Towards First Class References as a Security Infrastructure in Dynamically-Typed Languages

Jean-Baptiste Arnaud

Supervisor: Stéphane Ducasse
Co-Supervisor: Marcus Denker
RMoD team

February 18, 2013

Dynamically-typed languages

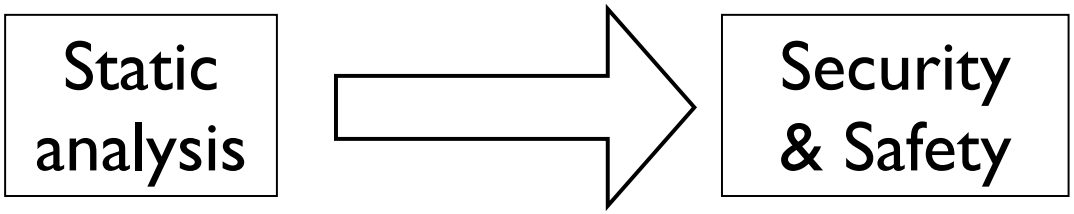
python™ JavaScript Ruby *A Programmer's Best Friend*

Dynamically-typed languages are becoming popular

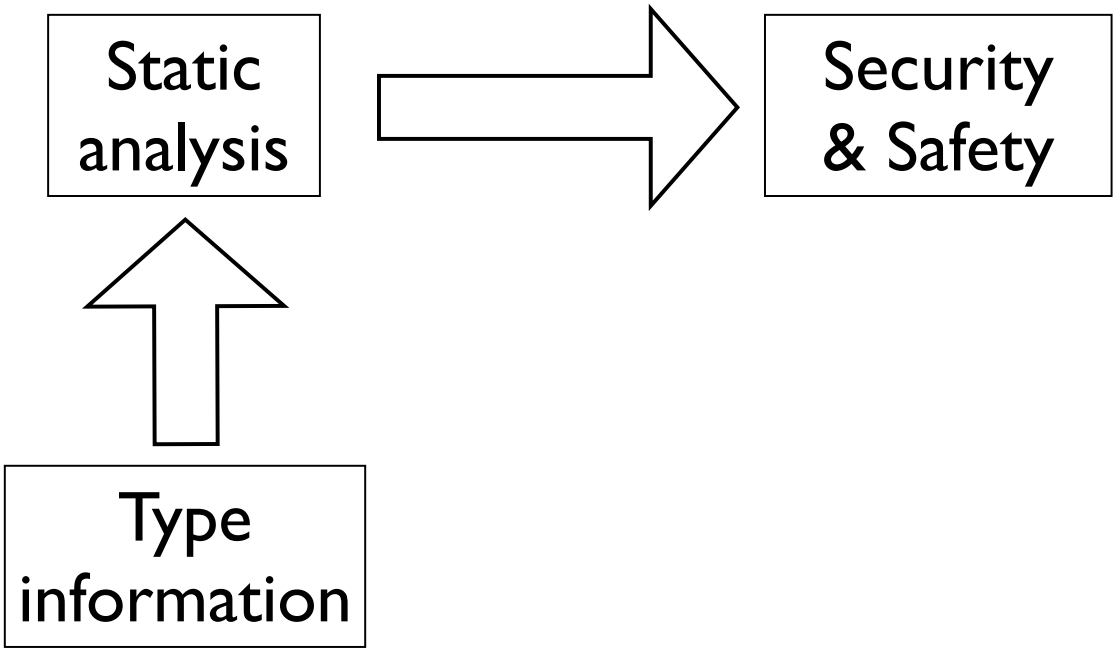
Need

Security & Safety

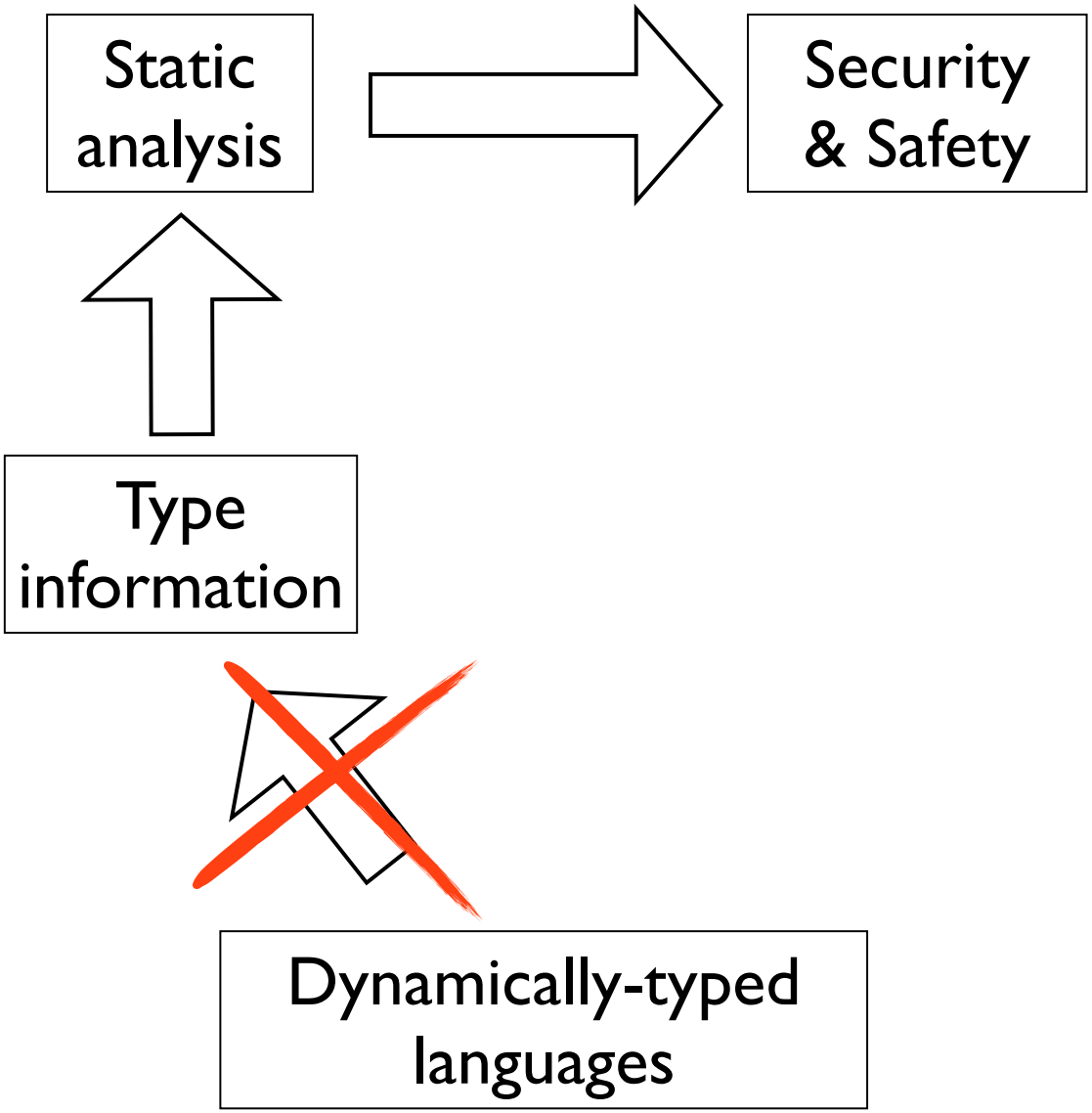
Security and Safety



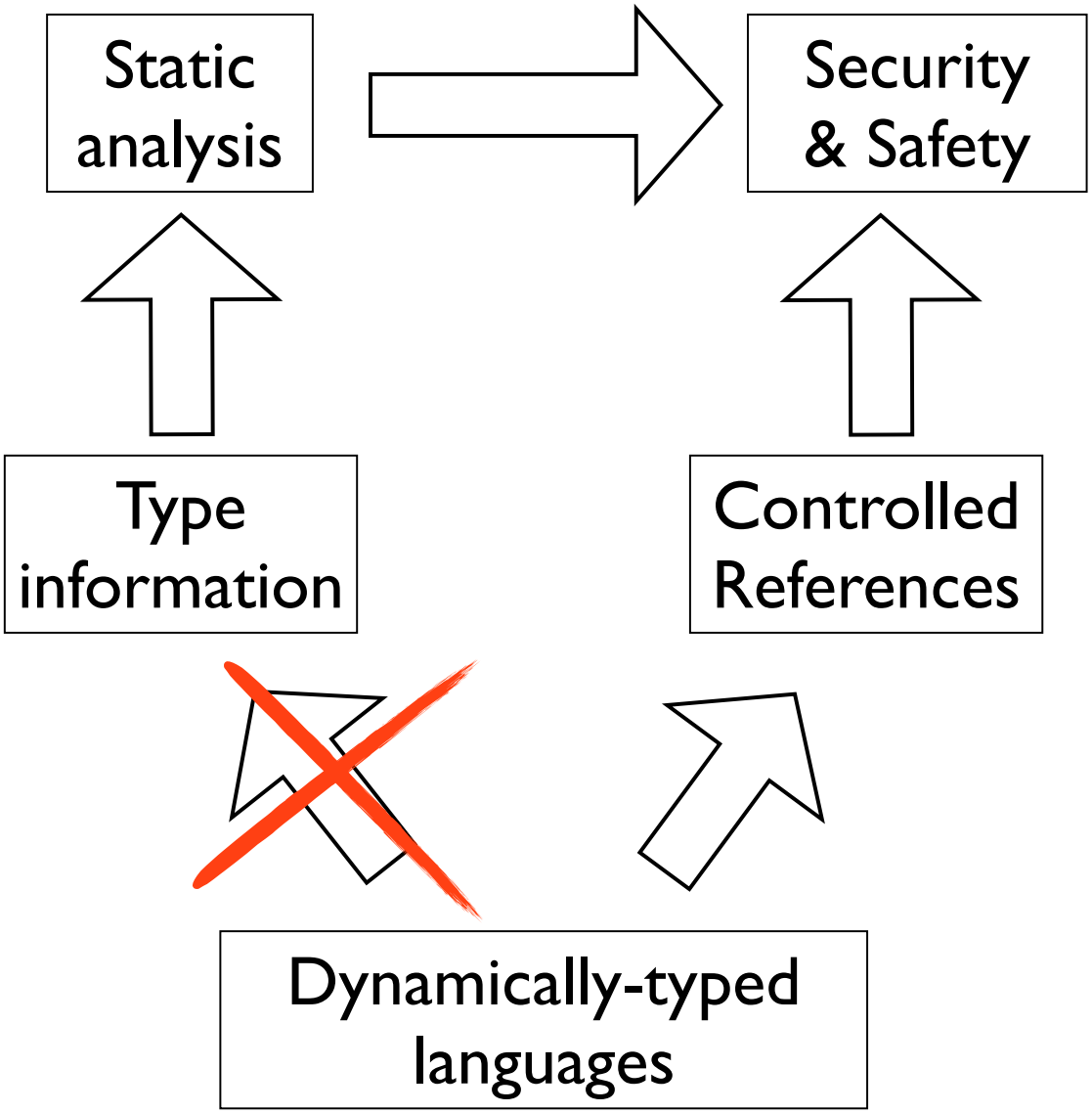
Security and Safety



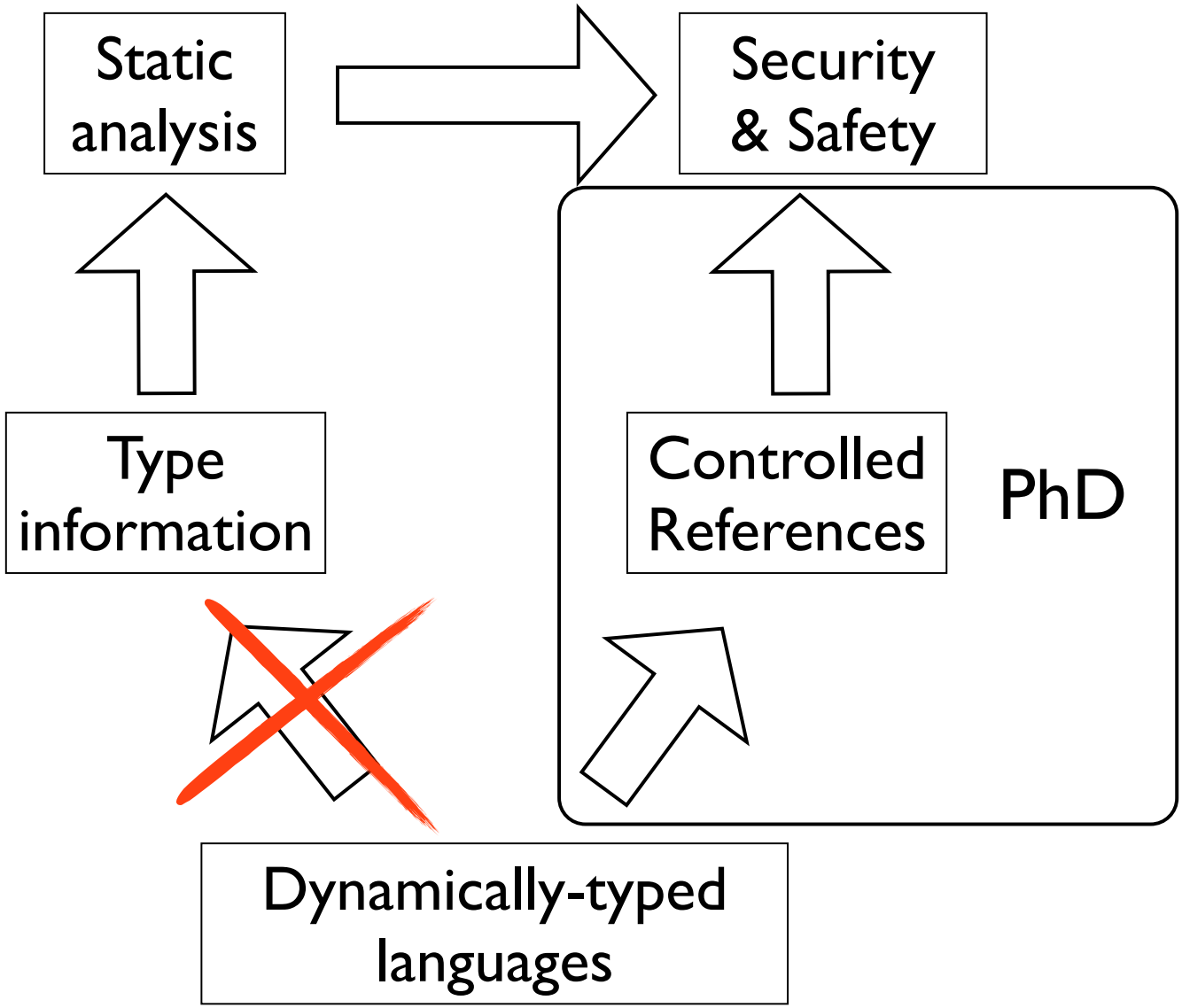
Security and Safety



Security and Safety



Security and Safety



Problem

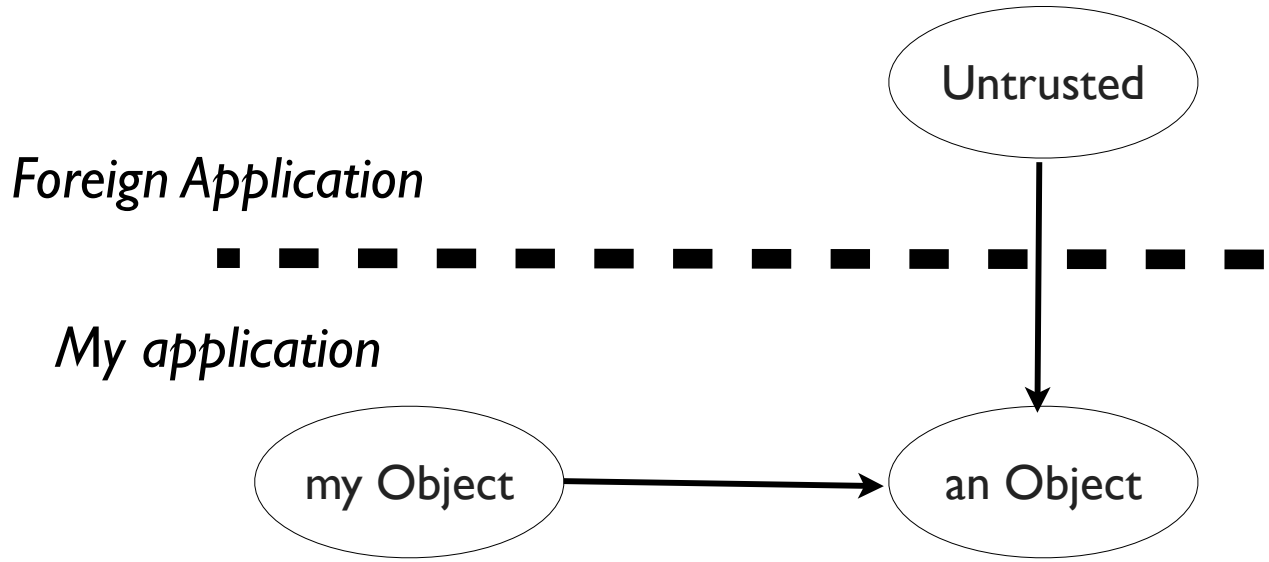
How can we provide a *general-purpose* way to control references in dynamically-typed languages?

Outline

Use cases for controlled references

Side effect management:

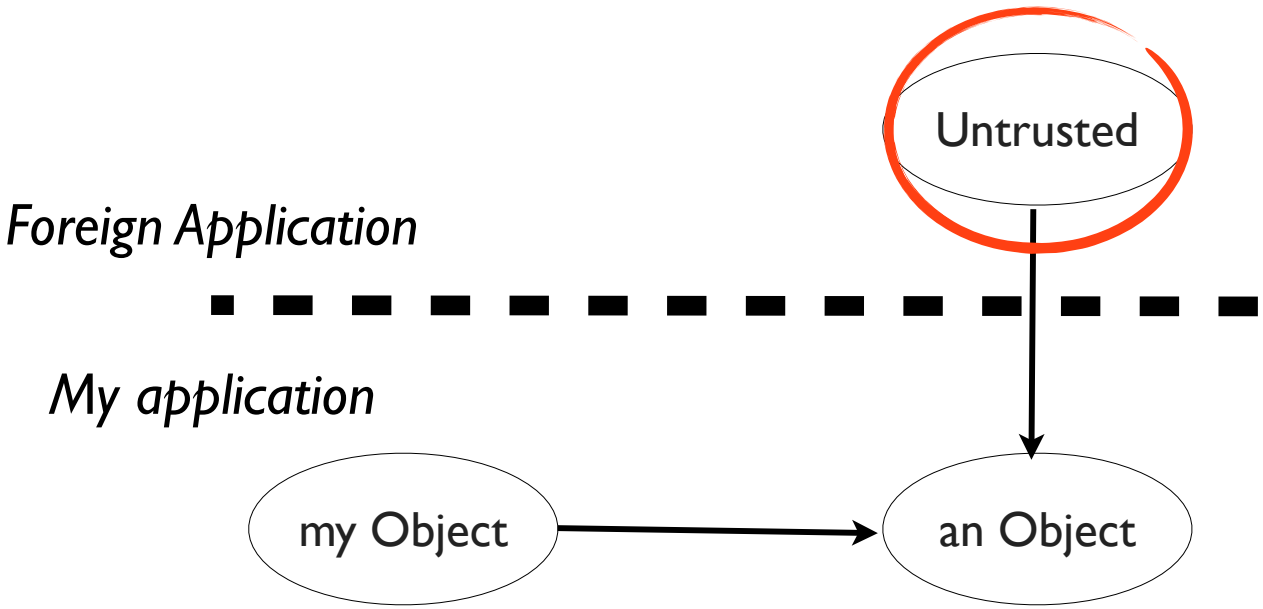
- Prevent side effect from occurring
- Accept side effect and check them



Use cases for controlled references

Side effect management:

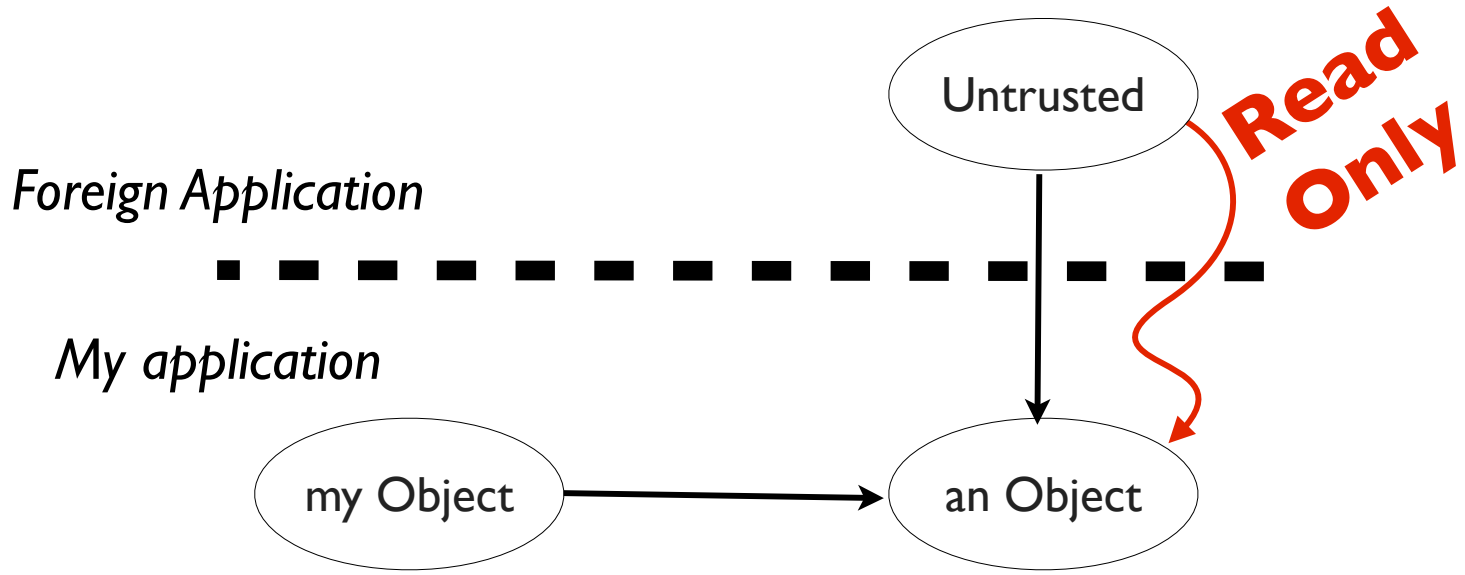
- Prevent side effect from occurring
- Accept side effect and check them



Use cases for controlled references

Side effect management:

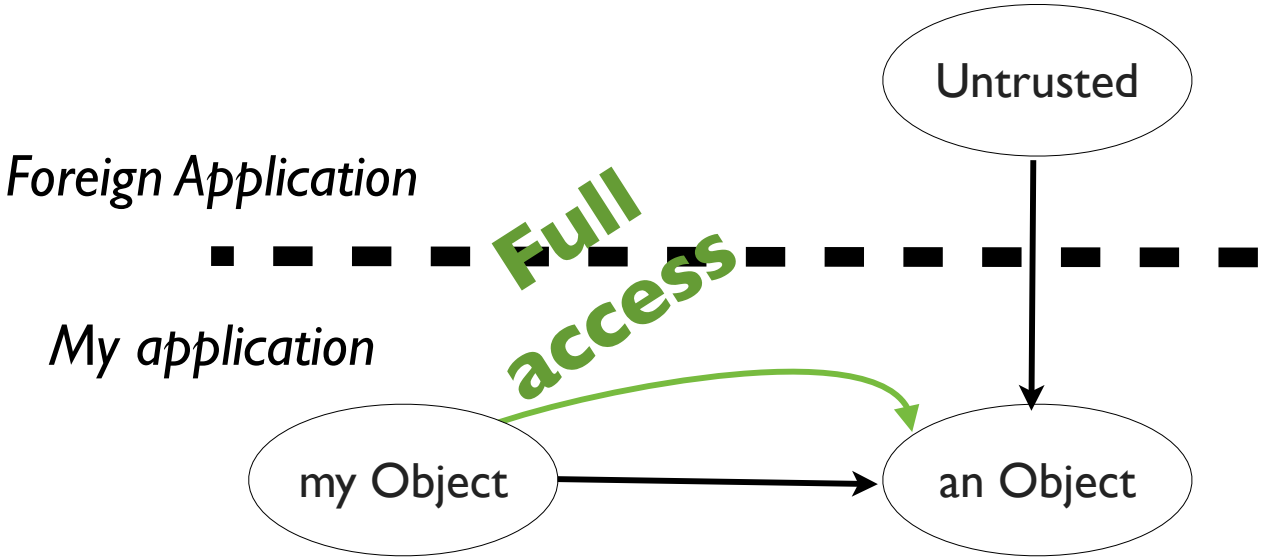
- Prevent side effect from occurring: ReadOnly Execution
- Accept side effect and check them



Use cases for controlled references

Side effect management:

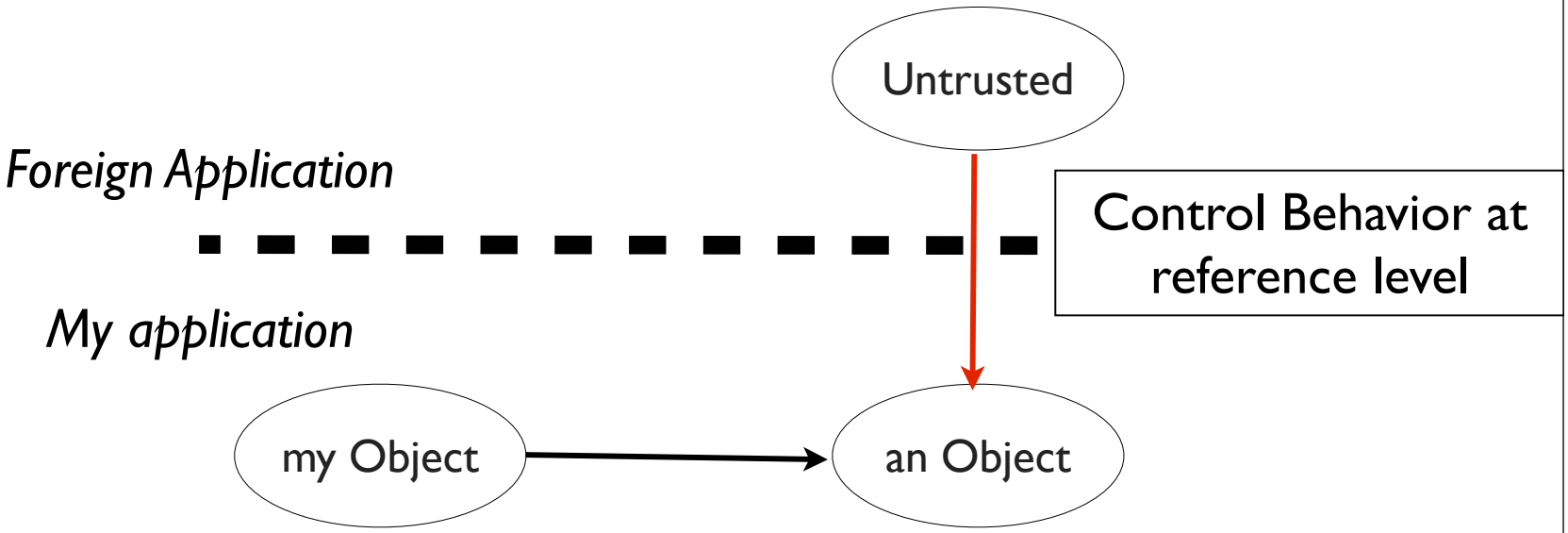
- Prevent side effect from occurring: ReadOnly Execution
- Accept side effect and check them



