FISF130020: Introduction to Computer Science

Lecture 3: Programming Languages

Hui Xu

xuh@fudan.edu.cn

Outline

1. Overview of Programming Languages
2. Features of PL: Rust as an Example
3. In-class Practice

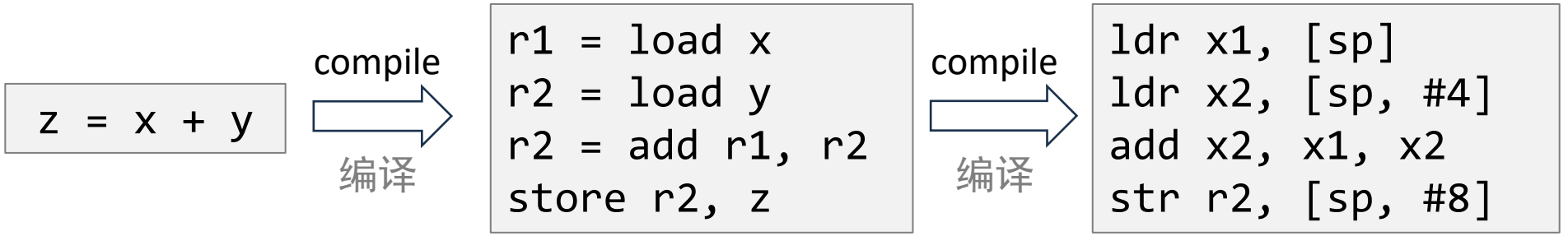
1. Programming Languages

Why (High-level) Programming Languages?

- Writing assembly code is painful for a large project.
- Reinventing a language is also important for different scenarios.

Comparison between Source and Assembly Code

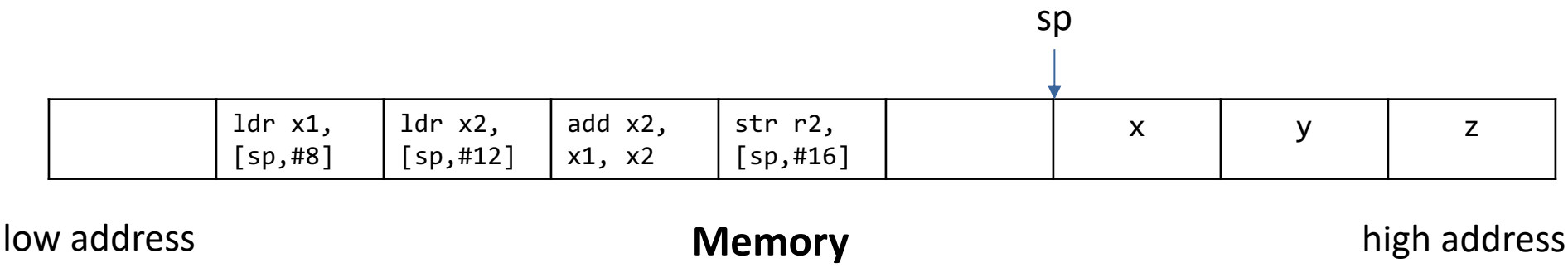
- Assembly code has no variable names, but only memory addresses.
- Access variables via a stack pointer (sp) + offset of the variable.



