Dynalink

Dynamic Linker Framework for Languages on the JVM

Attila Szegedi, Software Engineer, Twitter Inc.
@asz

Cross-language interop

From http://www.infoq.com/articles/neal-gafter-on-java:

Neal Gafter: … But another thing would be to **make it easier for languages to interoperate with each other on the platform**. And that was one of the focuses of Microsoft with the DLR and the addition of dynamic as a type to support language interoperability in the C# programming language.

The idea is that what Microsoft has done, it has made it so that these languages not only can execute efficiently, they have support for efficient dynamic dispatch, but **there's also a meta-object protocol so that objects from one dynamic language can have some reasonable semantics when used from another programming language, and so that you can interoperate between languages with some reasonable semantics**.

So I think that adding a meta-object protocol, either at the VM or a veneer above the VM possibly, would be valuable for interoperation between dynamic languages.

What is a MOP?

- Property access
  - `obj.foo obj.foo=value`
- Method invocation
  - `obj.doIt(arg1, arg2)`
- Element access
  - `arr[i] map[key]`
- These operations constitute the Metaobject Protocol

How to implement it?

- Force all language implementations to use a unified interface
- Use “navigators” (Jaxen style)
- Use Java 7 customized linking (“invokedynamic”)

How JVM links

```
package com.acme.buildings;

public class Shed {
  public setColor(int color) { ... }
}

...

Shed shed = new Shed();
shed.setColor(0xae17e3);
```

How JVM links

- One linking semantics
- Target method signature fixed at compile time for:
  - receiver type
  - overloaded method resolution
  - varargs
- Needs lot of static information

In a dynamic language

```
var jsShed = {}
var pojoShed = new com.acme.buildings.Shed();
var sheds = [jsShed, pojoShed];
for(var i in sheds) {
  sheds[i].color = 0xae17e3;
}
```

Polymorphic call site

- First dispatch has to go to ScriptableObject.setProperty("color", …)
- Second dispatch has to go to Shed.setColor(…)

POJOs everywhere

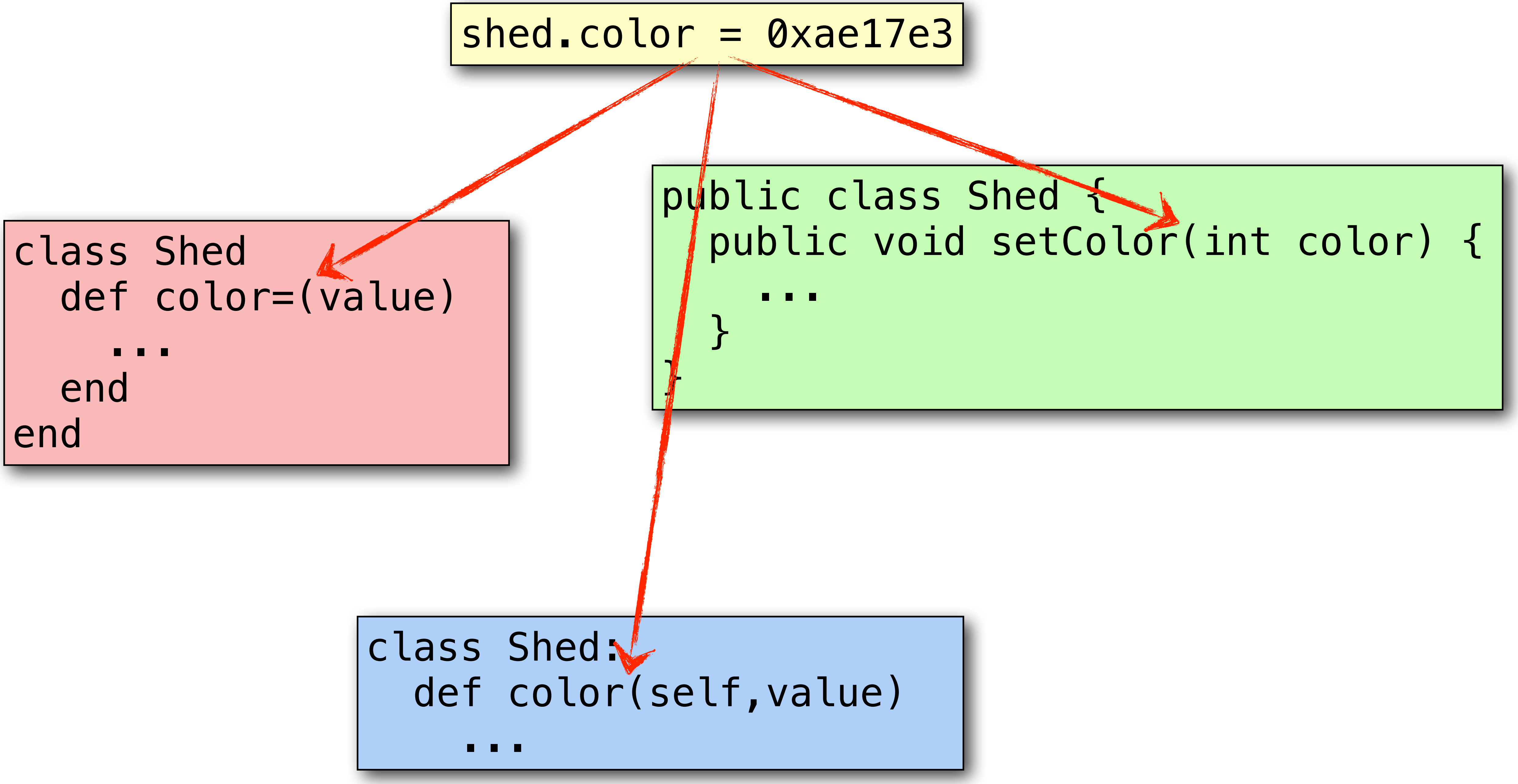
- Every JVM language runtime has a POJO interop.
- To access the vast amount of Java libraries. It's why it's on the JVM.
- Usually implemented by wrappers.

Add a twist

```
var jsShed = {}
var pojoShed = new com.acme.buildings.Shed();
var rubyShed = somehowObtainRubyShed();
var sheds = [jsShed, pojoShed, rubyShed];
for(var i in sheds) {
  sheds[i].color = 0xae17e3;
}
```

- Third dispatch has to go to IRubyObject.setInstanceVariable(…)
- Entirely different linking semantics than the JVM built-in ones.

What's the problem?



```
shed.color = 0xae17e3
```

```
class Shed
  def color=(value)
    ...
  end
end
```

```
public class Shed {
  public void setColor(int color) {
    ...
  }
}
```

```
class Shed:
  def color(self,value)
    ...
```

What's invokedynamic

- New feature in Java 7 where classes can:
  - specify that some of its call sites should be linked differently, and
  - specify a method that implements the semantics for linking these call sites, instead of having JVM link it.

