

Flash on the Beach 2009

Application Frameworks

The Good, the Bad & the Ugly

Richard Lord

Technical Architect

BrightTALK

www.brighttalk.com

Application Frameworks

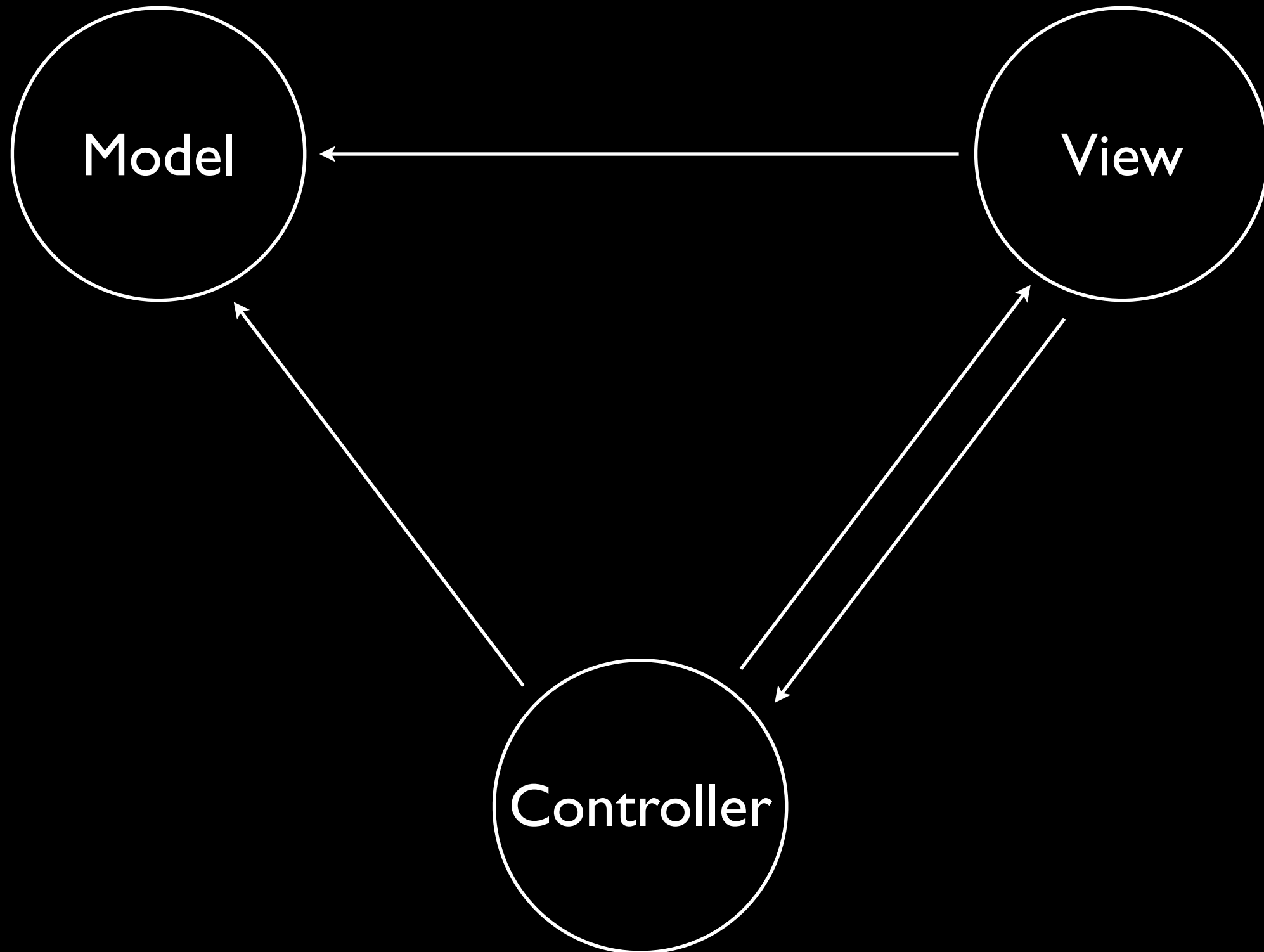
Cairngorm

Pure MVC

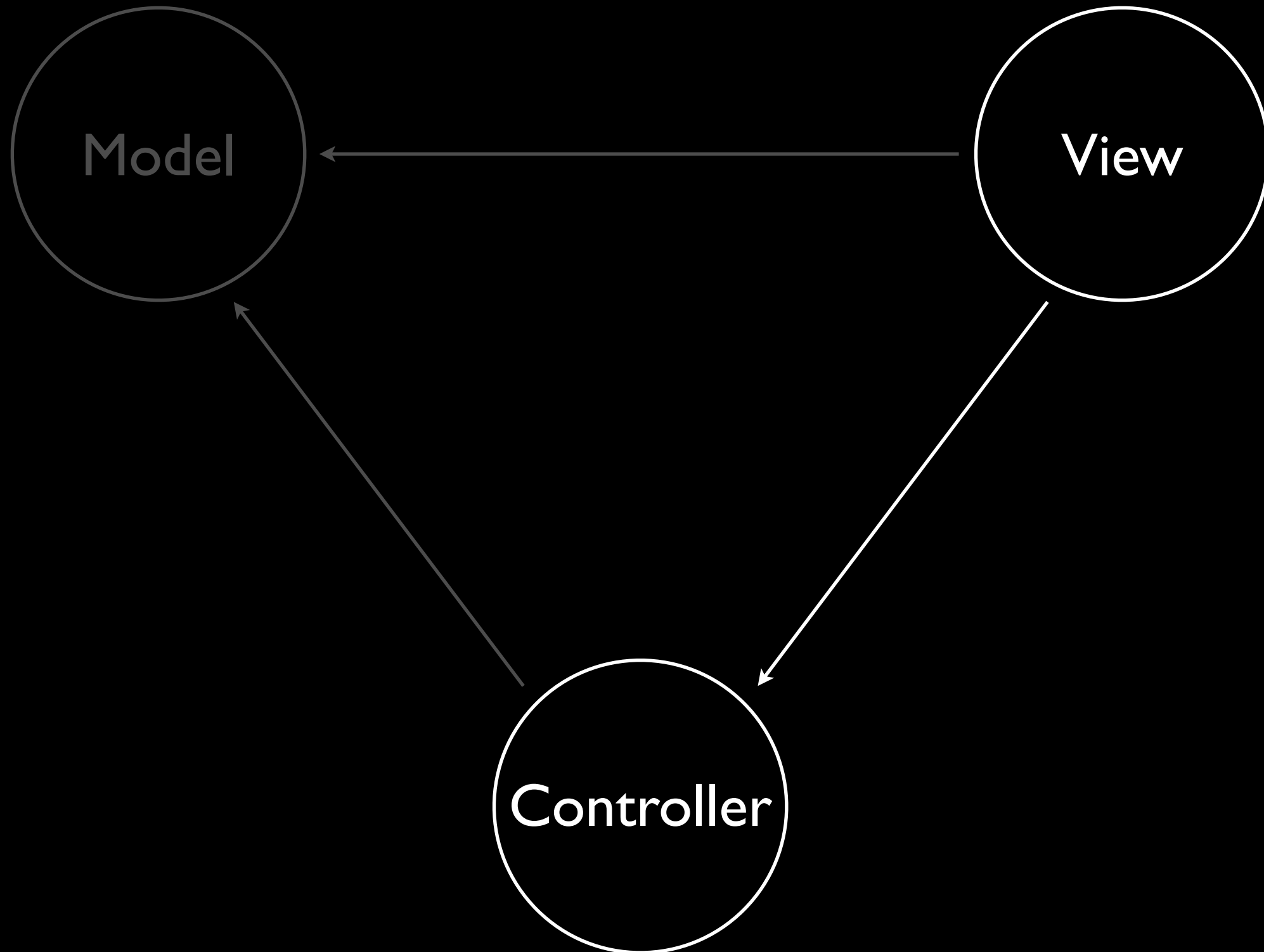
Mate

Swiz

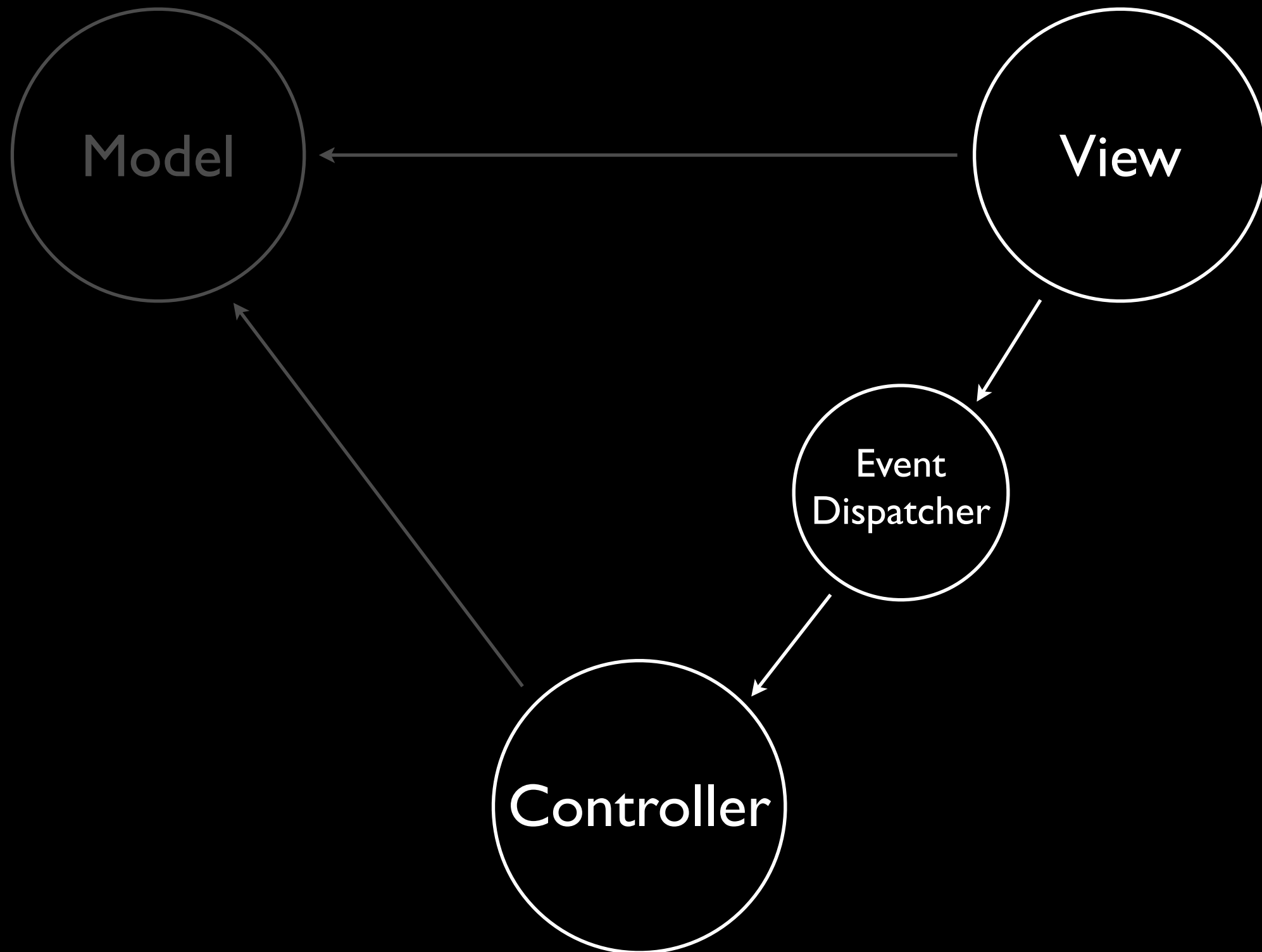
Model-View-Controller



View-Controller communication



Using an event dispatcher



Basic central event dispatcher

View

```
var loginEvent:LoginEvent = new LoginEvent(LoginEvent.LOGIN);  
loginEvent.username = username.text;  
loginEvent.password = password.text;  
globalDispatcher.dispatchEvent( loginEvent );
```

Controller

