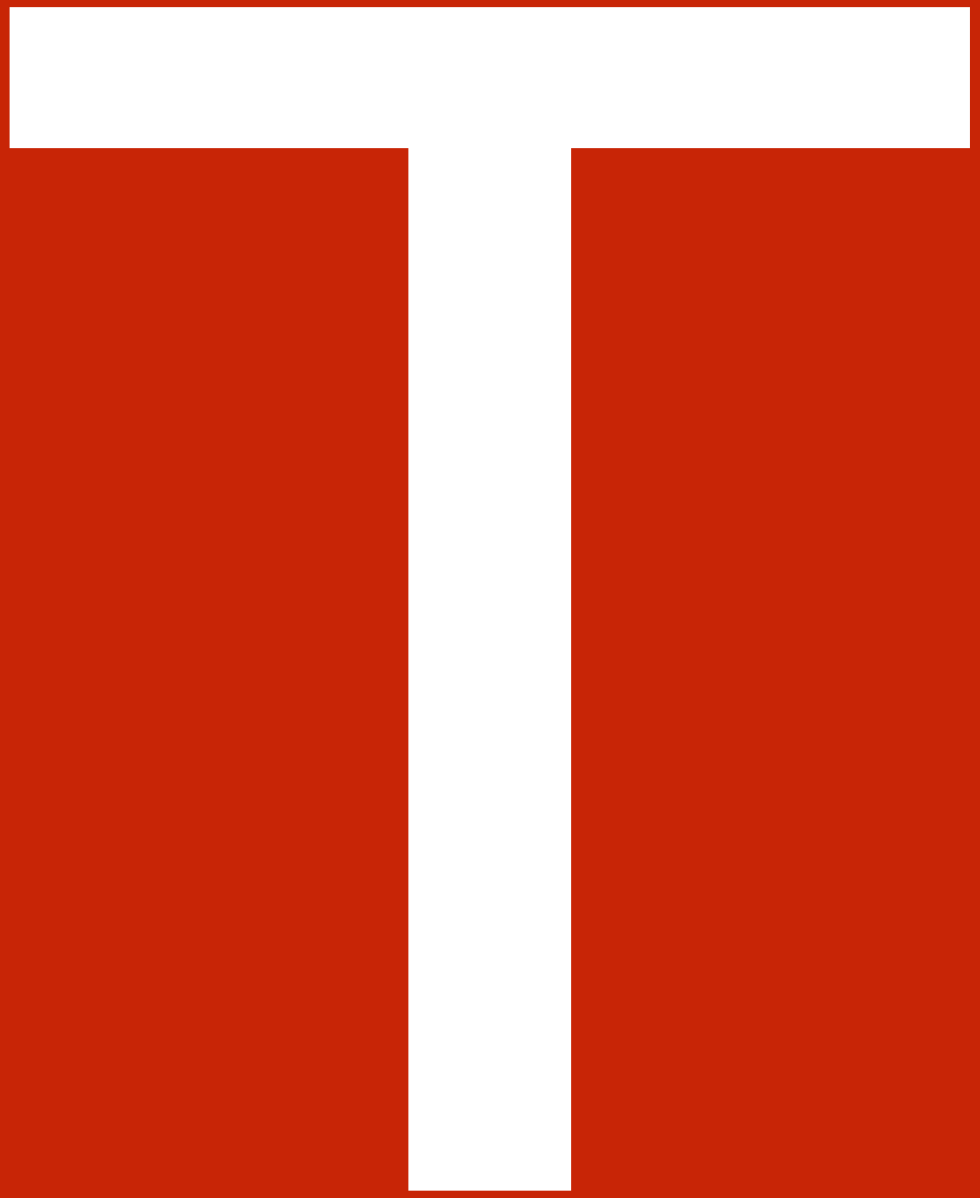
This is is not a type:



