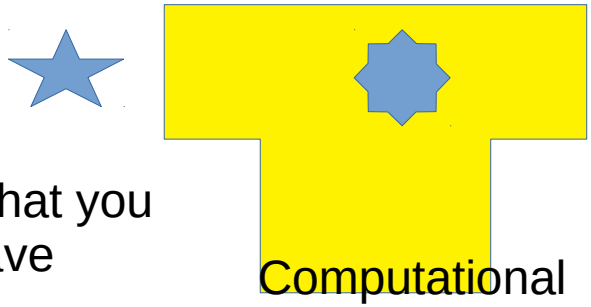
Adaptive Compilation

Jecel Mattos de Assumpção Jr

November 14, 2019

Computer Science ?

processes



What you have

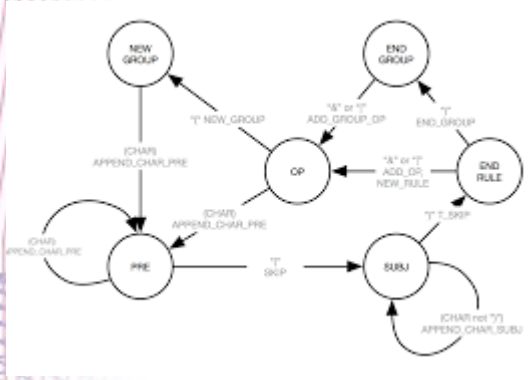
Computational model

What you want

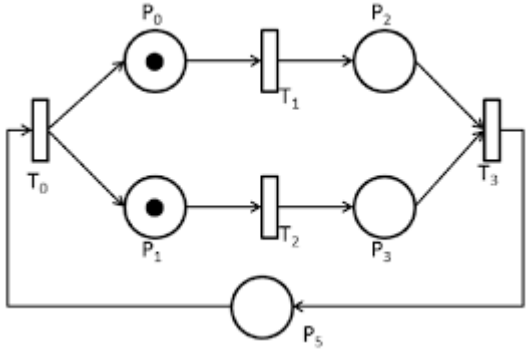
Computer science is no more about computers than astronomy is about telescopes.

Edsger Dijkstra

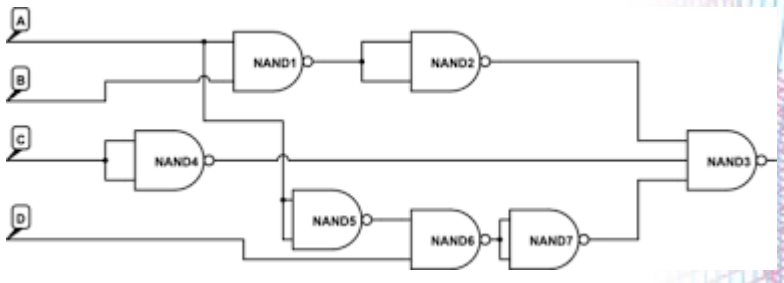
problem



Finite state machines



Petri nets



NANDs and wires

1936

Church-Turing Thesis

Alan Turing

Alonzo Church

Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.

Alan Perlis

230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions

AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

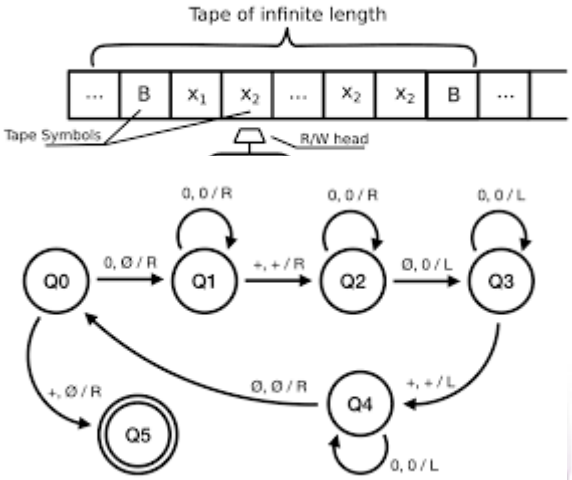
By ALONZO CHURCH.

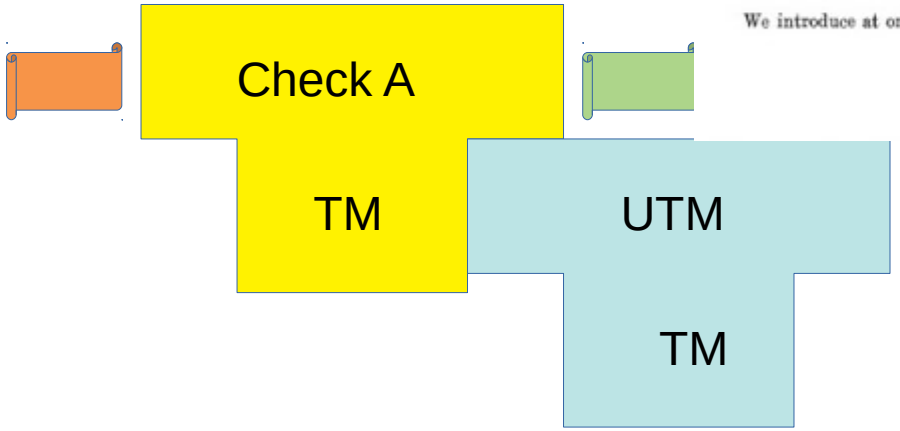
The purpose of the present paper is to propose a definition of effective calculability[3] which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which problems of this class are often stated, and to show, by means of an example, that not every problem of this class is solvable.

