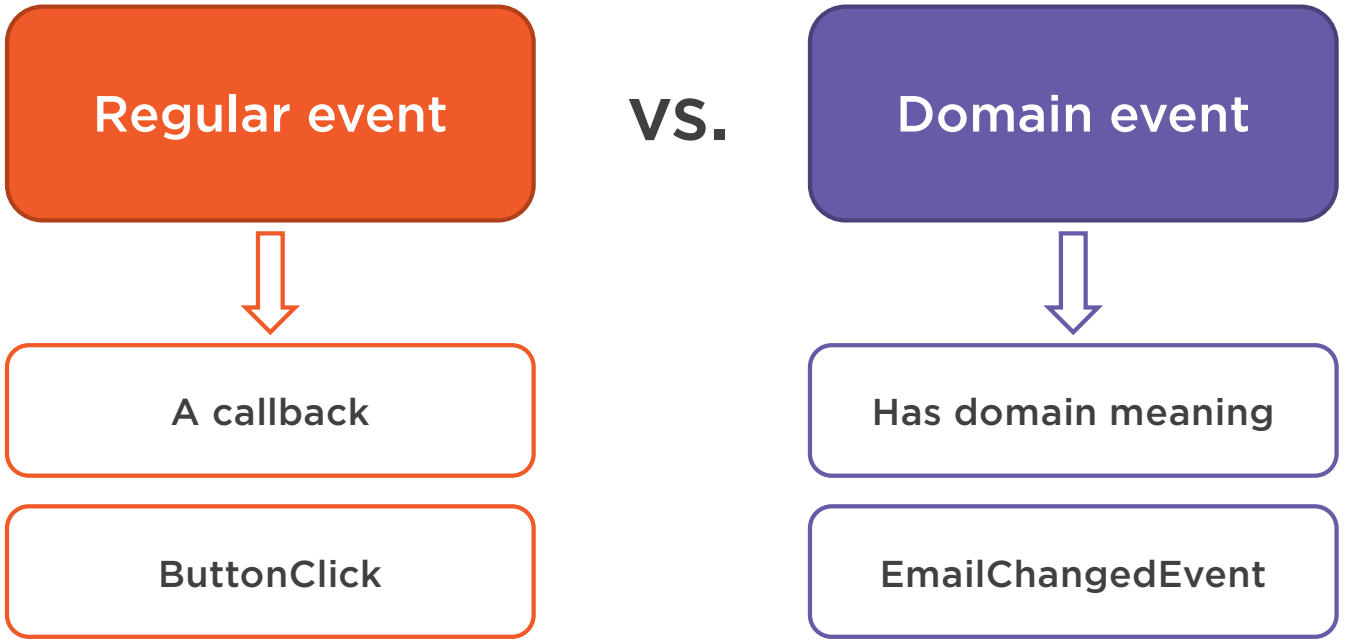
Implementing a Domain Event Dispatcher

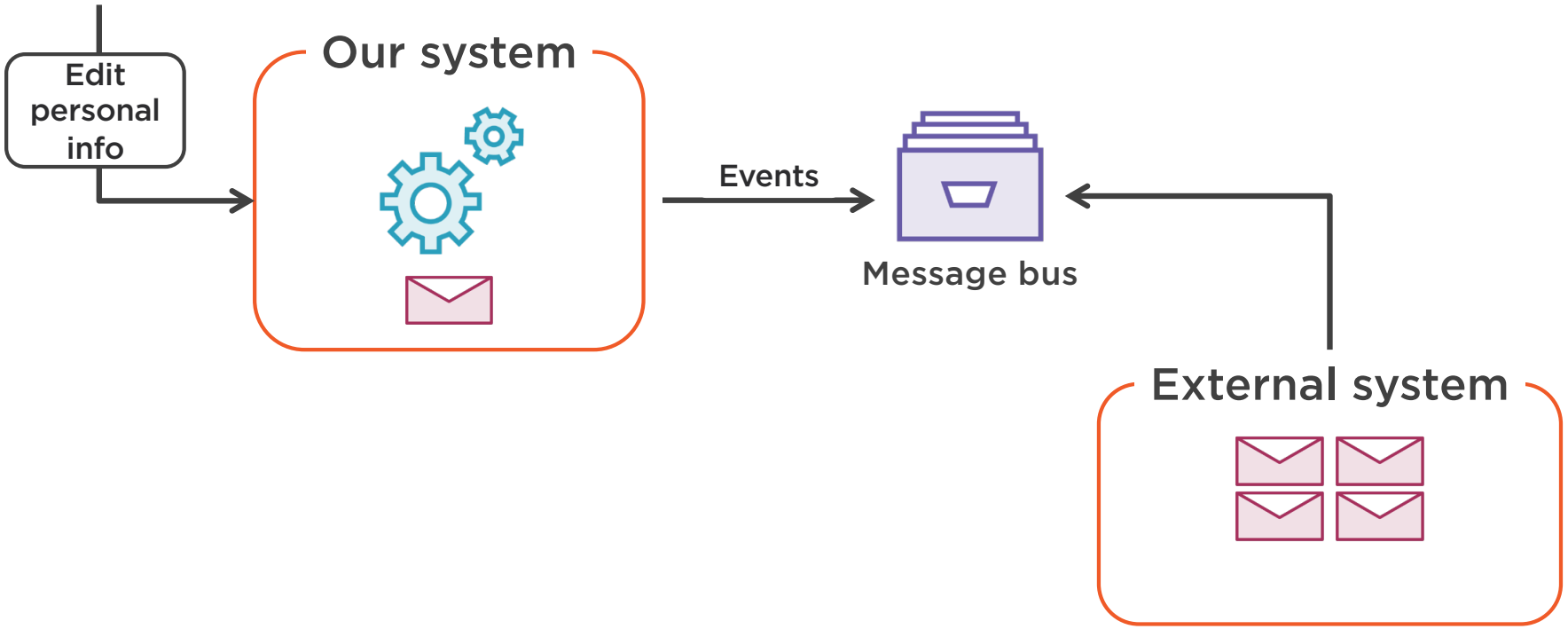
Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com

