Microsoft TechEd Developers

Designing High Performance, Persistent Domain Models [ARC 401]

Udi Dahan – The Software Simplist
.NET Development Expert & SOA Specialist

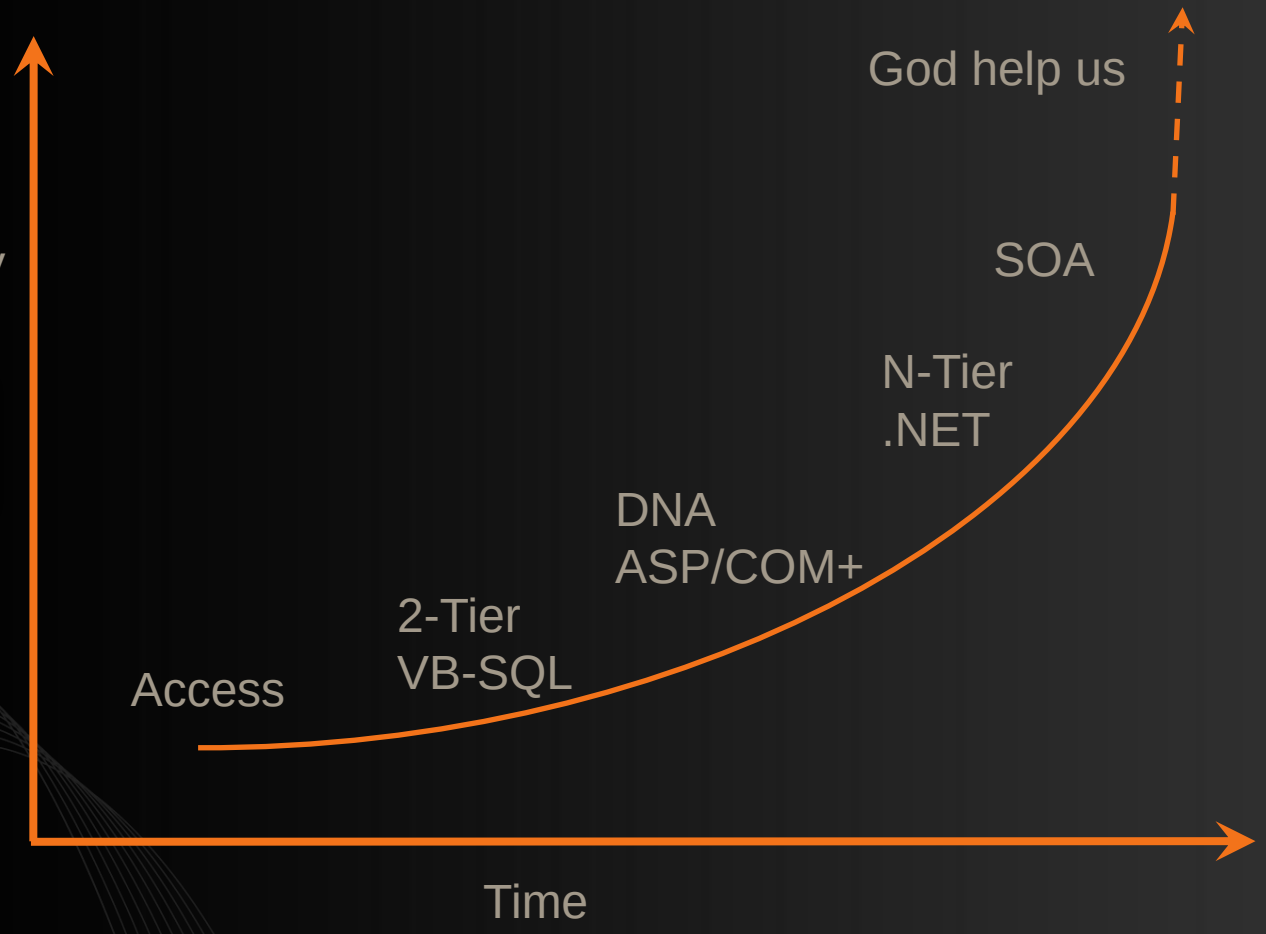
Dr. Dobb's Expert

ineta Speakers Bureau

MVP Microsoft Most Valuable Professional

