

PR2 - Programming 2

Lecture 1

Overview of Programming Languages

Outline

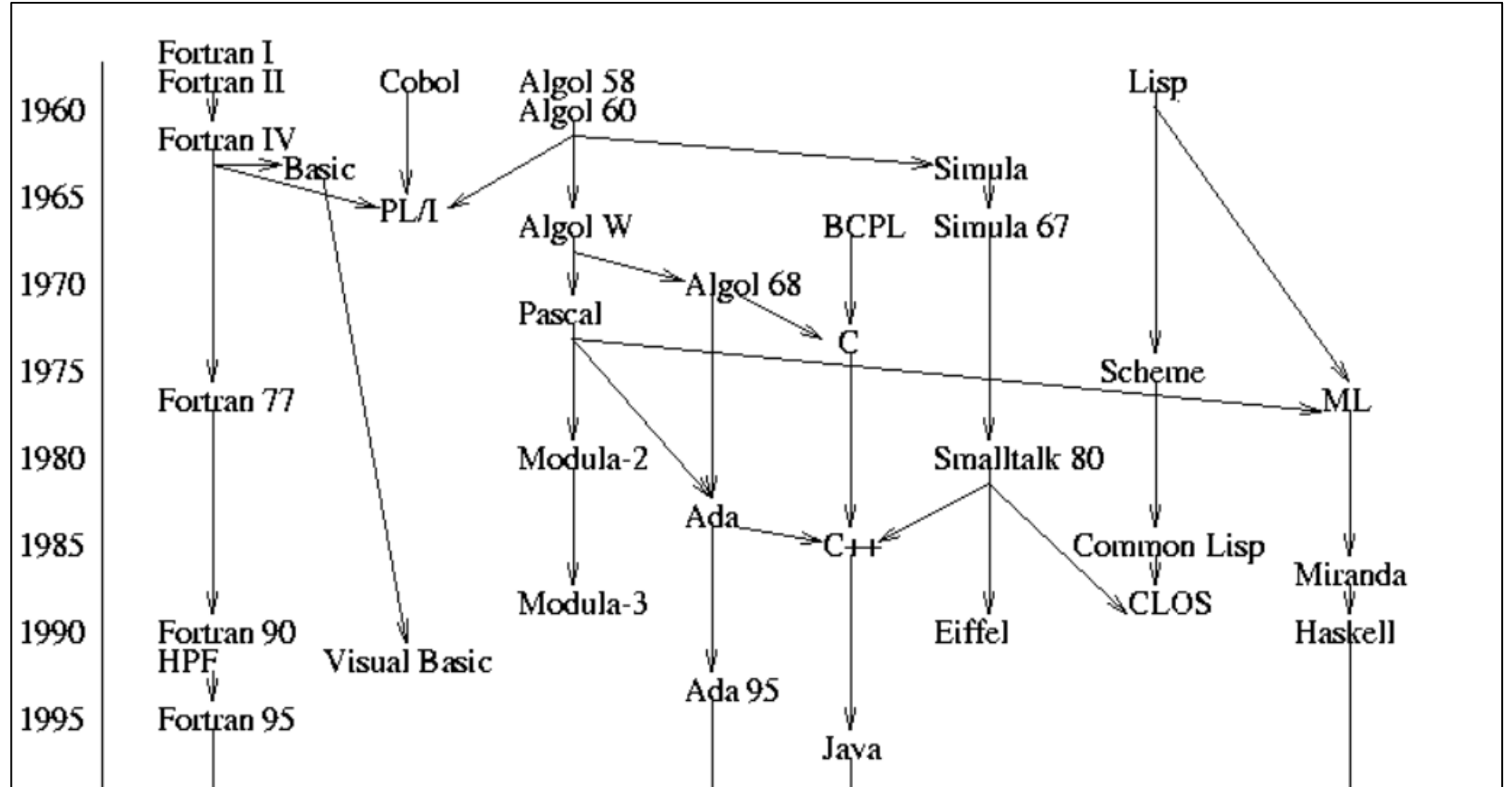
- History of Programming language
- PL features
- What makes a good PL?
- Why study PLs?

