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

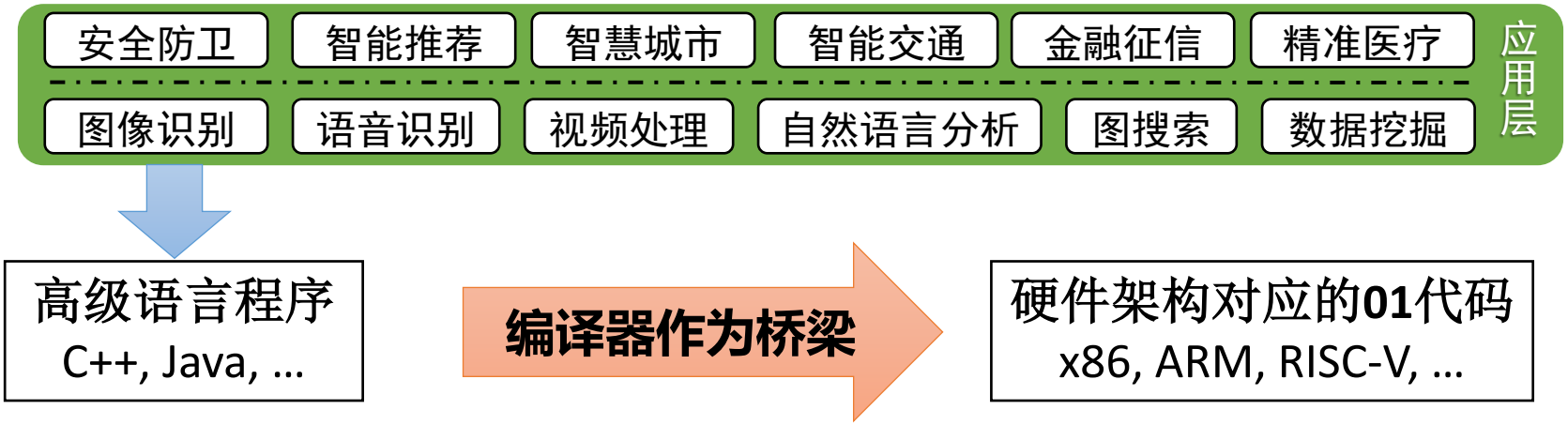
李 诚

2021-09-06

主要内容

- 课程架构及考评要求
- 编译器的历史
- 什么是编译器?
- 编译技术的应用与挑战

编译课程背景及意义



“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。”

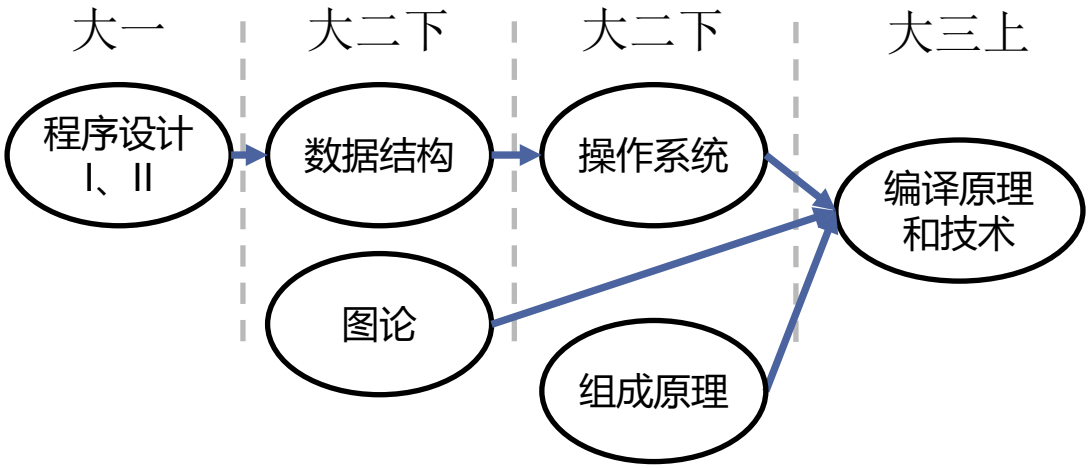
——著名计算机专家 *Alfred V. Aho*

“在供应链可控性上，仍存在编译工具依赖国外的情况。”

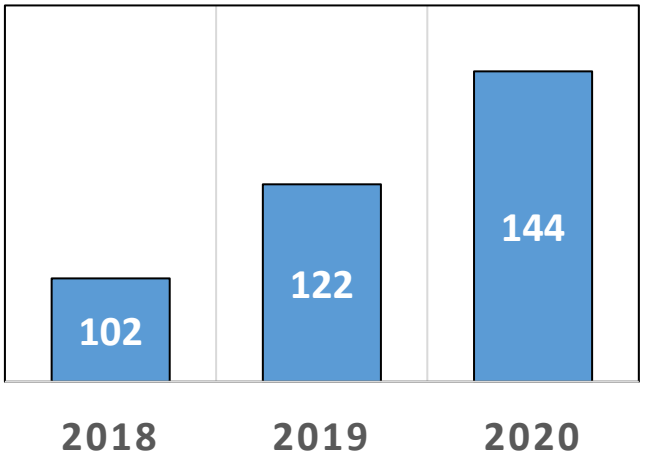
“我们应重点突破操作系统内核、编译器等关键技术。”

——《中国信息技术产品安全可控年度发展报告》

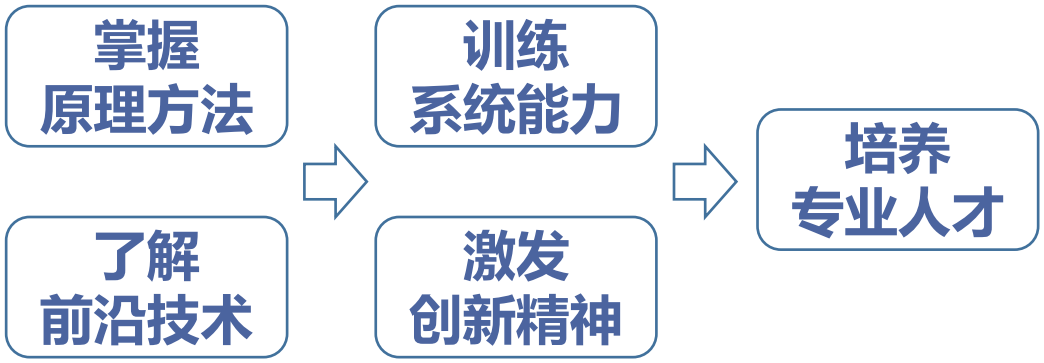
学情分析及教学目标



专业核心课及其诸多先导课程



选课人数逐年增多



课程教学目标艰巨

教学过程中面临很多挑战

课程设置

- 时间：每周一(1,2)、三(3,4)
- 地点：3C303
- 讲义+实验信息+课程讨论请移步gitlab
  - http://222.195.68.197/staff/2021fall-notice_board/-/boards

创建 Gitlab 账号

- 课程 Gitlab 网址： http://222.195.68.197/
- 使用大写学号作为 Username，且使用 USTC 邮箱进行注册
- 如果注册正确，你的主页 URL 应当是:

