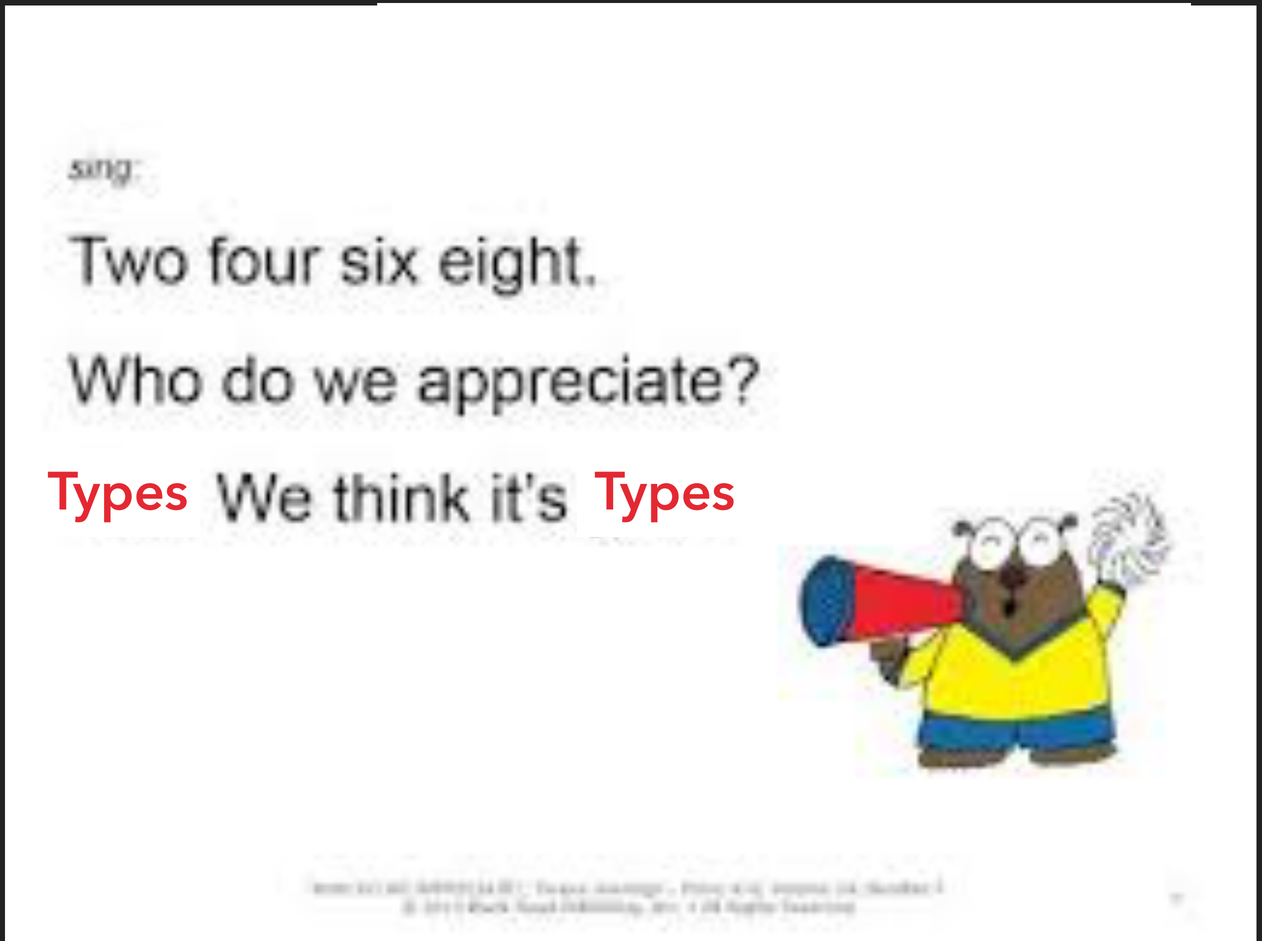
TYPES ARE LIKE THE WEATHER, TYPE SYSTEMS ARE LIKE WEATHERMEN

MATTHIAS FELLEISEN, RACKETEER



sing:

Two four six eight.

Who do we appreciate?

Types We think it's Types

[illegible]

Mr. Misunderstood

MY OWN PATH TO APPRECIATION

1978	Algol 60, Simula 67, Pascal, C	
1981	Prolog	
1984	Scheme 84	
1985	Russel	
1987	Types for Scheme	Robert “Corky” Cartwright
1991	The Meaning of Types	
1993	CMU: ML	Harper, Lee, Reynolds & Scott
1994	Soft Scheme, HM-based inference	Andrew Wright
1995	Racket, née PLT Scheme	Matthew Flatt
1997	MrSpidey, SBA-based inference	Cormac Flanagan
2005	Typed Racket, big-bang & universe	Sam Tobin-Hochstadt

MY OWN PATH TO APPRECIATION

1978	Algol 60, Simula 67, Pascal, C	
1981	Prolog	
1984	Scheme 84	
1985	Russel	
1987	Types for Scheme	
1991	The Meaning of Types	
1993	CMU: ML	
1994	Soft Scheme, HM-based inference	
1995	Racket, née PLT Scheme	Development & Maintenance
1997	MrSpidey, SBA-based inference	
2005	Typed Racket, big-bang & universe	

TWO TYPES OF TYPES

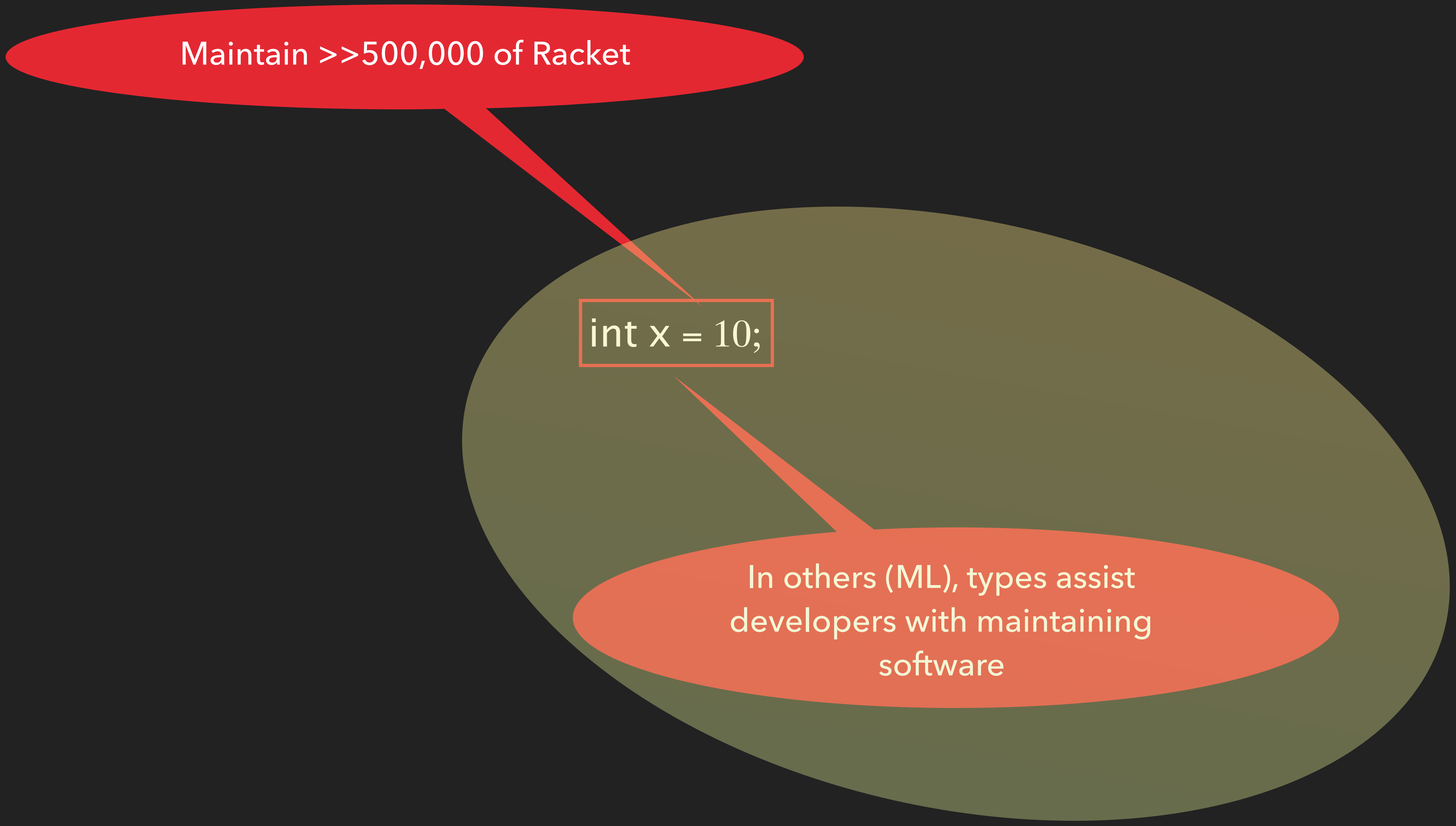
In some languages (C), types are merely instructions to the compiler.

```
int x = 10;
```

```
int x = 10;
```

In others (ML), types assist developers with maintaining software

MAINTENANCE OF LARGE CODE BASES

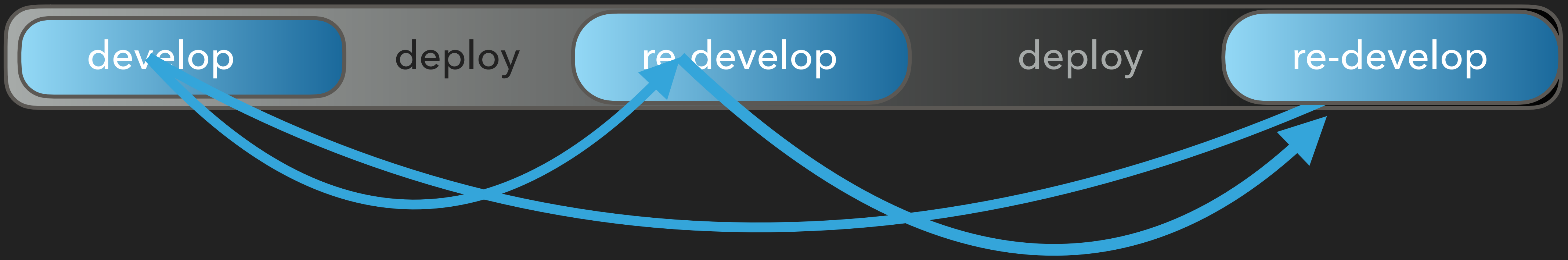


Maintain >>500,000 of Racket

int x = 10;

In others (ML), types assist developers with maintaining software

TWO MEANINGS OF "DEVELOP"



develop deploy re-develop deploy re-develop

Maintain 100Kloc – 500Kloc

```
int x = 10;
```

In others (ML), types assist developers with maintaining software

TYPES ARE LIKE THE WEATHER …

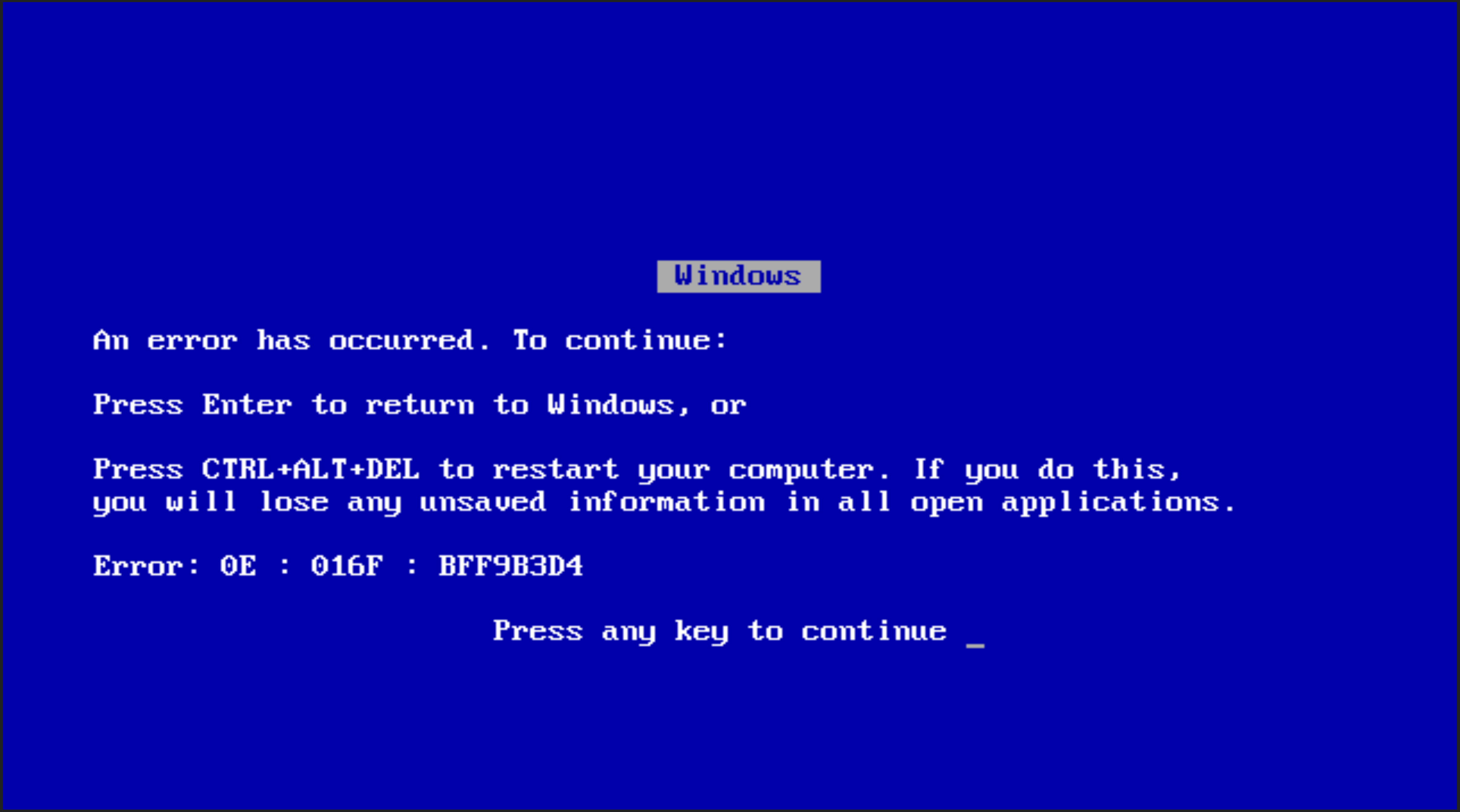
THERE IS NOTHING YOU CAN DO ABOUT IT. WEATHER HAPPENS.

THERE IS NOTHING YOU CAN DO ABOUT IT. COMPUTATION HAPPENS.

OFTEN, EVERYTHING’S JUST FINE. AND THE ANSWER IS ALWAYS

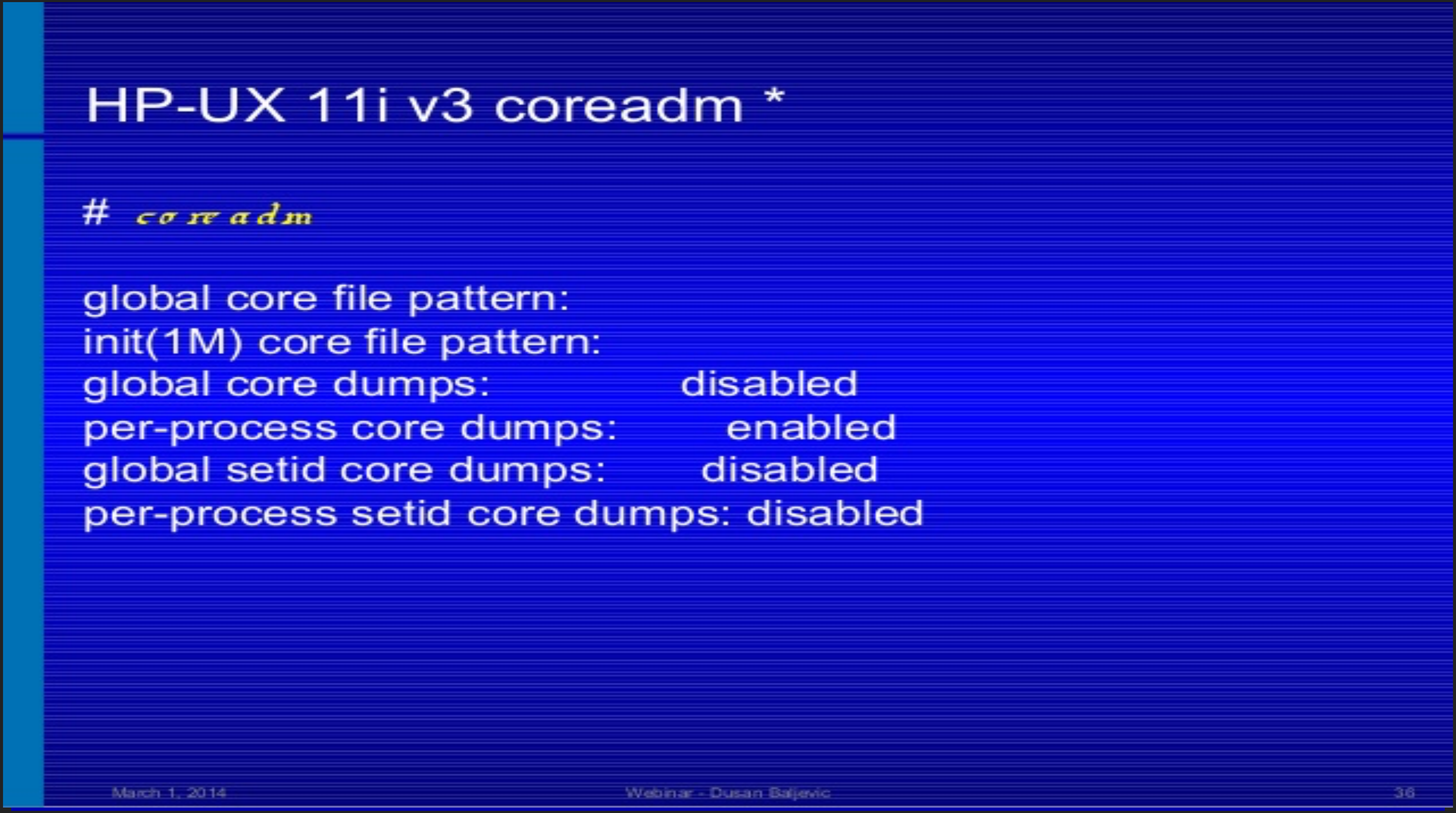
42

BUT IF YOU'RE OLD ENOUGH, YOU REMEMBER THE BLUE SCREEN OF DEATH.



```
                          Windows

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

                       Press any key to continue _
```

AND YES, YOU CAN GET THOSE ON UNIX AND LINUX SYSTEMS, TOO.

