并行编译与优化
Parallel Compiler & Optimization

计算机研究所编译系统室

Lecture one: Introduction

第一课：引言

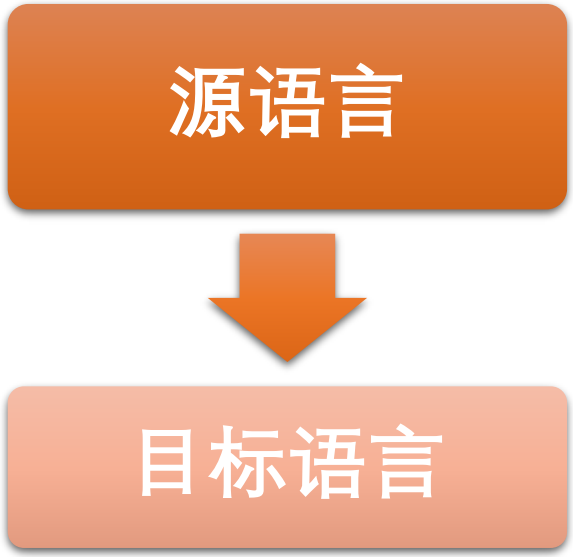
2024-02-27

内容

1、编译概念
2、编译过程
3、编译器结构
4、并行编译器
5、为什么学习编译课程?
6、课程安排

1.1 什么是编译（Compile）

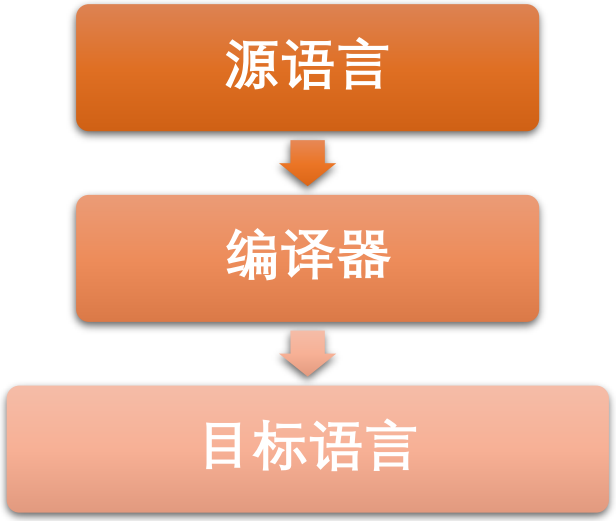
- 将一种语言（源语言）编写的程序，翻译成等价的、用另一种语言（目标语言）编写的程序

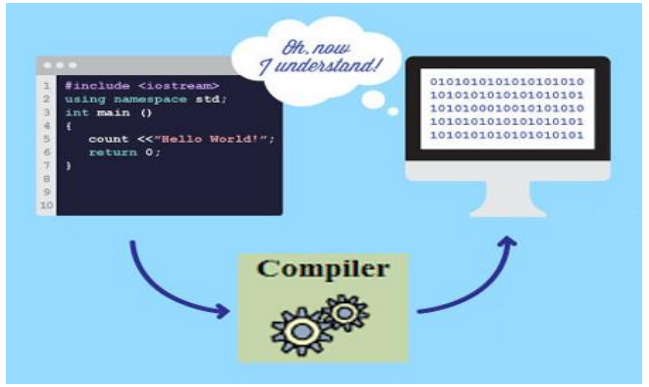


- 编译（Compile）
  - 翻译成另一种语言
  - 源语言与目标语言等价

1.2 什么是编译器（Compiler）

- 编译程序，将一种语言（源语言）编写的程序，翻译成等价的、用另一种语言（目标语言）编写的程序的计算机软件





1.2 什么是编译器（Compiler）

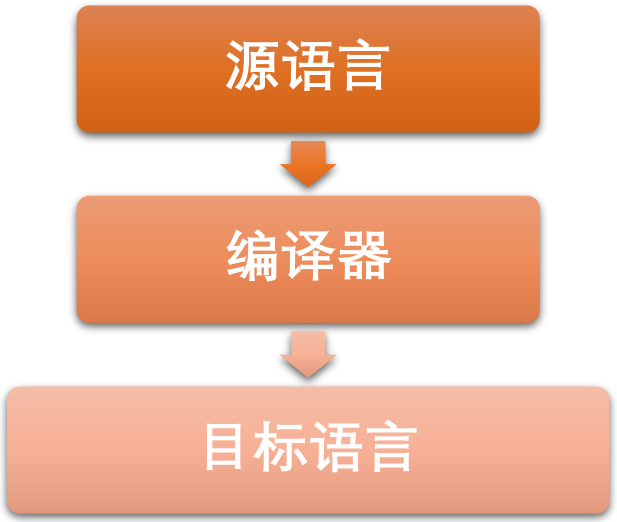
- 将一种语言（源语言）编写的程序，翻译成等价的、用另一种语言（目标语言）编写的程序
- A-0 Compiler
  - first compiler

"If you ask me what accomplishment I'm most proud of, the answer would be all the young people I've trained over the years; that's more important than writing the first compiler." Hopper, 1991

Grace Murray Hopper

1.2 什么是编译器（Compiler）

- 编译程序，将一种语言（源语言）编写的程序，翻译成等价的、用另一种语言（目标语言）编写的程序的计算机软件



IBM 704， 1954.05-1960.04

编译器最开始直接将高级语言编写的源程序翻译为可执行的目标代码。

1.2 什么是编译器（Compiler）

- 目标语言通常是目标机汇编语言

sin.c

```
int myfun(doule a)
{
  double result = sin(a);
  printf("result = %12.f\n", result);
   return 0;
}
```

#clang sin.c

1.2 什么是编译器

- 目标语言通常是目标机汇编语言

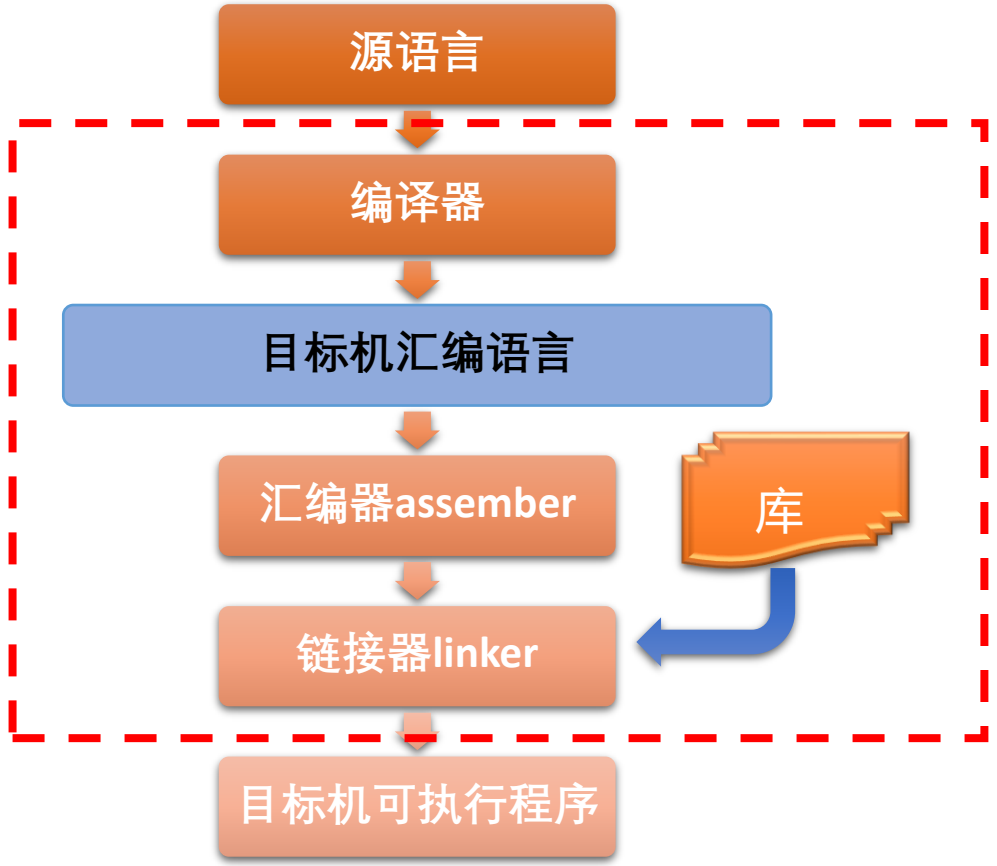
sin.c

```
#include <stdio.h>
#include <math.h>
int myfun(doule a)
{
  double result = sin(a);
  printf("result = %12.f\n", result);
   return 0;
}
```

#clang sin.c –lm

1.2 什么是编译器

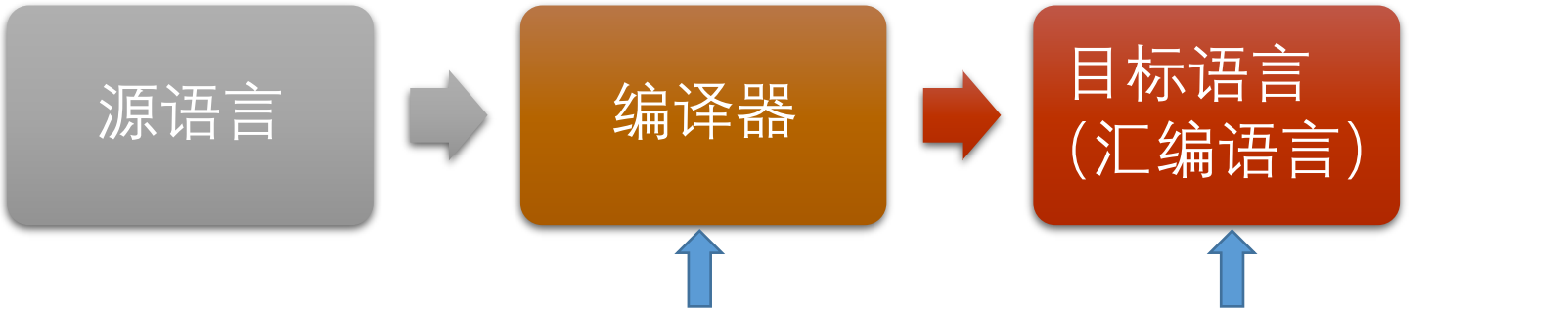
- 目标语言通常是目标机汇编语言



1.3 编译器分类

- 根据目标语言的不同

	目标语言
本地编译器（Native Compiler）	编译器运行平台的汇编语言
交叉编译器（Cross-Compiler）	另一种平台的汇编语言
源到源编译器（Source-to-Source Compiler）	另一种高级语言



