Extending the Bounded Context with Aggregates

Vladimir Khorikov

@vkhorikov | www.enterprisecraftsmanship.com

In This Module

- Purchase functionality
- Aggregates

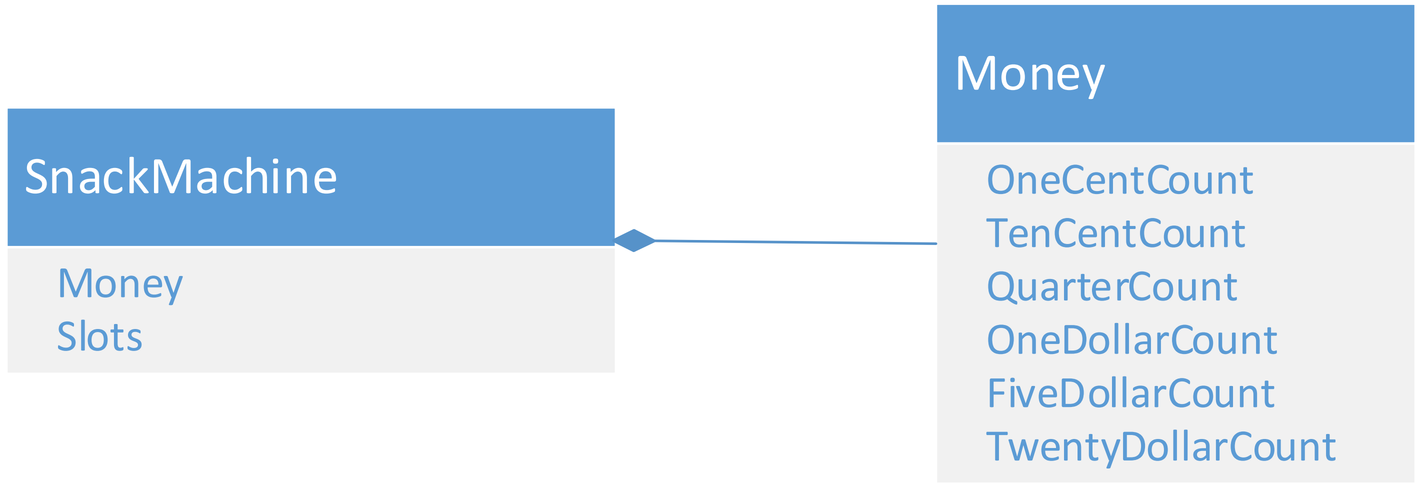
Problem Description

- 3 slots of snacks
- Return the change
- Check if inserted money is enough and the slot isn't empty
- Check if there's enough change