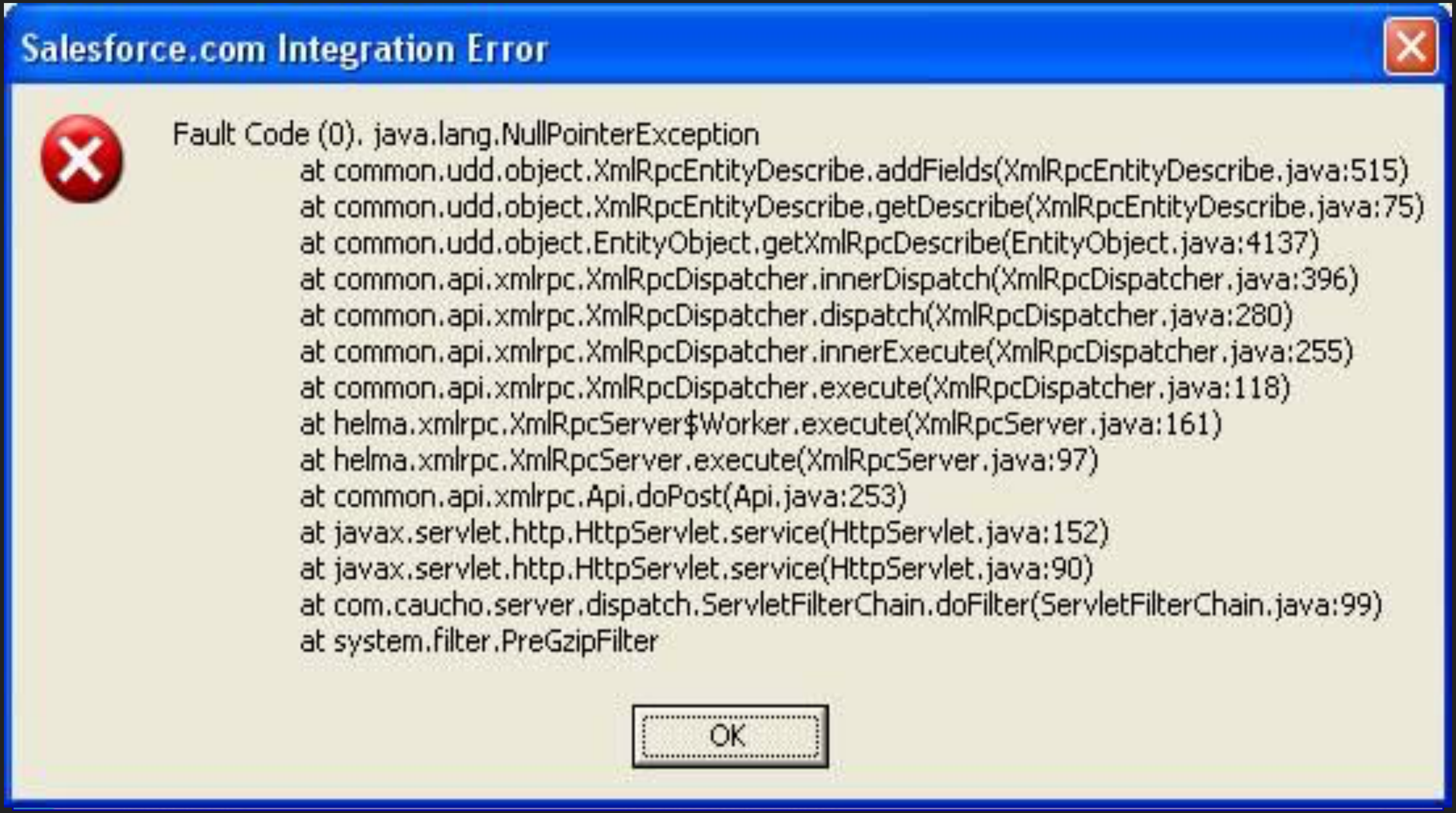


HP-UX 11i v3 coreadm *

```
# coreadm

global core file pattern:
init(1M) core file pattern:
global core dumps:                 disabled
per-process core dumps:              enabled
global setid core dumps:         disabled
per-process setid core dumps: disabled
```

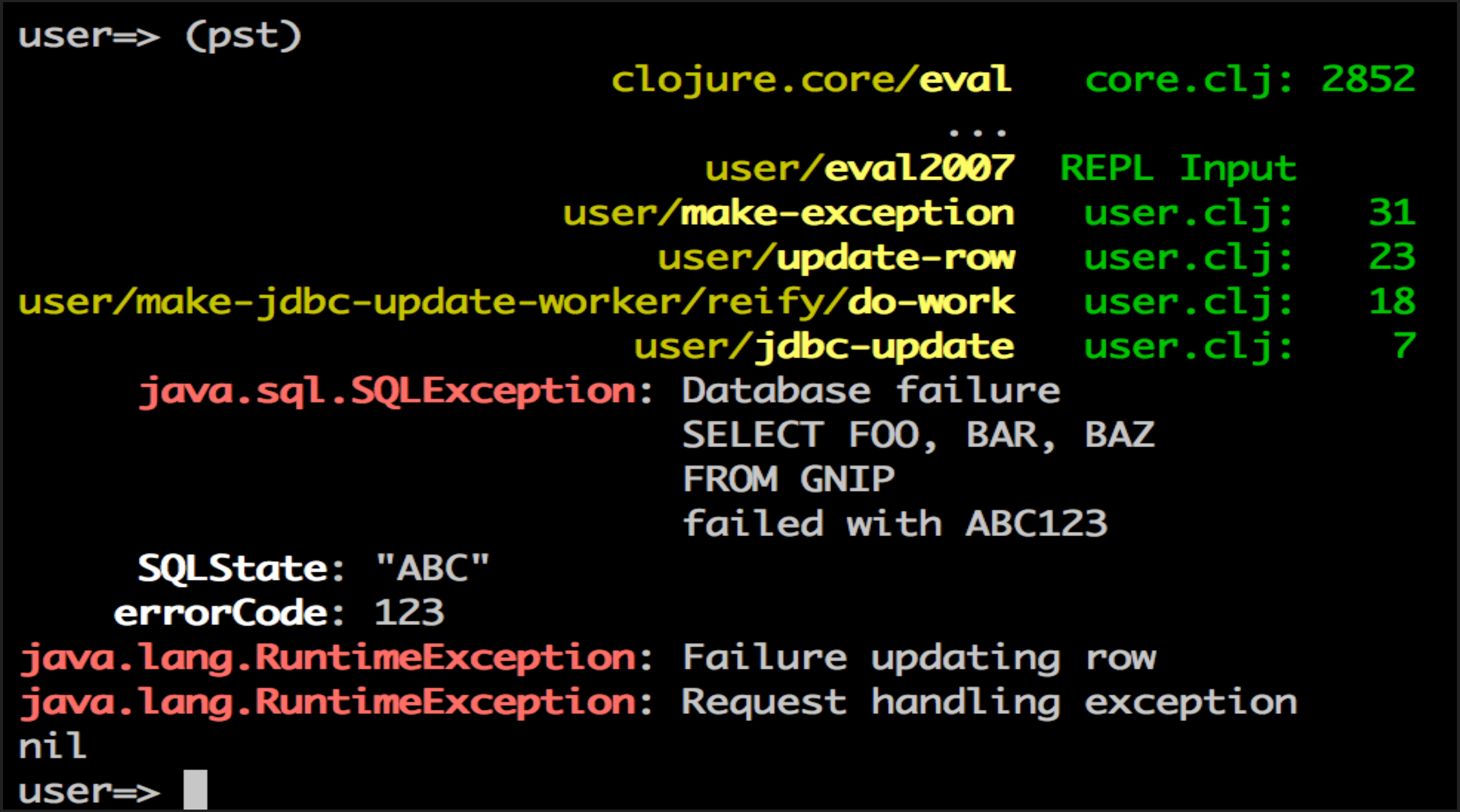
March 1, 2014 Webinar - Dusan Baljevic 36

PROGRESS! ALL YOU GOT WAS A NULL POINTER EXCEPTION.



```
Fault Code (0). java.lang.NullPointerException
        at common.udd.object.XmlRpcEntityDescribe.addFields(XmlRpcEntityDescribe.java:515)
        at common.udd.object.XmlRpcEntityDescribe.getDescribe(XmlRpcEntityDescribe.java:75)
        at common.udd.object.EntityObject.getXmlRpcDescribe(EntityObject.java:4137)
        at common.api.xmlrpc.XmlRpcDispatcher.innerDispatch(XmlRpcDispatcher.java:396)
        at common.api.xmlrpc.XmlRpcDispatcher.dispatch(XmlRpcDispatcher.java:280)
        at common.api.xmlrpc.XmlRpcDispatcher.innerExecute(XmlRpcDispatcher.java:255)
        at common.api.xmlrpc.XmlRpcDispatcher.execute(XmlRpcDispatcher.java:118)
        at helma.xmlrpc.XmlRpcServer$Worker.execute(XmlRpcServer.java:161)
        at helma.xmlrpc.XmlRpcServer.execute(XmlRpcServer.java:97)
        at common.api.xmlrpc.Api.doPost(Api.java:253)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:152)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:90)
        at com.caucho.server.dispatch.ServletFilterChain.doFilter(ServletFilterChain.java:99)
        at system.filter.PreGzipFilter
```

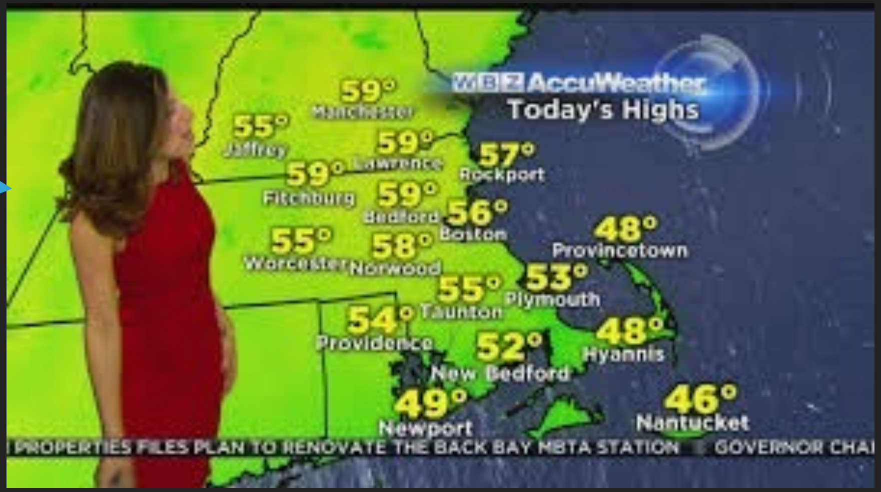
WELL, YES. EXCEPTIONS EXIST IN YOUR FAVORITE LANGUAGE, TOO.



```
user=> (pst)
                         clojure.core/eval   core.clj: 2852
                                       ...
                             user/eval2007  REPL Input
                       user/make-exception   user.clj:   31
                           user/update-row   user.clj:   23
user/make-jdbc-update-worker/reify/do-work   user.clj:   18
                          user/jdbc-update   user.clj:    7
     java.sql.SQLException: Database failure
                            SELECT FOO, BAR, BAZ
                            FROM GNIP
                            failed with ABC123
     SQLState: "ABC"
    errorCode: 123
java.lang.RuntimeException: Failure updating row
java.lang.RuntimeException: Request handling exception
nil
user=> 
```

TYPE SYSTEMS ARE LIKE THE WEATHERMEN

WEATHERMEN USE MATHEMATICAL MODELS TO PREDICT THE WEATHER



$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + \frac{Q(x,t)}{c\rho}$$

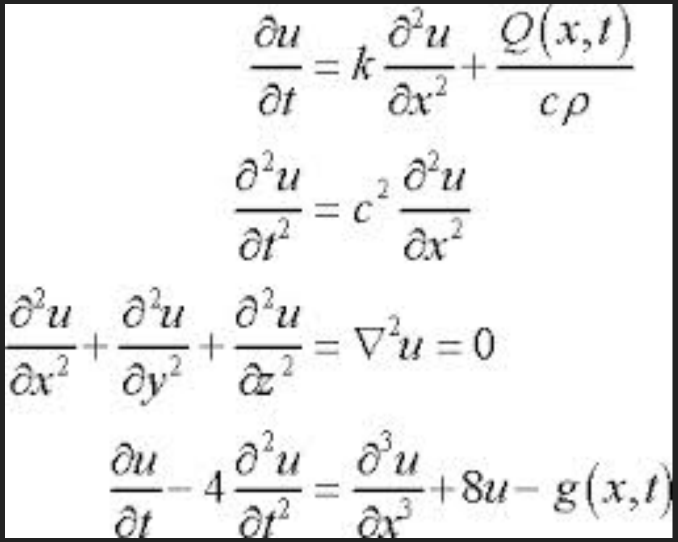
$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \nabla^2 u = 0$$

$$\frac{\partial u}{\partial t} - 4\frac{\partial^2 u}{\partial t^2} = \frac{\partial^3 u}{\partial x^3} + 8u - g(x,t)$$

- This prediction is *partial* but *useful*.
- It is mostly accurate.

WEATHERMEN USE MATHEMATICAL MODELS TO PREDICT THE WEATHER

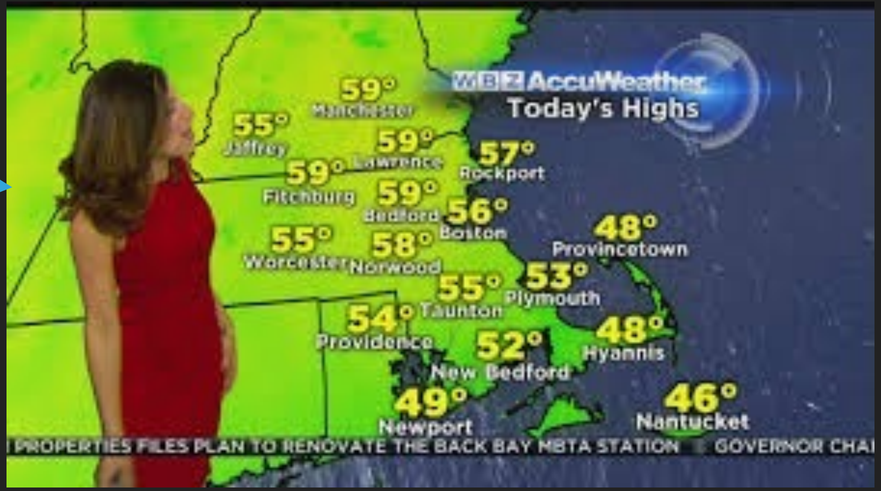


$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + \frac{Q(x,t)}{c\rho}$$

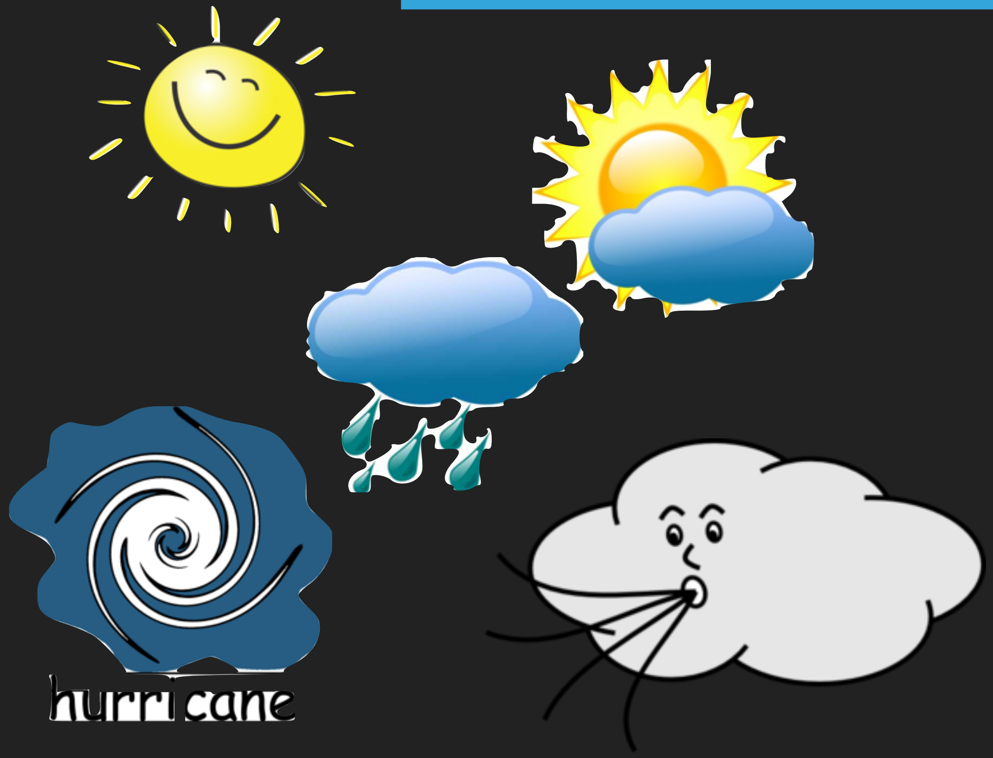
$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \nabla^2 u = 0$$

$$\frac{\partial u}{\partial t} - 4\frac{\partial^2 u}{\partial t^2} = \frac{\partial^3 u}{\partial x^3} + 8u - g(x,t)$$