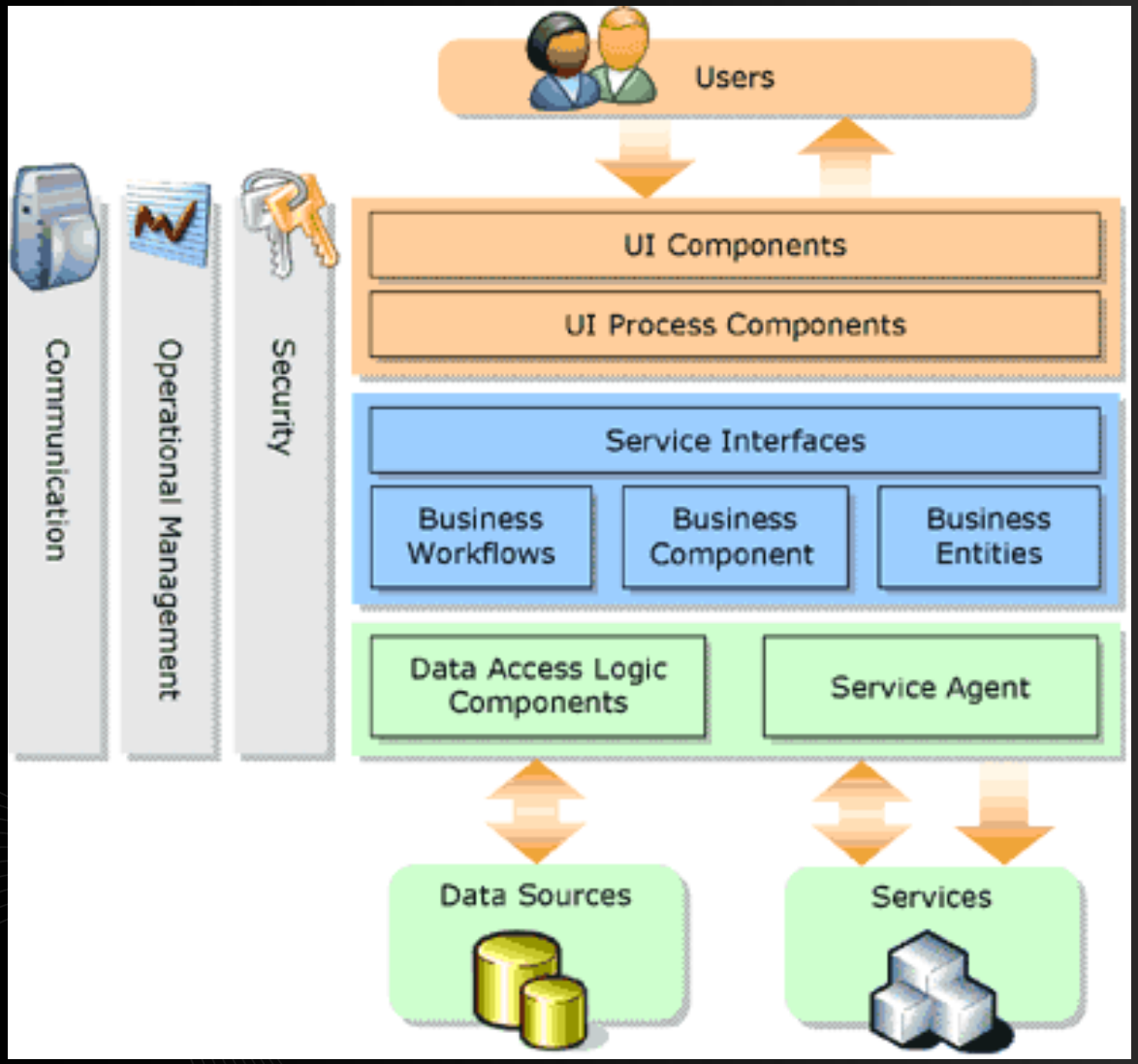
Moore’s law - kinda

You have to deal with twice as much %$&! this year as you did last year

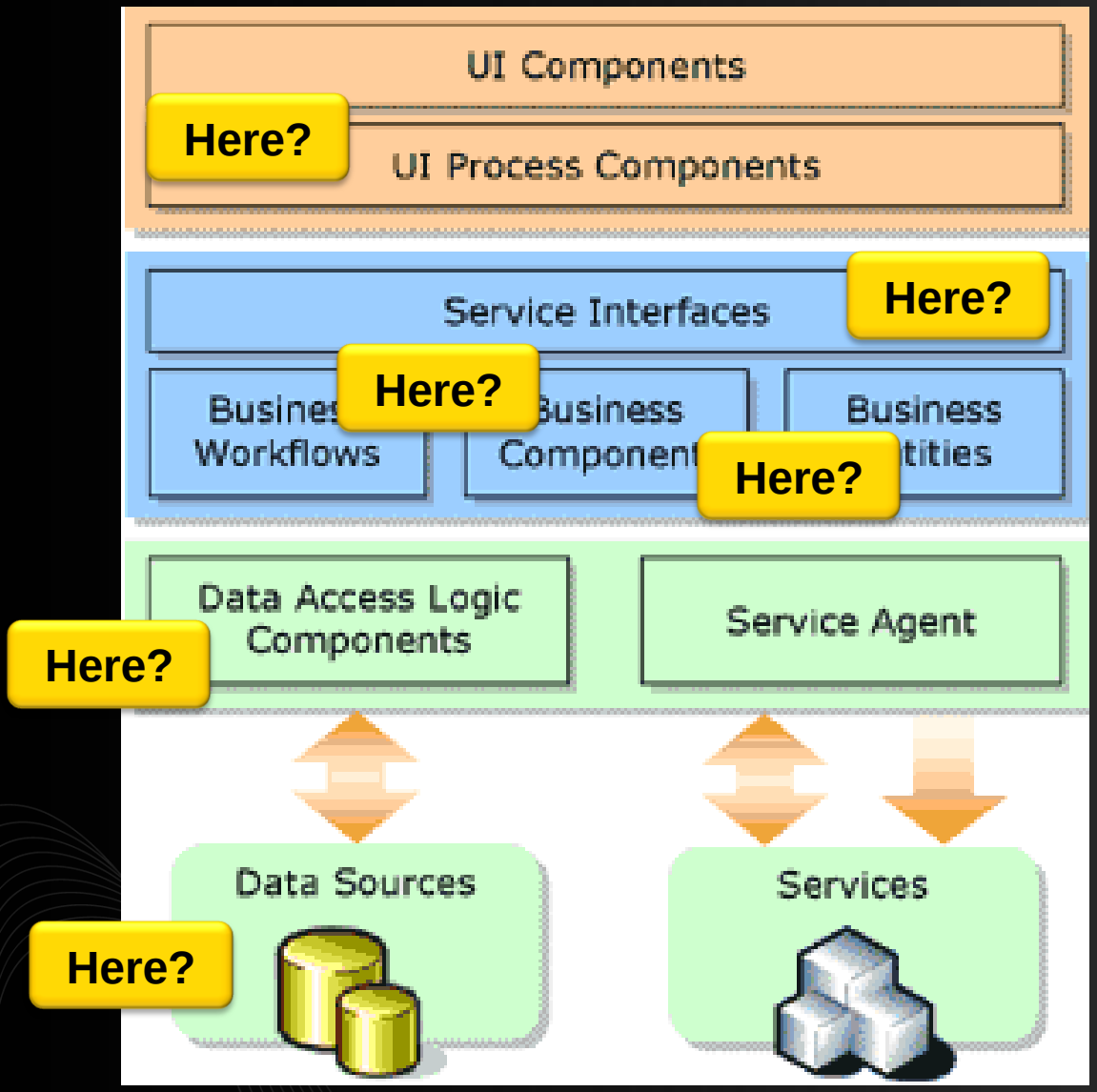


Not only tech
Business logic too

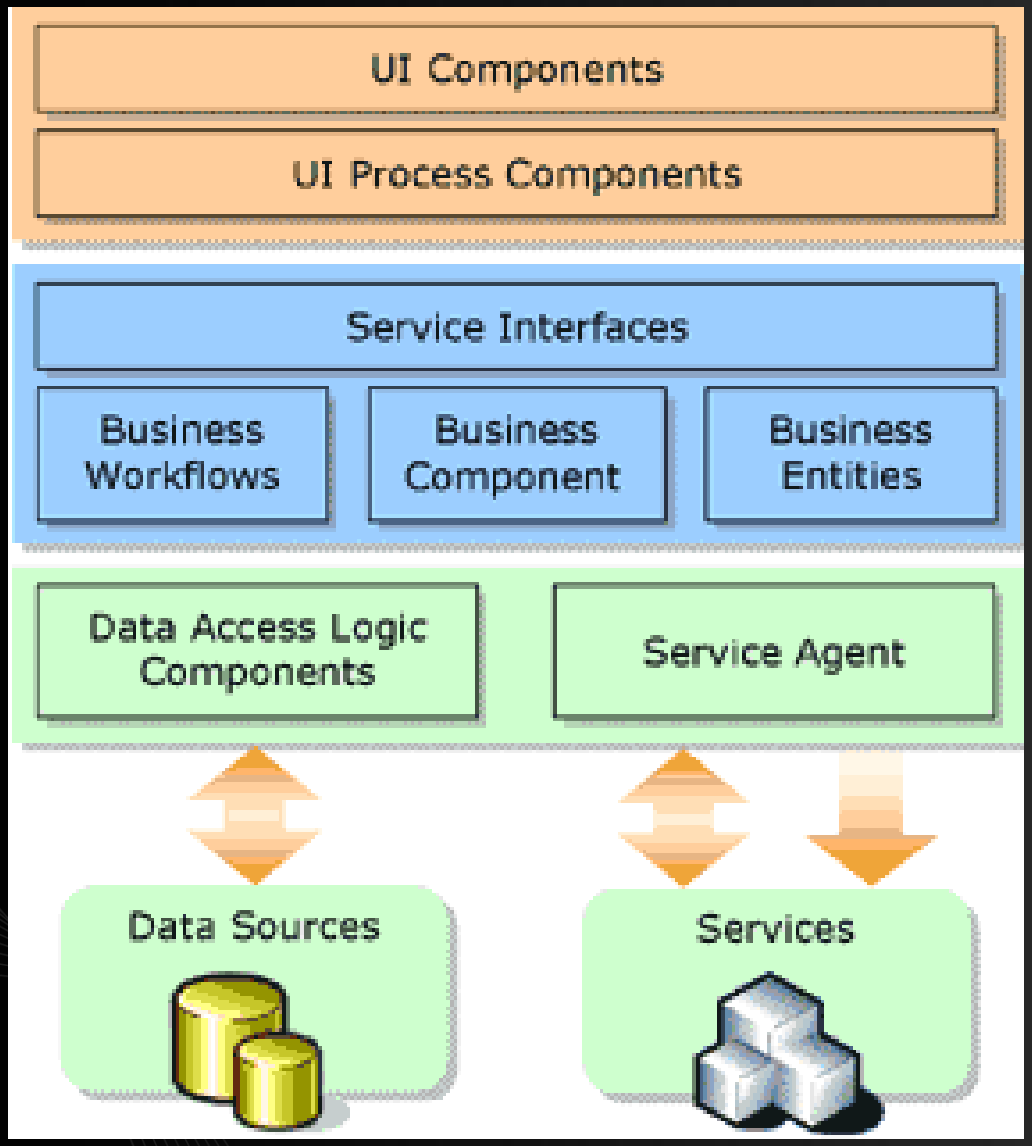
What most architecture looks like today



But where do business rules go?

