Principles of Programming Languages

COMP3031: Introduction

Prof. Dekai Wu

Human Language Technology Center
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Fall 2012

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*

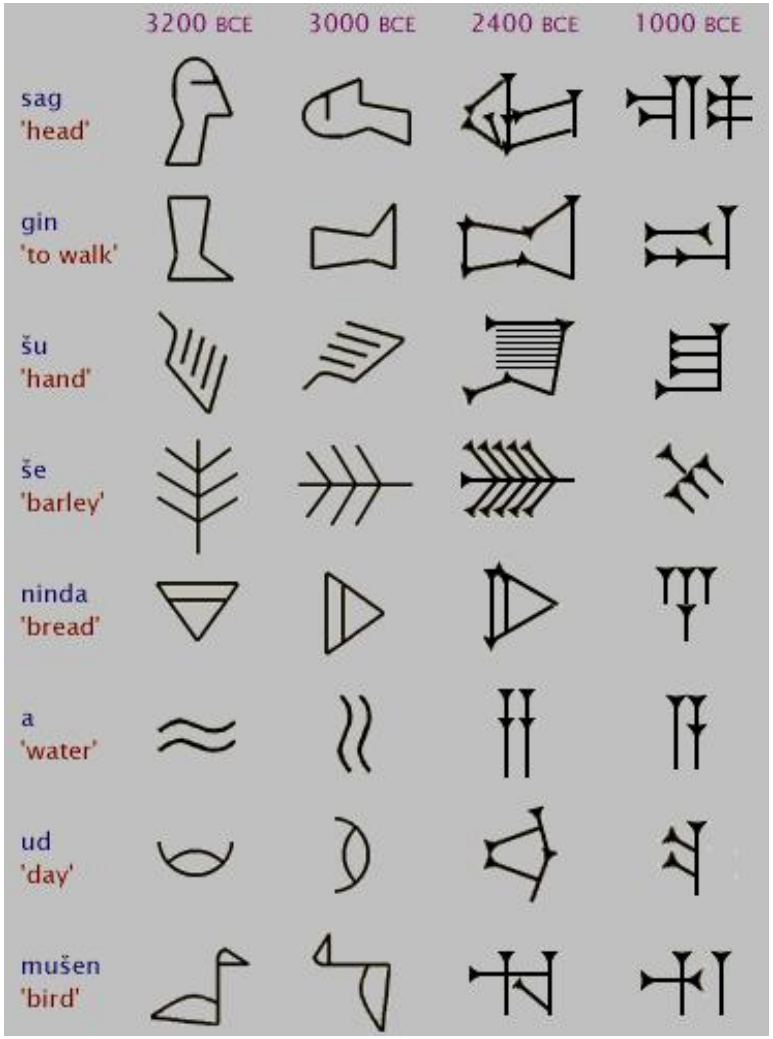
- *Japanese* vs. *English*:

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	A A	A A	A A	A A	A A	Α α	[a]	[a]
beta	B	B	ᗺ	ᑢ	B	Β β	[b]	[v]
gamma	Γ	Λ	<C	ᒣΛ	<C	Γ γ	[g]	[γ]
delta	Δ	Δ	Δ	D	ᐅ	Δ δ	[d]	[ð]
epsilon	ᖴE	ᖴE	ᖵ	ᖴE	ᖴE	Ε ε	[e]	[e]
digamma		ᖷ	ᖷ	ᖷF	ᖷ		[w]	
zeta	I	I	I	I	I	Ζ ζ	[zd]	[z]
eta	ᗮH					Η η	[e:]	[i]
heta		ᗮH	ᗮH	ᗮH	ᗮH		[h]	
theta	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	⊕⊗⊙	Θ θ	[t^h]	[θ]
iota	I	I	ᴝ	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	ᙢM	ᙢM	ᙢM	ᙢM	ᙢM	Μ μ	[m]	[m]
nu	ᙡN	ᙡN	ᙡN	ᙡN	ᙡN	Ν ν	[n]	[n]
xi	Ξ		Ξ	Ξ	X	Ξ ξ	[ks]	[ks]
omicron	O	O	O	O	O	Ο ο	[o]	[o]
pi	Γ	Γ	Γ	Γ	Γ	Π π	[p]	[p]
san			M	M	M		[s]	
koppa	ᑫ	ᑫ	ᑫ	ᑫ	ᑫ		[q]	
rho	PD	PR	PR	PR	P	Ρ ρ	[r]	[r]
sigma	ᴝ	ᴘ		ᴝ	ᴘ	Σ σς	[s]	[s]
tau	T	T	T	T	T	Τ τ	[t]	[t]
upsilon	VY	ᒣVY	ᒣVY	ᒣVY	ᒣVY	Υ υ	[u]	[i]
phi	Φ	ΦⒹ	ΦⒹ	ΦⒹ	ΦⒹ	Φ φ	[p^h]	[f]
khi	X	X	X	X	ΨΨ	Χ χ	[k^h]	[x]
psi	ΨΨ		ΨΨ	Ψ		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?

