Contracts for First-Class Modules

T. Stephen Strickland
sstrickl@ccs.neu.edu

Matthias Felleisen
matthias@ccs.neu.edu

Northeastern University

Boston, MA, USA

Debugging Plugins for DrScheme

A dynamic extension to DrScheme...

A dynamic extension to DrScheme...

... that itself has dynamic extensions...

A dynamic extension to DrScheme...

... that itself has dynamic extensions...

... that, unsurprisingly, had bugs.

Error without Contracts

```
cdr: expected argument of type <pair>;
given #<procedure>

 === context ===
plt/collects/stats/stats.ss:12:2: loop
plt/collects/stats/tool.ss:42:25
plt/collects/scheme/private/more-scheme.ss:158:2:
call-with-break-parameterization
plt/collects/scheme/private/more-scheme.ss:274:2:
call-with-exception-handler
```

This is the *complete* stack trace.

Error with Contracts

```
(unit pearson-pm) broke the contract
  ((cons/c
    string?
    ((listof number?) -> (real-in -1.0 1.0))
   ->
   void?
on enqueue; expected <cons>, given: #<procedure>
```

Software Contracts

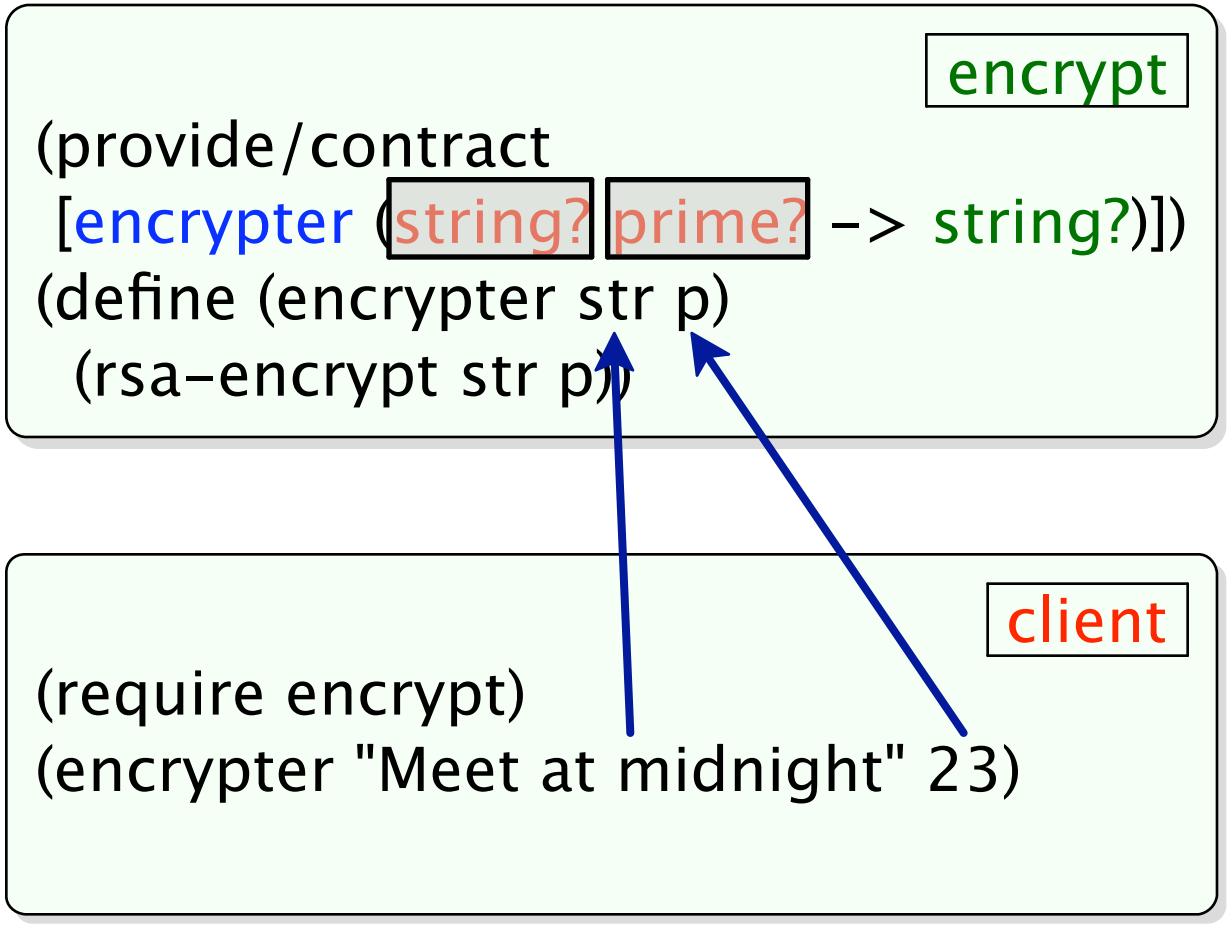
A software contract is a specification and an agreement.

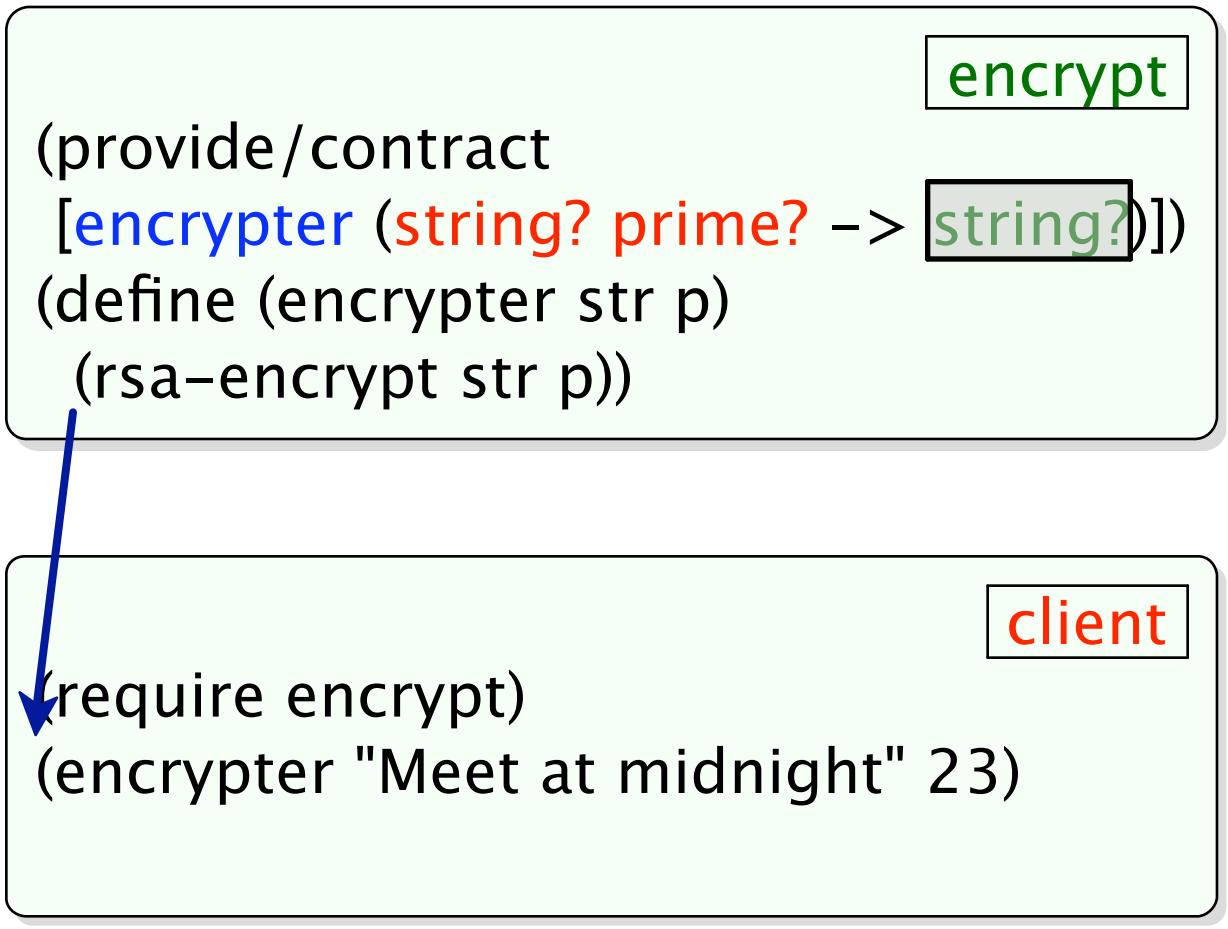
encrypt

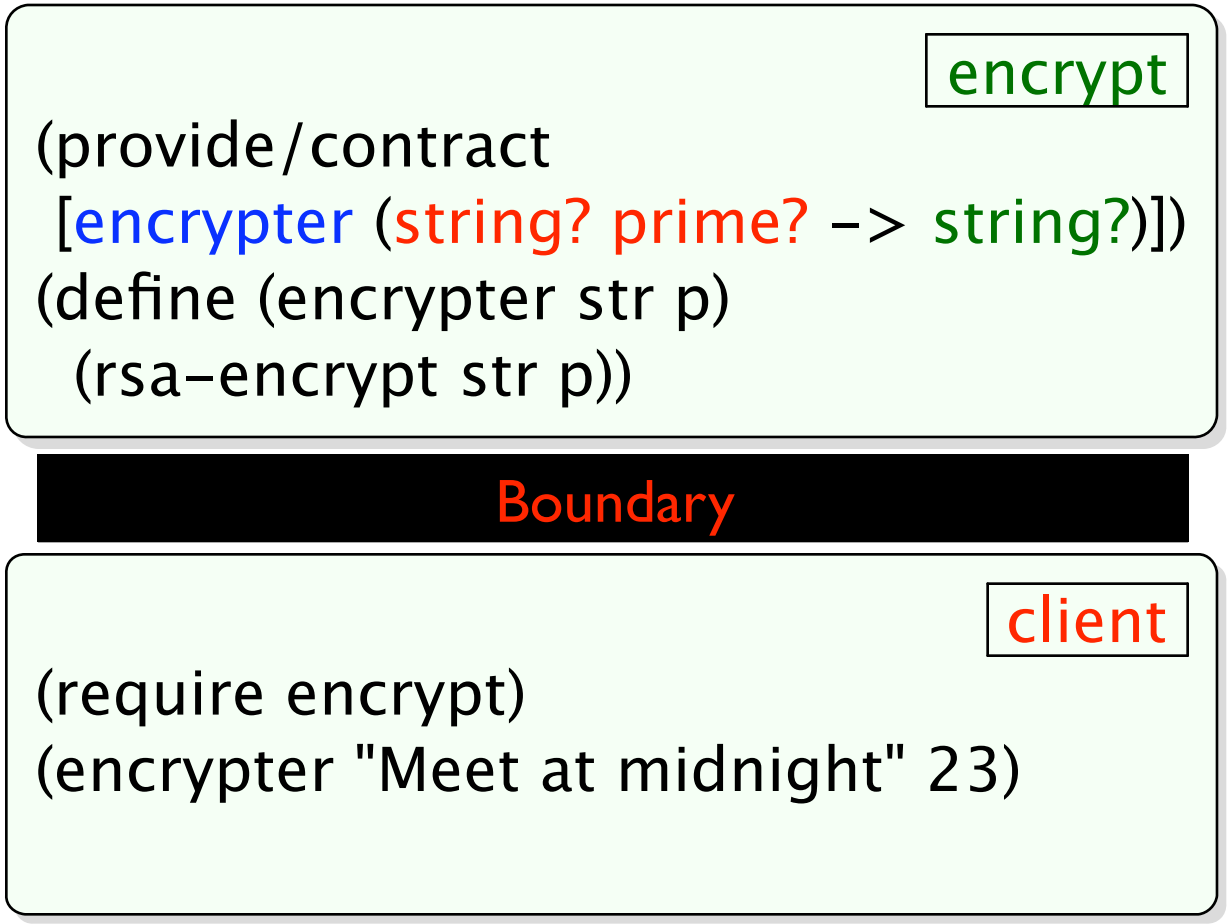
```
(provide/contract
 [encrypter (string? prime? -> string?)])
(define (encrypter str p)
  (rsa-encrypt str p))
```

