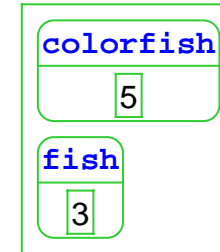


Object Implementation Overview

- **Inheritance:** superclass chain for fields and methods, part chain for objects
- **Overriding:** method dispatch uses object tag
- **Super calls:** `%super` hidden variable contains superclass name

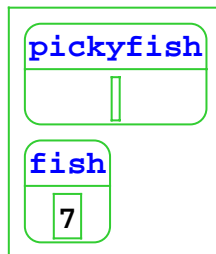
Object Representation

- An object = a list of *parts*
 - from instantiated class up to base class



Object Representation

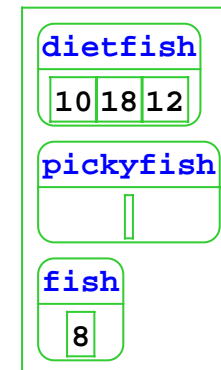
- An object = a list of *parts*
 - from instantiated class up to base class



Object Representation

- An object = a list of *parts*
 - from instantiated class up to base class

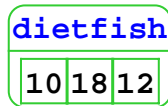
```
class dietfish
  extends pickyfish
  field carbos
  field sodium
  field cholestrol
  ...
```



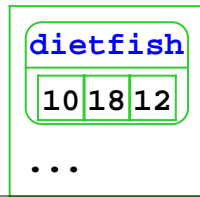
- Use part vectors in environments

Object Representation

```
(define-datatype part part?
  (a-part
   (class-name symbol?)
   (fields vector?)))
```

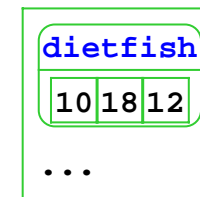


;; An object is a list of parts



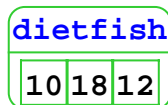
Object Representation

```
;; new-object : sym -> object
(define (new-object cls-name)
  (if (eq? cls-name 'object)
      '()
      (let ([c-decl (lookup-class cls-name)])
        (cons
         (make-first-part c-decl)
         (new-object (class-decl->super-name
                     c-decl)))))))
```



Object Representation

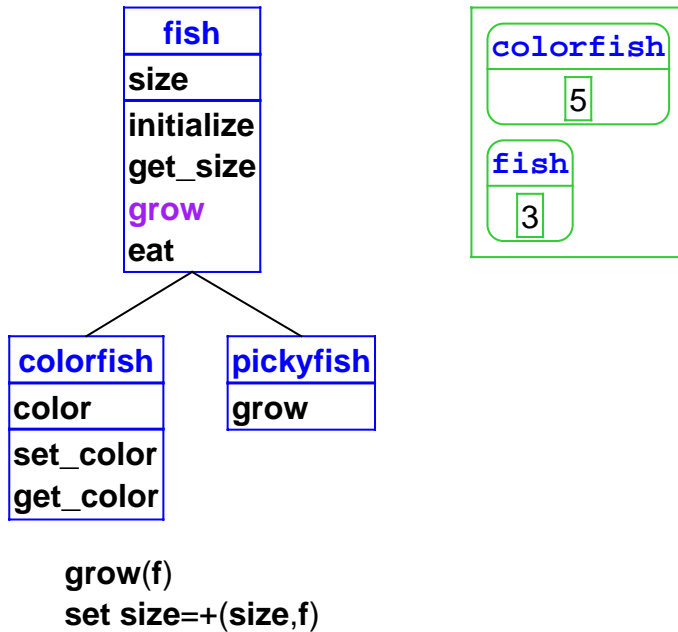
```
;; make-first-part : class-decl -> part
(define (make-first-part c-decl)
  (a-part
   (class-decl->class-name c-decl)
   (make-vector
    (length (class-decl->field-ids
              c-decl)))))
```



