in search of **aop** for **as3**

Maxim Porges

HIGHWINDS

what is **aop**?

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Number
        {
            return (numberOne + numberTwo);
        }

        public function divide(numerator : Number, denominator : Number) : Number
        {
            return (numerator / denominator);
        }

         public function multiply(numberOne : Number, numberTwo : Number) : Number
         {
            return (numberOne * numberTwo);
         }
    }
}
```

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Number
        {
            return (numberOne + numberTwo);
        }

        public function divide(numerator : Number, denominator : Number) : Number
        {
            return (numerator / denominator);
        }

        public function multiply(numberOne : Number, numberTwo : Number) : Number
        {
            return (numberOne * numberTwo);
        }
    }
}
```

i wonder how long it takes each of my methods to execute?

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne + numberTwo);
            trace("add took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function divide(numerator : Number, denominator : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numerator / denominator);
            trace("divide took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function multiply(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne * numberTwo);
            trace("multiply took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }
    }
}
```

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne + numberTwo);
            trace("add took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function divide(numerator : Number, denominator : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numerator / denominator);
            trace("divide took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function multiply(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne * numberTwo);
            trace("multiply took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }
    }
}
```

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Numb
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne + numberTwo);
            trace("add took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function divide(numerator : Number, denominator : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numerator / denominator);
            trace("divide took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function multiply(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne * numberTwo);
            trace("multiply took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }
    }
}
```

wow, this is repetitive

```
package amazing.code
{
    public class AwesomeClass
    {
        public function AwesomeClass() {  }

        public function add(numberOne : Number, numberTwo : Number) : Numb
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne + numberTwo);
            trace("add took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function divide(numerator : Number, denominator : Number)
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numerator / denominator);
            trace("divide took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }

        public function multiply(numberOne : Number, numberTwo : Number) : Number
        {
            var startTime : Number = new Date().getTime();
            var result : Number = (numberOne * numberTwo);
            trace("multiply took " + (new Date().getTime() - startTime) + " ms");

