Programming Languages

Objectives

At the end of the meeting, students should be able to:

- Classify programming languages based on different criteria
- Differentiate between compilers and interpreters

Programming Language

- System for describing computation.
- System of signs to communicate a task or algorithm to a computer, causing the task to be performed.

Syntax and Semantics

- **Syntax** is the form in which programs are written (expressions, commands, declarations).
- **Semantics** is the meaning given to the various syntactic constructs.

Hello World in Different PL's

1. **In Assembly Language**

```
dosseg
    .model small
    .stack 100h
    .data
    hello_message db 'Hello,
        World!', 0dh, 0ah, '$'

.code
    main proc
    mov ax, @data
    mov ds, ax
    mov ah, 9
    mov dx, offset hello_message
    int 21h

    mov ax, 4C00h
    int 21h
    main endp
    end main
```

Hello World in Different PL's

2. **In Java**

```
public class Hello {
    public static void main(String [] args) {
        System.out.println("Hello World");
    }
}
```

3. **In COBOL**

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.  Hello.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    PROCEDURE DIVISION.
        Display 'Hello, World'.
         STOP RUN.
```

Hello World in Different PL's

4. **In Scheme**

```
(define hello-world
    (lambda ()
        (writes 'nil "Hello, World!")))
```

5. **In LOLCODE**

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!“
KTHXBYE
```

Hello World in Different PL's

6. **In Brain****

```
+++++ +++++                 initialize counter (cell #0) to 10
[                            use loop to set the next four cells to 70/100/30/10
    > +++++ ++              add 7 to cell #1
    > +++++ +++++           add 10 to cell #2
    > +++                   add 3 to cell #3
    > +                     add 1 to cell #4
    <<<< -                  decrement counter (cell #0)
]
> ++ .                      print 'H'
> + .                       print 'e'
+++++ ++ .                  print 'l'
.                           print 'l'
+++ .                       print 'o'
> ++ .                      print ' '
<< +++++ +++++ +++++ .      print 'W'
> .                         print 'o'
+++ .                       print 'r'
----- - .                   print 'l'
----- --- .                 print 'd'
> + .                       print '!'
> .                         print '\n'
```

Classifying PL's

1. Levels of Abstraction
2. Generations
3. Programming Paradigms

Levels of Abstraction

	LOW LEVEL	HIGH LEVEL	VERY HIGH LEVEL
Instructions	Simple machine-like	Expressions and explicit flow of control	Fully abstract machine
Memory Handling	Direct memory access and allocation	Memory access and allocation through operations	Fully hidden memory access and automatic allocation
Examples	Machine, Assembly	C, Java	Logo

Levels of Abstraction

\- User friendly +

Low Level	High Level

\+ Control of computer -

Sneak Peak: Logo

```
FORWARD 100
LEFT 90
FORWARD 100
LEFT 90
FORWARD 100
LEFT 90
FORWARD 100
LEFT 90
```

http://www.mathsnet.net/logo/turtlelogo/index.html

Generations

(1) FIRST GENERATION **Low-Level Machine Language, Assembly Language**	**(2) SECOND GENERATION (early 1960's)** **ALGOL-60, BASIC, COBOL, FORTRAN**
(3) THIRD GENERATION (late 1960's to present) Pascal, C, ADA, Java, Eiffel	(4) FOURTH GENERATION (domain specific language) VB, SQL, Access, Excel

