Ch1. Preliminaries

§ 1.1 Reasons for Studying Concept of Programming Languages

- increased capacity to express ideas;

  ex) thoughts → language → expressing idea to others

- improved background for choosing appropriate languages;
- increased ability to learn new languages;
- better understanding of the significance of implementation;

  ex) why language A is designed in that way

- better use of languages already known, by using unknown (so far) features;
- ability to design/implement my/efficient language;

§ 1.2 Programming Domains – some historical view

<computer applications>		<PL's>
scientific applications	-----	Fortran, Algol, etc.
business applications	-----	Cobol
A.I. *(symbolic rather than numeric)*	-----	LISP (functional), Prolog (logical)
Web software	-----	HTML (markup language) ~ Java *(not a programming language)*
		scripting language (ability to compute): Javascript, PHP

§ 1.3 Language Evaluation Criteria – how to select a language

readability – easy to read and understand;

writability – easy to write a program;

reliability – should perform all specified tasks without problems;

- Readability

  what affect readability?

  - orthogonality: using a small set of constructs, building combined ones easily and correctly;
  - data types: ∃ facilities for defining data types and data structures;
  - syntax design: special words, statement design, etc.;

- Writability – how easily a language can be used to create programs for a given problem

  what affect writability?

  - simplicity and orthogonality: using small number of constructs is better;
  - expressivity: language should have convenient (shorter) ways of specifying computations;

- Reliability – a prog. is reliable if it performs all specified tasks under all cond's;

  what affect reliability?

  - type checking: checking type errors at compile/run time;
  - exception handling: intercept run time errors, ex) Ada, C++, C#, Java;
  - aliasing should be reduced: ∃ multiple names for a single mem. cell;

  more reliable → higher cost (of training programmers to use the language)

Table 1.1 Language evaluation criteria and the characteristics that affect them

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

<u>§ 1.4 Influences on Language Design</u>

What affect language design?

- ➢ computer architecture
- ➢ programming design methodologies

ex) structured programming: feedback (goto) in the loop construct;

top-down/bottom-up design;

data abstraction, encapsulation, object-oriented

procedure-oriented → <u>data-oriented</u> (abstract data types);

<u>§ 1.5 Language Categories (paradigms)</u>

Imperative

Object-oriented

concurrent

functional

logical (rule-based – no particular order)

Other paradigms:
- scripting languages
  (Perl, Javascript, Ruby, etc.);
- markup/programming hybrid
  (XML, JSTL, XSLT);

