JOHN JAY COLLEGE OF CRIMINAL JUSTICE

Programming Languages:

Lecture 1

Chapter 1: Introduction

Jinwoo Kim
jwkim@jjay.cuny.edu

Introduction

- Discussion of syllabus
- Please visit class homepage often for any announcement or updates
  - http://jjcweb.jjay.cuny.edu/jwkim/class/csci374-summer-25/
- Reading assignment:
  - Chapter 1 from textbook

Course Objective

- My goal is not to teach you a few more programming languages
  - You already know how to program
  - Comparative study of programming languages
  - General issues in design and implementation

- Topics
  - Programming paradigms
  - Syntax and semantics
  - Interpreters
  - Names, scope and binding
  - Types and type analysis
  - Memory management
  - Control abstraction

Chapter 1 Topics

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories
- Language Design Trade-Offs
- Implementation Methods
- Programming Environments

Eric Gunnerson - Why are there so many programming languages?

https://www.youtube.com/watch?v=mf8eihUbcjg

Case Study: Ruby

- Ruby was conceived on February 24, 1993 by Yukihiro Matsumoto who wished to create a new language that balanced functional programming with imperative programming

- Matsumoto has stated, "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language“

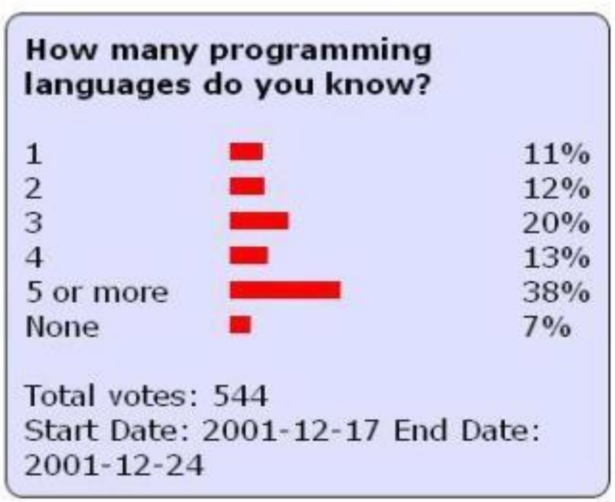
- At a Google Tech Talk in 2008 Matsumoto further stated, "I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."

- http://www.youtube.com/watch?v=ix2DeCzuckc

How many programming languages?

- Interesting survey from DoD in 1994
  - Try to compare the number of programming languages used in DoD as compared of 20 years ago
  - 1974: minimum of 450 languages were used in DoD
  - 1994: 37 used in major systems

- Interesting polls from "programmers heaven website" in December, 2001



PYPL Popularity of Programming Language

The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.

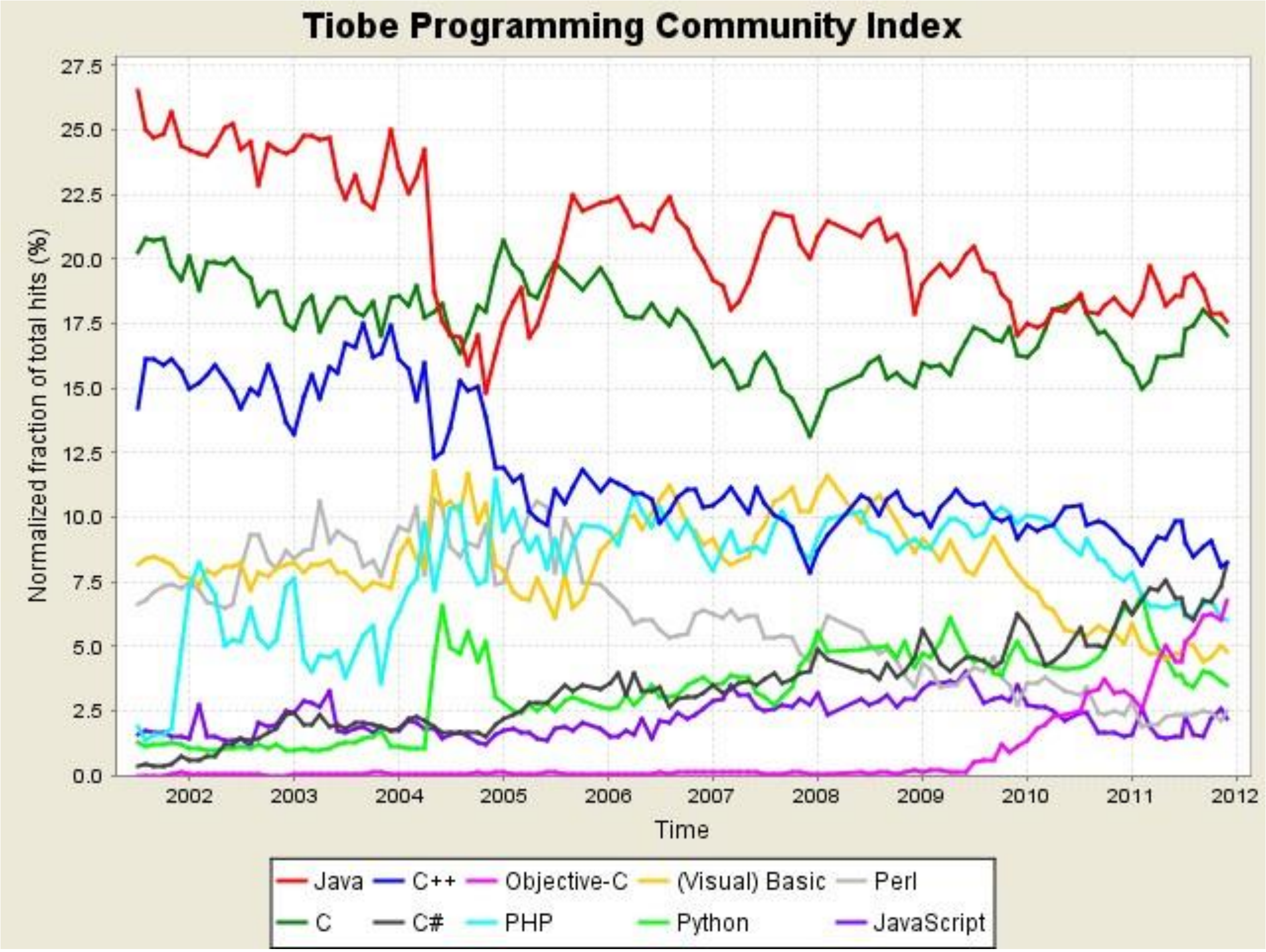
The more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from Google Trends.

If you believe in collective wisdom, the PYPL Popularity of Programming Language index can help you decide which language to study, or which one to use in a new software project.