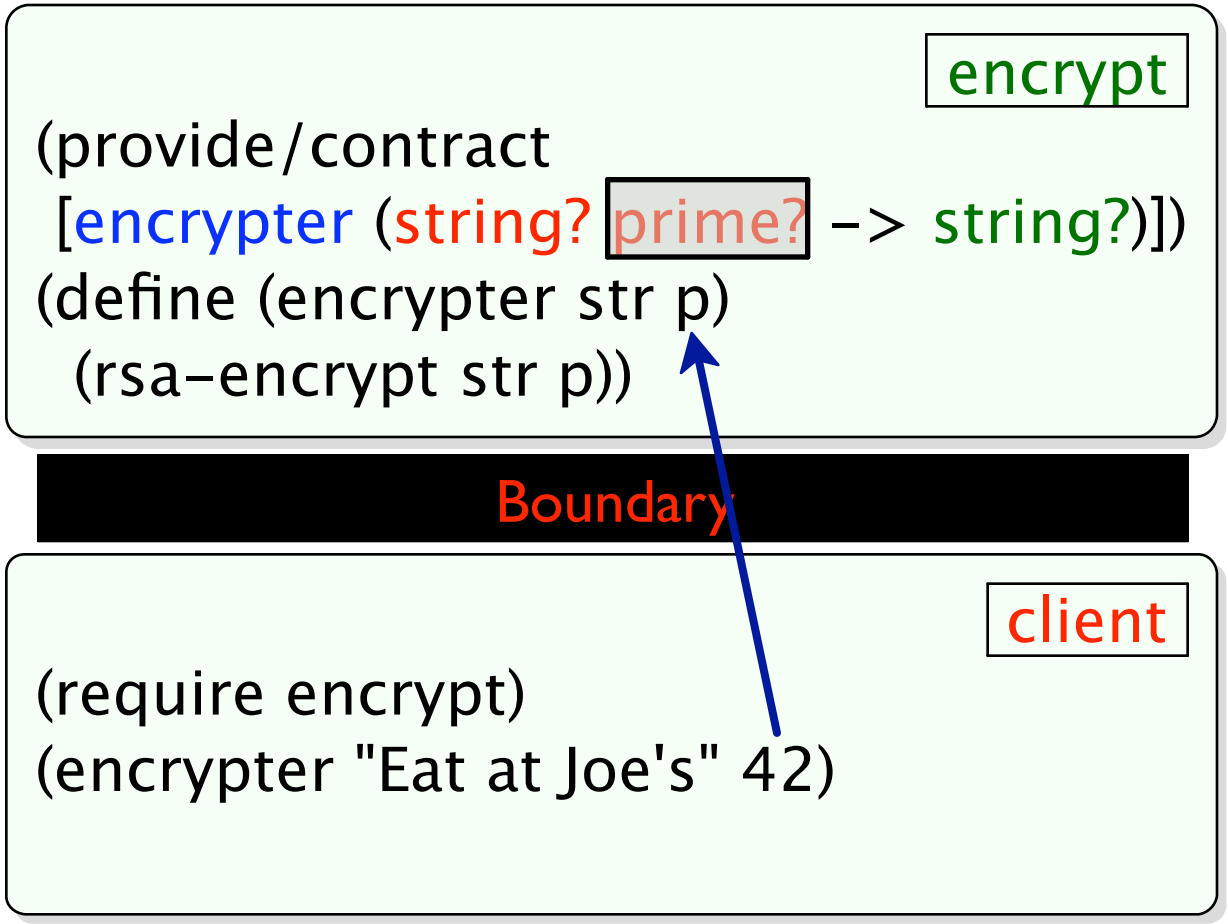
client

```
(require encrypt)
(encrypter "Meet at midnight" 23)
```


encrypt
(provide/contract
[encrypter (string? prime? -> string?)])
(define (encrypter str p)
(rsa-encrypt str p))
client
(require encrypt)
(encrypter "Meet at midnight" 23)


encrypt
(provide/contract
[encrypter (string? prime? -> string?)])
(define (encrypter str p)
(rsa-encrypt str p))
client
(require encrypt)
(encrypter "Meet at midnight" 23)


encrypt
(provide/contract
[encrypter (string? prime? -> string?)])
(define (encrypter str p)
(rsa-encrypt str p))
Boundary
client
(require encrypt)
(encrypter "Meet at midnight" 23)



client broke the contract (string? prime? -> string?) on encrypter; expected <prime?>, given: 42

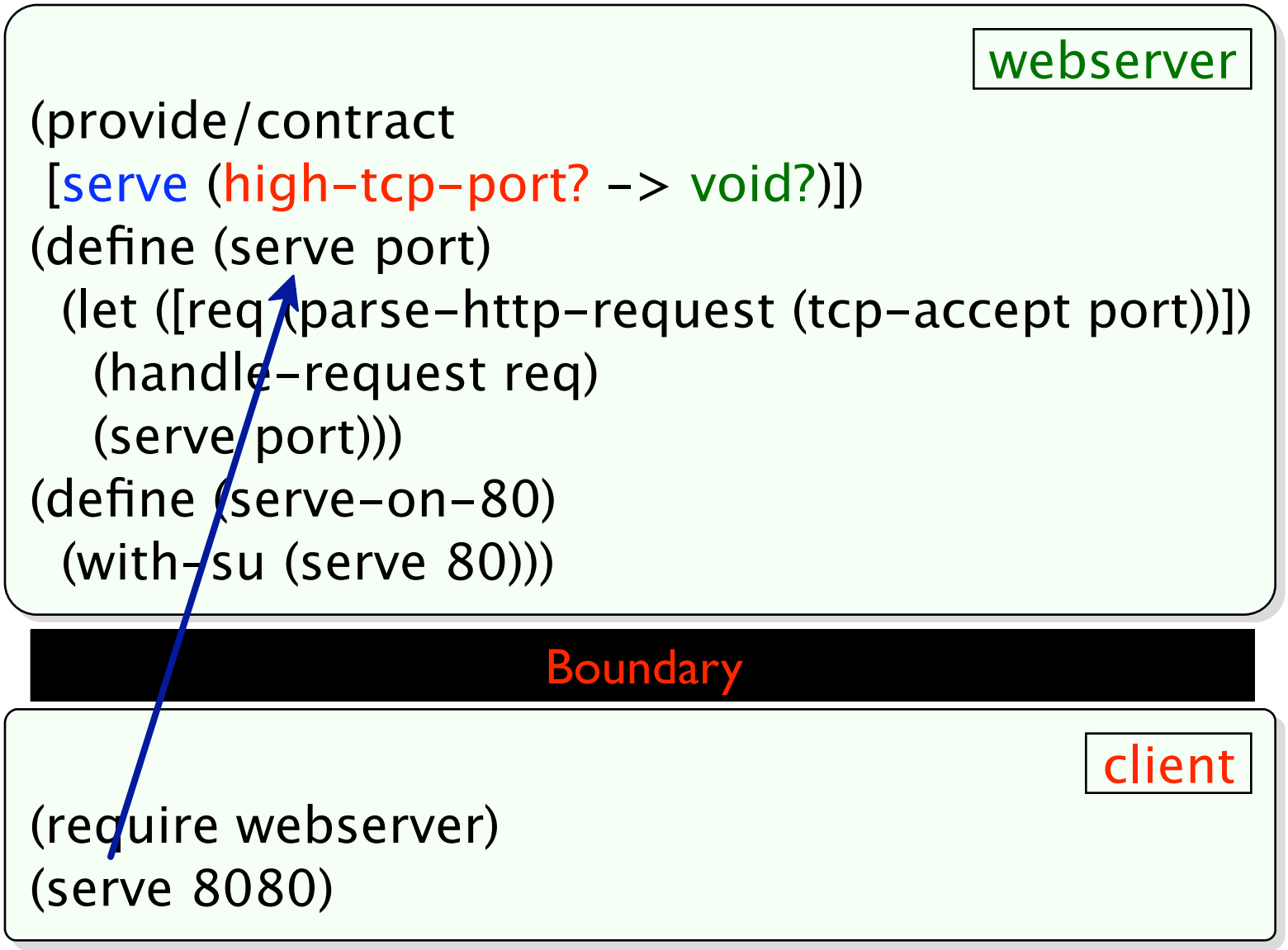
webserver

```
(provide/contract
 [serve (high-tcp-port? -> void?)])
(define (serve port)
  (let ([req (parse-http-request (tcp-accept port))])
    (handle-request req)
    (serve port)))
(define (serve-on-80)
  (with-su (serve 80)))
```

Boundary

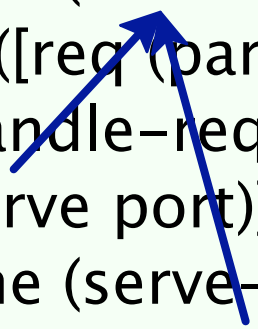
client

```
(require webserver)
(serve 8080)
```


webserver
(provide/contract
[serve (high-tcp-port? -> void?)])
(define (serve port)
(let ([req (parse-http-request (tcp-accept port))])
(handle-request req)
(serve port)))
(define (serve-on-80)
(with-su (serve 80)))
Boundary
client
(require webserver)
(serve 8080)

webserver

```
(provide/contract
 [serve (high-tcp-port? -> void?)])
(define (serve port)
  (let ([req (parse-http-request (tcp-accept port))])
    (handle-request req)
    (serve port)))
(define (serve-on-80)
  (with-su (serve 80)))
```



Boundary

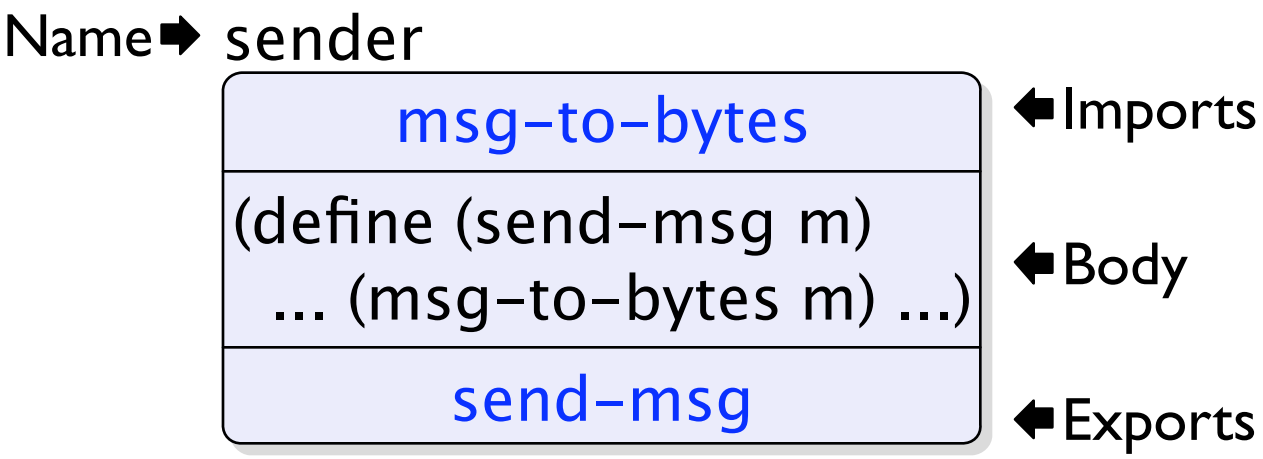
client

```
(require webserver)
(serve 8080)
```