Source Code 代码

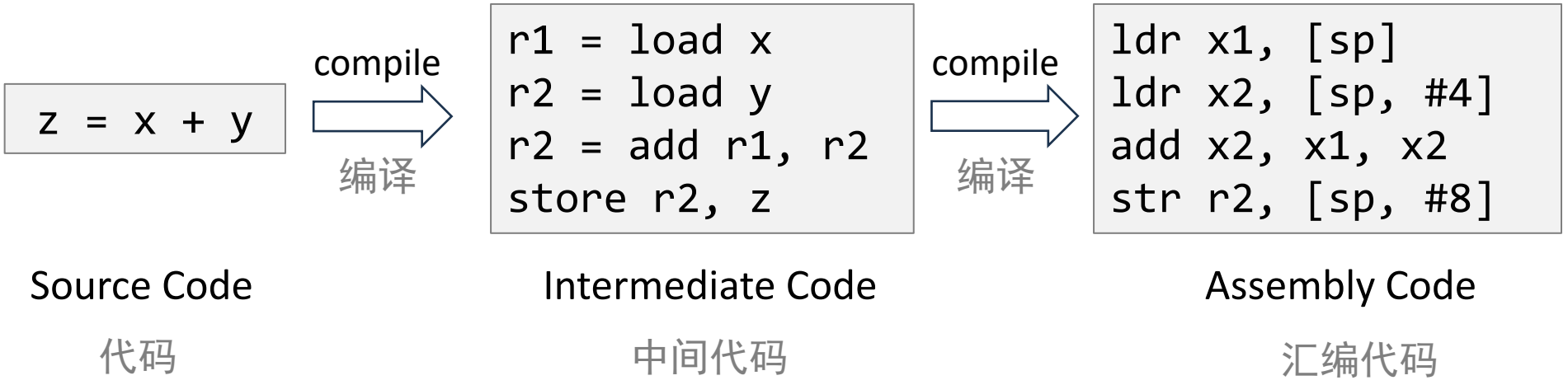
Intermediate Code 中间代码

Assembly Code 汇编代码



Features of Compilers

- Understanding: parse and interpret the source code
- Translation: generate equivalent assembly or intermediate code
- Optimization: improve the speed/size of the code

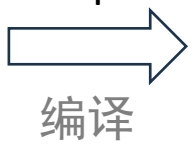


Case Study: Factorial in Rust

```
fn fac(n: u32) -> u32 {
    if n == 0 {
        1
    } else {
        n * fac(n - 1)
    }
}

fn main() {
    let num = 5;
    let r = fac(num);
    println!("{}", r);
}
```

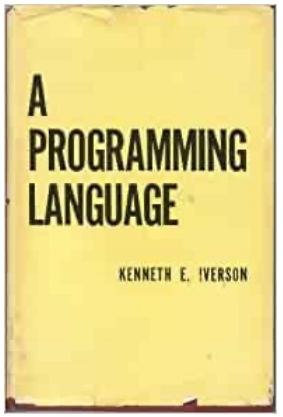
Rust Source Code




```
    sub sp, sp, #32
    stp x29, x30, [sp, #16]
    add x29, sp, #16
    str w0, [sp, #8]
    subs w8, w0, #0
    cset w8, ne
    tbnz w8, #0, LBB9_2
    b LBB9_1
LBB9_1:
    mov w8, #1
    stur w8, [x29, #-4]
    b LBB9_3
LBB9_2:
    ldr w8, [sp, #8]
    subs w9, w8, #1
    str w9, [sp, #4]
    subs w8, w8, #1
    cset w8, lo
    tbnz w8, #0, LBB9_5
    b LBB9_4
LBB9_3:
    ...
```

Concept of Programming Language

"Applied mathematics is largely concerned with the design and analysis of explicit procedures for calculating the exact or approximate values of various functions. Such explicit procedures are called algorithms or programs. Because an effective notation for the description of programs exhibits considerable syntactic structure, it is called a programming language."