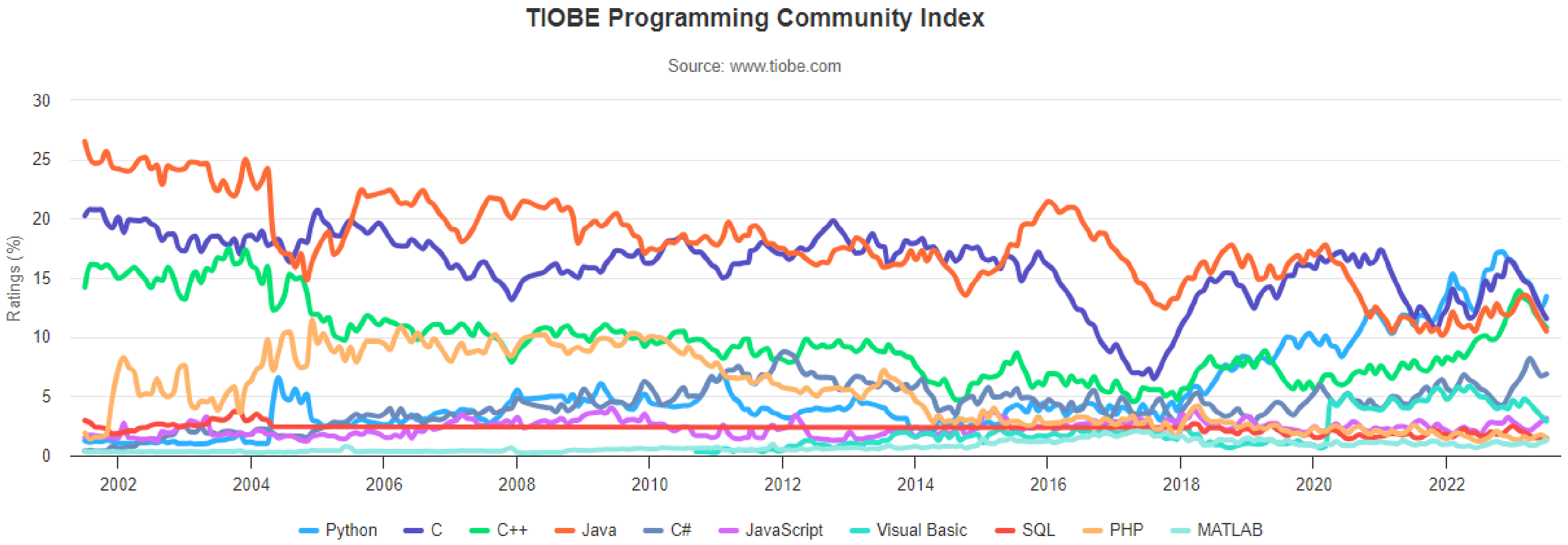
Worldwide, Dec 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.34 %	-1.0 %
2		Java	16.93 %	-0.8 %
3		JavaScript	9.28 %	+0.3 %
4		C#	6.89 %	-0.3 %
5		C/C++	6.64 %	-0.3 %
6		PHP	5.19 %	-1.0 %
7		R	3.98 %	-0.1 %
8	↑↑	TypeScript	2.79 %	+1.1 %
9	↑↑	Swift	2.23 %	+0.6 %
10	↓↓	Objective-C	2.22 %	+0.1 %
11	↑↑	Go	2.02 %	+0.7 %
12	↑↑↑	Rust	1.78 %	+0.8 %
13	↓↓↓↓	Kotlin	1.71 %	-0.0 %
14	↓↓	Matlab	1.61 %	+0.0 %
15	↑	Ruby	1.12 %	+0.2 %
16	↓↓	VBA	1.08 %	-0.1 %
17		Ada	0.96 %	+0.2 %
18	↑↑↑	Dart	0.85 %	+0.4 %
19	↓	Scala	0.69 %	-0.0 %
20	↑↑↑	Lua	0.65 %	+0.3 %
21	↓↓	Visual Basic	0.57 %	-0.1 %
22	↓↓	Abap	0.55 %	+0.1 %
23	↓	Perl	0.53 %	+0.1 %
24		Groovy	0.36 %	+0.0 %
25		Cobol	0.33 %	+0.0 %
26		Haskell	0.25 %	+0.0 %

The long term trends (2000 ~ 2012) for the top 10 programming languages



The long term trends for the top 10 programming languages



Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

Programming Domains

- Scientific applications
  - Large number of floating point computations
  - Use of arrays
  - Fortran
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL
- Artificial intelligence
  - Symbols rather than numbers manipulated
  - Use of linked lists
  - LISP
- Systems programming
  - Need efficiency because of continuous use
  - C
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

Interesting links

- How to shoot yourself in the foot with languages?
  - http://www-users.cs.york.ac.uk/~susan/joke/foot.htm
- ACM "Hello World" project
  - http://www2.latech.edu/~acm/HelloWorld.html
- 99 bottles of beer
  - http://web.mit.edu/kenta/www/two/beer.html
- The Great Computer Language Shootout – A collection of benchmarks performed on many languages
  - http://dada.perl.it/shootout/
- Scripting: Higher Level Programming for the 21st Century
  - http://www.tcl.tk/doc/scripting.html

Language Evaluation Criteria

- **Readability**: the ease with which programs can be read and understood
- **Writability**: the ease with which a language can be used to create programs
- **Reliability**: conformance to specifications
  - e.g., performs to its specifications
- **Cost**: the ultimate total cost

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Few feature multiplicity
    - **Less means of doing the same operation**
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures
    - **e.g., while statement**
- Data types and structures
  - The presence of adequate facilities for defining data types and structures

Evaluation Criteria: Readability (Continued)

- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful keywords

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
  - A set of relatively convenient ways of specifying operations
    - **e.g., the inclusion of `for` statement in many modern languages**

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support “natural” ways of expressing an algorithm will necessarily use “unnatural” approaches, and hence reduced reliability

Evaluation Criteria: Cost

- Training programmers to use the language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another
- Generality
  - The applicability to a wide range of applications
- Well-definedness
  - The completeness and precision of the language's official definition

Influences on Language Design

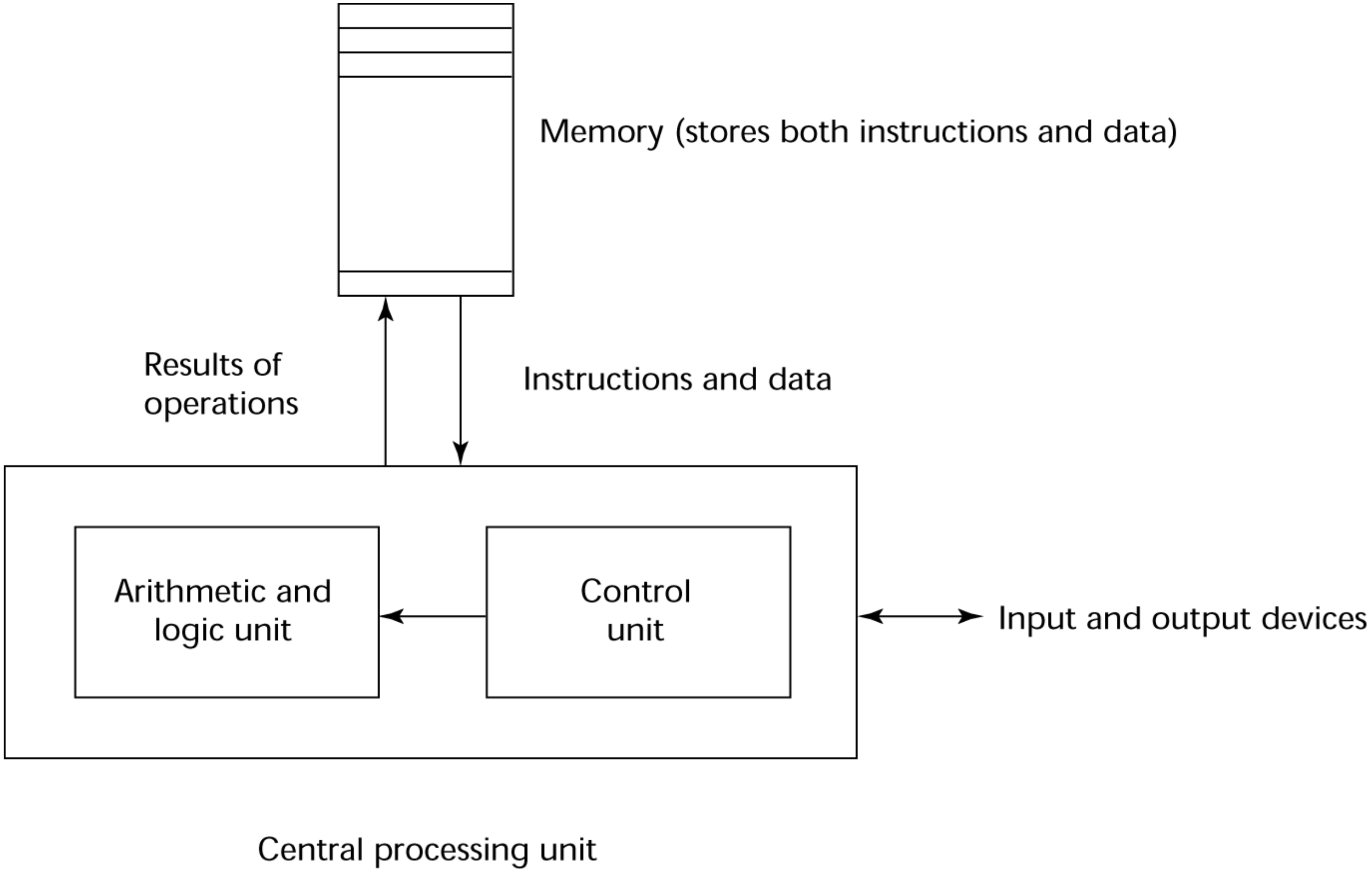
- Computer Architecture
  – Languages are developed around the prevalent computer architecture, known as the *Von Neumann* architecture