0000100100101110011001100110100101101100011001010000100100100010011011000110010101100011011101000111
0101011100100110010100110001001011100110001100100010000010100110011101100011011000110011001001011111
0110001101101111011011010111000001101001011011000110010101100100001011100011101000001010001011100111
0011011001010110001101110100011010010110111101101110000010010010001000101110011101000110010101111000
0111010000100010000010100000010010010111001100001011011000110100101100111011011100010000000110100000
1010000010010010111001100111011011000110111101100010011000010110110000100000011011010110000101101001
0110111000001010000010010010111001110100011110010111000001100101000010010010000001101101011000010110
1001011011100010110000100011011001100111010101101110011000110111010001101001011011110110111000001010
0000100100101110011100000111001001101111011000110000100100110000001101000000101001101101011000010110
1001011011100011101000001010000010010010000100100011010100000101001001001111010011000100111101000111
0101010101000101001000110010000000110000000010100000100101110011011000010111011001100101001000000010
0101011100110111000000101100001011010011000100110010001110000010110000100101011100110111000000001010
0000101000001001001000010010001101010000010100100100111101001100010011110100011101010101010001010010
0011001000000011000100001010000010010110110101101111011101100010000000110001001011000010010101101111
0011000000001010000010010111001101110100001000000010010101101111001100000010110001011011001001010110
0110011100000010110100110010001100000101110100001010000010010110110101101111011101100010000000110010
0010110000100101011011110011000000001010000010010111001101110100001000000010010101101111001100000010
1100010110110010010101100110011100000010110100110010001101000101110100001010000010010110110001100100
0010000001011011001001010110011001110000001011010011001000110000010111010010110000100101011011110011
0000000010100000100101101100011001000010000001011011001001010110011001110000001011010011001000110100
0101110100101100001001010110111100110001000010100000100101100001011001000110010000100000001001010110
1111001100000010110000100101011011110011000100101100001001010110111100110000000010100000100101110011
0111010000100000001001010110111100110000001011000101101100100101011001100111000000101101001100100011
1000010111010000101000001001011011010110111101110110001000000011000000101100010010101101001001100000
0000101000001001011000100010000000101110010011000100110000110001000010100000100101101110011011110111
0000000010100010111001001100010011000011000100111010000010100000100101110010011001010111010000001010
0000100101110010011001010111001101110100011011110111001001100101000010100010111001001100010011000110
0110011001010011000100111010000010100000100100101110011100110110100101110100110010100001001001000000
0110110101100001011010010110111000101100001011100100110001001100011001100110010100110001001011010110
1101011000010110100101101110000010100000100100101110011010010110010001100101011011100111010000001001
0010001001000111010000110100001100111010001000000010100001000111010011100101010100101001001000000011
0010001011100011100000101110001100010010001000001010

