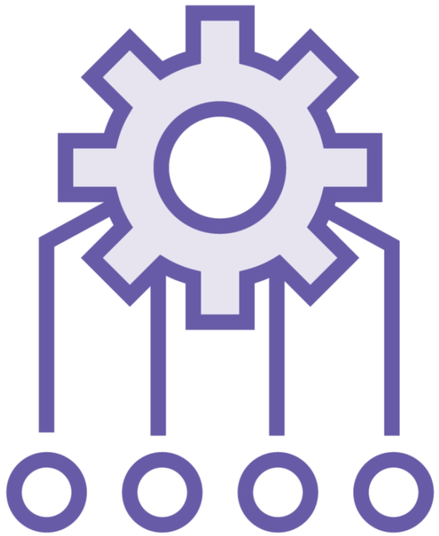
Modeling the Domain with Types

Zoran Horvat

CEO at Coding Helmet

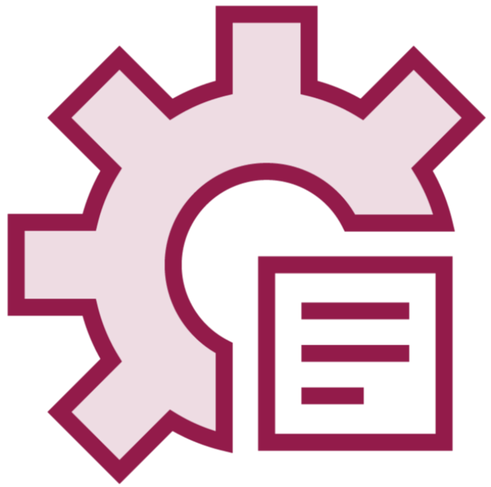
@zoranh75 https://codinghelmet.com

Functional vs. Object-oriented Design



Functional design

Separation of functions and types



Object-oriented design

Functions defined on types

EXPLORER

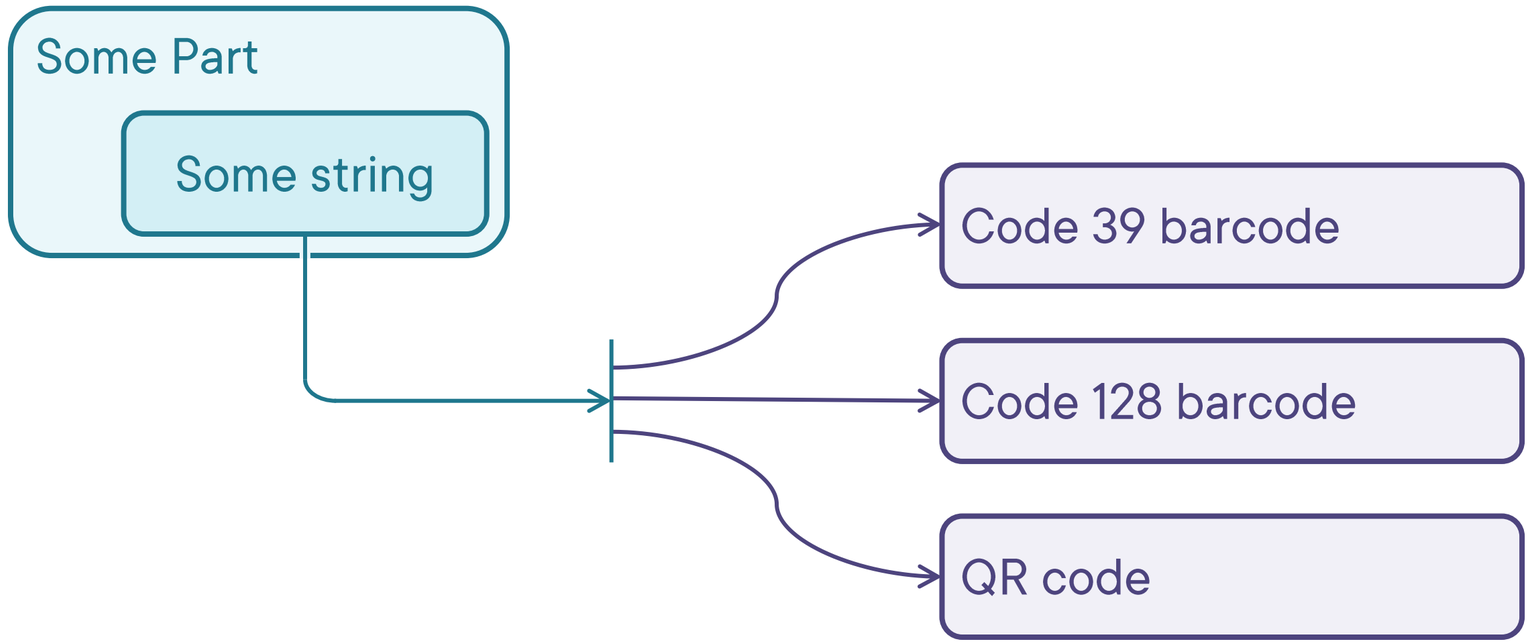
DEMO

- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```



TIMELINE

EXPLORER

DEMO

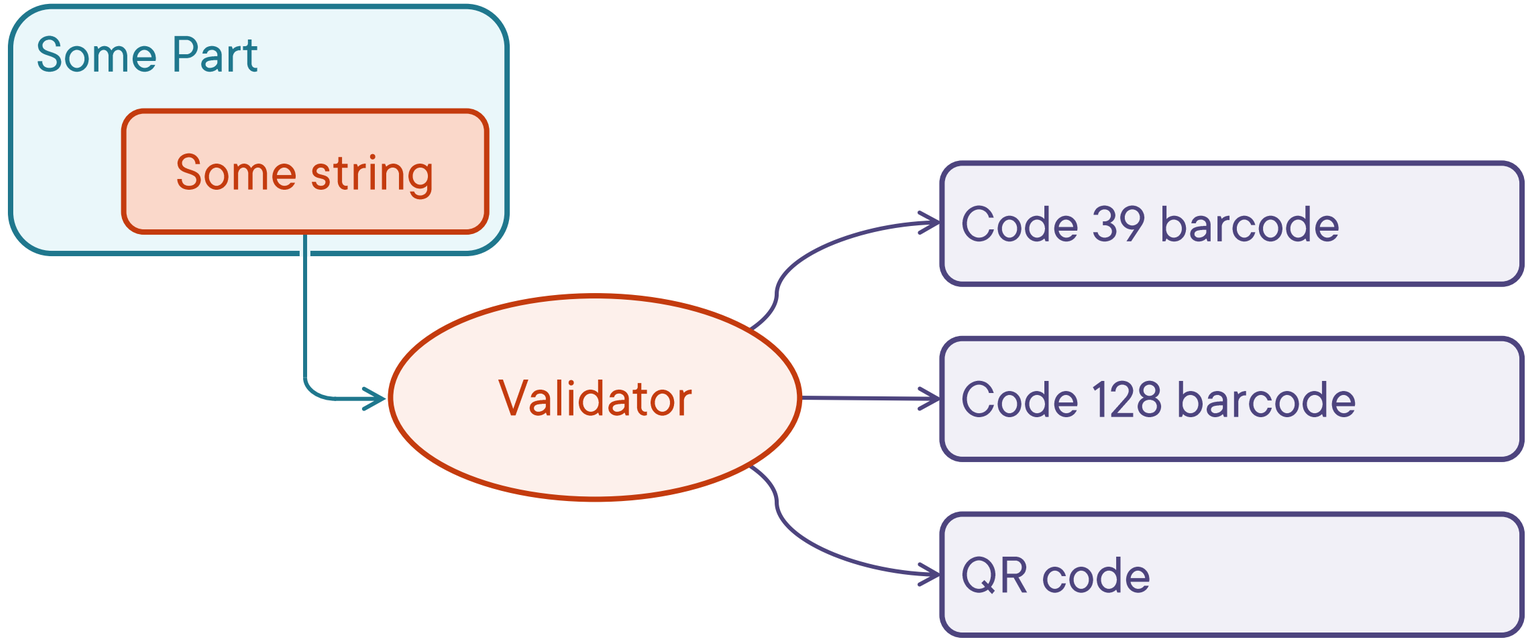
- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

