Principles of Programming Languages

COMP251: Introduction

Prof. Dekai Wu

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China

Fall 2006

Why Study PLs?

- Hundreds of different PLs have been designed and implemented.
- They may be grouped into different families of PLs.
- We are *not* surveying PLs, but studying the programming concepts and constructs behind the different designs.

Goal:

- Improve your understanding of the language you are using.
- Systematically learn the various programming concepts and constructs.
- Help you learn a new language.
- Make it easier to design a new language.
- Allow a better choice of programming language.

How About Human Languages?

- *Chinese* vs. *English*:

pictorial (WYSIWYG)	*vs.*	*phonetic*
hieroglyphic	*vs.*	*alphabetical*

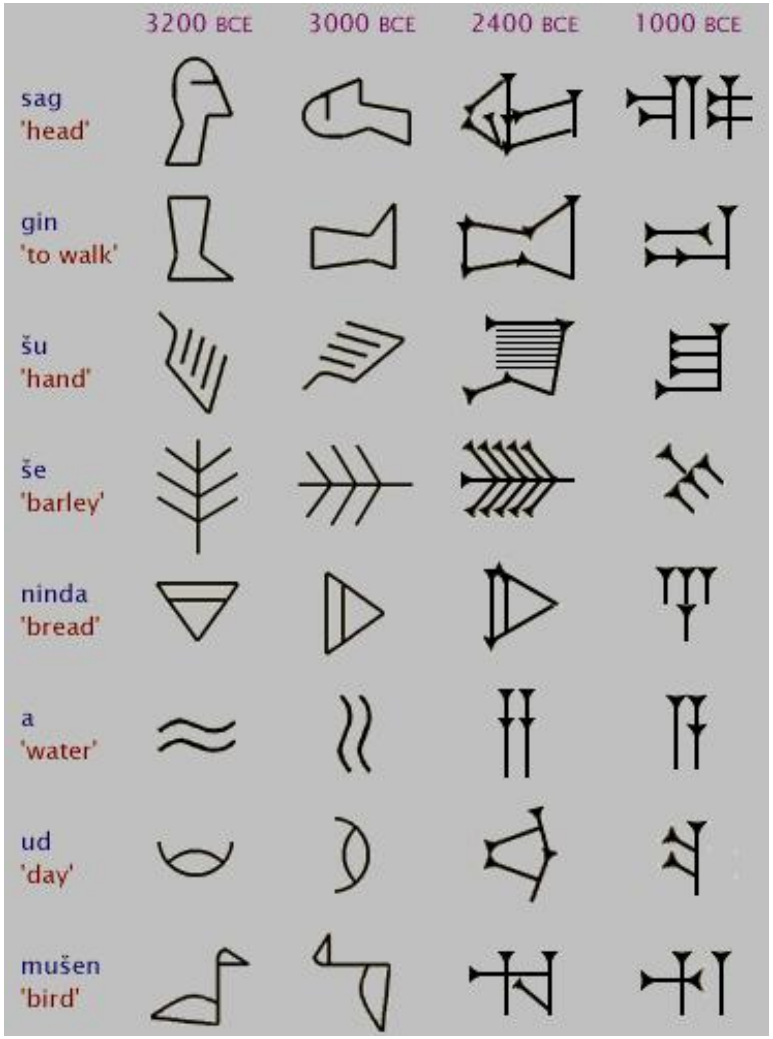
- *Japanese* vs. *English*:

wa-ta-shi-wa	*ni-hon-go*	*wa-ka-ri*	*ma-sen.*
I	Japanese	understand	don't.

An intriguing question: Do the differences in human language designs reflect how differently people think?