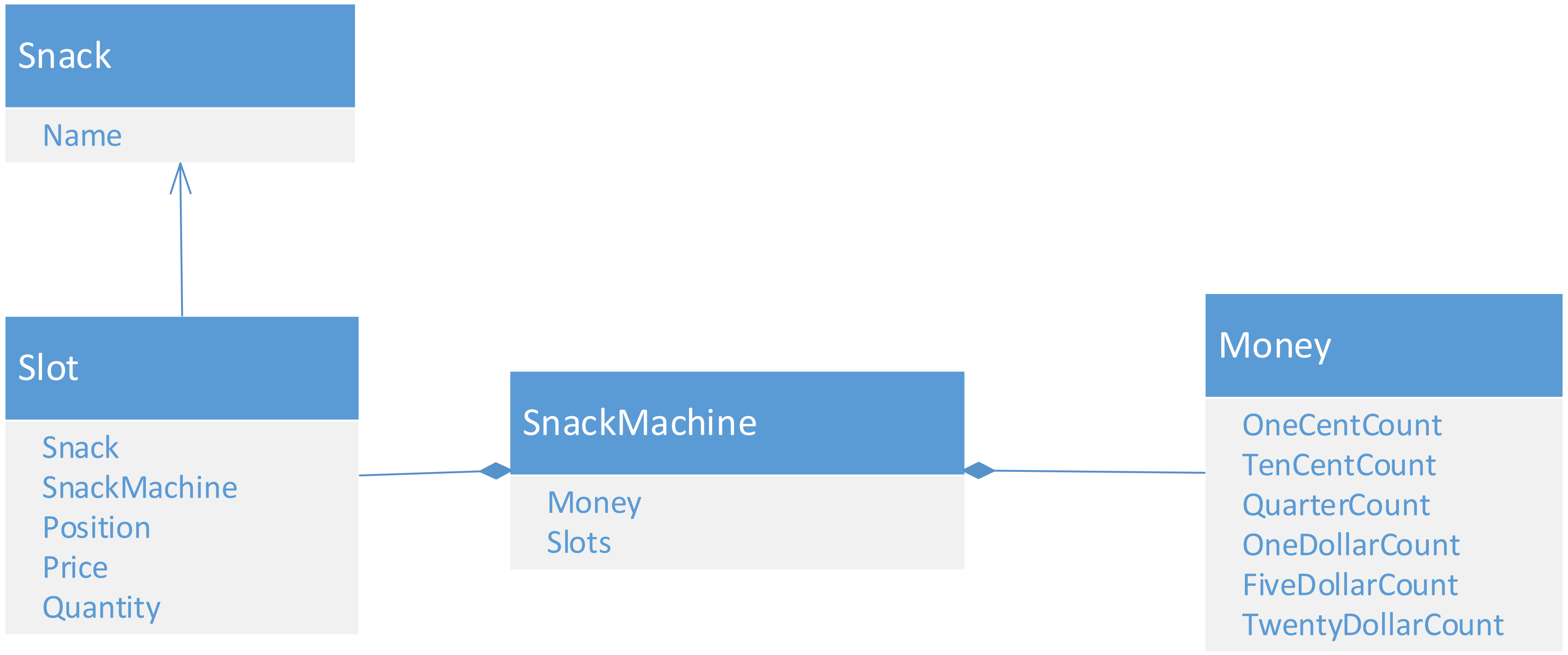
Scope for the Module

- Iterative approach to design
- 1-to-many relationships
- Combining several entities into aggregates