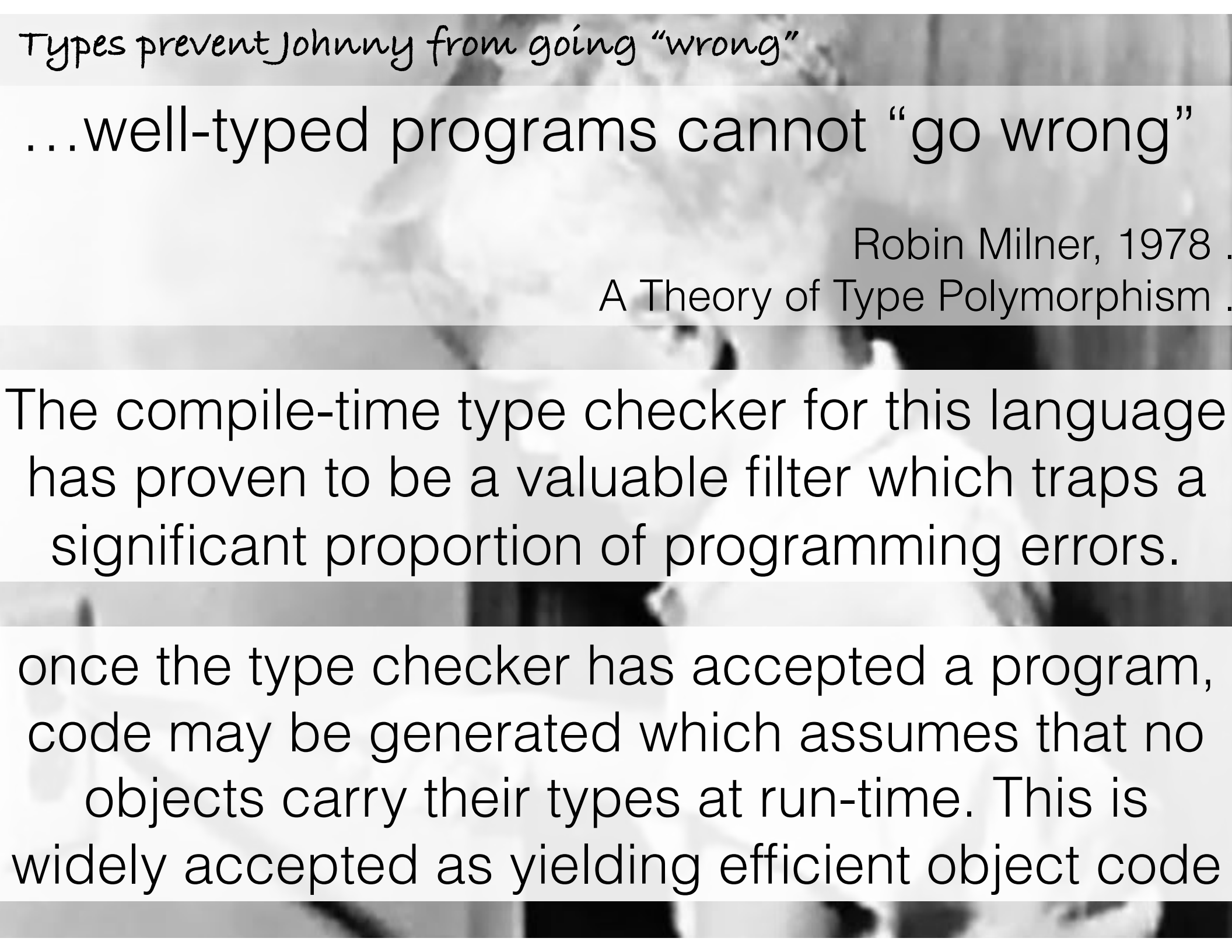
Gradual typing for R

Jan Vitek, Northeastern University

Types enhance productivity

The Iron Rolling Mill by Adolf Menzel

```
function(x) {
  var y = x ? 2 : "Y"
  if x    y += "ES"
  else  y += 40
  return y
}
```



