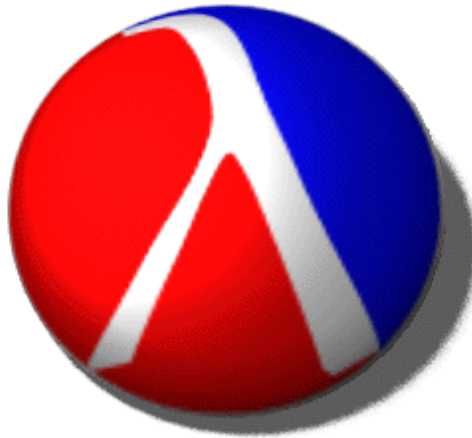


# Composable and Compilable Macros



Matthew Flatt

University of Utah

# A Scheme Programmer...



## ... Programming in Scheme



```
(define (parse file)
  ...)
```

```
-----:--- parse.scm
MzScheme version 103
> (parse "test")
```

```
-----:**- *scheme*
```

## ... in Macro-Extended Scheme!



```
(load "lex+yacc.scm")
(define (parse file)
  (lex [(+ digit) ...]
      ...))

-----:--- parse.scm
MzScheme version 103
> (parse "test")

-----:**- *scheme*
```

## ... Trying to Use a Compiler



```
(load "lex+yacc.scm")
(define (parse file)
  (lex [(+ digit) ...]
      ...))
```

-----:--- parse.scm

```
$ mzc parse.scm
parse.scm: bad syntax
$ █
```

-----:\*\*- \*shell\*

## ... Accomodating the Compiler



```
(eval-when (compile)
  (load "lex+yacc.scm"))
(define (parse file)
  (lex [(+ digit) ...])
-----:--- parse.scm
$ mzc parse.scm
$ █
-----:**- *shell*
```

## ... Trying a Different Compiler



```
(eval-when (compile)
  (load "lex+yacc.scm"))
(define (parse file)
  (lex [(+ digit) ...])
-----:---- parse.scm
$ gsc parse.scm
parse.scm: bad syntax
$ █
-----:**- *shell*
```

## ... Trying a Complex Library

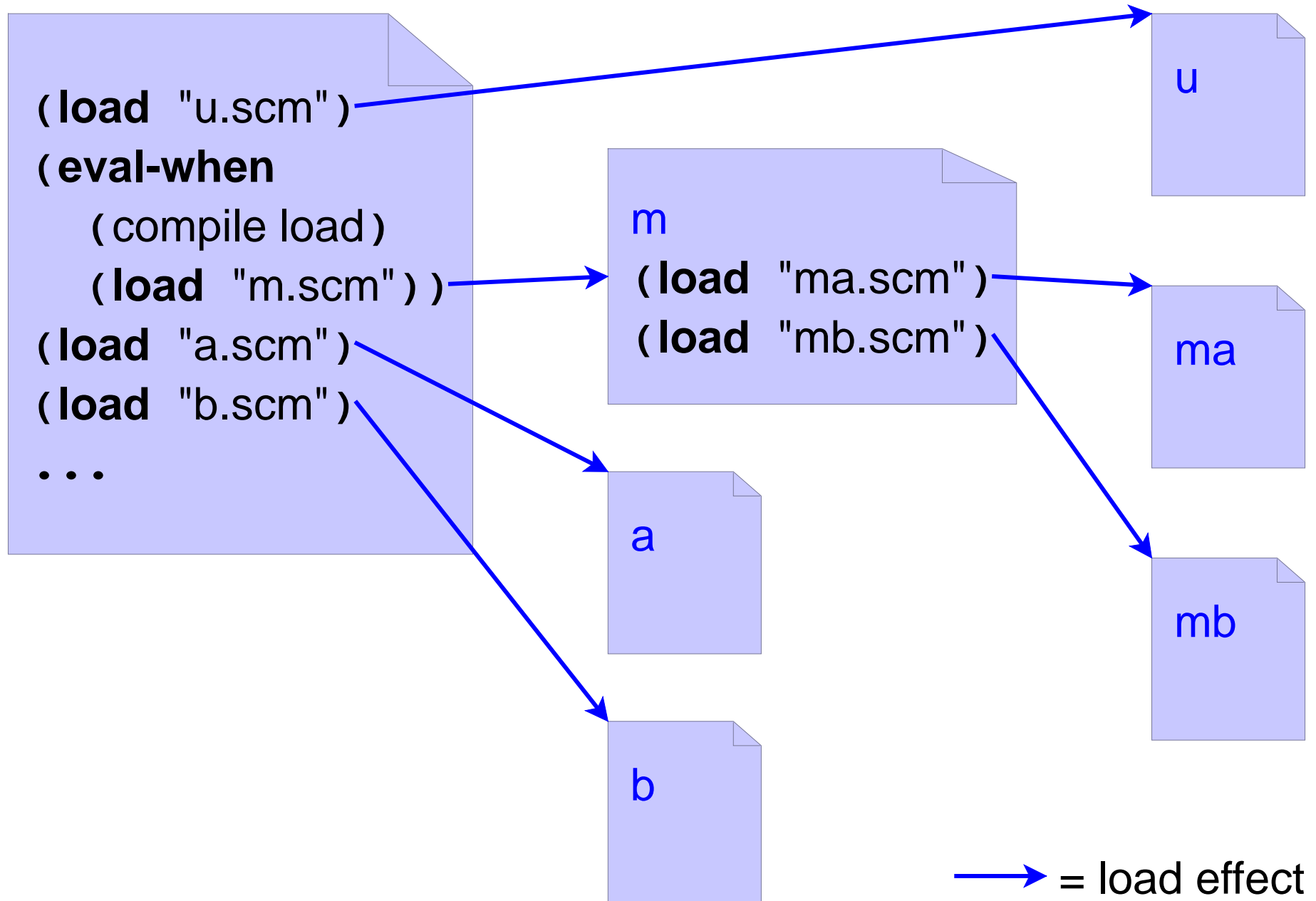


```
(eval-when ...
  (load "parselib.scm"))
(define (parse file)
  ...)

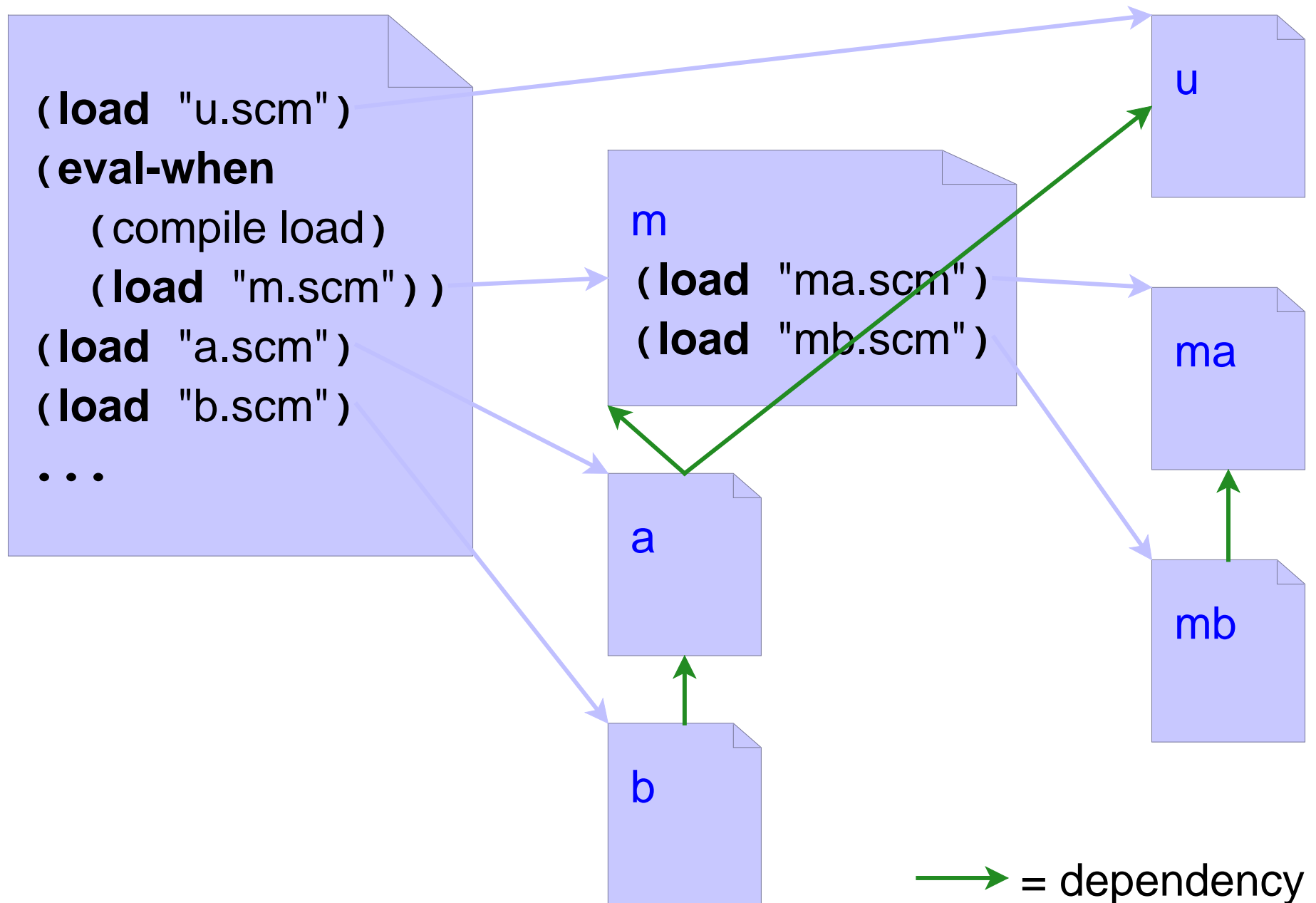
-----:---- parse.scm
$ mzc parse.scm
util.scm: bad syntax
$ █

-----:**- *shell*
```

