Intentions & Interfaces
Making patterns concrete

Udi Dahan – The Software Simplist

.NET Development Expert & SOA Specialist

www.UdiDahan.com

email@UdiDahan.com

Books, and books, and books...

Flexibility brings many benefits

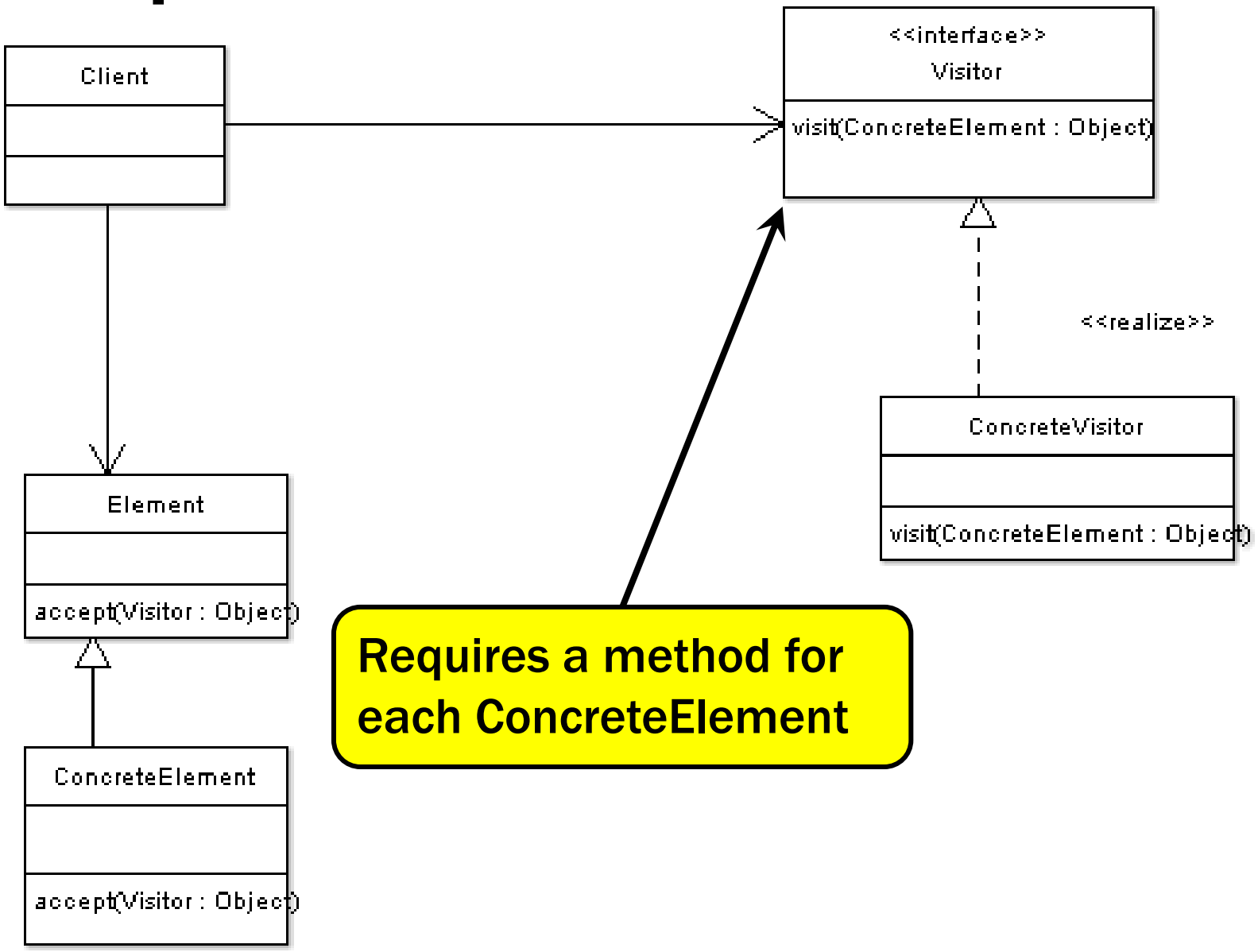
Preventing rigidity from creeping in

Existing solutions

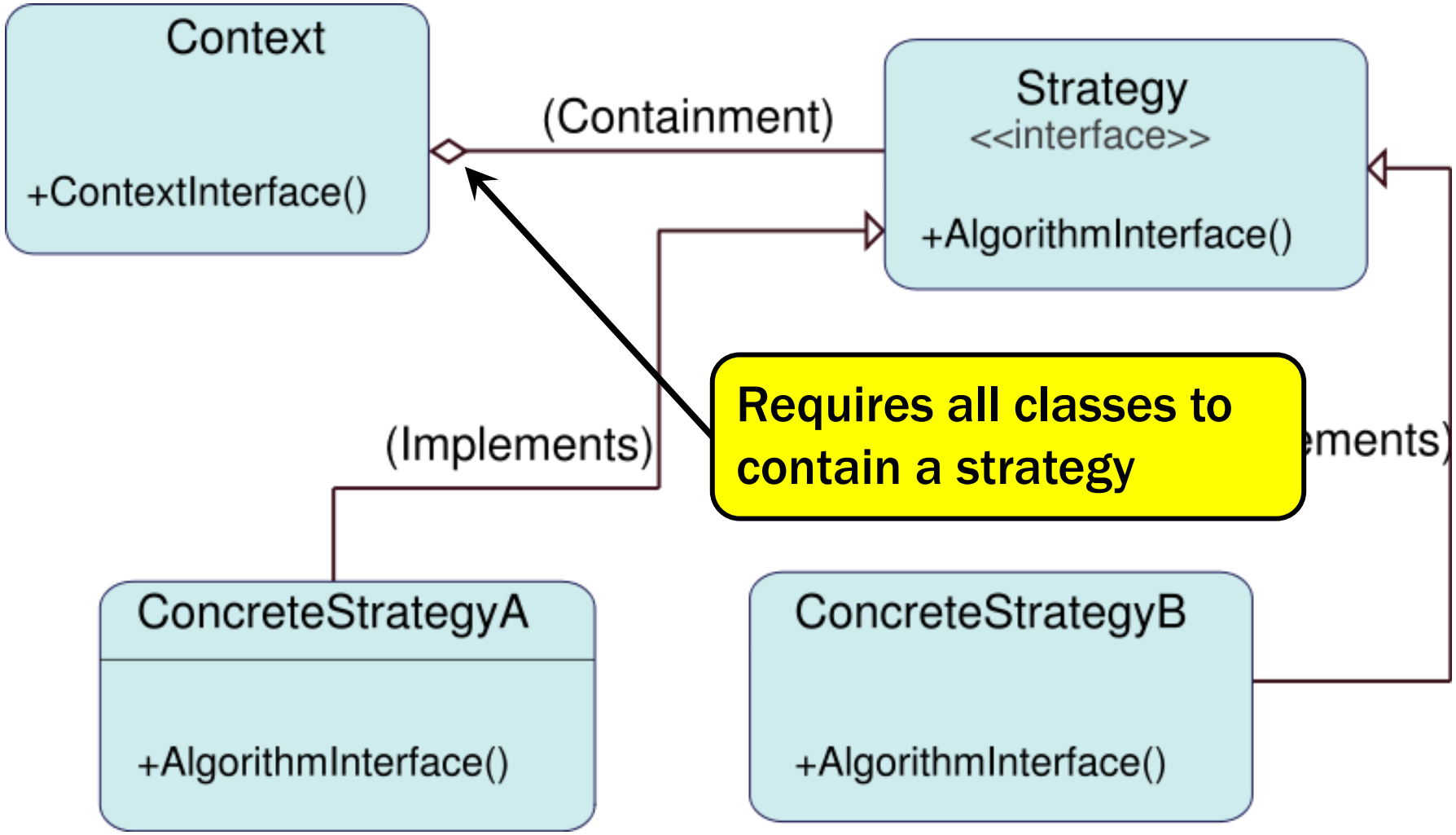
Visitor Pattern

Strategy Pattern

Visitor pattern



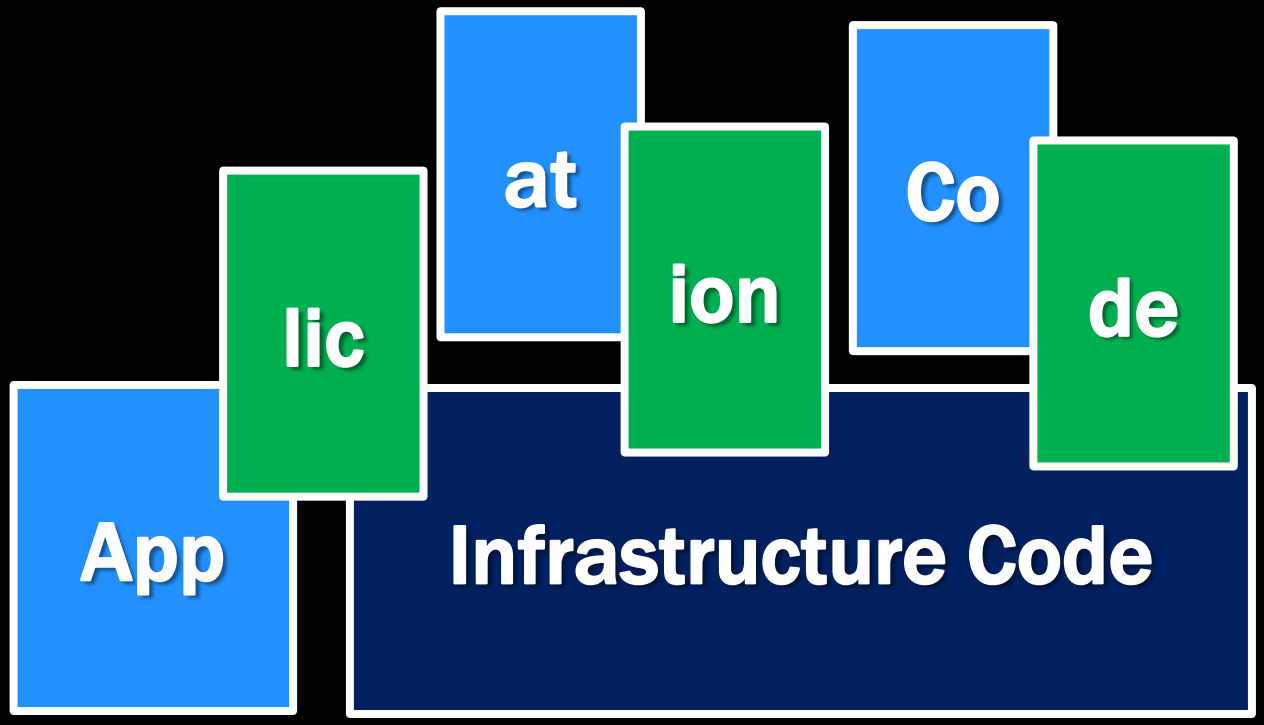