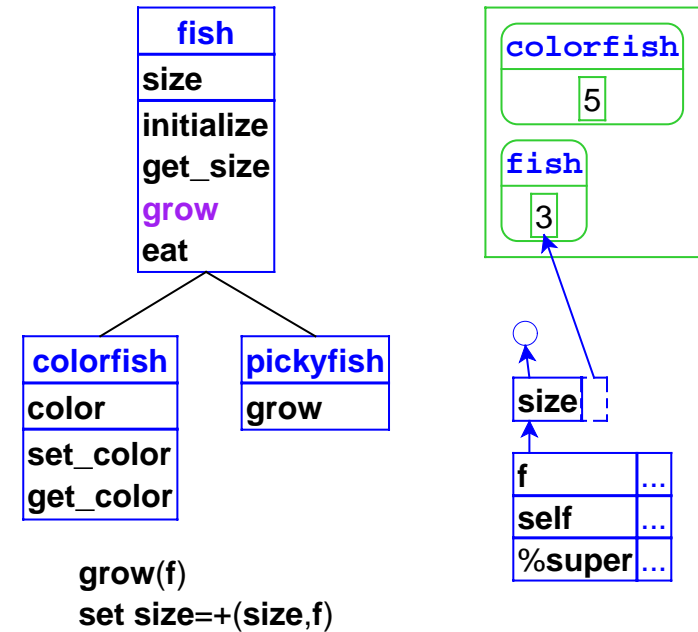
Method Search

```
(define find-method-and-apply
  (lambda (m-name host-name self args)
    (if (eq? host-name 'object)
        (eopl:error ...) ; not found
        (let ([m-decl
                (lookup-method-decl
                 m-name
                 (class-name->method-decls
                  host-name))])
          (if (method-decl? m-decl)
              (apply-method m-decl host-name
                             self args)
              (find-method-and-apply m-name
                                     (class-name->super-name
                                      host-name)
                                     self args))))))
```

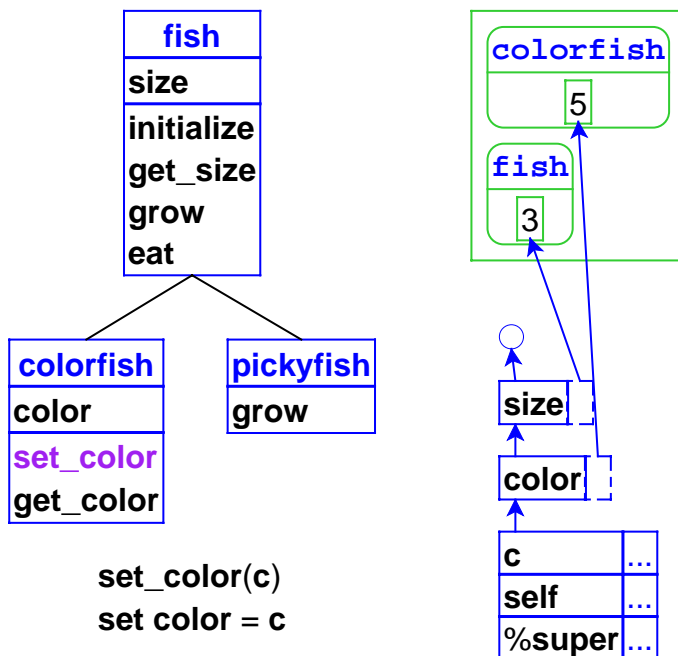
Method Application



Method Application



Method Application



Method Application

```
;; apply-method : method-decl sym object
;;               lstof-expval -> expval
(define apply-method
  (lambda (m-decl host-name self args)
    (let ([ids (method-decl->ids m-decl)]
          [body (method-decl->body m-decl)]
          [super-name
           (class-name->super-name host-name)])
      (eval-expression
        body
        (extend-env
          (cons '%super (cons 'self ids))
          (cons super-name (cons self args))
          (build-field-env
            (view-object-as self
                          host-name)))))))
```

Method Application

```
;; view-object-as : object sym -> lstof-part
(define (view-object-as parts class-name)
  (if (eq? (part->class-name (car parts))
        class-name)
      parts
      (view-object-as (cdr parts) class-name)))

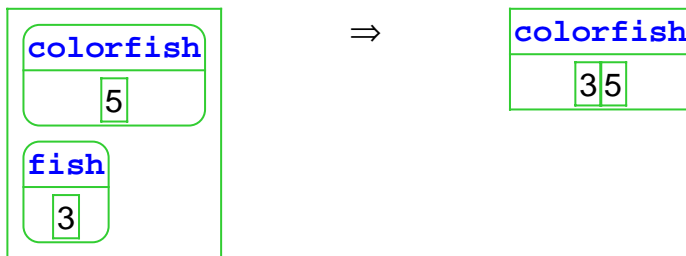
;; build-field-env : lstof-parts -> env
(define (build-field-env parts)
  (if (null? parts)
      (empty-env)
      (extend-env-refs
       (part->field-ids (car parts))
       (part->fields (car parts))
       (build-field-env (cdr parts)))))
```