  http://222.195.68.197/<学号>
- 阅读 notice_board 中助教发布的三个重要 issue

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：

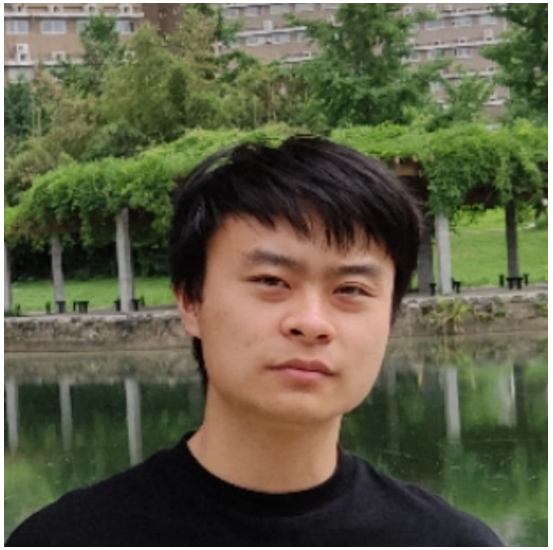
http://6.035.scripts.mit.edu/fa18/

助教天团

- 博士生：
  - 金泽文（组长）、李嘉豪

助教天团

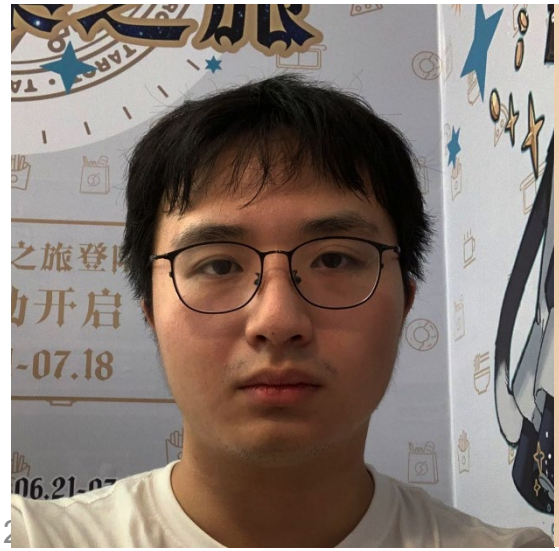
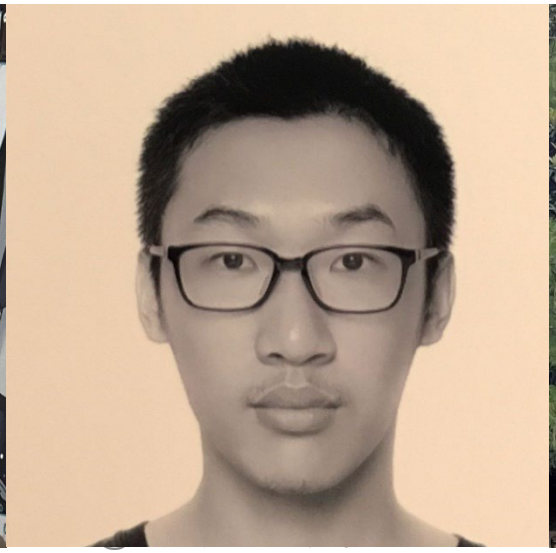
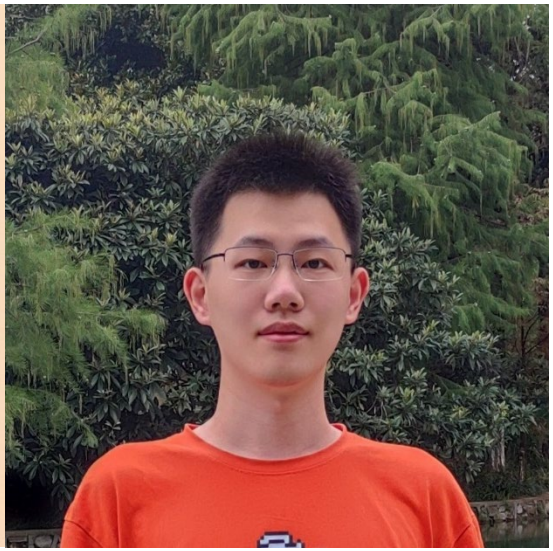
- 博士生：
  - 金泽文（组长）、李嘉豪
- 硕士生：
  - 马凯、龚平、陈清源（2020编译比赛冠军）



助教天团

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

- **博士生：**
  - 金泽文（组长）、李嘉豪
- **硕士生：**
  - 马凯、龚平、陈清源（2020编译比赛冠军）
- **大四学生：**
  - 朱恩佐（2021编译比赛三等奖）、张骏呈（2021编译比赛三等奖）、陈胤合

助教管理制度

- 每两人一组，负责一部分同学，全程跟踪所有的作业和实验
  - 等名单分配好之后，同学们就可以和对应的助教保持联系，以便我们把控学习过程质量
- 作业和实验反馈要及时、言语礼貌得当、衣着整洁得体
- 每个月至少1次习题课
  - 习题课的slides会上传到GitLab中
- 期中期末考试前进行集中答疑
- 平时的问题在GitLab中进行提问与分享

考核要求（Tentative）

- 考核内容包括理论学习、工程实践
- 成绩组成：
  - 平时考核（15%）：
    - 按时上课（特殊情况不能来需要书面请假）
    - 按时完成课后作业
  - 工程实践（45%）：
    - 个人或组队方式完成项目（>=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%）
  - Bonus加分

Bonus加分

- 课堂提问：
  - 回答正确1次0.5分（100分制）
  - 每人每学期至多5次加分
- gitlab issues：
  - 获得认可的1次0.5分（100分制）
  - 每人每学期至多5次加分
  - 包括tutorial、指出老师和助教的错误、第一时间正确回答其他同学提出的问题
- 挑战性实验：
  - 目前待定

实验安排

- Lab0： Week 1
- Lab1： Week 3
- Lab2： Week 5
- Lab3： Week 8
- Lab4： Week 10
- Lab5： Week 15

抄袭现象零容忍！

其他说明

- 课后作业上传到gitlab，使用markdown
- 课后作业可以豁免一次不交
- 工程实验作业可以晚交，但是得分会随时间衰减
- 组队作业得分与完成质量和个人贡献比有关系
  - 特别重视队员之间的协调配合，teamwork是考评的一部分

学习 Git 和 Markdown

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

- Git 使用心得分享 by 杨博远
- 在线 Git 游戏
- 给 Gitlab 设置 SSH key
- Markdown 教程

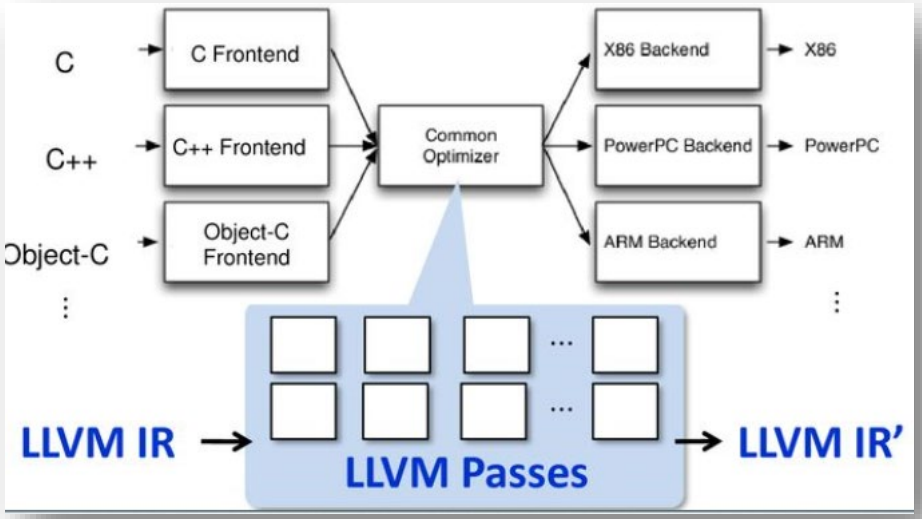
编译实验体系总览

- 目标：以**系统能力和创造力**的培养为导向，以“**金课**”标准为指导思想，根据计算机学科发展的趋势和新时代学生的成长需求构建**现代编译实验体系**，体现**先进性**、**挑战度**，增强学生**获得感**

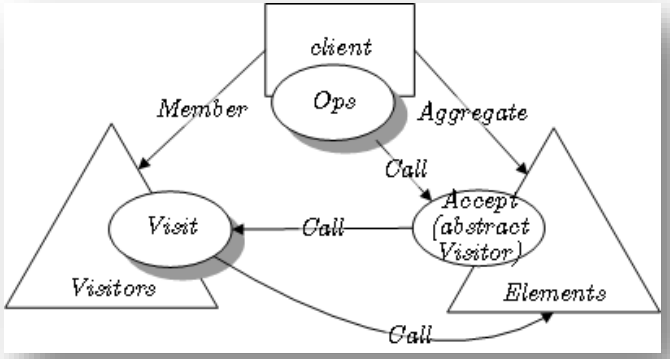
源语言特性	以Cminus语言为基础，增加对浮点数、数组等复杂特性的支持
词法语法分析器	使用Flex+Bison工具构造编译器前端(同时鼓励使用ANTLR)，重点考察对Cminus语言正则表达式和语法产生式的抽象能力
中间代码生成器	使用基于访问者模式、兼容LLVM的中间代码生成编程框架(**自研**)，编写生成器，自动化生成 Cminus程序的中间代码
中间代码优化器	使用封装好的基本块、流图、前驱后继结点、调用者使用者链表等常用数据结构及其接口(**自研**)，实现经典的代码优化算法
目标代码生成器（远期）	提供RISC-V、MIPS等多指令集后端，鼓励对寄存器分配算法、指令选择算法等进行探索验证，学会用模拟器评测目标代码

编译实验的改革——先进性

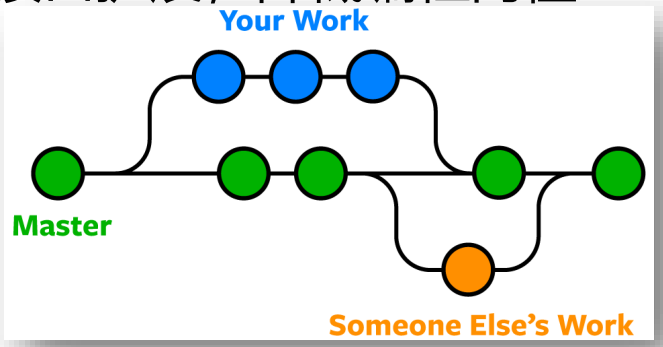
❑紧跟时代前沿，融合先进技术，架起课程教育与工业界技术革新之间的桥梁



在全国较早地将**LLVM——这一工业界广泛使用的、产品级现代编译工具链**引入到课程实验中，建立贴近业界需求、适合多种教学目标的循序渐进的课程实践体系（张昱老师发起）



