Groovy Coherence

YEAH BABY!

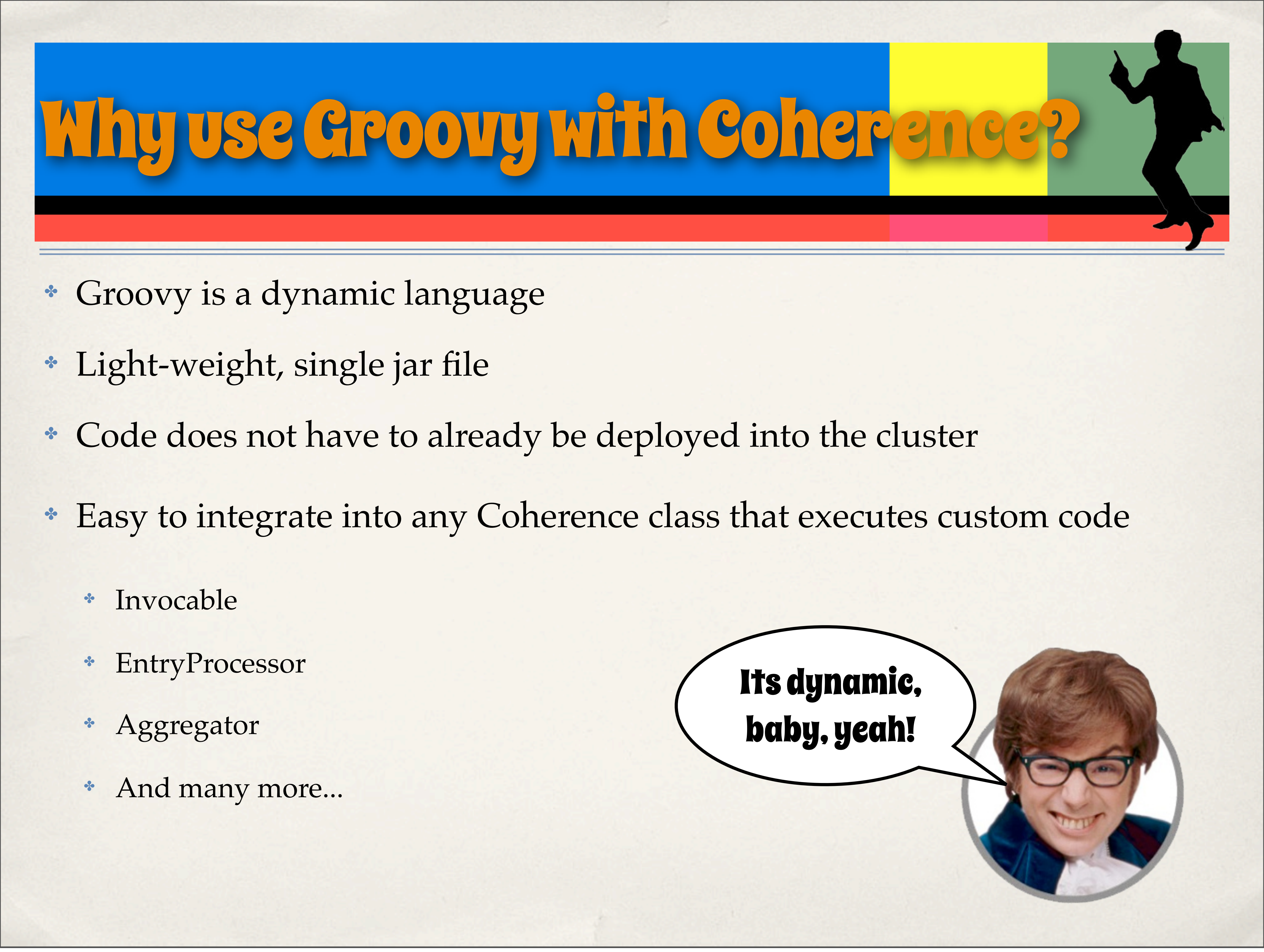
Jonathan Knight
thegridman.com

What is Groovy?

"Groovy is like a super version of Java.
It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing."

From http://groovy.codehaus.org/

Groovy is a light-weight, OO, dynamic programming language that runs on top of the JVM and has been around since about 2003.



Why use Groovy with Coherence?

- Groovy is a dynamic language
- Light-weight, single jar file
- Code does not have to already be deployed into the cluster
- Easy to integrate into any Coherence class that executes custom code
  - Invocable
  - EntryProcessor
  - Aggregator
  - And many more...

By dynamic, we mean it has the ability to change the behaviour and structure of objects at runtime. They allow you to do at runtime what static languages do at compile time. This can be very powerful but of course with great power comes great responsibility and overuse or the wrong use of dynamic features of a language can leave you in a mess.