```
globalDispatcher.addEventListener( LoginEvent.LOGIN, loginHandler )
```

View-Controller in Cairngorm

View

extends CairngormEvent



```
var loginEvent:LoginEvent = new LoginEvent( LoginEvent.LOGIN );  
loginEvent.username = username.text;  
loginEvent.password = password.text;  
loginEvent.dispatchEvent();
```

Front Controller

```
addCommand( LoginEvent.LOGIN, LoginCommand );
```

View-Controller in Pure MVC

View

```
var loginData:LoginData = new LoginData();  
loginData.username = username.text;  
loginData.password = password.text;  
sendNotification( NotificationNames.LOGIN, loginData );
```



Notifications are
like Events

Controller

```
registerCommand( NotificationNames.LOGIN, LoginCommand );
```


View-Controller in Swiz

View

```
var loginEvent:LoginEvent = new LoginEvent(LoginEvent.LOGIN, true);  
loginEvent.username = username.text;  
loginEvent.password = password.text;  
dispatchEvent( loginEvent );
```



Bubbling
Event

Controller

```
[Mediate(event="LoginEvent.LOGIN")]  
public function loginHandler( event:LoginEvent ):void {  
    ...  
}
```

View-Controller in Mate

View

```
var loginEvent:LoginEvent = new LoginEvent(LoginEvent.LOGIN, true);  
loginEvent.username = username.text;  
loginEvent.password = password.text;  
dispatchEvent( loginEvent );
```

