Summer Bootcamp 2021

Introduction to Computer Science

Lecture 2

Classification of Programming Languages

Artem Burmyakov

August 03, 2021

INNOPOLIS UNIVERSITY

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

PYPL: PopularitY of Programming Language

(based on how frequently language tutorials are searched on Google)

Rank	Change	Language	Share	Trend	Rank	Change	Language	Share	Trend
1		Python	31.59 %	+3.3 %	15	↓	VBA	1.19 %	-0.1 %
2		Java	16.9 %	-2.7 %	16		Scala	0.97 %	-0.1 %
3		Javascript	8.17 %	+0.0 %	17	↑	Rust	0.91 %	+0.3 %
4		C#	6.54 %	-0.7 %	18	↓	Visual Basic	0.82 %	-0.2 %
5	↑	C/C++	5.88 %	+0.1 %	19	↑↑↑↑↑	Dart	0.57 %	+0.2 %
6	↓	PHP	5.78 %	-0.7 %	20	↑↑↑	Ada	0.54 %	+0.2 %
7		R	4.18 %	+0.3 %	21	↑	Lua	0.52 %	+0.1 %
8		Objective-C	2.6 %	-0.0 %	22	↓↓↓	Perl	0.45 %	-0.1 %
9		Swift	2.35 %	-0.0 %	23	↓↓↓	Abap	0.44 %	-0.1 %
10	↑	TypeScript	1.94 %	+0.2 %	24	↑↑↑	Julia	0.43 %	+0.2 %
11	↓	Matlab	1.63 %	-0.2 %	25		Cobol	0.42 %	+0.1 %
12		Kotlin	1.57 %	+0.1 %	26	↓↓↓↓↓	Groovy	0.41 %	-0.1 %
13	↑↑	Go	1.39 %	+0.2 %	27	↓	Haskell	0.32 %	+0.0 %
14	↓	Ruby	1.22 %	-0.2 %	28		Delphi	0.28 %	+0.0 %

Source: http://pypl.github.io/PYPL.html, year 2020

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

- Focus on runtime performance;
- An “aging champion”;
- A diverse range of libraries is available

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

- Attempt to replace C++;
- Widely used for Google products;
- For now, usage is limited to web-backends

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

The support of purely functional programming paradigm

(still different from C)

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

For database manipulations

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Hardware Description Language (HDL), to design electrical circuits

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Languages differ in various aspects:

- design purpose
- runtime performance
- memory efficiency
- simplicity of learning
- other

Some Programming Languages

Ada	F#	Oberton	Visual Basic
ALGOL	Fortran	Objective-C	Latex
BASIC	Go	Pascal	SQL
Boo	Harbour	Perl	Matlab
C	Haskell	PHP	R
C++	Idris	Pike	Verilong
C#	Java	PL/I	VHDL
Clojure	JavaScript	Python	DOT
COBOL	Julia	RPG	Make
Crystal	Kotlin	Ruby	HTML
D	Lisp	Rust	PostScript
Dart	Lua	Scala	Standard ML
Eiffel	Modula-2	Simula	Bash
Elixir	Nim	Swift	ABAP
Erlang	NPL	Prolog	…

Languages differ in various aspects:

- design purpose
- runtime performance
- memory efficiency
- simplicity of learning
- other

More details on classification are discussed later

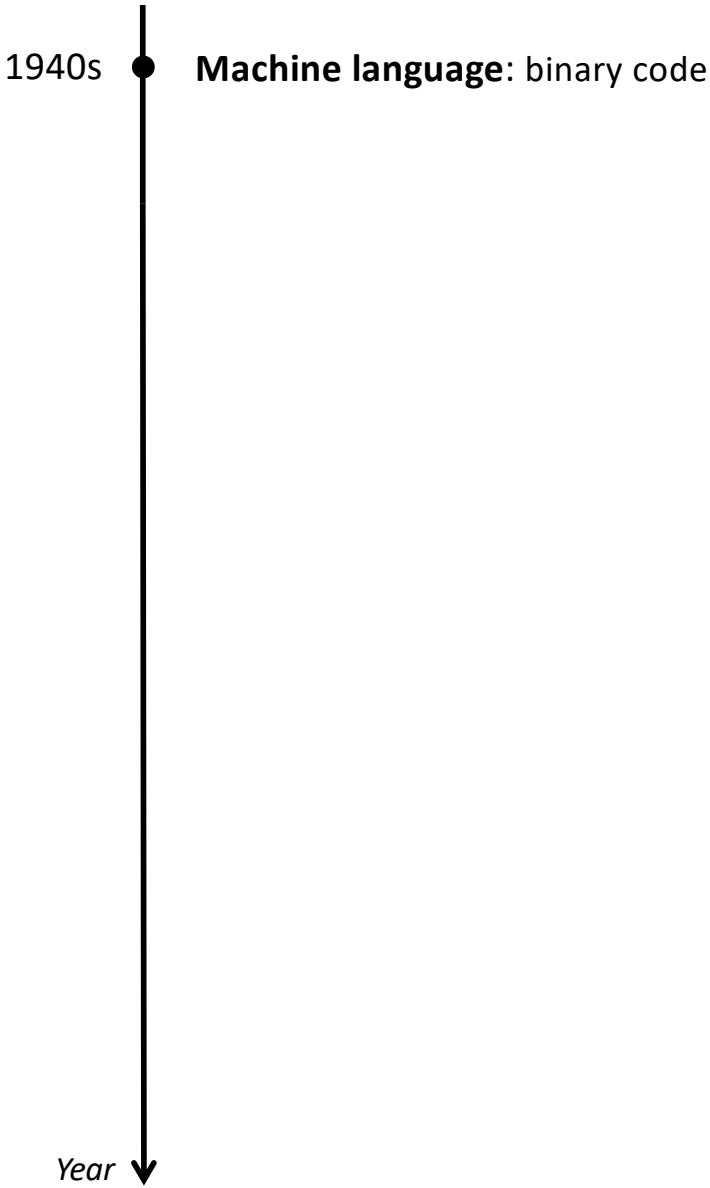
The Most Popular Development Tools

(based on the search frequency of a tool download page in Google)

Rank	Change	IDE	Share	Trend
1	↑	Visual Studio	25.29 %	+3.4 %
2	↑	Eclipse	17.36 %	-1.5 %
3	↓↓	Android Studio	12.91 %	-10.1 %
4	↑	Visual Studio Code	7.93 %	+3.3 %
5	↑↑	pyCharm	7.17 %	+2.5 %
6		IntelliJ	5.64 %	+1.0 %
7	↓↓↓	NetBeans	5.42 %	-0.4 %
8		Xcode	4.1 %	+0.5 %
9		Sublime Text	3.75 %	+0.1 %
10		Atom	3.7 %	+0.6 %
11		Code::Blocks	1.78 %	+0.3 %
12		Vim	0.93 %	+0.0 %
13	↑	PhpStorm	0.68 %	+0.0 %
14	↓	Xamarin	0.65 %	-0.1 %
15		Komodo	0.45 %	+0.0 %
16		Qt Creator	0.4 %	+0.1 %
17	↑↑	Emacs	0.28 %	+0.1 %
18		geany	0.27 %	+0.1 %
19	↓↓	JDeveloper	0.23 %	-0.0 %
20	↑	MonoDevelop	0.16 %	+0.0 %

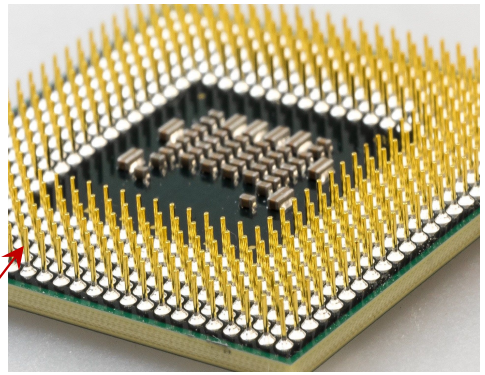
Source: http://pypl.github.io/IDE.html, year 2020

