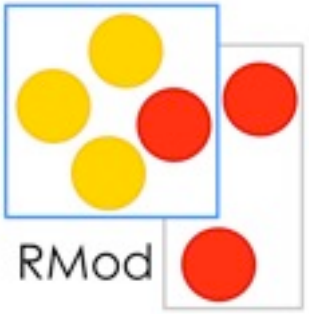


Read-Only Execution for Dynamic Languages

Jean Baptiste Arnaud, Marcus Denker, Stephane Ducasse,
Damien Pollet, Alexandre Bergel and Mathieu Suen

http://rmod.lille.inria.fr

Example

```
if (this.precondition())
{
    this.doSomething();
}
```

- Precondition might or might not make a side effect

Example

```
if (this.precondition())
{
    this.doSomething();
}
```

should ***only*** check the precondition

- Precondition might or might not make a side effect

Example

```
if (this.precondition())
{
    this.doSomething();
}
```

should ***only*** check the precondition

Shouldn't

- Precondition ~~might or might not~~ make a side effect

The problem

- Dynamic language
  - No Static analysis
  - Not Freeze the object
  - Propagation through object graph

- Related work
  - Dynamic Object Ownership (Noble, J)
  - Object-Oriented Encapsulation (Schärli, N)

The problem

- Dynamic language
  - No Static analysis
  - Not Freeze the object
  - Propagation through object graph

- Related work
  - Dynamic Object Ownership (Noble, J)
  - Object-Oriented Encapsulation (Schärli, N)

Ownership imposed by object graph

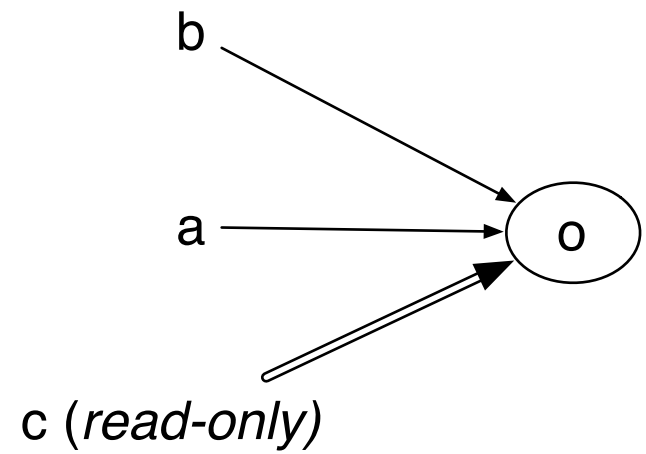
The problem

- Dynamic language
  - No Static analysis
  - Not Freeze the object
  - Propagation through object graph

- Related work
  - Dynamic Object Ownership (Noble, J)
  - Object-Oriented Encapsulation (Schärli, N)

Restrict the interface

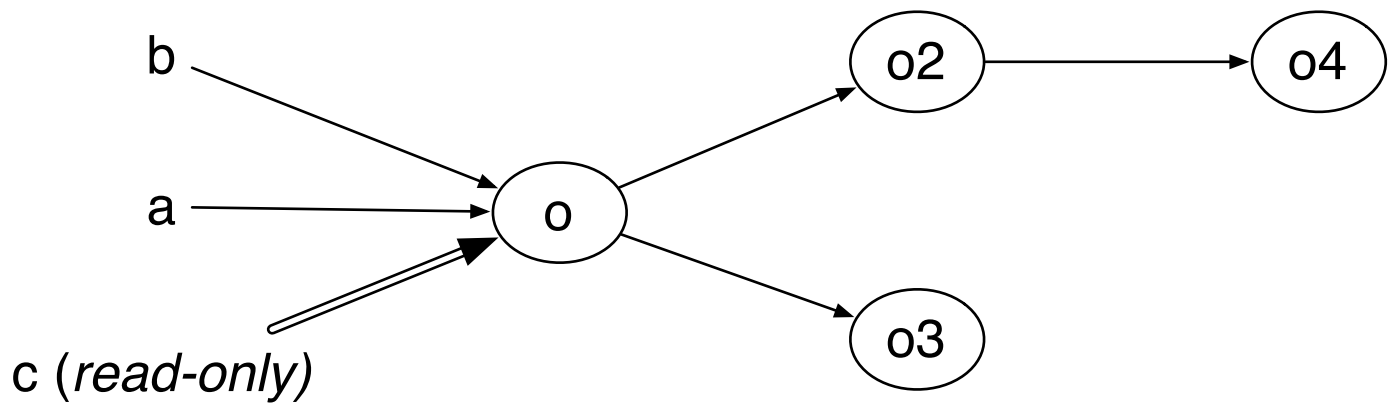
Read-only references



Read-only reference,
via a Handle (transparent proxy)

