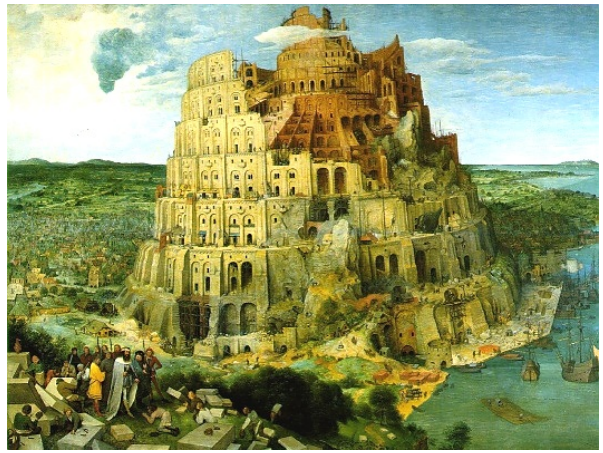
Programming Languages and Translators

Session 23, COMP 11

November 21, 2019



Pieter Bruegel, *The Tower of Babel*, 1563

What’s in a Language?

Components of a language: Syntax

How characters combine to form words, sentences, paragraphs.

The quick brown fox jumps over the lazy dog.

is syntactically correct English, but isn't a C++ program.

```
class Foo {
public:
  int foo(int k);
private:
  int j;
}
```

is syntactically correct C++, but isn't Java.

Specifying Syntax

Usually done with a context-free grammar.

Typical syntax for algebraic expressions:

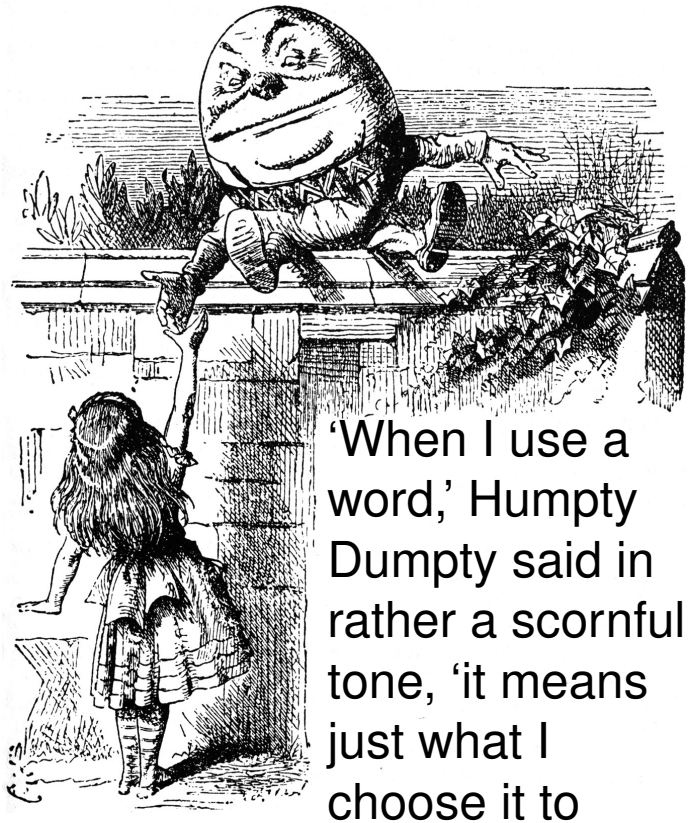
$$
\begin{array}{rcl}
\textit{expr} & \rightarrow & \textit{expr} + \textit{expr} \\
& | & \textit{expr} - \textit{expr} \\
& | & \textit{expr} * \textit{expr} \\
& | & \textit{expr} / \textit{expr} \\
& | & (\textit{expr}) \\
& | & \textbf{digits}
\end{array}
$$

Components of a language: Semantics

What a well-formed program "means."

The semantics of C++ says this computes the nth Fibonacci number.

```
int fib(int n)
{
  int a = 0, b = 1;
  int i;
  for (i = 1 ; i < n ; i++) {
    int c = a + b;
    a = b;
    b = c;
  }
  return b;
}
```



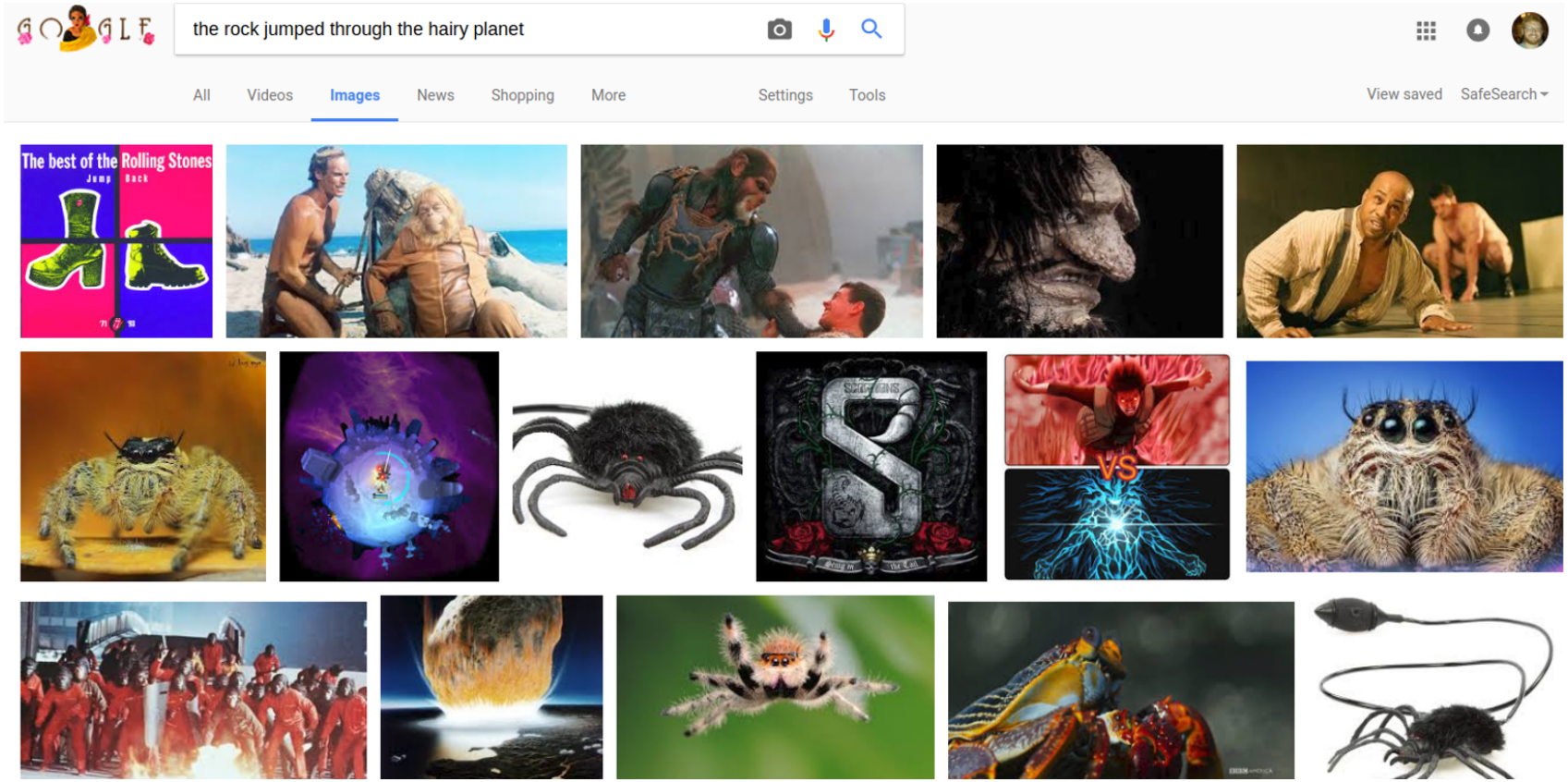
'When I use a word,' Humpty Dumpty said in rather a scornful tone, 'it means just what I choose it to mean—neither more nor less.'

