# Module 23: "Observer"

## Agenda

- Introductory Example: Stock Market
- Challenges
- Pattern: Observer
- Original Observer Pattern
- Observer Pattern in .NET: Events
- Implementing the Observer Pattern
- Overview of Observer Pattern
- Discussion

## Introductory Example: Stock Market

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

    private void OnStockTraded( string ticker, decimal latest )
    {
        Console.WriteLine( $"{ticker} traded at USD {latest:f2}");
    }
}
```

```
class StockObserver
{
    // ???
}
```

```
class OtherStockObserver
{
    // ???
}
```

## Challenges

- How do stock observers get the new stock prices as soon as they happen at the stock market?
  - ...without repeatedly polling?
  - ...without too tight coupling?

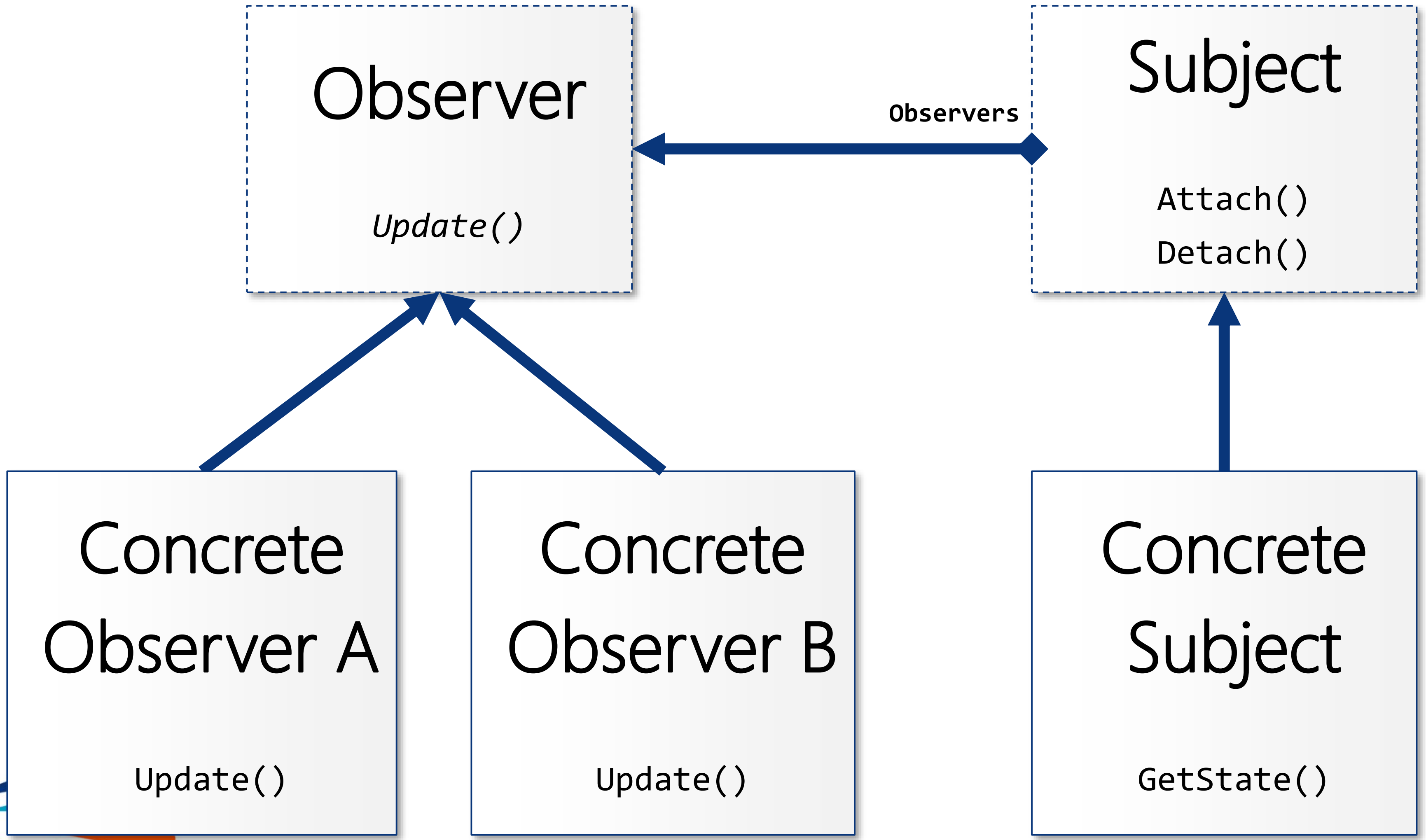
## Pattern: Observer

- *Define a one-to-many dependency relation between objects so that when one object changes state, all its dependents are notified and updated automatically.*

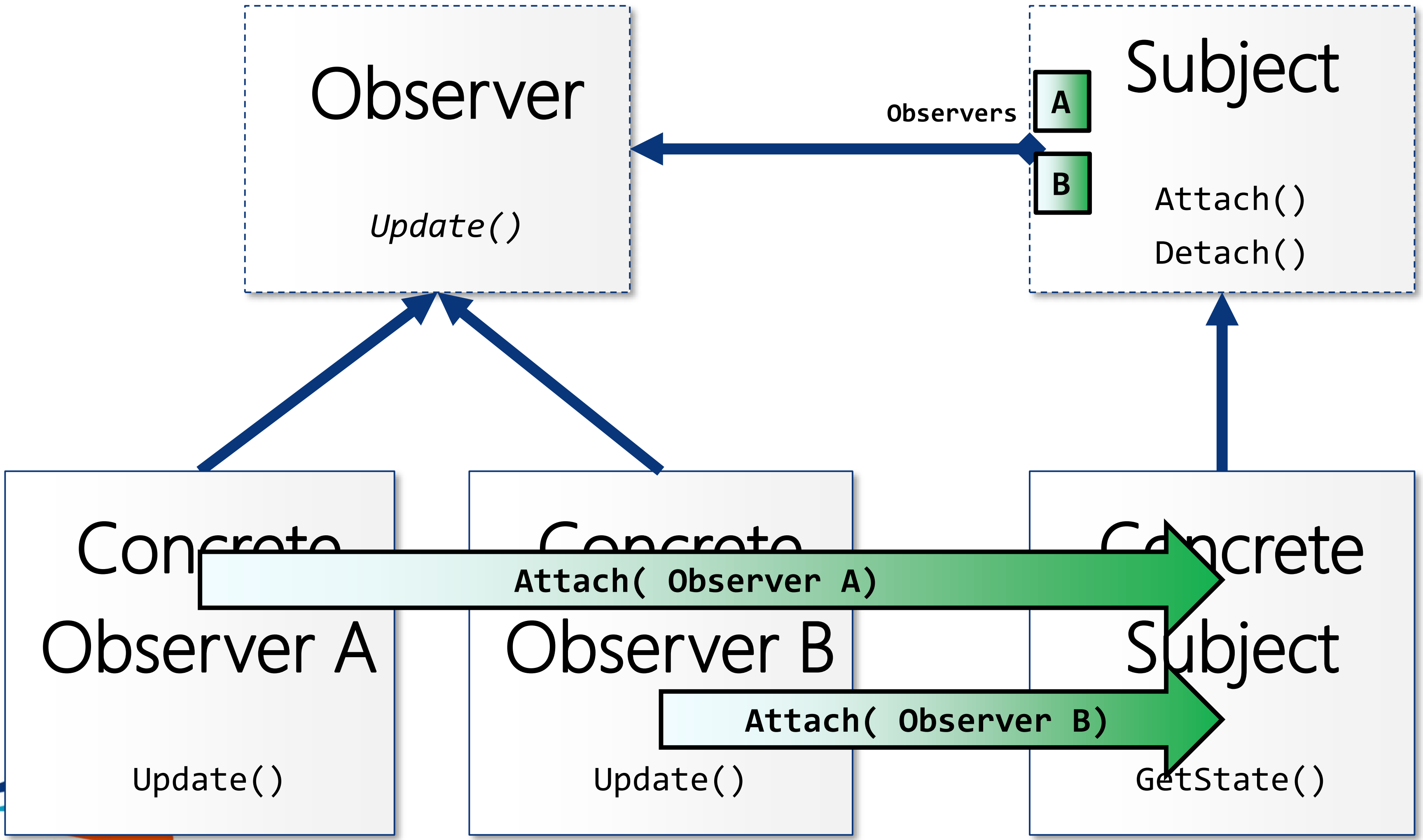
- Outline
  - Define Subject and Observer objects.
  - Let observers register and deregister with Subject
  - Ensure that when a Subject changes state, it will notify all registered Observers.

