



软件工程专业本科课程

编译原理

Compilers Principles

长安大学信息工程学院软件工程系

主讲：李 鹏

课程简介

课程内容

- 介绍编译器构造的一般原理和基本实现方法；
- 介绍的理论知识：形式语言和自动机理论、语法制导的定义和属性文法、类型论等；
- 强调形式化描述技术，并以语法制导定义作为翻译的主要描述工具；
- 强调对编译原理和技术的宏观理解。

学习的意义

- 对编程语言的设计和实现有深刻的理解，对和编程语言有关的理论有所了解；
- 所介绍的概念和技术能应用到一般的软件设计之中；
- 有助于提高程序语言的设计水平。
- 在软件逆向工程、软件再工程、程序理解和软件安全等方面有着广泛的应用。

与其它课程联系



教材和参考书

- 编译原理，作者：陈意云 张昱，高等教育出版社（教材）
- 编译原理 技术与工具(第二版)(英文版) **Compilers: Principles, Techniques, and Tools (2nd Edition)**，作者：Alfred V.Aho等，人民邮电出版社（龙书）
- 编译原理 技术与工具（英文版） **Compilers: Principles, Techniques and Tools**，作者：Alfred V.Aho等，人民邮电出版社（龙书）
- 现代编译原理—C语言描述 **Modern Compiler Implementation in C**，译者：赵克佳 黄春 沈志宇，人民邮电出版社（虎书）
- 现代编译器的Java实现 **Modern Compiler Implementation in Java**，译者：陈明，电子工业出版社（虎书）

教材和参考书

- 高级编译器设计与实现 **Advanced Compiler Design and Implementation** , 译者: 赵克佳等, 机械工业出版社 (鲸书)
- 深入理解计算机系统 **Computer Systems A Programmer's Perspective**, 译者: 龚奕利 雷迎春, 中国电力出版社
- 程序设计语言—实践之路 **Programming Language Pragmatics**, 译者: 裘宗燕, 电子工业出版社
- **LEX 与 YACC (第二版) Lex & Yacc, Second Edition**, 译者: 杨作梅 张旭东等, 机械工业出版社

课程要求

- 学期总评

考试成绩	70%
------	-----

程序设计	20%
------	-----

平时及作业	10%
-------	-----

- 作业要求

课后习题、程序源代码压缩后发至

chd.lipeng@gmail.com

主题： **P/H+专业+班号+学号+次数**

例如： 软件工程专业一班1号第3次编程作业

P06010103

软件工程专业一班1号第2次课后习题作业

H06010102


第一章 编译器概述

- 翻译器、编译器、解释器；
- 编译器从逻辑上可以分成若干阶段；
- 每个阶段把源程序从一种表示变换成另一种表示；
- 通过描述编译器的各个阶段来介绍编译系统。

程序设计语言，通常简称为编程语言，是一组用来定义计算机程序的语法规则。它是一种被标准化的交流技巧，用来向计算机发出指令。一种计算机语言让程序员能够准确地定义计算机所需要使用的数据，并精确地定义在不同情况下所应当采取的行动。（[wikimedia 维基百科](#)）

程序设计语言涉及以下四个方面：

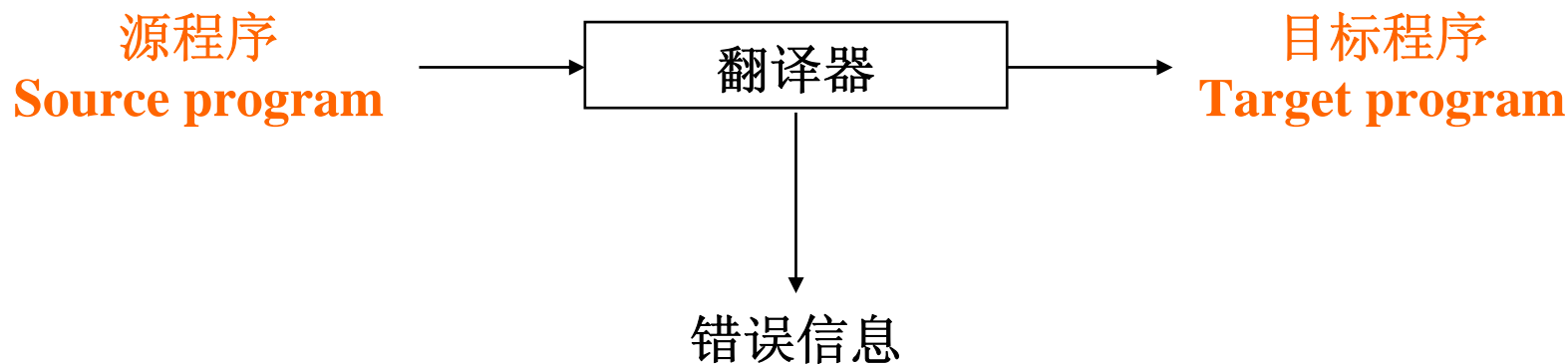
- 语法：表示构成语言句子的各个记号之间的组合规律；
- 语义：表示按照各种表示方法所表示的各个记号的特定含义；
- 语用：表示在各个记号所出现的行为中，它们的来源、使用和影响；
- 语境：指理解和实现程序设计语言的环境，包括编译环境和运行环境。