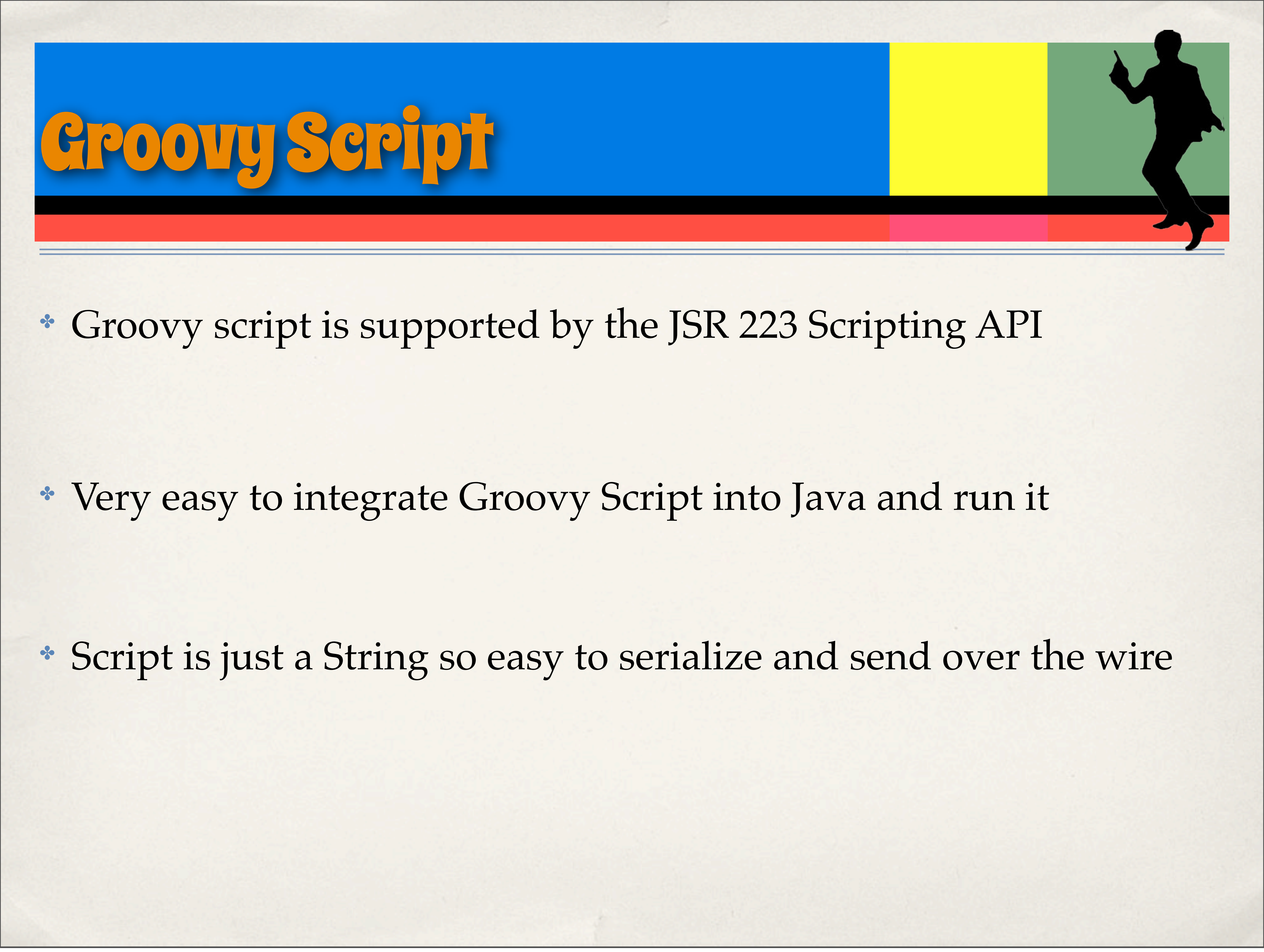
Groovy has a number of things going for it that make it ideal to integrate into Coherence. If you are a Java developer (which is highly likely if you are using Coherence) then the learning curve is low as Groovy's semantics are a lot like Java. As already mentioned it is dynamic which means you do not have to have thought of everything before you build and deploy your cluster. You can do things after the fact without having to have deployed code. For me when investigating problems in running clusters this is a massive advantage. Groovy also has a number of quite cool extensions to the JDK that make it easier to do things in less lines of code.

Using Groovy with Coherence

There are two ways to use Groovy and Coherence together

- Groovy Script embedded in Java code
- Use and extend Coherence APIs from Groovy code

You can use Groovy in two ways with Coherence. You can treat it just as a scripting language and only execute scripts from normal Java code or you can go the whole hog and use and extend Coherence from within Groovy code. We will cover both of these in turn.