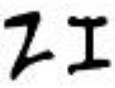
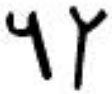
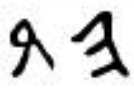
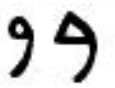
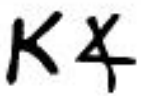
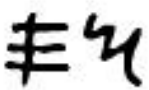
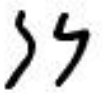
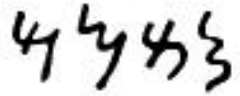
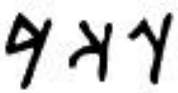
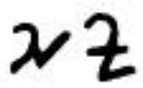
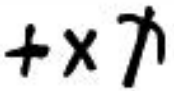
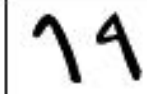
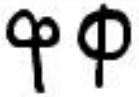
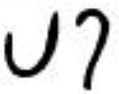
Development of Human Languages

1st written language: Sumerian, 3500 B.C.
(c.f. Chinese, Shang Dynasty, 2000 B.C.)



Development of Human Languages ..

1st alphabet: Phoenician, 1100 B.C.; only consonants.

 ḥēt ḥ	 zayin z	 wāw w	 hē h	 dālet d	 gīmel g	 bēt b	 ʾālef ʾ
 sāmek s	 nun n	 mēm m		 lāmed l	 kaf k	 yōd y	 ṭēt ṭ
 tāw t	 śin/šin š		 rēš r	 qōf q	 ṣādē ṣ	 pē p	 ʿayin ʿ

Development of Human Languages ...

1st complete alphabet: Greek, 800 B.C.; consonants + vowels.

	Ionia	Athens	Corinth	Argos	Euboea	Modern	AP	MP
alpha	Α Α	Α Α	Α Α	Α Α	Α Α	Α α	[a]	[a]
beta	Β	Β	𐌁	ᒐ	Β	Β β	[b]	[v]
gamma	Γ	Λ	ᐸC	ᒣΛ	ᐸC	Γ γ	[g]	[y]
delta	Δ	Δ	Δ	D	ᐅ	Δ δ	[d]	[ð]
epsilon	ᚼE	ᚼE	ᚼ	ᚼE	ᚼE	Ε ε	[e]	[e]
digamma		ᚹ	ᚹ	ᚹF	ᚹ		[w]	
zeta	I	I	I	I	I	Ζ ζ	[zd]	[z]
eta	⊟H					Η η	[e:]	[i]
heta		⊟H	⊟H	⊟H	⊟H		[h]	
theta	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	Θ θ	[t^h]	[θ]
iota	I	I	ᛊ	I	I	Ι ι	[i]	[i]
kappa	K	K	K	K	K	Κ κ	[k]	[k]
lambda	ᒣΛ	ᒪ	ᒣΛ	⊢	ᒪ	Λ λ	[l]	[l]
mu	ᗰM	ᗰM	ᗰM	ᗰM	ᗰM	Μμ	[m]	[m]
nu	ᘉN	ᘉN	ᘉN	ᘉN	ᘉN	Ν ν	[n]	[n]
xi	⋿		⋿	⋿	X	Ξ ξ	[ks]	[ks]
omicron	O	O	O	O	O	Ο ο	[o]	[o]
pi	ᒥ	ᒥ	ᒥ	ᒥ	ᒥ	Π π	[p]	[p]
san			ᗰ	ᗰ	ᗰ		[s]	
koppa	ᛩ	ᛩ	ᛩ	ᛩ	ᛩ		[q]	
rho	ᑭD	ᑭR	ᑭR	ᑭR	ᑭ	Ρ ρ	[r]	[r]
sigma	ᛊ	ᛊ		ᛊ	ᛊ	Σ σς	[s]	[s]
tau	T	T	T	T	T	Τ τ	[t]	[t]
upsilon	VY	ᚴVY	ᚴVY	ᚴVY	ᚴVY	Υ υ	[u]	[i]
phi	Φ	ΦⒹ	ΦⒹ	ΦⒹ	ΦⒹ	Φ φ	[p^h]	[f]
khi	X	X	X	X	ᛘᛘ	Χ χ	[k^h]	[x]
psi	ᛘᛘ		ᛘᛘ	ᛘ		Ψ ψ	[ps]	[ps]
omega	Ω					Ω ω	[o:]	[o]

What's a PL for?

Stroustrup (C++ designer, 1994):

- *tool* for instructing machines?
- *means* for communicating between programmers?
- *vehicle* for expressing high level designs?
- *notation* for algorithms?
- *way* of expressing relationships between concepts?
- *tool* for experimentation?
- *means* for controlling computerized devices?
- collection of "neat" features?

