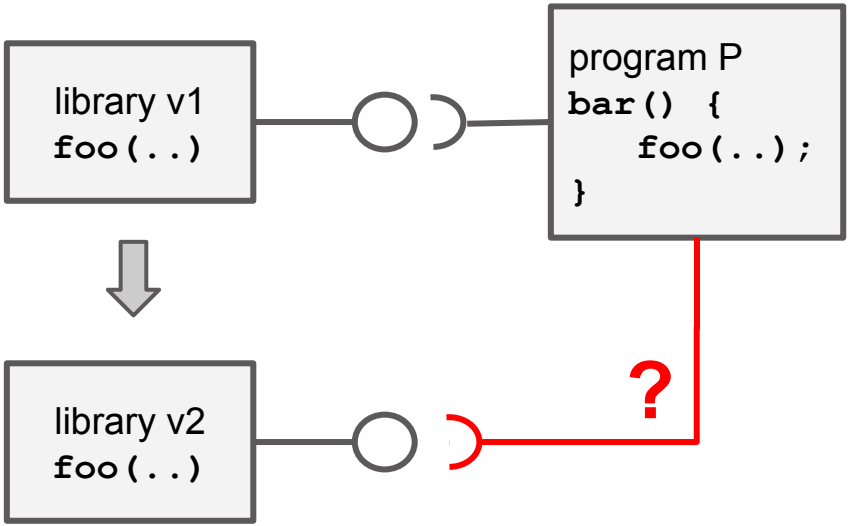
Magic with Dynamo

Flexible Cross-Component Linking for Java with Invokedynamic

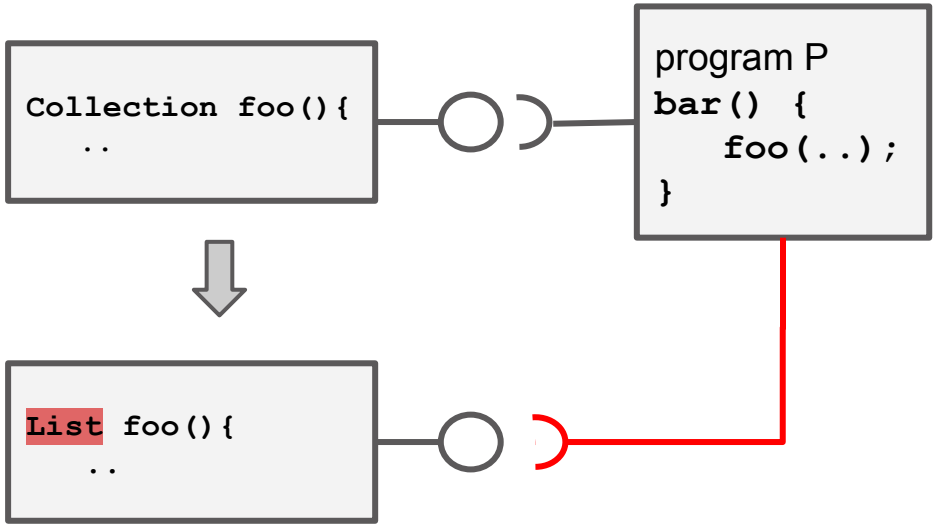
Kamil Jezek, Univ. of West Bohemia, Czech Republic
Jens Dietrich, Massey University, New Zealand

Background: Types of Compatibility



- P compiled against lib-v1
- **source compatibility**: can P be recompiled with lib-v2 ?
- **binary compatibility**: can P be linked with lib-v1?

Background: Types of Compatibility Ctd



+ source compatible

- binary compatibility

descriptor changes from:
`()Ljava/util/Collection;`
to: `()Ljava/util/List;`

The Big Picture

- JVM innovation focuses on security, scalability, ...
- language innovation focuses on programmer productivity

How do Developers Cope ?

from https://www.youtube.com/watch?v=M7FIvfx5J10 , Standard YouTube Licence

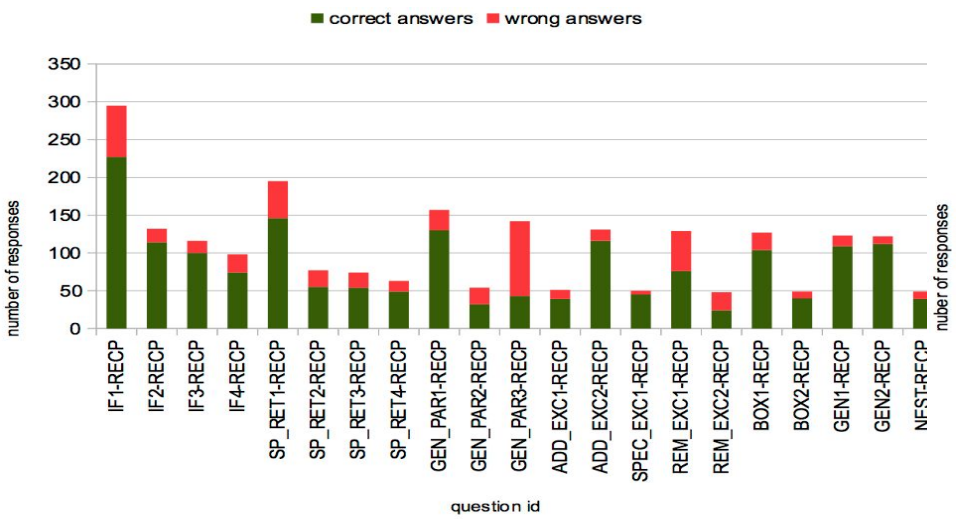
CSMR'14 Study

- studied 111 programs, 661 versions
- incompatible API upgrades are common in real-world libraries: 344/455 (75%) of upgrades affected
- incl commodity libraries: (antlr, ant, hibernate, weka, colt, junit)
- some cases of binary incompatible, but source compatible API Evolution