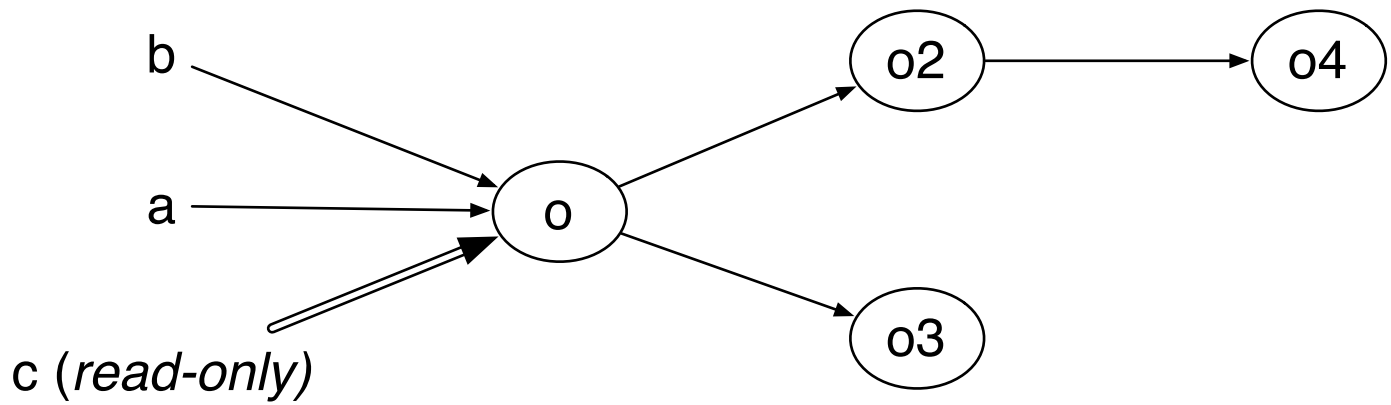
Principle of the Handle

Follow the execution flow



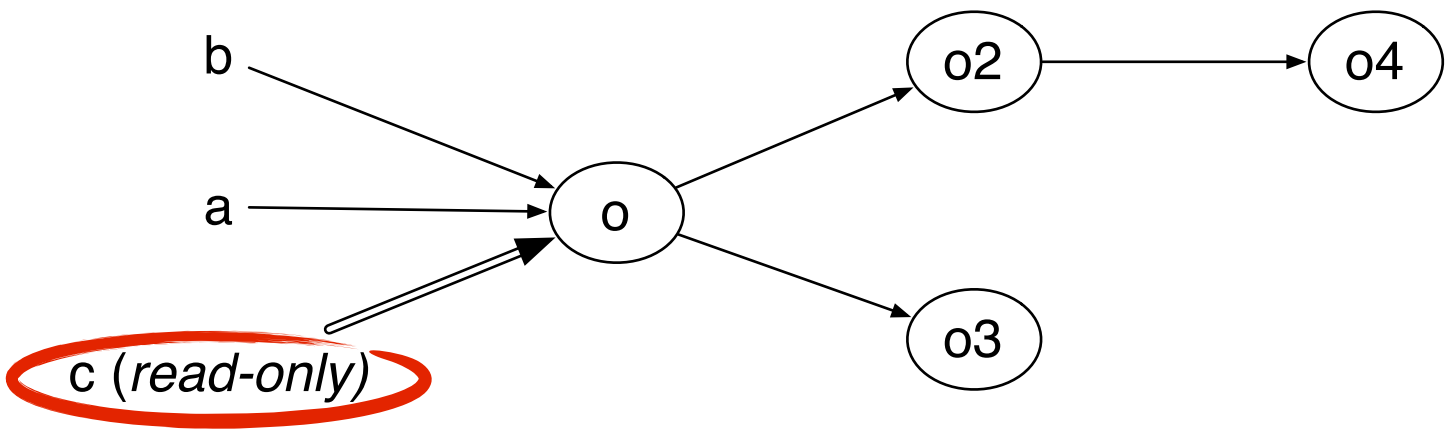
Principle of the Handle

The read-only mode propagates with execution



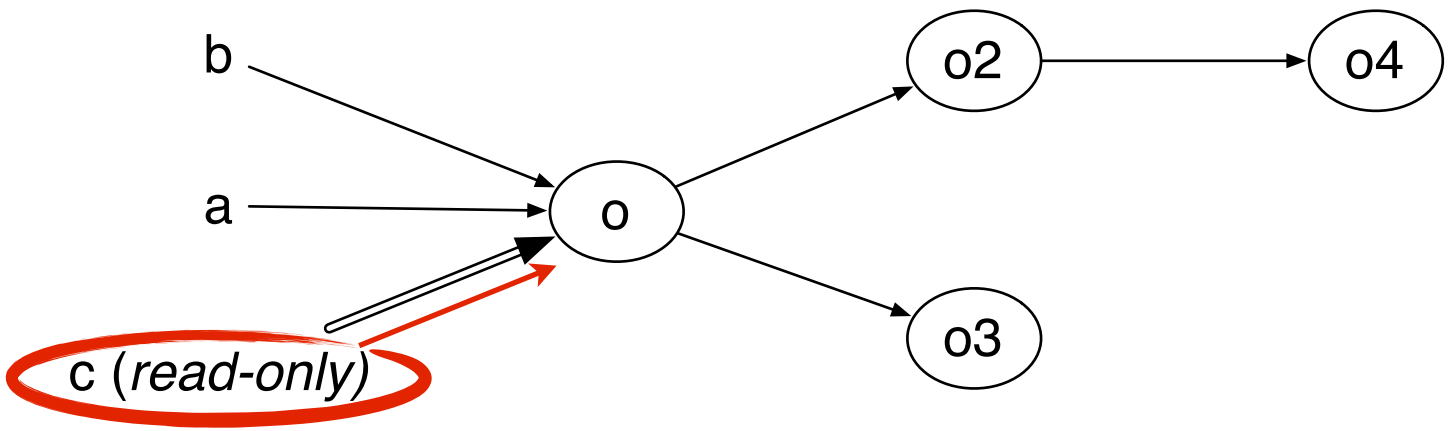
Principle of the Handle

The read-only mode propagates with execution



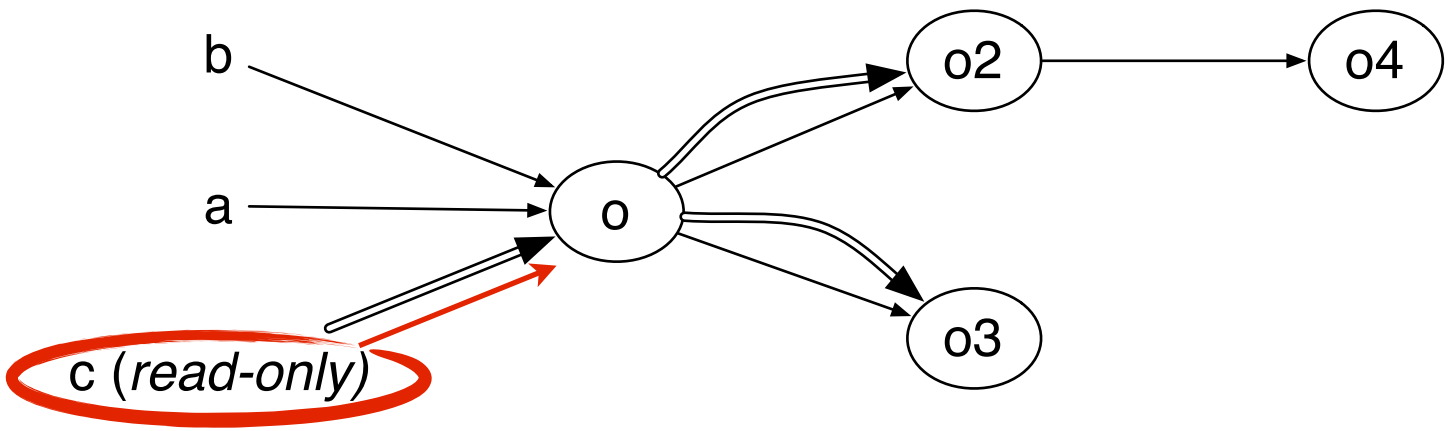
Principle of the Handle

The read-only mode propagates with execution



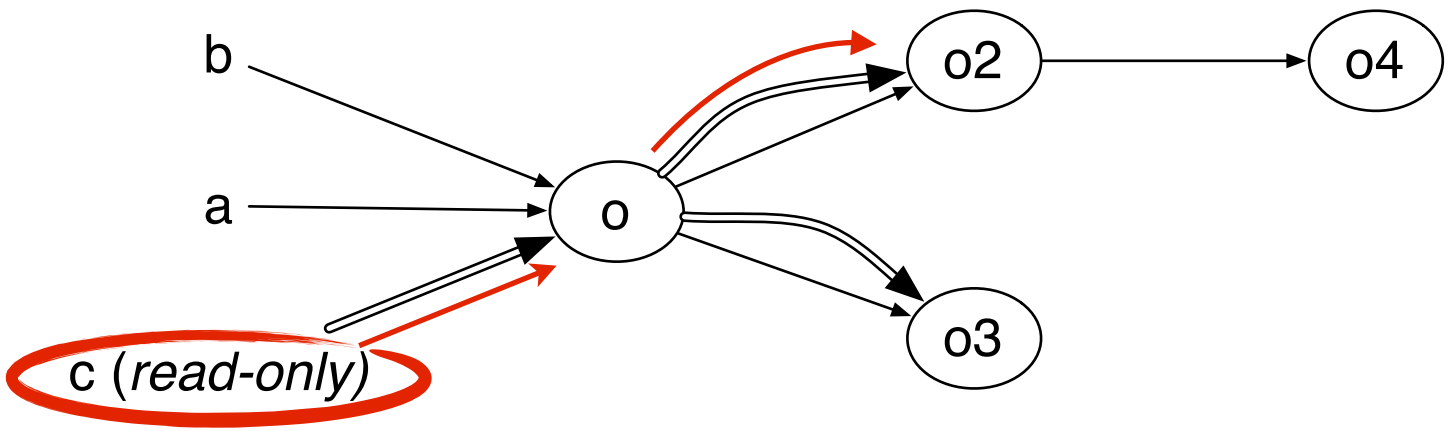
Principle of the Handle

