Principles of Programming Languages

Weng Kai

Chocolate Cake Receipt

- Materials:
  - 1/2 cup butter
  - 4 ounce bittersweet chocolate
  - 2 eggs
  - 2 egg yolks
  - 1/4 cup white sugar
  - 2 teaspoons all purpose flour
- Step:
  - Preheat oven to 450°F.
  - Heat butter and chocolate until chocolate is almost melted.
  - Beat eggs, yolks and sugar until light colored and thick.
  - Mix chocolate and butter, and slowly pour into egg mixture, stirring constantly. Stir in flour until just combined.
  - Pour batter into molds and bake for 6 to 7 minutes. Invert molds on plates, let sit 15 seconds, and unmold. Serve with whipped cream.

A Receipt Is Like a Program

Receipt:

- tells you how to make a chocolate cake
- Has inputs (butter, eggs, chocolate, flour, sugar) & output (chocolate cake)
- Define a procedure
- Instruct how processors (oven, mixer) process inputs to generate output
- Can be expressed in different languages

Program:

- You tell a computer how to do a computation
- Has inputs and outputs
- Define a procedure (algorithm)
- Instruct how processors process inputs to generate outputs
- Can be expressed in different languages

Questions

- Given two languages, how do they differ in expressing the same receipt/algorithm?
- Which language is better?

  → How to evaluate "goodness" of languages?

- Why are there so many different languages?
- What is "programming language" anyway?
- Why does a programming language have so many different features?
- How are these features implemented?
- …

A Programming Language Is …

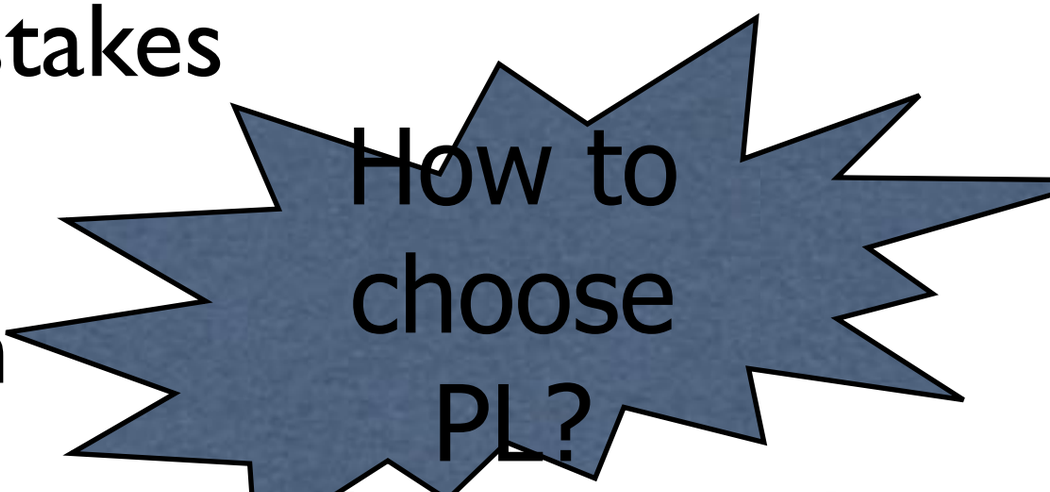
- An artificial language designed to express computations or algorithms that can be performed by a computer -- Wikipedia
  - A language is a means of expressing your thoughts to others
  - In the case of PL, it is a means of expressing your thoughts (algorithms) to a computer
  - Natural languages such as Chinese and English are not used because they cannot be easily translated into machine language executable by the computer

→ Keywords: expressiveness, implementation

- 编程语言是一种工作媒介，用于描述（可能非常复杂的）抽象机制
  - 一种形式化的描述方法
    - 《A Discipline of Programming》, by Edsger W. Dijkstra

Why PPL Important?

- A language is a framework for problem-solving
  - It may facilitate or hinder your thoughts and, thus, the abilities to solve problems
  - It may help you make fewer mistakes
  - Example: tense and gender, e.g. "He was doing great!" in English
  - Example: a C language that supports only static and global variables → no malloc()
    - How to implement hash table? linked list?