TIMELINE

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```



EXPLORER

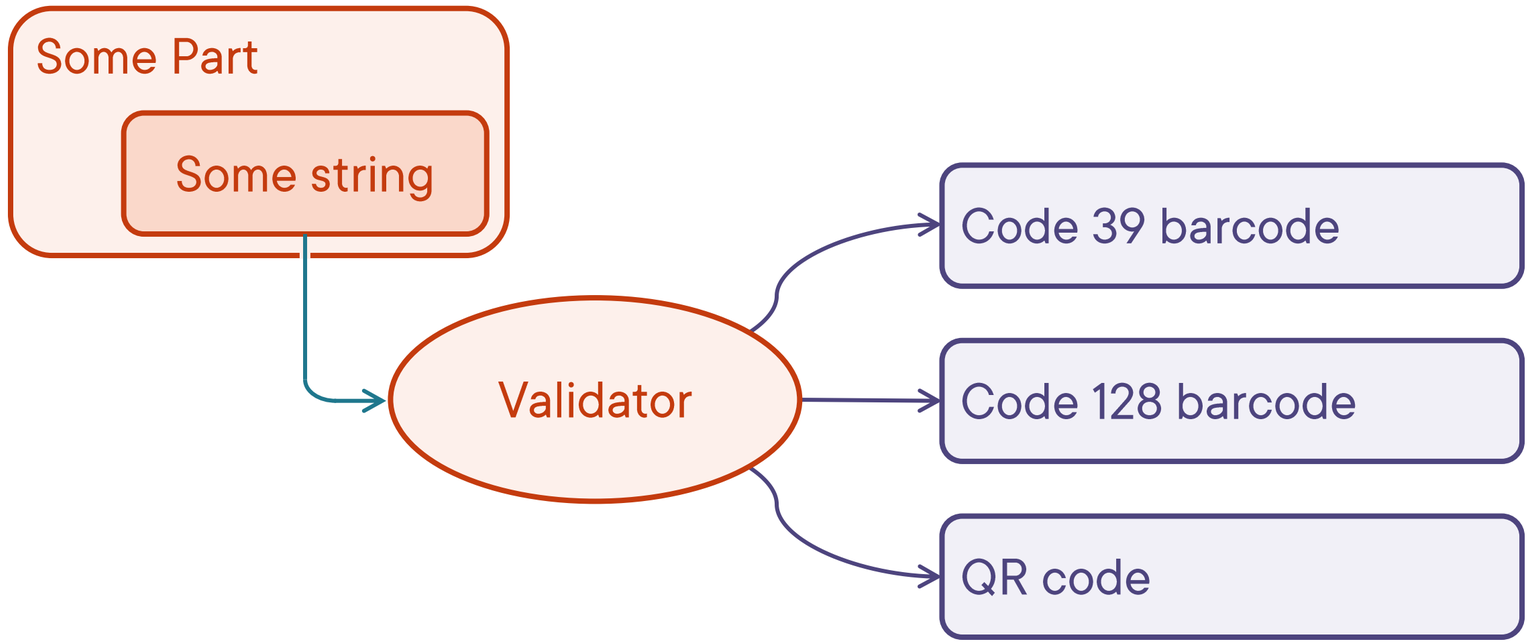
DEMO

- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```



TIMELINE

EXPLORER

DEMO

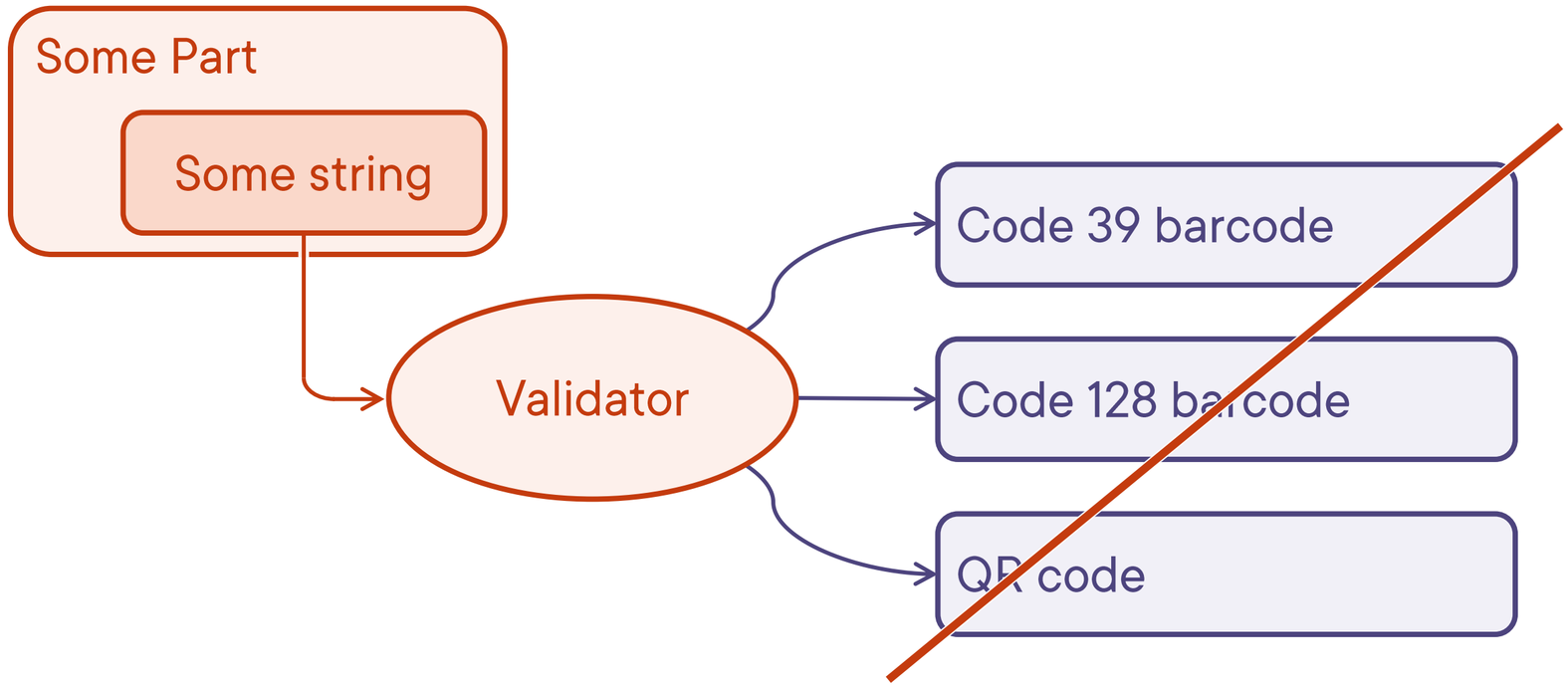
- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

TIMELINE

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```



EXPLORER

DEMO

- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

