13. Traits

Selected literature

- Cook. ***Interfaces and Specifications for the Smalltalk-80 Collection Classes***. OOPSLA 1992
- Taivalsaari. ***On the Notion of Inheritance***. ACM Computing Surveys, September 1996.
- Black, et al. ***Applying Traits to the Smalltalk Collection Hierarchy***. OOPSLA 2003
- Ducasse, et al. ***Traits: A Mechanism for fine-grained Reuse***. ACM TOPLAS, March 2006.
- Cassou, et al. ***Traits at Work: the design of a new trait-based stream library***. JCLSS 2009

http://scg.unibe.ch/scgbib?query=stlit-traits

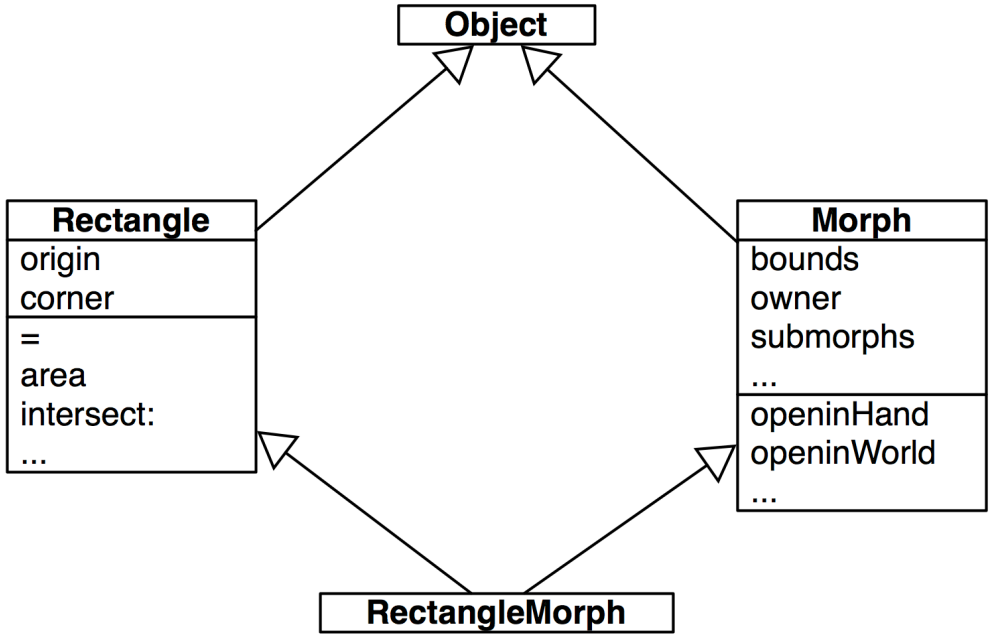
Roadmap

- Why traits?
- Traits in a Nutshell
- Case study — Streams
- Traits in Pharo
- Future of Traits

Roadmap

- **Why traits?**
- Traits in a Nutshell
- Case study — Streams
- Traits in Pharo
- Future of Traits

Problem: how to share behaviour across class hierarchies?



There are hundreds of methods we would like `RectangleMorph` to inherit from both `Rectangle` and `Morph`

The trouble with Single Inheritance

- Where to put the shared behaviour?
  - Sharing too high ⇒ inappropriate methods must be “cancelled”
- Duplicating code
  - Impacts maintenance
- Delegate
  - Ugly boilerplate delegation code

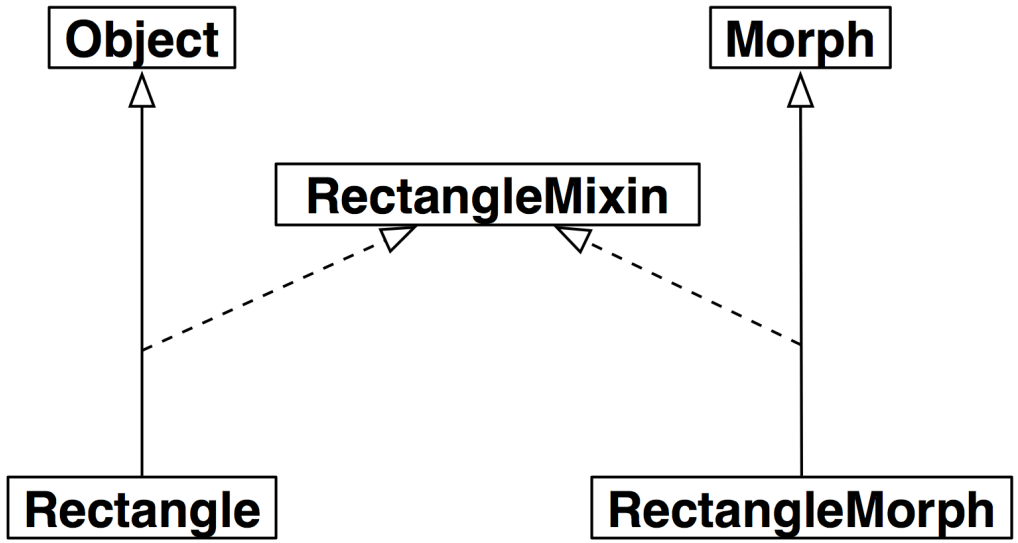
The trouble with Multiple Inheritance

```
Rectangle selectors select:
  [:s | Morph selectors includes: s]
```

- Conflicts must be resolved
  - Implicit resolution leads to fragility when refactoring
- No unique super class
  - Must explicitly name super methods to compose them
- Diamond problem
  - What to do about features inherited along two paths?

```
an IdentitySet
(#topRight
#align:with: #right:
#leftCenter #bottom
#center #height
#right #topCenter
#extent #bottomCenter
#topLeft #width
#printOn:
#containsPoint: #left
#top #intersects:
#bottomLeft #bottom:
#bottomRight #top:
#left: #rightCenter)
```

Mixins extend single inheritance with features that can be mixed into a class



The trouble with Mixins

- Mixins are composed linearly to resolve conflicts
  - Conflict resolution is sensitive to mixin composition order
  - Composing entity has no control!
- Fragile hierarchy
  - Changes may impact distant classes

