中国科学技术大学
University of Science and Technology of China

引 论

《编译原理和技术》

张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

ACM图灵奖

https://amturing.acm.org/bysubject.cfm

- 编程语言、编译相关的获奖者是最多的 占约1/3

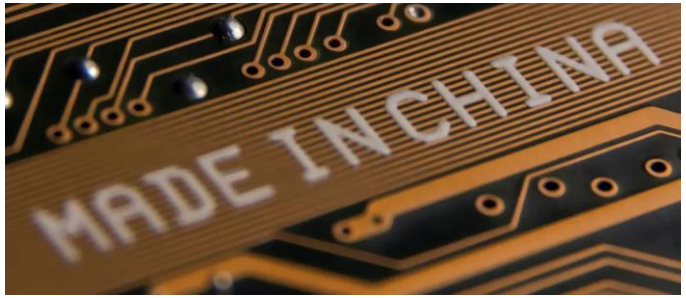
Analysis of Algorithms **Artificial Intelligence**
Combinatorial Algorithms Compilers **Computational Complexity**
Computer Architecture Computer Hardware **Cryptography**
Data Structures Databases Education Error Correcting Codes Finite Automata Graphics
Interactive Computing Internet Communications List Processing Numerical Analysis
Numerical Methods Object Oriented Programming Operating Systems Personal Computing
Program Verification Programming
Programming Languages Proof Construction Software
Theory Software Engineering
Verification of Hardware and Software Models Computer Systems Machine Learning
Parallel Computation

程序语言与编译系统发展的契机

- **人工智能的再次兴起，2021：人工智能的普及之年**
  - 人工智能加速芯片
  - 人工智能算法开发

  } 对程序语言与编译提出更高要求

- **国产芯片五年计划，2020年8月**
  - 到2025年将实现70%的芯片自给率
  - 2020年新增超过6万家芯片相关企业



- **➔面向应用/硬件的领域特定语言、软硬件协同的编译系统优化**

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. 编译技术的应用与挑战

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. 编译技术的应用与挑战

编程语言

- 什么是编程语言
  - A programming language is a notation for describing computations to people and to machines.
- 每种编程语言有自己的计算模型
  - 过程型(Procedural): C, C++, C#, Java，Go
  - 声明型(Declarative): SQL, …
  - 逻辑型(Logic): Prolog, …
  - 函数式(Functional): Lisp/Scheme, Haskell, ML, …
  - 脚本型(Scripting): AWK, Perl, Python, PHP, Ruby, …

求最大公约数 gcd

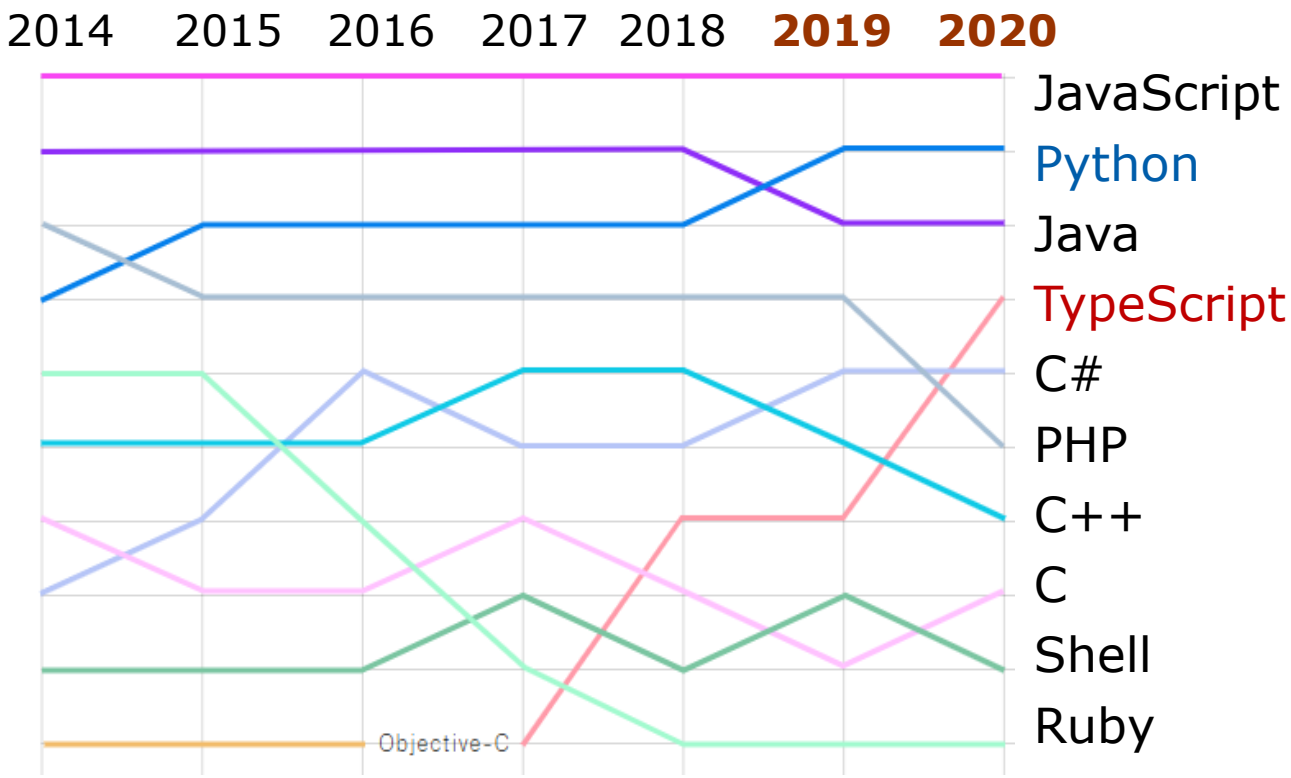
```c
int gcd(int a, int b) {                                  // C
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

```
let rec gcd a b =                                        (* OCaml *)
    if a = b then a
    else if a > b then gcd b (a - b)
         else gcd a (b - a)
```

```
gcd(A,B,G) :- A = B, G = A.                              % Prolog
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

编程语言众多且流行度在变化

- GitHub --开源项目涉及370种编程语言(2019.9)



- 2020
  Securing SW

https://octoverse.github.com/

编程语言众多且流行度在变化

https://octoverse.github.com/

- GitHub --开源项目涉及370种编程语言(2019.9)
- TIOBE

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

Aug 2021	Aug 2020	Change	Programming Language
1	1		C
2	3	↑	Python
3	2	↓	Java
4	4		C++
5	5		C#
6	6		Visual Basic
7	7		JavaScript
8	9	↑	PHP
9	14	⇈	Assembly language
10	10		SQL

编程语言
名人堂
2020，2018年

99 Bottles of Beer

http://www.99-bottles-of-beer.net/ 1500种语言编写

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.
.
.
.
2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.

[Traditional]

C: 99 Bottles of Beer

```
#define MAXBEER (99)
void chug(int beers);
main() {
  register beers;
  for(beers = MAXBEER; beers; chug(beers--))
    puts("");

  puts("\nTime to buy more beer!\n");
  exit(0);
}
void chug(register beers) {
  char howmany[8], *s;
  s = beers != 1 ? "s" : "";
  printf("%d bottle%s of beer on the wall,\n", beers, s);
  printf("%d bottle%s of beeeeer . . . ,\n", beers, s);
  printf("Take one down, pass it around,\n");

  if(--beers) sprintf(howmany, "%d", beers); else strcpy(howmany, "No more");
  s = beers != 1 ? "s" : "";
  printf("%s bottle%s of beer on the wall.\n", howmany, s);
}
```

[Bill Wein]

Java: 99 Bottles of Beer

```
class bottles {

  public static void main(String args[]) {
    String s = "s";

    for (int beers=99; beers>-1;) {
      System.out.print(beers + " bottle" + s + " of beer on the wall, ");
      System.out.println(beers + " bottle" + s + " of beer, ");

      if (beers==0) {
        System.out.print("Go to the store, buy some more, ");
        System.out.println("99 bottles of beer on the wall.\n");
        System.exit(0);
      } else
        System.out.print("Take one down, pass it around, ");

      s = (--beers == 1)?"":"s";
      System.out.println(beers + " bottle" + s + " of beer on the wall.\n");
    }
  }
}
```

