
VS
Богатая
Анемичная

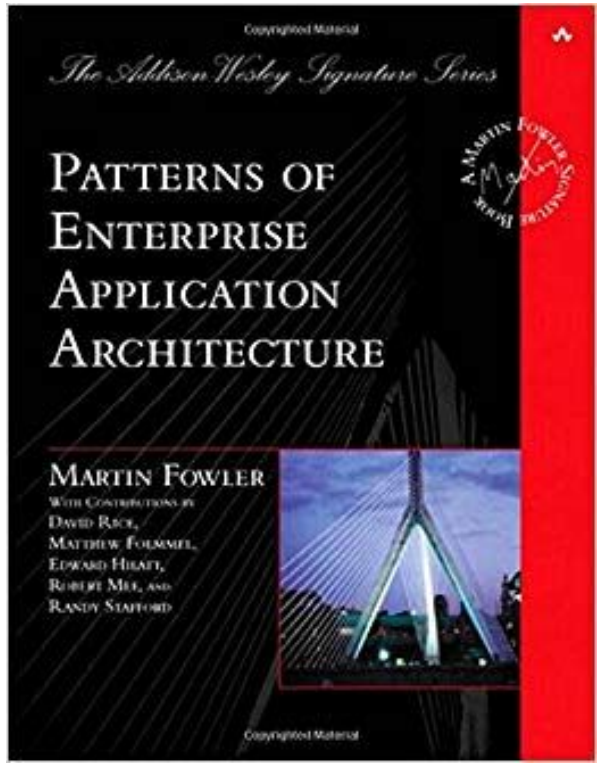
Максим Аршинов

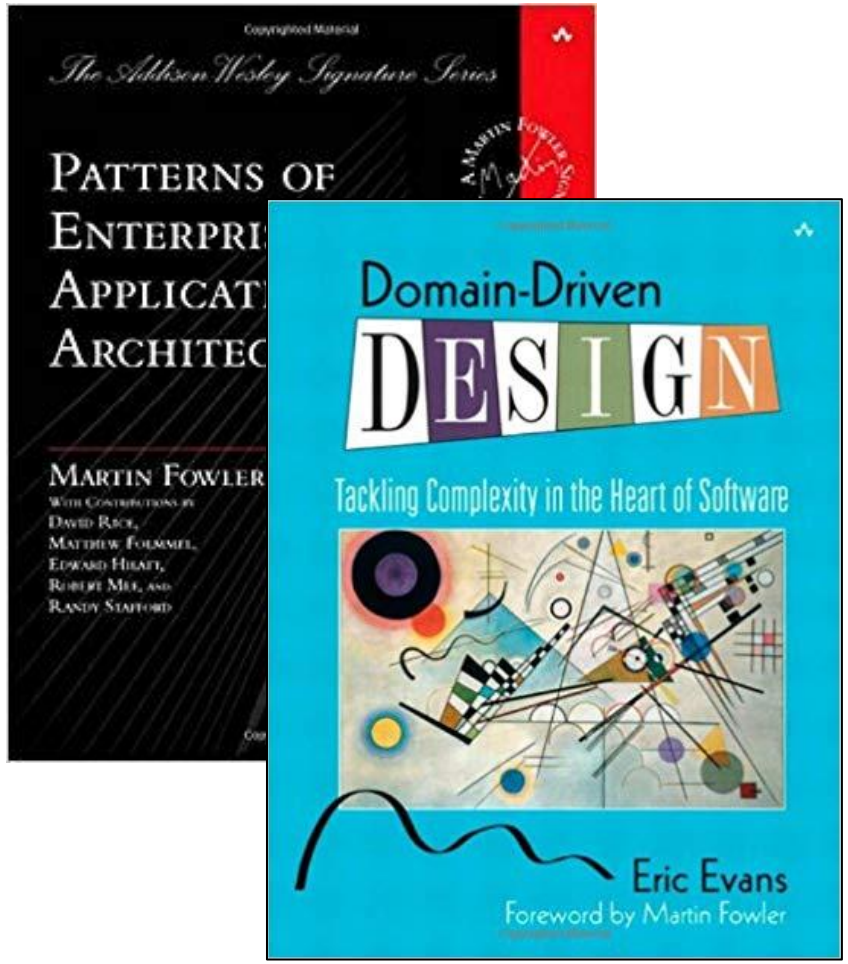
предприниматель, блогер, преподаватель

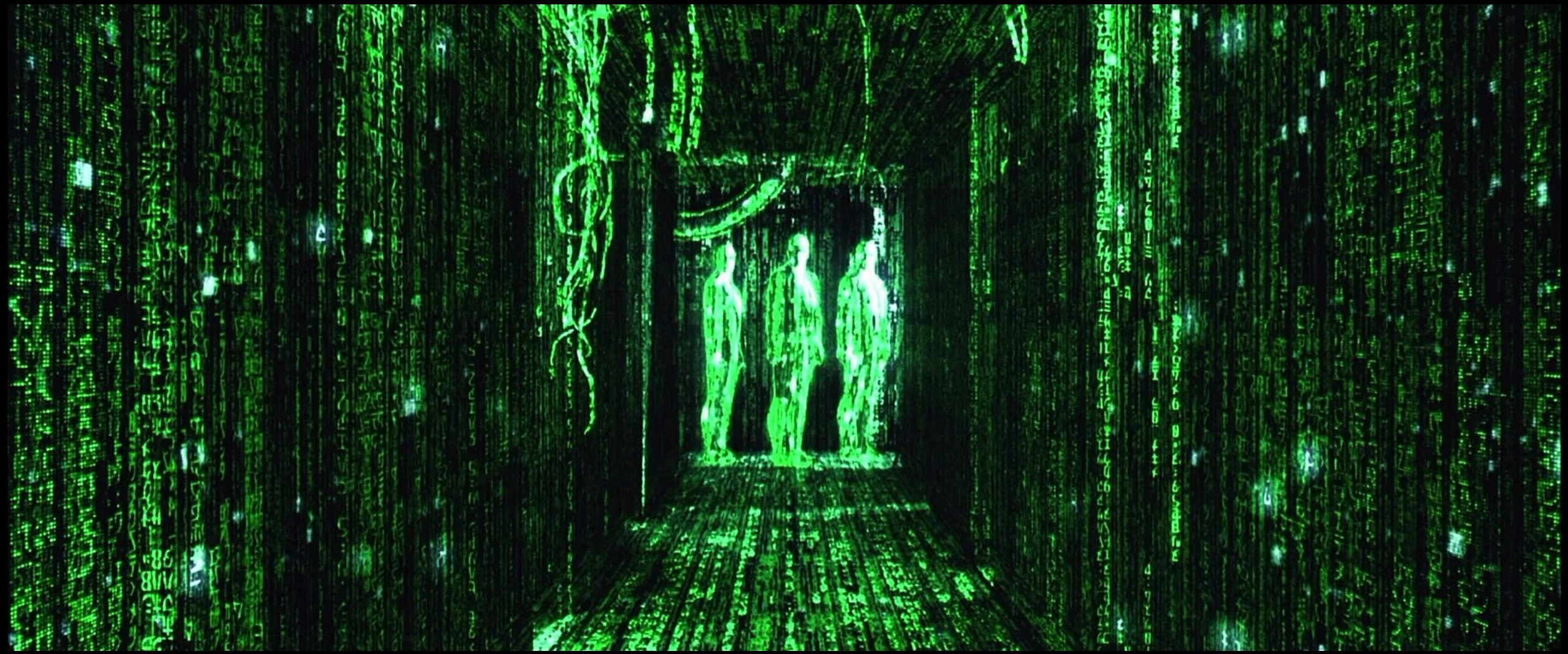
max@hightech.group
https://habr.com/users/marshinov

В докладе использованы картины Васи Ложкина

и кое-что еще...


Copyrighted Material
The Addison Wesley Signature Series
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
MARTIN FOWLER
WITH CONTRIBUTIONS FROM
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD
Copyrighted Material


Copyrighted Material
The Addison Wesley Signature Series
PATTERNS OF
ENTERPRI
APPLICAT
ARCHITEC
MARTIN FOWLER
Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler




Copyrighted Material
The Addison Wesley Signature Series
PATTERNS OF
ENTERPRI
APPLICAT
ARCHITEC
MARTIN FOWLER
WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD
Domain-Driven
DESIGN
Tackling Complexity in th
Foreword
DDD Community
IMPLEMENTING
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON
FOREWORD BY ERIC EVANS


Copyrighted Material
The Addison Wesley Signature Series
PATTERNS OF
ENTERPRI
APPLICAT
ARCHITEC
MARTIN FOWLER
Domain-Driven
DESIGN
Tackling Complexity in th
DDD Community
IMPLE
DOM
DE
VAUGH
The Pragmatic Programmers
Domain Modeling
Made Functional
Tackle Software Complexity with
Domain-Driven Design and F#
Scott Wlaschin
edited by Brian MacDonald


ЖЫЗНЬ С БОРОДОЙ

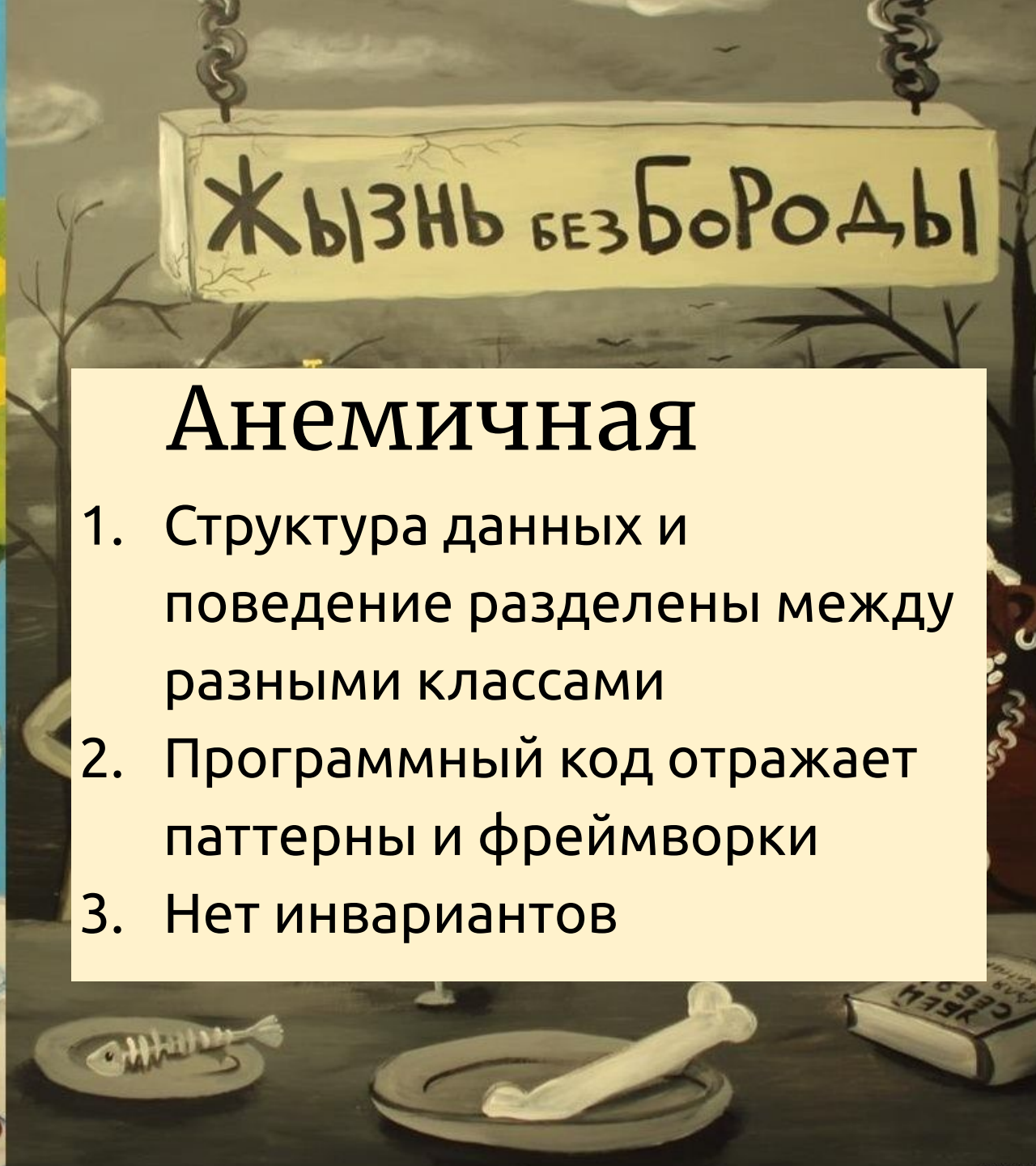

ЖЫЗНЬ БЕЗ БОРОДЫ

Богатая

1. Структура данных и поведение совмещены
2. Программный код отражает структуру домена
3. Соблюдение инварианта

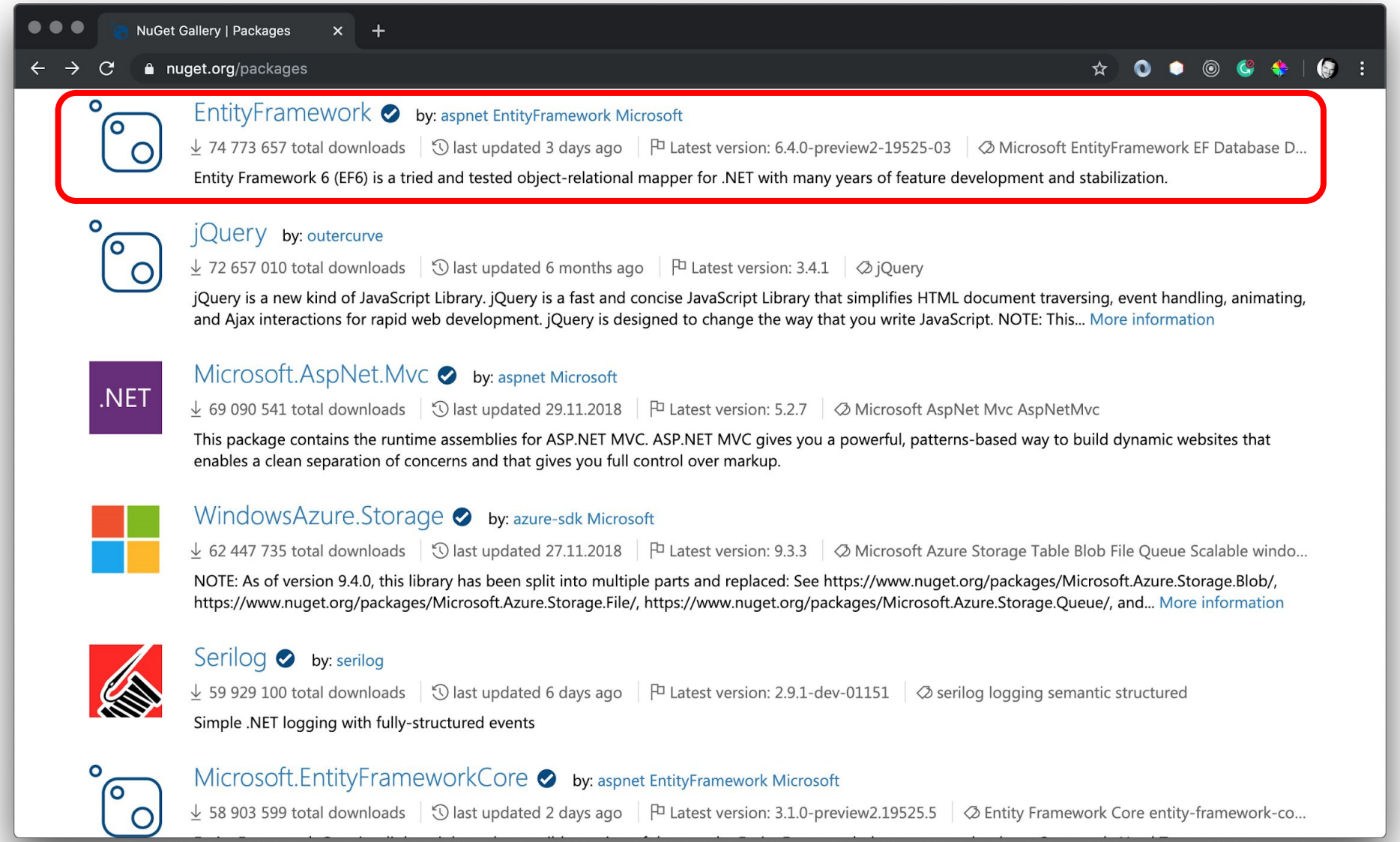
Богатая

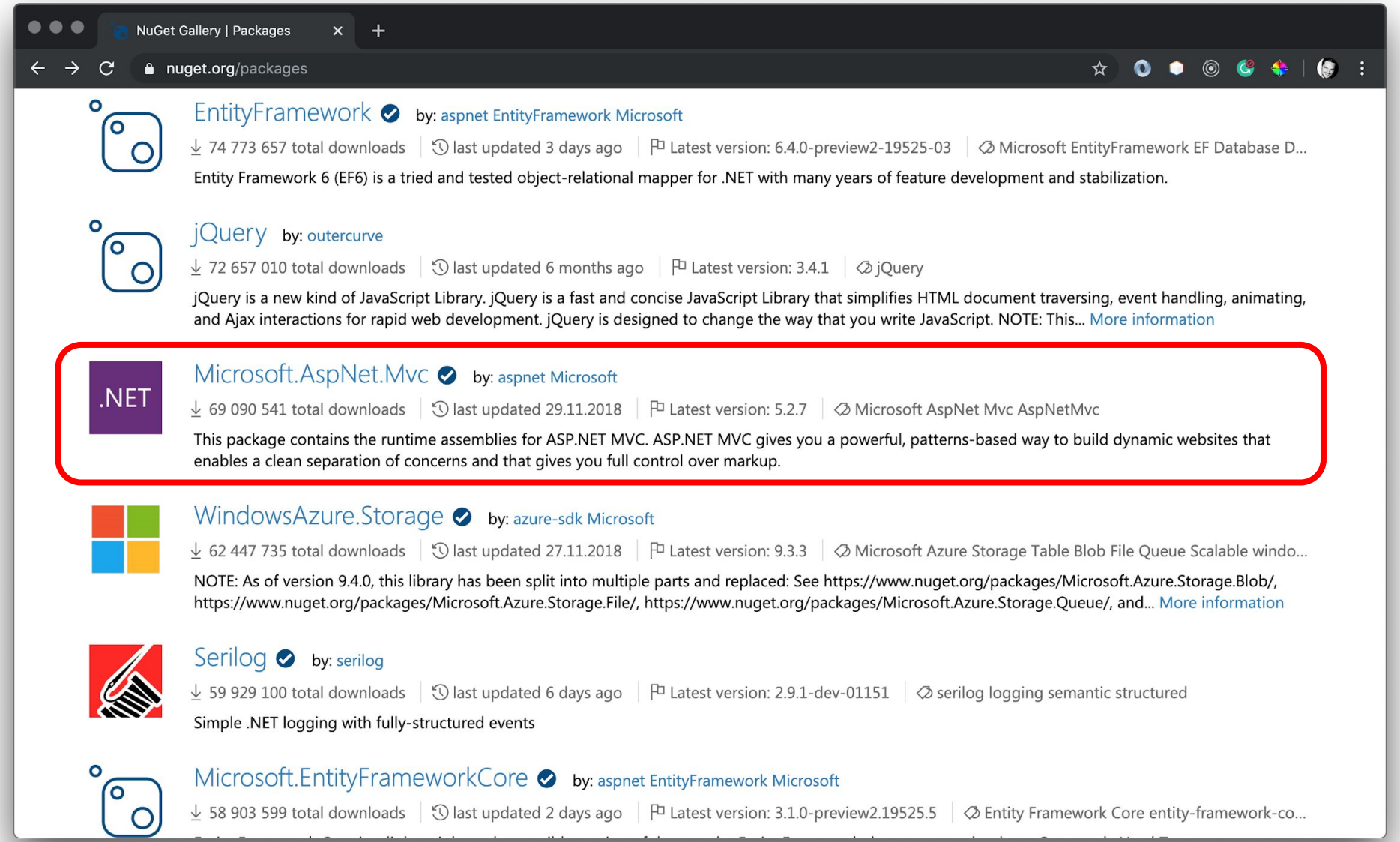
1. Структура данных и поведение совмещены
2. Программный код отражает структуру домена
3. Соблюдение инварианта

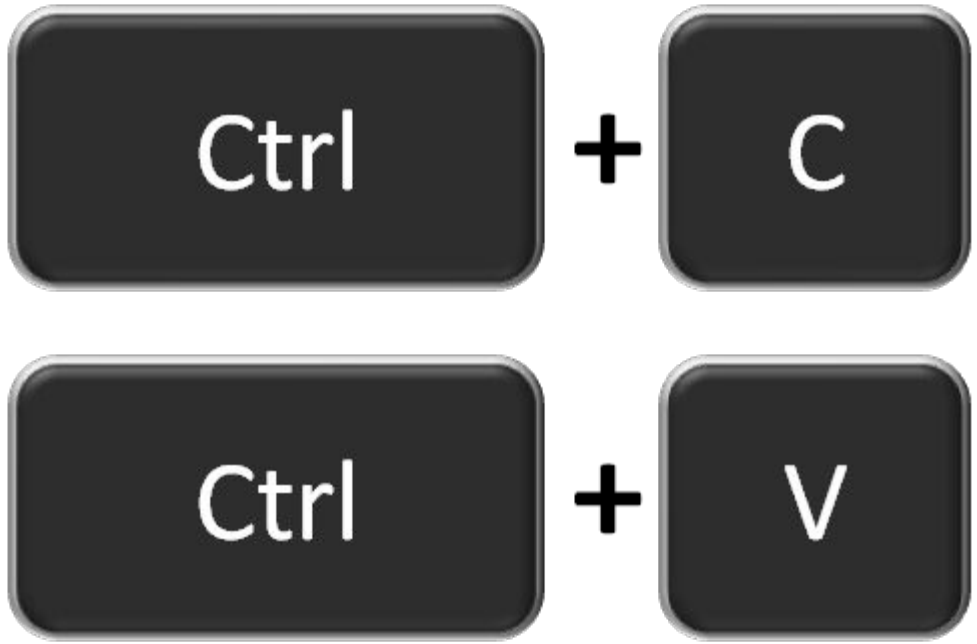


Анемичная

1. Структура данных и поведение разделены между разными классами
2. Программный код отражает паттерны и фреймворки
3. Нет инвариантов

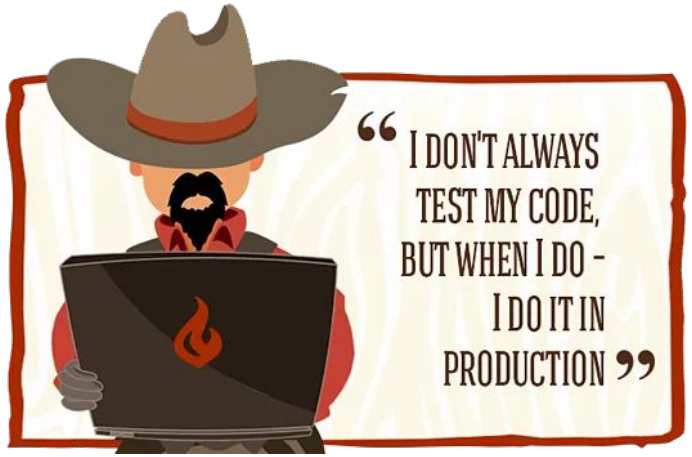

NuGet Gallery | Packages
nuget.org/packages
EntityFramework by: aspnet EntityFramework Microsoft
74 773 657 total downloads
last updated 3 days ago
Latest version: 6.4.0-preview2-19525-03
Microsoft EntityFramework EF Database D...
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many years of feature development and stabilization.
jQuery by: outercurve
72 657 010 total downloads
last updated 6 months ago
Latest version: 3.4.1
jQuery
jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript. NOTE: This... More information
.NET
Microsoft.AspNet.Mvc by: aspnet Microsoft
69 090 541 total downloads
last updated 29.11.2018
Latest version: 5.2.7
Microsoft AspNet Mvc AspNetMvc
This package contains the runtime assemblies for ASP.NET MVC. ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup.
WindowsAzure.Storage by: azure-sdk Microsoft
62 447 735 total downloads
last updated 27.11.2018
Latest version: 9.3.3
Microsoft Azure Storage Table Blob File Queue Scalable windo...
NOTE: As of version 9.4.0, this library has been split into multiple parts and replaced: See https://www.nuget.org/packages/Microsoft.Azure.Storage.Blob/, https://www.nuget.org/packages/Microsoft.Azure.Storage.File/, https://www.nuget.org/packages/Microsoft.Azure.Storage.Queue/, and... More information
Serilog by: serilog
59 929 100 total downloads
last updated 6 days ago
Latest version: 2.9.1-dev-01151
serilog logging semantic structured
Simple .NET logging with fully-structured events
Microsoft.EntityFrameworkCore by: aspnet EntityFramework Microsoft
58 903 599 total downloads
last updated 2 days ago
Latest version: 3.1.0-preview2.19525.5
Entity Framework Core entity-framework-co...