His answer: All of the above except the last one.

Can You Understand This?

00001001001011100110011001101001011011000110010100001001001000100111011000110010101100011011101000111
01010111001001100101001100010010111001100011001000100000101001100111011000110110001100110010010111111
01100011011011110110110101110000011010010110110001100101011001000010111000111010000010100010111001111
00110110010101100011011101000110100101101111011011100000100100100010001011100111010001100101011111000
01110100001000100000101000001001001011100110000101101100011010010110011101101110001000000011010000000
10100000100100101110011001110110110001101111011000100110000101101100001000000110110101100000101101001
01101110000010100000100100101110011101000111100101110000011001010000100100100000011011010110000010110
10010110111000101100001000110110011001110101011011100110001101110100011010010110111101101110000010100
00001001001011100111000001110010011011110110001100001001001100000011010000001010011011010110000101100
10010110111000111010000010100000100100100001001000110101000001010010010011110100110001001111010001110
01010101010001010010001100100000001100000000101000001001011100110110000010111011001100101001000000010
01010111001101110000001011000010110100110001001100100001110000010110000010010101110011011100000000101 0
00001010000010010010000100100011010100000101001001001111010011000100111101000111010101010100010100 10
00110010000000110001000010100000100101101101011011110111011000100000001100010010110000100101011011 11
00110000000010100000100101110011011101000010000000100101011011110011000000101100010110110010010101 10
01100111000000101101001100100011000001011101000010100000100101101101011011110111011000100000001100 10
00101100001001010111011110011000000001010000010010111001101110100001000000010010101101111001100000 10
11000101101100100101011001100111000000101101001100100011010001011101000010100000100101101100011001 00
00100000010110110010010101100110011100000010110100110010001100000101110100101100001001010110111100 11
00000000101000001001011011000110010000100000010110110010010101100110011100000001011010011001000110 100
01011101001011000010010101101111001100010000101000001001011000010110010001100100001000000010010101 10
11110011000000101100001001010110111100110001001011000010010101101111001100000000101000001001011100 11
01110100001000000010010101101111001100000010110001011011001001010110011001110000001011010011001000 11
10000101110100001010000010010110110101101111011101100010000000110000001011000010010101101001001100 00
00001010000010010110001000100000001011100100110001001100001100010000101000001001011011100110111101 11
00000000101000101110010011000100110000110001001110100000101000001001011100100110010101110100000010 10
00001001011100100110010101110011011101000110111101110010011001010000101000101110010011000100110001 10
01100110010100110001001110100000101000001001001011100111001101101001011110100110010100001001001000 00
01101101011000010110100101101110000101100001011100100110001001100011001100110010100110001001011010 10
11010110000101101001011011100000101000001001001011100110100101100100011001010110111001110100000010 01
00100010010001110100001101000011001110100010000000101000010001110100111001010101001010010010000000 11
00100010111000111000001011100011000100100010000010 10

How About This?

```
main:
        !#PROLOGUE# 0
        save %sp,-128,%sp

        !#PROLOGUE# 1
        mov 1,%o0
        st %o0,[%fp-20]
        mov 2,%o0
        st %o0,[%fp-24]
        ld [%fp-20],%o0
        ld [%fp-24],%o1
        add %o0,%o1,%o0
        st %o0,[%fp-28]
        mov 0,%i0
        nop
```