- Origin: Gang of Four

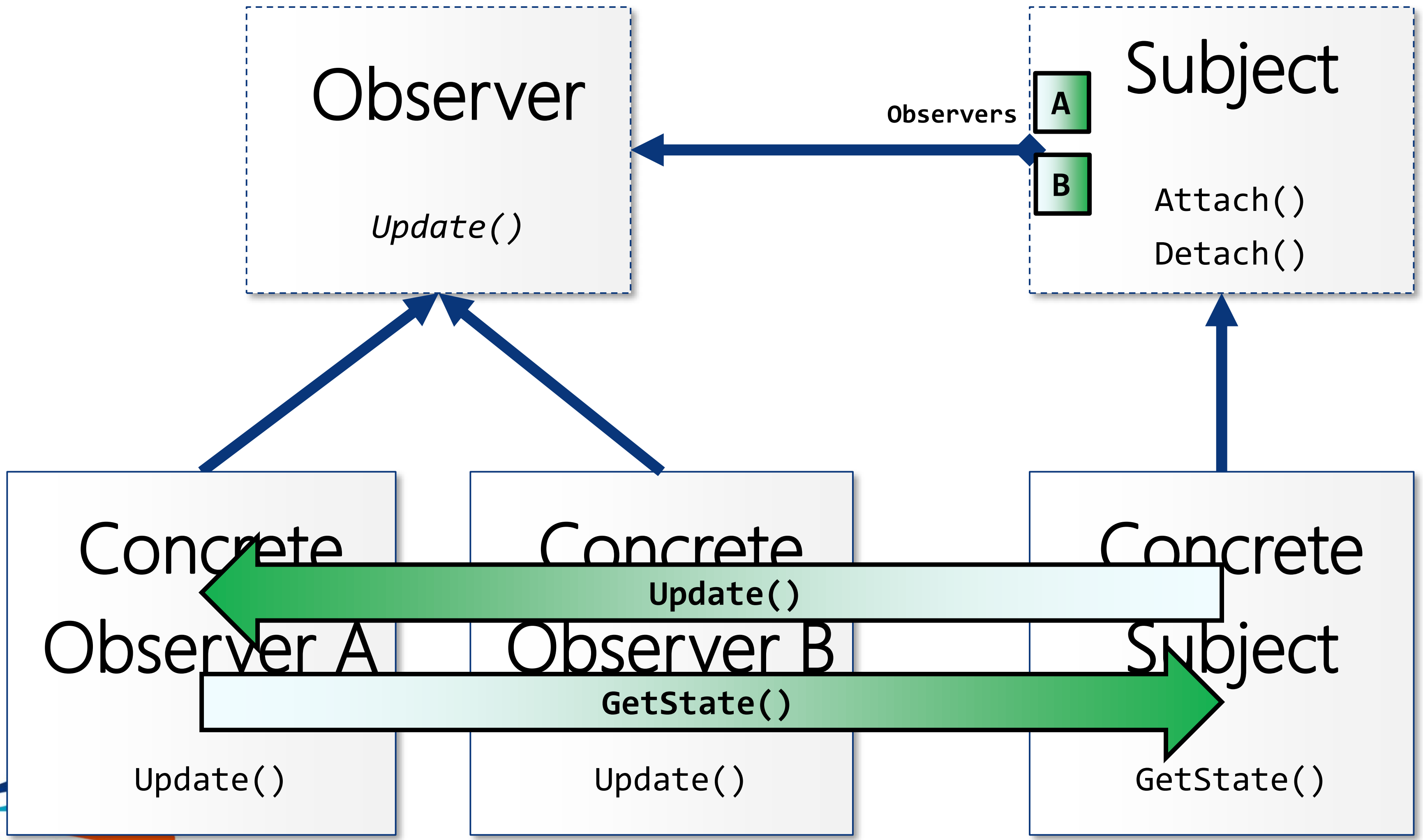
# Original Observer Pattern



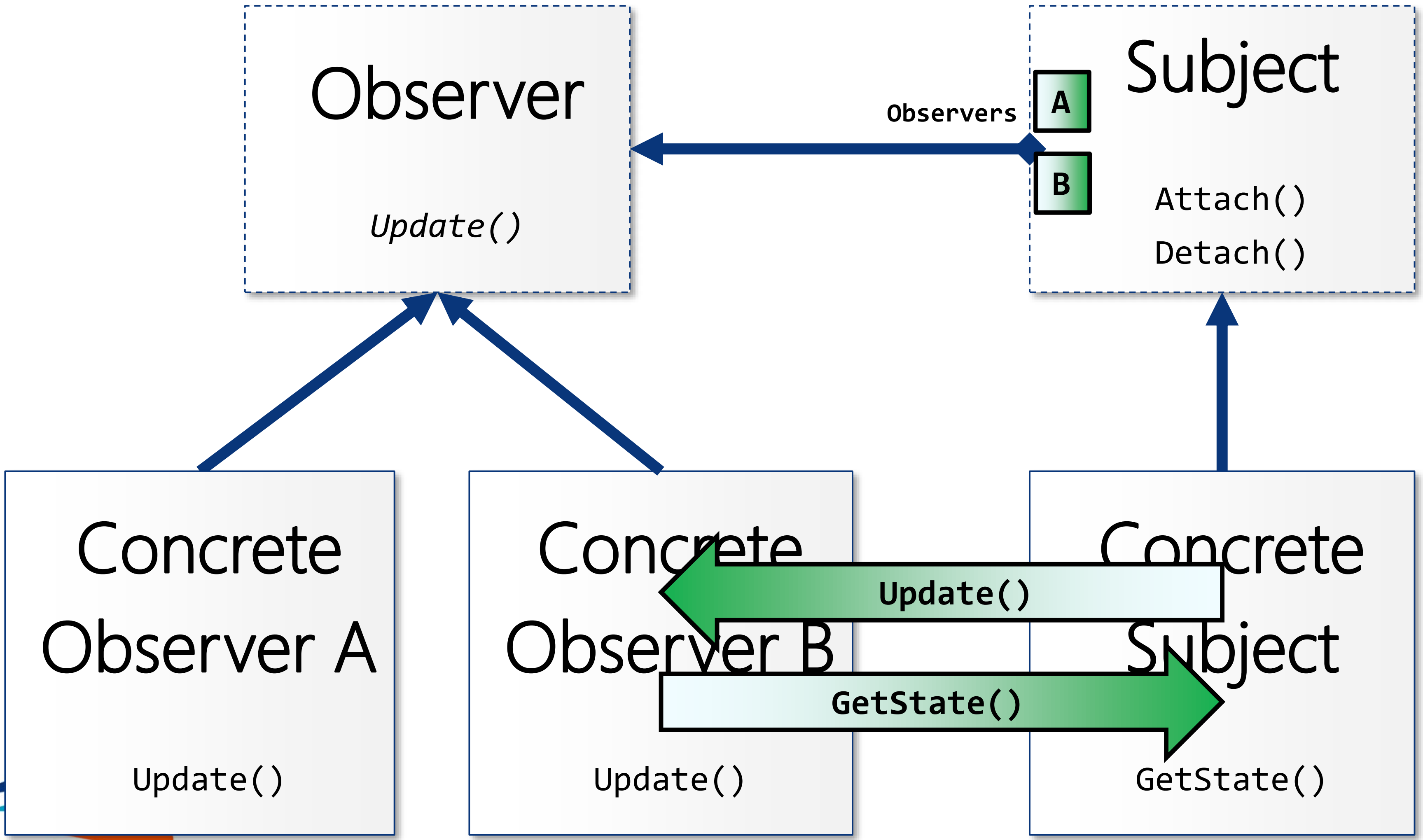
# Original Observer (Registration)



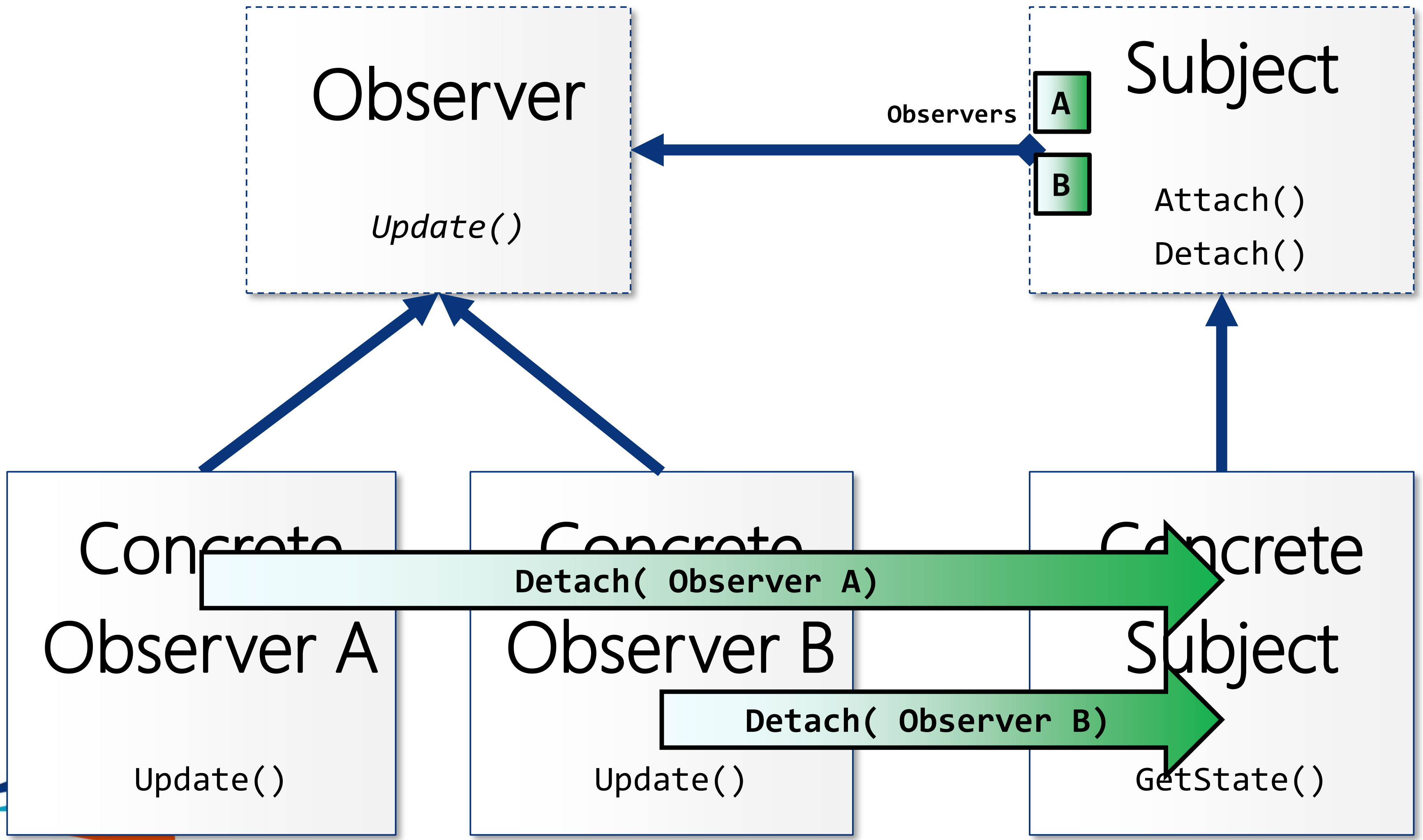
# Original Observer (Update)



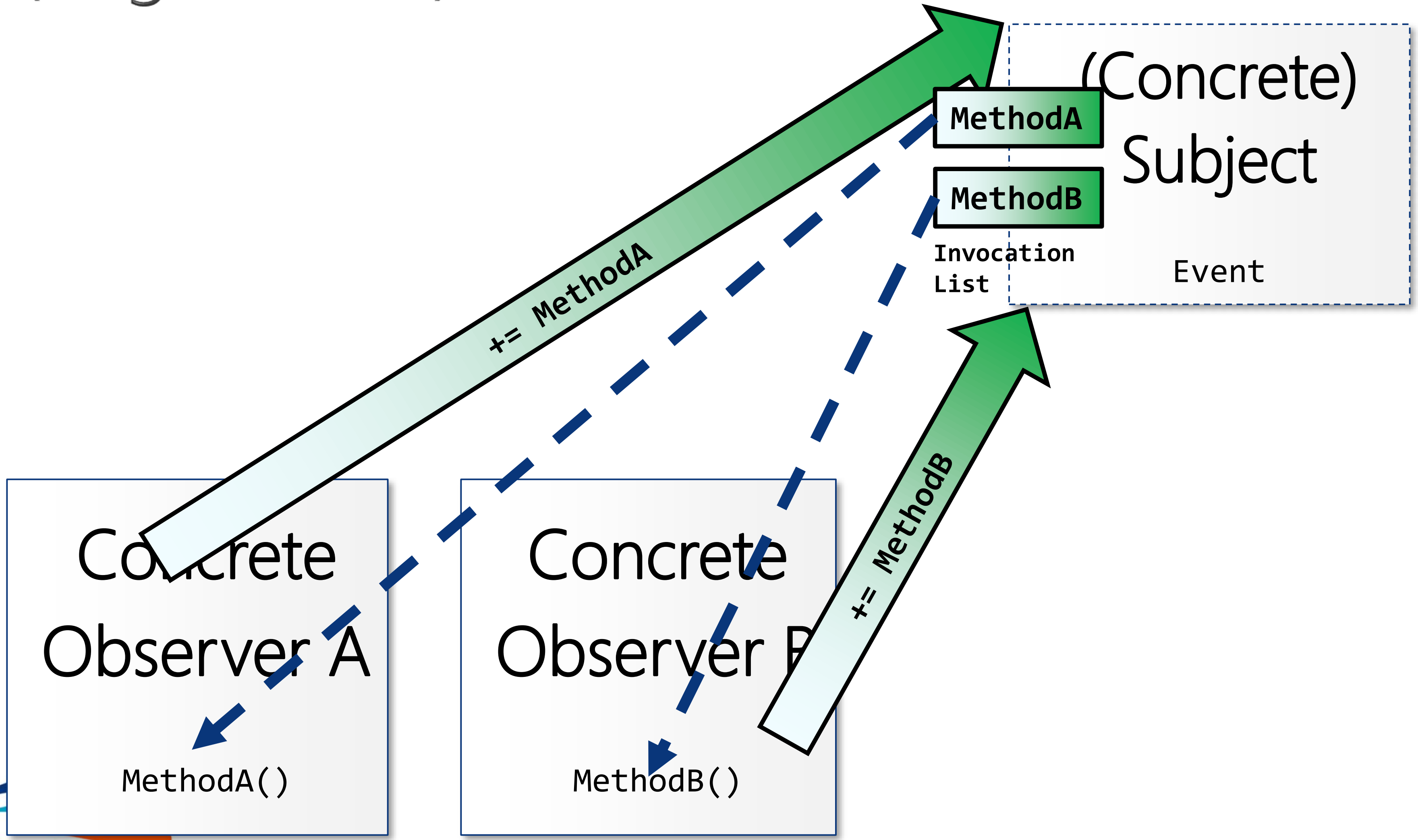
# Original Observer Pattern (Update)



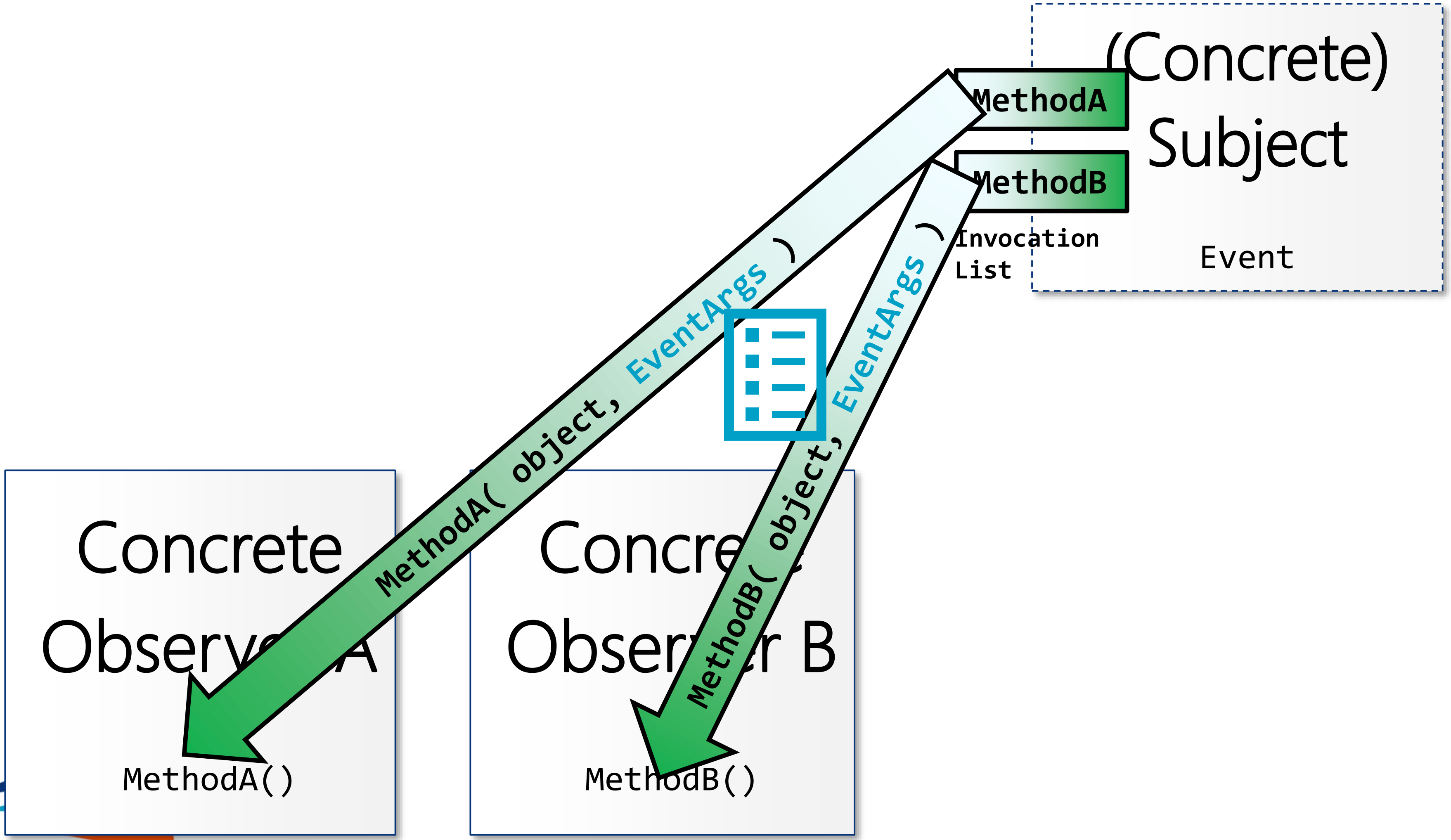