Use cases for controlled references

Side effect management:

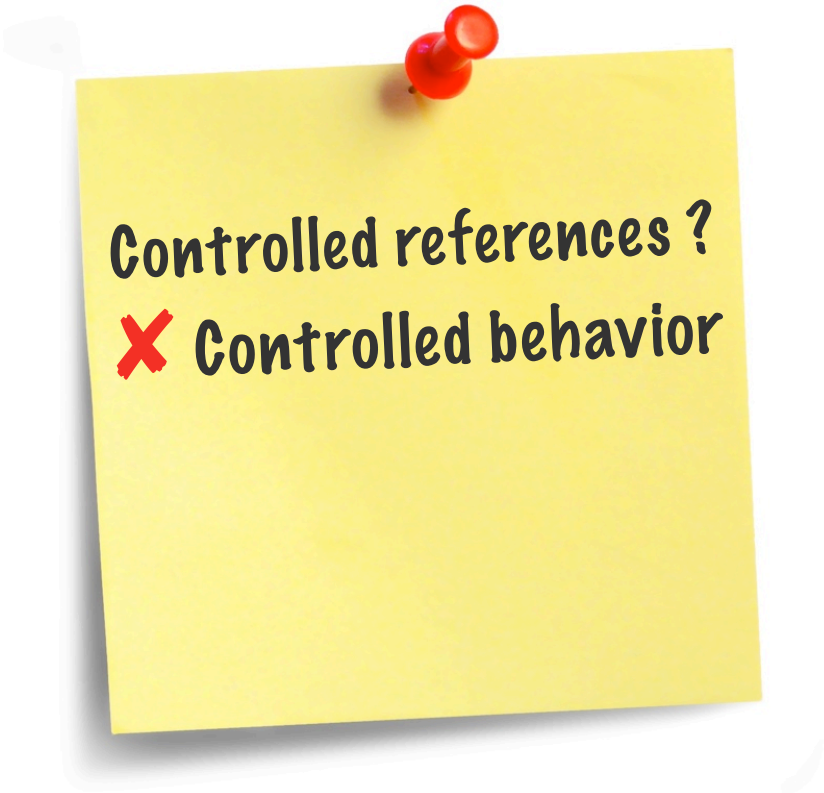
- Prevent side effect from occurring: ReadOnly Execution
- Accept side effect and check them



Use cases for controlled references

Side effect management:

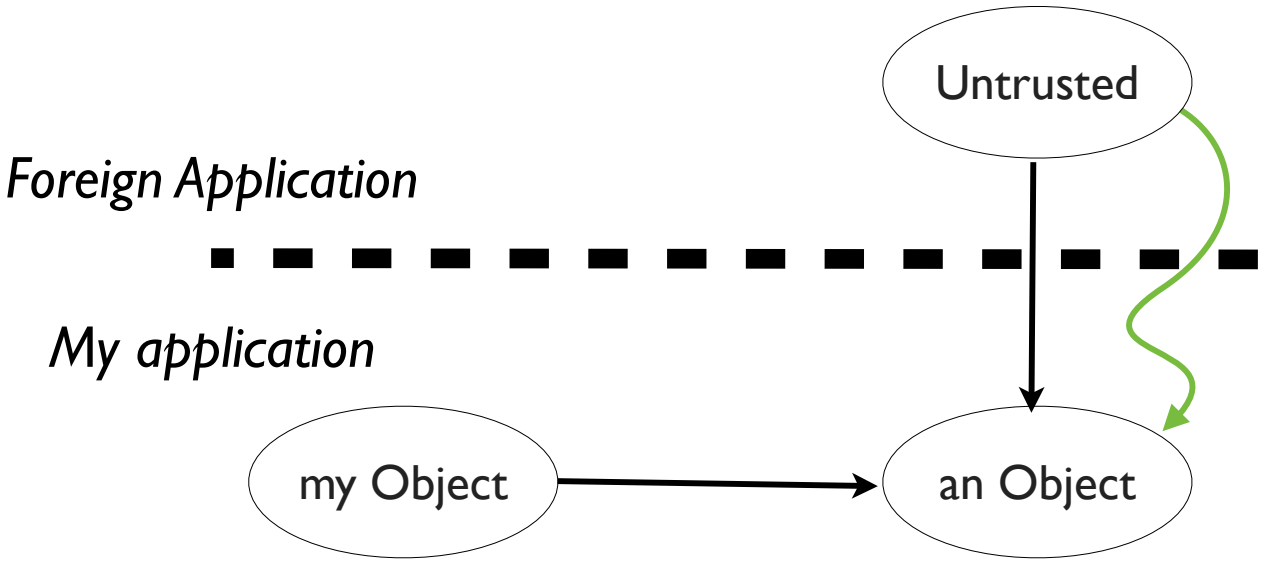
- Prevent side effect from occurring
- Accept side effect and check them



Use cases for controlled references

Side effect management:

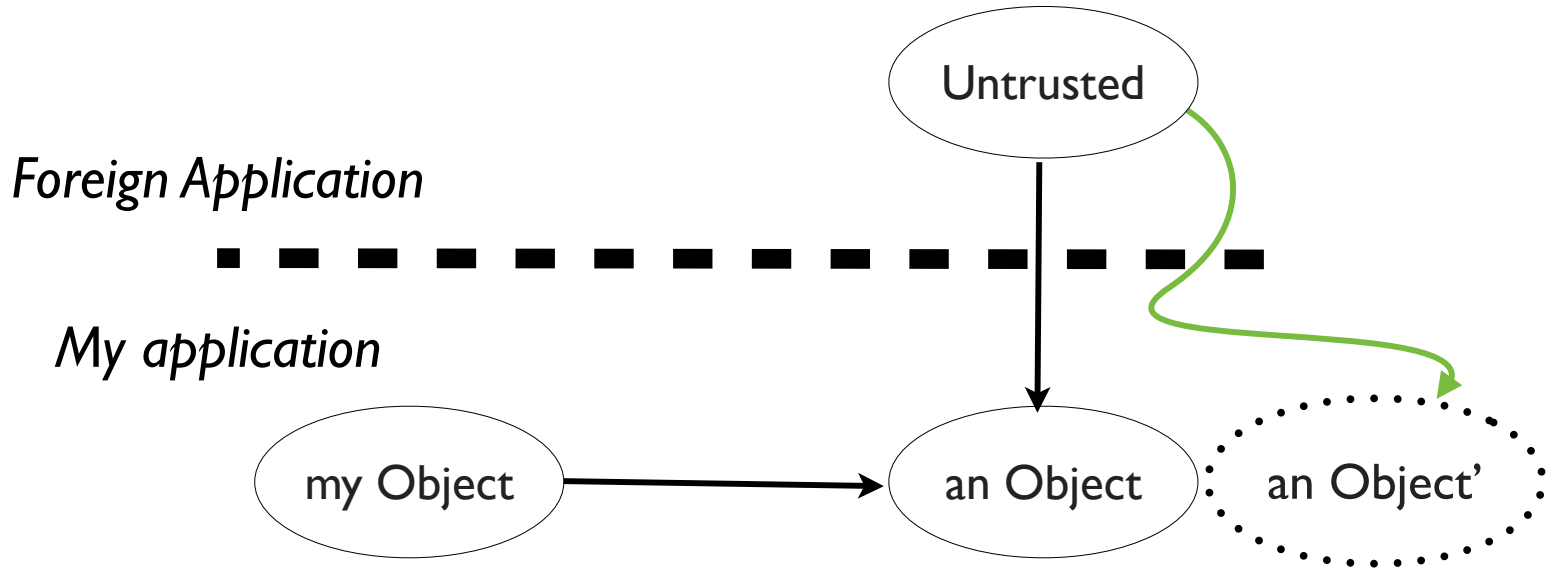
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



Use cases for controlled references

Side effect management:

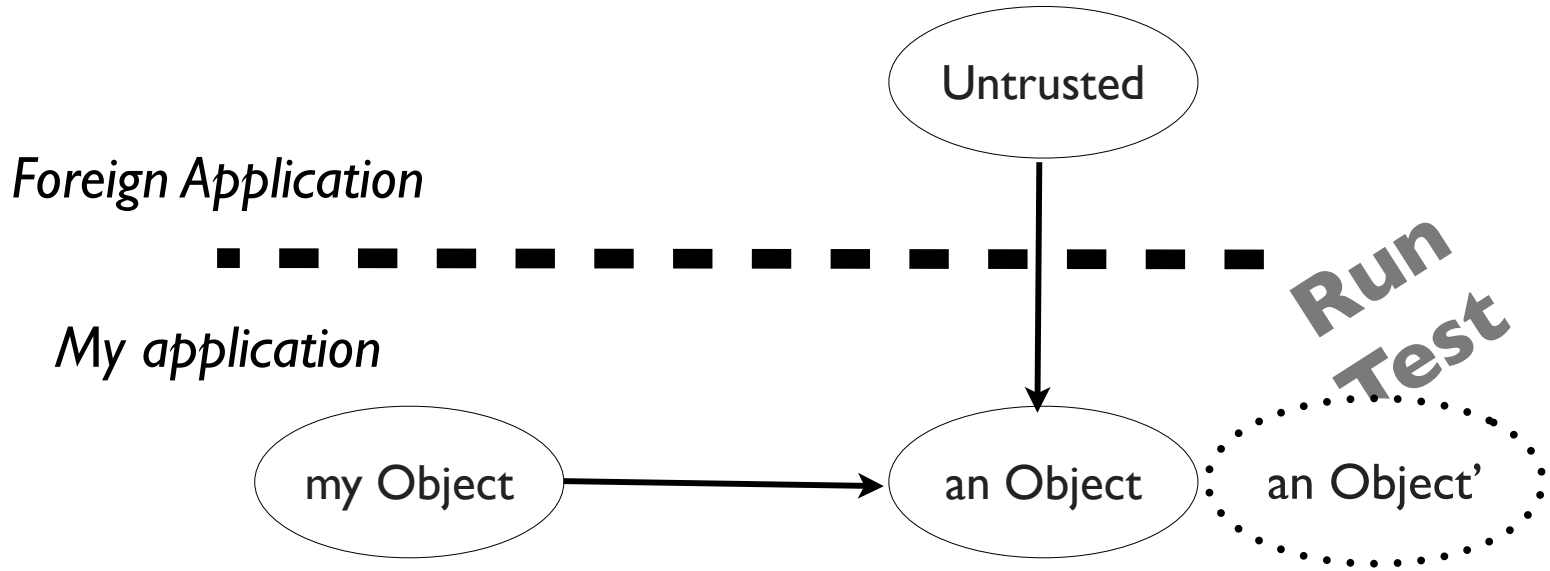
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



Use cases for controlled references

Side effect management:

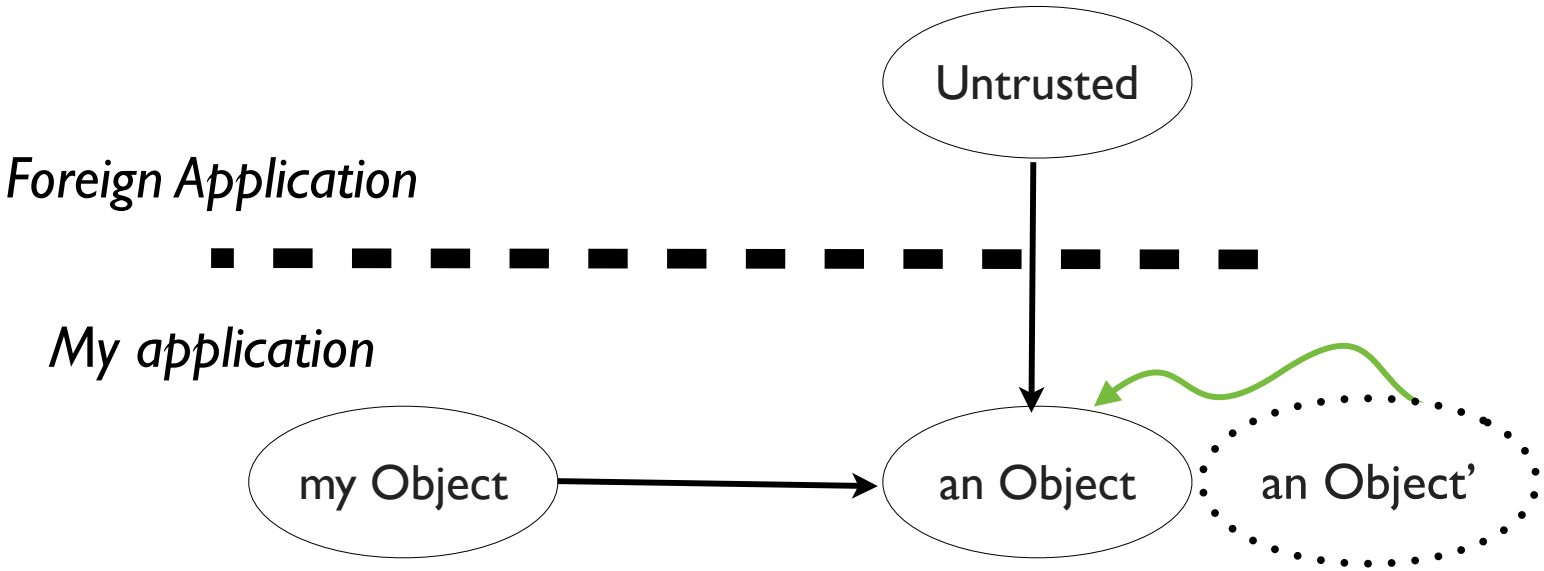
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



Use cases for controlled references

Side effect management:

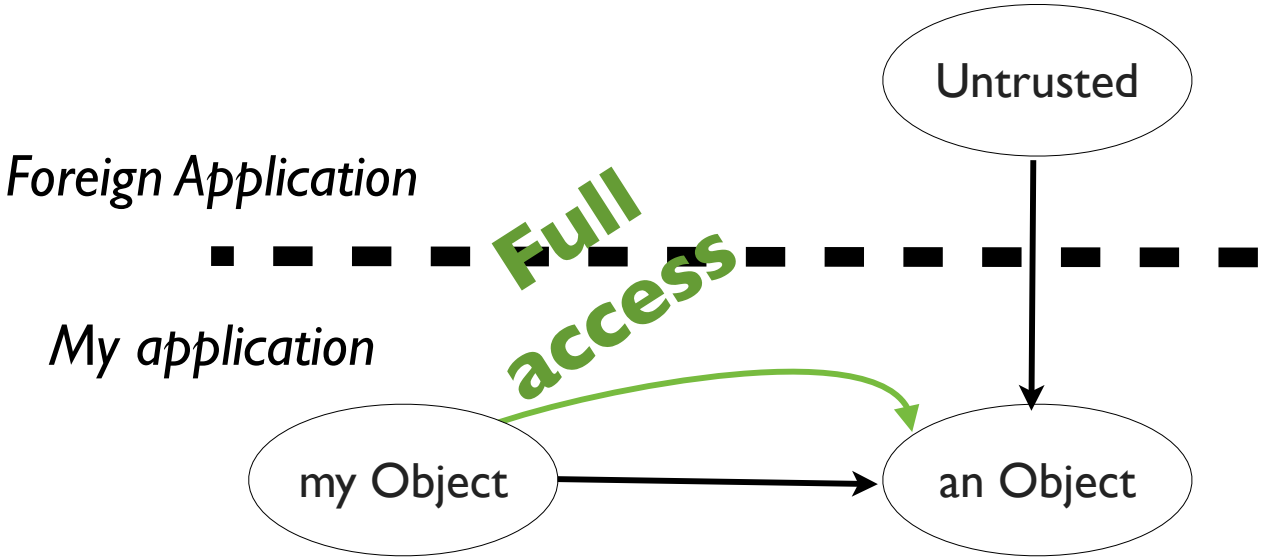
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



Use cases for controlled references

Side effect management:

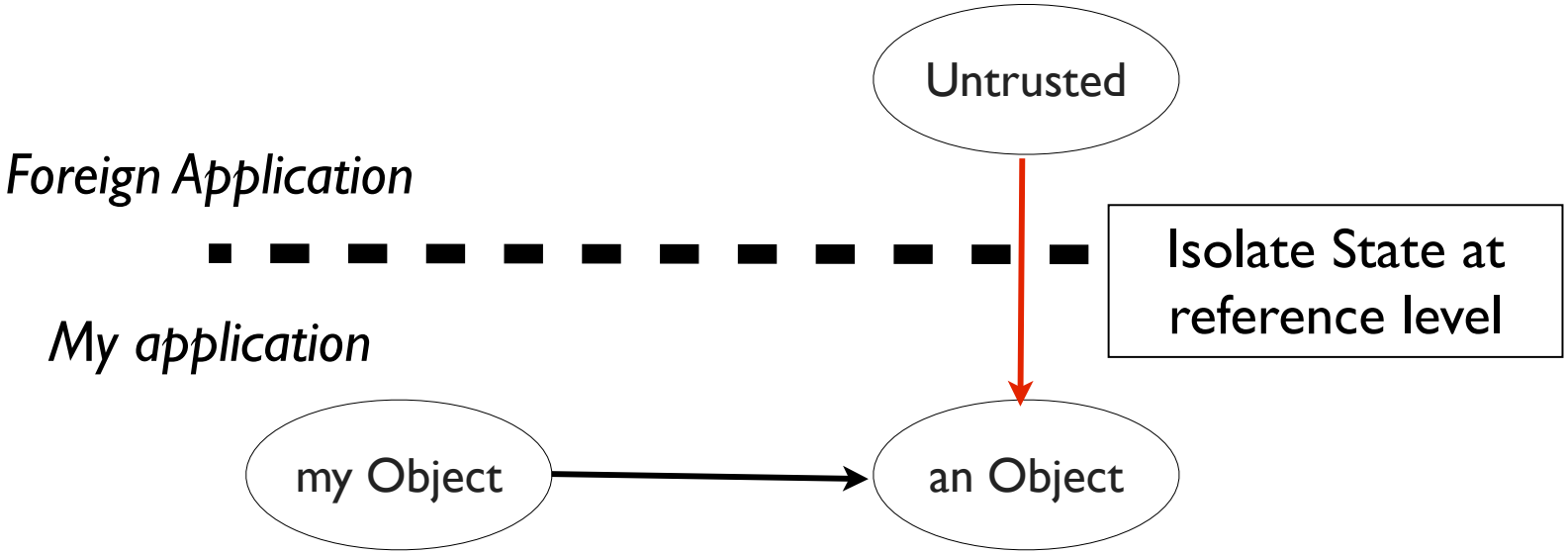
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



Use cases for controlled references

Side effect management:

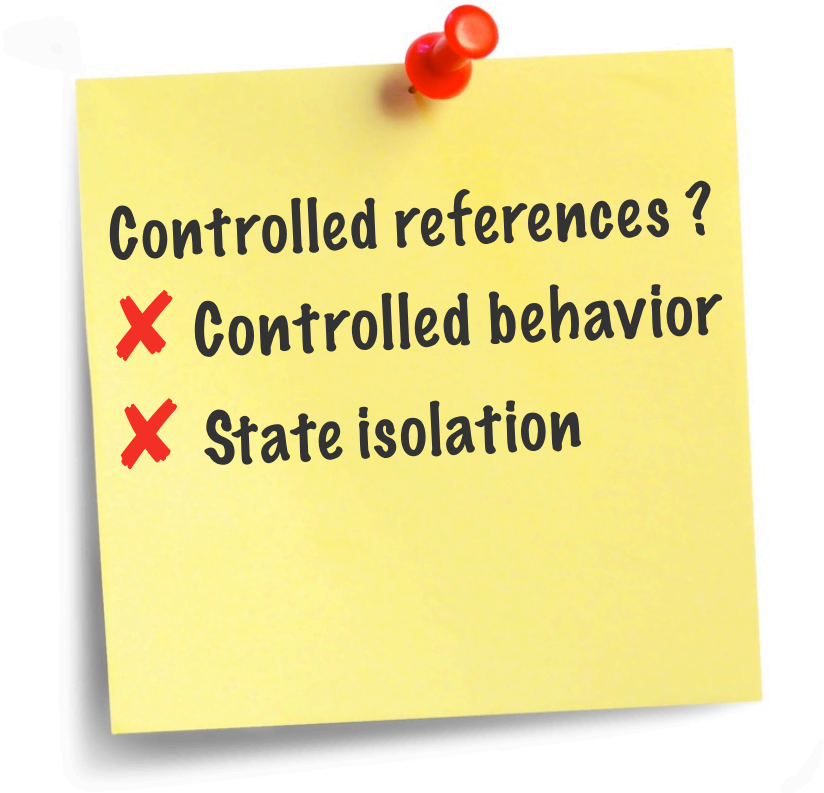
- Prevent side effect from occurring
- Accept side effect and check them: Isolated Execution



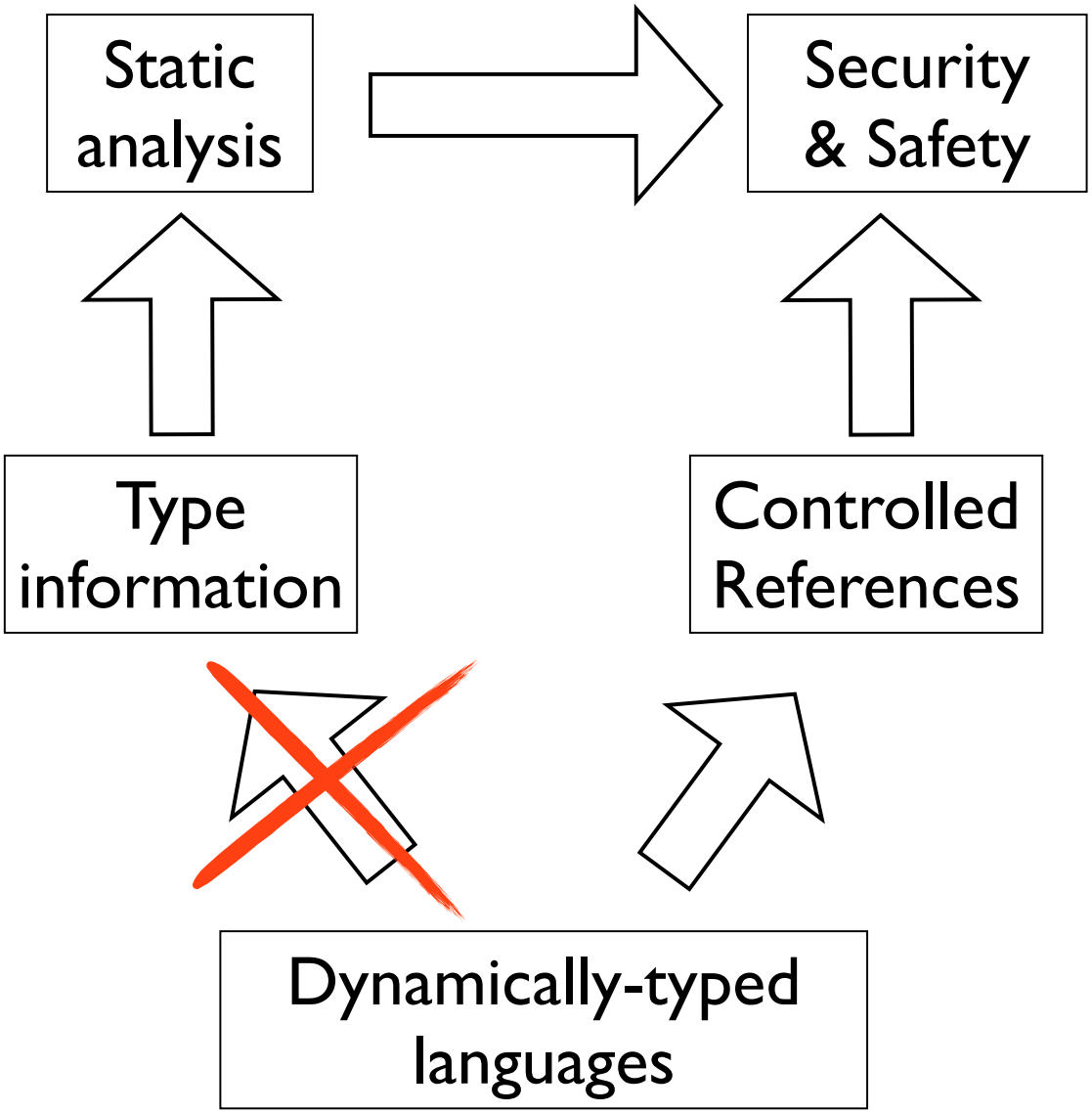
Use cases for controlled references

Side effect management:

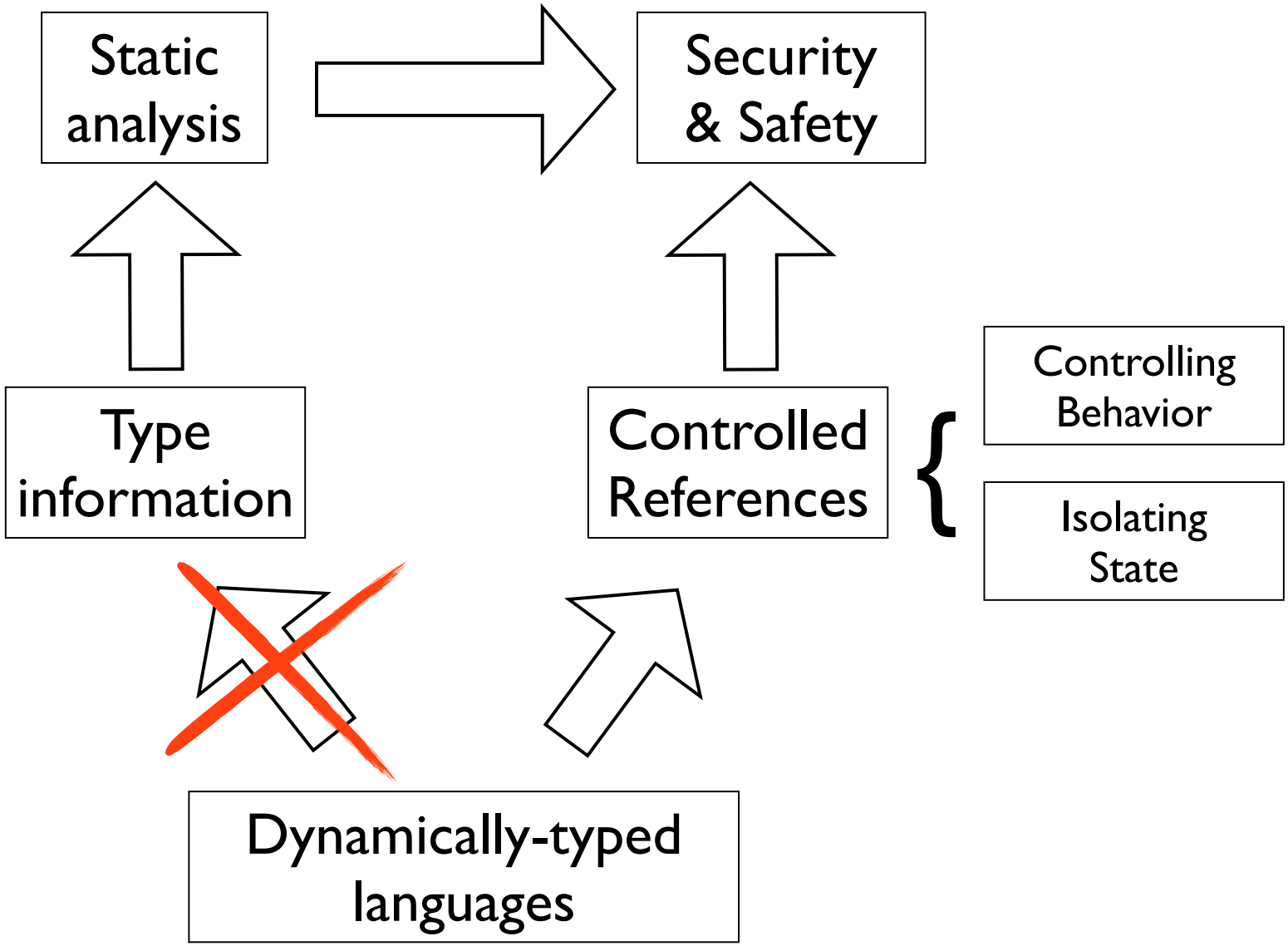
- Prevent side effect from occurring
- Accept side effect and check them



Security and Safety



Security and Safety



Controlled references

Use references to control the semantics of target object

Family's solution	Behavior control	State isolation
Object-oriented capabilities	Only Restrict behavior	No
Encapsulation	Only Restrict behavior	No
Dynamic ownership	Only Restrict behavior	No
Contextual value	No	Yes per thread

Controlled references

Use references to control the semantics of target object

Family's solution	Behavior control	State isolation
Object-oriented capabilities	Only Restrict behavior	No
Encapsulation	Only Restrict behavior	No
Dynamic ownership	Only Restrict behavior	No
Contextual value	No	Yes per thread

Mark Samuel Miller et al.
IAFOR'2003

Controlled references

Use references to control the semantics of target object

Family's solution	Behavior control	State isolation
Object-oriented capabilities	Only Restrict behavior	No
Encapsulation	Only Restrict behavior	No
Dynamic ownership	Only Restrict behavior	No
Contextual value	No	Yes per thread

Nathanael Schärli et al. OOPSLA'04

Controlled references

Use references to control the semantics of target object

Family's solution	Behavior control	State isolation
Object-oriented capabilities	Only Restrict behavior	No
Encapsulation	Only Restrict behavior	No
Dynamic ownership	Only Restrict behavior	No
Contextual value	No	Yes per thread

Gordon et al.
DLS'2007

Controlled references

Use references to control the semantics of target object

Family's solution	Behavior control	State isolation
Object-oriented capabilities	Only Restrict behavior	No
Encapsulation	Only Restrict behavior	No
Dynamic ownership	Only Restrict behavior	No
Contextual value	No	Yes per thread

Eric Tanter
DLS'2008

Problem

How can we provide a *general-purpose* way to control references in dynamically-typed languages?

