DDD and My ORM

Matthias Noback

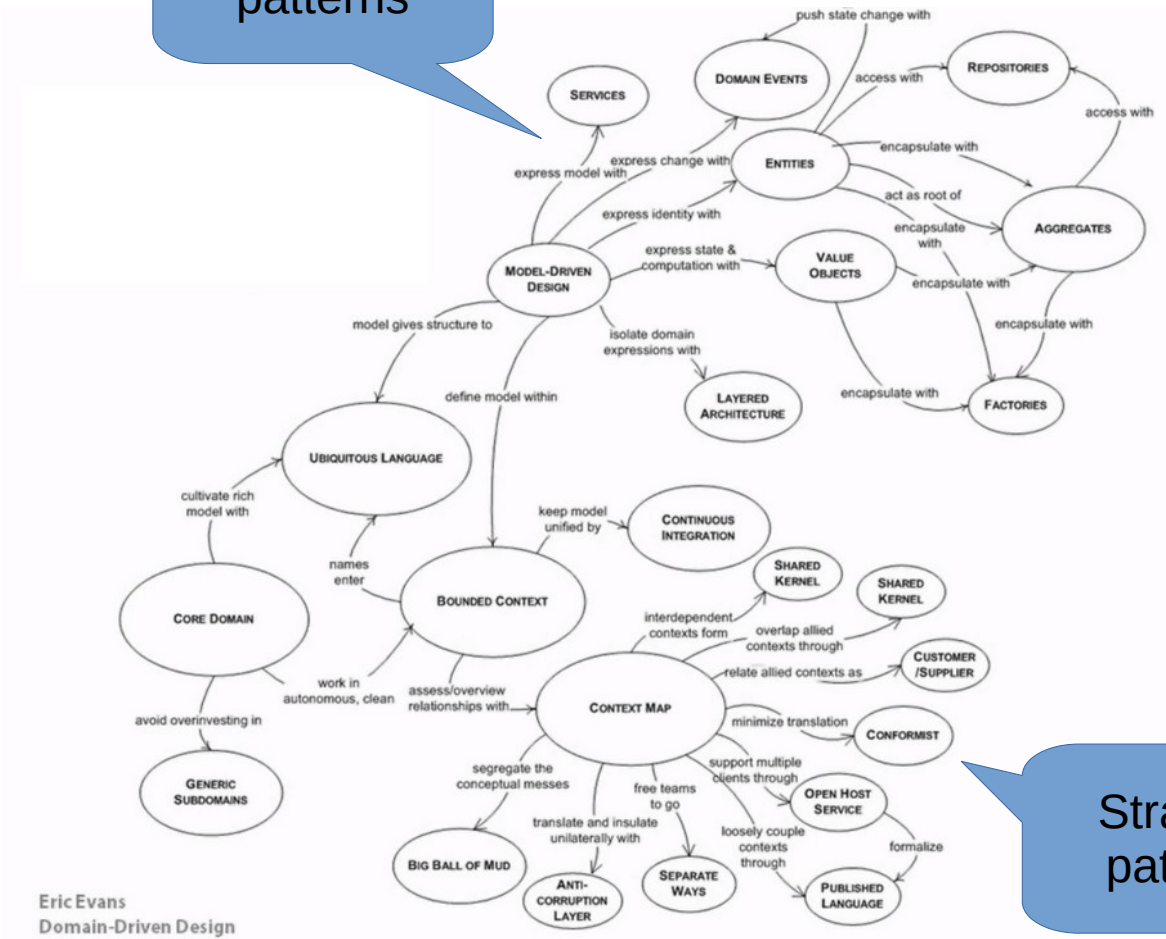
https://matthiasnoback.nl/

Background

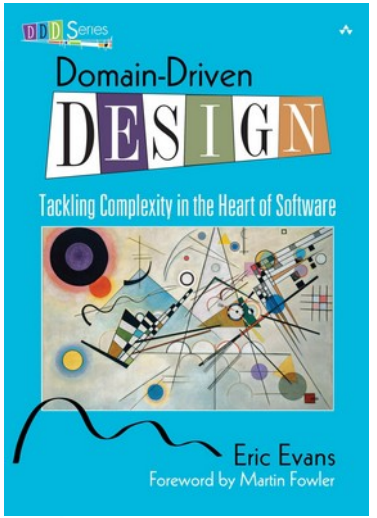
- Working in PHP web development since 2002
- Specialized in legacy modernization projects
- Trainer and consultant
- Technical author

DDD?

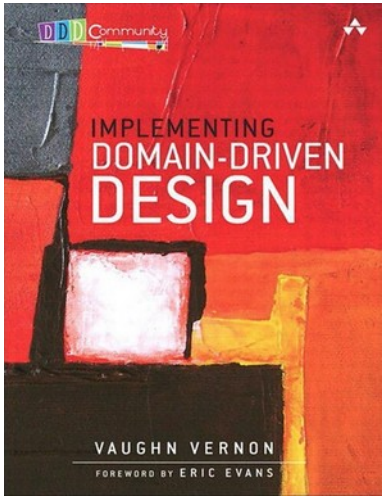
Tactical patterns



Eric Evans
Domain-Driven Design



2003



2013

Strategic patterns

An ORM?