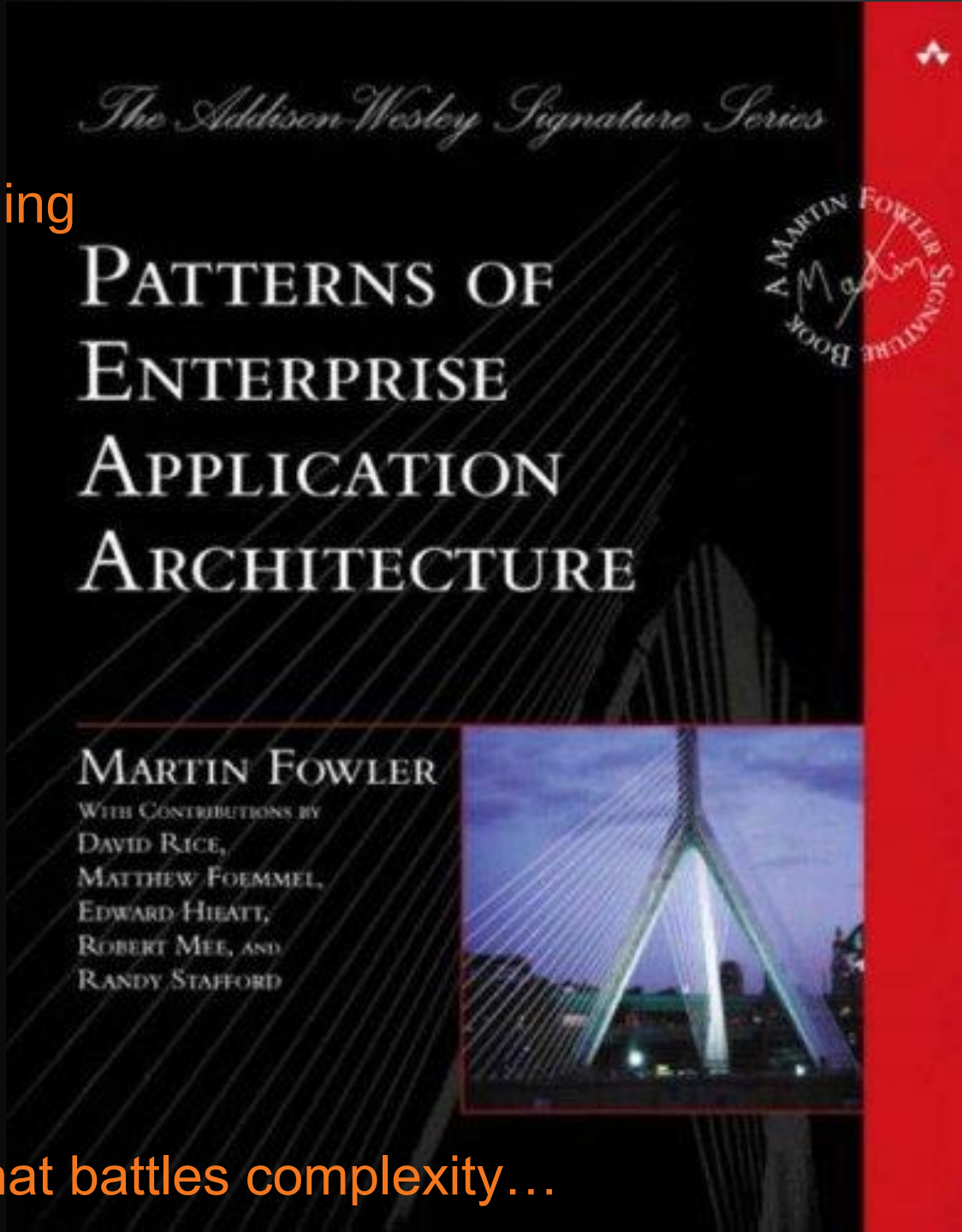


No real reuse between tiers



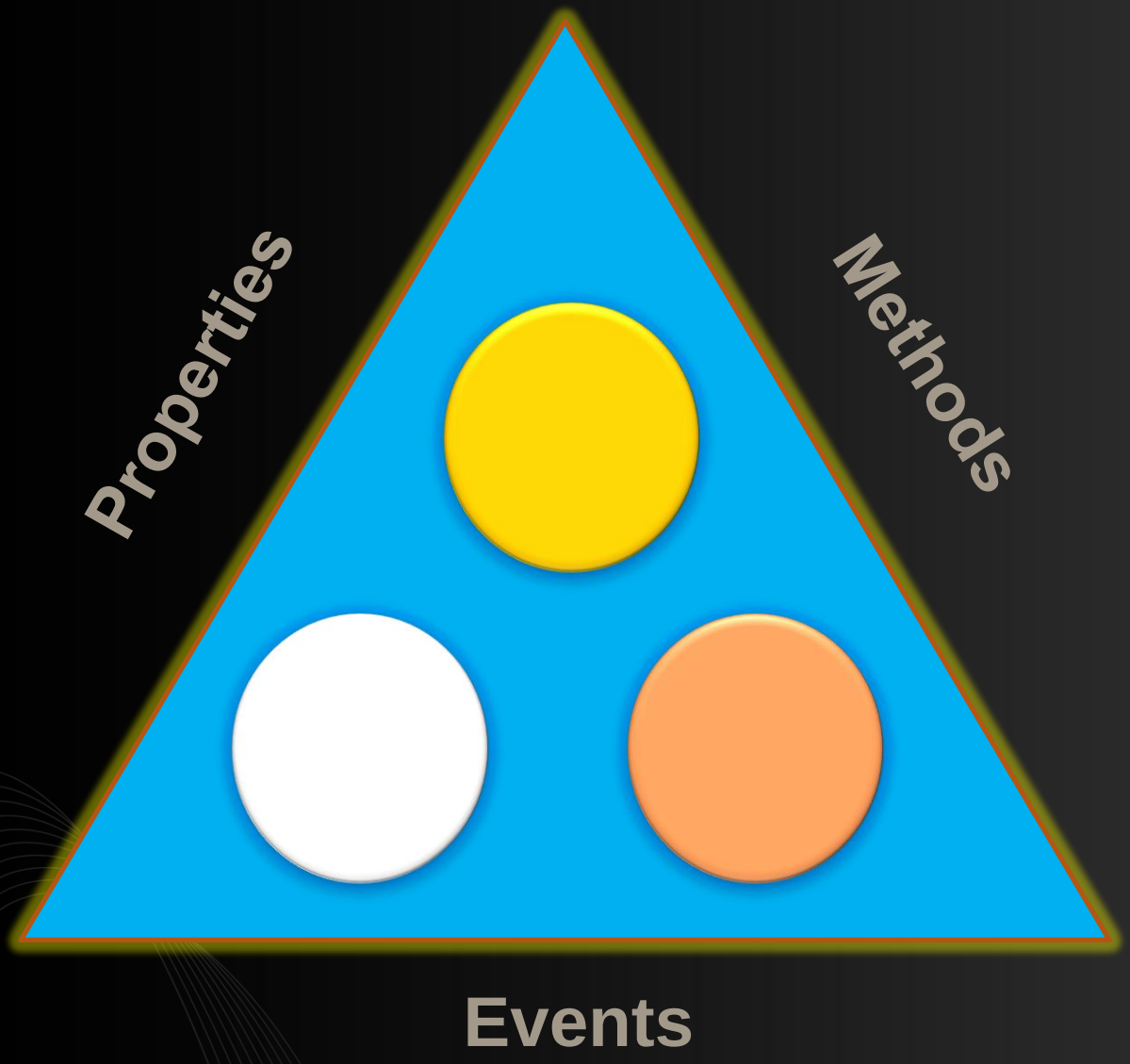
The book

that changed everything

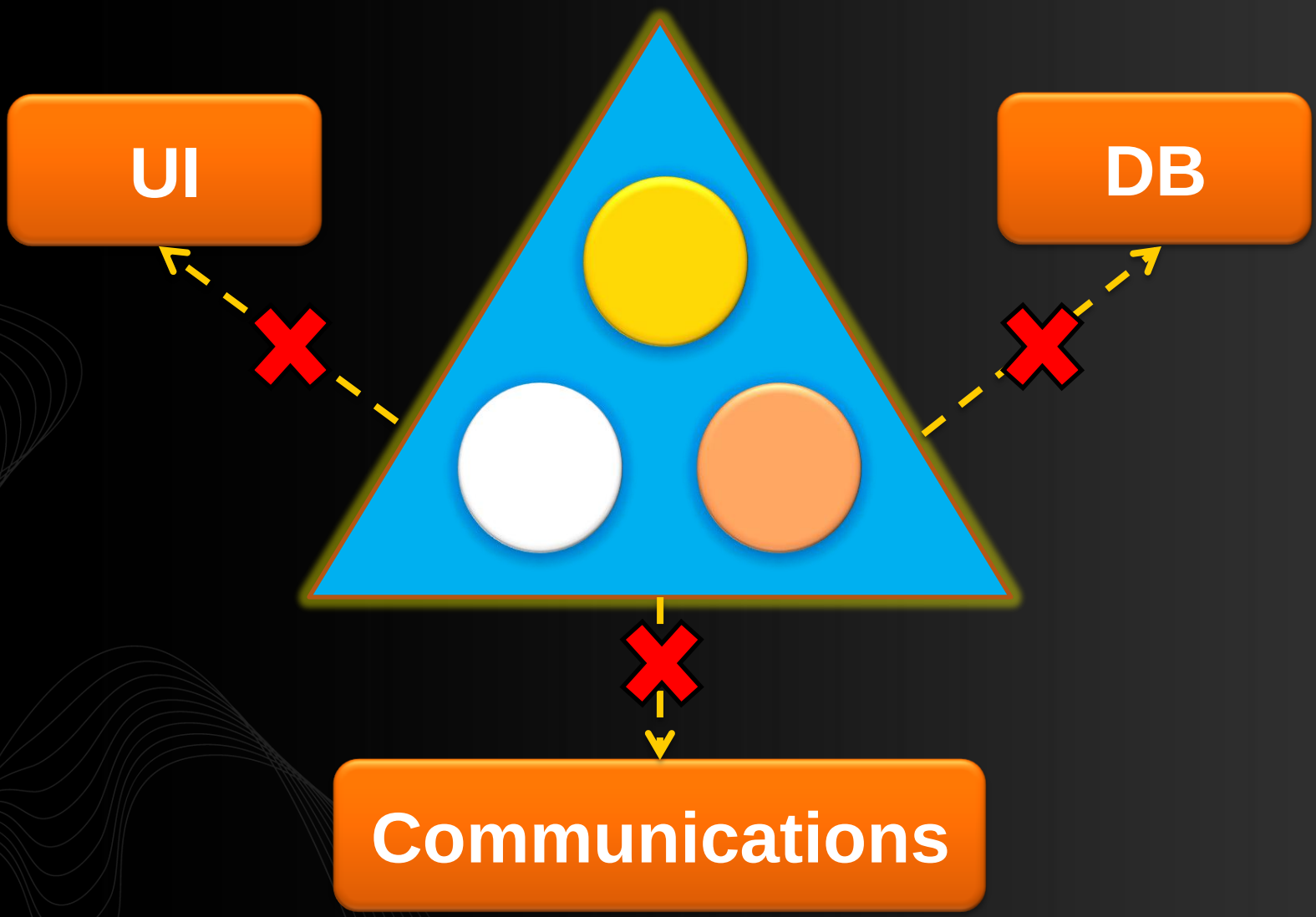


… and the pattern that battles complexity…

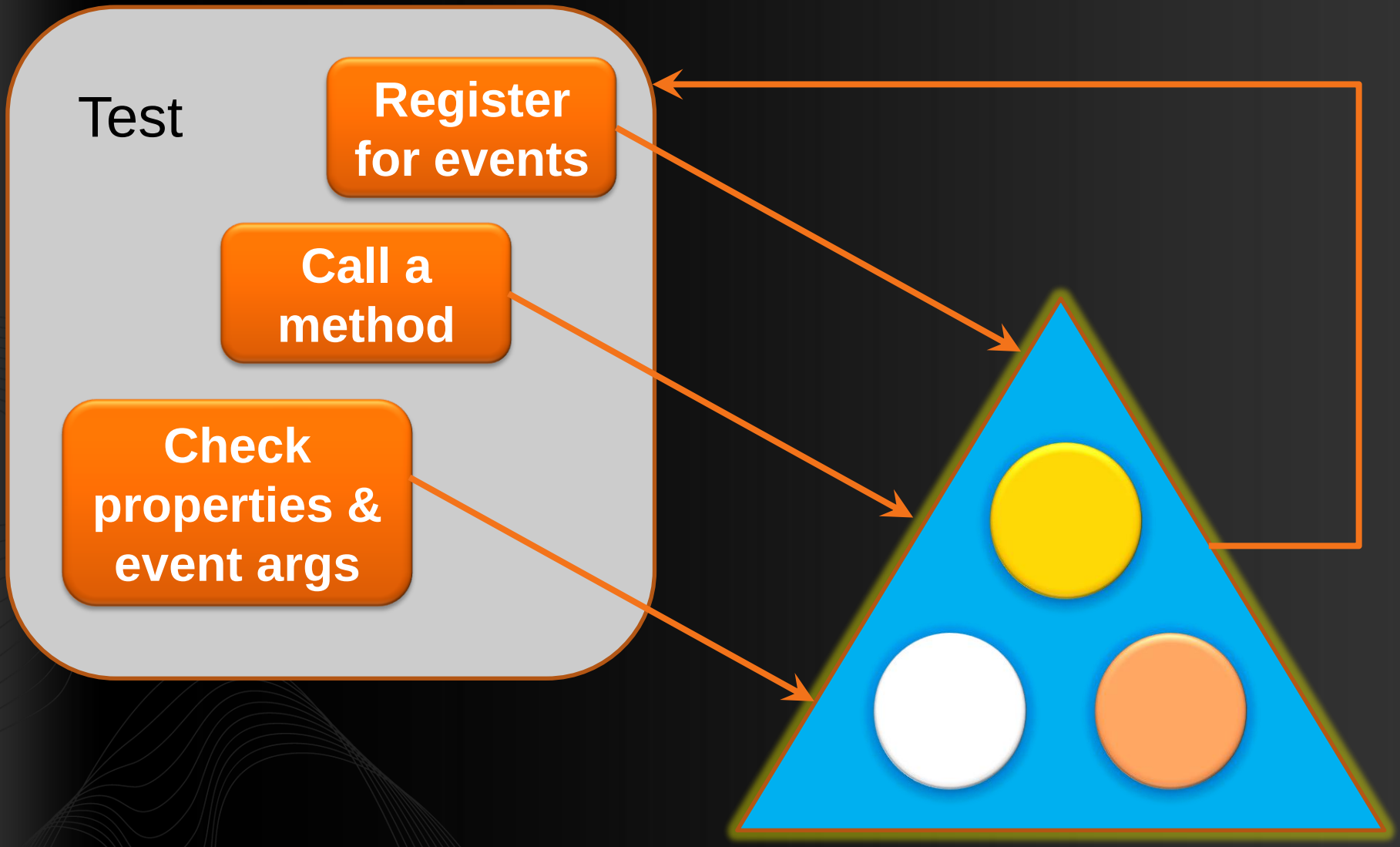
The Domain Model Pattern



