NJIT

New Jersey's Science & Technology University

THE EDGE IN KNOWLEDGE

COURSE INTRODUCTION

- Programming languages have common concepts that are seen in all languages
- This course will discuss and illustrate these common concepts:
  - Syntax
  - Names
  - Types
  - Semantics
  - Memory Management
- We will show them in the context of:
  - Imperative Programming
  - Object-Oriented Programming
  - Functional Programming (maybe)
  - Logic Programming (maybe)
- With some discussion of:
  - Memory management
  - Event Handling
  - Concurrency

A Brief History

How and when and why did programming languages evolve?

- All processors, even the earliest ones, have a small set of instructions that can be performed
- A binary machine language, very close to the metal, that controls the machine
- VERY early on, the notion of a stored program (as compared to wired in or fed in with cards or tape) developed
- VERY early on, simple mnemonic languages were developed to make programming a little bit easier

Assembler Language

- Machine code has a series of possible operations and operands, represented by unique bit patterns
- Some mnemonic representation of the operations helps make things readable
- Symbolic names for resources like processor registers helps a lot.
- Symbolic names for memory locations helps even more

Higher Levels of Language

- A higher level of abstraction
- Easier to read and write
- Easier to develop systems for more complex uses

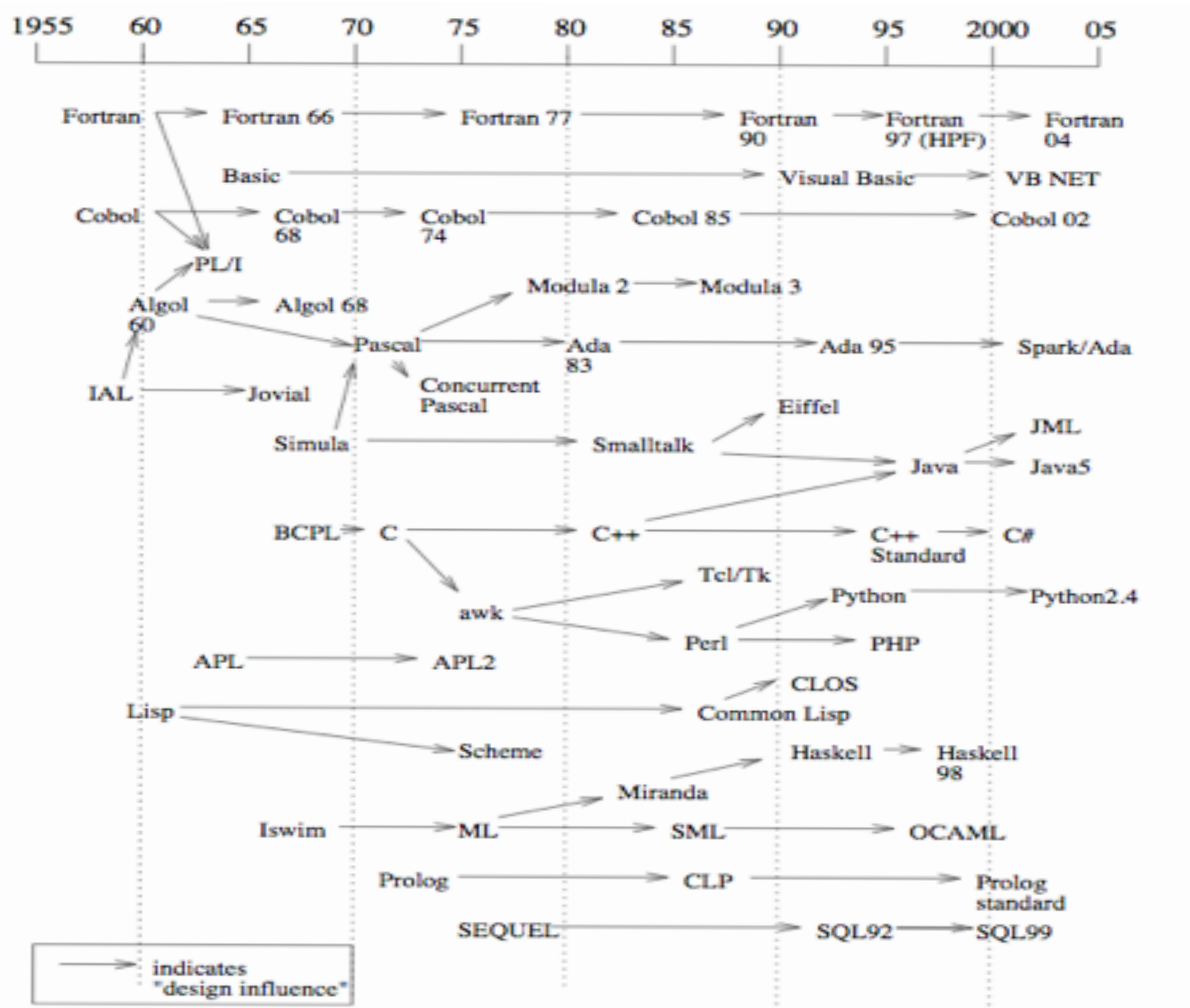


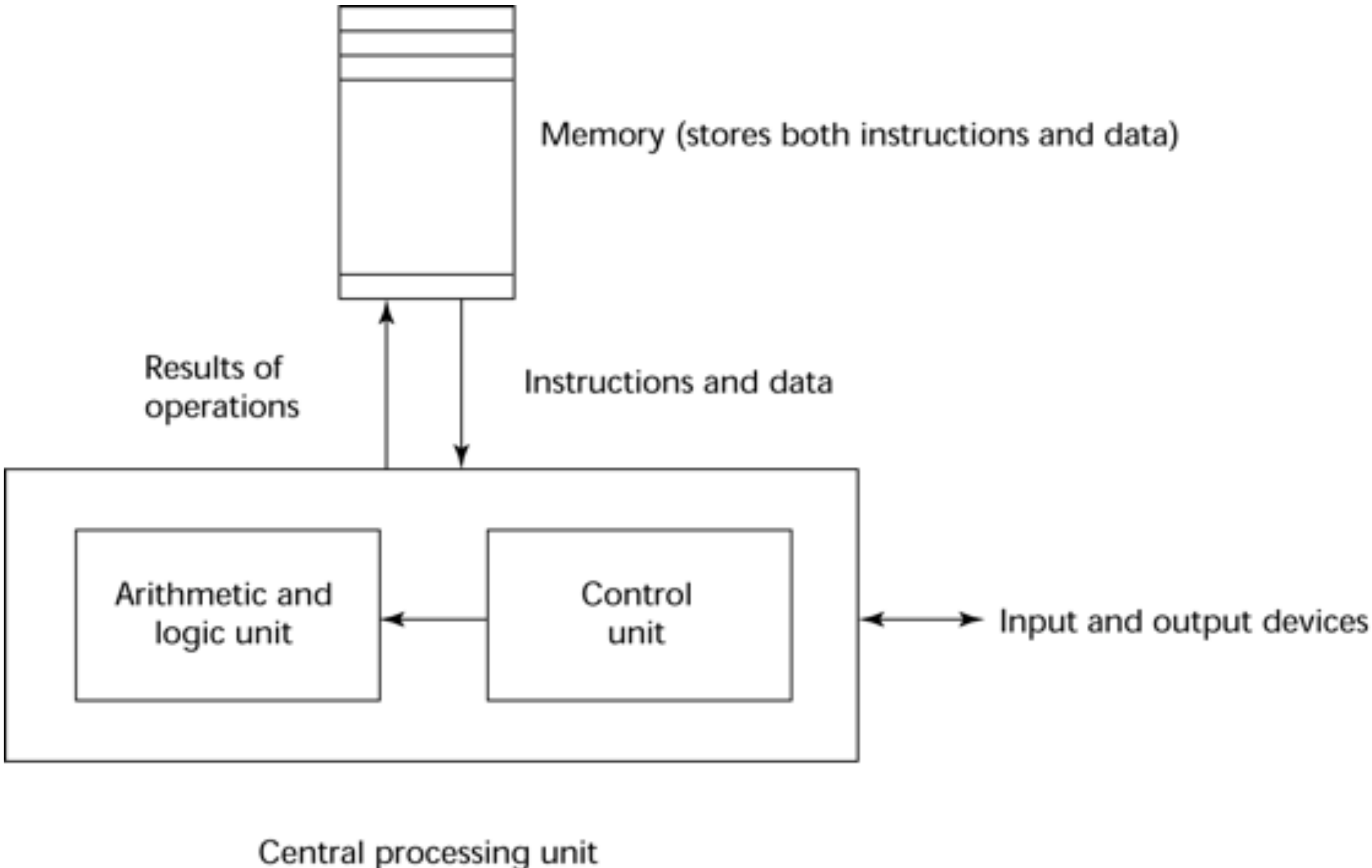
Figure 1.2: A Snapshot of Programming Language History