"Neither invoke, nor dynamic" – Charlie Nutter

Apply to example

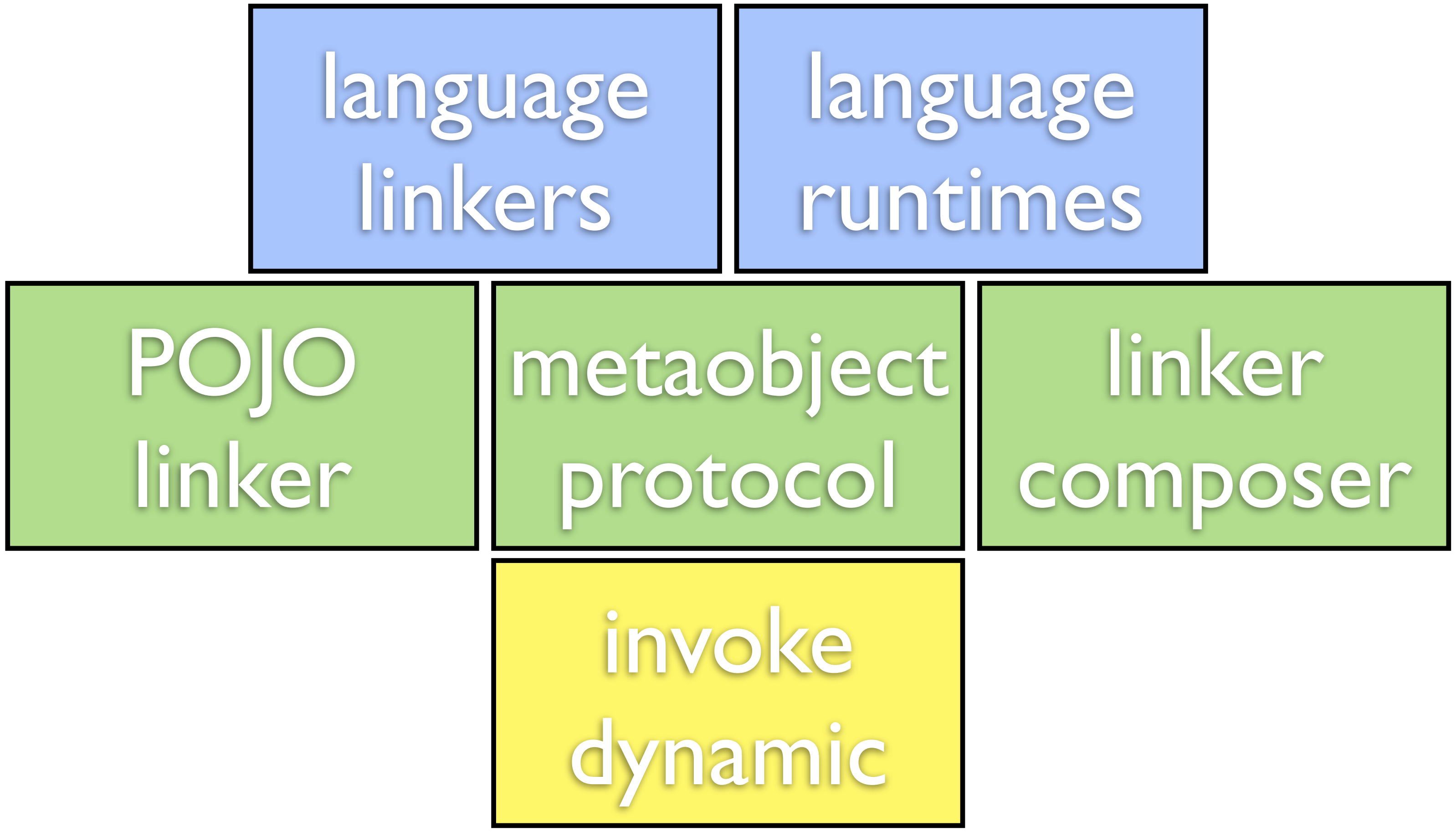
```
var jsShed = {}
var pojoShed = new com.acme.buildings.Shed();
var rubyShed = somehowObtainRubyShed();
var sheds = [jsShed, pojoShed, rubyShed];
for(var i in sheds) {
  sheds[i].color = 0xae17e3;
}
```

- You can compile to bytecode that'll specify a dynamic call site, and provide code to relink it when it encounters different kinds of targets.

What invokedynamic isn't

- Doesn't come with POJO semantics built-in
- Doesn't let your JavaScript code discover the Ruby method it has to link with
- Doesn't let your Ruby runtime advertise the availability of its methods to link with
- Does not establish a metaobject protocol