Roadmap

- Why traits?
- **Traits in a Nutshell**
- Case study — Streams
- Traits in Pharo
- Future of Traits

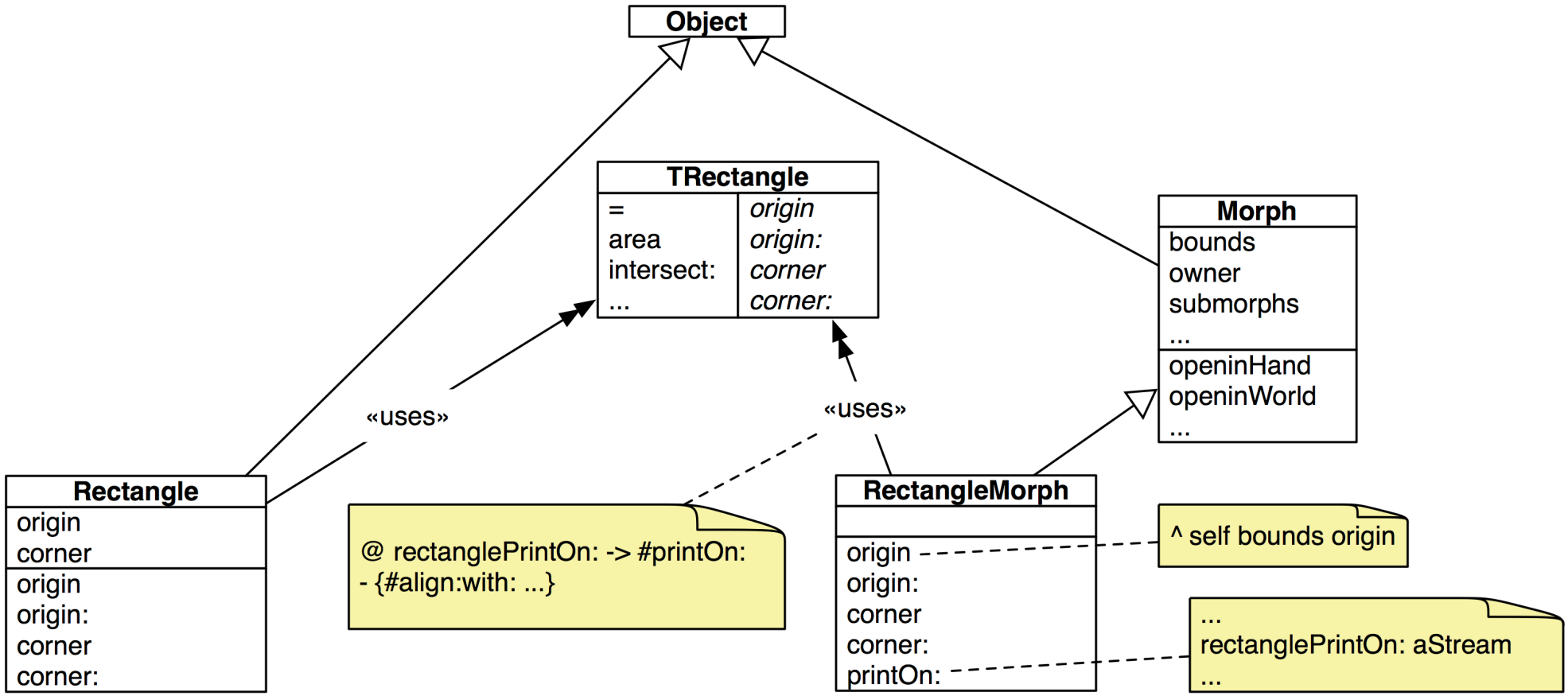
Traits are parameterized behaviours

- A trait
  - *provides* a set of methods
  - *requires* a set of methods
  - may be *composed* of other traits
- *Traits do not specify any state!*

TRectangle	
= area intersect: ...	*origin* *origin:* *corner* *corner:*

```
= aRectangle
  ^ self species = aRectangle species
    and: [self origin = aRectangle origin]
    and: [self corner = aRectangle corner]
```

Class = superclass + state + traits + glue



The class retains full control of the composition

Both traits and classes can be composed of traits

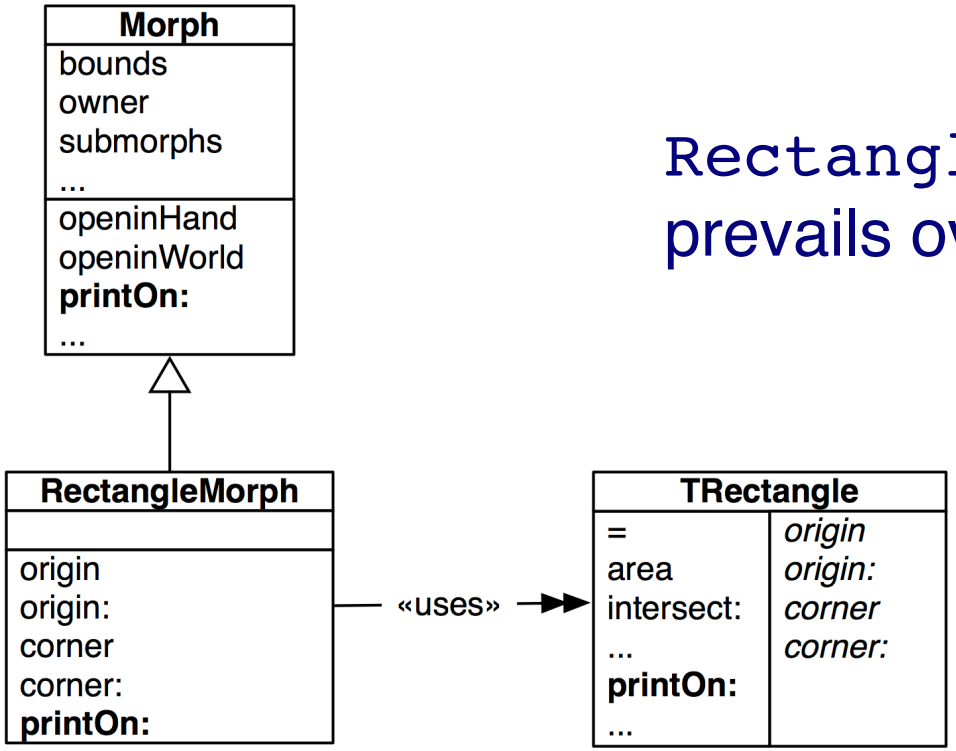
```
Trait named: #NSTPuttablePositionableStream
  uses: NSTPuttableStream + NSTPositionableStream
  category: 'Nile-Base-Traits'
```

```
Object subclass: #NSTextStream
  uses: NSTPuttablePositionableStream + NSTCharacterWriting
  instanceVariableNames: 'collection position writeLimit readLimit'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Nile-Clients-TextStream'
```