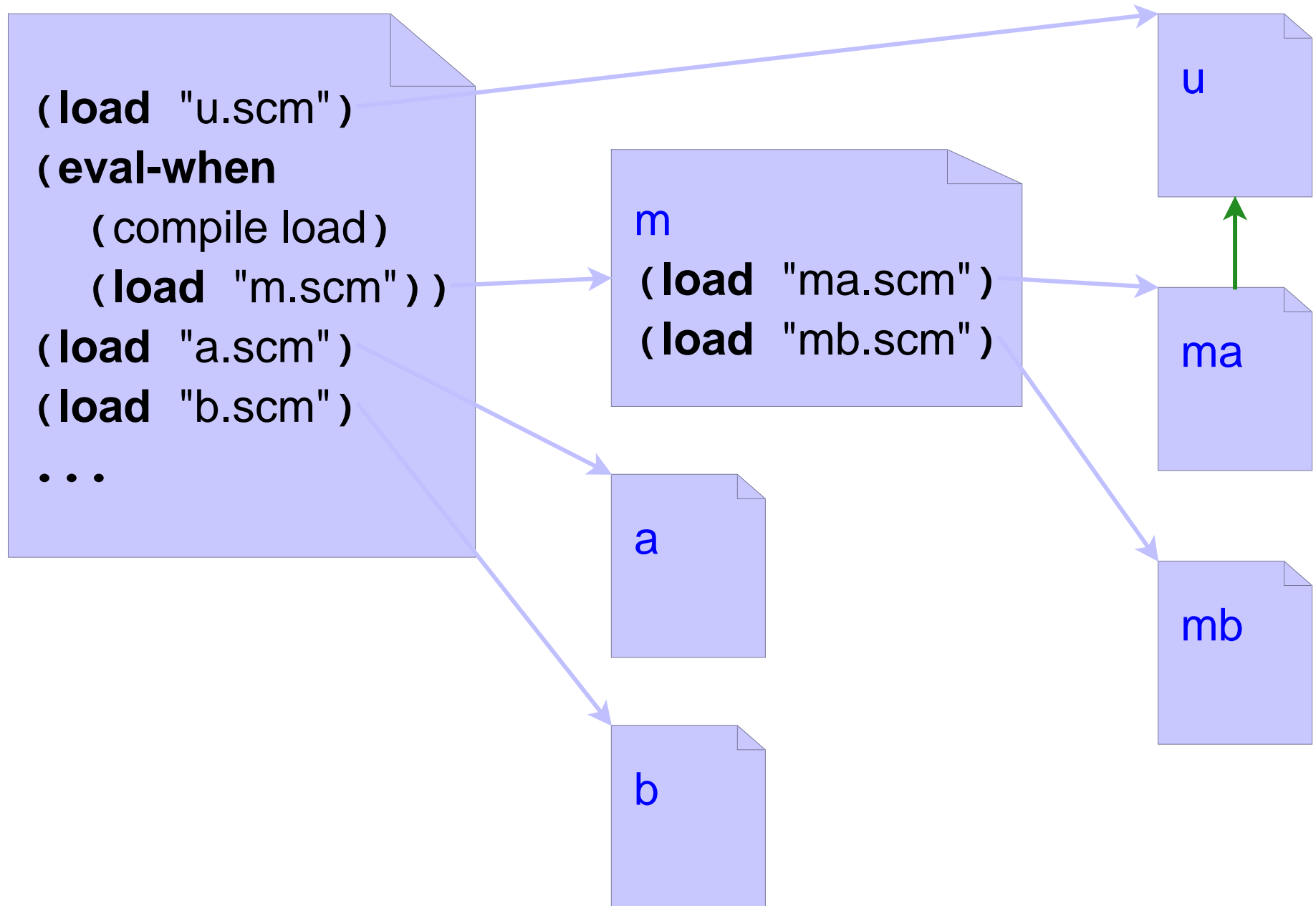
# The Trouble with Scripting the Compiler



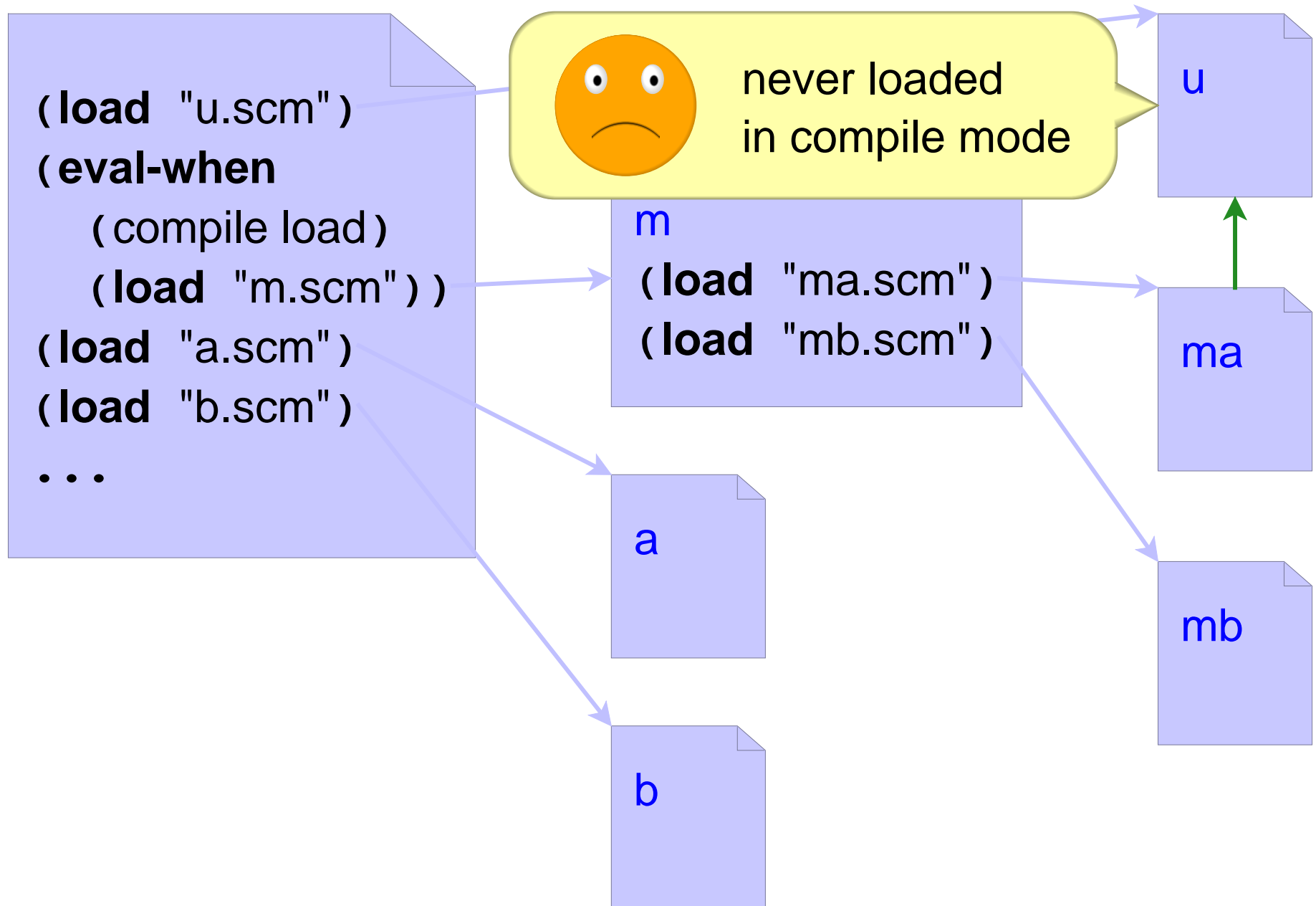
# The Trouble with Scripting the Compiler