使用适合编译器编写的**访问者设计模式**，将数据结构与操作分离开来，便于接口扩展，降低编程门槛



采用**git进行版本管理**，便于学生的敏捷开发、项目管理和协同工作

编译实验的改革——挑战度

❑合理增加实验难度、拓展实验深度，体现课程的高阶性、创新性和挑战度

编程与工程实践能力 →
- 5次大实验，要求学生独立完成至少1000行的C++代码
- 要求学生为每次实验撰写工程设计和测试文档，完整记录全过程

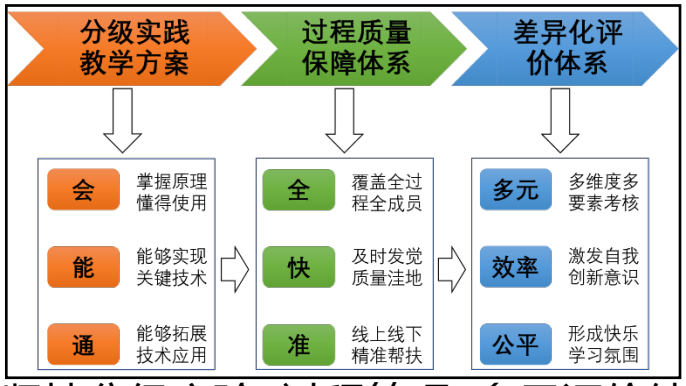
自主学习能力 →
在16周内，要求学生从零开始，学会使用C++ STL、LLVM、Flex、Bison、Clang、Git、GDB、VM、Docker等数十种新编程工具和系统软件

领导与团队协作能力 →
- 安排2次分组实验，对组长和组员进行分别考核，衡量贡献比
- 要求学生熟练使用Git进行合作开发

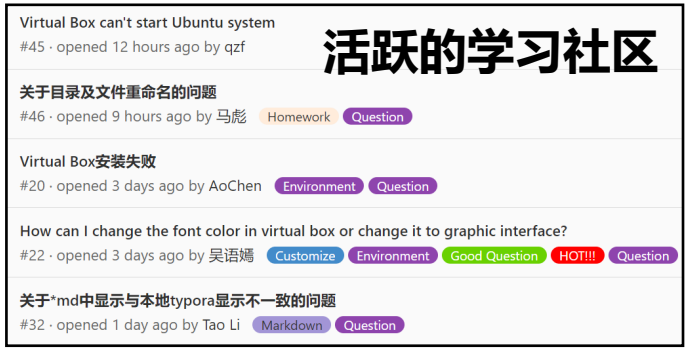
探索与创新能力 →
通过分级实验设计和加分机制，鼓励学有余力的学生选做更有挑战的项目，比如：函数内联，尾递归消除...

编译实验的改革——获得感

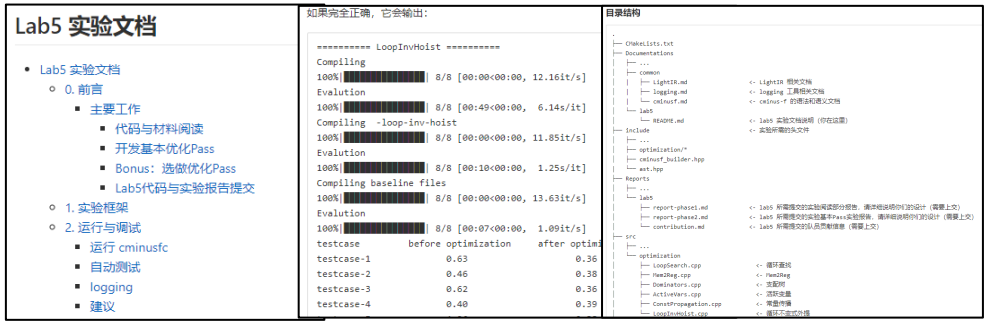
❑结合学生差异化需求，降低实践入门难度，设计递进式实践教学方案及过程质量管理和多元评价体系



坚持分级实验+过程管理+多元评价结合的指导思想，让每一位学生都**学有所思、学有所乐、学有所得**



增强**学生的主体意识**，2020年，144名同学提交258个内容丰富的gitlab issues，其中61名同学获得额外加分



- **自主编写**了45000余字的实验文档，**自主设计**了简化版的Light-LLVM-IR框架，明确实验目的和考核标准，降低了工程实践的门槛
- 将挑战度高的实验(如代码优化)**拆解**为若干层次分明、衔接有序的小实验，**及时反馈**结果

建设成效

- 形成了**助教轮值制度**，快速地解答学生提问，获得学生广泛好评。
- 2020年**提交率达到94.4%**，在选课人数增加了18.0%的前提下，相较2019年**提高了6.3%**。
- 以lab5为例，**57组中12组参加了高阶(非必须)实验**，完成质量有区分度，达到拔高目的。
- 2019年，1组同学**发现一个隐藏在LLVM中的bug**，获得官方人员的认可，该bug随后被修复。

