同济大学
TONGJI UNIVERSITY

编译原理

引论

丁志军

dingzj@tongji.edu.cn

龙书


Principles of Compiler Design
Alfred V. Aho
Jeffrey D. Ullman
百家号/学术头条


计 算 机 科 学 丛 书
第2版
编译原理
Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman
Compilers
Principles, Techniques and Tools
Second Edition
机械工业出版社
China Machine Press

2020年图灵奖获奖人：

- 哥伦比亚大学计算机科学名誉教授**阿尔佛雷德•艾侯(Alfred Aho)**
- 斯坦福大学计算机科学名誉教授**杰弗里•乌尔曼(Jeffrey Ullman)**

获奖理由

- 表彰两位在编程语言实现领域对基础算法和理论的贡献。
  - 编程语言理论和实现
  - 算法设计和分析
- 表彰两位将相关成果系统成书，并通过他们的若干著作影响了几代计算机科学家。

正是编译器的存在，才让人类语言转换为计算机可以理解的 0 和 1。

可以说，是Aho和Ullman的编译技术才使得计算机能看懂人类的编程语言。

内容线索

1. 什么叫编译程序
2. 编译过程概述
3. 编译程序的结构
4. 编译程序的生成
5. 总结

楔子

- **A机器仅具备二目四则运算（ +、 -、×、÷ ）的能力**

例如： **3+4**

4×2

2+6

8÷4

楔子

运算表达式的求解

$$3+4\times2+6$$

楔子

- 现要开发基于A机器的计算器，可以计算形式如下的表达式：

$$3+4\times2+6$$

- 要把上述表达式拆分成如下几步：

Step1: $T_1:=4\times2$

Step2: $T_2:=3+T_1$

Step3: $T_3:=T_2+6$

Step4: 输出T_3

楔子

- A机器仅具备二目四则运算（+、-、×、÷）的能力
- 现要开发基于A机器的计算器

  - 识别操作数、运算符
  - 根据表达式的运算规则分析表达式
  - 分解表达式为A机器可接受的一组二目运算表达式
  - 在A机器上执行，并输出结果

程序语言技术的发展

表示机器实际操作的数字代码

以符号形式给出指令和存储地址

类似于数学定义或自然语言的简洁形式编写程序的操作

机器语言

汇编语言

高级语言

C7 06 0000 0002

MOV X, 2

X=2

表示在IBM PC 上使用的Intel 8x86处理器将数字2移至地址0 0 0 0（16进制）的指令

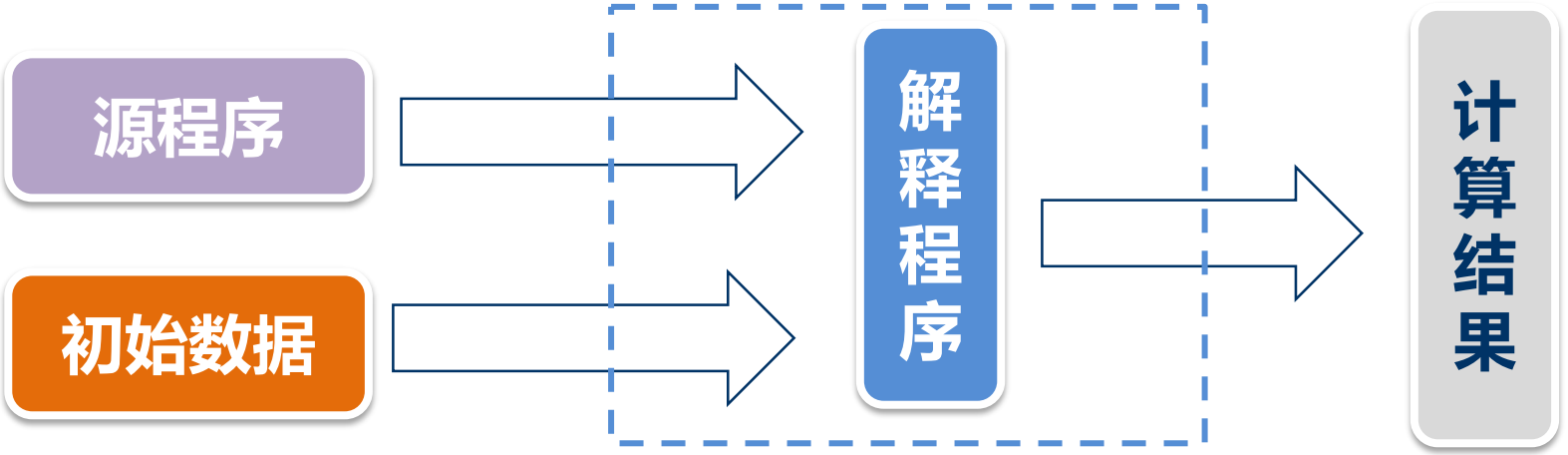
?

程序的执行方式

- 高级语言程序通常采用**解释方式**和**编译方式**两种方式执行

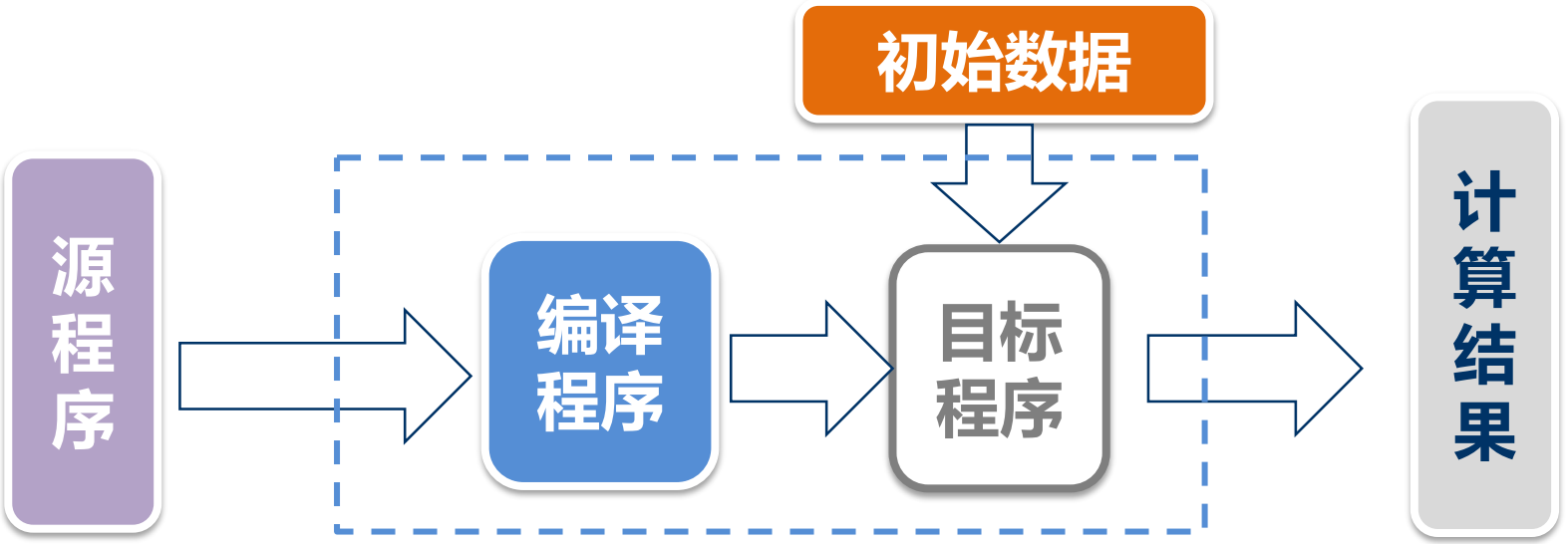
解释程序

边解释边执行源语言程序，不产生目标语言程序。



编译程序

能够把某种语言的程序转换成另一种语言的程序，而后者与前者在逻辑上是等价的。



解释程序和编译程序的区别

	功能	工作结果	实现技术上
编译程序	源程序的一个**转换**系统	源程序的**目标代码**	把中间代码转换成目标程序
解释程序	源程序的一个**执行**系统	源程序的**执行结果**	执行中间代码

简单的说： 编译就是全文翻译，全部翻译完才执行。

解释就相当于同声翻译，边翻译边执行。

解释程序和编译程序的区别

解释方式

逐个语句地分析和执行

- 如Basic，Ruby、Python
- 优点：易于查错 启动速度快
  内存使用少
  跨平台 移植性好
- 缺点：效率低，运行速度慢

编译方式

对整个程序进行分析
翻译成等价机器语言程序后执行

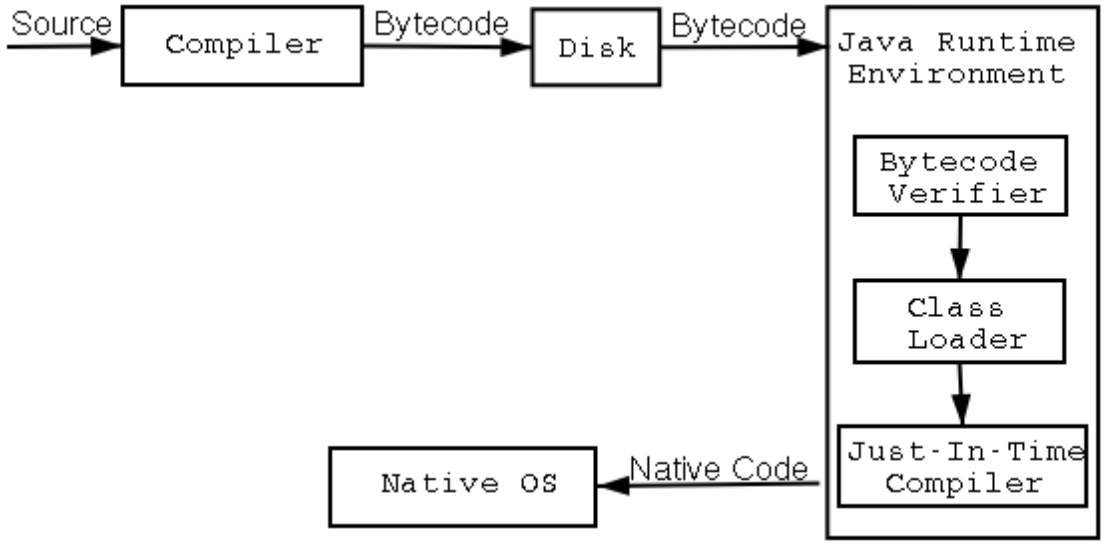
- 如Pascal，Fortran，C
- 优点：执行效率高
- 缺点：在运行中发现的错误
  必须查找整个程序确定
  移植性不好