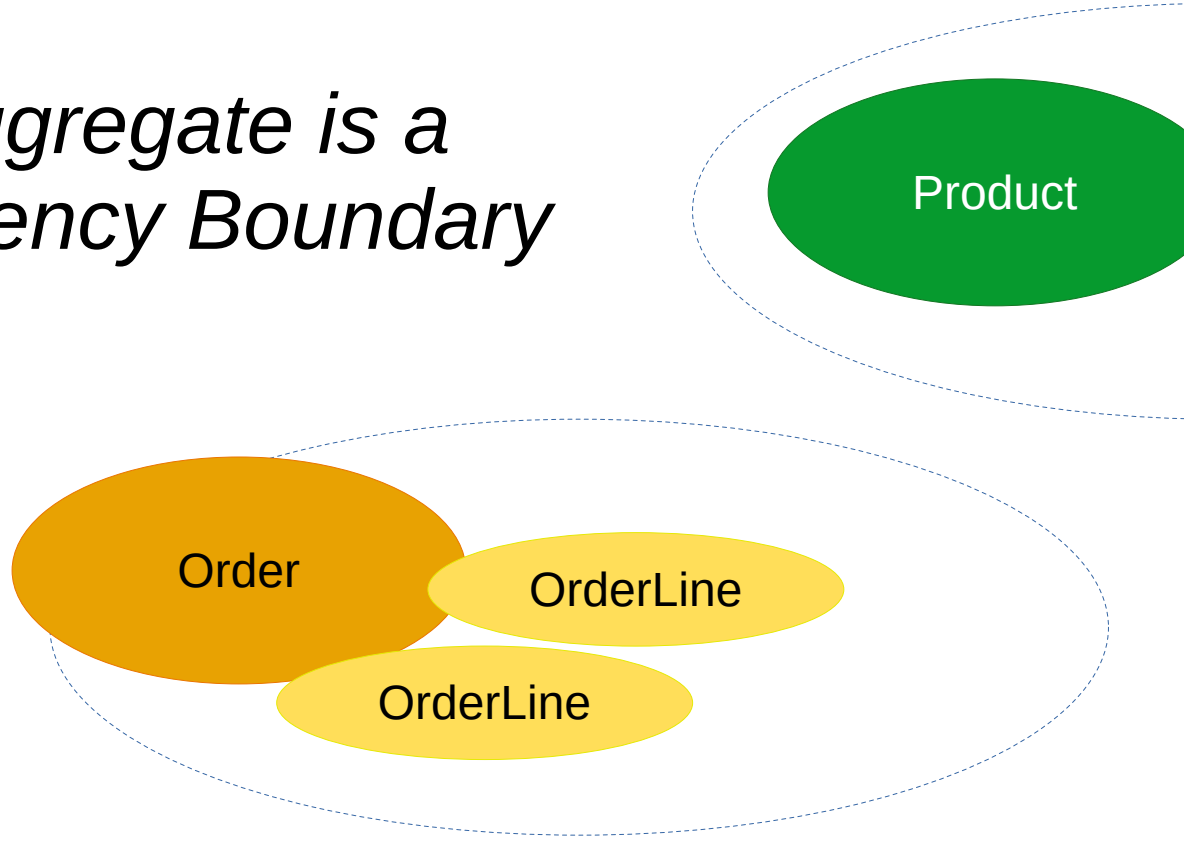
- **Maps** data in a database to objects in application memory, and back
- Including the **relations** between tables, maps them too
- Thinking mostly of **Doctrine ORM**

Objects?

- **Entities**
- Models
- Data objects

Summary of Tactical DDD

An Aggregate is a Consistency Boundary



Hypothesis:

Your ORM entity ~= a DDD aggregate

Choose Your Own Adventure

Domain Invariants	Intention-Revealing Methods	Lifecycle and State Machine
Root Entities vs Child Entities	Value Objects	Entity Identity
Transaction Size	Domain Events	Aggregate Relations
Write vs Read Model	Bounded Contexts	Conclusion

An Aggregate Protects its Domain Invariants

Invariant?

- Something that’s always true
- For models: a rule
- *Examples?*

Tips

- Find out what the actual rules are
- Typical counter-example: uniqueness, maximum string length, value is > 0

An ORM entity

- Offers **setters** for every field
- **Doesn’t enforce invariants** (other than nullability)
- Anything is allowed
- **Validation** happens **after** assigning or not all

```php
<?php
class User
{
    private $username;
    private $passwordHash;
    private $bans;

    public function getUsername(): string
    {
        return $this->username;
    }

    public function setUsername(string $username): void
    {
        $this->username = $username;
    }

    public function getPasswordHash(): string
    {
        return $this->passwordHash;
    }

    public function setPasswordHash(string $passwordHash): void
    {
        $this->passwordHash = $passwordHash;
    }
```

https://www.doctrine-project.org/projects/doctrine-orm/en/3.6/tutorials/getting-started.html#starting-with-the-product-entity

```php
// ...
use App\Entity\Author;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Validator\Validator\ValidatorInterface;

// ...
public function author(ValidatorInterface $validator): Response
{
    $author = new Author();

    // ... do something to the $author object

    $errors = $validator->validate($author);
```

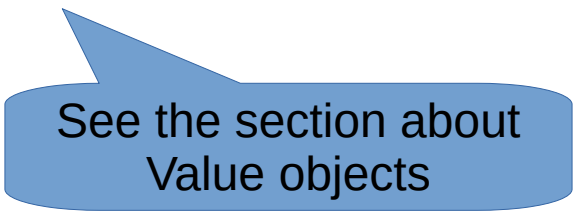
https://symfony.com/doc/current/validation.html

```php
// set up a fresh $task object (remove the example data)
$task = new Task();

$form = $this->createForm(TaskType::class, $task);

$form->handleRequest($request);
if ($form->isSubmitted() && $form->isValid()) {
    // $form->getData() holds the submitted values
    // but, the original `$task` variable has also been updated
    $task = $form->getData();

    // ... perform some action, such as saving the task to the database

    return $this->redirectToRoute('task_success');
}
```

https://symfony.com/doc/current/forms.html#validating-forms

How to improve?

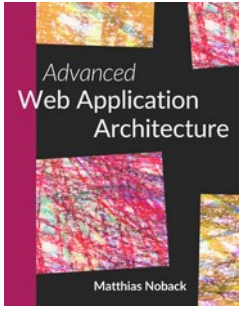
- Offer **real protection**: throw exceptions in modifiers and constructors
- Use **Value objects**:
  - As evidence of validation
  - To reuse validation logic



```
public function __construct(string $emailAddress)
{
    // Before assigning, we verify that the email address is valid
    if (!filter_var($emailAddress, FILTER_VALIDATE_EMAIL)) {
```



```
        // We don't return an error, but throw an exception
        throw new InvalidArgumentException(
            sprintf('Invalid email address: ' . $emailAddress)
        );
    }

    $this->emailAddress = $emailAddress;
}
```



```php
final class Line
{
    private EbookId $ebookId;

    private int $quantity;

    public function __construct(EbookId $ebookId, int $quantity)
    {
        if ($quantity <= 0) {
            throw new InvalidArgumentException(
                sprintf('Line quantity should be at least 1')
            );
        }

        $this->ebookId = $ebookId;
        $this->quantity = $quantity;
    }
}
```

```php
final class Quantity
{
    private int $quantity;

    public function __construct(int $quantity)
    {
        if ($quantity <= 0) {
            throw new InvalidArgumentException(
                sprintf('Line quantity should be at least 1')
            );
        }

        $this->quantity = $quantity;
    }
}
```

```php
    public function addLine(EbookId $ebookId, Quantity $quantity): void
    {
        $this->lines[] = new Line($ebookId, $quantity);
    }
}

final class Line
{
    private EbookId $ebookId;

    private Quantity $quantity;

    public function __construct(EbookId $ebookId, Quantity $quantity)
    {
        $this->ebookId = $ebookId;
        $this->quantity = $quantity;
    }
}
```

Aggregates Have Intention-Revealing Methods

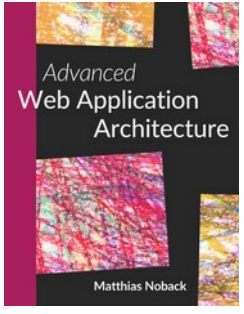
ORM entities

- `new User()`
  What does it mean in each case?
- `setIsActive(false)`
  Can we really toggle arbitrarily between true and false?
- `setStreet() setCity() setCountry()`
  What are we really doing? Can we change these values separately?

