

TECHODRAMA

— MAY 12-13 2015 —

Domain-Driven Design for the Database-Driven Mind

Julie Lerman @julielerman thedatafarm.com

Building Data-Centric Apps with
the ADO.NET Entity Framework

2nd Edition
Thoroughly Revised for EF4

Programming

Entity Framework

O'REILLY®



Creating and Configuring Data Models from Your Classes

Programming Entity Framework

Code First

O'REILLY®



Julia Lerman & Rowan Miller

Querying, Changing, and Validating
Your Data with Entity Framework

Programming Entity Framework

DbContext

O'REILLY®



Julia Lerman
& Rowan Miller

My Courses on pluralsight

- Domain-Driven Design Fundamentals
- Looking Ahead to Entity Framework 7
- EF6 Ninja Edition: What's New in EF6
- Automated Testing for Fraidy Cats Like Me
- Getting Started with Entity Framework 5
- Entity Framework in the Enterprise
- Entity Framework Code First Migrations
- Data Layer Validation with Entity Framework 4.1+
- Entity Framework 4.1 - DbContext Data Access
- Entity Framework 4.1 - Code First
- Querying the Entity Framework
- Designer Supported EDM Customization
- Entity Framework and Data Models
- Entity Framework 4.0 By Example

TECHORAMA



Forget persistence?

What about shared
data & types?

How will DDD
patterns resolve
with EF?

Yes You Can

Design with DDD, Today & Persist data with EF, Tomorrow

TECHORAMA

Your Workflow?



BreakAway Geek Adventures Contacts

⏪ ⏩ ⏴ ⏵ + X Save

Title: Ms.
First Name: Wilma
Last Name: Flintstone
Modified Date: 2/4/2010
Add Date: 2/4/2010
Primary Activity: ComboBox
Secondary Activity: ComboBox
Primary Destination: ComboBox
Notes: Some text
A second line of text
Reservations

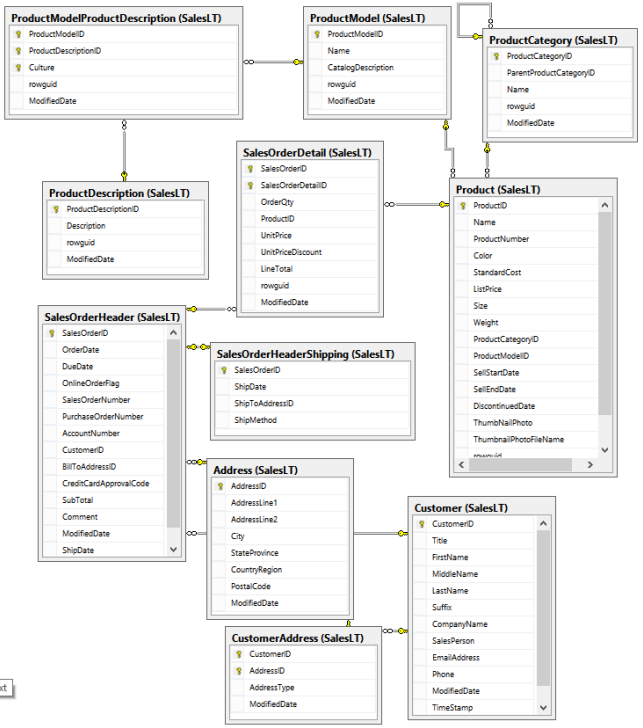
Res. Date	Start Date	End Date
2/1/2010	4/2/2010	4/2/2010

Manage Trips

Trip to	Start Date
Alaska	1/5/2010
Amazon	7/24/2008
Australia	3/2/2010
Belize	5/10/2009
Chile	2/1/2008
Costa Rica	6/8/2007

Start Date: 3/10/2010 End Date: 3/24/2010
Destination: Australia Lodging: In and Outback Inn
Activities on this trip
Item One
Item Two
Item Three
Activities: Horseback Riding
Add Activity to Trip

