Introduction to Compilation Principle

赵帅

计算机学院

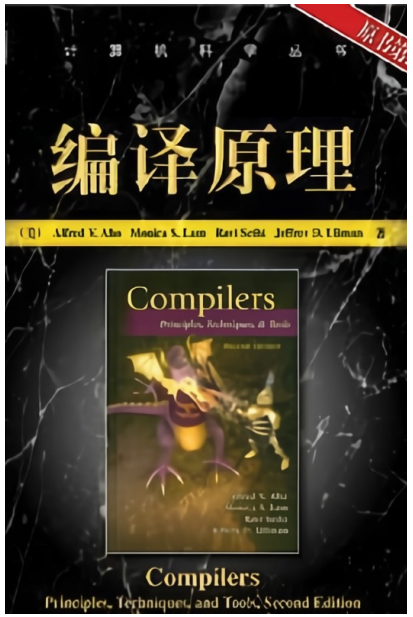
中山大学

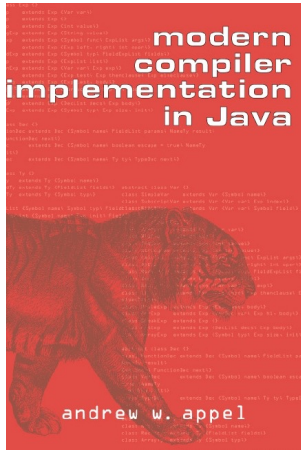
The Module[课程介绍]

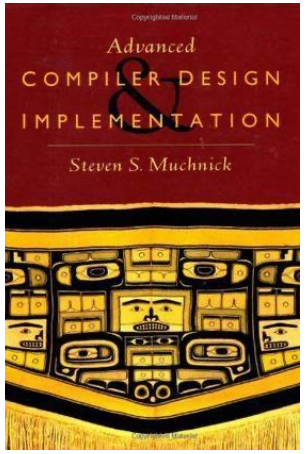
- 年级专业：
  - 21级计科(系统结构)、21级计科(人工大数据)
- 先修课程：
  - 计算机组成原理、汇编语言、数据结构、编程语言设计等
- 理论课：
  - 核心问题：**一个高级语言编写的程序，最后如何在计算机上运行？**
  - 讲解编译原理中的核心理论与技术方法
    - 词法分析、语法分析、语义分析、代码生成、代码优化等
- 实验课：
  - 独立课程、配合理论课协同进行
  - 应用编译原理中的技术方法
    - Java编程、语言处理、逆向工程工具等

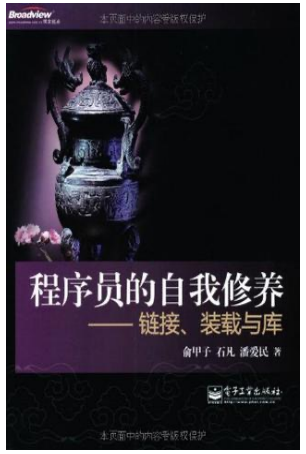
Textbook & Materials[教材资料]

- 主要教材:
  - 《Compilers: Principles, Techniques, and Tools》, 第二版
  - 课程将覆盖章节1-6与附件A
- 其他参考材料:
  - 《Modern compiler implementation in Java》
  - 《Advanced Compiler Design and Implementation》
  - 《程序员的自我修养 – 链接、装载与库》等等









Module Time & Location[时间地点]

- 理论课
  - 排课:
    - 周二: 第 1-9 周
    - 周四: 第 1-18 周
  - 时间:
    - 第 5 节: 14:20 – 15:05
    - 第 6 节: 15:15 – 16:00
  - 地点:
    - 教学大楼C105
- 实验课
  - 排课:
    - 周四: 第 1-18 周
  - 时间:
    - 第 7 节: 16:30 – 17:15
    - 第 8 节: 17:25 – 18:10
  - 地点:
    - 实验中心大楼B203

Schedule[进度安排]

周次	课程内容	周次	课程内容
第1周 (2.27 & 2.29)	周二：课程介绍、周四：词法分析	第11周 (5.9)	周四：中间代码
第2周 (3.5 & 3.7)	周二：词法分析、周四：词法分析	第12周 (5.16)	周四：代码优化
第3周 (3.12 & 3.14)	周二：词法分析、周四：语法分析	第13周 (5.23)	周四：代码优化
第4周 (3.19 & 3.21)	周二：语法分析、周四：语法分析	第14周 (5.30)	周四：代码优化
第5周 (3.26 & 3.28)	周二：语法分析、周四：语法分析	第15周 (6.6)	周四：目标代码生成
第6周 (4.2 & 4.4)	周二：语法分析、周四：语法分析	第16周 (6.13)	周四：目标代码生成
第7周 (4.9 & 4.11)	周二：语法分析、周四：语义分析	第17周 (6.20)	周四：目标代码生成
第8周 (4.16 & 4.18)	周二：语义分析、周四：语义分析	第18周 (6.27)	周四：课程复习
第9周 (4.23 & 4.25)	周二：语义分析、周四：中间代码	第19周	考试周
第10周 (5.2)	周四：中间代码	第20周	考试周

词法分析 (4)

语法分析 (8)

语义分析 (4)

中间代码 (3)

代码优化 (3)

代码生成 (3)

Subject to changes

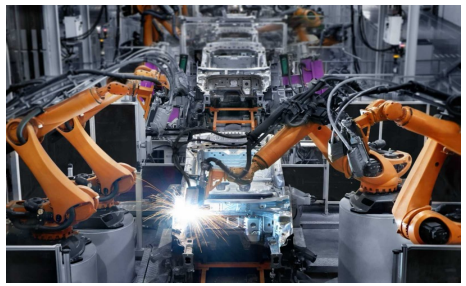
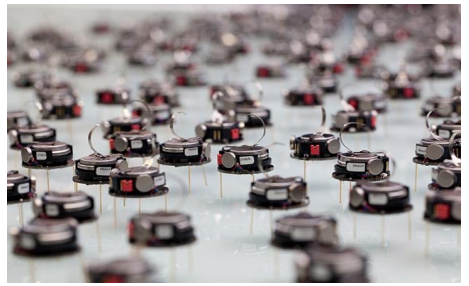
Grading[成绩构成]

- 学分与学时：
  - 理论课：3学分、54学时
  - 实验课：1学分、36学时
- 理论课分数：
  - （**40%**）课后作业
  - （**60%**）闭卷考试
- 实验课分数：
  - （**10%**）实验一：使用Java实现一个所得税计算器
  - （**25%**）实验二：改进一个已有的语言处理程序
  - （**30%**）实验三：实现一个基于表达式的计算器
  - （**35%**）实验四：开发一个程序逆向工程工具