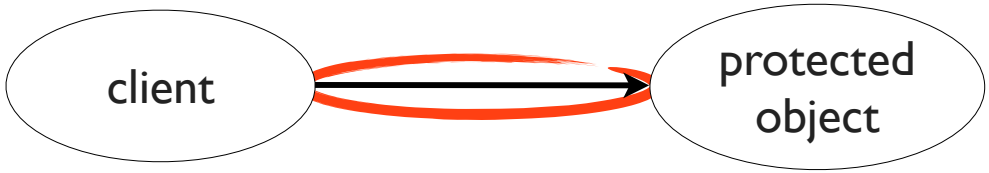
Thesis

In the context of dynamically-typed languages, reifying references, controlling behavior, and isolating state via such references, is a practical way to control references

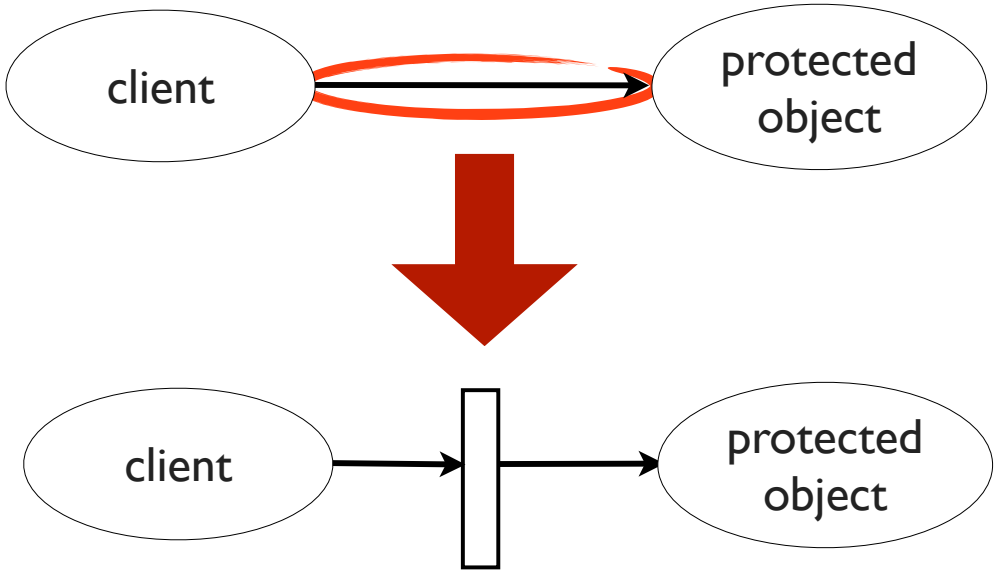
Solution

- ✗ Reifying references
  - ✗ Controlling behavior
  - ✗ Isolating state

Reifying references



Reifying references



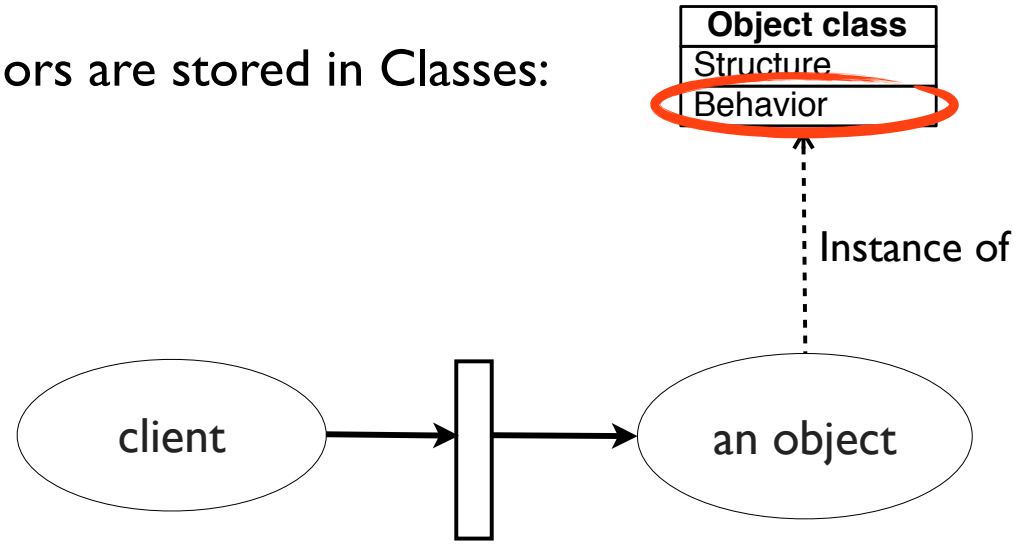
Handle

Solution


✔ Reifying references
✘ Controlling behavior
✘ Isolating state

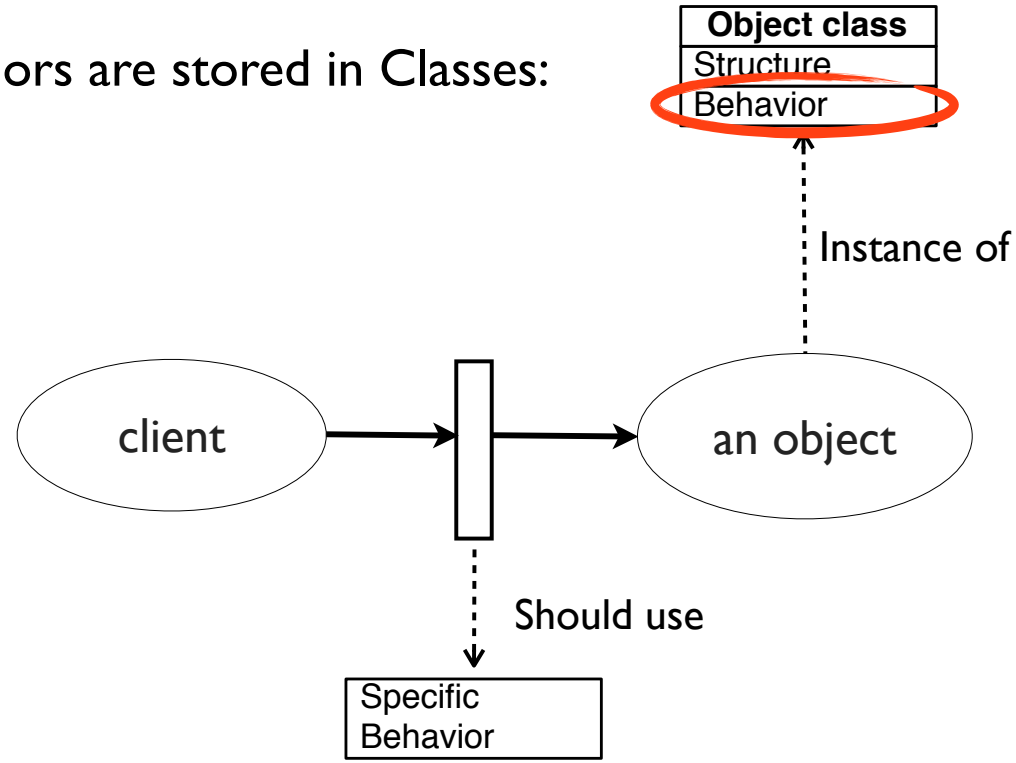
Controlling behavior

Behaviors are stored in Classes:

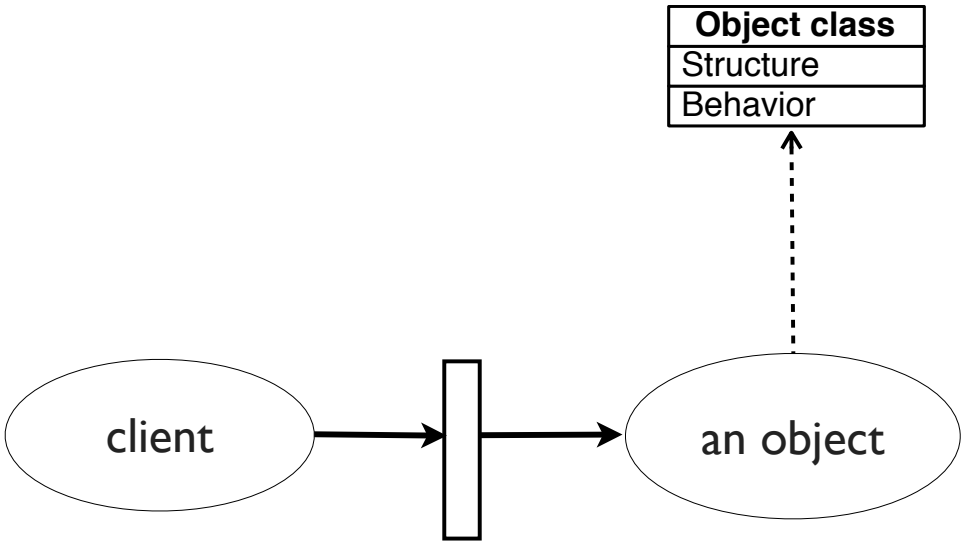


Controlling behavior

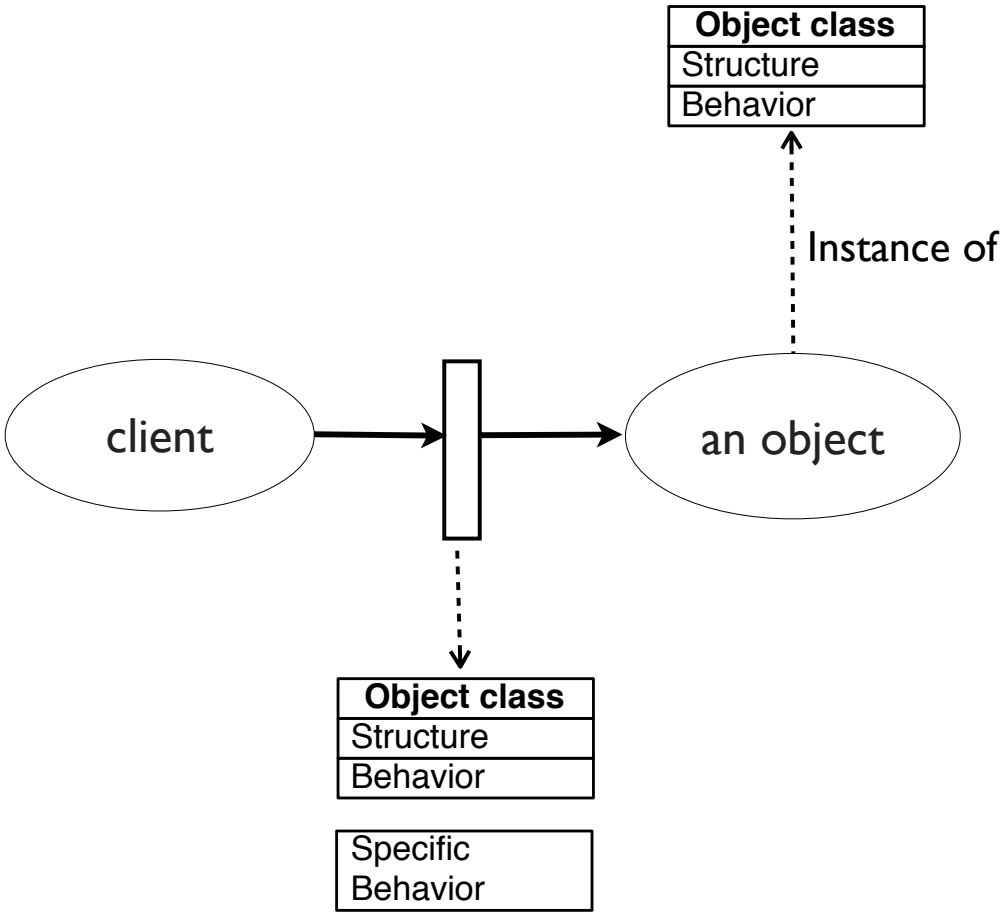
Behaviors are stored in Classes:



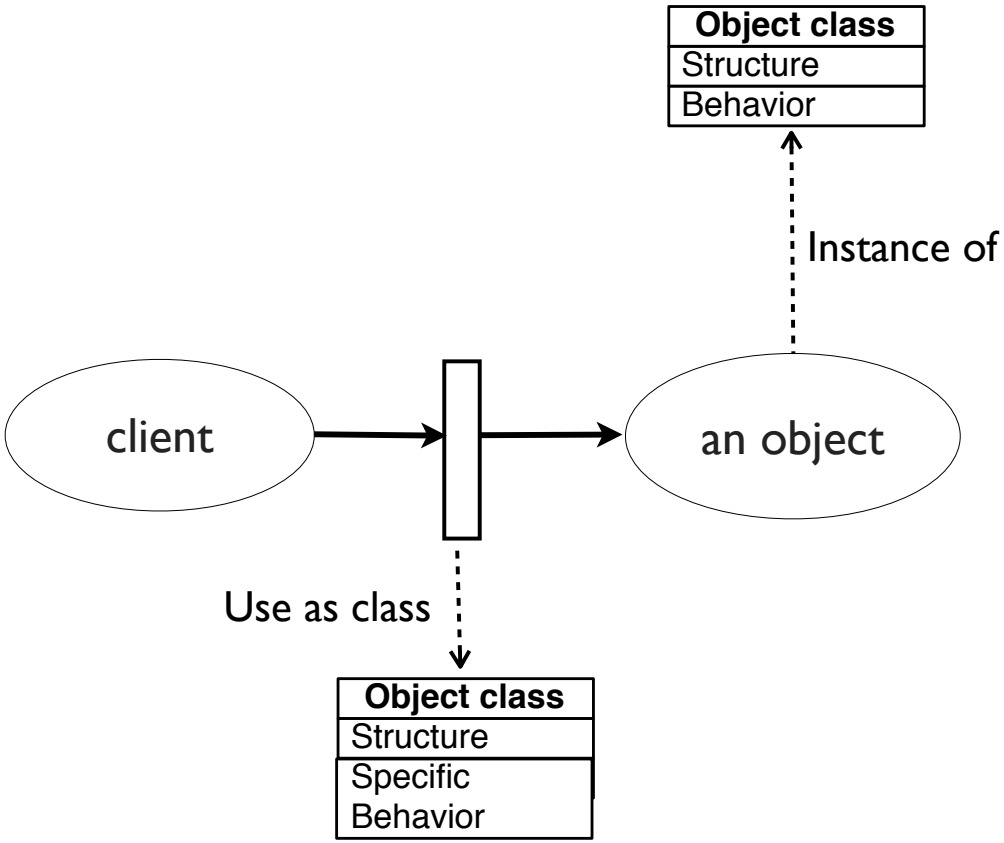
Controlling behavior



Controlling behavior



Controlling behavior

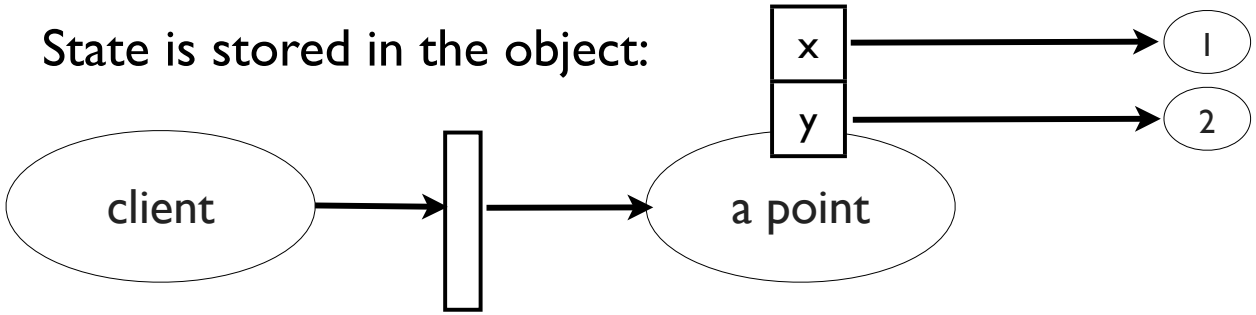


Solution

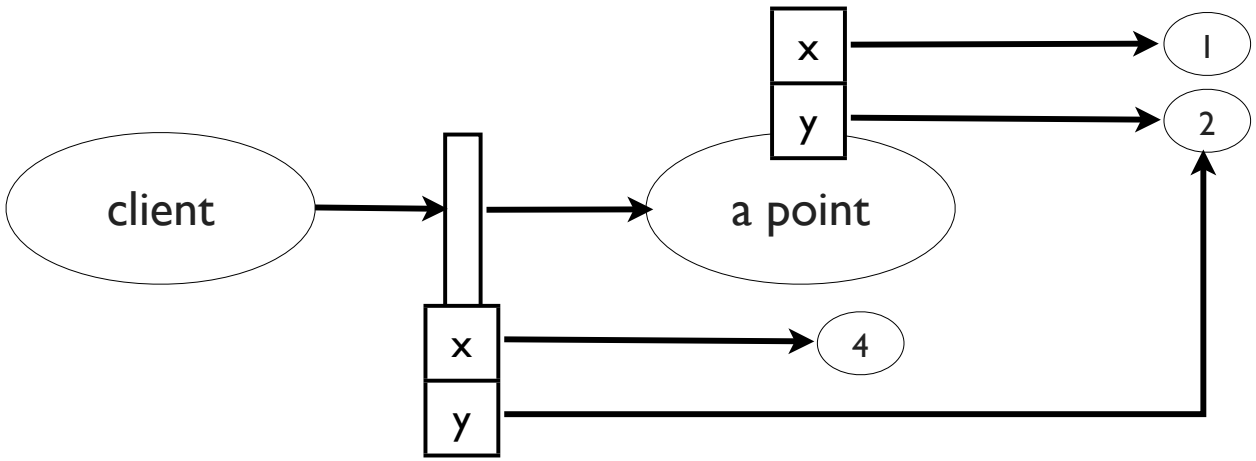

✔ Reifying references
✔ Controlling behavior
✘ Isolating state

Isolating State

State is stored in the object:



Isolating State




Core
✔ Reifying references
✔ Controlling behavior
✔ Isolating state

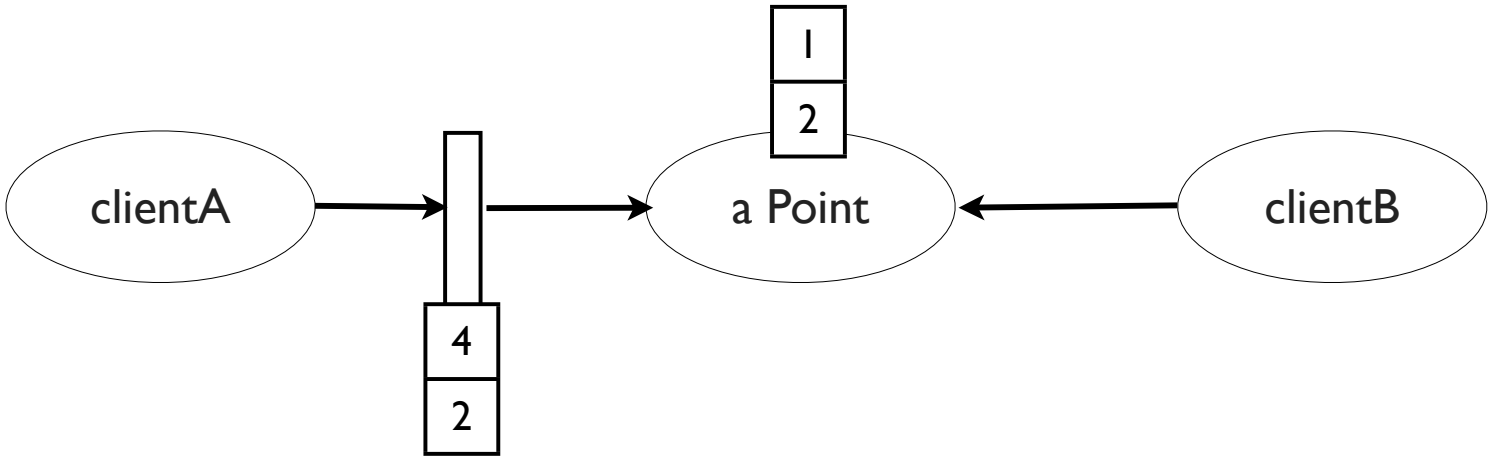
Core

- ✔ Reifying references
  - ✔ Controlling behavior
  - ✔ Isolating state

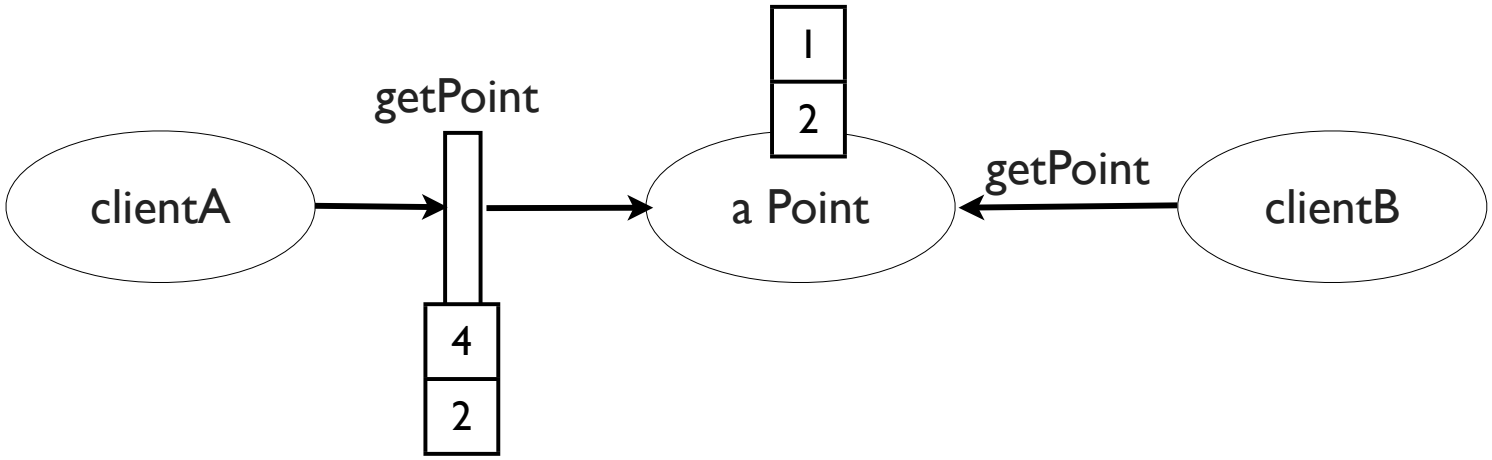
Consequences

- ✘ Identity is broken
- ✘ Graph control is absent
- ✘ Handle creation

Identity problem



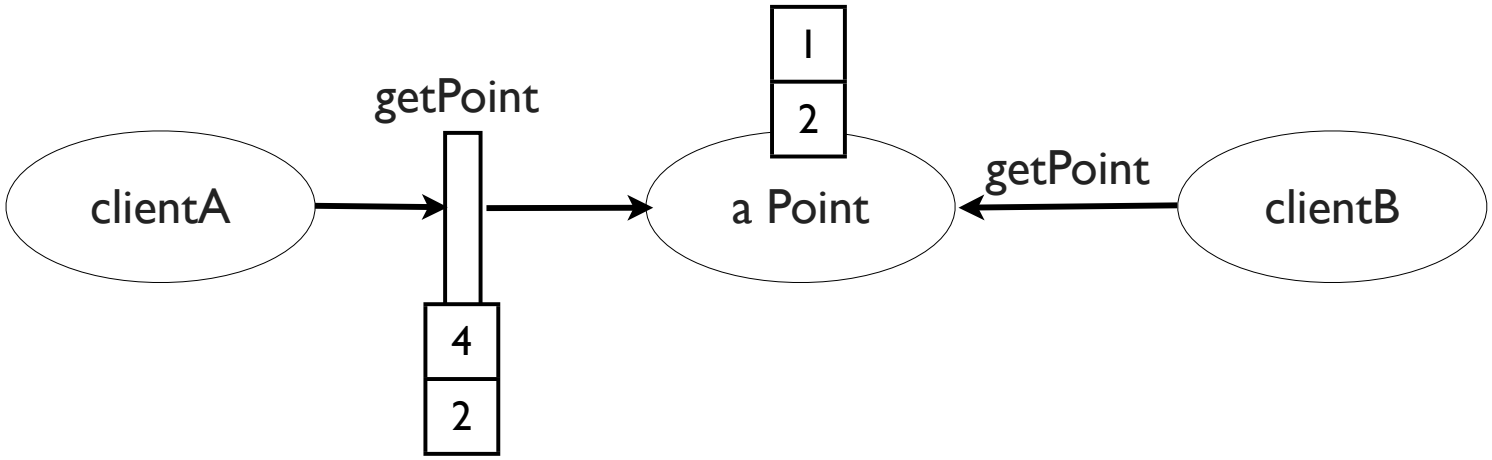
Identity problem



clientA getPoint **identity** clientB getPoint

True

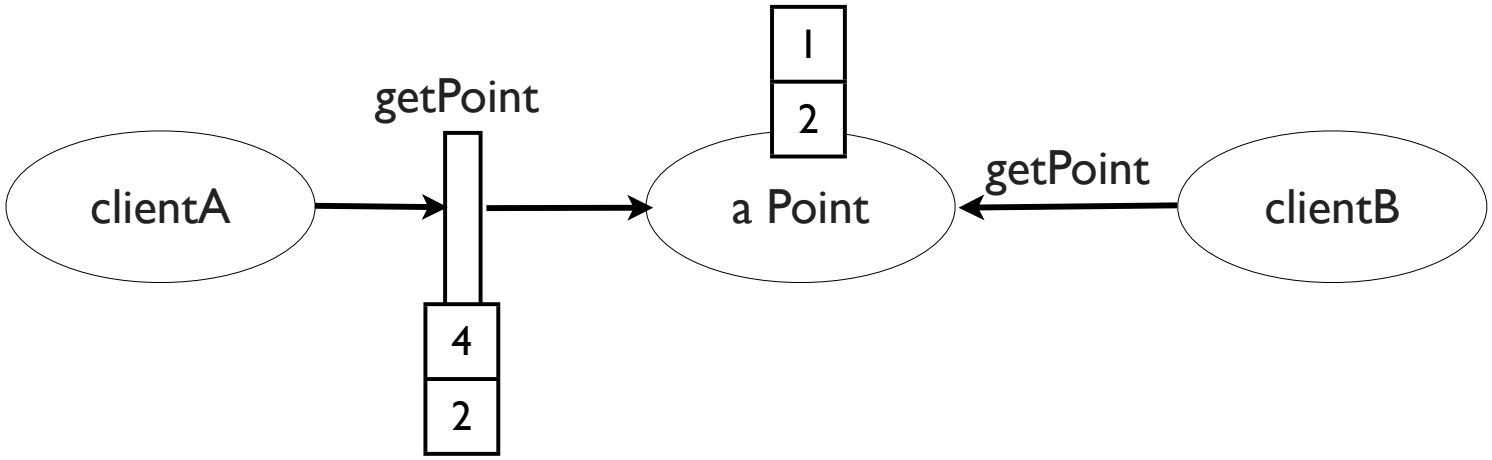
Identity problem



clientA getPoint | equality | clientB getPoint

False

Identity problem



clientA getPoint | equality | clientB getPoint

False

Identity does not imply equality anymore

Rewrite identity model

Equality

clientA getPoint | equality | clientB getPoint

False

Similarity (object identity)

