

London Flash Platform User Group, Oct 2009

# Application Frameworks

## The Good, the Bad & the Ugly

Richard Lord

Technical Architect

BrightTALK

[www.brighttalk.com](http://www.brighttalk.com)

# Application Frameworks

Cairngorm

PureMVC

Mate

Swiz

Parsley

RobotLegs

# Coming Up

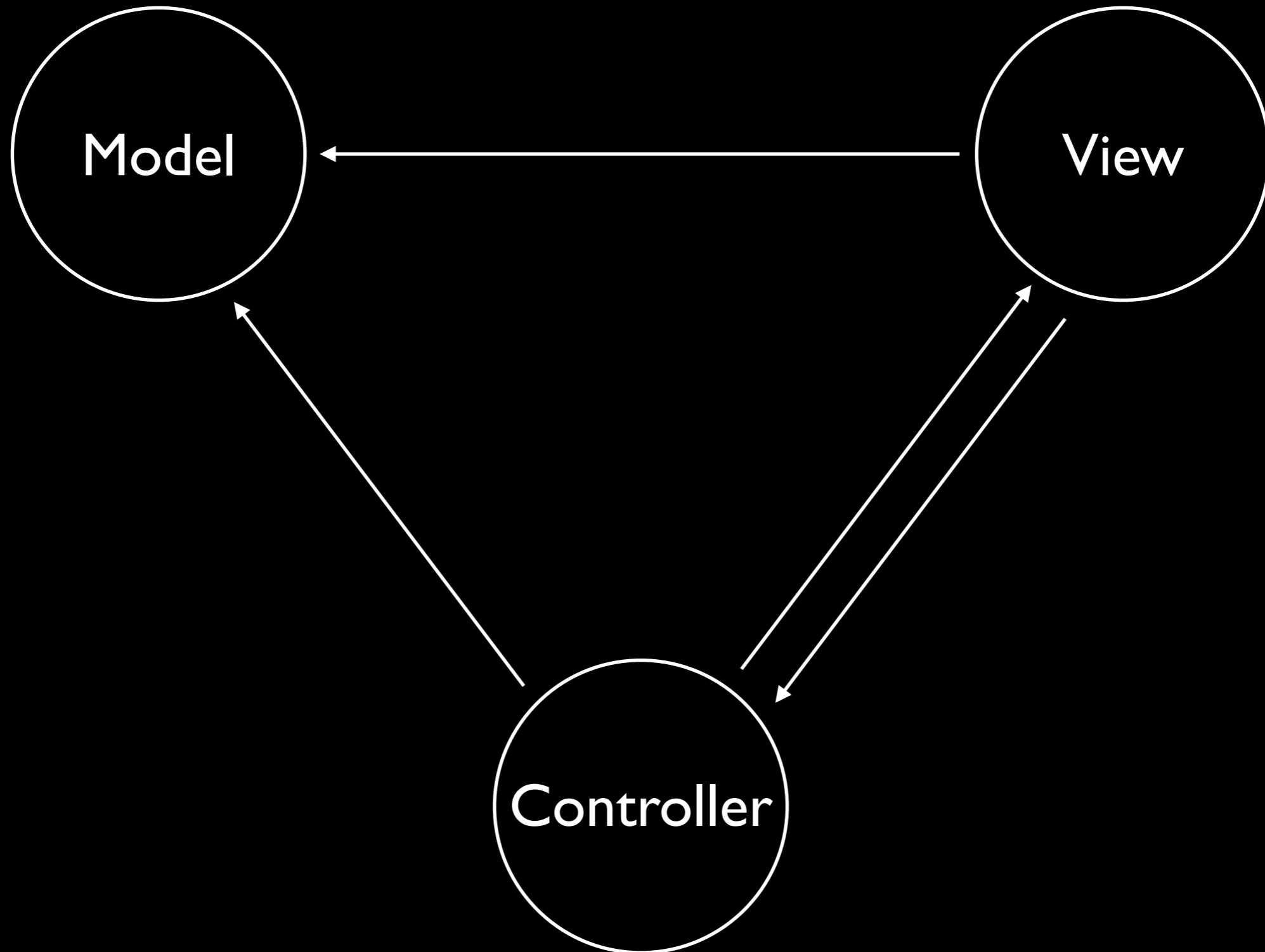
Model-View-Controller

Managing Dependencies

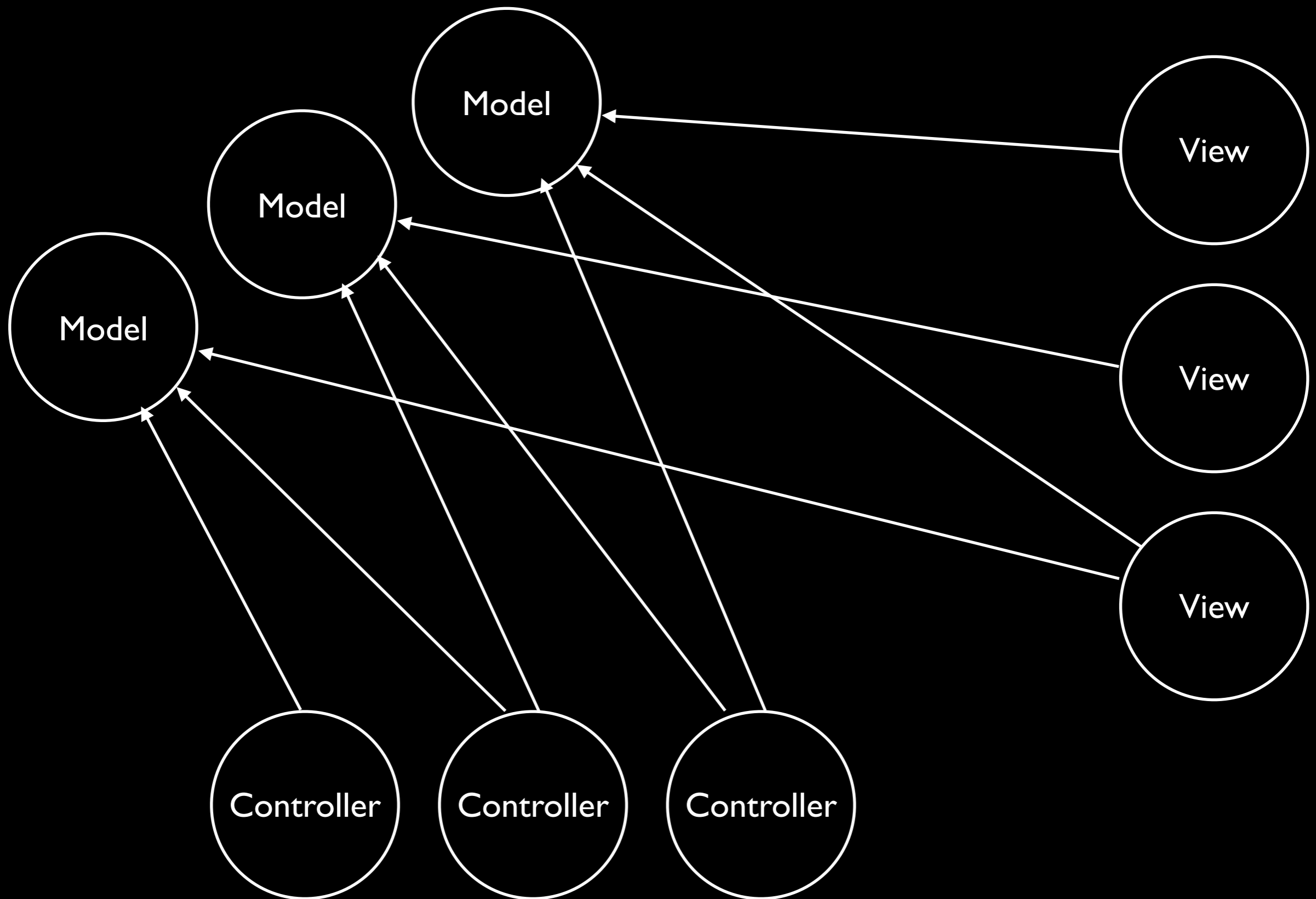
Managing Events

Some measured opinion

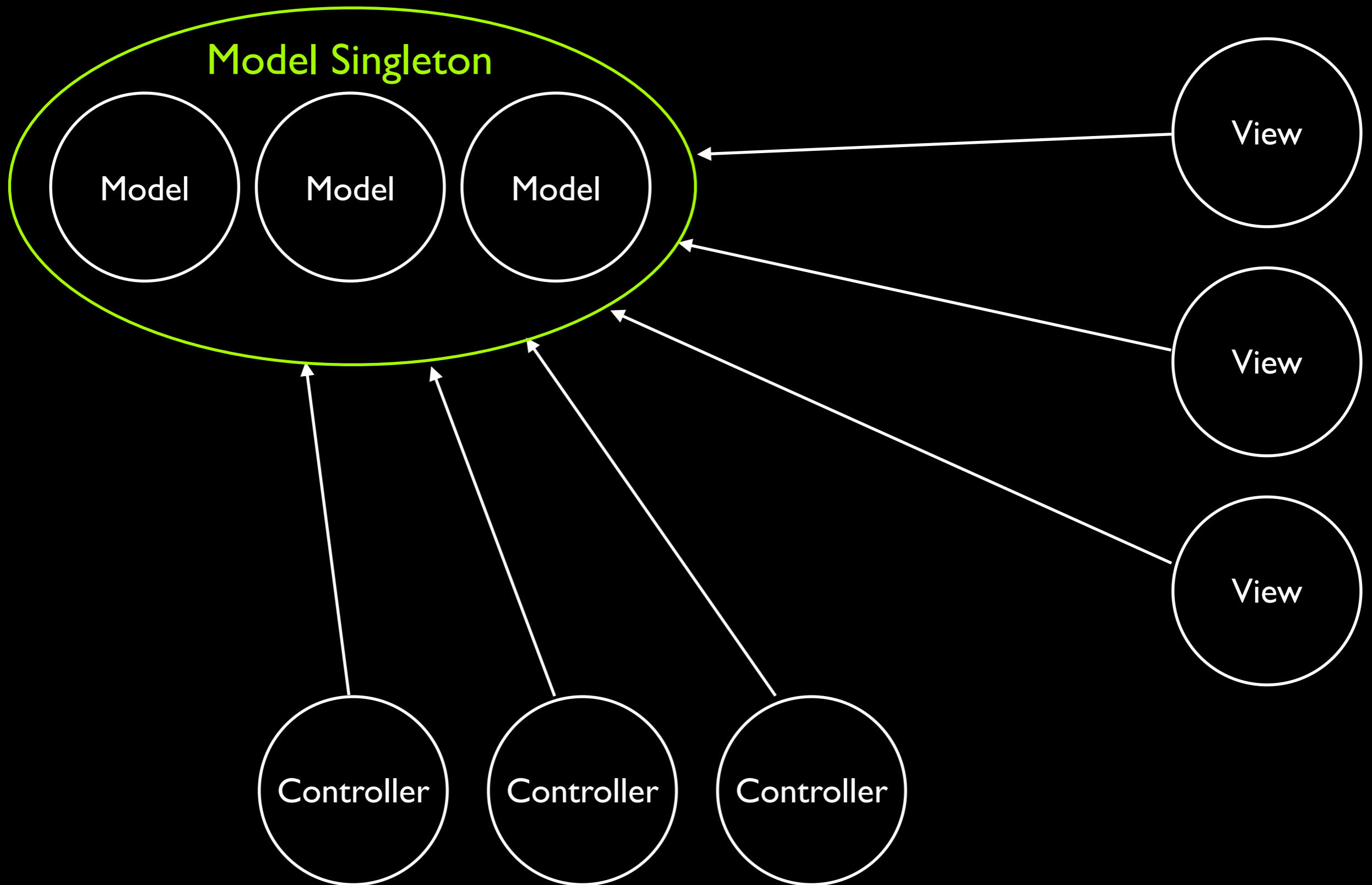
# Model-View-Controller



# Managing Dependencies



# Cairngorm uses global singletons



# Cairngorm uses a Singleton

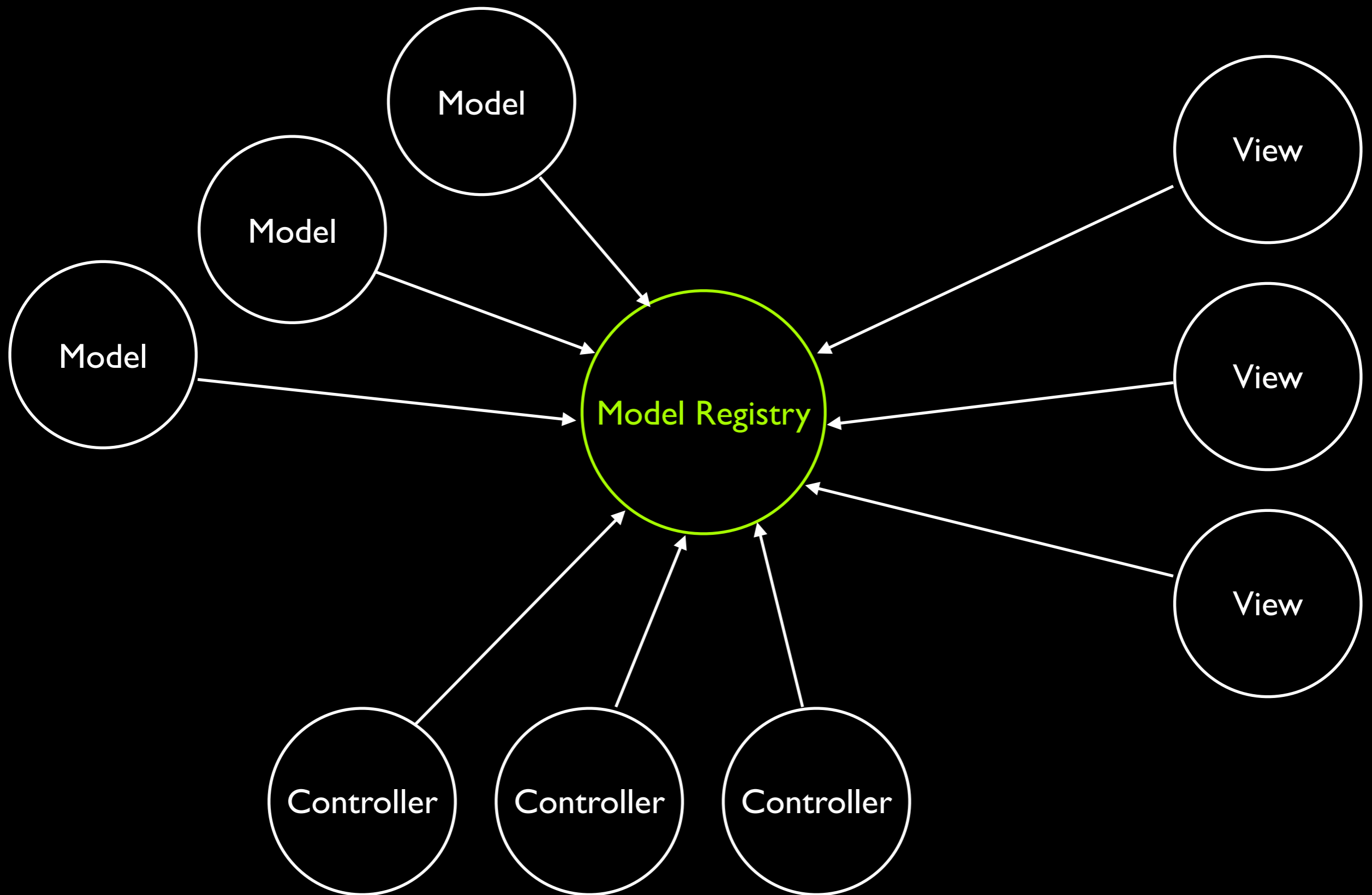
## Model is a Singleton

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

## View / Controller access that Singleton

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

# PureMVC uses a registry



# Accessing the model in Pure MVC

Model is a registry of proxies

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

Add object to registry



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

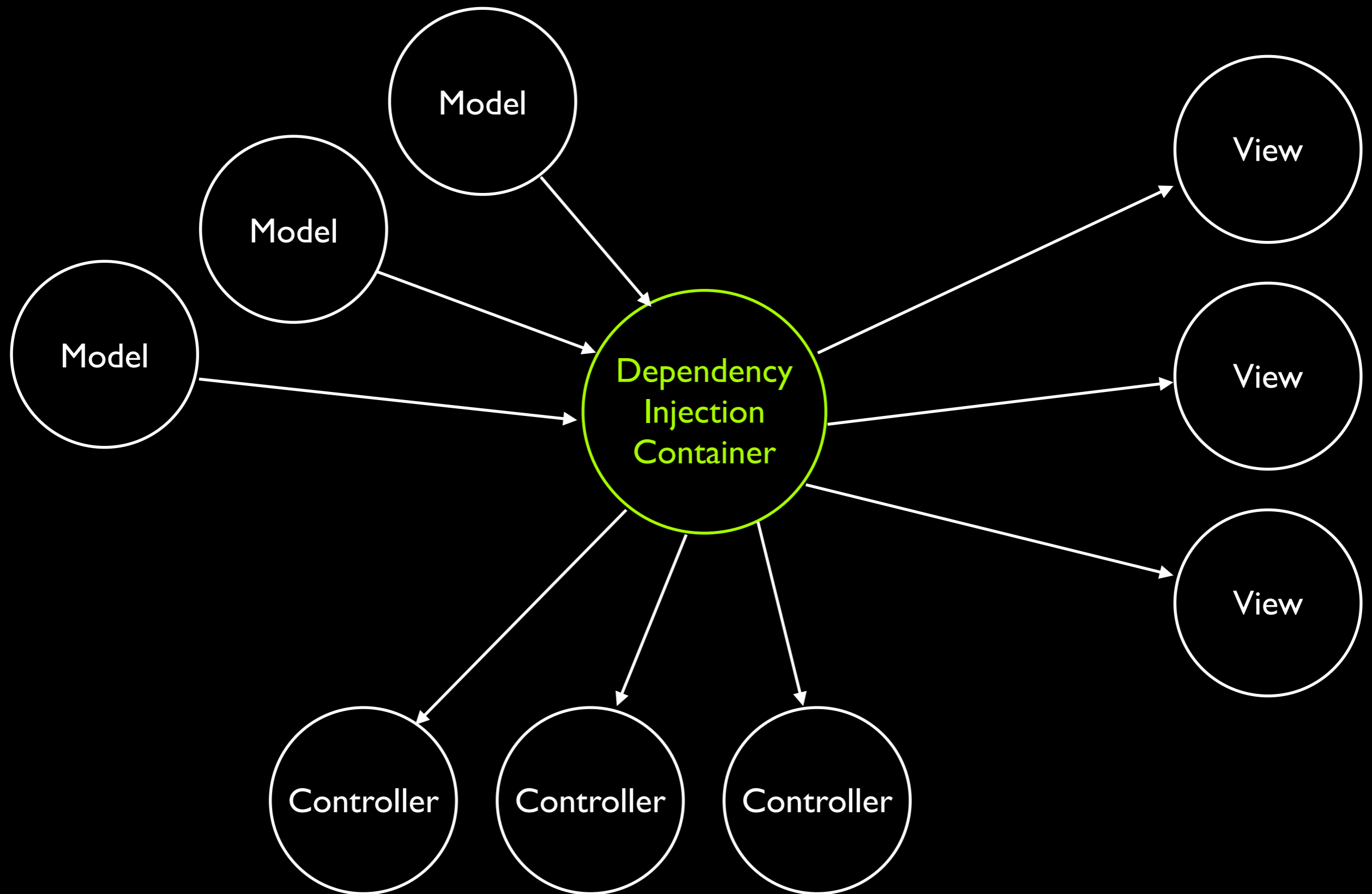
View / Controller fetch proxies from the registry

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

Fetch object from registry



# Others use dependency injection



# Accessing the model in Mate

Injection rules are defined in EventMaps

```
<mx:Script>  
  modelManager = new ModelManager();  
  modelManager.user = new User();  
</mx:Script>
```

The Model is one or more regular objects

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

Define injection rules

Properties are injected into the view

```
public class ProfileView {  
  public var user:User;  
  //...  
}
```


This will be set by the DI container

# Accessing the model in Swiz

Injection rules are defined in a BeanLoaders

```
<BeanLoader>  
  <User/>  
</BeanLoader>
```

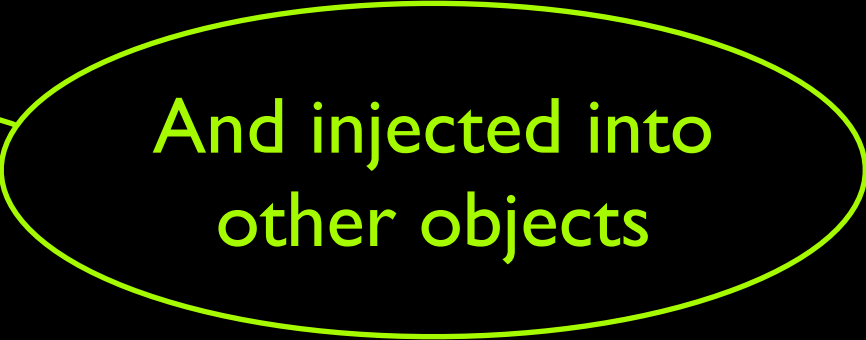
Regular object  
defined in MXML



Objects specify their requirements

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

And injected into  
other objects




# Accessing the model in RobotLegs

Injection rules are defined in a context

```
injector.mapSingleton( User );
```

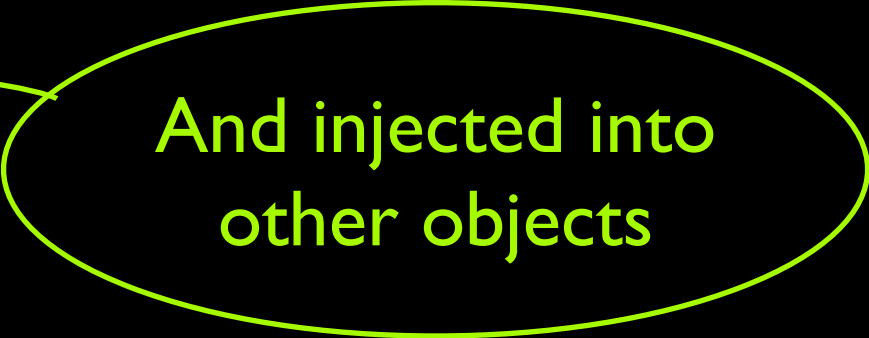
Injection rules are defined in the context



Objects specify their requirements

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

And injected into other objects

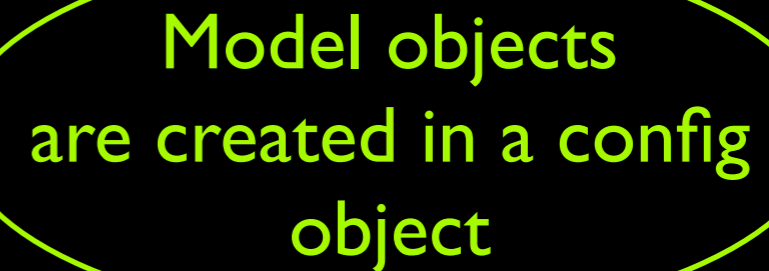


# Accessing the model in Parsley

Injection rules are defined in a configuration object

```
<mx:Object>  
  <User id="user"/>  
</mx:Object>
```

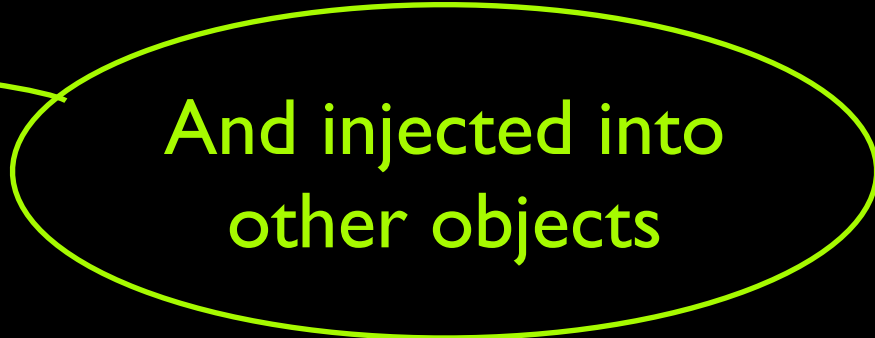
Model objects  
are created in a config  
object



Objects specify their requirements

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

And injected into  
other objects



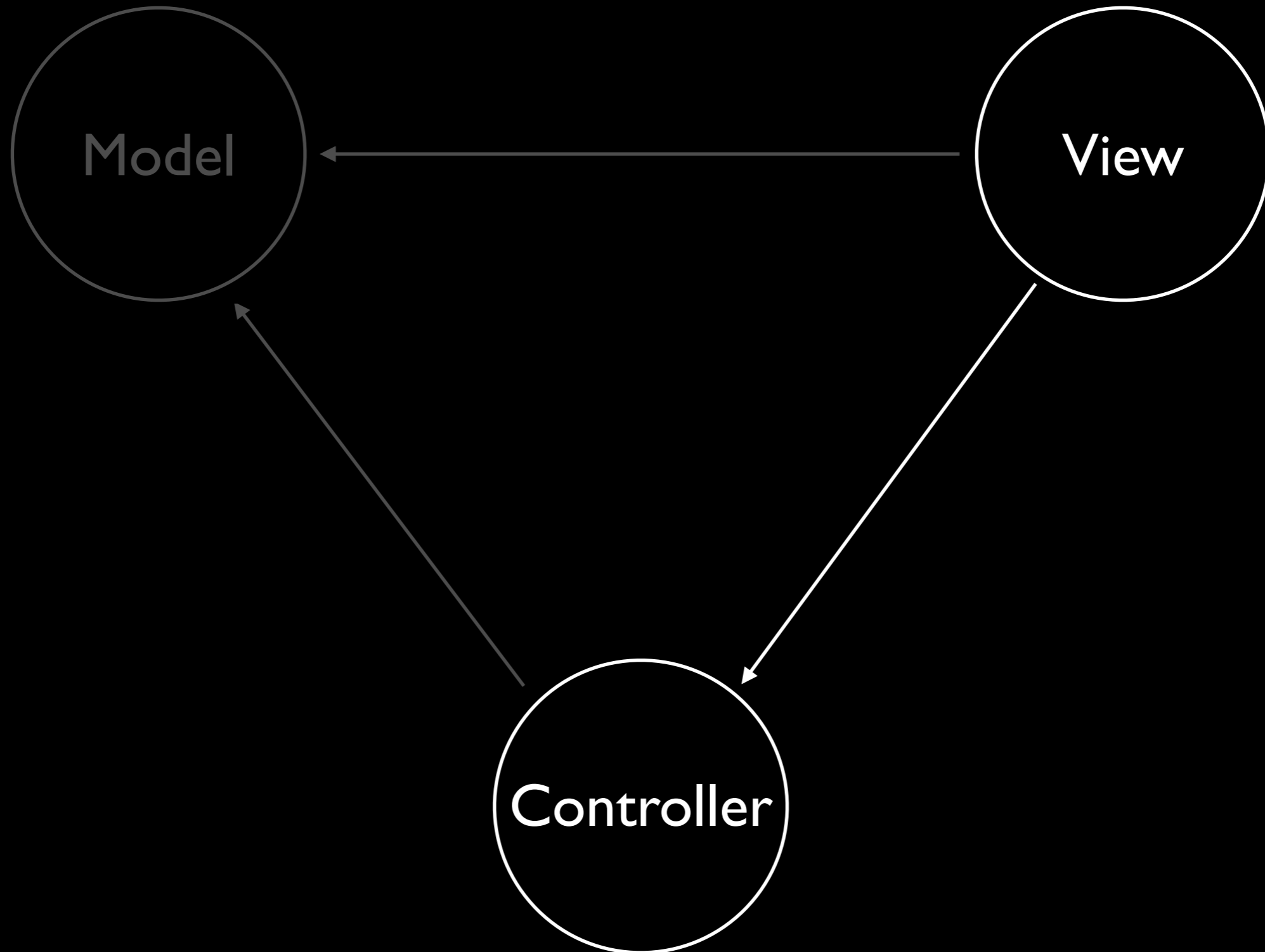
# Managing Dependencies

Singleton

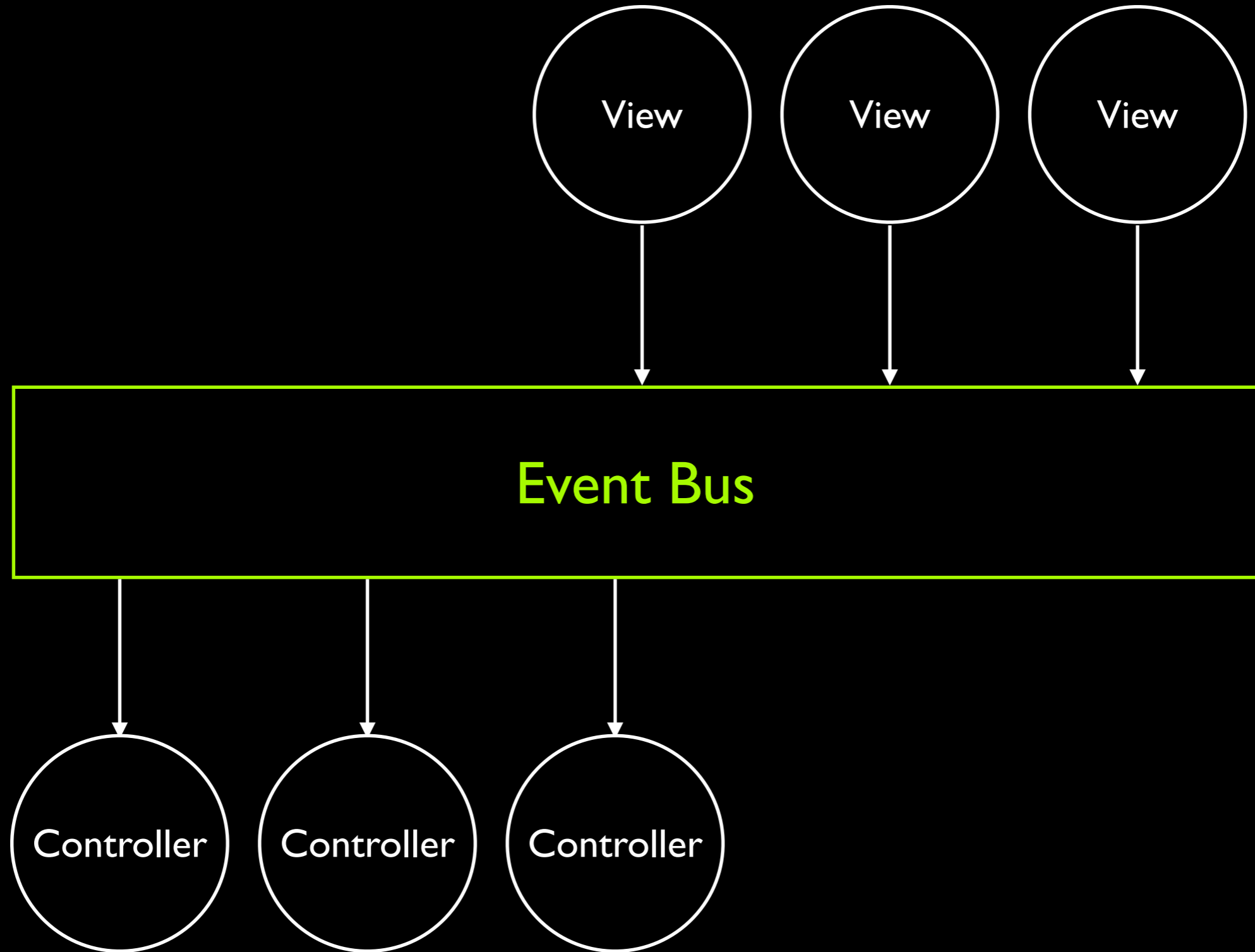
Registry

Dependency Injection

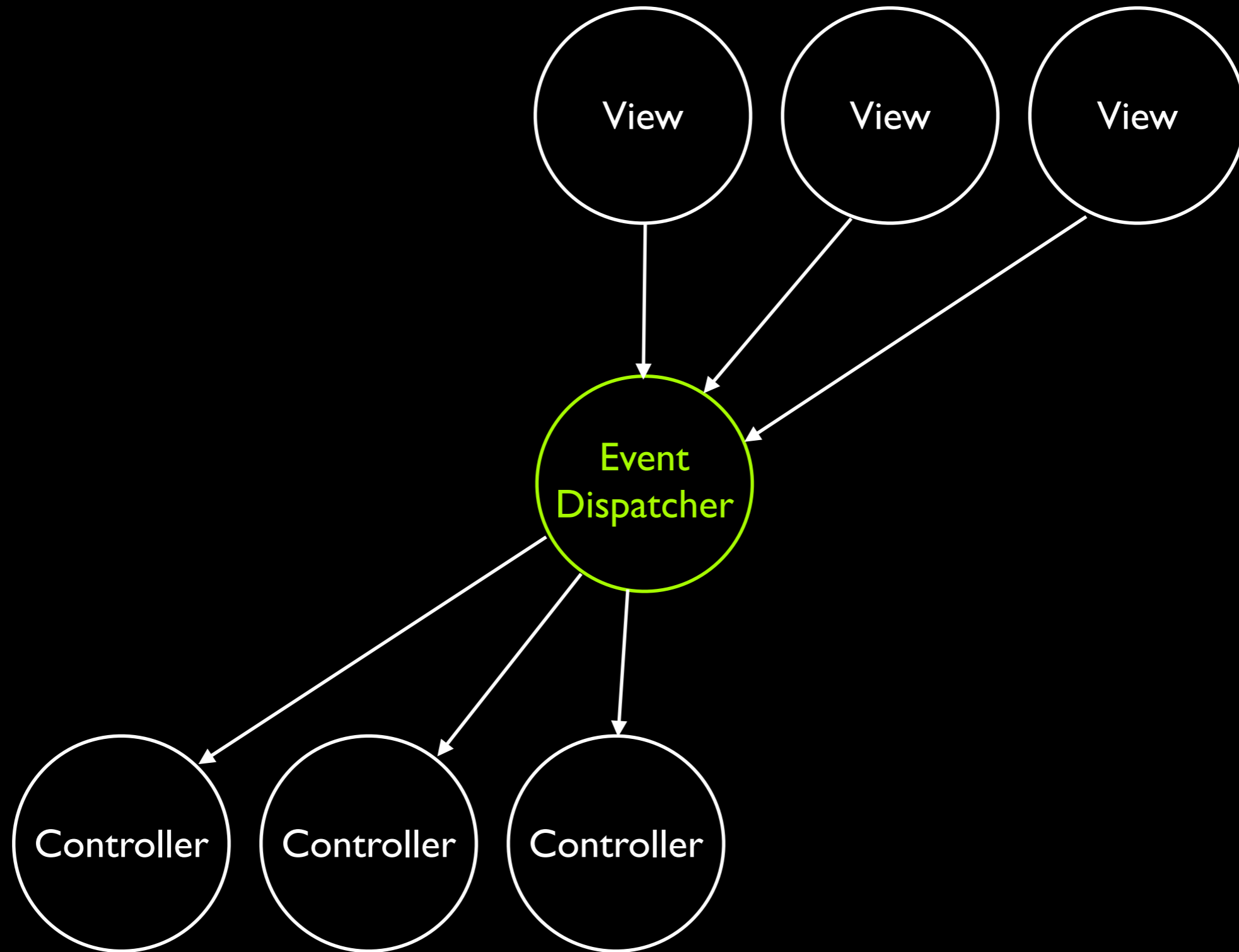
# Handling Events



# Event Bus



# Using a single event dispatcher



# Using a single event dispatcher

## View

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

## Controller

```
singleEventDispatcher.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 );
```

# Command pattern

The event handler



```
package
{
  public class LoginCommand extends Command
  {
    public function execute( event:Event ):void
    {
      var loginEvent:LoginEvent = event as LoginEvent;
      var loginService:LoginService = new LoginService();
      loginService.send( loginEvent.username, loginEvent.password );
      // ...
    }
  }
}
```

# 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

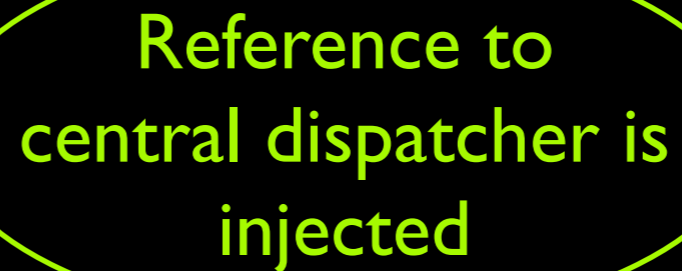
## Controller

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

# View-Controller in RobotLegs

## View

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



Reference to  
central dispatcher is  
injected

## Controller

```
commandMap.mapEvent( LoginCommand, LoginEvent.LOGIN );
```

# View-Controller in Parsley

## View

```
[Event(name="login", type="LoginEvent")]  
[ManagedEvents("login")]
```

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

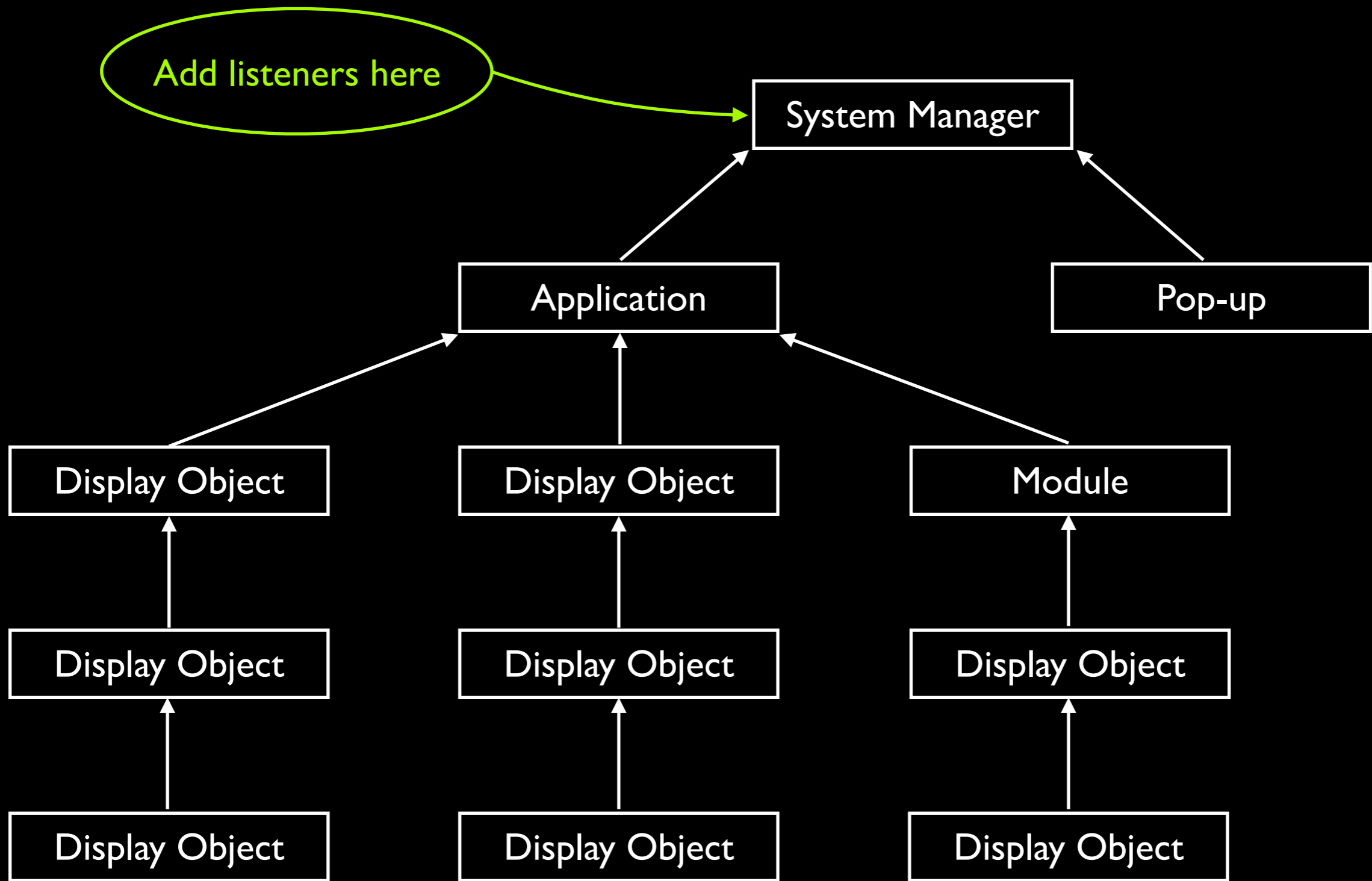
Central  
dispatcher should  
redispatch this event

## Controller

```
[ExceptionHandler(selector="login")]  
public function loginHandler( event:LoginEvent ):void {  
    ...  
}
```

The method  
handles this event

# Using the display list



# 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 );
```

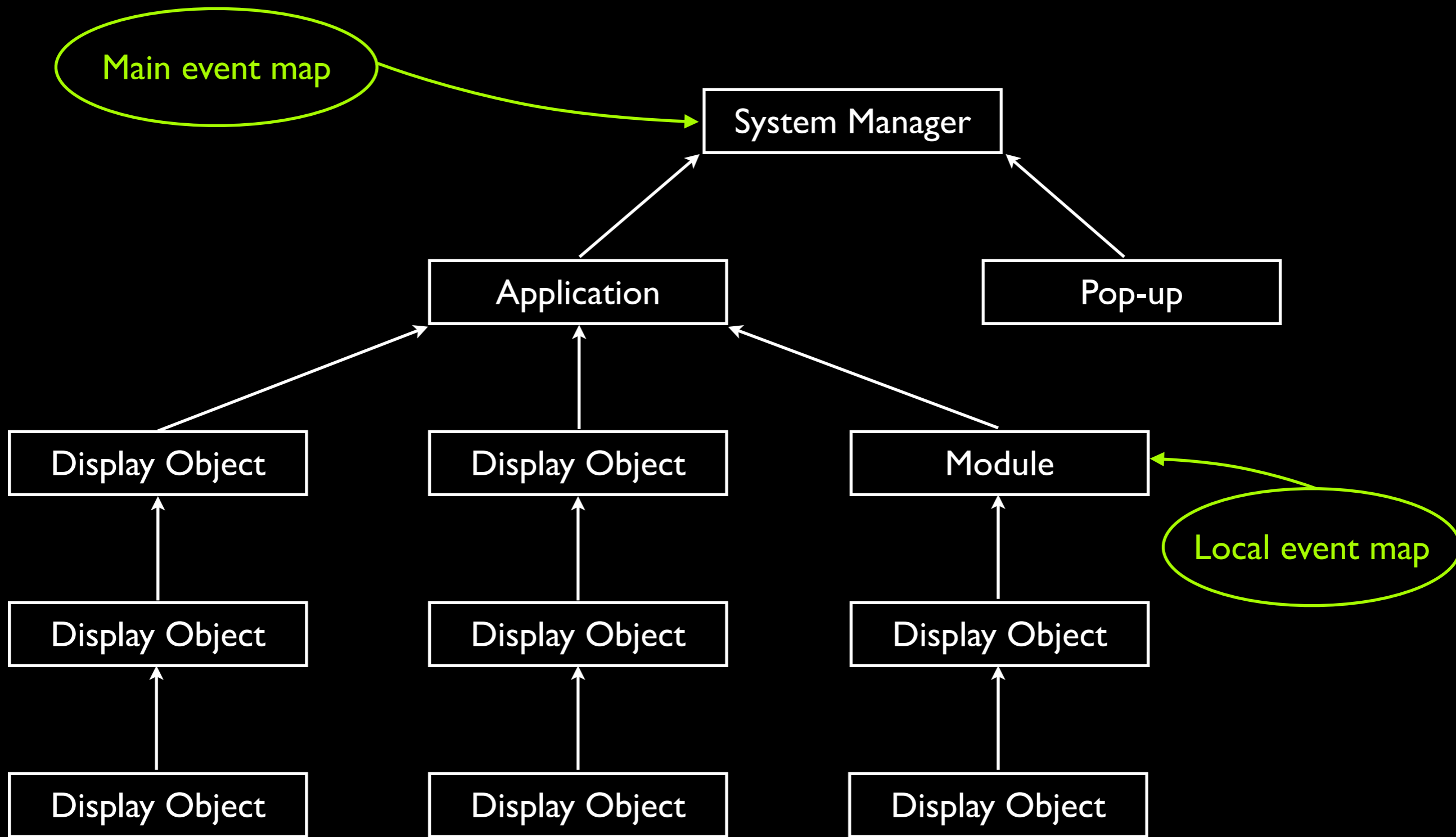


Bubbling  
Event

## Event Map

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

# Mate has local event maps



# Named Contexts

## PureMVC Multicore

```
Facade.getInstance( "local" );
```

## Parsley

```
[ManagedEvents("login", scope="local")]
```

```
[MessageHandler(selector="login", scope="local")]
```

## RobotLegs

```
public class TwitterClient extends Context...
```

# Event Bus

## Single Event Dispatcher

(may have multiple with access method)

## Display List

(may use localised event handling)

# Summary and Opinion

Cairngorm

PureMVC

Mate

Swiz

Parsley

RobotLegs

# Framework or Toolkit

## MVC Framework

Cairngorm

PureMVC

Mate

RobotLegs

## Toolkit

Swiz

Parsley

# Platforms

Flash & Flex

PureMVC

Parsley

RobotLegs

Flex only

Cairngorm

Mate

Swiz

# Modular features

Yes

No

PureMVC (multicore)

Cairngorm

Mate

PureMVC (standard)

Parsley

Swiz

RobotLegs

# Project developers

## Corporate

Cairngorm

Mate

Parsley

## Community

PureMVC

Swiz

RobotLegs

# History and evolution

Generation 1.0

Cairngorm

Out of date

Generation 1.5

PureMVC

Mate

Mature

Swiz

Generation 2.0

Parsley

Cutting edge

RobotLegs

This is me

[www.bigroom.co.uk](http://www.bigroom.co.uk)

[twitter.com/Richard\\_Lord](https://twitter.com/Richard_Lord)