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
```

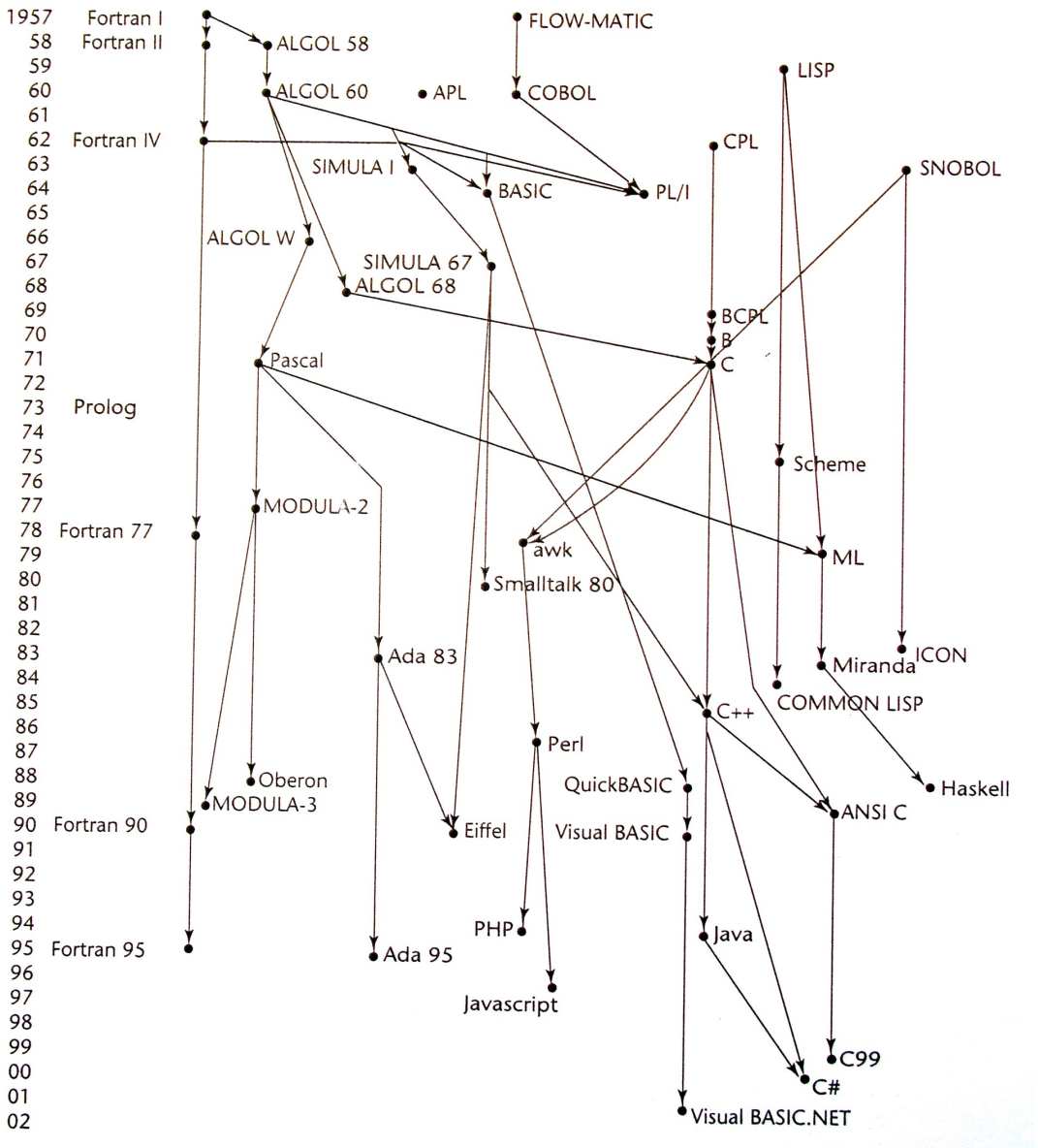
Is This Better Now?

