the racket manifesto

matthias, racketeer

26 January 1995

What is Racket?

Haskell is a purely functional, lazy language.

Python is about the one way, the obvious way.

So what about Racket?

racket is a
programming-
language
programming
language

The next 700 languages?

The next 7,000 languages?

The next 70,000 languages?

The next 700,000 languages?

Why many languages?
Isn't Racket enough?

Imagine Conference Management

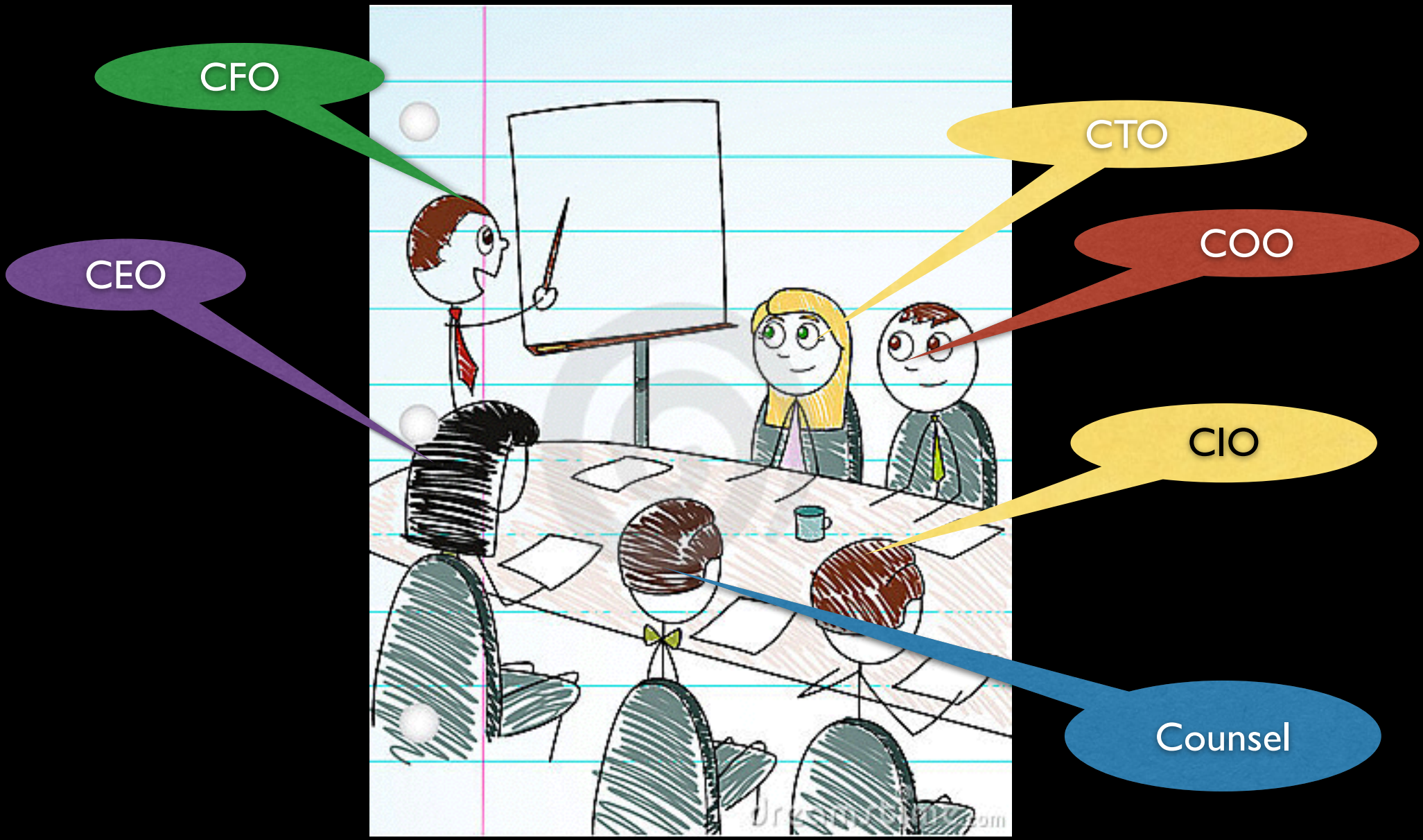
Problems to be solved:

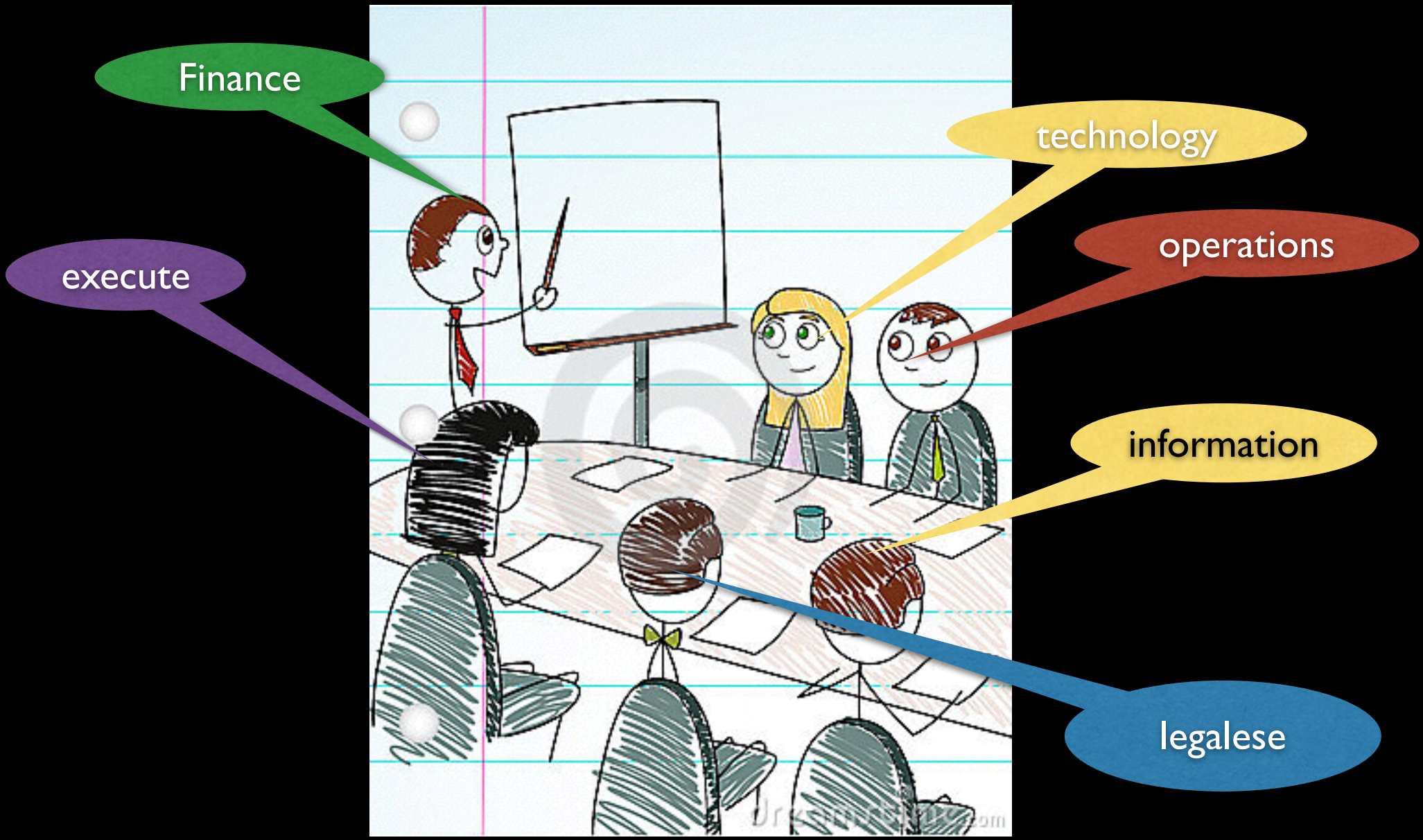
- database of people & roles
- security policies
- list of papers
- paper-reviewer mapping
- review policies
- …