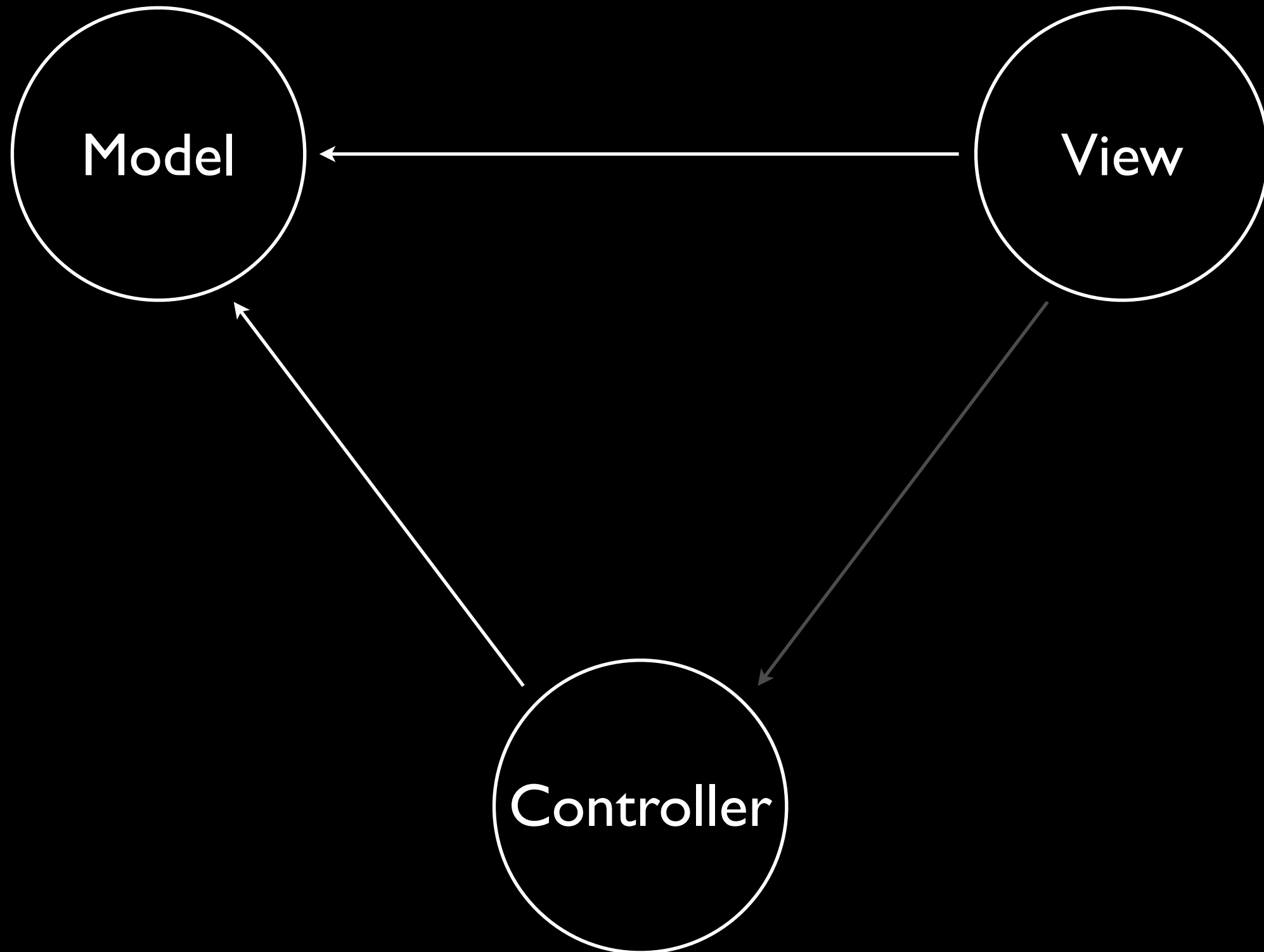


Bubbling
Event

Event Map

```
<EventHandlers type="{LoginEvent.LOGIN}">  
    ...  
</EventHandlers>
```

Accessing the Model



Accessing the model in Cairngorm

Model

```
class ModelLocator implements IModelLocator {  
    public static function getInstance():ModelLocator {  
        ...  
    }  
    [Bindable]  
    public var user:User;  
}
```