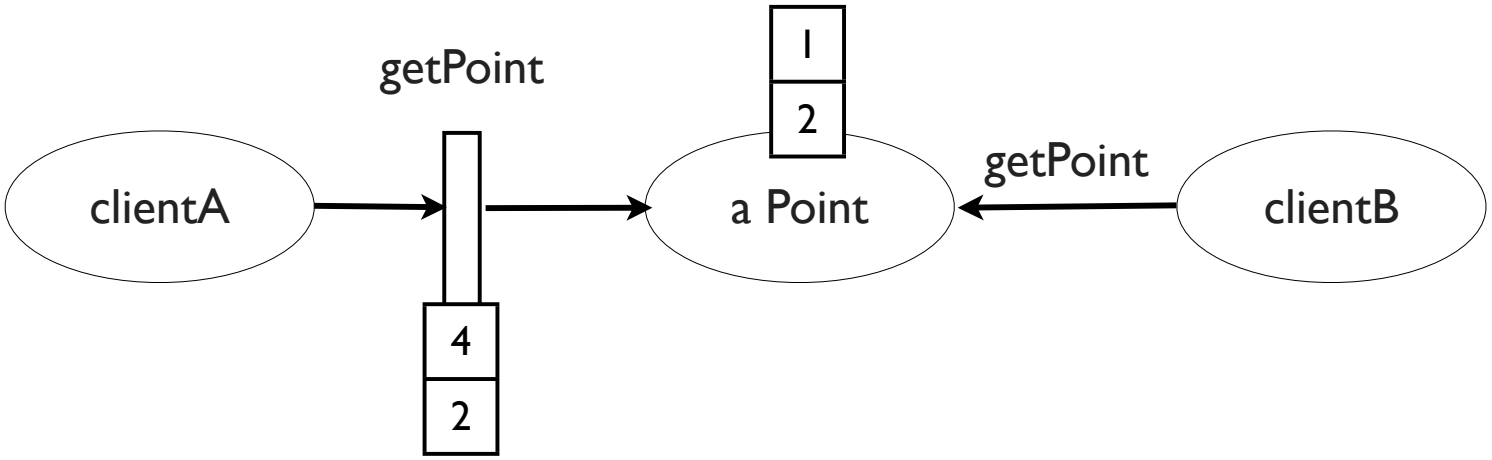
clientA getPoint | similarity | clientB getPoint

True

Identity (reference identity)

clientA getPoint | identity | clientB getPoint

False



Rewrite identity model

Equality

clientA getPoint | equality | clientB getPoint

False

Similarity (object identity)

clientA getPoint | similarity | clientB getPoint

True

Identity (reference identity)

clientA getPoint | identity | clientB getPoint

False

We can distinguish objects and references
Identity still means equality

The identity invariant is maintained

Core

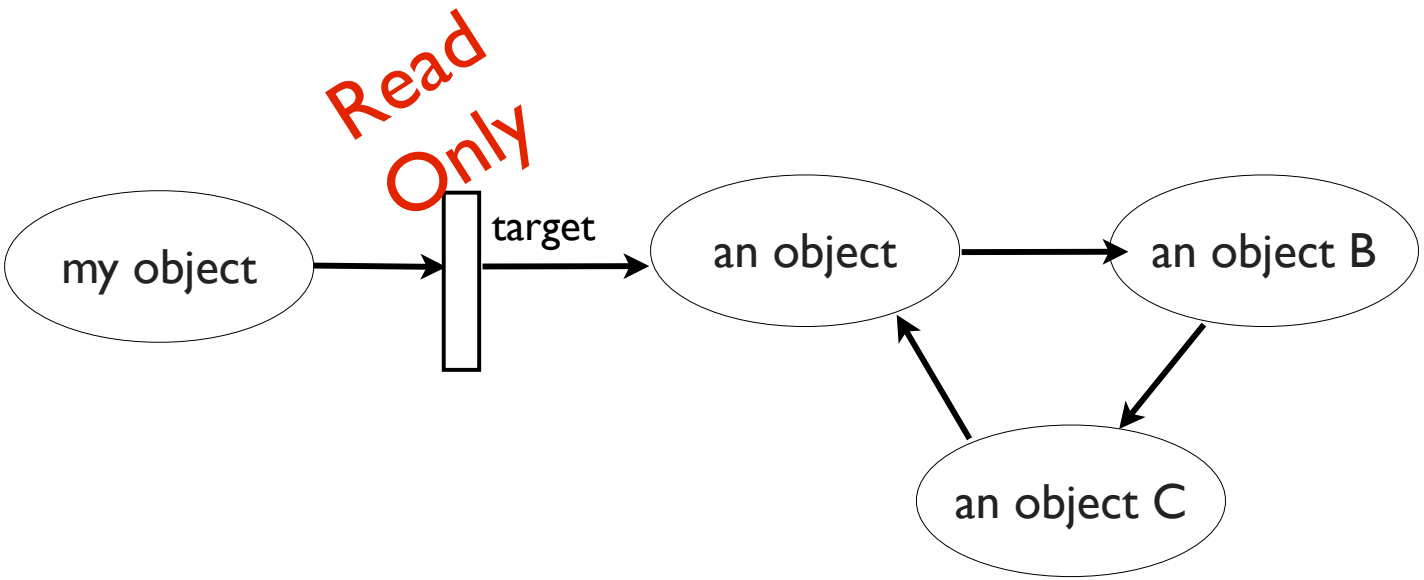
- ✔ Reifying references
  - ✔ Controlling behavior
  - ✔ Isolating state

Consequences

- ✔ Identity is preserved
- ✘ Graph control is absent
- ✘ Handle creation

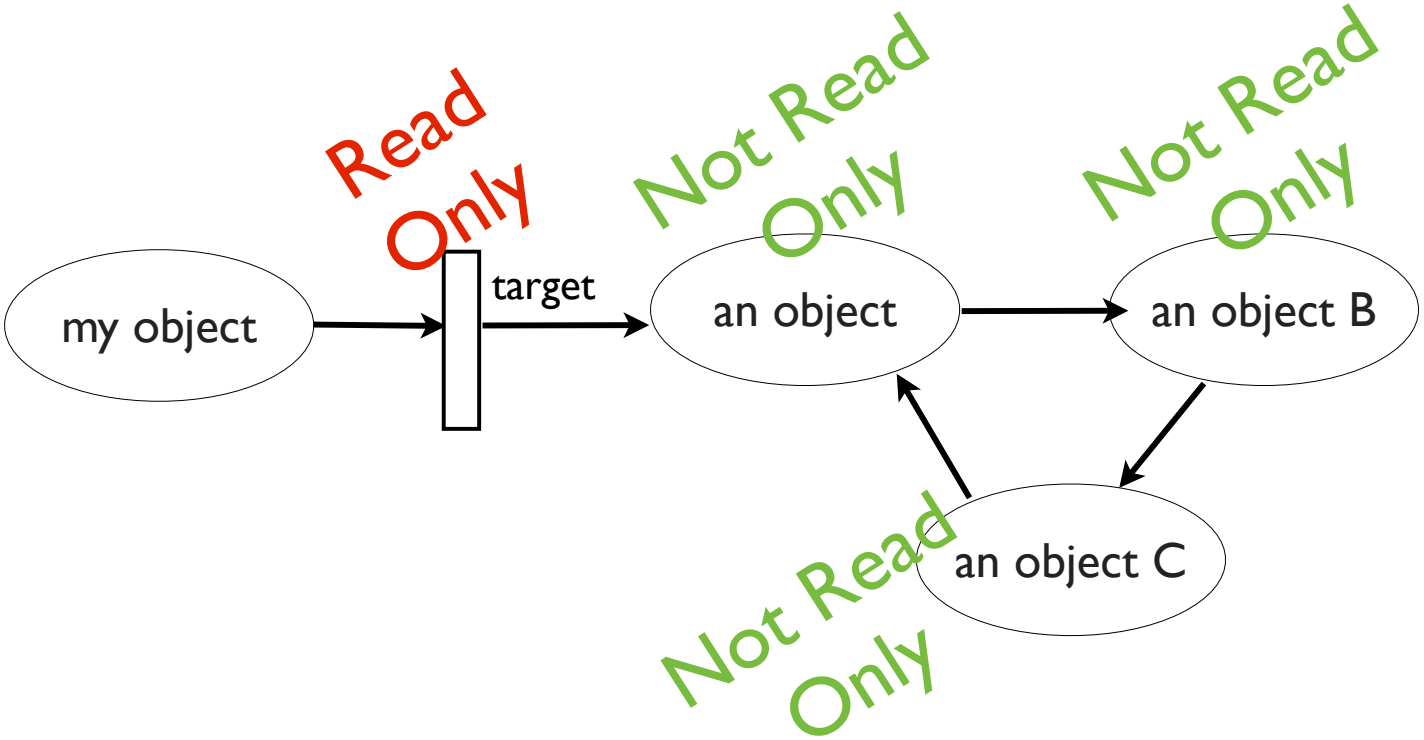
Controlling graph

Controlling one atomic reference is not enough



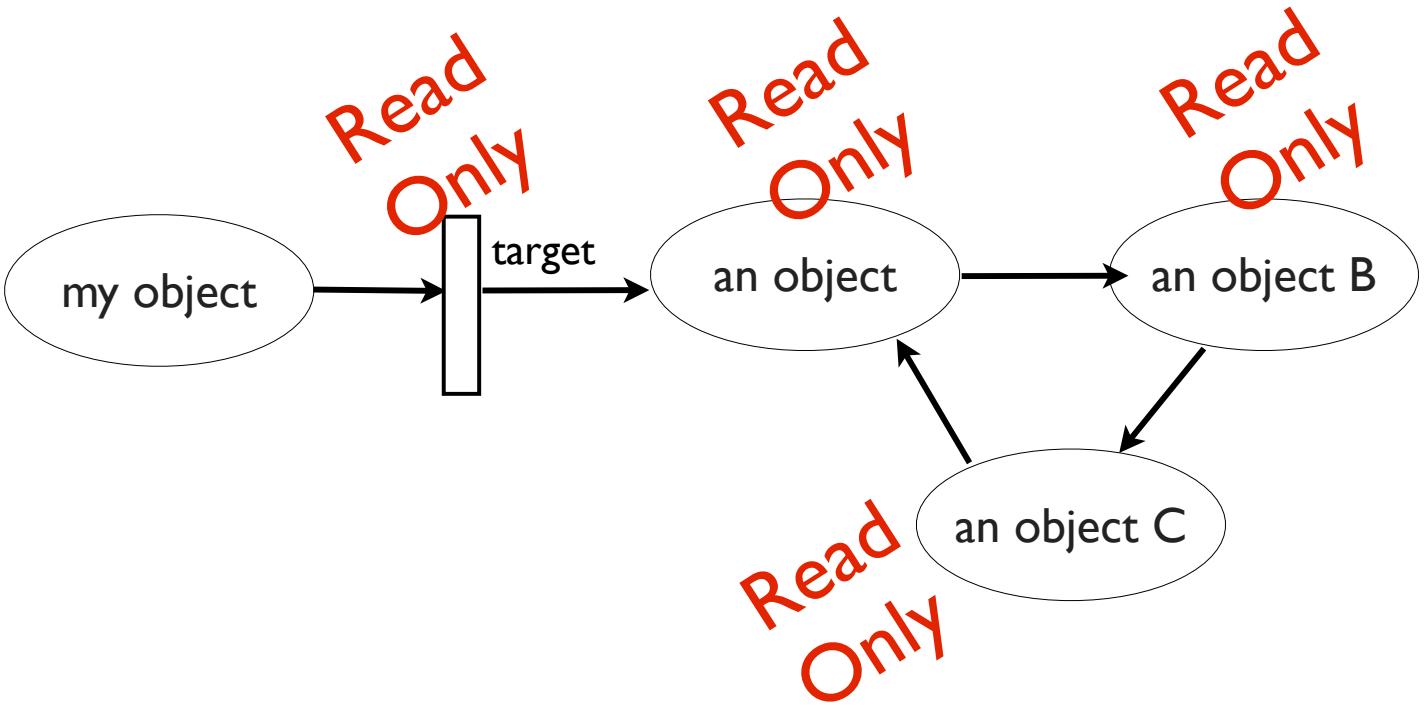
Controlling graph

Controlling one atomic reference is not enough

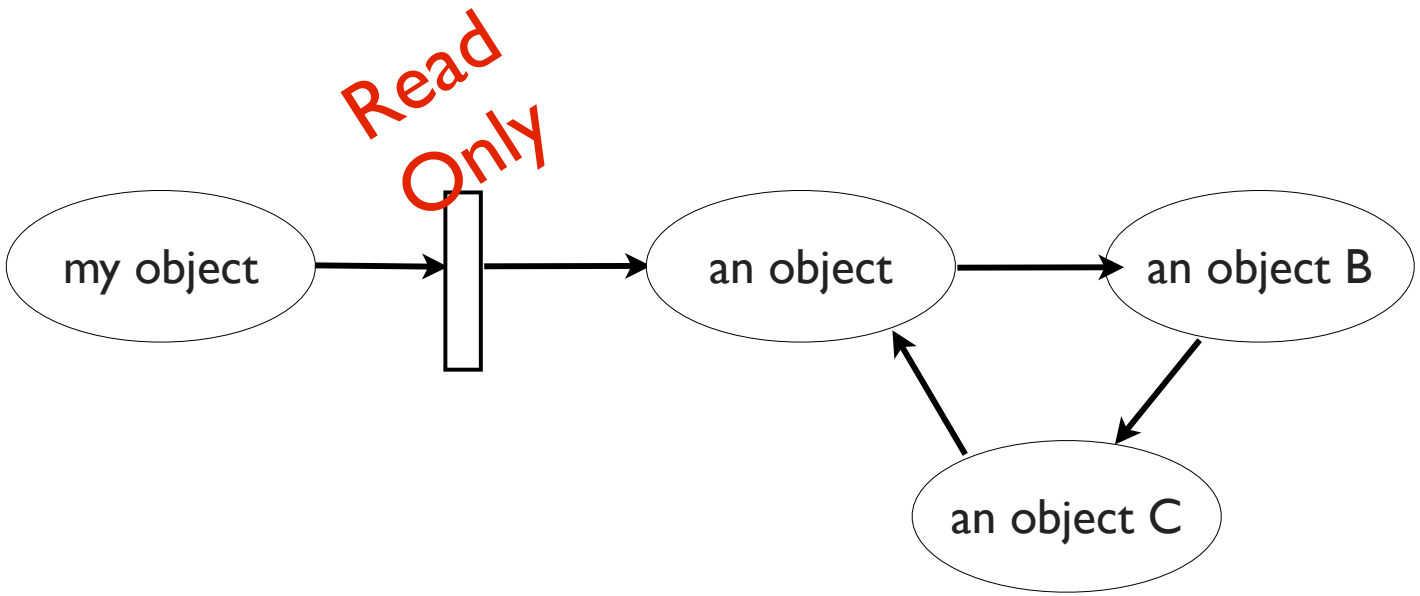


Controlling graph

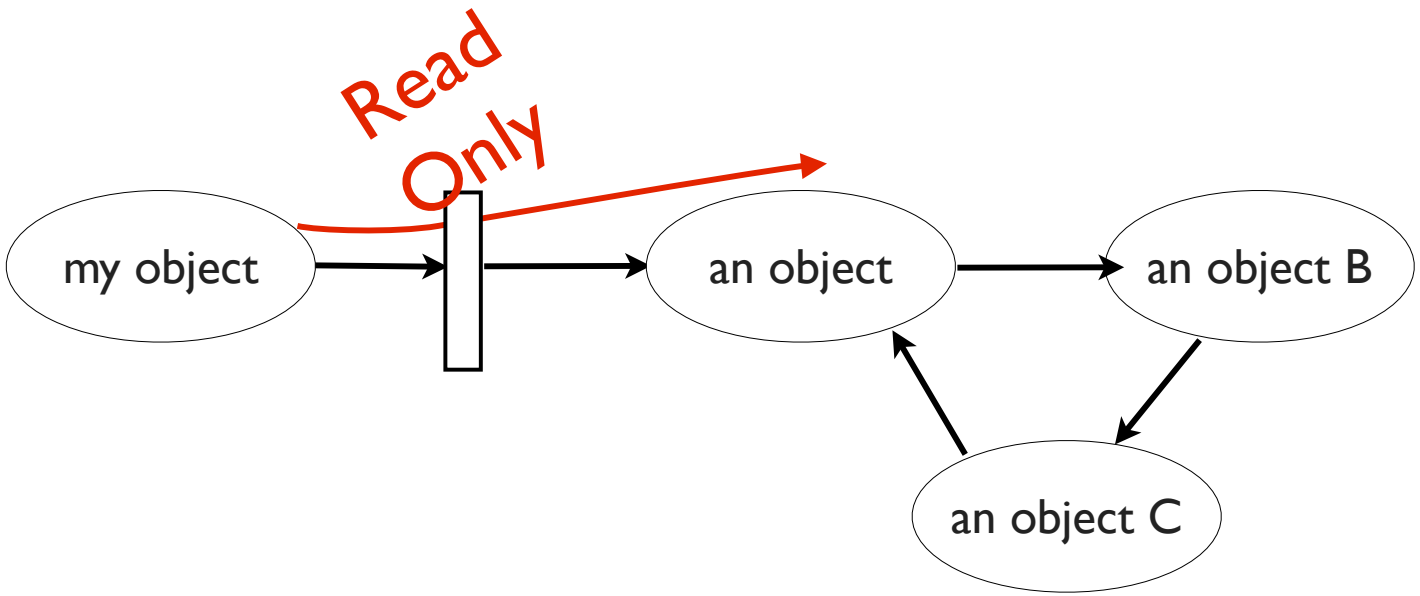
Controlling one atomic reference is not enough