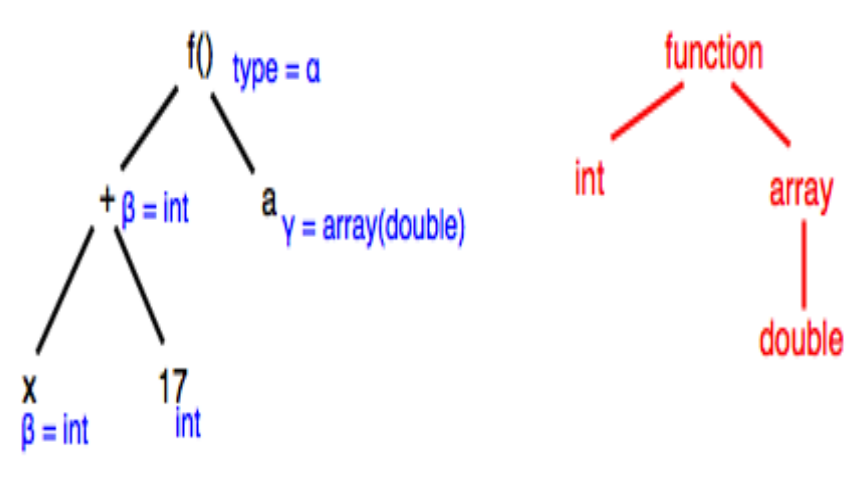


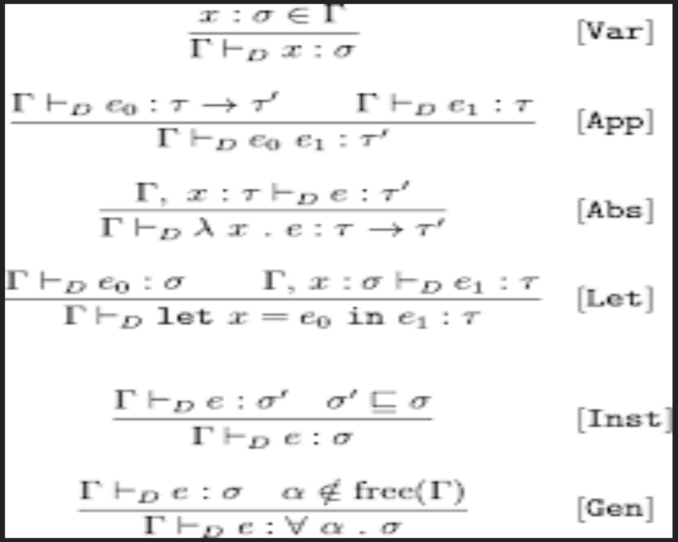
some partial predictions



and the emphasis is on *mostly* in accurate

LANGUAGES USE MATHEMATICAL MODELS TO PREDICT COMPUTATIONS





$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash_D x : \sigma} \quad \texttt{[Var]}$$

$$\frac{\Gamma \vdash_D e_0 : \tau \rightarrow \tau' \qquad \Gamma \vdash_D e_1 : \tau}{\Gamma \vdash_D e_0\ e_1 : \tau'} \quad \texttt{[App]}$$

$$\frac{\Gamma,\ x : \tau \vdash_D e : \tau'}{\Gamma \vdash_D \lambda\ x\ .\ e : \tau \rightarrow \tau'} \quad \texttt{[Abs]}$$

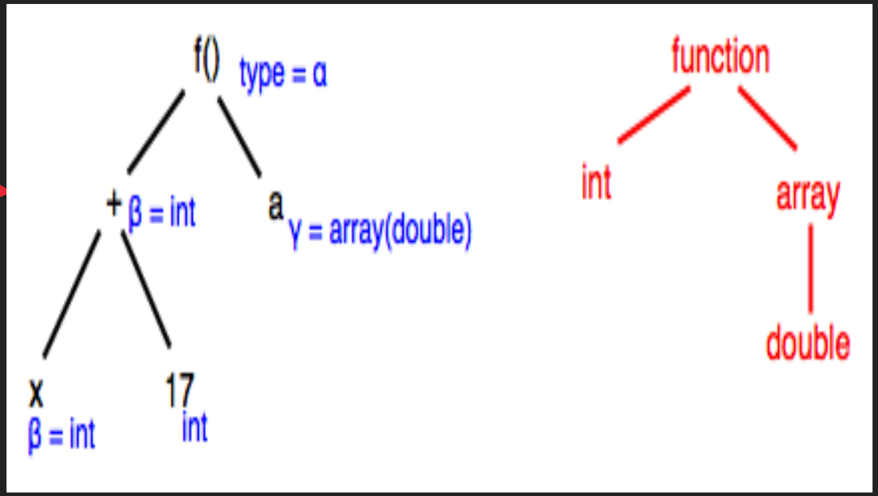
$$\frac{\Gamma \vdash_D e_0 : \sigma \qquad \Gamma, x : \sigma \vdash_D e_1 : \tau}{\Gamma \vdash_D \texttt{let}\ x = e_0\ \texttt{in}\ e_1 : \tau} \quad \texttt{[Let]}$$

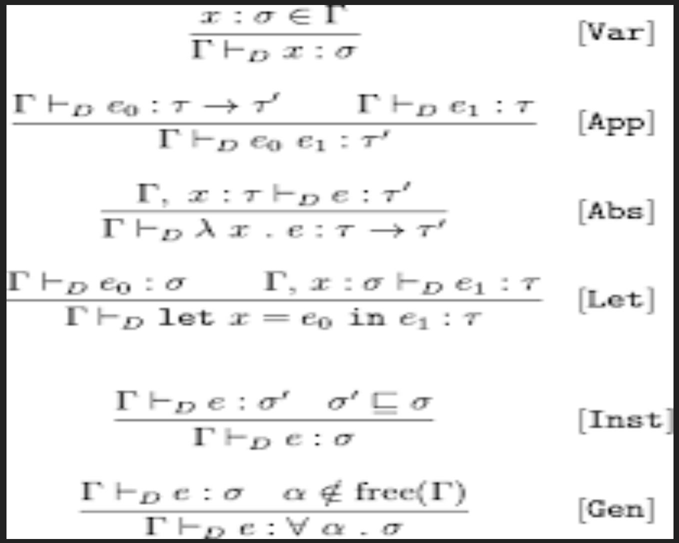
$$\frac{\Gamma \vdash_D e : \sigma' \quad \sigma' \sqsubseteq \sigma}{\Gamma \vdash_D e : \sigma} \quad \texttt{[Inst]}$$

$$\frac{\Gamma \vdash_D e : \sigma \quad \alpha \notin \mathrm{free}(\Gamma)}{\Gamma \vdash_D e : \forall\ \alpha\ .\ \sigma} \quad \texttt{[Gen]}$$

- This prediction is *partial* but *useful*.
- It is *mostly accurate.*

LANGUAGES USE MATHEMATICAL MODELS TO PREDICT COMPUTATIONS





$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash_D x : \sigma} \quad [\mathrm{Var}]$$

$$\frac{\Gamma \vdash_D e_0 : \tau \rightarrow \tau' \qquad \Gamma \vdash_D e_1 : \tau}{\Gamma \vdash_D e_0\ e_1 : \tau'} \quad [\mathrm{App}]$$

$$\frac{\Gamma,\ x : \tau \vdash_D e : \tau'}{\Gamma \vdash_D \lambda\ x\ .\ e : \tau \rightarrow \tau'} \quad [\mathrm{Abs}]$$

$$\frac{\Gamma \vdash_D e_0 : \sigma \qquad \Gamma,\ x : \sigma \vdash_D e_1 : \tau}{\Gamma \vdash_D \mathbf{let}\ x = e_0\ \mathbf{in}\ e_1 : \tau} \quad [\mathrm{Let}]$$

$$\frac{\Gamma \vdash_D e : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma \vdash_D e : \sigma} \quad [\mathrm{Inst}]$$

$$\frac{\Gamma \vdash_D e : \sigma \qquad \alpha \notin \mathrm{free}(\Gamma)}{\Gamma \vdash_D e : \forall\ \alpha\ .\ \sigma} \quad [\mathrm{Gen}]$$

```
(f
r
```

And what about *accuracy*?

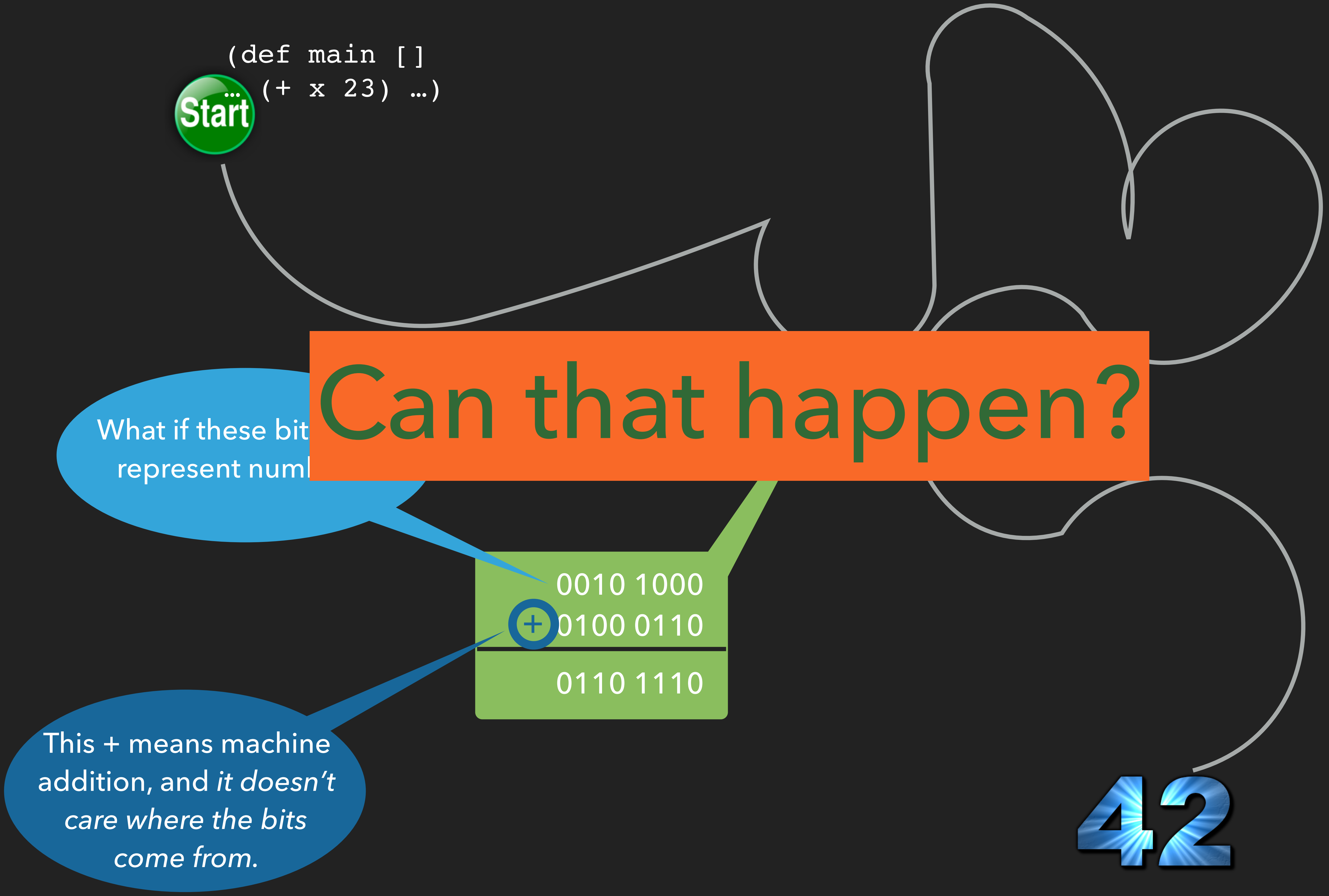
```
ever
give back”
```

TAKE AWAY 1

- Types are the language of prediction.
- Type systems use them to make more predictions.
- The questions are:
  - *Is that useful?*
  - *Is it meaningful?*

THE MEANING OF TYPES ~ SOUNDNESS

A COMPUTATION IS A RANDOM WALK IN THE UNIVERSE OF BITS.



```
(def main []
  … (+ x 23) …)
```

Can that happen?

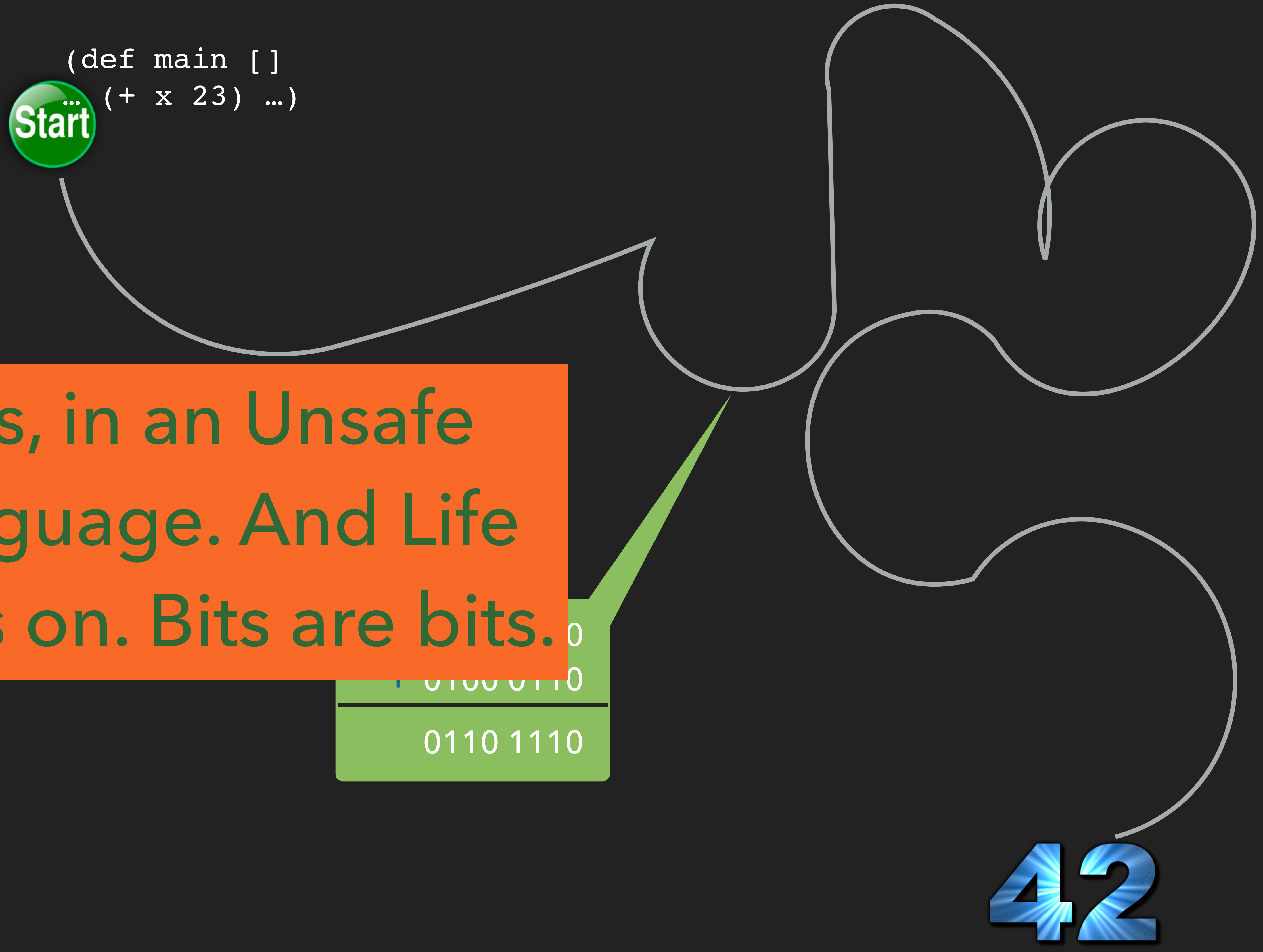
What if these bit
represent num

```
 0010 1000
+0100 0110
 0110 1110
```

This + means machine addition, and *it doesn't care where the bits come from.*

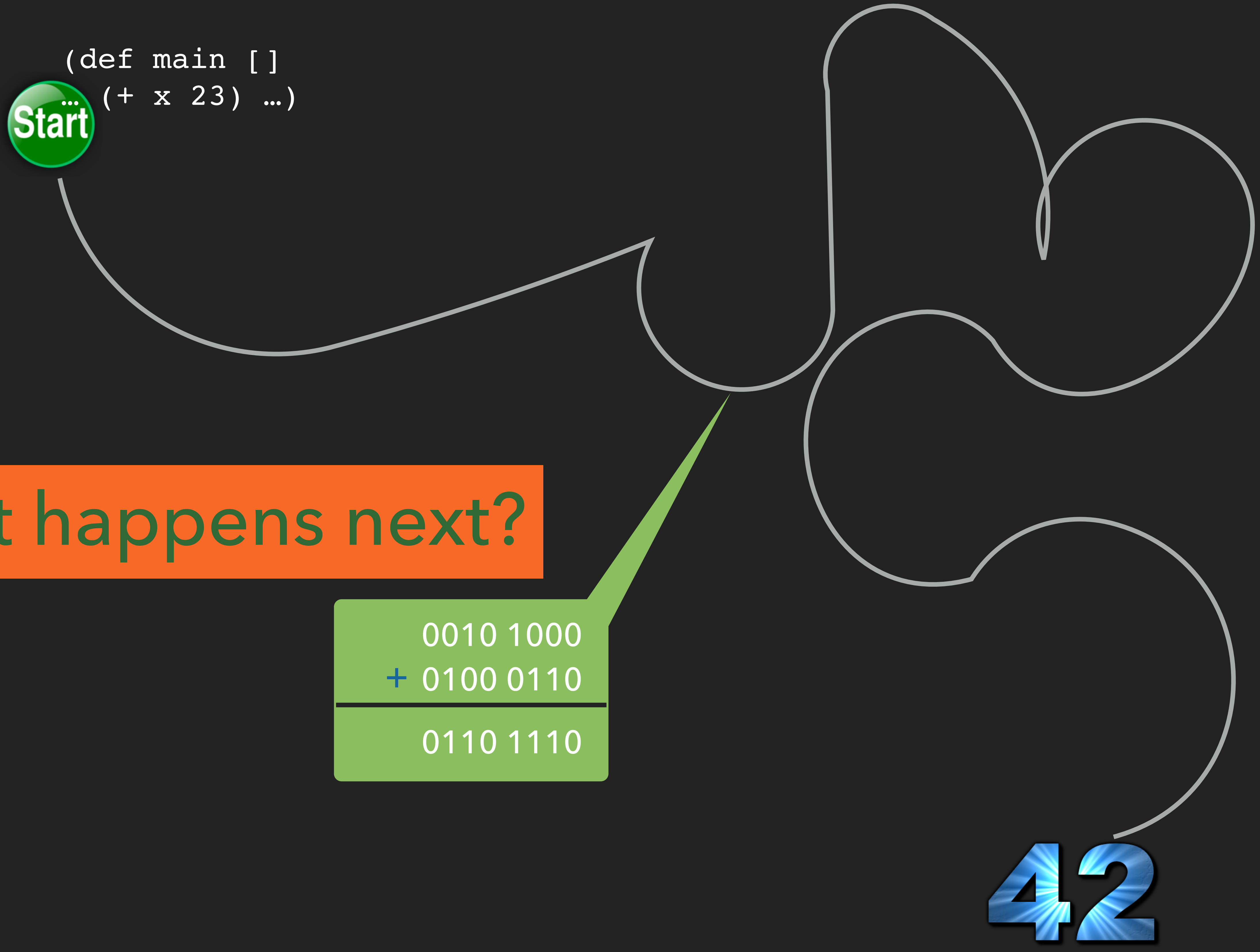
CAN IT GO WRONG? CAN WE FALL OFF THE CLIFF?

```
(def main []
 … (+ x 23) …)
```



Yes, in an Unsafe Language. And Life Goes on. Bits are bits.