The read-only mode propagates with execution



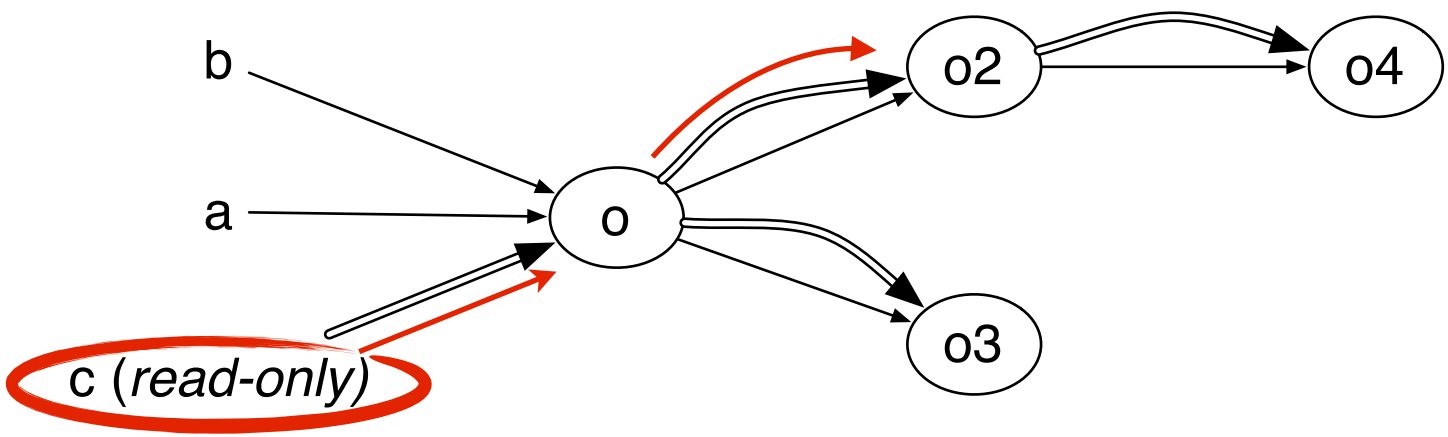
Principle of the Handle

The read-only mode propagates with execution



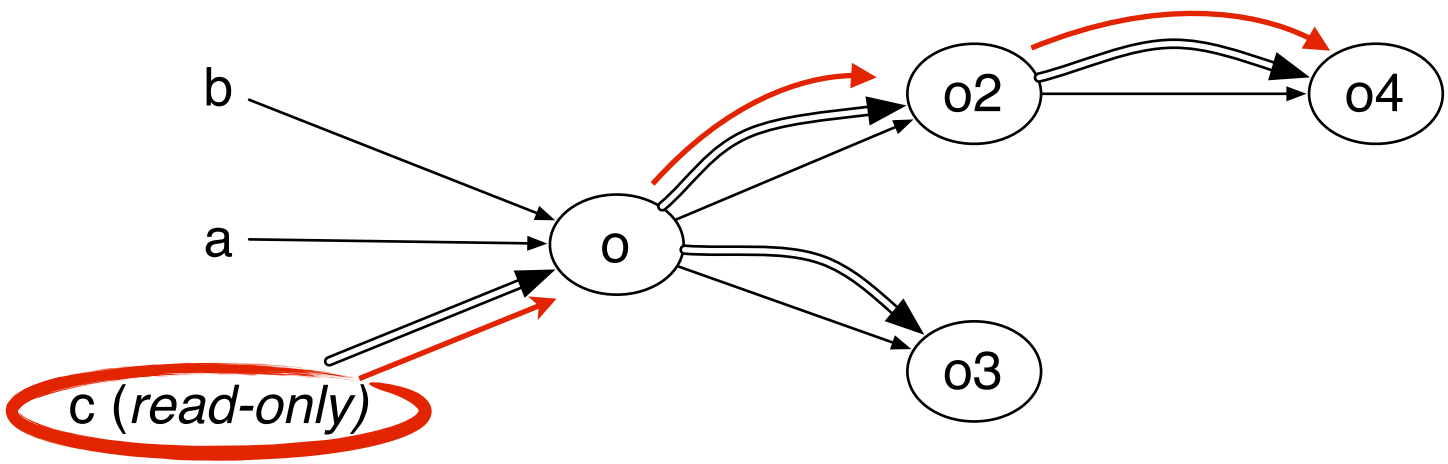
Principle of the Handle

The read-only mode propagates with execution



Principle of the Handle

The read-only mode propagates with execution

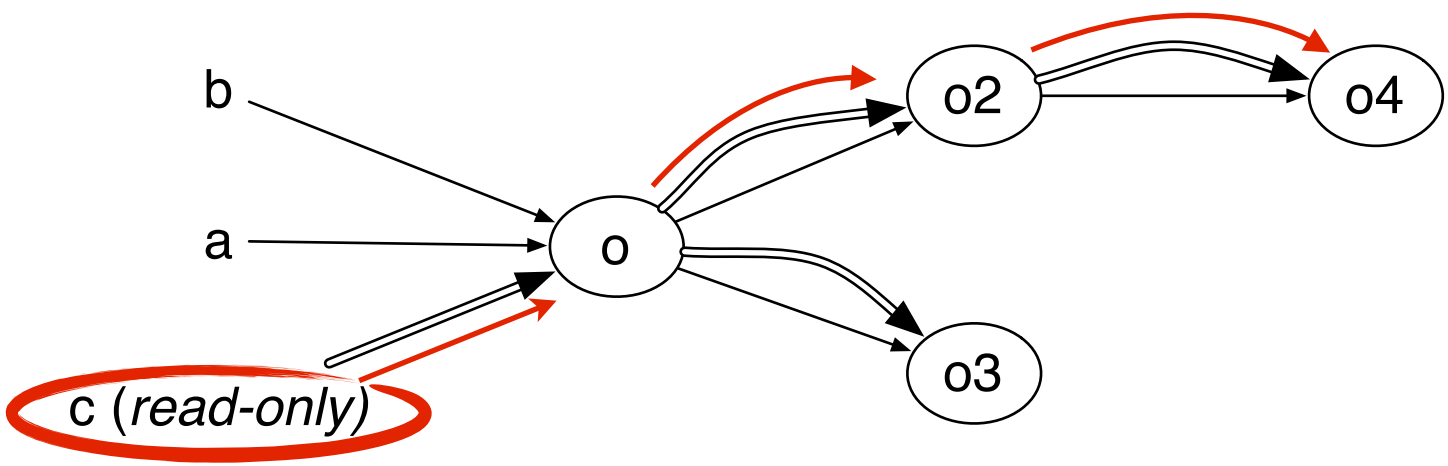


Principle of the Handle

The read-only mode propagates with execution

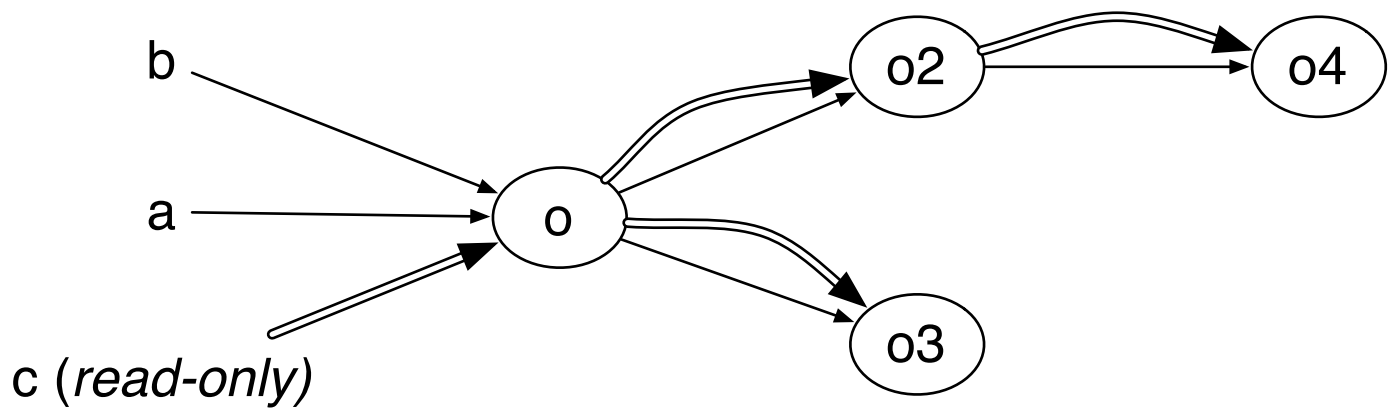
o4 attempts a side-effect!