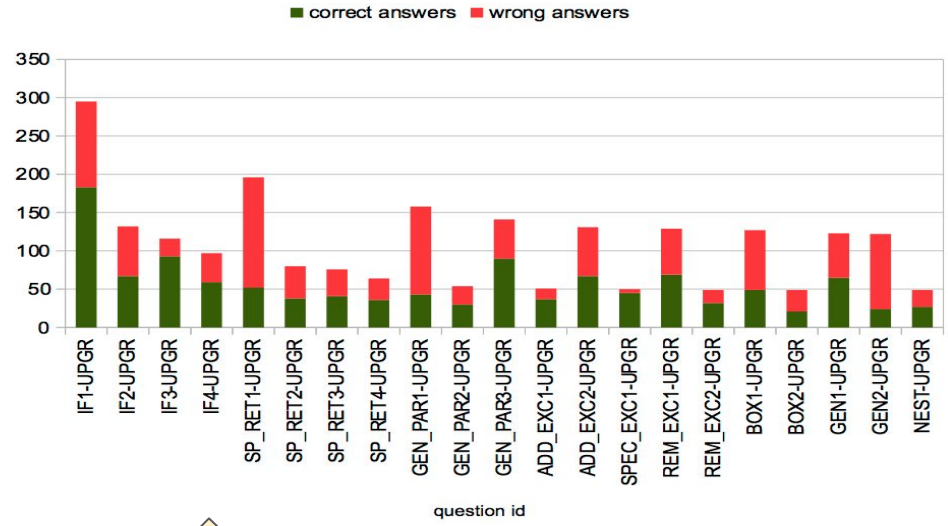
Puzzler Quiz (Journ. of Emp. SE ’15)

- use puzzlers to quiz developers
- 414 unique respondents, mostly from industry and with many years of relevant experience

Puzzler Quiz Results



source compatibility



binary compatibility

Runtime problem in GitHub issues

<error name> site:github.com inurl:issues [accessed 4 May 16]

java.lang.NoSuchMethodError:	**4,910**
java.lang.ClassCastException:	4,850
java.lang.StackOverflowError:	1,300
java.lang.OutOfMemoryError:	3,120
java.lang.NullPointerException:	26,700

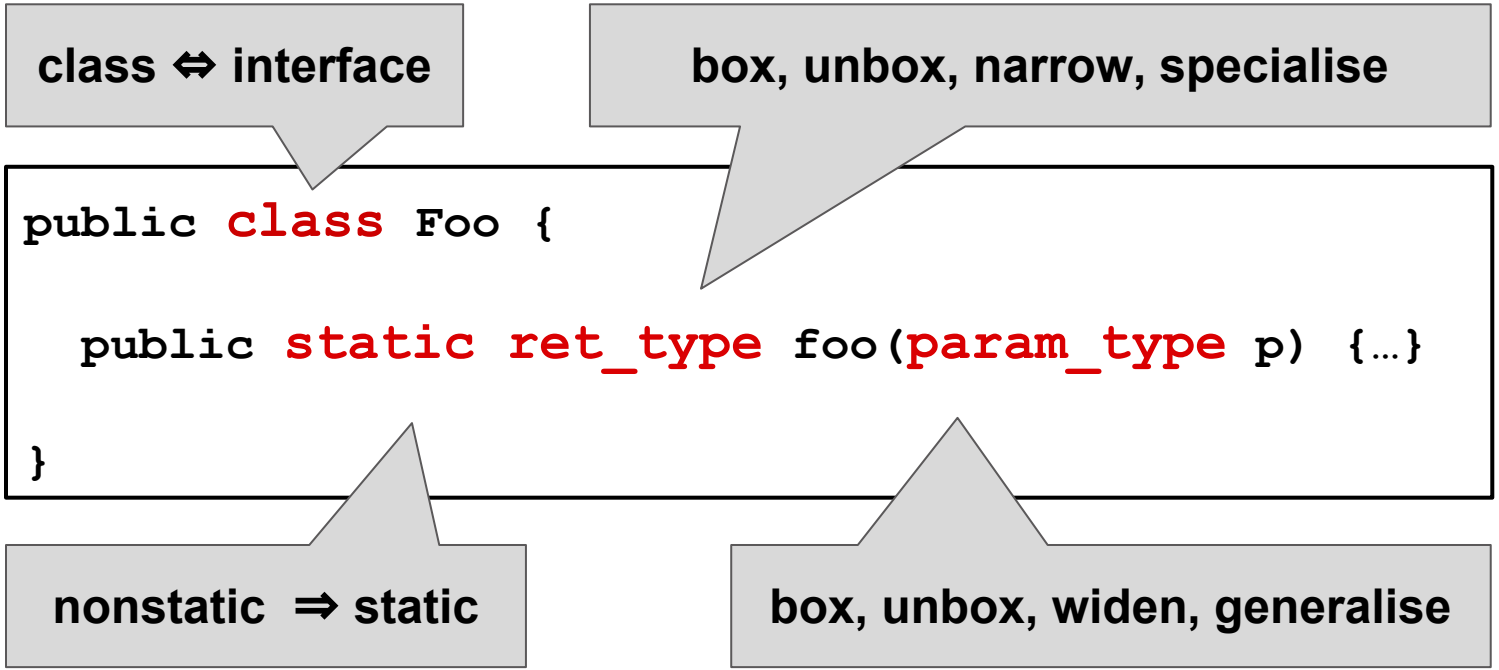
We have a Problem !