References

- Michael L.Scott, **Programming Language pragmatic**, chapter 1
- Robert W. Sebesta, **Concept of Programming Languages**, chapter 1
- Le Minh Duc, **Object Oriented Program Development**, chapter 1

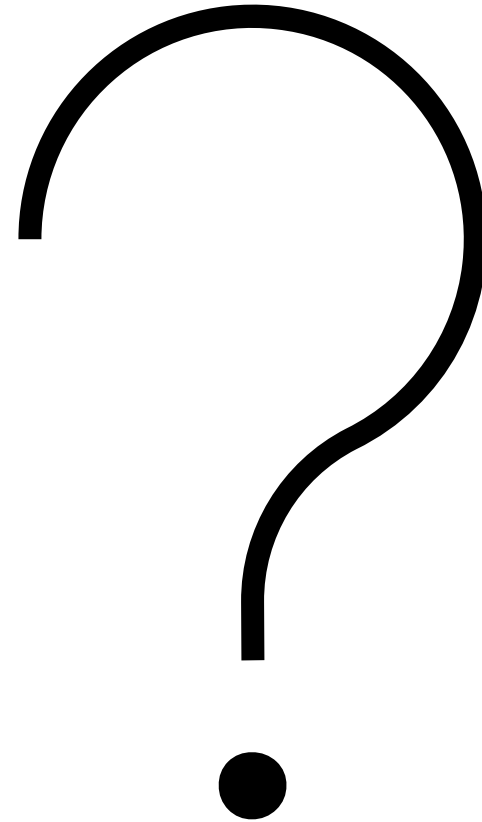
History of Programming language

- Thousands of Programming languages have been created



What is a Programming Language?

- Fortran
- Java
- C/C++
- Lisp
- JS
- HTML
- Kotlin
- Cobol
- PHP
- Algol 60
- Basic
- C#
- Pascal,...



Why so many PLs?

- **Evolution:** constantly finding better ways to do things
 - 1960-70: goto-based languages (Fortran, Cobol, Basic) → loops, case (switch) statements (higher level constructs)
 - late 1980s: nested block structure languages (Algol, Pascal, and Ada) → object-oriented languages (Smalltalk, C++, Eiffel, etc.)
- **Special purposes:**
 - designed for solving specific problem domain(s)
- **Personal preference:**
 - different programmers like different things