(Ref.: John Mitchell, http://www.stanford.edu/class/cs242)

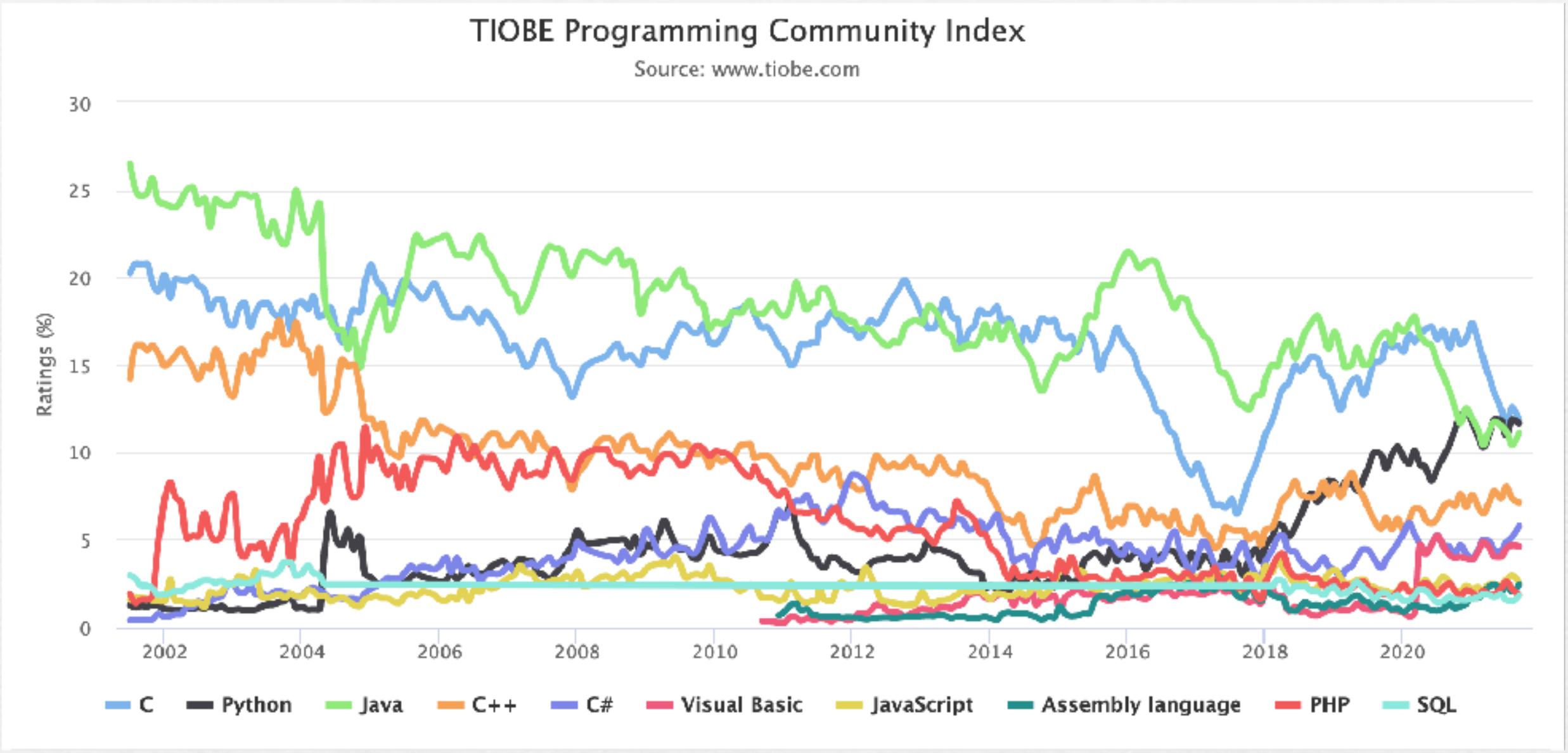
Top Languages

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	15.95%	+0.74%
2	1	▼	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	▲	PHP	2.49%	+0.62%
9	19	⏫	R	2.37%	+1.33%
10	8	▼	SQL	1.76%	-0.19%

图片来源：http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Sep 2021	Sep 2020	Change	Programming Language	Ratings	Change
1	1		C	11.83%	-4.12%
2	3	▲	Python	11.67%	+1.20%
3	2	▼	Java	11.12%	-2.37%
4	4		C++	7.13%	+0.01%
5	5		C#	5.78%	+1.20%
6	6		Visual Basic	4.62%	+0.50%
7	7		JavaScript	2.55%	+0.01%
8	14	⇈	Assembly language	2.42%	+1.12%
9	8	▼	PHP	1.85%	-0.64%
10	10		SQL	1.80%	+0.04%

Languages



图片来源：http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Important to Know PPL by Trend

- Increasing use of type-safe languages: Java, C#, …
- Scripting languages for web applications with increasing client-side functionality
- More on expressing algorithms than syntax
- Runtime environment and virtualization with continuous compilation, analysis, and checking
- More program analysis abilities: automated error detection and recovery

(Ref.: John Mitchell, http://www.stanford.edu/class/cs242)

Important to Know PPL by Tradeoffs

Factors influencing programming language

- Expressiveness:
  - Application domains
  - Programming methods: multiprogramming, interactive systems,…
- Implementation: efficiency
  - Computer architecture, OS, toolchain, library
  - Every convenience has its cost; must recognize cost of presenting an abstract view of machine
    - Understand trade-offs in programming language design

(Ref.: M. Sirjani, http://ut.ac.ir/classpages/ProgrammingLanguages)

PPL as a Course

- What is not
  - Do not teach you a programming language
  - Do not teach you how to program
- What is
  - Introduce fundamental concepts of programming languages
  - Discuss design issues of various language constructs
  - Examine design/implementation choices for these constructs
  - Compare design alternatives
- Need to be familiar in at least one PL

Why Study PPL?