Strategy pattern



May cause a collapse of application structure

Application Code

Infrastructure Code



Doing too much can hurt

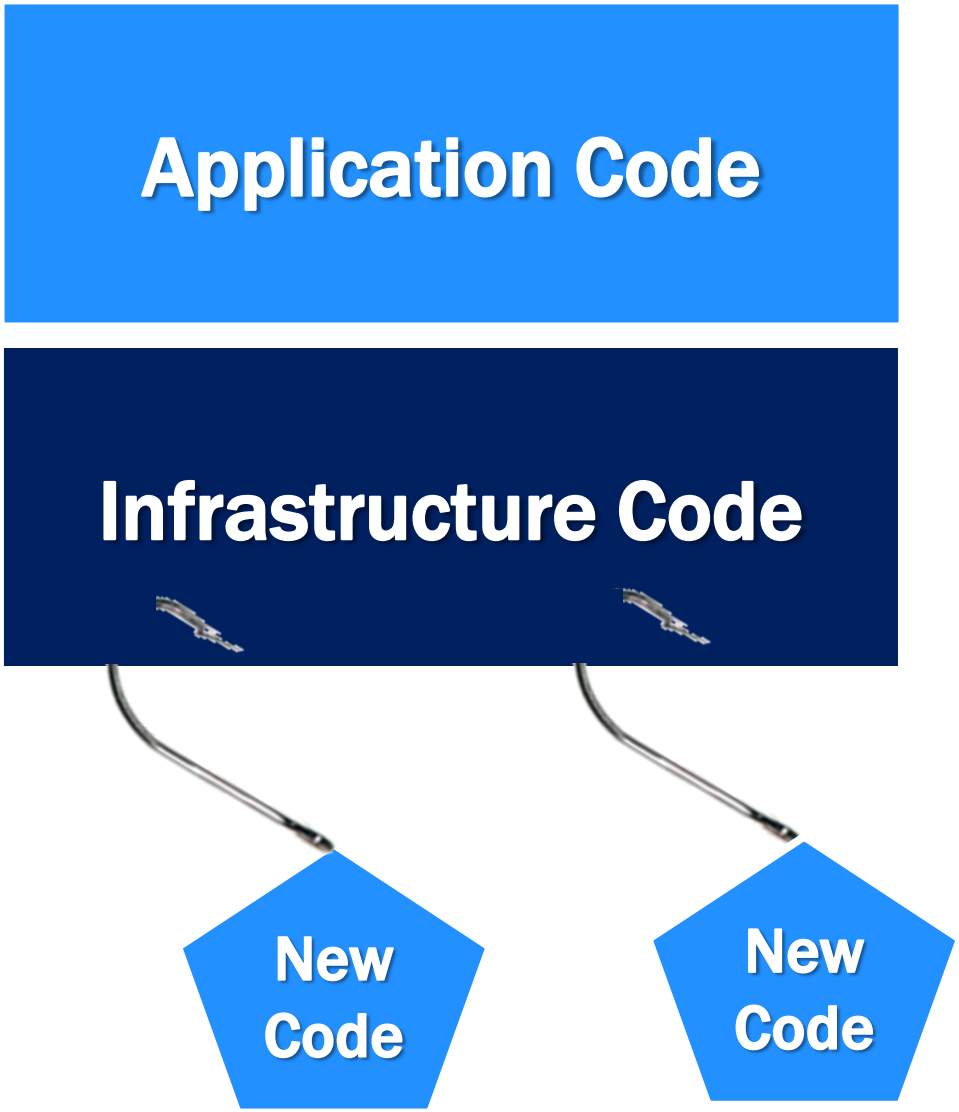
You Aren't Gonna Need It

"Prediction is very difficult, especially if it's about the future."

-- Niels Bohr
Physics Nobel prize 1922

Bolt on flexibility where you need it

Sometimes called “hooks”



Flexibility you seek?

Hmm?

Made your roles explicit, have you?

No?

That is why you fail.