IT DEPENDS.



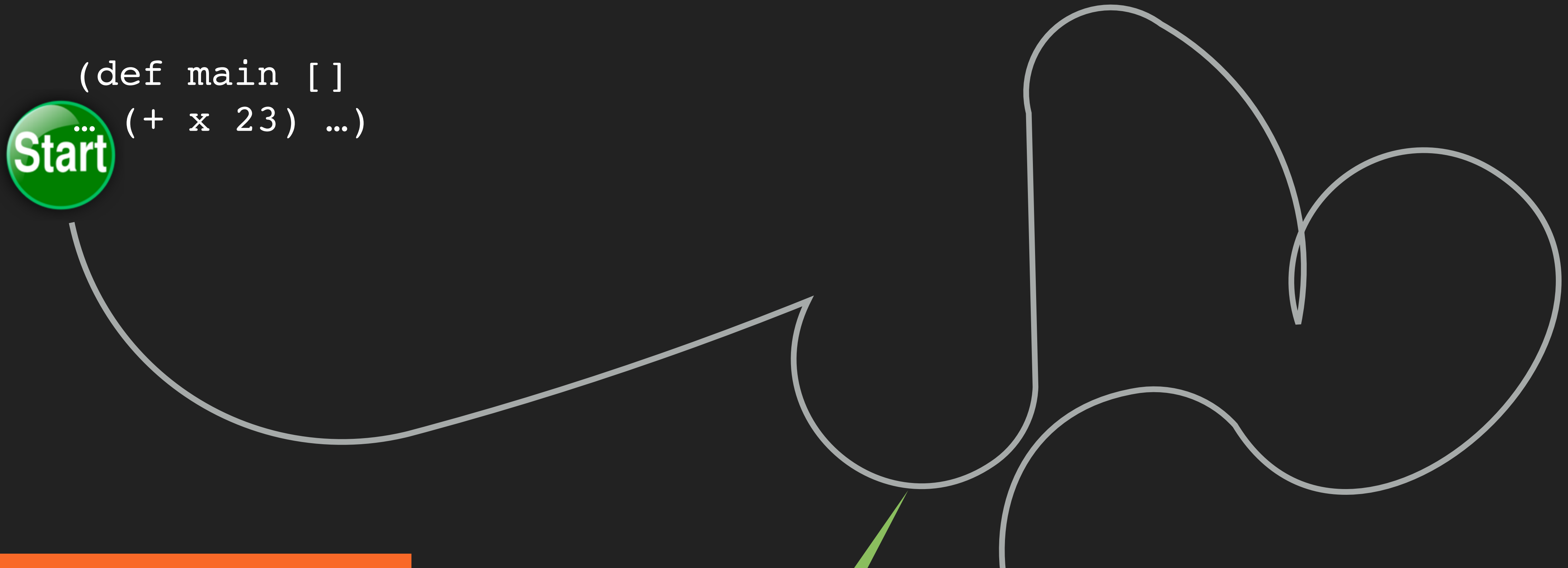
```
(def main []
  … (+ x 23) …)
```

What happens next?

```
  0010 1000
+ 0100 0110
-----------
  0110 1110
```

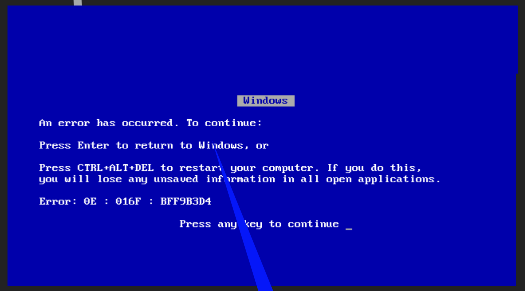
IN AN UNSOUND LANGUAGE SUCH AS C++:

```
(def main []
  … (+ x 23) …)
```



If you’re lucky:

```
  0010 1000
+ 0100 0110
-----------
  0110 1110
```

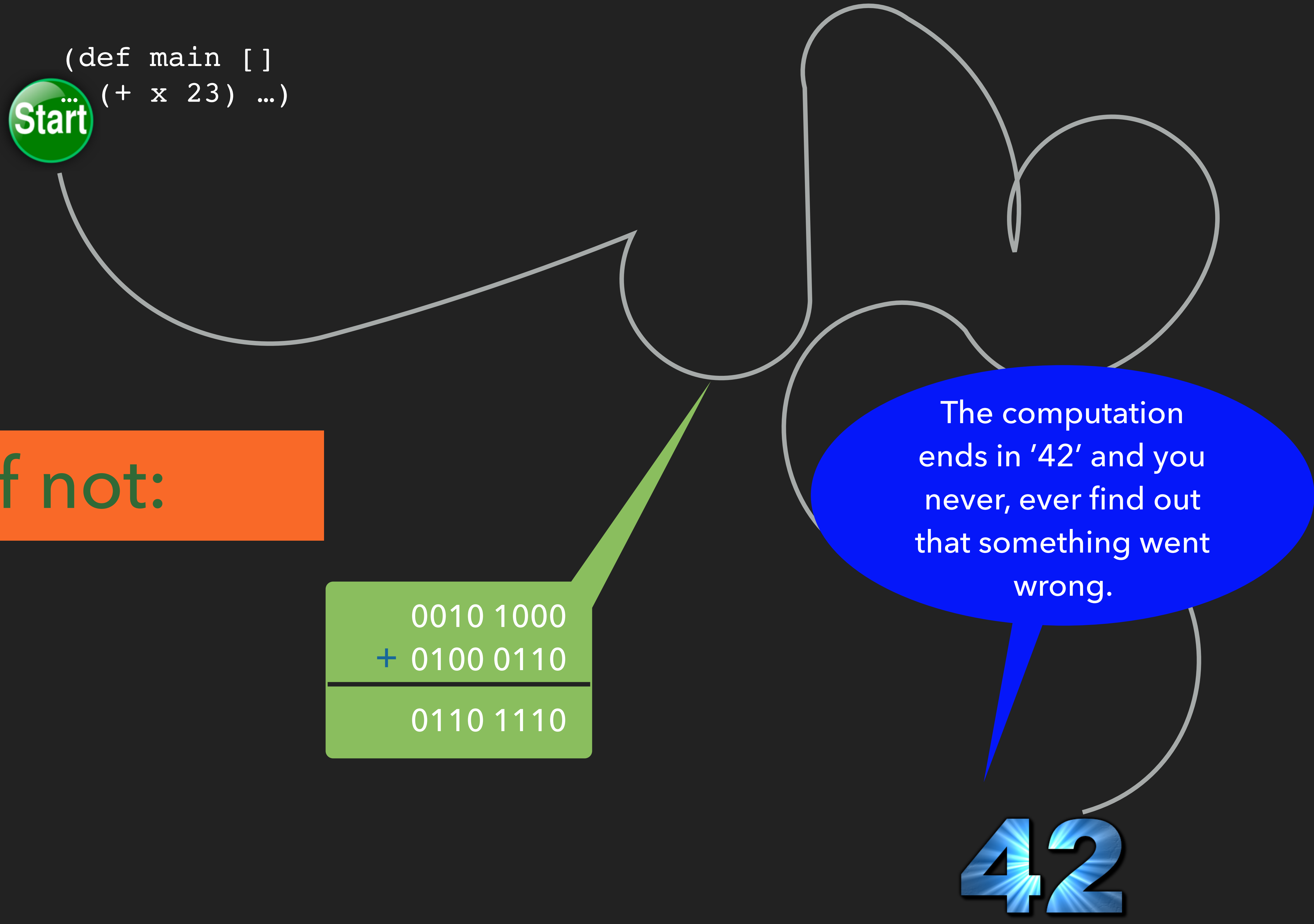


```
                          Windows

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

              Press any key to continue _
```

The computation ends in a segfault.

IN AN UNSOUND LANGUAGE SUCH AS C++:

And if not:



```
(def main []
  … (+ x 23) …)
```

```
  0010 1000
+ 0100 0110
-----------
  0110 1110
```

The computation ends in ’42’ and you never, ever find out that something went wrong.

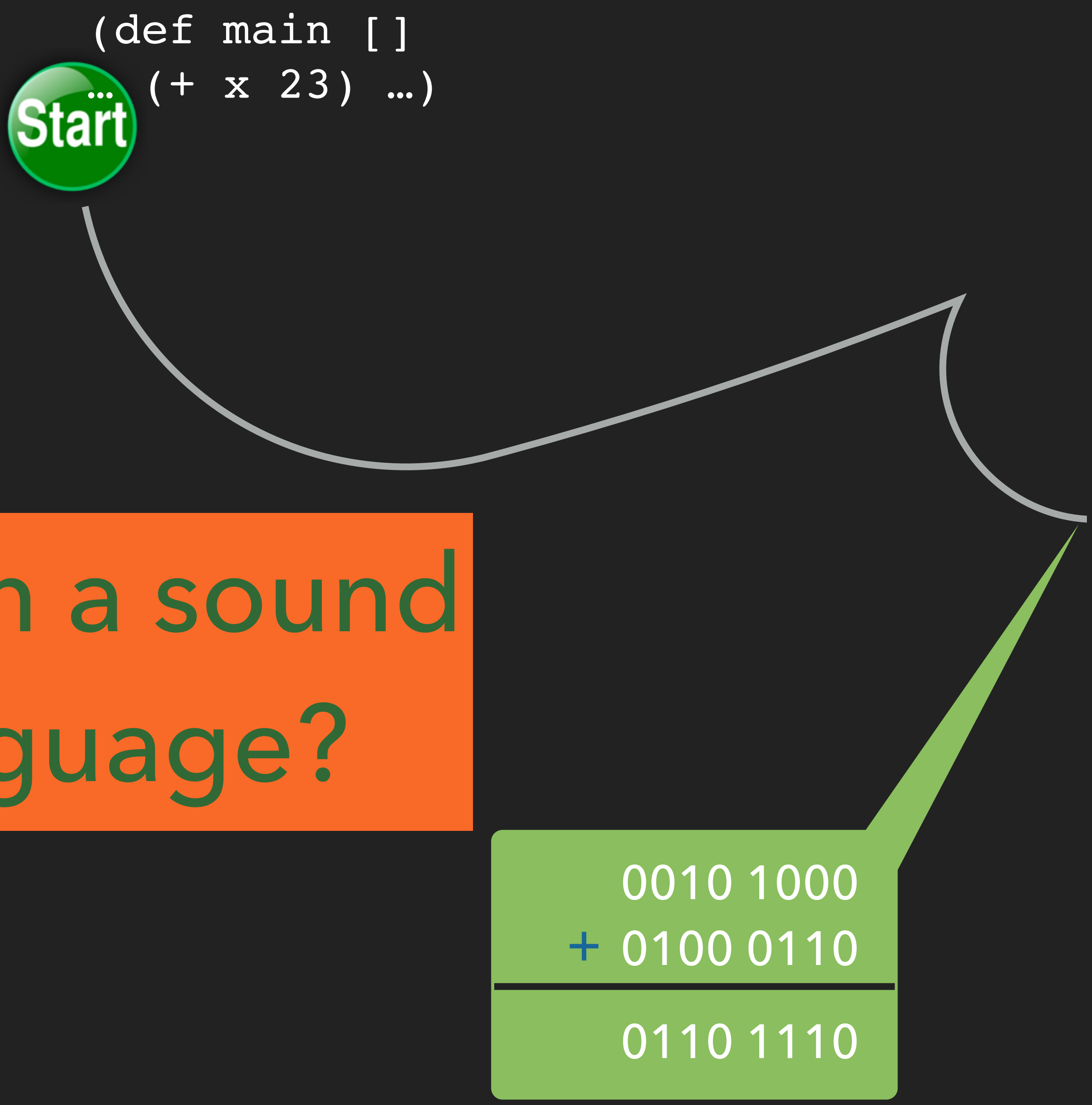
IN AN UNSOUND LANGUAGE SUCH AS C++:

Problematic bit manipulations may escape discovery during testing, even if your testing covers the particular path on which things go wrong.

CATASTROPHE!

Now imagine a program that controls your grandmother's heart pacemaker.

IN AN SOUND LANGUAGE SUCH AS ML:



```
(def main []
 ... (+ x 23) …)
```

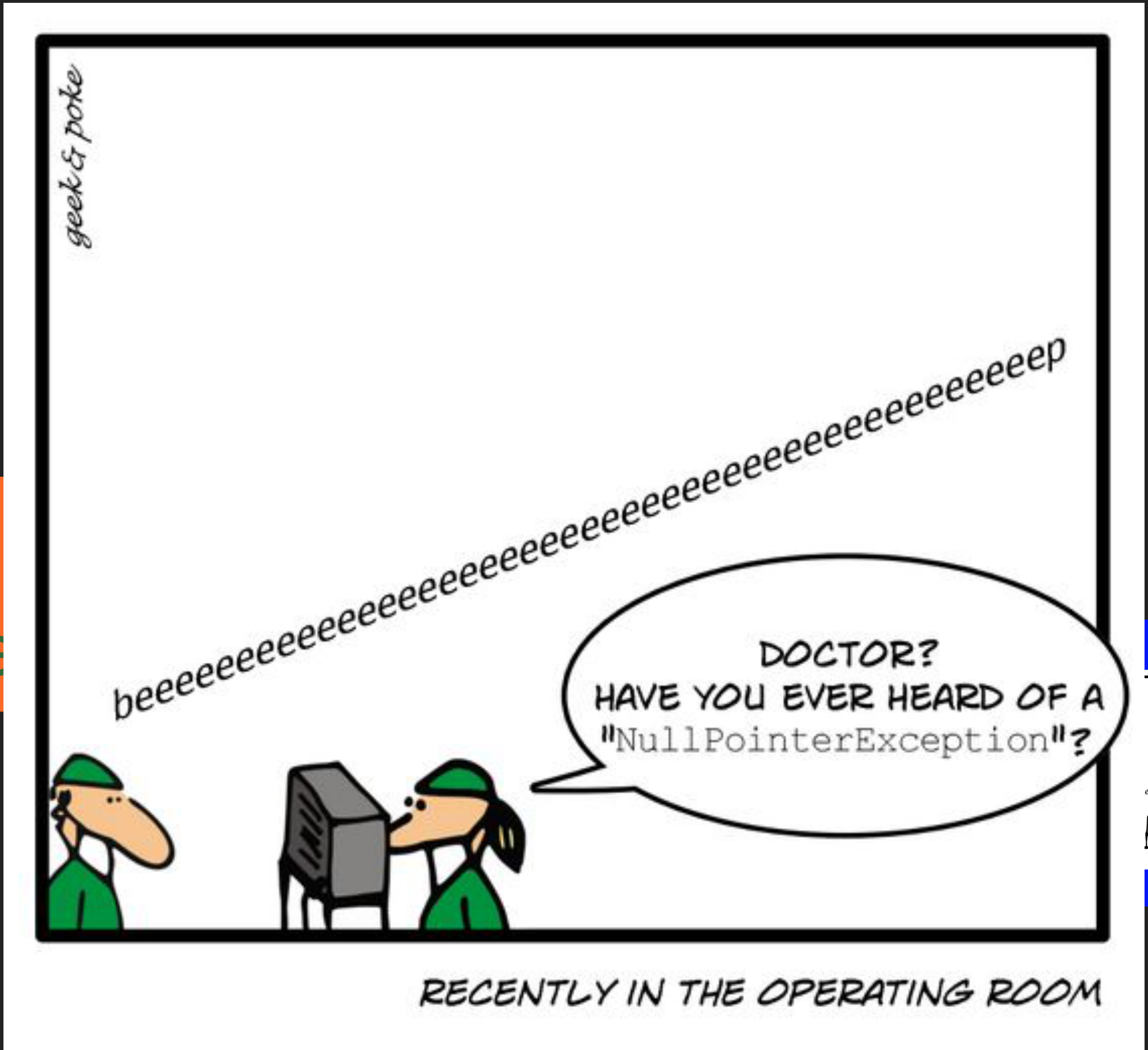
```
  0010 1000
+ 0100 0110
-----------
  0110 1110
```

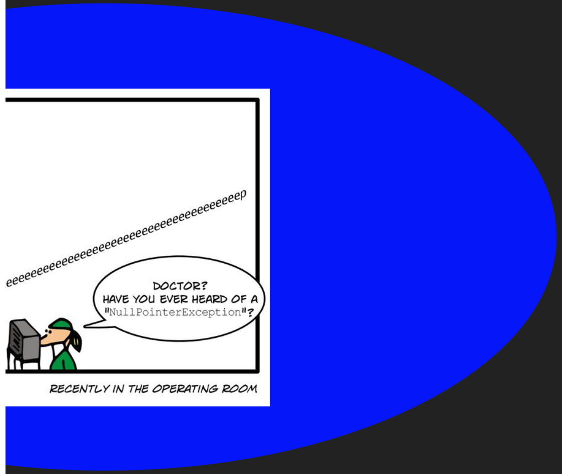
And in a sound language?