. . .

We introduce at once the following infinite list of abbreviations,

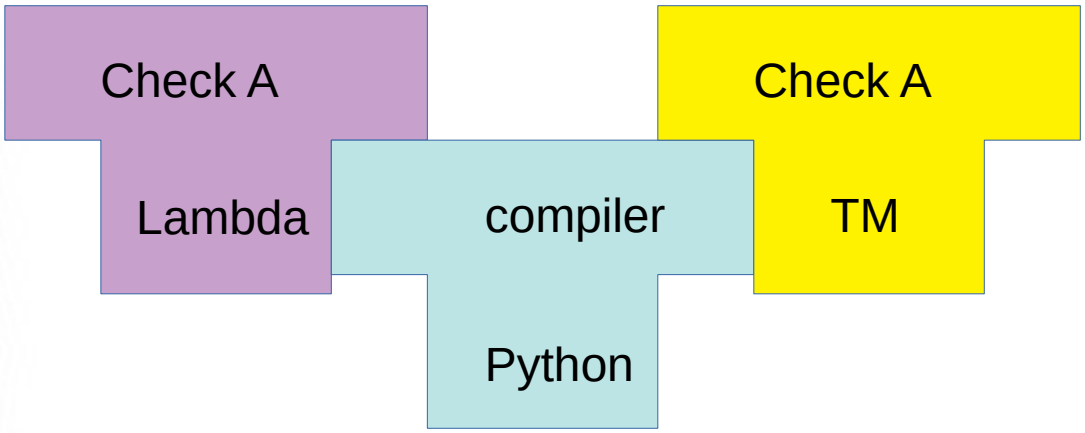
$$1 \rightarrow \lambda ab \cdot a(b),$$
$$2 \rightarrow \lambda ab \cdot a(a(b)),$$
$$3 \rightarrow \lambda ab \cdot a(a(a(b))),$$

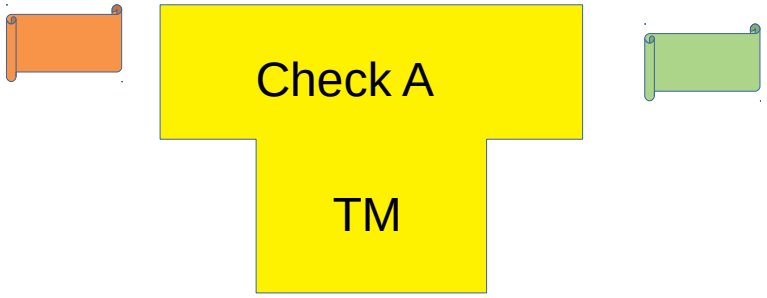




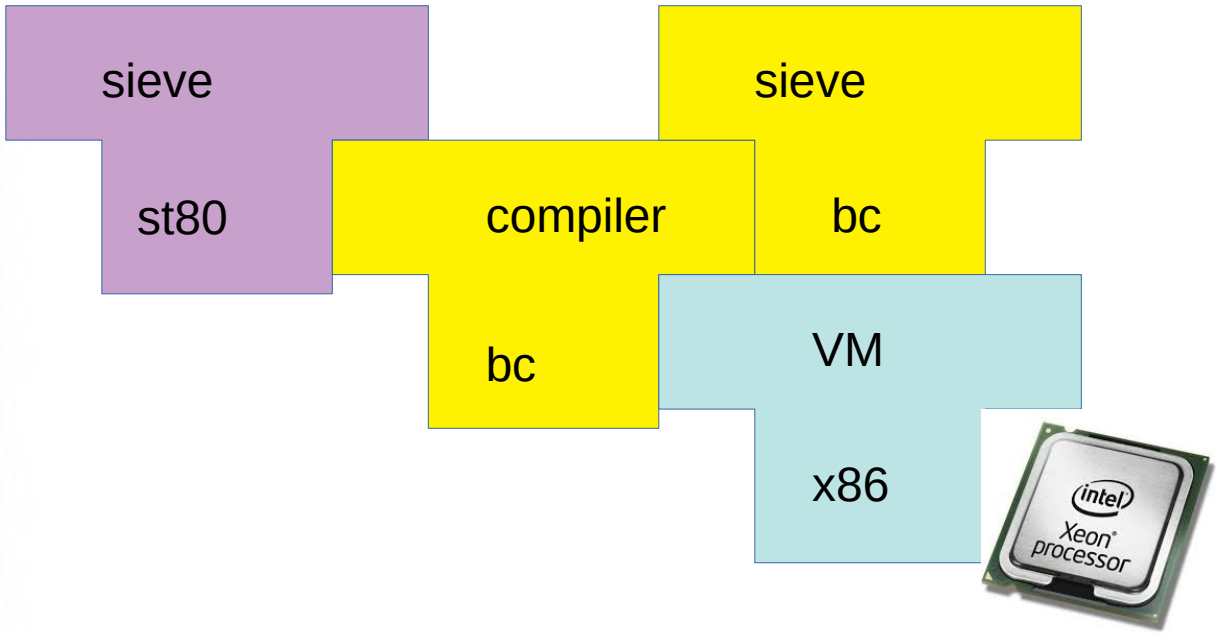
Simulation (Interpreter)

