# flex: no frameworks required

presented by maxim porges

# who is this guy?


Team Lead/Software Engineer

# in this session

stay here if one or more of the following apply to you:

- ✔ you like writing Flex apps
- ✔ you don't want to reinvent the wheel
- ✔ you haven't found a Flex framework you like yet
- ✔ you want to learn more about the patterns behind Flex development

# why frameworks?

# frameworks give us

best practice

# frameworks give us

best practice

# frameworks give us

familiarity

mindshare

best practice

# frameworks give us

familiarity

mindshare

best practice

# frameworks give us

community

familiarity

mindshare

best practice

common solutions

# frameworks give us

community

familiarity


but aren't we just
seeking patterns?

# the flex difference

flex presents challenges to its target audience

- ? stateful - more like a desktop app
- ? asynchronous
- ? data binding

- ✓ each flex app presents common tasks
- ✓ for each task there are common solutions

# the flex difference

flex presents challenges to its target audience

- ? stateful - more like a desktop app
- ? asynchronous
- ? data binding

unfamiliar territory

- ✓ each flex app presents common tasks
- ✓ for each task there are common solutions

# the flex difference

flex presents challenges to its target audience

- ? stateful - more like a desktop app
- ? asynchronous
- ? data binding

unfamiliar territory

- ✓ each flex app presents common tasks
- ✓ for each task there are common solutions

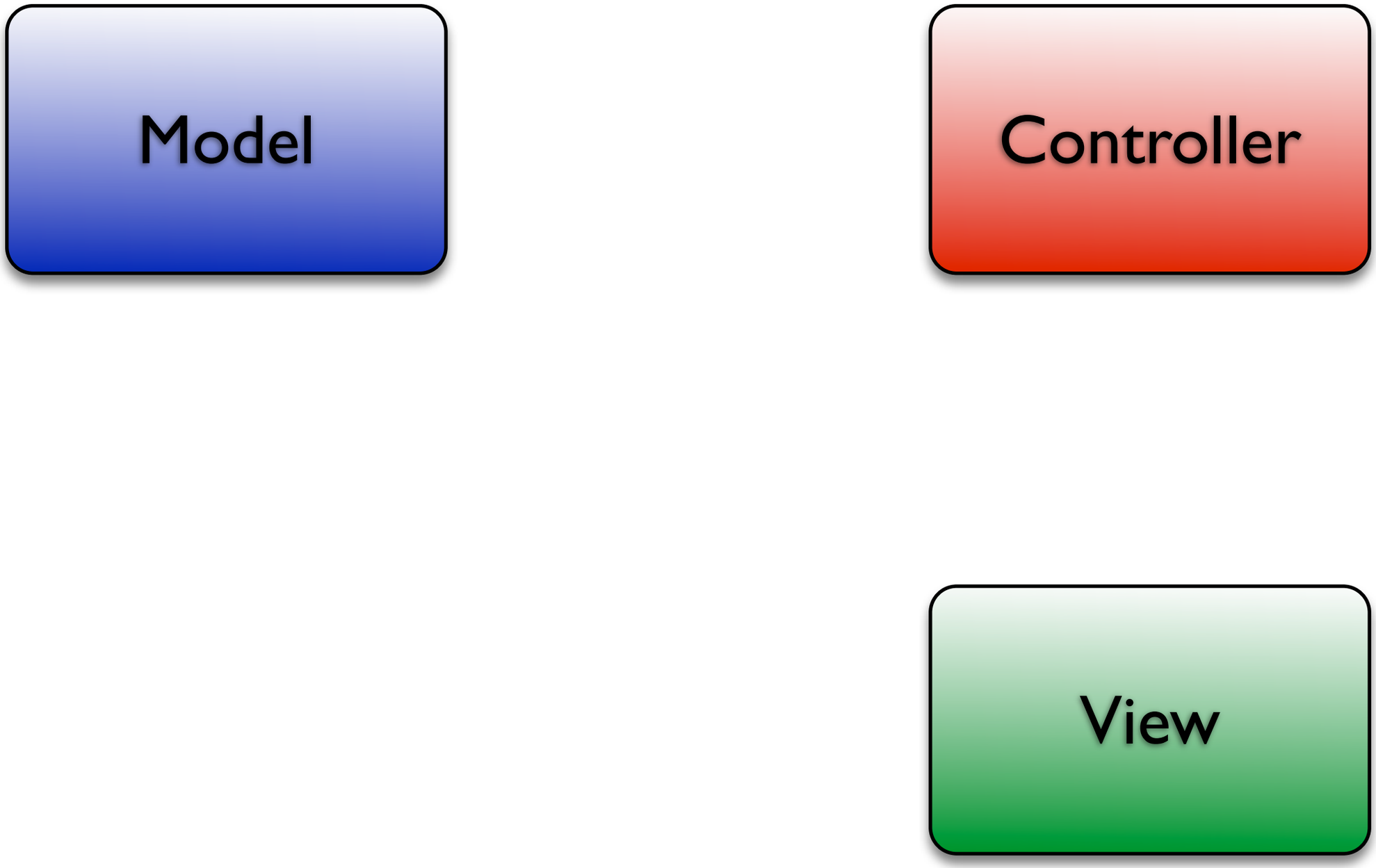
patterns

# ok - so, what next?

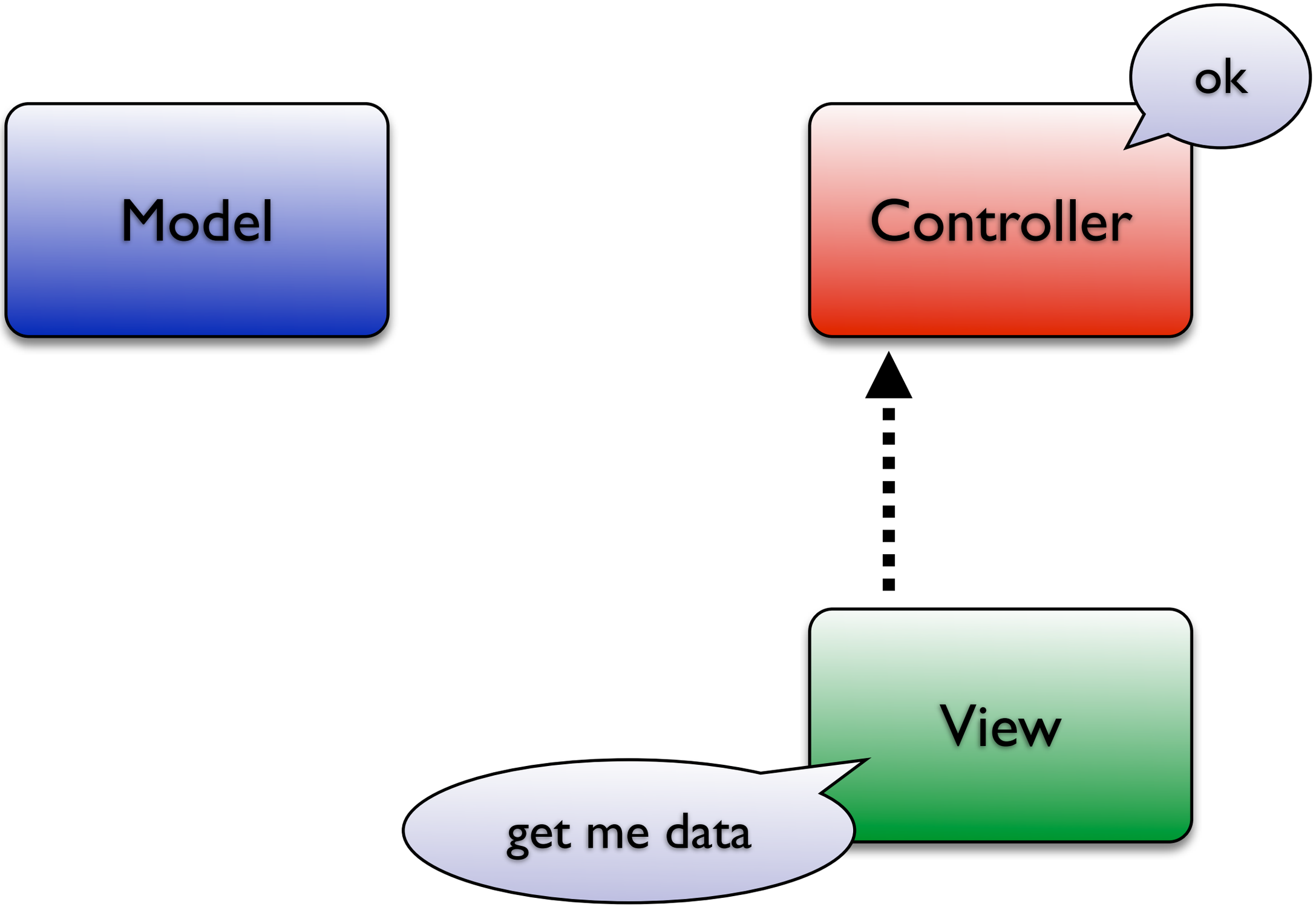
- ✔ get re-oriented with MVC
- ✔ look at common Flex development tasks
- ✔ look at solutions using patterns and code

