Programming

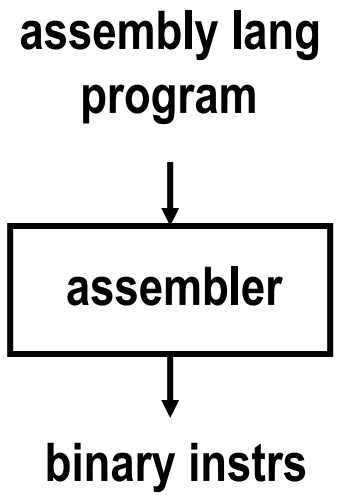
- **it's hard to do the programming to get something done**
- **details are hard to get right, very complicated, finicky**
- **not enough skilled people to do what is needed**
- **therefore, enlist machines to do some of the work**
  - leads to programming languages

- **it's hard to manage the resources of the computer**
- **hard to control sequences of operations**
- **in ancient times, high cost of having machine be idle**
- **therefore, enlist machines to do some of the work**
  - leads to operating systems

Evolution of programming languages

- **1940's: machine level**
  - use binary or equivalent notations for actual numeric values
- **1950's: "assembly language"**
  - names for instructions: ADD instead of 0110101, etc.
  - names for locations: assembler keeps track of where things are in memory; translates this more humane language into machine language
  - this is the level used in the "toy" machine
  - needs total rewrite if moved to a different kind of CPU

```
loop   get            # read a number
       ifzero  done   # no more input if number is zero
       add     sum    # add in accumulated sum
       store   sum    # store new value back in sum
       goto    loop   # read another number
done   load    sum    # print sum
       print
       stop
sum    0    # sum will be 0 when program starts
```



Evolution of programming languages, 1960's

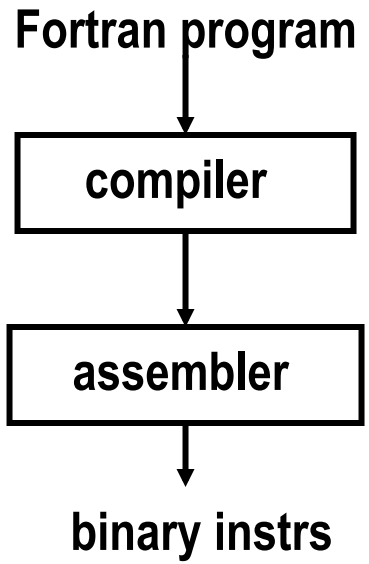
- **"high level" languages: Fortran, Cobol, Basic**
  - write in a more natural notation, e.g., mathematical formulas
  - a program ("compiler", "translator") converts into assembler
  - potential disadvantage: lower efficiency in use of machine
  - enormous advantages:

    accessible to much wider population of users

    portable: same program can be translated for different machines

    more efficient in programmer time

```
      sum = 0
10    read(5,*) num
      if (num .eq. 0) goto 20
      sum = sum + num
      goto 10
20    write(6,*) sum
      stop
      end
```

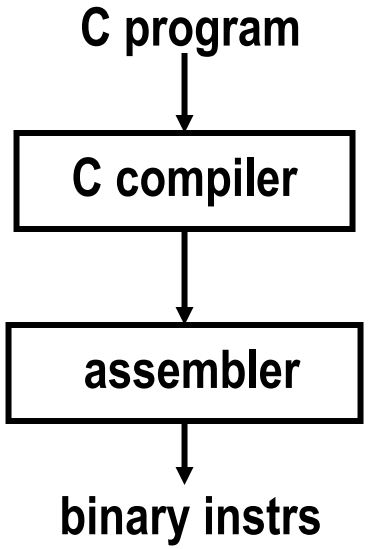


Evolution of programming languages, 1970's

- **"system programming" languages: C**
  - efficient and expressive enough to take on **any** programming task
    writing assemblers, compilers, operating systems
  - a program ("compiler", "translator") converts into assembler
  - enormous advantages:
    accessible to much wider population of programmers
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```
#include <stdio.h>
main() {
  int num, sum = 0;

  while (scanf("%d", &num) != -1 && num != 0)
       sum += num;
  printf("%d\n", sum);
}
```



C code compiled to assembly language (x86, Mac)

```
#include <stdio.h>
main() {
  int num, sum = 0;

  while (scanf("%d", &num) != -1
    && num != 0)
      sum = sum + num;
  printf("%d\n", sum);
}
```

(You are not expected to understand this!)