主要内容

- 课程架构及考评要求
- 编译器的历史
- 什么是编译器?
- 编译技术的应用与挑战

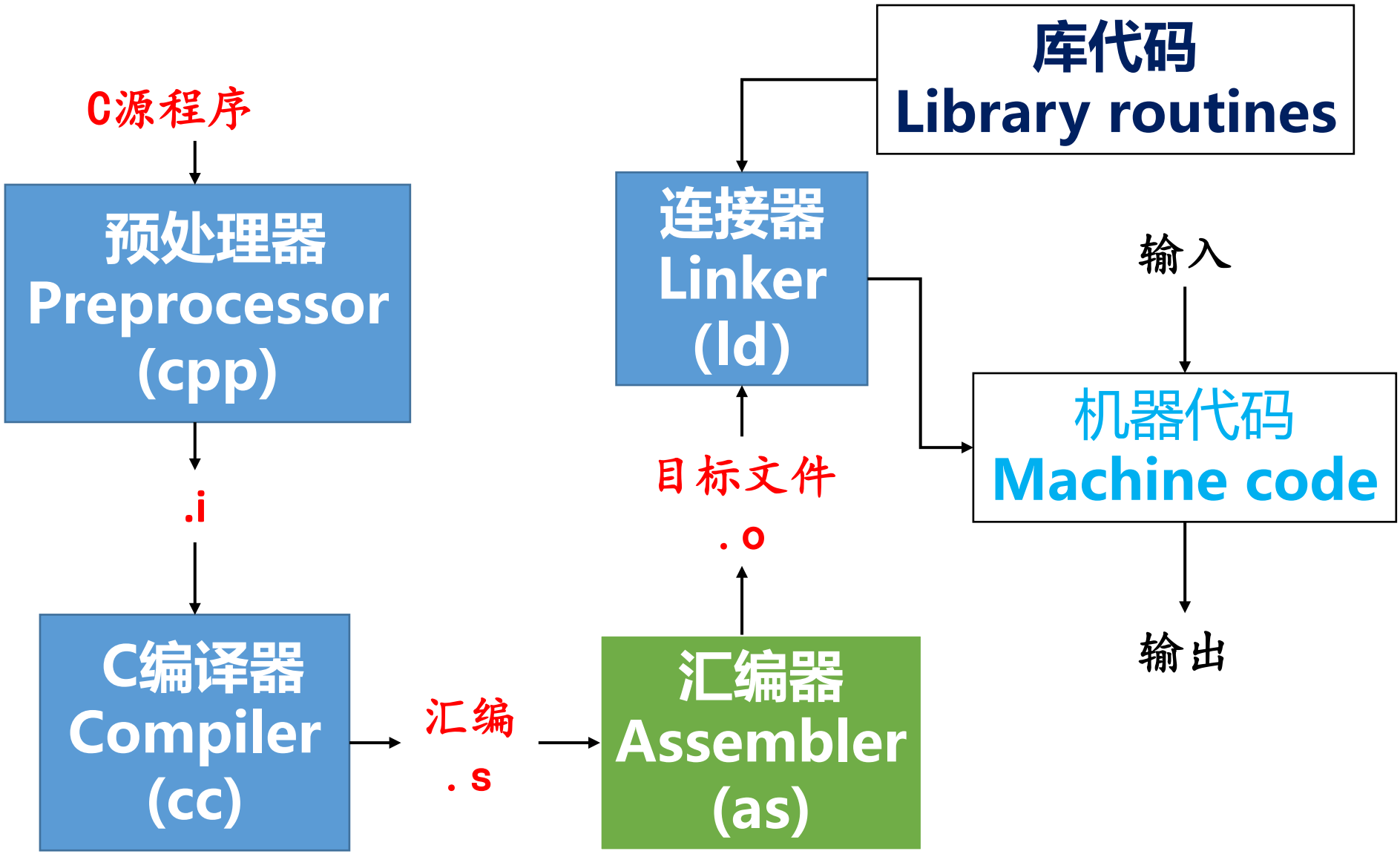
从C程序到可执行文件


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

/* helloworld.c */
```

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

C程序的编译过程分解

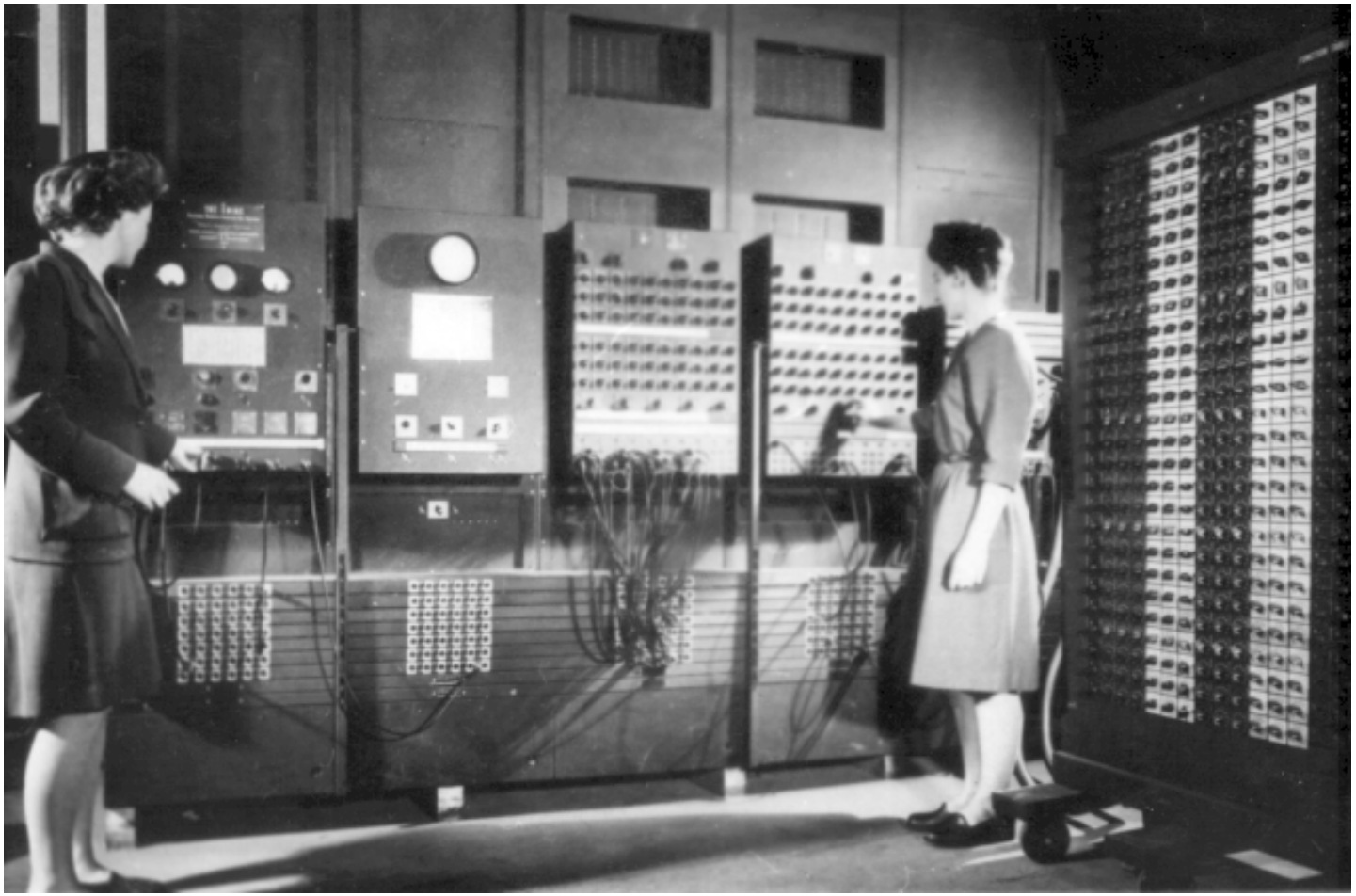

C源程序
预处理器
Preprocessor
(cpp)
.i
C编译器
Compiler
(cc)
汇编
.s
汇编器
Assembler
(as)
目标文件
.o
连接器
Linker
(ld)
库代码
Library routines
输入
机器代码
Machine code
输出

C程序的编译过程分解

- **预处理：**
  - **cpp** helloworld.c > helloworld.i
  - 将stdio.h内容放到helloworld.c中，并生成新文件
- **编译：**
  - **gcc** -S helloworld.i
  - 生成汇编代码
- **汇编：**
  - **as** helloworld.s -o helloworld.o
  - 翻译为机器码

- 链接：
  - **ld** -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib64/crt1.o /usr/lib64/crti.o /usr/lib64/crtn.o helloworld.o /usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtbegin.o -L /usr/lib/gcc/x86_64-redhat-linux/4.1.2/ -lgcc -lgcc_eh -lc -lgcc -lgcc_eh /usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtend.o -o helloworld
  - 将生成的代码与系统组件（比如标准库、动态链接库等）结合起来

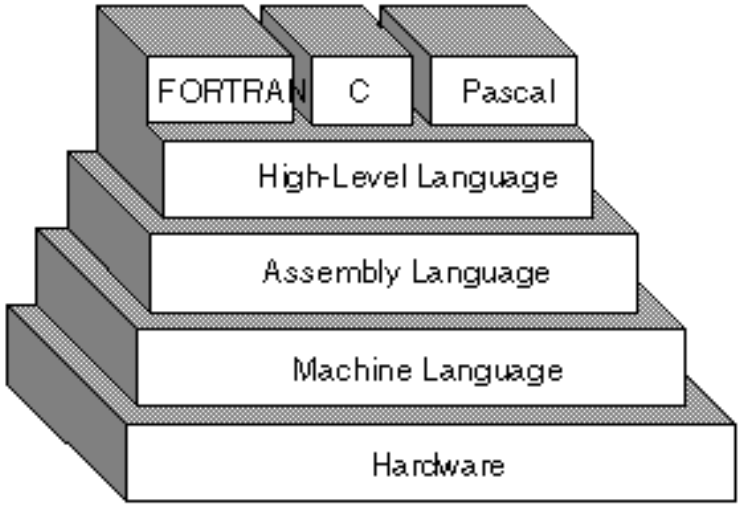
计算机编程的“恐龙”时代



程序员Betty Jean Jennings (左) 和 Fran Bilas (右) 在操作 ENIAC 的主控制面板