Trait composition rules

1. **Class methods take precedence over trait methods**
2. Conflicts are resolved explicitly
3. Traits can be flattened away

Class methods take precedence over trait methods



`RectangleMorph>>printOn:` prevails over `Morph>>printOn:`

Trait composition rules

1. Class methods take precedence over trait methods
2. **Conflicts are resolved explicitly**
3. Traits can be flattened away

Conflicts are resolved explicitly

```
RectangleMorph subclass: #Morph
  uses: TRectangle @ {rectanglePrintOn: -> #printOn:}
        – {#align:with: . #topRight . … }
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Morphic-TraitsDemo'
```

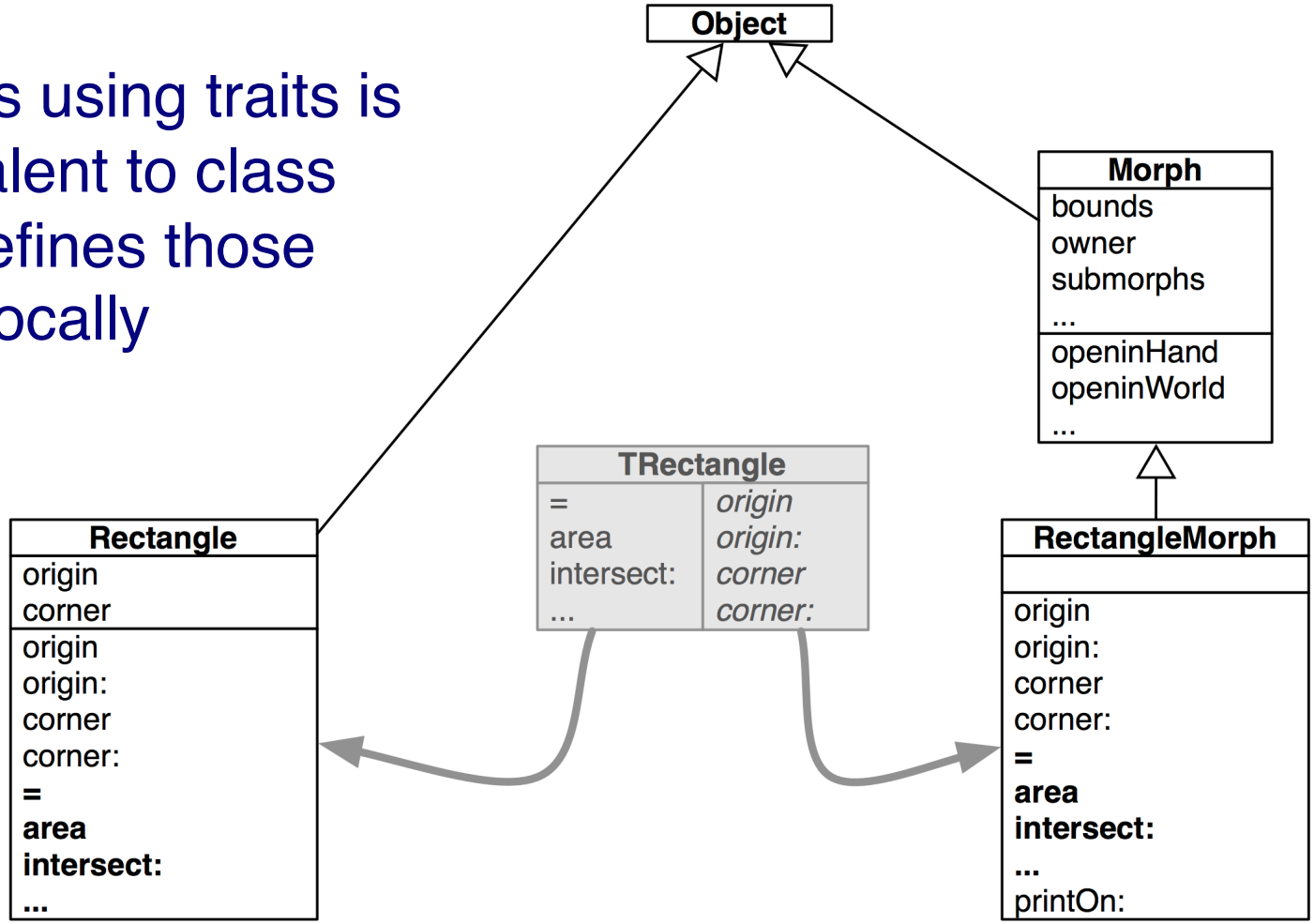
Aliasing introduces an additional name for a method
Exclusion removes a method from a trait

Trait composition rules

1. Class methods take precedence over trait methods
2. Conflicts are resolved explicitly
3. **Traits can be flattened away**