[Sean Russell]

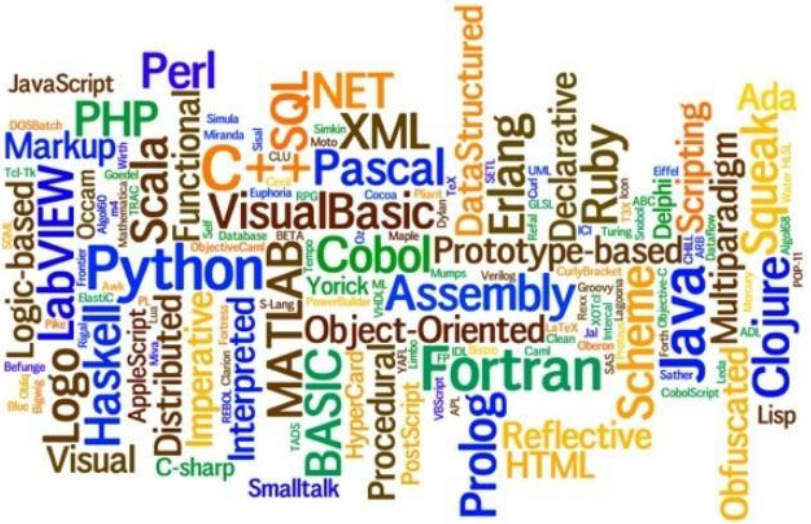
AWK: 99 Bottles of Beer

```
BEGIN {
  for(i = 99; i >= 0; i--) {
    print ubottle(i), "on the wall,", lbottle(i) "."
    print action(i), lbottle(inext(i)), "on the wall."
    print
  }
}
function ubottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s" : "")
}
function lbottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s" : "")
}
function action(n) {
  return sprintf("%s", n ? "Take one down and pass it around," : \
                         "Go to the store and buy some more,")
}
function inext(n) {
  return n ? n - 1 : 99
}
```

[Osamu Aoki, http://people.debian.org/~osamu]

编程语言不断演化和发展

- 编程语言自身在不断发展
  - CC90, C99, C11
  - C++ 1998,…, 2011, 2014, 2017, 2020



- 新语言不断产生
  - Go (2009), Rust (2010), Elixir (2011), Swift (2014)

领域特定语言

- 将高阶函数map、reduce等应用于大数据处理

  大数据处理：MapReduce, Hadoop,…
- 解耦计算的定义与调度实现

  图像处理：Halide (2014), …➔ 深度学习： TVM…
- 深度学习编程框架：TensorFlow, PyTorch, MindSpore, …

高阶函数→闭包

高阶函数在现代语言中被越来越多地支持

```
def outer(x):
    def inner(y):
        return x + y
    return inner
```

```
a = outer(2)
print('function:',a)
print('result:',a(3))
```

outer是返回函数inner的高阶函数

a得到函数inner

a(3) 调用时要计算 x+3

其中x是不在inner中定义的非局部变量

引入**闭包closure**：

将 x=2作为inner返回值的环境，形成闭包来返回

=>a(3) 调用时要计算 x+3，可从闭包中获取x的值

DSL领域特定语言

Halide: 面向图像处理的DSL

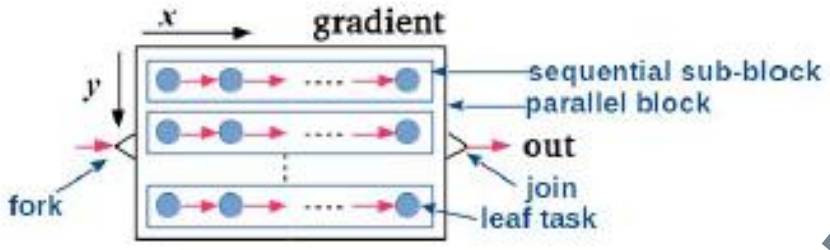
(a) Halide program example

```
Var x, y;
Func gradient;
gradient(x, y) = x + y;
gradient.parallel(y);
out = gradient.realize(1024, 1024);
```

计算的定义

计算的调度

(b) Scheduled task graph



(c) Intermediate representation

```
alloc gradient[1024][1024]
parallel for y in 0...1023:
  for x in 0...1023:
    gradient[y][x] = x + y
```

(d) Result after lowering parallel loop

```
define task_function(task_num, closure):
  gradient = unpacking(closure)
  for x in 0...1023:
    gradient[task_num][x] = x + task_num

alloc gradient[1024][1024]
closure = packing(gradient)
halide_do_par_for(task_function, 0, 1024, closure)
```

计算的定义与调度分离

编程语言的设计

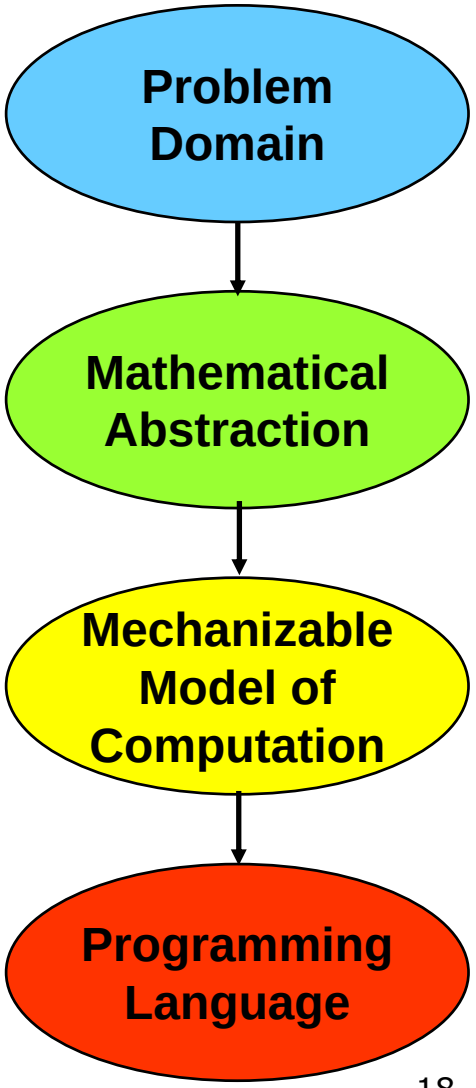
- 为什么那么多语言?
  - 单个语言不能适用所有应用
  - 程序员对语言的好坏、如何编程有自己的观点和看法
  - 没有评价语言好坏的普遍接受的标准
- 语言进化之驱动力
  - 应用的多样性
  - 提高软件开发生产力(productivity)
  - 改善软件的安全性、可靠性和可维护性
  - 支持并行(parallelism)与并发(concurrency)
  - 移动和分发、模块化、多范型

程序语言设计的计算思维

计算思维 (Computational Thinking)

Jeannette M. Wing
Computational Thinking
CACM, vol. 49, no. 3, pp. 33-35, 2006

Computational thinking is a **fundamental skill for everyone**, not just for computer scientists. To reading, writing, and arithmetic, **we should add computational thinking to every child's analytical ability**. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.



语言设计中的计算思维

主要内容

1. 编程语言及设计
2. **编译器及形式**
3. 编译器的阶段
4. 编译技术的应用与挑战

什么是编译?