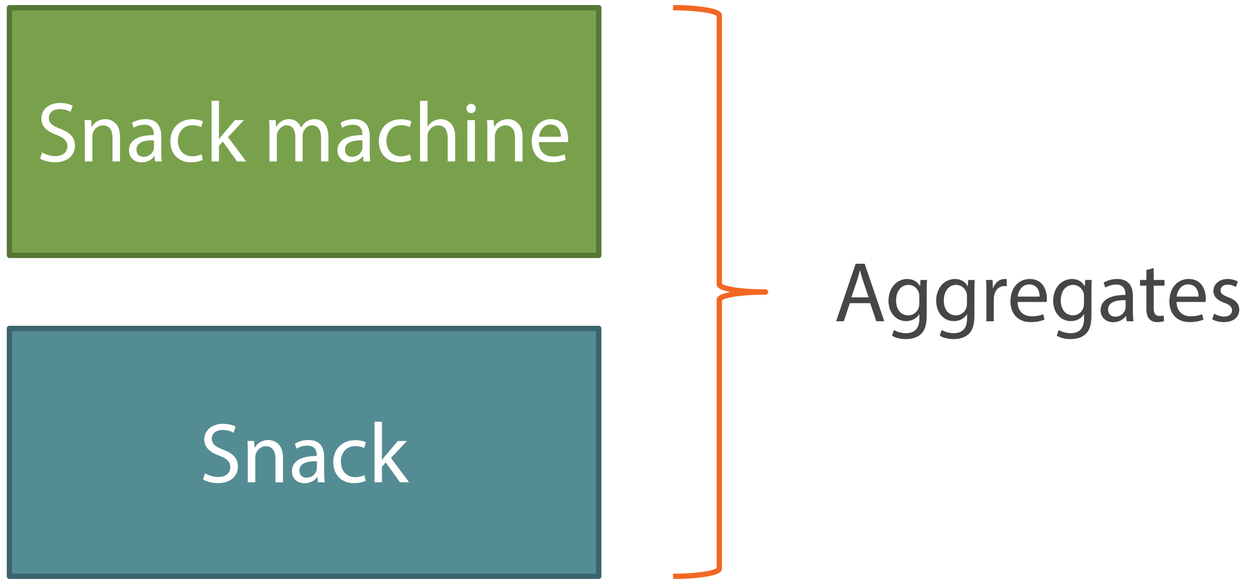
Domain Model

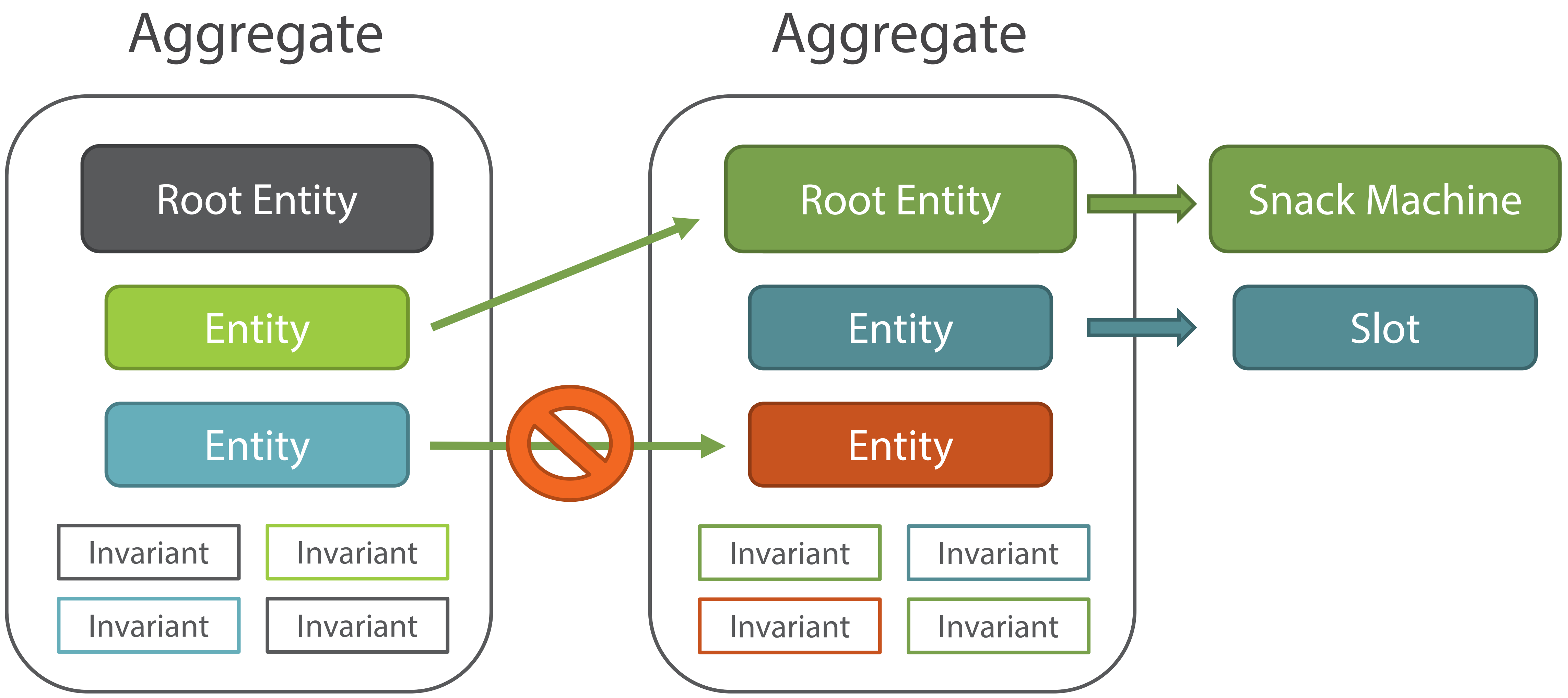


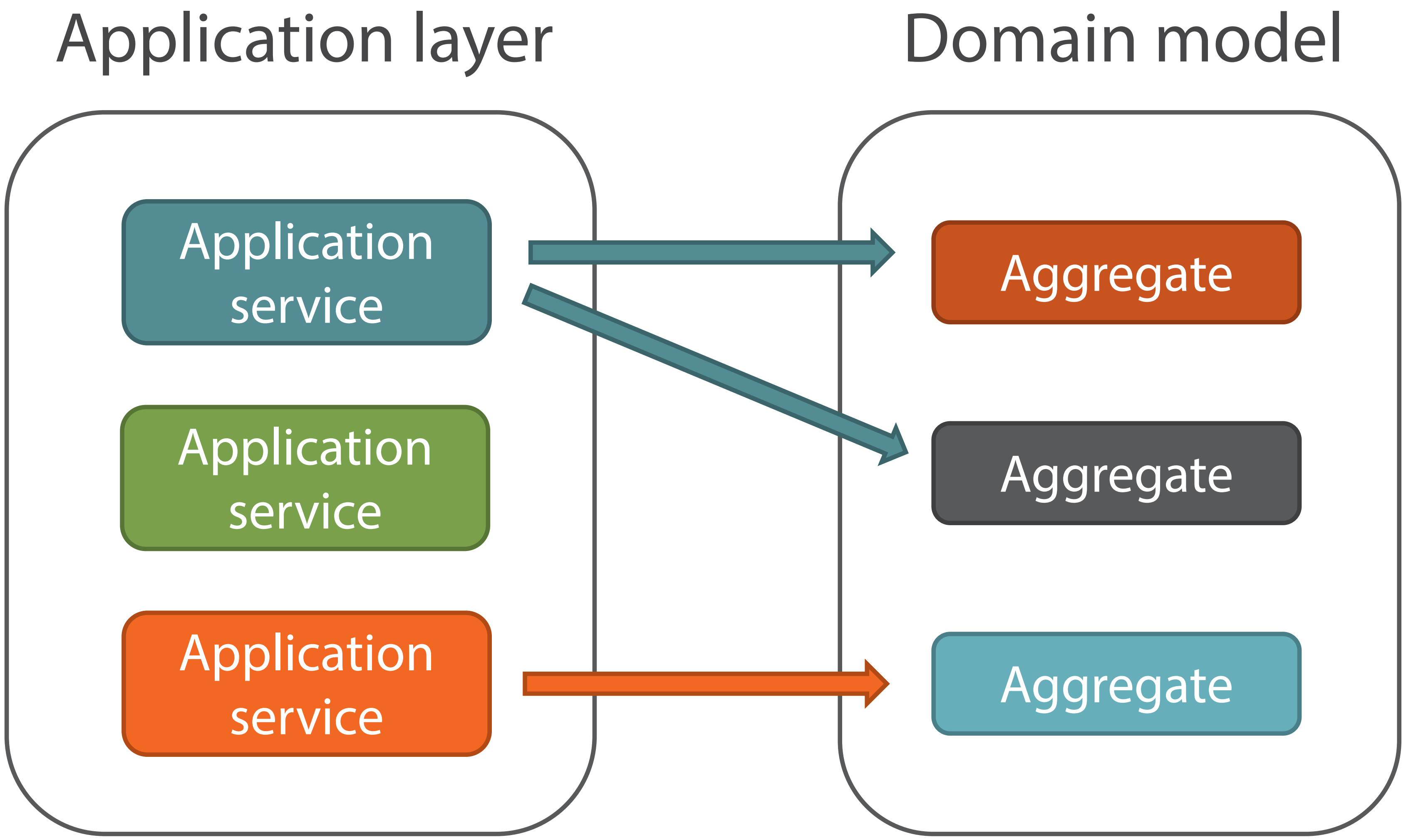
Domain Model