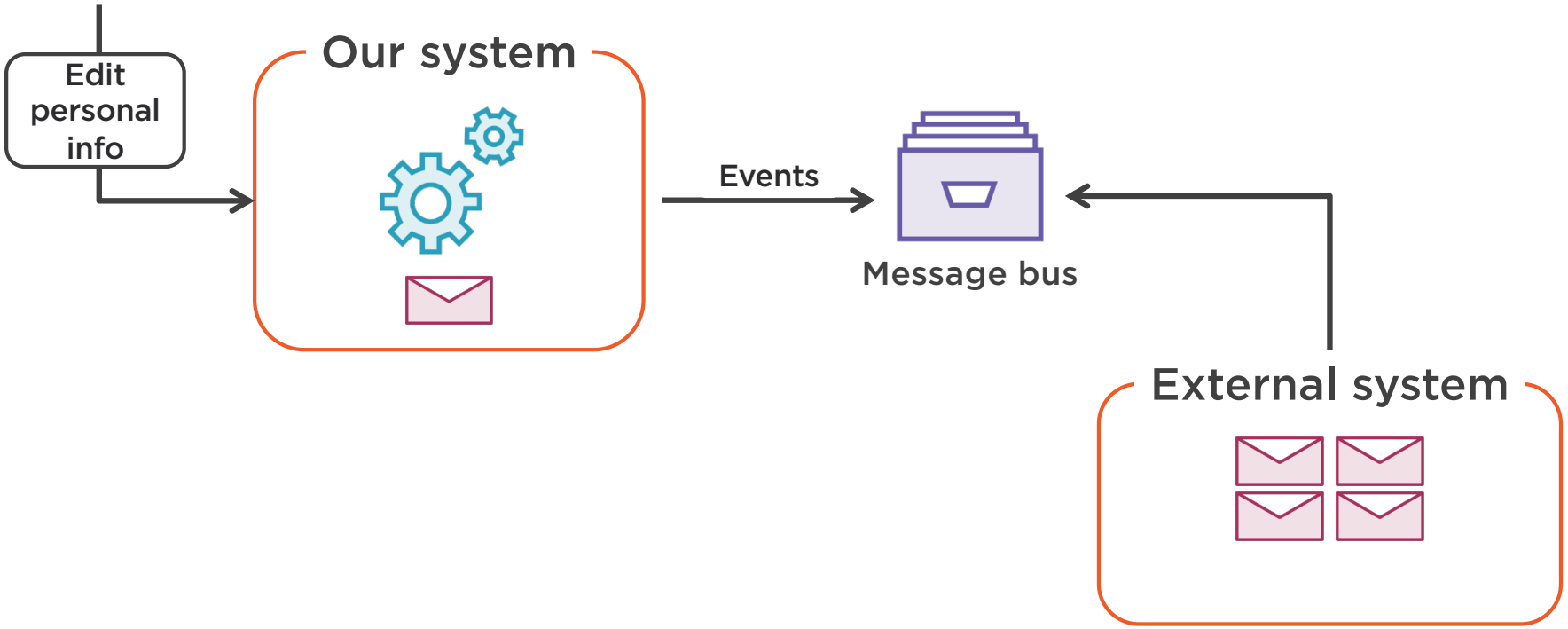
Domain Events



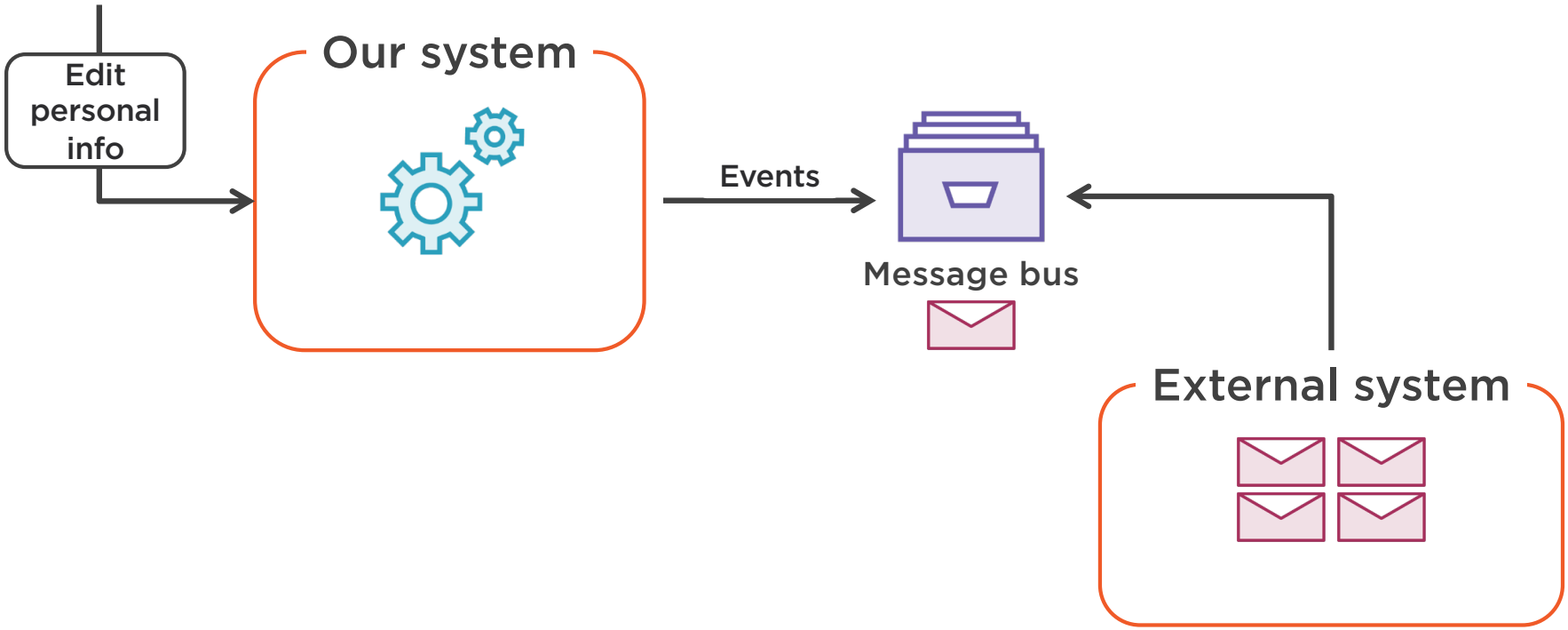
Domain Events



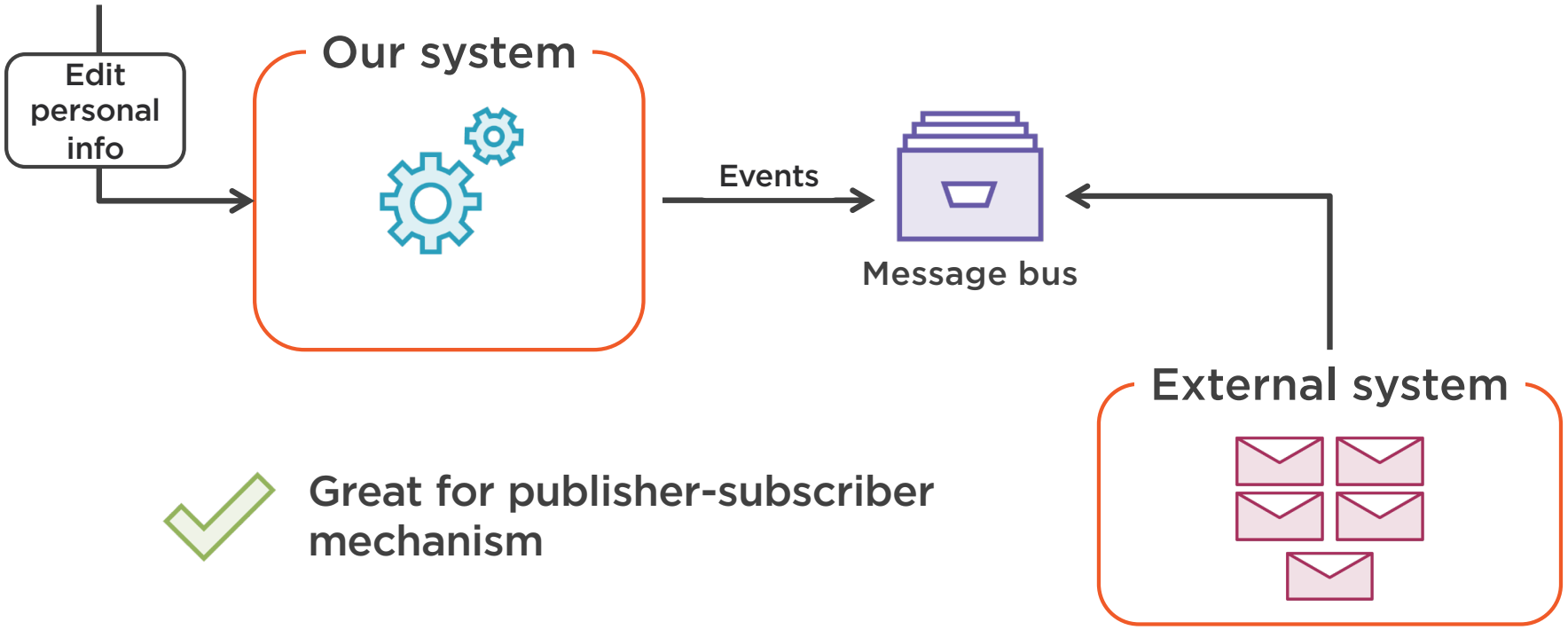
Domain Events



Domain Events

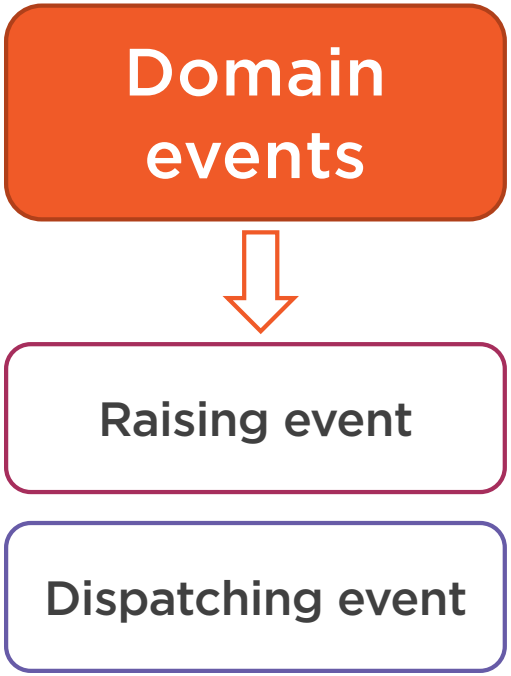


Domain Events



Great for publisher-subscriber mechanism

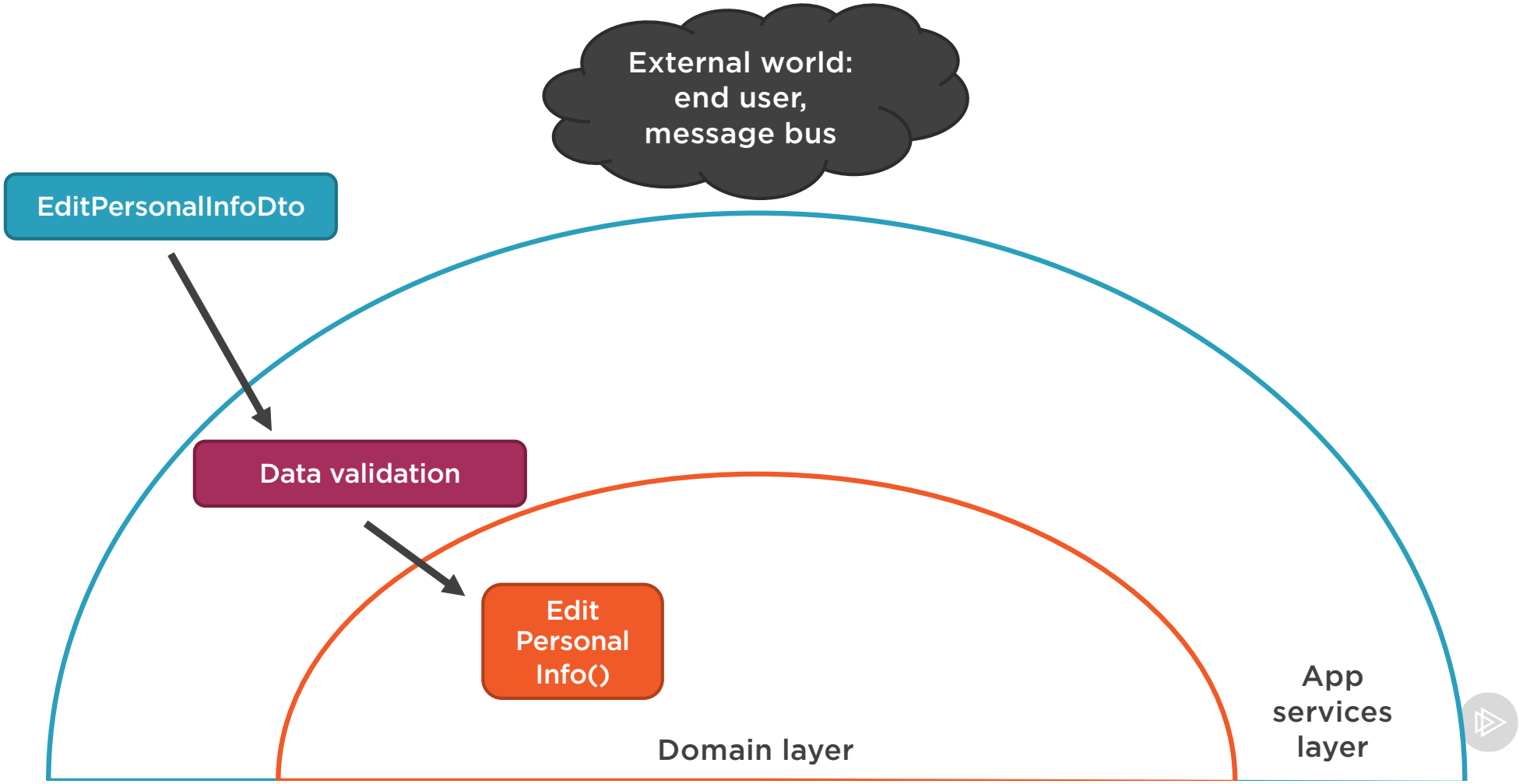
