



All events are testable!

Alberto Acerbis - Alessandro Colla

A dark, atmospheric photograph of a room. The ceiling is made of dark wooden beams. A window with a diamond-patterned glass is visible on the right wall. The walls are a mottled grey-blue color. The overall lighting is dim, creating a moody and mysterious atmosphere.

Repository

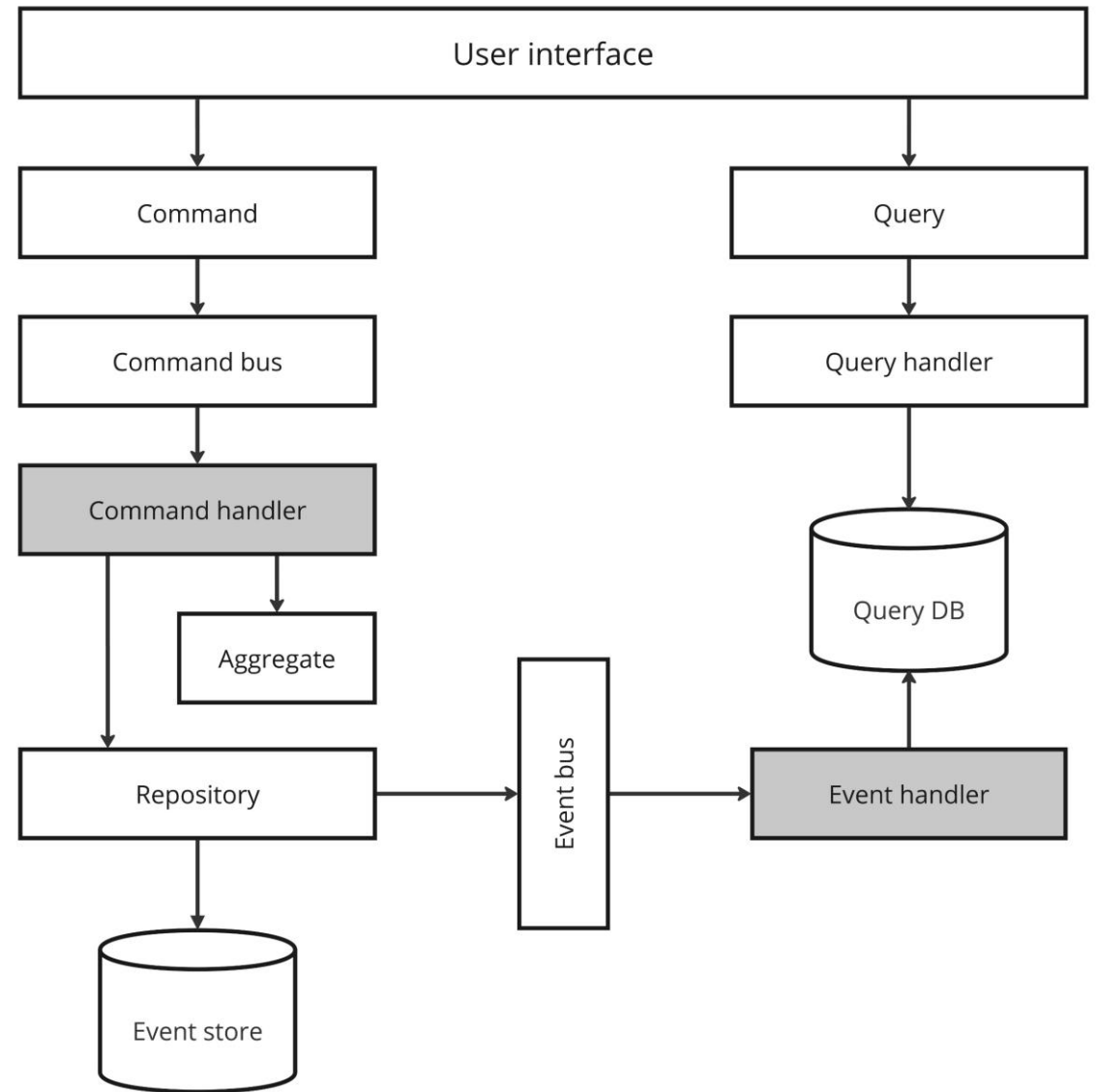
<https://github.com/BrewUp/KanD>  
DDinsky\_2025

# Schedule

- Introduction
- CQRS+ES fast recap
- A look at «BrewUp» and Solution structure
- A way to test domain events
- Exercises
- Q&A



# CQRS + ES fast recap



# What is BrewUp

- A demo ERP to handle a brewery's processes
- We purchase beers from neighbor breweries
- We make our own line of beers
- As you can imagine there is a lot to do (and drink!) in this domain