汇编语言和汇编器的出现

- 可以理解为对应硬件指令的助记符，有一一对应的关系
- 汇编器可将符号化的指令序列翻译为对应的二进制数字序列（机器码序列）

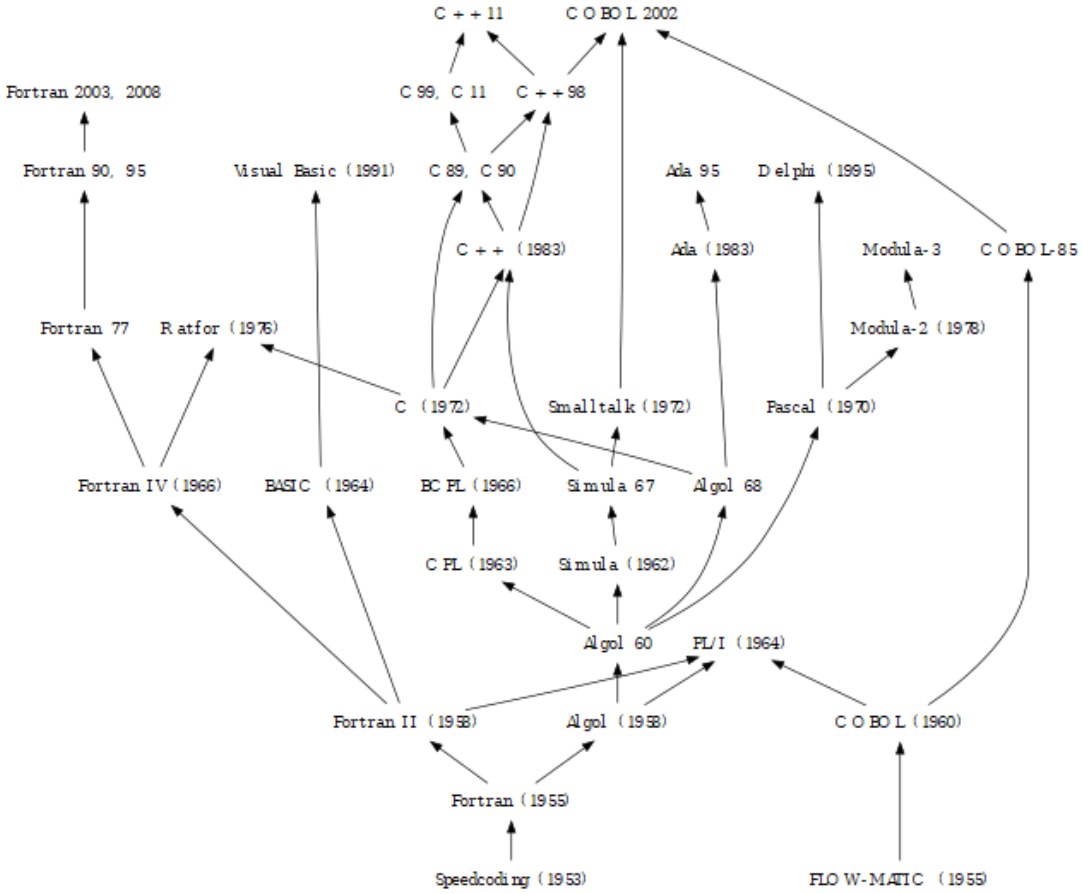


机器码：10110000 01100001

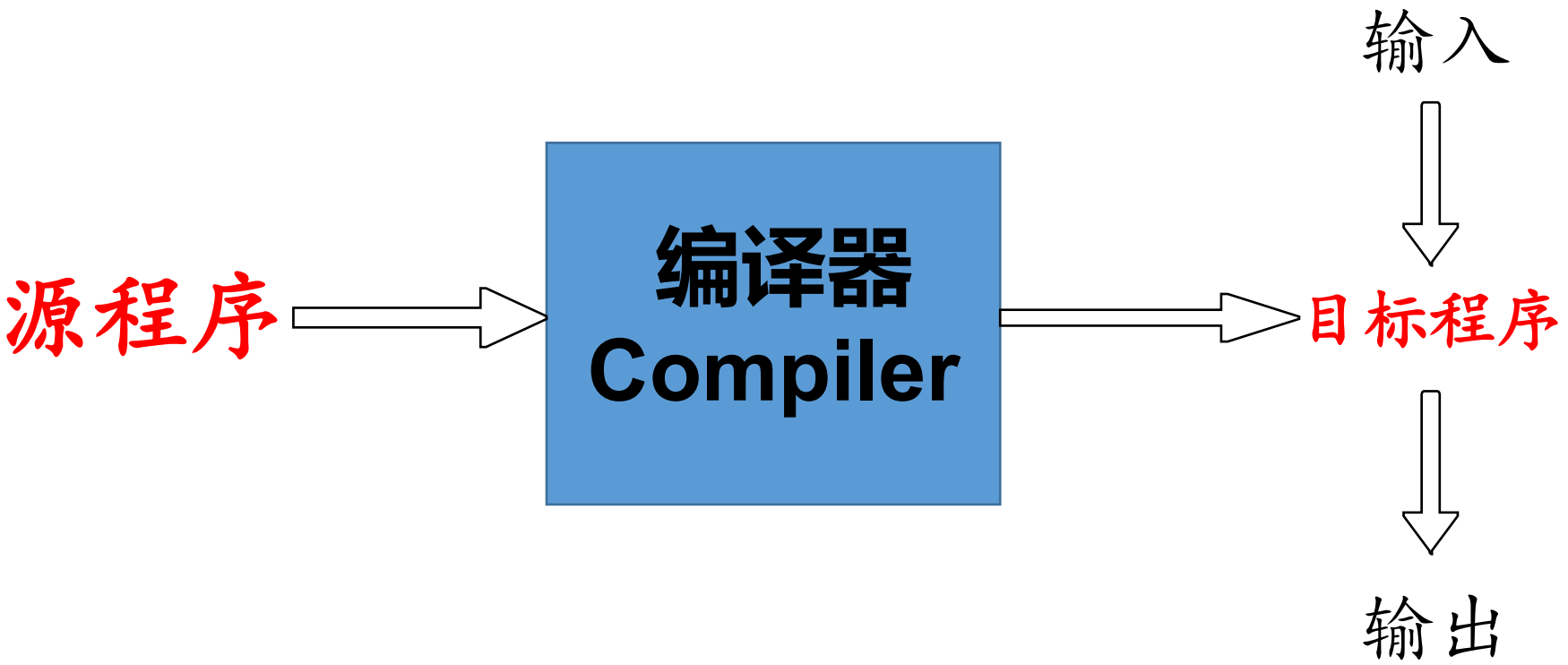
汇编码：MOV AL, 61h

“懒人”的进化过程

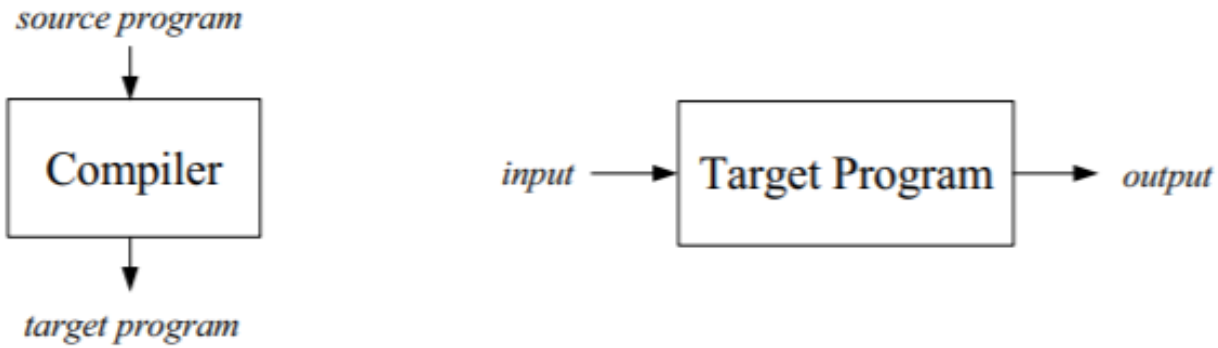
- 能不能像写数学公式一样写程序?
- 能不能像写剧本一样写程序?



编译器发挥了桥梁作用



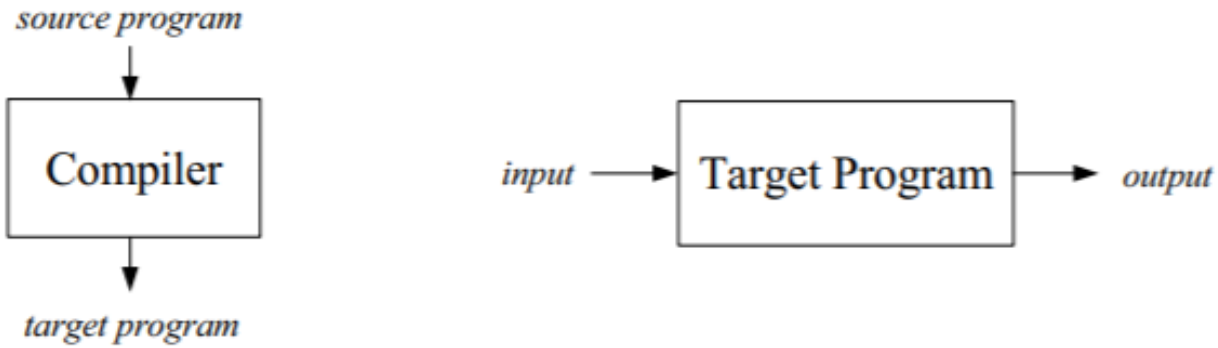
编译器与解释器的区别



1. Execution of a compiled program

2. Execution of an interpreted program

编译器与解释器的区别



1. Execution of a compiled program

2. Execution of an interpreted program

- 编译器产生的代码往往与硬件相关，不利于移植
- 解释器更通用，但程序的每一次运行都需要翻译一次
- Java是比较好的折中方案，编译成opcode，然后再解释opcode

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

编译器的历史

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

- 1952
  - Grace Hopper wrote the first compiler for A-0 programming language
- 1957-58
  - John Backus and his team wrote Fortran compiler, the first complete compiler
  - Optimization was an integral component of the compiler.
- 1960
  - COBOL was an early language to be compiled on multiple architectures.
- https://en.wikipedia.org/wiki/Compiler

主要内容

- 课程架构及考评要求
- 编译器的历史
- **什么是编译器?**