            return result;
        }
    }
}
```

wow, this is repetitive

makes my code messy, too...

cross-cutting concerns

- logging
- method-level security
- transaction management
- not for...

✔ **keep the real code and the "advice" code separate**

cross-cutting concerns

- logging
- method-level security
- transaction management
- not for...

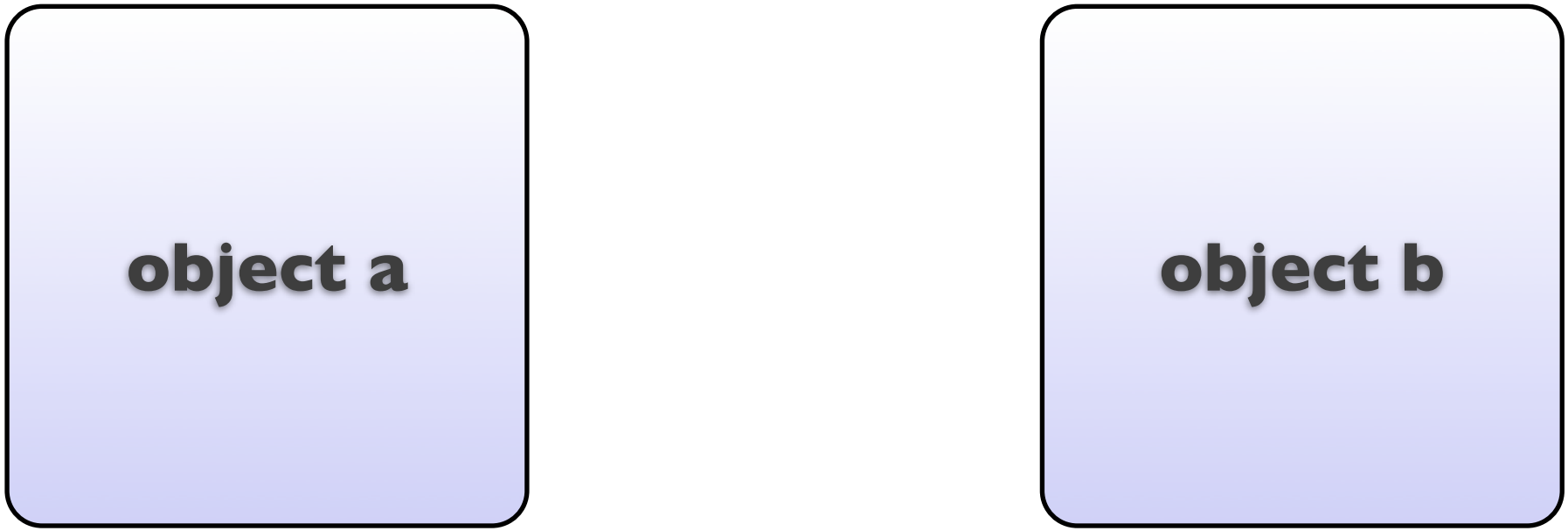
these are all "advice"

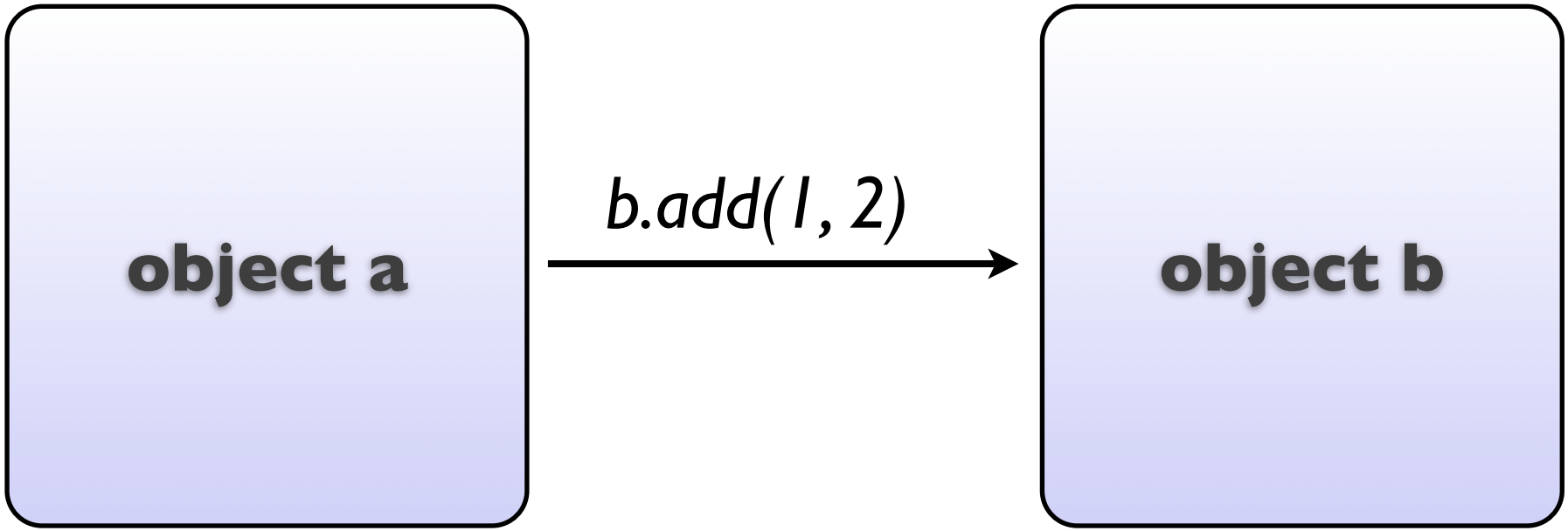
keep the real code and the "advice" code separate

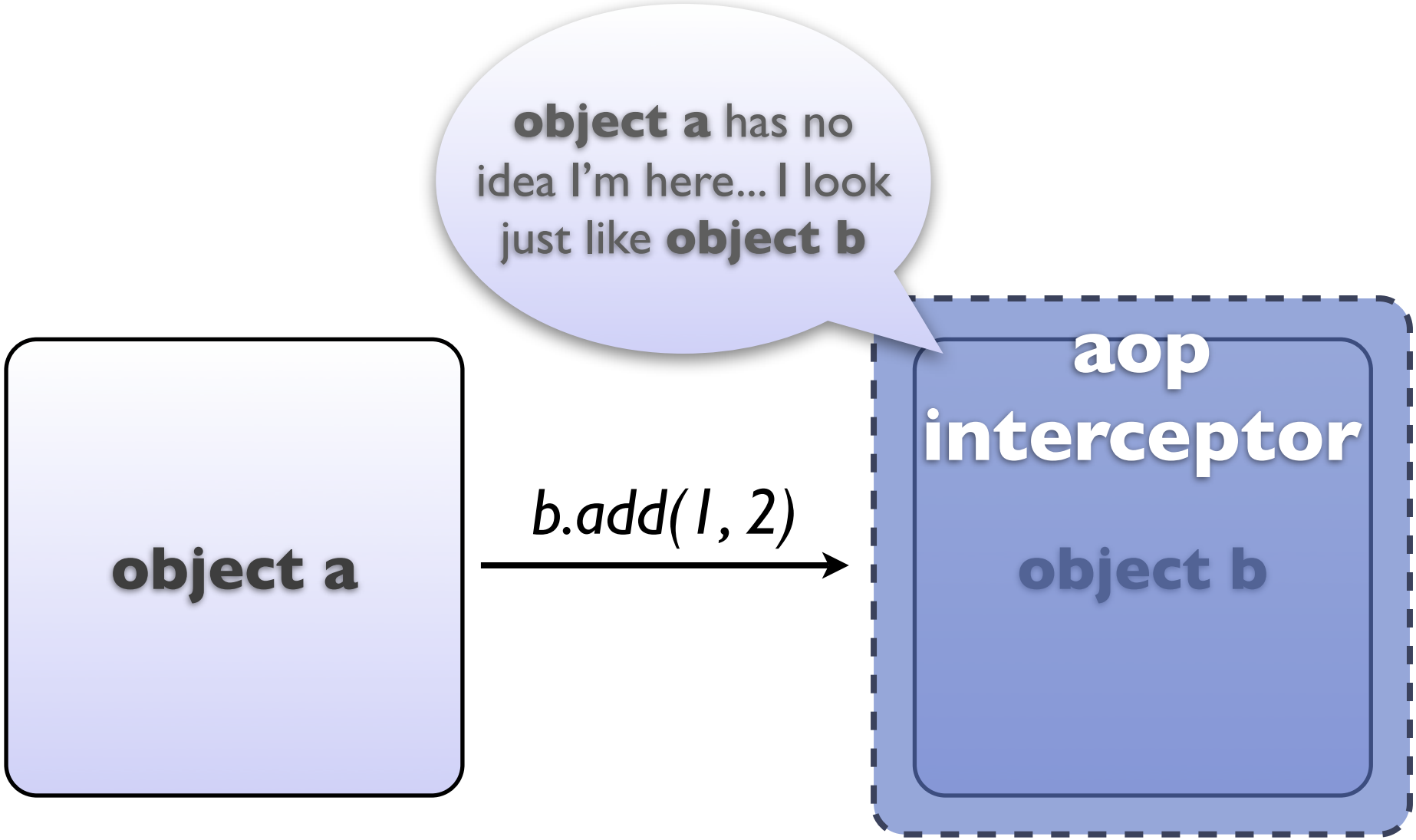
aop is...

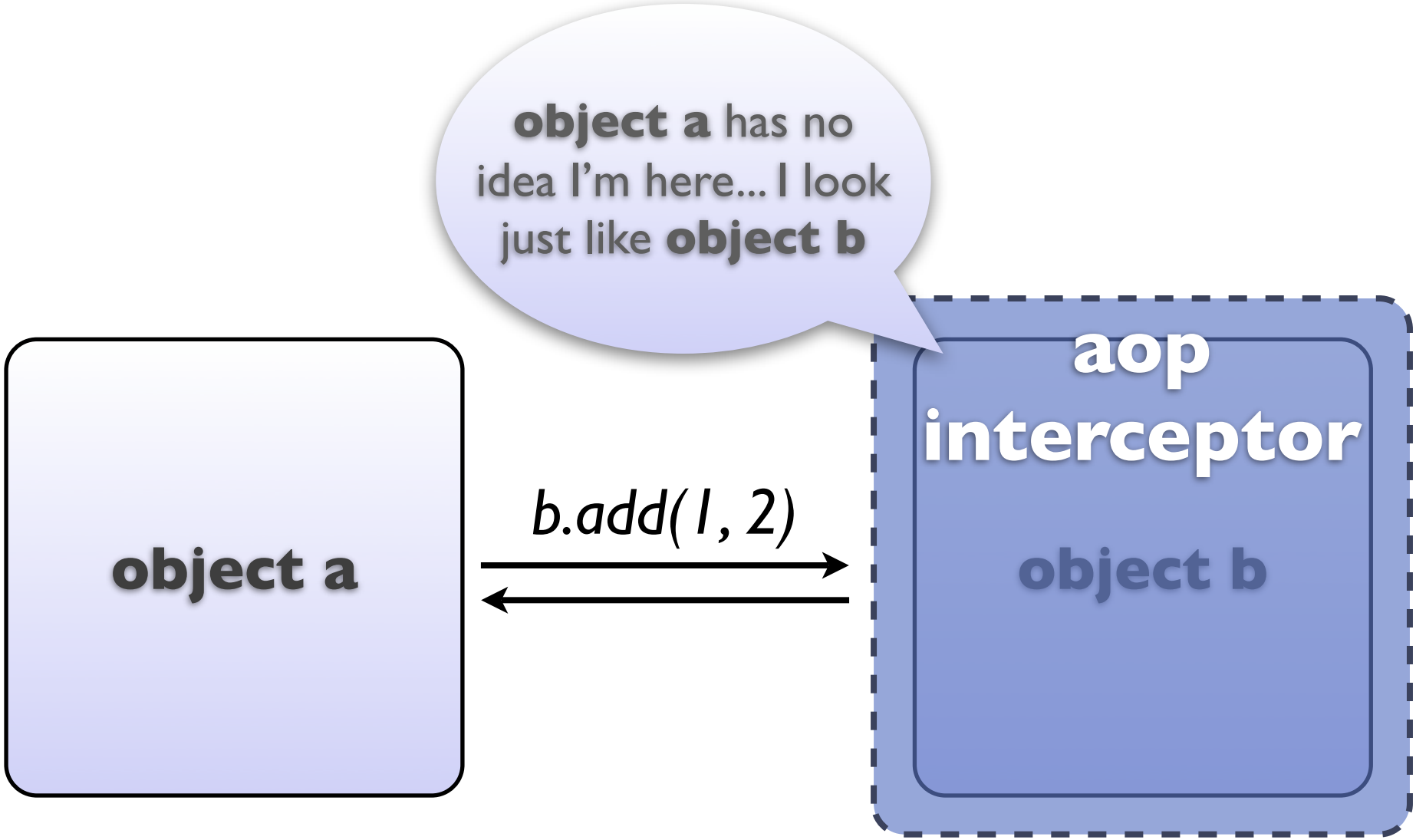
- a way to manage applying cross-cutting code wherever you want it
- aop lets you do this without having to scatter the code everywhere

how does **aop** work?


object a
object b


object a
b.add(1, 2)
object b


object a has no idea I'm here... I look just like object b
object a
b.add(1, 2)
aop interceptor
object b


object a has no idea I'm here... I look just like object b
object a
b.add(1, 2)
aop interceptor
object b

before advice