# mvc: where it all begins

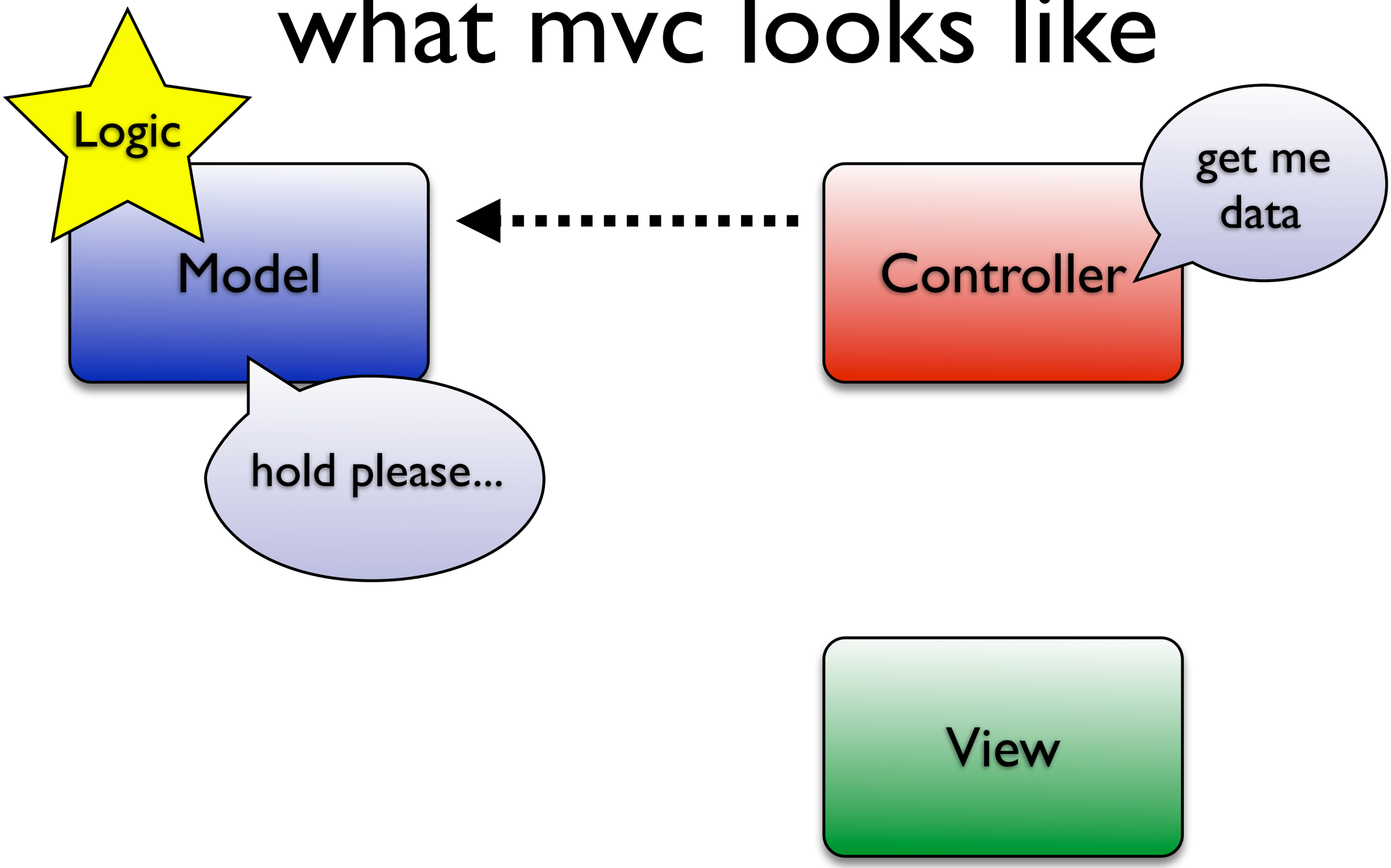
# what mvc looks like



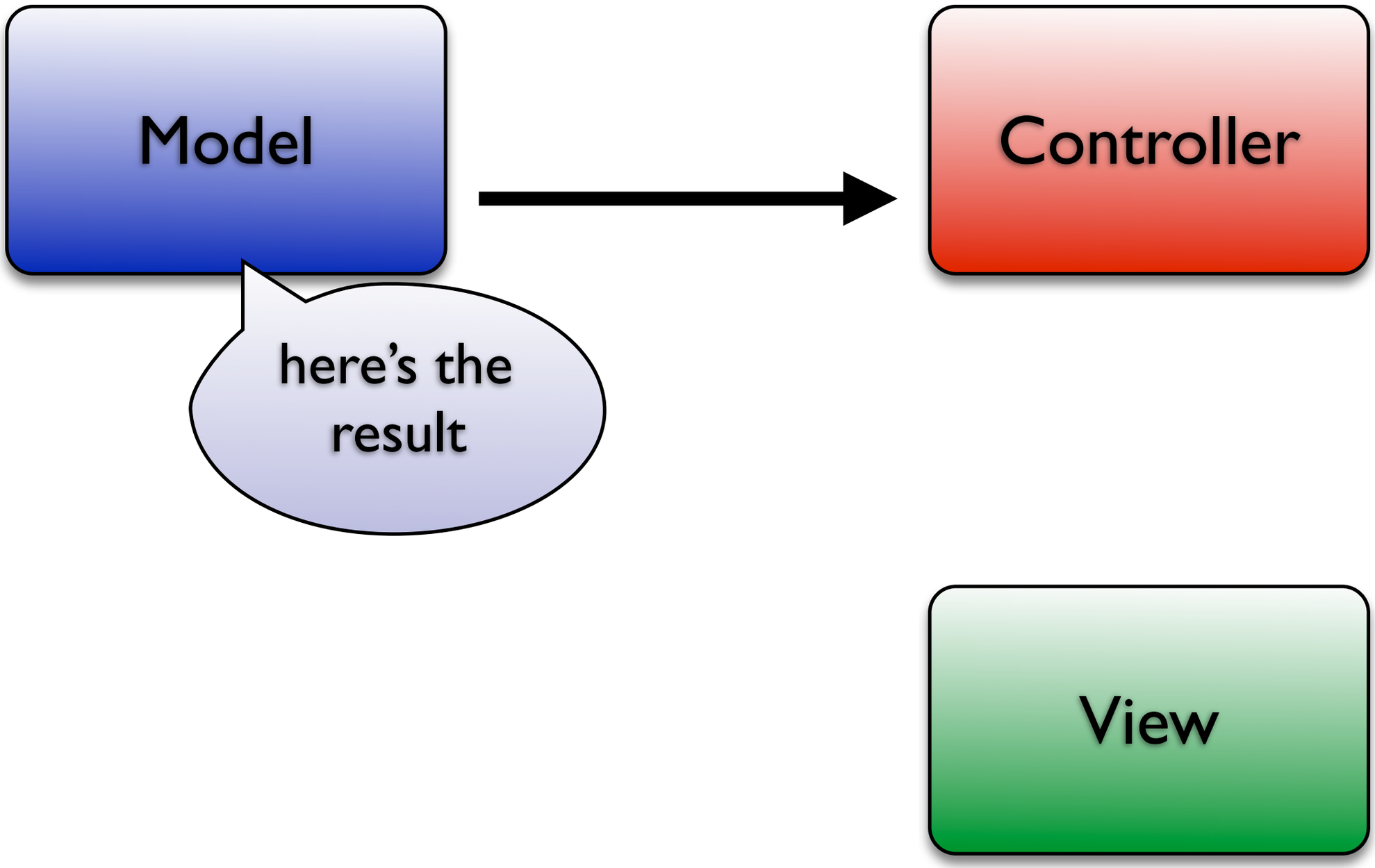
# what mvc looks like



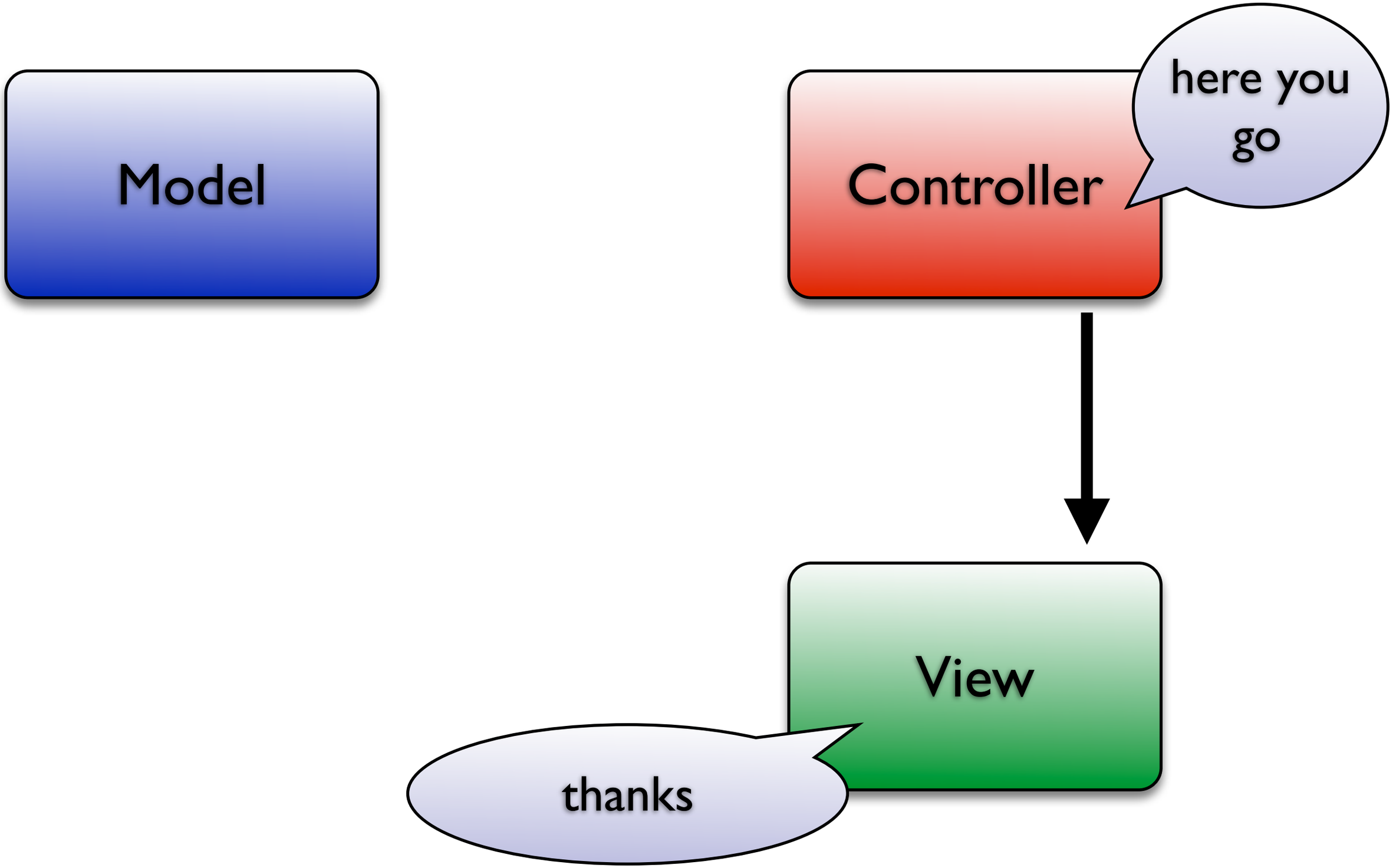
# what mvc looks like



# what mvc looks like



# what mvc looks like



# common flex tasks and solutions

# common flex tasks and solutions

challenges

patterns
and utilities

# **task:** handling events

event management is a staple of all flex apps

- ✔ events are customizable
- ✔ it's easy to handle reactions to events
- ✔ events promote asynchronous processing
- ✔ events promote loose coupling

# **solution:** Observer pattern

🎯 something is a target, which changes

👁 observers watch the target, and react

✔ already baked in to Flex:

`Event/EventDispatcher`

✔ events get produced by UI components and Services

✔ usually routed to Controllers for processing

🚫 it can be difficult to wire up all the observers

# **solution:** Observer pattern

- something is a target, which changes
- observers watch the target, and react

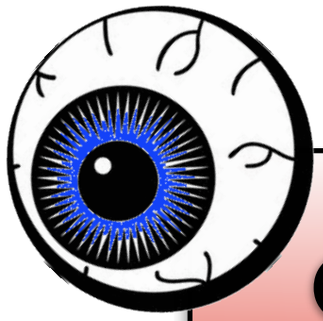
often in different ways

- already baked in to Flex:

  `Event/EventDispatcher`

- events get produced by UI components and Services
- usually routed to Controllers for processing
- it can be difficult to wire up all the observers

# Observer in action

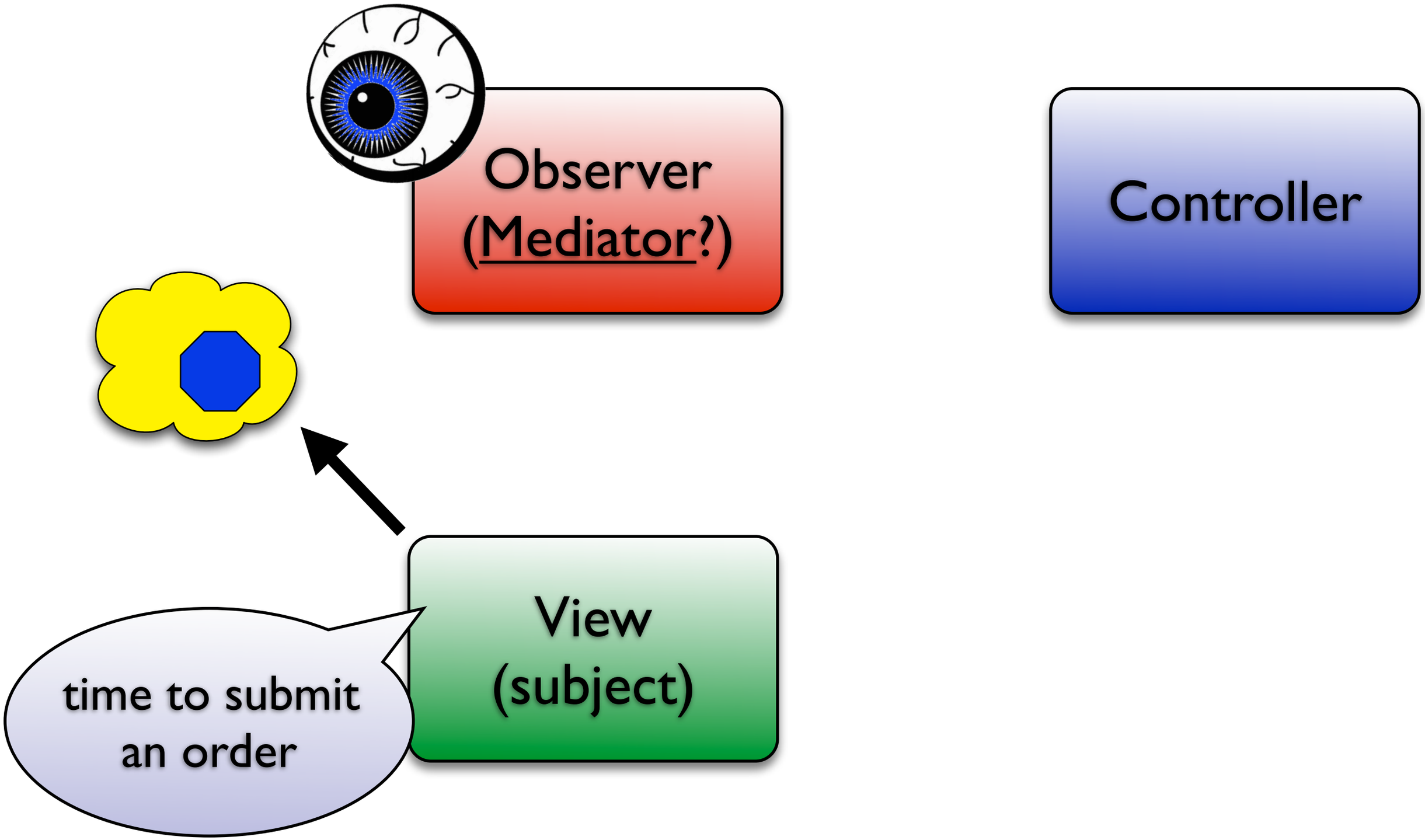


Observer
(Mediator?)

