Reasoning About Many-to-Many Variations

Julie Lerman

EF Core Expert and Serial DDD Advocate

@julielerman | thedatafarm.com

Many-to-Many's Many Options

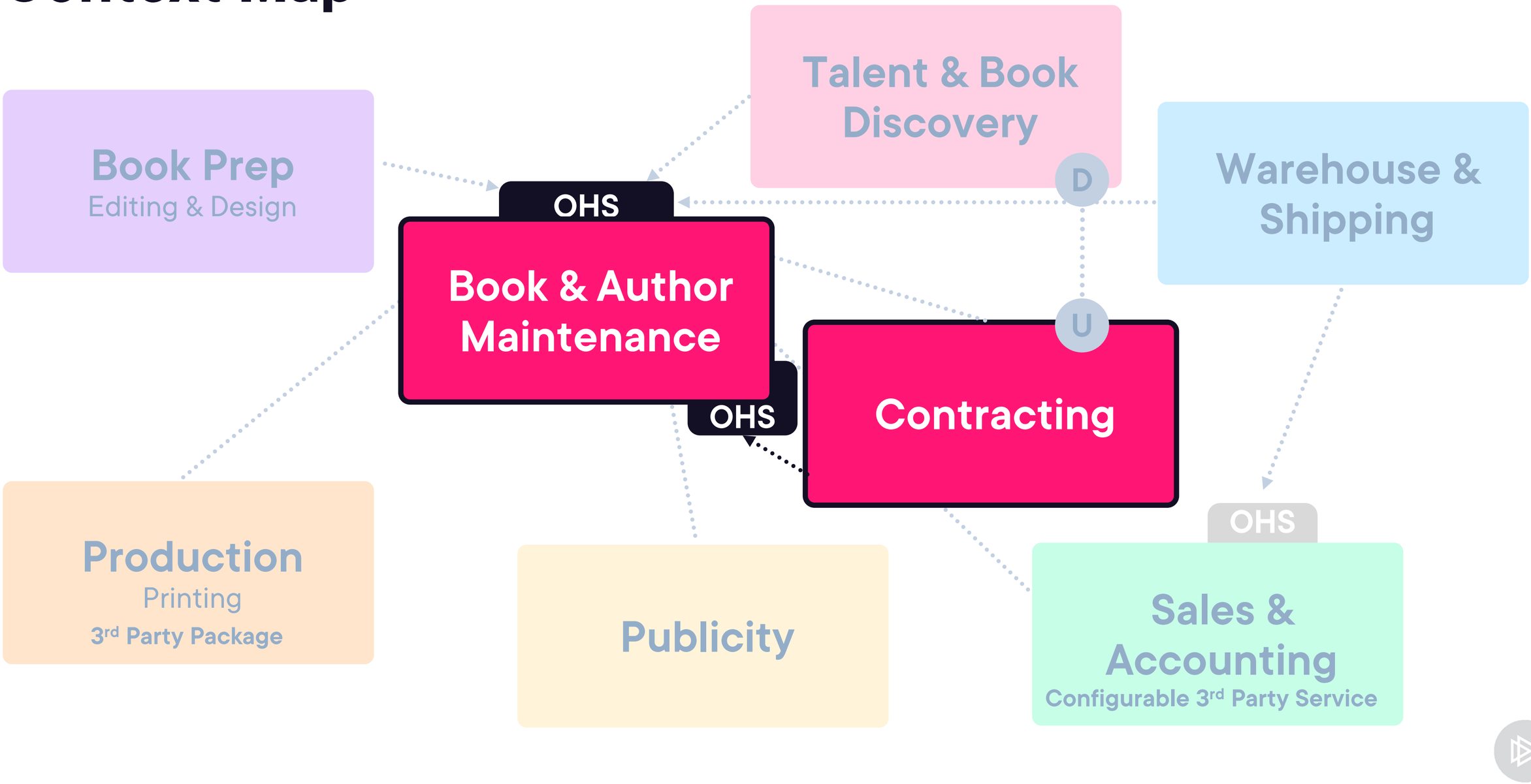
Choices to be made in modeling
Choices to be made in mapping

