In bed with ...

Rémi Forax

ParisJUG - sept 2013

Me, myself and I

Rémi Forax
UPEM (University Paris-East Marne-la-Vallée)

JCP.org

- JSR 292 (invokedynamic)
- JSR 335 (lambda)

OpenJDK

- Project lambda

ASM, Tatoo, etc.


DA JAVA
CODE


Beginning
Java EE 6 Platform
with GlassFish 3
From Novice to Professional

Java™

EVIL EDITION

JAVA EE

Java EE impls need love too

Modularity

- Java EE spec is modular
- Java EE implementations are modular
  but within themselves

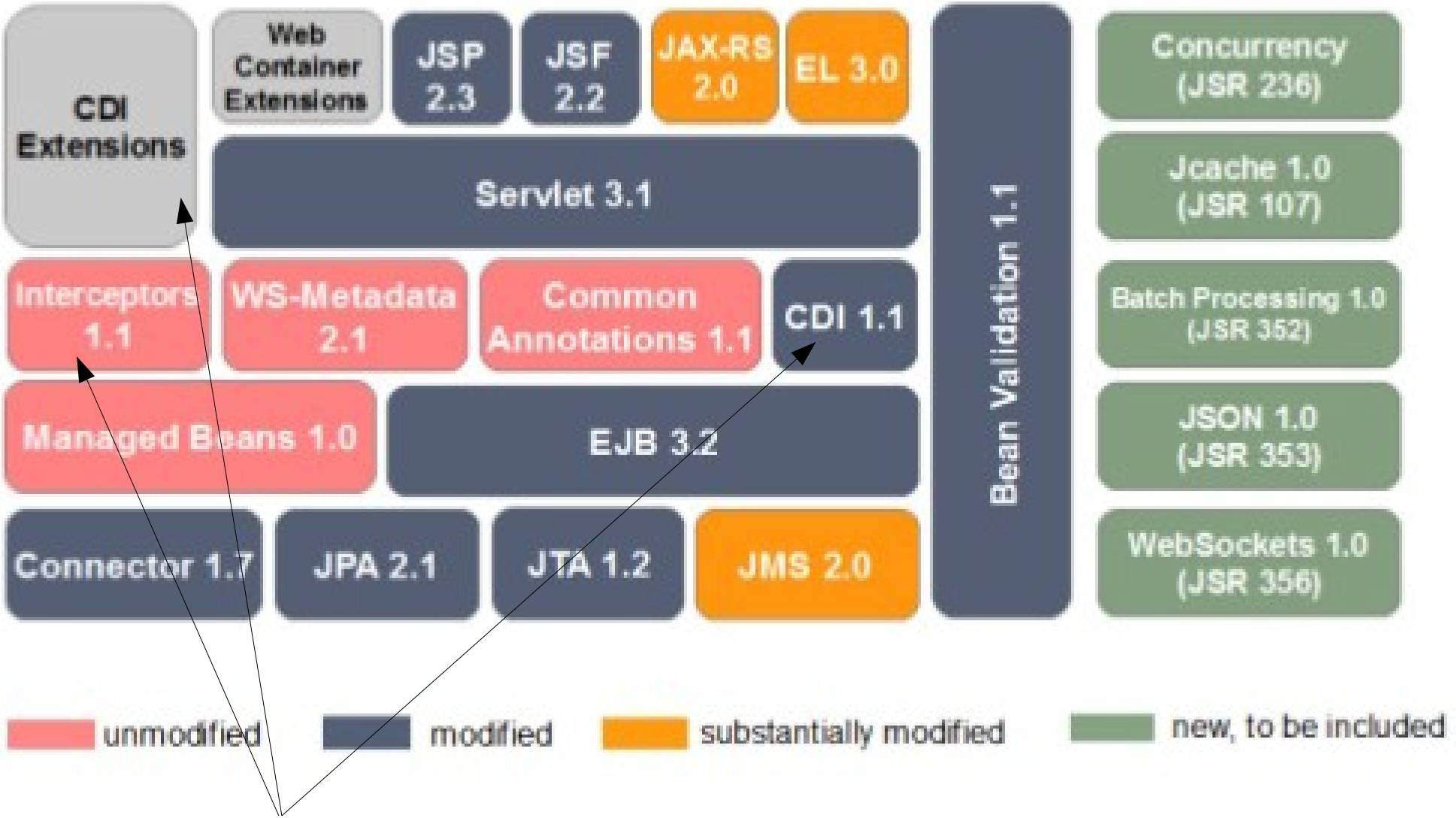
    The way annotations are linked to an implementation is JEE implementation specific

