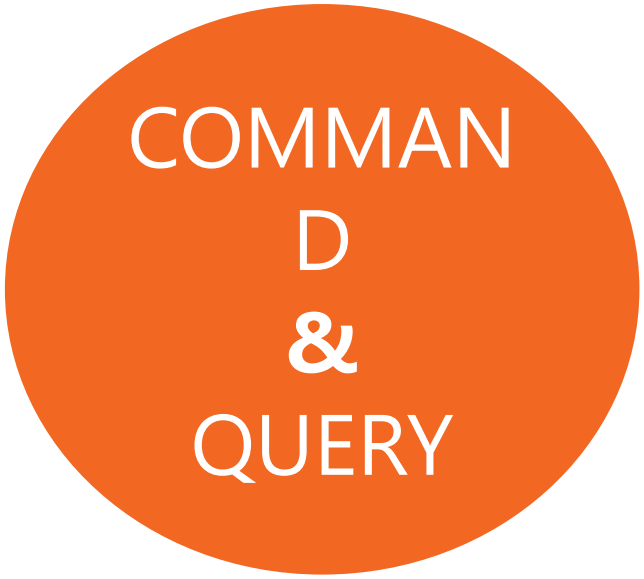
The CQRS Supporting Architecture

Dino Esposito
AUTHOR

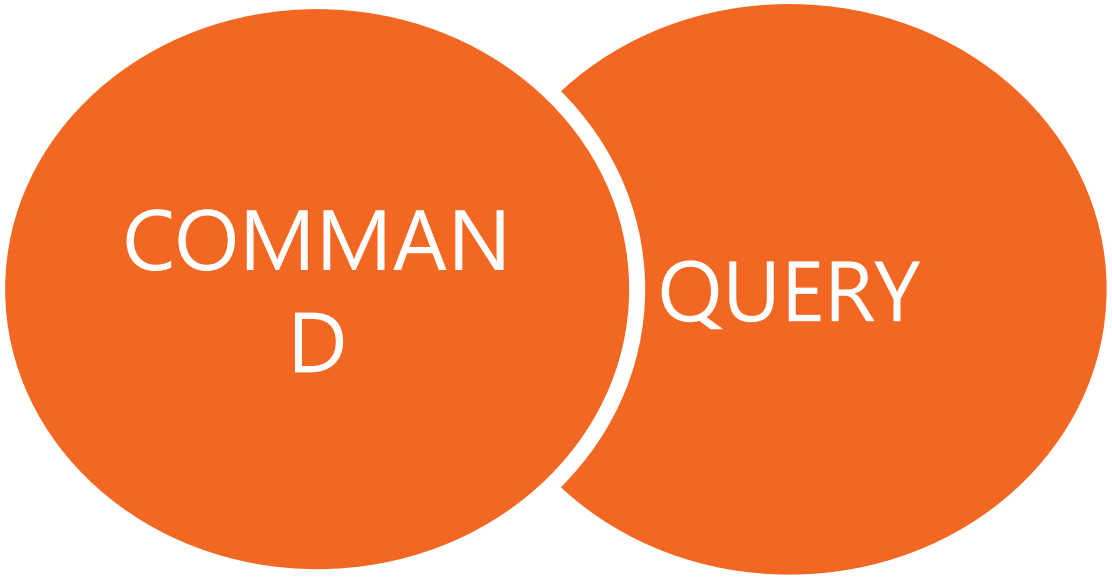
@despos www.software2cents.wordpress.com

All That Everybody Wanted Was **Software Model for the Business Domain**

Classic



New

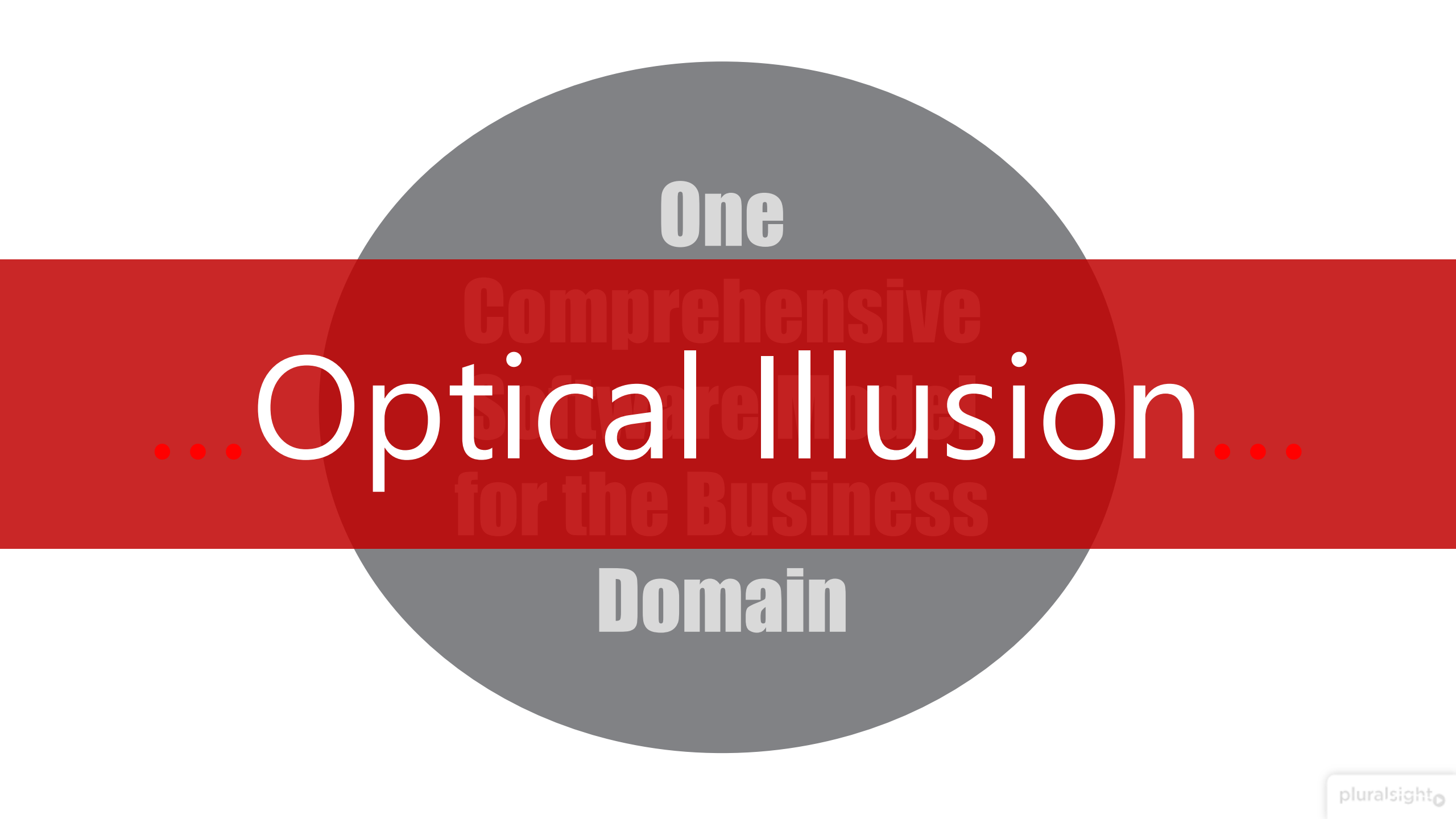


Key Points

CQRS
Regular

CQRS
Premium

CQRS
Deluxe


One
Comprehensive
Software Model
for the Business
Domain
...Optical Illusion...

Great for **commands**

```
public class Match
{
    public Match( ... ) { ... }
```

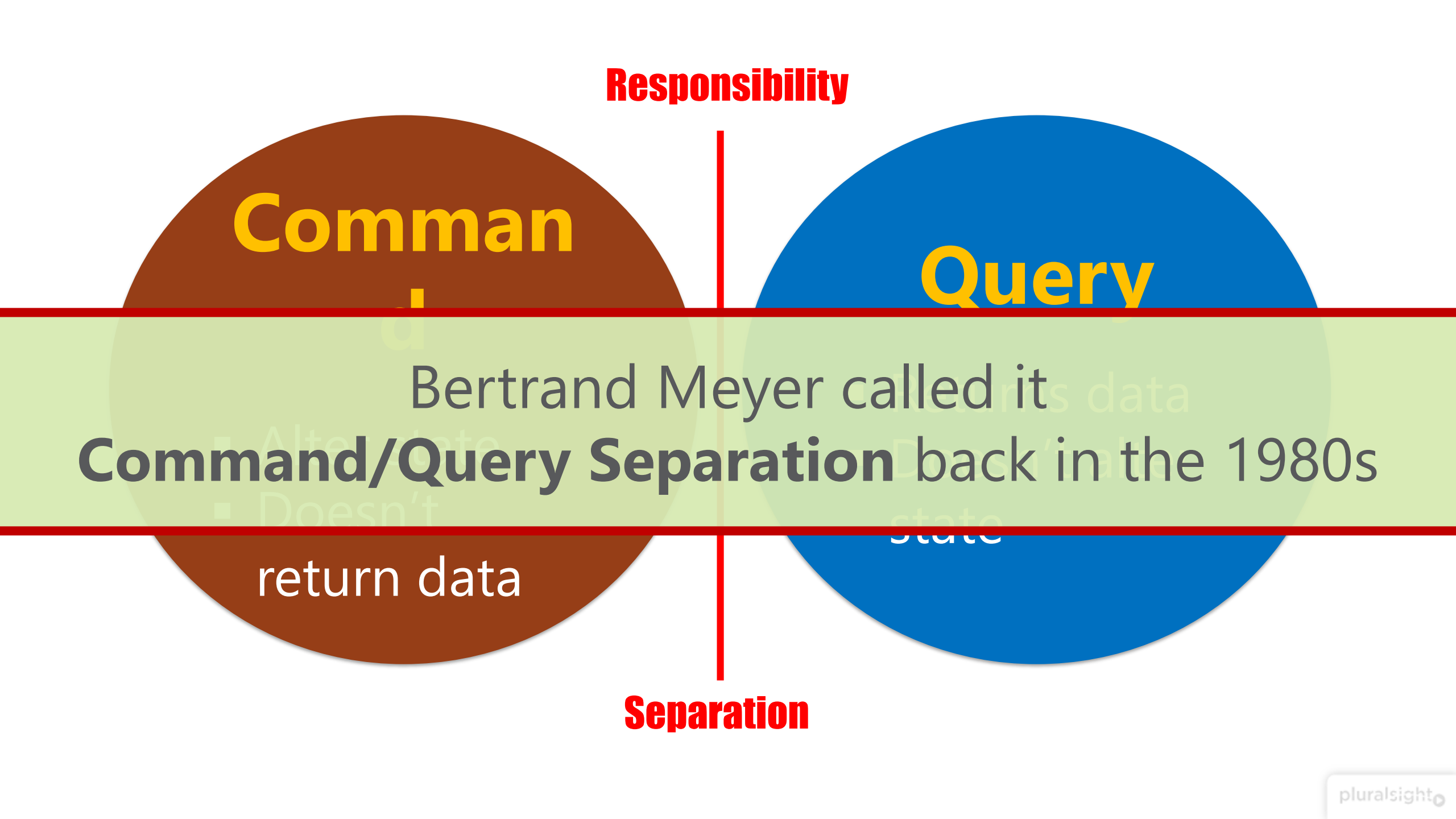
- Requires fixes for persistence
- Exposes behavior to presentation

Great for **queries**

```
public class Match
{
    public Score Score { get; set; }
    public int Period { get; set; }
```

