CSC324: Principles of Programming Languages

Lecture 0xA

The Frankenstein of a quiz...

Problems, and how we will address them

- The length of the quiz was too long for a 50 minute session
  - Grading for **Quiz 1** will take this into account (specifics TBD)
  - The design of **Quiz 2** will take this into account

- Information about the quiz & materials was hard to find
  - Let's move the course page onto **Quercus**. (I'll be very happy to stop hand-writing course page HTML.)

- Some say they are struggling with the practicalities of programming in Haskell and Racket
  - More to come on this…
  - ...starting with a worked example https://www.youtube.com/watch?v=ERxzfwxqXuM (38min)
    - Tell me if this is useful, or a waste of time

Some notes on Ex4...

… the average was lower than past exercises, so I think it'd be good to talk about some ways to improve your designs, as your programs become more involved

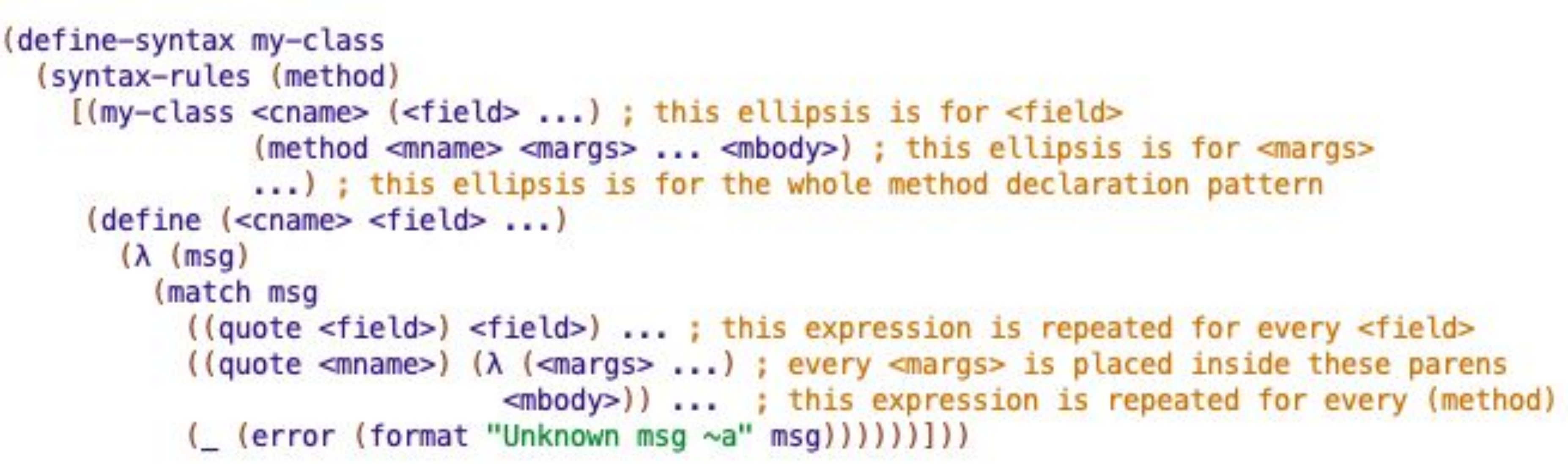
Onward!

Last time...

- We saw how a lexically-scoped closure could be used to implement a **message passing-based object system**
- We implemented a simple object system that uses the `(cond [...])` form to dispatch functionality

- We saw how an object needs a reference to itself in order to
  - make field accesses look more consistent, syntactically
  - make method calls within a method itself possible

Where we left off



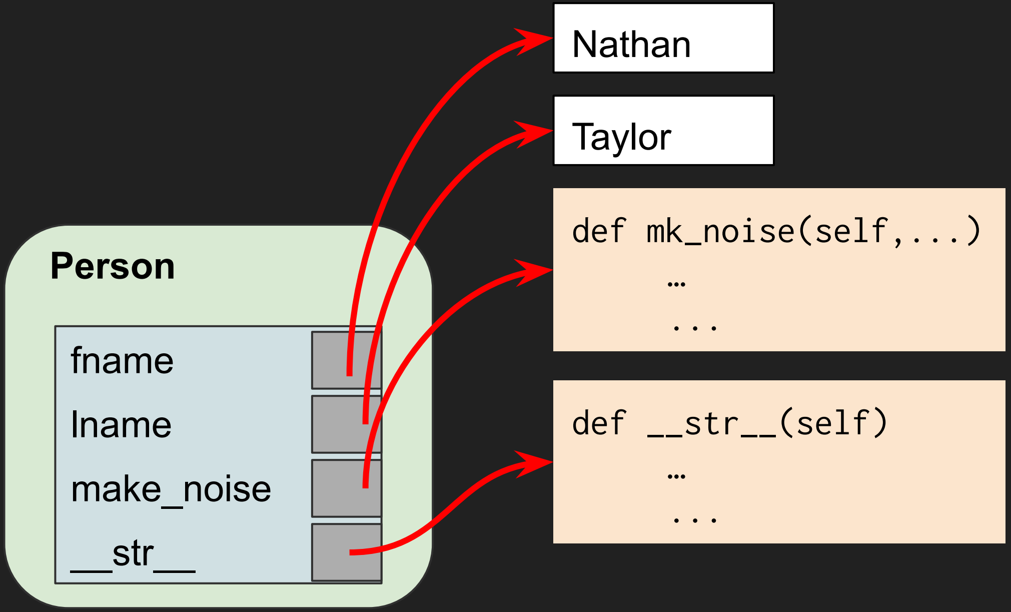
```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...) ; this ellipsis is for <field>
               (method <mname> <margs> ... <mbody>) ; this ellipsis is for <margs>
               ...) ; this ellipsis is for the whole method declaration pattern
     (define (<cname> <field> ...)
       (λ (msg)
         (match msg
           ((quote <field>) <field>) ... ; this expression is repeated for every <field>
           ((quote <mname>) (λ (<margs> ...) ; every <margs> is placed inside these parens
                               <mbody>)) ...  ; this expression is repeated for every (method)
           (_ (error (format "Unknown msg ~a" msg))))))]))
```