Make Roles Explicit

Some well known interfaces

ISerializable

Java

Serializable

IFather

IHusband

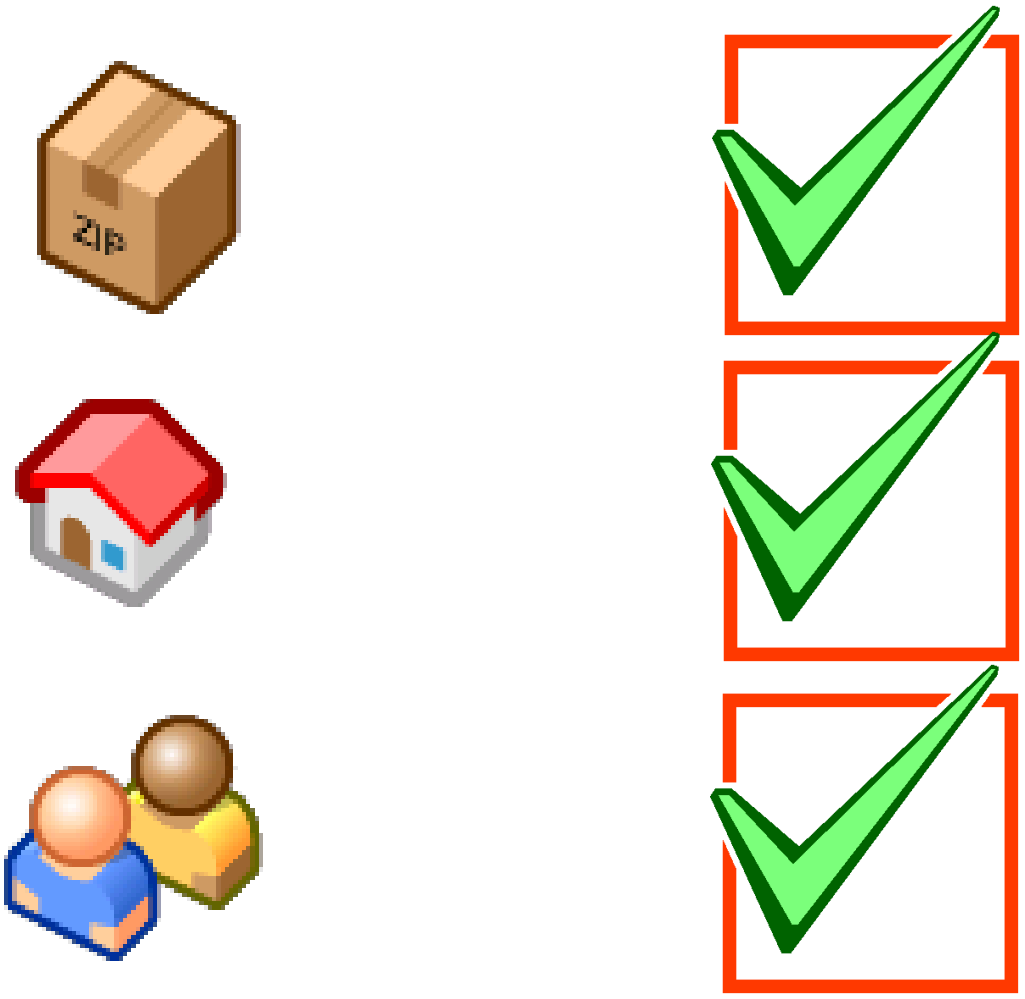
IGoToWork

IComeHome

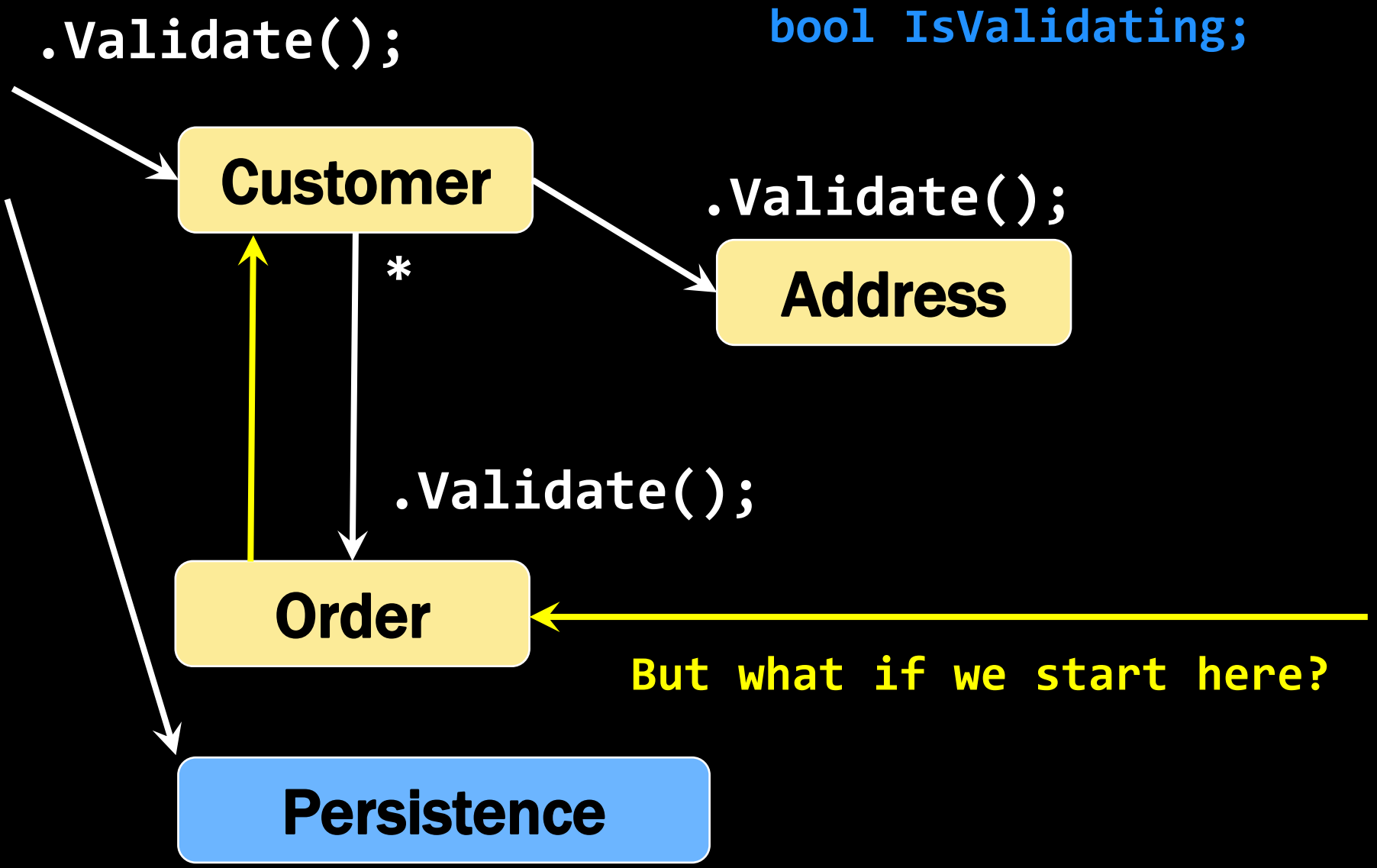
IWashTheDishes


Concrete

Custom entity validation before persistence

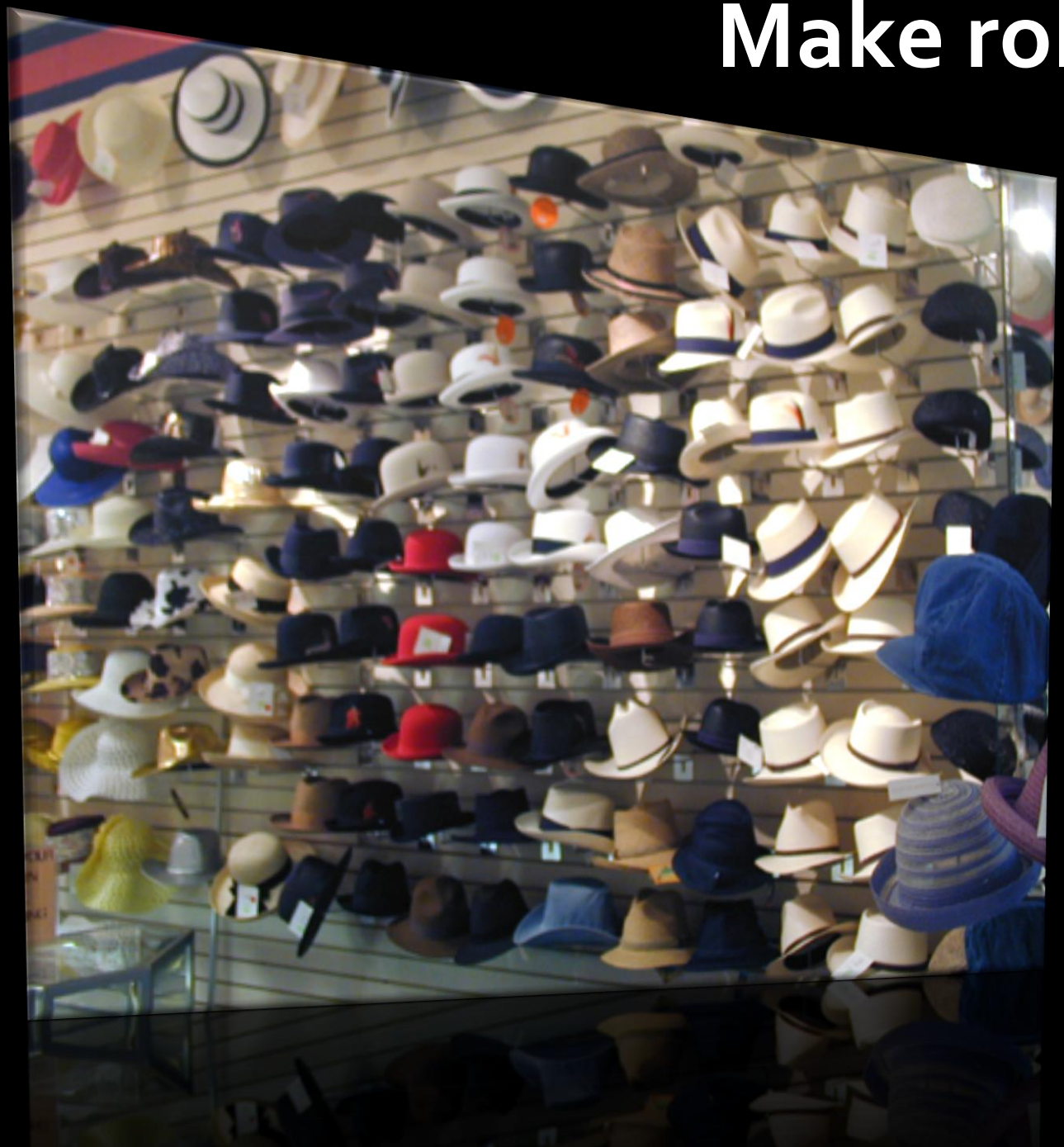