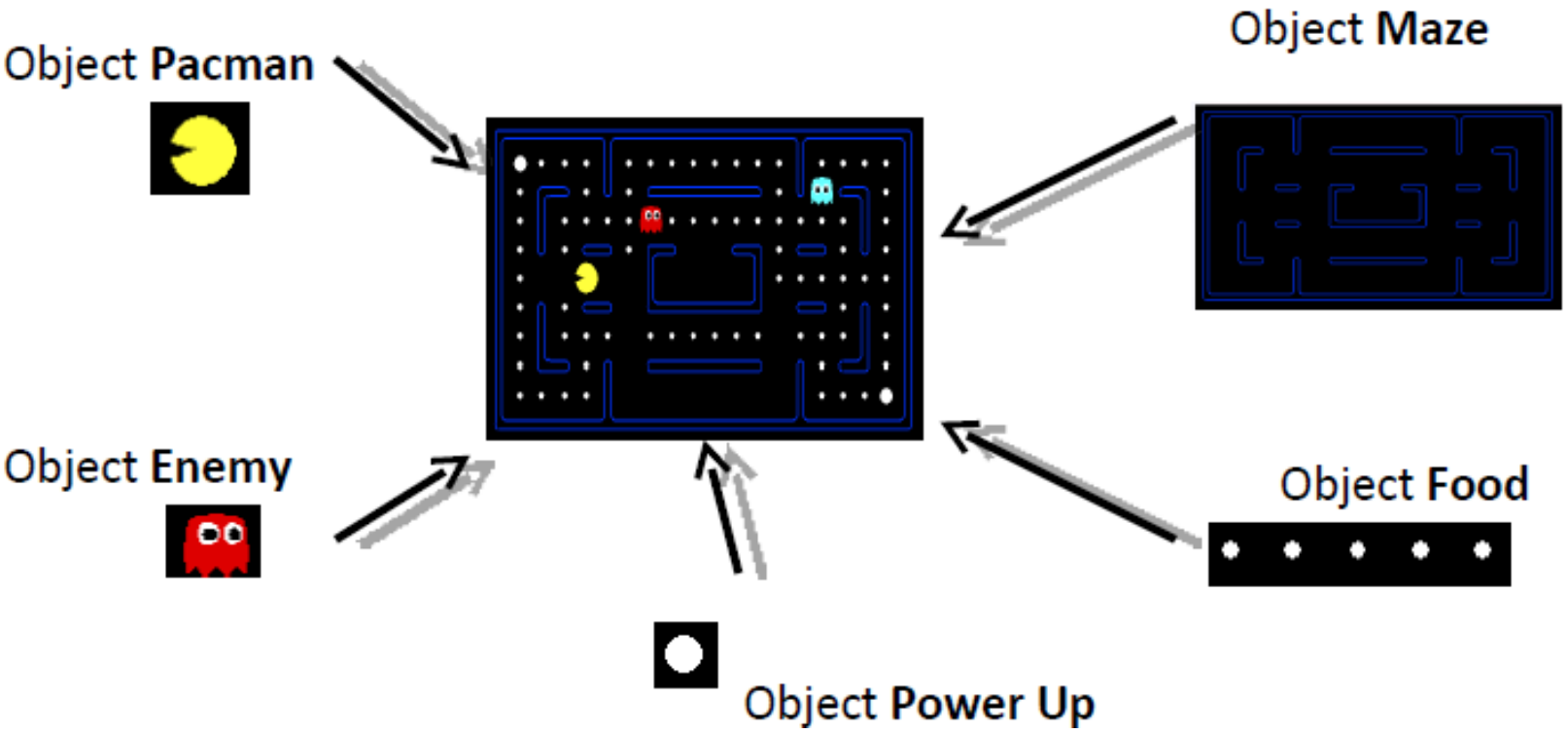
Programming Paradigms

1. **Imperative**
   - "How it is to be achieved"
   - To solve a problem, we specify the step-by-step procedure.
   - Central features are variables, assignment statements, and iteration

   a. **Block-Structured**
      - The procedure is the principal building block of the program.
      - Examples: Pascal, C

   b. **Object-Oriented**
      - Languages that employ objects.
      - An object is a group of procedures that share a state.
      - Examples: Java, Modula

Object-Oriented: Pacman



Programming Paradigms

2. **Declarative**
   - "What it is to be achieved"
   - Program requires specification of a relation or function.
   - Mainly based from math concepts on logic, theory on functions and relational calculus.

   a. **Logic**
      - Based on a subset of predicate calculus.
      - Axioms and rules are used to deduce new facts.
      - Example: Prolog

   b. **Functional**
      - Operate only through functions which return one value given a list of parameters.
      - Example: Lisp

Sneak Peak: Prolog (Facts)

```
valuable(gold).                 /*gold is valuable*/
valuable(money).                /*money is valuable*/

father(john,mary).              /*john is the
                                  father of mary*/
gives(john,book,mary).          /*john gives
                                  book to mary*/
```

Sneak Peak: Prolog (Queries)