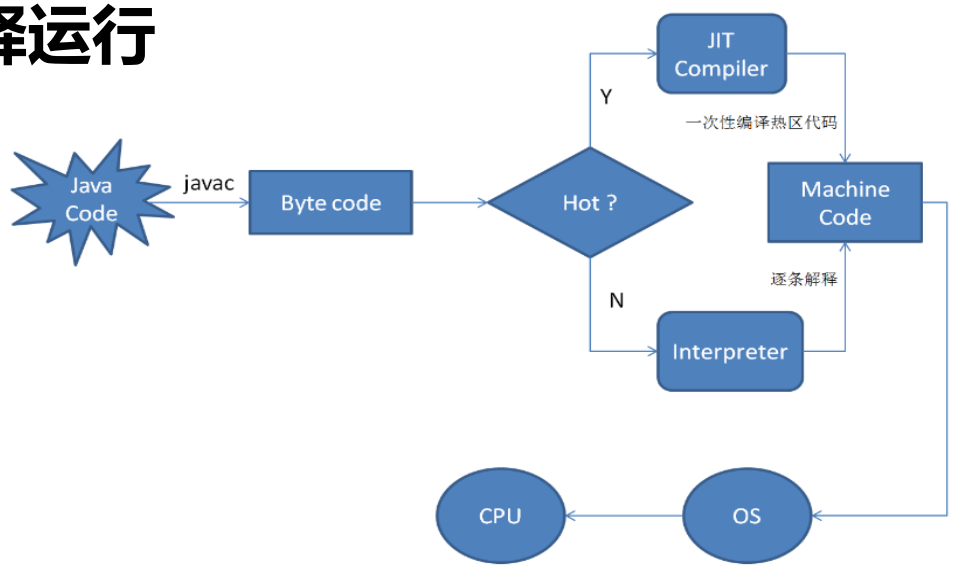
- Java程序从源文件创建到程序运行要经过两大步骤：

1、源文件由编译器编译成字节码（ByteCode）

2、字节码由Java虚拟机解释运行。

- Java程序既要编译同时也要经过JVM的解释运行





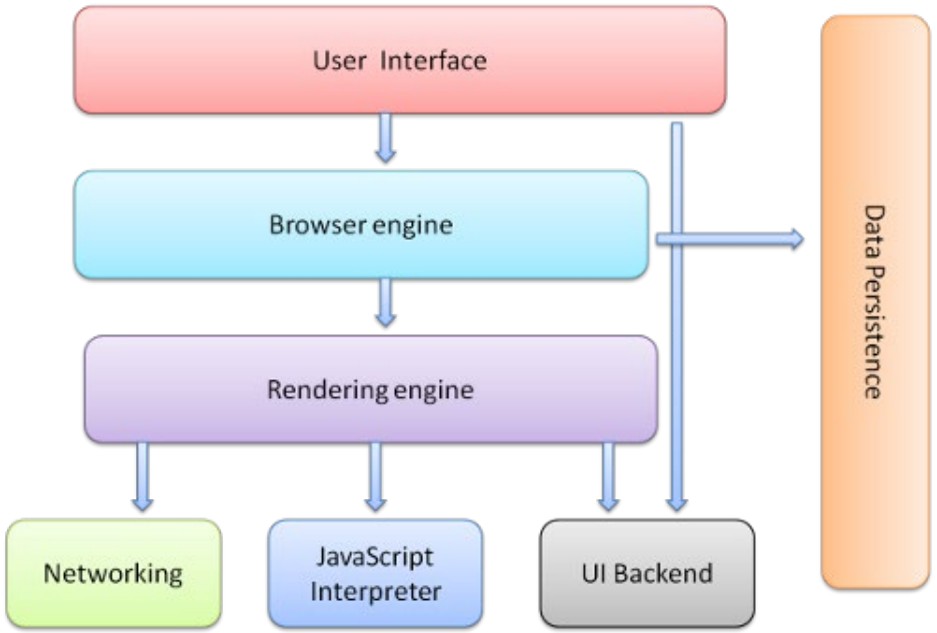
JIT （ just in time，即时编译）

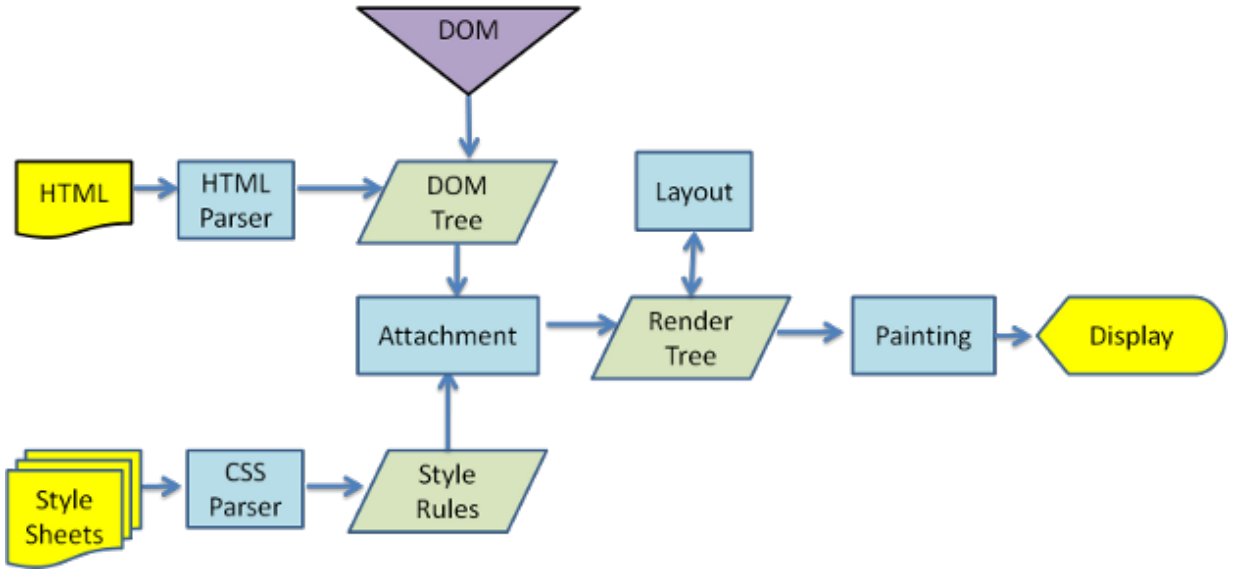
编译技术的应用

- 很多机器学习的算法都会用Python来实现
- Python作为开源的编程语言，有多个版本的解释器，
  - C实现的CPython（官方提供）
  - Java实现的Jpython
  - Python实现的PyPy（Armin Rigo公司开发）
    - PyPy采用JIT（just-in-time）技术，对Python代码进行动态编译
- 在Kaggle的生物加速计竞赛（Kaggle‘s Biometric Accelerometer Competition）中，用PyPy替换了CPython来对5650万行数据进行特征生成，数据处理时间由原来的12个小时缩短为20分钟

编译技术的应用

- **文档转换工具**
  - ps2pdf
  - Latex2html
- **Web 浏览器**
  - https://www.oschina.net/p/webkit



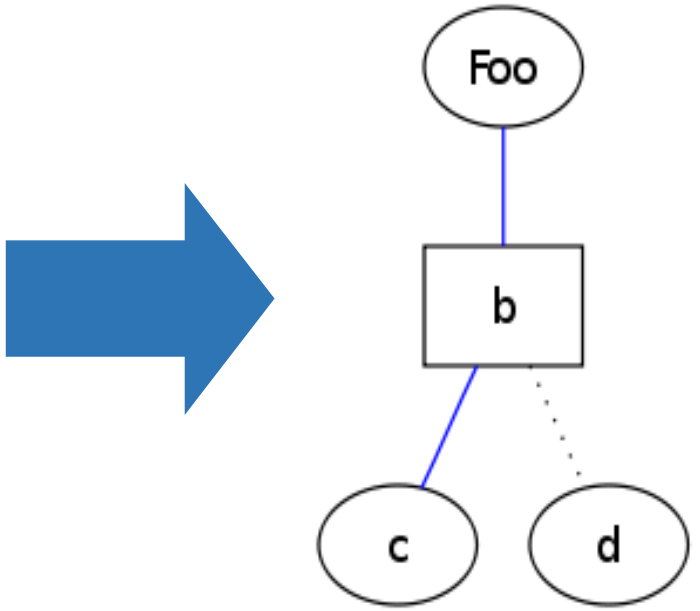


开源渲染引擎Webkit

编译技术的应用

- **图形绘制工具Dot**

```
graph graphname {
    // label属性可以改变节点的显示名称
    a [label="Foo"];
        // 节点形状被改变了
    b [shape=box];
        //  a-b边和b-c边有相同的属性
    a -- b -- c [color=blue];
    b -- d [style=dotted];
```



- **语言转换工具**
  - F2c （Fortran to C）
  - P2c （Pascal to C）

内容线索

- √ 什么叫编译程序
- 2. 编译过程概述
- 3. 编译程序的结构
- 4. 编译程序的生成
- 5. 总结

一个类C源程序片段

```
void main(int b,int c)
{
    int a;
    a=b+(c-2);
}#
```

?

```
MOV   EAX,    [EBP+8+0]
SUB   EAX,    2
MOV   EBX,    [EBP+8+4]
ADD   EBX,    EAX
MOV   [EBP-4], EBX
```

1. 词法分析
2. 语法分析
3. 语义分析中间代码生成
4. 优化
5. 目标代码生成

编译过程概述

- 掌握编译过程的五个基本阶段，是我们学习编译原理课程的基本内容，把编译的五个基本阶段与英译中的五个步骤相比较，有利于对编译过程的理解：

英译	编译
1. 识别出句子中的一个个单字	1. 词法分析
2. 分析句子的语法结构	2. 语法分析
3. 初步翻译句子的含意	3. 语义分析中间代码生成
4. 译文修饰	4. 优化
5. 写出最后译文	5. 目标代码生成

词法分析

- **词法分析程序又称扫描程序。**
  - **任务：**读源程序的字符流、识别单词（也称单词符号，或简称符号），如标识符、整数、界限符等，并转换成内部形式。
  - **输入：**源程序中的字符流
  - **输出：**等长的内部形式，即属性字。
- **在词法分析阶段工作所依循的是语言的词法规则。**
- **描述词法规则的有效工具是正规式和有限自动机。**
- **方法：状态图；DFA；NFA**

词法分析示例

```
void main(int b,int c)
{
    int a;
    a=b+(c-2);
}#
```

词法分析后返回 (如右图)：

单词值	单词类型
1: void	VOID
1: main	MAIN
1: (	LPAREN
1: int	INT
1: b	ID
1: ,	COMMA
1: int	INT
1: c	ID
1:)	RPAREN
2: {	LBBRACKET
3: int	INT
3: a	ID