The Instructor[关于教师]

- 教育经历：
  - 2013.10 - 2018.08，硕&博，约克大学
  - 2008.09 – 2012.07，本科，西安工业大学
- 工作经历：
  - 2022.09 - 至今，中山大学，副教授
  - 2018.10 - 2022.09，博士后，约克大学
- 研究方向：嵌入式与实时系统，操作系统
  - 任务调度与分配
  - 系统资源管理
  - 软硬件协同设计
  - 安全关键编程
- 课程：
  - 本科《编译原理》、《编译器构造实验》
  - 研究生《嵌入式系统》

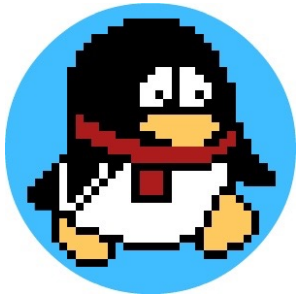


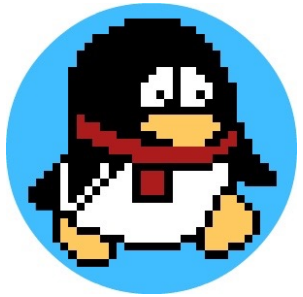
https://www.tiempo-secure.com/blog/
https://en.wikipedia.org/wiki/Regulation_of_unmanned_aerial_vehicles
https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20171023-1
https://web.uettaxila.edu.pk/cped/SwarmRoboticsLab.asp

Teaching Assistants[助教]

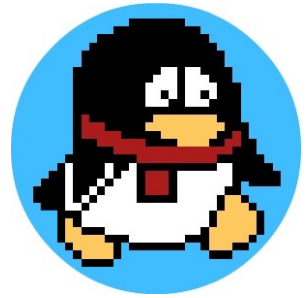
23级硕士研究生
苏若娴（组长）
surx3@mail2.sysu.edu.cn



23级硕士研究生
陈振东
chenzhd29@mail2.sysu.edu.cn



23级硕士研究生
易辉轩
yihx3@mail2.sysu.edu.cn



23级硕士研究生
陈浩然
chenhr56@mail2.sysu.edu.cn

24级硕士研究生
陈炜琰
chenwy93@mail2.sysu.edu.cn

24级硕士研究生
张美璇
zhangmx67@mail2.sysu.edu.cn

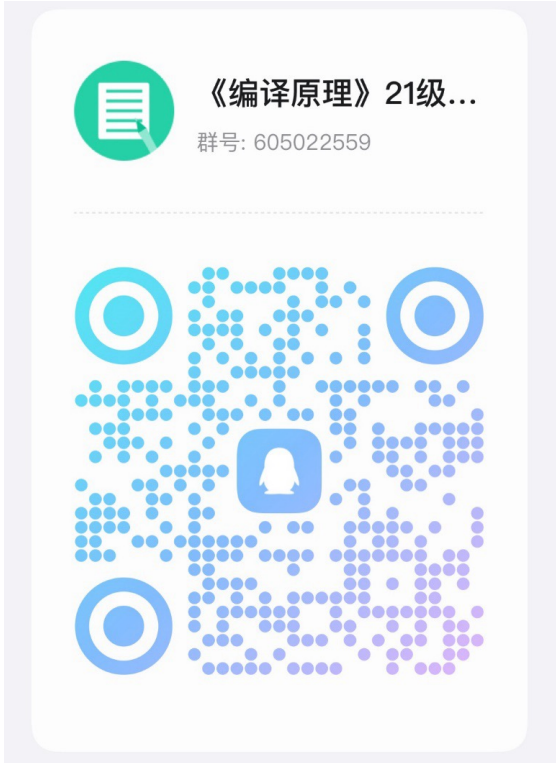
24级硕士研究生
姜婕妤
2416980629@qq.com

Contacts[联系方式]

- 任课教师
  - 赵帅，计算机学院，无人系统研究所

- 课程主页：
  - https://rtscompiler.github.io/dcs290/2024Spring/
  - 教材，资料
  - 作业，公告

- 交流方式
  - 邮件：zhaosh56@mail.sysu.edu.cn
  - **QQ群：605 022 559**（进群更改名称：学号+姓名）



The Laboratory[关于实验室]

- 常年招收硕/博士研究生，也欢迎感兴趣的本科生加入
  - 研究招收方向：嵌入式与实时操作系统、实时通信、体系结构
- **课题**
  - 复杂系统设计与分析：《高性能实时操作系统关键技术研究》
  - 系统资源共享与管理：《面向多核混合关键实时系统的高性能资源共享技术研究》
  - 共同感兴趣的研究题目
- **竞赛**
  - 中国机器人及人工智能大赛 **[组队中]**
  - 全国大学生计算机系统能力大赛 — 操作系统设计赛 **[组队中]**
  - “挑战杯” 大学生创业大赛
- 去年指导的比赛
  - 2023年中国计算机学会（CCF）移动机器人抓取和导航挑战赛 全国一等奖
  - 2023年全国大学生计算机系统能力大赛操作系统功能挑战赛 全国二等奖

中国机器人及人工智能大赛(CRAIC)官方公众号

CSCC 全国大学生计算机系统能力大赛

挑战杯 全国大学生课外学术科技作品竞赛

What is a Compilation[什么是编译]

- 问题：大家写程序都用过什么语言？
  - 高级语言C、C++、Java、Python、Ruby等等
  - **系统工程师**可以理解、使用这些语言
  - 专注于程序功能，不过多考虑计算机底层运行细节
- 计算机理解这些高级编程语言吗？
  - **计算机**只懂得机器语言，即二进制码
  - 不能直接理解并执行高级语言
- 如何让计算机执行高级语言编写的程序？
  - 需要**将高级语言翻译为机器码**
  - 兼具查错功能、性能、移植性等