Traits can be flattened away

A class using traits is equivalent to class that defines those traits locally

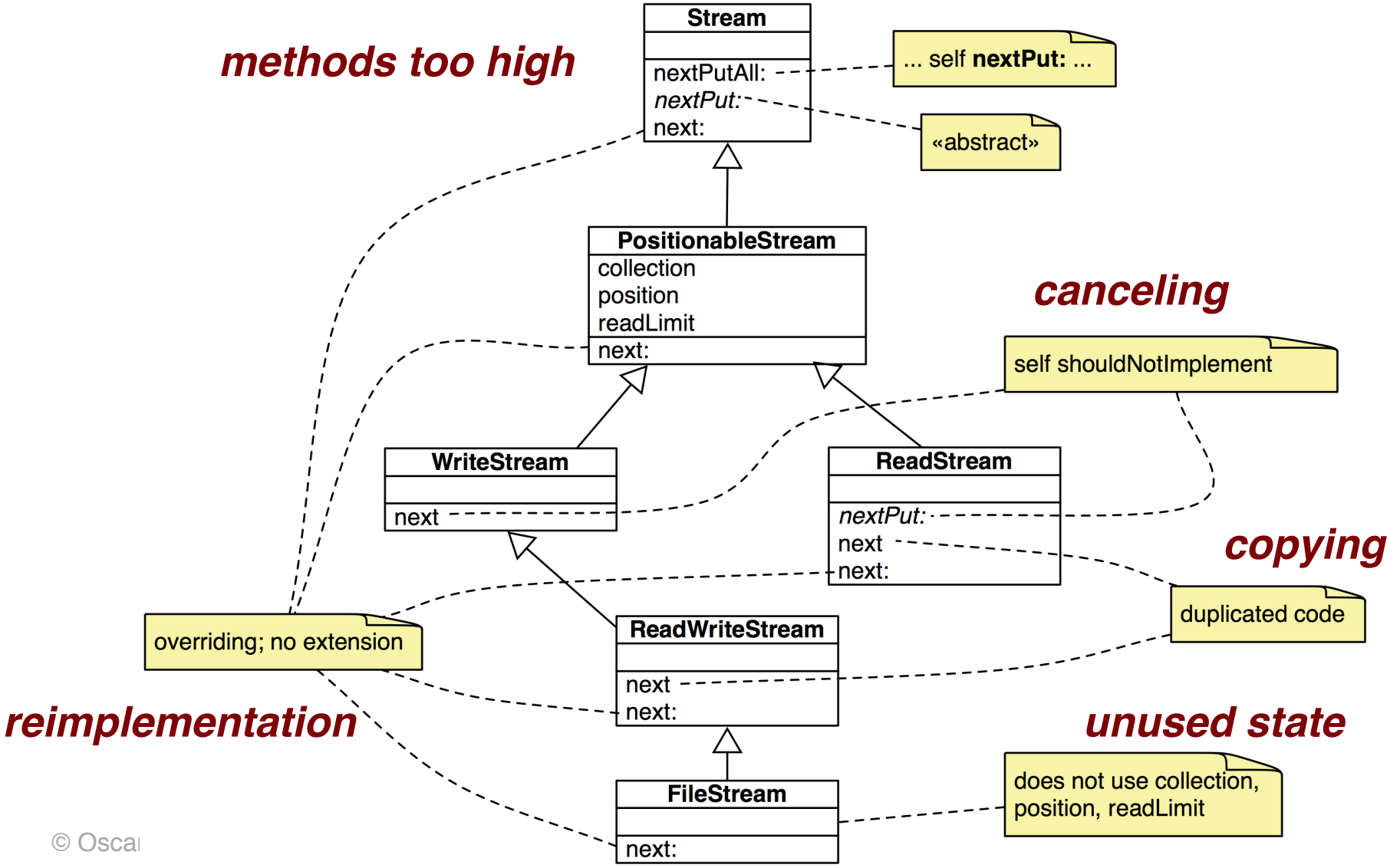


Roadmap

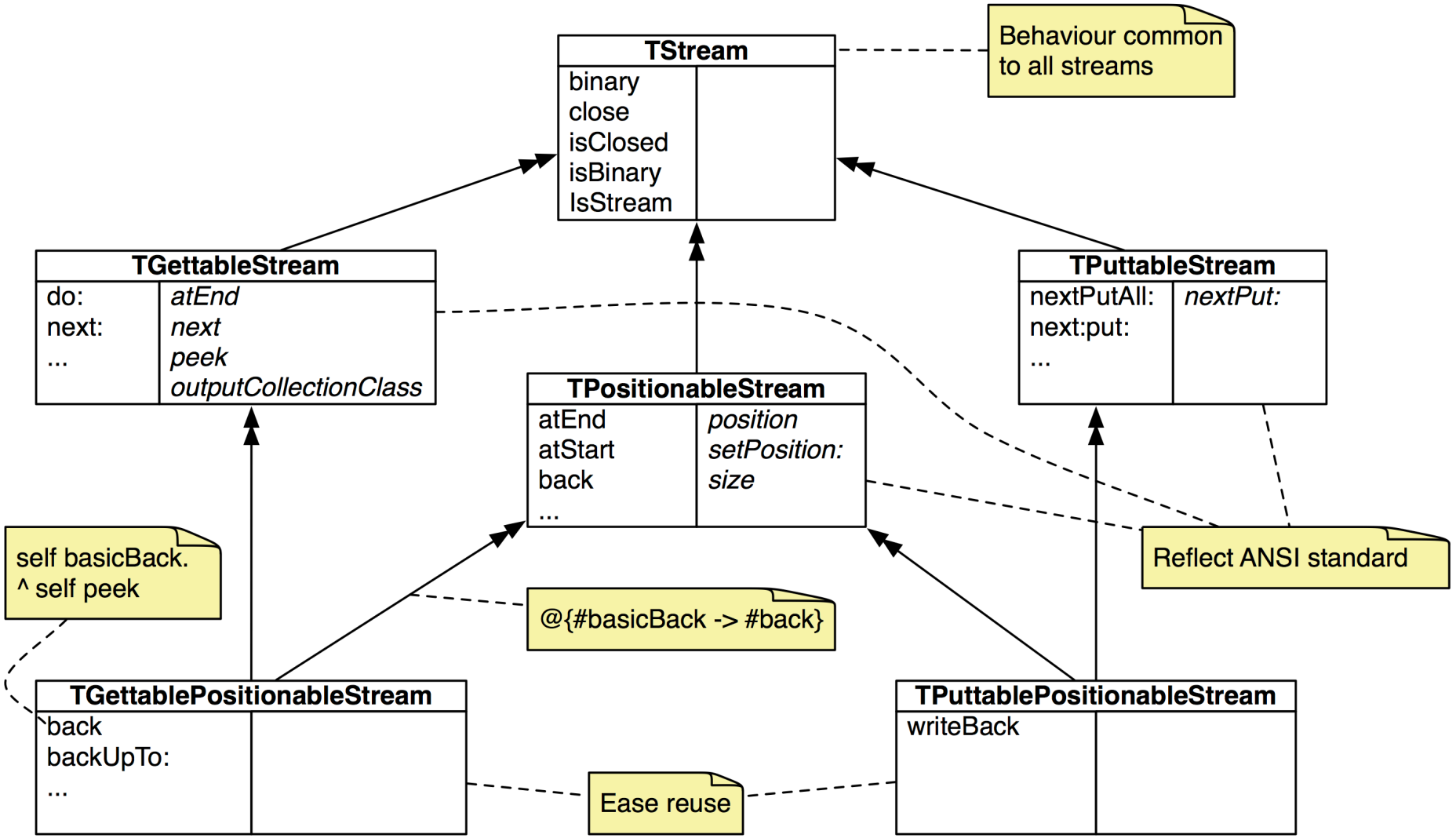
- Why traits?
- Traits in a Nutshell
- **Case study — Streams**
- Traits in Pharo
- Future of Traits