TIMELINE

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```

EXPLORER

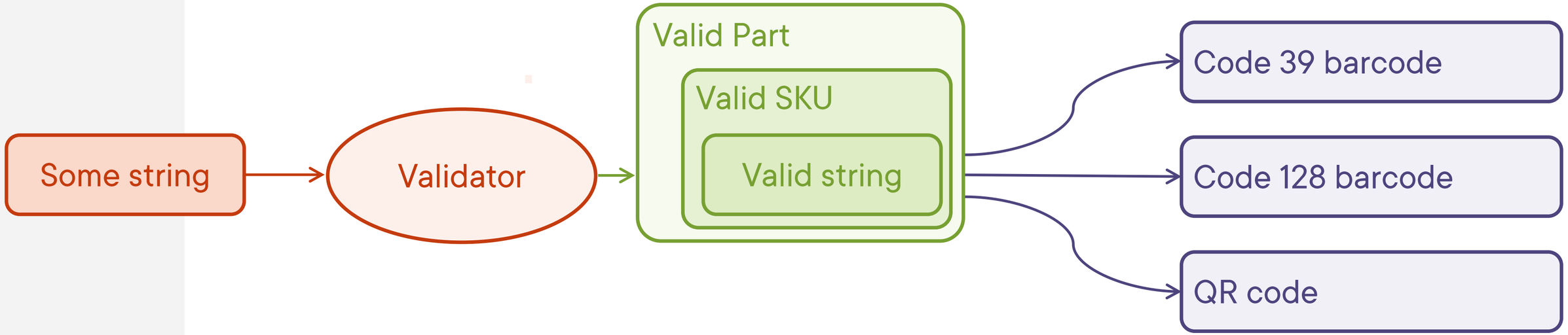
DEMO

- Models
  - Time
  - Types
    - Part.cs
  - Models.csproj
- Web
- Demo.sln

Part.cs

Models > Types > Part.cs > { } Models.Types > Models.Types.StockKeepingUnit

```
1 namespace Models.Types;
2
3 public record Part(string Name, StockKeepingUnit Sku);
4 public record StockKeepingUnit(string Value);
```