Singleton access

View / Controller

```
var user:User = ModelLocator.getInstance().user;
```

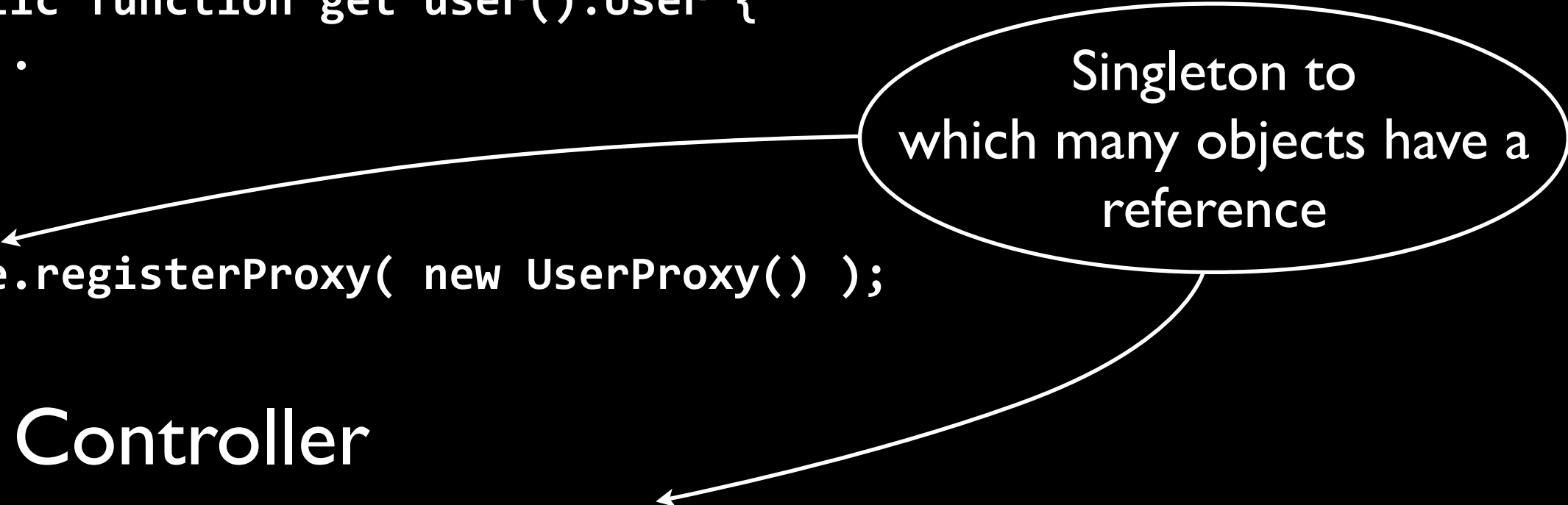
Accessing the model in Pure MVC

Model

```
class UserProxy extends Proxy implements IProxy {  
    public function get user():User {  
        ...  
    }  
}
```

```
facade.registerProxy( new UserProxy() );
```

Singleton to
which many objects have a
reference

A white oval callout box with a black border contains the text "Singleton to which many objects have a reference". Two arrows originate from this box: one points to the 'UserProxy' class definition in the code block above, and the other points to the 'retrieveProxy' method call in the code block below.

View / Controller

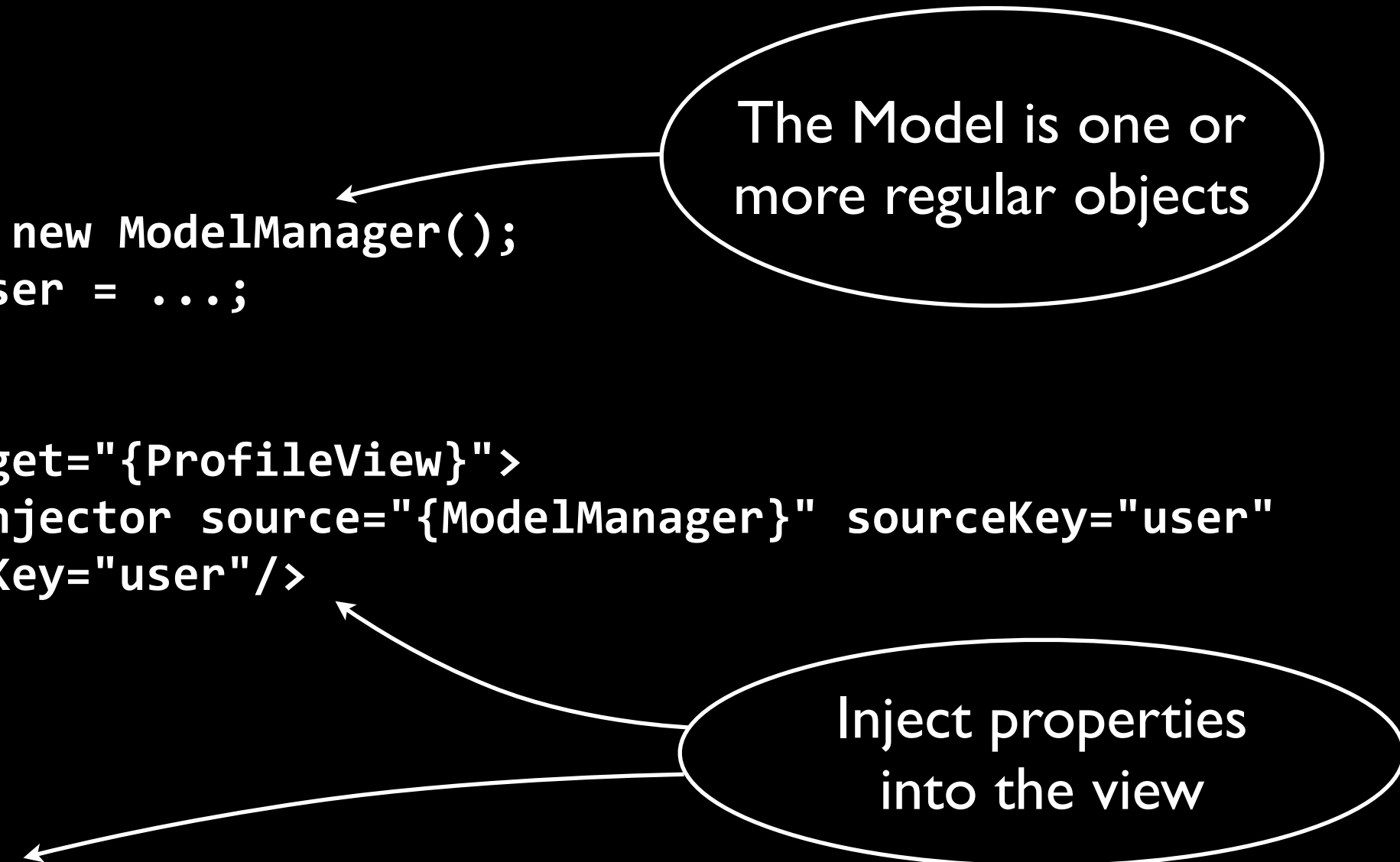
```
var userProxy:UserProxy = facade.retrieveProxy( "UserProxy" )  
                                     as UserProxy;  
var user:User = userProxy.user;
```