Complete Implementation

(implement in DrScheme)

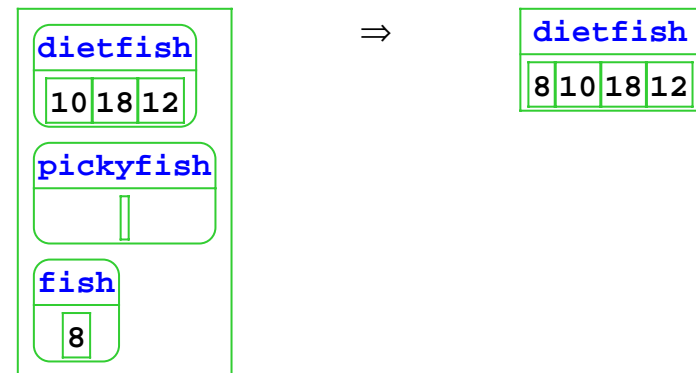
A More Realistic Object Representation

- A chain of parts wastes space
- Collapse vectors into one



A More Realistic Object Representation

- A chain of parts wastes space
- Collapse vectors into one



A More Realistic Object Representation

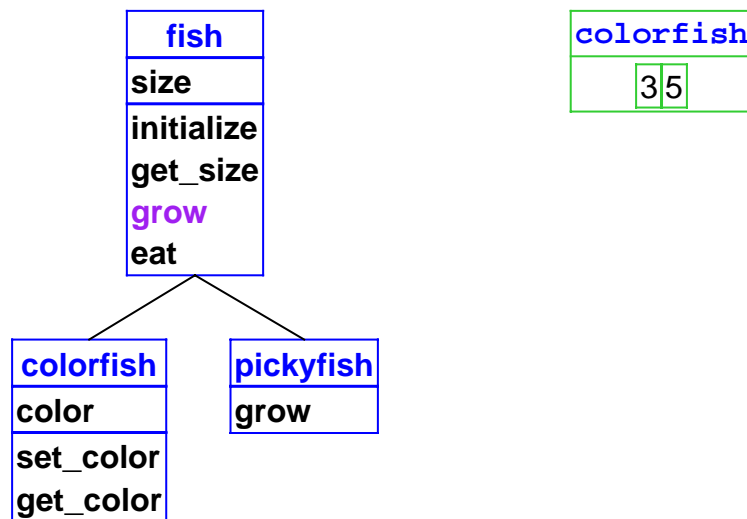
```
(define-datatype object object?
  (an-object
   (class-name symbol?)
   (fields vector?)))

;; new-object : sym -> object
(define (new-object class-name)
  (an-object
   class-name
   (make-vector
    (roll-up-field-length class-name))))
```

A More Realistic Object Representation

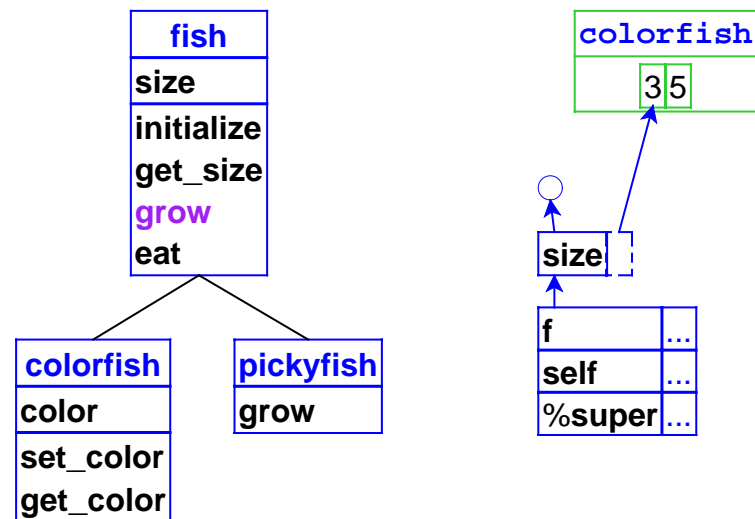
```
;; roll-up-field-length : sym -> num
(define roll-up-field-length
  (lambda (class-name)
    (if (eqv? class-name 'object)
        0
        (+ (roll-up-field-length
            (class-name->super-name
             class-name))
           (length
            (class-name->field-ids
             class-name))))))
```