单词值	单词类型
3: ;	SEMI
4: a	ID
4: =	ASSIGN
4: b	ID
4: +	PLUS
4: (	LPAREN
4: c	ID
4: -	SUB
4: 2	NUM
4:)	RPAREN
4: ;	SEMI
5: }	RBBRACKET
5: #	ENDFILE

语法分析

- 语法分析程序又称识别程序。
  - **任务**：读入由词法分析程序识别出的符号，根据给定语法规则，识别出各个语法单位（如：短语、子句、语句、程序段、程序），并生成另一种内部表示。
  - **输入**：由词法分析程序识别出并转换的符号。
  - **输出**：另一种内部表示，如语法分析树或其它中间表示。
- 语法规则通常用**上下文无关文法**描述。
- 方法：递归子程序法、LR分析法、优先分析法。

语法分析

- 例： a=b+(c-2)

规则

<赋值语句>::=<标识符>“=”<表达式>

<表达式>::=<表达式>“+/-”<表达式>

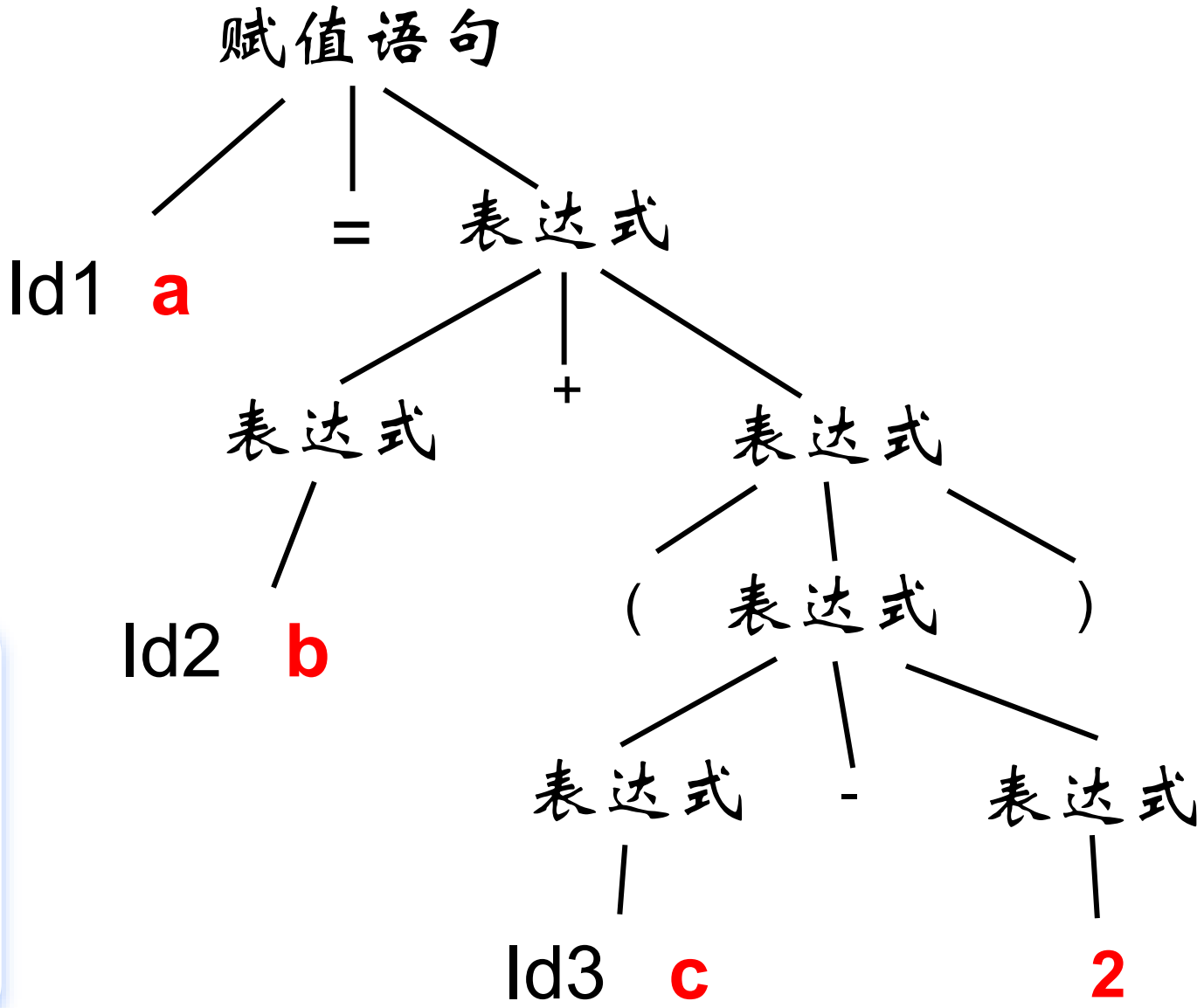
<表达式>::=“(”<表达式>“)”

<表达式>::=<标识符>

<表达式>::=<整数>

语法分析

- 例： a=b+(c-2)
  的语法树



其中：id1、id2、id3分别表示a、b、c的机器内部表示

语义分析

- **对语法分析树或其他内部中间表示进行静态语义检查，如果正确则进行另一方面的工作，即进行中间代码的翻译。**

  - **按照语法树的层次关系和先后次序，逐个语句地进行语义处理。**
  - **主要任务：** 进行类型审查，审查每个算符是否符合语言规范，不符合时应报告错误。
    - 类型匹配
    - 类型转换

语义分析示例

```
void main(int b,int c)
{
    int a;
    a=b+(c2-2);
}#
```

中间代码

- 中间代码是一种独立于具体硬件的记号系统，或者与现代计算机的指令形式有某种程度的接近，或者能比较容易地变换成机器指令。

  - **任务**：将各类语法单位，如“表达式”、“语句”、“程序”等翻译为中间代码序列。
  - **输入**：句子。
  - **输出**：中间代码序列。

- 中间代码的形式：常见的有四元式、三元式和逆波兰式等
- 方法：语义子程序；DAG图，语法制导翻译

- 四元式的形式为：

（算符，运算对象1，运算对象2，结果）；

id1　id2　id3

- 对于源程序 a = b + (c-2) 可以生成如下所示的四元式：

ID	中间代码
main	
0	- , c, 2, T0
1	+ , b, T0, T1
2	= , T1, , a

T0,T1，是临时变量

id1、id2、id3分别表示a、b、c的机器内部表示

T0,T1是临时生成的名字，表示中间运算结果。

语义分析示例

例1

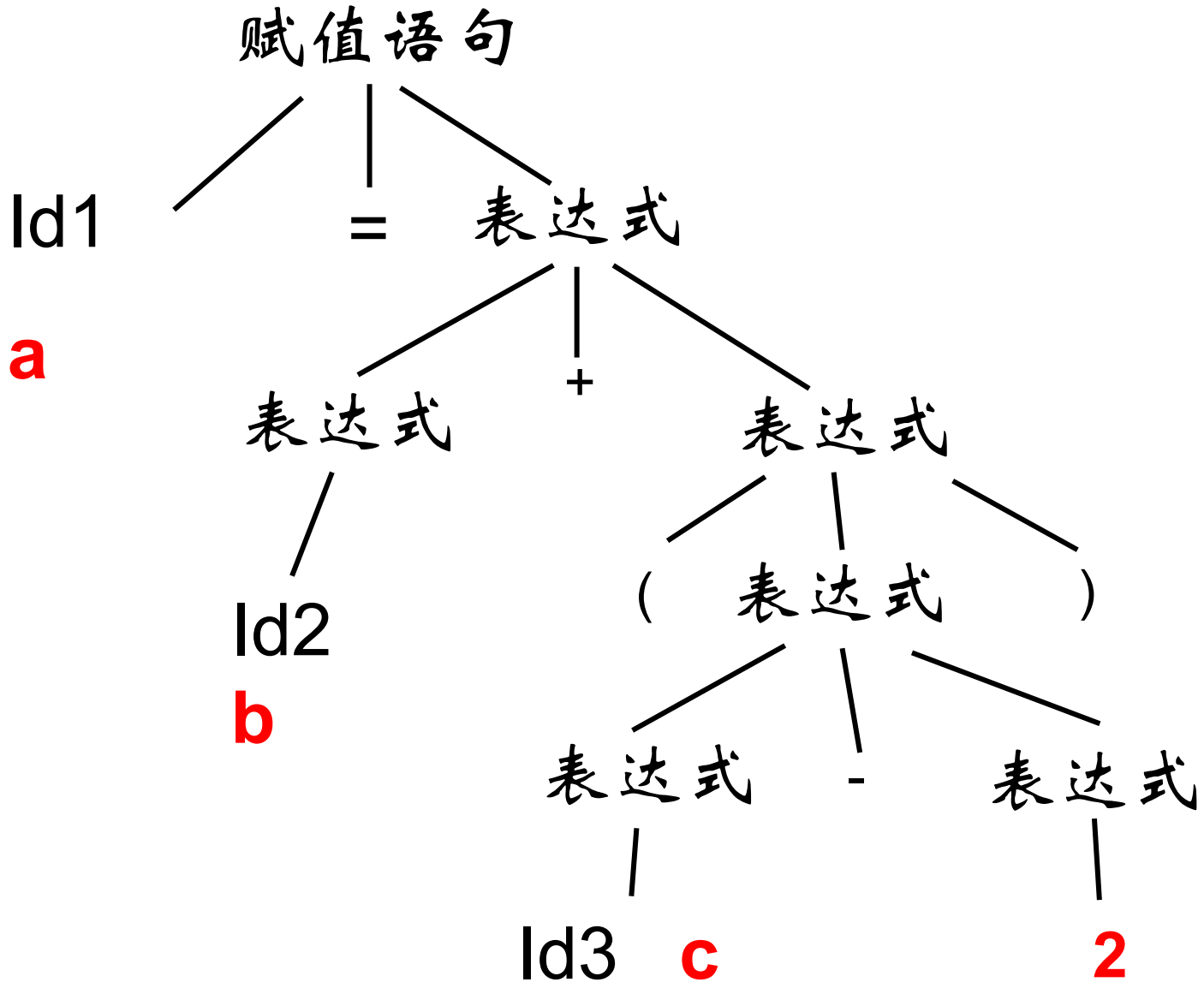
```
void main(int b,int c)
{
    int a;
    a=b+(c-2);
}#
```

语义分析

a=b+(c-2)的语法树

其中：id1、id2、id3分别表示a、b、c的机器内部表示

ID	中间代码
main	
0	- , c, 2, T0
1	+ , b, T0, T1
2	= , T1, , a



例2

- C语言的源程序

```
a = b*c + b*d
```

三地址序列（赋值语句形式的四元式）

```
(1) t1 := b*c
(2) t2 := b*d
(3) t3 := t1 + t2
(4) a := t3
```