  => no way to use JBoss Foo with Spring Bar

=> so the ecosystem can not evolve incrementally

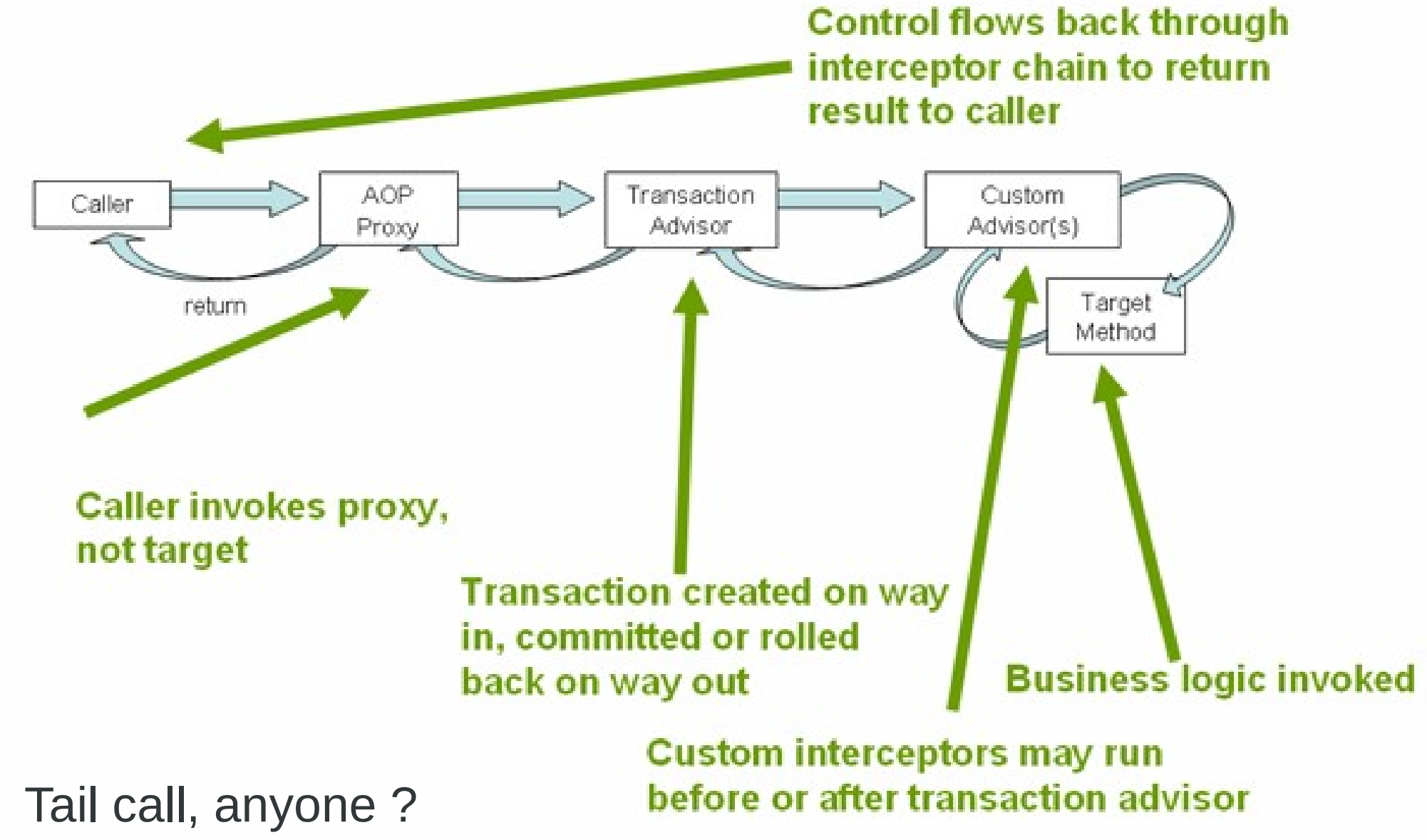
=> it raises the bar for any new implementations

Java EE architecture (not the real one BTW)



It's the same thing, no ?

