Growing Software

From Scripts to Programs

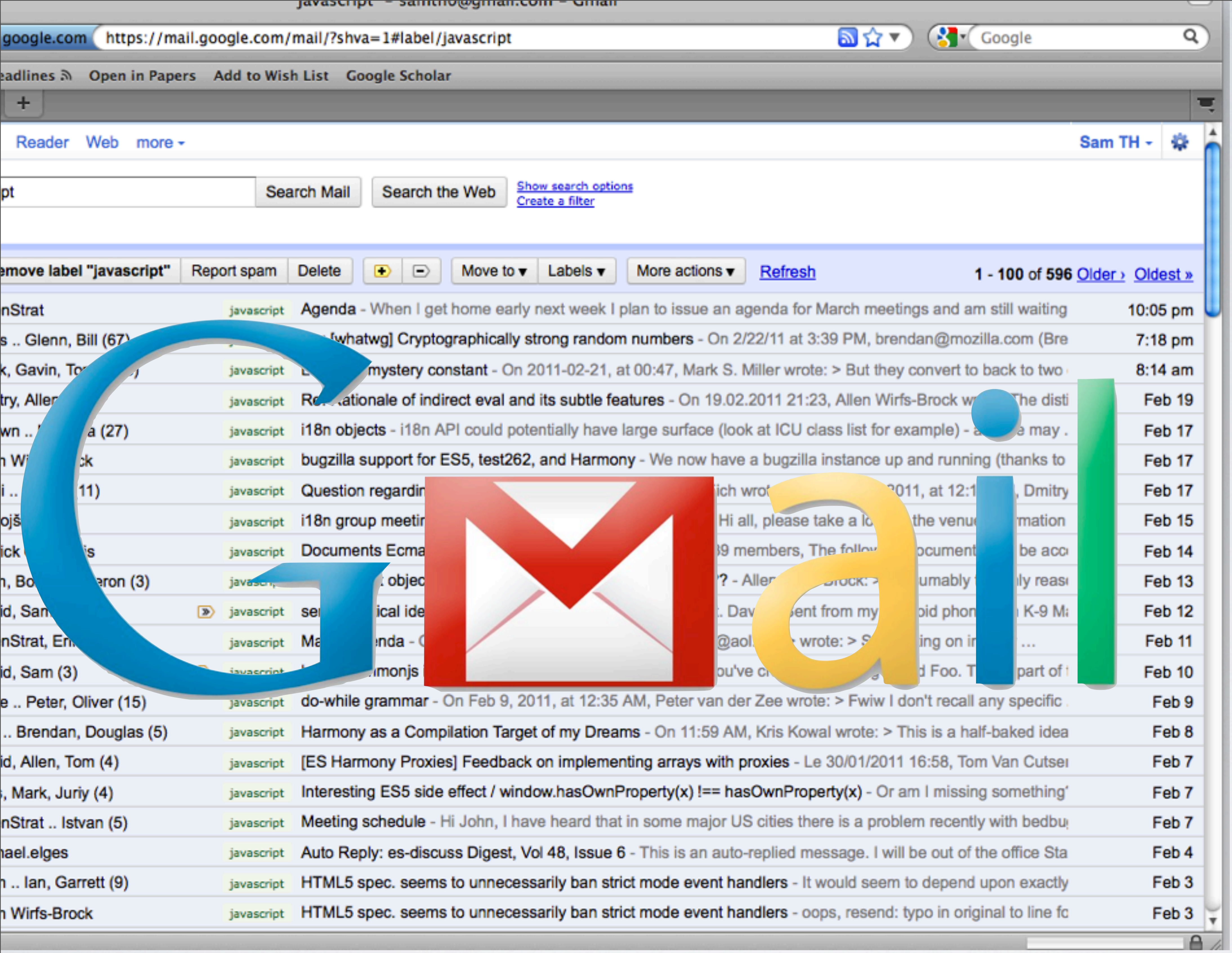
Sam Tobin-Hochstadt

March 2, 2011

Oregon State University

The Rise Of Scripting

A brief tour



javascript – samth0@gmail.com – Gmail

google.com https://mail.google.com/mail/?shva=1#label/javascript

Google

eadlines Open in Papers Add to Wish List Google Scholar

Reader Web more ▾

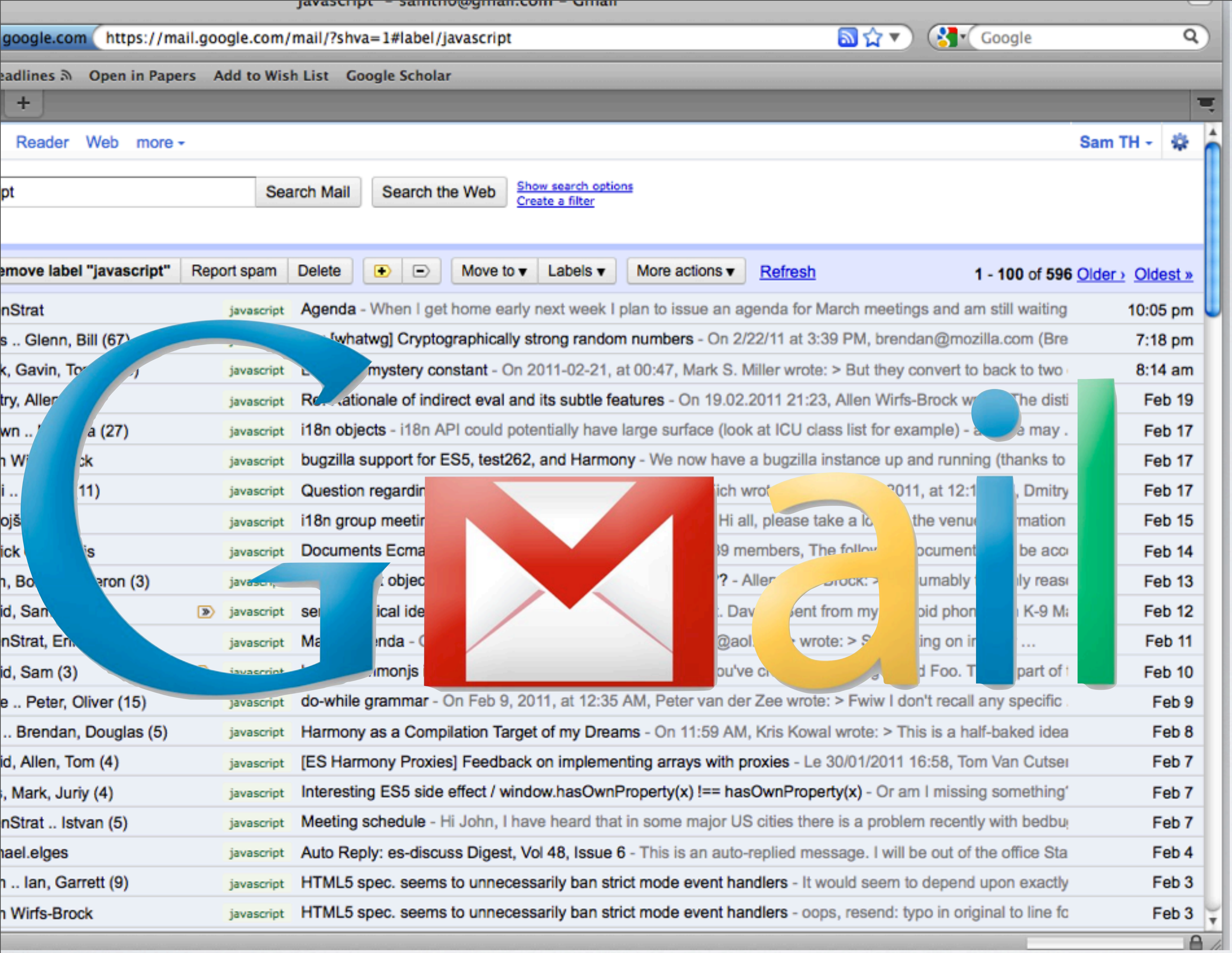
Sam TH ▾

pt Search Mail Search the Web Show search options Create a filter

emove label "javascript" Report spam Delete Move to ▾ Labels ▾ More actions ▾ Refresh

1 - 100 of 596 Older › Oldest »

nStrat	javascript	Agenda - When I get home early next week I plan to issue an agenda for March meetings and am still waiting	10:05 pm
s .. Glenn, Bill (67)		[whatwg] Cryptographically strong random numbers - On 2/22/11 at 3:39 PM, brendan@mozilla.com (Bre	7:18 pm
k, Gavin, To)	javascript	mystery constant - On 2011-02-21, at 00:47, Mark S. Miller wrote: > But they convert to back to two	8:14 am
try, Allen	javascript	Re ationale of indirect eval and its subtle features - On 19.02.2011 21:23, Allen Wirfs-Brock w The disti	Feb 19
wn .. a (27)	javascript	i18n objects - i18n API could potentially have large surface (look at ICU class list for example) - a e may .	Feb 17
n Wi ck	javascript	bugzilla support for ES5, test262, and Harmony - We now have a bugzilla instance up and running (thanks to	Feb 17
i .. 11)	javascript	Question regardin ich wrot 2011, at 12:1 , Dmitry	Feb 17
ojš	javascript	i18n group meetir Hi all, please take a lo the venu mation	Feb 15
ick is	javascript	Documents Ecma 39 members, The follo ocument be acc	Feb 14
n, Bo eron (3)	java	objec ? - Aller Brock: > umably ly reas	Feb 13
id, Sam	javascript	se ical ide . Dav ent from my oid phon K-9 Ma	Feb 12
nStrat, Er	javascript	Ma nda - @aol wrote: > S ing on ir …	Feb 11
id, Sam (3)	javascript	imonjs ou've c d Foo. T part of	Feb 10
e .. Peter, Oliver (15)	javascript	do-while grammar - On Feb 9, 2011, at 12:35 AM, Peter van der Zee wrote: > Fwiw I don't recall any specific	Feb 9
.. Brendan, Douglas (5)	javascript	Harmony as a Compilation Target of my Dreams - On 11:59 AM, Kris Kowal wrote: > This is a half-baked idea	Feb 8
id, Allen, Tom (4)	javascript	[ES Harmony Proxies] Feedback on implementing arrays with proxies - Le 30/01/2011 16:58, Tom Van Cutser	Feb 7
, Mark, Juriy (4)	javascript	Interesting ES5 side effect / window.hasOwnProperty(x) !== hasOwnProperty(x) - Or am I missing something'	Feb 7
nStrat .. Istvan (5)	javascript	Meeting schedule - Hi John, I have heard that in some major US cities there is a problem recently with bedbu	Feb 7
nael.elges	javascript	Auto Reply: es-discuss Digest, Vol 48, Issue 6 - This is an auto-replied message. I will be out of the office Sta	Feb 4
n .. Ian, Garrett (9)	javascript	HTML5 spec. seems to unnecessarily ban strict mode event handlers - It would seem to depend upon exactly	Feb 3
n Wirfs-Brock	javascript	HTML5 spec. seems to unnecessarily ban strict mode event handlers - oops, resend: typo in original to line fc	Feb 3



javascript" – samth@gmail.com – Gmail

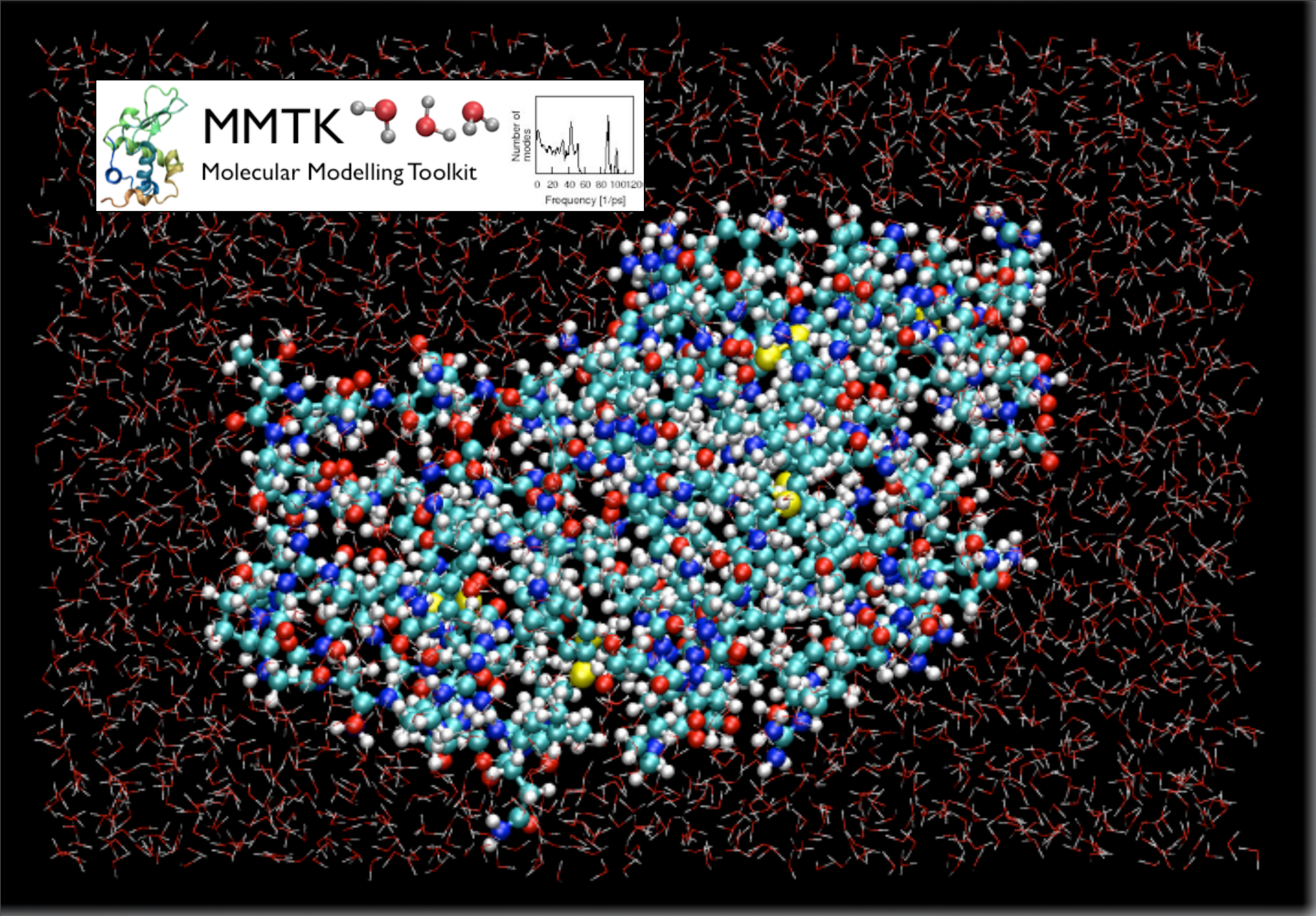
google.com https://mail.google.com/mail/?shva=1#label/javascript

1 - 100 of 596 Older › Oldest »

nStrat	javascript	Agenda - When I get home early next week I plan to issue an agenda for March meetings and am still waiting	10:05 pm
s .. Glenn, Bill (67		[whatwg] Cryptographically strong random numbers - On 2/22/11 at 3:39 PM, brendan@mozilla.com (Bre	7:18 pm
k, Gavin, To)	javascript	mystery constant - On 2011-02-21, at 00:47, Mark S. Miller wrote: > But they convert to back to two	8:14 am
try, Allen	javascript	Ra ationale of indirect eval and its subtle features - On 19.02.2011 21:23, Allen Wirfs-Brock w The disti	Feb 19
wn .. a (27)	javascript	i18n objects - i18n API could potentially have large surface (look at ICU class list for example) - a e may .	Feb 17
n Wi ck	javascript	bugzilla support for ES5, test262, and Harmony - We now have a bugzilla instance up and running (thanks to	Feb 17
i .. 11)	javascript	Question regardin ich wrot 2011, at 12:1 Dmitry	Feb 17
ojš	javascript	i18n group meetin Hi all, please take a lo the venu mation	Feb 15
ick is	javascript	Documents Ecma 39 members, The follo ocument be acc	Feb 14
n, Bo eron (3)	java	objec ? - Allen Brock: > umably ly reas	Feb 13
id, Sam	javascript	se ical ide . Dav ent from my oid phon K-9 M	Feb 12
nStrat, Er	javascript	Ma enda - @aol wrote: > S ing on i …	Feb 11
id, Sam (3)	javascript	monjs ou've c d Foo. T part of	Feb 10
e .. Peter, Oliver (15)	javascript	do-while grammar - On Feb 9, 2011, at 12:35 AM, Peter van der Zee wrote: > Fwiw I don't recall any specific	Feb 9
.. Brendan, Douglas (5)	javascript	Harmony as a Compilation Target of my Dreams - On 11:59 AM, Kris Kowal wrote: > This is a half-baked idea	Feb 8
id, Allen, Tom (4)	javascript	[ES Harmony Proxies] Feedback on implementing arrays with proxies - Le 30/01/2011 16:58, Tom Van Cutser	Feb 7
, Mark, Juriy (4)	javascript	Interesting ES5 side effect / window.hasOwnProperty(x) !== hasOwnProperty(x) - Or am I missing something	Feb 7
nStrat .. Istvan (5)	javascript	Meeting schedule - Hi John, I have heard that in some major US cities there is a problem recently with bedbu	Feb 7
ael.elges	javascript	Auto Reply: es-discuss Digest, Vol 48, Issue 6 - This is an auto-replied message. I will be out of the office Sta	Feb 4
n .. Ian, Garrett (9)	javascript	HTML5 spec. seems to unnecessarily ban strict mode event handlers - It would seem to depend upon exactly	Feb 3
n Wirfs-Brock	javascript	HTML5 spec. seems to unnecessarily ban strict mode event handlers - oops, resend: typo in original to line fo	Feb 3

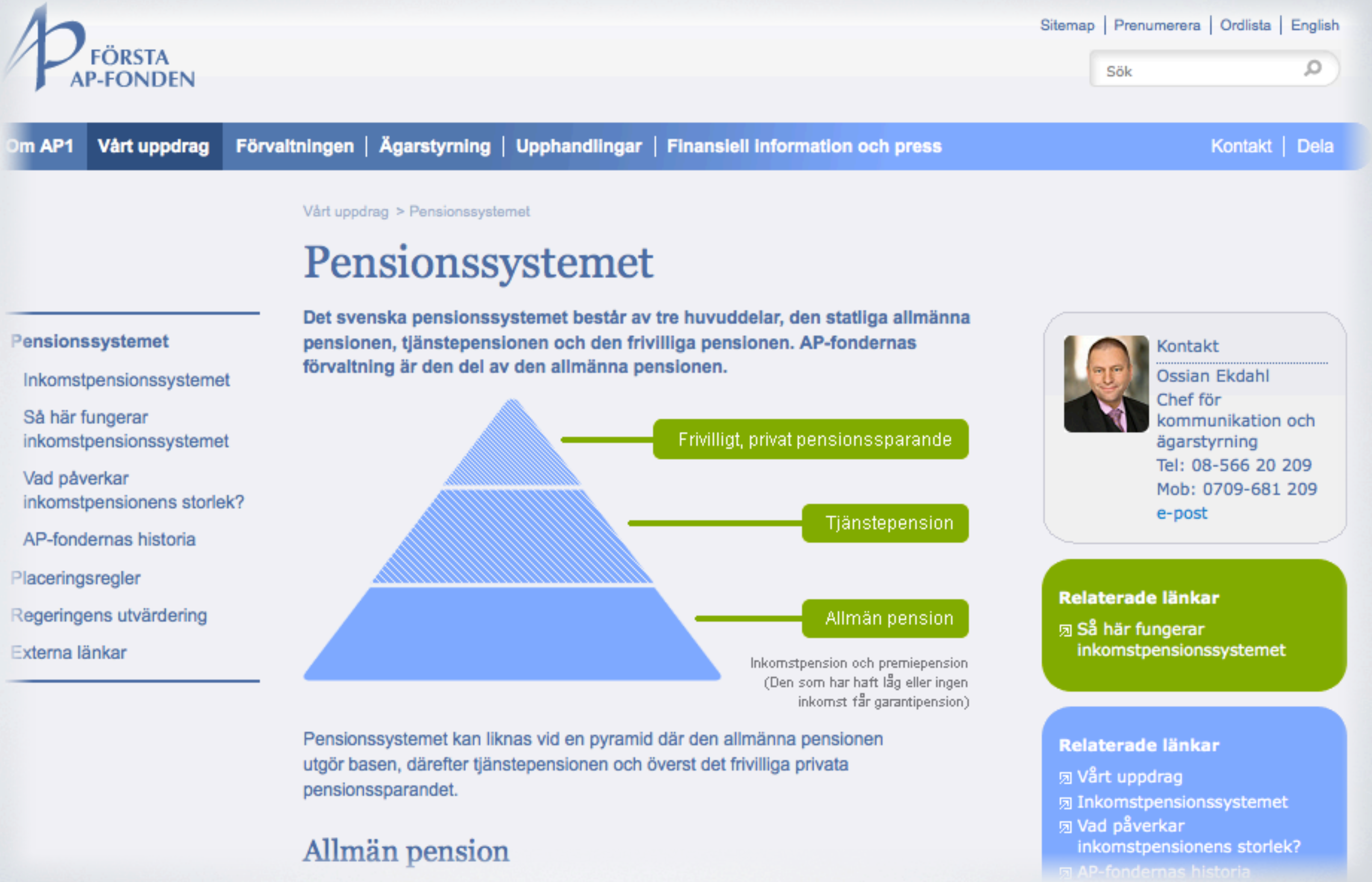
JavaScript

Lua


MMTK
Molecular Modelling Toolkit
Number of modes
0 20 40 60 80 100120
Frequency [1/ps]

Python

Perl



Sitemap | Prenumerera | Ordlista | English

Sök

Om AP1 | Vårt uppdrag | Förvaltningen | Ägarstyrning | Upphandlingar | Finansiell information och press | Kontakt | Dela

Vårt uppdrag > Pensionssystemet

Pensionssystemet

- Pensionssystemet
  - Inkomstpensionssystemet
  - Så här fungerar inkomstpensionssystemet
  - Vad påverkar inkomstpensionens storlek?
  - AP-fondernas historia
- Placeringsregler
- Regeringens utvärdering
- Externa länkar

Det svenska pensionssystemet består av tre huvuddelar, den statliga allmänna pensionen, tjänstepensionen och den frivilliga pensionen. AP-fondernas förvaltning är den del av den allmänna pensionen.

Frivilligt, privat pensionssparande
Tjänstepension
Allmän pension
Inkomstpension och premiepension
(Den som har haft låg eller ingen
inkomst får garantipension)

Pensionssystemet kan liknas vid en pyramid där den allmänna pensionen utgör basen, därefter tjänstepensionen och överst det frivilliga privata pensionssparandet.