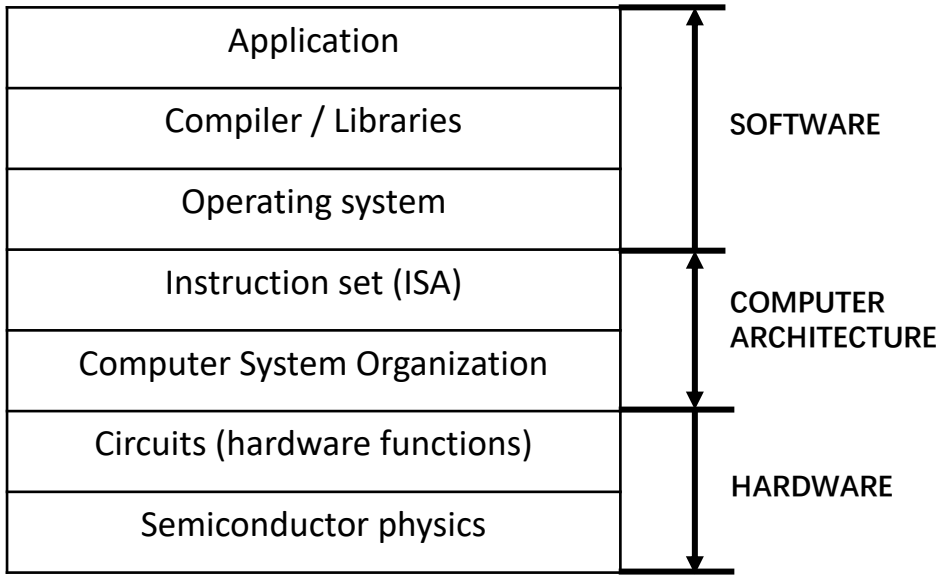
```
while (y<z){
    int x = a + b;
    y += x;
}
```

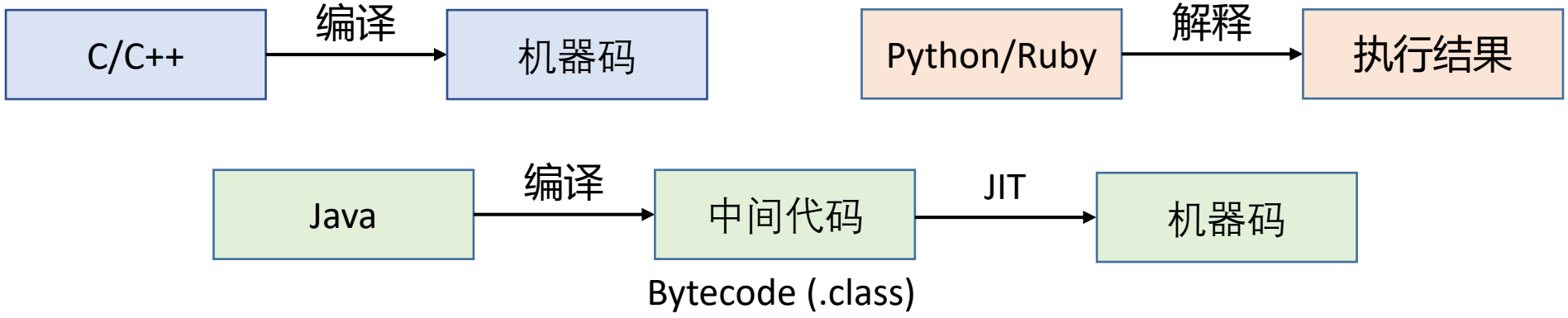
HOW?

```
   eb 06       jmp   L2
   8b 55 f4    mov   edx, DWORD PTR [rbp-12]
   8b 45 f0    mov   eax, DWORD PTR [rbp-16]
   01 d0       add   eax, edx
   89 45 ec    mov   DWORD PTR [rbp-20], eax
L3:8b 45 ec    mov   eax, DWORD PTR [rbp-20]
   01 45 fc    add   DWORD PTR [rbp-4], eax
L2:8b 45 fc    mov   eax, DWORD PTR [rbp-4]
   3b 45 f8    cmp   eax, DWORD PTR [rbp-8]
   7c f2       jl    L3
```

What is a Compilation[什么是编译]

- 不同编程语言有不同的翻译方式
  - **编译**：通常针对较为“底层”的语言
    - C、C++
    - 执行前全文翻译（Ahead of Time, AOT）
  - **解释**：通常针对较更“上层”的语言
    - Python、Ruby
    - 边翻译边执行（During Execution）
  - 混合：编译 + 即时编译（Just-In-Time, JIT）
    - Java



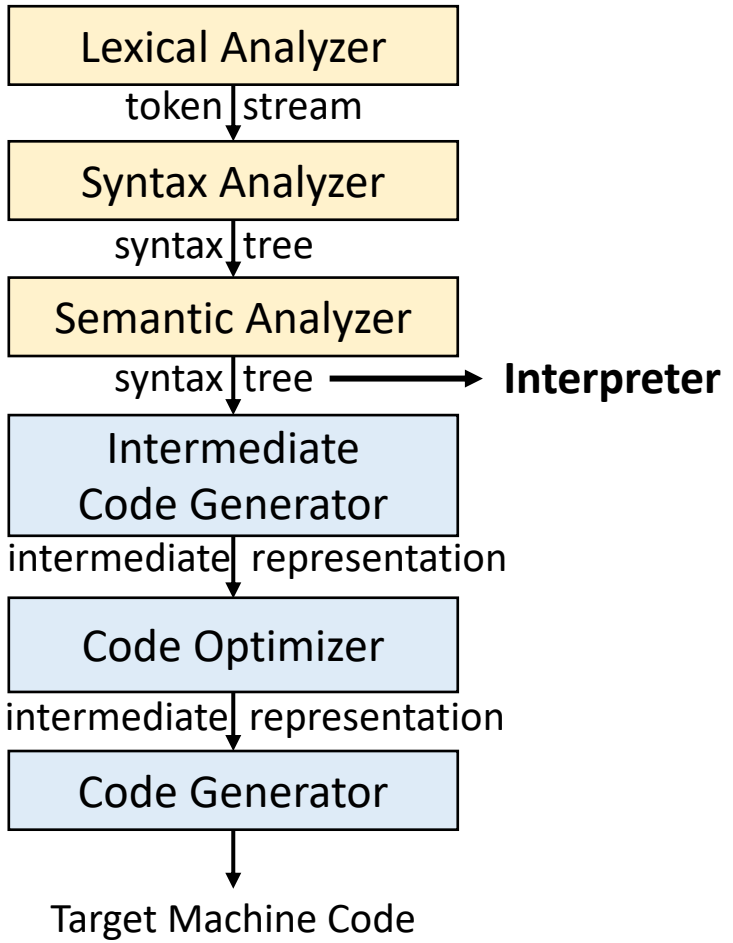


Interpret vs Compile[解释 vs. 编译]

- 编译：翻译成机器语言后方能运行
  - 目标程序独立于源程序（修改→再编译→运行）
  - 分析程序上下文，易于整体性优化
  - 性能更好（因此核心代码通常为C/C++）