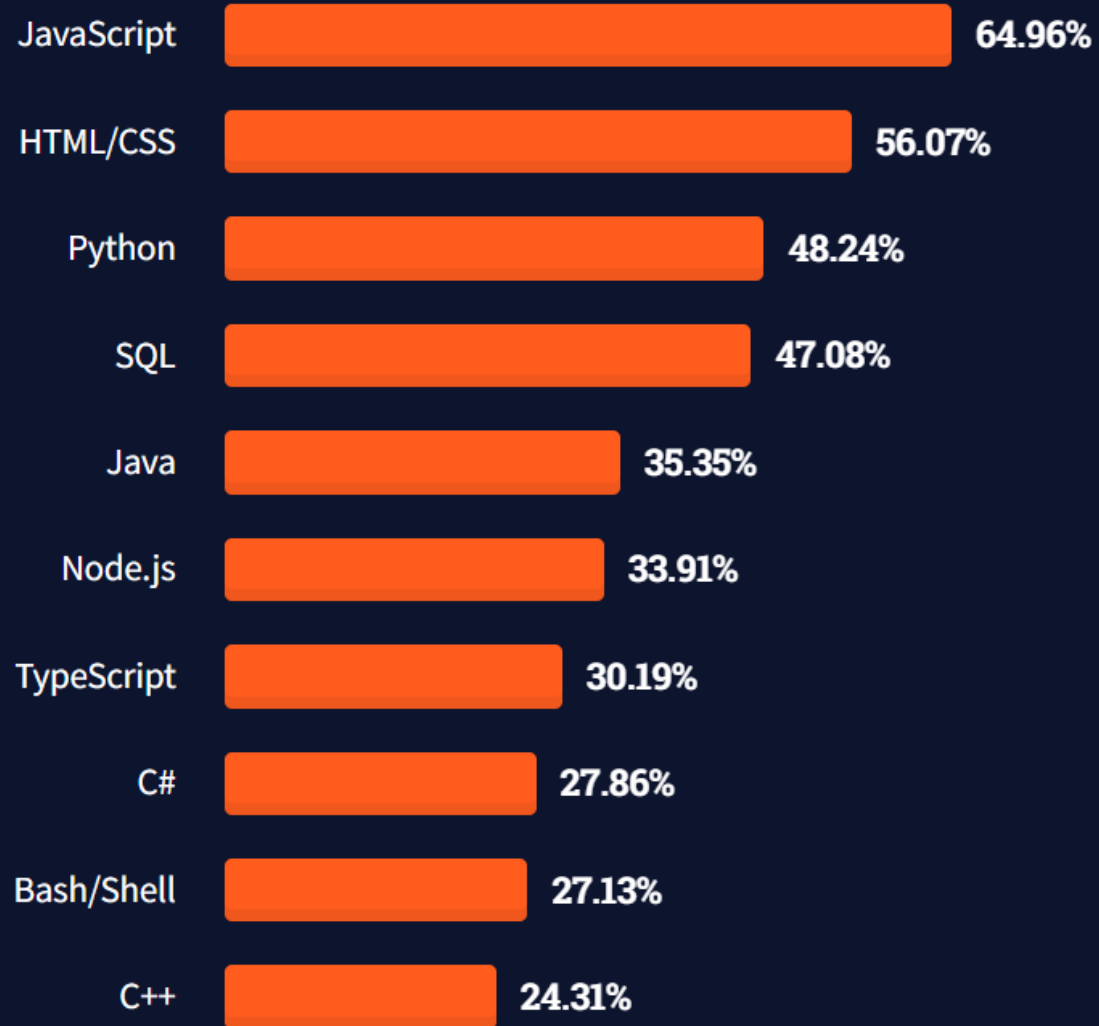
Programming Domains

- Scientific Applications:
 - Fortran, ALGOL 60, ...
- Business Applications:
 - COBOL
- Artificial Intelligence:
 - Lisp, Prolog, ...
- Web Software:
 - HTML, Js, Php, ...

What makes languages evolve?

- Changes in hardware or implementation platform
- Changes in attitudes to safety and risk
- New ideas from academic or industry

Most popular PLs Survey (2021)



Classification of Programming Languages

- **Declarative language**

- A type of programming that specifies what to do.
- E.g. SQL statements such as “select * from table” tell a program to get information from a database, but not how to do so