§ 1.7 Implementation

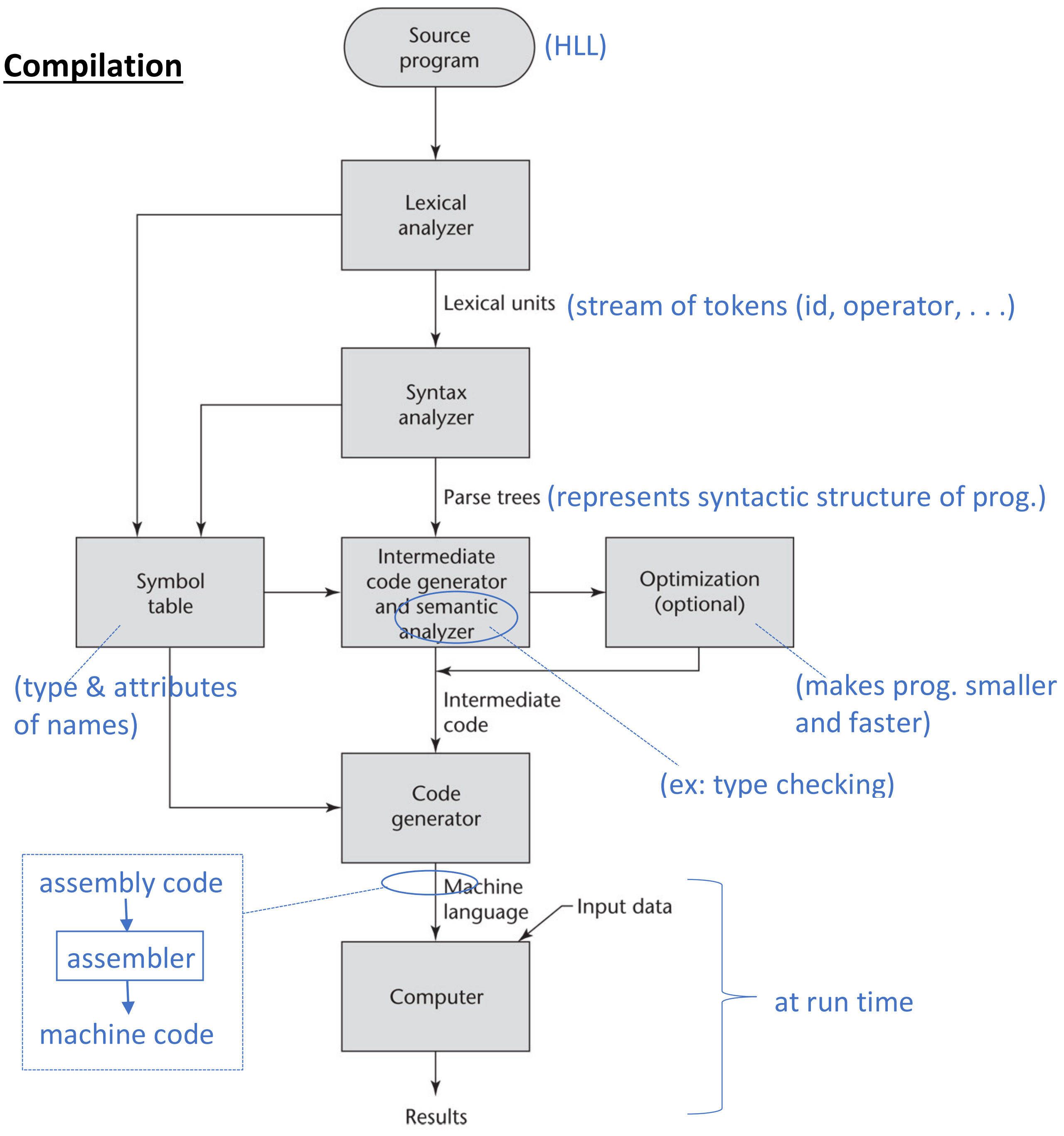
compilation

interpretation (pure)

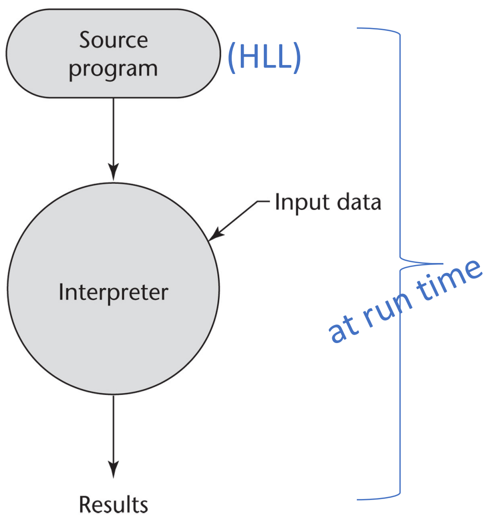
hybrid implementation (compilation + interpretation)

preprocessing

Compilation



(pure) Interpretation



ex) LISP, Javascript, PHP

advntages: easy debugging at source level;

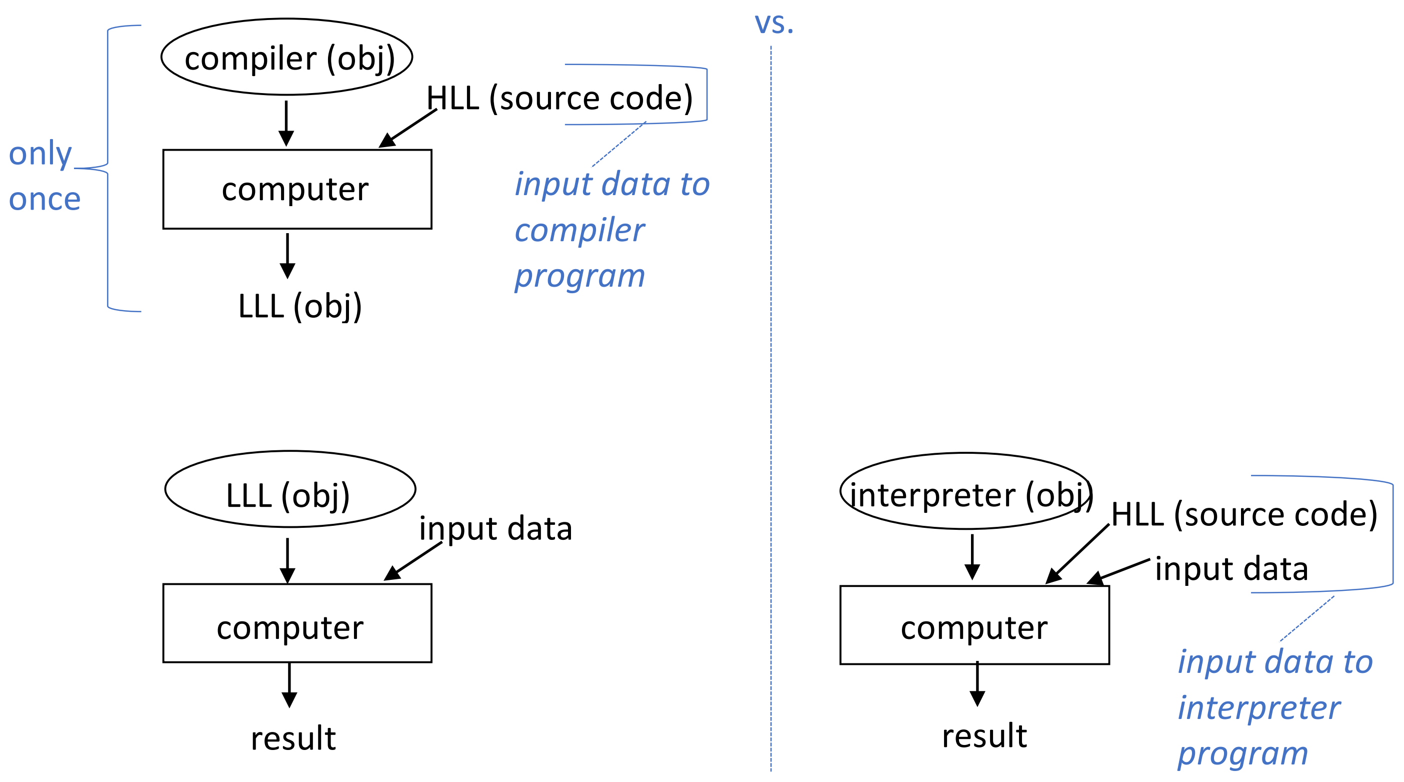
disadvantages:

execution time is slower than compiled code;

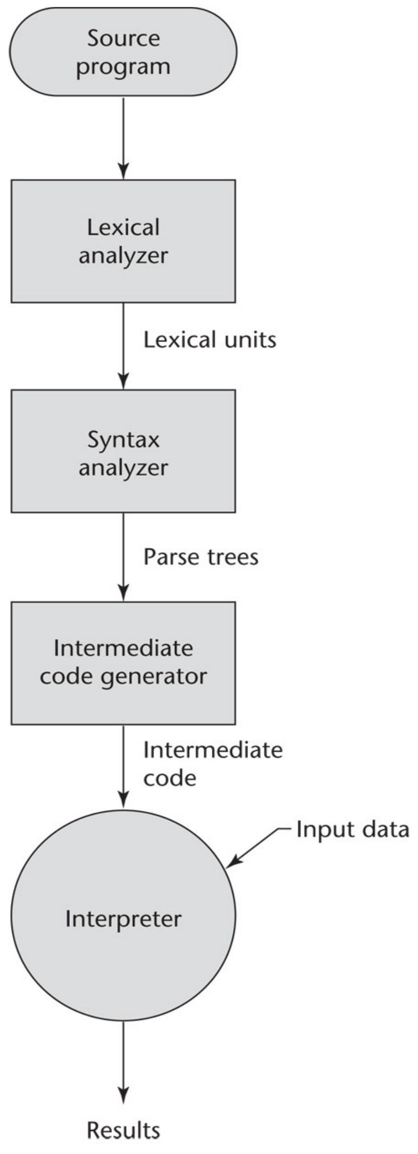
more space (data space at run time, e.g.,
symbol table, source code);

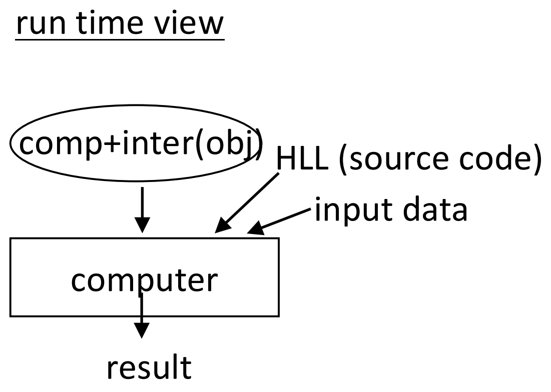
statement decoding at run time – slower;

Compilation vs. Interpretation – run time view

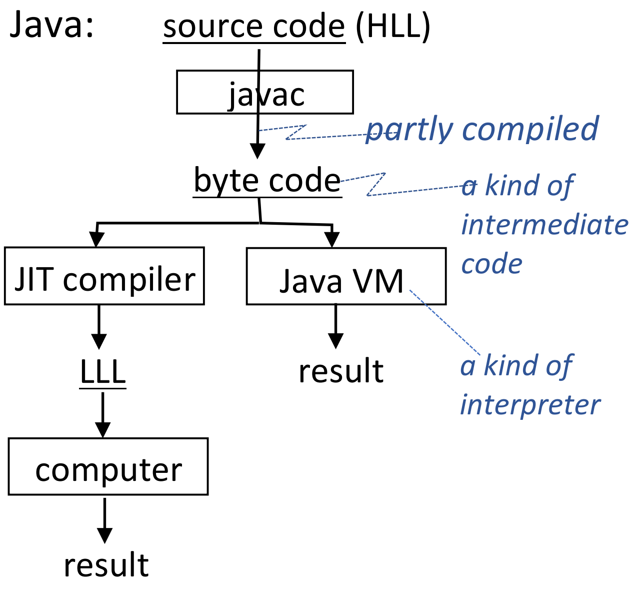


Hybrid Implementation (compilation + interpretation)





ex) Perl – partly compiled to reduce error to interpreter;



