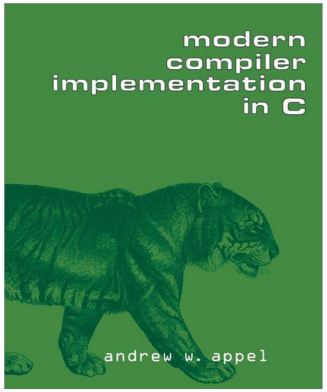
编译原理
1. 概述

rainoftime.github.io
浙江大学
计算机科学与技术学院

课程信息

主讲老师	pyaoaa@zju.edu.cn 研究方向: 程序分析/验证/合成, 数理逻辑, 软件安全
课程助教	陈楷骐, chiaki_cage@zju.edu.cn
时间地点	每周二(3、4、5节)，曹西101
课程网站	学在浙大, https://rainoftime.github.io/teaching/

Textbook:
<Modern Compiler Implementation in C>,
Andrew W. Appel



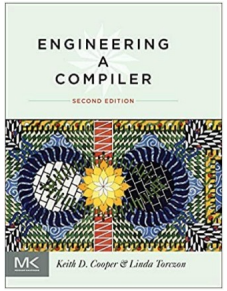
参考书籍

- [虎书]Andrew W. Appel. *Modern Compiler Implementation in Java/ C / ML*

  现代编译原理—C语言描述；人民邮电出版社—Java语言描述

- [龙书]Alfred V. Aho, Monica S. Lam, et al. *Compilers: Principles, Techniques, and Tools(2nd Ed.)*

  编译原理. 影印本-2011、译本-2009, 机械工业出版社



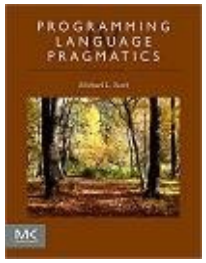
- Keith Cooper & Linda Torczon, *Engineering a Compiler*

  编译器工程，机械工业出版社

参考书籍

- [鲸书] Steven Muchnick. *Advanced Compiler Design and Implementation.*

  高级编译器设计与实现，影印本-2003, 译本-2005



- Michael L. Scott. *Programming Language Pragmatics*,

  程序设计语言：实践之路 (第1-3版)，电子工业出版社

- [LCC] Christopher W. Fraser, David R. Hanson. *A Retargetable C Compiler.*

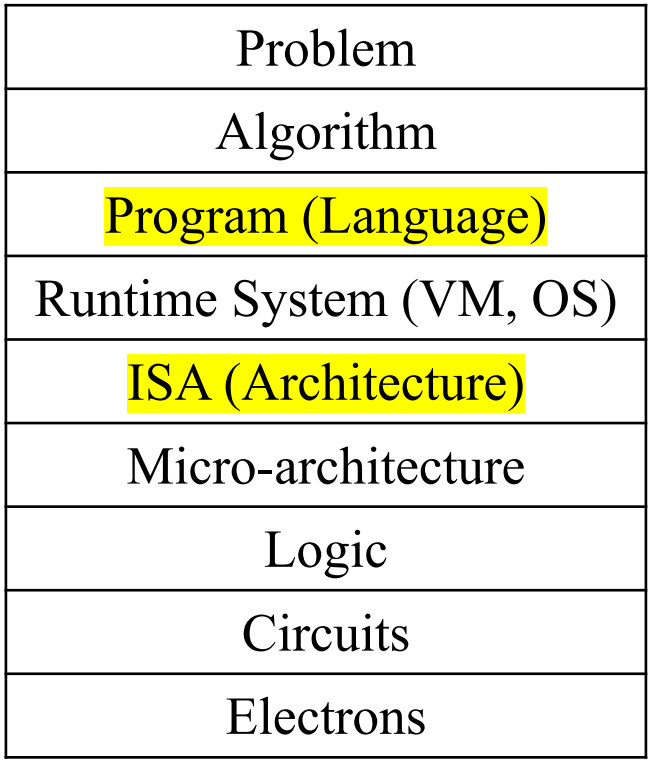
  可变目标C编译器—设计与实现.

相关课程资源

- **Stanford课程主页**
  http://web.stanford.edu/class/cs143/
- **MIT课程主页**
  https://github.com/6035/sp21
- **UC Berkeley课程主页**
  https://inst.eecs.berkeley.edu/~cs164/fa21/

编译器是学科交叉的产物

- **理论计算机科学**
  - 有限自动机, 形式文法
- **编程语言**
  - 类型系统，不动点迭代
- **算法与数据结构**
  - 树/图的遍历，动态规划
- **系统和体系结构**
  - 内存管理, 指令集, 并行



计算机系统层次

课程内容

1. **Introduction**
2. Lexical Analysis
3. Parsing
4. Abstract Syntax
5. Semantic Analysis
6. Activation Record
7. Translating into Intermediate Code
8. Basic Blocks and Traces
9. Instruction Selection
10. Liveness Analysis
11. Register Allocation
13. Garbage Collection
14. Object-oriented Languages
18. Loop Optimizations

考核方式

- 课程作业(课后小型练习题) = 10%
- 随堂测验 = 10%
- 期中考试 = 15%
- 综合性课程设计 = 25%
- 期末考试 = 40%

Note: Final Exam < 40/100 ➔ Final Grade < 60/100

本讲内容

1. 编程语言及设计
2. 编译器及其形式
3. 编译器的阶段
4. 案例: Tiger编译器

1. 编程语言及设计

编程语言相关的ACM图灵奖

- 图灵奖自1966年颁发以来，共有75名获奖者，其中编程语言、编译相关的科研人员有21位，占比28%，Alan J. Perlis因编译技术贡献成为第一位获得图灵奖的科学家

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

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

部分相关图灵奖获得者

年份	科学家	贡献	年份	科学家	贡献
1966	Alan J. Perlis	高级程序设计技巧，编译器构造	1972	Edsger Dijkstra	程序设计语言的科学与艺术
1974	Donald E. Knuth	算法分析、程序设计语言的设计、程序设计	1976	MichaelO.Rabin, DanaS.Scott	非确定性自动机
1977	John Backus	高级编程系统，程序设计语言规范的形式化定义	1979	Kenneth E.Iverson	程设语言和数学符号，互动系统的设计，程设语言的理论与实践
1980	C.Antony R.Hoare	程序设计语言的定义与设计	1983	KenThompson, DennisM.Ritchie	UNIX 操作系统和C 语言
1984	Niklaus Wirth	程序设计语言设计、程序设计	2001	Ole-JohanDahl KristenNygaard	面向对象编程
1987	John Cocke	编译理论，大型系统的体系结构，及RISC计算机的开发	2005	Peter Naur	Algol60语言
2003	Alan Kay	面向对象编程	2008	Barbara Liskov	编程语言和系统设计的实践与理论
2006	Frances E. Allen	优化编译器	2020	Jeffrey David Ullman, Alfred Vaino Aho	推进编程语言实现的基础算法和理论、教材撰写
2021	Jack J. Dongarra	线性代数运算的高效数值算法、并行计算编程机制和性能评估工具			

什么是编程语言

- **A programming language is a notation for describing computations to people and to machines.**

- **不同的编程语言范型 (Paradigms)**
  - **过程式**(Procedural): C, Fortran, Pascal,…
  - **函数式**(Functional): Lisp/Scheme, Haskell, …
  - **逻辑式**(Logic): Prolog, Datalog, …
  - **面向对象**(Object-Oriented): Smalltalk, Java, Eiffel, …

编程语言 = 语法 + 语义

- **语法**: What sequences of characters are valid programs?
  - 通常由上下文无法文法定义

- **语义**: What is the behavior of a valid programs?
  - **操作语义**: How can we execute a program?
  - **公理语义**: What can we prove about a program
  - **指称语义**: What math function does the program compute?

例: 不同语言实现最大公约数

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

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

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

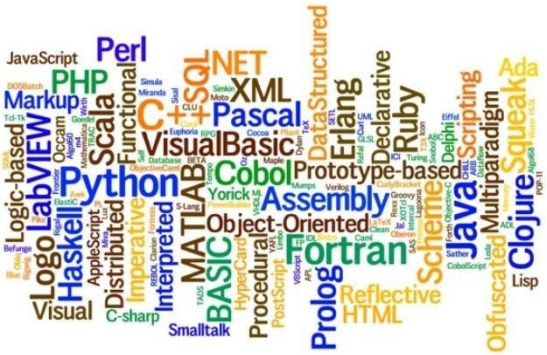
编程语言不断演化和发展

- **编程语言自身在不断发展**
  - C: C90, C99, C11
  - C++: 1998, .., 2011, 2014, 2017…
- **新的编程语言不断诞生**
  - **通用编程语言:** Go (2009), Rust (2010), Swift (2014),…
  - **领域特定语言**
    - 日常工具

      Latex, SVG, DOT, Markdown,…
    - 将高阶函数用于大数据处理

      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,…



2. 编译器及其形式

编程器是什么? 有什么作用?