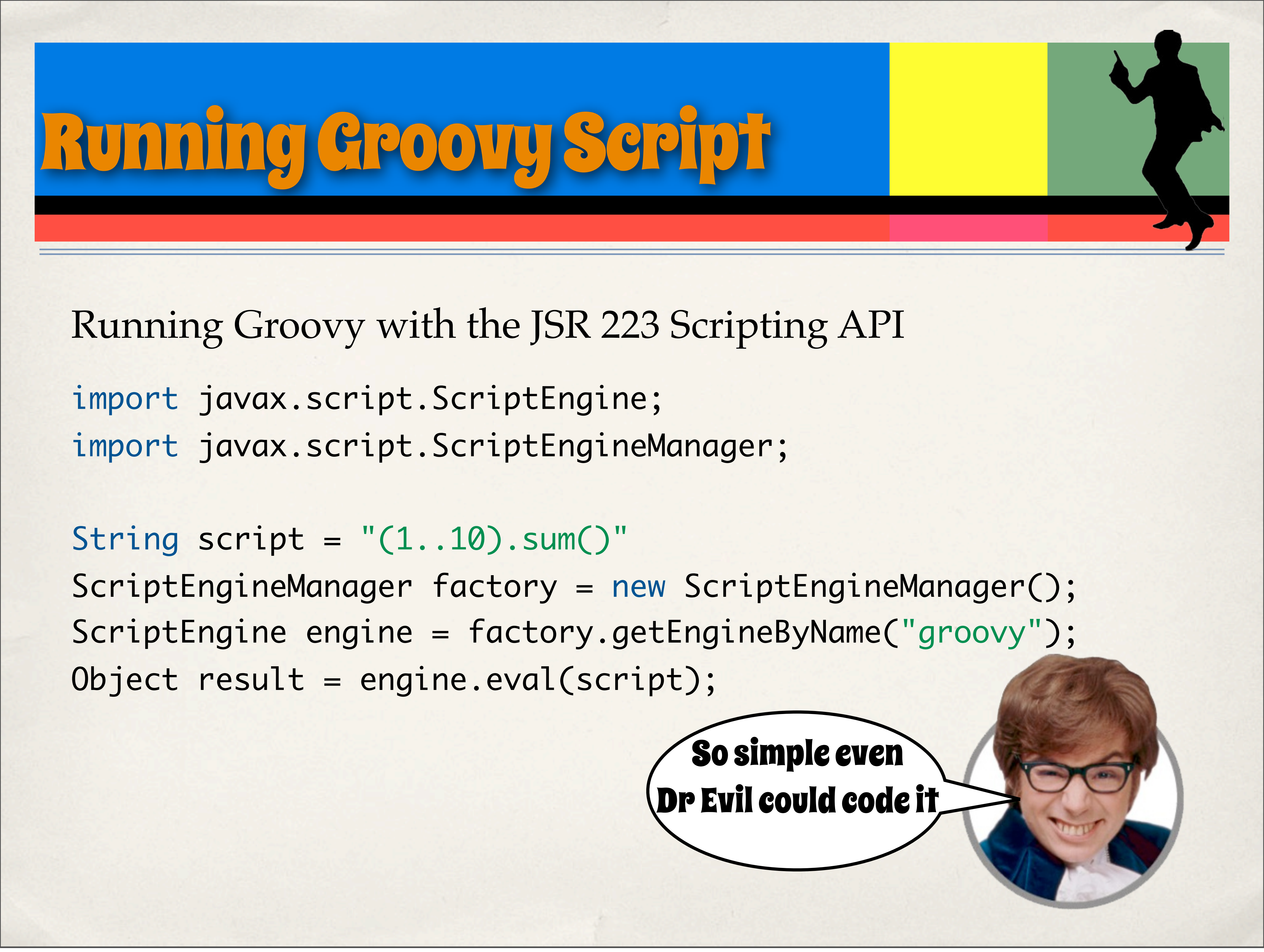
Groovy Script

- Groovy script is supported by the JSR 223 Scripting API
- Very easy to integrate Groovy Script into Java and run it
- Script is just a String so easy to serialize and send over the wire

Groovy Script is basically a snippet of Groovy code that is executed as a standalone piece of script from within an application. This can be very useful for adding dynamic functionality to applications as any ah-hoc piece of script can be compiled and executed at runtime.

Groovy is one of the languages supported by JSR 223 - the scripting API so this makes it very easy to execute Groovy Script from within a Java application.

Obviously there are a number of uses for dynamic script in Coherence. For me personally, the place I would be most likely to use it would be when investigating issues or bugs as it allows me to execute code to query the state of data or Coherence services without having to have thought up the scenarios I might need in advance. It is always the case that it is not until I need to investigate an issue with a running cluster that I wish I had thought of deploying a bit of code - an example would be when we had corruption of indexes a year or so back and we had no code on the server side that allowed us to look in detail at what indexes were present on a node and what data they contained. Using Groovy Script it would have been much easier.



Running Groovy Script

Running Groovy with the JSR 223 Scripting API

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;

String script = "(1..10).sum()"
ScriptEngineManager factory = new ScriptEngineManager();
ScriptEngine engine = factory.getEngineByName("groovy");
Object result = engine.eval(script);
```

The code above shows how simple it is. We have a String that contains the Groovy code (in this case we just sum up all the integers from 1 to 10). We then use the Java Scripting API to run the code. In the code above we end up executing the Groovy script and the return value of 55 is assigned to the result object.

As you can see Groovy script is just a String and is simple to execute which makes it very easy to integrate into various parts of Coherence. It becomes very easy to build a Groovy Script Invocable, EntryProcessor, Aggregator, in fact many parts of Coherence that execute ad-hoc code can be integrated with Groovy script.

A Groovy Script Invocable

```java
public class GroovyScriptInvocable extends AbstractInvocable implements PortableObject {
    private String script;

    public GroovyScriptInvocable() {
    }

    public GroovyScriptInvocable(String script) {
        this.script = script;
    }

    @Override
    public void run() {
        try {
            ScriptEngineManager factory = new ScriptEngineManager();
            ScriptEngine engine = factory.getEngineByName("groovy");
            engine.put("invocationService", getService());
            setResult(engine.eval(script));
        } catch (ScriptException e) {
            throw ensureRuntimeException(e, "Error running Groovy script\n" + script);
        }
    }

    @Override
    public void readExternal(PofReader pofReader) throws IOException {
        this.script = pofReader.readString(0);
    }

