Module 04:

"CQRS and Mediator"

Agenda

- Introduction
- **CQRS**
- Mediator
- Mapping
- Summary

CQS = Command Query Separation

- Coding principle concerning methods in imperative programming
  - A *query*
    - has return value
    - should never mutate state
  - A *command*
    - returns void
    - Is allowed to mutate state

```
class BookHandler
{
    public Book CreateBook(string title) { ... }
    public Book GetBookByTitle(string title) { ... }
    public IEnumerable<Book> GetAll() { ... }
}
```

❌

```
class BookHandler
{
    public void CreateBook(string title) { ... }
    public Book GetBookByTitle(string title) { ... }
    public IEnumerable<Book> GetAll() { ... }
}
```

✓

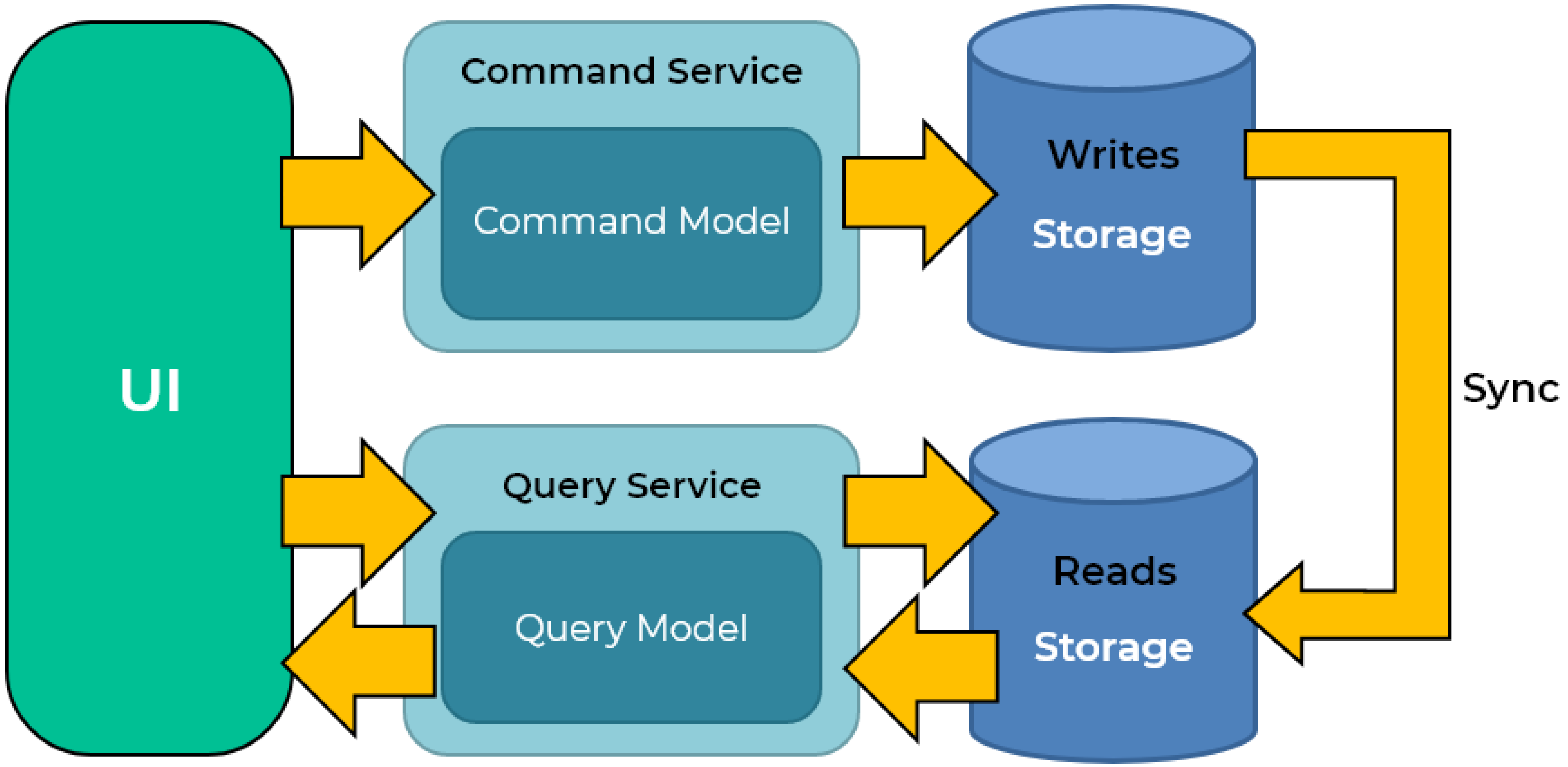
CQRS = Command Query Responsibility Segregation