More on Advanced Mappings

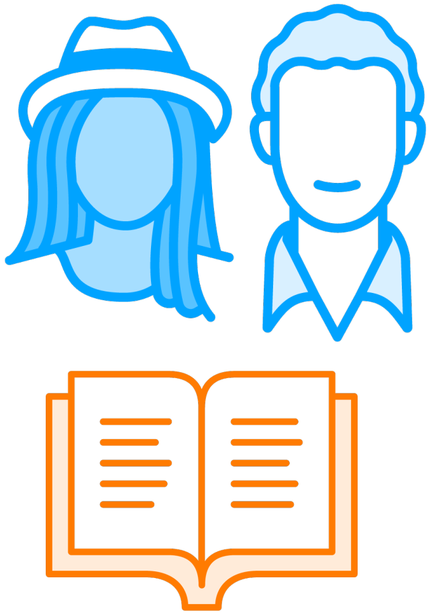
Mapping Real-world Classes in EF Core 6

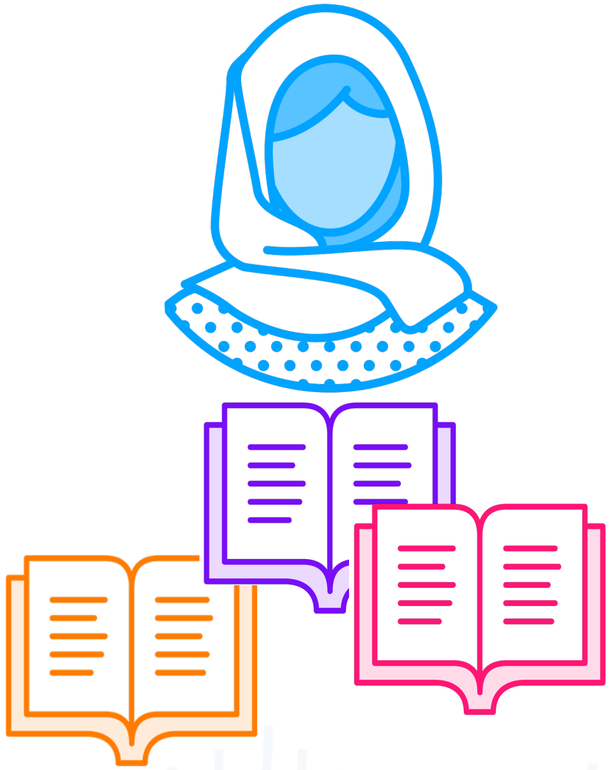
Torben Boeck Jensen

Context Map


Talent & Book Discovery
D
Book Prep
Editing & Design
Warehouse & Shipping
OHS
Book & Author Maintenance
U
OHS
Contracting
Production
Printing
3rd Party Package
Publicity
OHS
Sales & Accounting
Configurable 3rd Party Service

Authors Can Co-Author Books or Write Many Books





Author Book Maintenance Bounded Context

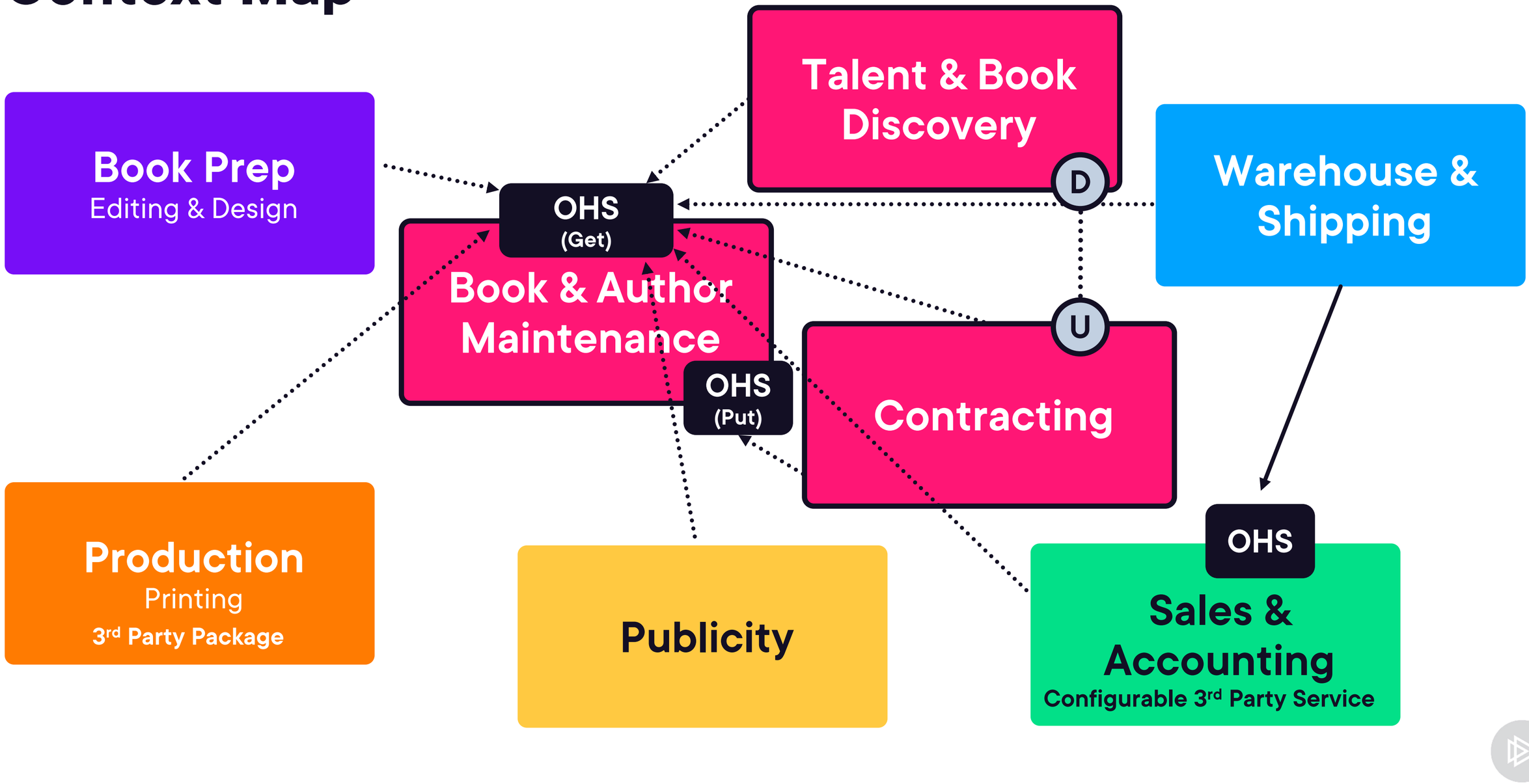
Does not have complex behaviors or invariants