```
public function divide(a : Number, b : Number) : Number
{
    return (a / b);
}
```

after advice

```
public function divide(a : Number, b : Number) : Number
{
    return (a / b);
}
```

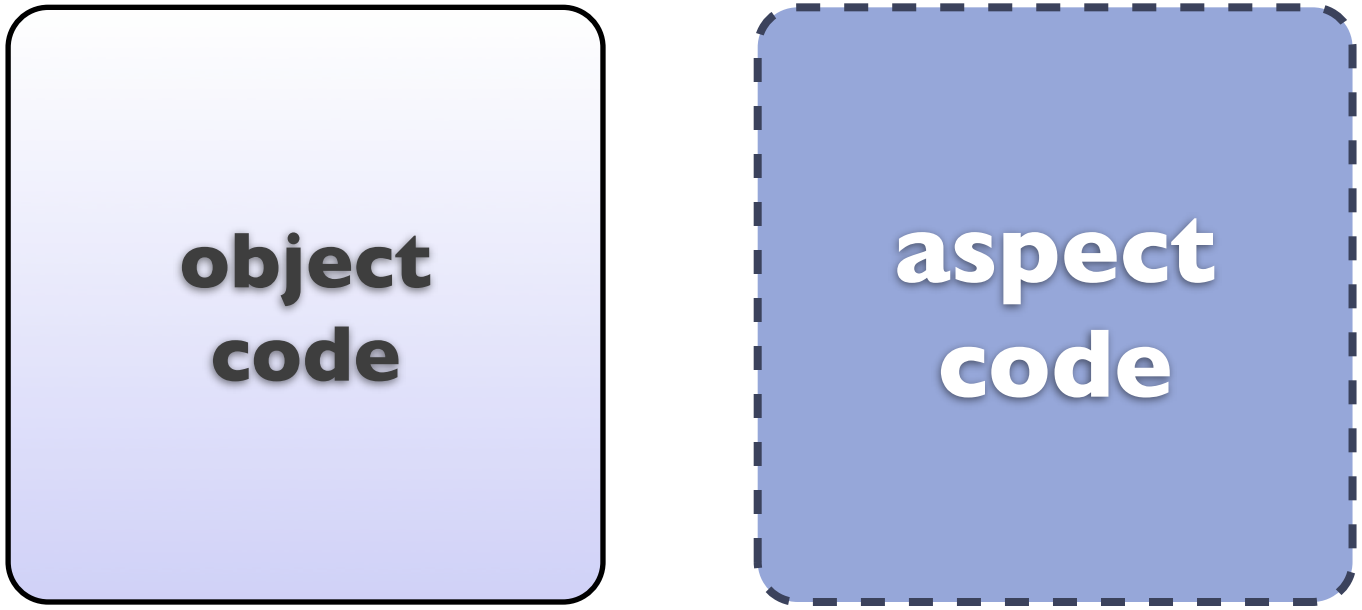
around advice

