Incorporating Domain Events and Anti-corruption Layers

Steve Smith

Force Multiplier for
Dev Teams

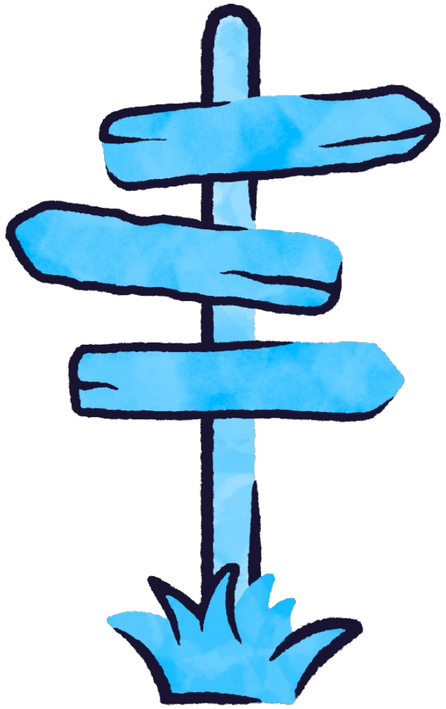
@ardalis | ardalis.com

Julie Lerman

Software Coach,
DDD Champion

@julielerman | thedatafarm.com

Module Overview



Introducing domain events

Identifying domain events

Designing domain events

A simple example

Domain events in our application

Introducing anti-corruption layers

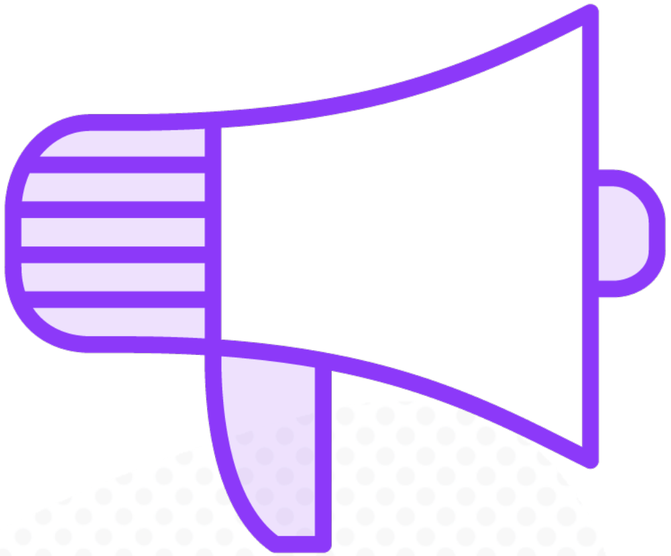
Introducing Domain Events

Domain Events

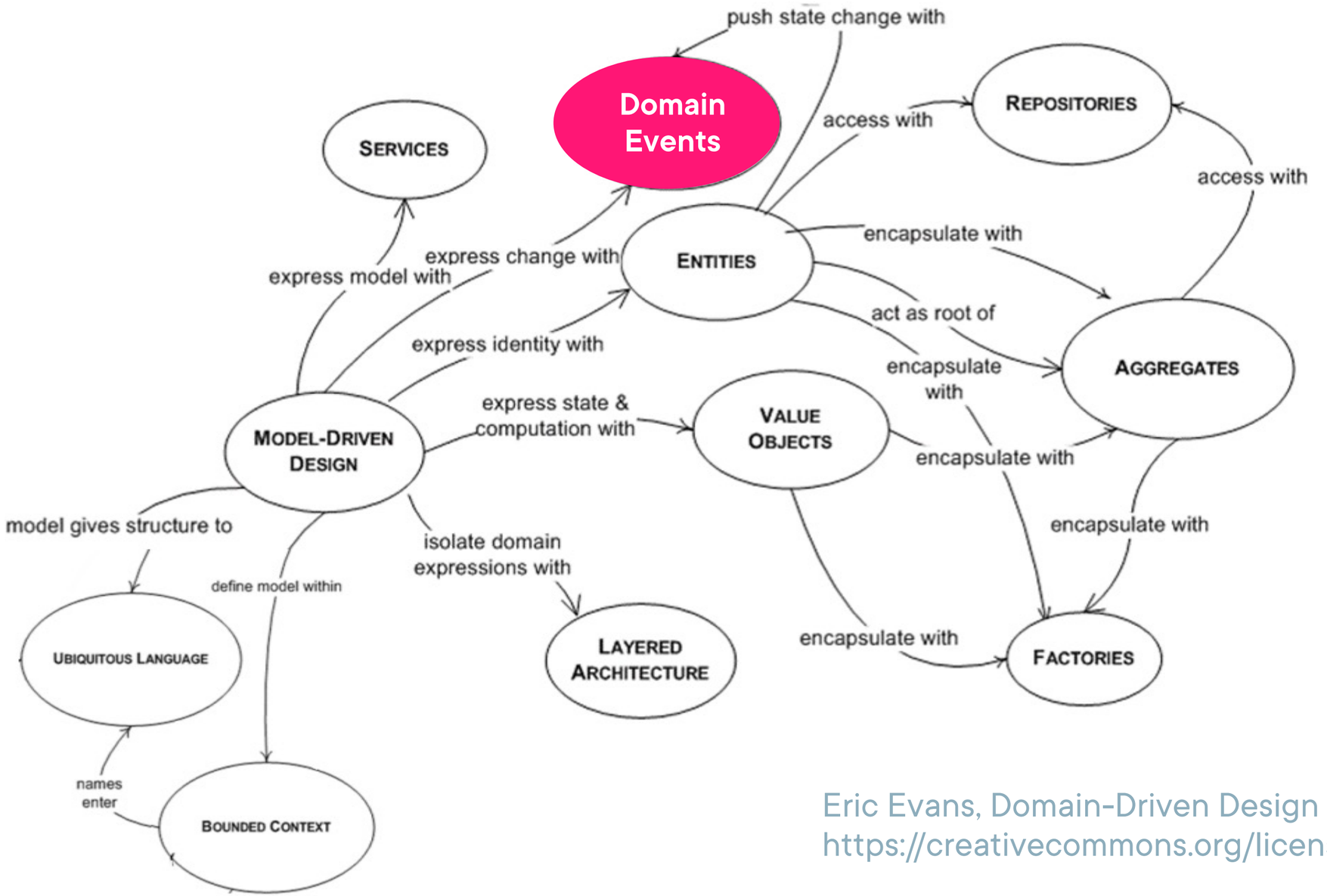
Alert that some activity occurred

Or some state changed in the context

Other domains can subscribe to the “news”