valuable(gold)? /*Is gold valuable?*/
Prolog Reply: yes

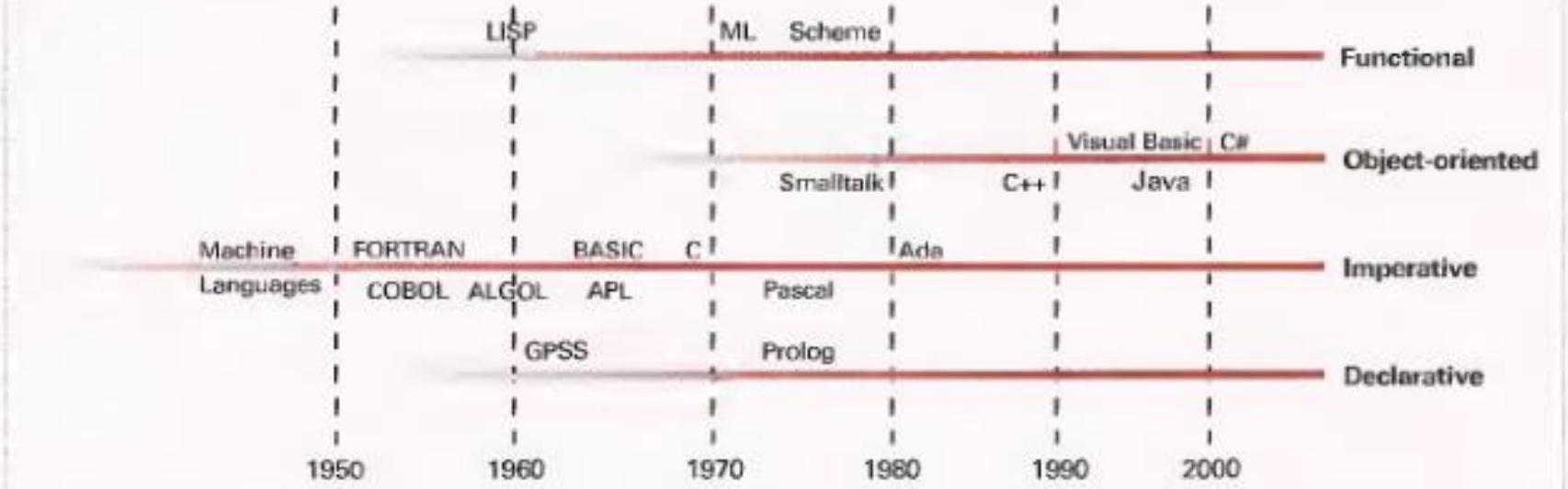
father(X,mary)? /*Who is the father of mary?*/
X = john