TIMELINE

DEMO
Models
Time
Types
Part.cs
StockKeepingUnit.cs
Models.csproj
Web
Demo.sln

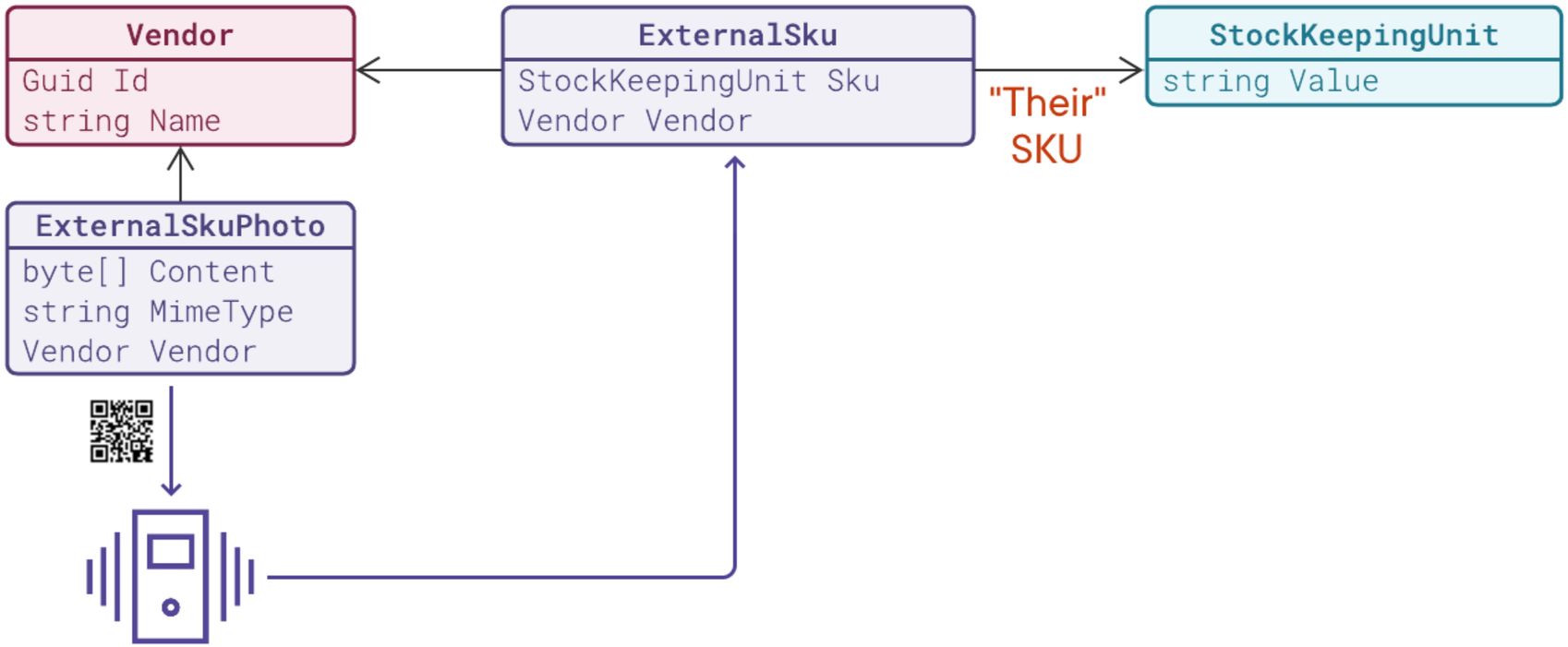
```
1 namespace Models.Types;
2
3 public record StockKeepingUnit(string Value);
4 public record ExternalSku(StockKeepingUnit Sku, Vendor Vendor);
5 public record Vendor(Guid Id, string Name);
```

DEMO
Models
Time
Types
Part.cs
StockKeepingUnit.cs
Models.csproj
Web
Demo.sln

```
1 namespace Models.Types;
2
3 public record StockKeepingUnit(string Value);
4 public record ExternalSku(StockKeepingUnit Sku, Vendor Vendor);
5 public record Vendor(Guid Id, string Name);
```

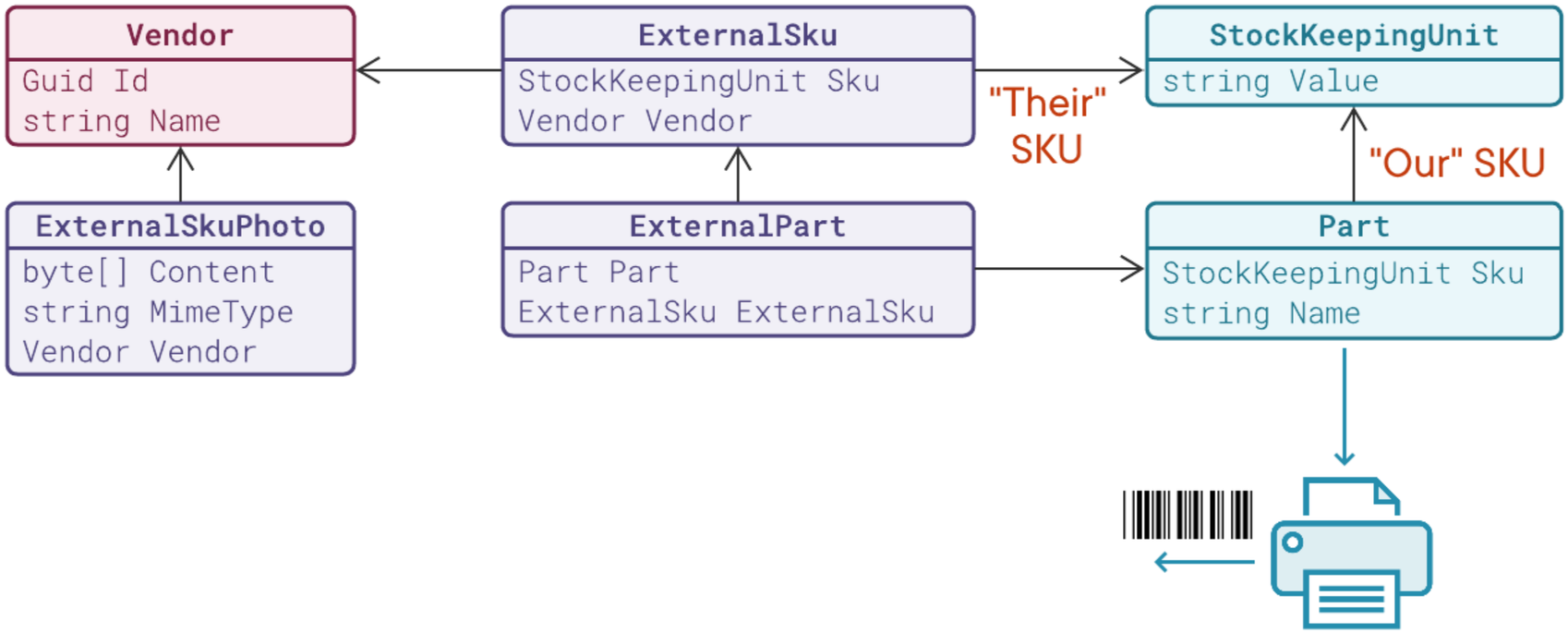
DEMO
Models
Time
Types
Part.cs
StockKeepingUnit.cs
Models.csproj
Web
Demo.sln

```
1 namespace Models.Types;
2
3 public record StockKeepingUnit(string Value);
4 public record ExternalSku(StockKeepingUnit Sku, Vendor Vendor);
5 public record Vendor(Guid Id, string Name);
6 public record ExternalPart(Part Part, ExternalSku ExternalSku);
7 public record ExternalSkuPhoto(byte[] Content, string MimeType, Vendor Vendor);
```