```
Ltmp2:
   movl $0, -8(%rbp)
   movl $0, -12(%rbp)
   jmp  LBB1_2
LBB1_1:
   movl -12(%rbp), %eax
   movl -8(%rbp), %ecx
   addl %eax, %ecx
   movl %ecx, -8(%rbp)
LBB1_2:
   leaq -12(%rbp), %rax
   xorb %cl, %cl
   leaq L_.str(%rip), %rdx
   movq %rdx, %rdi
   movq %rax, %rsi
   movb %cl, %al
   callq       _scanf
   movl %eax, %ecx
   cmpl $-1, %ecx
   je   LBB1_4
   movl -12(%rbp), %eax
   cmpl $0, %eax
   jne  LBB1_1
LBB1_4:
```

C code compiled to assembly language (ARM64)

```
#include <stdio.h>
main() {
  int num, sum = 0;

  while (scanf("%d", &num) != -1
    && num != 0)
      sum = sum + num;
  printf("%d\n", sum);
}
```

(You are not expected to understand this!)

```
   stp  x29, x30, [sp, -32]!
   add  x29, sp, 0
   str  wzr, [x29, 28]
   str  wzr, [x29, 24]
   b    .L2
.L4:
   ldr  w0, [x29, 24]
   ldr  w1, [x29, 28]
   add  w0, w1, w0
   str  w0, [x29, 28]
.L2:
   add  x1, x29, 24
   adrp x0, .LC0
   add  x0, x0, :lo12:.LC0
   bl   __isoc99_scanf
   cmn  w0, #1
   beq  .L3
   ldr  w0, [x29, 24]
   cmp  w0, 0
   bne  .L4
.L3:
   adrp x0, .LC1
   add  x0, x0, :lo12:.LC1
   ldr  w1, [x29, 28]
   bl   printf
   mov  w0, 0
   ldp  x29, x30, [sp], 32
   ret
```

Evolution of programming languages, 1980's

- **"object-oriented" languages: C++**
  - better control of structure of really large programs
    better internal checks, organization, safety
  - a program ("compiler", "translator") converts into assembler or C
  - enormous advantages:
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```
#include <iostream>
main() {
   int num, sum = 0;

   while (cin >> num && num != 0)
        sum += num;
   cout << sum << endl;
}
```

Evolution of programming languages, 1990's

- **"scripting", Web, component-based, ...:**

  Java, Perl, Python, Ruby, Visual Basic, Javascript, ...