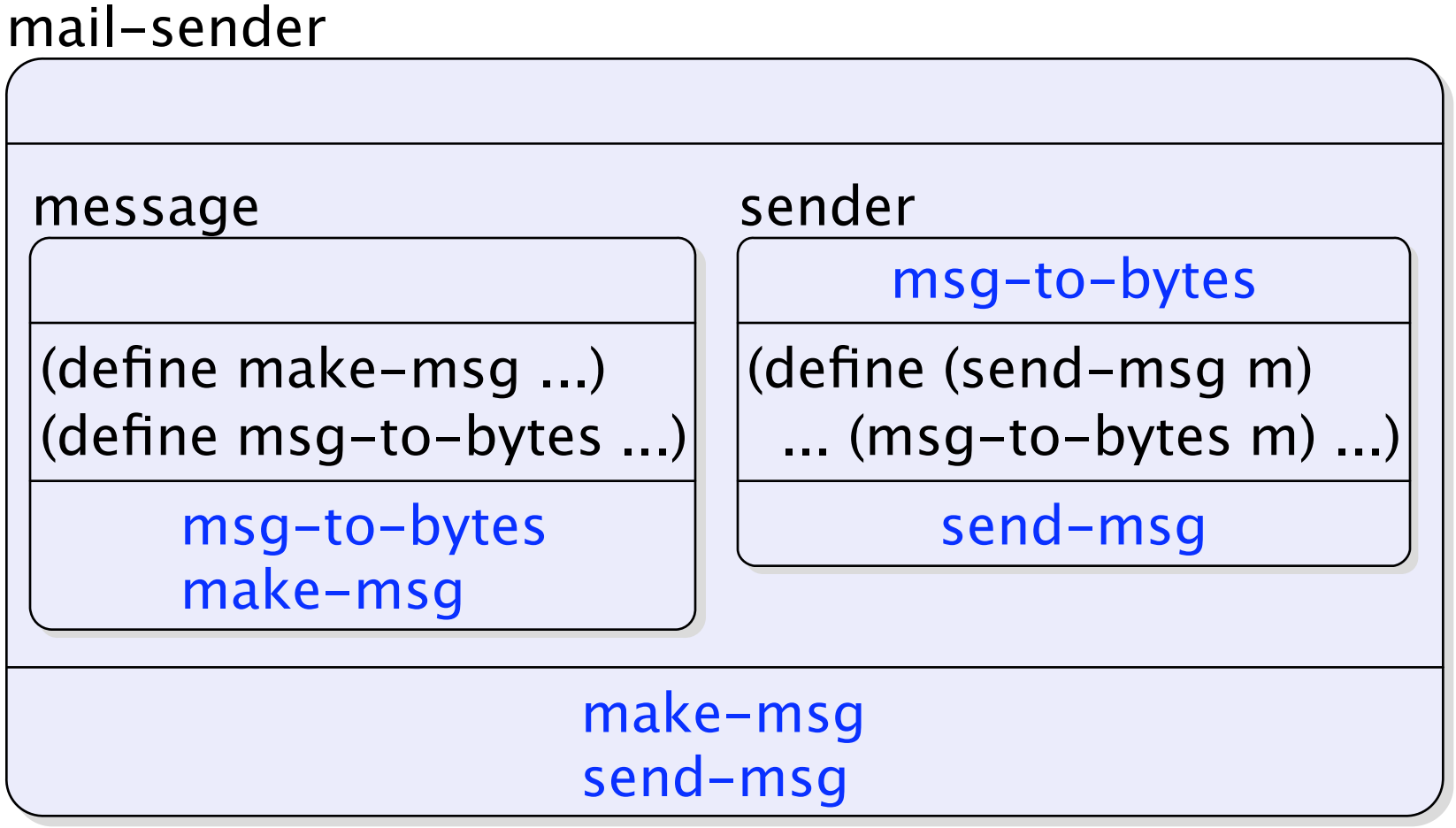
Units

PLT Scheme units are first-class, dynamically-linked modules.

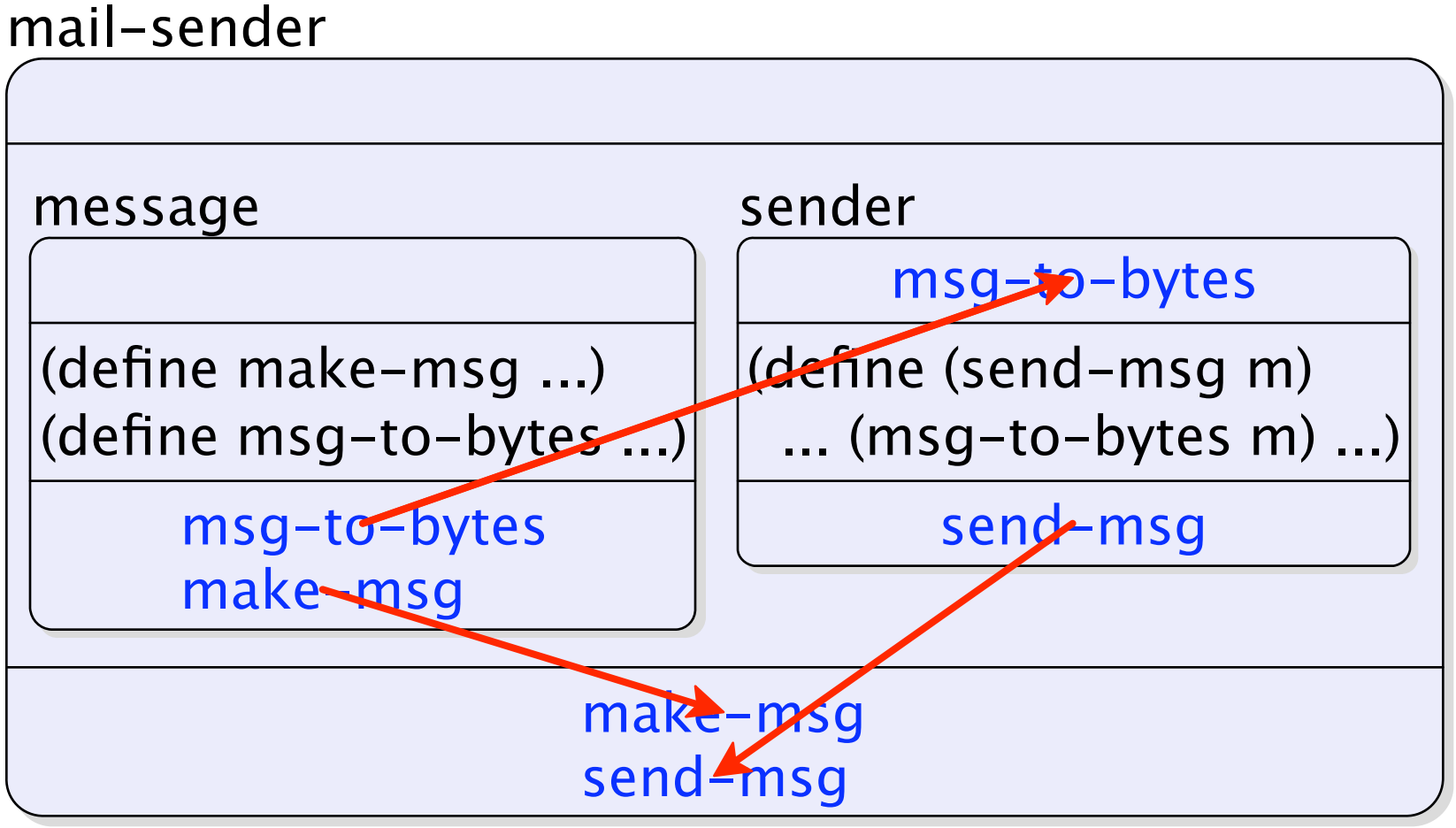
Other dynamic languages have similar features.


Name➡ sender
msg-to-bytes
(define (send-msg m)
... (msg-to-bytes m) ...)
send-msg
⬅Imports
⬅Body
⬅Exports