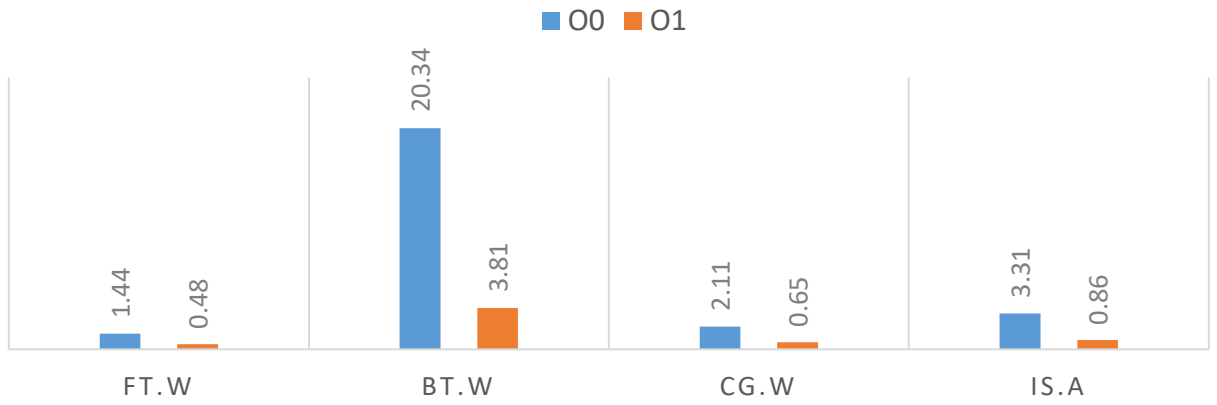
1.3 编译器分类

- 根据源语言的不同

C语言编译器	icc，gcc，clang
C++语言编译器	icpc，g++，clang++
Fortran语言编译器	ifort，gfotran，flang

1.4 编译器功能

- 将源语言程序翻译为机器的汇编语言或者其他语言，使用户可以通过该源语言编写程序
- 在编译过程中，帮助用户发现、定位程序中的错误，提高程序的产出率和可靠性
- 通过代码转换，优化产生的目标语言程序质量



GCC（10.2）FT2000+处理器
NPB3.3.1

1.5 优化编译器

- 优化编译器 = 翻译 + 优化
- 优化（Optimization ）
  - 通过分析编译过程中的中间表示，对其进行转换，以便生成更好的目标代码
- 优化目标— “更好的” 目标代码
  - 执行时间更短
  - 目标代码更小
  - 执行能耗更低
  - 浮点精度更高
  - ……

1.5 优化编译器

- 优化编译器 = 翻译 + 优化
- 优化必须
  - 保持程序语义（安全的）
  - 有价值的
    - 普遍适用的
    - 执行代价可接受的
- 优化 = 分析 + 转换

1.5 优化编译器

- 关键的分析
  - 控制流分析（Control-flow analysis）
  - 数据流分析（Data-flow analysis）
  - 别名分析（Alias analysis）
  - 依赖关系分析（Dependence analysis）
  - ……
- 优化 = 分析 + 转换

1.5 优化编译器

- 优化：基于分析的结果进行代码转换

```
for(i = 0; i< a.length - foo; i++)
    sum += a[i];
```

到达-定值分析 ➔ 循环不变量外提优化

```
t = a.length - foo;
for(i = 0; i< t; i++)
    sum += a[i];
```

1.6 编译器和解释器

- 解释器（Interpreter）

  - 按照源程序中指令或语句的动态执行顺序，逐条翻译，并立即解释执行相应功能的软件系统

- 编译和解释是程序语言实现的两种主要途径

1.6 编译器和解释器

编译 vs 解释

解释	编译
Online： 程序翻译和执行交错进行	Offline： 将源程序翻译成语义等价的目标代码
通常语句会反复翻译，例如循环等	以函数/文件为单位
代表语言：Perl，Shell，Pdf等	代表语言：Fortran，C/C++等

解释

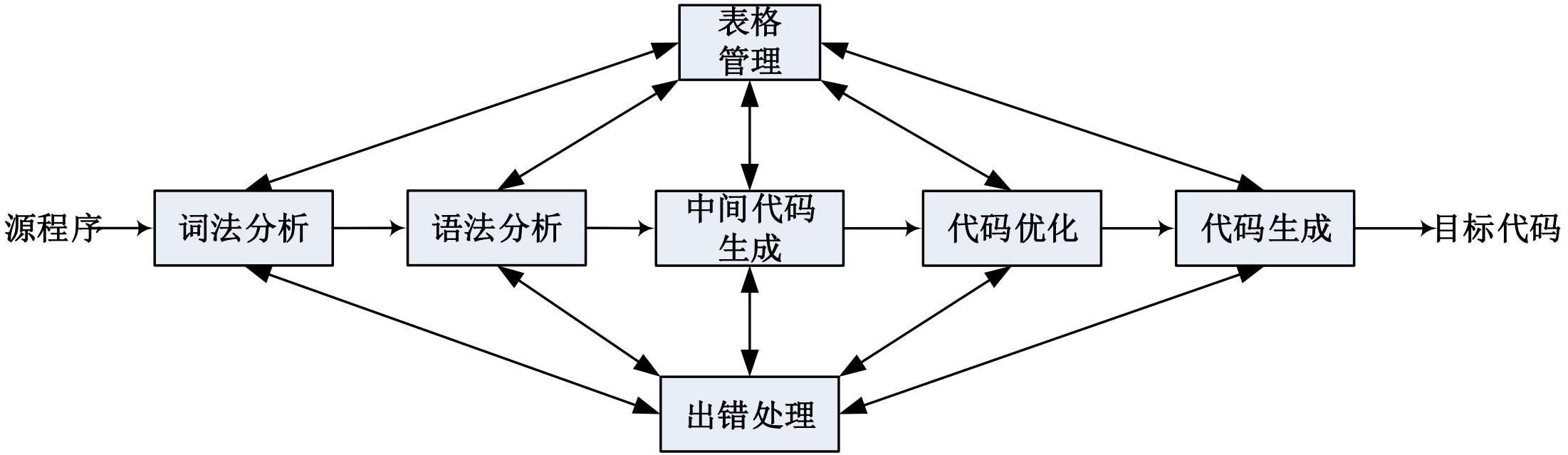
- 优点：实现算法简单，易于在解释过程中灵活、方便地插入所需的修改和调试措施
- 缺点：运行效率低

1.7 编译器的作用

- 应用程序、编程语言和体系结构之间的桥梁
- 帮助程序员发现和改正错误，提高程序可靠性和生产率
- 编译优化能够极大地提高程序性能

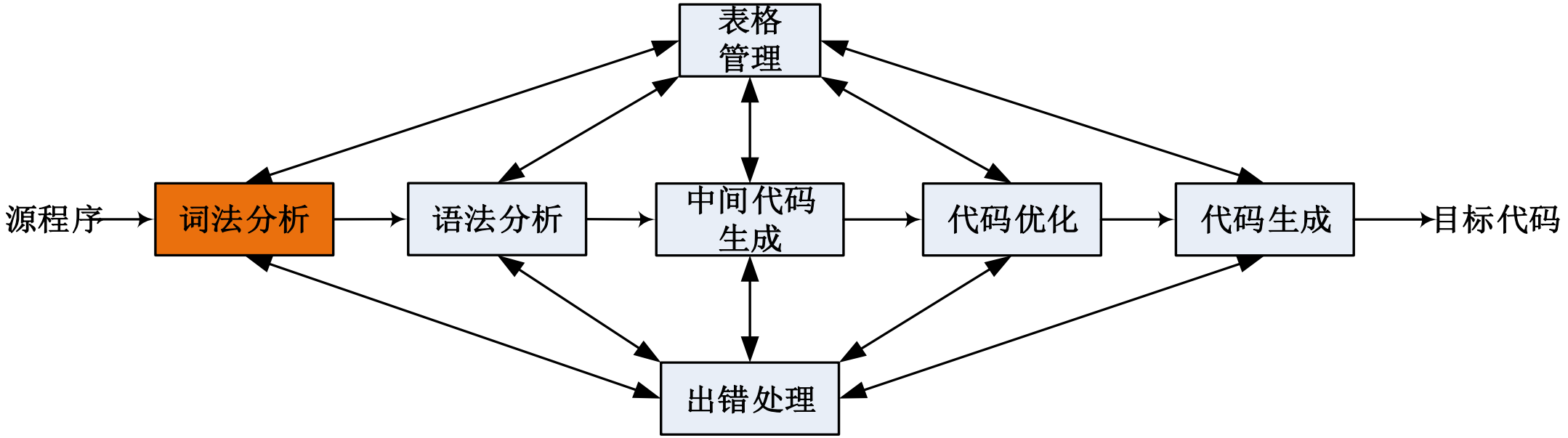
- 评估计算机体系结构的重要工具
- 支撑其他程序开发和调试工具
  - 性能分析工具、调试器、错误检测工具……