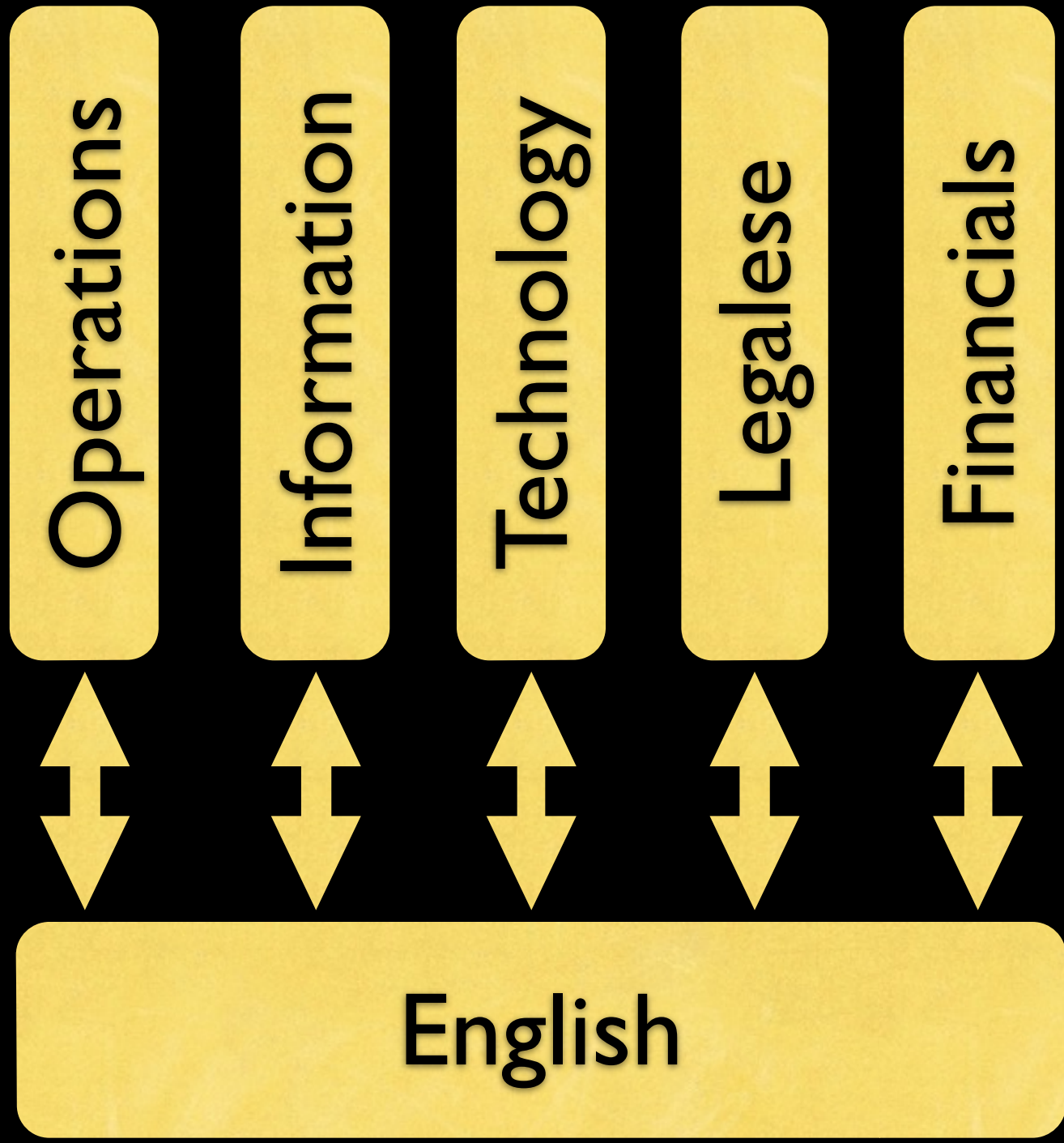
Features supported:

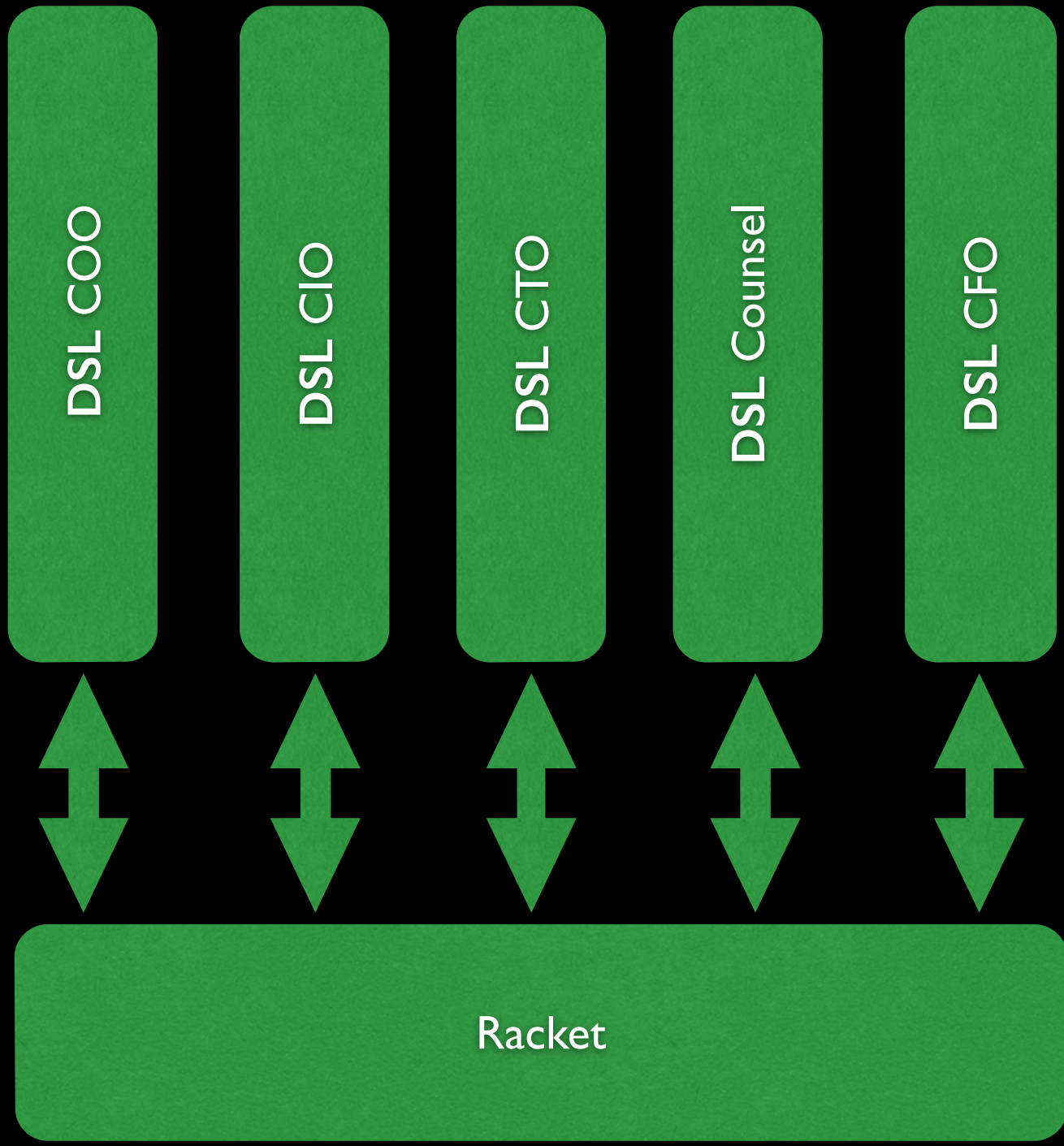
- `for` and `while` loops
- methods
- classes
- modules
- packages