Independent of all concerns

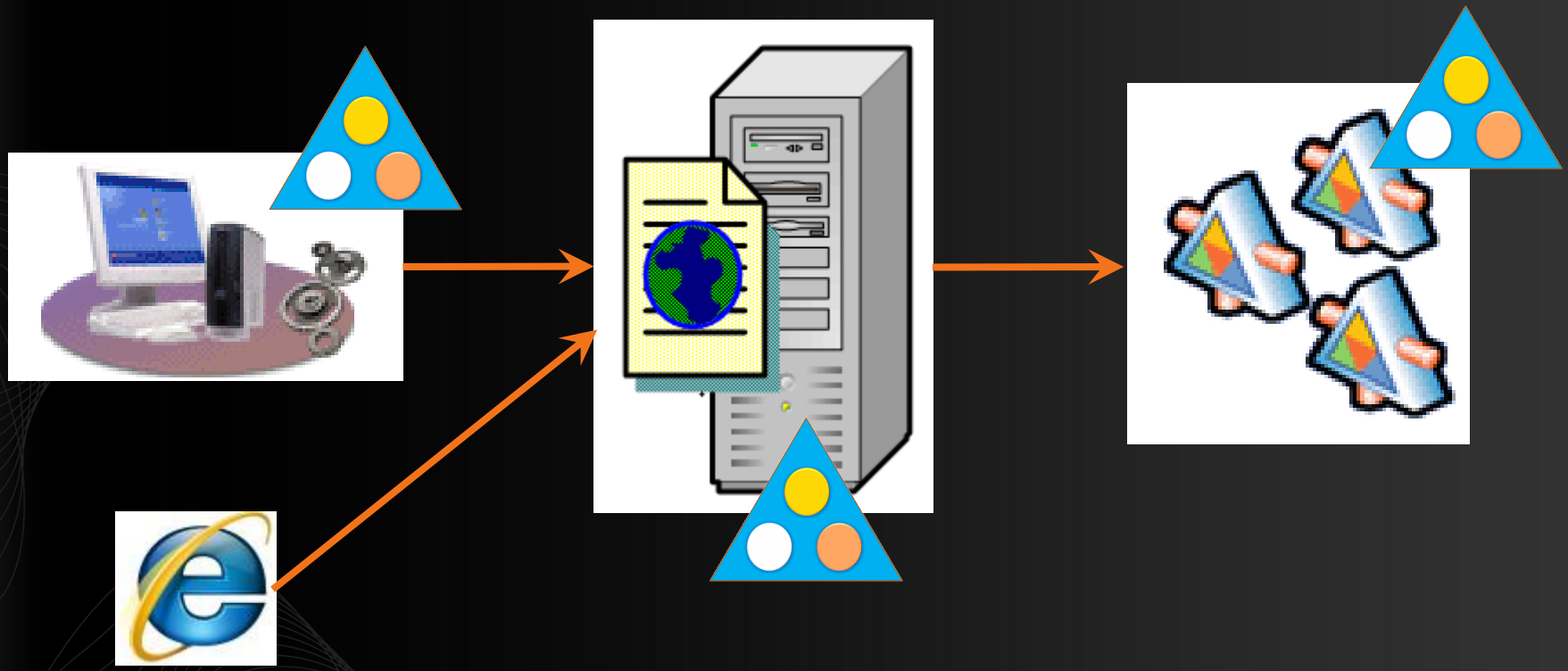


Highly testable

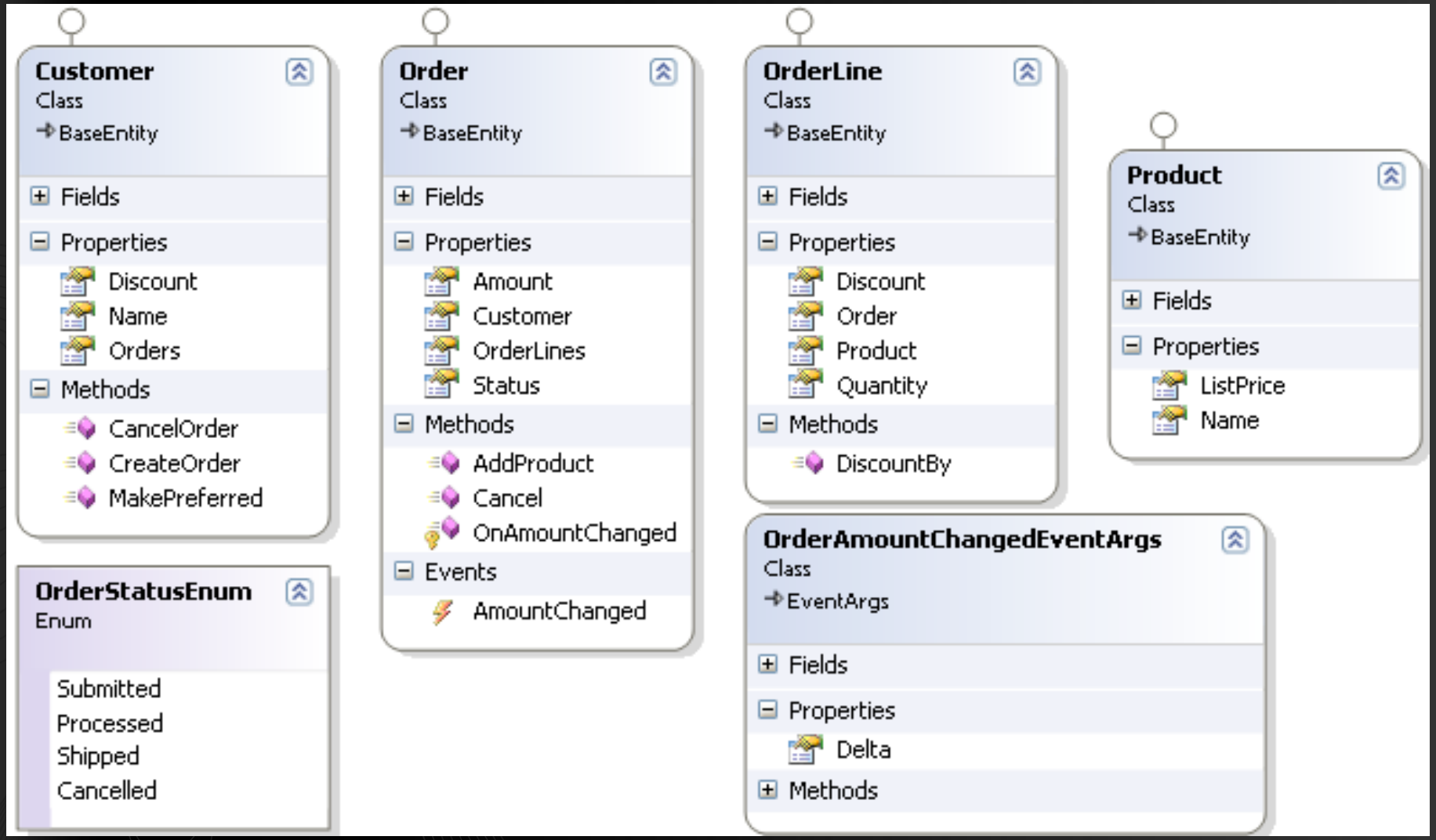


Can be deployed Multi-Tier

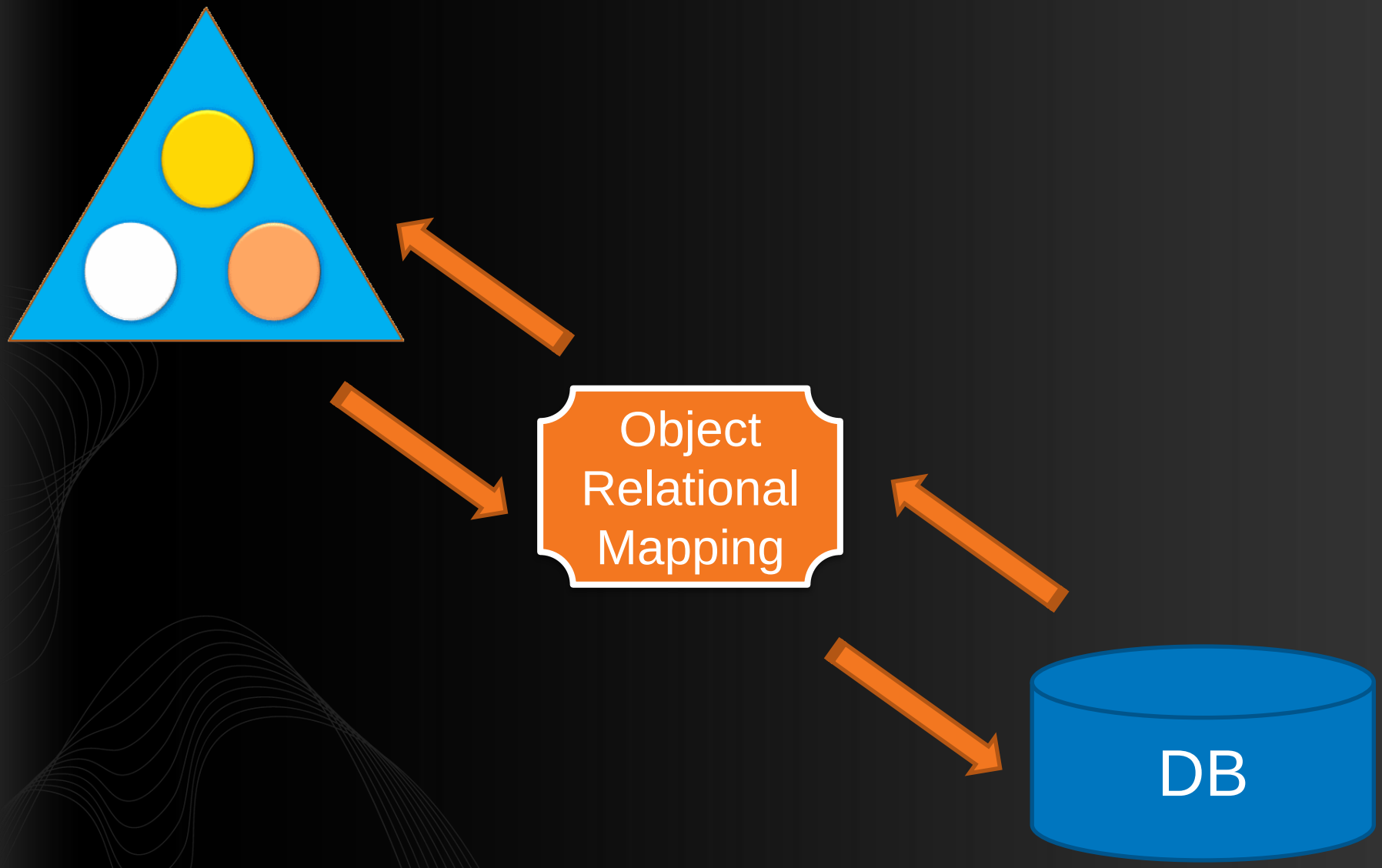
it's just a bunch of DLLs – or even just one



Domain Models are made of "Plain old .NET objects"