Allmän pension

Kontakt
Ossian Ekdahl
Chef för kommunikation och ägarstyrning
Tel: 08-566 20 209
Mob: 0709-681 209
e-post

Relaterade länkar
- Så här fungerar inkomstpensionssystemet

Relaterade länkar
- Vårt uppdrag
- Inkomstpensionssystemet
- Vad påverkar inkomstpensionens storlek?
- AP-fondernas historia

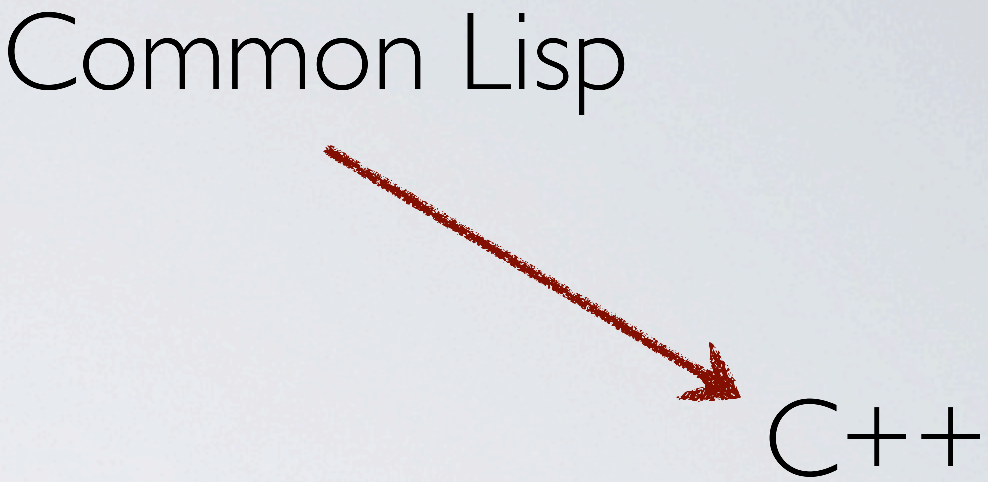
PPM Swedish Pensions

Quick hack to critical system:
The paradigmatic scripting story

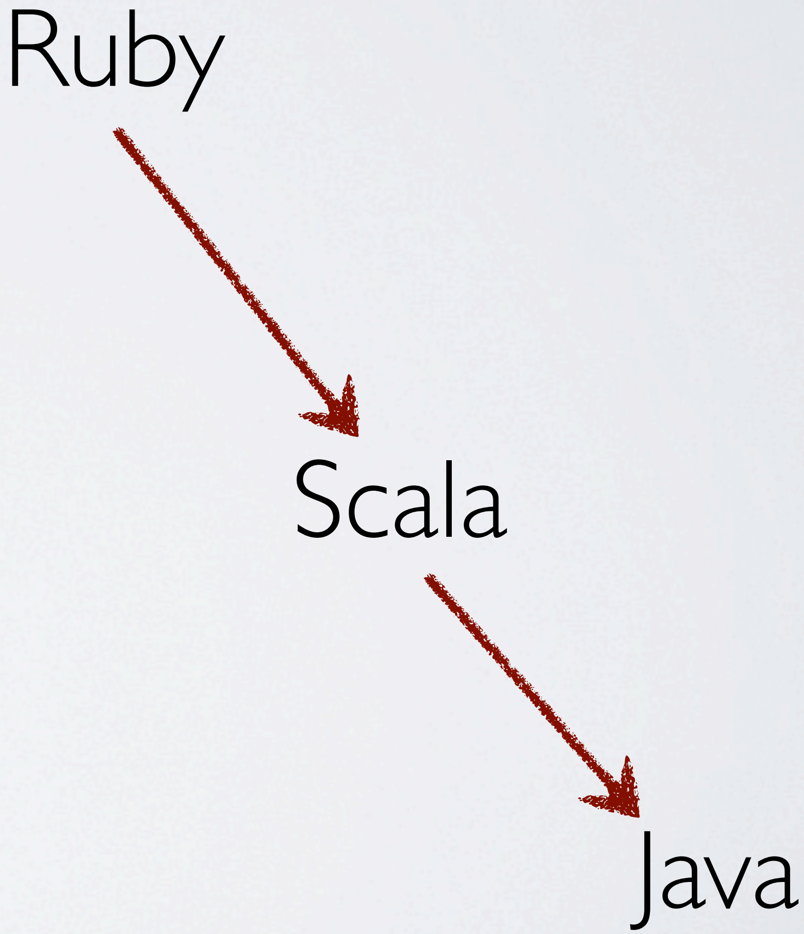
Started as a backup system
Ended managing billions in assets

“whipitupitude” — Larry Wall

viaweb


Common Lisp

C++


Ruby

Scala

Java

Addressing the Challenge

Non-Solutions

Waterfall development of spec and code

Replace all scripting languages

Omniscient program analysis

Non-Solutions

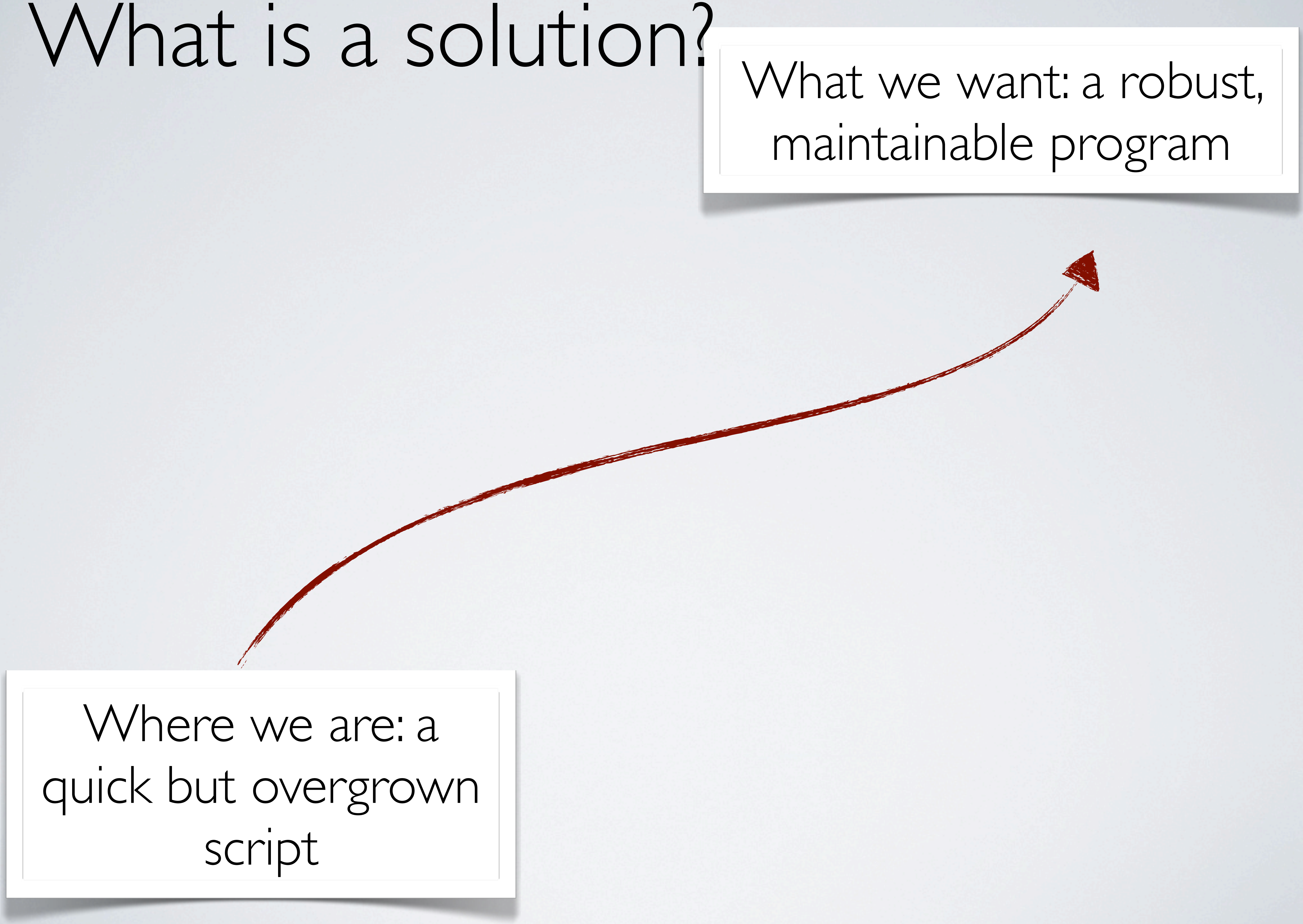
Waterfall development of spec and code

Replace all scripting languages

Omniscient program analysis

The all-too-common result: rewrite in C++/Java

What is a solution?



What we want: a robust, maintainable program

Where we are: a quick but overgrown script

What is a solution?

What we want: a robust, maintainable program

Existing PL technology:

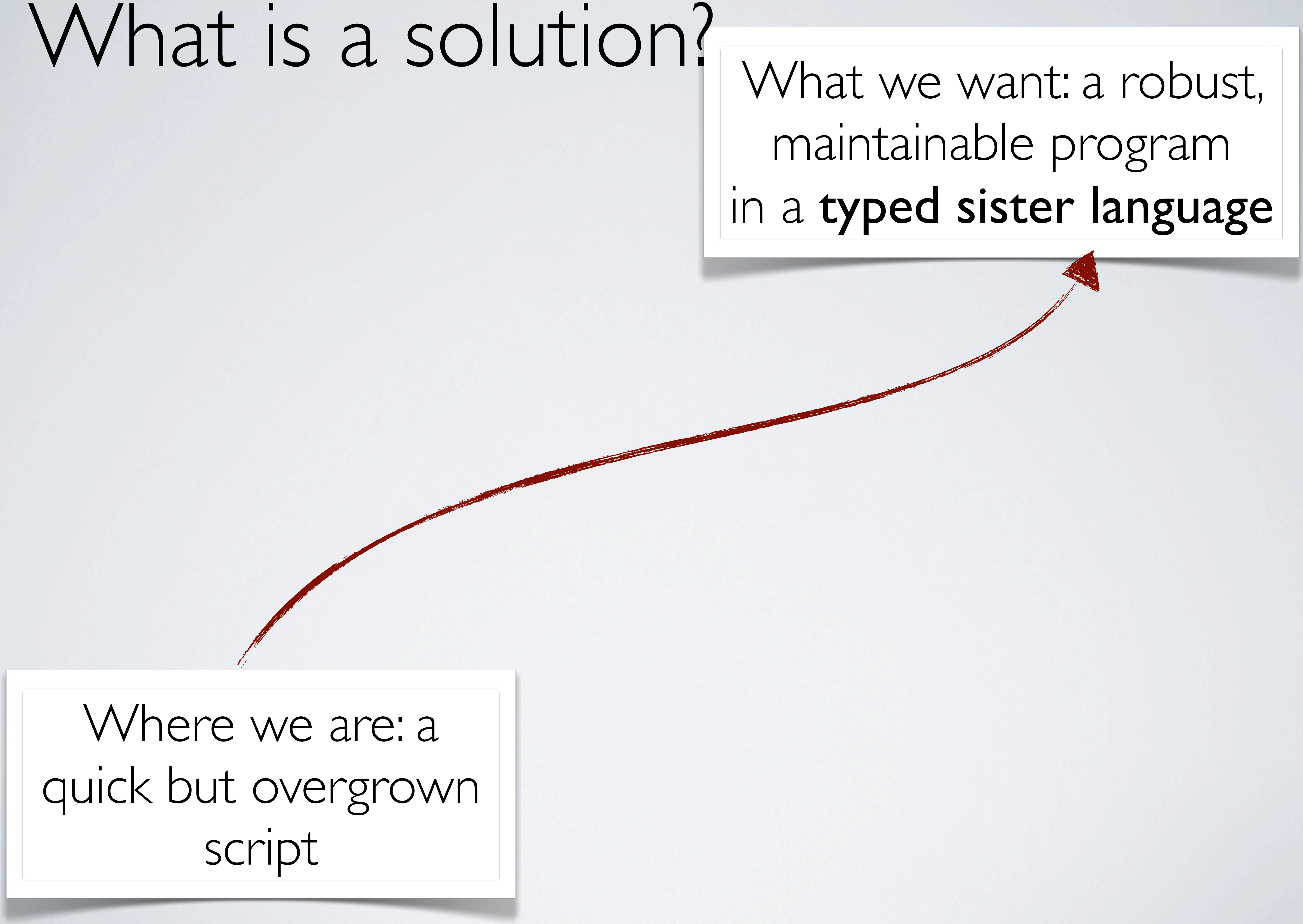
Types as lightweight specifications

- Robustness via static enforcement
- Maintainability via checked specs
- Evolution via refactoring support

W

quick but overgrown script

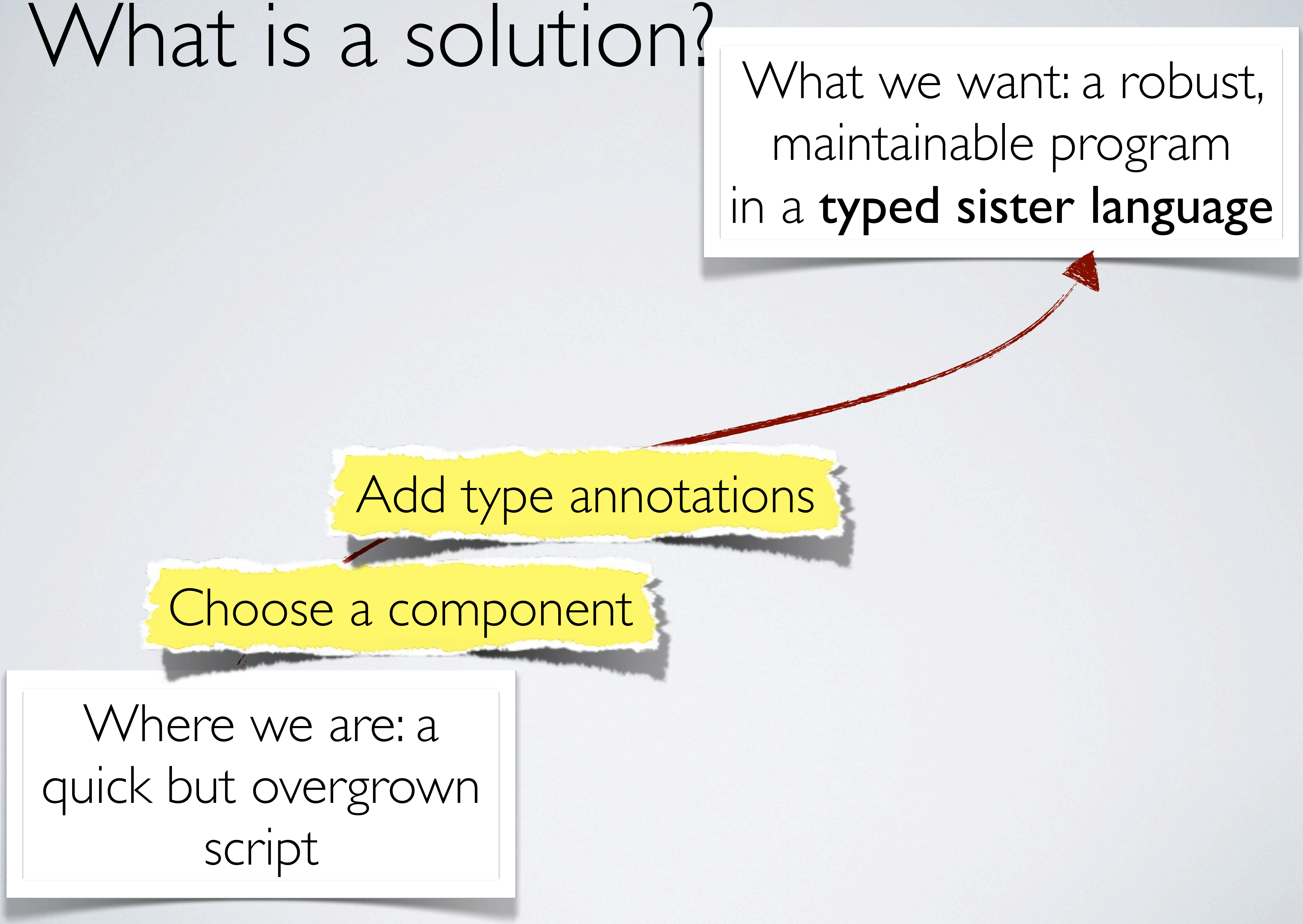
What is a solution?



What we want: a robust, maintainable program in a **typed sister language**

Where we are: a quick but overgrown script

What is a solution?



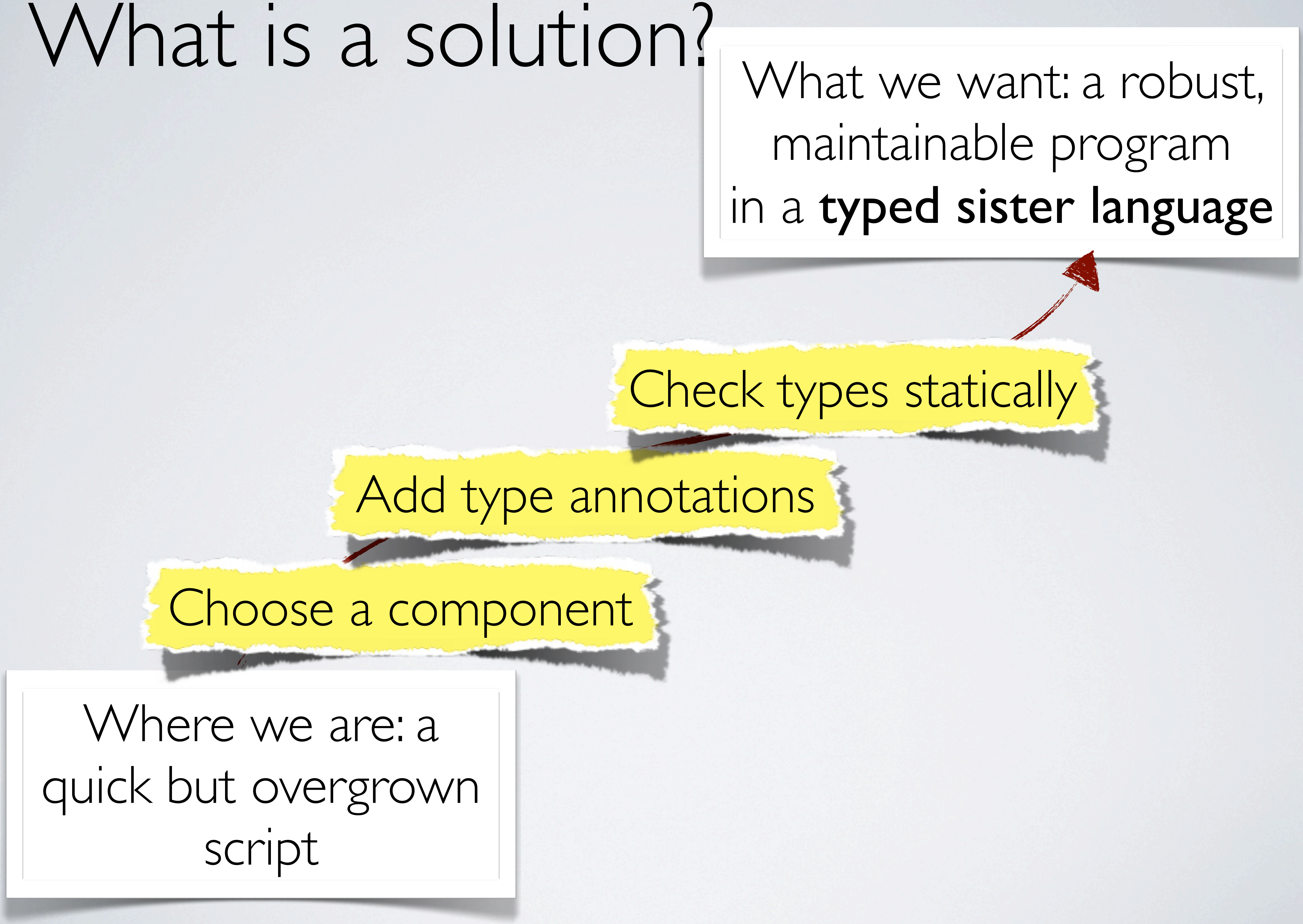
What we want: a robust, maintainable program in a **typed sister language**

Add type annotations

Choose a component

Where we are: a quick but overgrown script

What is a solution?



Where we are: a quick but overgrown script

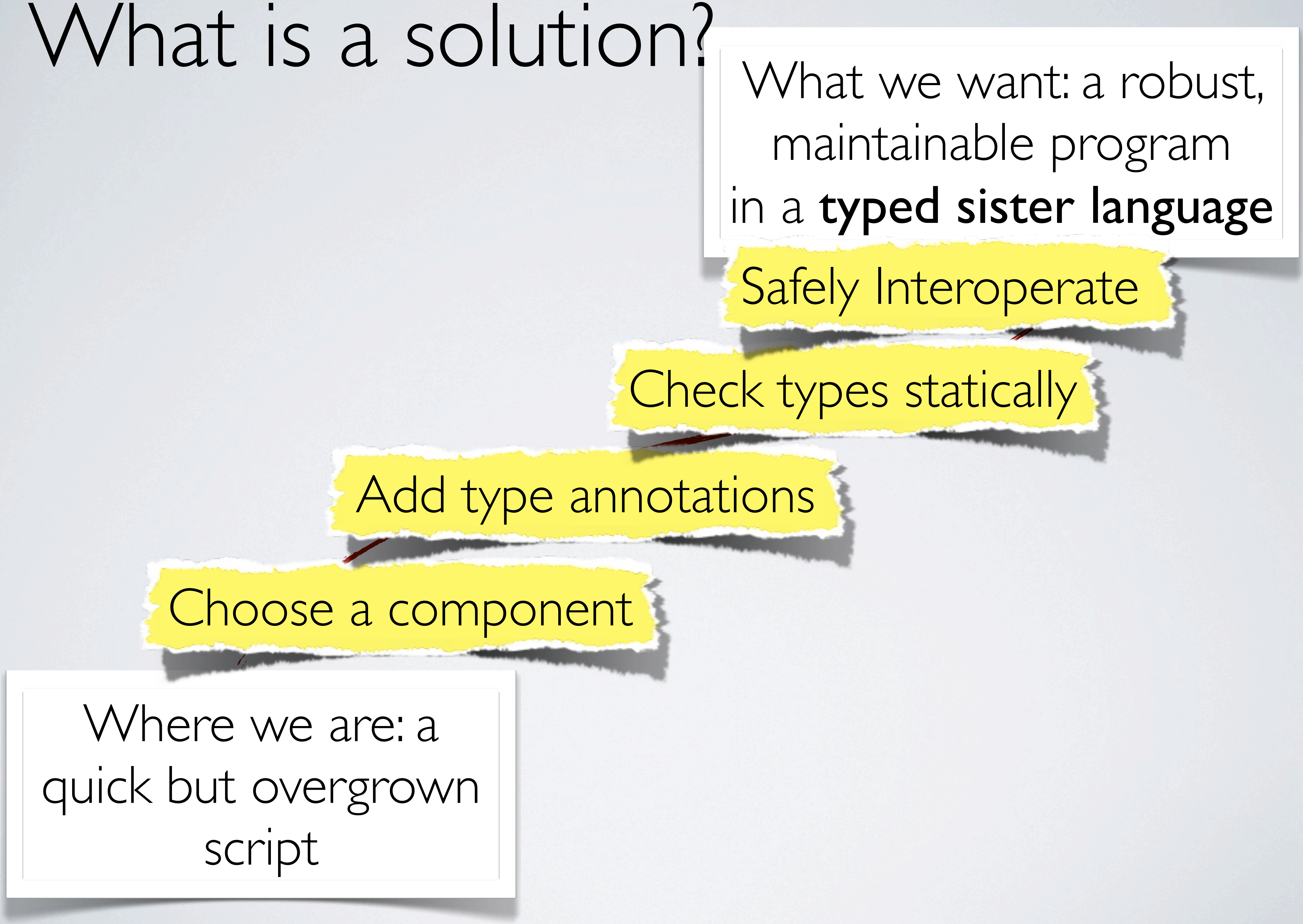
Choose a component

Add type annotations

Check types statically

What we want: a robust, maintainable program in a **typed sister language**

What is a solution?