Cassou, et al. *Traits at Work: the design of a new trait-based stream library*. JCLSS 2009.

The trouble with Streams

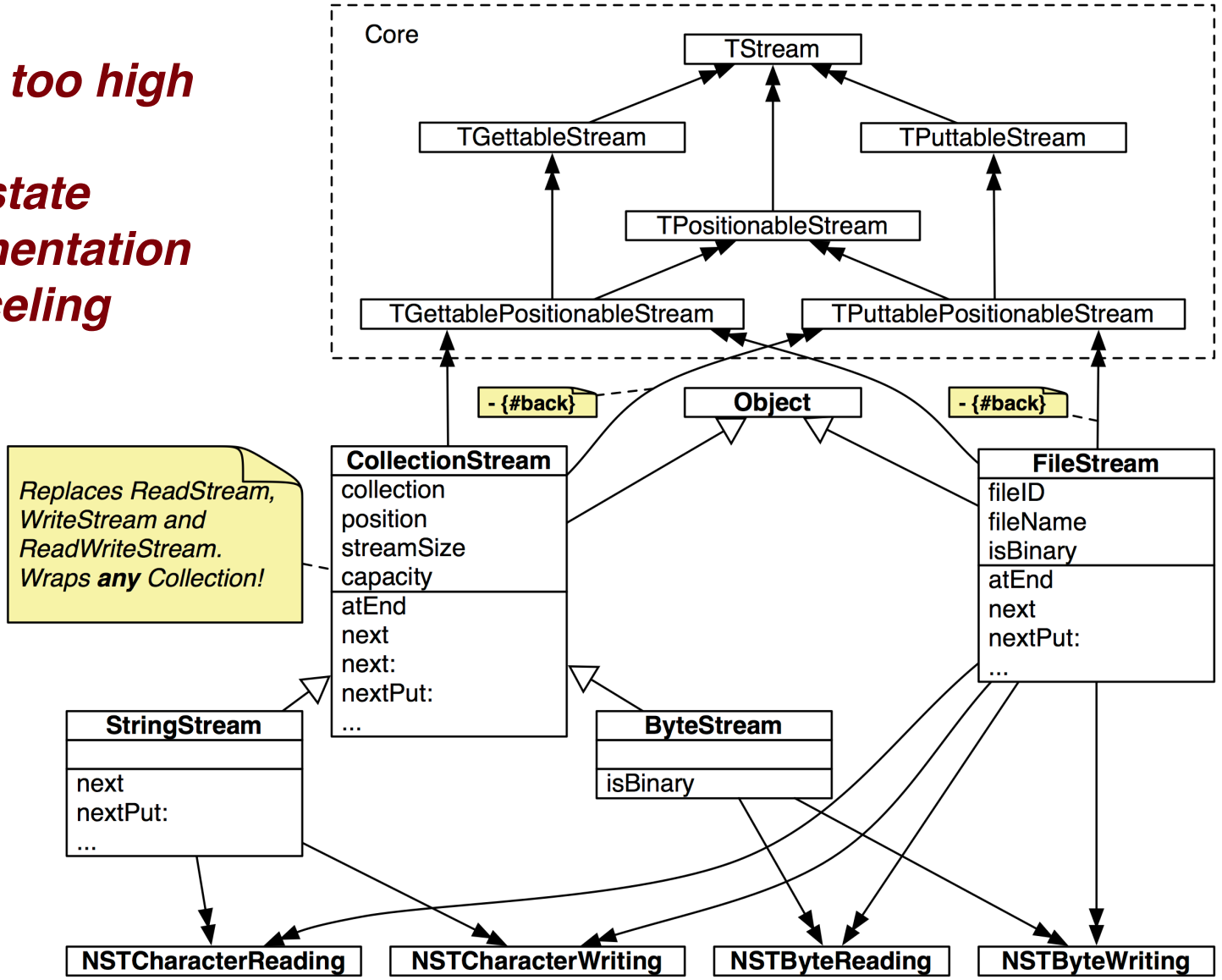


The Nile core

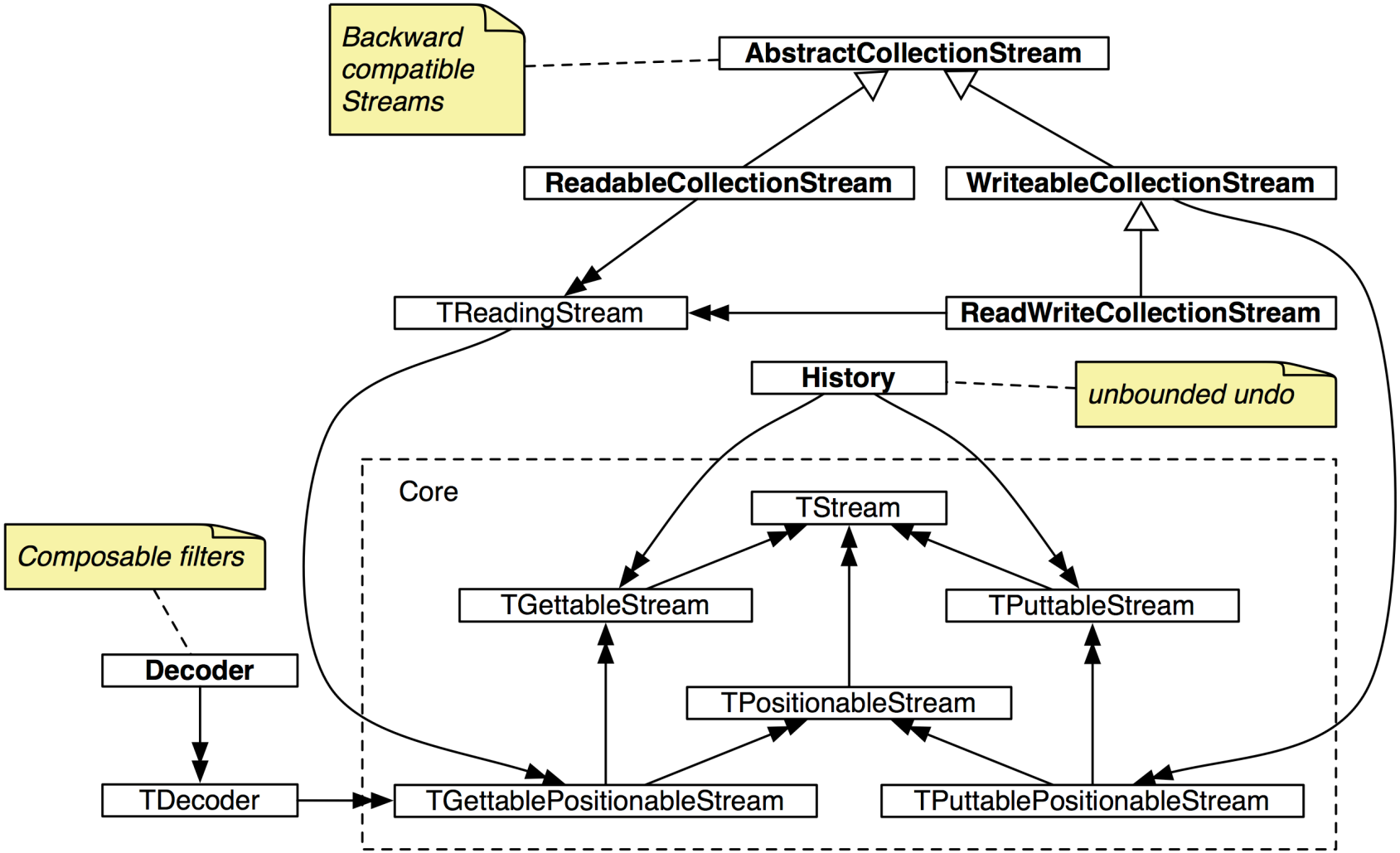


Nile Stream classes

no methods too high
no copying
no unused state
no reimplementation
limited canceling



Other Nile Stream classes



Assessment

- High reuse achieved
  - 40% less code in Stream hierarchy
- More general abstractions
  - Streams on *any* Collection
  - With equal or better performance
- Design traits around abstractions, not reuse
  - Avoid too fine-grained traits
- Traits or classes?
  - Prefer classes — use traits to resolve design conflicts

Roadmap

- Why traits?
- Traits in a Nutshell
- Case study — Streams
- **Traits in Pharo**
- Future of Traits