- Kenneth Iverson

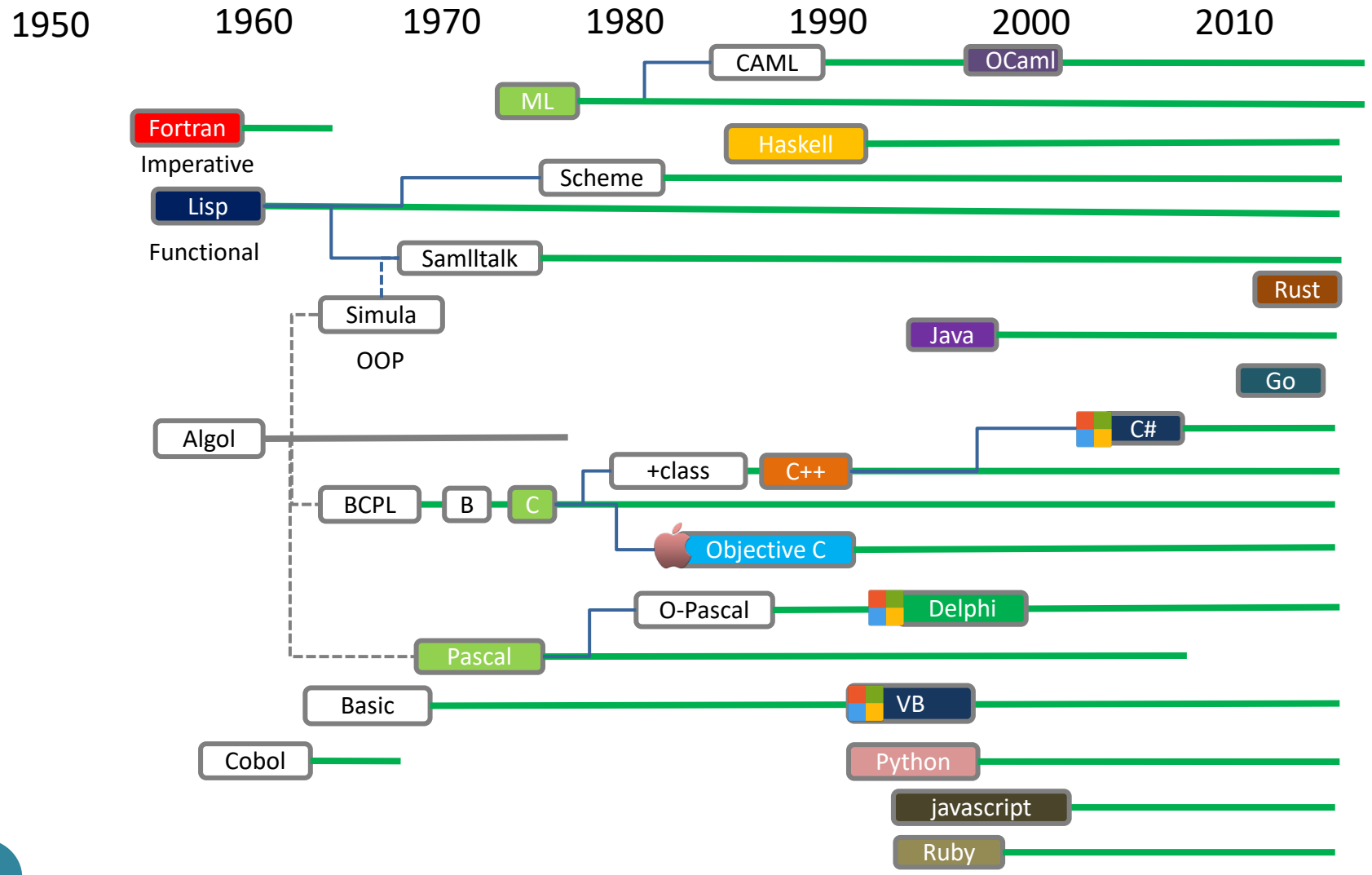
What Language do You Speak?

TIOBE Index

Aug 2024	Aug 2023	Programming Language	Ratings	Change
1	**1**	**Python**	**18.04%**	**+4.71%**
2	3	C++	10.04%	-0.59%
3	2	C	9.17%	-2.24%
4	4	Java	9.16%	-1.16%
5	5	C#	6.39%	-0.65%
6	6	JavaScript	3.91%	+0.62%
7	8	SQL	2.21%	+0.68%
8	7	Visual Basic	2.18%	-0.45%
9	12	Go	2.03%	+0.87%
10	14	Fortran	1.79%	+0.75%
11	13	MATLAB	1.72%	+0.67%
12	23	Delphi/Object Pascal	1.63%	+0.83%
13	10	PHP	1.46%	+0.19%
14	**19**	**Rust**	**1.28%**	**+0.39%**
15	17	Ruby	1.28%	+0.37%
16	18	Swift	1.28%	+0.37%
17	**9**	**Assembly language**	**1.21%**	**-0.13%**
18	27	Kotlin	1.13%	+0.44%
19	16	R	1.11%	+0.19%
20	11	Scratch	1.09%	-0.13%