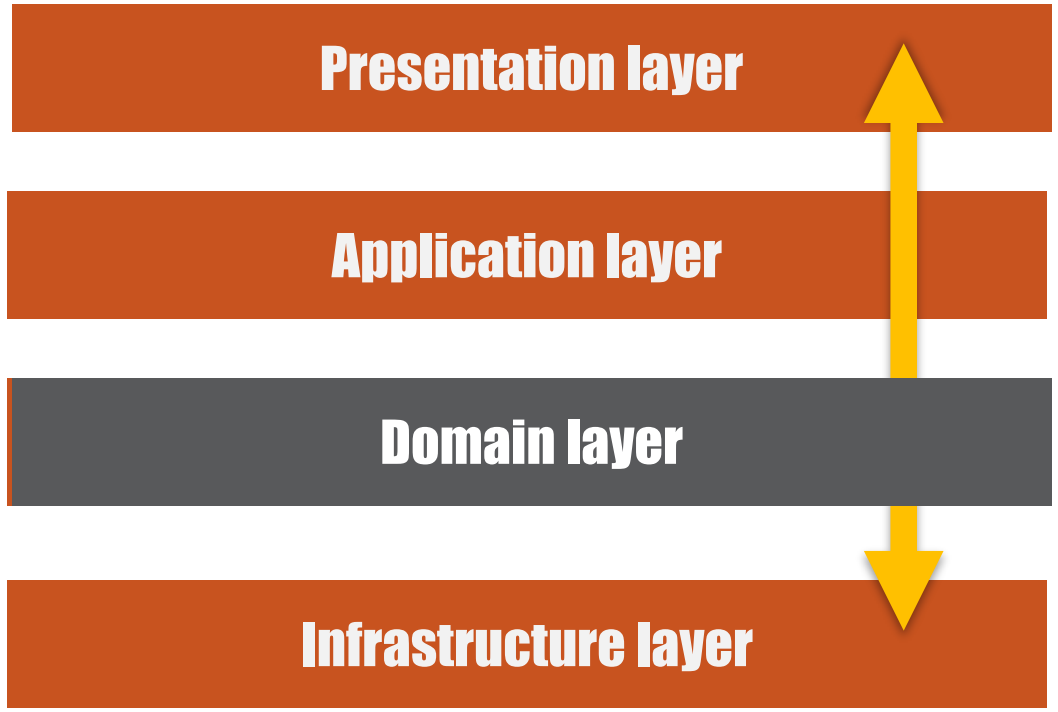
- No business rules in the class
- Risk of getting into incongruent state

CQRS

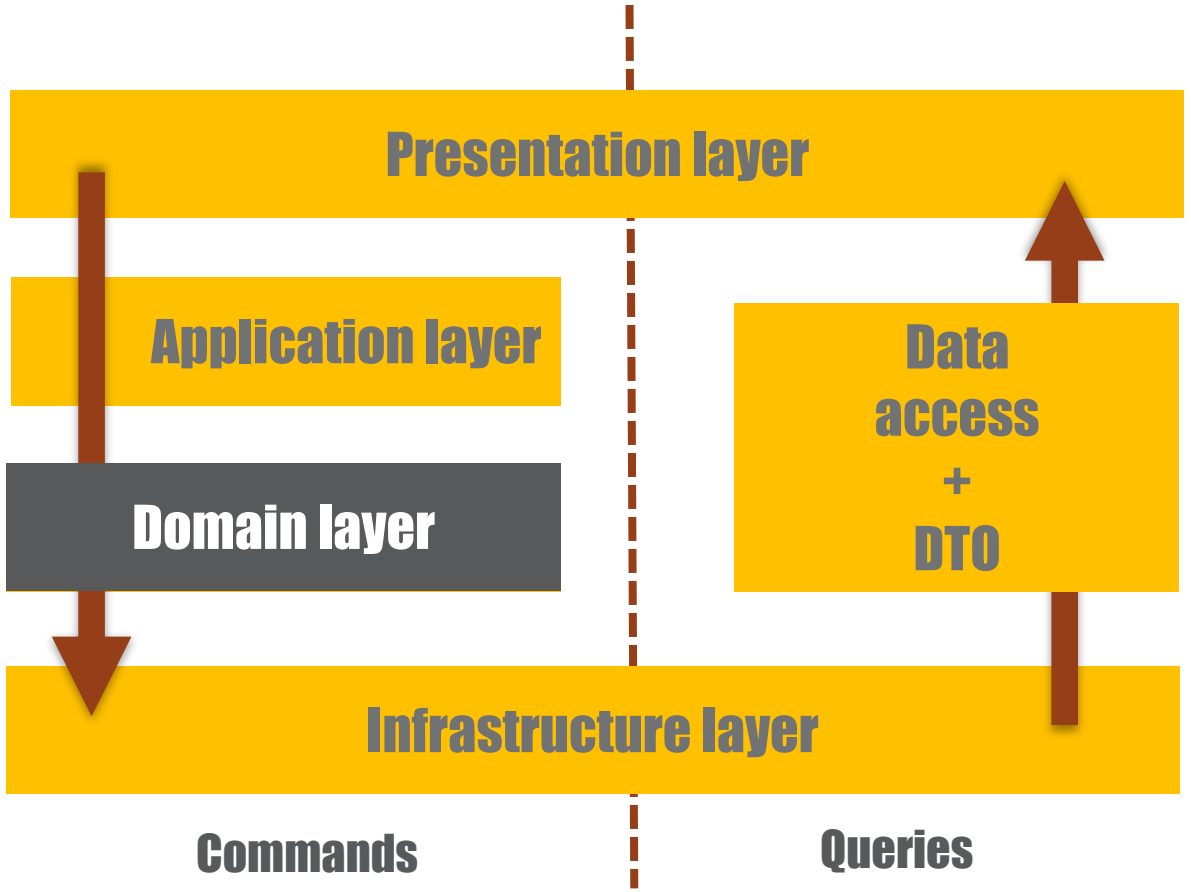


Bertrand Meyer called it
Command/Query Separation back in the 1980s

Canonical
Layered Architecture



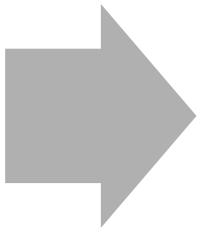
CQRS



Aspects of **CQRS**

Benefits

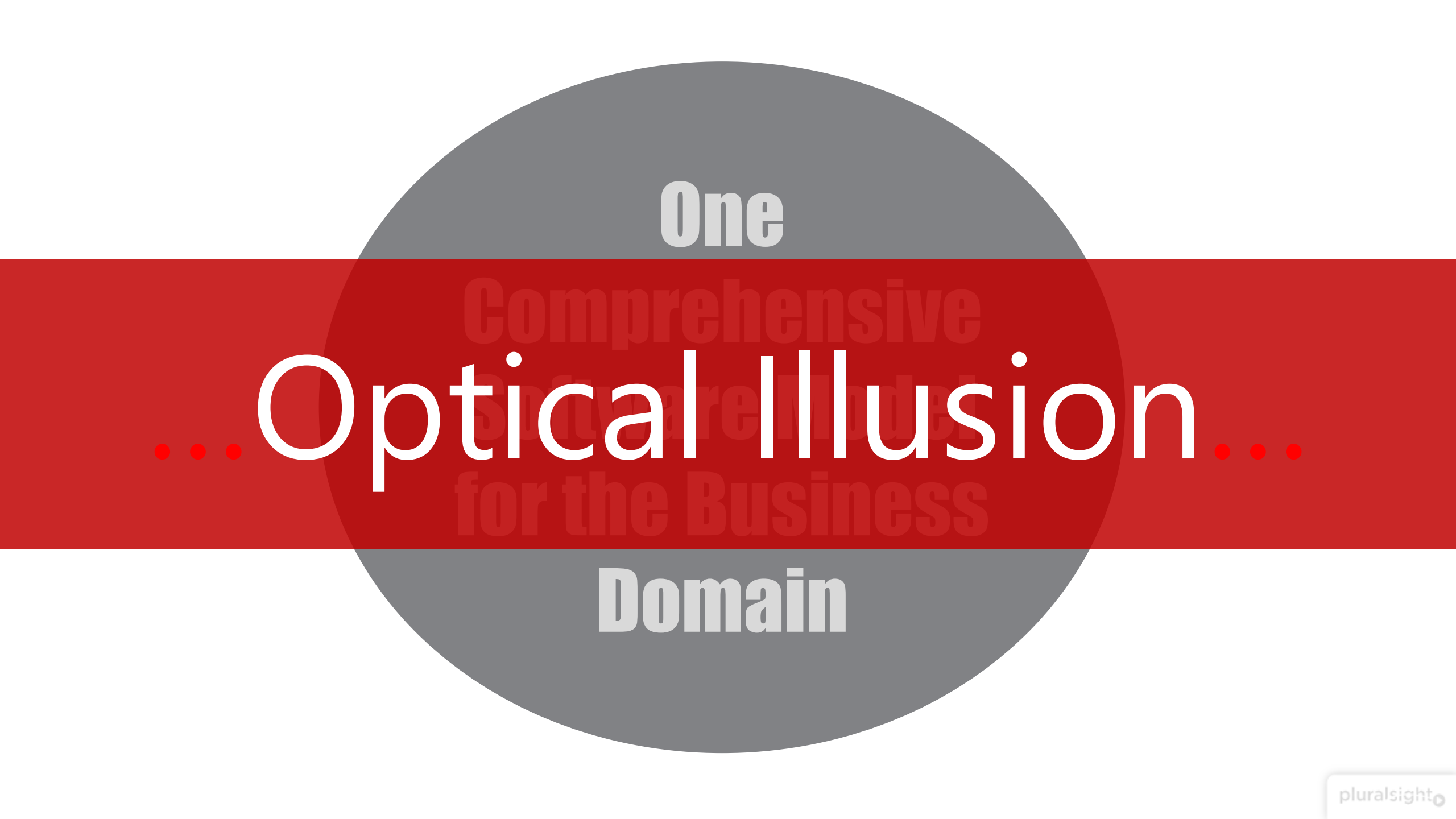
- Distinct **optimization**
- **Scalability** potential



Side effects

- **Simplified** design
- Hassle-free stacks **enhancement**

Flavors of CQRS


One
Comprehensive
Software Model
for the Business
Domain
...Optical Illusion...

Great for **commands**

```
public class Match
{
    public Match( ... ) { ... }
```

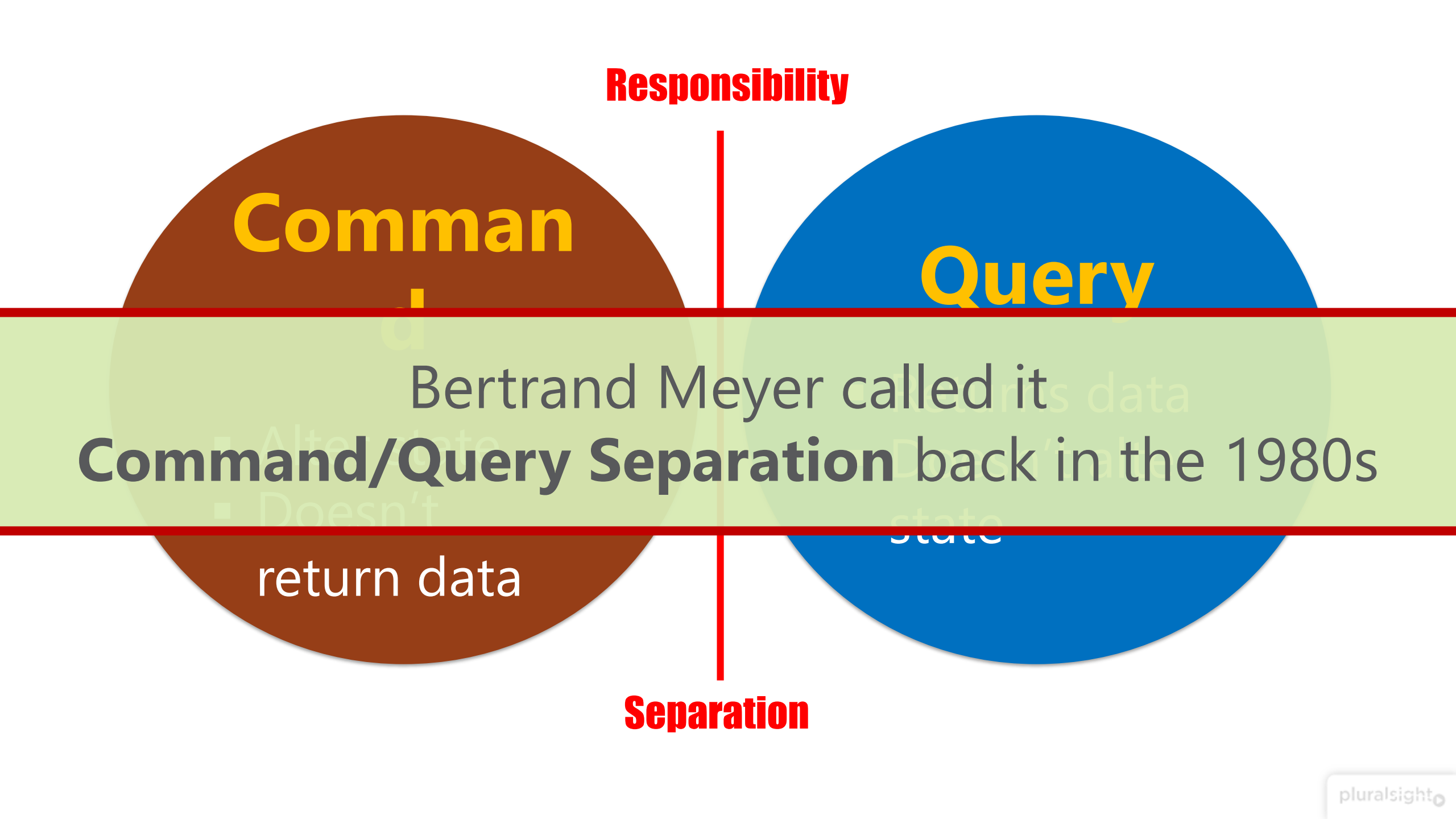
- Requires fixes for persistence
- Exposes behavior to presentation