Types prevent Johnny from going "wrong"

…well-typed programs cannot "go wrong"

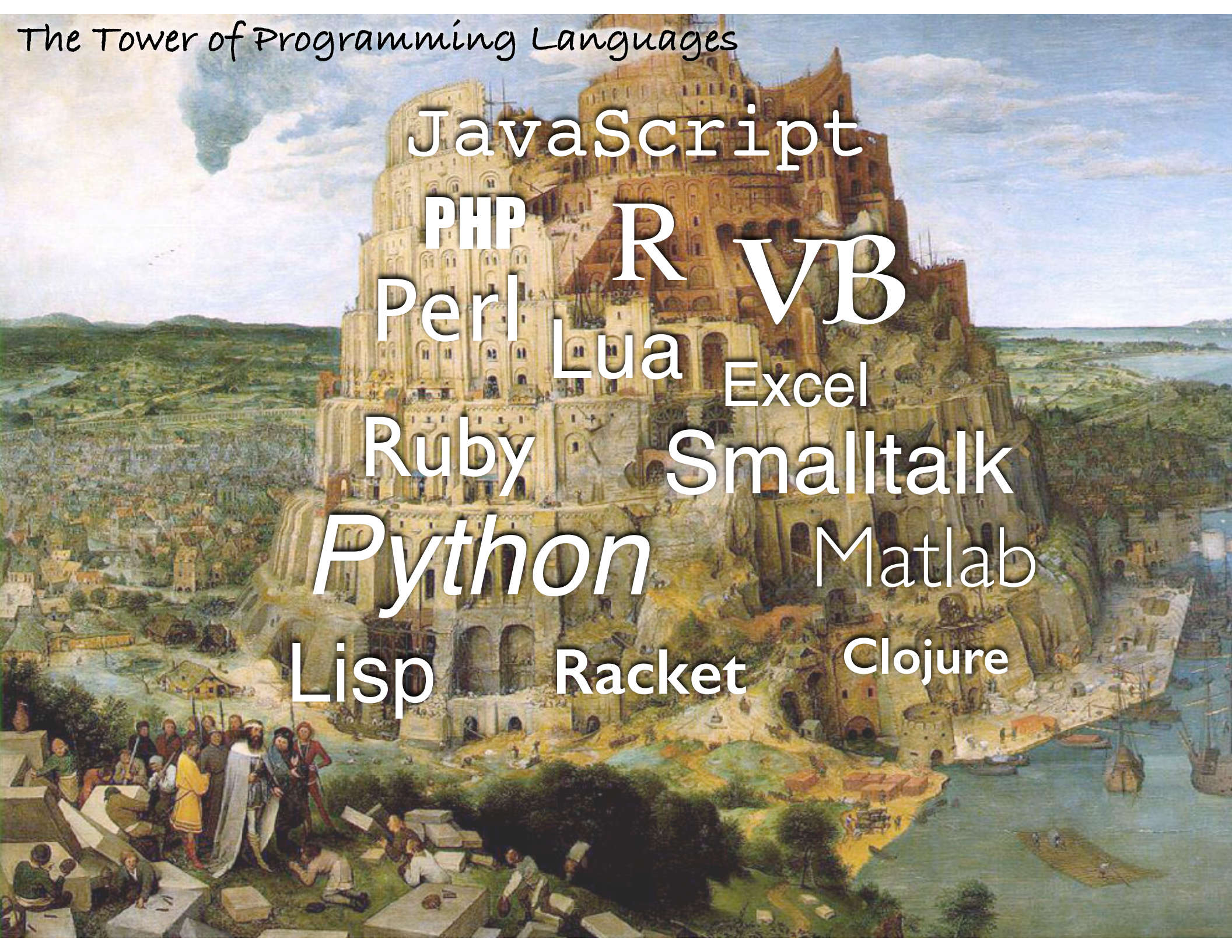
Robin Milner, 1978 .
A Theory of Type Polymorphism .