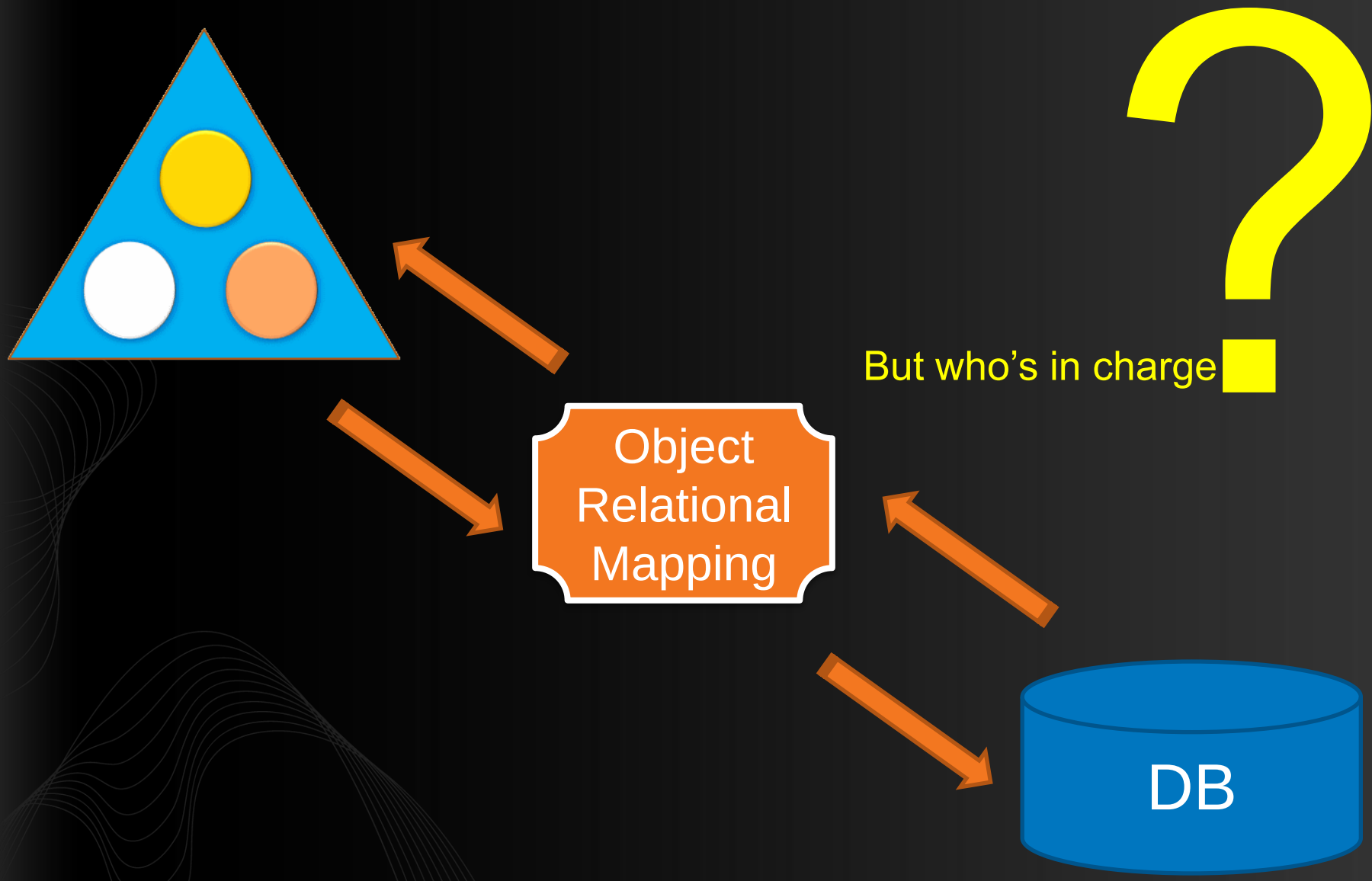


Persisting Domain Models to the DB



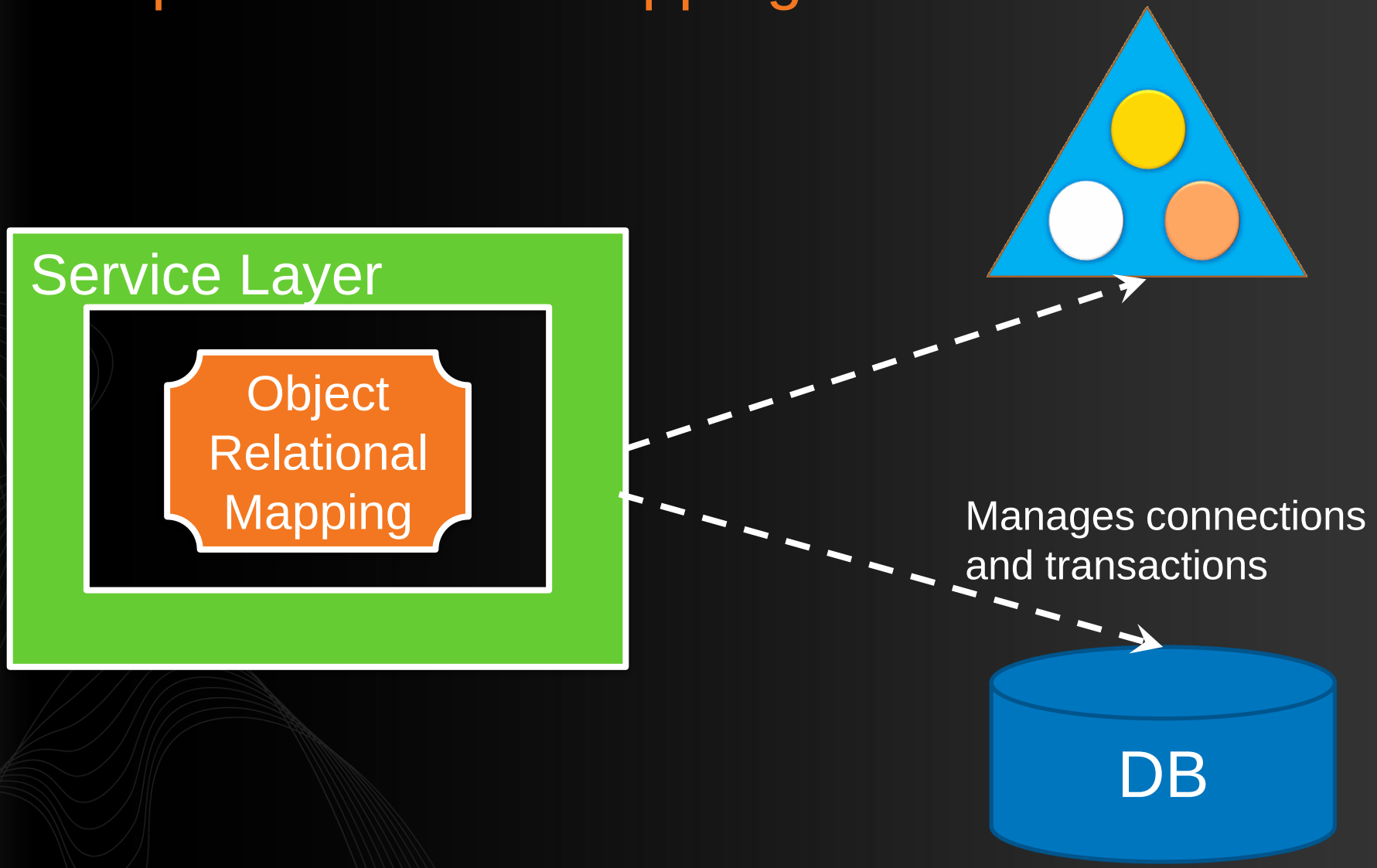
Basic O/R Mapping Demo

Persisting Domain Models to the DB



Service Layer orchestrates persistence

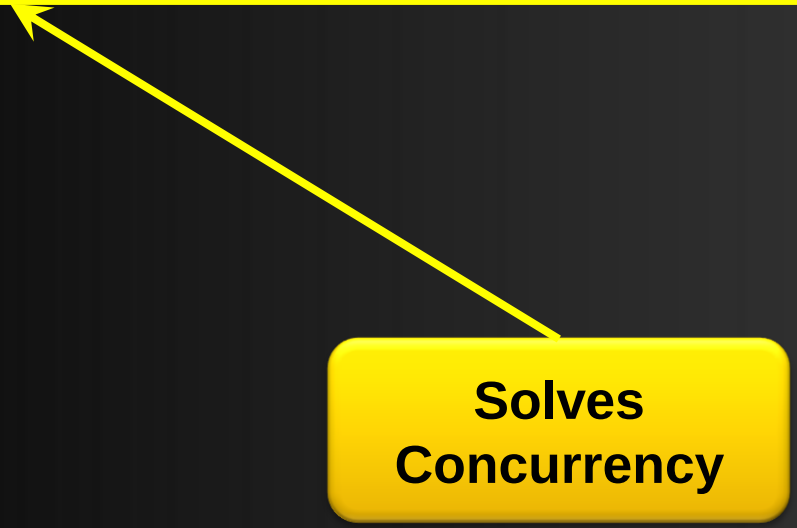
encapsulates O/R Mapping code



Service Layer Example Code

Service Layer sample code

```
public void MakeCustomerPreferred(Guid id)
{
  using(ISession s = ORM.OpenSession())
  using(ITransaction tx = s.BeginTransaction())
  {
    Customer c = s.Get<Customer>(id);
    c.MakePreferred();

    tx.Commit();
  }
}
```

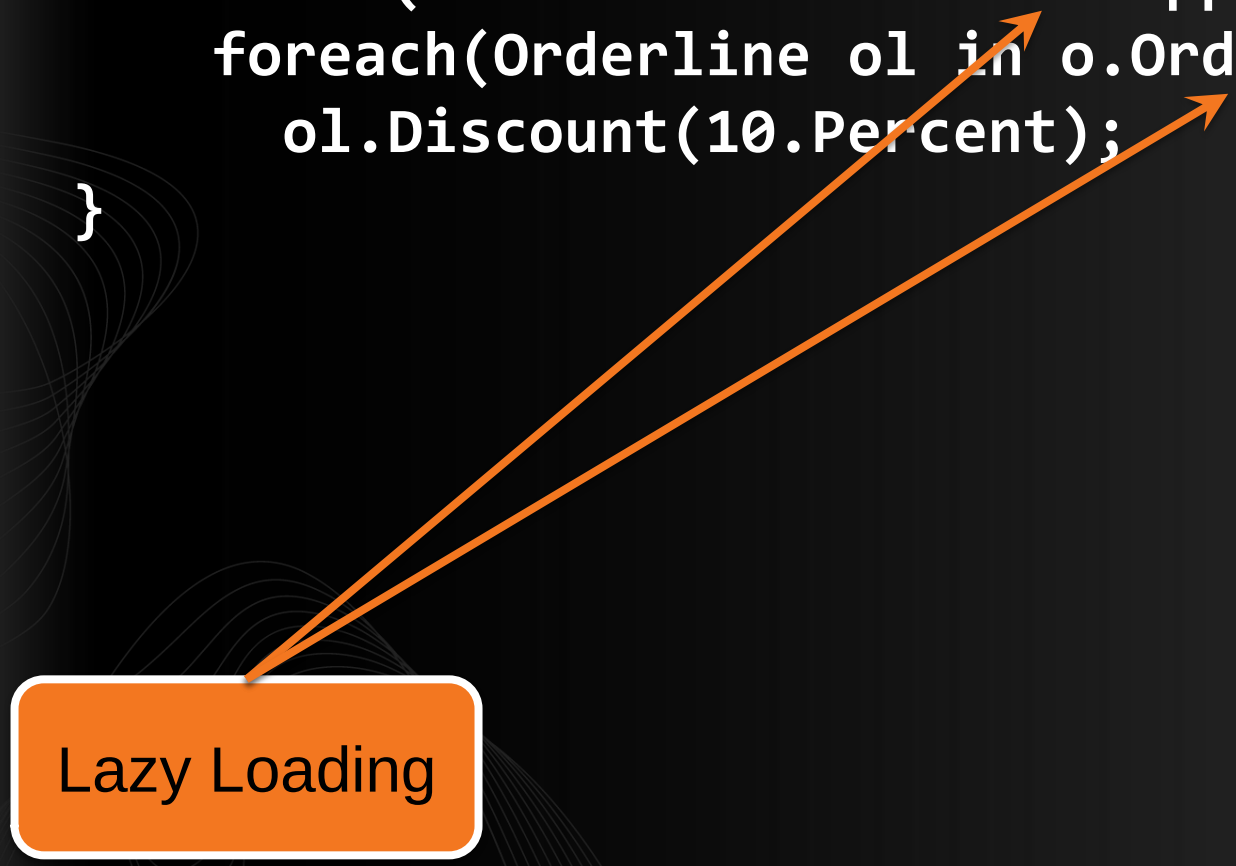


Service Layer sample code(2)