Evolution of Programming Languages



Evolution of Programming Languages

1940s — **Machine language**: binary code



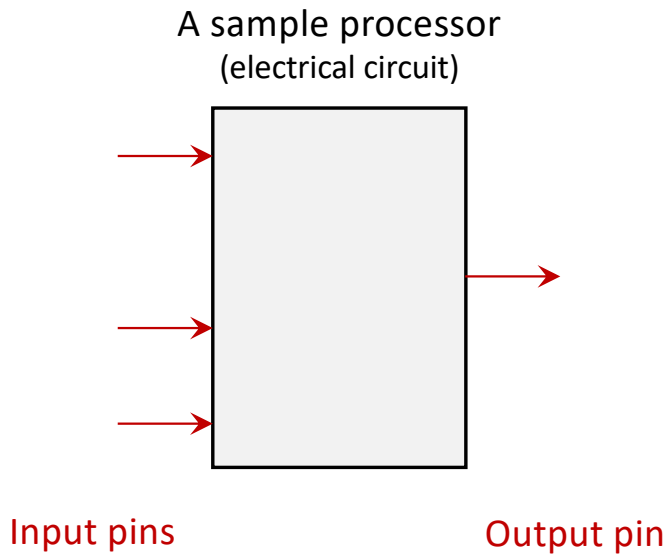
Every processor has input and output pins

Year ↓

Images are taken from https://i.pinimg.com/

Evolution of Programming Languages

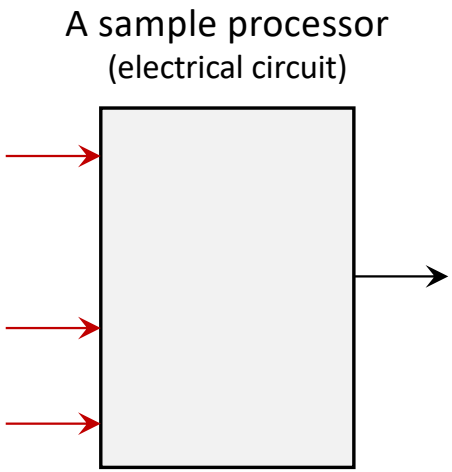
1940s — **Machine language**: binary code



Year ↓

Evolution of Programming Languages

1940s — **Machine language**: binary code



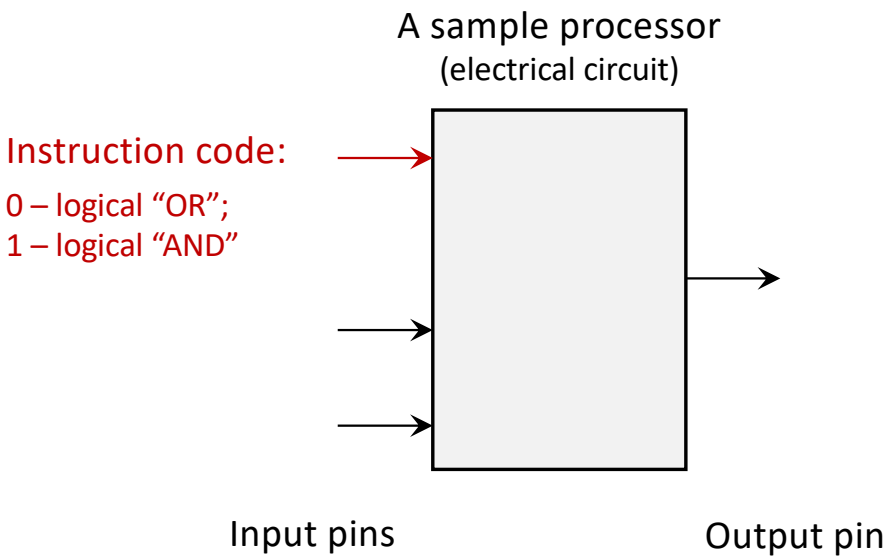
Input pins:
- Input arguments
- Instruction code

Output pin

Year ↓

Evolution of Programming Languages

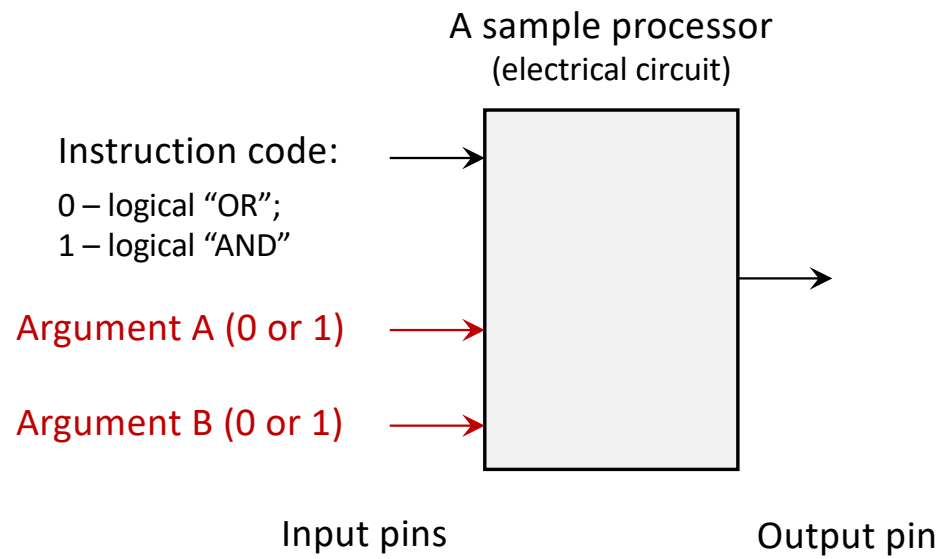
1940s — **Machine language**: binary code



Year ↓

Evolution of Programming Languages

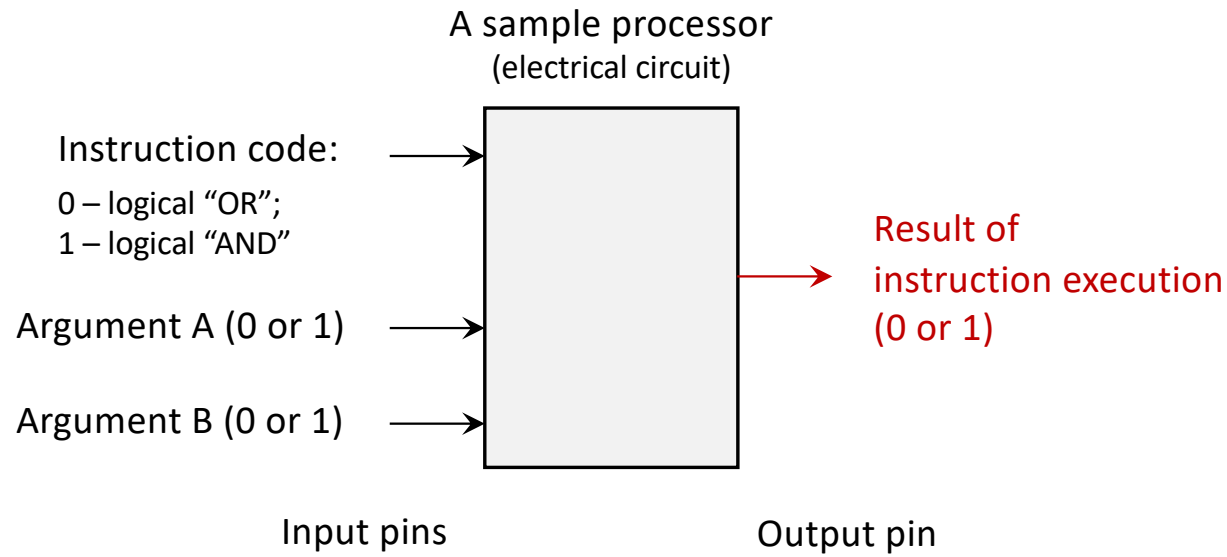
1940s — **Machine language**: binary code



