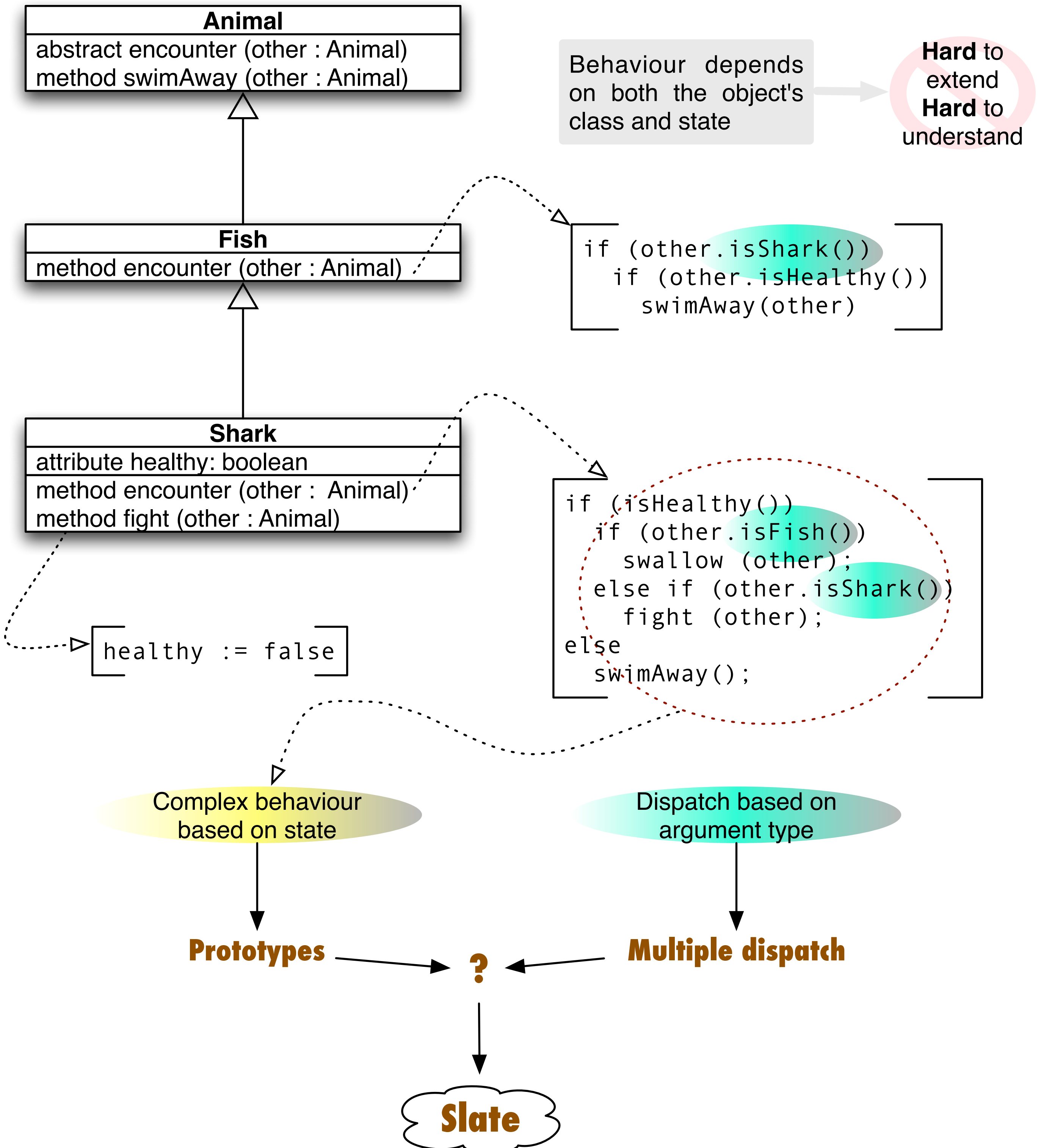
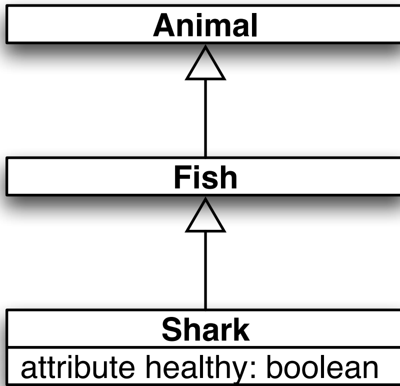
Prototypes with Multiple Dispatch: A motivation