例3

- C语言的源程序

```
if (a<=b)
    a=a-c;
c=b*c;
```

翻译成

四元式

```
100  (j<=,a,b,102)
101  (j, _, _, 104)
102  (-, a, c, t1)
103  (=, t1, _, a)
104  (*, b, c, t2)
105  (=, t2, _, c)
```

优化

- 优化的任务在于对前段产生的中间代码进行加工，把它变换成功能相同，但功效更高的优化了的中间表示代码，以期在最后阶段产生更为高效（省时间和空间）的代码
- 优化所依循的原则是程序的等价变换规则
- 其方法有：公共子表达式的提取、循环优化、删除无用代码等。

优化举例

- GCC的不同优化等级，代表着不同的优化程度。

  -O0、-O1、-O2、-O3代码优化程度依次递增

```
test.c
int main() {
    int i,j;
    i=5+6-7-4;
    while(i<1000*1000*1000)
      {
          i++;
          j=1+2+3+4;
       }
    return 0;
}
```

- 用gcc -O0 test.c -o test0生成可执行文件test0，运行时间1.367s
- 用gcc -O1 test.c -o test1生成可执行文件test1，运行时间0.241s
- 用gcc -O2 test.c -o test2生成可执行文件test2，运行时间0.000s（几乎忽略不计）

目标代码生成

- 这一阶段的任务：把中间代码（或经优化处理后）变换成特定机器上的低级语言代码。它有赖于硬件系统结构和机器指令含义。
  - 与机器相关

ID	中间代码
main	
0	- , c, 2, T0
1	+ , b, T0, T1
2	= , T1, _, a

```
MOV   EAX,    [EBP+8+0]
SUB   EAX,    2
MOV   EBX,    [EBP+8+4]
ADD   EBX,    EAX
MOV   [EBP-4], EBX
```

内容线索

- √ 什么叫编译程序
- √. 编译过程概述
- 3. 编译程序的结构
- 4. 编译程序的生成
- 5. 总结

编译程序的结构


源程序(字符流)
词法分析器
单词符号(符号流)
语法分析器
语法单位(语法树)
语义分析与中间代码产生
中间代码
优化器
中间代码
目标代码生成器
目标代码
表格管理
出错处理

表格与表格管理

- 编译程序涉及的表格有

符号表

- 在编译程序使用的表格中最重要的是符号表
  - 记录源程序中使用的名字(标识符)
  - 收集每个名字的各种属性信息
    - 类型、作用域、分配存储信息

名 字	种 类	类 型	层 次	偏移量
m	过 程		0	
a	变 量	real	1	d
b	变 量	real	1	d+4
c	变 量	real	1	d+8

出错处理

- 程序中的错误可分为**语法错误**和**语义错误**两类。
- 语法错误可在词法分析和语法分析阶段查出来。
- 例如，下列错误都属于语法错误：

(1) 缺少分隔符号：DIMENSION A(10) B(10)

(2) 保留字拼写错误：DEMENSION A(10)

(3) 括号不配对：

```
WRITE (6,10) (A (1, J), J = 1, 10), I = 1, 10)
```

此外还有多余分隔符号，无循环终结语句等等。

语义错误

- 语义错误有些可在**编译时**查出来，有些则需在**运行时**才能查出来。
- 典型的语义错误：
  - 标识符没有说明就使用；
  - 标号有引用而无定义；
  - 形式参数和实在参数结合时在类型、个数、位置等方面不一致等等；
- 这些错误可在编译时查出来，而另一些错误如下标越界、运算溢出、调用某些标准函数时自变量的值不符合要求等，则需要到程序运行时才能查出来。

- 一个好的编译器应该：

全	最大限度发现错误
准	准确指出错误的性质和发生地点
局部化	将错误的影响限制在尽可能小的范围内

若能**自动校正错误**则更好，但其**代价非常高**

编译阶段的组合

- 编译程序可以从逻辑上分成几个阶段，对于各个阶段的划分仅仅是指其逻辑结构，而在具体实现时，经常是将几个阶段组合在一起。例如，可以将各部分组合成前端和后端。

主要与源语言有关

编译过程

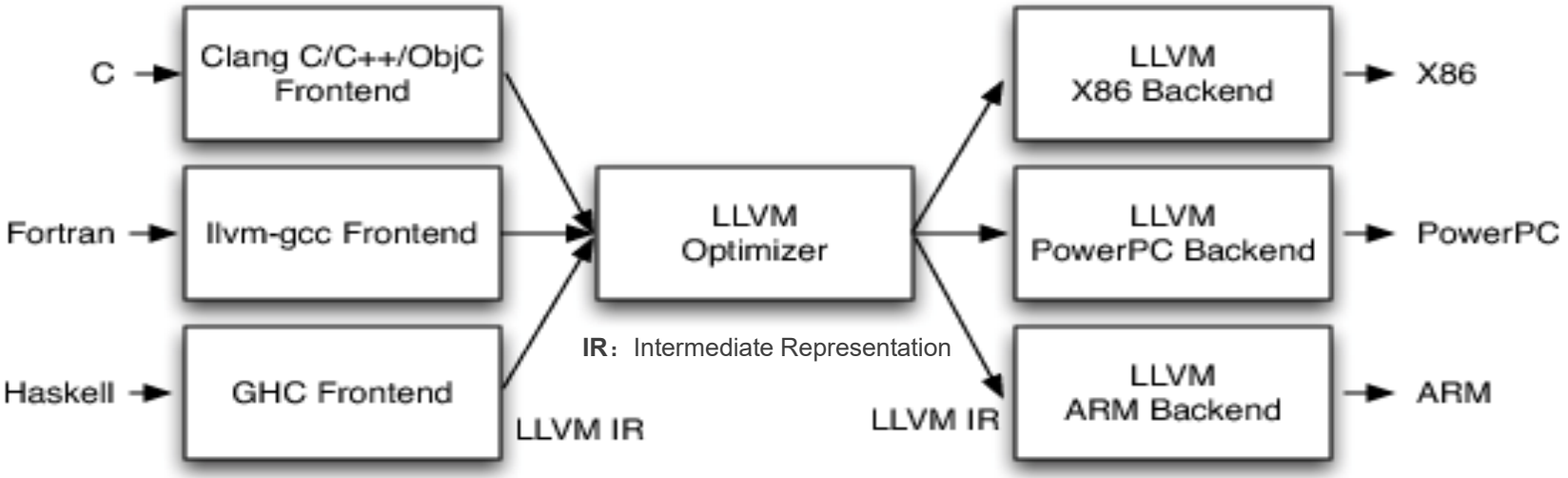
前端：词法分析、语法分析、语义分析、中间代码生成、优化工作。

后端：目标代码生成、出错处理、符号表操作。

主要与目标代码有关

编译器

- GCC（GNU Compiler Collection，GNU编译器集合）
  - 1987 年， Richard Stallman
- LLVM（low level virtual machine）
  - 2003年，Chris Lattner



并发程序的例子

```
int x; //共享变量x
void* tprocess1(void* args) {
    x := 42;
}
void* tprocess2(void* args) {
    x := x + 1;
}
int main() {
    x = 0;
    pthread_t t1;
    pthread_t t2;
    pthread_create(&t1, NULL, tprocess1, NULL);
    pthread_create(&t2, NULL, tprocess2, NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    assert(x>=42);
}
```

测试方法

执行10000次该pthread程序出现

- **9992次assert(x>=42)正常执行**
- **8次assert(x>=42)报错**

并发程序的例子

```
int x; //共享变量x
void* tprocess1(void* args) {
    x := 42;
}
void* tprocess2(void* args) {
    x := x + 1;
}
int main() {
    x = 0;
    pthread_t t1;
    pthread_t t2;
    pthread_create(&t1, NULL, tprocess1, NULL);
    pthread_create(&t2, NULL, tprocess2, NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    assert(x>=42);
}
```

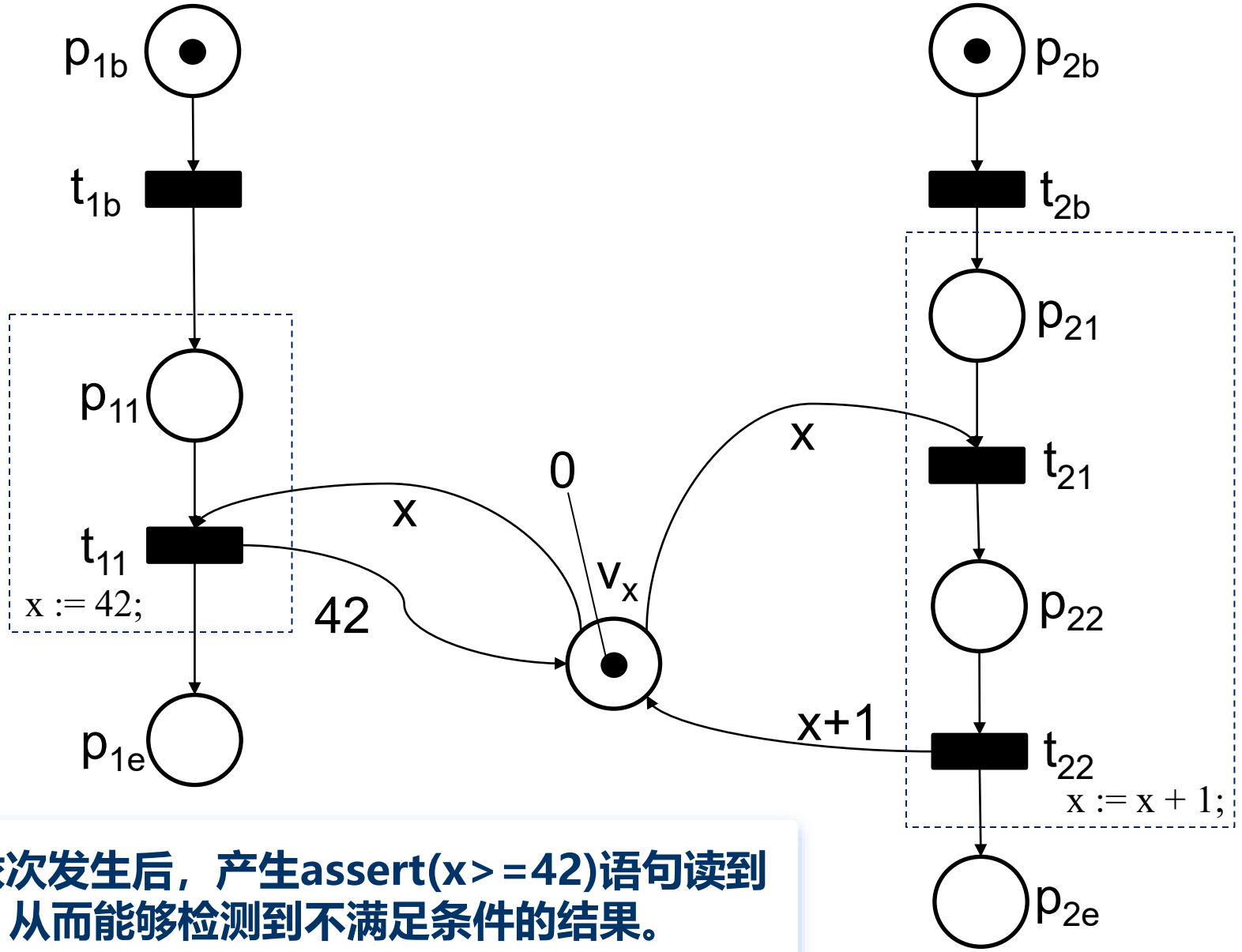
- 若语句都是具有原子性的，可能的结果如下：
  - 若x := 42;在x := x + 1;之前执行，assert(x>=42)语句读到的x为43。
  - 若x := x + 1;在x := 42;之前执行，assert (x>=42)语句读到的x为42。
- 这两种情况下，assert语句中的(x>=42)条件都是满足的。

