Applying Aggregates and Associations

Steve Smith

Force Multiplier for
Dev Teams

@ardalis | ardalis.com

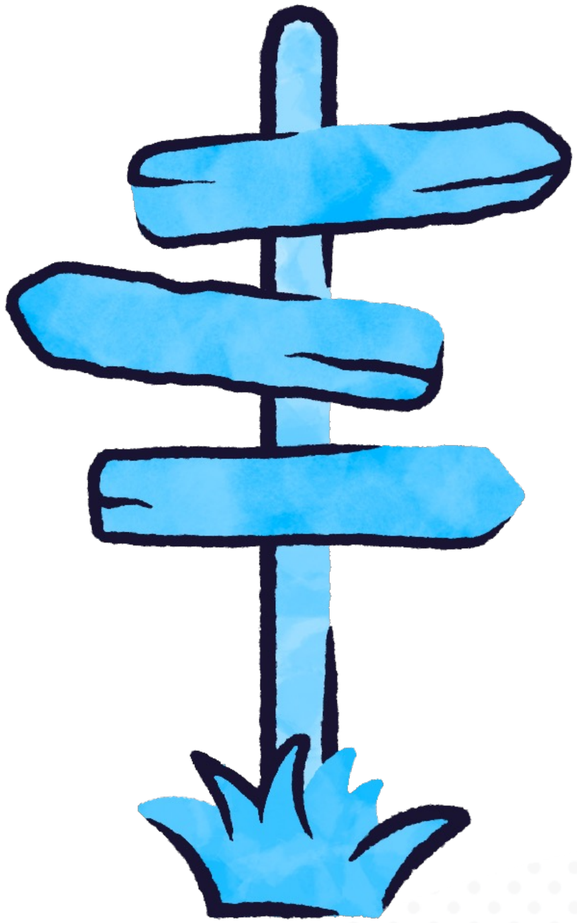
Julie Lerman

Software Coach,
DDD Champion

@julielerman | thedatafarm.com

DDD is for solving complex problems.

Module Overview

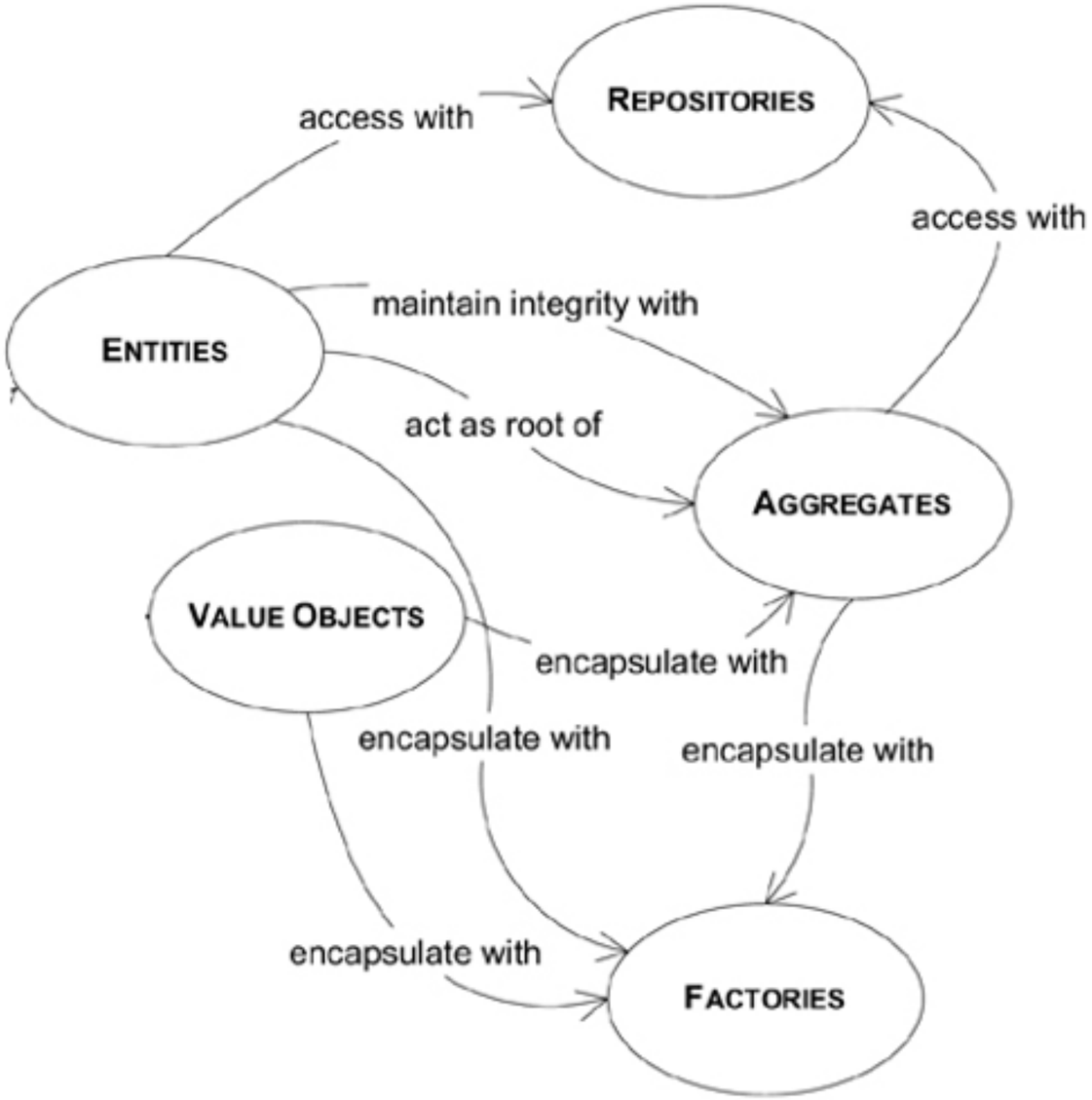


- Tackling data complexity
- What are aggregates?
- The importance of aggregate roots
- Invariants in aggregates
- Associations & relationships among entities
- Evolving the aggregate design in our sample solution
- Implementing aggregates in code

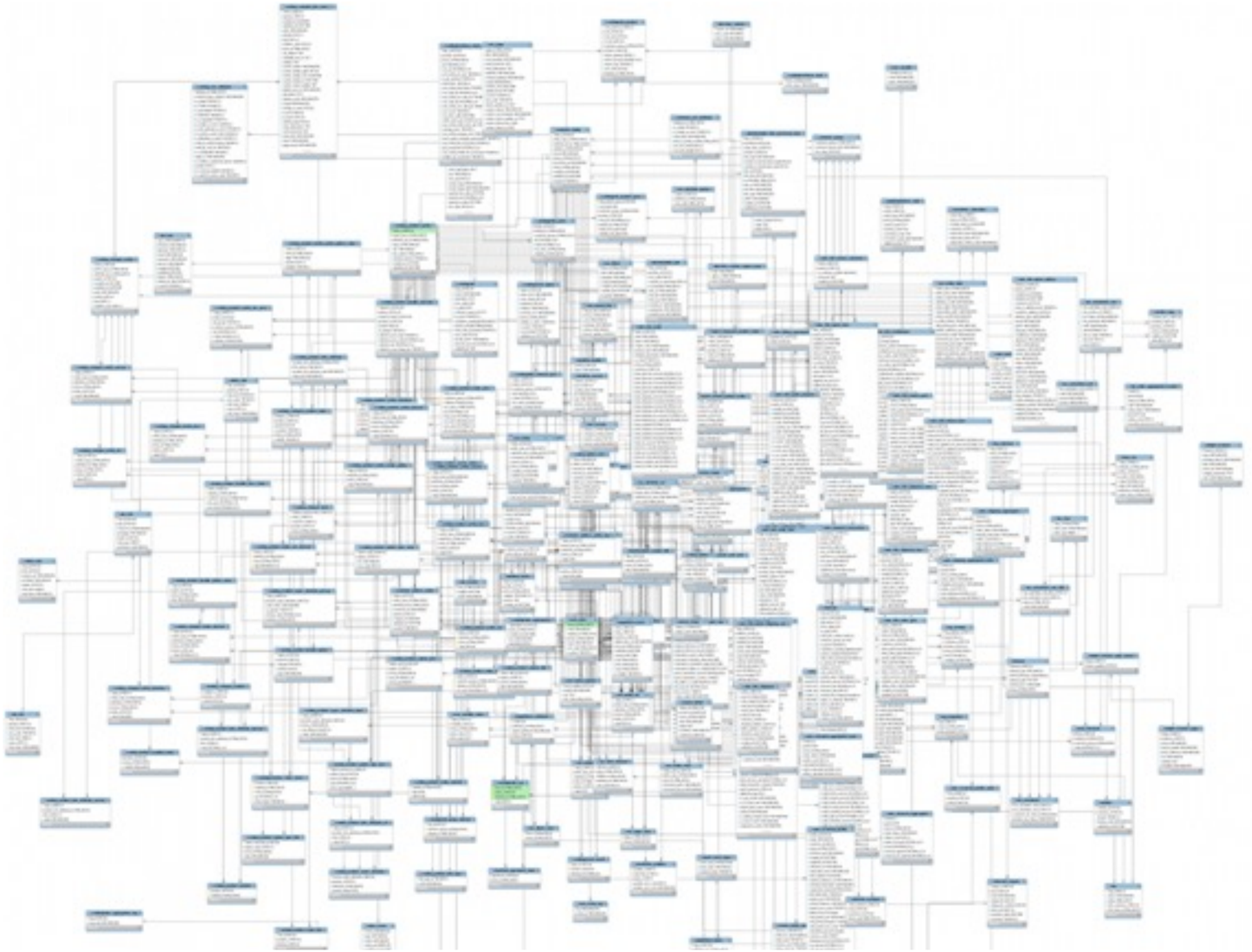
Tackling Data Complexity

Large systems often lead to complex data models.

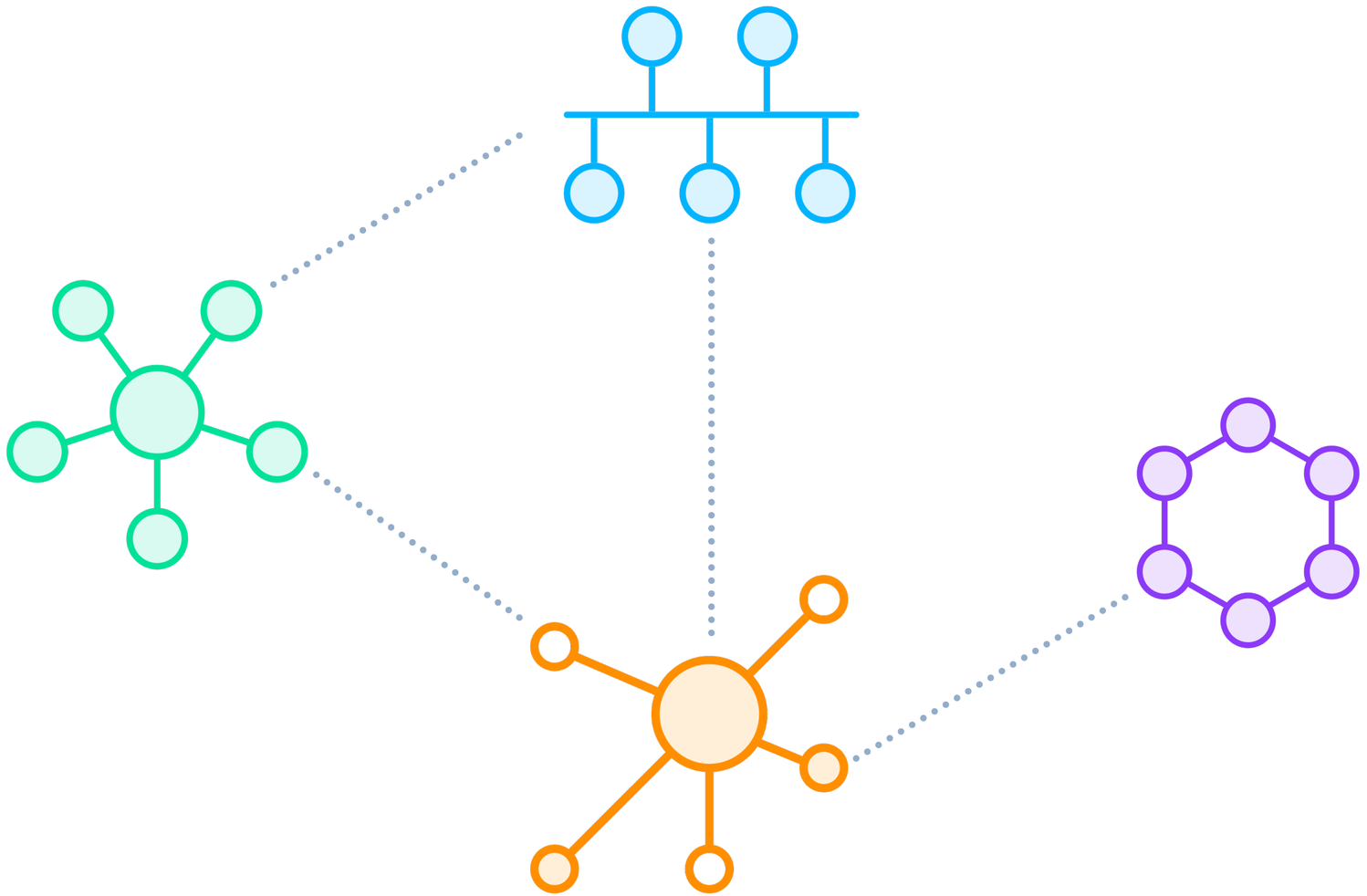
Aggregates and Aggregate Root in the DDD Mind Map