- 解释：源程序作为输入，边解释边执行
  - 不生成目标程序，可迁移性高
  - 逐句执行，很难进行优化
  - 性能通常不会太好

Quiz: Java是解释型还是编译型语言？



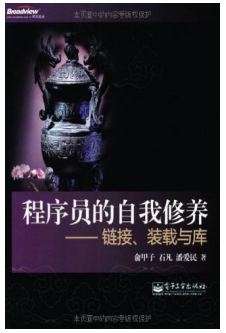
JIT[即时编译]

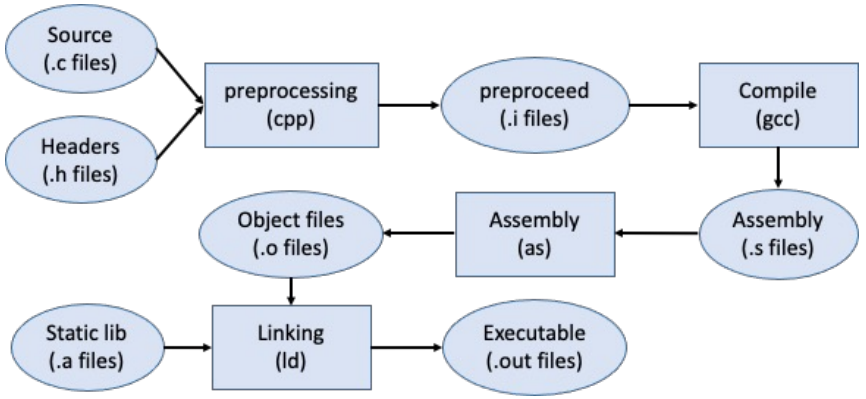
- 即时编译 (Just-In-Time Compiler)： 运行时执行程序编译操作
  - 弥补解释执行的不足
    - 把翻译过的机器代码保存起来，以备下次使用
  - 传统编译 (AOT, Ahead-Of-Time)：先编译后运行

- JIT vs. AOT
  - JIT具备解释器的灵活性
    - 只要有JIT编译器，代码即可运行
  - 性能上基本和AOT等同
    - 运行时编译操作带来一些性能上的损失
    - 但可以利用程序运行特征进行动态优化

Compilation for C [C语言编译]

- 源程序（source.c）→ 可执行文件（a.out）
  - 预处理（preprocessing）
    - 汇合源程序，展开宏定义，生成.i文件
  - 编译（compiling）
    - 将.i文件翻译为.s文件（汇编代码，assembly）
  - 汇编（assembling）
    - .s文件转换为.o文件（可重定位对象，machine code）
  - 连接（linking）
    - 连接库代码生成可执行文件（机器码，machine code ）

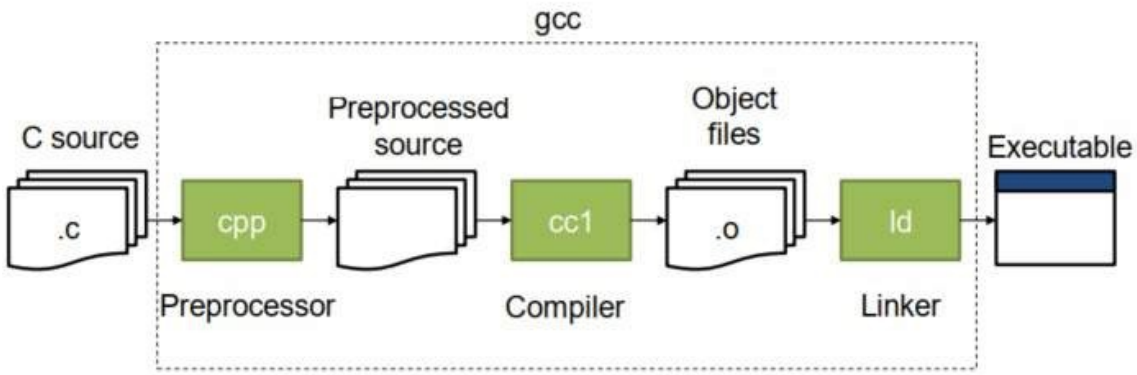





```
$ gcc source.c –o a.out
$ ./a.out
```

```
#include <stdio.h>

int main( )
{
   printf("Hello World!\n");
   return 0;
}
```

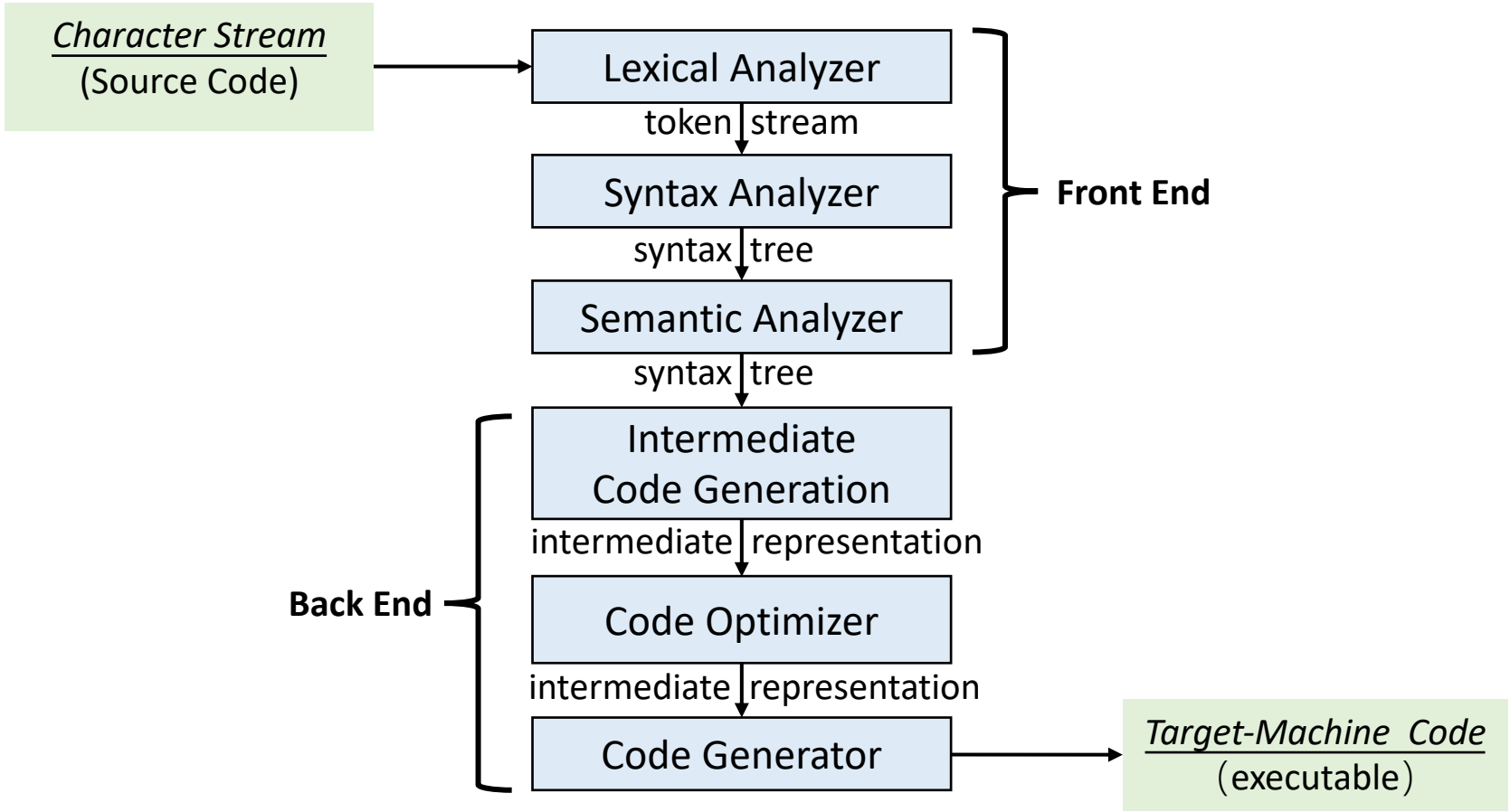



```
55
48 89 e5
Bf d0 05 40 00
E8 d6 fe  ff  ff
B8 00 00 00 00
5d
c3
```