    @Override
    public void writeExternal(PofWriter pofWriter) throws IOException {
        pofWriter.writeString(0, this.script);
    }
}
```

As an example we will write a Groovy Invocable as this is about as basic as we can get. All our Invocable needs to do is take a piece of Groovy script as a parameter, execute the script and return any result.

As you can see it is pretty basic. The run() method does the following
Sets up the scripting engine to execute Groovy script
We then set a script variable into the engine; in this case we add the InvocationService that is executing the Invocable as a parameter with the name invocationService. This InvocationService will then be available to code in the script by using the specified name.
Finally we execute the script and any return value is set as the result of the Invocable.

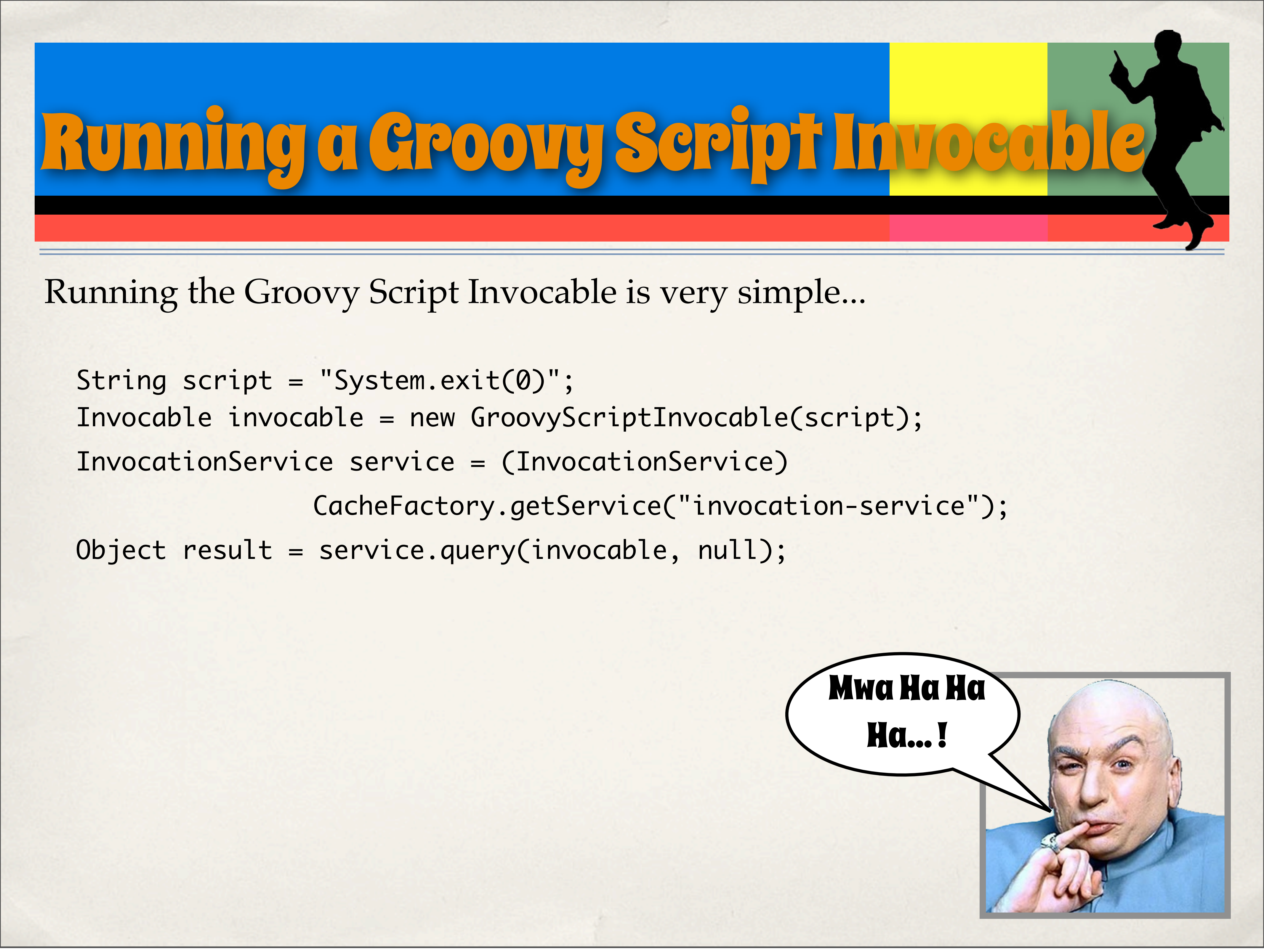
Running a Groovy Script Invocable

Running the Groovy Script Invocable is very simple...