编译器是什么

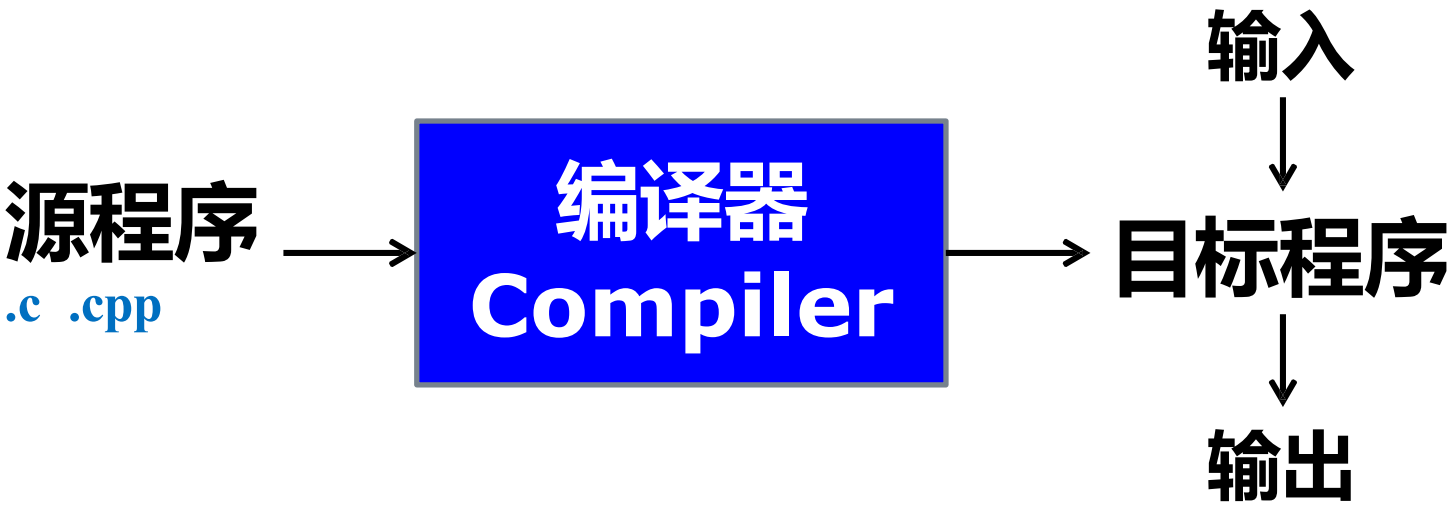
- **语法**: What sequences of characters are valid programs?
  - 通常由上下文无法文法定义

- **语义**: What is the behavior of a valid programs?
  - **操作语义**: How can we execute a program?
  - **公理语义**: What can we prove about a program
  - **指称语义**: What math function does the program compute?

- **编译器**: Translate from the syntax of one language to another but preserve the semantics.

编译器是什么

- **编译器是一个程序，读入源程序并将其翻译成语义等价的目标程序**
  - 目标程序: 如果是可执行的机器语言程序，则可以被用户调用，处理输入并产生输出
  - 目标程序: 如果是汇编语言程序，则须经汇编器汇编后方可执行



编译器是什么

- **狭义看法**
  - 源程序：用某种高级语言编写
  - 目标程序：用目标代码或机器语言编写

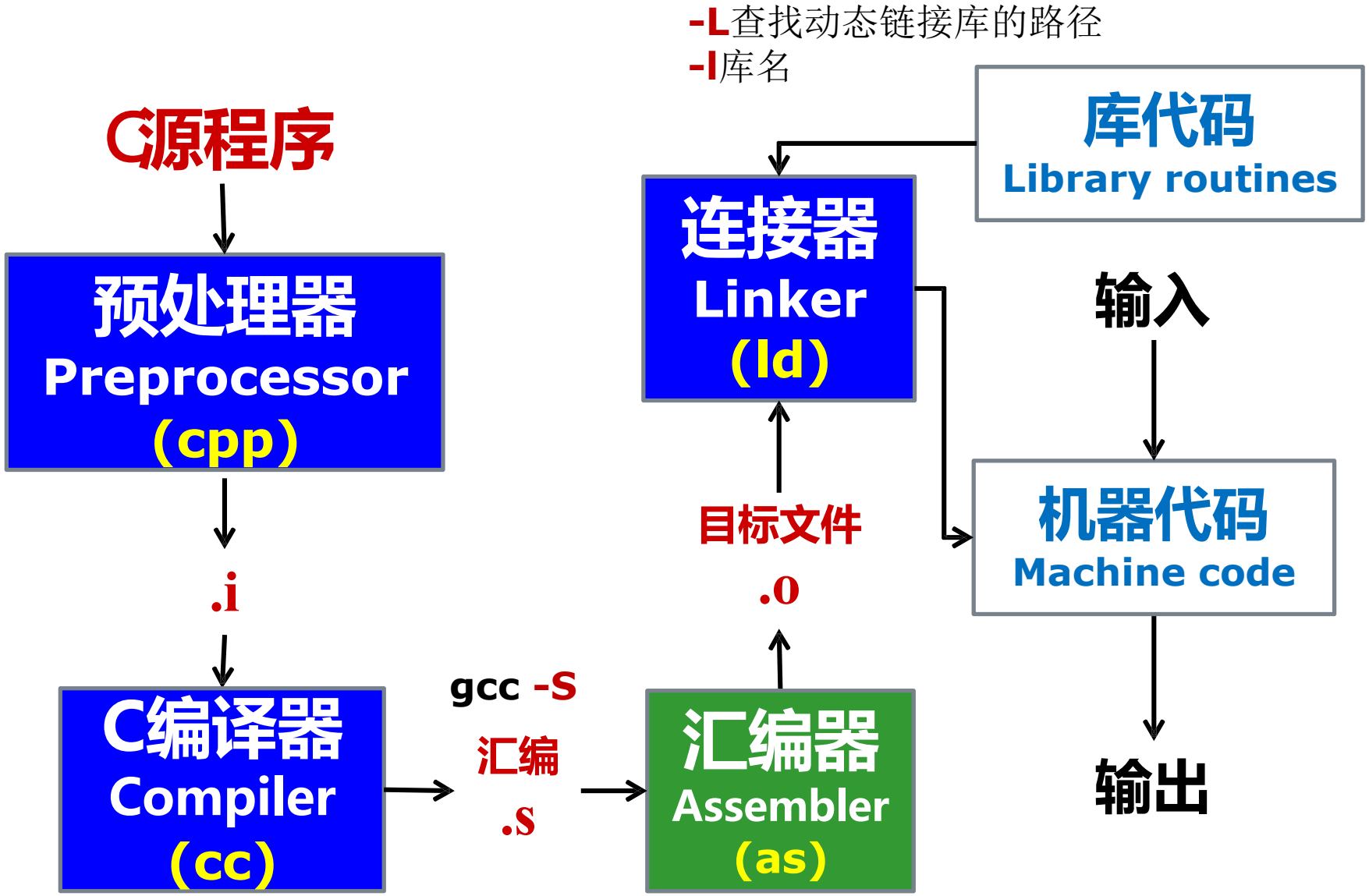
$$C++ \Rightarrow 机器语言$$

- **广义看法**
  - 目标程序：介于源语言和机器语言之间的“中间语言”，甚至可以是另外一种高级语言

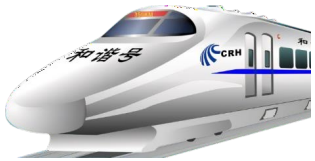
$$C++ \Rightarrow C$$

$$Pascal \Rightarrow C$$

例: C语言编译器


-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

例: clang编译器的优化等级



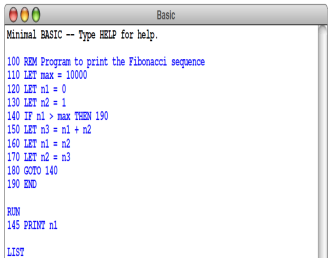
优化等级	简要说明
-Ofast	在-O3级别的基础上，开启更多激进优化项，该优化等级不会严格遵循语言标准
-O3	在-O2级别的基础上，开启了更多的高级优化项，以编译时间、代码大小、内存为代价获取更高的性能。
-Os	在-O2级别的基础上，开启降低生成代码体量的优化
-O2	开启了大多数中级优化，会改善编译时间开销和最终生成代码性能
-O/-O1	优化效果介于-O0和-O2之间
-O0	默认优化等级，即不开启编译优化，只尝试减少编译时间

延伸阅读：https://clang.llvm.org/docs/CommandGuide/clang.html#code-generation-options

解释器

- **解释器**: 在一种语言的机器上，**直接执行**用另一种语言写的程序的过程，称为解释。
  - 接受用户提供的输入，进行解释并逐句执行

- **关于解释器的另一种理解方式:**
  - 先把源程序“编译”到字节码（如Java字节码）
  - 再通过特定的虚拟机”执行“字节码



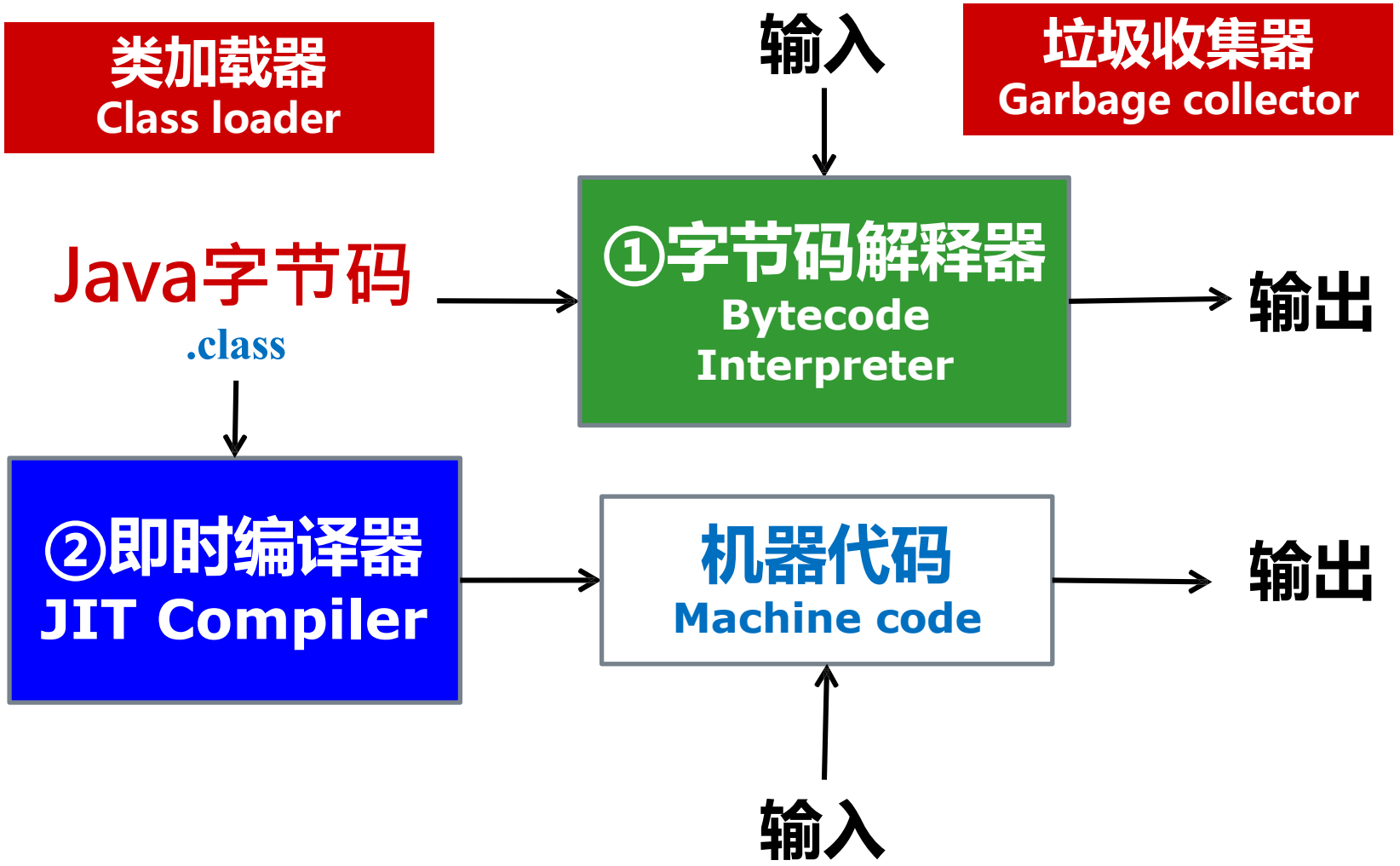
Basic解释器

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)**
  - 增量地编译源程序(只编译修改的部分)
- **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)**
  - 在程序执行前将IR翻译成本地码，如ART中的AOT