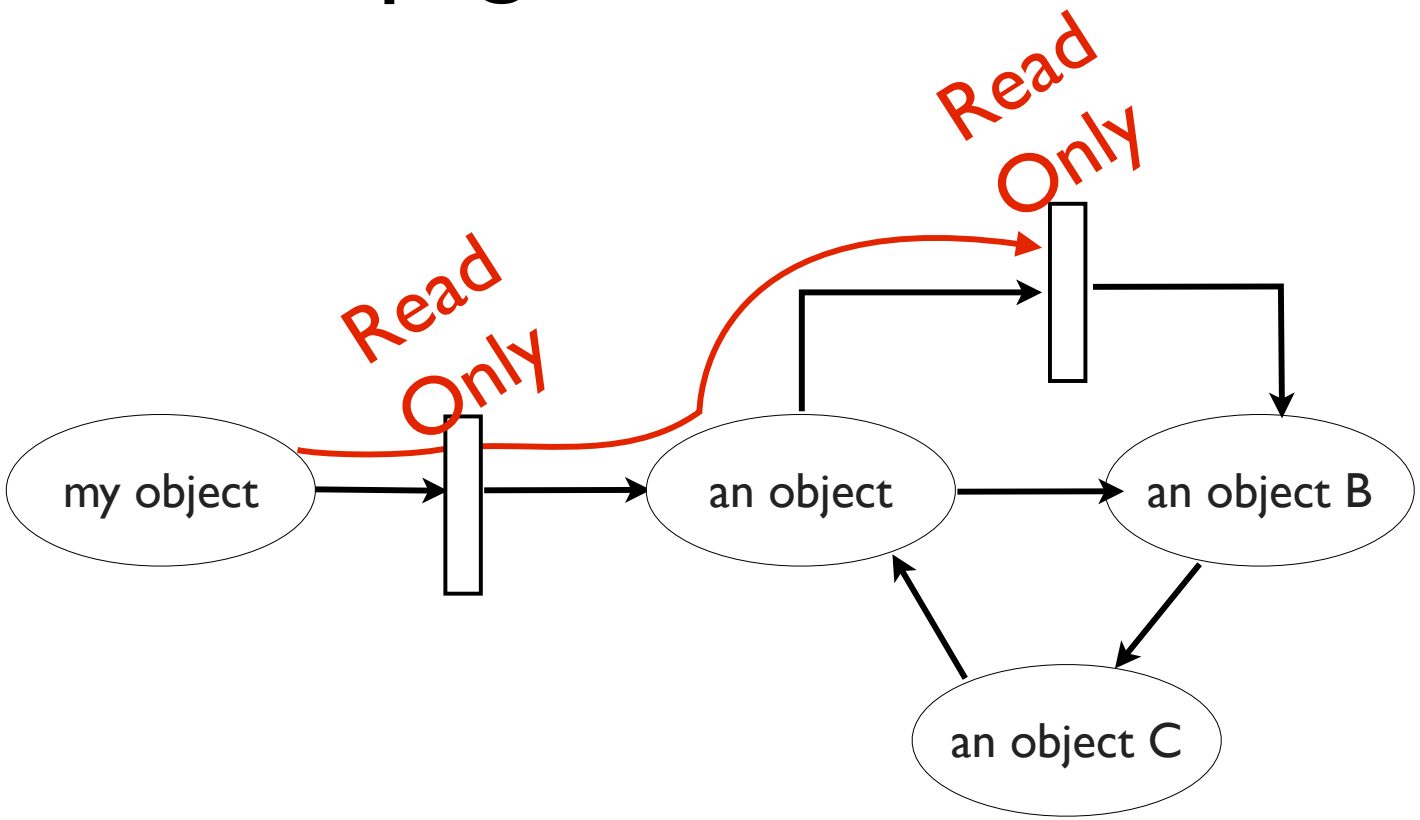
Propagation Semantics



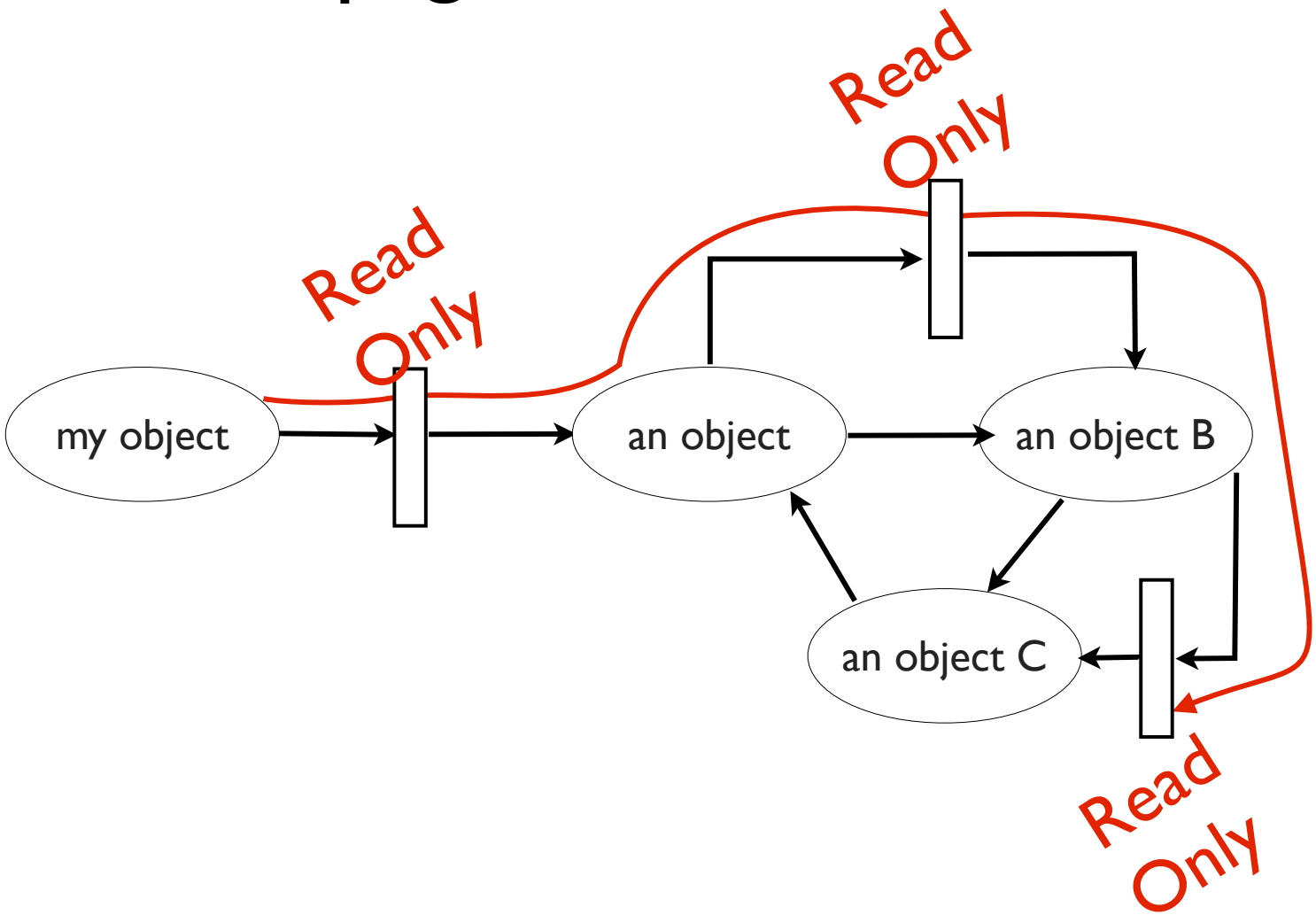
Propagation Semantics



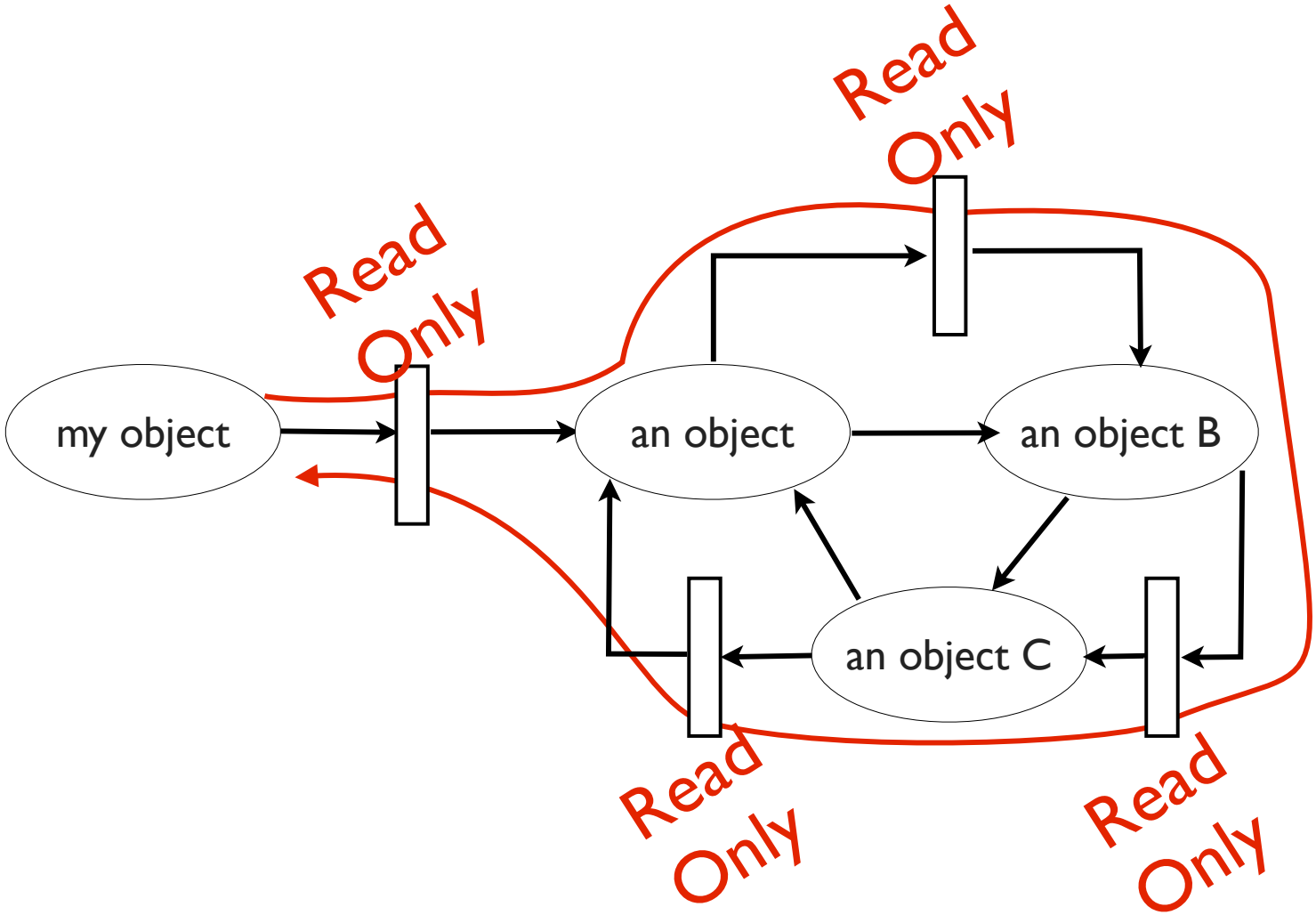
Propagation Semantics



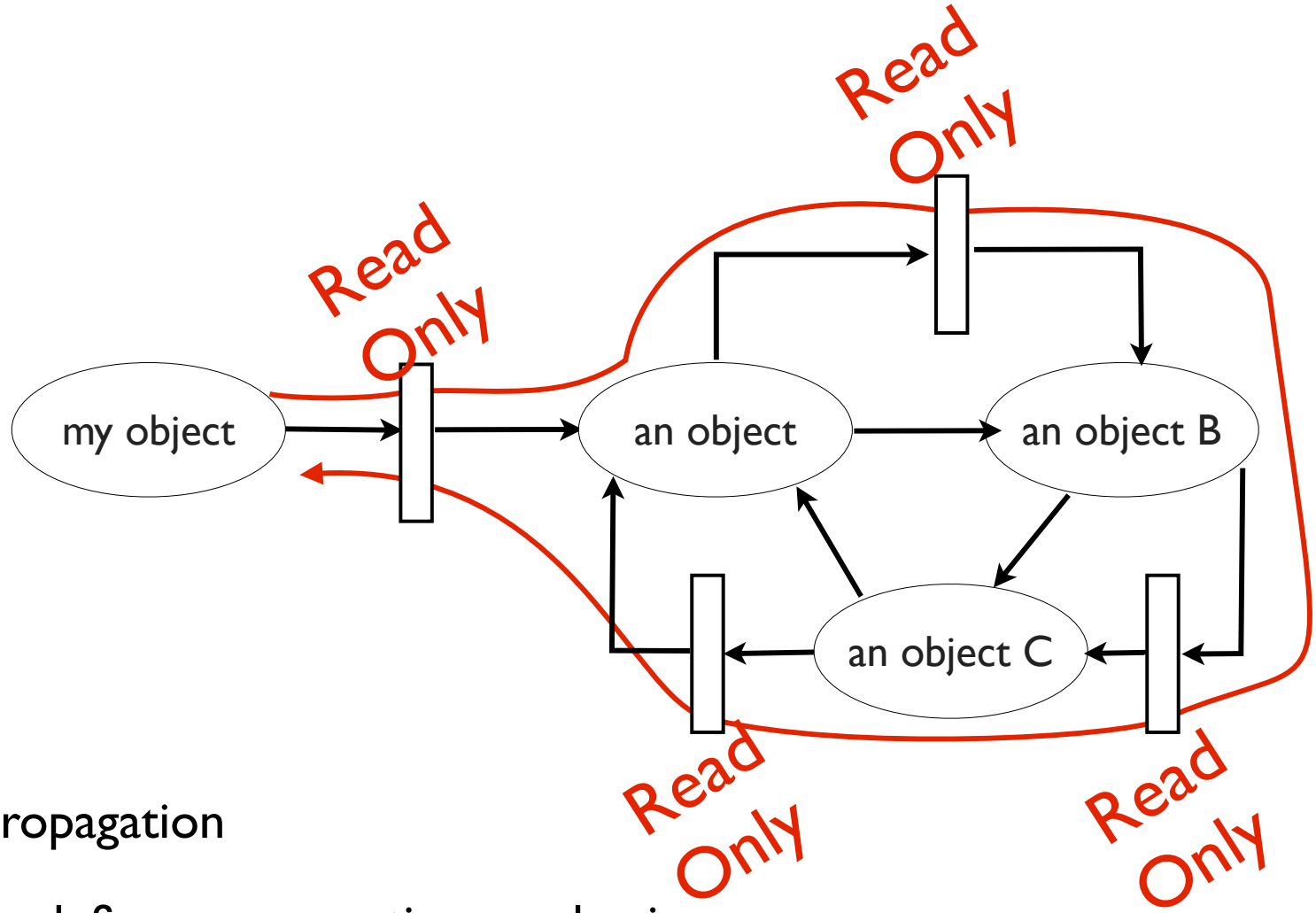
Propagation Semantics



Propagation Semantics



Propagation Semantics



Handle propagation

- Handles define a propagation mechanism
- Propagation automatically performed by the Virtual Machine

Core

- ✔ Reifying references
  - ✔ Controlling behavior
  - ✔ Isolating state

Consequences

- ✔ Identity is preserved
- ✔ Propagated semantics
- ✘ Handle creation

Handle: life cycle

Phases	1 Initialization	2 Execution
Representation	Object	Handle
Possibles actions	Configuration	Apply the semantics defined

Handle: life cycle

Phases	1 Initialization	2 Execution
Representation	Object	Handle
Possibles actions	Configuration	Apply the semantics defined

Handle: life cycle

Phases	1 Initialization	2 Execution
Representation	Object	Handle
Possibles actions	Configuration	Apply the semantics defined

Core

- ✔ Reifying references
  - ✔ Controlling behavior
  - ✔ Isolating state

Consequences

- ✔ Identity is preserved
- ✔ Propagated semantics
- ✔ Handle creation

Core

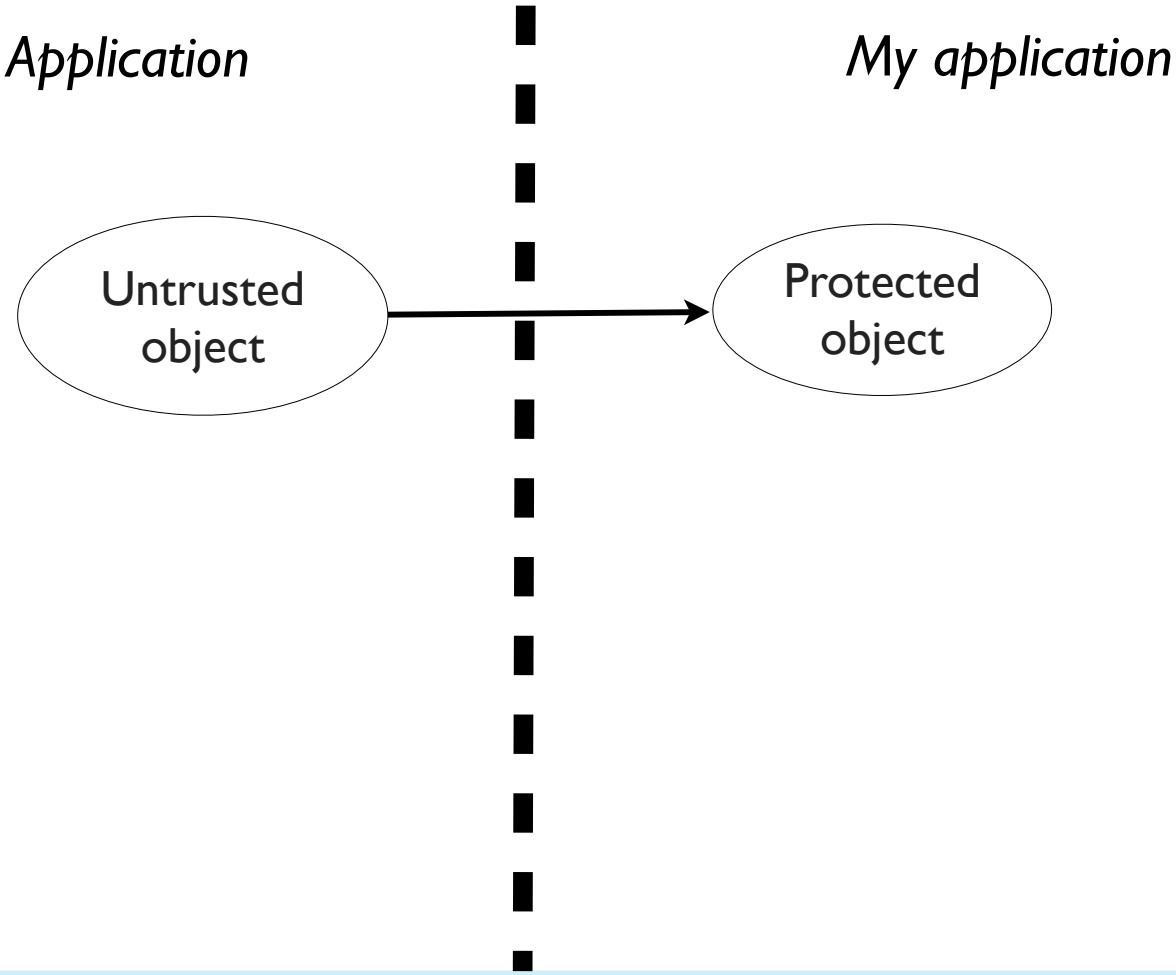
- ✔ Reifying references
  - ✔ Controlling behavior
  - ✔ Isolating state

Consequences

- ✔ Identity is preserved
- ✔ Propagated semantics
- ✔ Handle creation
  - ✘ Handle control ?

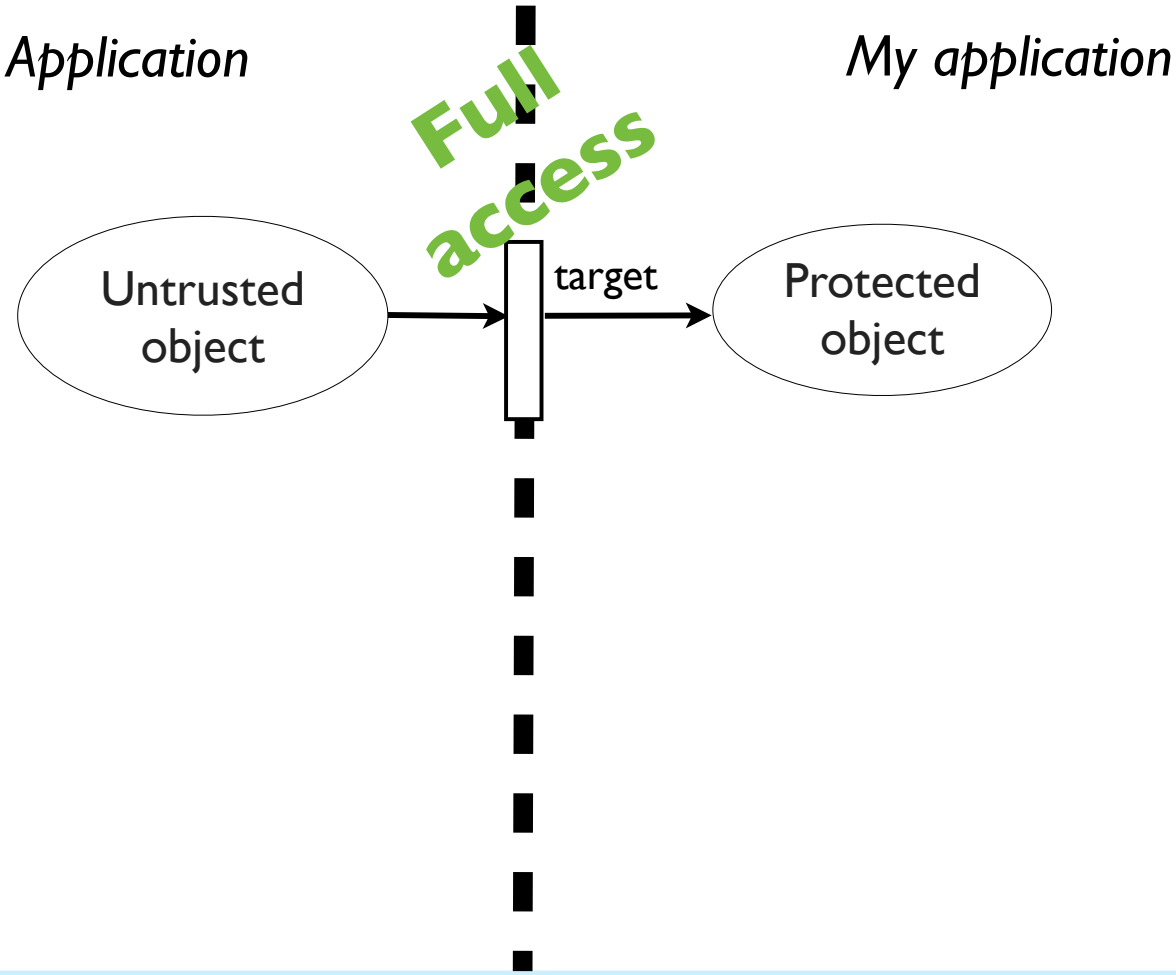
Handle: control ?

If we want to change the semantics:



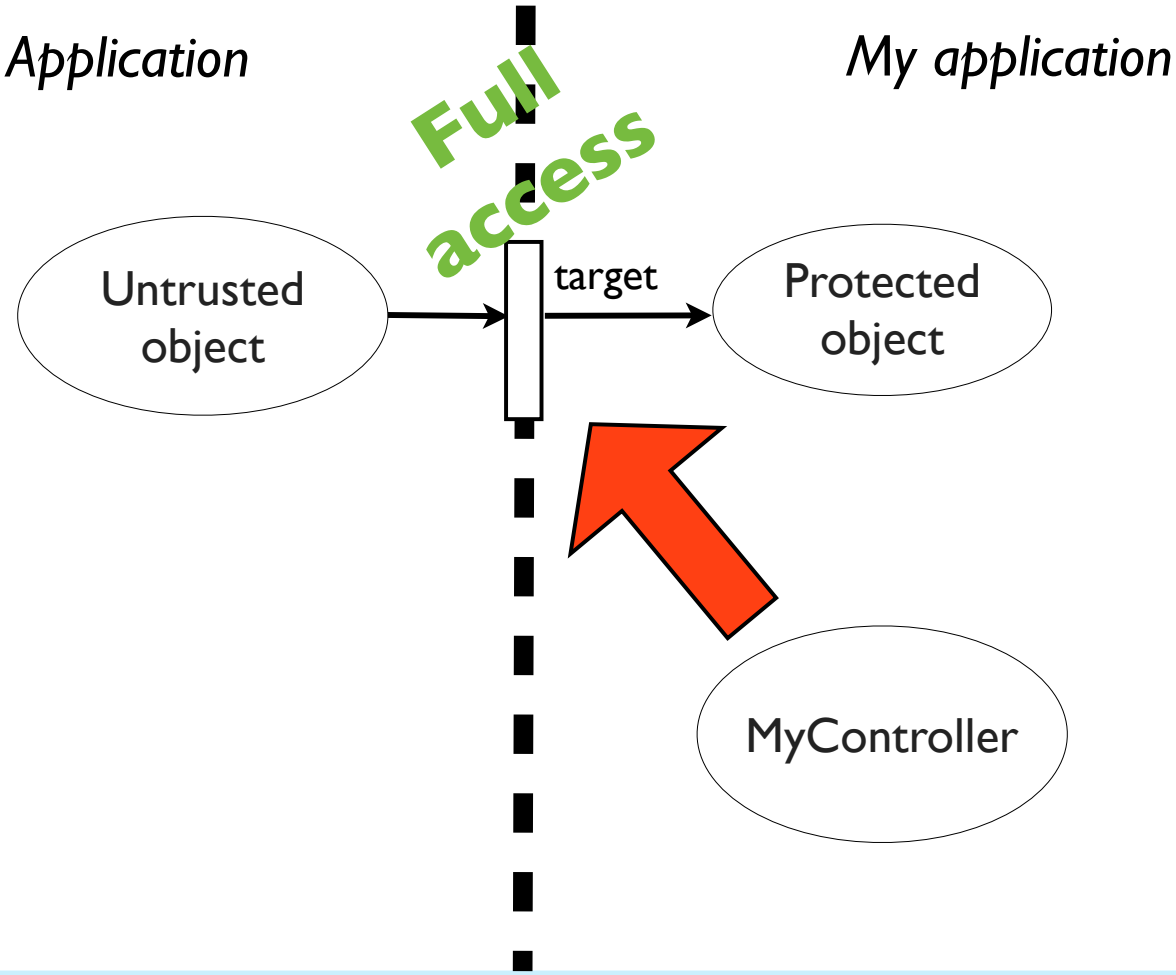
Handle: control ?

If we want to change the semantics:



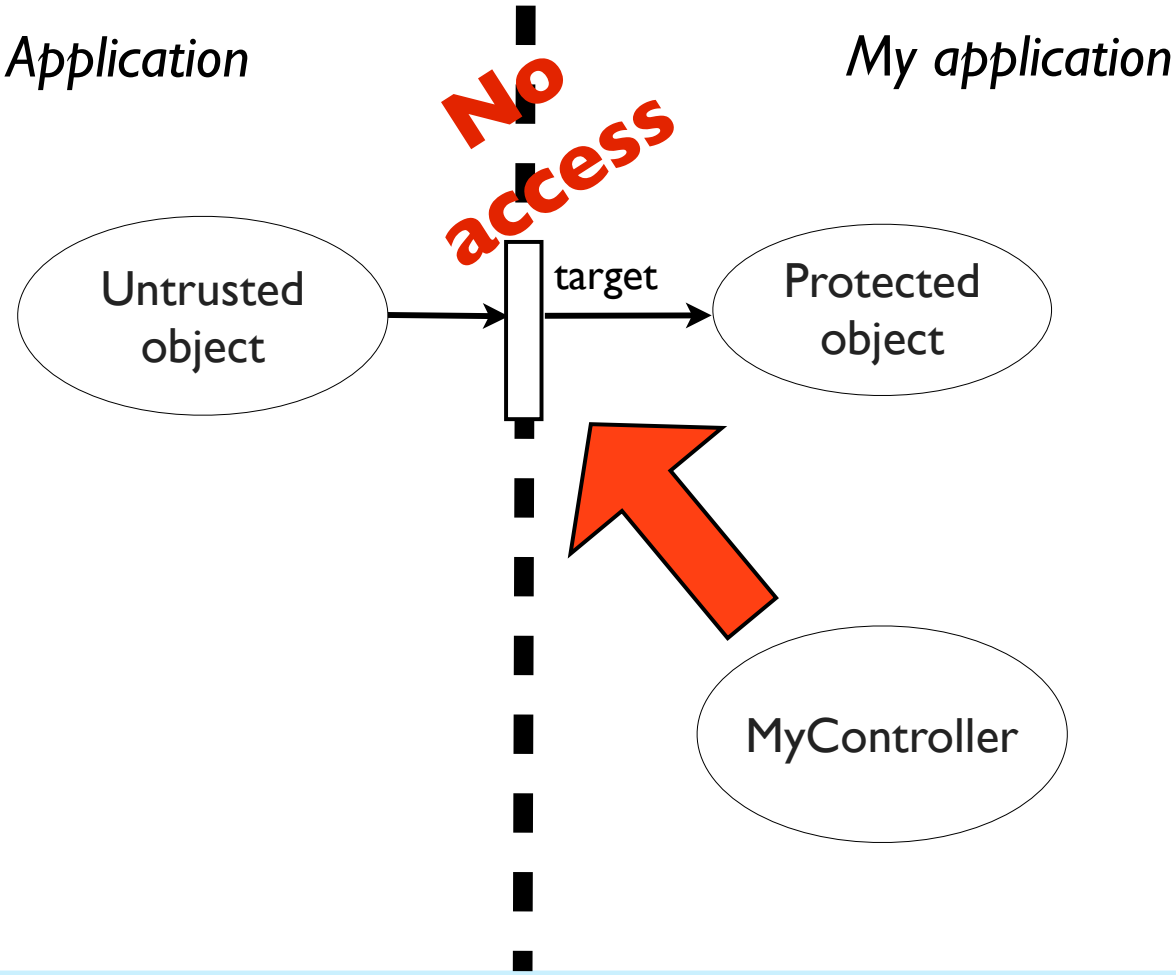
Handle: control ?

If we want to change the semantics:



Handle: control ?