- **编译技术的应用与挑战**

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

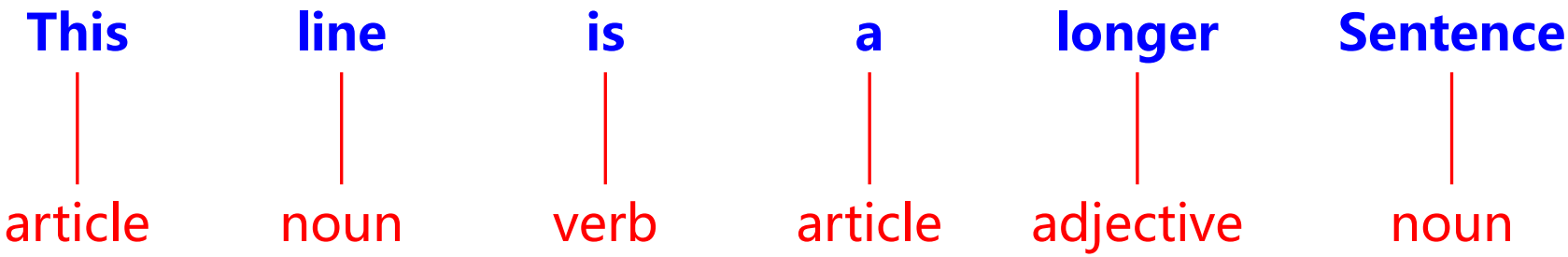
Source code 源程序

Token Stream 记号流

Symbol Table 符号表

Error Handler 错误处理

- 人类在理解自然语言时，首先要识文断字

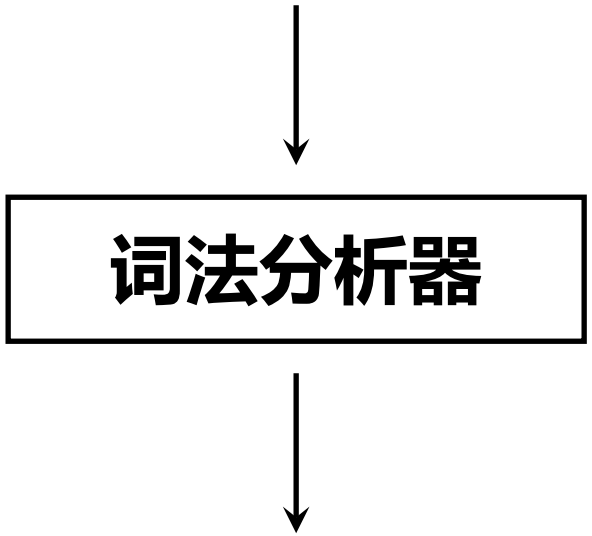


- 通常称为线性分析（linear analysis）

❑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

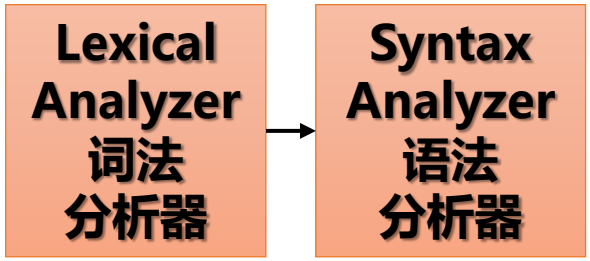


符 号 表

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

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

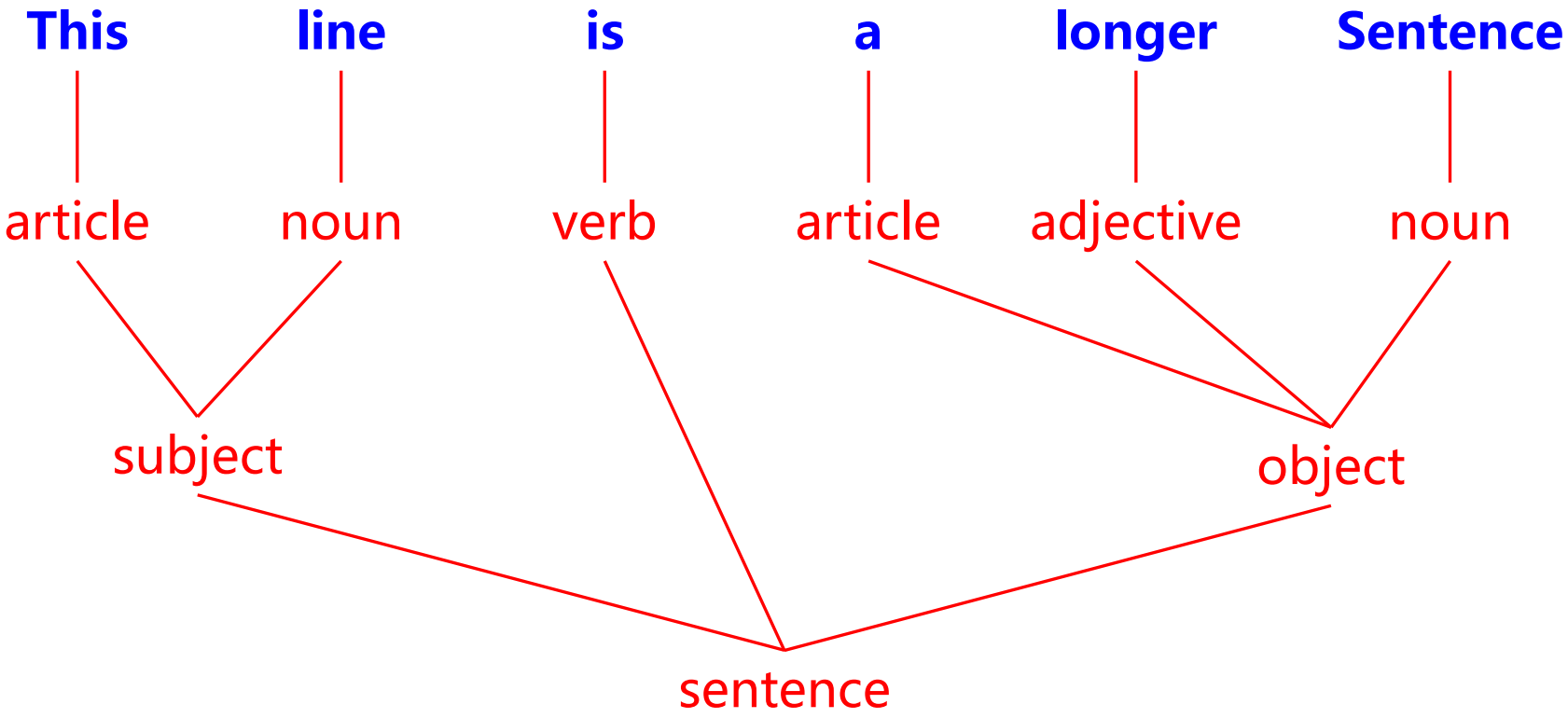
编译器的构造/阶段



Symbol Table 符号表

Error Handler 错误处理

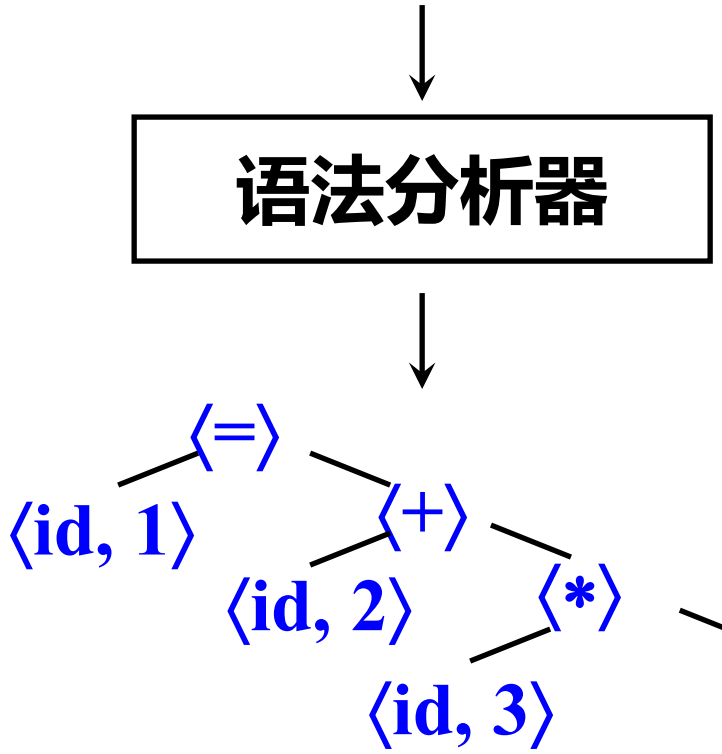
❑人类在理解自然语言时，其次要理解句子结构



❑也称为层次分析（Hierarchical analysis）

- 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

Symbol Table 符号表

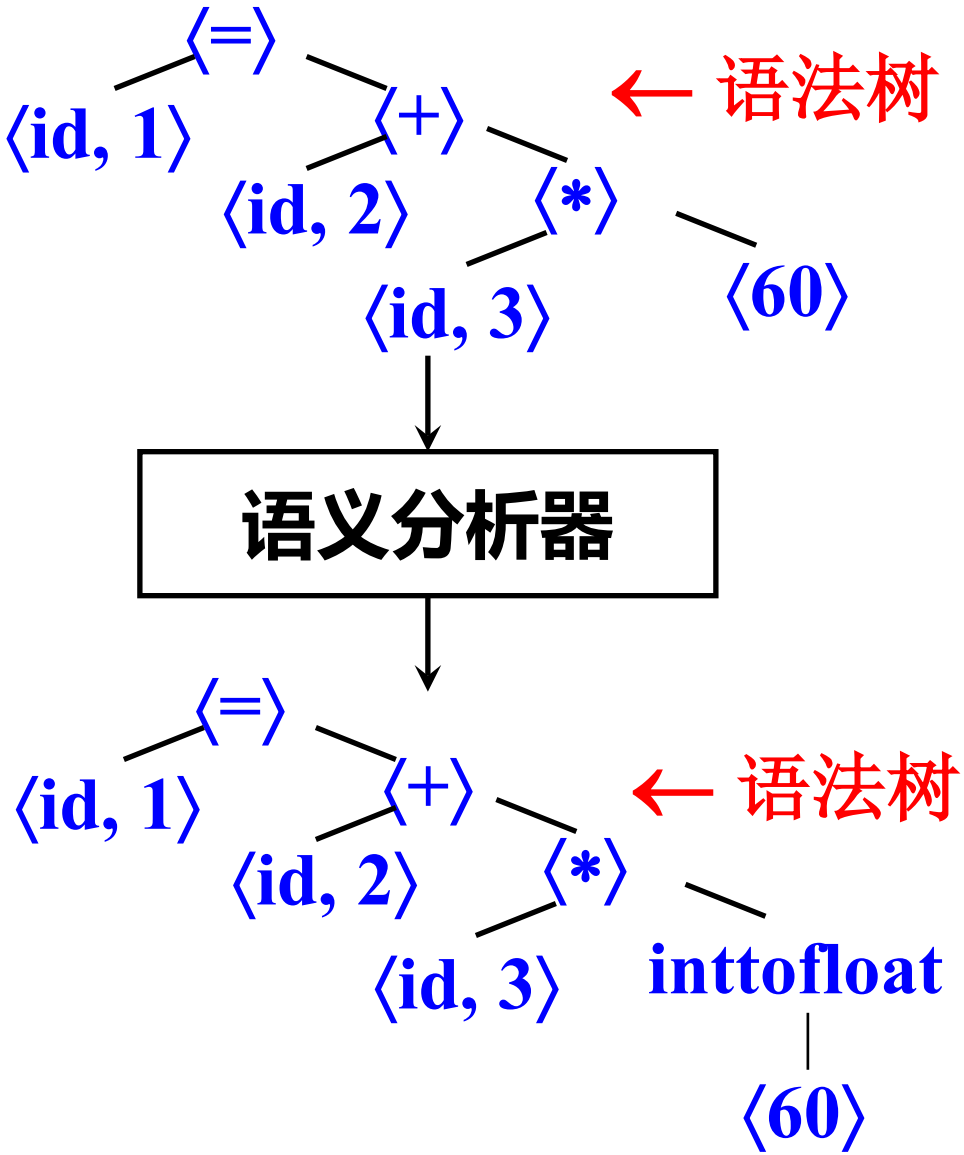
Error Handler 错误处理

- 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）

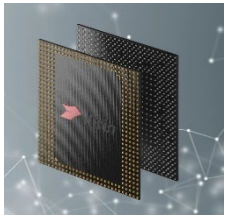
符 号 表

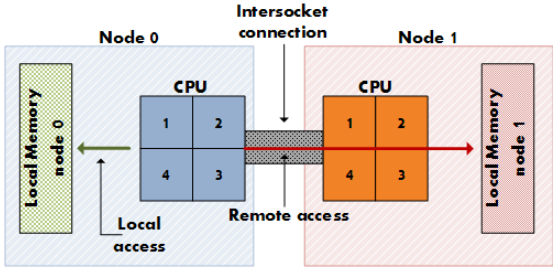
1	position	· · ·
2	initial	· · ·
3	rate	· · ·



编译器面临的挑战

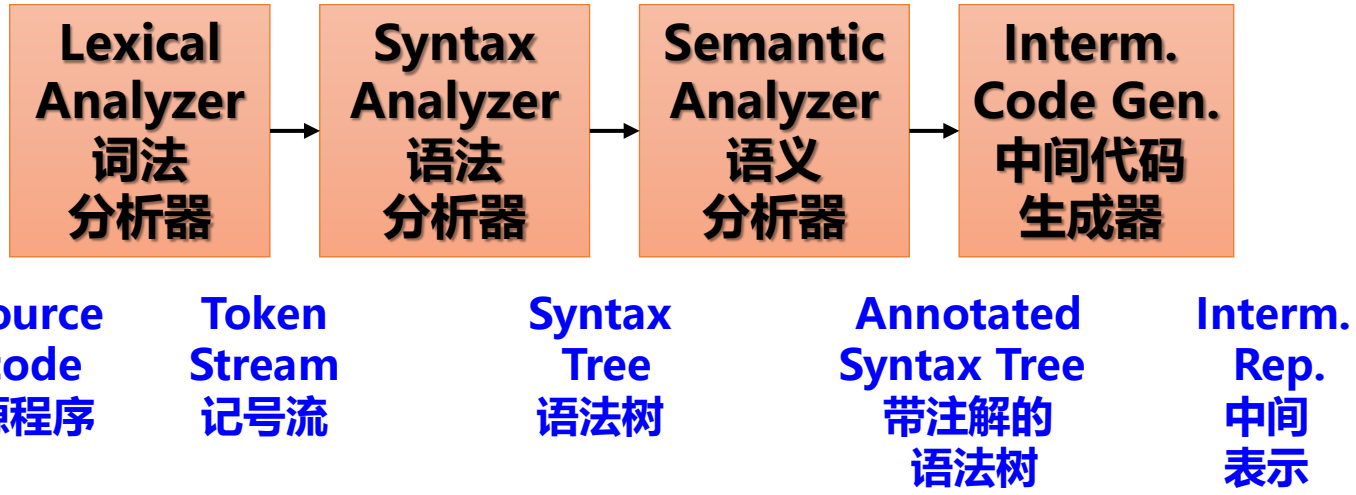
- 计算机是不断进化的
  - 体系结构的改变→编译器的改变
  - 新的特征产生新的问题