例: 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

编译器的作用

- **计算场景**
  - 传统场景（通用计算、科学计算等）
  - 新兴场景（AI、隐私计算等）
- **硬件架构**
  - 单核（标量、超标量、SIMD…）
  - 多核/众核、GPU、FPGAs…
- **基础理论**
  - 形式化语言与自动机 、格…
  - 图论、整数规划…
  - …

- **开发效率 Productivity**
  - 屏蔽硬件架构信息
  - 支持高层编程抽象

编得快

- **运行性能 Performance**
  - 硬件无关编译优化
  - 硬件相关编译优化

跑得快

- **安全可靠 Safety & Security**
  - 类型安全
  - 功能正确
  - 信息流安全

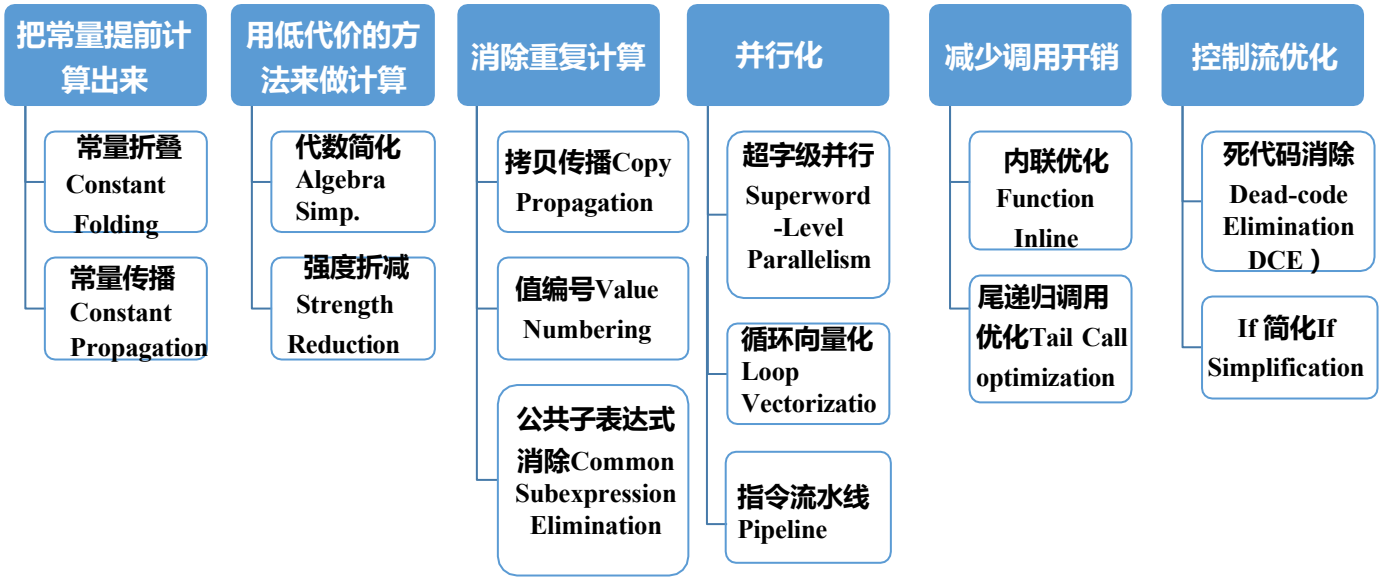
信得过

“编写编译器的原理和技术具有普遍的意义，以至于在每个计算机科学家的研究生涯中，该书中的原理和技术都会**反复用到**。”

——著名计算机专家 Alfred V.Aho

例: 编译优化

- 常见的编译优化思路



- 优化的分类(可能在不同阶段进行)
  - 局部，全局，过程间
  - 机器无关，机器相关
  - ...?

例: LLVM用于代码安全

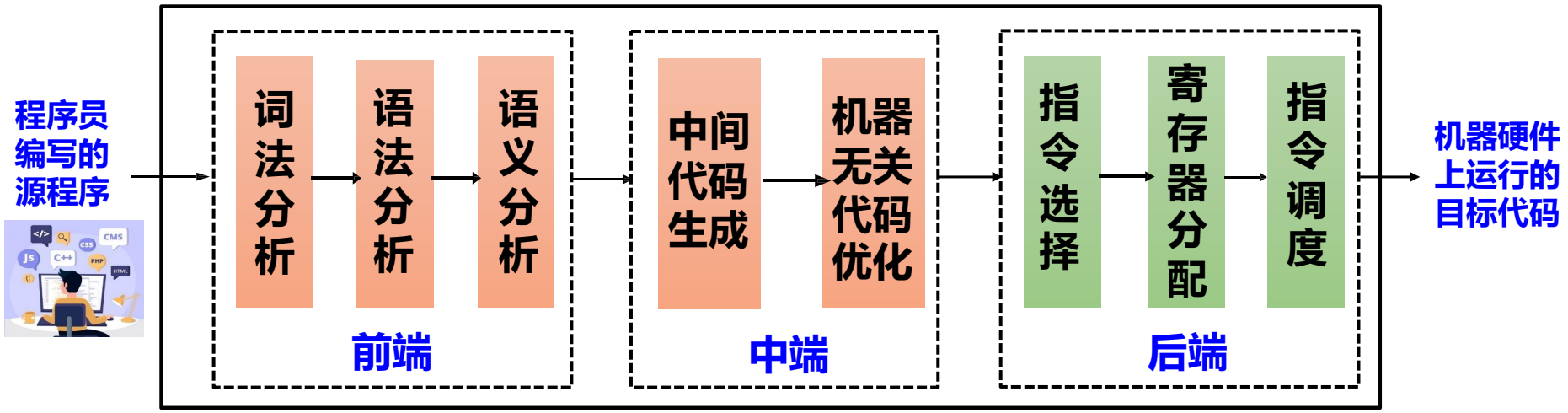
- LLVM编译器框架 https://llvm.org/
  - LLVM提供除“编译“(clang)之外的工具链

工具	作用
Clang static analyzer	静态代码差错
llcov	动态监控覆盖率
AddressSanitizer (ASan)	动态监控安全问题
DataflowSanitizer(DFSan)	动态污点分析
libFuzzer	模糊测试
LLDB	调试器
…	…

3. 编译器的阶段

编程器是如何构造的?

编译过程概览



Symbol Table 符号表

- 记录程序中变量、函数等对象的各种属性
- 符号表可由编译器的各个步骤使用

Error Handler 错误处理

- 语法错误: 如算术表达式的括号不配对
- 语义错误: 如算符作用于不相容的运算对象

词法分析(Lexing/Scanning/Lexical Analysis)

