Design Patterns: Mediator

About me

- Mediator Mediatorovich
- Mediator Engineer at Mediator.corp

Intro

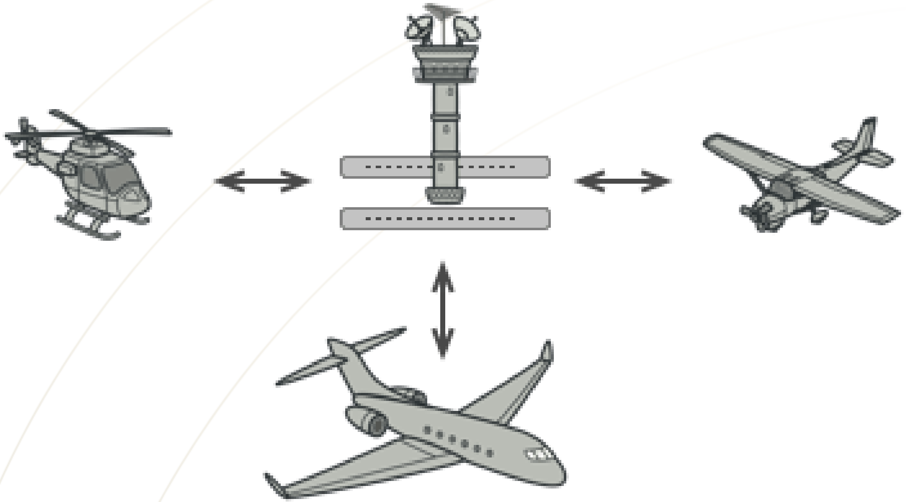
Mediator

Belongs to the Behavior Patterns group

Reduces dependencies and coupling between objects/components

Takes a role of coordinator

Real-life Analogy



Aircraft pilots don’t talk to each other directly when deciding who gets to land their plane next.
All communication goes through the control tower.

Credits for analogy and picture: https://refactoring.guru/design-patterns/mediator

My findings

Good separation of Cross-cutting concerns (logging, error handling, etc)

Mediator object becomes a Configuration object which makes it easy to test

Easy integration with other patterns and transformation to some patterns

When to use

Hard to change some of the classes because they are tightly coupled to a bunch of other classes

Can’t reuse a component in a different program because it’s too dependent on other components

Reuse group of components in a different configuration

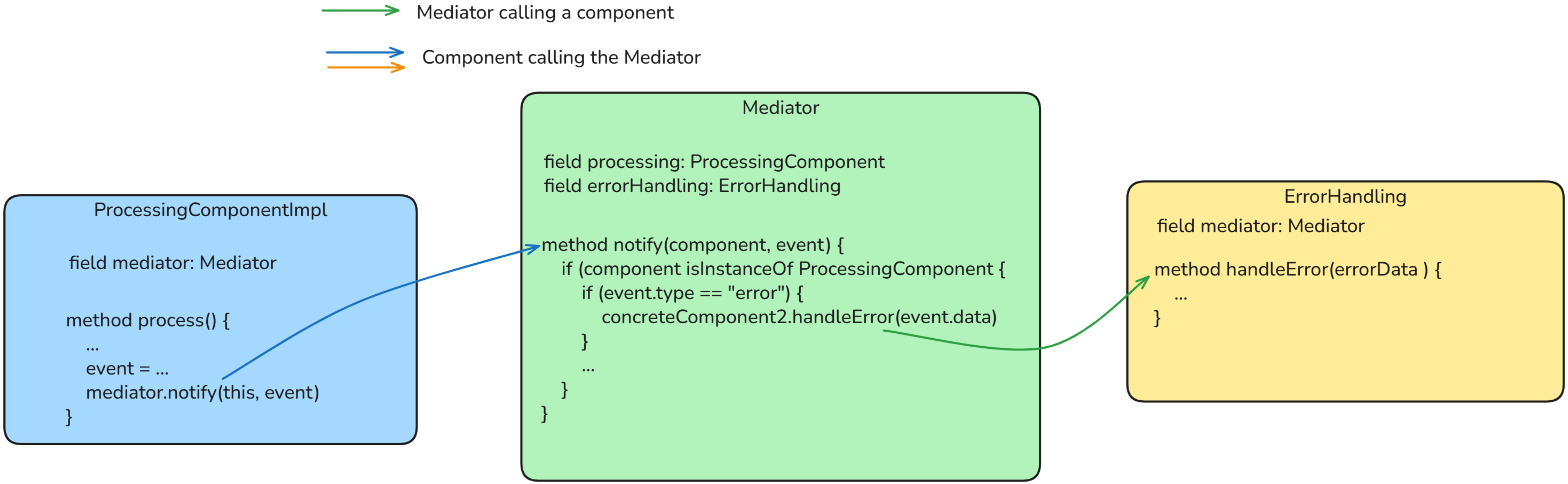
Classic Implementation

Utilizes Visitor pattern-like mechanism which is unnatural and breaks some programming principles