Method and field lookup via a dictionary

Recall that a vtable-based object is:

- A structure containing the object's fields
- A function pointer table containing the object's methods

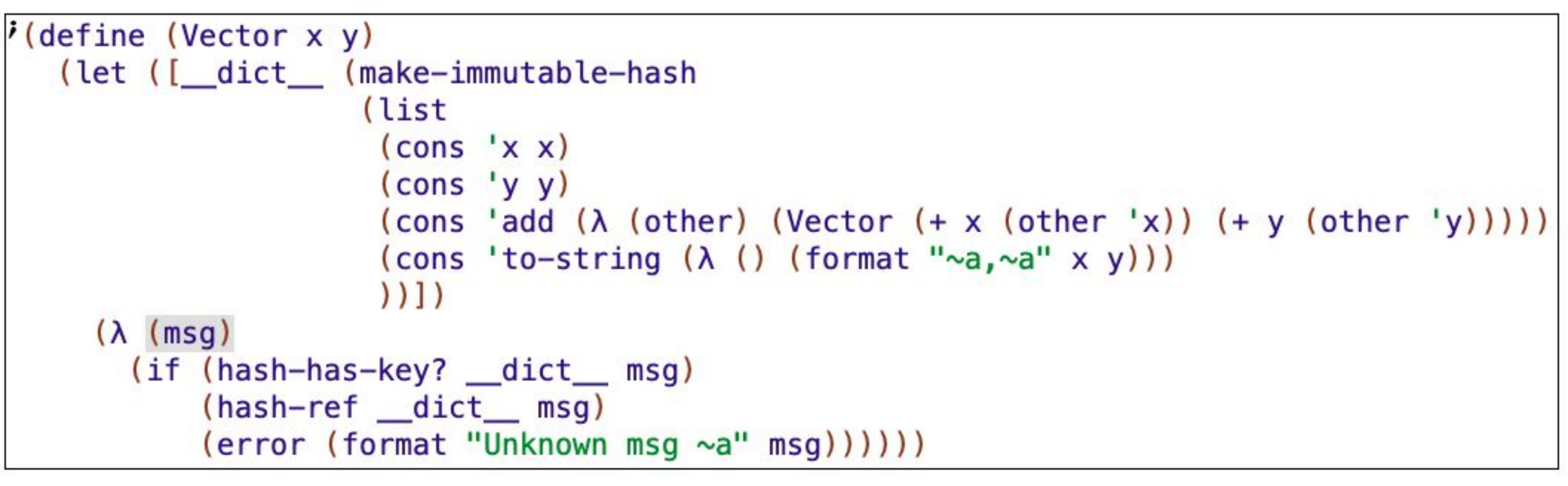
Such a vtable could be implemented by a **dictionary** data structure, mapping field names to values and method names to functions



```
def mk_noise(self,...)
    …
     ...
```

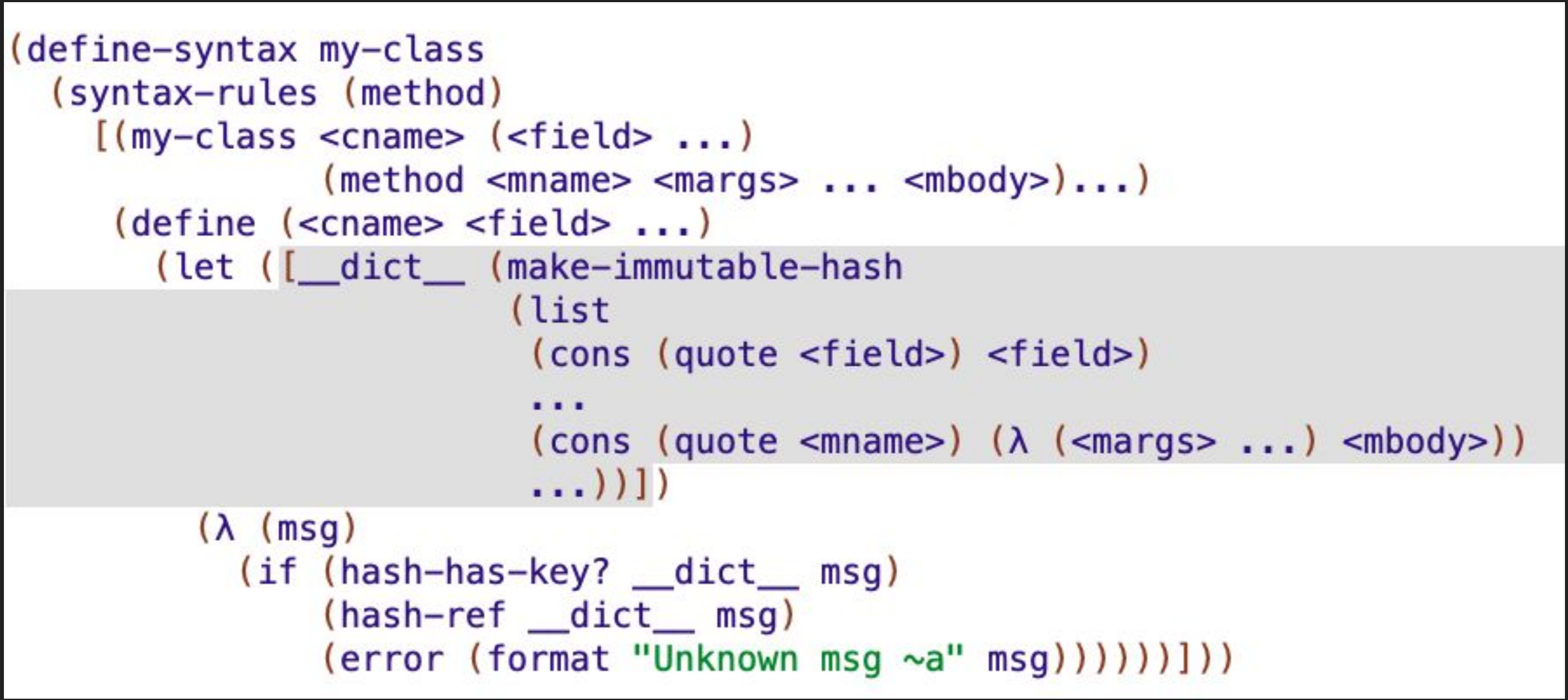
```
def __str__(self)
    …
     ...
```