- 但是实际上x := x + 1;在实际执行中并非原子的，其中从内存中读x的当前值的操作与将x+1的计算结果写到内存的操作可能与另一线程的写操作交织，使得assert (x>=42)语句读到的x为1，不满足条件。

形式化验证

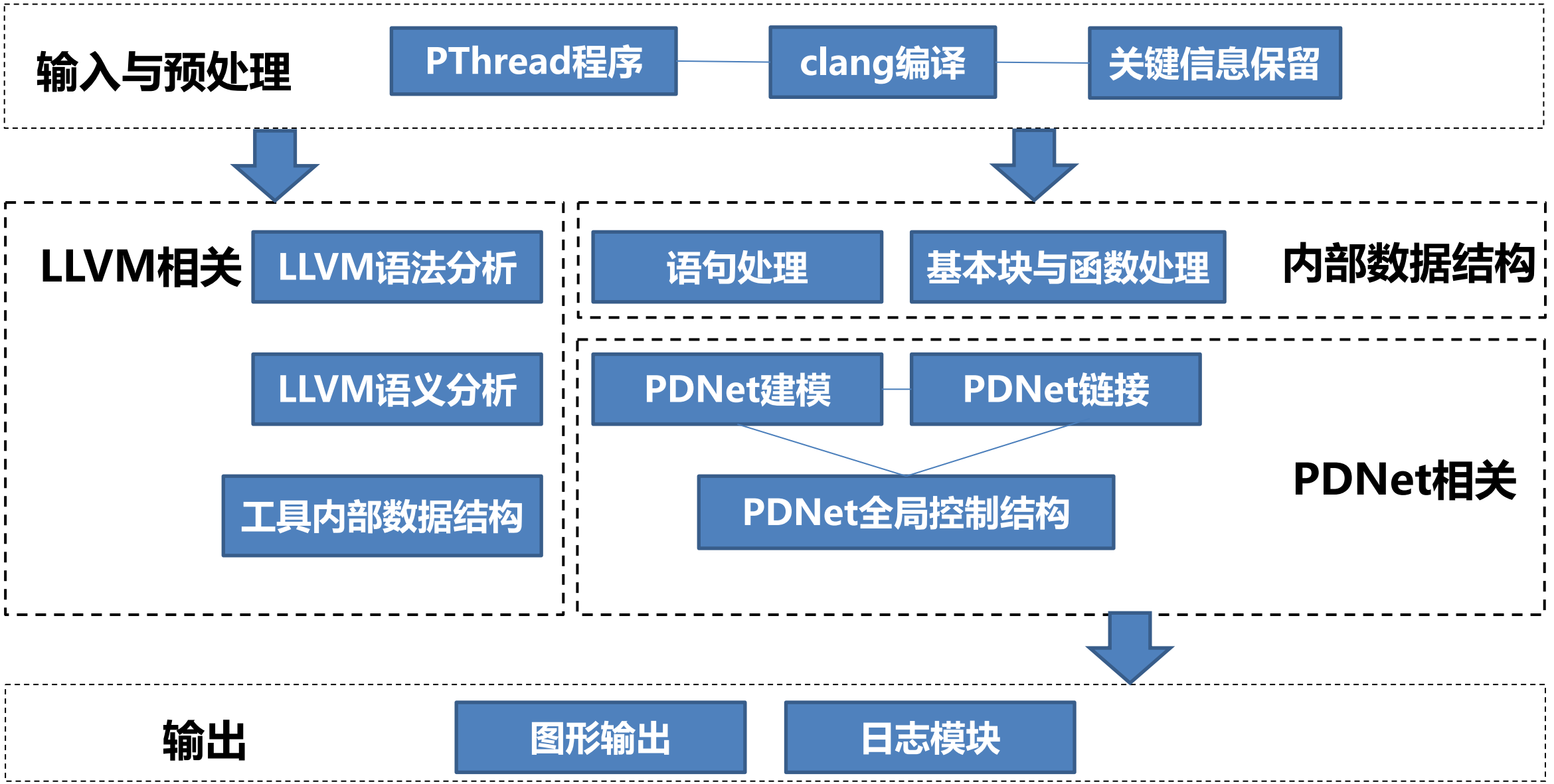
- 由于大量的不确定交织的存在，并发程序的性质验证存在更大的挑战。
- 形式化验证能够模拟并发程序所有可能的执行，考虑任意线程交错，这一现象在程序测试和定理证明中都很难捕捉到。
- 对程序员编写的代码，即由高级编程语言构成的代码，进行自动的形式验证，保证其正确性、安全性等是极其重要的。
- 自动化工具能够直接对高级编程语言进行模型检测，而无需模型检查中通常需要的专业工作是期望的。