Intention-revealing?

- What are you trying to accomplish? Is there some concept behind it? What's the action we're doing here?
- `User::signUp()` or `User::importFromLegacySystem()`
- `$account->deactivate()`
- `$contact->changeAddress()`

```php
}

final class ChangeDeliveryAddressService
{
    private OrderRepository $orderRepository;

    public function __construct(OrderRepository $orderRepository)
    {
        $this->orderRepository = $orderRepository;
    }

    public function __invoke(OrderId $orderId /* ... */): void
    {
        $order = $this->orderRepository->getById($orderId);

        $order->changeDeliveryAddress(/* ... */);

        $this->orderRepository->save($order);
    }
}
```



```php
final class OrderService
{
    private OrderRepository $orderRepository;

    public function __construct(
        OrderRepository $orderRepository
    ) {
        $this->orderRepository = $orderRepository;
    }

    public function createOrder(/* ... */): OrderId
    {
        $orderId = $this->orderRepository->nextIdentity();

        // ...

        return $orderId;
    }

    public function changeDeliveryAddress(/* ... */): void
    {
        // ...
    }

    public function markAsDelivered(/* ... */): void
    {
        // ...
    }

    public function cancel(/* ... */): void
    {
        // ...
    }
}
```

An Aggregate has an Explicit Lifecycle and Behaves Like a State Machine

Lifecycle?

- Entity is created for some reason
- May be modified several times
- Might be finalized in some way
- Becomes obsolete (“long tail”)

State machine

- Depending on the phase of its life, only certain actions are possible
- *Examples?*

With an ORM

- Fetch any entity at any time
- Do anything with it at any time
- The object enforces nothing

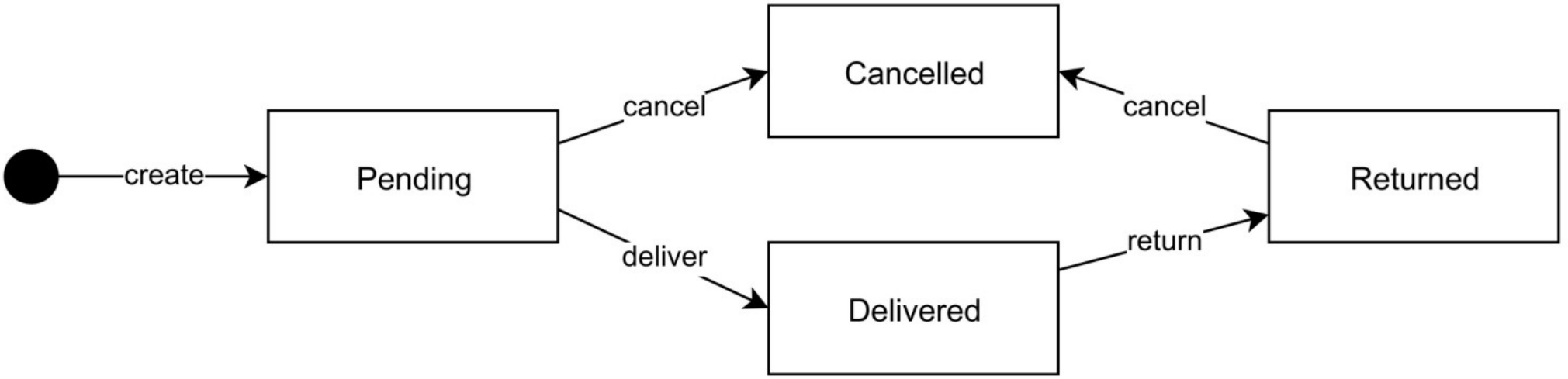


Figure 11.1. Entity states and state transitions

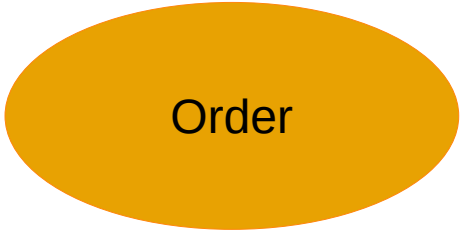
Tips

- Keep an indicator of the current state in the entity (don’t offer a setter for it!)
- Guard against invalid actions, given the current state

An Aggregate Consists of a Root Entity and any Number of Child Entities

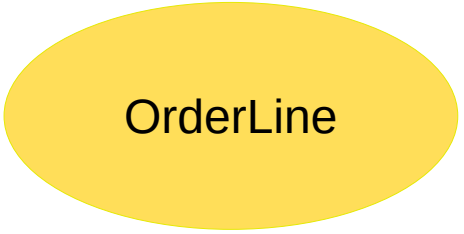
Root entity

- Has an absolute ID
- Has a repository for saving and retrieving instances
- Tracks change



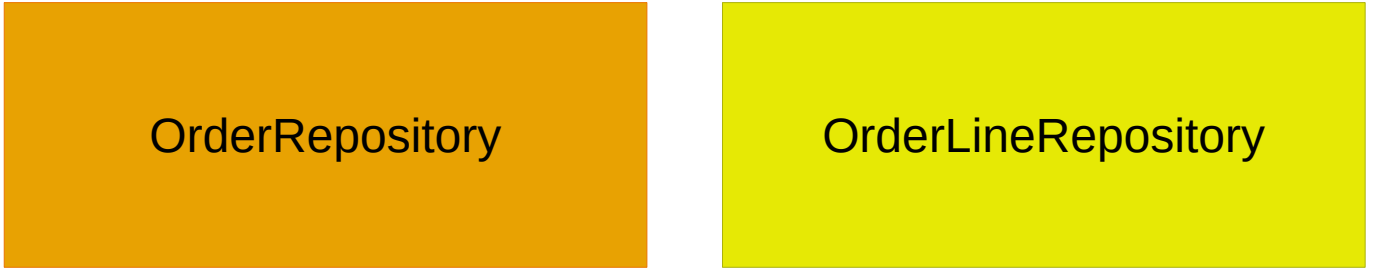
Child entity

- Has an ID relative to its root entity
- Modified only via its root entity
- No repository



ORM

- Allows fetching a child entity directly from its repository
- (Gives it a global ID)
- Allows modifing child entities without talking to the root entity



How to improve?