AOP is the backbone



Tail call, anyone ?

Java EE impls use Proxy

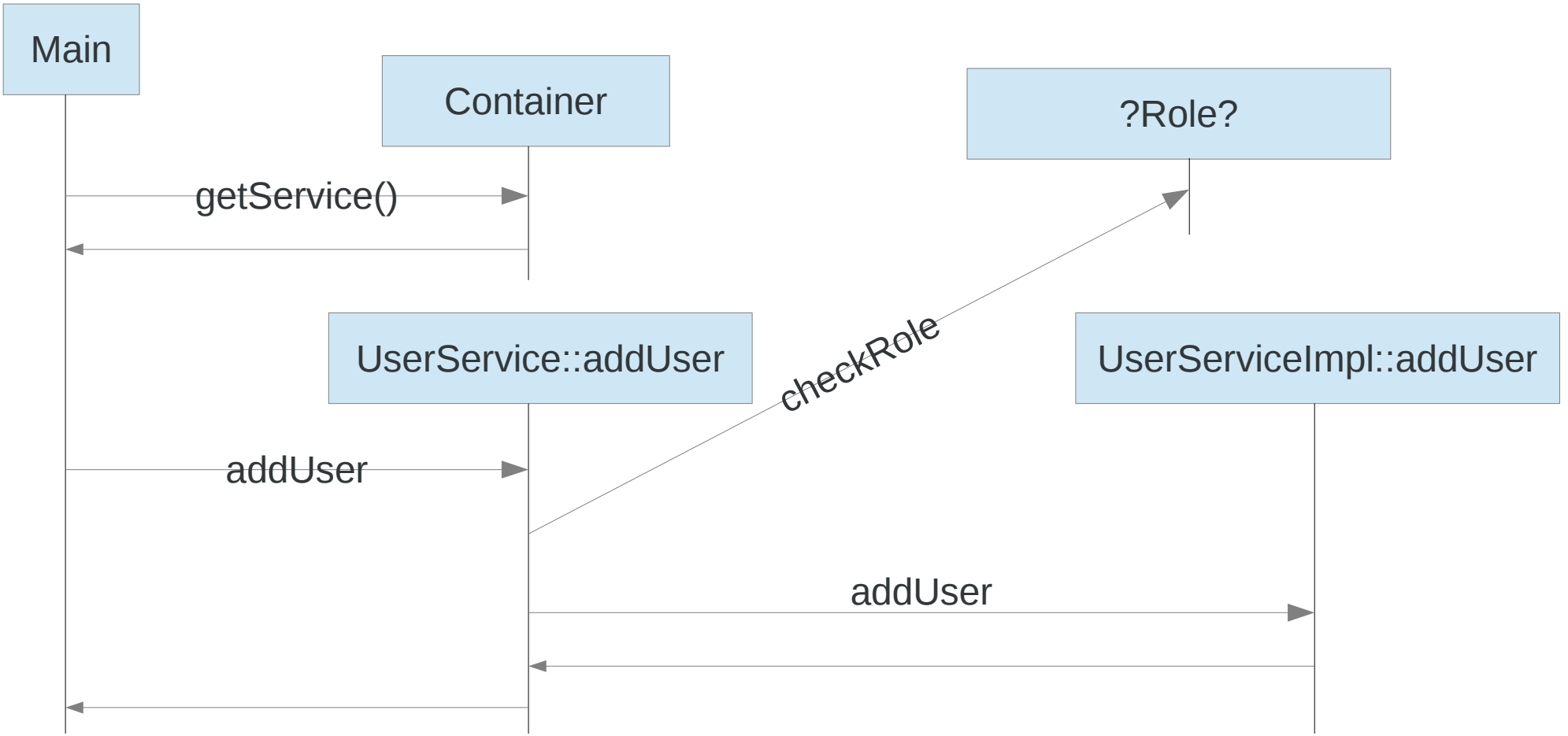
A lot of issues

- Create an inside/ouside
  - Divorce with Java SE
- 2 objects when one is enough
  - All dynamic languages use one object (Ror, ...)
- Looong stack traces

=> Does the really VM like proxies ?

