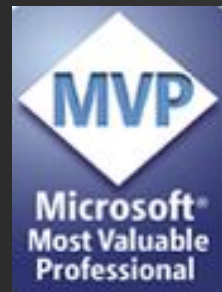




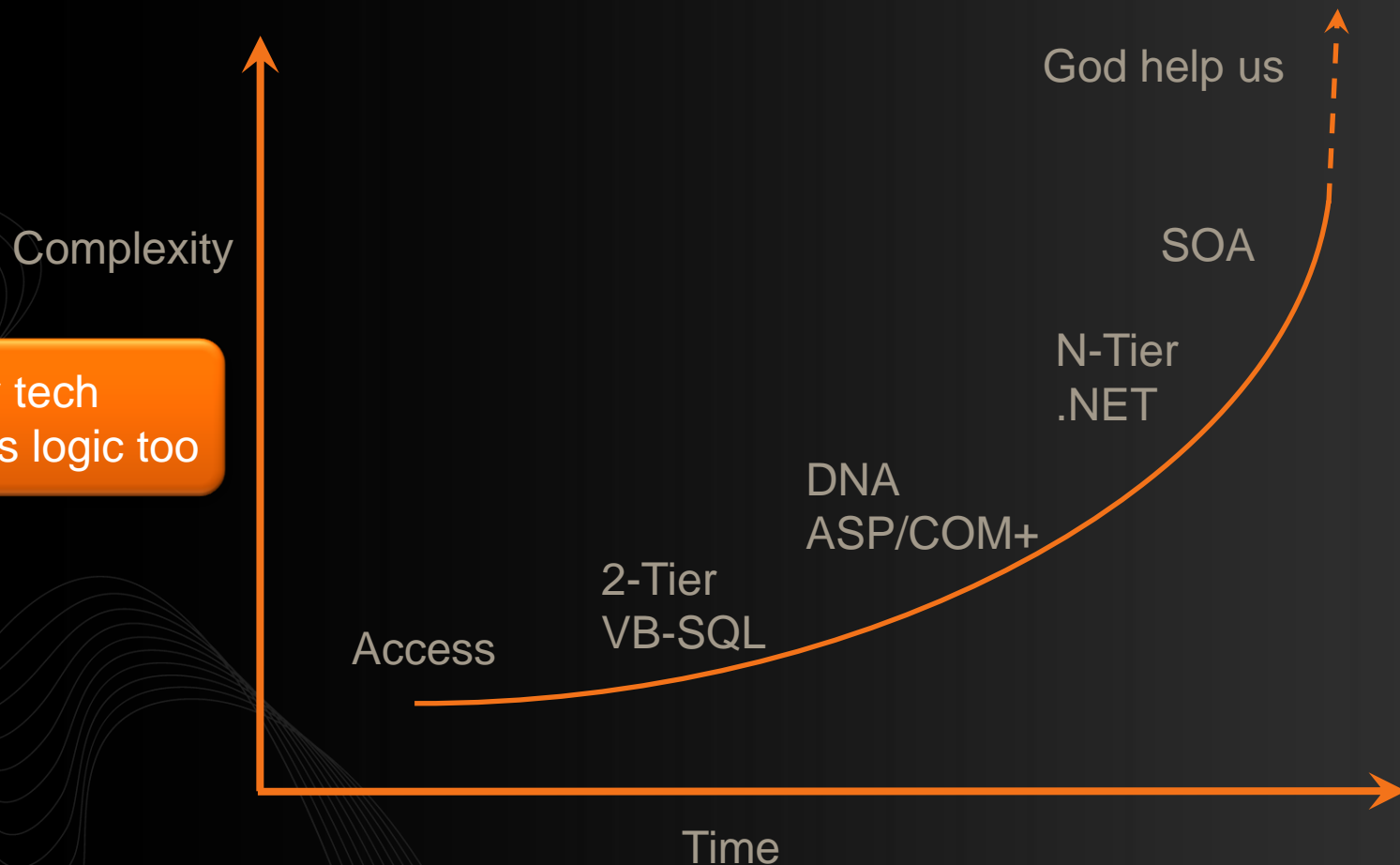
Designing High Performance, Persistent Domain Models [ARC 401]