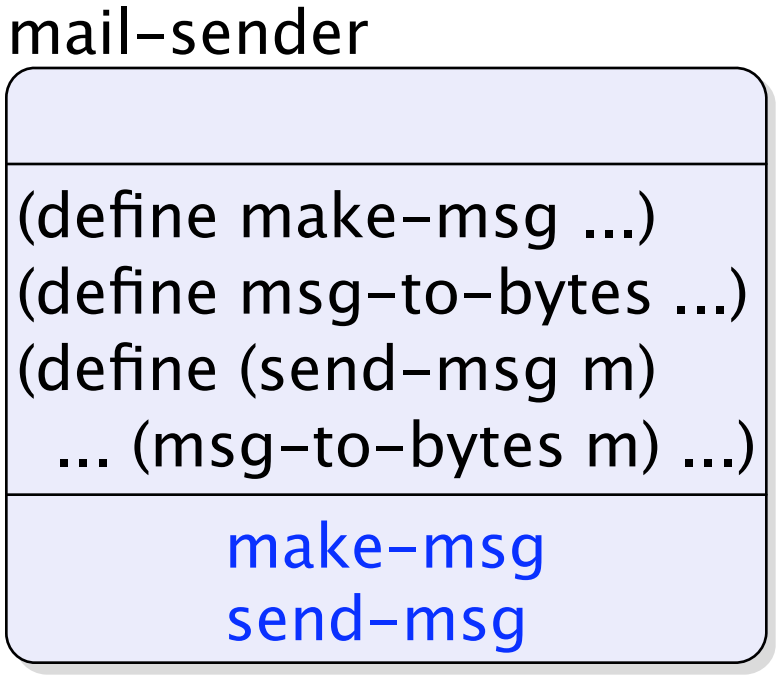
Linking Units



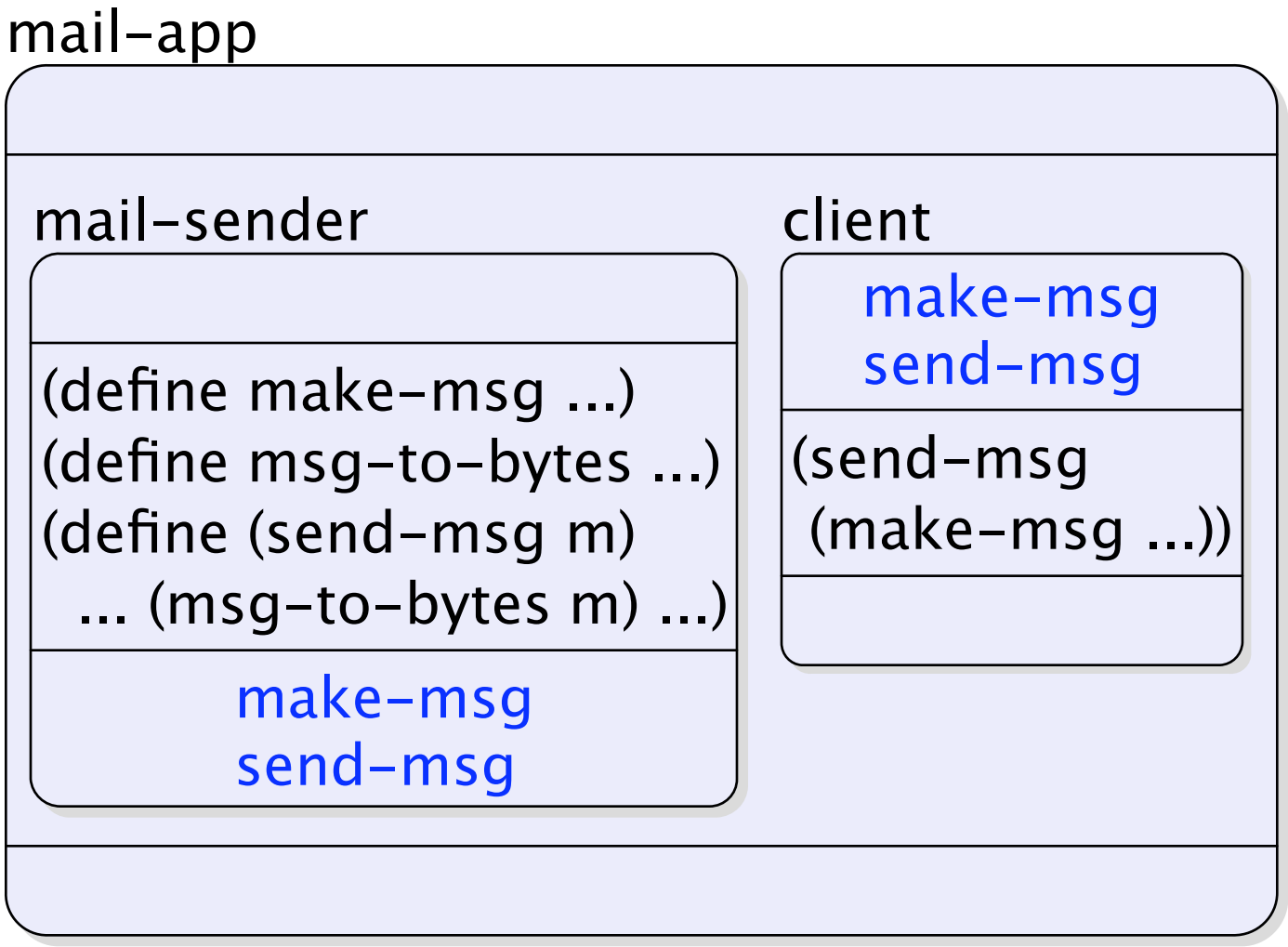
Linking Units



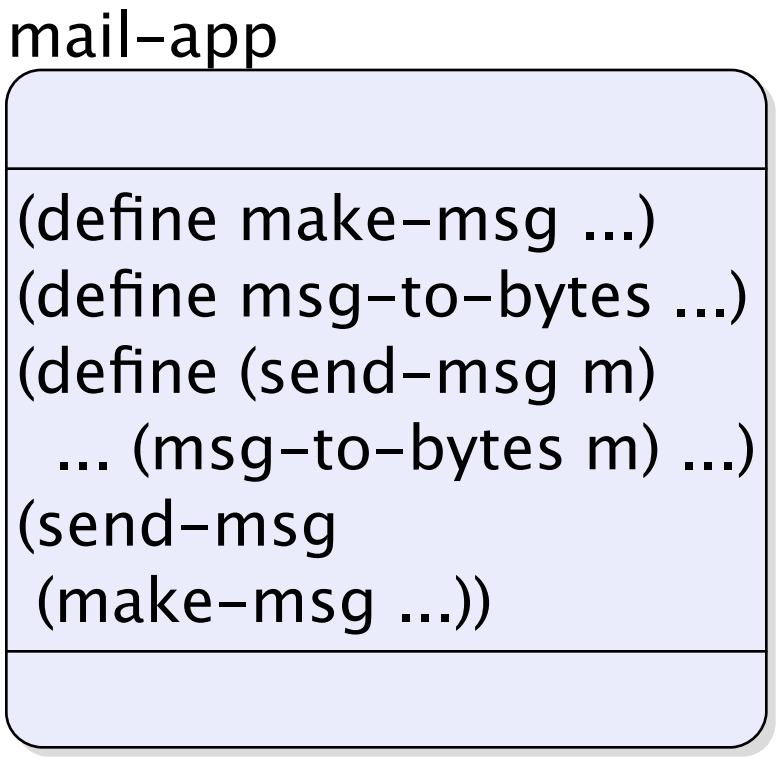
Linking Units



Linking Units

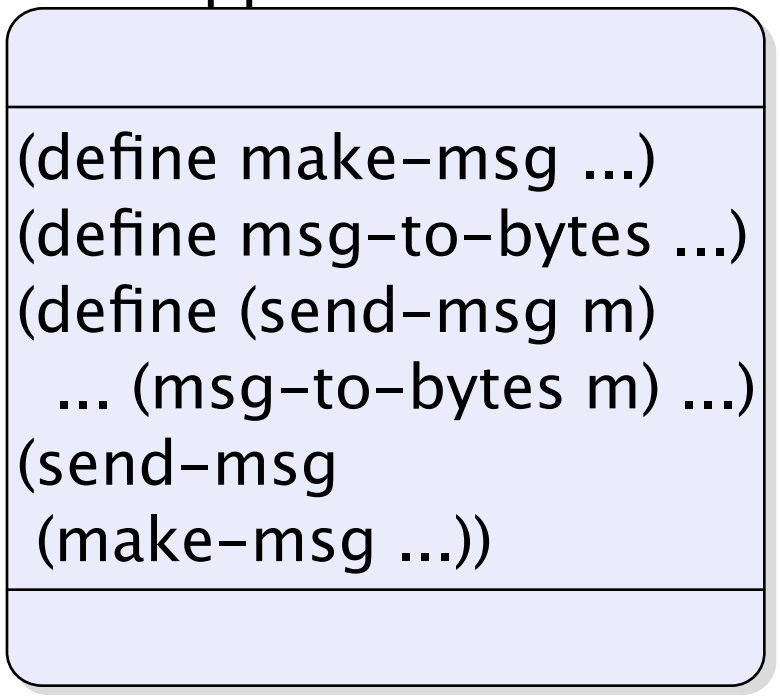


Linking Units

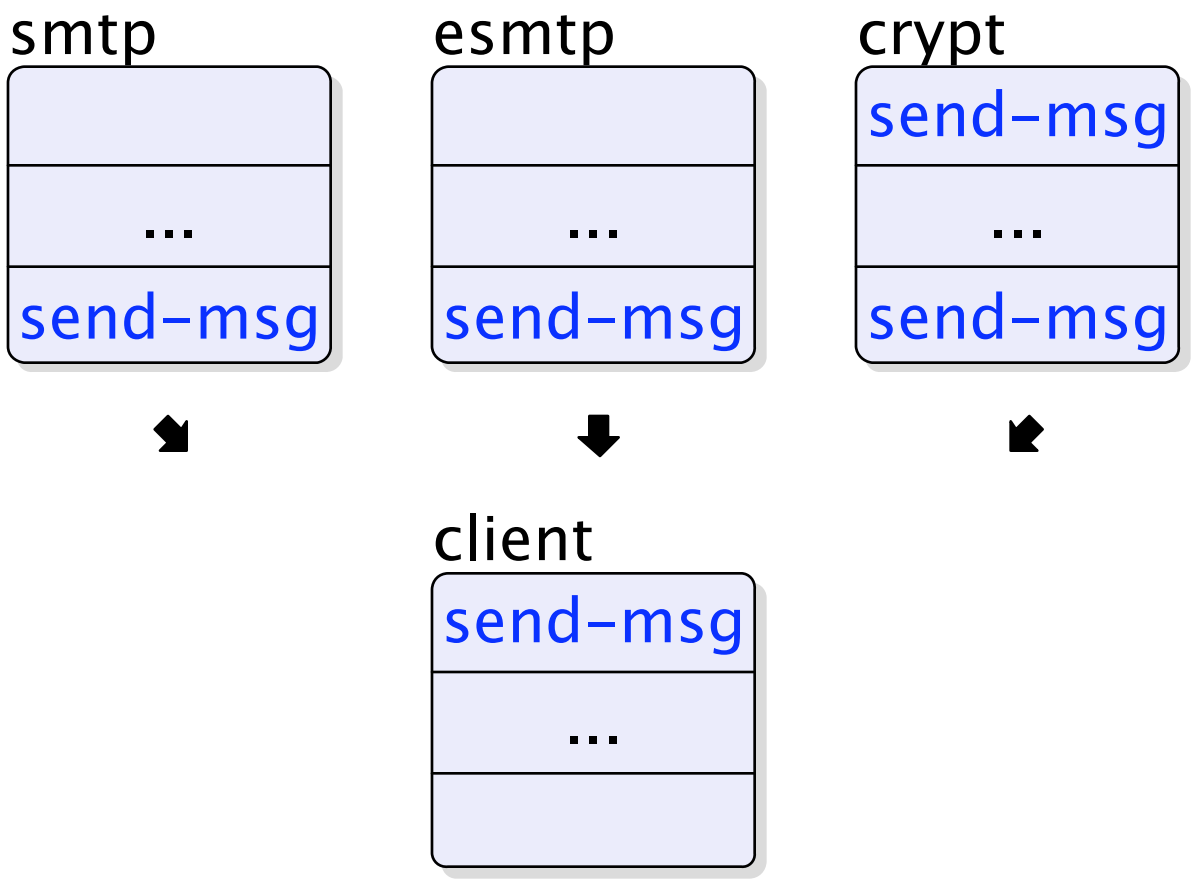


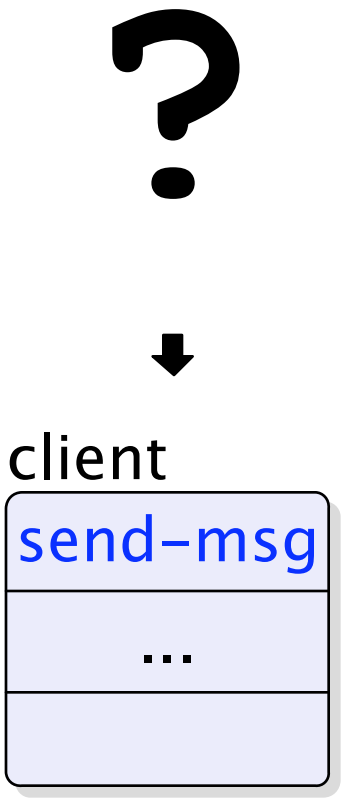
Invoking Units

mail-app



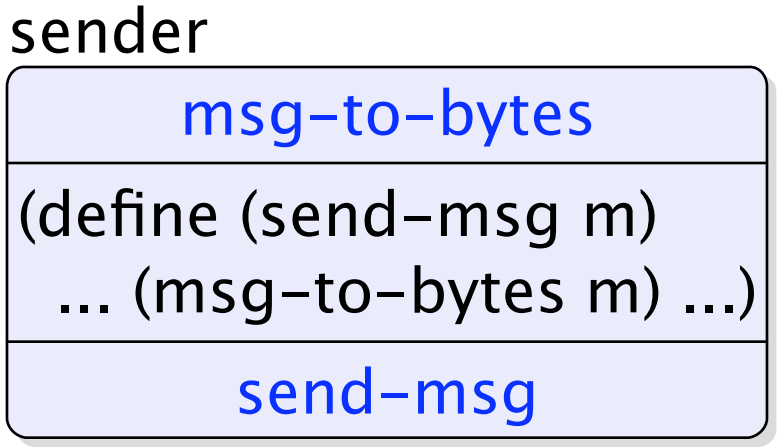
```
(let ()
  (define make-msg ...)
  (define msg-to-bytes ...)
  (define (send-msg m)
    ... (msg-to-bytes m) ...)
  (send-msg
   (make-msg ...)))
```


smtp
...
send-msg
esmtp
...
send-msg
crypt
send-msg
...
send-msg
client
send-msg
...