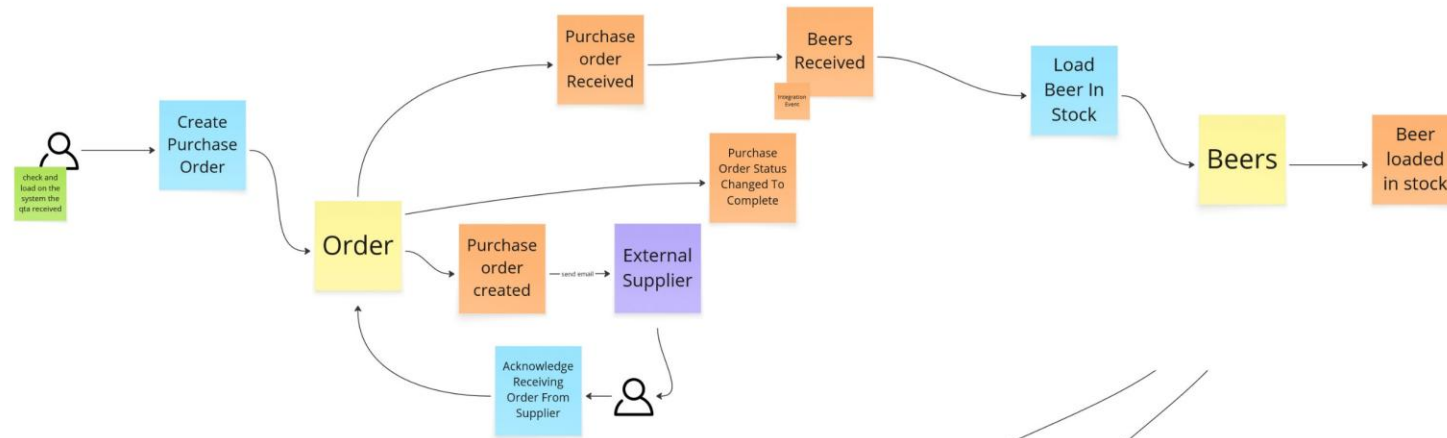


# Scenario for today

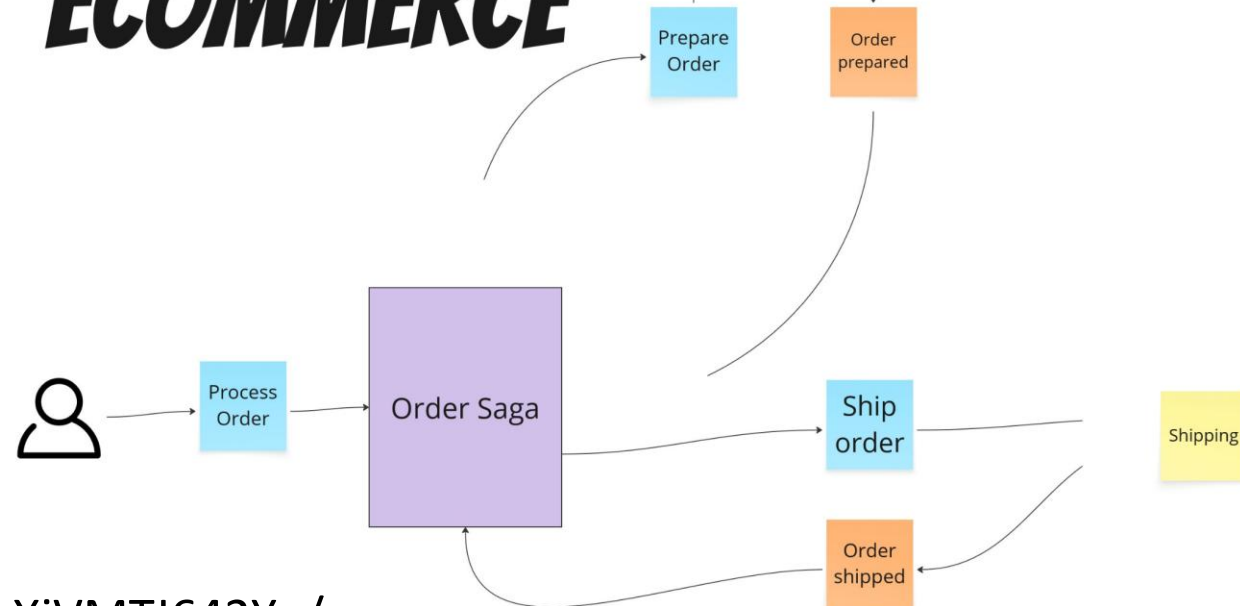
- We must handle a customer order made of beers that we do not make
- We must check availabilities in our warehouse
- In case, create an order to the external supplier
- Handle the receiving of the ordered beers
- Ship the order to the customer



# MANAGEMENT



# ECOMMERCE



Let's look at the code

# How we test it?

- Only with unit tests?
- Would we not limit ourselves to the single function, properties or simple state testing?
- Are these sufficient to represent the complexities of the domain?
- Isn't it better to test the behaviour?