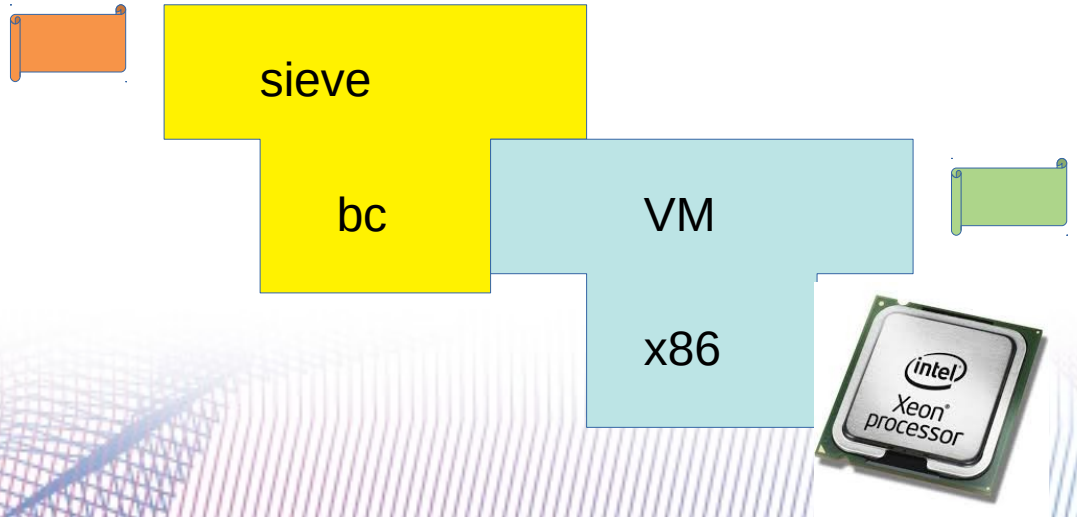
Translators (Compilers)





Hybrid


sieve
st80
compiler
bc
sieve
bc
VM
x86
intel
Xeon
processor


sieve
bc
VM
x86
intel
Xeon
processor

Constants and Performance

```
<T> v[100];
....
x = v[i];
....
```

```
;; r3 = calculate sizeOf(T)
;; r2 = calculate &v[0]
;; r4 = calculate i
mult r5,r4,r3
add r6,r2,r5
load r1,r6
```

Constants and Performance

```
double v[100];
                    ;; r2 = calculate &v[0]
                    ;; r4 = calculate i
....
x = v[i];           mult r5,r4,#8
....                add r6,r2,r5
                    load r1,r6
```

Constants and Performance

```
double v[100];
                    ;; r2 = calculate &v[0]
....
i = 3;
x = v[i];           add r6,r2,#(3*8)
....                load r1,r6
```

Constants and Performance

```
static double v[100];
```

```
....
i = 3;
x = v[i];       load r6,#(0x28f0+3*8)
....            load r1,r6
```

Constants and Performance

```
static double v[100] =
{ 1.2, 0.7, 8.9,
2.5,.....};
```

```
....
i = 3;
x = v[i];
....
```

```
load r1,#2.5
```

1984: Dynamic Compilation (JIT)