- 程序语言
- 目标机器
- 编译系统

```c
#include <stdio.h>
int main()
{
  printf("hello, world!\n");
}

/* helloworld.c */
```

```
[root@host ~]# gcc helloworld.c -o helloworld
[root@host ~]# ./helloworld
hello, world!
```

注意：gcc是驱动程序
(根据命令行参数调用相应的处理程序)

编译系统的作用

- 翻译
  - 支持高层的编程抽象
  - 支持底层的硬件体系结构
- 优化
  - 更快的执行速度
  - 更少的空间
- 分析
  - 程序理解
  - Safety：自身的稳定状态，功能正确
  - Security：免受外部伤害

举例：性能与安全

```
for (i=0; i<n; i++)  a[i] = 1;
```

```
pend = a+n;
for (p=a; p<pend; p++) *p = 1;
```

哪个更快，Why？

```
foo (char * s)
{
    char buf[32];
    strcpy (buf, s);
}
```

调用foo( )会如何？

举例：性能与安全

```
for (i=0; i<n; i++)  a[i] = 1;
```

```
pend = a+n;
for (p=a; p<pend; p++) *p = 1;
```

哪个更快，Why？

```
foo (char * s)
{
    char buf[32];
    strcpy (buf, s);
}
```

调用foo( )会如何？

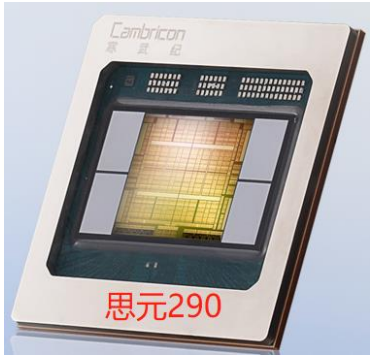
若s指向的串的长度超出31，则复制时会超出buf数组的有效区域

目标语言

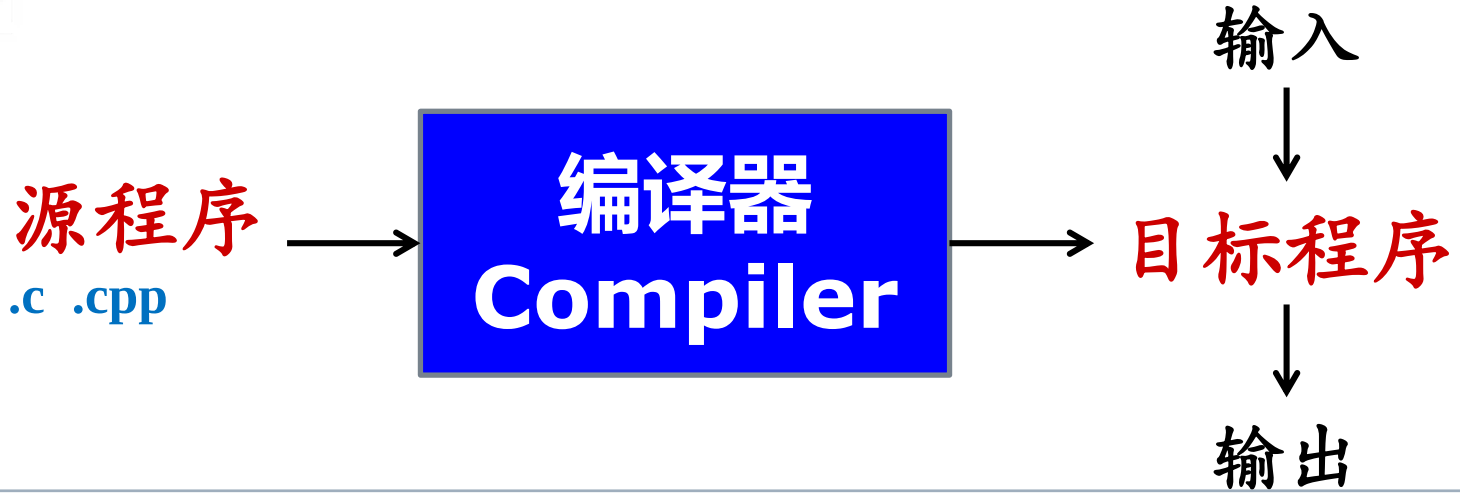
- 另一种编程语言
- CISCs（复杂指令集）：x86、IA64、…
- RISCs（精简指令集）：MIPS、ARM、LoongArch指令集、…
- 多核/众核
- GPUs : CUDA、OpenCL
- FPGAs
- 异构编程SYCL
- 量子计算机
- TPU，NPU
- …

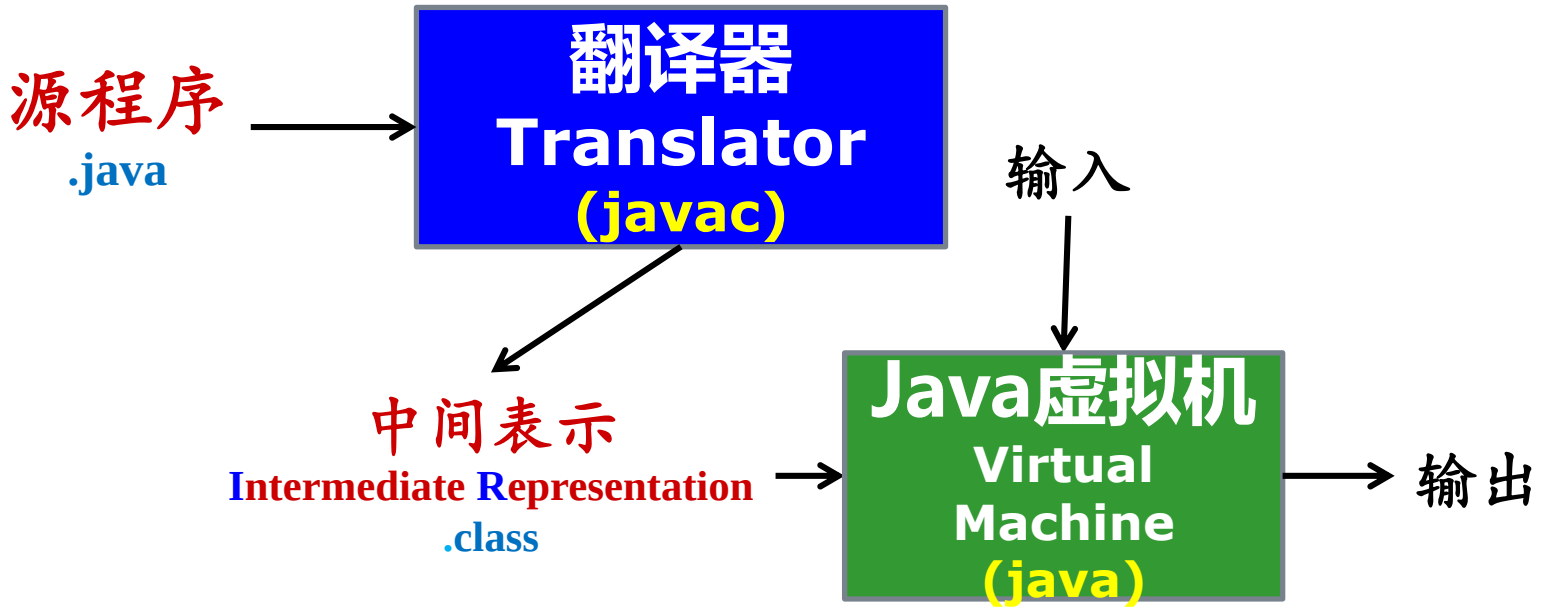
龙芯3A5000


龙芯3A5000计算机



编译器是什么


输入
源程序
.c .cpp
编译器
Compiler
目标程序
输出


源程序
.java
翻译器
Translator
(javac)
输入
中间表示
Intermediate Representation
.class
Java虚拟机
Virtual
Machine
(java)
输出