The ~~old~~ "object-oriented" way

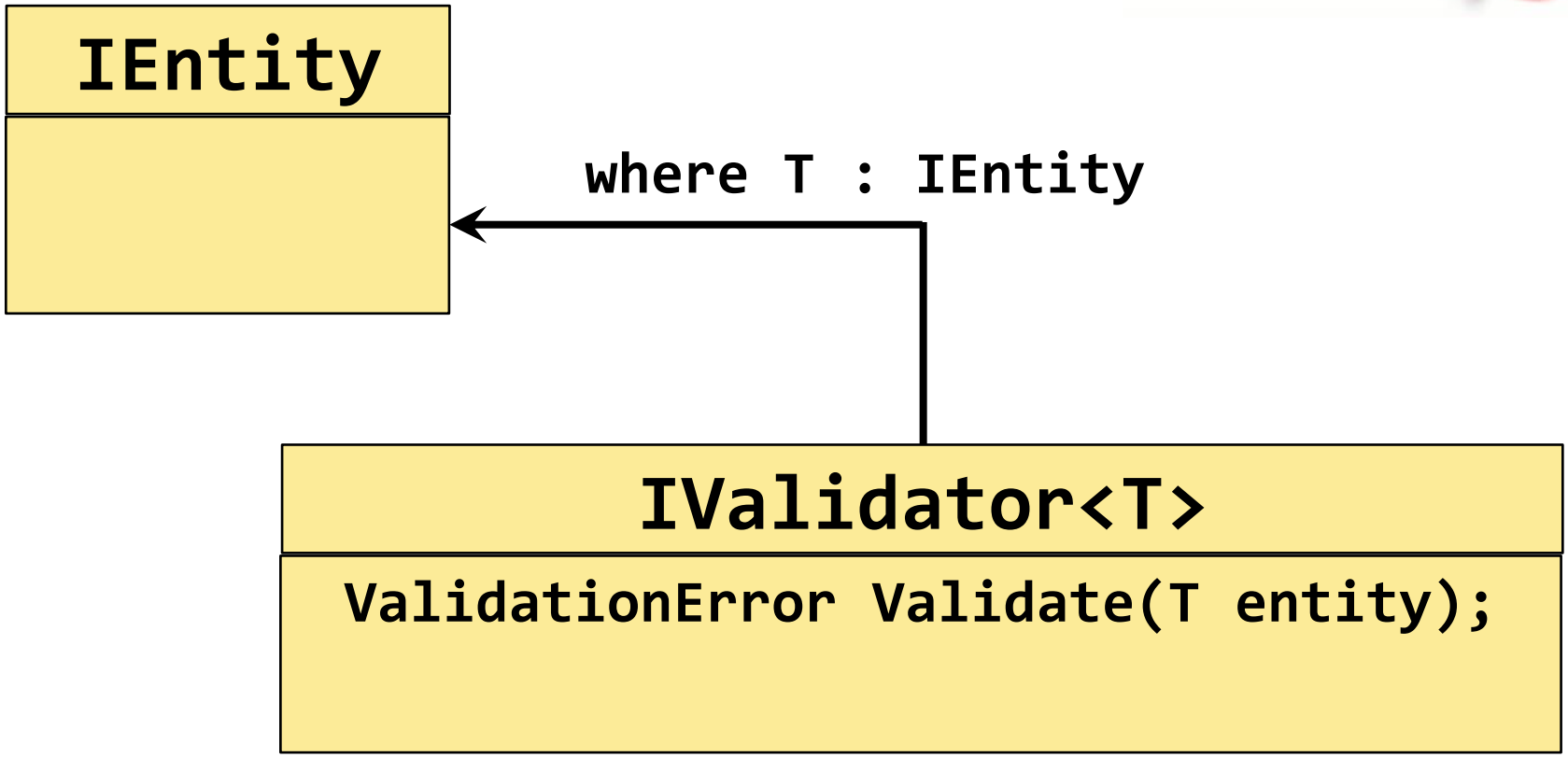


It looks like our objects have too many roles to play

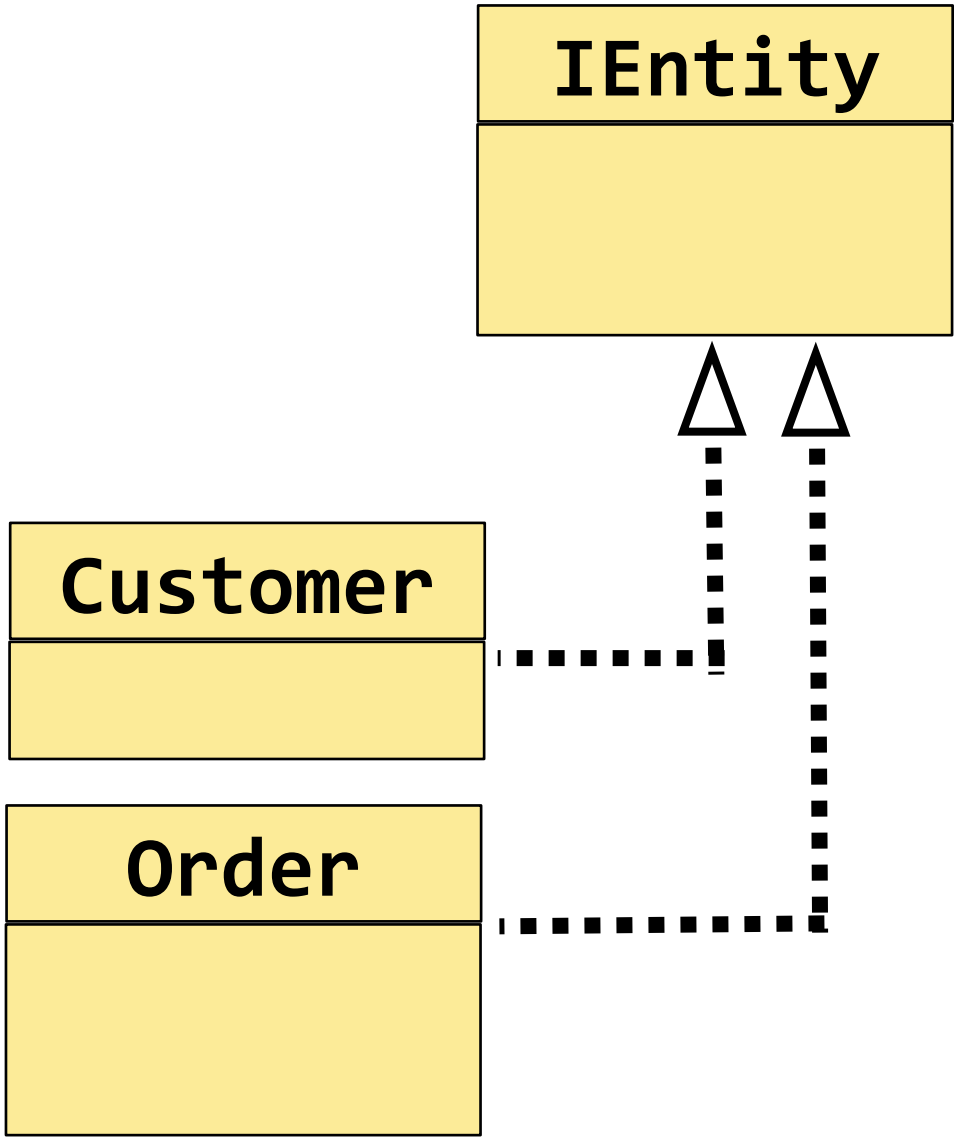
Make roles explicit



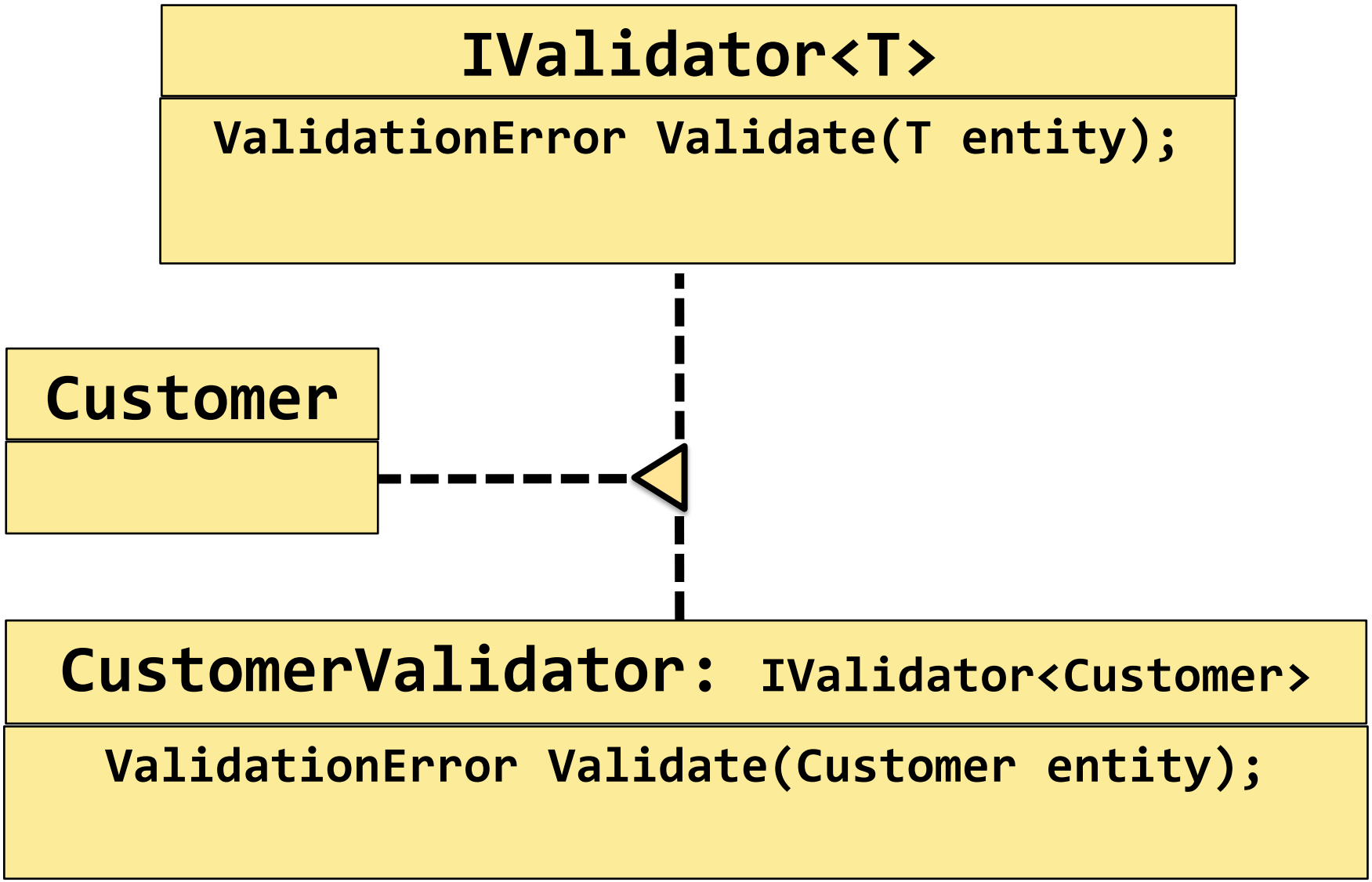
Add a marker interface here, and an interface there….



