
        .Where(x => x is IHasDomainEvents)
        .SelectMany(x => ((IHasDomainEvents)x)
            .GetDomainEvents())
        .ToList()
        .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
    _dbContext.ChangeTracker
        .Entries()
        .Where(x => x is IHasDomainEvents)
        .SelectMany(x => ((IHasDomainEvents)x)
            .GetDomainEvents())
        .ToList()
        .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
     dbContext.ChangeTracker
          .SelectMany(x => ((IHasDomainEvents)x)
              .GetDomainEvents())
          .ToList()
          .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

- Почему просто не переопределить метод SaveChanges()?

```
public TOut Handle(TIn input)
{
    var res = _decorated.Handle(input);
     dbContext.ChangeTracker
          .ForEach(_dispatcher.Handle);
    _dbContext.SaveChanges();
    return res;
}
```

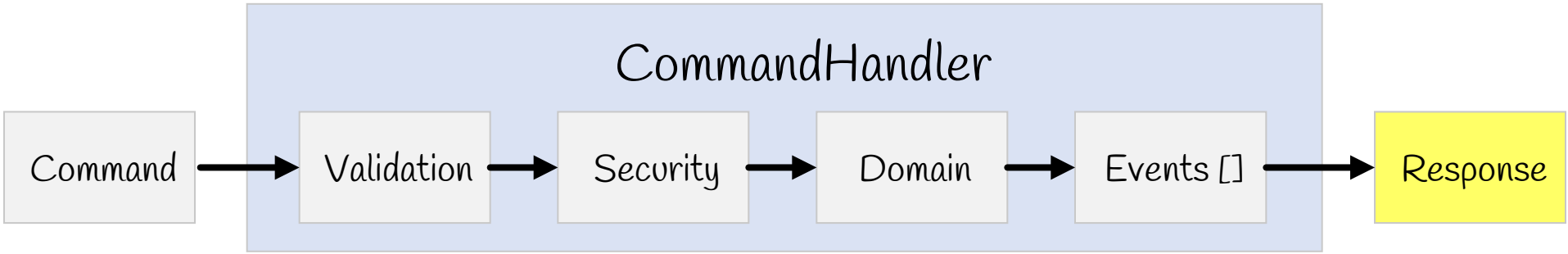
- Почему просто не переопределить метод SaveChanges()?
- Чтобы избежать циклических зависимостей

Логирование и профилирование

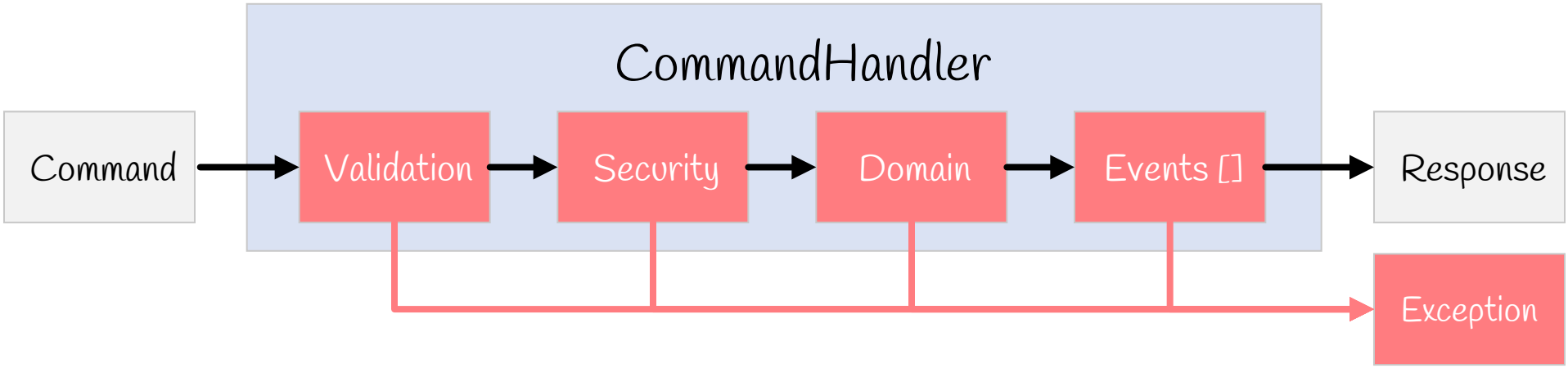
```
public override TOut Handle(TIn input)
{
    var output = Decorated.Handle(input);
    _logger.LogInformation(
        $"{_decoratedType}: {input} => {output}");
    return output;
}

public override TOut Handle(TIn input)
{
    using (MiniProfiler.Current.Step(_decoratedType.ToString()))
    {
        return Decorated.Handle(input);
    }
}
```

```
public override TOut Handle(TIn input)
{
    var output = Decorated.Handle(input);
    _logger.LogInformation(
        $"{_decoratedType}: {input} => {output}");
    return output;
}

public override TOut Handle(TIn input)
{
    using (MiniProfiler.Current.Step(_decoratedType.ToString()))
    {
        return Decorated.Handle(input);
    }
}
```

Response

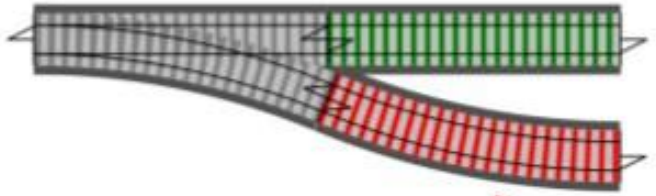

CommandHandler
Command
Validation
Security
Domain
Events []
Response

Мы кое-что забыли



Railway Oriented Programming

A functional approach to error handling



What do railways have to do with programming?

```
public class Result<TSuccess, TFailure>
{
    public Result(TSuccess success)
    {
        _success = success;
        _isSuccess = true;
    }

    public Result(TFailure failure)
    {
        _failure = failure;
    }
}
```

```
public class Result<TSuccess, TFailure>
{
    public Result(TSuccess success)
    {
        _success = success;
        _isSuccess = true;
    }

    public Result(TFailure failure)
    {
        _failure = failure;
    }
}
```

```
public class Result<TSuccess, TFailure>
{
    public Result(TSuccess success)
    {
        _success = success;
        _isSuccess = true;
    }

    public Result(TFailure failure)
    {
        _failure = failure;
    }
}
```

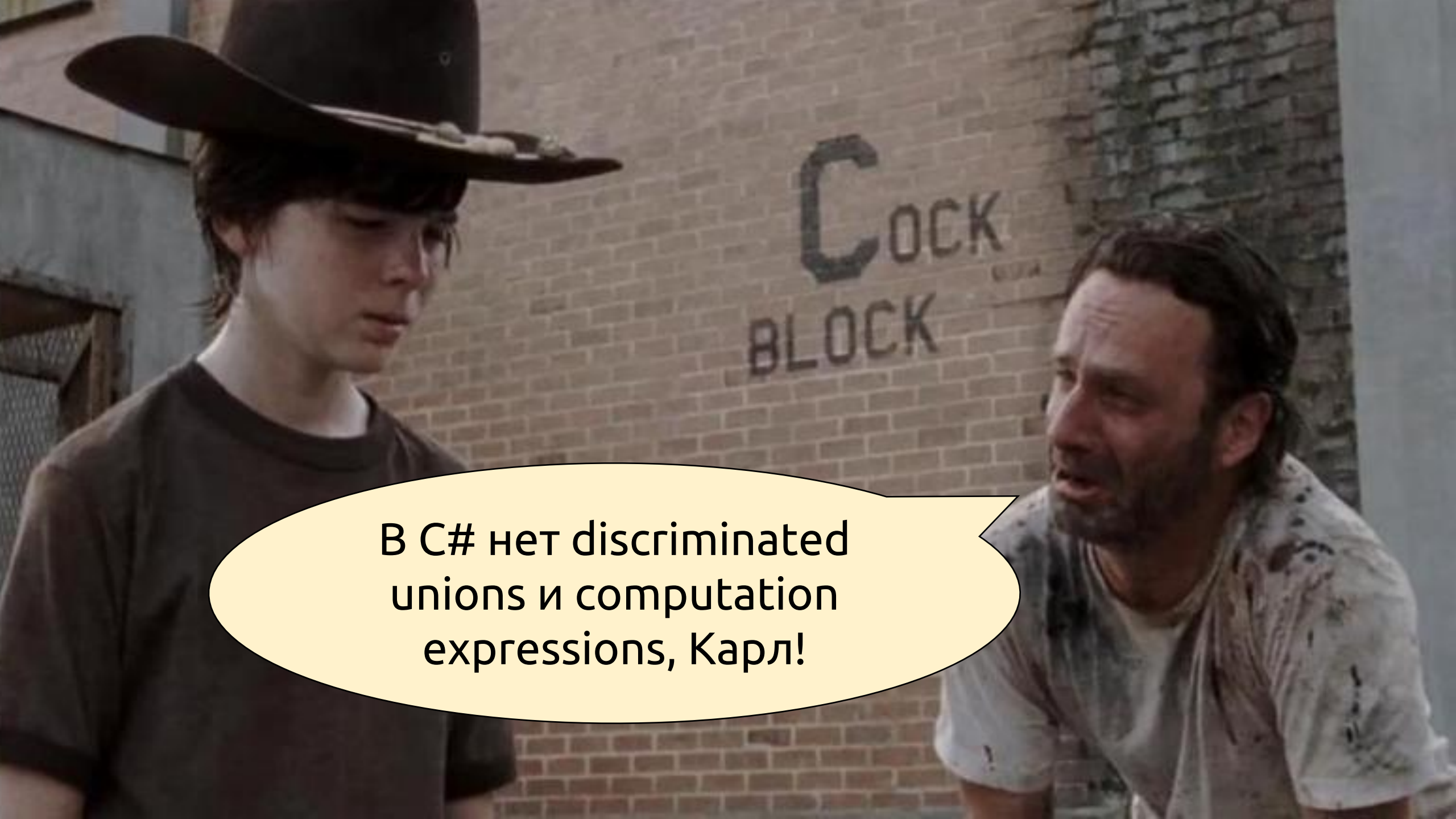
```
public TResult Match<TResult>(
    Func<TSuccess, TResult> success,
    Func<TFailure, TResult> failure)
    => _isSuccess
        ? success(_success)
        : failure(_failure);
```

```
public TResult Match<TResult>(
    Func<TSuccess, TResult> success,
    Func<TFailure, TResult> failure)
    => _isSuccess
        ? success(_success)
        : failure(_failure);
```

```
public TResult Match<TResult>(
    Func<TSuccess, TResult> success,
    Func<TFailure, TResult> failure)
    => _isSuccess
        ? success(_success)
        : failure(_failure);
```

```
from r1 in result1
from r2 in result2
from r3 in result3
select r1 + r2 + r3;
```