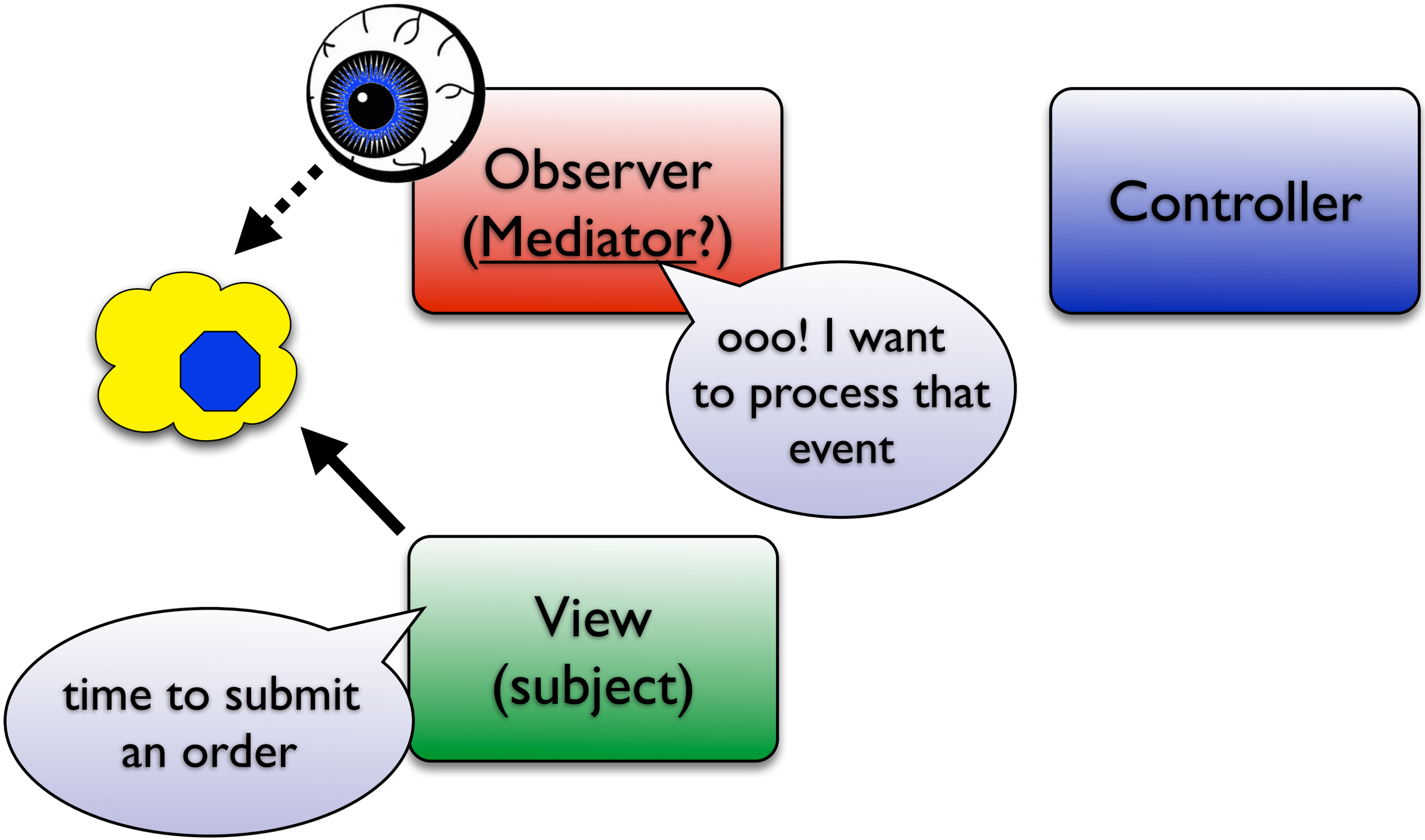
Controller

View
(subject)