Domain Events



: Part of the domain layer's logic

: Converting the event into a side effect

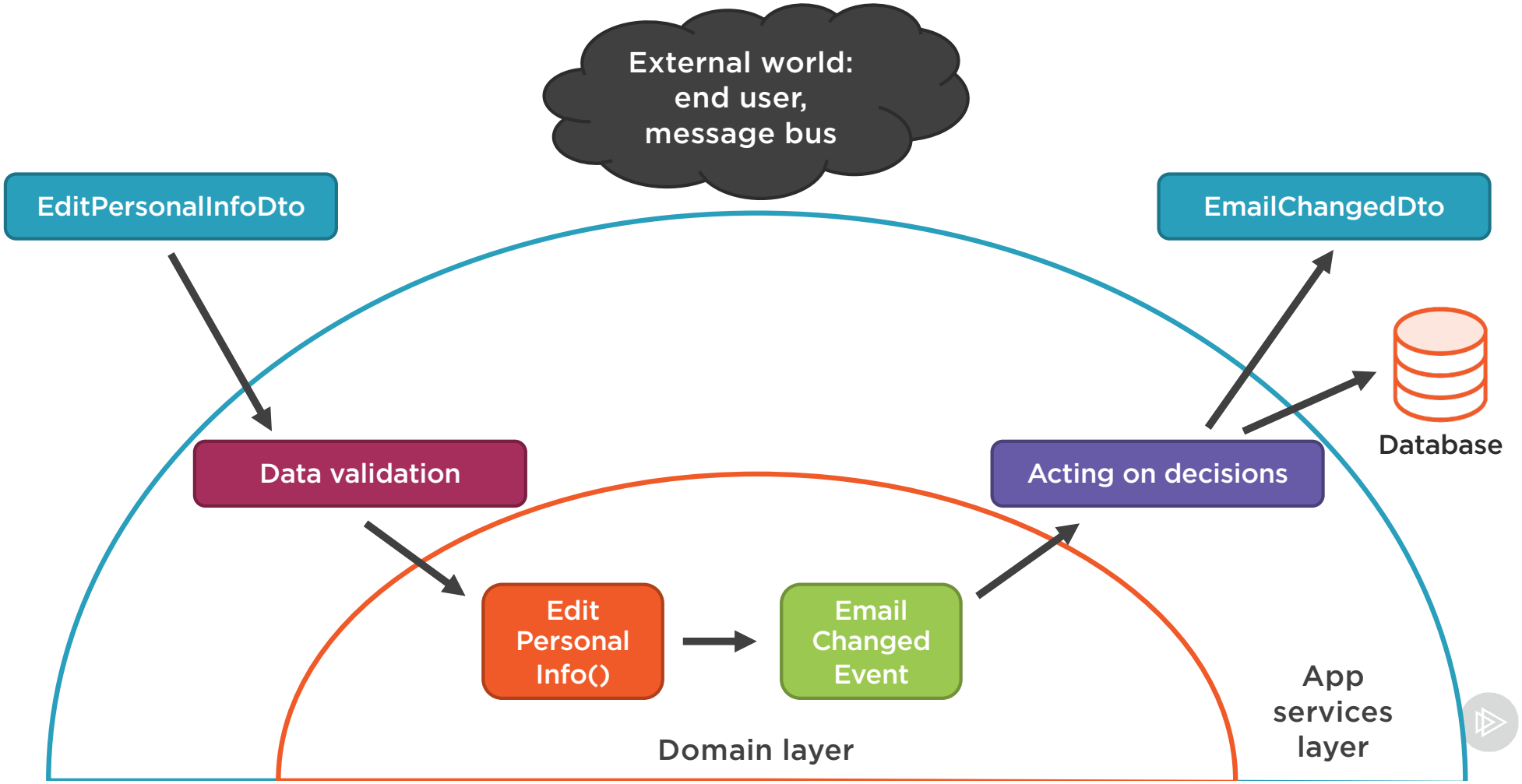
Domain Events



Domain Events

```
public string EditPersonalInfo(long studentId, string email)
{
    Student student = _repository.GetById(studentId);
    if (student == null)
        return "Student not found";

    Result<Email> emailResult = Email.Create(email);
    if (emailResult.IsFailure)
        return emailResult.Error;

    /* Other conversions */

    student.EditPersonalInfo(emailResult.Value);

    _context.SaveChanges();
}
```

Domain Events



Recap: Implementing Domain Events

Introduced StudentEmailChangedEvent

Raising event

Dispatching event

Recap: Implementing Domain Events

Raising a domain event

- Responsibility of the domain model
- Aggregate roots raise events

Dispatching a domain event

- Responsibility of the app services layer

Recap: Implementing Domain Events

```
public sealed class Bus : IBus {
    public void Send(string message)
}
```