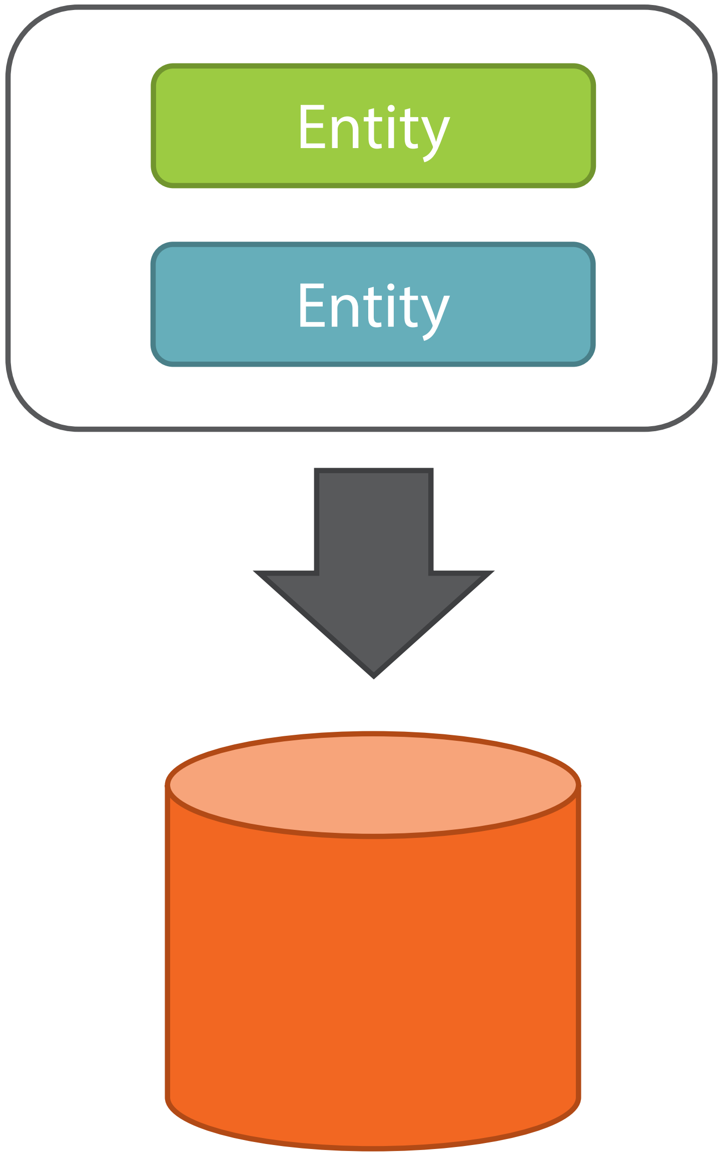
Aggregates




Aggregate
Aggregate
Root Entity
Root Entity
Snack Machine
Entity
Entity
Slot
Entity
Entity
Invariant
Invariant
Invariant
Invariant
Invariant
Invariant
Invariant
Invariant


Application layer
Domain model
Application service
Application service
Application service
Aggregate
Aggregate
Aggregate