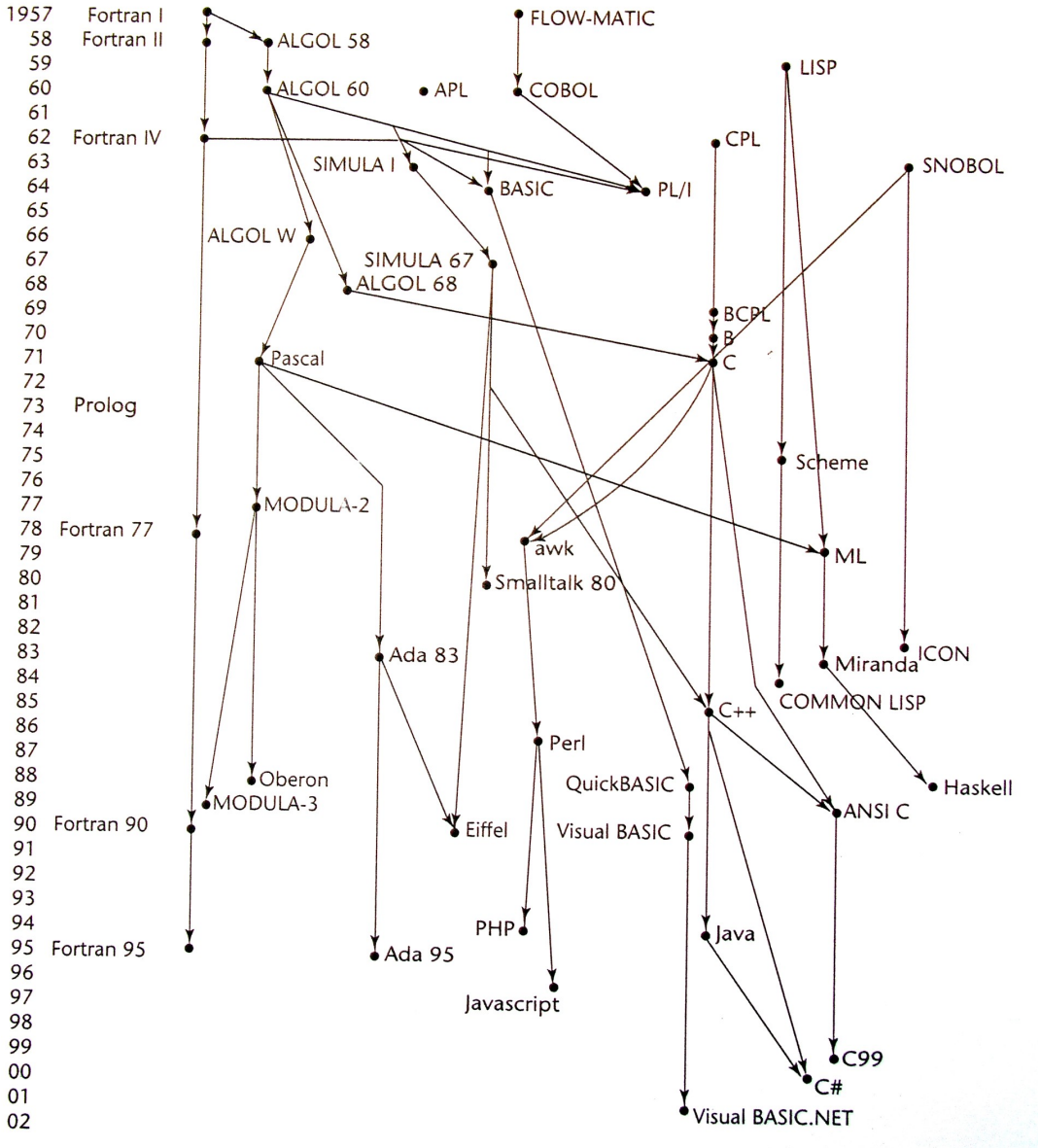
Is This Better Now?

```
#include <stdio.h>

int main()
{
    int x, y, z;

    x = 1;
    y = 2;
    z = x+y;

    return 0;
}
```