Implementing dictionary-based table lookup



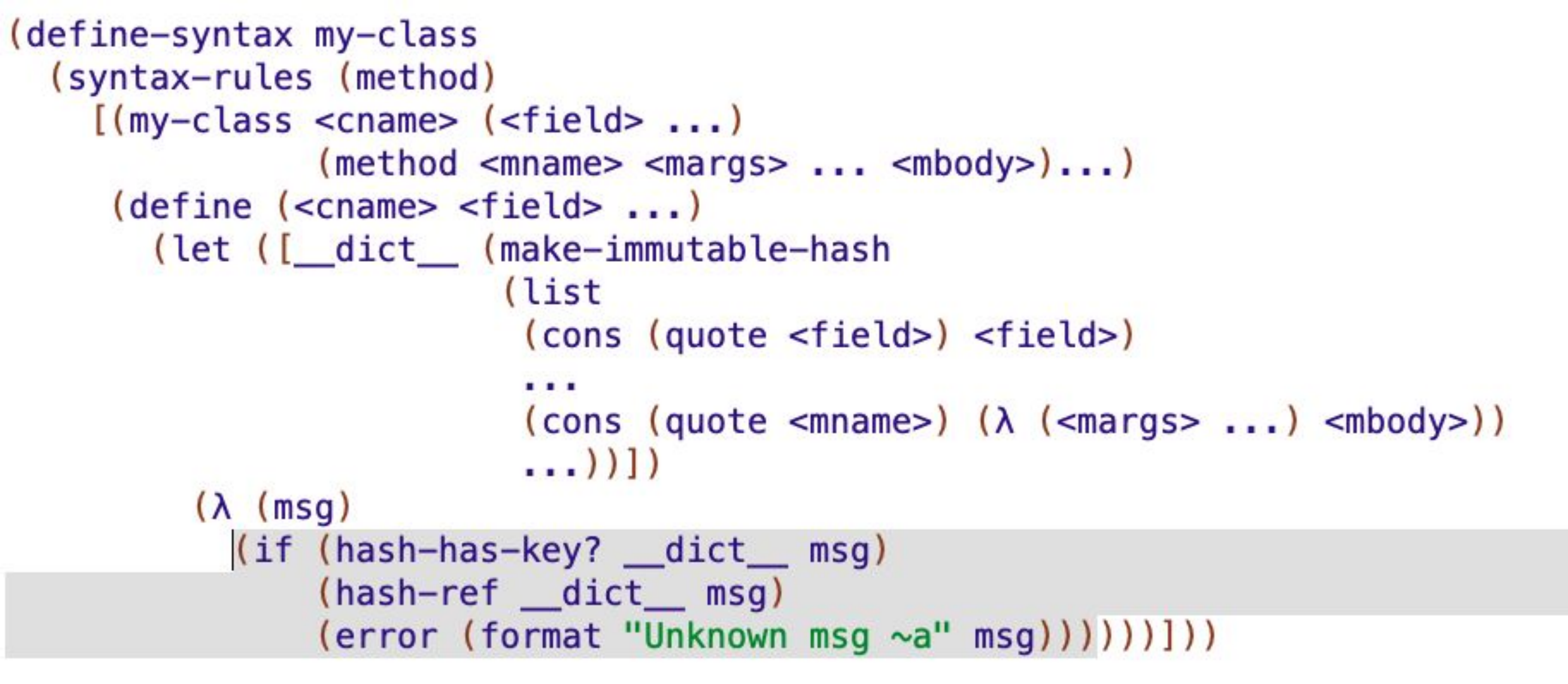
```
;(define (Vector x y)
  (let ([__dict__ (make-immutable-hash
                   (list
                    (cons 'x x)
                    (cons 'y y)
                    (cons 'add (λ (other) (Vector (+ x (other 'x)) (+ y (other 'y)))))
                    (cons 'to-string (λ () (format "~a,~a" x y)))
                    ))])
    (λ (msg)
      (if (hash-has-key? __dict__ msg)
          (hash-ref __dict__ msg)
          (error (format "Unknown msg ~a" msg))))))
```

Implementing dictionary-based table lookup



```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (let ([__dict__ (make-immutable-hash
                        (list
                         (cons (quote <field>) <field>)
                         ...
                         (cons (quote <mname>) (λ (<margs> ...) <mbody>))
                         ...))])
         (λ (msg)
           (if (hash-has-key? __dict__ msg)
               (hash-ref __dict__ msg)
               (error (format "Unknown msg ~a" msg))))))]))
```