由于计算机硬件只懂得自己的指令系统，因此对于高级程序设计语言编写的程序无法直接识别。

为解决这个问题，我们需要对所编写的程序进行改进，改进的方法有两种：**翻译和解释**。

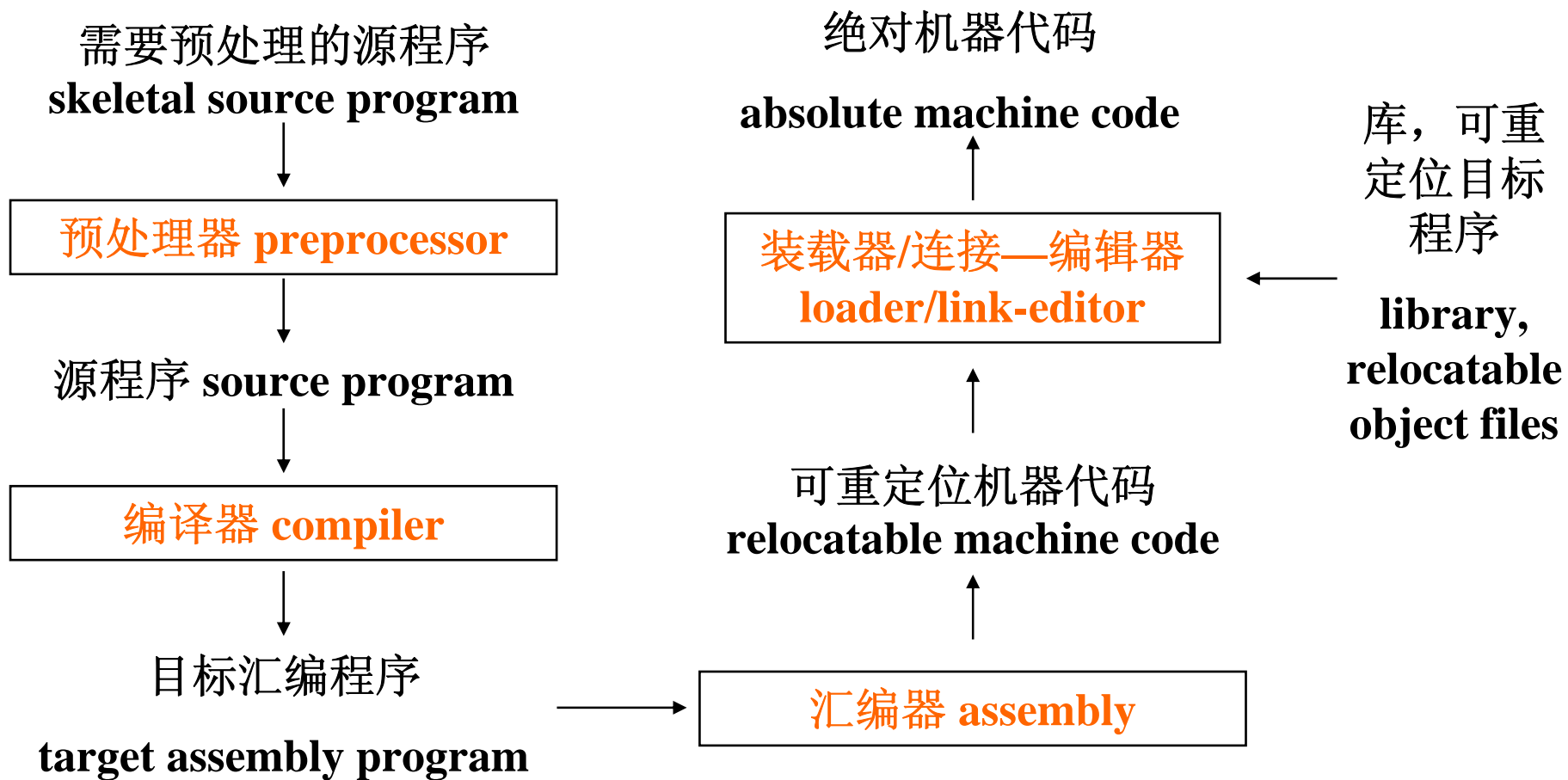
翻译: 把一种语言编写的程序（源程序）通过一个翻译器翻译成为与之等价的另一种语言的程序（目标程序）。



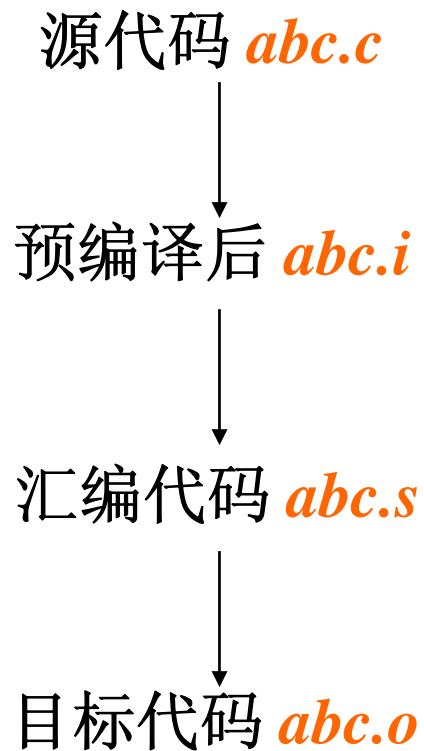
根据源语言与目标语言的不同，有着不同的翻译器。

- 源：汇编语言 目：机器语言 汇编程序（asm）；
- 源：高级语言 目：机器/汇编语言 编译器（compiler）；
- 源语言所包含内容比目标语言大，这时我们可以通过一个翻译程序把源语言超出的成分翻译成目标语言来表示。这个翻译器称为预处理器（preprocessor）；
- 源语言为低层次语言，目标语言为高级语言，翻译器称为反汇编器（disassembler）或反编译器（decompiler）。

