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

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
导论

计算机科学与技术学院

李 诚

03/09/2018

主要内容

- 课程设置情况
- 编译器的由来与挑战
- 编译器的构造

课程设置

- 时间：每周一(6,7)、四(3,4)
- 地点：3B201
- 课程主页（课件、试题等）：
  - http://staff.ustc.edu.cn/~chengli7/courses/compiler18/
- 邮件列表：
  - 我们会自动将大家的邮箱加入
- QQ讨论群：
  - 把编译进行到底（群号：851075885）

- 李诚 （先进数据系统实验室，研究方向：大规模、实时、高可靠分布式系统）
  - Contact: chengli7@ustc.edu.cn
  - Official Hours：每周一12:00 – 13:30
  - 地点：东校区高性能中心503

助教团队

白有辉（博一）
byh0912@mail.ustc.edu.cn

王佳玮（研二）
wang.jw@yahoo.com

邵新洋（研二）
sxy799@mail.ustc.edu.cn

王一多（研一）
duo@mail.ustc.edu.cn

许冠斌（研一）
Web manager

参考资料

❑教材和参考书

- ❖陈意云、张昱，编译原理（第3版），高等教育出版社，2014
- ❖A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools , 2nd edition, Addison-Wesley, 2007
- ❖A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman 著，赵建华等译，编译原理，机械工业出版社，2017

❑其他资料

- ❖Stanford课程主页

http://web.stanford.edu/class/cs143/

- ❖MIT课程主页：

http://6.035.scripts.mit.edu/fa18/

考核要求（Tentative）

- 考核内容包括理论学习和工程实践
- 理论学习（20%）：
  - 按时上课（特殊情况不能来需要书面请假）
  - 按时完成课后书面作业
- 工程实践（40%）：
  - 以组队（<=3人）完成若干项目（>=4）
  - 完成一个简单的编译器
  - 每个项目要求提交代码+文档
  - 通过git commit来评估同组人员的工作量
- 考试（40%）：
  - 期中（TBD，可以是开卷）
  - 期末（TBD，可以是开卷）