Source: Lewis Carroll, *Through the Looking Glass*, 1872.

Semantics

Something may be syntactically correct but semantically nonsensical

The rock jumped through the hairy planet.



Or ambiguous

The chickens are ready to eat.

Semantics

Nonsensical in C++:

```
class Foo {
  int bar(int x) { return Foo; }
}
```

Ambiguous in C++:

```
class Bar {
public:
  float foo() { return 0; }
  int foo() { return 1; }
}
```

Specifying Semantics

A couple different approaches; one is operational semantics (formally define how a program should be evaluated).

Typically specified with inference rules:

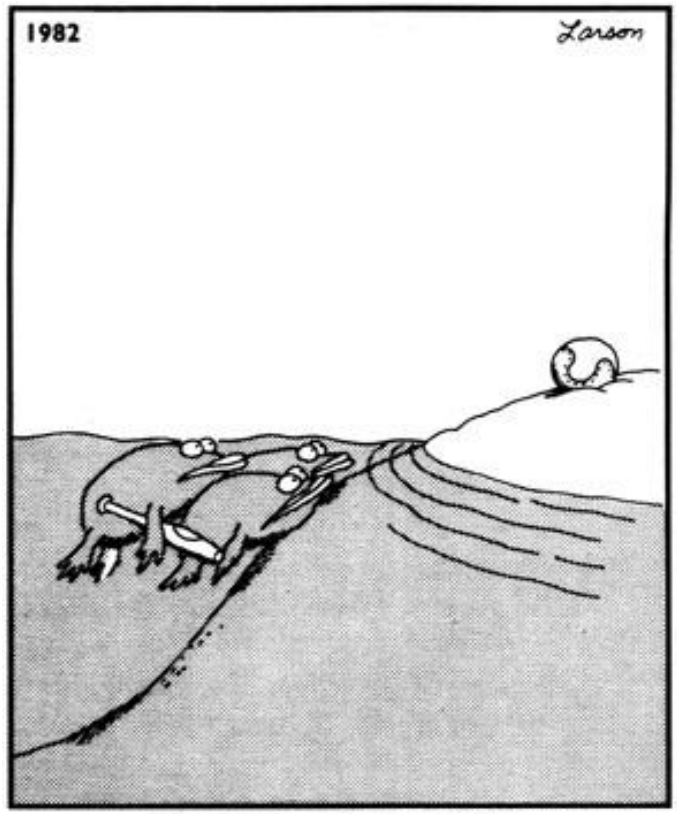
$$\frac{}{\textbf{digits} \Downarrow \textbf{digits}}$$

"A number written out has the value of that number."

$$\frac{expr_1 \Downarrow \textbf{digits}_1 \quad expr_2 \Downarrow \textbf{digits}_2 \quad \textbf{digits}_3 = \textbf{digits}_1 + \textbf{digits}_2}{expr_1 + expr_2 \Downarrow \textbf{digits}_3}$$

"$expr_1$ + $expr_2$ evaluates to **digits**$_3$ if $expr_1$ evaluates to **digits**$_1$, $expr_2$ evaluates to **digits**$_2$, and **digits**$_3$ is the sum of **digits**$_1$ and **digits**$_2$."

Great Moments in Evolution



Great moments in evolution

Assembly Language

Before: numbers

```
55
89E5
8B4508
8B550C
39D0
740D
39D0
7E08
29D0
39D0
75F6
C9
C3
29C2
EBF6
```

After: Symbols

```
gcd: pushl %ebp
     movl  %esp, %ebp
     movl  8(%ebp), %eax
     movl  12(%ebp), %edx
     cmpl  %edx, %eax
     je    .L9
.L7: cmpl  %edx, %eax
     jle   .L5
     subl  %edx, %eax
.L2: cmpl  %edx, %eax
     jne   .L7
.L9: leave
     ret
.L5: subl  %eax, %edx
```

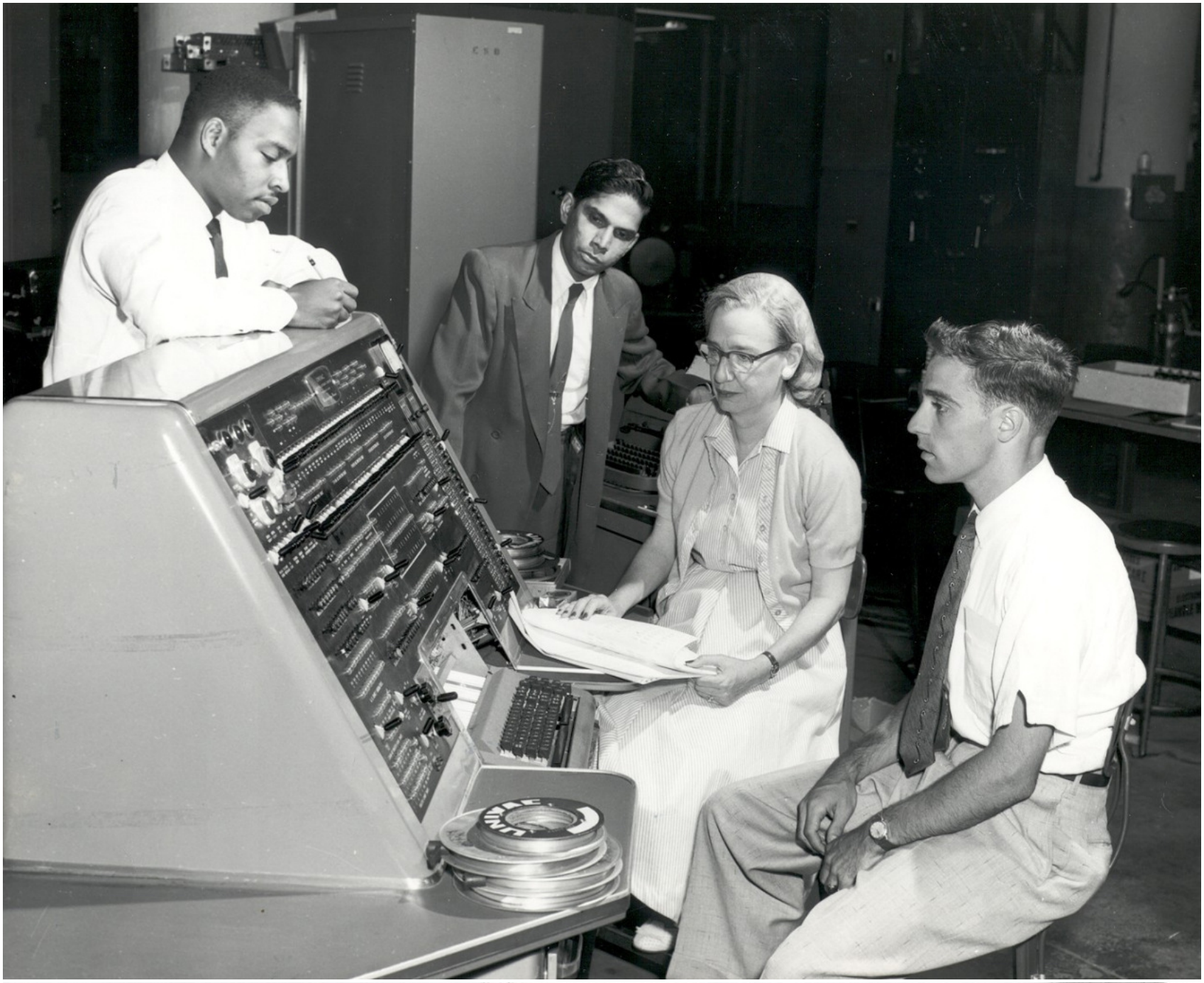
FORTRAN

Expressions, control-flow

```
10    if (a .EQ. b) goto 20
      if (a .LT. b) then
         a = a - b
      else
         b = b - a
      endif
      goto 10
20    end
```