https://www.tiobe.com/tiobe-index/

Genealogy of Programming Languages



Comparison among Different Languages

```rust
fn fac(n: u32) -> u32 {
    if n == 0 {
        1
    } else {
        n * fac(n - 1)
    }
}
```

Rust Code

```cpp
int fac(int n) {
    if (n == 0)
        return 1;
    else {
        int r = n * fac(n - 1);
        return r;
    }
}
```

C/C++ Code

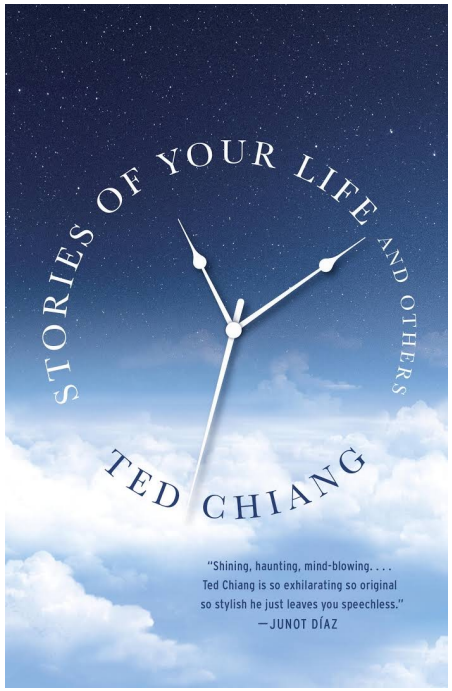
```python
def factorial(n):
    if n == 0:
        return 1
    else:
        r = n * fac(n - 1)
        return n * fac(n - 1)
```

Python Code

$$f(x) = \begin{cases} 1, & x = 1 \\ x * (f(x) - 1), & x > 1 \end{cases}$$

Math Formula

Sapir-Whorf Hypothesis



降临 Arrival (2016)

导演: 丹尼斯·维伦纽瓦
编剧: 埃里克·海瑟尔 / 姜峯楠
主演: 艾米·亚当斯 / 杰瑞米·雷纳 / 福里斯特·惠特克 / 迈克尔·斯图巴 / 马泰 / 更多...
类型: 剧情 / 科幻
官方网站: www.arrivalmovie.com
制片国家/地区: 美国 / 加拿大
语言: 英语 / 俄语 / 汉语普通话
上映日期: 2017-01-20(中国大陆) / 2016-09-01(威尼斯电影节) / 2016-11-11(美国)
片长: 116分钟
又名: 天煞异降(港) / 异星入境(台) / 你一生的故事 / 抵达 / 抵达者 / Story of Your Life
IMDb: tt2543164

豆瓣评分

7.8 ★★★★☆ 380703人评价

5星 23.1%
4星 47.4%
3星 25.5%
2星 3.2%
1星 0.8%

好于 85% 科幻片
好于 66% 剧情片

Programming with Natural Language/Pseudocode

ChatGPT

Write a Rust program to calculate the value of y based on the input x. The rule is that if x equals 1, y equals 1; otherwise, y equals x times the result of y for x - 1.

This rule describes the calculation of a **factorial** function, where the value of `y` is the factorial of `x`. Here's a Rust program that implements this logic:

Rust Code:

rust Copy code

```rust
// Function to calculate factorial recursively
fn factorial(x: u32) -> u32 {
    if x == 1 {
        1
    } else {
        x * factorial(x - 1)
    }
}
```

Ambiguity of Code

A programmer’s wife asks him to go to the grocery. She says "Get a gallon of milk. If they have eggs, get 12."