L Peter Deutsch

Allan M. Schiffman

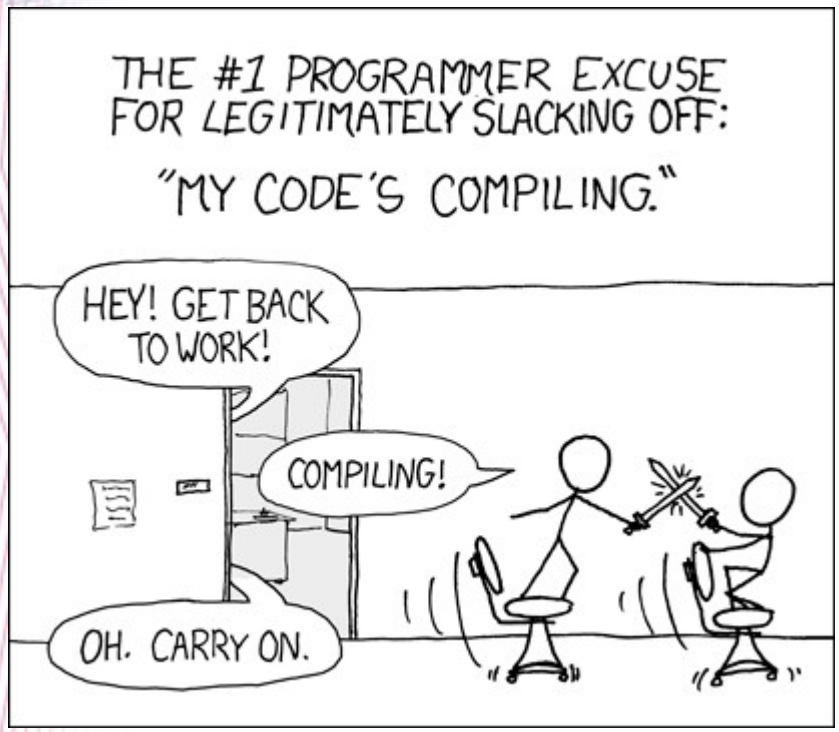
Interpreted Code

Compiled Code

Compiler

Constants x Time

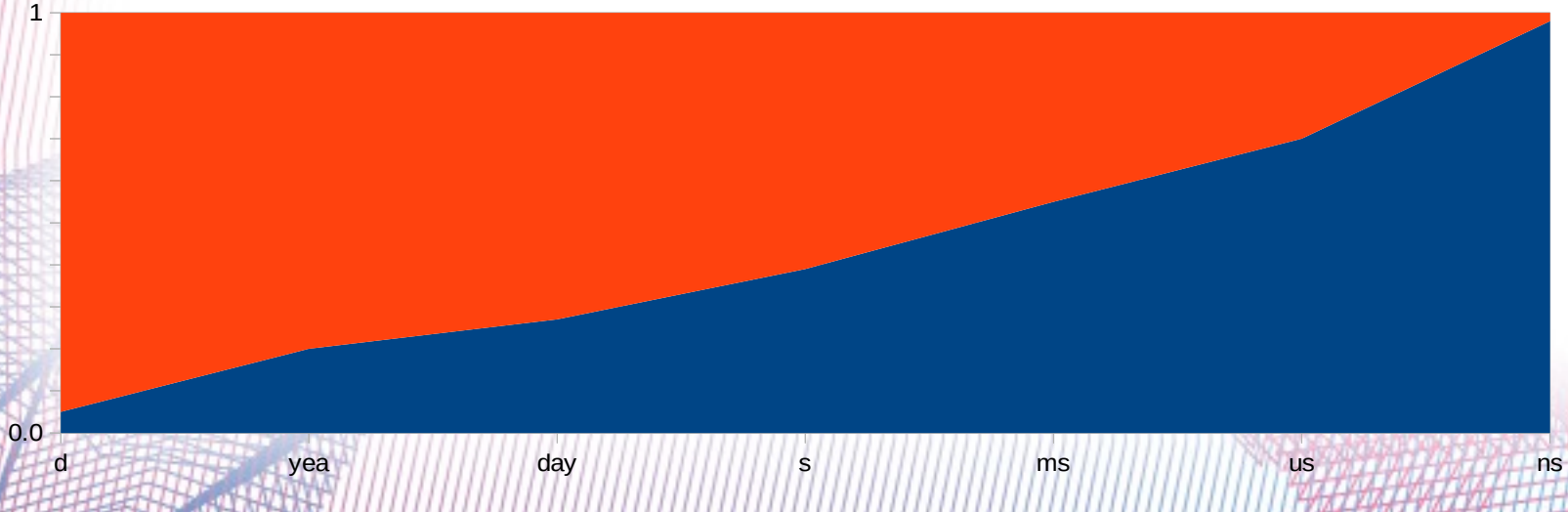
XKCD 303



Ahead of time compilation

One man's constant is another man's variable.