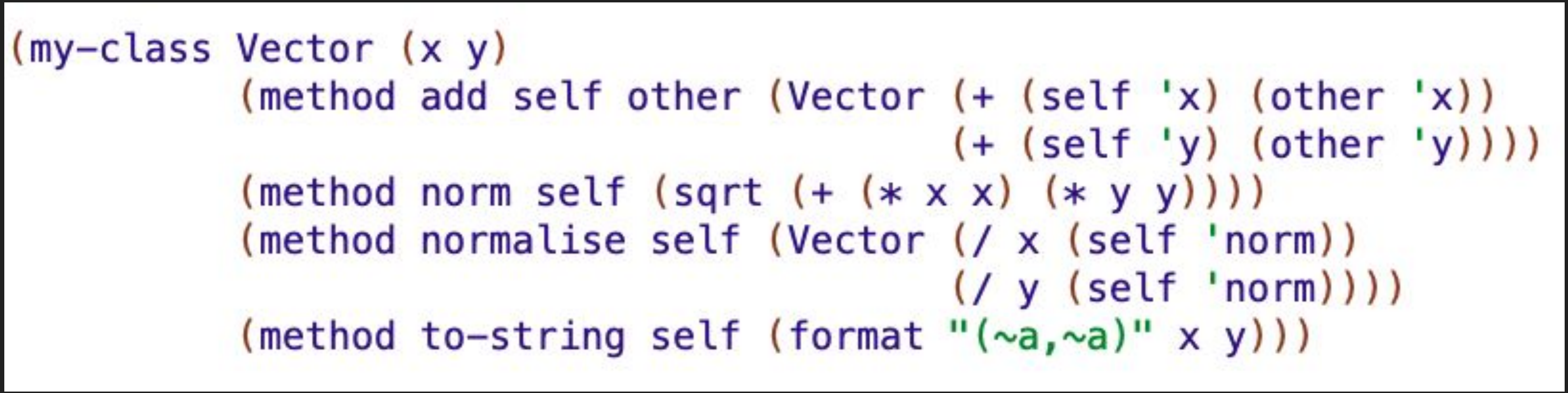
Implementing dictionary-based table lookup



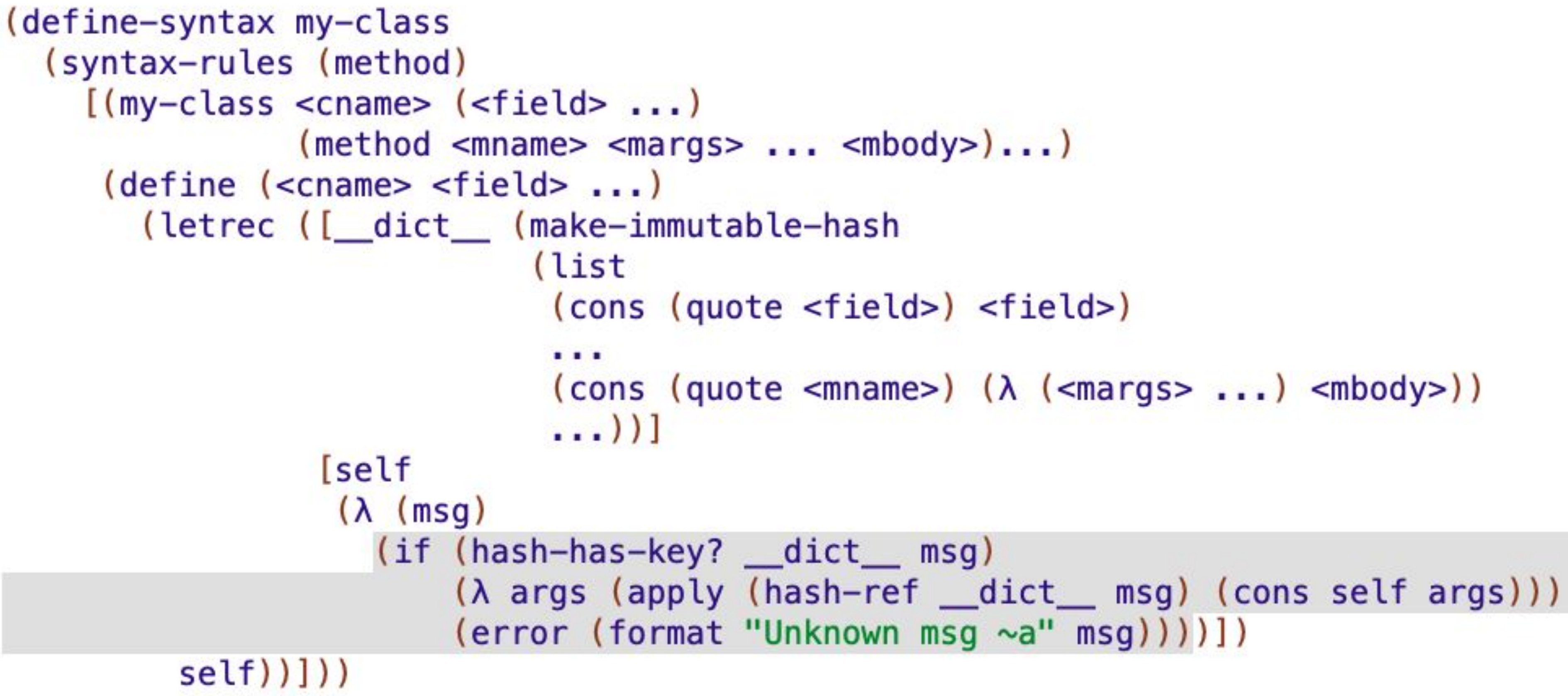
```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (let ([__dict__ (make-immutable-hash
                        (list
                         (cons (quote <field>) <field>)
                         ...
                         (cons (quote <mname>) (λ (<margs> ...) <mbody>))
                         ...))])
         (λ (msg)
           (if (hash-has-key? __dict__ msg)
               (hash-ref __dict__ msg)
               (error (format "Unknown msg ~a" msg))))))]))
```

Implementing self

Does a dictionary-based lookup get us any closer to being able to implement self?