解释器

直接在输入上执行源程序

如Python等脚本语言

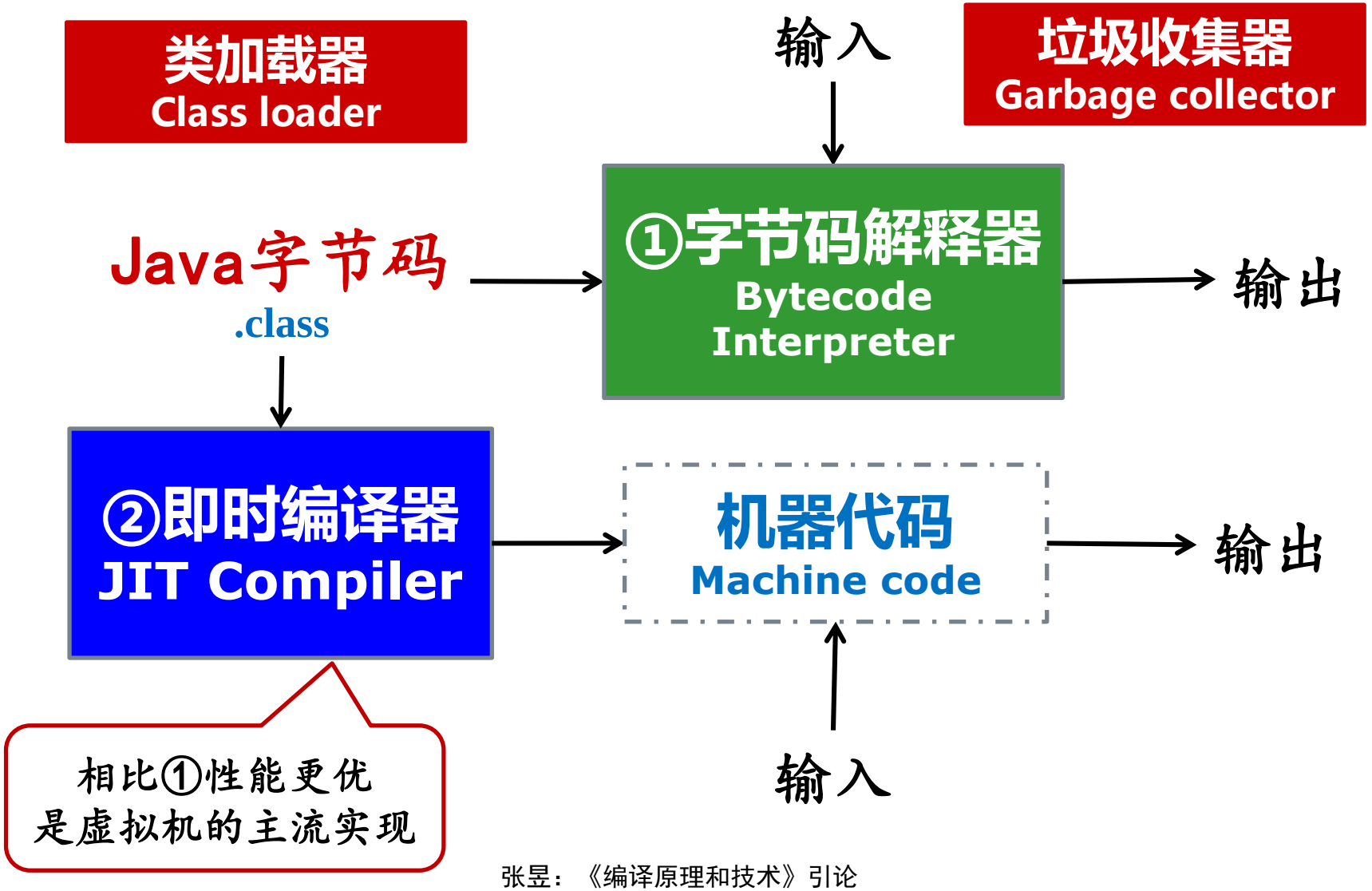
执行效率低，但容易编写

编译器的其他形式

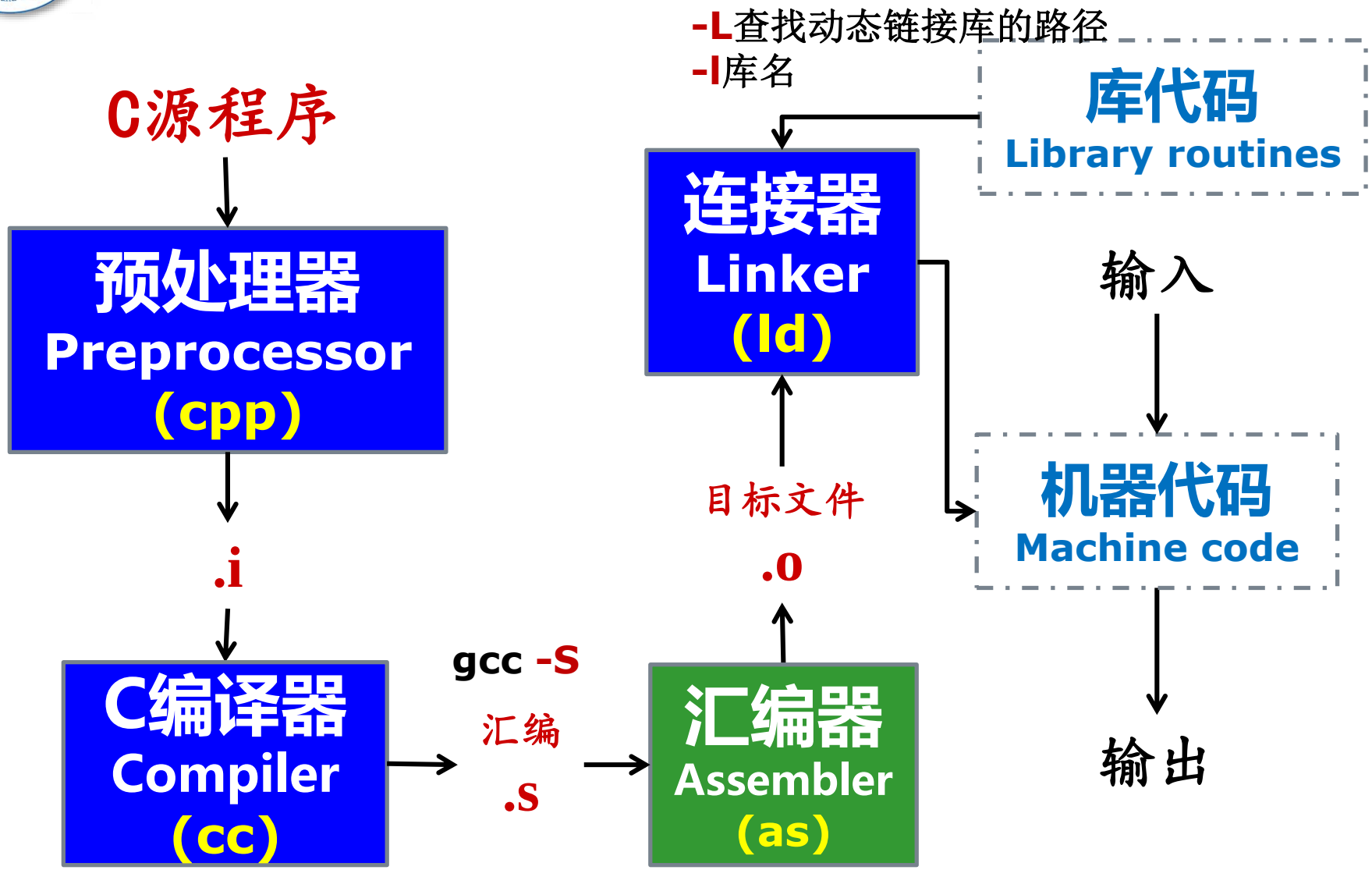
- 交叉编译器（Cross compiler）
  - 在一个平台上生成另一个平台上的代码

    PC → arm-linux-gcc → ARM
- 增量编译器（Incremental compiler）
  - 以增量地编译源程序,只编译修改的部分, 如 Freeline
- 即时编译器（Just-in-time compiler）
  - 在运行时对IR中每个被调用的方法进行编译，得到目标机器的本地代码，如 Java VM 中的即时编译器
- 预先编译器（Ahead-of-time compiler ）
  - 在程序执行之前将IR翻译成本地码，如 ART中的AOT

Java虚拟机


类加载器
Class loader
输入
垃圾收集器
Garbage collector
Java字节码
.class
①字节码解释器
Bytecode
Interpreter
输出
②即时编译器
JIT Compiler
机器代码
Machine code
输出
输入
相比①性能更优
是虚拟机的主流实现

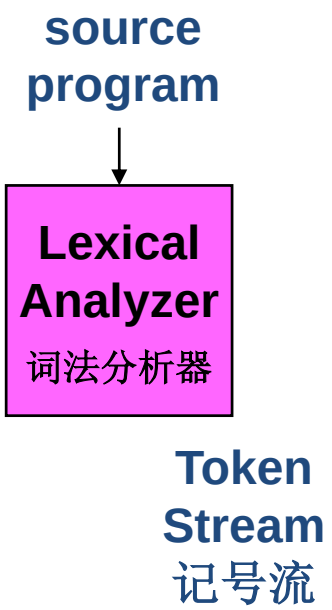
C 编译器


-L查找动态链接库的路径
-l库名
C源程序
预处理器
Preprocessor
(cpp)
.i
C编译器
Compiler
(cc)
gcc -S
汇编
.s
汇编器
Assembler
(as)
目标文件
.o
连接器
Linker
(ld)
库代码
Library routines
输入
机器代码
Machine code
输出