Backus, IBM, 1957.
First compiled language.
Imperative language for science and engineering.
Arithmetic expressions, If, Do, and Goto statements.
Fixed format lines (for punch cards).
Scalar (number) and array types.
Limited string support.
Still common in high-performance computing.
Inspired most modern languages, especially BASIC.

COBOL



LISP, Scheme, Common LISP

Functional, high-level languages

```
(defun append (l1 l2)
  (if (null l1)
      l2
      (cons (first l1) (append (rest l1) l2))))
```

- McCarthy, MIT, 1958
- Functional: recursive, list-focused functions
- Simple, heavily parenthesized “S-expression” syntax
- Dynamically typed, automatic garbage collection

APL

Powerful operators, interactive, custom character set

```
[0]    Z←GAUSSRAND N;B;F;M;P;Q;R
[1]    ⍝Returns ⍵ random numbers having a Gaussian normal distribution
[2]    ⍝ (with mean 0 and variance 1)  Uses the Box-Muller method.
[3]    ⍝ See Numerical Recipes in C, pg. 289.
[4]    ⍝
[5]    Z←⍳0
[6]    M←¯1+2*31          ⍝ largest integer
[7]   L1:Q←N-⍴Z           ⍝ how many more we need
[8]    →(Q≤0)/L2          ⍝ quit if none
[9]    Q←⌈1.3×Q÷2         ⍝ approx num points needed
[10]   P←¯1+(2÷M-1)×¯1+?(Q,2)⍴M  ⍝ random points in -1 to 1 square
[11]   R←+/P×P            ⍝ distance from origin squared
[12]   B←(R≠0)∧R<1
[13]   R←B/R ⋄ P←B⌿P      ⍝ points within unit circle
[14]   F←(¯2×(⊛R)÷R)*.5
[15]   Z←Z,,P×F,[1.5]F
[16]   →L1
[17]  L2:Z←N↑Z
[18]   ⍝ ArchDate: 12/16/1997 16:20:23.170
```

“Emoticons for Mathematicians”

Source: Jim Weigang, http://www.chilton.com/~jimw/gsrand.html

At right: Datamedia APL Keyboard

99 Bottles of Beer in APL

```
⍝ APL (A Programming Language)

⍝ Program written by JT. Taylor, www.jttaylor.net

T1←98↑[1]⍬⍴1 99⍴⍳99

T4←⍬⍴1 98⍴⍳98

T1,(98 30⍴' BOTTLES OF BEER ON THE WALL, '),T1,
(98 47⍴'BOTTLES OF BEER, TAKE ONE DOWN, PASS IT
AROUND,'),T4,(98 28⍴'BOTTLES OF BEER ON THE
WALL ,')

'1 BOTTLE OF BEER ON THE WALL, 1 BOTTLE OF BEER,
TAKE IT DOWN, PASS IT AROUND, NO BOTTLES OF BEER
ON THE WALL.'
```

http://www.99-bottles-of-beer.net/language-apl-715.html

Algol, Pascal, Clu, Modula, Ada

Imperative, block-structured language, formal syntax definition, structured programming

```
PROC insert = (INT e, REF TREE t)VOID:
   # NB inserts in t as a side effect #
   IF TREE(t) IS NIL THEN
     t := HEAP NODE := (e, TREE(NIL), TREE(NIL))
   ELIF e < e OF t THEN insert(e, l OF t)
   ELIF e > e OF t THEN insert(e, r OF t)
   FI;

 PROC trav = (INT switch, TREE t, SCANNER continue,
              alternative)VOID:
   # traverse the root node and right sub-tree of t  only. #
   IF t IS NIL THEN continue(switch, alternative)
   ELIF e OF t  <=  switch THEN
         print(e OF t);
         traverse( switch, r OF t, continue, alternative)
   ELSE  # e OF t > switch #
         PROC defer = (INT sw, SCANNER alt)VOID:
               trav(sw, t, continue, alt);
         alternative(e OF t, defer)
   FI;
```

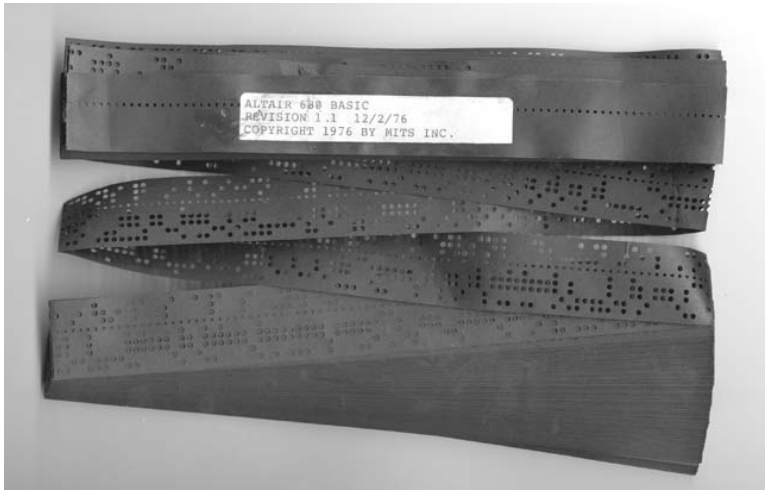
Algol-68, source http://www.csse.monash.edu.au/~lloyd/tildeProgLang/Algol68/treemerge.a68

BASIC

Programming for the masses

```
10 PRINT "GUESS A NUMBER BETWEEN ONE AND TEN"
20 INPUT A$
30 IF A$ <> "5" THEN GOTO 60
40 PRINT "GOOD JOB, YOU GUESSED IT"
50 GOTO 100
60 PRINT "YOU ARE WRONG. TRY AGAIN"
70 GOTO 10
100 END
```

John George Kemeny and Thomas Eugene Kurtz, Dartmouth, 1964. Started the whole Bill Gates/ Microsoft thing.