The programmer returns with 12 gallons of milk.

Ambiguity of Code: Example

- What is the return value if a = 1, b = 0?

```
int fac(int n) {
    if (a>0)
        if (b>0)
            return 0;
    else
        return 1;
    return 2;
}
```

C/C++ Code

```
int fac(int n) {
    if (a>0) {
        if (b>0) {
            return 0;
        }
    } else {
        return 1;
    }
    return 2;
}
```

Interpretation 1

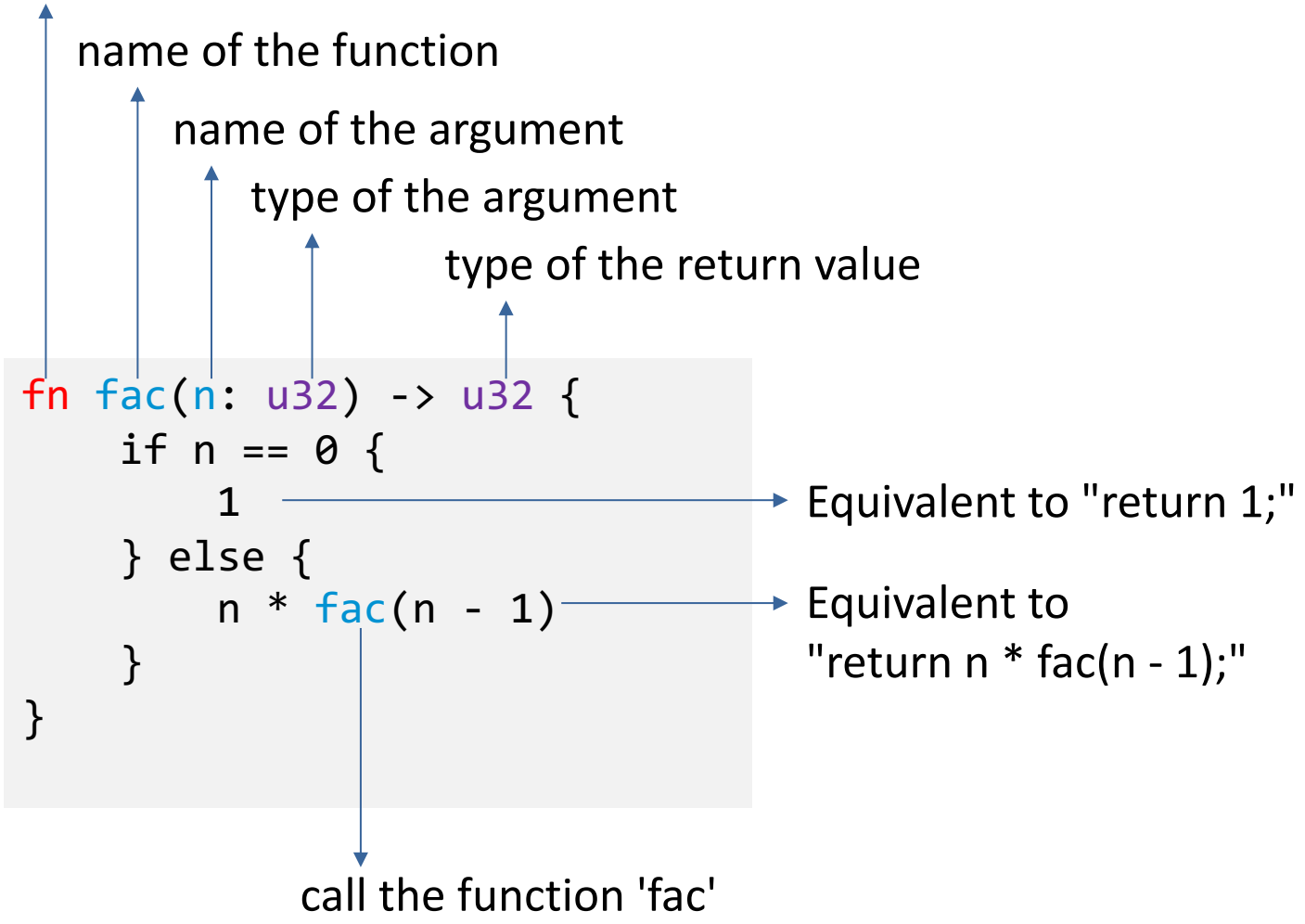
```
int fac(int n) {
    if (a>0) {
        if (b>0) {
            return 0;
        } else {
            return 1;
        }
    }
    return 2;
}
```