Alan Perlis



1990: the evolution of the Self virtual machine

Craig Chambers

Self 1.0

Self 2.0

Self 1 and 2: type inference

Craig Chambers

Polymorphism

Factored code

Generic code

Dynamic types

PICs

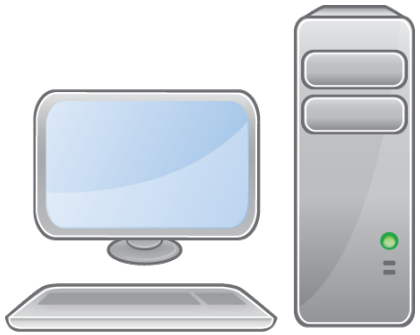
Inlining

Customization

Type inference

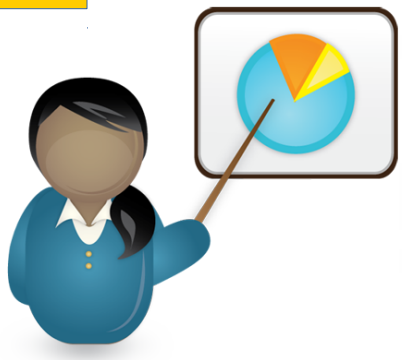
Language evolution

Compiler technology



The computer wants to run

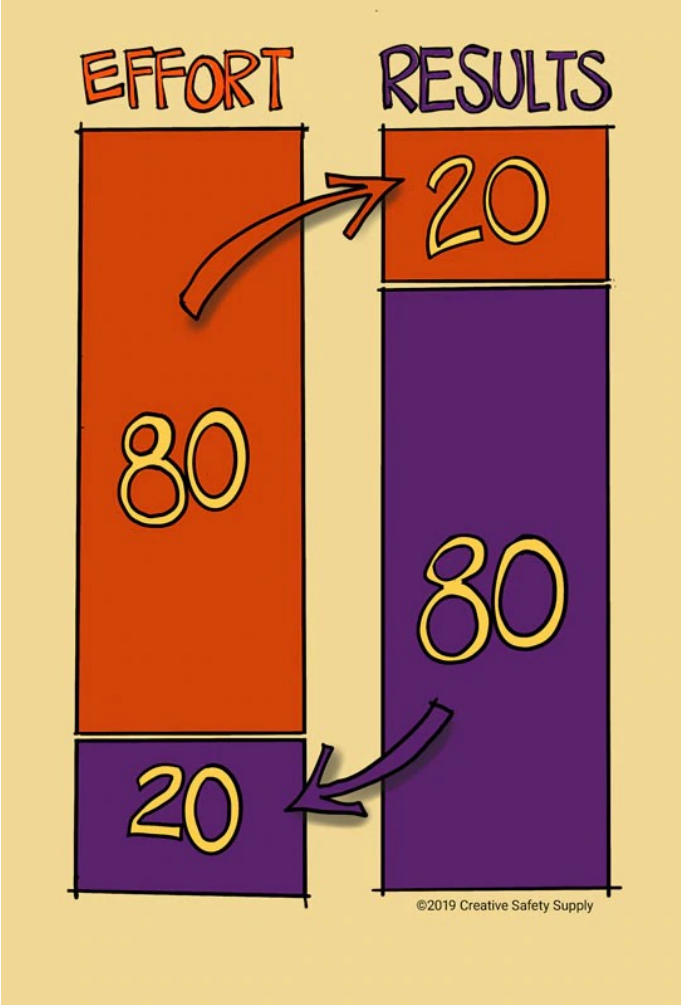
Fortran (or C)