Aggregate



Aggregates

Entity

- ☐ Can belong to a single aggregate only

Value Object

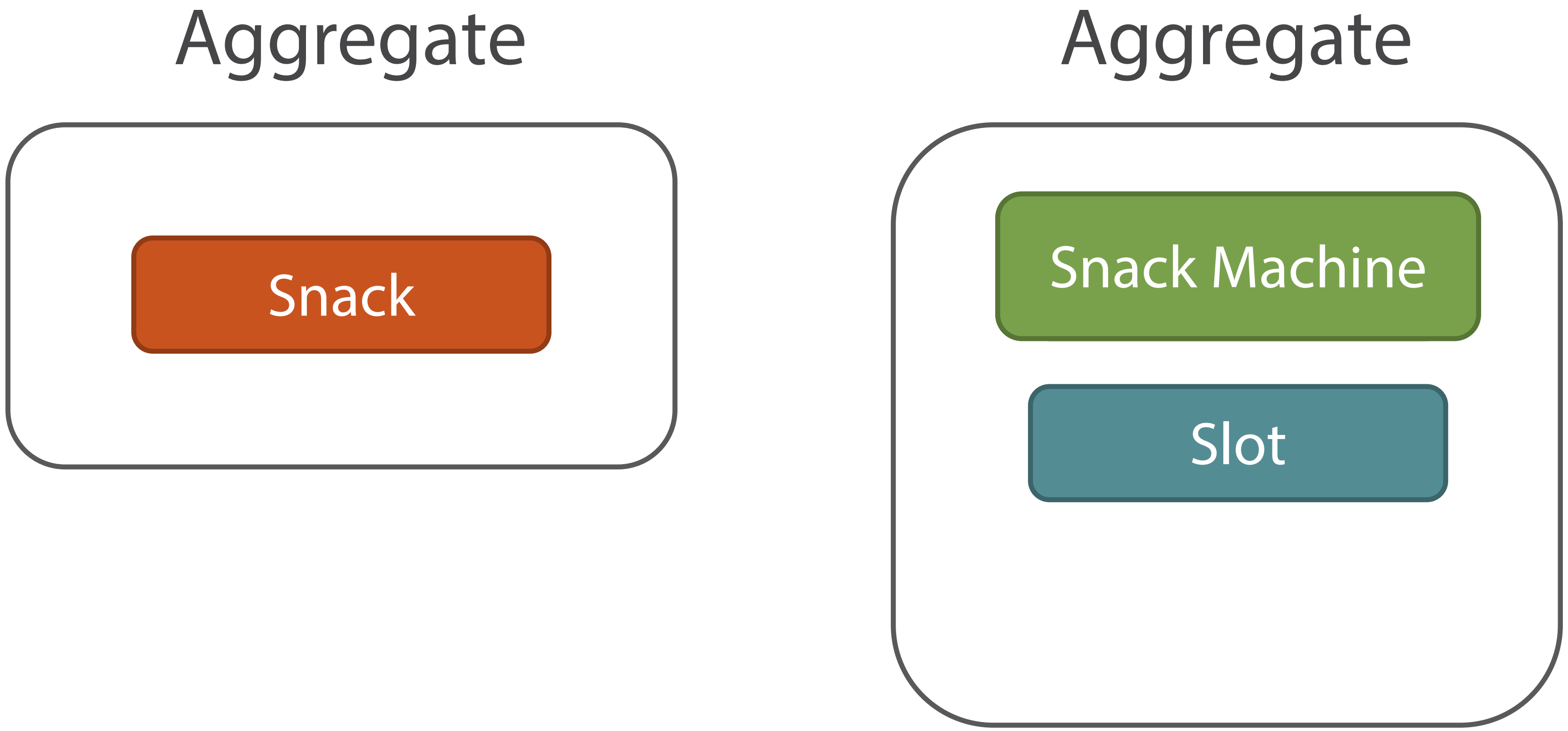
- ☐ Can belong to multiple aggregates

How to choose boundaries for Aggregates?

How to Find Boundaries for Aggregates

- Entities inside comprise a cohesive group of classes
- Don’t hesitate to change boundaries when you discover more information
- Don’t create aggregates that are too large