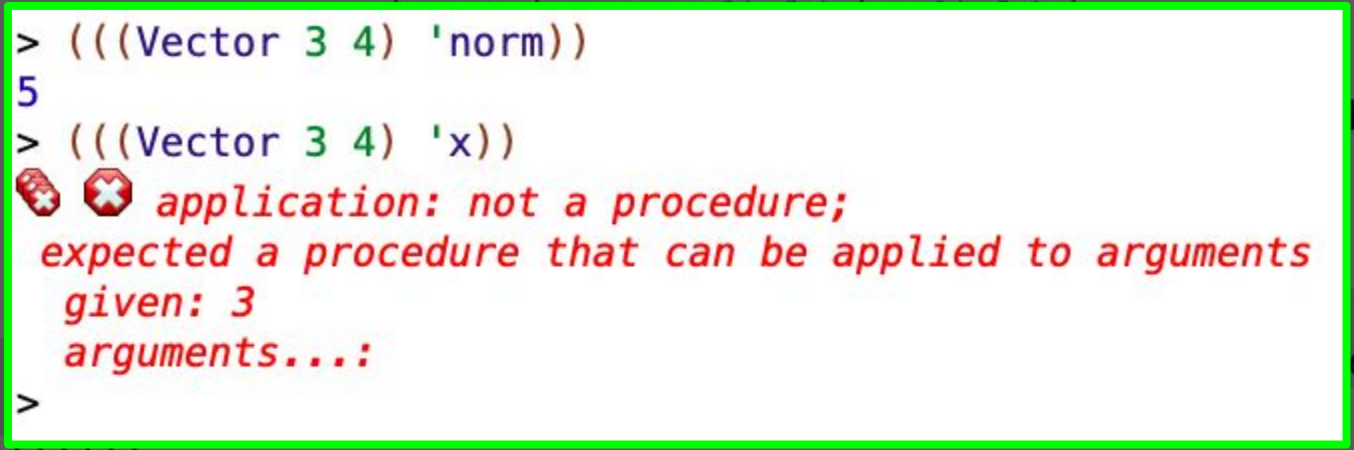


```
(my-class Vector (x y)
          (method add self other (Vector (+ (self 'x) (other 'x))
                                         (+ (self 'y) (other 'y))))
          (method norm self (sqrt (+ (* x x) (* y y))))
          (method normalise self (Vector (/ x (self 'norm))
                                         (/ y (self 'norm))))
          (method to-string self (format "(~a,~a)" x y)))
```



```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (letrec ([__dict__ (make-immutable-hash
                           (list
                            (cons (quote <field>) <field>)
                            ...
                            (cons (quote <mname>) (λ (<margs> ...) <mbody>))
                            ...))]
                [self
                 (λ (msg)
                   (if (hash-has-key? __dict__ msg)
                       (λ args (apply (hash-ref __dict__ msg) (cons self args)))
                       (error (format "Unknown msg ~a" msg))))])
         self))]))
```

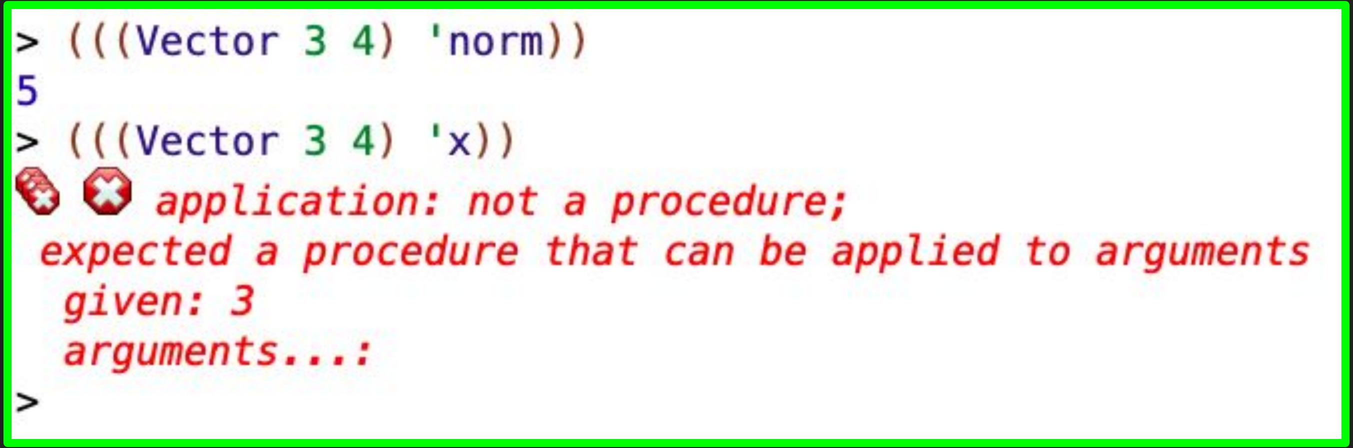
```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (letrec ([__dict__ (make-immutable-hash
                           (list
                                                                        body>))
                                                                           elf args)))
         self))]))
```



```
> (((Vector 3 4) 'norm))
5
> (((Vector 3 4) 'x))
application: not a procedure;
 expected a procedure that can be applied to arguments
  given: 3
  arguments...:
>
```

Two steps forward, one step back...