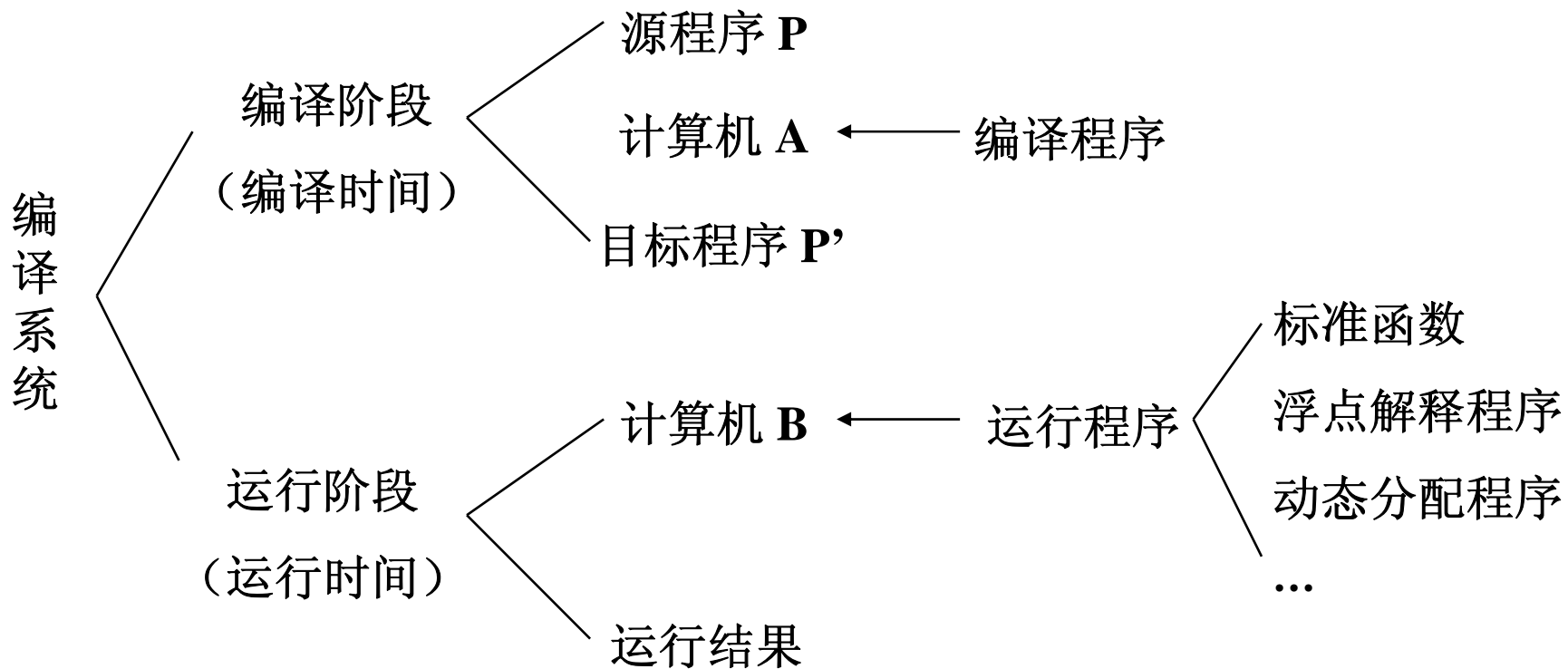
程序设计语言的典型处理过程如下：



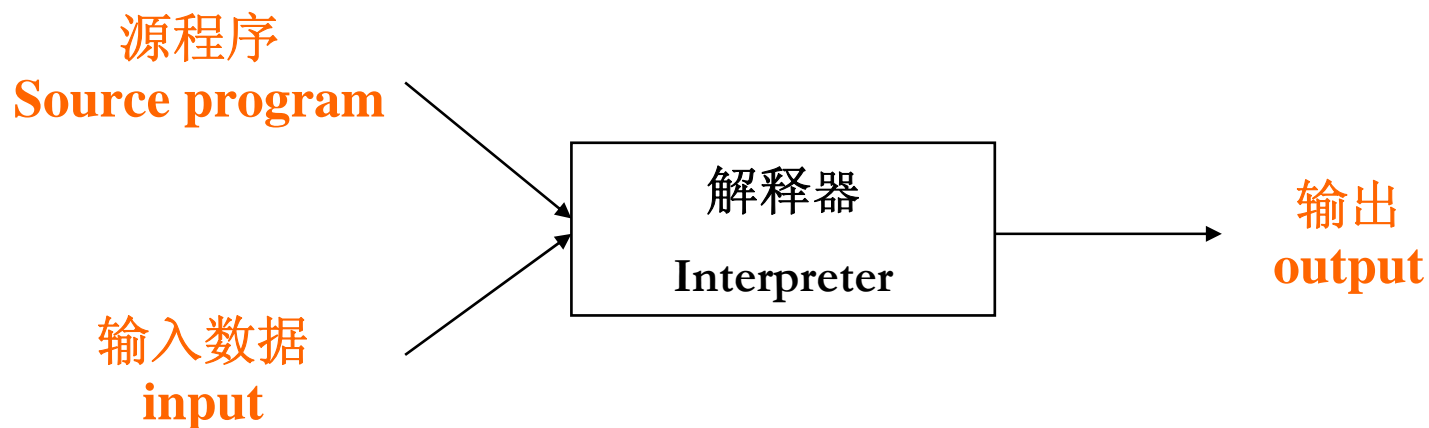
C语言编译举例：




编译程序和运行系统合称为**编译系统**。



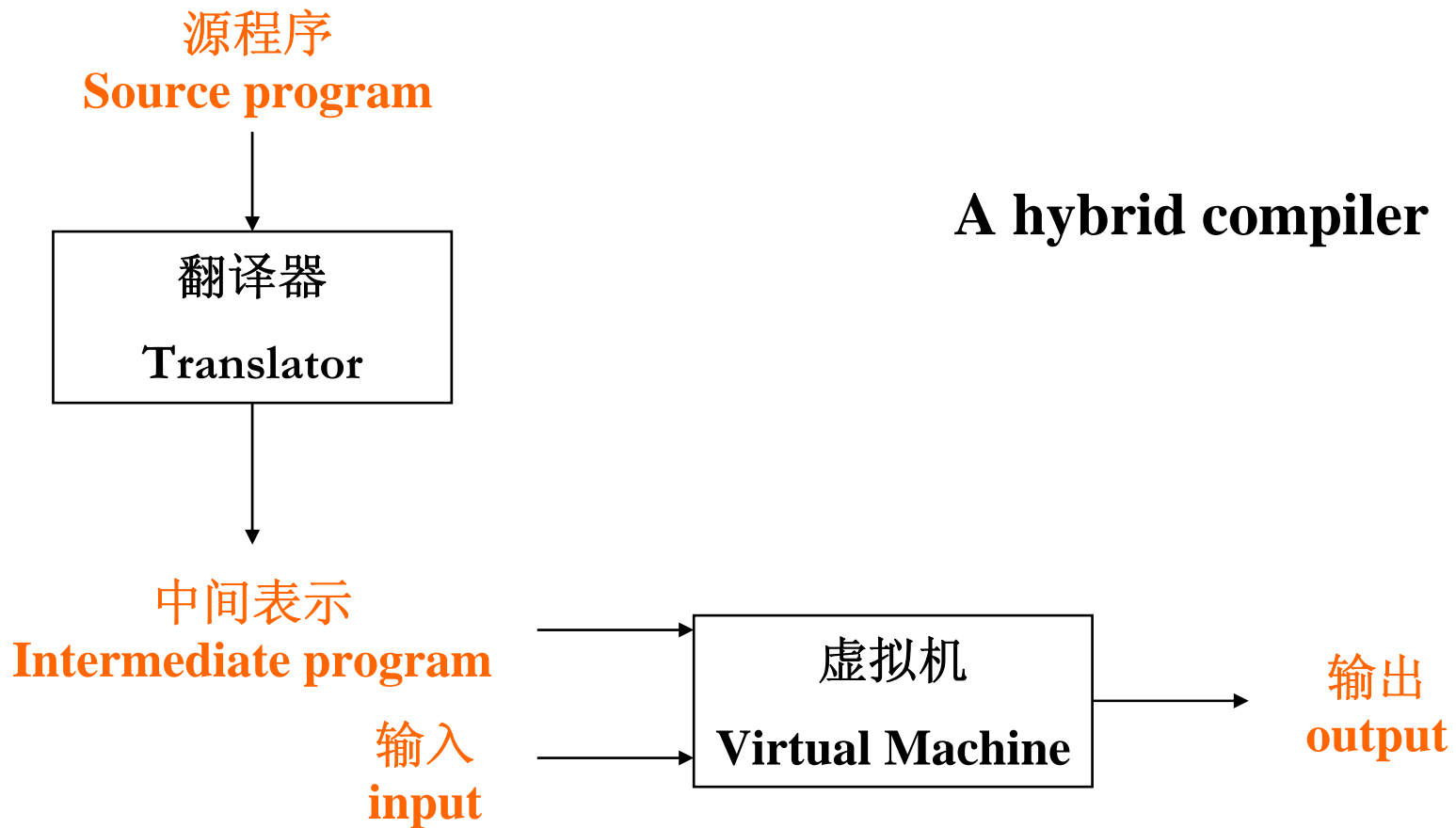