Usability depends on the underlying system

Suits best for components without entry point and event/action-based communication (e.g. GUI)

Classic Implementation



→ Mediator calling a component

→ Component calling the Mediator

ProcessingComponentImpl

```
field mediator: Mediator

method process() {
    ...
    event = ...
    mediator.notify(this, event)
}
```

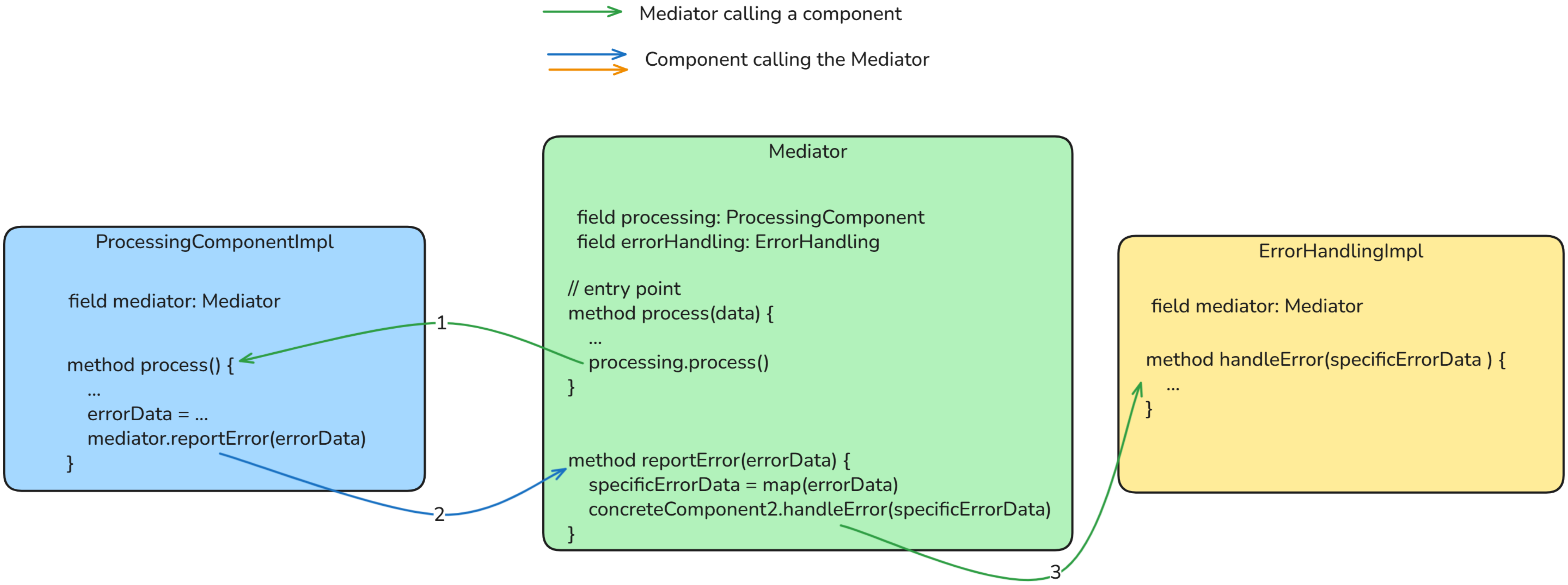
Mediator

```
field processing: ProcessingComponent
field errorHandling: ErrorHandling

method notify(component, event) {
    if (component isInstanceOf ProcessingComponent {
        if (event.type == "error") {
            concreteComponent2.handleError(event.data)
        }
        ...
    }
}
```

ErrorHandling

```
field mediator: Mediator

method handleError(errorData ) {
    ...
}
```

Classic Implementation



Mediator calling a component

Component calling the Mediator

ProcessingComponentImpl

```
field mediator: Mediator


method process() {
   ...
   errorData = ...
   mediator.reportError(errorData)
}
```