主要内容

- 课程设置情况
- **编译器的由来与挑战**
- **编译器的构造**

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
  - 声明型(Declarative): SQL, …
  - 逻辑型(Logic): Prolog, …
  - 函数式(Functional): Lisp/Scheme, Haskell, ML, Ocaml, …
  - 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).
```

快速发展的编程语言

- PYPL Index is created by analyzing how often language tutorials are searched on Google.

Worldwide, Aug 2018 compared to a year ago:

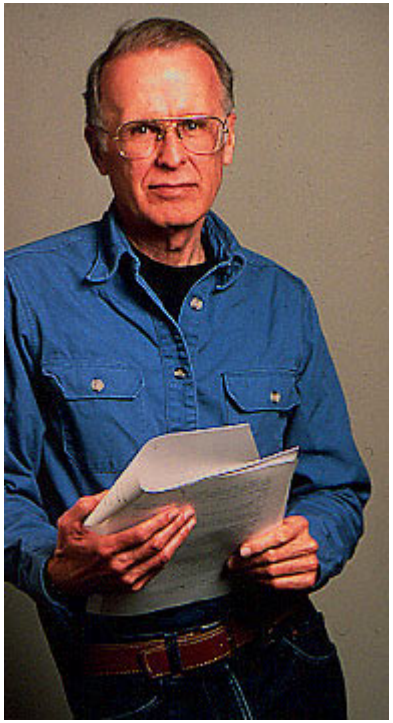
Rank	Change	Language	Share	Trend
1	↑	Python	24.21 %	+5.7 %
2	↓	Java	22.27 %	-0.7 %
3	↑	Javascript	8.45 %	+0.1 %
4	↓	PHP	7.88 %	-1.5 %
5		C#	7.74 %	-0.4 %
6		C/C++	6.19 %	-0.8 %
7	↑	R	4.15 %	-0.1 %
8	↓	Objective-C	3.33 %	-1.0 %

从无到有，由弱到强

- 1954年IBM研发了704机
- 但是，软件开发的成本超过了硬件
- 所有的程序均由汇编语言开发

FORTRAN I

- John Backus （1977图灵奖)
  - 巴科斯范式（Backus–Naur Form）

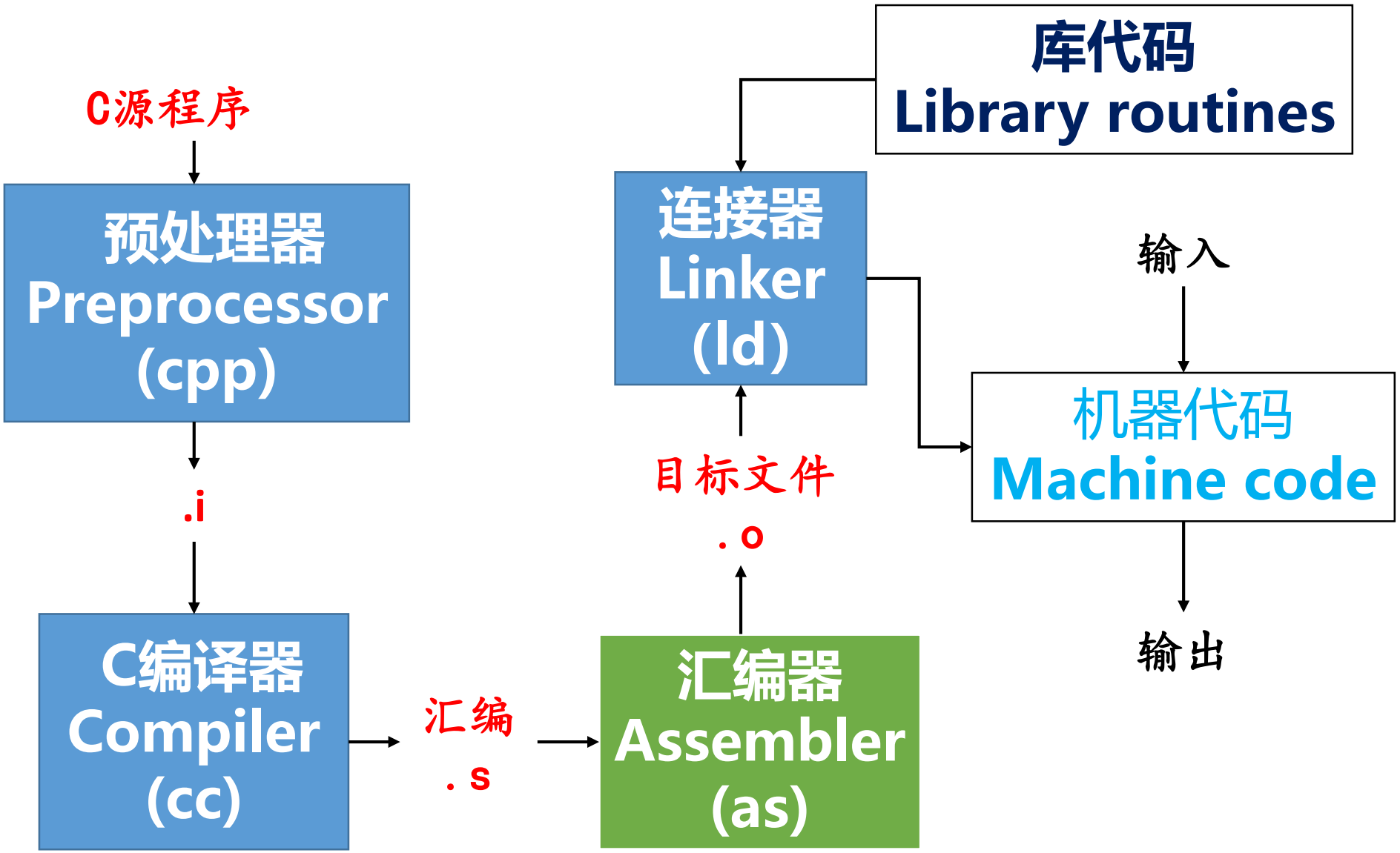


- 基本想法:
  - 将高级语言翻译为低级语言
  - 开发时间减半