Method Application with Flat Objects



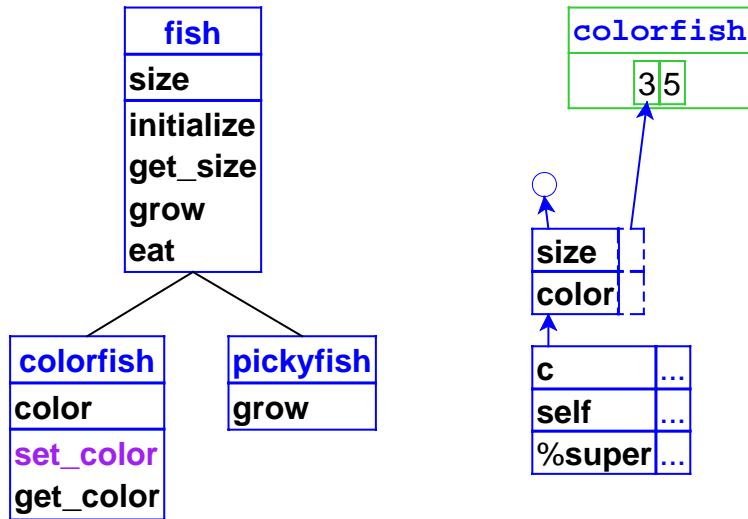
```
grow(f)
set size=+(size,f)
```

Method Application with Flat Objects



```
grow(f)
set size=+(size,f)
```

Method Application with Flat Objects



`set_color(c)`
`set color = c`

Method Application with Flat Objects

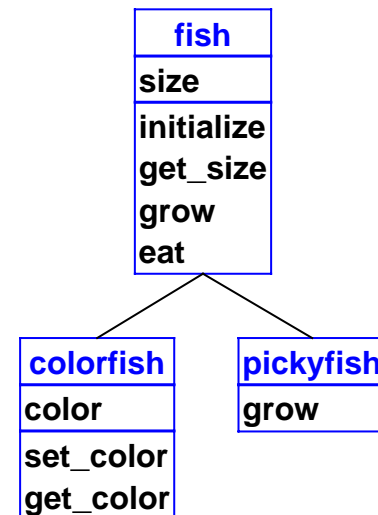
```
(define apply-method
  (lambda (m-decl host-name self args)
    (let ([ids (method-decl->ids m-decl)]
          [body (method-decl->body m-decl)]
          [super-name (class-name->super-name
                        host-name)]
          [field-ids (roll-up-field-ids
                      host-name)]
          [fields (object->fields self)])
      (eval-expression
        body
        (extend-env
          (cons '%super (cons 'self ids))
          (cons super-name (cons self args))
          (extend-env-refs field-ids fields
                           (empty-env)))))))
```

Complete Implementation of Flat Objects

(implement in DrScheme)

A More Realistic Class Representation

Eliminate tree walks: object creation, method calls



Object Creation without Tree Walks

Current interpreter:

1. Find class
2. Get field list (walk tree)
3. Allocate field array and object

To eliminate tree walks:

2. Extract flat field list from class

Method Calls without Tree Walks

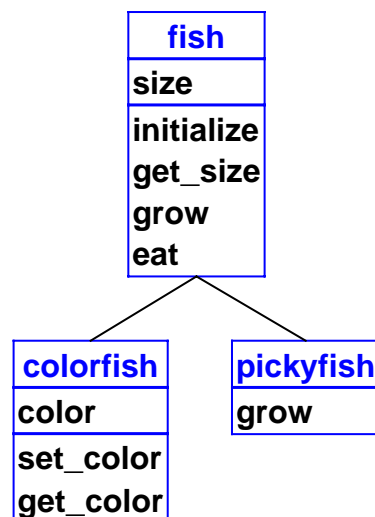
After object and arguments are determined:

1. Lookup object class
2. Find class containing method (walk tree)
3. Get field variables for class (walk tree)
4. Create environment: fields + %**super** + **self** + args
5. Evaluate method body