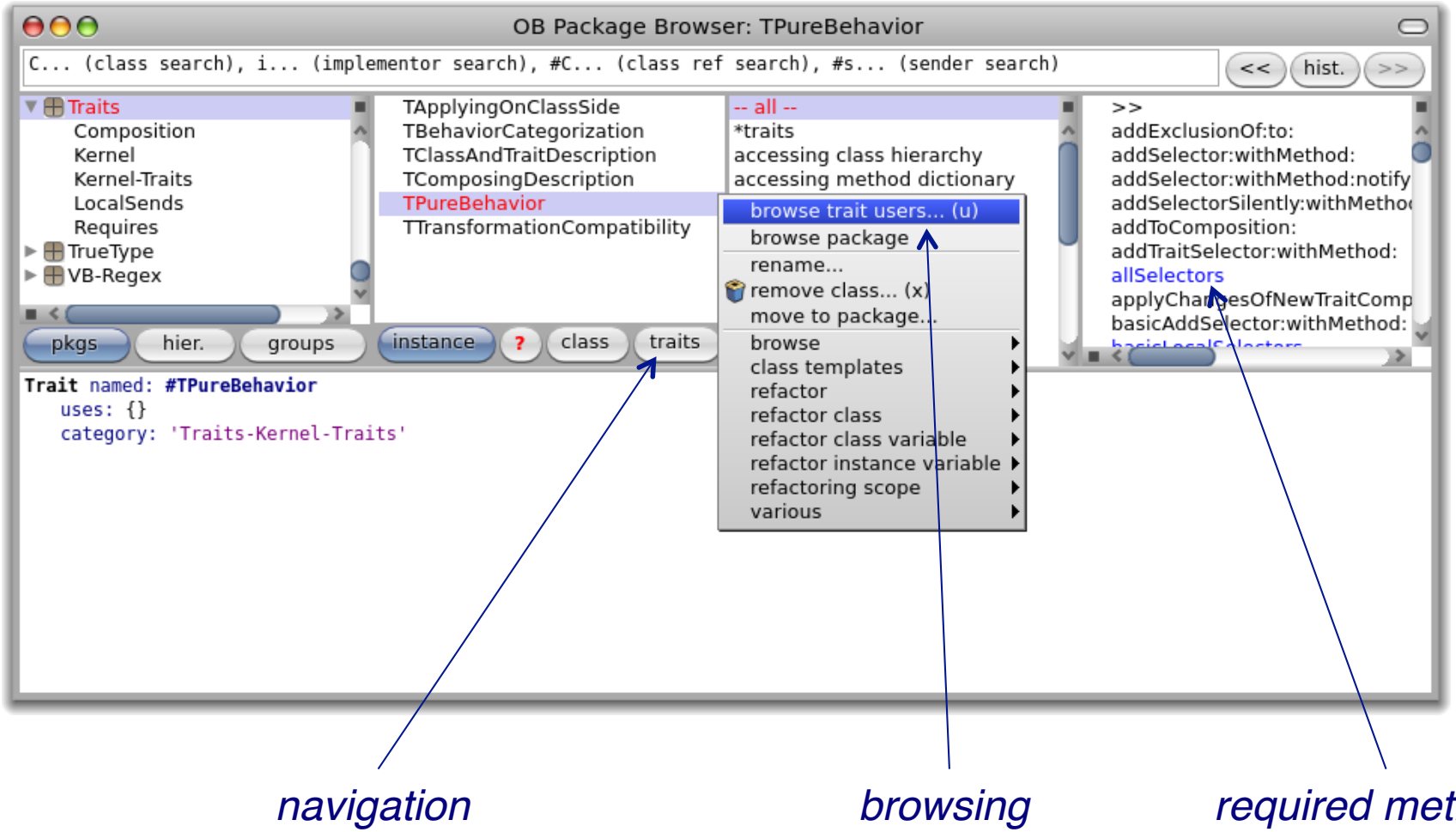
Traits in Pharo

- Language Extension
  - Extended the language kernel to represent traits
  - Modified the compilation process for classes built from traits
- No changes to the VM
  - Essentially no runtime performance penalty
  - Except indirect instance variable access
  - But: This is common practice anyway
- No duplication of source code
  - Only byte-code duplication when installing methods