A Very Complex Model, Indeed



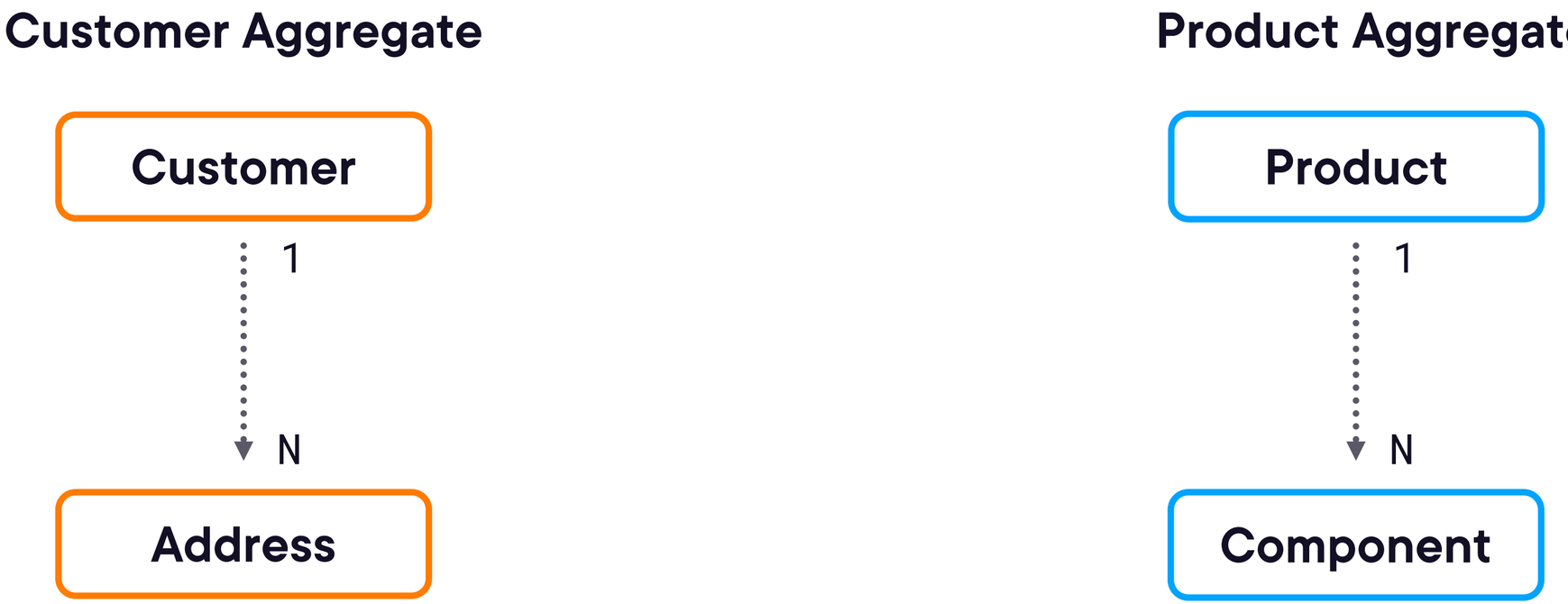
Smaller Models Can Still Interconnect



A big ball of mud!

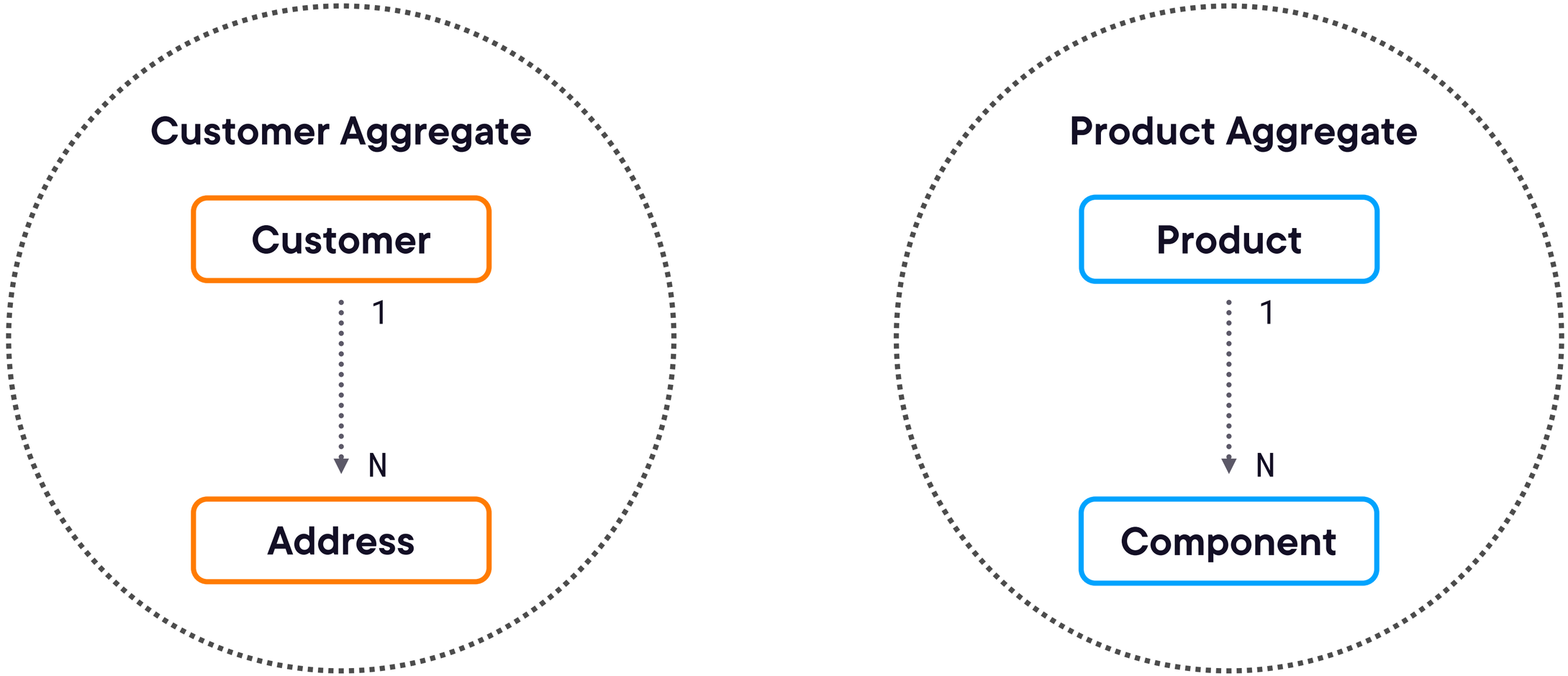
Introducing Aggregates and Aggregate Roots