主要内容

1. 编程语言及设计
2. 编译器及形式
3. **编译器的阶段**
4. 编译技术的应用与挑战

编译器的阶段



- 词法分析：将程序字符流分解为记号（Token）序列
  - 形式：<token_name, attribute_value>

Symbol Table 符号表

Error Handler 错误处理

编译器的阶段

position = initial + rate * 60 ← 字符流

source program

Lexical Analyzer
词法分析器

词法分析器

Token Stream
记号流

⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩ ← 记号流

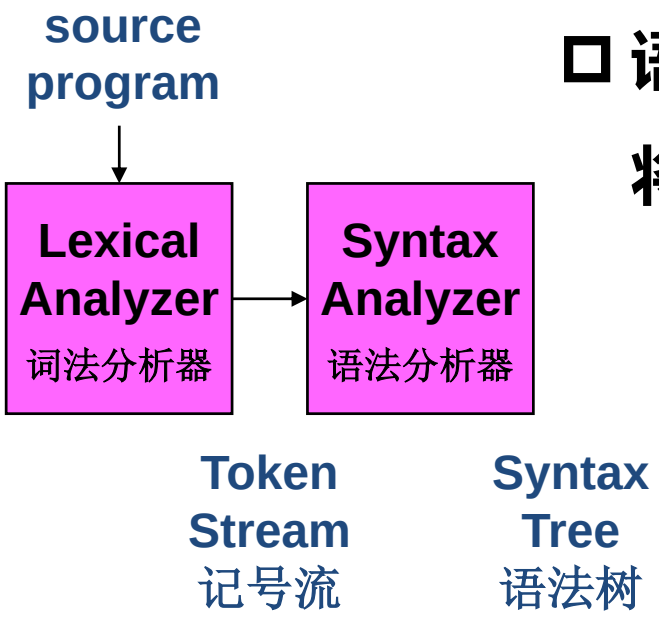
符 号 表

1	position	· · ·
2	initial	· · ·
3	rate	· · ·

Symbol Table 符号表

Error Handler 错误处理

编译器的阶段



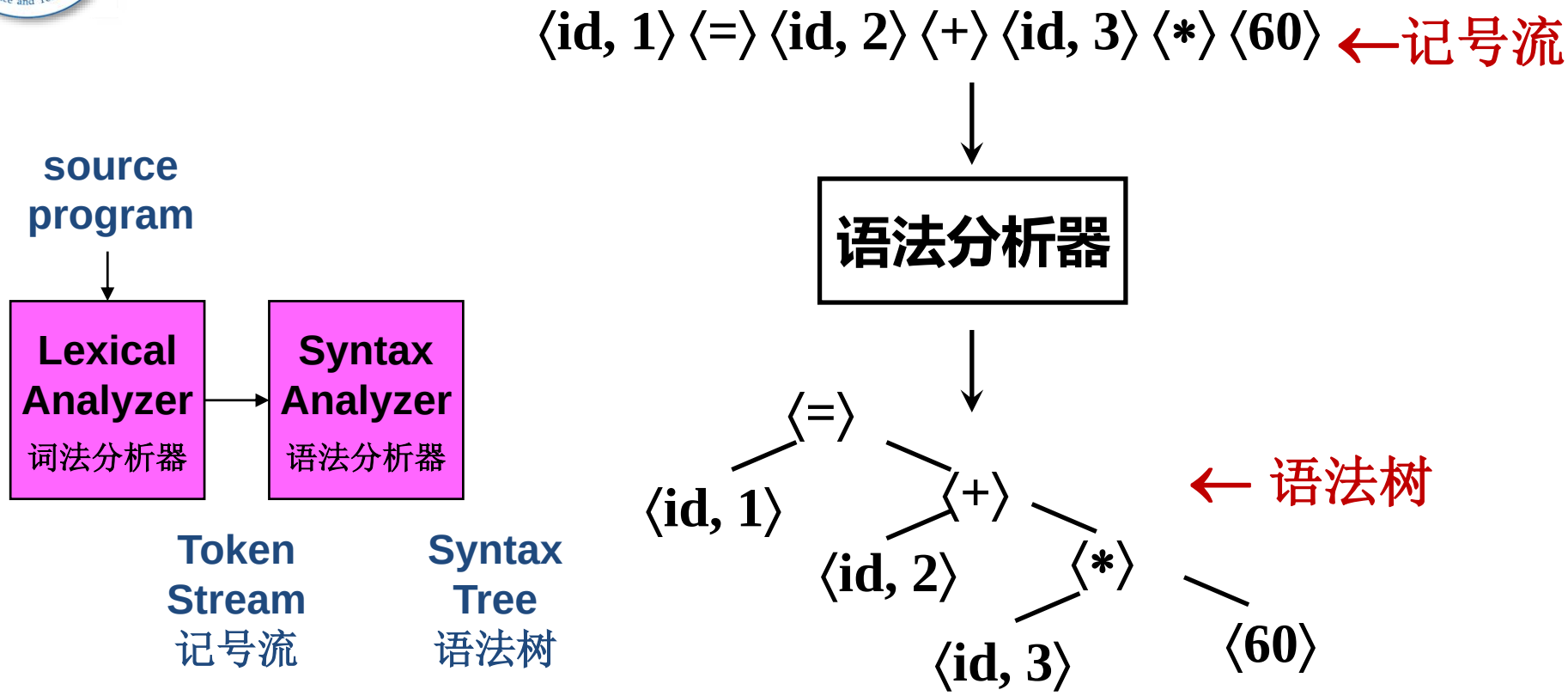
- 语法分析：也称解析(Parsing)

  将记号序列解析为语法结构

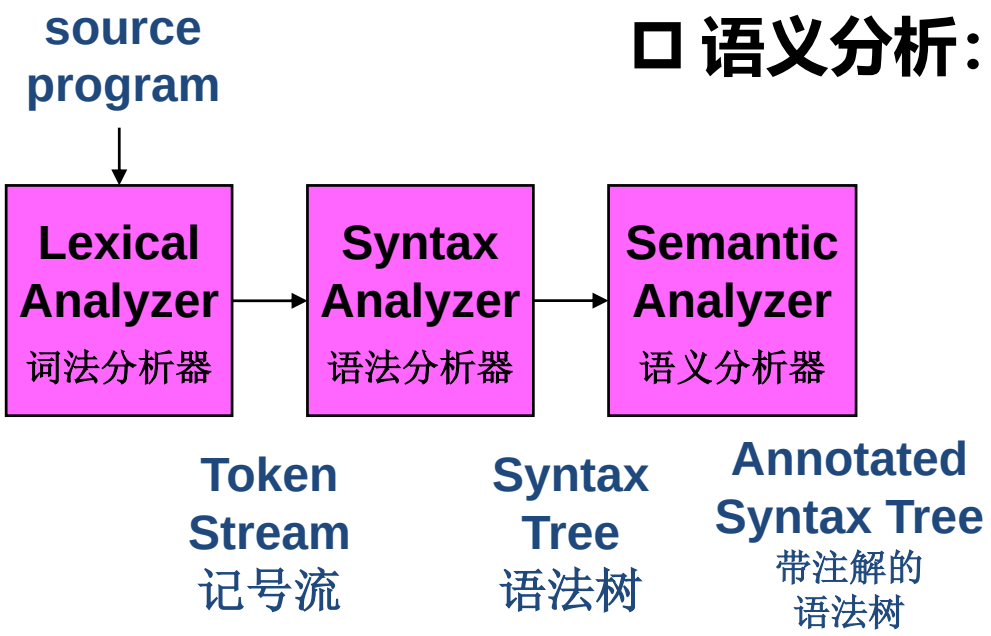
Symbol Table 符号表

Error Handler 错误处理

编译器的阶段


⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
←记号流
source program
Lexical Analyzer
词法分析器
Syntax Analyzer
语法分析器
语法分析器
Token Stream
记号流
Syntax Tree
语法树
⟨=⟩
⟨id, 1⟩
⟨+⟩
⟨id, 2⟩
⟨*⟩
⟨id, 3⟩
⟨60⟩
← 语法树
Symbol Table 符号表
Error Handler 错误处理