- **词法分析: 将程序字符流分解为记号 (Token)序列：**

position = initial + rate * 60 ← 字符流

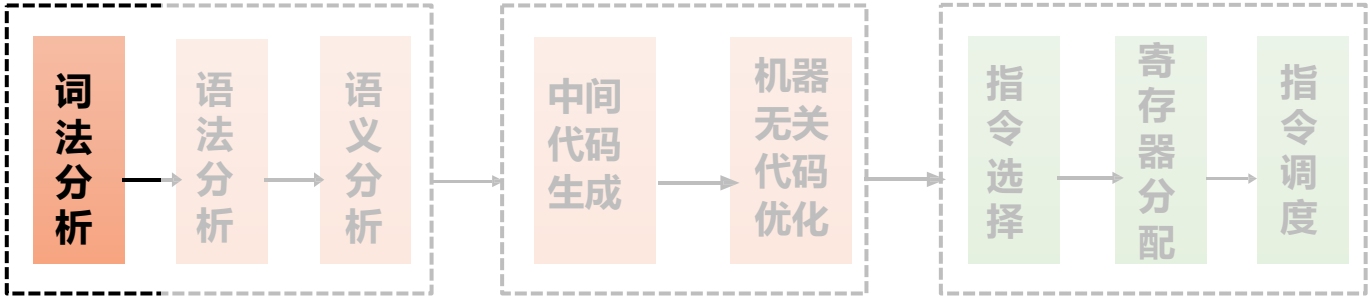
↓

词法分析器

↓

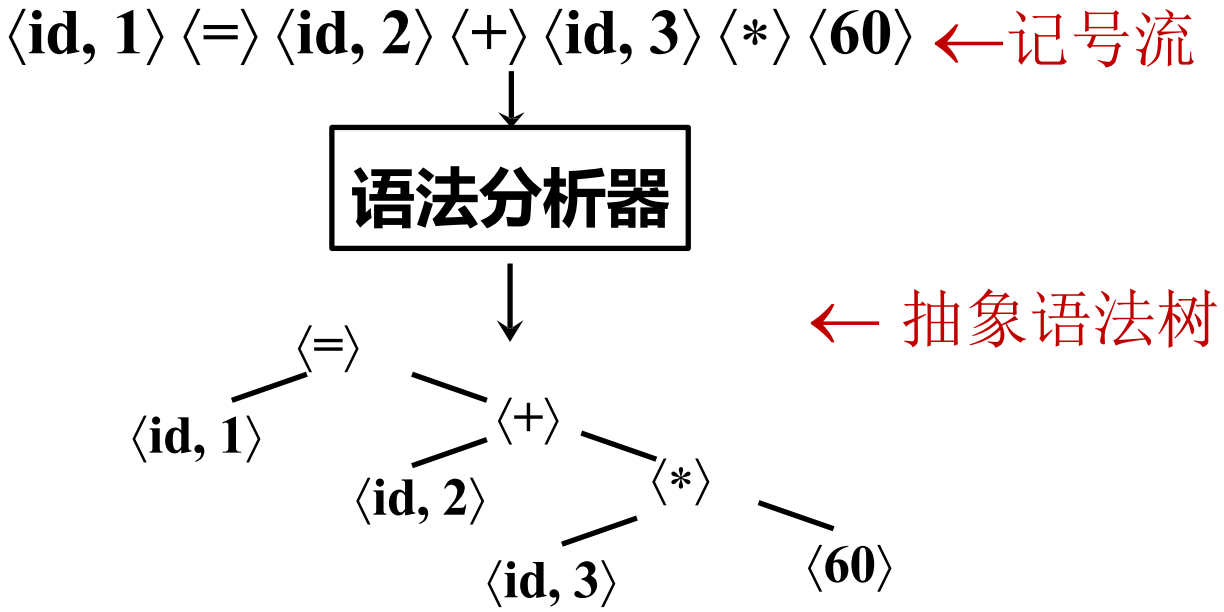
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩ ← 记号流

命令行输入：clang -cc1 -dump-tokens xx.c

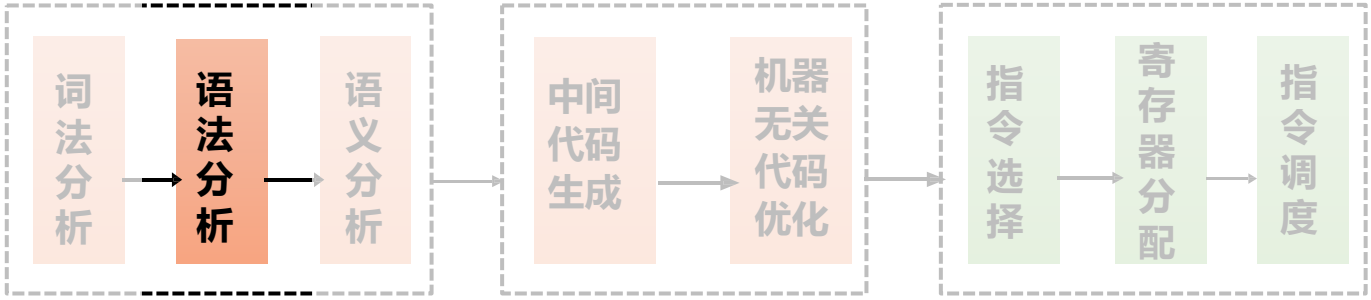


语法分析(Parsing/Syntax Analysis)

- **语法分析: 将记号序列解析为语法结构**

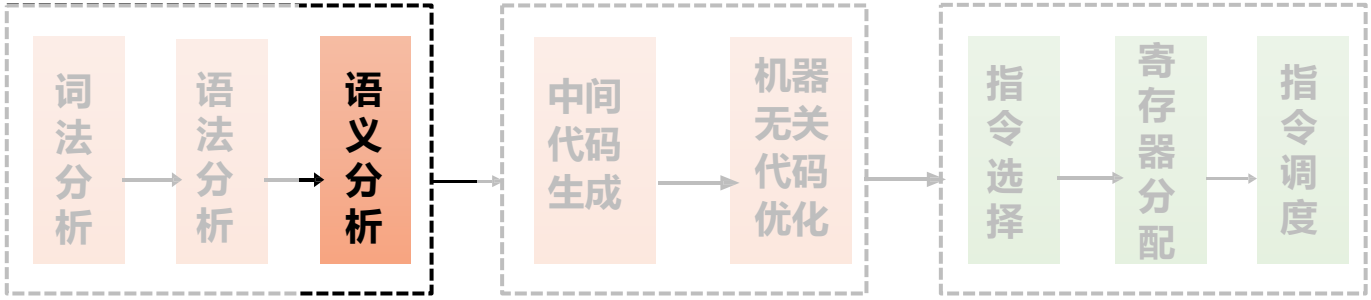


命令行输入：clang -cc1 -dump-tokens xx.c



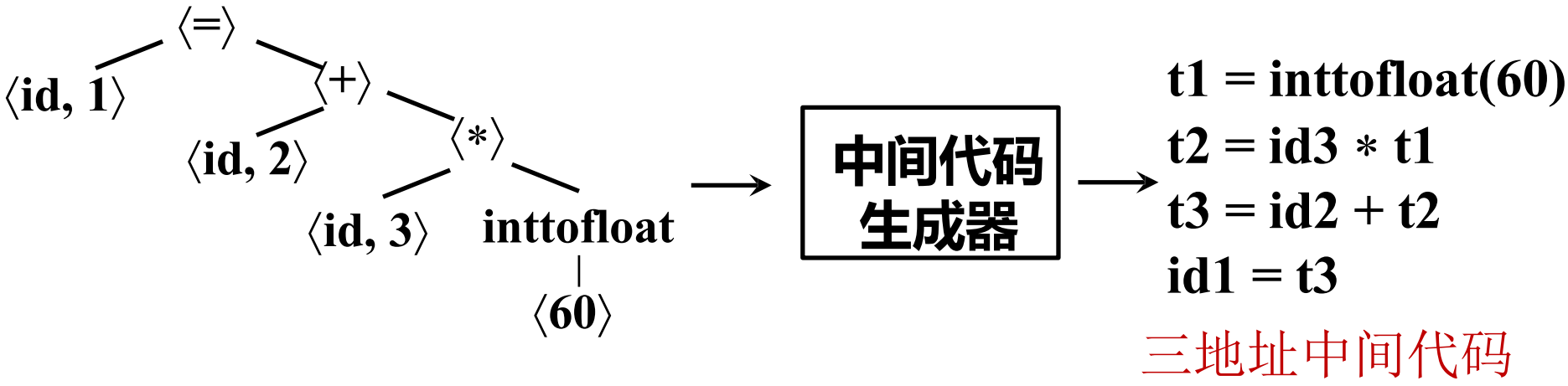
语义分析(Semantic Analysis)

- 语义分析: 收集标识符的类型等属性信息



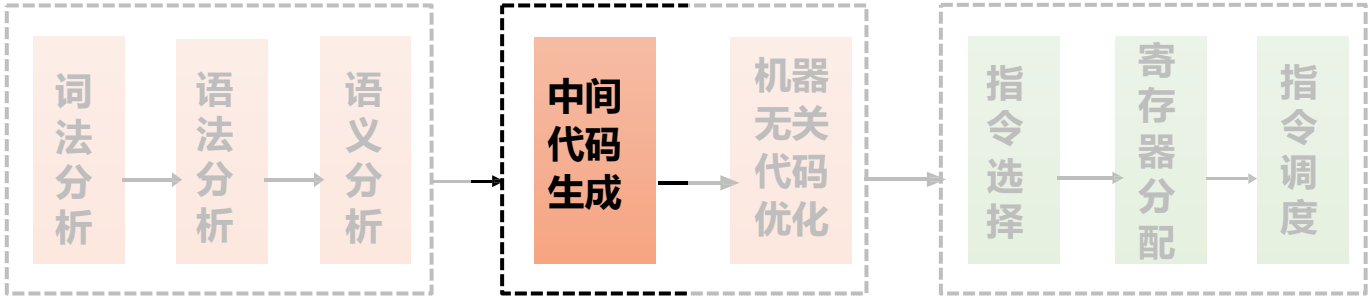
中间代码生成

- **中间代码/表示: 源语言与目标语言之间的桥梁**



三地址中间代码

命令行输入：clang -cc1 xx.c -emit-llvm -o xx.llç



(基于中间表示的)优化

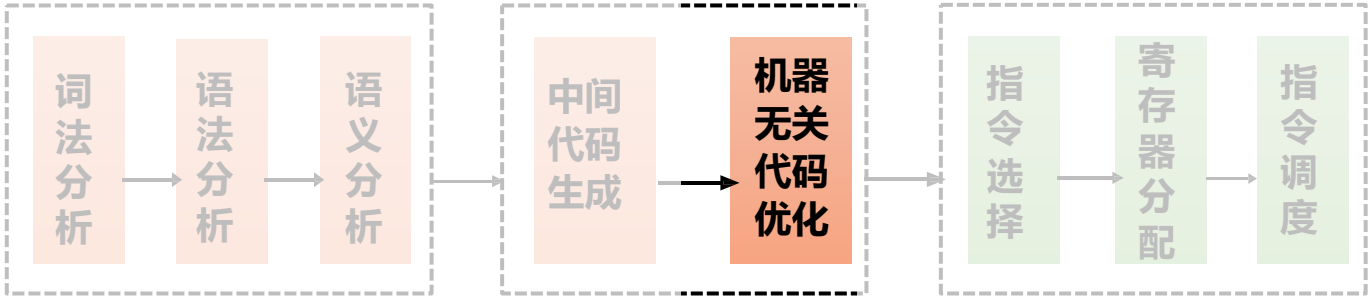
- **基于中间表示进行分析与变换**

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

→ 代码优化器 →

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

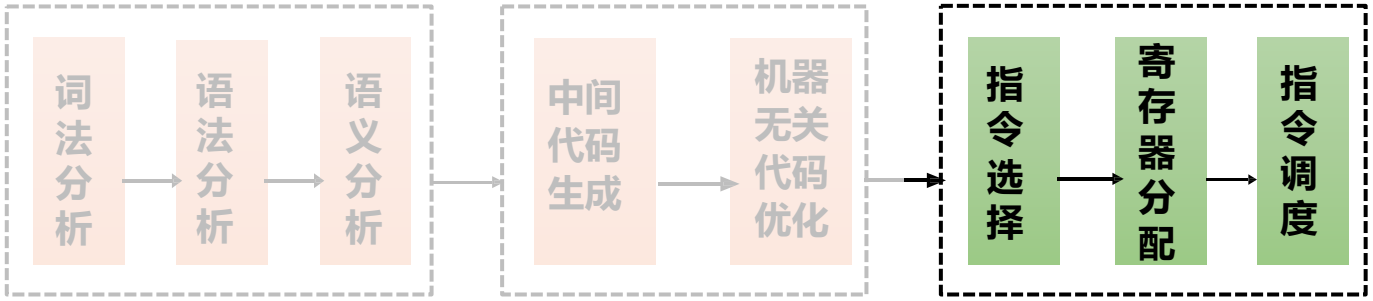
命令行输入：clang -S -emit-llvm -O3 test.c -o test-new.ll