Great for **queries**

```
public class Match
{
    public Score Score { get; set; }
    public int Period { get; set; }
```

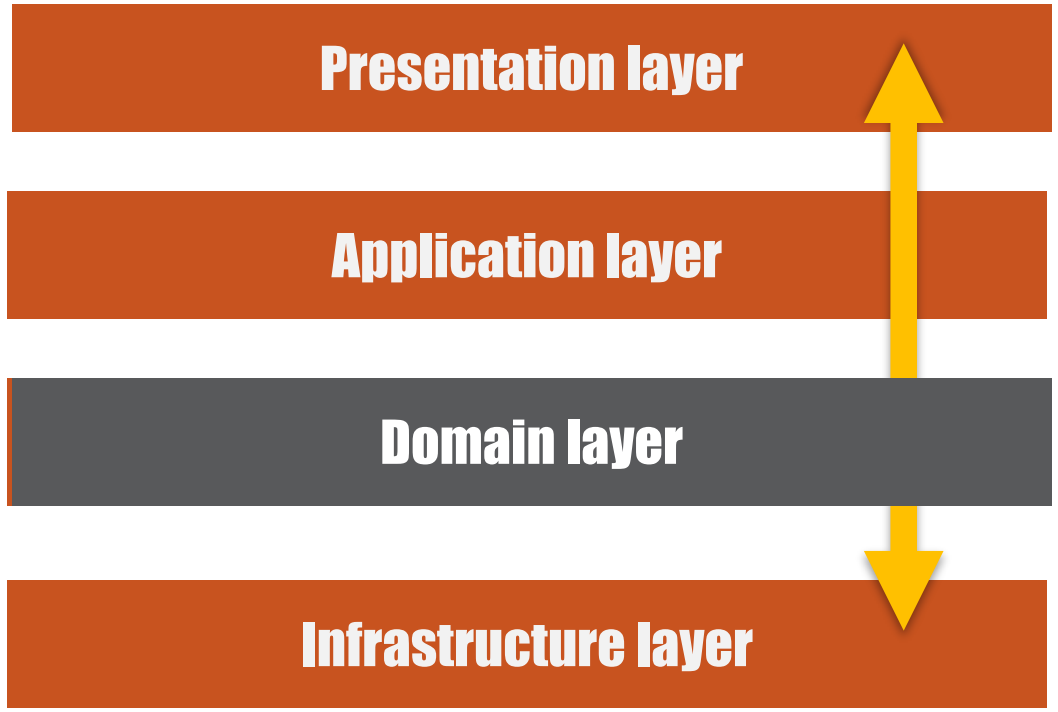
- No business rules in the class
- Risk of getting into incongruent state