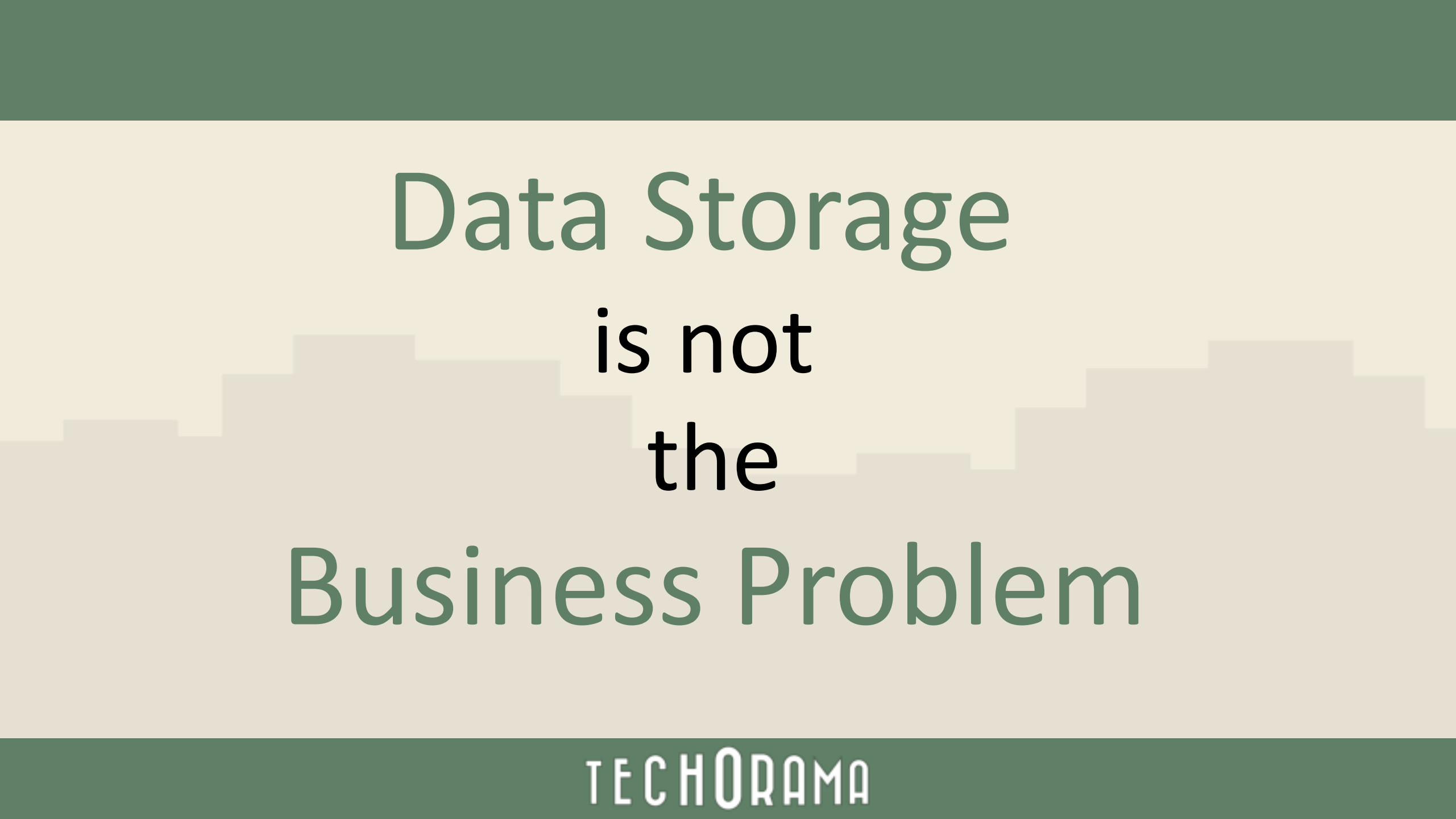
Software Solution





Focus on the Business Problem

TECHORAMA



Data Storage is not the Business Problem

TECHORAMA



Domain
Problem

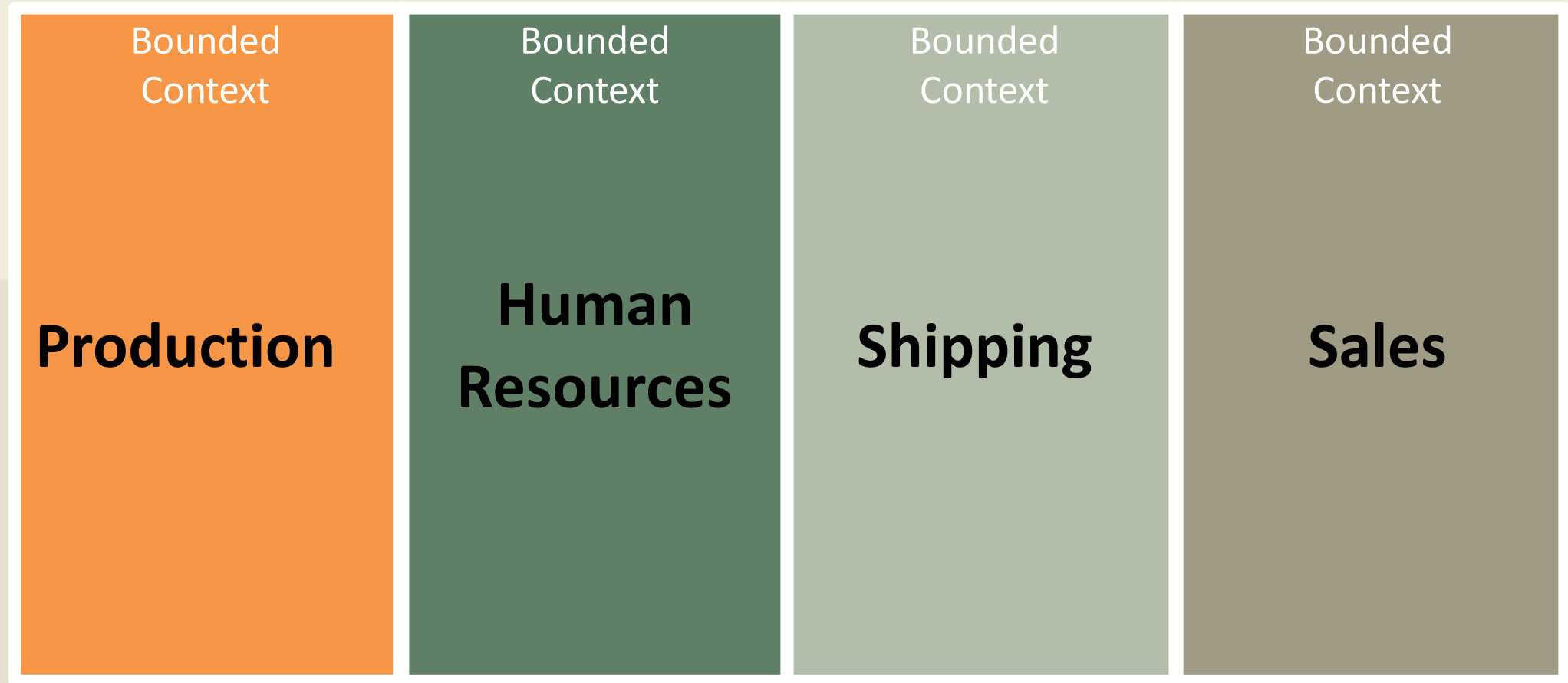