# Once upon a time...there was an attempt...

```
[Test]
public void HandleChangeUserPassword()
{
    var handler = GetHandler();
    var dateRef = DateTime.Now;
    var command = new ChangeUserPassword(guid, "newpwd", dateRef);
    handler.Handle(command);
    Assert.AreEqual(guid, user.Id);
    Assert.AreEqual("newpwd", user.Password);
    Assert.AreEqual(dateRef, user.LastPasswordChangedDate);
    repository.Verify(x => x.Save(user, It.IsAny<Guid>(), It.IsAny<Action<IDictionary<string, object>>>()), Times.Once());
}
```

```
[Test]
public void HandleLockUser()
{
    var handler = GetHandler();
    var dateRef = DateTime.Now;
    var command = new LockUser(guid, dateRef);
    handler.Handle(command);
    Assert.AreEqual(guid, user.Id);
    Assert.AreEqual(dateRef, user.LastLockedOutDate);
    Assert.IsTrue(user.IsLockedOut);
    repository.Verify(x => x.Save(user, It.IsAny<Guid>(), It.IsAny<Action<IDictionary<string, object>>>()), Times.Once());
}
```

# Let's try with specification testing

- Tests should be able to represent any complex behaviour with simplicity
- They must be comprehensible to everyone
- They represent the thinking behind the design of the aggregate
- "Given, When, Then" with events, commands and command handlers



# Given, When, Then

- The idea is to separate a scenario into three sections:
- **Given**: describes the state of the aggregate before sending the command. A kind of prerequisite for the test.
- **When**: represents the command to be sent to the aggregate.
- **Then/Expect**: describes the state changes expected as a result of the command.



# Questions?

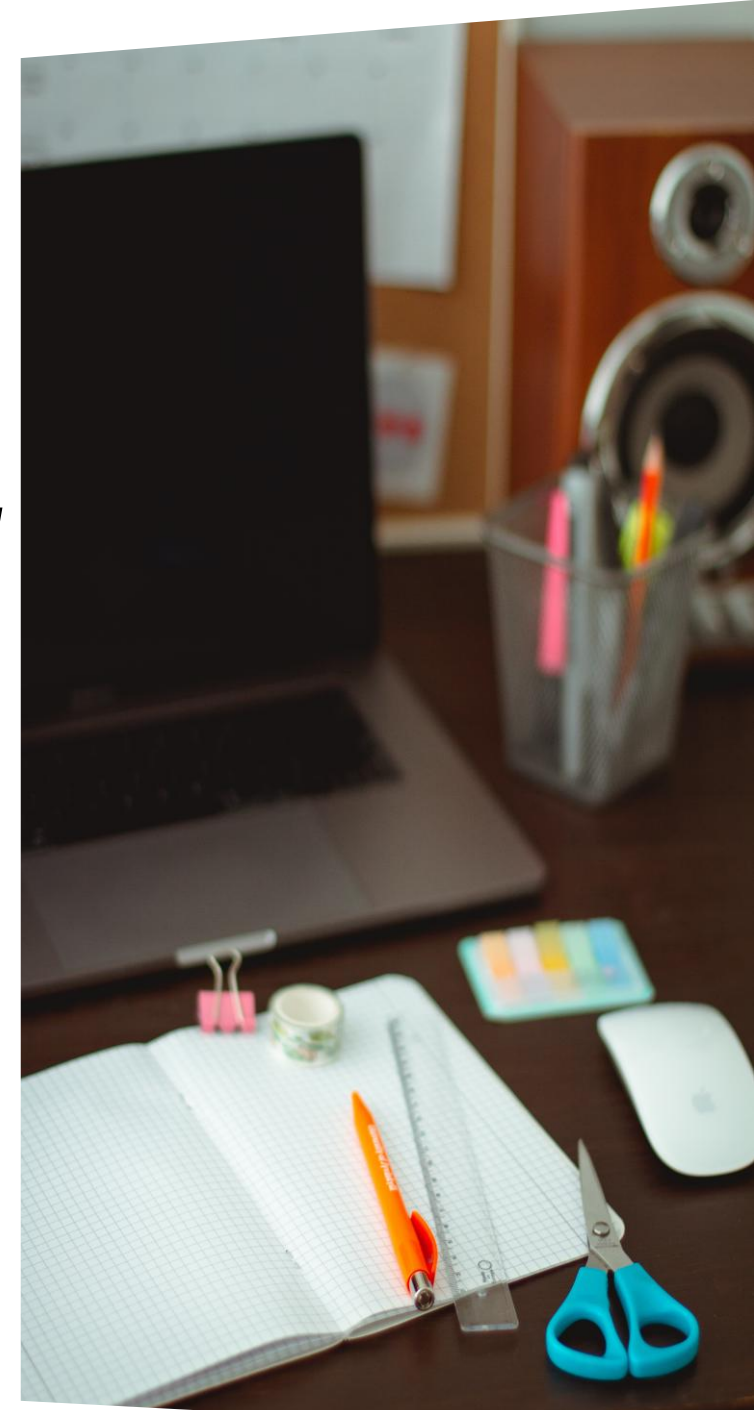


Let's do the first test together

# Exercises

Write tests and events to handle:

- ***CreatePurchaseOrder -> PurchaseOrderCreated***
- UpdatePurchaseOrderState -> PurchaseOrderReceived
- PurchaseOrderReceived -> BeersReceived
- LoadBeerInStock -> BeerLoadedInStock