解释器不把源程序翻译成可执行的目标代码，其工作过程为：





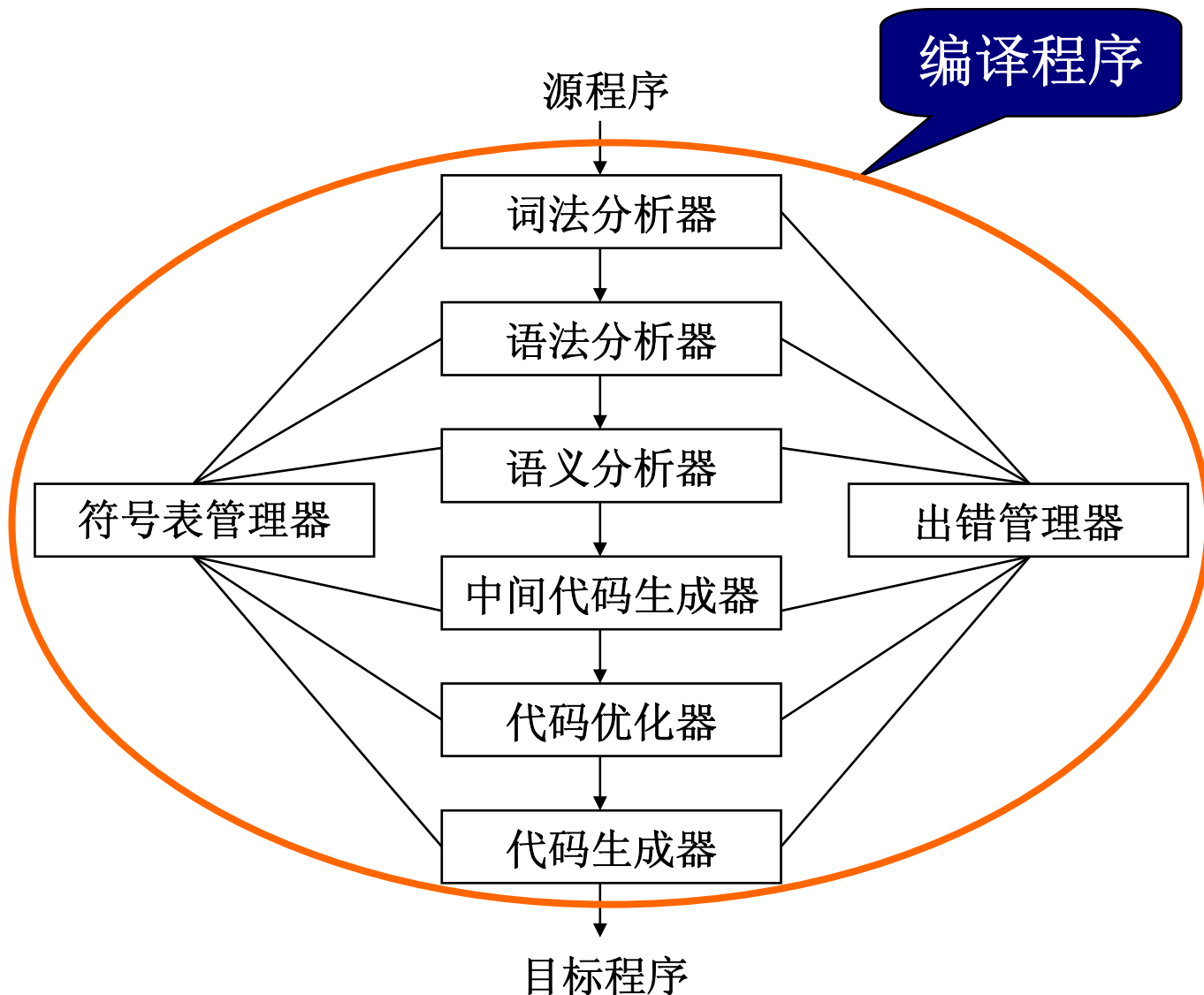
解释性程序的结构比编译程序简单，它的执行方式是一边翻译一边执行，因此其执行效率一般偏低，但是解释器的实现较为简单，而且编写源程序的高级语言可以使用更加灵活和富于表现力的语法。（[wikimedia](#) 维基百科）

目前语言的发展趋势是，解释性程序与编译程序相结合，两者之间区分的界线模糊。



编译程序是

把用高级程序设计语言编写的源程序翻译成为等价的目标程序的软件。



词法分析 (Lexical analysis / linear analysis / scanning)