Cross-Cutting Concerns

Data
Persistence

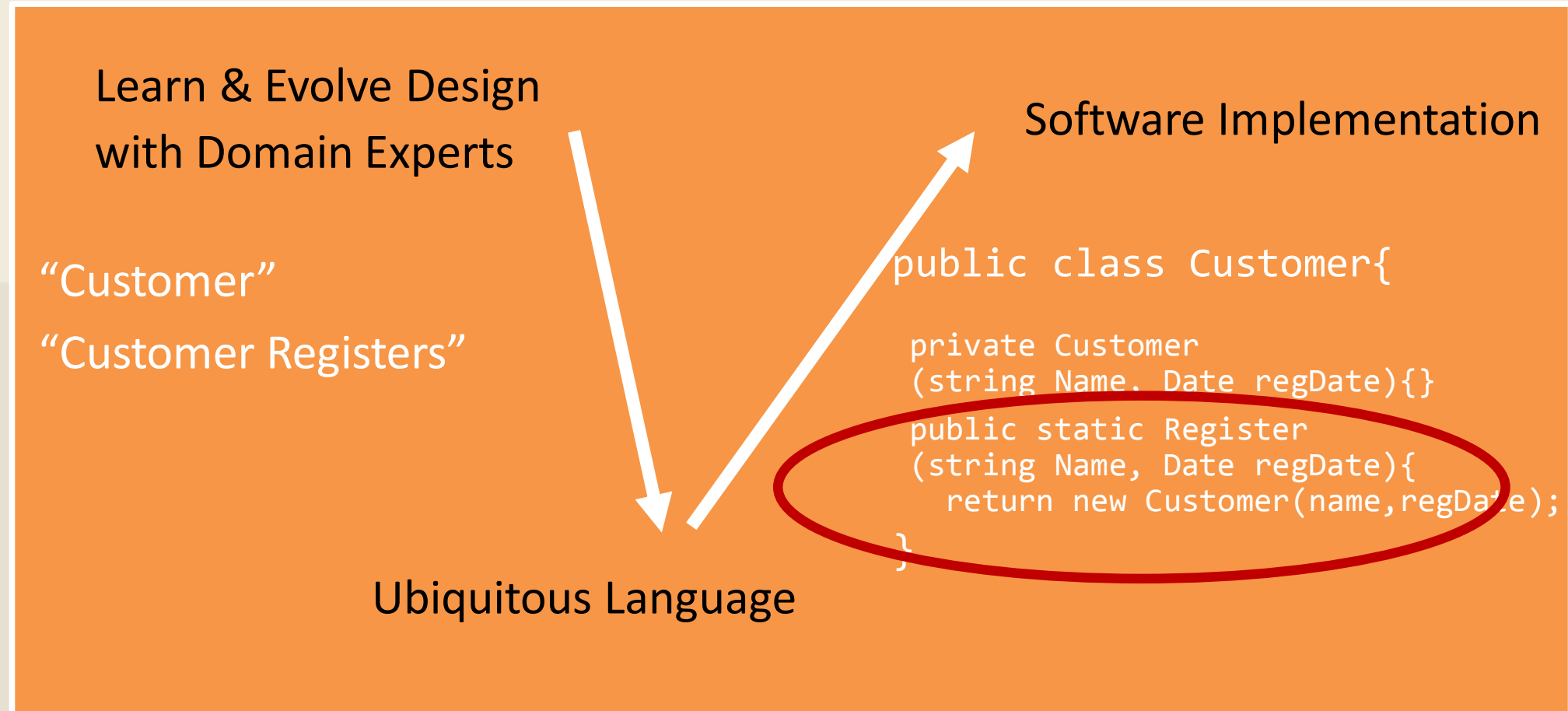
TECHORAMA

Bounded Context & Reuse

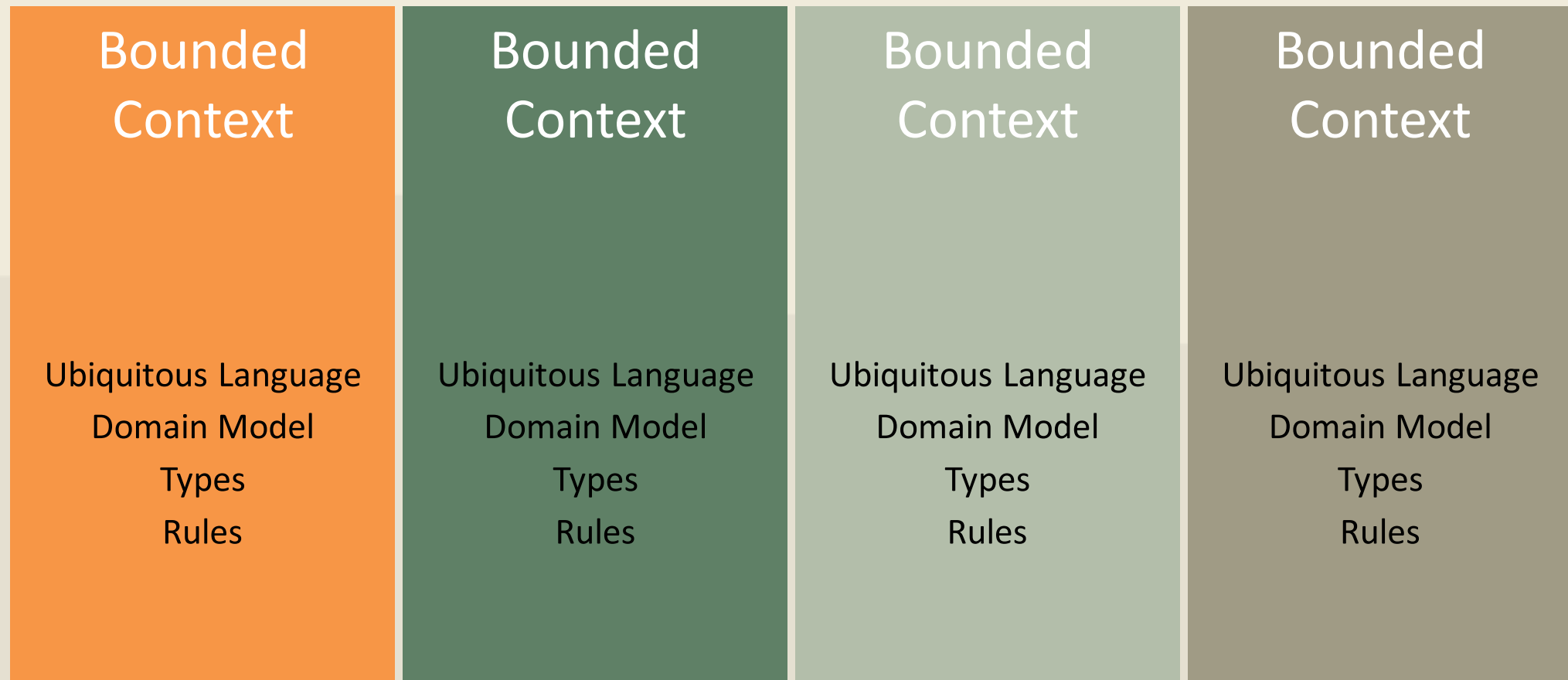
DDD Thinking: Divide and Conquer



Bounded Context



Domain-Driven Design Thinking



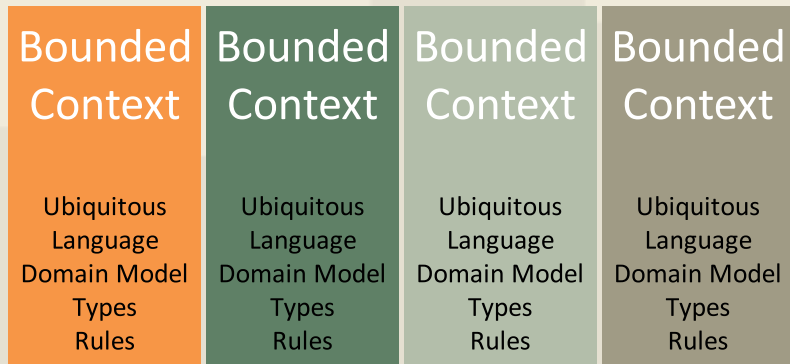
Shared Data?
Shared Types?



PANIC!

TECHORAMA

Data Sharing Patterns & Anti-Patterns



Bad (No-Op)