Aggregates in Our Model

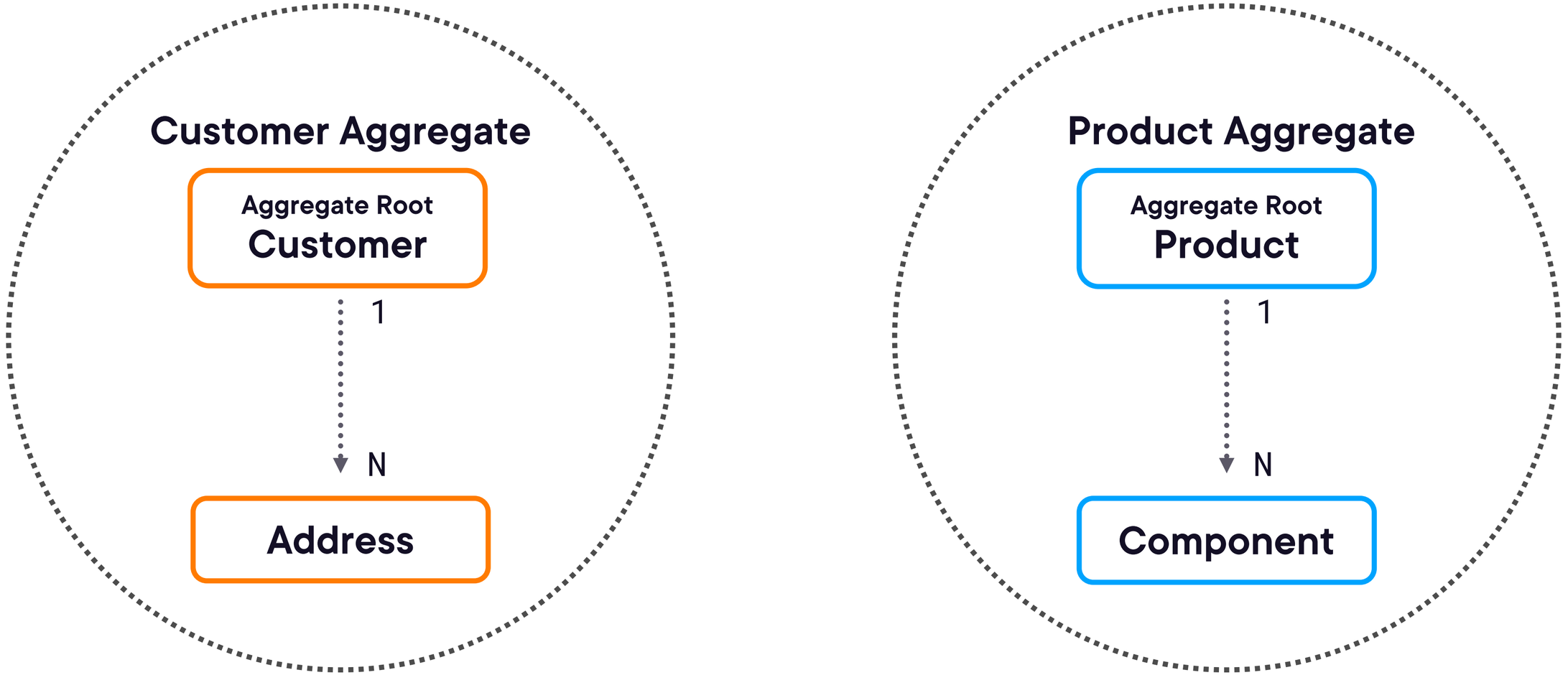


Aggregates in Our Model

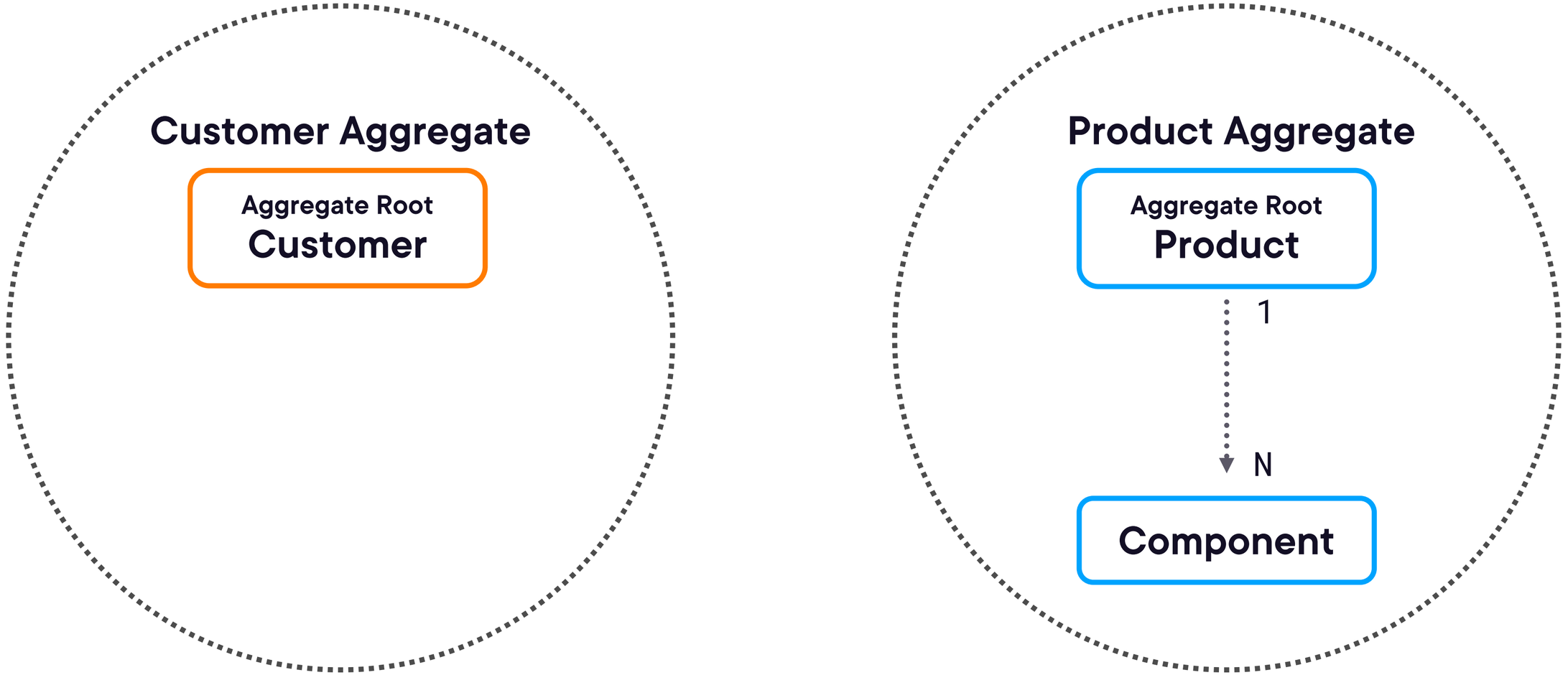
Aggregates in Our Model



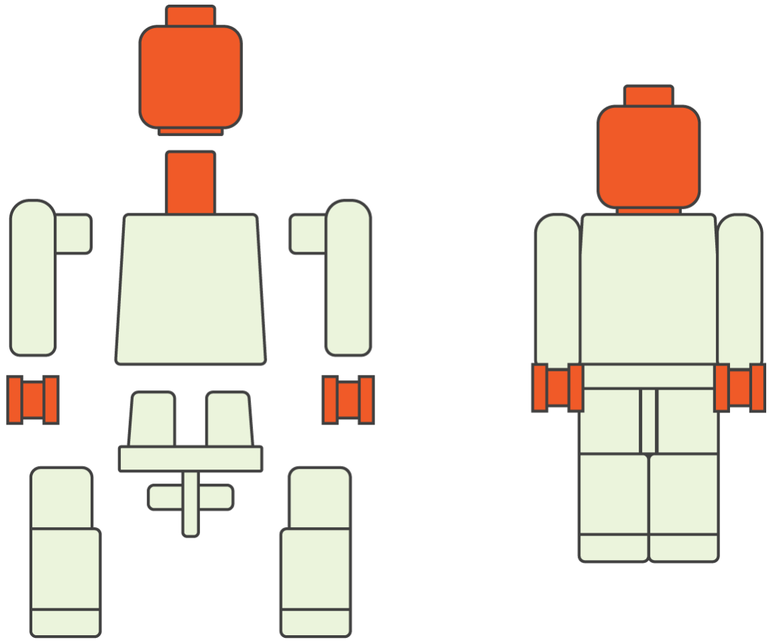
Aggregates in Our Model

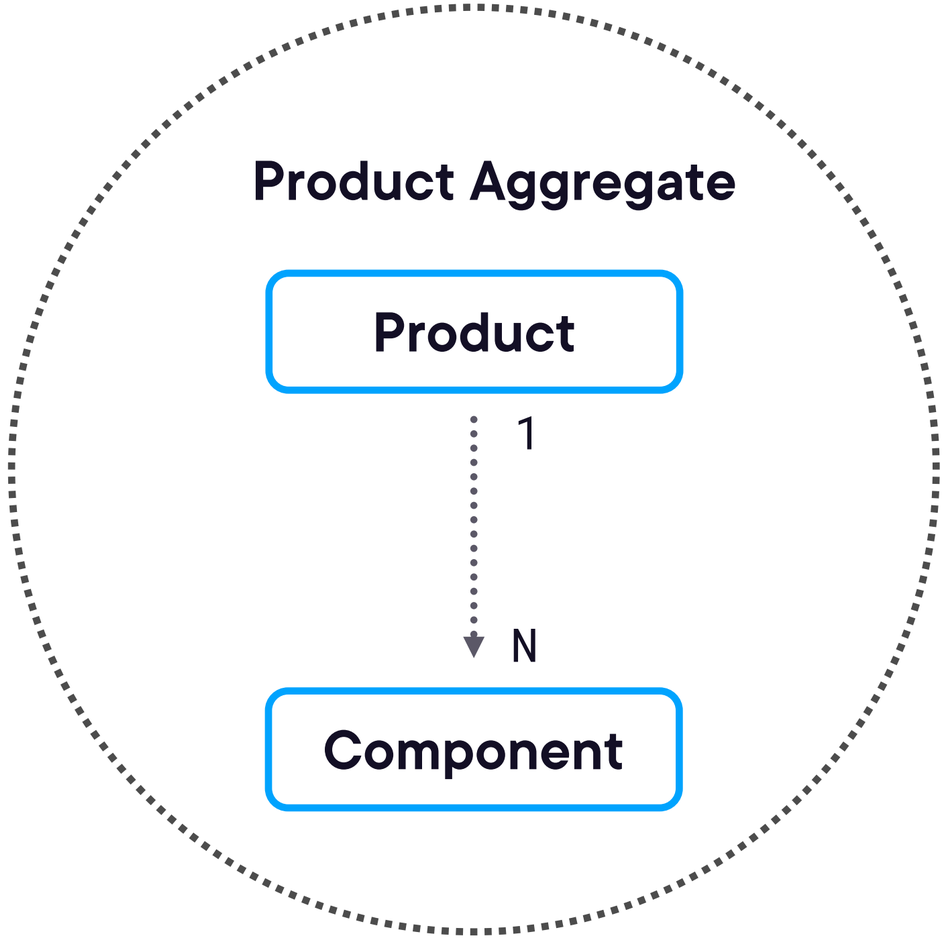


Aggregates in Our Model

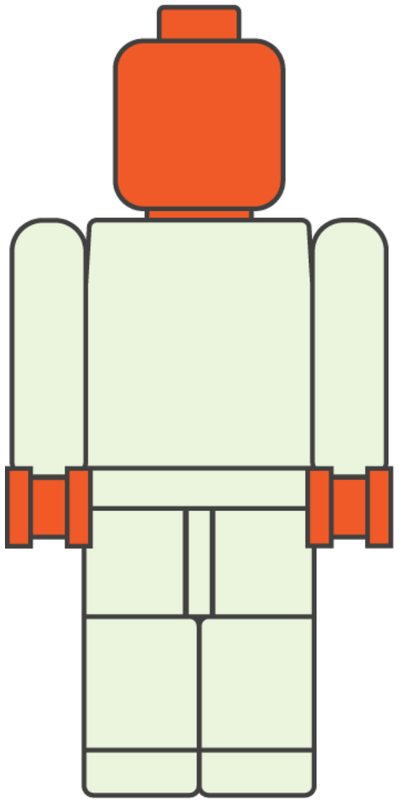


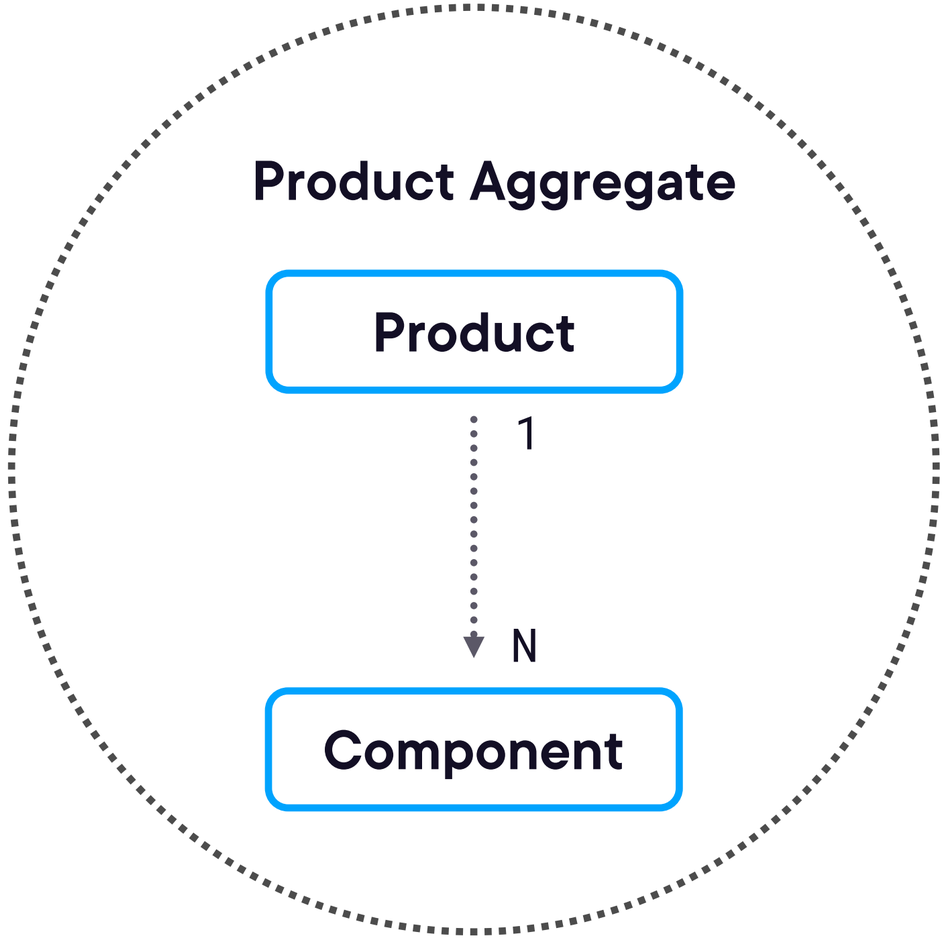
Aggregate Should Enforce Data Consistency





Aggregate Should Ensure a Product Is Valid





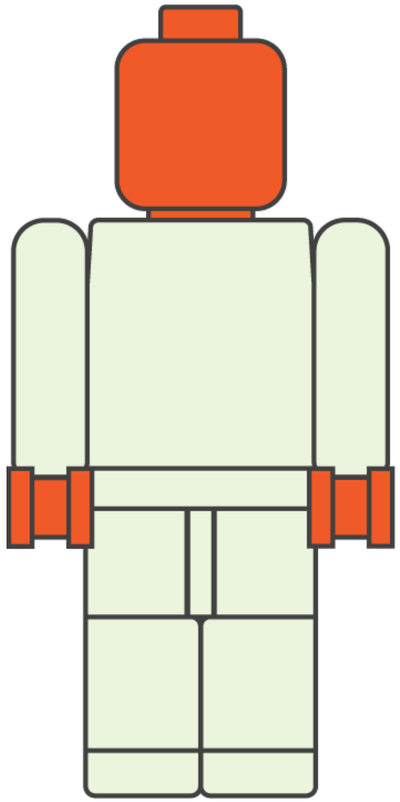
Data Changes Should Follow ACID

Atomic

Consistent

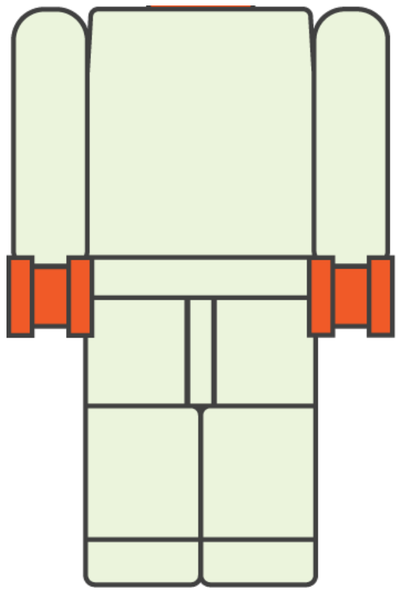
Isolated

Durable



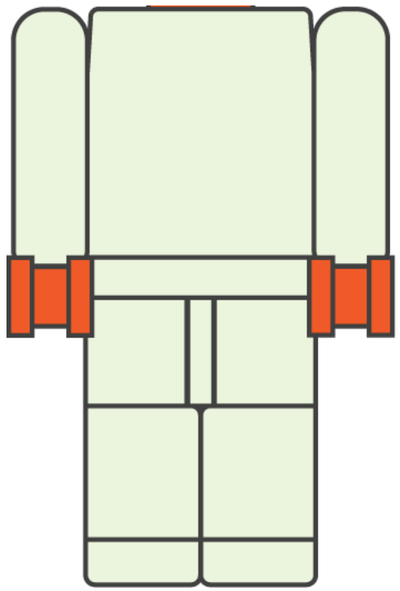
Aggregate Root Is Responsible for Maintaining Invariants

- ✓ 2 arms
- ✓ 2 legs
- ✓ 1 body
- ✓ 2 hands
- ✓ 1 pelvis
- ✓ 1 head



Aggregate Root is Responsible for Maintaining Invariants

- ✓ 2 arms
- ✓ 2 legs
- ✓ 1 body
- ✓ 2 hands
- ✓ 1 pelvis
- ✓ 1 head

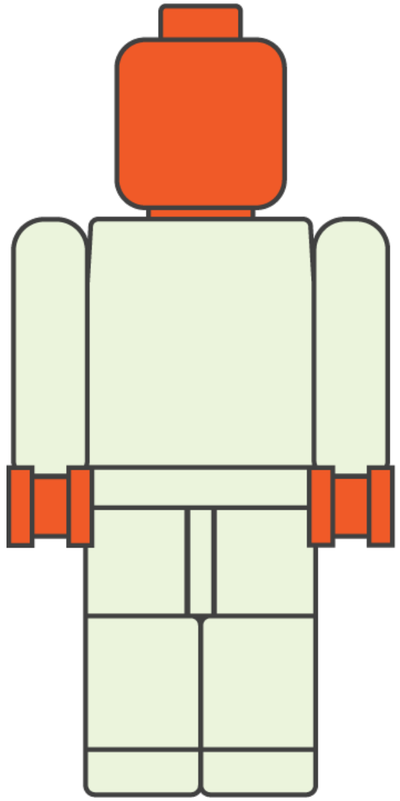


Aggregate Root is Responsible for Maintaining Invariants

- ✓ 2 arms
- ✓ 2 legs
- ✓ 1 body
- ✓ 2 hands
- ✓ 1 pelvis
- ✓ 1 head

Deleting Is a Good Test to Discover Root

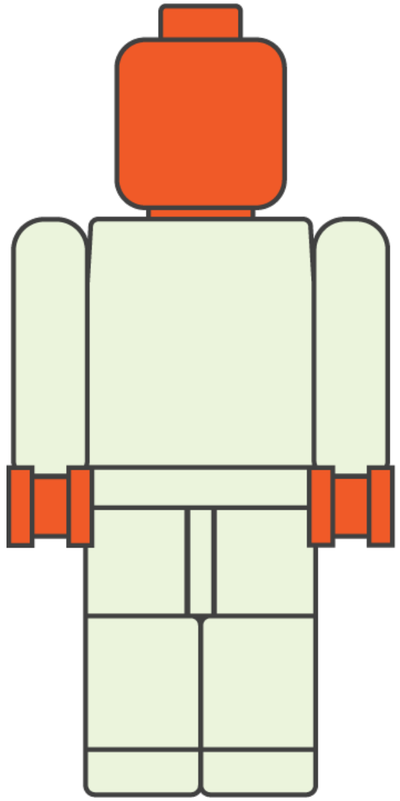
Deleting the Root Should Delete the Entire Aggregate



DELETE

- ✓ 2 arms
- ✓ 2 legs
- ✓ 1 body
- ✓ 2 hands
- ✓ 1 pelvis
- ✓ 1 head

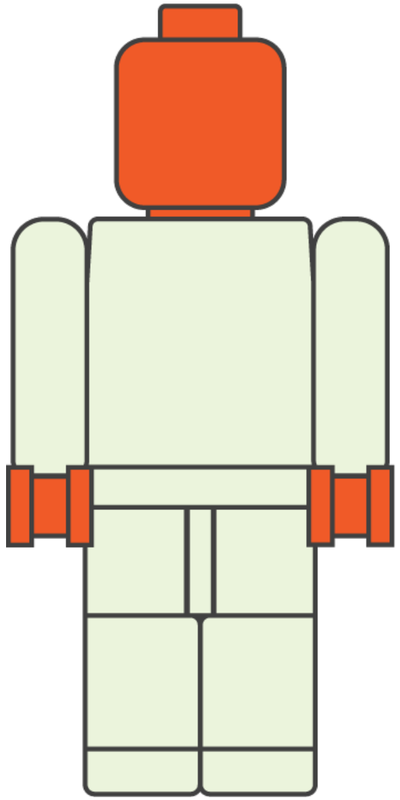
Deleting the Root Should Require Deleting the All the Objects in the Aggregate



DELETE

- ✓ 2 arms
- ✓ 2 legs
- ✓ 1 body
- ✓ 2 hands
- ✓ 1 pelvis
- ✓ 1 head

How Are An Aggregate's Objects Affected?