编译过程


表格
管理
源程序
词法分析
语法分析
中间代码
生成
代码优化
代码生成
目标代码
出错处理

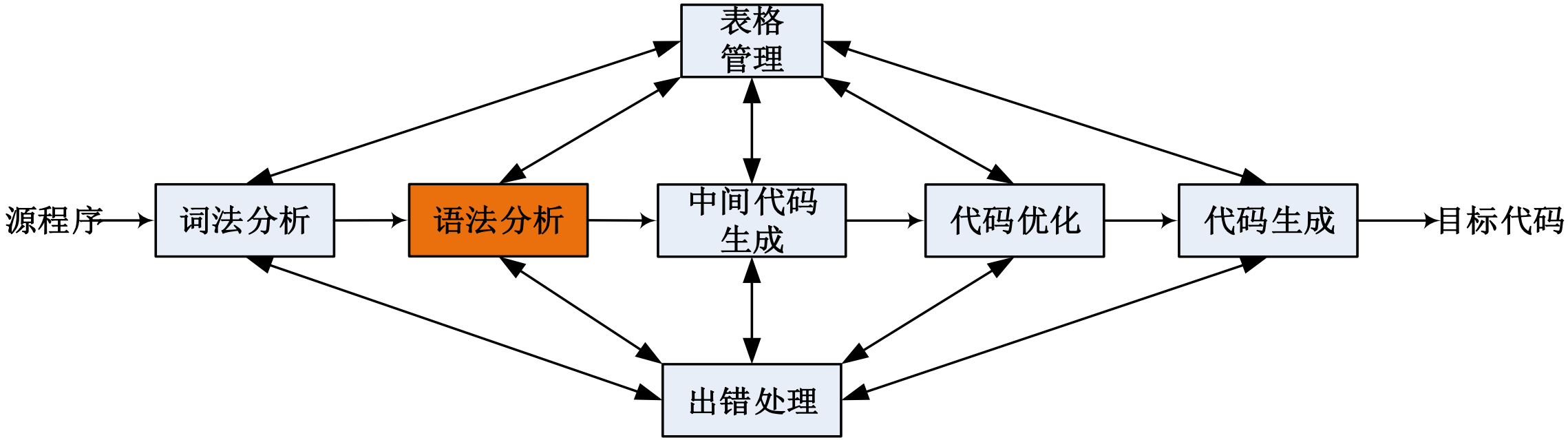

词法分析
语法分析
中间代码
生成
代码优化
代码生成

2.1 编译过程



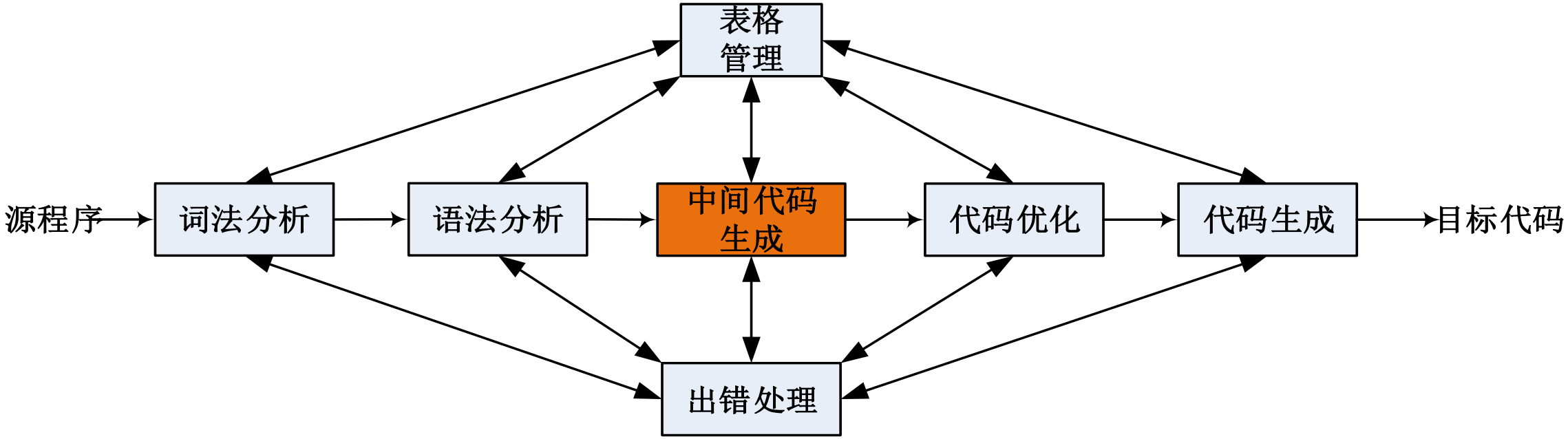
- 词法分析
  - 对构成源程序的字符流进行扫描，将字符组成有意义的单词符号序列

2.1 编译过程



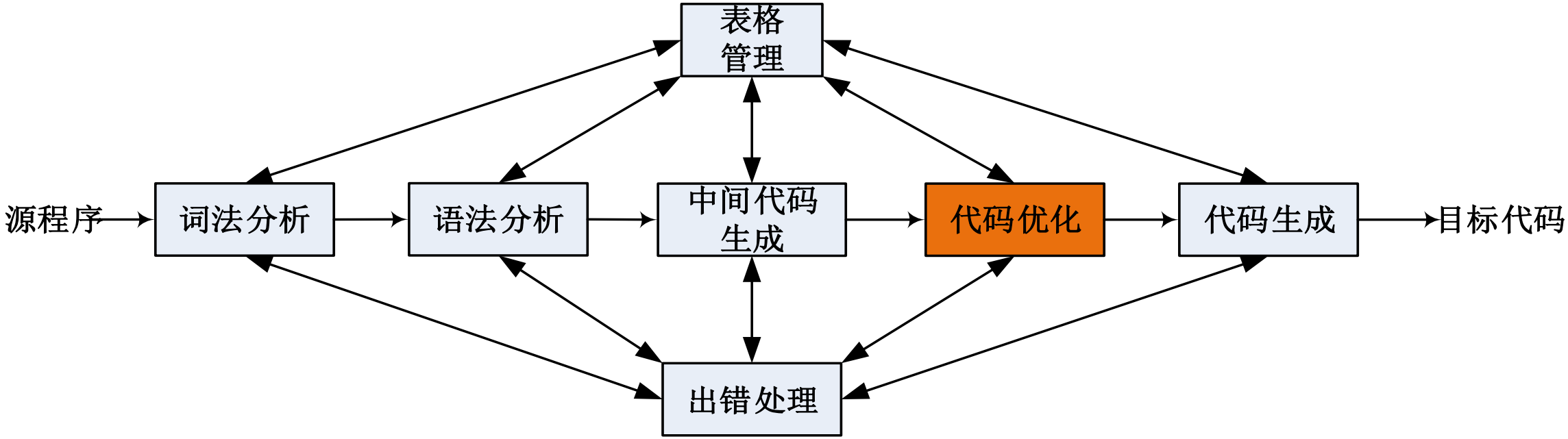
- 语法分析
  - 根据程序设计语言的语法规则，把单词符号串分解成各类语法单位，确定整个输入符号串是否符合语言的语法规范

2.1 编译过程



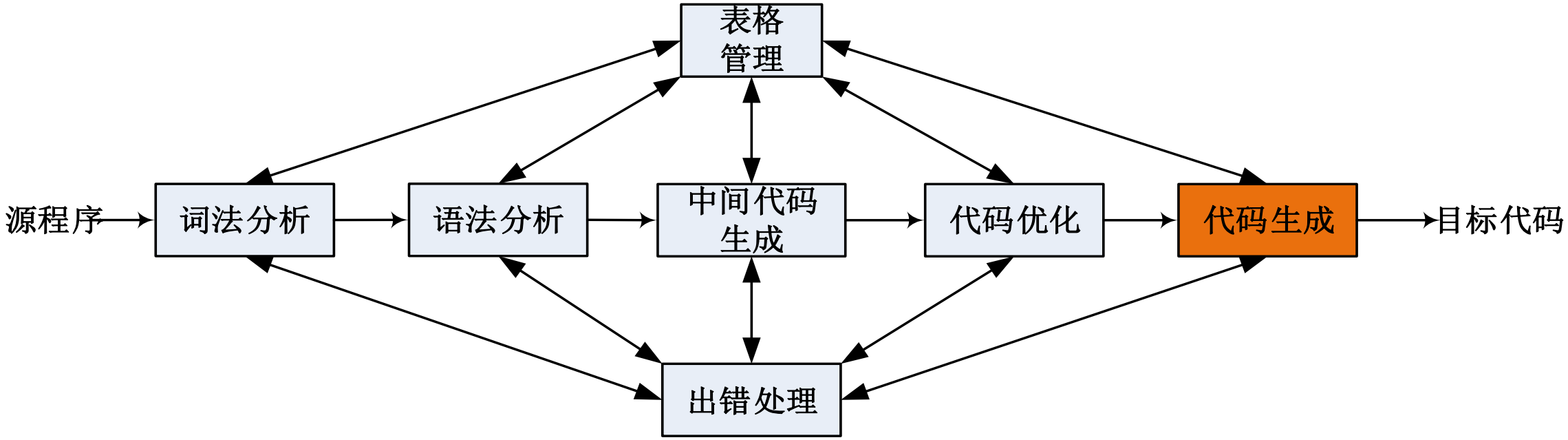
- 中间代码生成
  - 对语法分析所识别出的各类语法范畴，分析其含义，并产生中间代码