Lee Salzman and Jonathan Aldrich's Paper
Lee's presentation



Multiple dispatch

CLOS Dylan



Methods are split apart

```
method swimAway (a : Animal, other : Animal)

method swallow (animal : Shark, other : Animal)

method fight (animal : Shark, other : Shark)
  animal.healthy := False;

method encounter (animal : Fish, other : Animal)

method encounter (animal : Fish, other : Shark)
  if other.healthy then
   swimAway (animal, other)

method encounter (animal : Fish, other : Shark)
  if other.healthy then
   swimAway (animal, other)

method encounter (animal : Shark, other : Fish)
  if animal.healthy then
   swallow (animal, other)
  else
   swimAway (animal, other)

method encounter (animal : Shark, other : Shark)
  if animal.healthy then
   fight (animal, other)
  else
   swimAway (animal, other)
```

Methods are selected taking into account **all** argument types.

Cleaner implementation logic
no conditionals on argument type

Easy extension
no modifications to existing code to introduce new subtypes

BUT

Still complex logic associated with state

Prototypes

In **single** dispatch languages:

Use the double dispatch pattern...

CS 535 Object-Oriented Programming
Spring Semester, 2003
Double Dispatch

Contents of Doc 13, Double Dispatch

Double Dispatch

or libraries:

O'REILLY ONLamp.com

Advanced OOP: Multimethods

by David Mertz

Introduction

Dr. Carlo Pescio

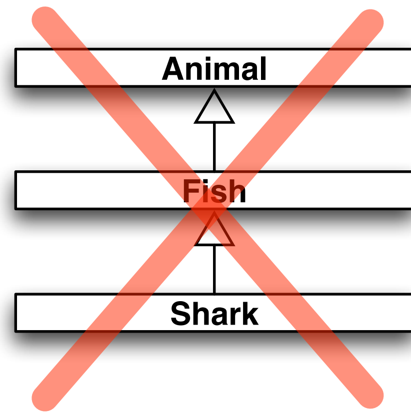
Multiple Dispatch
A new approach using templates and RTTI

Published in C++ Report, June 1998

Why multiple dispatch?

vanilla C++ does **not** multi-dispatch

Prototype-based programming,
the ultimate dynamic



No classes, just objects

```
object Animal
object Fish
object Shark
object HealthyShark
object DyingShark
```

Since we work with concrete instances, state can be dispensed with. There are no variables, only slots.

Objects, created on the fly, inherit behaviour slots by delegation

```
addDelegation (Fish, Animal)
addDelegation (Shark, Animal)
addDelegation (Shark, HealthyShark)
```

Methods are created and attached also on the fly

```
method Animal.swimAway () { ... }
method HealthyShark.swallow () {
...
}
```

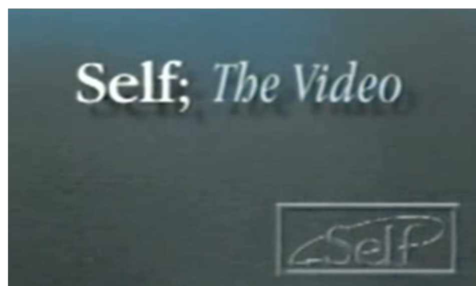
```
method HealthyShark.fight (other : Shark) {
  removeDelegation(HealthyShark);
  addDelegation(DyingShark);
}
```

Dynamic changes in delegation relations model variations in both **state** and **behaviour**.

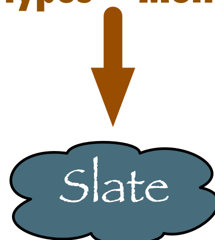
```
method Fish.encounter(other) {
  if (other.isA(HealthyShark))
  swimAway();
}
method HealthyShark.encounter (other) {
  if (other.isFish())
    swallow (other)
  else if (other.isShark())
    fight (other)
}
method DyingShark.encounter (other) {
  swimAway();
}
```

But we re-encounter the complexity of type-based branching.

Implementations:



Self tutorial



Other languages...

Prototypes + Multimethods

Slate