CQRS

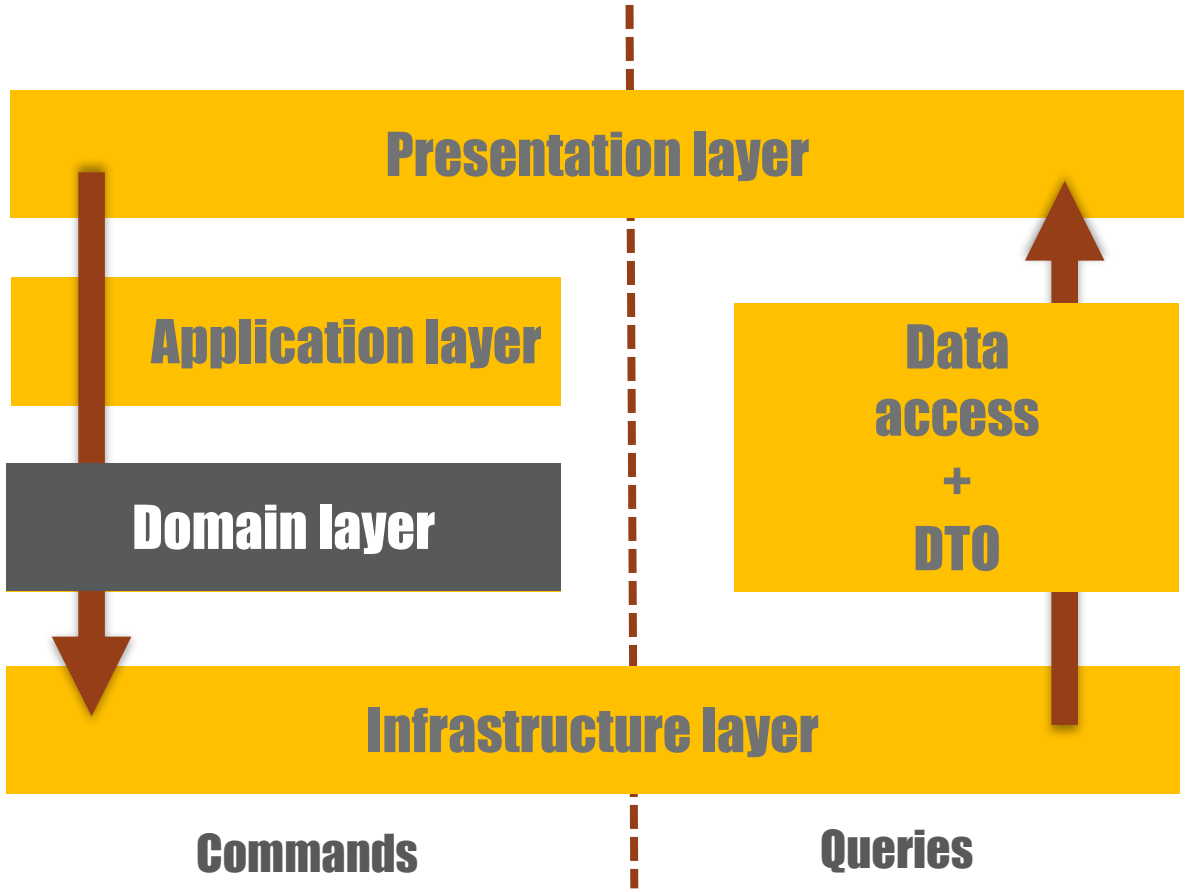


Bertrand Meyer called it
Command/Query Separation back in the 1980s

Canonical
Layered Architecture



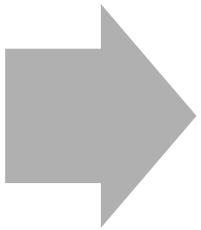
CQRS



Aspects of **CQRS**

Benefits

- Distinct **optimization**
- **Scalability** potential

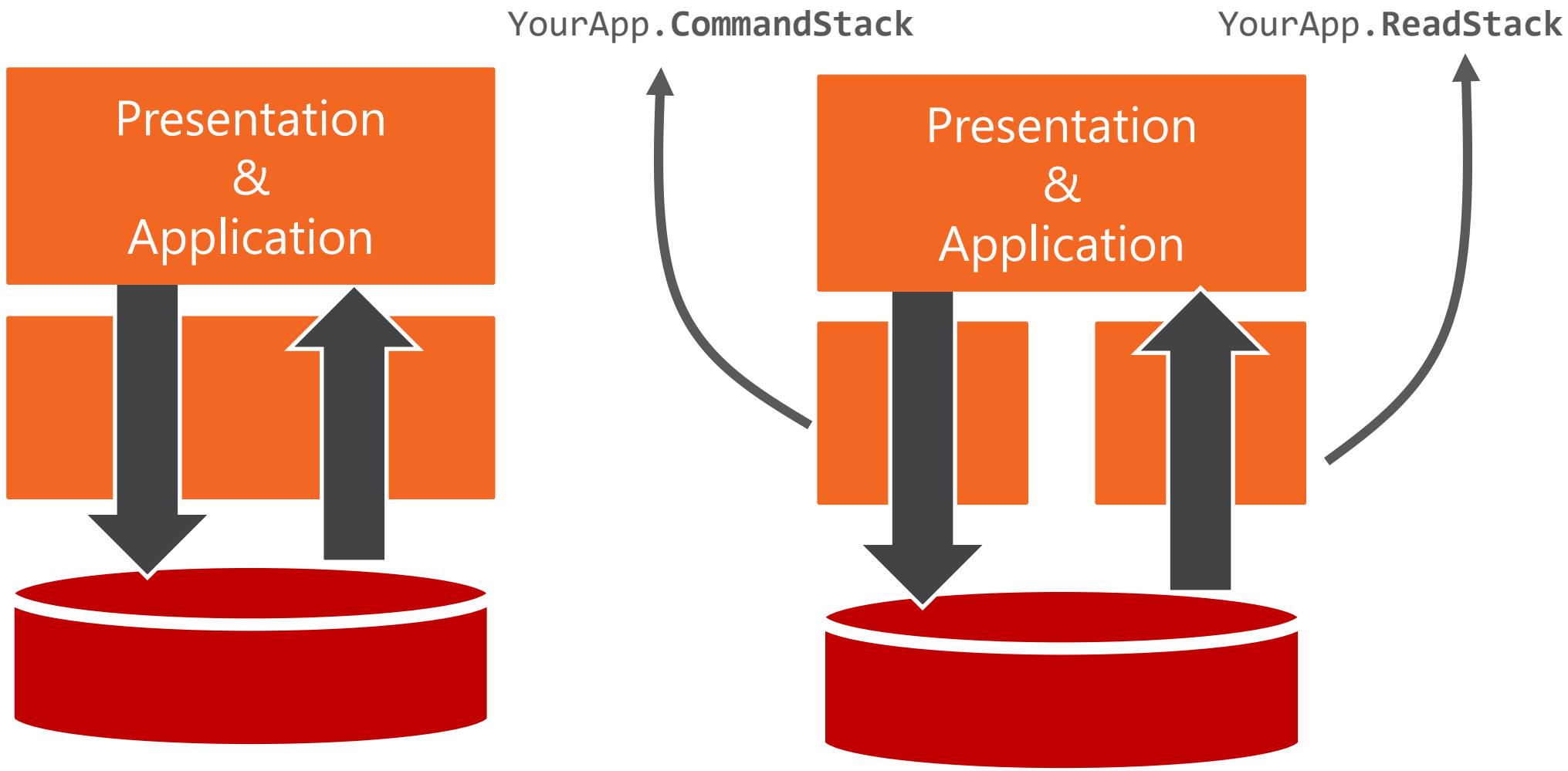


Side effects

- **Simplified** design
- Hassle-free stacks **enhancement**

Flavors of CQRS

CQRS for Plain CRUD Applications



Command Stack

Use just the pattern that fits better

Read Stack

Use just the code that does the job

- O/RM of choice
- LINQ
- Database in use

TIP

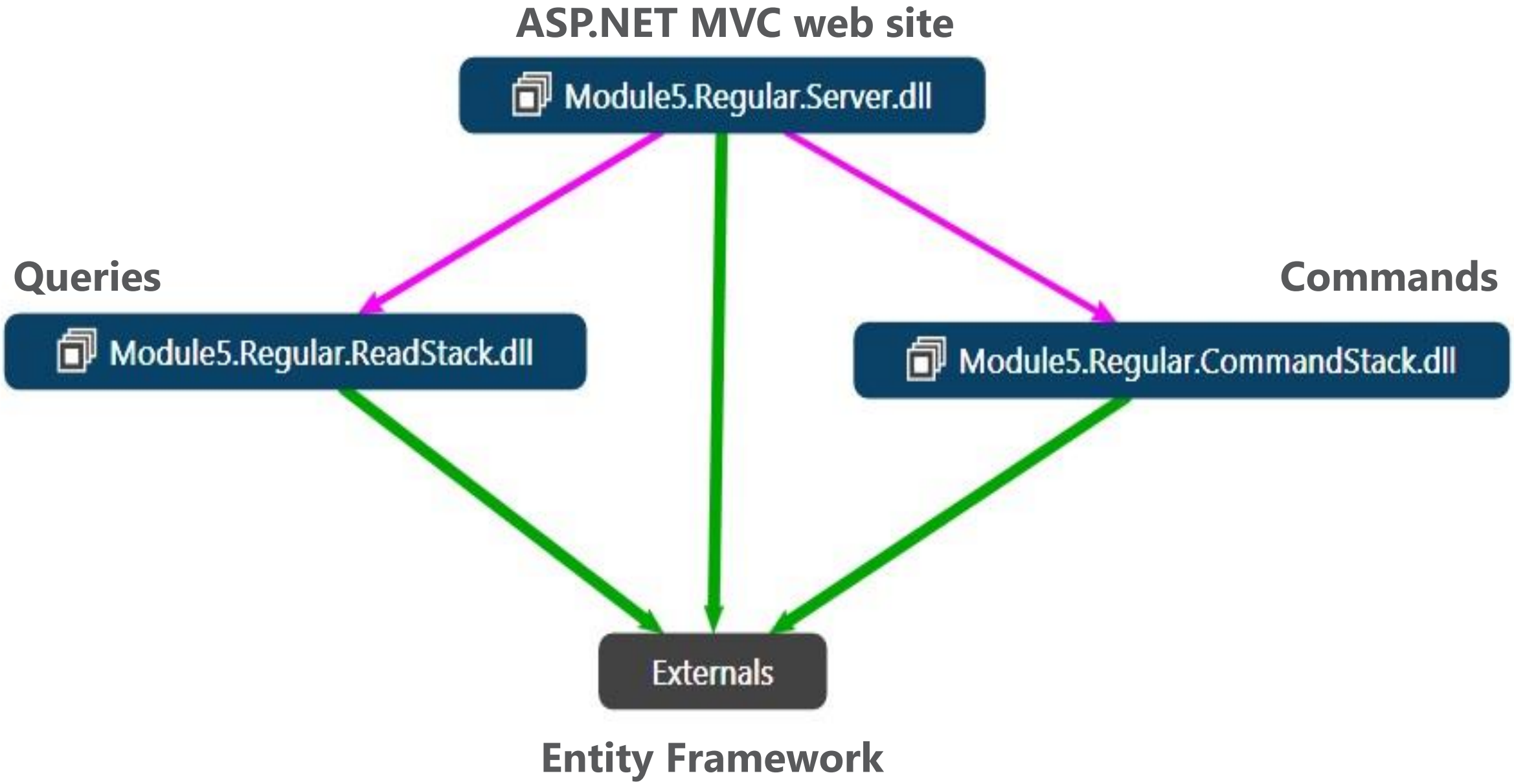
Use a read-only wrapper for the **DbContext** instance you use in the read stack.

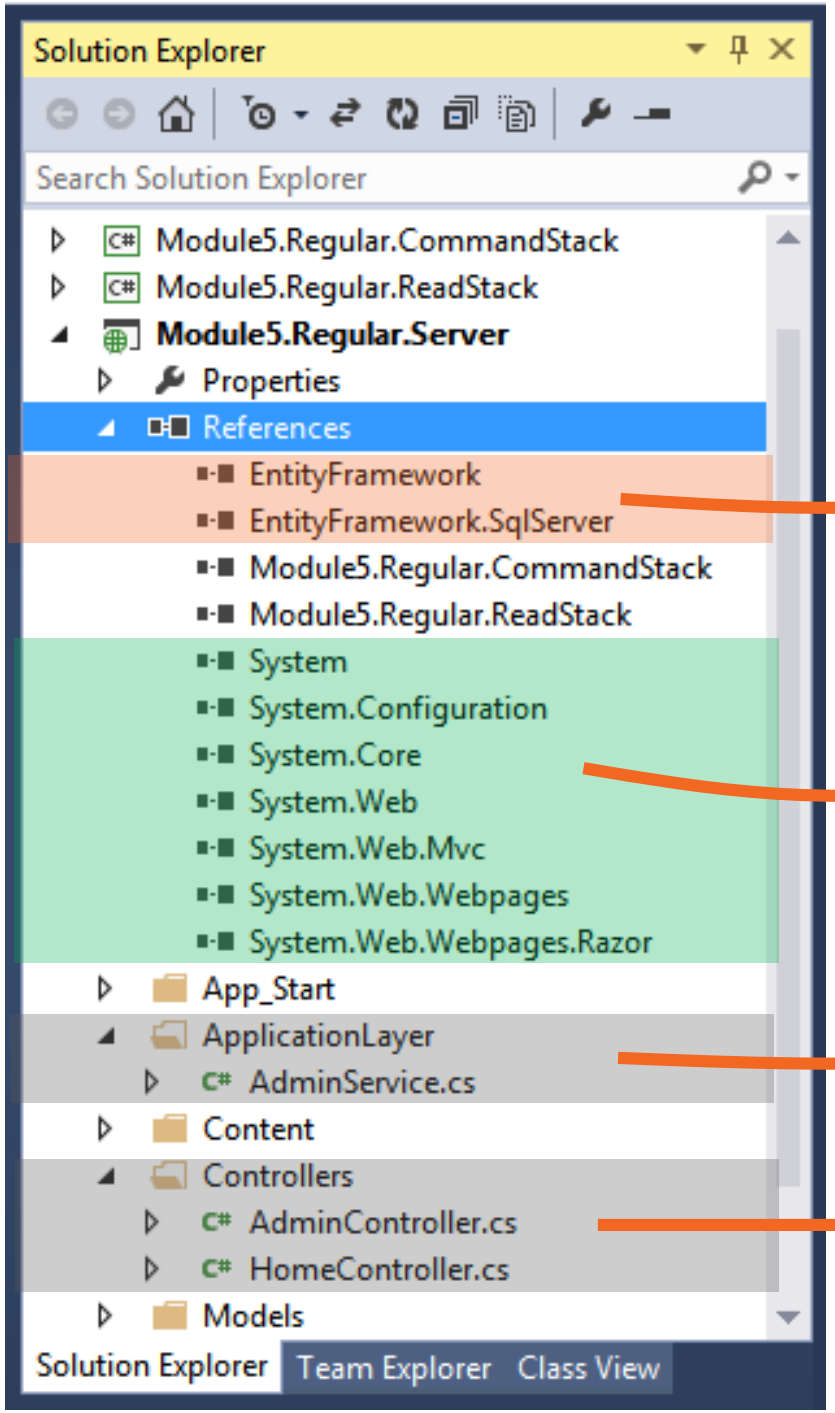
Read-only Database Facade

```
public class Database : IDisposable
{
    private readonly QueryDbContext _db = new QueryDbContext();
    public IQueryable<Customer> Customers
    {
       get { return _db.Customers; }
    }
    public void Dispose()
    {
       _db.Dispose();
    }
}
```

DEMO

CQRS Regular
in action


ASP.NET MVC web site
Module5.Regular.Server.dll
Queries
Commands
Module5.Regular.ReadStack.dll
Module5.Regular.CommandStack.dll
Externals
Entity Framework