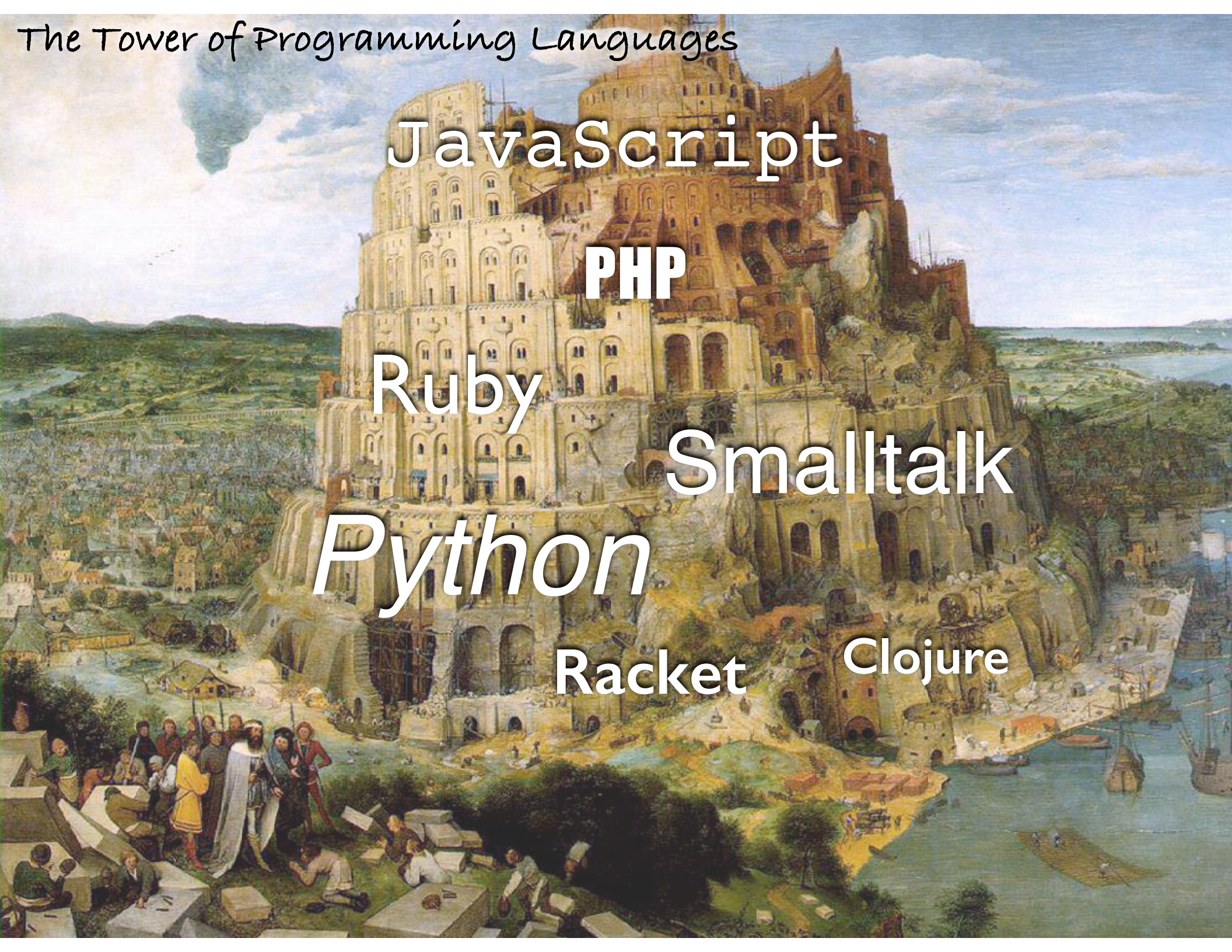
The compile-time type checker for this language has proven to be a valuable filter which traps a significant proportion of programming errors.