Read/Write from BCs to common tables

Good

Read reference data from shared tables

Write BC types to different tables

Better

Each BC persists to its own tables

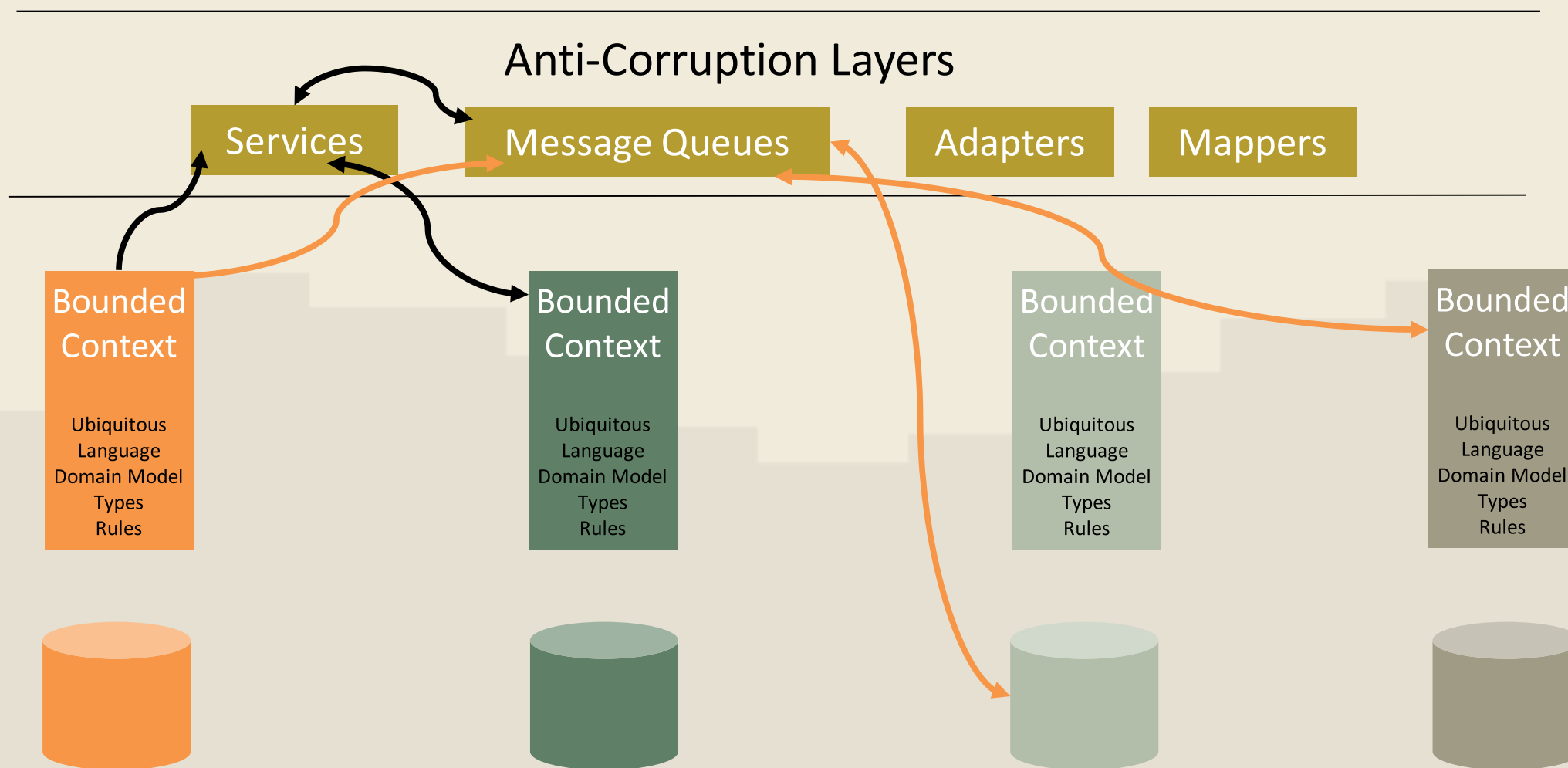
Best

Database or DB Schema per model

Even Better than Best

Event Sourcing (rather than persist state)

ACLs Between Bounded Contexts, Too



When to Share Types

Shared Kernel

Tightly coordinated Entities and Value Objects

Common schema and behavior

“Reduce duplication, but not eliminate it” (*Eric Evans, DDD book*)