Domain Events within the DDD Mind Map



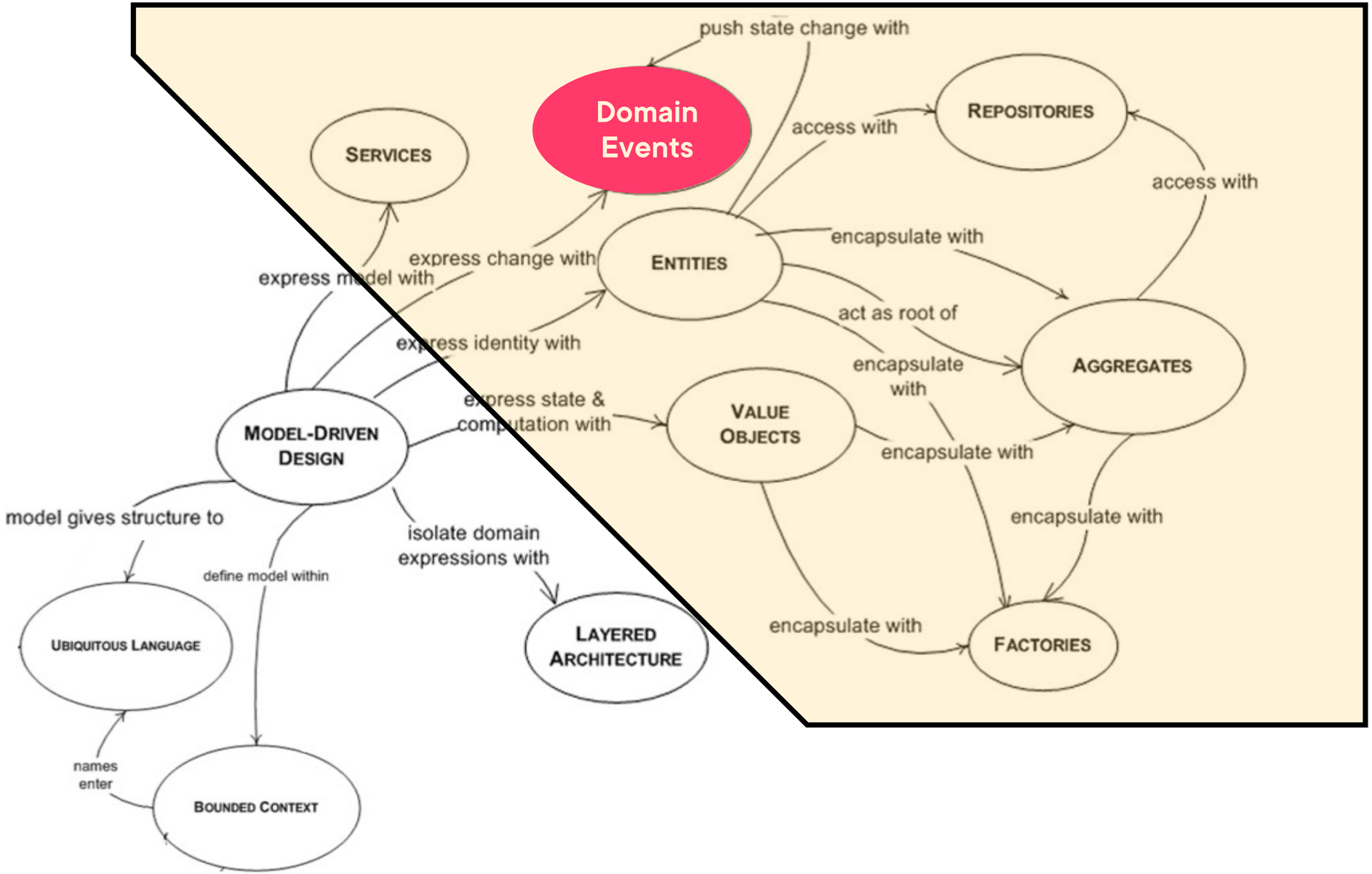
More Domain Event Features

Can communicate outside of the domain

Encapsulated as objects

Each event is a full-fledged class

Domain Events within the DDD Mind Map

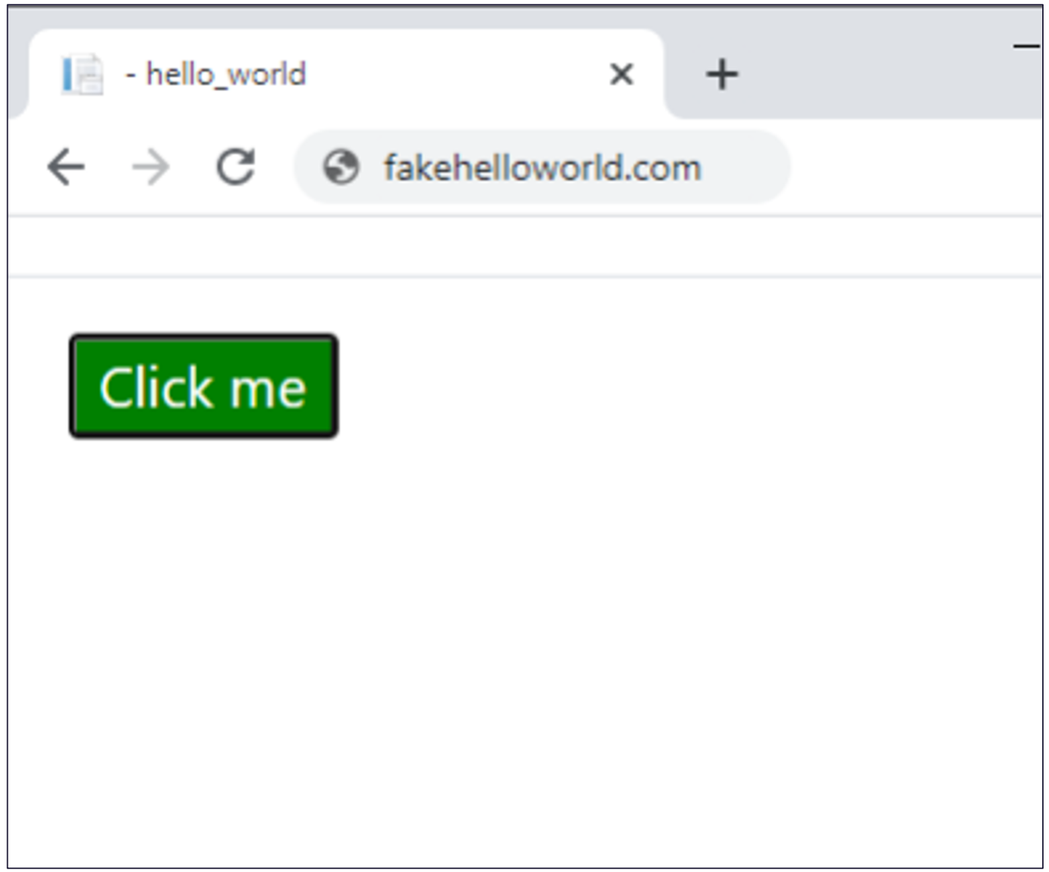


Vaughn Vernon

“Use a domain event to capture an occurrence of something that happened in the domain.”

Citation: Implementing Domain-Driven Design

```
<button id=“b"

  style="background-color: blue;

         color:white"

  onclick="changeColor()"> Click me
</button>

<script>

  function changeColor() {

    document

     .getElementById(“b");

     .style.backgroundColor = "green";

    }

</script>
```



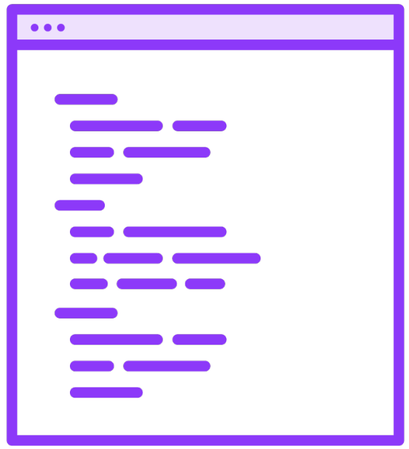
User Interface Events vs. Domain Events

User Interface Events

onclick

onkeypress

onsubmit



Domain Events

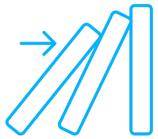
AppointmentScheduled

AppointmentConfirmed

ClientRegistered

Identifying Domain Events in Our System

Domain Events Pointers



When this happens, then something else should happen.
“If that happens…”, “Notify the user when…” , “Inform the user if…”

Domain events represent the past

Typically, they are immutable

Name the event using the bounded context’s ubiquitous language

Use the command name causing the event to be fired

Domain Event Examples

User Authenticated

Appointment Confirmed

Payment Received

Create events as needed, not just in case

YAGNI.

You Ain’t Gonna Need It.

Designing Domain Events