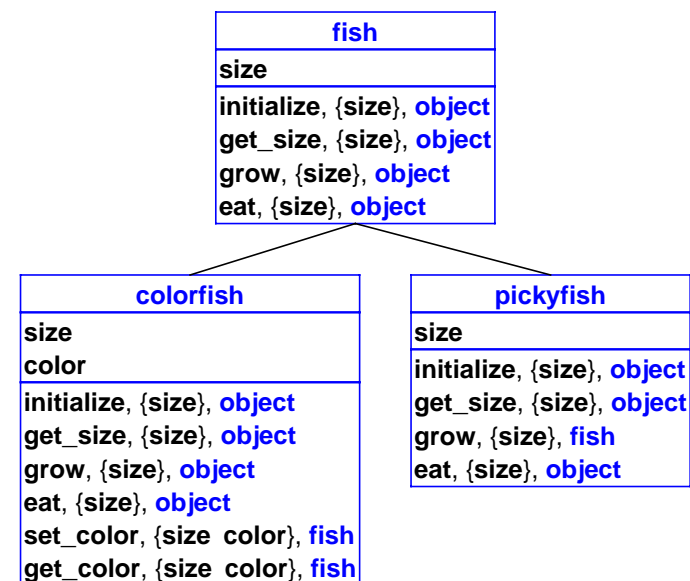
To eliminate tree walks:

- 2 & 3. Find method in current class, extract field list and superclass name

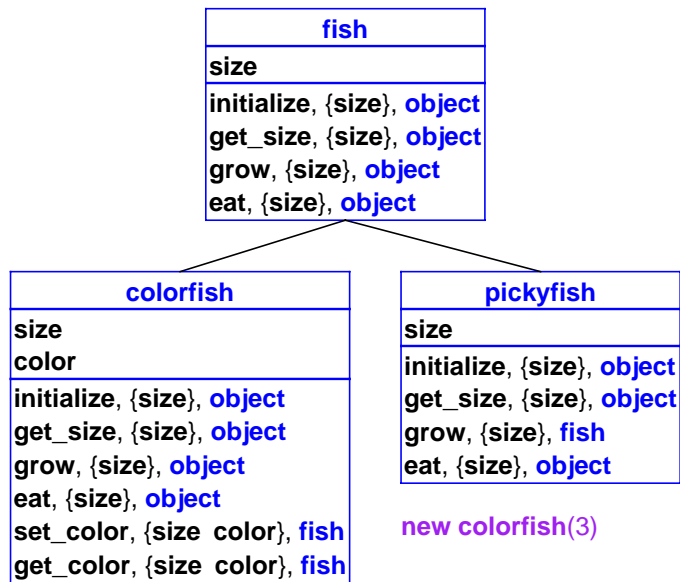
Class Elaboration



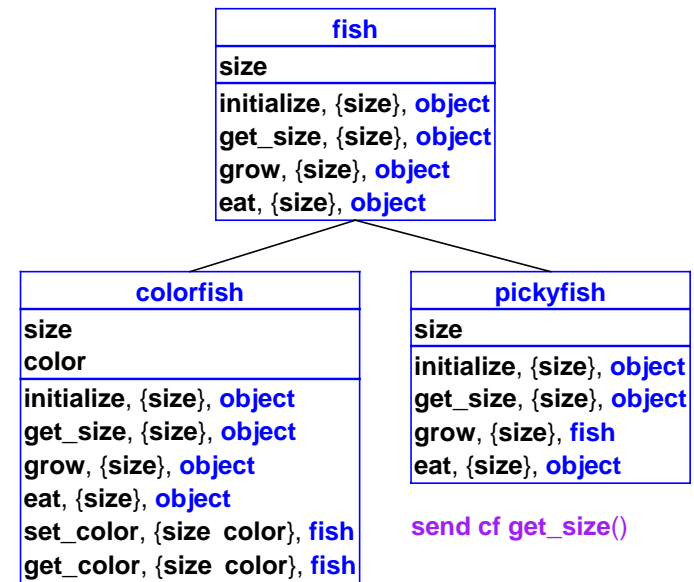
Class Elaboration



Class Elaboration



Class Elaboration



Implementation

- Change `elaborate-class-decls!` to build annotated tree
- Change `new-object` to use class's immediate field list
- Change `apply-method` to work with annotated methods

(implement in DrScheme)