- To improve your ability to develop effective algorithms and to use your language
  - O-O features, recursion
  - Call by value, call by reference
- To allow a better choice of PL
- Increased ability to learn new languages
- To make it easier to design a new language
- To understand significance of implementation
  - E.g. the efficiency of a recursive function

林汉钊

17:47

最近在 Google 要做的事用到了之前 ppl 那个 js 解释器的知识，受益匪浅

18:15

我们做的事情在国内非常的没有基础

我最近要做 SQL 的 parse 和 type checking

套路和那个解释器其实差不多

所以这门课很赞...很实用

林汉钊

和 type checking

不多

谢谢！我要截屏给下一届的孩子们看

？？要截屏？补上这个
课友内推联系 magical@google.com

18:25

教材

- 《编程语言原理（第10版）》
- 《程序设计语言——实践之路》
- 《编程语言实现模式》
- 《程序设计语言的形式语义》
- 《计算机程序的构造和解释》

目标

- 理解语言的概念、语法与语义的不同
- 了解编程语言的发展历史和当前研究方向
- 掌握用BNF和EBNF来描述语言和推导语句
- 理解命令式、函数式和逻辑式语言的定义和特点
- 理解程序执行的编译和解释两种方式
- 掌握变量的名-值关系、类型、运算与实现（包括编译和解释）
- 掌握命令式语言中的控制结构及编译实现
- 掌握函数的实现，尤其是局部空间的实现
- 理解结构化、面向对象、基于构件、泛型四种设计范式的定义和特点
- 理解虚拟机机制，了解JVM的实现机制
- 理解函数式编程概念和常用手段
- 理解并行计算概念和常用手段
- 掌握递归计算概念和常用手段

内容

- 编程语言基本概念
- 编程语言基本元素
- 编程语言设计范式与实现
- 函数式计算、并行计算和递归计算

作业和考试

- 论文阅读、撰写文献综述（个人）
- 单元小作业（个人）
- 编写函数式语言的解释器（个人）
- 期末考试

Assessment

- 课内讨论与测验：5%
- 单元小作业：15%（具体内容待定）
- 文献综述：10%
- 解释器：30%
- 期末考试：40%

单元作业I

- 用BNF表述语言
- MUA解释器的第一阶段设计

单元作业II

- 理解编译结果（汇编级）
- MUA解释器的第二阶段设计

单元作业III

- 待定
- MUA解释器的第三阶段设计

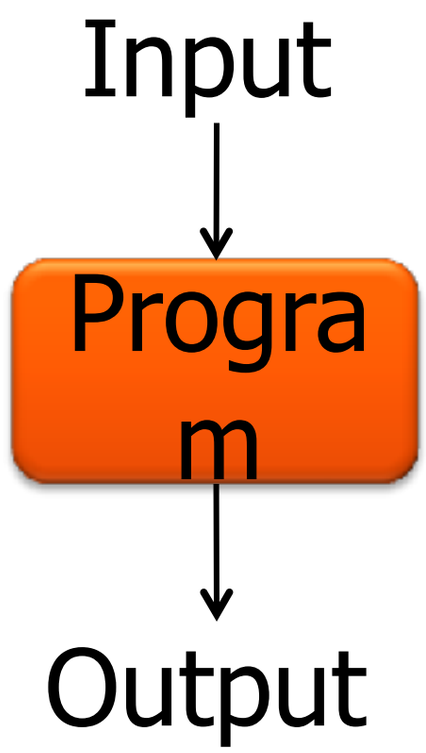
单元作业IV

- 文献综述
- 完成MUA解释器

Programming Language

- A programming language is an artificial language designed to express computations or algorithms that can be performed by a computer -- Wikipedia

- A program is computer coding of an algorithm that
  - Takes input
  - Performs some calculations on the input
  - Generates output



Models of Programming Languages

Programming is like …

程序是规则的表达?

```
int x;
int y = 3*x;
x = readInt();
print(y);
```

- 书上明明有写“程序是顺序执行的”