The first part is trivial:



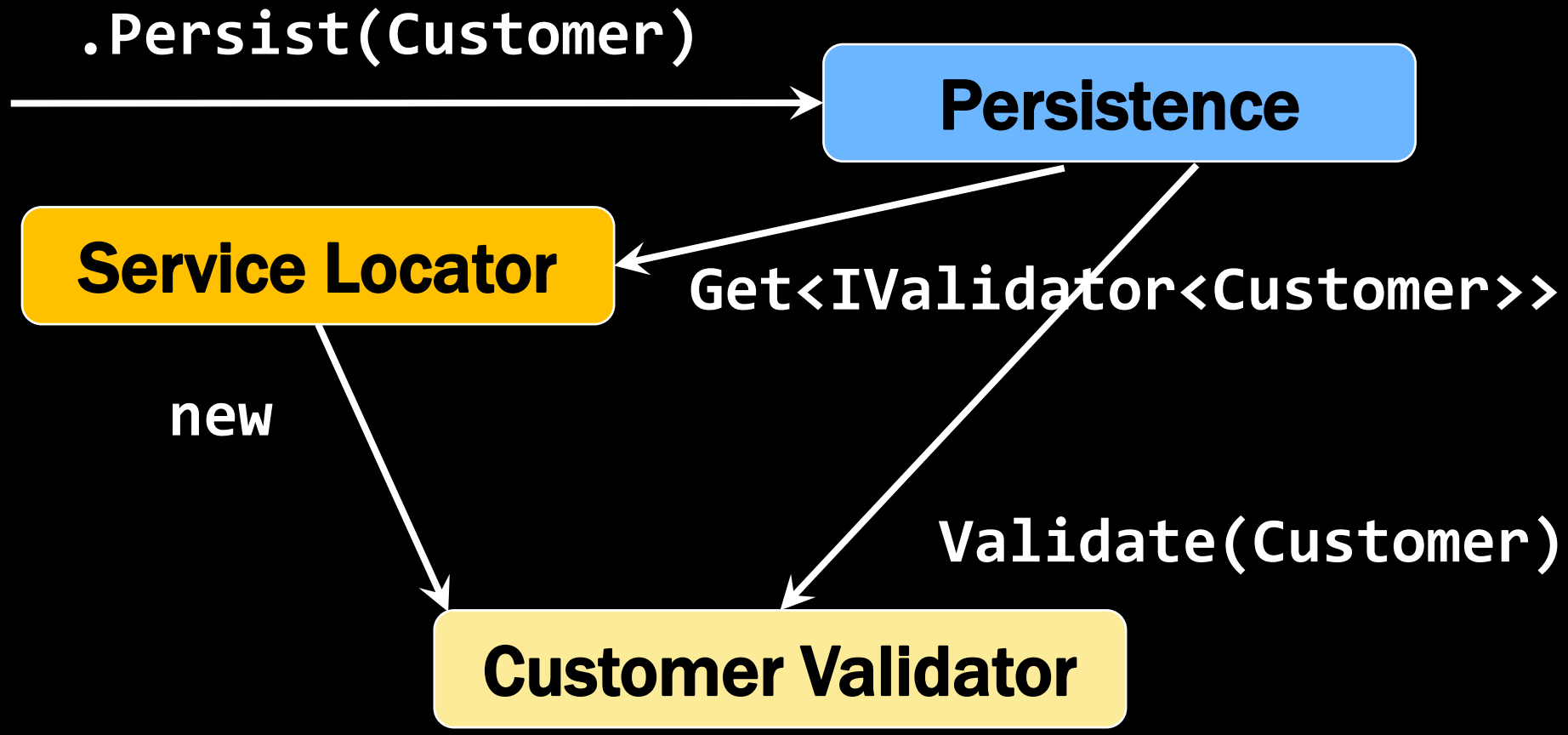
The second part is more interesting

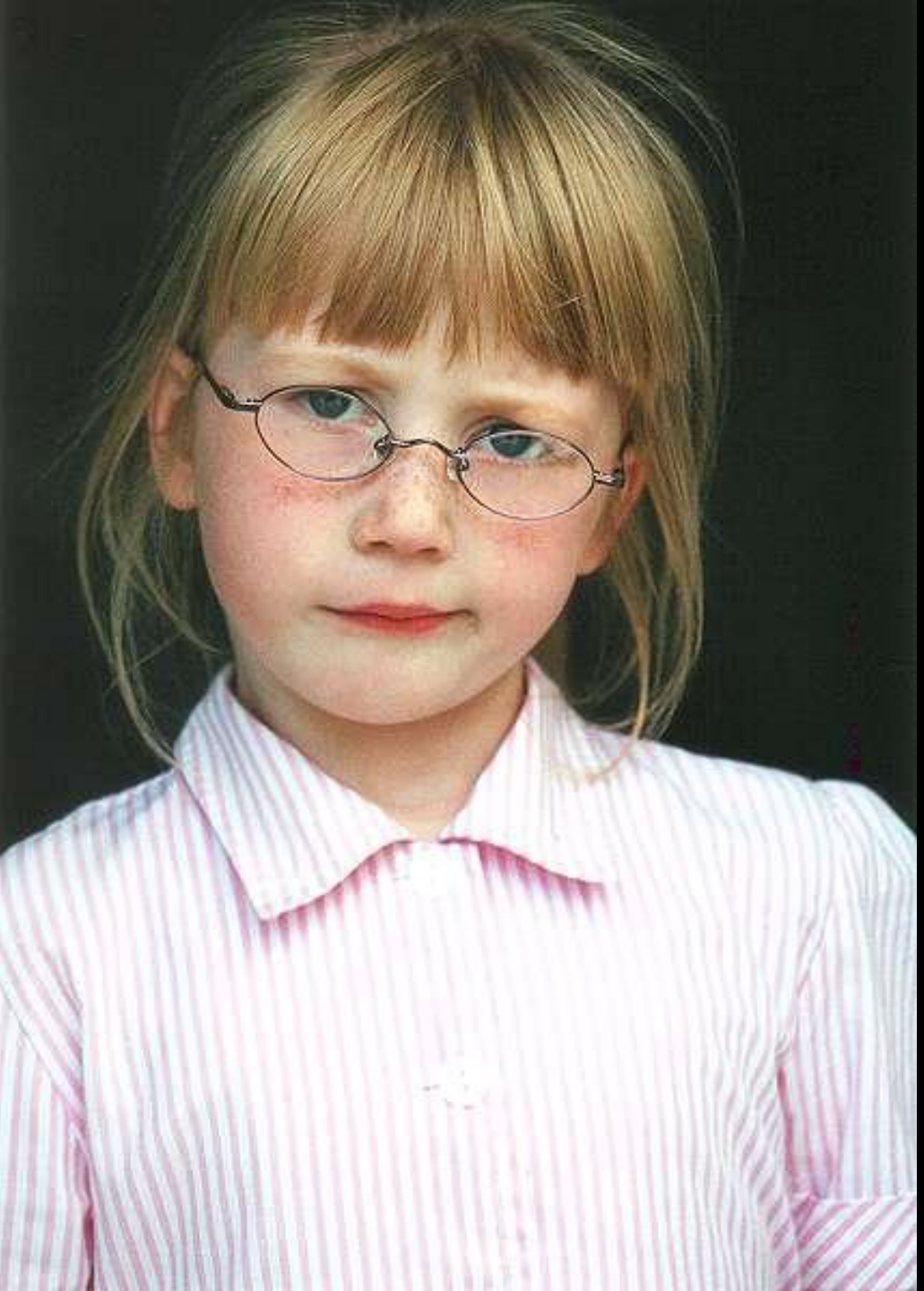


Add a dash of Inversion of Control

Service-Locator style

The extensible way

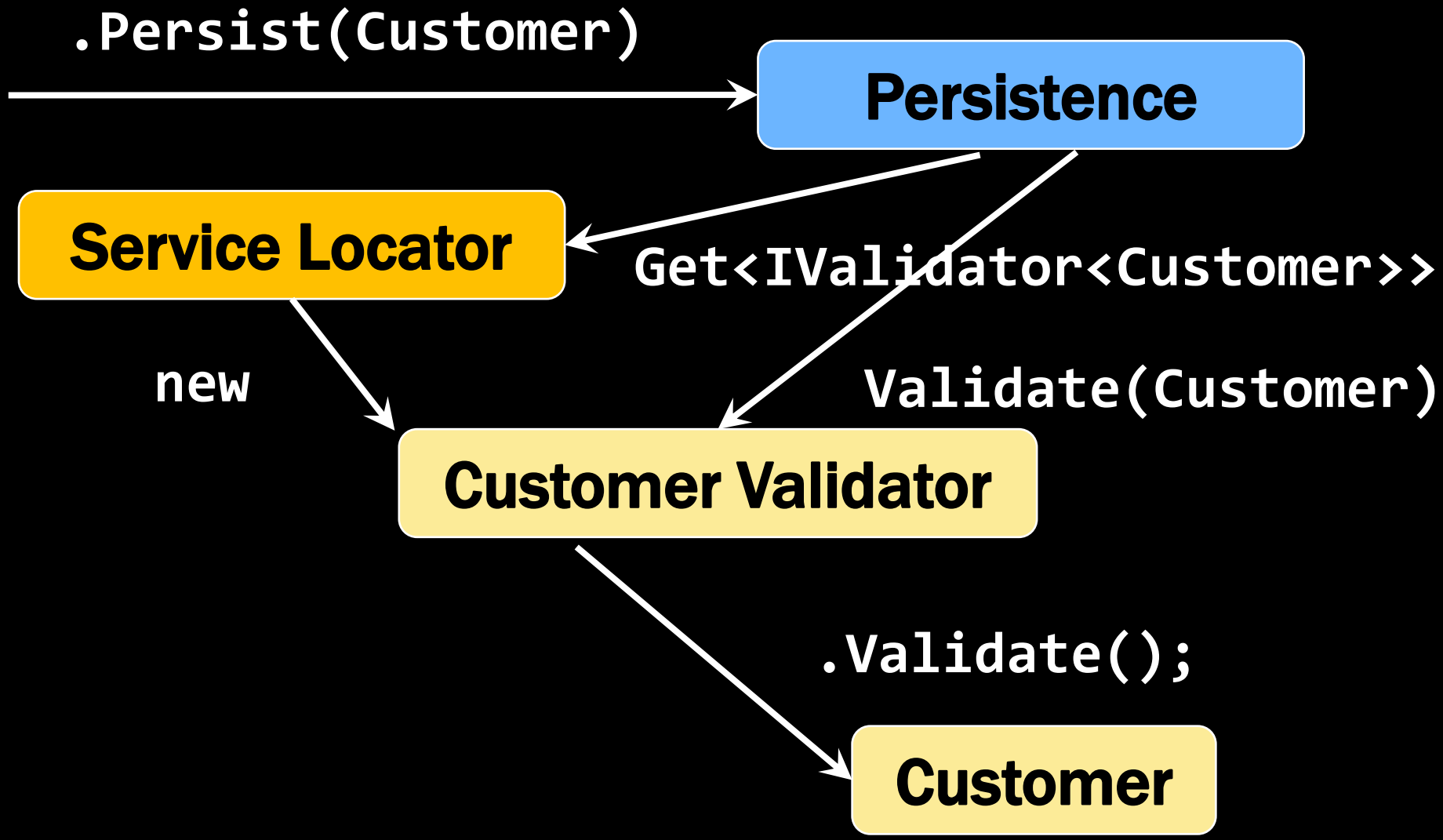




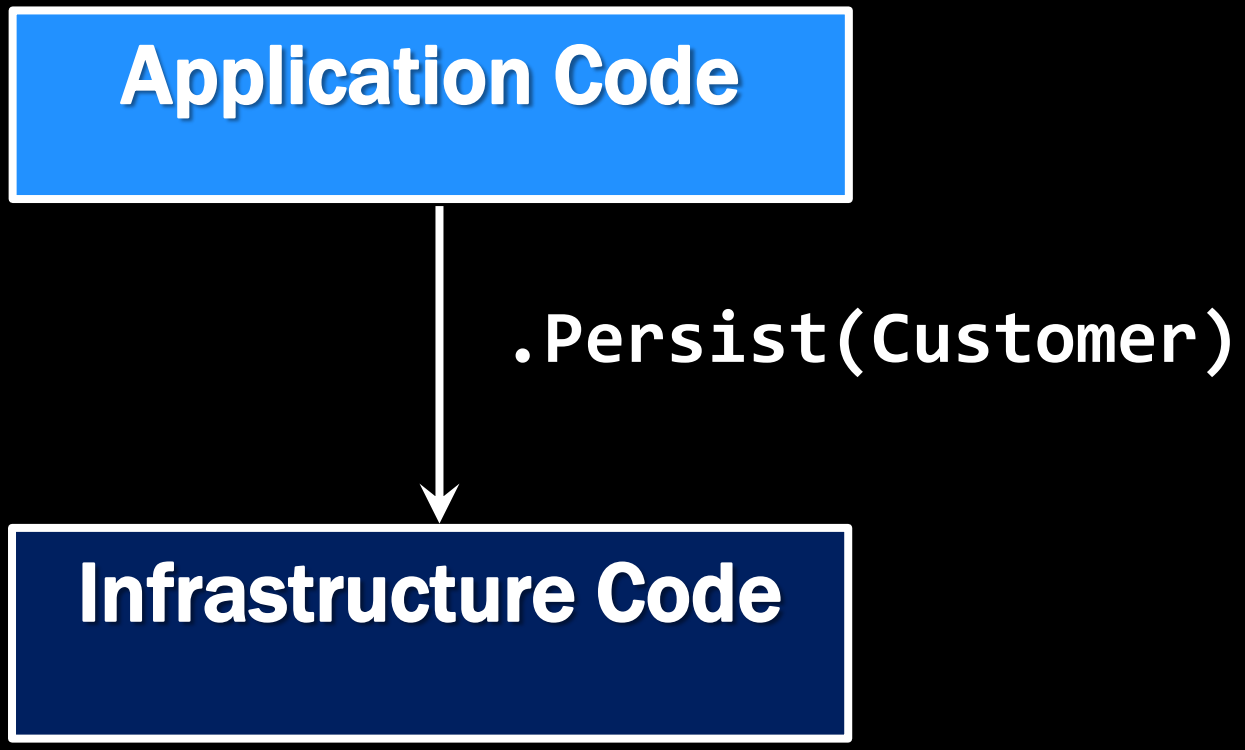
But that's not Object Oriented

Is it?

Extensible and Object Oriented

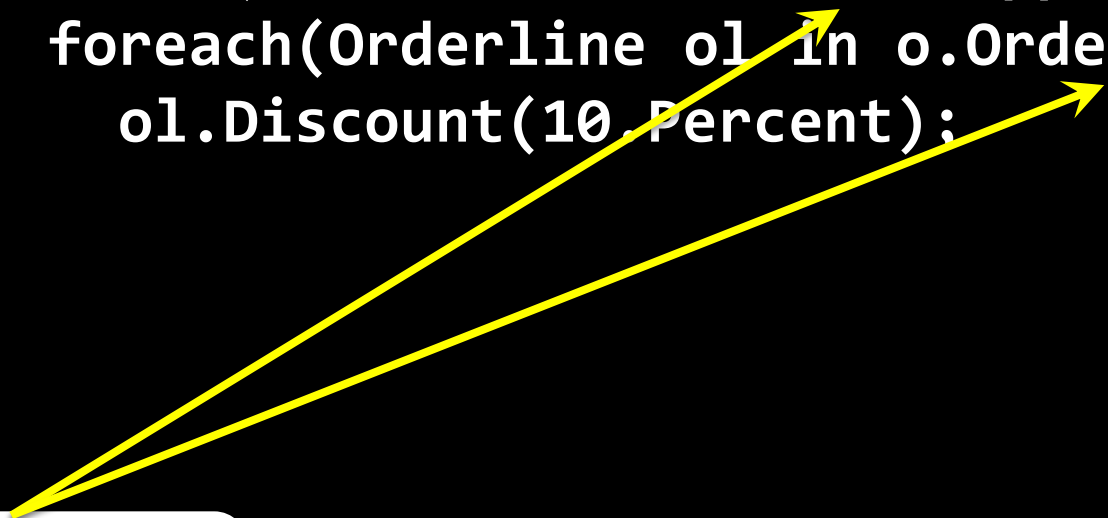


And application code stays simple



Loading objects from the DB

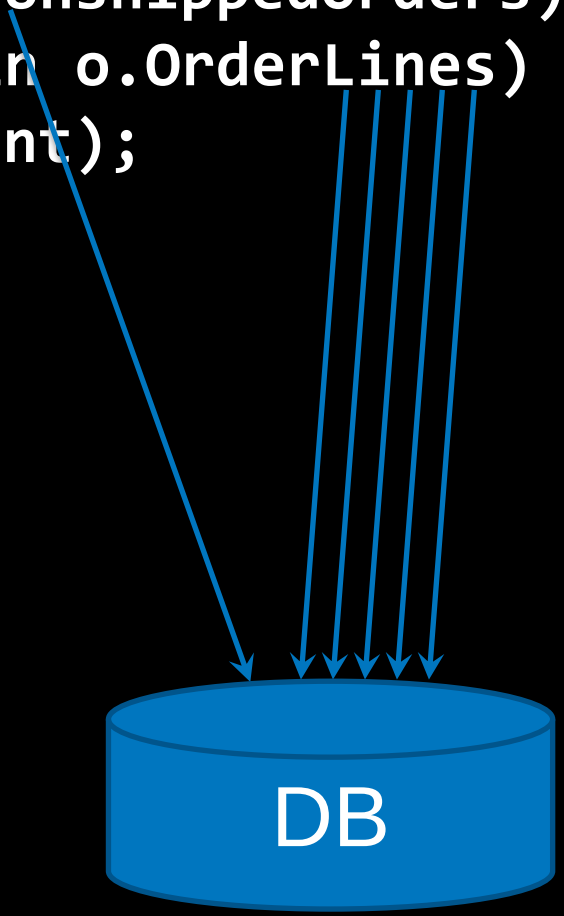
```
public class Customer
{
  public void MakePreferred()
  {
    foreach(Order o in this.UnshippedOrders)
      foreach(Orderline ol in o.OrderLines)
        ol.Discount(10.Percent);
  }
}
```