Where we are: a quick but overgrown script

Choose a component

Add type annotations

Check types statically

Safely Interoperate

What we want: a robust, maintainable program in a **typed sister language**

What is a solution?

What we want: a robust, maintainable program in a **typed sister language**

Safely Interoperate

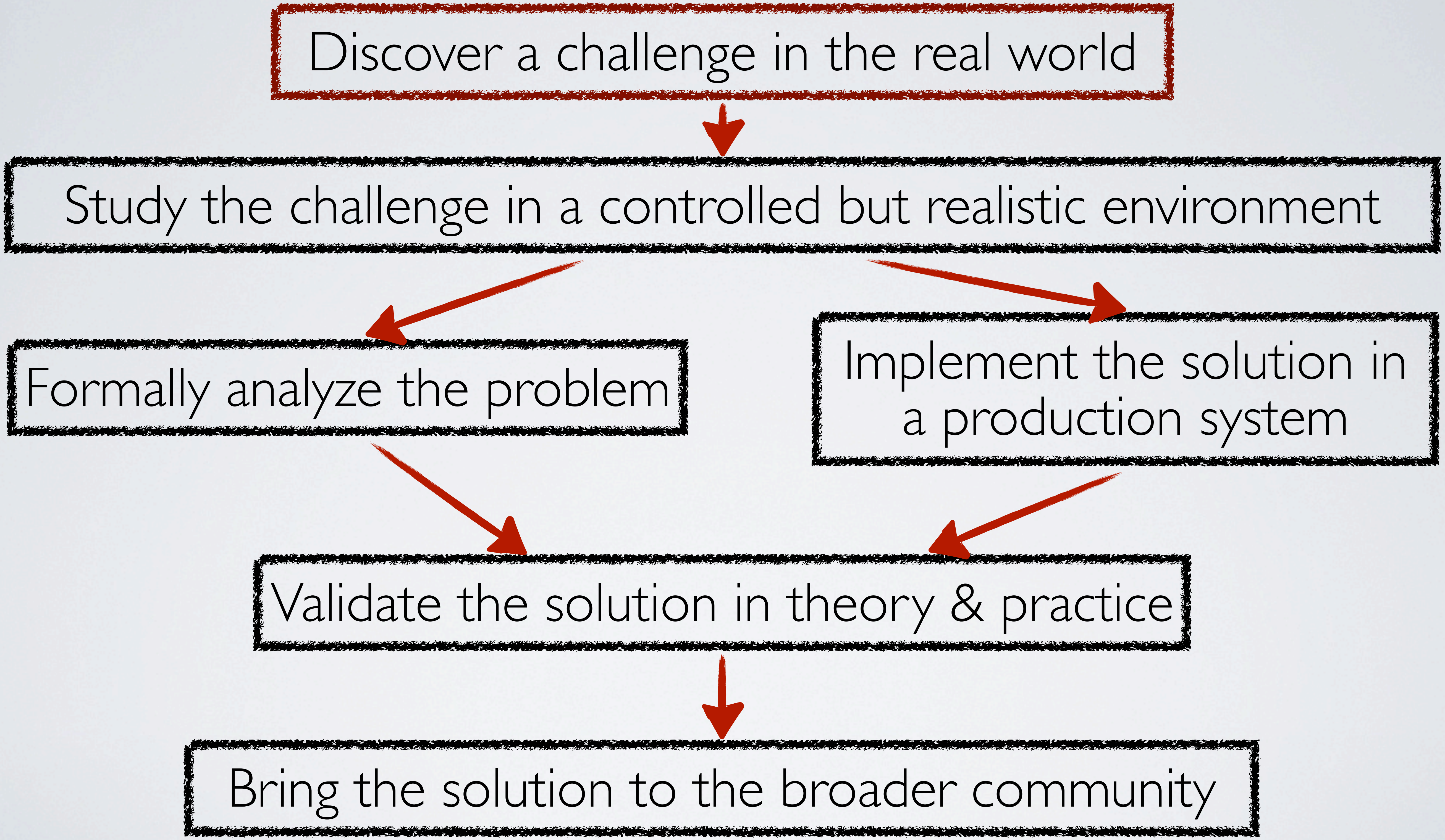
Check types statically

Add type annotations

Choose a component

Where we are: a quick but overgrown script

My Research Methodology



Discover a challenge in the real world

Study the challenge in a controlled but realistic environment

Formally analyze the problem

Implement the solution in a production system

Validate the solution in theory & practice

Bring the solution to the broader community

Racket

A descendant of Lisp & Scheme

15 years of development

20+ current developers

Used in dozens of companies, 120 universities, 200 schools

500,000 line code base

Ideal environment for investigating script to program evolution

Typed Racket

A typed dialect of Racket

Publicly distributed for 4+ years

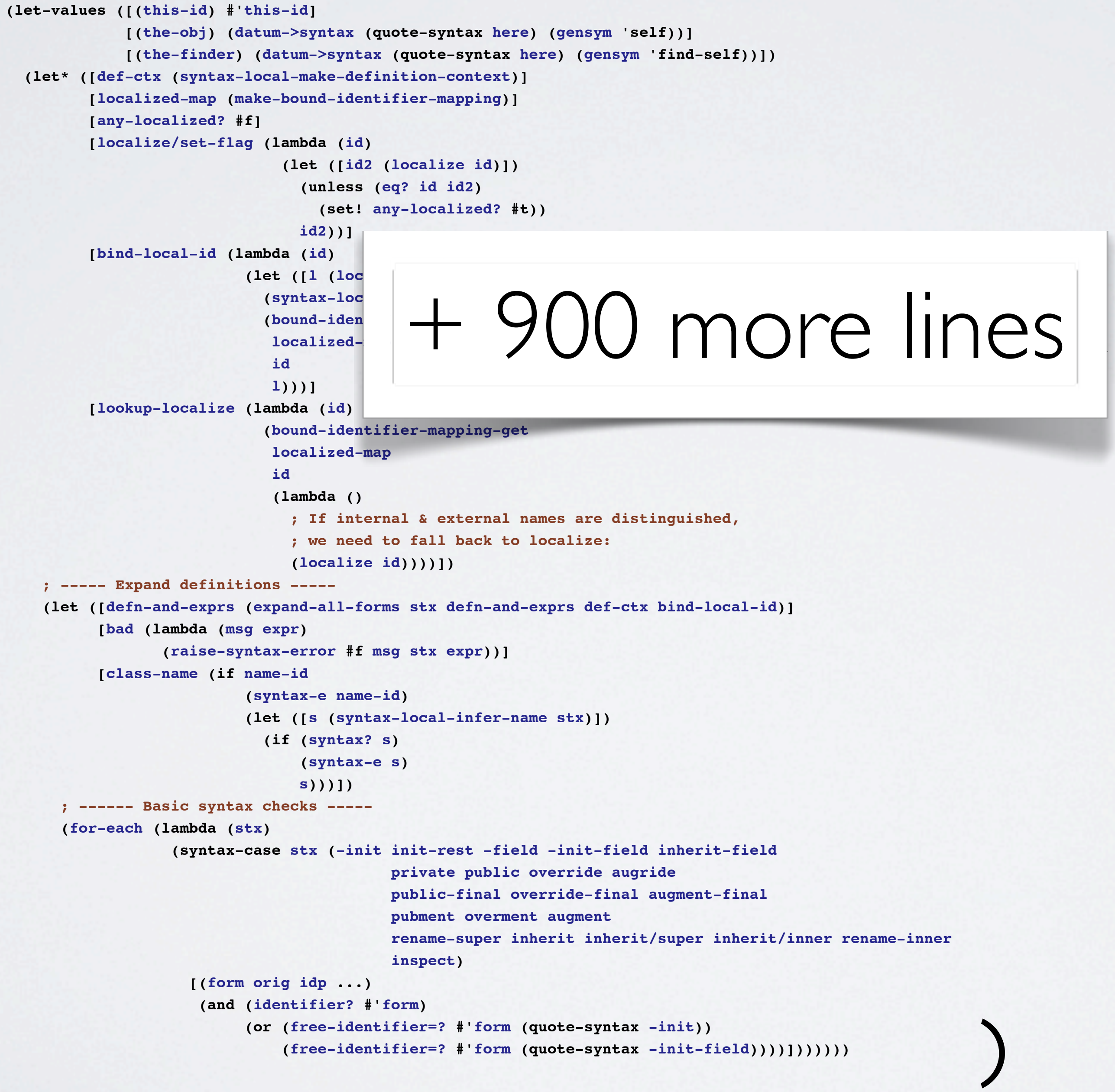
Used in key Racket systems

Used in multiple companies and several college courses

Supports dozens of existing libraries

A testbed for scripts-to-programs research

```
(define (main stx trace-flag super-expr
              deser-id-expr name-id
              ifc-exprs defn-and-exprs)
```



```
(let-values ([(this-id) #'this-id]
             [(the-obj) (datum->syntax (quote-syntax here) (gensym 'self))]
             [(the-finder) (datum->syntax (quote-syntax here) (gensym 'find-self))])
  (let* ([def-ctx (syntax-local-make-definition-context)]
         [localized-map (make-bound-identifier-mapping)]
         [any-localized? #f]
         [localize/set-flag (lambda (id)
                              (let ([id2 (localize id)])
                                (unless (eq? id id2)
                                  (set! any-localized? #t))
                                id2))]
         [bind-local-id (lambda (id)
                          (let ([l (loc
                            (syntax-loc
                            (bound-iden
                             localized-
                             id
                             l))))]
         [lookup-localize (lambda (id)
                            (bound-identifier-mapping-get
                             localized-map
                             id
                             (lambda ()
                               ; If internal & external names are distinguished,
                               ; we need to fall back to localize:
                               (localize id)))))])
    ; ----- Expand definitions -----
    (let ([defn-and-exprs (expand-all-forms stx defn-and-exprs def-ctx bind-local-id)]
          [bad (lambda (msg expr)
                 (raise-syntax-error #f msg stx expr))]
          [class-name (if name-id
                          (syntax-e name-id)
                          (let ([s (syntax-local-infer-name stx)])
                            (if (syntax? s)
                                (syntax-e s)
                                s)))])
      ; ------ Basic syntax checks -----
      (for-each (lambda (stx)
                  (syntax-case stx (-init init-rest -field -init-field inherit-field
                                          private public override augride
                                          public-final override-final augment-final
                                          pubment overment augment
                                          rename-super inherit inherit/super inherit/inner rename-inner
                                          inspect)
                    [(form orig idp ...)
                     (and (identifier? #'form)
                          (or (free-identifier=? #'form (quote-syntax -init))
                              (free-identifier=? #'form (quote-syntax -init-field))))])))))
```

+ 900 more lines

)

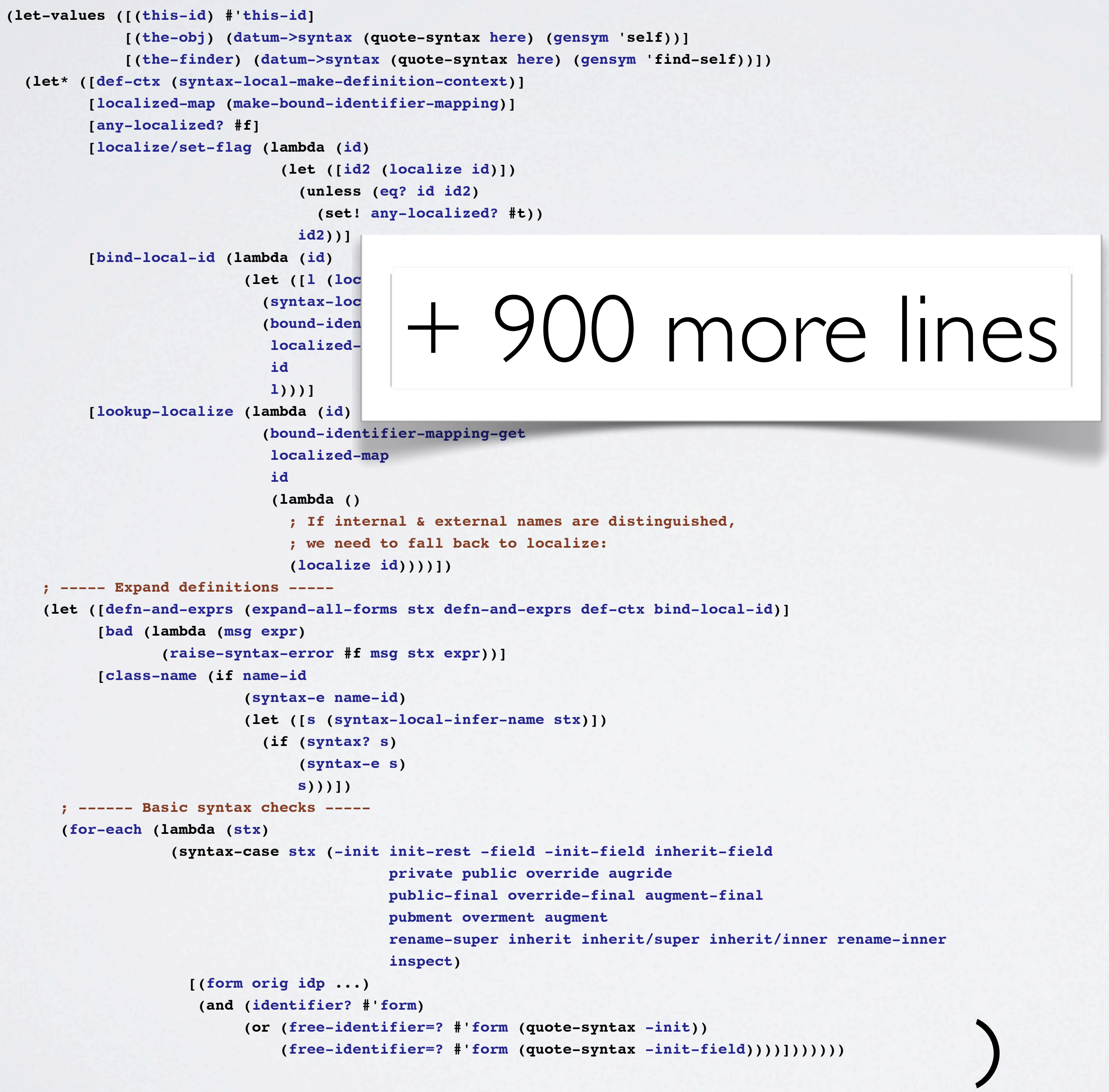
```
;; Start Here
(define (main stx trace-flag super-expr
              deser-id-expr name-id
              ifc-exprs defn-and-exprs)
```

```
(let-values ([(this-id) #'this-id]
             [(the-obj) (datum->syntax (quote-syntax here) (gensym 'self))]
             [(the-finder) (datum->syntax (quote-syntax here) (gensym 'find-self))])
  (let* ([def-ctx (syntax-local-make-definition-context)]
         [localized-map (make-bound-identifier-mapping)]
         [any-localized? #f]
         [localize/set-flag (lambda (id)
                              (let ([id2 (localize id)])
                                (unless (eq? id id2)
                                  (set! any-localized? #t))
                                id2))]
         [bind-local-id (lambda (id)
                          (let ([l (loc
                            (syntax-loc
                            (bound-iden
                             localized-
                             id
                             l)))]
         [lookup-localize (lambda (id)
                            (bound-identifier-mapping-get
                             localized-map
                             id
                             (lambda ()
                               ; If internal & external names are distinguished,
                               ; we need to fall back to localize:
                               (localize id))))])
    ; ----- Expand definitions -----
    (let ([defn-and-exprs (expand-all-forms stx defn-and-exprs def-ctx bind-local-id)]
          [bad (lambda (msg expr)
                 (raise-syntax-error #f msg stx expr))]
          [class-name (if name-id
                          (syntax-e name-id)
                          (let ([s (syntax-local-infer-name stx)])
                            (if (syntax? s)
                                (syntax-e s)
                                s)))])
      ; ------ Basic syntax checks -----
      (for-each (lambda (stx)
                  (syntax-case stx (-init init-rest -field -init-field inherit-field
                                          private public override augride
                                          public-final override-final augment-final
                                          pubment overment augment
                                          rename-super inherit inherit/super inherit/inner rename-inner
                                          inspect)
                    [(form orig idp ...)
                     (and (identifier? #'form)
                          (or (free-identifier=? #'form (quote-syntax -init))
                              (free-identifier=? #'form (quote-syntax -init-field))))])))))
```

+ 900 more lines

)

```
(: main : Stx Bool Expr (or #f Id) ... -> Expr)
(define (main stx trace-flag super-expr
              deser-id-expr name-id
              ifc-exprs defn-and-exprs)
```



```
(let-values ([(this-id) #'this-id]
             [(the-obj) (datum->syntax (quote-syntax here) (gensym 'self))]
             [(the-finder) (datum->syntax (quote-syntax here) (gensym 'find-self))])
  (let* ([def-ctx (syntax-local-make-definition-context)]
         [localized-map (make-bound-identifier-mapping)]
         [any-localized? #f]
         [localize/set-flag (lambda (id)
                              (let ([id2 (localize id)])
                                (unless (eq? id id2)
                                  (set! any-localized? #t))
                                id2))]
         [bind-local-id (lambda (id)
                          (let ([l (loc
                            (syntax-loc
                            (bound-iden
                             localized-
                             id
                             l)))]
         [lookup-localize (lambda (id)
                            (bound-identifier-mapping-get
                             localized-map
                             id
                             (lambda ()
                               ; If internal & external names are distinguished,
                               ; we need to fall back to localize:
                               (localize id))))])
    ; ----- Expand definitions -----
    (let ([defn-and-exprs (expand-all-forms stx defn-and-exprs def-ctx bind-local-id)]
          [bad (lambda (msg expr)
                 (raise-syntax-error #f msg stx expr))]
          [class-name (if name-id
                          (syntax-e name-id)
                          (let ([s (syntax-local-infer-name stx)])
                            (if (syntax? s)
                                (syntax-e s)
                                s)))])
      ; ------ Basic syntax checks -----
      (for-each (lambda (stx)
                  (syntax-case stx (-init init-rest -field -init-field inherit-field
                                          private public override augride
                                          public-final override-final augment-final
                                          pubment overment augment
                                          rename-super inherit inherit/super inherit/inner rename-inner
                                          inspect)
                    [(form orig idp ...)
                     (and (identifier? #'form)
                          (or (free-identifier=? #'form (quote-syntax -init))
                              (free-identifier=? #'form (quote-syntax -init-field))))])))))
```

+ 900 more lines

```
)
```

Safe Interoperation

Modular Programs,
Modular Checking

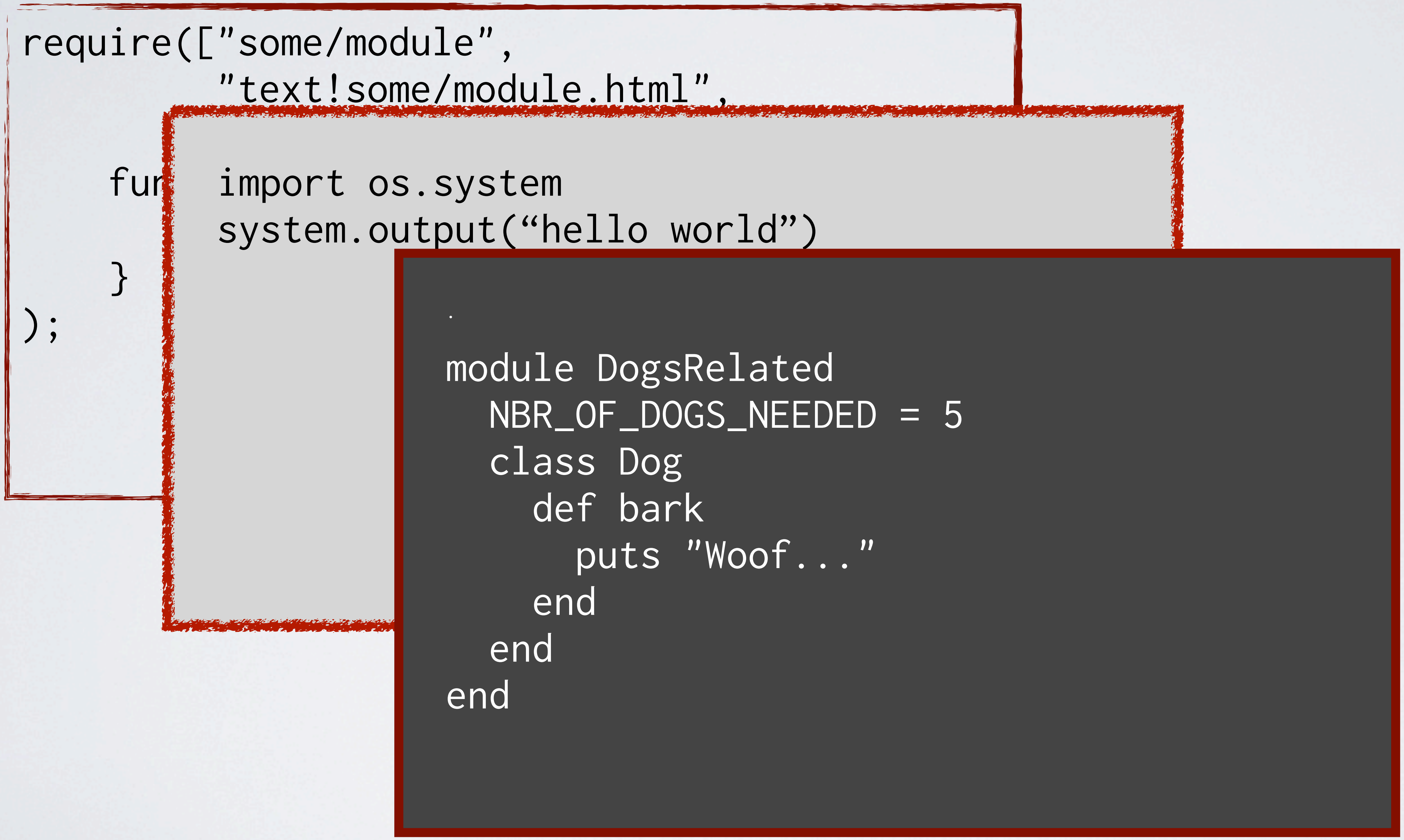
```
require(["some/module",
         "text!some/module.html",
         "text!some/module.css"],
    function(module, html, css) {
        return style_with(html, css);
    }
);
```

Modular Programs,
Modular Checking

```
require(["some/module",
         "text!some/module.html",


     fun

     }
);
```

```
import os.system
system.output(“hello world”)
```