基于Petri网的程序模型

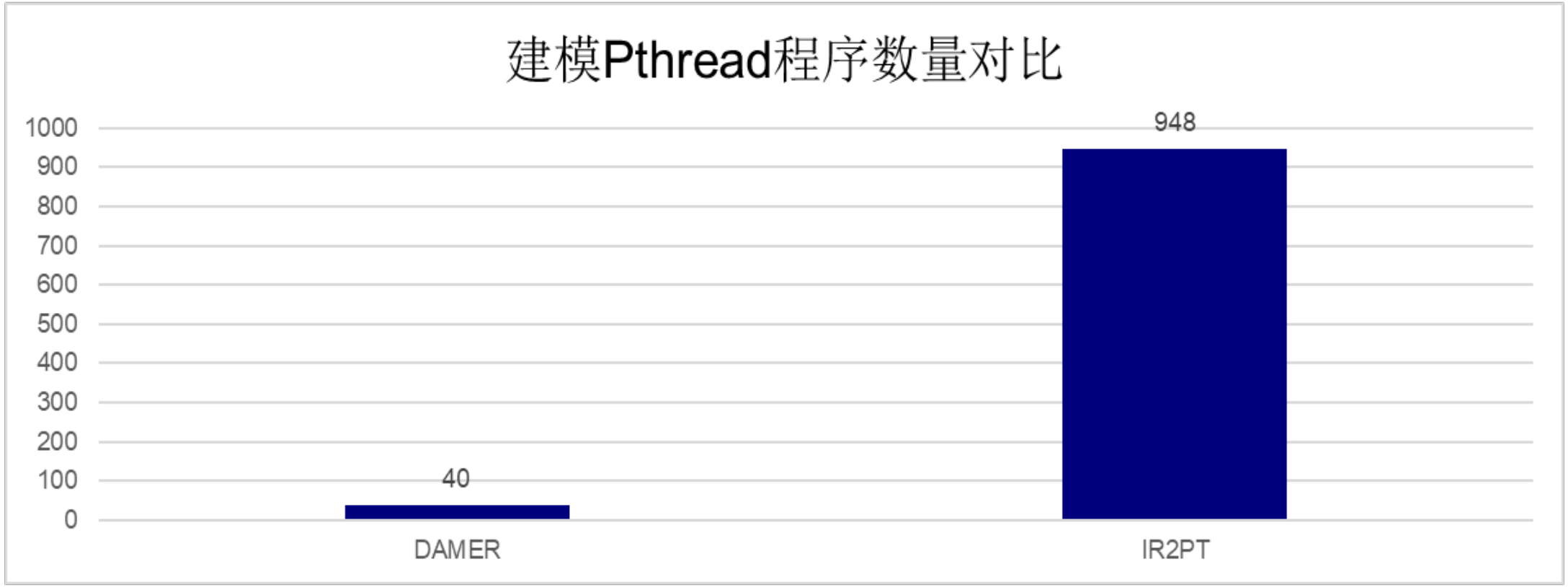


变迁t_{21} t_{11} t_{22}依次发生后，产生assert(x>=42)语句读到的x为1的情况，从而能够检测到不满足条件的结果。

基于IR的PDNet自动化建模



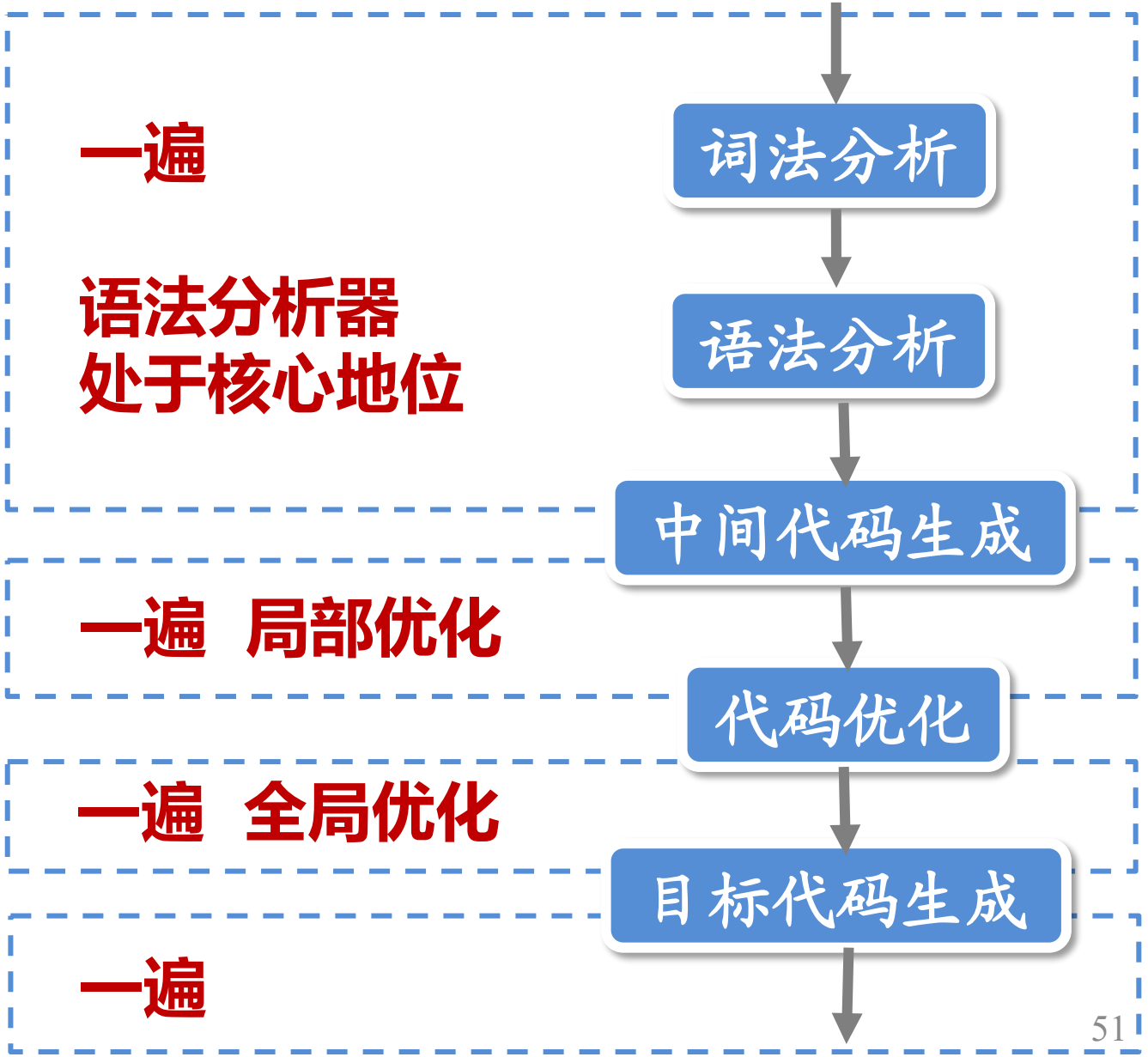
基于IR的PDNet自动化建模



实验测试集共有1044个Pthread程序测试用例，可以看到我们的方法实现测试点覆盖率达90.8%。

遍（Pass）

- 对源程序或源程序的中间结果从头到尾扫描一次，并做相关处理，生成新的中间结果或目标程序的过程。
- “遍”是处理数据的一个完整周期，每遍工作从外存上获得前一遍的中间结果（源程序），完成它所含的有关工作之后，再把结果记录于外存。



遍的次数和效果

- 一个编译程序可由一遍、两遍或多遍完成。每一遍可完成不同的阶段或多个阶段的工作

从时间和空间角度看

- 多遍编译 —— 少占内存，多耗时间
- 一遍编译 —— 多占内存，少耗时间

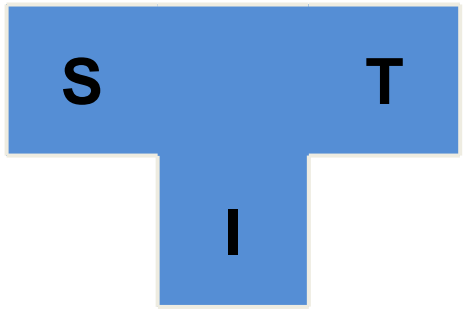
内容线索

- √ 什么叫编译程序
- √. 编译过程概述
- √. 编译程序的结构
- 4. 编译程序的生成
- 5. 总结

编译程序实现语言

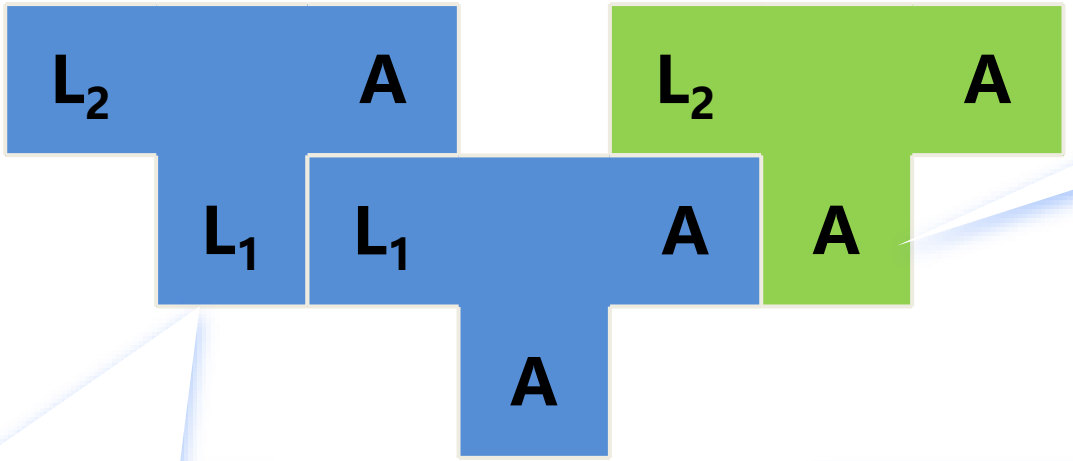
- 机器语言
- 汇编语言
  - 充分发挥各种不同硬件系统的效率
  - 满足各种不同的具体要求
- 高级语言
  - 大大节省程序设计的时间
  - 构造出来的编译程序易于阅读、维护和移植

T形图



- **其中：**
  - S：源语言(程序)，Source language(program)
  - T：目标语言(程序), target/object language(program)
  - I：实现语言, implementation language