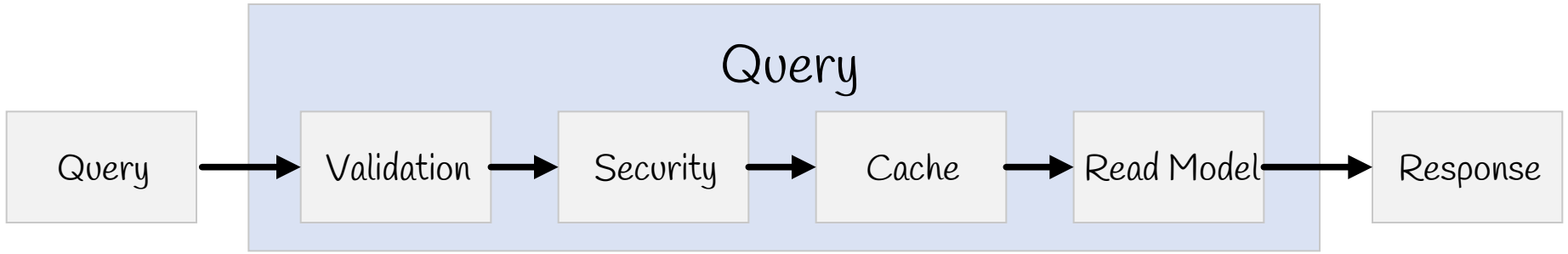

Exception’ы – зло!
Пишите непонятный код


COCK
BLOCK
В C# нет discriminated unions и computation expressions, Карл!

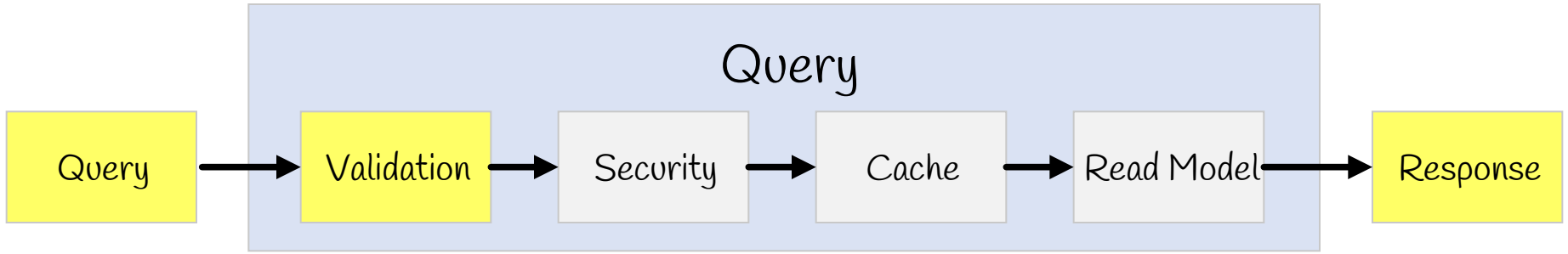

```
app.UseExceptionHandler(appError =>
{
    appError.Run(async context =>
    {
        //…
    });
});
```

- Валидация – 422
- Аутентификация - 401
- Авторизация – 403
- Сервер – 500
- IHasUserMessage

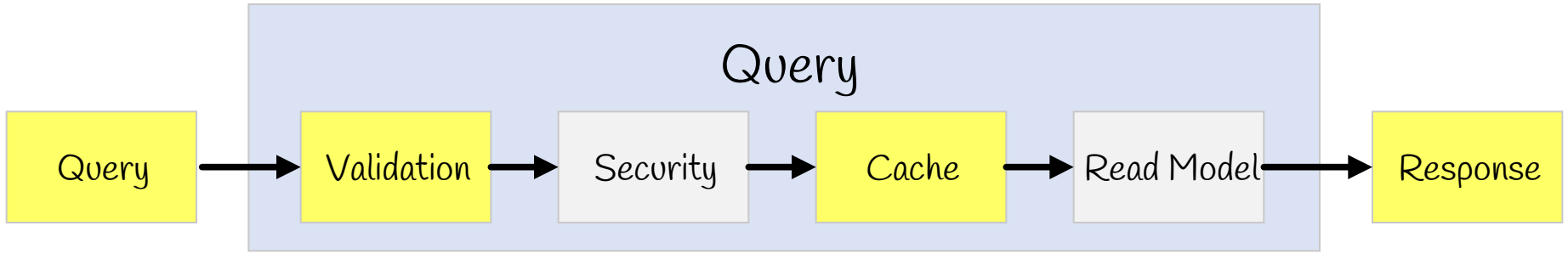
Query Pipeline



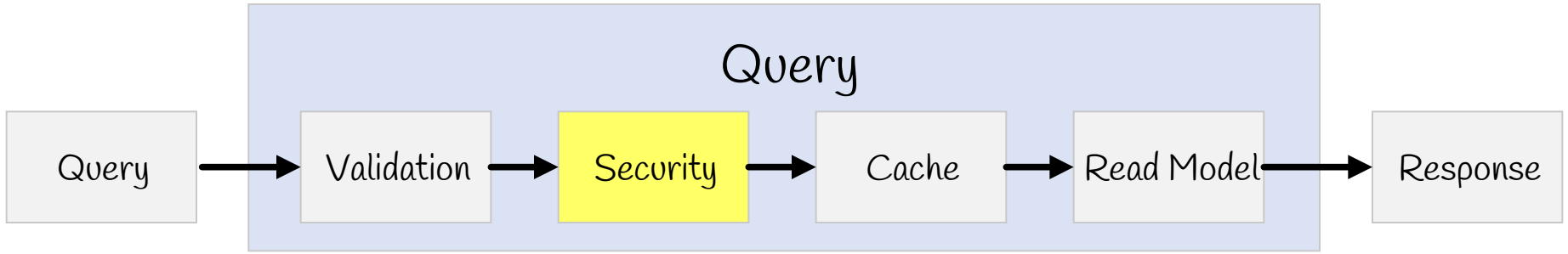
Query Pipeline



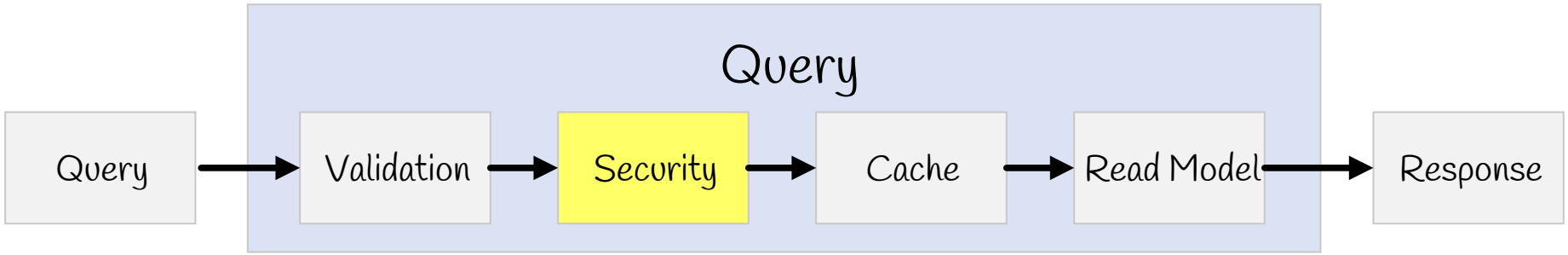
Query Pipeline



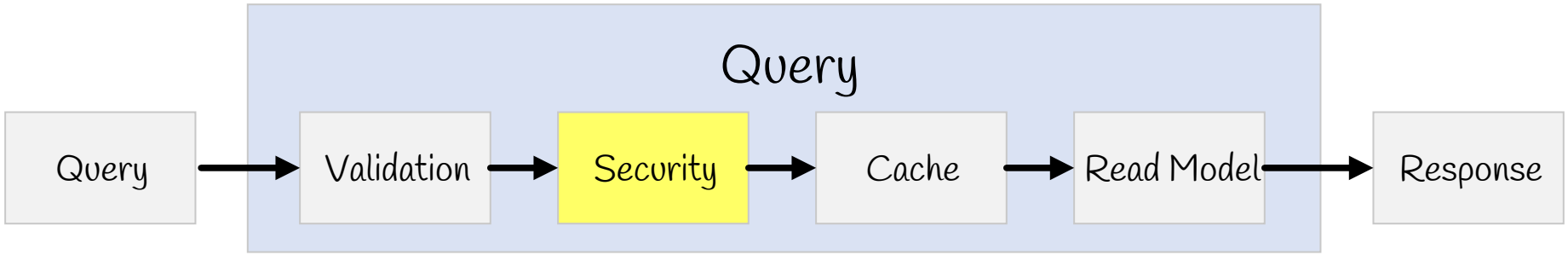
Security


Query
Query
Validation
Security
Cache
Read Model
Response

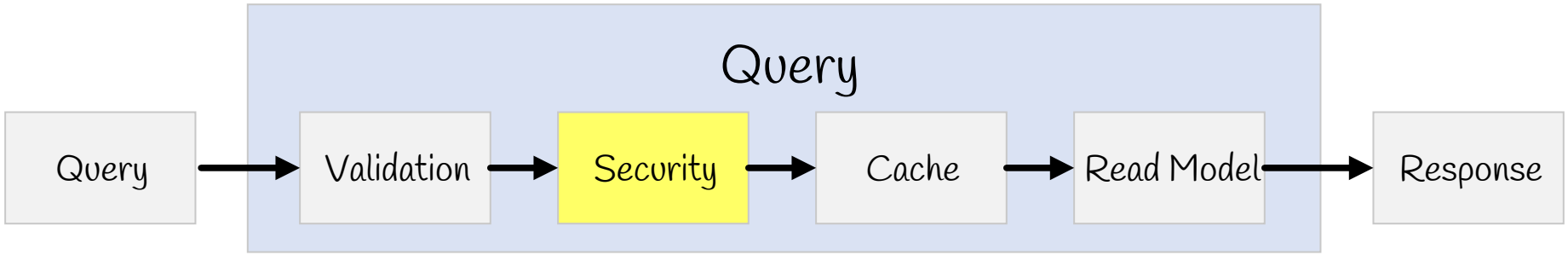
```
public interface IPermissionFilter<T>
{
    IQueryable<T> GetPermitted(IQueryable<T> queryable);
}
```



```
public interface IPermissionFilter<T>
{
    IQueryable<T> GetPermitted(IQueryable<T> queryable);
}
```



```
public interface IPermissionFilter<T>
{
    IQueryable<T> GetPermitted(IQueryable<T> queryable);
}

public IQueryable<T> WithFilters()
    => _permissionFilters.Aggregate(
        (IQueryable<T>) _dbContext.Set<T>(),
        (current, permissionFilter)
            => permissionFilter.GetPermitted(current));
```