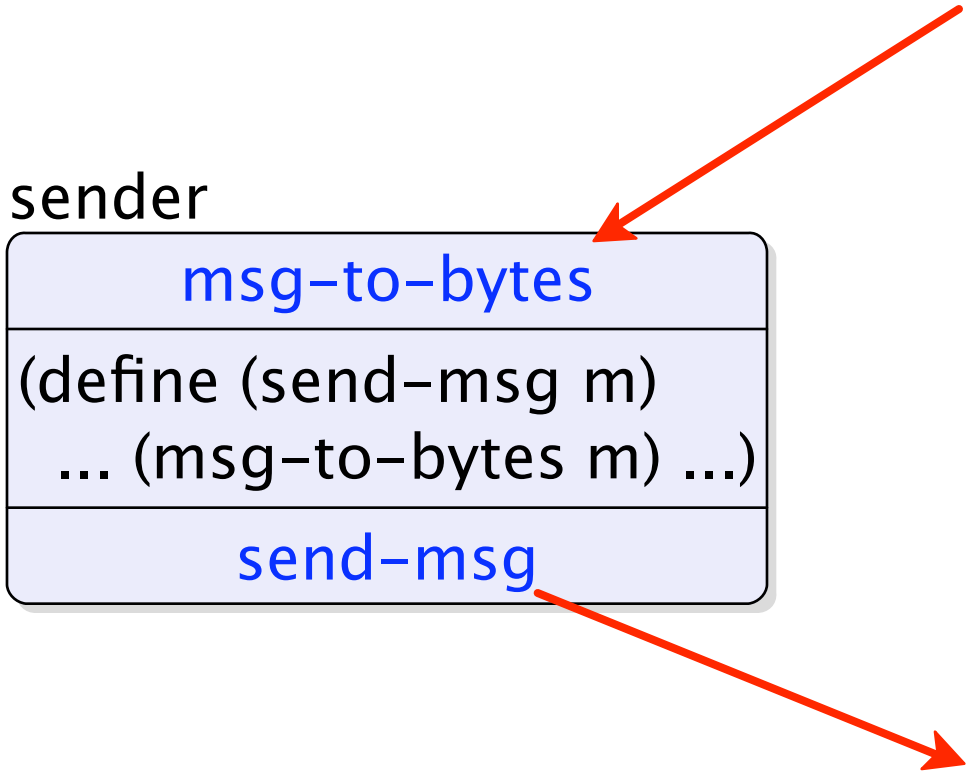

?
client
send-msg
...