Book and author can exist independently

There is no aggregate, nor an aggregate root

You can still benefit from DDD practices

Context Map


Book Prep
Editing & Design
Talent & Book Discovery
D
Warehouse & Shipping
OHS
(Get)
Book & Author Maintenance
OHS
(Put)
U
Contracting
Production
Printing
3rd Party Package
Publicity
OHS
Sales & Accounting
Configurable 3rd Party Service

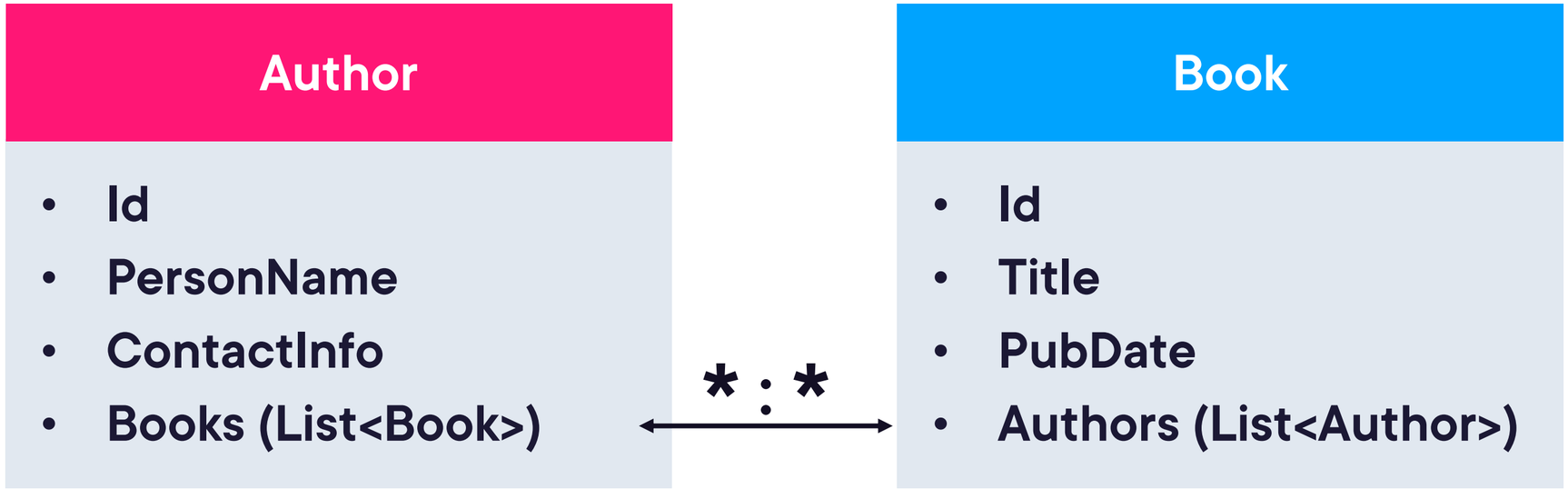
Evolution of EF Core's M2M Mappings

Early versions of EF Core forced you to comply with EF Core when designing M2M in your aggregate.

EF Core 5, 6 and 7 have brought vast improvements to M2M!

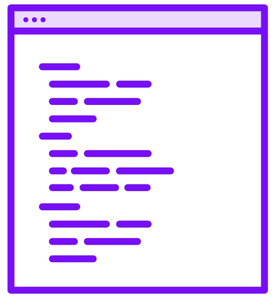
Starting with the Simplest Ways to Express and Map M2M