An example

Let see on a small example !



EnsureRole meta protocol

Declaration:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface EnsureRole {
  String value() default "user";
}
```

Use:

```
public interface UserService {
  @EnsureRole("manager")
  public void addUser(String userName,
                      String userMailAddress, boolean admin);
}
```

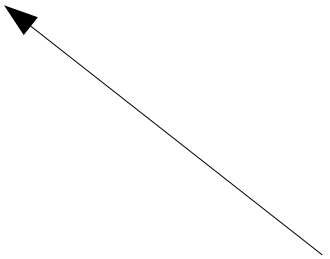
Main

Configuration should be done in Java (no XML !)

```
public static void main(String[] arguments) {
    Container container = new Container();
    container.addAdvice(...);

    UserService userService = container.getService(
                UserService.class, UserServiceImpl.class);

    userService.addUser("Darth Vador",
                "1 Force Street, Death Star", true);
  }
}
```

UserServiceImpl::addUser

```
public class UserServiceImpl implements UserService {
  private static int COUNTER;


  @Override
  public void addUser(String userName,
                      String userMailAddress, boolean admin) {
    // add a new user to the database
    // send a mail to invite the new user
    COUNTER++;
  }
}
```



Just because I'm afraid that the VM may do nothing if the body is empty

Advice/AdviceContext

Define a chain advice that will check if an annotation exist and call the implementation

```
Container container = new Container();
container.addAdvice(
  (AnnotatedElement ae, Object r, Object[] as,
   AdviceContext ac) -> {
      EnsureRole er = ae.getAnnotation(EnsureRole.class);
      if (er != null) {
        checkRole(ae, er);
      }
      return ac.call(ae, r, as);
  });
```

Container::addAdvice

```
AdviceContext context = new AdviceContextImpl(
  (AnnotatedElement ae, Object r, Object[] as, AdviceContext ac) -> {
    try {
      return ((Method)ae).invoke(r, as);
    } catch (...) {
    }
  }, null);

public void addAdvice(Advice a) {
  context = new AdviceContextImpl(advice, context);
}

static class AdviceContextImpl implements AdviceContext {
  private final Advice advice;
  private final AdviceContext next;
  public Object call(AnnotatedElement ae, Object e, Object[] as) {
    return advice.chain(ae, r, as, next);
  }
}
```

Container::getService

```
UserService userService = container.getService(UserService.class,
                                               UserServiceImpl.class);

public <S> S getService(Class<S> sItf, Class<? extends S> sImpl) {
  S impl;
  try {
    impl = sImpl.newInstance();
  } catch (...) { ... }
  class ServiceInvocationHandler implements InvocationHandler {
    public Object invoke(Object proxy, Method m, Object[] as) ... {
      return adviceContext.call(m, impl, as);
    }
  }
  return sIntf.cast(
    Proxy.newProxyInstance(sItf.getClassLoader(),
      new Class<?>[] { sItf },
      new ServiceInvocationHandler()));
}
```

Ask the VM !

PrintCompilation (product)

- Print hot method and

PrintInlining (diagnostic UnlockDiagnosticVMOptions)

- Print inlining tree and class profile

LogCompilation (diagnostic)

- More info, branch profile info, etc

PrintAssembly (diagnostic + hsdis.so)

- Annotated generated assembly code

more stable between executions:
-Xbatch (one thread), -XX:-TieredCompilation

Time to switch to Eclipse !