Year ↓

Evolution of Programming Languages

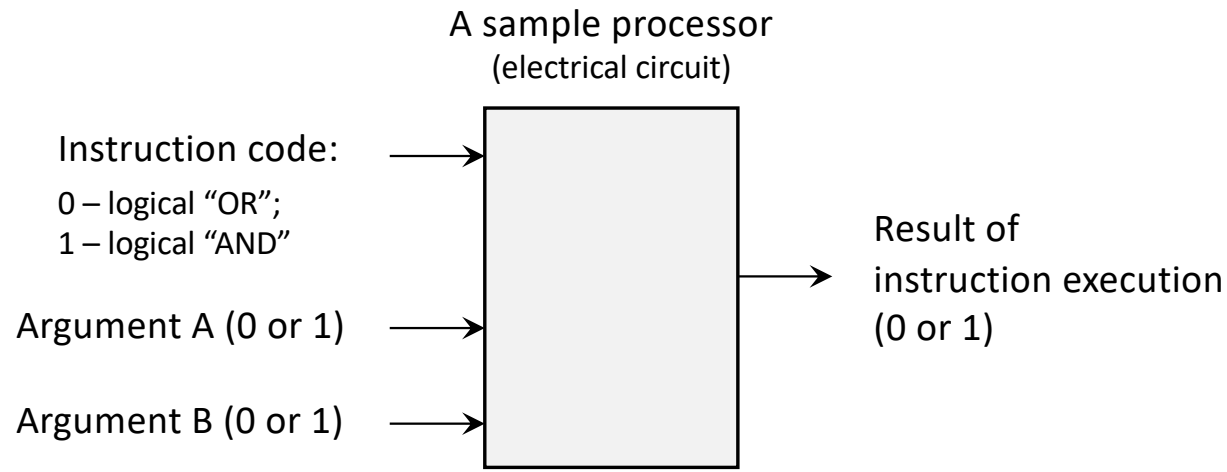
1940s — **Machine language**: binary code



Year ↓

Evolution of Programming Languages

1940s — **Machine language**: binary code



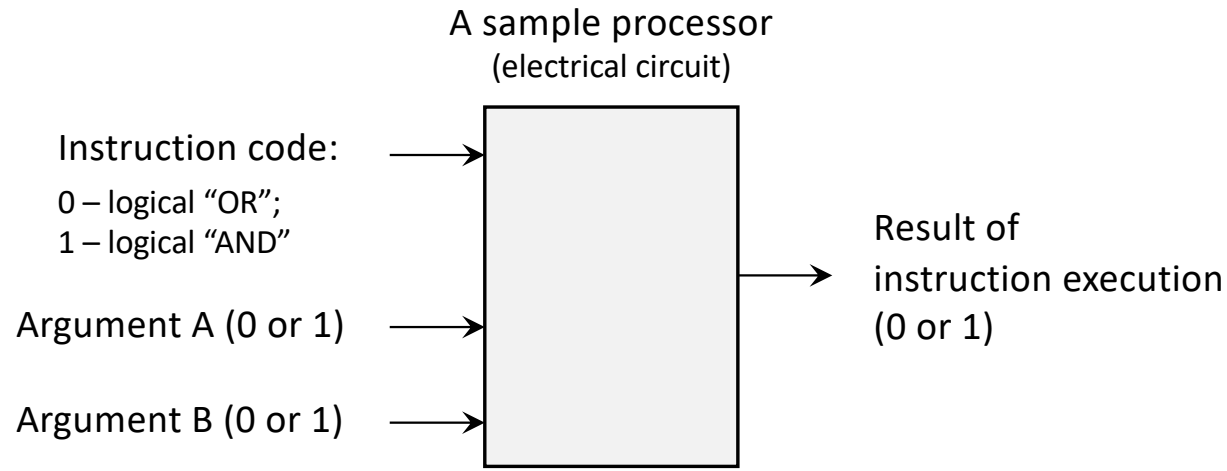
Machine instruction – string of *0s* and *1s*:

<Operation code> <Value A> <Value B>

Year ↓

Evolution of Programming Languages

1940s — **Machine language**: binary code



Machine instruction – string of *0s* and *1s*:

<Operation code> <Value A> <Value B>

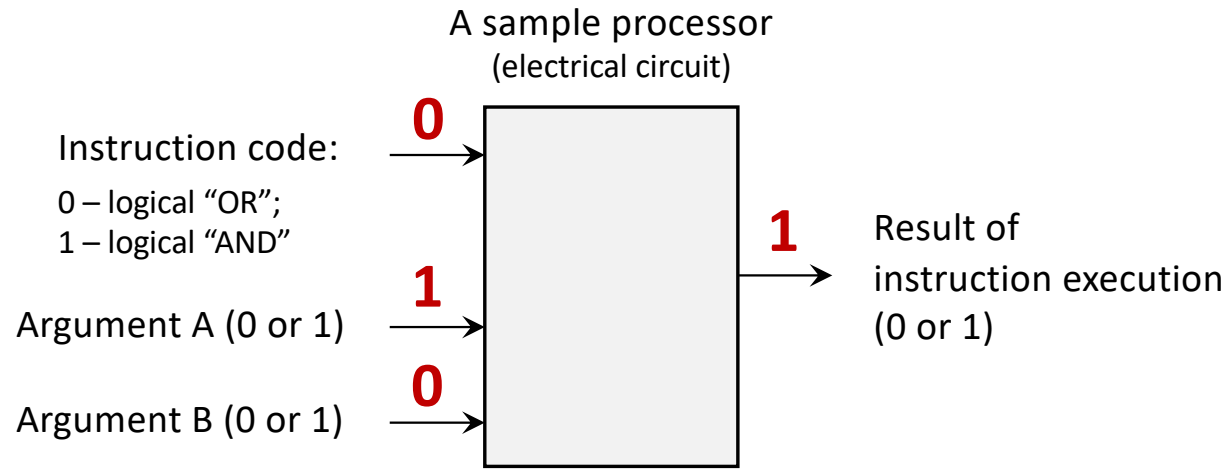
Example: **0 1 0**

Instr. code — Value A — Value B

Year ↓

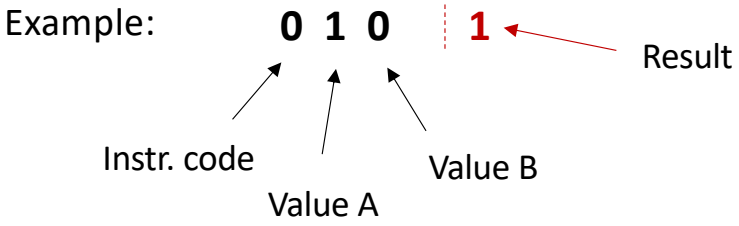
Evolution of Programming Languages

1940s — **Machine language**: binary code



Machine instruction – string of *0s* and *1s*:

<Operation code> <Value A> <Value B>



Year

Evolution of Programming Languages

1940s **Machine language**: binary code

1950s **Assembler (or asm)**:
Introduced English mnemonics, to increase code readability

Machine code:	Assembly language:	
0 1 0	**OR**	1 0
1 1 0	**AND**	1 0

Mnemonics (English abbreviations for computer instructions)

Evolution of Programming Languages

1940s — **Machine language**: binary code

1950s — **Assembler (or asm)**:
Introduced English mnemonics, to increase code readability

⋮

1954 IBM — **Fortran** (**FOR**mula **TRAN**slation):
the first widely available high-level language, designed for time-intensive numeric computations

Originally met with skepticism, due to a lower performance than for Assembler;
Later became widely adopted and appreciated;
Still widely used in high-performance computing, to program the most powerful supercomputers

Evolution of Programming Languages

1940s — **Machine language**: binary code

1950s — **Assembler (or asm)**:
Introduced English mnemonics, to increase code readability

⋮

1954 IBM — **Fortran** (**FOR**mula **TRAN**slation):
the first widely available high-level language, designed for time-intensive numeric computations

~1960 MIT — **LISP** (**LIS**t **P**rocessing):
the first to support recursive function calls, linked lists, tree data structures

~1960 ETH — **ALGOL** (**ALGO**rithmic **L**anguage):
Focused on implementing advanced algorithms (introduced begin-end code blocks, nested functions, etc.);
Heavily influenced all following languages

Evolution of Programming Languages

1940s — **Machine language**: binary code

1950s — **Assembler (or asm)**:
Introduced English mnemonics, to increase code readability

1954 IBM — **Fortran** (**FOR**mula **TRAN**slation):
the first widely available high-level language, designed for time-intensive numeric computations