- Don’t use child entity repositories
- Don’t expose child entities via getters/setters
- Result: **encapsulated child entities**

Value Objects Describe Aspects of an Entity

Value object

- Is immutable
- Has no identity
- Offers protection
- Provides evidence of validation

```php
final readonly class Money
{
    private float $amount;
    private string $currency;

    public function __construct(float $amount, string $currency)
    {
        $this->ensureAmountIsPositive($amount);

        $this->amount = $amount;
        $this->currency = $currency;
    }

    public function getAmount(): float
    {
        return $this->amount;
    }

    public function getCurrency(): string
    {
        return $this->currency;
    }

    public function add(Money $other): Money
    {
        $this->assertSameCurrency($other);

        return new Money($this->amount + $other->getAmount(), $this->currency);
    }
```

https://wendelladriel.com/blog/understanding-value-objects-in-php

```php
<?php
#[Entity]
class User
{
    // ...

    #[ManyToOne(targetEntity: Address::class)]
    #[JoinColumn(name: 'address_id', referencedColumnName: 'id')]
    private Address|null $address = null;
}

#[Entity]
class Address
{
    // ...
}
```

```php
<?php

#[Entity]
class User
{
    #[Embedded(class: Address::class)]
    private Address $address;
}

#[Embeddable]
class Address
{
    #[Column(type: "string")]
    private string $street;

    #[Column(type: "string")]
    private string $postalCode;

    #[Column(type: "string")]
    private string $city;

    #[Column(type: "string")]
    private string $country;
}
```

https://www.doctrine-project.org/projects/doctrine-orm/en/3.6/tutorials/embeddables.html

Mixed roles

- Is identity required?
  - Do we need to access it in a stand-alone way?
    - Yes: Root entity
    - No: Child entity
- If not, make it a Value object

Tips

- When using an ORM, manually map from and to primitive values

```php
/**
 * @ORM\Column(type="string")
 * @var string
 */
private $supplierId;

private function __construct(
    PurchaseOrderId $purchaseOrderId,
    SupplierId $supplierId
) {
    // ...

    // convert to a string
    $this->supplierId = $supplierId->asString();
}

public function supplierId(): SupplierId
{
    // and back to an object, if necessary
    return SupplierId::fromString($this->supplierId);
}
```

https://matthiasnoback.nl/2018/06/doctrine-orm-and-ddd-aggregates/

An Aggregate’s Root Entity has an Absolute ID

ORM entity

- **Begins life without an ID**
- After saving, it gets an **auto-generated ID** from the database
- Mostly a problem if you want to use **Domain events**

```php
<?php
$user = new User;
$user->setName('Mr.Right');
$em->persist($user);
$em->flush();
```

How to improve?

- Generate an ID in the application:
  - Get next ID from a sequence
  - Or generate a UUID, etc.

Each Transaction Contains Changes for Just One Aggregate

DDD: Scaling your Domain Model

- The more you change in one transaction,
- The likelier you run into conflicts

The Aggregate as Consistency Boundary

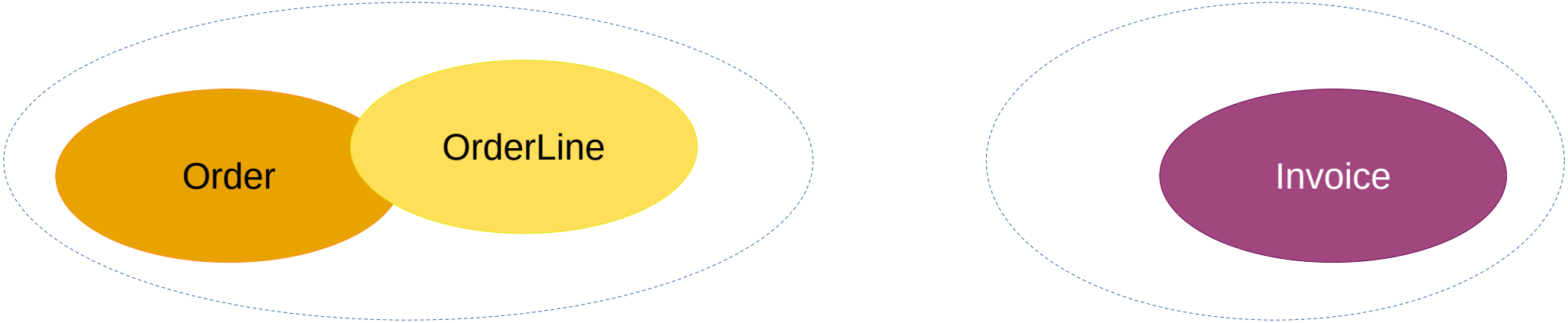
- Fetch an aggregate
- Make a change
- Persist the change
- (repeat if necessary)

ORM

- Has a **Unit-of-Work**
- Allows fetching and changing **any number of entities**
- When flushing, **persists all changes**

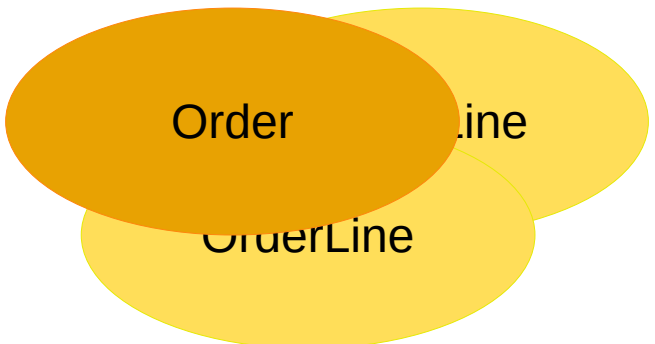
How to improve?

- Change only one aggregate (root entity and child entities), flush, then clear the entity manager
- Make further changes in another transaction



How to improve? - 2

- Cascade persist/remove only for child entities
- On delete cascade for child entities



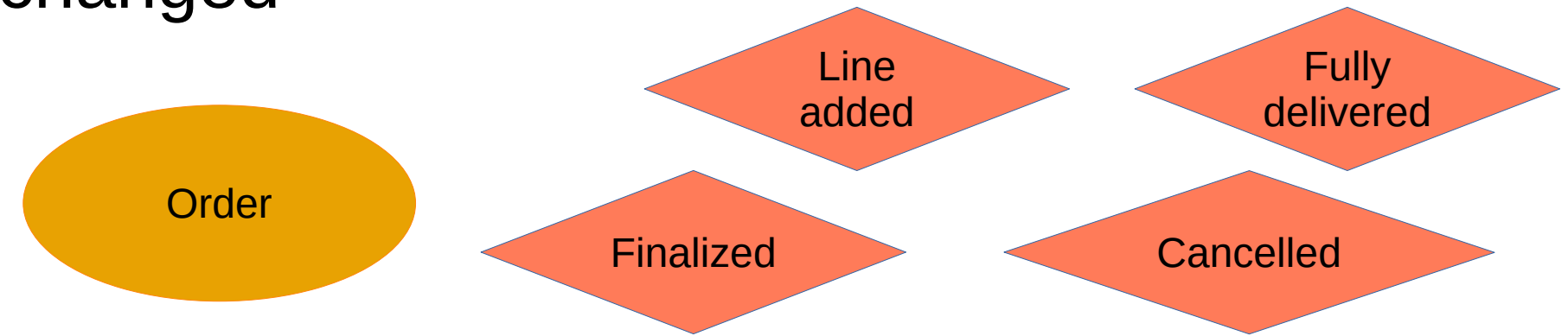
What should change in the first transaction?