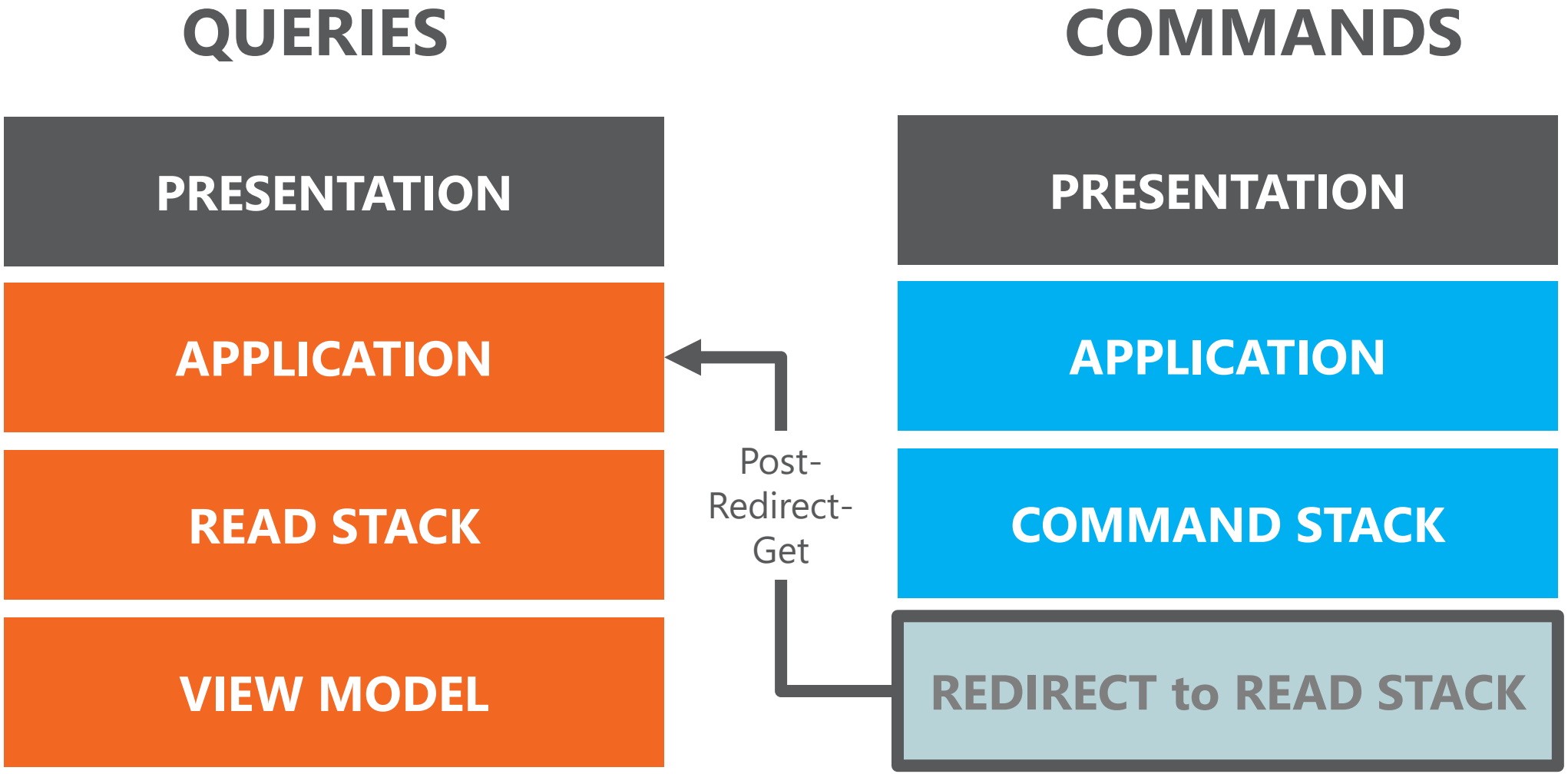


Externals

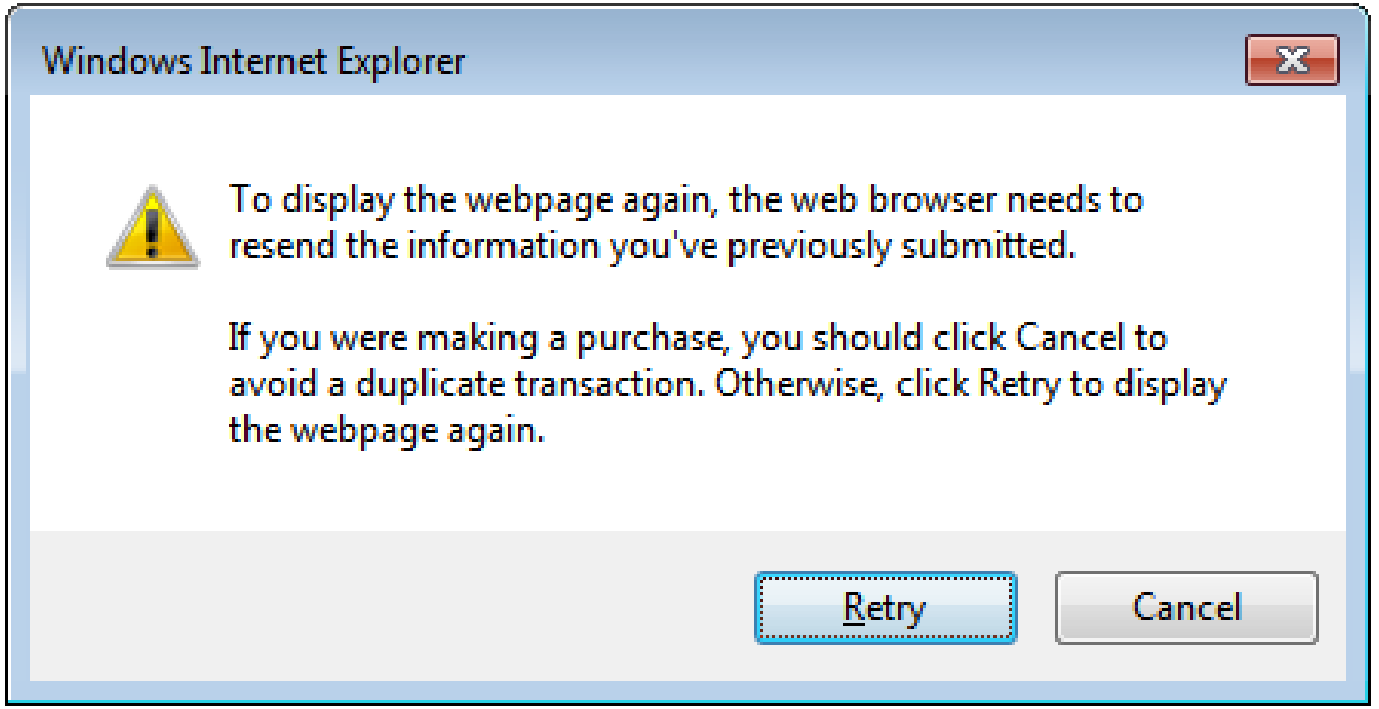
ASP.NET

Application Layer

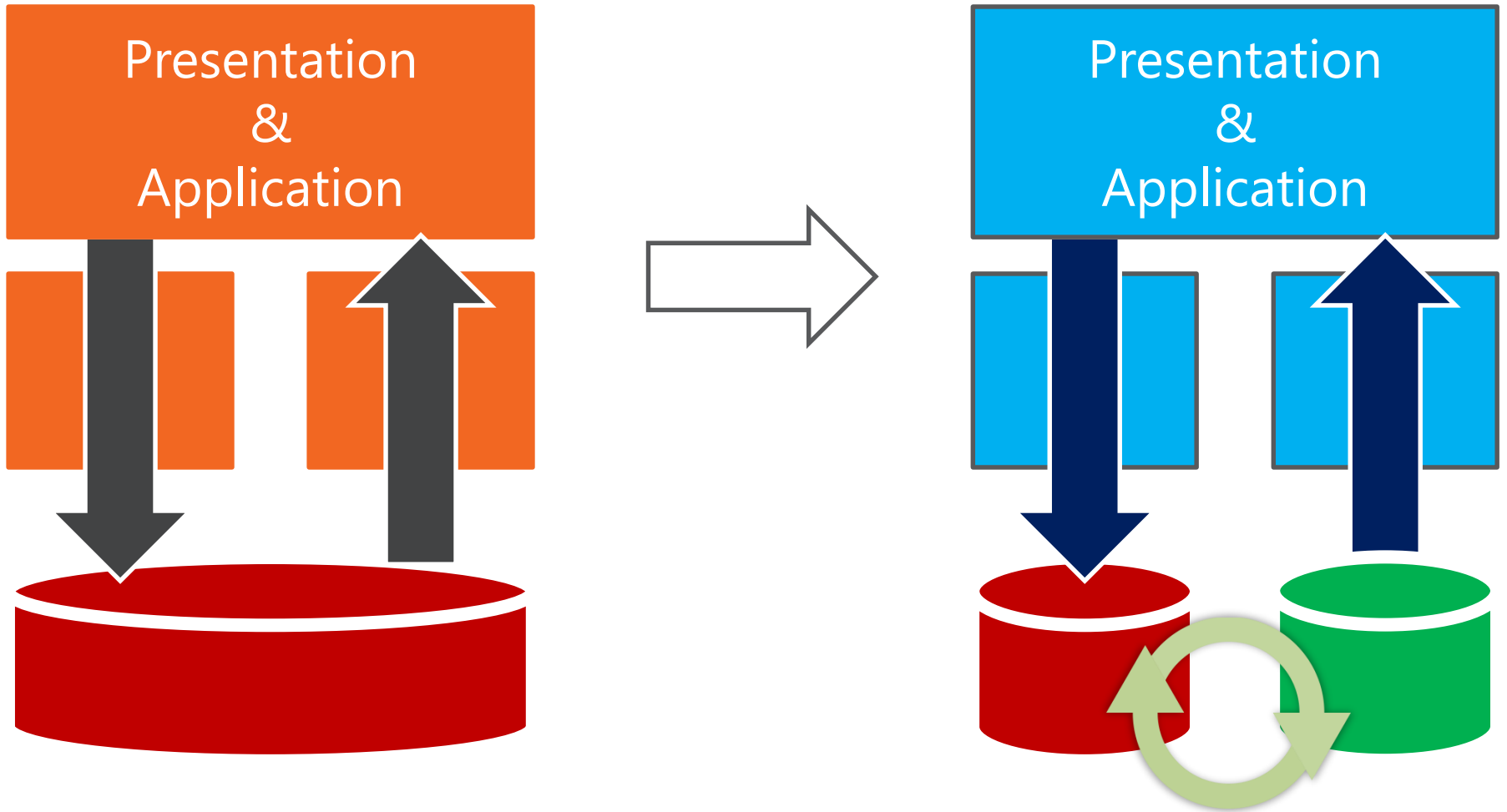
Presentation Layer



CQRS and **Post-Redirect-Get** web pattern



CQRS Premium



Command Stack

Use just the pattern that fits better

Read Stack

Use just the code that does the job

O/RM of choice

LINQ

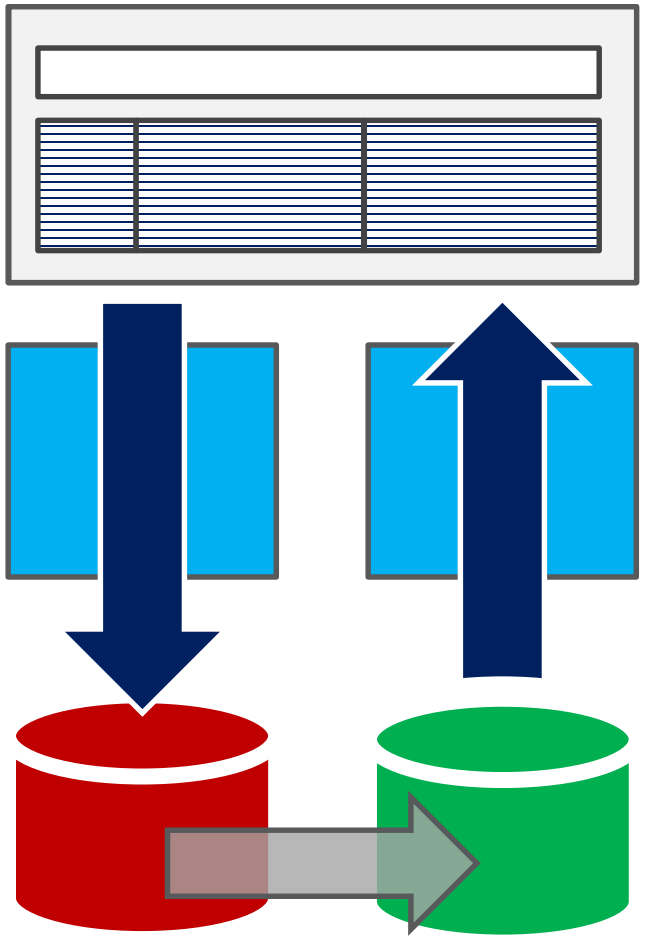
Ad-hoc storage

Relational

ISSUE

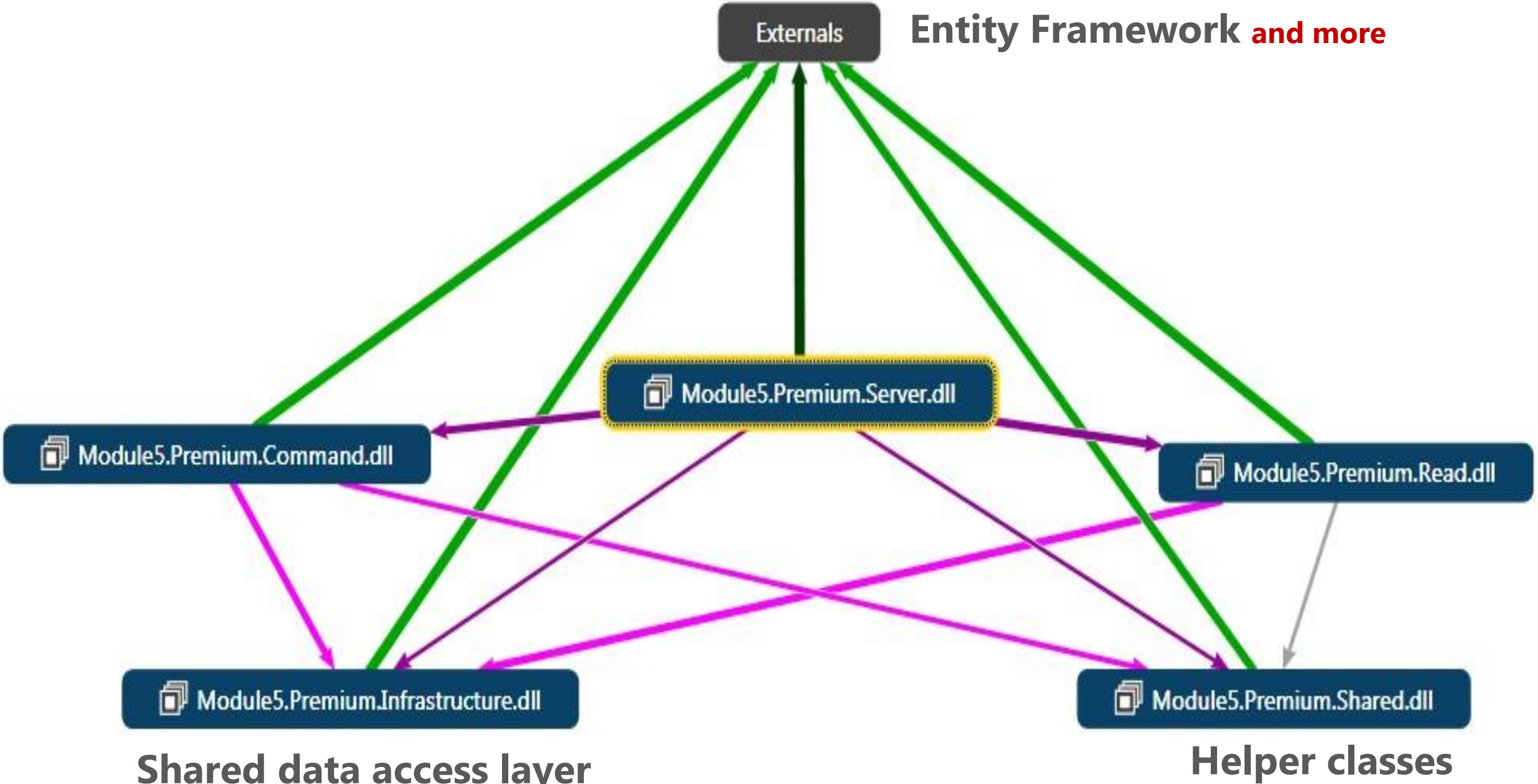
How would you define stale data? How critical is stale data for the app?