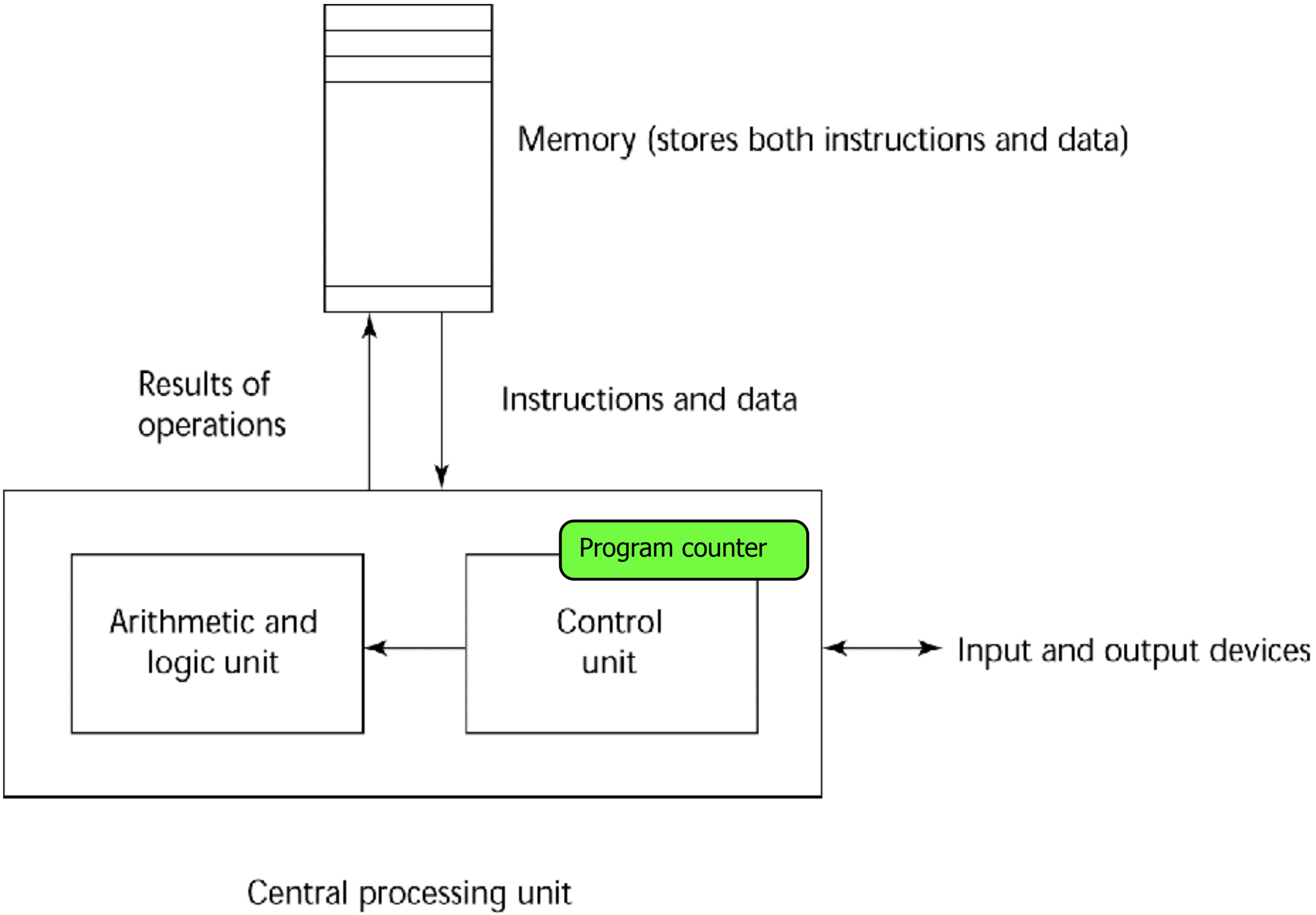
1st View: Imperative & Procedural

- Computers take commands and do operations
- Thus, programming is like …
  issuing procedural commands to the computer
  - Example: a factorial function in C

```
int fact(int n) {
   int sofar = 1;
   while (n>0) sofar *= n--;
   return sofar;
}
```

- Since almost all computers today use the von Neumann architecture → PL mimic the arch.