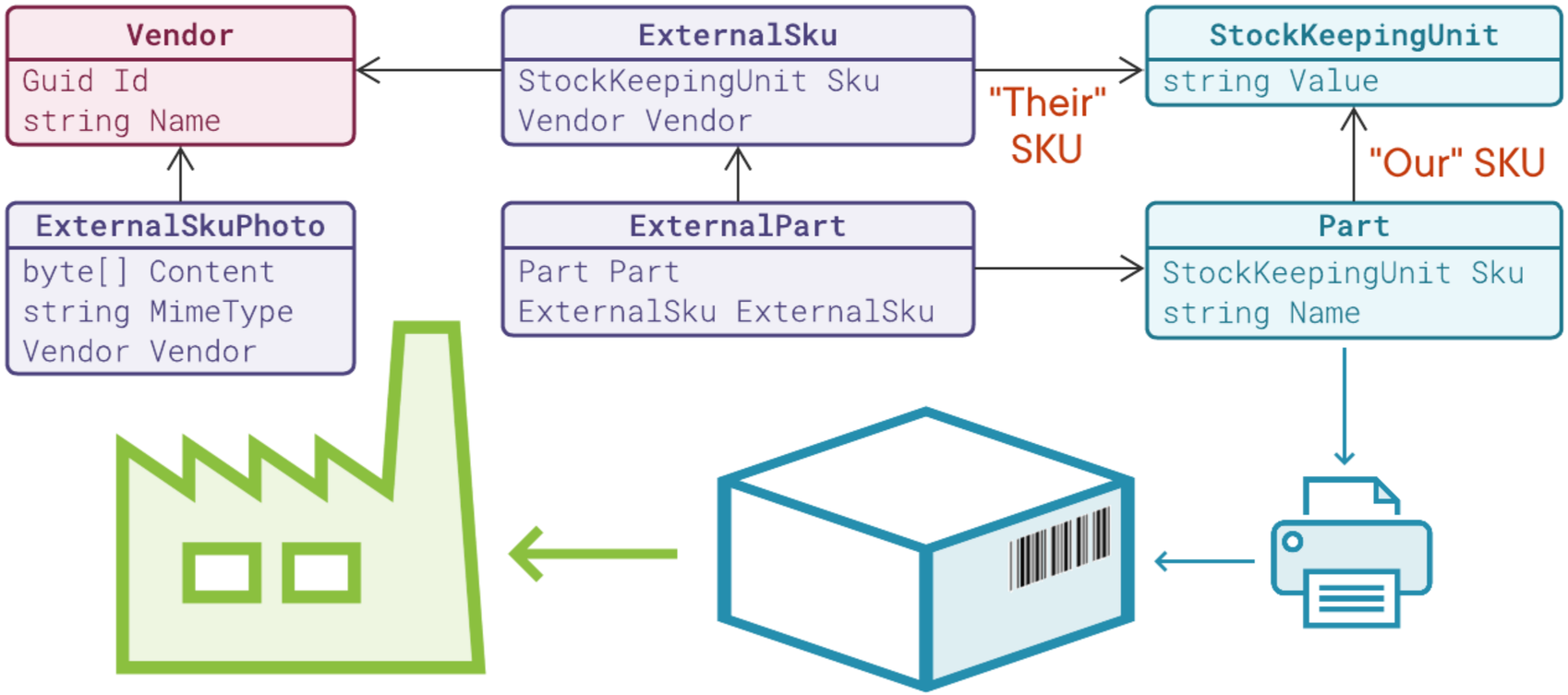
DEMO
Models
Time
Types
Part.cs
StockKeepingUnit.cs
Models.csproj
Web
Demo.sln

```
1 namespace Models.Types;
2
3 public record StockKeepingUnit(string Value);
4 public record ExternalSku(StockKeepingUnit Sku, Vendor Vendor);
5 public record Vendor(Guid Id, string Name);
6 public record ExternalPart(Part Part, ExternalSku ExternalSku);
7 public record ExternalSkuPhoto(byte[] Content, string MimeType, Vendor Vendor);
```



DEMO
Models
Time
Types
Part.cs
StockKeepingUnit.cs
Models.csproj
Web
Demo.sln

```
1 namespace Models.Types;
2
3 public record StockKeepingUnit(string Value);
4 public record ExternalSku(StockKeepingUnit Sku, Vendor Vendor);
5 public record Vendor(Guid Id, string Name);
6 public record ExternalPart(Part Part, ExternalSku ExternalSku);
7 public record ExternalSkuPhoto(byte[] Content, string MimeType, Vendor Vendor);
```



Principles of Functional Modeling

Separate types from functions

- There are types to describe elements of the business
- There are functions to describe business processes

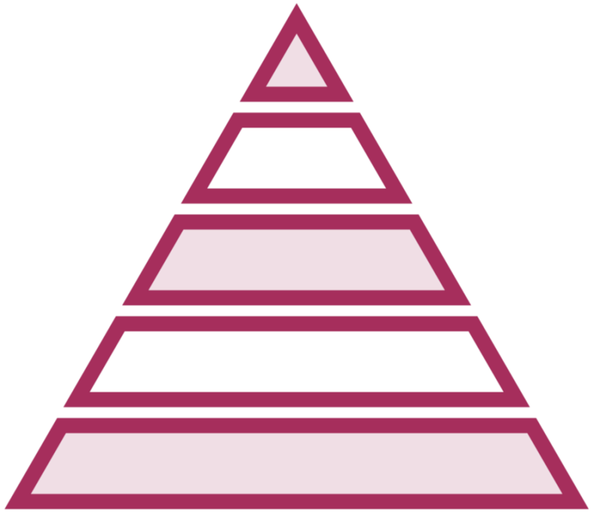
The number of types is bounded in any business domain