=> Exception



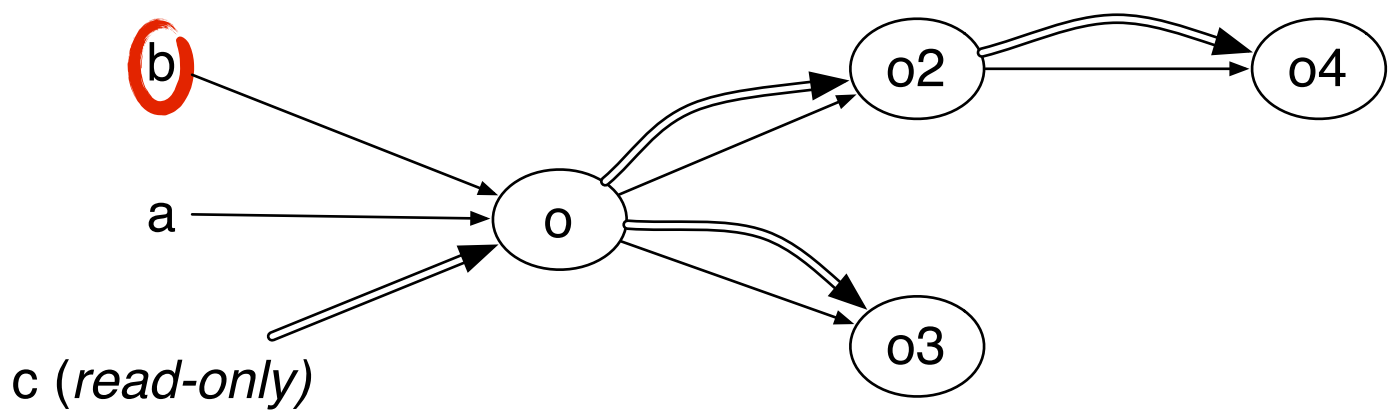
Principle of the Handle

Via normal reference,
normal execution



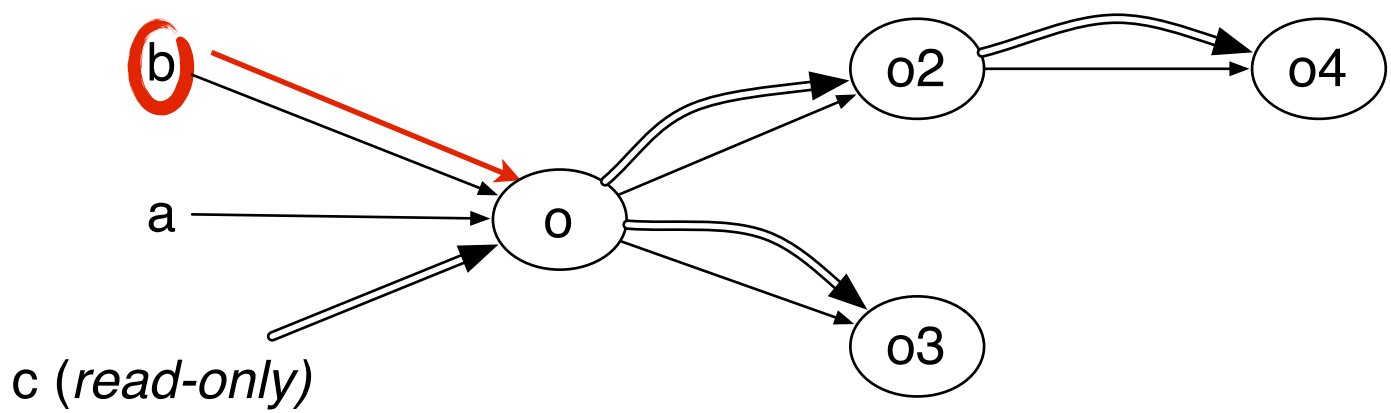
Principle of the Handle

Via normal reference,
normal execution



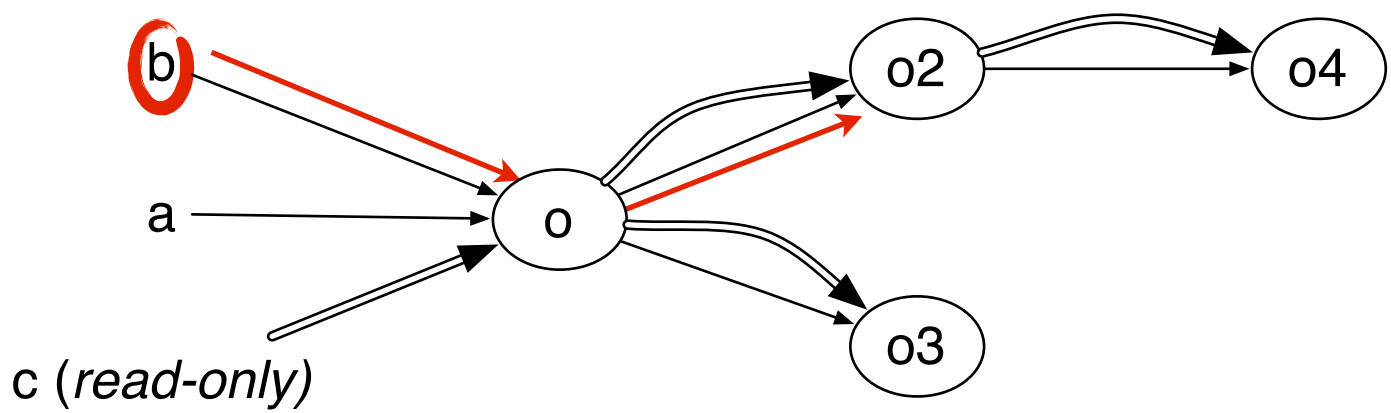
Principle of the Handle

Via normal reference,
normal execution



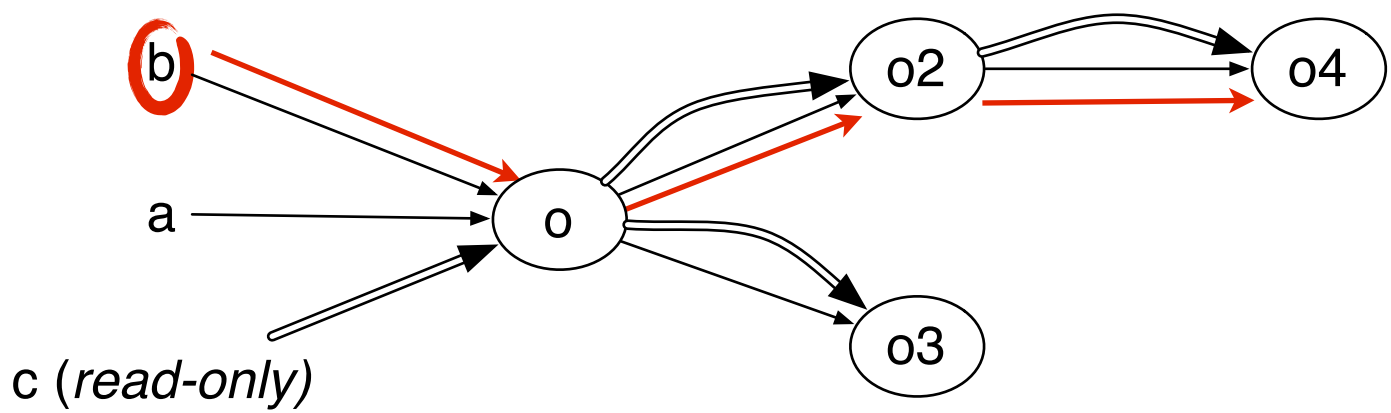
Principle of the Handle

Via normal reference,
normal execution