Why Compiler [为什么学编译原理]

- 为了更好的编程?
  - 有助于产出高质量代码
  - 大部分编程工作并不需要编译器知识
- 找到好工作?
  - 编译器研究员（华为）、游戏编译器工程师（腾讯）、编译器优化工程师（英特尔）等等
  - 大部分计算机相关工作并不要求编译器背景
- 包含计算机学科的**主要设计理念**:抽象，自顶 (底) 向下 (上) 等
- 提供**理论方法与实践应用**的深度结合

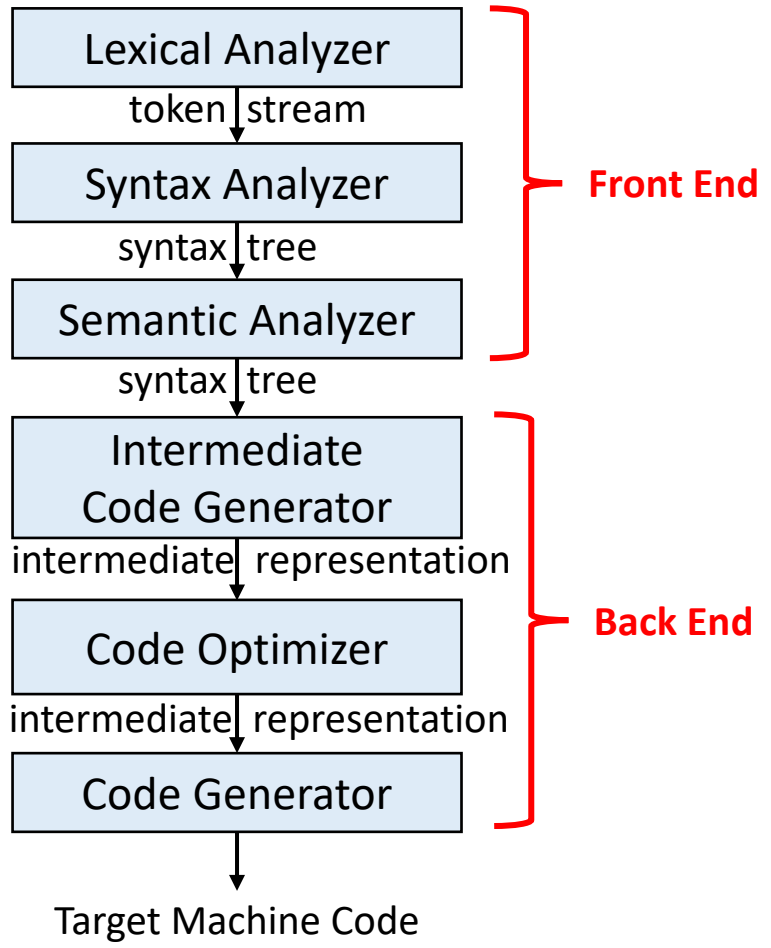
Overview of Compiler [编译结构总览]



Compilation Procedure[编译过程]

- 前端 (for Analysis)：对源程序，识别语法结构信息，理解语义信息，反馈出错信息
  - 词法分析（Lexical Analysis）
  - 语法分析（Syntax Analysis）
  - 语义分析（Semantic Analysis）

- 后端 (for Synthesis)：综合分析结果，生成语义上等价于源程序的目标程序
  - 中间代码生成（Intermediate Code Generation）
    - Intermediate Representation (IR)
  - 代码优化（Code Optimizer）
  - 目标代码生成（Code Generation）
    - Target Machine Code



Lexical Analysis[词法分析]

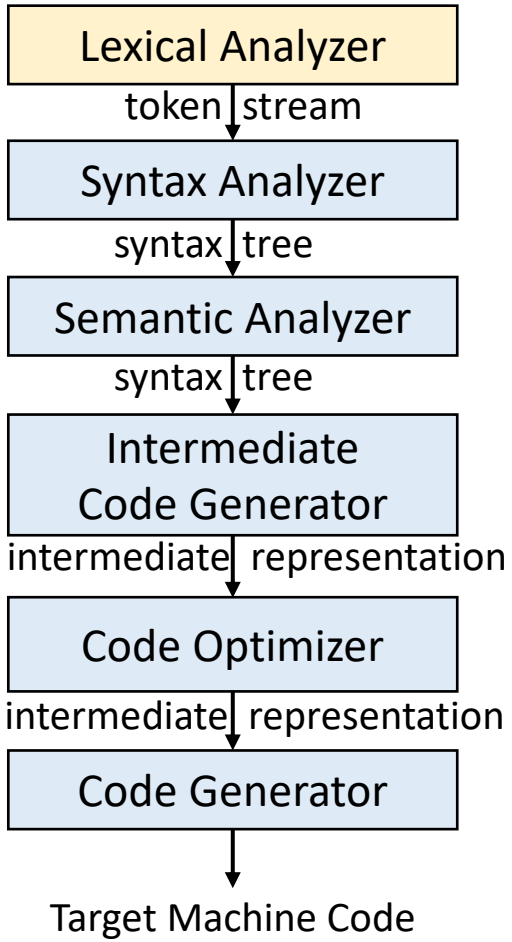
- 扫描源程序字符流，识别并分解出有词法意义的单词或符号（token）
  - 输入：源程序；输出：token序列
  - token表示：（类别，属性值）
    - 关键字、标识符、常量、运算符等
  - token是否符合词法规则？
    - 0var, $num

```
while (y<z){
    int x = a + b;
    y += x;
}
```