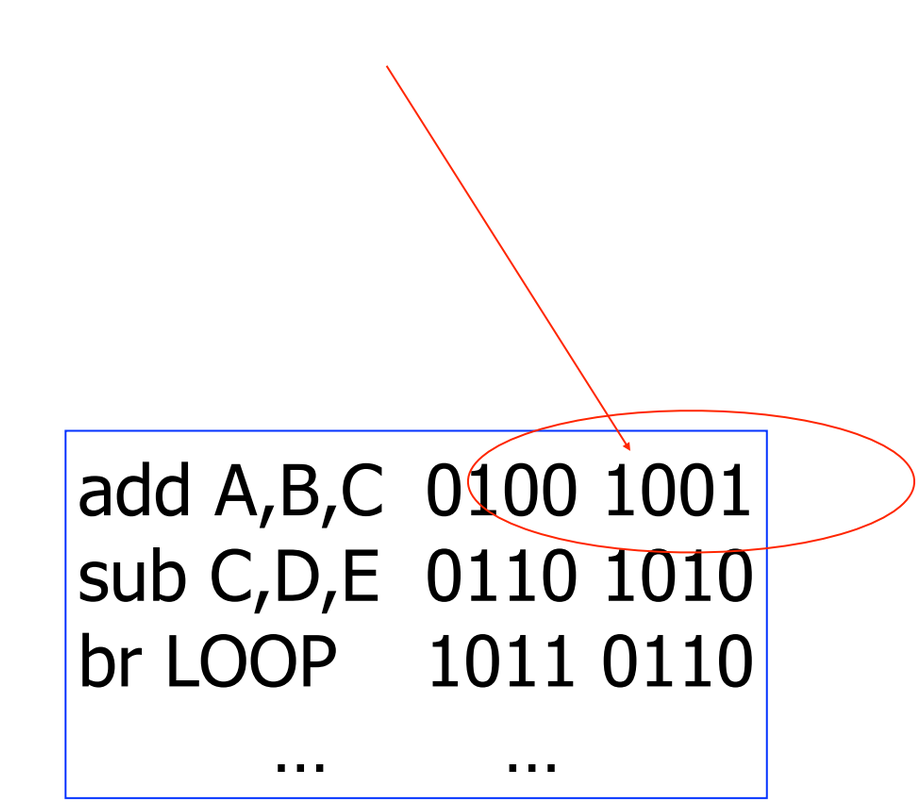
von Neumann Architecture



von Neumann Architecture

- Key features:
  - Data and programs stored in memory
  - Instructions and data are piped from memory to CPU
  - Fetch-execute-cycle for each machine instruction

```
initialize the program counter (PC)

repeat forever

   fetch the instruction pointed by PC

   increment the counter

   decode the instruction

   execute the instruction

end repeat
```



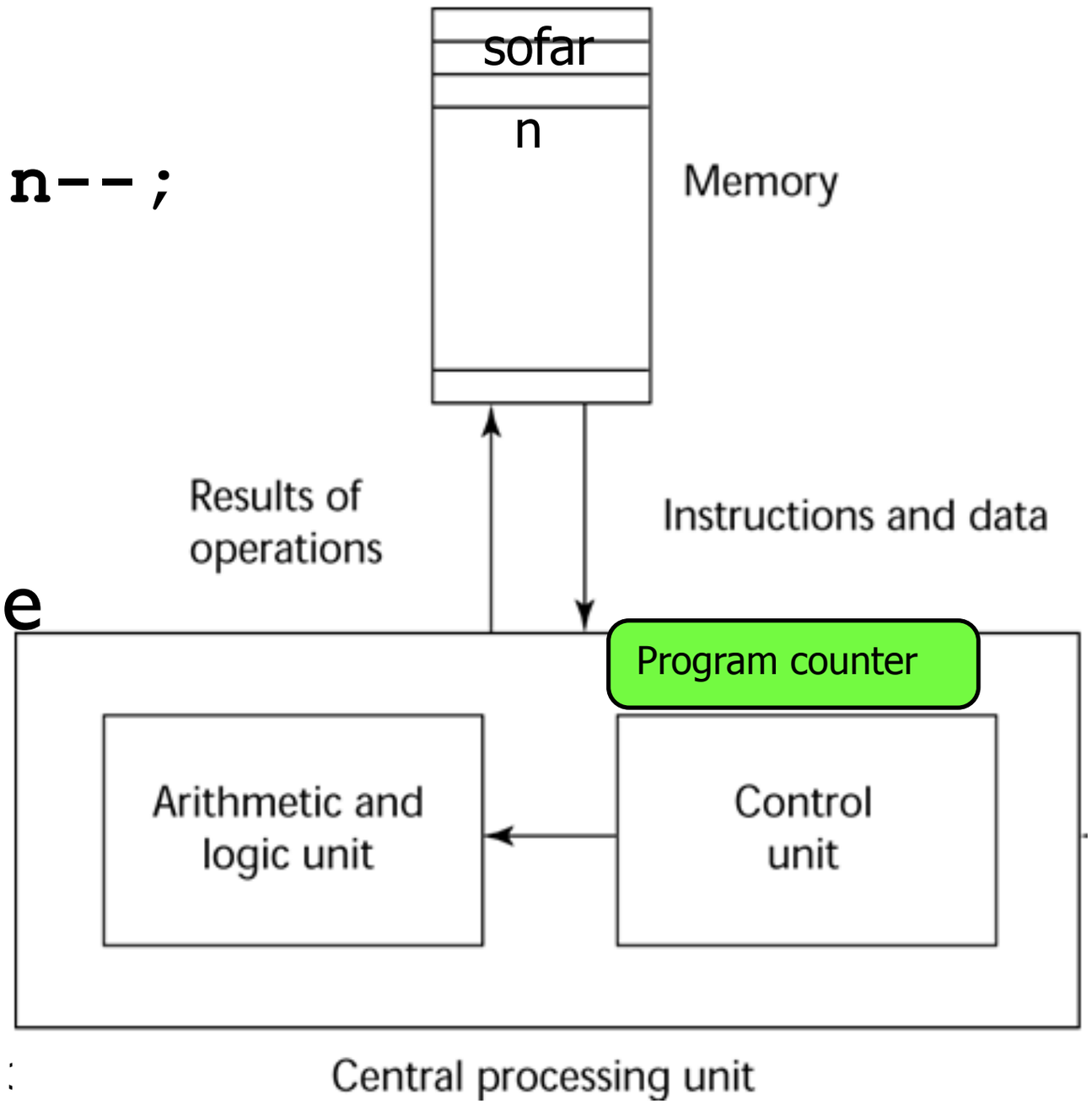
Imperative Language and Arch.