- 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)

编译器的构造/阶段

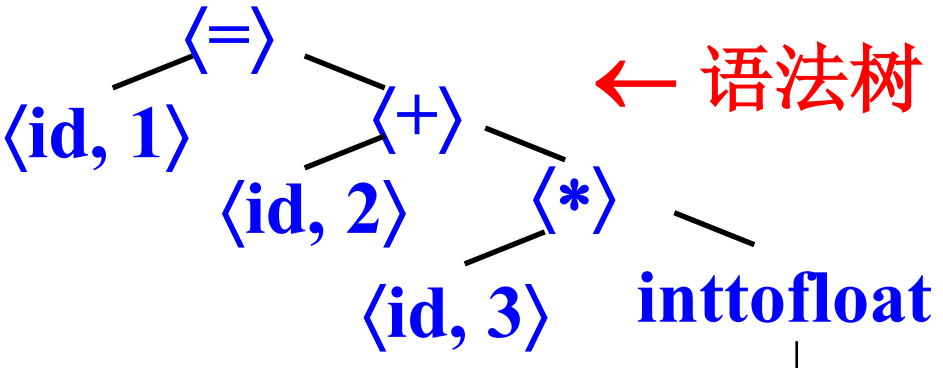


Symbol Table 符号表

Error Handler 错误处理

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

Symbol Table 符号表

Error Handler 错误处理

代码优化

- 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	· · ·

↓

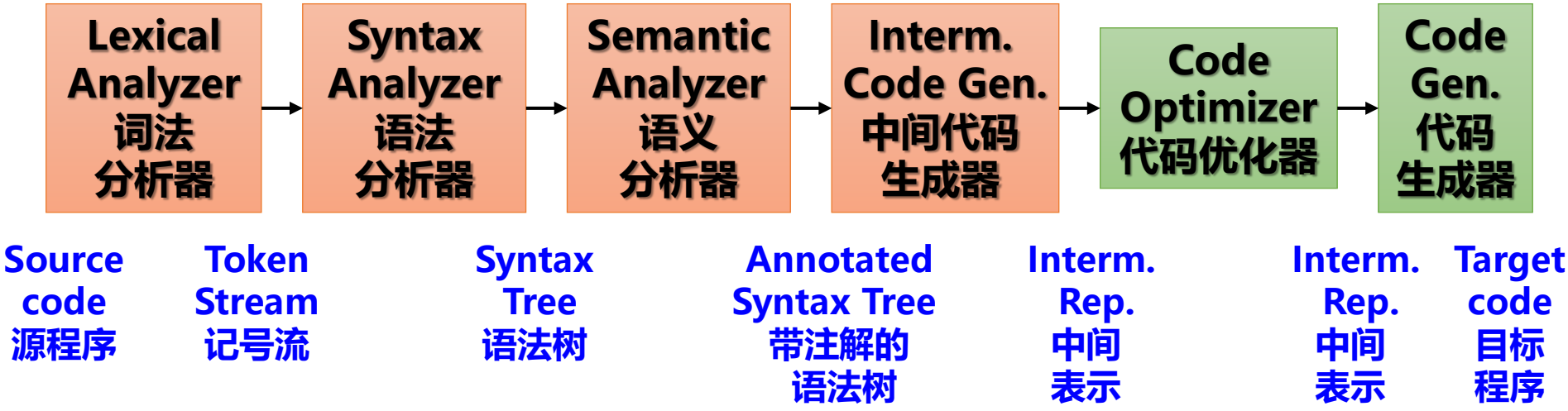
代码优化器

↓

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

← 中间代码

编译器的构造/阶段



Symbol Table 符号表

Error Handler 错误处理

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

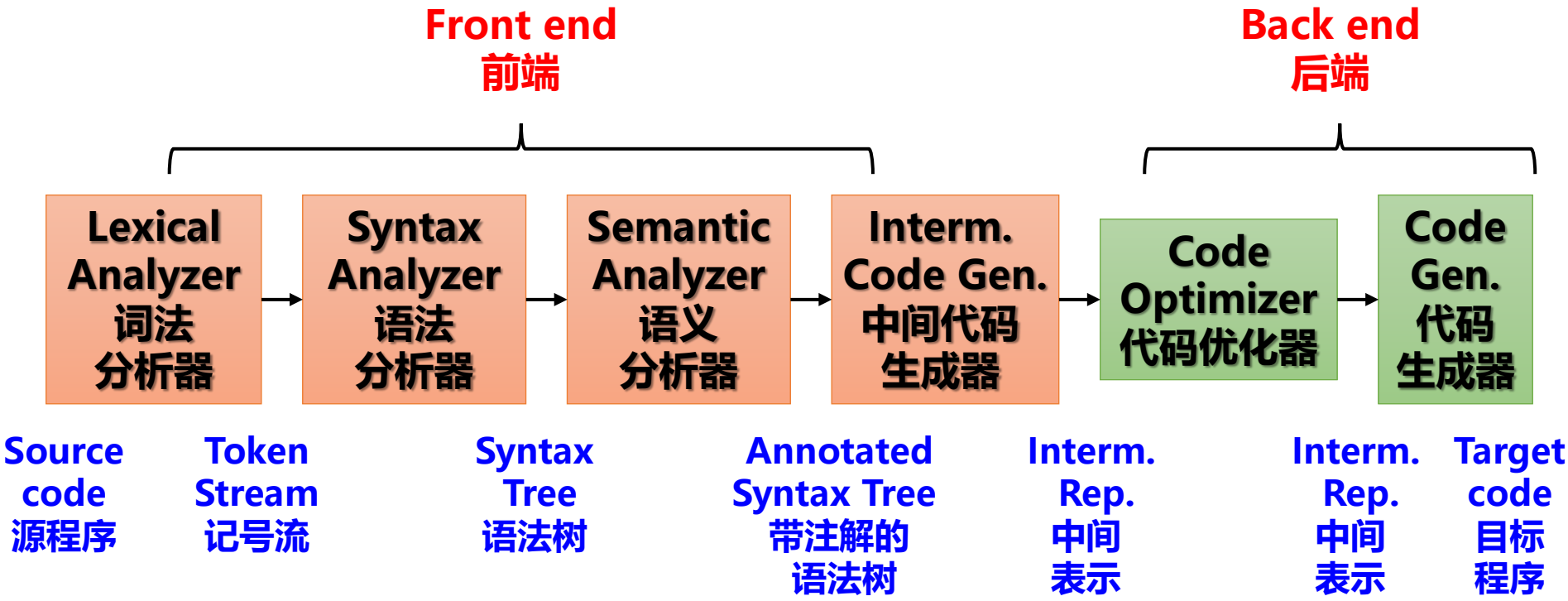
↓

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



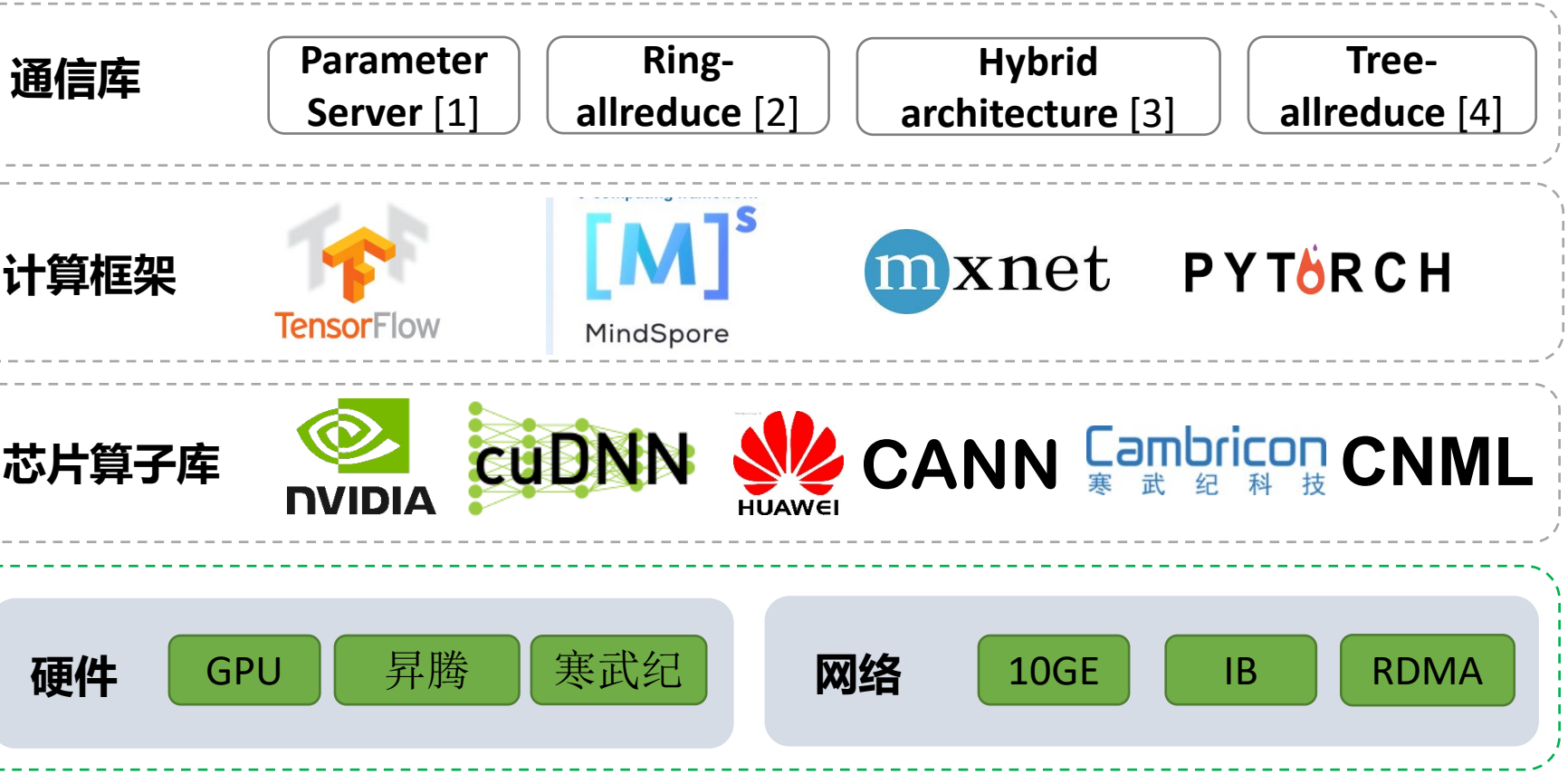
Symbol Table 符号表

Error Handler 错误处理

主要内容

- 课程架构及考评要求
- 编译器的历史
- 什么是编译器?
- **编译技术的应用与挑战**

深度学习框架



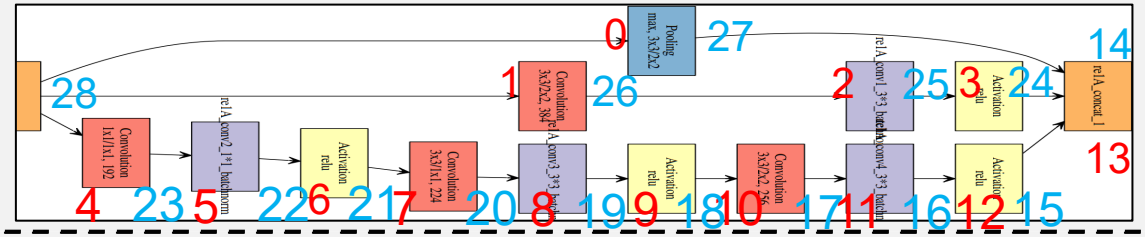
[1] Scaling distributed machine learning with the parameter server. In OSDI, 2014
[2] Horovod: fast and easy distributed deep learning in tensorflow. arXiv 2018
[3] Parallax: Sparsity-aware Data Parallel Training of Deep Neural Networks, EuroSys 2019
[4] https://devblogs.nvidia.com/fast-multi-gpu-collectives-nccl/

编译深度学习任务

user define NN