Delete a MiniFig

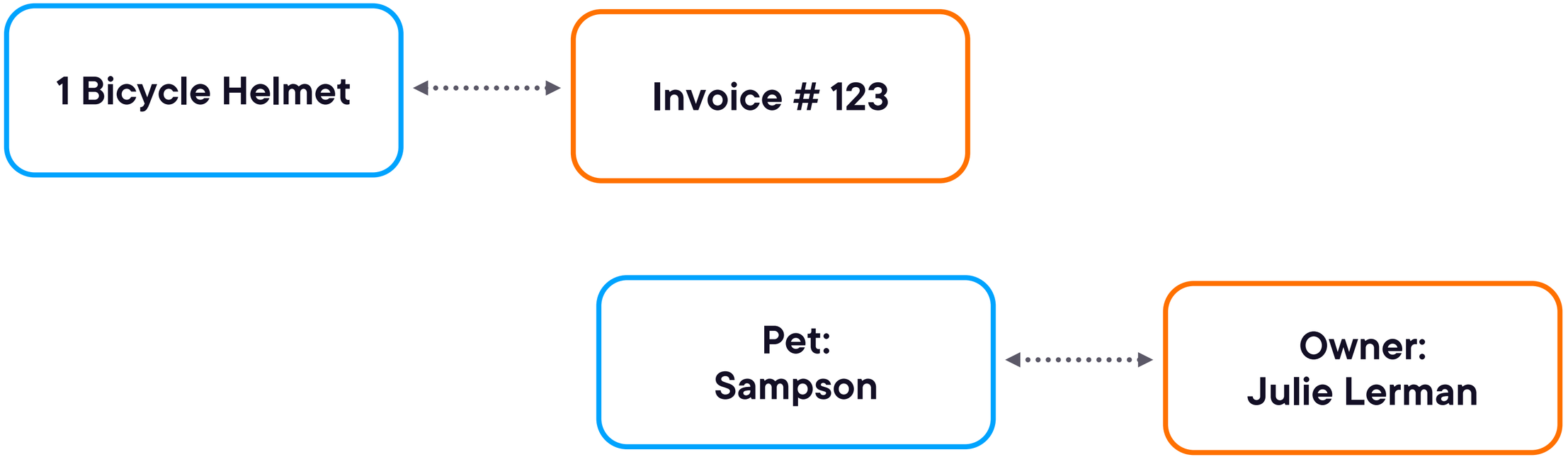
Delete a head component

Eric Evans

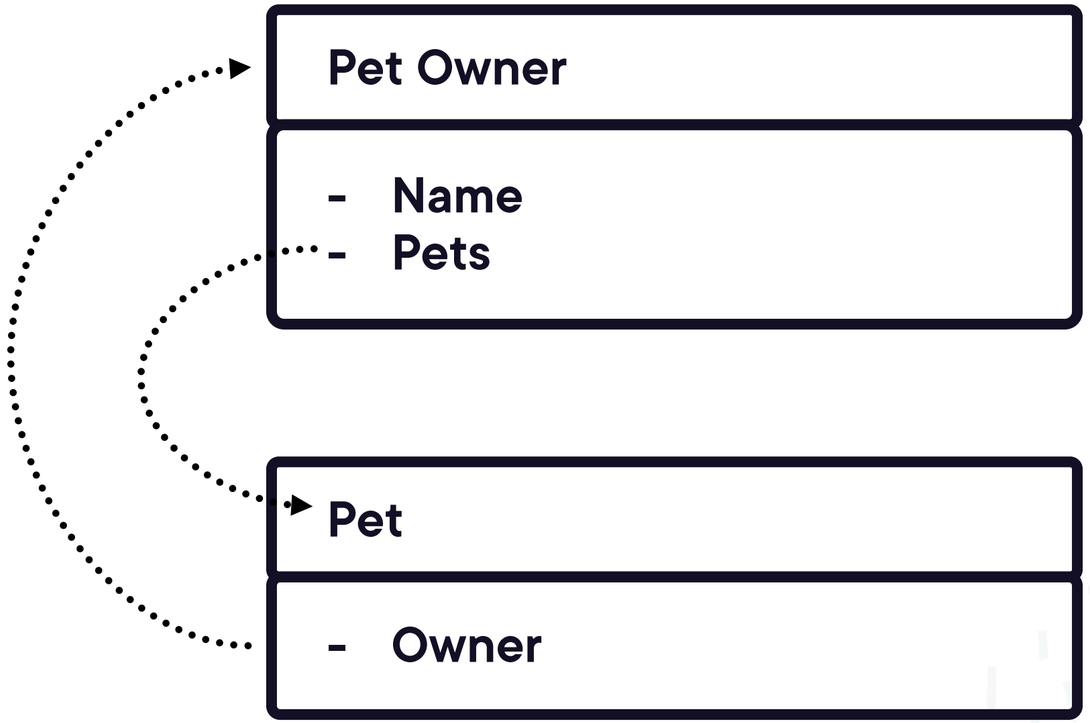
An aggregate is a cluster of associated objects that we treat as a unit for the purpose of data changes.

Considering Associations in Aggregates

Bi-directional Relationships

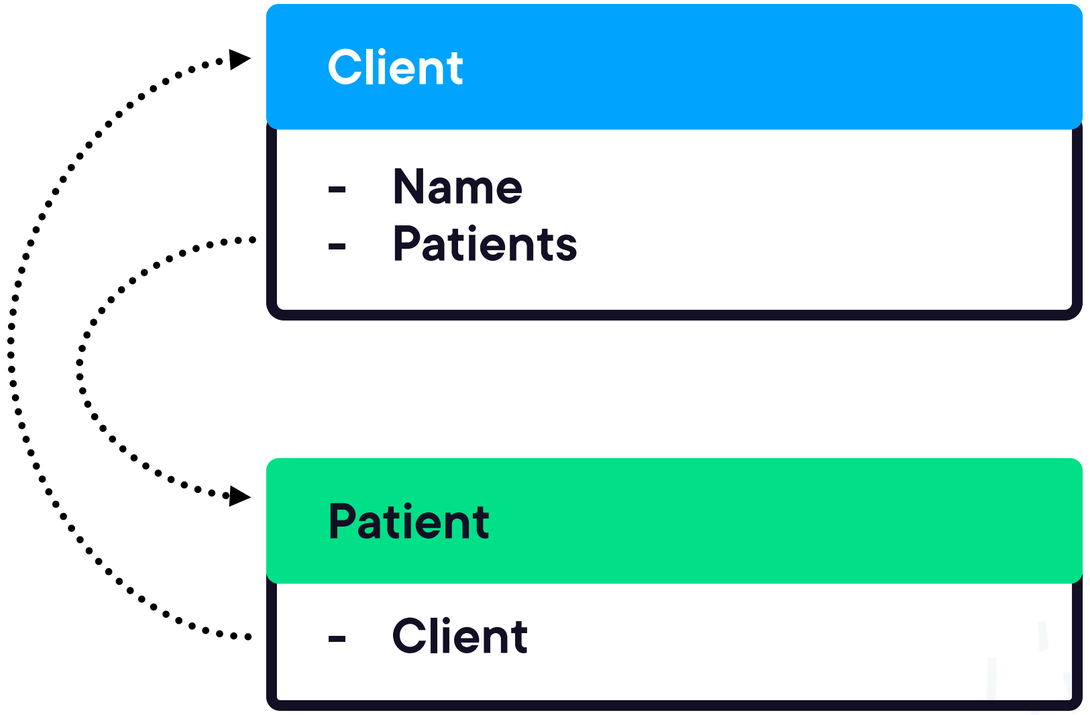


Bi-directional Relationship Using Properties



Default to one-way associations.

Bi-directional Relationship Using Properties



Eric Evans

A bidirectional association means that both objects can be understood only together. When application requirements do not call for traversal in both directions, adding a traversal direction reduces interdependence and simplifies the design.

Do I Need a Bi-directional Relationship Here?

Which Single Direction Makes Sense?

Schedule an appointment?

Pay a bill?

Which Single Direction Makes Sense?

Schedule an appointment?

Pay a bill?