- Example: a factorial function in C

```
int fact(int n) {
   int sofar = 1;
   while (n>0) sofar *= n--;
   return sofar;
}
```

- Indicates that data **`n`**, **`sofar`**, and program code are stored in memory
- Program code instructs CPU to do operations

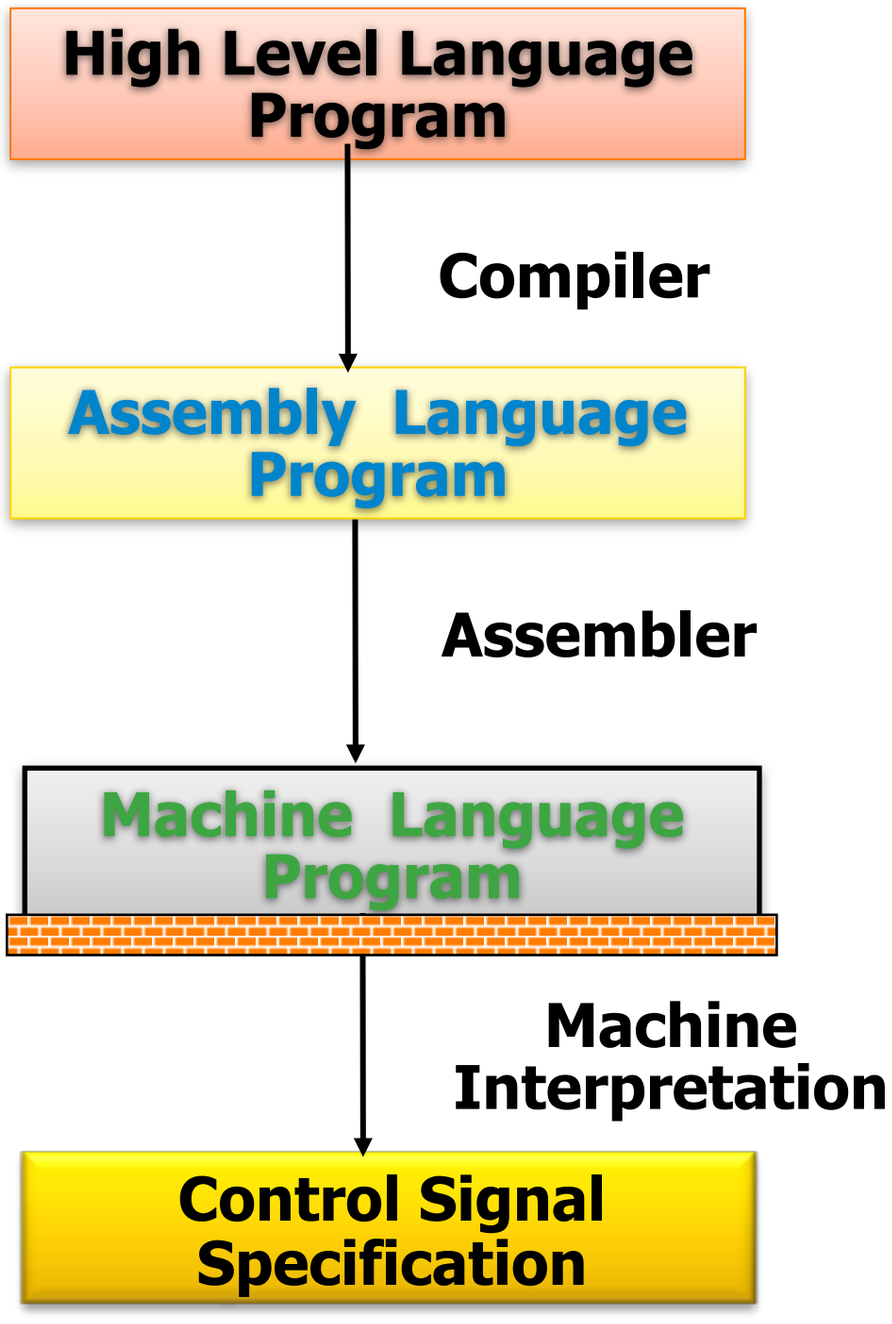
```
int fact(int n) {
  int sofar = 1;
  while (n>0) sofar *= n--;
  return sofar; }
```



Imperative Languages and Arch.