2.1 编译过程



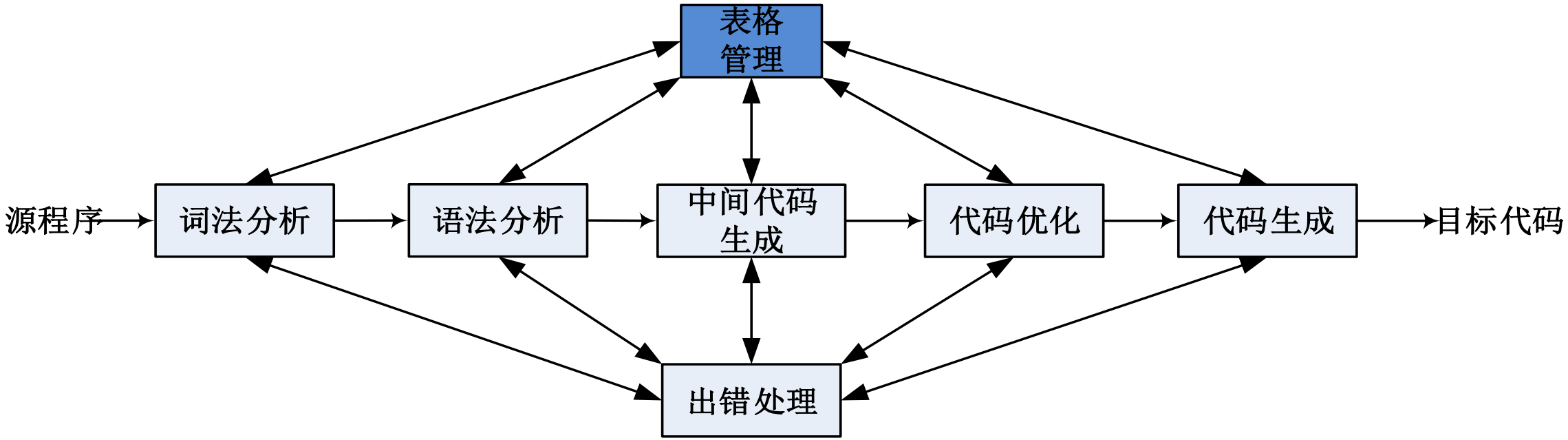
- 代码优化
  - 分析中间代码，对其进行转换，以便生成更好的目标代码

2.1 编译过程



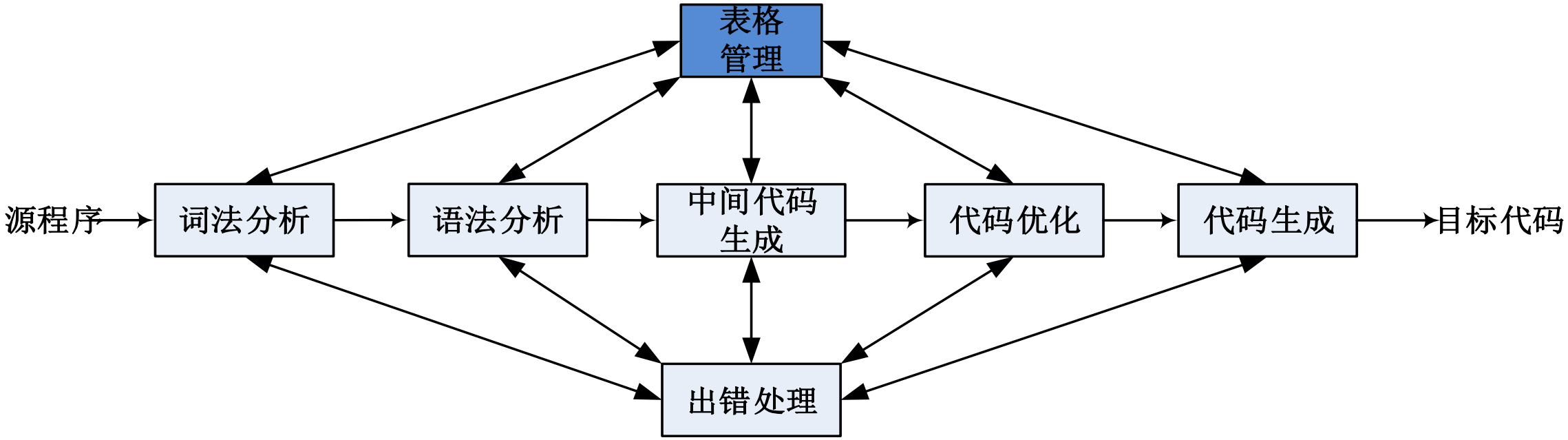
- 代码生成
  - 将中间代码翻译成目标程序

2.1 编译过程



- 表格管理
  - 维持一系列表格，记录源程序的各类信息，供编译的各个阶段使用
  - 符号表

2.1 编译遍



- 5个阶段仅仅是逻辑功能上的划分，在实现时，往往将编译器组织成若干**遍（Pass）**

2.2 编译遍

- 编译遍
  - 从头到尾扫描一遍源程序或中间代码程序并完成所规定的任务，生成新的中间代码或者目标程序
- 编译遍之间顺序执行，前一个遍的输出作为后一个遍的输入
- 为什么分遍？
  - 各遍功能独立单一，逻辑结构清晰
  - 不同的编译遍可以相对独立的开发
  - 通过遍管理器打开或关闭特定遍
  - 提高代码的可重用性，降低编译器开发的代价

2.2 编译遍

- 编译遍

  例 LLVM定义的遍

```
MODULE_ANALYSIS("callgraph", CallGraphAnalysis())
MODULE_ANALYSIS("lcg", LazyCallGraphAnalysis())
MODULE_ANALYSIS("module-summary", ModuleSummaryIndexAnalysis())
MODULE_ANALYSIS("no-op-module", NoOpModuleAnalysis())
MODULE_ANALYSIS("profile-summary", ProfileSummaryAnalysis())
MODULE_ANALYSIS("stack-safety", StackSafetyGlobalAnalysis())
MODULE_ANALYSIS("targetlibinfo", TargetLibraryAnalysis())
```

- 根据编译遍所完成的任务，可以将编译遍分成四类：分析遍、代码转换遍、优化遍及辅助遍

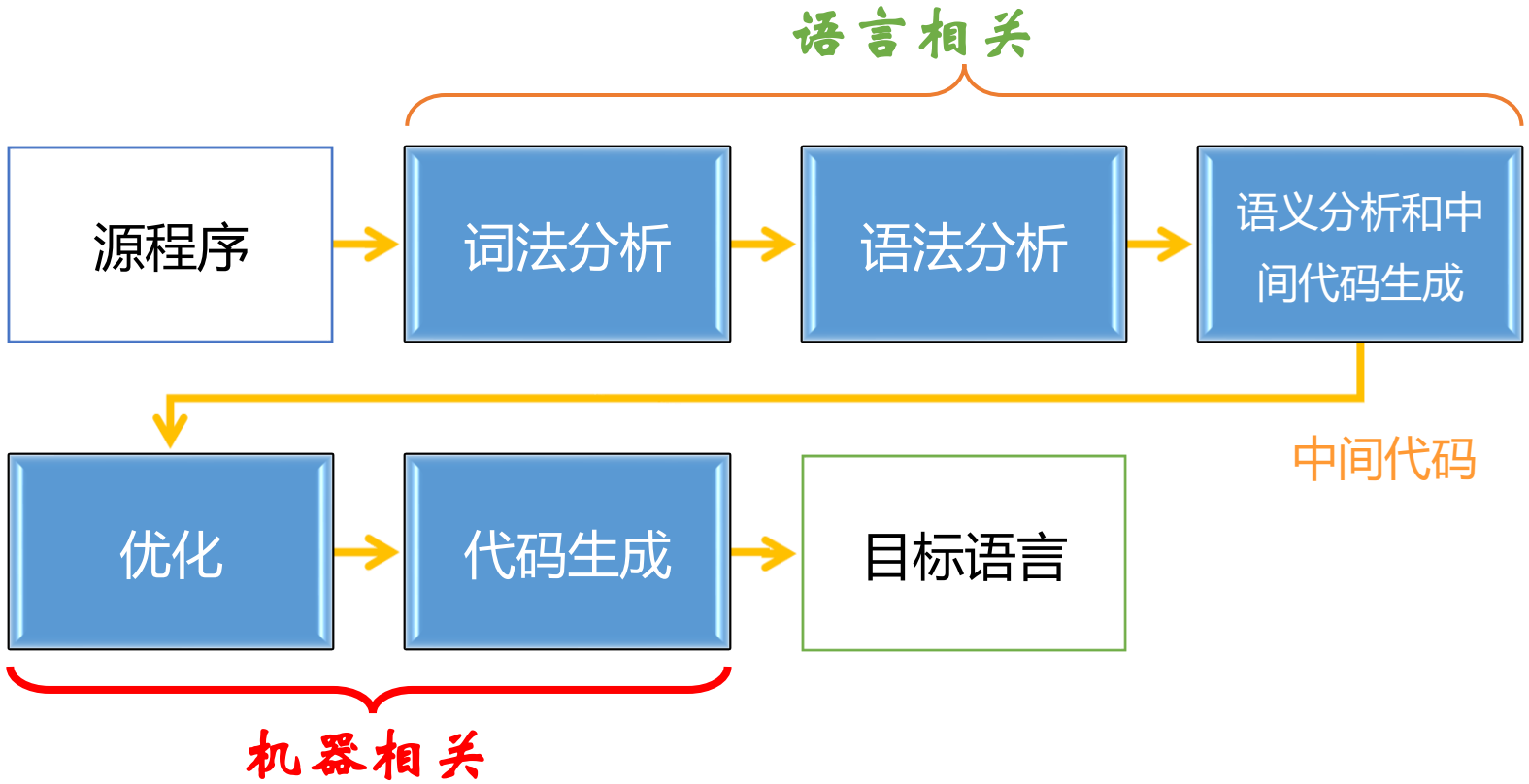
COMP CCRG Open MP

2.2 编译遍

- 分析遍
  - 收集信息，提供给其他遍使用，或用来调试、程序可视化等
  - 例如：控制流分析、数据流分析、构造必经节点树……
- 代码转换遍
  - 将代码从一种表示方式转换成另一种表示方式
  - 例如：源语言到编译器中间语言代码的转换……
- 优化遍
  - 对代码进行优化转换
  - 例如：常数传播、标量替换、循环展开……
- 辅助遍
  - 提供辅助功能，例如代码验证遍用于验证变换后的代码是否正确