Principle of the Handle

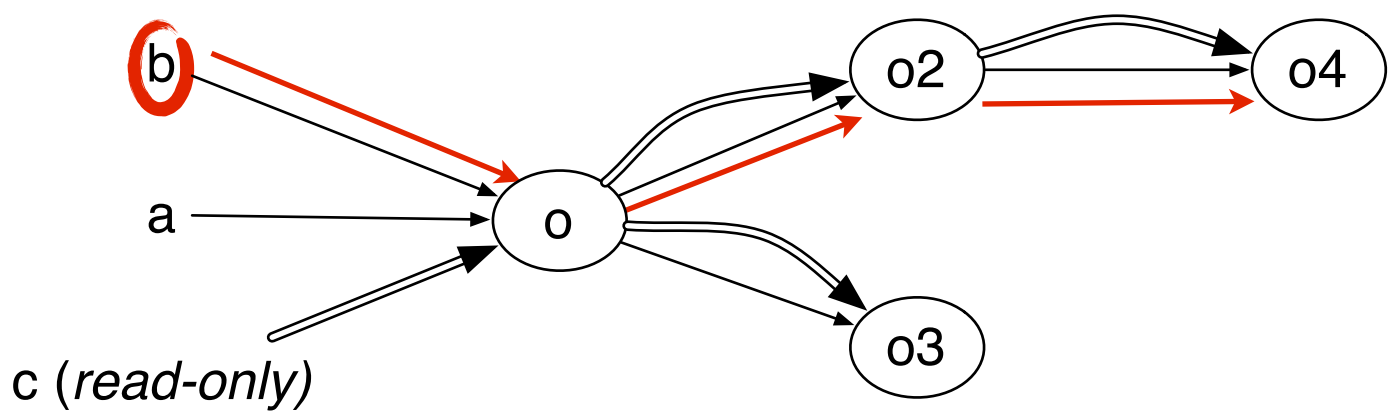
Via normal reference,
normal execution



Principle of the Handle

Via normal reference,
normal execution

State changed



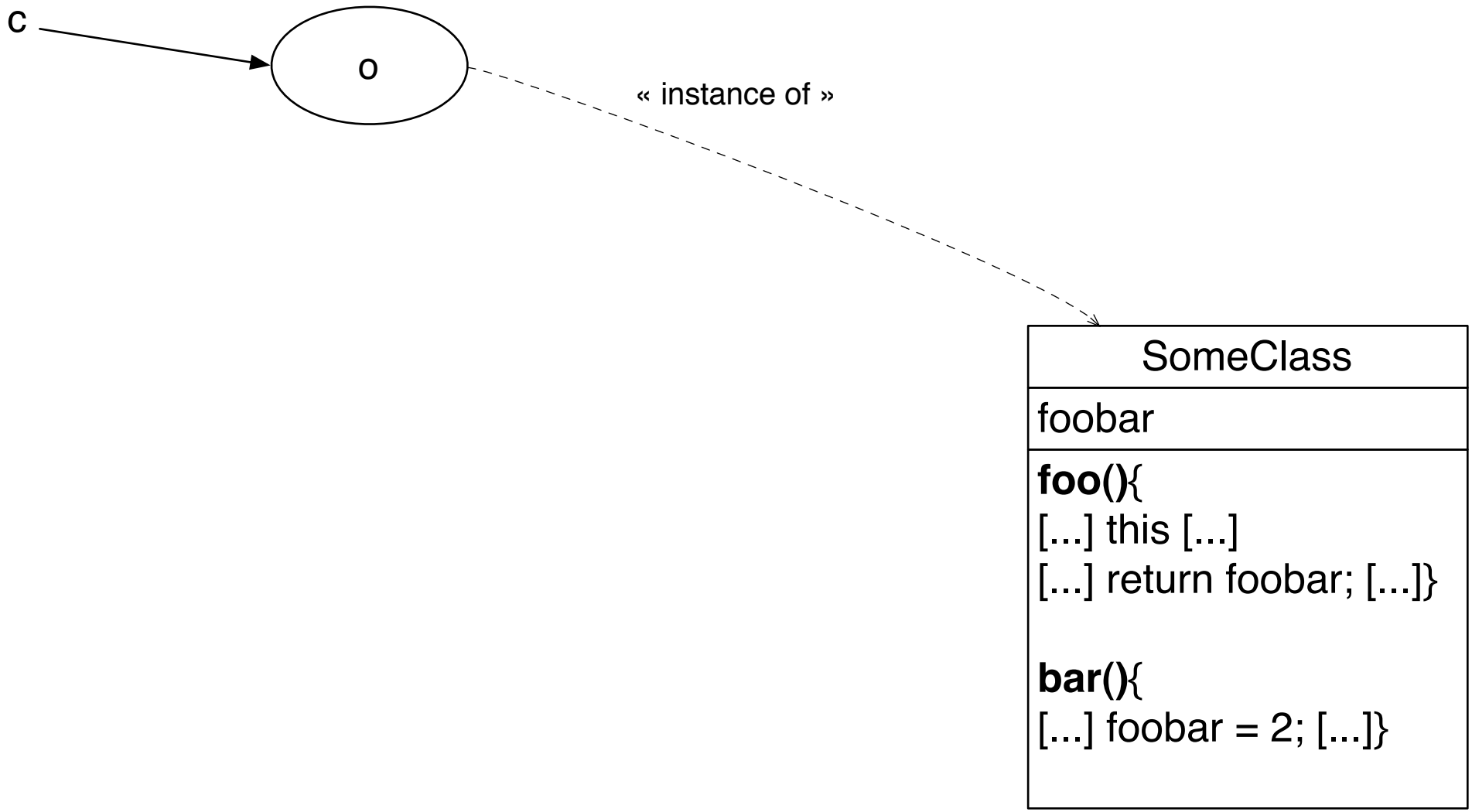
Implementation

- Bytecode rewriting to transform bytecodes of methods

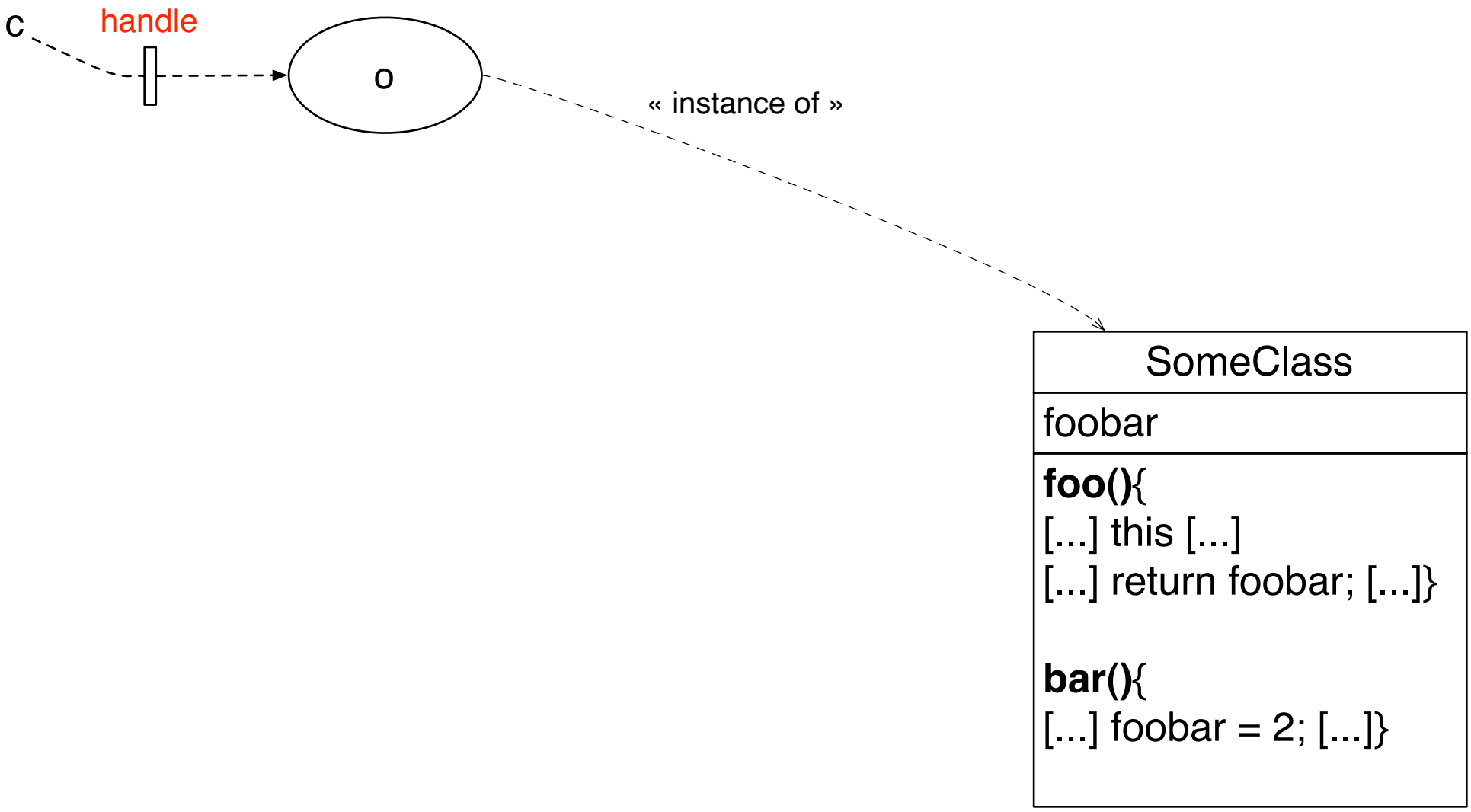
- VM modification to create the "Handle"
  - Change the lookup.
  - The Handle representing the object aliased (same identity)