- Essentially the broader architectural pattern version of CQS
  - Separate Mutation/Write and Querying/Read
  - "Service per use-case" scenario will emerge

```
class BookCommandHandler
{
    public void CreateBook(string title) { ... }
}
```

```
class BookQueryHandler
{
    public Book GetBookByTitle(string title) { ... }
    public IEnumerable<Book> GetAll() { ... }
}
```

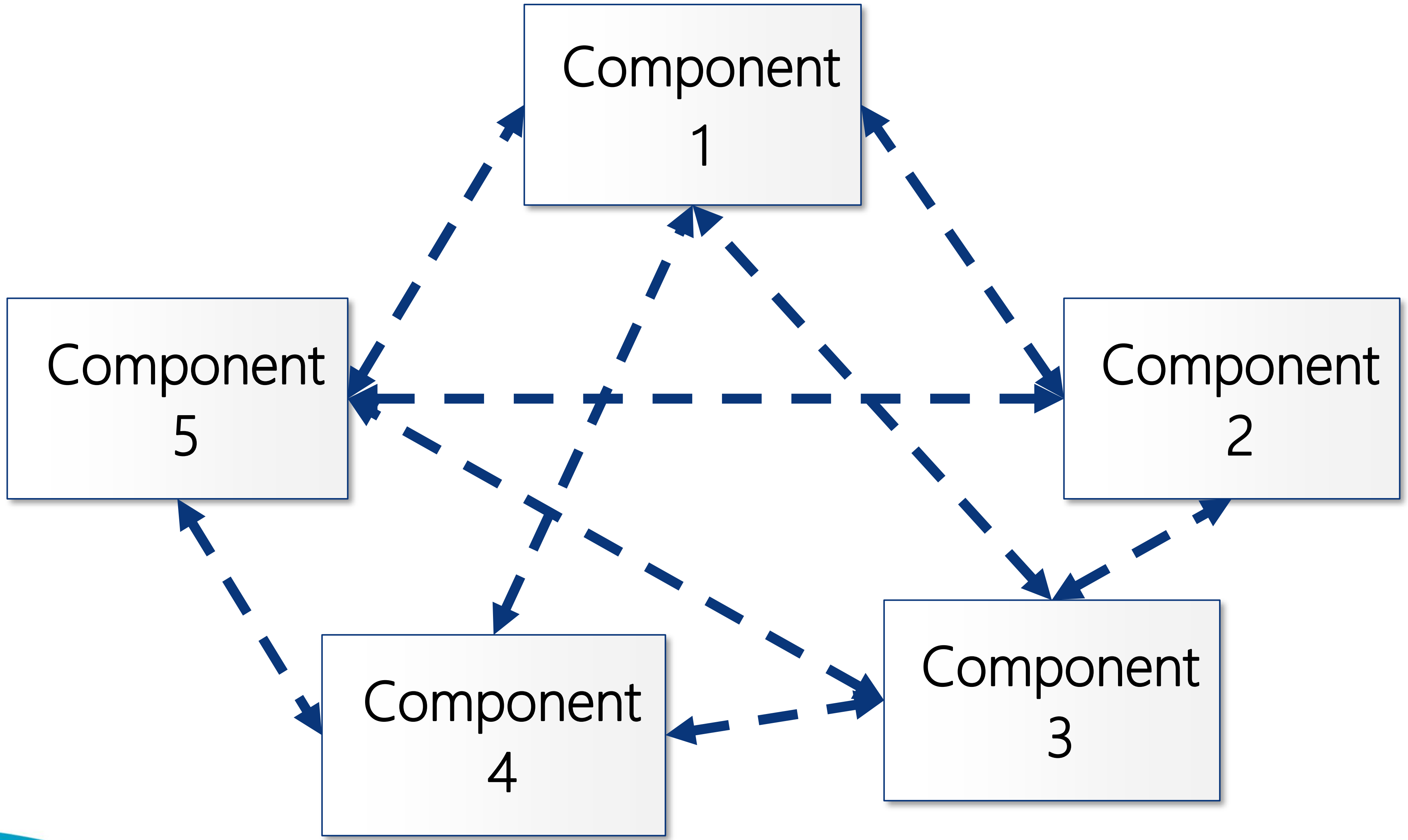
CQRS in General



Agenda

- Introduction
- CQRS