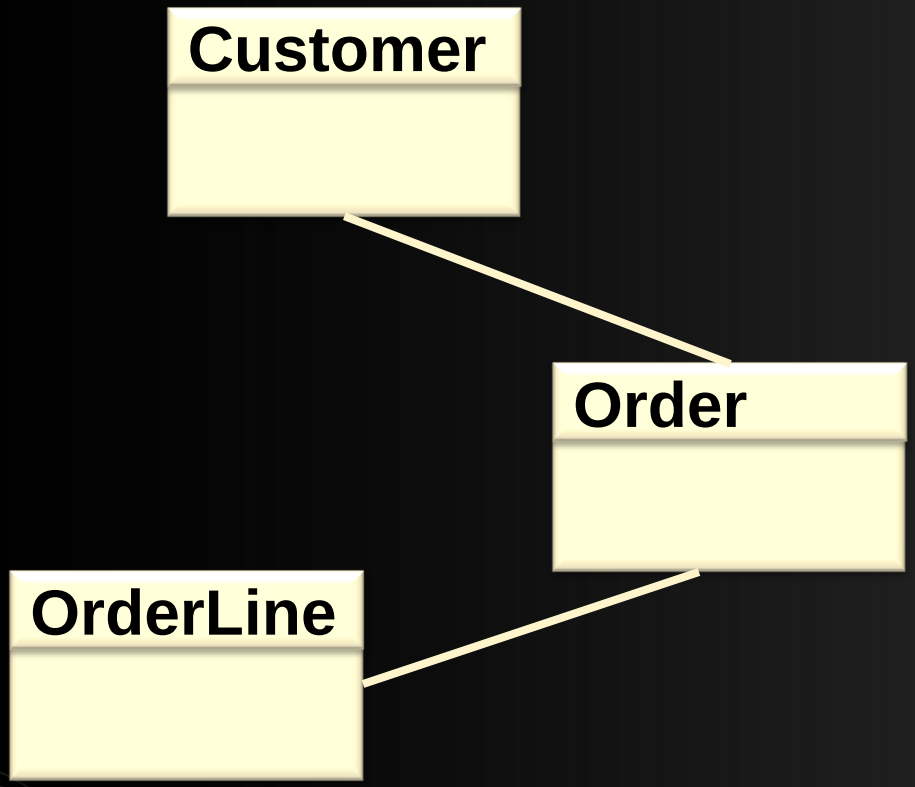
```
public void AddOrderToCustomer(
                Guid customerId, OrderData data)
{
  using(ISession s = ORM.OpenSession())
  using(ITransaction tx = s.BeginTransaction())
  {
    Customer c = s.Get<Customer>(customerId);

    c.AddOrder(Converter.ToOrder(data));

    tx.Commit();
  }
}
```

Domain Model sample : Customer

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



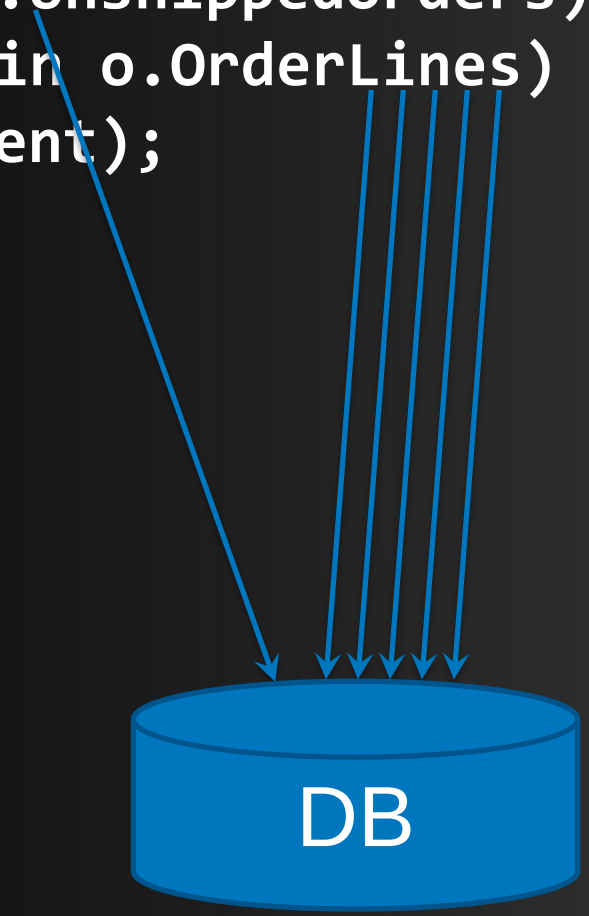
Lazy Loading – load what you use



Why load Orders and Order Lines
when you just want to add a new order

Dangers of lazy loading

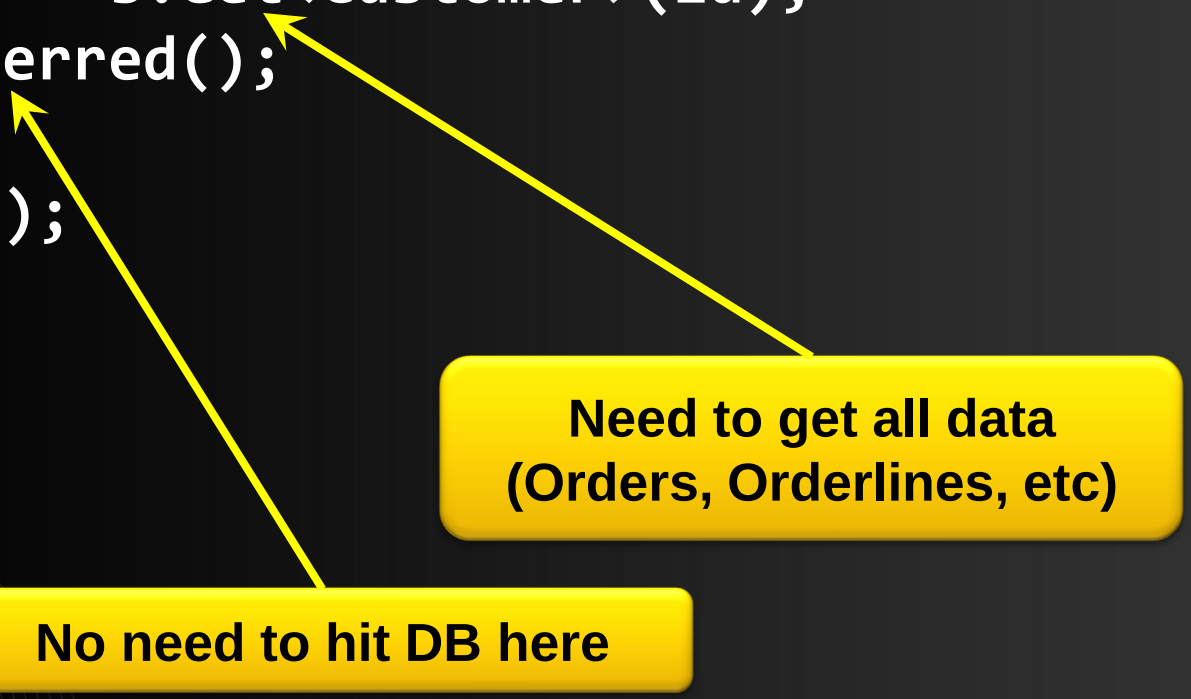
```
public void MakePreferred()
{
  foreach(Order o in this.UnshippedOrders)
    foreach(Orderline ol in o.OrderLines)
      ol.Discount(10.Percent);
}
```



Lazy Loading – DB profiling

Minimize DB roundtrips: Eager Fetching

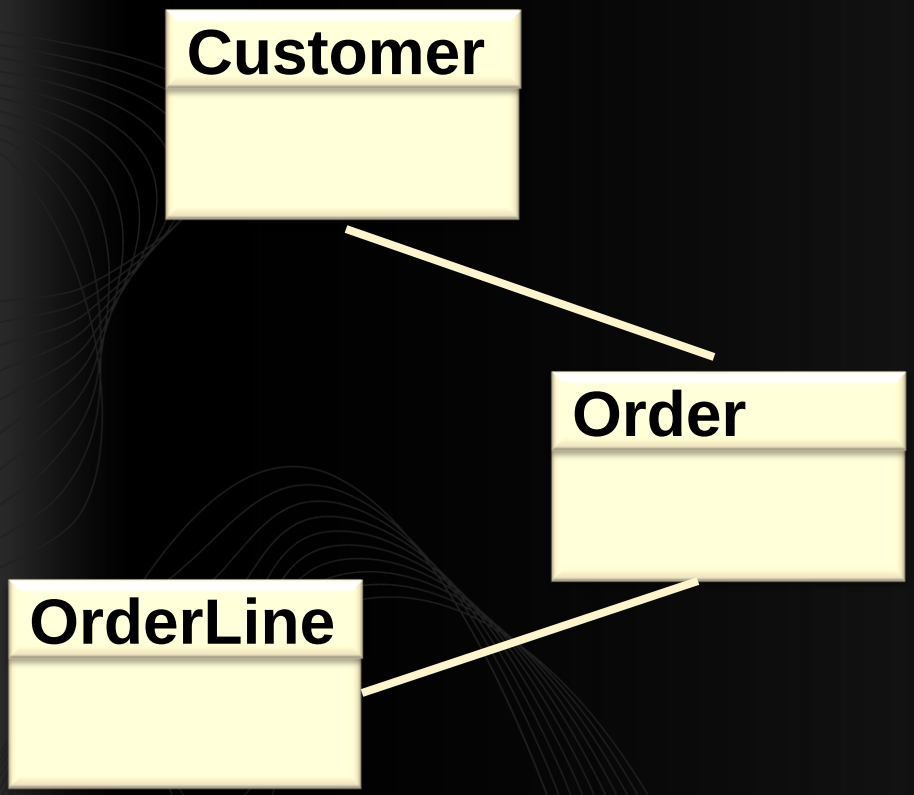
```
public void MakeCustomerPreferred(Guid id)
{
  using(ISession s = ORM.OpenSession())
  using(ITransaction tx = s.BeginTransaction())
  {
    Customer c = s.Get<Customer>(id);
    c.MakePreferred();

    tx.Commit();
  }
}
```



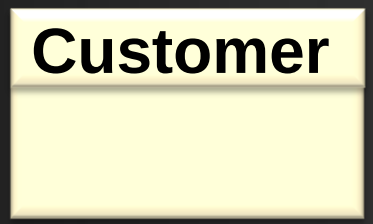
Easier said than done

the same entity can be loaded many ways

Making a customer "preferred"



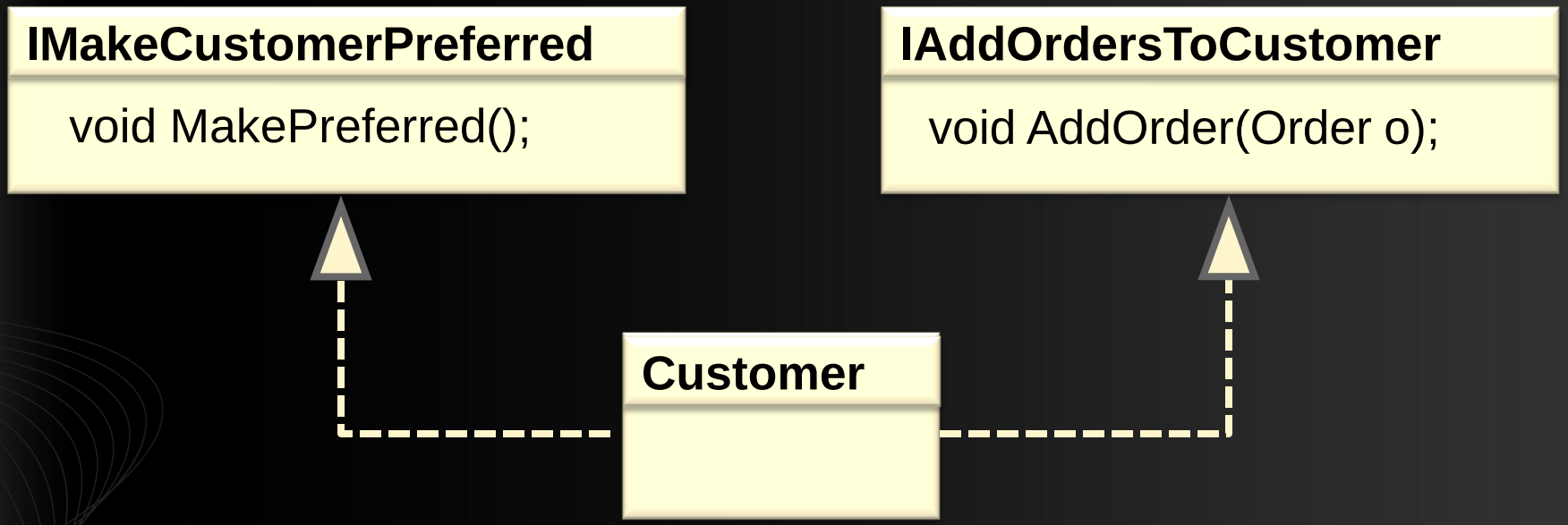
Adding an Order



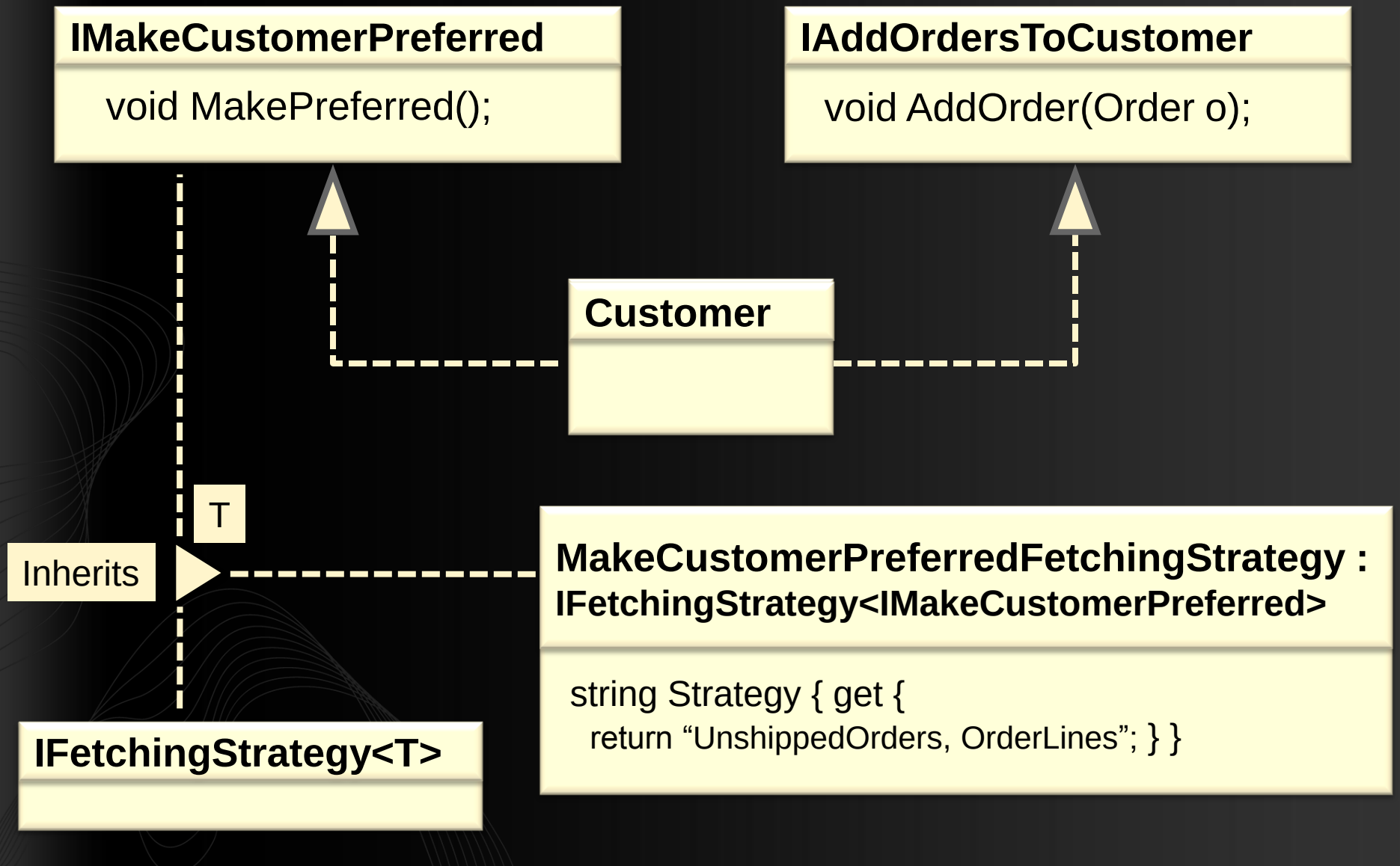
I want it all

Pretty please?