Simplest Expression of M2M in Code



EF Core Manages Skip Navigations

Join the Ends with Class Properties



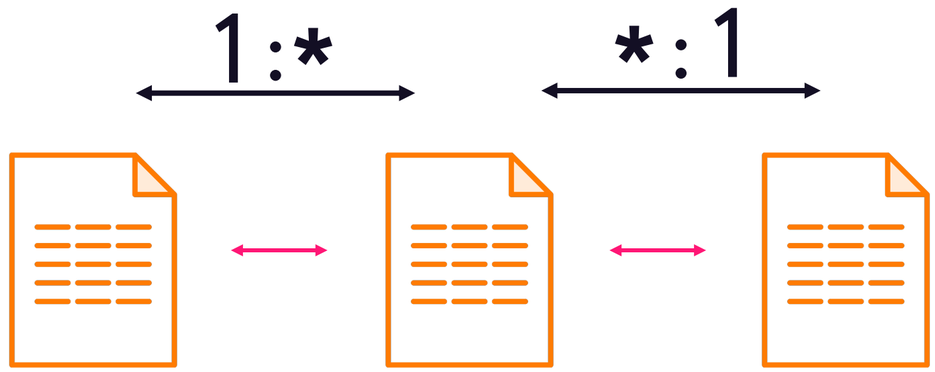
Author

List<Book>

Book

List<Author>

Relational Database: Join Table



Authors

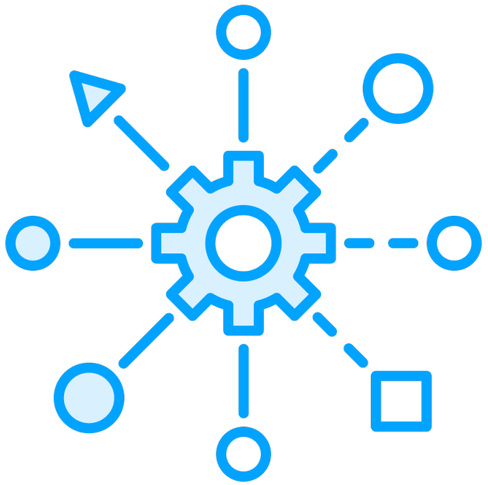
AuthorId

AuthorsBooks

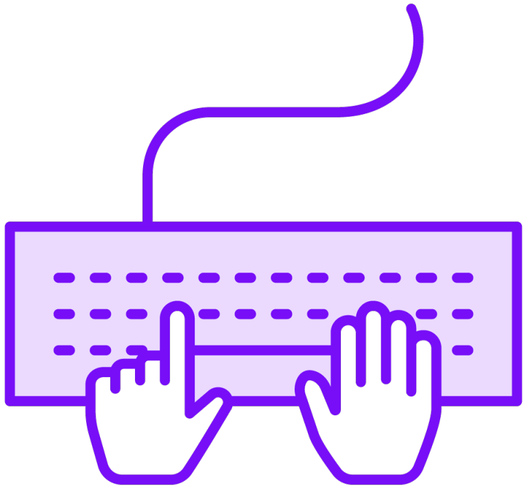
AuthorId
BookId

Books

BookId



EF Core Manages
the Underlying Relationship



You just focus on
Author.Books and Book.Authors

But does this make sense for our bounded context?

Author

- Id
- PersonName
- ContactInfo
- Books (List<Book>)

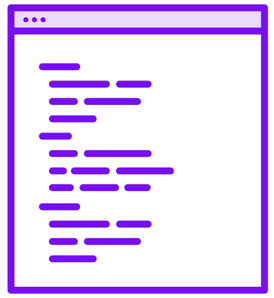
Book

- Id
- Title
- PubDate
- Authors (List<Author>)

Unidirectional Navigation Between M2M

Supported Starting with EF Core 7!

Only one end is aware of the relationship

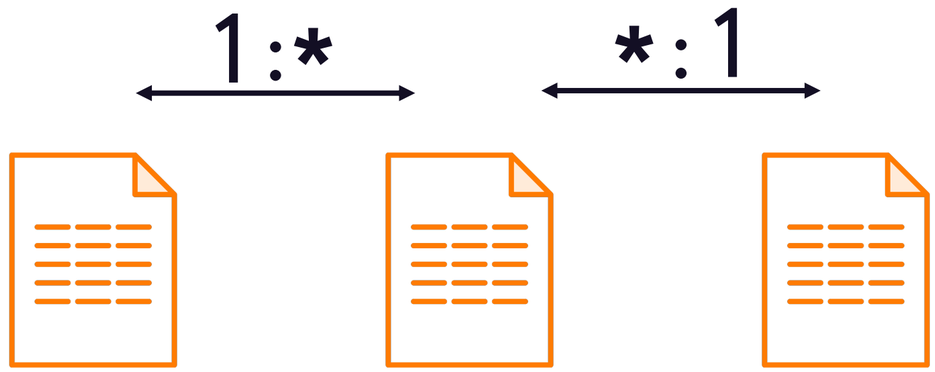


Author

List<Book>

Book

The database is aware of the full M2M



Authors

AuthorId

AuthorBook

AuthorId
BookId

Books

BookId

```
modelBuilder.Entity<Author>().HasMany(a => a.Books).WithMany();
```

HasMany/WithMany Mapping

EF Core 6: WithMany requires a parameter
EF Core 7: WithMany parameter can be empty
(There’s another, clunkier path for EF Core 6!)

But does this make sense for our bounded context?

Author
• Id • PersonName • ContactInfo • Books (List<Book>)

Book
• Id • Title • PubDate

Refactoring to Suit the Needs of the Bounded Context

Using a List of GUIDs to Identify the Authors

Satisfies the needs of the aggregate

EF Core only recognizes
lists of entities,
not lists of scalars!

But this is the design that I want!

Value Converters and JSON to the Rescue!