Wrapper on top of SDK library

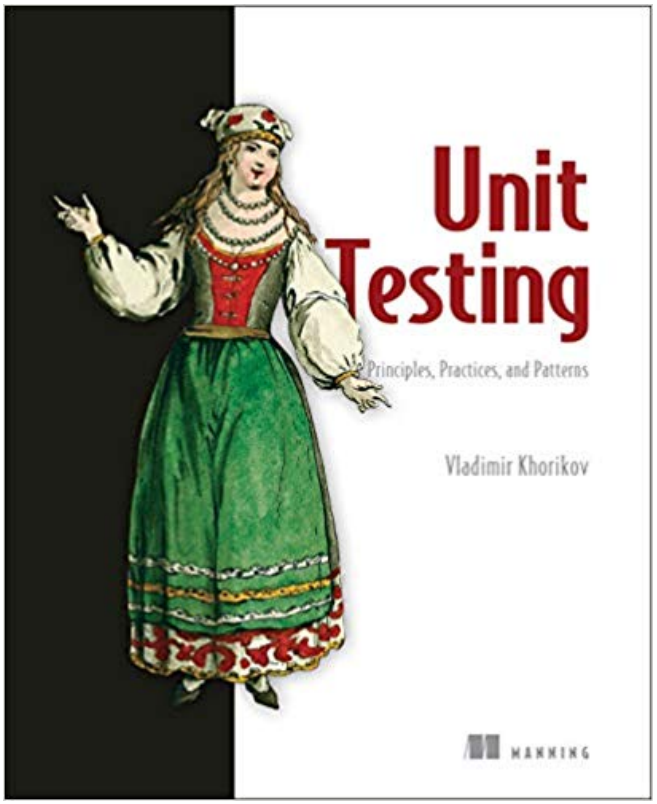
```
public sealed class MessageBus {
    public MessageBus(IBus bus)
    public void SendEmailChangedMessage(
        studentId, newEmail)
}
```

Contains all domain-specific messages

```
public sealed class EventDispatcher {
    public EventDispatcher(MessageBus messageBus)
    public void Dispatch(
        IEnumerable<IDomainEvent> events)
}
```