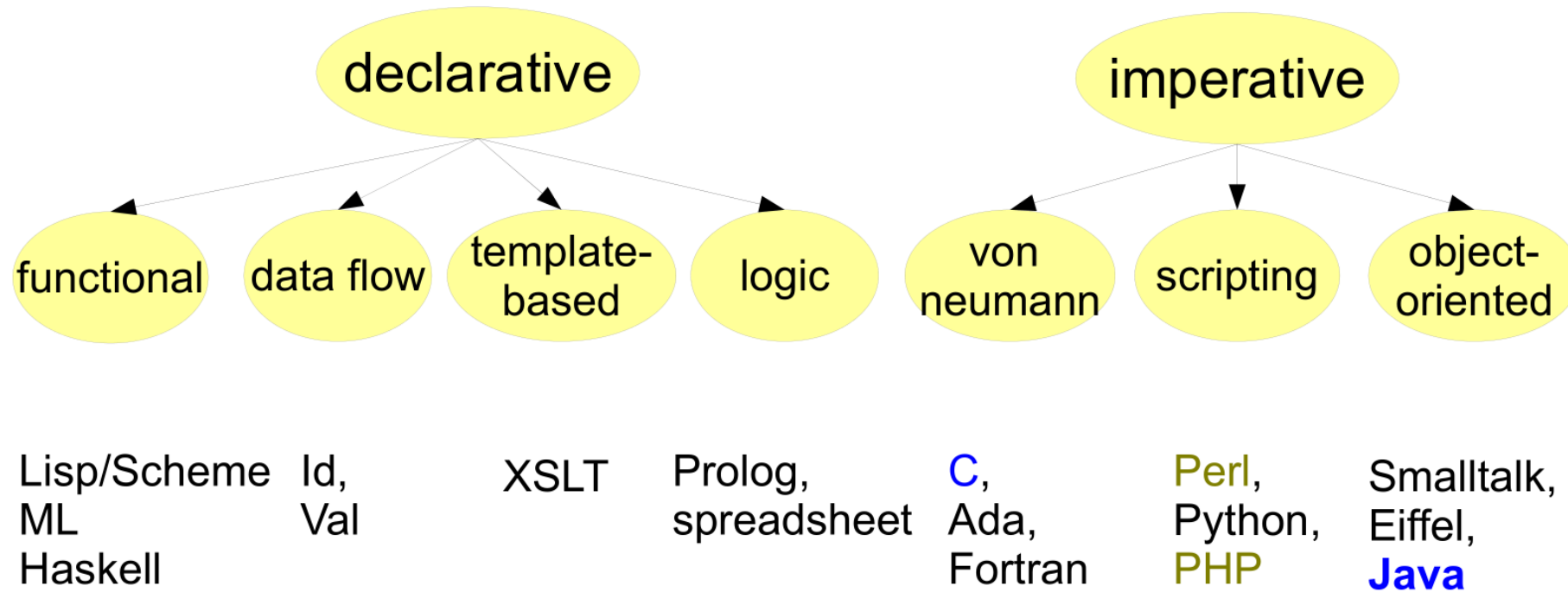
- **Imperative language**

- A type of programming that specifies how to do.
- E.g. when we want to calculate “triple x”, imperatively we would have “ $x = x * 3$ ” or “ $x = x + x + x$ ”

Declarative language vs Imperative language

Declarative programming is like describing your problem to a mathematician. Imperative programming is like giving instructions to an idiot.