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

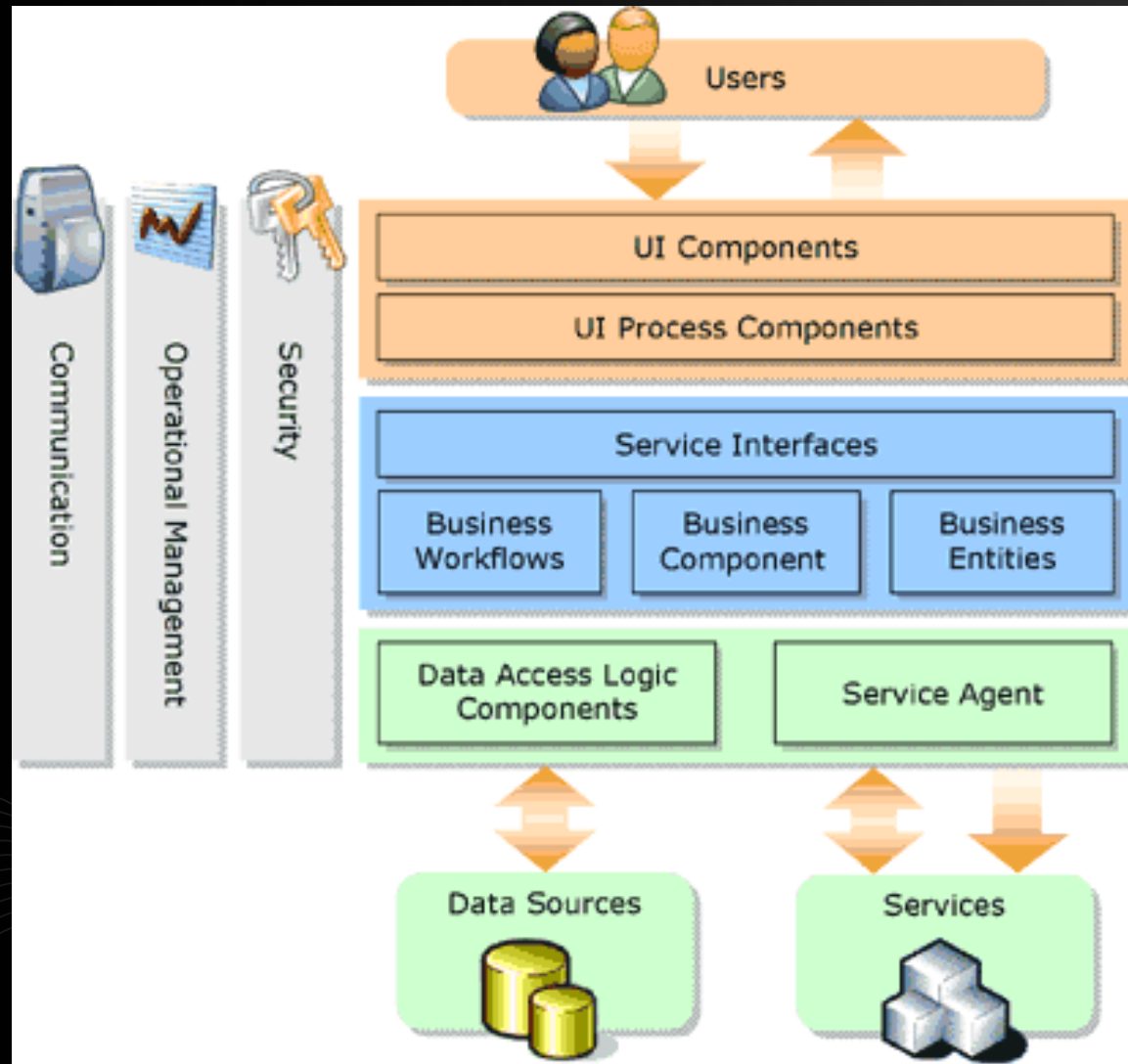


Moore's law - kinda

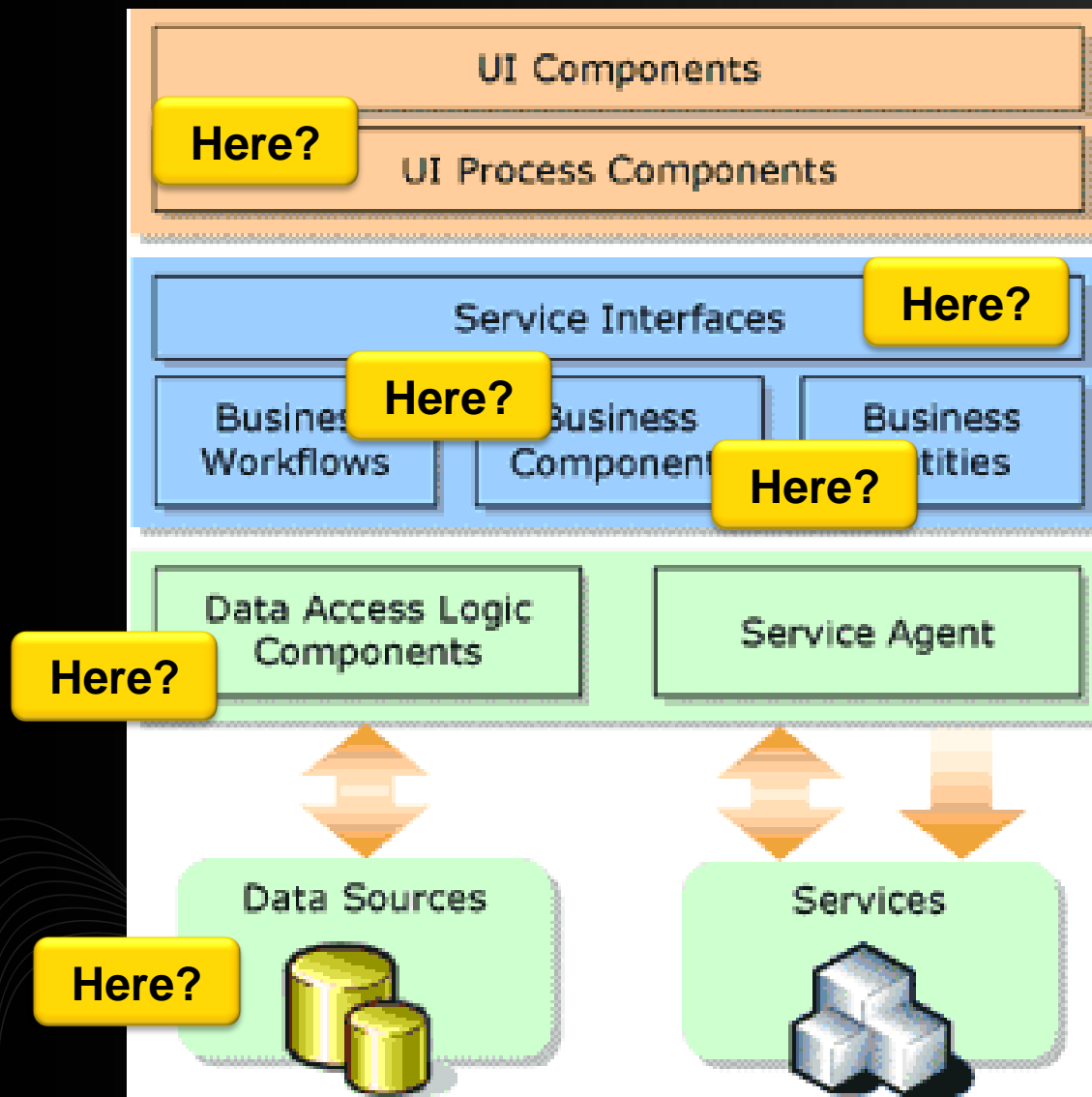
You have to deal with twice as much %\$&! this year as you did last year



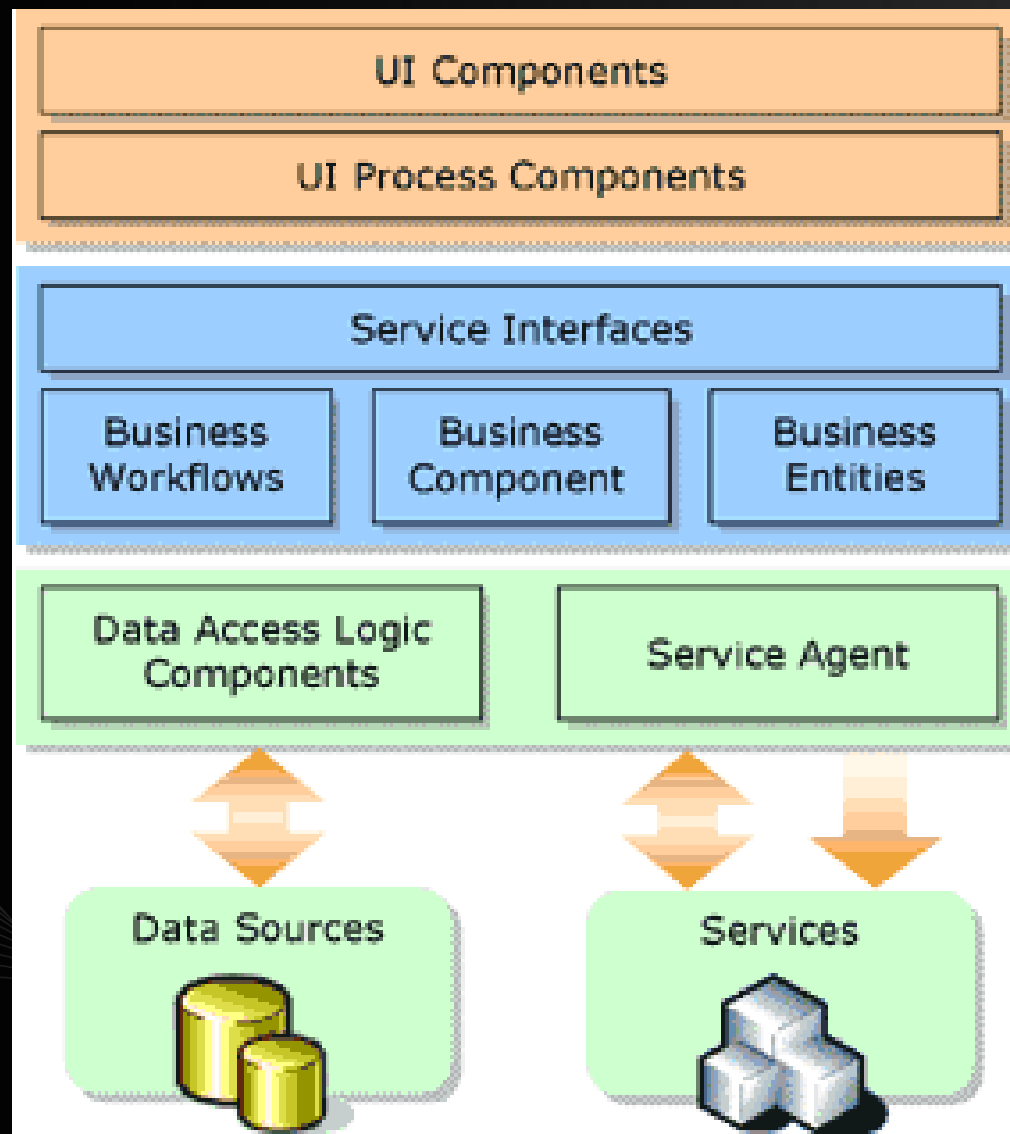
What most architecture looks like today



But where do business rules go?