- Do what the user wants, and save the data the user has provided
- Everything else is about the system catching up with the change (eventual consistency): effects

Aggregates Express Change Using Domain Events

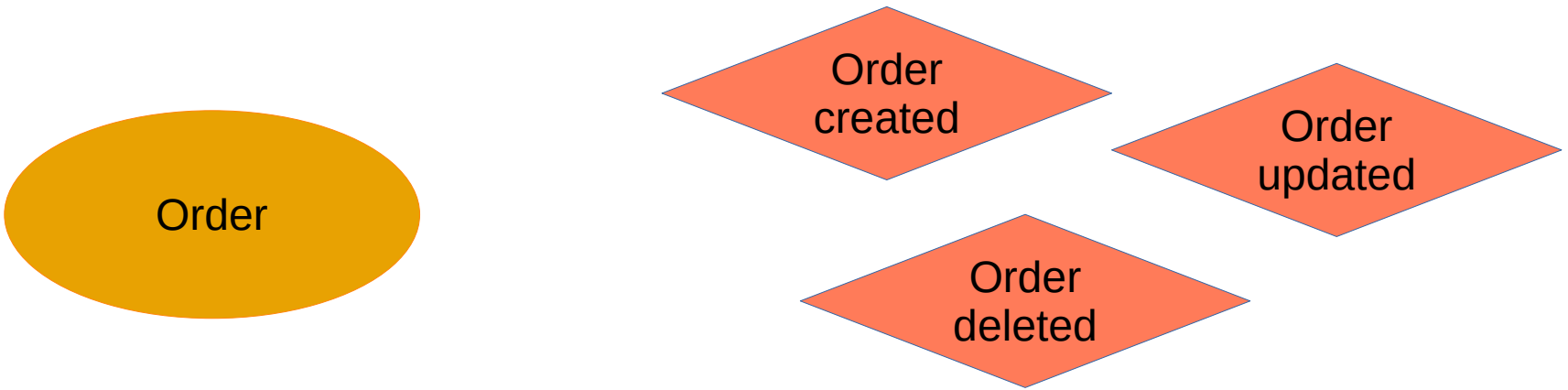
Aggregate

- Modifies its own state
- May decide not to make a change
- Is the only thing that can tell if and what really changed



ORM

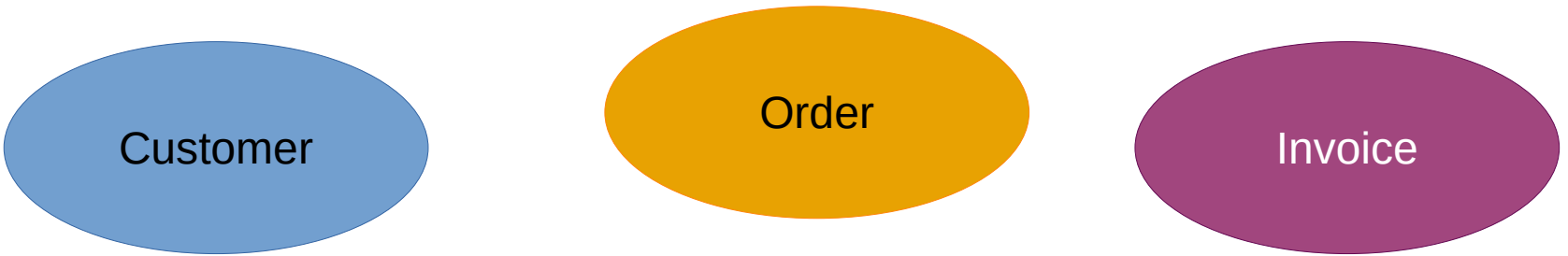
- Dispatches only generic events
- Listeners need to find out what changed by inspecting the event data



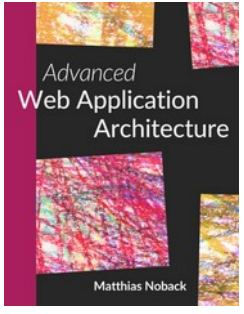
Event	Dispatched by	Lifecycle Callback	Passed Argument
preRemove	`$em->remove()`	Yes	PreRemoveEventArgs
postRemove	`$em->flush()`	Yes	PostRemoveEventArgs
prePersist	`$em->persist()` on ***initial*** persist	Yes	PrePersistEventArgs
postPersist	`$em->flush()`	Yes	PostPersistEventArgs
preUpdate	`$em->flush()`	Yes	PreUpdateEventArgs
postUpdate	`$em->flush()`	Yes	PostUpdateEventArgs
postLoad	Loading from database	Yes	PostLoadEventArgs
loadClassMetadata	Loading of mapping metadata	No	LoadClassMetadataEventArgs
`onClassMetadataNotFound`	`MappingException`	No	OnClassMetadataNotFoundEventArgs
preFlush	`$em->flush()`	Yes	PreFlushEventArgs
onFlush	`$em->flush()`	No	OnFlushEventArgs
postFlush	`$em->flush()`	No	PostFlushEventArgs
onClear	`$em->clear()`	No	OnClearEventArgs

How to improve?

- Collect explicit domain events inside the entity
- Dispatch them after saving/flushing
- *Examples?*



```php
final class Order
{
    // ...

    public function markLineAsDelivered(int $lineNumber): void
    {
        $this->line($lineNumber)->markAsDelivered();
        $this->events[] = new LineDelivered($this->id, $lineNumber);

        if ($this->allLinesHaveBeenDelivered()) {
            $this->events[] = new OrderFullyDelivered($this->id);
        }
    }
}
```



```php
public function create(CreateOrder $command): OrderId
{
    // ...

    $order = new Order(/* ... */);

    // First, save the Order

    // Then dispatch the events that were recorded inside the event:
    $this->eventDispatcher->dispatchAll($order->releaseEvents());

    return $orderId;
}
```

```php
final class CreateInvoice
{
    private InvoicingService $invoicingService;

    public function __construct(
        InvoicingService $invoicingService
    ) {
        $this->invoicingService = $invoicingService;
    }

    public function whenOrderFullyDelivered(
        OrderFullyDelivered $event
    ): void {
        $this->invoicingService->createInvoiceFromOrder(
            $event->orderId(),
            /* ... */
        );
    }
}
```