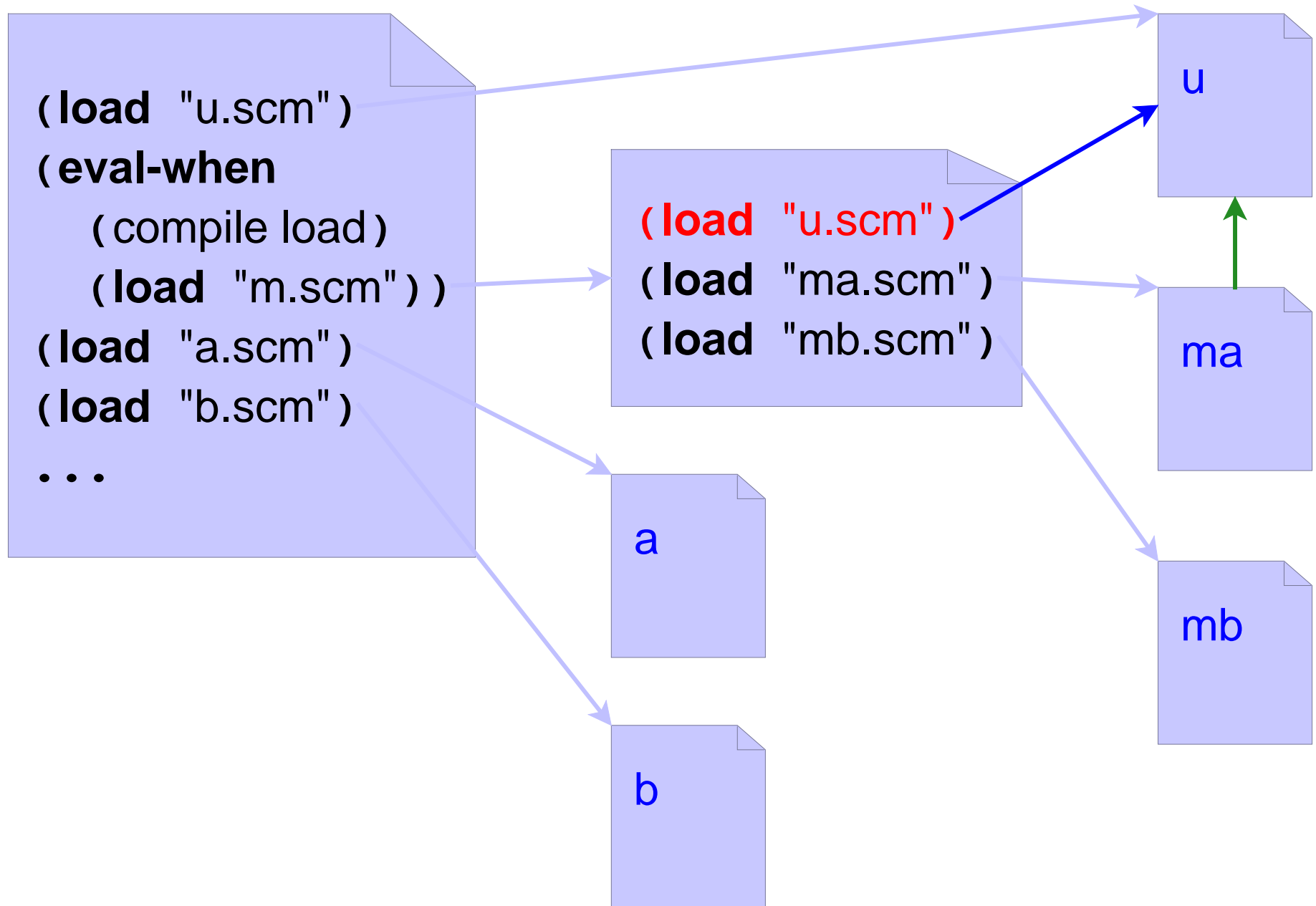
# The Trouble with Scripting the Compiler



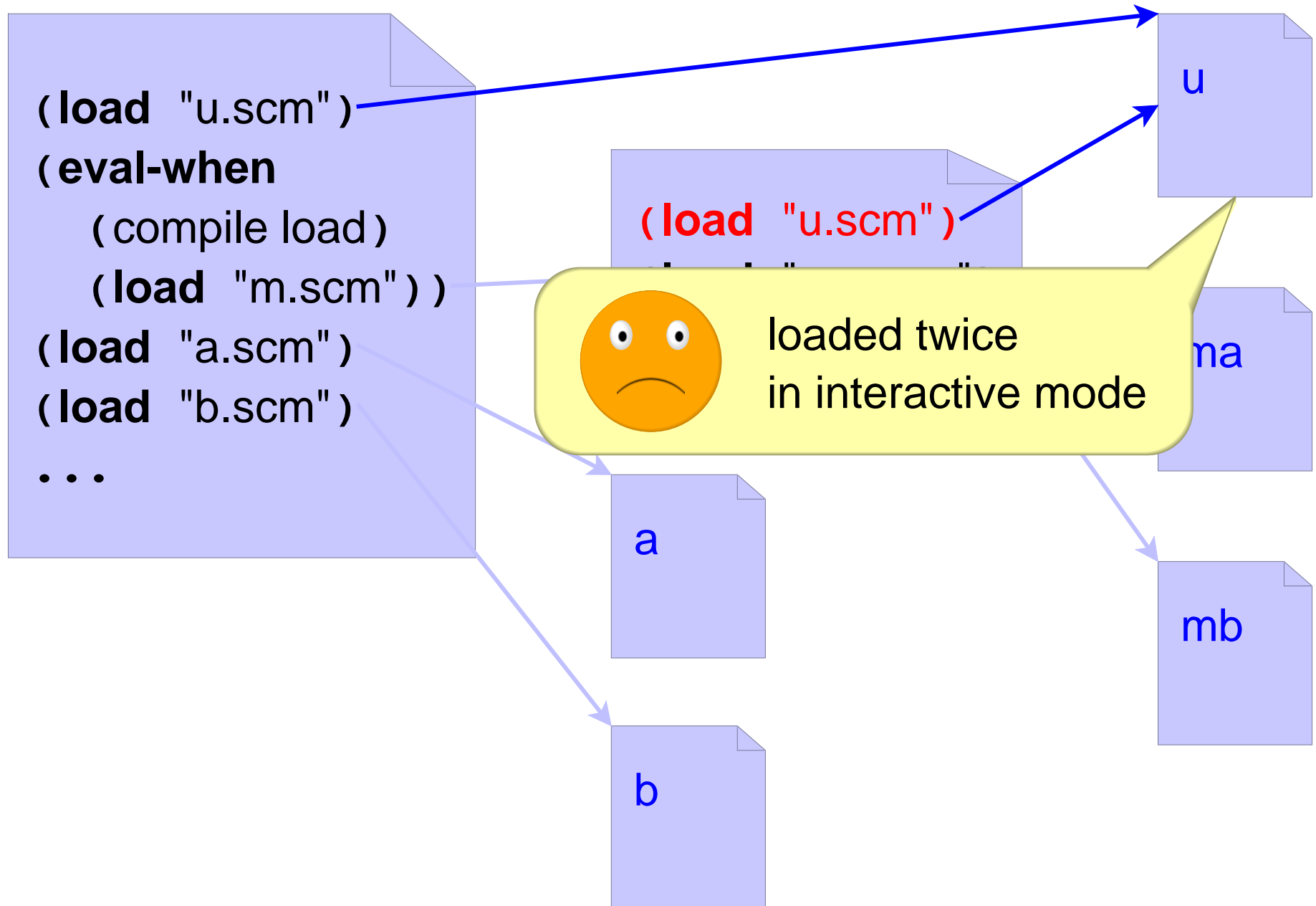
# The Trouble with Scripting the Compiler



# The Trouble with Scripting the Compiler



# The Trouble with Scripting the Compiler



➤ **Problem**

➤ **Solution**

- Declarative
- Consistent success/failure in all compilation modes
- Supports
  - lexically scoped macros
  - macro-defining macros

➤ **Example**



➤ **Experience**

# Modules and Macros

```
(module a  
  (define-syntax m ...)  
  (define f ...)  
  ...)
```




# Modules and Macros

 = run-time expressions

```
(module a
  (define-syntax m ...)
  (define f )
  )
```




# Modules and Macros

 = compile-time expressions

```
(module a
  (define-syntax m )
  (define f )
  )
```