- **Mediator**
- Mapping
- Summary

Without the Mediator Pattern



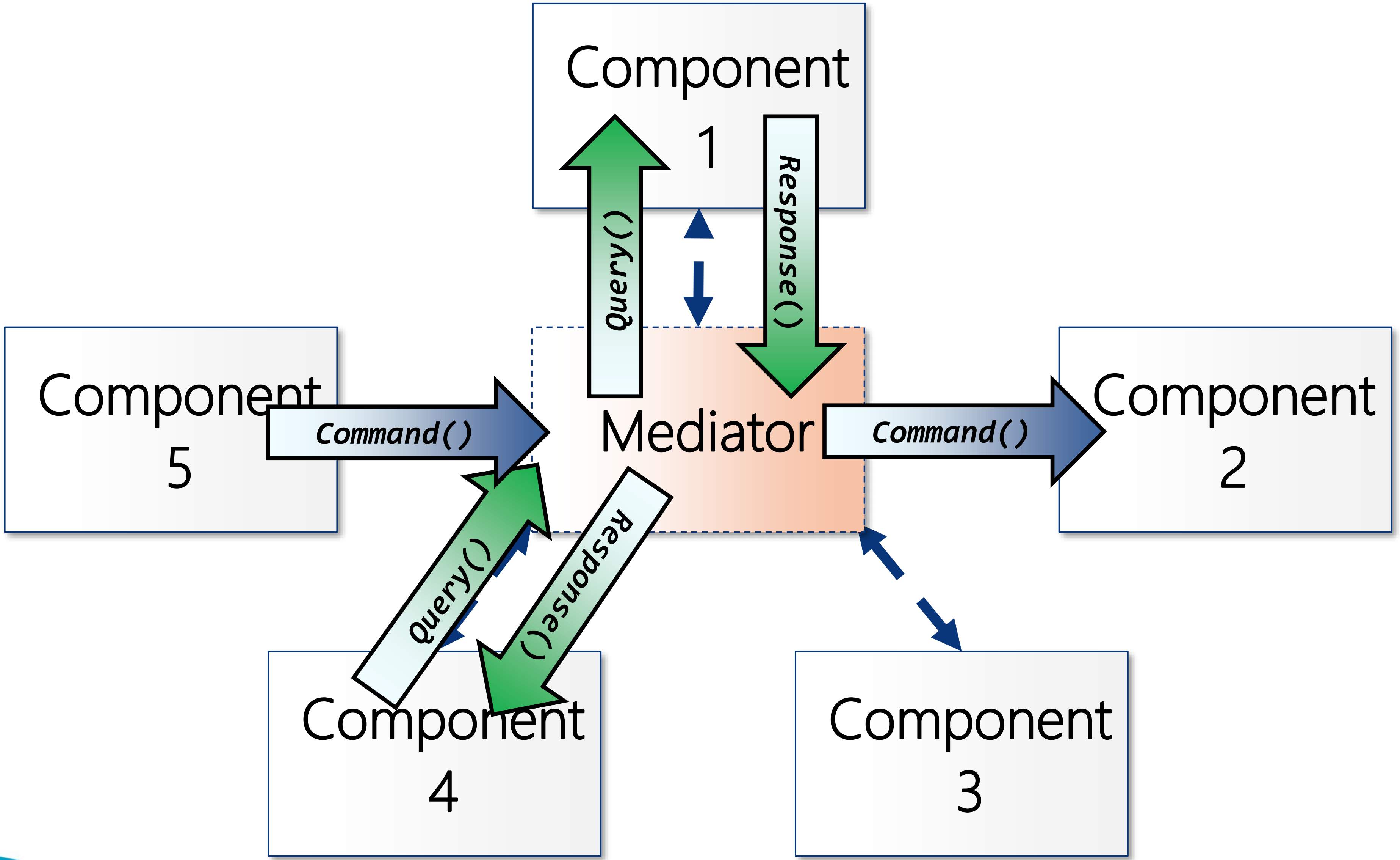
Pattern: Mediator

- *Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interactions independently.*

- Outline
  - Define a separate object ("mediator") that encapsulates the interactions between objects
  - All objects interact with the mediator instead of interacting with each other directly
  - Objects have no explicit knowledge of other objects than the mediator