~1960 MIT — **LISP** (**LIS**t **P**rocessing):
the first to support recursive function calls, linked lists, tree data structures

~1960 ETH — **ALGOL** (**ALGO**rithmic **L**anguage):
Focused on implementing advanced algorithms (introduced begin-end code blocks, nested functions, etc.);
Heavily influenced all following languages

1972 Bell Labs — **C**:
for system programming (user applications – to services the end user; system programming – to service user applications)

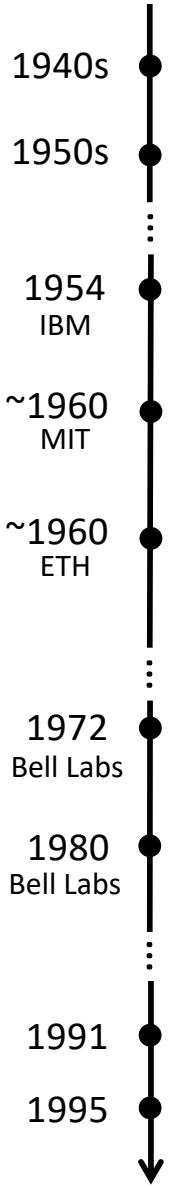
1980 Bell Labs — **C++** (original name "C with classes"):
Combination of system programming and object-oriented programming

Ada, Matlab, Eiffel,…

1991 — **Python**: focus on code readability, simplicity of learning, quick and easy prototyping of algorithms

1995 — **Java**: focus on portability (ability of the same program to run on different computer architectures)

Evolution of Programming Languages



1940s — **Machine language**: binary code

1950s — **Assembler (or asm)**:
Introduced English mnemonics, to increase code readability

1954 IBM — **Fortran** (**FOR**mula **TRAN**slation):
the first widely available high-level language, designed for time-intensive numeric computations

~1960 MIT — **LISP** (**LIS**t **P**rocessing):
the first to support recursive function calls, linked lists, tree data structures

~1960 ETH — **ALGOL** (**ALGO**rithmic **L**anguage):
Focused on implementing advanced algorithms (introduced begin-end code blocks, nested functions, etc.);
Heavily influenced all following languages

1972 Bell Labs — **C**:
for system programming (user applications – to services the end user; system programming – to service user applications)

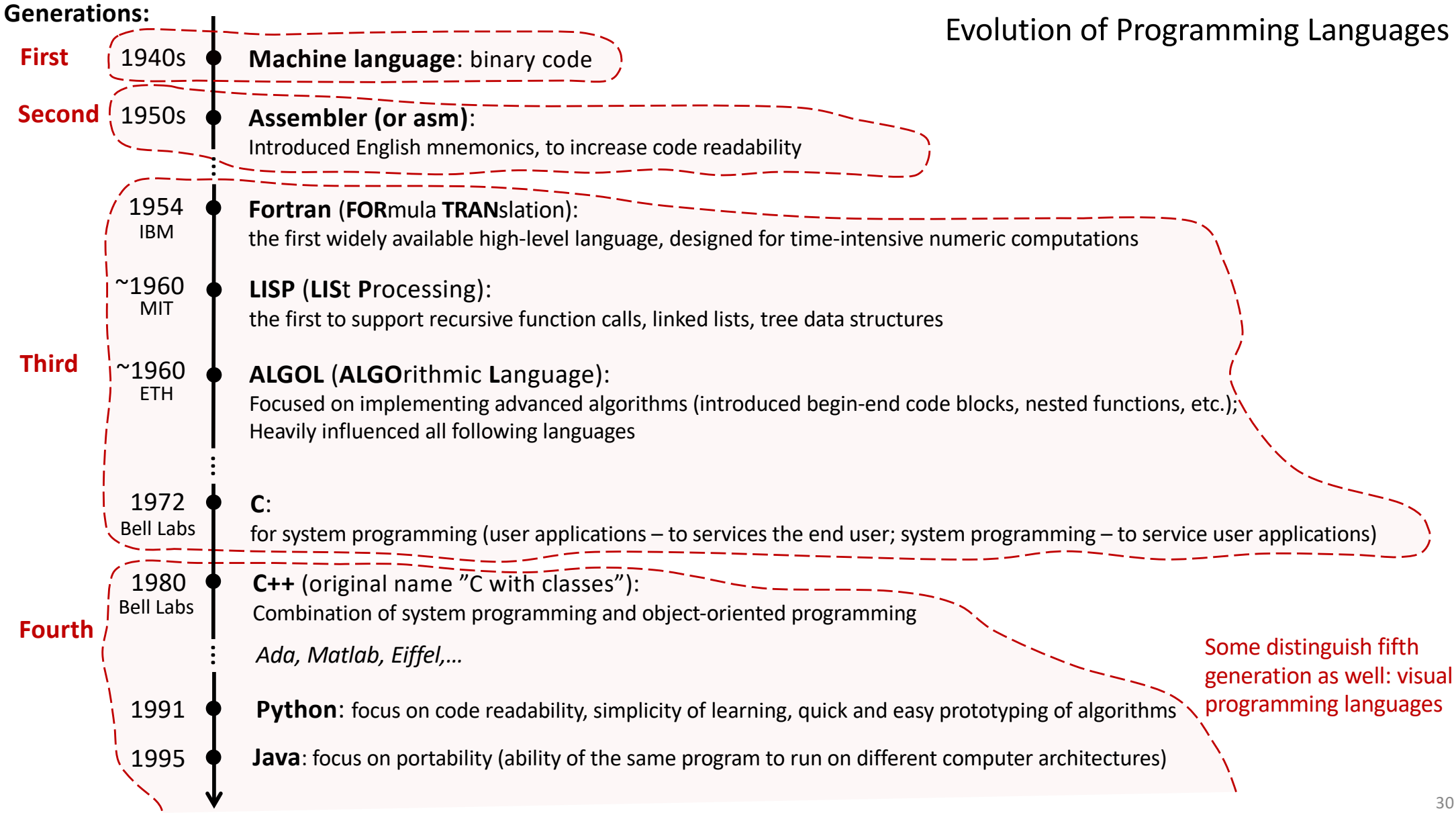
1980 Bell Labs — **C++** (original name "C with classes"):
Combination of system programming and object-oriented programming

Ada, Matlab, Eiffel,…

1991 — **Python**: focus on code readability, simplicity of learning, quick and easy prototyping of algorithms

1995 — **Java**: focus on portability (ability of the same program to run on different computer architectures)