- 对计算机科学影响巨大
  - 诞生了许多理论研究成果
  - 现代编译器还保留了FORTRAN I的大概架构

为什么需要编译器?

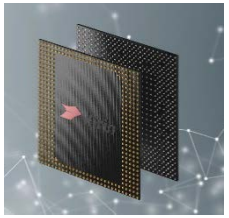
- 编译器使得开发者可以使用容易理解和掌握的高级语言，而非晦涩的机器指令。
  - 可移植性、模块化、简单化、编程效率高
  - 程序开销小、效率高
- 是不可或缺的编程工具
- 同时也是最复杂的系统软件之一

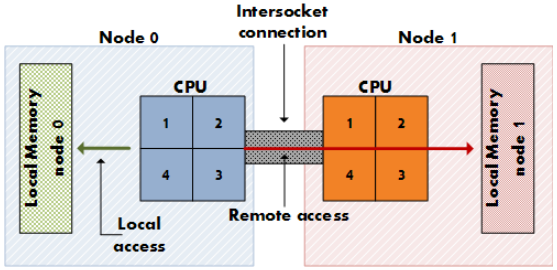
C程序的编译过程


C源程序
预处理器
Preprocessor
(cpp)
.i
C编译器
Compiler
(cc)
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

编译器面临的挑战

- 计算机是不断进化的
  - 体系结构的改变→编译器的改变
  - 新的特征产生新的问题





- 语言也在不断演化
  - C - C90, C99, C11； C++ - 1998, 2003, 2006, 2011, 2014
  - 新的语言不断诞生： Go (2009), Rust (2010), Elixir (2011), Swift (2014)

编译器是学科交叉的产物

- 计算机理论
  - 有限自动机, 文法, 数据流
- 算法
  - 树/图的遍历和修改，动态规划
- 数据结构
  - 符号表, 抽象语法树, 图
- 系统
  - 内存空间分配与命名, 多趟系统, 编译器的构造

编译器是学科交叉的产物

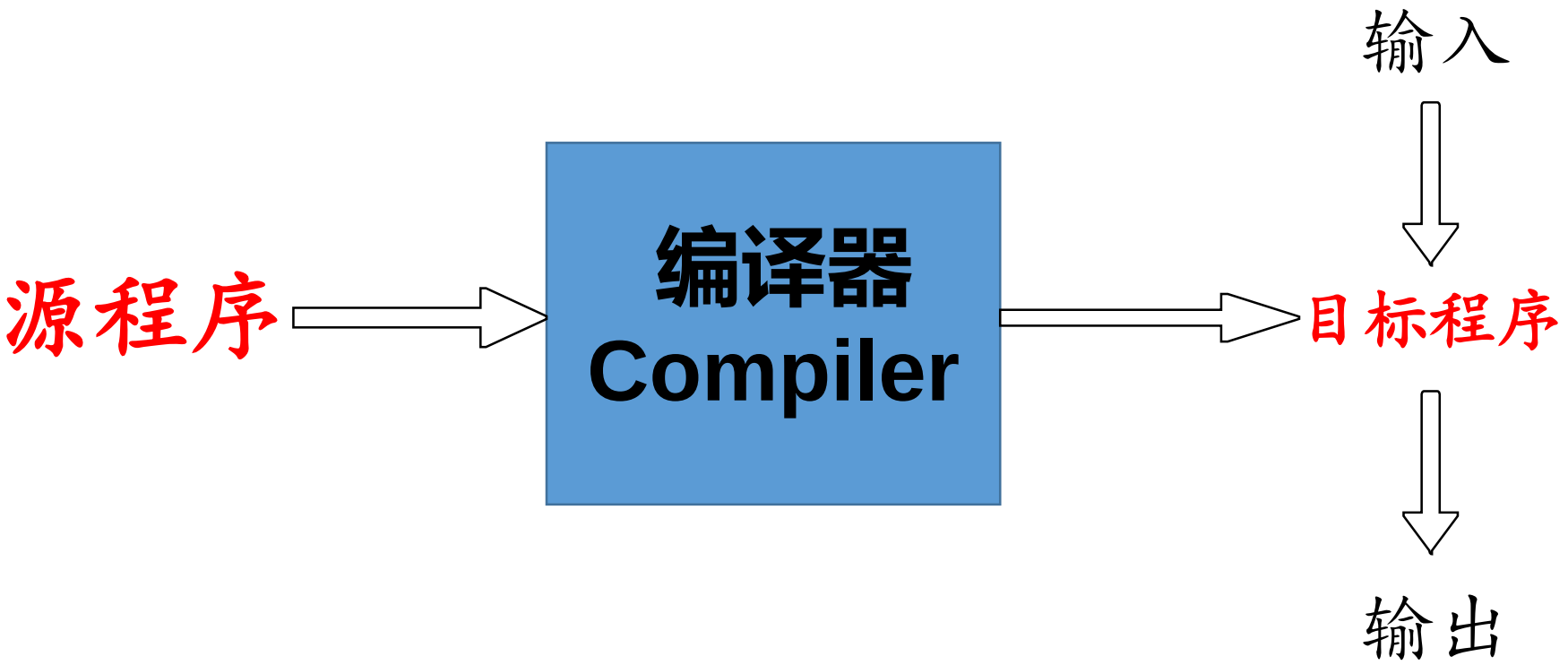
中国科学技术大学
University of Science and Technology of China

- 体系结构
  - 内存层次,指令选择,并行
- 安全
  - 寻找漏洞和攻击防御
- 软件工程
  - 软件开发环境, 调试
- 人工智能
  - 启发式代码优化

主要内容

- 课程设置情况