The One-way Client:Patient Relationship in the Scheduling Bounded Context

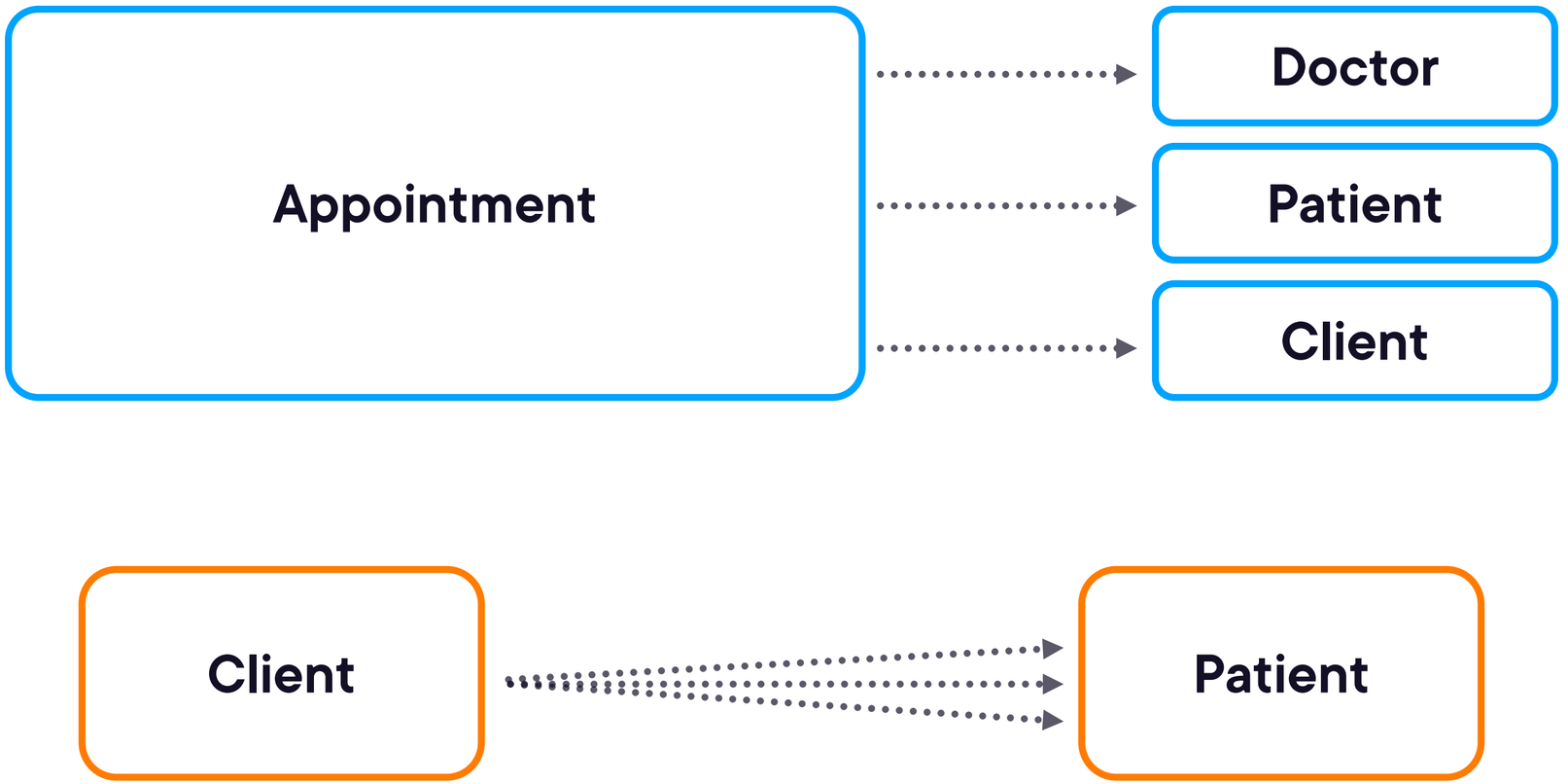
Client :Entity<Int>

- **FullName**
- **Patients**

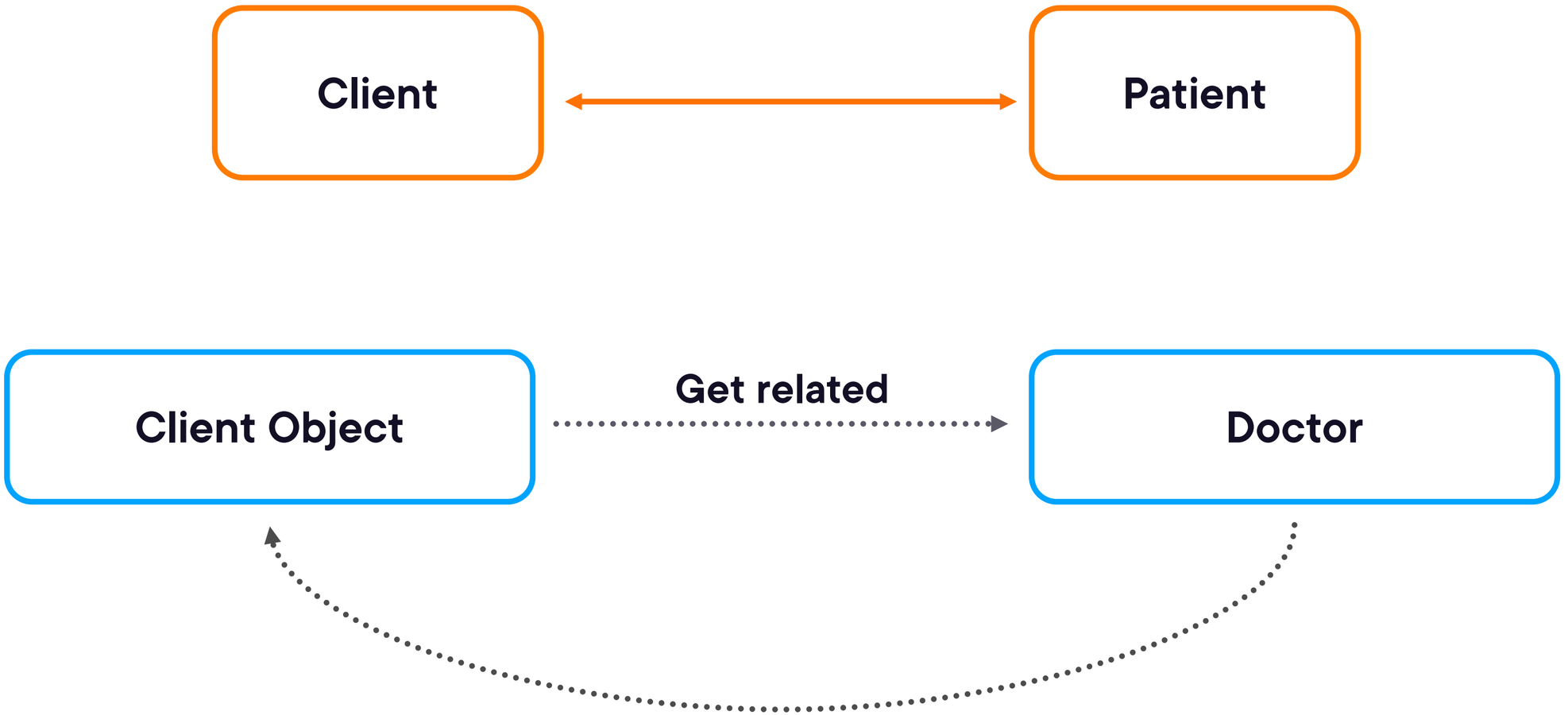
Patient :Entity<Int>

- **AnimalType**
- **ClientId**
- **Gender**
- **Name**
- **PreferredDoctor**

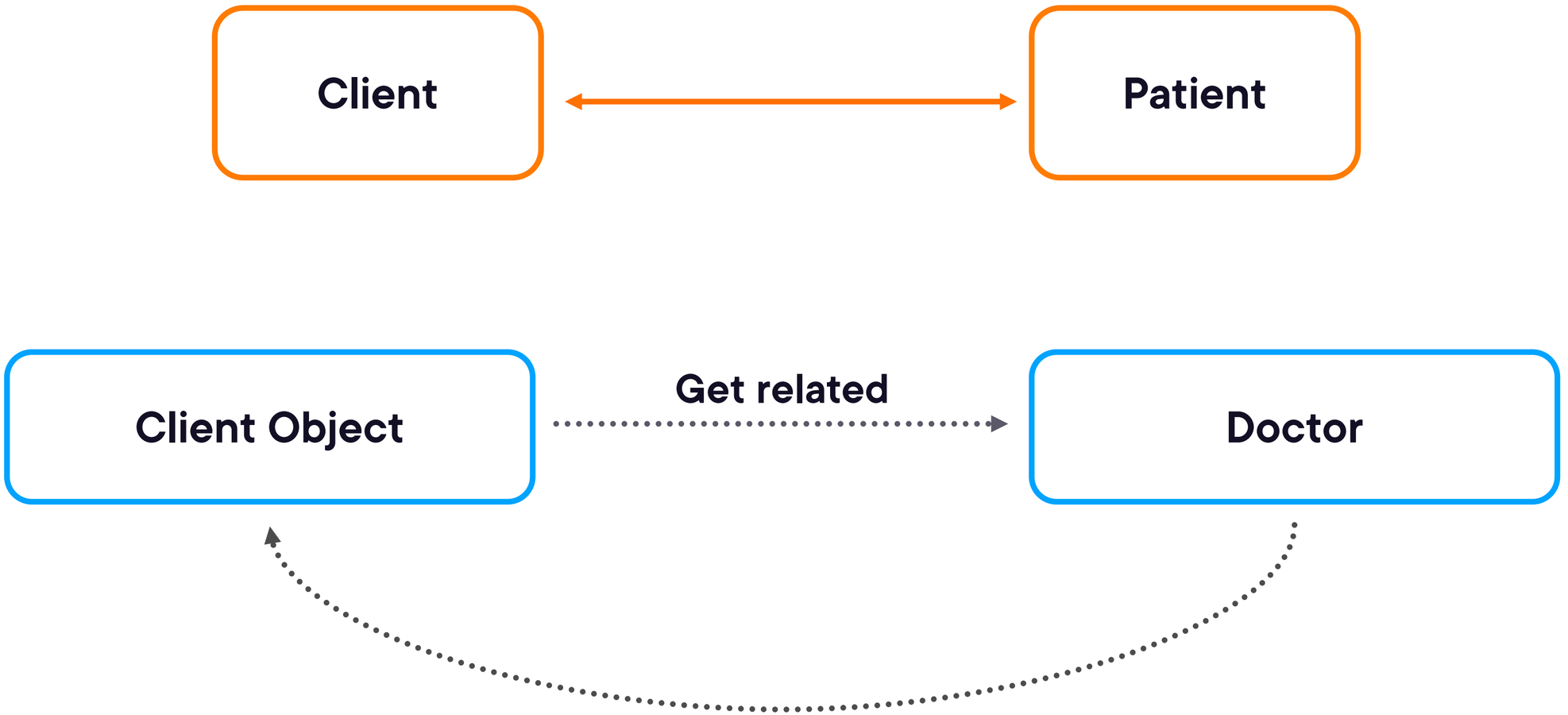
Uni-directional Associations



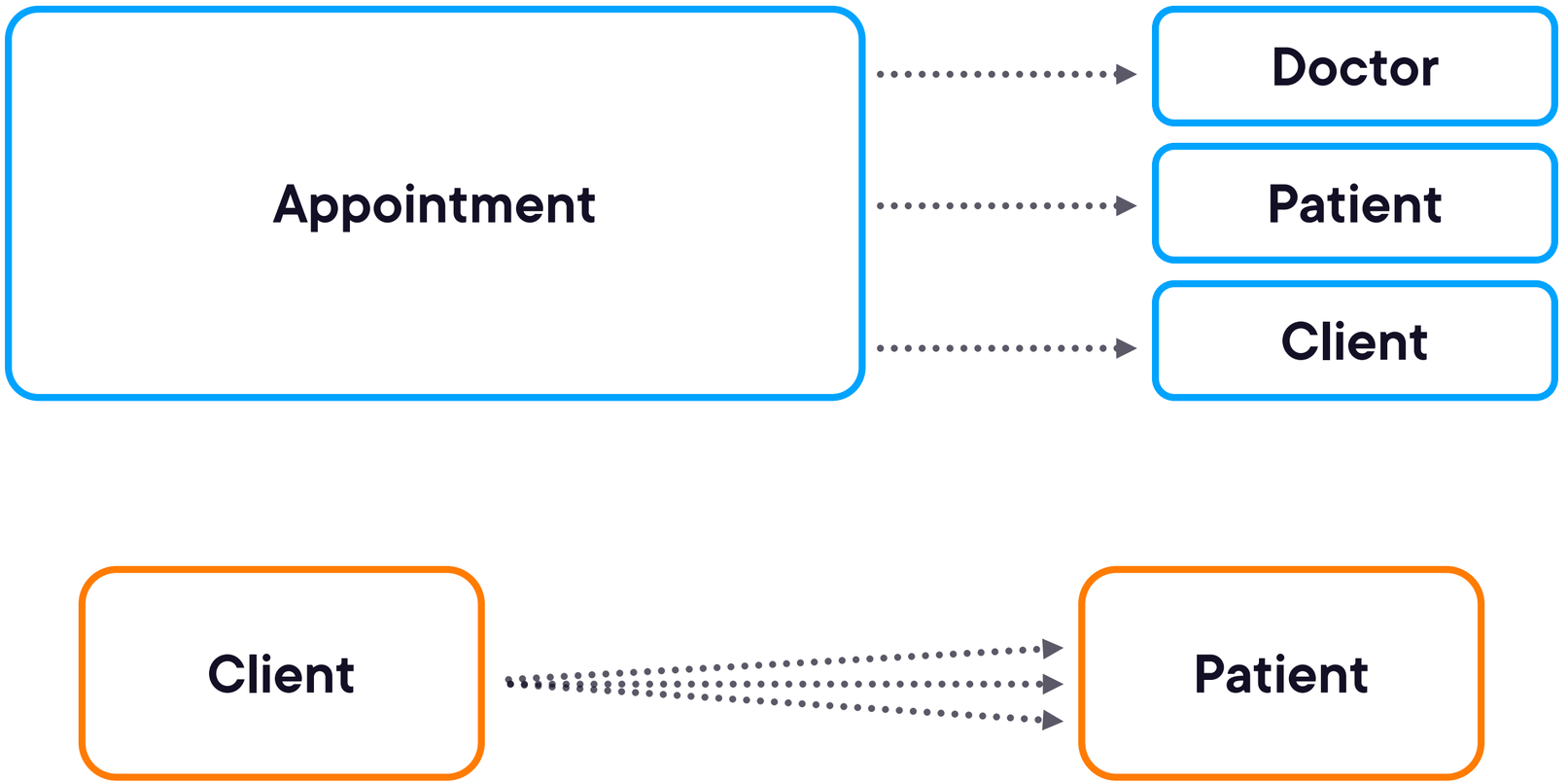
Uni-directional Associations



Uni-directional Associations



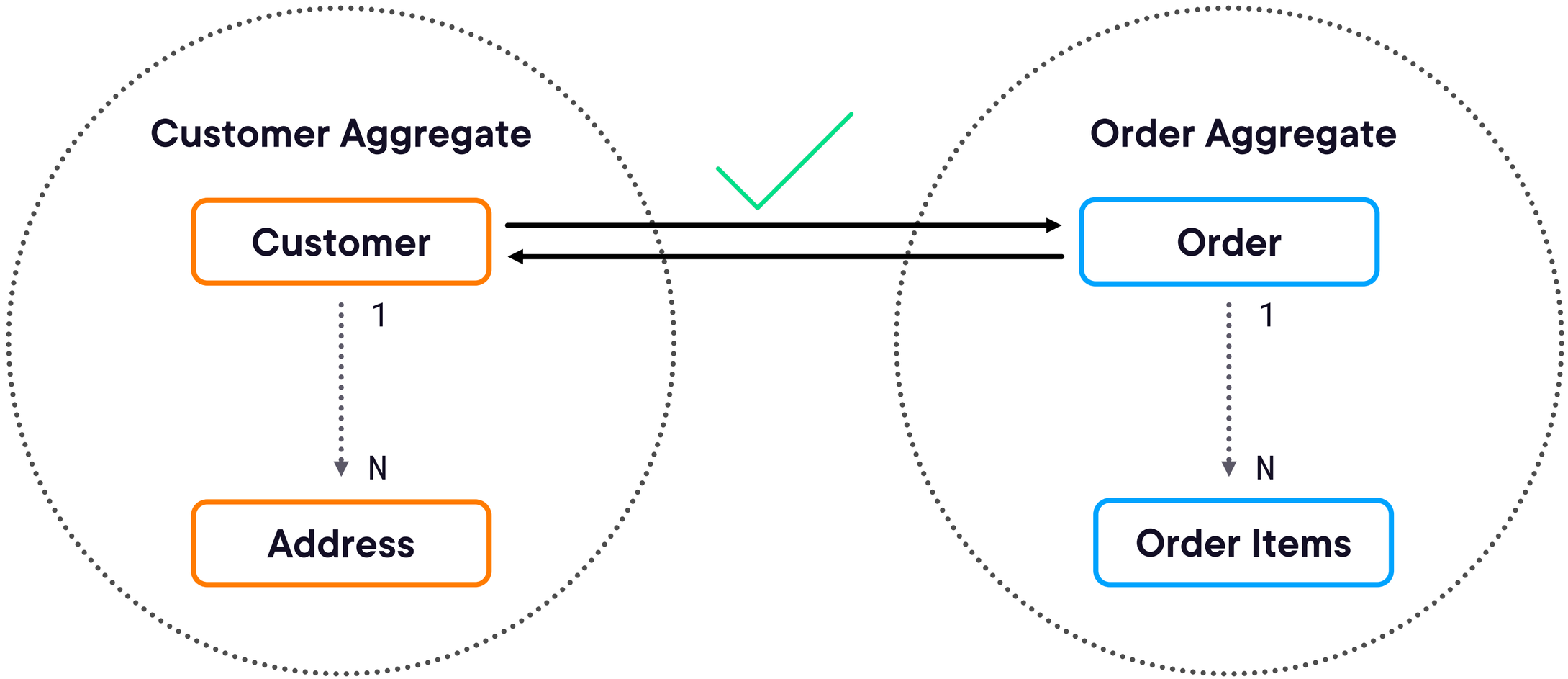