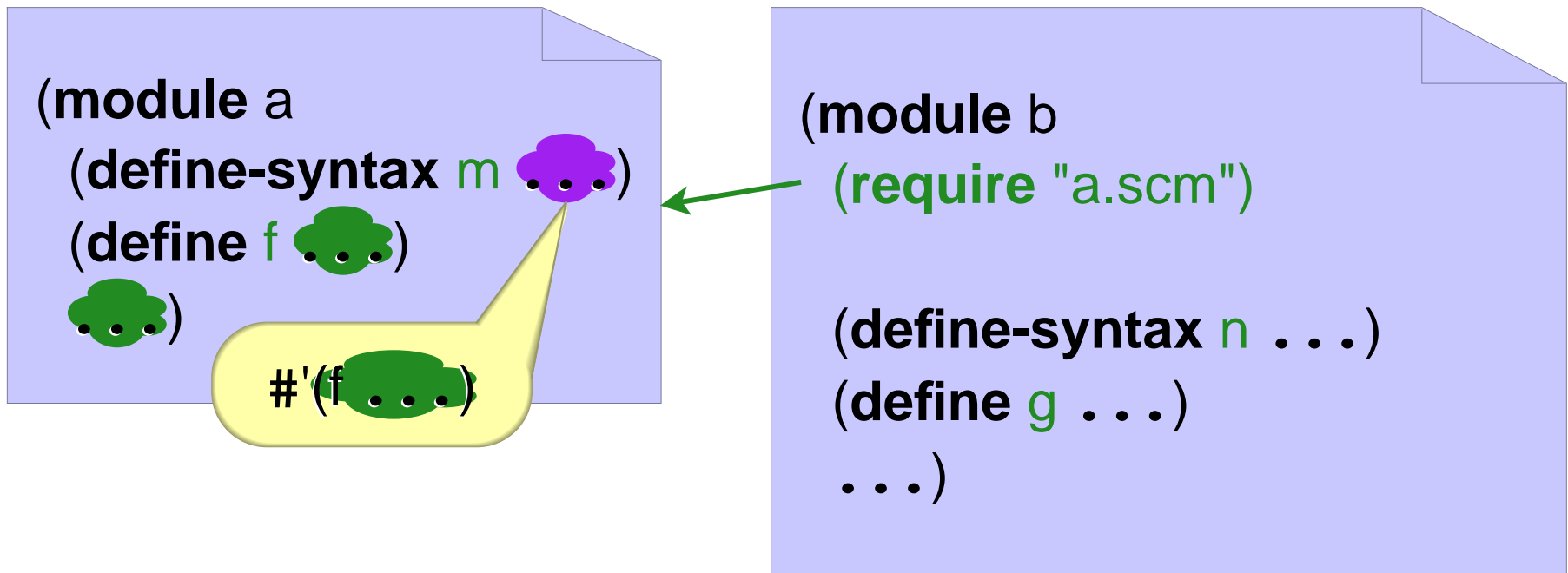
# Modules and Macros

#' in  escapes back to 

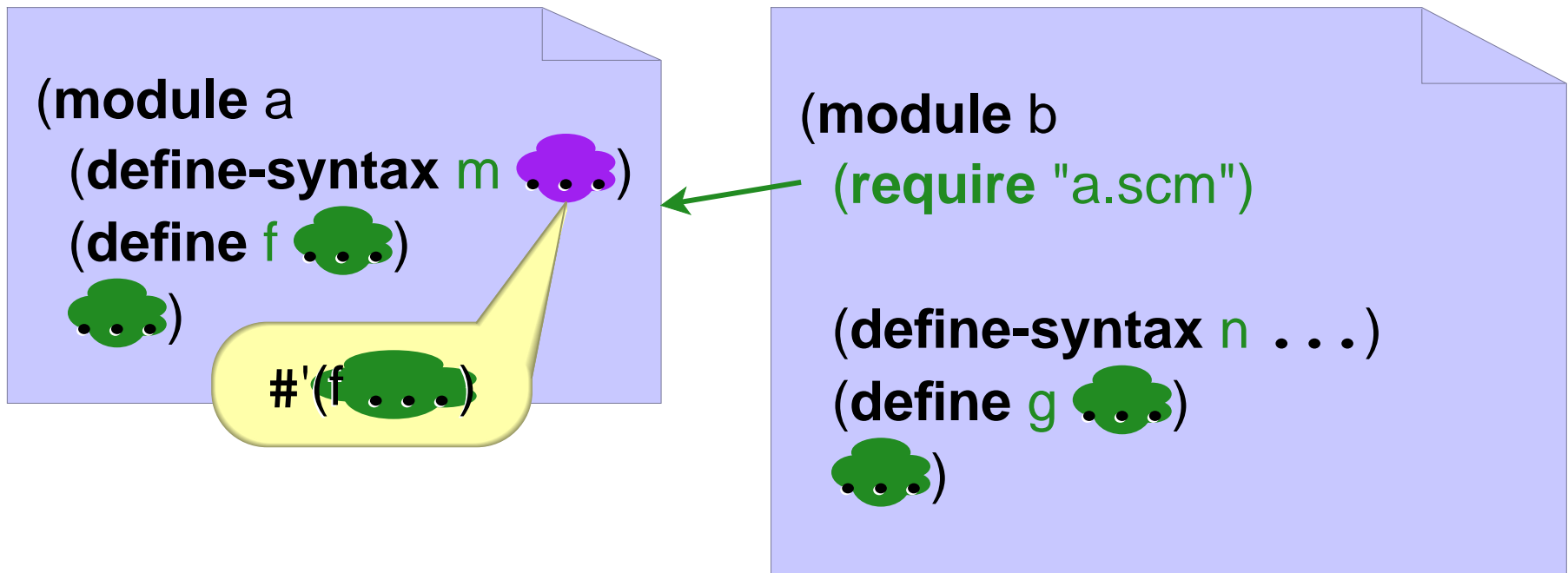
```
(module a  
  (define-syntax m )  
  (define f )  
  )
```

#'(f )

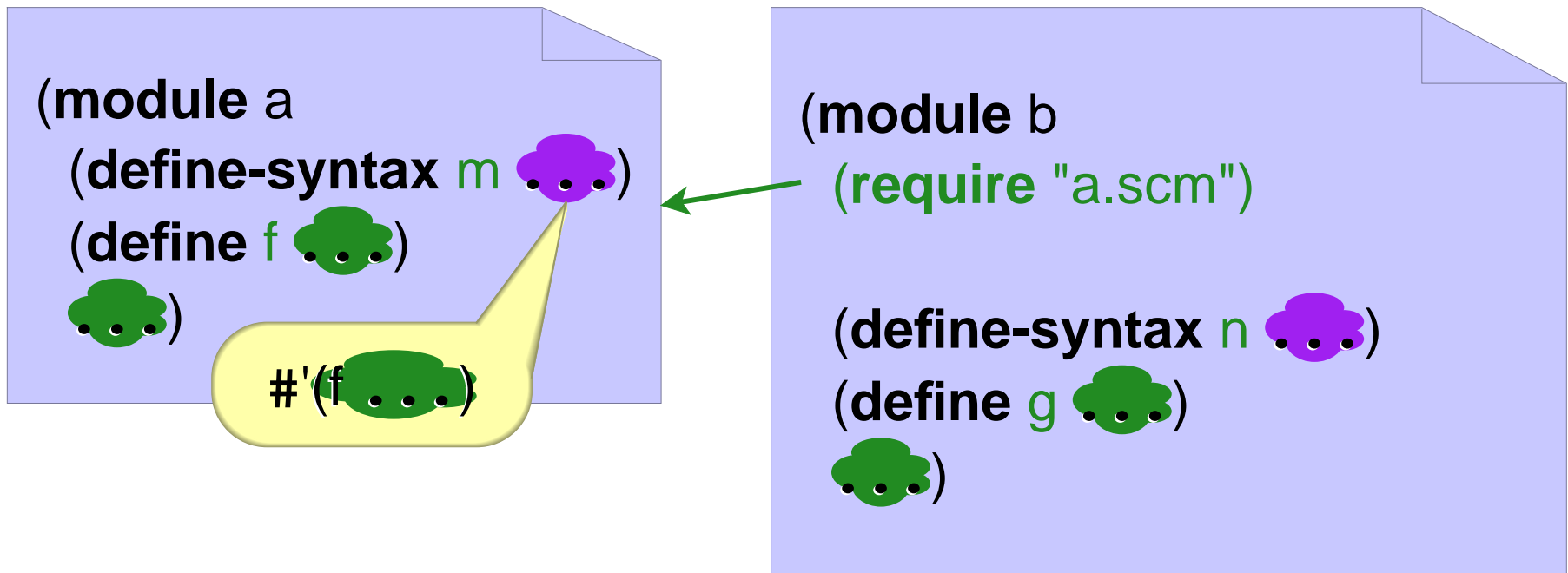
# Modules and Macros



# Modules and Macros



# Modules and Macros



# Modules and Macros

```
(module u  
  (define fold ...)  
  ...)
```




```
(module a  
  (define-syntax m ...)  
  (define f ...)  
  ...)
```




```
#'(f ...)
```

```
(module b  
  (require "a.scm")  
  (require-for-syntax "u.scm")  
  (define-syntax n ...)  
  (define g ...)  
  ...)
```