```
public function divide(a : Number, b : Number) : Number
{
   return (a / b);
}
```

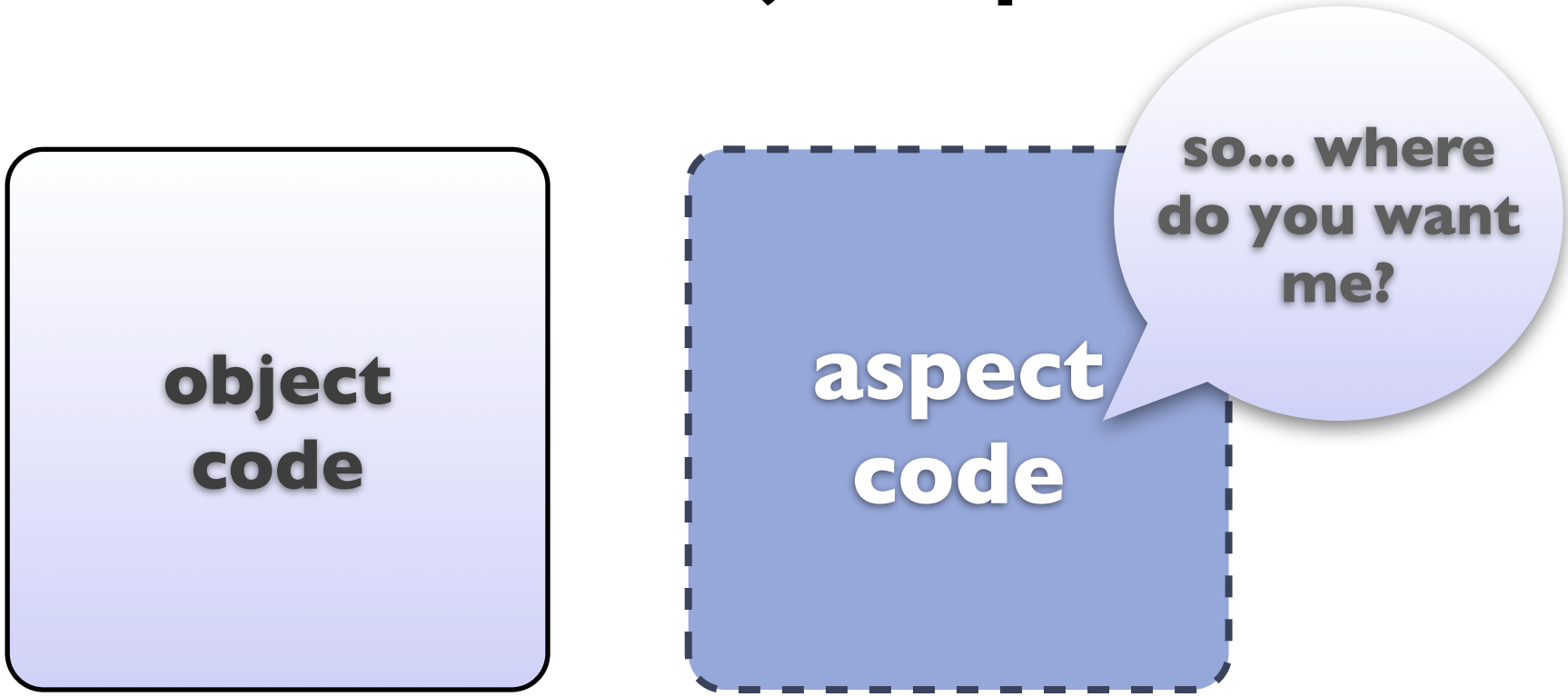
throws advice

```
public function divide(a : Number, b : Number) : Number
{
    throw new Error("Oops!");
}
```

pointcuts and joinpoints



pointcuts and joinpoints



a pointcut is...

- a kind of query that tells the aop framework where/how to apply the advice

```
@Around("call(* AwesomeClass.*(..))")
```

a joinpoint is...

- where to apply the advice
  - constructor
  - method call
  - exception flow
  - etc.

a logging example

a logging example

```java
package amazing.code;


public class AwesomeClass
{
   public AwesomeClass() { }

   public int add(int numberOne, int numberTwo)
   {
      return (numberOne + numberTwo);
   }

   public Double divide(Double numerator, Double denominator)
   {
      return (numerator / denominator);
   }

   public Double multiply(Double numberOne, Double numberTwo)
   {
      return (numberOne * numberTwo);
   }
}
```

```java
@Aspect
public class LoggingAspect
{
   @Around("call(* AwesomeClass.*(..))")
   public Object logMethodAccess(ProceedingJoinPoint joinPoint) throws Throwable
   {
      System.out.println(">>> A call is being made to " +
          joinPoint.getSignature() + " with arguments " +
          StringUtils.join(joinPoint.getArgs(), ", ")
      );

      long startTime = System.currentTimeMillis();
      try
      {
         return joinPoint.proceed();
      }
      finally
      {
         System.out.println(">>> Call to " +
             joinPoint.getSignature() + " took " +
             (System.currentTimeMillis() - startTime) + " ms"
         );
      }
   }
}
```

summary

- aop advice is added to your classes for you by an aop library (such as AspectJ)
- aop advice plugs in to existing code seamlessly
- several kinds of advice
- pointcuts introduce advice to joinpoints

aop in actionscript

the plan

- change the behavior of a class at runtime
- use method closures as advice
- aop proxy must be type-compatible
- non-dynamic (i.e. no performance hit)
- try using the tools that as3 provides out-of-the-box

thought #1: prototype

thought #2: strict=false

thought #2: strict=false

thought #3: Proxy

thought #3: Proxy

solution: dynamic subclass

- as3 out-of-the-box has no support for aop, so...
- as3 lets you load code in to the Flash Player at runtime with `flash.display.Loader`
- if you could assemble the bytecode for a subclass on the fly and load it in to the Flash Player, surely you could make it do whatever you wanted it to...

what would it look like?

a really simple base class

```
package amazing.code
{
   public class BaseClass
   {
      public function doSomething() : void
      {
         trace("doSomething() in BaseClass.");
      }
   }
}
```

a “dynamic” subclass

```
public class SubClass extends BaseClass
{
    public var closures : Dictionary;

    public function SubClass()
    {
        super();
        closures = new Dictionary();
    }

    override public function doSomething() : void
    {
        if (closures["doSomething"])
        {
            closures["doSomething"].apply(this, arguments);
        }