No real reuse between tiers



The book

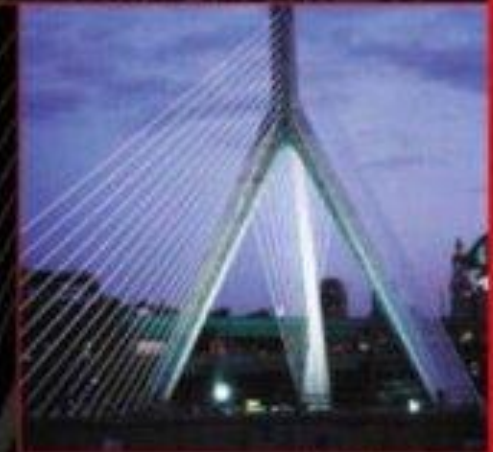
that changed everything

The Addison-Wesley Signature Series

PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE

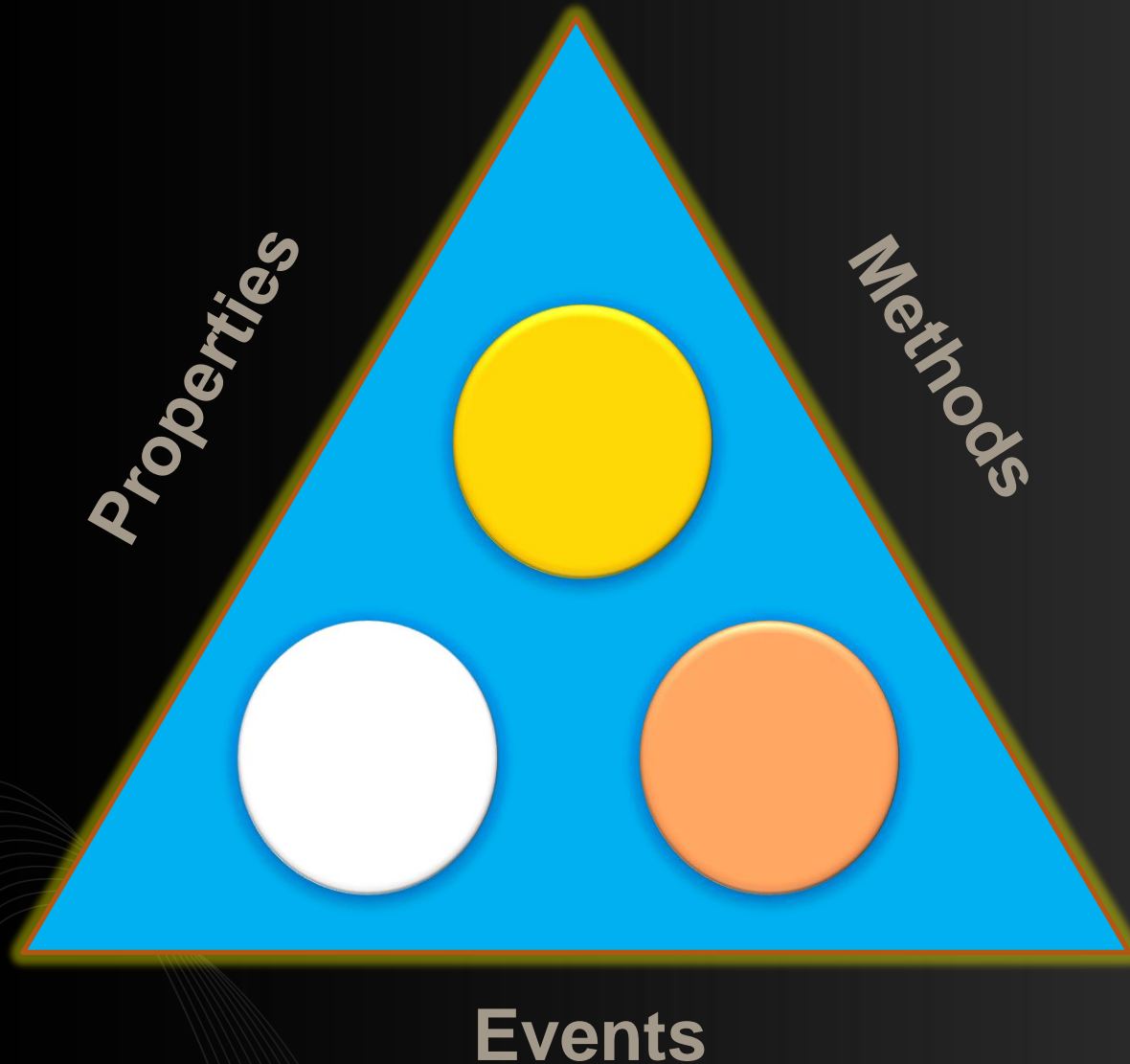
MARTIN FOWLER

WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HEATT,
ROBERT MEE, AND
RANDY STAFFORD

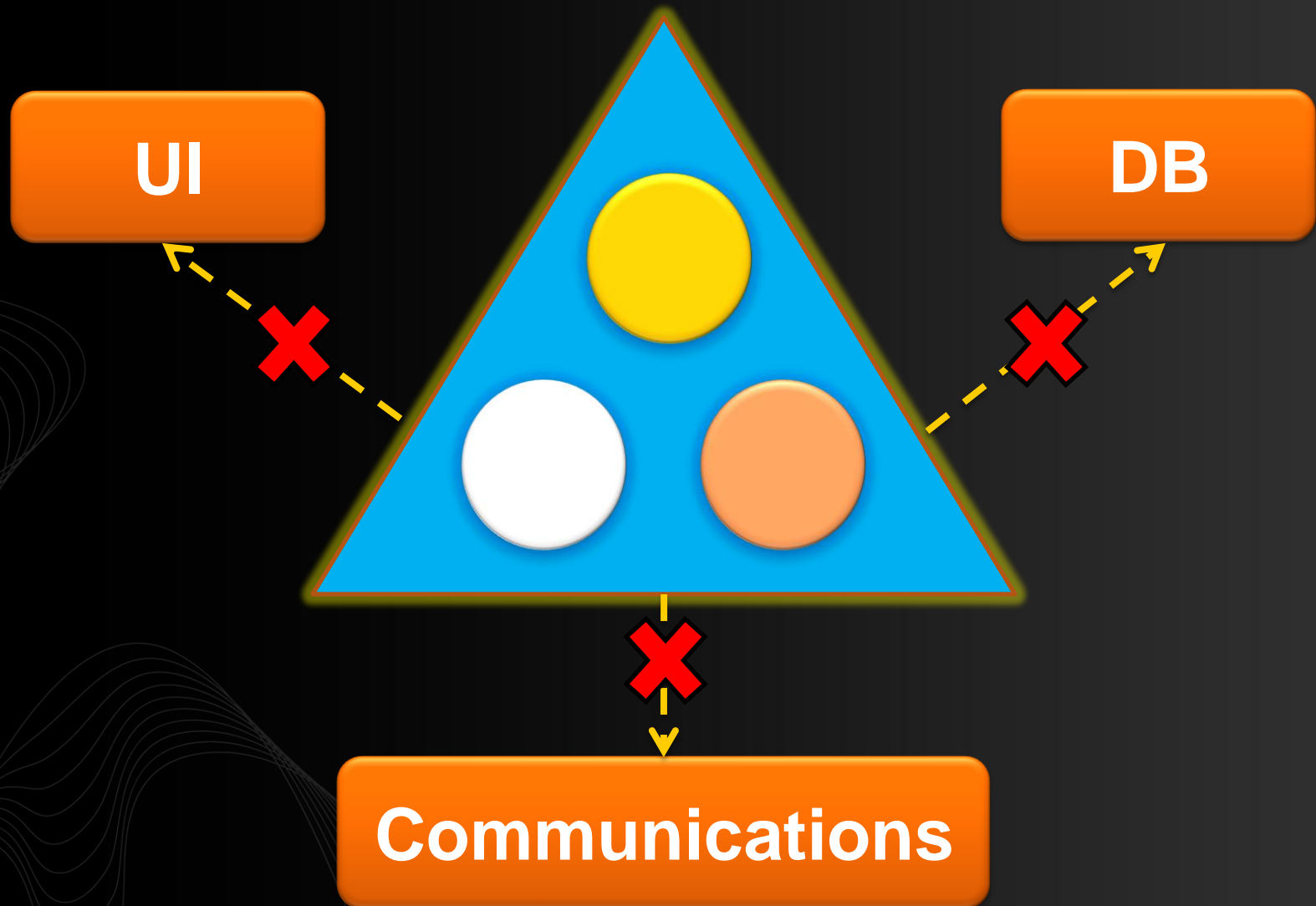


... and the pattern that battles complexity...