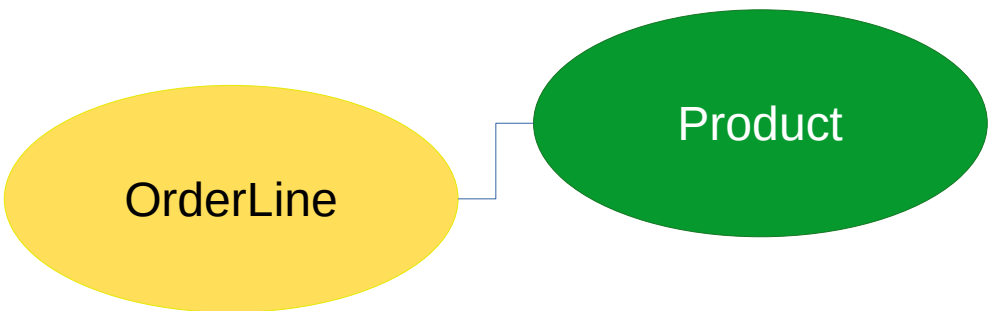
Aggregates Refer to Other Aggregates by their ID Only

Aggregate relations by ID only

- To ensure the aggregate doesn’t cross its own boundaries

ORM

- Needs entire objects



```php
<?php
class Article
{
    private Collection $tags;

    public function addTag(Tag $tag): void
    {
        $tag->addArticle($this); // synchronously updating inverse side
        $this->tags[] = $tag;
    }
}

class Tag
{
    private Collection $articles;

    public function addArticle(Article $article): void
    {
        $this->articles[] = $article;
    }
}
```

How to improve

- Obey the ORM: pass entire objects, not just IDs
- Just don’t read from or change passed objects from other aggregates

18

You can use Reference Proxies and do something like this:

```
$categoryId = 25;
$product->setCategory(
    $entityManager->getReference(Category::class, $categoryId)
);
$entityManager->persist($product);
$entityManager->flush();
```

Copy

Share Improve this answer Follow

answered Oct 11, 2021 at 12:25

Michaël Perrin

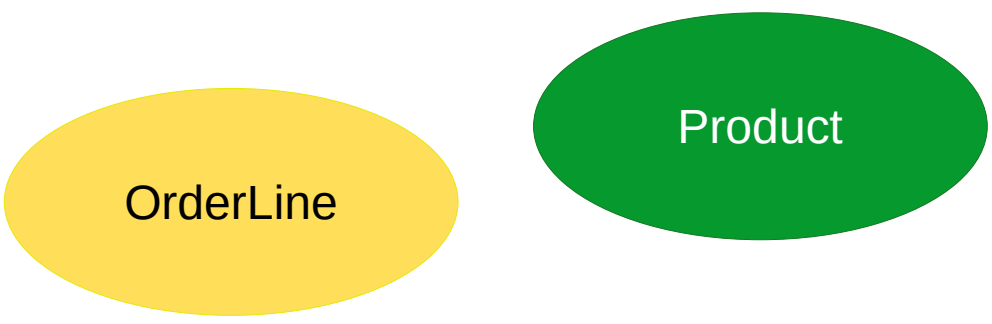
6,328 ● 5 ● 45 ● 70

https://stackoverflow.com/questions/60337274/doctrine-use-only-the-id-to-set-a-related-entity

Aggregates Are Used Only To Make Changes

Aggregate

- The amount of available state is limited
- Doesn’t modify or read beyond its own boundaries
- State is encapsulated, not exposed via getters

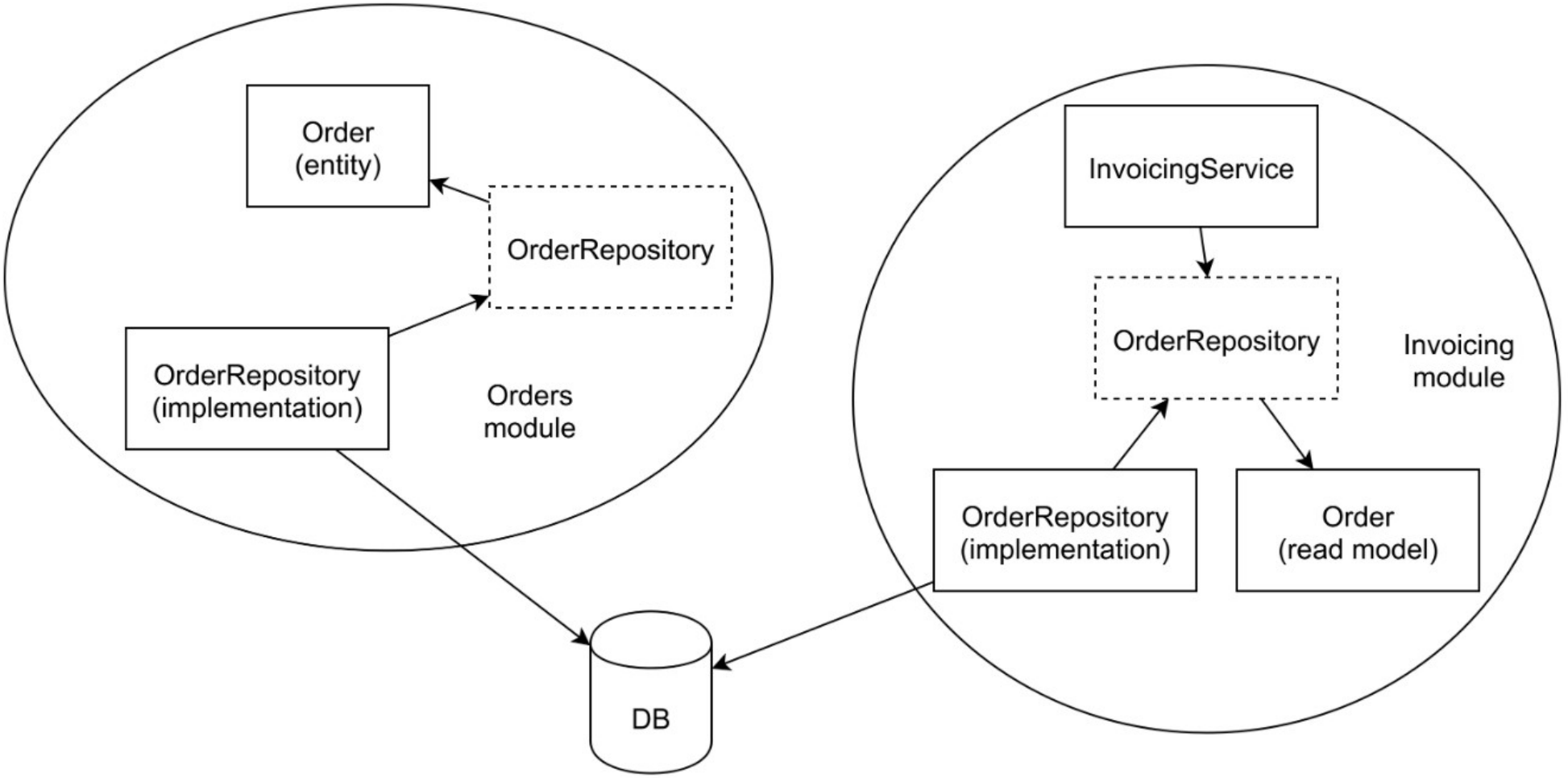


ORM

- An entity is used inside templates
- An entity has getters to expose relevant state, even related entities

How to improve?