```
#include <stdio.h>

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

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

    return 0;
}
```

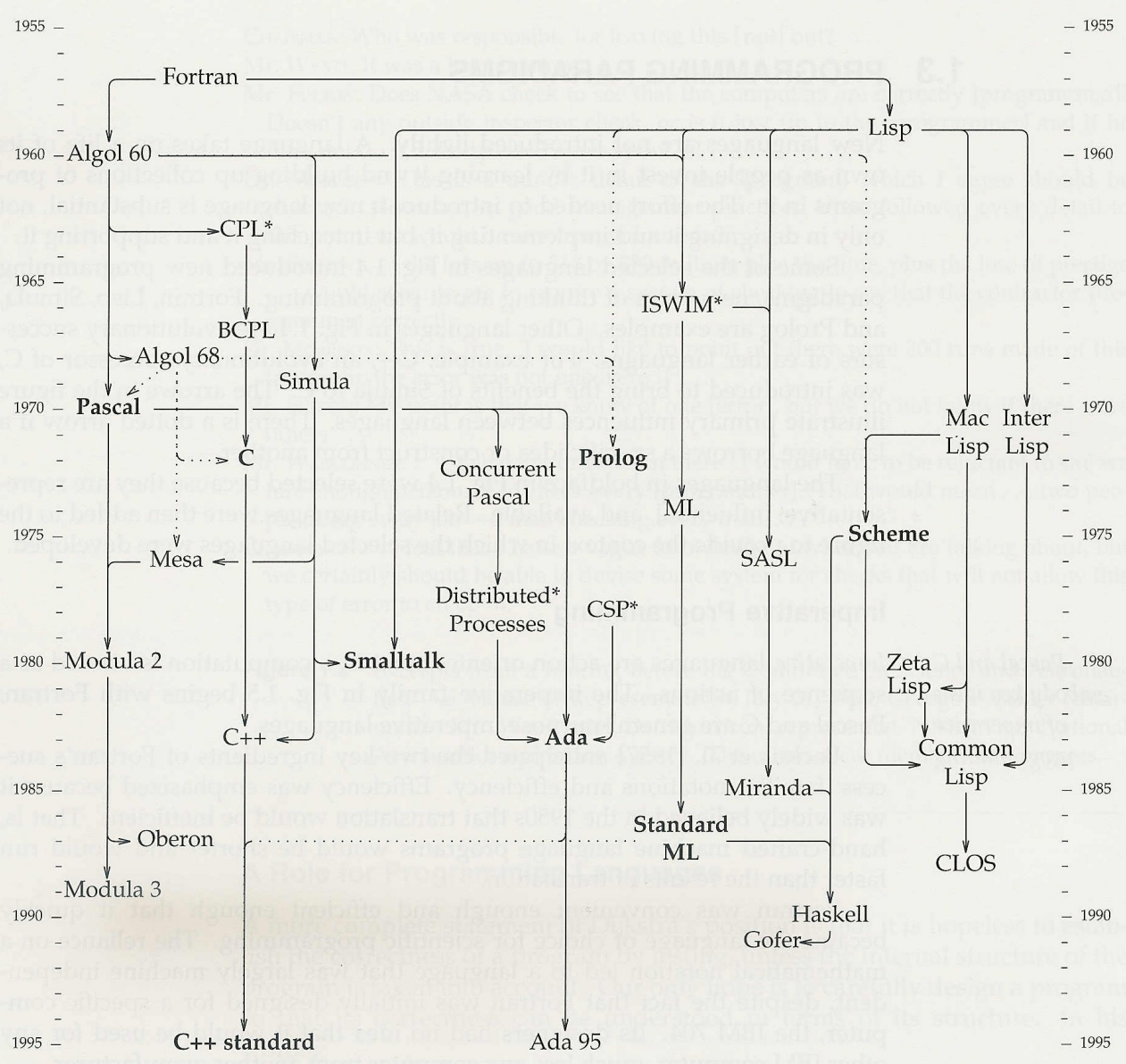
Levels of PLs

- 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

- Procedural Programming (PP) or Imperative Programming (IP)
  - 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!

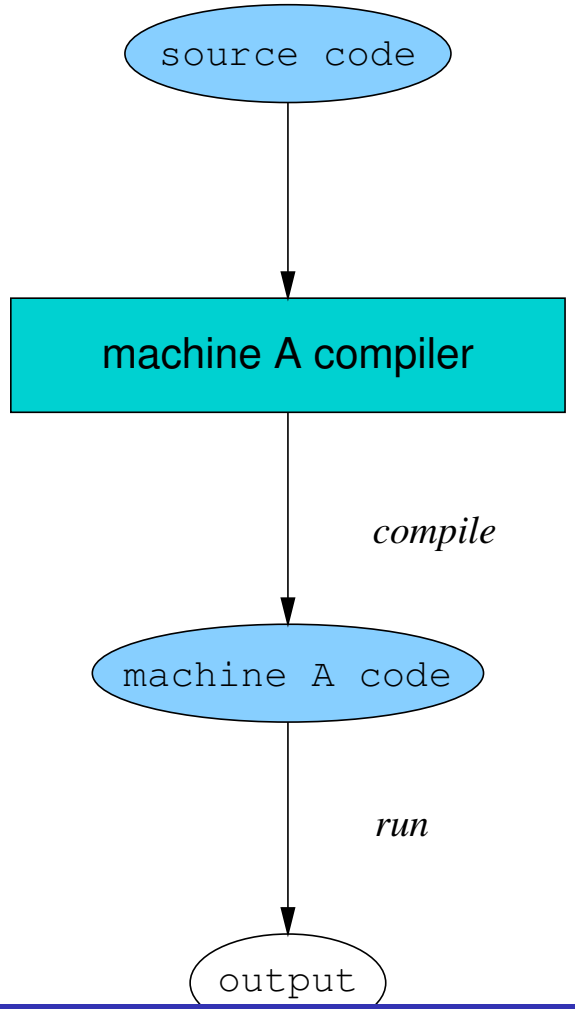
IP/PP, OOP, FP, LP

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: $x \rightarrow f(x)$	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.