Converts events to side effects

Recap: Implementing Domain Events

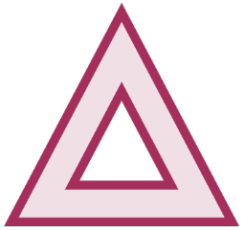


Unit Testing Principles, Practices, and Patterns

http://bit.ly/testing-book

Recap: Implementing Domain Events

Communication with external systems

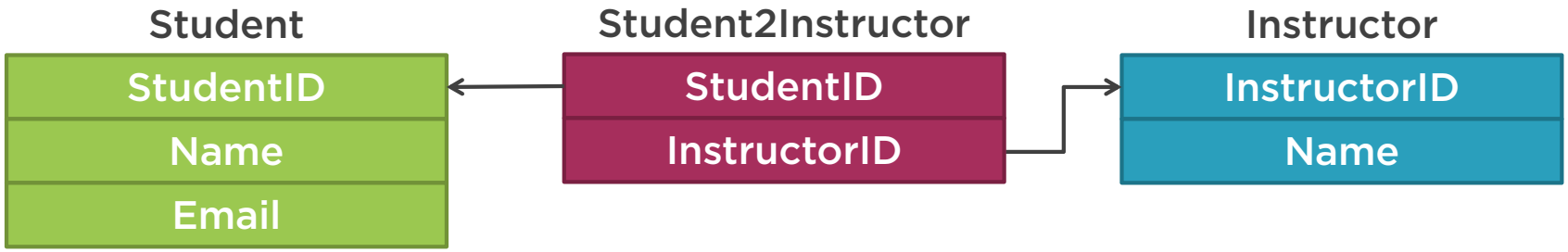


Communication inside the same application

Doesn’t add value

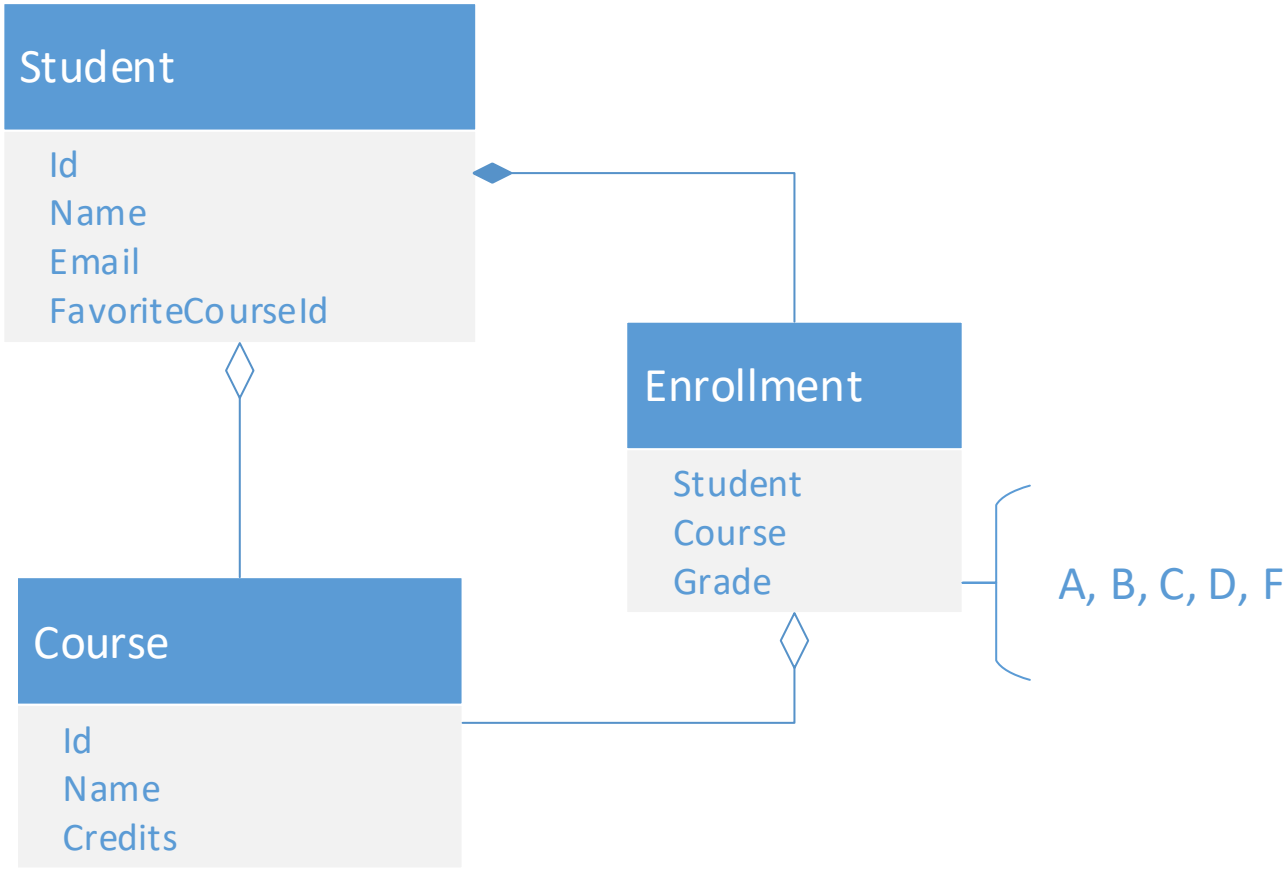
Best represented as explicit flow

Many-to-many Relationships



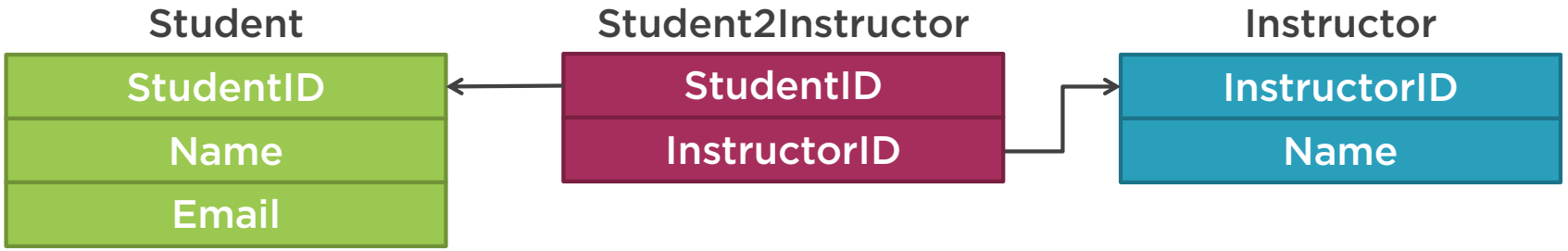
- Can be represented as two one-to-many relationships
- The intermediate table must be elevated into an entity

Many-to-many Relationships



Many-to-many Relationships

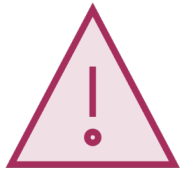
EF Core doesn't support
many-to-many relationships



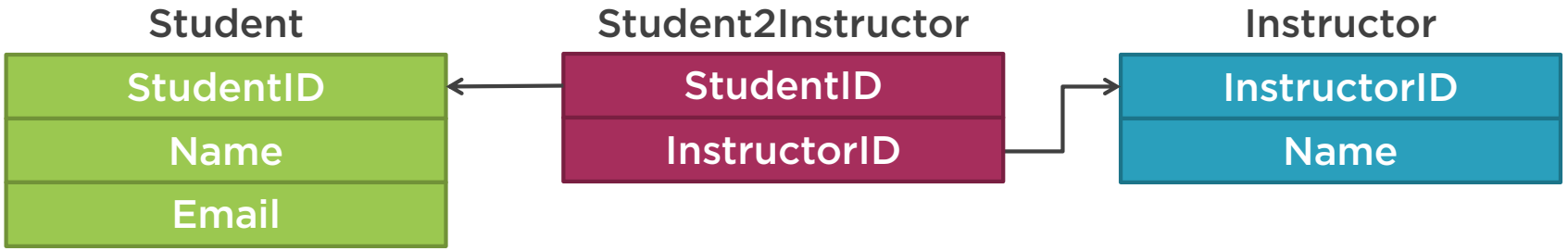
Have to introduce a Student2Instructor entity