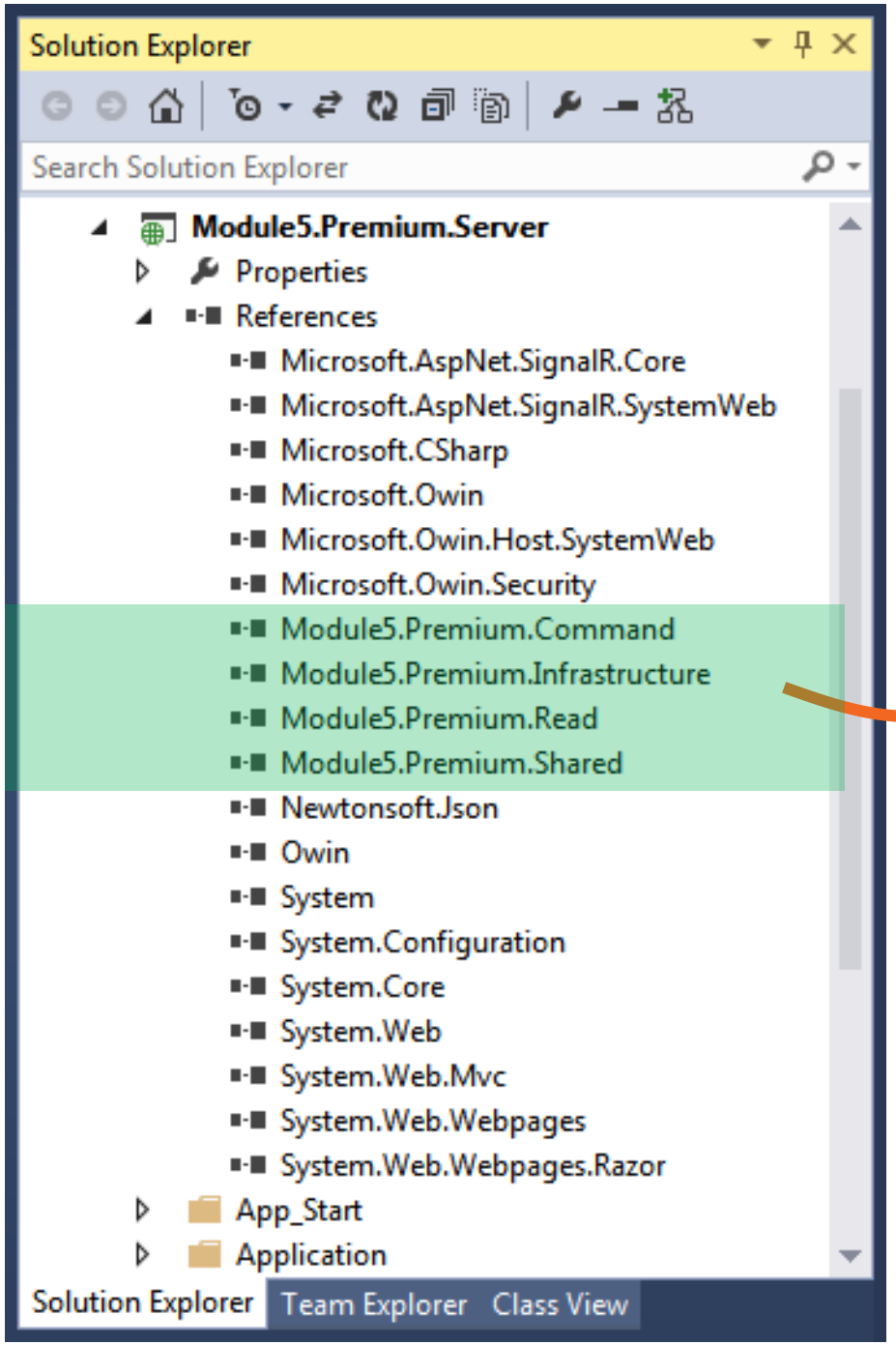
Command & Query **Storage**



Command & Query **Storage Synchronization**

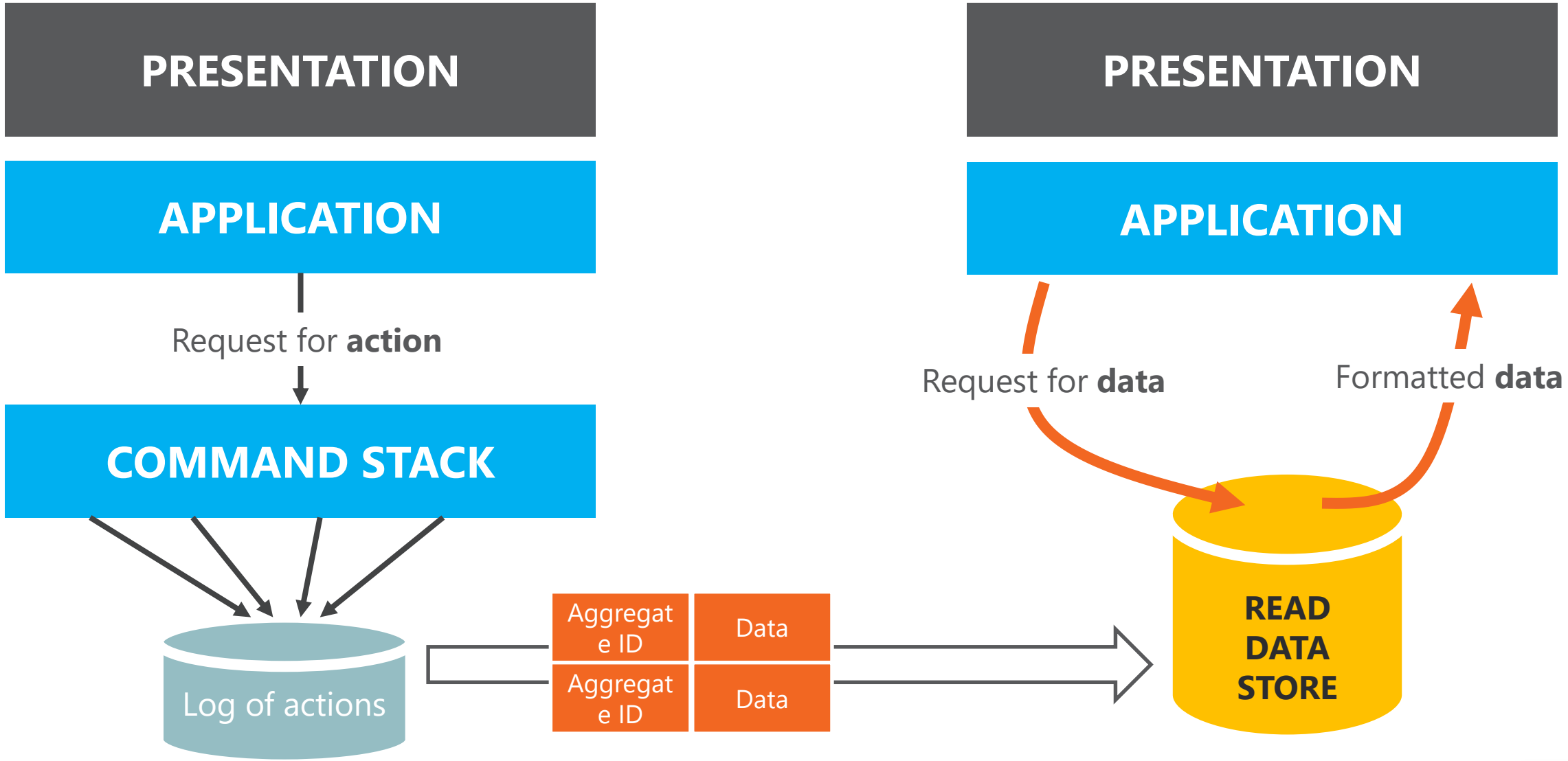
Automatically up-to-date	**Eventually up-to-date**	**Controlled staleness**	**Controlled up-to-date**
Synchronous	Asynchronous	Scheduled	On-demand
Every command triggers sync updates	Every command triggers async updates	A job runs periodically and updates the read storage	Updates triggered by requests (if older enough)


Externals
Entity Framework and more
Module5.Premium.Server.dll
Module5.Premium.Command.dll
Module5.Premium.Read.dll
Module5.Premium.Infrastructure.dll
Module5.Premium.Shared.dll
Shared data access layer
Helper classes



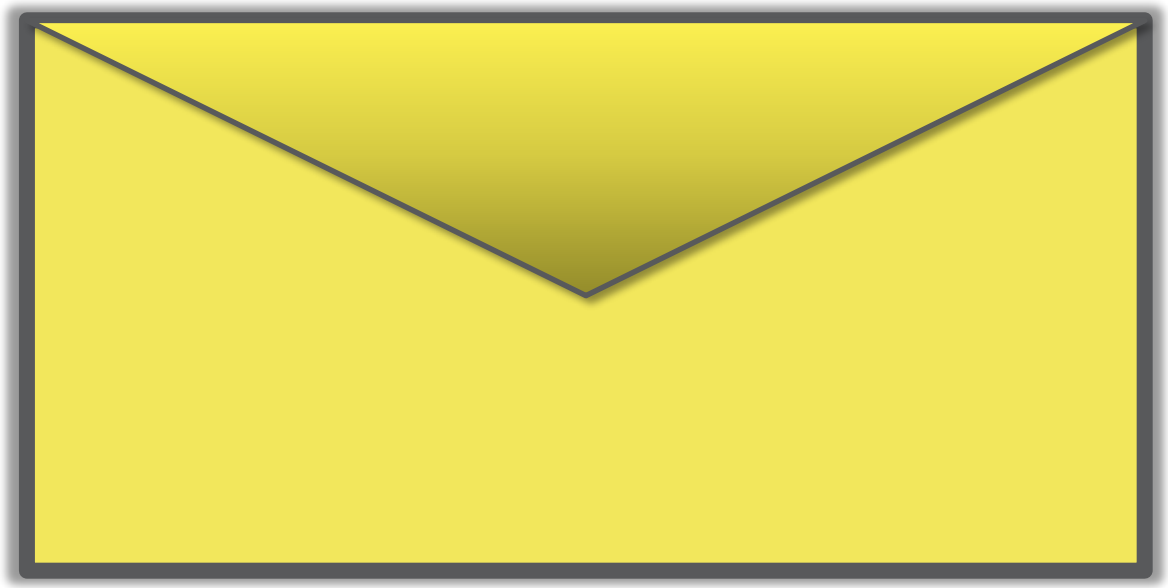
Command stack
Read stack
Infrastructure layer