编译器的阶段

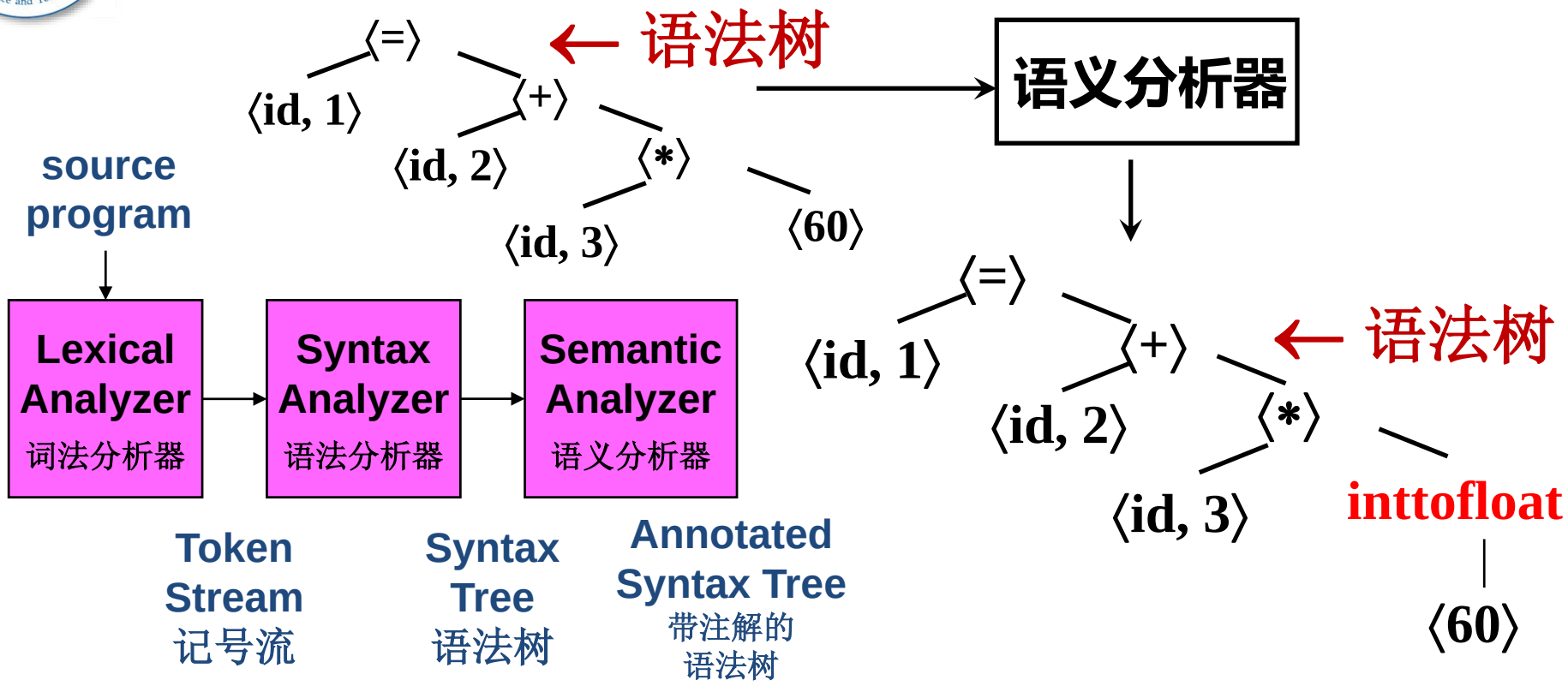


- 语义分析：类型检查、一致性检查等

Symbol Table 符号表

Error Handler 错误处理

编译器的阶段

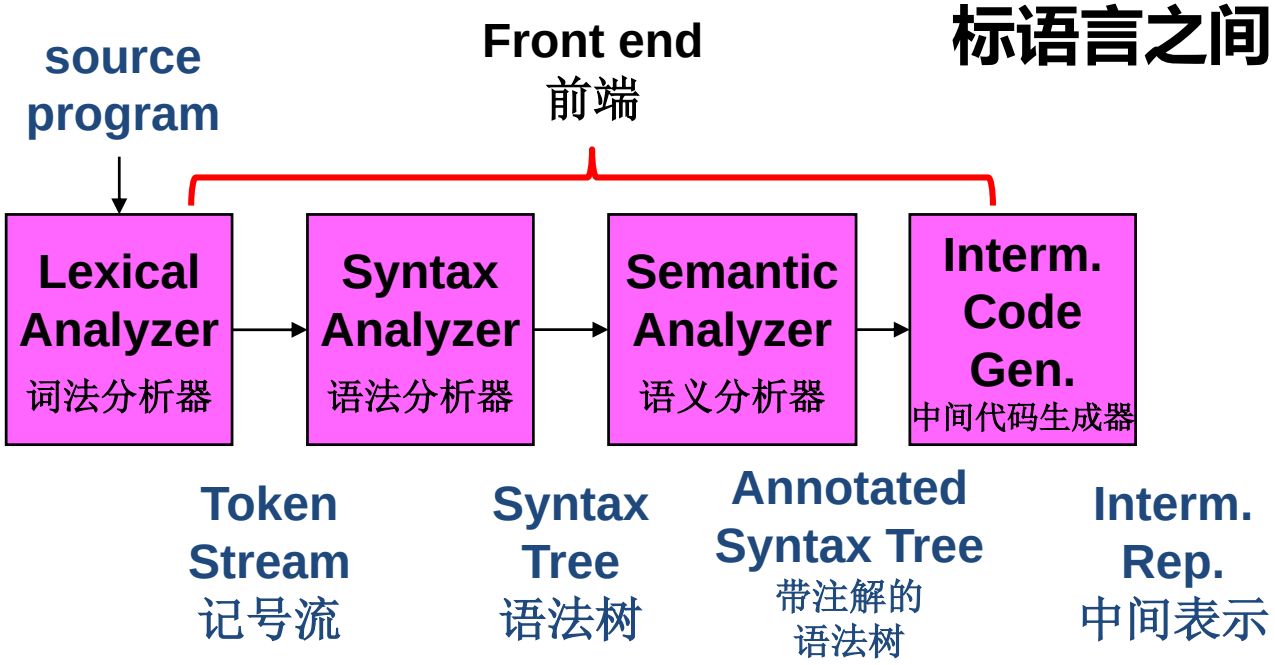


Symbol Table 符号表

Error Handler 错误处理

编译器的阶段

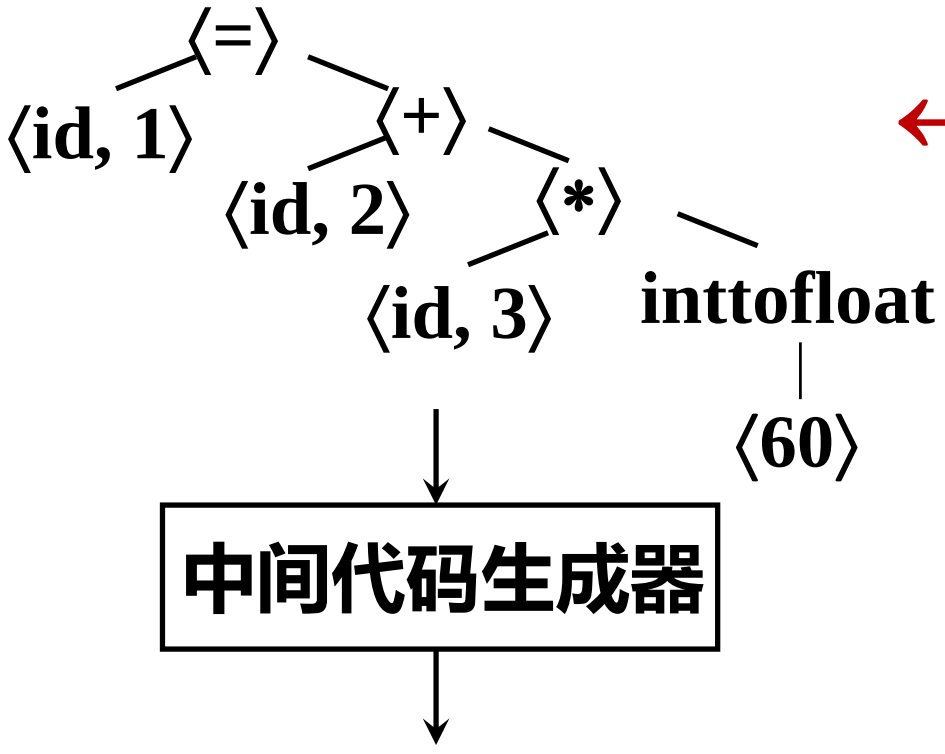
- 中间代码生成：源语言与目标语言之间的桥梁



Symbol Table 符号表

Error Handler 错误处理

编译器的阶段



← 语法树

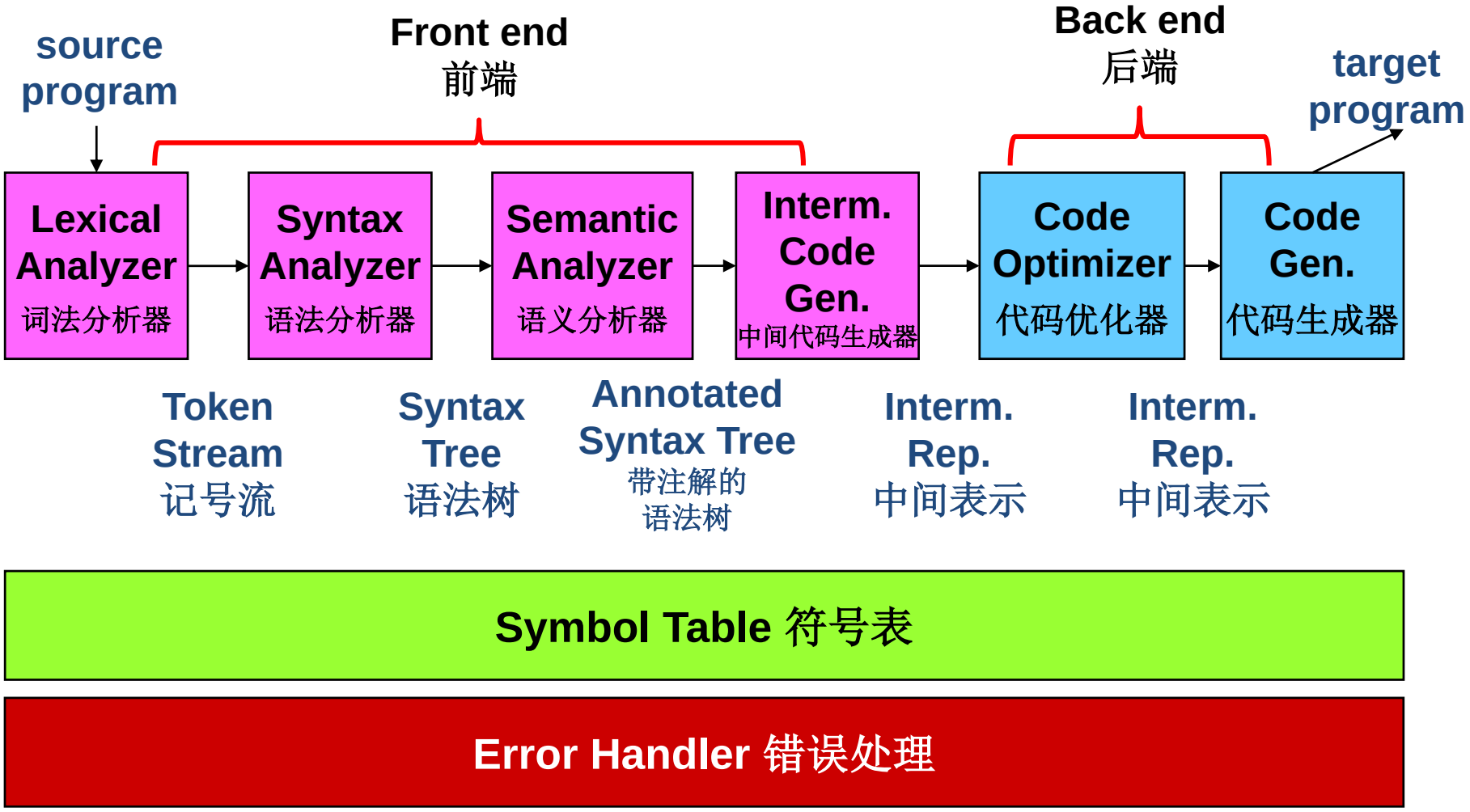
符 号 表

1	position	· · ·
2	initial	· · ·
3	rate	· · ·

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

← 三地址中间代码

编译器的阶段


source program
Front end
前端
Back end
后端
target program
Lexical Analyzer
词法分析器
Syntax Analyzer
语法分析器
Semantic Analyzer
语义分析器
Interm. Code Gen.
中间代码生成器
Code Optimizer
代码优化器
Code Gen.
代码生成器
Token Stream
记号流
Syntax Tree
语法树
Annotated Syntax Tree
带注解的语法树
Interm. Rep.
中间表示
Interm. Rep.
中间表示
Symbol Table 符号表
Error Handler 错误处理