```
String script = "CacheFactory.getCluster().getLocalMember().toString()";
Invocable invocable = new GroovyScriptInvocable(script);
InvocationService service = (InvocationService)
               CacheFactory.getService("invocation-service");
Object result = service.query(invocable, null);
```

The above code sets a String variable with some Groovy Script – in this case we are summing the integer values between 1 and 10 (which comes to 55).
After the call to service.query the result variable will be set to 55.

Looking at the above code you can see it would be equally as simple to then write a Groovy Script EntryProcessor as the code would be virtually identical. This time though instead of setting an InvocationService as the parameter we can pass the InvocableMap.Entry the EntryProcessor is being invoked against.



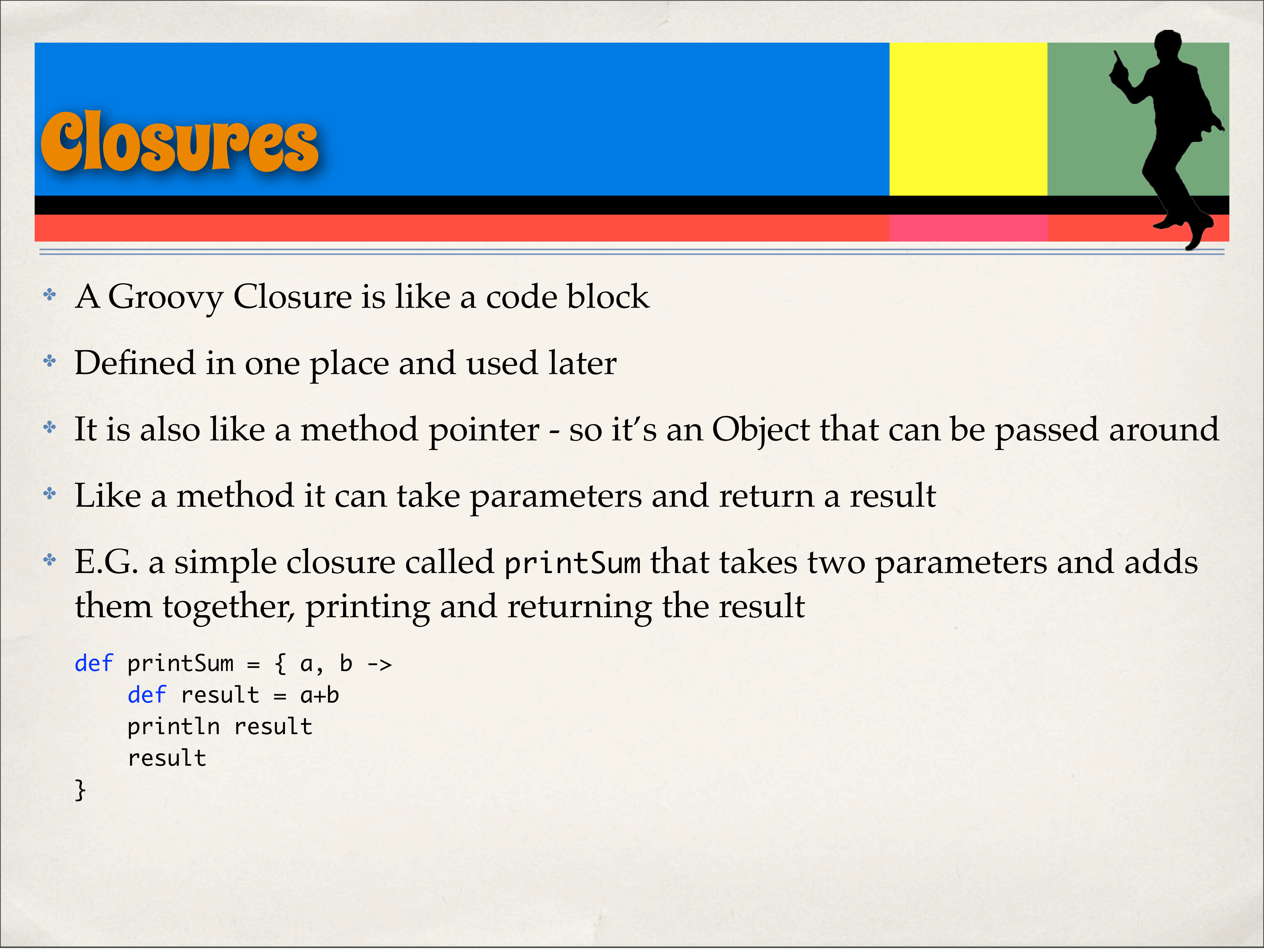
Running a Groovy Script Invocable

Running the Groovy Script Invocable is very simple...