We got "self" working with methods, but we now broke field accesses

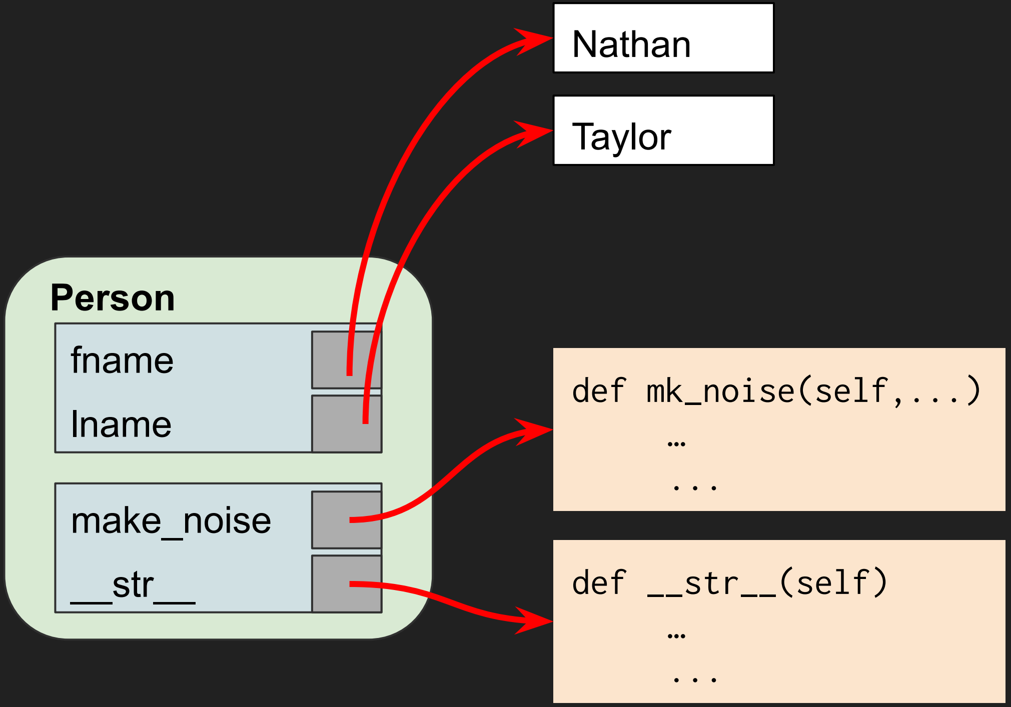


```
> (((Vector 3 4) 'norm))
5
> (((Vector 3 4) 'x))
application: not a procedure;
 expected a procedure that can be applied to arguments
  given: 3
  arguments...:
>
```

Method and field lookup via two dictionaries

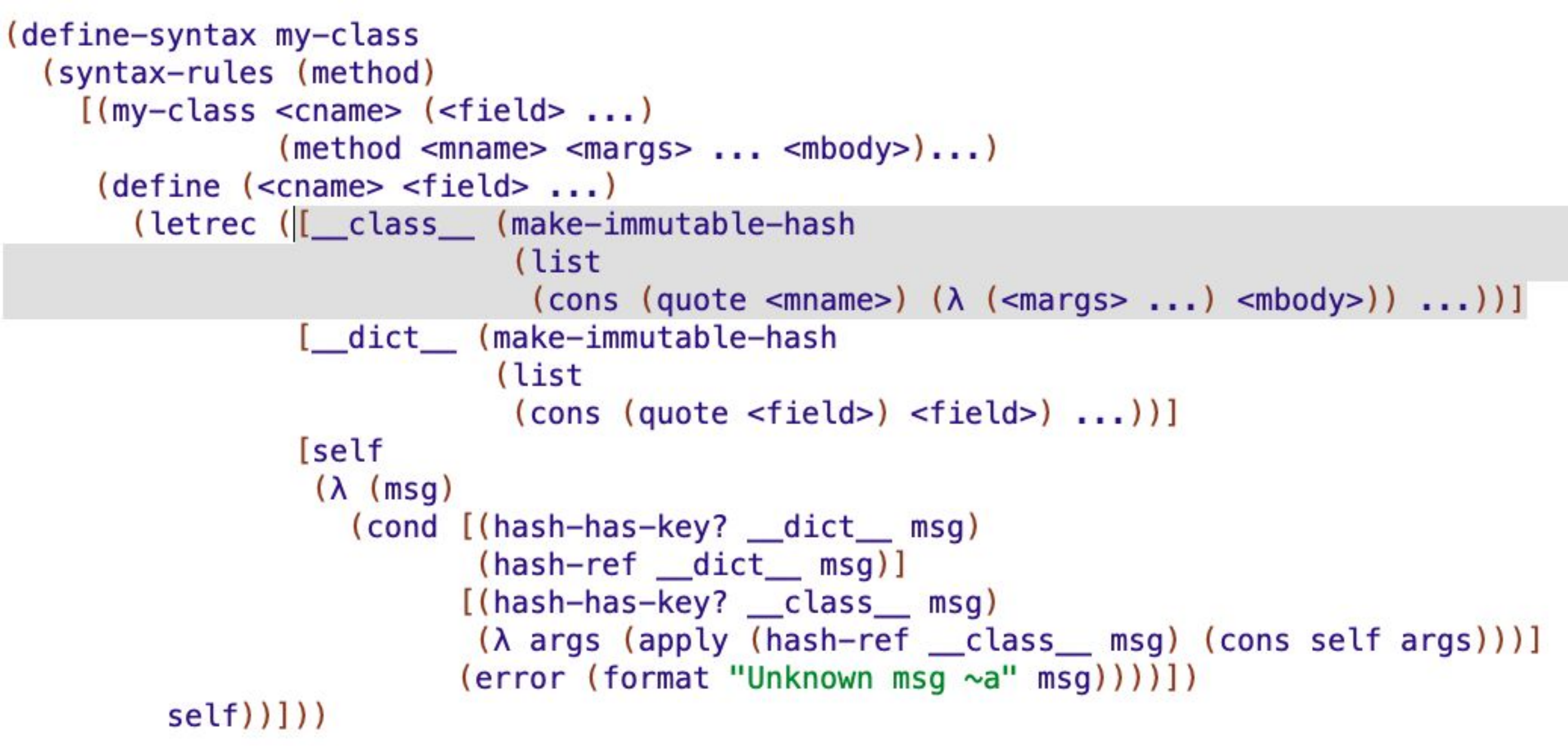
- __dict__ to hold instance fields
- __class__ to hold methods

- instance field lookups will return the field as we previously did
- class method lookups will return the "fix-first self" patched lambda expression as we just did