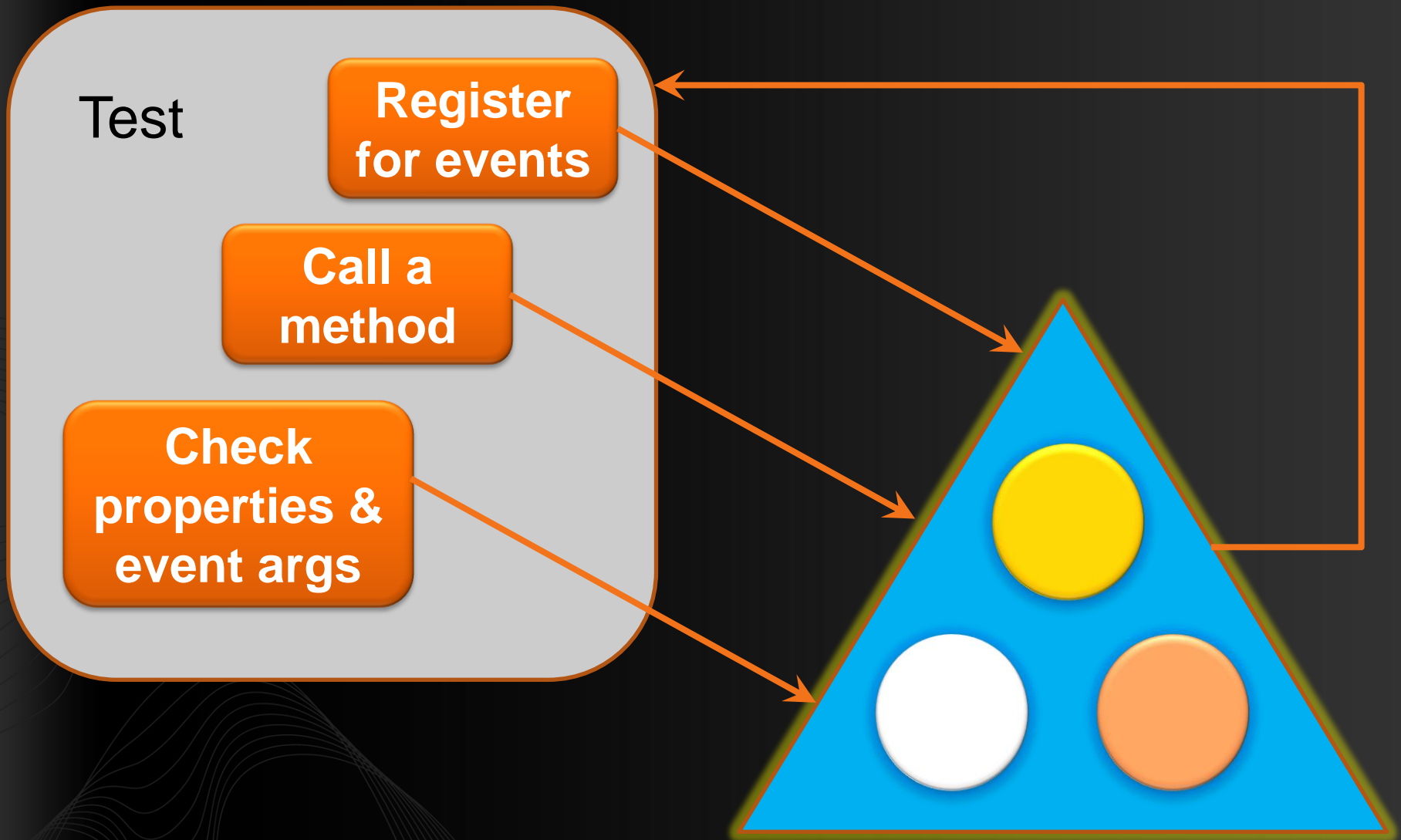
The Domain Model Pattern



Independent of all concerns

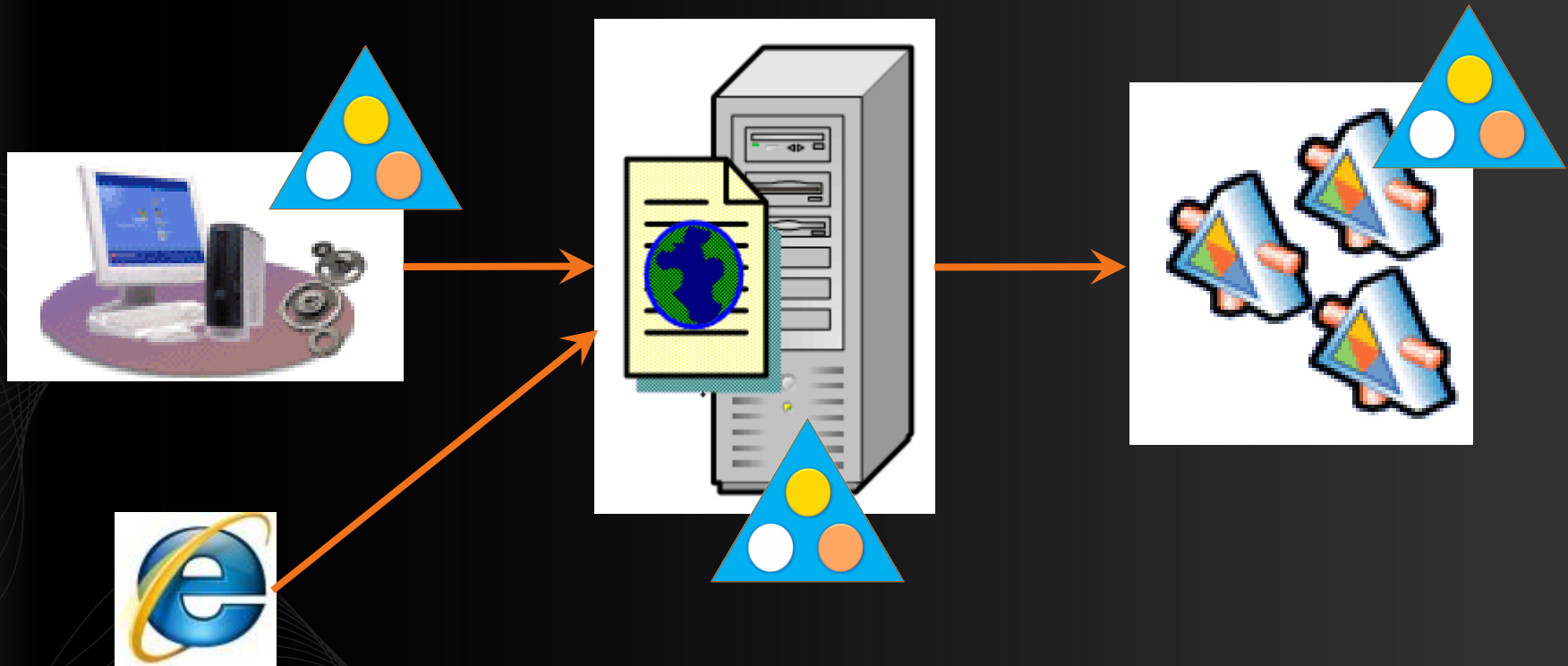


Highly testable

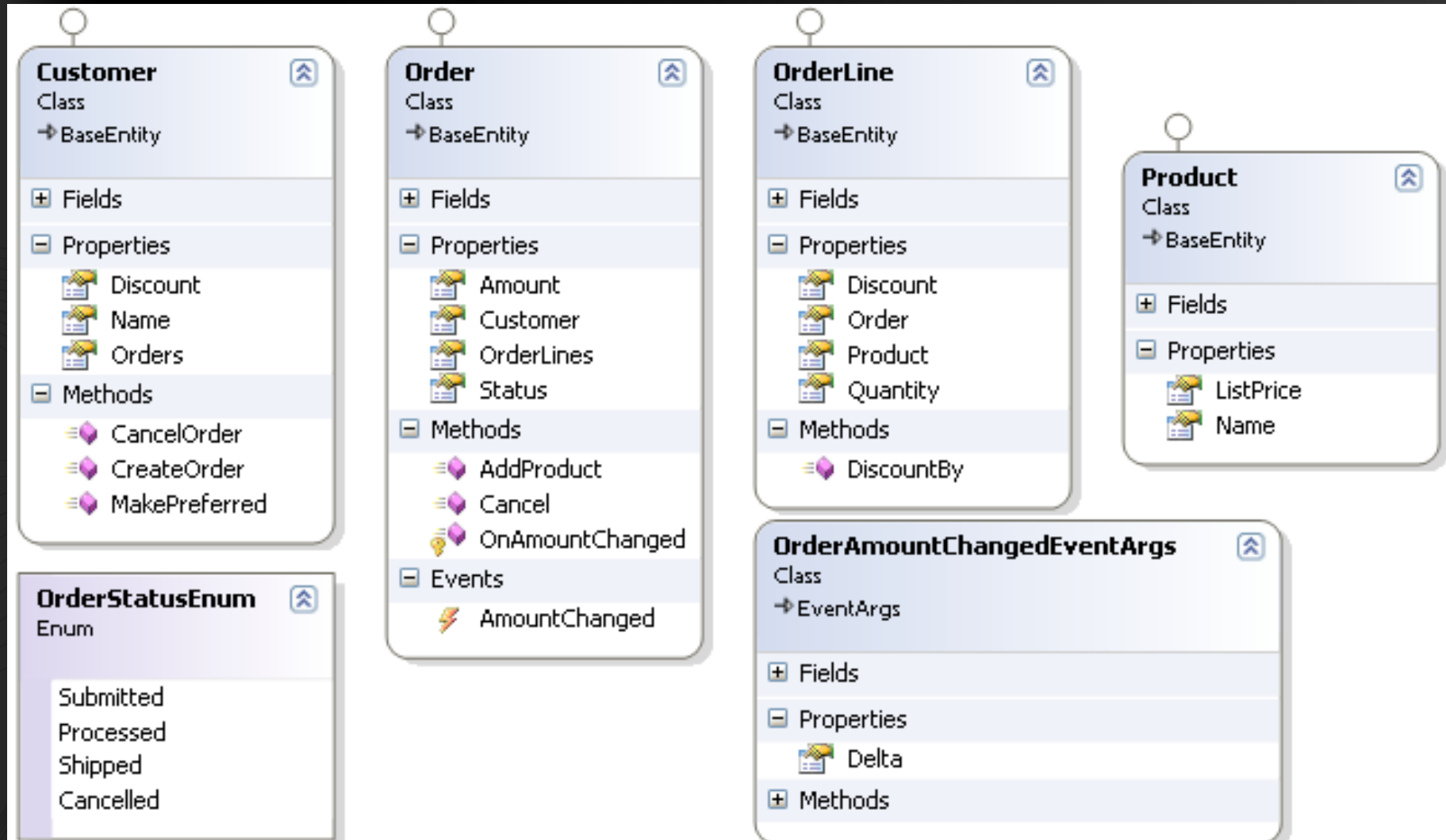