Evolution of Programming Languages



Family Tree of Fortran, Algol, and COBOL

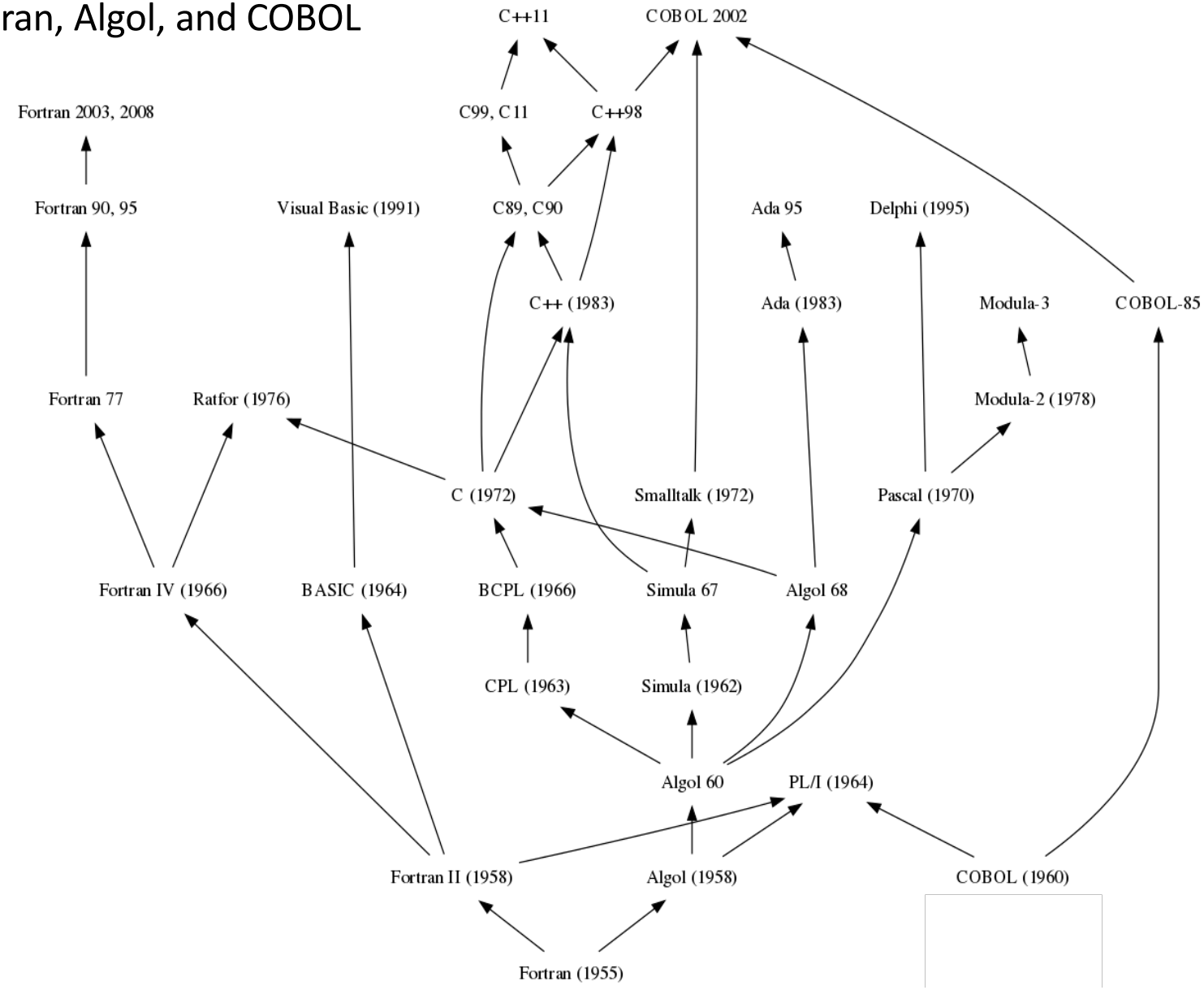
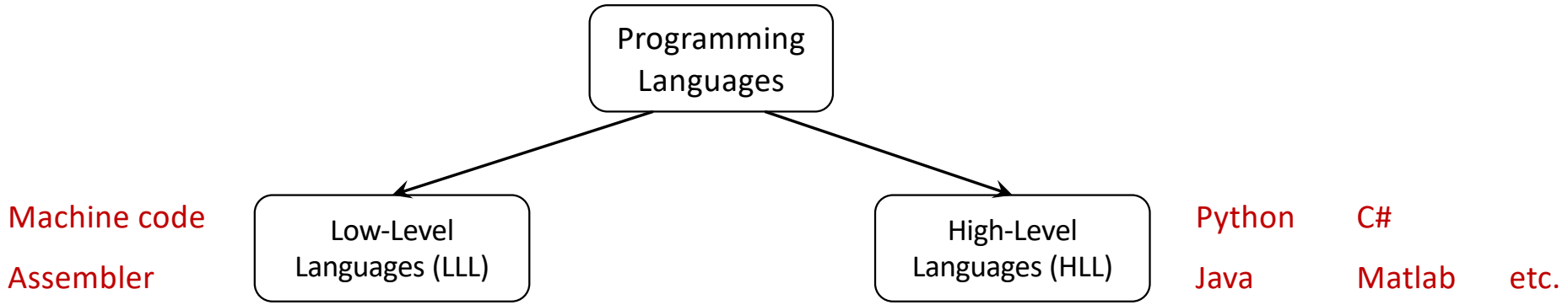
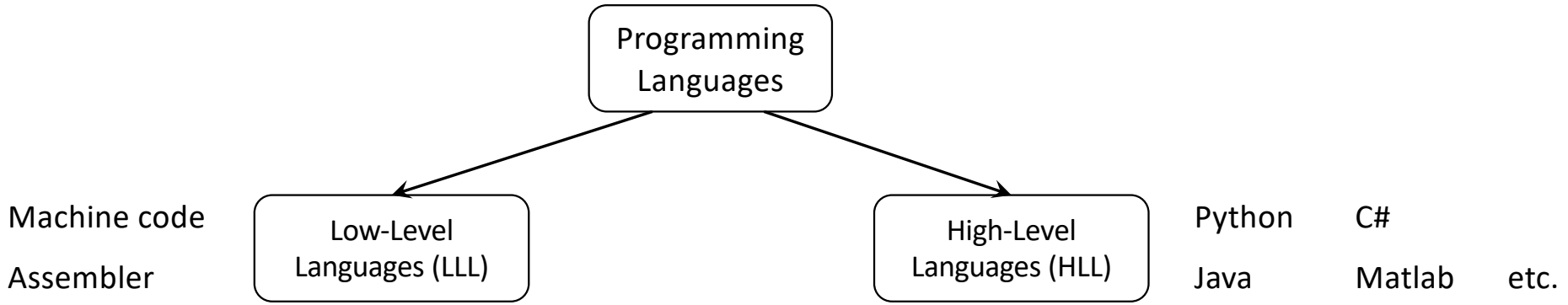


Image taken from: wikipedia.com

High-Level and Low-Level Programming Languages
(based on hardware abstraction level)



High-Level and Low-Level Programming Languages (based on hardware abstraction level)



Advantages:

+ Fast and memory-efficient;
+ Direct access to entire functionality of processor and memory;
+ Does not require compiler or interpreter;

Disadvantages:

- Programs are machine-dependent and not portable;
- Require a good knowledge of computer architecture;
- Slow and complex development process;
- Not human-friendly, error prone

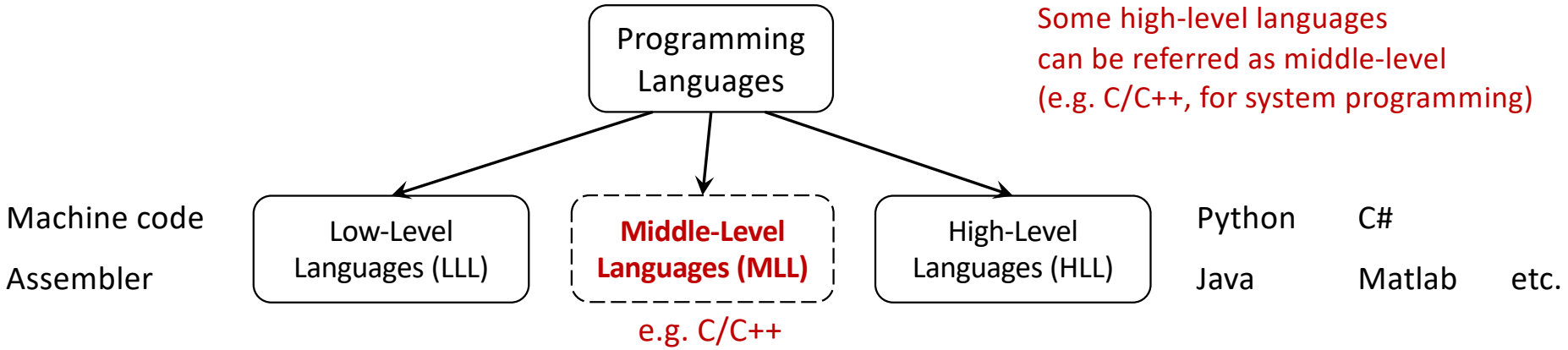
Advantages:

+ Simpler coding (human-friendly syntax);
+ A faster development process;
+ Machine-independent (with a few exceptions)

Disadvantages:

- Programs are slower;
- Lower memory efficient;
- Need to be compiled or interpreted into machine code;
- Cannot communicate directly with hardware

High, Low, and Middle-Level Programming Languages (based on hardware abstraction level)



Advantages:

+ Fast and memory-efficient;
+ Direct access to entire functionality of processor and memory;
+ Does not require compiler or interpreter;

Disadvantages:

- Programs are machine-dependent and not portable;
- Require a good knowledge of computer architecture;
- Slow and complex development process;
- Not human-friendly, error prone