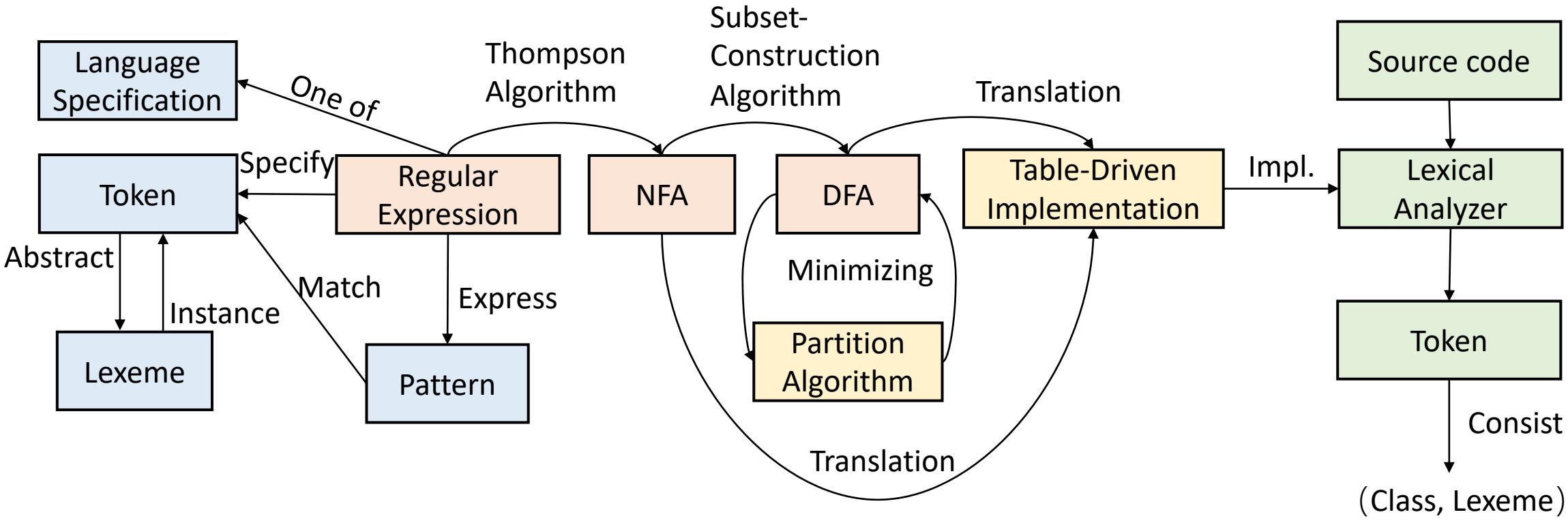
⇨

(keyword, **while**)
(id, **y**)
(sym, **<**)
(id, **z**)
(id, **x**)
(id, **a**)
(sym, **+**)
(id, **b**)
(sym, **;**)
(id, **y**)
(sym, **+=**)
(id, **x**)
(sym, **;**)

Source Code　　Token Stream



Lexical Analysis[词法分析]

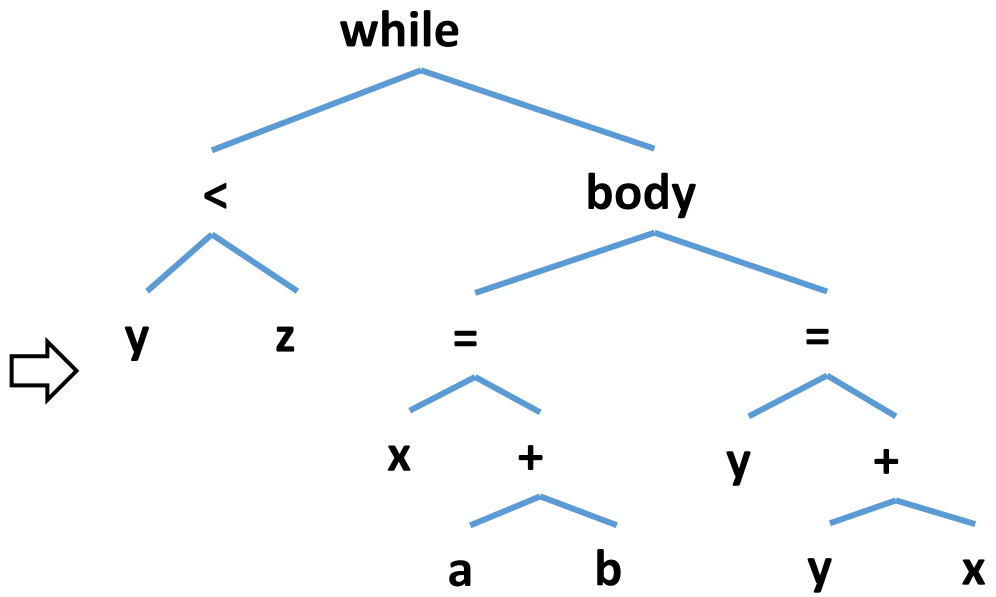


Syntax Analysis[语法分析]

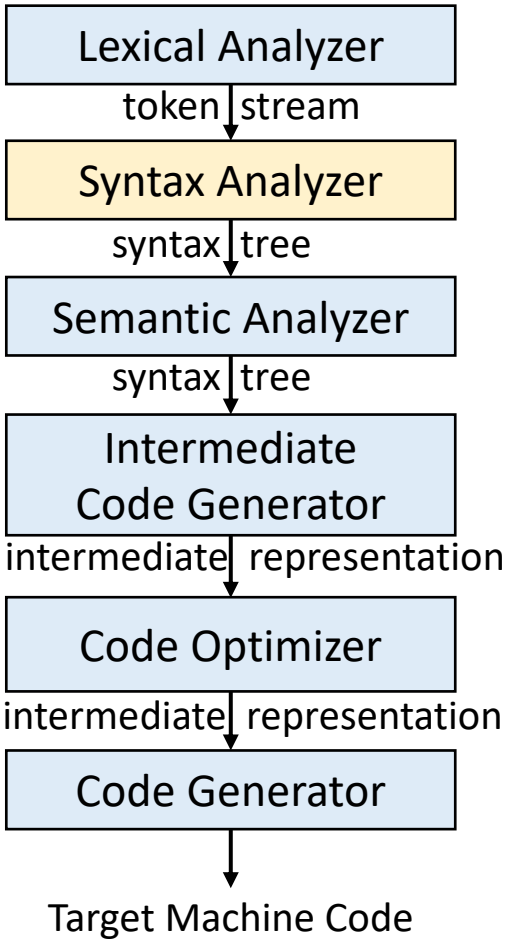
- 解析源程序对应的token序列，生成语法分析结构（syntax tree, 语法分析树）
  - 输入：单词流；输出：语法树
  - 输入程序是否符合语法规则?
    - x*+
    - a += 5;