- Imperative languages, e.g., C, C++, Java, which dominate programming, mimic von Neumann architecture
  - Variables ←→ memory cells
  - Assignment statements ←→ data piping between memory and CPU
  - Operations and expressions ←→ CPU executions
  - Explicit control of execution flows ←→ prog. counter
- Allow efficient mapping between language and hardware for good execution performance, but limited by von Neumann bottleneck

Layers of Abstraction/ Translation




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15, 0($2)
lw      $16, 4($2)
sw$16, 0($2)
sw$15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

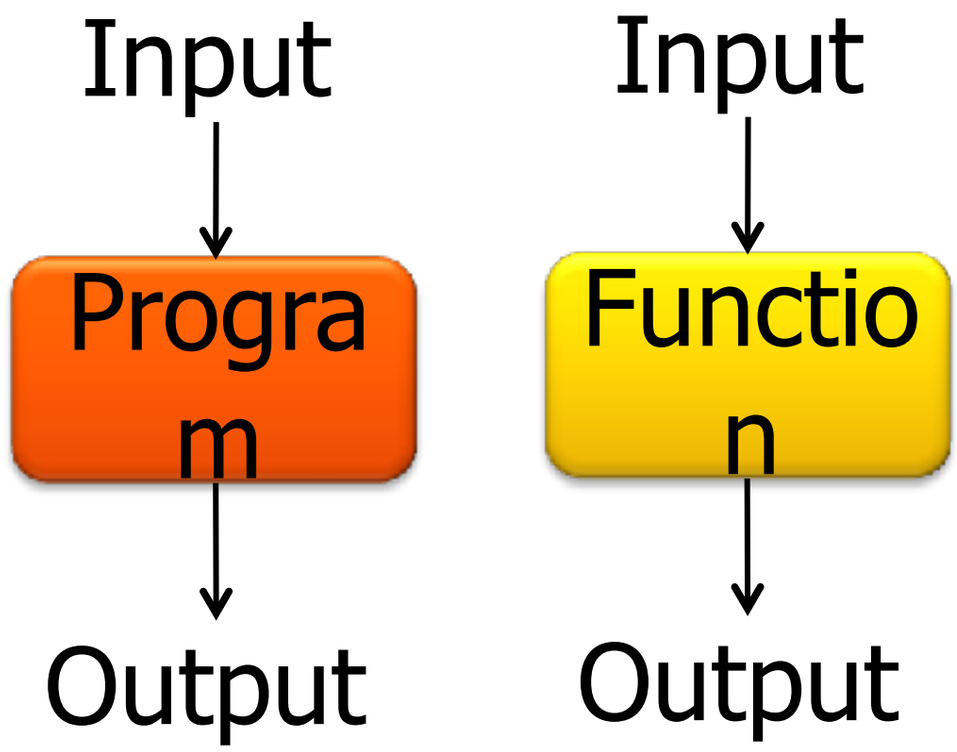
All imperative!

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

2nd View: Functional

- Programming is like …
  solving mathematical functions, e.g.,
  z = f(y, g(h(x)))
  - A program, and its subprograms, are just implementations of mathematical functions
  - Example: a factorial function in ML

```
fun fact x =
   if x <= 0
   then 1
   else x * fact(x-1);
```



Another Functional Language: LISP

- Example: a factorial function in Lisp

```
(defun fact (x)
  (if (<= x 0) 1 (* x (fact (- x 1)))))
```

  - Computations by applying functions to parameters

  - No concept of variables (storage) or assignment
    - Single-valued variables: no assignment, not storage
  - Control via recursion and conditional expressions
    - Branches → conditional expressions
    - Iterations → recursion
  - Dynamically allocated linked lists

- 2nd-oldest general-purpose PL still in use (1958)

3rd View: Logic

- Programming is like …
  logic induction
  - Program expressed as rules in formal logic
  - Execution by rule resolution
  - Example: relationship among people