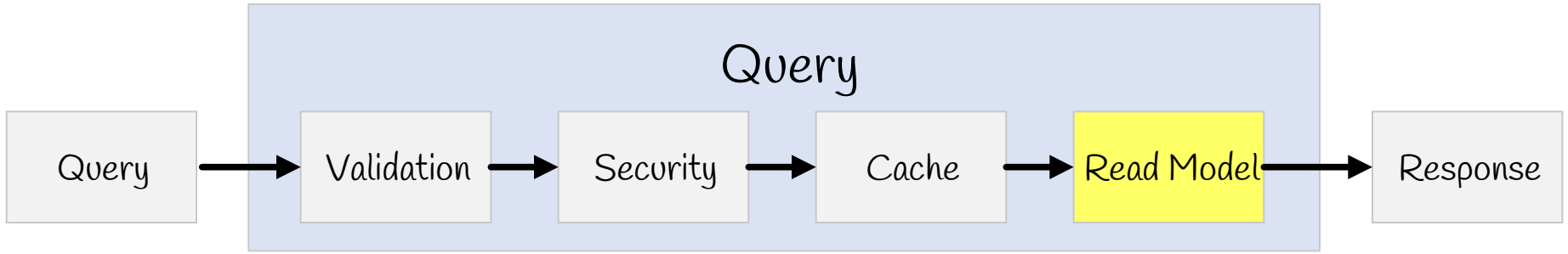
```
public interface IPermissionFilter<T>
{
    IQueryable<T> GetPermitted(IQueryable<T> queryable);
}

public IQueryable<T> WithFilters()
    => _permissionFilters.Aggregate(
        (IQueryable<T>) _dbContext.Set<T>(),
        (current, permissionFilter)
            => permissionFilter.GetPermitted(current));
```

- Почему не Global Query Filters?

```
public interface IPermissionFilter<T>
{
    IQueryable<T> GetPermitted(IQueryable<T> queryable);
}

public IQueryable<T> WithFilters()
    => _permissionFilters.Aggregate(
        (IQueryable<T>) _dbContext.Set<T>(),
        (current, permissionFilter)
            => permissionFilter.GetPermitted(current));
```

- Почему не Global Query Filters?
- Чтобы избежать циклических зависимостей

Query Pipeline



```
public class LinqQueryHandler<TQuery, TEntity, TProjection>
    : IQueryHandler<TQuery, IEnumerable<TProjection>>

    where TQuery
        : IQuery<IEnumerable<TProjection>>
        , IFilter<TProjection>
        , ISorter<TProjection>

    where TEntity : class
```

```
public class LinqQueryHandler<TQuery, TEntity, TProjection>
    : IQueryHandler<TQuery, IEnumerable<TProjection>>

    where TQuery
        : IQuery<IEnumerable<TProjection>>
        , IFilter<TProjection>
        , ISorter<TProjection>

    where TEntity : class
```

```
public class LinqQueryHandler<TQuery, TEntity, TProjection>
    : IQueryHandler<TQuery, IEnumerable<TProjection>>

    where TQuery
        : IQuery<IEnumerable<TProjection>>
        , IFilter<TProjection>
        , ISorter<TProjection>

    where TEntity : class
```

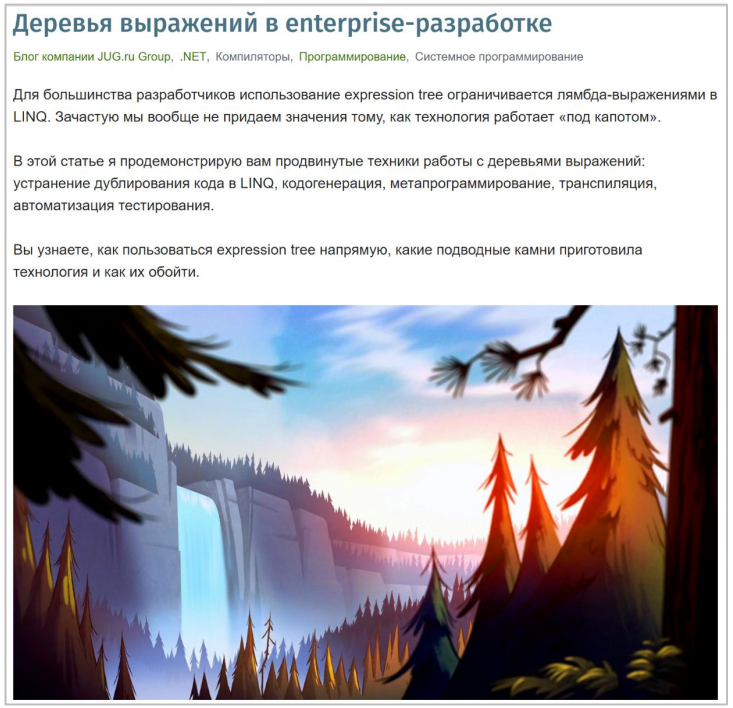
```
public class LinqQueryHandler<TQuery, TEntity, TProjection>
    : IQueryHandler<TQuery, IEnumerable<TProjection>>

    where TQuery
        : IQuery<IEnumerable<TProjection>>
        , IFilter<TProjection>
        , ISorter<TProjection>

    where TEntity : class
```

```
public IEnumerable<TProjection> Handle(TQuery query)
        => _entities
            .TryFilter(query)
            .ProjectTo<TProjection>()
            .FilterAndSort(query)
            .ToList();
```

```
public IEnumerable<TProjection> Handle(TQuery query)
        => _entities
            .TryFilter(query)
            .ProjectTo<TProjection>()
            .FilterAndSort(query)
            .ToList();
```

```
public IEnumerable<TProjection> Handle(TQuery query)
        => _entities
            .TryFilter(query)
            .ProjectTo<TProjection>()
            .FilterAndSort(query)
            .ToList();
```

```
public IEnumerable<TProjection> Handle(TQuery query)
        => _entities
            .TryFilter(query)
            .ProjectTo<TProjection>()
            .FilterAndSort(query)
            .ToList();
```



Не все выражения одинаково ~~полезны~~ транслируются в SQL

```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```

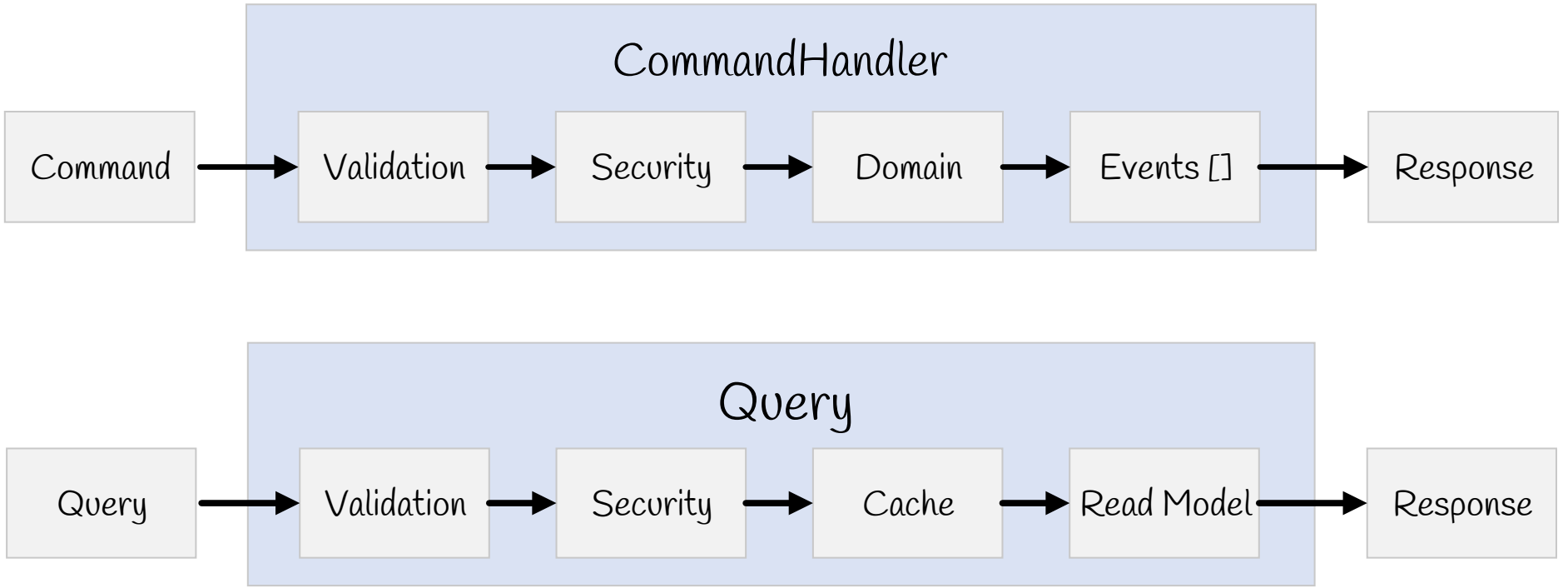
```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```