10 minutes

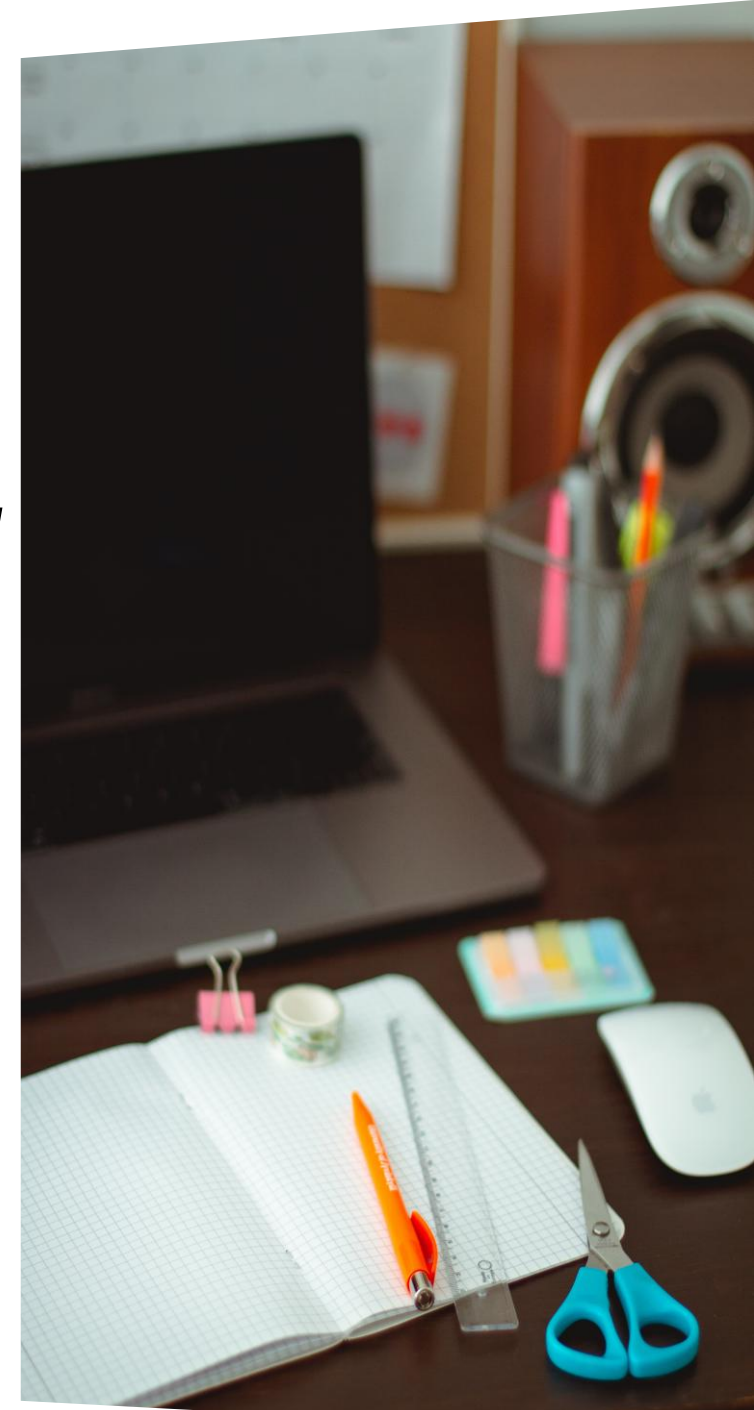
---



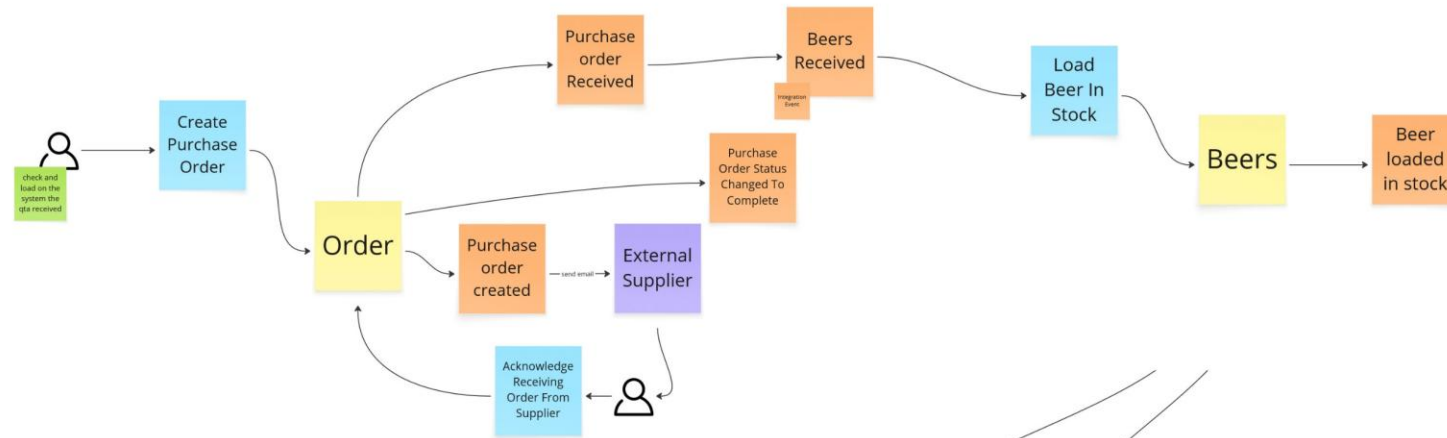
# Exercises

Write tests and events to handle:

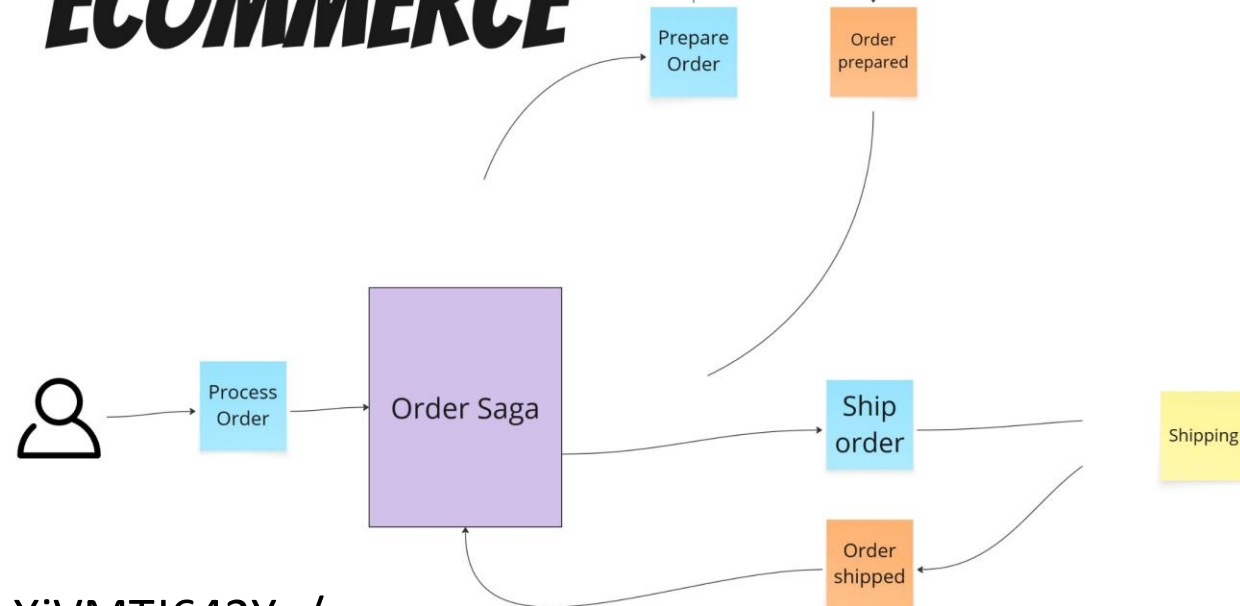
- ***CreatePurchaseOrder -> PurchaseOrderCreated***
- SendPurchaseOrderToSupplier-> PurchaseOrderSentToSupplier
- ReceivePurchaseOrderFromSupplier -> PurchaseOrderStatusChangedToComplete
- LoadBeerInStock -> BeerLoadedInStock