        super.doSomething();
    }
}
```

how would we do it?

mozilla tamarin

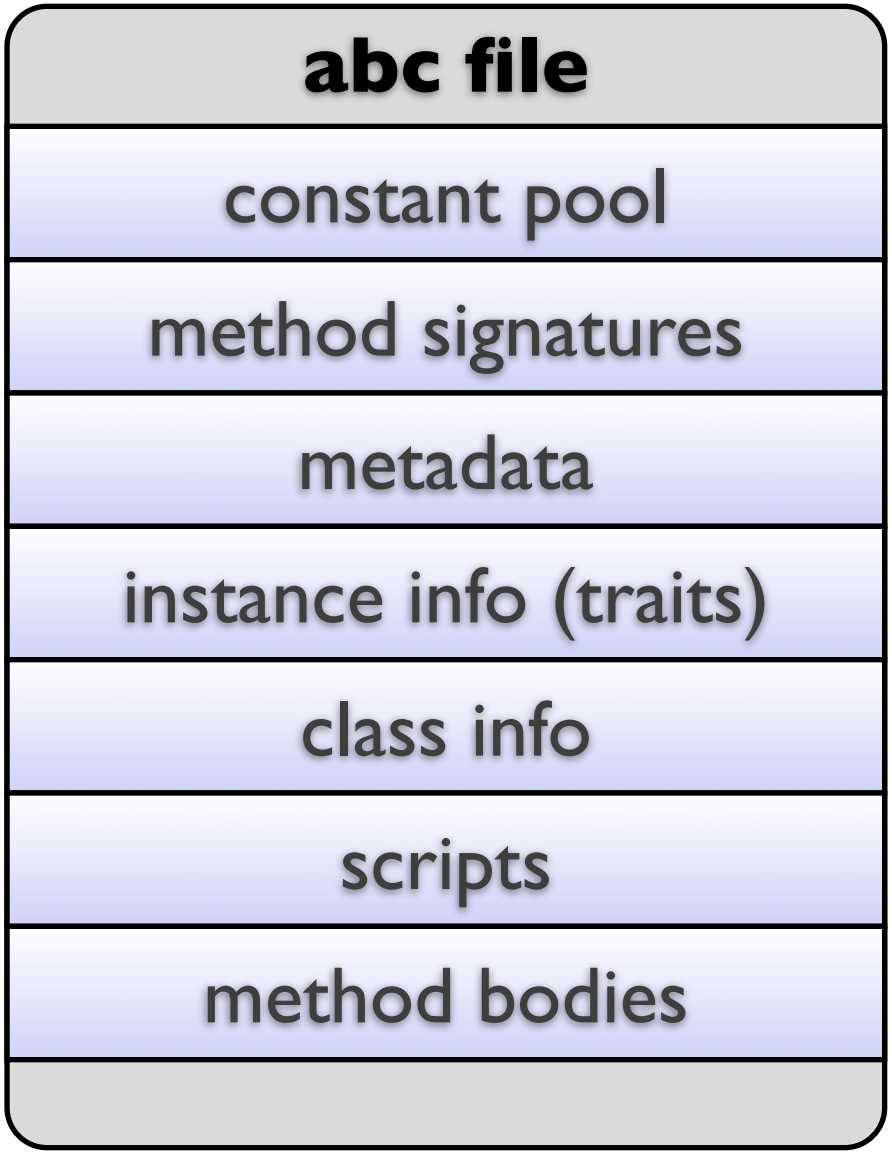
- open source as3 compiler/ interpreter project (es3 too)
- lots of command-line tools
- implements the AVM2 spec (ActionScript Virtual Machine 2)

useful bits of tamarin

- asc - ActionScript Compiler
- abcdump - shows the bytecode
- avmplus - an AVM2 shell
- the AVM2 spec itself (108 pages)