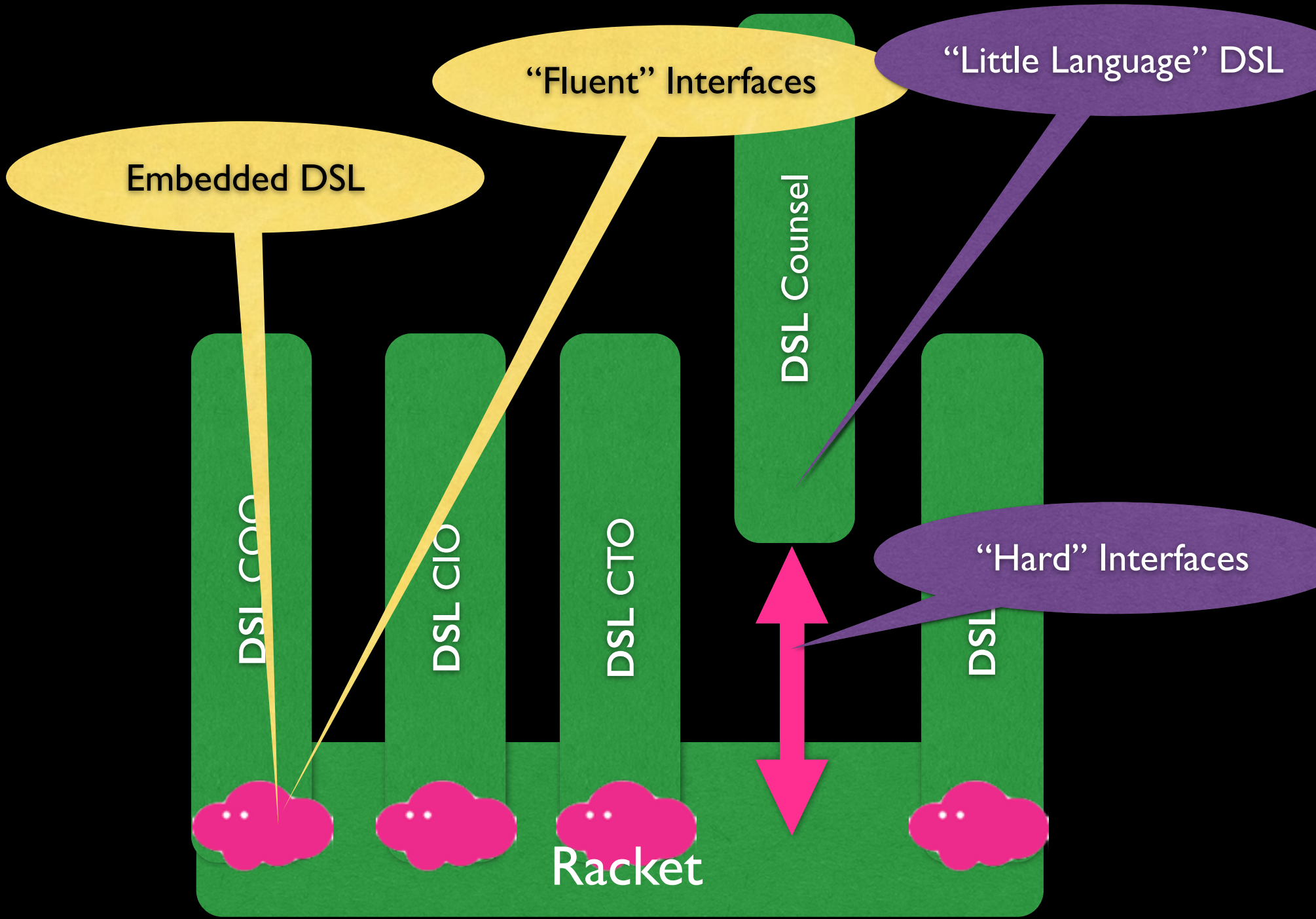
People don’t speak one English.
They speak many.


CFO
CTO
COO
CEO
CIO
Counsel


Finance
technology
operations
execute
information
legalese


Operations
Information
Technology
Legalese
Financials
English


DSL COO
DSL CIO
DSL CTO
DSL Counsel
DSL CFO
Racket


"Fluent" Interfaces
"Little Language" DSL
Embedded DSL
DSL Counsel
DSL COO
DSL CIO
DSL CTO
DSL
"Hard" Interfaces
Racket

How do you build these “DSLs”?