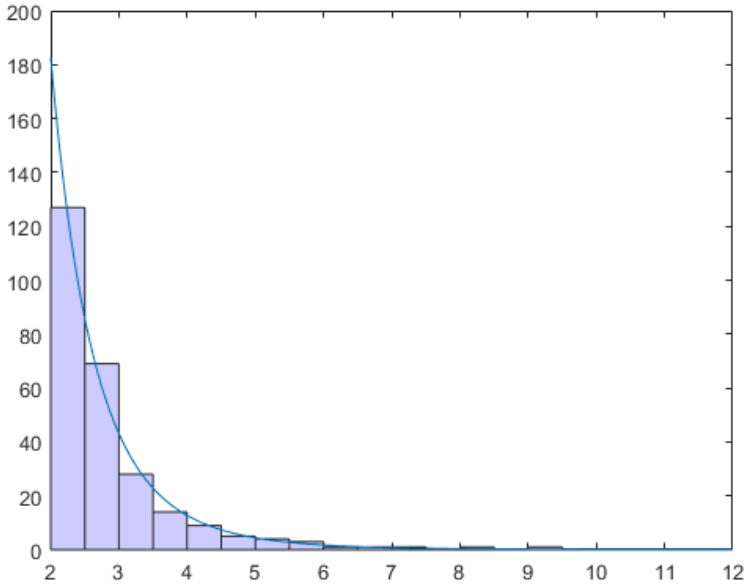


The programmer wants to write in

Self

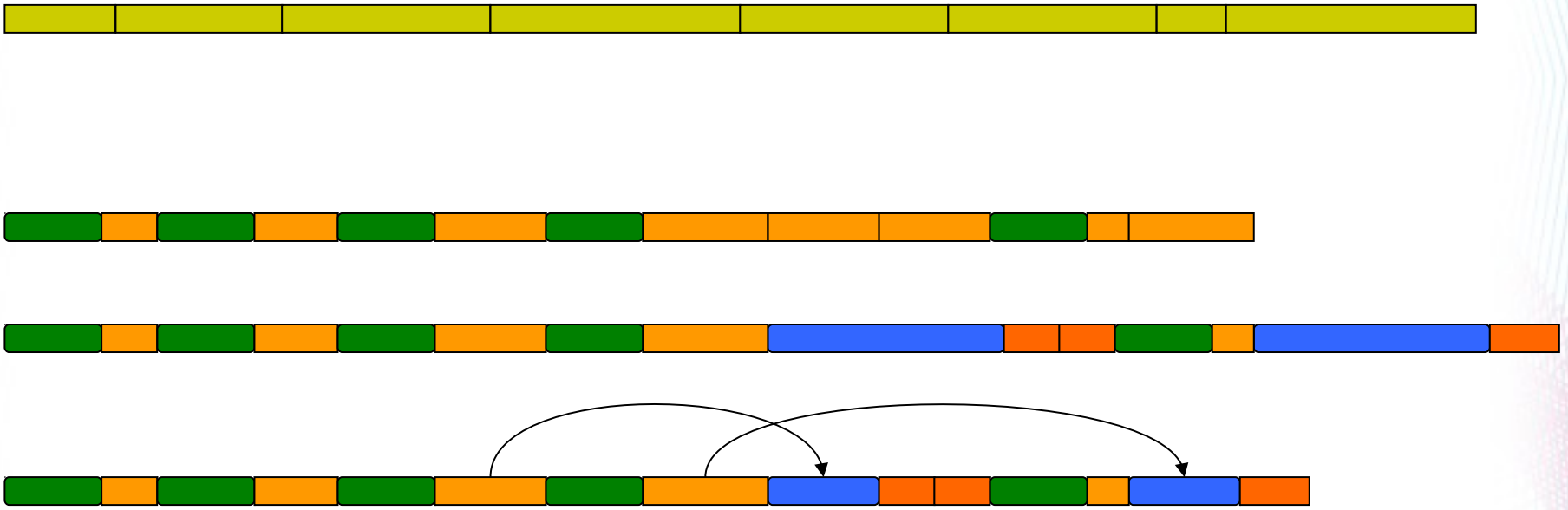
Hot Spots: Pareto distribution





1992: Adaptive Compilation

Urs Hölzle

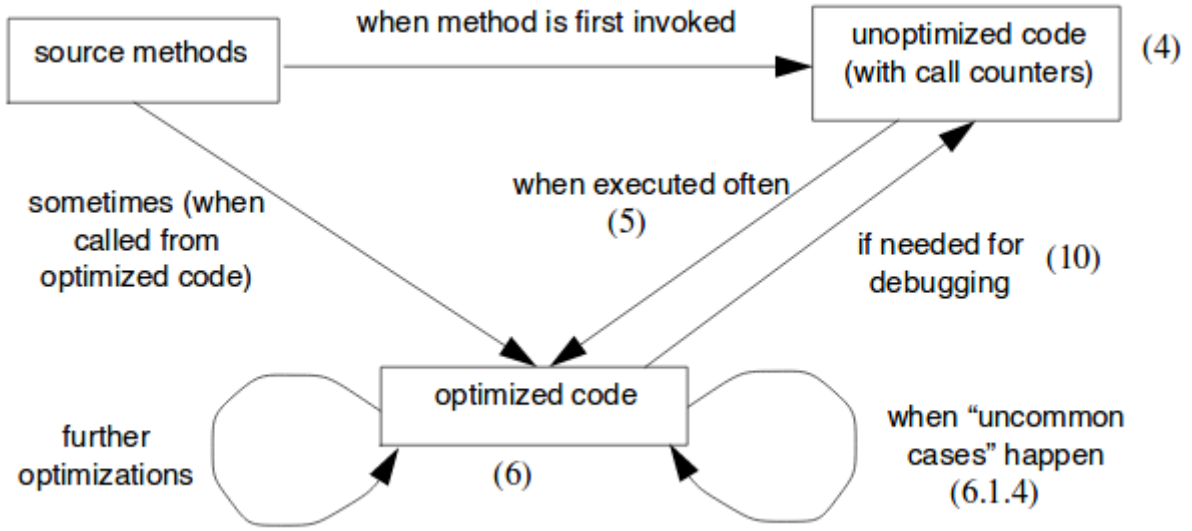


“type feedback”