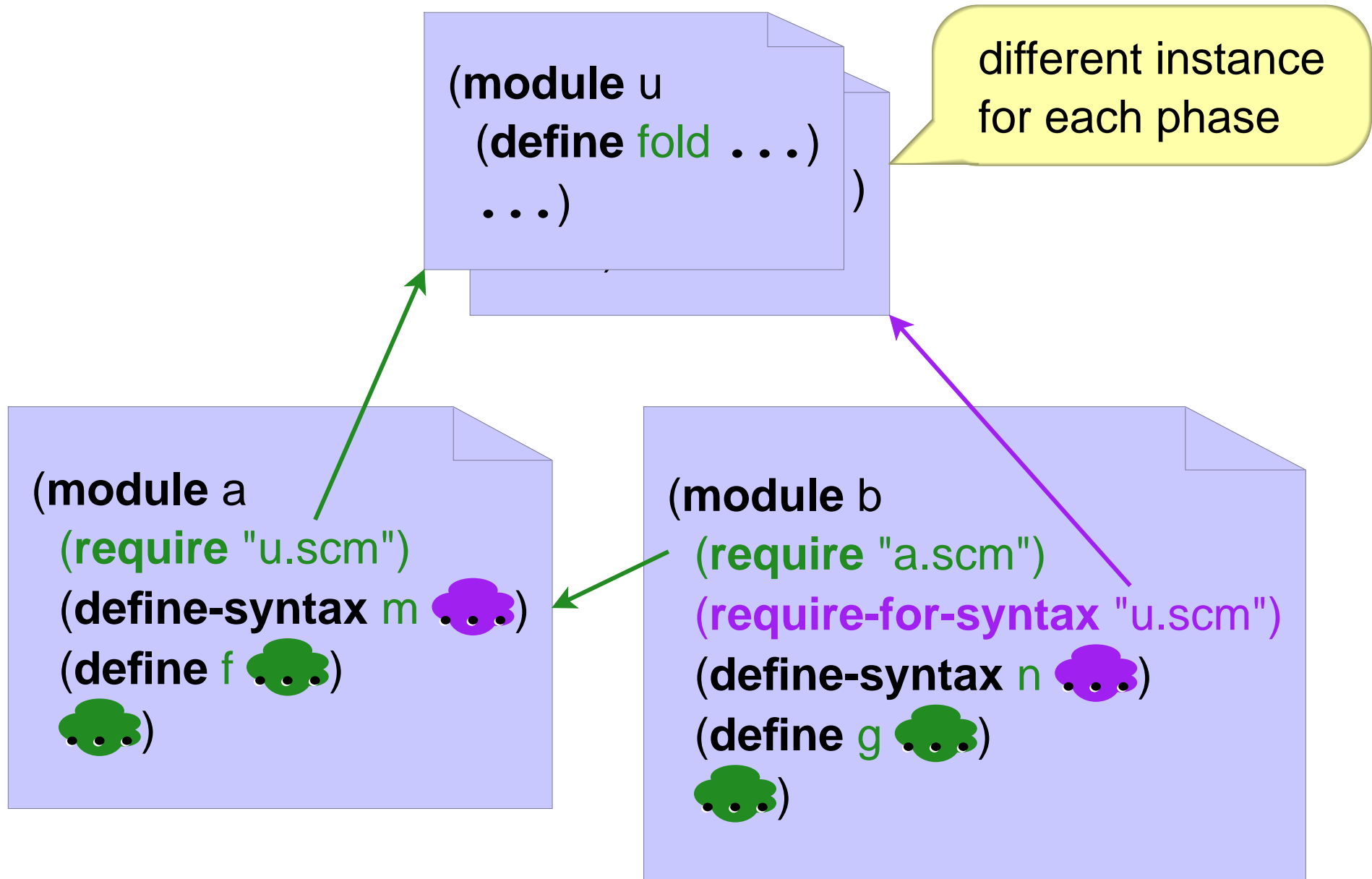
# Modules and Macros

```
(module u  
  (define fold ...)  
  ...)
```

```
(module a  
  (require "u.scm")  
  (define-syntax m ...)   
  (define f   
  )
```

```
(module b  
  (require "a.scm")  
  (require-for-syntax "u.scm")  
  (define-syntax n ...)   
  (define g   
  )
```

# Modules and Macros



➤ **Problem**

➤ **Solution**

➤ **Example**

- Record matching with static checks

➤ **Experience**

# Records and Matching

(**define-record** zebra (weight stripes))

## Records and Matching

**(define-record zebra (weight stripes))**

**(zebra 500 24) ; = a zebra instance**

## Records and Matching

**(define-record zebra (weight stripes))**

(zebra 500 24) ; = a zebra instance

---

**(define-record lizard (weight length color))**

(lizard 2 5 'green) ; = a lizard instance

## Records and Matching

**(define-record zebra (weight stripes))**

(zebra 500 24) ; = a zebra instance

---

**(define-record lizard (weight length color))**

(lizard 2 5 'green) ; = a lizard instance

---

**(record-switch . . .**

((zebra w s) . . .)

((lizard w l c) . . .))

## Records and Matching

**(define-record zebra (weight stripes))**

(zebra 500 24) ; = a zebra instance

---

**(define-record lizard (weight length color))**

(lizard 2 5 'green) ; = a lizard instance

---

**(define (animal-weight a)**

**(record-switch a**

((zebra w s) w)

((lizard w l c) w)))

(animal-weight (zebra 500 24)) ; = 500

(animal-weight (lizard 2 5 'green)) ; = 2

## Records and Matching

**(define-record** zebra (weight stripes))

(zebra 500 24) ; = a zebra instance

---

**(define-record** lizard (weight length color))

(lizard 2 5 'green) ; = a lizard instance

---

**(define** (animal-weight a)

**(record-switch** a

((zebra w s **c**) w) **syntax error**

((lizard w l c) w)))

## Records and Matching

**(define-record zebra (weight stripes))**

(zebra 500 24) ; = a zebra instance

---

**(define-record lizard (weight length color))**

(lizard 2 5 'green) ; = a lizard instance

---

**define** (animal-weight a)

**(record-switch a**

((zebra w s **c**) w) **syntax error**

((lizard w l c) w)))

**compile-time communication  
across module boundaries**

## Implementing Records

```
(module record
```

```
  (define-syntax define-record
```

```
    . . .)
```

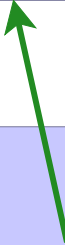
```
  (define-syntax record-switch
```

```
    . . .))
```

## Implementing Records

```
(module record  
  
  (define-syntax define-record  
    . . .)  
  (define-syntax record-switch  
    . . .))
```

```
(module zoo  
  (require "record.scm")  
  (define-record zebra (wgt sc))  
  (record-switch . . .  
    ((zebra w s) . . .))
```

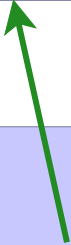


## Implementing Records

```
(module record

  (define-syntax define-record
    ...#'(define name (mkrec ...))...)
  (define-syntax record-switch
    ...#'(is-a name ...) ...))
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch . . .
    ((zebra w s) . . .))
```



## Implementing Records

```
(module record
```

```
(define-syntax define-record  
  ...#'(define name (mkrec ...))...)  
(define-syntax record-switch  
  ...#'(is-a name ...) ...))
```

```
(module zoo  
  (require "record.scm")  
  (define-record zebra (wgt sc))  
  (record-switch ...  
    ((zebra w s) ...))
```

expands to  
(**define** zebra (mkrec ...))

# Implementing Records

```
(module record
```

```
  (define-syntax define-record
```

```
    ...#'(define name (mkrec ...))...)
```

```
  (define-syntax record-switch
```

```
    ...#'(is-a name ...) ...))
```

```
(module zoo
```

```
  (require "record.scm")
```

```
  (define-record zebra (wgt sc))
```

```
  (record-switch ...
```

```
    ((zebra w s) ...))
```

expands to

```
(define zebra (mkrec ...))
```

expands to

```
(is-a zebra ...)
```

# Implementing Records

```
(module record
  (require "rt.scm")

  (define-syntax define-record
    ...#'(define name (mkrec ...))...)
  (define-syntax record-switch
    ...#'(is-a name ...) ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

expands to  
(define zebra (mkrec ...))

expands to  
(is-a zebra ...)

## Implementing Records

```
(module record
  (require "rt.scm")

  (define-syntax define-record
    ... table ...)
  (define-syntax record-switch
    ... table ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ... table ...)
  (define-syntax record-switch
    ... table ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ... table ...)
  (define-syntax record-switch
    ... table ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

adds zebra to table  
while expanding

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ... table ...)
  (define-syntax record-switch
    ... table ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

adds zebra to table  
while expanding

finds zebra in table  
while expanding

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```



# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

```
(module client
  (require "zoo.scm")
  (record-switch ...
    ((zebra w s) ...)))
```

# Implementing Records


```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((ze
```

```
(module client
  (require "zoo.scm")
  (record-switch ...
    ((zebra w s) ...)))
```

 cannot find zebra in table

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...#' (for-syntax ...table...)...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...)
  ...)
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

```
(module client
  (require "zoo.scm")
  (record-switch ...
    ((zebra w s) ...)))
```