Mediator

```
 field processing: ProcessingComponent
 field errorHandling: ErrorHandling

// entry point
method process(data) {
   ...
   processing.process()
}


method reportError(errorData) {
   specificErrorData = map(errorData)
   concreteComponent2.handleError(specificErrorData)
}
```

ErrorHandlingImpl

```
 field mediator: Mediator

method handleError(specificErrorData ) {
   ...
}
```

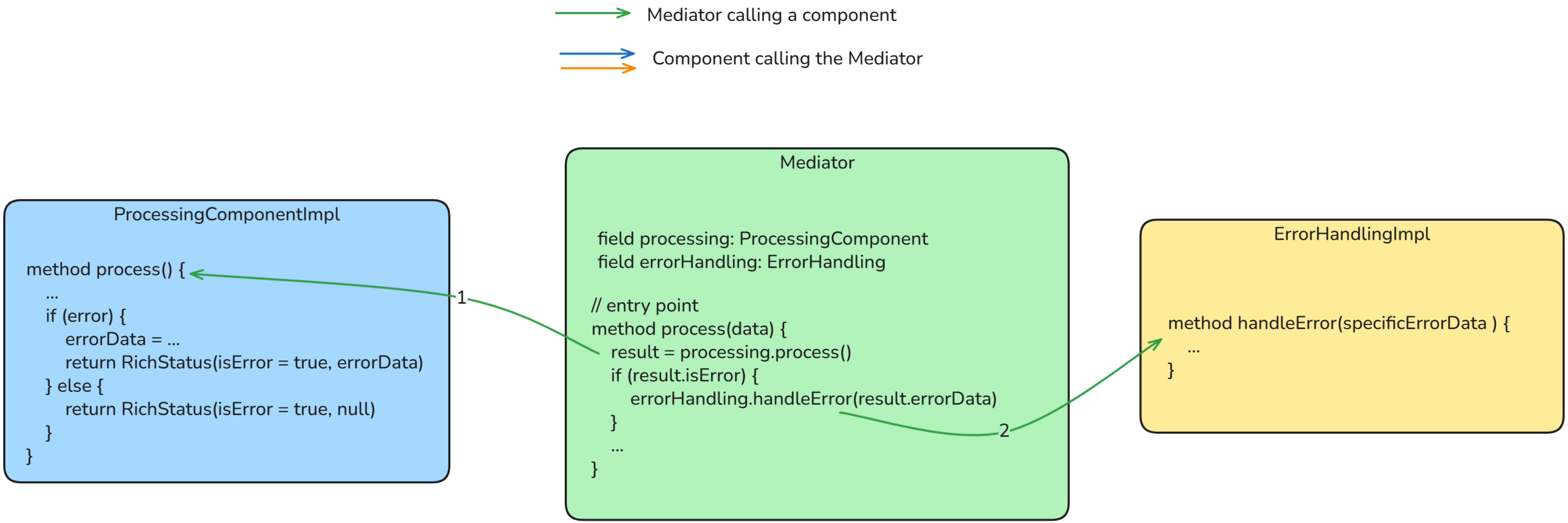
Facade-like Implementation

Uses Rich Objects (DTOs) to arrange object communication

Simple and may require direct changes to extend capabilities

Good encapsulation of coordination logic

Facade-like Implementation



→ Mediator calling a component

→ Component calling the Mediator

ProcessingComponentImpl

```
method process() {
   ...
   if (error) {
      errorData = ...
      return RichStatus(isError = true, errorData)
   } else {
      return RichStatus(isError = true, null)
   }
}
```

Mediator

```
 field processing: ProcessingComponent
 field errorHandling: ErrorHandling

// entry point
method process(data) {
   result = processing.process()
   if (result.isError) {
      errorHandling.handleError(result.errorData)
   }
   ...
}
```