```
com.sun.proxy.$Proxy0::addUser (47 bytes)
  @ 23   java.lang.Boolean::valueOf (14 bytes)   inline (hot)
  @ 27   proxy.Container$1ServiceInvocationHandler::invoke (17 bytes)   inline (hot)
    \-> TypeProfile (6700/6700 counts) = proxy/Container$1ServiceInvocationHandler
     @ 13   proxy.Container$AdviceContextImpl::call (17 bytes)   inline (hot)
       @ 11   proxy.Container$$Lambda$1::chain (9 bytes)   inline (hot)
       @ 11   proxy.Main$$Lambda$2::chain (9 bytes)   inline (hot)
        \-> TypeProfile (3350/6701 counts) = proxy/Main$$Lambda$2
        \-> TypeProfile (3351/6701 counts) = proxy/Container$$Lambda$1
          @ 5   proxy.Main::lambda$1 (34 bytes)   inline (hot)
            @ 3   java.lang.reflect.Method::getAnnotation (6 bytes)   inline (hot)
              ...
            @ 21   proxy.Main::checkRole (1 bytes)   inline (hot)
          @ 5   proxy.Container::lambda$0 (69 bytes)   too big

proxy.Container::lambda$0 (69 bytes)
  @ 6   java.lang.reflect.Method::invoke (62 bytes)   inline (hot)
      ...
        @ 65   java.lang.Boolean::booleanValue (5 bytes)   accessor
        @ 79   proxy.UserServiceImpl::addUser (9 bytes)   inline (hot)

proxy.Main::main @ 31 (55 bytes)
  @ 43   com.sun.proxy.$Proxy0::addUser (47 bytes)   already compiled big method
   \-> TypeProfile (11264/11264 counts) = com/sun/proxy/$Proxy0
```

Oh no !

The VM doesn't like proxies too !

Proxies

- Box arguments (+ array)
  - slooooow !
- Use reflection to call the target method
  - Fast if no security manager
  - generate code at runtime :(

Usually too complex to be fully inlined :(

- no de-virtualization
- boxing not removed
- service code not specialized with calling data

It's invokedynamic stupid !

We have spent 5 years to specify how to do method calls for dynamic languages

- invokedynamic & method handle

JEE impls are like dynamic language runtime

- Method call with a specific meta protocol

The VM knowns how to optimize invokedynamic and method handles well

Interceptable

Modify Java the language !

Add a keyword interceptable on class, method or field

javac generates an invokedynamic when an interceptable member is called

```
public interceptable interface UserService {
    @EnsureRole("manager")
    public void addUser(String userName,
                String userMailAddress, boolean admin);
}
```

Calling addUser in bytecode

```
UserService userService = ...
userService.addUser("Darth Vador",
    "1 Force Street, Death Star", true);
```

```
38: aload_1
39: ldc           #11  // String Darth Vador
41: ldc           #12  // String 1 Force Street, ...
43: iconst_1
44: invokedynamic #231,  0
  // addUser(UserService;String;String;Z)V
  // Container::bootstrap()
```

Container::bootstrap

```
public static CallSite bootstrap(Lookup lookup,
  String name,   MethodType methodType, MethodHandle impl) {
    AnnotatedElement ae = Magic.reflect(impl);
    MethodHandle mh = impl;
    for(Advice advice: advices) {
      mh = advice.chain(ae, mh);
    }
    return new ConstantCallSite(mh);
}

private static final ArrayList<Advice> advices = new ArrayList<>();