Pieces of the puzzle



language linkers

language runtimes

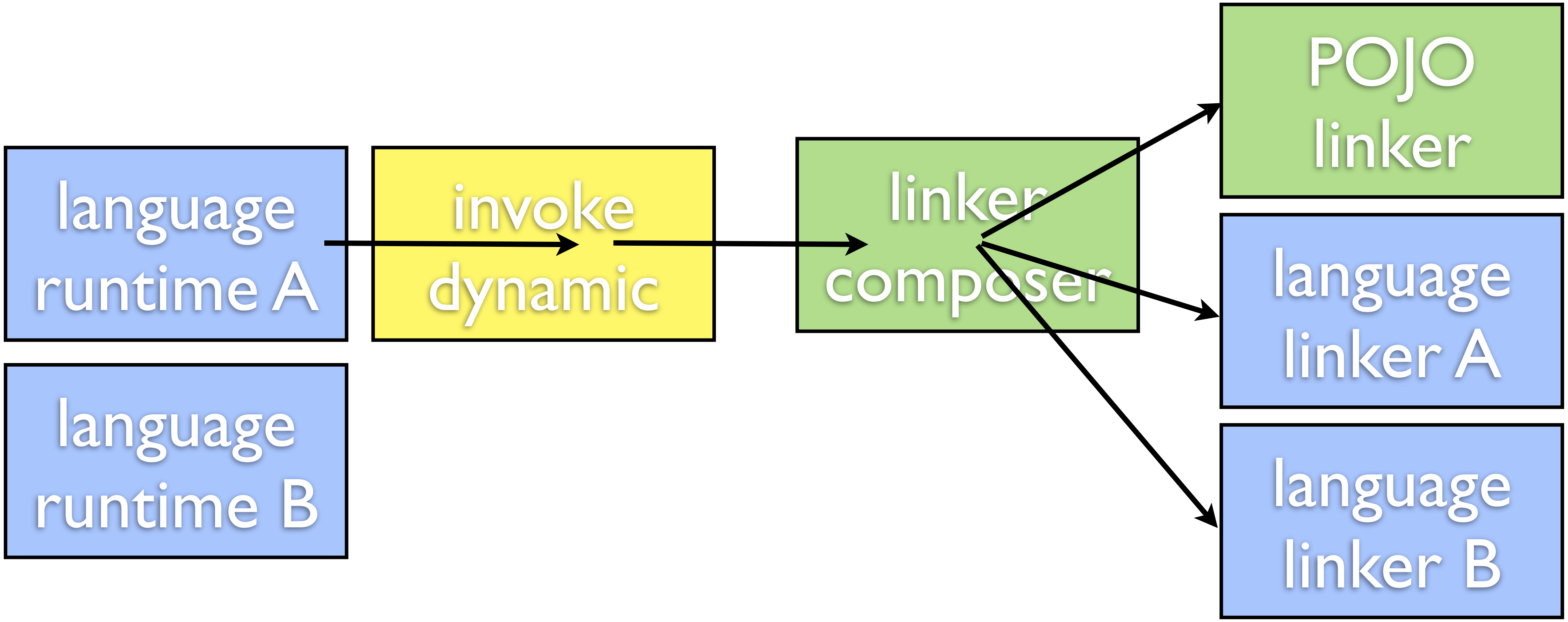
POJO linker

metaobject protocol

linker composer

invoke dynamic

Pieces of the puzzle



language runtime A

language runtime B

invoke dynamic

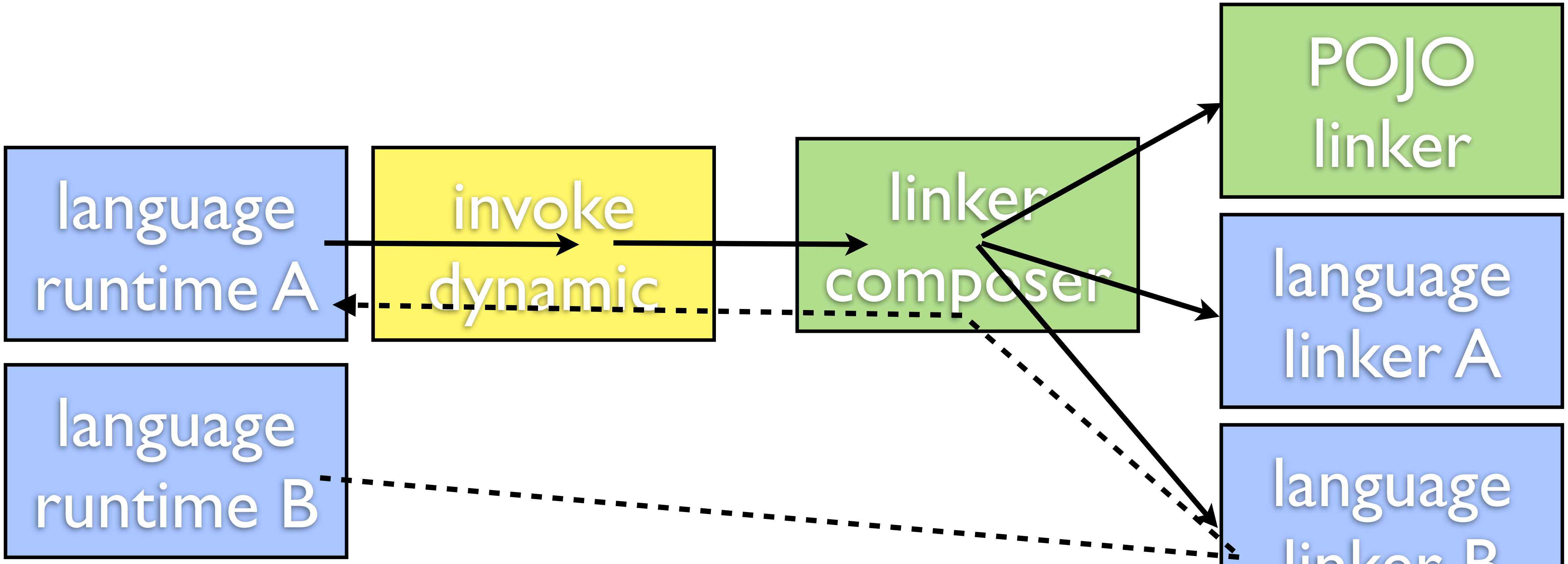
linker composer

POJO linker

language linker A

language linker B

Pieces of the puzzle



Why do you care?

- The promise of Java library interop extended to JVM language interop.
- Language runtimes on JVM seldom work on their native objects only.
- At the very least, they also want to provide integration with POJOs.
- Dynalink lets you have invokedynamic call sites that automatically relink between different language object behaviors.

Is linking POJOs such a big deal?

- Actually, yes. There's lots of edge cases in:
  - overloaded methods
  - vararg methods
  - type conversions

…you can do combinations.

- Your users will bug you and/or lose confidence if you get this wrong

Ok, how do I use the POJO linker with ASM?

```
    import org.objectweb.asm.MethodHandle;
import static org.objectweb.asm.Opcodes.MH_INVOKESTATIC;
...
private static final MethodHandle BOOTSTRAP =
    new MethodHandle(MH_INVOKESTATIC,
        "org/dynalang/dynalink/support/DefaultBootstrapper","bootstrap",
        MethodType.methodType(CallSite.class, MethodHandles.Lookup.class,
            String.class, MethodType.class).toMethodDescriptorString());
private static final Object[] BOOTSTRAP_ARGS = new Object[0];
...
mv.visitIndyMethodInsn("dyn:setProp:color", "(Ljava/lang/Object;I)V",
    BOOTSTRAP, BOOTSTRAP_ARGS);
mv.visitIndyMethodInsn("dyn:getProp:shape", "(Ljava/lang/Object;)Ljava/lang/String;",
    BOOTSTRAP, BOOTSTRAP_ARGS);
```

And how do I use it with BiteScript?

```
handle = compiler.method.mh_invokestatic(
  org.dynalang.dynalink.DefaultBootstrapper,
  "bootstrap",
  java.lang.invoke.CallSite,
  java.lang.invoke.MethodHandles::Lookup,
  java.lang.String,
  java.lang.invoke.MethodType)

compiler.method.invokedynamic(
  "dyn:setProp:color",
  [void, java.lang.Object, int],
  handle)

compiler.method.invokedynamic(
  "dyn:getProp:shape",
  [java.lang.String, java.lang.Object],
  handle)
```

What is a MOP? (repeated)