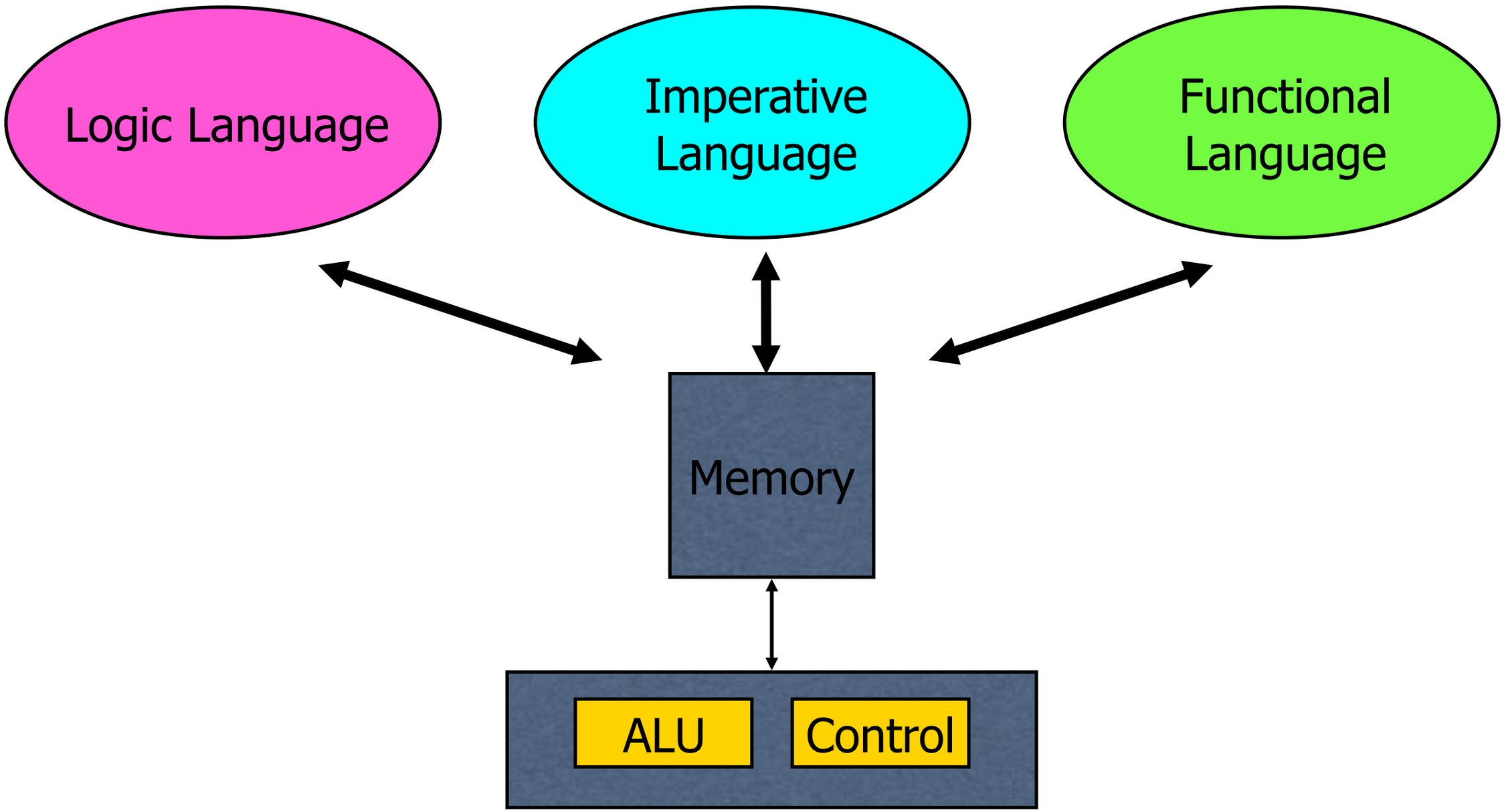
```
fact:   mother(joanne,jake).
        father(vern,joanne).

rule:   grandparent(X,Z)  :-
parent(X,Y),
                    parent(Y,Z).
goal:   grandparent(vern,jake).
```

Logic Programming

- Non-procedural
  - Only supply relevant facts (predicate calculus) and inference rules (resolutions)
  - System then infer the truth of given queries/goals
- Highly inefficient, small application areas (database, AI)
  - Example: a factorial function in Prolog

```
fact(X,1) :-  X =:= 1.
fact(X,Fact) :-
   X > 1, NewX is X - 1,
   fact(NewX,NF),
   Fact is X * NF.
```

Summary: Language Categories



von Neumann Architecture

Summary: Language Categories

- Imperative
  - Variables, assignment statements, and iteration
  - Include languages that support object-oriented programming, scripting languages, visual languages
  - Ex.: C, Java, Perl, JavaScript, Visual BASIC .NET
- Functional
  - Computing by applying functions to given parameters
  - Ex.: LISP, Scheme, ML
- Logic
  - Rule-based (rules are specified in no particular order)
  - Ex.: Prolog

单元作业1-1

- 根据学号最后一位数字确定论文
- 阅读指定的论文
- 展开延伸阅读