Units with Contracts

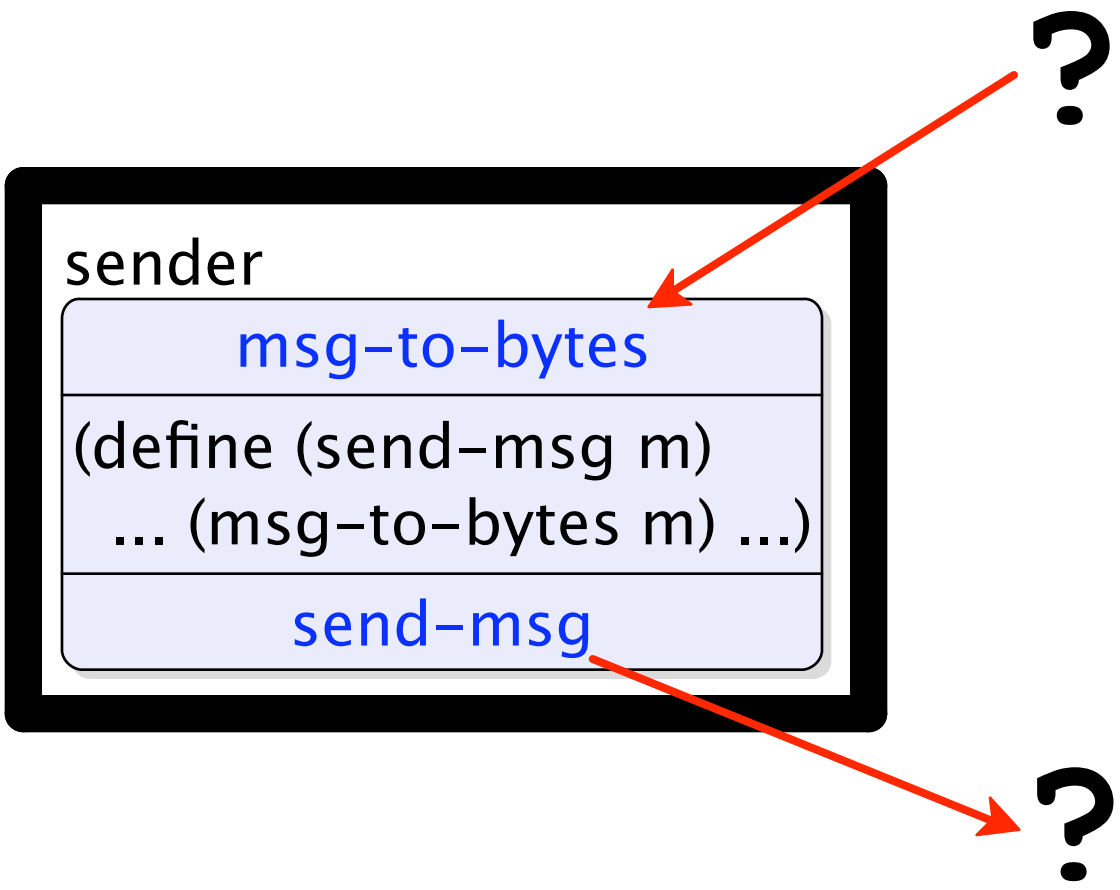
Dynamic Contract Boundaries

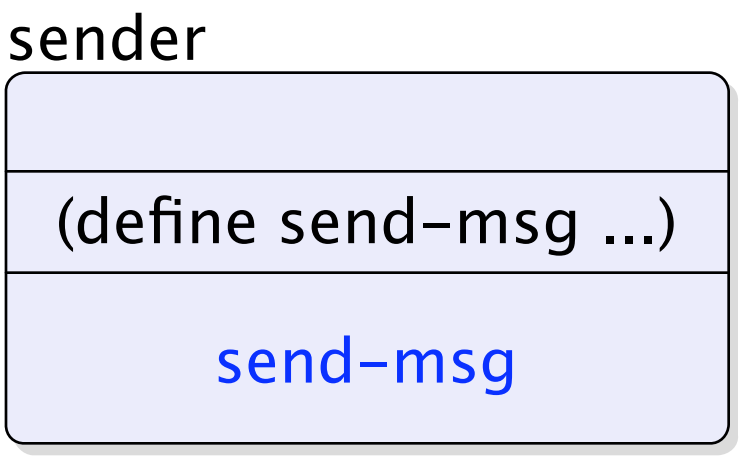


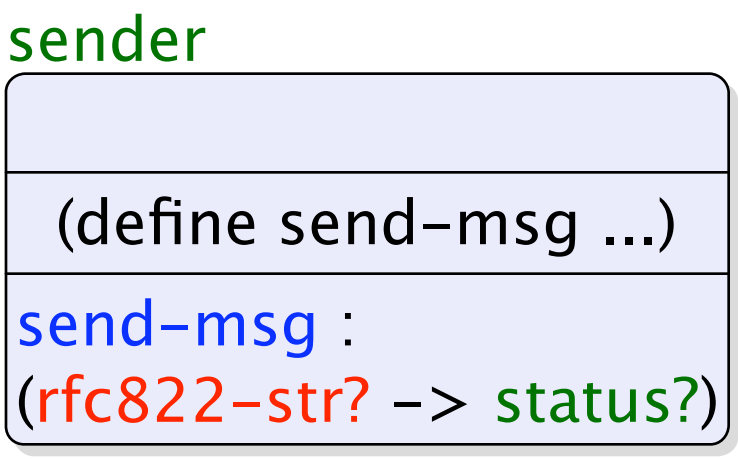
Dynamic Contract Boundaries

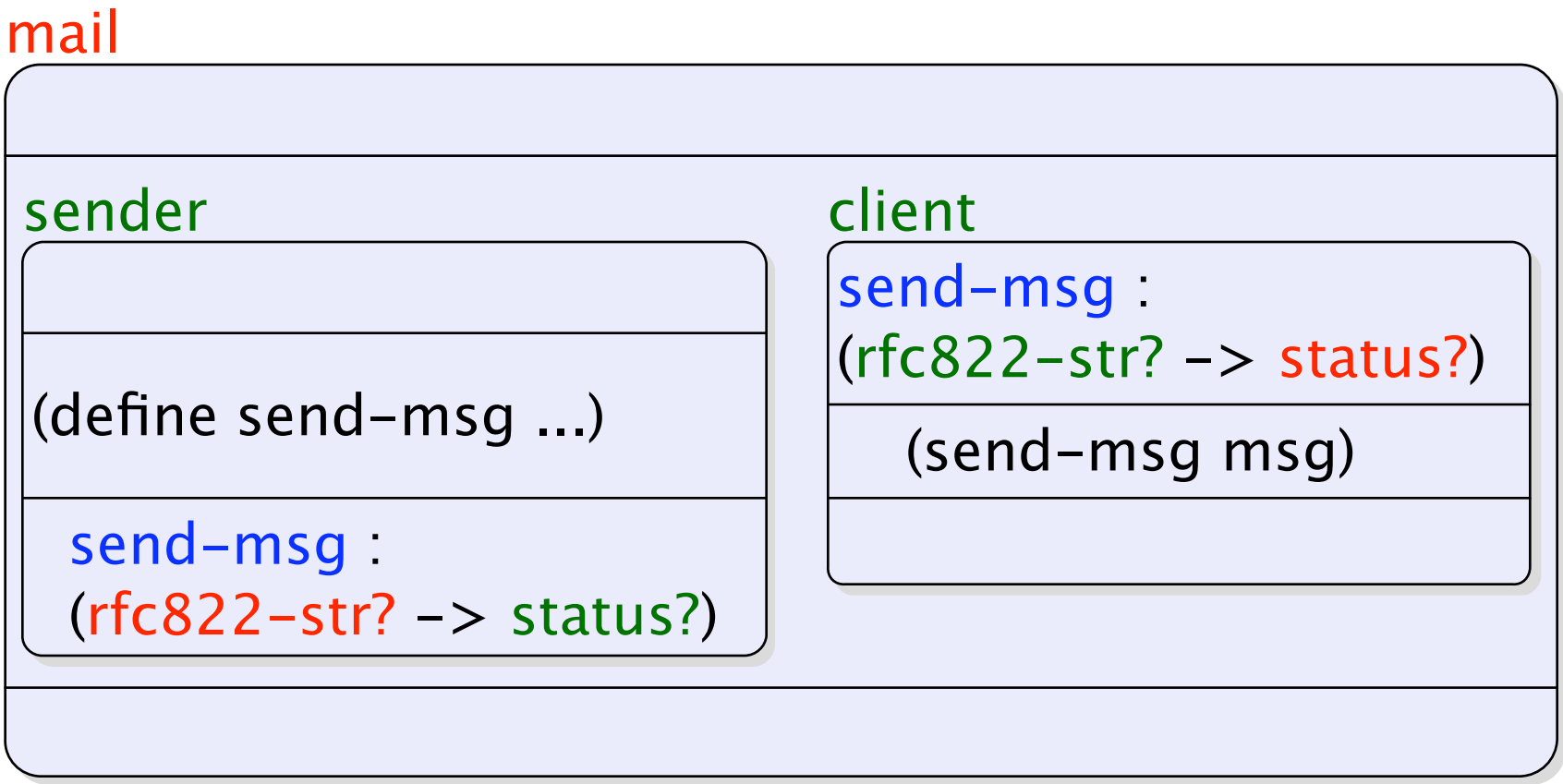


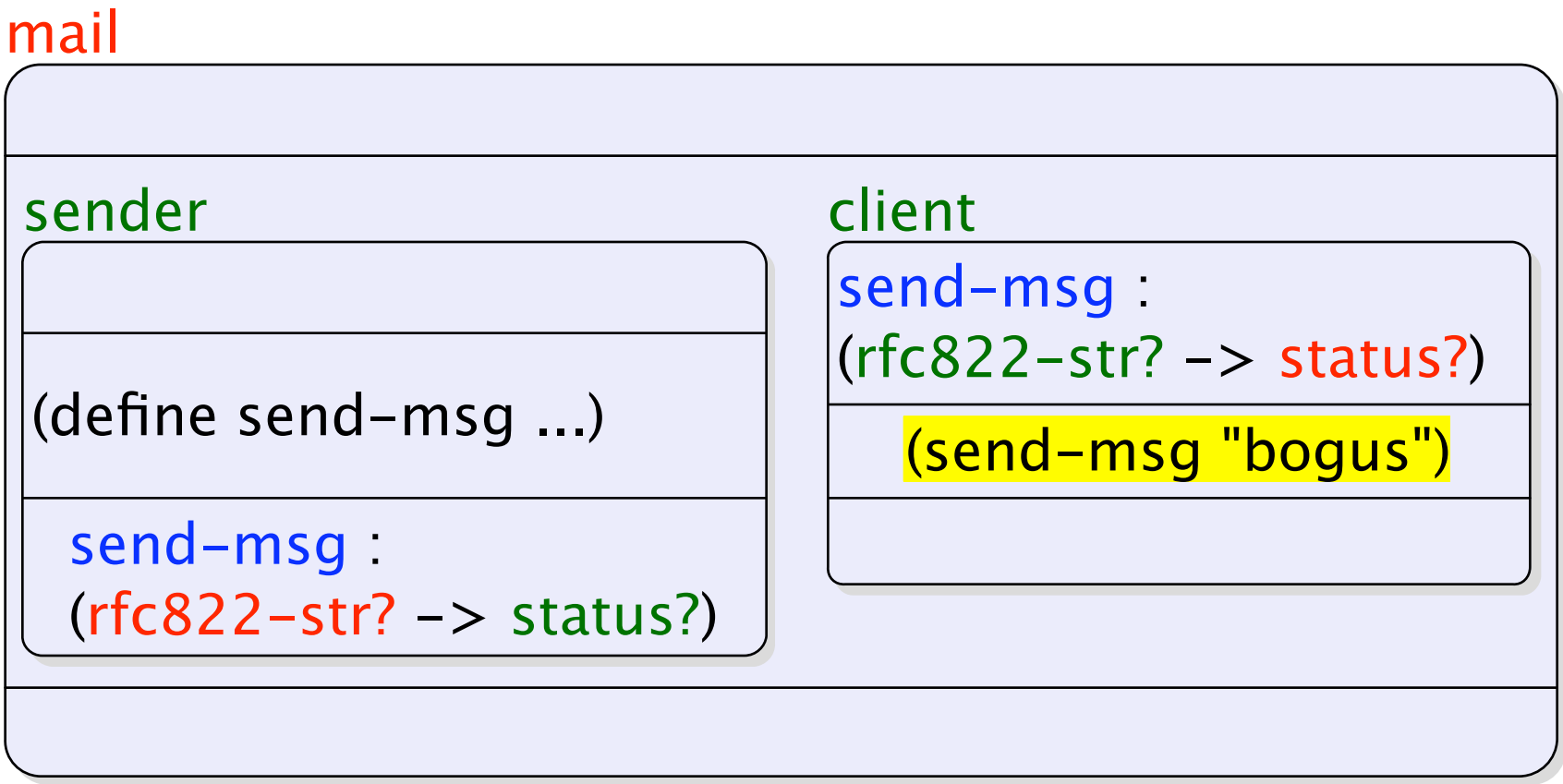
Dynamic Contract Boundaries

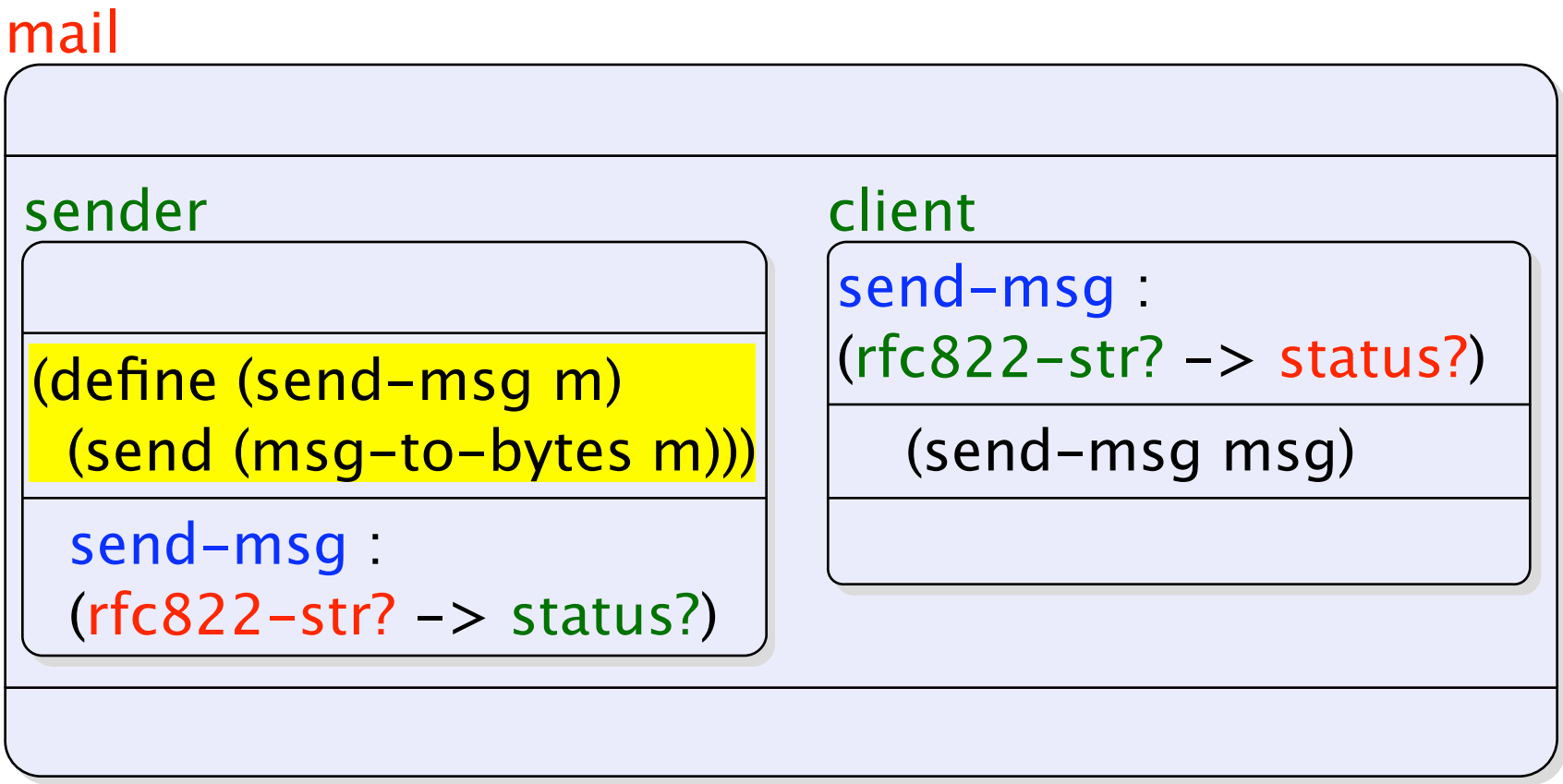


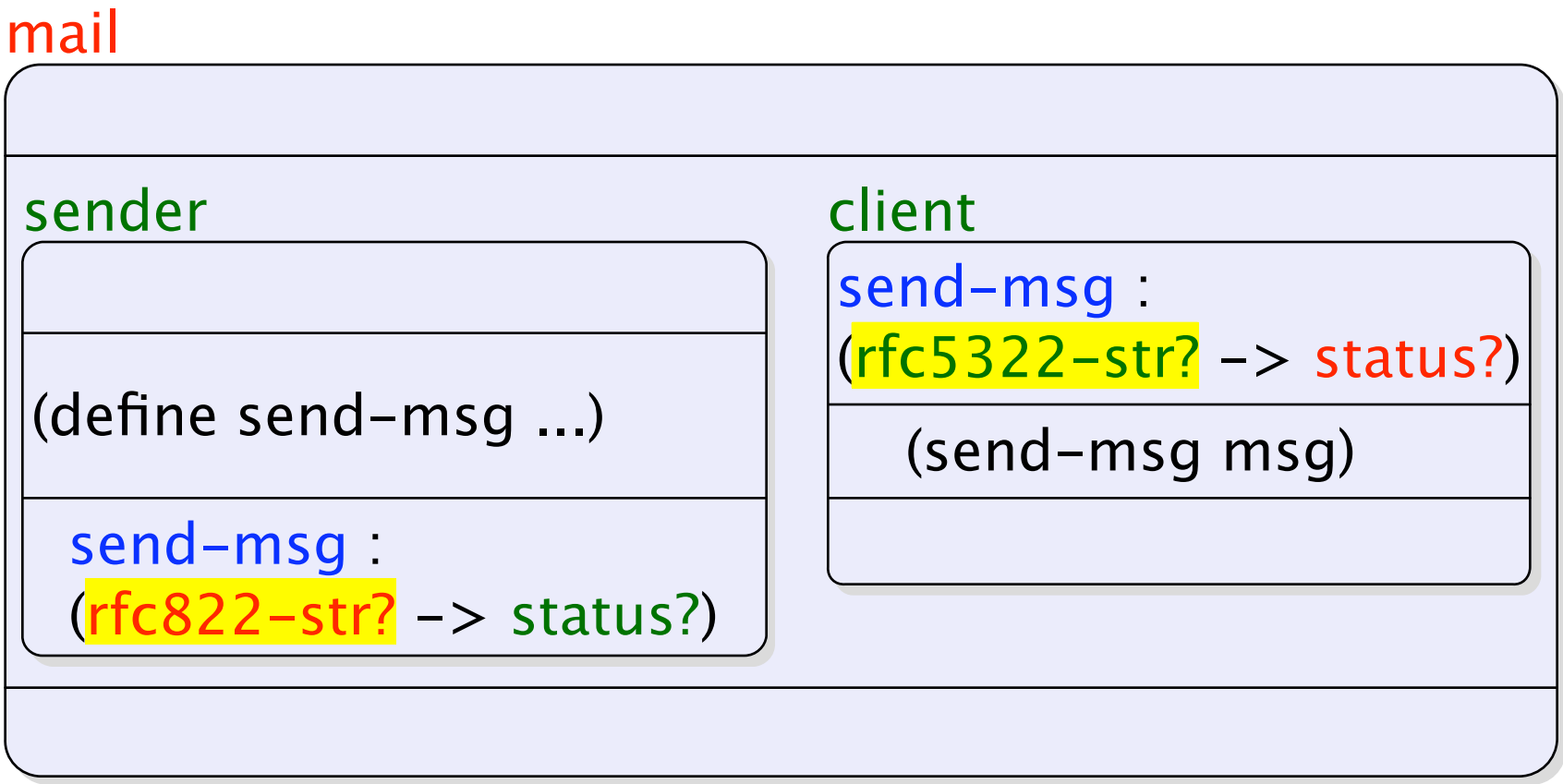

sender
(define send-msg ...)