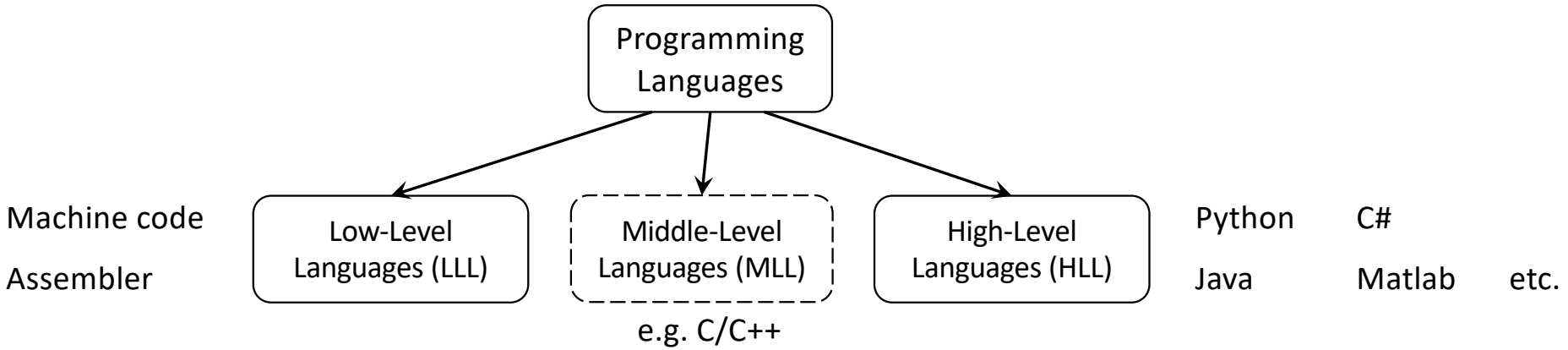
Advantages:

+ Simpler coding (human-friendly syntax);
+ A faster development process;
+ Machine-independent (with a few exceptions)

Disadvantages:

- Programs are slower;
- Lower memory efficient;
- Need to be compiled or interpreted into machine code;
- Cannot communicate directly with hardware

High, Low, and Middle-Level Programming Languages (based on hardware abstraction level)

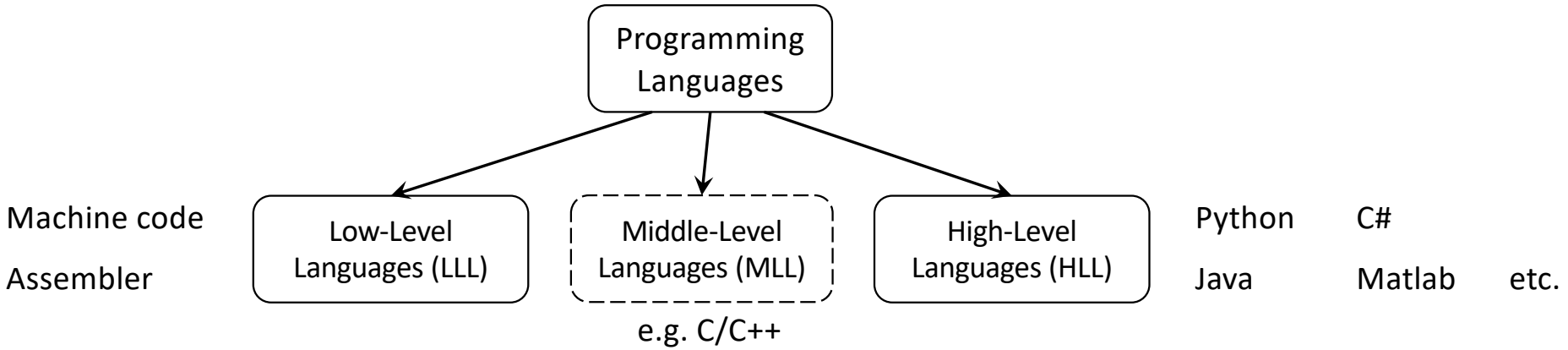


End-user applications:

- web browsers
- games
- office applications
- many other

End-user – (*informally*) the user of a computer, that is not necessarily a professional software developer

High, Low, and Middle-Level Programming Languages (based on hardware abstraction level)



System programming:

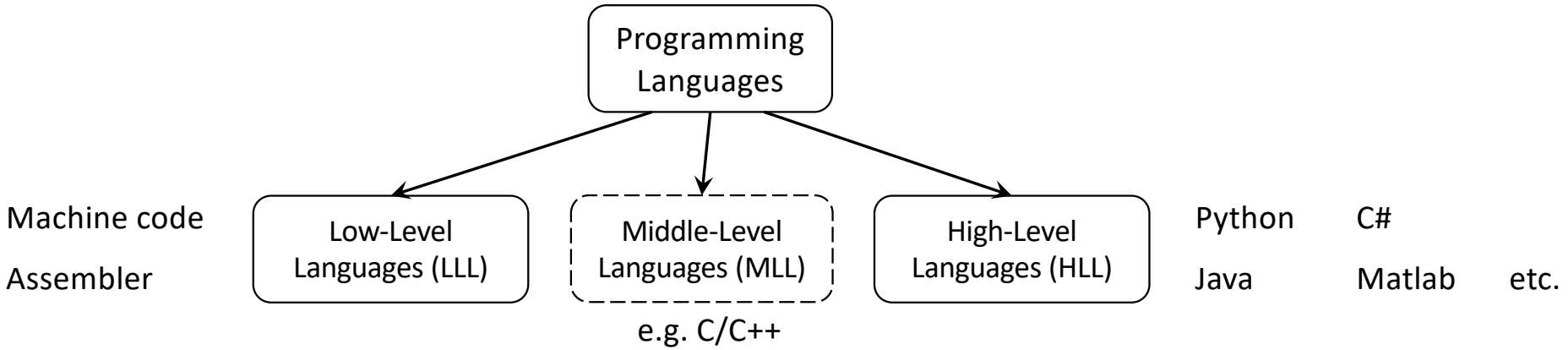
- operating systems
- device drivers
- high performance end-user applications

End-user applications:

- web browsers
- games
- office applications
- many other

End-user – (*informally*) the user of a computer, that is not necessarily a professional software developer

High, Low, and Middle-Level Programming Languages (based on hardware abstraction level)



Typical usage nowadays is limited to device drivers

System programming:

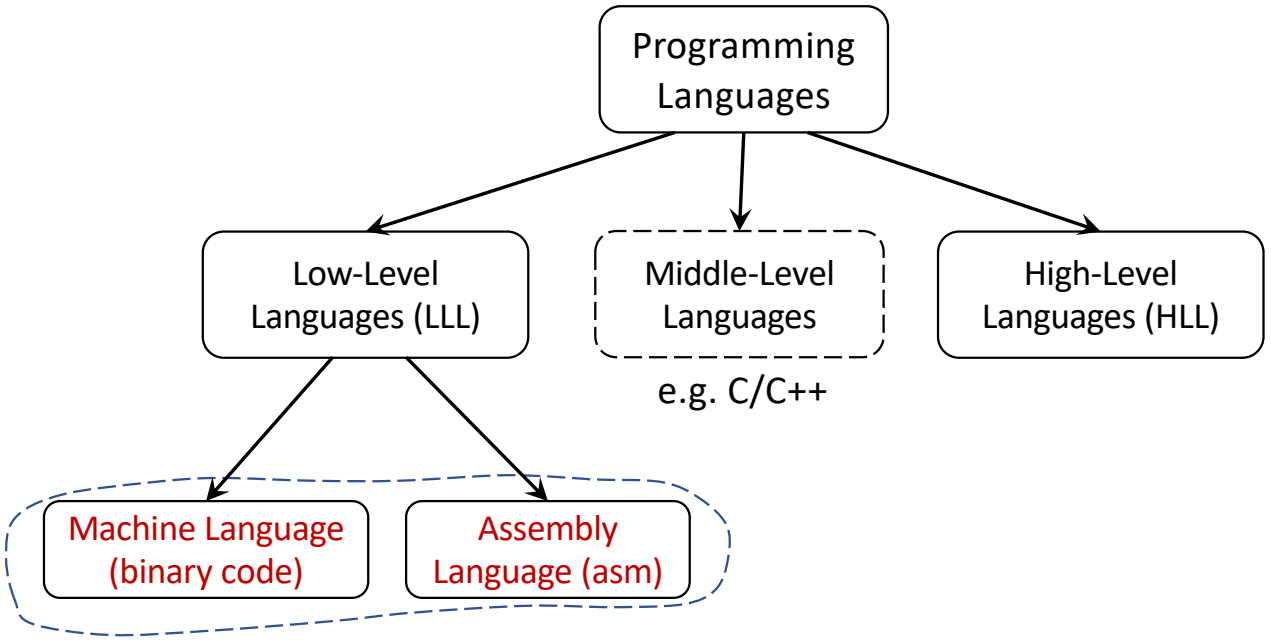
- operating systems
- device drivers
- high performance end-user applications