Modular Programs,
Modular Checking

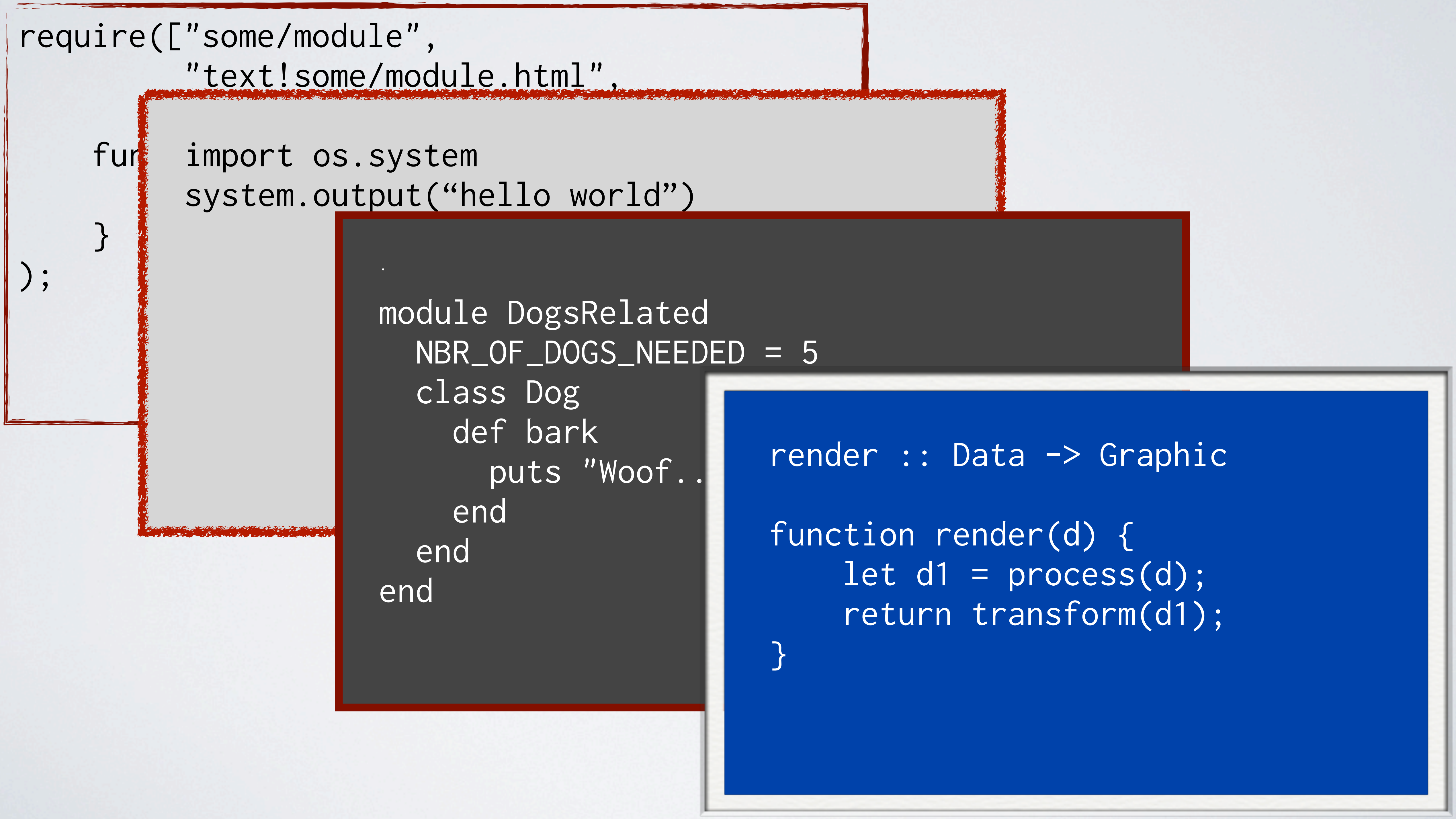


```
require(["some/module",
         "text!some/module.html",

    fur

    }
);
```

```
import os.system
system.output(“hello world”)
```

```
.

module DogsRelated
  NBR_OF_DOGS_NEEDED = 5
  class Dog
    def bark
      puts "Woof..."
    end
  end
end
```

Modular Programs, Modular Checking



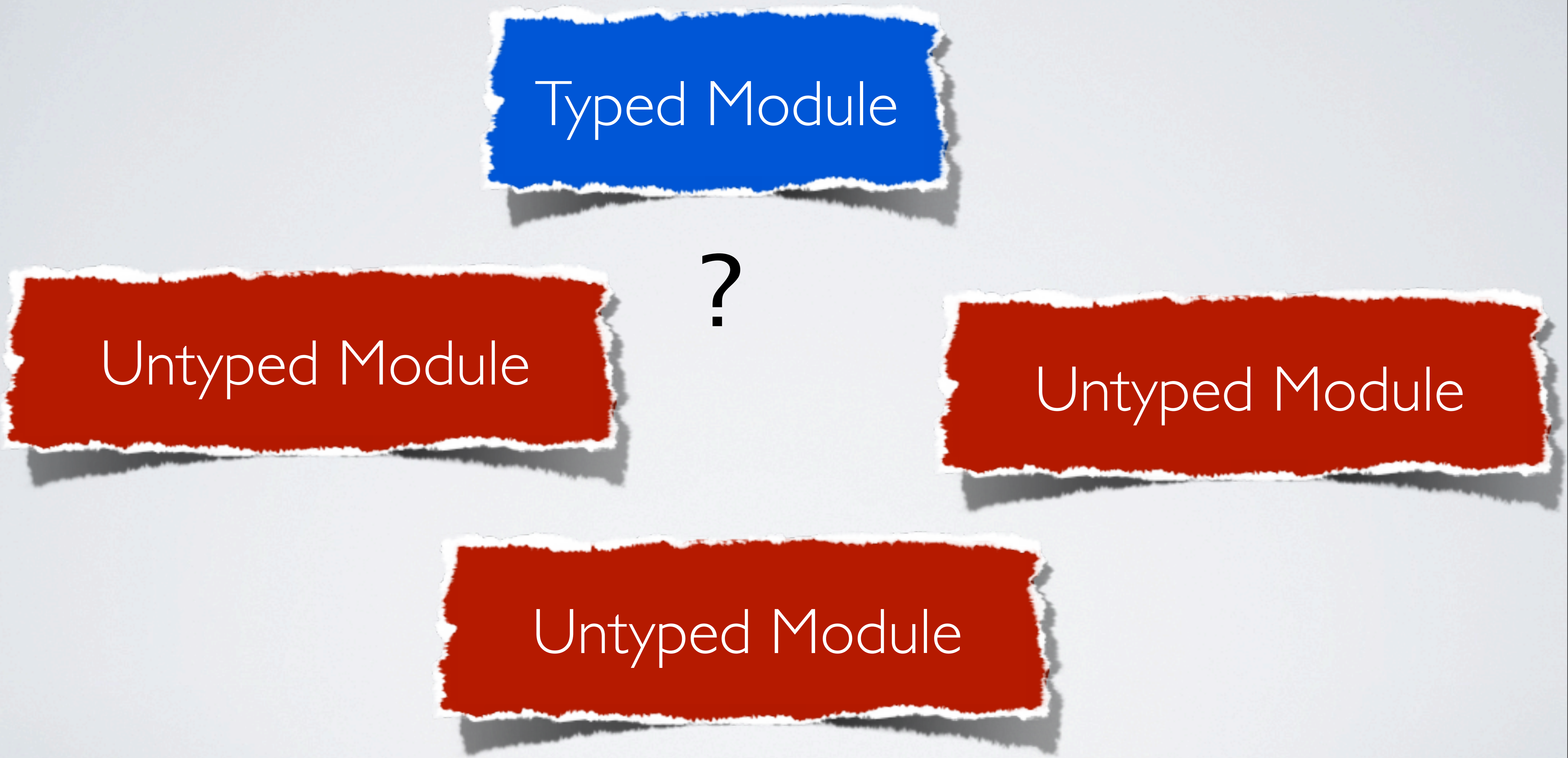
```
require(["some/module",
         "text!some/module.html",

    fun

    }
);
```

```
import os.system
system.output(“hello world”)
```

```
module DogsRelated
  NBR_OF_DOGS_NEEDED = 5
  class Dog
    def bark
      puts "Woof..
    end
  end
end
```

```
render :: Data -> Graphic

function render(d) {
    let d1 = process(d);
    return transform(d1);
}
```

Making Interoperation Safe



Typed Module

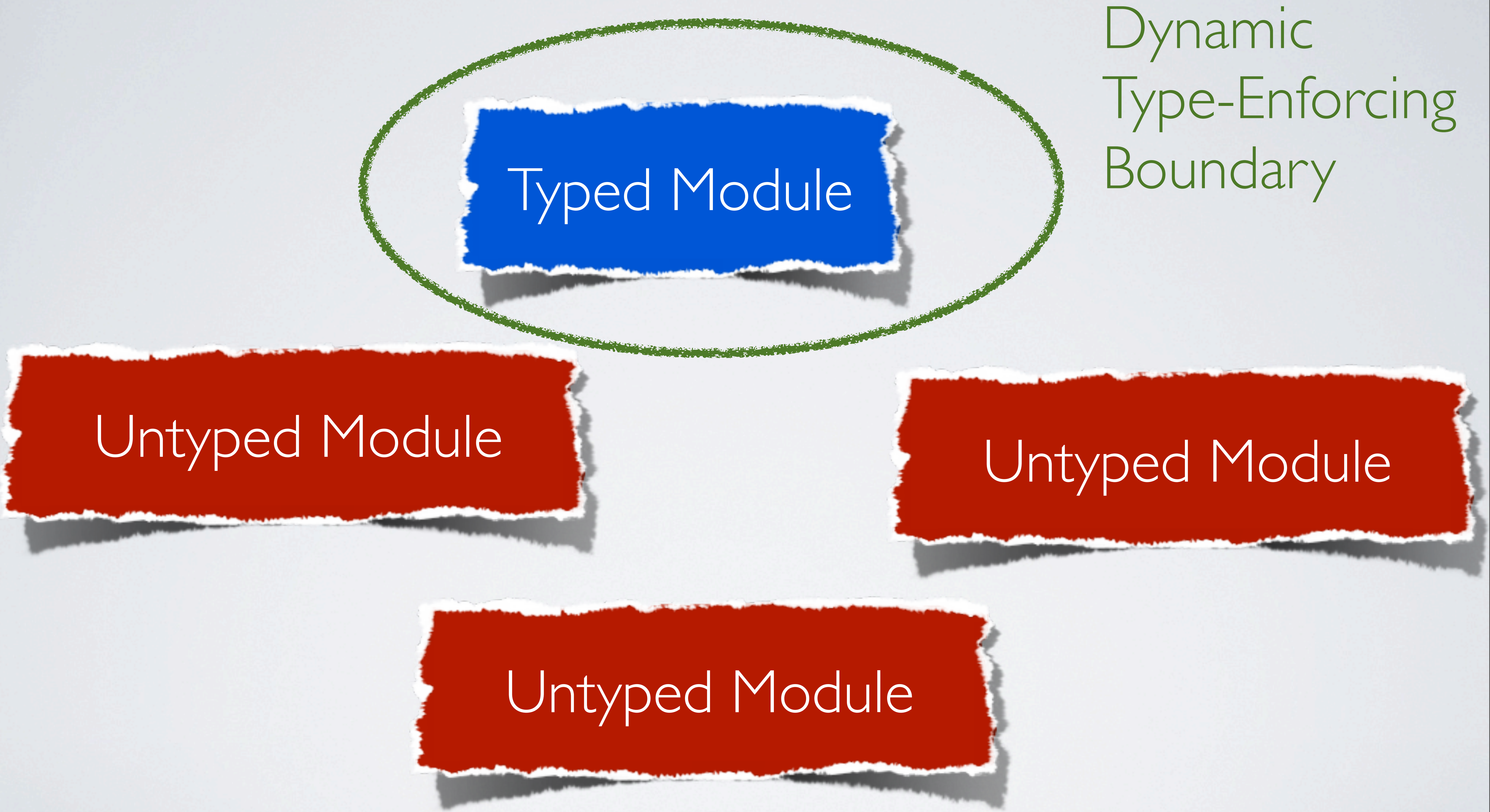
?

Untyped Module

Untyped Module

Untyped Module

Making Interoperation Safe



Dynamic
Type-Enforcing
Boundary

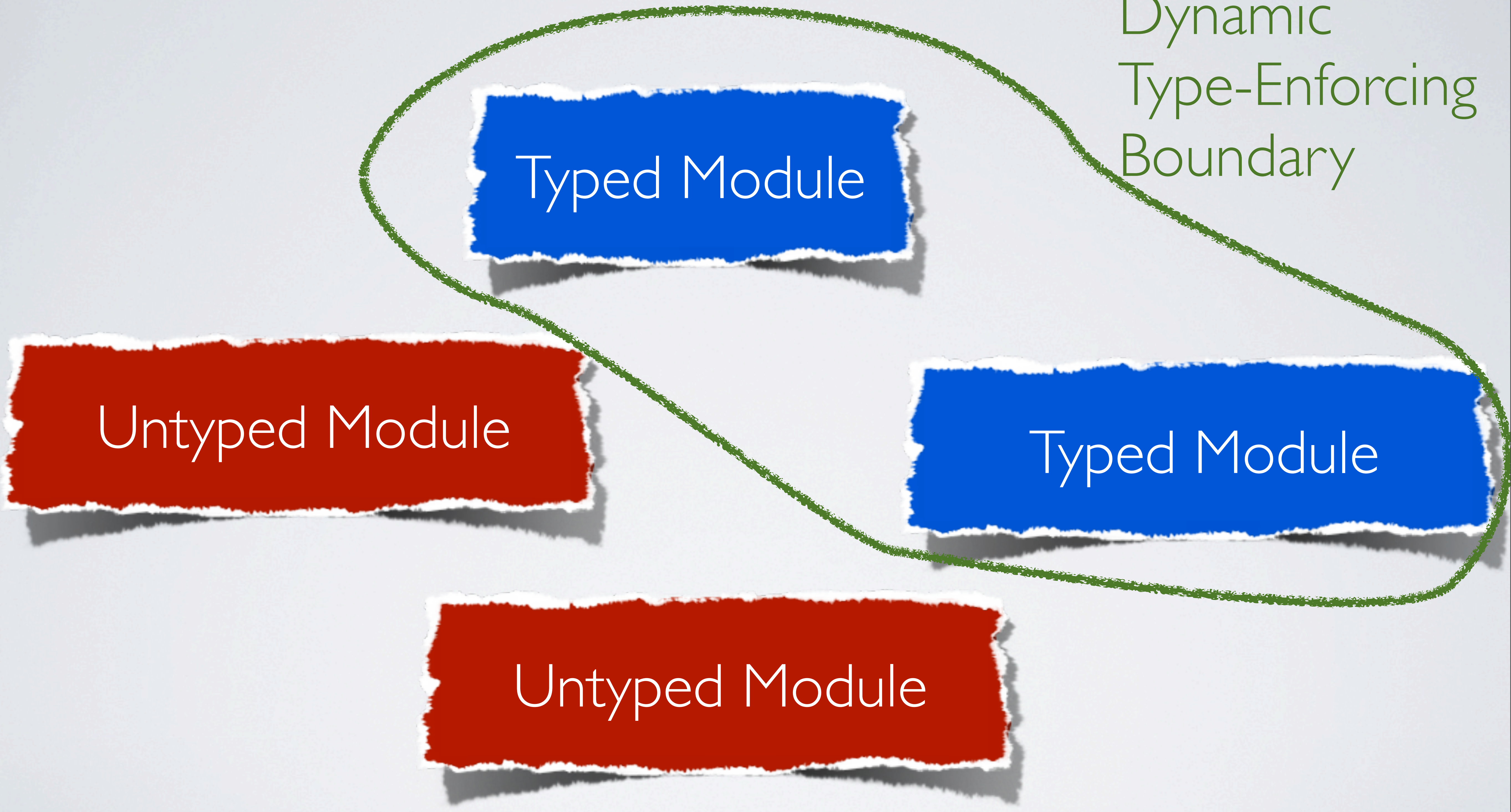
Typed Module

Untyped Module

Untyped Module

Untyped Module

Making Interoperation Safe



Dynamic Type-Enforcing Boundary

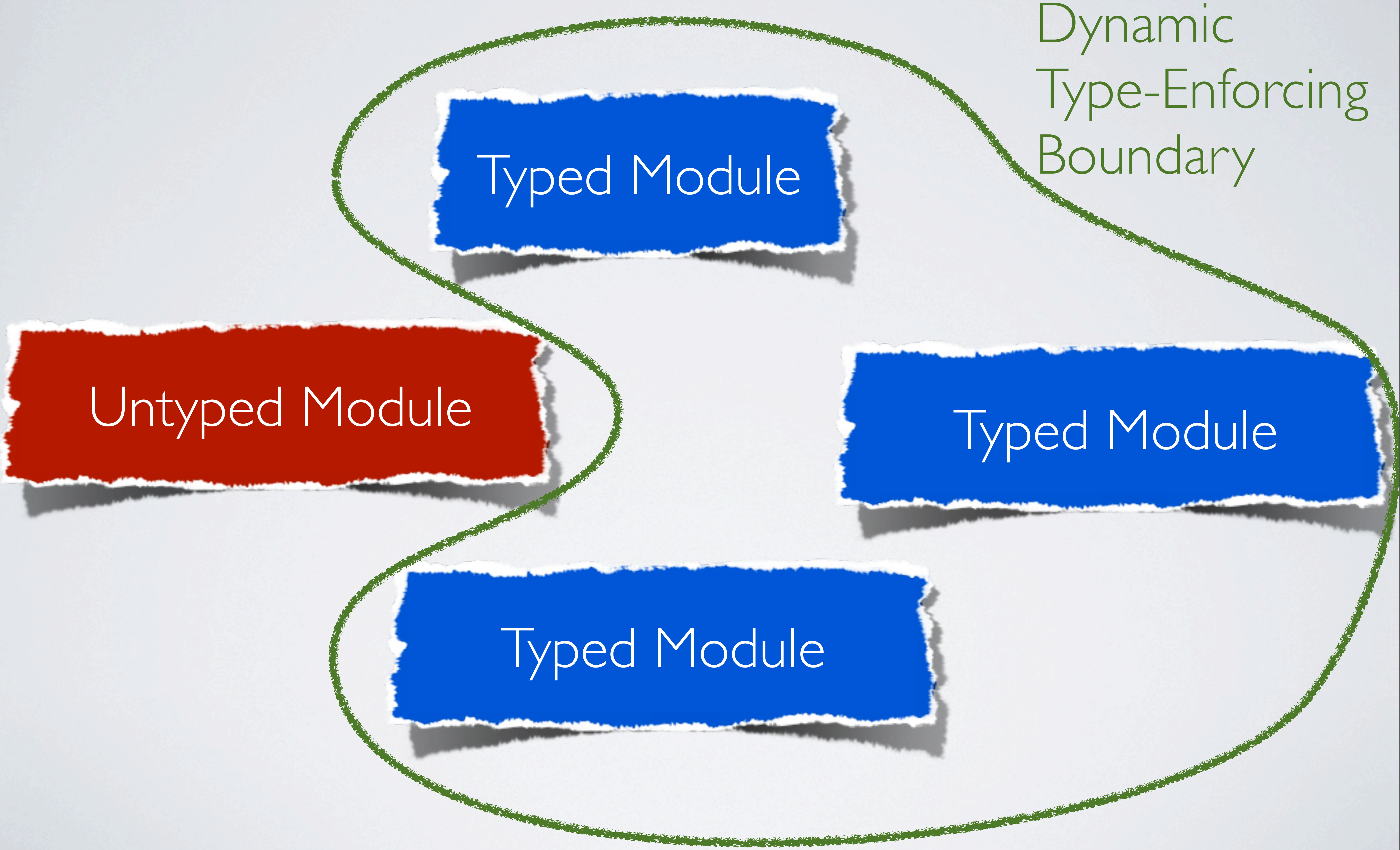
Typed Module

Untyped Module

Typed Module

Untyped Module

Making Interoperation Safe



Dynamically Enforcing Types

Static Type	Synthesized Dynamic Check
`Number`	`is_numeric`
`Listof[String]`	`s.all(is_string)`

Dynamically Enforcing Types

Static Type	Synthesized Dynamic Check
`Number`	`is_numeric`
`Listof[String]`	`s.all(is_string)`
`InFile -> OutFile`	preconditions/postconditions

server

```
#lang      racket


(define (add5 x) (+ x 5))
```

client

```
#lang      racket

(require server)
(add5 7)
```

server

```
#lang racket

(define (add5 x) (+ x 5))
```

client

```
#lang racket

(require server)
(add5 ”seven”)
```

```
+: expected number, but got “seven”
```

server

```
#lang typed/racket

(: add5 : Number -> Number)
(define (add5 x) (+ x 5))
```

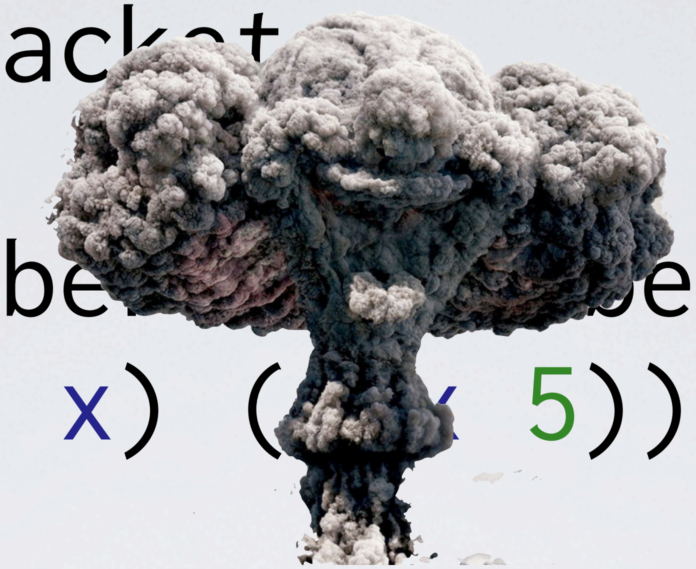
client

```
#lang       racket

(require server)
(add5 ”seven”)
```

+: expected number, but got “seven”

server

```
#lang typed/rack t

(: add5 : Numbe     er)
(define (add5 x) (    5))
```



client

```
#lang         racket

(require server)
(add5 "seven")
```

+: expected number, but got “seven”

```
#lang typed/racket
```

server

```
(: add5 : Number -> Number)
(define (add5 x) (+ x 5))
```

```
#lang       racket
```

client

```
(require server)
(add5 ”seven”)
```

client broke the specification on add5

server

```
#lang        racket

(define (add5 x) ”x plus 5”)
```

client

```
#lang typed/racket

(require server
         [add5 (Number -> Number)])
(add5 7)
```

server interface broke the specification on add5

Dynamically Enforcing Types

Static Type	Synthesized Dynamic Check
`Number`	`is_numeric`
`Listof[String]`	`s.all(is_string)`
`InFile -> OutFile`	preconditions/postconditions
`(ℝ -> ℝ) -> (ℝ -> ℝ)`	

Dynamically Enforcing Types

Static Type	Synthesized Dynamic Check
`Number`	`is_numeric`
`Listof[String]`	`s.all(is_string)`
`InFile -> OutFile`	preconditions/postconditions
`(ℝ -> ℝ) -> (ℝ -> ℝ)`	higher-order contracts