Each Event Is Its Own Class

```
public class AppointmentScheduled
{
  . . .
}

public class AppointmentConfirmed
{
  . . .
}
```

Include When the Event Took Place

A base class can help!

```
namespace PluralsightDdd.SharedKernel
{public abstract class BaseDomainEvent
 :INotification
 {
  public DateTimeOffset DateOccurred
  { get; protected set; } = DateTimeOffset.UtcNow;
 }
}
```

Capture Event-specific Details

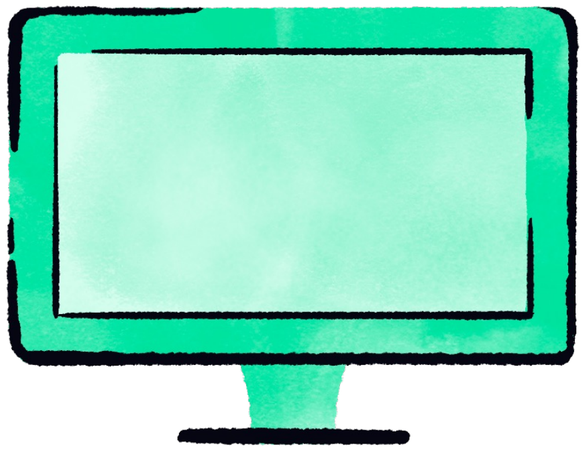
```
public class AppointmentScheduledEvent
    : BaseDomainEvent
{
  public AppointmentScheduledEvent(
           Appointment appointment) : this()
  {
    AppointmentScheduled = appointment;
  }
}
```

Event Fields Are Initialized in Constructor

```
public AppointmentScheduledEvent()

{
    this.Id = Guid.NewGuid();

}
```

No behavior or side effects

Applying Domain Events to a Simple App



This demo: Add domain events to a minimal app for easier focus

Next clip: See domain events in the Front Desk app

Notifications and emails are sent before the data is saved.

Services and Repositories for the Same Tasks?

The domain model should work with either workflow

Putting all logic into services leads to anemic domain models

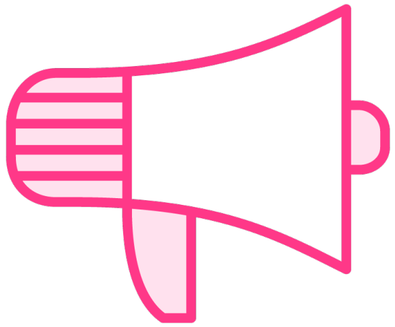
Aggregates should work whether accessed directly or through services

Hollywood Principle

Hollywood Principle in Software

Similar to dependency inversion

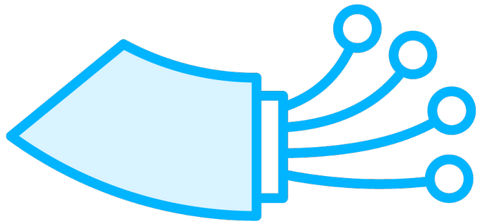
Aggregate doesn't need to know what actions must be performed