# Original Observer (Deregistration)



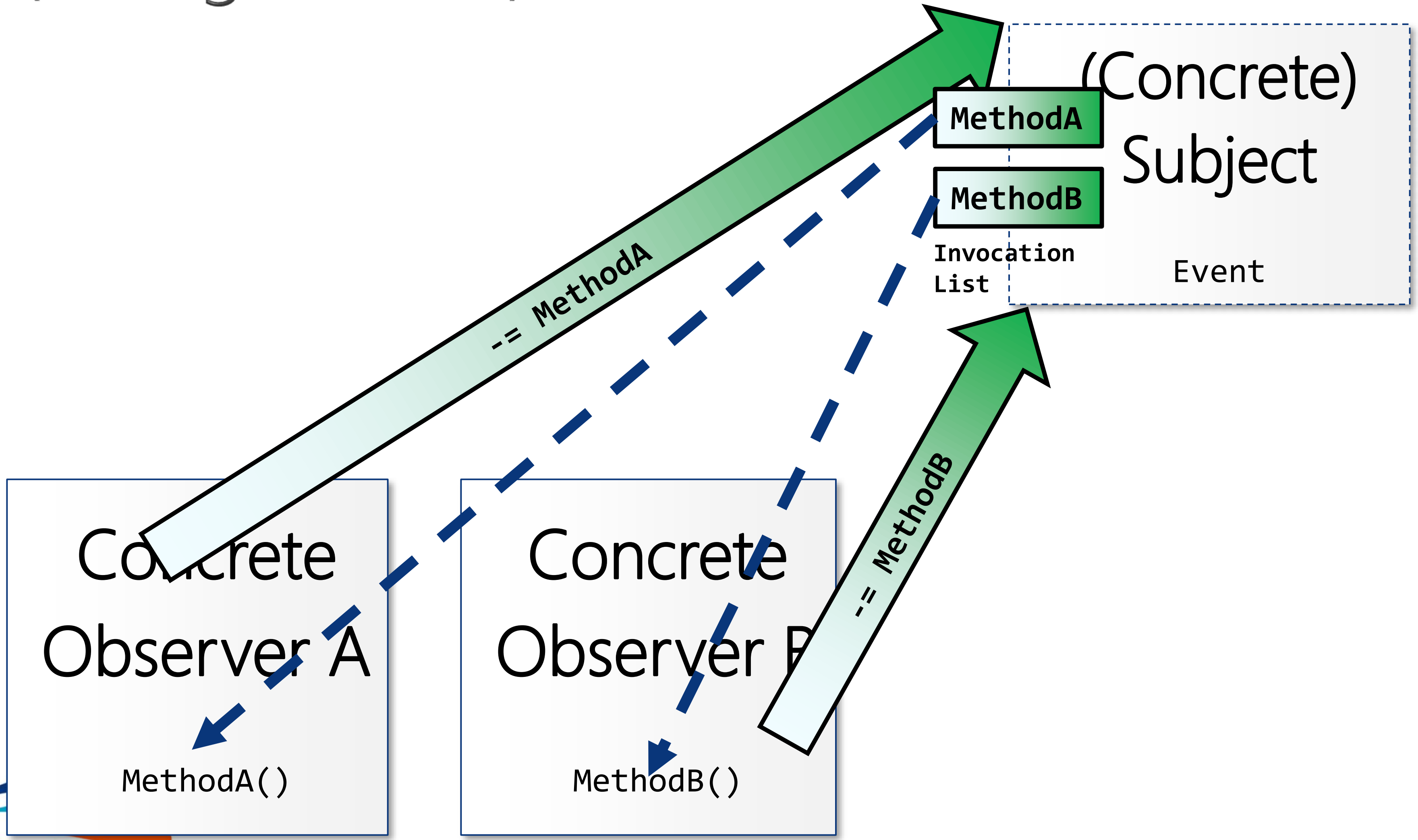
# .NET Observer Pattern: Events (Registration)



# .NET Observer Pattern: Events (Update)



# .NET Observer Pattern: Events (Deregistration)



## Overview of Observer Pattern

- (Concrete) Subject
  - Class with event **`Event`** defined