What Is Common

- All Languages will have
  - a syntax – what is and is not valid in the language
  - semantics – what is the meaning of elements of the language
  - names – what are the rules for names for things in the language
  - types – what values can things in the language take on

- All Languages run on some sort of machine

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 the 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 much faster than the speed of the connection; the connection speed thus results in a *bottleneck*
- Known as the *von Neumann bottleneck*; it is the primary limiting factor in the speed of computers

Computer Architecture Influences Languages

- 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

Programming Methodologies over Time

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include scripting languages
  - Include the visual languages
  - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme, ML, F#
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Markup/programming hybrid
  - Markup languages extended to support some programming
  - Examples: JSTL, XSLT

Language Design Trade-Offs

- Reliability vs. cost of execution
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- Readability vs. writeability

  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
- Writeability (flexibility) vs. reliability
  - Example: C++ pointers are powerful and very flexible but can be unreliable

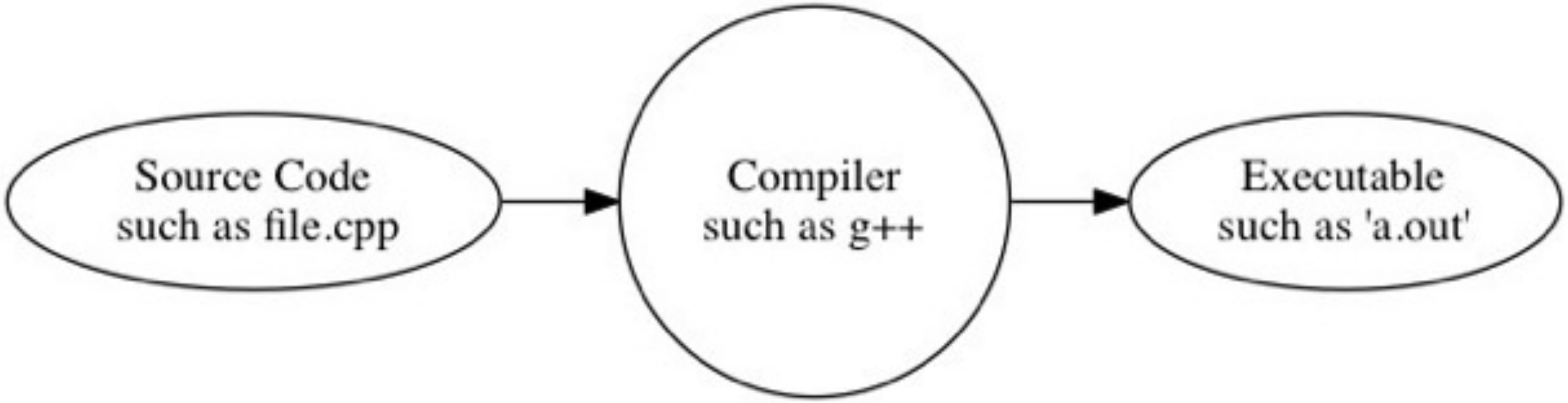
Implementation Methods

- Compilation
  - Programs are translated into machine language; includes JIT systems
  - Use: Large commercial applications

- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
  - Use: Small programs or when efficiency is not an issue

- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
  - Use: Small and medium systems when efficiency is not the first concern