[Findler & Felleisen ICFP 02]

server

```
#lang typed/racket

(: deriv : (ℝ -> ℝ) -> (ℝ -> ℝ))
(define (deriv f) (lambda (x) ...))
```

client

```
#lang       racket

(require server)
(define cos (deriv sin))
(cos ”bad”)
```

```
#lang typed/racket                                server

(: deriv : (ℝ -> ℝ) -> (ℝ -> ℝ))
(define (deriv f) (lambda (x) ...))
```

```
#lang        racket                               client

(require server)
(define cos (deriv sin))
(cos "bad")
```

client broke the specification on deriv

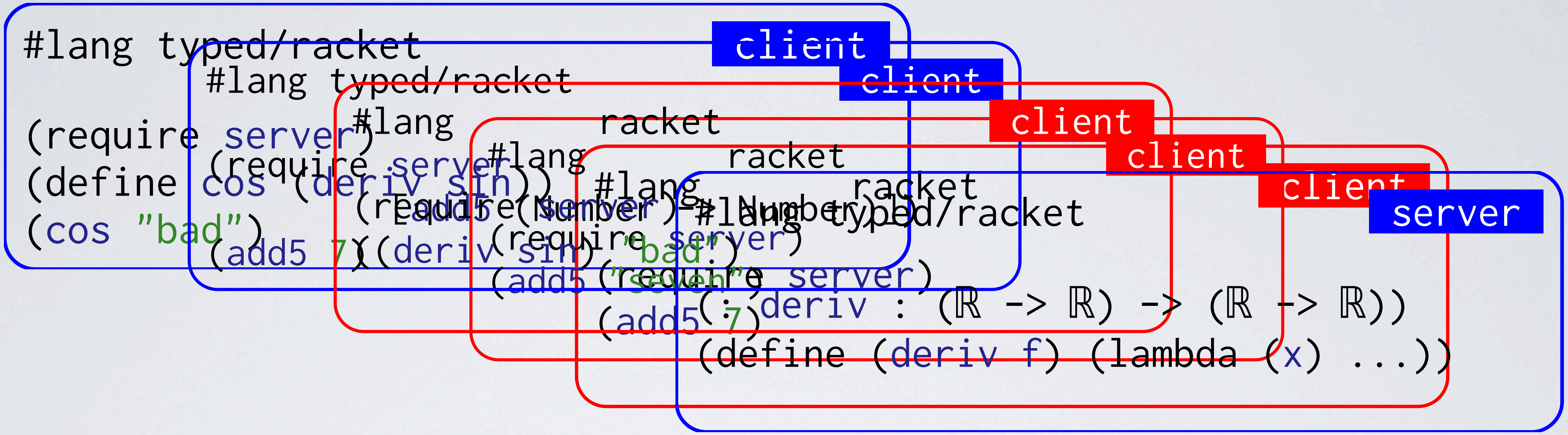
server

```
#lang typed/racket

(: deriv : (ℝ -> ℝ) -> (ℝ -> ℝ))
(define (deriv f) (lambda (x) ...))
```

client

```
#lang typed/racket

(require server)
(define cos (deriv sin))
(cos ”bad”)
```

typechecker: incorrect argument to deriv



Key Elements

Multi-language Infrastructure [PLDI 11]

Automatically Synthesizing Dynamic Checks from Types [DLS 06]

More Efficient, More Expressive Contracts [Work in progress]

Static Guarantees from Blame

```
server interface broke the specification on add5

client broke the specification on add5

client broke the specification on deriv
```

Static Guarantees from Blame

```
server interface broke the specification on add5

client broke the specification on add5

client broke the specification on deriv
```

Contracts and blame give us a soundness theorem:

Dynamic type errors always blame the untyped modules

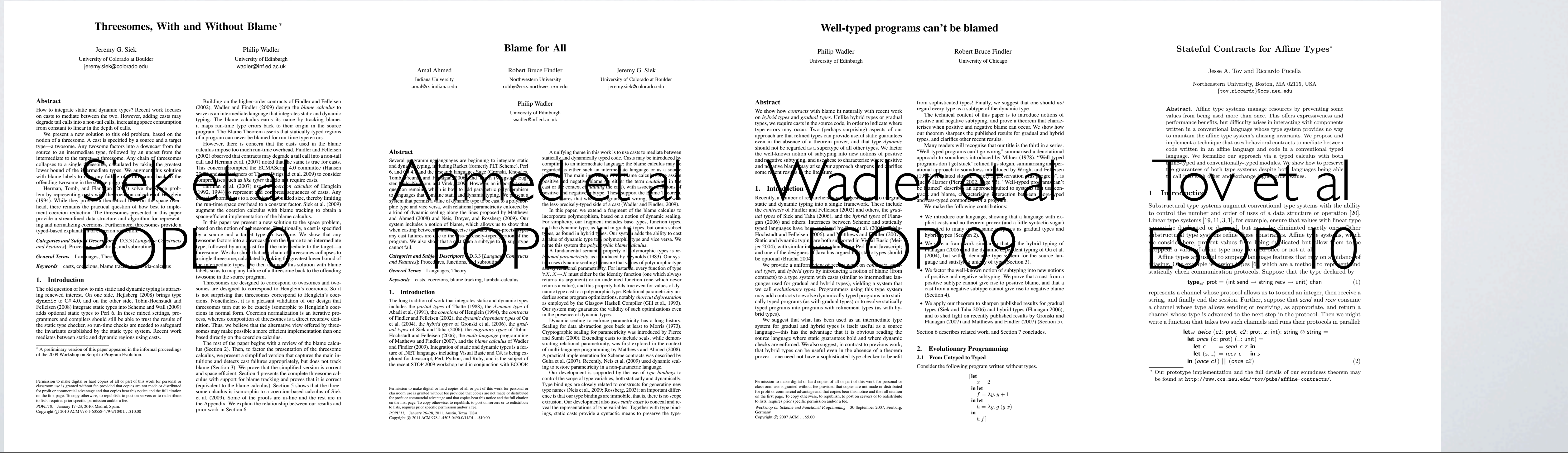
[DLS 2006]

Static Guarantees from Blame

Contracts and blame give us a soundness theorem:

Dynamic type errors always blame the untyped modules

[DLS 2006]



Threesomes, With and Without Blame

Jeremy G. Siek, University of Colorado at Boulder, jeremy.siek@colorado.edu
Philip Wadler, University of Edinburgh, wadler@inf.ed.ac.uk

Blame for All

Amal Ahmed, Indiana University, amal@cs.indiana.edu
Robert Bruce Findler, Northwestern University, robby@eecs.northwestern.edu
Jeremy G. Siek, University of Colorado at Boulder, jeremy.siek@colorado.edu
Philip Wadler, University of Edinburgh, wadler@inf.ed.ac.uk

Well-typed programs can't be blamed

Philip Wadler, University of Edinburgh
Robert Bruce Findler, University of Chicago

Stateful Contracts for Affine Types

Jesse A. Tov and Riccardo Pucella
Northeastern University, Boston, MA 02115, USA
{tov,riccardo}@ccs.neu.edu

Siek et al POPL'10

Ahmed et al POPL'11

Wadler et al ESOP'09

Tov et al ESOP'10

Why Multilanguage Soundness?

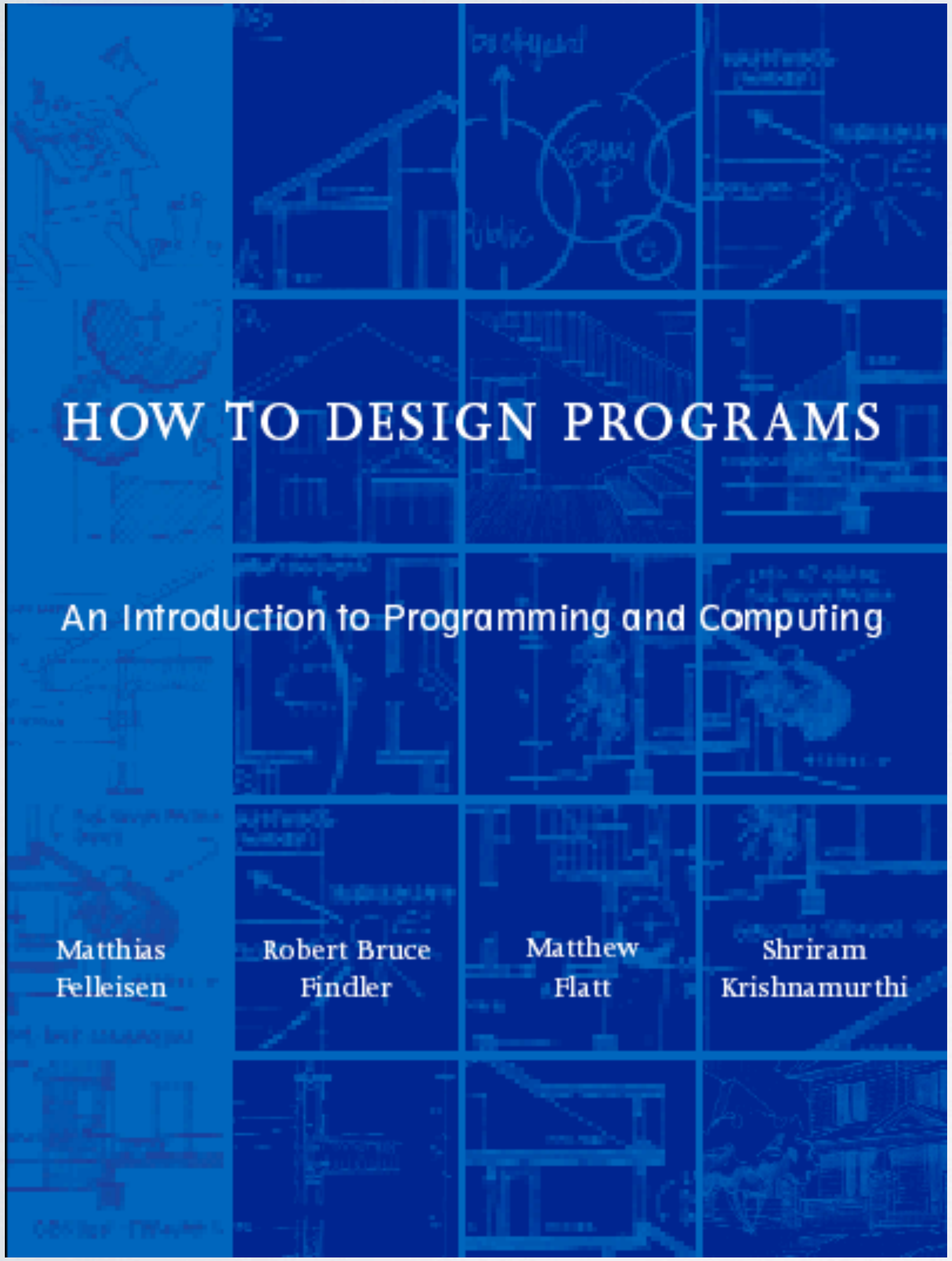
Support local reasoning

Static guarantee only depends on typed modules

Tunable levels of checking

Types for Untyped Languages

All programmers reason about their programs



HOW TO DESIGN PROGRAMS

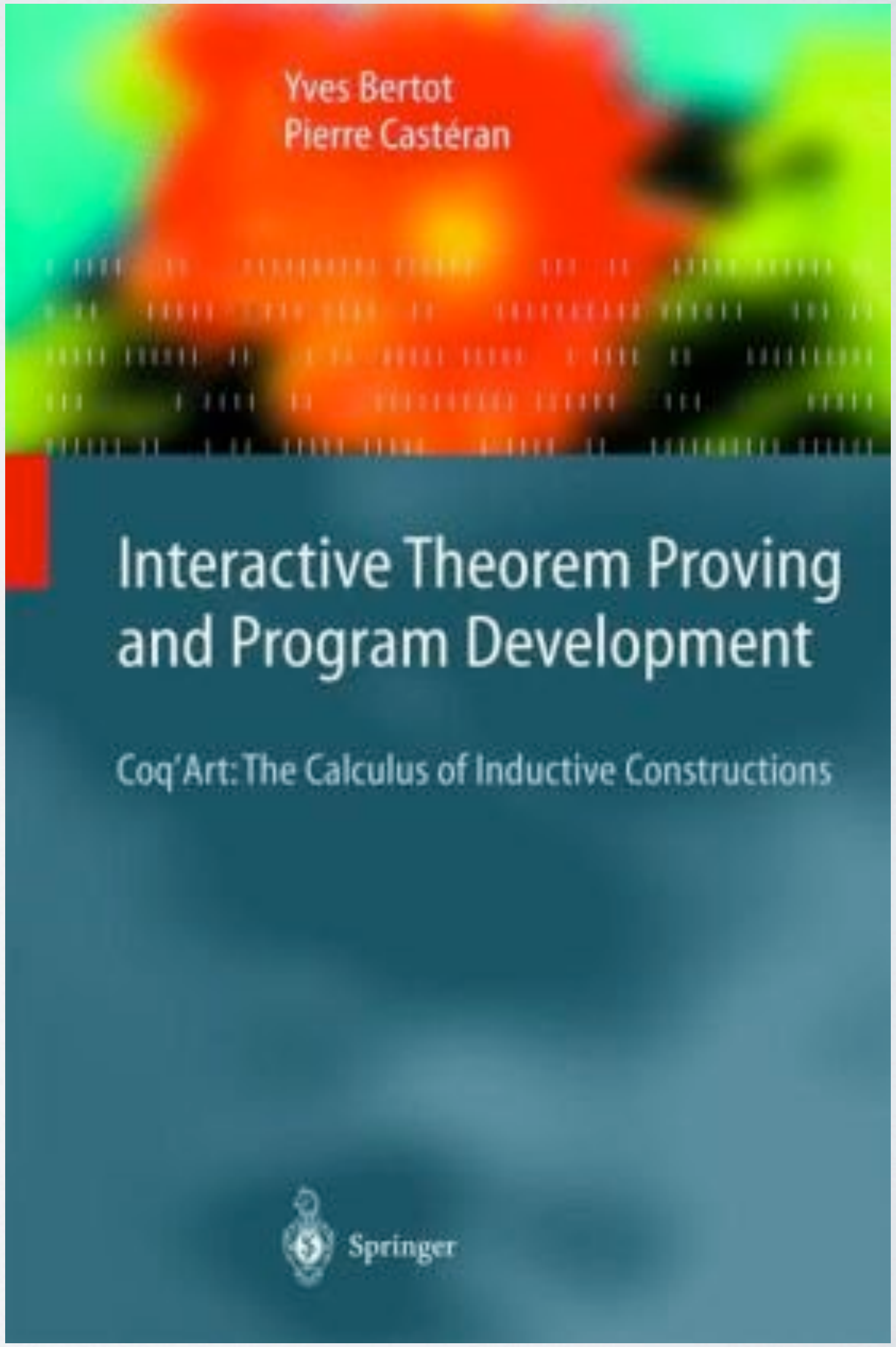
An Introduction to Programming and Computing

Matthias Felleisen

Robert Bruce Findler

Matthew Flatt

Shriram Krishnamurthi



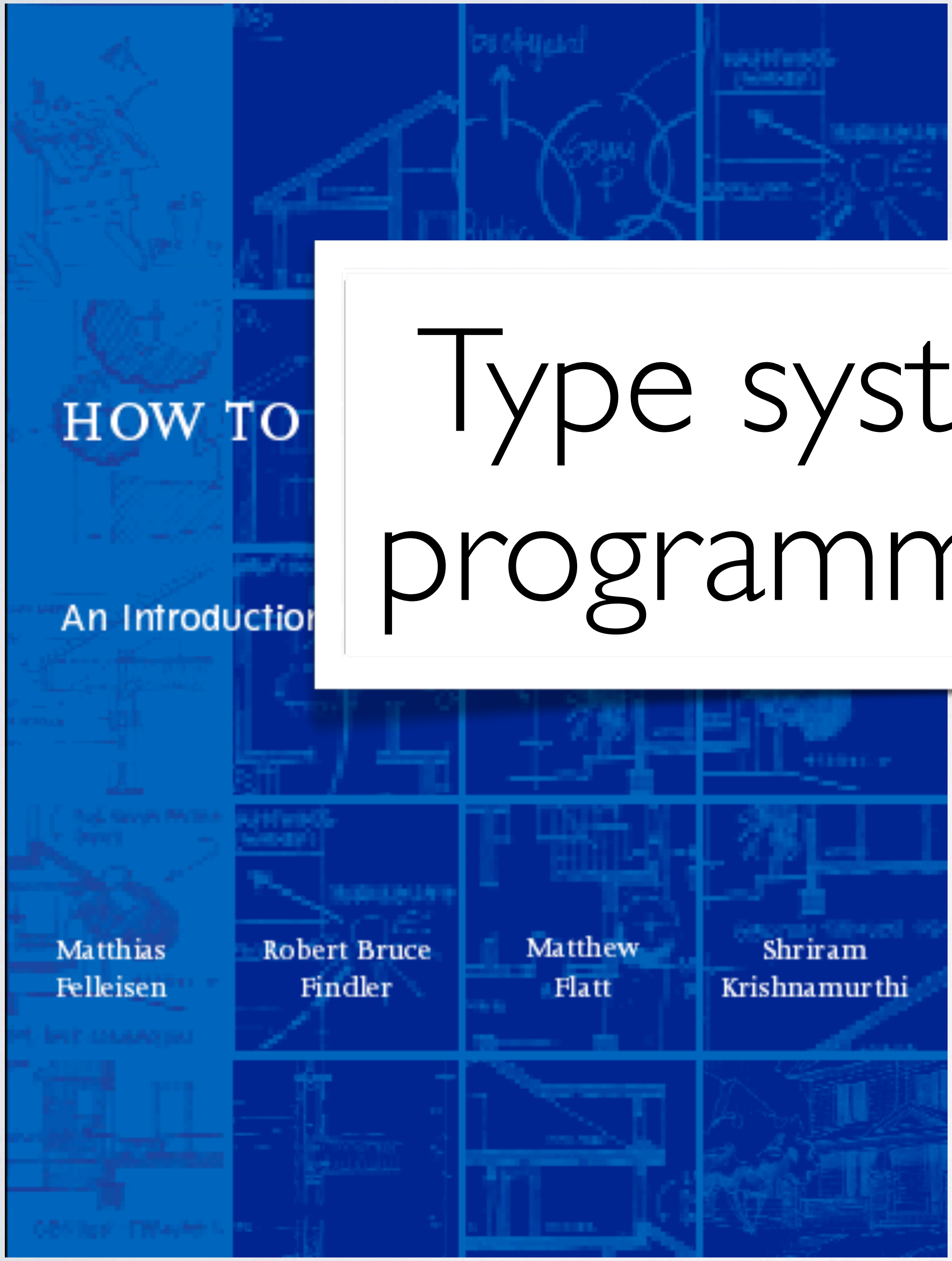
Yves Bertot
Pierre Castéran

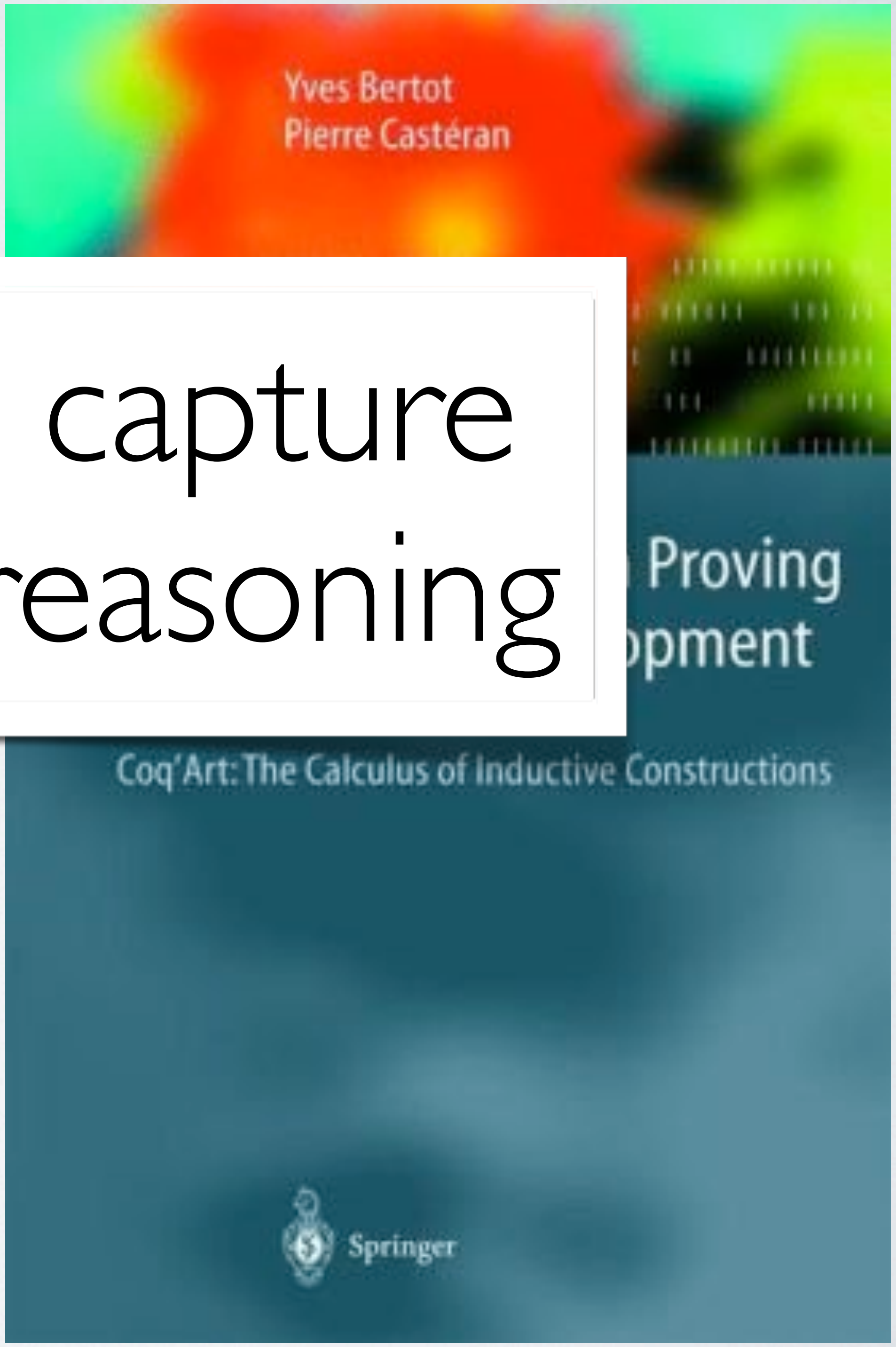
Interactive Theorem Proving and Program Development

Coq'Art: The Calculus of Inductive Constructions

Springer

All programmers reason about their programs





Type systems capture programmer reasoning

Programs in Lua don’t use the Java type system

Perl ML

Python Haskell

Ruby Scala

Programs in Lua don’t use the Java type system

Clojure C#

Javascript C++

PHP Pascal

Perl ML

Solution: design a type system based on the existing idioms of the language

PHP Pascal

Types for Existing Programs

Unions, Structures, Polymorphism	Standard
Occurrence Typing	[POPL 08] [ICFP 10]
Refinement Types	[HOSC 11]
Variable-Arity	[ESOP 09]
Numerics	in preparation

Types for Existing Programs

Unions, Structures, Polymorphism	Standard
Occurrence Typing	[POPL 08] [ICFP 10]
Refinement Types	[HOSC 11]
Variable-Arity	[ESOP 09]
Numerics	in preparation