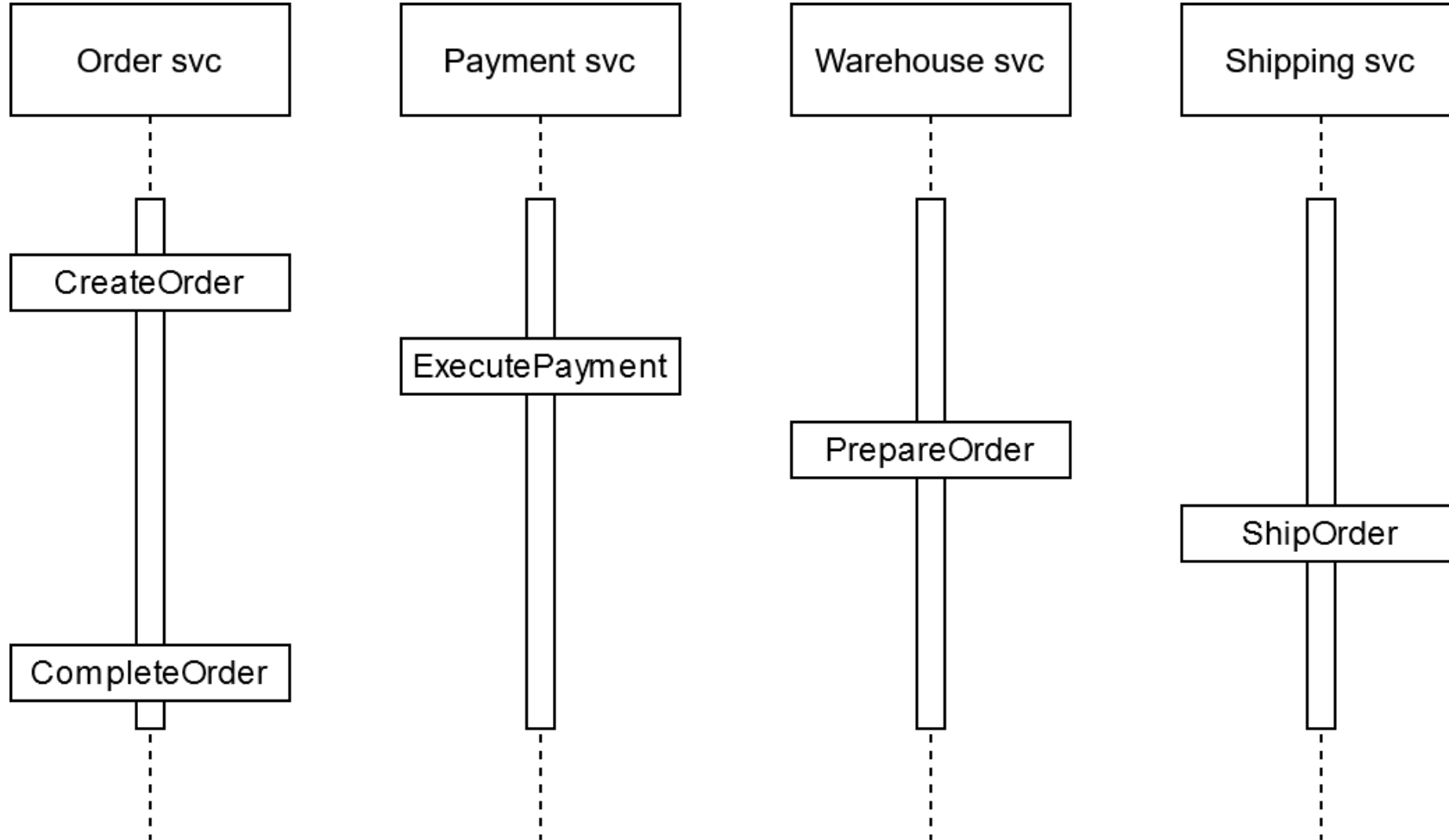
# MANAGEMENT



# ECOMMERCE

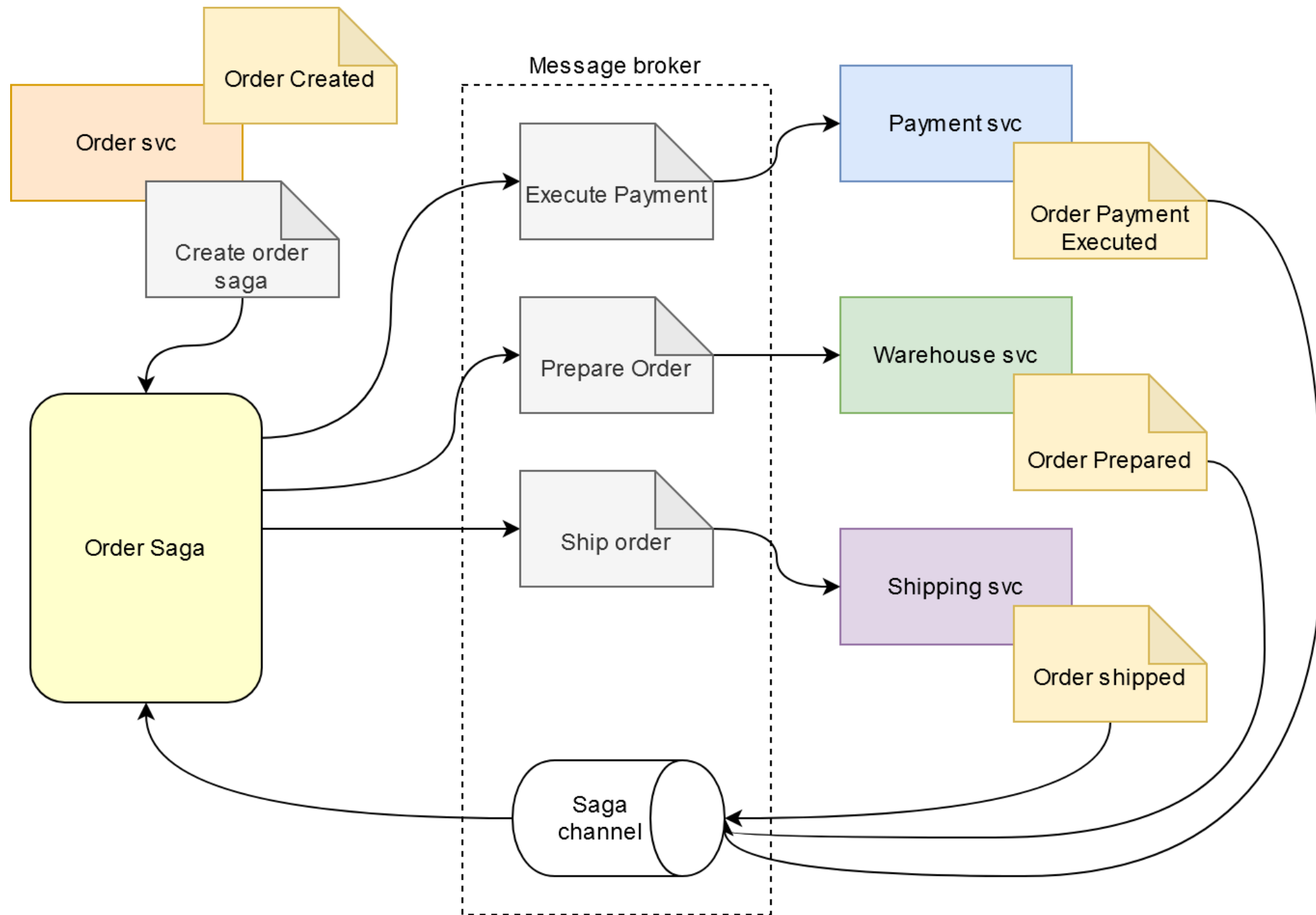


# Scenario

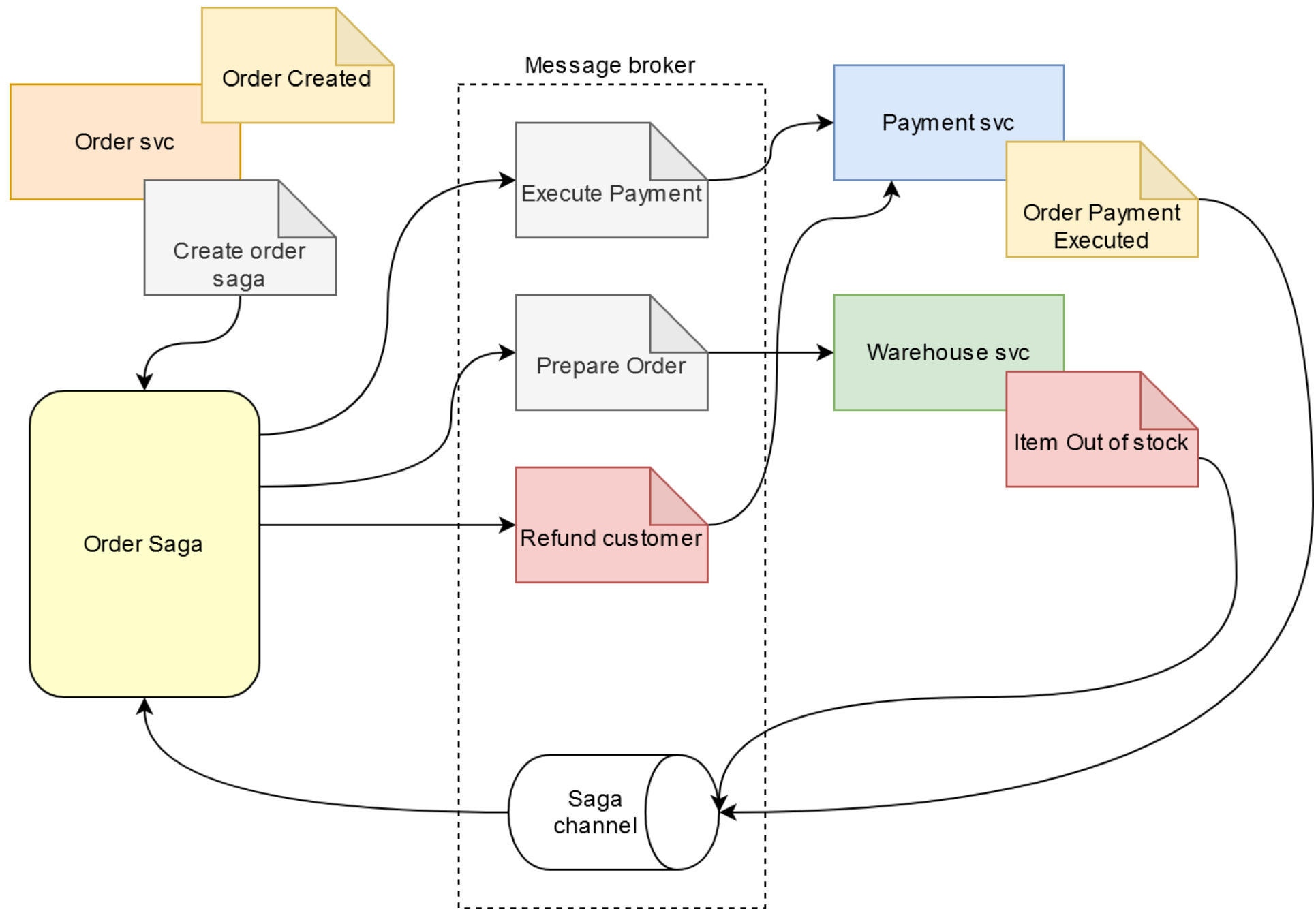


Choreography (Events)

Orchestration (Command)



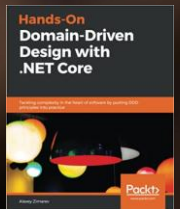
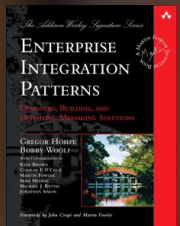
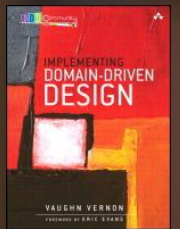
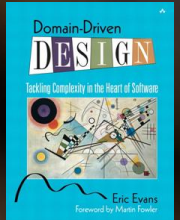
What if there is no product in stock?