Compiling

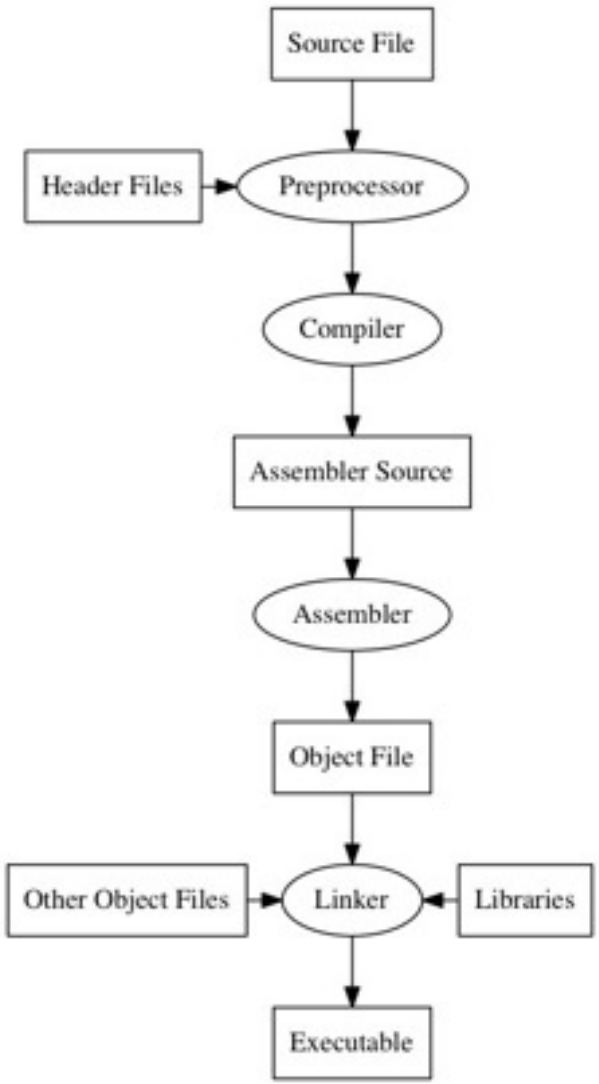


Source Code
such as file.cpp
Compiler
such as g++
Executable
such as 'a.out'

Compilation

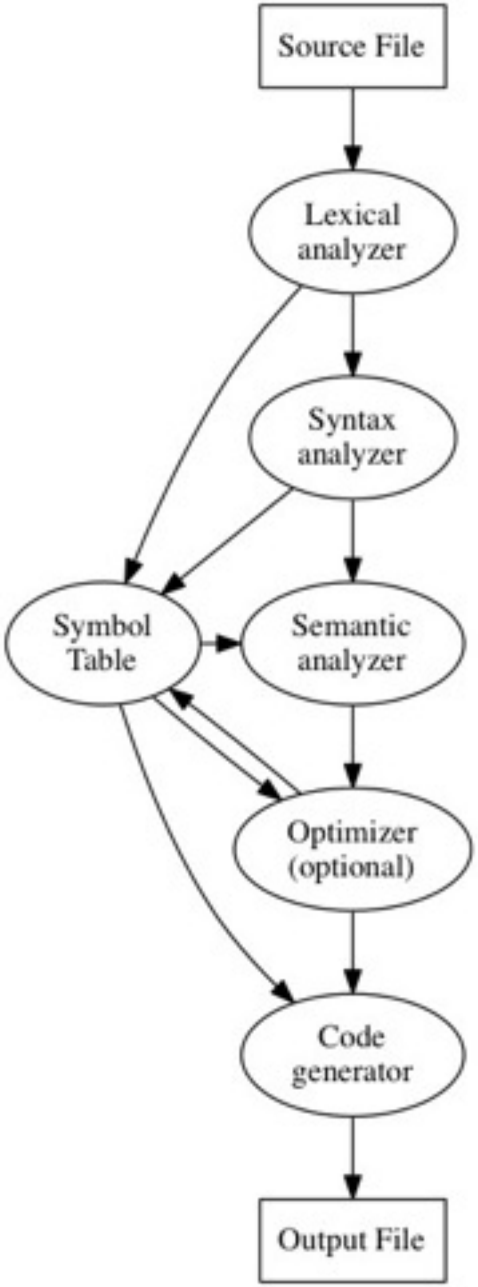
- Translate high-level program (source language) into machine code (machine language)
- This is specific to the machine you are compiling for
- Compiling on one machine to run on another is called “cross-compilation”
- Slow translation, fast execution

Steps In Compile



Inside a Compiler

- lexical analyzer converts characters in the source program into lexical units
- syntax analyzer transforms lexical units into *parse trees* which represent the syntactic structure of program
- Semantics analyzer enforces the semantic rules of the language
- Optimizer improves the code
- Code generator creates the output (assembler or byte code)



Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately be 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)

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*)

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

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

- A collection of tools used in software development
- Every system has a compiler or interpreter and extra tools such as archivers (to make libraries) and debuggers
- Available compilers
  - gcc and g++
  - clang
  - MinGW
  - Visual C/C++
- Debuggers allow for breakpointing and single stepping through your program, examining the value of variables and the flow of the program

Programming Environments

- Some environments provide smart editors with features like syntax coloring, coding help, templates
- Common ones
  - Xcode – Macintosh: iPhone/iPad development, supports C, C++, Objective C, Swift
  - Eclipse – Android development, supports C, C++, Java, others
  - Qt – cross-platform: C, C++
  - Code::Blocks, C, C++
  - Microsoft Visual Studio.NET
    - Used to build Web applications and non-Web applications in any .NET language
  - NetBeans
    - Related to Visual Studio .NET, except for applications in Java

NJIT

THE EDGE IN KNOWLEDGE