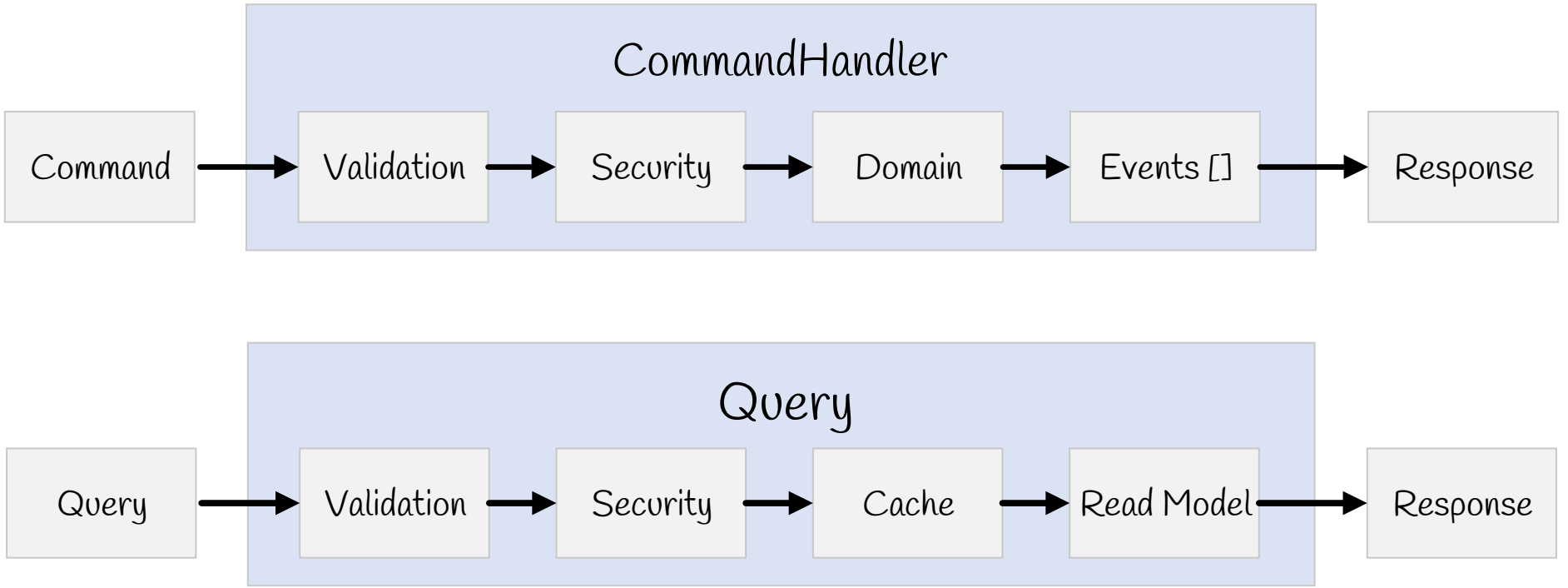
```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```

```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```

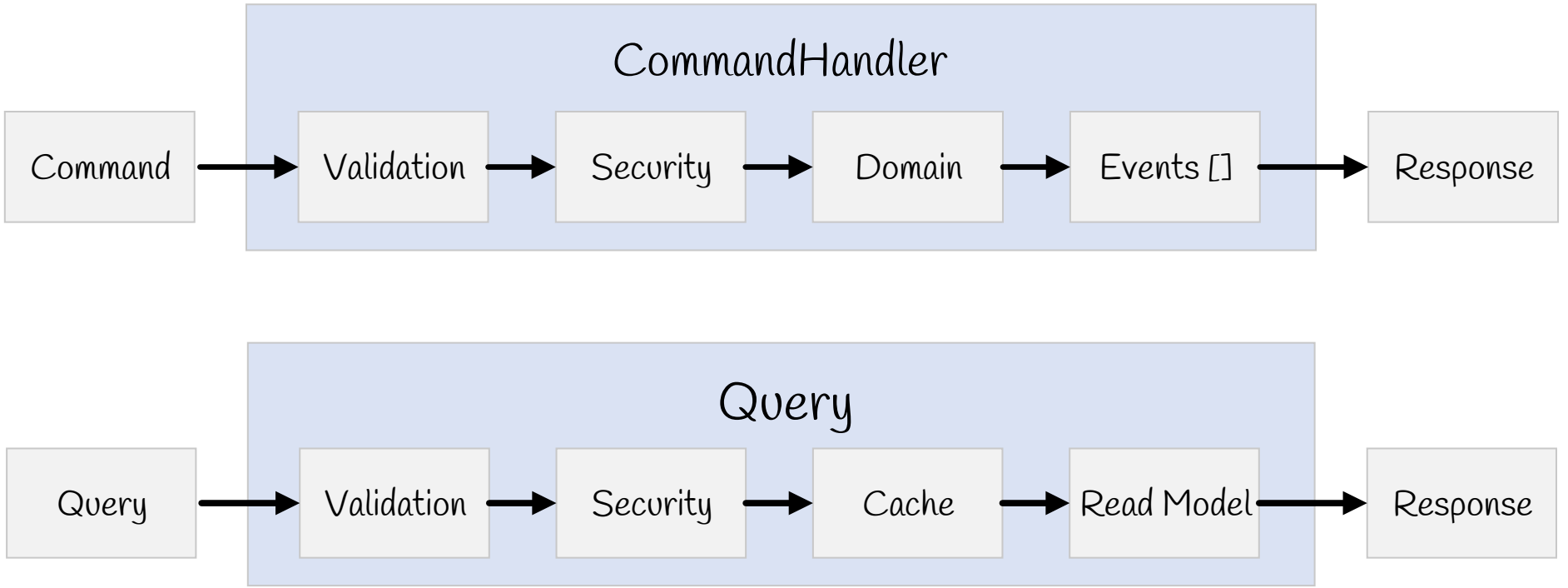
```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```

```
public class PostListDto : HasIdBase
{
    public string Title { get; set; }

    [JsonIgnore]
    public DateTime Created { get; set; }

    [JsonIgnore]
    public DateTime? LastUpdated { get; set; }

    public string SubTitle => LastUpdated.HasValue
        ? $"{Created.ToString(DateTimeFormats.Default)} /" +
{LastUpdated.Value.ToString(DateTimeFormats.Default)}"
        : Created.ToString(DateTimeFormats.Default);

    public static void CreateMap(IMappingExpression<Post, PostListDto> mappingExpression)
        => mappingExpression
            .ForMember(x => x.Title, o => o.MapFrom(x => $"{x.Hub.Name} / {x.Name}"));
}
```


CommandHandler
Command
Validation
Security
Domain
Events []
Response
Query
Query
Validation
Security
Cache
Read Model
Response



- Разные pipeline – разный набор декораторов



- Разные pipeline – разный набор декораторов
- Декораторы можно оформить в виде отдельных библиотек

Регистрация декораторов

```
var updateEmailHandler =
    new SaveChangesDecorator<UpdateUserEmail, Unit>(
        new ProfilerDecorator<UpdateUserEmail, Unit>(
            new LoggerDecorator<UpdateUserEmail, Unit>(
                new UpdateEmailHandler(
                    dbContext.Set<User>()),
                    logger)),
    dbContext, dispatcher);
```

Не айс ☹

```
var updateEmailHandler =
    new SaveChangesDecorator<UpdateUserEmail, Unit>(
        new ProfilerDecorator<UpdateUserEmail, Unit>(
            new LoggerDecorator<UpdateUserEmail, Unit>(
                new UpdateEmailHandler(
                    dbContext.Set<User>()),
                    logger)),
    dbContext, dispatcher);
```


MediatR

MediatR

Simple Injector

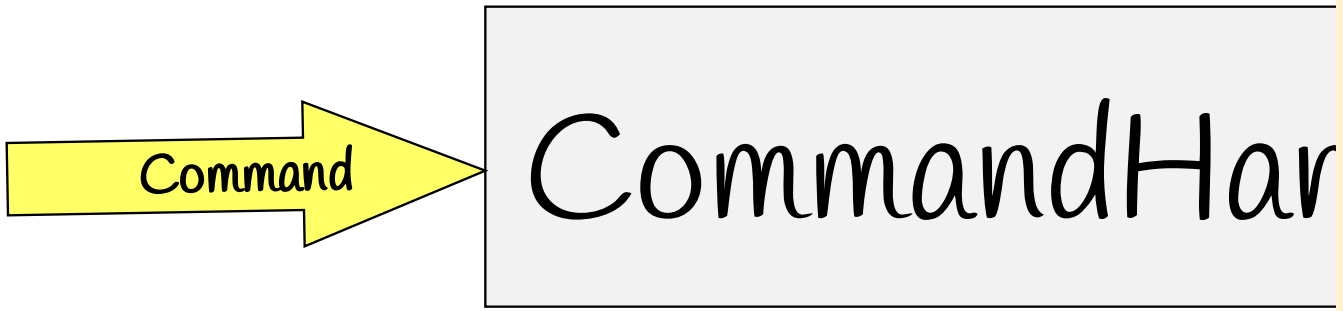
- Помните этот слайд?