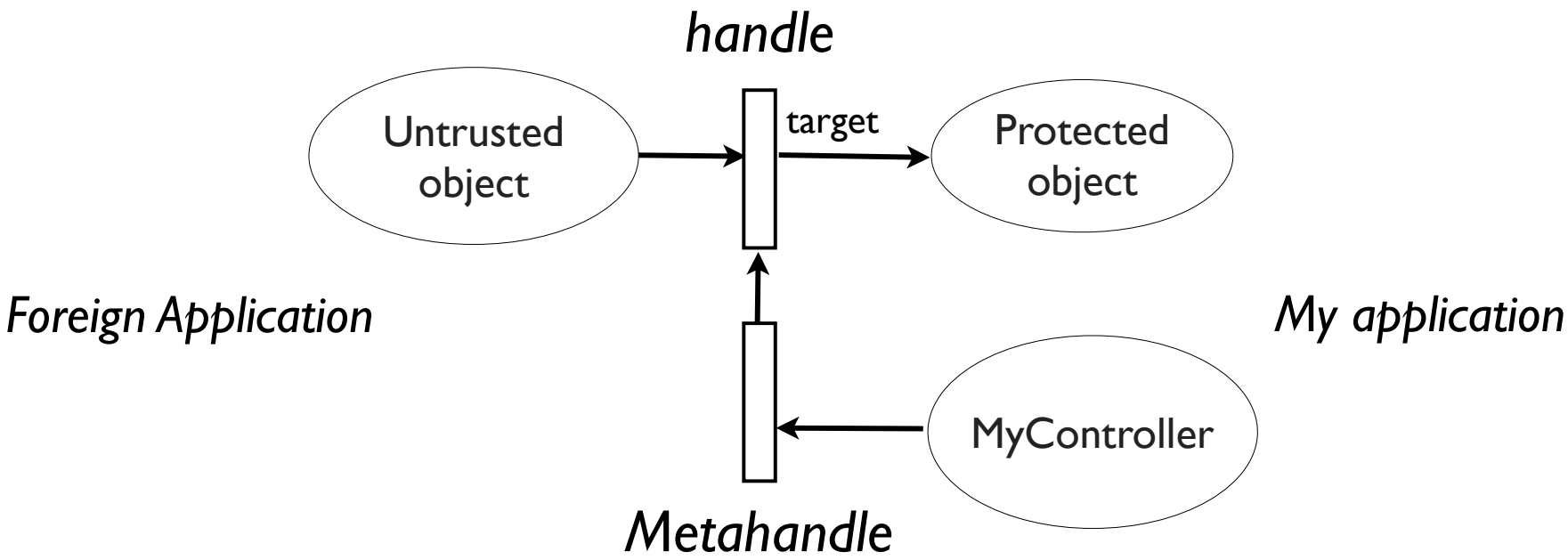
If we want to change the semantics:



Handle: Metahandle

Metahandle

- Handle on Handle
- Metahandle can be created during initialization phase
- Enable message passing to activated handle



Handle: Reflection

Reflection:

Possibility to examine, change and execute behavior of objects at runtime

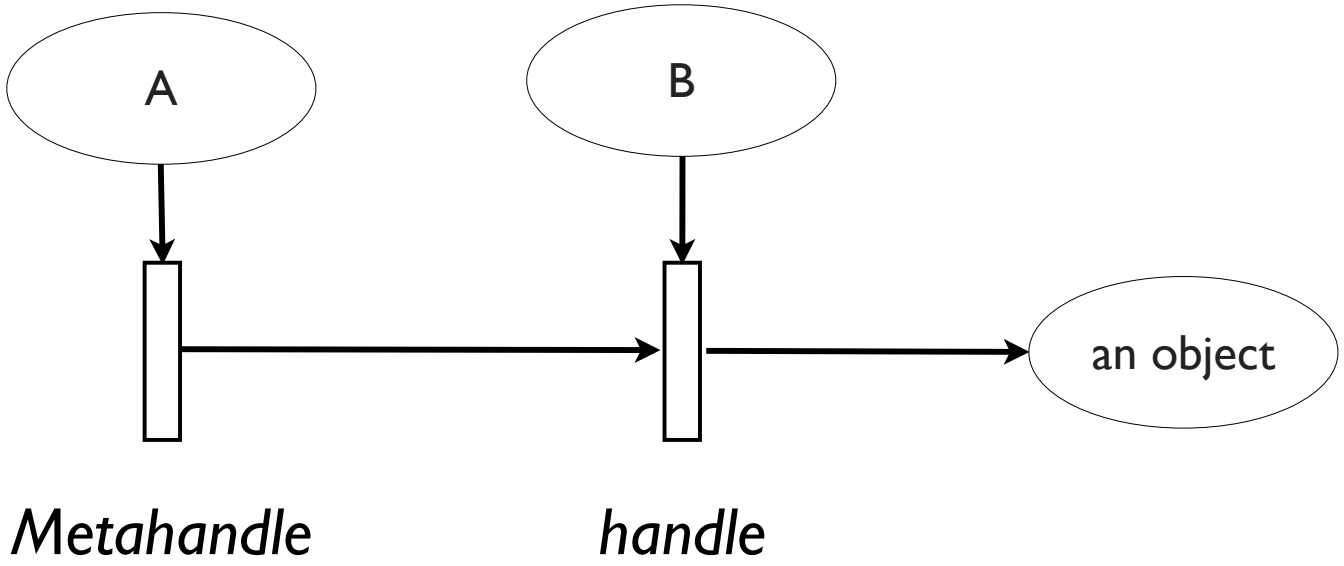
- Using reflection for meta-programming
- Using reflection for debugging

Handle: Reflection

Reflection:

Possibility to examine, change and execute behavior of objects at runtime

- Using reflection for meta-programming
- Using reflection for debugging



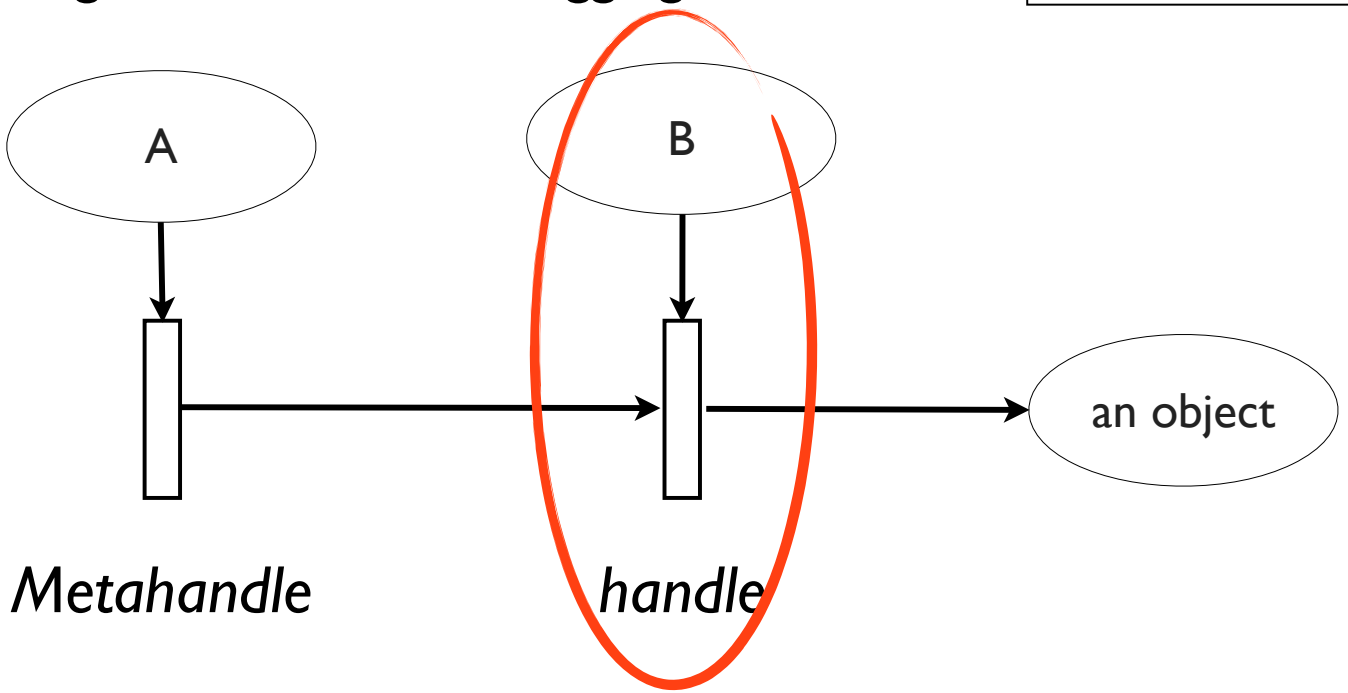
Handle: Reflection

Reflection:

Possibility to examine, change and execute behavior of objects at runtime

- Using reflection for meta-programming
- Using reflection for debugging

Apply reflective API to target object



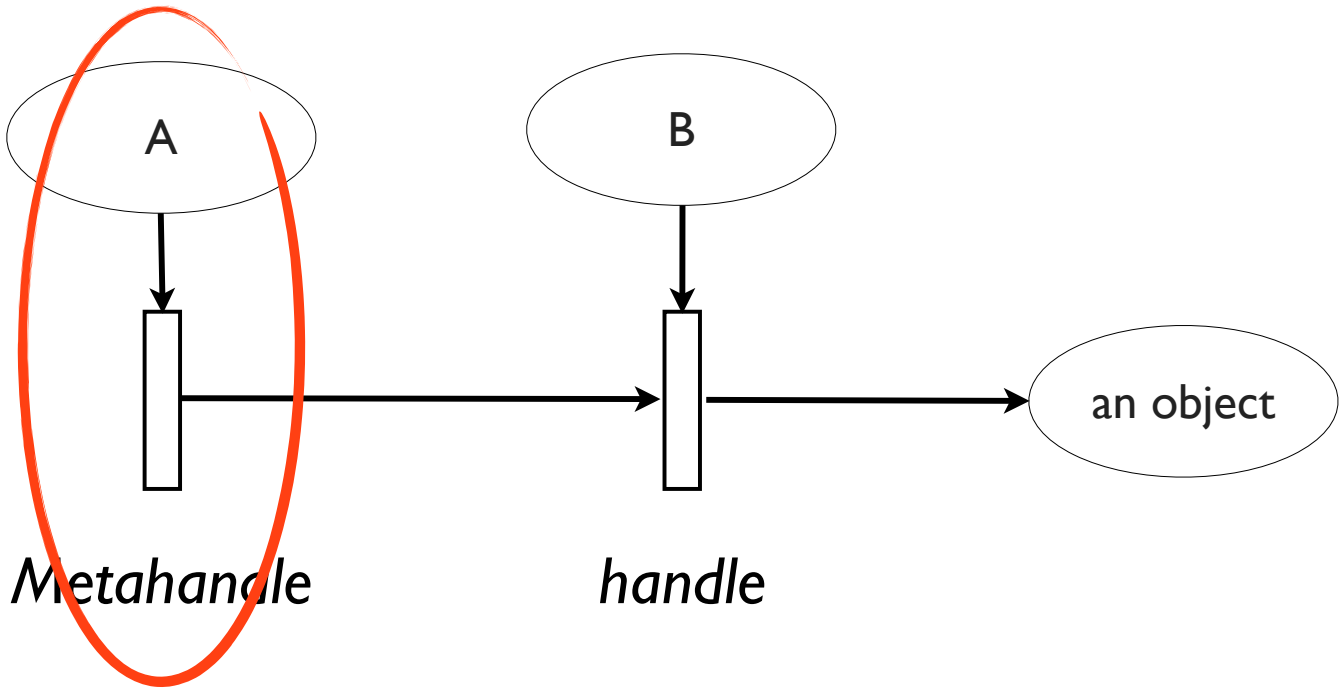
Handle: Reflection

Reflection:

Possibility to examine, change and execute behavior of objects at runtime

- Using reflection for meta-programming
- Using reflection for debugging

Apply reflective API to target object



The basics

- ✔ Reifying references
- ✔ Controlling behavior
- ✔ Isolating state

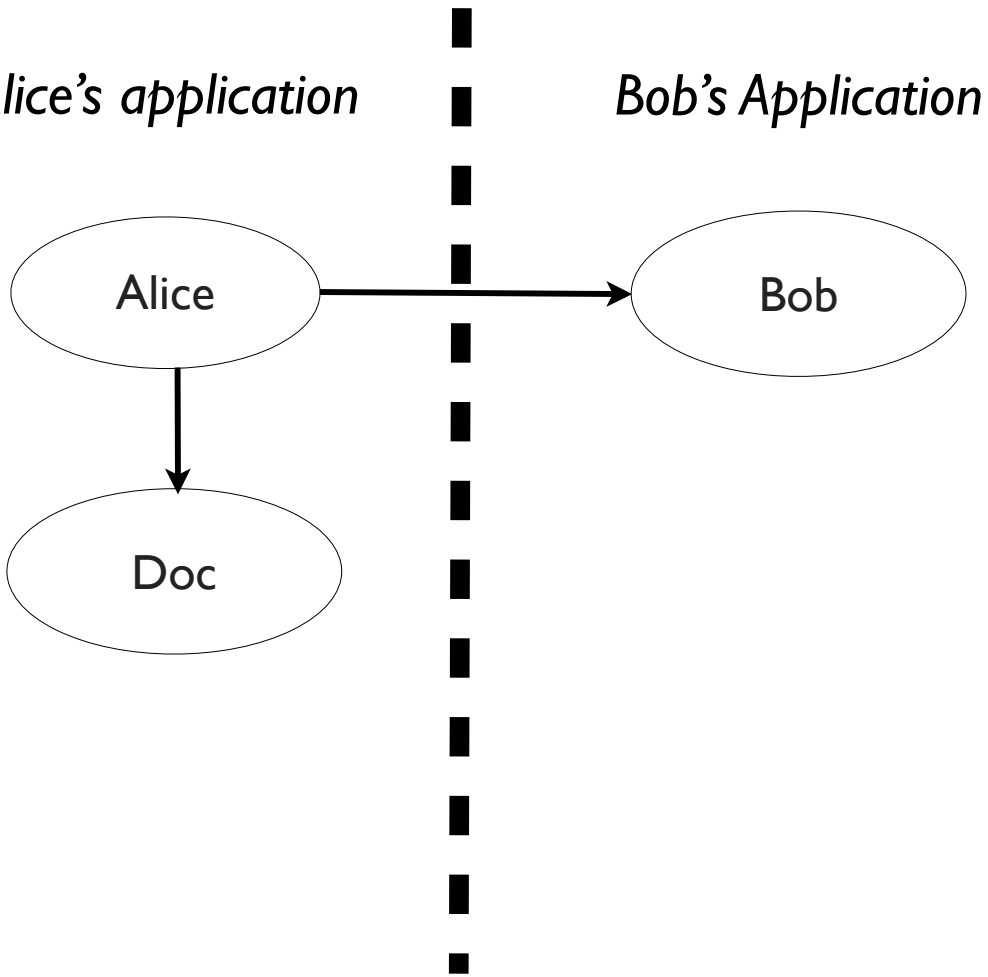
Consequences

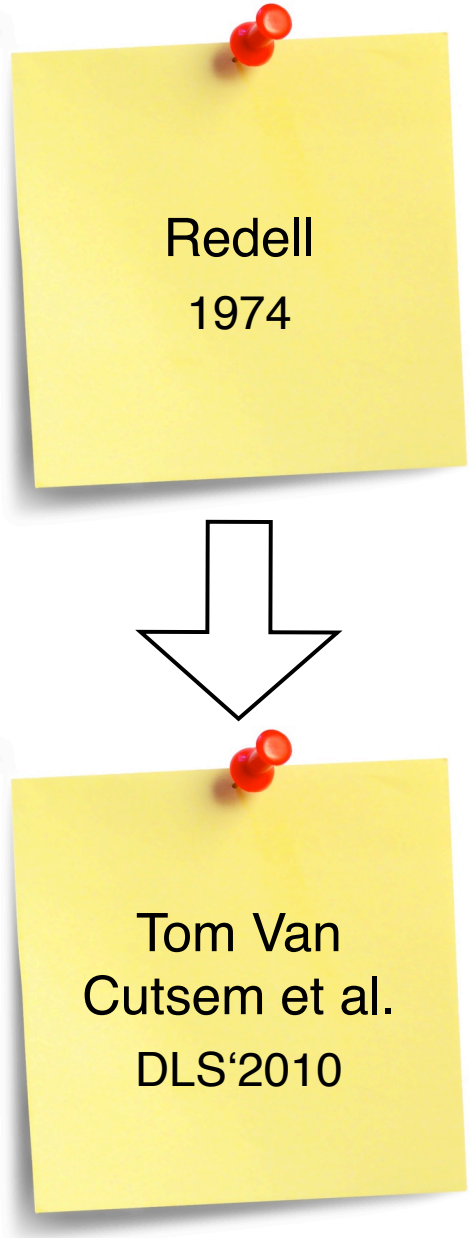
- ✔ Identity is preserved
- ✔ Propagated semantics
- ✔ Real time Control
- ✔ Reflection