ErrorHandlingImpl

```
method handleError(specificErrorData ) {
   ...
}
```

Event-Driven Implementation

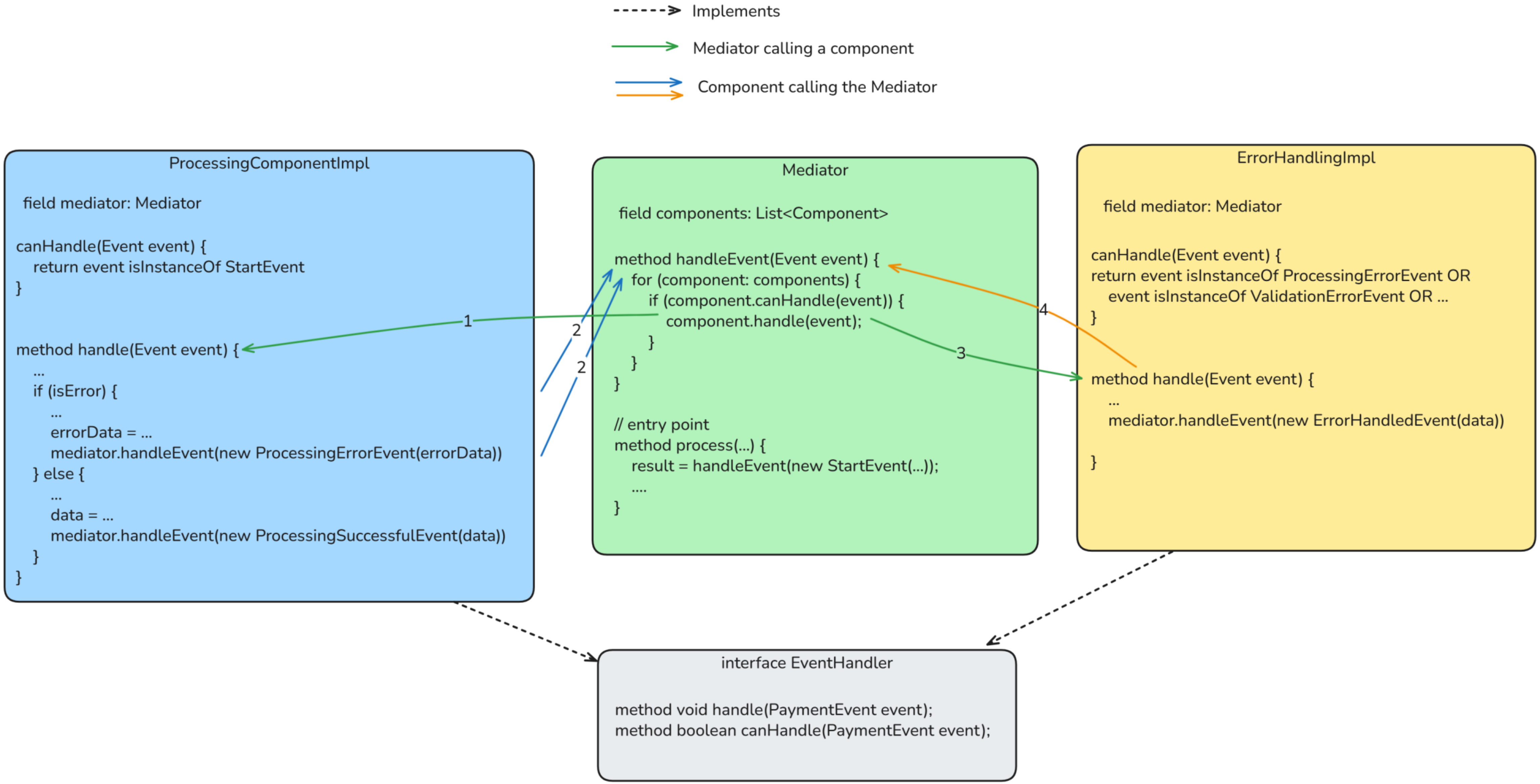
Uses Rich Objects (DTOs) as event messages

Typically allows multiple components to handle the same event

Good encapsulation of coordination logic and solid extensibility

Enables easy communication protocol changes

Event-Driven Implementation



⇢ Implements

→ Mediator calling a component

⇉ Component calling the Mediator

```
ProcessingComponentImpl

 field mediator: Mediator

canHandle(Event event) {
   return event isInstanceOf StartEvent
}


method handle(Event event) {
   ...
   if (isError) {
      ...
      errorData = ...
      mediator.handleEvent(new ProcessingErrorEvent(errorData))
   } else {
      ...
      data = ...
      mediator.handleEvent(new ProcessingSuccessfulEvent(data))
   }
}
```

```
Mediator

 field components: List<Component>

method handleEvent(Event event) {
   for (component: components) {
      if (component.canHandle(event)) {
         component.handle(event);
      }
   }
}

// entry point
method process(...) {
   result = handleEvent(new StartEvent(...));
   ....
}
```

```
ErrorHandlingImpl

 field mediator: Mediator

canHandle(Event event) {
return event isInstanceOf ProcessingErrorEvent OR
   event isInstanceOf ValidationErrorEvent OR ...
}

method handle(Event event) {
   ...
   mediator.handleEvent(new ErrorHandledEvent(data))

}
```

```
interface EventHandler

method void handle(PaymentEvent event);
method boolean canHandle(PaymentEvent event);
```