    - Specifies **`EventArgs`** class with state
  - Raises event when there is new state (**`EventArgs`** object) to notify observers

- Concrete Observer
  - Concrete class with **`Method()`** of same signature as **`Event`**
  - Registers with **`+=`**
  - Receives state from Subject through **`EventArgs`** object
  - Deregisters with **`-=`**

# Pros and Cons of Observer

- Pros
  - Very easy to use
  - Supported by native syntax in C#
  - Used extensively throughout all of .NET
  - Much simpler, nicer, and cleaner than original Observer Pattern
  - Works elegantly with many-to-many relationships

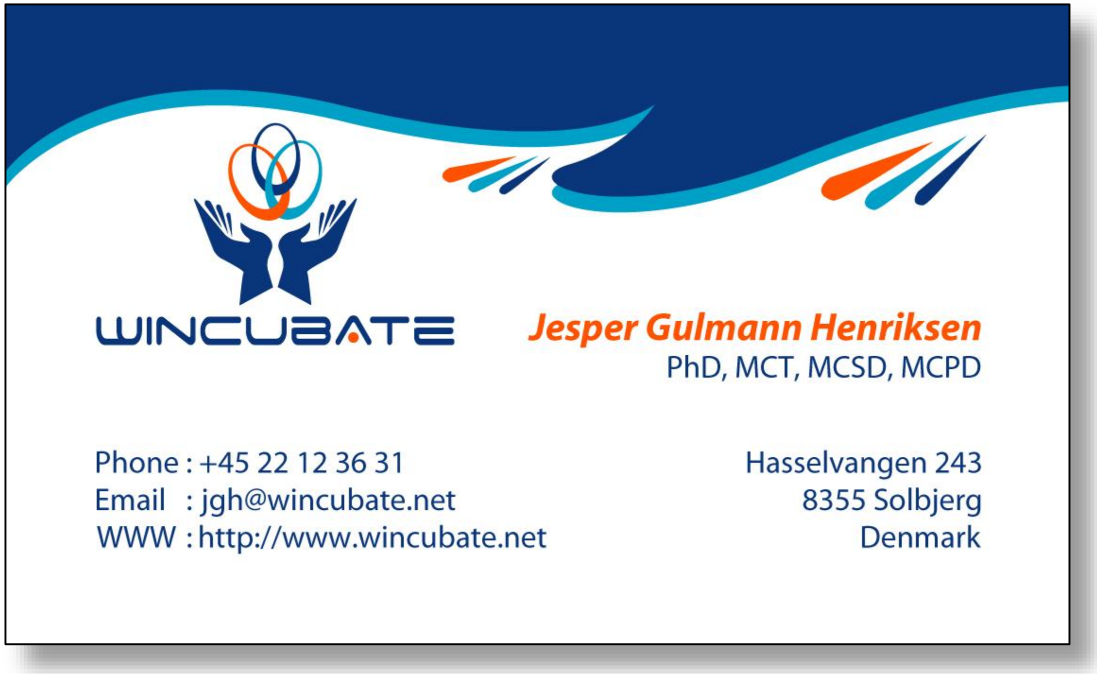
- Cons
  - Danger of resource leaks
    - Consider deregistering observer! Maybe IDisposable? But...
    - Cannot deregister lambda expressions and anonymous methods
  - Be careful about multi-threading and serialization
  - No obvious way of propagating Subject errors to Observer

# IObservable<T> and IObserver<T>

- More modern Observer Pattern interfaces were added in .NET 4.0 (See Lab 23.1)

```
public interface IObservable<out T>
{
    IDisposable Subscribe( IObserver<T> observer );
}
```

```
public interface IObserver<in T>
{
    void OnCompleted();
    void OnError( Exception error );
    void OnNext( T value );
}
```



WINCUBATE

*Jesper Gulmann Henriksen*
PhD, MCT, MCSD, MCPD

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

Hasselvangen 243
8355 Solbjerg
Denmark