Inheritance

Infrastructure

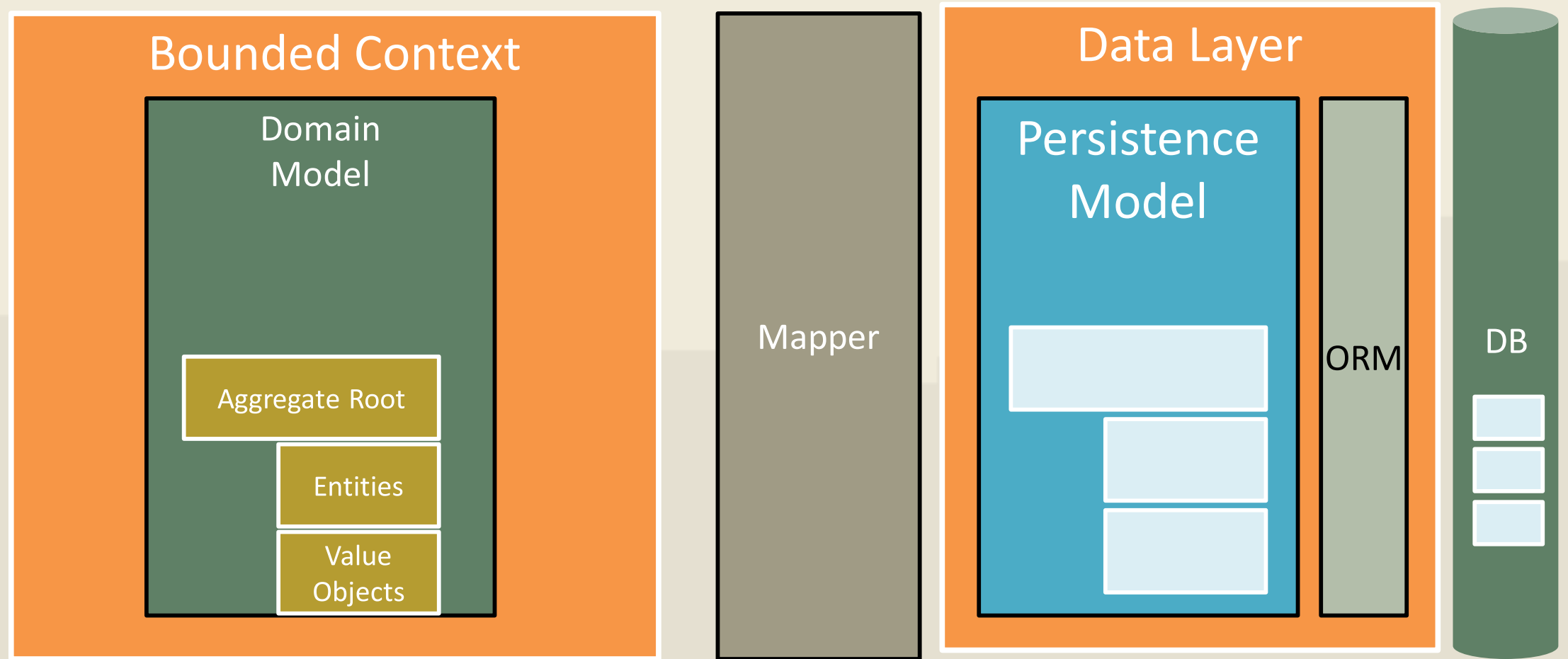
Not domain types

Favor composition over inheritance

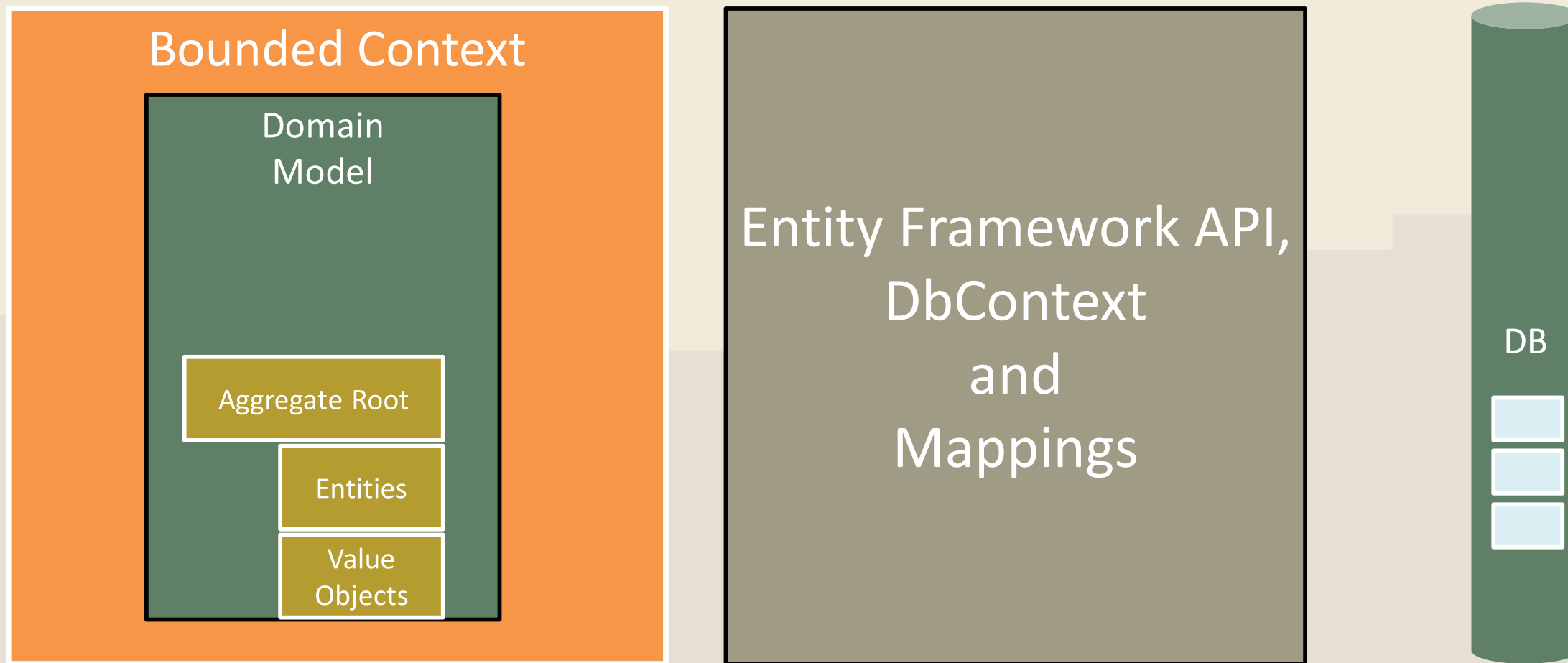
A Question of Models

TECHORAMA

Domain Model and Persistence Model



Domain as Persistence Model



DDD Entity Design Patterns & Entity Framework

Rich Domain Models over Anemic Types

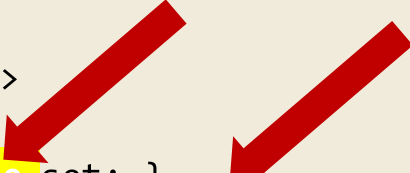
Anemic for CRUD

```
public class Patient
{
    public int Id { get; set; }
    public Client Owner { get; set; }
    public int ClientId { get; set; }
    public string Name { get; set; }
    public Gender Gender { get; set; }
    public int? DoctorId { get; set; }
}
```

Rich for DDD

```
public class Schedule : Entity<Guid>
{
    public int ClinicId { get; private set; }
    public DateTimeRange DateRange { get; private set; }
    private List<Appointment> _appointments;
    public IEnumerable<Appointment> Appointments {...}
    public Schedule(Guid id, DateTimeRange dateRange,
        int clinicId, IEnumerable<Appointment> appointments)
        : base(id) {
        ...
        MarkConflictingAppointments();
        DomainEvents.Register<AppointmentUpdatedEvent>(Handle);
    }

    public Appointment AddNewAppointment(Appointment appt) {
        if (_appointments.Any(a => a.Id == appt)) {
            throw new ArgumentException("Duplicate", "appointment"); }
        appt.State = TrackingState.Added;
        _appointments.Add(appt);
    }
}
```



DDD Private Setters & Constructors & EF

How can EF Materialize objects/properties with private setters?