NuGet Gallery | Packages
nuget.org/packages
EntityFramework by: aspnet EntityFramework Microsoft
74 773 657 total downloads
last updated 3 days ago
Latest version: 6.4.0-preview2-19525-03
Microsoft EntityFramework EF Database D...
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many years of feature development and stabilization.
jQuery by: outercurve
72 657 010 total downloads
last updated 6 months ago
Latest version: 3.4.1
jQuery
jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript. NOTE: This... More information
.NET
Microsoft.AspNet.Mvc by: aspnet Microsoft
69 090 541 total downloads
last updated 29.11.2018
Latest version: 5.2.7
Microsoft AspNet Mvc AspNetMvc
This package contains the runtime assemblies for ASP.NET MVC. ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup.
WindowsAzure.Storage by: azure-sdk Microsoft
62 447 735 total downloads
last updated 27.11.2018
Latest version: 9.3.3
Microsoft Azure Storage Table Blob File Queue Scalable windo...
NOTE: As of version 9.4.0, this library has been split into multiple parts and replaced: See https://www.nuget.org/packages/Microsoft.Azure.Storage.Blob/, https://www.nuget.org/packages/Microsoft.Azure.Storage.File/, https://www.nuget.org/packages/Microsoft.Azure.Storage.Queue/, and... More information
Serilog by: serilog
59 929 100 total downloads
last updated 6 days ago
Latest version: 2.9.1-dev-01151
serilog logging semantic structured
Simple .NET logging with fully-structured events
Microsoft.EntityFrameworkCore by: aspnet EntityFramework Microsoft
58 903 599 total downloads
last updated 2 days ago
Latest version: 3.1.0-preview2.19525.5
Entity Framework Core entity-framework-co...


Ctrl
+
C
Ctrl
+
V

Always valid

миф или реальность?



Always valid

миф или реальность?

Always Valid

Always valid

миф или реальность?

Always Valid

Always valid

миф или реальность?

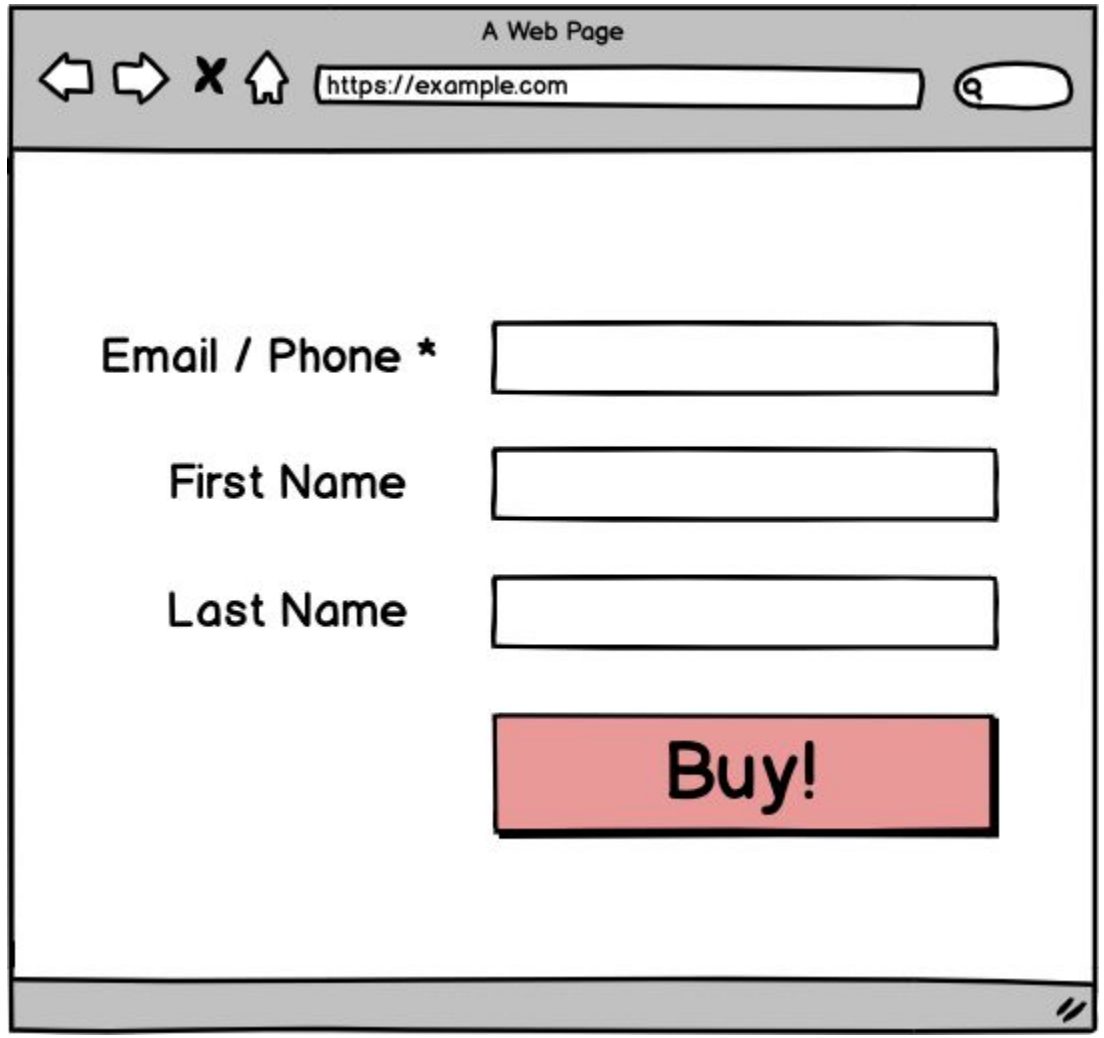
Fallacy!

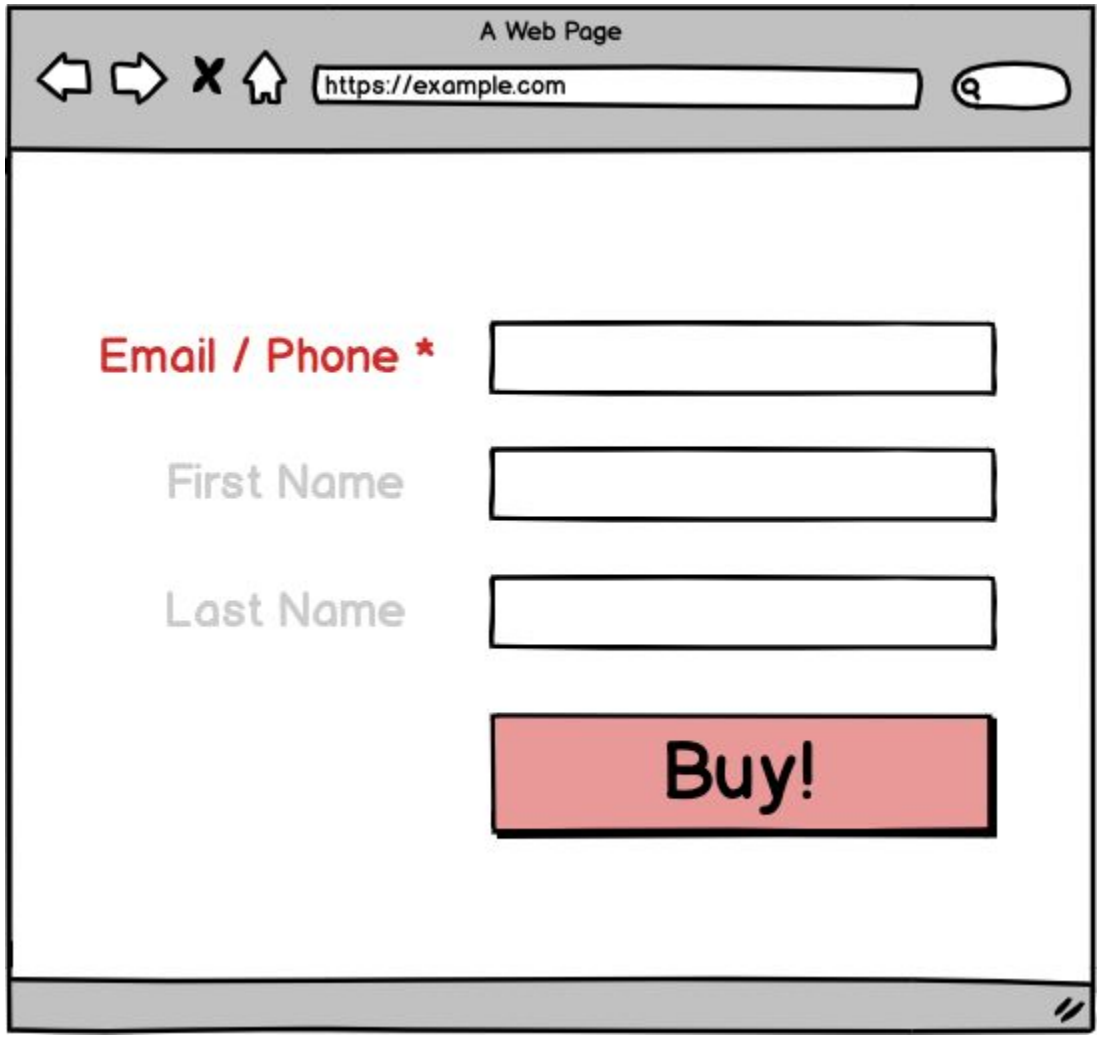
```
public string Name
{
    get { return _name; }
    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new Exception("Name is required");
        }
        _name = value;
    }
}
```

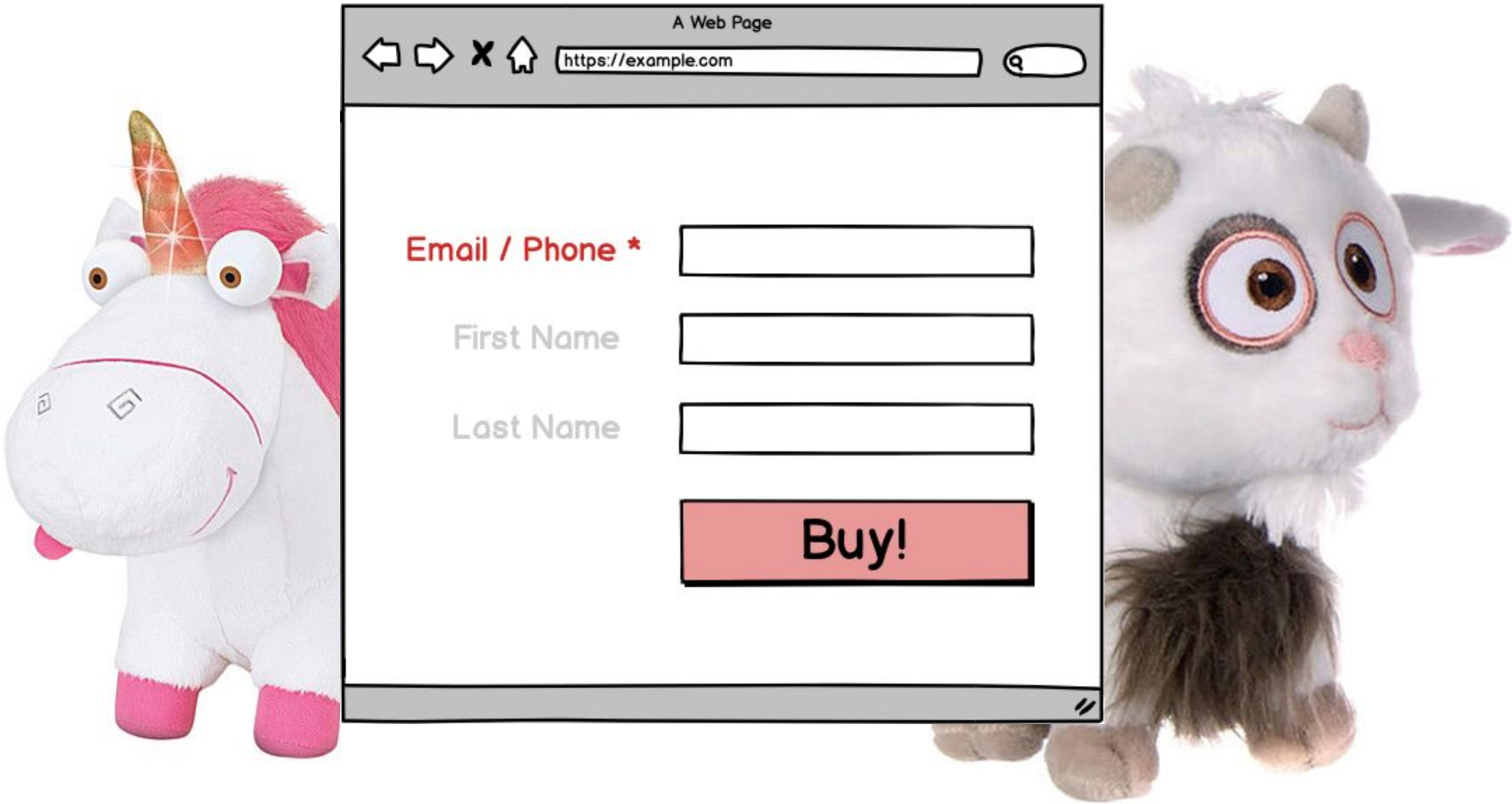
```
public string Name
{
    get { return _name; }
    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new Exception("Name is required");
        }
        _name = value;
    }
}
```

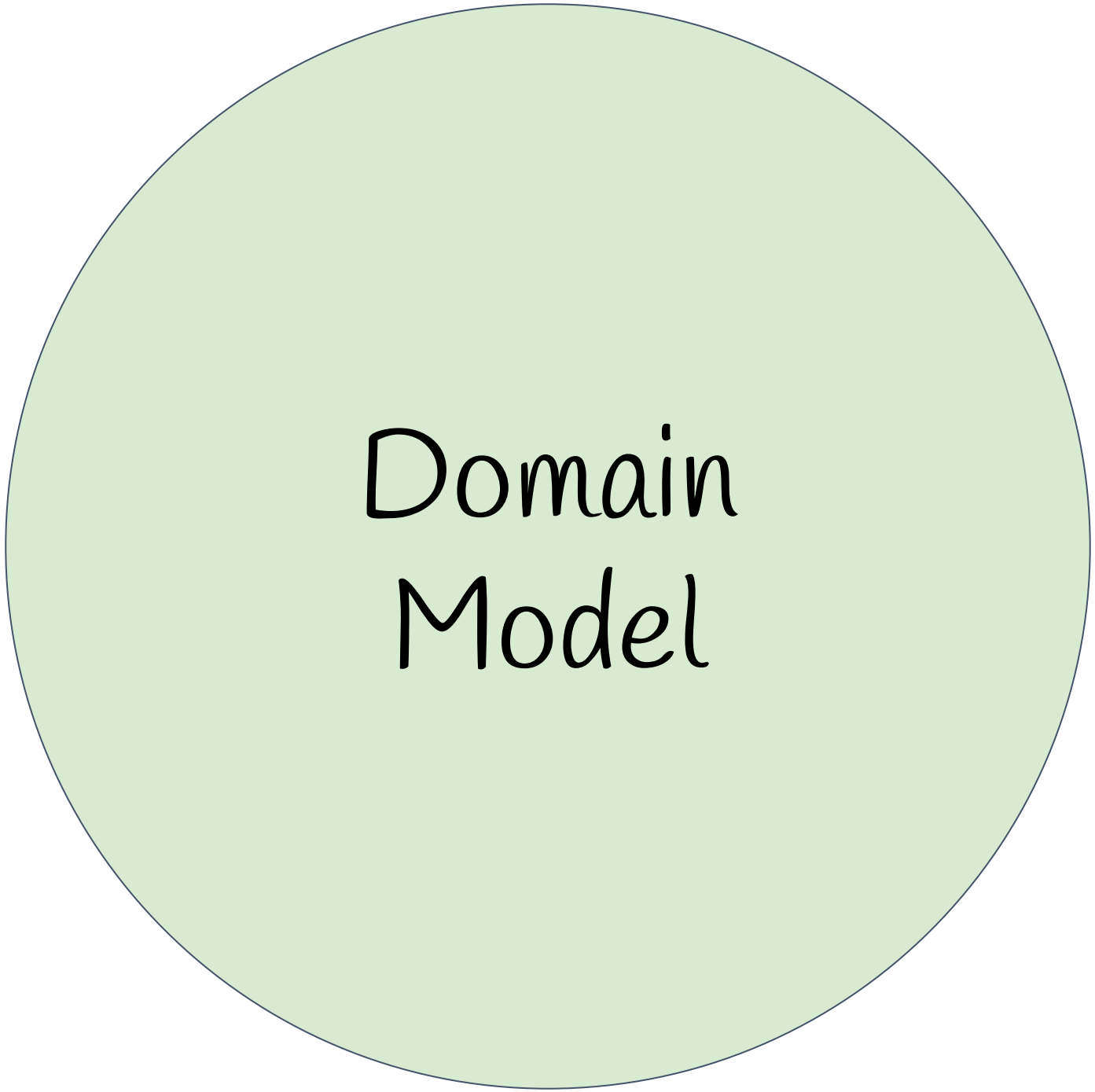
```
public interface IValidator<T>
{
    ValidationResult Validate(T obj);
}
```

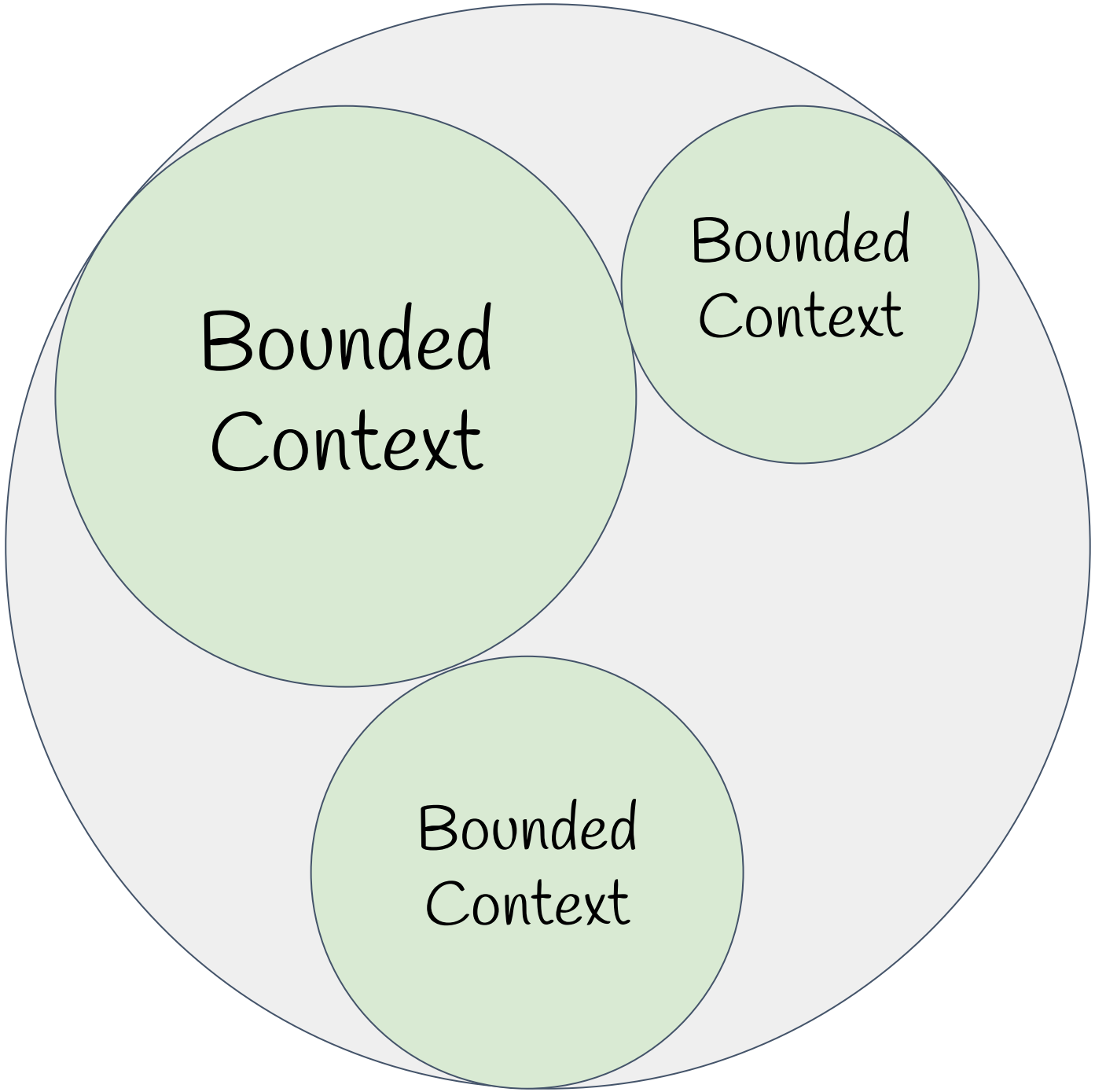
```
public interface IValidator<T>
{
    ValidationResult Validate(T obj);
}
```

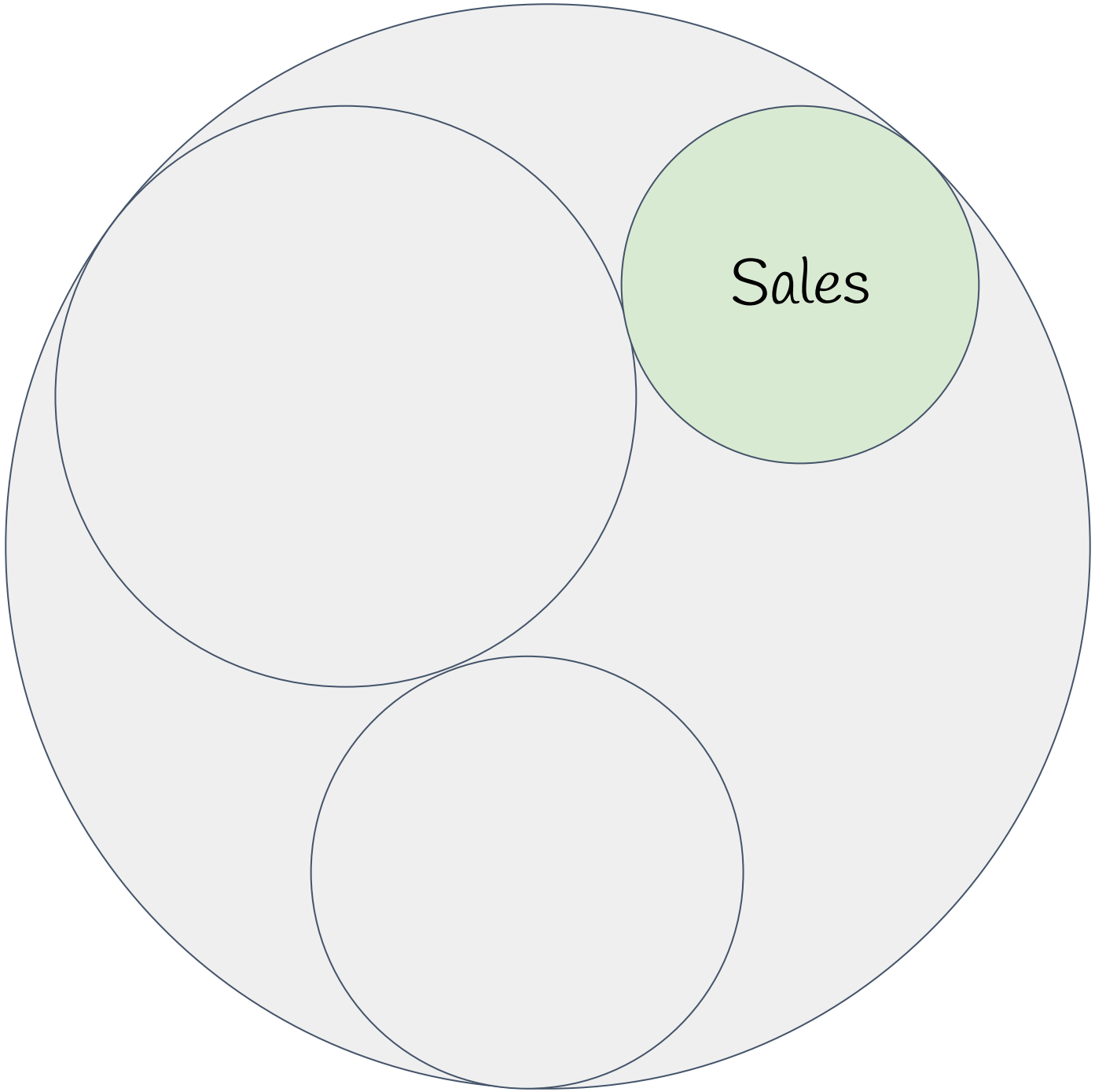

A Web Page
https://example.com
Email / Phone *
First Name
Last Name
Buy!