Dynamic Type Tests

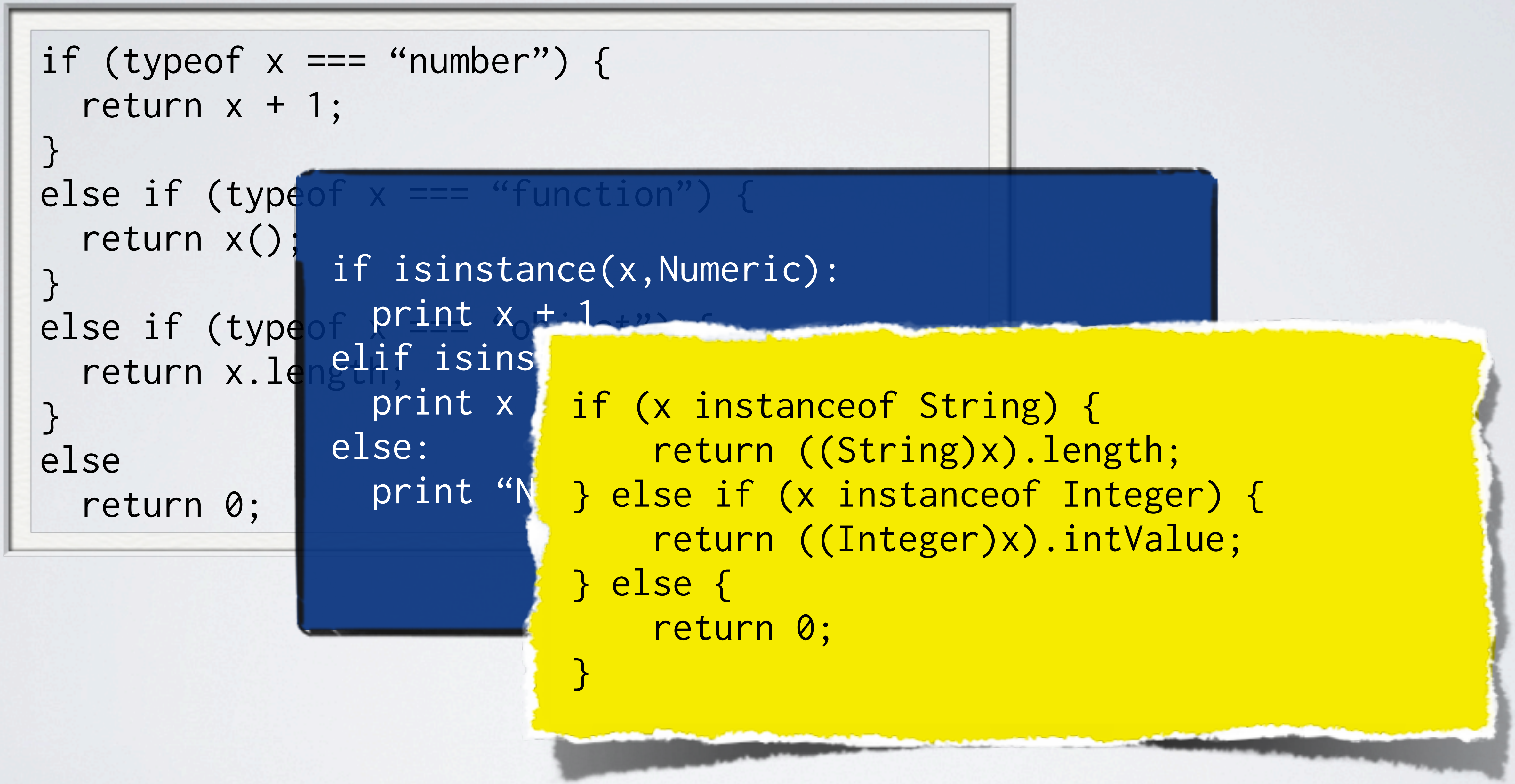
```
if (typeof x === “number”) {
  return x + 1;
}
else if (typeof x === “function”) {
  return x();
}
else if (typeof x === “object”) {
  return x.length;
}
else
  return 0;
```

Dynamic Type Tests

```
if (typeof x === “number”) {
  return x + 1;
}
else if (typeof x === “function”) {
  return x();
}
else if (typeof x === “object”) {
  return x.length;
}
else
  return 0;
```

```
if isinstance(x,Numeric):
  print x + 1
elif isinstance(x,String):
  print x
else:
  print “Nothing”
```

Dynamic Type Tests



```
if (typeof x === “number”) {
  return x + 1;
}
else if (typeof x === “function”) {
  return x();
}
else if (typeof x === “object”) {
  return x.length;
}
else
  return 0;
```

```
if isinstance(x,Numeric):
  print x + 1
elif isins
  print x
else:
  print “N
```

```
if (x instanceof String) {
    return ((String)x).length;
} else if (x instanceof Integer) {
    return ((Integer)x).intValue;
} else {
    return 0;
}
```

```
;; sum : BT -> Number
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
               (sum (left bt))
               (sum (right bt)))]))
```

```
(define-type BT (U Number (Pair BT BT)))


(: sum : BT -> Number)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
               (sum (left bt))
               (sum (right bt)))]))
```

```
(define-type BT (U Number (Pair BT BT)))
```

bt : BT

```
(: sum : BT -> Number)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
               (sum (left bt))
               (sum (right bt)))]))
```

```
(define-type BT (U Number (Pair BT BT)))


(: sum : BT -> Number)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
                (sum (left bt))
                (sum (right bt)))]))
```

bt : BT

bt : Number

bt : (Pair BT BT)

```
(define-type BT (U Number (Pair BT BT)))

(: sum : BT -> Number)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
                 sum (left bt))
                 sum (right bt)))]))
```

bt : BT

bt : Number

number? : $\text{Any} \xrightarrow{\text{Number}} \text{Bool}$

```
(define-type BT (U Number (Pair BT BT)))

(: sum : BT -> Number)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+ (sum (left bt))
                 (sum (right bt)))]))
```

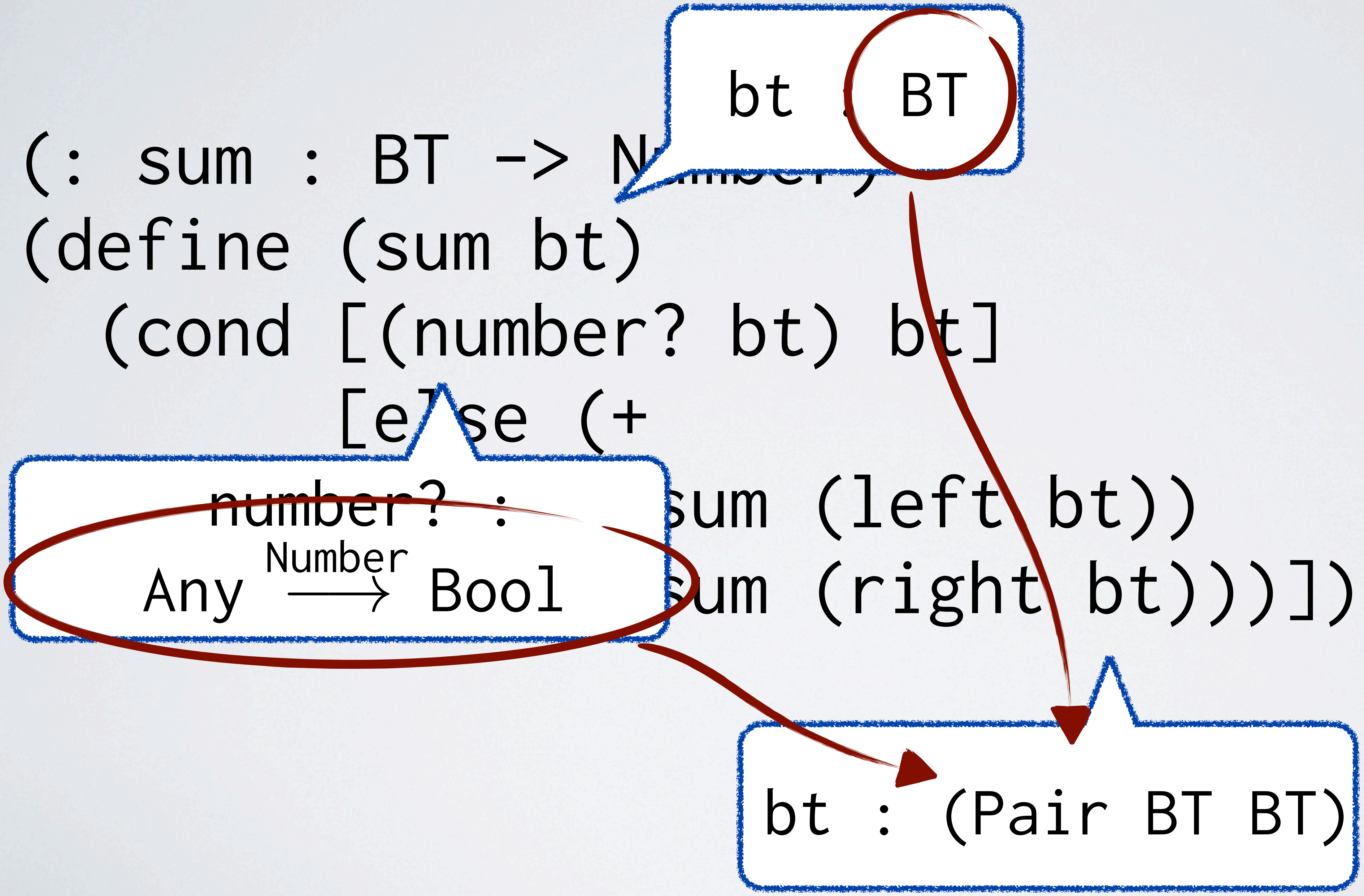
bt : BT

bt : Number

number? :

Any $\xrightarrow{\text{Number}}$ Bool

```
(define-type BT (U Number (Pair BT BT)))


(: sum : BT -> N      umber)
(define (sum bt)
  (cond [(number? bt) bt]
        [else (+
                sum (left bt))
                sum (right bt)))]))
```



bt : BT

number? : Any $\xrightarrow{\text{Number}}$ Bool

bt : (Pair BT BT)

```
(map rectangle-area
     (filter rectangle? list-of-shapes))
```

`filter :`

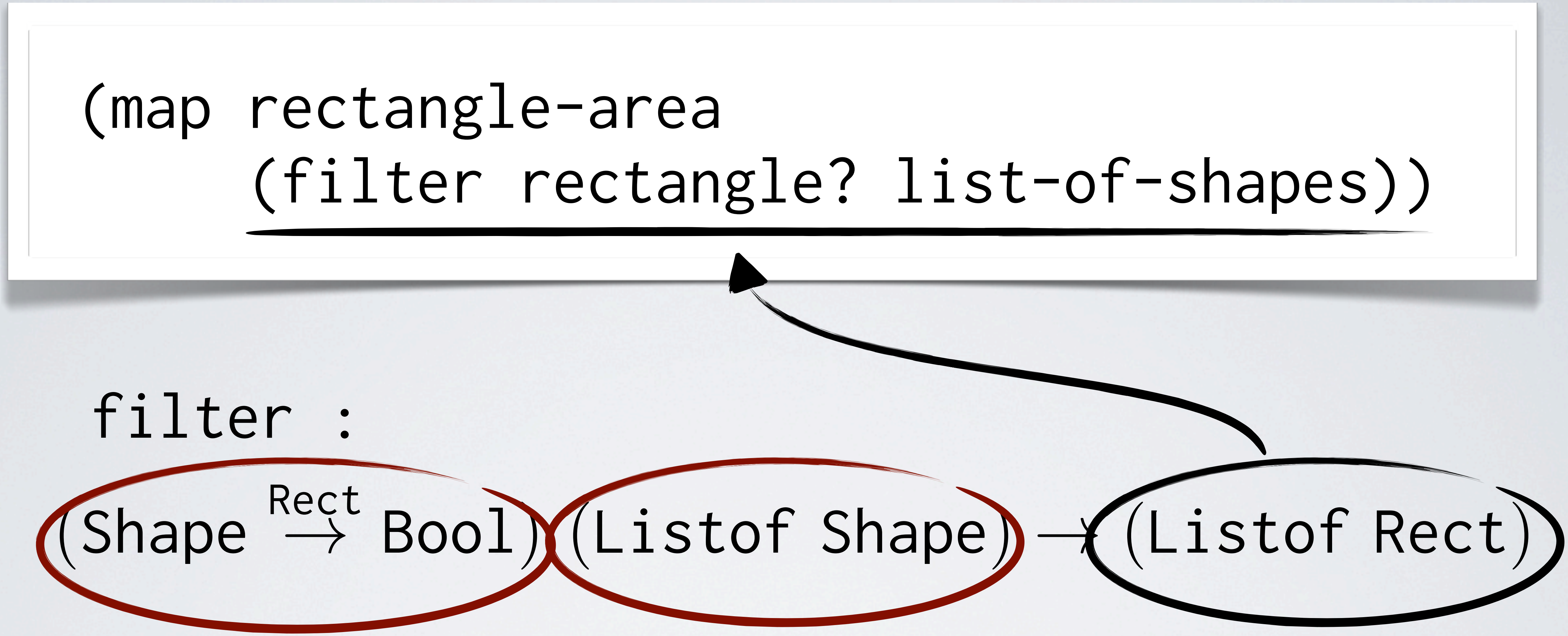
$$\forall \alpha\beta.(\alpha \overset{\beta}{\rightarrow} \texttt{Bool})\ (\texttt{Listof}\ \alpha) \rightarrow (\texttt{Listof}\ \beta)$$

```
(map rectangle-area
     (filter rectangle? list-of-shapes))
```

filter :

$$(\text{Shape} \xrightarrow{\text{Rect}} \text{Bool})\ (\text{Listof Shape}) \to (\text{Listof Rect})$$

$$\forall \alpha\beta.(\alpha \xrightarrow{\beta} \text{Bool})\ (\text{Listof } \alpha) \to (\text{Listof } \beta)$$



```
(map rectangle-area
     (filter rectangle? list-of-shapes))
```

filter :

$(\text{Shape} \xrightarrow{\text{Rect}} \text{Bool})\ (\text{Listof Shape}) \to (\text{Listof Rect})$

$$\forall\alpha\beta.(\alpha \xrightarrow{\beta} \text{Bool})\ (\text{Listof}\ \alpha) \to (\text{Listof}\ \beta)$$

Key Idea 1:
A logic to prove facts
about variables and types

$$\textsc{L-Sub}\quad \frac{\Gamma \vdash \tau_x \qquad \vdash \tau <: \sigma}{\Gamma \vdash \sigma_x}$$

Key Idea 1:
A logic to prove facts
about variables and types

Key Idea 2:
An environment of
general propositions

$$\textsc{L-Sub}\quad \frac{\Gamma \vdash \tau_x \qquad \vdash \tau <: \sigma}{\Gamma \vdash \sigma_x}$$

$$\textsc{T-Var}\quad \frac{\Gamma \vdash \tau_x}{\Gamma \vdash x : \tau \; ; \; \mathbf{\#f}_x | \mathbf{\#f}_x \; ; \; x}$$

Key Idea 1:
A logic to prove facts
about variables and types

L-SUB

$$\frac{\Gamma \vdash \tau_x \qquad \vdash \tau <: \sigma}{\Gamma \vdash \sigma_x}$$

Key Idea 2:
An environment of
general propositions

T-VAR

$$\frac{\Gamma \vdash \tau_x}{\Gamma \vdash x : \tau \; ; \; \mathbf{\#f}_x | \mathbf{\#f}_x \; ; \; x}$$

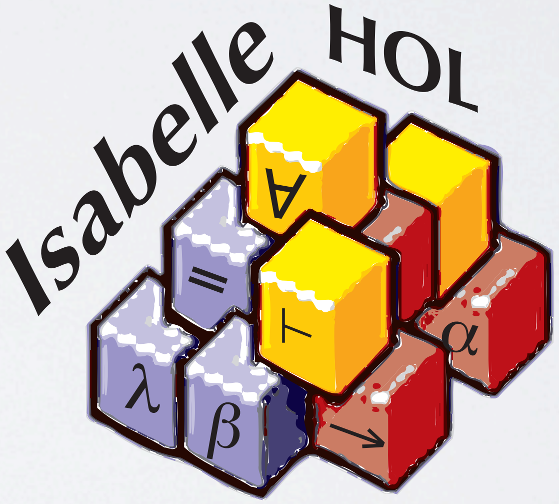
Result:
Rich type system that can
follow sophisticated reasoning

Soundness: if $\mathsf{e} : \tau$ and $\mathsf{e} \rightarrow \mathsf{v}$, then $\mathsf{v} : \tau$

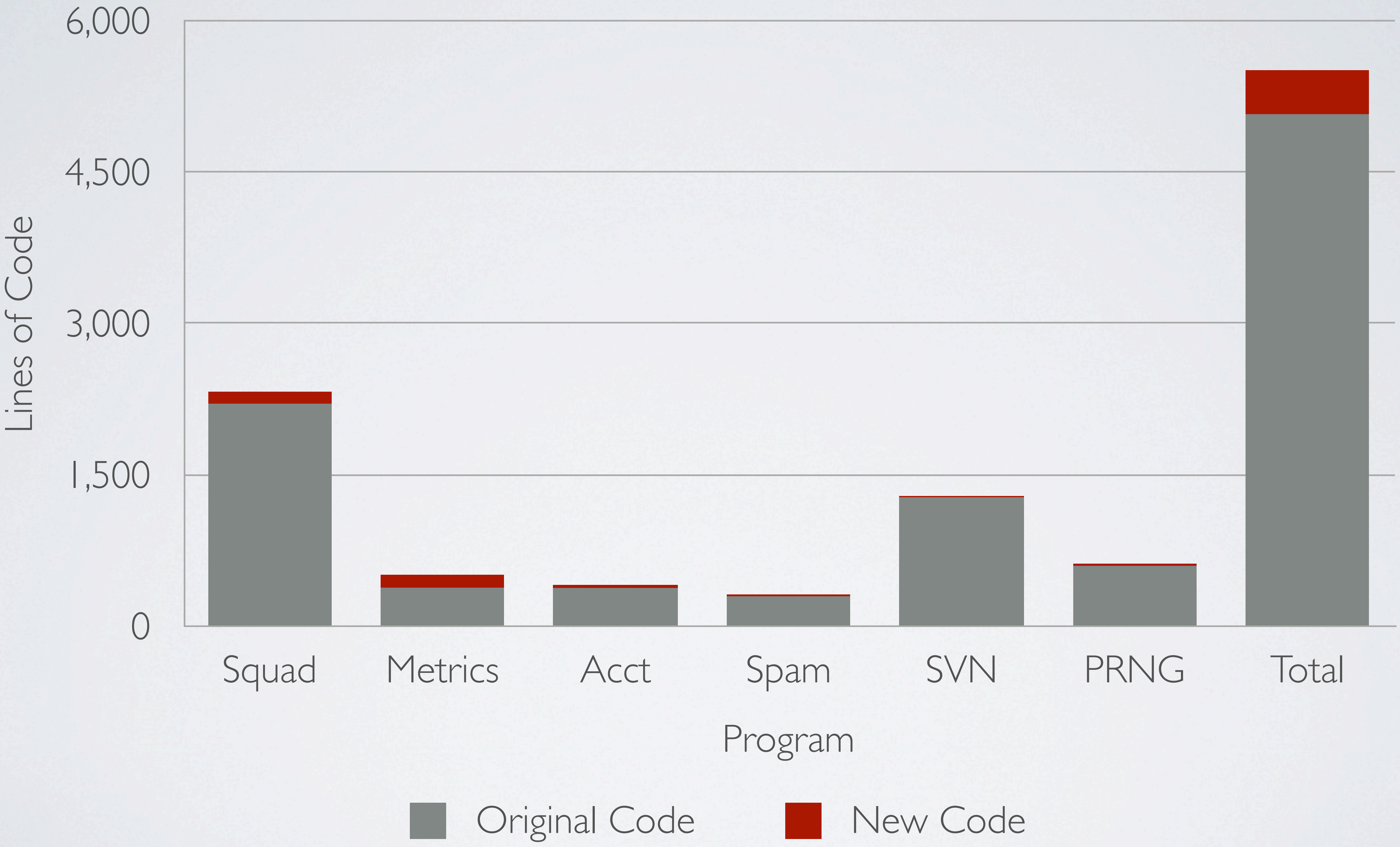
In other words, we can trust our types.

