

# Module 15: "Chain of Responsibility"



# Agenda

- ▶ Introductory Example: Screening Messages
- ▶ Challenges
- ▶ Implementing the Chain of Responsibility Pattern
- ▶ Pattern: Chain of Responsibility
- ▶ Overview of Chain of Responsibility Pattern



# Introductory Example: Screening Messages

```
class SalesScreener : IMessageScreener
{
    public ScreeningResponse Screen( IMessage message ) =>
        message.Contents.ContainsAnyOf("order", "purchase", "buy") ?
            ScreeningResponse.Accepted : ScreeningResponse.Unprocessed;
}
```

```
foreach (IMessageScreener screener in screeners)
{
    ScreeningResponse response = screener.Screen(message);
    if( response != ScreeningResponse.Unprocessed)
    {
        Console.WriteLine( $"Message was {response}");
        break;
    }
}
```

# Challenges

- ▶ The request sender has to provide the bookkeeping logic of asking each screener – and process return values
- ▶ Each request sender is coupled to all receivers
- ▶ This logic is present on the wrong level
  - not the responsibility of the request sender
- ▶ This logic will have to be repeated for any other receiver

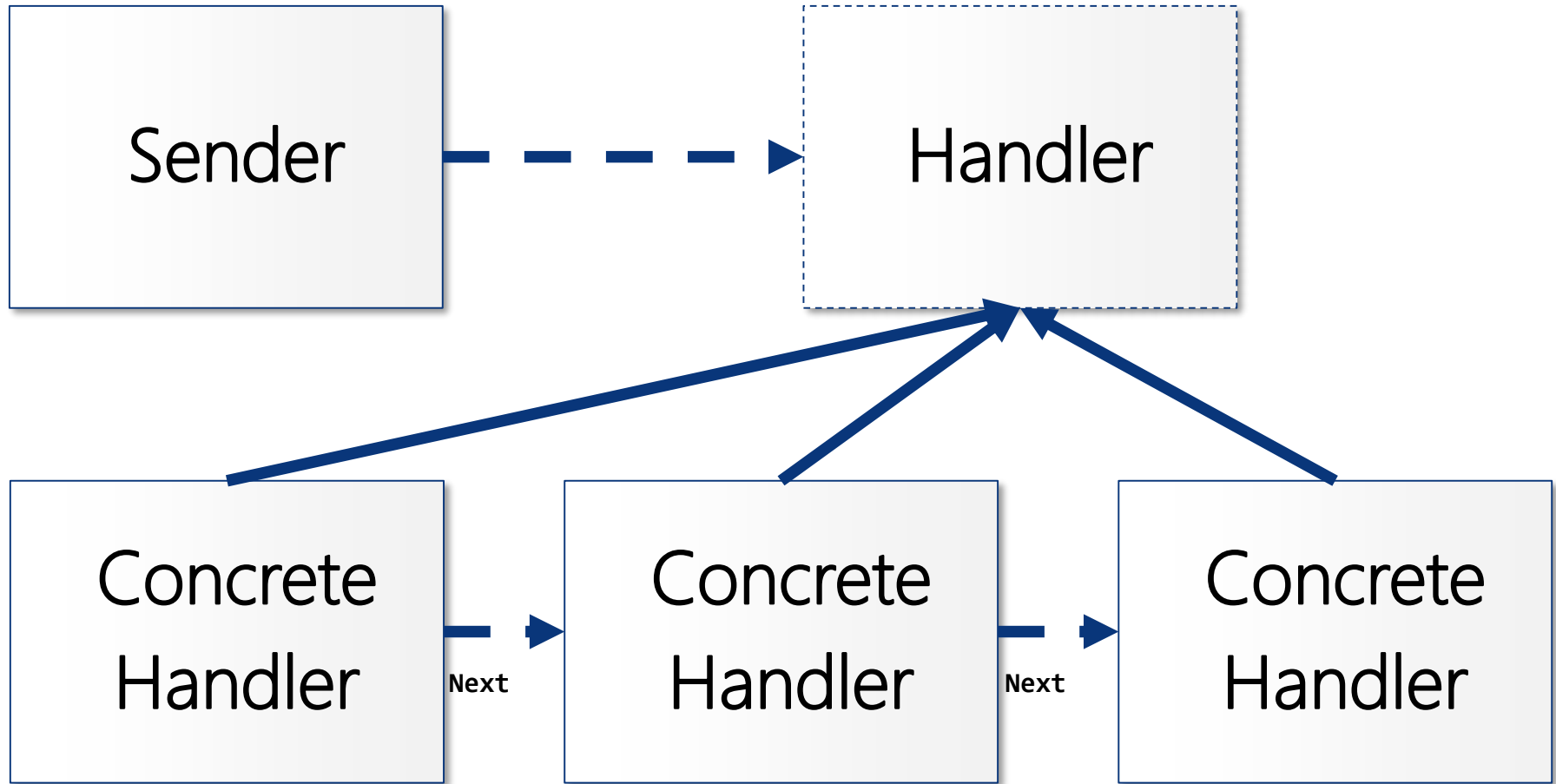


# Pattern: Chain of Responsibility

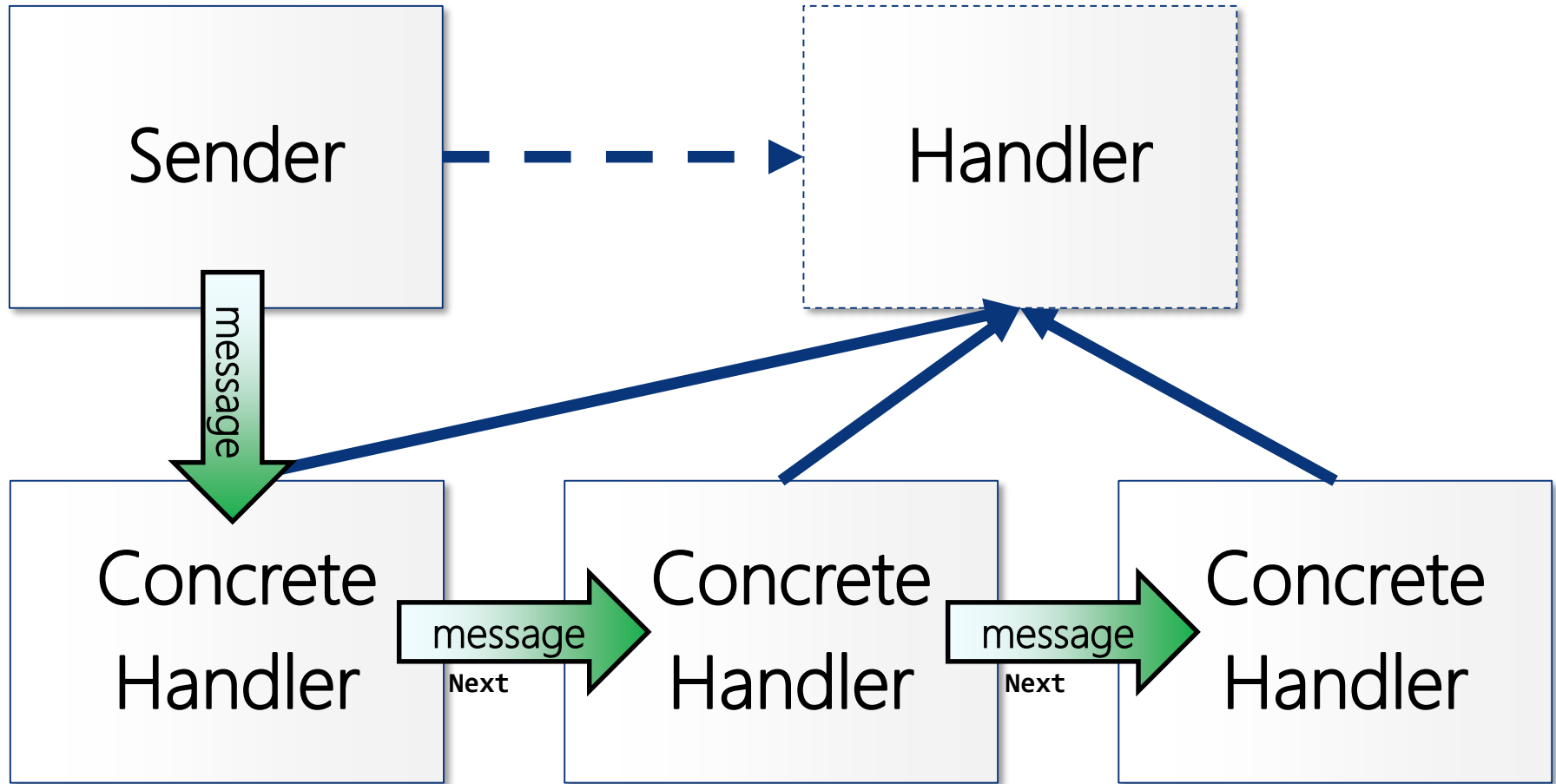
- ▶ *Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.*
- ▶ Purpose
  - Senders and receivers are decoupled
  - It is possible for more than one receiver to handle a request
- ▶ Origin: Gang of Four



# Overview of Chain of Responsibility Pattern



# Overview of Chain of Responsibility Pattern



# Overview of Chain of Responsibility Pattern

- ▶ Sender
  - Interacts with only a single Handler
  - Has no knowledge of which concrete handler is called
- ▶ Handler
  - Interface or base class to handling functionality
- ▶ Concrete Handler
  - Handler instance class performing concrete handling functionality
  - Decoupled from Sender and other Concrete Handlers