- Origin: Gang of Four

With the Mediator Pattern

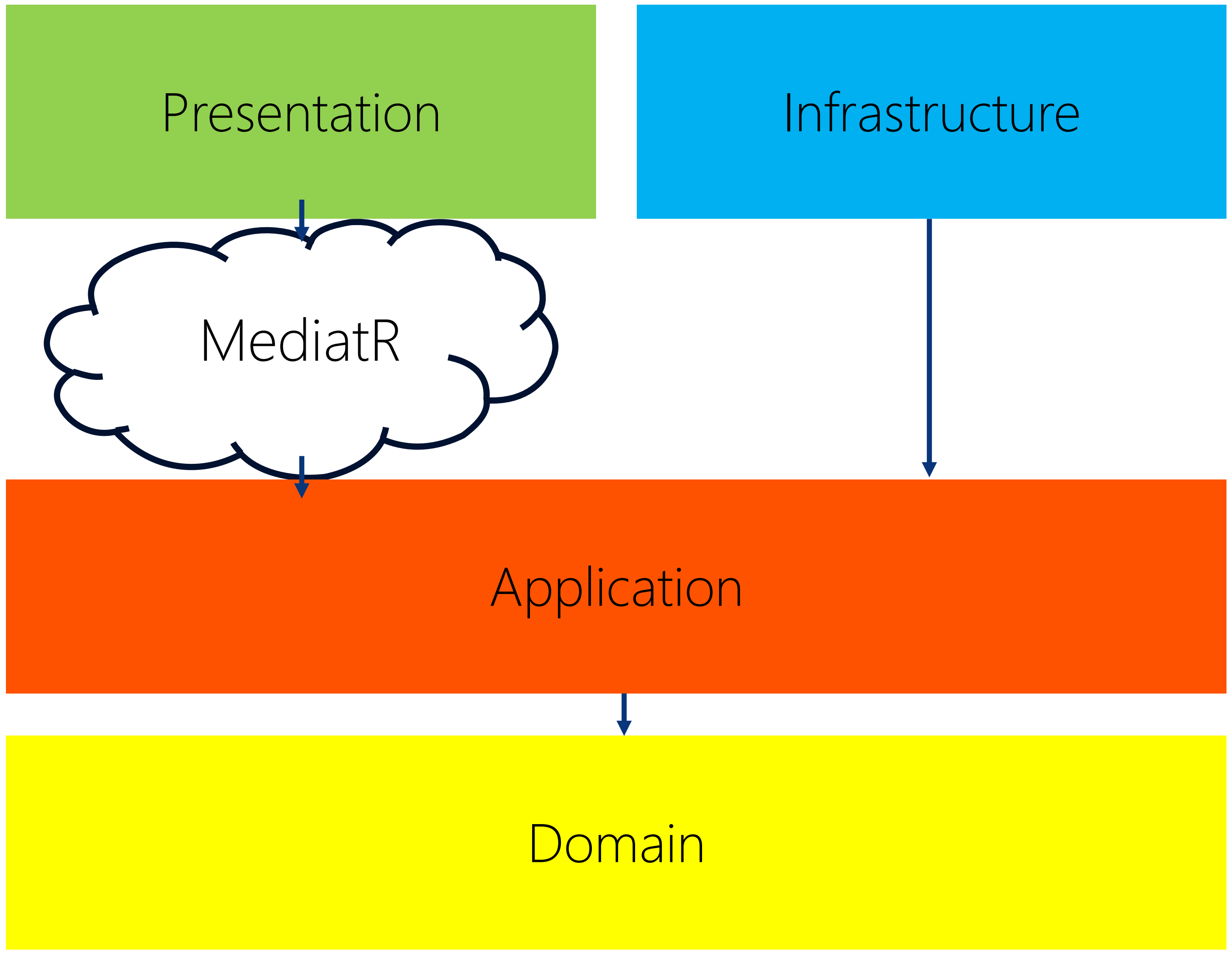


MediatR

- Freely available nuget package
  - https://github.com/jbogard/MediatR/wiki

- Handles two kinds of messages
  - Request/Response messages
    - Dispatched to a single handler
  - Notification messages
    - Dispatched to multiple handlers

The Big Picture



MediatR Basics

- Define components for Request, Response, and RequestHandler

```
public record class Ping : IRequest<string>