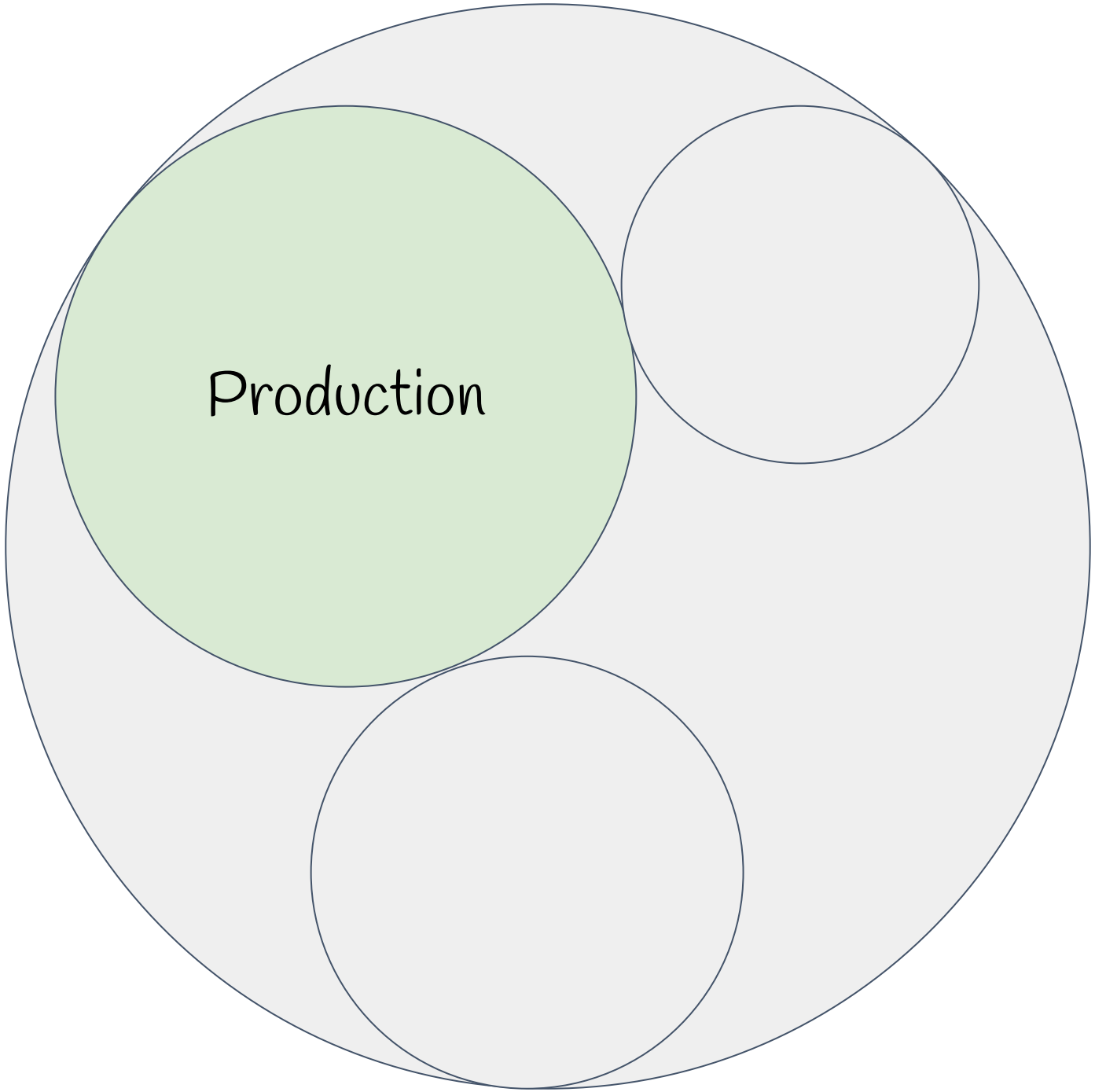

A Web Page
https://example.com
Email / Phone *
First Name
Last Name
Buy!

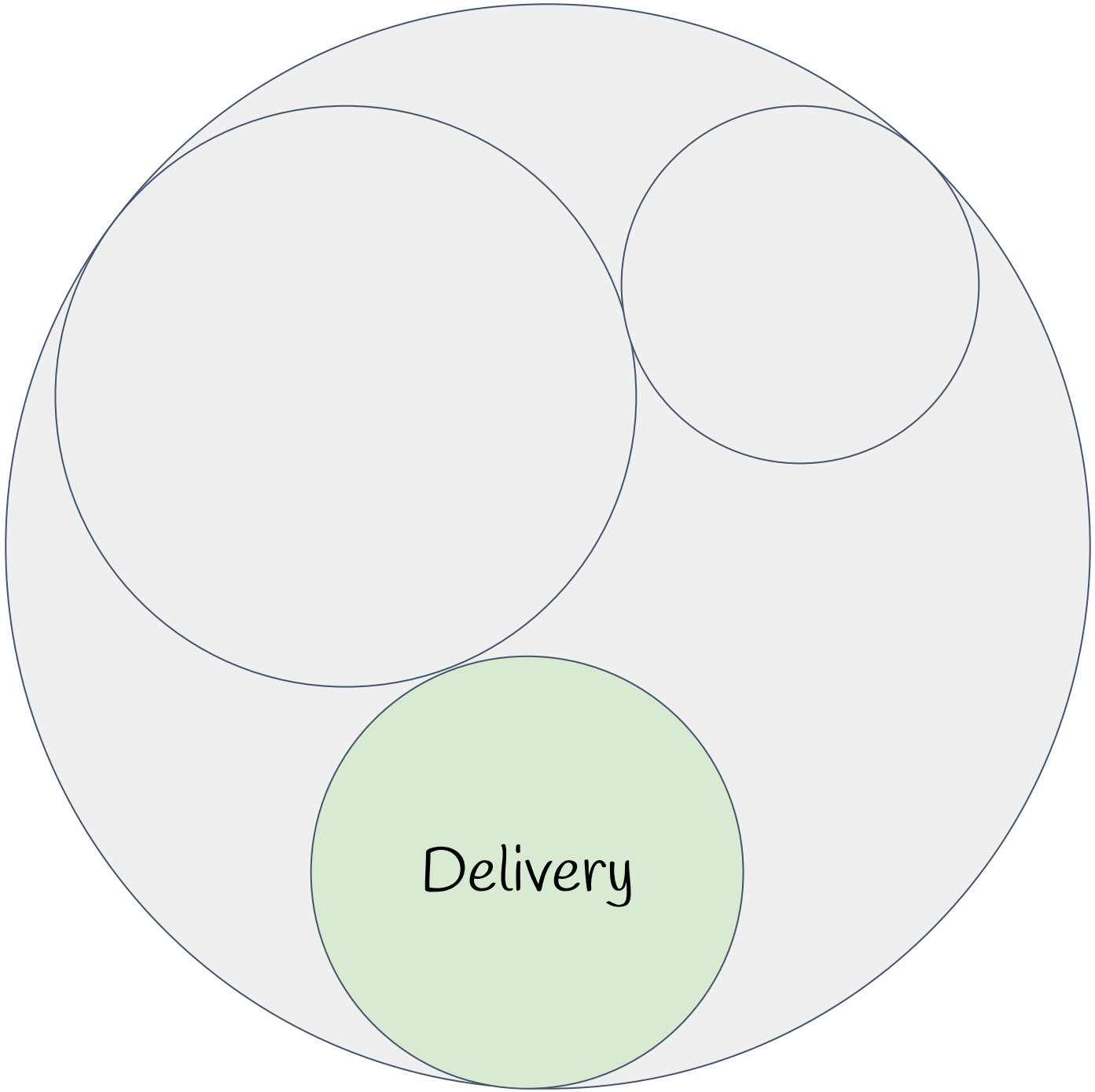

A Web Page
https://example.com
Email / Phone *
First Name
Last Name
Buy!

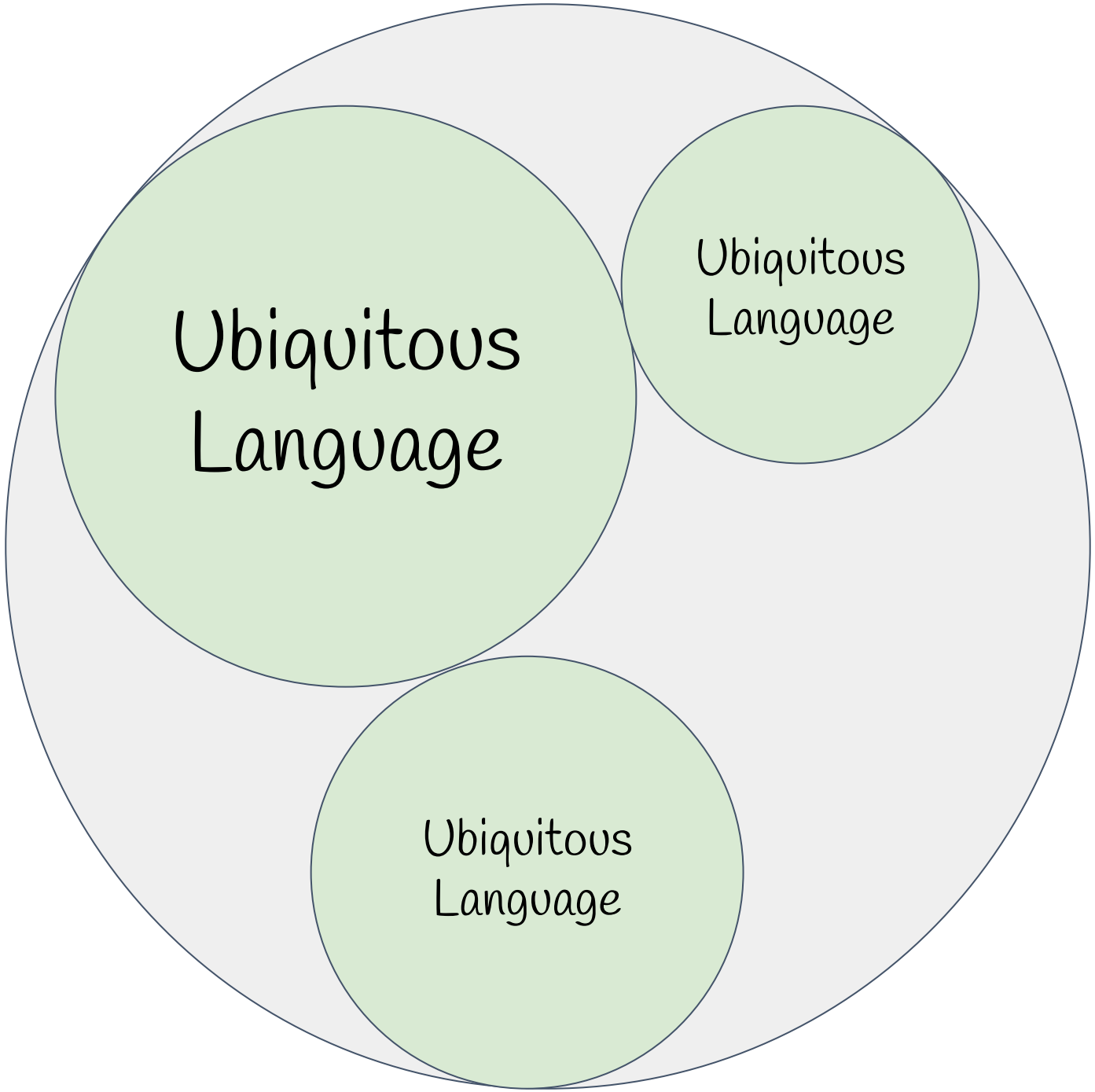

Domain
Model

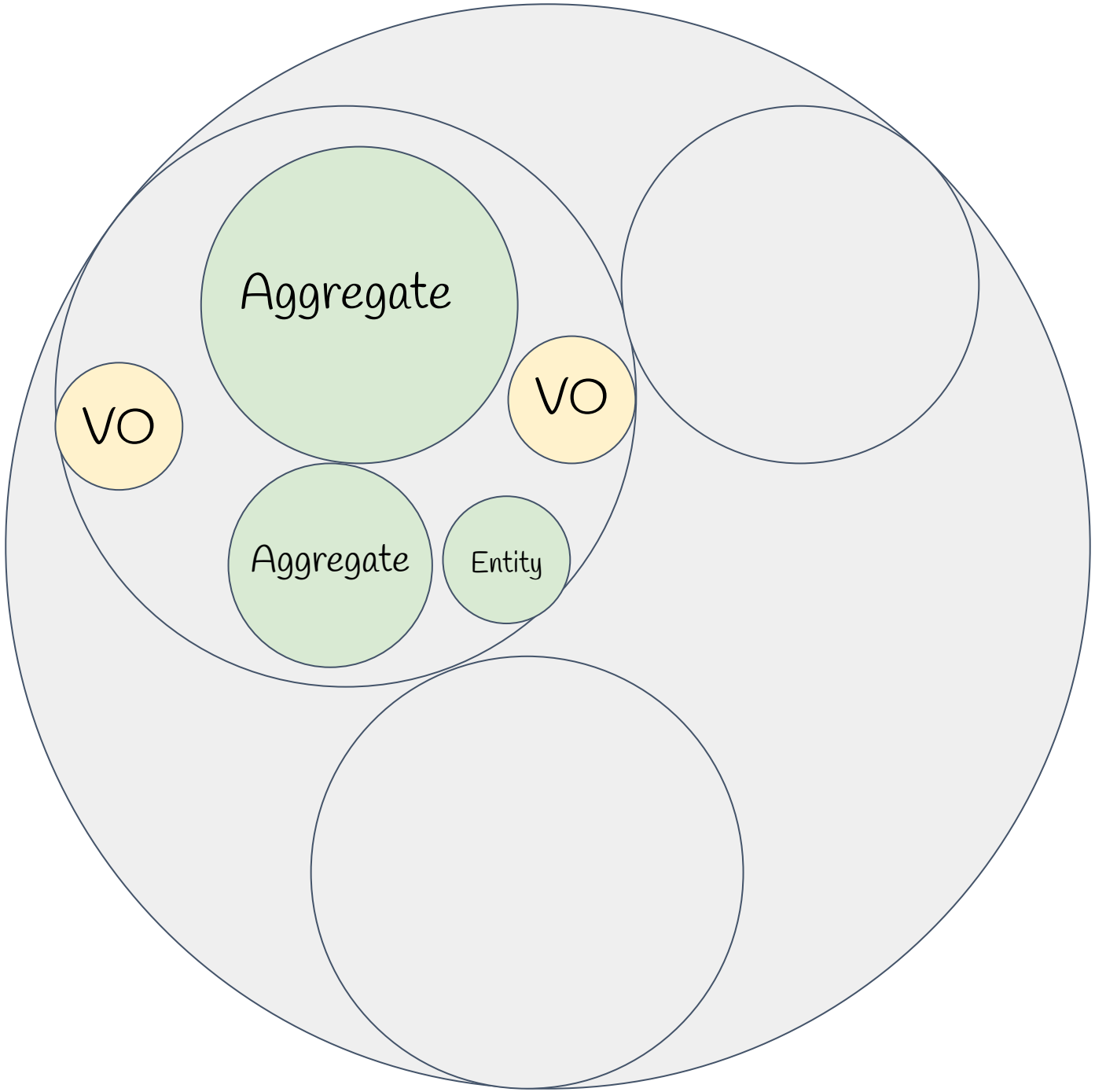

Bounded Context
Bounded Context
Bounded Context

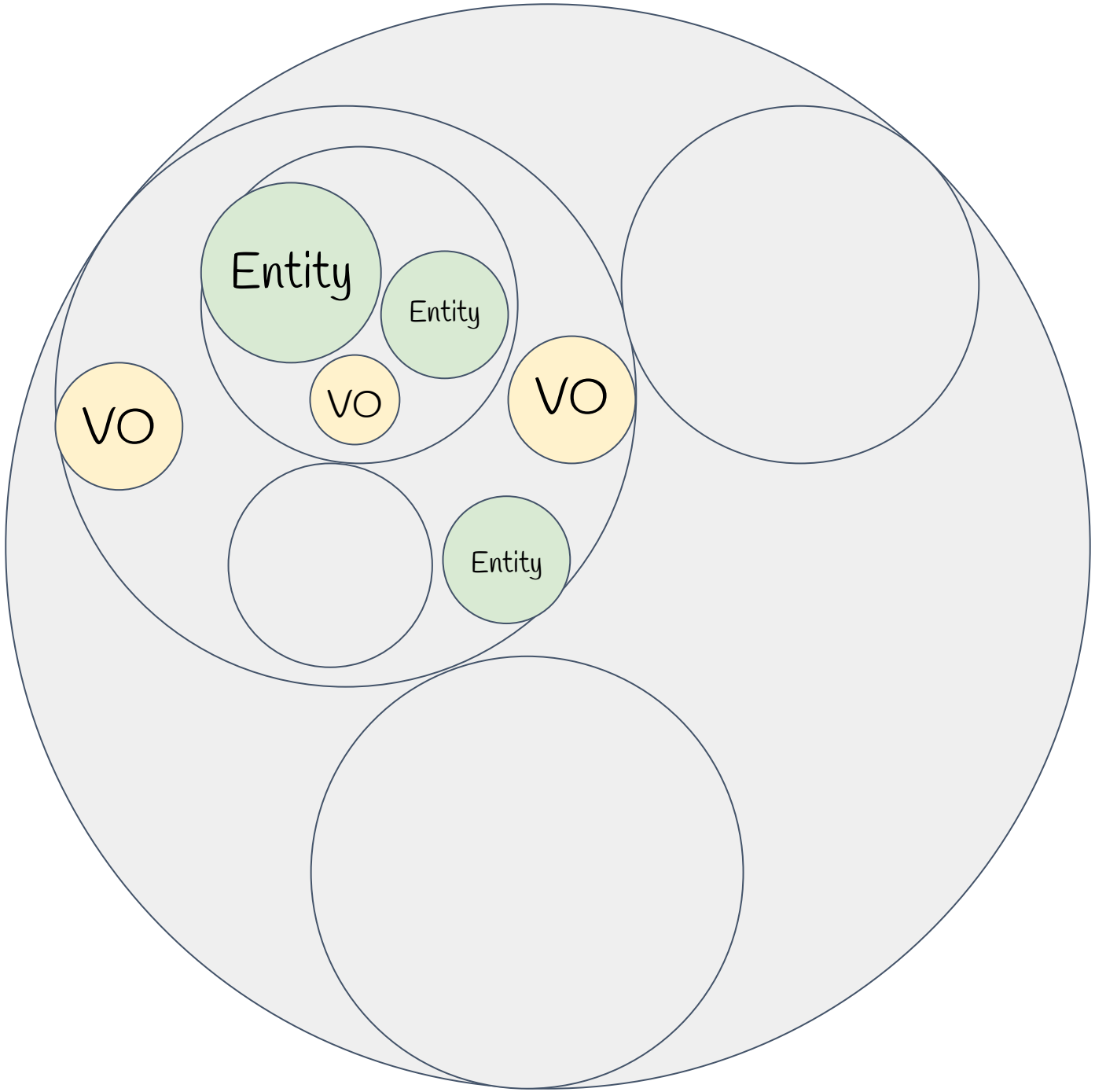

Sales


Production


Delivery


Ubiquitous Language
Ubiquitous Language
Ubiquitous Language


Aggregate
VO
VO
Aggregate
Entity


Entity
Entity
VO
VO
VO
Entity

Дизайн на основе типов

```
public class User
{
    public string FirstName { get; set; }

    public string MiddleName { get; set; }

    public string LastName { get; set; }

    public string Email { get; set; }

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

```
public class User
{
    public string FirstName { get; set; }

    public string MiddleName { get; set; }

    public string LastName { get; set; }

    public string Email { get; set; }

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

Один тип

```
public class User
{
    public string FirstName { get; set; }

    public string MiddleName { get; set; }

    public string LastName { get; set; }

    public string Email { get; set; }

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

Другой