Reflection FTW!

Except those cranky collections

TECHORAMA

Favor One-Way Relationships



TECHORAMA

Has This Happened to You?

Utah

Vermont

Virginia

Washington

West Virginia

Wisconsin

```
myNewAddress.State=(State)myDropDownBox.SelectedItem;  
myNewAddress.StateId=myDropDownBox.SelectedValue; //e.g. 17  
myContext.Addresses.Add(myAddress);  
myContext.SaveChanges();
```

Utah

Vermont

Vermont

Virginia

Washington

West Virginia

“

A bidirectional association means that both objects can be **understood only together**.

When application requirements do not call for traversal in both directions, adding a traversal direction reduces interdependence and **simplifies the design**.

”

- Eric Evans

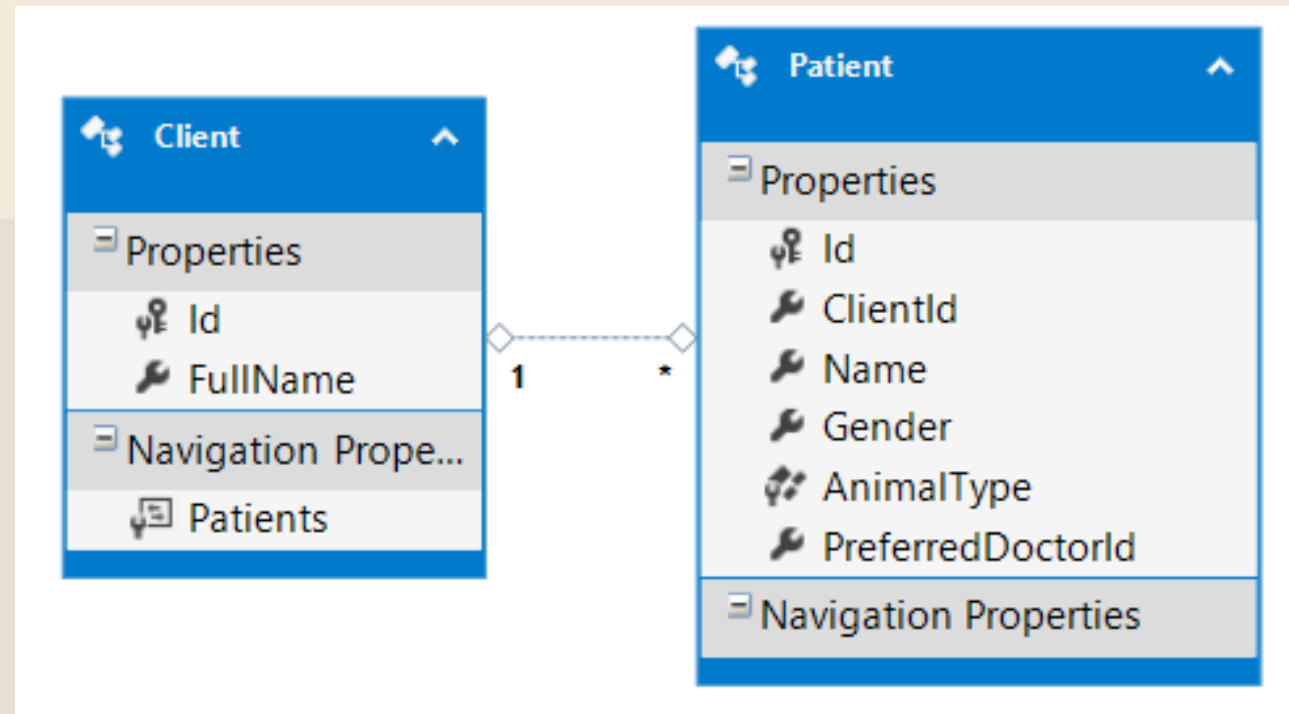
DDD One-Way Relationships/Navigations & EF