AVM2: all about the abc file

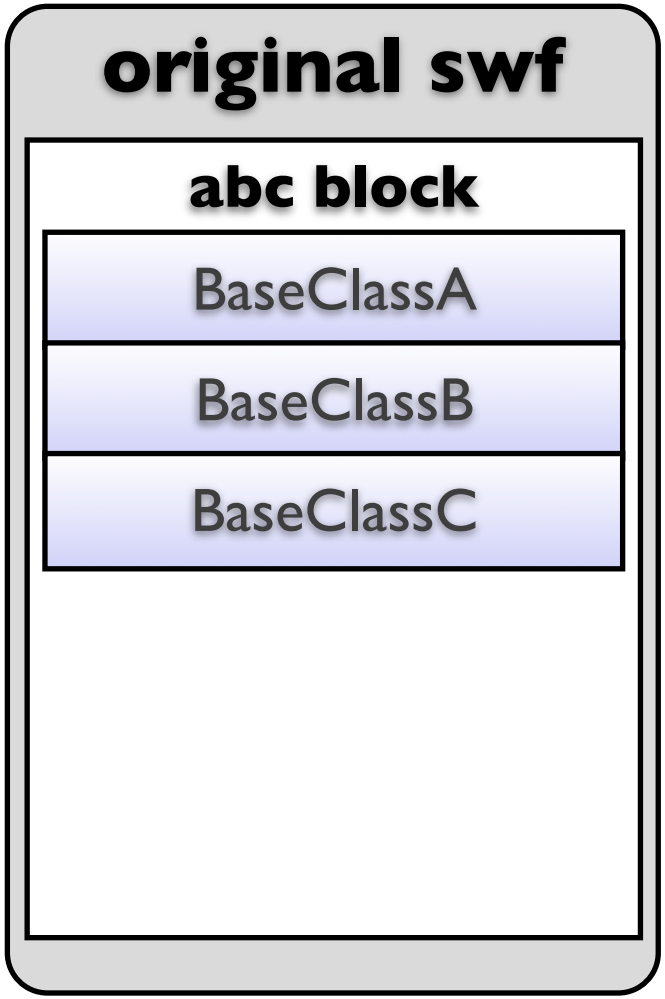
```
abcFile
{
  u16 minor_version
  u16 major_version
  cpool_info constant_pool
  u30 method_count
  method_info method[method_count]
  u30 metadata_count
  metadata_info metadata[metadata_count]
  u30 class_count
  instance_info instance[class_count]
  class_info class[class_count]
  u30 script_count