Simula, Smalltalk, C++, Java, C#

The object-oriented philosophy

```
class Shape(x, y); integer x; integer y;
virtual: procedure draw;
begin
   comment - get the x & y coordinates -;
   integer procedure getX;
      getX := x;
   integer procedure getY;
      getY := y;

   comment - set the x & y coordinates -;
   integer procedure setX(newx); integer newx;
      x := newx;
   integer procedure setY(newy); integer newy;
      y := newy;
end Shape;
```

- Simula: 1965, introduced objects, classes, inheritance
- Smalltalk: 1972, everything is an object, message-passing, reflection

C

Efficiency for systems programming

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

- Dennis Ritchie, Bell labs, 1969
- Harmonizes with processor architecture
- Statically typed (but liberal conversion policies)
- Unsafe by design (pointers?)
- Language of choice for OS development

ML, Caml, Haskell

Functional languages with types and syntax

```
structure RevStack = struct
  type  'a stack =  'a list
  exception Empty
  val empty = []
  fun isEmpty (s:'a stack):bool =
    (case s
       of [] => true
        | _ => false)
  fun top (s:'a stack): =
    (case s
       of [] => raise Empty
        | x::xs => x)
  fun pop (s:'a stack):'a stack =
    (case s
        of [] => raise Empty
         | x::xs => xs)
  fun push (s:'a stack,x: 'a):'a stack = x::s
  fun rev (s:'a stack):'a stack = rev (s)
end
```

sh, awk, perl, tcl, python, php

Scripting languages: glue for binding the universe together

```
class() {
  classname=‘echo "$1" | sed -n '1 s/ *:.*$//p'‘
  parent=‘echo "$1" | sed -n '1 s/^.*: *//p'‘
  hppbody=‘echo "$1" | sed -n '2,$p'‘

  forwarddefs="$forwarddefs
  class $classname;"

  if (echo $hppbody | grep -q "$classname()"); then
    defaultconstructor=
  else
    defaultconstructor="$classname() {}"
  fi
}
```

Why PHP is terrible

AWK (99 bottles of beer, bottled version)

- Aho, Weinberger, and Kernighan, Bell Labs, 1977
- Interpreted domain-specific scripting langauge for text processing.
- Pattern-action statements matched against input lines

Wilhelm Weske,
http://www.99-bottles-of-beer.net/language-awk-1910.html

```
BEGIN{
split( \
"no mo"\
"rexxN"\
"o mor"\
"exsxx"\
"Take "\
"one dow"\
"n and pas"\
"s it around"\
", xGo to the "\
"store and buy s"\
"ome more, x bot"\
"tlex of beerx o"\
"n the wall" , s,\
"x"); for( i=99 ;\
i>=0; i--){ s[0]=\
s[2] = i ; print \
s[2 + !(i) ] s[8]\
s[4+ !(i-1)] s[9]\
s[10]", " s[!(i)]\
s[8] s[4+ !(i-1)]\
s[9]".";i?s[0]--:\
s[0] = 99; print \
s[6+!i]s[!(s[0])]\
s[8] s[4 +!(i-2)]\
s[9]s[10] ".\n";}}
```

Prolog

Logic Language

```
witch(X)  <= burns(X) and female(X).
burns(X)  <= wooden(X).
wooden(X) <= floats(X).
floats(X) <= sameweight(duck, X).

female(girl).            {by observation}
sameweight(duck,girl). {by experiment }

? witch(girl).
```

- Alain Colmerauer et al., 1972
- Programs are relations: facts and rules
- Program execution consists of trying to satisfy queries
- Designed for natural language processing, expert systems, and theorem proving

The Whitespace Language

Edwin Brady and Chris Morris, April 1st, 2003
Imperative, stack-based language
Space, Tab, and Line Feed characters only
Number literals in binary: Space=0, Tab=1, LF=end
Less-than-programmer-friendly syntax; reduces toner consumption

Andrew Kemp,
https://en.wikipedia.org/wiki/Whitespace_(programming_language)

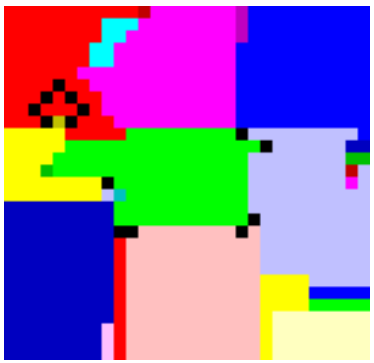
Other Esoteric Languages

```
HAI 1.0
CAN HAS STDIO?
I HAS A VAR
IM IN YR LOOP
    UP VAR!!1
    VISIBLE VAR
    IZ VAR BIGGER THAN 10? KTHX
IM OUTTA YR LOOP
KTHXBYE
```

LOLCode

```
chicken chicken chicken chicken chicken chicken chicken chicken chicken
chicken chicken
chicken chicken chicken chicken chicken chicken
```