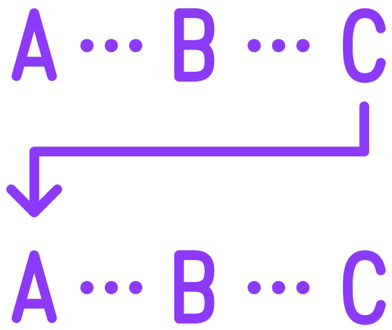
Inform the app about an event

App triggers the needed actions

Prepare for Domain Events



“Plumbing” needs to be in place



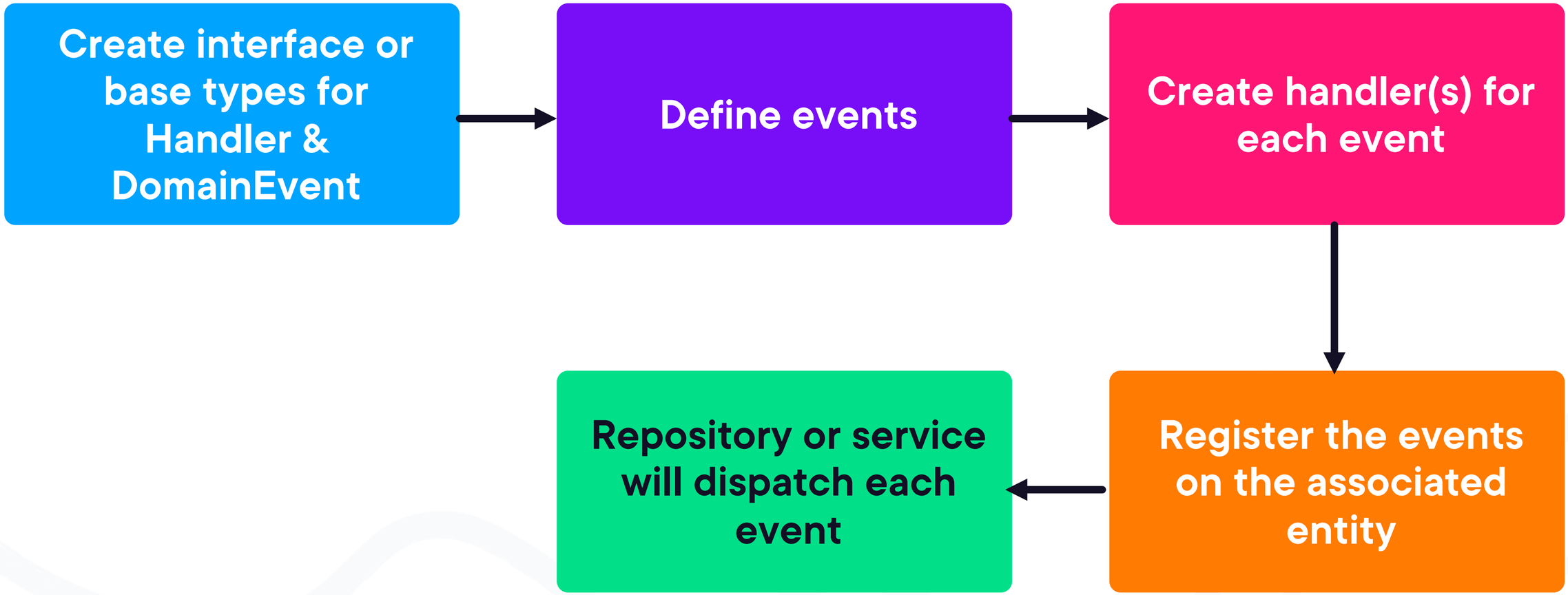
Consider the order of operations

E.g., persistence should succeed before notifications are sent

Inform the app about an event

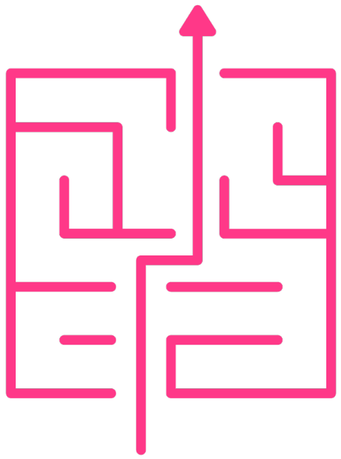
App triggers the needed actions

Domain Events Workflow

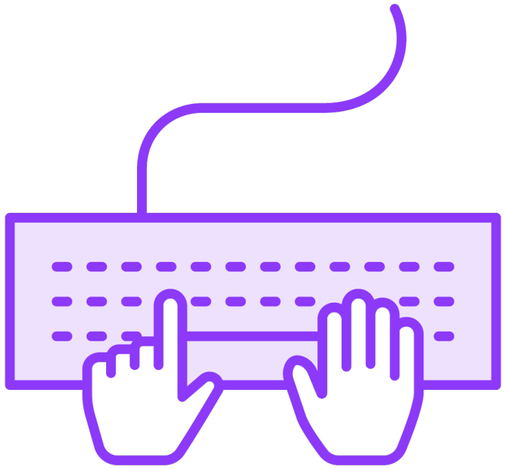


Exploring Domain Events in Our Application

Following the Flow of Events



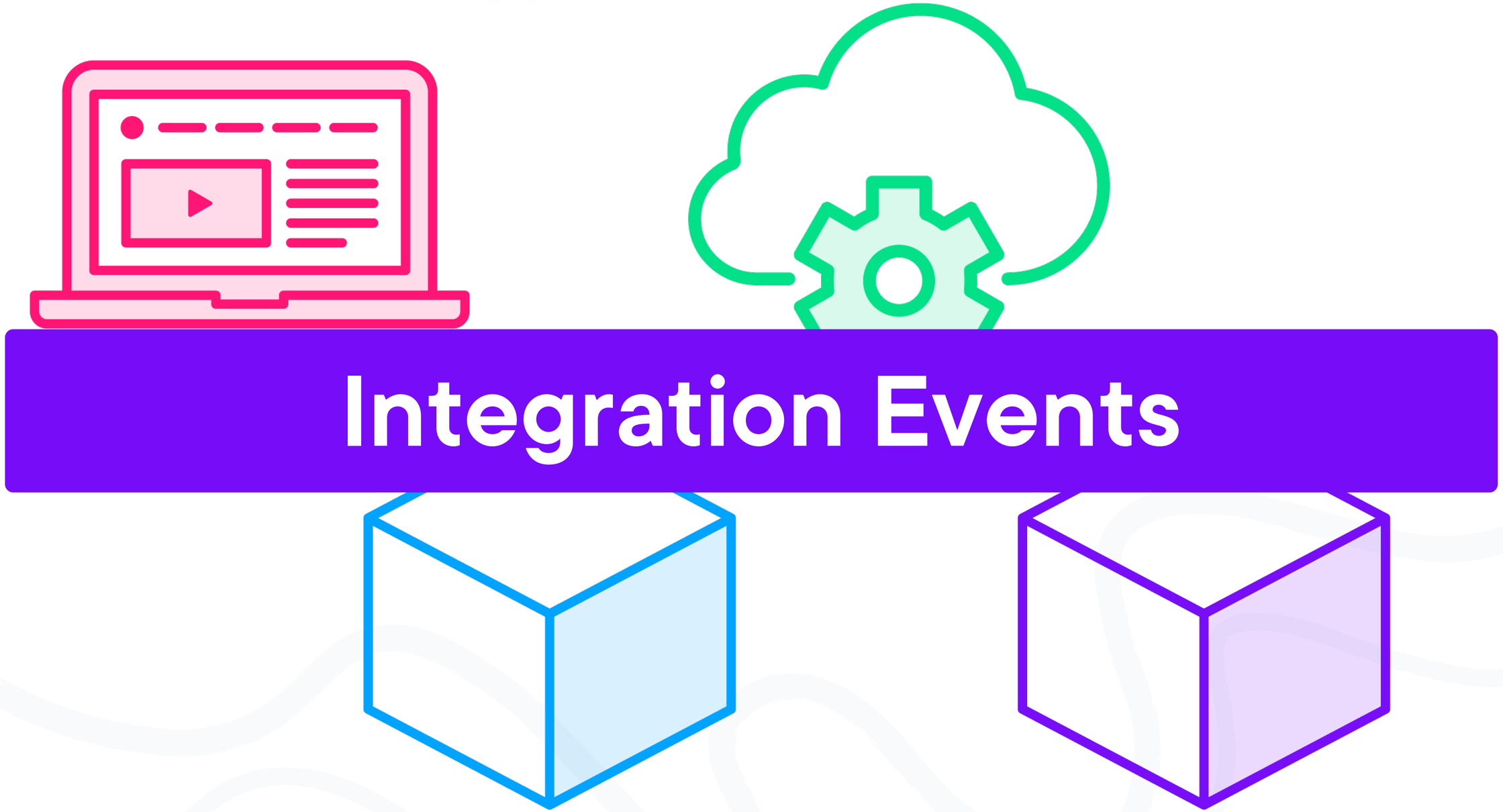
It may seem daunting at first



With practice it will become easier

Focus on an event object's references

Events Between Apps, Services, or BCs



Integration Event Message Types Must Match

Class names can differ; property names and types must match