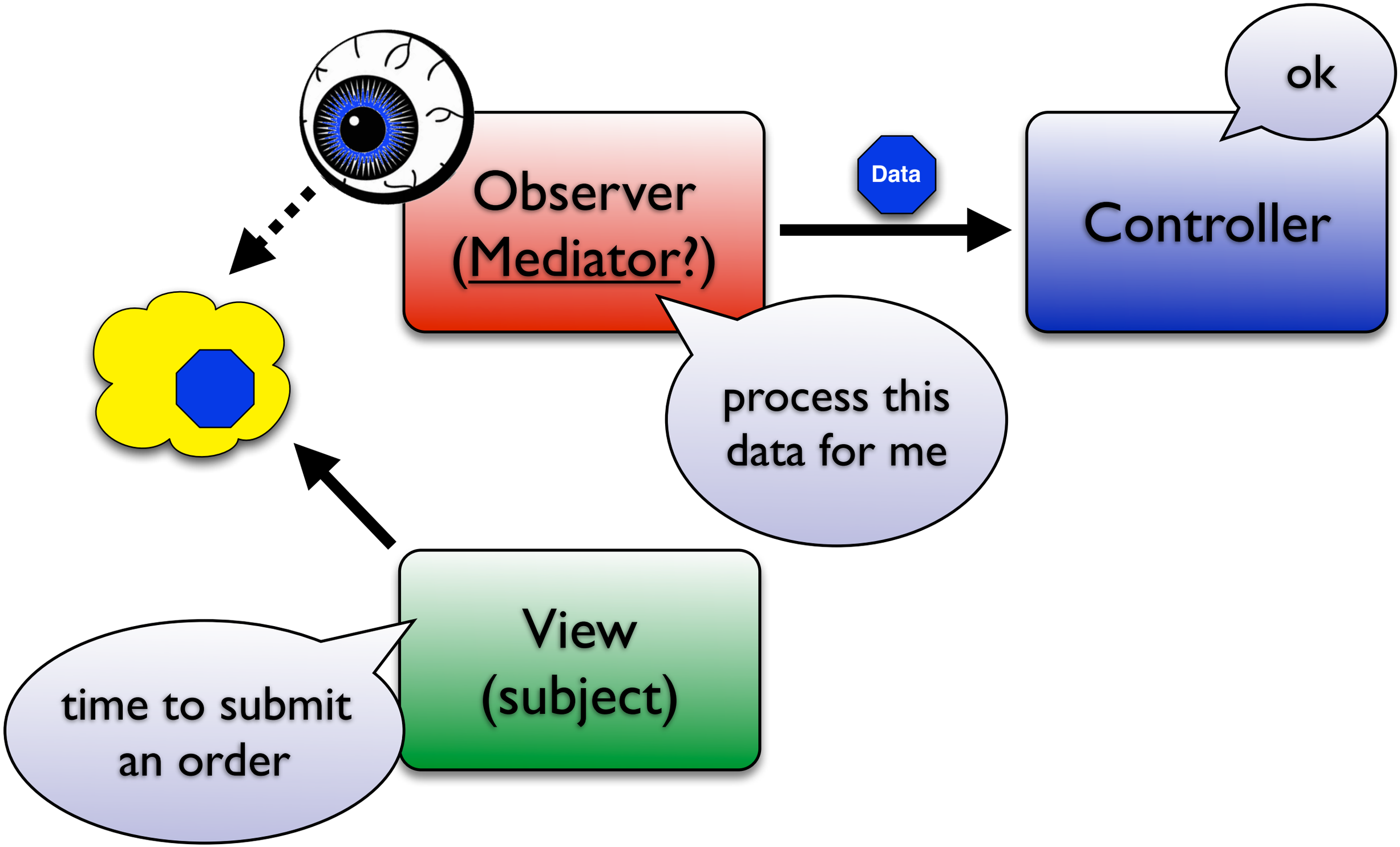
# Observer in action



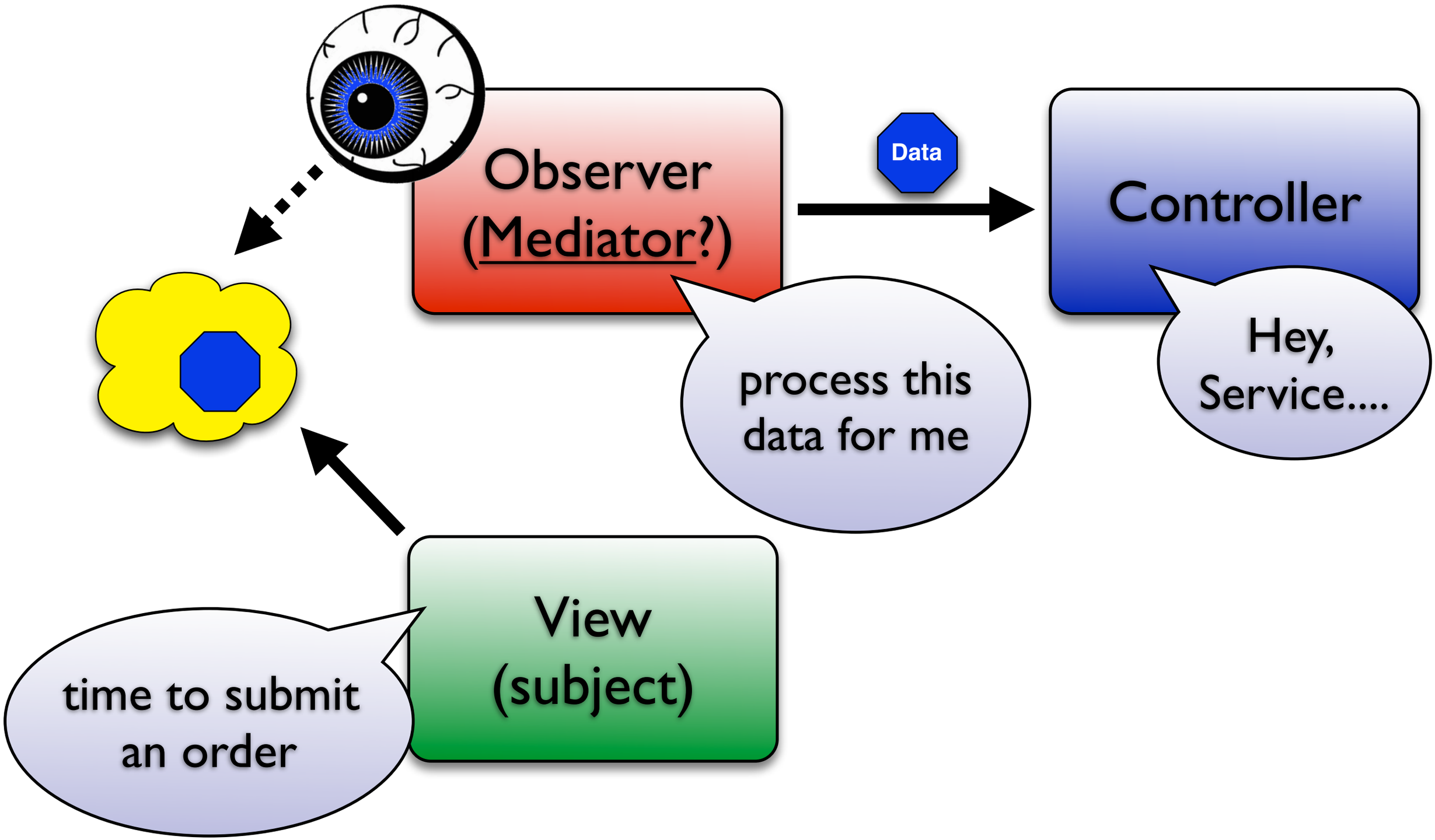
# Observer in action



# Observer in action

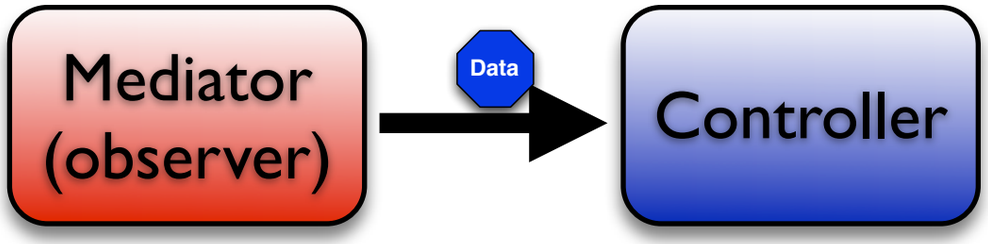


# Observer in action



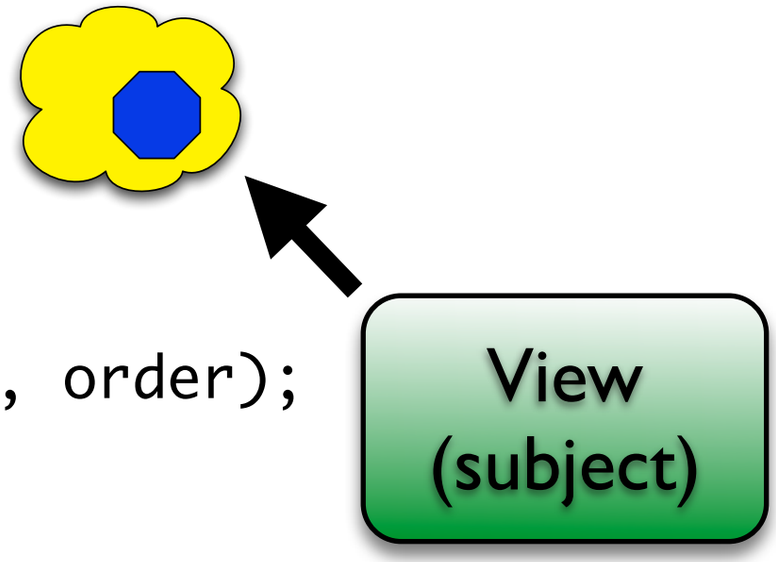
# observer (OrderMediator)

```
private function init() : void
{
   orderForm.addEventListener(
      OrderEvent.PLACE_ORDER, dispatchOrderRequest);
}

private function dispatchOrderRequest(event : OrderEvent)
{
   controller.processOrder(event.order);
}
```



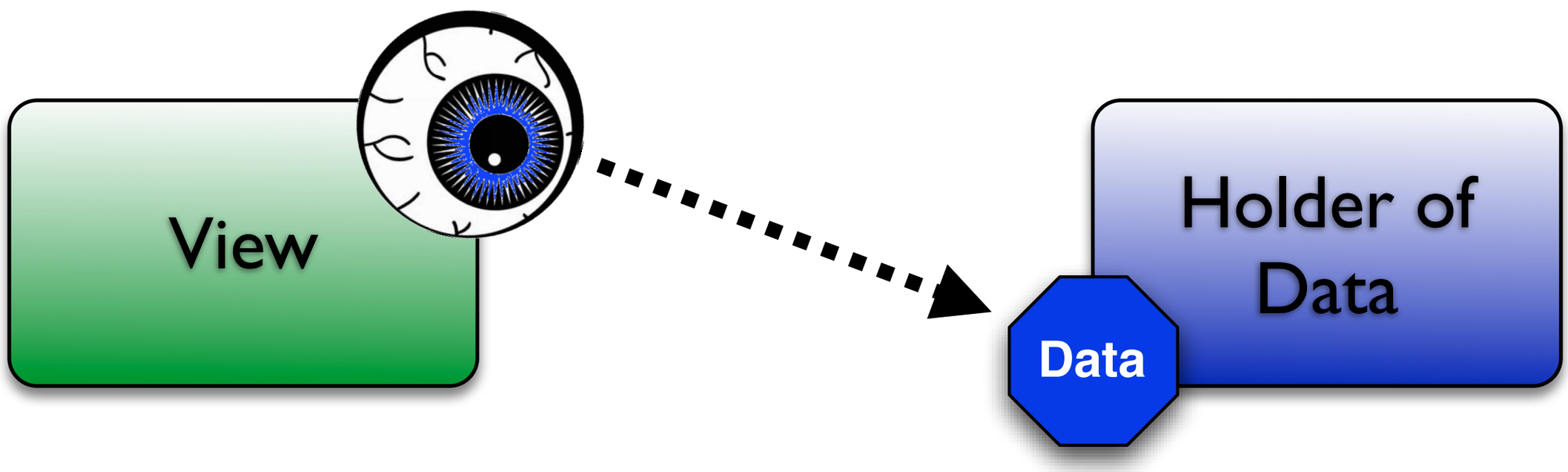

# subject (OrderForm)

```
private function orderSubmission() : void
{
   var order : Order = new Order();
   ... (fill order with data) ...
   var orderEvent : OrderEvent =
      new OrderEvent(OrderEvent.PLACE_ORDER, order);

   dispatchEvent(orderEvent);
}
```



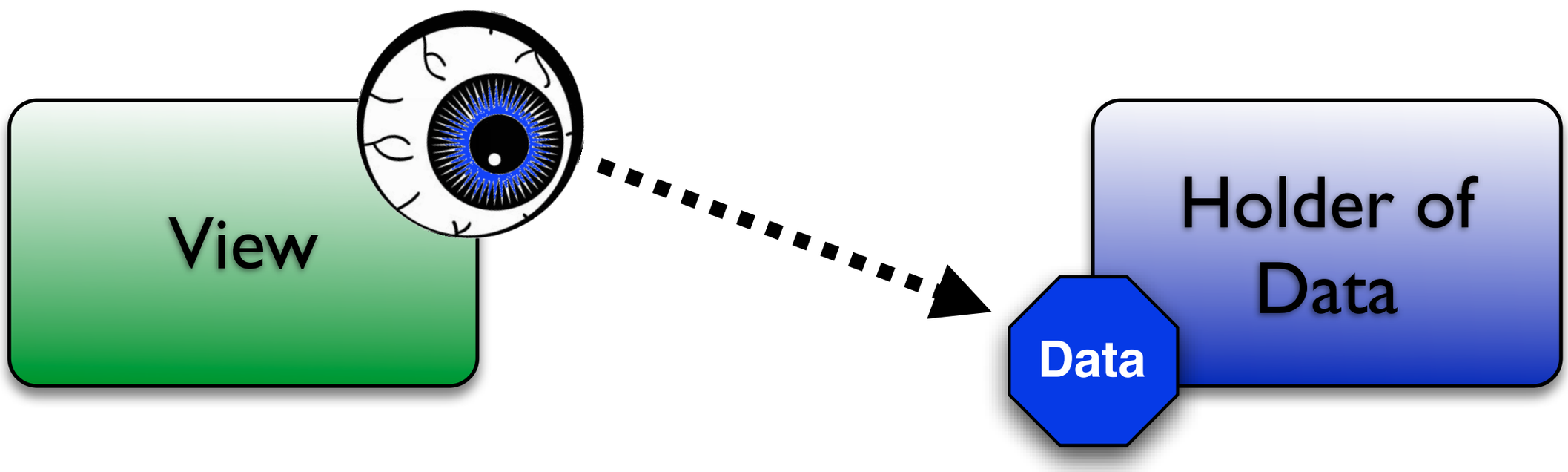
# **task:** binding to stuff

- ✔ data binding is an Observer implementation, too
- ✔ Flex uses {...} as shorthand notation



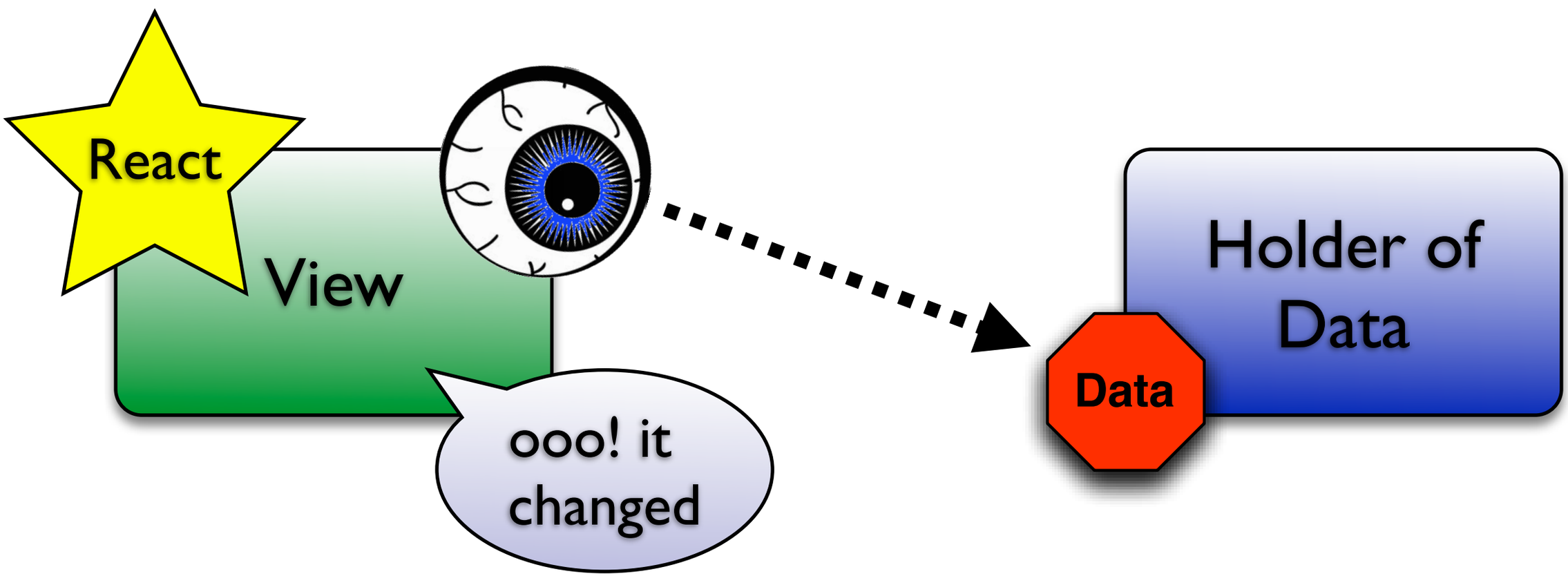
# **solution:** Observer Pattern

- ✔ data binding is an Observer implementation, too
- ✔ Flex uses {...} as shorthand notation



# **solution:** Observer Pattern

- ✔ data binding is an Observer implementation, too
- ✔ Flex uses {...} as shorthand notation