Using interfaces to differentiate roles

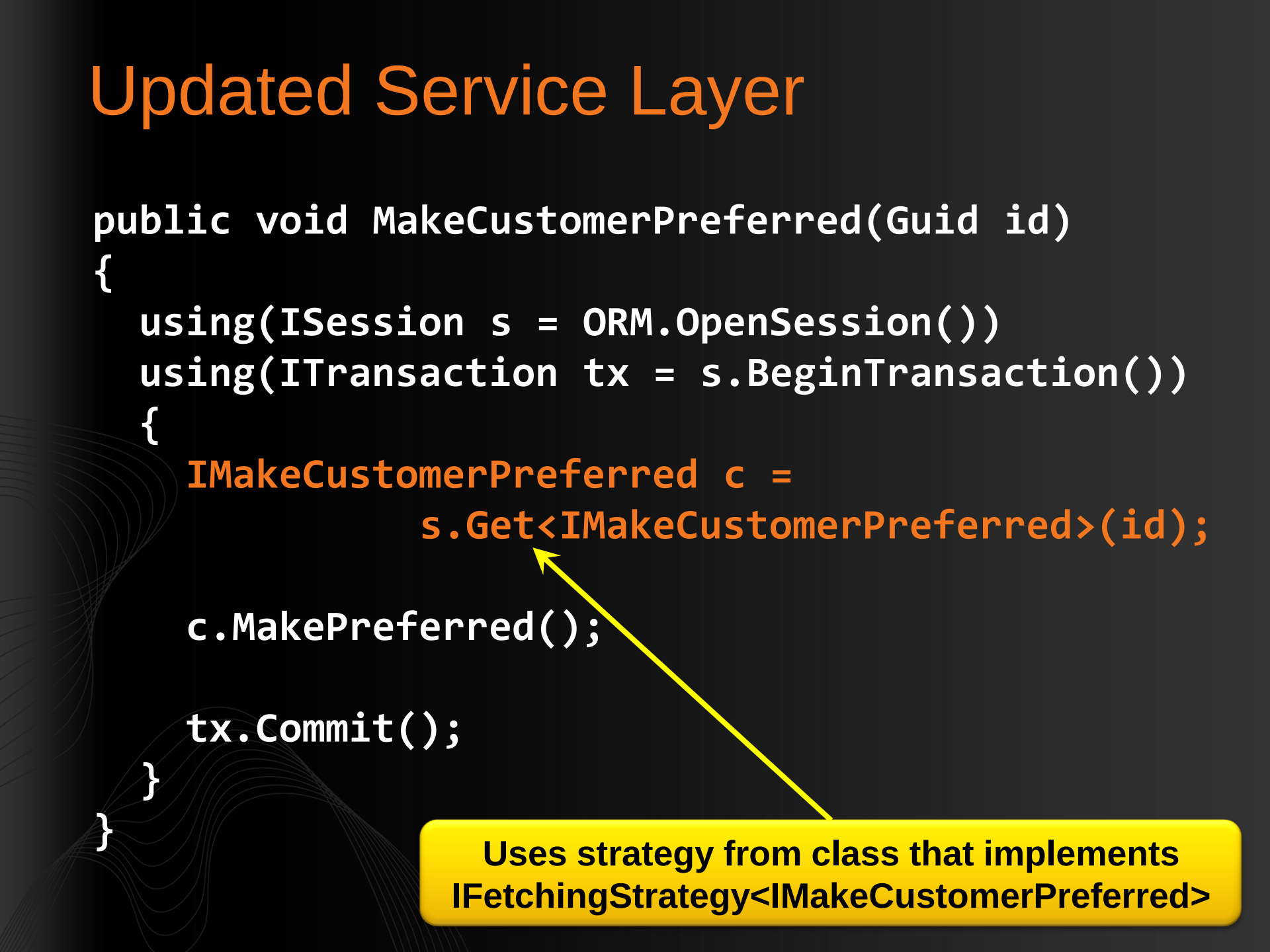


Specifying behavior per role



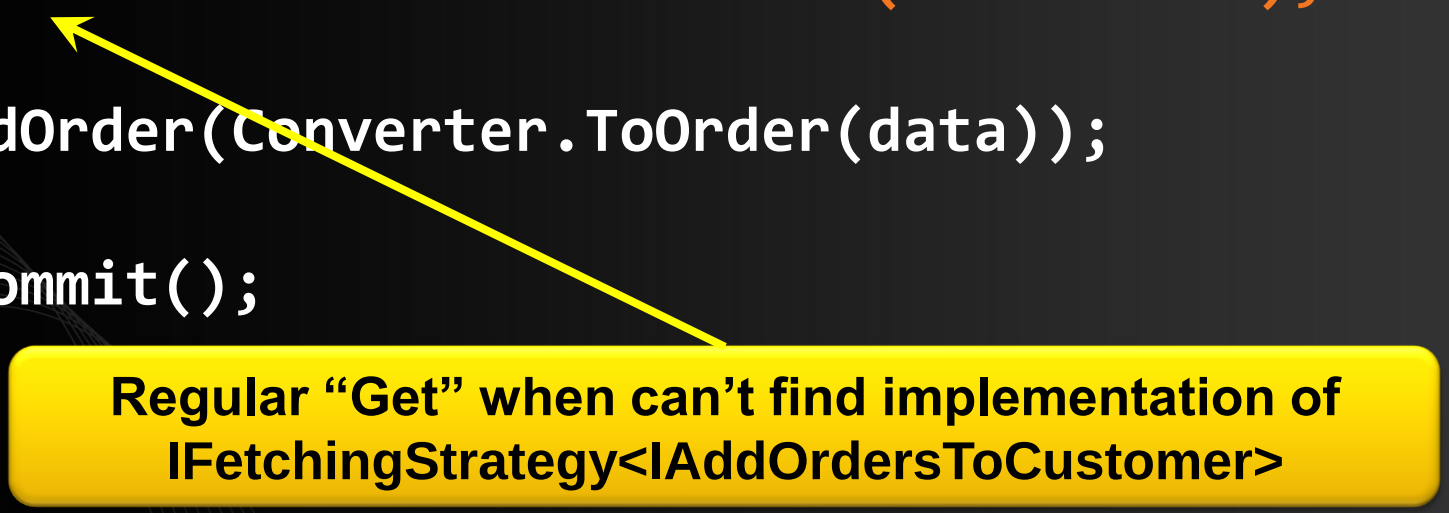
Updated Service Layer

```
public void MakeCustomerPreferred(Guid id)
{
  using(ISession s = ORM.OpenSession())
  using(ITransaction tx = s.BeginTransaction())
  {
    IMakeCustomerPreferred c =
              s.Get<IMakeCustomerPreferred>(id);

    c.MakePreferred();

    tx.Commit();
  }
}
```



Updated Service Layer – Add Order

```
public void AddOrderToCustomer(
                Guid customerId, OrderData data)
{
  using(ISession s = ORM.OpenSession())
  using(ITransaction tx = s.BeginTransaction())
  {
    IAddOrdersToCustomer c =
      s.Get<IAddOrdersToCustomer>(customerId);

    c.AddOrder(Converter.ToOrder(data));

    tx.Commit();
  }
}
```



Eager Fetching – DB profiling

But some rules *have* to be in the DB don't they?

No Duplicate user names

@

No Duplicate email addresses

Unique Constraints work

Memory of a single server can hold:

1 million user names = ~ 1GB

@

1 million email addresses, same

Partition data load across servers for much more

* 1 KB per username / email

Don’t copy EBay or Amazon
just think about what they’re doing

Amazon’s “Dynamo”, an in-memory, replicated grid

EBay shards their DB’s over many machines

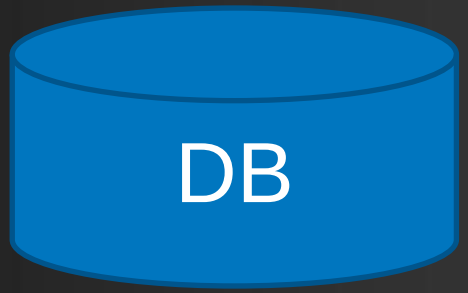
Both minimize DB access

Joins hurt scalability: De-Normalize

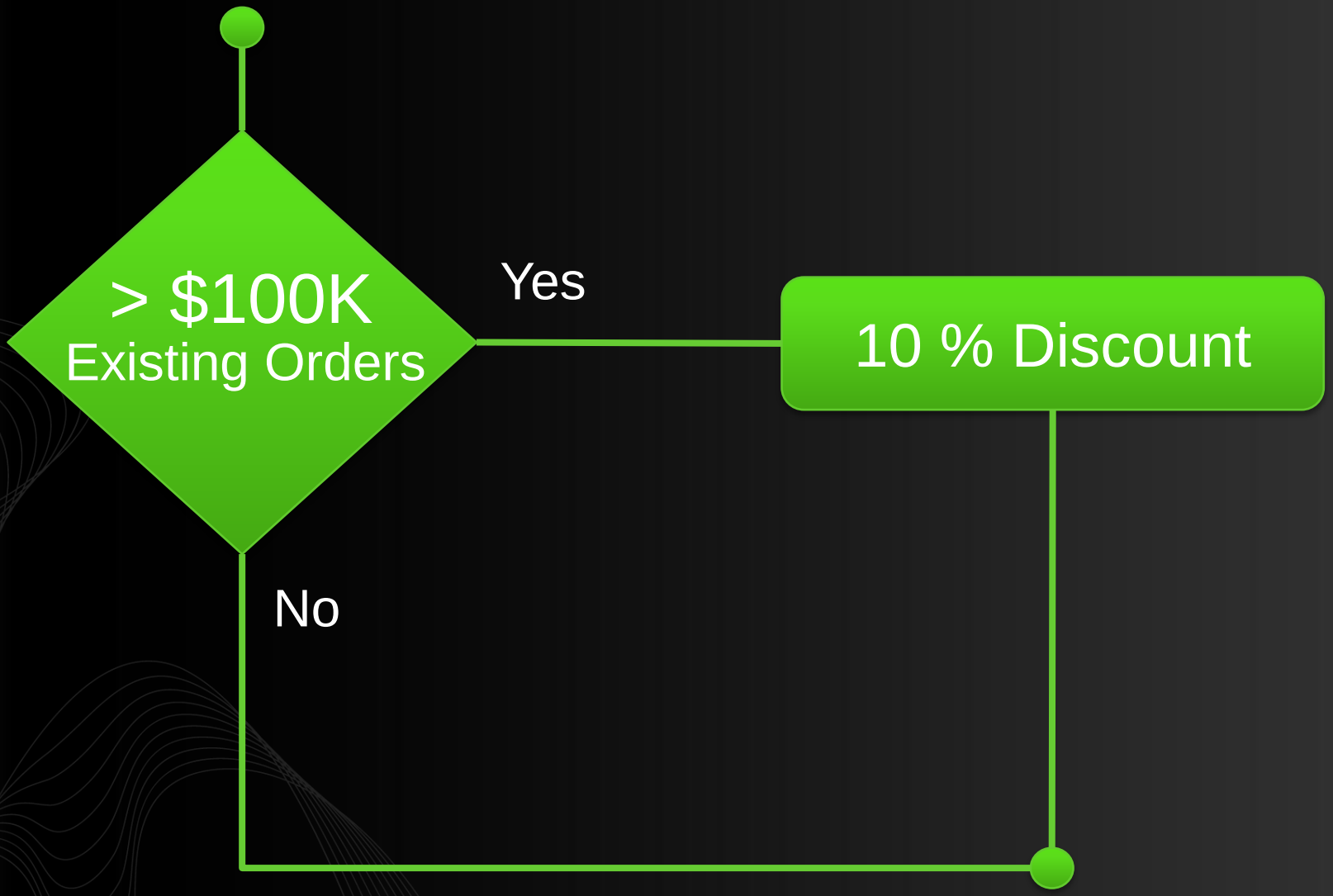
Keep the same interface, change internals

Object Relational Mapping

Update mapping & schema to correspond



New Rule for “Add Order”



Customer class, AddOrder method

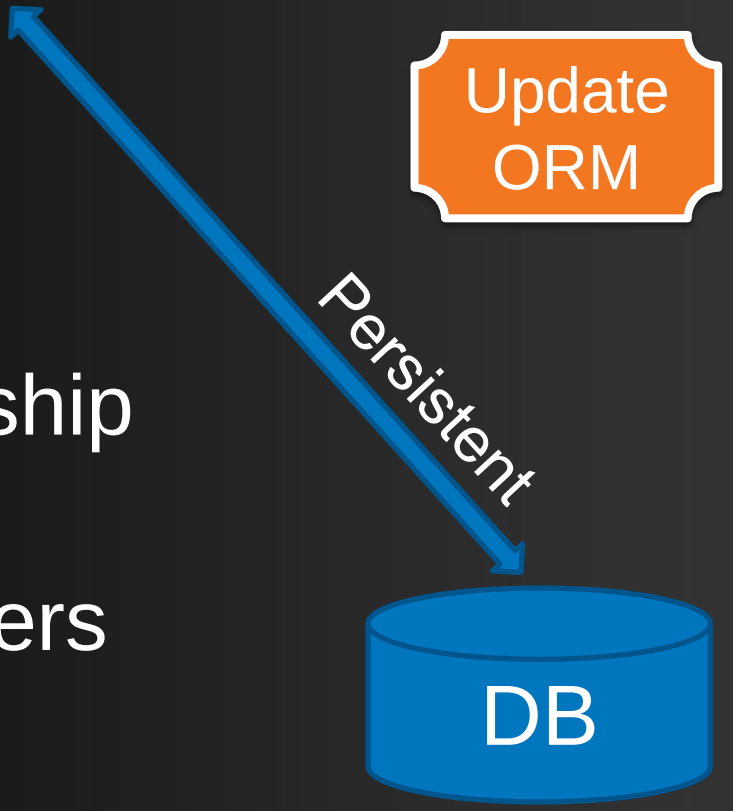
```
public void AddOrder(Order o)
{
  float sum;

  foreach(Order order in this.ShippedOrders)
    sum += order.Amount;

  if (sum > 100*1000.0)
    o.Discount(10/100);

  this.Orders.Add(o);
  o.Customer = this;
}
```

Introduce member to replace iteration

sumOfShippedOrders

Update when orders ship

Use when adding orders



Thank you

Udi Dahan – The Software Simplist
.NET Development Expert & SOA Specialist

www.UdiDahan.com

Resources

- **Blog post on Fetching Strategy Design**
  http://udidahan.weblogs.us/2007/04/23/fetching-strategy-design/

- **Changes to NHibernate to support Fetching Strategies**
  http://udidahan.weblogs.us/2007/09/16/fetching-strategy-nhibernate-implementation-available/

- **My blog**
  http://www.UdiDahan.com or http://udidahan.weblogs.us