Interpretation 2

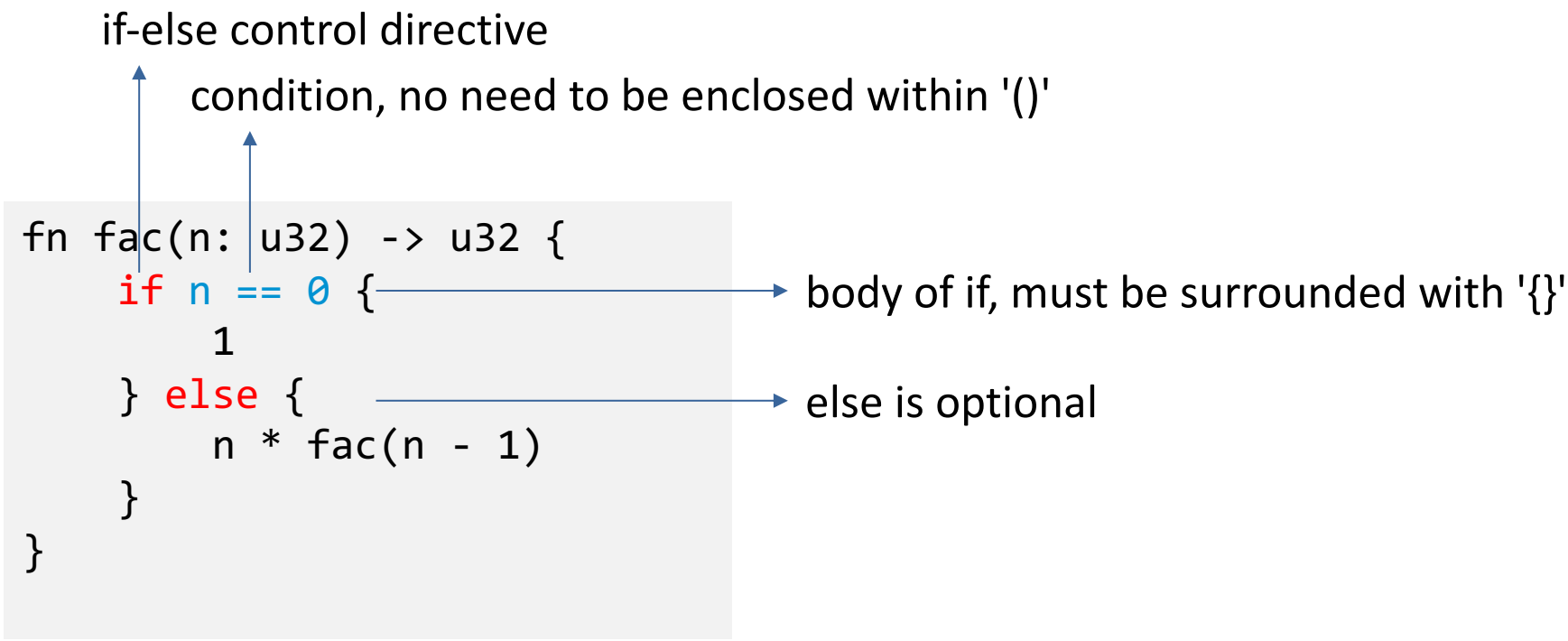
2. Features of PL: Rust as an Example