目标代码生成

- 把中间表示形式翻译到目标语言
  - 指令选择、寄存器分配、指令调度

命令行输入：llc-14 xxx.ll -o xxx.s



例: Clang编译器的编译过程

- 从C语言源码到汇编语言

[root@host ~]# clang test.c -o test

```c
#include <stdio.h>

int factorial(int n) {
  int acc = 1;
  while (n > 0) {
    acc = acc * n;
    n = n - 1;
  }
  return acc;
}

int main(int argc, char *argv[]) {
  printf("factorial(6) = %d\n", factorial(6));
}
```

```
factorial:
        movl    $1, %rax
        cmpq    $2, %rdi
        jl      .LBB0_2
.LBB0_1:
        imulq   %rdi, %rax
        decq    %rdi
        cmpq    $1, %rdi
        jg      .LBB0_1
.LBB0_2:
        retq

main:
        movl    $.str, %rdi
        movl    $720, %rsi
        callq   printf
        retq

.globl  .str
.str:
        .asciz  "Factorial is %ld\n"
```

例: Clang编译器的编译过程

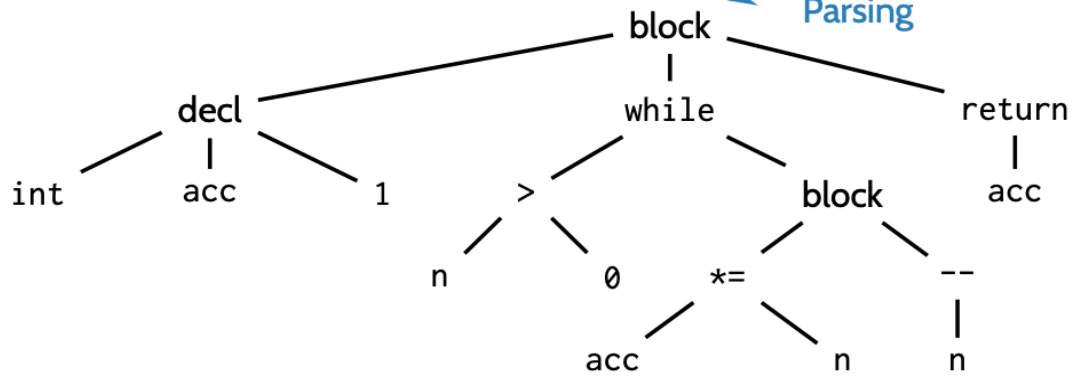
- 词法分析 (Lexing) & 语法分析 (Parsing)

Lexing

```
1   int acc = 1;
2   while (n > 0) {
3     acc *= n;
4     n --;
5   }
6   return acc;
```

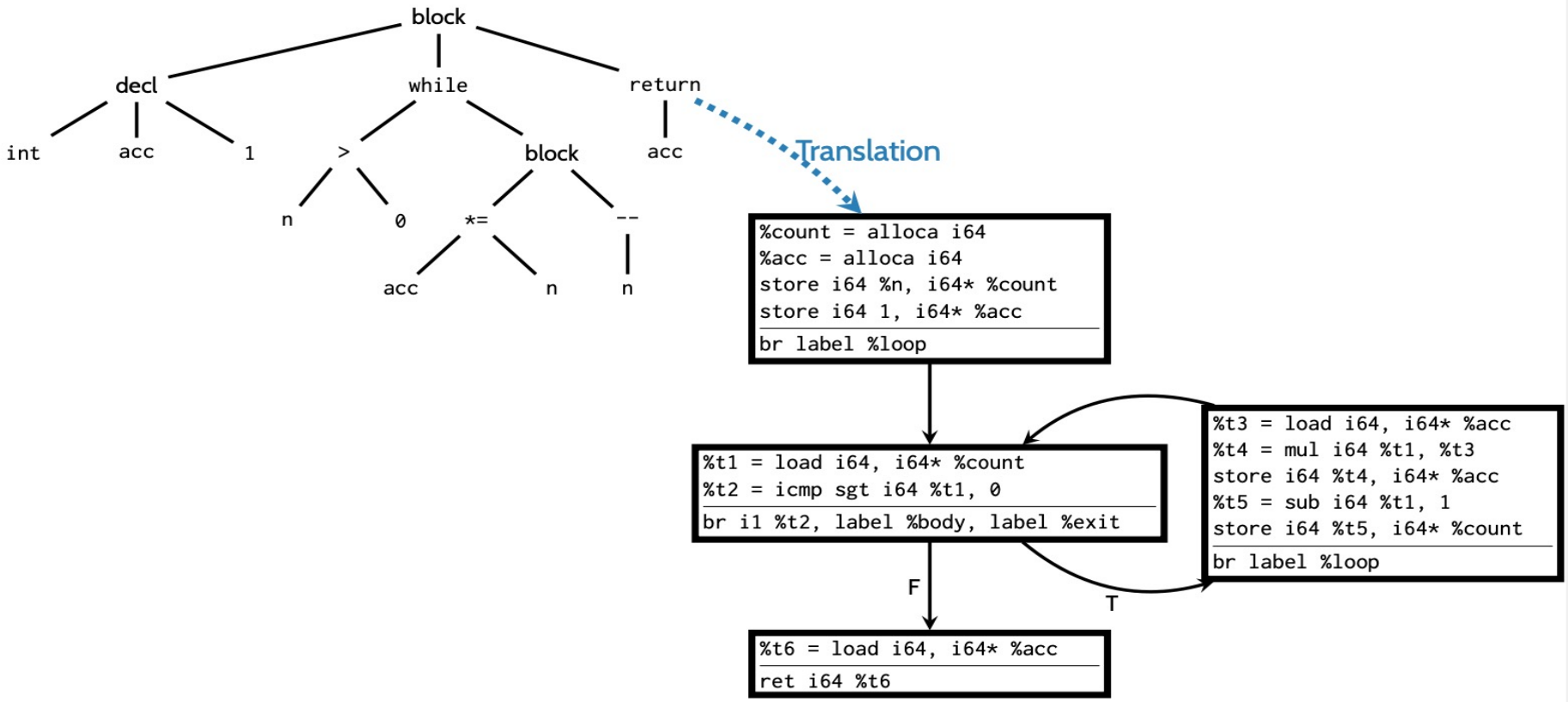
```
1   INT, IDENT "acc", EQUAL, INT 1, SEMI,
2   WHILE, LPAREN, IDENT "n", GT, INT 0, RPAREN, LBRACE,
3   IDENT "acc", TIMESEQUAL, IDENT "n", SEMI,
4   IDENT "n", DECREMENT, SEMI,
5   RBRACE
6   RETURN, IDENT "acc", SEMI
```