Uni-directional Associations



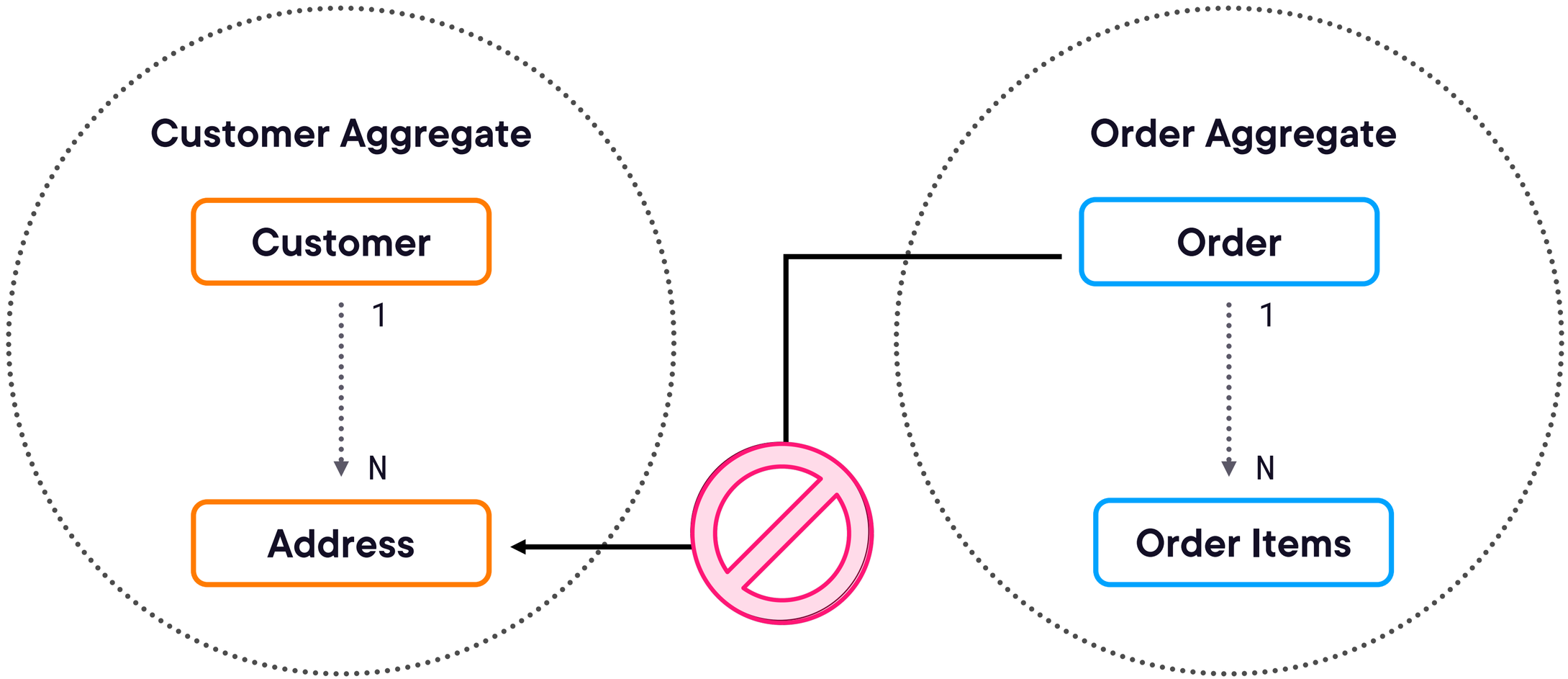
Handling Relationships that Span Aggregates

External objects should interact with only the aggregate root.

Enforcing Boundaries Between Aggregates

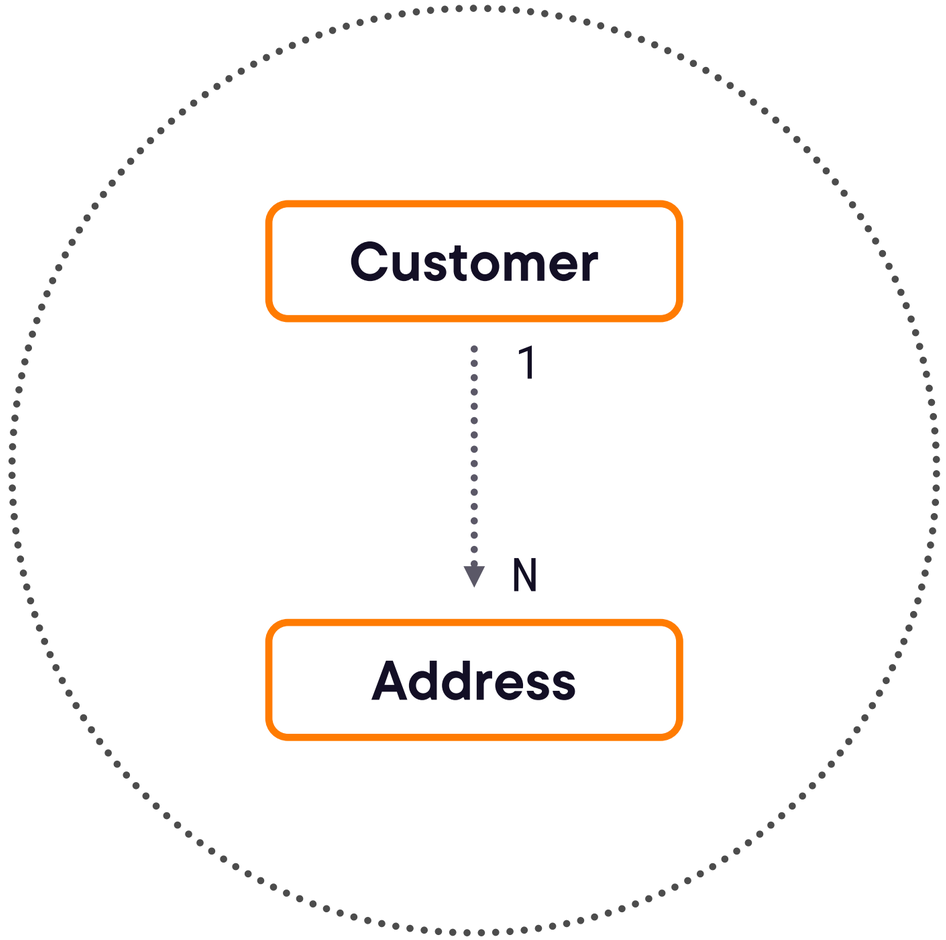


Enforcing Boundaries Between Aggregates

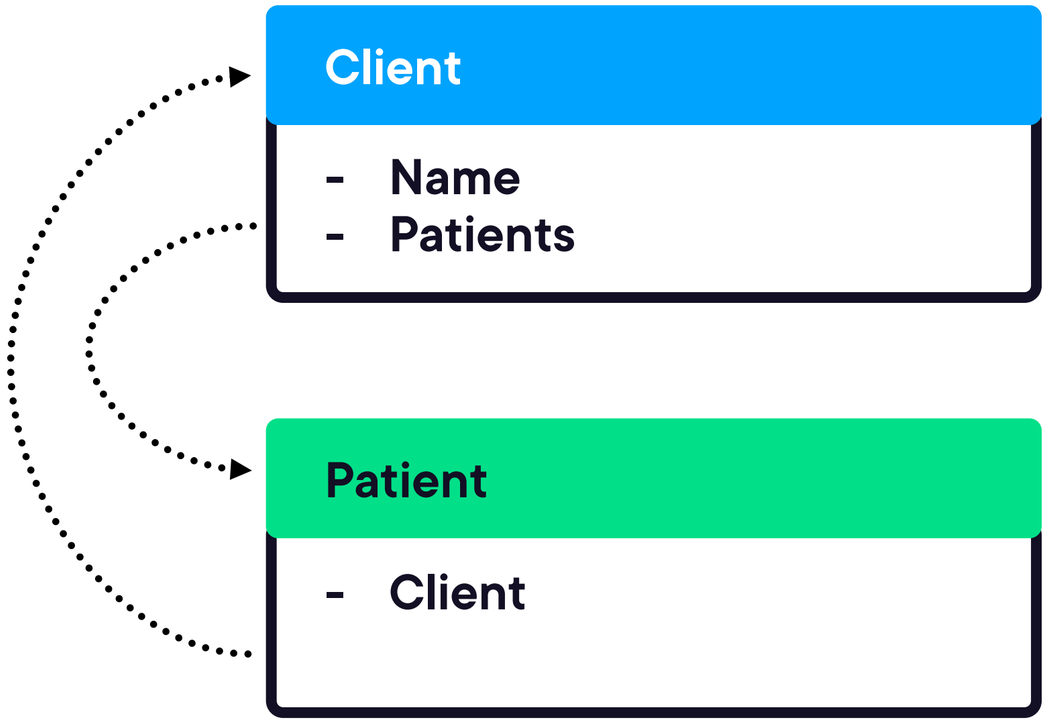


Object relationships are not the same as relationships between persisted data.

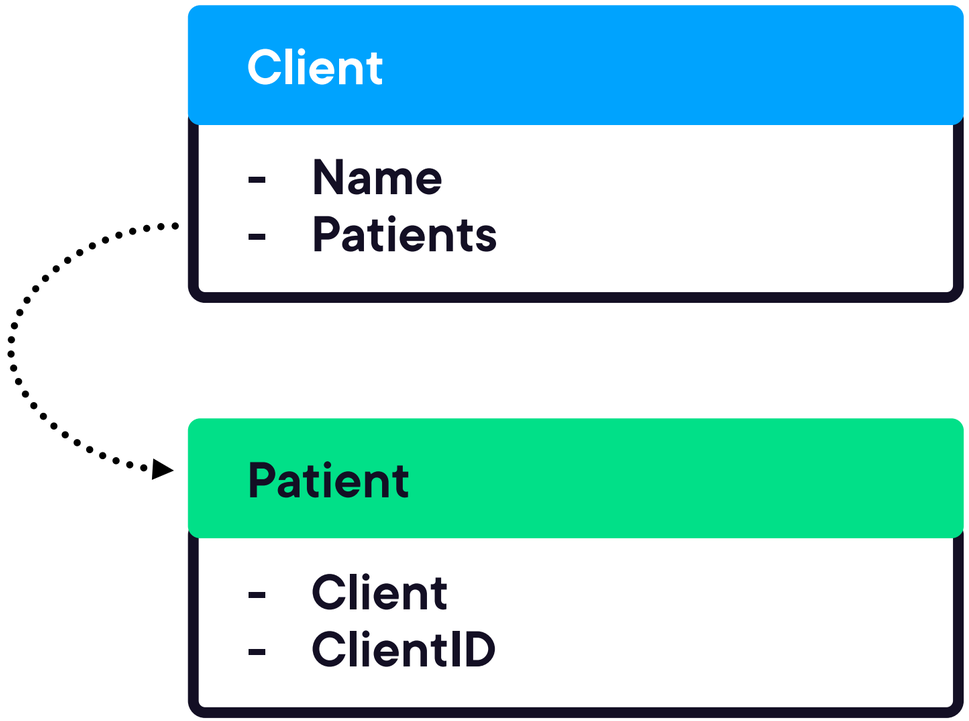
Another Downside of Bi-directional Navigation