public static void addAdvice(Advice interceptor {
  advices.add(interceptor);
}
```

Advice with MethodHandle

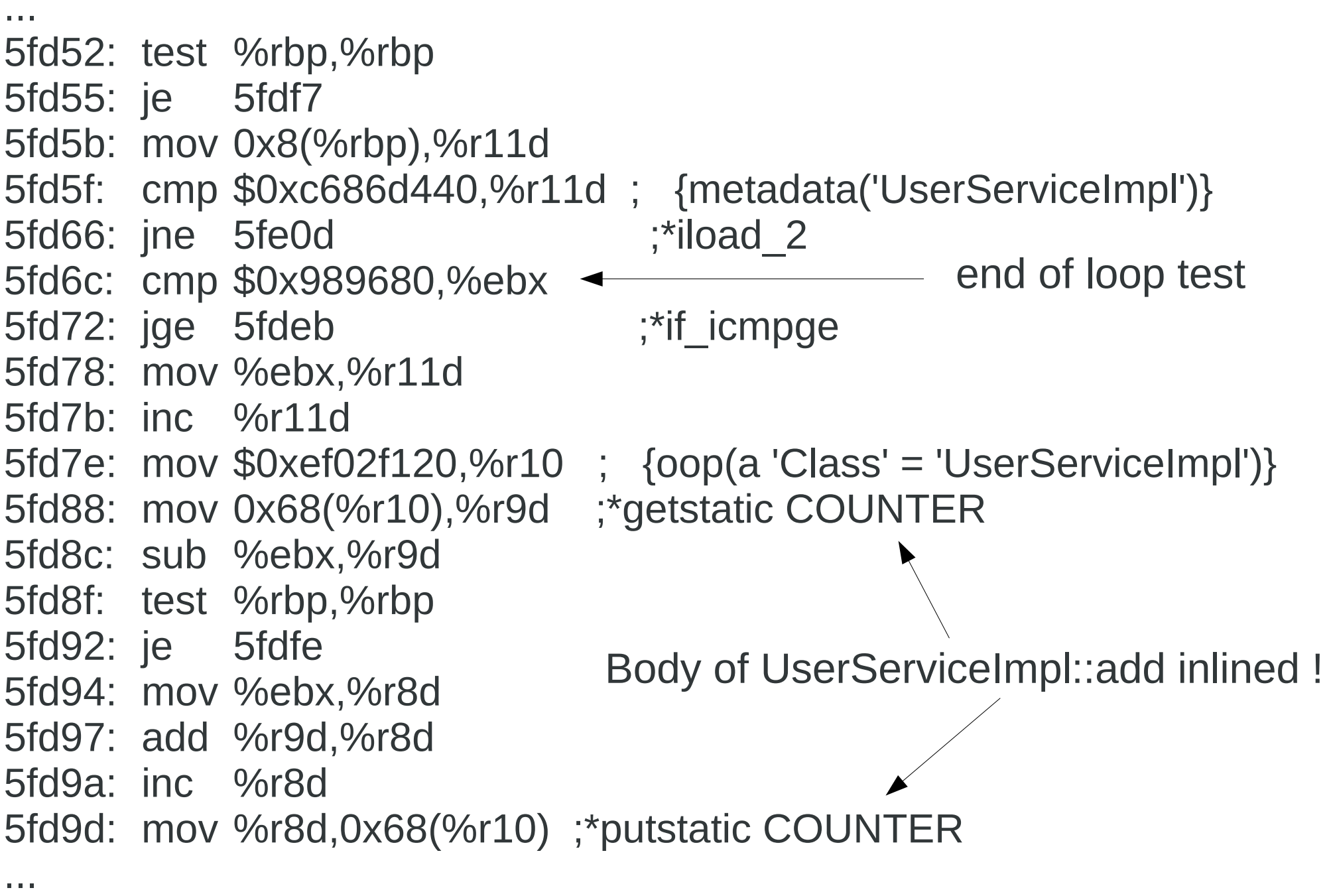
```
Container.addAdvice(
  (AnnotatedElement ae, MethodHandle mh) -> {
    EnsureRole er = ae.getAnnotation(EnsureRole.class);
    if (er == null) {
      return mh;
    }
    MethodHandle cb = MethodHandles.insertArguments(
       CHECK_ROLE, 0, ae, er);
    return MethodHandles.foldArguments(mh, combiner);
  });

// no proxy !
UserService userService = new UserServiceImpl();
```

Time to switch to Eclipse again !