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...#' (for-syntax ...table...)...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...))
```

now expands to  
(for-syntax ...table...)  
(define zebra (mkrec ...))

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch ...
    ((zebra w s) ...))
```

```
(module client
  (require "zoo.scm")
  (record-switch ...
    ((zebra w s) ...)))
```

# Implementing Records

```
(module record
  (require "rt.scm")
  (require-for-syntax "ct.scm")
  (define-syntax define-record
    ...#' (for-syntax ...table...)...)
  (define-syntax record-switch
    ...))
```

```
(module rt
  (define mkrec ...)
  (define is-a ...))
```

```
(module ct
  (define table ...))
```

```
(module zoo
  (require "record.scm")
  (define-record zebra (wgt sc))
  (record-switch
    ((zebra w s ...))
```

now expands to  
(for-syntax ...table...)  
(define zebra (mkrec ...))

```
(module client
  (require "zoo.scm")
  (record-switch ...
    ((zebra w s ...)))
```



finds zebra  
in table

- **Problem**
- **Solution**
- **Example**
- **Experience**

- The many languages & compilers of PLT Scheme

# PLT Scheme

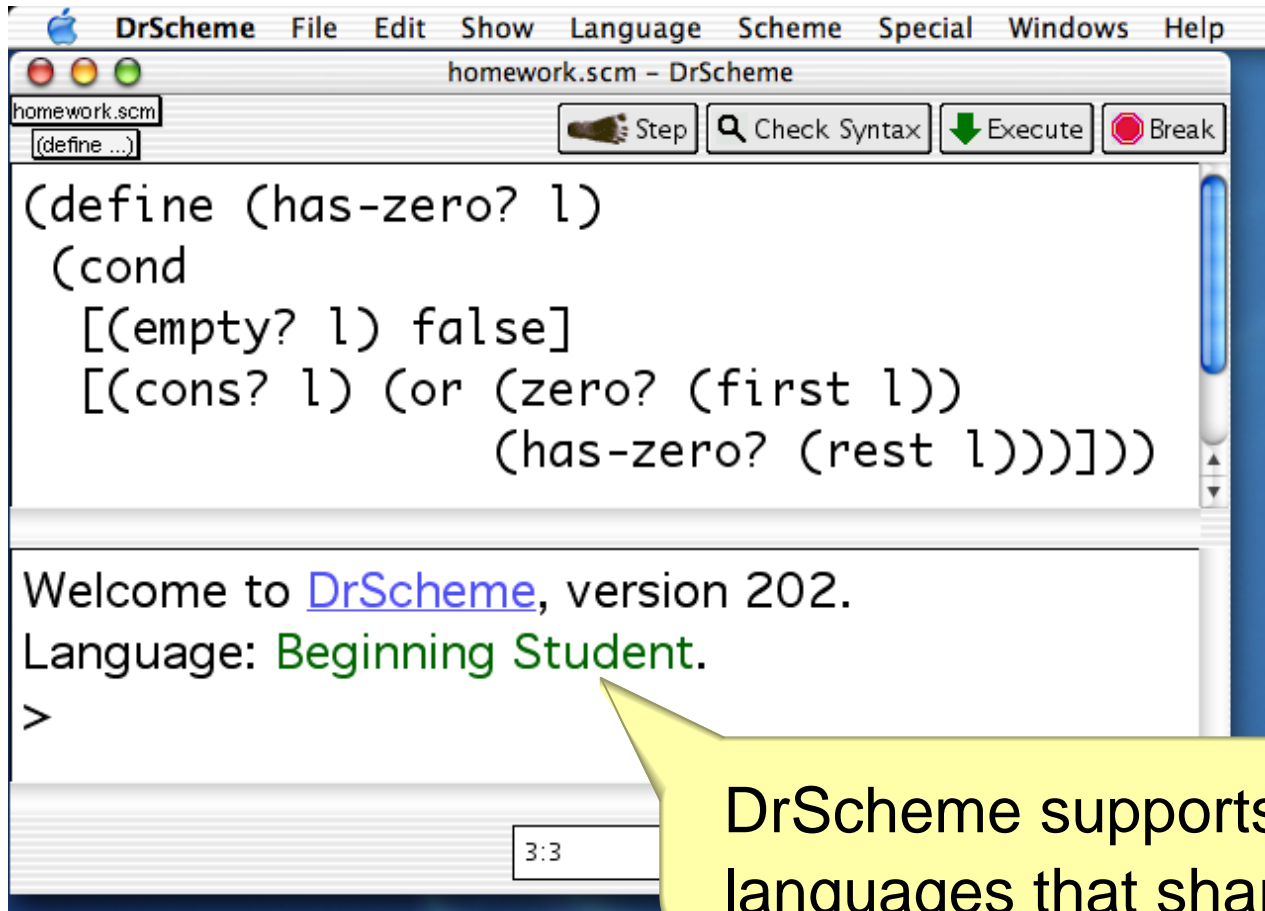
```
DrScheme File Edit Show Language Scheme Special Windows Help
homework.scm - DrScheme
homework.scm
(define ...) Step Check Syntax Execute Break

(define (has-zero? l)
  (cond
    [(empty? l) false]
    [(cons? l) (or (zero? (first l))
                   (has-zero? (rest l)))]))

Welcome to DrScheme, version 202.
Language: Beginning Student.
>

3:3 Read/Write not running
```

# PLT Scheme



The image shows a screenshot of the DrScheme IDE. The window title is "DrScheme" and the file name is "homework.scm". The menu bar includes "File", "Edit", "Show", "Language", "Scheme", "Special", "Windows", and "Help". The toolbar contains buttons for "Step", "Check Syntax", "Execute", and "Break". The main text area contains the following Scheme code:

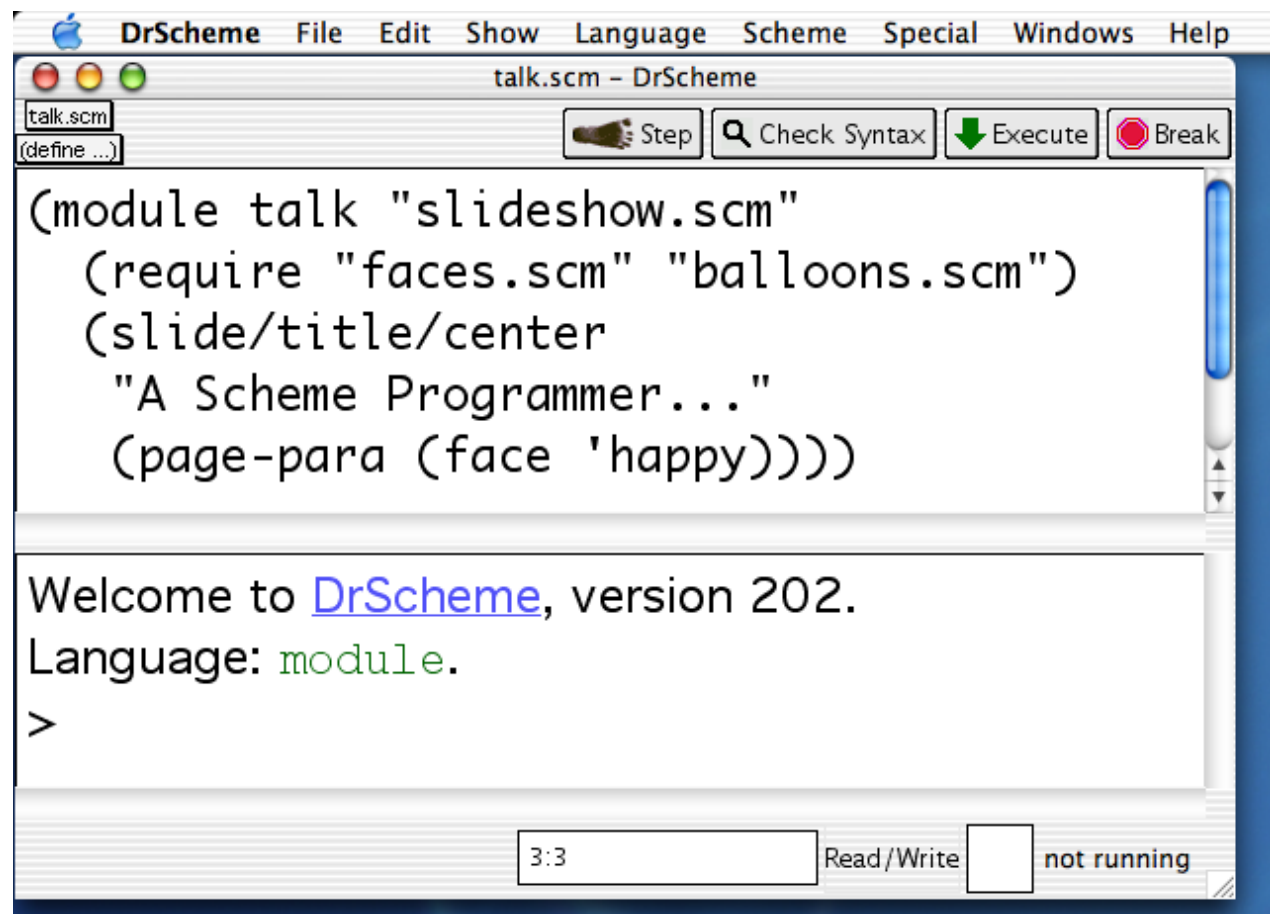
```
(define (has-zero? l)
  (cond
    [(empty? l) false]
    [(cons? l) (or (zero? (first l))
                   (has-zero? (rest l)))]))
```

Below the code, a message box displays the following text:

```
Welcome to DrScheme, version 202.
Language: Beginning Student.
>
```

A yellow callout bubble points to the "Beginning Student" language name, containing the text: "DrScheme supports multiple languages that share libraries".