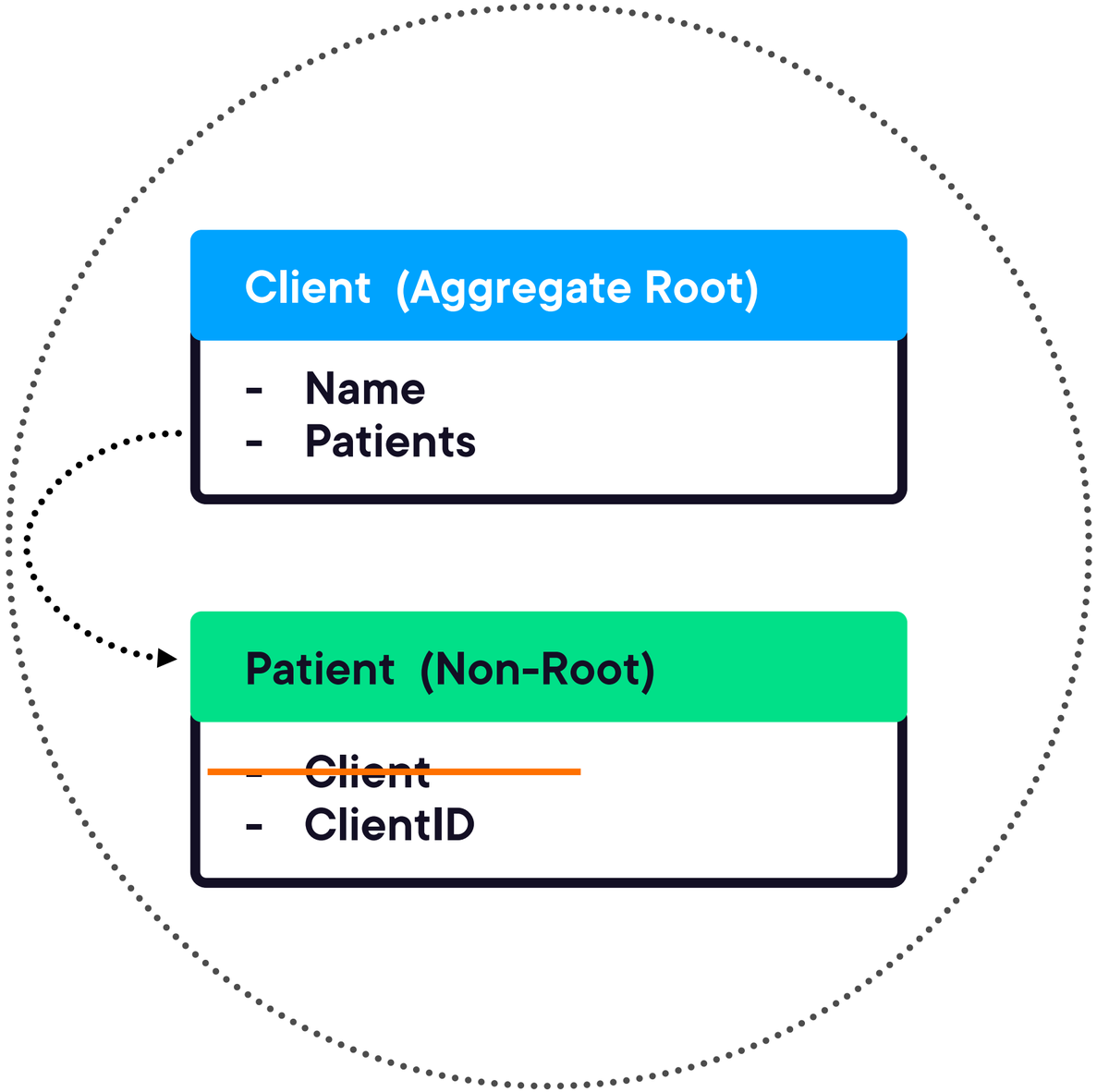
Reference ID Value Can Save ORM Confusion



Reference ID Value Can Save ORM Confusion

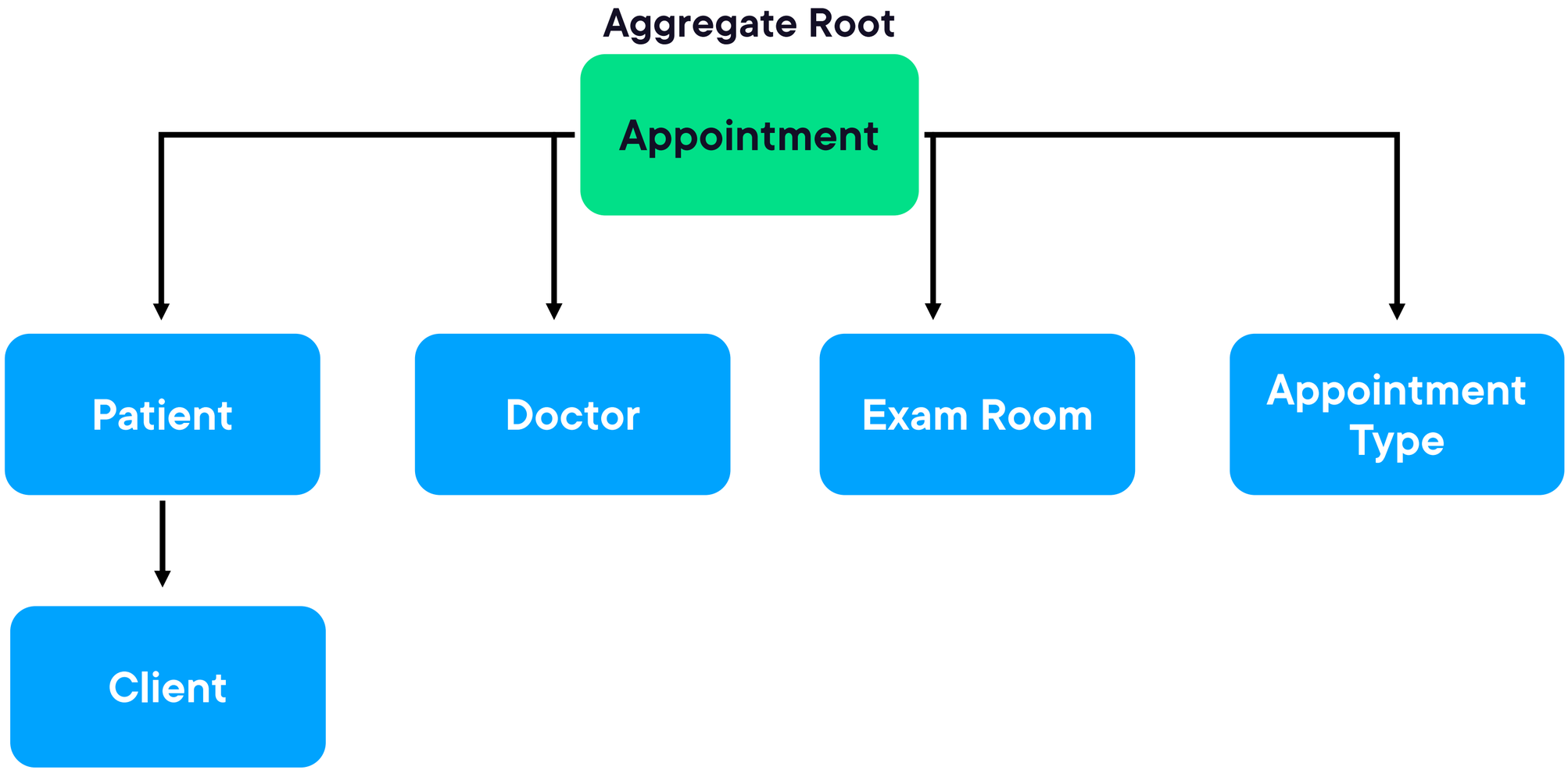


Reference ID Value Can Save ORM Confusion

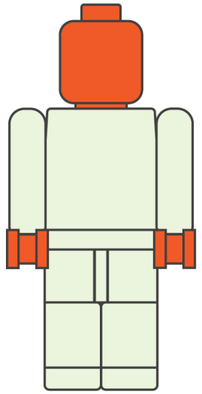


Evolving the Appointments Aggregate

Our First Draft Design: Appointment Aggregate



Recall a Test to Determine an Aggregate's Root



When you delete an aggregate's root, the other objects in the aggregate should get deleted

When you delete an appointment, should you also delete the patient, doctor, exam room, appointment type and client?

Our Revised Design

Appointment :Entity<Guid>

- AppointmentTypeId
- ClientId
- DoctorId
- DateTimeConfirmed
- PatientId
- RoomId
- IsPotentiallyConflicting
- DateTimeRange

Using Invariants to Better Understand our Aggregate

Our Revised Design

Appointment :Entity<Guid>

- AppointmentTypeId
- ClientId
- DoctorId
- DateTimeConfirmed
- PatientId
- RoomId
- IsPotentiallyConflicting
- DateTimeRange

Maintained in other bounded contexts

Client | Exam Room | Patient | Doctor | Appointment Type

Aggregate root is responsible for maintaining the rules of the aggregate.

But what is an invariant?

Speed of Light Is an Invariant

1,079,252,849 km/h

670,616,629 mph

Invariant

A condition that should always be true for the system to be in a consistent state.

Examples of Aggregate Invariants

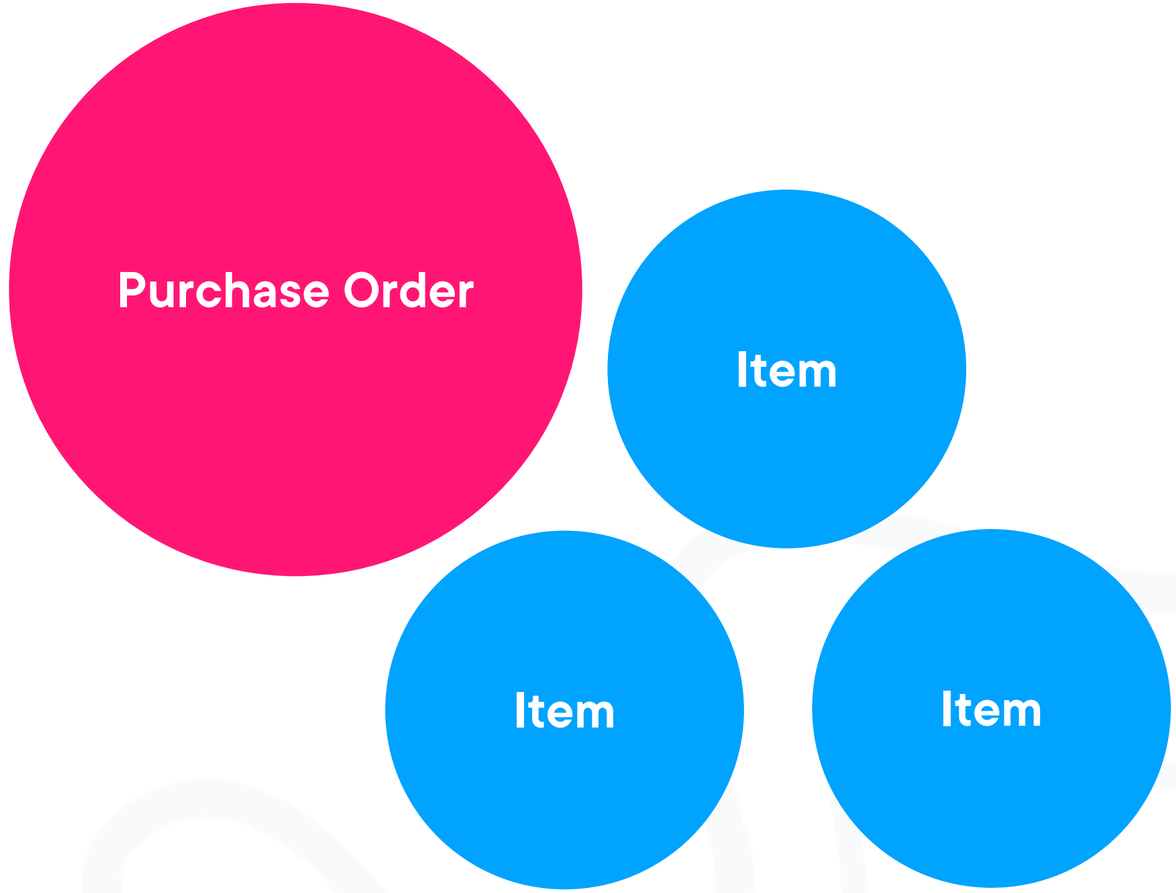
Total items on purchase order do not exceed limit

Two appointments do not overlap one another

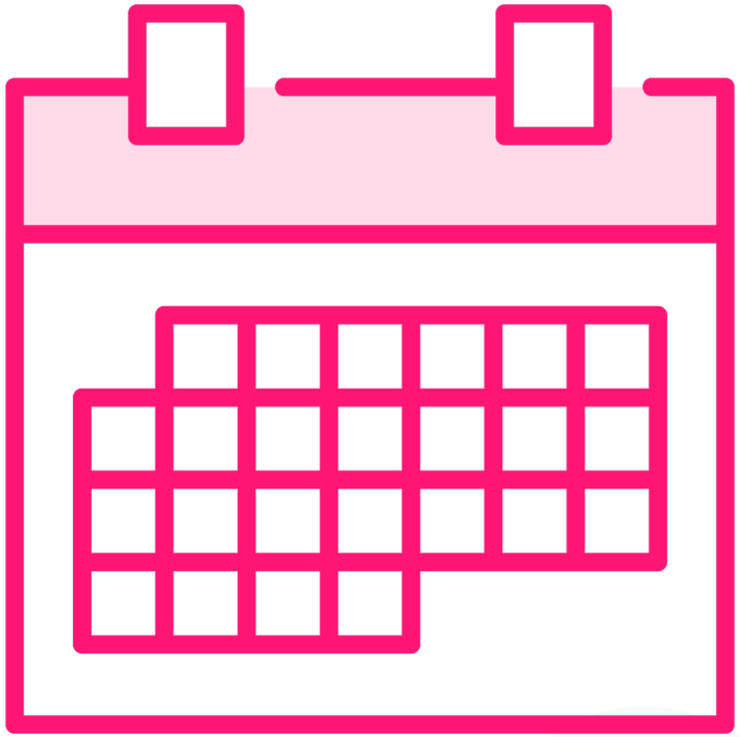
End date follows begin date

Purchase Order (Root) Maintains This Invariant

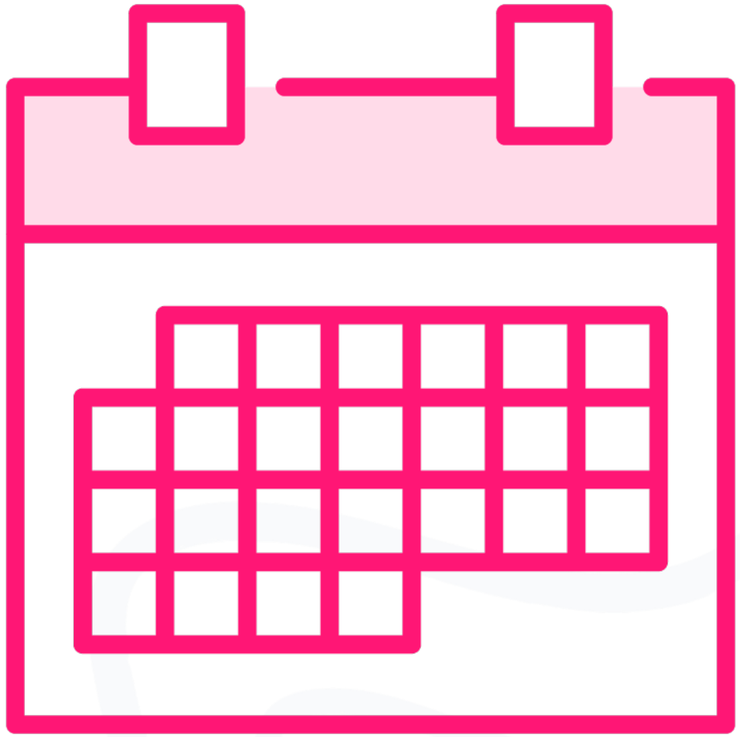
Total items on purchase order do not exceed limit