- 编译器的由来与挑战
- **编译器的构造**

编译器是什么?



编译器的输入

- 标准的指令式语言(Java, C, C++)
  - 状态
    - 变量
    - 结构
    - 数组
  - 计算
    - 表达式 (arithmetic, logical, etc.)
    - 赋值语句
    - 条件语句 (conditionals, loops)
    - 函数

编译器的输出

- 状态
  - 寄存器
  - 内存单元
- 机器码 – load/store architecture
  - Load, store instructions
  - 寄存器操作– Arithmetic, logical operations
  - 分支指令– Branch instructions

编译器的构造/阶段

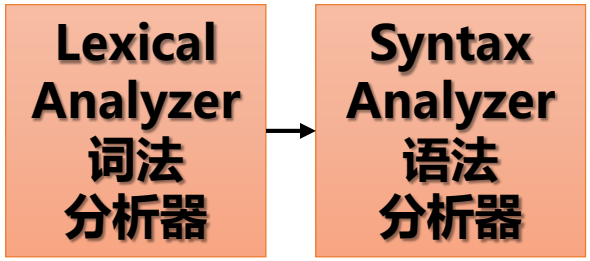
Lexical Analyzer 词法分析器

Source code 源程序

Token Stream 记号流

Symbol Table 符号表

编译器的构造/阶段

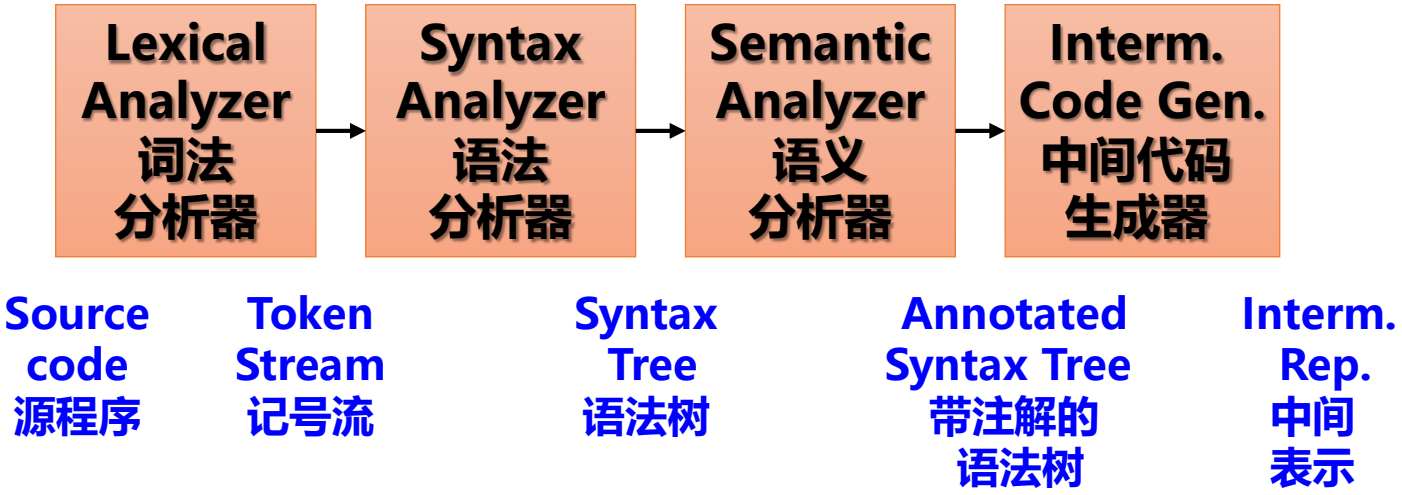


Symbol Table 符号表

编译器的构造/阶段

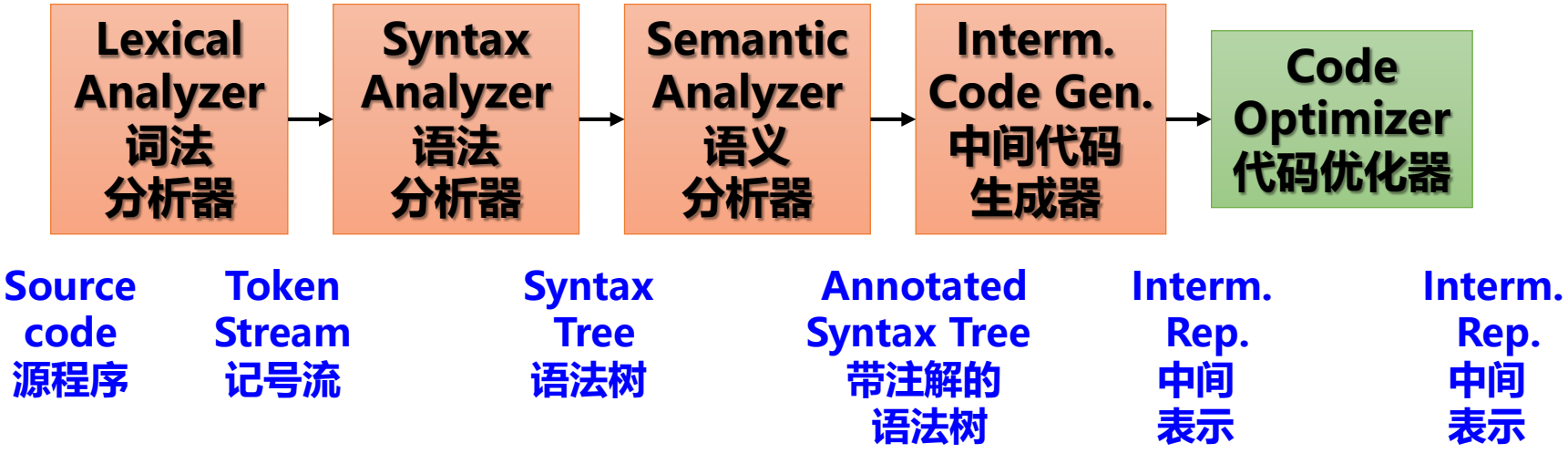
Symbol Table 符号表

编译器的构造/阶段

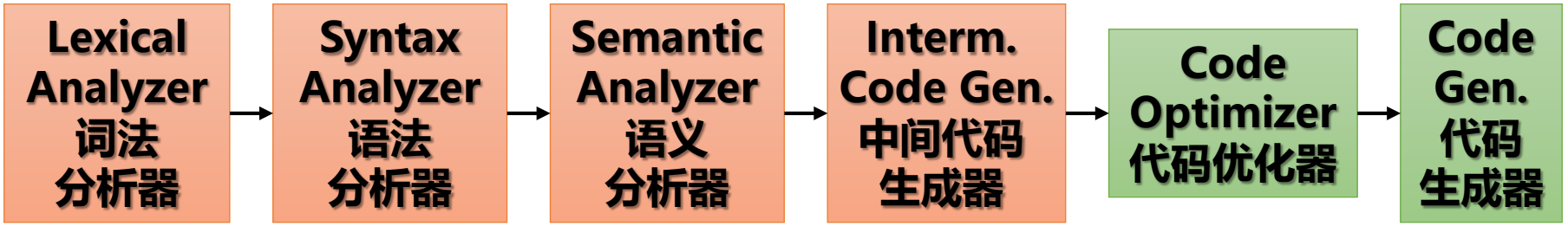


Symbol Table 符号表

编译器的构造/阶段

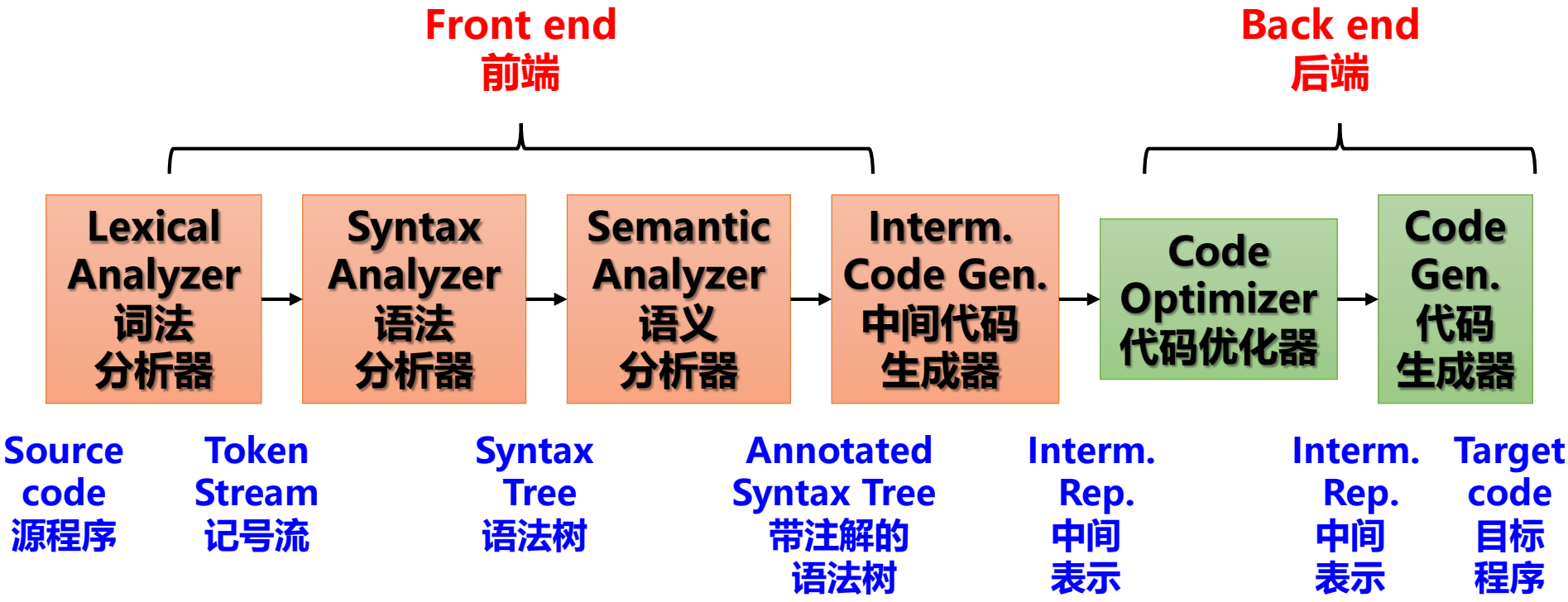

Lexical Analyzer 词法分析器
Syntax Analyzer 语法分析器
Semantic Analyzer 语义分析器
Interm. Code Gen. 中间代码生成器
Code Optimizer 代码优化器
Source code 源程序