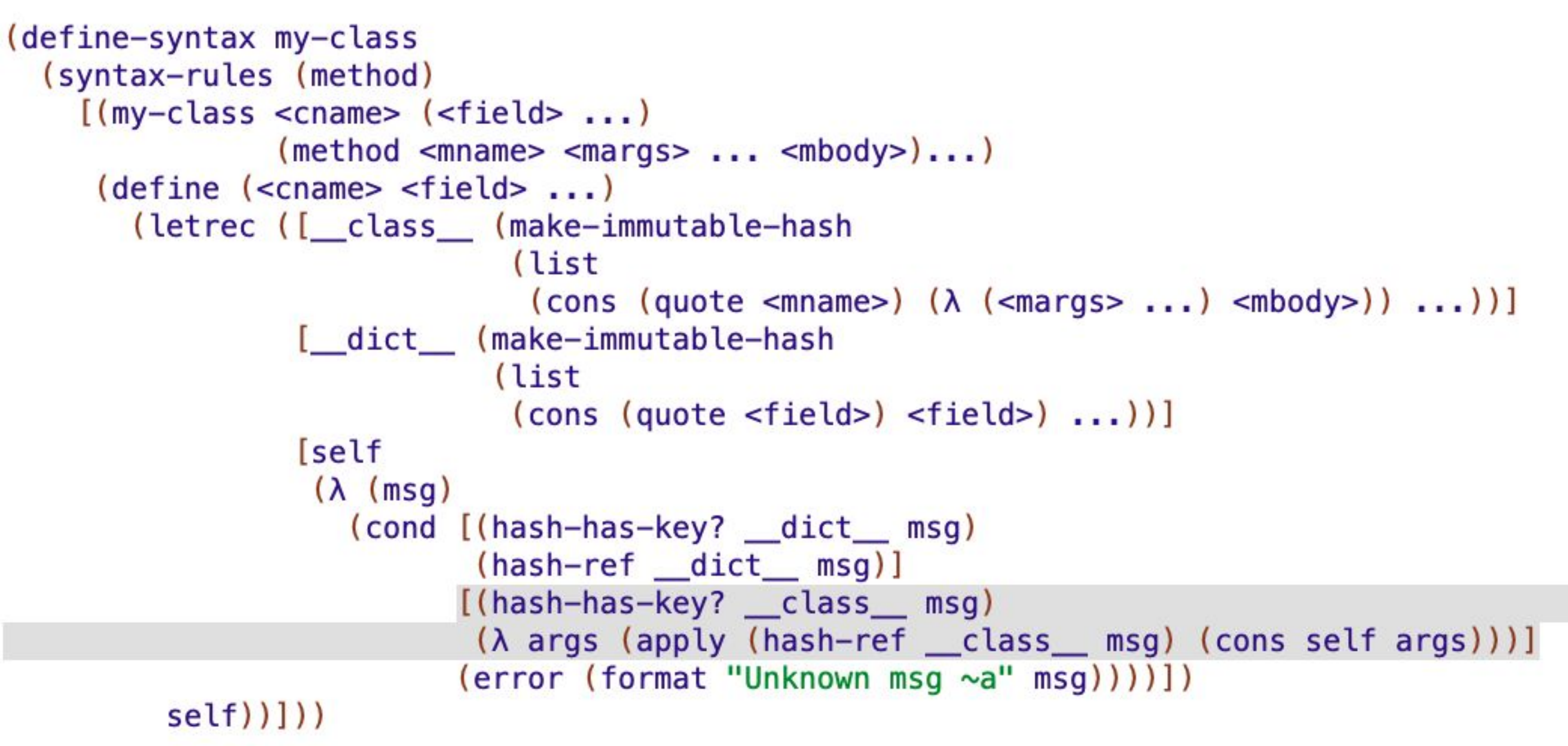


```
def mk_noise(self,...)
      …
      ...
```