Syntax: Function

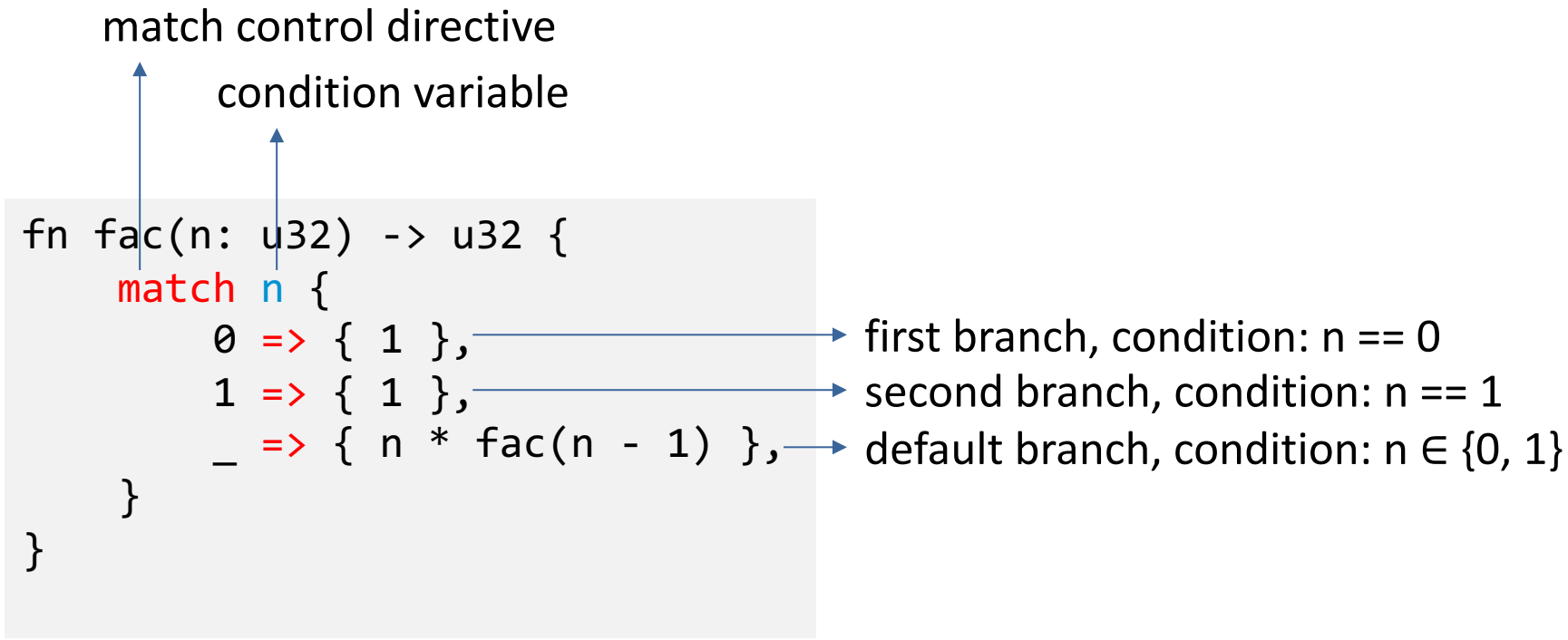
define a function: start with the 'fn' keyword