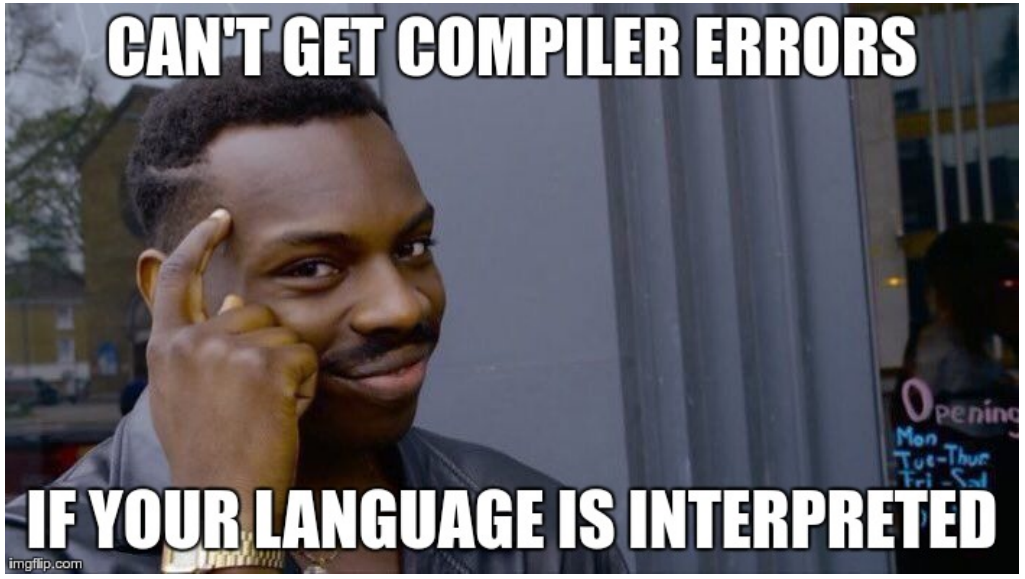
Chicken



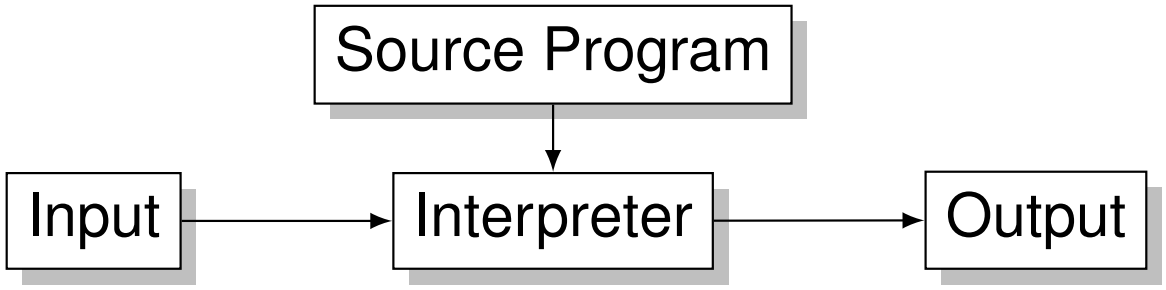
Piet

www.dangermouse.net/esoteric/piet/samples.html

Language Processors



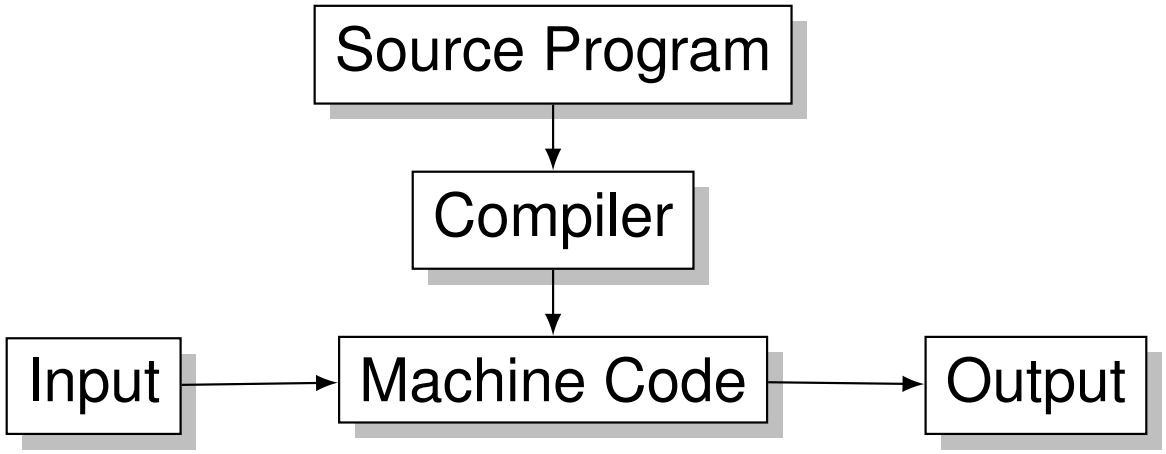
Interpreter



- Executes the program (like a VM)
- Can include robust debugger
- Good for programs that can generate their own code

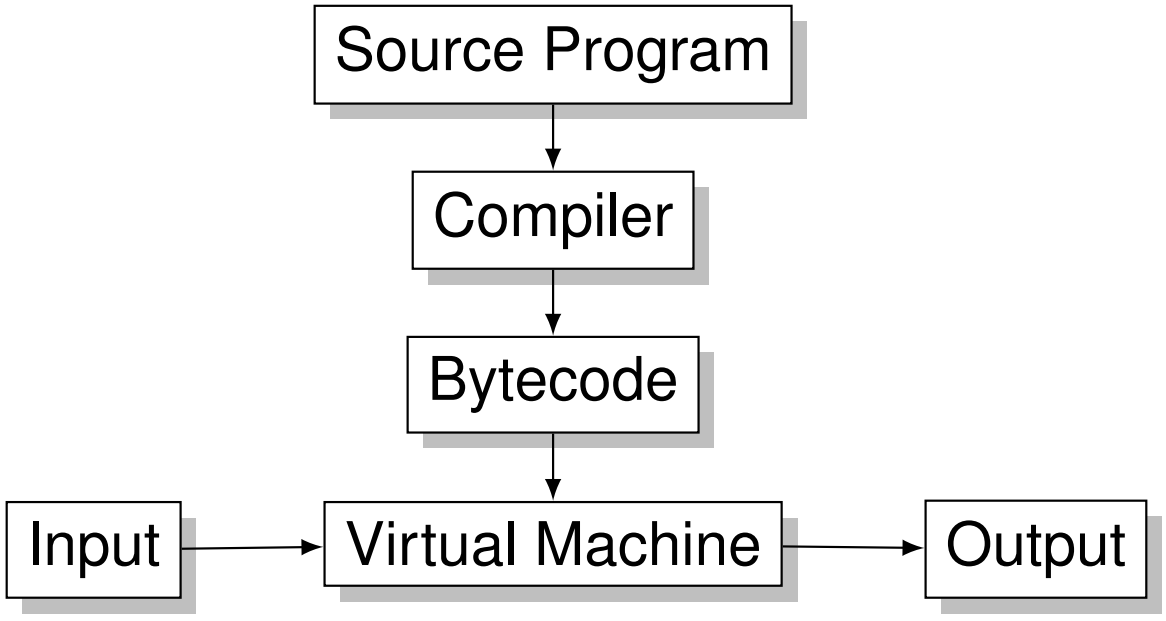
```
(defmacro eight () '(+ 3 5))
(eight) ; invoking this returns (+ 3 5),
        ; which is then evaluated
```

Compiler



- Translate high-level source to lower-level target
- Better performance than interpreting
- Two major characteristics: thorough analysis and nontrivial transformation

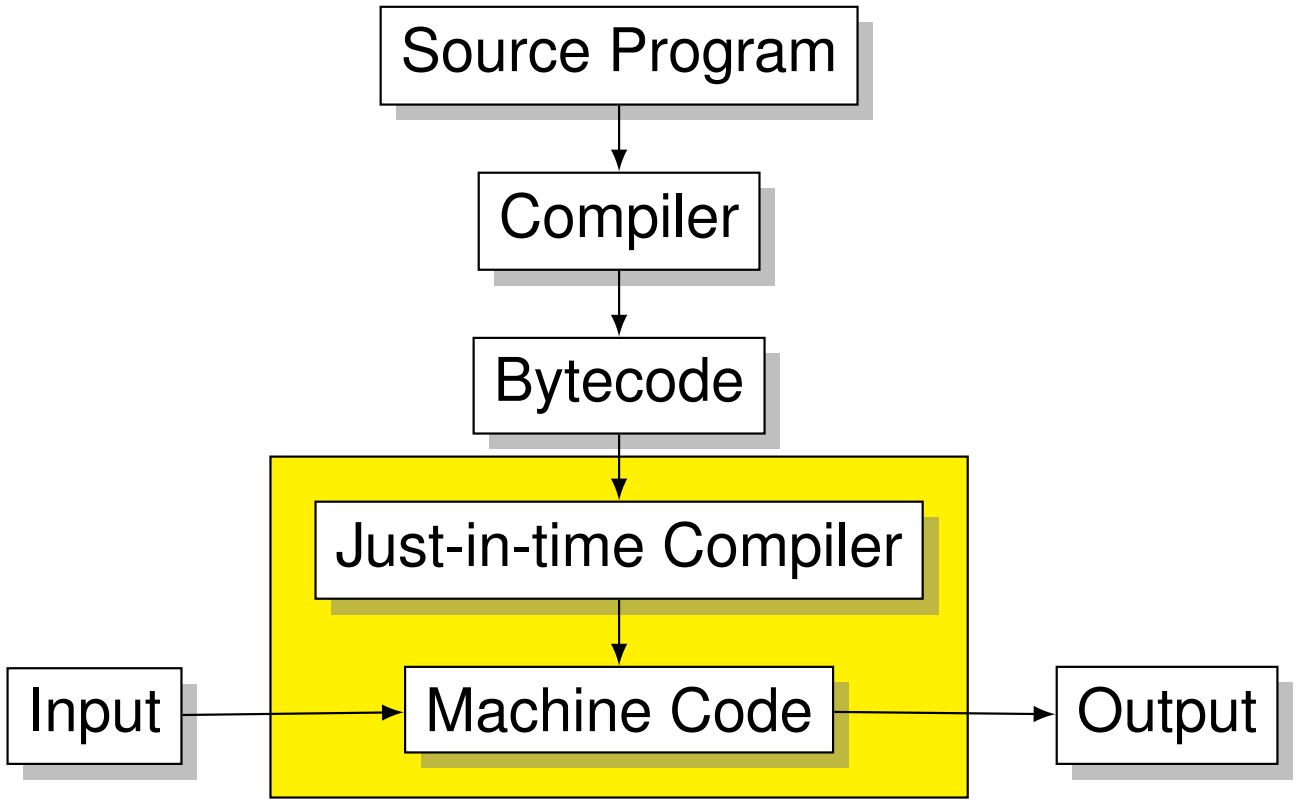
Bytecode Interpreter



Why interpret compiled bytecode (instead of interpreting source or compiling to machine code)?