- incompatible API upgrades are common (CSMR’14)
- commodity libs are affected: antlr, ant, hibernate, junit
- developers lack knowledge about bin compatibility (Emp. SE’15)
- issue in bug tracking systems and stackoverflow

The Plan

- improve consistency between compiler and linker
- compile cross-component calls differently
- reduce link-related errors in programs

Evolution Patterns



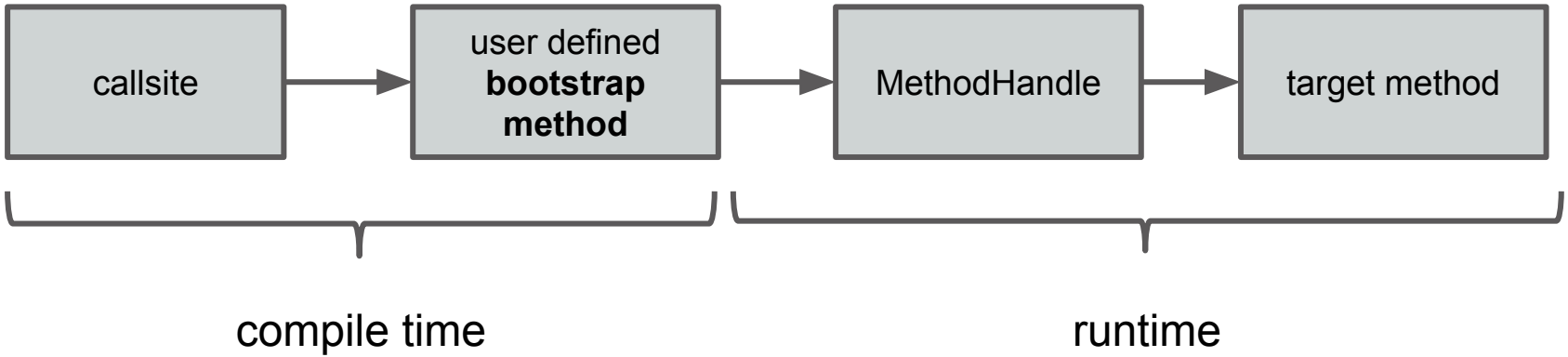
Engineering Options

- use protocols such as **`#doesNotUnderstand`** in libs
- use adapters / proxies
- change the JVM / linker
- **design goal: transparency for client & lib**
- we need magic !

To the rescue: `invokedynamic`

- new bytecode instruction added in Java 7
- to facilitate dynamic languages on the JVM
- compile lambdas in Java 8

The mechanics of `invokedynamic`

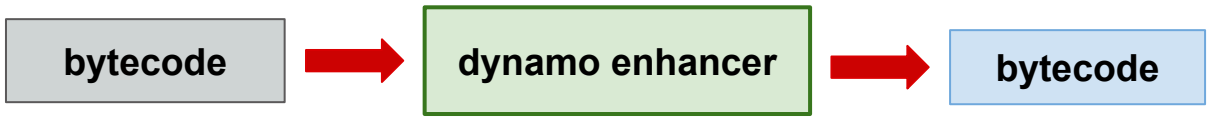


Invocation instruction comparison

	selection of target name + descriptor	**selection of actual method**
`invokestatic`	compiletime	compiletime
`invokevirtual`, `invokeinterface`, `invokespecial`	compiletime	runtime
`invokedynamic`	runtime	runtime

Dynamo: Bytecode enhancer

- modify bytecode of cross-component callsites
- construct invokedynamic callsite specifier
- provide reference to bootstrap method



Example 1: `foo.setValue(42)`

```
bipush 42
invokevirtual lib/Foo.setValue:(I)V
```

```
bipush 42
invokedynamic <bootstrap> setValue:(Llib/Foo;I)V
```

`invokespecial` .. being special

1. private method invocations: ignore
2. invocations via super: handled like invokevirtual
3. constructor invocations **`<init>`**:
   - must detect **bytecode behaviour**:

     `new C, dup, …, invokespecial C.<init>:(A*)V`
   - replace by single **invokedynamic**

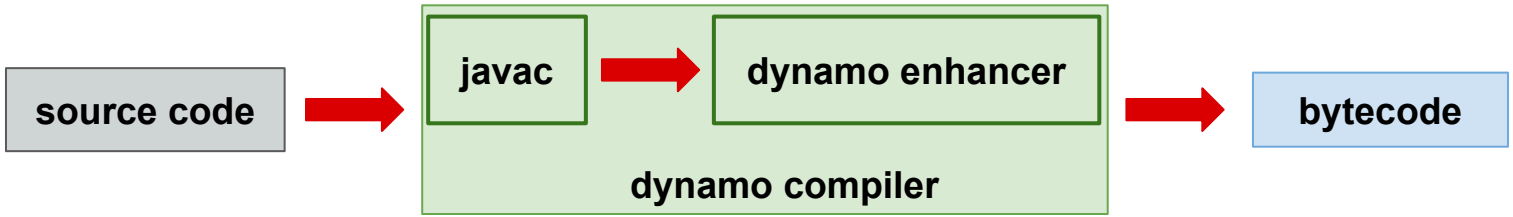
Example 2: `new lib.Foo(42)`

```
new lib/Foo
dup
bipush 42
invokespecial lib/Foo."<init>":(I)V
```

```
bipush 42
invokedynamic C$D:(I)Llib/Foo;
```

Dynamo: Compiler

- post-compiler of standard **javac**
- use javac via **JSR-199**
- parameter to customise what to compile special



The Filter DSL

what to enhance

```
-callsite com.foo.Bar
-target com.foo.*
+target java.lang.String#substring
```