Accessing the model in Mate

Controller

```
modelManager = new ModelManager();  
modelManager.user = ...;
```

The Model is one or more regular objects



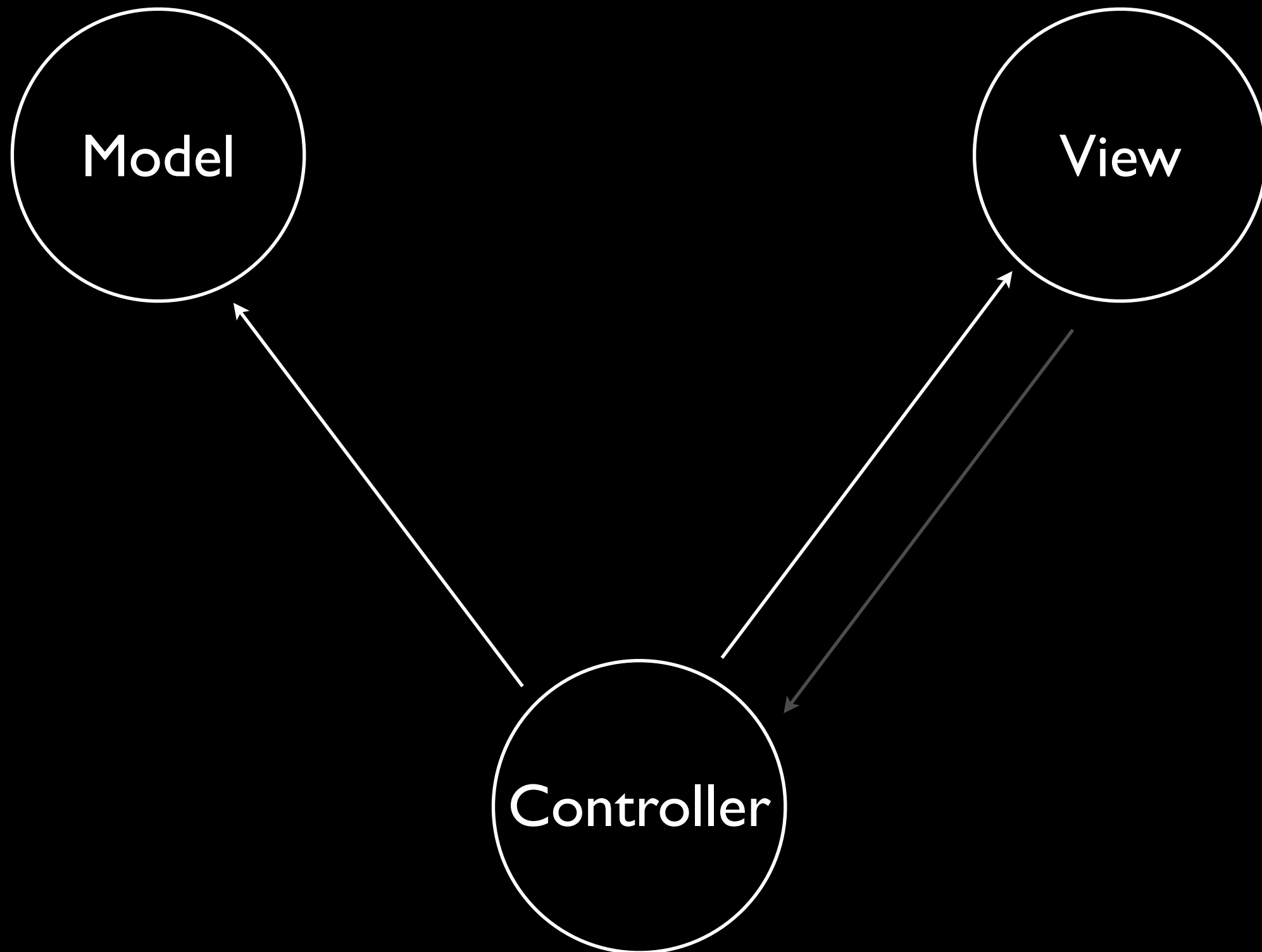
```
<Injectors target="{ProfileView}">  
  <PropertyInjector source="{ModelManager}" sourceKey="user"  
    targetKey="user"/>  
</Injectors>
```

View

```
public var user:User;
```

Inject properties into the view

Accessing the model in Mate




Accessing the model in Swiz

Model / BeanLoader

```
<BeanLoader>  
  <User id="user"/>  
</BeanLoader>
```

Model objects
are created in a
BeanLoader

A white arrow points from the text in the oval to the `<User id="user"/>` element in the XML code block above.

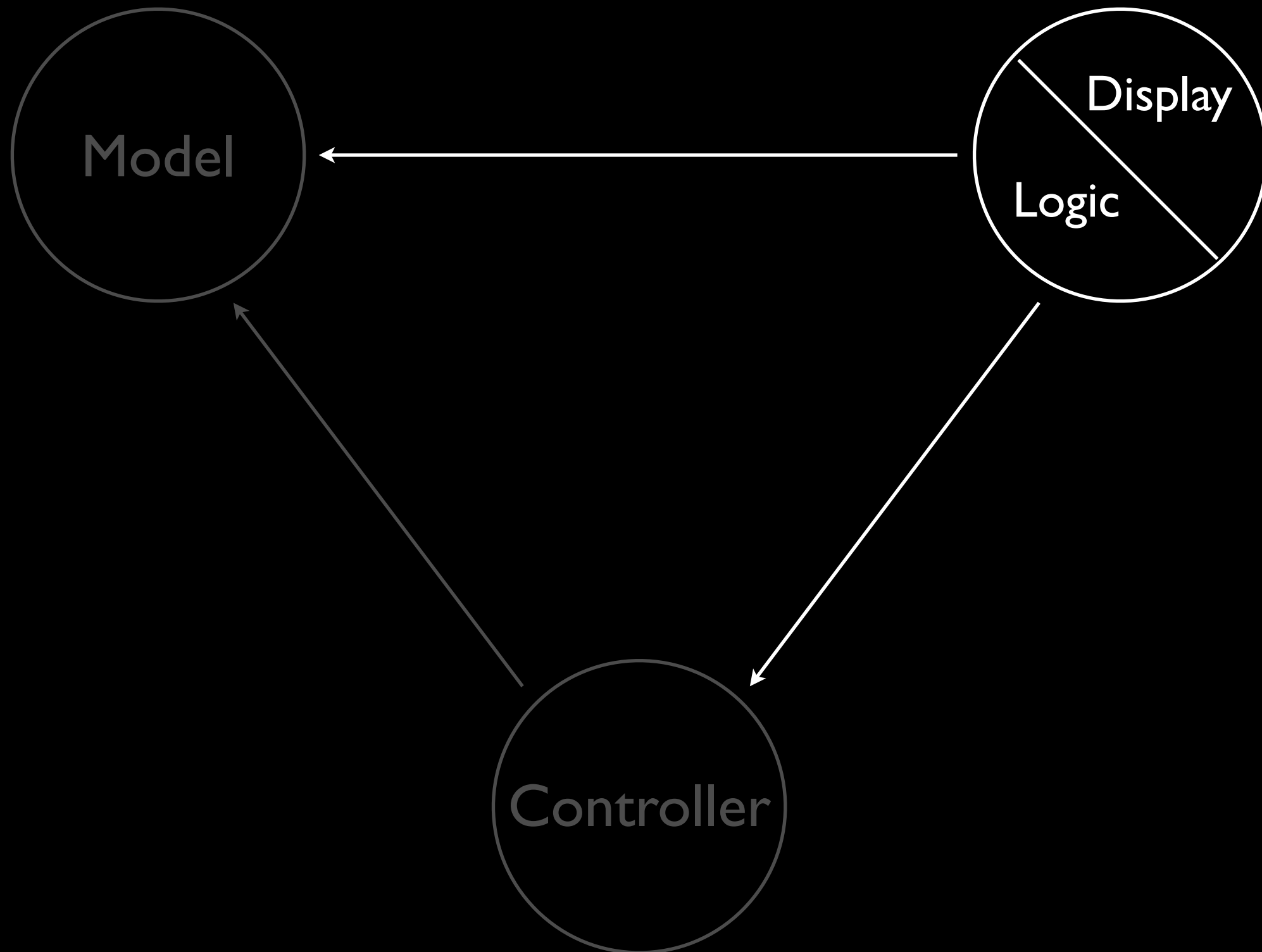
View / Controller

```
[Autowire]  
public var user:User;
```