Can be deployed Multi-Tier

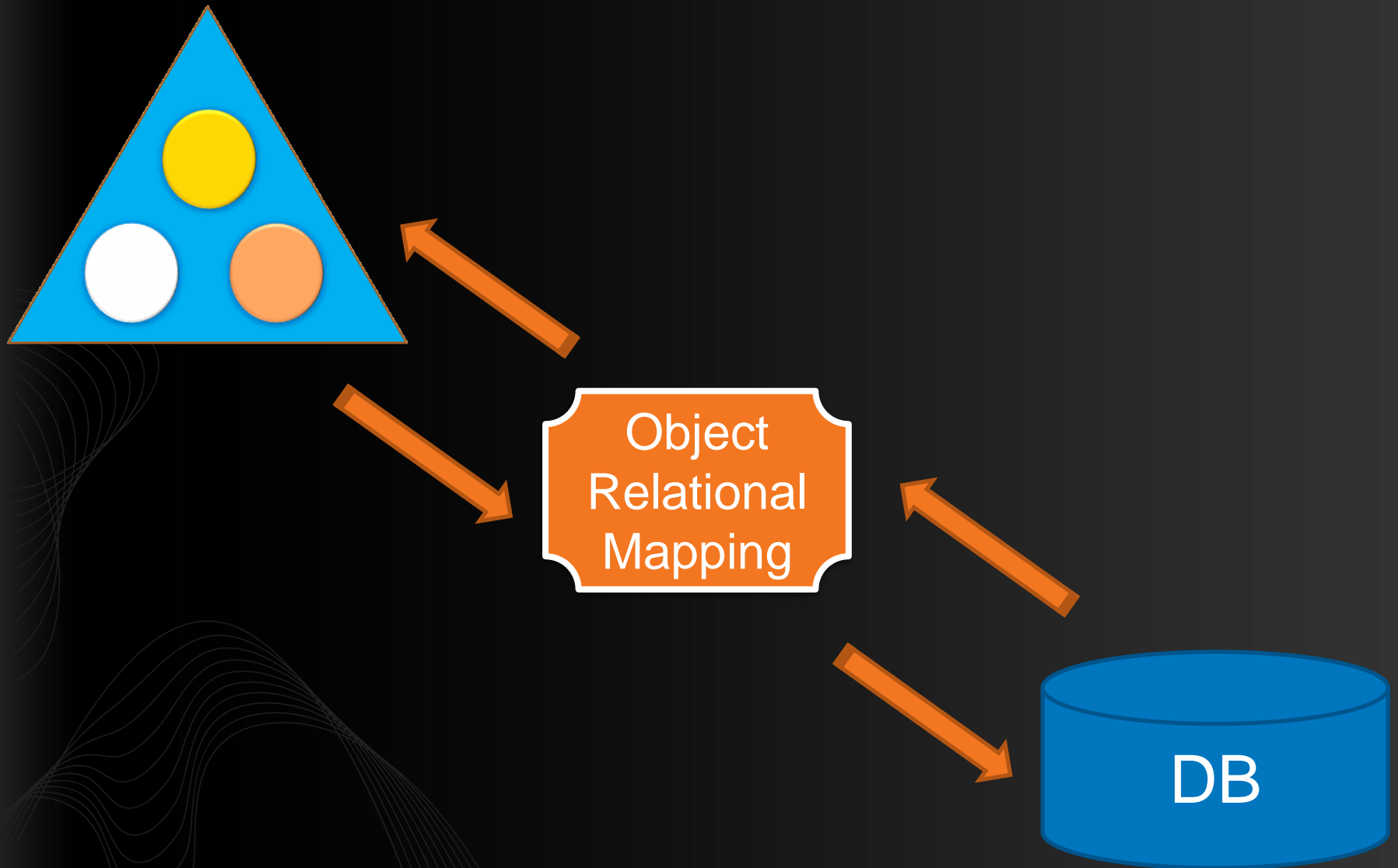
it's just a bunch of DLLs – or even just one



Domain Models are made of “Plain old .NET objects”