Parsing



例: Clang编译器的编译过程

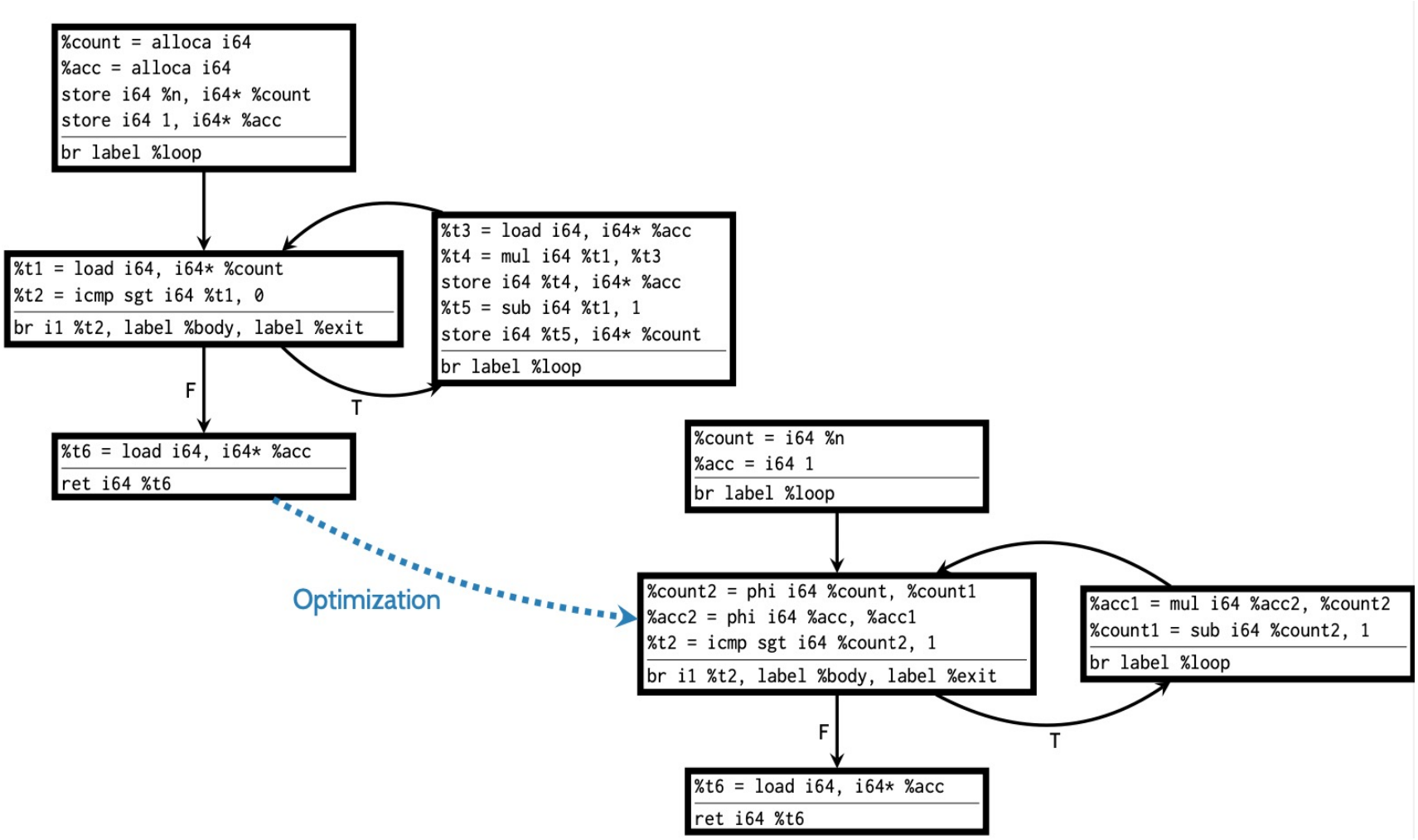
- 语义分析 & 中间表示生成



此处假设为生成LLVM IR

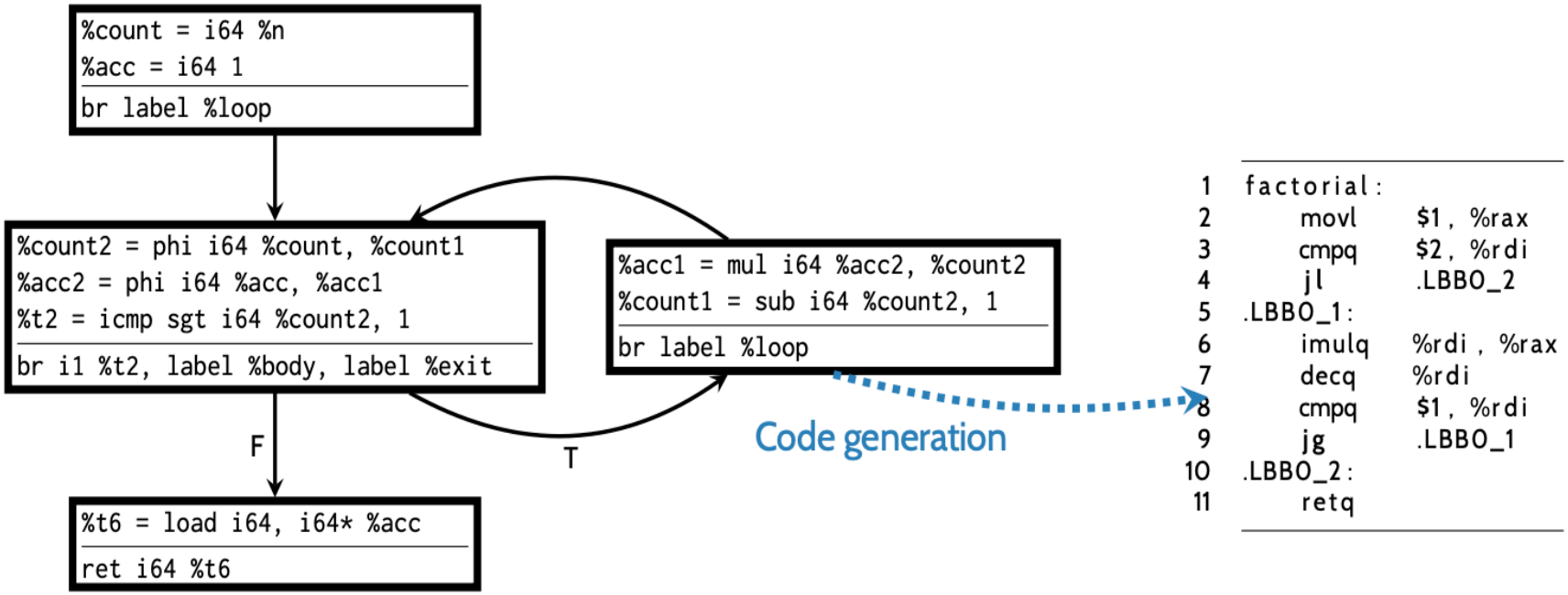
例: Clang编译器的编译过程

- (基于中间表示的)编译优化



例: Clang编译器的编译过程

- 目标代码生成(和优化)

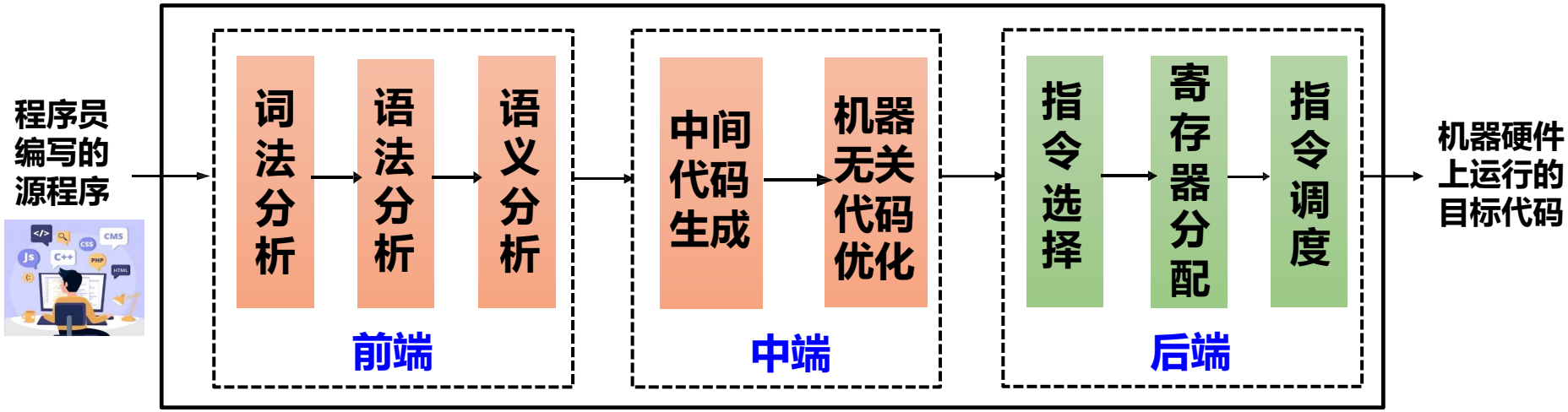


4. 案例: Tiger编译器

虎书有什么不一样?

回顾: 编译器流程

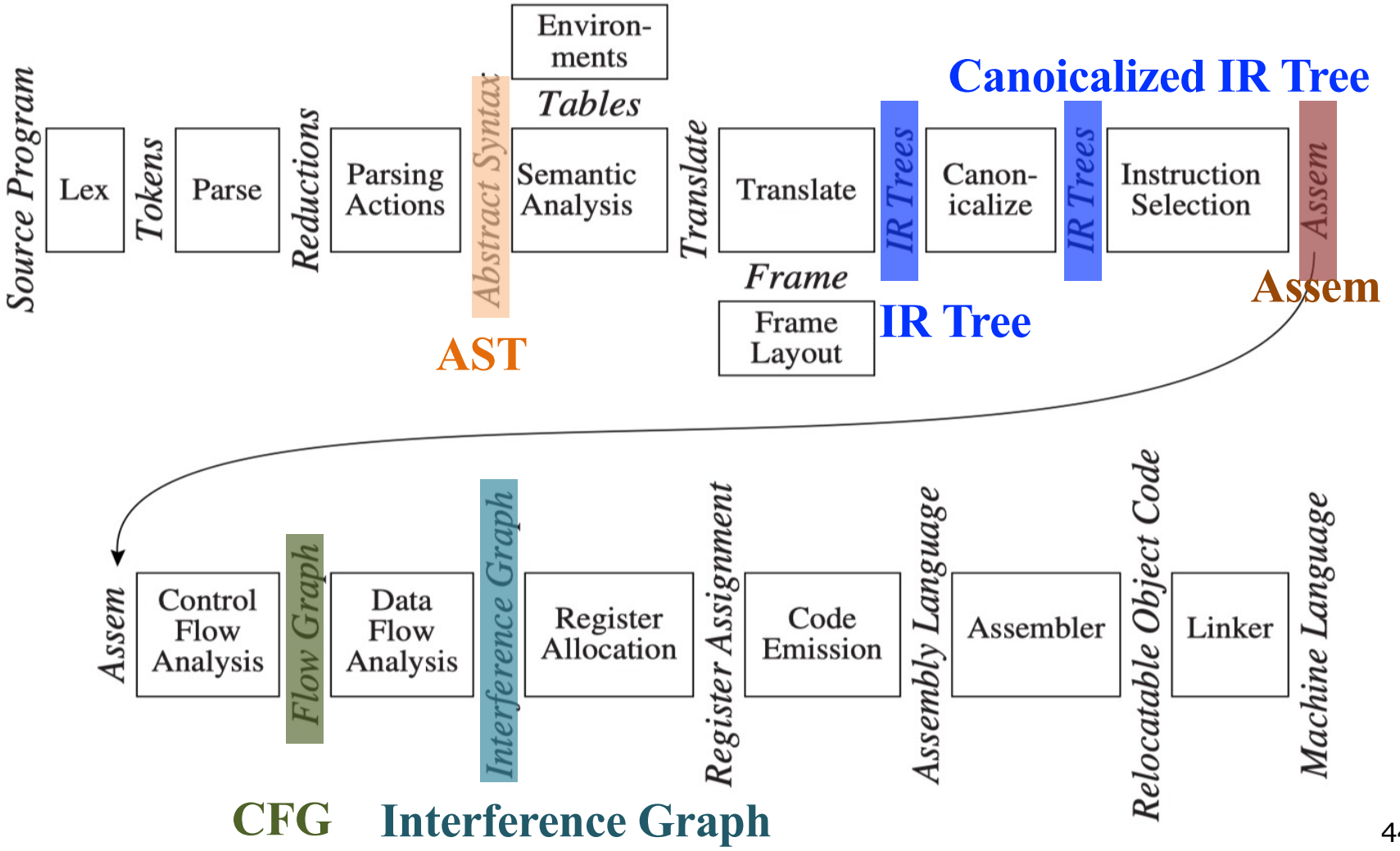
- 从源程序开始，经过若干步骤，最终变换成目标程序



- 例: Clang编译器(基于LLVM框架）的Passes
  - 分析 (Analysis Pass): 从IR中获得信息如Liveness
  - 变换 (Transformation Pass): 对IR做优化、生成新的IR