(keyword, **while**)
(id, **y**)
(sym, **<**)
(id, **z**)
(id, **x**)
(id, **a**)
(sym, **+**)
(id, **b**)
(sym, **;**)
(id, **y**)
(sym, **+=**)
(id, **x**)
(sym, **;**)

Token Stream

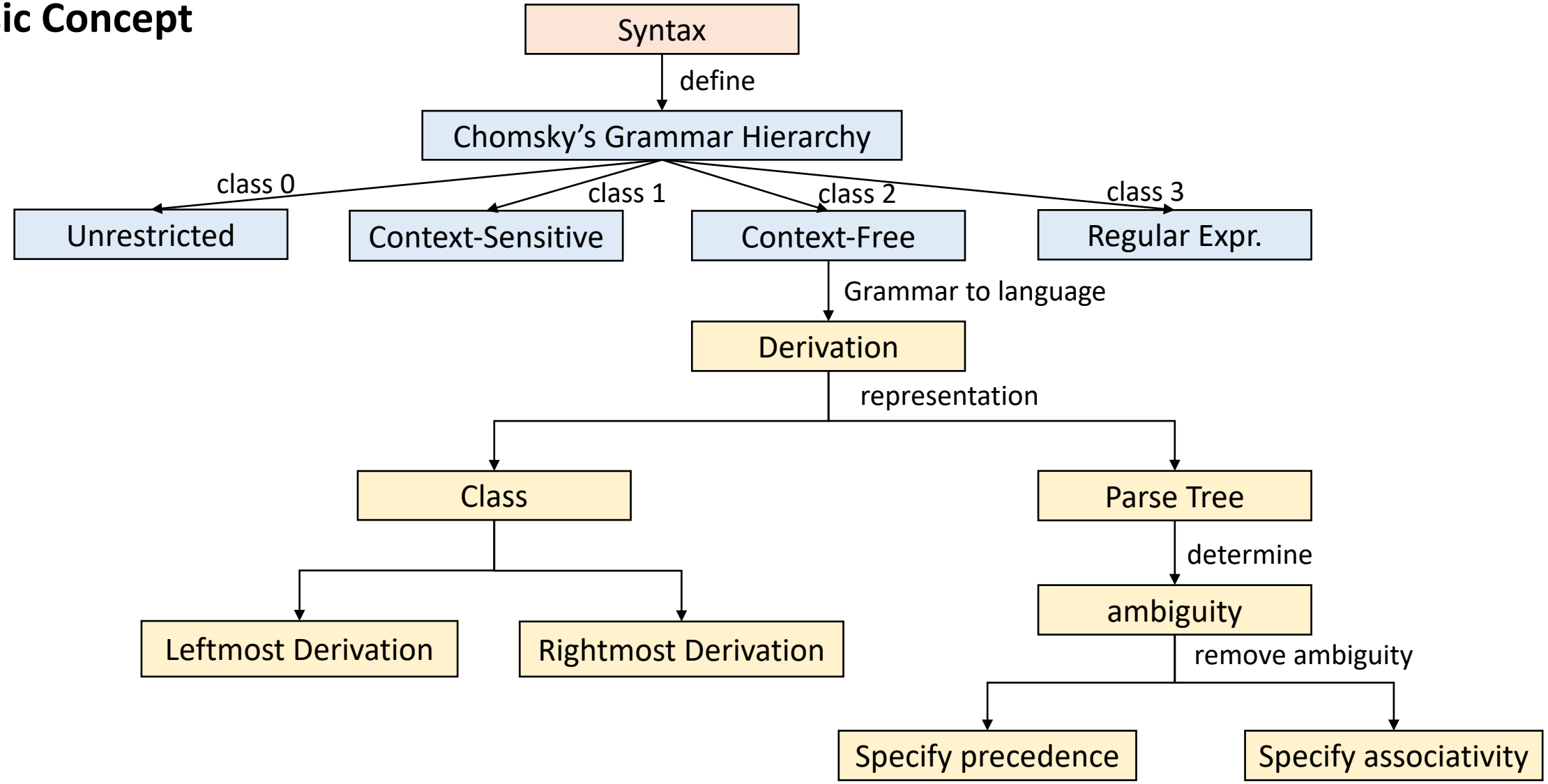


Abstract Syntax Tree (AST)