Token Stream 记号流
Syntax Tree 语法树
Annotated Syntax Tree 带注解的语法树
Interm. Rep. 中间表示
Interm. Rep. 中间表示
Symbol Table 符号表

编译器的构造/阶段

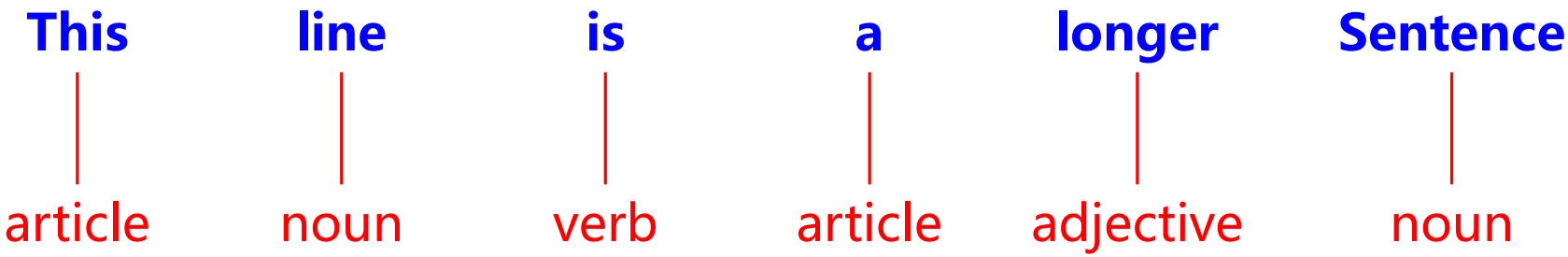


Symbol Table 符号表

编译器的构造/阶段


Front end
前端
Back end
后端
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
Source code 源程序
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
Target code 目标程序
Symbol Table 符号表

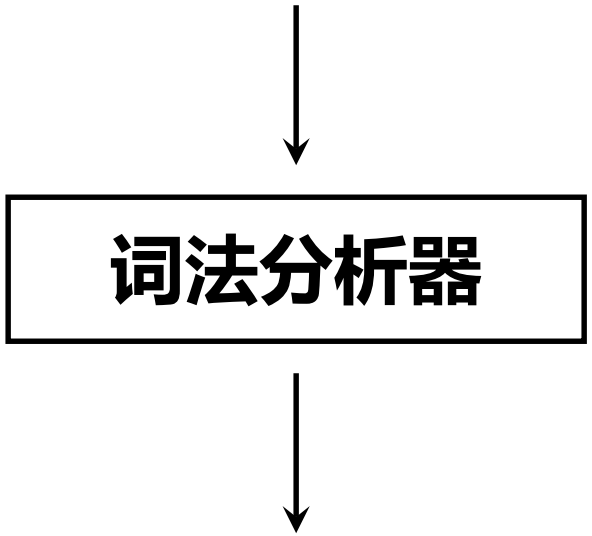
❑人类在理解自然语言时，首先要识文断字



❑将程序字符流分解为记号（Token）序列

❖形式：<token_name, attribute_value>

position = initial + rate * 60 ← 字符流

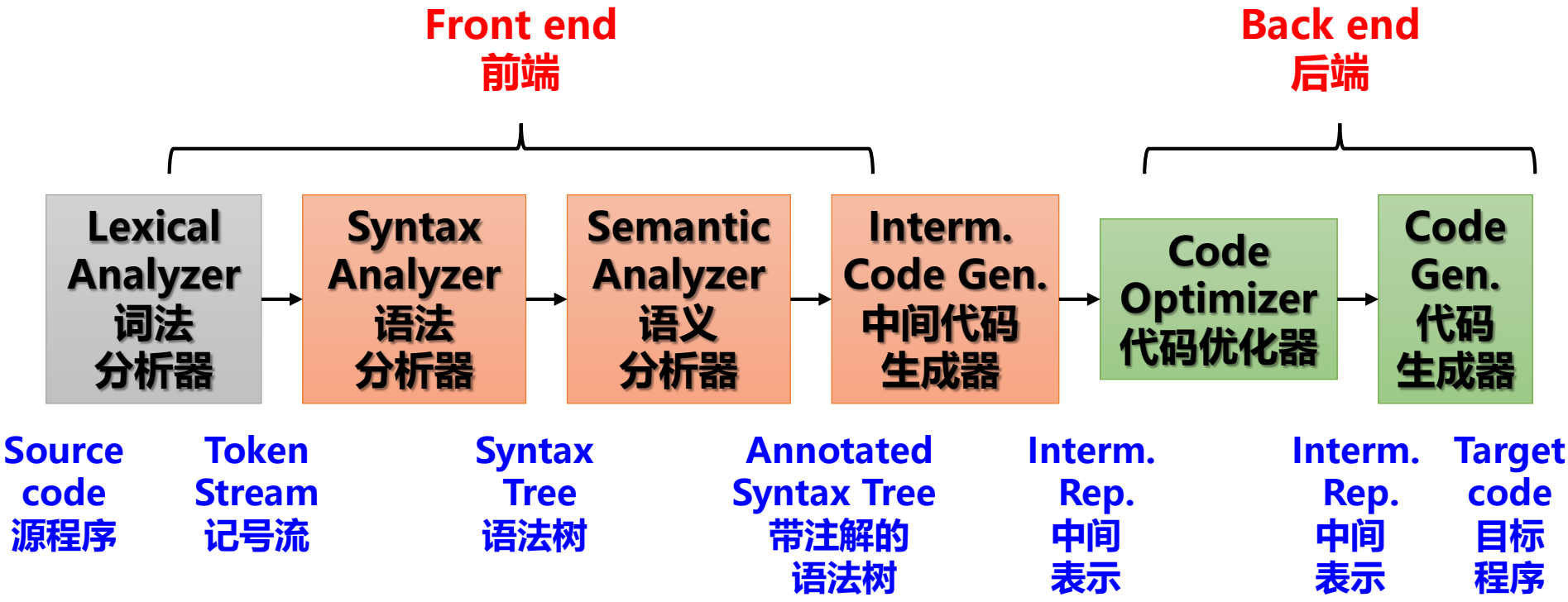


符 号 表

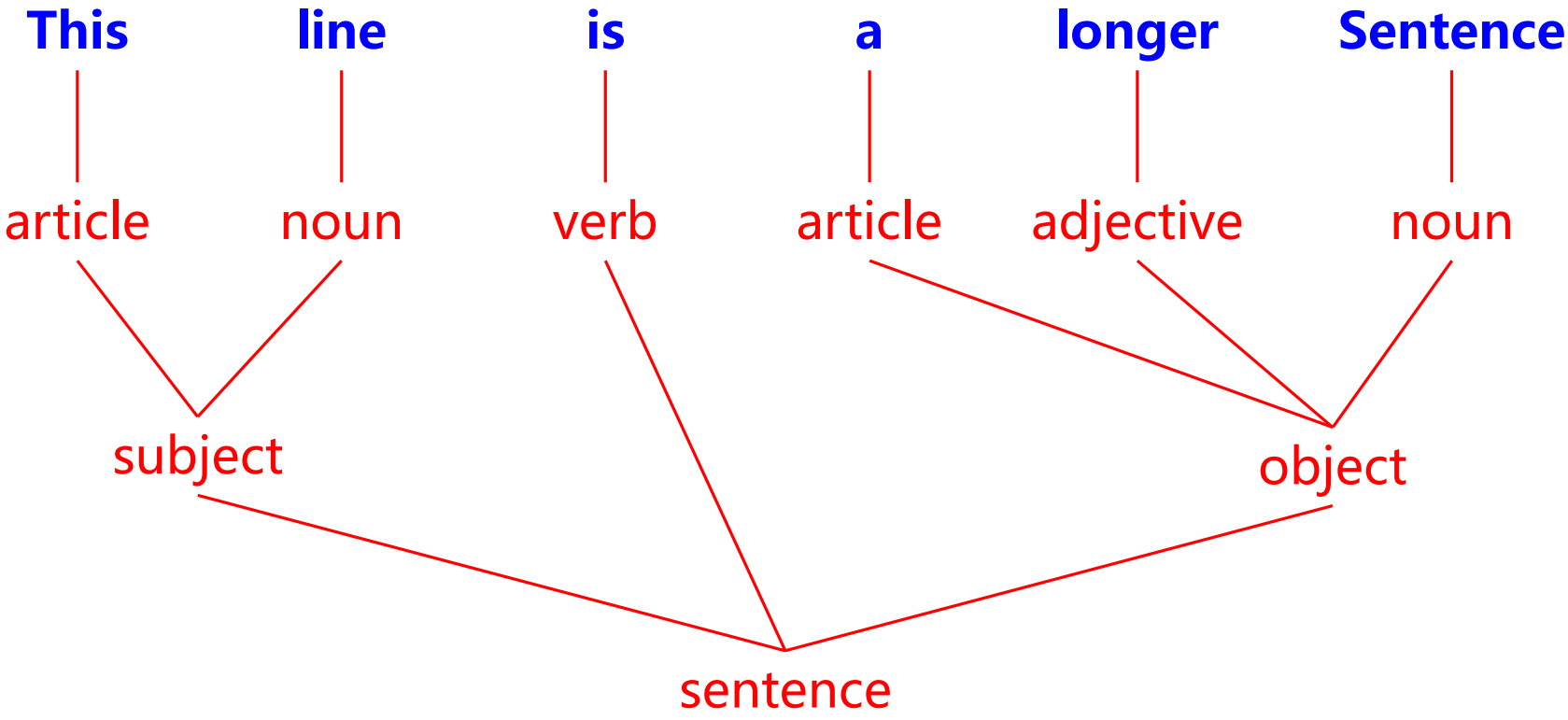
1	position	· · ·
2	initial	· · ·
3	rate	· · ·

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

编译器的构造/阶段

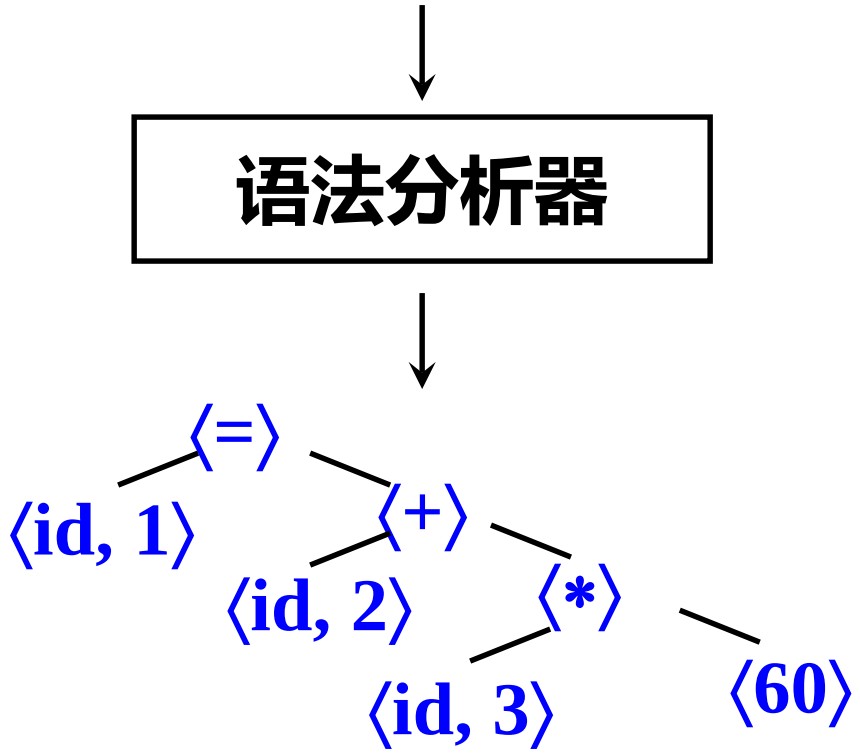

Front end
前端
Back end
后端
Lexical Analyzer 词法分析器
Syntax Analyzer 语法分析器
Semantic Analyzer 语义分析器
Interm. Code Gen. 中间代码生成器