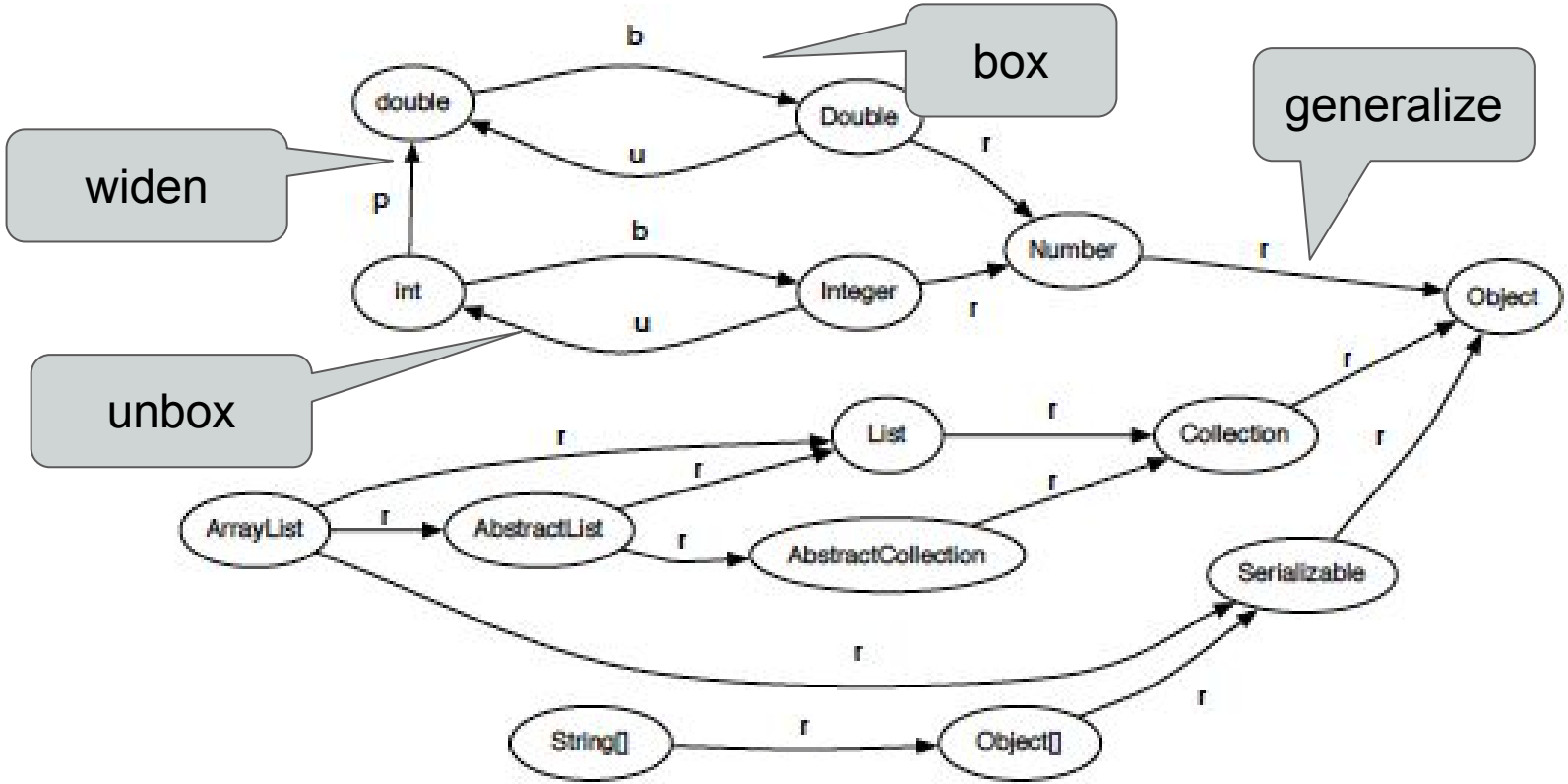
Dynamo: Bootstrap methods

- provided by dynamo runtime library
- compute instance of `java.lang.invoke.CallSite`
- callsite provides ref to target method
- only done once

Linking (Runtime Resolution)

- need to find method **similar** to target of replaced invoke
- deal with ambiguity: locate unique **best fit**
- conceptual model to summarise and simplify JLSpec conversion rules to locate “most specific method”

The Type Conversion Graph (TCG)



Adaptability

- adaptability of types defined as reachability in TCG
- reachability restricted by patterns labels must form: **`p|r*|br*|up`**
- then extend adaptability to descriptors, different directions for parameter and return types
- add owner type to descriptor (**extended descriptor XD**)
- model defines a partial order **<** “adaptable”

Adaptability Examples

- `Collection foo()` > `List foo()`
- `void foo(List)` > `void foo(Collection)`
- `void foo(int)` > `void foo(Integer)`
- `void foo(int)` > `void foo(Object)`
- `long foo()` > `int foo()`

Runtime Resolution Rules

1. only pick user-defined methods with matching name
2. pick unique most specific adaptable method w.r.t. adaptability
3. try to resolve ambiguity: if there are multiple most specific methods with same param types, pick one with most specific return type
4. otherwise generate **`NoSuchMethodError`**

Ambiguity Example

- replace `foo(java.util.ArrayList)` by:
  1. `foo(java.io.Serializable)` and
  2. `foo(java.util.AbstractList)`

- results in `NoSuchMethodError`

Benchmarks

- use micro benchmarks to measure overhead of compilation / runtime resolution
- based on test cases representing evolution patterns
- use JMH
- 15 warmups and 30 trial runs

Compiler Benchmark Results

benchmarks	average runtime (ms)	stdev (ms)	bench-mark count	average runtime single benchm. (ms)	confidence interval (ms) (99.9%)
classic	1923	65	49	39.24	[1857, 1988]
dynamo	2142	91	49	43.71	[2051, 2233]

Runtime Benchmark Results

benchmarks	average runtime (ms)	stdev (ms)	bench-mark count	average runtime single benchm. (ms)	confidence interval (ms) (99.9%)
all	0.180	0.001	14	0.013	[0.179, 0.181]

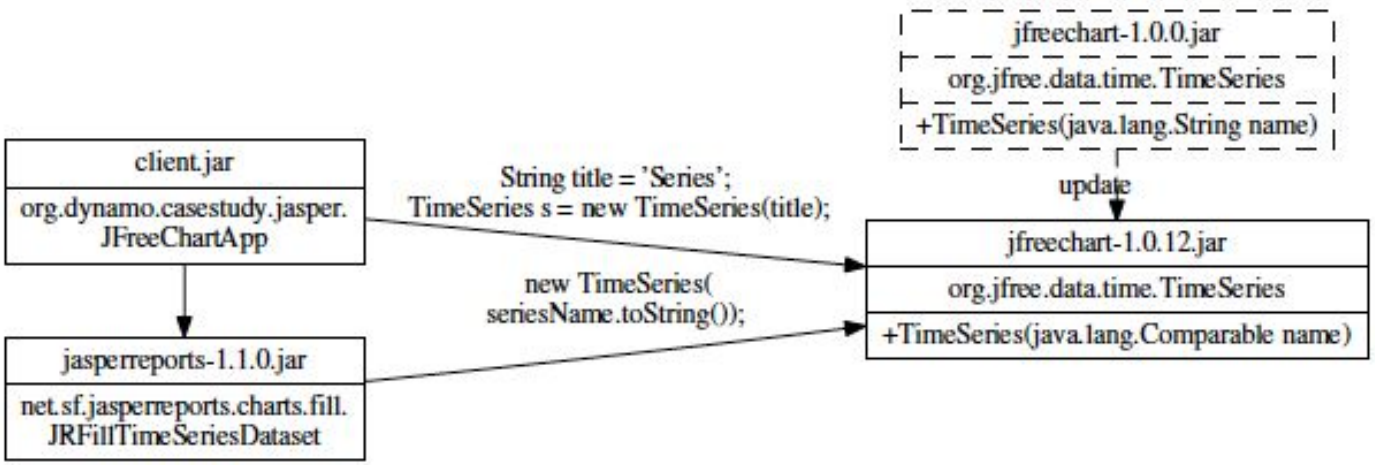
Case Study 1: jasperreport and jfreechart