Bad for separation of concerns

Many-to-many Relationships



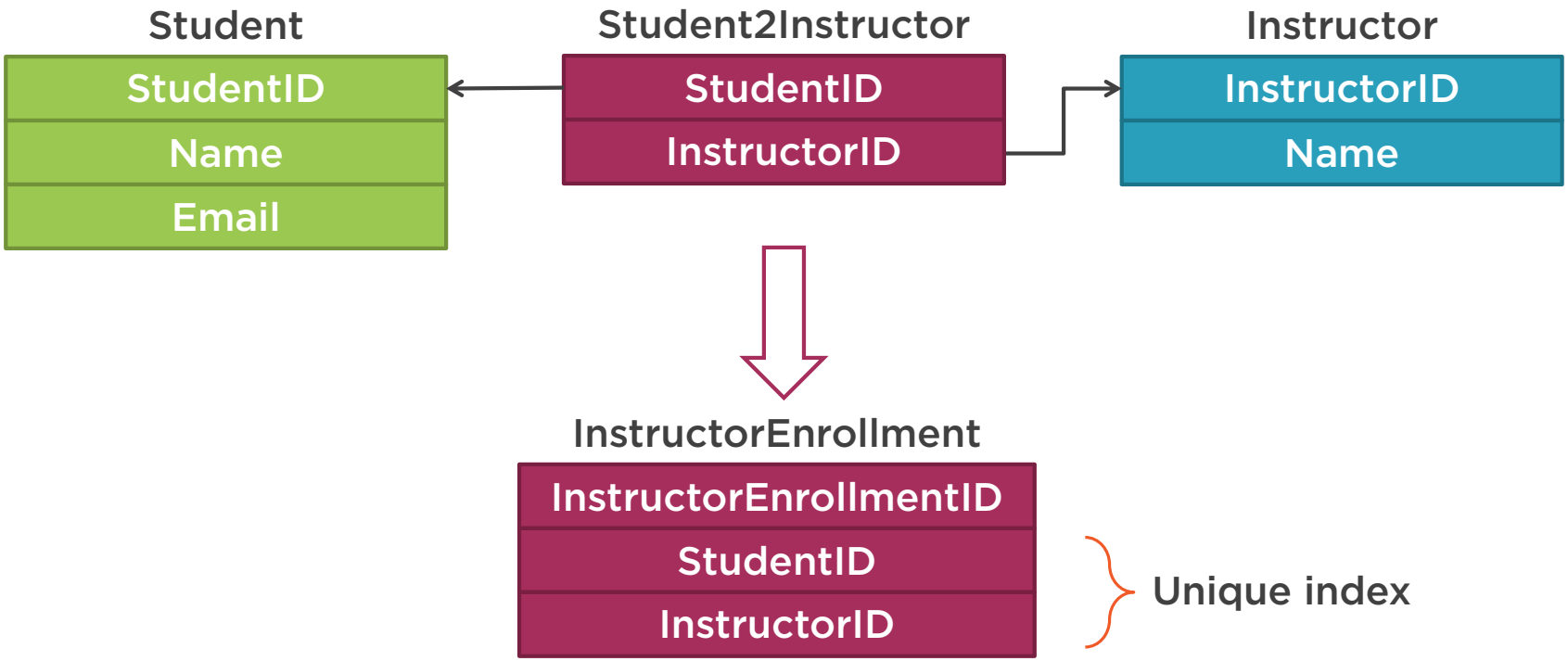
EF Core doesn't support
many-to-many relationships



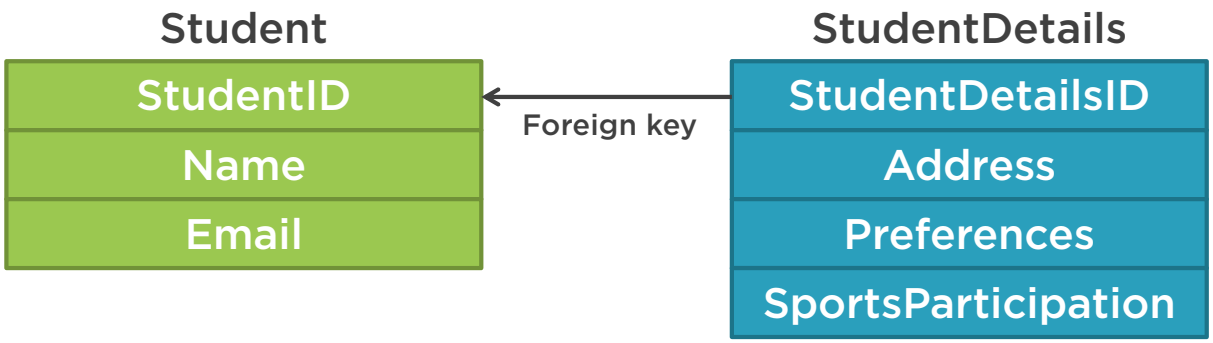
Student2Instructor -> InstructorEnrollment

Own dedicated primary key

Many-to-many Relationships



One-to-one Relationships




```
student.Details
```

```
studentDetails.Student
```

- Similar to Value Objects
- Always prefer Value Objects over one-to-one relationships

Resource List

Source code	https://github.com/vkhorikov/DddAndEFCore
	http://bit.ly/ddd-ef
Domain-Driven Design in Practice	https://www.pluralsight.com/courses/domain-driven-design-in-practice
	http://bit.ly/ddd-ps
Value Converters and nulls	https://github.com/dotnet/efcore/issues/13850
Value Converters and multiple columns	https://github.com/dotnet/efcore/issues/13947
Discussion about many-to-many relationships	https://github.com/dotnet/efcore/issues/1368
Implementing many-to-many relationships	https://github.com/dotnet/efcore/issues/10508
Unit Testing Principles, Practices, and Patterns	https://www.amazon.com/gp/product/1617296279
	http://bit.ly/testing-book

Course Summary

Encapsulation and separation of concerns

Many-to-one relationships

- Prefer navigation properties over ids

Prefer using lazy loading by default

One-to-many relationships

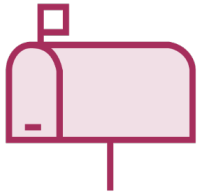
- Drawbacks in EF Core's lazy loading

Working with detached objects

Mapping value objects

Implementing domain events

Contacts



http://bit.ly/vlad-updates

@vkhorikov

https://enterprisecraftsmanship.com