```
public interface ICommandHandler<in TIn, out TOut>
    : IHandler<TIn, TOut>
    where TIn: ICommand<TOut>
```

```
public interface ICommandHandl
    : IHandler<TIn, TOut>
    where TIn: ICommand<TOut>
```

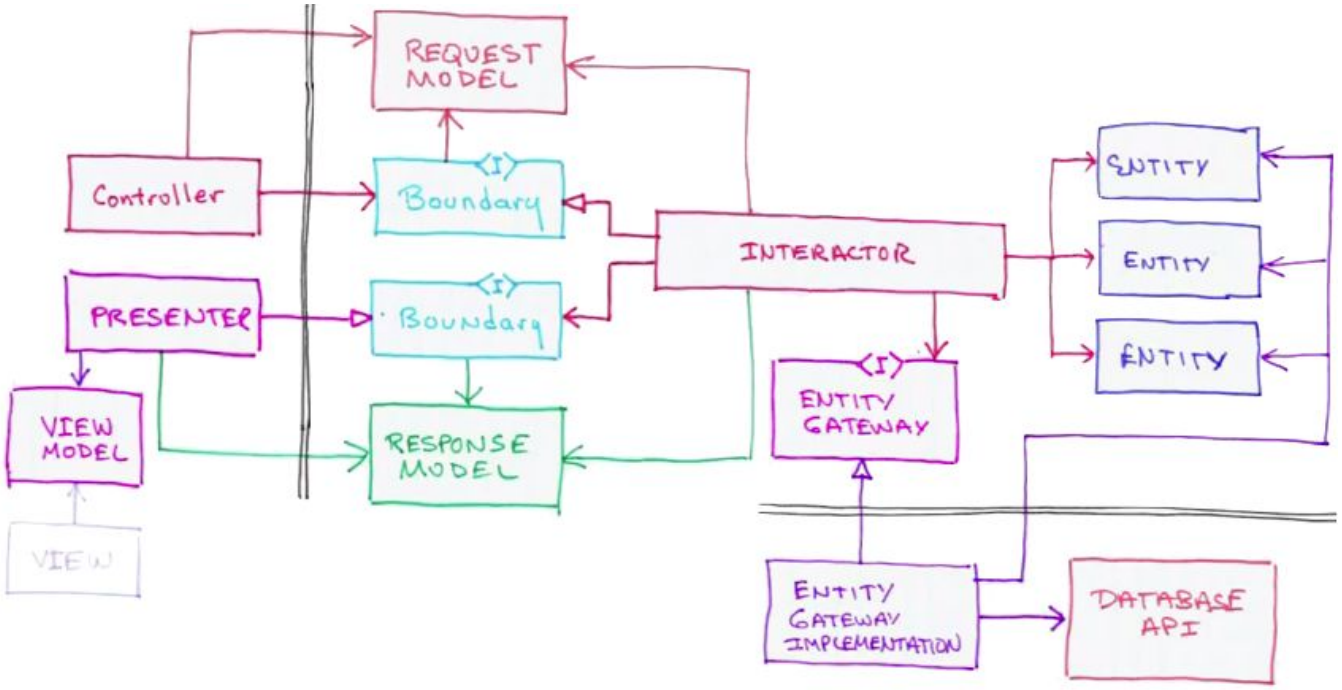
- Помните этот слайд?
- Simple Injector поддерживает ограничения типов в дженерика

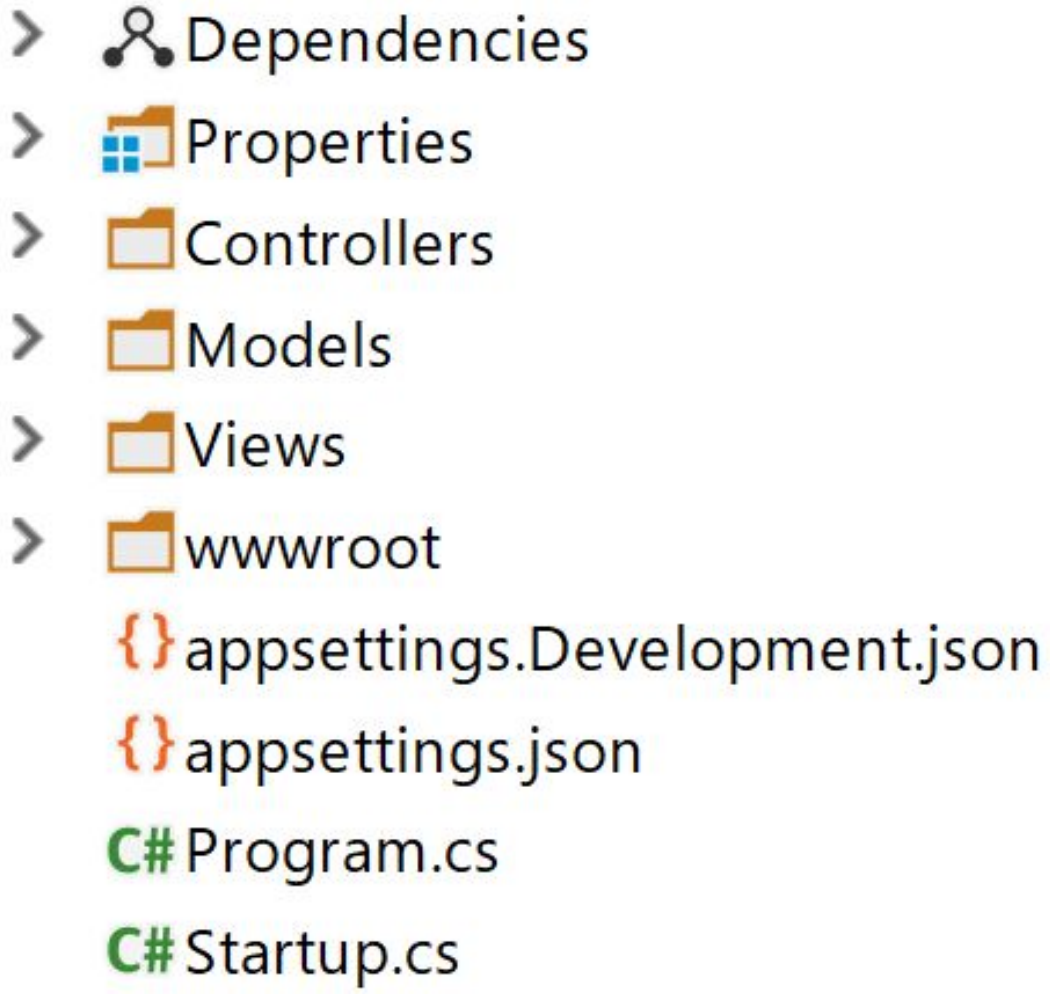
```
public interface ICommandHandl
    : IHandler<TIn, TOut>
    where TIn: ICommand<TOut>
```

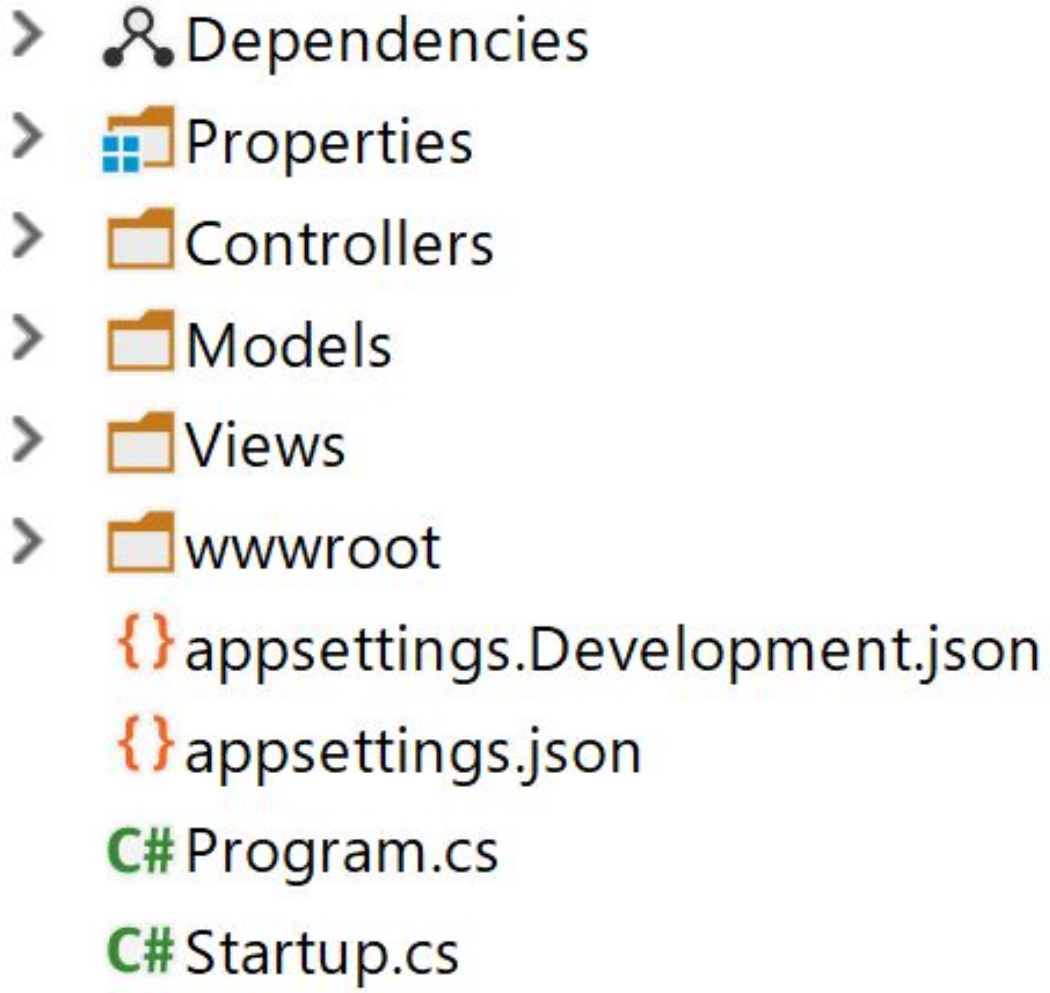


- Помните этот слайд?
- Simple Injector поддерживает ограничения типов в дженерика
- CommandHandler закрывает транзакцию, а Handler - нет

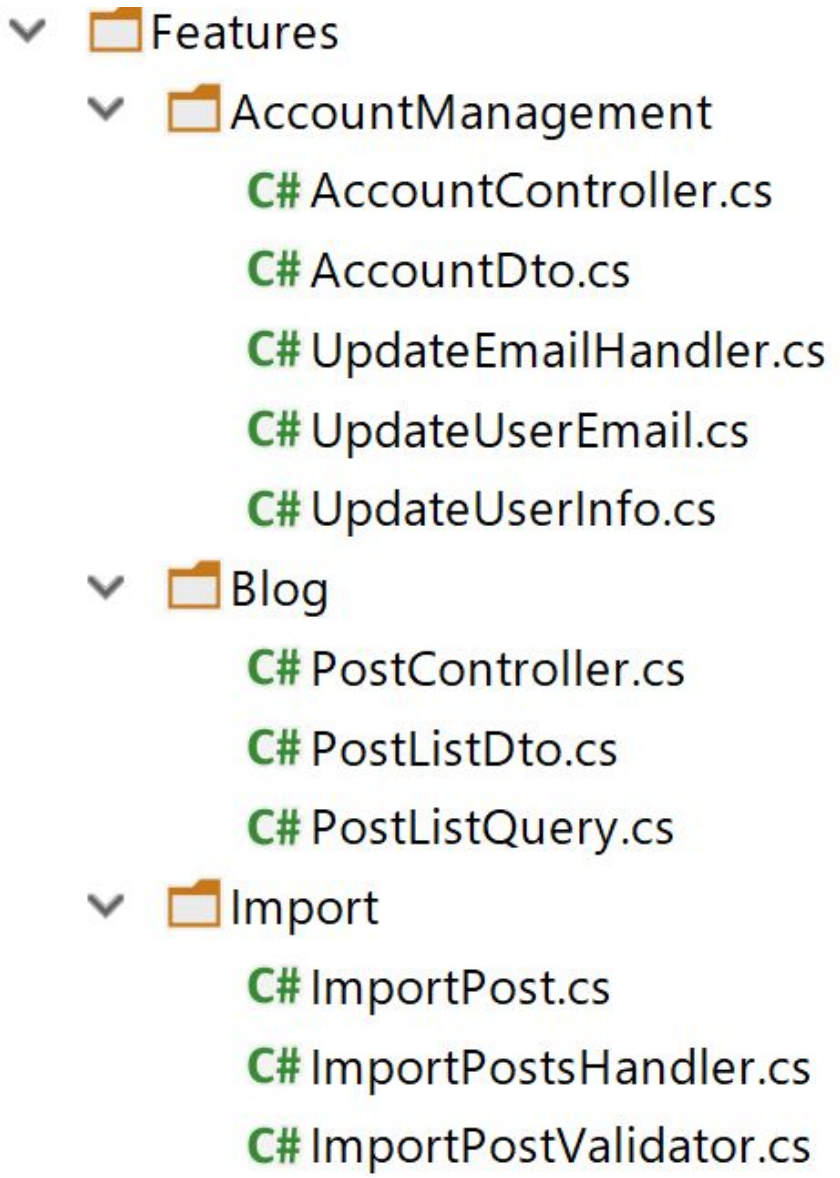
Организация по модулям, а не слоям


REQUEST MODEL
Controller
<I>
Boundary
INTERACTOR
ENTITY
ENTITY
ENTITY
PRESENTER
<I>
Boundary
VIEW MODEL
<I>
ENTITY GATEWAY
RESPONSE MODEL
VIEW
ENTITY GATEWAY IMPLEMENTATION
DATABASE API

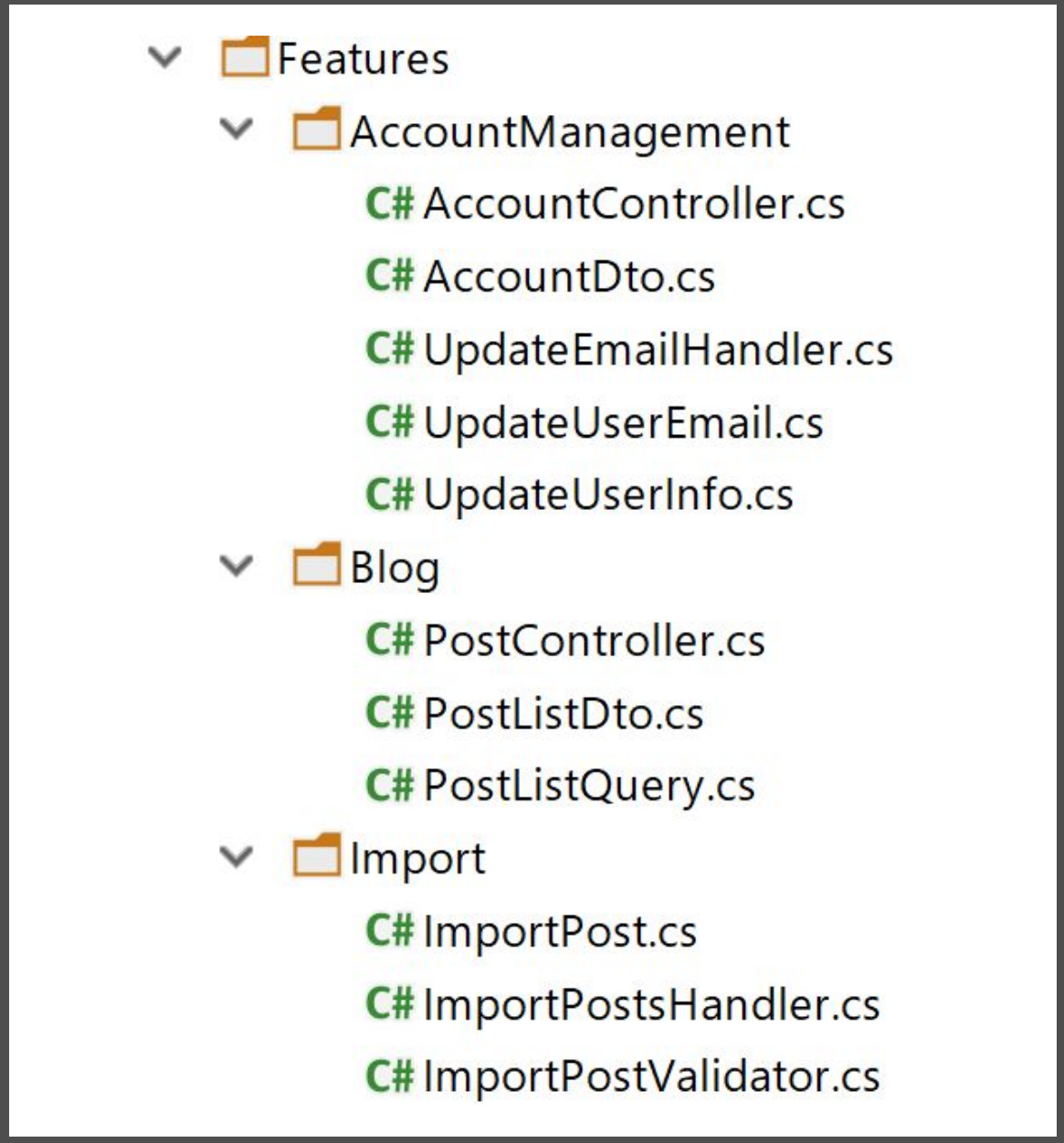

Dependencies
Properties
Controllers
Models
Views
wwwroot
appsettings.Development.json
appsettings.json
Program.cs
Startup.cs

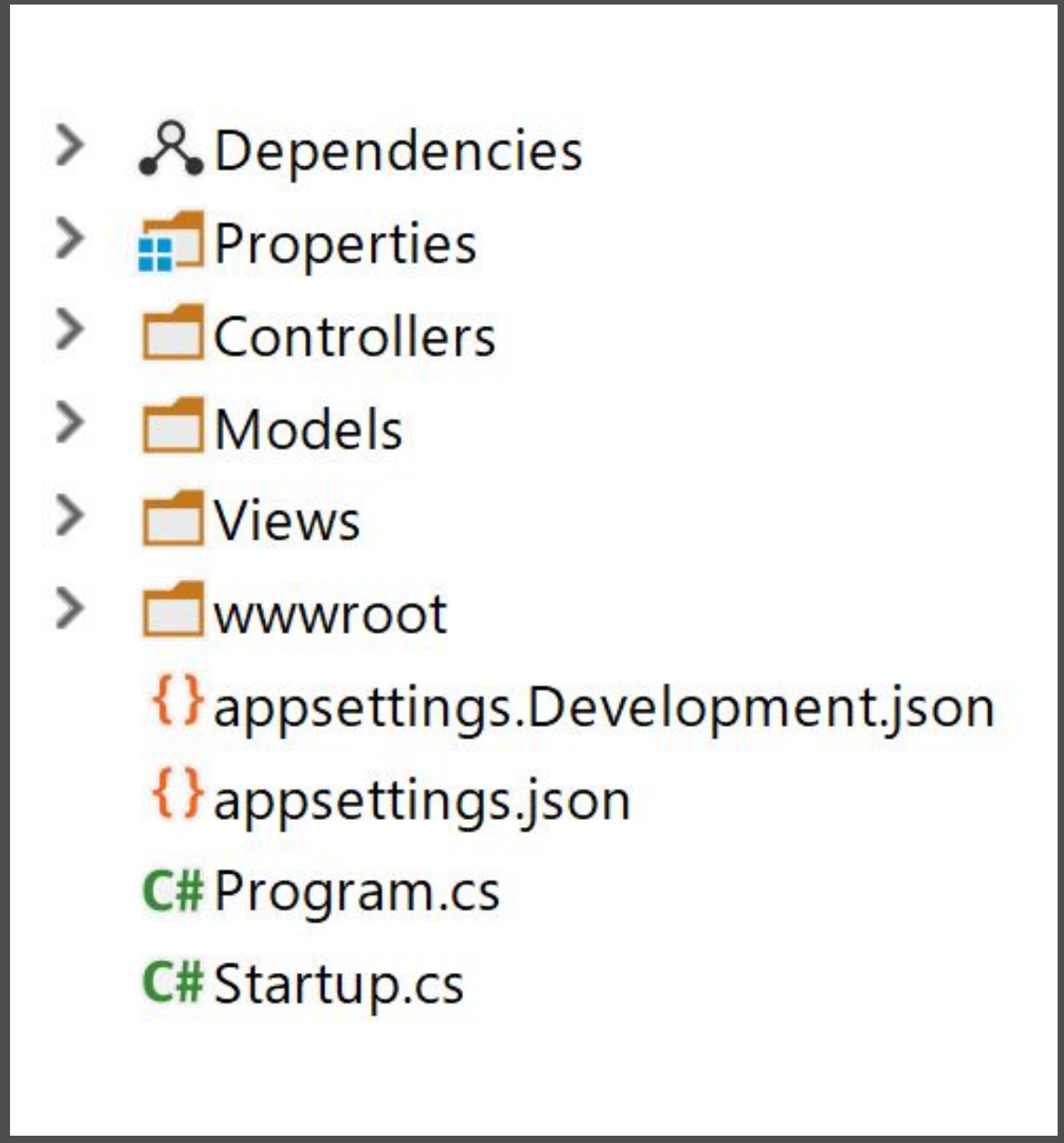


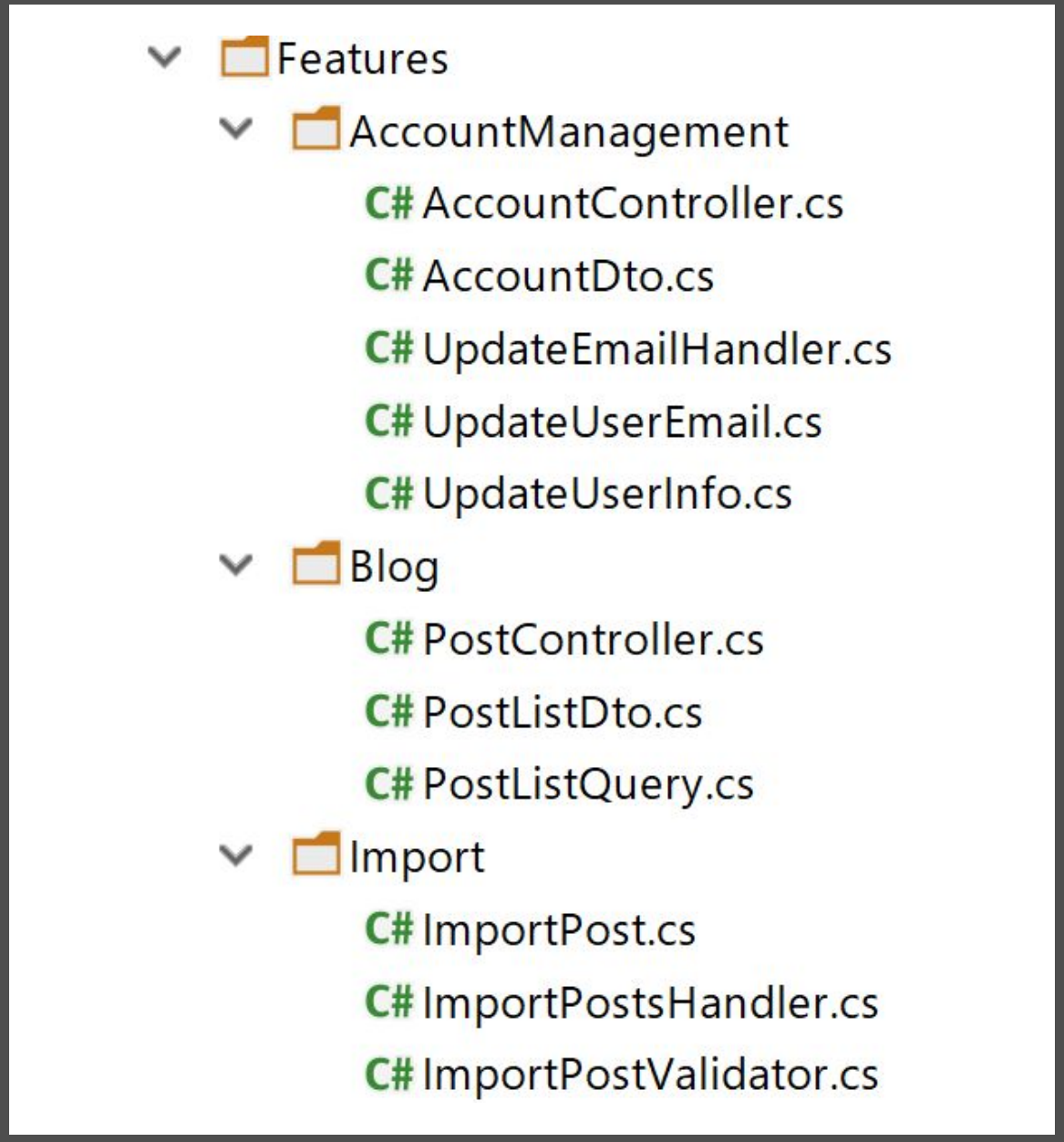
Фреймворк диктует структуру

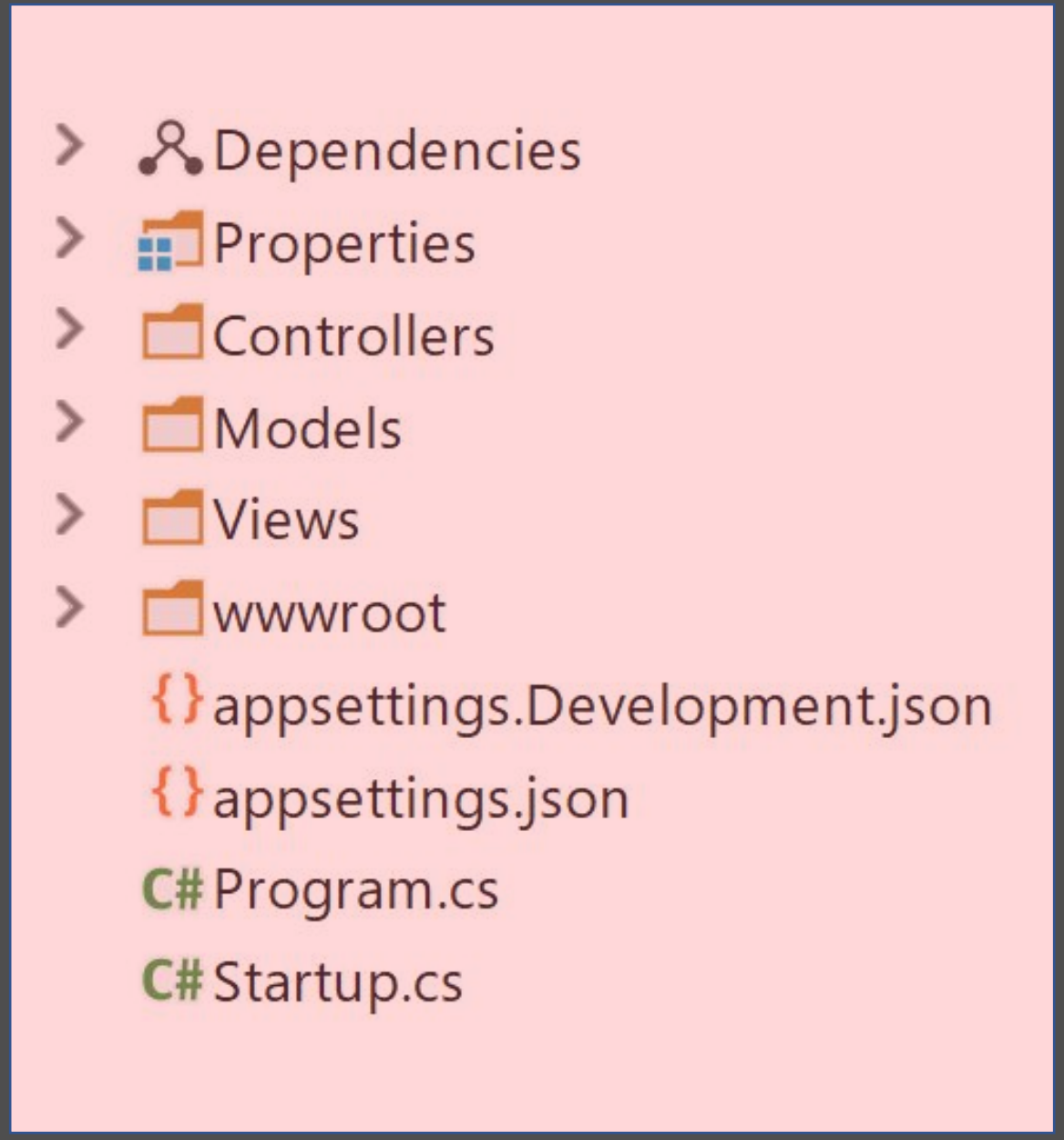

Features
AccountManagement
AccountController.cs
AccountDto.cs
UpdateEmailHandler.cs
UpdateUserEmail.cs
UpdateUserInfo.cs
Blog
PostController.cs
PostListDto.cs
PostListQuery.cs
Import