AppA.SomeEvent.cs

```
// Publishing app
// Changes to this type must also be
// made in all consuming apps.
public class SomeEvent
{
  public Guid CustomerId {get; set;}
  public string Fullname {get; set;}
  public string Email {get; set;}
}
```

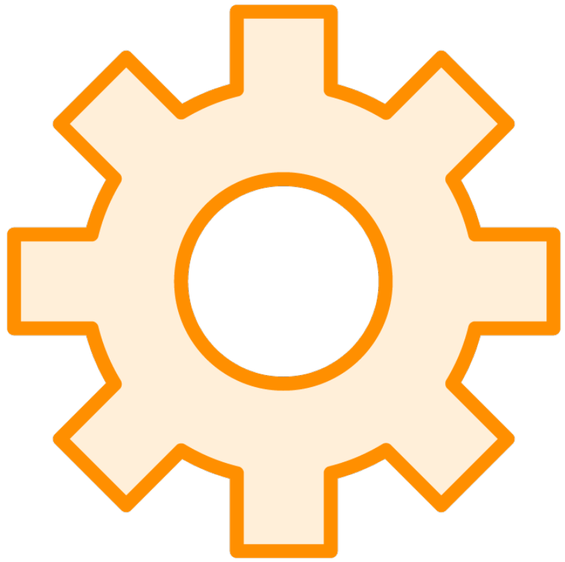
AppB.SomethingEvent.cs

```
// Consuming app
// Class name can differ; props match
// (a shared class would sync easily)
public class SomethingEvent
{
  public Guid CustomerId {get; set;}
  public string Fullname {get; set;}
  public string Email {get; set;}
}
```

Don’t expect integration events to match your domain events.

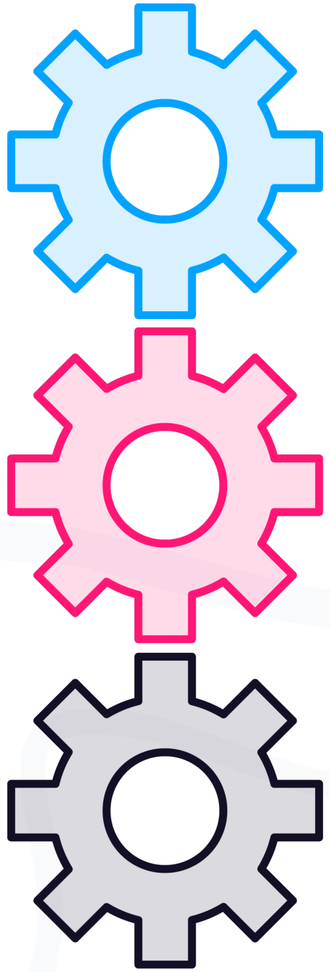
Events with Insufficient Data . . .

Source App



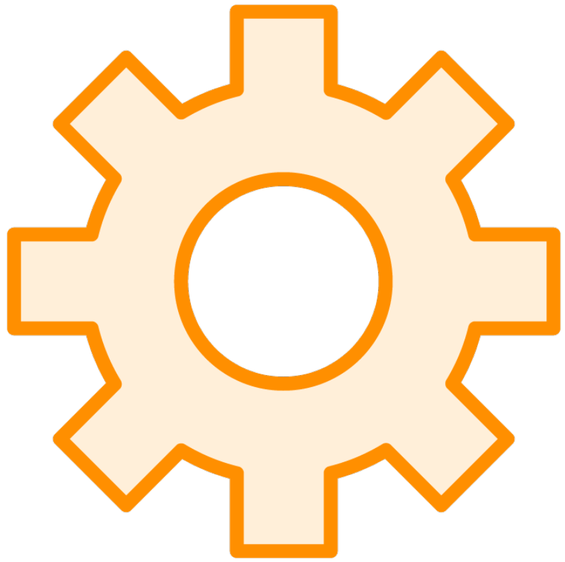
```
{
   “event-type” : “OrderCreated”,
   “customerId” : “123”,
   “orderId” : “456”
}
```