Syntax: If-Else Condition Flow



Syntax: Match-Case Control Flow



Syntax: Variable

declare/define a function: start with the 'let' keyword

declare the mutability of the variable; immutable if no 'mut' provided

name of the variable

```
fn fac(n: u32) -> u32 {
    let mut result = 1;

    for i in 1..=n {
        result *= i;
    }

    result
}
```

modify the value of the variable;
only allowed for mutable variable

Syntax: Data Structure

```
struct Person {
    name: String,
    tel: String,
}
impl Person {
    fn new(name: String, tel: String) -> Person {
        Person { name, id, birth, tel }
    }
    fn display(&self) {
        println!("Name: {}, Tel: {}", self.name, self.tel);
    }
}
fn main() {
    let alice = Person::new(
        String::from("Alice"),
        String::from("555-1234")
    );
    alice.display();
}
```

start with 'struct', followed by the name of the struct

fields of the struct in the format of (field name : type,)

implement the features of a struct

constructor

method of the struct

construct an object of the struct

call the method via the object

Syntax: Trait

- Shared behavior among multiple types

```
trait Human {
    fn speak(&self);
    fn eat(&self);
}
impl Human {
    ...
}
trait Kid: Human {
    fn play(&self);
}
impl Human for Person {}
fn main() {
    let alice = Person::new(
        String::from("Alice"),
        String::from("555-1234")
    );
    alice.speak();
}
```

start with 'trait', followed by the name of the trait

methods of the trait

implement the methods of the trait; similar as struct

implement the 'Human' trait for struct 'Person'

invoke the trait method 'speak'

Semantic: Ownership and Borrow (&)

- Each value is owned by one variable
- Borrowed ownership will be returned automatically if no longer used
- Each value can be immutably borrowed by several variables
- Rule: it cannot be both borrowed and mutable at the same time

```
let mut alice = Box::new(1);
let bob = &alice;
//*alice = 2;
println!("bob:{}", bob);
println!("alice:{}", alice);
```

bob borrows the ownership

mutation is not allowed as alice is borrowed

bob returns the ownership to alice automatically

Example of Mutability

```
let mut alice = 1;
alice+=1;
```

mutable object

```
let alice = 1;
alice+=1;
```

immutable object

```
let mut alice = 1;
let bob = &mut alice;
*bob+=1;
```

mutable borrow of a mutable object

```
let mut alice = 1;
let bob = &alice;
*bob+=1;
```

immutable borrow of a mutable object

```
let alice = 1;
let bob = &mut alice;
*bob+=1;
```

mutable borrow of an immutable object

Example of Mutability cont'd

```
let mut alice = 1;
let mut carol = 1;
let mut bob = &mut alice;
*bob+=1;
bob = &mut carol;
*bob+=1;
```

mutable object bob + mutable borrow

```
let mut alice = 1;
let mut carol = 1;
let bob = &mut alice;
*bob+=1;
bob = &mut carol;
*bob+=1;
```

immutable object bob + mutable borrow

bob cannot be modified

Semantic: Move (=)

- If a type is Copy (impl the Copy trait), move only copies the value
- If a type is not Copy, move transfers the ownership.
  - e.g., Box<T> is not copy

```
let mut alice = 1;
let bob = alice;
alice = 2;
println!("bob:{}", bob);
```

alice owns the object

copy the object to bob


```
let mut alice = Box::new(1);
let bob = alice;
println!("bob:{}", bob);
println!("alice:{}", alice);
```

alice owns the object

transfer the onwership to bob;
alice loses the ownership

not allowed

More Features of Rust

- RAII: Each variable is initialized upon creation and dropped when no longer in use.
- Lifetime: Constraints on variable liveness.
- Unsafe: Code that may incur undefined behaviors.
- Raw pointer: A type of unsafe code used for low-level control.
- Reference counter: A type of smart pointer.
- Interior mutability: Wrapping unsafe code within a safe abstraction.
- Generic: A function, struct, or trait that can operate on multiple types.
- Functional programming: Functions can be treated as first-class variables.
- Closure: An anonymous function, also known as lambda calculus.
- …

In-class Practice

Practice

- Design a question and prompt an LLM (GPT/Qianwen/Doubao) to write a Rust program for you.
  1) Verify the correctness of the code by compiling and executing it (e.g., via Rust Playground).
  2) Explain the meaning of each line of code.

https://www.doubao.com
https://tongyi.aliyun.com