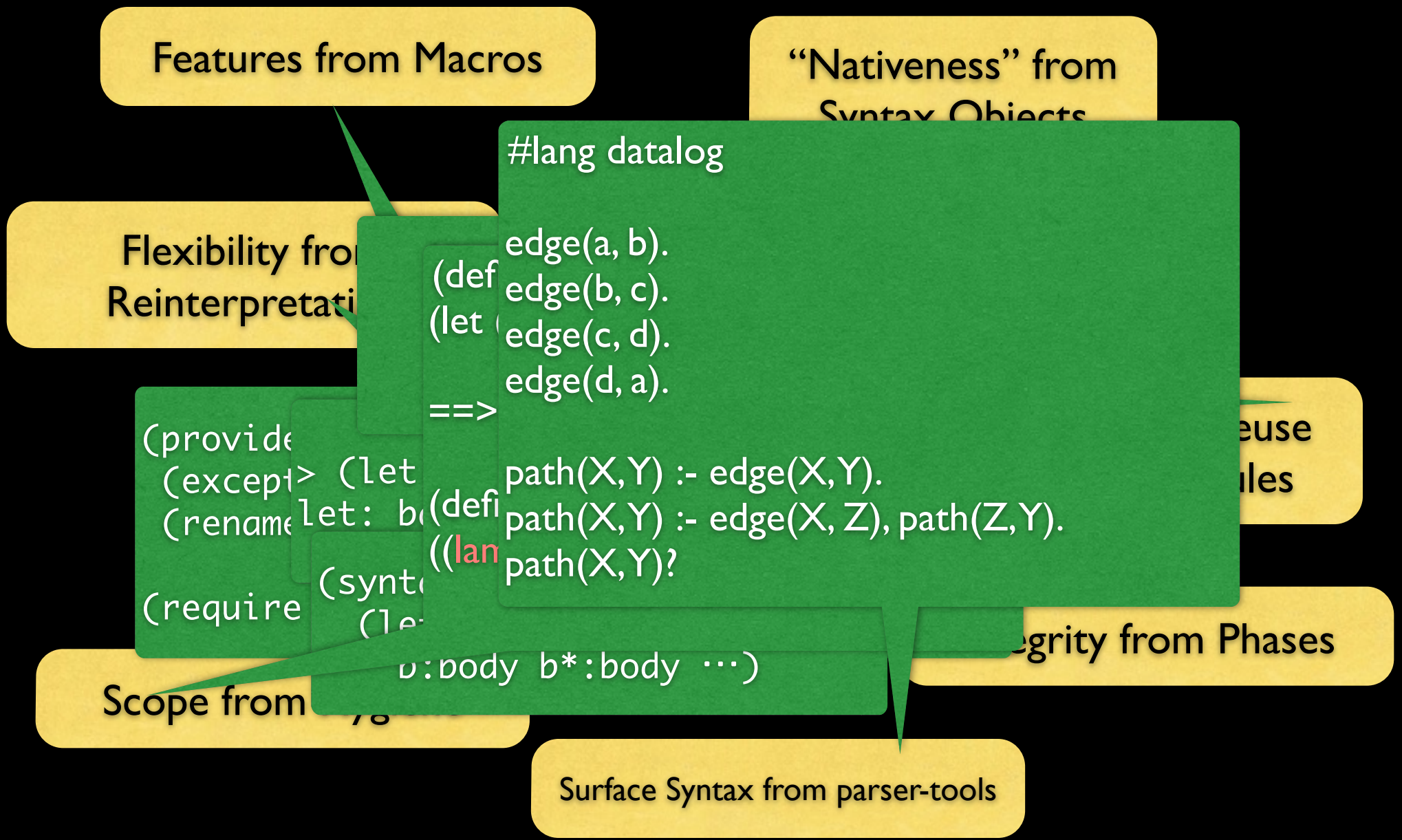

Hygienic
Macros!

```
#lang setup/infotab
```

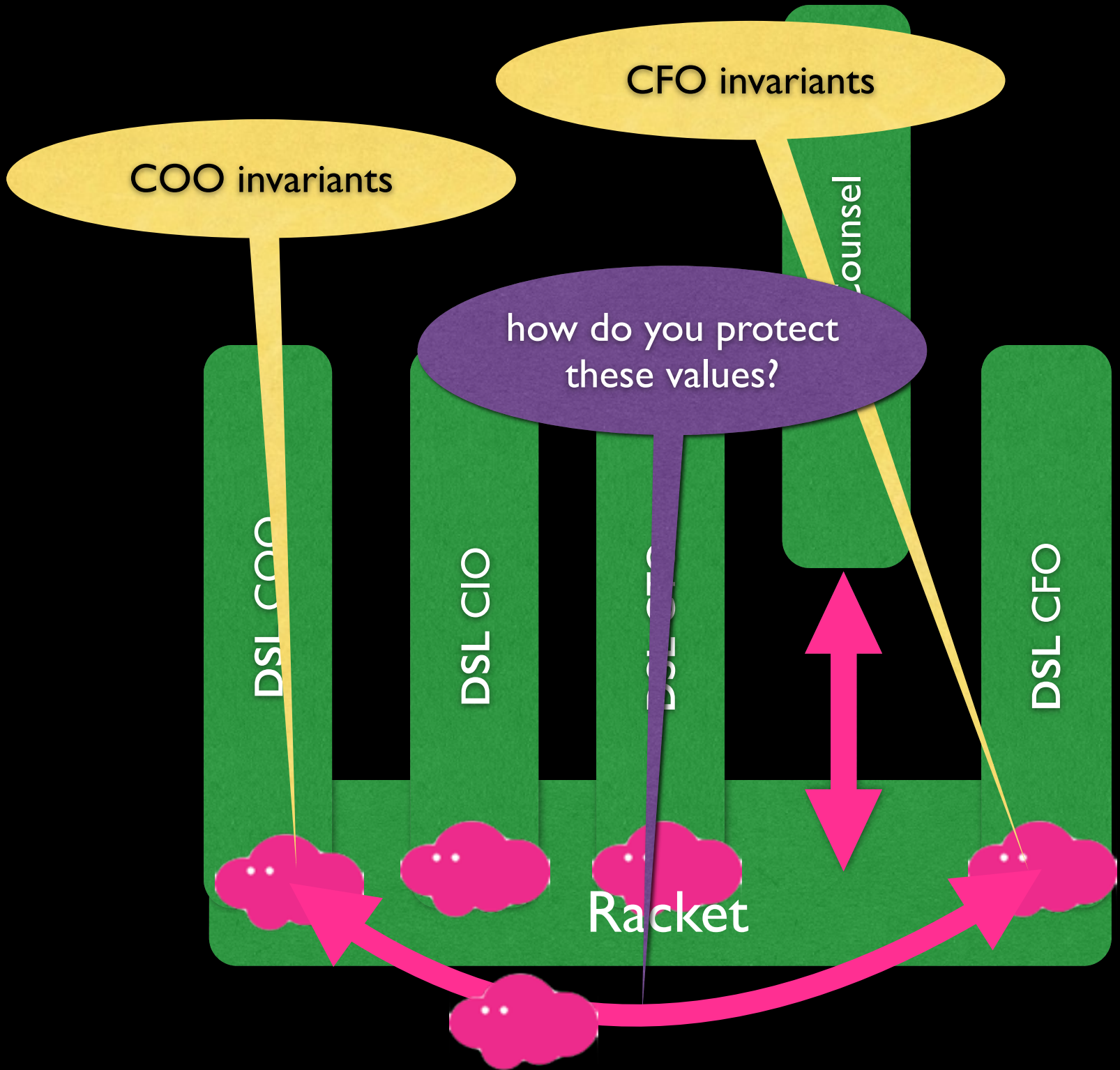
```
#lang scribble/manual
```

```
#lang redex
```

(Embedded) DSL Compilers



How do you *safely*
compose components
in different DSLs?

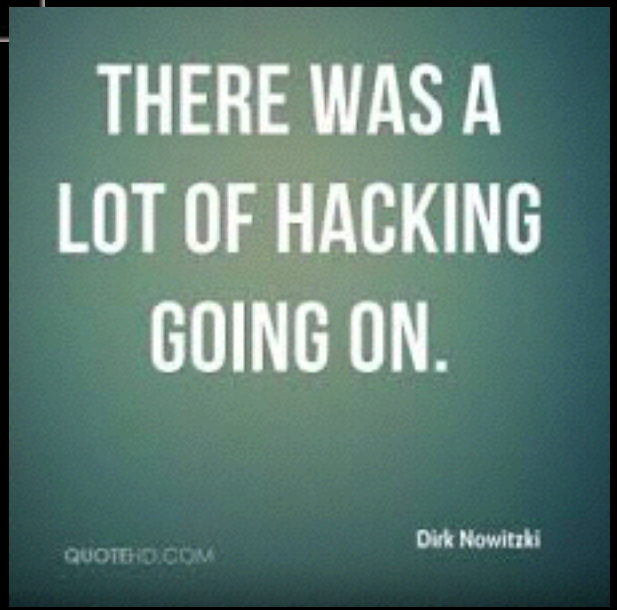

CFO invariants
COO invariants
how do you protect these values?
DSL COO
DSL CIO
DSL CFO
Racket


HOPE
FOR
THE
BEST


Good
Luck


DON'T DO IT.


think
positive!!


THERE WAS A
LOT OF HACKING
GOING ON.
QUOTEHD.COM
Dirk Nowitzki

Composing DSL Components

```
#lang racket

(provide
  (contract-out
    (open
      ;; pops up a currently invisible area
      (-> (and/c window? invisible?)
          (and/c window? top-level?)))))
```