position = initial + rate * 60

词法记号 (token) 流

标识符 **position**

赋值符 **=**

标识符 **initial**

加号 **+**

标识符 **rate**

乘号 *****

数字 **60**

position = initial + rate * 60



词法分析器



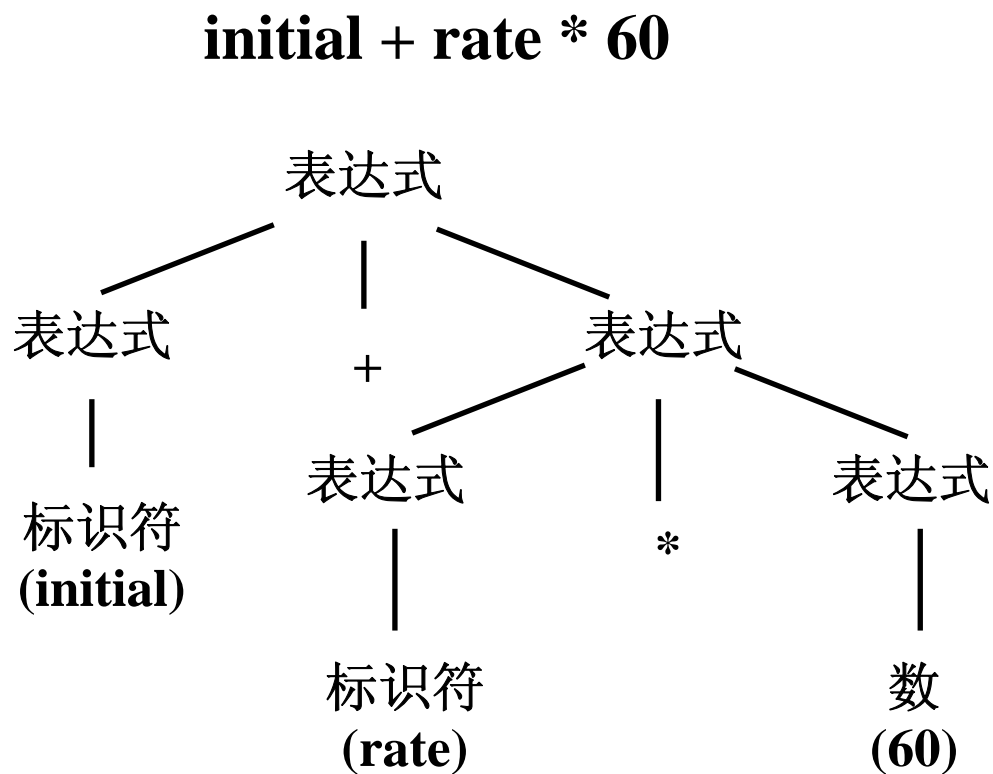
id₁ = id₂ + id₃ * 60

符号表

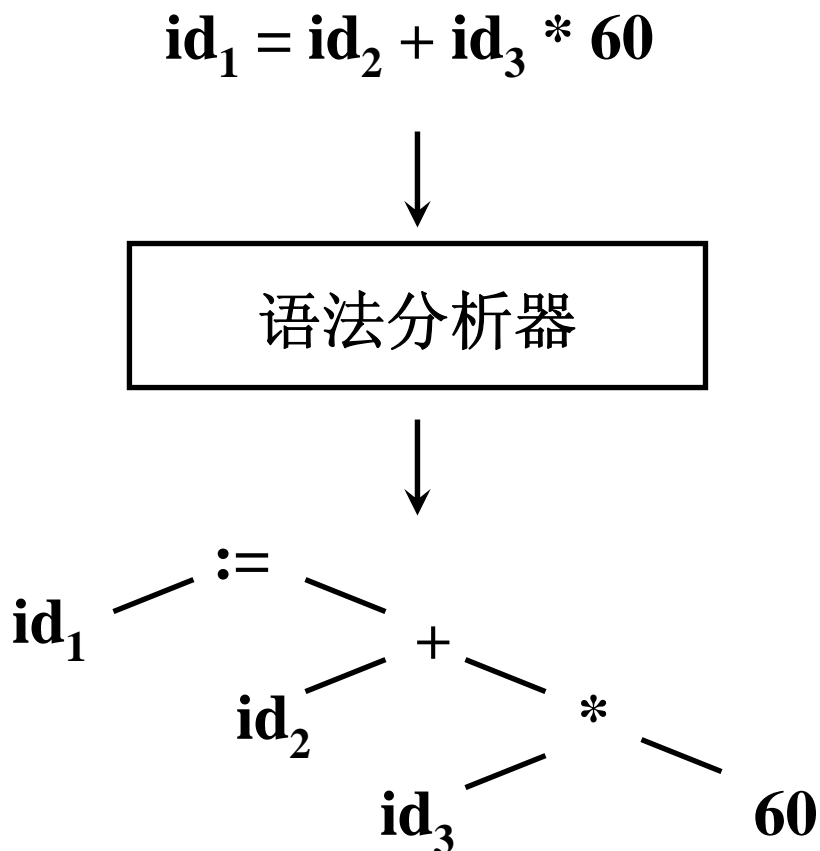
1	position	...
2	initial	...
3	rate	...

语法分析 (parsing / syntax analysis, hierarchical analysis)

- 任何一个标识符都是表达式;
- 任何一个数都是表达式;
- 如果 e_1 和 e_2 都是表达式, 那么
 - ◆ $e_1 + e_2$
 - ◆ $e_1 * e_2$
 - ◆ (e_1)也都是表达式



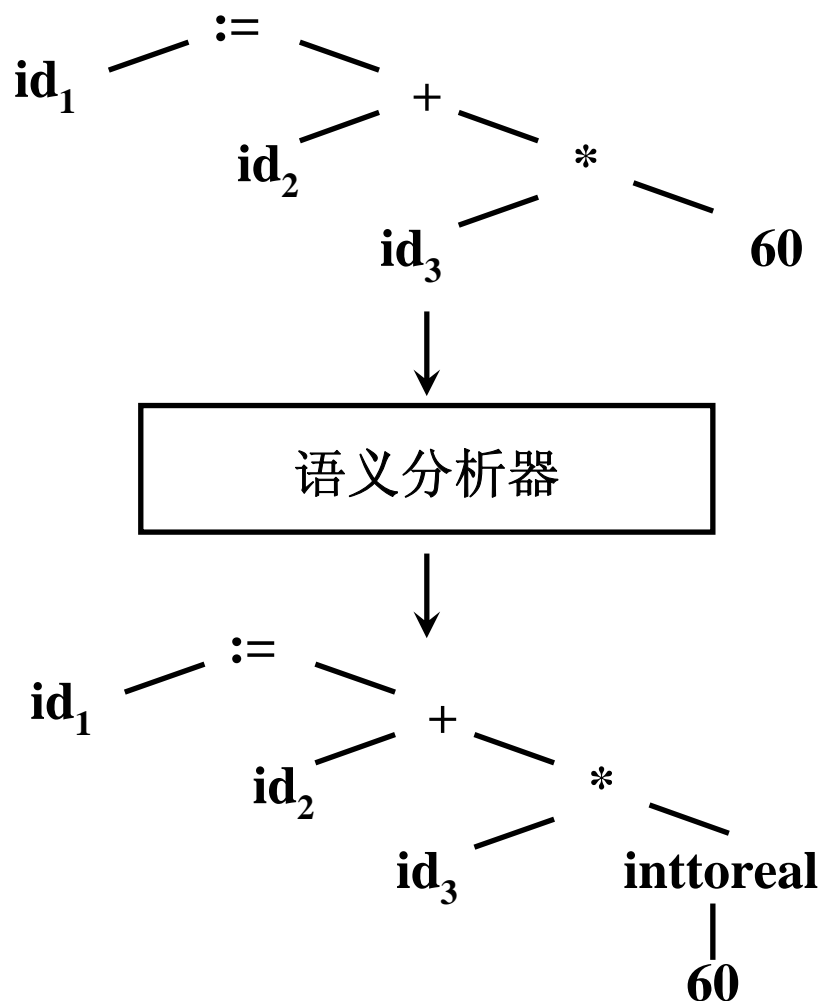
语法分析 (parsing / syntax analysis, hierarchical analysis)