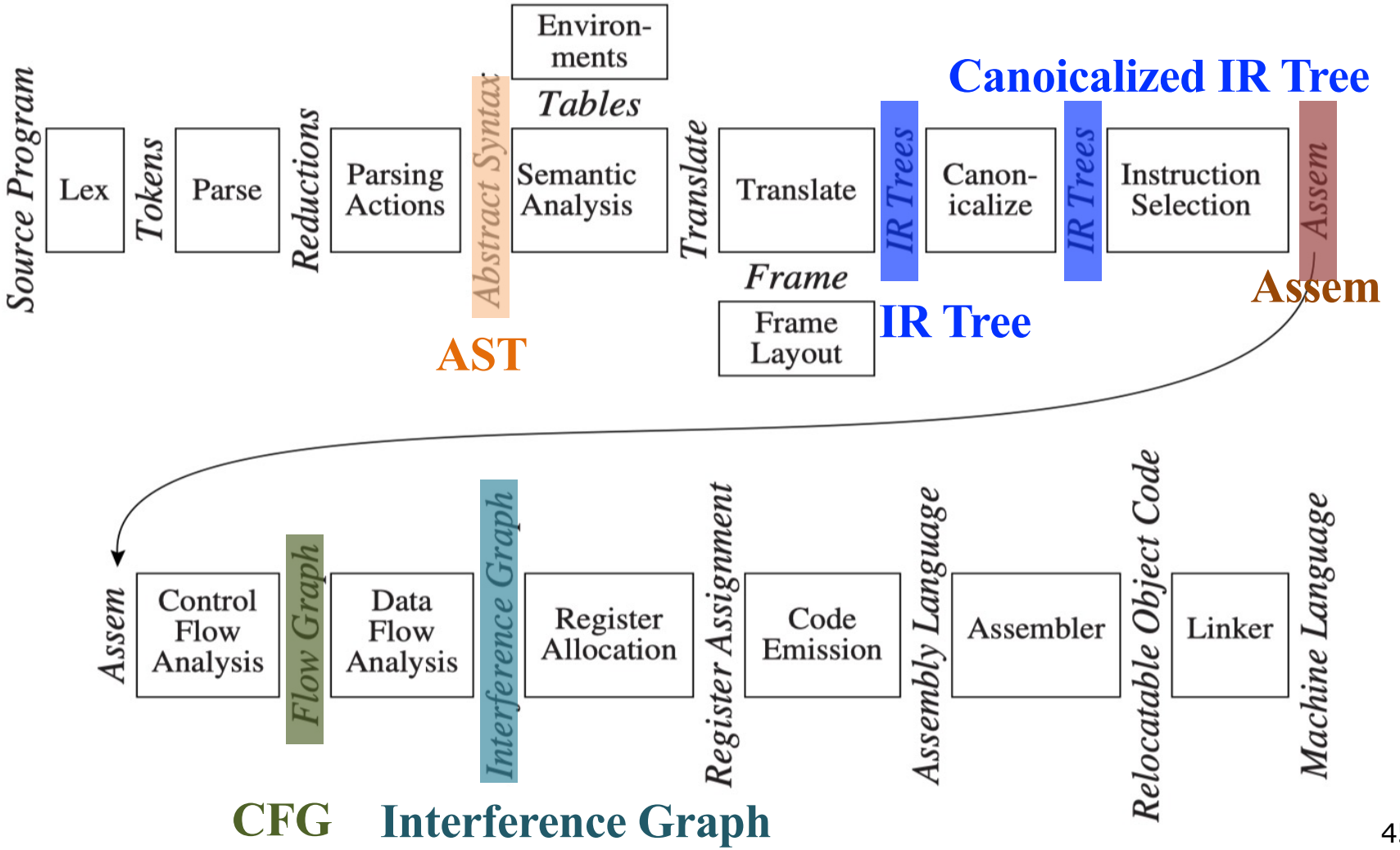
例: Tiger编译器的流程

- AST(抽象语法树): 语法分析 + “Parsing Actions”生成



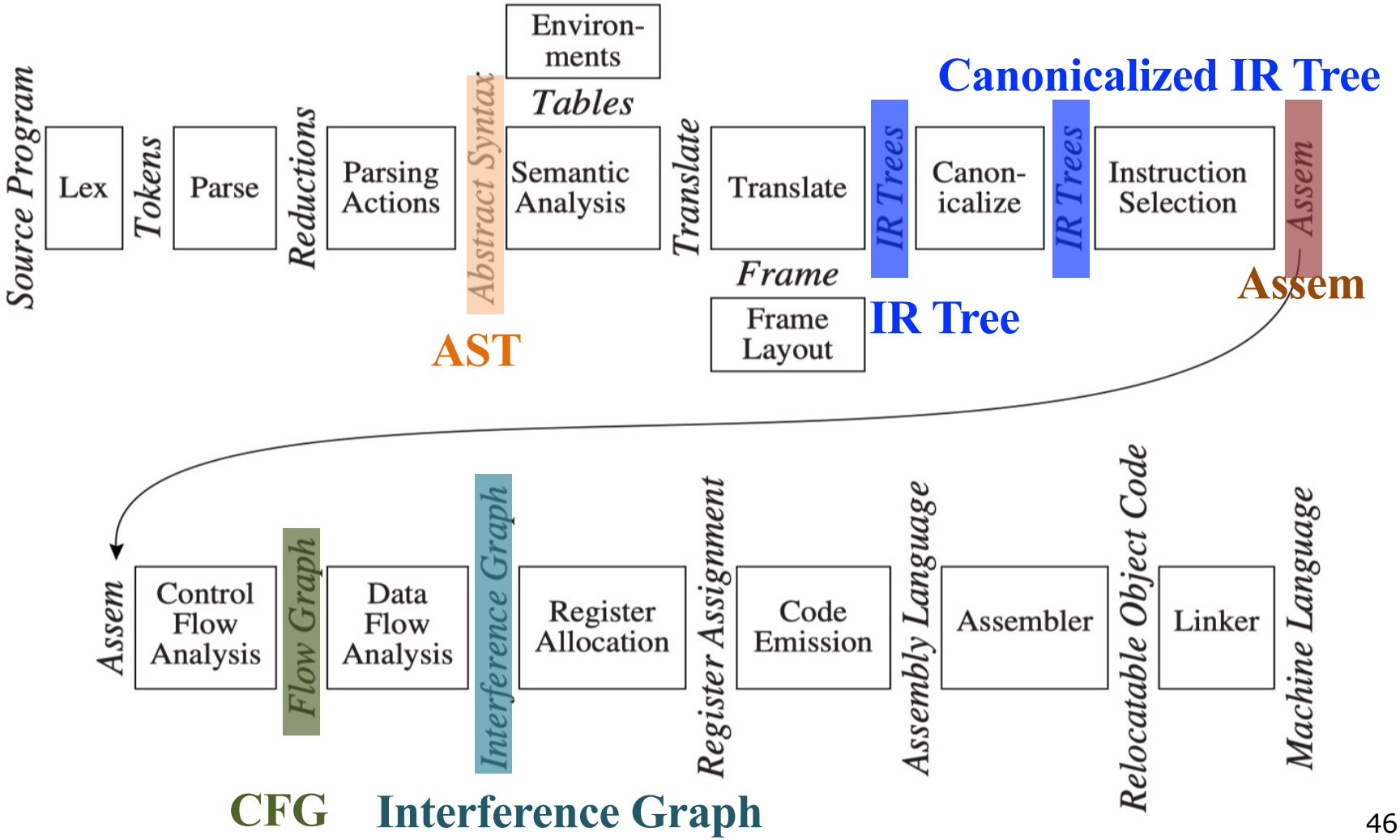
例: Tiger编译器的流程

- **IR Tree: 语义分析后按一定规则生成(树型中间表示)**



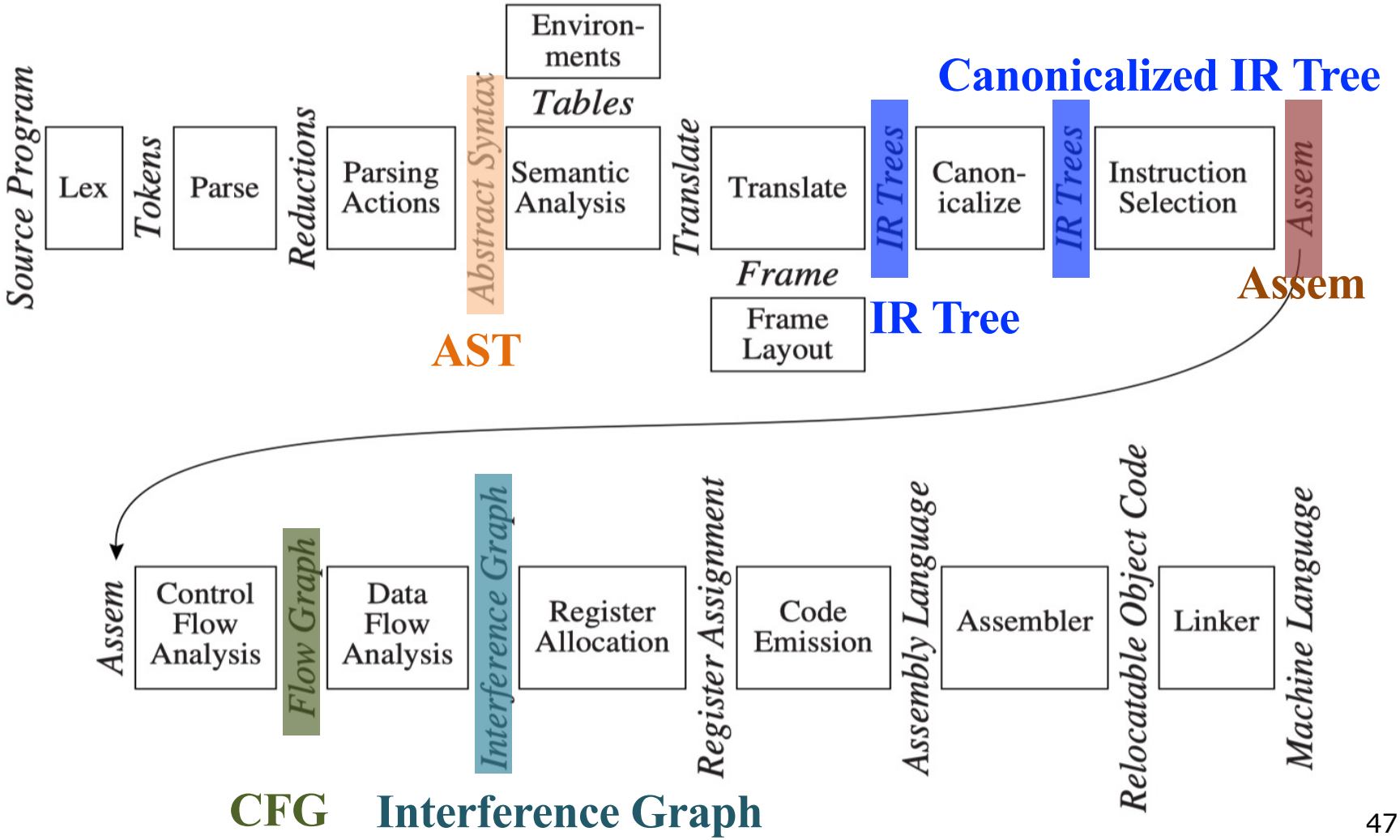
例: Tiger编译器的流程

- **Canonicalized IR Tree: 对IR Tree做变换所得(方便生成汇编)**



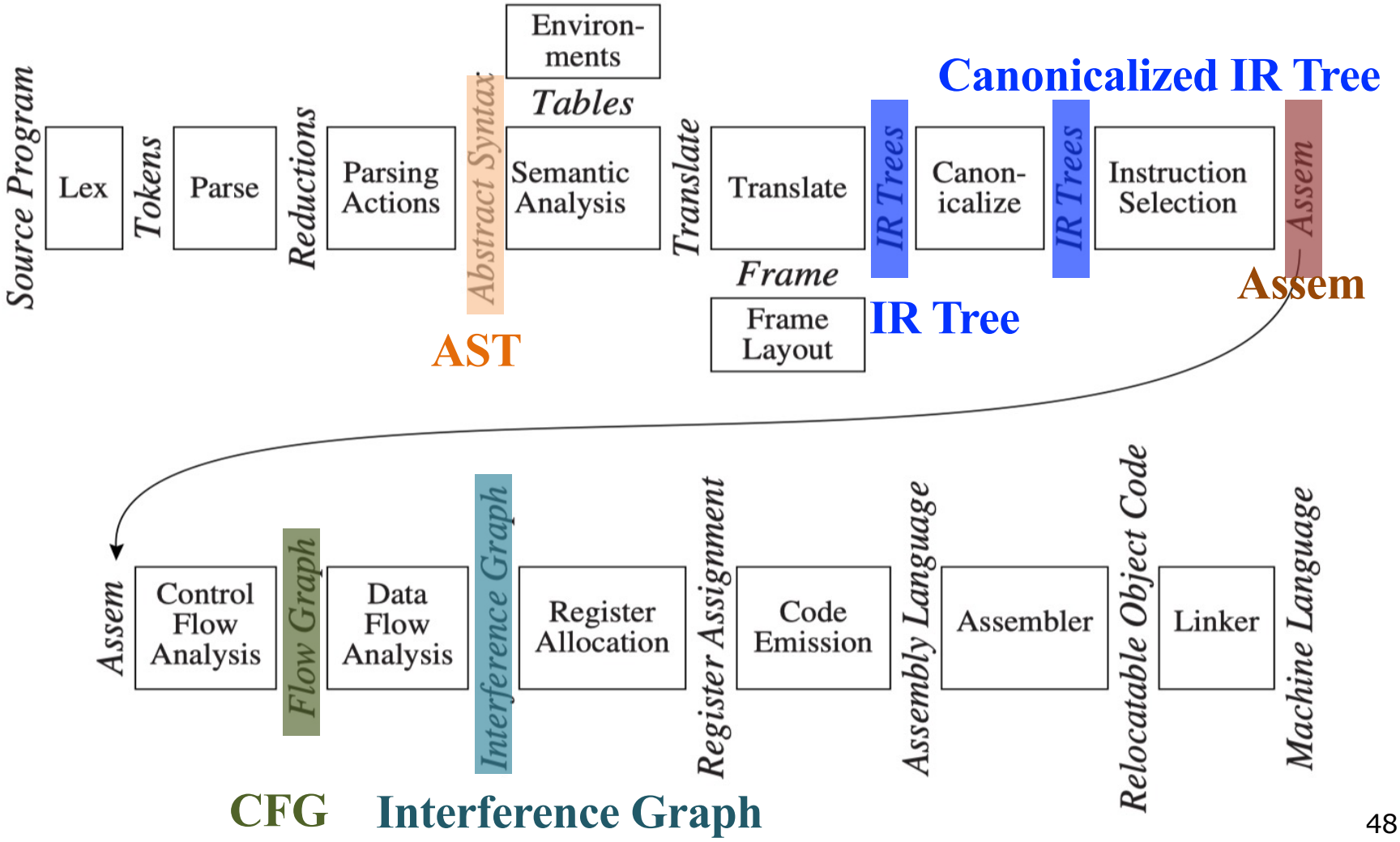
例: Tiger编译器的流程