- Use entities only for making changes (remove getters)
- Pass use-case specific read models to templates
- Populate read models with database data directly


Order
(entity)
OrderRepository
OrderRepository
(implementation)
Orders
module
InvoicingService
OrderRepository
Invoicing
module
OrderRepository
(implementation)
Order
(read model)
DB

An Aggregate is Owned by a Single Bounded Context

A domain?

- A sphere of activity, or knowledge
- E.g. bookkeeping, e-commerce, payments, etc.

Subdomains?

- Smaller domains within the larger domein
- Some generic, some specific
- *Examples?*

A context?

- Implements the model for a subdomain

A bounded context?

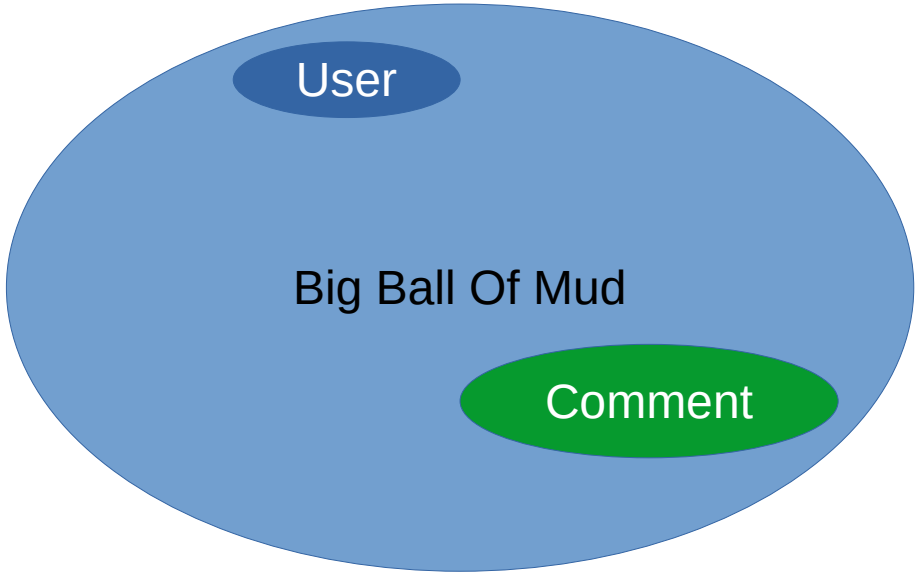
- Has an encapsulated model
- Offers an explicit API for accessing the model (changing, reading)

ORM

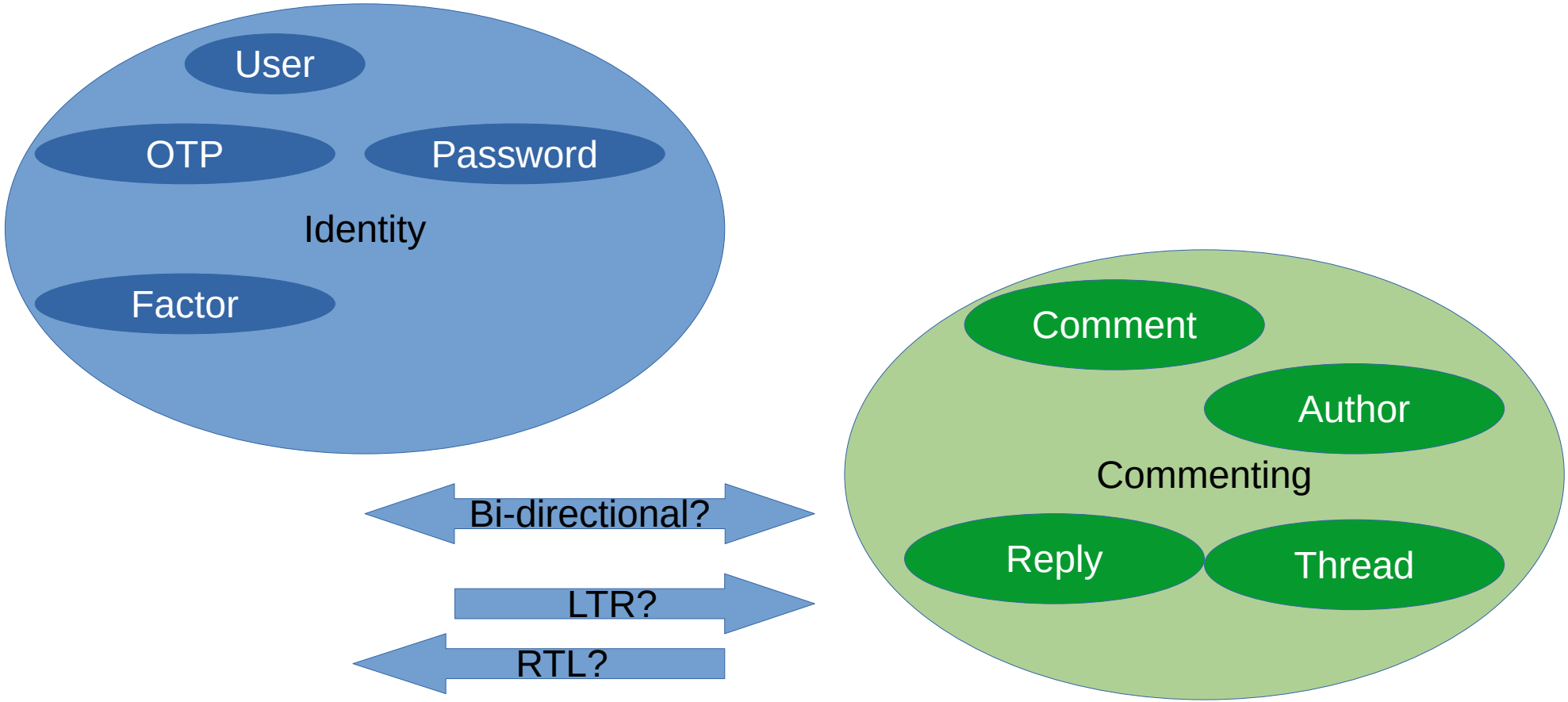
- Makes no distinction between contexts
- Change anything, anywhere
- **Big Ball of Mud**

```php
<?php
class User
{
    // ..

    /** @return Collection<int, Comment> */
    public function getAuthoredComments(): Collection {
        return $this->commentsAuthored;
    }

    /** @return Collection<int, Comment> */
    public function getFavoriteComments(): Collection {
        return $this->favorites;
    }
}

class Comment
{
    // ...
```

https://www.doctrine-project.org/projects/doctrine-orm/en/3.6/reference/working-with-associations.html#working-with-associations