符号表

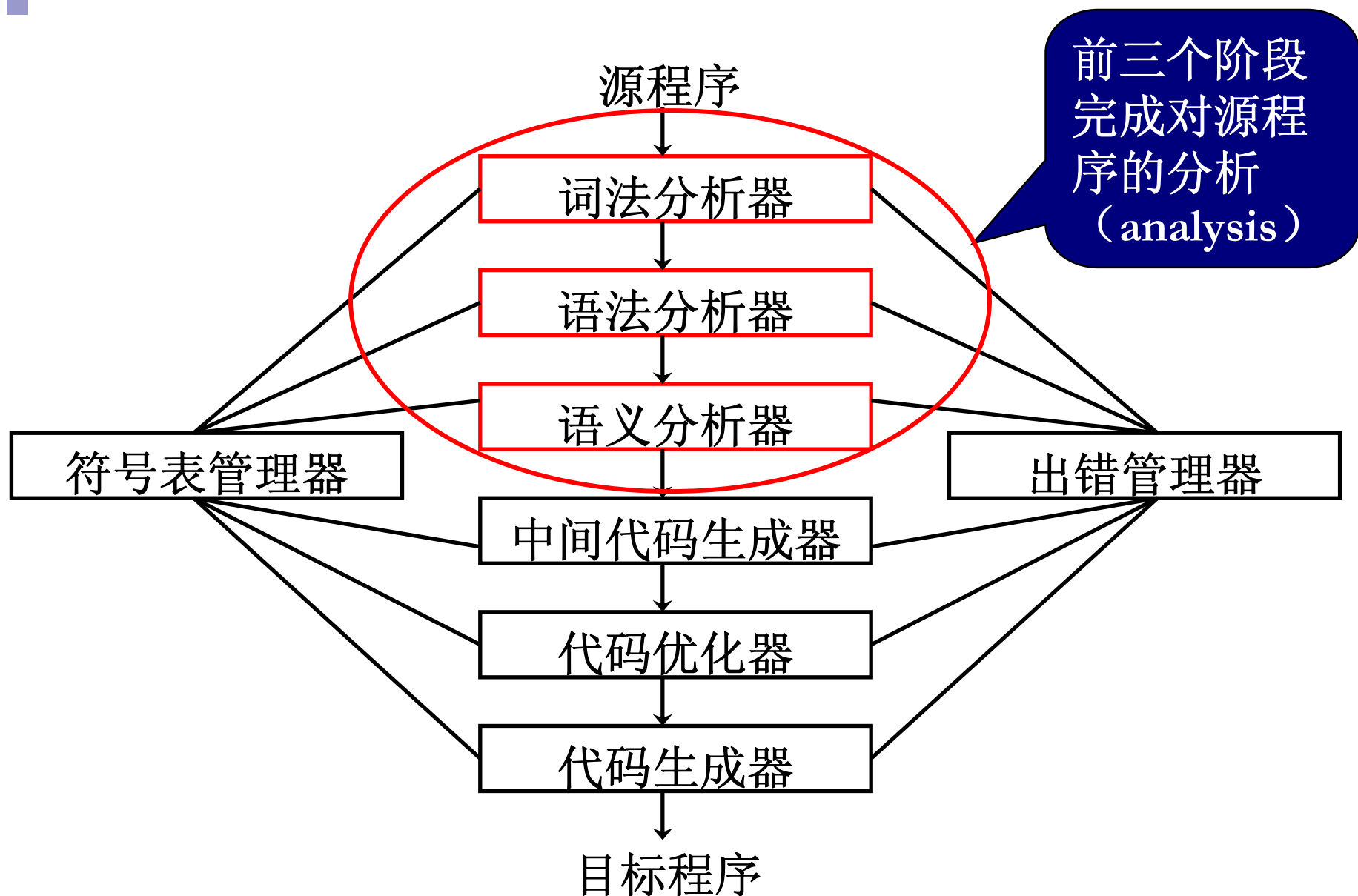
1	position	...
2	initial	...
3	rate	...

语义分析 (semantic analysis)

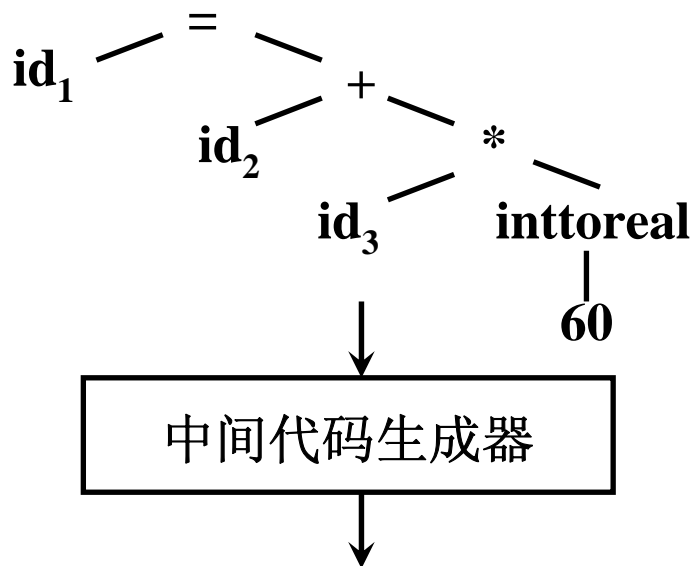


符号表

1	position	...
2	initial	...
3	rate	...