- Adding more types to the model at first
- No more types to add beyond certain point

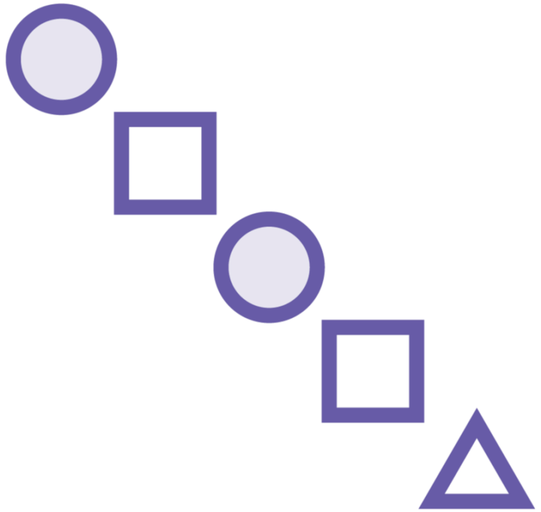
Modeling includes defining functions to model behavior

- Functions apply to instances of types, returning instances of types
- There will always be new behavior to add to the model

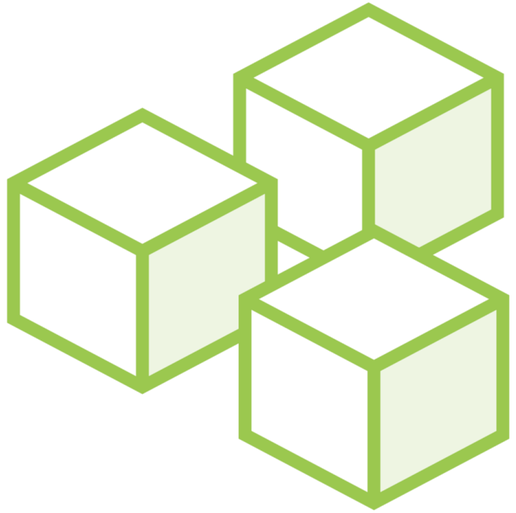
Object-oriented vs. Functional Modeling



It is hard to add a new method to a hierarchy of classes

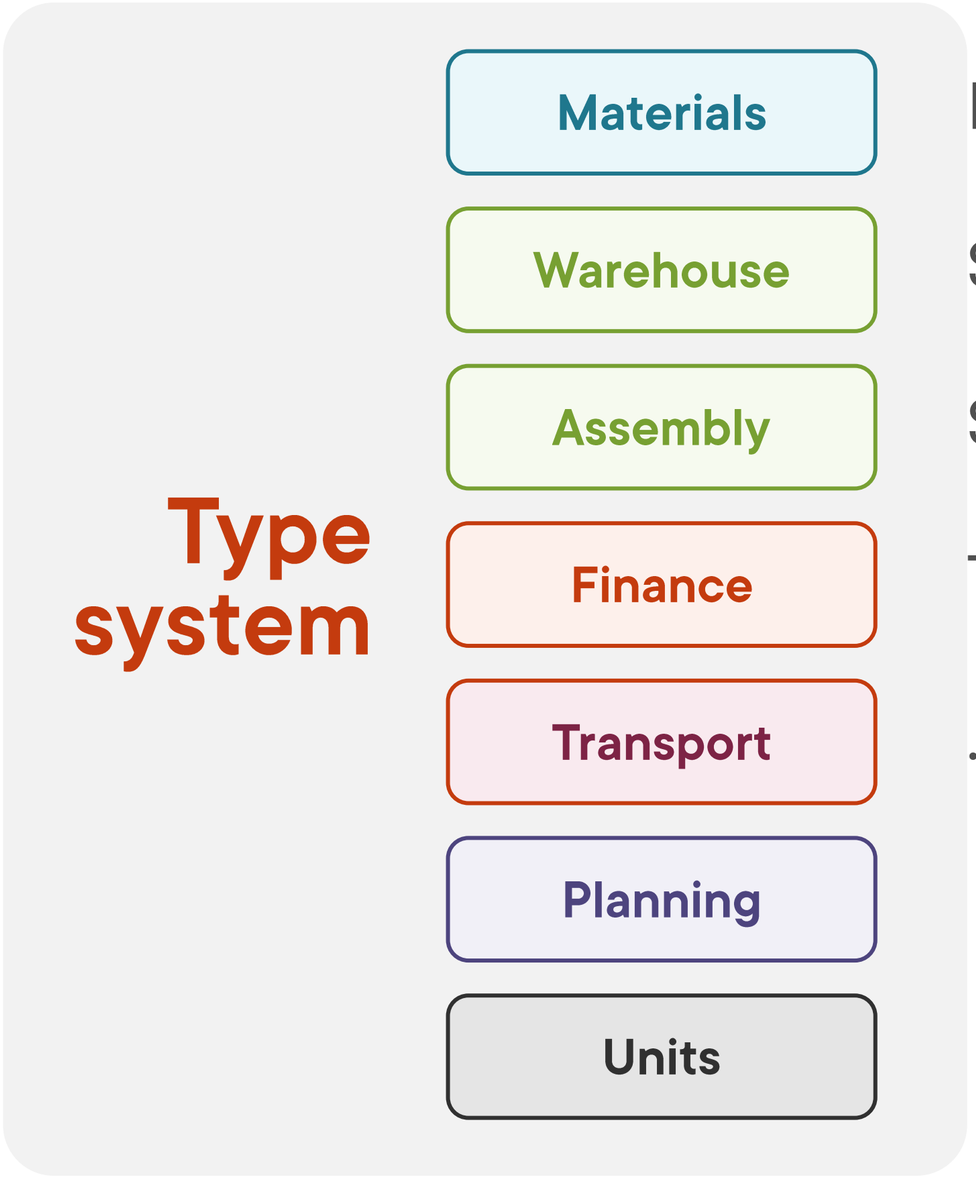


It is easy to add a new function to existing functions



Modern OOP favors object composition over inheritance