send-msg

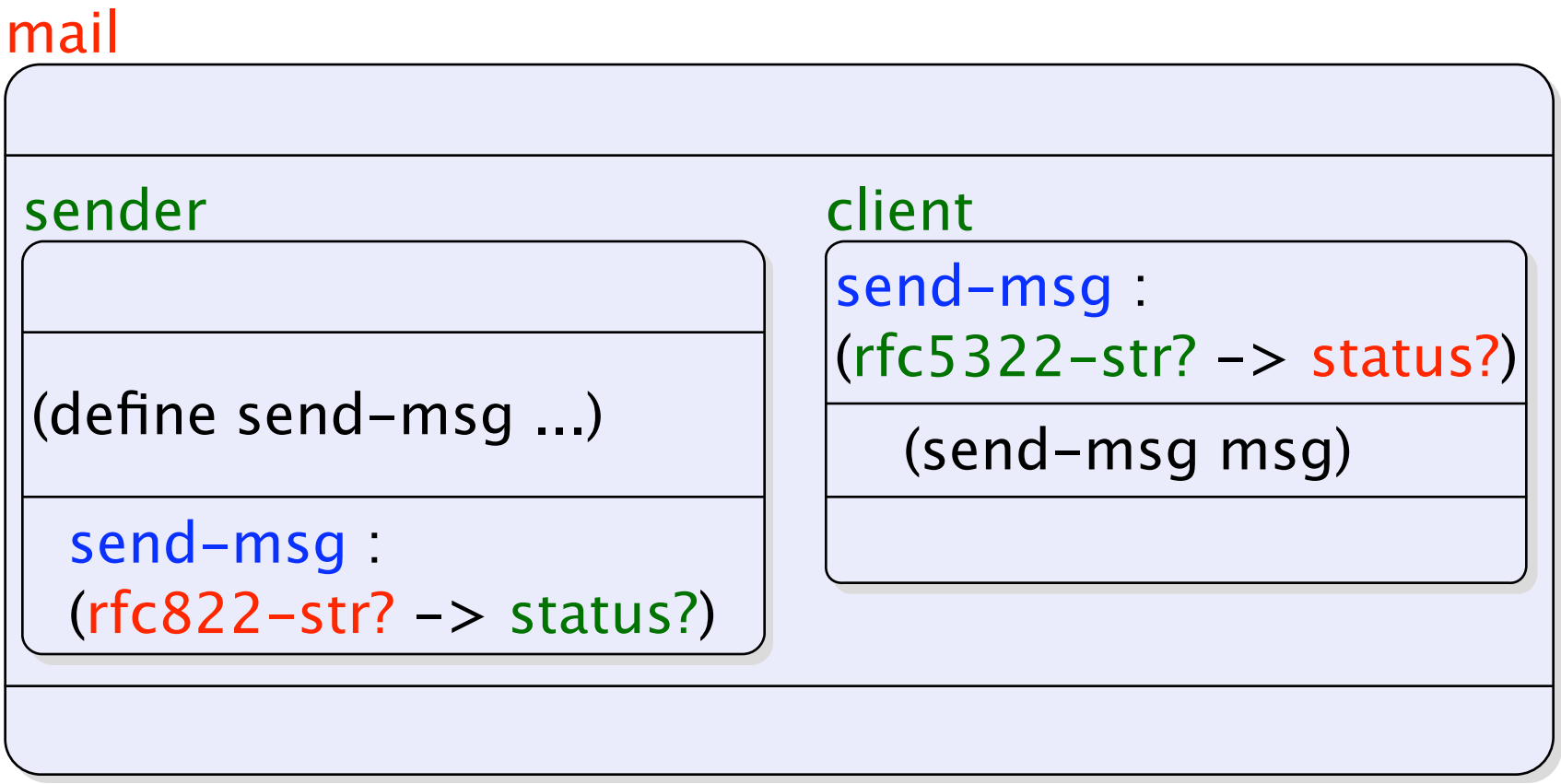

sender
(define send-msg ...)
send-msg :
(rfc822-str? -> status?)


mail
sender
(define send-msg ...)
send-msg :
(rfc822-str? -> status?)
client
send-msg :
(rfc822-str? -> status?)
(send-msg msg)

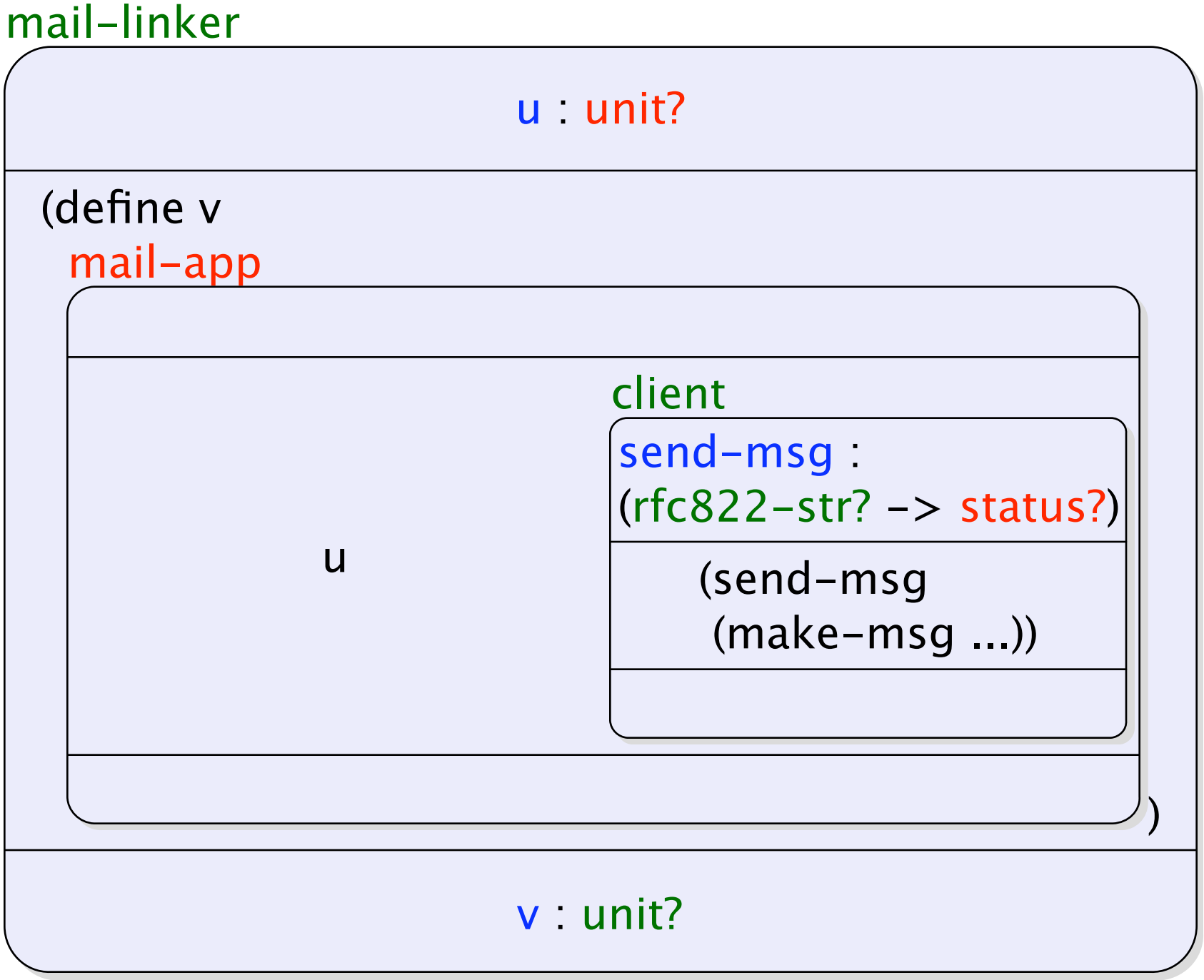

mail
sender
(define send-msg ...)
send-msg :
(rfc822-str? -> status?)
client
send-msg :
(rfc822-str? -> status?)
(send-msg "bogus")