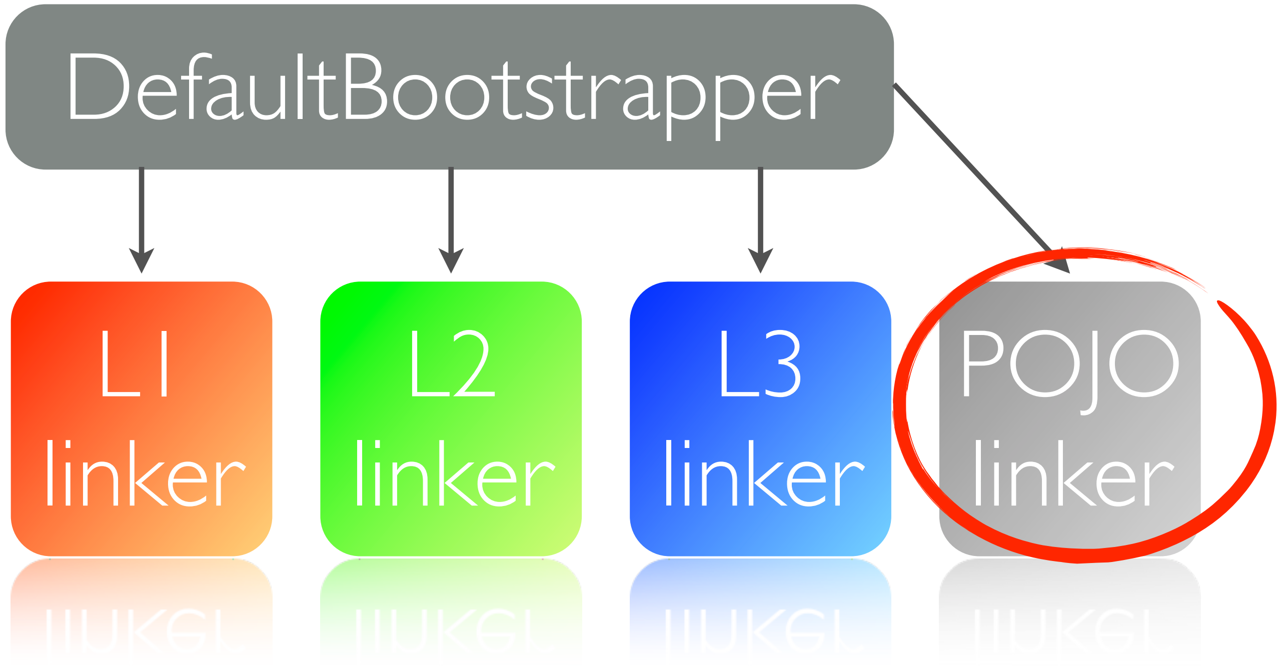
- Property access
  - `obj.foo`
  - `obj.foo=value`
- Method invocation
  - `obj.doIt(arg1, arg2)`
- Element access
  - `arr[i] map[key]`

MOP naming conventions

- Property access
  - `dyn:getProp:foo`
    `dyn:setProp:foo`
- Method invocation
  - `dyn:callPropWithThis`
- Element access
  - `dyn:getElem dyn:setElem`

And how do I link with other languages?

- You just did. No extra steps are needed.
- You call site will be linked with any dynalink aware language runtime when invoked on its objects.
- 2-for-1 deal.



Dynalink city motto:
Come for POJO linking, stay for the cross-language interop.

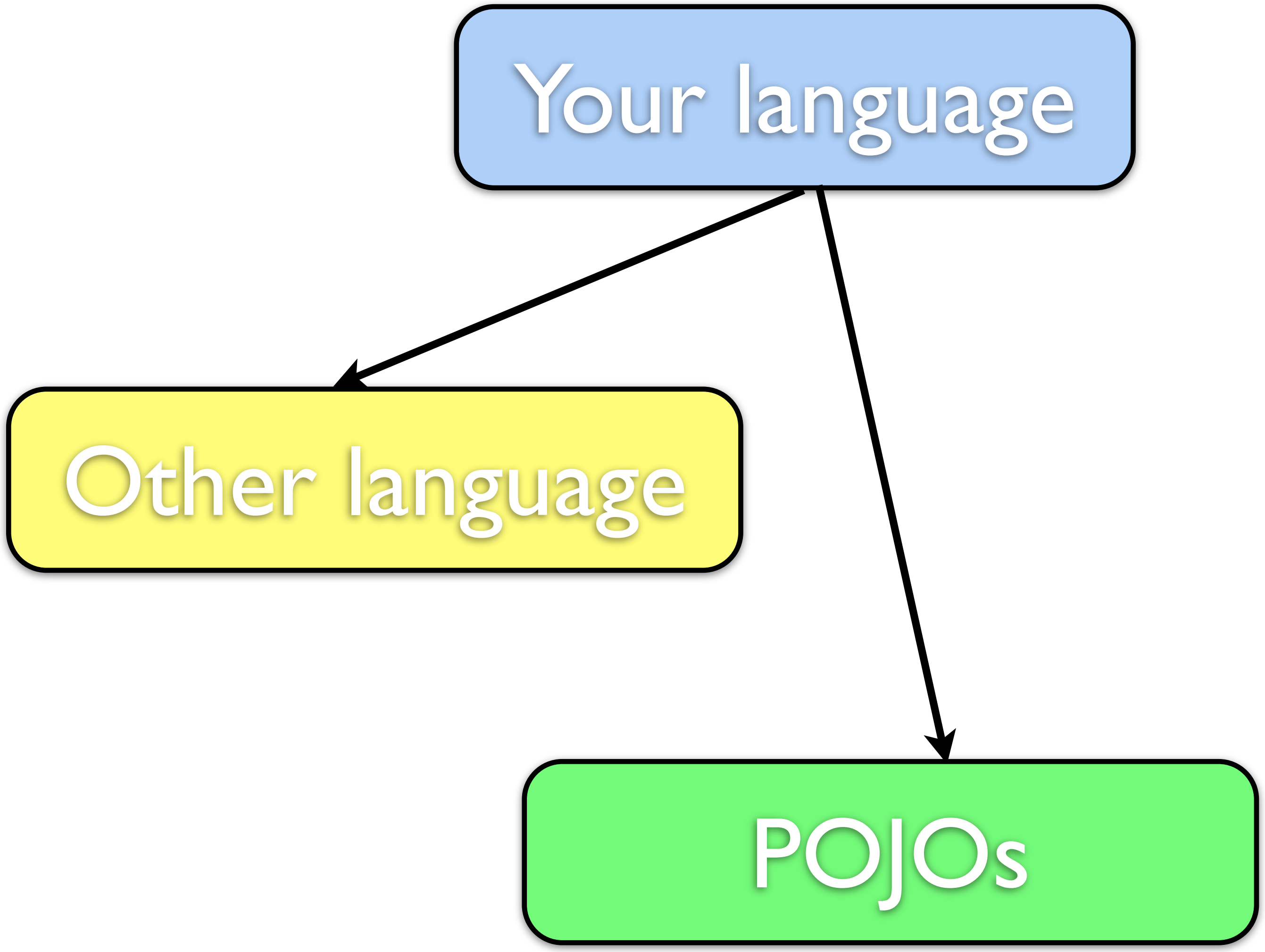
POJO linker specifics

- Element access and length works with Java arrays, j.u.List, and j.u.Map, relinks as necessary:

```
objs = [someJavaList, someJavaMap]
indices = [3, “foo”]
for(i = 0; i < 2; ++i) {
    print(objs[i][indices[i]]);
}
```