- Cache that associates a class name to the rewrote methods

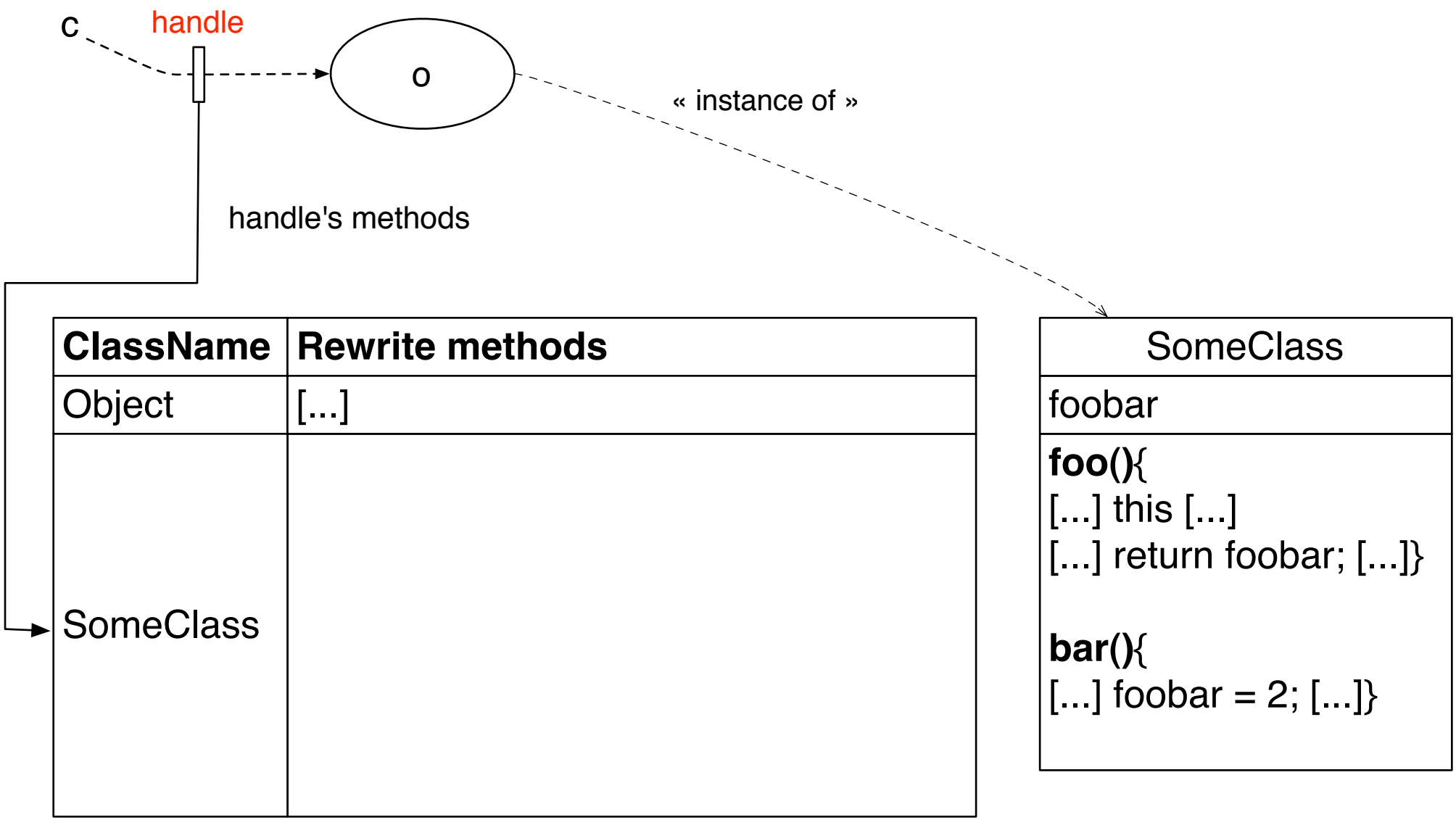
Rewriting Process



Rewriting Process

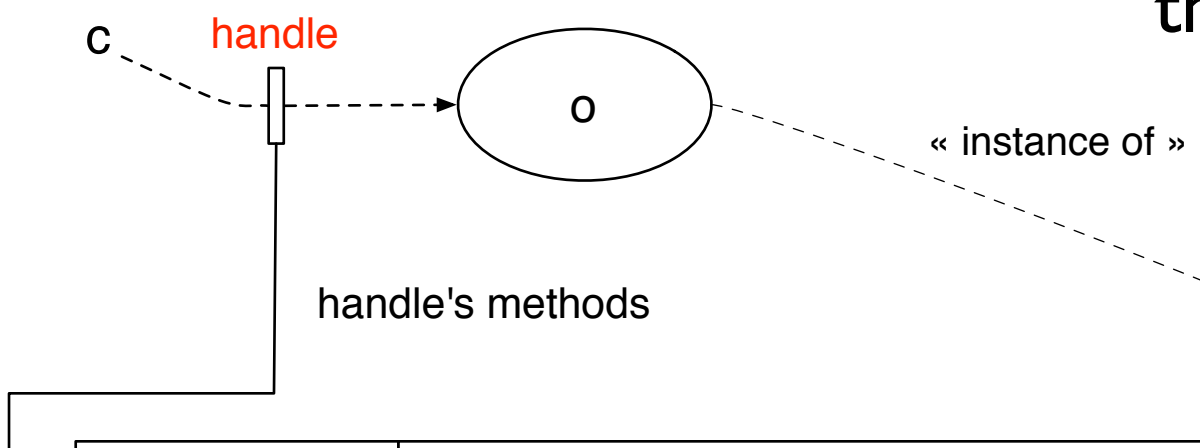


Rewriting Process



Rewriting Process

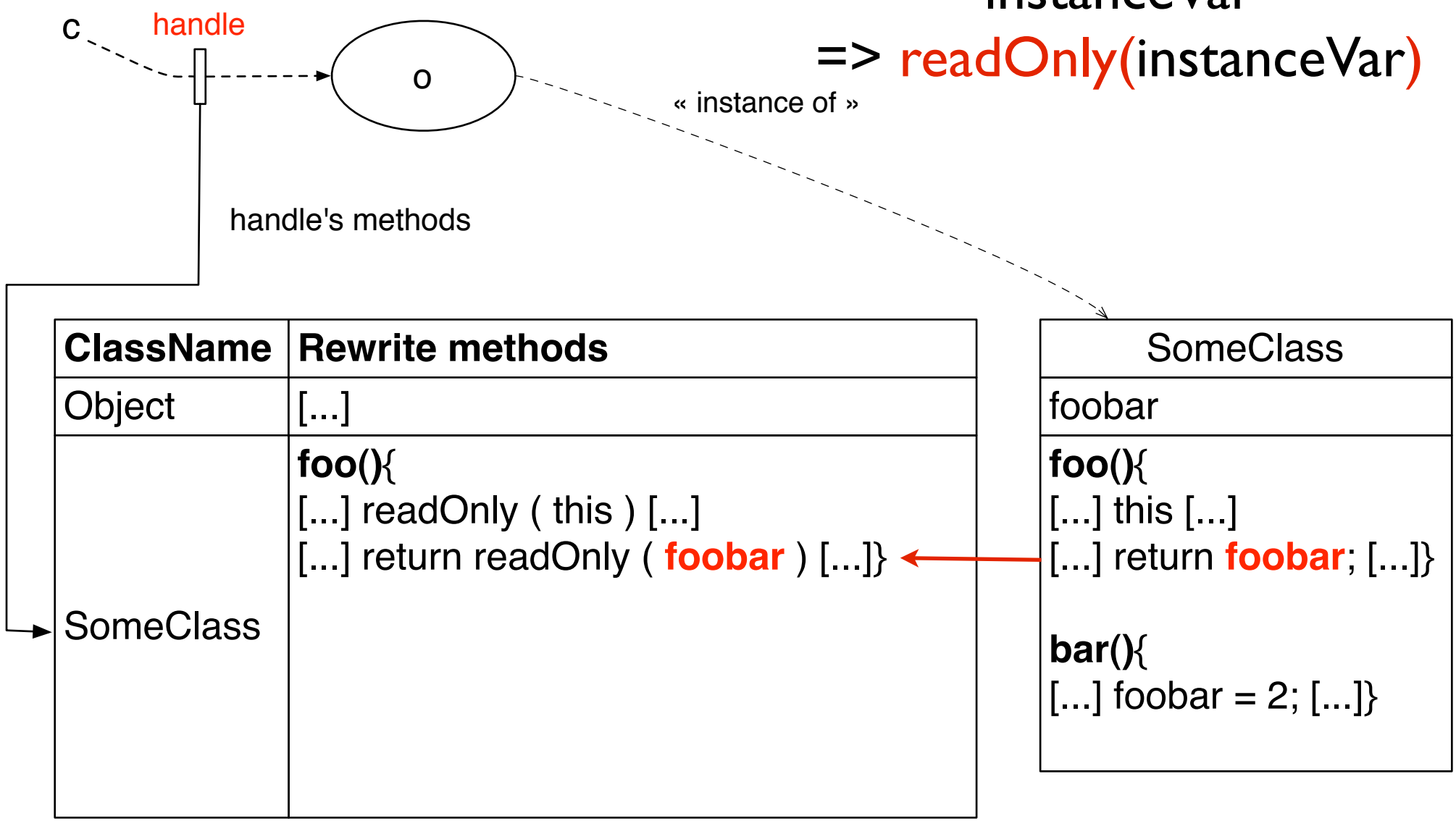
this => readOnly(this)



ClassName	Rewrite methods
Object	[...]
SomeClass	**foo(){** [...] readOnly (**this**) [...]