```
String script = "System.exit(0)";
Invocable invocable = new GroovyScriptInvocable(script);
InvocationService service = (InvocationService)
               CacheFactory.getService("invocation-service");
Object result = service.query(invocable, null);
```

The above code sets a String variable with some Groovy Script – in this case we are summing the integer values between 1 and 10 (which comes to 55).
After the call to service.query the result variable will be set to 55.

Looking at the above code you can see it would be equally as simple to then write a Groovy Script EntryProcessor as the code would be virtually identical. This time though instead of setting an InvocationService as the parameter we can pass the InvocableMap.Entry the EntryProcessor is being invoked against.



Closures

- A Groovy Closure is like a code block
- Defined in one place and used later
- It is also like a method pointer - so it’s an Object that can be passed around
- Like a method it can take parameters and return a result
- E.G. a simple closure called `printSum` that takes two parameters and adds them together, printing and returning the result

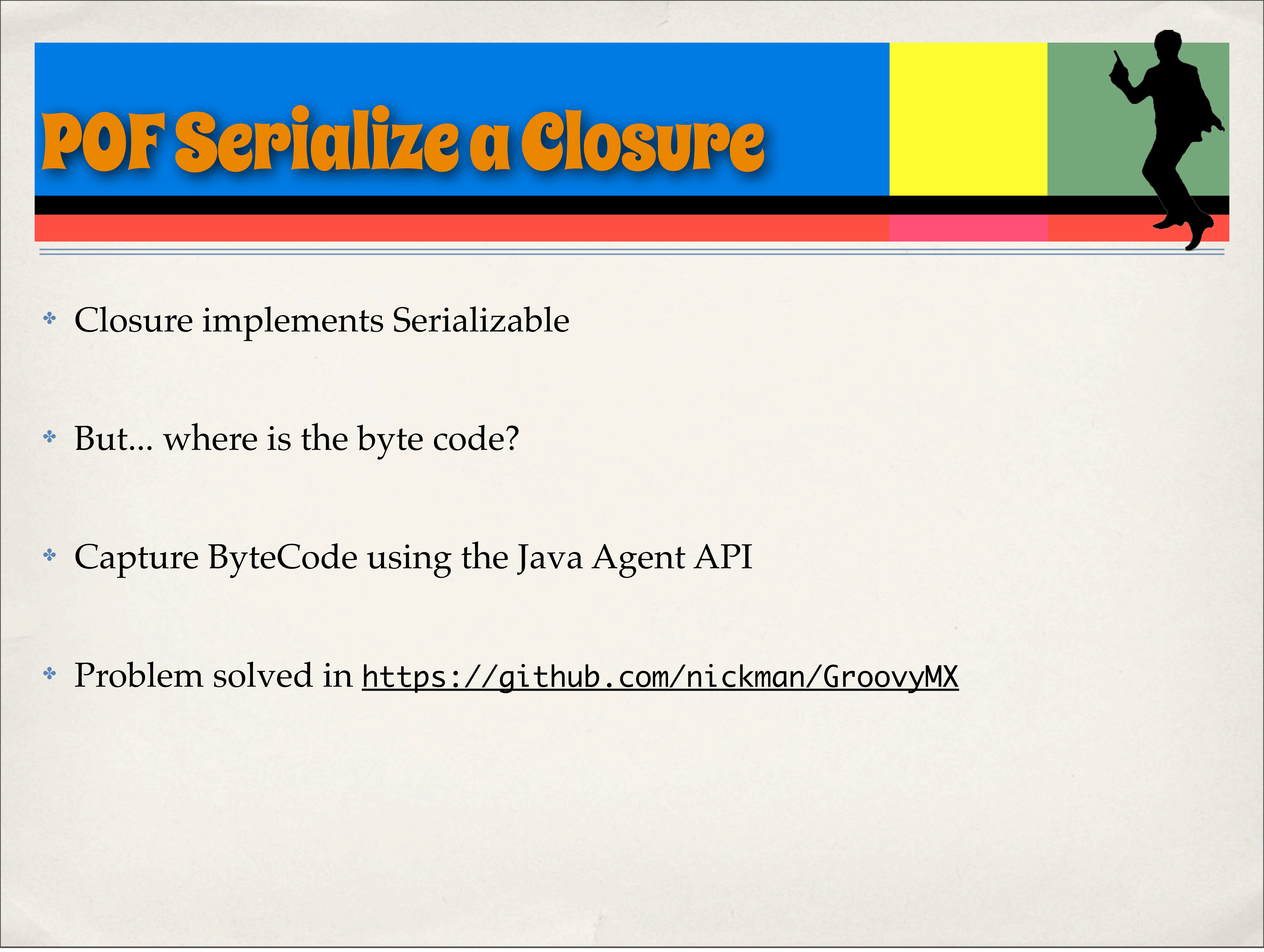
```
def printSum = { a, b ->
    def result = a+b
    println result
    result
}
```

Now we have seen how easy it is to use Groovy Script we can get onto the cooler stuff and integrate Coherence directly into Groovy so that you can use it from Groovy code. Obviously as Coherence is a Java application you could use it from Groovy anyway without any changes by calling the standard API methods. What we are going to look at though is how we can extend the Coherence API in Groovy to make use of some of Groovy's cooler features especially Closures.