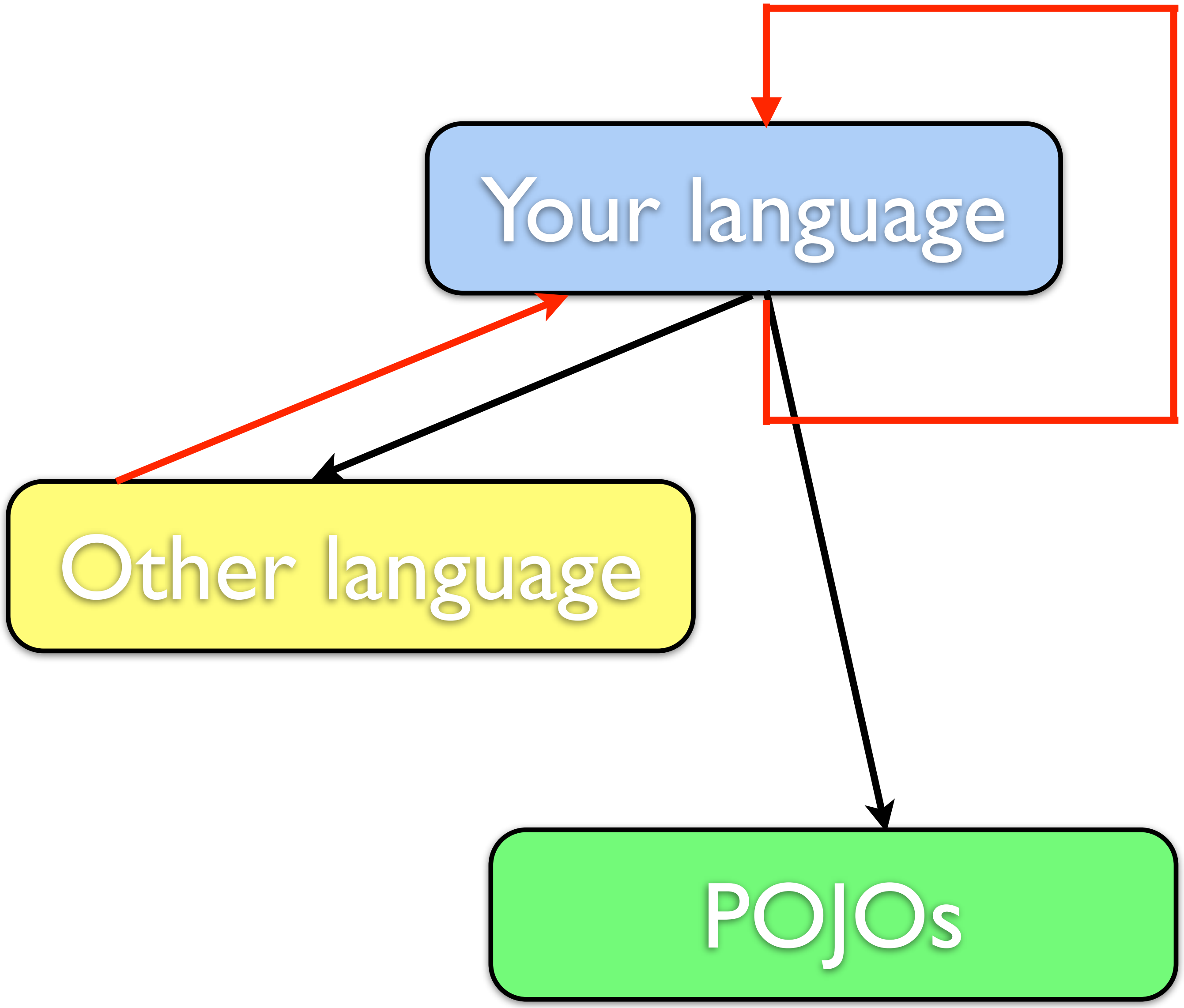
With code so far...

- You get fully JLS-compliant dynamic linkage to POJOs.
- 2-for-1 deal gives you automatic dynamic linkage to other languages.



What you still can't do

- let other language runtimes link to code for your objects.
- Wait, you can't link to your own objects either!



Time to write some code

- You need to actually write the code for the linker for your language!
- … to which you say, "awesome, let's do it!"

Grab the code

- Binary JAR through Maven/Ivy/SBT:
  - groupId=org.dynalang artifactId=dynalink
- Binary JAR directly from:
  - http://repo1.maven.org/org/dynalang/dynalink/0.2/dynalink-0.2.jar
- Source:
  - https://github.com/szegedi/dynalink

Documentation, documentation, documentation

- JavaDoc:
  - http://szegedi.github.com/dynalink/javadoc/index.html
- User's guide:
  - https://github.com/szegedi/dynalink/wiki

You need to implement one interface with one method.

You need to implement one interface with one method.

(I’m making this too easy for you.)

GuardingDynamicLinker

GuardingDynamicLinker

- Linker: it provides method handles to link call sites

GuardingDynamicLinker

- Linker: it provides method handles to link call sites
- Dynamic: it links at run time. Actually, invocation time.

GuardingDynamicLinker

- Linker: it provides method handles to link call sites
- Dynamic: it links at run time. Actually, invocation time.
- Guarding: provides means to invalidate the call site and trigger relinking.

GuardingDynamicLinker

```
GuardedInvocation getGuardedInvocation(
    LinkRequest linkRequest,
    LinkerServices linkerServices
)
```

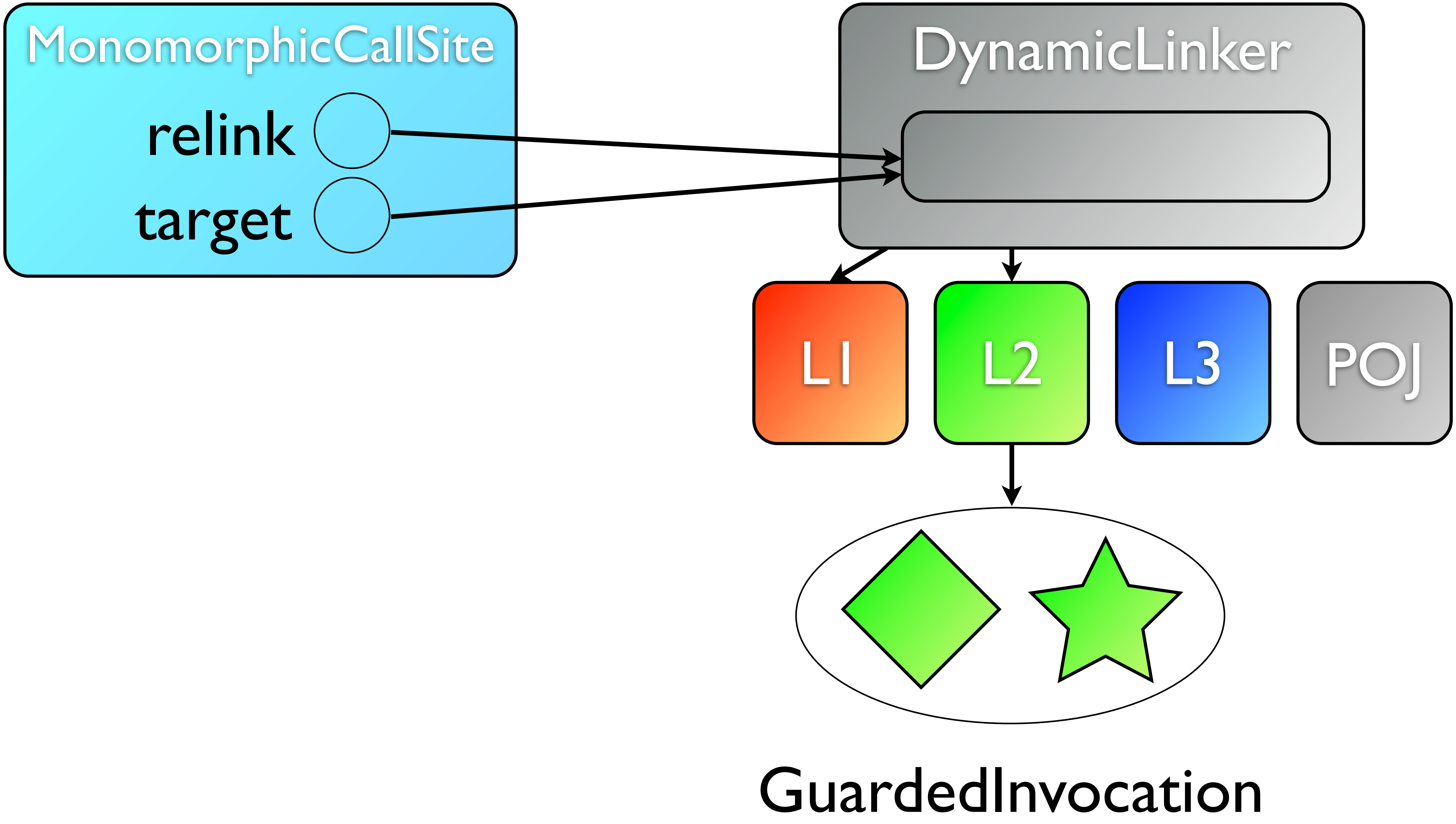
GuardingDynamicLinker

```
public class GuardedInvocation {
  private final MethodHandle invocation;
  private final MethodHandle guard;
  private final SwitchPoint  switchPoint;
  ...
}
```