Here We Need that Unidirectional Navigation

EF Core 7: The easy way

```
modelBuilder.Entity<Author>()
.HasMany(a => a.Books).WithMany();
```

EF Core 6: The annoying way, but it works

```
[In Book.cs]
private List<Author> _authors;

modelBuilder.Entity<Author>()
.HasMany(a => a.Books).WithMany("_authors");
```

Now We Really Need that Unidirectional Navigation

```
modelBuilder.Entity<Author>()
.HasMany(a => a.Books)
.WithMany();
```

```
private List<Author> _authors;
```

```
modelBuilder.Entity<Author>()
.HasMany(a => a.Books)
.WithMany("_authors");
```

EF Core 7: The easy way

- HasMany/WithMany with empty parameter

EF Core 6: The annoying way, but it works

- Private field in the class that should not expose the other end (i.e., Book)
- Map the HasMany/WithMany pointing to that private field

But does this make sense for our bounded context?

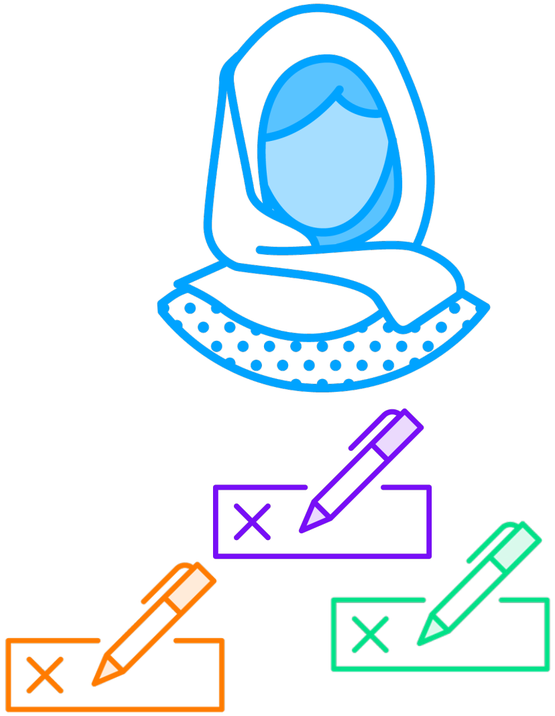
Author
• Id • PersonName • ContactInfo • Books (List<Book>)

Book
• Id • Title • PubDate • AuthorIds (List<Guids>)

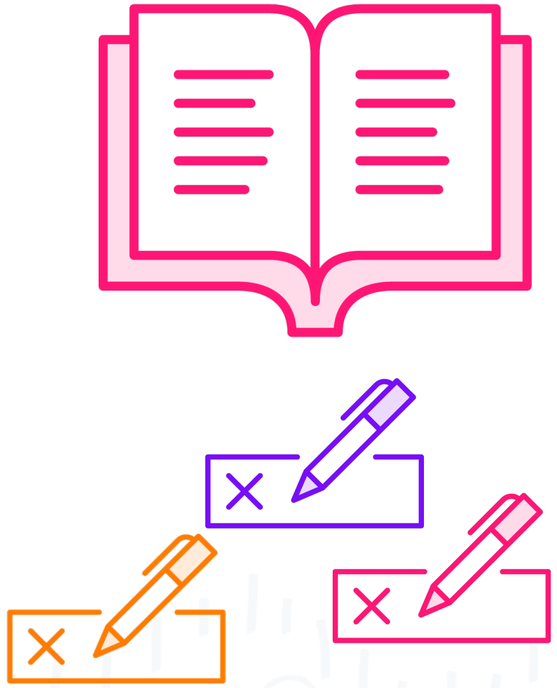
Refactoring the domain & remapping EF Core one last time!

Exploring Another Possibility for the Many-to-Many Relationship

Authors Can Sign Many Contracts

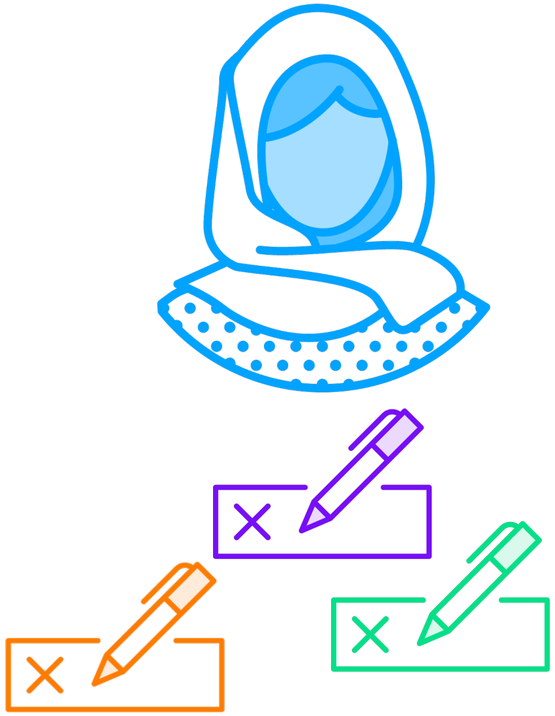


Many Contracts for One Book?