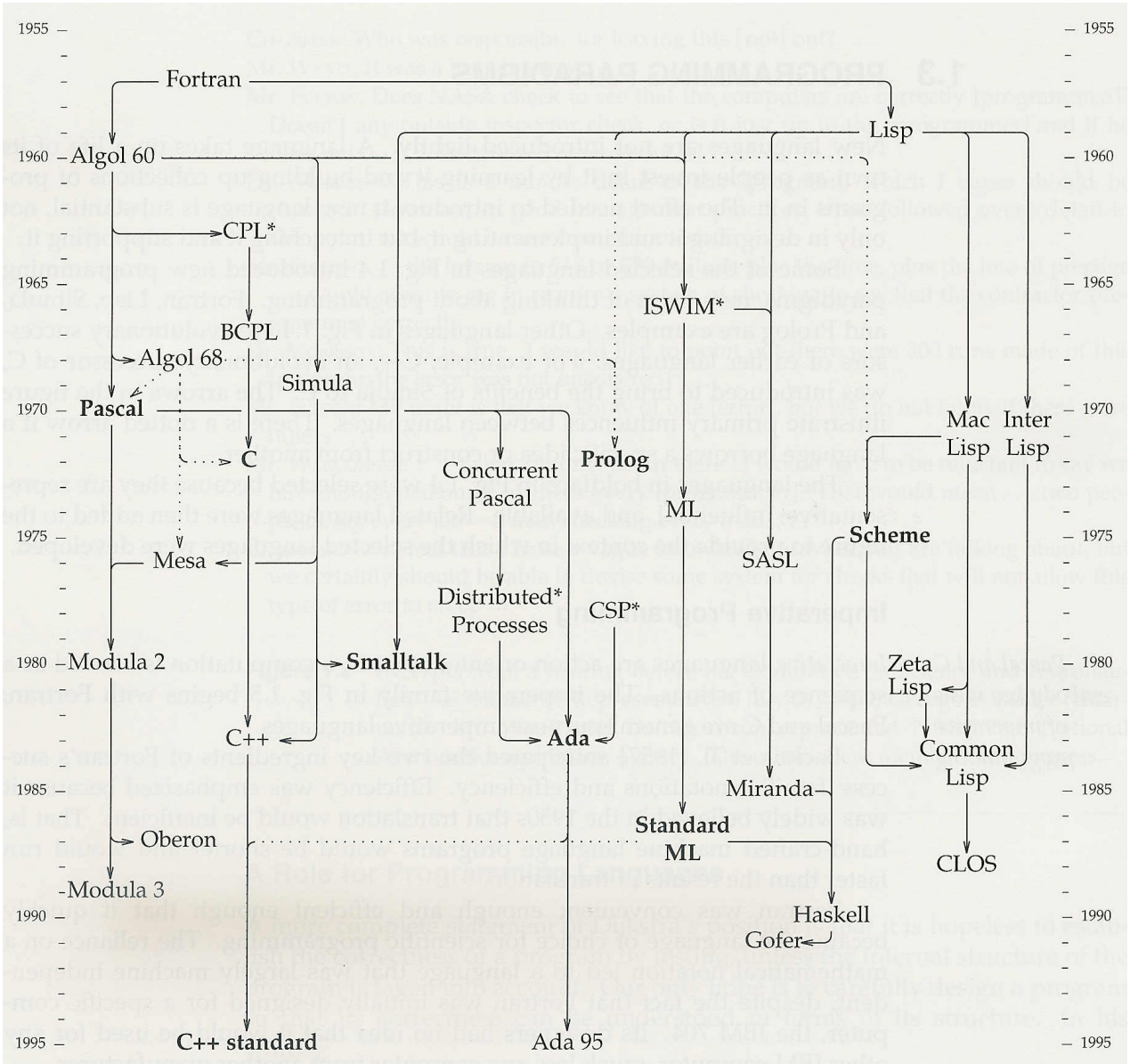
Levels of PLs

- machine (binary) language is unintelligible
- assembly language is low level
  - mnemonic names for machine operations
  - explicit manipulation of memory addresses/contents
  - machine dependent
- high level language
  - readable
    - instructions are easy to remember
    - faster coding
    - less error-prone (fewer bugs?)
    - easier to maintain
  - no mention of memory locations
  - machine independent = portable

Genealogy of Common PLs



Genealogy of Common PLs (Sethi 1996)



4 Paradigms of PL Design

- Imperative Programming (IP) or Procedural Programming (PP)
  - See http://en.wikipedia.org/wiki/Procedural_programming
- Object-Oriented Programming (OOP)
- Declarative Programming
  - Functional Programming (FP)
  - Logic Programming (LP)

PL design is a balance among:

- efficiency
- readability
- support
- taste!