{
}
```

```
public class PingHandler : IRequestHandler<Ping, string>
{
    return Task.FromResult("Pong");
}
```

- Activate through **`IMediator`** instance

```
string response = await mediator.Send(new Ping());
Debug.WriteLine(response); // "Pong"
```

MediatR Registrations

- Works seamlessly with `IServiceCollection`

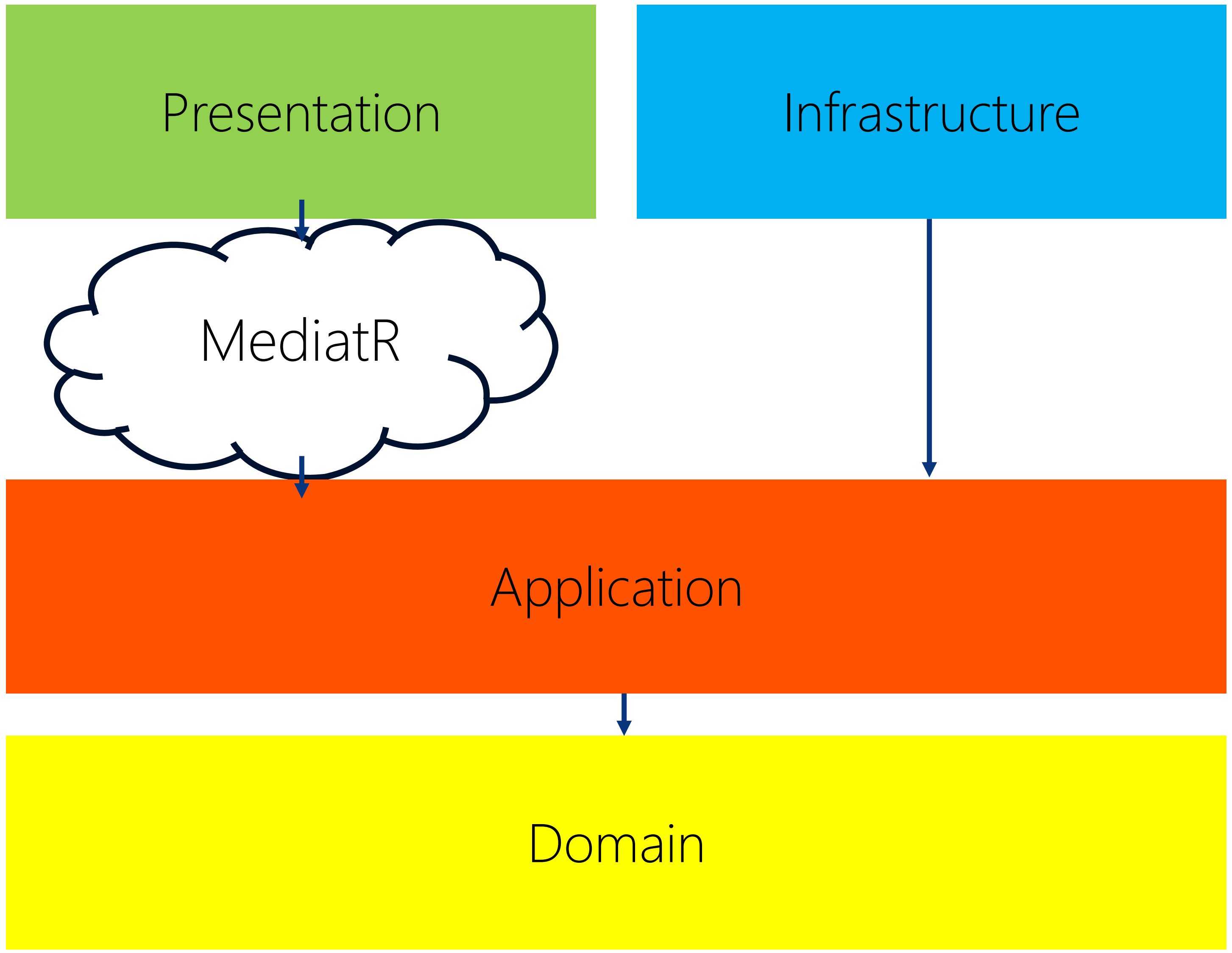
```
services.AddMediatR(cfg =>
{
    cfg.RegisterServicesFromAssembly(typeof(Program).Assembly);
});
```