valuable(X)? /*Which objects are valuable?*/
X=gold
X=money

gives(john,X,mark),valuable(X)
/*Does john gives X to mark and X is valuable*/

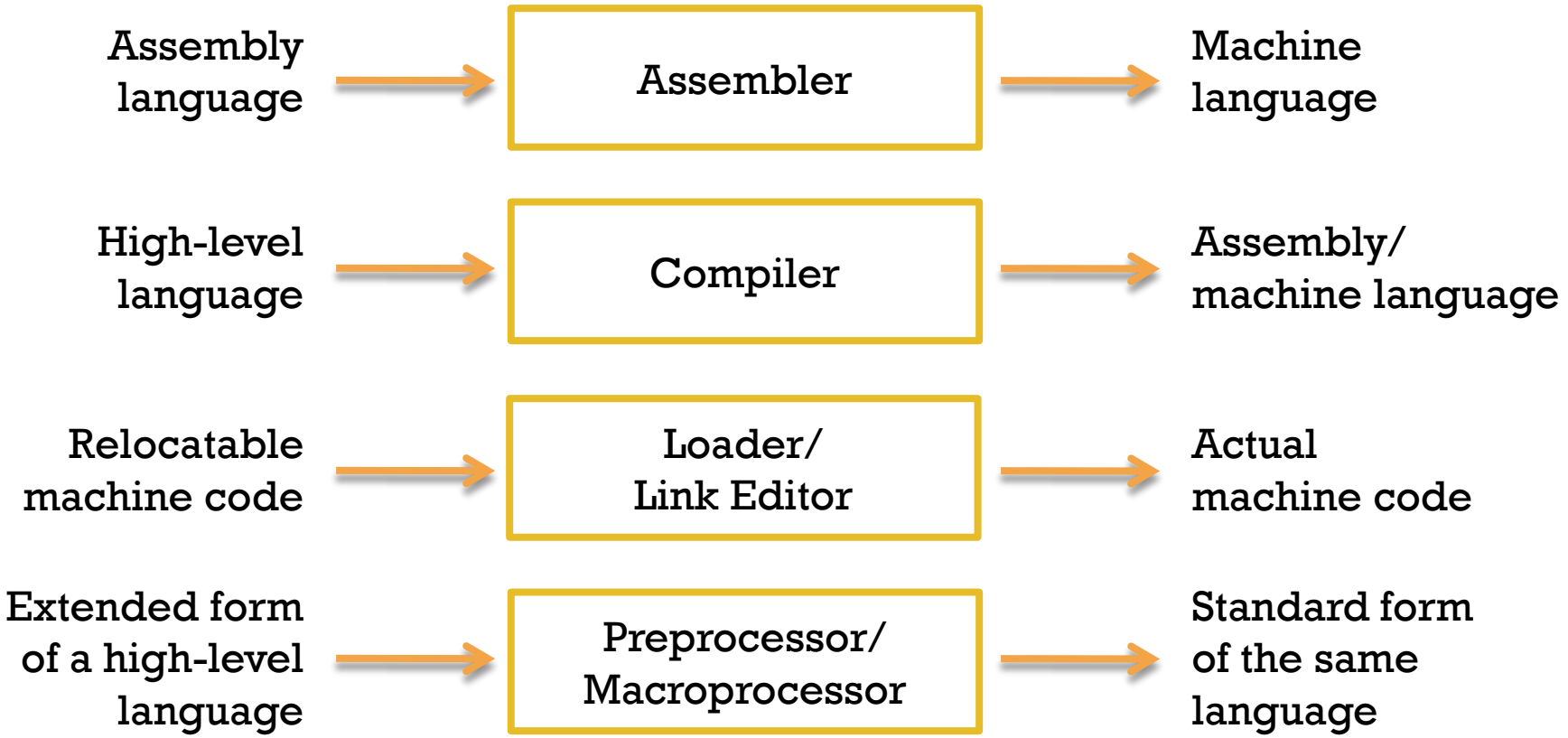
Programming Paradigms

Figure 6.2 The evolution of programming paradigms



Translators

Translators



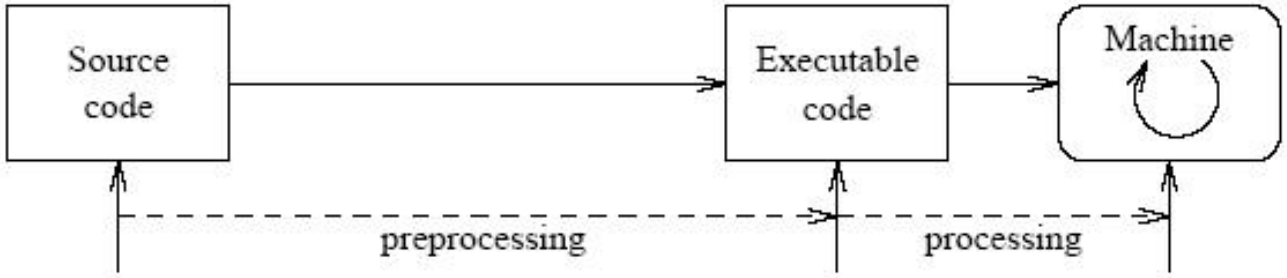
Execution models

1. Compilation
   - The high-level language is translated into another language usually assembly or machine.
   - The translator is called a compiler.
   - Approaches:
     - High-level **->** Machine
     - High-level **->** Assembly **->** Machine
     - High-level **->** Intermediate **->** Assembly or Machine

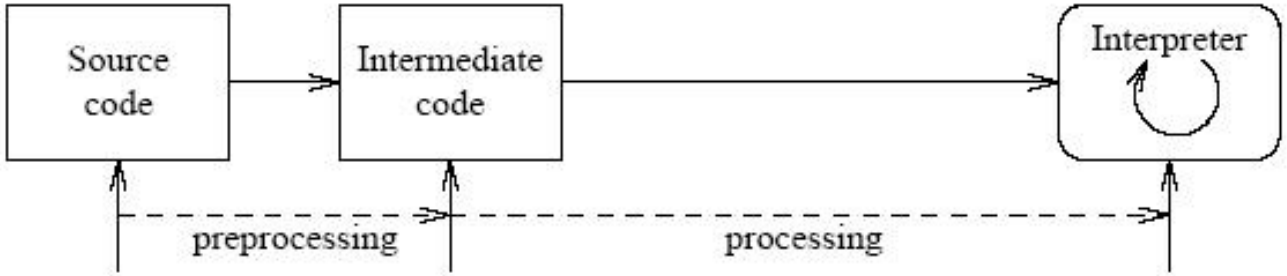
Execution models

2. Interpretation
   - Method that simulates, through a program running on another host computer, a computer whose machine language is the high-level language.
   - It is as if the instructions of the high-level language is executed directly.

Compilation vs. Interpretation



Compilation



Interpretation

Compilation vs. Interpretation

- Compiler characteristics:
  - spends a lot of time analyzing and processing the program
  - the resulting executable is some form of machine- specific binary code
  - the computer hardware interprets (executes) the resulting code
  - program execution is fast
- Interpreter characteristics:
  - relatively little time is spent analyzing and processing the program
  - the resulting code is some sort of intermediate code
  - the resulting code is interpreted by another program
  - program execution is relatively slow