inspectors and code control

ts and event spaces

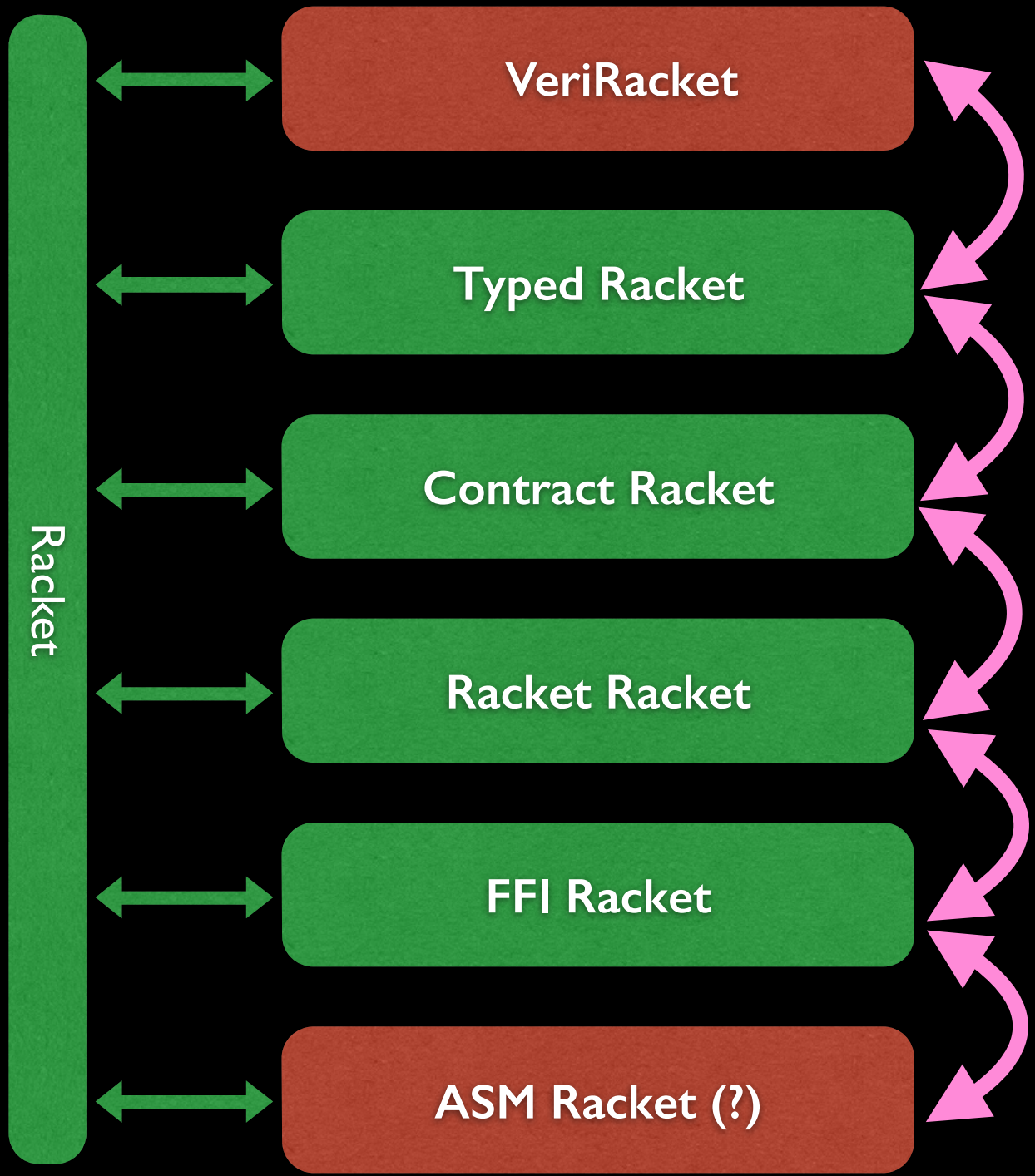
impersonators, chaperones, and contracts