ImportPost.cs
ImportPostsHandler.cs
ImportPostValidator.cs

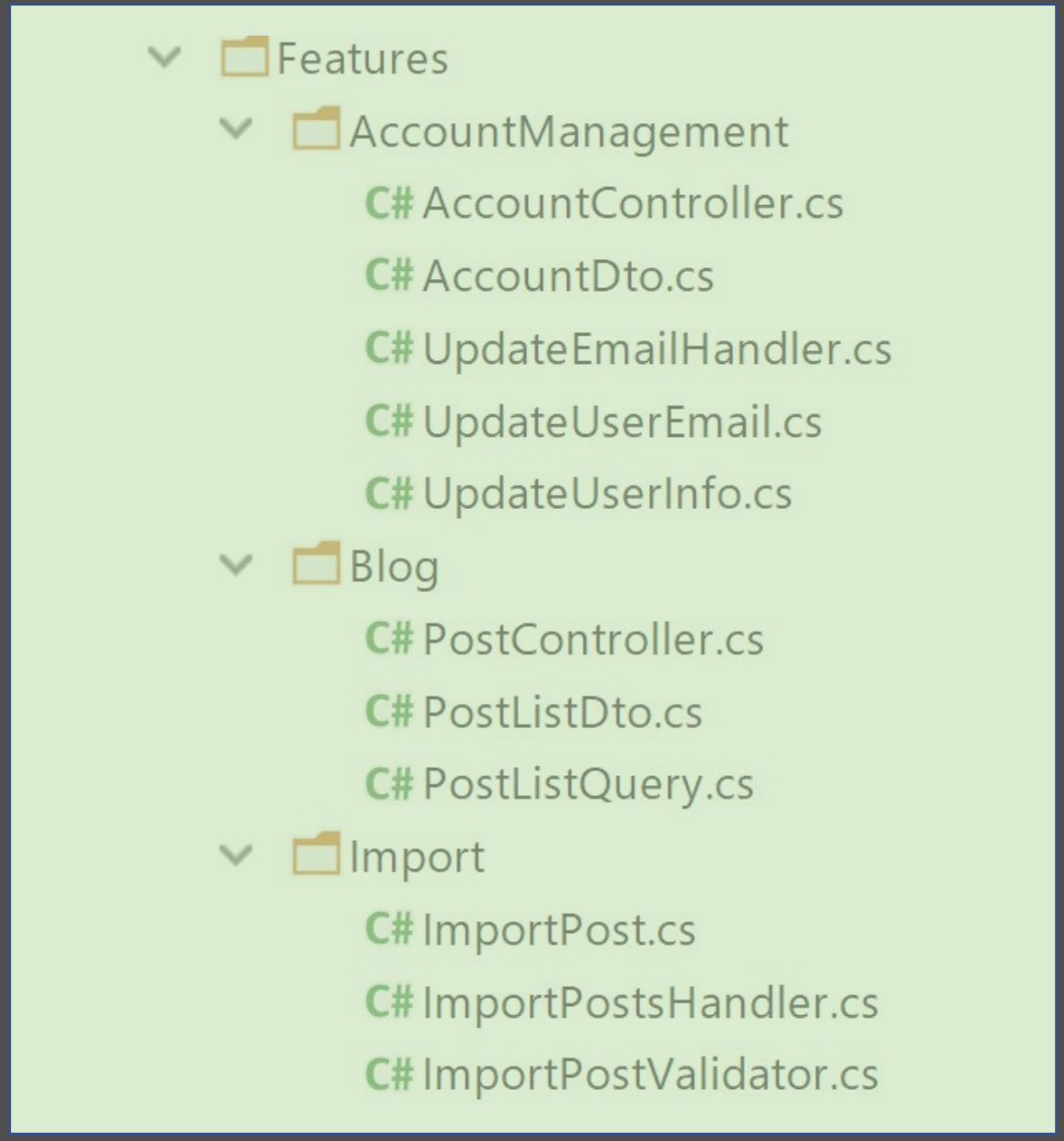
Структура соответствует функциональности системы




Dependencies
Properties
Controllers
Models
Views
wwwroot
appsettings.Development.json
appsettings.json
Program.cs
Startup.cs


Features
AccountManagement
AccountController.cs
AccountDto.cs
UpdateEmailHandler.cs
UpdateUserEmail.cs
UpdateUserInfo.cs
Blog
PostController.cs
PostListDto.cs
PostListQuery.cs
Import
ImportPost.cs
ImportPostsHandler.cs
ImportPostValidator.cs


Dependencies
Properties
Controllers
Models
Views
wwwroot
appsettings.Development.json
appsettings.json
Program.cs
Startup.cs


Features
AccountManagement
AccountController.cs
AccountDto.cs
UpdateEmailHandler.cs
UpdateUserEmail.cs
UpdateUserInfo.cs
Blog
PostController.cs
PostListDto.cs
PostListQuery.cs
Import
ImportPost.cs
ImportPostsHandler.cs
ImportPostValidator.cs

Преимущества

- Код добавляется, а не редактируется
- Меньше конфликтов в VCS

Преимущества

- Код добавляется, а не редактируется
- Меньше конфликтов в VCS
- Лучшее разделение на Bounded Context
- Фичу можно удалить целиком, удалив папку

Преимущества

- Код добавляется, а не редактируется
- Меньше конфликтов в VCS
- Лучшее разделение на Bounded Context
- Фичу можно удалить целиком, удалив папку
- Упрощает работу численными методами
- Упрощает коммуникацию

Недостатки

- Не работает из коробки