Outline

Handles at work

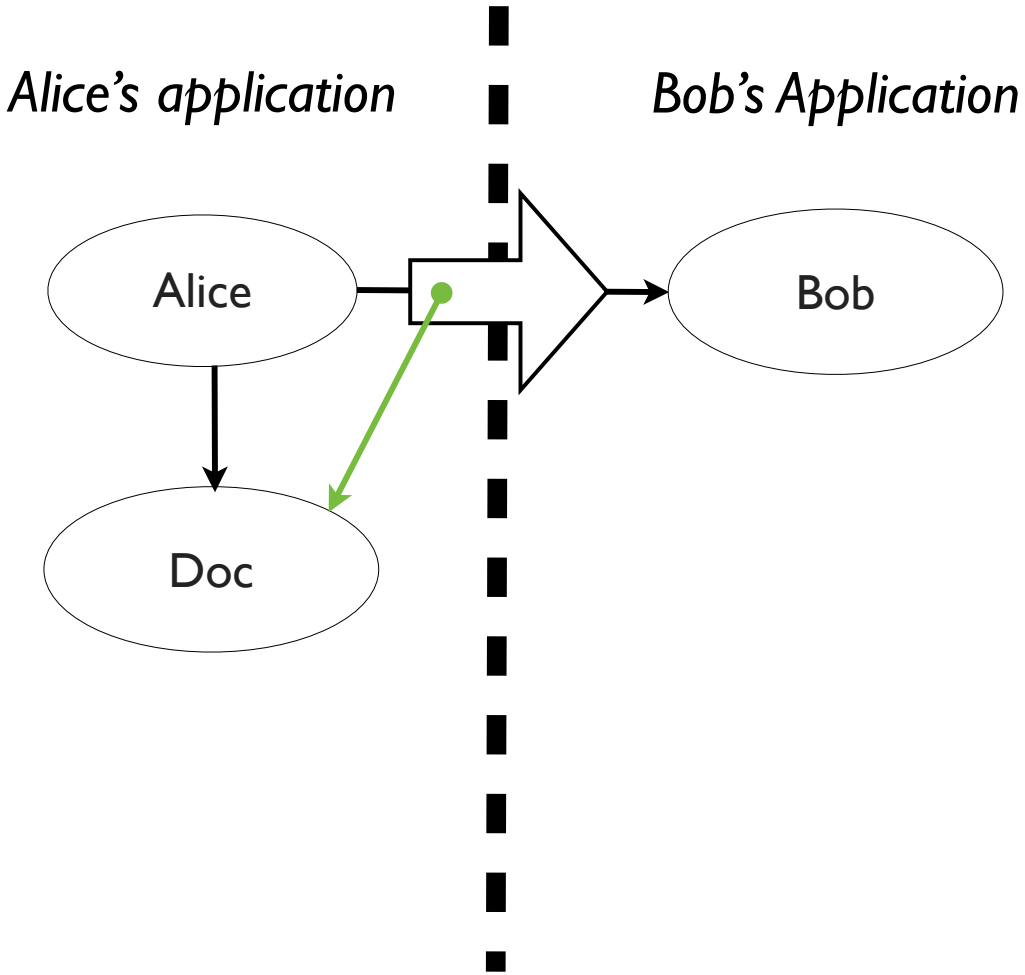
Revocable references

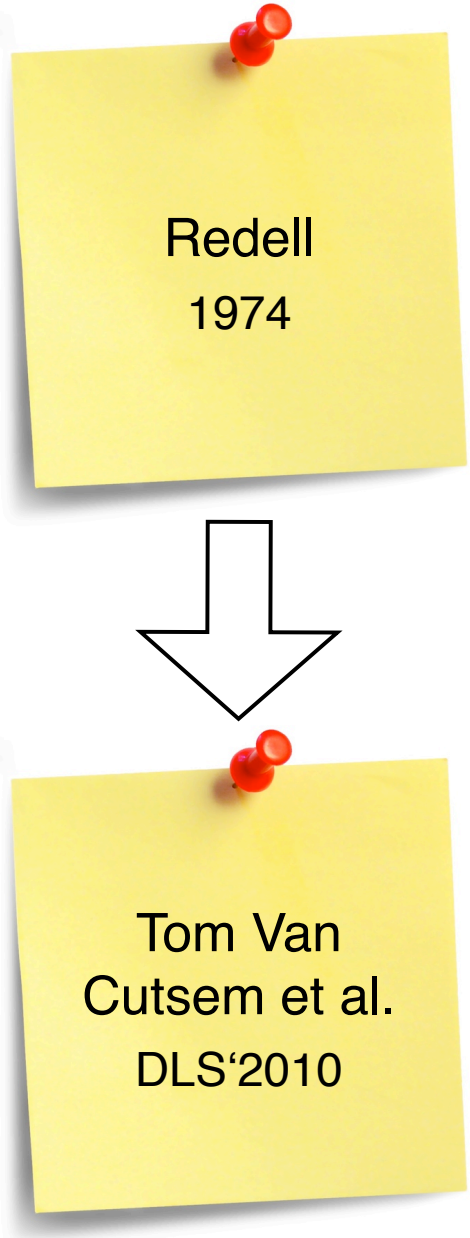




Handles at work

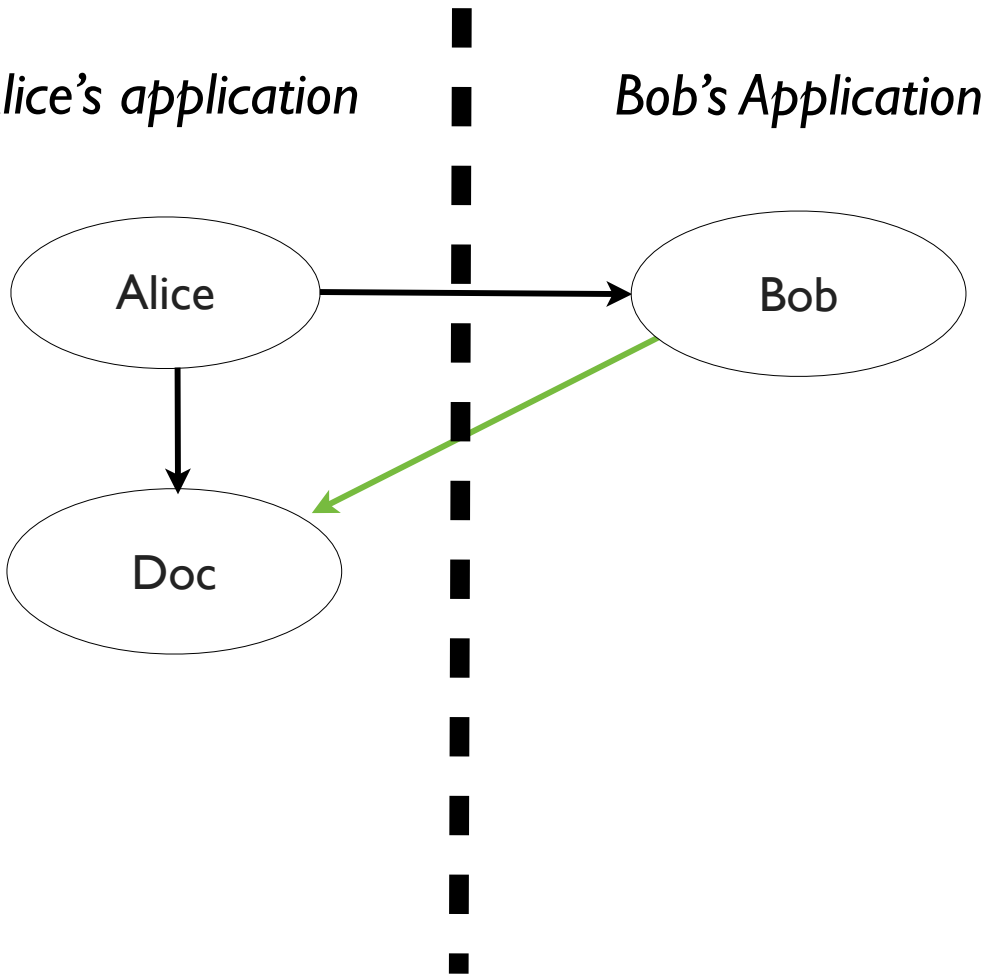
Revocable references

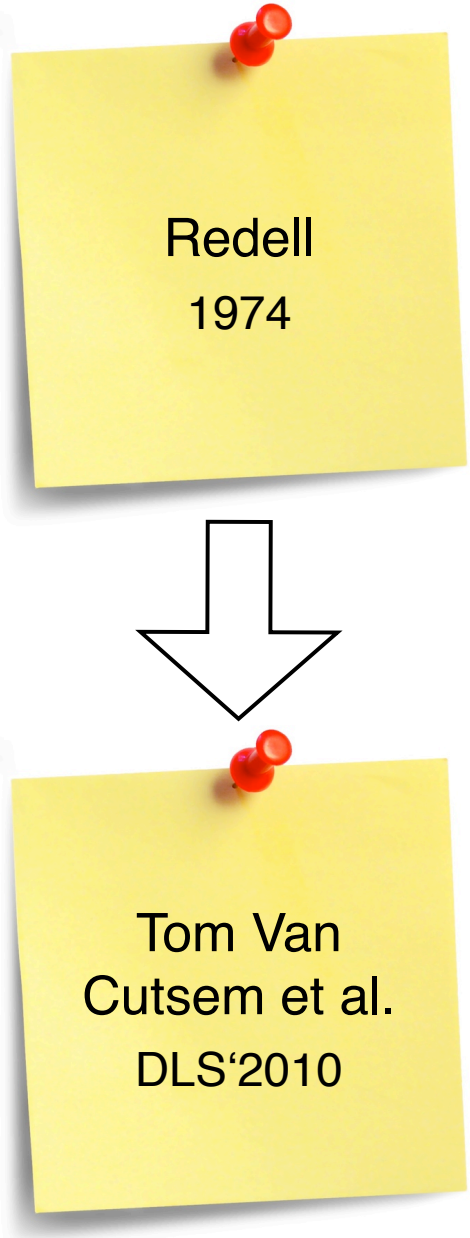




Handles at work

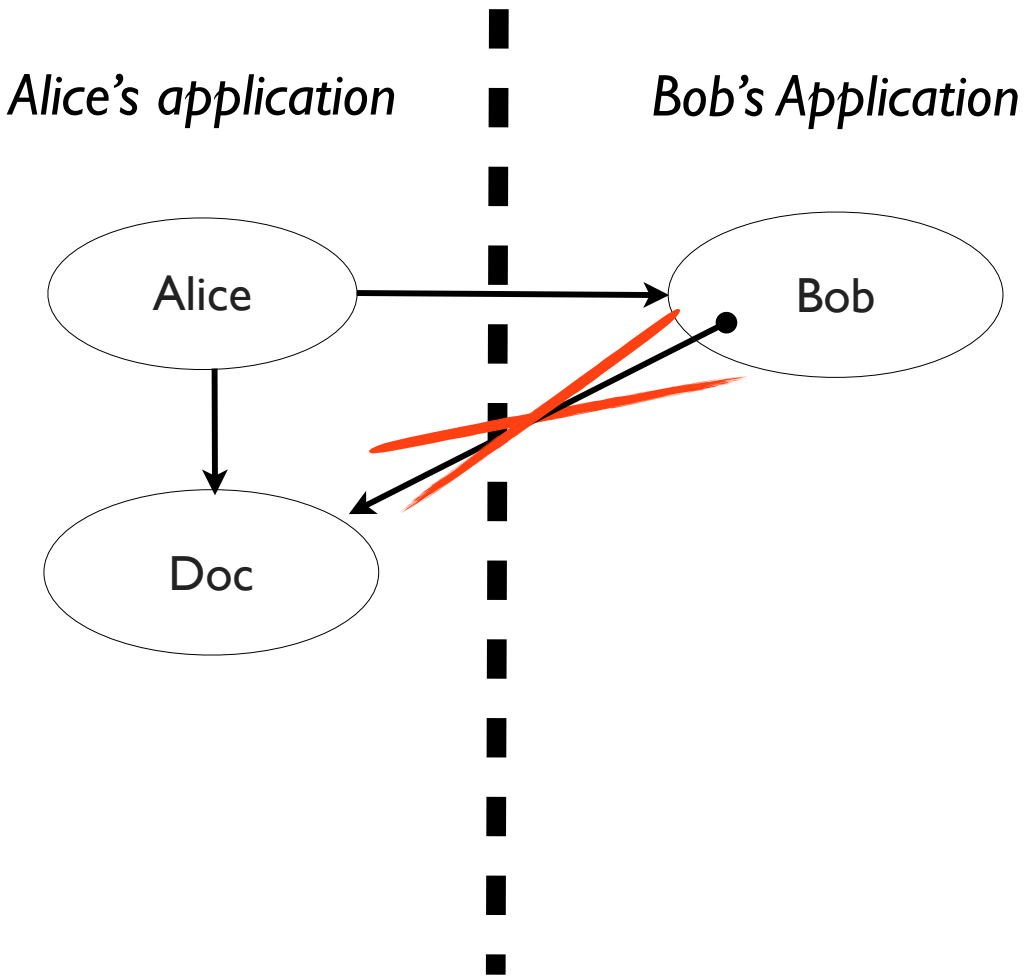
Revocable references





Handles at work

Revocable references

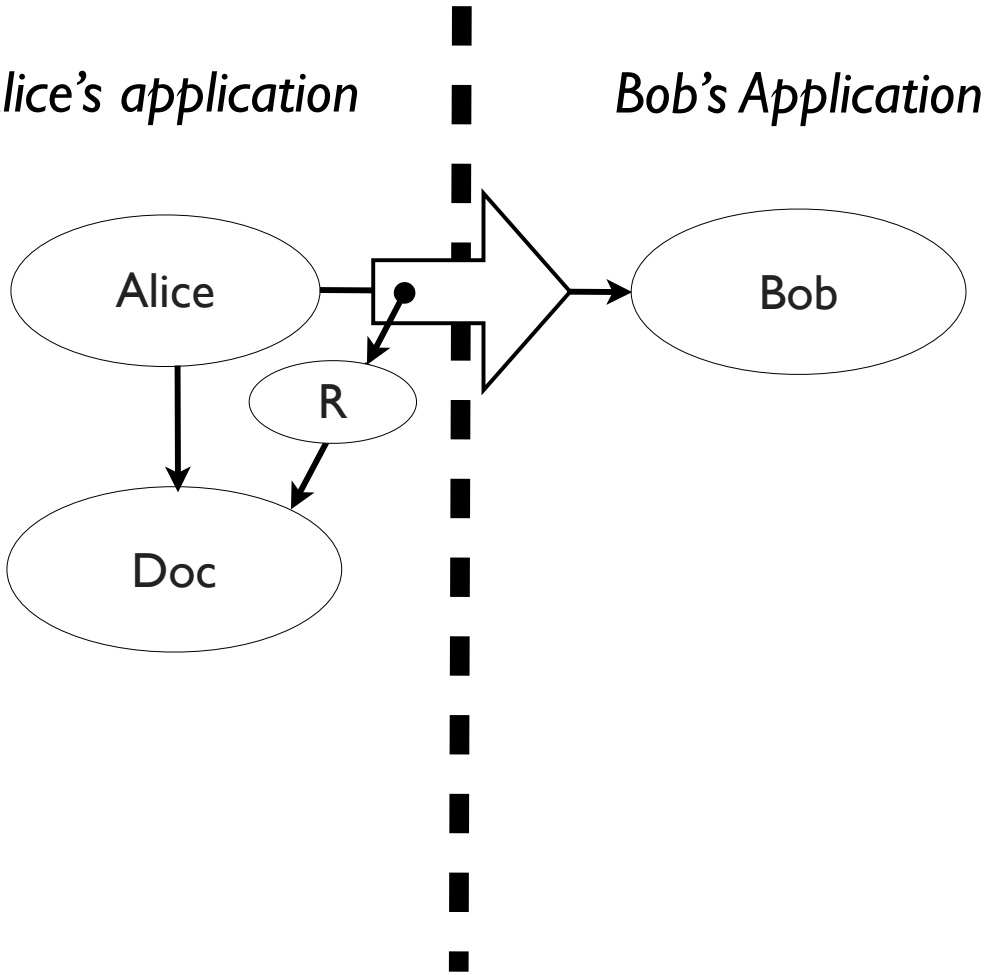


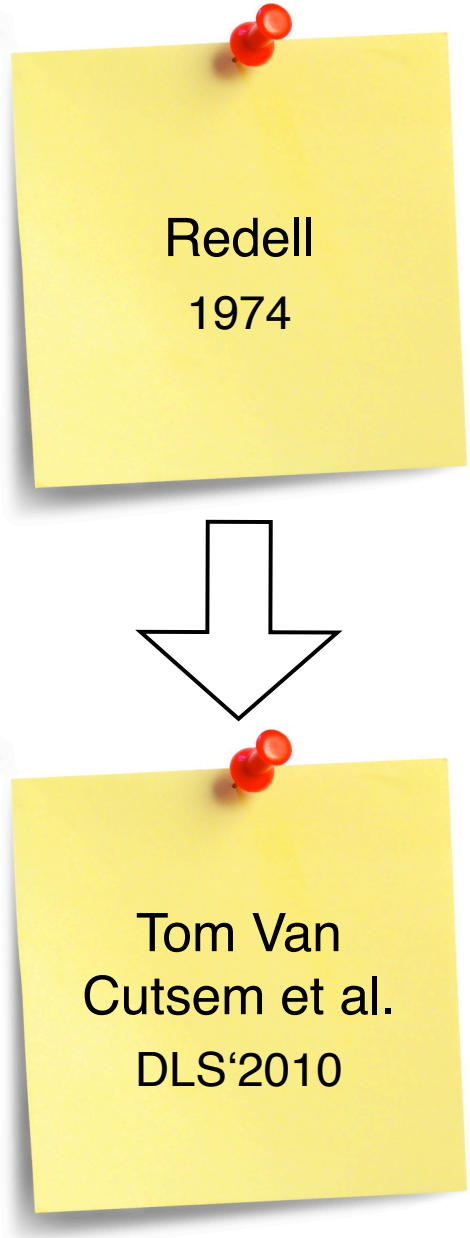
Redell
1974

Tom Van Cutsem et al.
DLS'2010

Handles at work

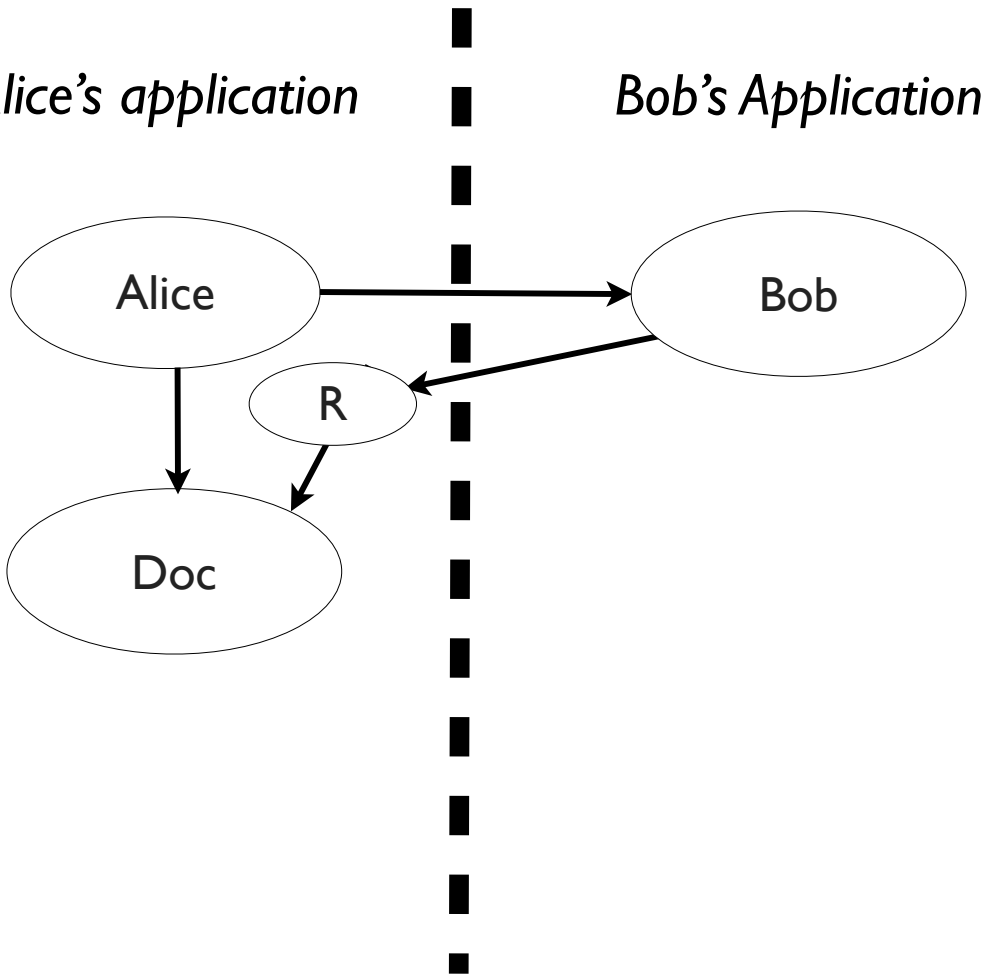
Revocable references

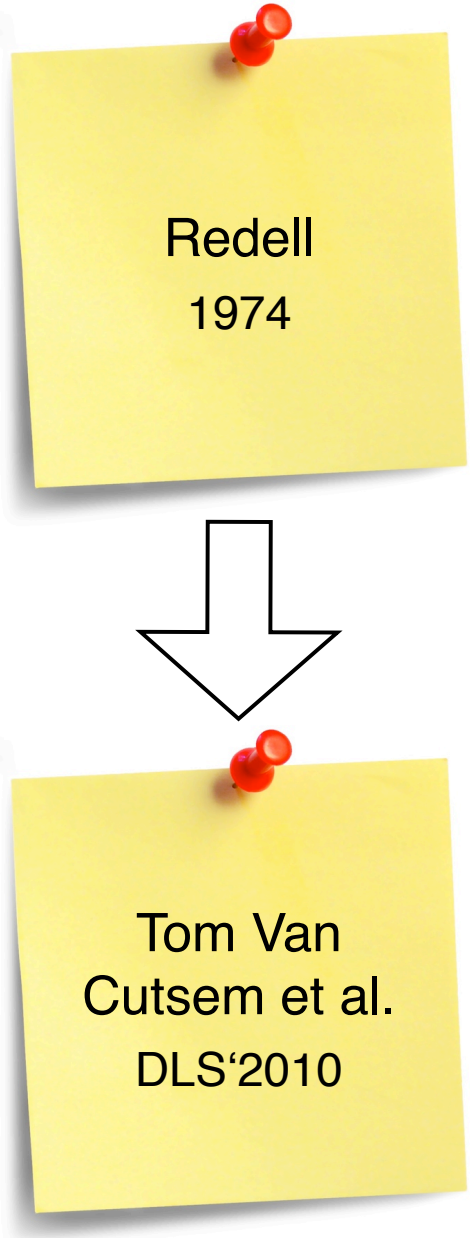




Handles at work

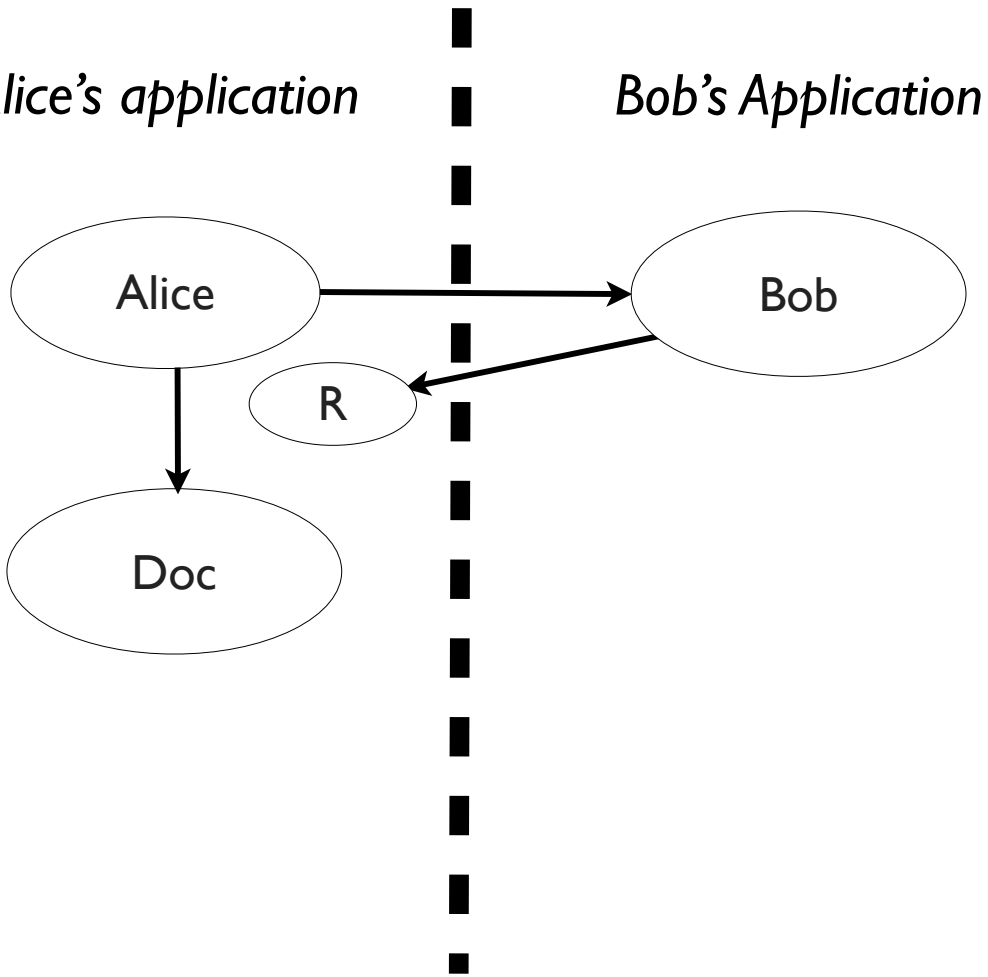
Revocable references

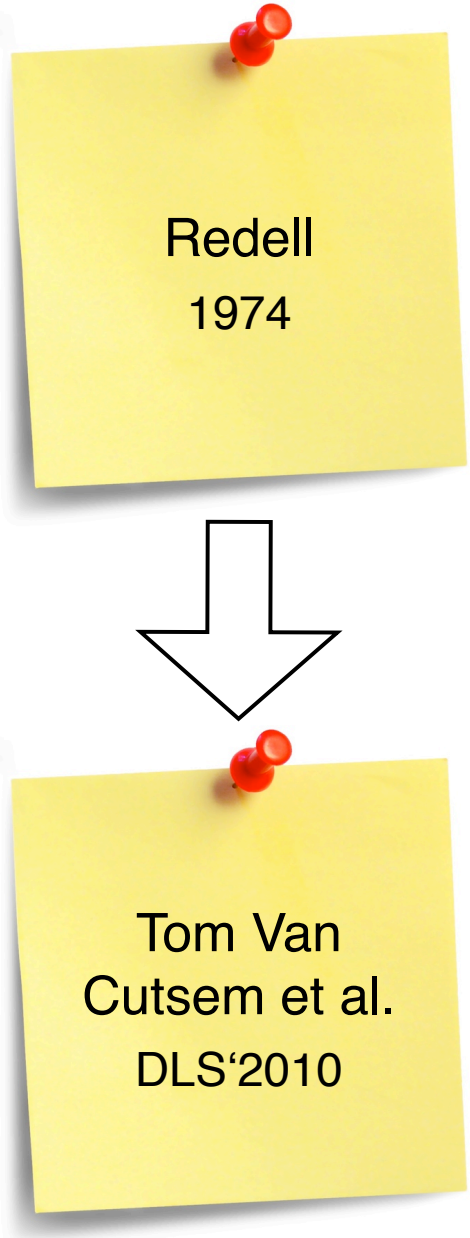


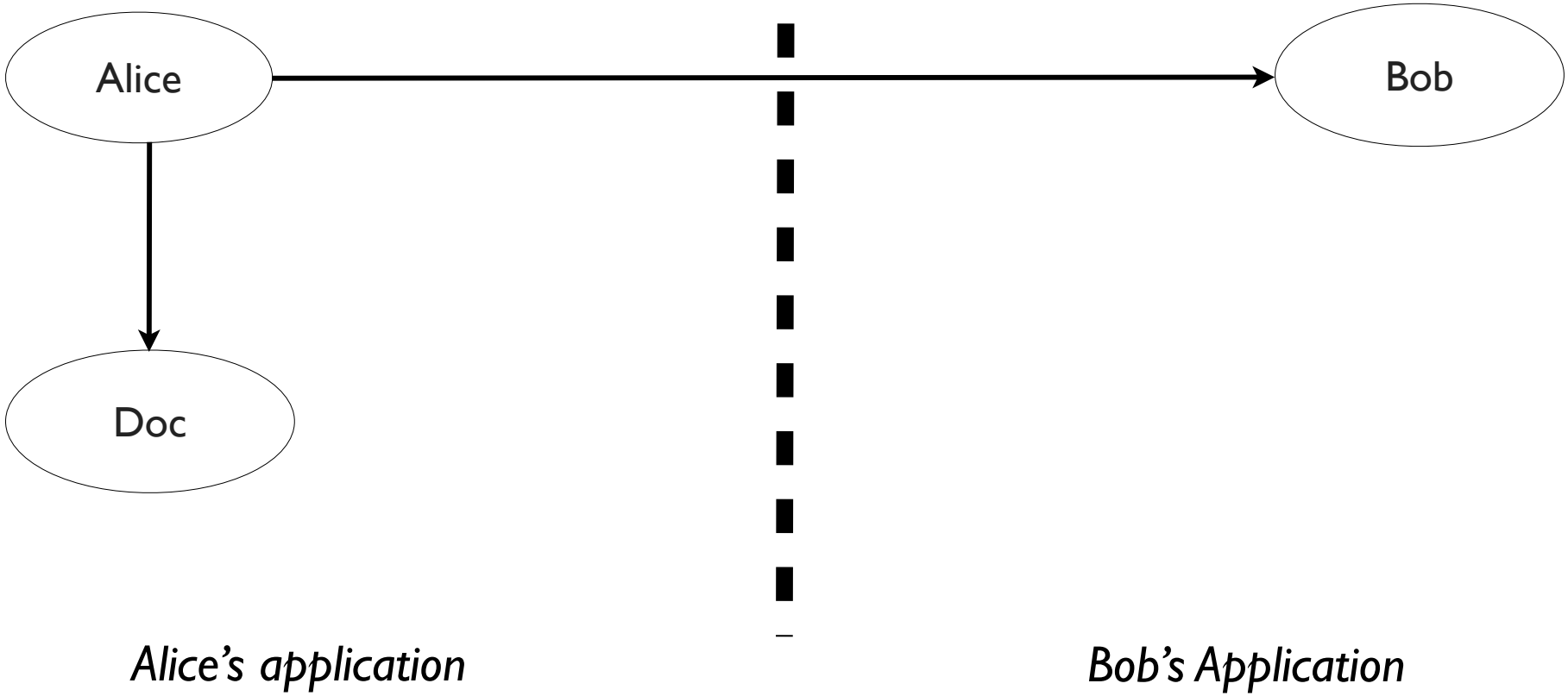


Handles at work

Revocable references

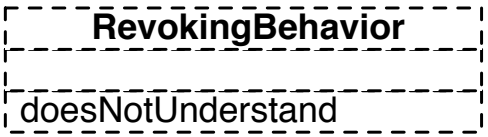


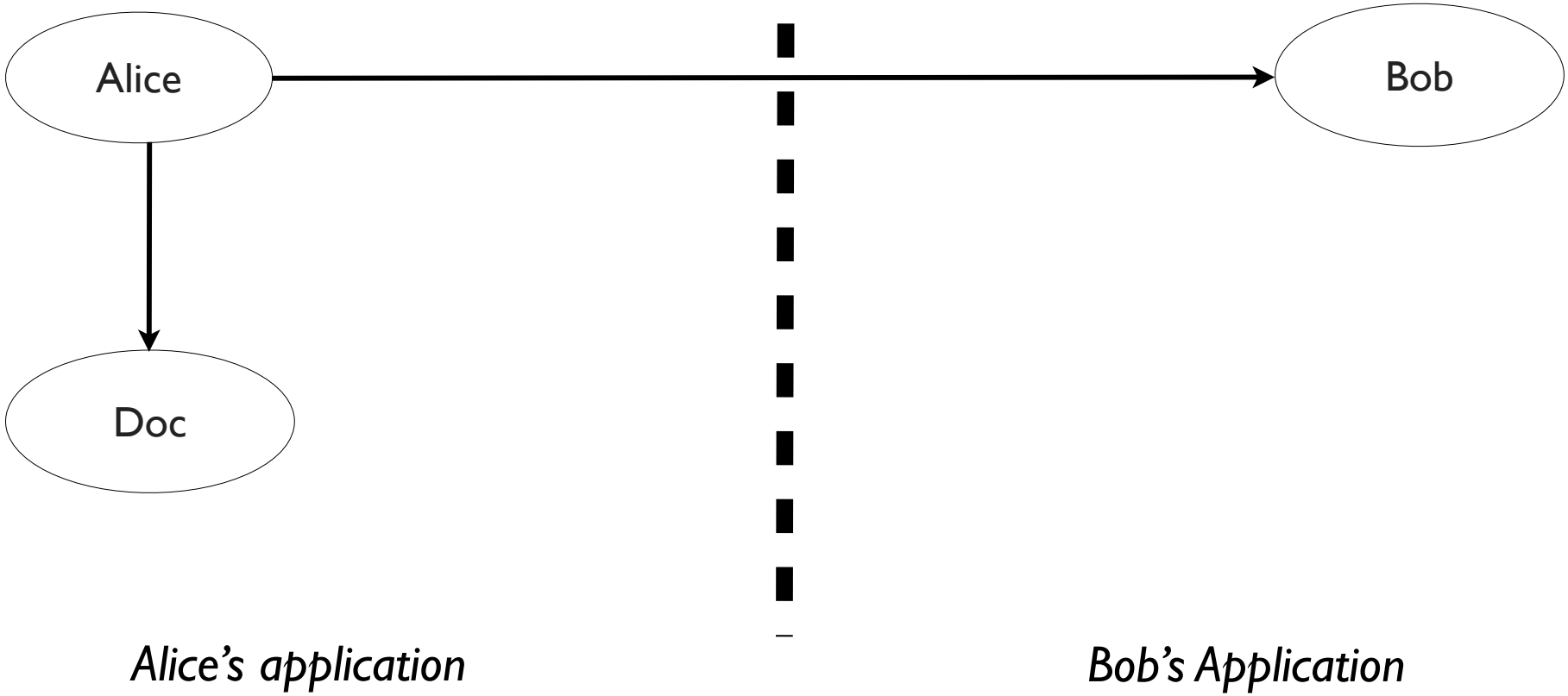




Specific semantics:

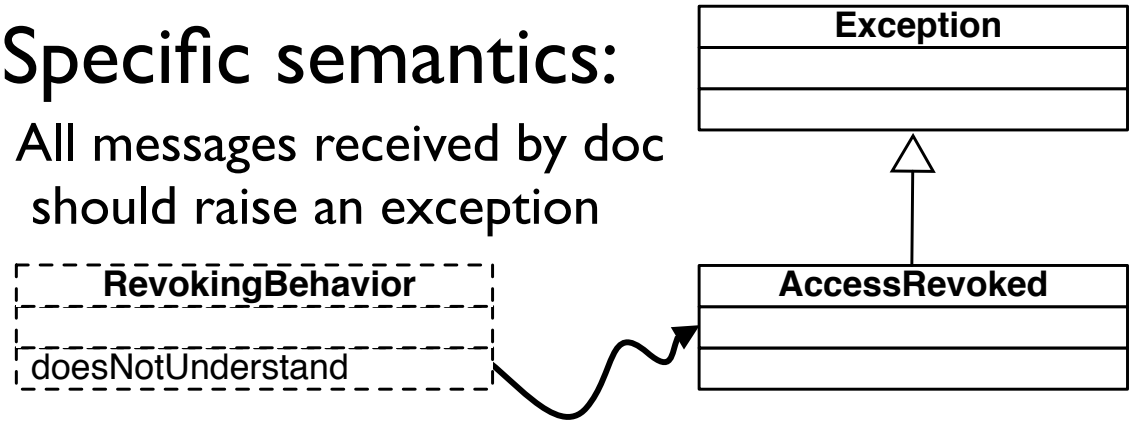
All messages received by doc should raise an exception