# PLT Scheme



The image shows a screenshot of the DrScheme IDE. The window title is "talk.scm - DrScheme". The menu bar includes "DrScheme", "File", "Edit", "Show", "Language", "Scheme", "Special", "Windows", and "Help". The toolbar contains buttons for "Step", "Check Syntax", "Execute", and "Break". The main text area contains the following Scheme code:

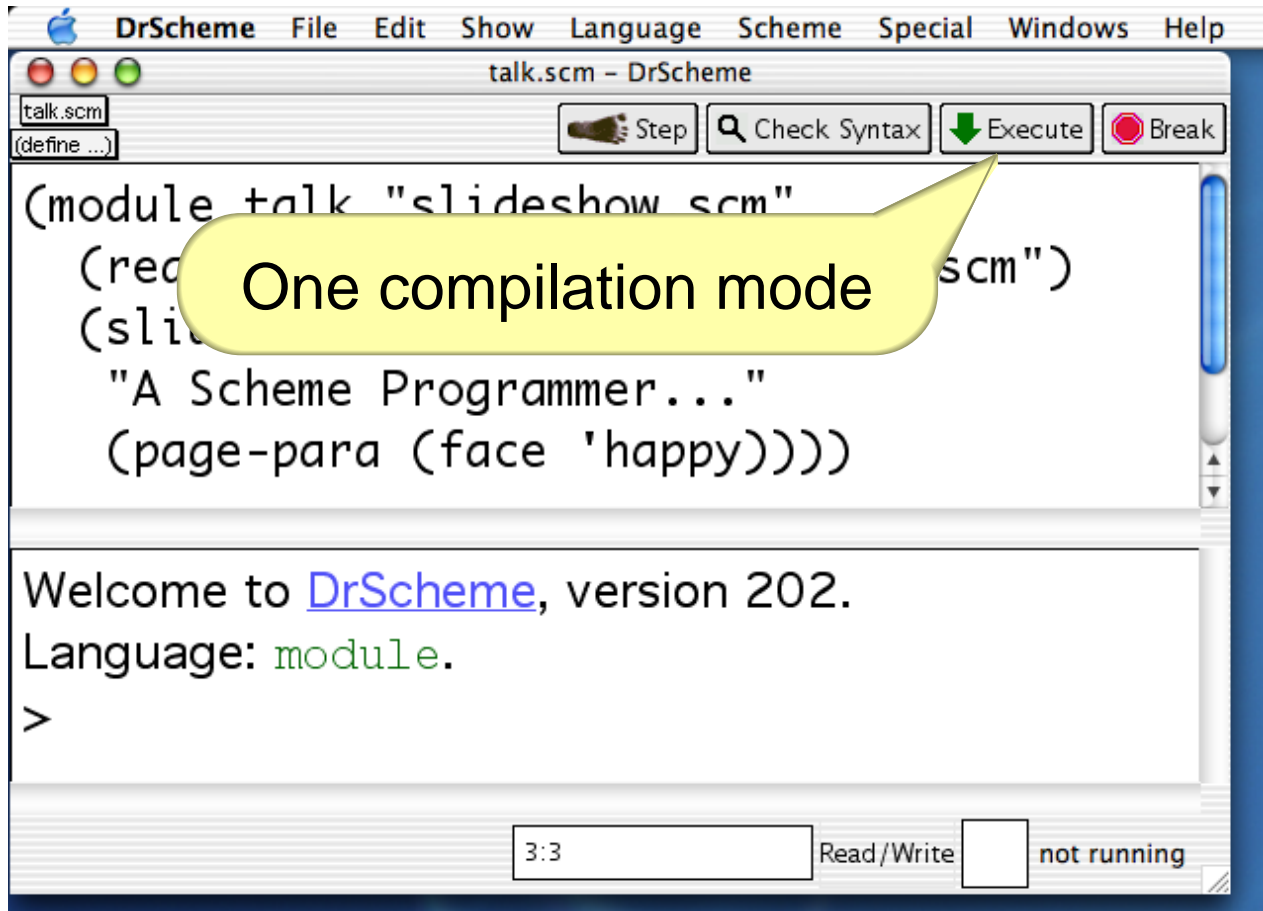
```
(module talk "slideshow.scm"
  (require "faces.scm" "balloons.scm")
  (slide/title/center
    "A Scheme Programmer..."
    (page-para (face 'happy))))
```

Below the code, the output area displays the following text:

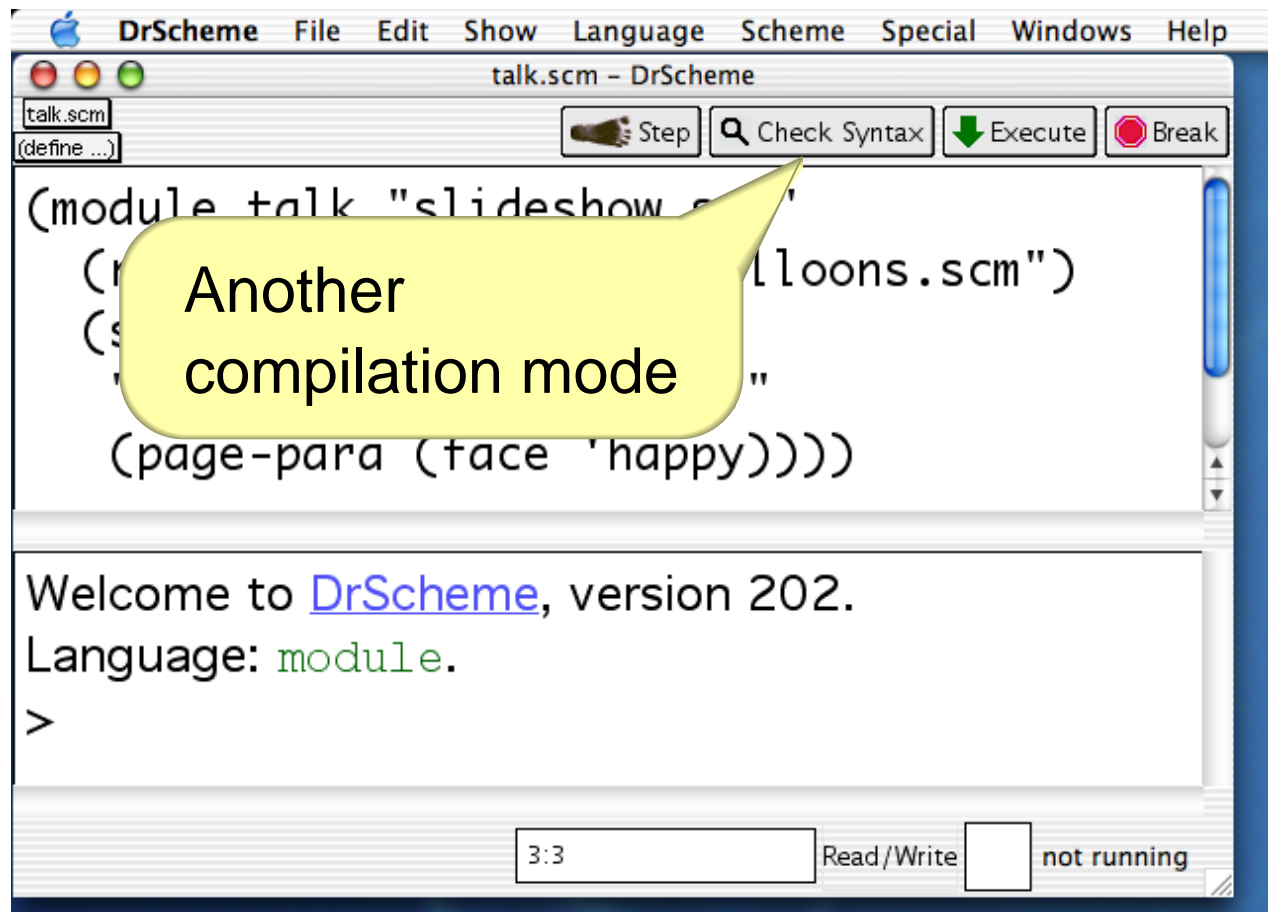
```
Welcome to DrScheme, version 202.
Language: module.
>
```

At the bottom of the window, there is a status bar showing "3:3" in a text box, followed by "Read/Write" and a checkbox, and "not running" next to another checkbox.