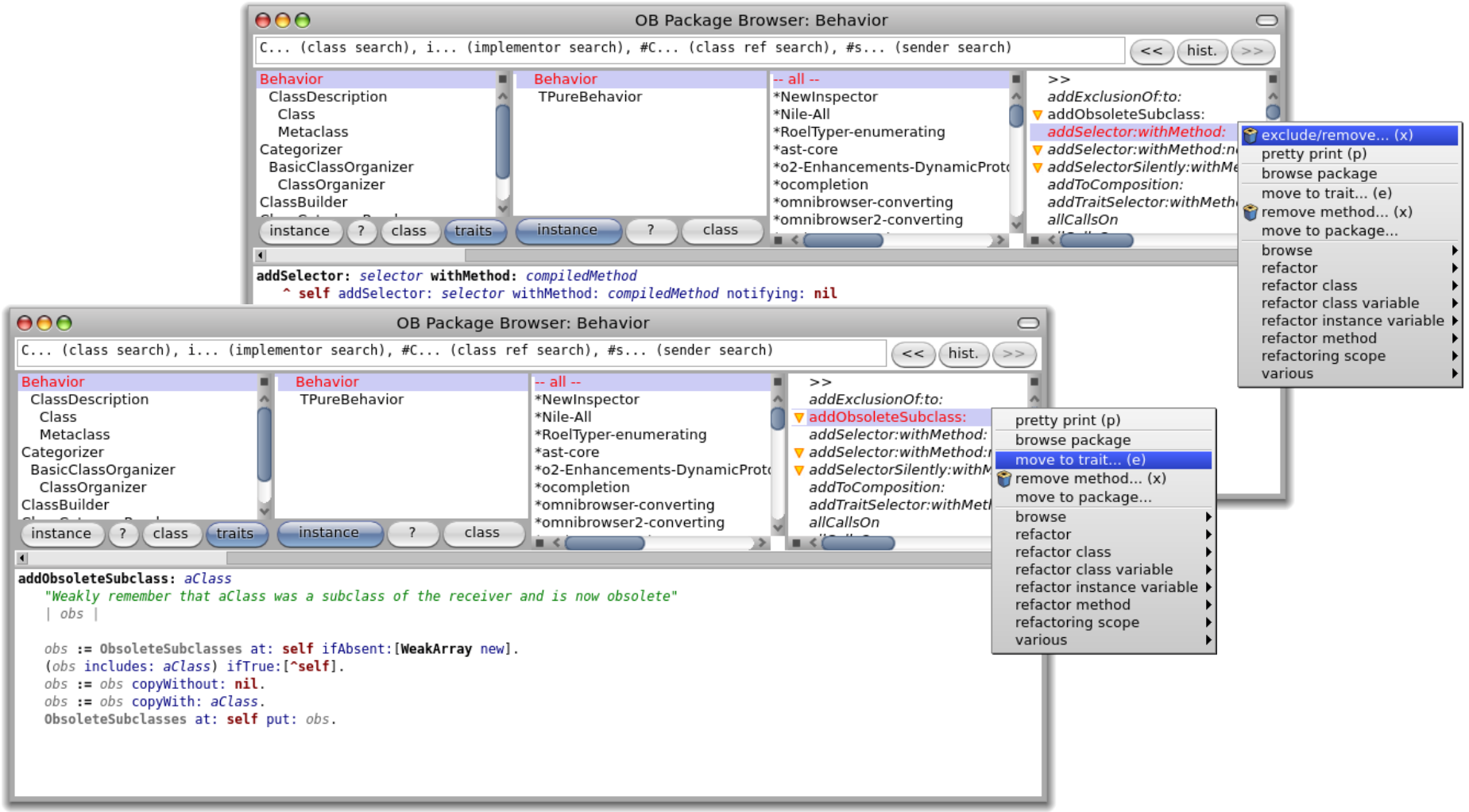
Traits in Pharo 1.0

```
Object subclass: #Behavior
   uses: TPureBehavior @
      { #basicAddTraitSelector:withMethod:
         -> #addTraitSelector:withMethod: }
   instanceVariableNames: 'superclass methodDict format
      traitComposition localSelectors'
   classVariableNames: 'ObsoleteSubclasses'
   poolDictionaries: ''
   category: 'Kernel-Classes'
```