- *jasperreport* is a popular open source library
- depends on other libs, incl *jfreechart*
- jfreechart **1.0.0** → **1.0.12** is incompatible:

```
org.jfree.data.time.TimeSeries(String) >
org.jfree.data.time.TimeSeries(Comparable)
```

- this breaks *jasperreport* with a **`NoSuchMethodError`**
- compiling *jasperreport* with **dynamo will fix this**
- macro-benchmarks confirm micro-benchmarks

Case Study 1 (ctd)



Case Study 2: The hazard of covariant return types and bridge methods

- issues observed when refactoring `clone()` to use co-variant return types
- blog post by Ian Robertson (2013)
- stack overflow errors can occur when combining:
  - deep class hierarchy across multiple libraries
  - dynamic dispatch
  - covariant return types (bridge methods)

Conclusion & Future Work

- code evolution is interesting
- designing and implementing a PL should take evolution into account
- interesting questions:
  - types of contracts between artefacts (types, semantics, QOS, licensing …)
  - how do contracts evolve?
  - semantic versioning

Acknowledgements

- Alex Buckley, Alex Potanin, Nic Hollingum for feedback
- Oracle Labs Australia for funding
- Dynamo the Magician

QUESTIONS ?

repository: https://goo.gl/I0HY1F

Additional Slides

Case Study 2: sources 1.0

```
class Wrapper
```

```
public Collection wrap(Object o) {...}
```

```
class WrapperChild extends Wrapper
```

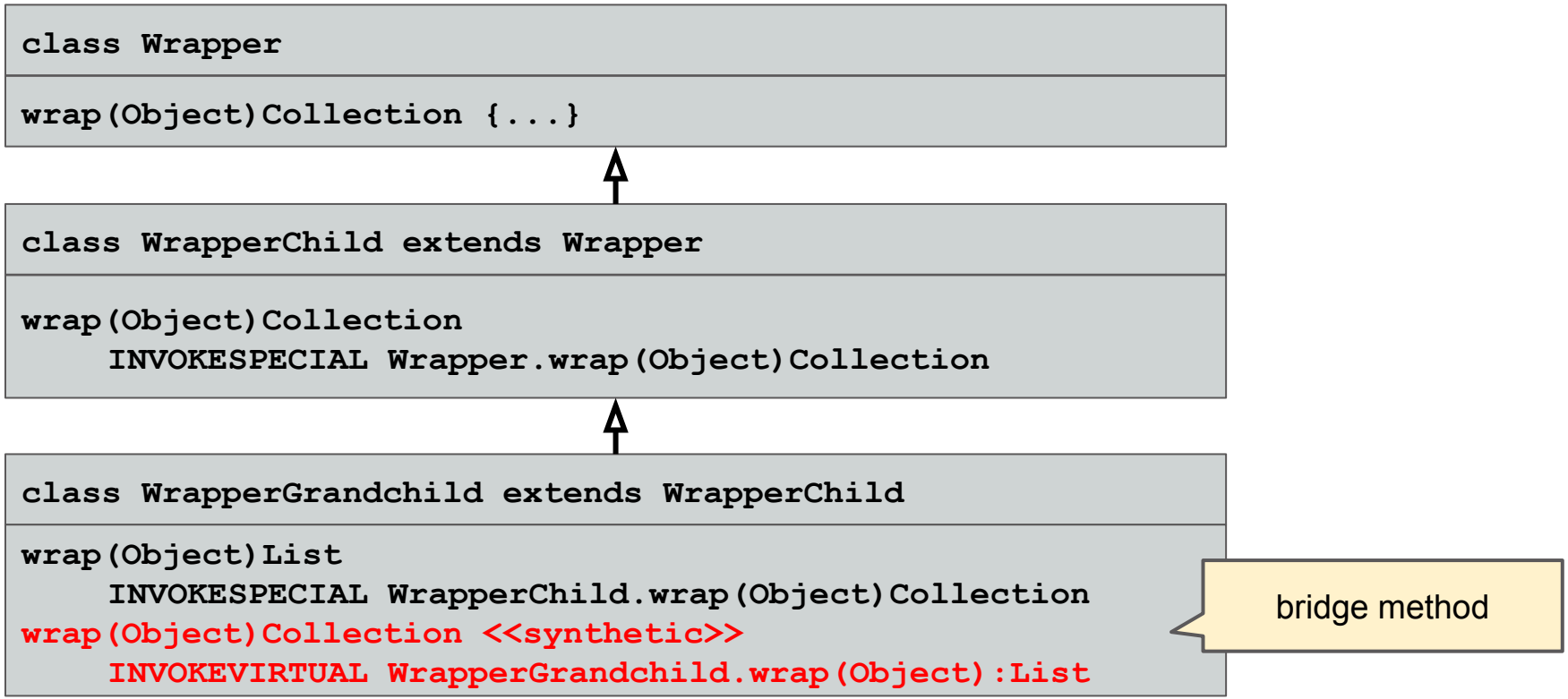
```
@Override public Collection wrap(Object o) {
    return super.wrap(o);
}
```

```
class WrapperGrandchild extends WrapperChild
```