- Programming Methodologies
  – New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - **Variables model memory cells**
    - **Assignment statements model piping**
    - **Iteration is efficient**

The Von Neumann Architecture



The Von Neumann Architecture

- Fetch-execute-cycle (on a Von Neumann architecture computer)

```
initialize the program counter
repeat forever
    fetch the instruction pointed by instruction counter
    increment the counter
    decode the instruction
    execute the instruction
end repeat
```

Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed a lot faster than the above connection speed; the connection speed thus results in a *bottleneck*
- Known as Von Neumann bottleneck; it is the primary limiting factor in the speed of computers

Programming Methodologies Influences

- 1950s and early 1960s
  - Simple applications
  - worry about machine efficiency
  - Assembly and Fortran Languages

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
     jmp   .L2
```

Before

```
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
     jmp   .L2
```

After: 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
```

Programming Methodologies Influences

- Late 1960s
  - People efficiency became important
    - **readability, better control structures**
  - structured programming
  - top-down design and step-wise refinement

Programming for the masses

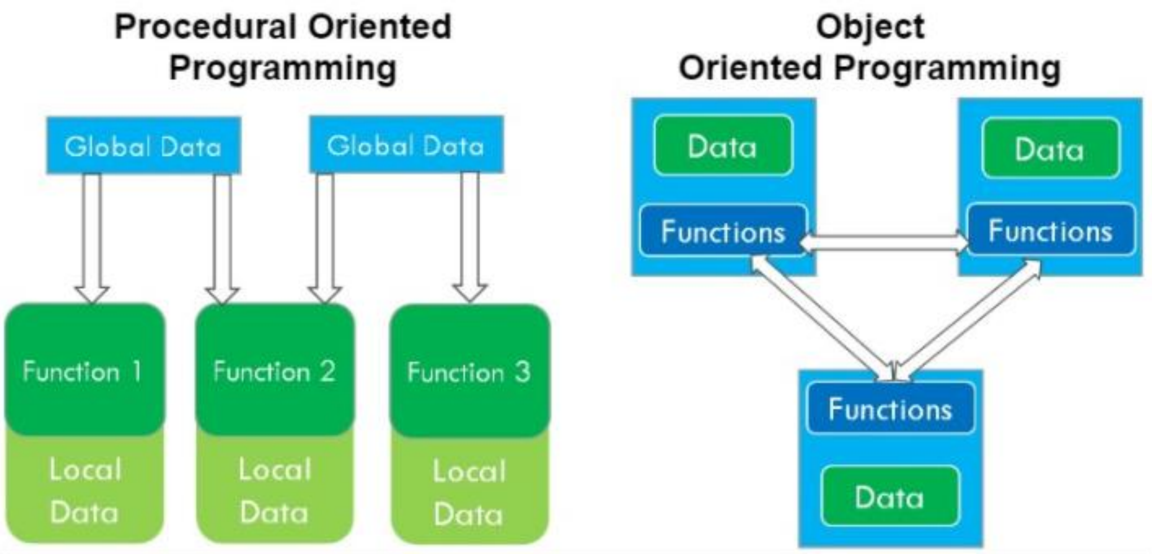
```
10 PRINT "GUESS A NUMBER BETWEEN ONE AND TEN"
20 INPUT A$
30 IF A$ <> "5" THEN GOTO 60
40 PRINT "GOOD JOB, YOU GUESSED IT"
50 GOTO 100
60 PRINT "YOU ARE WRONG. TRY AGAIN"
70 GOTO 10
100 END
```

Efficiency for systems programming

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

Programming Methodologies Influences

- Late 1970s:
  - Process-oriented to data-oriented
    - **data abstraction**
- Middle 1980s: Object-oriented programming
  - Data abstraction + Inheritance + Polymorphism



Comparing procedural vs object-oriented programming. Source: Sakpal 2018.

Language Categories

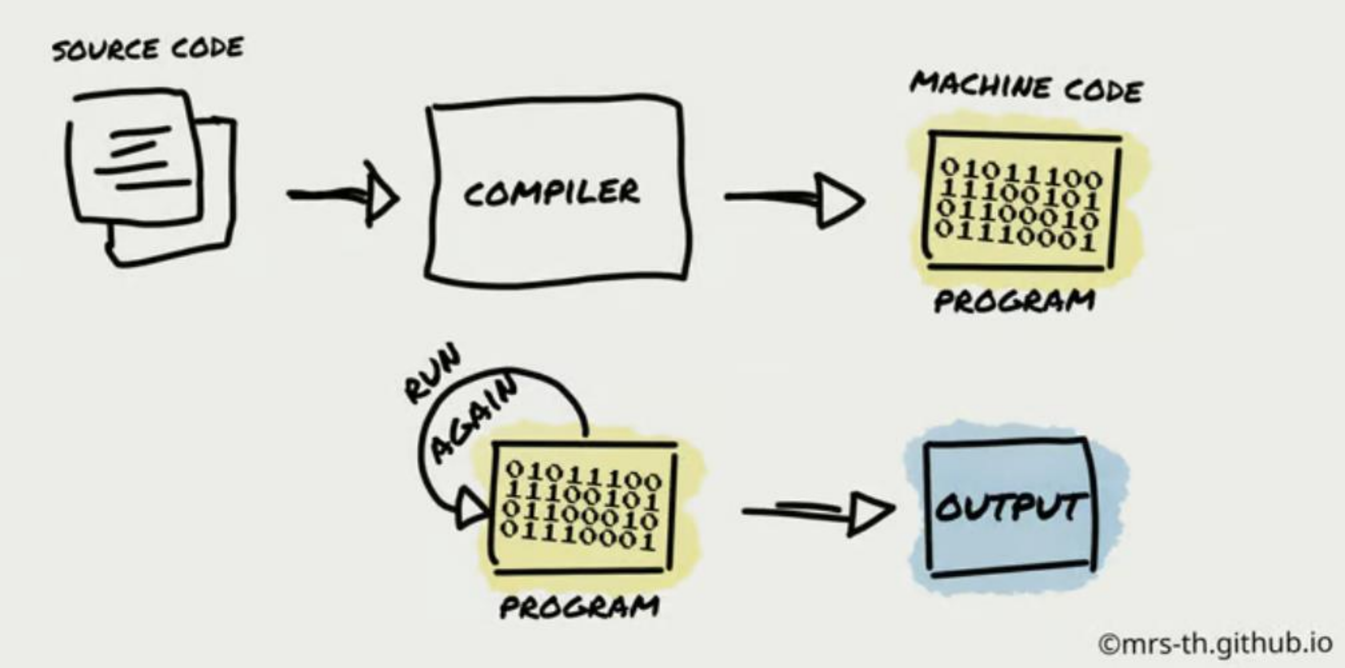
- Imperative
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include script languages and visual languages
  - Examples: C, C++, Java, Perl, JavaScript, Visual Basic, etc
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, ML, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Object-oriented
  - Data abstraction, inheritance, late binding
  - Examples: Java, C++
- Markup/programming hybrid
  - Markup languages extended to support some programming
  - Examples: JSTL, XSLT