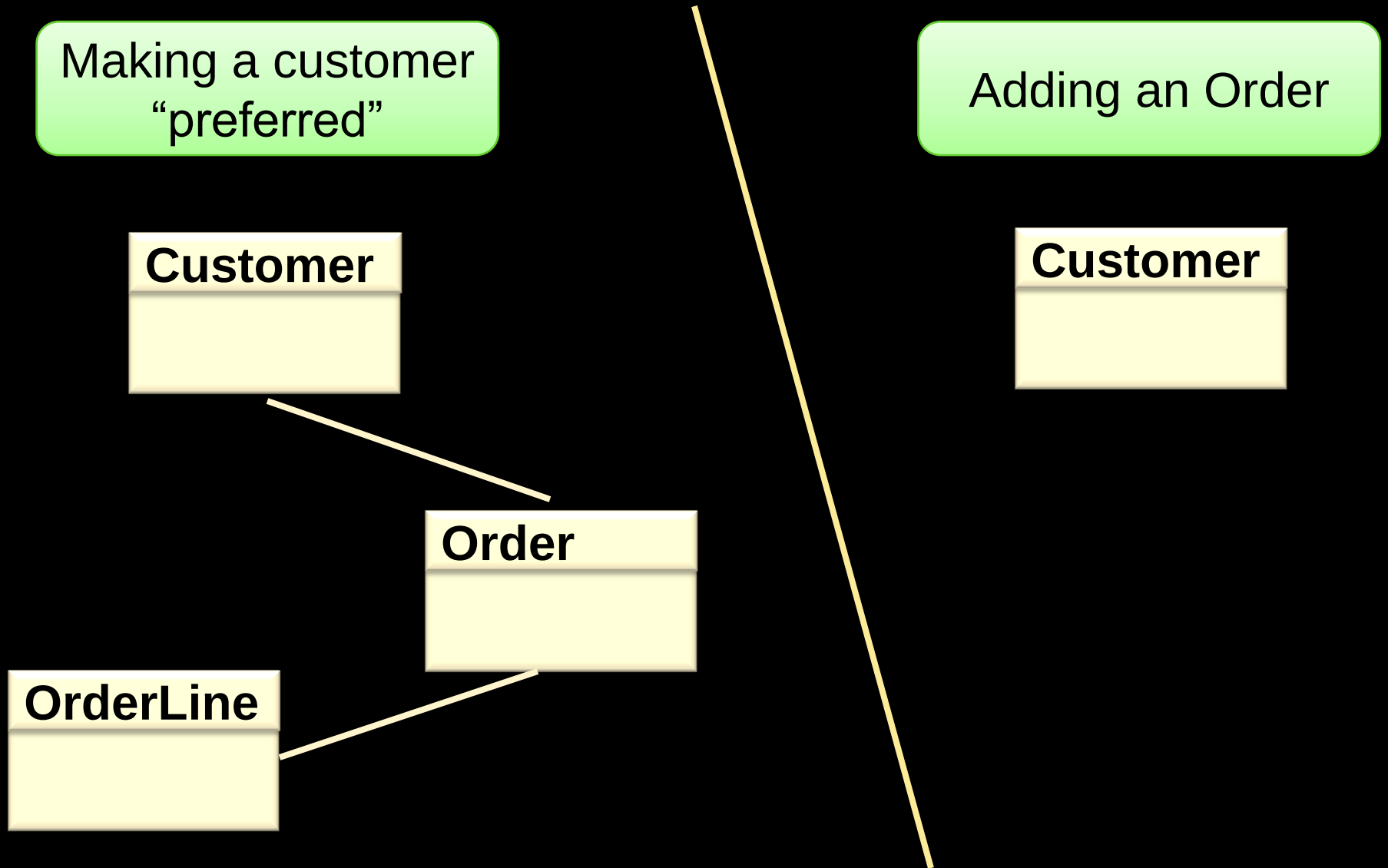
Lazy Loading

Dangers of Lazy Loading

```
public void MakePreferred()
{
  foreach(Order o in this.UnshippedOrders)
    foreach(Orderline ol in o.OrderLines)
      ol.Discount(10.Percent);
}
```



Loading objects from the DB



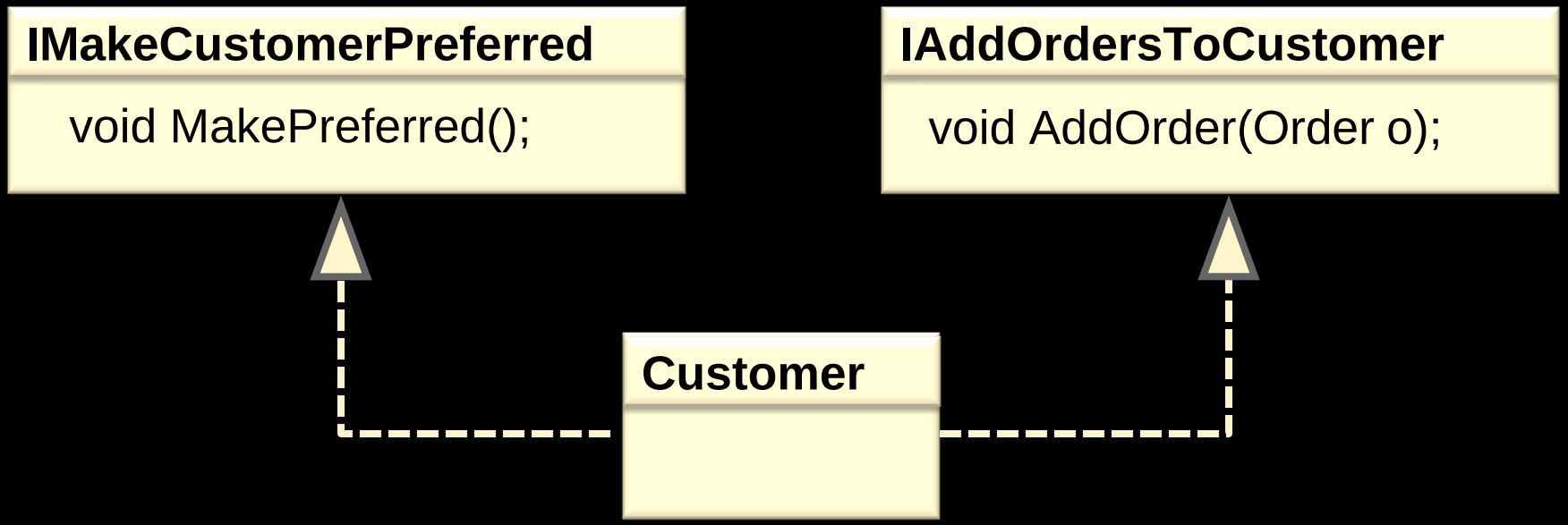
Need Different "Fetching Strategies"

```
public class ServiceLayer
{
  public void MakePreferred(Id customerId)
  {
    Customer c = ORM.Get<Customer>(customerId);
    c.MakePreferred();
  }

 public void AddOrder(Id customerId, OrderInfo o)
  {
    Customer c = ORM.Get<Customer>(customerId);
    c.AddOrder(o);
  }
}
```