示 例

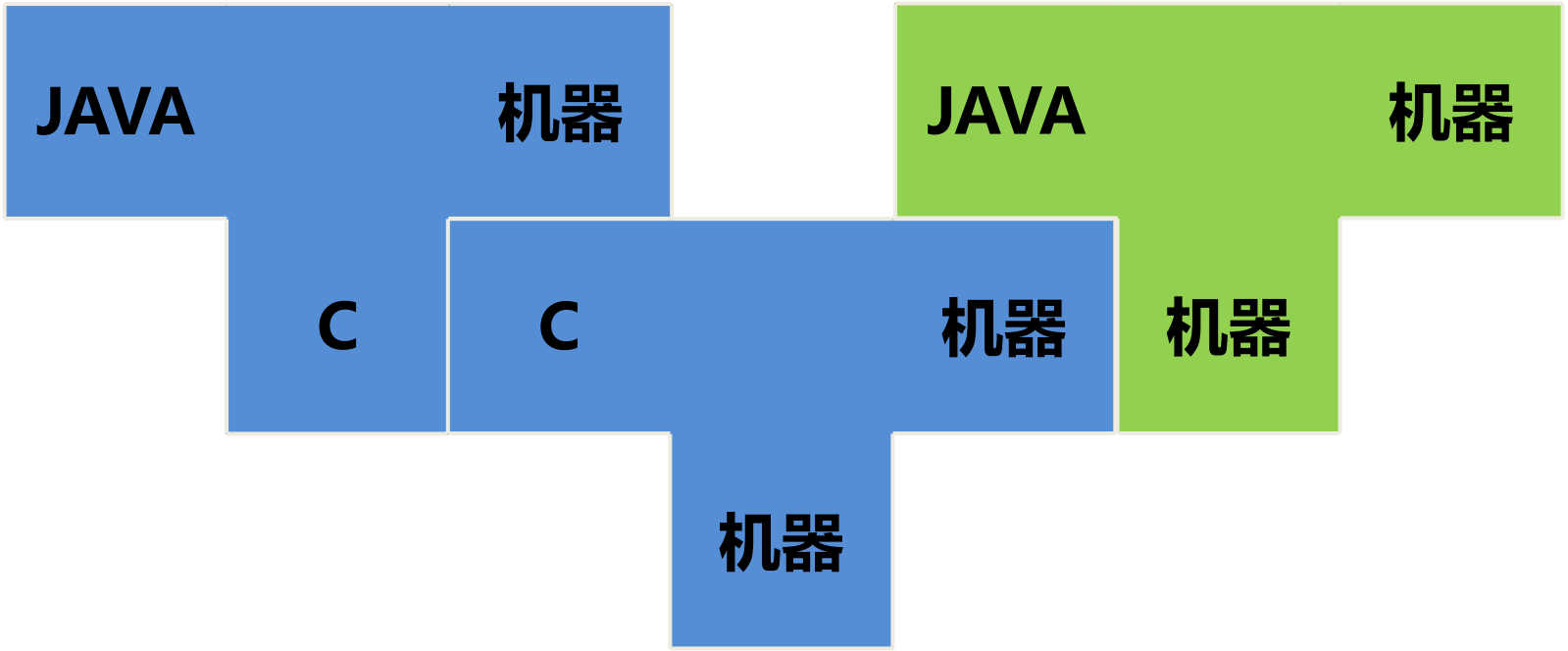


A机器代码实现的L_2编译程序

用L_1语言编写另一种高级语言L_2的编译程序

A机器上已有一个用A机器码实现的某高级语言L_1的编译程序

示例

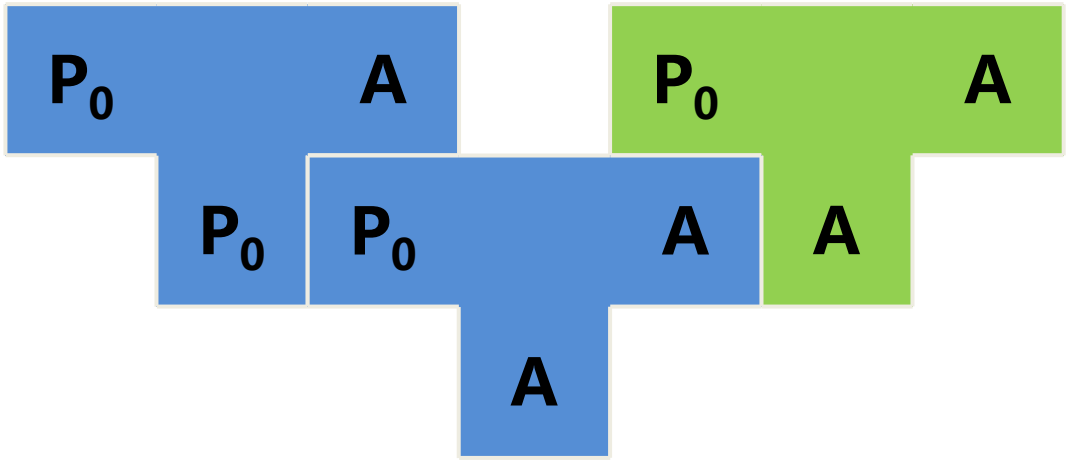


编译程序的生成技术

- 自编译
- 交叉编译
- 自展
- 移植

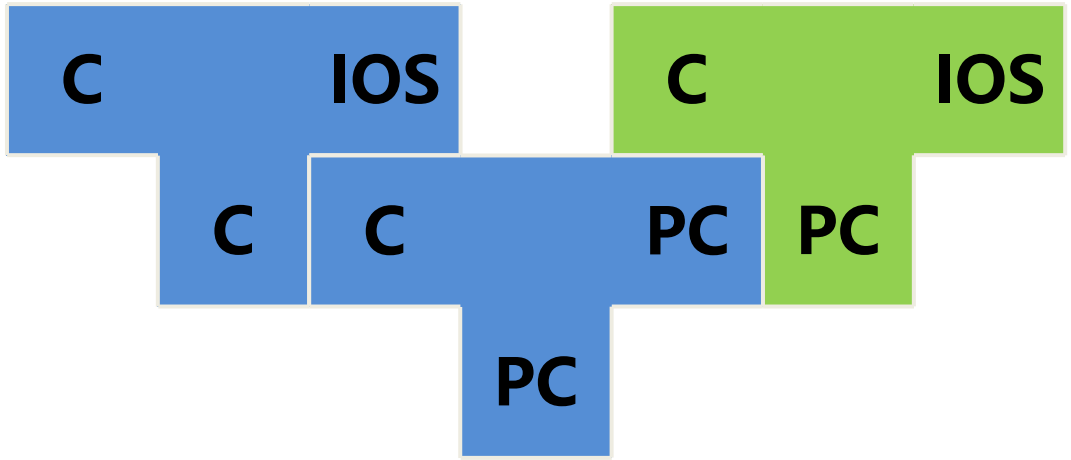
自编译

- 用某种高级语言书写自己的编译程序称为自编译。
  - 例如,假定A机器上已有一个PASCAL语言编译程序，则可用PASCAL语言编写一个功能更强的PASCAL语言编译程序，然后借助于原有的编译程序对新编写的PASCAL编译程序进行编译，从而得到一个能在A机器上运行的功能更强的PASCAL编译程序。



交叉编译

- 用x机器上的编译程序产生可在y机器上运行的目标代码称为交叉编译。
  - 例如，若PC机器上已有C语言编译程序,则可用PC机器中的C语言书写一个编译程序，该编译程序的源程序是C语言程序,而产生的目标程序则是基于IOS终端的，即产生在IOS终端上执行的低级语言程序。



- 适用于嵌入式系统、移动端开发
- 上述两种方法假定已有一个编译程序，若没有，则可采用自展或移植法。

自展

- 首先确定一个非常简单的核心语言L0，然后用机器语言或汇编语言书写出其编译程序T0；
- 到L1，L0 ⊂ L1，并用L0编写L1的编译程序T1(自编译)；
- 然后再把语言L1扩充为L2，L1 ⊂ L2，并用L1编写L2的编译程序T2；

……

这样不断扩展，直到完成所要求的编译程序为止。

移植

- 移植是指A机器上的某种高级语言的编译程序稍加改动后能在B机器上运行。
- 一个程序若能较易地从A机移到B机上运行，则称该程序可移植。

自动编译

- 自动生成编译程序的软件工具，只要把源程序的定义及机器语言的描述输入到该软件中，就能自动生成该语言的编译程序。

- 目前的编译程序自动生成系统，如LEX和YACC等。

编译程序的分类

- 依据编译程序的不同用途和侧重，可分类为:
  - 诊断型编译程序
  - 优化型编译程序
  - 交叉型编译程序
  - 可变目标型编译程序

编译技术的发展

- 在1954年至1957年期间，IBM的John Backus带领的一个研究小组对FORTRAN语言及其编译器的开发
  - 将算术公式翻译成机器代码。
- Noam Chomsky 开始了自然语言结构的研究。他的发现最终使得编译器结构异常简单
- 70年代，有穷自动机和形式语言的研究，促进了编译器的发展
  - Steve Johnson为Unix系统编写了Yacc （yet another compiler-compiler）。
  - Mike Lesk为Unix系统开发的 Lex

编译技术的发展

1990年代：开源工具链与跨平台编译的兴起

- GCC（GNU Compiler Collection）
  - 首个开源、多语言（C/C++等）、多平台支持的编译器。
  - 插件化架构，奠定可扩展编译框架基础。
  - 推动Linux生态崛起（如Linux内核依赖GCC编译）。
- LLVM（Low Level Virtual Machine）
  - 模块化设计（前端Clang、中端优化器、后端代码生成）。
  - 解耦编译器开发，加速优化算法迭代（如苹果Swift语言依赖LLVM）。
  - 支持JIT编译，为后续动态语言优化铺路。
- Java虚拟机（JVM）
  - “Write Once, Run Anywhere” 字节码（.class文件）机制。
  - JIT技术初现：HotSpot虚拟机动态优化热点代码（解释执行→即时编译）。

编译技术的发展

21世纪——性能优化与AI赋能

- JIT（即时编译）与动态语言优化
  - V8引擎（2008）：
    - 关键技术：隐藏类（Hidden Class）、内联缓存（Inline Cache）
    - 影响：Chrome浏览器、Node.js生态的基石
  - 其他案例：PyPy（Python JIT）、LuaJIT
- AI辅助编译
  - 机器学习预测代码优化策略（如循环展开、向量化）
  - 自动调优编译器参数（如AutoTVM优化深度学习算子）

国内编译器研究历史

- **20世纪60年代**
  - 董韫美院士、杨芙清院士分别在中科院和北大领导研究组开发编译器，面向的高级语言是 ALGOL和FORTRAN，目标机是国产机。
  - 在改革开放前，由于国家需要，中科院、国防科大、江南计算所、北大等单位一直在研制国产计算机，包括大型机和高性能计算机（如向量机、并行机），相应的也在研制高级语言编译器。
  - 中科院计算所以董韫美院士领导的研究组先后开发了119机、109机的类 ALGOL语言编译器 BCY。国防科大开发了向量编译器和向量识别器。

国内编译器研究历史

- **20世纪70年代**
  - **70年代中科院计算所张兆庆教授研究组（ACTGroup）开始在国产机上研制FORTRAN语言编译器，先后参与了众多的院级和国家级科研攻关项目，主持开发了013，757，KJ8920等国产大型机系统中的FORTRAN语言编译器，所研制的编译器支持了数百万行应用软件的运行。**

国内编译器研究历史

- **20世纪90年代**
  - **90年代ACTGroup承担科学院重大项目，国家攻关项目，863项目，以及国际合作项目，先后开发了共享内存多处理机的并行识别器，分布式内存多处理机的并行识别器， SIMD芯片和VLIW芯片的并行优化 C编译器。将编译技术与图形学结合，ACTGroup还推出了集成化、可视化的并行编程环境。**
  - **ACTGroup在先进编译技术和并行编程环境方面的研究工作获国内外专家高度评价，国际著名学者评价此研究组居编译领域的世界先进行列。**

国内编译器研究历史

- **20世纪90年代**
  - 90年代，国内主要以研制并行机为主，相应的并行编译器研制也在国内开展起来。
  - 清华大学在软流水优化技术上做了很优秀的研究工作，开发了交互式并行化系统 TIPSExplorer。
  - 北京大学在HPF（High Performance Fortran）编译器方面做了多年工作，取得很好的研究成果。
  - 此外，国防科大、江南计算所等单位也都有从事并行编译技术研究。
  - 随着芯片研制，国内还有若干单位也在开展基于GCC生成面向特定芯片的编译器工作。

国内编译器发展现状

■ 热点产品——华为方舟编译器

- **是华为公司专门为软件厂商研发的统一编程平台，**包含编译器、工具链、运行时等关键部件。该编译器支持多种编程语言、多种芯片平台的联合编译与运行，能够有效解决安卓程序“边解释边执行”的低效率问题。
- **是华为推出的首个完全自主研发的编译器平台**，于2019年8月31日正式上线并部分开源。
- 目前仍在持续更新。

国内编译器发展现状

■ 热点产品——华为毕昇编译器

- **是华为公司针对鲲鹏硬件平台的高性能编译器，**是一款高性能、高可信及易扩展的编译器工具链，增强和引入了多种编译优化技术，支持C/C++/Fortran等编程语言。提供深度优化的编译技术，增强多核并行化，自动矢量化等，大幅提升指令和数据吞吐量，针对某些应用场景进行优化，尤其在高性能计算（HPC）场景下能获得更优的性能收益。
- **基于开源LLVM 10.0.1版本开发，并进行了优化和改进，**于2020年9月30日正式上线。
- 当前最新版本为 2.1.0，于 2021 年 12 月 30 日发布。

国内编译器发展现状

■ 热点产品——腾讯Yadcc编译器

- **是腾讯公司研发并开源的C++分布式编译系统，**针对实际的工业生产环境进行了性能、可靠性、易用性等方面优化，可以利用几百乃至1000+核同时编译，大大加快构建速度。通过创建符号链接至Yadcc客户端以截获编译器调用，再将任务分发至分布式编译机编译，降低本地负载。用于支撑腾讯广告的日常开发及流水线，目前在腾讯1700+核的集群中每天编译300,0000+个目标文件，产出约3~5TB。
- **调研学习业界开源编译加速系统后，根据实际工业生产场景自行设计，**于2021年6月正式对外开源。

Tencent 腾讯

国内编译器发展现状

- **热点产品——阿里BladeDISC编译器**
  - **是阿里巴巴集团自主研发并开源的深度学习编译器**，旨在为用户提供通用、透明、易用的深度学习性能优化能力，支持主流的机器学习框架和主流硬件。在架构层面根本地解决了深度学习领域具有动态尺寸 Tensor 的优化难题，提供了多种灵活的部署方案，包括插件模式集成、独立运行与 AOT 编译。是业内较早投入实际业务应用的编译器。
  - **基于MLIR编译器框架开发**，于2022年1月28日正式开源。

Alibaba Group 阿里巴巴集团

热点产品对比

比较	华为方舟	华为毕昇	腾讯Yadcc	阿里BladeDISC
简介	为软件厂商研发的统一编程平台	针对鲲鹏硬件平台的高性能编译器	支撑腾讯广告的日常开发及流水线的编译器	投入公司实际业务应用的自用编译器
特色	能够有效解决安卓程序“边解释边执行”的低效率问题	针对高性能计算场景优化，能获得更优的性能收益	针对分布式工业生产环境进行了性能、可靠性、易用性等方面优化	在架构层面根本地解决了深度学习领域具有动态尺寸 Tensor 的优化难题
基础框架	完全自研	基于LLVM	自行设计	基于MLIR
开源时间	2019.8.31	2020.9.30	2021.6	2022.1.28