Language Design Trade-Offs

- Reliability vs. Cost of Execution
  - Conflicting criteria
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- Readability vs. Writability
  - Another conflicting criteria
  - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. Reliability
  - Another conflicting criteria
  - Example: C++ pointers are very powerful and flexible but are unreliable

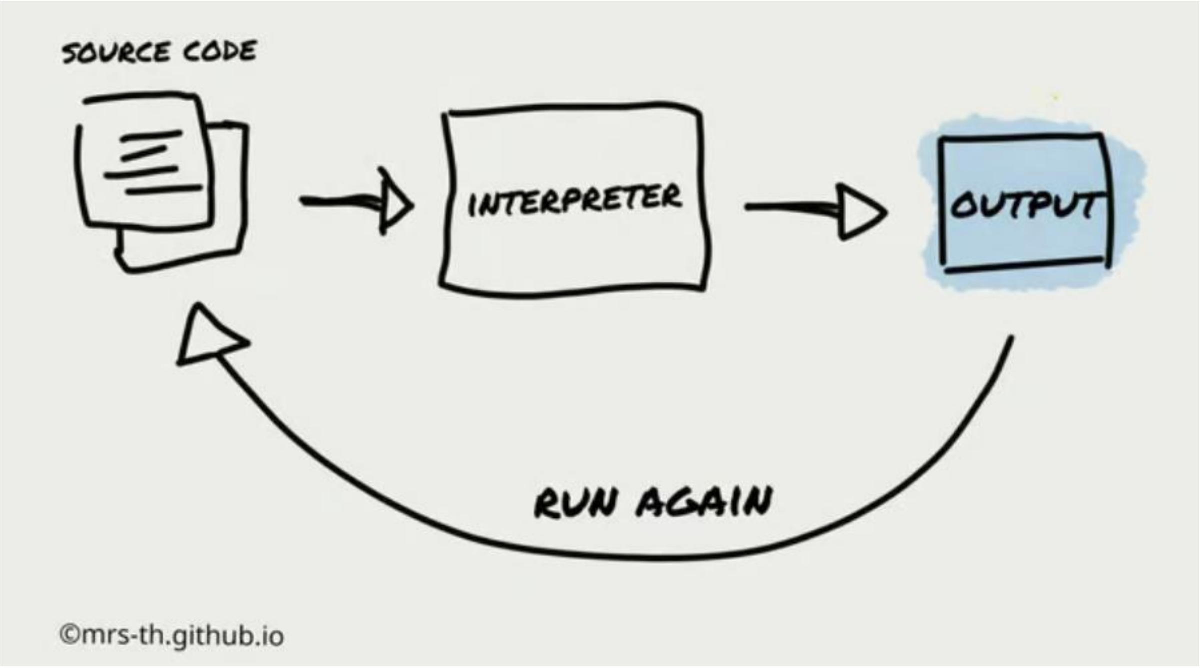
Implementation Methods

- Compilation
  - Programs are translated into machine language



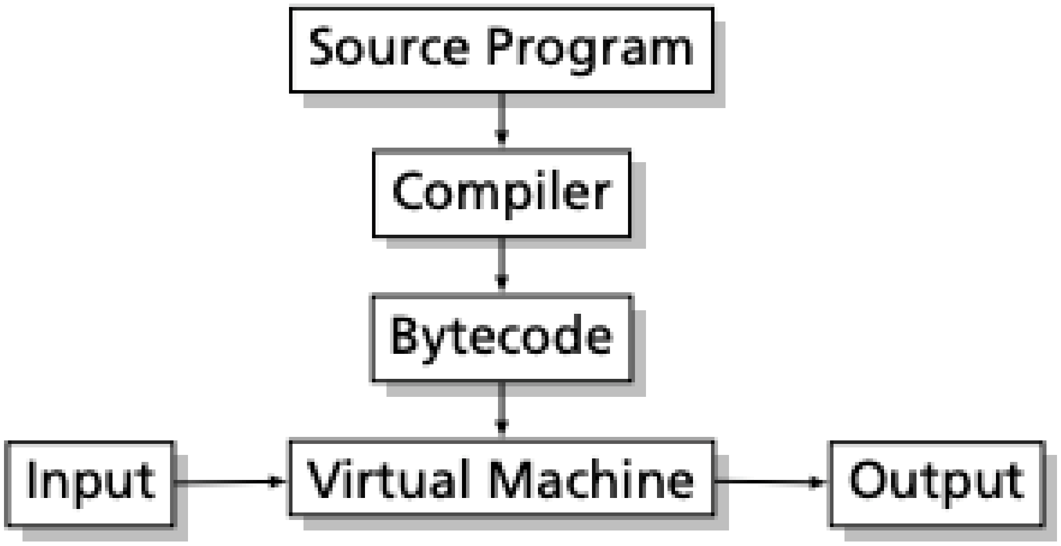
Implementation Methods (Continued)

- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter



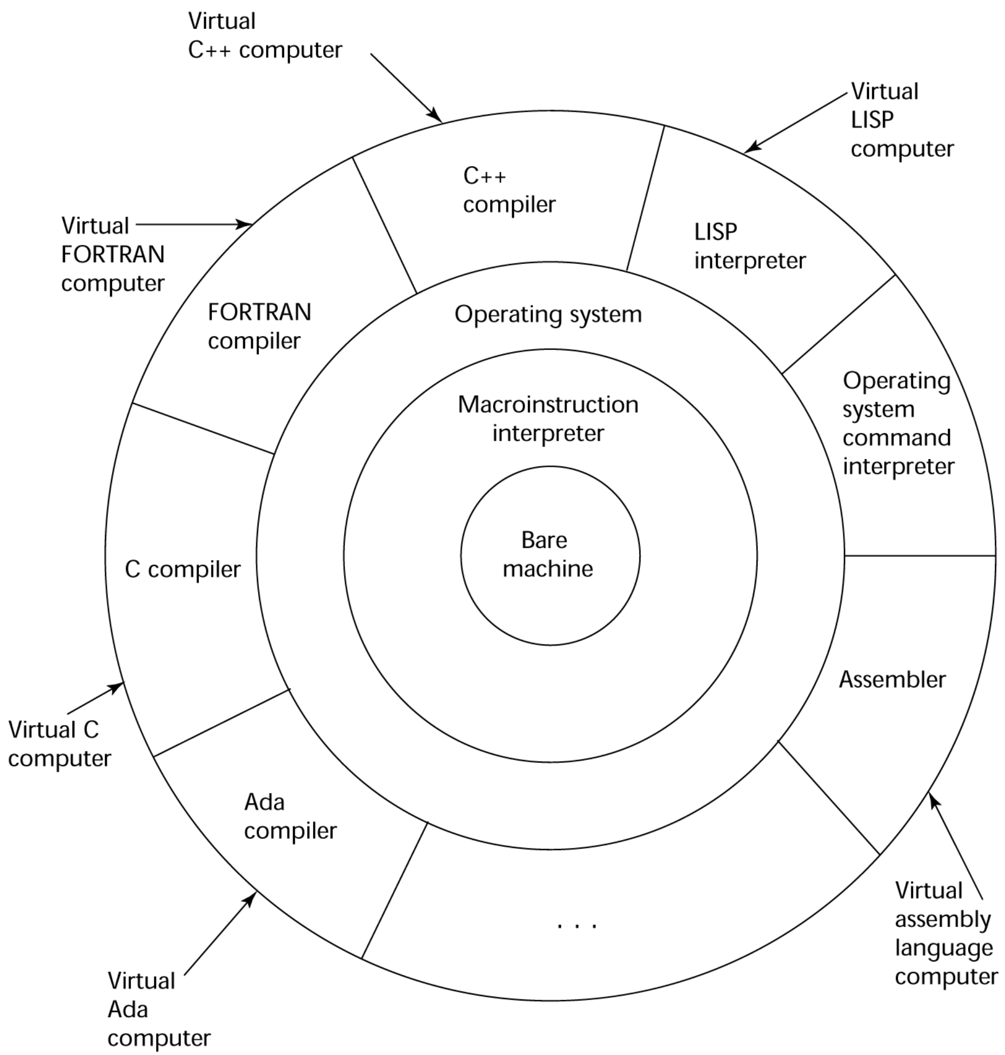
Implementation Methods (Continued)

- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
  - Ex. Java Bytecode Interpreter



Layered View of Computer

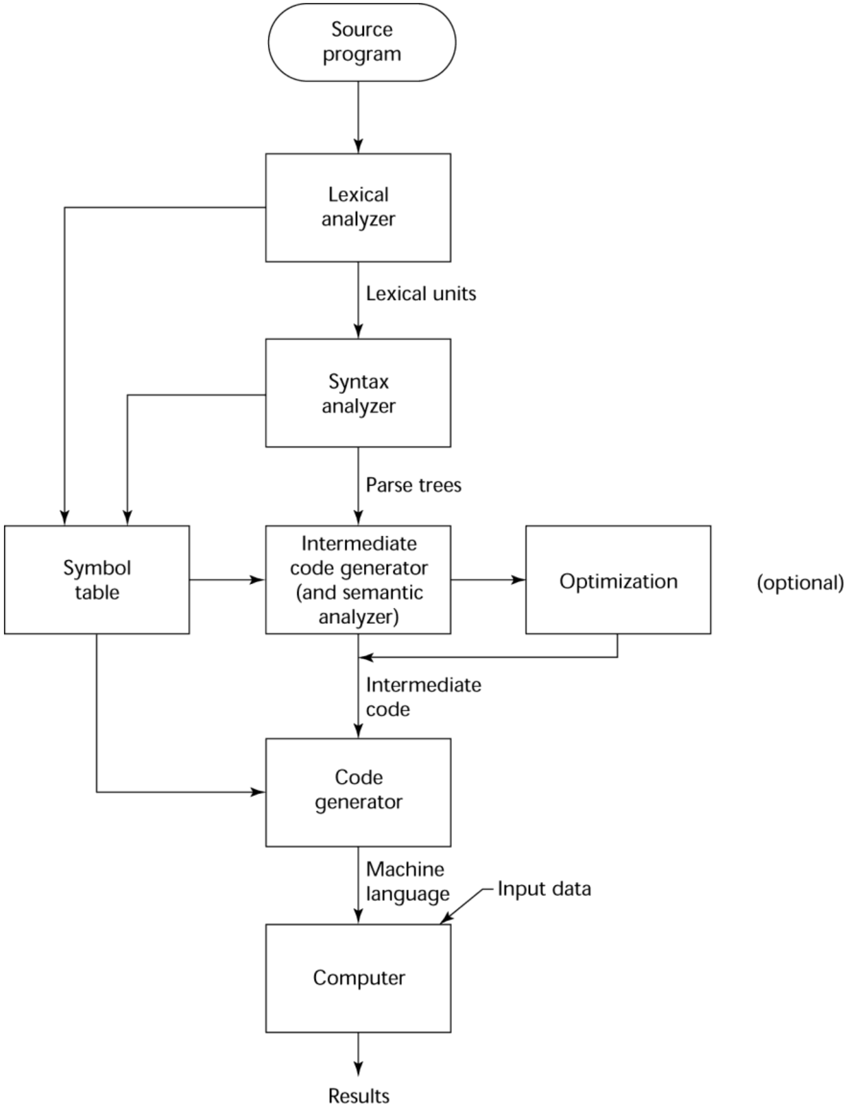
The operating system and language implementation are layered over Machine interface of a computer



Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
  - lexical analysis: converts characters in the source program into lexical units
  - syntax analysis: transforms lexical units into *parse trees* which represent the syntactic structure of program
  - semantics analysis: generate intermediate code
  - code generation: machine code is generated