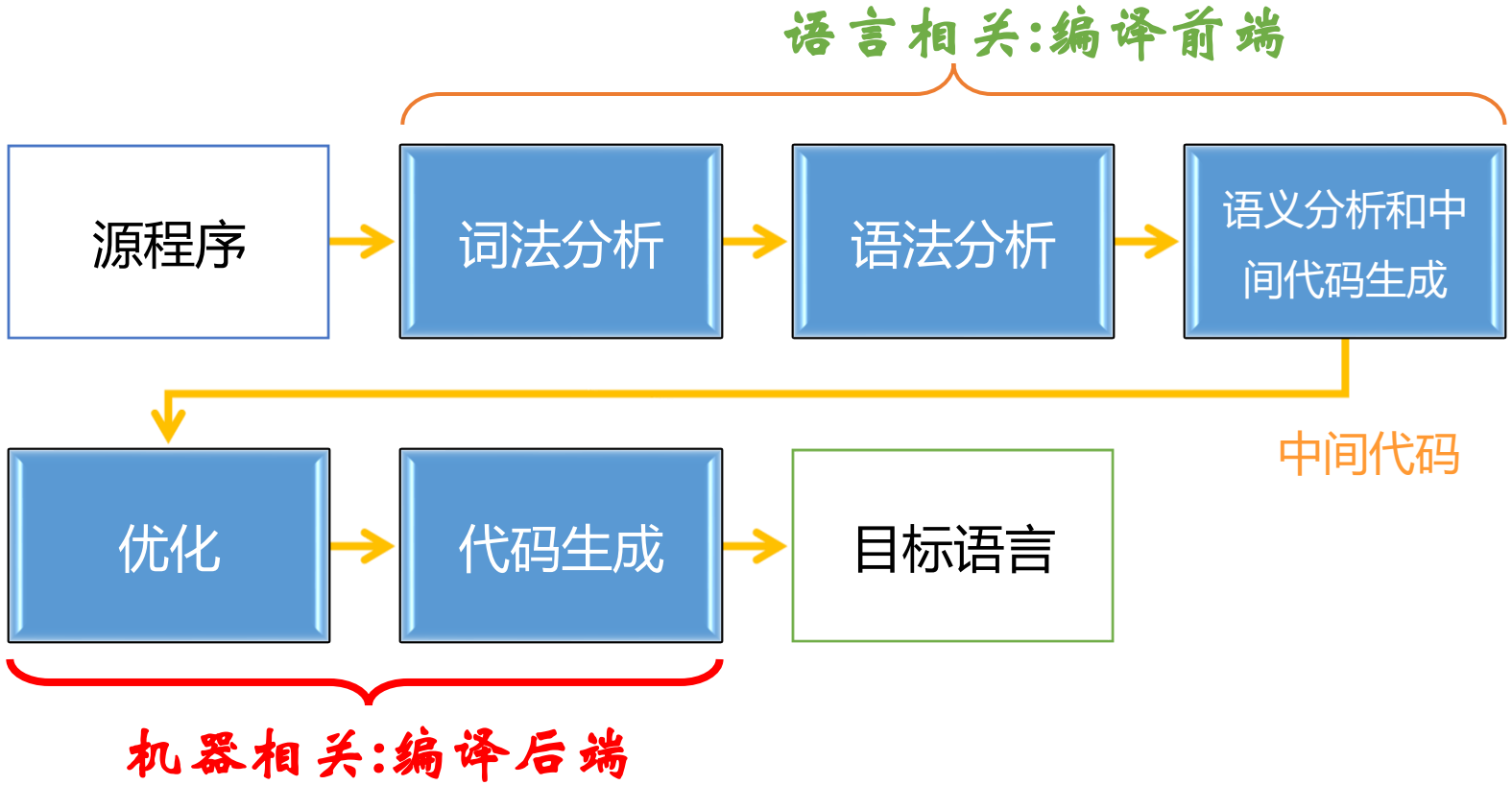
3 编译器结构

总体结构



3 编译器结构

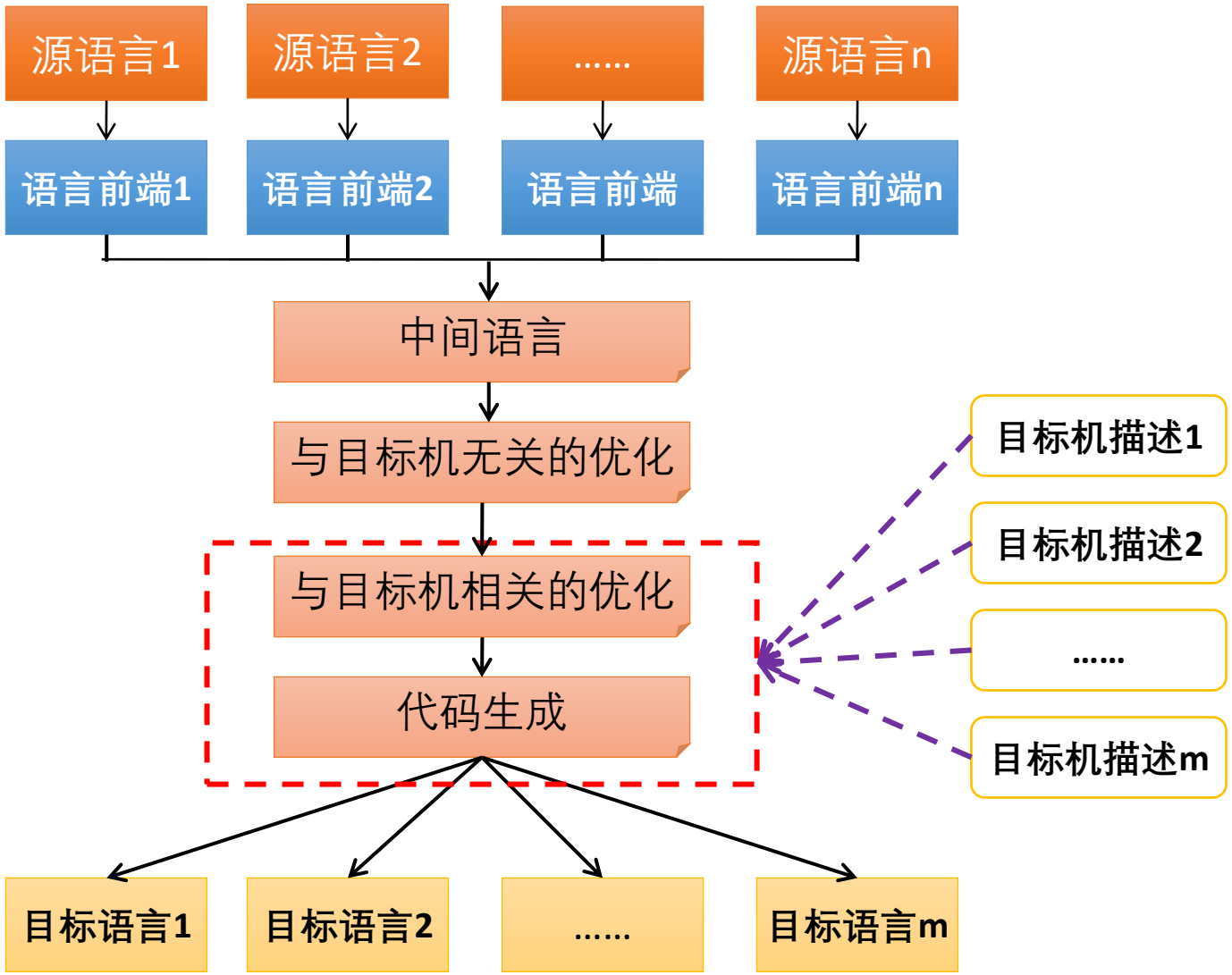
总体结构


语言相关:编译前端
源程序
词法分析
语法分析
语义分析和中间代码生成
中间代码
优化
代码生成
目标语言
机器相关:编译后端

3.1 多语言、多目标编译器

- 50%以上的代码在优化、代码生成阶段
- 这部分代码大多数针对中间表示进行，与目标机无关
- 如何用n + m个模块生成m×n个编译器?
  - n种语言
  - m个目标机

3.1 多语言、多目标编译器



3.2 程序表示

- 程序表示
  - 编译过程中，编译器内部使用的、能够表示源代码和源代码相关信息的数据结构或代码。
- 关键的程序表示
  - 中间代码
- 中间代码
  - 又称中间表示或中间语言
  - 中间表示是一种含义明确、便于处理的记号系统，通常独立于硬件

COMP CCRG Open MP

3.2 程序表示

- 常用的中间表示
  - 后缀式
    - 逆波兰表示法
    - 表达式的操作数写在前面、操作符写在后面（后缀），例如a+b写成ab+
  - 三地址代码
    - x:=y op z
    - 三地址代码的具体表示方法包括：三元式、间接三元式、四元式等
  - 图表示
    - 抽象语法树
    - 很多内部数据结构采用了图的形式

3.2 程序表示

- 为什么使用中间代码
  - 有利于编译器重定向（语言和新的平台），支持多种编程语言和多种后端平台的编译器可以通过使用一种中间表示来实现
  - 便于进行与平台无关的优化
  - 编译程序本身结构更清晰、更模块化

3.3 编译遍顺序

- 编译器通过遍管理器 (Pass Manager) 管理编译遍
- 编译器在初始化过程将各遍按一定顺序注册到遍管理器中，遍管理器根据遍提供的信息及注册顺序来进行调度执行
- 编译遍需要向遍管理器提供的信息包括**什么时候执行**、**如何执行**、**执行需要的条件**等。

```
OptimizePM.addPass(LoopDistributePass());
OptimizePM.addPass(LoopVectorizePass());
OptimizePM.addPass(LoopLoadEliminationPass());
```

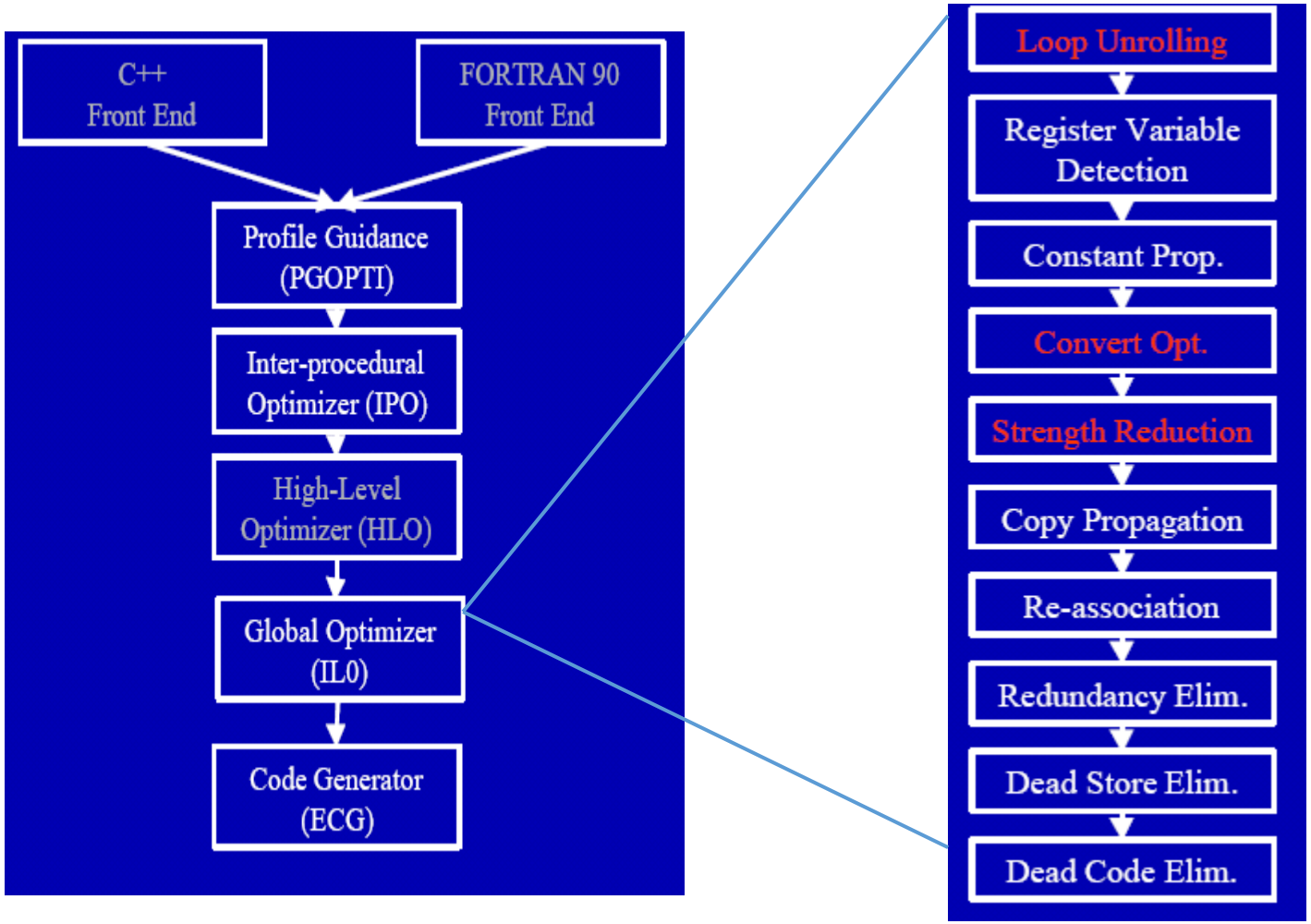
示例：LLVM 初始化遍顺序

3.3 编译遍顺序

- 如何组织编译遍顺序?
  - 遍之间具有简单的依赖关系
    - 一个优化为另一个优化创造机会
    - 例如：Copy传播和死代码删除
  - 遍之间具有循环依赖关系
    - 常数折叠和常数传播
  - 遍之间互相反作用
    - 公用子表达式删除和寄存器分配
    - 寄存器分配和指令调度