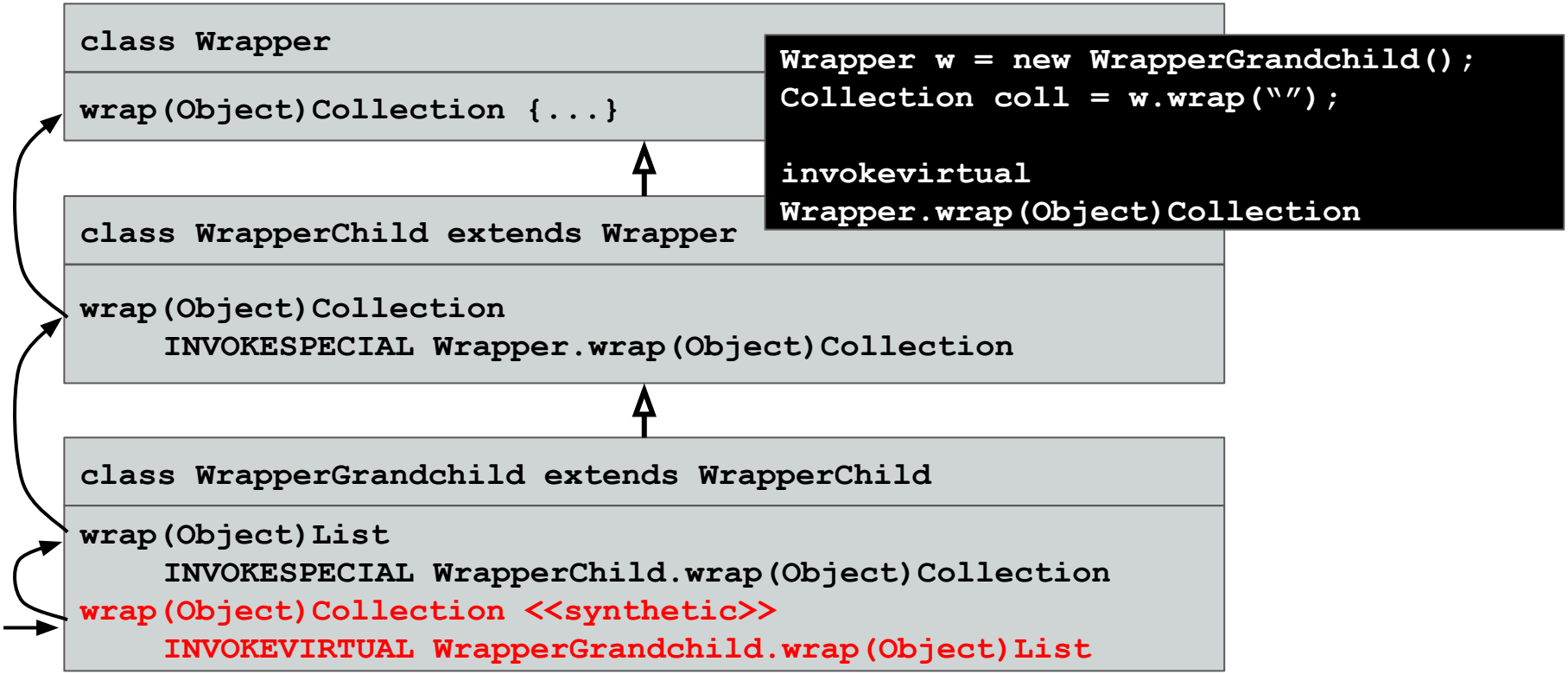
```
@Override public List wrap(Object o) {
    return (List)super.wrap(o);
}
```