IN AN SOUND LANGUAGE SUCH AS ML:

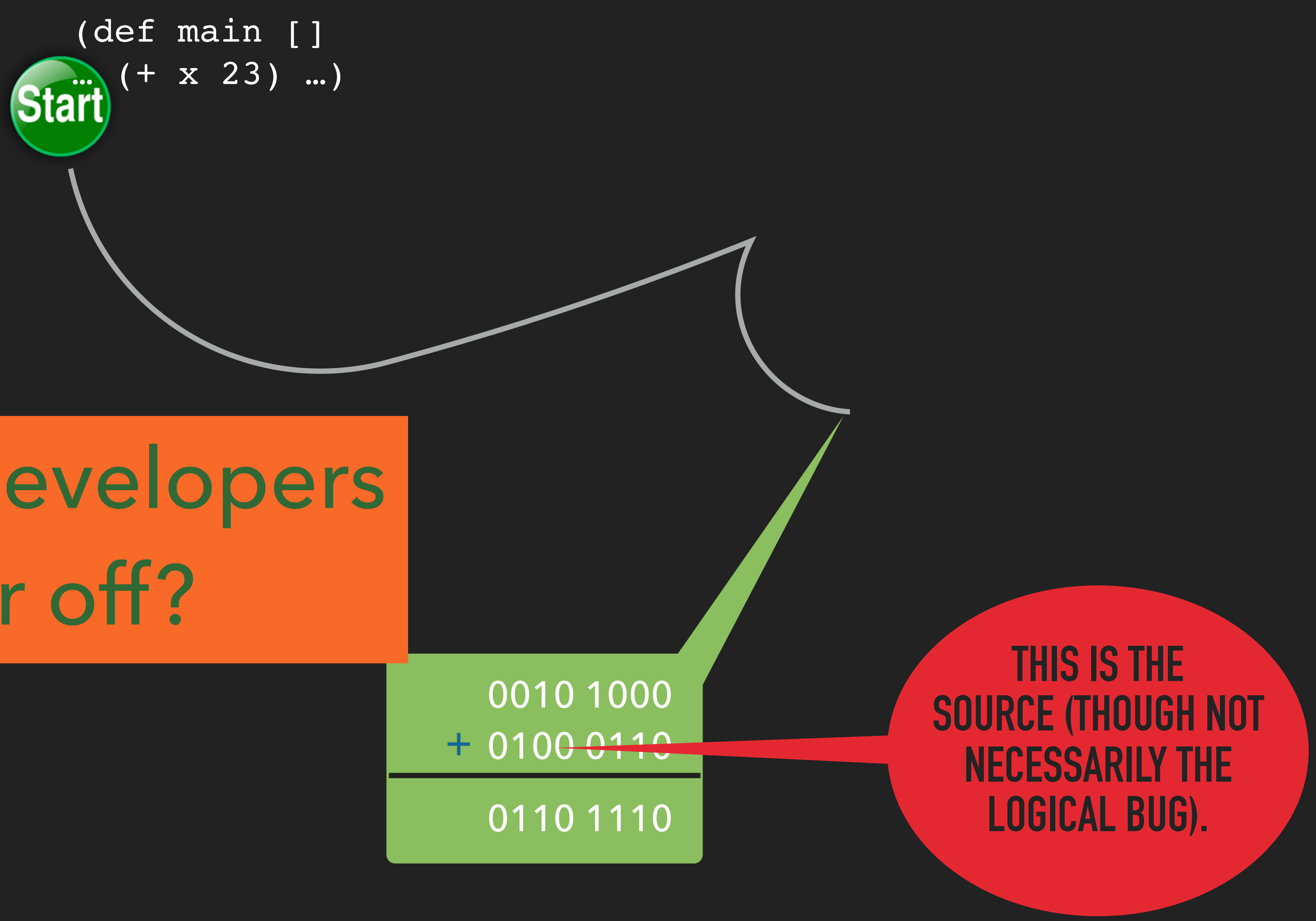
It *im*

raise


geek & poke
beeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeep
DOCTOR? HAVE YOU EVER HEARD OF A "NullPointerException"?
RECENTLY IN THE OPERATING ROOM


DOCTOR? HAVE YOU EVER HEARD OF A "NullPointerException"?
RECENTLY IN THE OPERATING ROOM

IN AN SOUND LANGUAGE SUCH AS ML:



```
(def main []
  … (+ x 23) …)
```

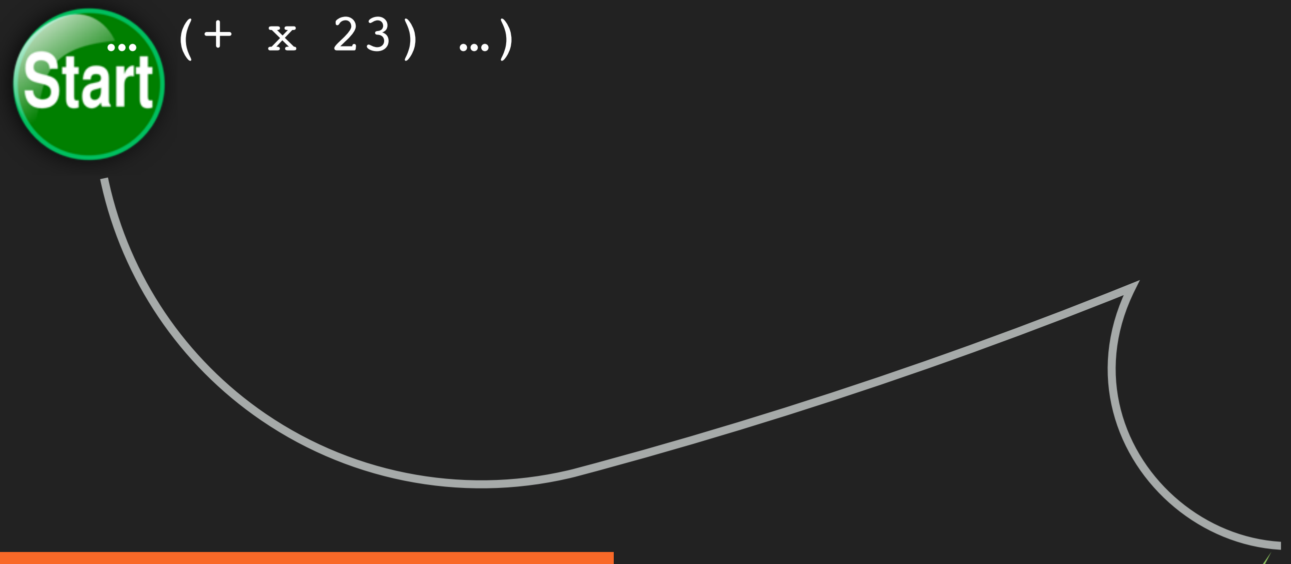
Are developers better off?

```
  0010 1000
+ 0100 0110
-----------
  0110 1110
```

THIS IS THE SOURCE (THOUGH NOT NECESSARILY THE LOGICAL BUG).

IN AN SOUND LANGUAGE SUCH AS ML:

```
(def main []
 ... (+ x 23) …)
```



Are users better off?

```
  0010 1000
+ 0100 0110

  0110 1110
```

SOMETHING BAD HAPPENED. SOMETHING WORSE MAY HAVE BEEN PREVENTED.

TAKE AWAY 2

- in an *unsound* language
- As a user, don't trust anything a program outputs.
- As a developer, beware of programs that seem work.
- Even segfaults can happen far, far away in different galaxy.

- in an *sound* language
- As a user, consider yourself lucky *when* you encounter an exception.
- As a developer, an EXN ch closer to a segfault.

The benefits of soundness make up a wide spectrum, but they shouldn't be ignored.

IS CLOJURE SOUND?

Clojure comes with a single type: “the program will run”.

Bob Harper citing Dana Scott

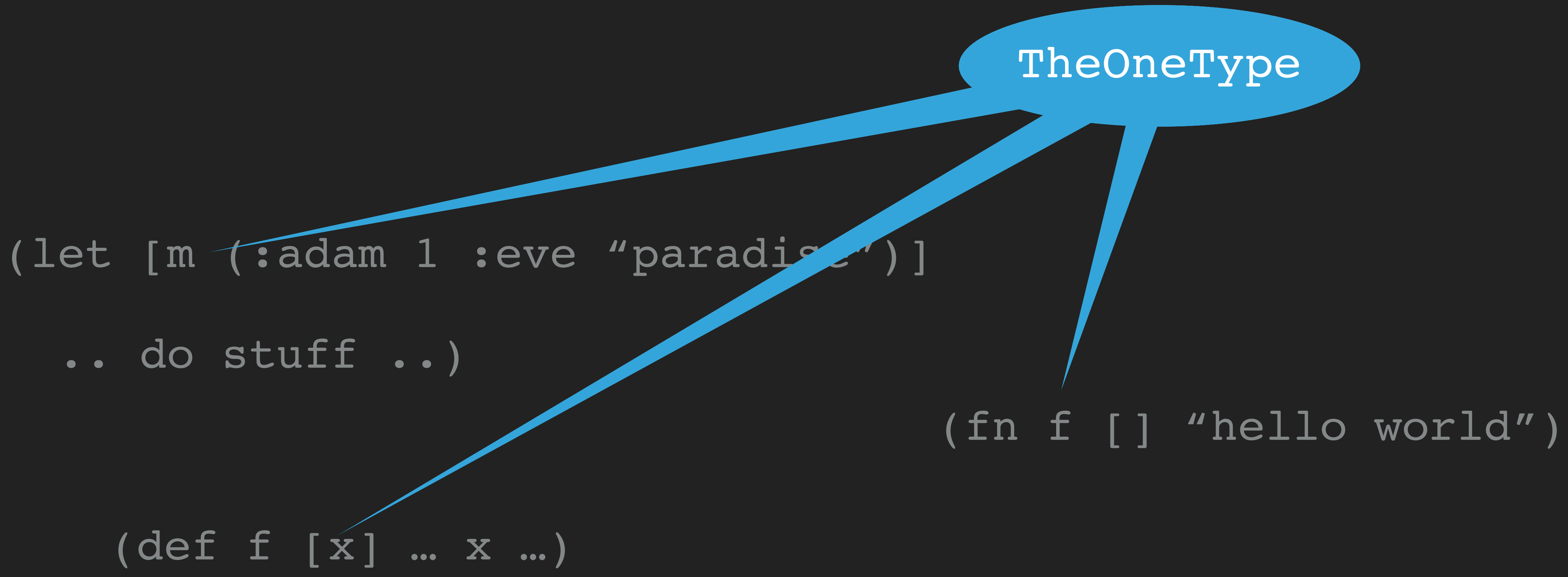
A language with a single type can be sound.

Matthias with Andrew Wright

THE USEFULNESS OF TYPES

THE EXPRESSIVE POWER OF TYPES

A single type isn't very useful, except that it frees the developer from writing it down everywhere.

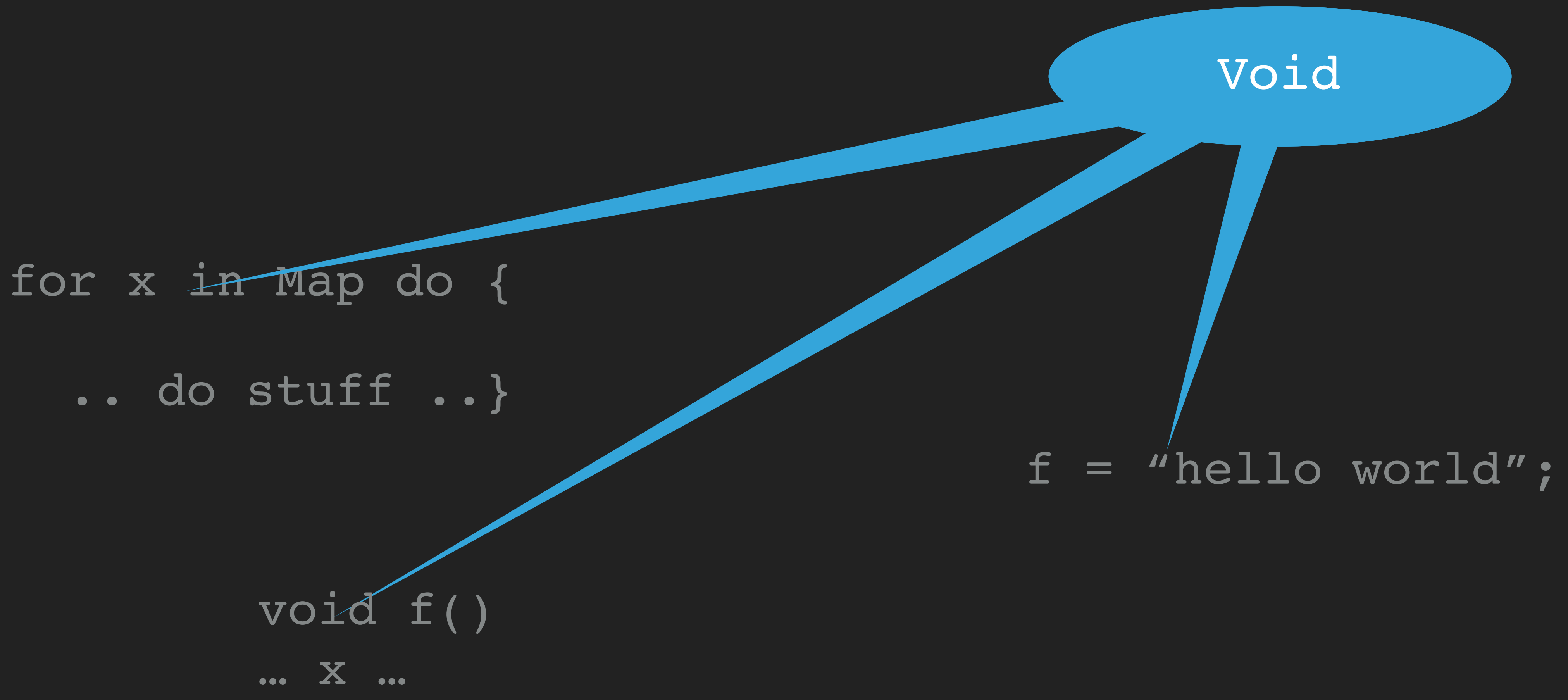


```
(let [m (:adam 1 :eve "paradise")]
  .. do stuff ..)
```

```
(fn f [] "hello world")
```

```
(def f [x] … x …)
```

In an imperative world, *Void* is almost like the one type that some languages provide.



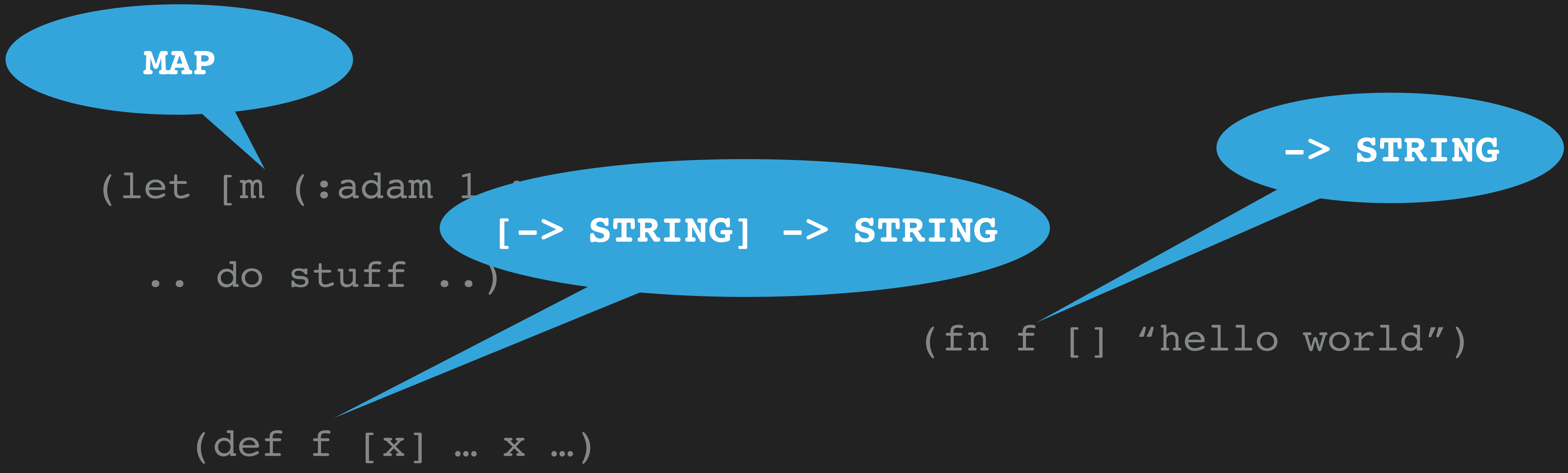
```
for x in Map do {
   .. do stuff ..}
```

```
f = “hello world”;
```

```
void f()
… x …
```

THE EXPRESSIVE POWER OF TYPES

Clojure developers have many types in their mind.
They just don’t have a language to write them down.



```
(let [m (:adam 1
  .. do stuff ..)
```

```
(fn f [] “hello world”)
```

```
(def f [x] … x …)
```

WE CAN ALWAYS WRITE DOWN TYPES AS COMMENTS.

Developers have these thoughts because this is how they ‘predict’ that their programs work correctly.

But some languages do not provide the means to write down these thoughts other than in comments.

And that is a problem, because code is written for others to understand the developers thoughts, and it accidentally runs on computers.

SO HERE IS A RACKET PROGRAM FROM 15 YEARS AGO

```
;; start reading here:

(define (compile-block decls statements next-label context

  (let* ([labels-with-numbers (map car statements)]

         [labels (map (lambda (l)

                        (if (stx-number? l)

                            (datum->syntax l  (string->symbol (format "~a" (syntax-e l)))  l l)

                            l))

                      labels-with-numbers)]

          .. 138 more lines like this .. ))
```

A MISTAKE!
OH NO!!

20 MINUTES LATER: THE L SHOULD HAVE
BEEN A 1, EASY!

WE LEARNED OUR LESSON. WE WROTE DOWN COMMENTS!

```
;; start reading here:
;; [Listof D 1 rations] [Li 2 Statement] [ 3 of Symbol] Bo 4 n -> Code
(define (compile-block A ls stat B ts next C el con D add-to E -level?)
  (let* ([labels-with-numbers (map car statements)]
         [labels (map (lambda (l)
                        (if (stx-number? l)
                            (datum->syntax l  (string->symbol (format "~a" (syntax-e l)))  l l)
                            l))
                      labels-with-numbers)]
          .. 138 more lines like this .. ))
```

4 INPUT TYPES FOR 5 PARAMETERS!

WHAT'S THE PROBLEM NOW?