COMP CCRG Open MP

3.4 Intel编译器优化顺序



Intel Compiler

软件革命—并行

David Patterson，ACM&IEEE院士，ACM前任主席，提出RISC, RAID, NOW等

“在我看来，并行性是自从高级程序设计语言出现以来的最大挑战。在今后50年中它都将是最重要的课题，因为工业界已经把自己的未来押宝在‘并行程序设计是可行的’这一假定上。”

ACM Queue Interview with John Hennessy and David Patterson, 2007年1月

4.1 多层次并行

- 无处不在的并行
  - 超标量流水线 （指令级并行）
    - 通过多个功能部件，使处理器每个时钟周期可以处理多条指令的流水线设计
  - 单指令多数据（SIMD）结构（向量并行）
    - 向量执行是现代处理器提升计算能力的重要途径之一
    - 128位 → 256位→512位→1024位
  - 多核/众核处理器
    - 每个处理器核拥有独立的寄存器文件、流水线以及计算单元，通过线程级或者进程级并行执行来加速程序运行
- 现代处理器常将这几种并行方式配合起来，具有深流水线、超标量、SIMD和多个计算核心

4.1 多层次并行

- 并行计算机系统
  - 通过网络互连将多个处理器连接在一起，能够共同完成一个计算任务
    - 多路服务器
    - Cluster
- 超级计算机
  - 通过定制网络互连子系统将上万、甚至几十万个计算节点连接在一起，能够共同完成一个计算任务

4.2 并行编译

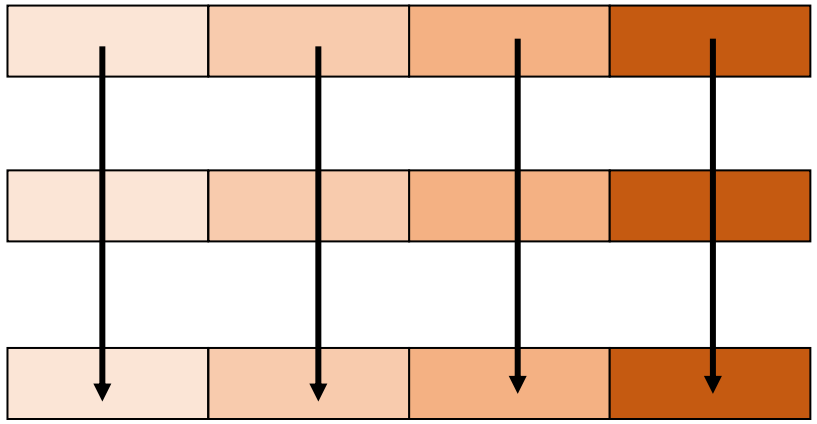
- 并行编译
  - 将源程序转换为能够并行执行的目标语言的过程
  - 源程序可以是串行程序，也可以是采用并行编程语言书写的并行程序
- 并行编译器
  - 实现并行编译的计算机软件
  - 通常基于串行编译器实现

4.2并行编译

- 指令级并行
  - 编译优化，例如指令调度、寄存器分配、循环优化等对提高指令级并行仍旧至关重要
  - 超长指令字（Very Long Instruction Word, VLIW）架构的处理器(例如DSP）依赖编译优化技术来挖掘指令中的并行进行指令填充打包

4.2并行编译

- 向量并行
  - 编译器自动向量化
  - 编译器提供intrinsic接口



```
for (j=0; j<N; j++)
    b[j] = scalar*c[j];
```

自动向量化

```
.L19    ld1d    z0.d, p0/z, [x4, x1, lsl 3]
        fmul    z0.d, z0.d, z1.d
        st1d    z0.d, p0, [x2, x1, lsl 3]
        incd    x1
        whilelo p0.d, w1, w0
        b.any   .L19
```

4.2并行编译

- 核级并行
  - 编译器自动并行化

```
for (i=0; i< N; i++)
    sum += c[i]*b[i-1];
```

  - 编译器实现并行编程接口
    - 指导命令OpenMP API

4.2并行编译

- 处理器间并行
  - 编译器实现并行编程接口
    - 基于指导命令的并行编程接口：OpenMP、OpenACC
    - 并行语言：CUDA、OpenCL……

```
#pragma parallel for
{
      for(i=0;i<N;i++)
        work();
}
```

基于FT2500的多路服务器

4.2并行编译

- 处理器间并行
  - 编译器实现并行编程接口
    - OpenMP、OpenACC
    - CUDA、OpenCL……

Cloud Hin GS226X
（Intel Xeon /TESLA V100 ）

```
cudaMallocManaged(data, N);
fread(data, 1, N, fp);
qsort<<<...>>>(data, N, 1,compare);
cudaDeviceSynchronize();
usedata(data);
free(data);
```

4.3 并行编译器实现

- 基于串行编译器扩充编译前端以支持新的语言结构，将这些语言编写的并行程序转换为对应的中间表示
- 并行语义处理部分进行转换，调用运行时库共同实现并行
- 并行语言的运行时库负责
  - 并行任务的管理、划分和调度
  - 并行执行体（进程/线程）的管理和调度

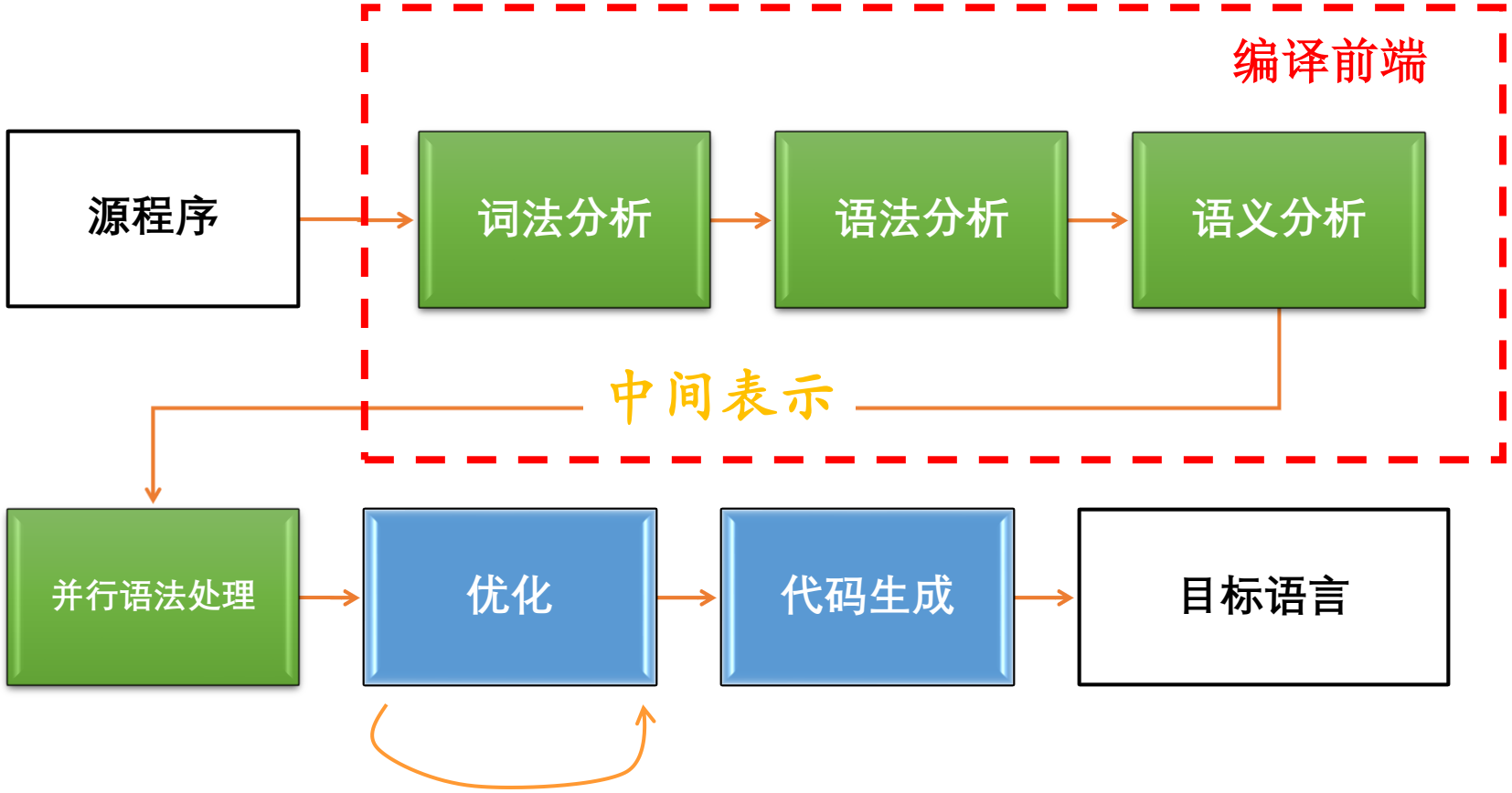
4.3 并行编译器实现

```
void work(){
#pragma omp parallel
    printf("I am thread %d\n", omp_get_thread_num());
}
```

```
Void work(){
   GOMP_parallel(work.omp, 0);
}
```

```
void work.omp(){
     printf("I am thread %d\n", omp_get_thread_num());
}
```