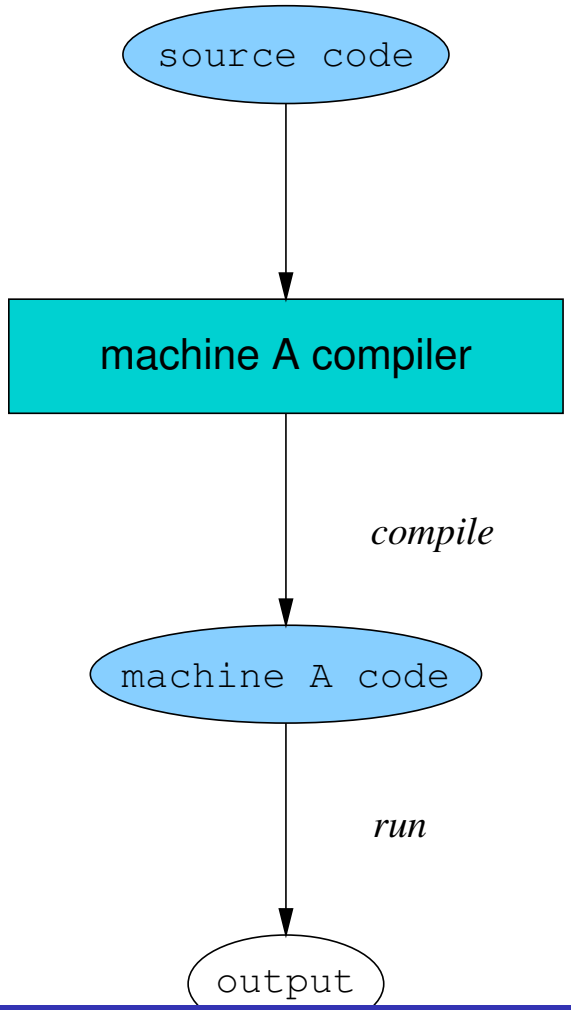
IP/PP, OOP, FP, LP

IP/PP	OOP	FP	LP
FORTRAN Pascal C	Smalltalk C++ Java	LISP Scheme SML	Prolog
action-oriented	object-oriented	function-oriented	logic-oriented
procedural assignments	classes inheritance	functions mapping: $x \longrightarrow f(x)$	logic reasoning "Are you sick?"
algorithm design	system building reusable software	formal specification program correctness	expert system database queries
compile	compile	interpret	interpret

Compilation: From Source to Runnable Program

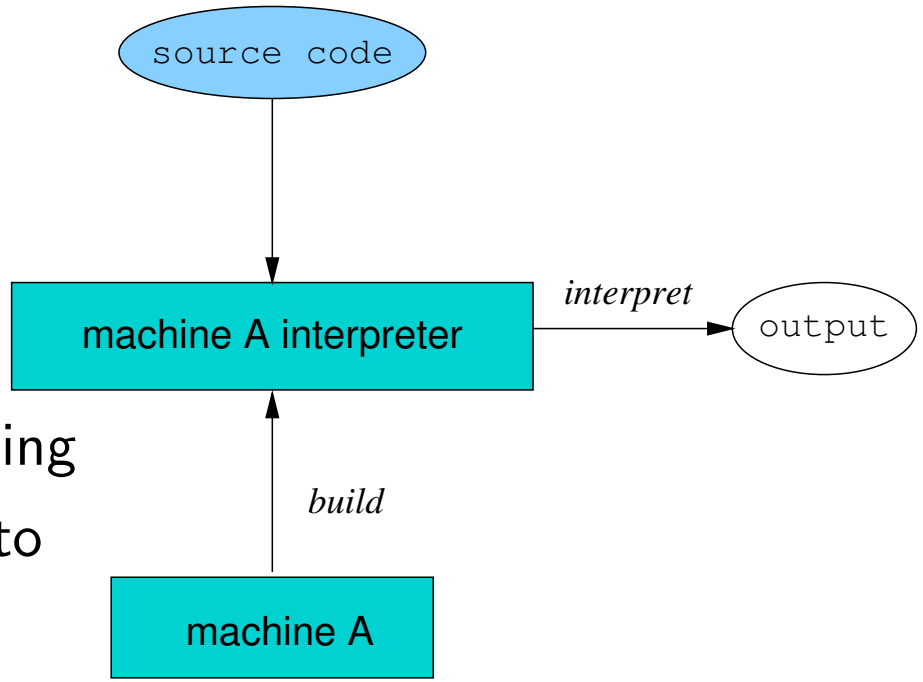
A **compiler** translates source programs into machine codes that run directly on the target computer. e.g. a.c $\longrightarrow$ a.out.