WRITE DOWN TYPES WHEN YOU STRUGGLE TO RECONSTRUCT THEM, AND GET THEM CHECKED.

```
;; start reading here:
(: compile-block [Listof Declarations] [Listof Statement] [Listof Symbol] [Listof Symbol] Boolean
   -> Code)
(define (compile-block decls statements next-label context add-to-top-level?)
  (let* ([labels-with-numbers (map car statements)]
         [labels (map (lambda (l)
                        (if (stx-number? l)
                            (datum->syntax l  (string->symbol (format "~a" (syntax-e l)))  l l)
                            l))
                                         )]
                                 this .. ))
```

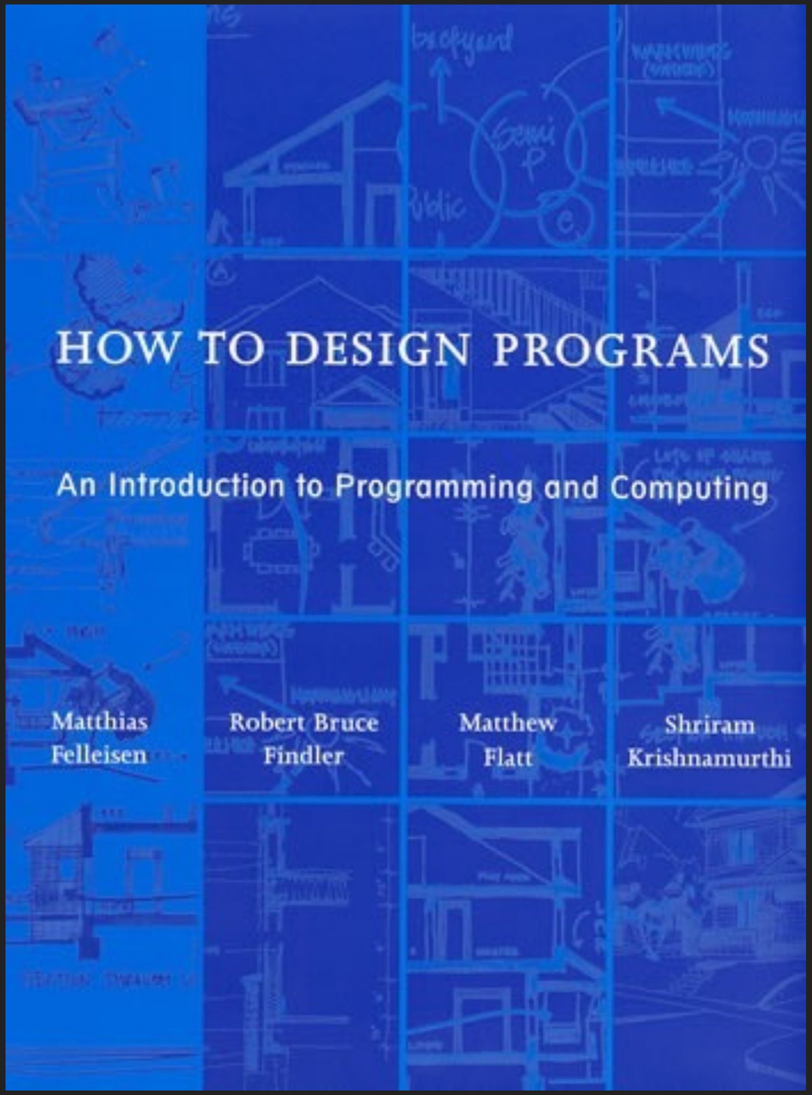
TYPES ARE CHECKED

A MAINTAINER CAN RELY ON THEM

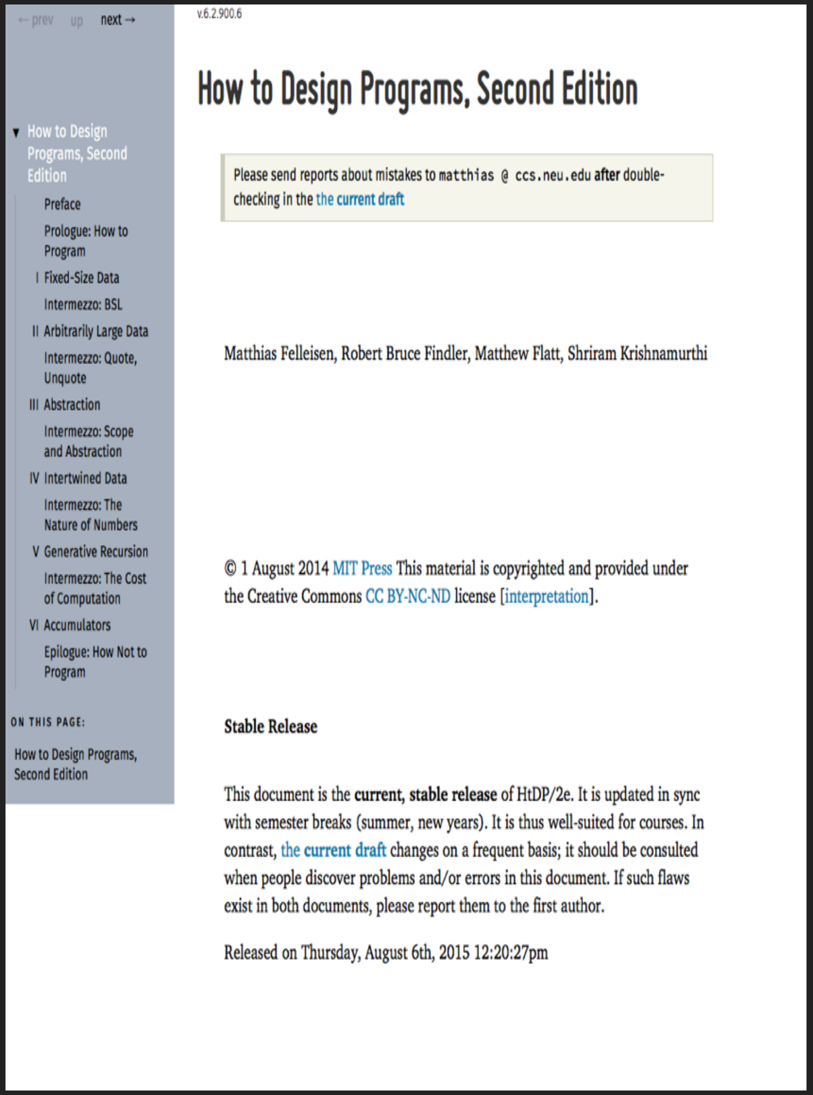
TYPES ALSO HELP DEVELOP MAINTAINABLE CODE IN THE FIRST PLACE

… even in an Untyped language such as Clojure …

MIT Press



ccs.neu.edu/home/matthias/HtDP2e/



← prev up next →

- ▾ How to Design Programs, Second Edition
  - Preface
  - Prologue: How to Program
  - I Fixed-Size Data
  - Intermezzo: BSL
  - II Arbitrarily Large Data
  - Intermezzo: Quote, Unquote
  - III Abstraction
  - Intermezzo: Scope and Abstraction
  - IV Intertwined Data
  - Intermezzo: The Nature of Numbers
  - V Generative Recursion
  - Intermezzo: The Cost of Computation
  - VI Accumulators
  - Epilogue: How Not to Program

ON THIS PAGE:

How to Design Programs, Second Edition

v.6.2.900.6

How to Design Programs, Second Edition

Please send reports about mistakes to `matthias @ ccs.neu.edu` after double-checking in the the current draft

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi

© 1 August 2014 MIT Press This material is copyrighted and provided under the Creative Commons CC BY-NC-ND license [interpretation].

Stable Release

This document is the **current, stable release** of HtDP/2e. It is updated in sync with semester breaks (summer, new years). It is thus well-suited for courses. In contrast, the current draft changes on a frequent basis; it should be consulted when people discover problems and/or errors in this document. If such flaws exist in both documents, please report them to the first author.

Released on Thursday, August 6th, 2015 12:20:27pm

TAKE AWAY 3

- All developers “think” types while they create code.
- In some languages they can’t write down those thoughts and get them cross-checked with the program.
- If they can’t write down types, they must reconstruct them.
- That costs time (with spouses, kids, vacation) and money.
  - What can we do about this?

CAN'T WE JUST INFER THE TYPES?

HOW ABOUT TYPE INFERENCE? HASKELL IS SO COOL.

No.

HOW ABOUT TYPE INFERENCE? ML HAS IT, TOO.

No, it’s really not a good idea.

HOW ABOUT TYPE INFERENCE?

Why are you asking again? I said ‘no’ twice.

CAN'T WE JUST RECONSTRUCT THEM FROM THE SOURCE TEXT?

after 15 years of research

- Hindley-Milner type inference (ML, Haskell)
- Hindley-Milner with revised type algebra
- type inference with set-based analysis
- … with s

FUNDAMENTALLY, WE NEED A LANGUAGE OF TYPES FIRST, AND UNTYPED LANGUAGES DON'T HAVE ONE BY DEFINITION.

ADDING TYPES TO AN UNTYPED LANGUAGE

HOW ADDING EXPLICIT STATIC TYPES OUGHT TO WORK

Incremental

When you have a code base of 500,000 lines, you *cannot* add types to all of this at once.

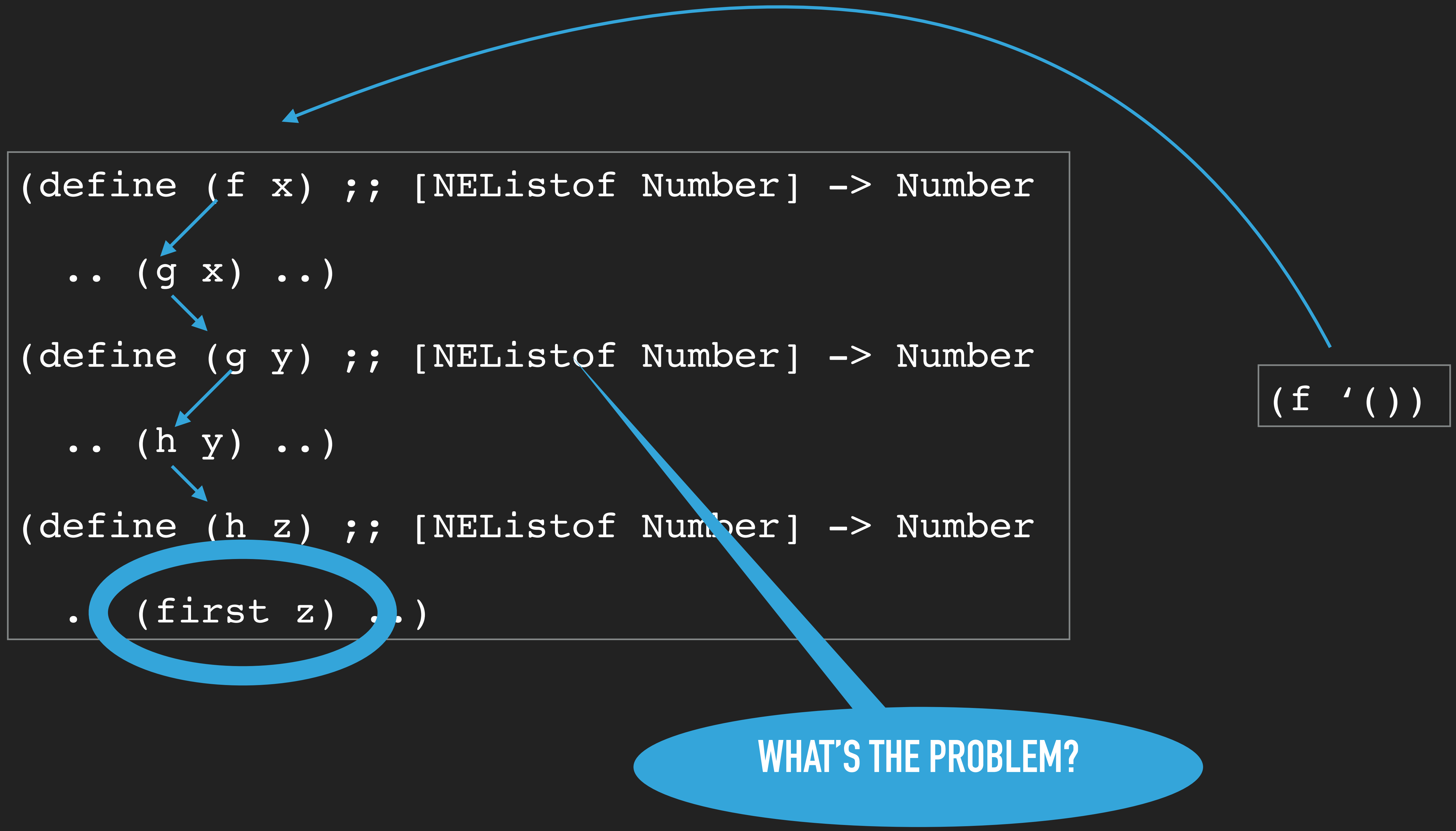
Idiomatic

Just add types. Otherwise code must not change, because it works.

Sound

The addition of types ought to narrow down the source of exceptions to cut down on future development time.

SOUNDNESS IN AN UNTYPED WORLD

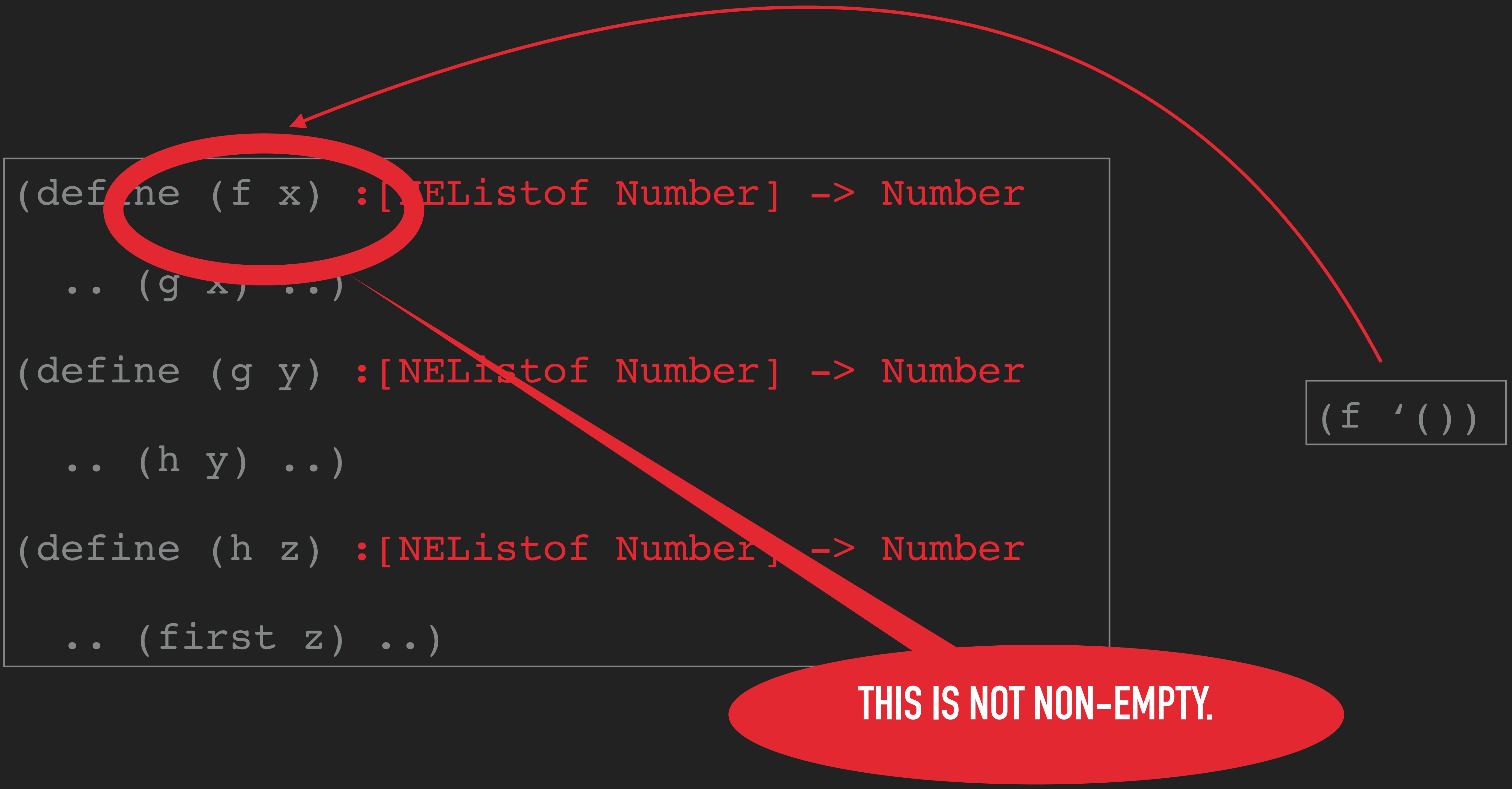


```
(define (f x) ;; [NEListof Number] -> Number
  .. (g x) ..)
(define (g y) ;; [NEListof Number] -> Number
  .. (h y) ..)
(define (h z) ;; [NEListof Number] -> Number
  .. (first z) ..)
```

```
(f '())
```

WHAT'S THE PROBLEM?