once the type checker has accepted a program, code may be generated which assumes that no objects carry their types at run-time. This is widely accepted as yielding efficient object code

```
m ( Object [] argh ) {
   argh [0] = new Object ()
}

m( new String[“hi”] )
```


The Tower of Programming Languages
JavaScript
PHP
R
VB
Perl
Lua
Excel
Ruby
Smalltalk
Python
Matlab
Lisp
Racket
Clojure


The Tower of Programming Languages
JavaScript
PHP
Ruby
Smalltalk
Python
Racket
Clojure

The Gradual Typing Hypothesis

A gradual type system can **gradually** enrich "scripts" with **explicit** and **sound** types **without changing** code

— Matthias Felleisen, TLDI 2010

From Static to Dynamic

Adding dynamic types to C♯

ECOOP 2010

Gavin Bierman[1], Erik Meijer[2], and Mads Torgersen[2]

```
dynamic doc = HtmlPage.Document;
dynamic win = HtmlPage.Window;
string latitude, longitude, name, address;
...

dynamic map = win.CreateInstance("VEMap", "myMap");
map.LoadMap();
map.DeleteAllShapes();

var x = win.CreateInstance("VELatLong", latitude, longitude);
var pin = map.AddPushpin(x);

doc.Title = "Information for: " + name;
pin.SetTitle(name);
pin.SetDescription(address);
map.SetCenterAndZoom(x, 9);
```

Runtime errors possible in dynamic operations.

Otherwise sound.

Optional types

```
<?hh // strict

function annotating(?string $x): string {
  return $x === null ? "Hello" : "Bye";
}

function f(): void {
  // UNSAFE
  annotating(6);
}
function g(): void {
  // UNSAFE
  annotating(true);
}
```

Runtime errors may occur anywhere, the dynamic type system ensure memory safety but programs are unsound

Array Operators Using Multiple Dispatch

A design methodology for array implementations in dynamic languages

Jeff Bezanson Jiahao Chen Stefan Karpinski Viral Shah Alan Edelman

```
type Rational{T<:Integer} <: Real
    num::T
    den::T

    function Rational(num::T, den::T)
        if num == 0 && den == 0
            error("invalid rational: 0//0")
        end
        g = gcd(den, num)
        new(div(num, g), div(den, g))
    end
end
```

Runtime errors may occur at any function invocation as there are no checks, the dynamic type system ensure memory safety but programs are unsound

The Design and Implementation of Typed Scheme

POPL 2008

Sam Tobin-Hochstadt Matthias Felleisen

PLT, Northeastern University

```
#lang racket

(provide (struct-out pt)
         distance)

(struct pt (x y))

; distance : pt pt -> real
(define (distance p1 p2)
  (sqrt (+ (sqr (- (pt-x p2) (pt-x p1)))
           (sqr (- (pt-y p2) (pt-y p1))))))
```

```
#lang typed/racket

(require/typed "distance.rkt"
    [#:struct pt ([x : Real] [y : Real])]
    [distance (-> pt pt Real)])

(distance (pt 3 5) (pt 7 0))
```

Typed Racket is sound but does not preserve all correct untyped programs

Errors can occur anywhere but are caught and properly blamed

Thorn—Robust, Concurrent, Extensible Scripting on the JVM

OOSPLA 2009

Bard Bloom[1], John Field[1], Nathaniel Nystrom[2*], Johan Östlund[3],
Gregor Richards[3], Rok Strniša[4], Jan Vitek[3], Tobias Wrigstad[5†]

[1] IBM Research [2] University of Texas at Arlington [3] Purdue University
[4] University of Cambridge [5] Stockholm University