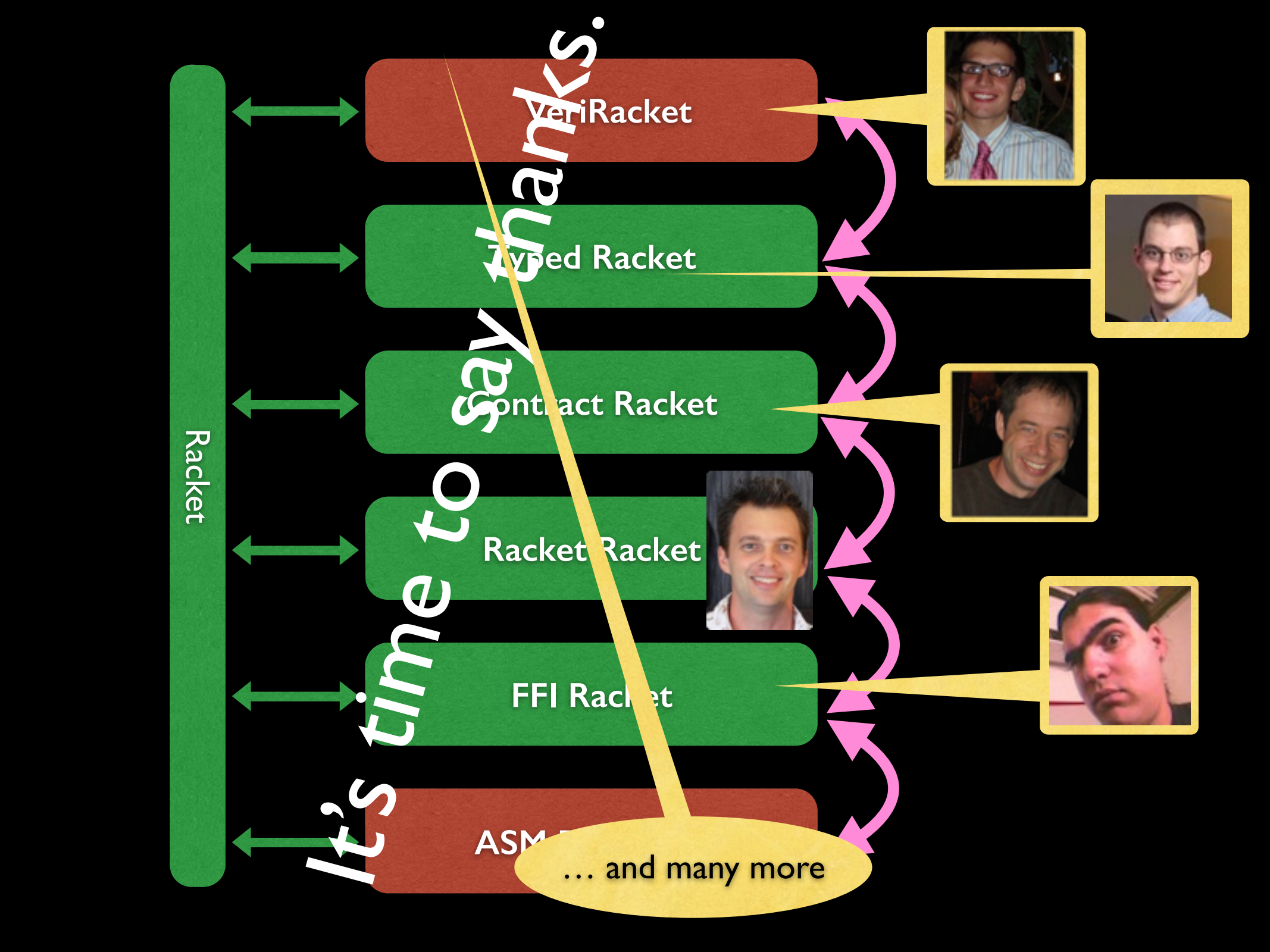
sandboxes and access

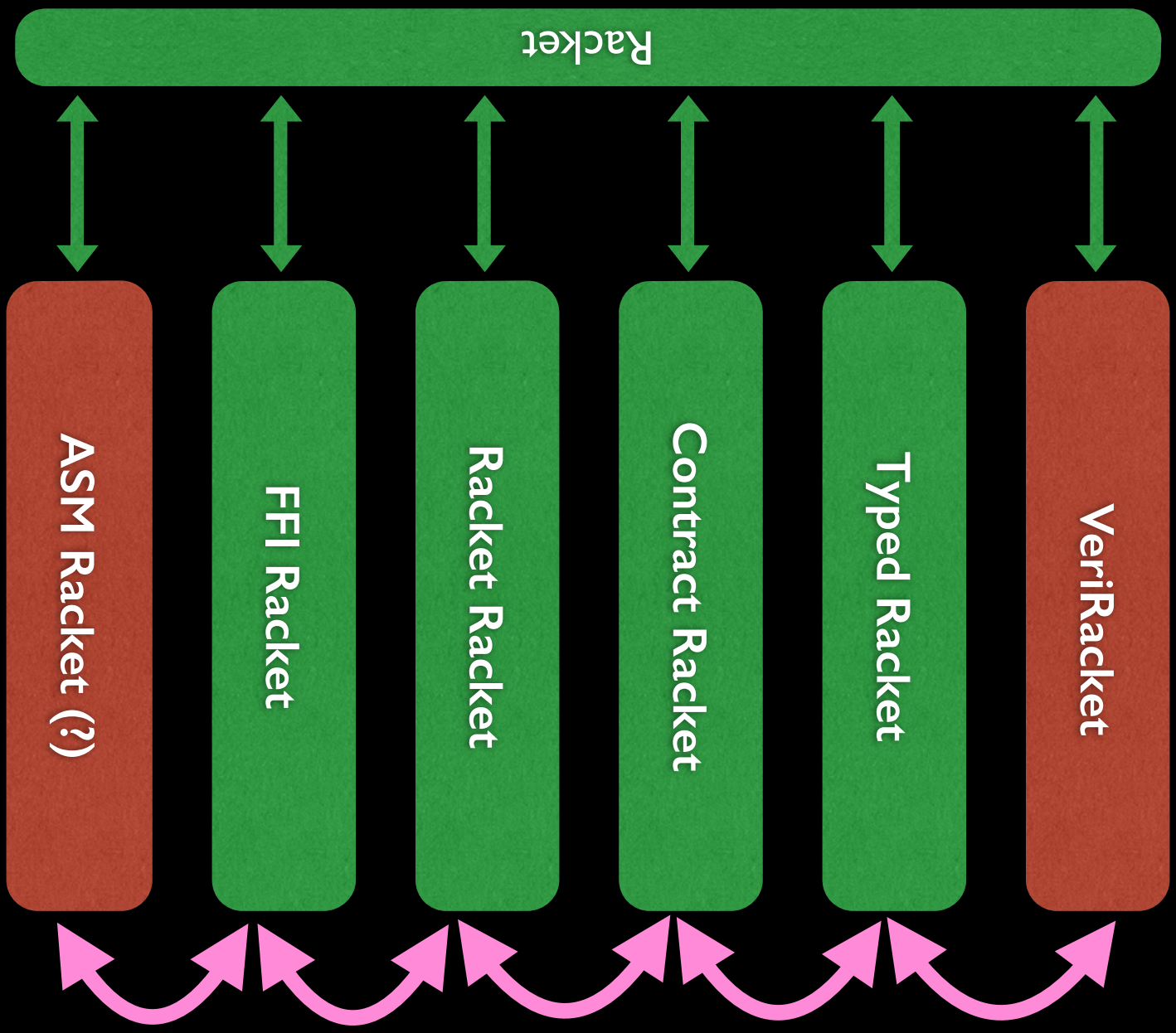
wills and executors

Are all DSL problems solved?

racket is a full-spectrum programming language


Racket
VeriRacket
Typed Racket
Contract Racket
Racket Racket
FFI Racket
ASM Racket (?)


Racket
VeriRacket
Typed Racket
Contract Racket
Racket Racket
FFI Racket
ASM
… and many more
It's time to say thanks.



full-*stack* language

```
#lang racket

(provide
 ;; Image Number Number Image -> Image
 ;; (place obj x y bg) puts obj at (x,y) on bg
 place)

...


(define (place obj x y background)
  (define width (width background))
  (define hight (height background))
  (unless (and (<= 0 x) (< x width))
    (error 'place "bad x"))
  (unless (and (<= 0 y) (< x hight))
    (error 'place "bad y"))
  (place-proper obj x y background))
```

```
#lang racket

(provide
 (contract-out

  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg
  (place
   (->i ((obj image?)
         (x (bg) (and/c (>=/c 0) (</c (width bg))))
         (y (bg) (and/c (>=/c 0) (</c (height bg))))
         (bg image?))
        (result image?)))))
...

(define (place obj x y background)
 (place-proper obj x y background))
```

```
#lang racket

(provide
 (contract-out

  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg
  (place
   (->i ((obj image?)
         (x (bg) (and/c (>=/c 0) (</c (width bg))))
         (y (bg) (and/c (>=/c 0) (</c (height bg))))
         (bg image?))
        (result image?)))))
...

(define (place obj x y background)
 (place-proper obj x y background))
```

```
#lang typed/racket

(provide
 (contract-out

  ;; (place obj x y bg) puts obj at (x,y) on bg
  (place
   (->i ((obj image?)
         (x (bg) (and/c (>=/c 0) (</c (width bg))))
         (y (bg) (and/c (>=/c 0) (</c (height bg))))
         (bg image?))
        (result image?)))))
...

(: place (-> Image Number Number Image Image))
(define (place obj x y background)
 (place-proper obj x y background))
```

Brian LaChance

```
#lang typed/racket

(provide
 (contract-out

  ;; (place obj x y bg) puts obj at (x,y) on bg
  (place
   (->i ((obj image?)
         (x (bg) (and/c (>=/c 0) (</c (width bg))))
         (y (bg) (and/c (>=/c 0) (</c (height bg))))
         (bg image?))
        (result image?)))))
...

(: place (-> Image Number Number Image Image))
(define (place obj x y background)
 (place-proper obj x y background))
```

```
#lang typed/racket

(define-signature Server%
  ;; (place obj x y bg) puts obj at (x,y) on bg
  ([place : (-> Image Number Number Image)]))

(define-type Server@
  (Unit
   (import Server%)
   (export Server%)
   Boolean))
```

```
#lang dt/racket

(provide
  ;; (place obj x                                  bg
  place)

...
```

```
(: place (-> Image Number Number Image Image)
 suchthat
 (->i ((obj image?)
       (x (bg) (and/c (>=/c 0) (</c (width bg))))
       (y (bg) (and/c (>=/c 0) (</c (height bg))))
       (bg image?))
      (result image?)))

(define (place obj x y background)
 (place-proper obj x y background))
```

And we can also go in the other direction.

```
#lang racket

(provide
 ;; Image Number Number Image -> Image
 ;; (place obj x y bg) puts obj at (x,y) on bg
 place)

...


(define (place obj x y background)
  (define width (width background))
  (define hight (height background))
  (unless (and (<= 0 x) (< x width))
    (error 'place "bad x"))
  (unless (and (<= 0 y) (< x hight))
    (error 'place "bad y"))
  (place-proper obj x y background))
```

```
#lang racket

(provide
 ;; Image Number Number Image -> Image
 ;; (place obj x y bg) puts obj at (x,y) on bg
 place)

...

(define (place obj x y background)
```