An Example: Modeling a Manufacturing Business



Parts, materials, elements…

Stockkeeping of parts, materials…

Specifications, blueprints, assembly instructions…

Tracking money, costs, procurement…

…

An Example: Modeling a Manufacturing Business

Type system

- Materials
- Warehouse
- Assembly
- Finance
- Transport
- Planning
- Units

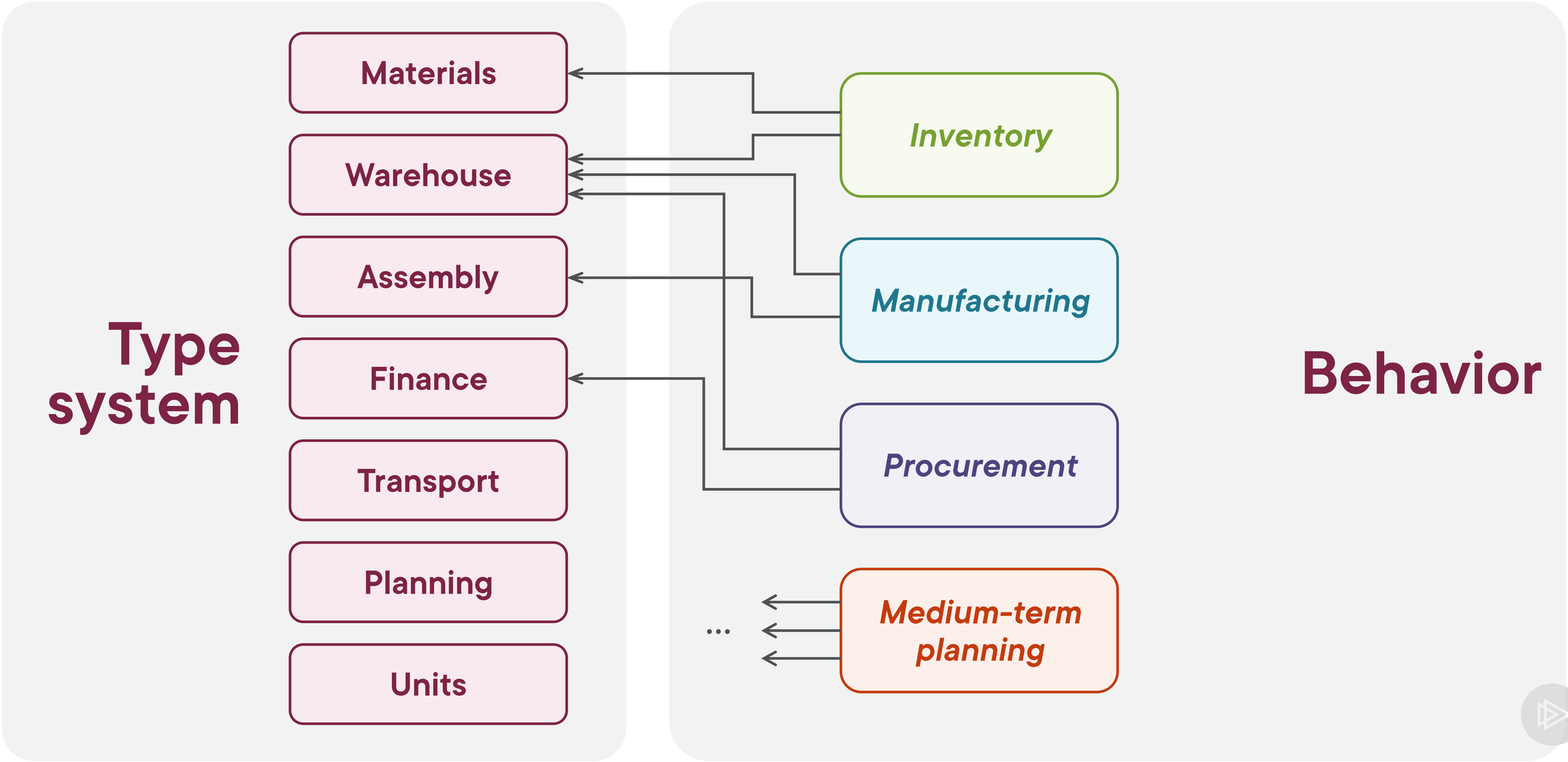
```
Part
StockKeepingUnit Sku
string Name
```

```
StockKeepingUnit
string Value
```

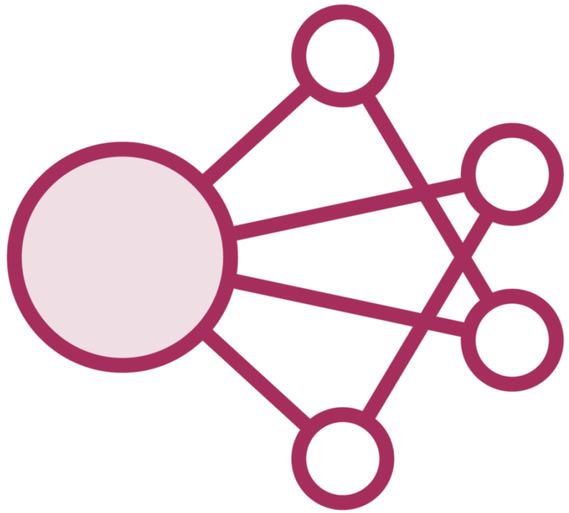
```
PhysicalPart
Part Part
Weight Weight
Volume Volume
```

```
Package
PhysicalPart Part
Amount Amount
```