```
public class User : EntityBase
{
    public UserName FullName { get; }

    public Contact Contact { get; }
}
```

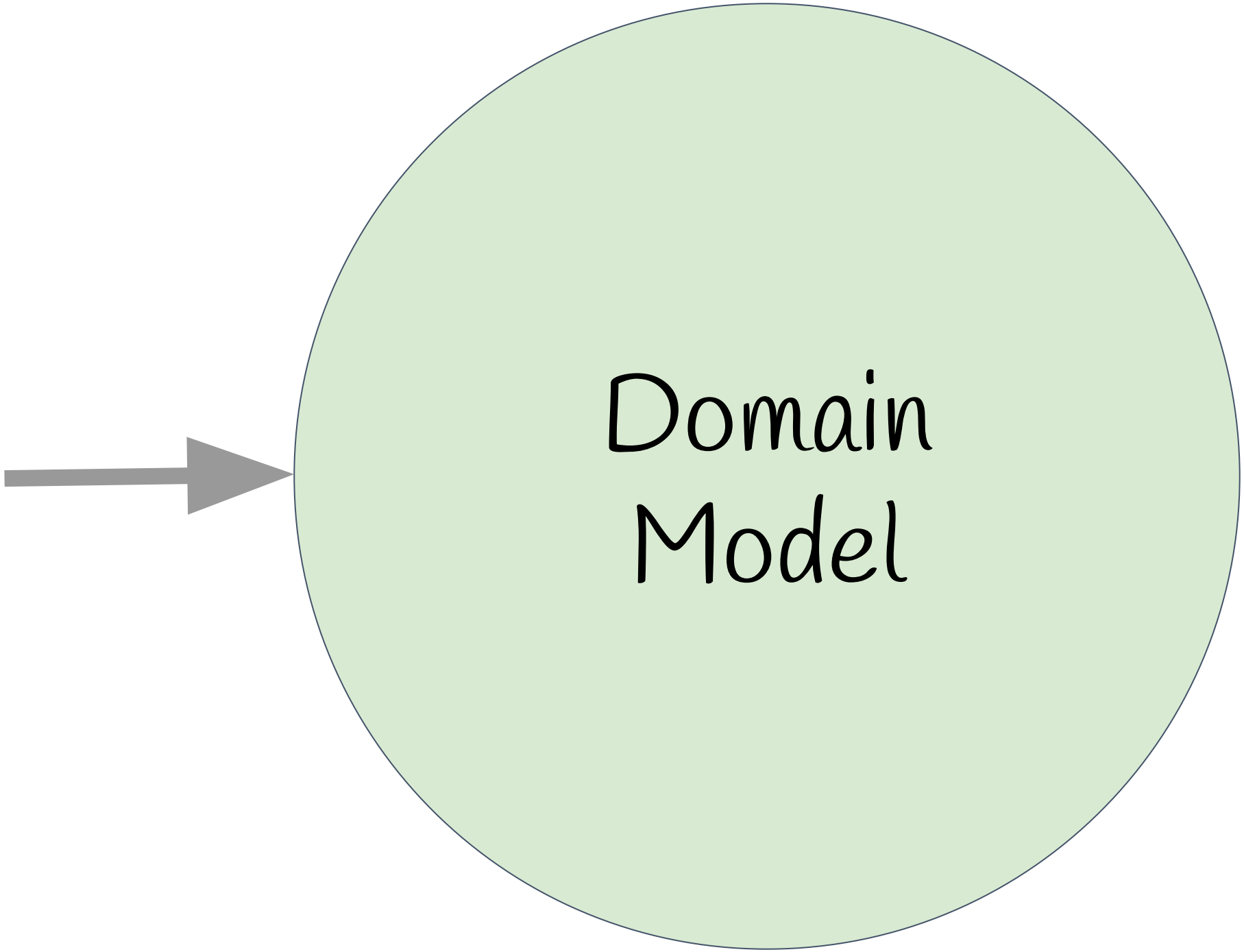
```
public class User : EntityBase
{
    public UserName FullName { get; }

    public Contact Contact { get; }
}
```

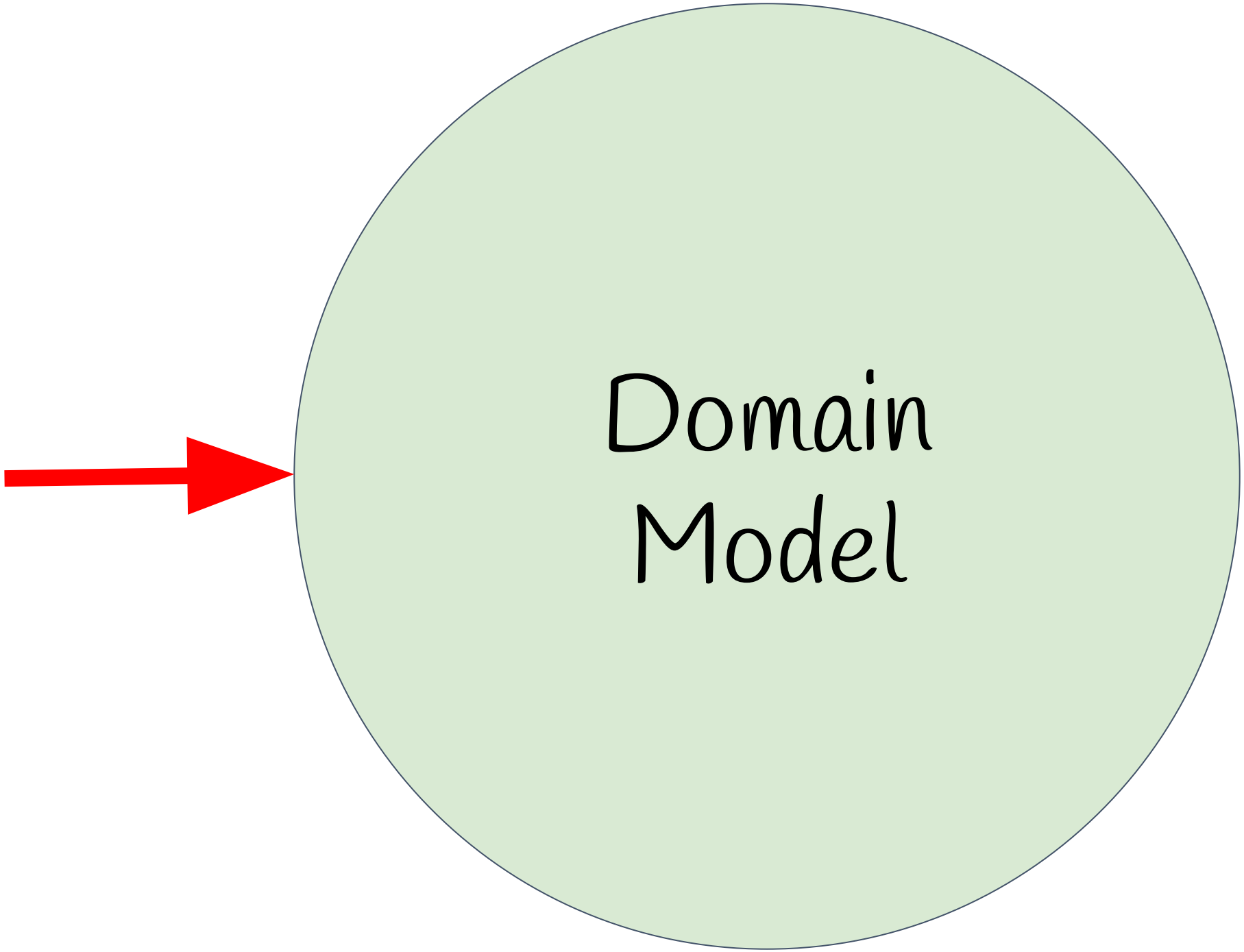
```
public class UserName
{
    [Required, StringLength(255)]
    public string FirstName { get; }

    [Required, StringLength(255)]
    public string LastName { get; }

    [StringLength(1)]
    public string MiddleName { get; }
}
```

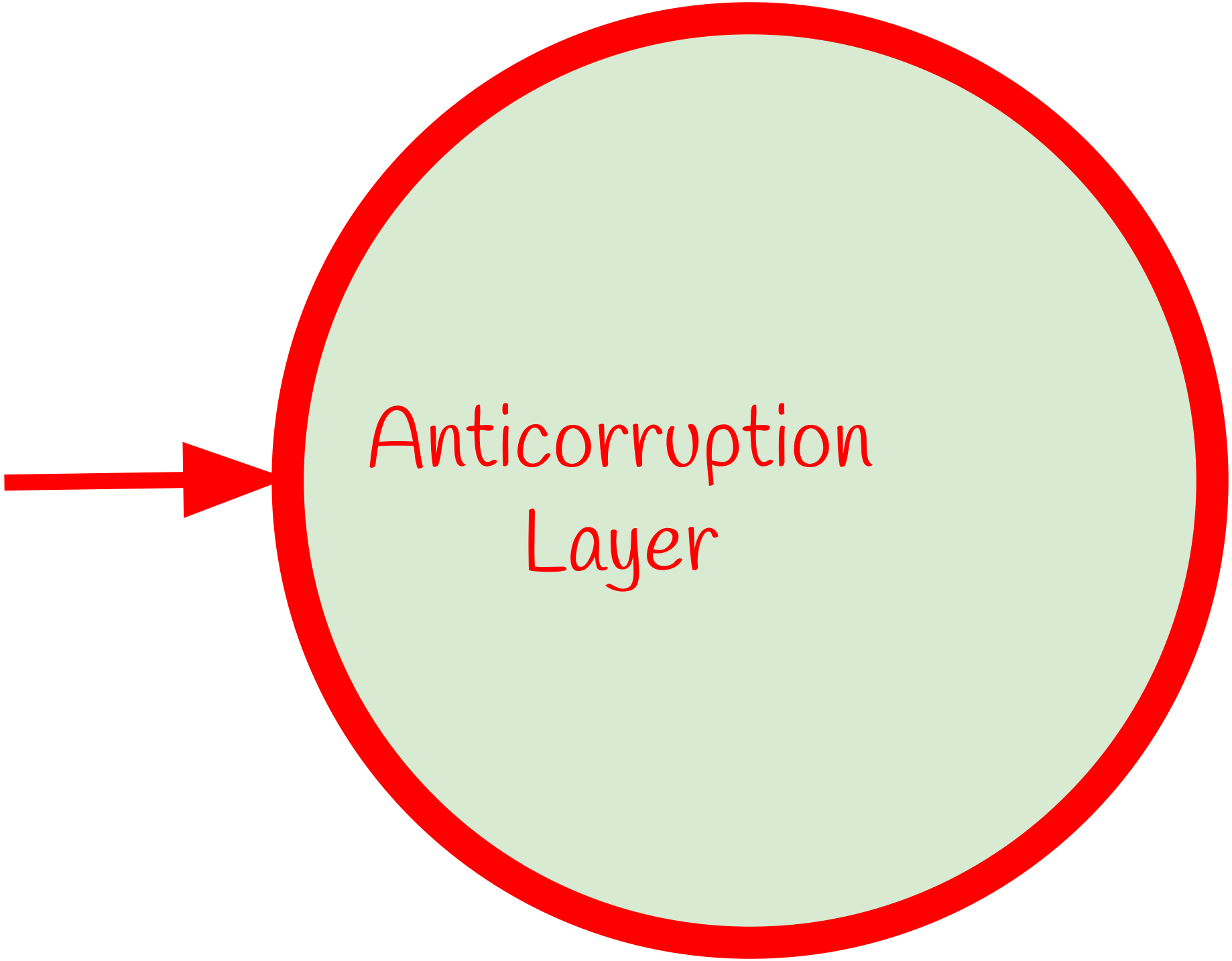
```
public class User : EntityBase
{
    public UserName FullName { get; }

    public Contact Contact { get; }
}
```

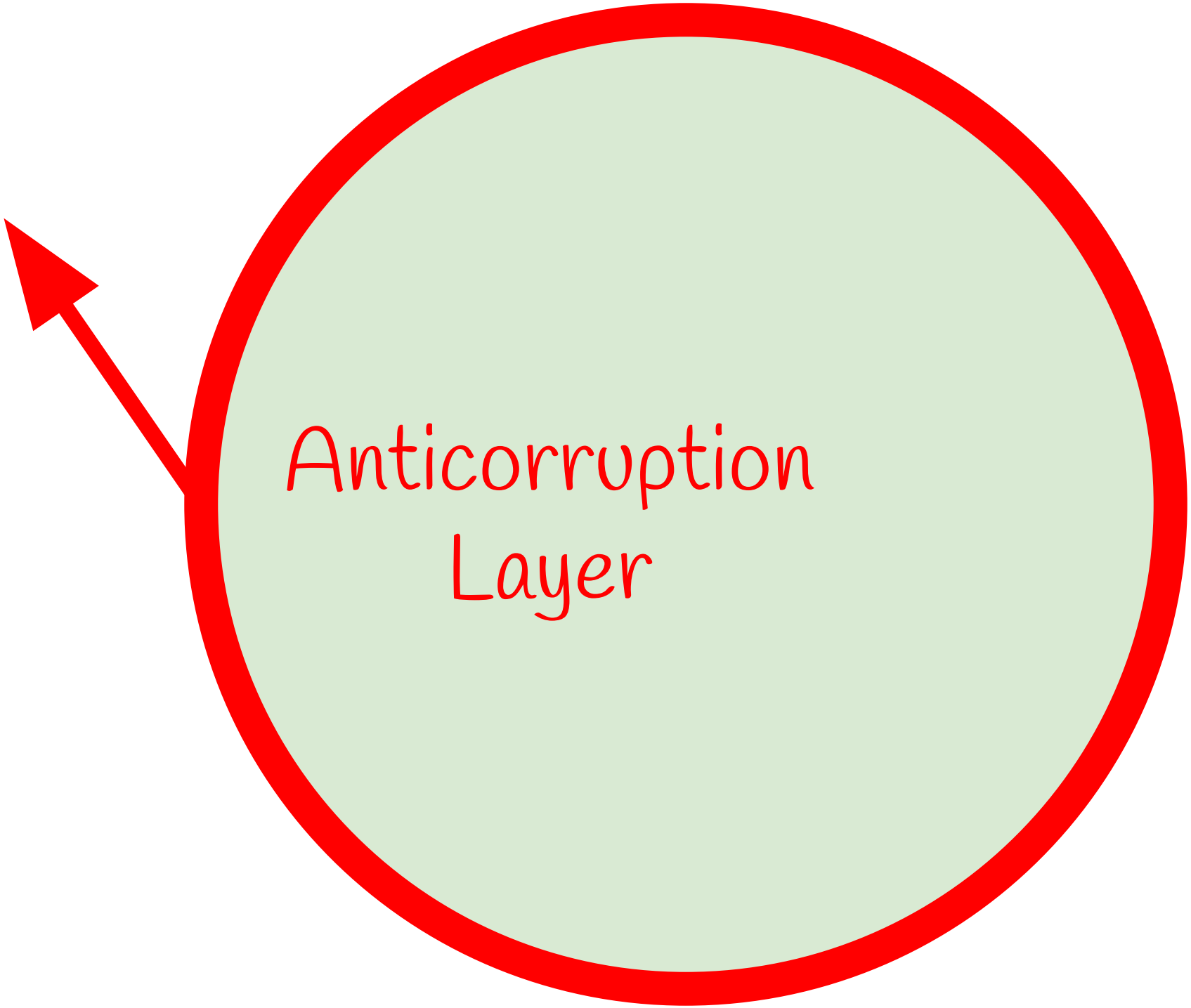
```
public class Contact
{
    public string Email { get; }

    public string Phone { get; }
}
```

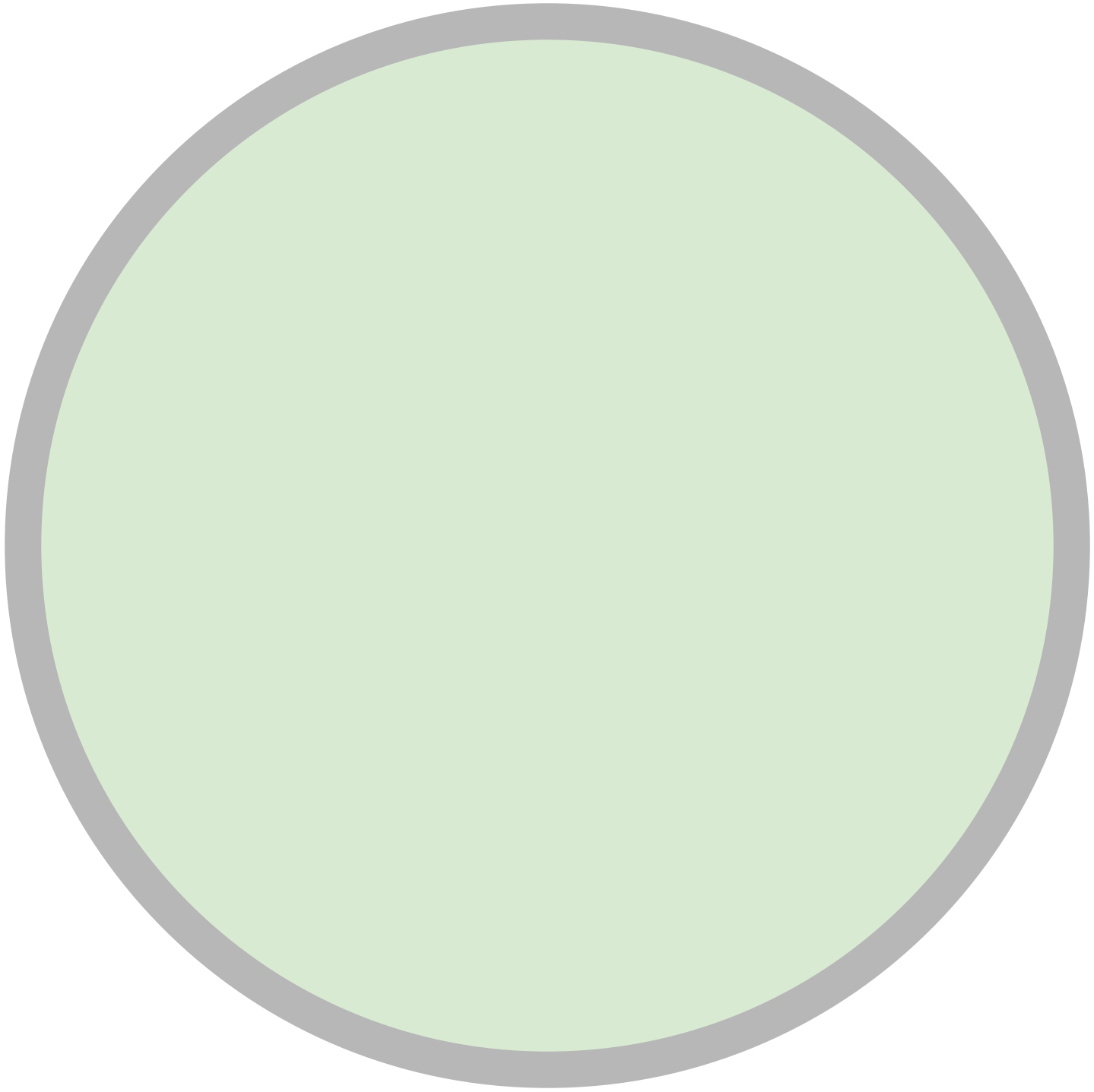
Вынести IO на границы

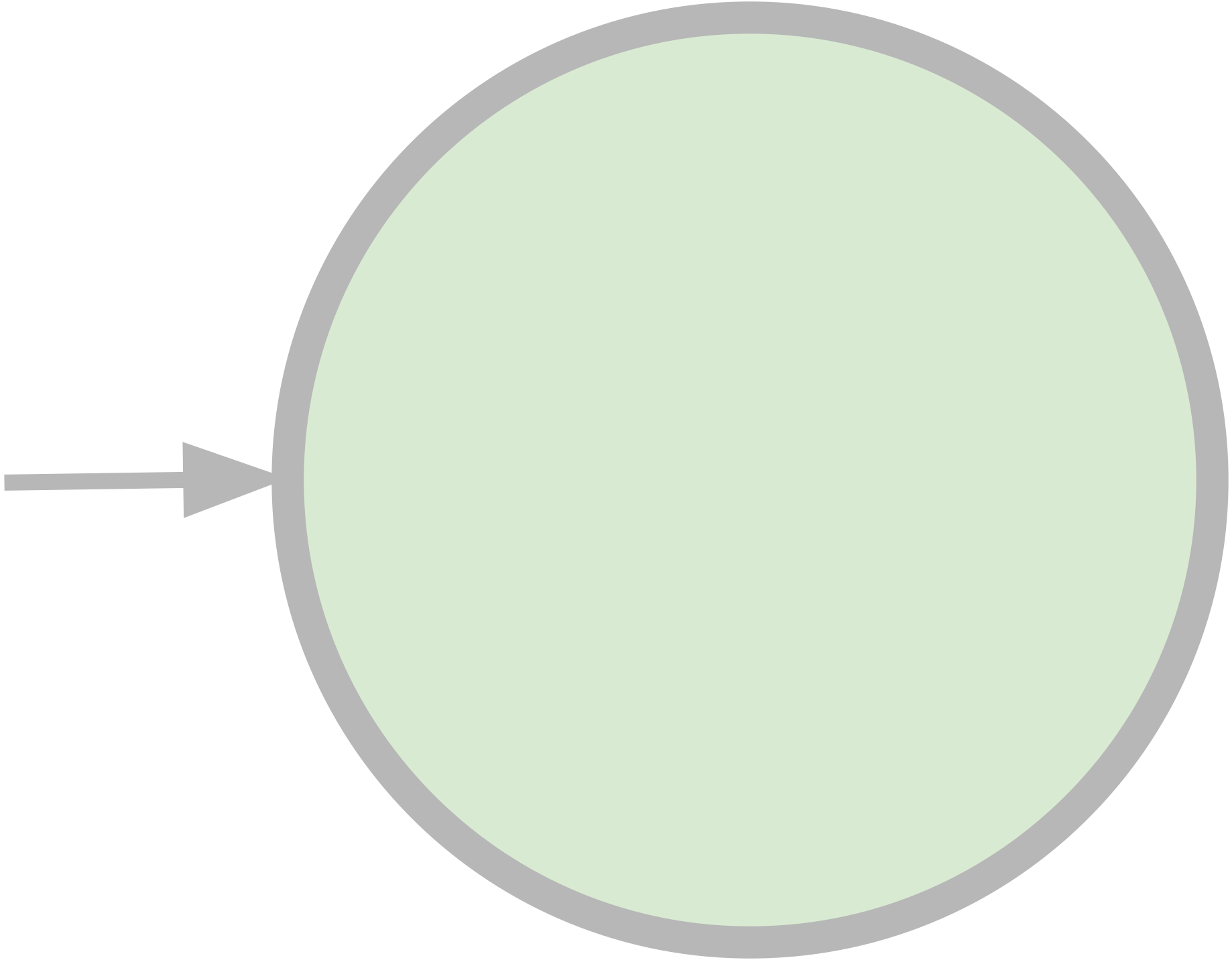

Domain
Model

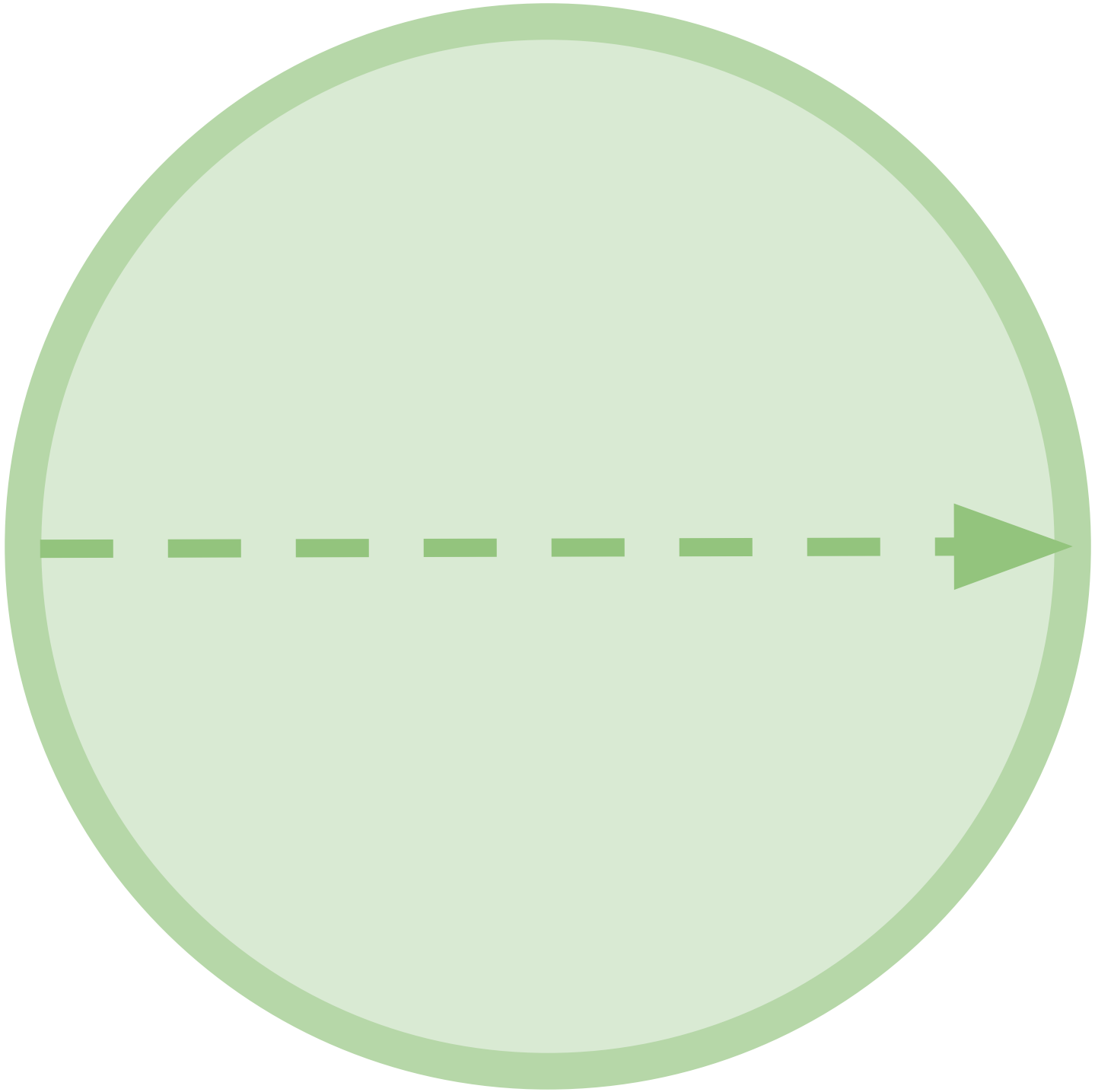

Domain
Model

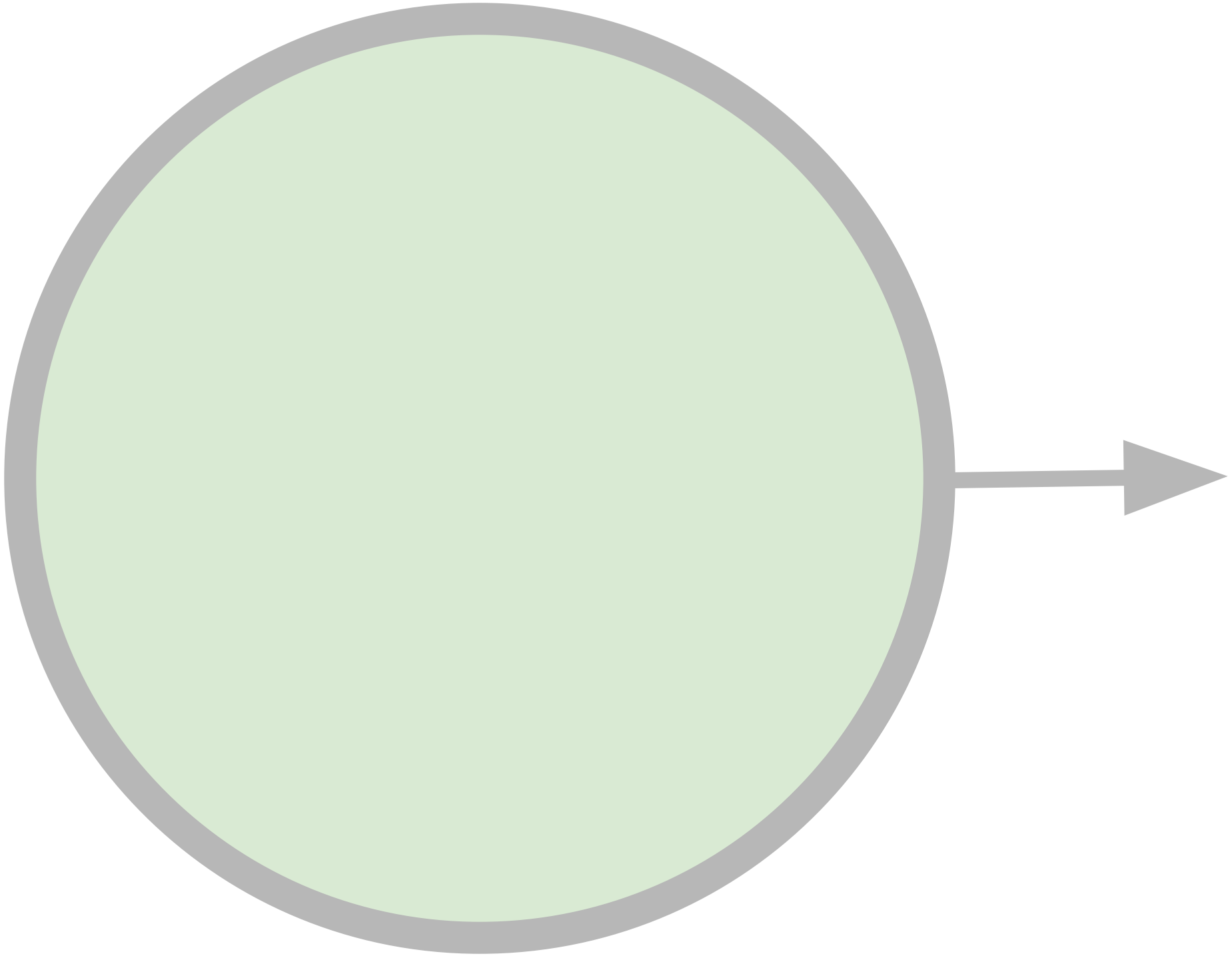

Anticorruption
Layer


Anticorruption
Layer









Crud

у всего есть начало...

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

Entity

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

Value Objects

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName = null)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName = null)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName = null)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    protected Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    protected Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    protected Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    public Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    public Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    const string PhonePattern = @"\+?\d";

    [EmailAddress]
    public string? Email { get; }

    [RegularExpression(PhonePattern)]
    public string? Phone { get; }

    private Contact(string? email, string? phone)
    {
        Email = email;
        Phone = phone;
    }
}
```

```
public class Contact
{
    public static bool TryParsePhone(string phone, out Contact c)
    {
        if (PhoneRegex.IsMatch(phone))
        {
            c = new Contact(null, phone);
            return true;
        }

        c = null;
        return false;
    }
}
```

```
public class Contact
{
    public static bool TryParsePhone(string phone, out Contact c)
    {
        if (PhoneRegex.IsMatch(phone))
        {
            c = new Contact(null, phone);
            return true;
        }

        c = null;
        return false;
    }
}
```

```
public class Contact
{
    public static bool TryParsePhone(string phone, out Contact c)
    {
        if (PhoneRegex.IsMatch(phone))
        {
            c = new Contact(null, phone);
            return true;
        }

        c = null;
        return false;
    }
}
```

```
public class Contact
{
    public static bool TryParsePhone(string phone, out Contact c)
    {
        if (PhoneRegex.IsMatch(phone))
        {
            c = new Contact(null, phone);
            return true;
        }

        c = null;
        return false;
    }
}
```

```
public class UserName
{
    public UserName(string firstName, string lastName, string? middleName)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
        Validator.ValidateObject(this, new ValidationContext(this));
    }

    [Required, StringLength(255)]
    public string FirstName { get; protected set; }

    [Required, StringLength(255)]
    public string LastName { get; protected set; }

    [StringLength(1)]
    public string? MiddleName { get; protected set; }
}
```

```
public class UserName
{
    public UserName(string firstName, string lastName, string? middleName)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
        Validator.ValidateObject(this, new ValidationContext(this));
    }

    [Required, StringLength(255)]
    public string FirstName { get; protected set; }

    [Required, StringLength(255)]
    public string LastName { get; protected set; }

    [StringLength(1)]
    public string? MiddleName { get; protected set; }
}
```

```
public class UserName
{
    public UserName(string firstName, string lastName, string? middleName)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
        Validator.ValidateObject(this, new ValidationContext(this));
    }

    [Required, StringLength(255)]
    public string FirstName { get; protected set; }

    [Required, StringLength(255)]
    public string LastName { get; protected set; }

    [StringLength(1)]
    public string? MiddleName { get; protected set; }
}
```

```
public class UserName
{
    public UserName(string firstName, string lastName, string? middleName)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
        Validator.ValidateObject(this, new ValidationContext(this));
    }

    [Required, StringLength(255)]
    public string FirstName { get; protected set; }

    [Required, StringLength(255)]
    public string LastName { get; protected set; }

    [StringLength(1)]
    public string? MiddleName { get; protected set; }
}
```

```
public class UserName
{
    public UserName(string firstName, string lastName, string? middleName)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
        Validator.ValidateObject(this, new ValidationContext(this));
    }

    [Required, StringLength(255)]
    public string FirstName { get; protected set; }

    [Required, StringLength(255)]
    public string LastName { get; protected set; }

    [StringLength(1)]
    public string? MiddleName { get; protected set; }
}
```