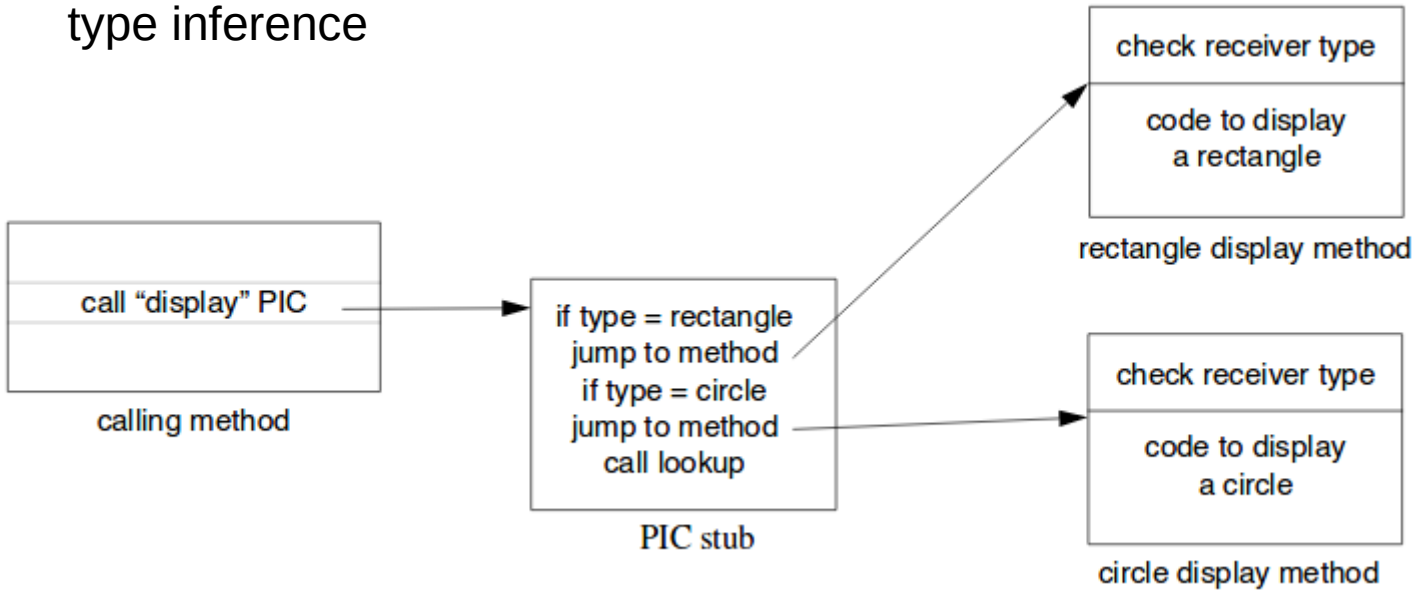
Self 3.0

Self 3 and 4: adaptive compilation

Urs Hölzle



Type feedback instead of type inference

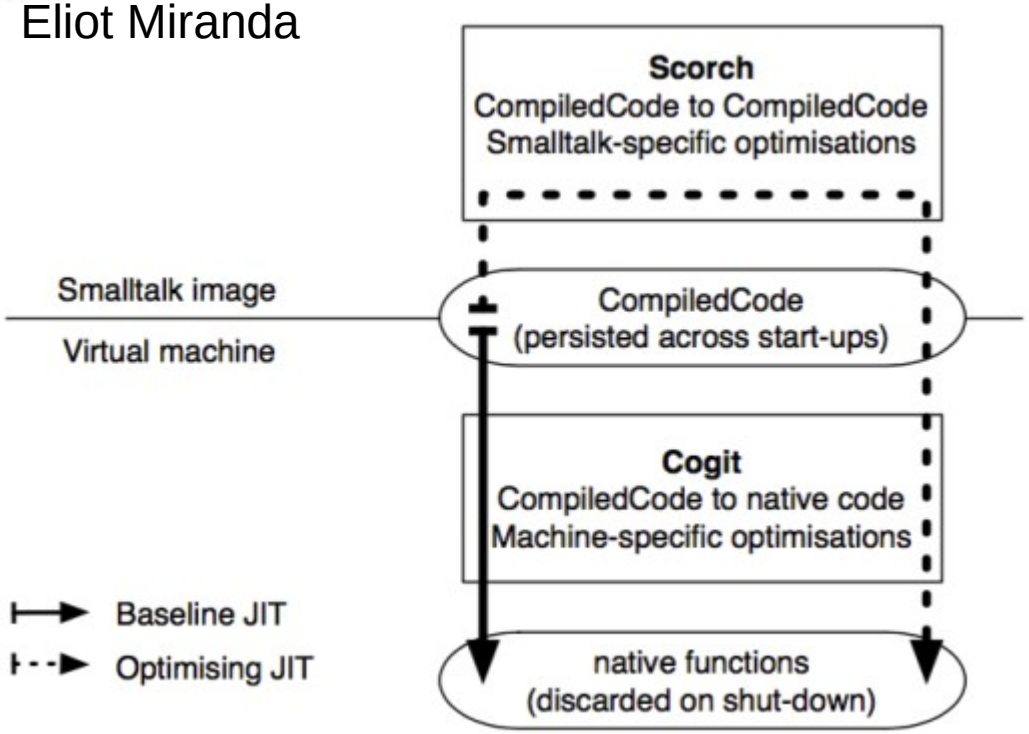


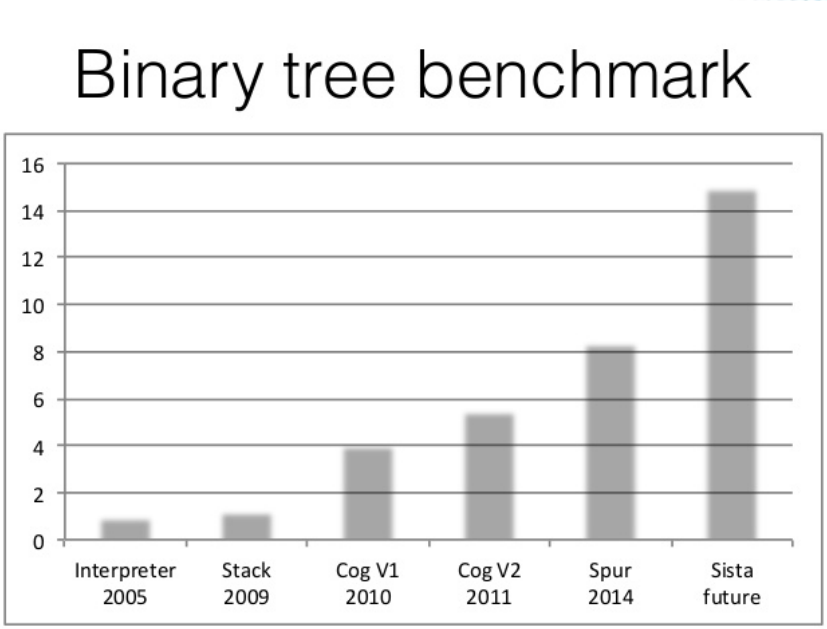
Two compilers: simple and fast generates instrumented code, good compiler for “hot spots”