FFI Calls for Speed

```
  )
```

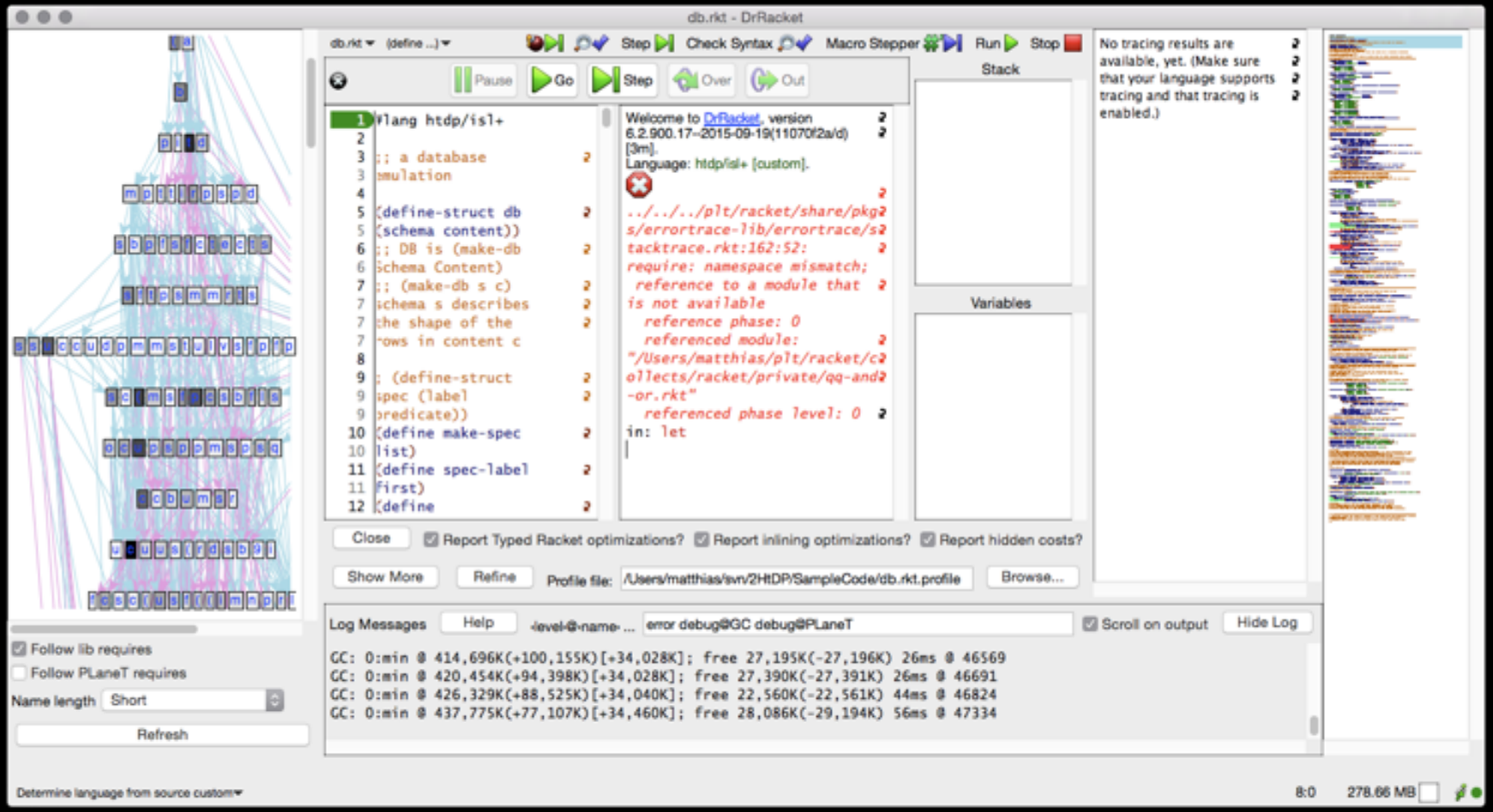
```
#lang racket

(provide
 ;; Image Number Number Image -> Image
 ;; (place obj x y bg) puts obj at (x,y) on bg
 place)

...

(define (place obj x y background)
```

ASM Code
for Speed

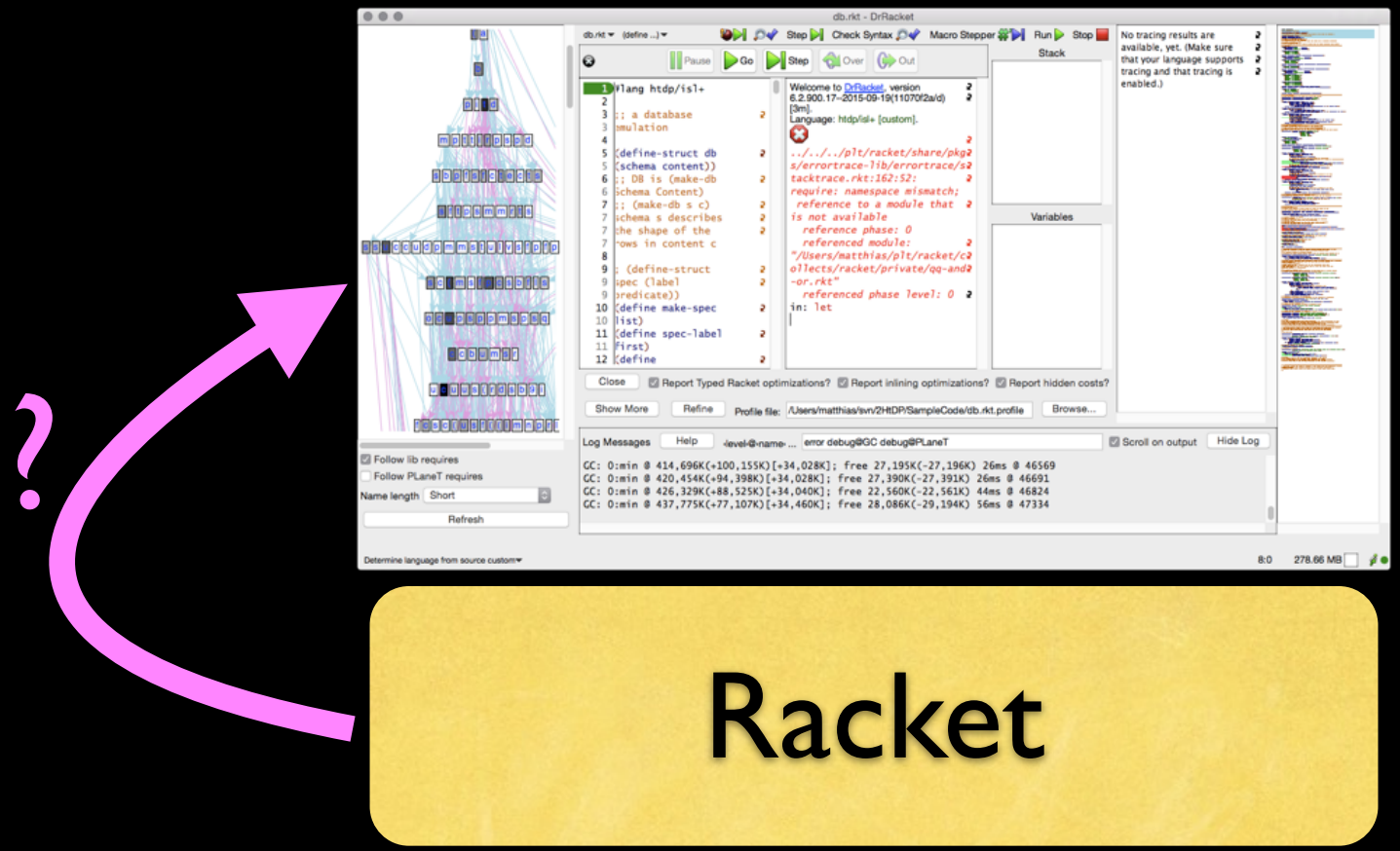
```
  )
```

More work to be done.
Coming to a RacketCon near you real soon.

racket internalizes IDE tools and operating system concepts


db.rkt - DrRacket
db.rkt
(define ...)
Step
Check Syntax
Macro Stepper
Run
Stop
Pause
Go
Step
Over
Out
Stack
Variables
No tracing results are available, yet. (Make sure that your language supports tracing and that tracing is enabled.)
#lang htdp/isl+
;; a database
emulation
(define-struct db
(schema content))
;; DB is (make-db
Schema Content)
;; (make-db s c)
schema s describes
the shape of the
rows in content c
; (define-struct
spec (label
predicate))
(define make-spec
list)
(define spec-label
first)
(define
Welcome to DrRacket, version 6.2.900.17--2015-09-19(11070f2a/d) [3m].
Language: htdp/isl+ [custom].
../../../plt/racket/share/pkgs/errortrace-lib/errortrace/stacktrace.rkt:162:52:
require: namespace mismatch;
reference to a module that is not available
reference phase: 0
referenced module: "/Users/matthias/plt/racket/collects/racket/private/qq-and-or.rkt"
referenced phase level: 0
in: let
Close
Report Typed Racket optimizations?
Report inlining optimizations?
Report hidden costs?
Show More
Refine
Profile file:
/Users/matthias/svn/2HtDP/SampleCode/db.rkt.profile
Browse...
Log Messages
Help
level@name ...
error debug@GC debug@PLaneT
Scroll on output
Hide Log
GC: 0:min @ 414,696K(+100,155K)[+34,028K]; free 27,195K(-27,196K) 26ms @ 46569
GC: 0:min @ 420,454K(+94,398K)[+34,028K]; free 27,390K(-27,391K) 26ms @ 46691
GC: 0:min @ 426,329K(+88,525K)[+34,040K]; free 22,560K(-22,561K) 44ms @ 46824
GC: 0:min @ 437,775K(+77,107K)[+34,460K]; free 28,086K(-29,194K) 56ms @ 47334
Follow lib requires
Follow PLaneT requires
Name length
Short
Refresh
Determine language from source custom
8:0
278.66 MB

What does it take to build DrRacket in Racket?