Consuming Apps

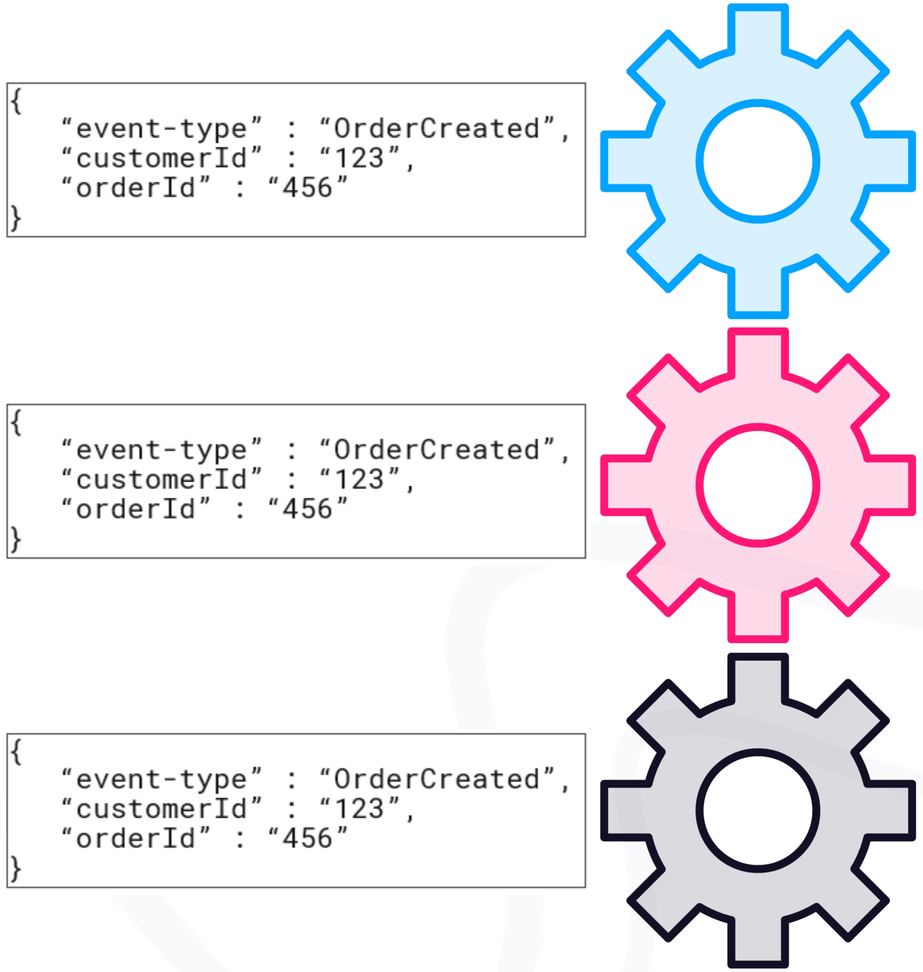


Events with Insufficient Data . . .