covariant return type

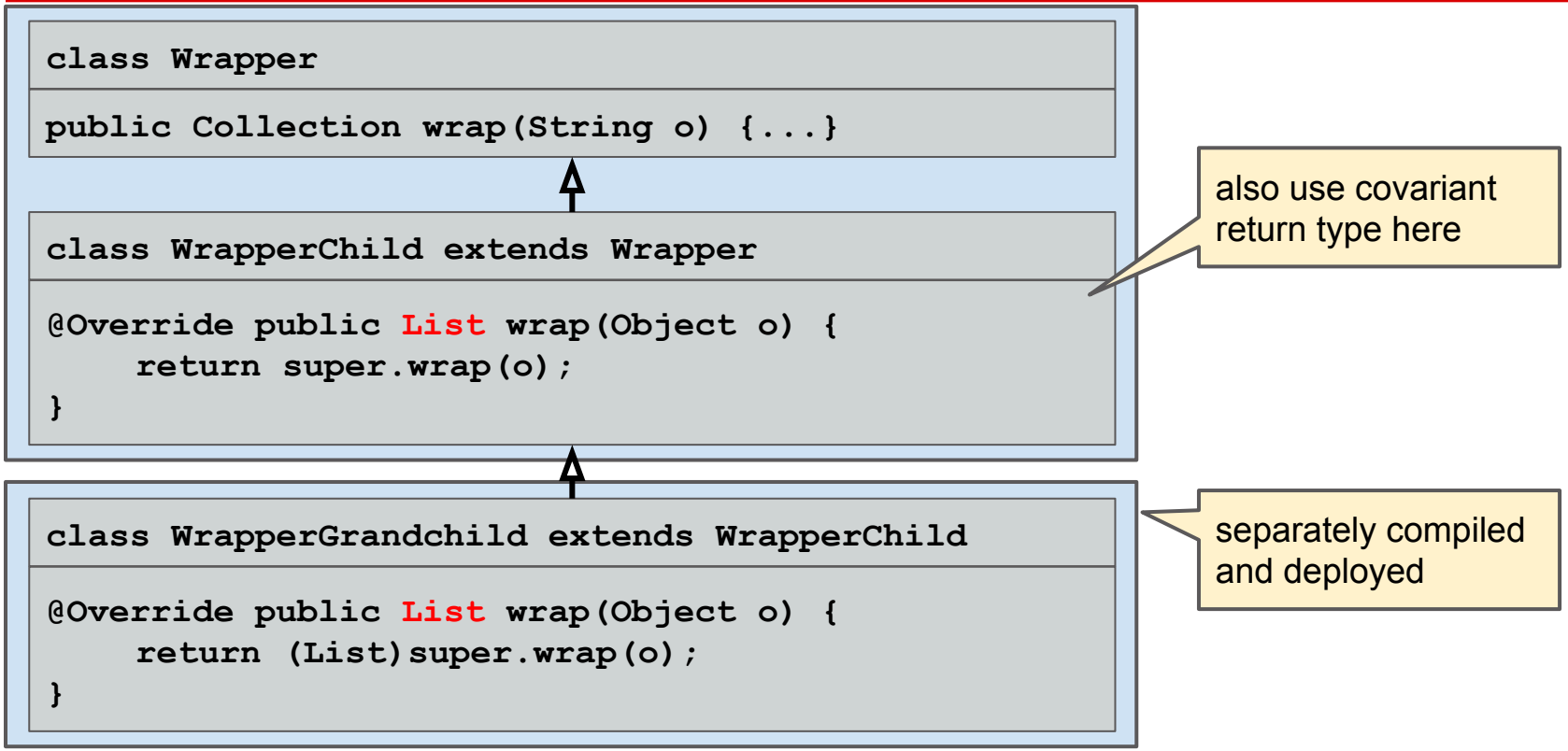
Case Study 2: bytecode 1.0



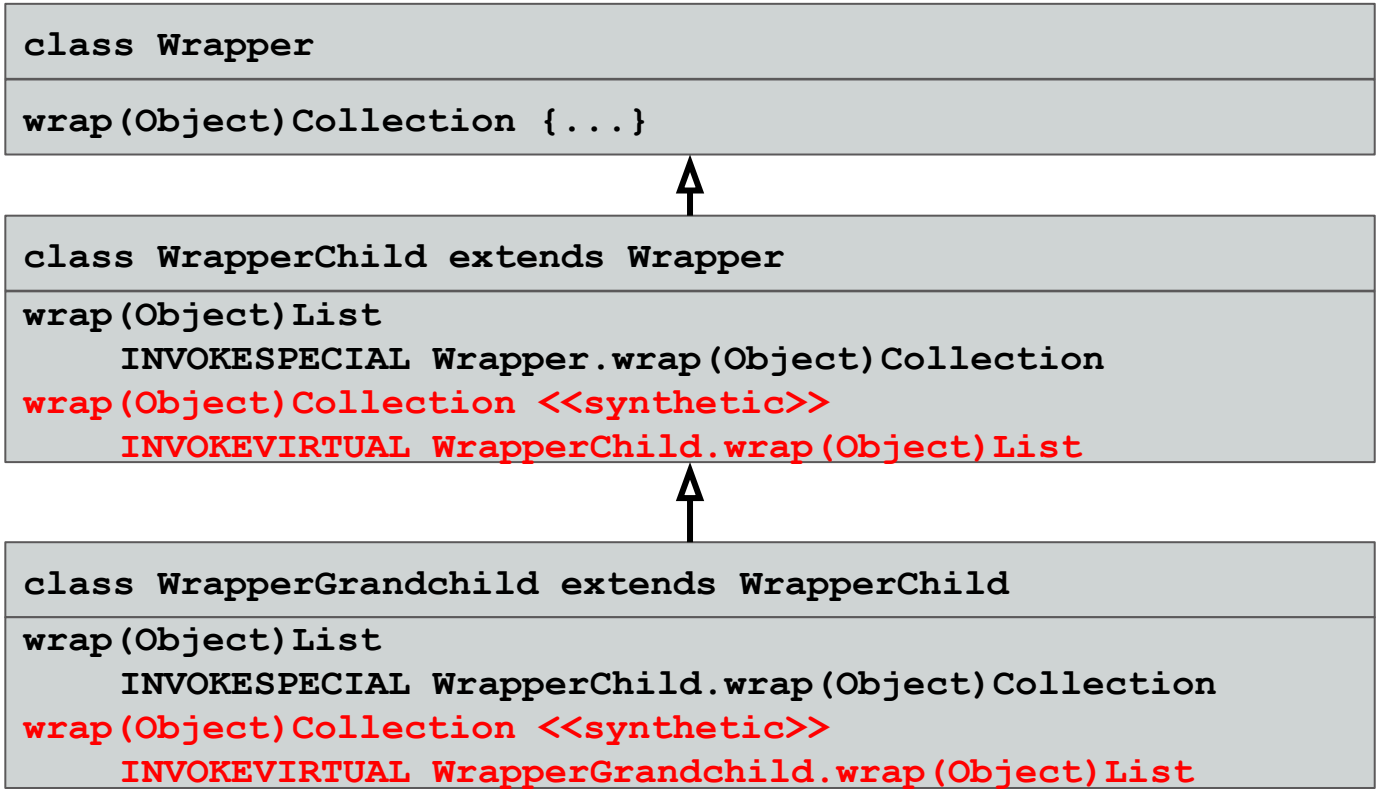
Case Study 2: callgraph 1.0



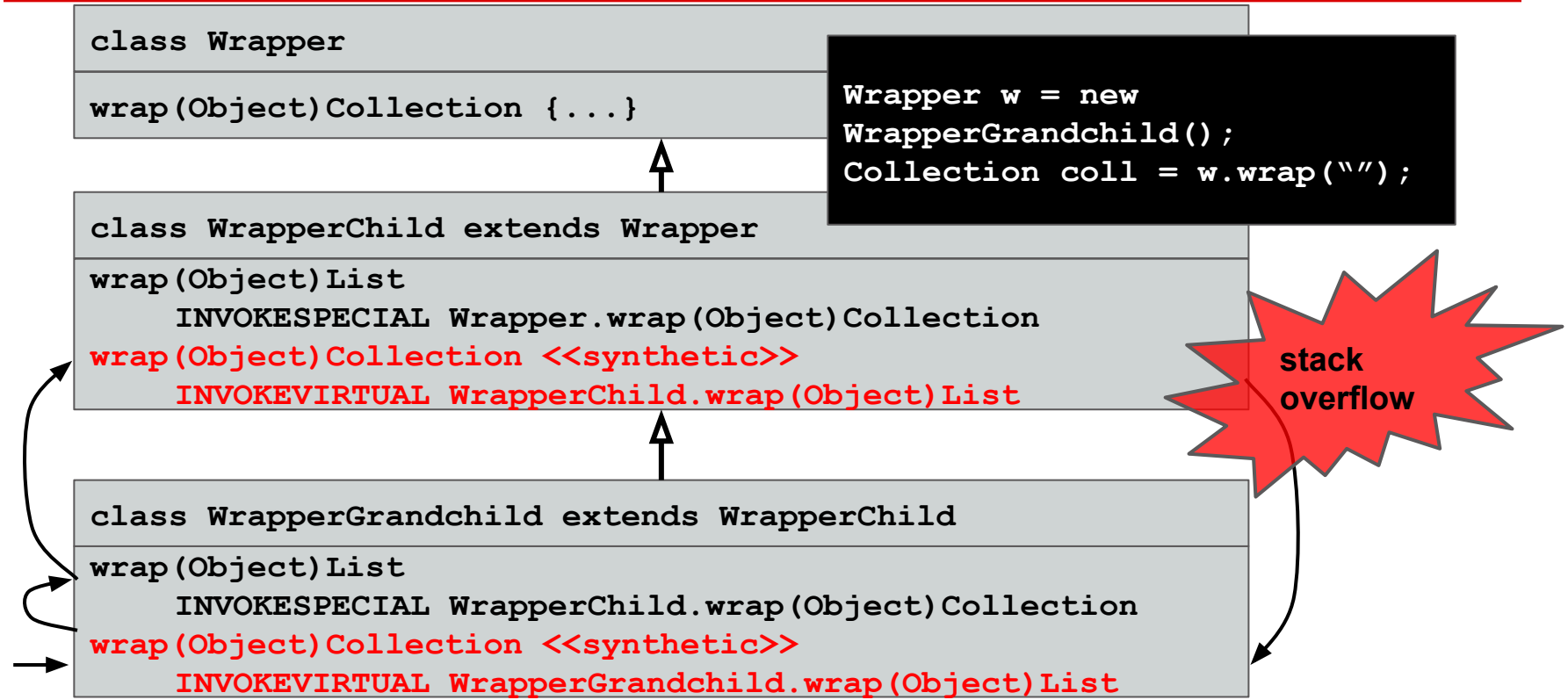