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName = null)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User(Contact contact, UserName? userName = null)
    {
        Contact = contact;
        UserName = userName;
    }
}
```

Откуда берутся пользователи?

```
public class User : EntityBase
{
    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User (SignUp command)
    {
        Contact = command.Contact;
        UserName = command.UserName;
    }
}
```

- Пользователь может зарегистрироваться сам
- …

```
public class User : EntityBase
{
    public User? Invitee { get; protected set;}

    public Contact Contact { get; protected set; }

    public UserName? UserName { get; protected set; }

    public User (InviteUser command)
    {
        Contact = command.Contact;
        UserName = command.UserName;
        Invitee = command.Invitee;
    }
}
```

- Пользователь может зарегистрироваться сам
- Или его может пригласить другой пользователь

```
public class User : EntityBase
{
    public User? Invitee { get; protected set;} // For ORM

    public Contact Contact { get; protected set; } // For ORM

    public UserName? UserName { get; protected set; } // For ORM

    protected User () { } // For ORM

    public User (InviteUser command)

    public User (SignUp command)
}
```

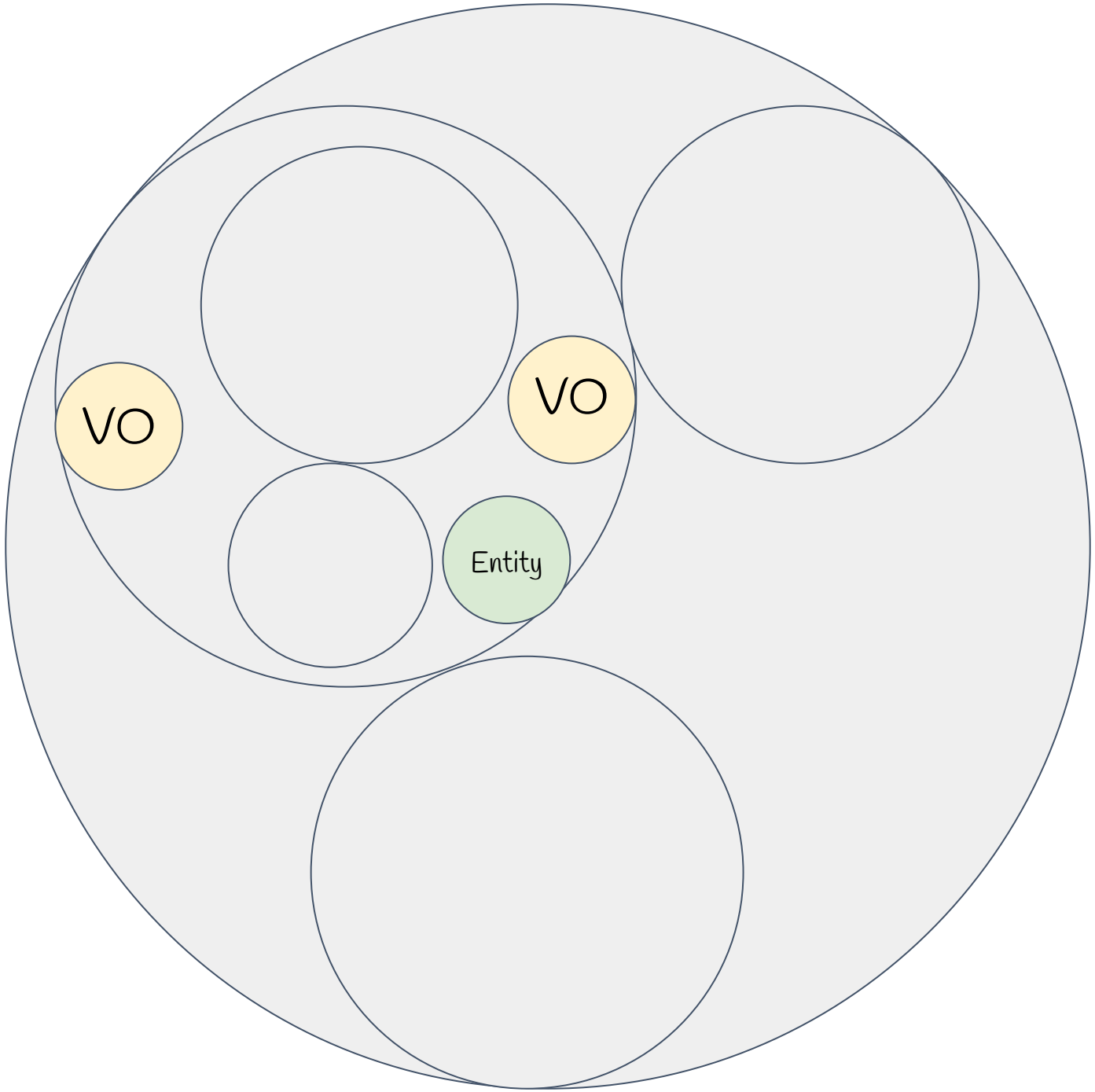
```
public class User : EntityBase
{
    public User? Invitee { get; protected set;} // For ORM

    public Contact Contact { get; protected set; } // For ORM

    public UserName? UserName { get; protected set; } // For ORM

    protected User () { } // For ORM

    public User (InviteUser command)

    public User (SignUp command)
}
```

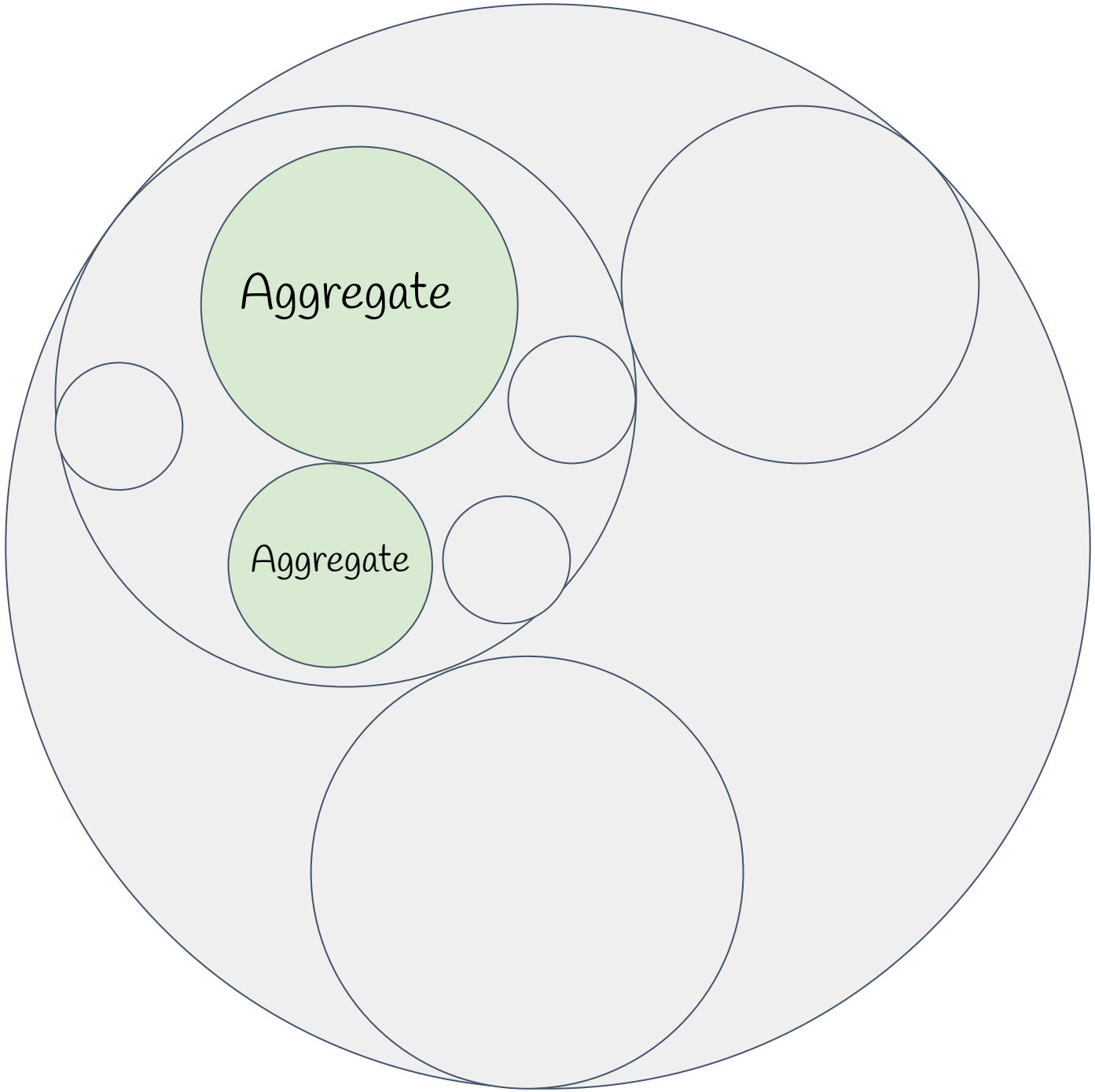
Для ORM

```
public class User : EntityBase
{
    public User? Invitee { get; protected set;} // For ORM

    public Contact Contact { get; protected set; } // For ORM

    public UserName? UserName { get; protected set; } // For ORM

    protected User () { } // For ORM

    public User (InviteUser command)

    public User (SignUp command)
}
```

Для программистов

1:0


VO
VO
Entity


Aggregate
Aggregate

```
public class Order : EntityBase
{
    private List<CartItem> _cartItems = new List<CartItem>();

    public IReadOnlyList<CartItem> CartItems => _cartItems;

    public void Add(Product product)
    {
        _cartItems.Add(new CartItem(this, product));
    }
}
```

```
public class Order : EntityBase
{
    private List<CartItem> _cartItems = new List<CartItem>();

    public IReadOnlyList<CartItem> CartItems => _cartItems;

    public void Add(Product product)
    {
        _cartItems.Add(new CartItem(this, product));
    }
}
```

```
public class Order : EntityBase
{
    private List<CartItem> _cartItems = new List<CartItem>();

    public IReadOnlyList<CartItem> CartItems => _cartItems;

    public void Add(Product product)
    {
        _cartItems.Add(new CartItem(this, product));
    }
}
```

```
public class Order : EntityBase
{
    private List<CartItem> _cartItems = new List<CartItem>();

    public IReadOnlyList<CartItem> CartItems => _cartItems;

    public void Add(Product product)
    {
        _cartItems.Add(new CartItem(this, product));
    }
}
```

```
public class Order : EntityBase
{
    private List<CartItem> _cartItems = new List<CartItem>();

    public IReadOnlyList<CartItem> CartItems => _cartItems;

    public void Add(Product product)
    {
        _cartItems.Add(new CartItem(this, product));
    }
}
```

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

Эти методы имеют смысл только в определенных состояниях!

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

Как и эти данные…

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```

```
public class Order
{
    public decimal TotalPrice { get;}
    public string TrackingUrl { get;}
    public string CancellationReason { get; }
    public bool IsValidated { get; }
    public bool IsShipped { get; }
    public bool IsCancelled { get; }
    public void Validate();
    public void Ship(string trackingUrl);
    public void Cancel(string reason);
}
```


1:1

В мире F# все иначе

```
type UnvalidatedOrder = {
    Details : OrderDetails
}

type ValidatedOrder = {
    Details : OrderDetails
    ValidationTime : DateTime
}
```

```
type PricedOrder = {
    Details : OrderDetails
    TotalPrice : decimal
}

type ShippedOrder = {
    Details : OrderDetails
    Uri : TrackingUrl
}

type CancelledOrder = {
    Details : OrderDetails
    Reason : string
}
```

Record

```
type UnvalidatedOrder = {
    Details : OrderDetails
}

type ValidatedOrder = {
    Details : OrderDetails
    ValidationTime : DateTime
}
```

```
type PricedOrder = {
    Details : OrderDetails
    TotalPrice : decimal
}

type ShippedOrder = {
    Details : OrderDetails
    Uri : TrackingUrl
}

type CancelledOrder = {
    Details : OrderDetails
    Reason : string
}
```

```
type UnvalidatedOrder = {
    Details : OrderDetails
}
```

```
type ValidatedOrder = {
    Details : OrderDetails
    ValidationTime : DateTime
}
```

Еще один

```
type PricedOrder = {
    Details : OrderDetails
    TotalPrice : decimal
}
```

```
type ShippedOrder = {
    Details : OrderDetails
    Uri : TrackingUrl
}
```

```
type CancelledOrder = {
    Details : OrderDetails
    Reason : string
}
```

Records

```
type UnvalidatedOrder = {
    Details : OrderDetails
}

type ValidatedOrder = {
    Details : OrderDetails
    ValidationTime : DateTime
}
```

```
type PricedOrder = {
    Details : OrderDetails
    TotalPrice : decimal
}

type ShippedOrder = {
    Details : OrderDetails
    Uri : TrackingUrl
}

type CancelledOrder = {
    Details : OrderDetails
    Reason : string
}
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

Discriminated Union

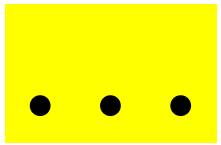
```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

```
type UnvalidatedOrder = {
    Details : OrderDetails
}
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

```
type ValidatedOrder = {
    Details : OrderDetails
    ValidationTime : DateTime
}
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```



```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```fsharp
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

Это просто функция

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

А это pattern matching

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```fsharp
let getStateName order =
    match order with
        | Order.CancelledOrder -> "Cancelled"
        | Order.PricedOrder -> "Priced"
        | Order.ShippedOrder -> "Shipped"
        | Order.UnvalidatedOrder -> "Unvalidated"
        | Order.ValidatedOrder -> "Validated"
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
```

```
type Order =
    | UnvalidatedOrder
    | ValidatedOrder
    | PricedOrder
    | ShippedOrder
    | CancelledOrder
    | EvilOrder
```

2:1

Распределенные транзакции

и файл сохранить и в базу записать

```
public abstract class Document<T> : Document
{
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
}
```

```
public abstract class Document<T> : Document
{
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
}
```

```
public abstract class Document<T> : Document
{
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
}
```

```
public abstract class Document<T> : Document
{ try {
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
} }
   catch(FileStorageException)
   catch(SqlException)
```

```
public abstract class Document<T> : Document
{ try {
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
} }
  catch(FileStorageException)
  catch(SqlException)
```

Где мои гарантии?

2:2

```
public abstract class Document<T> : Document
{ try {
    public Document(UploadedFile file, T meta);

    public void Update(UploadedFile file, T meta);
} }
  catch(FileStorageException)
  catch(SqlException)
```

Где мои гарантии?

```
public abstract class Document<T> : Document
{ try {
    internal Document(UploadedFile file, T meta

    internal void Update(UploadedFile file, T meta);
} }
  catch(FileStorageException)
  catch(SqlException)
```

Где мои гарантии?

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data);}
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data);}
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

Что-то пошло не так

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

Что-то пошло не так, когда что-то итак уже шло не так...