SYNCHRONIZATION



DEMO

Power to Messages

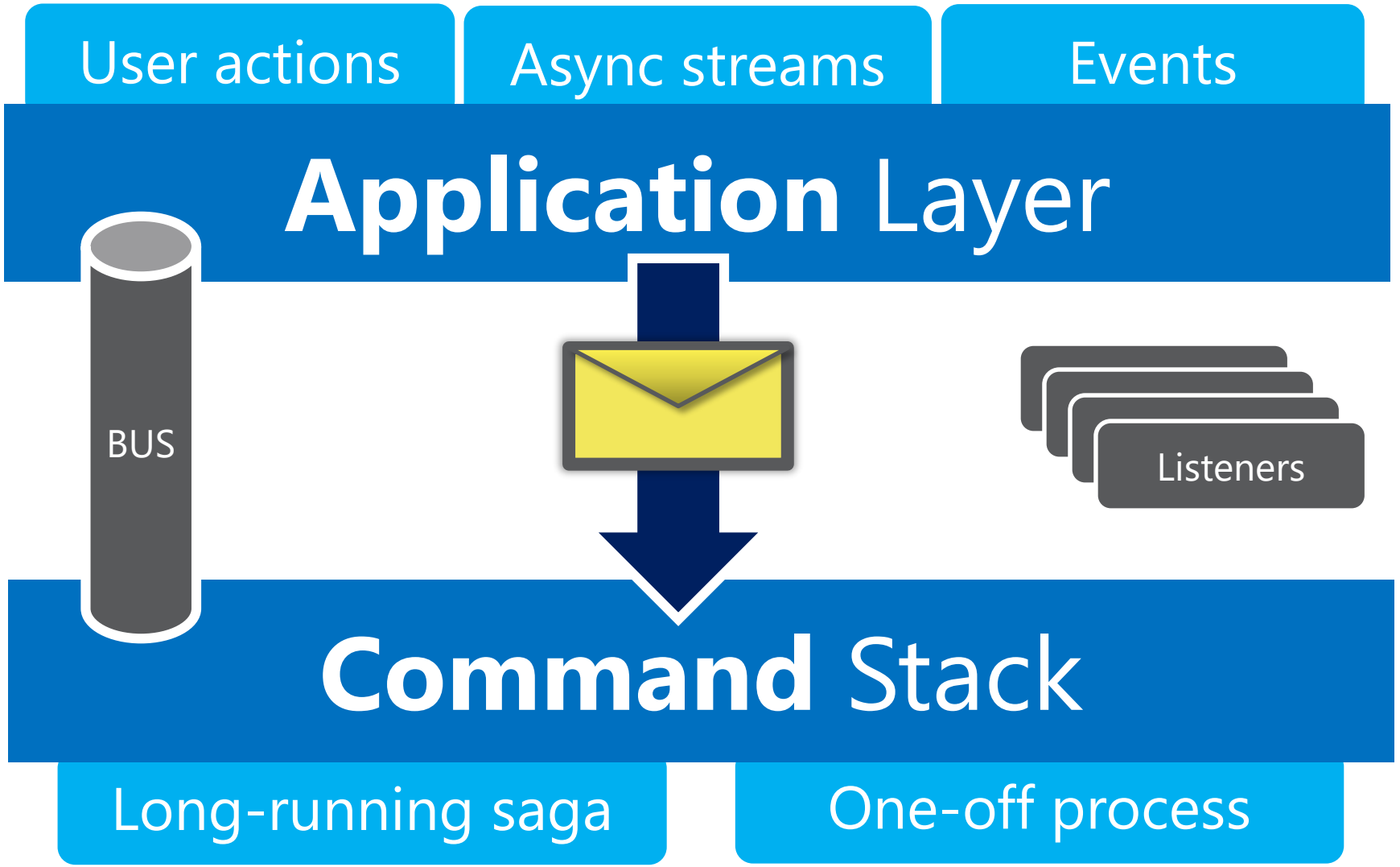


Representing Messages

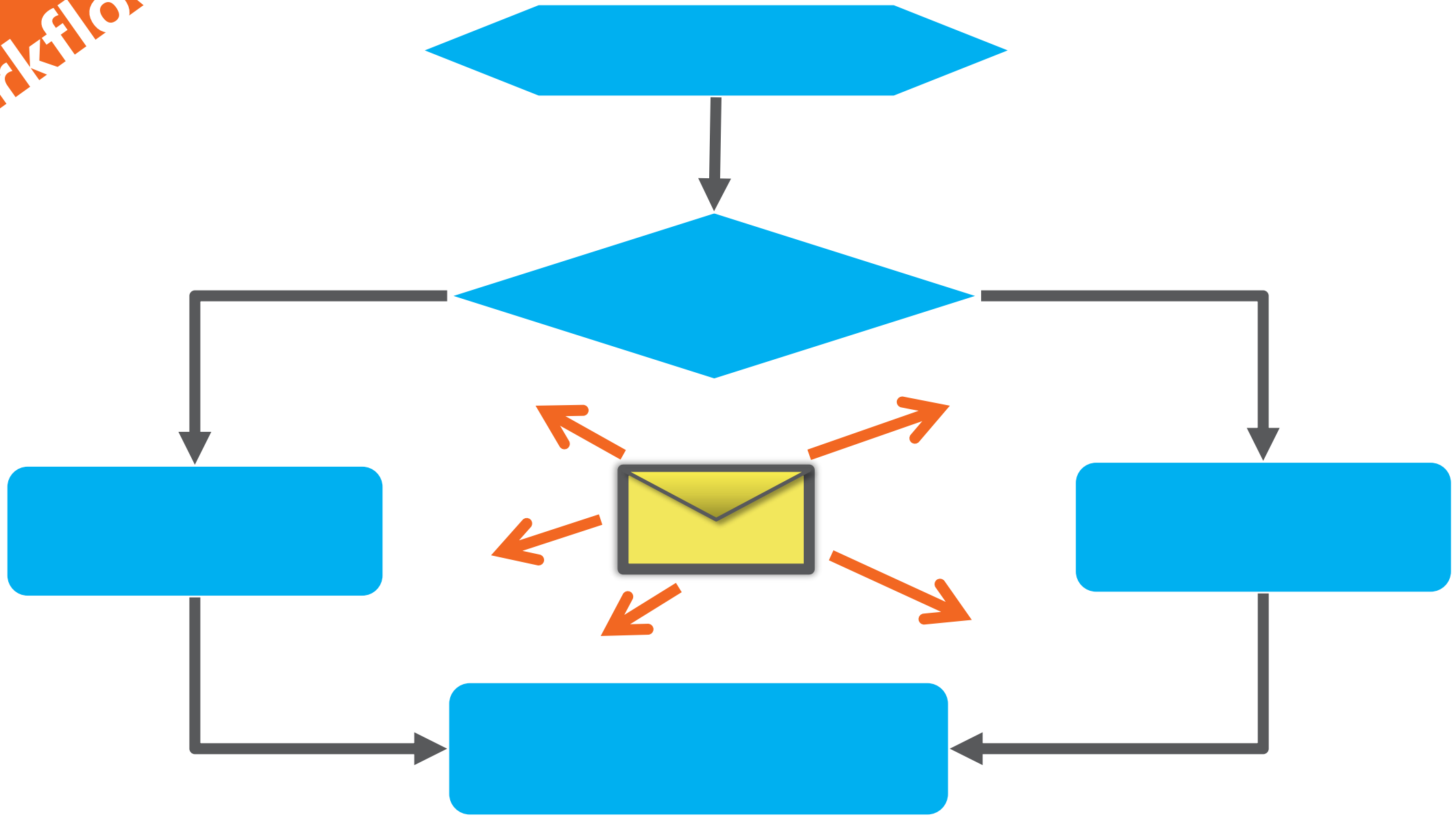
```
public class Message
{
    public DateTime TimeStamp { get; protected set; }
    public String SagaId { get; protected set; }
}

public class Command : Message
{
    public String Name { get; protected set; }
}

public class Event : Message
{
    // Any properties that may help retrieving and persisting events.
}
```

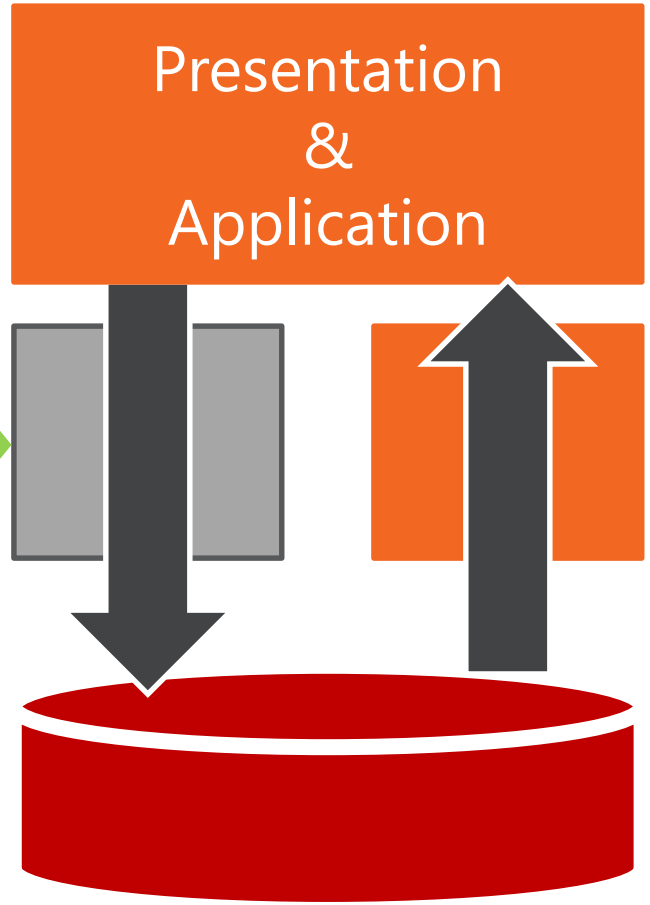

User actions
Async streams
Events
Application Layer
BUS
Listeners
Command Stack
Long-running saga
One-off process