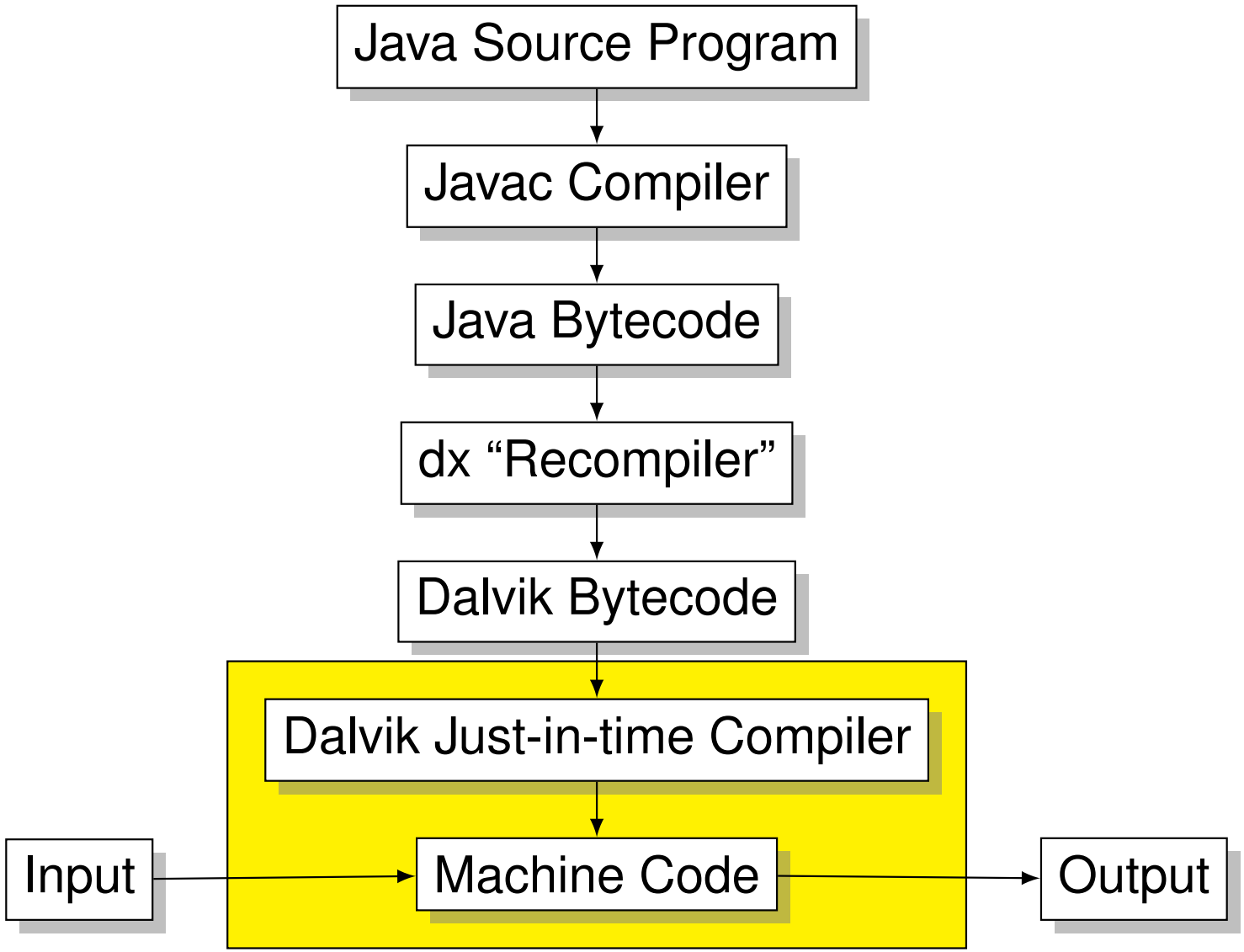
- Bytecode more portable than machine code
- Bytecode interpreter does less work than source-code interpreter