Semantics Engineering with PLT Redex

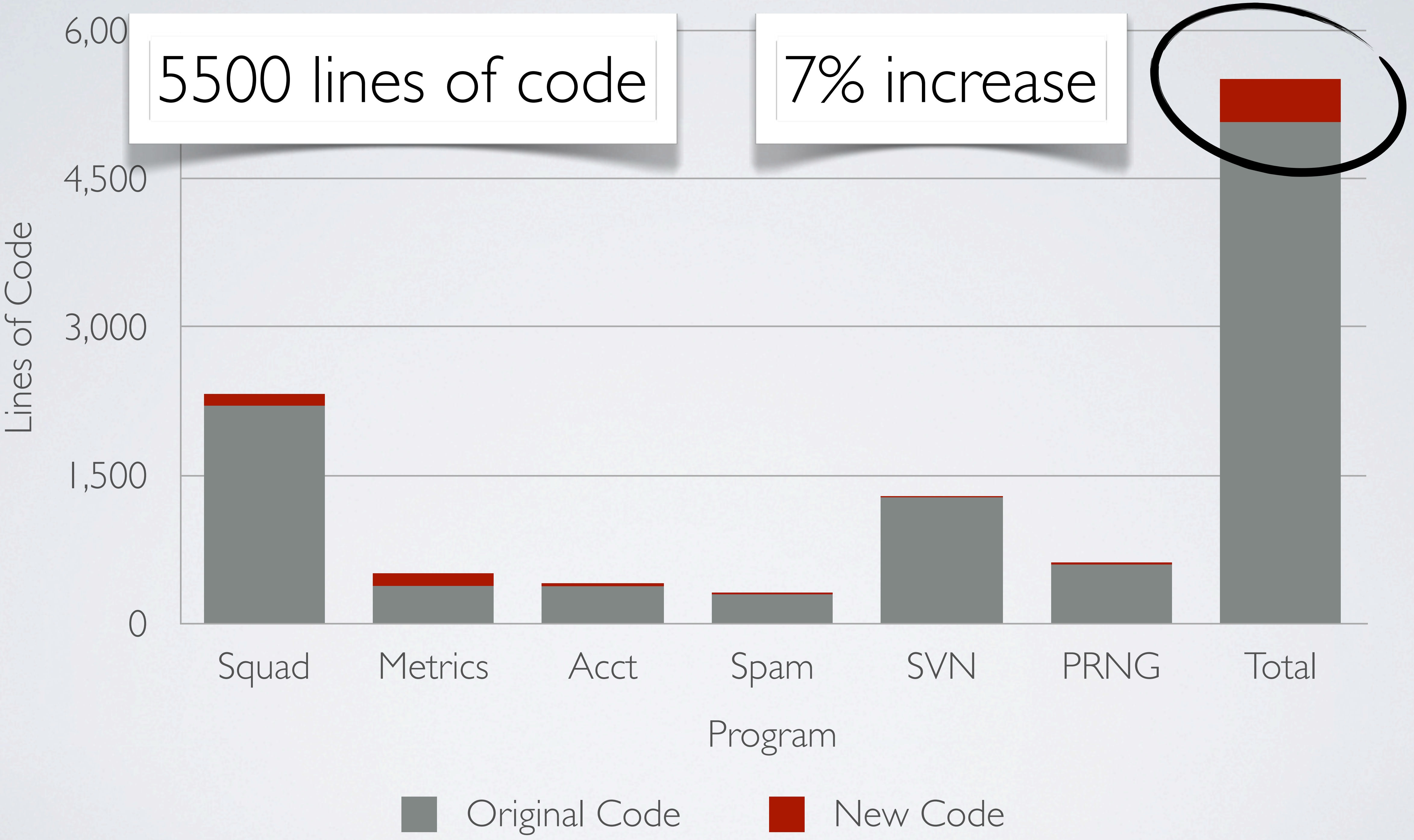
Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt



Validation: Existing Code



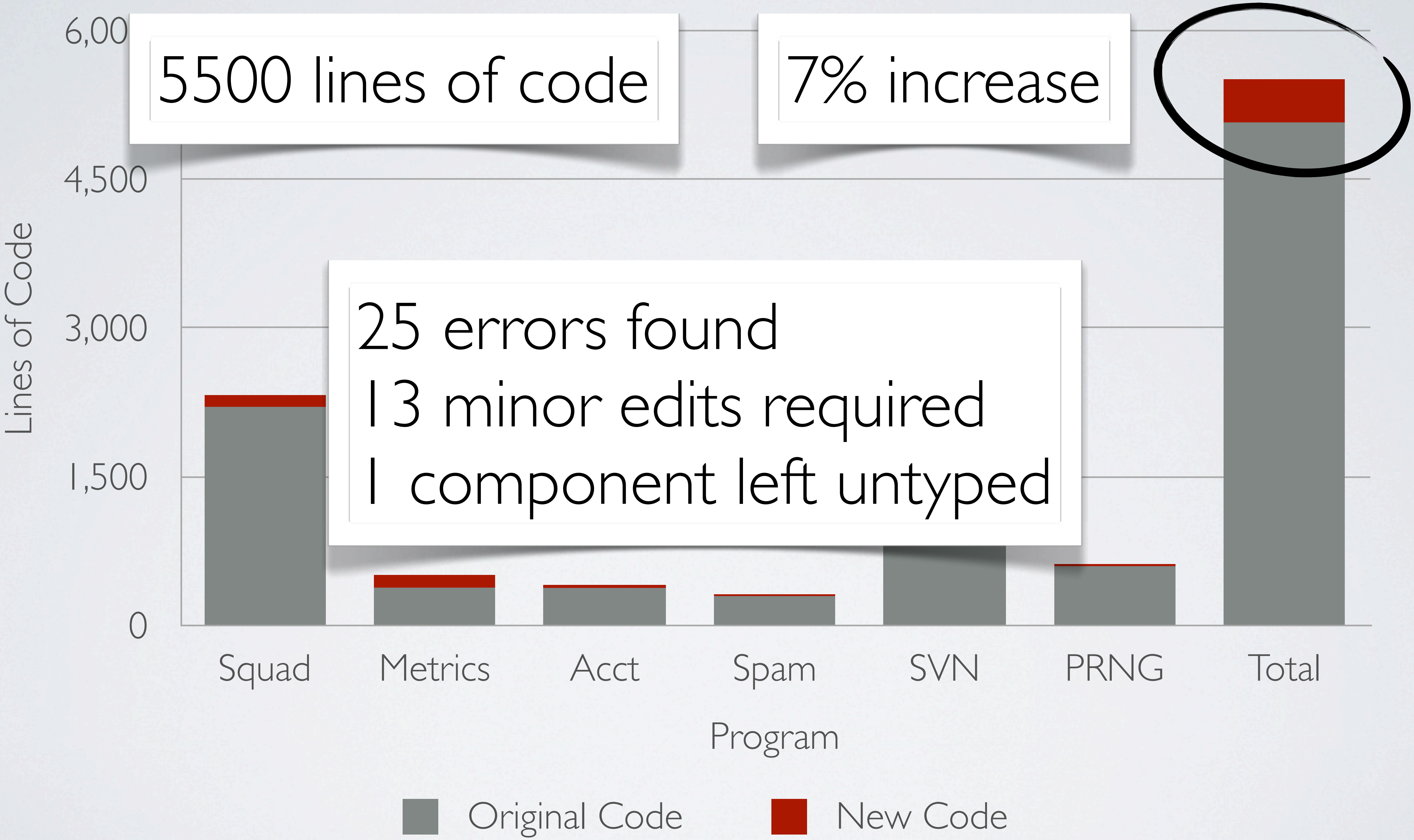
Validation: Existing Code



5500 lines of code

7% increase

Validation: Existing Code



5500 lines of code

7% increase

25 errors found
13 minor edits required
1 component left untyped

Validation: Comparative

```
fun balance T (B, T(R, T(R, a, x, b), y, c), z, d) = T(R, T(B, a, x, b), y, T(B, c, z, d))
  | balance T (B, T(R, a, x, T(R, b, y, c)), z, d) = T(R, T(B, a, x, b), y, T(B, c, z, d))
  | balance T (B, a, x, T(R, T(R, b, y, c), z, d)) = T(R, T(B, a, x, b), y, T(B, c, z, d))
  | balance T (B, a, x, T(R, b, y, T(R, c, z, d))) = T(R, T(B, a, x, b), y, T(B, c, z, d))
  | balance T body = T body
```

```
(define (balance tree)
  (match tree
    [(T B (T R (T R a x b) y c) z d)   (T R (T B a x b) y (T B c z d))]
    [(T B (T R a x (T R b y c)) z d)   (T R (T B a x b) y (T B c z d))]
    [(T B a x (T R (T R b y c) z d))   (T R (T B a x b) y (T B c z d))]
    [(T B a x (T R b y (T R c z d)))   (T R (T B a x b) y (T B c z d))]
    [else tree]))
```

[Prashanth Thesis 2011]

Contracts to Dynamically Enforce Types

Blame for Soundness

DLS 2006, STOP 2009

Contracts to Dynamically Enforce Types

Blame for Soundness

Type System for Language Idioms

Validation on Existing Programs

POPL 2008, ESOP 2009, ICFP 2010, HOSC 2011, Prashanth Thesis

Contracts to Dynamically Enforce Types

Blame for Soundness

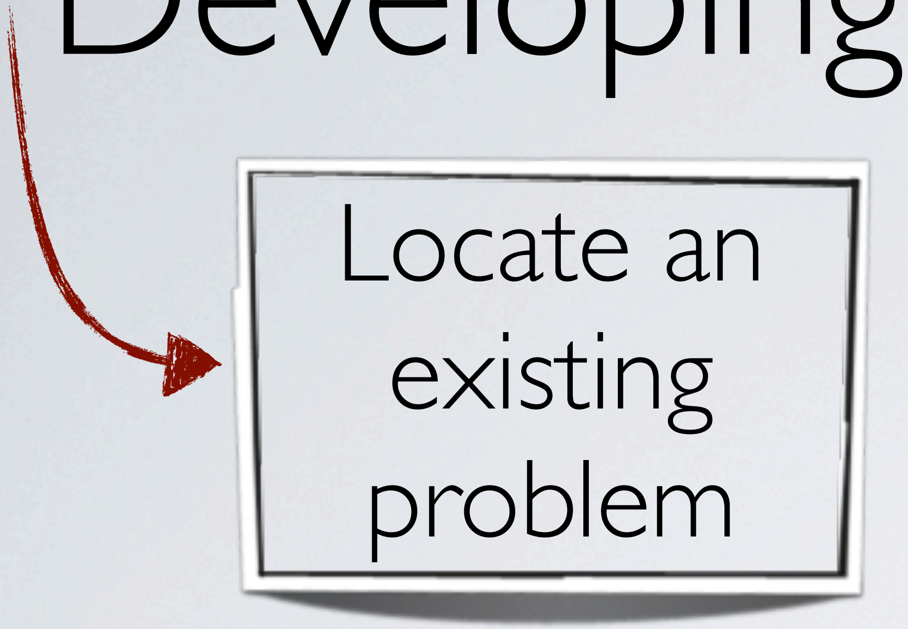
Type System for Language Idioms

Validation on Existing Programs

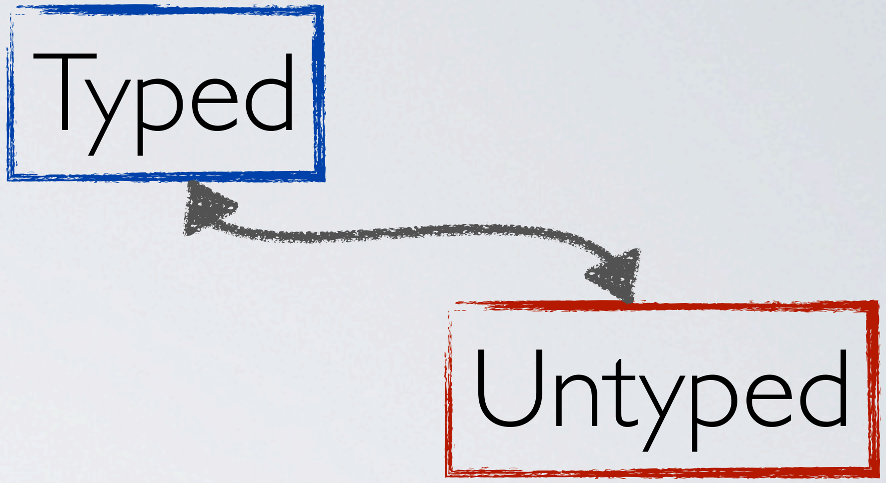
Multilanguage Development Infrastructure

Scheme 2007, PLDI 2011

Developing a solution



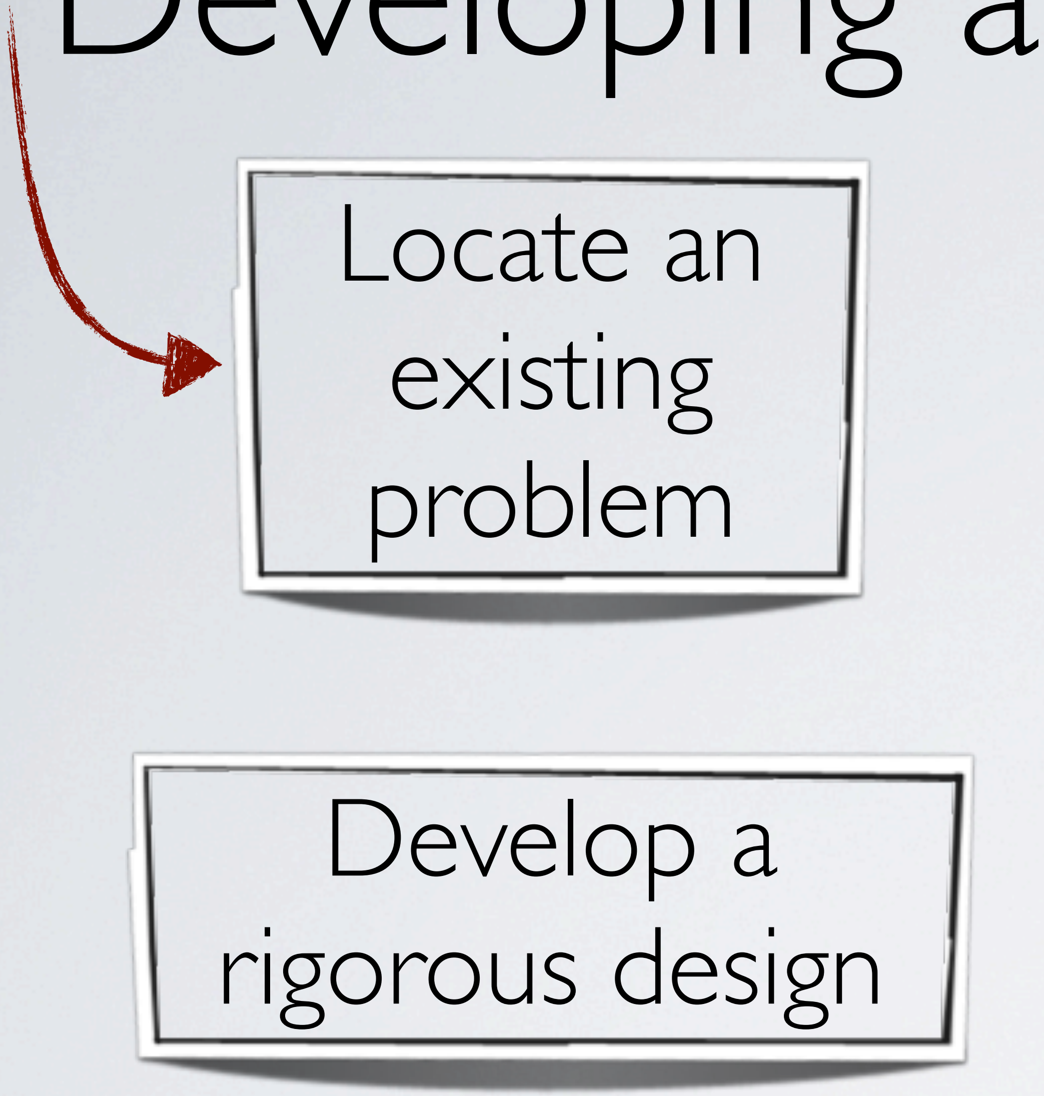
Locate an existing problem



Typed

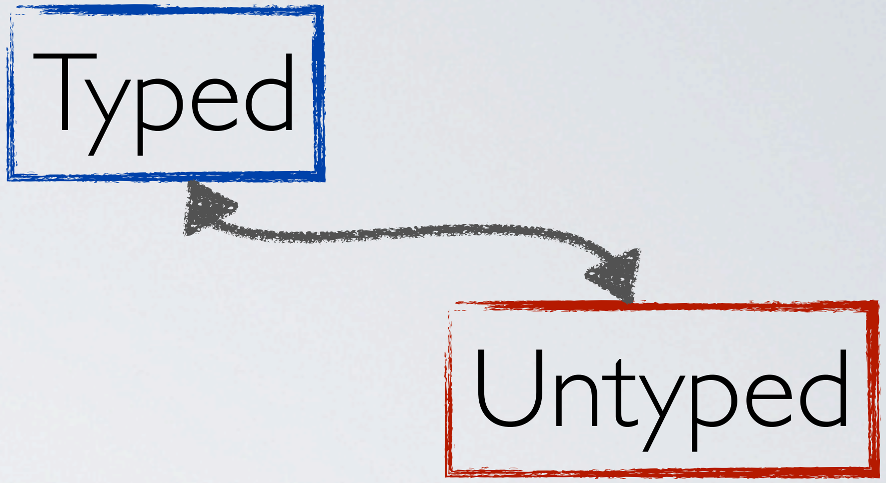
Untyped

Developing a solution



Locate an existing problem

Develop a rigorous design

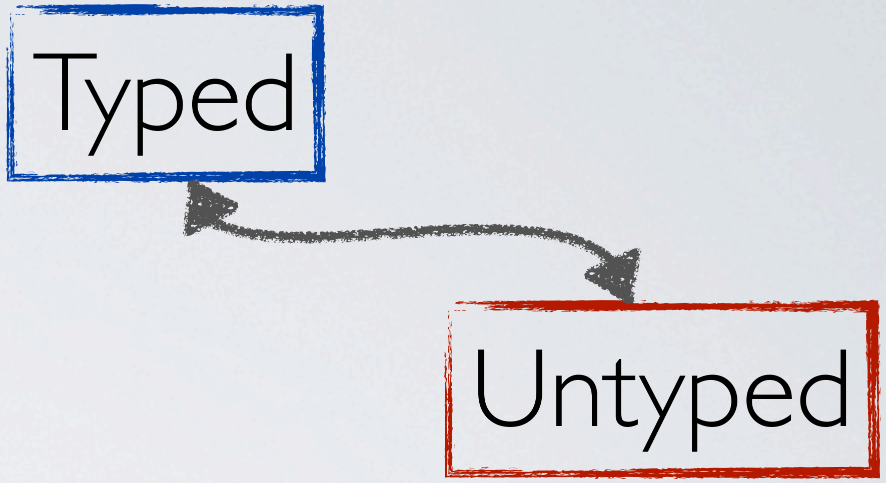


ST-Abs

$$\frac{\Gamma, x : t \vdash^{ST} e : s; e'}{\Gamma \vdash^{ST} (\lambda x : t.e) : (t \rightarrow s); (\lambda x : t.e')}$$

Developing a solution

Locate an existing problem



Develop a rigorous design

$$\textsc{ST-Abs}\quad \frac{\Gamma, x : t \vdash^{ST} e : s; e'}{\Gamma \vdash^{ST} (\lambda x : t.e) : (t \rightarrow s); (\lambda x : t.e')}$$

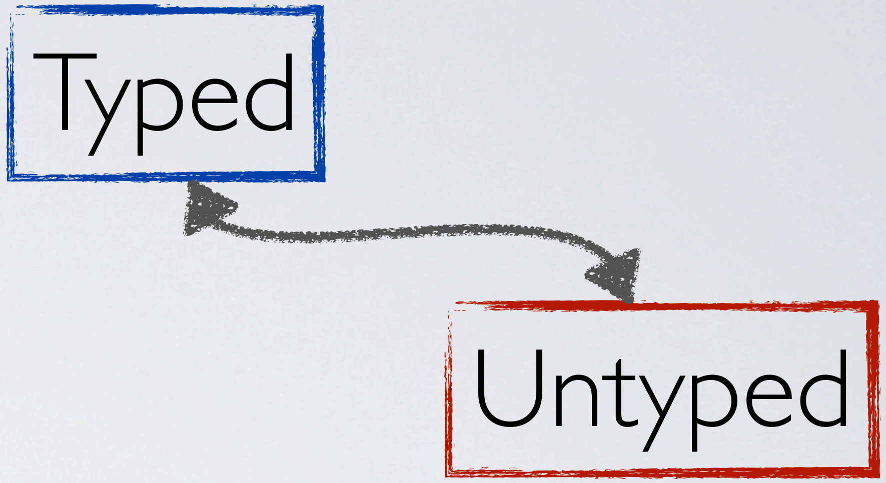
Validate by implementation & experiment

Developing a solution

Locate an existing problem

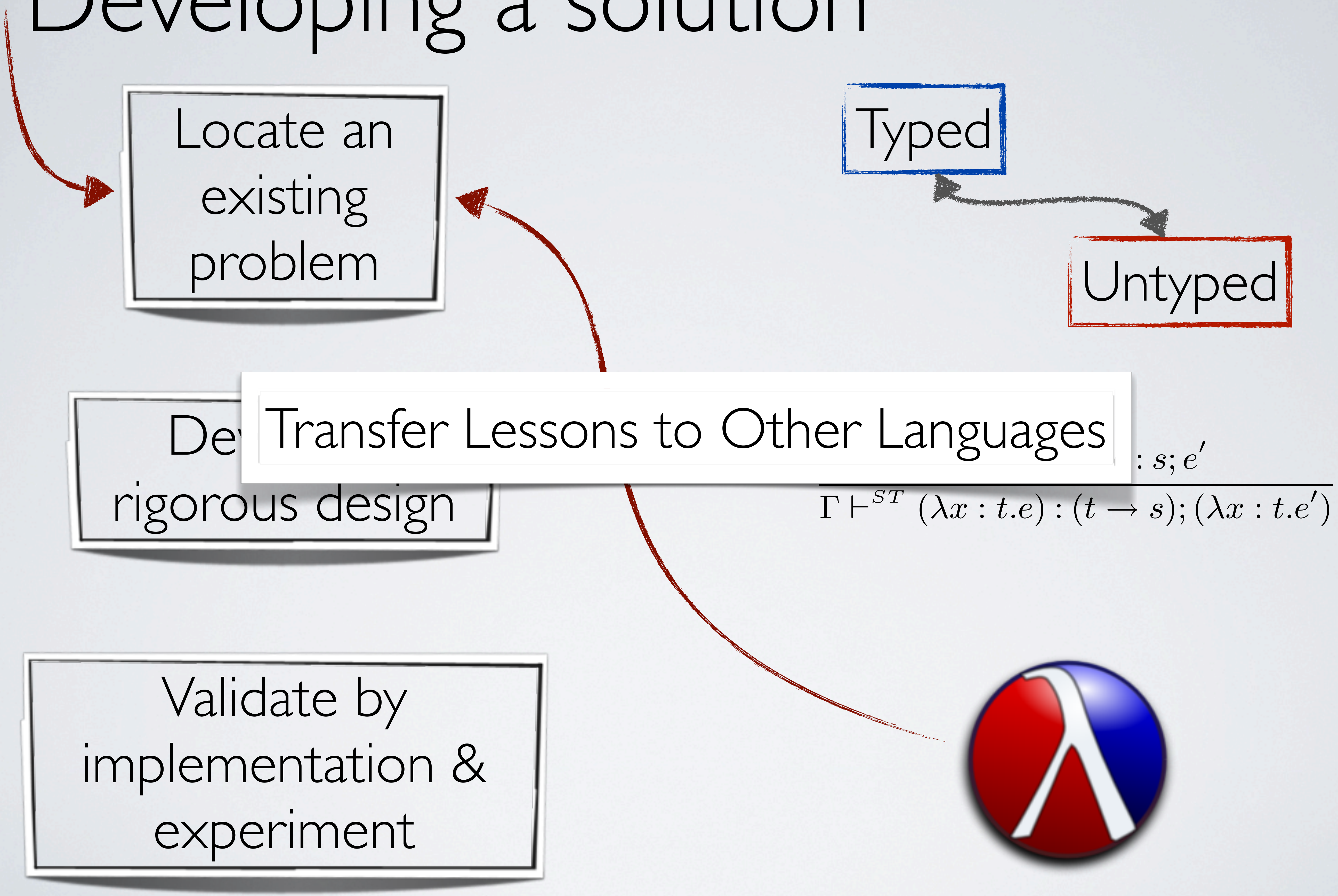
Develop a rigorous design

Validate by implementation & experiment



$$\textsc{ST-Abs}\quad \frac{\Gamma, x : t \vdash^{ST} e : s; e'}{\Gamma \vdash^{ST} (\lambda x : t.e) : (t \rightarrow s); (\lambda x : t.e')}$$

Developing a solution



Locate an existing problem

De
rigorous design

Transfer Lessons to Other Languages

Validate by implementation & experiment

Typed

Untyped

$$\frac{\quad : s; e'}{\Gamma \vdash^{ST} (\lambda x : t.e) : (t \rightarrow s); (\lambda x : t.e')}$$