EF does not require navigation properties

Except for those pesky 1:1 relationships

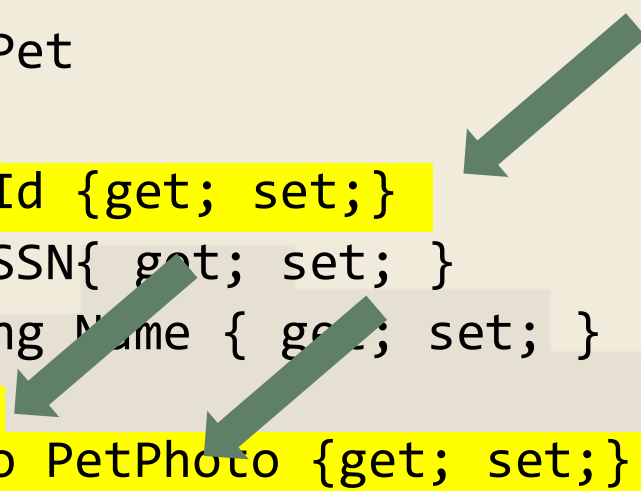
Many tasks are simpler with FKs

E.g. reference properties

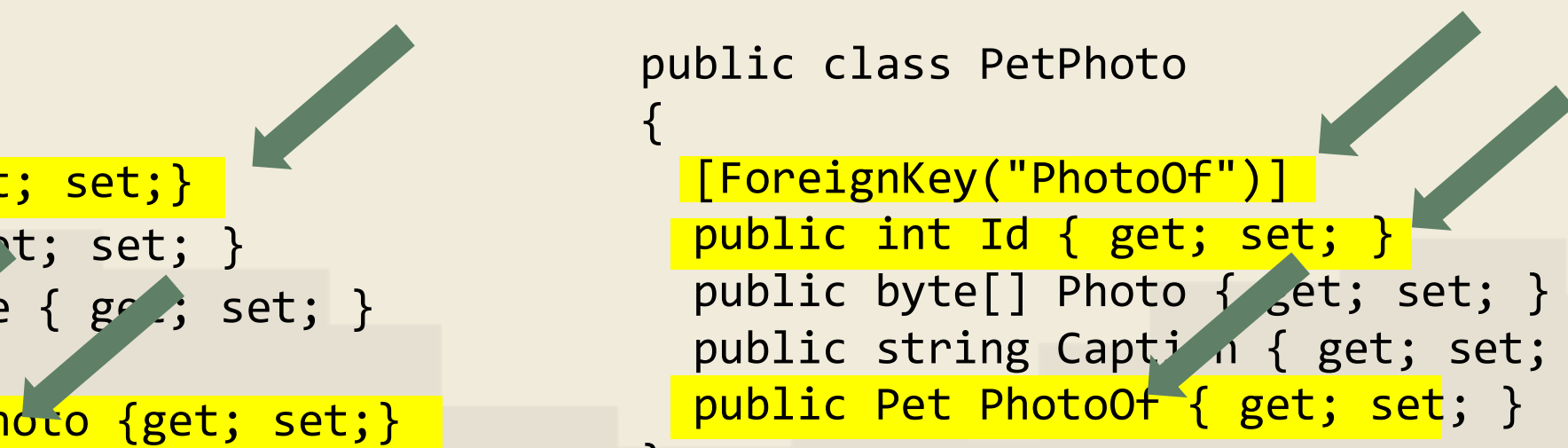


EF Makes Demands on 1:1 Relationships

```
public class Pet
{
    public int Id {get; set;}
    public int SSN{ get; set; }
    public string Name { get; set; }
    [Required]
    public Photo PetPhoto {get; set;}
}
```



```
public class PetPhoto
{
    [ForeignKey("PhotoOf")]
    public int Id { get; set; }
    public byte[] Photo { get; set; }
    public string Caption { get; set; }
    public Pet PhotoOf { get; set; }
}
```



- Shared Primary Key e.g. FK is the PK
- Navigation Properties on both ends
- Annotation/Fluent API to define principal & dependent

DDD: Favor Value Objects Over 1:1 Relationships

Value Object Attributes

- Immutable
- Combo of all values == identity
- No side effects
- Ability to compare using all values

Easy V.O. Types

- Measurements 3&Feet, 2&Meters
- Financial Value 100&USD, 100&€

```
public class PetPhoto{
    public byte[] Photo { get; private set; }
    public string Caption { get; private set; }
    public PetPhoto(byte[] photo, string caption)
    {
        Photo = photo;
        Caption = caption;
    }
    public override bool Equals(object obj)
    {
        var otherType = obj as PetPhoto;
        if (otherType == null)
        { return false; }
        return Photo == otherType.Photo
            && Caption == otherType.Caption;
    }
    public override int GetHashCode() { . . . }
}
```

Favor Explicit Updates over EF Update Magic



EF relationship magic

Insert/update FK fix-up

Eager/Lazy/Explicit loading

Helpful for traversing in code



Nav. props not always desired

Cause confusing EF side effects

Borrow from CQRS* to Balance Navigations

Read model

- Include helpful navigation properties

Write model

- Eliminate navigation properties & their side-effects

- Control relationships via FK properties

[*CQRS Journey from MS Patterns & Practices: bit.ly/cqrsjourney]

Some Downsides with EF & DDD

Nullability concerns with value objects

Jimmy Bogard concludes that when using the domain model directly with EF, just say No to value objects

Cannot hide properties completely

Scenarios where mapping to fields is preferable

Yes You Can

- Build great software & reduce complexity by focusing on domain not data persistence
- Recover from years of “reuse everything” mantra
- Leverage patterns to share data stored in different locations
- Learn to divide and conquer. Smaller problems are usually easier to solve.

Yes You Can

Design with DDD, Today & Persist data with EF, Tomorrow

TECHORAMA

Resources

- Domain-Driven Design Fundamentals on Pluralsight bit.ly/PS-DDD
- All of my Pluralsight courses: pluralsight.com/author/julie-lerman
- Domain-Driven Design, Eric Evans amzn.to/1kstiRg
- Implementing Domain-Driven Design, Vaughn Vernon amzn.to/1dgYRY3
- Domain Modeling with Entity Framework Scorecard, Jimmy Bogard, bit.ly/1x925bu
- Coding for Domain-Driven Design: Tips for Data-Focused Devs (3 Parts)
MSDN Magazine, Aug, Sept & Oct 2013 bit.ly/15xMIDL
- CQRS Journey from MS Patterns & Practices: bit.ly/cqrsjourney
- **A Pattern for Sharing Data Across Domain-Driven Design Bounded Contexts (Part 1&2)**
MSDN Magazine, Oct & Dec 2014: bit.ly/DataPoints_Dec2014
- **Entity Framework Model Partitioning in Domain-Driven Design**

Contact Information

Julie Lerman

Email: julielerman@gmail.com

Web: theDataFarm.com

Twitter: [@julielerman](https://twitter.com/julielerman)