  – write big programs by combining components already written

  – often based on "virtual machine": simulated, like fancier toy computer

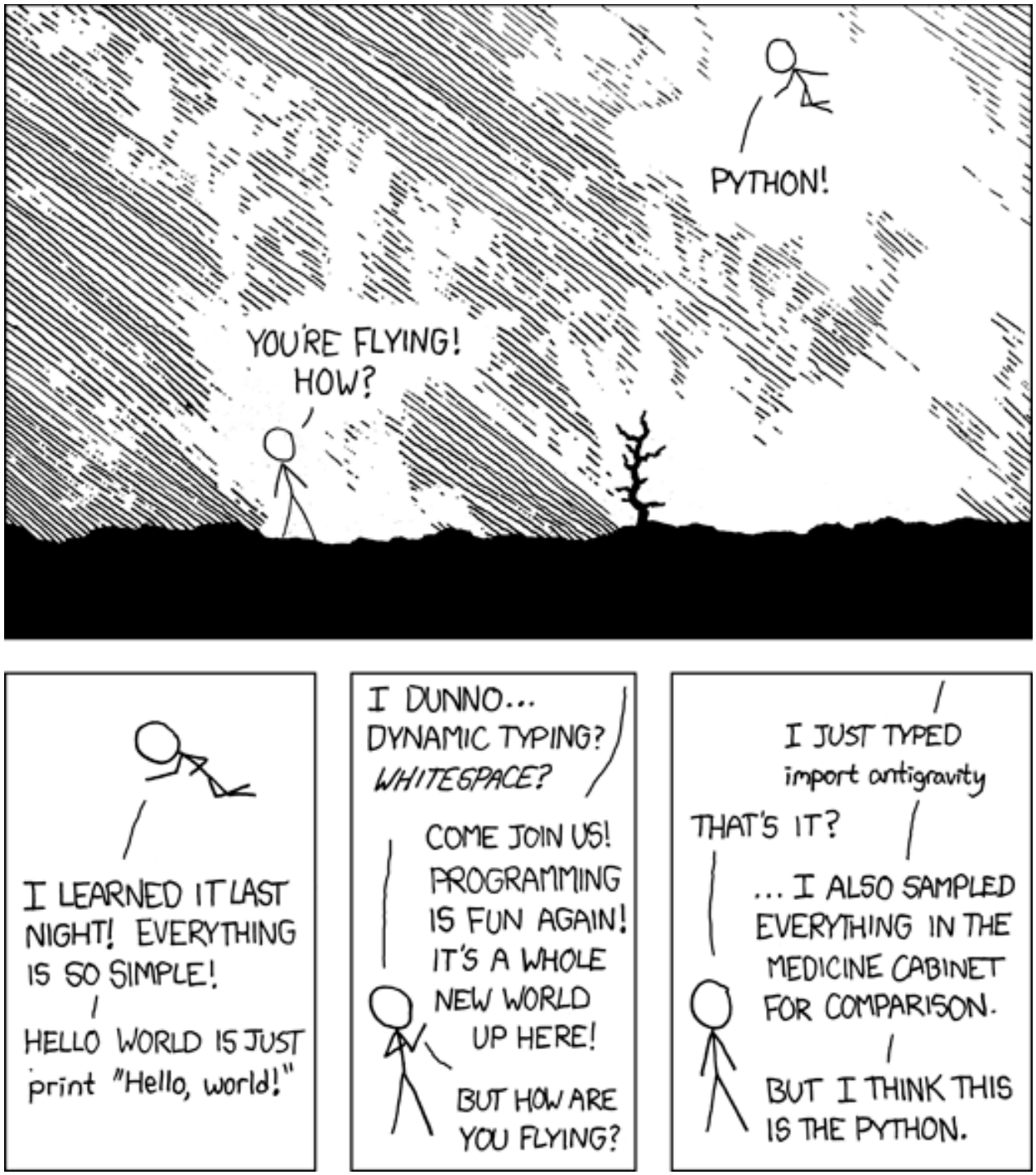
  – enormous advantages:

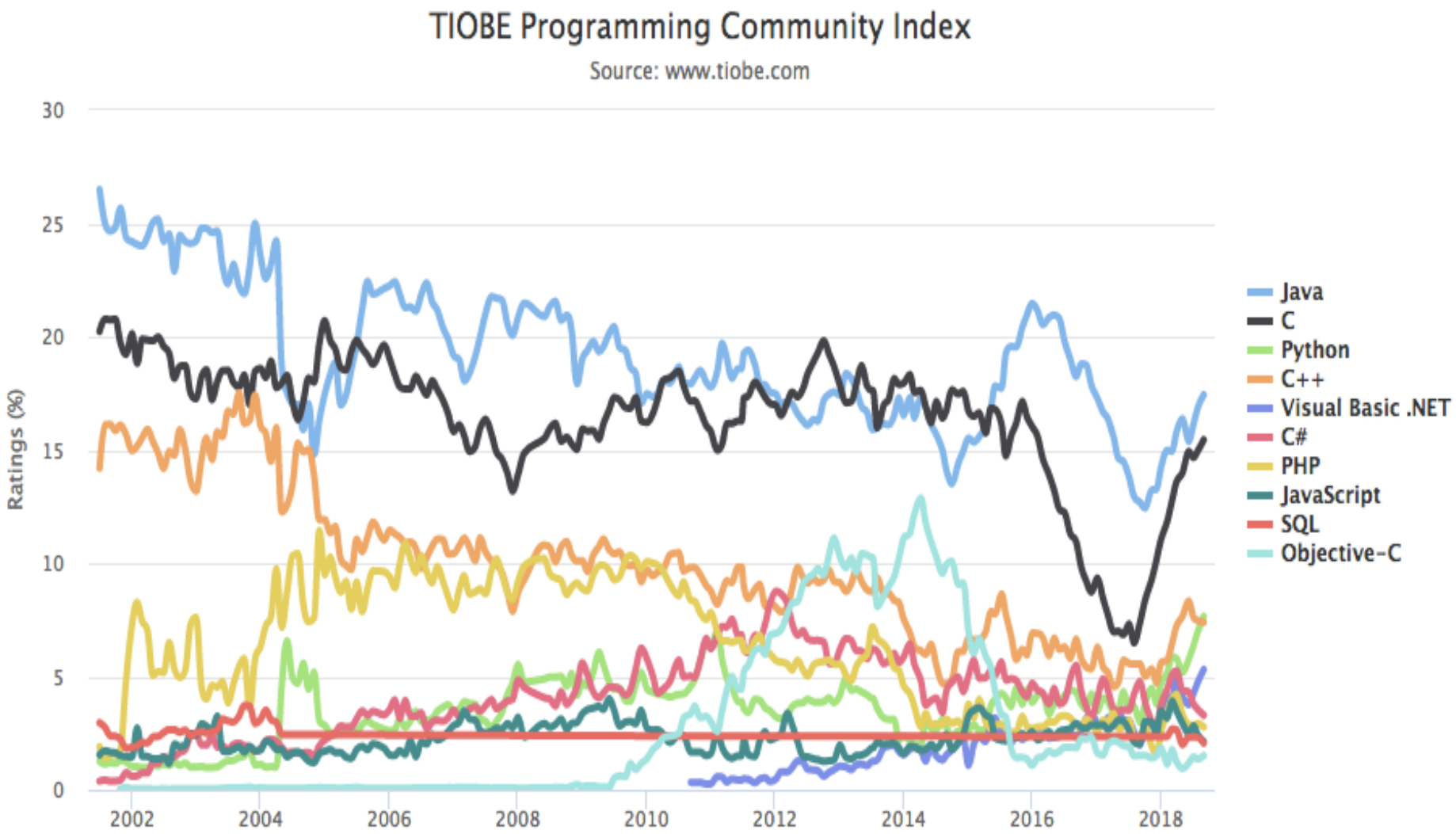
  portable: same program can be translated for different machines

  faster, cheaper hardware helps make this happen

```
var sum = 0;  // javascript
var num = prompt("Enter new value, or 0 to end")
while (num != 0) {
       sum = sum + parseInt(num)
       num = prompt("Enter new value, or 0 to end")
}
alert("Sum = " + sum)
```

xkcd.com/353


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Java
C
Python
C++
Visual Basic .NET
C#
PHP
JavaScript
SQL
Objective-C

Why so many programming languages?

- **every language is a tradeoff among competing pressures**
  - reaction to perceived failings of others; personal taste
- **notation is important**
  - "Language shapes the way we think and determines what we can think about."

    Benjamin Whorf
  - the more natural and close to the problem domain, the easier it is to get the machine to do what you want

- **higher-level languages hide differences between machines and between operating systems**
- **we can define idealized "machines" or capabilities and have a program simulate them -- "virtual machines"**
  - programming languages are another example of Turing equivalence

xkcd.com/303


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.