Preprocessors

before compilation starts, preprocessor is invoked for, e.g., macro (in-line) expansion, including library code, etc.

ex) #define max(A, B) ((A) > (B)? (A) : (B)) //macro definition
   x = max (2*y, z/1.73); //macro call
   → expansion: x = ((2*y) > (z/1.73)? (2*y) : (z/1.73));

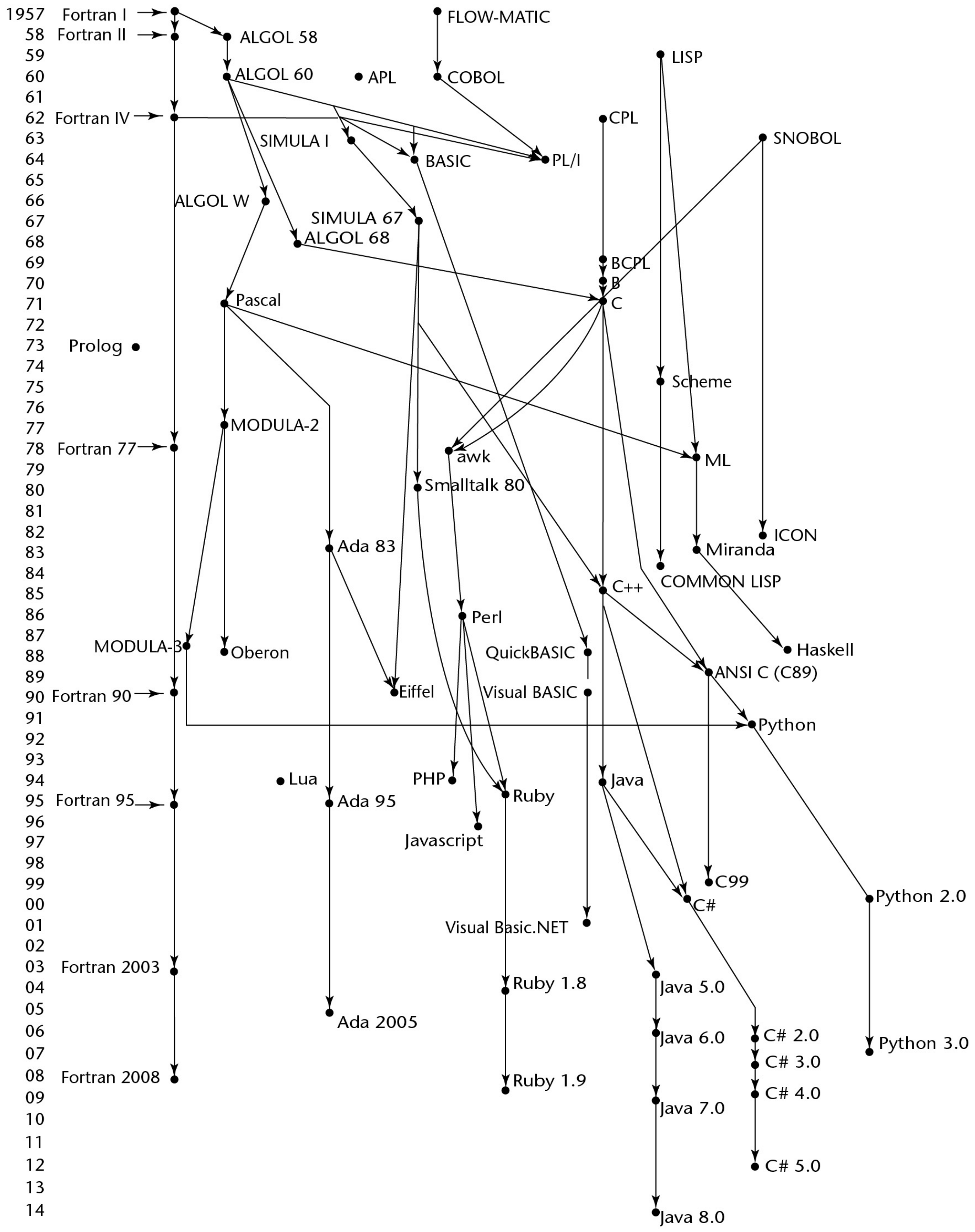


Figure 2.1 Genealogy of common high-level programming languages