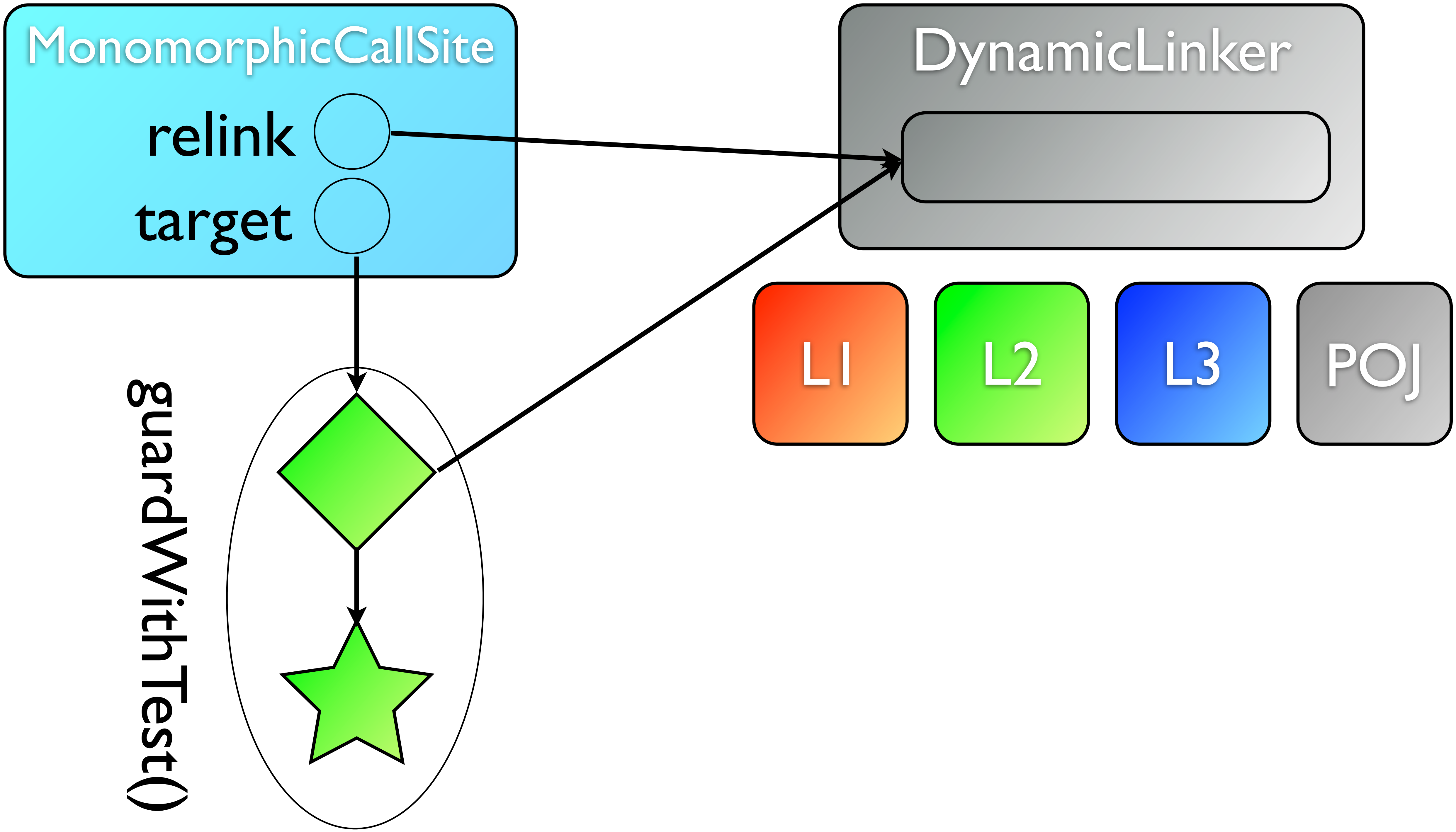
GuardingDynamicLinker

```
public interface LinkRequest {
  public CallSiteDescriptor getCallSiteDescriptor();
  public Object[] getArguments();
}
```