?

Racket

Racket internalizes features of IDEs and Operating Systems.

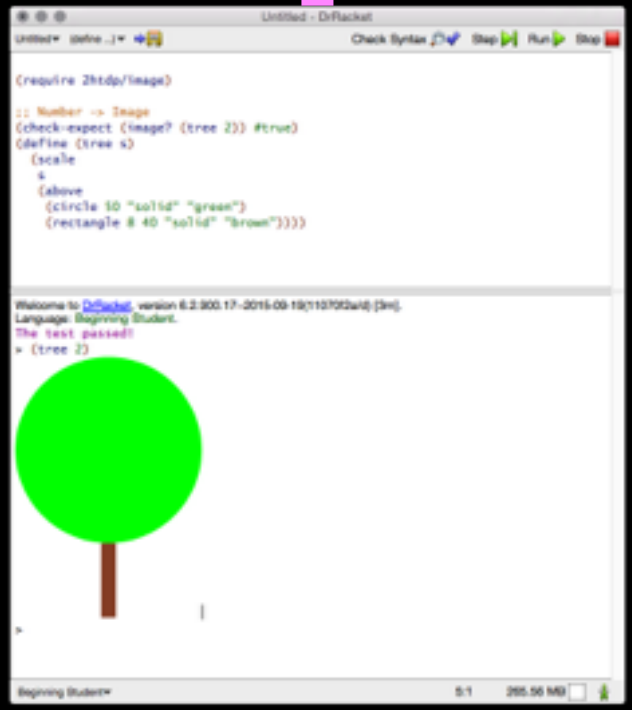
Dinner: Brian, Dunkin, Fare, and Matthias

Brian:
Isn't it amazing that you never need to program the compiler and macro stages explicitly?

Fare, Dunkin:
What???

Brian:
Just use (require x) (for-syntax x) (for-template x))
and Racket FIGURES IT ALL OUT ON ITS OWN.

racket design
needs
a feedback loop
...

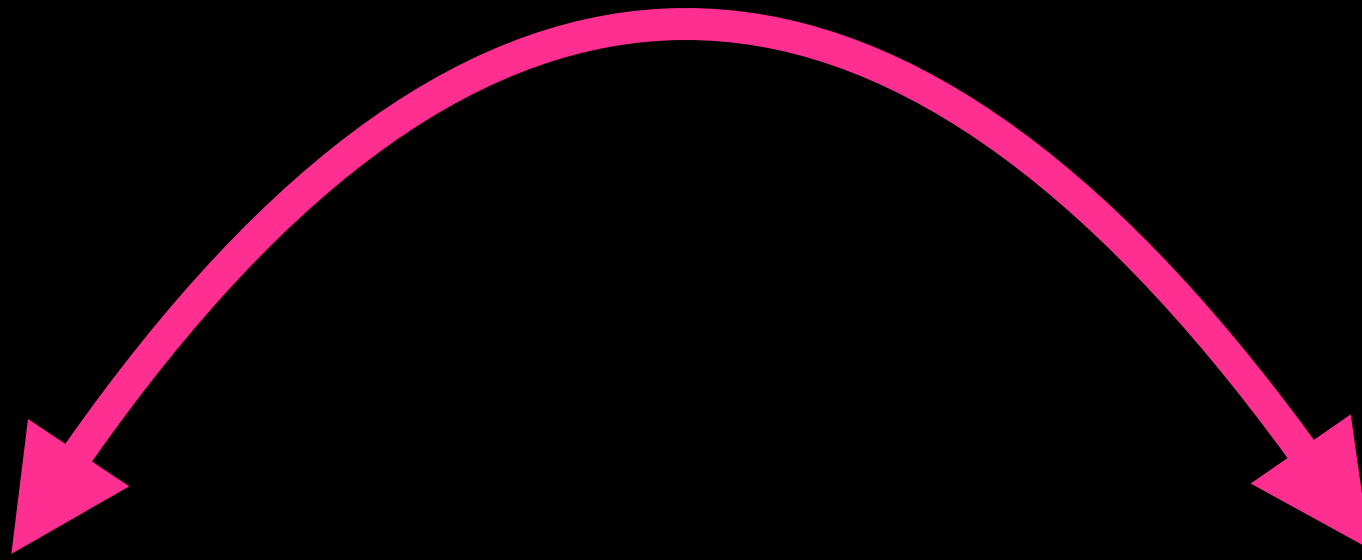

Untitled - DrRacket
Check Syntax
Step
Run
Stop
(require 2htdp/image)
;; Number -> Image
(check-expect (image? (tree 2)) #true)
(define (tree s)
(scale
4
(above
(circle 10 "solid" "green")
(rectangle 8 40 "solid" "brown"))))
Welcome to DrRacket, version 6.2.900.17--2015-09-19(1107f2a/d) [3m].
Language: Beginning Student.
The test passed!
> (tree 2)
>
Beginning Student
5:1
265.56 MB


WE NEED YOU!

take away

1. Racket is a programming-language programming language.
2. Racket is a full-spectrum programming language.
3. Racket internalizes facilities from its context (IDE, OS) *as needed*.

Racket lives inside an *academic feedback loop*.



Racket needs you.

thank you

Claire Alvis, Yavuz Arkun, Ian Barland, Eli Barzilay, Gann Bierner, Stephen Bloch, Matthew Butterick, Filipe Cabecinhas, Stephen Chang, Richard Cleis, John Clements, Richard Cobbe, Greg Cooper, Ryan Culpepper, Eric Dobson, Carl Eastlund, Moy Easwaran, Will Farr, Michael Filonenko, Burke Fetscher, Kathi Fisler, Cormac Flanagan, Sebastian Good, Paul Graunke, Kathy Gray, Dan Grossman, Arjun Guha, Dave Gurnell, Tobias Hammer, Bruce Hauman, Dave Herman, Blake Johnson, Casey Klein, Alex Knauth, Geoffrey S. Knauth, Mark Krentel, Mario Latendresse, Guillaume Marceau, Gustavo Massaccesi, Jacob Matthews, Jay McCarthy, Mike T. McHenry, Philippe Meunier, Scott Owens, David T. Pierson, Jon Rafkind, Jamie Raymond, Grant Rettke, Paul Schlie, Dorai Sitaram, Francisco Solsona, Mike Sperber, Vincent St-Amour, Paul Steckler, Stevie Strickland, James Swaine, Jens Axel Søgaard, Sam Tobin-Hochstadt, Neil Van Dyke, David Van Horn, Anton van Straaten, Asumu Takikawa, Kevin Tew, Neil Toronto, Dale Vaillancourt, Dimitris Vyzovitis, Stephanie Weirich, Noel Welsh, Adam Wick, Danny Yoo, and ChongKai Zhu.