Closures In Coherence

- Dynamic so like Groovy Script the code does not need to be deployed to the cluster
- Closures are good as EntryProcessors and Invocables
- Could actually be used anywhere custom code is used in Coherence
- Combined with the Groovy Console can make a powerful diagnostic and “hacking” tool

Like Groovy Script a closure can be very useful in Coherence
If we want to fully integrate Coherence with the Groovy language we need to make it support Closures



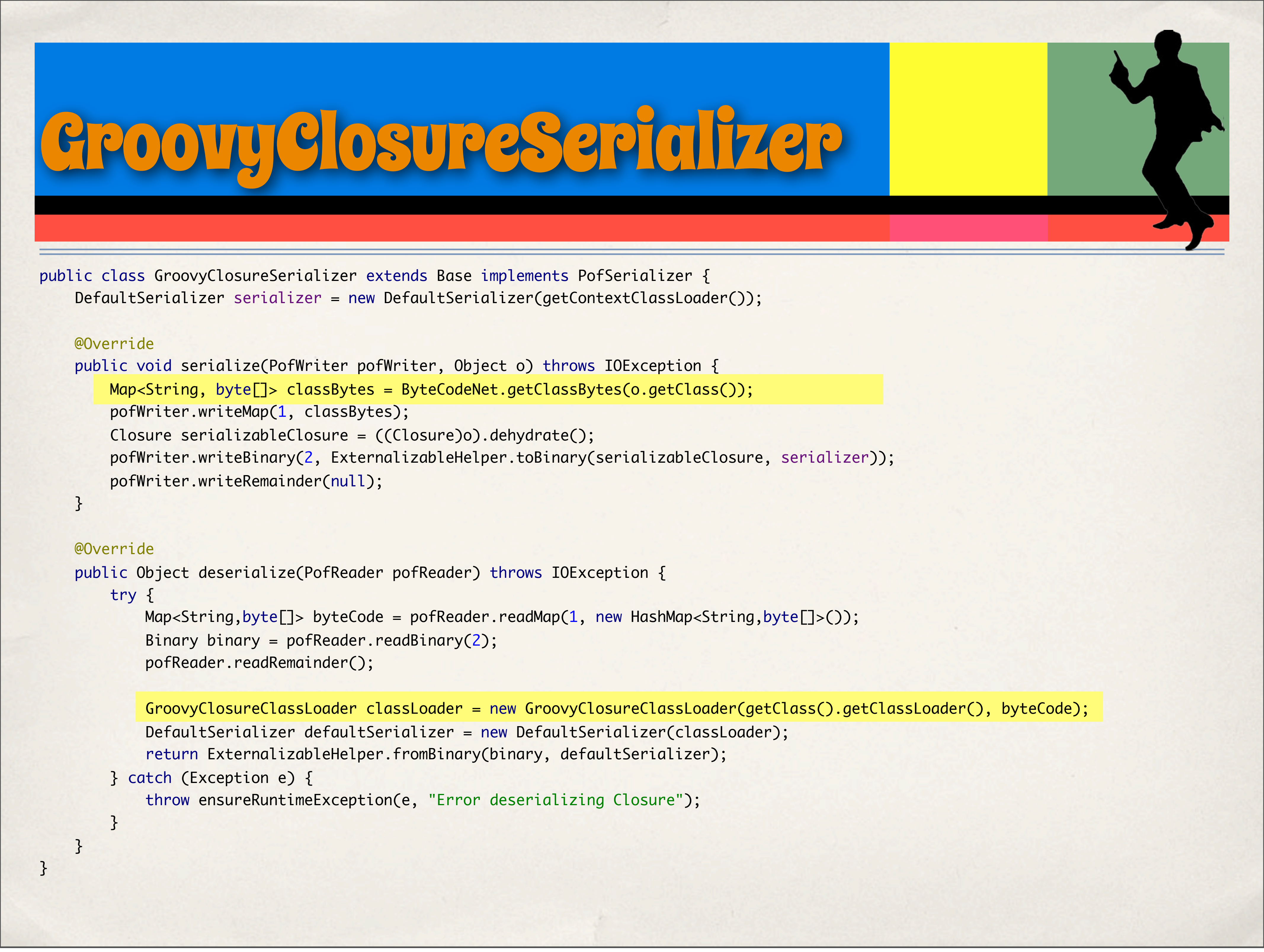
POF Serialize a Closure

- Closure implements Serializable
- But… where is the byte code?
- Capture ByteCode using the Java Agent API
- Problem solved in https://github.com/nickman/GroovyMX

As a Groovy Closure is an Object then it should be possible to pass it into a Coherence cluster

When I started to look at integrating Groovy and Coherence I was vey pleased to see that the Closure class implements Serializable, so joy of joys, it should be easy to serialize over the wire to a Coherence cluster. Well, my joy was to be short lived.

Whilst a Closure is indeed serializable it contains some internal fields that are not going over the wire and typically may not be Serializable. After a bit of Googling I found that later versions of Groovy have added a dehydrate method to a Closure which returns the same closure with these fields set to null so it can be serialized. Just about every cluster I have ever worked on uses POF so the next step was to write a POF serializer that can serialize the Closure.

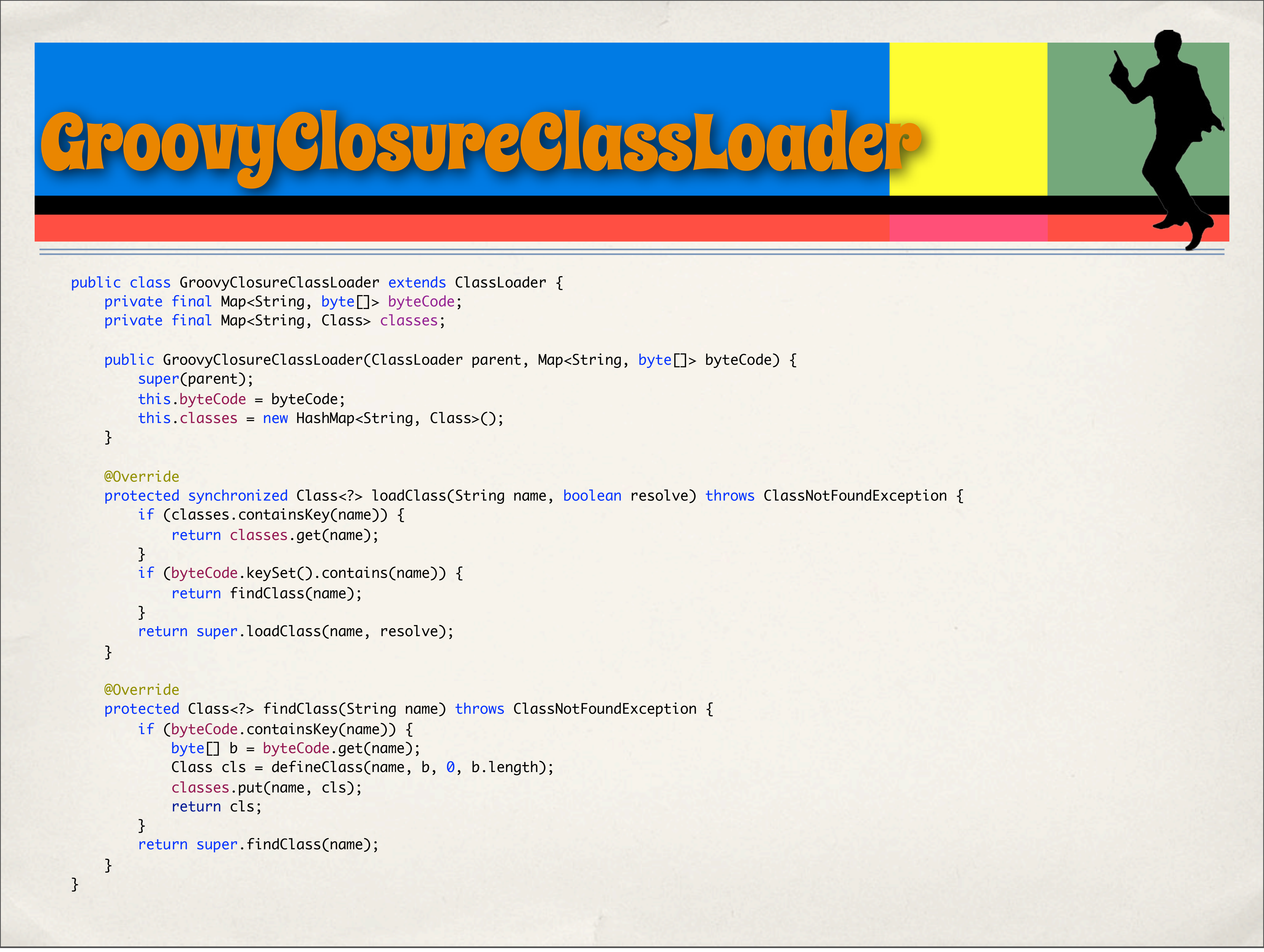


GroovyClosureSerializer