Just-In-Time Compiler

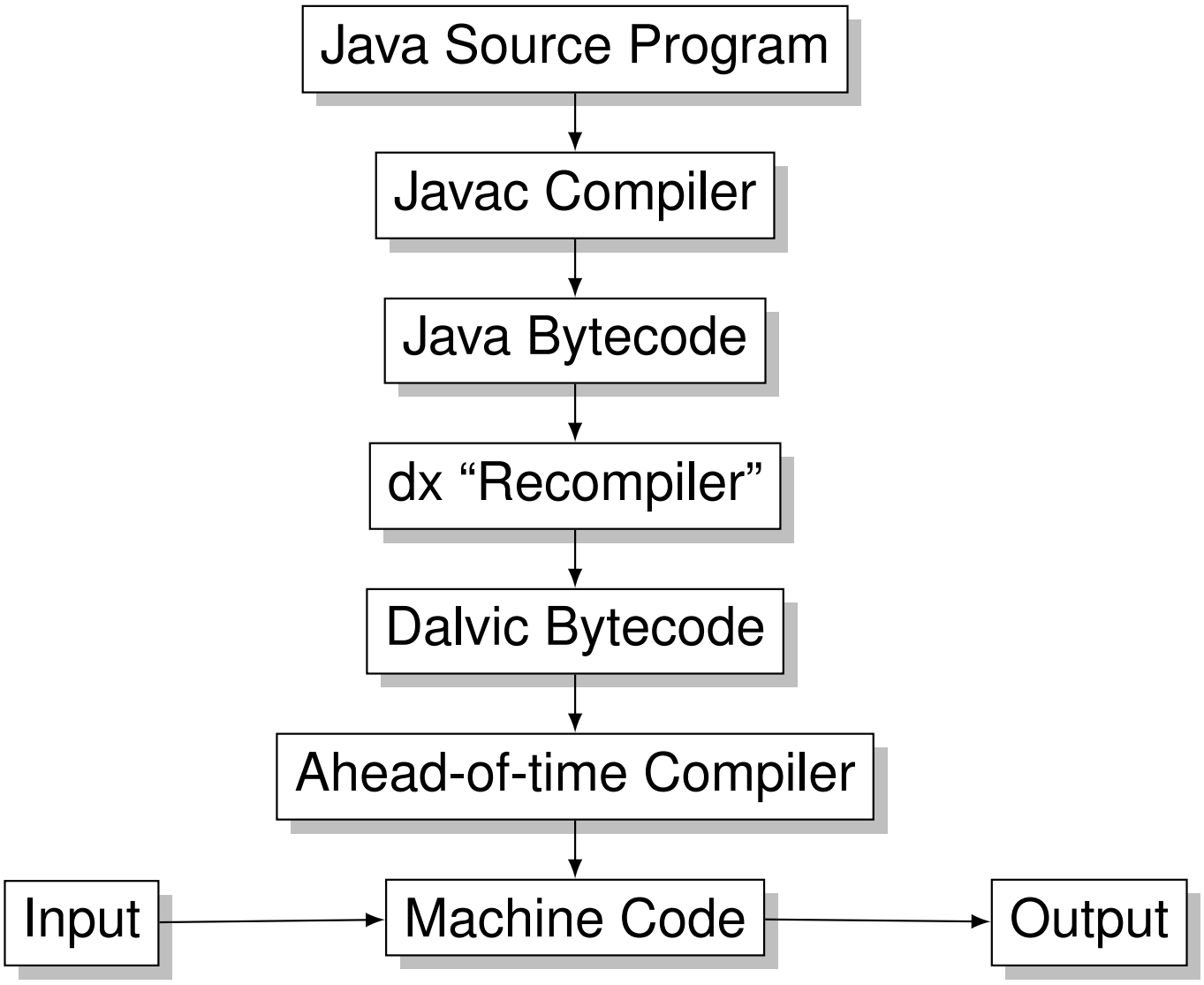


- Compile bytecode to machine code as needed
- Monitor and optimize code during runtime

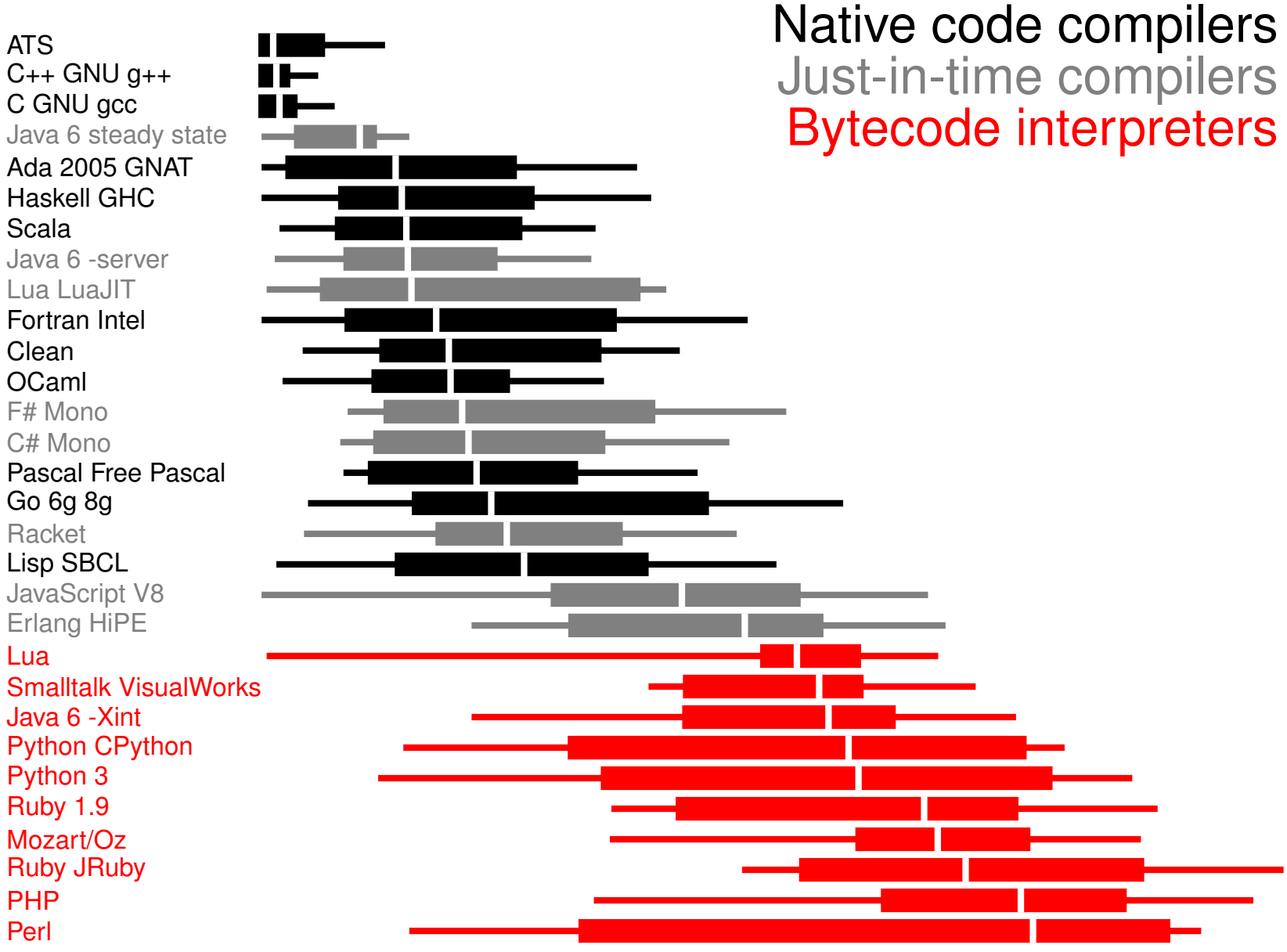
Android 4.4 KitKat and earlier



Android 5.0 Lollipop



Language Speeds Compared



Source: http://shootout.alioth.debian.org/

Compiling a Simple Program

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

What the Compiler Sees

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

```
i  n  t sp  g  c  d  (  i  n  t sp  a  , sp  i
n  t sp  b  ) nl  { nl sp sp  w  h  i  l  e sp
(  a sp  !  = sp  b  ) sp  { nl sp sp sp sp  i
f sp  (  a sp  > sp  b  ) sp  a sp  -  = sp  b
; nl sp sp sp sp  e  l  s  e sp  b sp  -  = sp
a  ; nl sp sp  } nl sp sp  r  e  t  u  r  n sp
a  ; nl  } nl
```

Just a sequence of characters

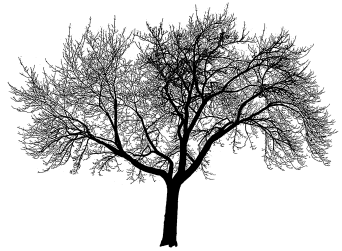
Lexical Analysis (Scanning) Gives Tokens