  script_info script[script_count]
  u30 method_body_count
  method_body_info
  method_body[method_body_count]
}
```

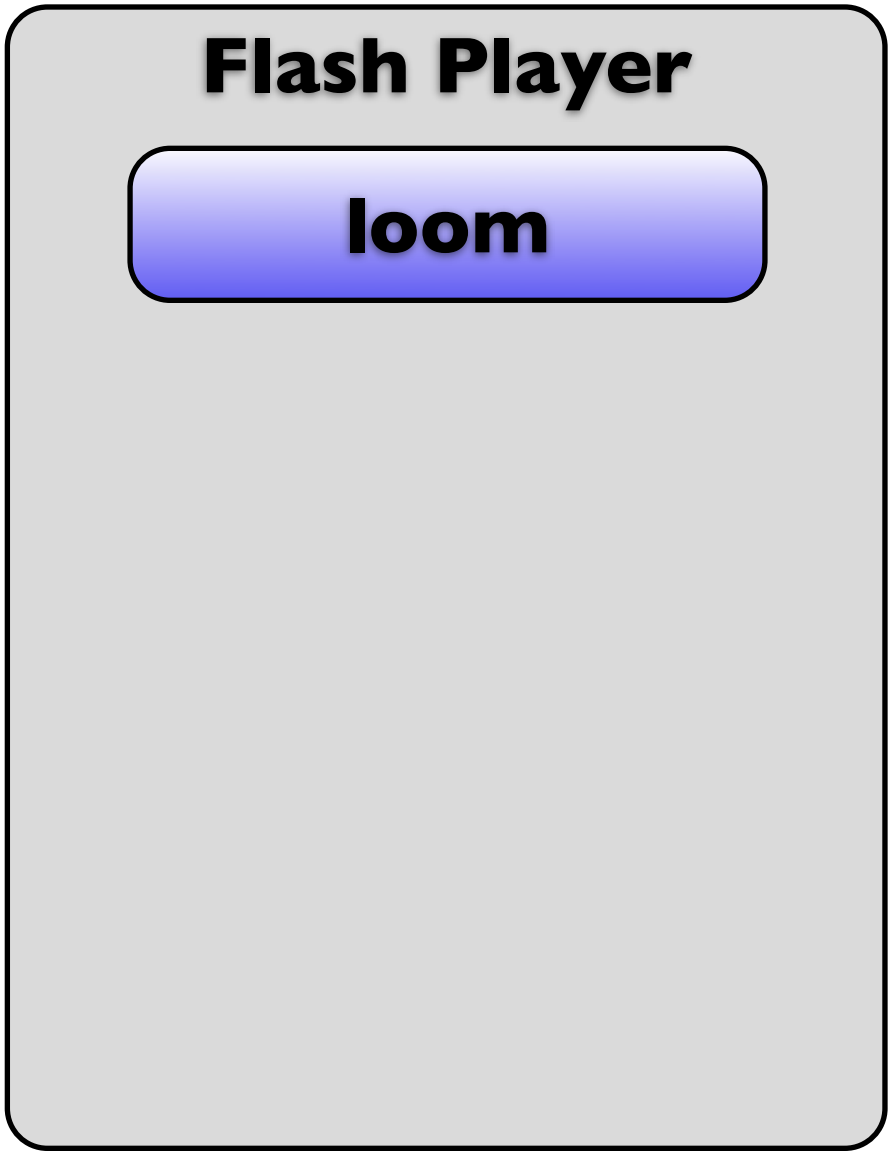


let’s see some bytecode

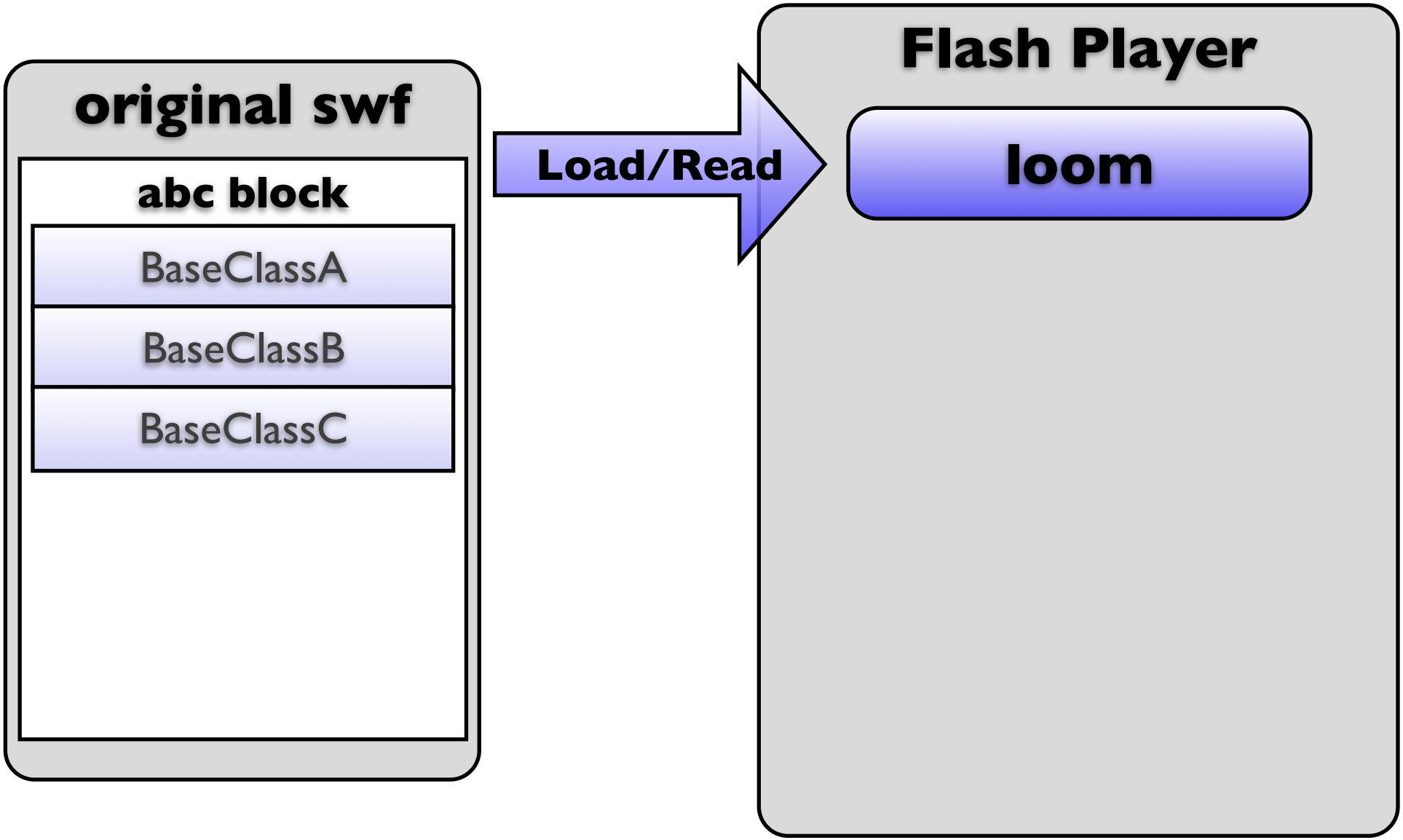
loom - a bytecode weaver

how loom works

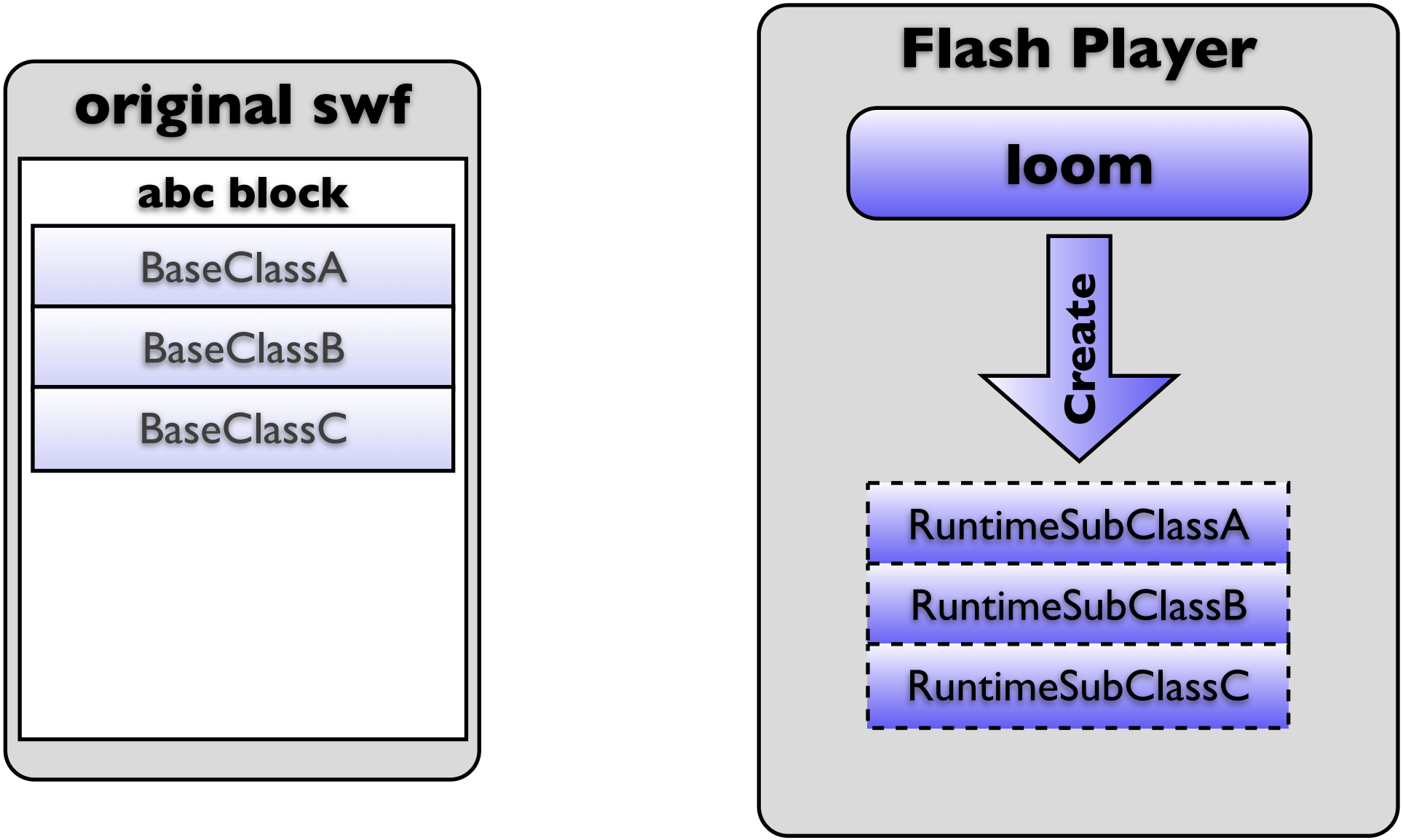




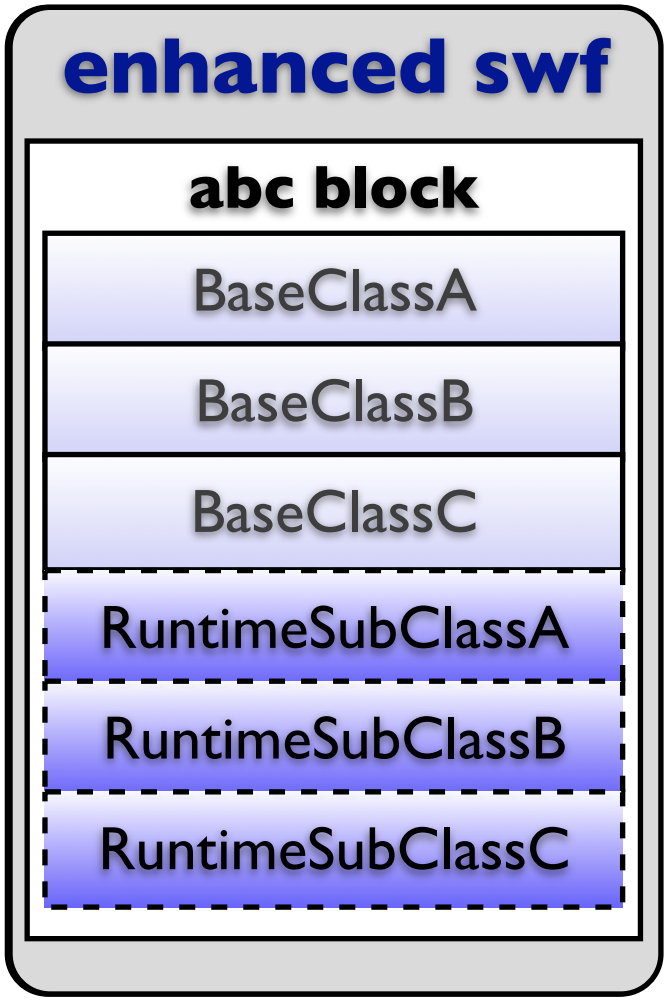
how loom works

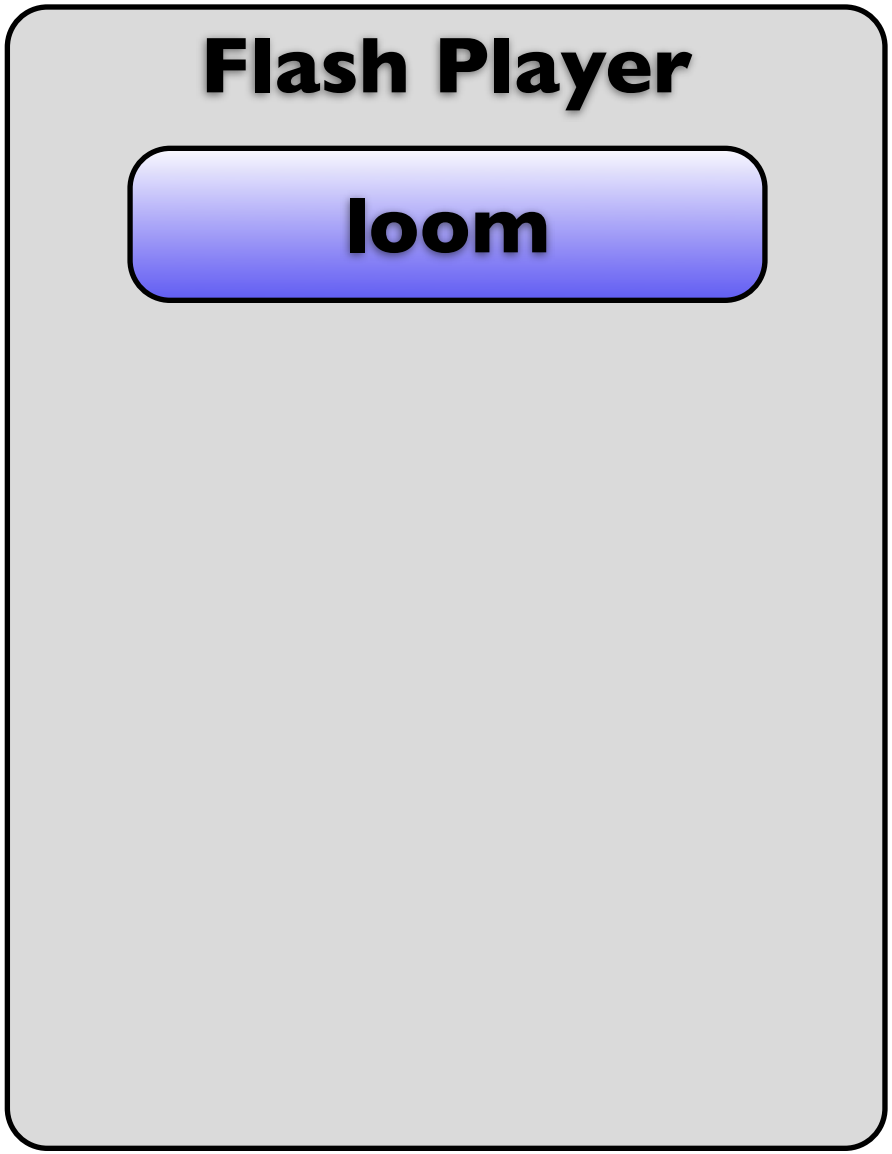