The Way Forward

Bringing the solution to the broader community

Next Stop: JavaScript

Contracts

Language Infrastructure

Modules

In collaboration with ®

Next Stop: JavaScript

Language Infrastructure

Contracts

Modules

In collaboration with

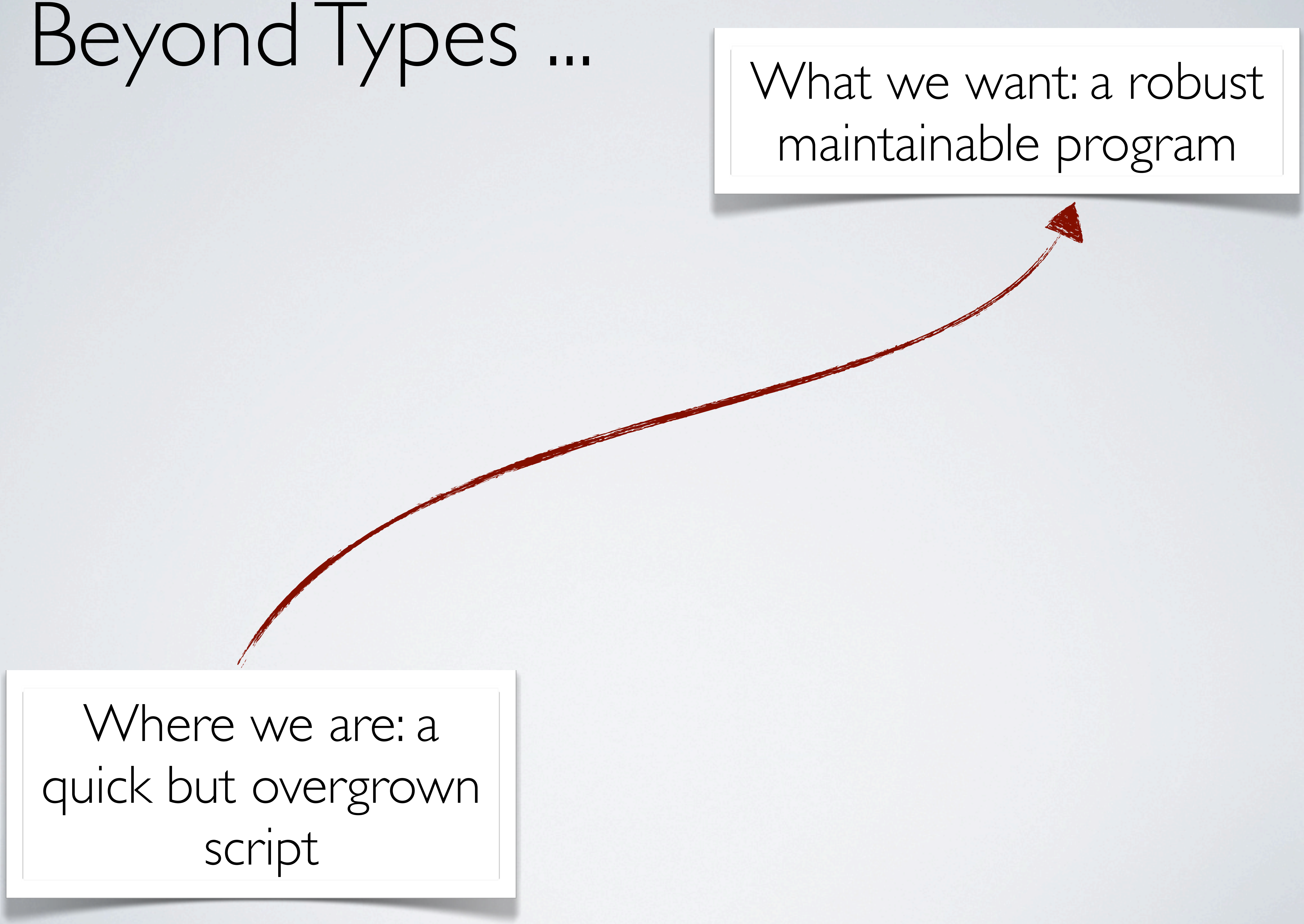
Modules on the Web

```
module $ = “http://jquery.com/jquery.js”;

$(document).ready(function() {
    alert(“hello world”);
})
```

Naming Scoping Pre-fetching, parsing, compiling

Sandboxing Cross-Origin Security

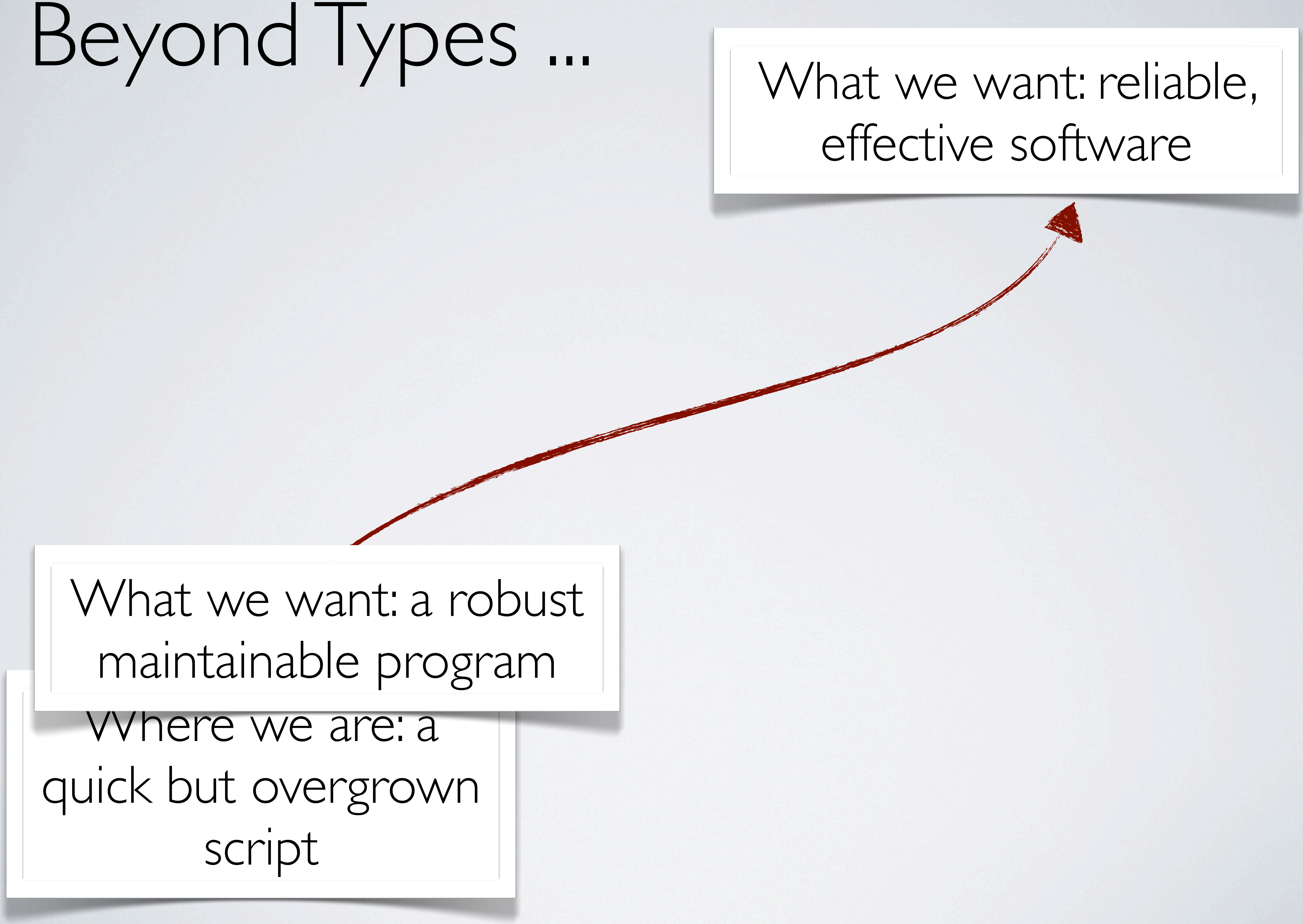
Beyond Types ...



What we want: a robust maintainable program

Where we are: a quick but overgrown script

Beyond Types ...

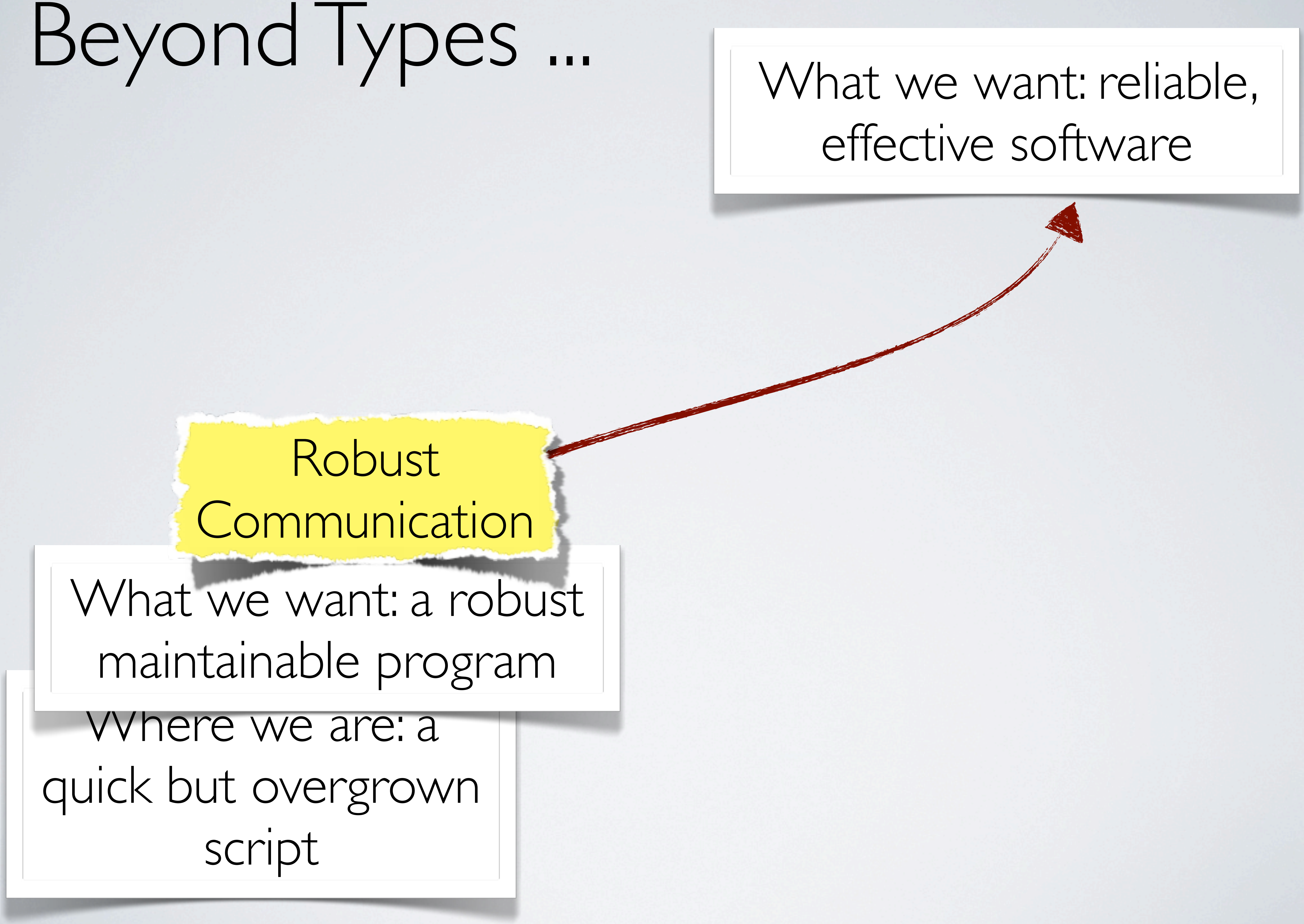


What we want: reliable, effective software

What we want: a robust maintainable program

Where we are: a quick but overgrown script

Beyond Types ...



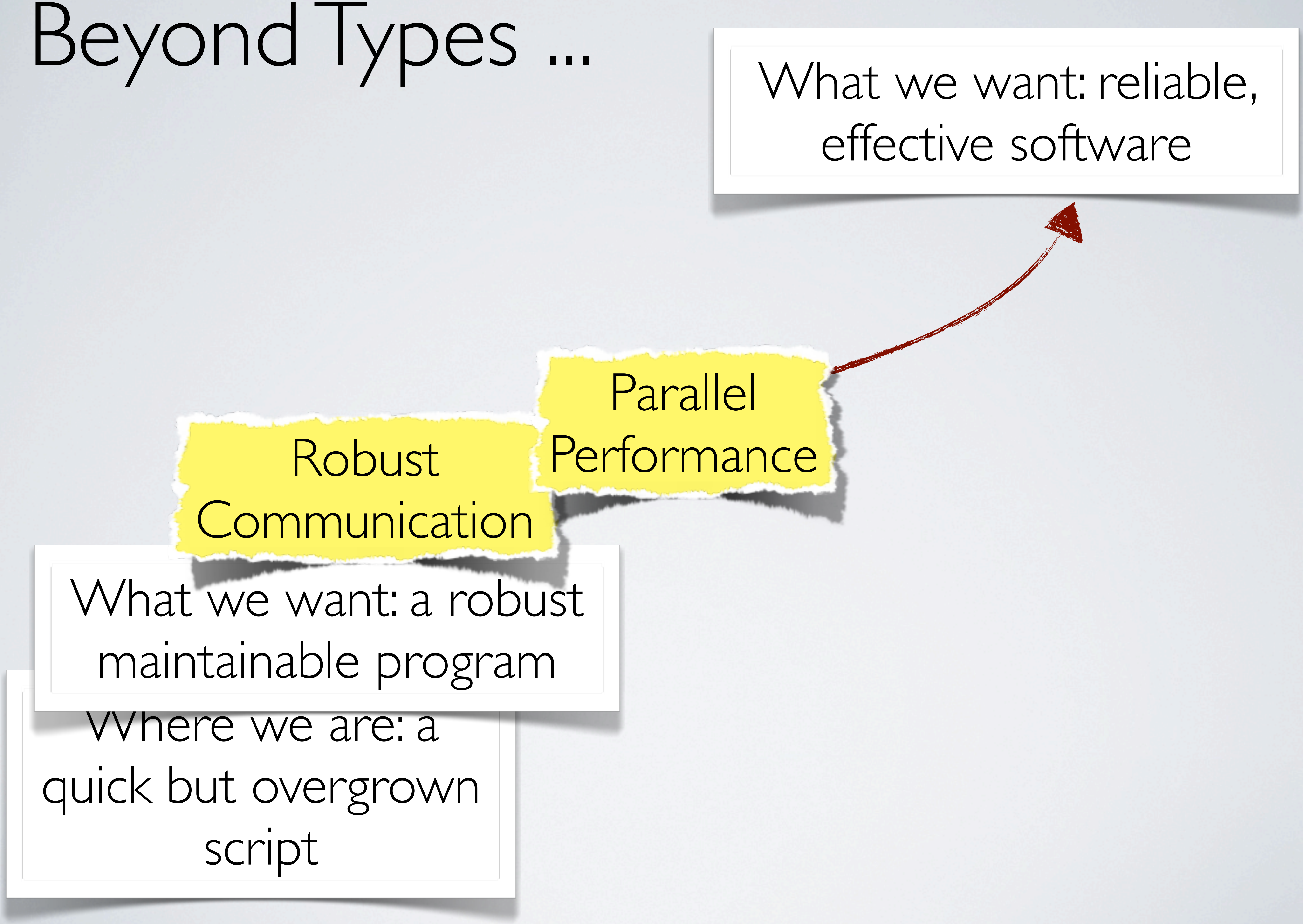
What we want: reliable, effective software

Robust Communication

What we want: a robust maintainable program

Where we are: a quick but overgrown script

Beyond Types ...



Where we are: a quick but overgrown script

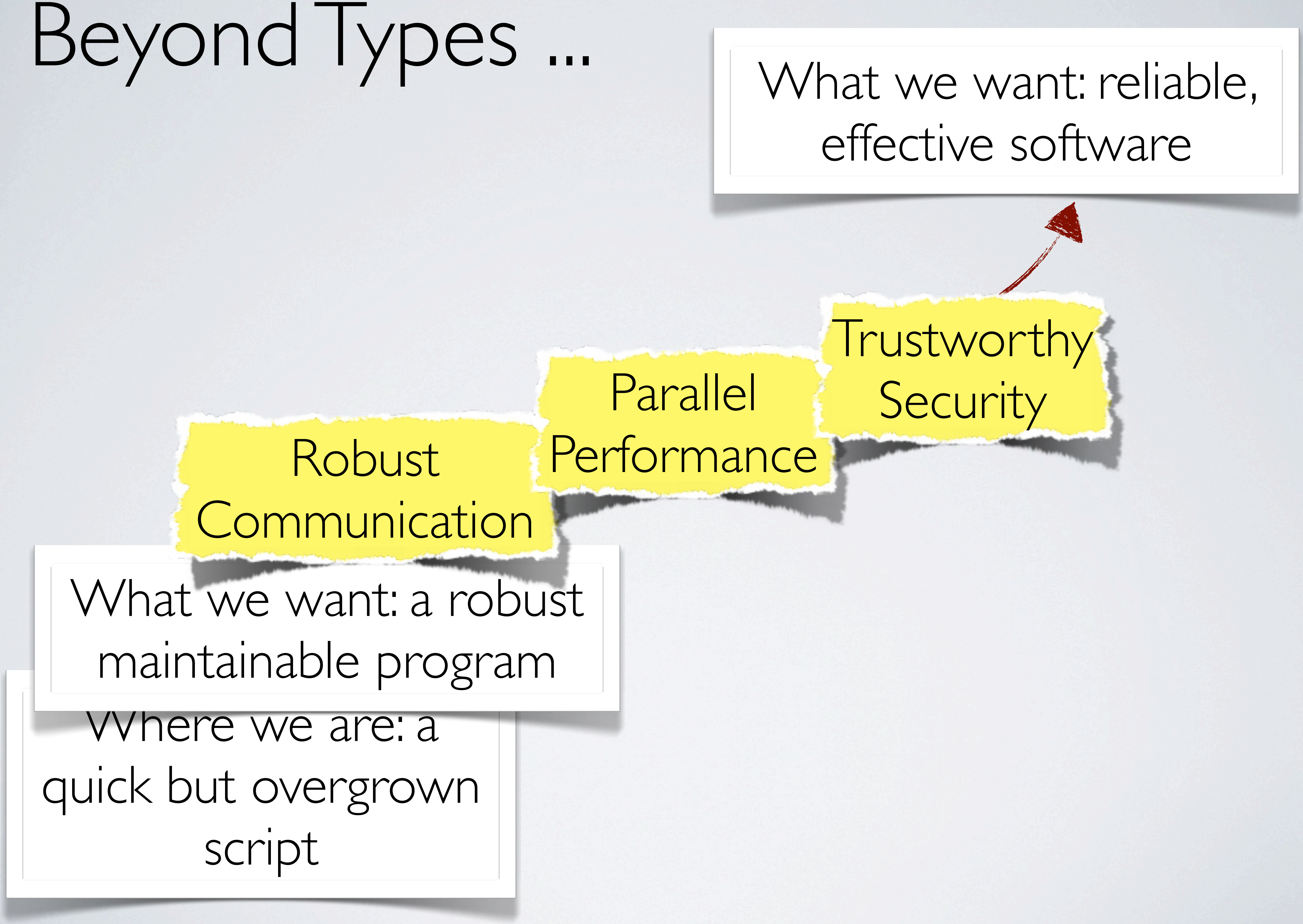
What we want: a robust maintainable program

Robust Communication

Parallel Performance

What we want: reliable, effective software

Beyond Types ...



Where we are: a quick but overgrown script

What we want: a robust maintainable program

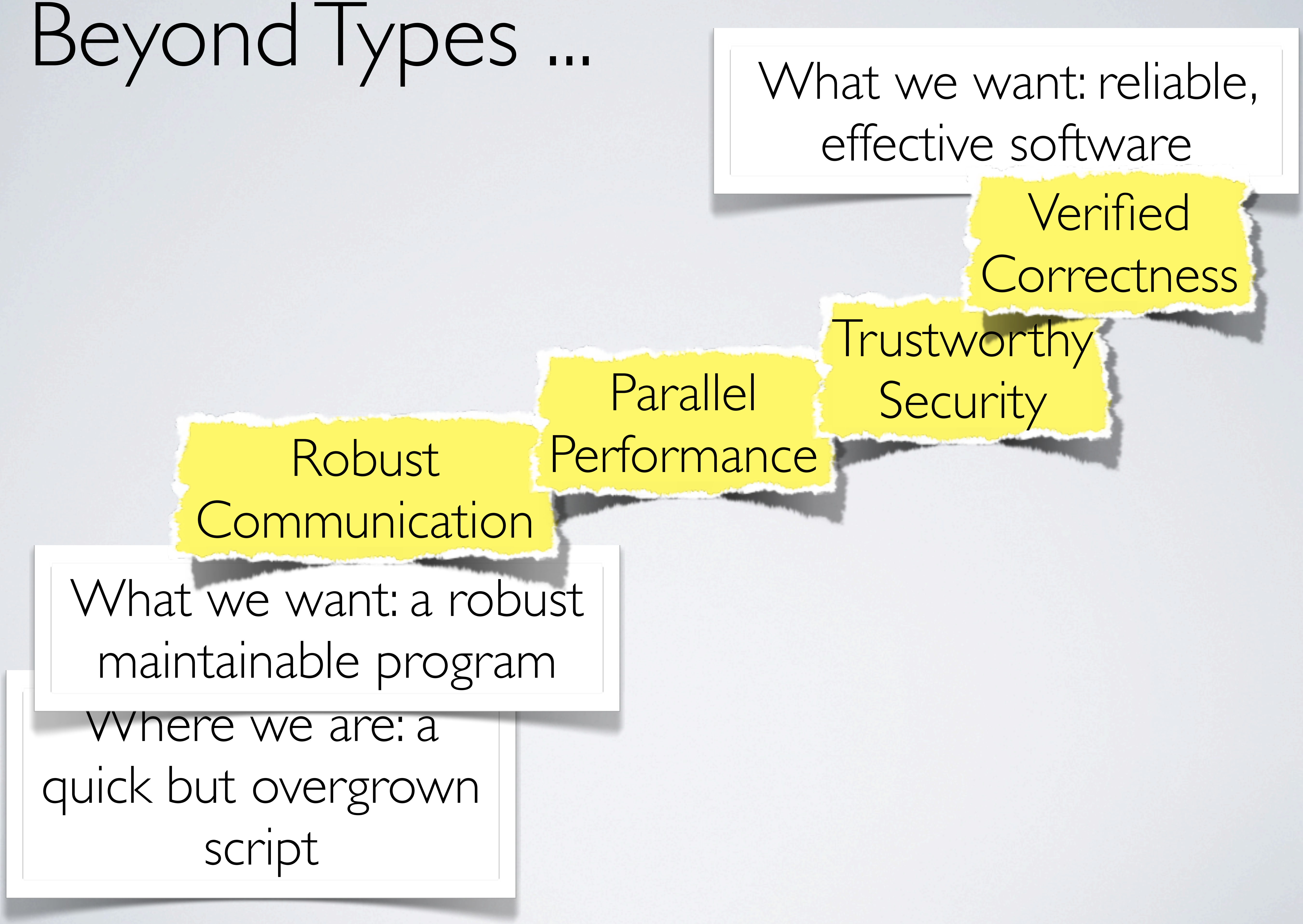
Robust Communication

Parallel Performance

Trustworthy Security

What we want: reliable, effective software

Beyond Types ...



Where we are: a quick but overgrown script

What we want: a robust maintainable program

Robust Communication

Parallel Performance

Trustworthy Security

Verified Correctness

What we want: reliable, effective software

The Big Picture

Scripts *can* become robust programs

.... modularly, soundly, and effectively

New challenges and new opportunities

The Big Picture

Scripts *can* become robust programs

.... modularly, soundly, and effectively

New challenges and new opportunities

Thank you