Code Optimizer 代码优化器
Code Gen. 代码生成器
Source code 源程序
Token Stream 记号流
Syntax Tree 语法树
Annotated Syntax Tree 带注解的语法树
Interm. Rep. 中间表示
Interm. Rep. 中间表示
Target code 目标程序
Symbol Table 符号表

- 人类在理解自然语言时，其次要理解句子结构



- 也称为解析（Parsing），在词法记号的基础上，创建语法结构

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

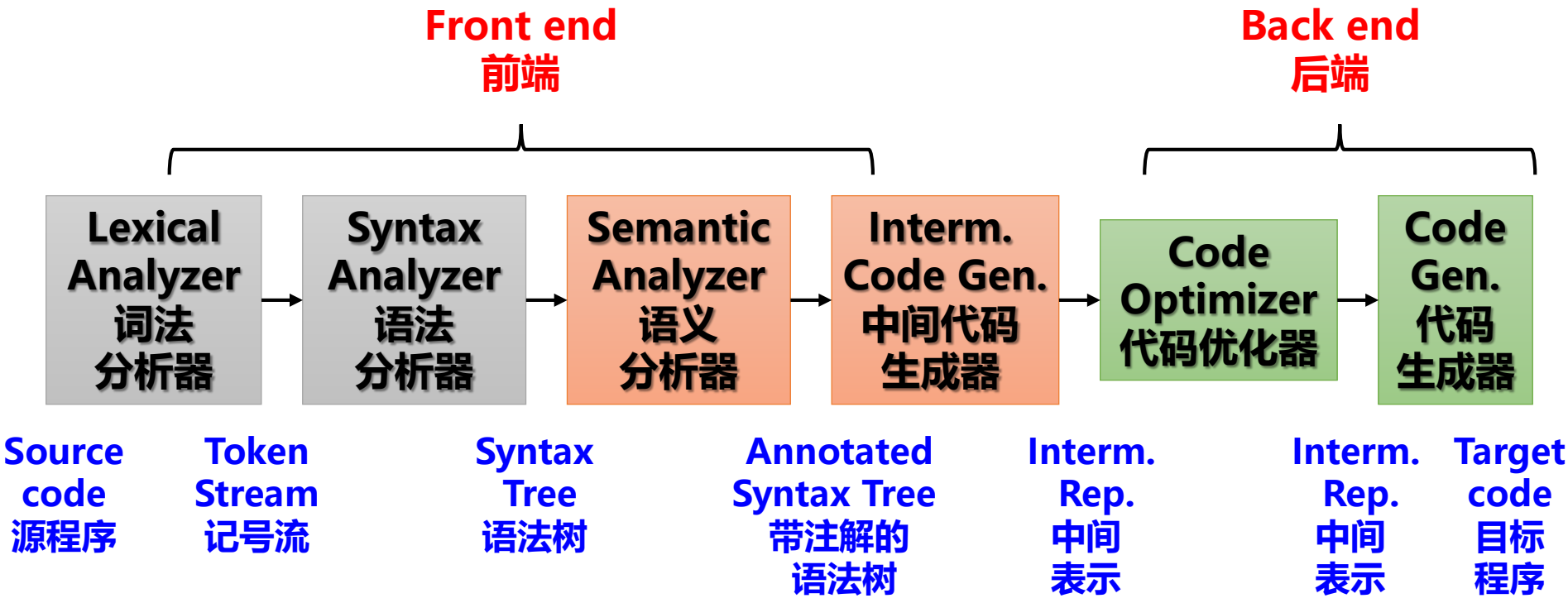


← 语法树

符 号 表

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

编译器的构造/阶段

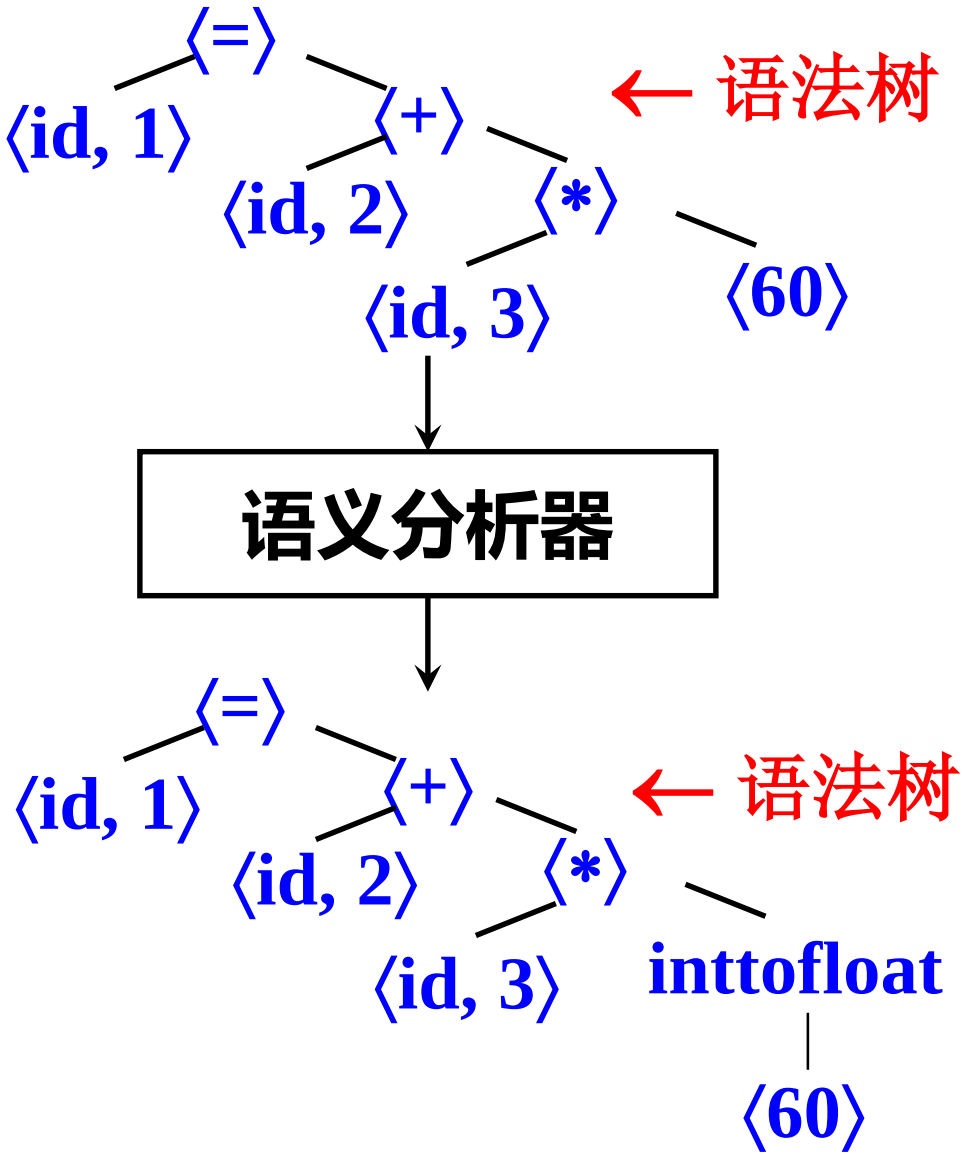

Front end
前端
Back end
后端
Lexical Analyzer 词法分析器
Syntax Analyzer 语法分析器
Semantic Analyzer 语义分析器
Interm. Code Gen. 中间代码生成器
Code Optimizer 代码优化器
Code Gen. 代码生成器
Source code 源程序
Token Stream 记号流
Syntax Tree 语法树
Annotated Syntax Tree 带注解的语法树
Interm. Rep. 中间表示
Interm. Rep. 中间表示
Target code 目标程序
Symbol Table 符号表

- 人类在理解自然语言时，最后要理解句子的含义
  - Jack said Jerry left his assignment at home.
    - What does “his” refer to? Jack or Jerry?