代码优化

- 机器无关的优化、机器相关的优化
- 降低执行时间，减少能耗、资源消耗等

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

← 三地址中间代码

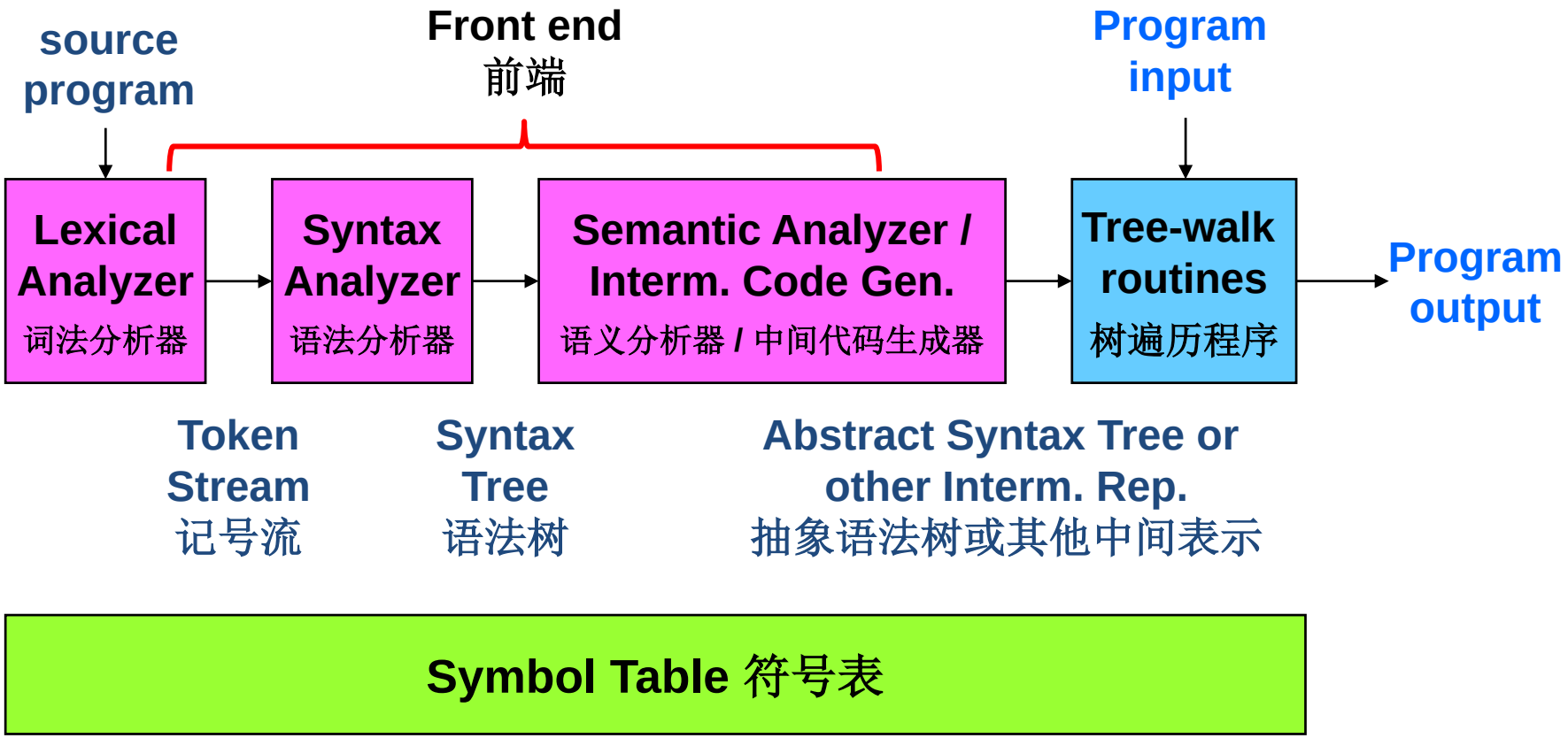

```
t1 = id3 * 60.0
id1 = id2 + t1
```

← 三地址中间代码

符 号 表

1	position	· · ·
2	initial	· · ·
3	rate	· · ·

编译器的阶段


source program
Front end
前端
Program input
Lexical Analyzer
词法分析器
Syntax Analyzer
语法分析器
Semantic Analyzer / Interm. Code Gen.
语义分析器 / 中间代码生成器
Tree-walk routines
树遍历程序
Program output
Token Stream
记号流
Syntax Tree
语法树
Abstract Syntax Tree or other Interm. Rep.
抽象语法树或其他中间表示
Symbol Table 符号表

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. **编译技术的应用与挑战**

人工智能应用及深度学习框架

无人驾驶系统的软件栈

国产系统软件与硬件

MindSpore

HUAWEI

CANN

Compute Architecture for Neural Networks

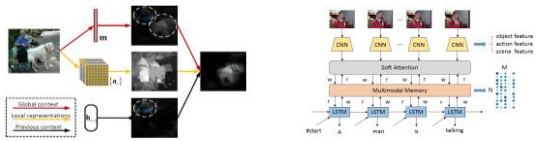
Powered by Ascend

人工智能应用及深度学习框架

无人驾驶系统的软件栈

应用层

模型层



框架层

mxnet PyTorch Keras TensorFlow

高层中间表示

NNVM Relay XLA

编译

算子实现层

arm compute library tvm cuDNN

优化

硬件层

Nvidia Drive PX2

Nvidia Jetson Nano

地平线“征途2.0”

和

国产系统软件与硬件

科学计算应用/人工智能应用

编程框架/库/中间件

SWTVM | SWTensorFlow | SWPyTorch | SWMind | 应用平台基础框架

人工智能算子库 | 基础数学库 | SWBlas | SWSparse | Shentu

编程语言

C | C++ | Fortran | Python | MPI | OpenACC | 并行C

基础组件

众核基础编译器 | Athread运行时 | 动态运行支撑

支持SACA的神威平台

“神威”系列超级计算机 | “神威”众核服务器

多语言软件及优化

- Python + C / C + +
- Java + C / C + +
- JavaScript + C / C + +
- Go + C / C + +

跨语言程序分析

类型推断、跨语言程序调用图、资源分析与管理、信息流分析等

安全可靠、性能极致

关键科学问题

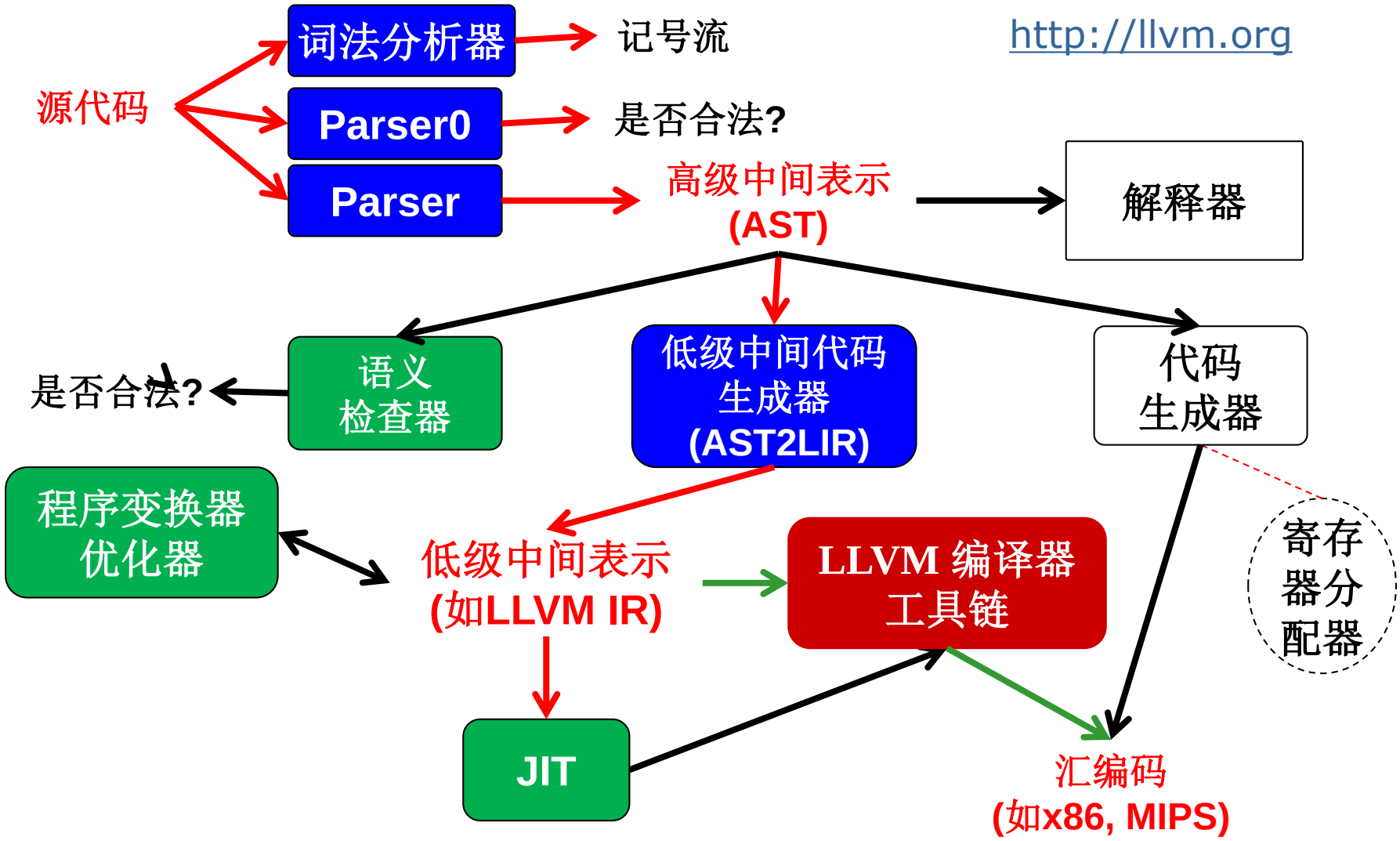
- **语言定义**
  - 如何抽象和形式化
  - 如何推陈出新

正规式
上下文无关文法
类型系统

- **应对不断发展的应用和硬件**
  - 发挥硬件及指令集优势的代码生成
  - 软硬件协同设计
  - 增强软硬件系统的健壮性

中间表示设计与生成
数据流/控制流分析
代码生成与优化

基础实验的考虑


词法分析器
记号流
http://llvm.org
源代码
Parser0
是否合法?
Parser
高级中间表示
(AST)
解释器
语义
检查器
是否合法?
低级中间代码
生成器
(AST2LIR)
代码
生成器
程序变换器
优化器
低级中间表示
(如LLVM IR)
LLVM 编译器
工具链
寄存
器分
配器
JIT
汇编码
(如x86, MIPS)

我听到的会忘掉，

我看到的能记住，

我做过的才真正明白。