- Нужно писать много инфраструктурного кода и переопределить стандартное поведение

Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции

Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка

Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка
- Система типов + Инварианты > Валидация (но не на границах)
- Ждем C#8, а пока пишем гарды на NRE

Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка
- Система типов + Инварианты > Валидация (но не на границах)
- Ждем C#8, а пока пишем гарды на NRE
- События в рамках транзакции

Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка
- Система типов + Инварианты > Валидация (но не на границах)
- Ждем C#8, а пока пишем гарды на NRE
- События в рамках транзакции
- Exception’ы – для ошибок (либо писать на F#)

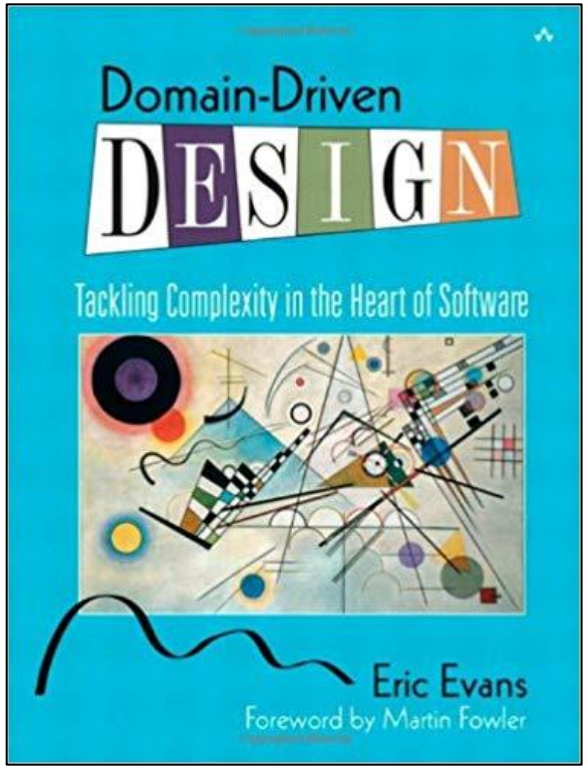
Все вместе

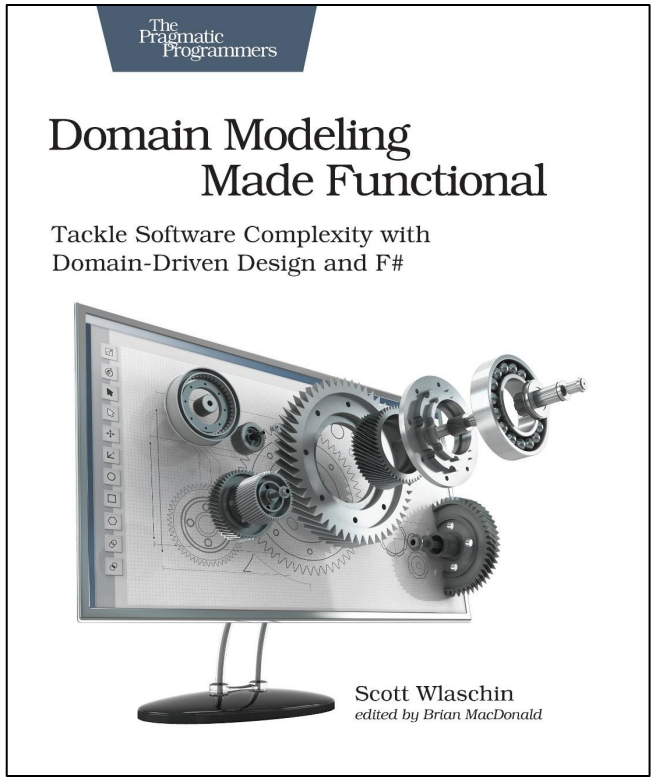
- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка
- Система типов + Инварианты > Валидация (но не на границах)
- Ждем C#8, а пока пишем гарды на NRE
- События в рамках транзакции
- Exception’ы – для ошибок (либо писать на F#)
- LINQ, ProjectTo, Permission Filters и Expressions – строительный блок, для получения данных

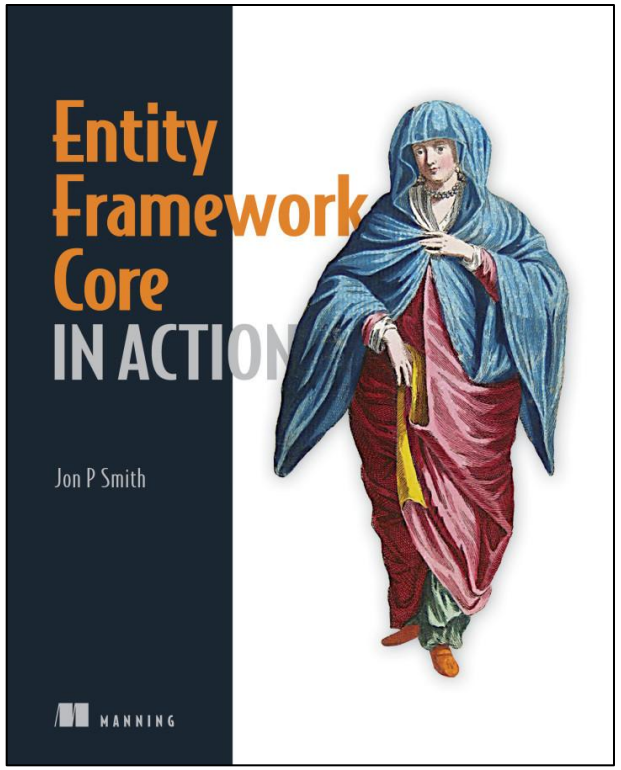
Все вместе

- IHandler<TIn, TOut> - строительный блок
- ICommandHandler, IQueryHandler – холистические абстракции
- Декораторы = separation of concerns. Регистрируем средствами контейнера или фреймворка