Examples of Aggregate Invariants

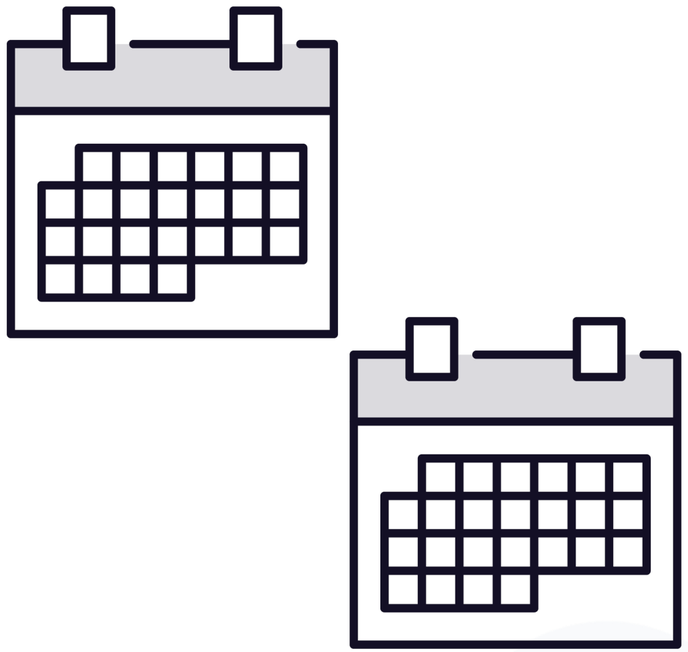


Two appointments
do not overlap
one another



Modeling Breakthroughs and Refactoring

Examining Invariants Leads to Discovery



Invariant:
Two appointments do not overlap one another

Aggregate Advice:
Appointments should not know about each other

Is appointment really a good aggregate root?

Citation: flickr.com/photos/wocintechchat/22543230821/

It’s normal and expected for models to evolve as you learn more about the domain.

Eric Evans

“Each refinement of code and model gives developers a clearer view. This clarity often creates the potential for a breakthrough of insights.”

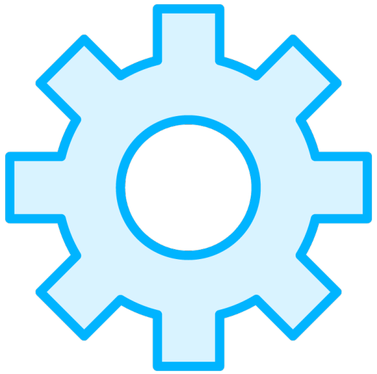
Citation: Domain-Driven Design, Chapter 8

Recognizing Signs of a Misidentified Aggregate

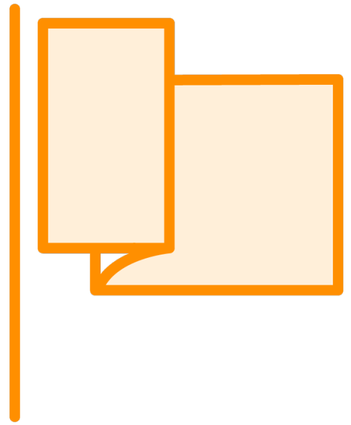
As our understanding of the domain evolved, schedule appeared to be a more appropriate aggregate root.

Cross-aggregate invariants should not be enforced by any one aggregate.

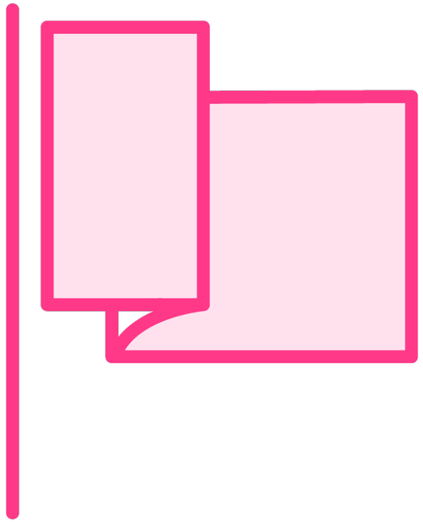
Cross-aggregate Invariants



Domain Services are one solution



Possible red flag about your domain model design



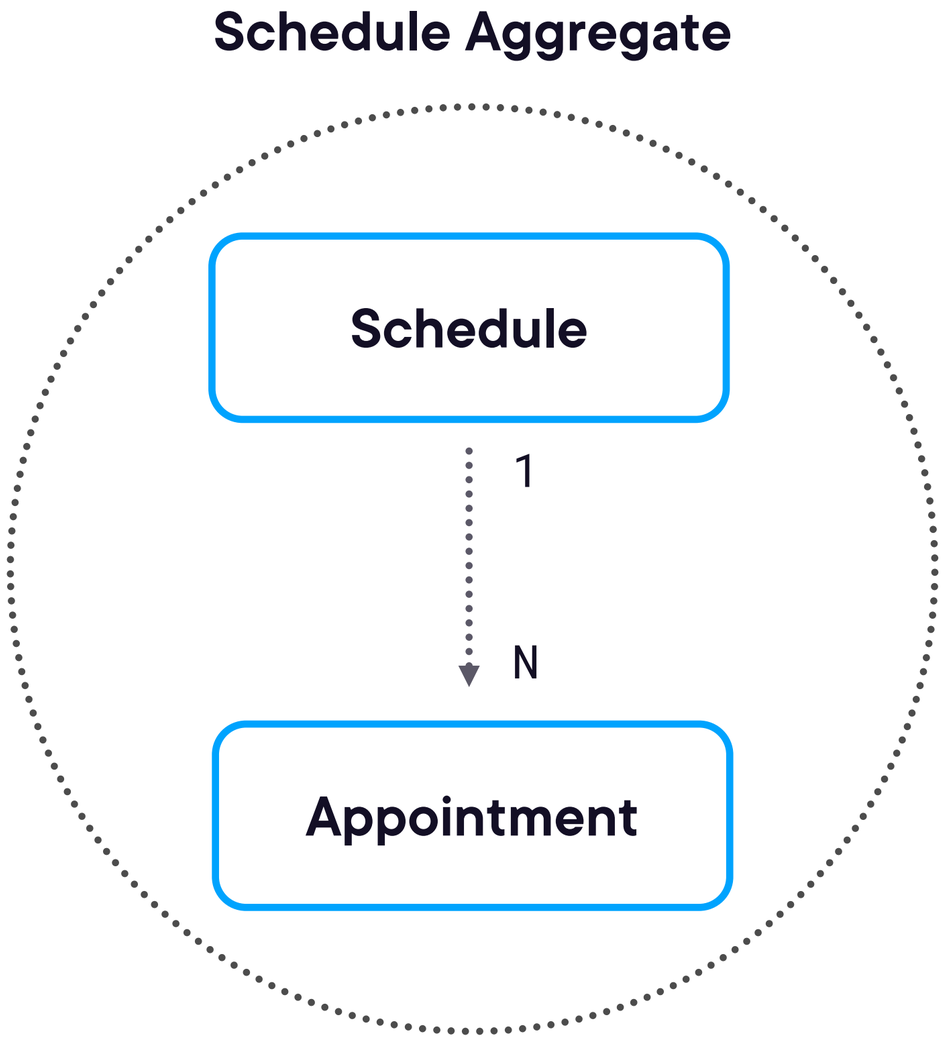
Our Red Flag

“Schedule”

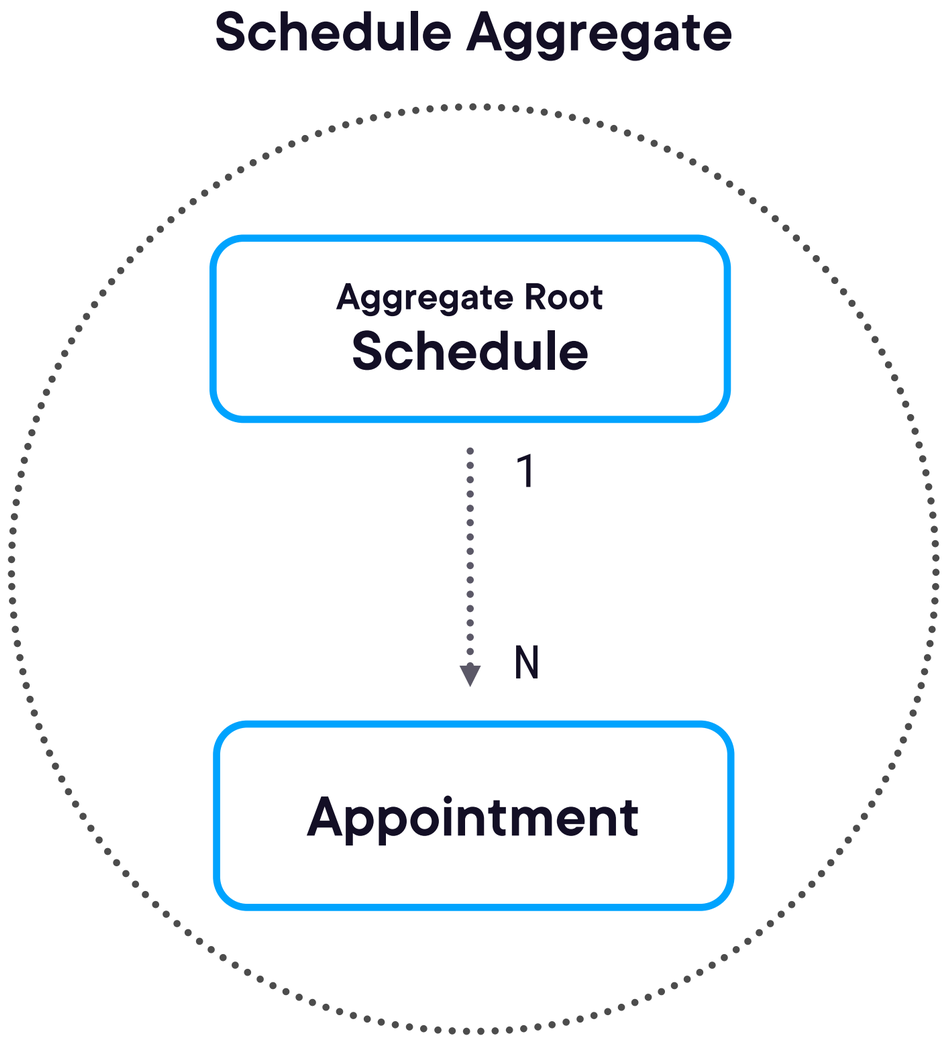
Introducing:
The Schedule Aggregate

Considering Schedule as Our New Aggregate

Our Newly Revised Design



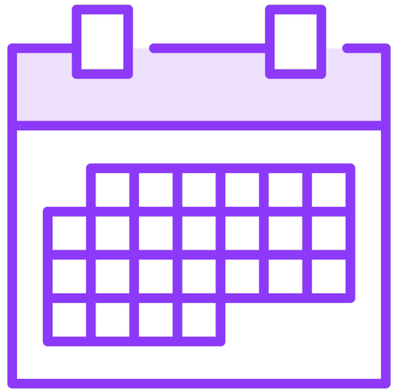
Does Schedule Make a Good Aggregate Root?



- Enforces invariants
- Saving changes can save entire aggregate
- Cascading delete is okay

Schedule as the Root Feels Right!

Each clinic has its own schedule



A schedule for a clinic has a series of appointments

Appointment :Entity<Int>

- AppointmentTypeId
- ClientId
- DoctorId
- DateTimeConfirmed
- PatientId
- RoomId
- IsPotentiallyConflicting
- DateTimeRange

Appointment coordinates date, time, patient, doctor and more

Exploring the Schedule Aggregate in Our Application

Sharing Our Tips for Aggregate Design

Aggregate Tips

Aggregates are not always the answer

Aggregates can connect only by the root

Don’t overlook using FKs for non-root entities

Too many FKs to non-root entities may suggest a problem

“Aggregates of one” are acceptable

“Rule of Cascading Deletes”

Module Review with Key Terms

Aggregate

A transactional graph of objects

Aggregate Root

The entry point of an aggregate which ensures the integrity of the entire graph

Invariant

A condition that should always be true for the system to be in a consistent state

Associations

The modeled relationship between entities

Navigation Properties

An ORM term to describe properties that reference related objects

Unidirectional Relationships

Associations between two entities that can only be navigated in one direction

Key Takeaways

Avoid big ball of mud models

Break the model up into aggregates

An aggregate represents a graph of objects in a transaction

Aggregates encapsulate business rules and invariants

Default to one-way relationships when modeling associations

Don’t fear evolving the design of your aggregates as you learn more about the domain

Repositories are a critical pattern.