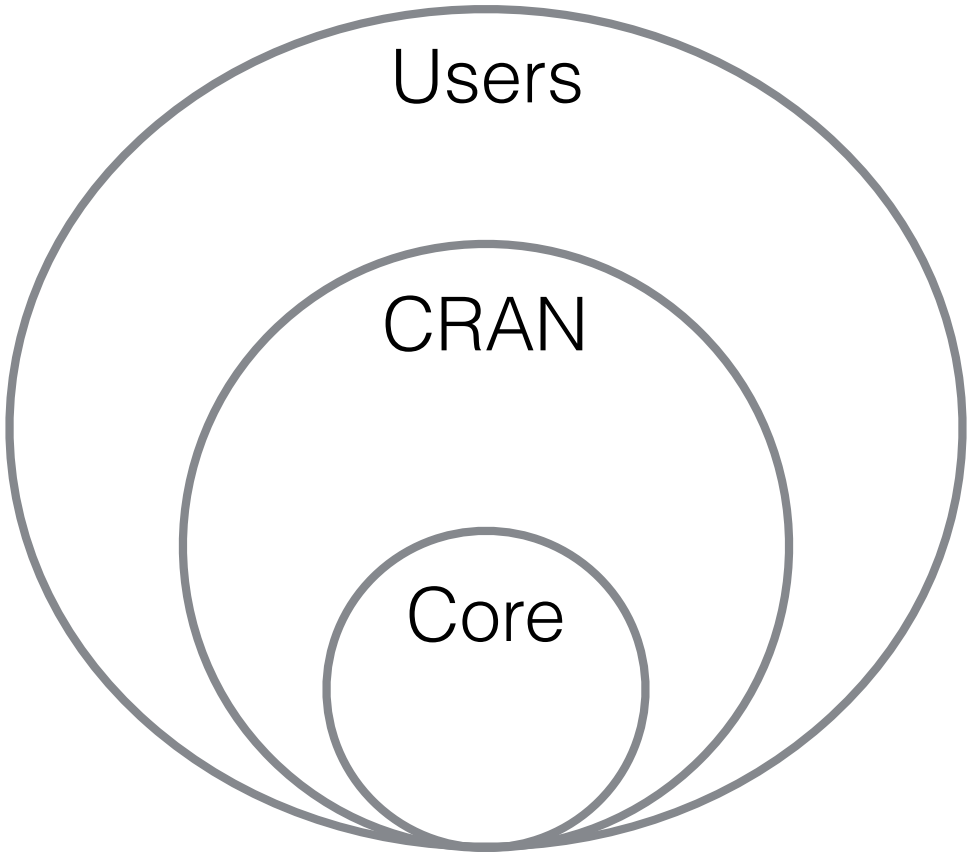
```
fun move(p: like Point) {
  x := p.getX();
  y := p.getY();
  # p.hog();     raises compile-time err
}

fun move(p: Point) {
  x := p.getX();
  y := p.getY();
}
```

Runtime errors may occur in dynamic and like type code, they are dynamically caught

Everywhere else we have soundness

What Types for R?



Why Types for R?

```
function (x, na.rm = FALSE, dims = 1L) {
    if (is.data.frame(x))
        x <- as.matrix(x)
    if (!is.array(x) || length(dn <- dim(x)) < 2L)
        stop("'x' must be an array of at least 2D")
    if (dims < 1L || dims > length(dn) - 1L)
        stop("invalid 'dims'")
```

Use types to systematize expectations made by a function on its arguments

```
function
    (x :~ Matrix(N,…),
     na.rm :: Logical = FALSE,
     dims :: Range(1,dim(x))= 1L){
```

Why Types for R?

```
function (x, i) {
    while ( x < i )
        x++
        …
```

Use types to avoid unnecessary allocation and to generate efficient native code

```
function {T<:Numeric}(x :: T, i :: T) {
    while ( x < i )
        x++
        …
```

```
x :: Int
```

```
x :: Int[]
```

```
x :: Int[2]
```

```
x :: Int[2,…]
```

```
x ~: Logical
```

```
x :: Int[2,…]
```

```
{N}(x::Int[N],y::Logical[N])
```

Open questions

- Types for data frames?
- Types for S3, S4, and … ?
- Types for functions…