General Benefits

Makes target components closer to Single Responsibility Principle

Reduces dependencies and coupling between objects/components

Increases re-usability of target components

Follows Open/Closed Principle*

- some implementation may violate this

General Drawbacks

May become God object

Incorrect use may lead to unwanted complexity

Integration may require lots of changes on target components

Integration with other Patterns

Strategy/Chain of Responsibility patterns – mechanisms to handle events in Event-Driven Mediator

Proxy/Decorator patterns - adding extra logic for components;
Mediator may handle proxy creation

Transformation into other Patterns

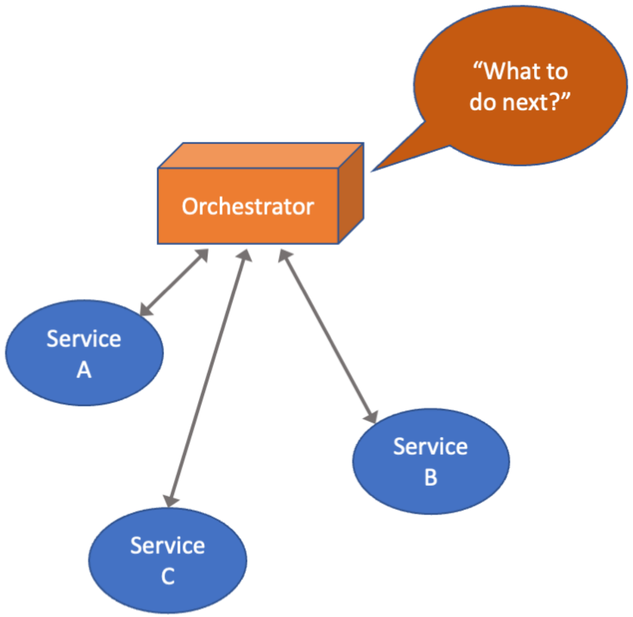
Facade pattern – personally, seems like interchangeable with Mediator in real apps or context-dependent

Abstract Factory pattern – Mediator may have extra responsibility to instantiate target components for specific

Observer pattern – Mediator may have extra responsibility to instantiate target components for specific

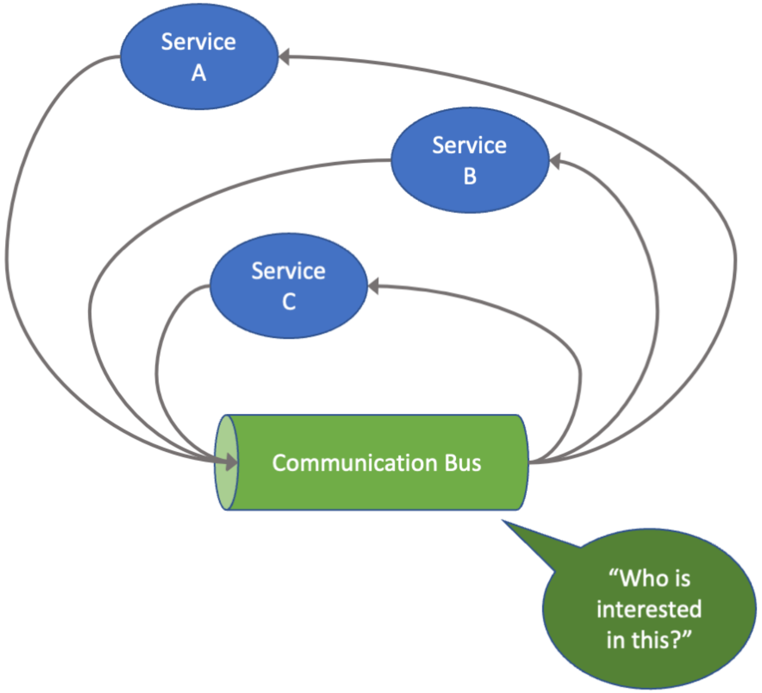
Extra analogy for Mediator vs Observer

Orchestration



Mediator

Choreography



Observer

Thank you

- Author: Serhii Kravchuk
- My LinkedIn: Link
- Date: September 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn