WireBox

Next Generation DI/AOP

Who am I?

- Luis Majano - Computer Engineer
- Born in San Salvador, El Salvador -->
- President of Ortus Solutions
- Manager of the IECFUG (www.iecfug.com)

- Creator of ColdBox, MockBox, LogBox, CacheBox, WireBox, CodexWiki, or anything Box!
- Documentation Freak!

Ortus Solutions
A Software Revolution

Professional Open Source

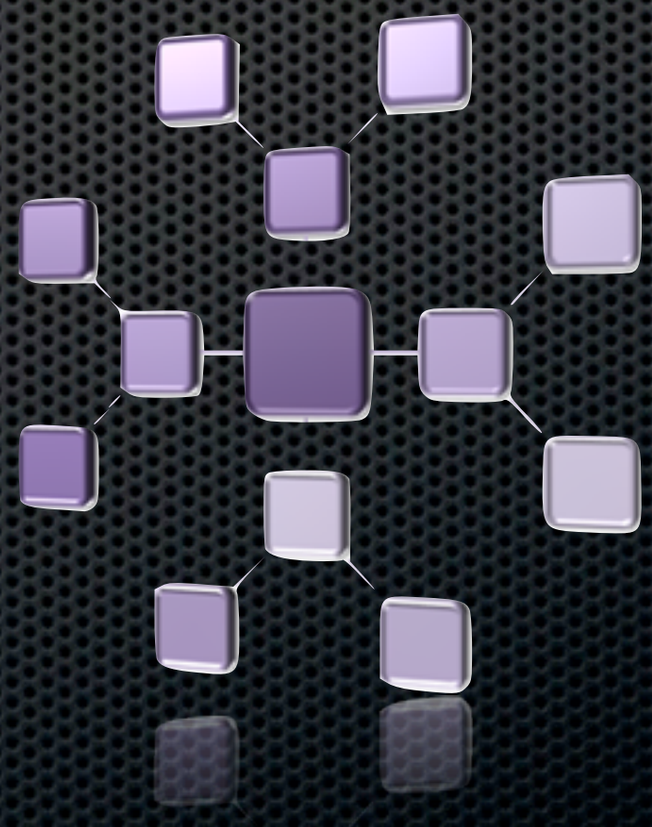
- **ColdBox Platform** is POSS
- Professional Training & Books
- Support & Mentoring Plans
- Architecture & Design Sessions
- Server Tuning & Setup
- Code Reviews & Sanity Checks
- Consulting
- We can even brew coffee for you!

Ortus Solutions
A Software Revolution

www.ortussolutions.com
consulting@ortussolutions.com

What is WireBox?

"A next generation conventions based dependency injection and AOP framework for ColdFusion"



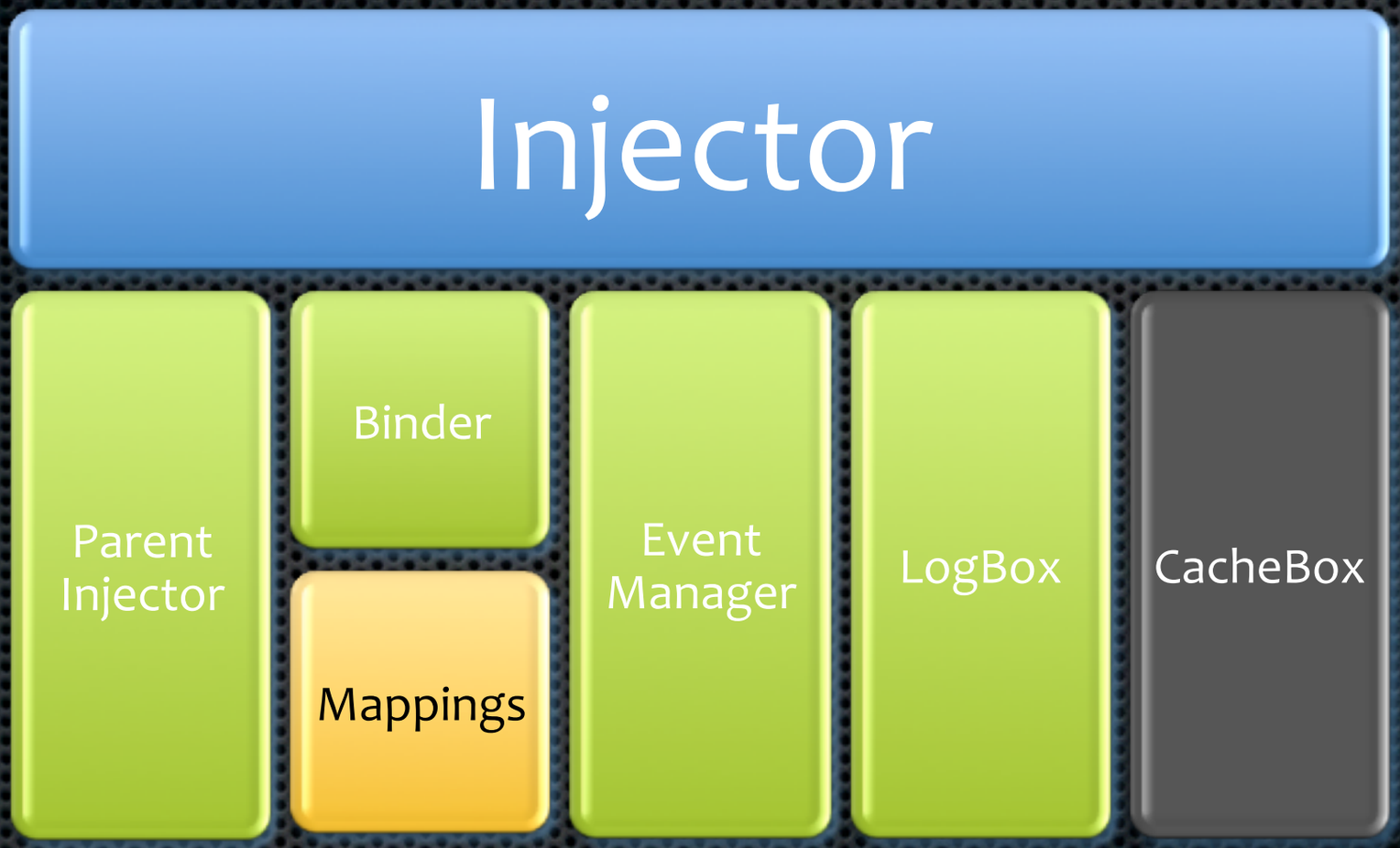
Requirements

- ColdFusion 8 and above
- Railo 3.1 and above
- Usage of various CF scopes

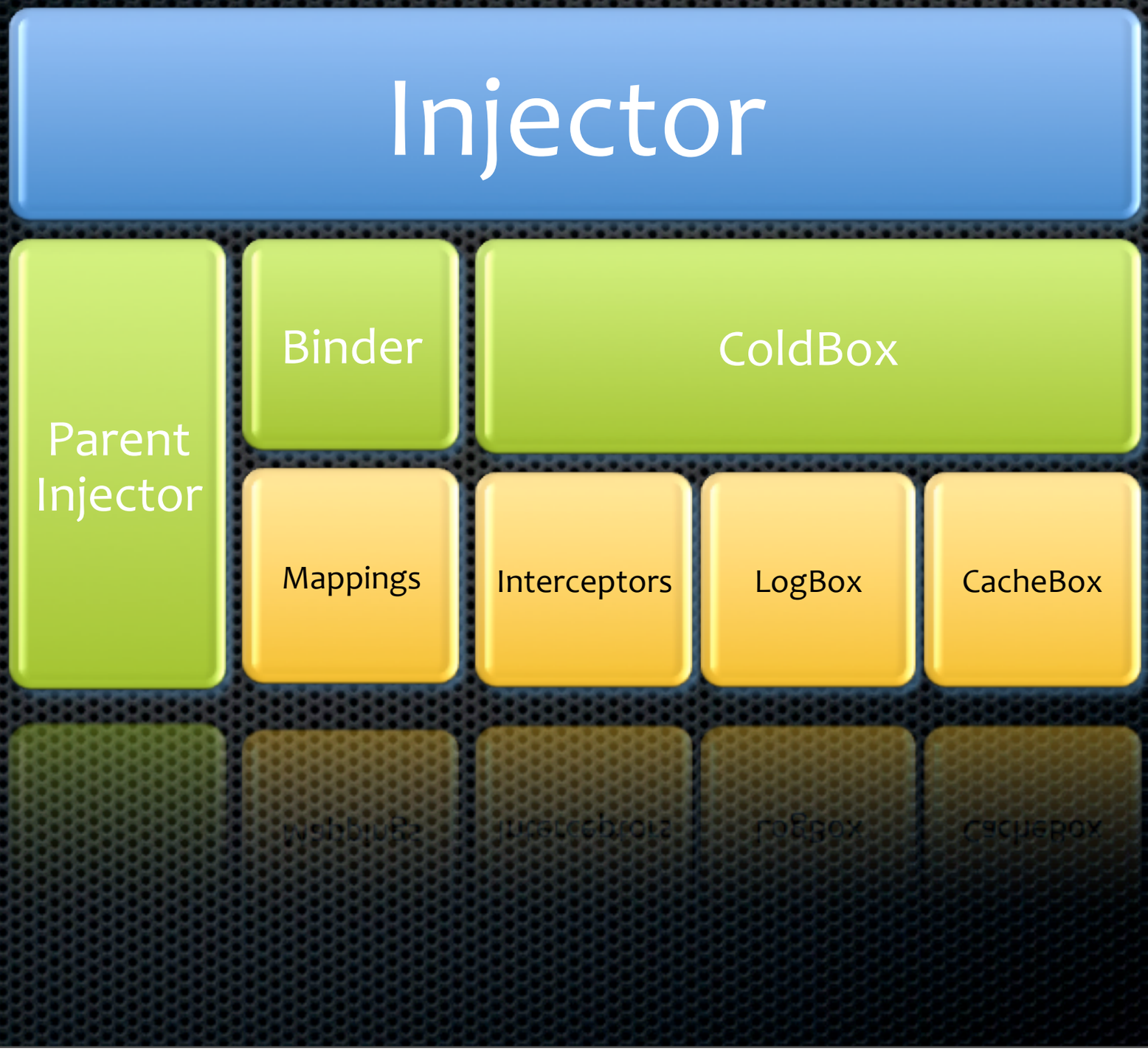
WireBox has 2 flavors?

cherry and vanilla?

WireBox Standalone



WireBox ColdBox Context



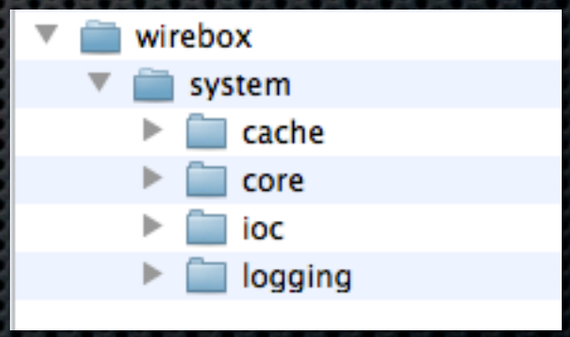
Installation

1

- Download ColdBox or Standalone
  - http://coldbox.org/download

2

- If standalone, drop in the webroot as **wirebox**
- or create a **wirebox** mapping



3 Relax and start coding!

DI Basics

- Replace manual/boilerplate object creation and/or object factories
- No more **createObject**() or **new**() to avoid coupling
- Abstract object dependencies
- Dependencies = Objects, Settings, Data, Etc.
- Manage persistence for you
- Makes objects more testable and mockable

```
component{
    function init(){
        variables.spellChecker = new SpellChecker();
    }
}
```

DI Basics

- Different types of dependency injection idioms
  - Constructor Arguments
  - Setters/Methods
  - Mixins of variables
- **Autowiring** = Discovery of DI metadata and wiring of dependencies into a target object
- Ability to do AOP and OO funkiness

DI Basics

- Instead of pulling your dependencies in, you opt to receive them from someplace and you don't care where they come from.
- **Hollywood principle**

Don't call us; we'll call you!

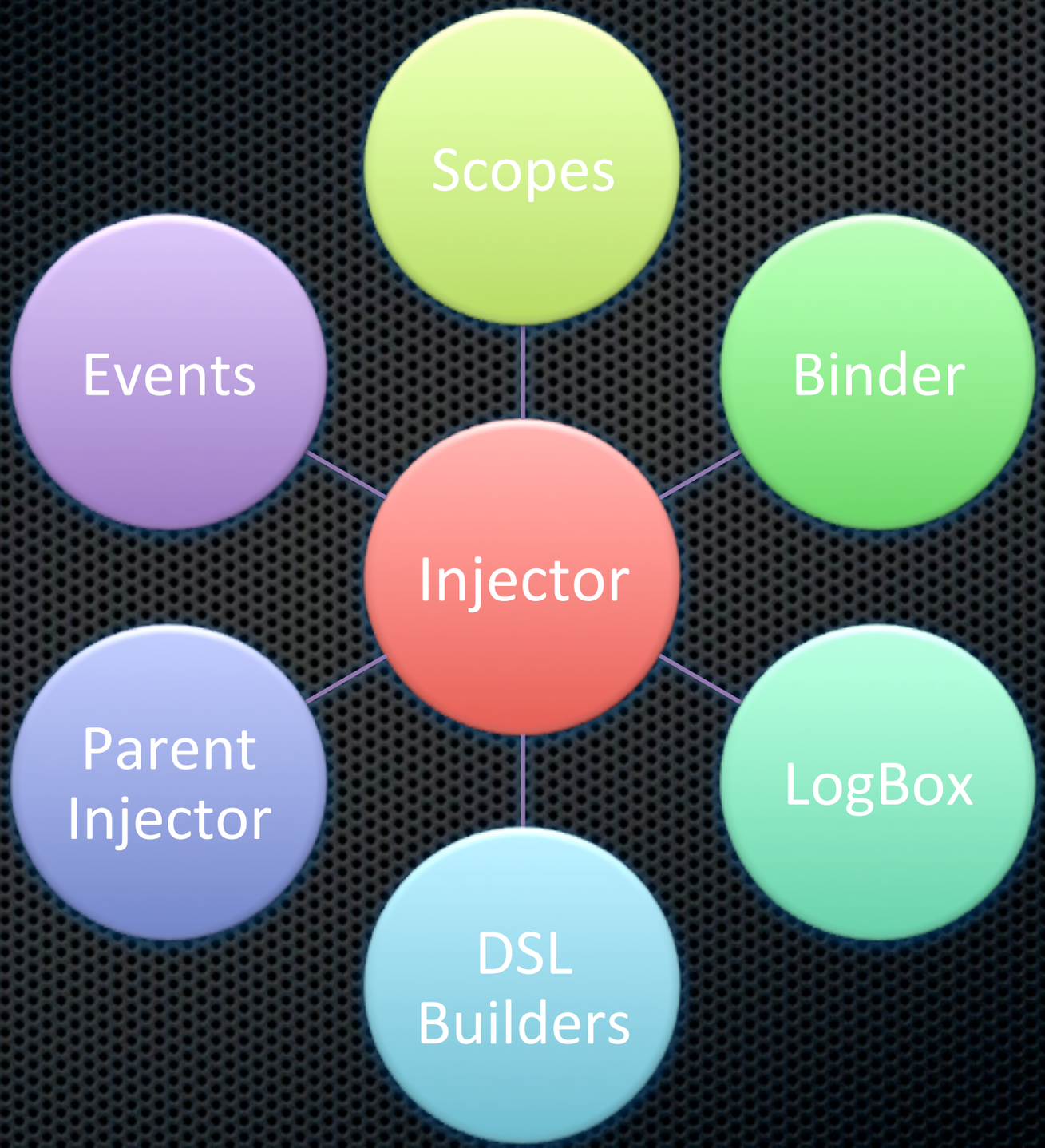
Why WireBox?

Because it is beautiful?

Features

- Documentation & Professional Services
- Annotation driven DI
- 0 configuration or programmatic configuration mode (NO XML)
- Creation & DI of : CFCs, java, webservices, rss, constants, etc
- Multiple Injection Styles: setter, method, constructor, mixins
- Persistence scopes: singleton, session, request, cache, etc.
- Integrated logging and debugging via LogBox
- Scoped object providers
- Object Life Cycle Events
- Automatic CF Scope registration

WireBox Universe



WireBox Injector

- WireBox Main Class
- Creates and wires all objects for you
- Announces events
- Can use an optional configuration binder
- Arguments
- configuration binder
- properties structure

```
injector = createObject("component","wirebox.system.ioc.Injector").init();

injector = createObject("component","wirebox.system.ioc.Injector").init
(binder="path.to.Binder",properties={prop1=val1,porp2=val2});
```

Configuration Binder

- A CFC simple or inherits from: **coldbox.system.ioc.config.Binder**
- Implements one method:
  - **Configure(binder)** : No inheritance
  - **Configure()** : With inheritance (Our Preference)
- Define WireBox Settings
- Define Object Mappings via our programmatic mapping DSL

Mapping DSL

- A programmatic DSL to define object mappings and relationships
- Used by concatenating calls to itself, returns the binder always
- Very readable bursts of logic:
  - ***map(“Luis”).toJava(“cool.Java.App”).into(this.SCOPES.Singleton);***
- Extend it!
  - Create your own methods in your binder, that’s it!

```
component extends=”wirebox.system.ioc.config.Binder”{

     configure(){

        map(“Luis”).toJava(“cool.java.Service”).asSingleton().asEagerInit();

     }

}
```

```
map("MyService")
     .to("model.MyService")
     .onDICOmplete(["start","processRules"])
mapPath("model.MyService");

// Recursively map all objects in the model folder
mapDirectory('model');

// Eager initialized objects
map("luis,joe").to("model.Path").in(this.SCOPES.SINGLETON).asEagerInit()

// map to Custom DSL
map("bob").toDSL("bob:build")

// using initWith() for passing name-value pairs or positional arguments for direct initialization of a
mapping
map("transferConfig")
     .to("transfer.com.config.Configuration")
     .initWith(datasourcePath=getProperty('datasourcePath'),
               configPath=getProperty('configPath'),
               definitionPath=getProperty('definitionPath'));

// Now doing with setter injection
map("transferConfig")
     .to("transfer.com.config.Configuration")
     .setter(name="datasourcePath", value=getProperty("datasourcePath"))
     .setter(name="configPath", dsl="property:configPath")
     .setter(name="definitionPath", value=getProperty("definitionPath") );

// Map with constructor arguments
map("transfer")
     .to("transfer.com.Transfer")
     .in(SCOPES.SINGLETON)
     .noAutowire()
     .asEagerInit()
     .initArg(name="configuration",ref='transferConfig');  //ref = name by default, or have an explicit name
```

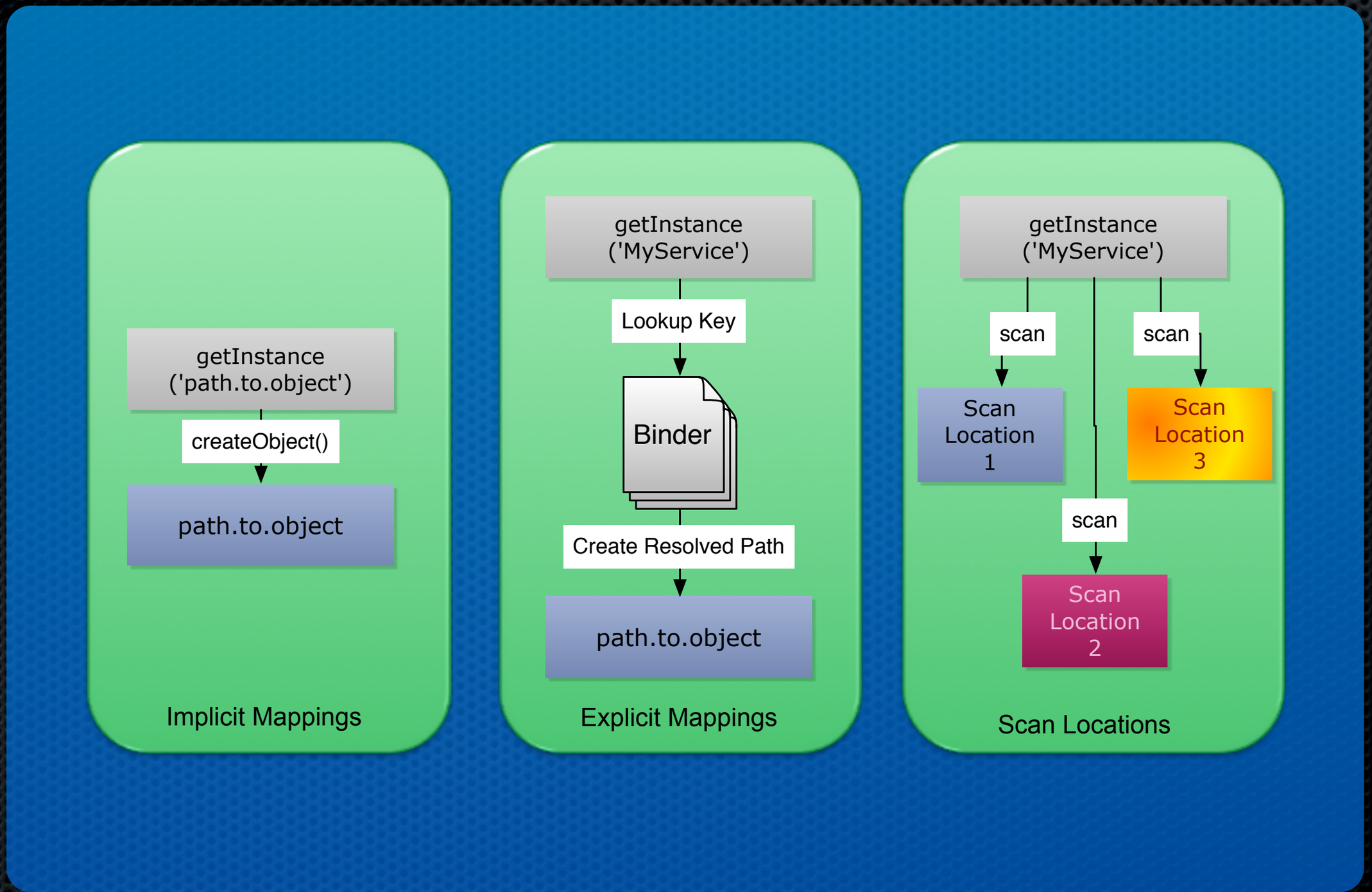
```
// RSS Integration With Caching.
map("googleNews")
    .toRSS("http://news.google.com/news?pz=1&ned=us&hl=en&topic=h&num=3&output=rss")
    .asEagerInit()
    .inCacheBox(timeout=20,lastAccessTimeout=30,provider="default",key="google-news");

// Java Integration with init arguments
map("Buffer").
    toJava("java.lang.StringBuffer").
    initArg(value="500",javaCast="long");

// Java integration with initWith() custom arguments and your own casting.
map("Buffer").
    toJava("java.lang.StringBuffer").
    initWith( javaCast("long",500) );

// Property injections
map("SecurityService")
    .to("model.security.SecurityService")
    .in(this.SCOPES.SERVER)
    .property(name="userService", ref="UserService", scope="instance")
    .property(name="logger", dsl="LogBox:root", scope="instance")
    .property(name="cache", dsl="CacheBox:Default", scope="instance")
    .property(name="maxHits", value=20, scope="instance"
```

WireBox Injector

- Then we ask the injector for objects
  - **getInstance('name or path')**
- WireBox has several creation & discovery styles
- WireBox has several injection styles

```
obj = injector.getInstance("model.path.Service");

obj2 = injector.getInstance("NamedKey");
```

Creation Styles



More than CFCs

Type	Description
CFC	ColdFusion Component
JAVA	Any Java object
WEBSERVICE	Any WSDL
RSS	Any RSS or Atom feed
DSL	Any registered or core injection DSL string
CONSTANT	Any value
FACTORY	Any other mapped object
PROVIDER	A registered provider object

More than CFC's

```
// map google news
map("GoogleNews")
     .toRSS("http://news.google.com/news?output=rss")
     .inCacheBox(timeout="30",lastAccessTimeout=10);

// Wire up java objects
map("SecurityService")
   .toJava("org.company.SecurityService)
   .setter(name="userService",ref="userService")
   .asSingleton();
map("UserService")
   .toJava("org.company.UserService")
   .asSingleton();

// wire up factory methods
map("FunkyEspresso")
   .toFactoryMethod(factory="SecurityService",method="getFunkyEspresso");

// Alias constant values
map("ToEmail").toConstant( getProperty('toEmail' );
```

Scopes

- Scopes manage created object persistence
- Default Scope -> **NOSCOPE (Transient Objects)**
- Scopes live inside of an Injector
- Several core scopes
- Develop custom scopes
- Override core scopes

WireBox Scopes

Scope	Comment
NOSCOPE	Transient objects
PROTOTYPE	Transient objects
SINGLETON	Only one instance of the object exists
SESSION	The CF Scope
REQUEST	The CF Scope
APPLICATION	The CF Scope
SERVER	The CF Scope
CACHEBOX	Time persisted objects in any CacheBox provider

Scoping

```
// map google news
map(“GoogleNews”)
     .toRSS(“http://news.google.com/news?output=rss”)
     .asEagerInit()
     .inCacheBox(timeout=”30”,lastAccessTimeout=10);

// Wire up java objects
map(“SecurityService”)
   .toJava(“org.company.SecurityService)
   .asSingleton()
   .setter(name=”userService”,ref=”userService”);
map(“UserService”)
   .asSingleton()
   .toJava(“org.company.UserService”);

// request based objects
map(“SearchCriteria”)
   .to(“model.search.Criteria”)
   .into(this.SCOPES.REQUEST);

// session based objects
map(“UserPreferences”)
   .to(“model.user.Preferences”)
   .into(this.SCOPES.SESSION);
```

Scoping

```
component{}

component singleton{}

component scope="session"{}

component scope="request"{}

component cache cacheTimeout="30"{}

component cachebox="ehCache" cacheTimeout="30"{}
```

Injection Styles

Style	Order	Motivation	Comments
Constructor	1	Mandatory dependencies for object creation	Each argument receives an inject annotation with its required injection DSL. Circular dependencies will fail via constructor injection unless WireBox Providers are used.
CFProperty	2	Great documentable approach to variable mixins to reduce getter/setter verbosity. Great for visualizing object dependencies. Safe for circular dependencies.	Mixin variables at runtime by using the cfproperty annotations. Cons is that you can not use the dependencies in an object's constructor method.
Setter/ Methods	3	Legacy or classic style	The inject annotation MUST exist on the setter method if the object is not mapped. Mapping must be done if you do not have access to the source or you do not want to touch the source.

Dependencies, Scope, Names?

Style	Pros	Cons
Annotations	• Documentable • Better visibility • Rapid Workflows • Just Metadata!	• Can pollute code • Some call intrusive • Unusable on compiled code
Configuration	• Compiled/Legacy Code • Multiple configurations per object • Visible Object Map	• Tedious • Slower workflow • Lower visibility

Injection Annotation

- Use one annotation: **inject**
- The value is our injection DSL
  - Tells the Injector what to inject in that placeholder
  - Concatenated strings separated by ":"
  - First section is called DSL namespace
  - **inject="id:MyService", inject="coldbox:plugin:Logger"**

Namespace	Description
ID-Model-Empty	Mapped references by key
WireBox	WireBox related objects: parent injectors, binder, properties, scopes
CacheBox	CacheBox related objects: caches, cache keys, etc
Provider	Object Providers
LogBox	LogBox related objects: loggers, root loggers
ColdBox	ColdBox related objects: interceptors, entity services, etc
Custom	Your own live annotations

Annotations

```
// CF Property Injection
property name="service" inject;
property name="service" inject="id:CoolService";
property name="log" inject="logbox:logger:{this}";

// Constructor
function init(service inject){
   variables.service = arguments.service;
   return init;
}

// Setter
function setService(service) inject{
  variables.service = arguments.service;
}
```

Component Annotations

- **@autowire** = boolean [true]
- **@alias** = list of know names for this CFC
- **@eagerInit** [false]
- **@scope** = valid scope
- **@singleton**
- **@cachebox** = cache provider [default]
- **@cache** [default]
- **@cacheTimeout** = minutes
- **@cacheLastAccessTimeout** = minutes

Property Annotations

- **inject** - Our main injection DSL
- **scope** - The scope in the CFC to inject into: **variables**

```
property name="securityService" inject="id:Security" scope="instance";

property name="MyDAO" inject="id:MyDAO";

property name="User" inject="provider:User";

property name="Log" inject="logbox:root";

property name="cache" inject="cachebox:ehcache";

property name="customService" inject="customDSL:my:annotations";
```

Method Annotations

- **inject** - Our main injection DSL
- **onDIComplete** - Mark a method to be executed at end of DI
- **provider** - Replace the method with a lookup provider method for that dependency

```
function setService(service) inject="id:MyService"{
  variables.service = arguments.service;
}

function startupService() onDIComplete{
  // let's startup this service here.
}

function getUser() provider="id:user"{}
```

Method Conventions

- If the following method signatures are found, we call them for you!
  - Injector Awareness
    - setBeanFactory()
    - setInjector()
  - ColdBox Awareness
    - setColdBox()
  - onDIComplete()

Event Model

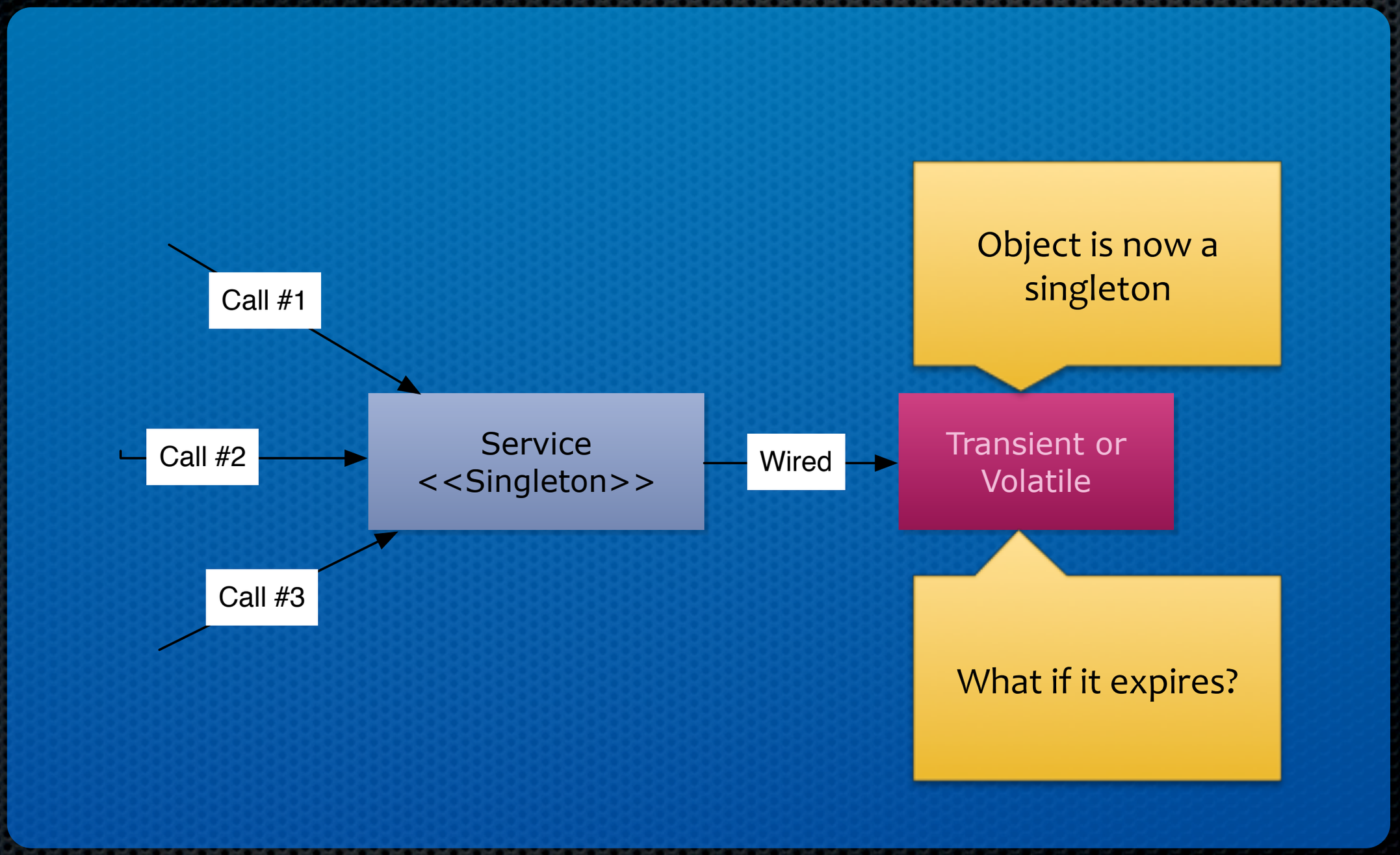
- Announce events throughout injector and object life cycles
- Create simple CFC listeners or enhanced ColdBox interceptors
- Modify CFCs, metadata, etc
- Extend WireBox-CacheBox YOUR WAY!

```
component{

    configure(injector,properties){}

    afterInstanceCreation(interceptData){
       var target = arguments.interceptData.target;

       target.$formatdate = variables.formatDate;
    }

    function formatDate(){}
}
```

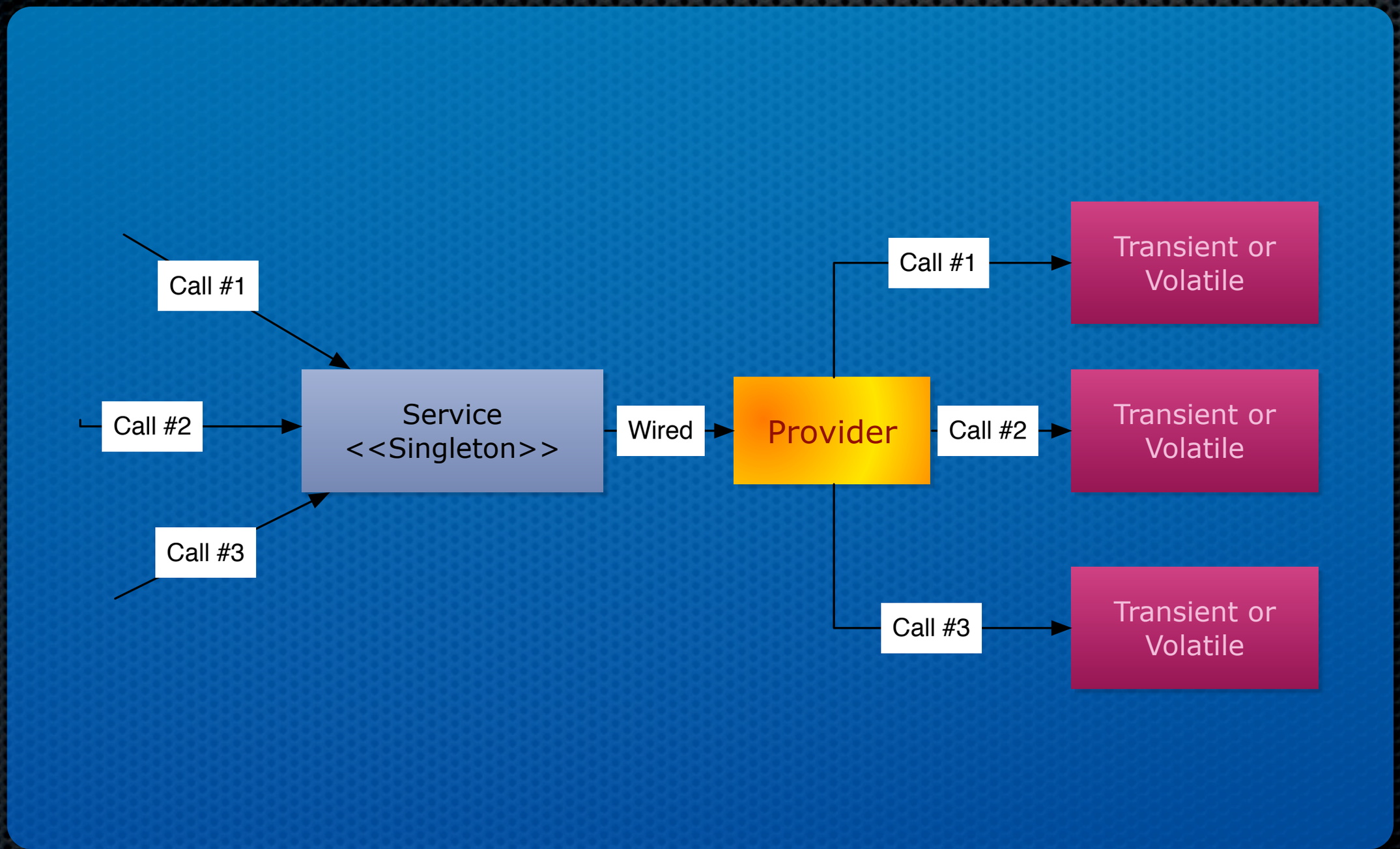
Event Model

Event	Data	Description
afterInjectorConfiguration	•**injector** : The calling injector reference	Called right after the injector has been fully configured for operation.
beforeInstanceCreation	•**mapping** : The mapping called to be created •**injector** : The calling injector reference	Called right before an object mapping is built via our internal object builders or custom scope builders.
afterInstanceInitialized	•**mapping** : The mapping called to be created •**target** : The object that just go constructed and initialized •**injector** : The calling injector reference	Called after an object mapping gets constructed and initialized. The mapping has NOT been placed on a scope yet and no DI/AOP has been performed yet.
afterInstanceCreation	•**mapping** : The mapping called to be created •**target** : The object that just go built, initialized and DI/AOP performed on it •**injector** : The calling injector reference	Called once the object has been fully created, initialized, stored, and DI/AOP performed on it. It is about to be returned to the caller via its *getInstance()* method.

Object Providers

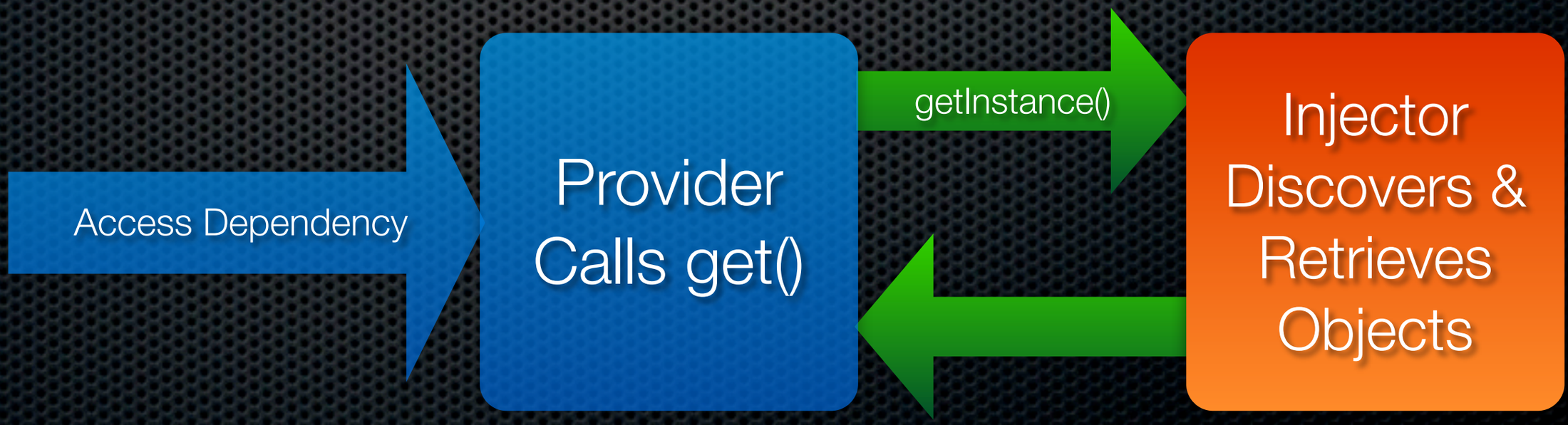


Object Providers



Object Providers

- Used For:
  - Delay construction of dependency
  - Transient dispenser
  - Volatile scoped objects (scope widening injection)
  - Legacy funkiness!
- Virtual Providers
- Custom Providers



Object Providers

```
// use the provider DSL namespace on a property
property name="searchCriteria" inject="provider:requestCriteria";

// To use it
searchCriteria.get().populate( form );
coolObjectProvider.get().executeSomeMethod();
// Or Proxy in onMissingMethod calls
searchCriteria.populate( form );
coolObjectProvider.executeSomeMethod(args);

// Lookup Methods
function getCoolObject() provider="HardToConstructObject"{}
// Use it
getCoolObject().executeMethod();


// Inject provider methods even if they don't exist from the binder
map("Service").toProviderMethod("getCriteria", "requestCriteria");
```

AOP The WireBox Way

Binder

```
// Map an aspect (it can have any dependency)
mapAspect("TransactionAspect").to("model.aspects.TransactionAspect");
mapAspect("MethodLogger").to("model.aspects.MethodLogger");

// Bind the aspects
bindAspect(classes=match().any(),
           methods=match().annotatedWith("transactional"),
           aspects="TransactionAspect");
```

Aspects

```
component implements="wirebox.system.aop.MethodInterceptor"{

      function invokeMethod(invocation){

         transaction{
            var results = invocation.proceed();
         }

         if( !isNull(results) ){
            return results;
         }

      }
}
```

AOP

- Matchers
  - any()
  - annotatedWith(annotation,value)
  - isInstanceOf()
  - regex()
  - mappings(list or single)
  - returns(type)
  - and(), or()
- Aspect Annotations
  - classMatcher
  - methodMatcher

```
/**
* @classMatcher any
* @methodMatcher annotatedWith(transactional)
*/
component
implements="wirebox.system.aop.MethodInterceptor"{

      function invokeMethod(invocation){

         transaction{
            var results = invocation.proceed();
         }

         if( !isNull(results) ){
            return results;
         }

      }
}
```

RoadMap

- AOP Finalization
- Scope Annotation aliases: *SessionScoped, RequestScoped, {scope}Scoped*
- Virtual Inheritance - Mix in methods & create virtual **$super** scope
- **Mix('udf')** - Runtime UDF mixins
- Binder Provider Methods: Creation of methods in your binder with **@provides**
- WireBox Spyglass Module
  - Code Coverage Aspect
  - Scope Analyzer & Visualizer
  - Object Graph Visualizer

Source Code

- If you are a source junky and want to help out:
- https://github.com/ColdBox/coldbox-platform

Issues & Mailing List

- Bugs, enhancements, ideas:
- https://github.com/coldbox/coldbox-platform/issues
- http://groups.google.com/group/coldbox

Q & A

Thanks!