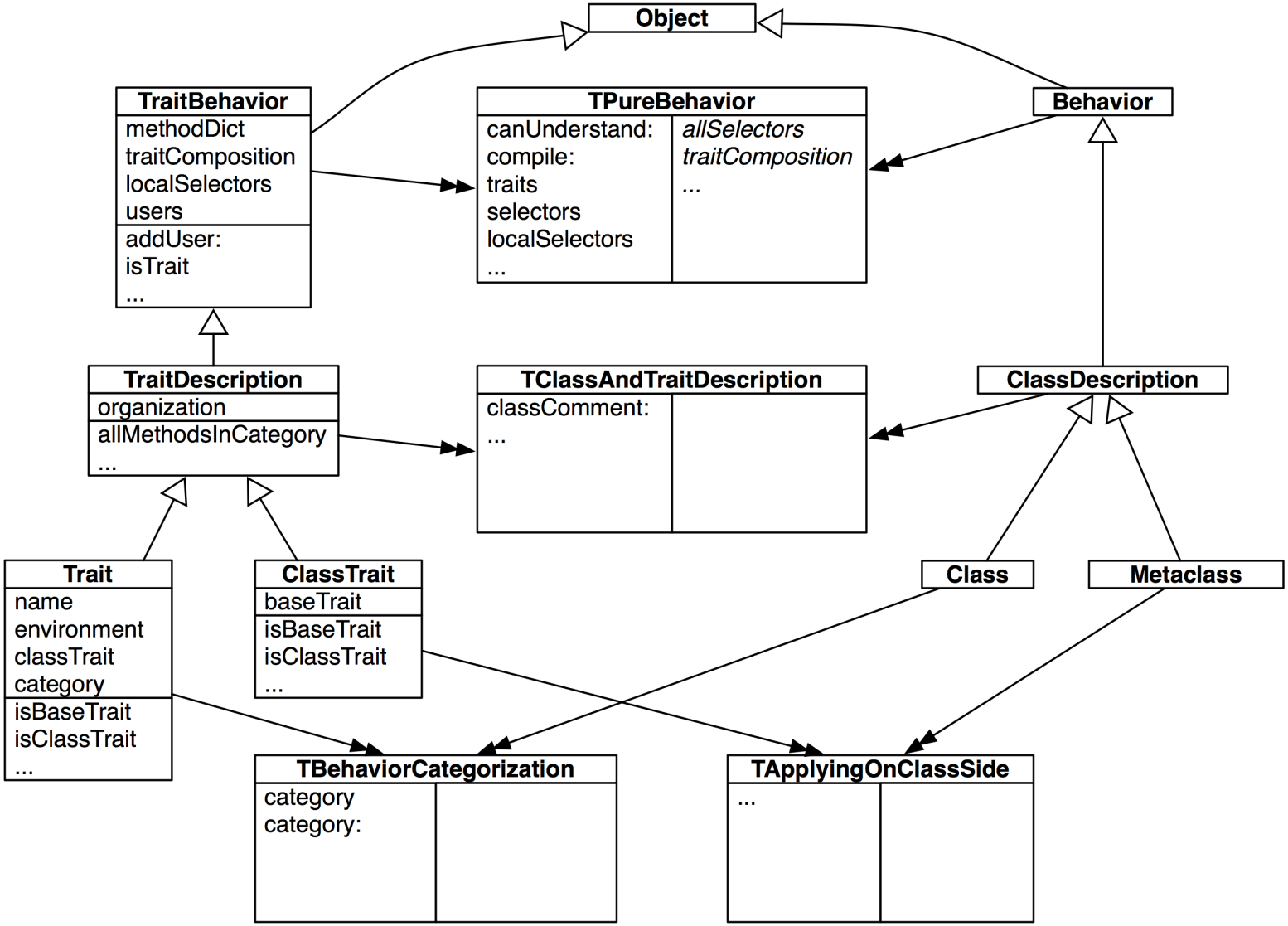
OmniBrowser supports trait browsing and navigation



Traits can be manipulated from the browser



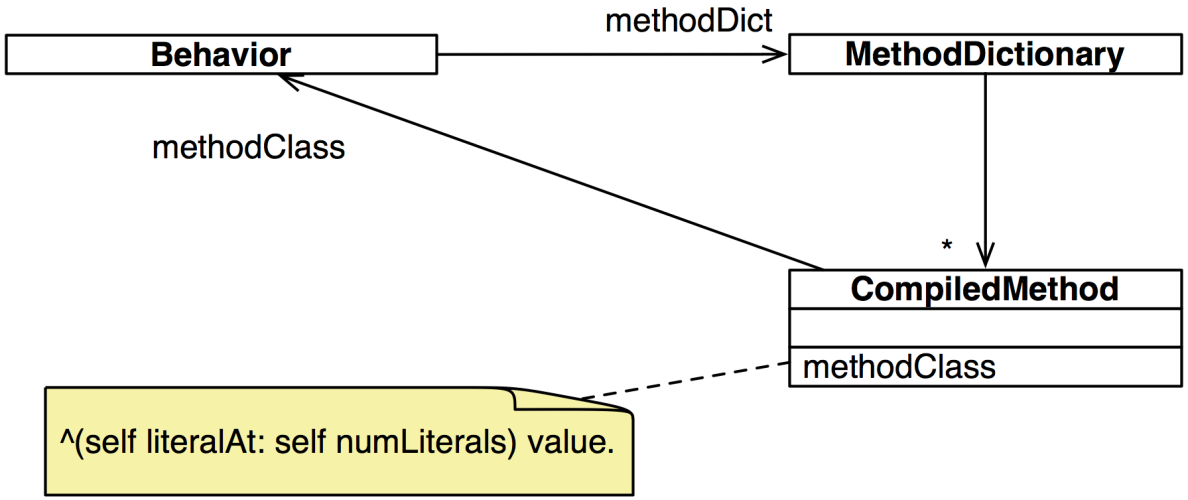
Traits and Classes share common behaviour



Can classes share compiled methods from traits?

Two problems:

1. super is statically bound
2. compiled methods know their class



⇒methods are *copied* to method dictionaries when they are installed

Roadmap

- Why traits?
- Traits in a Nutshell
- Case study — Streams
- Traits in Pharo
- **Future of Traits**

The future of Traits

- Stateful traits
  - some experimental solutions …
- Tool support
  - limited browser support in Pharo
- Automatic refactoring
  - some experiments with formal concept analysis
- Pure trait-based language
  - can traits and classes be unified?
- Traits in other languages
  - Perl, Scala, Fortress, …

License

http://creativecommons.org/licenses/by-sa/3.0/