4.3 并行编译器实现

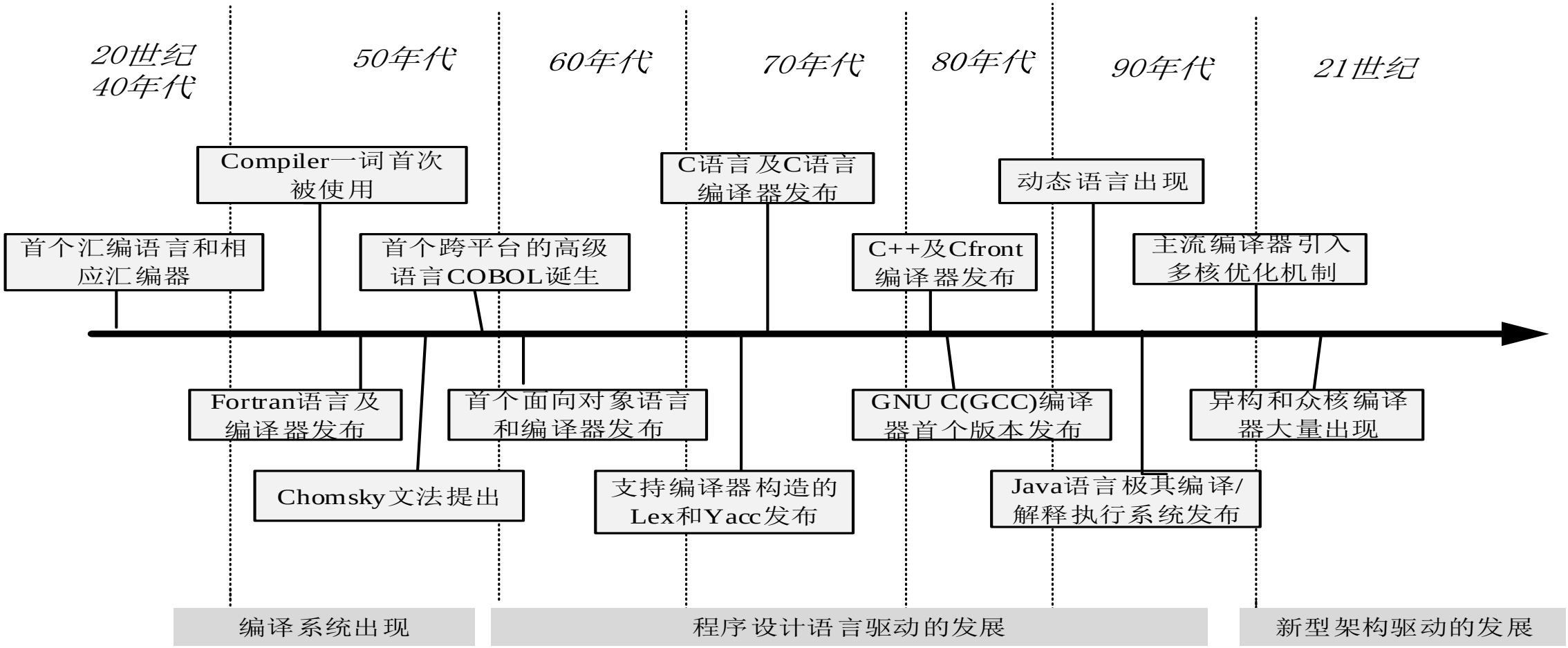


5 为什么学习本课程

- 作为基础软件，编译器是应用、语言、体系结构之间的桥梁
- 编译一直是计算机的研究热点
  - 应用在不断变化
  - 体系结构在不断变化
  - 语言在不断变化
  - 编译的目标在不断变化
  - 编译技术本身在不断变化

5 为什么学习本课程

编译系统的历史发展



5 为什么学习本课程

Compiler Research: The Next 50 years
Mary Hall, David Padua and Keshav Pingali, 2009

- 程序优化
  - 使并行程序设计成为主流
  - 针对体系结构的优化
- 正确性和安全性
  - 使软件开发像大飞机一样可靠
  - 使系统软件在各个层次上都安全
  - 软件栈的自动验证

5 为什么学习本课程

- 作为基础软件，编译器是应用、语言、体系结构之间的桥梁
- 编译是理论和工程实践完美融合的体现
- 成为更好的程序开发者
- 编译技术无处不在

5 为什么学习本课程

- 成为更好的程序开发者

哪一段程序执行更快？

```
!$omp parallel for private(t)
for (i=0; i< N; i++) {
   t +=index[i]
#pragma omp critical
    a[t] = …
}
```

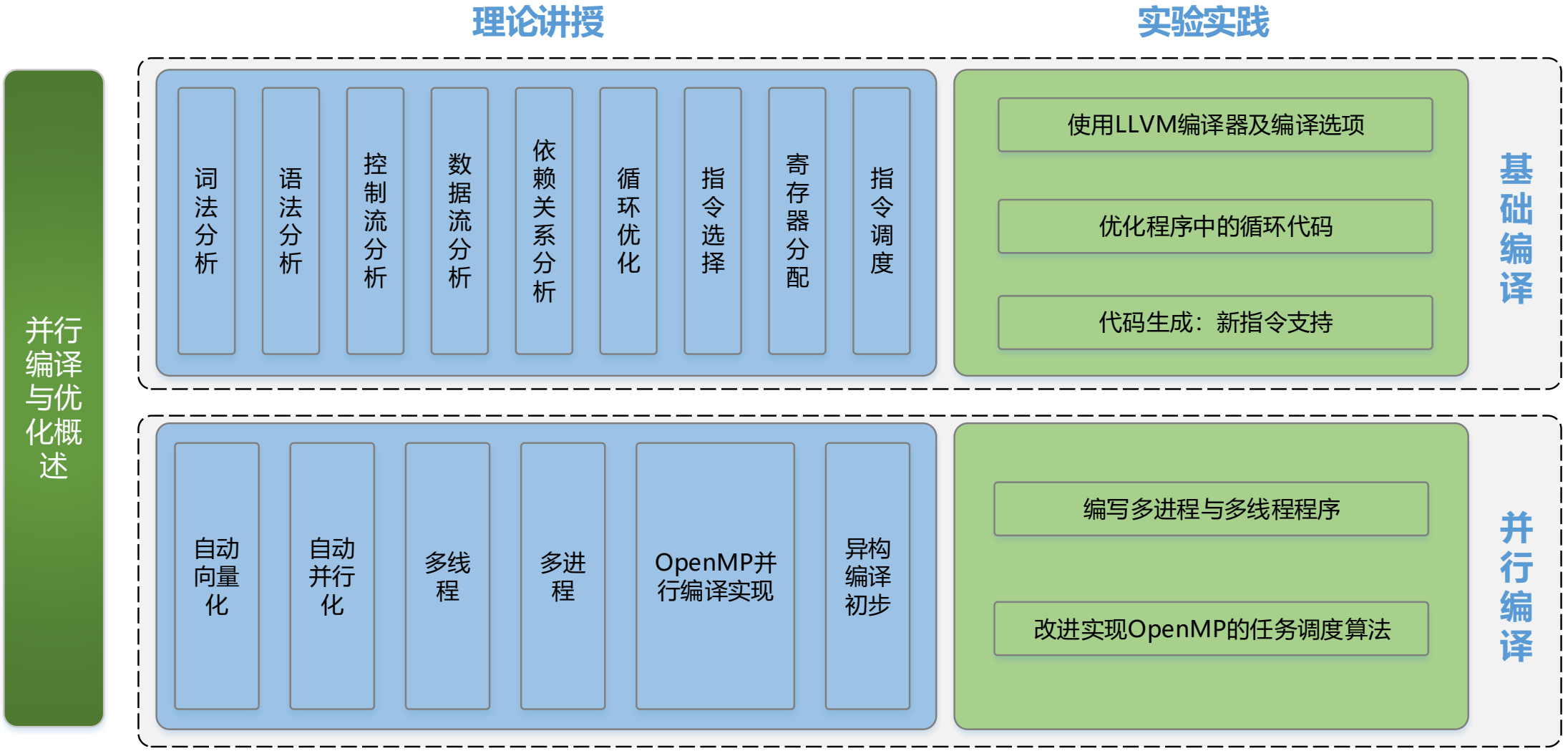
```
!$omp parallel for private(t)
for (i=0; i< N; i++) {
   t +=index[i]
#pragma omp atomic write
    a[t] = …
}
```

5 为什么学习本课程

- 作为基础软件，编译器是应用、语言、体系结构之间的桥梁
- 编译是理论和工程实践完美融合的体现
- 成为更好的程序开发者
- 编译技术无处不在

- 有一天你也可能会写一个编译器！

6.1 课程内容


理论讲授
实验实践
并行编译与优化概述
词法分析
语法分析
控制流分析
数据流分析
依赖关系分析
循环优化
指令选择
寄存器分配
指令调度
使用LLVM编译器及编译选项
优化程序中的循环代码
代码生成：新指令支持
基础编译
自动向量化
自动并行化
多线程
多进程
OpenMP并行编译实现
异构编译初步
编写多进程与多线程程序
改进实现OpenMP的任务调度算法
并行编译

6、课程安排

6.2 课程安排

COMP CCRG Open MP

6.3 课程计划

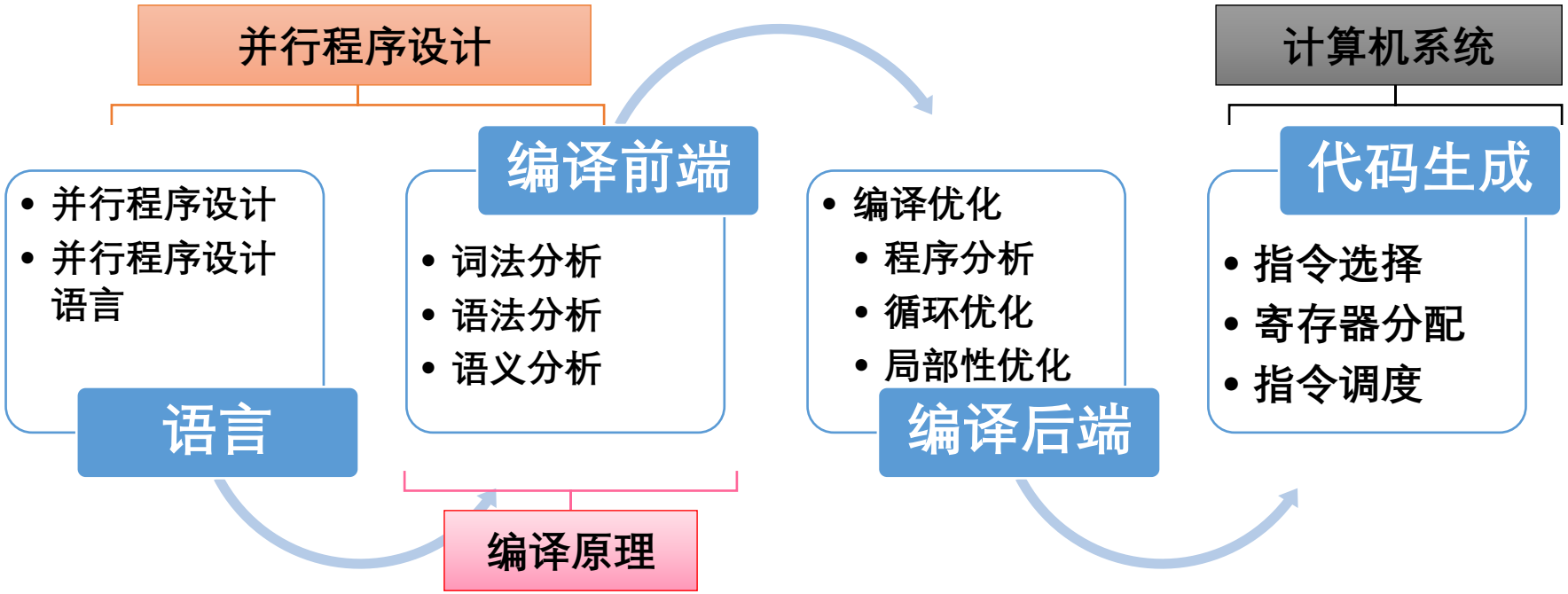
- 课时安排
  - 课堂讲授40学时
  - 实验24学时
- 课程考核
  - 实验40%
  - 考试40%
  - 其他20%
    - 作业、课堂表现等