# issues with Observer

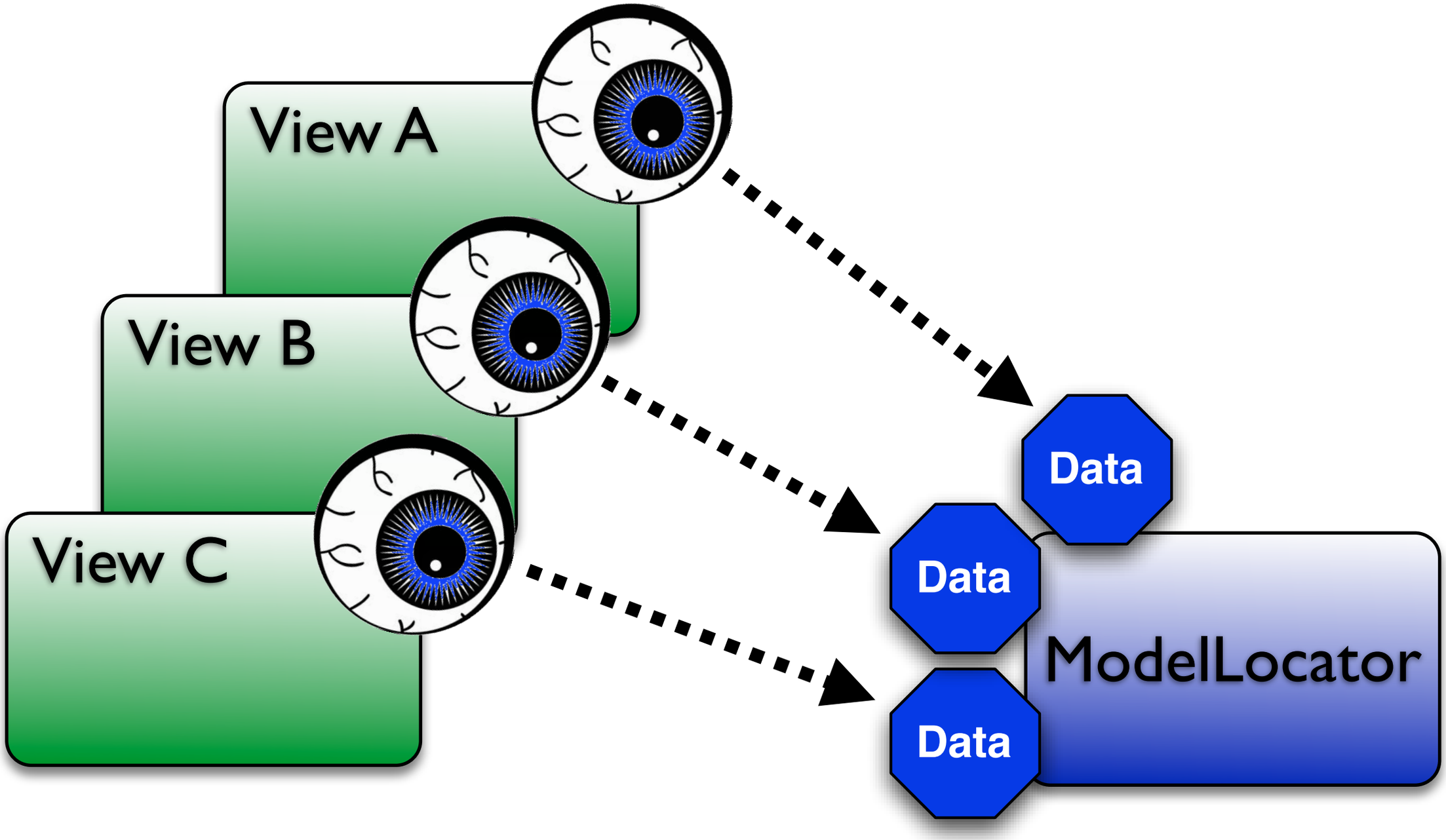
🚫 bindings are great, but promote coupling

? how do you wire all the Observers together?

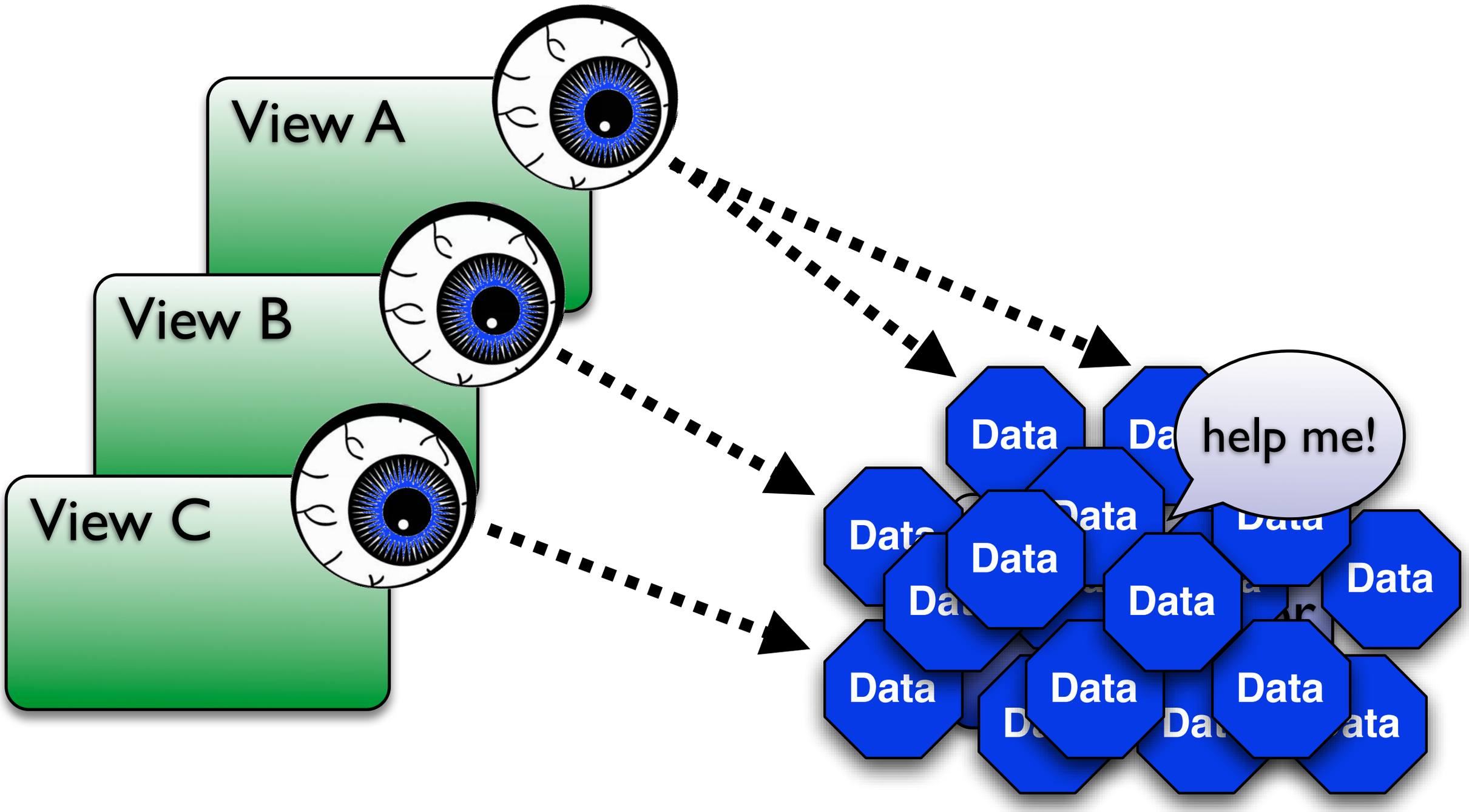
# **solution:** bind to a common place

- ✔ Controllers are a great place to put things... follows original MVC approach
- ? what about ModelLocator?

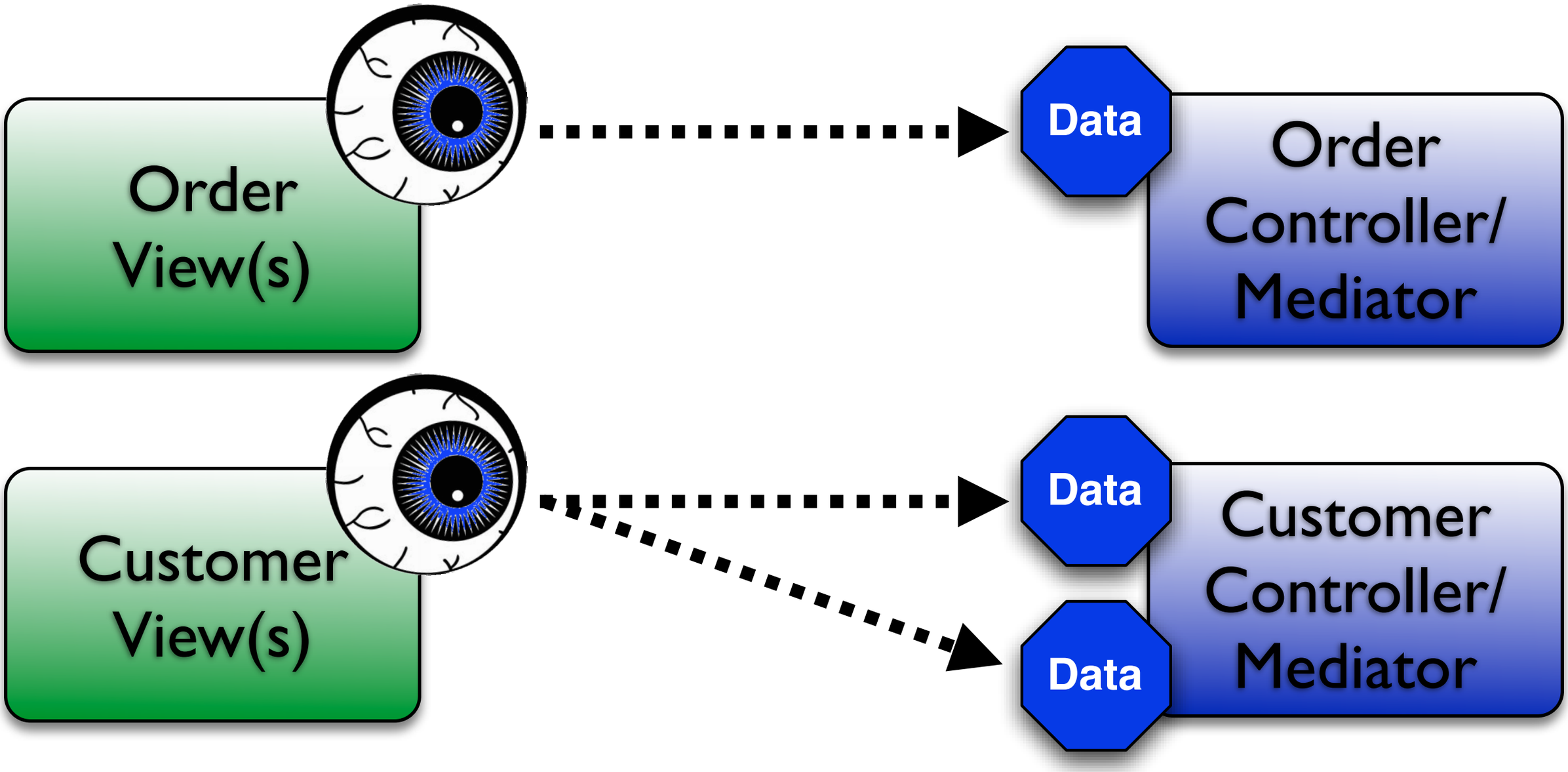
# ModelLocator in Cairngorm



# ModelLocator in Cairngorm



# an alternate implementation




etc...

```
public class CartController extends EventDispatcher
{
    [Bindable]
    public var products : ArrayCollection;
    public var cart : ShoppingCart;

    private var _gateway : ProductGateway;

    public function CartController()
    {
        products = new ArrayCollection();
        cart = new ShoppingCart();

        _gateway = new ProductGateway();
        _gateway.fillWithProducts(products);
    }
}
```

```
<view:ShoppingCartView
  id="shoppingCartComp"
  width="100%"
  height="100%"
  cart="{Registry.cartController.cart}" />
```

---

Inside ShoppingCartView:

```
[Bindable]
public var cart : ShoppingCart;

public function addProductToShoppingCart(product : Product) : void
{
   cart.addItemToCart(product);
}
```

# **task:** finding stuff

one of the biggest problems in Flex is finding stuff

- **?** how do Views find Mediators, Controllers, and/or bindings?
- **?** how do you find “global” objects?