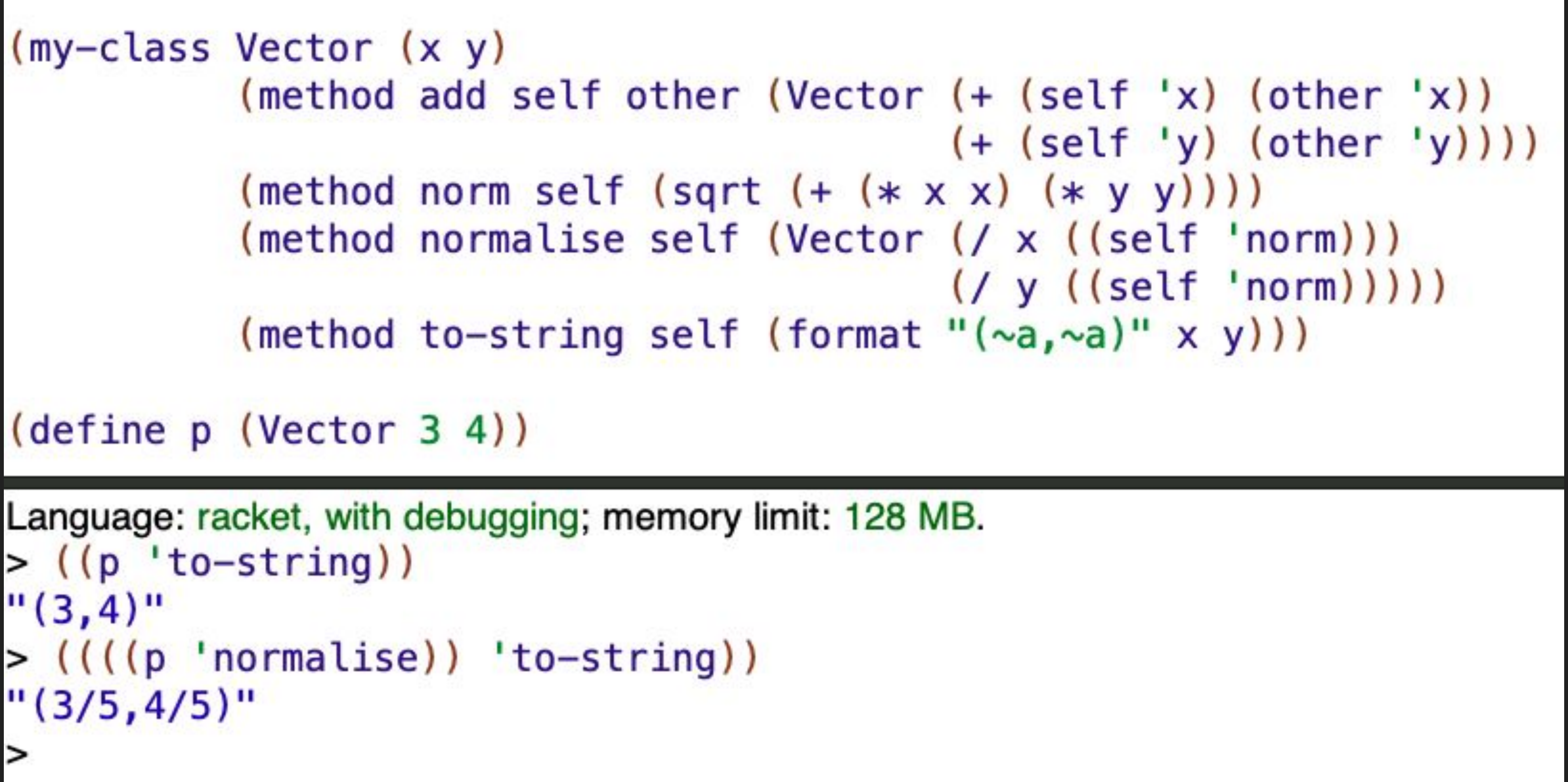
```
def __str__(self)
      …
      ...
```



```racket
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (letrec ([__class__ (make-immutable-hash
                            (list
                             (cons (quote <mname>) (λ (<margs> ...) <mbody>)) ...))]
                [__dict__ (make-immutable-hash
                           (list
                            (cons (quote <field>) <field>) ...))]
                [self
                 (λ (msg)
                   (cond [(hash-has-key? __dict__ msg)
                          (hash-ref __dict__ msg)]
                         [(hash-has-key? __class__ msg)
                          (λ args (apply (hash-ref __class__ msg) (cons self args)))]
                         (error (format "Unknown msg ~a" msg))))])
         self))]))
```



```
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <cname> (<field> ...)
               (method <mname> <margs> ... <mbody>)...)
     (define (<cname> <field> ...)
       (letrec ([__class__ (make-immutable-hash
                            (list
                             (cons (quote <mname>) (λ (<margs> ...) <mbody>)) ...))]
                [__dict__ (make-immutable-hash
                           (list
                            (cons (quote <field>) <field>) ...))]
                [self
                 (λ (msg)
                   (cond [(hash-has-key? __dict__ msg)
                          (hash-ref __dict__ msg)]
                         [(hash-has-key? __class__ msg)
                          (λ args (apply (hash-ref __class__ msg) (cons self args)))]
                         (error (format "Unknown msg ~a" msg))))])
         self))]))
```



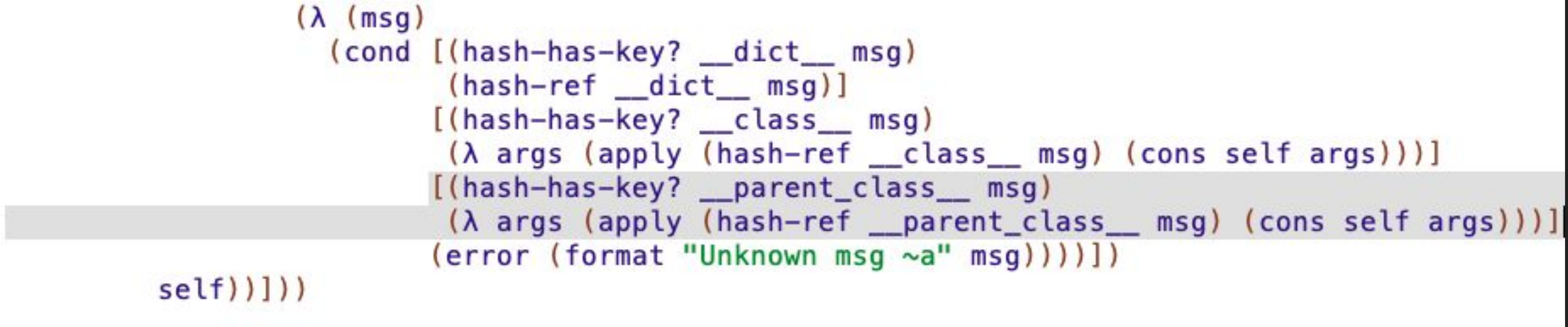
```
(my-class Vector (x y)
          (method add self other (Vector (+ (self 'x) (other 'x))
                                         (+ (self 'y) (other 'y))))
          (method norm self (sqrt (+ (* x x) (* y y))))
          (method normalise self (Vector (/ x ((self 'norm)))
                                         (/ y ((self 'norm)))))
          (method to-string self (format "(~a,~a)" x y)))

(define p (Vector 3 4))
```

```
Language: racket, with debugging; memory limit: 128 MB.
> ((p 'to-string))
"(3,4)"
> ((((p 'normalise)) 'to-string))
"(3/5,4/5)"
>
```

...a first attempt at inheritance…?

What else do you wish your object-oriented language had? You can add it now!



```
        (λ (msg)
          (cond [(hash-has-key? __dict__ msg)
                 (hash-ref __dict__ msg)]
                [(hash-has-key? __class__ msg)
                 (λ args (apply (hash-ref __class__ msg) (cons self args)))]
                [(hash-has-key? __parent_class__ msg)
                 (λ args (apply (hash-ref __parent_class__ msg) (cons self args)))]
                (error (format "Unknown msg ~a" msg))))])
self))]))
```

Next time

We will return to evaluation semantics, and discuss how macros help us cleanly implement lazy data structures in eager languages.