Make Roles Explicit

Use interfaces to differentiate roles

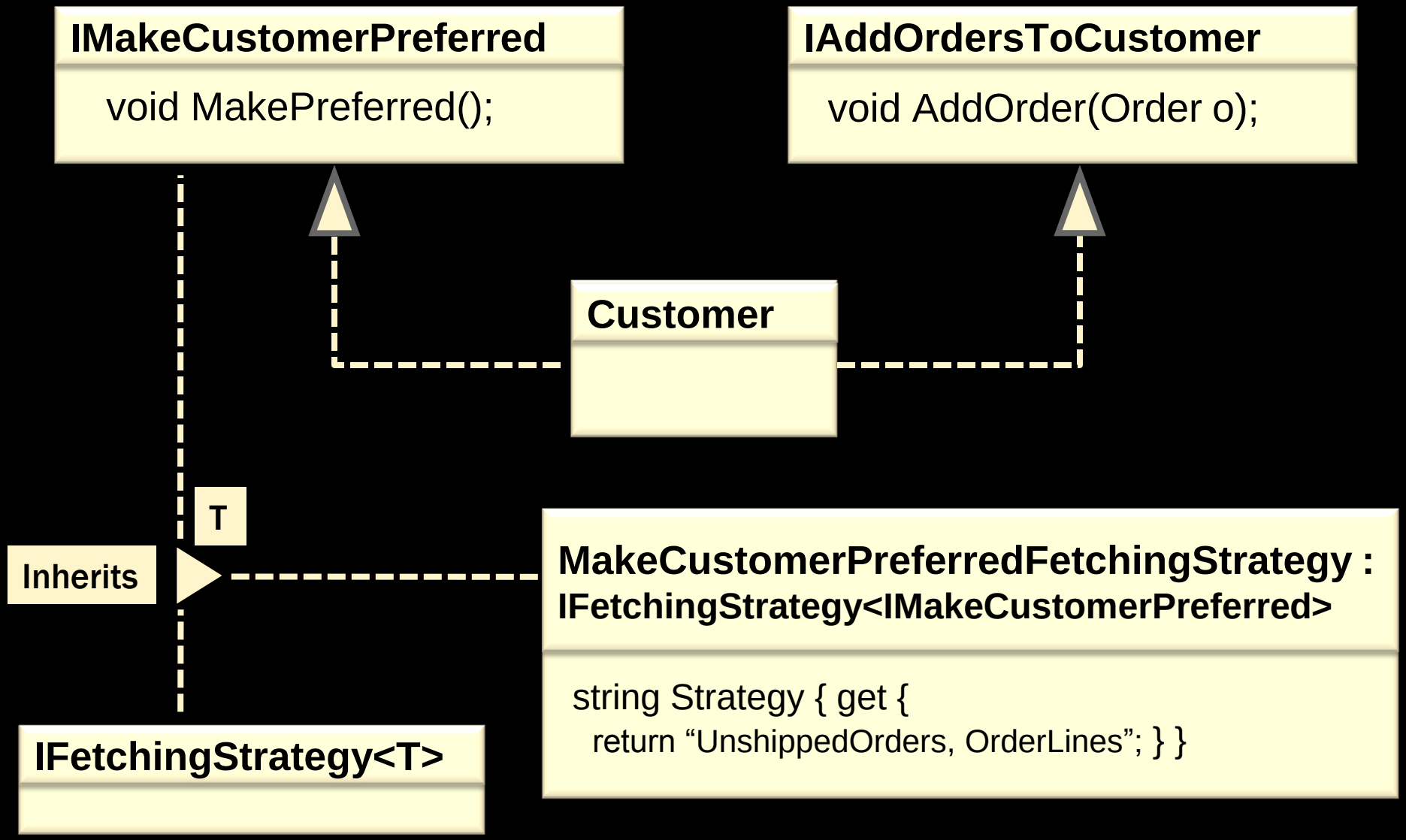


Application code specifies role

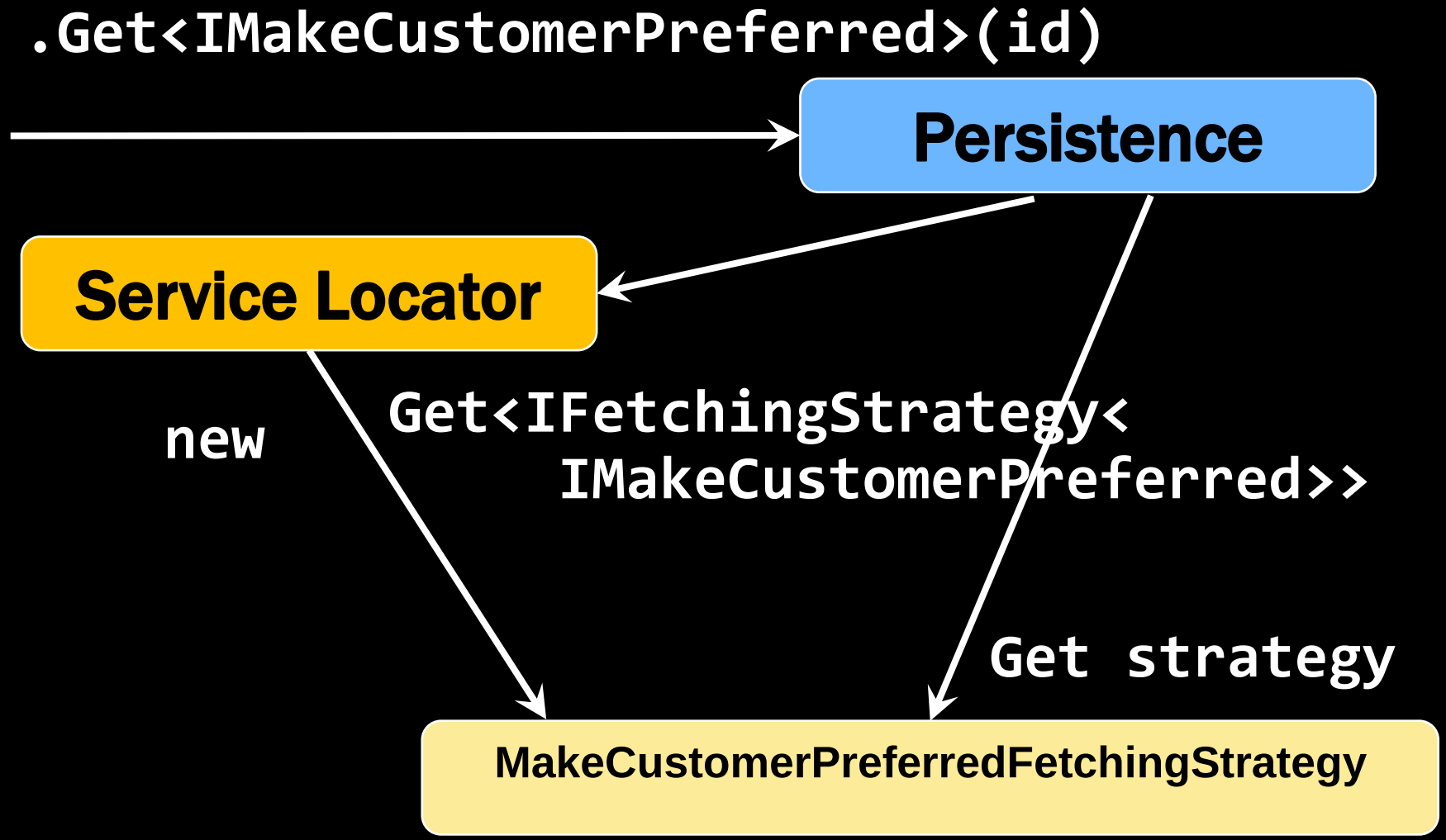
```
public class ServiceLayer
{
  public void MakePreferred(Id customerId)
  {
    IMakeCustomerPreferred c = ORM
        .Get< IMakeCustomerPreferred>(customerId);
    c.MakePreferred();
  }

 public void AddOrder(Id customerId, OrderInfo o)
  {
    IAddOrdersToCustomer c = ORM
         .Get< IAddOrdersToCustomer>(customerId);
    c.AddOrder(o);
  }
}
```

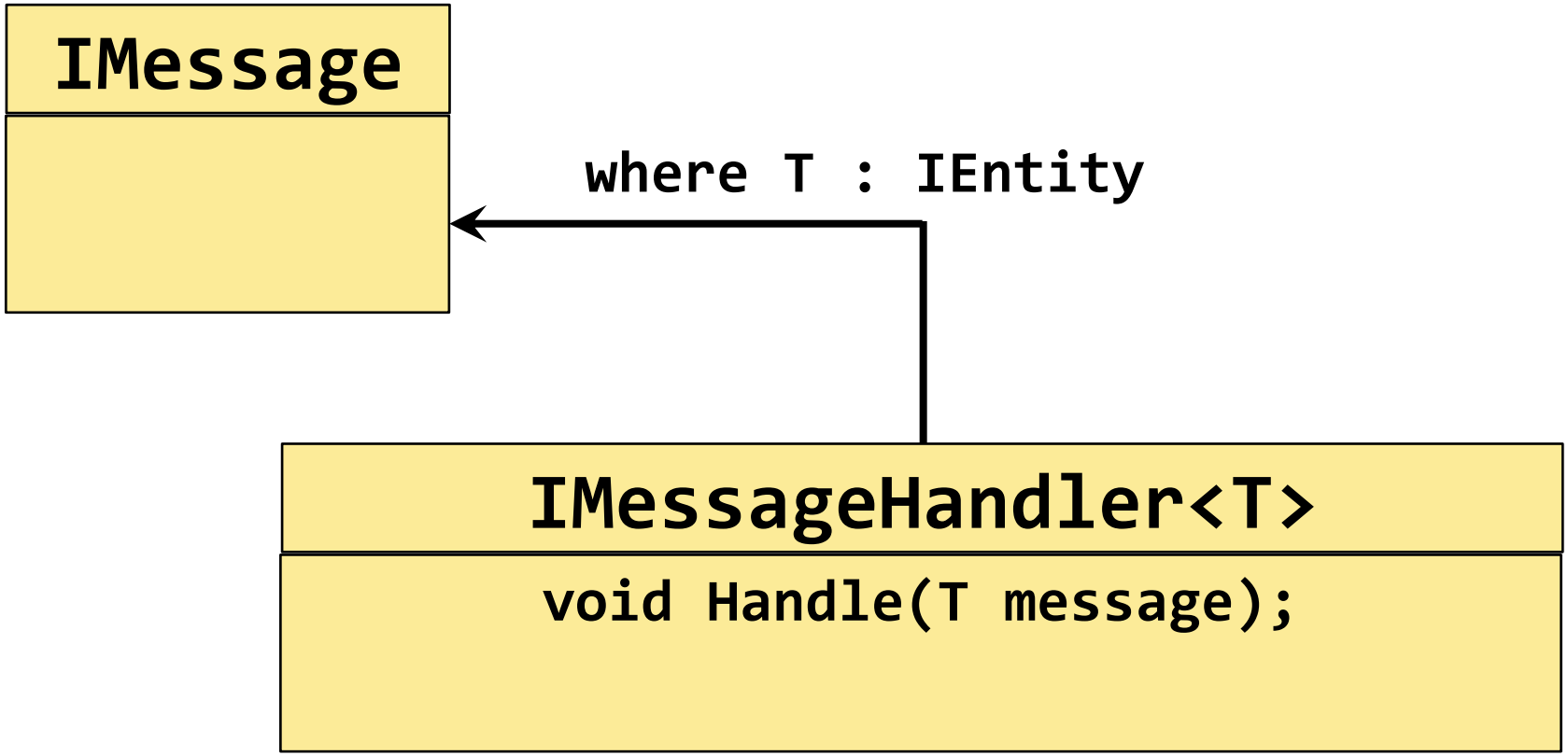
Extend behavior around role



The extensible way



And the pattern repeats…



Once your roles made explicit, you have…

Extensibility and flexibility - simple they will be

Thank you

Udi Dahan – The Software Simplist

.NET Development Expert & SOA Specialist

www.UdiDahan.com

email@UdiDahan.com