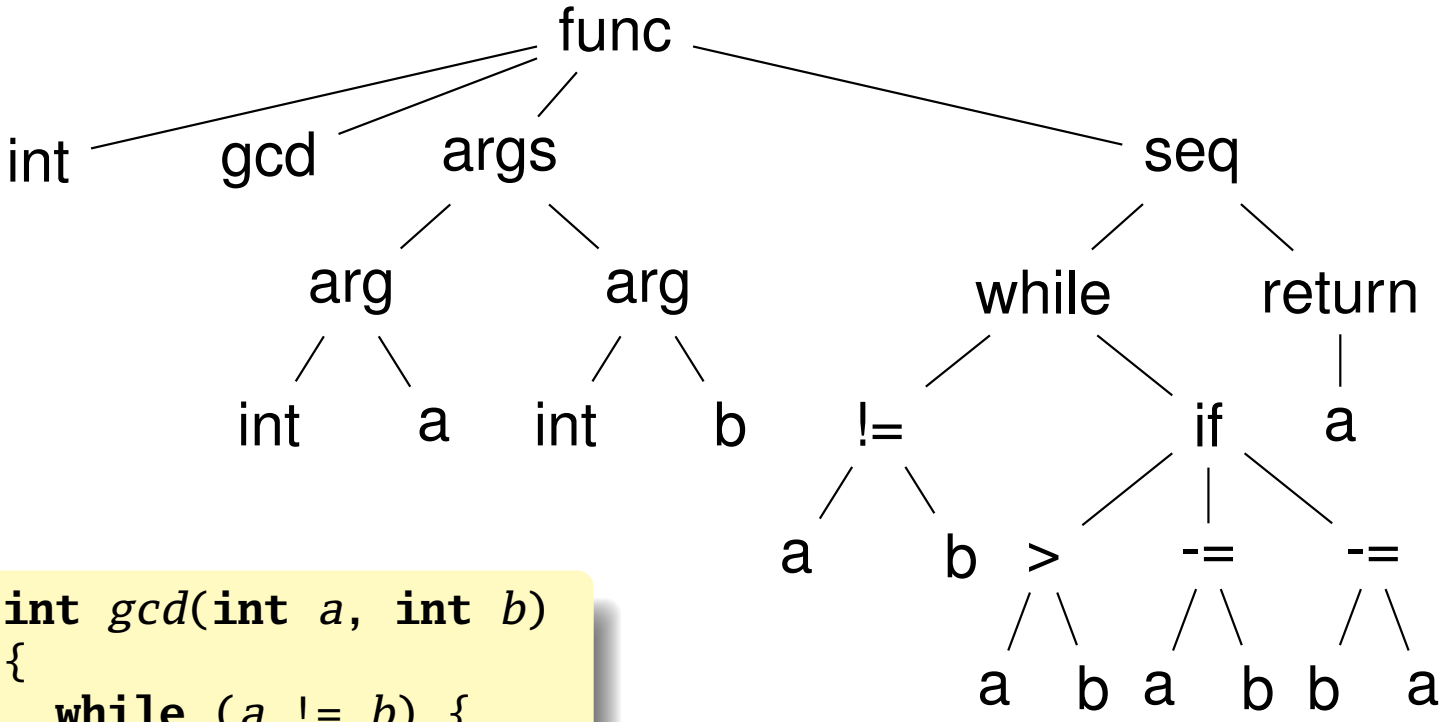
```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

```
int gcd ( int a , int b ) { while ( a
!= b ) { if ( a > b ) a -= b ; else
b -= a ; } return a ; }
```

A stream of tokens. Whitespace, comments removed.

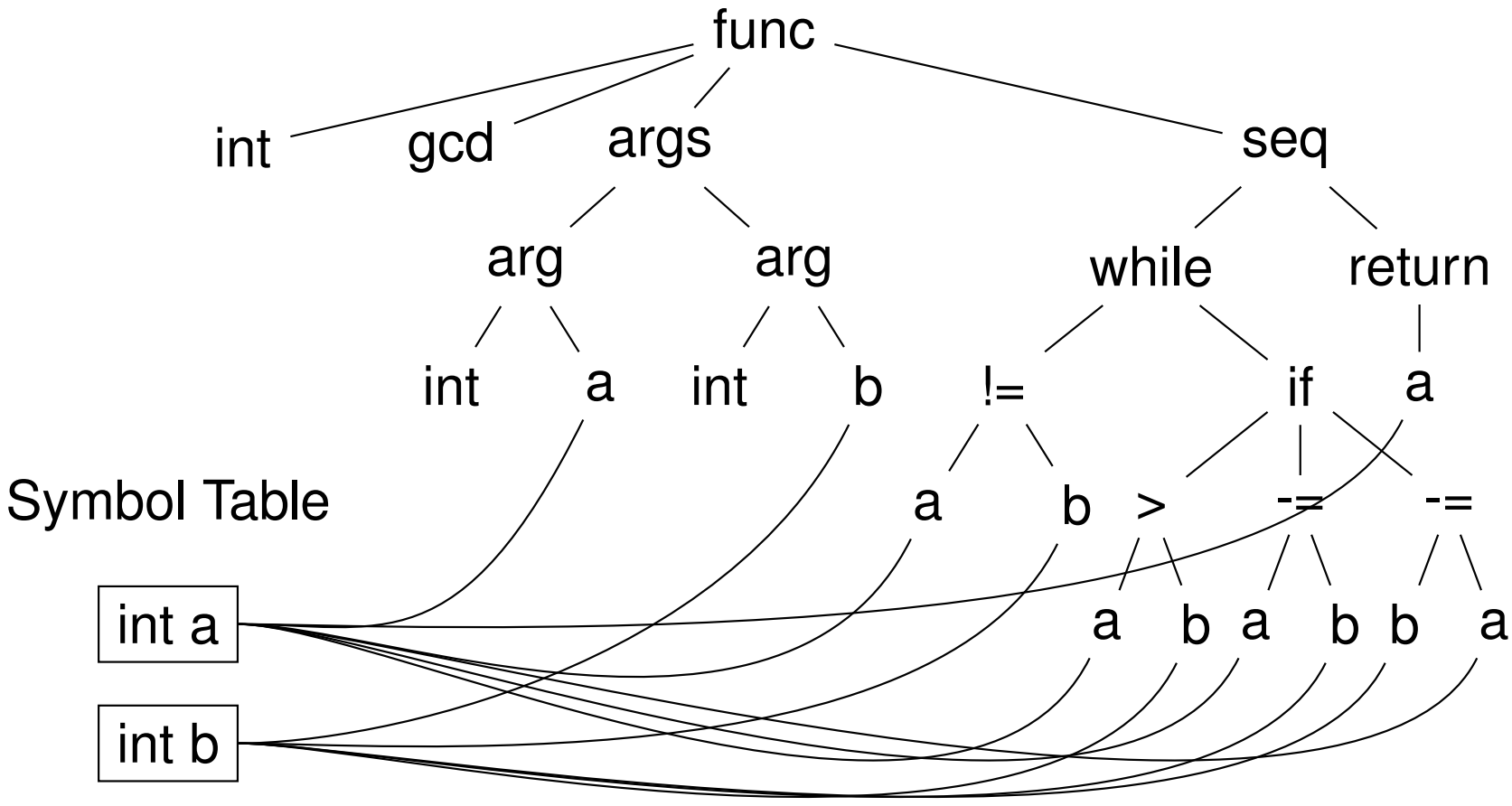
Parsing Gives an Abstract Syntax Tree






```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

Semantic Analysis: Resolve Symbols; Verify Types



- Sometimes outputs a semantically-checked abstract syntax tree (SAST) for next pass

Translation into Intermediate Representation (IR)

```
L0: $0 = a == b
    if $0 goto L1    # while (a != b)
    $1 = b > a
    if $1 goto L4    # if (a < b)
    a = a - b # a -= b
    goto L5
L4: b = b - a # b -= a
L5: goto L0
L1: ret a
```

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

3-address code:
Idealized assembly language w/ infinite registers

Generation of 80386 Assembly

```
gcd:   pushl %ebp              # Save BP
       movl  %esp,%ebp
       movl  8(%ebp),%eax      # Load a from stack
       movl  12(%ebp),%edx     # Load b from stack
.L8:   cmpl  %edx,%eax
       je    .L3               # while (a != b)
       jle   .L5               # if (a < b)
       subl  %edx,%eax         # a -= b
       jmp   .L8
.L5:   subl  %eax,%edx         # b -= a
       jmp   .L8
.L3:   leave                   # Restore SP, BP
       ret
```

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```