Case Study 2: sources 2.0



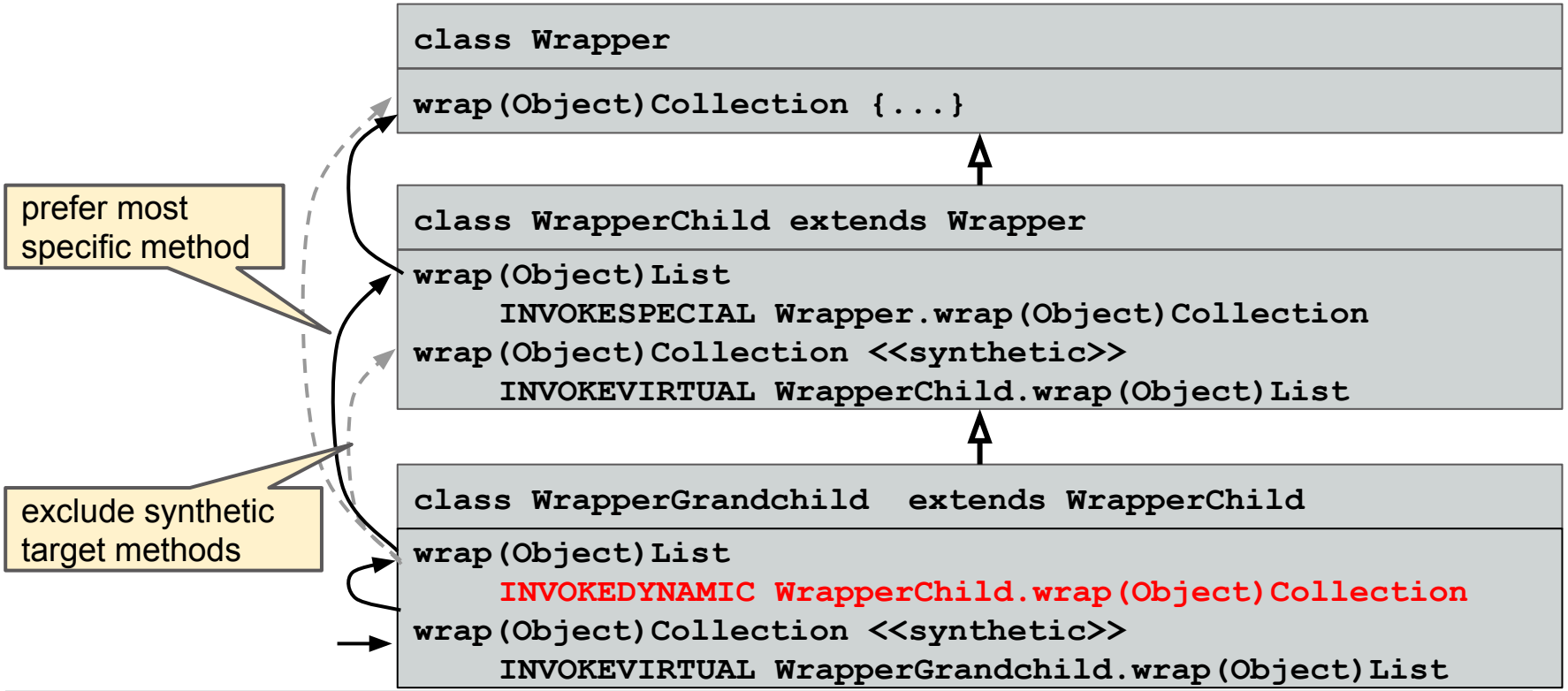
Case Study 2: bytecode 2.0



Case Study 2: callgraph 2.0



Case Study 2: dynamo to the rescue



Dynamo: Overview