```
public void Update<T>(Document<T> entity, IFormFile file, T data) {
    UploadedFile uploaded = fileStorage.Upload(file);
    try { entity.Update(uploaded, data); }
    catch (SqlException se)
    {
        try { fileStorage.Delete(uploaded); }
        catch (FileStorageException fe)
        {
            logger.Fatal(fe.Message);
            throw;
        }
        throw;
    }
}
```

Все вместе

Все вместе

1. Entity + Value Objects: свойства объектов корректны по отдельности и вместе

Все вместе

1. Entity + Value Objects: свойства объектов корректны по отдельности и вместе
2. Aggregate: группы объектов корректны

Все вместе

1. Entity + Value Objects: свойства объектов корректны по отдельности и вместе
2. Aggregate: группы объектов корректны
3. Pattern Matching: разное поведение для разных состояний

Все вместе

1. Entity + Value Objects: свойства объектов корректны по отдельности и вместе
2. Aggregate: группы объектов корректны
3. Pattern Matching: разное поведение для разных состояний
4. Сервисы + internal: инфраструктура и домен синхронизированы

Главный вопрос DDD

Что делать, когда операция падает
с `OutOfMemoryException`
или выполняется 10 часов?


Все еще
хочешь банан?

2:3

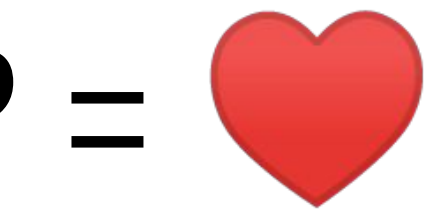
cRud

cRud

Написать SQL-запрос

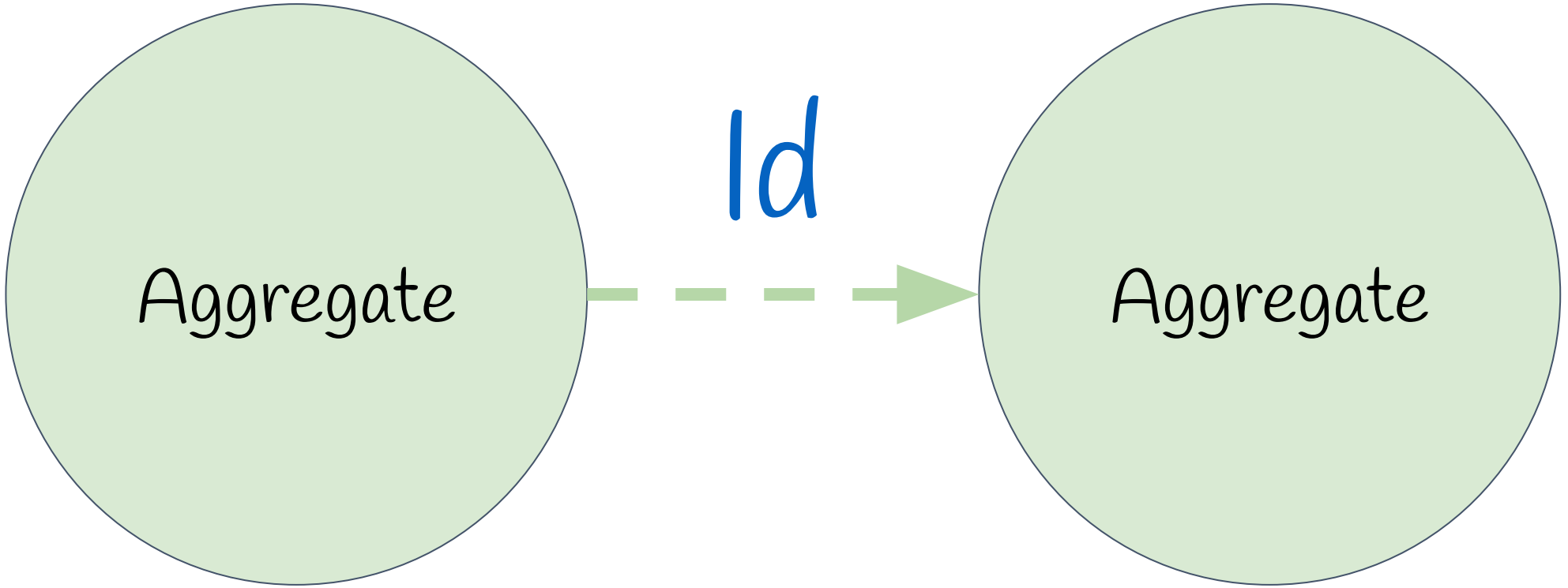
cRud

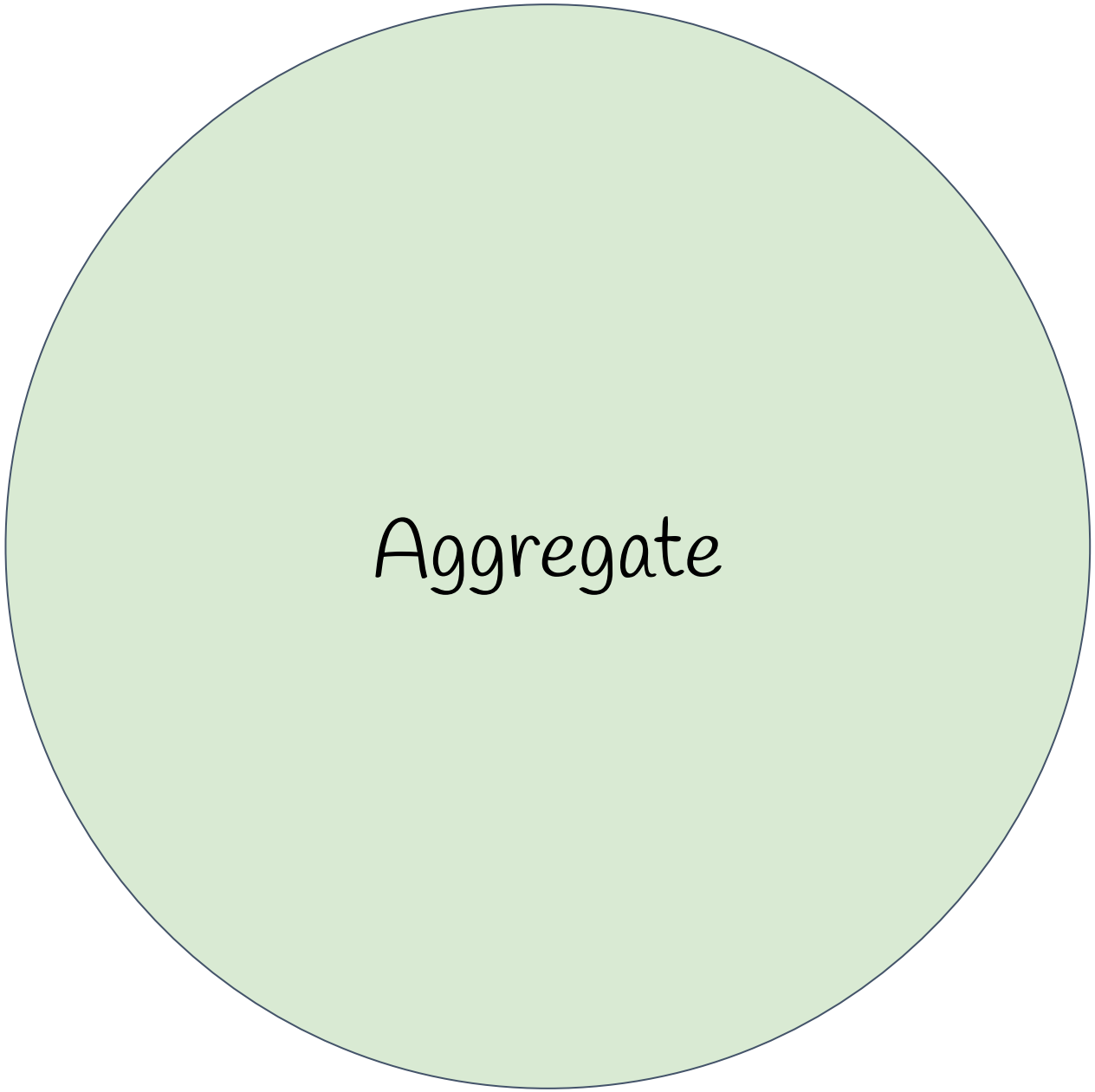
Написать SQL-запрос

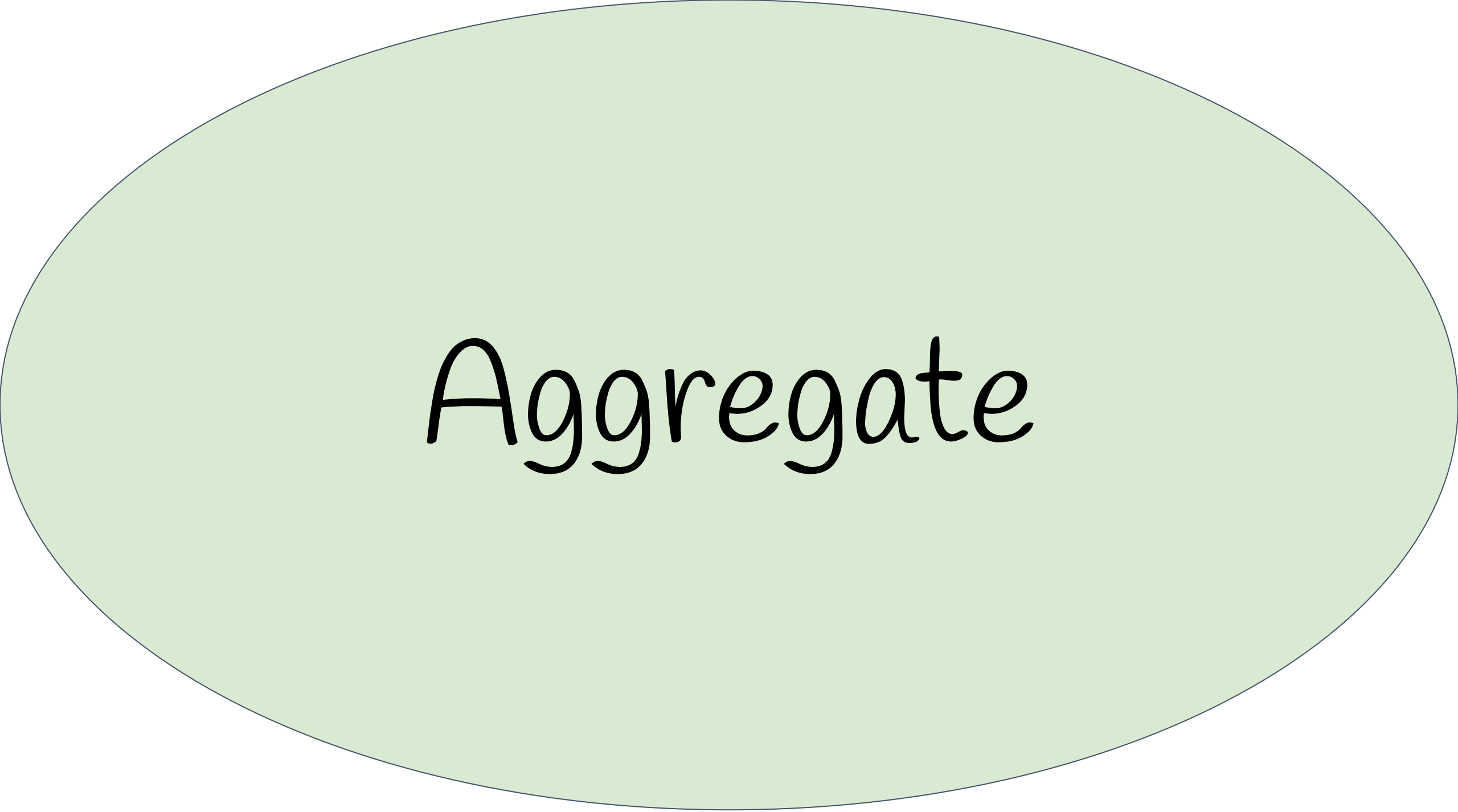
CQRS + HTTP = 

CrUD

?


Aggregate
Id
Aggregate


Aggregate


Aggregate

Делаем SOLIDно?

```
public partial class Order : EntityBase
{
    public IQueryable<OrderItem> ItemsQuery
        => _dbContext
            .Set<OrderItem>
            .Where(x => x.OrderId == Id);
}
```

```
public partial class Order : EntityBase
{
    public IQueryable<OrderItem> ItemsQuery
        => _dbContext
            .Set<OrderItem>
            .Where(x => x.OrderId == Id);
}
```

```
public partial class Order : EntityBase
{
    public IQueryable<OrderItem> ItemsQuery
        => _dbContext
            .Set<OrderItem>
            .Where(x => x.OrderId == Id);
}
```

```
public partial class Order : EntityBase
{
    public IQueryable<OrderItem> ItemsQuery
        => _dbContext
            .Set<OrderItem>
            .Where(x => x.OrderId == Id);
}
```

Lazy Load

за и против

За

- Tell Don’t Ask aka Закон Деметры
- Persistence Ignorance
- Меньше boilerplate
- Простота использования

Против

- Производительность
- Непредвиденное IO
- Возможны проблемы с Reflection или кодом, сгенерированным в runtime
- Массовые операции

За

- Tell Don’t Ask aka Закон Деметры
- Persistence Ignorance
- Меньше boilerplate
- Простота использования

Против

- Производительность
- Непредвиденное IO
- Возможны проблемы с Reflection или кодом, сгенерированным в runtime
- Массовые операции

3:3

Expressions

Specification Pattern

правила бизнес логики в виде объектов,
связанных операциями булевой логики

```
public Spec<Product> IsForSale { get; } =
    new Spec<Product>(x => x.Price > 0);
```

```
public Spec<Product> IsForSale { get; } =
    new Spec<Product>(x => x.Price > 0);
```

```
public Spec<Product> IsForSale { get; } =
    new Spec<Product>(x => x.Price > 0);
```

```
var products
    .Where(Product.Specs.IsForSale)
    .ToList();
```

```
SELECT *
FROM Products
WHERE Price > 0
```

```
var products
    .Where(Product.Specs.IsForSale)
    .ToList();
```

```
var products
    .Where(Product.Specs.IsForSale)
    .Select(x => new
    {
        x.Id,
        x.Name,
        x.Price
    })
    .ToList();
```

```
var products
    .Where(Product.Specs.IsForSale)
    .Select(x => new
    {
        x.Id,
        x.Name,
        x.Price
    })
    .ToList();
```

```
SELECT Id, Name, Price
FROM Products
WHERE Price > 0
```

```
var products
    .Where(Product.Specs.IsForSale)



    .ProjectToType<ProductListItem>()


    .ToList();
```

Default implementations in interfaces

```
public class Product : EntityBase
{
    public decimal Price { get; }
    public decimal DiscountPercent { get; }

    public decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public class Product : EntityBase
{
    public decimal Price { get; }
    public decimal DiscountPercent { get; }

    public decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public interface IHasDiscount
{
    decimal Price { get; set; }
    decimal DiscountPercent { get; set; }

    decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public interface IHasDiscount
{
    decimal Price { get; set; }
    decimal DiscountPercent { get; set; }

    decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public class ProductListItem : IHasDiscount

public interface IHasDiscount
{
    public decimal Price { get; }
    public decimal DiscountPercent { get; }

    public decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public class ProductListItem : IHasDiscount

public interface IHasDiscount
{
    public decimal Price { get; }
    public decimal DiscountPercent { get; }

    public decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

```
public class Product : IHasDiscount

public interface IHasDiscount
{
    public decimal Price { get; }
    public decimal DiscountPercent { get; }

    public decimal DiscountedPrice()
        => Price - Price / 100 * DiscountPercent;
}
```

Массовые операции

```
dbContext
    .Set<Product>()
    .Where(!Product.Specs.IsForSale)
    .BatchDelete();

dbContext
    .Set<Product>()
    .Where(Product.Specs.IsForSale)
    .BatchUpdate(x => new { Price = x.Price + 100 });
```

```
dbContext
    .Set<Product>()
    .Where(!Product.Specs.IsForSale)
    .BatchDelete();
```

```
DELETE
FROM Products
WHERE NOT Price > 0
```

```
dbContext
    .Set<Product>()
    .Where(Product.Specs.IsForSale)
    .BatchUpdate(x => new { Price = x.Price + 100 });
```

```
dbContext
    .Set<Product>()
    .Where(!Product.Specs.IsForSale)
    .BatchDelete();
```

```
dbContext
    .Set<Product>()
    .Where(Product.Specs.IsForSale)
    .BatchUpdate(x => new { Price = x.Price + 100 });
```

```
dbContext
    .Set<Product>()
    .Where(!Product.Specs.IsForSale)
    .BatchDelete();
```

```
UPDATE Products
SET Price =
    Price + 100
WHERE Price > 0
```

```
dbContext
    .Set<Product>()
    .Where(Product.Specs.IsForSale)
    .BatchUpdate(x => new { Price = x.Price + 100 });
```

F#?

F#?

Bounded Context

разделяй и властвуй

```
public class User : EntityBase
{
    public ICollection<Order> Orders => _orders;

    public ICollection<Setting> Settings => _settings;

    public ICollection<Comment> Comments => _comments;

    // ...
}
```

```
public class User : EntityBase
{
    public ICollection<Order> Orders => _orders;

    public ICollection<Setting> Settings => _settings;

    public ICollection<Comment> Comments => _comments;

    // ...
}
```

Божественный объект

```
public class User : EntityBase
{
    public ICollection<Order> Orders => _orders;

    public ICollection<Setting> Settings => _settings;

    public ICollection<Comment> Comments => _comments;

    // ...
}
```

Циклические зависимости

```
public class User : EntityBase
{
    public ICollection<Order> Orders => _orders;

    public ICollection<Setting> Settings => _settings;

    public ICollection<Comment> Comments => _comments;

    // ...
}
```

Ложный агрегат

Как выбрать корень агрегации?

Как выбрать корень агрегации?

1. Есть ли аналог в реальном мире?
2. Когда вы рассказываете о своем дизайне бизнес-пользователям, считают ли они вас местным сумасшедшим?

Как выбрать корень агрегации?

1. Есть ли аналог в реальном мире?
2. Когда вы рассказываете о своем дизайне бизнес-пользователям, считают ли они вас местным сумасшедшим?

1. Агрегат не пересекает Bounded Context?
2. Есть ли инвариант для этой группы объектов?
3. Данные должны изменяться в рамках одной транзакции?

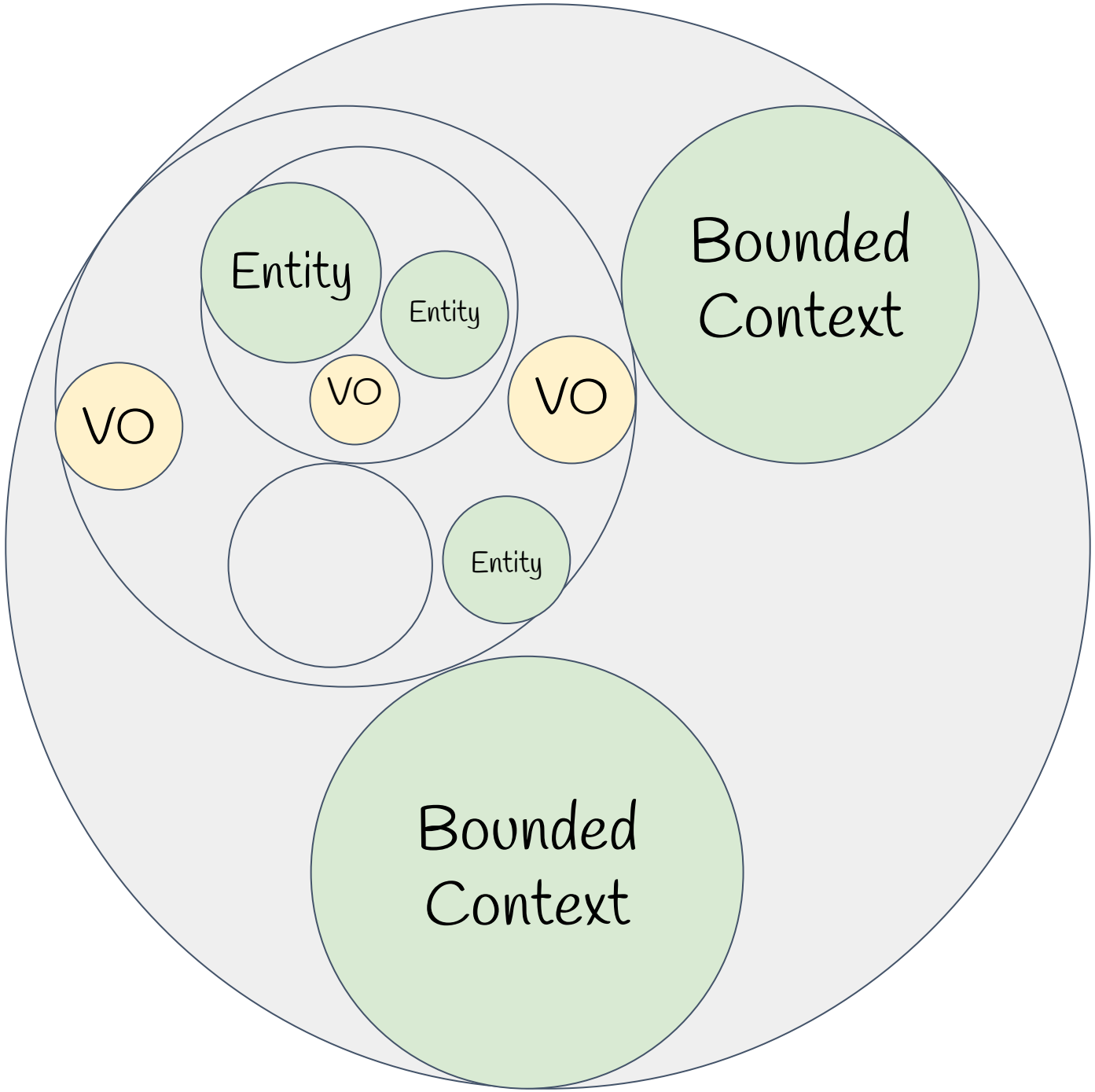
```
public class Client : User
{
    public ICollection<Order> Orders => _orders;
}
public class Account : User
{
    public ICollection<Setting> Settings => _settings;
}
public class Blogger : User
{
    public ICollection<Comment> Comments => _comments;
}
```

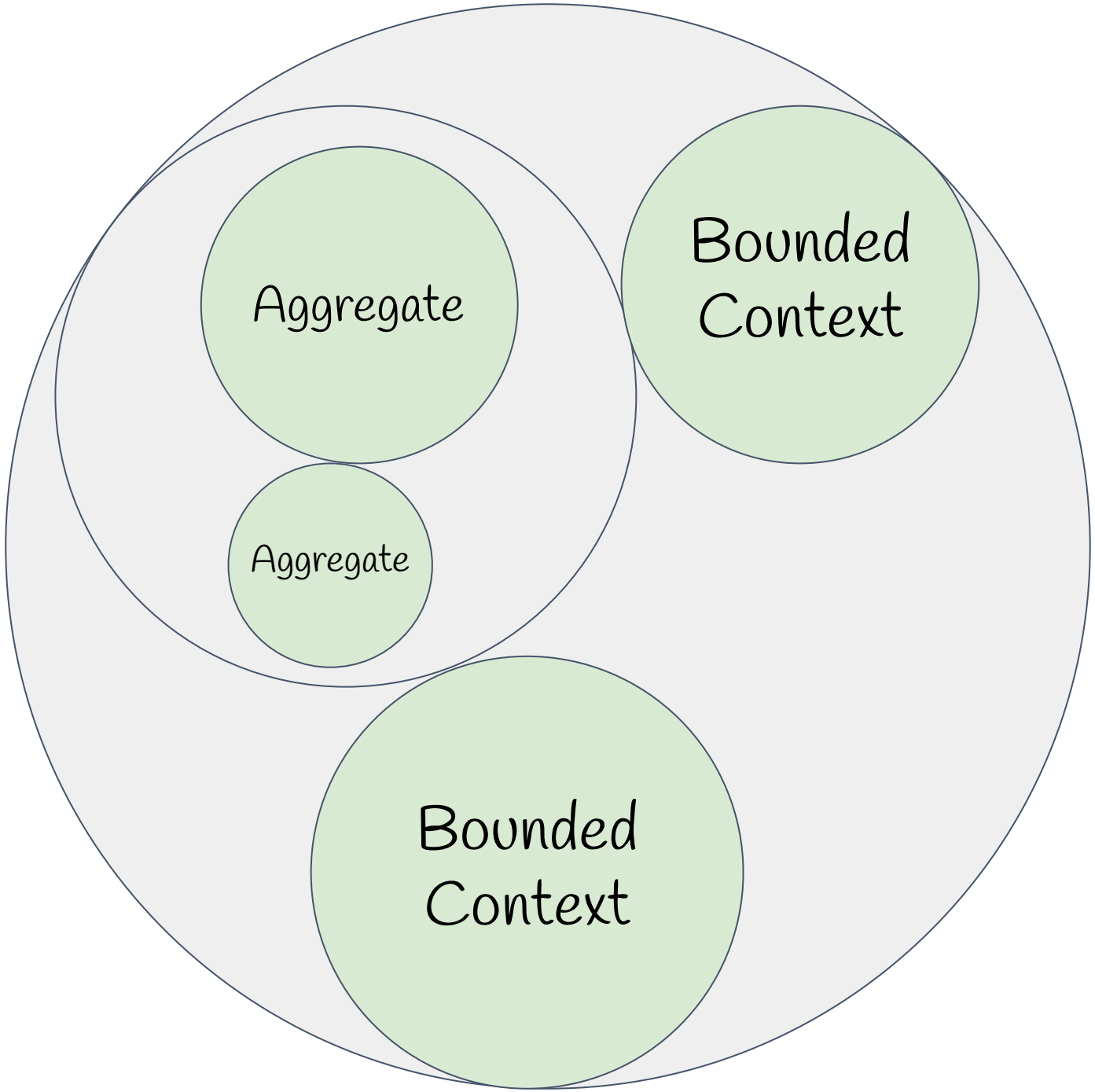
```
public class Client : User
{
    public ICollection<Order> Orders => _orders;
}
public class Account : User
{
    public ICollection<Setting> Settings => _settings;
}
public class Blogger : User
{
    public ICollection<Comment> Comments => _comments;
}
```