- Registers
  - `IMediator` (and `ISender`) as transients
  - `IRequestHandler<,>` as transients

Agenda

- Introduction
- CQRS
- Mediator
- **Mapping**
- Summary

Beware of Object Leaking!



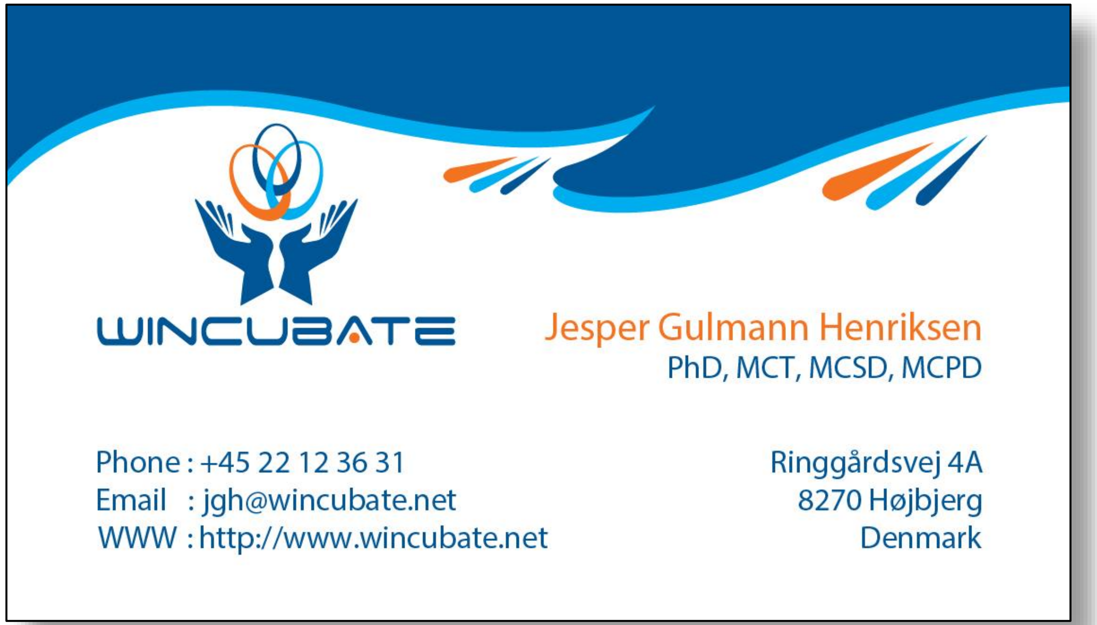
The Mapping Controversy

- There are many mapping tools
  - Automapper
  - Mapster
  - …
- But don’t use these!

- Instead write explicit mapping code
  - Constructors
  - Extension Methods
  - Operators

Summary

- Introduction
- CQRS
- Mediator
- Mapping



WINCUBATE

Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net

Ringgårdsvej 4A
8270 Højbjerg
Denmark