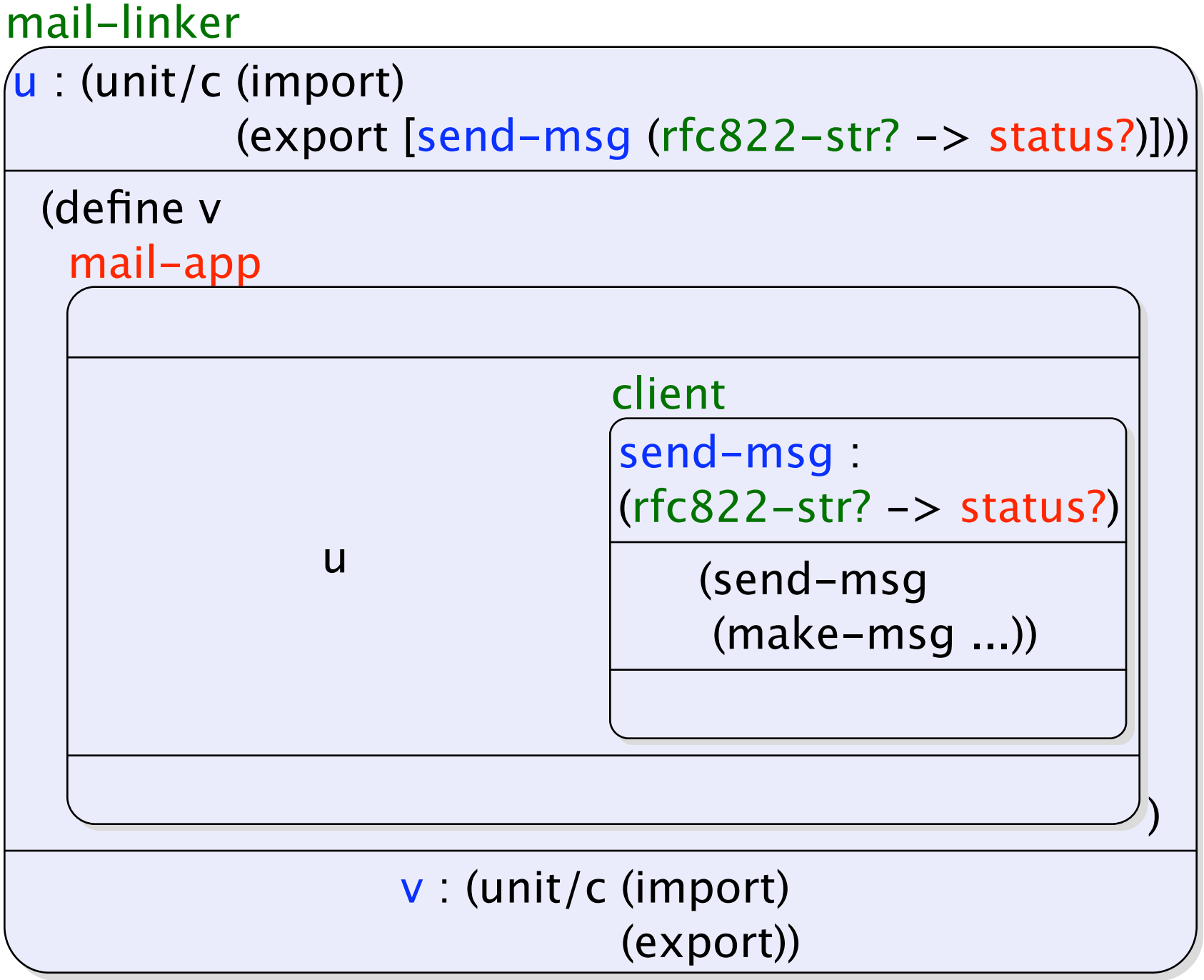

mail
sender
(define (send-msg m)
(send (msg-to-bytes m)))
send-msg :
(rfc822-str? -> status?)
client
send-msg :
(rfc822-str? -> status?)
(send-msg msg)


mail
sender
(define send-msg ...)
send-msg :
(rfc822-str? -> status?)
client
send-msg :
(rfc5322-str? -> status?)
(send-msg msg)



sender ⟷ linked ⟷ client


mail-linker
u : unit?
(define v
mail-app
client
send-msg :
(rfc822-str? -> status?)
u
(send-msg
(make-msg ...))
)
v : unit?


mail-linker
u : (unit/c (import)
(export [send-msg (rfc822-str? -> status?)]))
(define v
mail-app
client
send-msg :
(rfc822-str? -> status?)
u
(send-msg
(make-msg ...))
)
v : (unit/c (import)
(export))

Realizing Unit Contracts

Model

Started with a model for first-class modules.

Extended the model with contracts.

Implemented the model in PLT Redex * and tested it.

Proved contract behavior for the extended model.

Contracts should not change the meaning of valid programs.

* **`http://redex.plt-scheme.org`**

Implementation

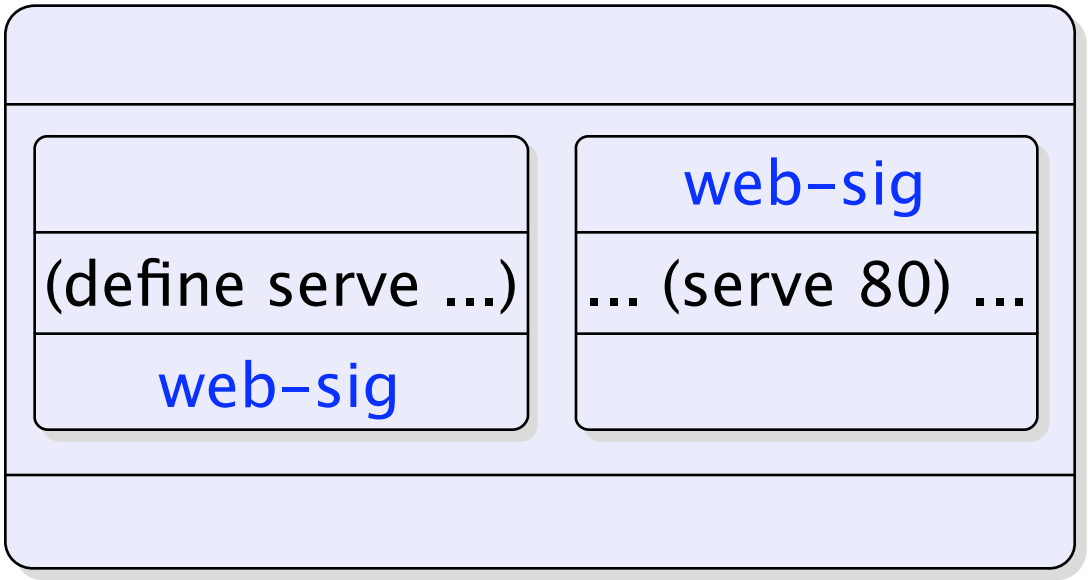
PLT Scheme's unit system is mostly nominal.

```
(define-signature web-sig (serve))
(define-signature mail-sig (serve))
```

Implementation

PLT Scheme's unit system is mostly nominal.

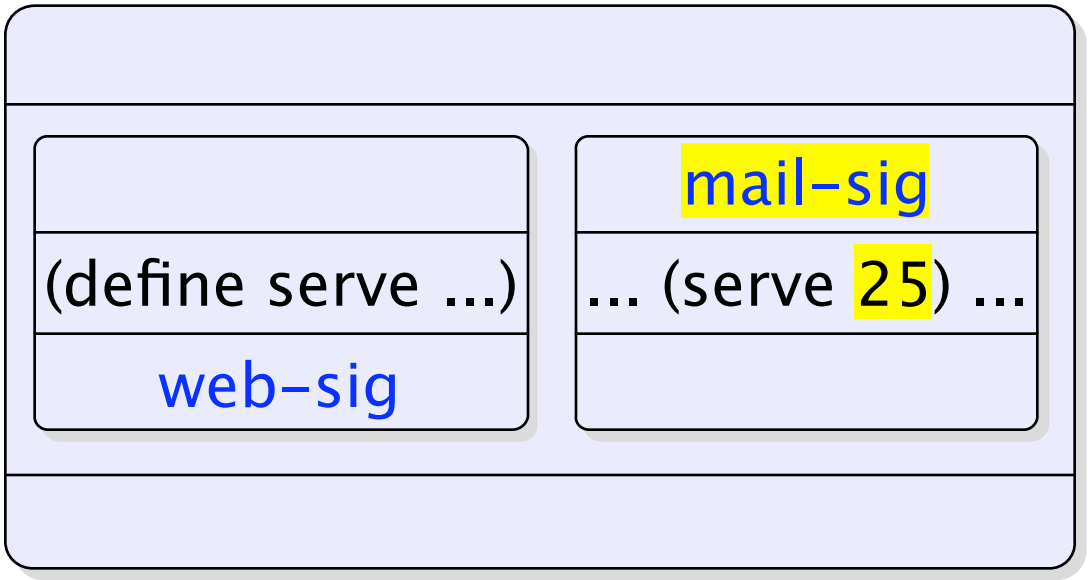
```
(define-signature web-sig (serve))
(define-signature mail-sig (serve))
```



Implementation

PLT Scheme's unit system is mostly nominal.

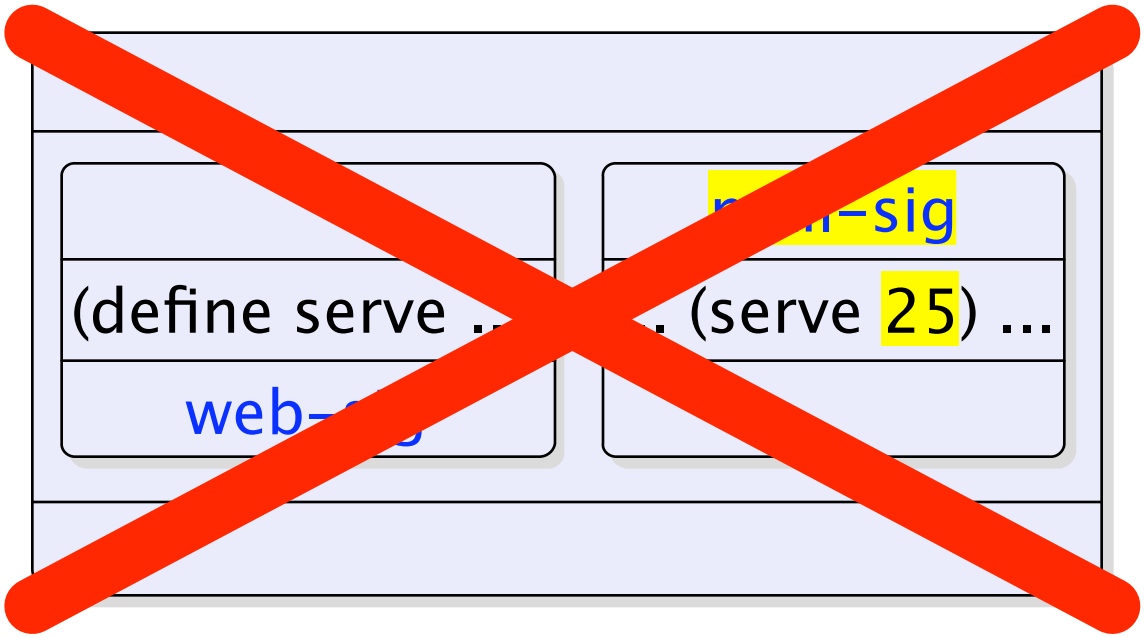
```
(define-signature web-sig (serve))
(define-signature mail-sig (serve))
```



Implementation

PLT Scheme's unit system is mostly nominal.

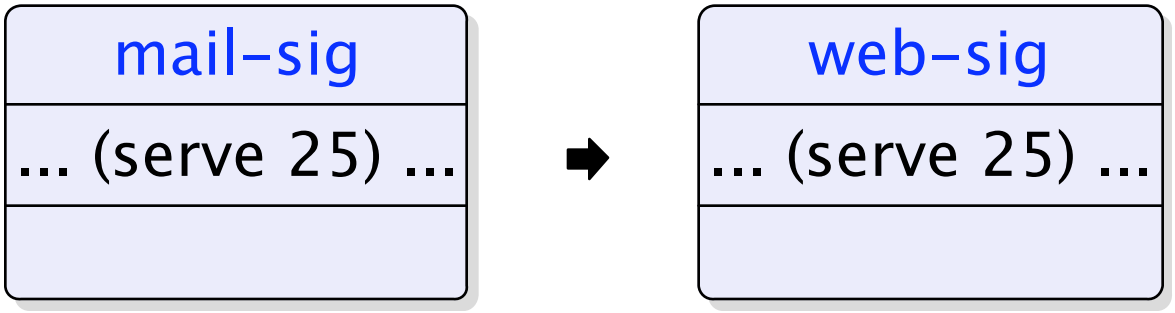
```
(define-signature web-sig (serve))
(define-signature mail-sig (serve))
```



Implementation

However, it has some structural features.

```
(define-signature web-sig (serve))
(define-signature mail-sig (serve))
```



Also, the **`unit/c`** contract combinator is inherently structural.

Conclusion

Blame tracking helps pinpoint locations of errors.

Dynamic features do not make blame tracking impossible.

PLT Scheme now has contracts for first-class modules.

`http://www.plt-scheme.org`