中间代码生成 (intermediate code generator)



```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

符号表

1	position	...
2	initial	...
3	rate	...

中间代码要求:

1. 易于产生;
2. 易于编译成目标程序。

代码优化 (code optimizer)

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```



```
temp1 = id3 * 60.0
id1 = id2 + temp1
```

符号表

1	position	...
2	initial	...
3	rate	...

衡量目标程序质量
高低主要标准:

1. 空间指标;
2. 时间指标。

代码生成 (target code generator)

temp1 = id3 * 60.0

id1 = id2 + temp1



代码生成器



MOVF id3, R2

MULF #60.0, R2

MOVF id2, R1

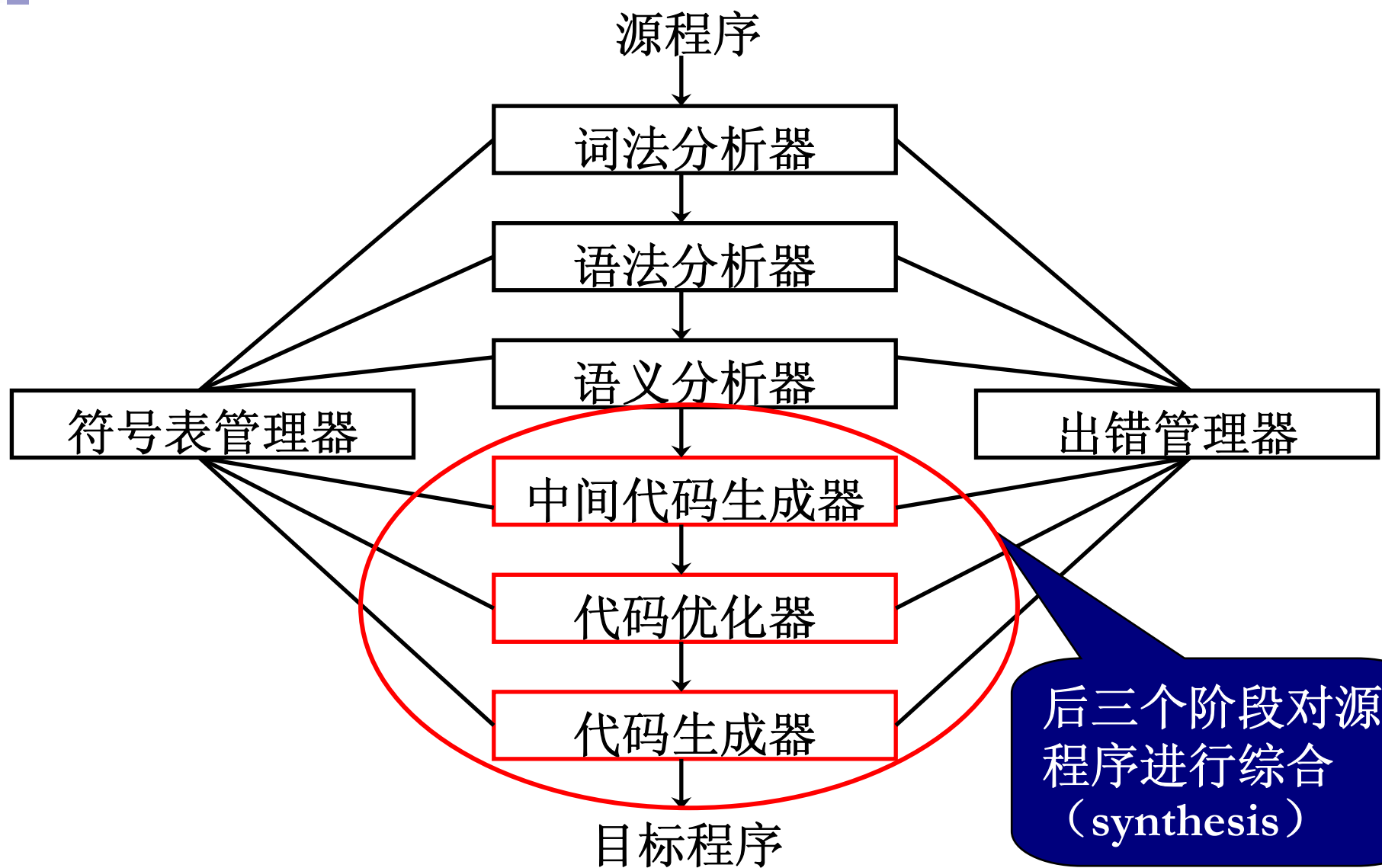
ADDF R2, R1

MOVF R1, id1

符号表

1	position	...
2	initial	...
3	rate	...

目标代码生成总的要求是所生成的目标代码有较高的执行效率。



symbol table management

符号表是为每个标识符保存一个记录的数据结构，记录的域是标识符的属性。

符号表管理器

源程序

词法分析器

语法分析器

语义分析器

中间代码生成器

代码优化器

代码生成器

目标程序


出错管理器

error detection
and reporting
编译的每个阶段
都会发现源程序
中的错误。

编译的多个阶段可以分为**前端**和**后端**两个大的阶段。

前端（front end）包括依赖于源语言并在很大程度上独立于目标机器的某些阶段或者某些阶段的某些部分。前端一般包括词法分析、语法分析、符号表的建立、语义分析、中间代码生成以及相关的错误处理。相当一部分代码优化工作也在前端完成。

后端（back end）包括编译器中依赖于目标机器的阶段或某些阶段的某些部分。一般来说，后端完成的任务不依赖于源语言而只依赖于中间语言。后端主要包括代码优化、代码生成以及相关的错误处理和符号表操作。



编译的若干个阶段通常是以一遍（pass）来实现的，每遍读一次输入文件，产生一个输出文件。

一个编译过程可由一遍、两遍或多遍完成。所谓遍，是对源程序或其等价的中间语言从头到尾扫描并完成规定任务的过程。每一遍扫描可完成上述一个或多个阶段的工作。



Thanks!