2024春《并行编译与优化》教学日历

课次	周次	时间		节次	内容
1	第1周	02.27	周二	9-10	概述
2	第1周	02.29	周四	9-10	词法分析和语法分析
3	第2周	03.05	周二	9-10	实验1：Antlr实验
4	第2周	03.07	周四	9-10	语义分析与中间表示
5	第3周	03.12	周二	9-10	实验2：生成中间表示
6	第3周	03.14	周四	9-10	实验2：生成中间表示
7	第4周	03.19	周二	9-10	控制流分析
8	第4周	03.21	周四	9-10	指令选择
9	第5周	03.26	周二	9-10	实验3：代码生成
10	第5周	03.28	周四	9-10	实验3：代码生成
11	第6周	04.02	周二	9-10	数据流分析
12	第6周	04.07	周日	9-10	标量优化
13	第7周	04.09	周二	9-10	实验4：标量优化
14	第7周	04.11	周四	9-10	依赖关系分析
15	第8周	04.16	周二	9-10	依赖关系分析
16	第8周	04.18	周四	9-10	寄存器分配
17	第9周	04.23	周二	9-10	寄存器分配
18	第9周	04.25	周四	9-10	指令调度
19	第11周	05.07	周二	9-10	指令调度
20	第11周	05.09	周四	9-10	实验5：寄存器分配
21	第12周	05.14	周二	9-10	循环优化
22	第12周	05.16	周四	9-10	循环优化
23	第13周	05.21	周二	9-10	实验6：程序优化
24	第13周	05.23	周四	9-10	实验6：程序优化
25	第14周	05.28	周二	9-10	并行化
26	第14周	05.30	周四	9-10	向量化
27	第15周	06.04	周二	9-10	多进程与多线程
28	第15周	06.06	周四	9-10	实验7：程序并行
29	第16周	06.11	周二	9-10	实验7：程序并行
30	第16周	06.13	周四	9-10	实验7：程序并行
31	第17周	06.18	周二	9-10	OpenMP实现
32	第17周	06.20	周四	9-10	异构计算
					复习答疑

上课教室	黄春
302-501	方建滨
	沈洁

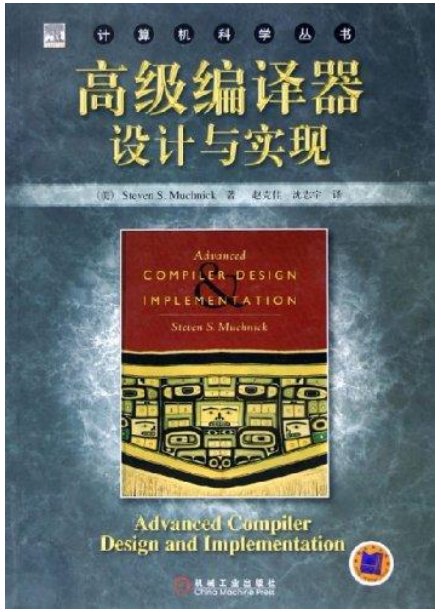
6.4 和其它课程的关系



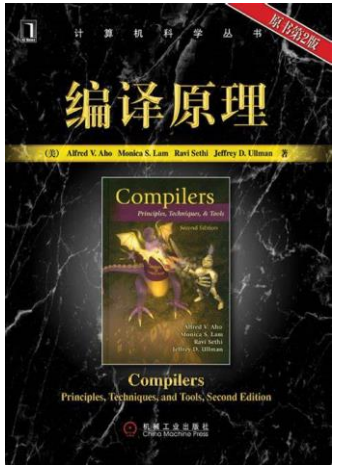
- 预修课程
  - 计算机系统
  - 数据结构
  - 并行程序设计

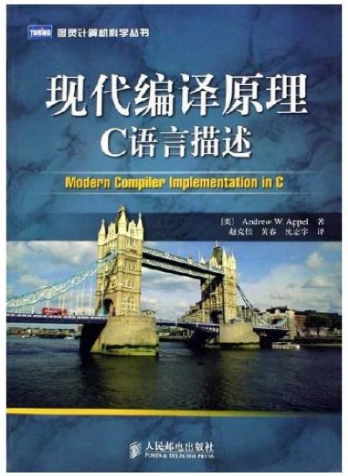
6.5 教材与参考资料

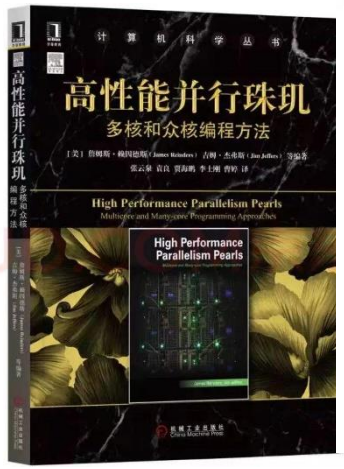
教材

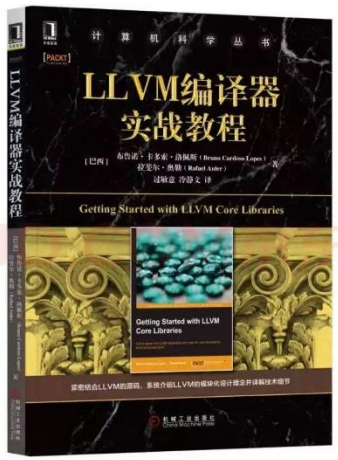


参考书









6.6 任课教员

- 黄春（chunhuang@nudt.edu.cn）
- 方建滨（j.fang@nudt.edu.cn）
- 沈洁（j.shen@nudt.edu.cn）
- 课程平台：头哥
- 答疑
  - 天河北楼428、520
  - 微信群、Email
  - 助教：卢晓波、刘锡泰

全国大学生计算机系统能力大赛

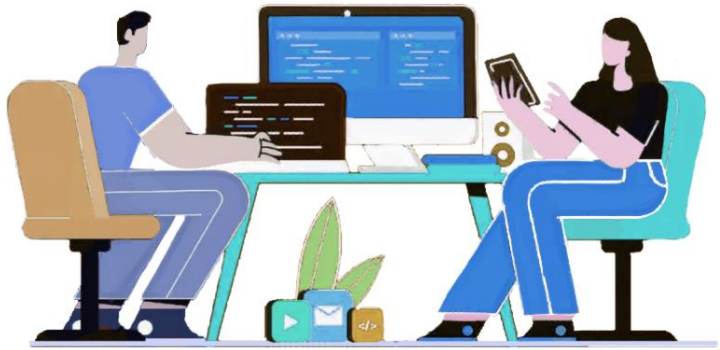
CSCC 全国大学生计算机系统能力大赛

首页 成绩查询 我的报名 个人信息 通知|新闻 查看更多比赛 登录

2023全国大学生计算机系统能力大赛
编译系统设计赛

编译系统设计赛

即将开启



编译系统设计赛

大赛时刻 往届作品开源 技术支持 等级评定标准 常见问题

全国大学生计算机系统能力大赛（以下简称"大赛"）是由系统能力培养研究专家组发起、由全国高校计算机教育研究会主办、面向高校大学生的全国性大赛。

大赛目标是以学科竞赛推动专业建设和计算机领域创新人才培养体系改革，培育我国高端芯片、关键基础软件的后备人才。大赛鼓励学生设计、实现综合性的计算机系统，培养系统级的设计、分析、优化与应用能力，提升学生的技术创新、工程实践、团队协作能力。大赛服务国家人才培养战略，以赛促学、以赛促教，为高水平计算机人才成长搭建交流、展示、合作的开放平台。

编译系统设计赛要求各参赛队综合运用各种知识（包括但不局限于编译技术、操作系统、计算机体系结构等），构思并实现一个综合性的编译系统，以展示面向特定目标平台的编译器构造与编译优化的能力。

为保证大赛顺利进行，全国大学生计算机系统能力培养大赛在机械工业出版社华章分社（北京华章图文信息有限公司）设立大赛秘书处;大赛下设指导委员会、技术委员会、评审委员会、监督委员会，负责指导、执行和监督大赛的组织、运营和奖项评审工作;本次大赛分为初赛和决赛两个阶段，初赛胜出者则有资格参加决赛。

全国大学生计算机系统能力大赛

COMP CCRG Open MP

- 2022华为毕昇杯编译系统设计赛
  - 从头开发C语言子集编译器，从功能和性能两方面进行评价
  - 第三届：全国62所高校的152支参赛队、498位队员参赛
  - 第一次组队参赛：初赛19名，决赛第6名，全国二等奖

2022年全国大学生计算机系统能力大赛编译系统设计赛（华为毕昇杯）
全国总决赛获奖名单
[按学校名称拼音排序，队员排名不分先后]

参赛队伍	学校	赛队成员
特等奖		
赫露艾斯塔	清华大学	焦景辉 王建楠 王子元 李欣隆
一等奖		
喵喵队仰卧起坐	北京航空航天大学	徐睿远 刘传 董翰元 杨宜凡
啊对对队	清华大学	范如文 张齐颢 郝子宵 徐文博
二等奖		
HexonZ	北京理工大学	朱桂潮 陈新 林恺
邯郸路企鹅编译器	复旦大学	陈立达 杜雨轩 聂绍珩 谭一凡
嘉然今天偷看乐	国防科技大学	黄子潇 熊思民 黎梓浩 贺子然
萝杨空队	哈尔滨工业大学（深圳）	梁韬 杨博康 苏亦凡
NKUER4	南开大学	时浩铭 严诗慧 林坤 梅骏逸
从容应队	西北工业大学	王翰墨 王玉佳 郑世杰 乔袁飞龙