编译器发展趋势：智能化、安全与跨域融合

1. AI驱动的编译优化

- **智能决策**：
  - 机器学习预测最佳优化策略（如循环展开、向量化）
  - 案例：Google MLIR、华为MindSpore自动算子优化
- **自适应编译**：
  - 动态感知硬件状态（如功耗、温度）实时调整代码生成

2. 跨平台与异构计算支持

- **统一中间表示（IR）：**
  - LLVM/MLIR支持多硬件（CPU/GPU/TPU/量子芯片）
    案例：苹果M1芯片（ARM+GPU+NPU统一编译）
- **RISC-V生态崛起：**
  - 开源指令集催生定制化编译器（如中科院香山处理器工具链）

3. 绿色编译与能效优化

- **功耗感知编译：**
  - 生成低功耗指令序列（如ARM定制指令集优化）
- **碳足迹追踪：**
  - 编译阶段预估代码执行能耗（如阿里云绿色计算工具链）

4. 安全增强与形式化验证

- **内存安全：**
  - Rust风格所有权模型集成到编译器（如微软Verona项目）
- **抗攻击编译：**
  - 自动插入防侧信道攻击代码（如AWS Nitro Enclaves安全验证）
- **形式化证明：**
  - 数学方法验证编译器正确性（如CompCert C已验证编译器）

5. 开发者体验革新

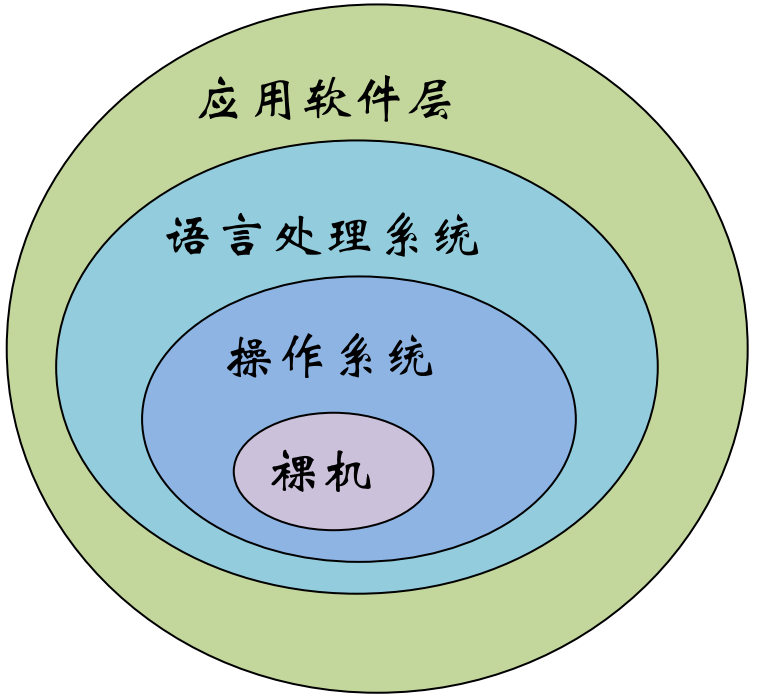
- **交互式编译：**
  - 实时编译反馈（VS Code插件提示优化建议）
- **低代码编译：**
  - 自然语言生成代码（GitHub Copilot + 编译器协同）

内容线索

- √ 什么叫编译程序
- √. 编译过程概述
- √. 编译程序的结构
- √. 编译程序的生成
- 5. 总结

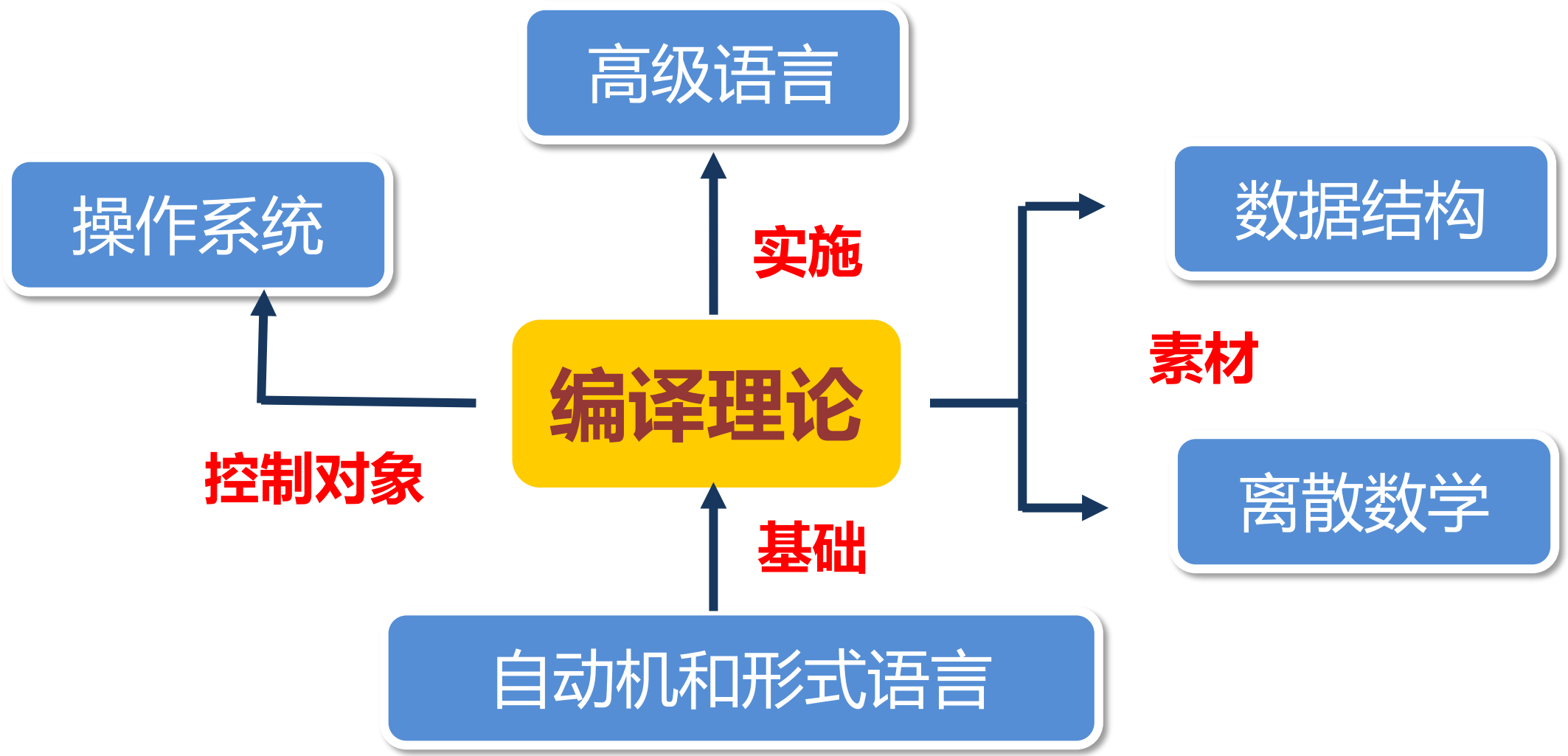
编译程序在计算机系统中的地位

- **编译系统是一种系统软件**
  - **软件：计算机系统中的程序及其文档。**
  - **系统软件：居于计算机系统中最靠近硬件的一层，其他软件一般都通过系统软件发挥作用。和具体的应用领域无关，如编译系统和操作系统等。**
    - **语言处理系统：把软件语言书写的各种程序处理成可在计算机上执行的程序，如编译系统。**



计算机系统

编译理论与其他课程的关系



为什么要学习编译原理

- **必修主干课程**，操作系统和编译系统构成程序设计者与计算机之间的基本界面。
- 通过学习编译原理可以更好地**理解程序语言的内部机制**，从而更好地**理解和运用程序设计语言**。
- 能运用编译程序构造的原理和技术完成相关软件工具的设计和开发工作。

教材及参考书

- 教材
  - 《程序设计语言编译原理》（第3版）， 陈火旺等著，国防工业出版社。
- 参考书
  - Aho, Lam, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley.（龙书）
  - 《Modern Compiler Implementation in C》（《现代编译原理C语言描述》）Andrew W.Appel著（赵克佳等译），人民邮电出版社。
  - 《编译原理》，吕映芝、张素琴，清华大学出版社。
  - 《编译方法》，胡笔蕊等，电子工业出版社。
  - ……

工程认证要求

支撑毕业要求指标点	课程如何支撑毕业要求
1-4具备对复杂工程问题进行计算机求解的专业知识	通过课程知识的学习，使学生深入理解程序语言编译的原理，掌握编译程序设计的基本知识和基本方法，为今后解决复杂的计算机系统设计和实现奠定扎实的专业知识。
2-3具备对复杂工程问题进行分析和求解的能力	通过课程学习，结合词法分析器设计、语法分析器设计等大作业，培养学生运用有限状态自动机、BNF范式、语法分析算法等解决程序语言编译具体问题的能力。
3-2熟练掌握对复杂计算机系统进行分析和总体设计的方法	通过词法分析器设计、语法分析器设计等大作业，培养学生针对实际设计任务开展方案设计和系统实现的能力。
10-1具备撰写复杂工程方案技术报告和设计文稿的能力	通过要求学生撰写课程实验设计报告，培养学生撰写技术报告和方案的能力。

要求及学习方法

课程特点：理论性强，算法复杂

- 平时
  - 出勤
  - 听课+思考
  - 认真完成课堂和课后作业
- 课堂讨论
  - 积极参与课堂讨论
- 大作业
- 期末: 闭卷笔试

考试

- 出勤及课堂表现：10%
- 平时作业：10%
- 大作业1（词法和语法分析）：10%
- 大作业2（中间代码生成）：10%
- 期末考试：60%

本书结构

- 第一章 引论
- 第二章 高级语言及其语法描述
- 第三章 词法分析 ★
- 第四章 语法分析——自上而下分析 ★
- 第五章 语法分析——自下而上分析 ★
- 第六章 属性文法和语法制导翻译 ☆
- 第七章 语义分析和中间代码产生 ★
- 第八章 符号表
- 第九章 运行时存储空间组织 ★
- 第十章 优化 ☆
- 第十一章 目标代码生成 ☆

作业题

- 高级程序设计语言有哪两种执行方式？其特点是什么？
- 什么是编译程序？它的功能是什么？
- 一个编译程序由哪几个阶段构成？

思考题

- 比较你所使用过的一些语言的编译程序：他们的编译速度，出错信息的可读性，有无优化选择等？