- static codes
- compile once, run many
- optimized codes
  $\Rightarrow$ more efficient
- examples: FORTRAN, Pascal, C++



Interpretation: From Source to Program Output

An **interpreter** is a virtual machine implemented on a target computer which runs a source program directly.

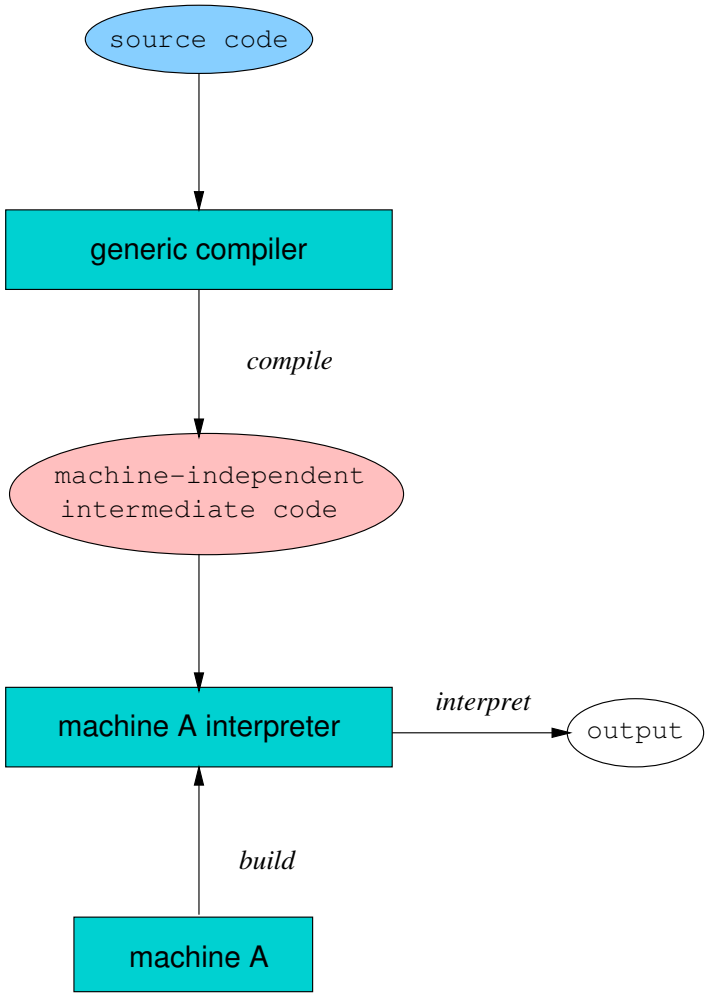


- slower
- interpret many, run many
- interactive mode: easy debugging
- more flexible: allow programs to be changed "on the fly"
- examples: many script languages (sh, csh, tcl, awk), ML, PROLOG

Hybrid Implementation System

A hybrid system translates high-level source programs to an intermediate language which then allows fast and easy interpretation.

- compile once, interpret many
- Examples: UCSD Pascal, Perl, Python, Java



Recapitulate

- √ There are hundreds of different PLs.
- √ It is easier to write (large) programs with a high-level PL.
- √ Will emphasize the basic programming concepts/constructs.
- √ Will address 4 programming paradigms: IP/PP, OOP, FP, LP.
- √ 2 approaches to types within the FP paradigm: latent typing vs. static typing with type inference.
- √ 2 ways to implement PLs: compilation vs. interpretation.