PL detail classification

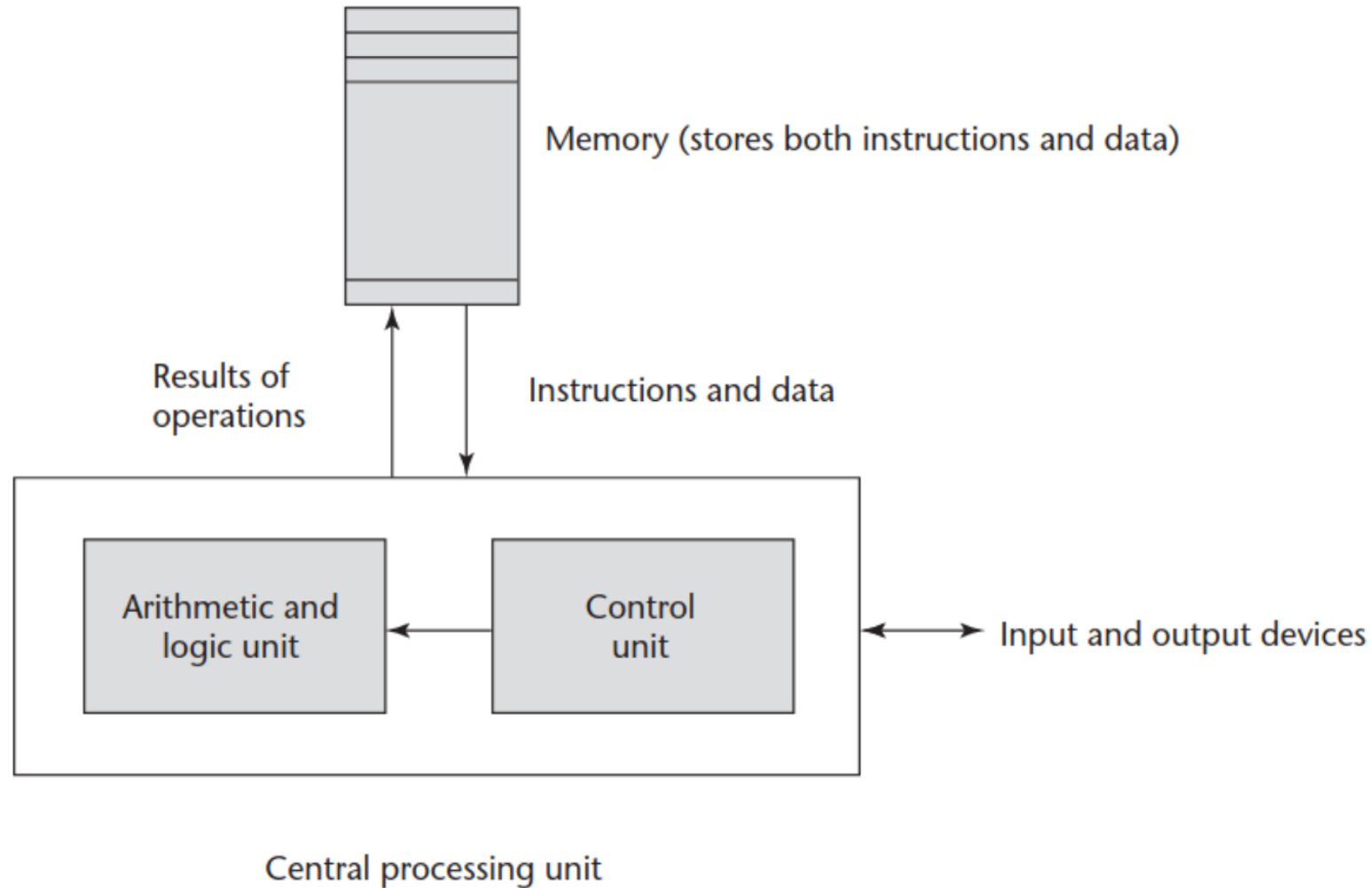


based on model of computation

Von Neumann PLs

- The (once) most successful PL type:
 - Past: Fortran, Ada 83, C
 - Present: Java, Python, JavaScript
- Important subtypes
 - Scripting PL (e.g. TypeScript/Javascript): high-level expressions over a library of components
 - OOPL (e.g. Java, C#): objects and their interactions
- Based on the Von Neumann computer architecture:
 - data (run-time memory) + processing (CPU)
- Declaration and modification of variables

von Neumann computer architecture



Example: Gcd in C

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

Example: Gcd in Java

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


PL features

- Expressive power
 - write clear, concise & maintainable code
 - abstraction
- Ease of use
 - low learning curve for beginning programmers
- Ease of implementation
 - suitable for average machines
 - accessible (e.g. free for educational use)

PL features

- Standardization
 - following standards
 - ensure the portability of code across platforms
- Open source
 - at least one open-source compiler / interpreter of the language
- Excellent compilers
 - fast code generation
 - other support tools to help manage large projects

PL features

- Economics, Patronage, and Inertia
 - backing of powerful sponsor(s)/communities
 - e.g. C# backed by Microsoft, Java backed by Sun/Oracle
 - Python, PHP, JavaScript, Node.js... backed by communities

A good PL is a compromise...

- Between the desires of programmer and PL implementor
- Programmer: PL user
 - language = means of expressing algorithms
 - main concern: conceptual clarity
- PL implementor: PL designer/creator
 - language = means of instructing computers
 - main concern: implementation efficiency
- To achieve both concerns require making compromises!

Why study principles of PLs?

- Know the special features
- Understand different ways of expressing
- Able to choose a PL for a given task
- Make it easier to learn new PLs
- Use language tools to speed up development process (debuggers, assemblers, linkers)
- Add new useful features to a language
- Make better use of language technology
 - e.g. to write better programs

Summary

- Many PLs exist because of *evolution, purpose & preference*
- PL spectrum includes a categorisation of PLs into *declarative* and *imperative* forms
- A good PL strikes a balance between conceptual *clarity* and implementation *efficiency*
- Study PLs help raise the awareness of and choose suitable PL(s) for a given task