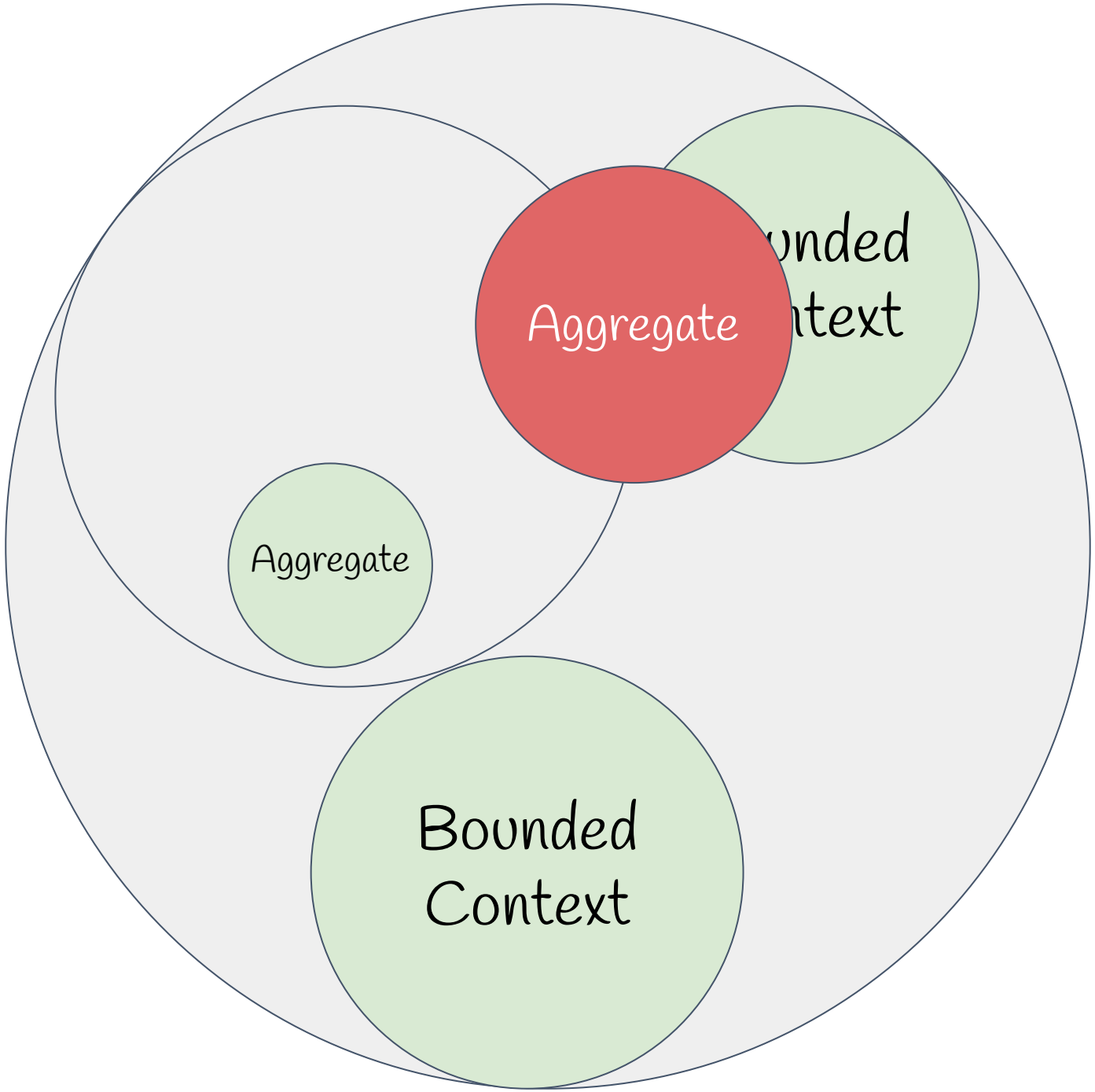
```
public class Client : User
{
    public ICollection<Order> Orders => _orders;
}
public class Account : User
{
    public ICollection<Setting> Settings => _settings;
}
public class Blogger : User
{
    public ICollection<Comment> Comments => _comments;
}
```

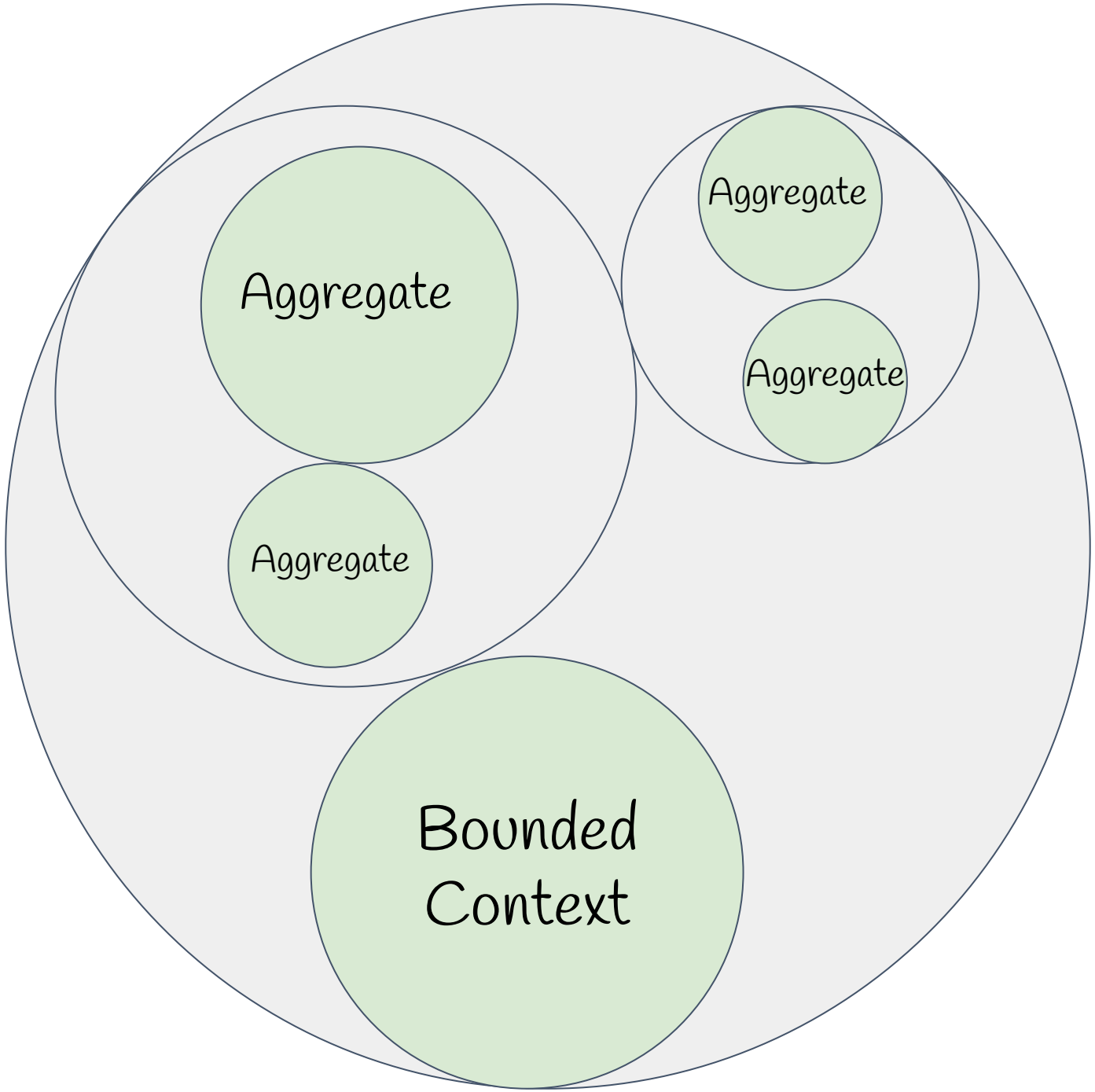
```
public class Client : User
{
    public ICollection<Order> Orders => _orders;
}
public class Account : User
{
    public ICollection<Setting> Settings => _settings;
}
public class Blogger : User
{
    public ICollection<Comment> Comments => _comments;
}
```

Как делить Domain Model на Bounded Context?


Entity
Entity
VO
VO
VO
Entity
Bounded Context
Bounded Context


Aggregate
Aggregate
Bounded Context
Bounded Context

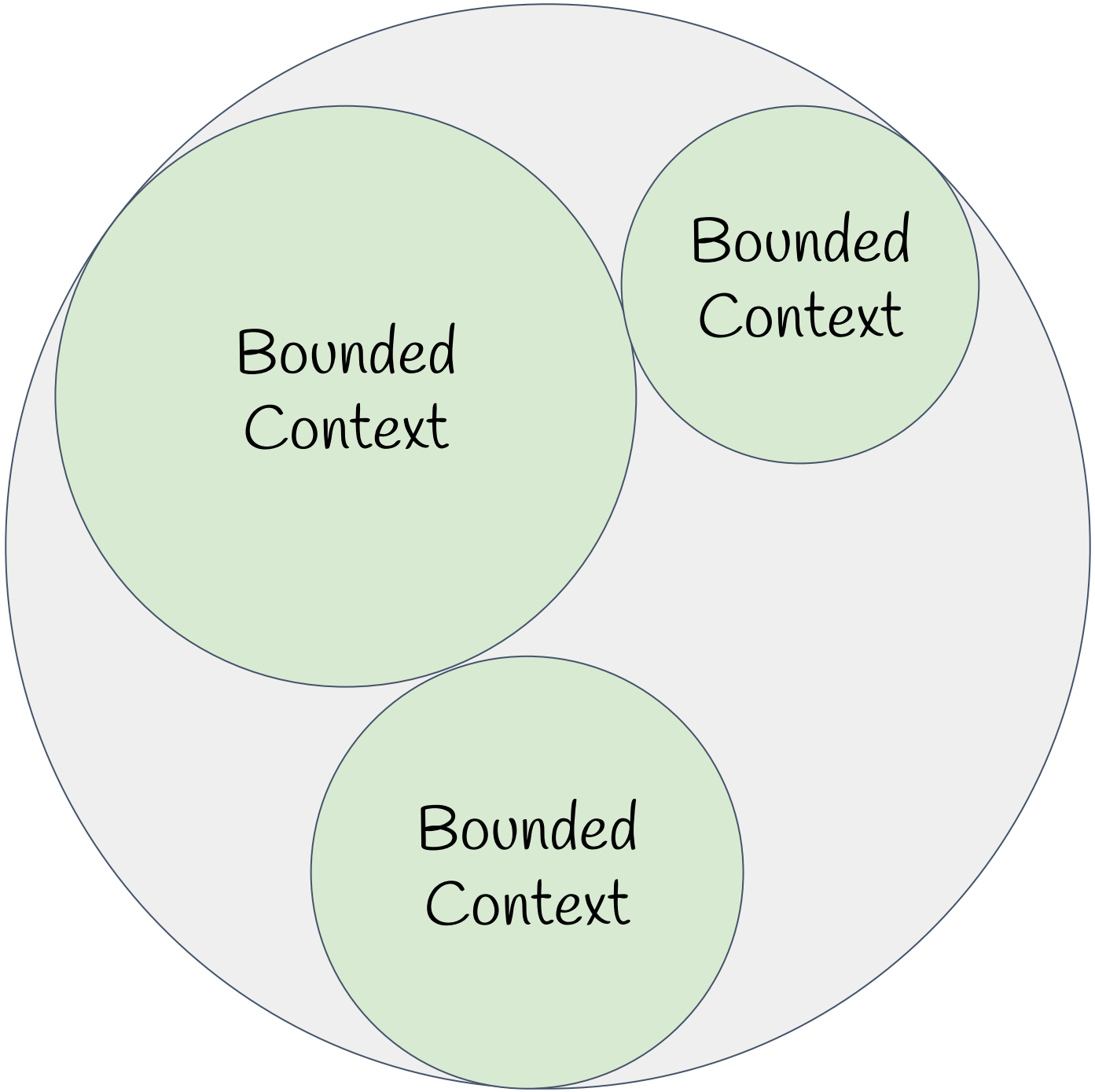

Aggregate
Aggregate
Bounded
Context

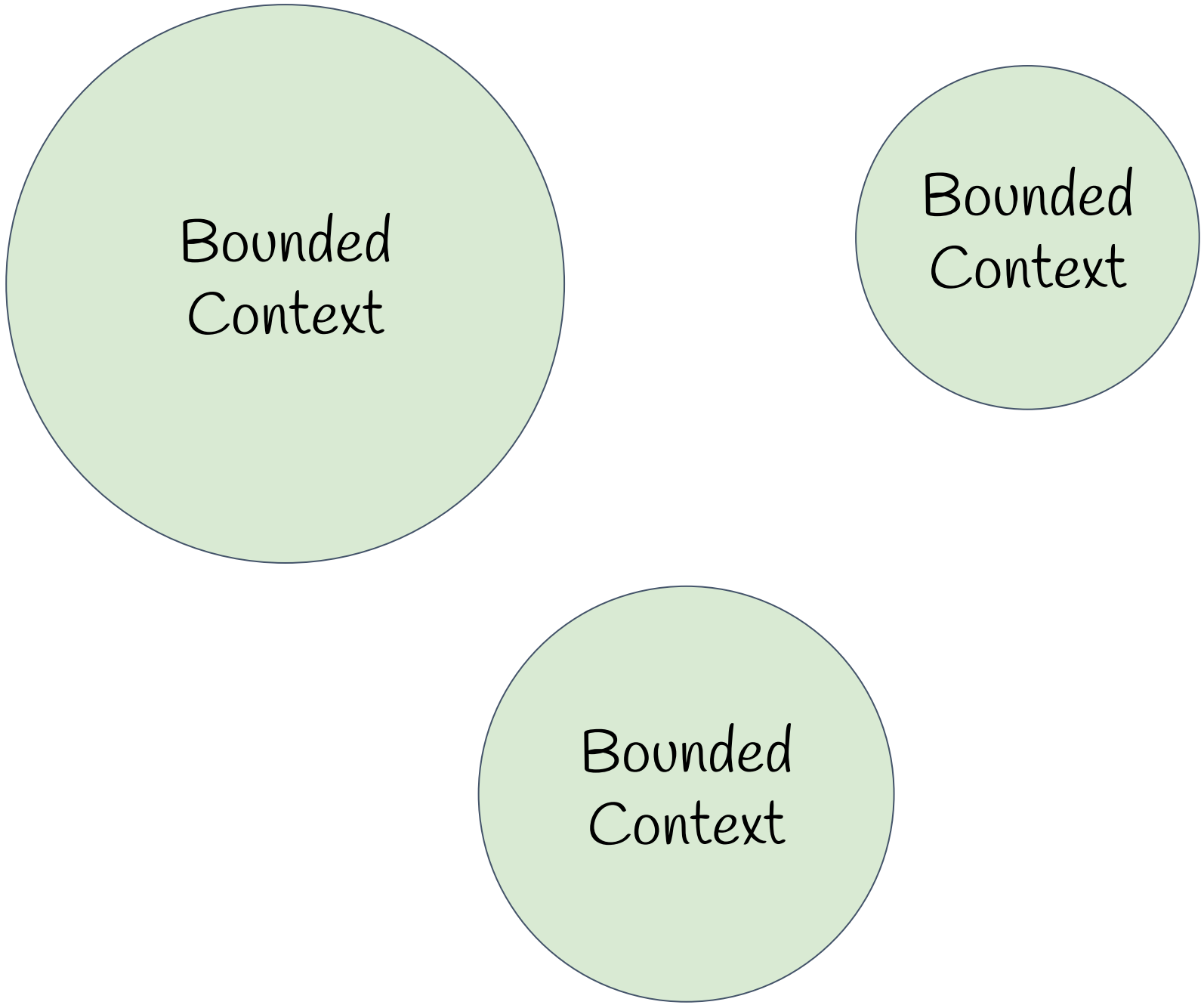

Aggregate
Aggregate
Aggregate
Aggregate
Bounded Context

Как делить Domain Model на Bounded Context?

По границам агрегатов

Как делить ~~Domain Model~~ монолит на ~~Bounded Context~~ микросервисы?

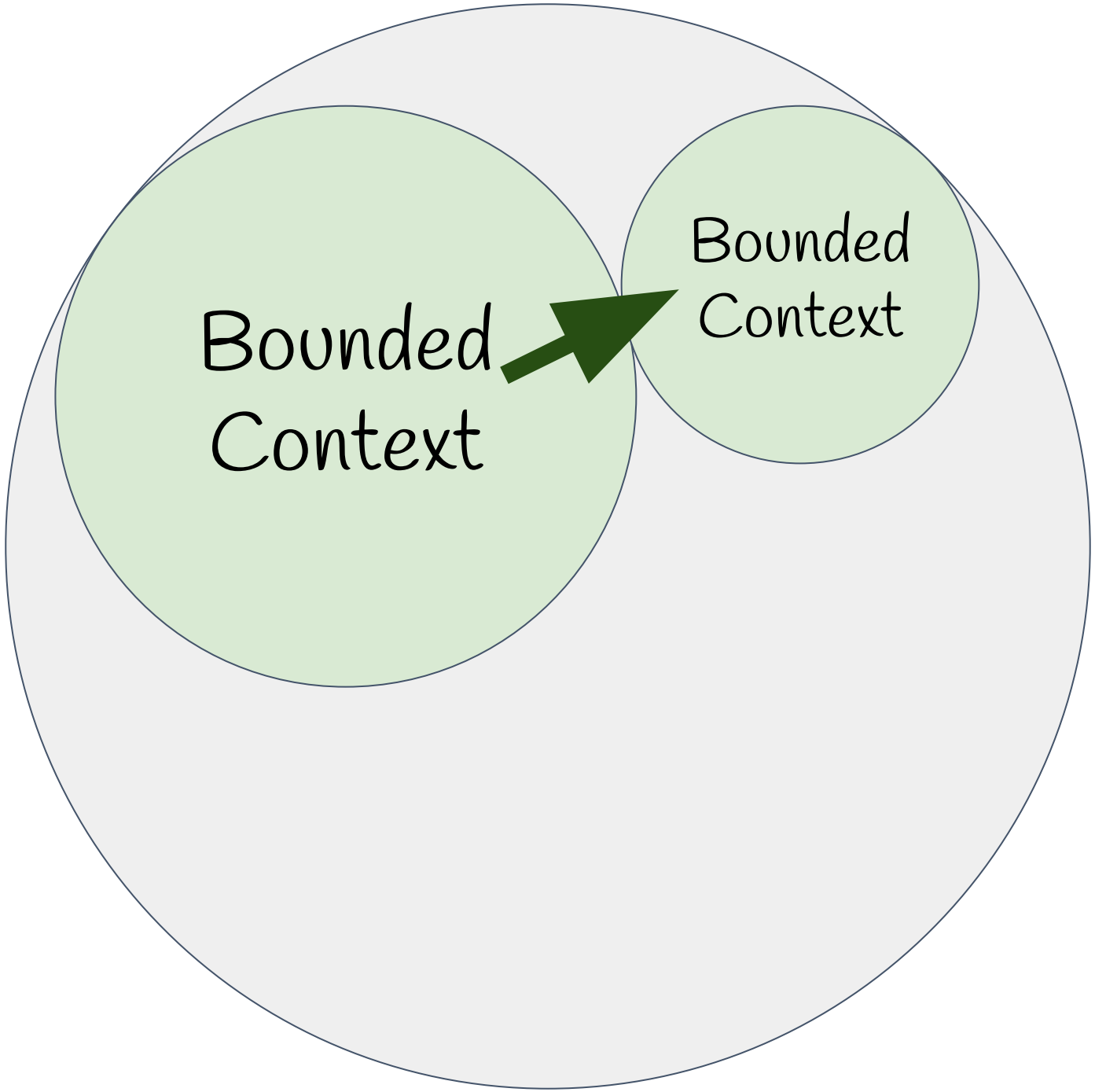

Bounded Context
Bounded Context
Bounded Context

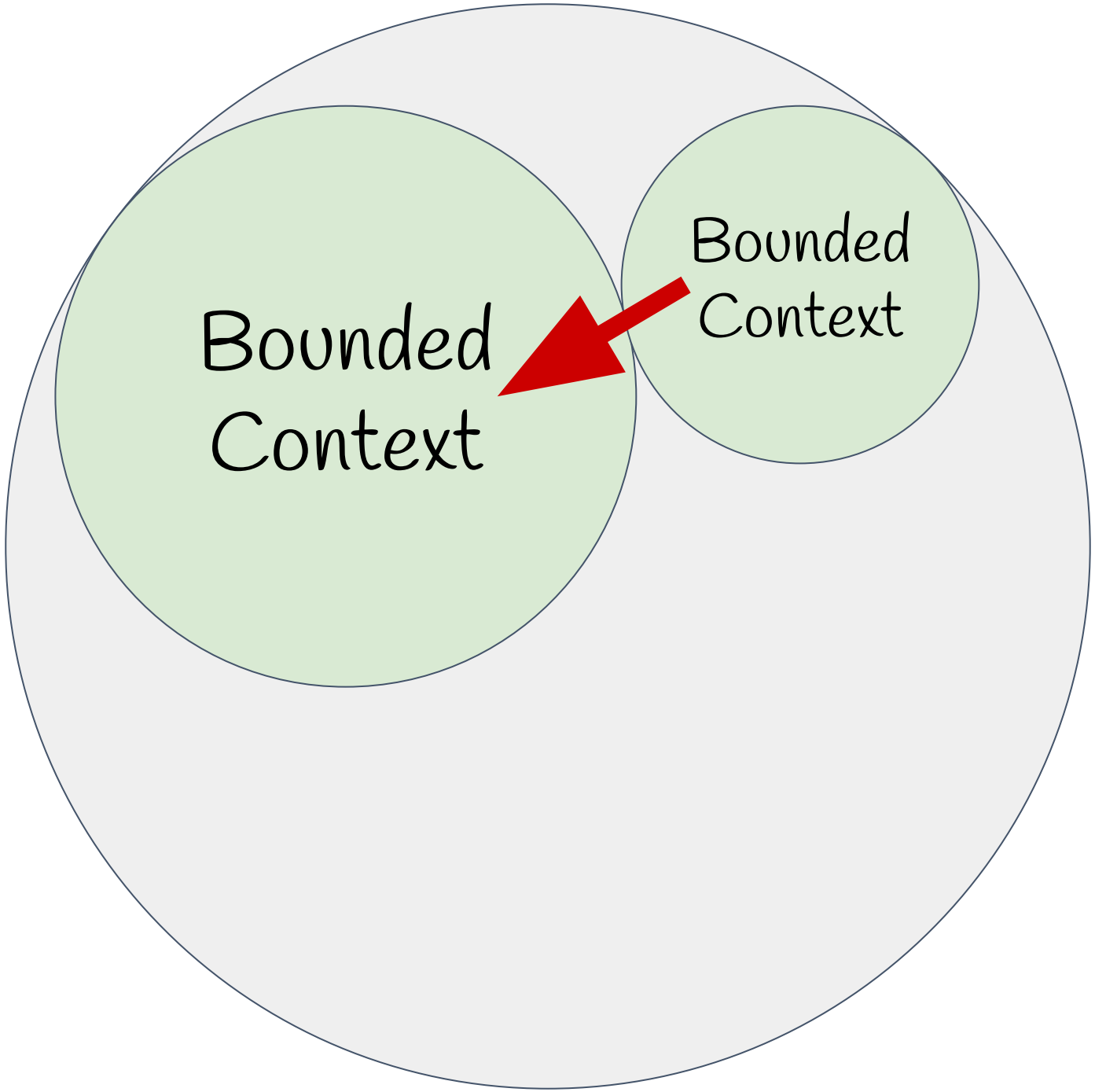

Bounded Context
Bounded Context
Bounded Context

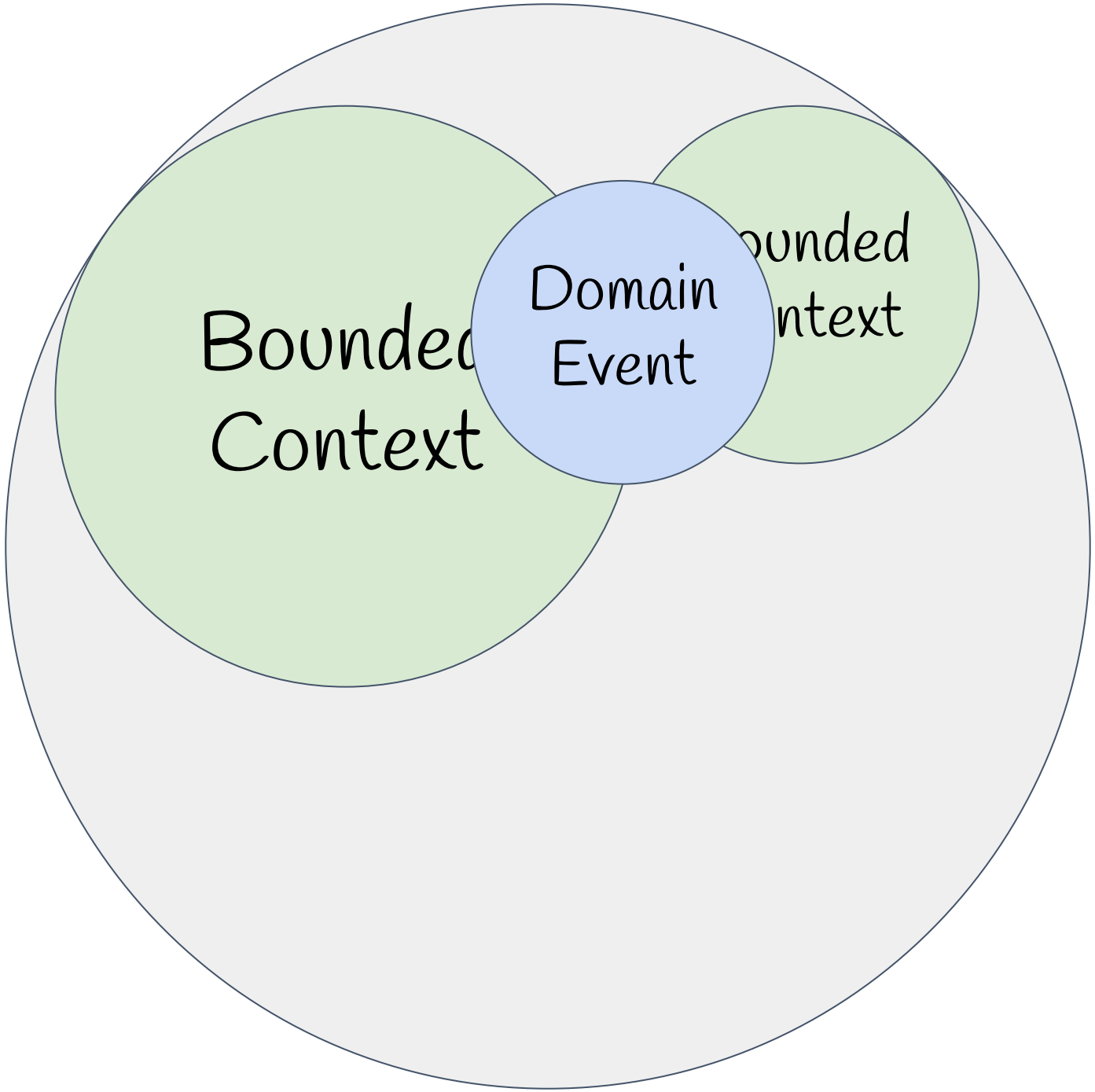
Как делить ~~Domain Model~~ монолит на ~~Bounded Context~~ микросервисы?