Recap

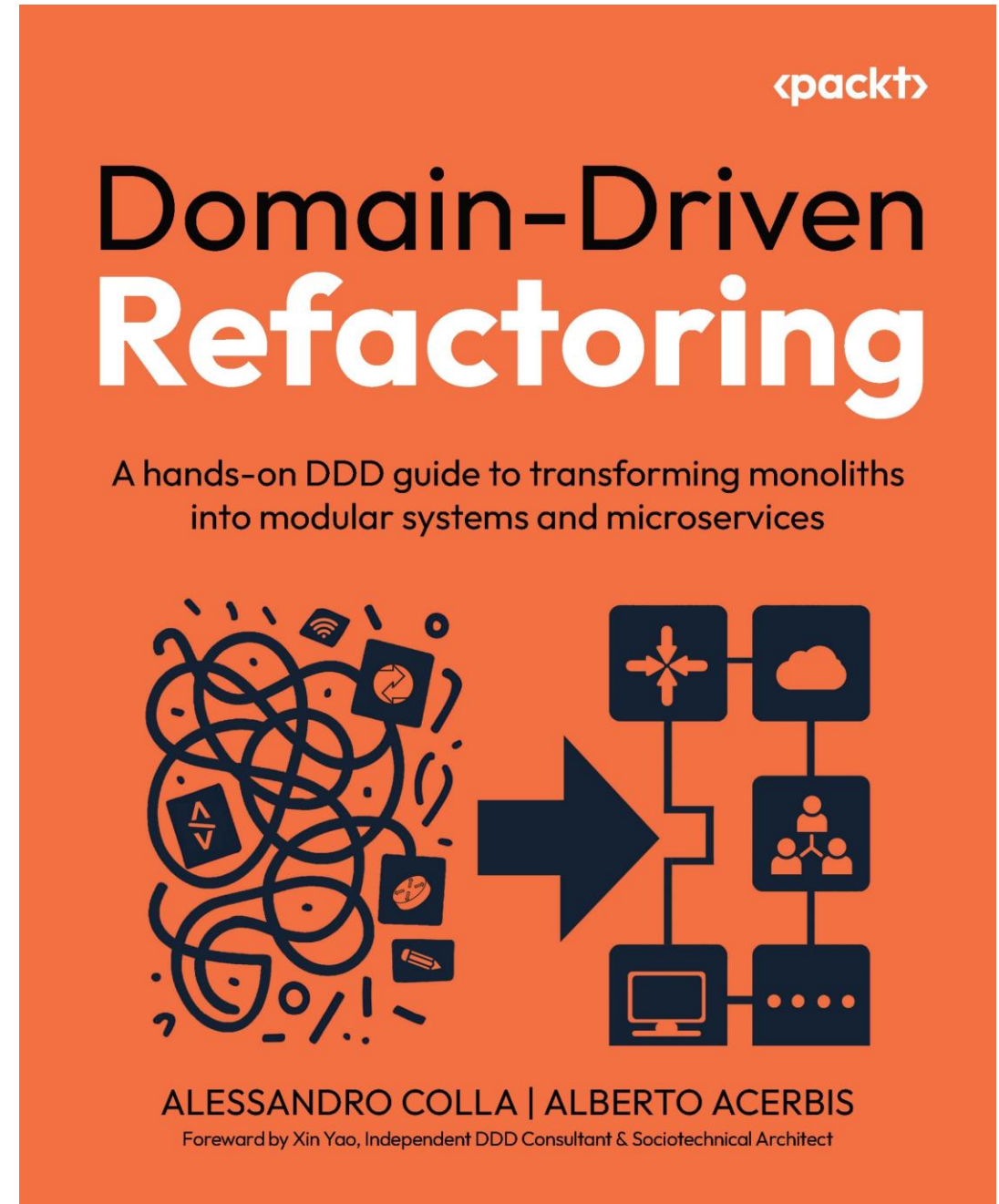
# Bibliografy

- Domain-Driven Design: Tackling Complexity in the Heart of Software (E. Evans)  
<https://www.amazon.it/Domain-Driven-Design-Tackling-Complexity-Software-ebook/dp/B00794TAUG>
- Implementing Domain-Driven Design (V. Vernon)  
<https://www.amazon.it/Implementing-Domain-Driven-Design-Vaughn-Vernon-ebook/dp/B00BCLEBN8/>
- REST in Practice: Hypermedia and Systems Architecture (J. Webber)  
<https://www.amazon.it/REST-Practice-Hypermedia-Systems-Architecture/dp/0596805829>
- Enterprise Integration Patterns (G. Hohpe)  
<https://www.amazon.it/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683/>
- Hands-On Domain-Driven Design with .NET Core (A. Zimarev)  
<https://www.amazon.it/Hands-Domain-Driven-Design-NET/dp/1788834097>



This workshop was based on some concepts explained in our recently released book

Use **DDD20** for a 20% discount during the conference days



# THANKS!

You can find us at:

 @aacerbis  
 <https://www.linkedin.com/in/aacerbis/>  
 alberto.acerbis@intre.it

 @collaalessandro  
 alessandrocolla  
 alessandro.colla@evoluzione.agency