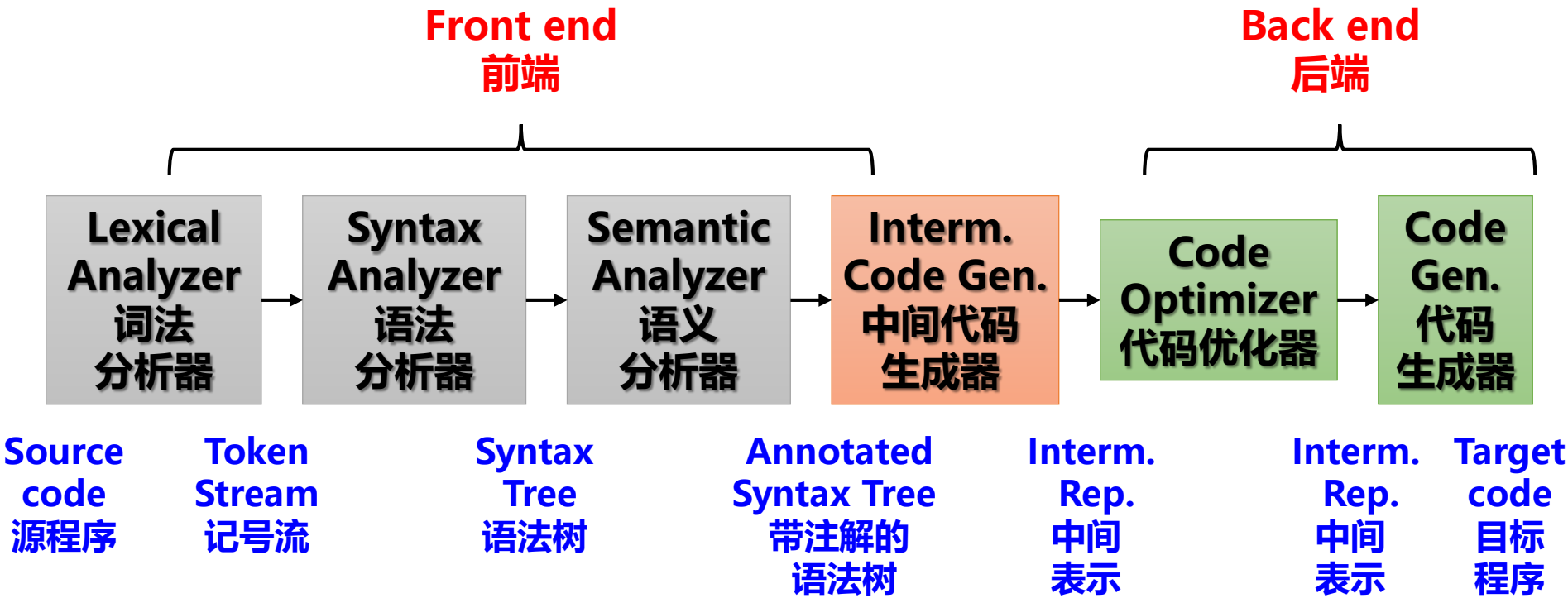
- 编译器会检查程序中的不一致
  - 如：类型检查（type checking）

符 号 表

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

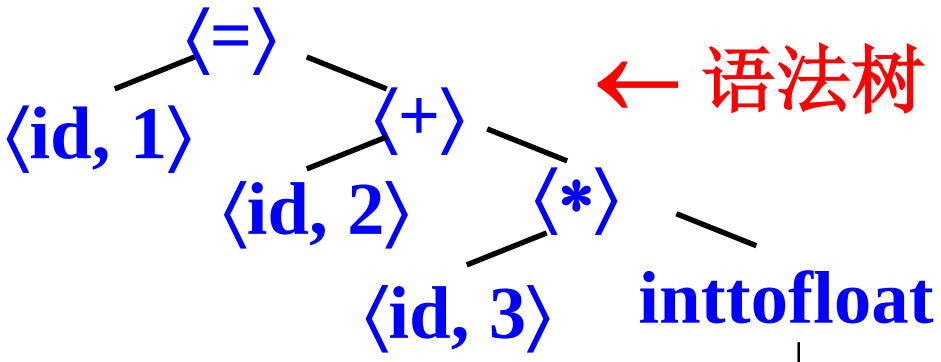


编译器的构造/阶段


Front end
前端
Back end
后端
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
Source code 源程序
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
Target code 目标程序
Symbol Table 符号表

中间代码生成

- 是源语言与目标语言之间的桥梁



符 号 表

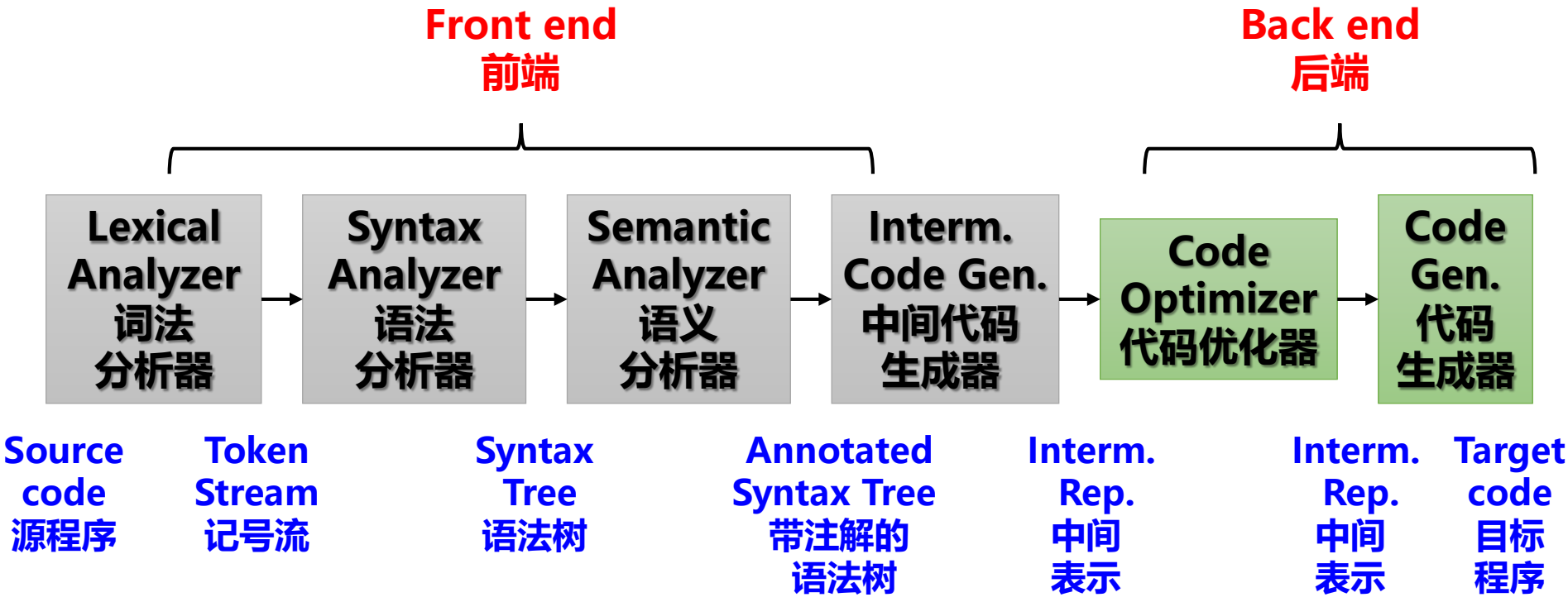
1	position	· · ·
2	initial	· · ·
3	rate	· · ·

中间代码生成器

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

← 中间代码

编译器的构造/阶段


Front end
前端
Back end
后端
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
Source code 源程序
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
Target code 目标程序
Symbol Table 符号表

- 机器无关的代码优化便于生成执行时间更快、更短或能耗更低的目标代码

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

← 中间代码

符 号 表