```java
public class GroovyClosureSerializer extends Base implements PofSerializer {
    DefaultSerializer serializer = new DefaultSerializer(getContextClassLoader());

    @Override
    public void serialize(PofWriter pofWriter, Object o) throws IOException {
        Map<String, byte[]> classBytes = ByteCodeNet.getClassBytes(o.getClass());
        pofWriter.writeMap(1, classBytes);
        Closure serializableClosure = ((Closure)o).dehydrate();
        pofWriter.writeBinary(2, ExternalizableHelper.toBinary(serializableClosure, serializer));
        pofWriter.writeRemainder(null);
    }

    @Override
    public Object deserialize(PofReader pofReader) throws IOException {
        try {
            Map<String,byte[]> byteCode = pofReader.readMap(1, new HashMap<String,byte[]>());
            Binary binary = pofReader.readBinary(2);
            pofReader.readRemainder();

            GroovyClosureClassLoader classLoader = new GroovyClosureClassLoader(getClass().getClassLoader(), byteCode);
            DefaultSerializer defaultSerializer = new DefaultSerializer(classLoader);
            return ExternalizableHelper.fromBinary(binary, defaultSerializer);
        } catch (Exception e) {
            throw ensureRuntimeException(e, "Error deserializing Closure");
        }
    }
}
```

This is the code for the serializer
Not too complicated
Uses the Coherence DefaultSerializer to take care of the Closure – as this implements Serializable
Byte code is captured using the GroovyMX ByteCodeNet class then serialized as the map of byte arrays

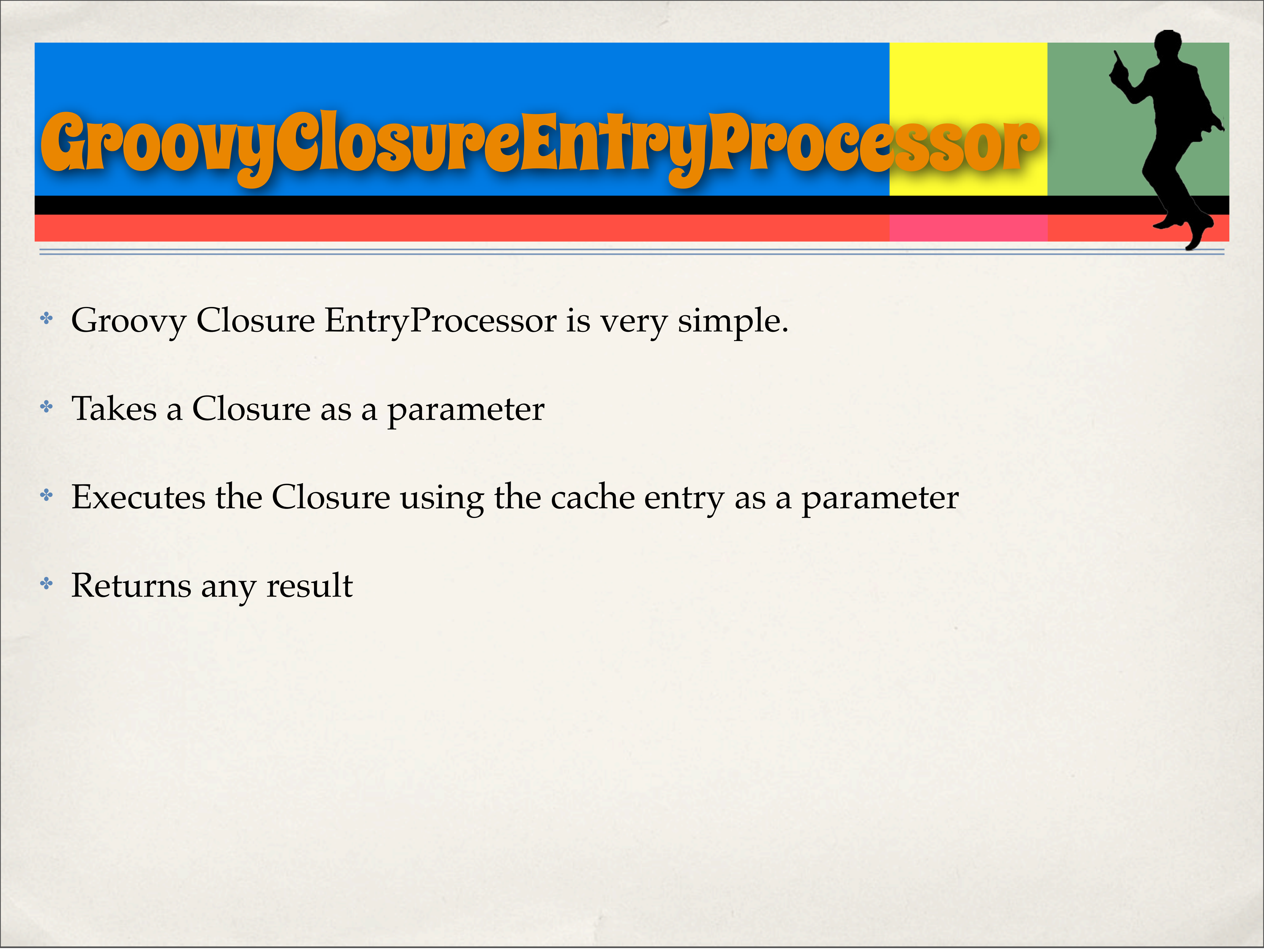
Deserialization is the reverse with the addition of a special classloader that can load the byte code



GroovyClosureClassLoader

```java
public class GroovyClosureClassLoader extends ClassLoader {
    private final Map<String, byte[]> byteCode;
    private final Map<String, Class> classes;

    public GroovyClosureClassLoader(ClassLoader parent, Map<String, byte[]> byteCode) {
        super(parent);
        this.byteCode = byteCode;
        this.classes = new HashMap<String, Class>();
    }

    @Override
    protected synchronized Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException {
        if (classes.containsKey(name)) {
            return classes.get(name);
        }
        if (byteCode.keySet().contains(name)) {
            return findClass(name);
        }
        return super.loadClass(name, resolve);
    }

    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        if (byteCode.containsKey(name)) {
            byte[] b = byteCode.get(name);
            Class cls = defineClass(name, b, 0, b.length);
            classes.put(name, cls);
            return cls;
        }
        return super.findClass(name);
    }
}
```

ClassLoader code - gain not too complicated
Just a normal ClassLoader but loads specific classes from the specified Map of byte code



GroovyClosureEntryProcessor

- Groovy Closure EntryProcessor is very simple.
- Takes a Closure as a parameter
- Executes the Closure using the cache entry as a parameter
- Returns any result

So now we can use Closures in Coherence
Simple example - an EntryProcessor that takes a Closure

GroovyClosureEntryProcessor

```java
public class GroovyEntryProcessor extends AbstractProcessor implements PortableObject
{
    private Closure closure;

    public GroovyEntryProcessor()
    {
    }

    public GroovyEntryProcessor(Closure closure)
    {
        this.closure = closure;
    }

    @Override
    public Object process(InvocableMap.Entry entry)
    {
        return closure.call(entry);
    }

    @Override
    public void readExternal(PofReader pofReader) throws IOException
    {
        closure = (Closure) pofReader.readObject(0);
    }

    @Override
    public void writeExternal(PofWriter pofWriter) throws IOException
    {
        pofWriter.writeObject(0, closure);
    }
}
```

Code is very easy
Can use the same technique for other Coherence classes such as Invocables etc...

Using the Groovy EntryProcessor

- Using our GroovyClosureEntryProcessor from Groovy Code...

The clunky way

```groovy
def cache = CacheFactory.getCache("accounts");
def key = new AccountId("0441234");
def expireClosure = { entry ->
  entry.expire(60000)
};
def processor = new GroovyEntryProcessor(expireClosure);
cache.invoke(key, processor);
```

Now we can properly integrate Coherence into Groovy
Use the Groovy metaClass to add the relevant methods to Coherence classes at runtime
Now we can use a closure as an EntryProcessor

Now it gets cool...

- We need an invoke method that takes a Closure as a parameter
- Use Groovy metaClass to add functionality to Coherence classes

```
InvocableMap.metaClass.invoke = { Object key, Closure c ->
   invoke(key, new GroovyEntryProcessor(c))
}
```

- Using our Closure EntryProcessor the Groovy Way...

```
def cache = CacheFactory.getCache("accounts");
def key = new AccountId("0441234");
cache.invoke(key, { entry ->
   entry.expire(60000)
});
```

Now we can properly integrate Coherence into Groovy
Use the Groovy metaClass to add the relevant methods to Coherence classes at runtime
Now we can use a closure as an EntryProcessor

And Finally...

- Write up posted on my blog http://thegridman.com
- Groovy Site - http://groovy.codehaus.org/
- Last but by no means least...

THANK YOU TO EVERYONE

WHO DONATED AT

http://www.justgiving.com/GridMan