SOUNDNESS IN AN TYPED WORLD



```
(define (f x) :[NEListof Number] -> Number
  .. (g x) ..)
(define (g y) :[NEListof Number] -> Number
  .. (h y) ..)
(define (h z) :[NEListof Number] -> Number
  .. (first z) ..)
```

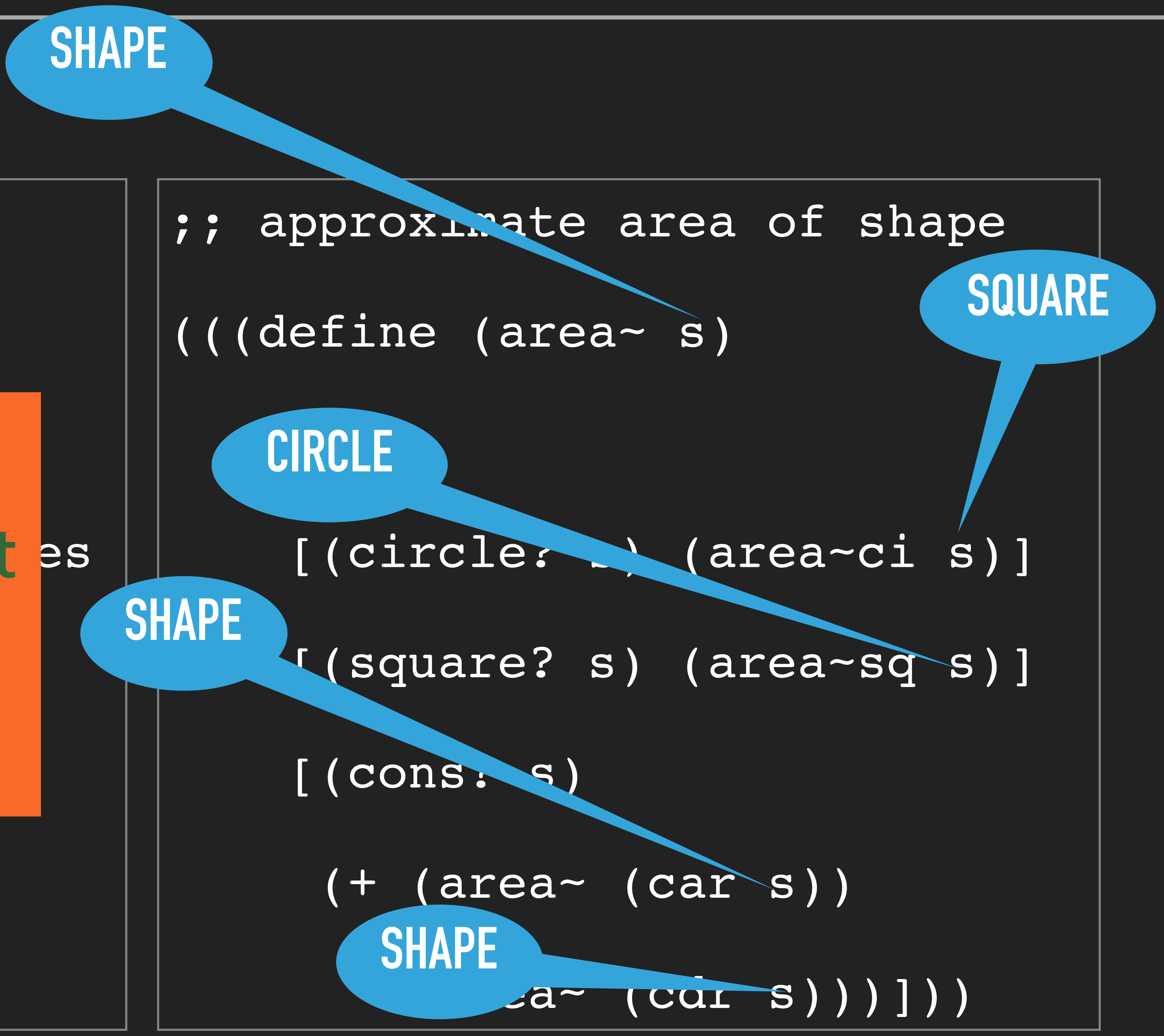
```
(f ‘())
```

THIS IS NOT NON-EMPTY.

UNTYPED IDIOMS COME FROM SET-BASED THINKING AND BASIC LOGIC

```
;; shape is one of:
;; — [square size]
;;
;;                              es
;;
(de
(define c [circle 2])
(define p (cons s c))
```

One and the same variable has different types – depending on where it occurs.



```
;; approximate area of shape
(((define (area~ s)

    [(circle? s) (area~ci s)]
    [(square? s) (area~sq s)]
    [(cons? s)
     (+ (area~ (car s))
        (area~ (cdr s)))]))
```

A TYPE SYSTEM FOR AN UNTYPED LANGUAGE MUST UNDERSTAND THIS TOO.

```
;; shape is one of:
;; — [square size]
;;
;;                                           es
;;
(de
(define c [circle 2])
(define p (cons s c))
```

Occurrence typing combines simple set-based reasoning with basic logic.

```
(: area~ (-> Shape Number))
(((define (area~ s)

     [(circle? s) (area~ci s)]
     [(square? s) (area~sq s)]
     [(cons? s)
      (+ (area~ (car s))
         (area~ (cdr s)))]))
```

SHAPE

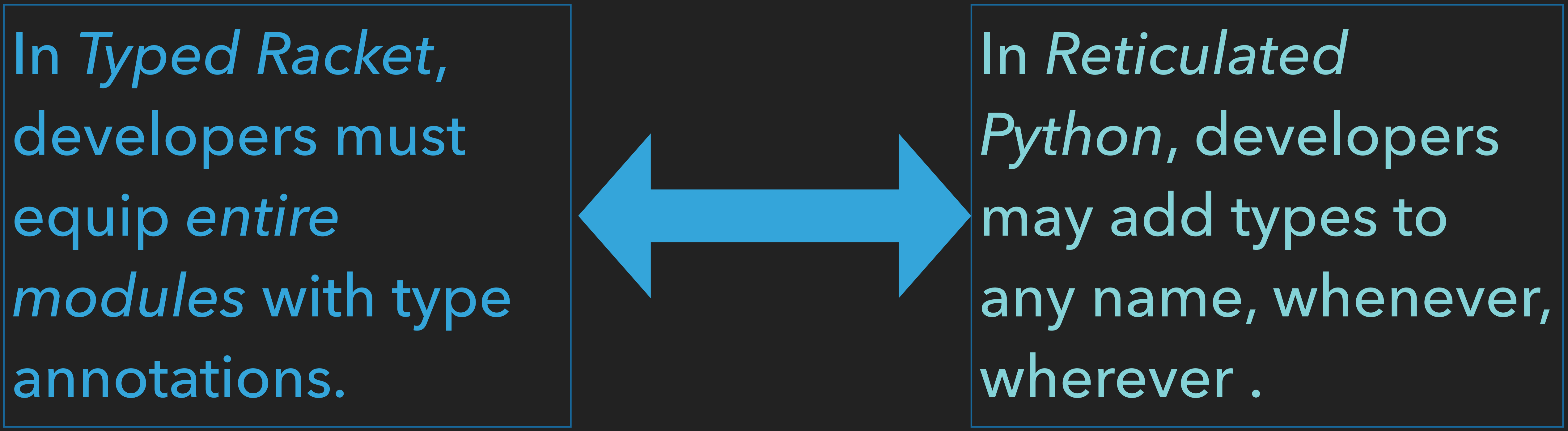
SQUARE

CIRCLE

SHAPE

SHAPE

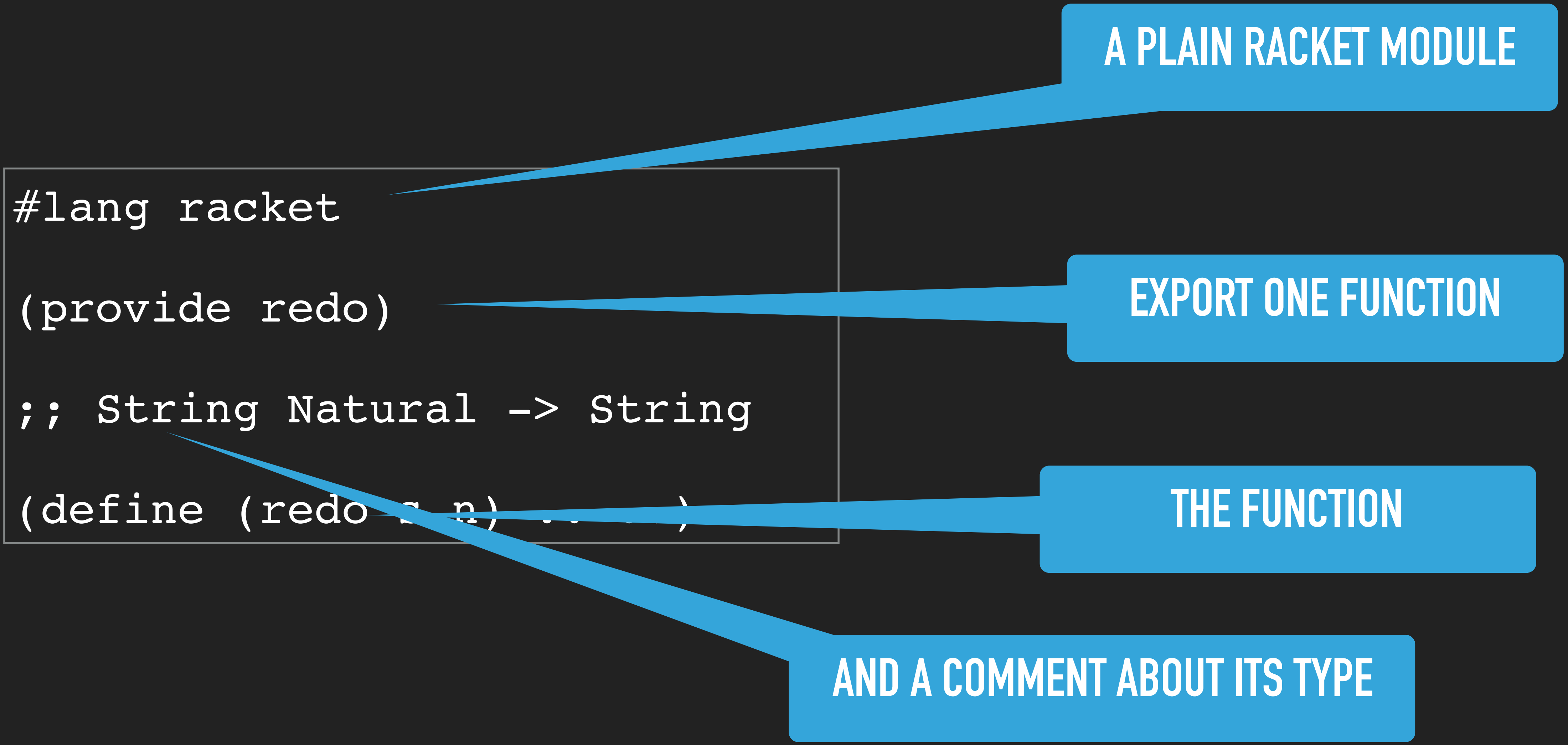
HOW ARE TYPES ADDED INCREMENTALLY?



In *Typed Racket*, developers must equip *entire modules* with type annotations.

In *Reticulated Python*, developers may add types to any name, whenever, wherever .

SOUNDNESS REVISITED



```
#lang racket

(provide redo)

;; String Natural -> String

(define (redo s n) ... )
```

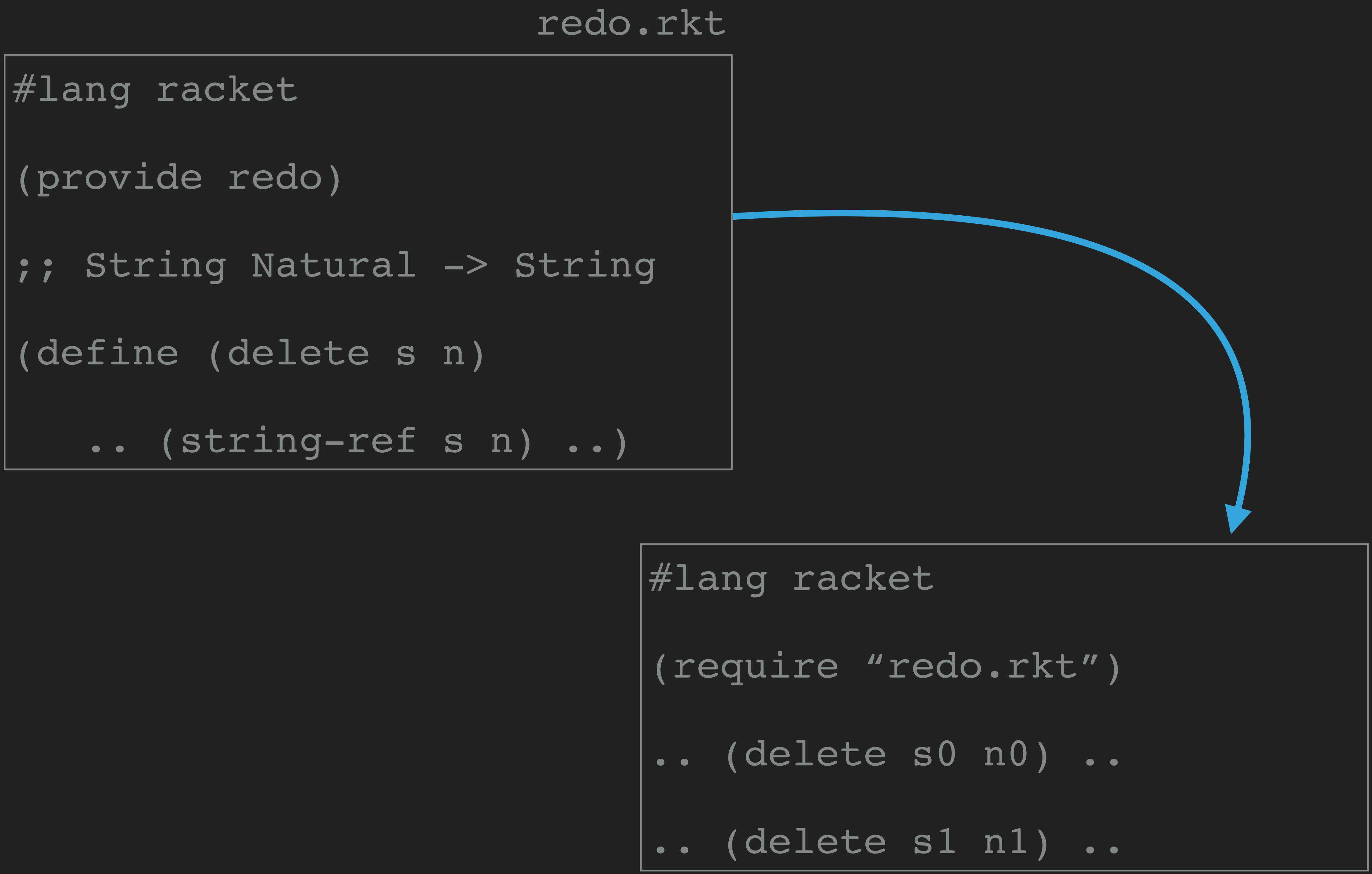
A PLAIN RACKET MODULE

EXPORT ONE FUNCTION

THE FUNCTION

AND A COMMENT ABOUT ITS TYPE

SOUNDNESS REVISITED

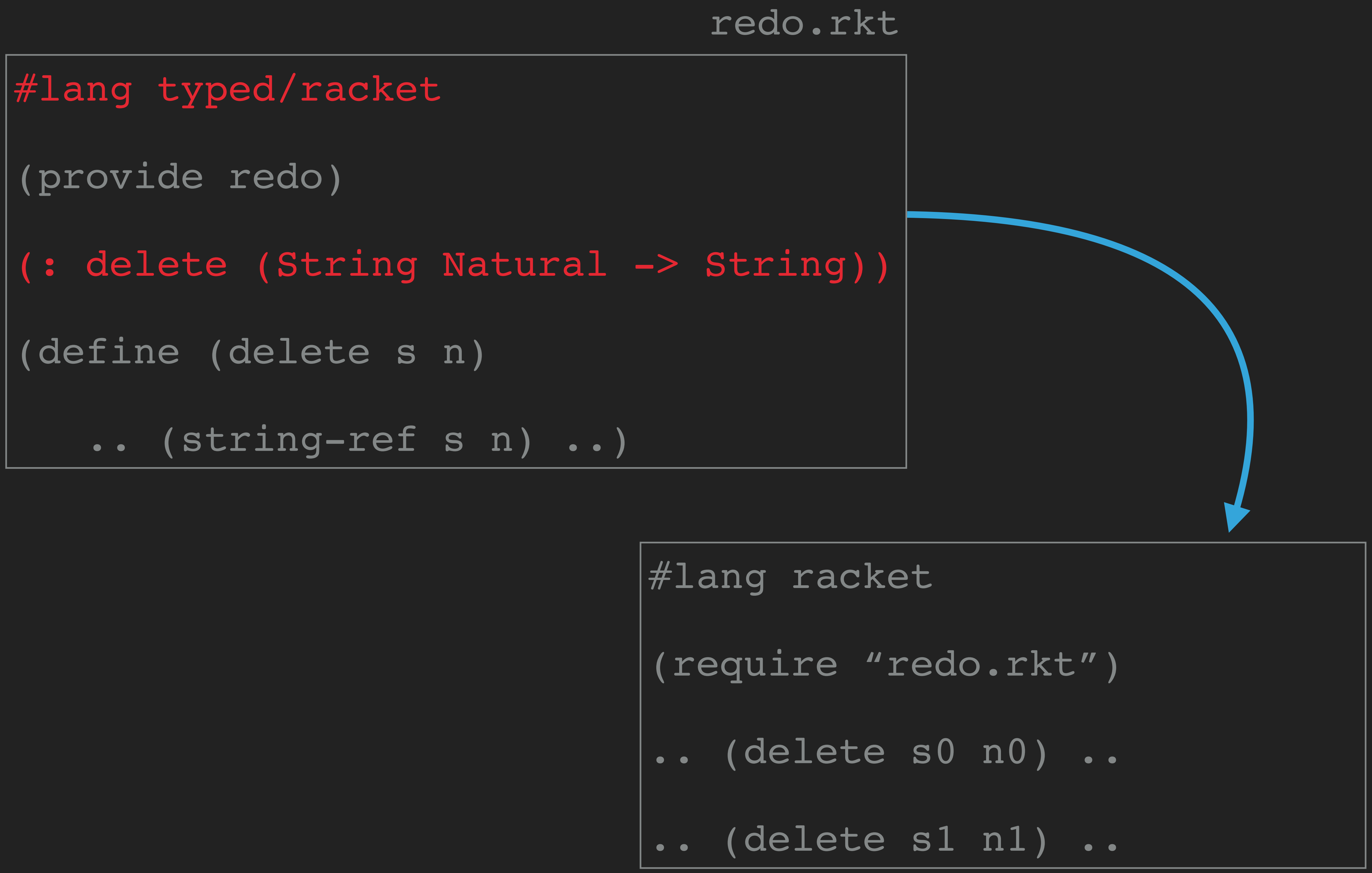


redo.rkt

```
#lang racket

(provide redo)

;; String Natural -> String

(define (delete s n)

   .. (string-ref s n) ..)
```

```
#lang racket

(require “redo.rkt”)

.. (delete s0 n0) ..

.. (delete s1 n1) ..
```

SOUNDNESS REVISITED



redo.rkt

```
#lang typed/racket

(provide redo)

(: delete (String Natural -> String))

(define (delete s n)

   .. (string-ref s n) ..)
```

```
#lang racket

(require “redo.rkt”)

.. (delete s0 n0) ..

.. (delete s1 n1) ..
```