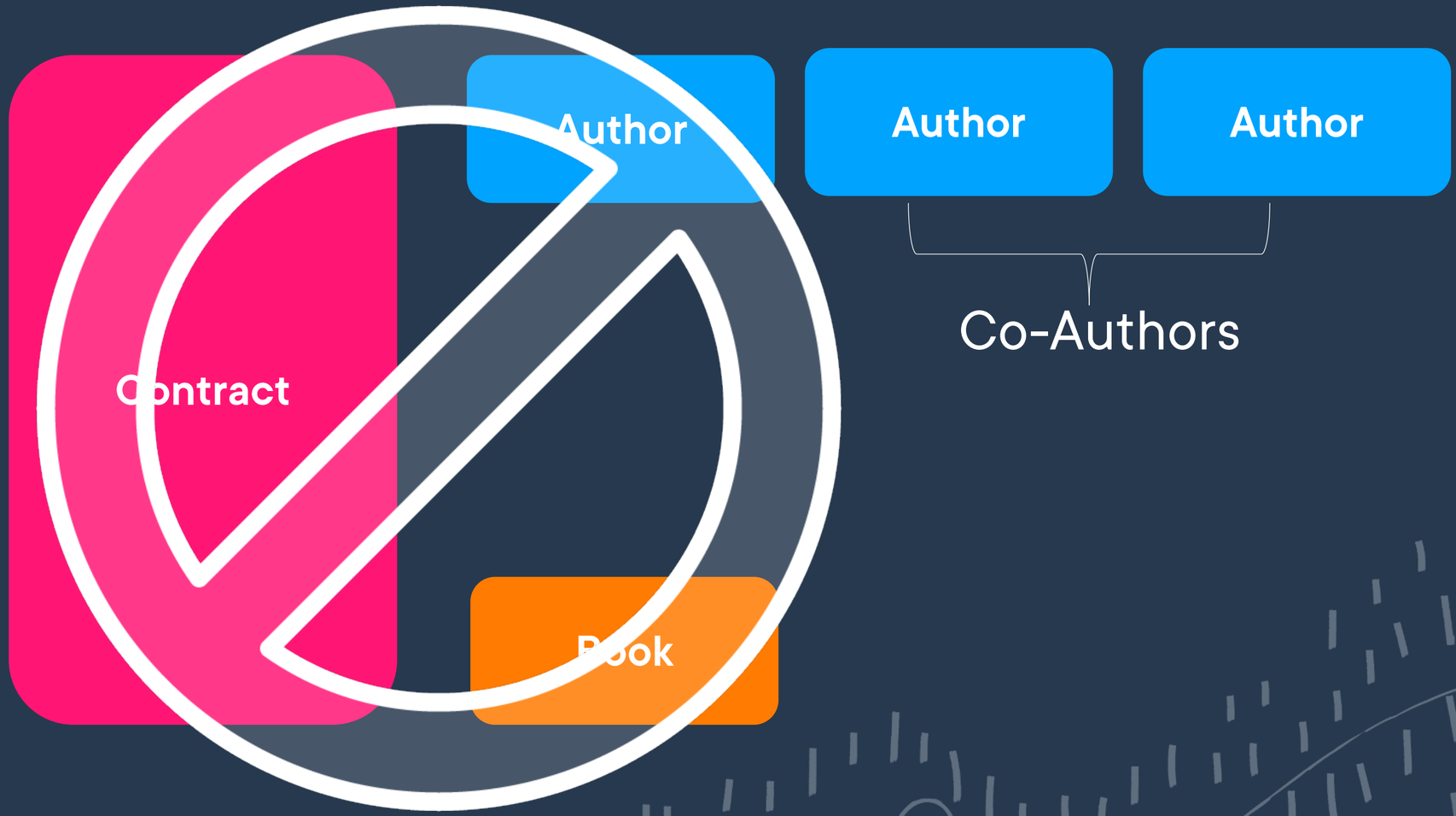
In Our Publishing House

Authors can sign many contracts

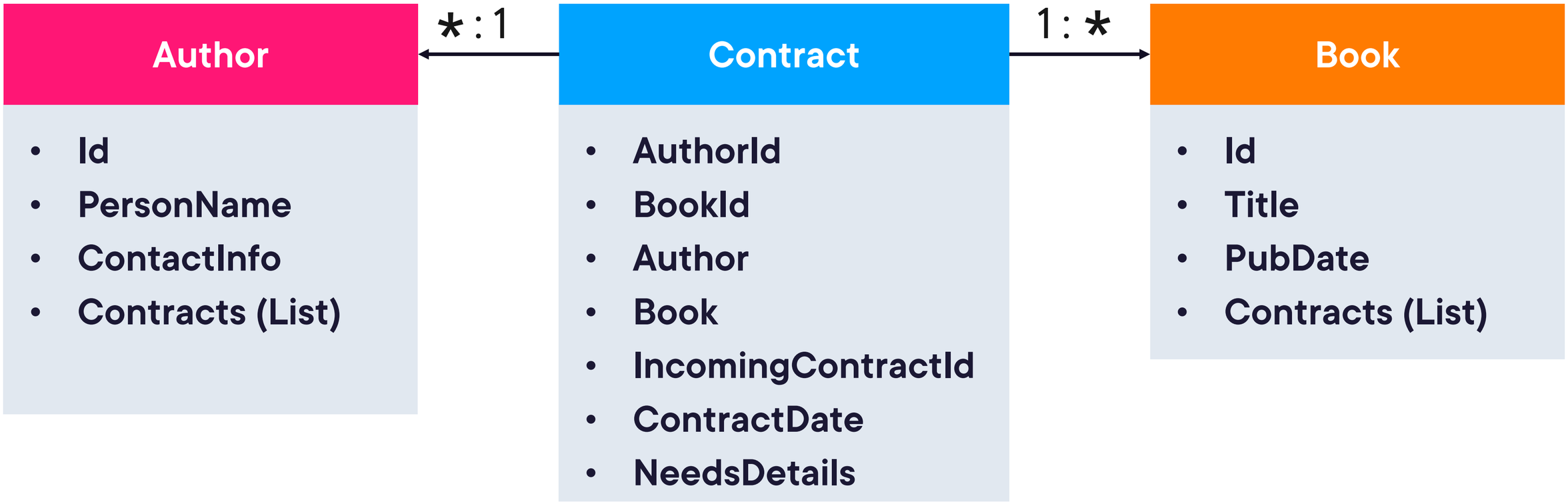


Only one contract per book