Cog and Sista

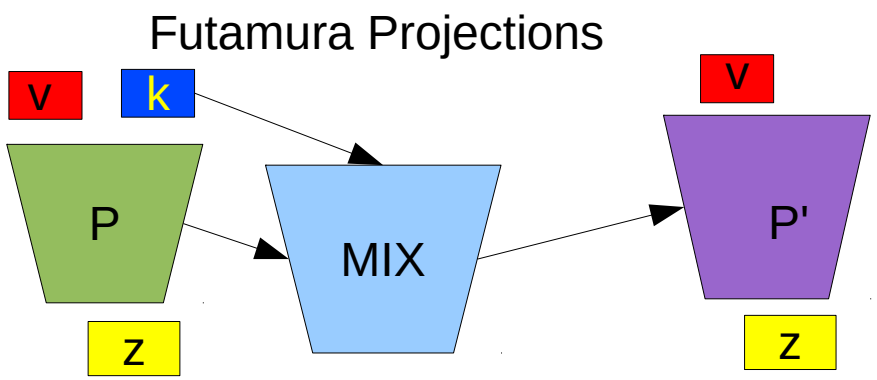
Eliot Miranda

Clément Béra





Partial Evaluation



First: compiler

P = interpreter; k = source; P' = executable; v = data; z = output;

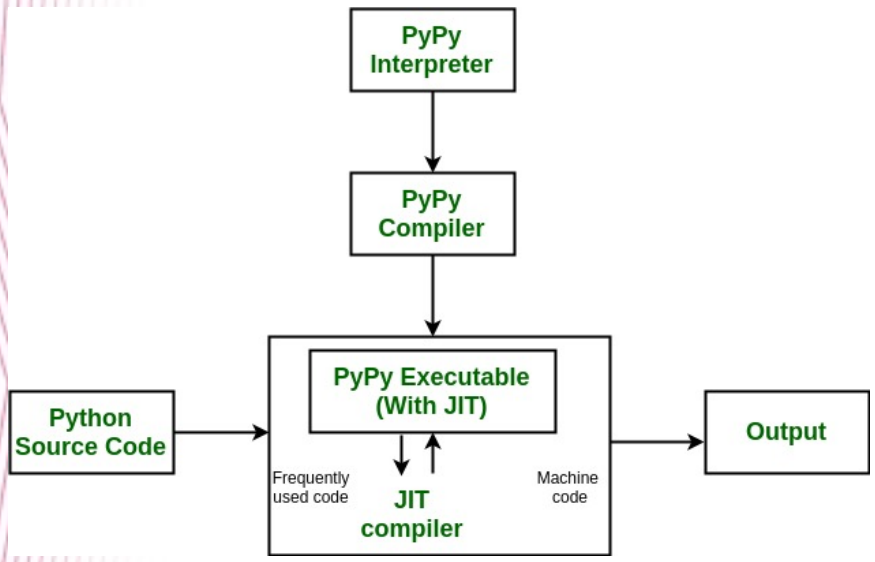
Second: compiler generator

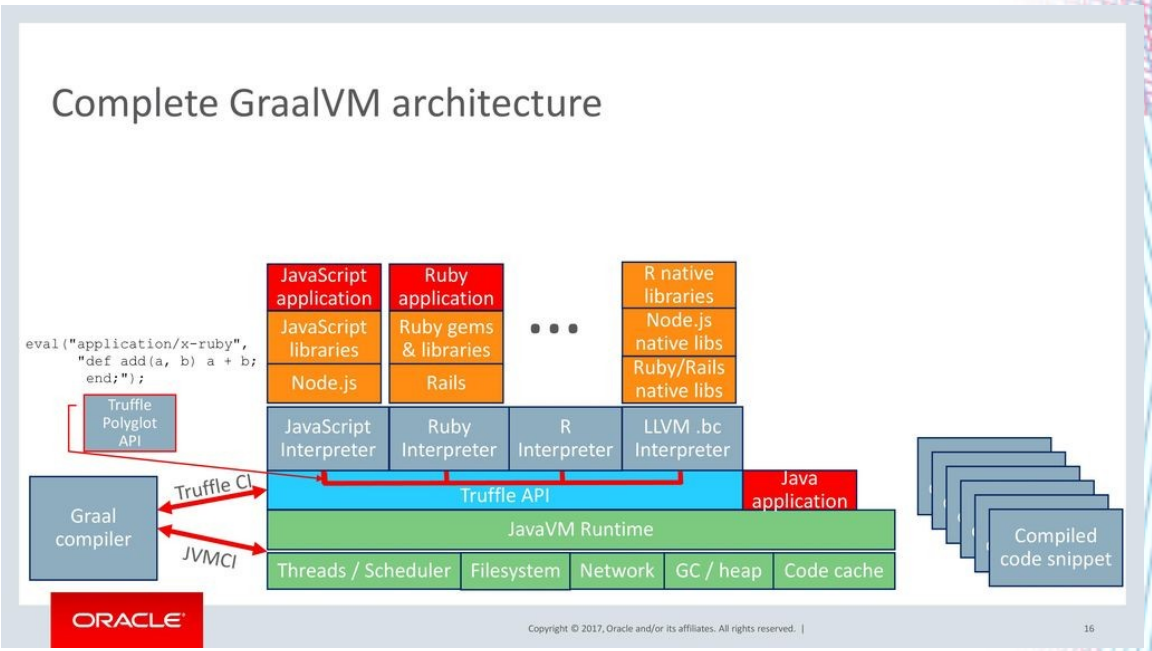
P = MIX; k = interpreter; P' = compiler; v = source; z = executable;

Third: universal compiler generator

P = MIX; k = MIX; P' = generator; v = interpreter; z = compiler

From Interpreter to Compiler





Thanks!

Questions?