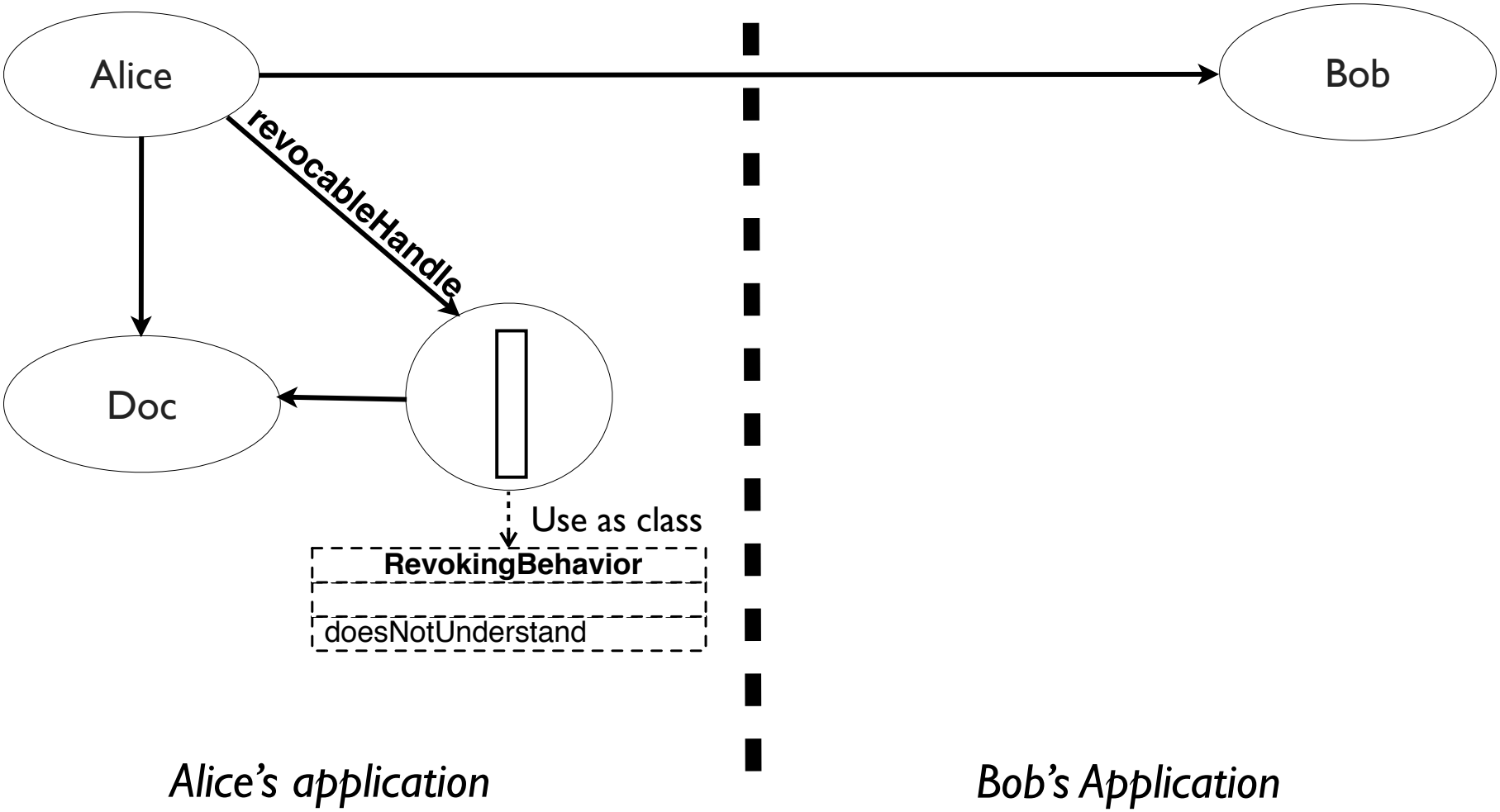


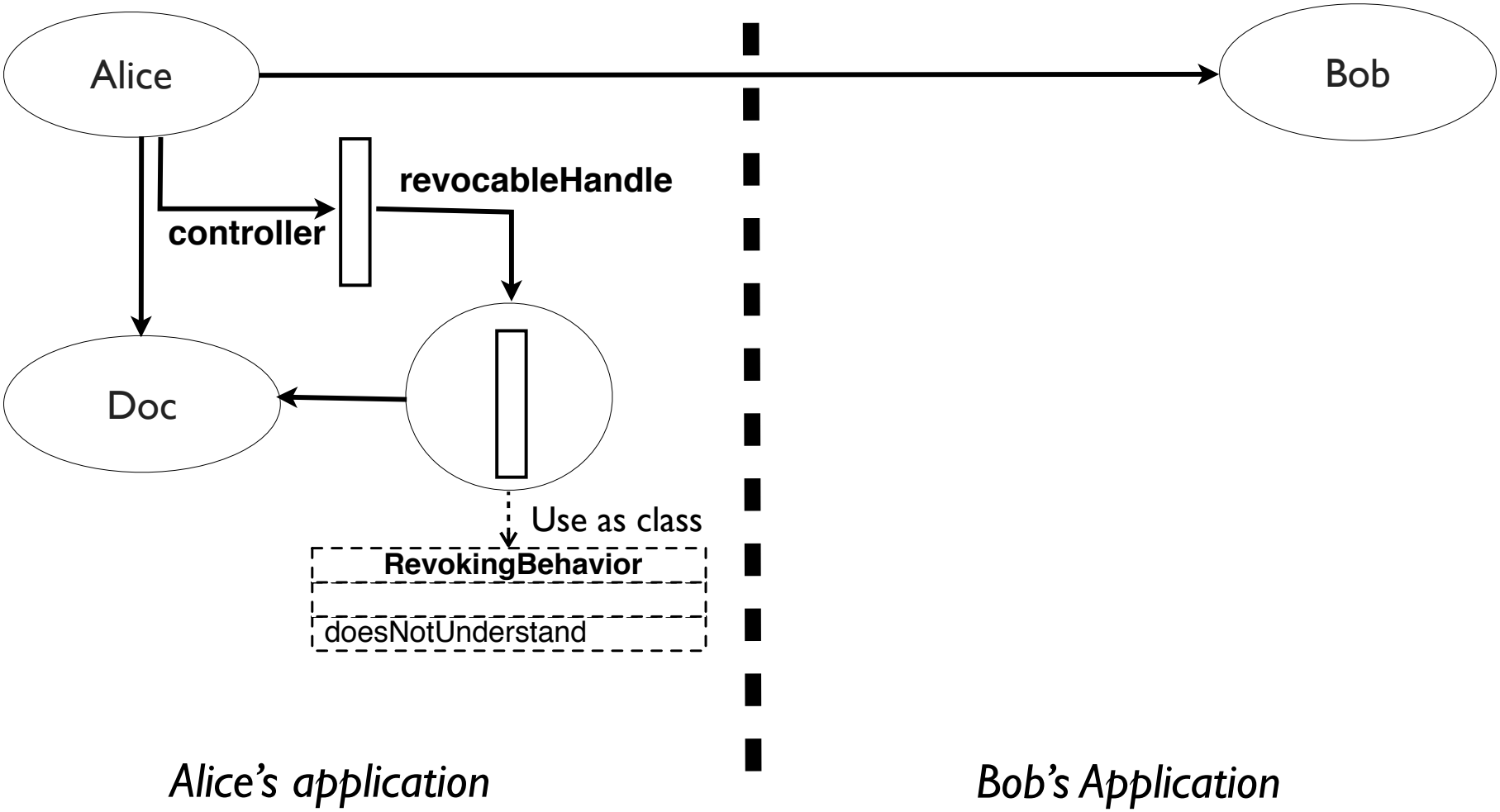


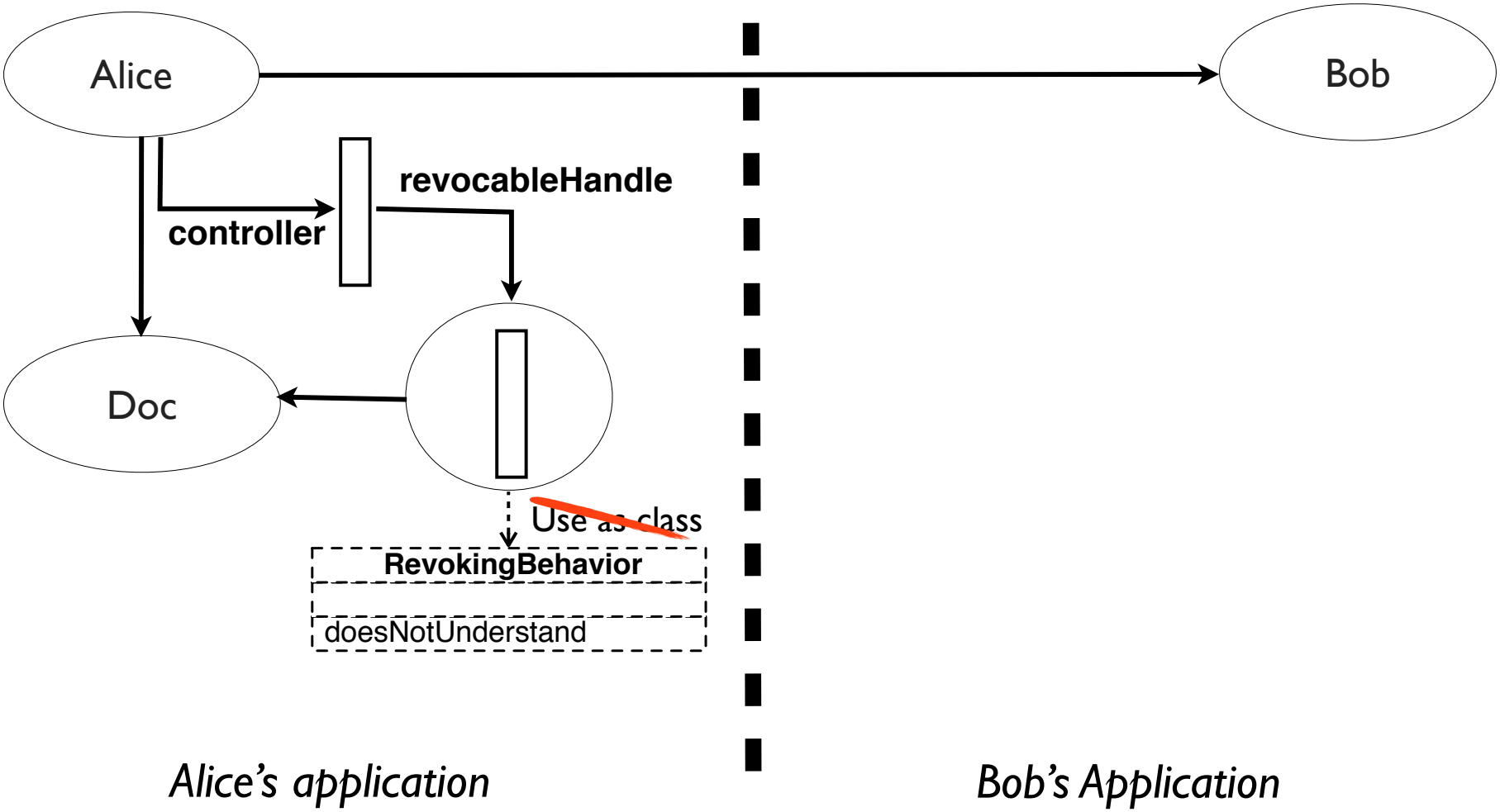
Specific semantics:

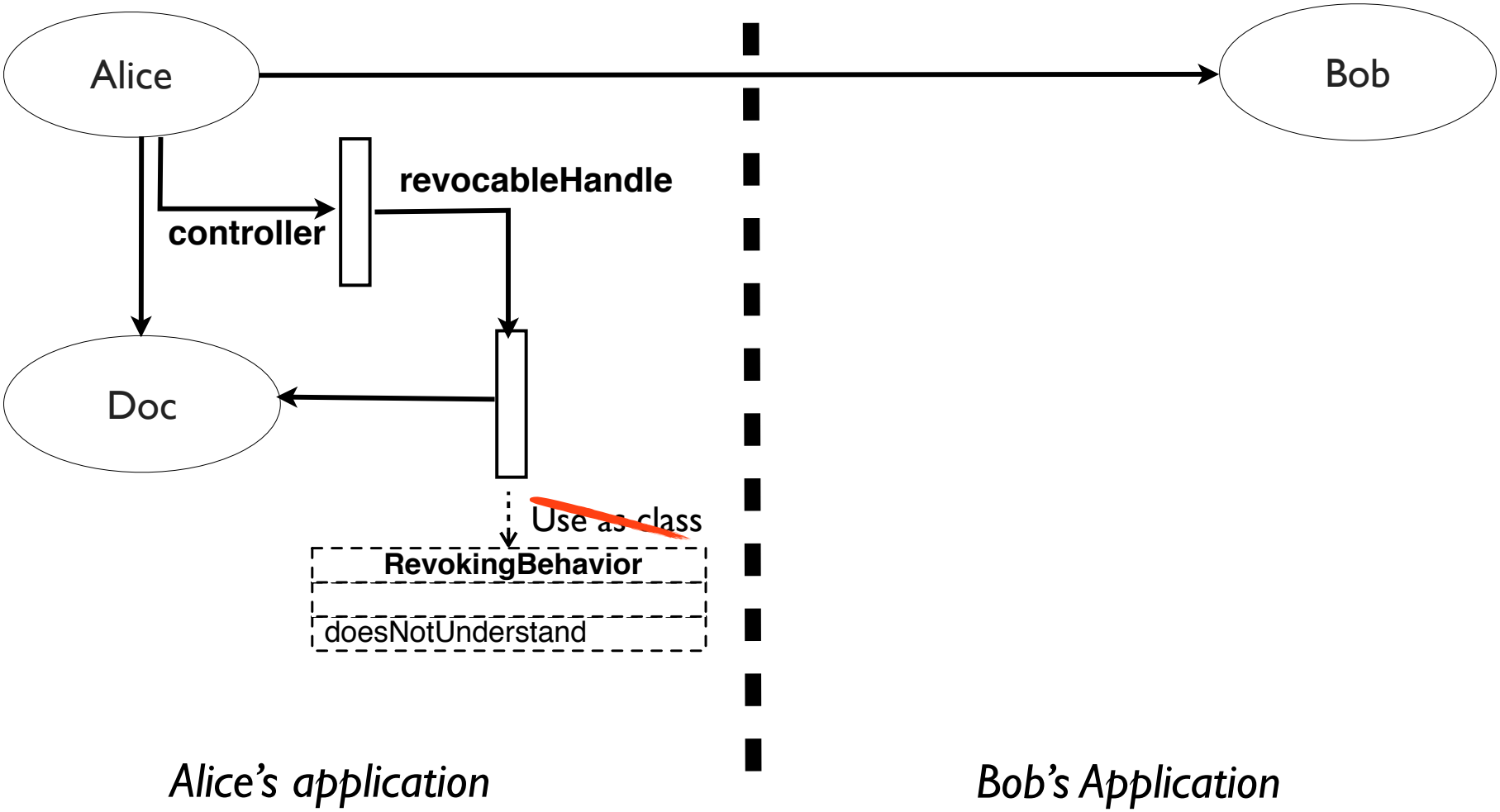
All messages received by doc should raise an exception

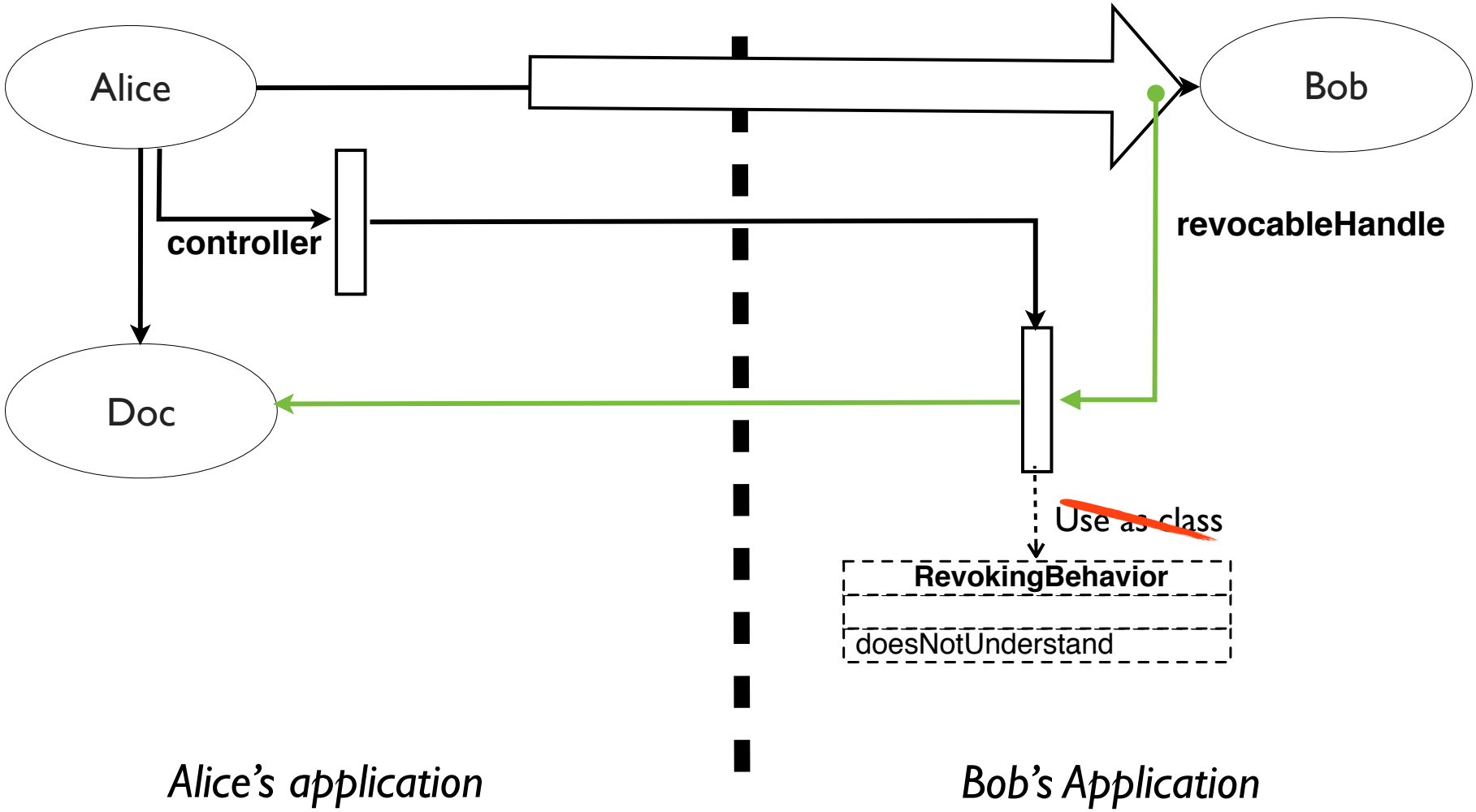


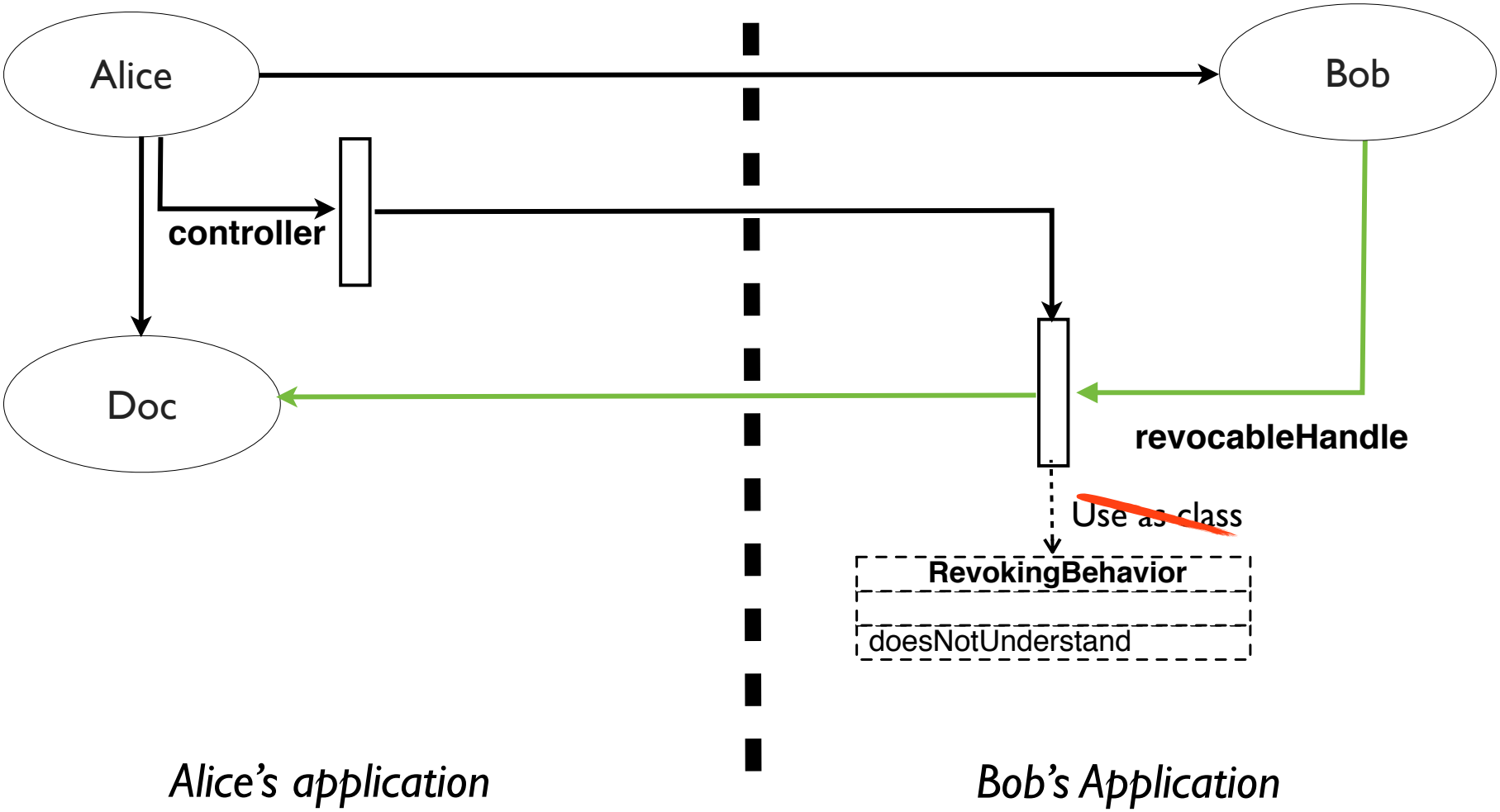

Alice
Bob
revocableHandle
Doc
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application

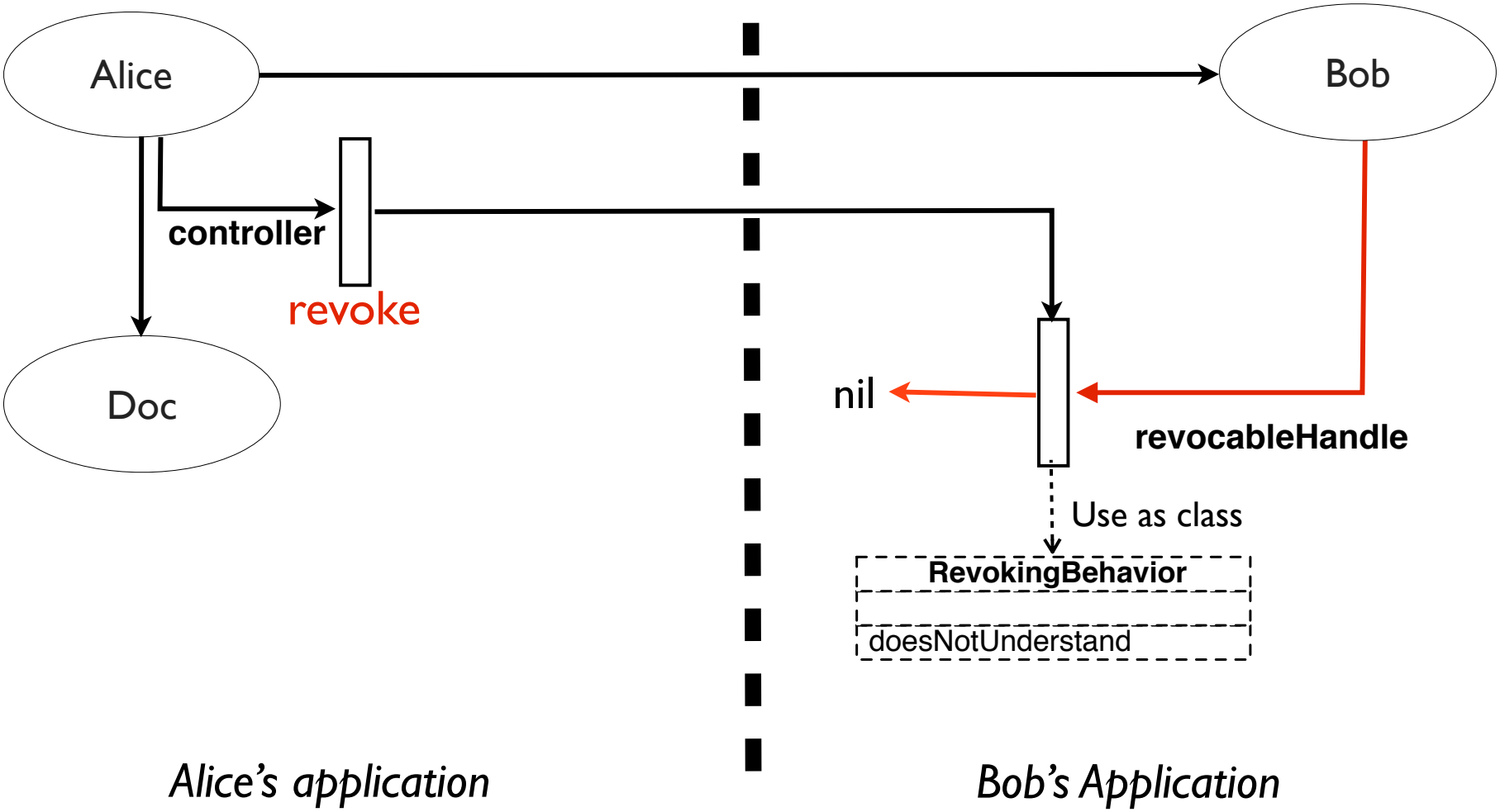

Alice
Bob
revocableHandle
controller
Doc
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application


Alice
Bob
revocableHandle
controller
Doc
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application


Alice
Bob
revocableHandle
controller
Doc
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application


Alice
Bob
controller
revocableHandle
Doc
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application


Alice
Bob
controller
Doc
revocableHandle
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application


Alice
Bob
controller
revoke
Doc
nil
revocableHandle
Use as class
RevokingBehavior
doesNotUnderstand
Alice's application
Bob's Application

I. Motivation

II. Approaches

III. Solution

IV. Consequences

V. Usages

VI. Validation

VII. Conclusion

Validation I

We formally described the Handle model:

Using the Felleisen and Hieb's notation

define properties at formal level

$p(c) \quad = \quad c'$

Where c' respects the properties p.

And c is a class.

Add handle to redex

$\varepsilon \quad = \quad [...] \mid h_p^o$

Handle send

$$P \vdash \langle E[h_p^o.m(v^*)], \mathcal{S}\rangle \hookrightarrow \langle E[h_p^o[\![e[v^*/x^*]]\!]_{c''}], \mathcal{S}\rangle \qquad [handled\ send]$$

Where $\langle c', m, x^*, e\rangle \in_P^* c''$

And c' *is a class supporting the property* p $(via\ p(c) = c')$

And c *is the class of the object* o

Validation 2

Implementation of the Handle's model on Pharo

Virtual machine:

- ✔ Send / State manage
- ✔ Propagated semantics
- ✔ Primitives

Language:

- ✔ Language interfaces
- ✔ Semantics

Validation 3

N-bodies benchmark: Stress the message send

	Means	Standard deviation
normal VM	4444.50ms	38.36ms
handle VM (Normal reference)	4772.80ms	20.68ms

7,36%

Binaries tree benchmark: Stress the state accesses

	Means	Standard deviation
normal VM	21167.00ms	106.26ms
handle VM (Normal reference)	22321.57ms	66.35ms

5,45%

Validation 3

N-bodies benchmark: Stress the message send

	Means	Standard deviation
normal VM	4444.50ms	38.36ms
handle VM (Normal reference)	4772.80ms	20.68ms
revocable nbody	8172.12ms	31.01ms

cost of the semantics 71%

Binaries tree benchmark: Stress the state accesses

1,747,535 graphs between 4 and 65,536 nodes

	Means	Standard deviation
normal VM	21167.00ms	106.26ms
handle VM (Normal reference)	22321.57ms	66.35ms
revocable Binaries tree	68094.23ms	70.06ms

cost of the semantics 205%

Validation 3

Behavior semantics changes

- Read-Only executions
- Revocable references

State semantics changes

- Worlds
- Software transactional memory

Validation 4

Behavior semantics changes

- Read-Only executions
- Revocable references

State semantics changes

- Worlds
- Software transactional memory

Implemented with:

- 2 Classes
- 14 methods
- Less than 50 lines of codes

Validation 4

Behavior semantics changes

- Read-Only executions
- Revocable references

State semantics changes

- Worlds
- Software transactional memory

Implemented with:

- 5 Classes
- 19 methods
- Less than 100 lines of codes

Validation 4

Behavior semantics changes

- Read-Only executions
- Revocable references

State semantics changes

- Worlds
- Software transactional memory

Implemented with:

- 2 Classes
- 16 methods
- Less than 50 lines of codes

Validation 4

Behavior semantics changes

- Read-Only executions
- Revocable references

State semantics changes

- Worlds
- Software transactional memory

Implemented with:

- 4 Classes
- 35 methods
- Less than 200 lines of codes

Publications

International conference

Read-Only Execution for Dynamic Languages. In Proceedings of the 48th International Conference Objects, Models, Components, Patterns (TOOLS'10), Malaga, Spain, June 2010.

Journal paper

Behavior-Propagating First Class References For Dynamically-Typed Languages. In Science of Computer Programming, 2013. Under-revision.

Conclusion

Our Solution

- ✔ Reifying references
- ✔ Controlling behavior
- ✔ Isolating state
- ✔ Identity is preserved
- ✔ Propagated semantics
- ✔ Real time Control
- ✔ Reflection

Validation

- ✔ Operational semantic
- ✔ Implementation
- ✔ Acceptable performance
- ✔ Semantics

Future work:

- Using Handle on a real world application
- Composition model (ReadOnly + Revocable)
- State control (add instance variables, deltas, etc.)

Towards First Class References as a Security Infrastructure in Dynamically-Typed Languages

Jean-Baptiste Arnaud

Supervisor: Stéphane Ducasse
Co-Supervisor: Marcus Denker
RMoD team

February 18, 2013