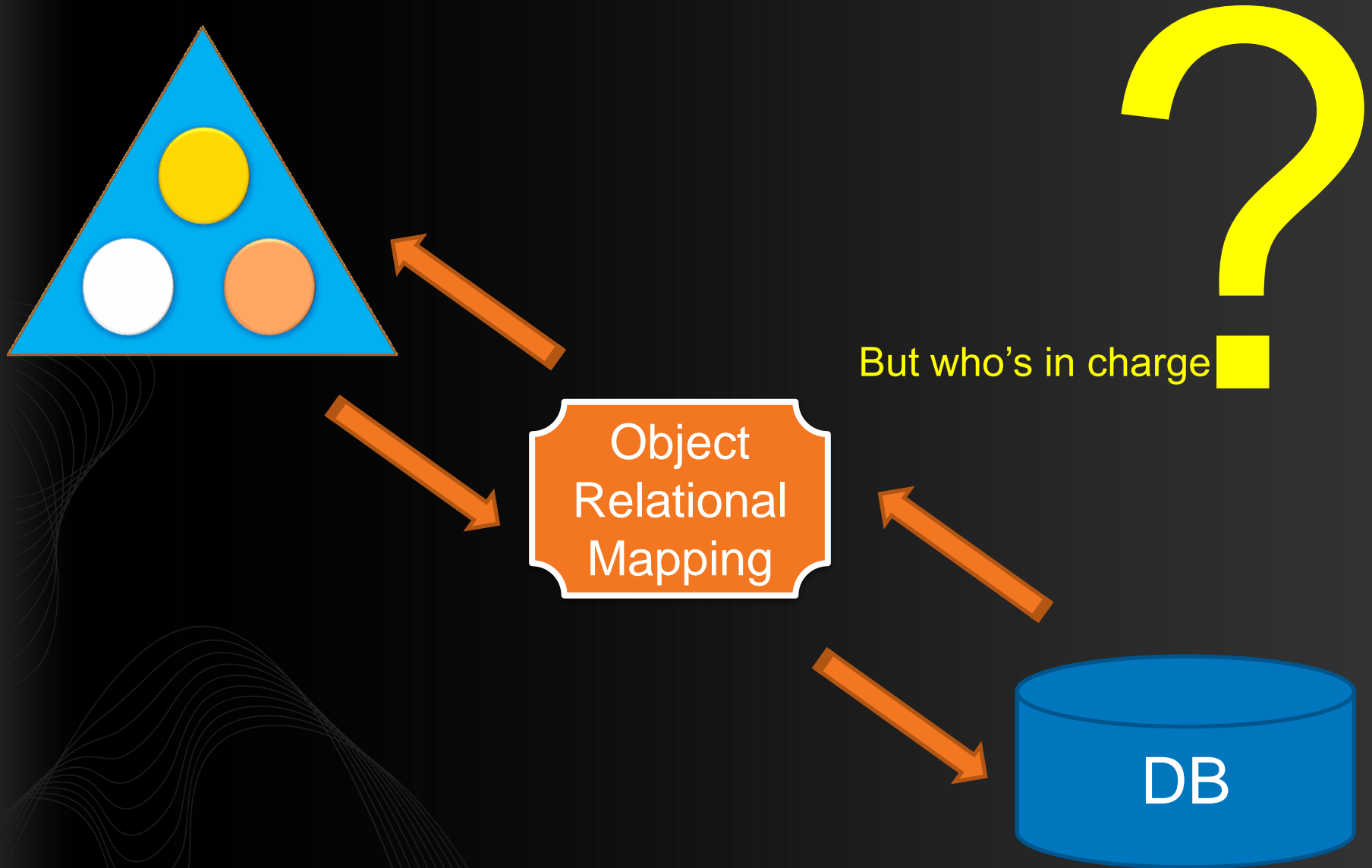
Persisting Domain Models to the DB



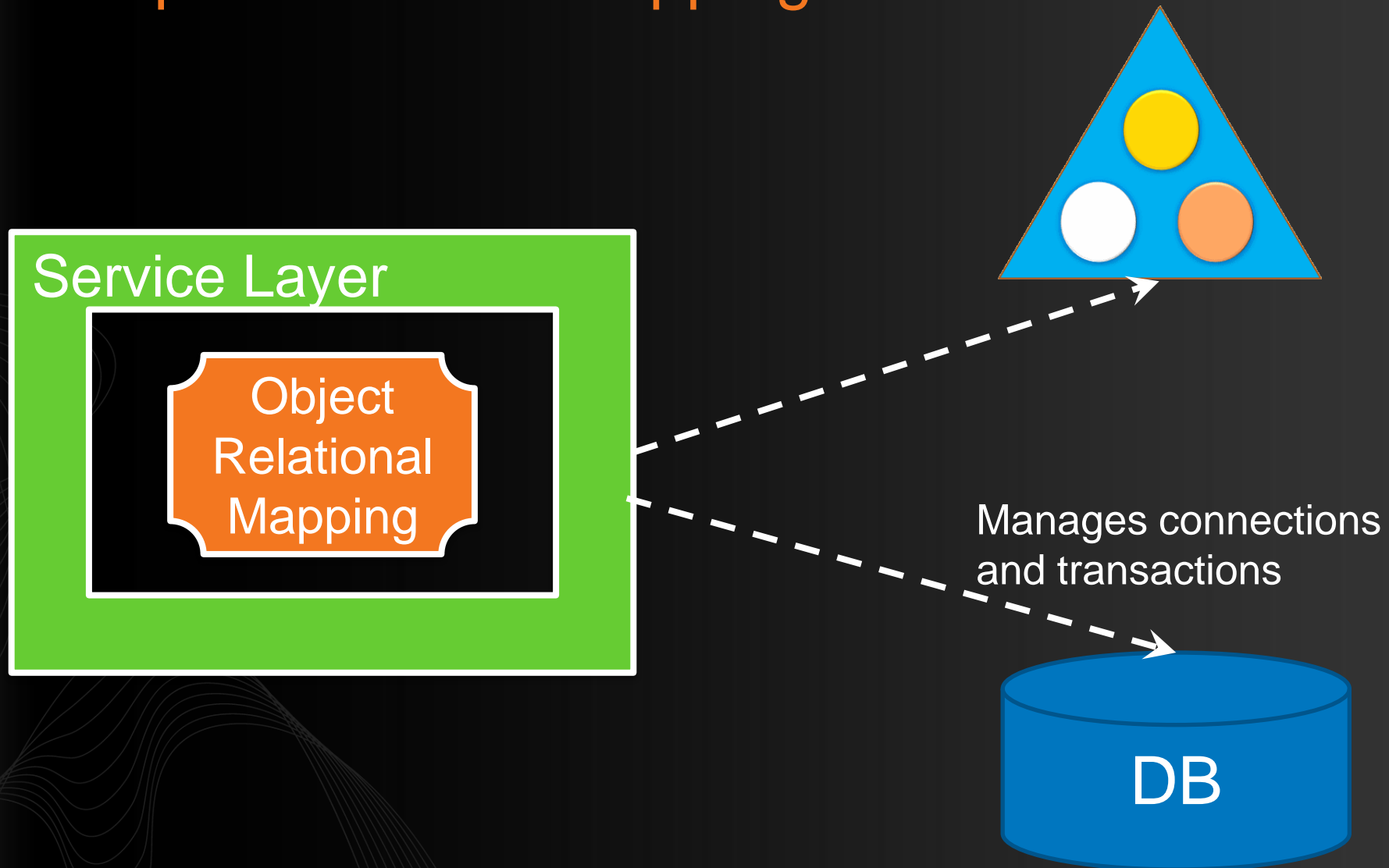


Basic O/R Mapping Demo

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();
    }
}
```

**Solves
Concurrency**



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

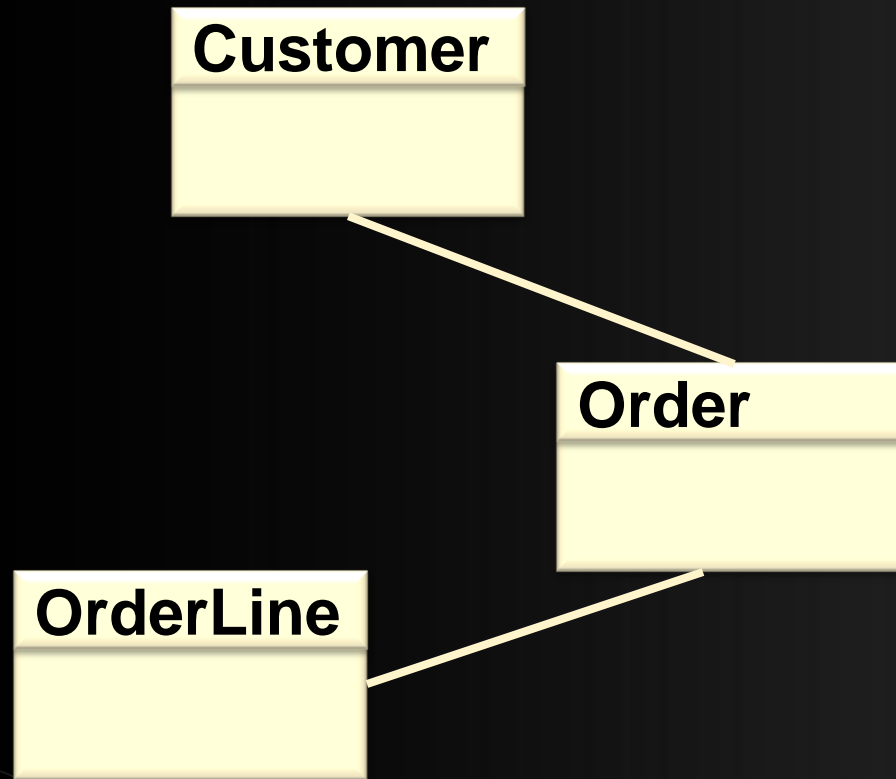
```
public void MakePreferred()  
{  
    foreach(Order o in this.UnshippedOrders)  
        foreach(Orderline ol in o.OrderLines)  
            ol.Discount(10.Percent);  
}
```



Lazy Loading

The diagram illustrates the concept of lazy loading in the provided code. An orange rounded rectangle labeled 'Lazy Loading' has two orange arrows pointing from its top-right corner to the nested loops in the code: the first arrow points to the 'o.OrderLines' property access, and the second arrow points to the 'ol' variable in the inner loop's 'foreach' statement. This indicates that the inner loop's execution is deferred until it is actually needed.

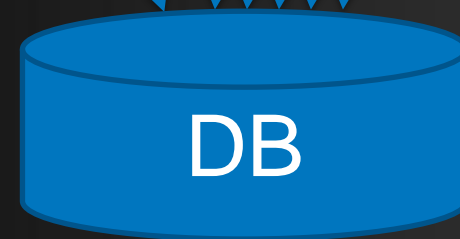
Lazy Loading – load what you use



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);  
}
```