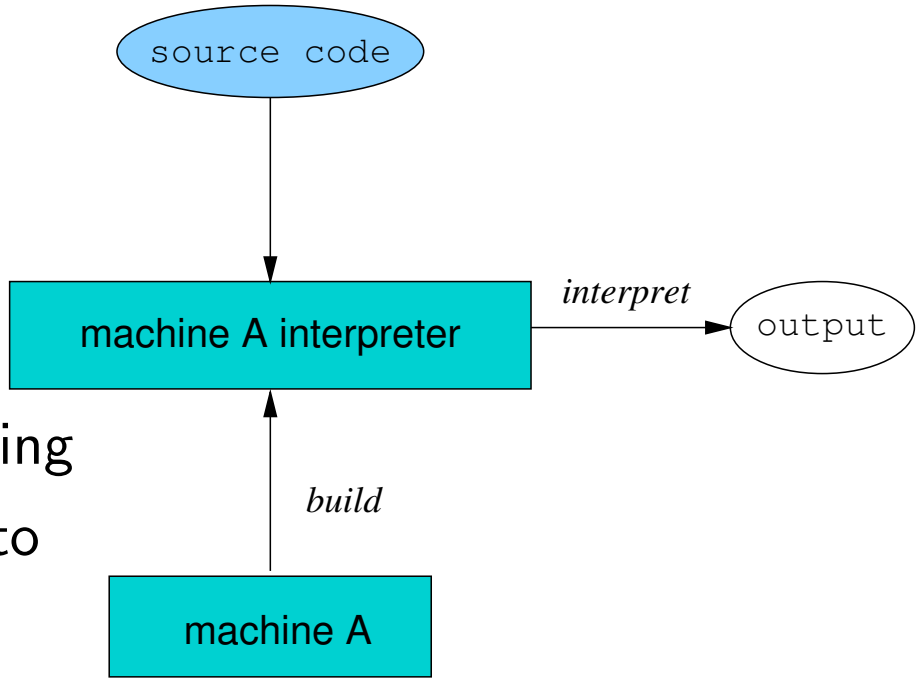
- static codes
- compile once, run many
- optimized codes
  $\Rightarrow$ more efficient
- examples: FORTRAN, Pascal, C++



Interpretation: From Source to Program Output

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

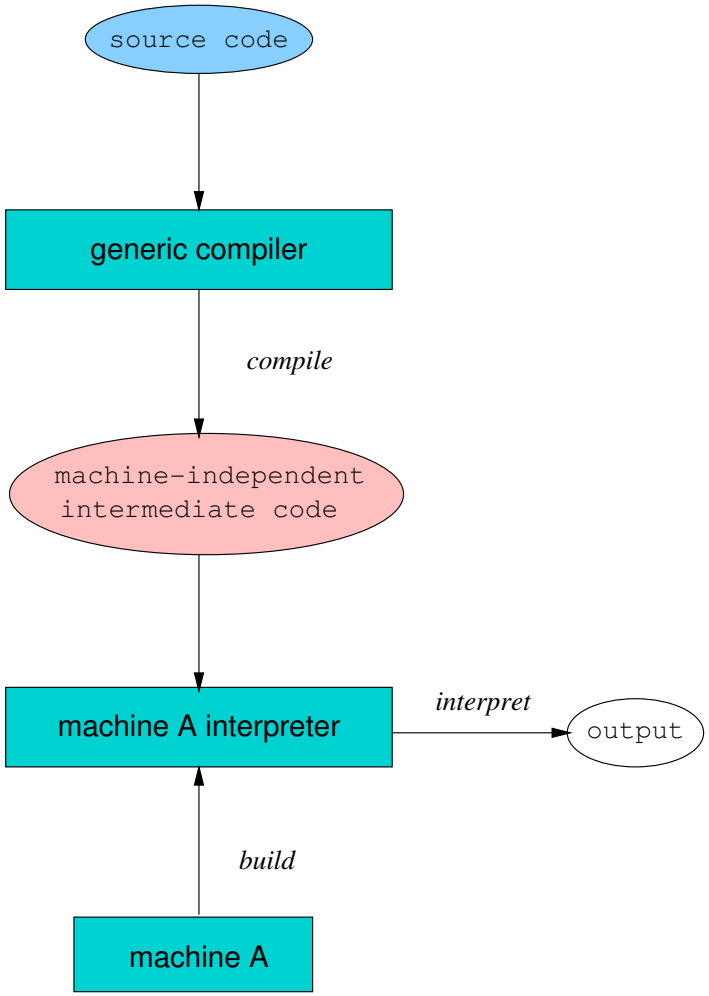
- slower
- interpret many, run many
- interactive mode: easy debugging
- more flexible: allow programs to be changed "on the fly"
- examples: many script languages (sh, csh, tcl, awk), ML, PROLOG



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.