По границам контекстов

Как пересечь границу контекста?


Bounded
Context
Bounded
Context


Bounded
Context
Bounded
Context


Bounded
Context
Domain
Event
ounded
ntext

Как пересечь границу контекста?

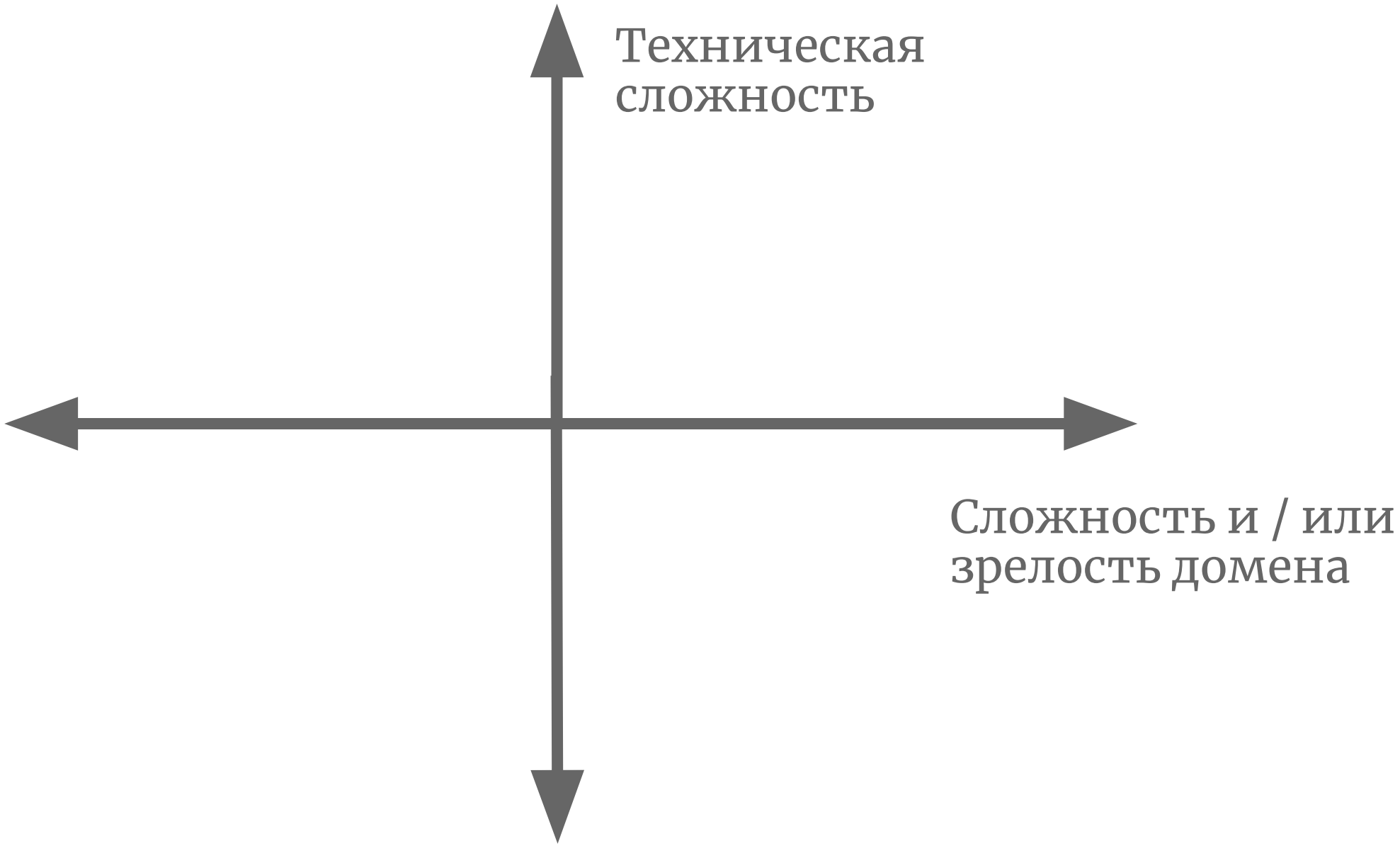
Использовать события

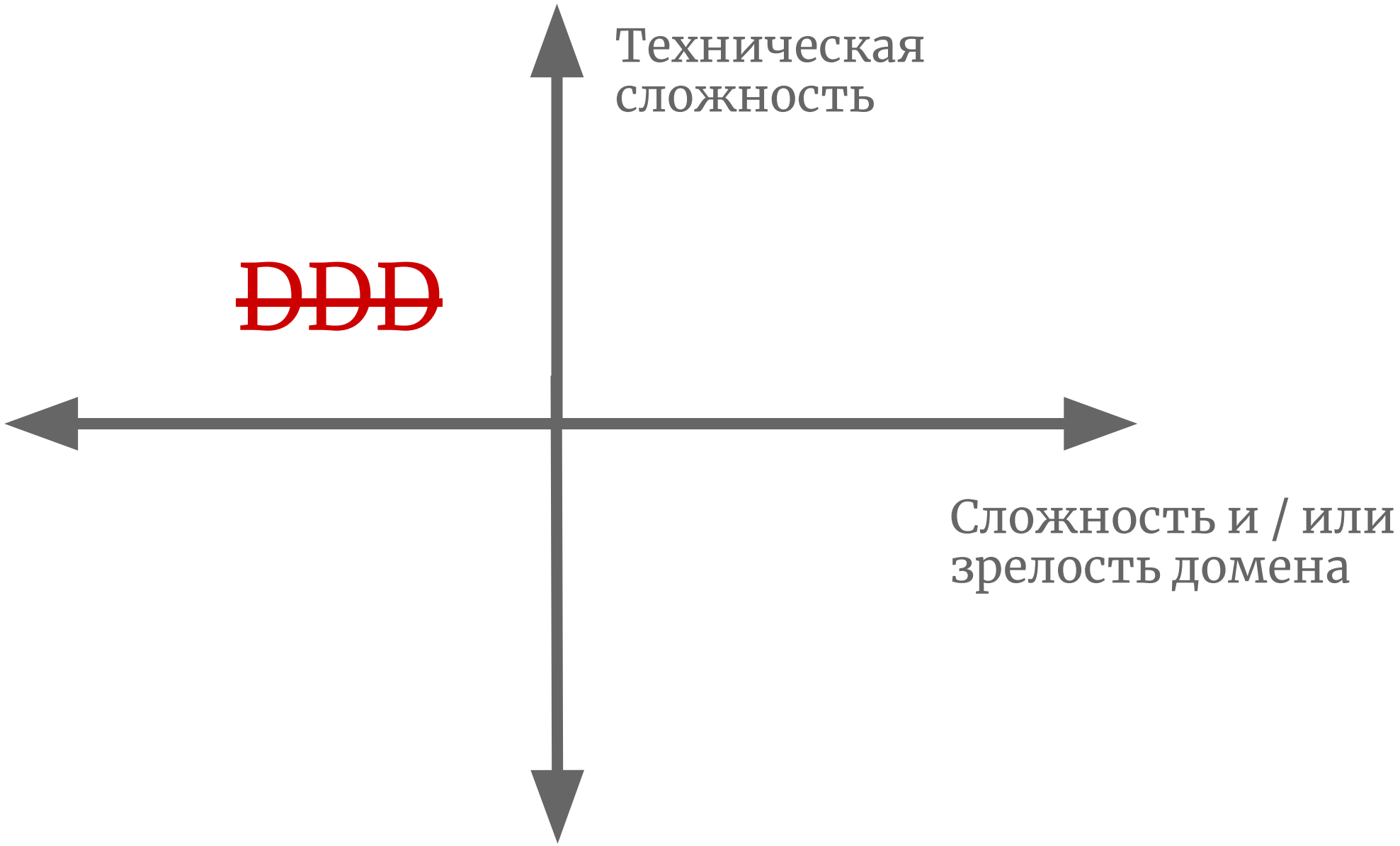
Что выбрать?

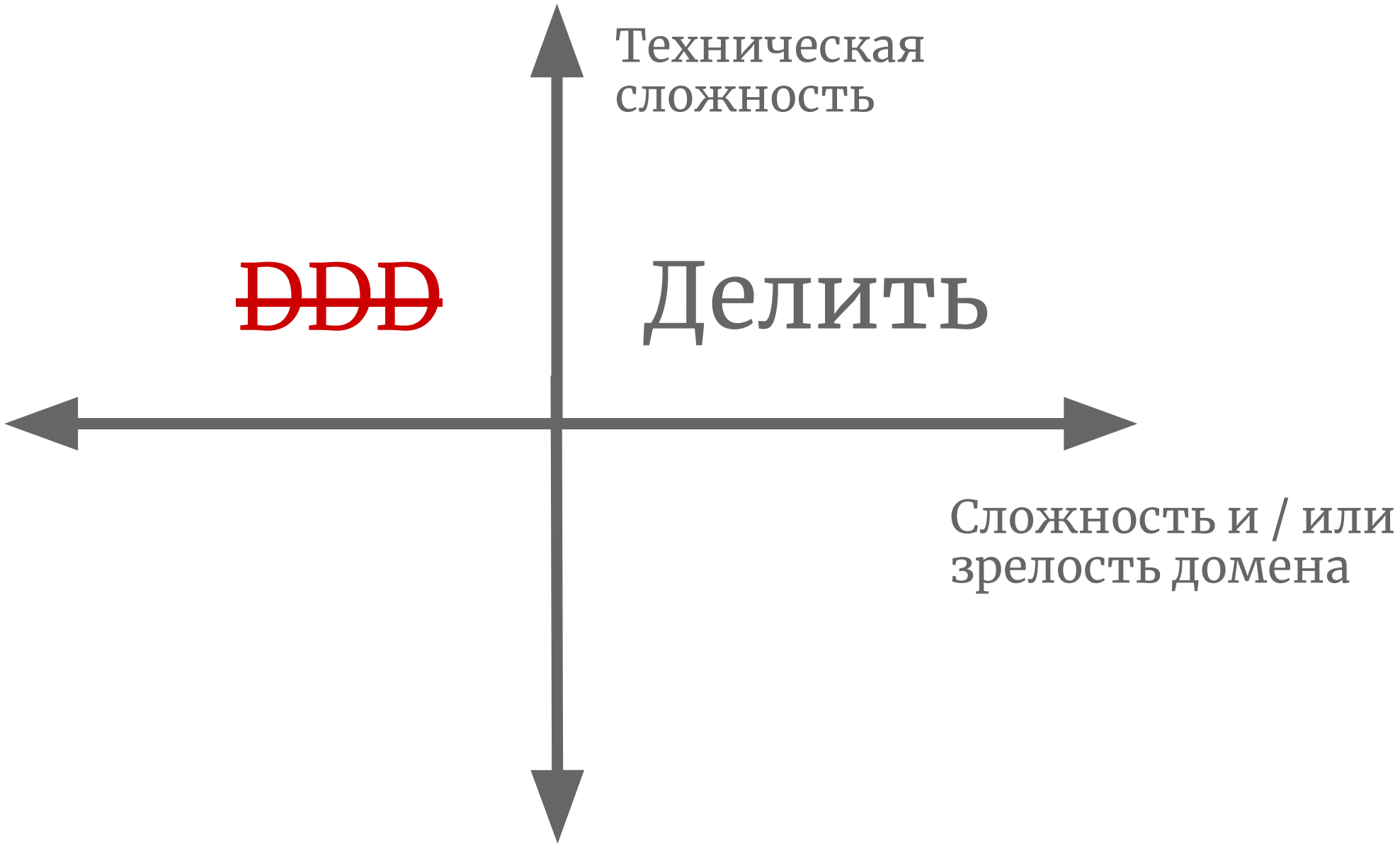
DDD - это дорого

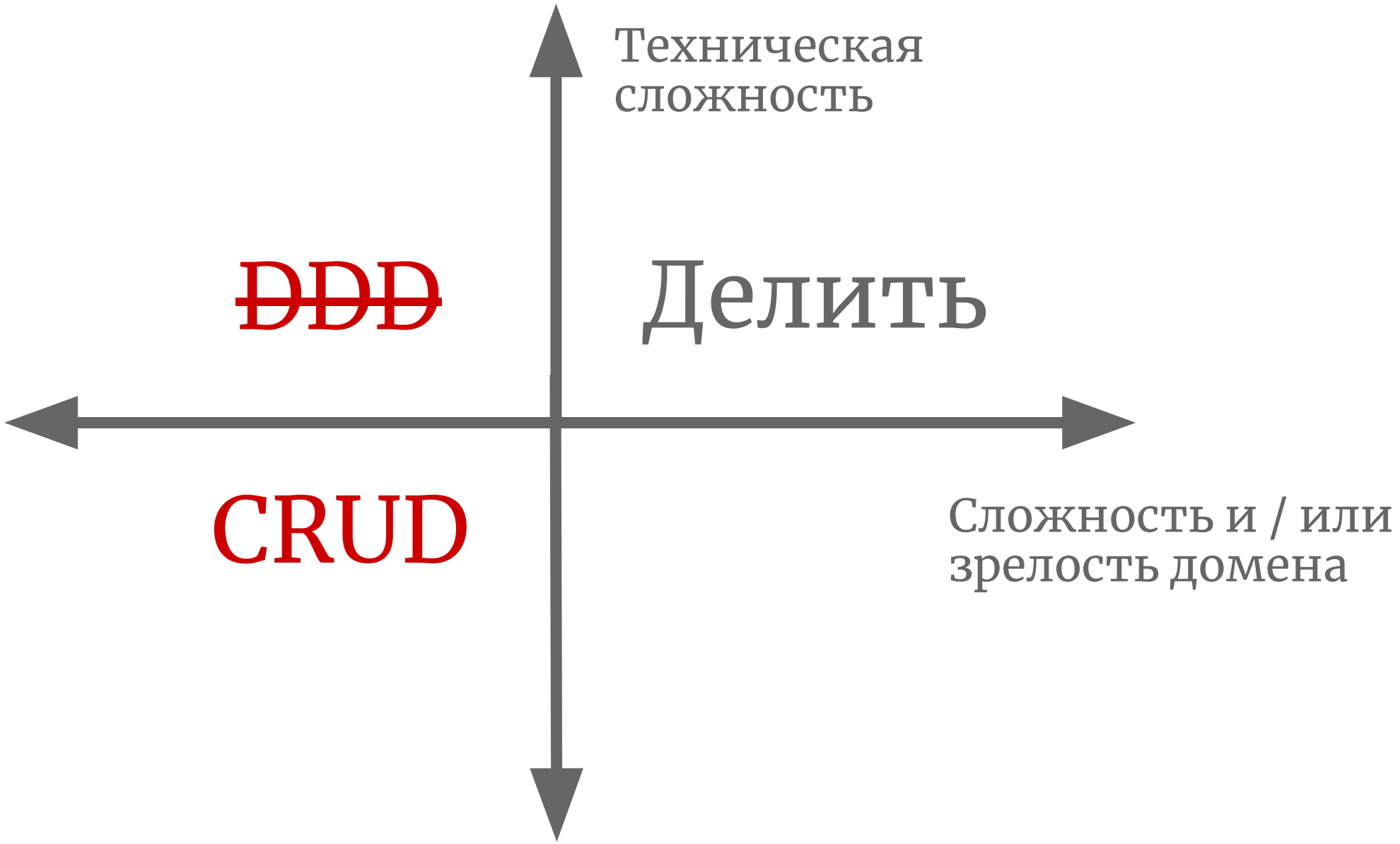
Что выбрать?

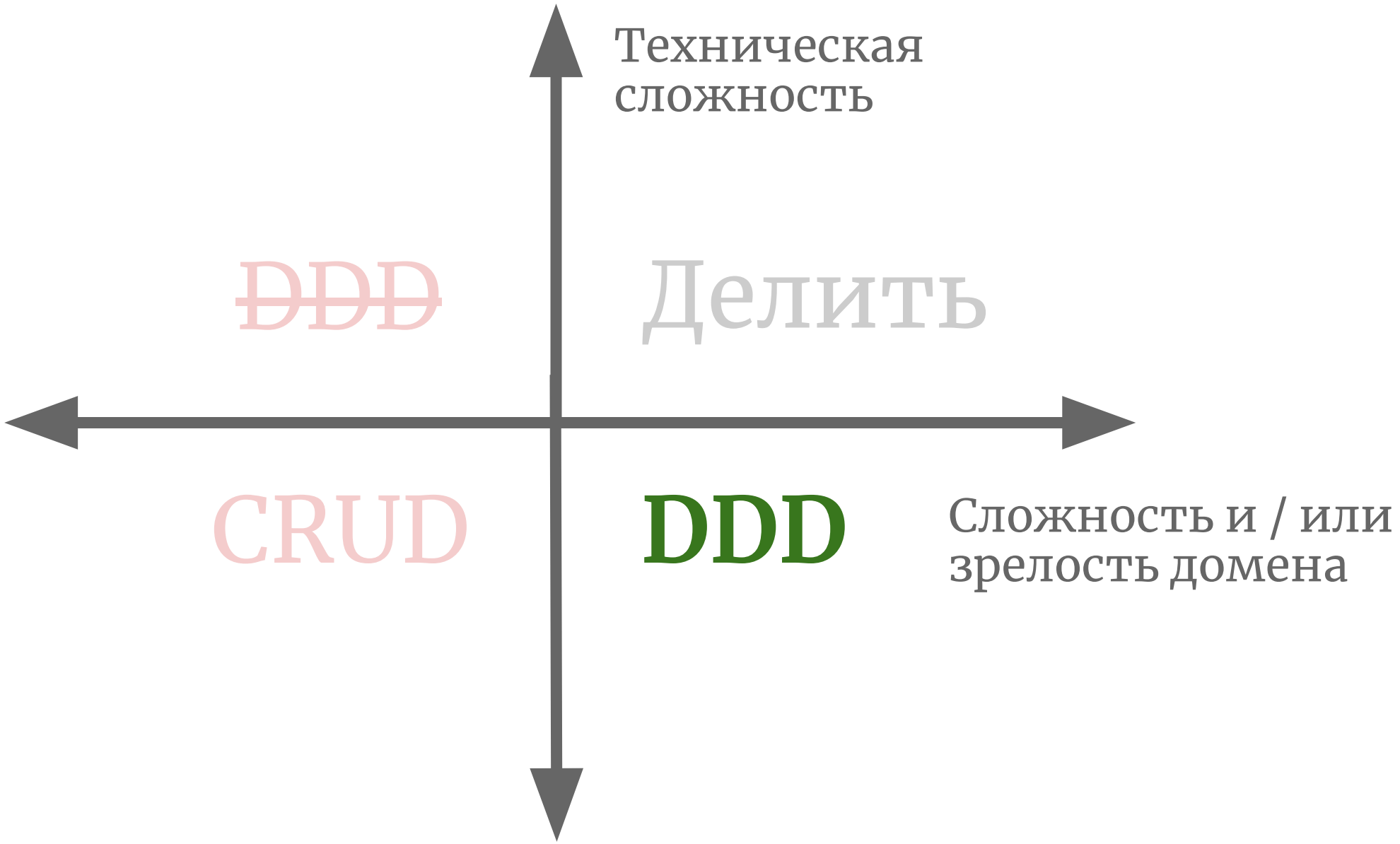
DDD - это **дорого**


Техническая
сложность
Сложность и / или
зрелость домена


Техническая
сложность
~~DDD~~
Сложность и / или
зрелость домена


Техническая
сложность
~~DDD~~
Делить
Сложность и / или
зрелость домена


Техническая
сложность
~~DDD~~
Делить
CRUD
Сложность и / или
зрелость домена


Техническая
сложность
~~DDD~~
Делить
CRUD
DDD
Сложность и / или
зрелость домена

Эволюционный рефакторинг

Анемичная -> Богатая

- Инкапсуляция
- Спецификации
- Entity и Value Objects

- Aggregate
- Bounded Context
- Pattern Matching

- Массовые операции
- События
- IHas-интерфейсы

Анемичная -> Богатая

- Инкапсуляция
- Спецификации
- Entity и Value Objects

- Aggregate
- Bounded Context
- Pattern Matching

- Массовые операции
- События
- IHas-интерфейсы

Анемичная -> Богатая

- Инкапсуляция
- Спецификации
- Entity и Value Objects

- Aggregate
- Bounded Context
- Pattern Matching

- Массовые операции
- События
- IHas-интерфейсы

Хорошо дополняют доклад

По-русски

1. Деревья выражений в Enterprise-разработке
2. Быстрорастворимое проектирование
3. Жизнь после бизнес-объектов
4. Domain-driven design: рецепт для прагматика
5. Сущности в DDD-стиле с Entity Framework Core
6. Шпаргалка по основам DDD

In English

7. Tackling Complexity in the Heart of Software
8. Domain Modeling Made Functional with the F# Type System
9. Don’t delete
10. Design Smell: Redundant Required Attribute
11. EF Core 2.1 vs NHibernate 5.1: DDD perspective
12. Value Object: a better implementation

Вопросы

https://hightech.group/ru/ddd