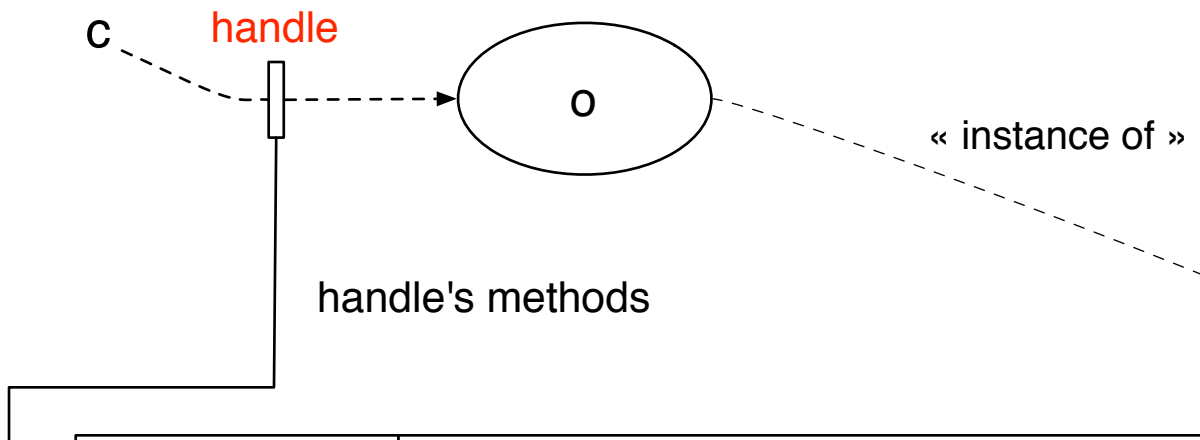
SomeClass
foobar
foo(){ [...] **this** [...] [...] return foobar; [...]} **bar(){** [...] foobar = 2; [...]}

Rewriting Process



Rewriting Process

a Method have Side effect => raise an Exception



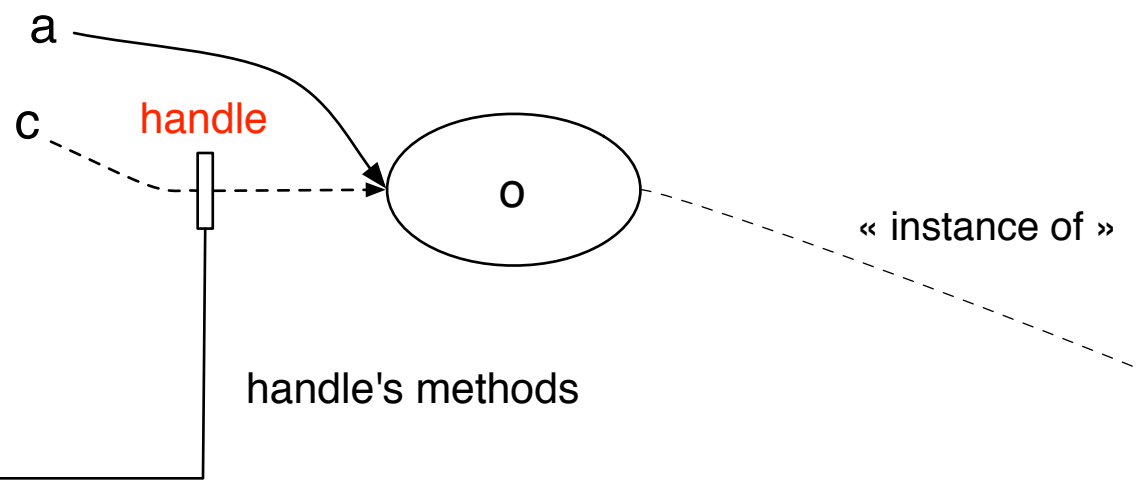
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] **foobar =** 2; [...]}

Rewriting Process



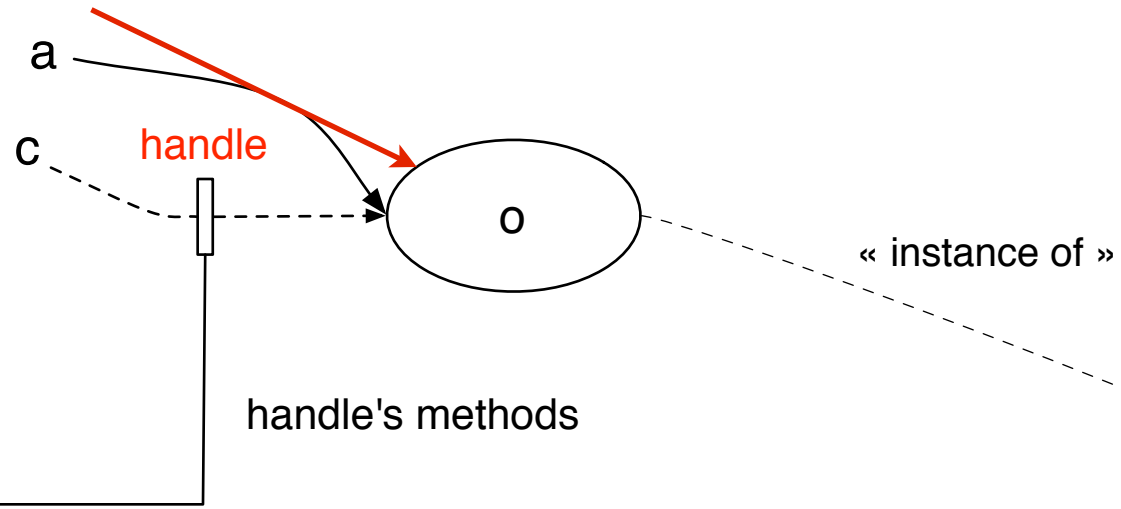
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Rewriting Process



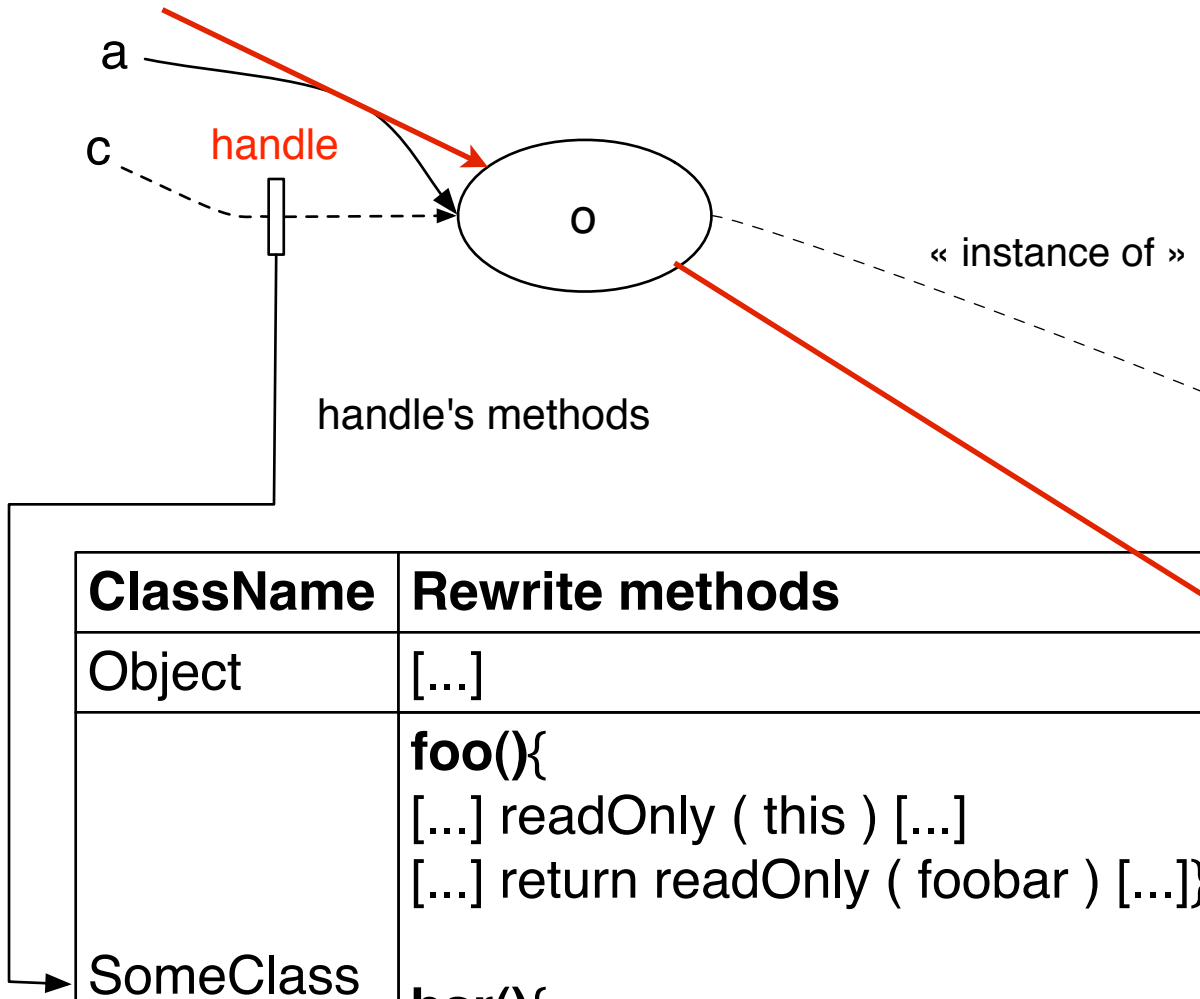
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Rewriting Process