```
java9.Main::main @ 18 (42 bytes)
  @ 30   j.l.i.LambdaForm$MH/558638686::linkToCallSite (20 bytes)   inline (hot)
    @ 2   jj.l.i.Invokers::getCallSiteTarget (8 bytes)   inline (hot)
      @ 4   j.l.i.ConstantCallSite::getTarget (20 bytes)   inline (hot)
    @ 16   j.l.i.LambdaForm$MH/2074407503::collect (30 bytes)   inline (hot)
      @ 5   j.l.i.LambdaForm$BMH/999966131::reinvoke (34 bytes)   inline (hot)
        @ 20   j.l.i.BoundMethodHandle$Species_LLL::reinvokerTarget (8 bytes)   inline (hot)
        @ 30   j.l.i.LambdaForm$DMH/644117698::invokeStatic_LL_V (15 bytes)   inline (hot)
          @ 1   j.l.i.DirectMethodHandle::internalMemberName (8 bytes)   inline (hot)
          @ 11   java9.Main::checkRole (1 bytes)   inline (hot)
      @ 26   j.l.i.LambdaForm$DMH/1746572565::invokeInterface_LLLI_V (20 bytes)   inline (hot)
        @ 1   j.l.i.DirectMethodHandle::internalMemberName (8 bytes)   inline (hot)
        @ 16   java9.UserServiceImpl::addUser (9 bytes)   inline (hot)
```

HOORAY

=> FULL INLINE !


...
5fd52: test %rbp,%rbp
5fd55: je 5fdf7
5fd5b: mov 0x8(%rbp),%r11d
5fd5f: cmp $0xc686d440,%r11d ; {metadata('UserServiceImpl')}
5fd66: jne 5fe0d ;*iload_2
5fd6c: cmp $0x989680,%ebx
end of loop test
5fd72: jge 5fdeb ;*if_icmpge
5fd78: mov %ebx,%r11d
5fd7b: inc %r11d
5fd7e: mov $0xef02f120,%r10 ; {oop(a 'Class' = 'UserServiceImpl')}
5fd88: mov 0x68(%r10),%r9d ;*getstatic COUNTER
5fd8c: sub %ebx,%r9d
5fd8f: test %rbp,%rbp
5fd92: je 5fdfe
Body of UserServiceImpl::add inlined !
5fd94: mov %ebx,%r8d
5fd97: add %r9d,%r8d
5fd9a: inc %r8d
5fd9d: mov %r8d,0x68(%r10) ;*putstatic COUNTER
...

So ...

JEE impls should use invokedynamic

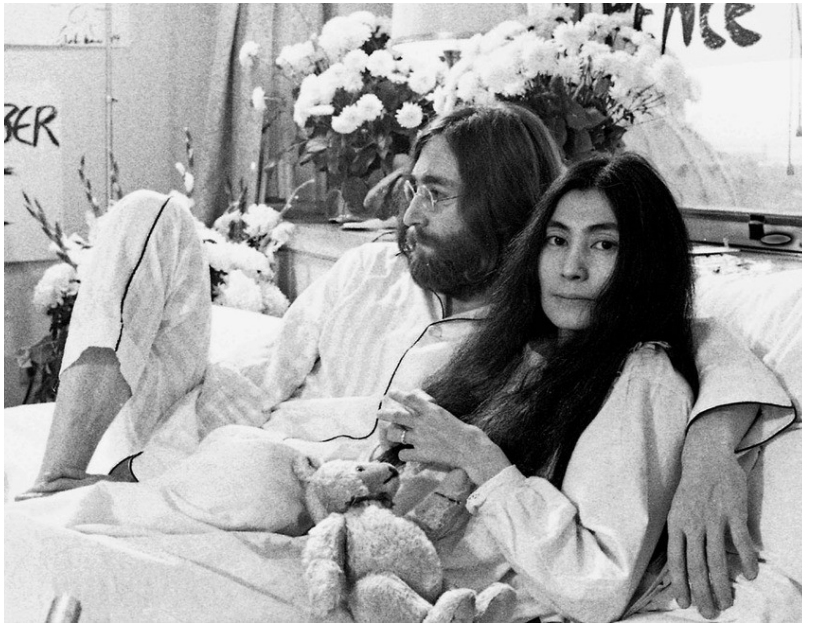
But this requires to wait Java 9 :(

to have interceptable in the language

In between, we can use bytecode rewriting tool

- rewrite the bytecode when packaging the war/ear/... ?
- or use an agent at runtime

Summary



- JavaEE impls use AOP
  - and rely on proxy
    - We don't like proxy, the VM don't like it too !
- Introduce interceptable/ghost in Java 9
  - Anybody can provide an implementation for an annotation
    - A project can use a specific annotation, ...
  - Java EE will be
    - More modular, simpler, faster and still typesafe