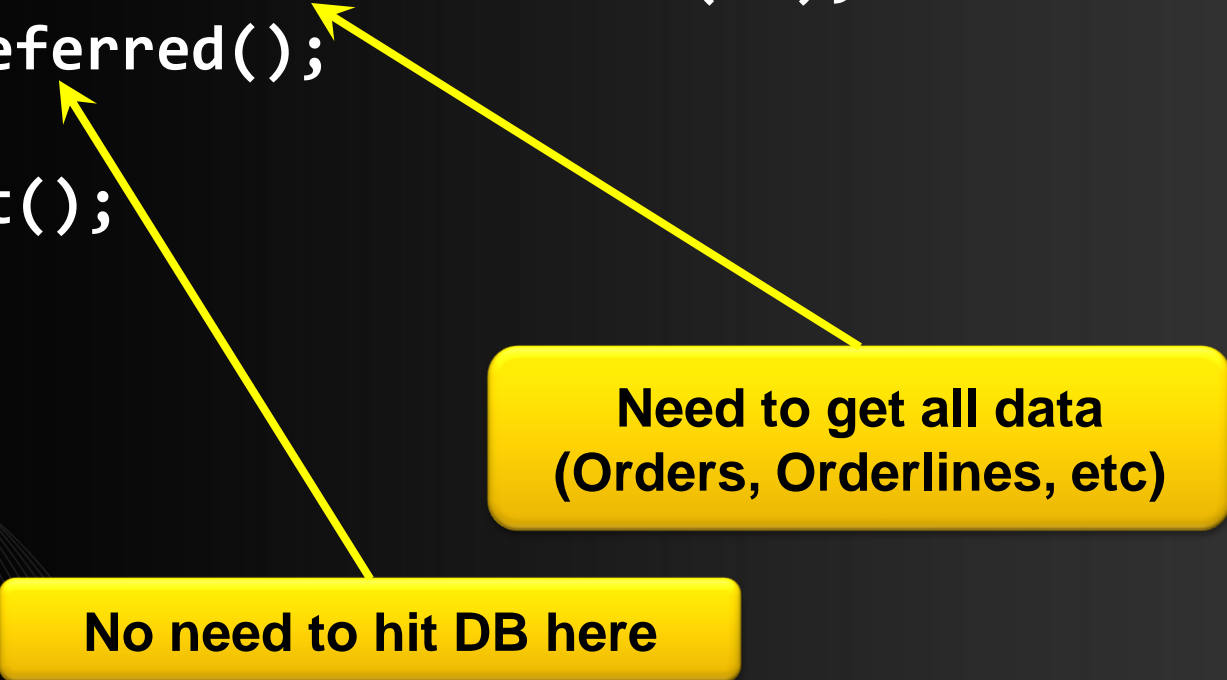




Lazy Loading – DB profiling

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();
    }
}
```



No need to hit DB here

Need to get all data
(Orders, Orderlines, etc)

Easier said than done

the same entity can be loaded many ways

Making a customer
“preferred”

Adding an Order

Customer

```
classDiagram
    class Customer
    class Order
    class OrderLine
    Customer --> Order
    Order --> OrderLine
```

Order

OrderLine

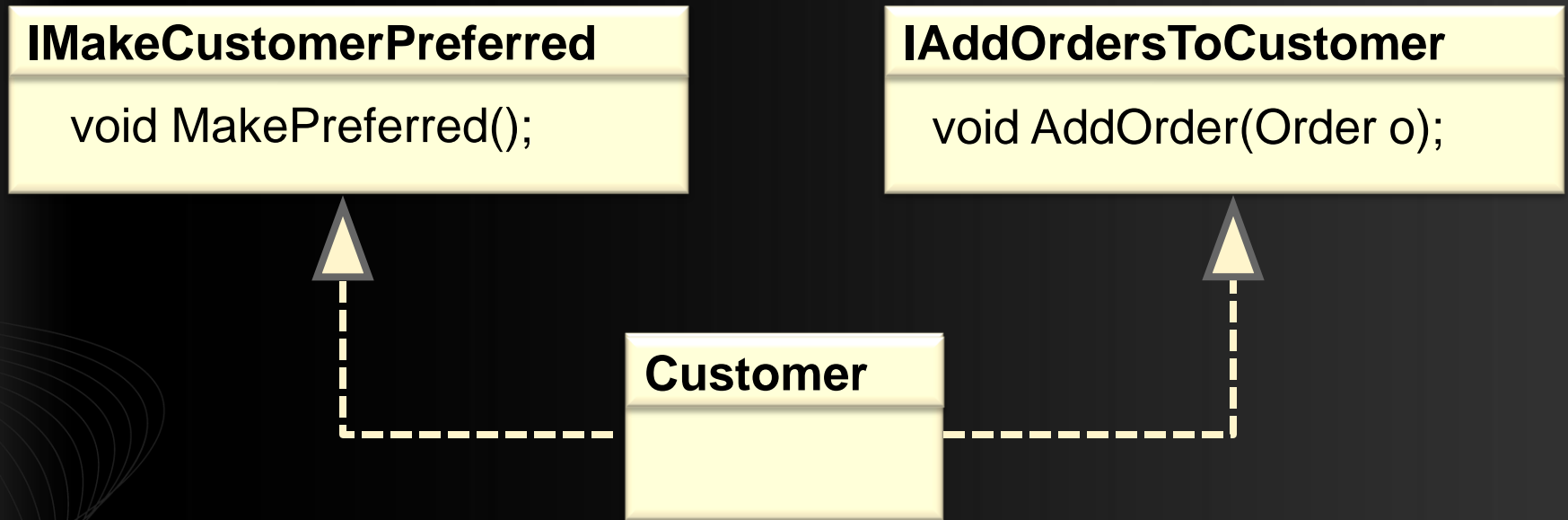
Customer

I want it all

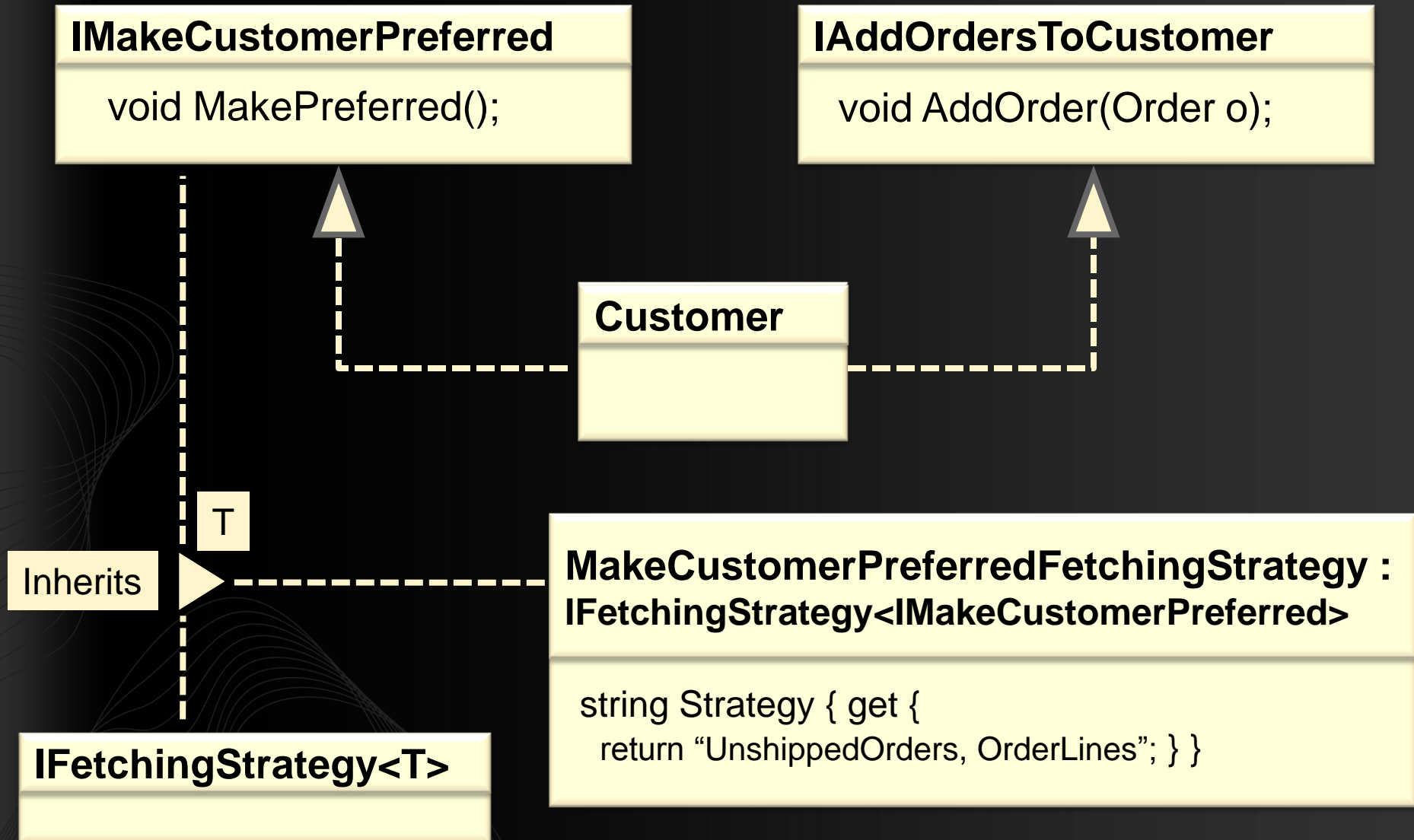


Pretty please?

Using interfaces to differentiate roles



Specifying behavior per role

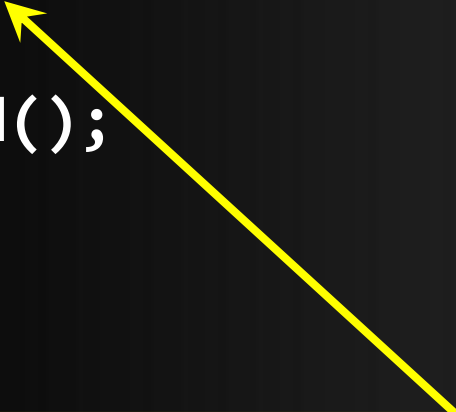


Updated Service Layer

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

        c.MakePreferred();

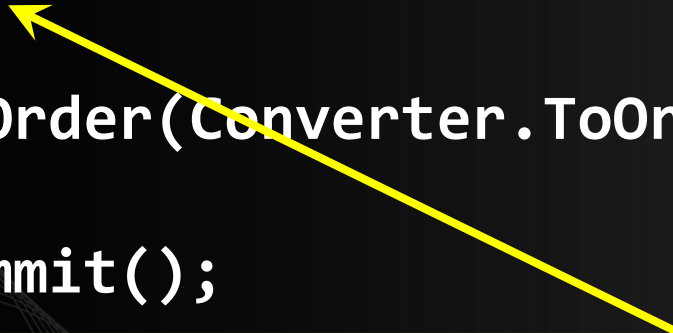
        tx.Commit();
    }
}
```