Workflow



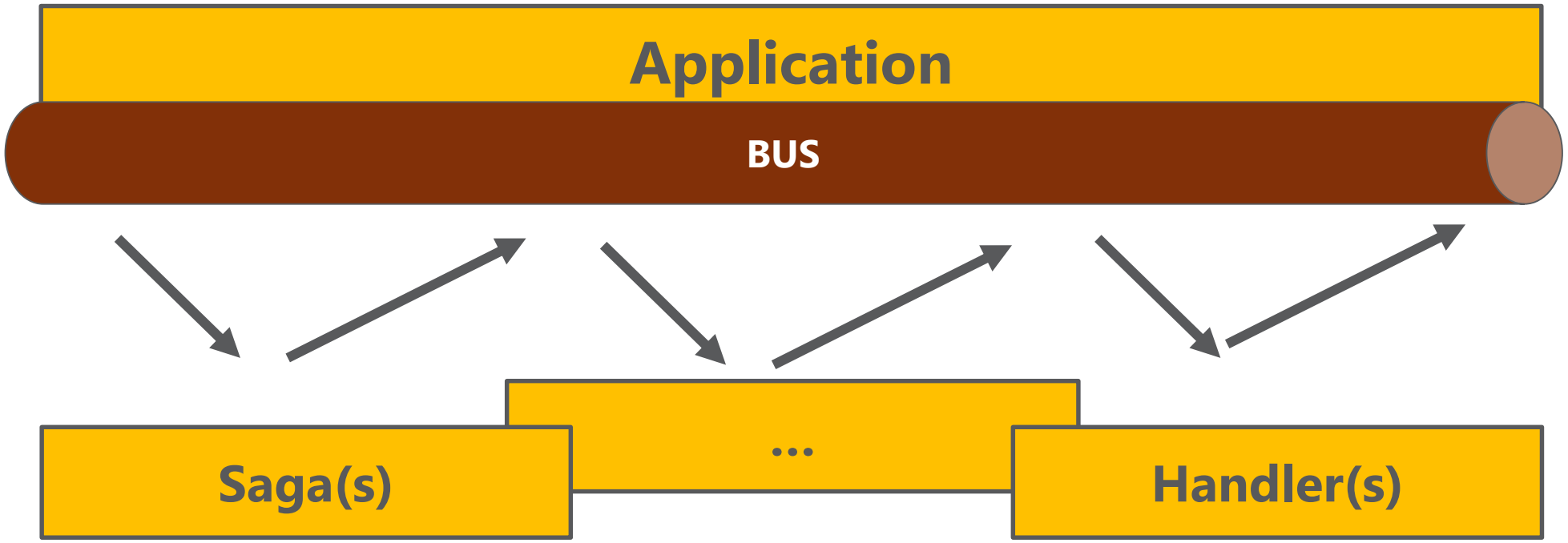
CQRS **Deluxe**

MESSAGE-based
ORCHESTRATION
OF BUSINESS TASKS



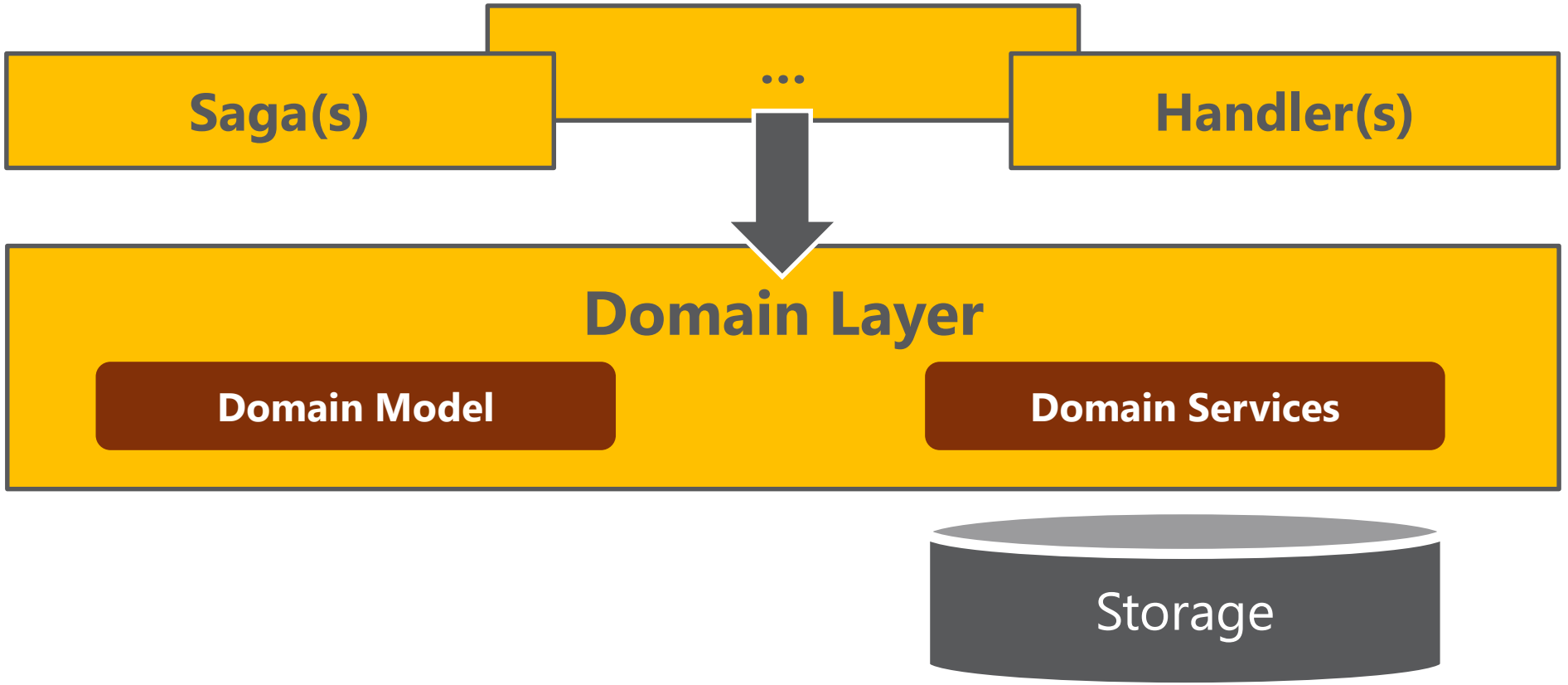
Command Stack

Use just the pattern that fits better

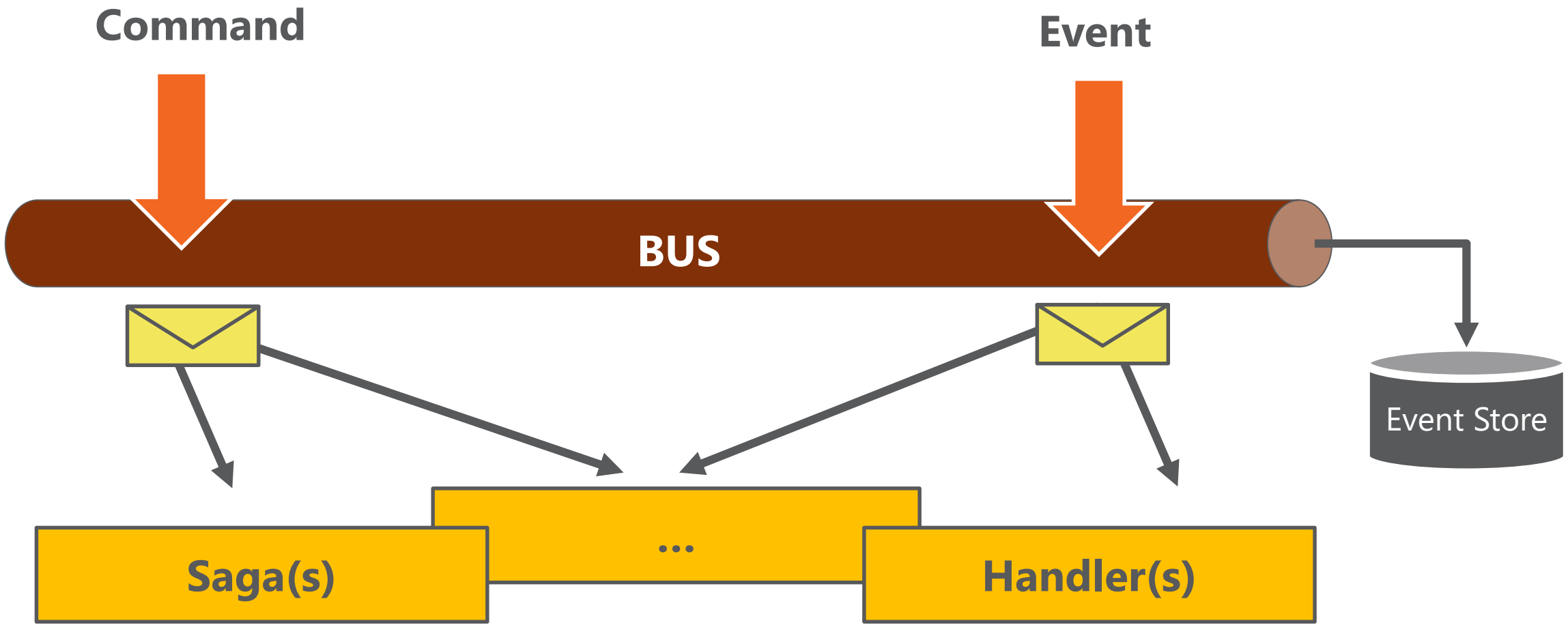


Command Stack

Use just the pattern that fits better



Storage



Read Stack

Use just the code that does the job

O/RM of choice

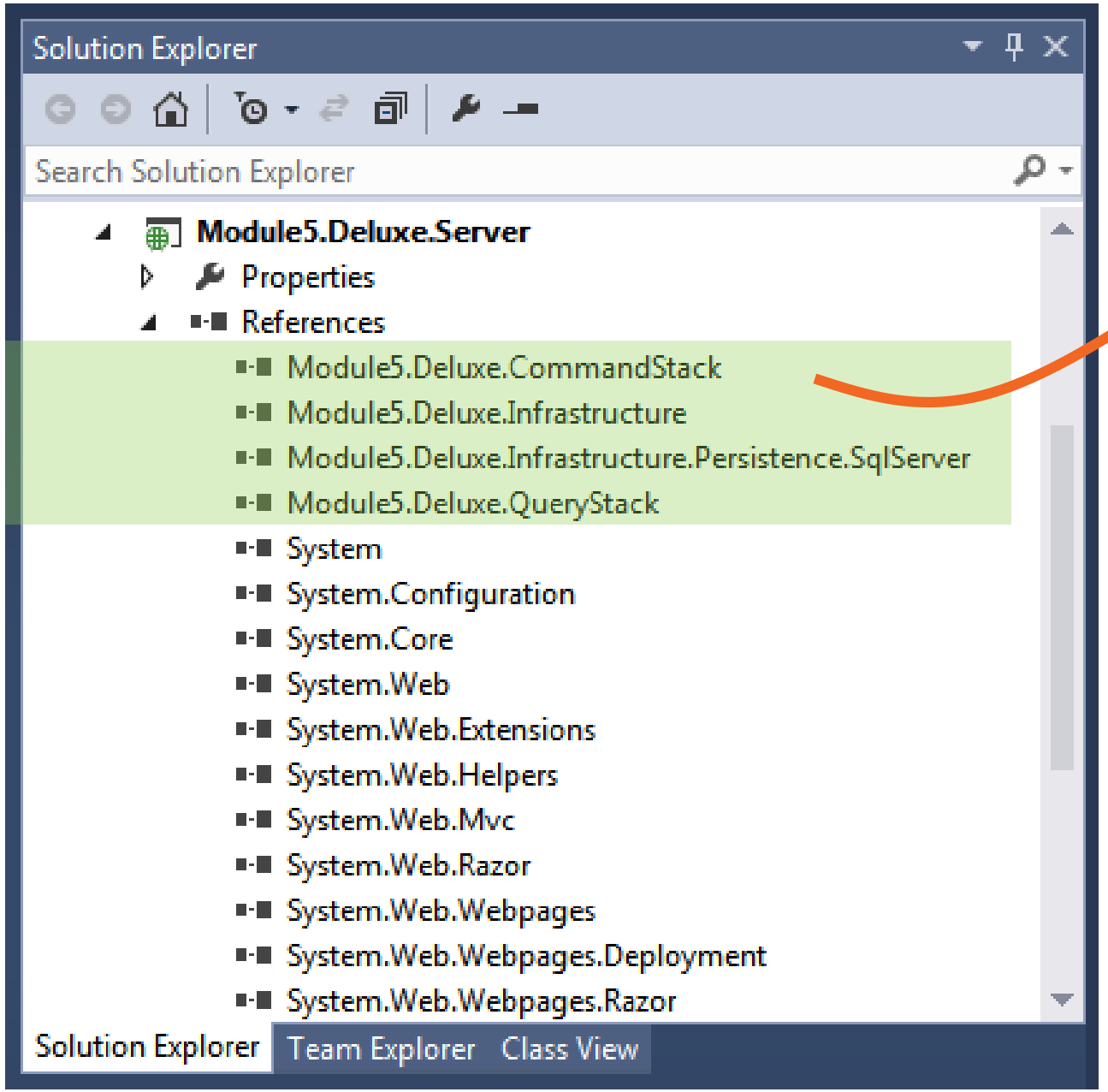
LINQ

Ad-hoc storage

Relational

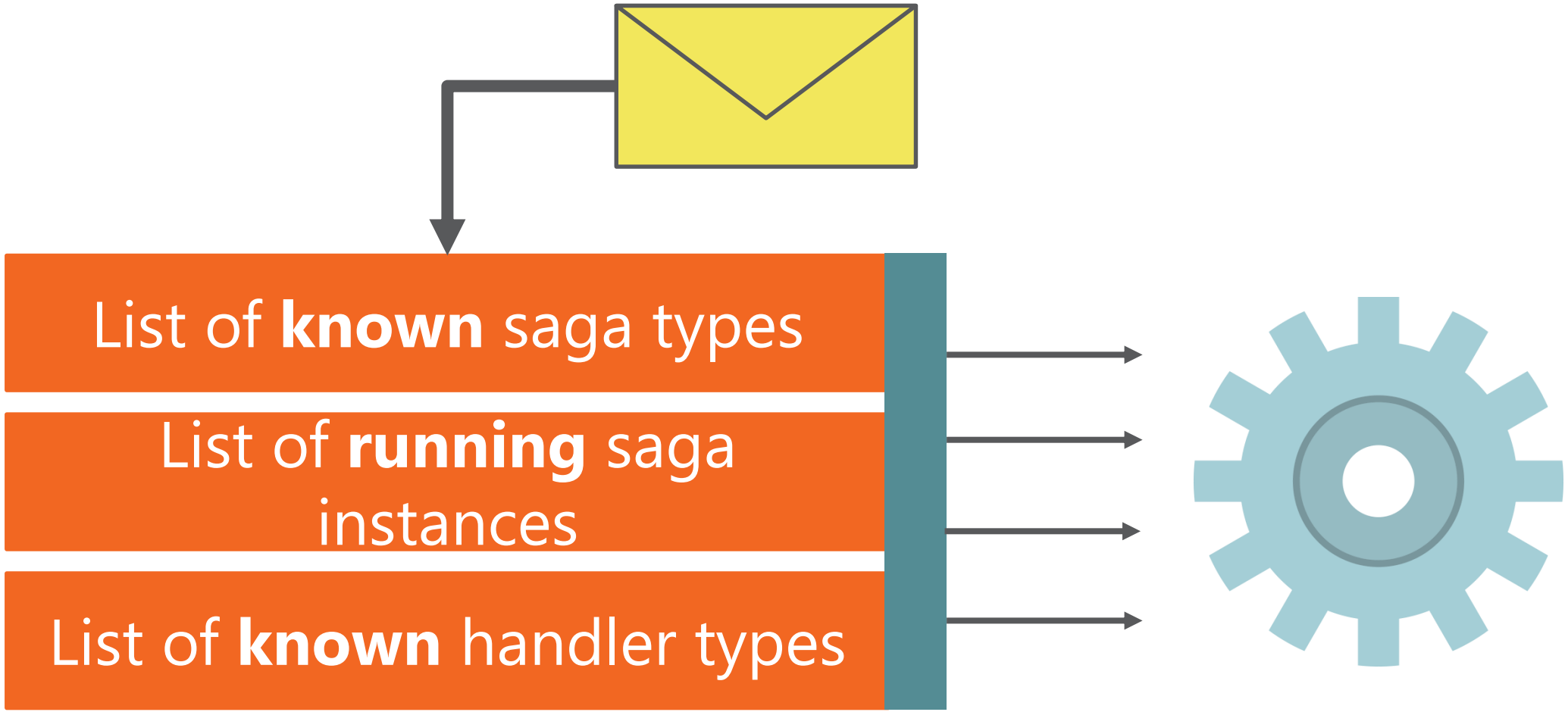
How would you build the read snapshot database?

Use a handler!



- **Command stack**
- **Read stack**
- **Infrastructure layer**
  (with persistence)

INSIDE THE BUS



INSIDE THE SAGA

Command or event that starts the associated business process

List of commands the saga can handle

List of events the saga is interested in

```
public class CheckoutSaga : Saga<CheckoutSagaData>,
        IStartWith<StartCheckoutCommand>,
        ICanHandle<CancelCheckoutCommand>,
        ICanHandle<PaymentCompletedEvent>,
        ICanHandle<PaymentDeniedEvent>,
        ICanHandle<DeliveryRequestRefusedEvent>,
        ICanHandle<DeliveryRequestApprovedEvent>
{
    public void Handle(StartCheckoutCommand message)
    {
        :
    }

    ...
}
```

More About Sagas

Sagas must be identified by a unique ID

- Can be a GUID
- Can be the ID of the aggregate the saga is all about
- Can be a combination of values that is unique in the context

Sagas might be persistent and stateful

- Persistence is care of the bus
- State of the associated aggregate must be persisted

Sagas might be stateless

- Mere task executor getting any data from the start command

Extending a Solution

Got a new handling scenario for an existing event?

Got a new handling scenario for a new feature?

Write a new saga or handler and **register** it with the bus.
That's it.

More About the Bus

- **Can write your own bus**
  - Mostly about real traffic hitting the application
  - Plug in some queue engine
  - Plug in some persistence mechanism

- **Look into existing products and frameworks**
  - **NServiceBus** from Particular Software: particular.net/nservicebus
  - **Rebus** from Rebus-org: github.com/rebus-org/Rebus
  - **MassTransit** from Pandora: masstransit-project.com

DEMO