How to Find Boundaries for Aggregates



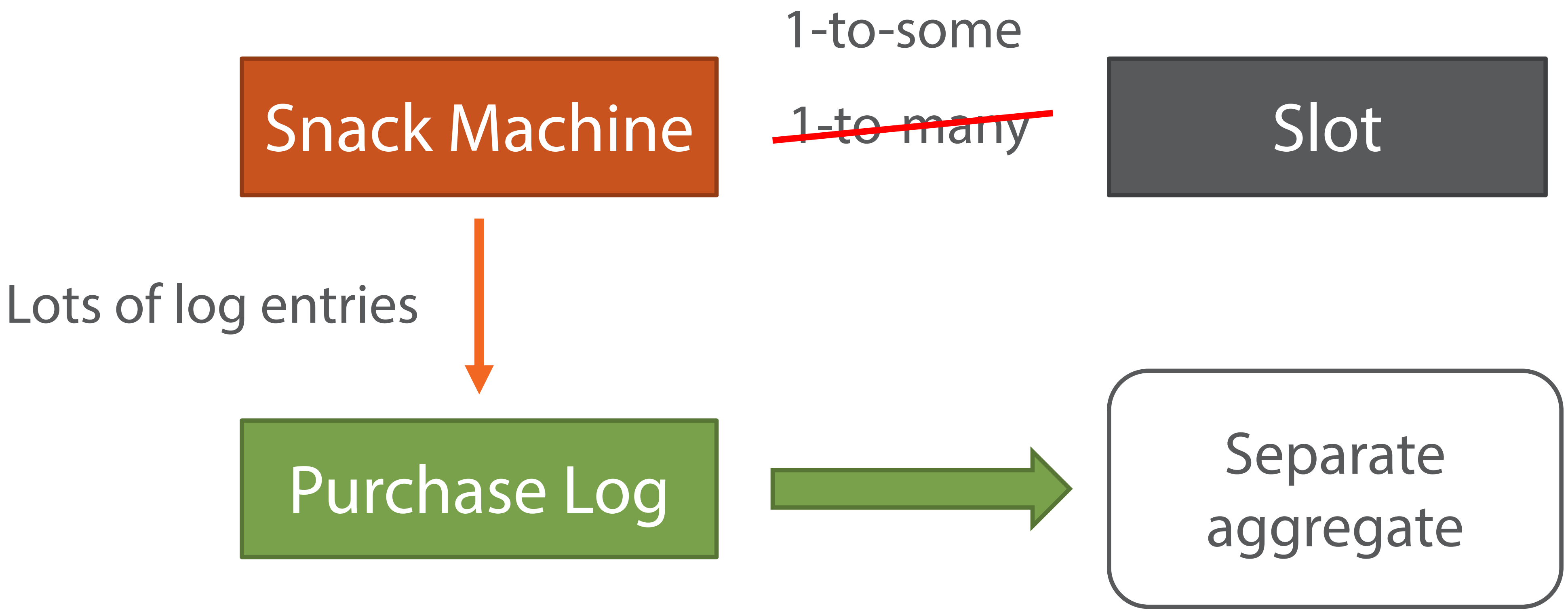
Hard to maintain consistency

How to Find Boundaries for Aggregates

Simplicity VS Performance

- ☐ Most aggregates consist of 1 or 2 entities
- ☐ 3 entities per aggregate is usually a max
- ☐ The number of Value Objects per aggregate is unlimited

How to Find Boundaries for Aggregates



Aggregate Root Base Class

```
public abstract class AggregateRoot : Entity
{
}
```

```
public class SnackMachine : AggregateRoot
{
}
```

Aggregate Root Base Class

```
public abstract class AggregateRoot : Entity
{
    public virtual int Version { get; protected set; }
    private List<DomainEvent> _events = new List<DomainEvent>();
}


public class SnackMachine : AggregateRoot
{
}
```

Aggregate Root Base Class

```
public abstract class AggregateRoot : Entity
{
    public virtual int Version { get; protected set; }
    private List<DomainEvent> _events = new List<DomainEvent>();
}


public class SnackMachine : AggregateRoot
{
}
```

Recap: Refactoring the Snack Machine Aggregate

- Fully encapsulated aggregate
- Not exposing internal Slot entity
- Exposing SnackPile value object instead
- New abstraction to resolve the awkwardness

Recap: Refactoring the Snack Machine Aggregate

```
public virtual void BuySnack(int position)
{
    Slot slot = GetSlot(position);
    slot.SnackPile = slot.SnackPile.SubtractOne();
}


public sealed class SnackPile : ValueObject<SnackPile>
{
    public SnackPile SubtractOne()
    {
        return new SnackPile(Snack, Quantity - 1, Price);
    }
}
```

Implementing Missing Requirements

- Inserted money is sufficient
- Snack pile is not empty
- Return the change
- The amount of money inside is sufficient to return the change

Recap: Revealing a Hidden Requirement

- Inserted money is sufficient
- Snack pile is not empty
- Return the change
- The amount of money inside is sufficient to return the change
- Retain small coins and notes

Recap: Revealing a Hidden Requirement

```
public class SnackMachine : AggregateRoot
{
    public virtual Money MoneyInside { get; protected set; }
    ~~public virtual Money MoneyInTransaction { get; protected set; }~~
    public virtual decimal MoneyInTransaction { get; protected set; }
}
```

Summary

- Aggregates gather multiple entities under a single abstraction
  - Conceptual whole
  - Root entity
  - Single operational unit for the application layer
  - Consistency boundaries
- How to find proper boundaries for aggregates
  - Does an entity makes sense by its own?
  - Try not to expose internal entities outside the aggregate
- Revealing a hidden abstraction

In the Next Module

Introducing repositories