```
p1 = mx.sym.Pooling(input, kernel=(3, 3), stride=(2, 2), name='%s_maxpool_1' %name)
c2 = Conv(input, 384, kernel=(3, 3), stride=(2, 2), name='%s_conv1_3*3' %name)
c3 = Conv(input, 192, kernel=(1, 1), pad=(0, 0), name='%s_conv2_1*1' %name)
c3 = Conv(c3, 224, kernel=(3, 3), pad=(1, 1), name='%s_conv3_3*3' %name)
c3 = Conv(c3, 256, kernel=(3, 3), stride=(2, 2), pad=(0, 0), name='%s_conv4_3*3' %name)
concat = mx.sym.Concat(*[p1, c2, c3], name='%s_concat_1' %name)
```

user level CG



NNVM indexed CG (**DFS**)

fwd	Id	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	op	pl	cv	bn	ac	cv	bn	ac	cv	bn	ac	cv	bn	ac	ct
bwd	Id	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	op	ct	ac	bn	cv	ac	bn	cv	ac	bn	cv	ac	bn	cv	pl	ad

- Computation graph (DAG)作为一种中间表示（IR），被广泛应用于大数据处理的各个领域

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，栈，队列
- 系统
  - 内存空间分配与命名，运行时支持，多趟系统，操作系统

编译器是学科交叉的产物

- 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，程序分析
- 软件工程
  - 软件开发环境，调试
- 人工智能
  - 启发式代码优化

编译技术中的关键科学问题

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

- 如何抽象和形式化描述推陈出新的语言?
- 如何应对不断发展的硬件及指令集?
- 如何做好代码优化?

编译技术中的关键科学问题

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

- 如何抽象和形式化描述推陈出新的语言?
  - 正则表达式
  - 自动机
  - 上下文无关文法
- 如何应对不断发展的硬件及指令集?
  - 语言的中间表达形式
  - 语法制导翻译技术
- 如何做好代码优化?
  - 数据流、控制流分析等

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

征途漫漫，惟有奋斗！