how loom works

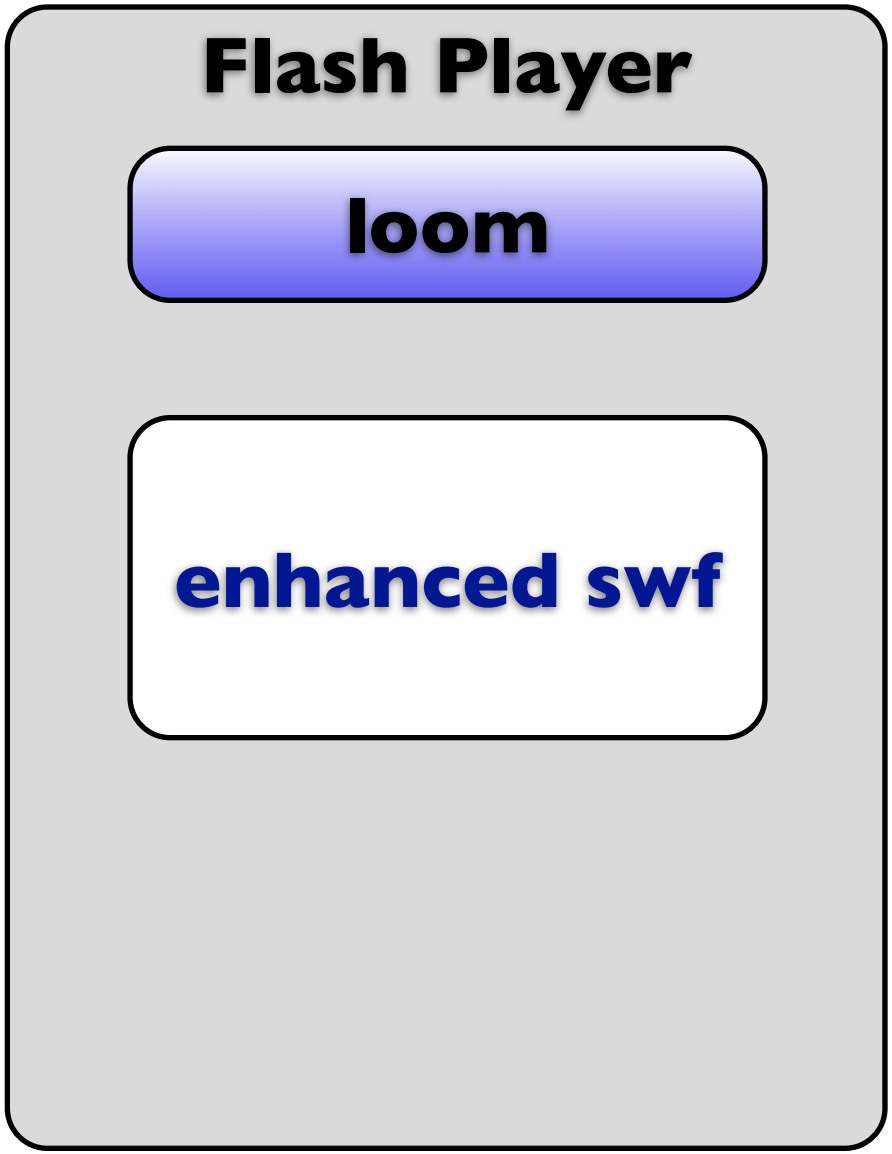


how loom works





how loom works



loom roadmap

- load-time and run-time bytecode weaving
- reflection API
- AOP via advice closures

where to find more

Tamarin

http://www.mozilla.org/projects/tamarin/

AVM2 Spec

Google “avm2overview”

Loom

http://code.google.com/p/loom-as3

http://www.maximporges.com

maxim.porges@yahoo.com

thanks :)