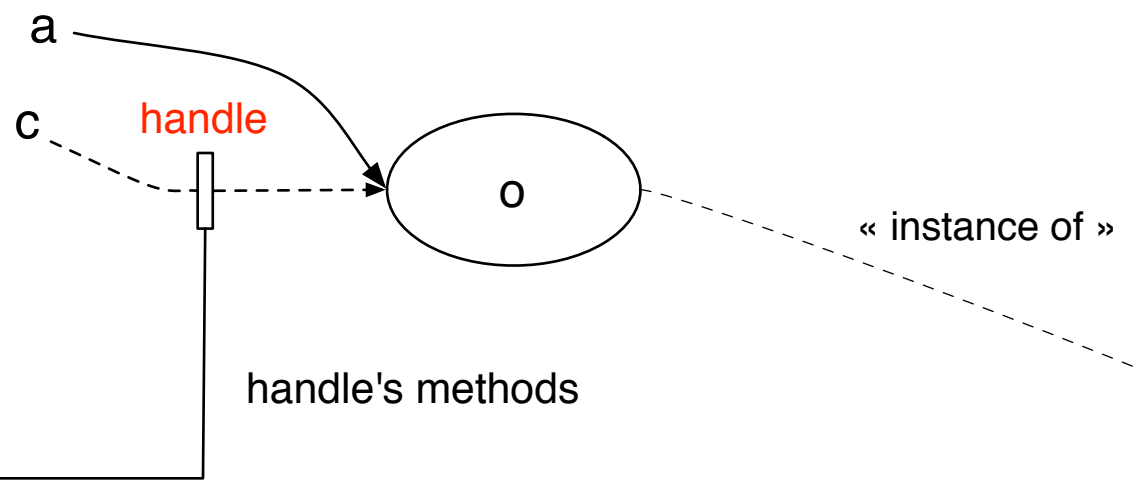
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Rewriting Process



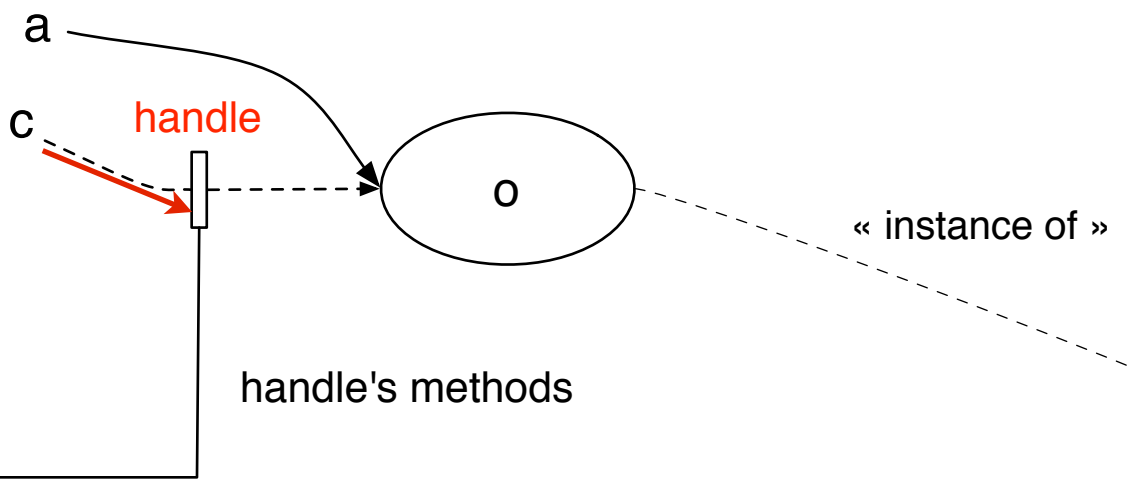
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Rewriting Process



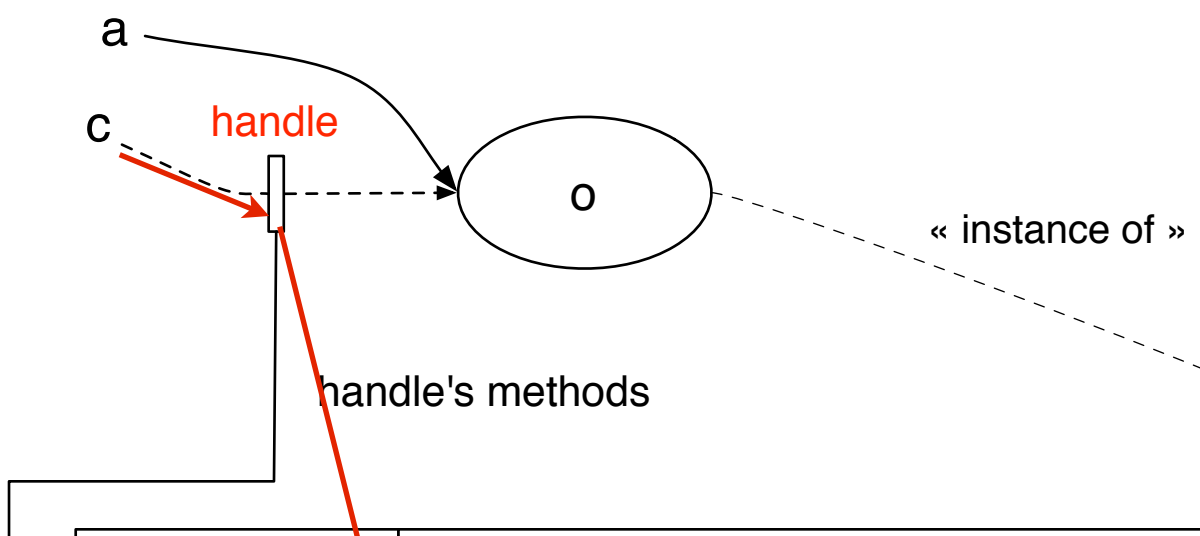
ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Rewriting Process



ClassName	Rewrite methods
Object	[...]

SomeClass	**foo()**{
	[...] readOnly (this) [...]
	[...] return readOnly (foobar) [...]}
	
	bar(){
	[...]throw new
	SideEffectDetectedException();[...]}

SomeClass
foobar

foo(){
[...] this [...]
[...] return foobar; [...]}
 	
bar(){
[...] foobar = 2; [...]}

Performance

- Proof-of-concept
- Bytecode rewriting : near 2.5 second for 450 methods
- Simple literal return, One millions send
  - Normal Virtual machine : 1109.3 ms
  - DRO Virtual Machine : 1200.5 ms
  - DRO Virtual Machine (to Handle) : 1290.1ms

Future Work

- Research
  - Generalize the idea
  - State
- Engineering
  - Primitives
  - Optimizing & Weak references

- ✔ Read-only Behavior
- ✔ Reference Based
- ✔ Propagation
- ✘ Change the object graph
- ✘ Static analysis
- ✘ Freeze the object