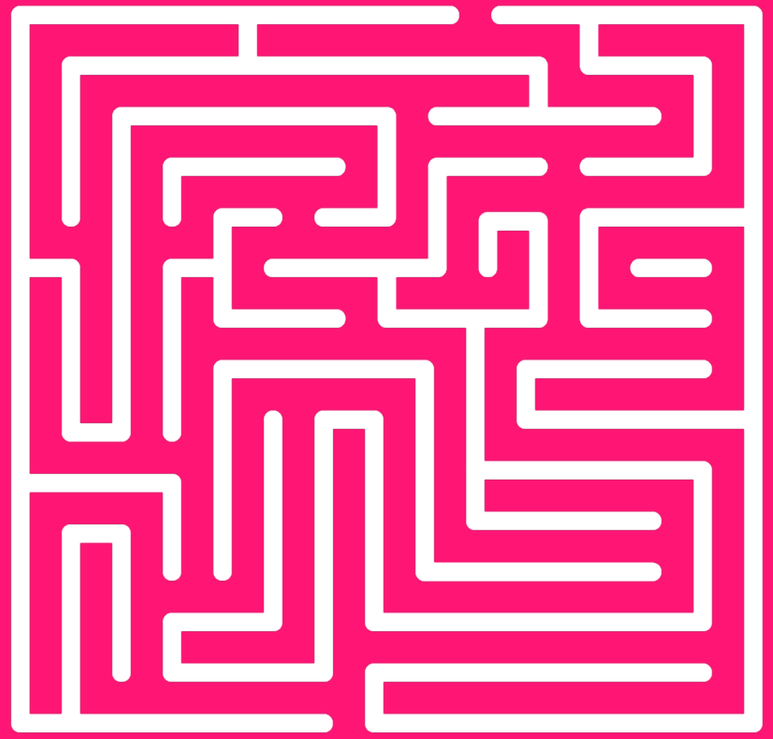
Is Contract an Aggregate Root?



What If a Contract is What Joins Author and Book?



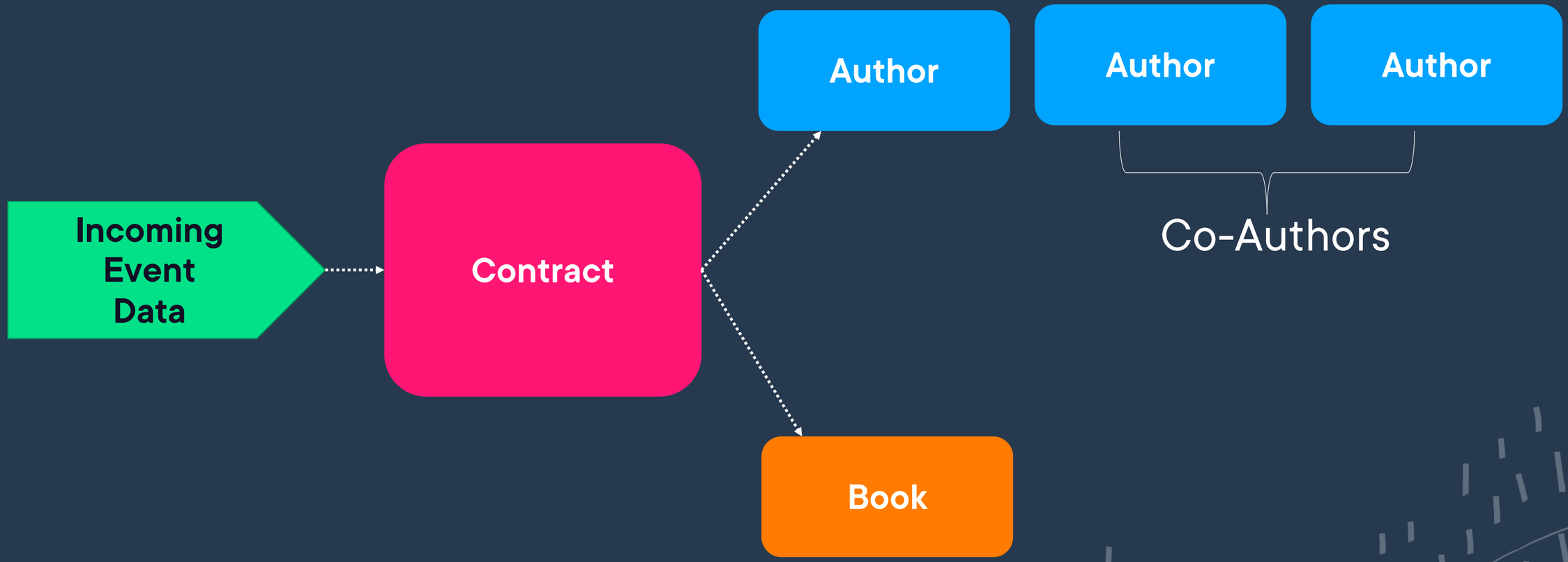
"Indirect Many-to-Many Relationship"