End-user applications:

- web browsers
- games
- office applications
- many other

End-user – (*informally*) the user of a computer, that is not necessarily a professional software developer

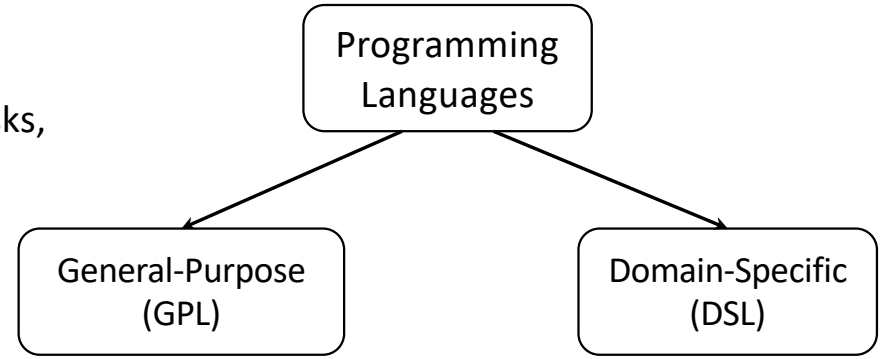
High-Level and Low-Level Programming Languages (based on hardware abstraction level)



Performance is nearly the same for both

Requires assembler – tool to replace mnemonics with corresponding binary codes of instructions

General-Purpose and Domain-Specific Programming Languages



Designed for a wide range of tasks, like C/C++, Java, Python, Fortran

Designed for a specific task, e.g.:

- Shell scripting (Bash, Make);
- Web documents design (HTML);
- Scientific computing (Matlab);
- Hardware design (Verilog, VHDL);
- Database queries (SQL);
- Document layout (Latex)

Advantages:

+ Provide a wide range of functionality, to solve various problems;
+ No need to learn the syntax of additional (e.g. domain-specific) languages

Disadvantages:

- Can be less efficient as compared to a more appropriate domain-specific language (e.g. Matlab is probably better for matrix multiplication than C++);
- Harder and more time consuming to learn

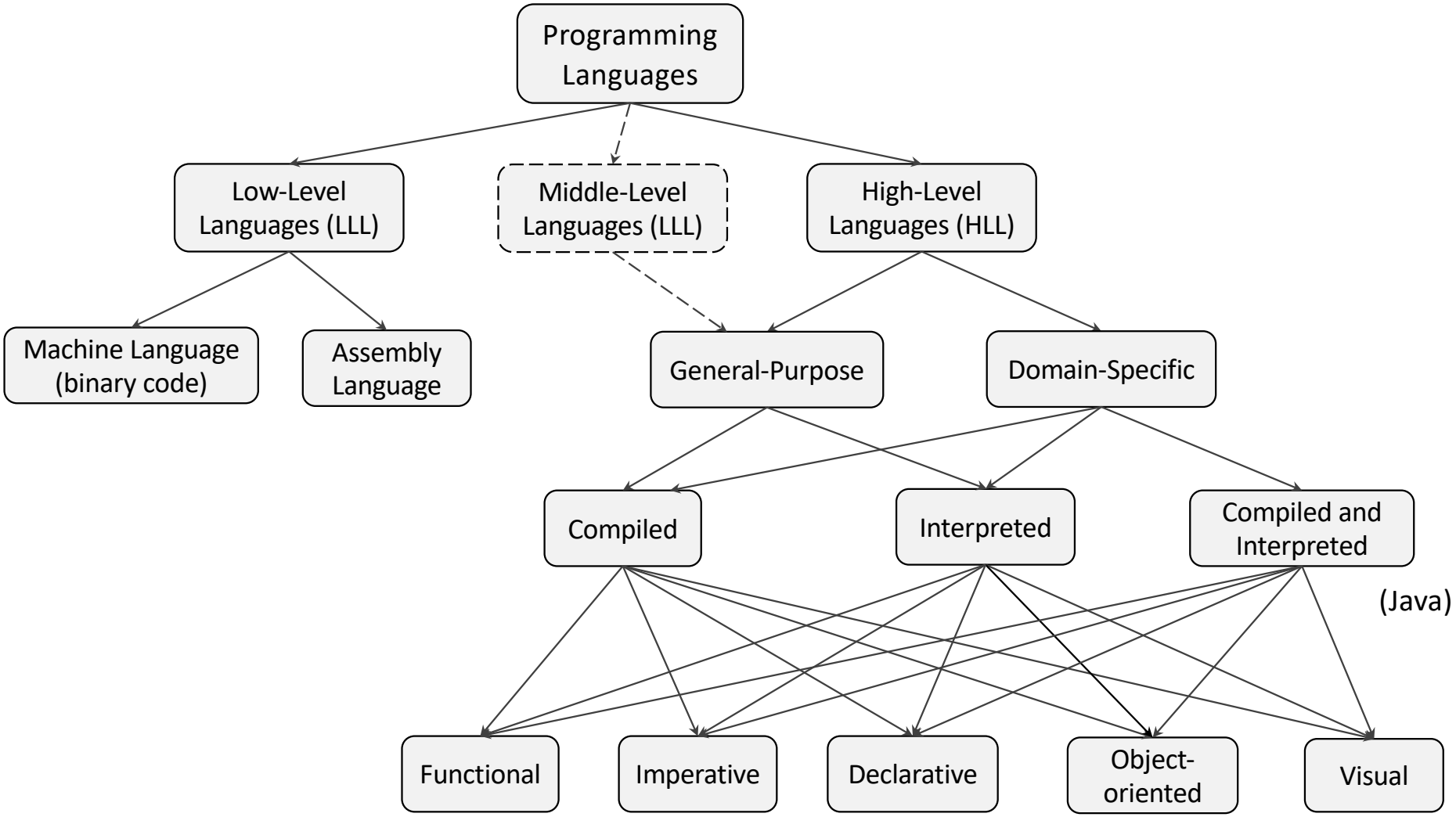
Advantages:

+ Optimized for solving a specific task;
+ Easier to learn, and more convenient for not professional program developers

Disadvantages:

- Can be used only for a limited set of tasks;
- Usually have a very different syntax from GPL