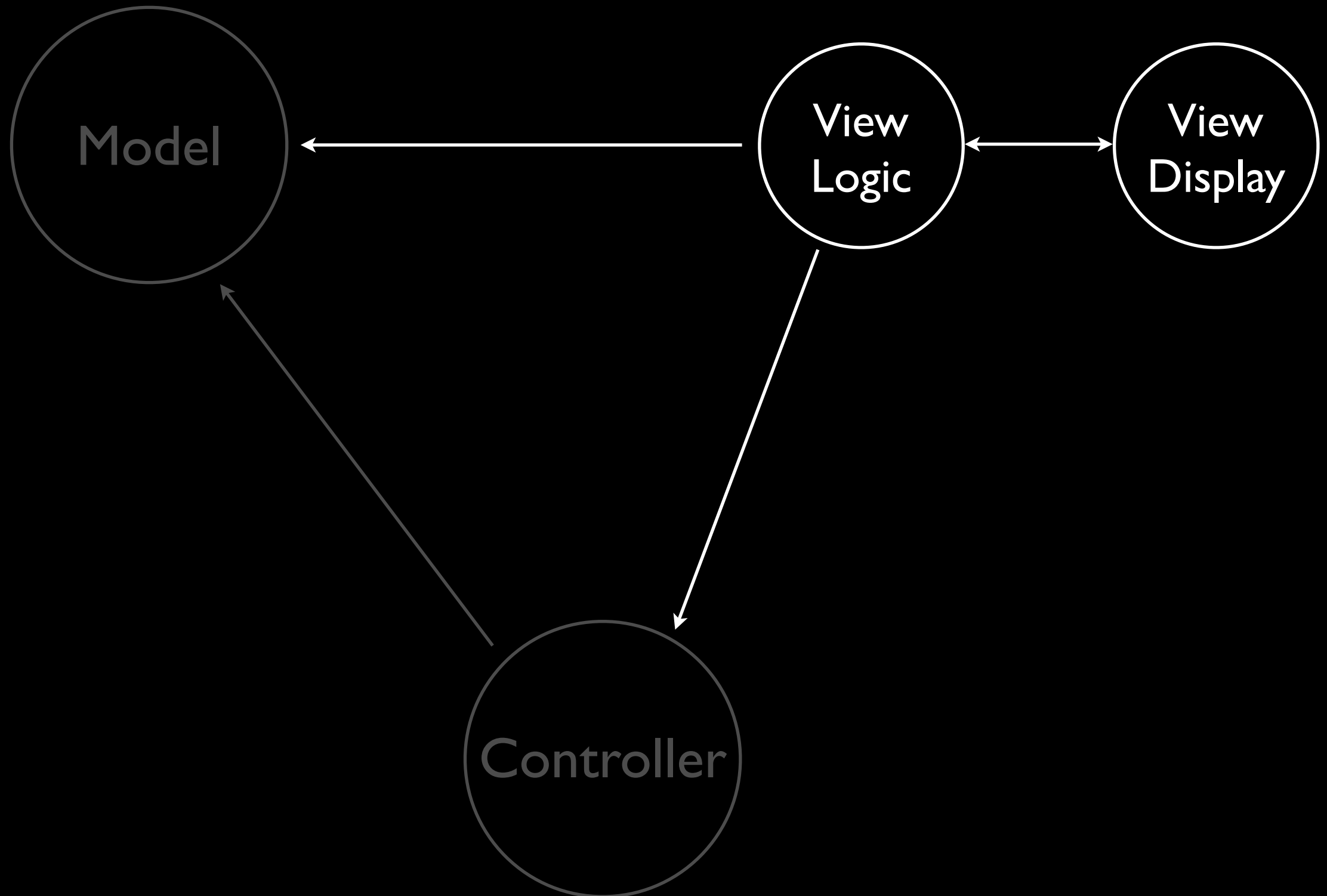
And injected into
other objects

A white arrow points from the text in the oval to the `[Autowire]` annotation in the code block above.

Separating view logic and display



Separating view logic and display



View logic solutions

Presentation Model

```
// Display's login button click handler calls logic method  
presentationModel.loginHandler( username.text, password.text );
```

Passive View

```
// Logic listens for click event  
passiveView.loginButton.addEventListener( MouseEvent.CLICK,  
                                           loginHandler );
```

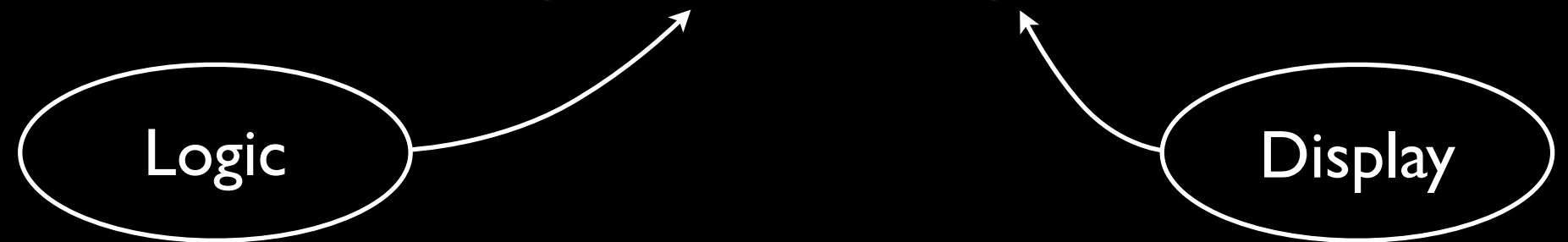
Supervising Controller

```
// Display's login button click handler  
dispatchEvent( new LoginEvent( LoginEvent.LOGIN,  
                                username.text, password.text ) );  
  
// Logic listens for login event  
display.addEventListener( LoginEvent.LOGIN, loginHandler );
```

View Logic in Frameworks

Pure MVC

```
facade.registerMediator( new LoginMediator( loginView ) );
```



Cairngorm, Mate, Swiz

X

Summary and Opinion

Cairngorm

Pure MVC

Mate

Swiz

Continue the discussion later

On my blog: www.bigroom.co.uk

On twitter: twitter.com/Richard_Lord