SOUNDNESS REVISITED

redo.rkt

```
#lang typed/racket

(provide redo)

(: delete (String Natural -> String))

(define (delete s n)

      (string-ref s n) ..)
```

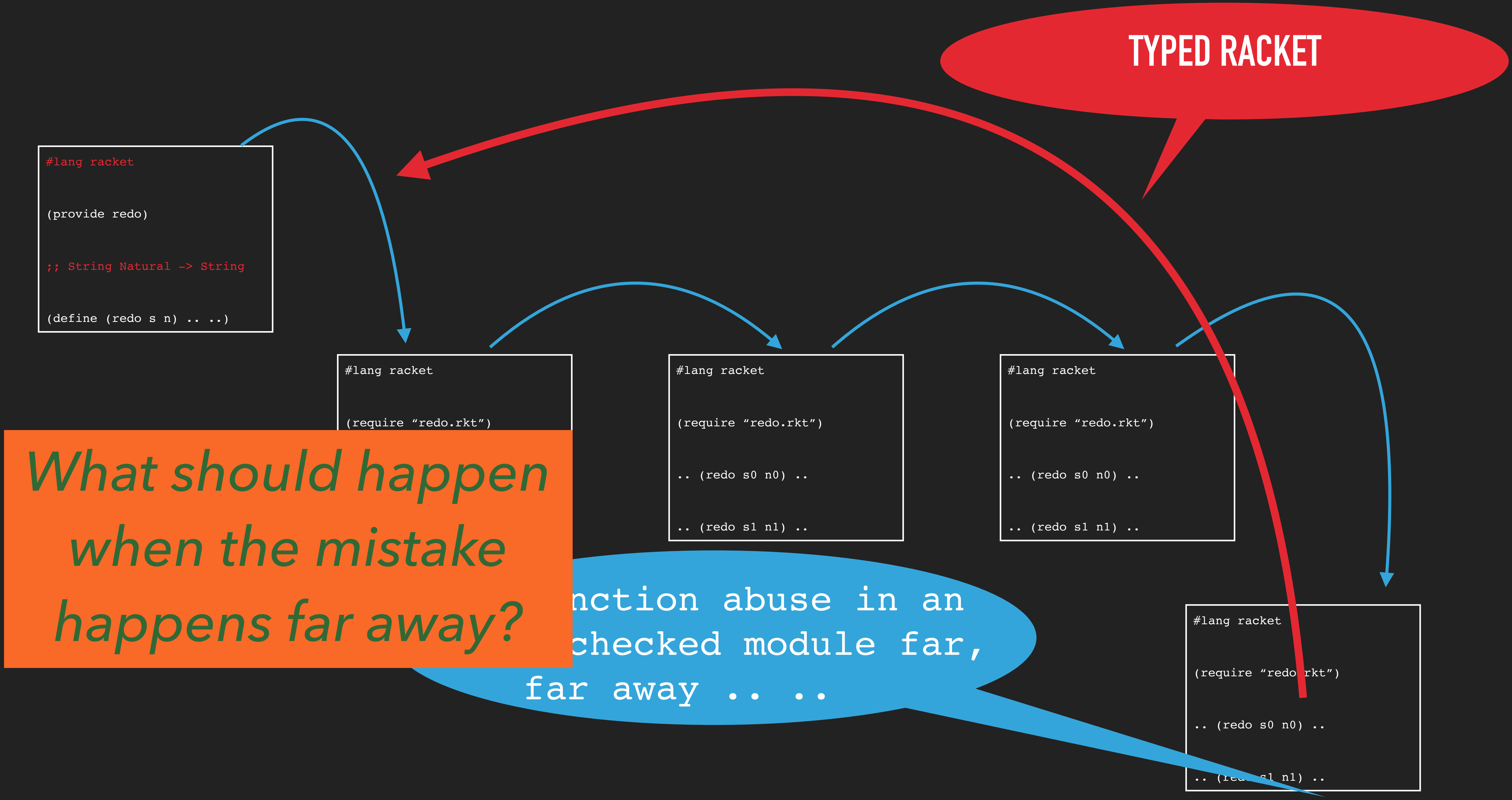
WHAT'S THE PROBLEM?

What should happen?

```
#lang racket

(require “redo.rkt”)

.. (delete s0 n0) ..

.. (delete 5 “hello”) ..
```

Function abuse in
an unchecked module

SOUNDNESS REVISITED



TYPED RACKET

```
#lang racket

(provide redo)

;; String Natural -> String

(define (redo s n) .. ..)
```

```
#lang racket

(require “redo.rkt”)
```

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

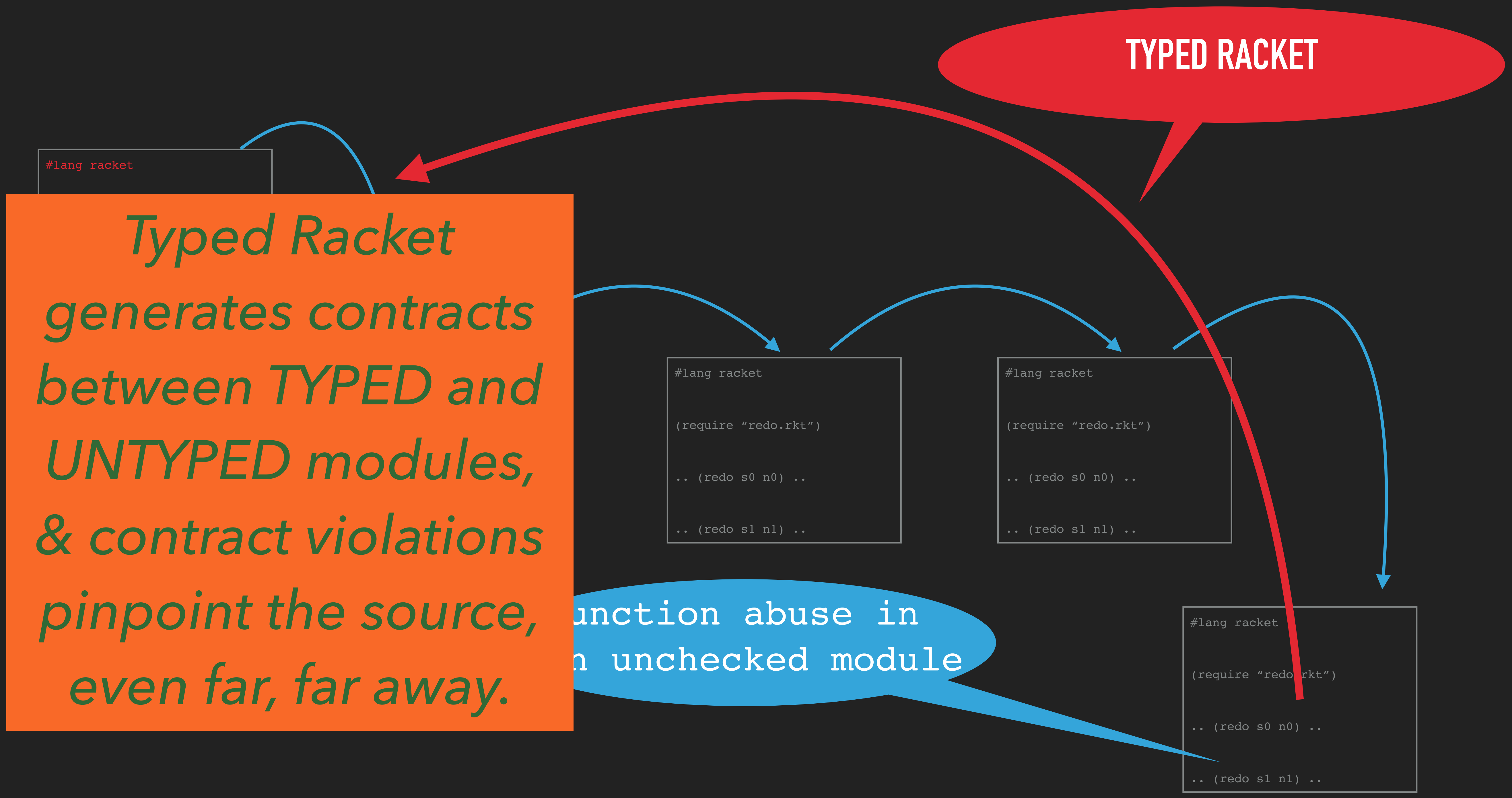
```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

nction abuse in an
checked module far,
far away .. ..

What should happen when the mistake happens far away?

SOUNDNESS REVISITED



```
#lang racket
```

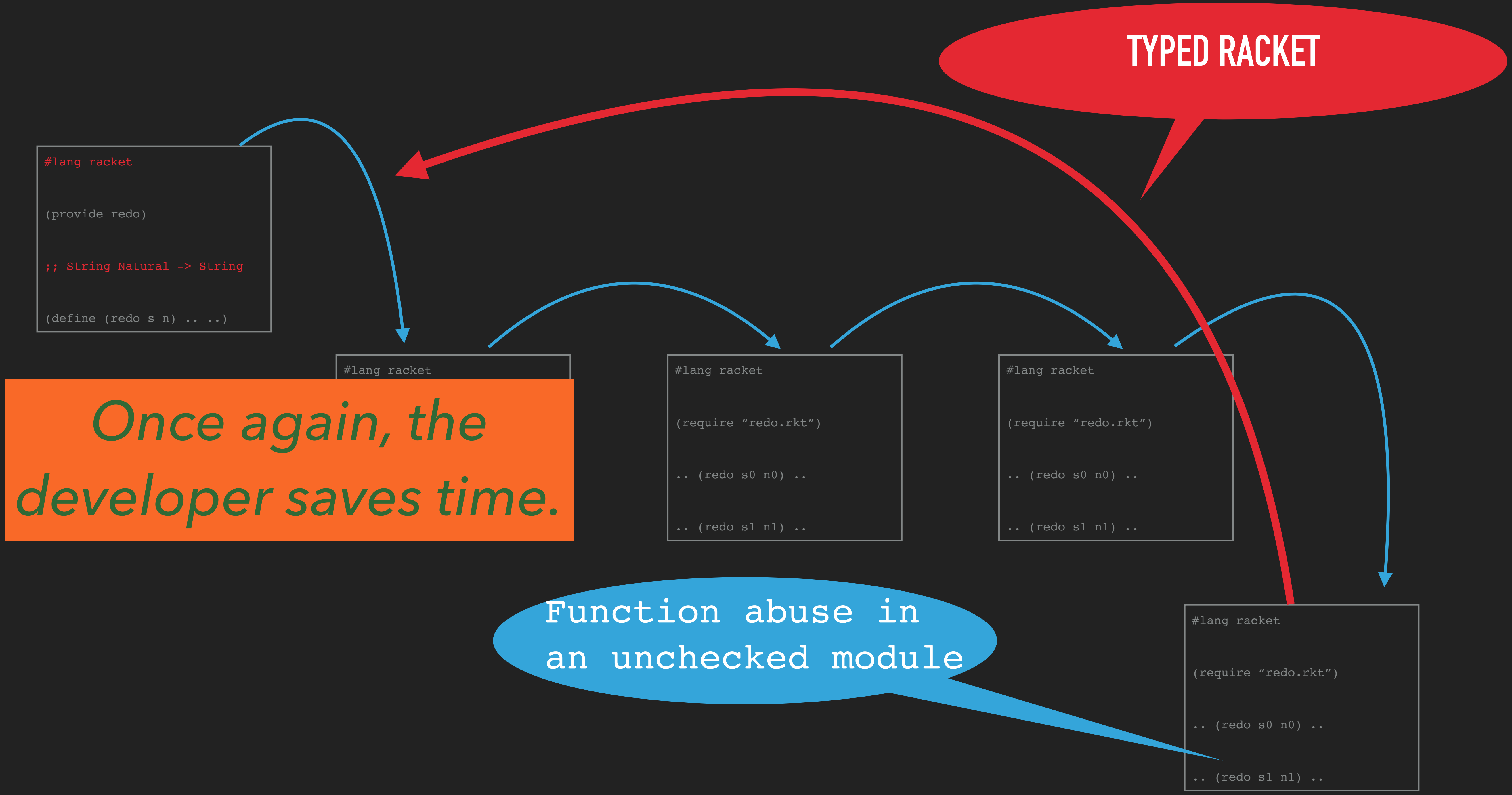
```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

Typed Racket generates contracts between TYPED and UNTYPED modules, & contract violations pinpoint the source, even far, far away.

SOUNDNESS REVISITED



TYPED RACKET

```
#lang racket

(provide redo)

;; String Natural -> String

(define (redo s n) .. ..)
```

```
#lang racket
```

Once again, the developer saves time.

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

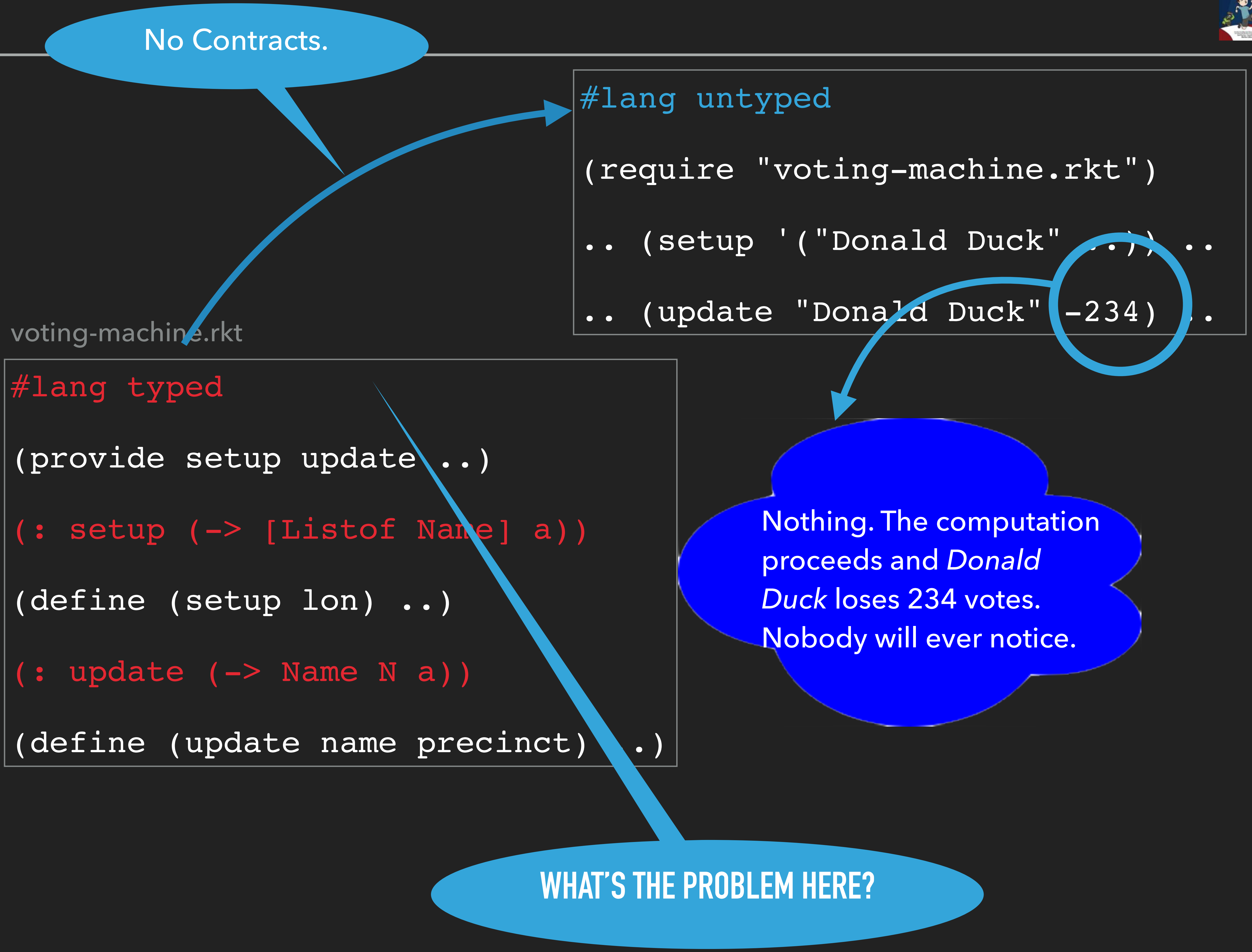
```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

Function abuse in an unchecked module

```
#lang racket

(require “redo.rkt”)

.. (redo s0 n0) ..

.. (redo s1 n1) ..
```

DO WE NEED TO KNOW THIS?

What happens if we don’t generate contracts?



No Contracts.

```
#lang untyped

(require "voting-machine.rkt")

.. (setup '("Donald Duck" ..)) ..

.. (update "Donald Duck" -234) ..
```

voting-machine.rkt

```
#lang typed

(provide setup update ..)

(: setup (-> [Listof Name] a))

(define (setup lon) ..)

(: update (-> Name N a))

(define (update name precinct) ..)
```

Nothing. The computation proceeds and *Donald Duck* loses 234 votes. Nobody will ever notice.

WHAT'S THE PROBLEM HERE?

DO WE NEED TO KNOW THIS?

And that’s precisely what *Typed Clojure* does ~ it masks the bugs.

Without contracts, you get all the unsoundness of C++ back.

TAKE AWAY 4

Types for Untyped languages

- .. must speak the *grown idioms.*
- .. must allow *gradual additions.*
- .. ought to come with *soundness* because
  - it reduces developer time
  - it won't mask errors

THE COST IS AN OPEN PROBLEM.

THE BIG TAKE-AWAY

THE BIG TAKE AWAY

> Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. *John F. Woods*

THE BIG TAKE-AWAY: VALUE YOUR DEVELOPERS AND USERS

UNTYPED PROGRAMMING MAKES FOR A GOOD START

ADD TYPES IF YOU VALUE YOUR DEVELOPER'S TIME.

ADD TYPES IF YOU VALUE YOUR GRANDMOTHER'S LIFE.

WE ARE BUILDING HYBRID LANGUAGES BUT TO SOME EXTENT, IT'S ALL STILL RESEARCH.

THE END

- Matthew Flatt, the Racket Man
- Robby Findler, Dr. Racket, a Man with Contracts
- Cormac Flanagan, Mr. Spidey
- Stevie Strickland, with Class
- Sam Tobin-Hochstadt, Typed
- Asumu Takikawa, TOOR
- Ben Greenman and Max New, Performance Matters
- Alex Knauth, Alexis King, 2 wonderful freshmen

- … and many many others for contributions to the code base
- and even more for theoretical underpinnings, ideas, etc.

QUESTIONS?