- Система типов + Инварианты > Валидация (но не на границах)
- Ждем C#8, а пока пишем гарды на NRE
- События в рамках транзакции
- Exception’ы – для ошибок (либо писать на F#)
- LINQ, ProjectTo, Permission Filters и Expressions – строительный блок, для получения данных
- By Feature > By Layer

- Vertical Slices
  - https://www.youtube.com/watch?v=SUiWfhAhgQw
  - https://www.cuttingedge.it/blogs/steven/pivot/entry.php?id=91
  - https://cuttingedge.it/blogs/steven/pivot/entry.php?id=92
- Одержимость примитивами
  - https://habr.com/post/266937/
  - https://habr.com/post/205088/
  - https://habr.com/post/205108/
- Domain Events
  - http://udidahan.com/2009/06/14/domain-events-salvation/
  - https://lostechies.com/jimmybogard/2014/05/13/a-better-domain-events-pattern/
  - https://enterprisecraftsmanship.com/2017/10/03/domain-events-simple-and-reliable-solution/
- DDD: https://habr.com/post/334126/
- ROP
  - https://fsharpforfunandprofit.com/rop/
  - https://habr.com/post/339606/
  - https://habr.com/post/347284/
- LINQ и Expressions: https://habr.com/company/jugru/blog/423891/
- Clean Architecture: https://www.youtube.com/watch?v=JEeEic-c0D4


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


The Pragmatic Programmers
Domain Modeling
Made Functional
Tackle Software Complexity with
Domain-Driven Design and F#
Scott Wlaschin
edited by Brian MacDonald


Entity
Framework
Core
IN ACTION
Jon P Smith
MANNING

Вопросы