Uses strategy from class that implements
`IFetchingStrategy<IMakeCustomerPreferred>`

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();  
    }  
}
```



Regular “Get” when can’t find implementation of
IFetchingStrategy<IAddOrdersToCustomer>



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

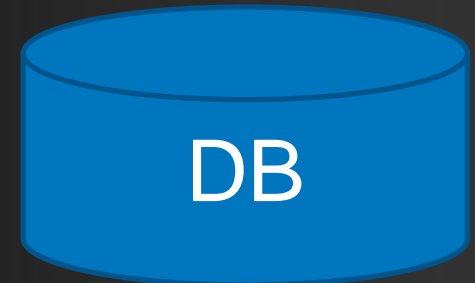
Joins hurt scalability: De-Normalize



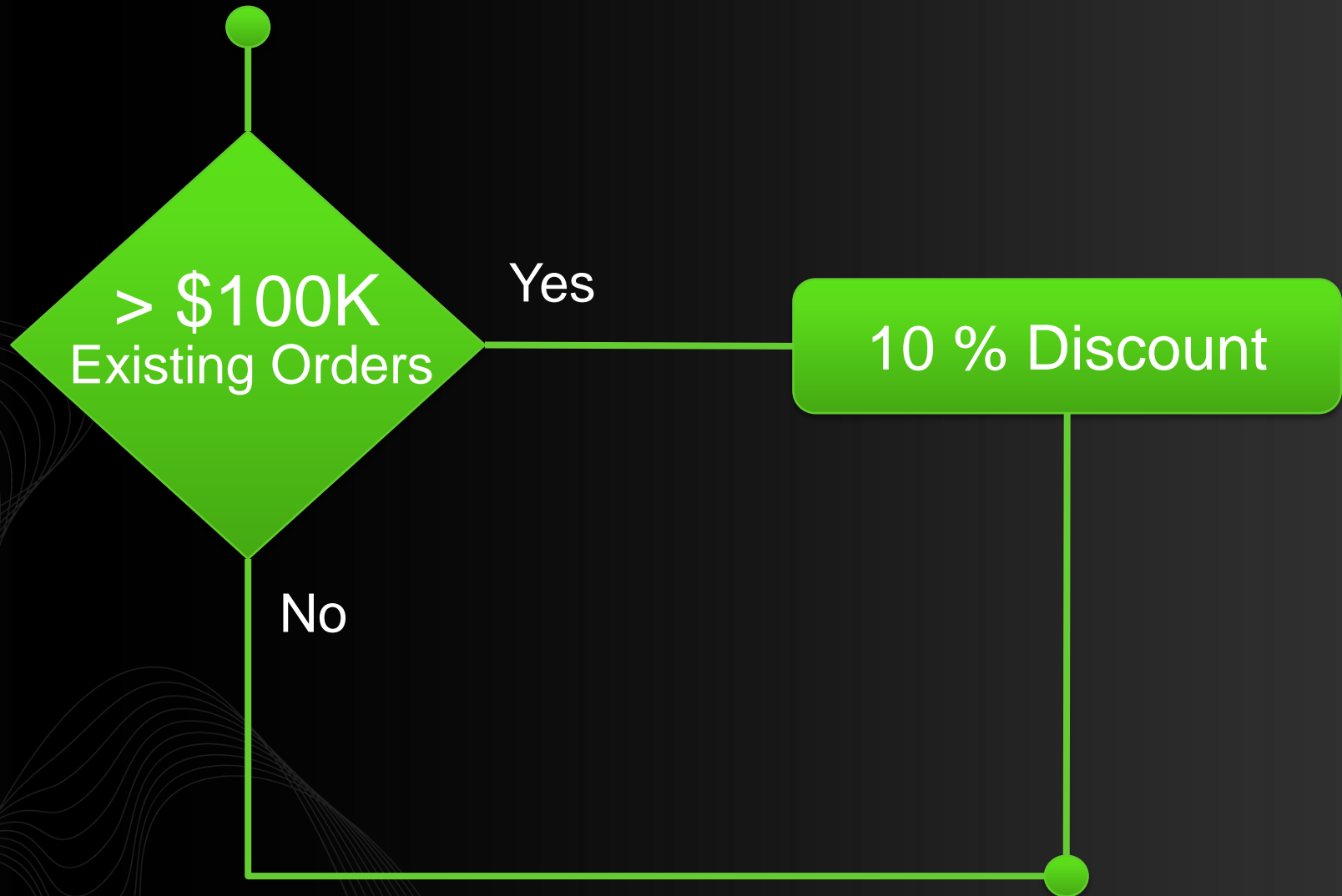
Keep the same interface,
change internals

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;
}
```



Can eagerly fetch, but
will bring a lot of data

Introduce member to replace iteration

Σ

sumOfShippedOrders



Update when orders ship

+

Use when adding orders

Update
ORM

Persistent





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>