The Compilation Process

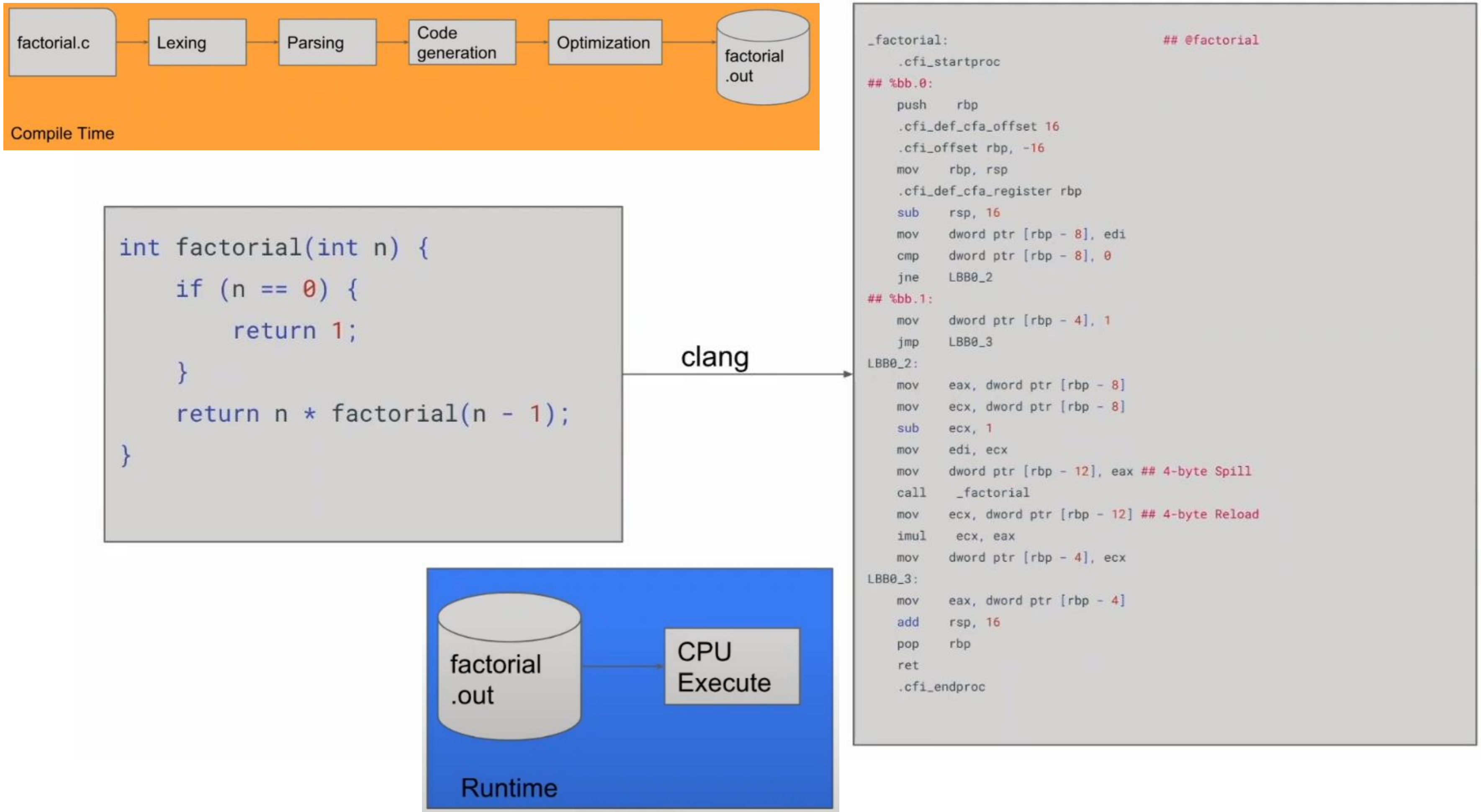


Additional Compilation Terminologies

- **Load module** (executable image): the user and system code together

- **Linking and loading**: the process of collecting system program and linking them to user program

Compilation Takes Time…



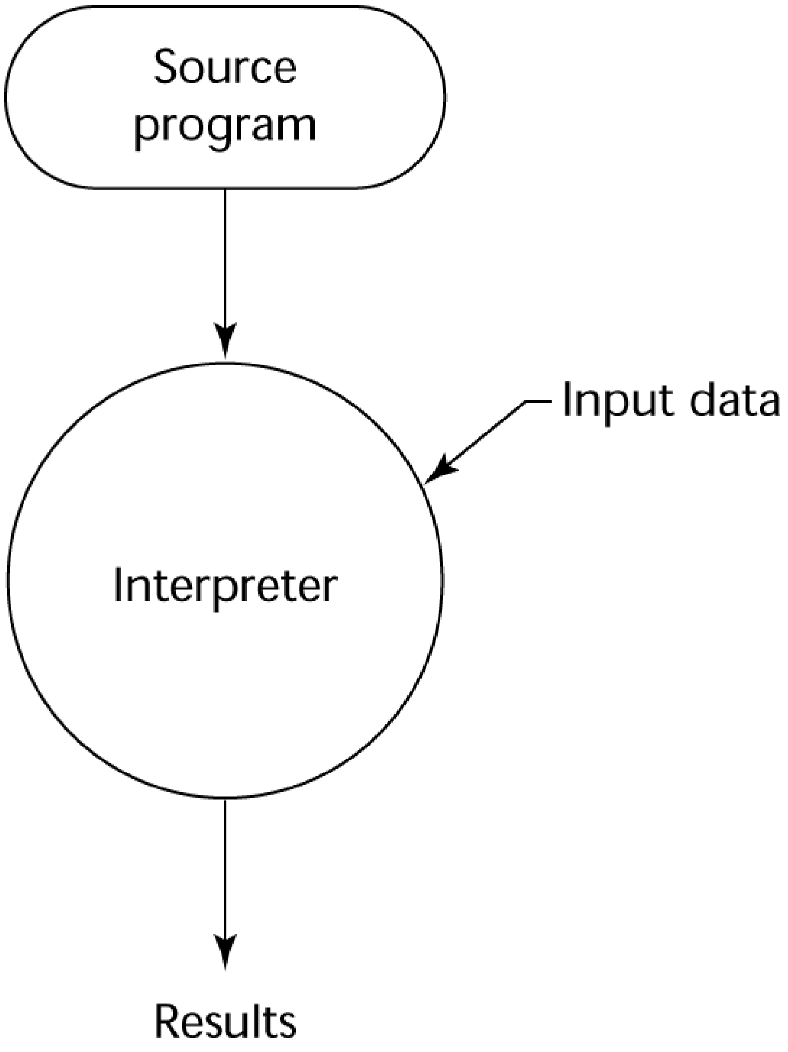
Compilation Overhead

- Compilers are hard to write
- Compilation process can be quite slow
- Hard to port to different CPU architectures
- Machine code is not efficiently distributable. Binaries have to be FAT to work on multiple architectures.
- Hard to port to different operating systems.
  - OS's have different binary executable formats, environments, runtimes, syscalls
  - e.g Mach-O on OSX/iOS. EXE on Windows, ELF on linux
- Slower development cycle.