# **solution:** Registry pattern

Registry is a common interface for finding things

- ✔ code calls static methods on Registry to “find” objects
- ✔ Registry interface hides how objects are instantiated/located
- ✔ usually Registry returns references to Singletons, but it doesn’t have to

# **solution:** Registry pattern

Registry is a common interface f

IoC solves this problem in other languages

- ✔ code calls static methods on R d" objects
- ✔ Registry interface hides how objects are instantiated/located
- ✔ usually Registry returns references to Singletons, but it doesn't have to

```
public class Registry
{
    private var _controller : Controller;

    // create and initialize registry
    private static var registryInstance : Registry;
    {
        registryInstance = new Registry();
        registryInstance._controller = new Controller();
    }

    public static function get controller() : Controller
    {
        return registryInstance._controller;
    }

    ... (more controllers and objects if needed)...
}
```

# **task:** organize your code

each framework presents a way to organize your code

- ✔ you've (hopefully) been doing this for years
- ⊘ ...so you probably don't need a framework to tell you how to do this
- ✔ pick something you like and stick with it

# **solution:** follow MVC/layers

- your **M** is your domain/business logic
- **V** will usually be app-specific
- **C** will always be app-specific (anybody disagree?)
- services and other integration code may or may not be closely tied to your app/**M**

# a suggested way to organize

- src
  - assets
  - com
    - mycompany
      - ...
  - events
  - view
  - controller

# a suggested way to organize

- src
  - assets
  - com
    - mycompany
      - ...
  - events
  - view
  - controller

this could even go in a separate Flex project

# **task:** asynchronous processing

another big problem is asynchronous processing

1. maintaining state
2. tying multiple send()s to specific responders
3. common responder code