An Example: Modeling a Manufacturing Business



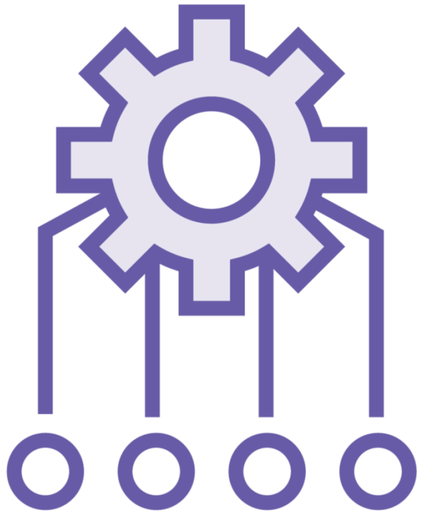
Object-oriented vs. Functional Modeling



Object-oriented Design

Working actively to keep the design maintainable

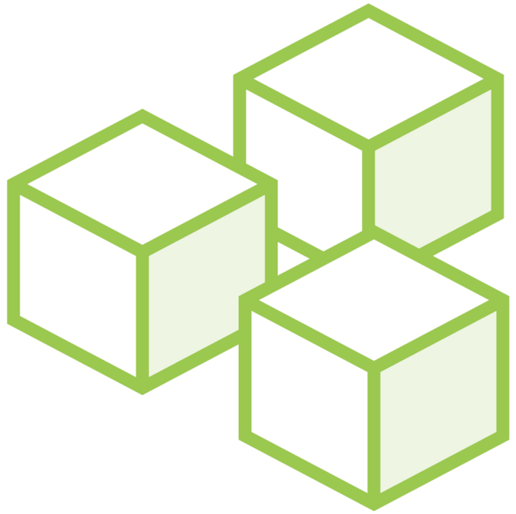
Requires knowledge



Functional Design

Forced to separate concepts by design

Design starts off good!



Functional C#

Only use common C# features

Attain functional design

EXPLORER

DEMO

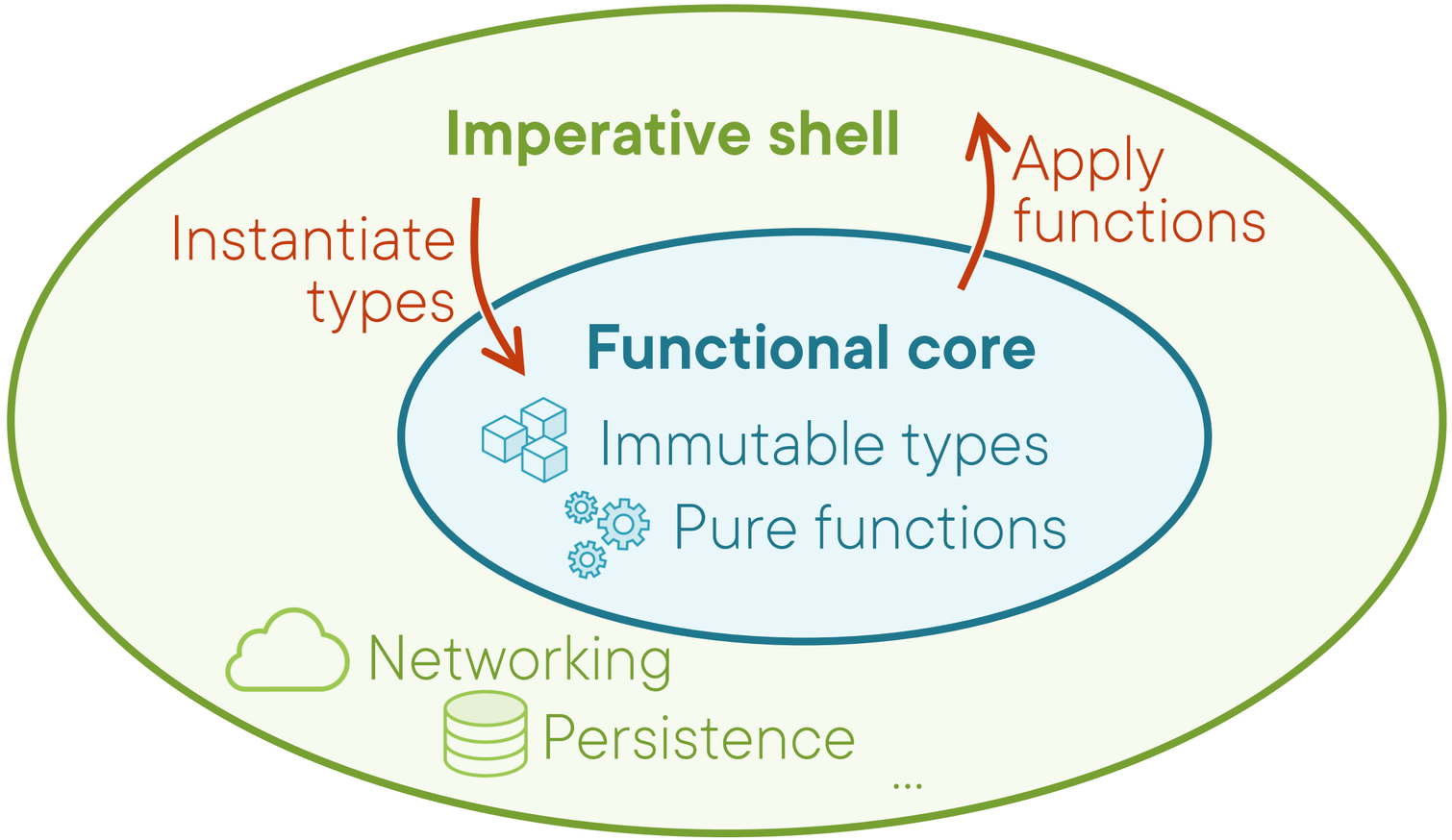
- Application
  - Persistence
    - IReadOnlyRepository.cs
  - Application.csproj
- Models
- TestPersistence
- Web
- Demo.sln

TIMELINE

IReadOnlyRepository.cs

Application > Persistence > IReadOnlyRepository.cs > { } Application.Persistence

```
1 namespace Application.Persistence;
2
3 public interface IReadOnlyRepository<T>
4 {
5     IEnumerable<T> GetAll();
6 }
```



0 0 Demo.sln Ln 1, Col 1 Spaces: 4 UTF-8 CRLF C#

Summary

Modeling the domain using types

- Type contains other types, including primitive types
- Types expose no behavior

Modeling behavior with functions

- We can attach extension methods to types

Functional modeling process

- Add more types to the model
- Define more functions that apply to types
- Many types can be .NET records

Up Next:
Designing Pure Functions