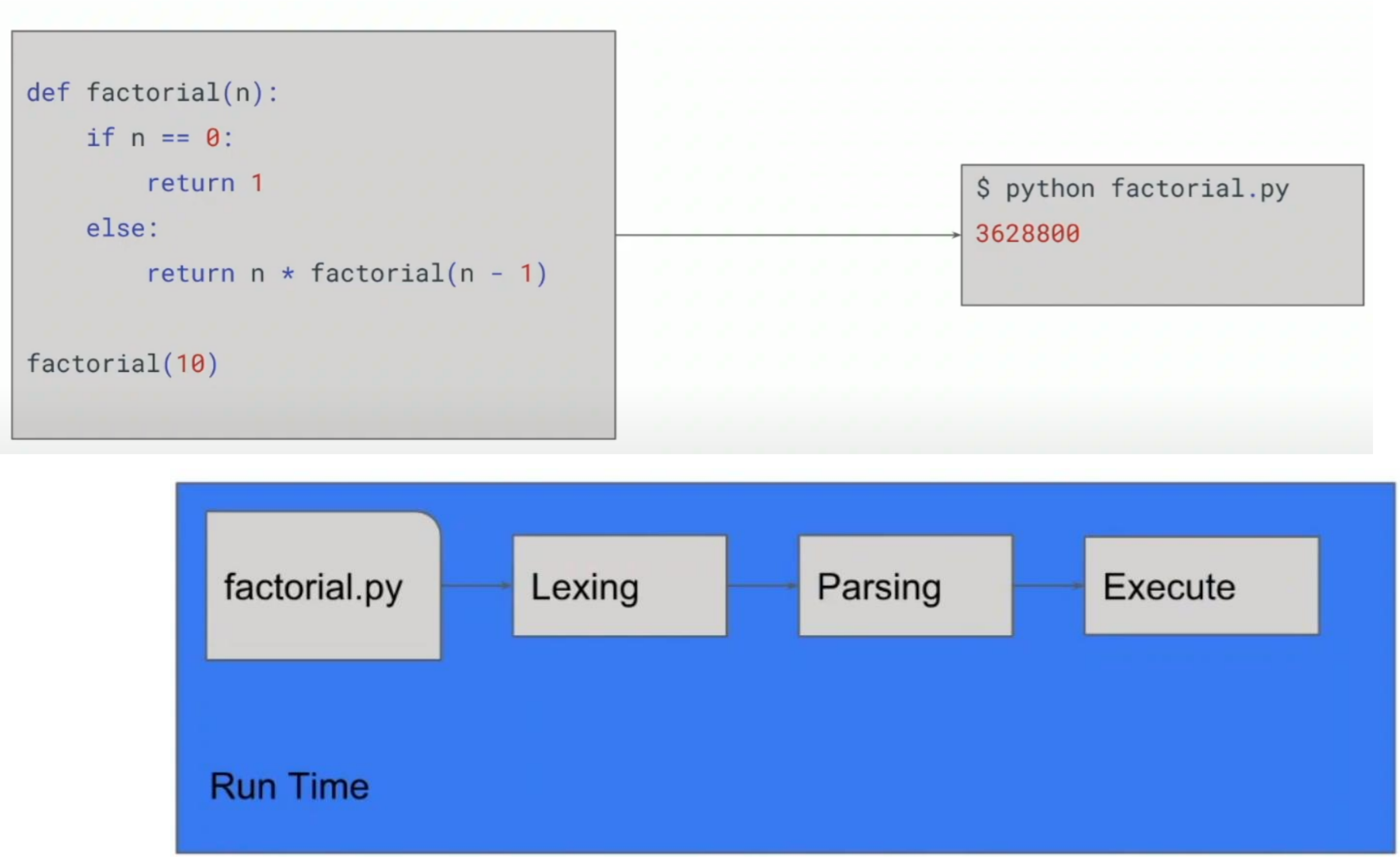
Pure Interpretation

- No translation
- Easier implementation of programs
  - run-time errors can be easily and immediately displayed
- Slower execution
  - 10 to 100 times slower than compiled programs
- Often requires more space
- Now rare for traditional high-level languages
- Significant comeback with some Web scripting languages
  - e.g., JavaScript, PHP

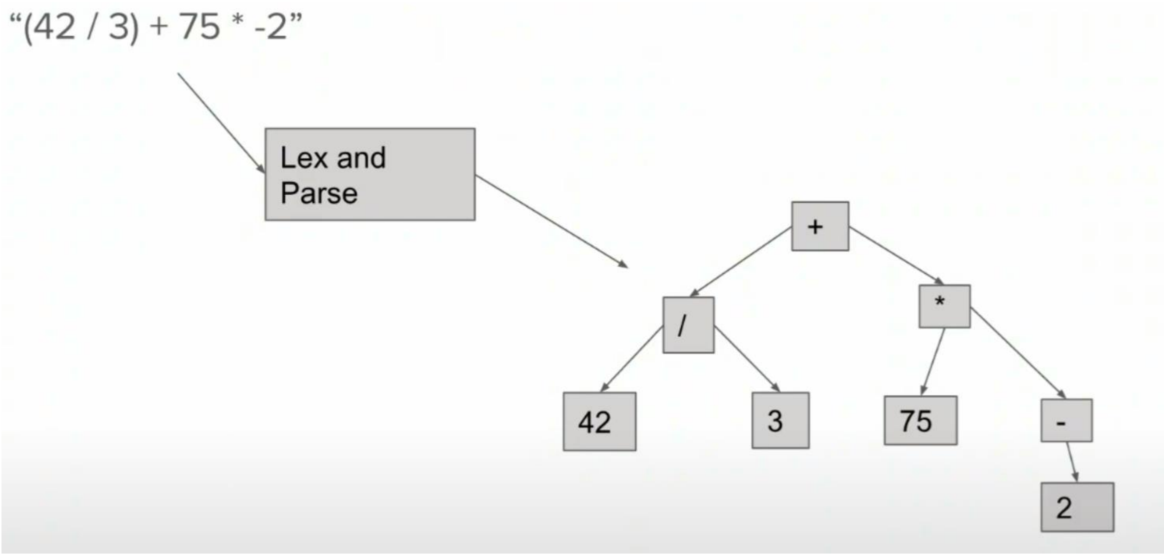
Pure Interpretation Process



Pure Interpretation Process



How Interpreter Works?



- Source code is portable. "Write once, run everywhere"
- Host just needs to have an interpreter available.
- EASY TO WRITE
- "Compilation" is faster. Mainly cos there's no code generation/optimization steps.

Performance of Interpreter Bad…

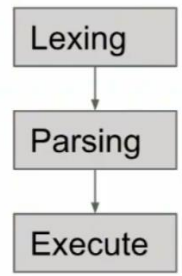
"x + 5"

Compiled

```
add    edi, 5
```

```
    mov     qword ptr [rbp - 16], rdi
    mov     rdi, qword ptr [rbp - 16]
    mov     eax, dword ptr [rdi]
    mov     edi, eax
    mov     rcx, rdi
    sub     rcx, 8
    mov     qword ptr [rbp - 32], rdi ## 8-byte Spill
    mov     qword ptr [rbp - 40], rcx ## 8-byte Spill
    ja     LBB0_16
## %bb.18:
    lea     rax, [rip + LJTI0_0]
    mov     rcx, qword ptr [rbp - 32] ## 8-byte Reload
    movsxd     rdx, dword ptr [rax + 4*rcx]
    add     rdx, rax
    jmp     rdx
LBB0_1:
    mov     rax, qword ptr [rbp - 16]
    mov     rax, qword ptr [rax + 8]
    mov     ecx, dword ptr [rax]
```

Interpreted



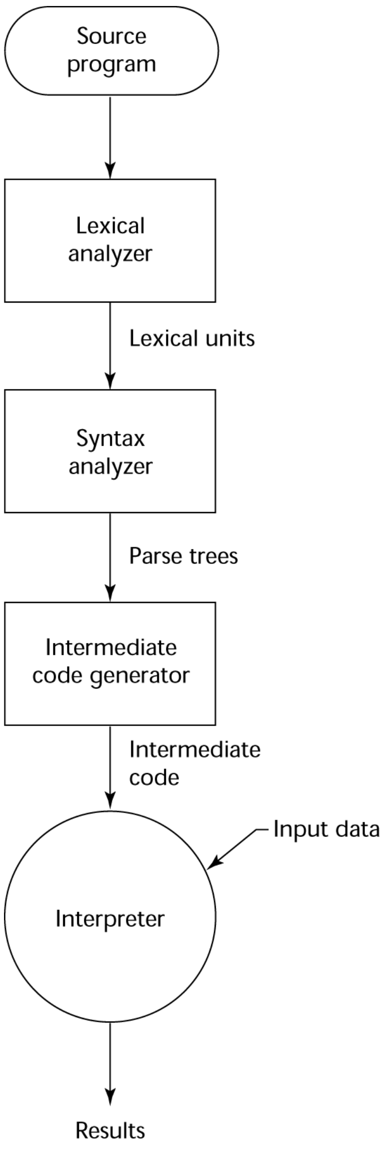

```
        dword ptr [rbp - 4], ecx
    jmp     LBB0_17
)_2:
    mov     rax, qword ptr [rbp - 16]
    mov     ecx, dword ptr [rax + 8]
    mov     dword ptr [rbp - 4], ecx
    jmp     LBB0_17
)_3:
    mov     rax, qword ptr [rbp - 16]
    mov     rdi, qword ptr [rax + 8]
    call     _executeIntExpression
    mov     dword ptr [rbp - 20], eax
    mov     rdi, qword ptr [rbp - 16]
    mov     rdi, qword ptr [rdi + 16]
    call     _executeIntExpression
    mov     dword ptr [rbp - 24], eax
    mov     rdi, qword ptr [rbp - 16]
    mov     eax, dword ptr [rdi]
    add     eax, -2
    mov     edi, eax
    sub     eax, 3
    mov     qword ptr [rbp - 48], rdi ## 8-byte Spill

LBB0_4:
    mov     eax, dword ptr [rbp - 20]
    add     eax, dword ptr [rbp - 24]
    mov     dword ptr [rbp - 4], eax
    jmp     LBB0_17
```

Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called *Java Virtual Machine*)

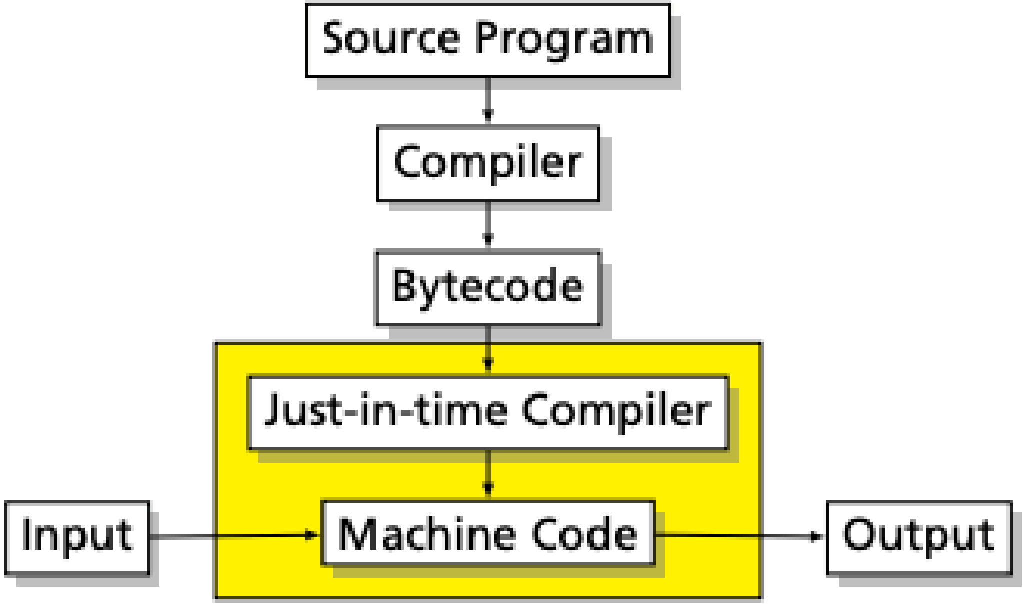
Hybrid Implementation Process



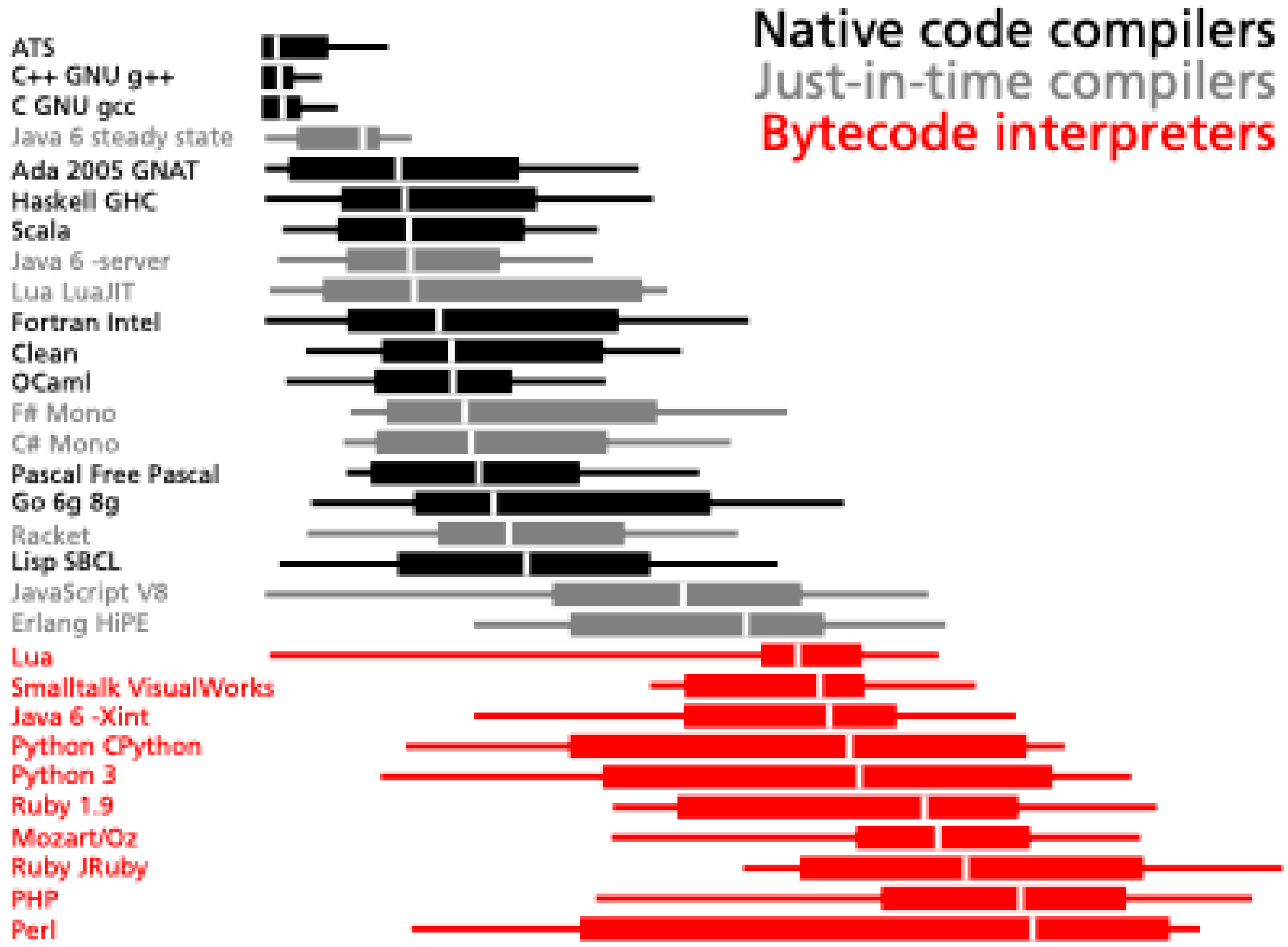
Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile intermediate language into machine code
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system

Just-in-Time Implementation Systems



Language Speeds Compared



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

Preprocessors

- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
- A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
- A well-known example: C preprocessor
  - expands `#include`, `#define`, and similar macros

Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that run on top of UNIX
- Borland JBuilder
  - An integrated development environment for Java
- Microsoft Visual Studio.NET
  - A large, complex visual environment
  - Used to program in C#, Visual BASIC.NET, Jscript, J#, and C++

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation

Homework

- Homework submission should follow schedule on the class homepage

- Read articles introduced in this lecture
  - Scripting: Higher Level Programming for the 21st Century
    - http://jjcweb.jjay.cuny.edu/jwkim/class/csci374-summer-25/scriptHistory.pdf
  - Who is John Von Neumann
    - http://jjcweb.jjay.cuny.edu/jwkim/class/csci374-summer-25/VonNeumann.pdf