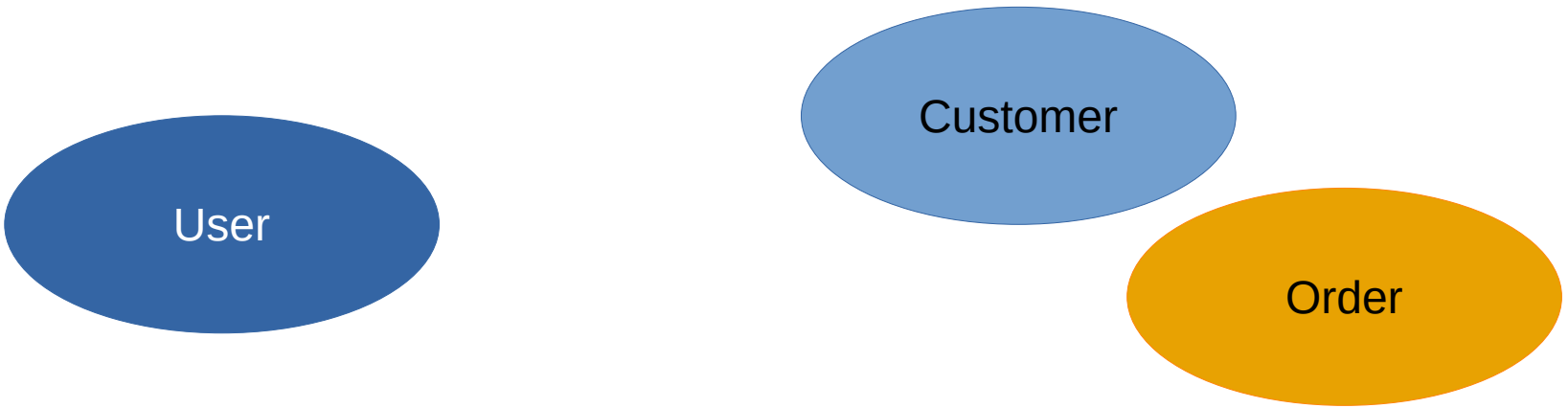

User
Big Ball Of Mud
Comment

The model evolves



Customer !== User

- They share the same ID
- They mean different things
- They carry different selections of data

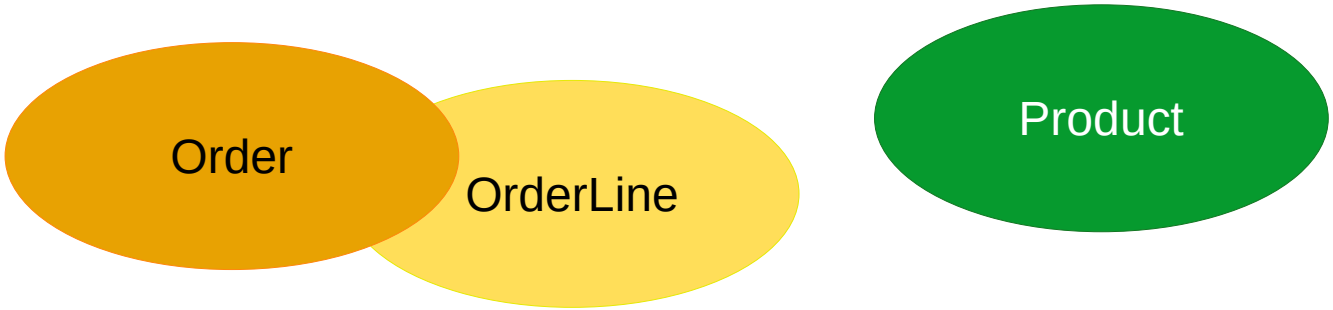


What if?

These contexts are separate processes?

We don’t need the full product

- What even is the “full” product?
- We may need just an ID, a price, etc.



How to improve?

- Pass **use-case specific read model objects** instead of full write model objects
- **Treat other contexts as “remote”**: talk through an interface

How to improve? - 2

- Design a programming interface for between-context communication:
  - What **read models** are accessible?
  - Which **domain events** can be subscribed to?

Hypothesis:

Your ORM entity ~= a DDD aggregate

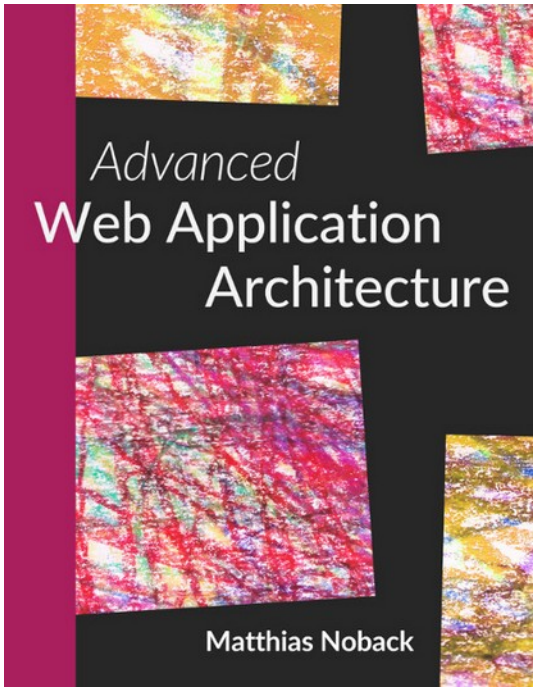
Summarizing

- Entities almost **never protect their own state**, offer intention-revealing methods, or act like a state machine
- No distinction between **Root and Child entities**
- Entities used as (collections of) **Value objects**
- An entity gets its **identity quite late**
- **Referencing not by ID** but by passing entire object
- **No boundaries** between contexts (everything can be related to everything)
- **Multiple changes** in one transaction (flush)
- Entities used as both **read and write models** at the same time
- ORM only provides **generic change events**

More?

Questions?

https://www.dddcommunity.org/library/vernon_2011/



Get in touch:

info@matthiasnoback.nl

https://matthiasnoback.nl/

LinkedIn
Bluesky
Mastodon