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

↓

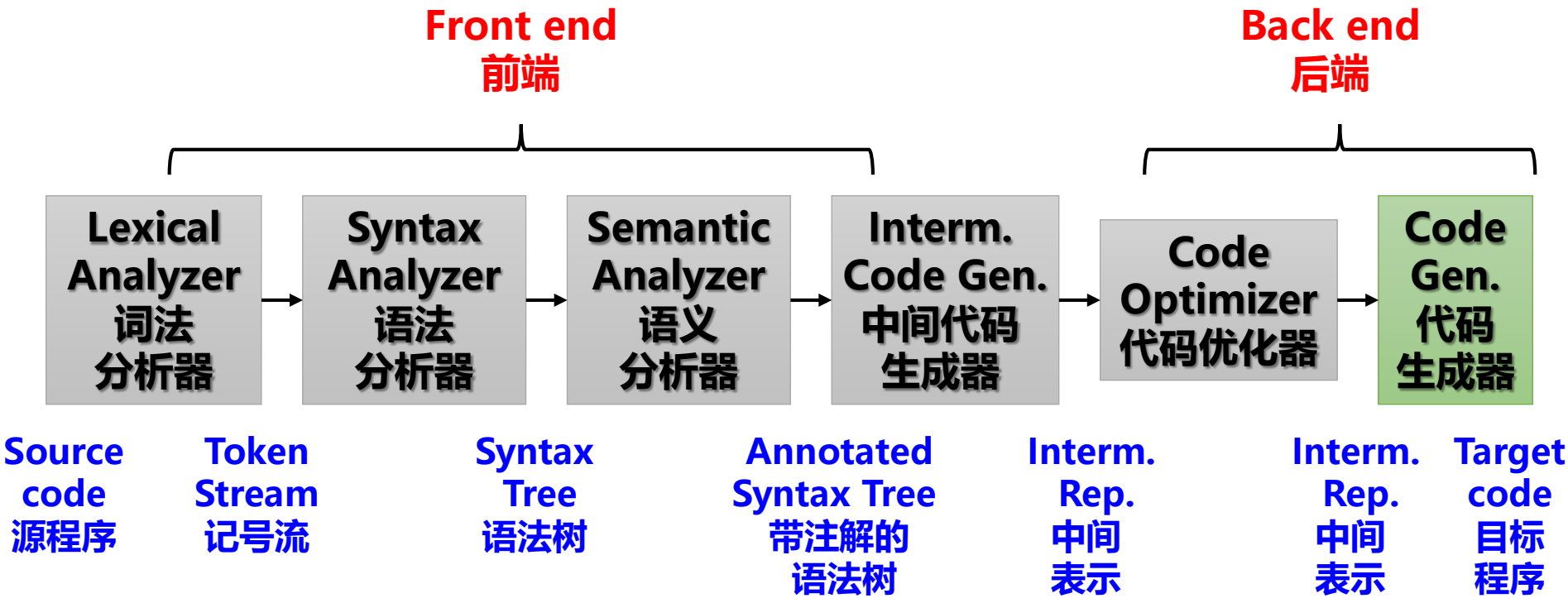
代码优化器

↓

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

← 中间代码

编译器的构造/阶段


Front end
前端
Back end
后端
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
Source code 源程序
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
Target code 目标程序
Symbol Table 符号表

代码生成

- 如果目标语言是机器代码，必须为变量选择寄存器或内存位置

符 号 表

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

t1 = id3 * 60.0
id1 = id2 + t1 ← 中间代码

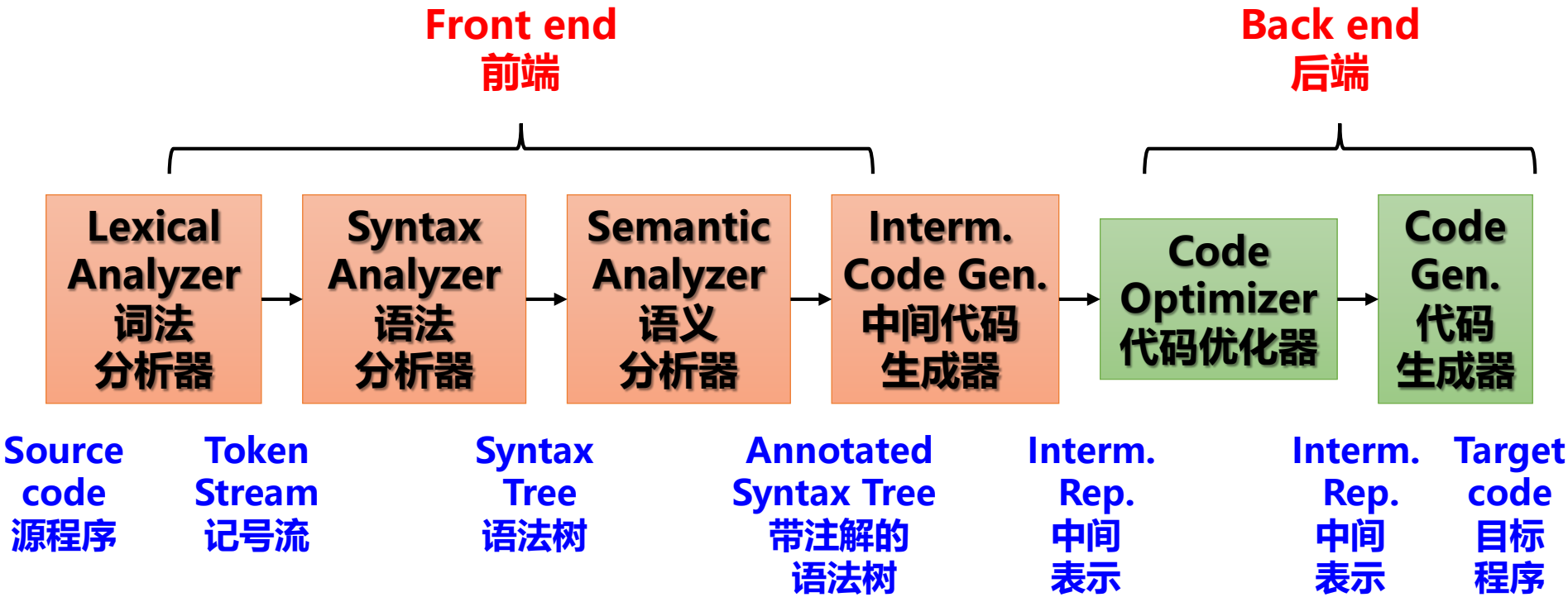
↓

代码生成器

↓

LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2 ← 汇编代码
ADDF R1, R1, R2
STF id1, R1

编译器的构造/阶段


Front end
前端
Back end
后端
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
Source code 源程序
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
Target code 目标程序
Symbol Table 符号表

《编译原理与技术》

导论

笋因落箨方成竹，鱼为奔波始化龙。

曾记少年骑竹马，看看又是白头翁。

——出自《增广贤文》