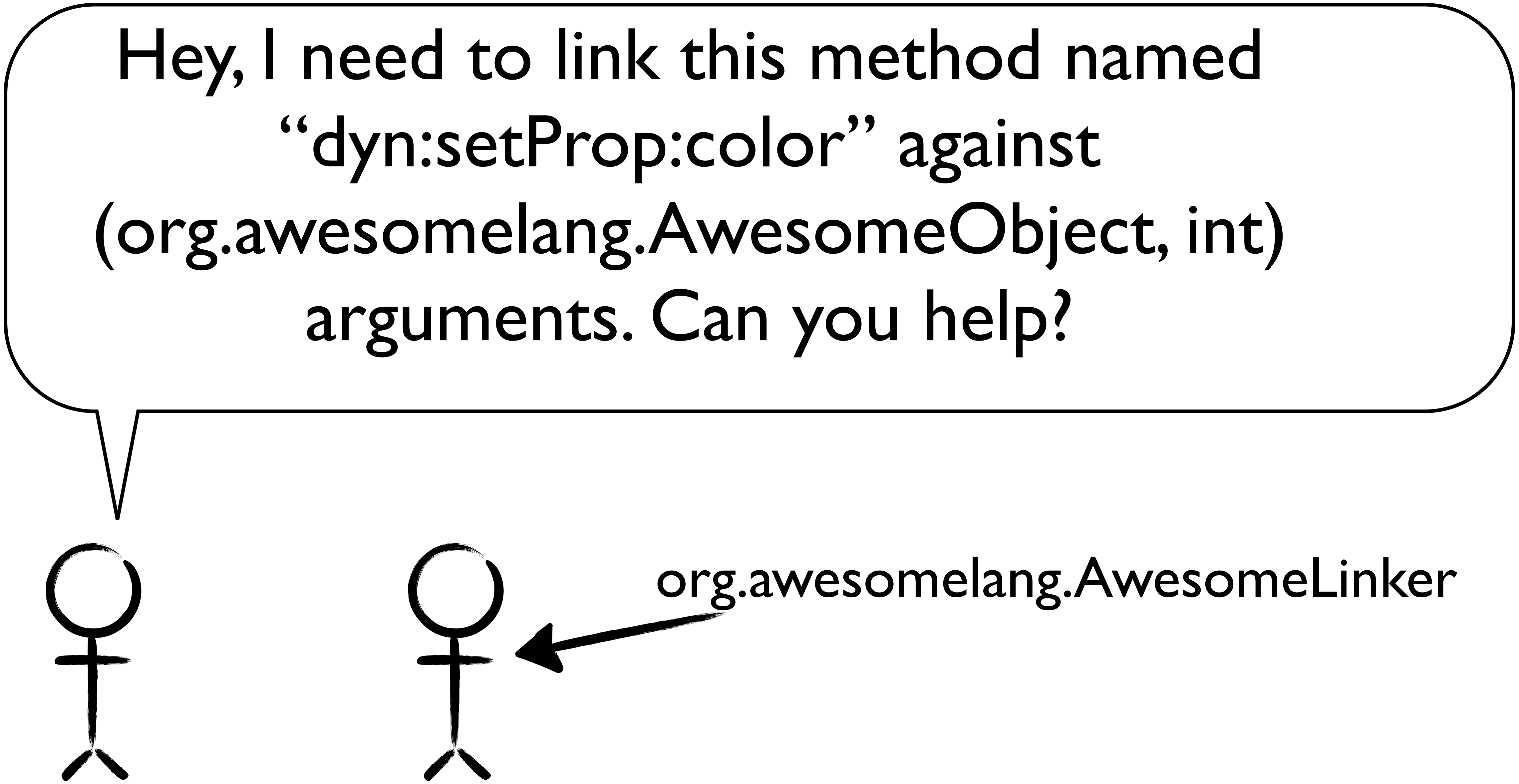
Linking a call site



Linking a call site



Linkers, how do they work?



Hey, I need to link this method named "dyn:setProp:color" against (org.awesomelang.AwesomeObject, int) arguments. Can you help?

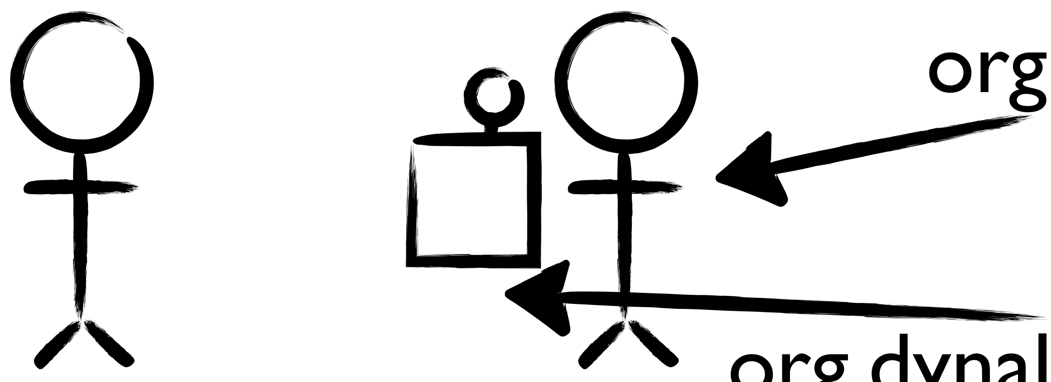
org.awesomelang.AwesomeLinker

Linkers, how do they work?

Sure, here’s a MethodHandle (“colors an AwesomeObject when invoked…”)

…and a guard MethodHandle (“…as long as it’s an AwesomeObject…”)

…and a SwitchPoint (“…and as long it’s the right way to color an AwesomeObject.”)



org.awesomelang.AwesomeLinker

org.dynalang.dynalink.linker.GuardedInvocation

Linker/CallSite concern separation

- Linker finds the method and specifies guard and/or switchpoint
- RelinkableCallSite implementation decides how to compose them
    - MonomorphicCallSite included

Linker implementation constraints

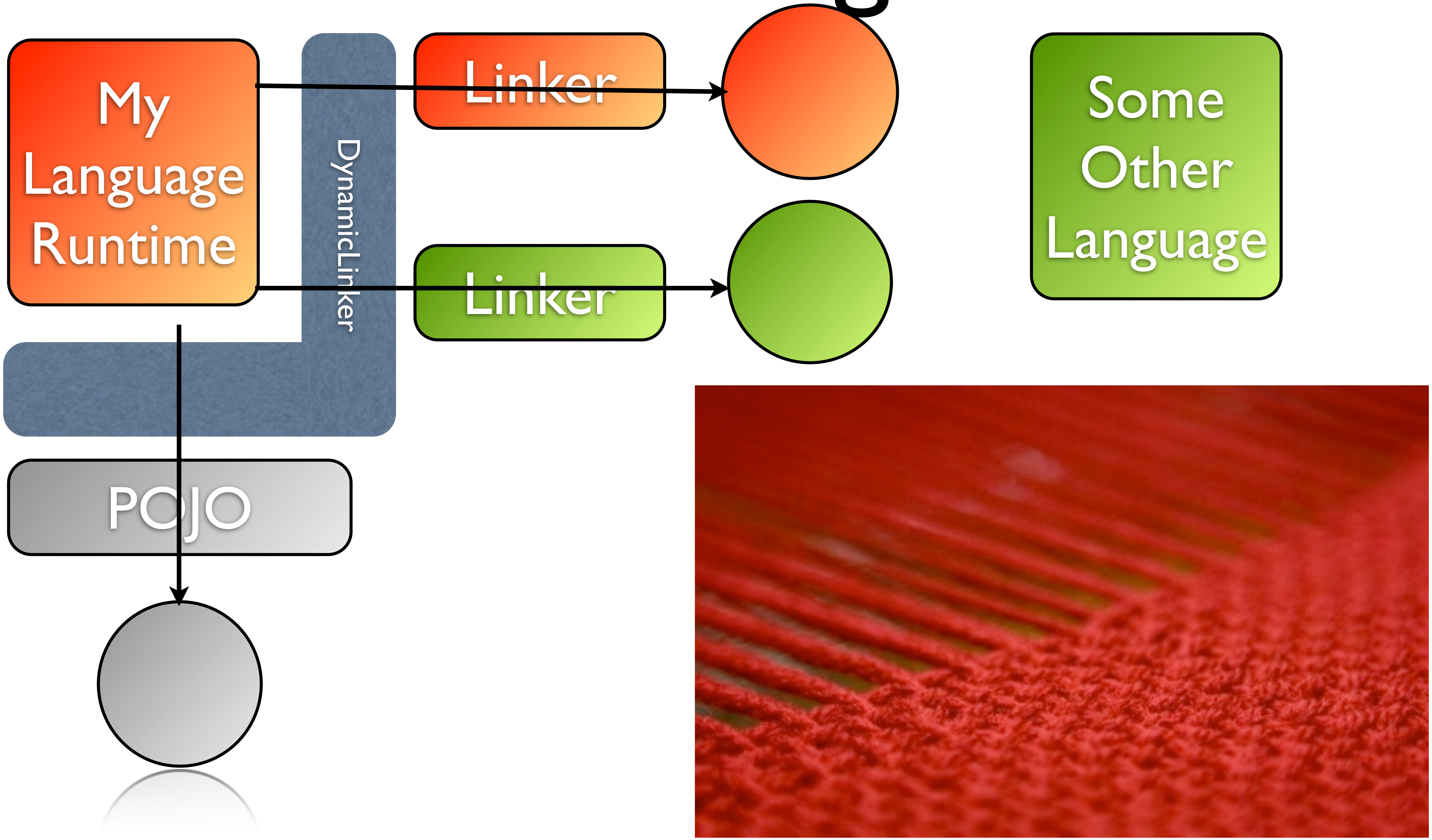
- There are really none…
- You can have switchpoint-invalidated metaclasses in the background if you want