# **solutions:** Responder & A.C.T.

**solution:** Responder pattern (flex-specific Observer)

Asynchronous Completion Token pattern

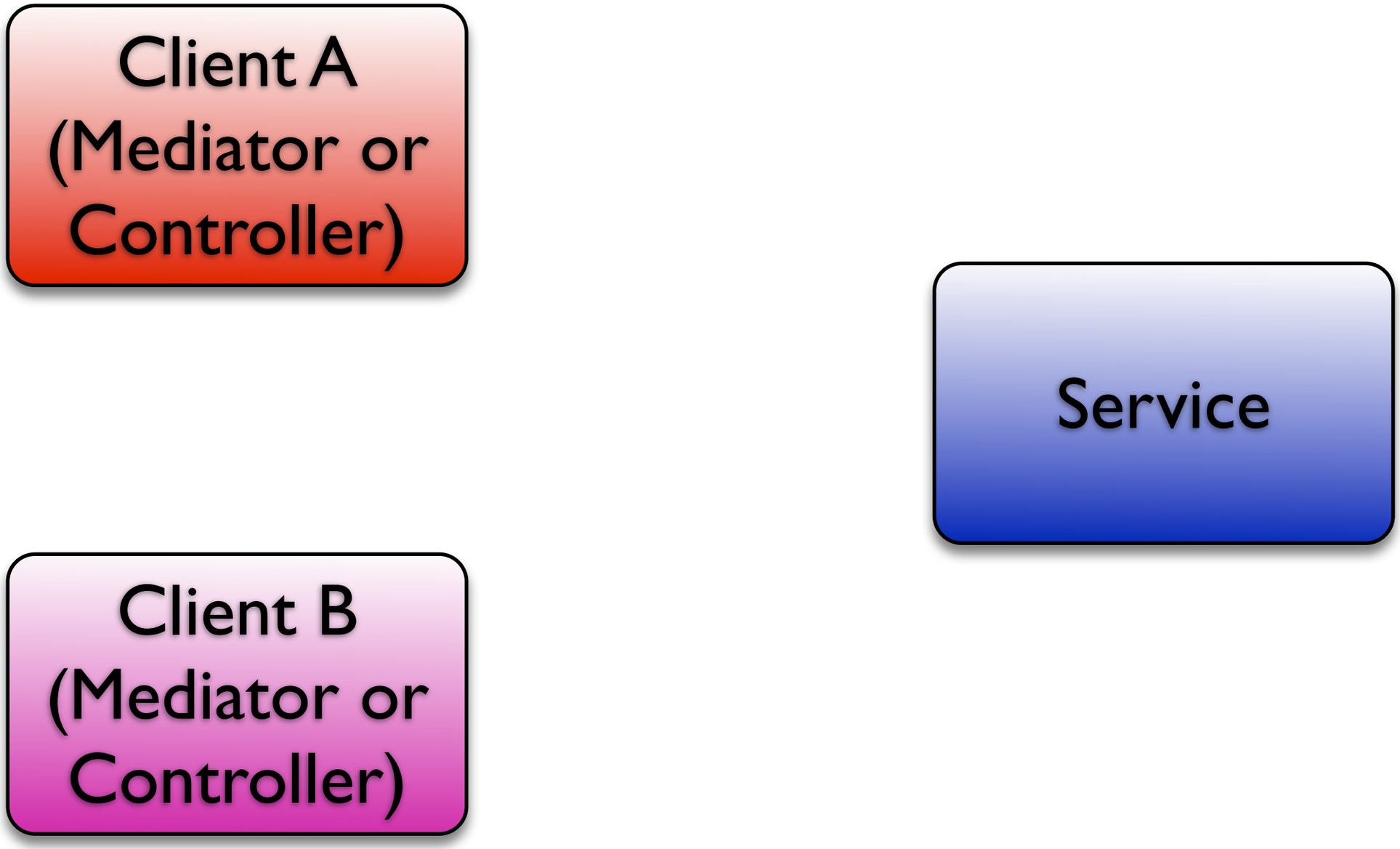
**flex classes:** `mx.rpc.AsyncToken`

`mx.rpc.IResponder`

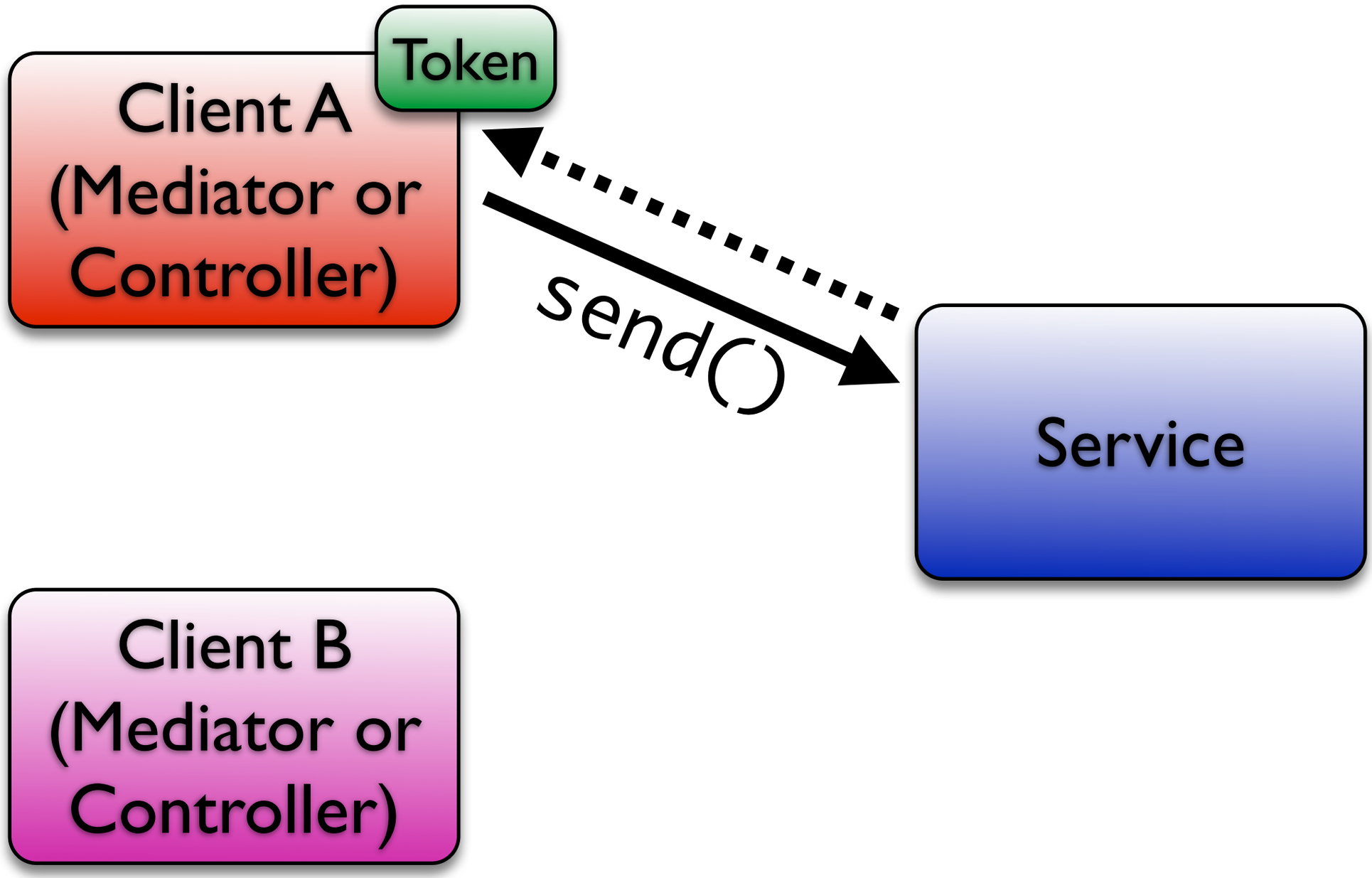
`mx.rpc.Responder`

`mx.rpc.AsyncResponder`

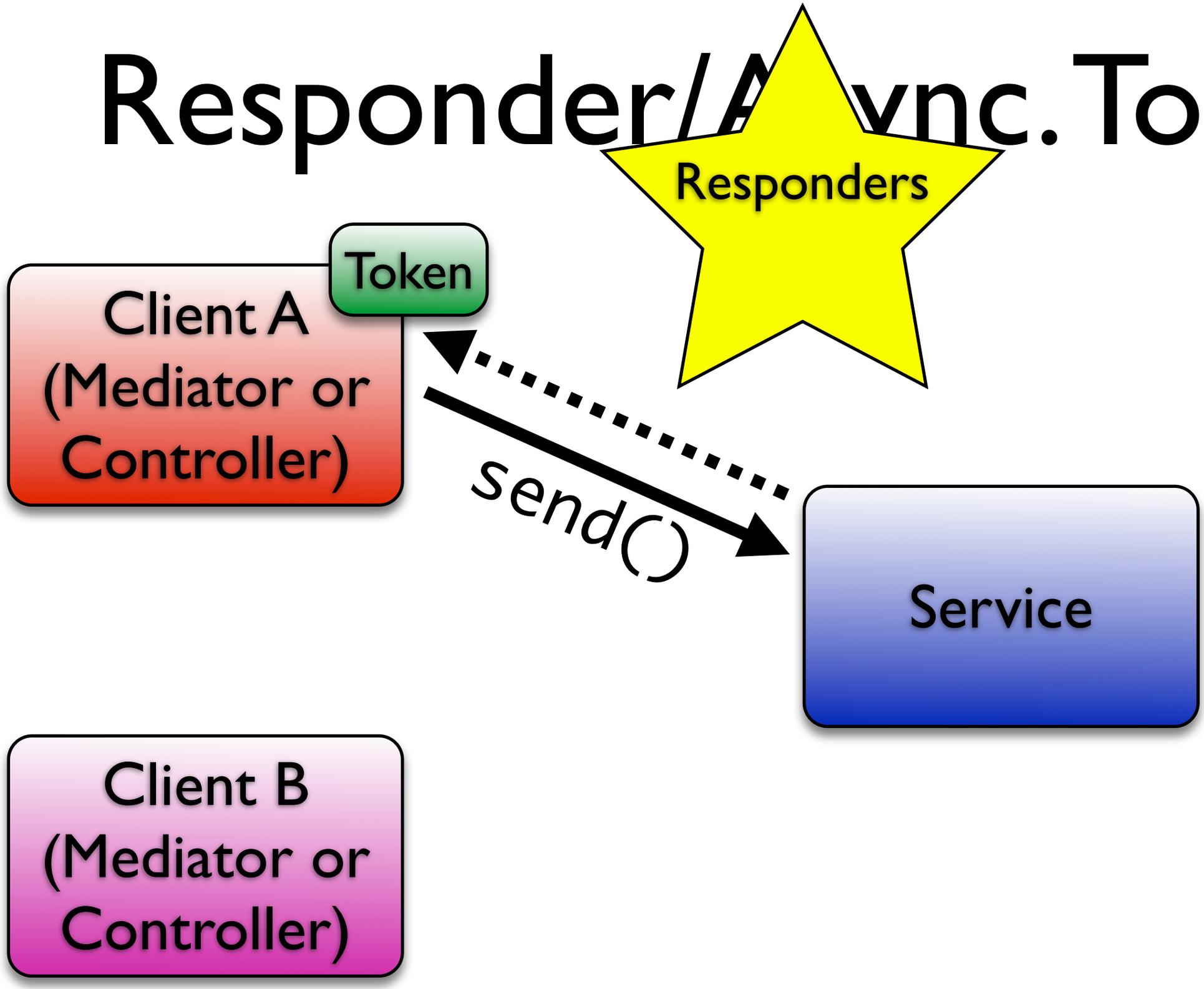
# Responder/Async. Token



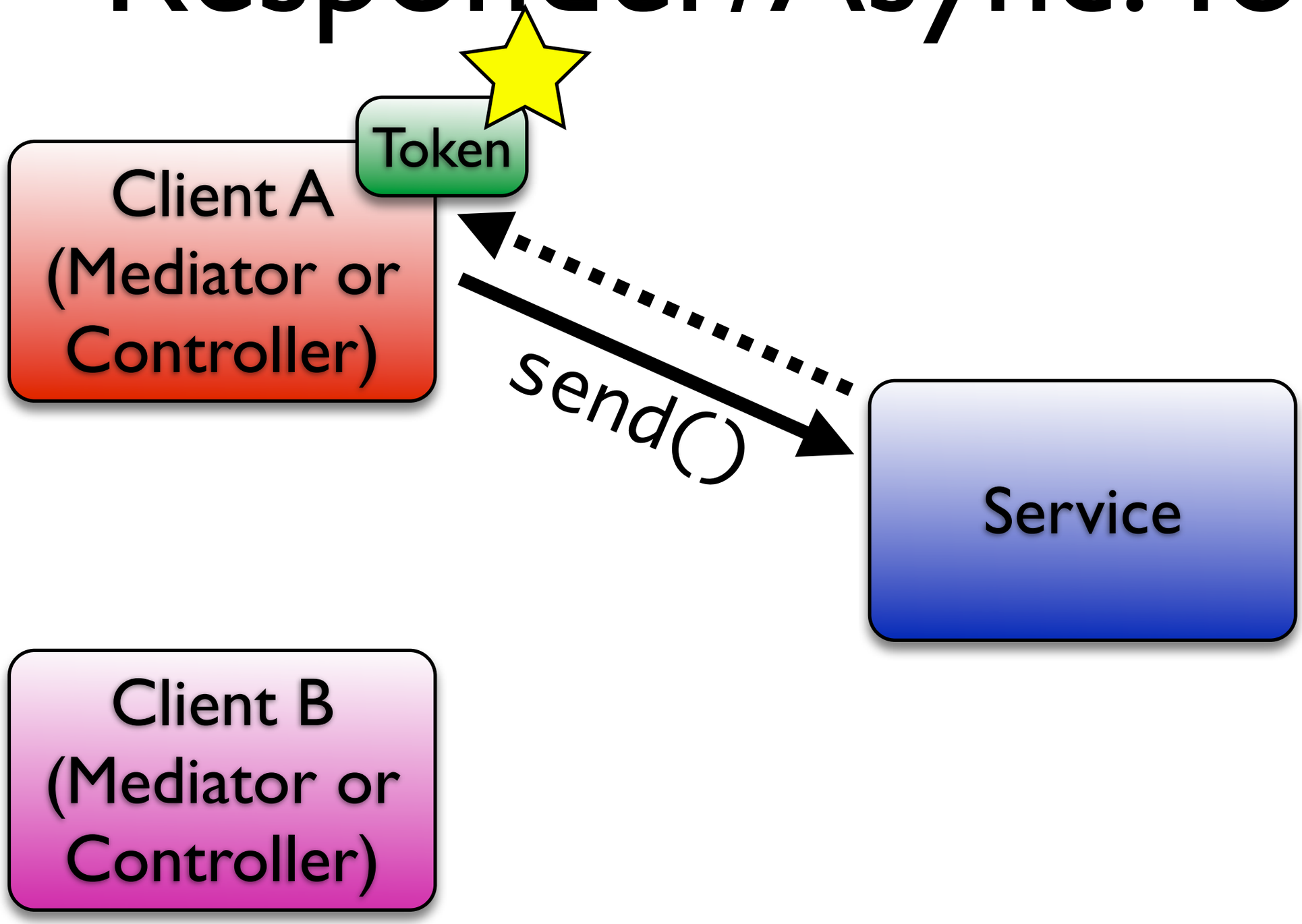
# Responder/Async. Token



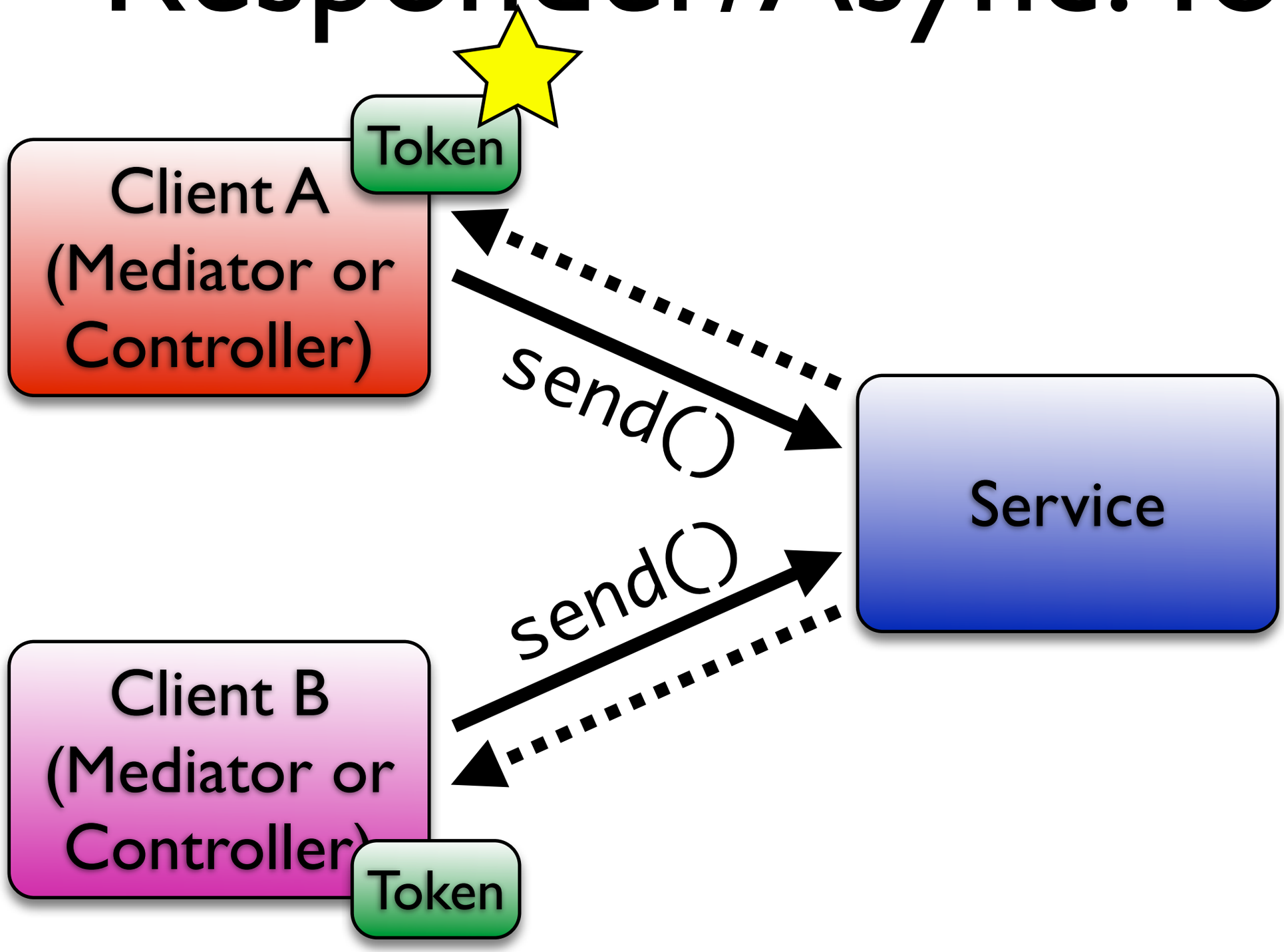
# Responder/Async. Token



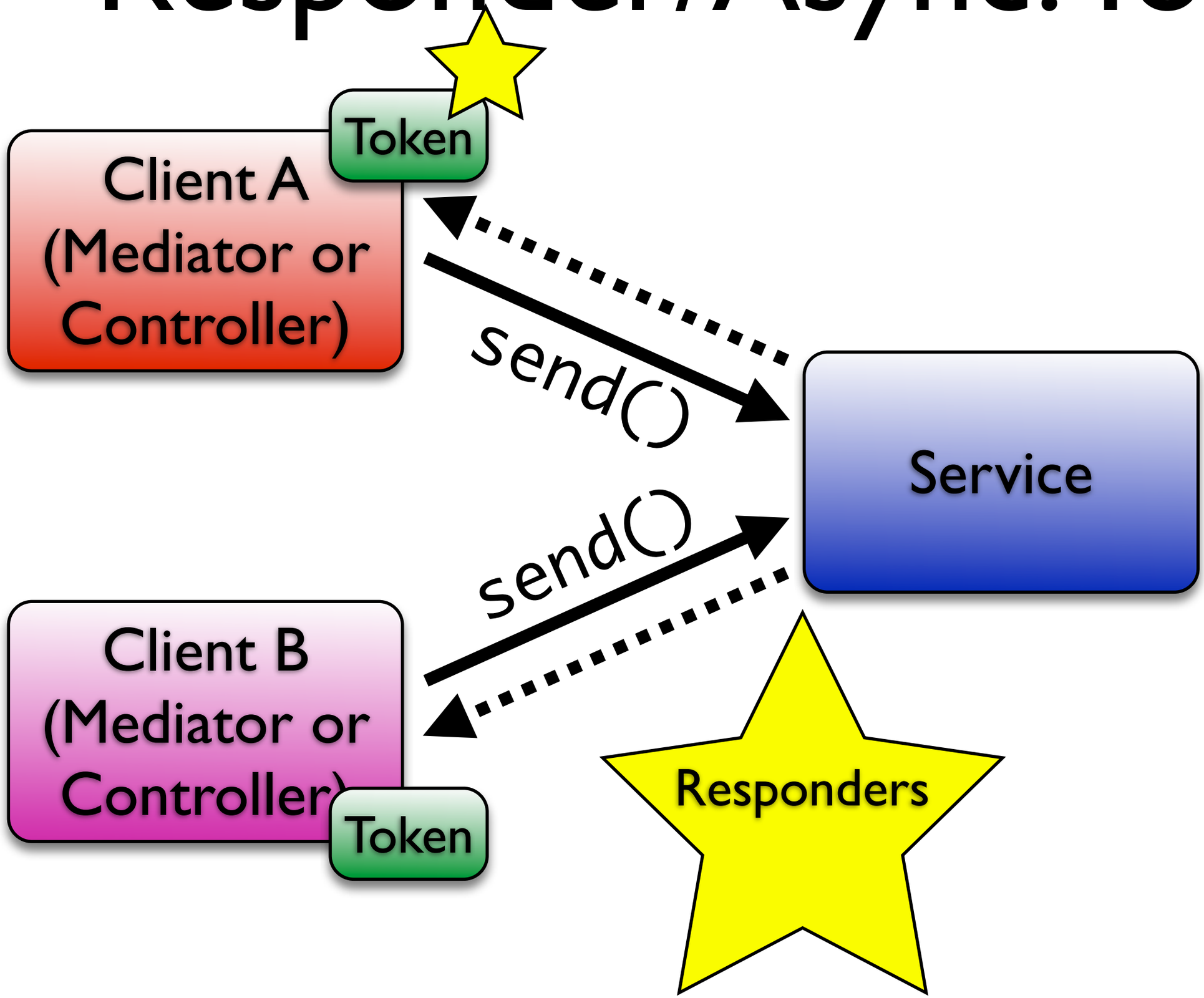
# Responder/Async. Token



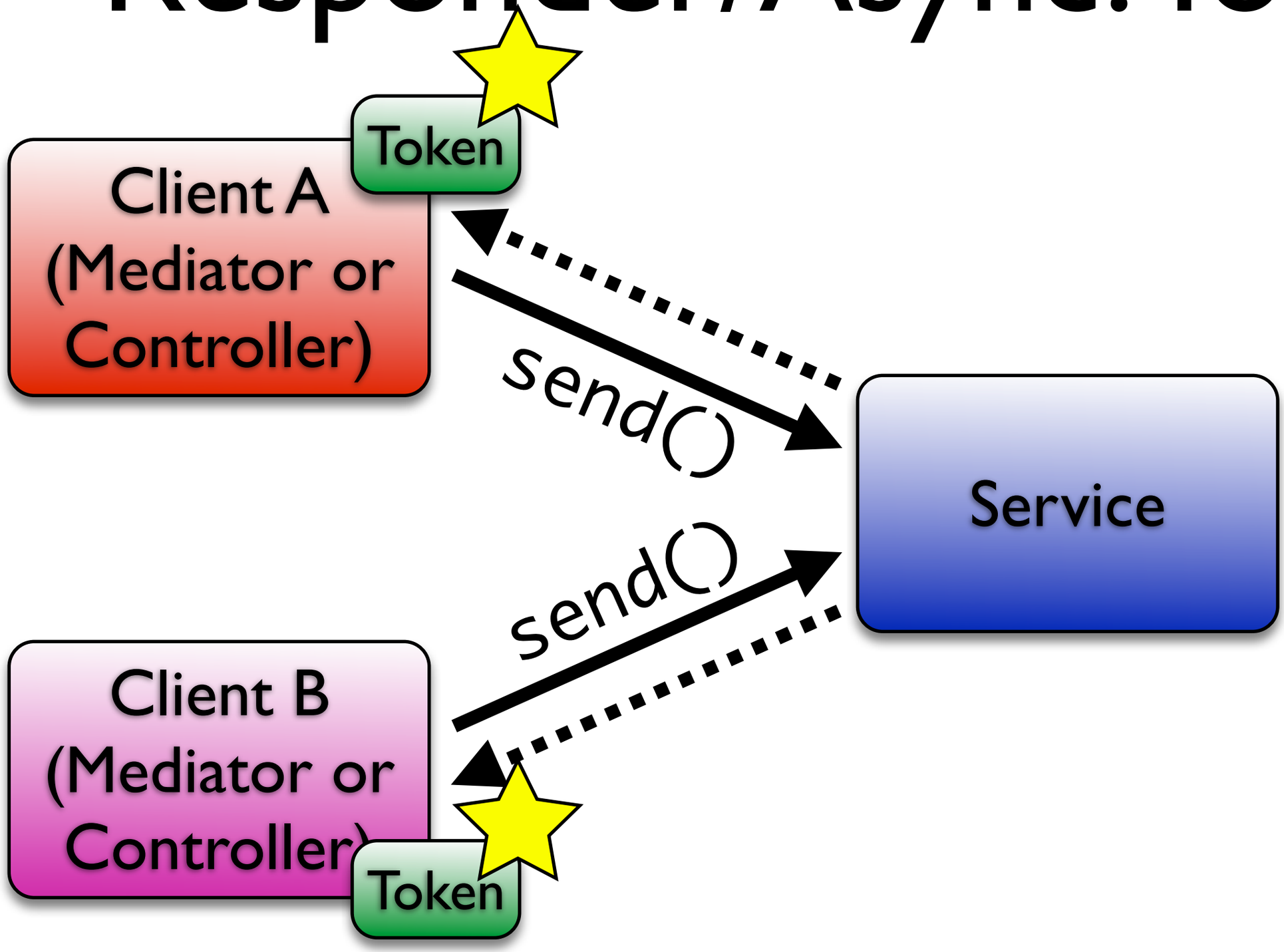
# Responder/Async. Token



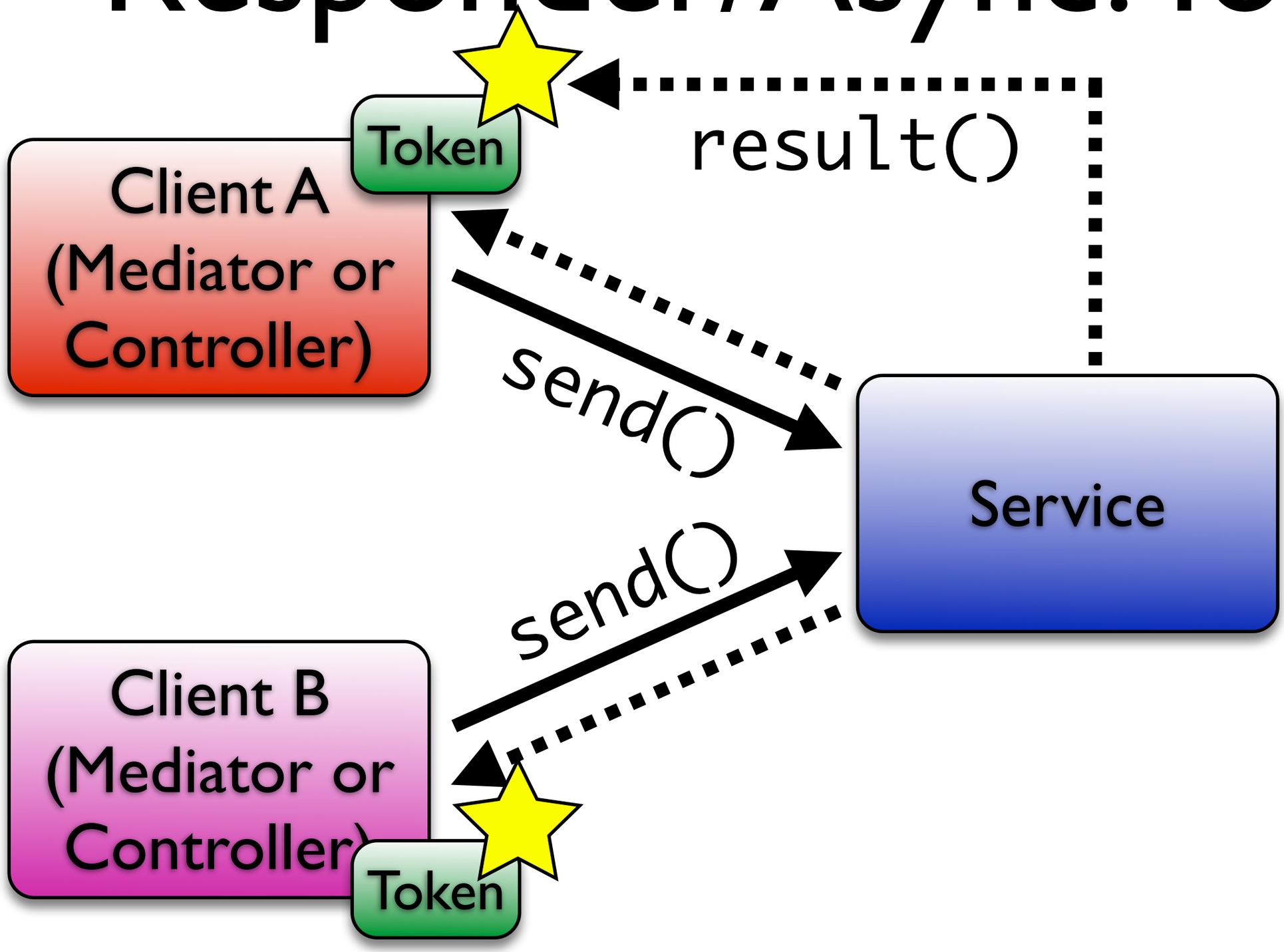
# Responder/Async. Token



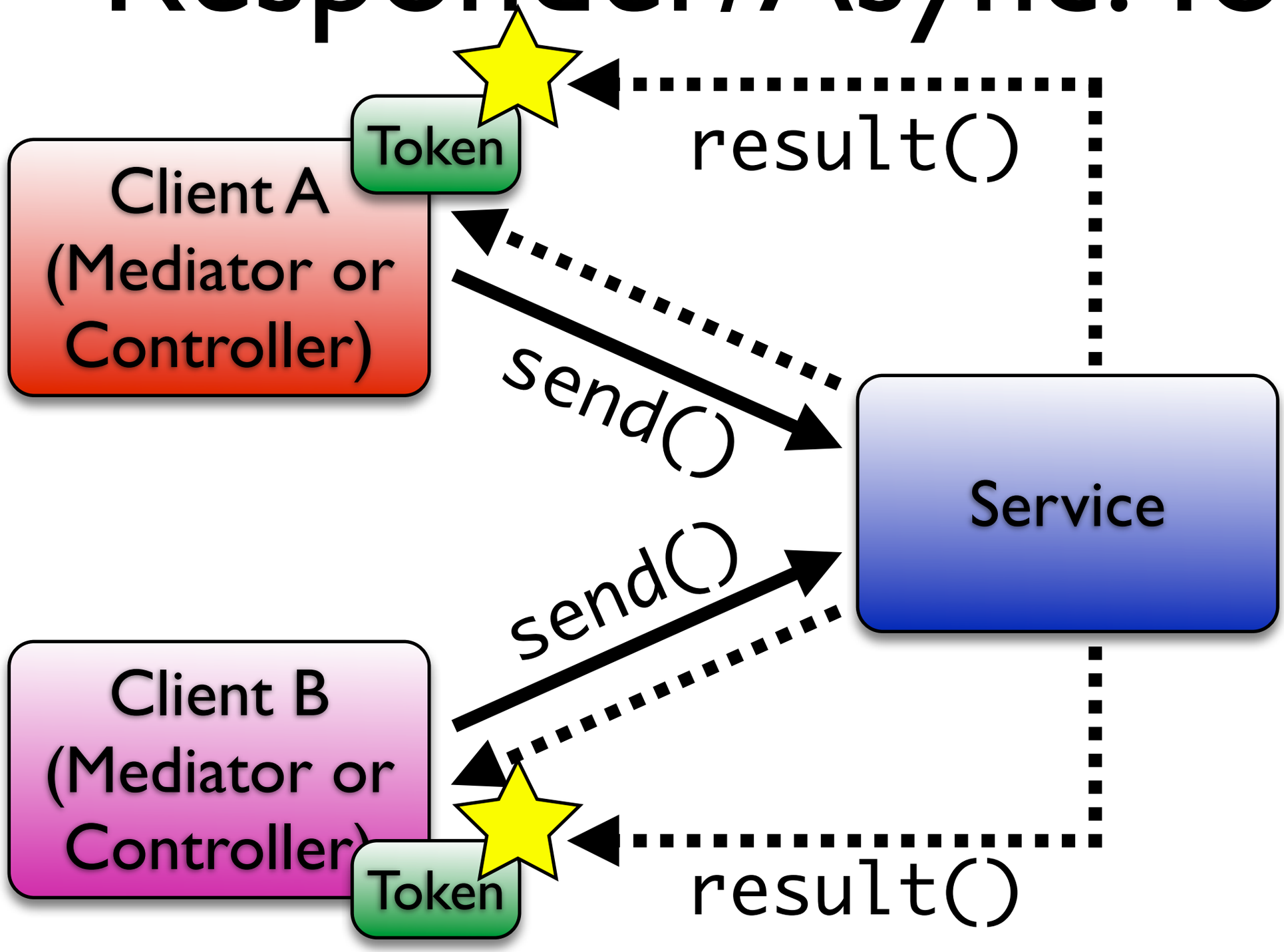
# Responder/Async. Token



# Responder/Async. Token



# Responder/Async. Token



```
public function fillWithProducts(collectionToFill : ArrayCollection)
{
    var token : AsyncToken = _productService.send();
    token.collectionToFill = collectionToFill;
    token.addResponder(new AsyncResponder(result, fault, token));
}

public function result(event : ResultEvent, token : AsyncToken) : void
{
    var collectionToFill : ArrayCollection = token.collectionToFill;

    // Convert XML to domain objects
    var response : XML = event.result as XML;
    for each (var productNode : XML in response.product)
    {
        collectionToFill.addItem(xmlToProduct(productNode));
    }
}
```

# sample application

# summary

- ✔ use a Flex framework if it supports good design
- ✔ use best practices and solid design patterns

**do these things and you'll be a happy developer**

# q&a

**contact** maxim.porges@yahoo.com
**blog** http://www.maximporges.com

thanks :)