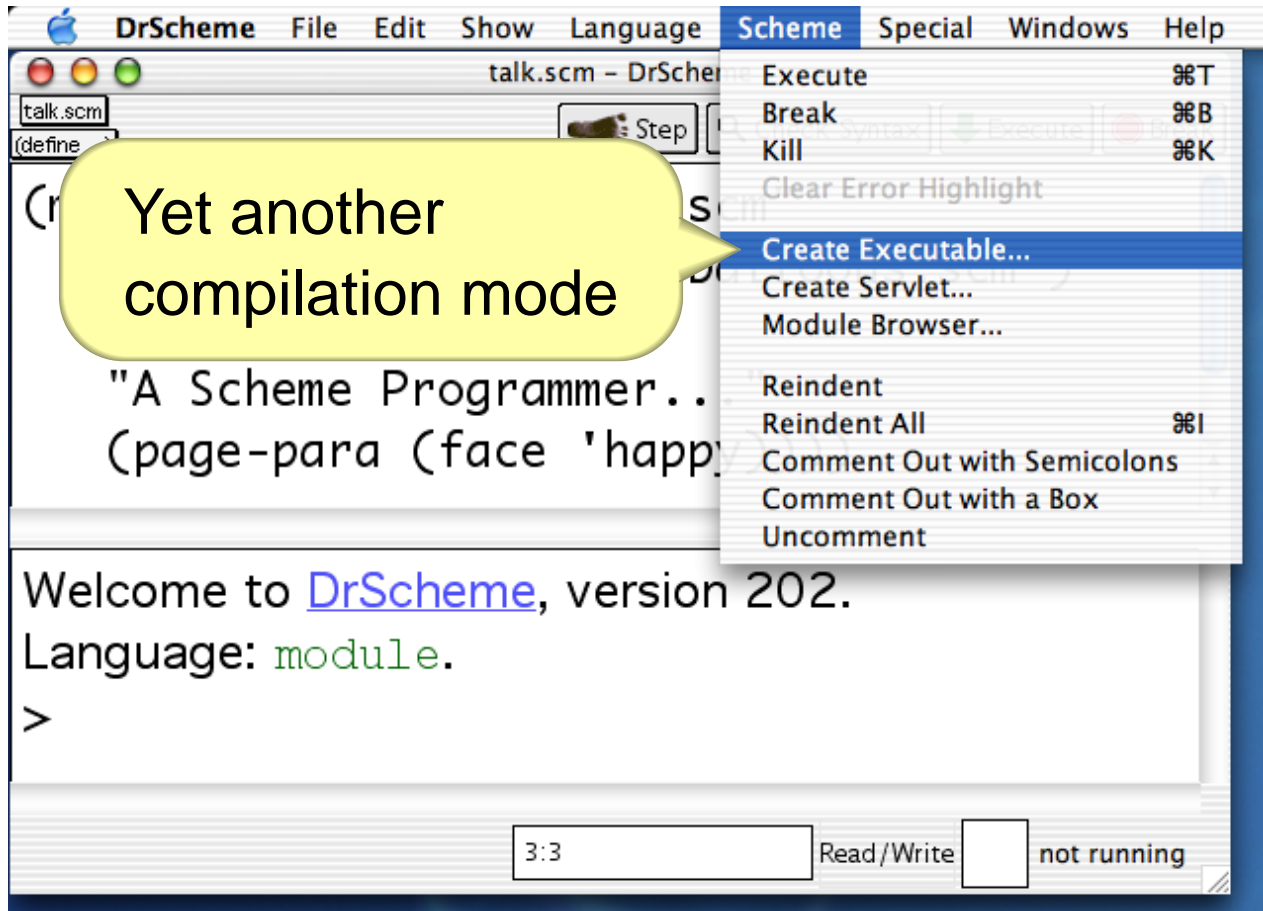
# PLT Scheme



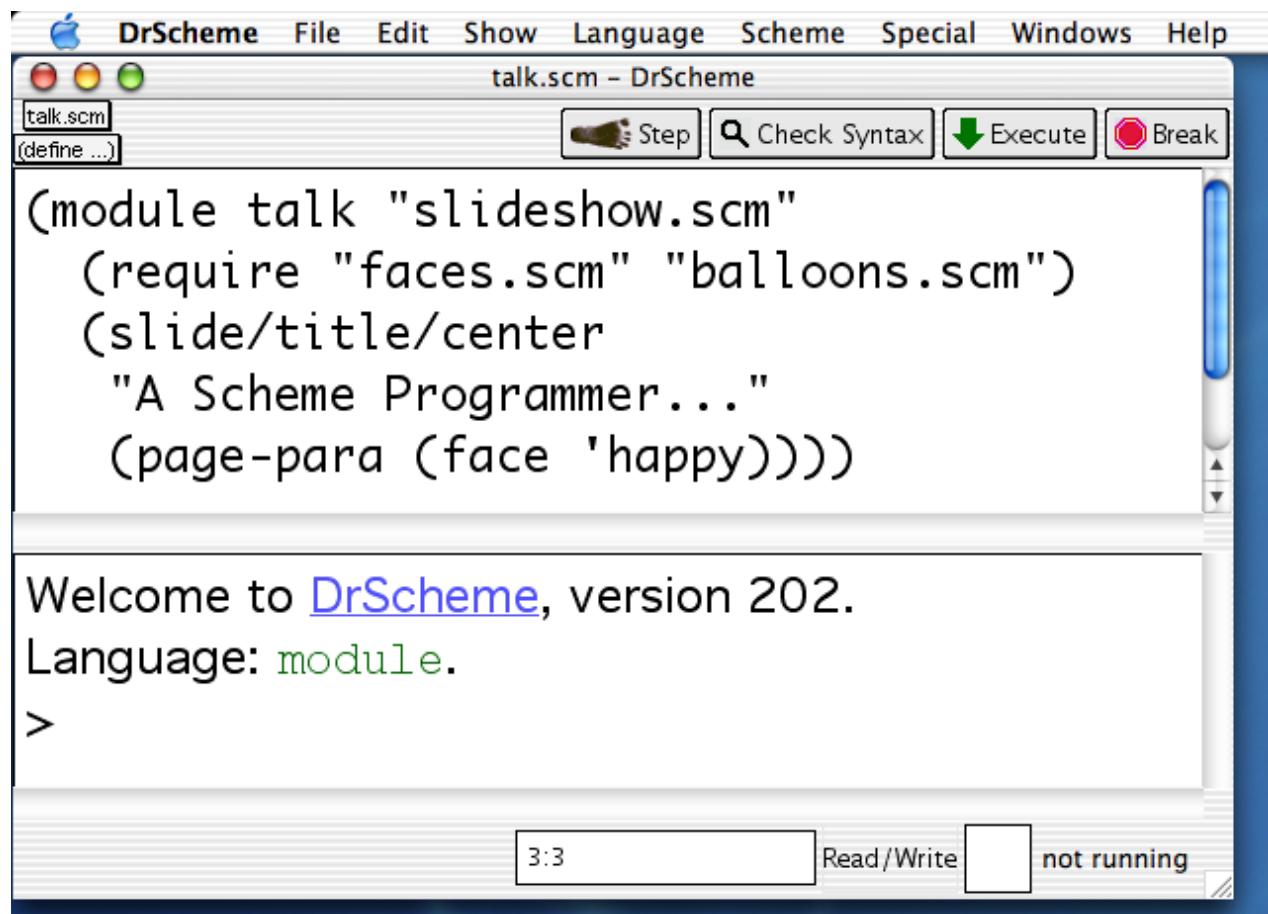
# PLT Scheme



# PLT Scheme



# PLT Scheme



The screenshot shows the DrScheme IDE interface. The menu bar includes DrScheme, File, Edit, Show, Language, Scheme, Special, Windows, and Help. The window title is "talk.scm - DrScheme". The toolbar contains buttons for Step, Check Syntax, Execute, and Break. The main text area contains the following Scheme code:

```
(module talk "slideshow.scm"
  (require "faces.scm" "balloons.scm")
  (slide/title/center
    "A Scheme Programmer..."
    (page-para (face 'happy))))
```

The output area shows the following text:

```
Welcome to DrScheme, version 202.
Language: module.
>
```

The status bar at the bottom indicates the current position is 3:3, the file is Read/Write, and the program is not running.



# Acknowledgements

- **Modules**

- Influenced by practically every implemented module system

- **Macros**

- Long Scheme history
- Builds directly on Dybvig-Hieb-Bruggeman (1993)

- **Making it work**

- PLT members