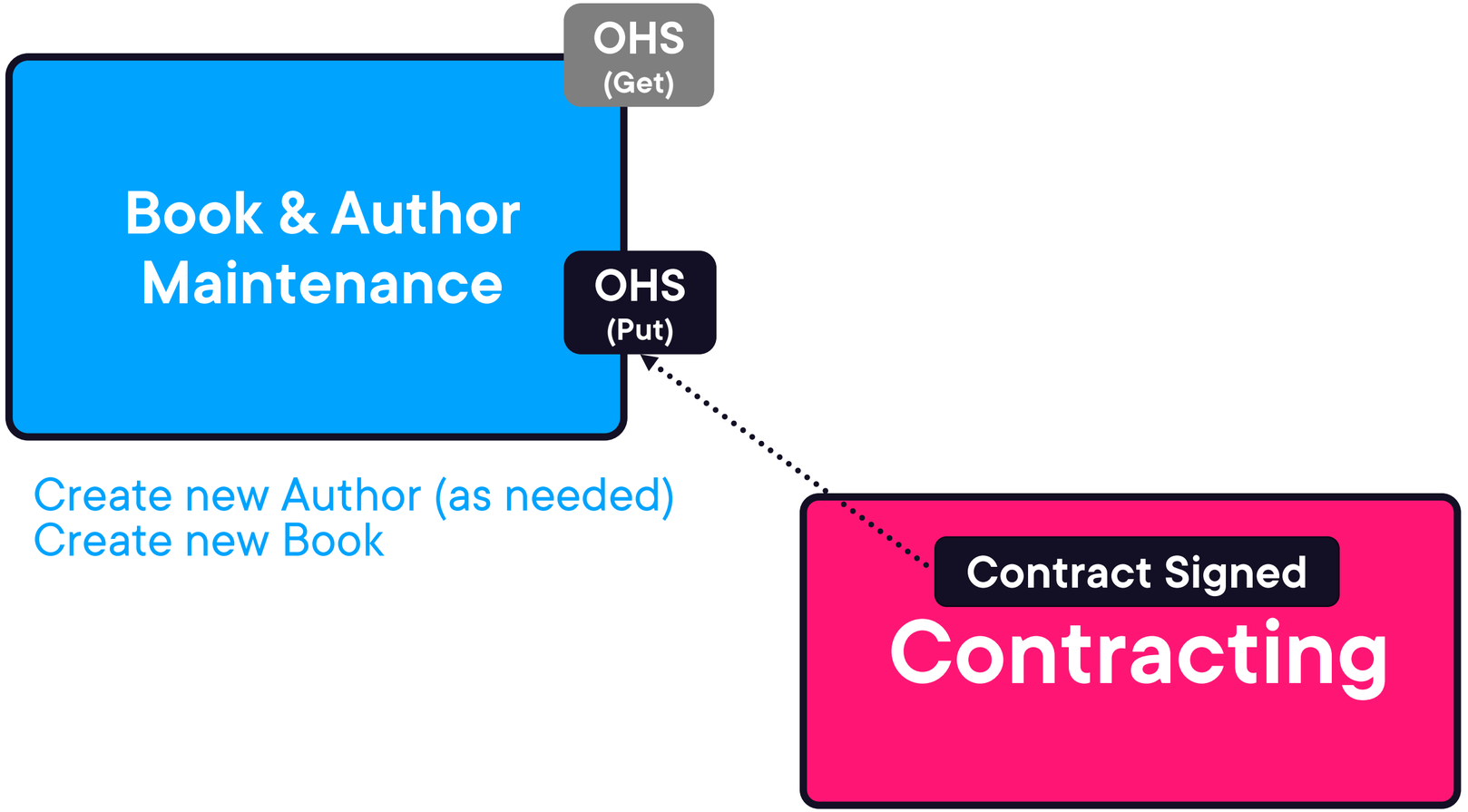
Complications are a “red flag” in DDD

I’d rather have complexity in my persistence than in my domain logic

Contract is a Conduit



Message Contains Author & Book Info



- Create new Author (as needed)
- Create new Book

Overview

Domain design & EF Core mapping offer a lot of flexibility

Think out of the “go-to patterns” box for M2M

M2M skip navigations are the simplest

Unidirectional navigation is easier in 7

Indirect M2M allows for extra data, but is trickier to use

Don’t forget JSON + value converters!

Up Next:

Mapping Aggregates to Azure CosmosDB