- **Assem: 指令选择器生成(一种特殊的汇编)**



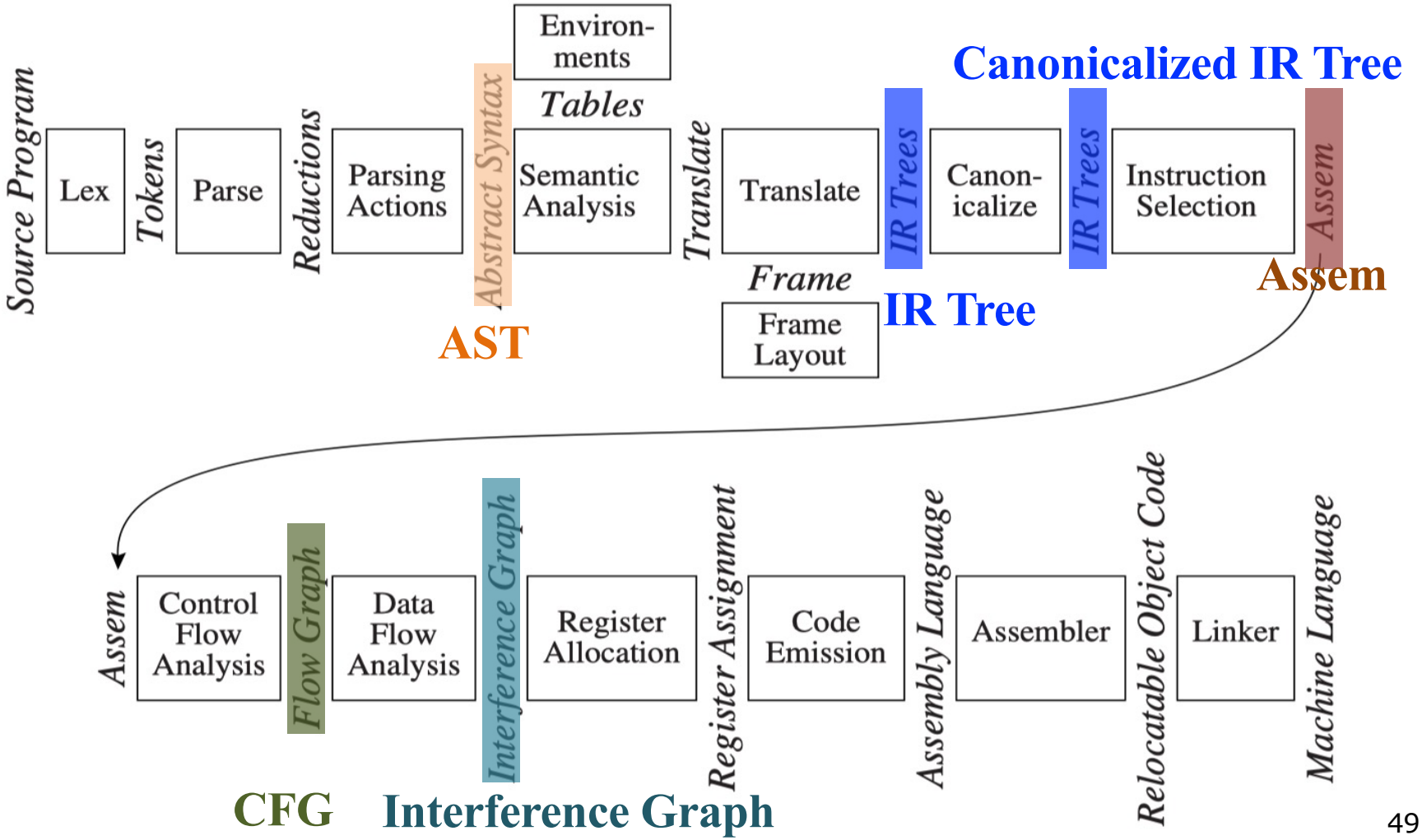
例: Tiger编译器的流程

- CFG(Control Flow Graph) 控制流图: 方便进行数据流分析
  - 如活跃变量分析(Liveness Analysis)



例: Tiger编译器的流程

- **Interference Graph: 从活跃变量分析的结果构造，用于指导寄存器分配**



What is the Key Mission of Computer Science?

"To help people turn creative ideas into working systems"

Thank you all for your attention