Source App

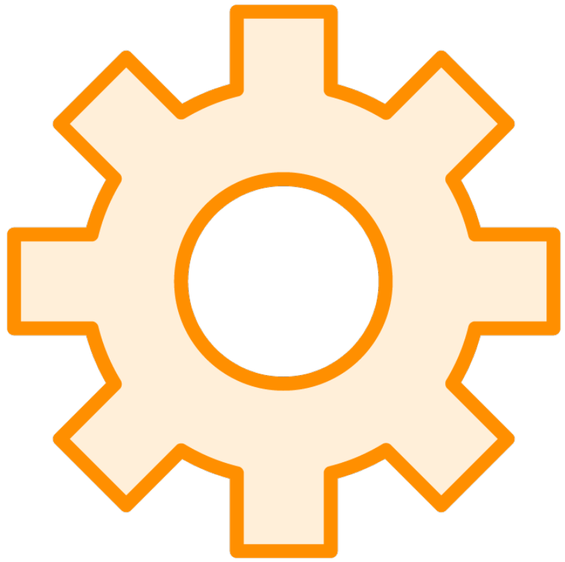


Consuming Apps

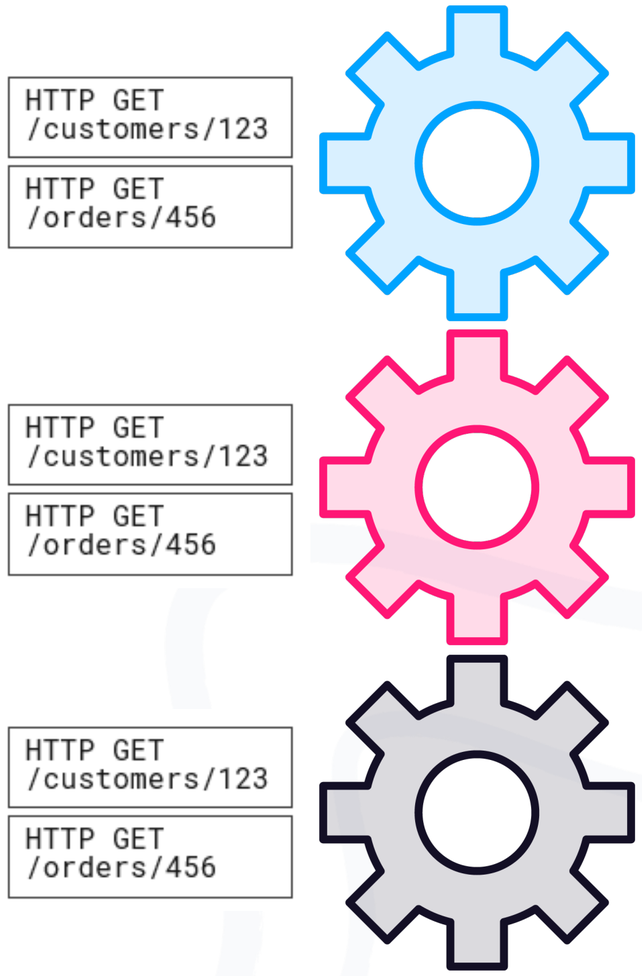


. . . Can Cause a Lot of Unneeded Chattiness

Source App

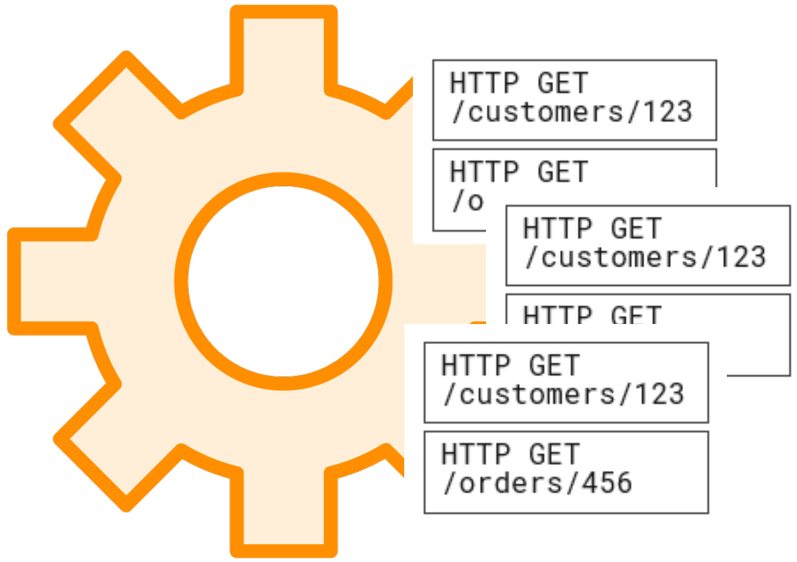


Consuming Apps

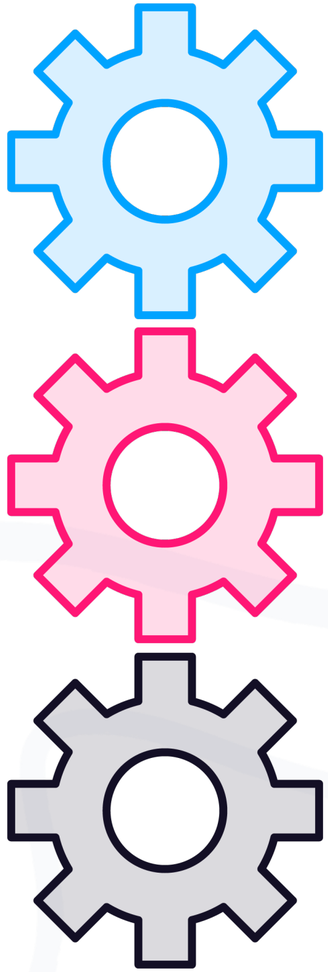


. . . Can Cause a Lot of Unneeded Chattiness

Source App



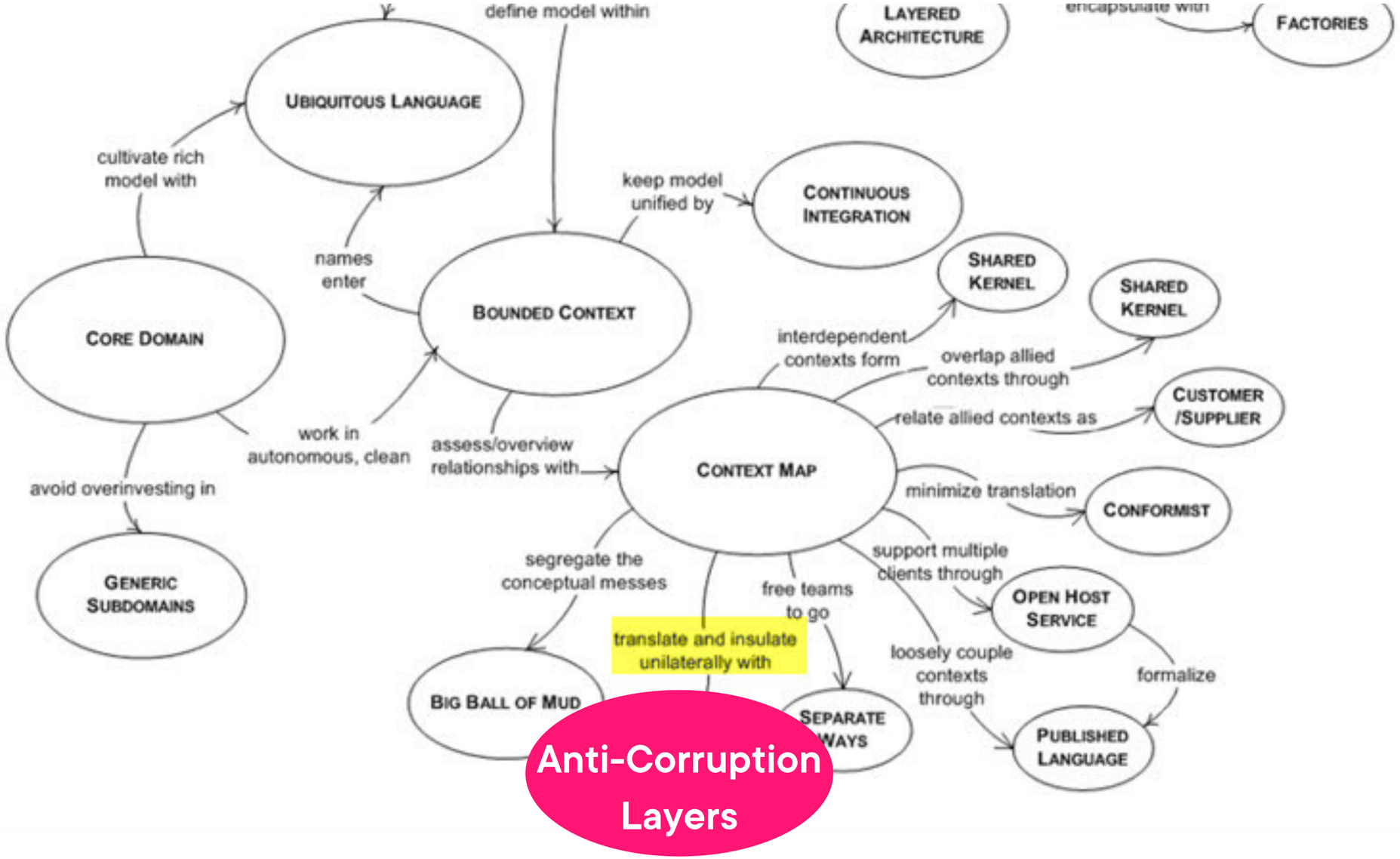
Consuming Apps



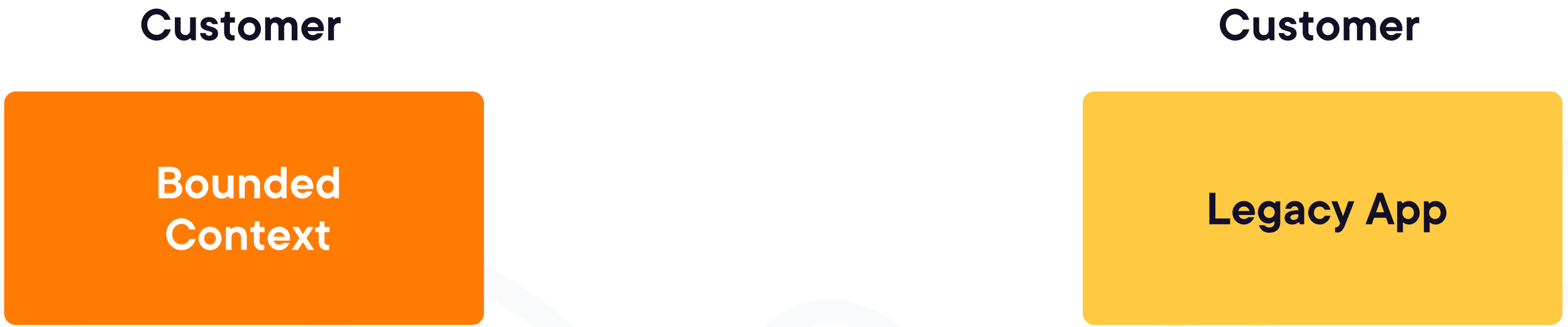
Introducing Anti-corruption Layers

Anti-corruption Layers

Anti-corruption Layer in the Mind Map



Anti-corruption Layers Insulate Bounded Contexts

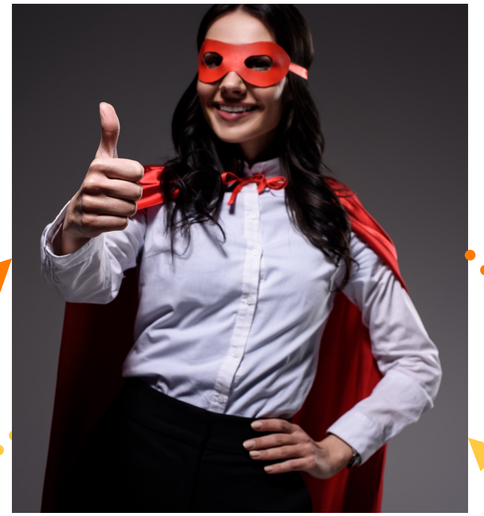


Anti-corruption Layers Insulate Bounded Contexts

Bounded Context

Legacy App

Anti-corruption Layers Insulate Bounded Contexts



Bounded Context

Legacy App

Anti-corruption Layer Is Not a Design Pattern

Its job is to translate between foreign systems' models and your own

Simplifies communication between systems

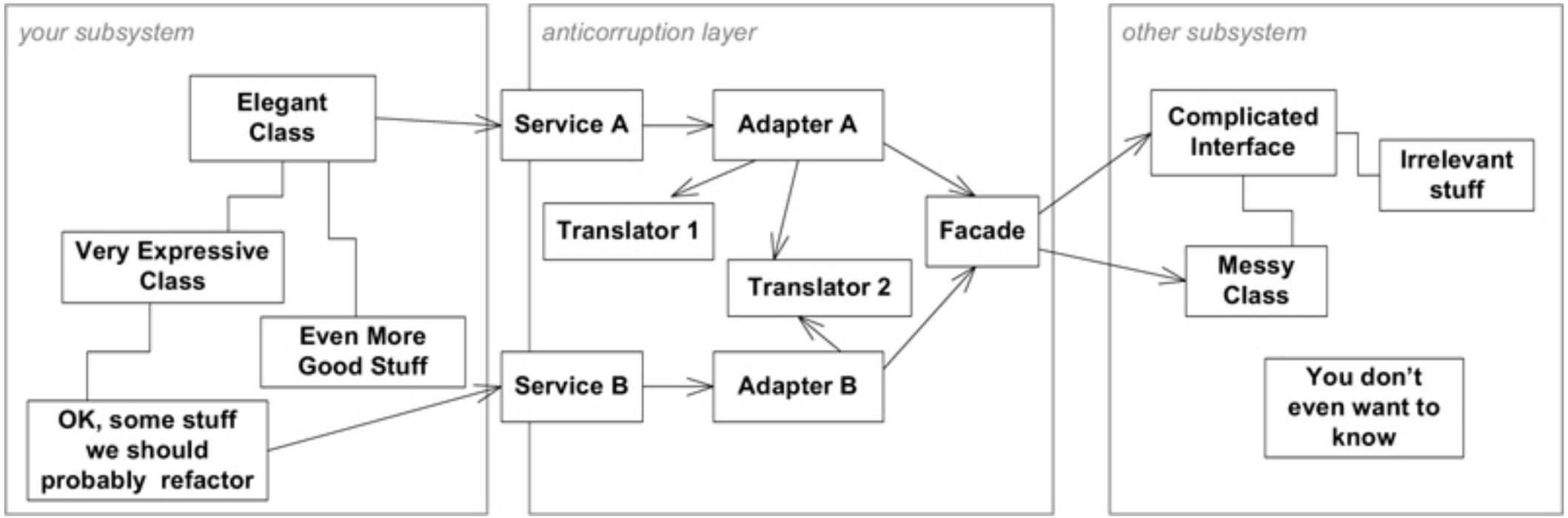
May employ design patterns such as façade or adapter

Eric Evans

“Even when the other system is well designed, it is not based on the same model as the client.

And often the other system is not well designed.”

The Structure of an Anti-corruption Layer



Citation: Evans, Domain-Driven Design, p. 367

Module Review and Resources

Domain Event

A class that captures the occurrence of an event in a domain object

Hollywood Principle

“Don’t call us, we’ll call you”

Anti-Corruption Layer

Functionality that insulates a bounded context and handles interaction with foreign systems or contexts

Key Takeaways

What are domain events?

When would we use this pattern in our model?

How are they designed and used?

Domain events in action

Anti-corruption layers protect your models while interacting with other systems

Resources Referenced in this Course

Getting Started with DDD when Surrounded by Legacy Systems II, Eric Evans - https://www.domainlanguage.com/ddd/surrounded-by-legacy-software/

Lost in bounded context translations with Julie, Indu, Michael and Nick VirtualDDD meetup youtu.be/u-5sKvh48-g

On Pluralsight: C# Design Patterns: Adapter by Steve Smith bit.ly/PS-Adapter

On Pluralsight: C# Design Patterns: Façade by David Starr bit.ly/PS-Facade

Let’s add a new feature to our application