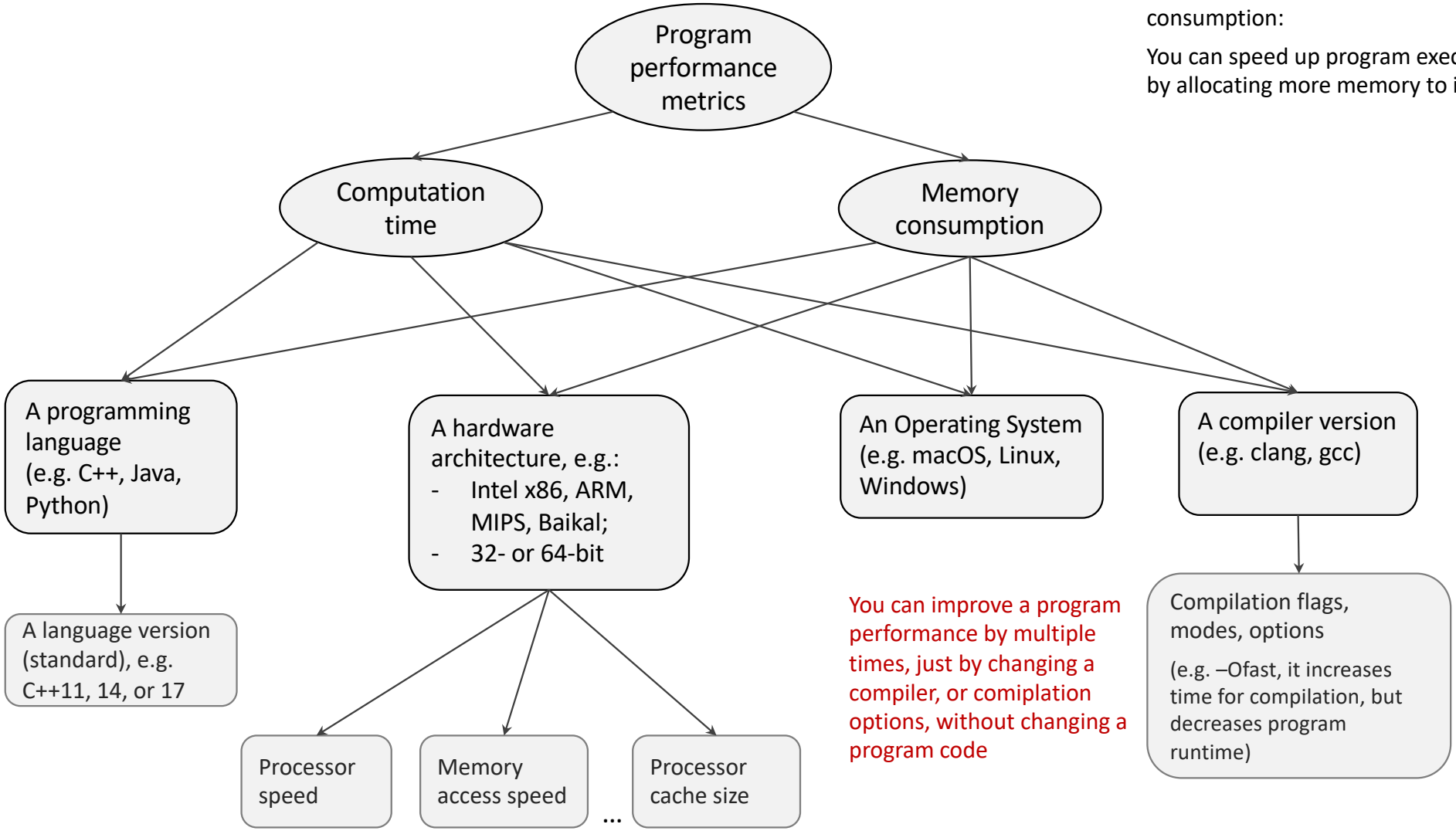
Classification of Programming Languages



Performance Characteristics of a Computer Program

Note: Frequently there is a trade-off between runtime and memory consumption:

You can speed up program execution by allocating more memory to it

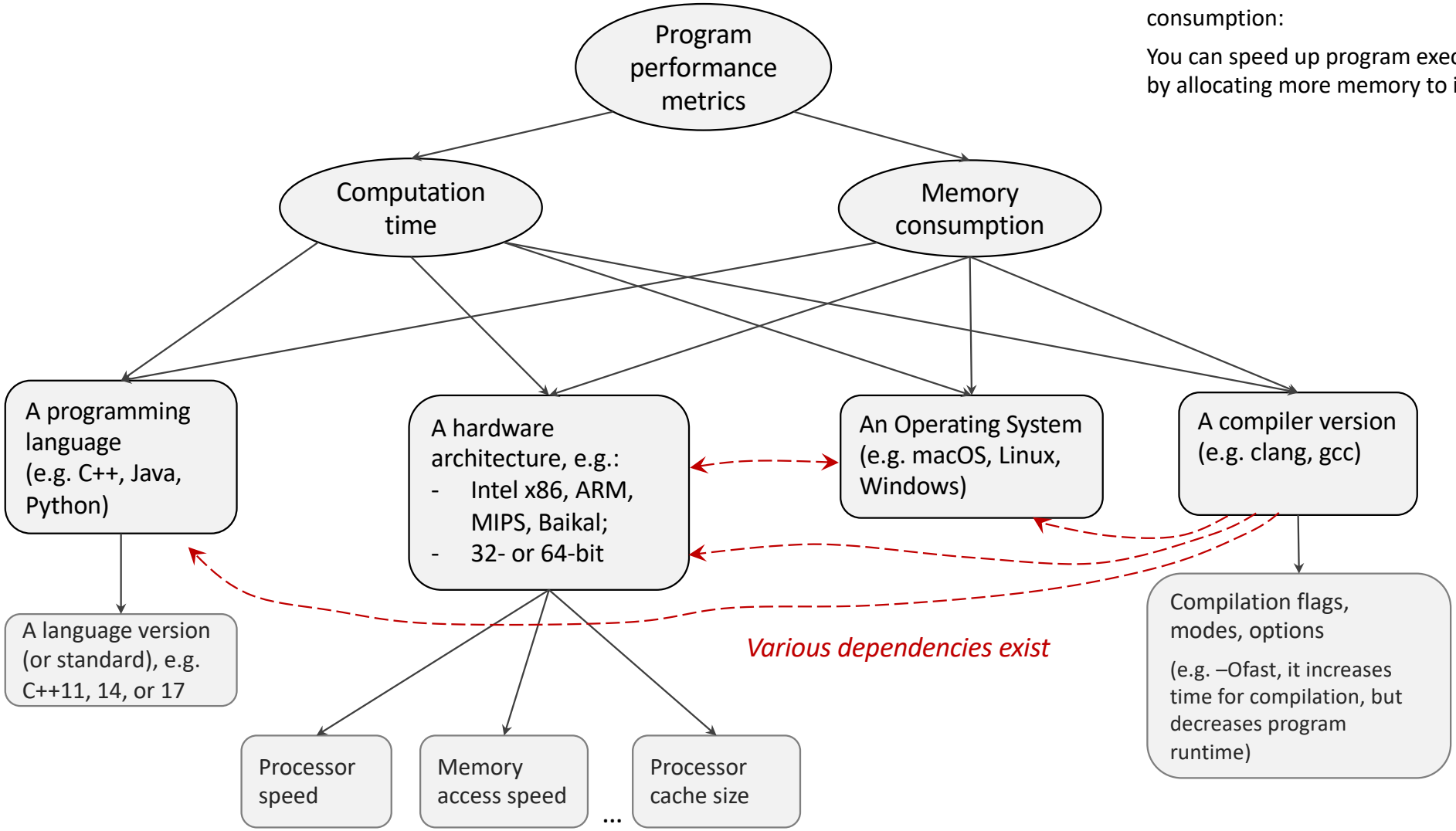


You can improve a program performance by multiple times, just by changing a compiler, or comiplation options, without changing a program code

Performance Characteristics of a Computer Program

Note: Frequently there is a trade-off between runtime and memory consumption:

You can speed up program execution by allocating more memory to it



Comparison of C/C++, Java, and Python

Criteria	C/C++	Java	Python
Primary objective	System programming, operating systems development, device drivers, etc.	Development of user applications, that are highly portable	Quick prototyping, easy learning of programming, usage by non-professional program developers in various areas, such as Data Science, etc.
Portability	High; Need to recompiled for each hardware architecture	Very high (thanks to Java Virtual Machine)	High
Performance (runtime, memory)	Very high	Lower than for C/C++; Comparable to Python	Lower than for C/C++; Comparable to Java
Complexity	Above average	Average	Very simple; human-friendly syntax
Abstraction level from hardware	Middle-level	Likely middle-level	Definitely high-level
Compiled or interpreted?	Compiled	Compiled and interpreted: Program is first compiled into bytecode, and then interpreted by Java Virtual Machine (JVM)	Interpreted
Compilation time	Faster than for Java	Slower than for C++	
Support of object-oriented programming	C++ - yes; C - no	Yes	Yes
Concurrency support	C: poor (and only after 2011); C++: included later, reasonable support of multithreading	Yes; Designed with concurrency in mind	Poor support; Not efficient
Code Length	~1.5 less than for Java	Huge code in size	~3-4 times less than for Java