Let them find you

- Now you have your linker, you still need to make sure other language runtimes can find it in the classpath.
- Name your linker class in

```
META-INF/services/
 org.dynalang.dynalink.linker.GuardingDynamicLinker
```

Unified API for object handling



Some more advanced concepts

Give yourself priority

- Your code manipulates native objects most of the time.
- Therefore, you want your language linker to enjoy priority for linking your own call sites.
- There's a way to do that.

Customized Linker Factory

```
import org.dynalang.dynalink.*;

public class AwesomeBootstrapper {
  private static final DynamicLinker dynamicLinker;
  static {
    final DynamicLinkerFactory factory = new DynamicLinkerFactory();
    final GuardingDynamicLinker awesomeLanguageLinker = new AwesomeLanguageLinker();
    factory.setPrioritizedLinker(awesomeLanguageLinker);
    dynamicLinker = factory.createLinker();
  }

  public static CallSite bootstrap(MethodHandles.Lookup caller, String name, MethodType type) {
    final MonomorphicCallSite callSite = new MonomorphicCallSite(caller, name, type);
    dynamicLinker.link(callSite);
    return callSite;
  }
```

The modified ASM example

```
    import org.objectweb.asm.MethodHandle;
import static org.objectweb.asm.Opcodes.MH_INVOKESTATIC;
...
private static final MethodHandle BOOTSTRAP =
    new MethodHandle(MH_INVOKESTATIC,
        "org/awesomelang/AwesomeBootstrapper","bootstrap",
        MethodType.methodType(CallSite.class, MethodHandles.Lookup.class,
            String.class, MethodType.class).toMethodDescriptorString());
private static final Object[] BOOTSTRAP_ARGS = new Object[0];
...
mv.visitIndyMethodInsn("dyn:setProp:color",
    "(Ljava/lang/Object;I)V",
    BOOTSTRAP, BOOTSTRAP_ARGS);
mv.visitIndyMethodInsn("dyn:getProp:shape", "(Ljava/lang/Object;)Ljava/lang/String;",
    BOOTSTRAP, BOOTSTRAP_ARGS);
```

Type-based linker

- Your linker can declare it’s the authority for linking certain types.
- DynamicLinkerFactory can create optimized type-based composite linkers

Type-based linker

```
public class AwesomeLinker implements TypeBasedGuardingDynamicLinker {
    public boolean canLinkType(Class<?> type) {
        return AwesomeObject.class.isAssignableFrom(type);
    }

    public GuardedInvocation getGuardedInvocation(LinkRequest linkRequest,
        LinkerService linkerServices)
    {
        // Can assume there's at least one arg, and it's of the required type
        AwesomeObject target = (AwesomeObject)linkRequest.getArguments()[0];
        ...
    }
}
```

Implicit type conversions

- Does your language allows an implicit Object-to-boolean conversion?
  - Yeah, it isn’t necessarily a good idea.
  - But if it does, there’s an interface for that!

Implicit type conversions

```
public class AwesomeLinker
implements TypeBasedGuardingDynamicLinker, GuardingTypeConverterFactory {
    private static final MethodHandle CONVERT_AWESOME_TO_BOOLEAN = ...;
    private static final GuardedInvocation GUARDED_CONVERT_AWESOME_TO_BOOLEAN =
        new GuardedInvocation(
            CONVERT_AWESOME_TO_BOOLEAN,
            Guards.isInstance(AwesomeObject.class, CONVERT_AWESOME_TO_BOOLEAN.type());

    public GuardedInvocation convertToType(Class<?> sourceType, Class<?> targetType) {
        if(AwesomeObject.class.isAssignableFrom(sourceType) && Boolean.class == targetType) {
            return GUARDED_CONVERT_AWESOME_TO_BOOLEAN;
        }
        return null;
    }

    ...
}
```

Private linking

- Rhino: need customized linking for Double, Boolean, String for call sites in its own code.
- This semantics however shouldn't be exposed to other language runtimes.

Private linking

```
import org.dynalang.dynalink.*;

public class RhinoBootstrapper {
  private static final DynamicLinker dynamicLinker;
  static {
    final DynamicLinkerFactory factory = new DynamicLinkerFactory();
    final GuardingDynamicLinker rhinoPrimitiveLinker = new RhinoPrimitiveLinker();
    final GuardingDynamicLinker rhinoLinker = new RhinoLinker();
    factory.setPrioritizedLinkers(rhinoPrimitiveLinker, rhinoLinker);
    dynamicLinker = factory.createLinker();
  }

  public static CallSite bootstrap(MethodHandles.Lookup caller, String name, MethodType type) {
     ...
  }
```

Runtime context args

- Many language runtimes pass context/ thread specific information on stack
- Support for automatic stripping of these across language boundaries
- Runtimes should make sure these are still accessible in thread-locals.

Things still to do

- POJO linker should:
  - Use Java access rules at call site to allow linking non-public members
  - Expose constructors somehow
  - link to field getter/setter for accessible fields
  - standardized iterator and block support

Dynalink

https://github.com/szegedi/dynalink

Attila Szegedi
@asz