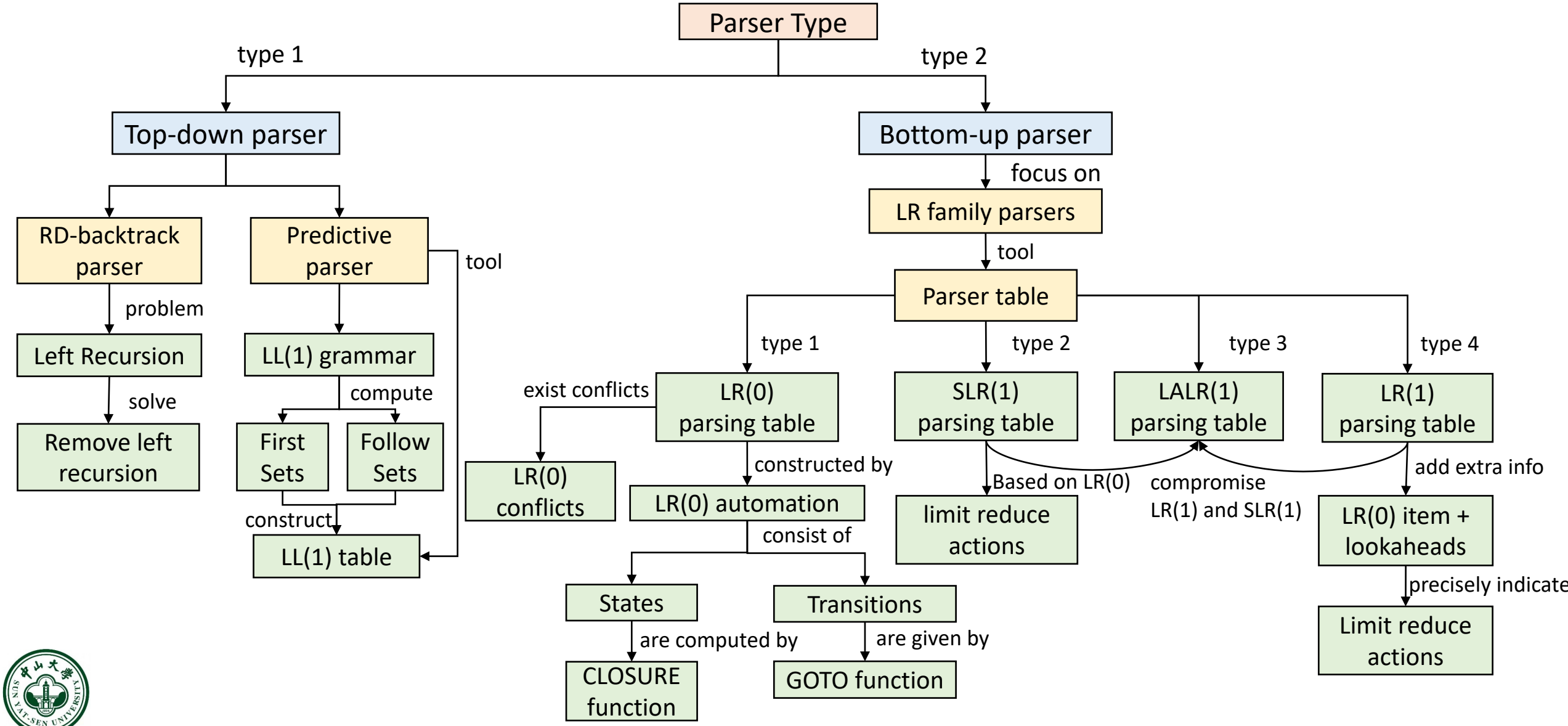


Syntax Analysis[语法分析]

Basic Concept

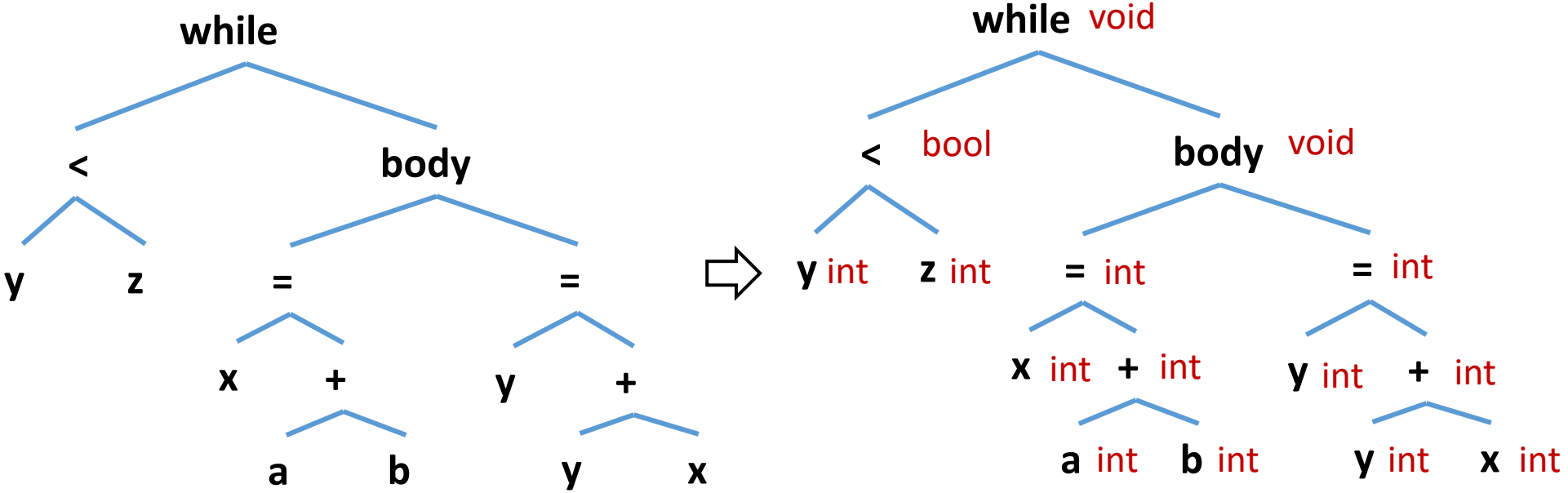


Syntax Analysis[语法分析]



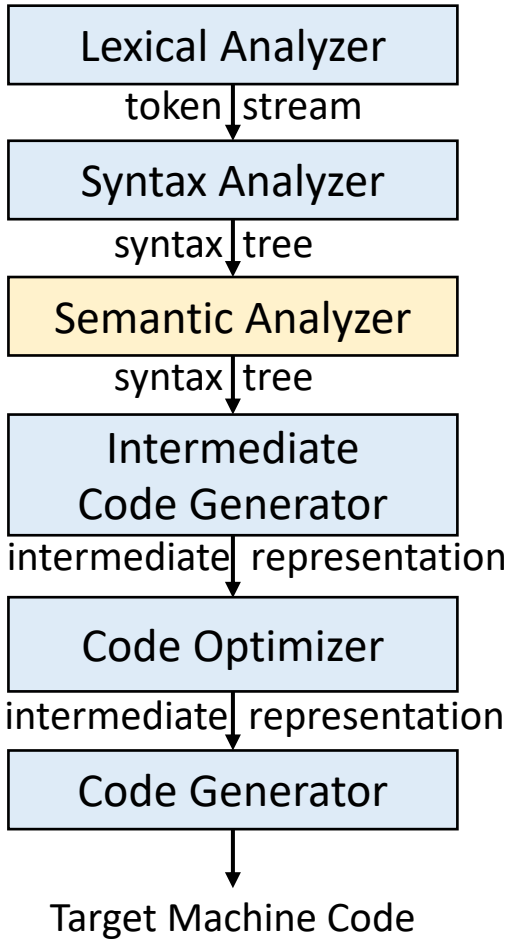
Semantic Analysis[语义分析]

- 基于语法结果进一步分析语义
  - 输入：语法树；输出：语法树+符号表
  - 收集标识符的属性信息（type, scope等）
  - 输入程序是否符合语义规则？
    - 变量未声明即使用，重复声明

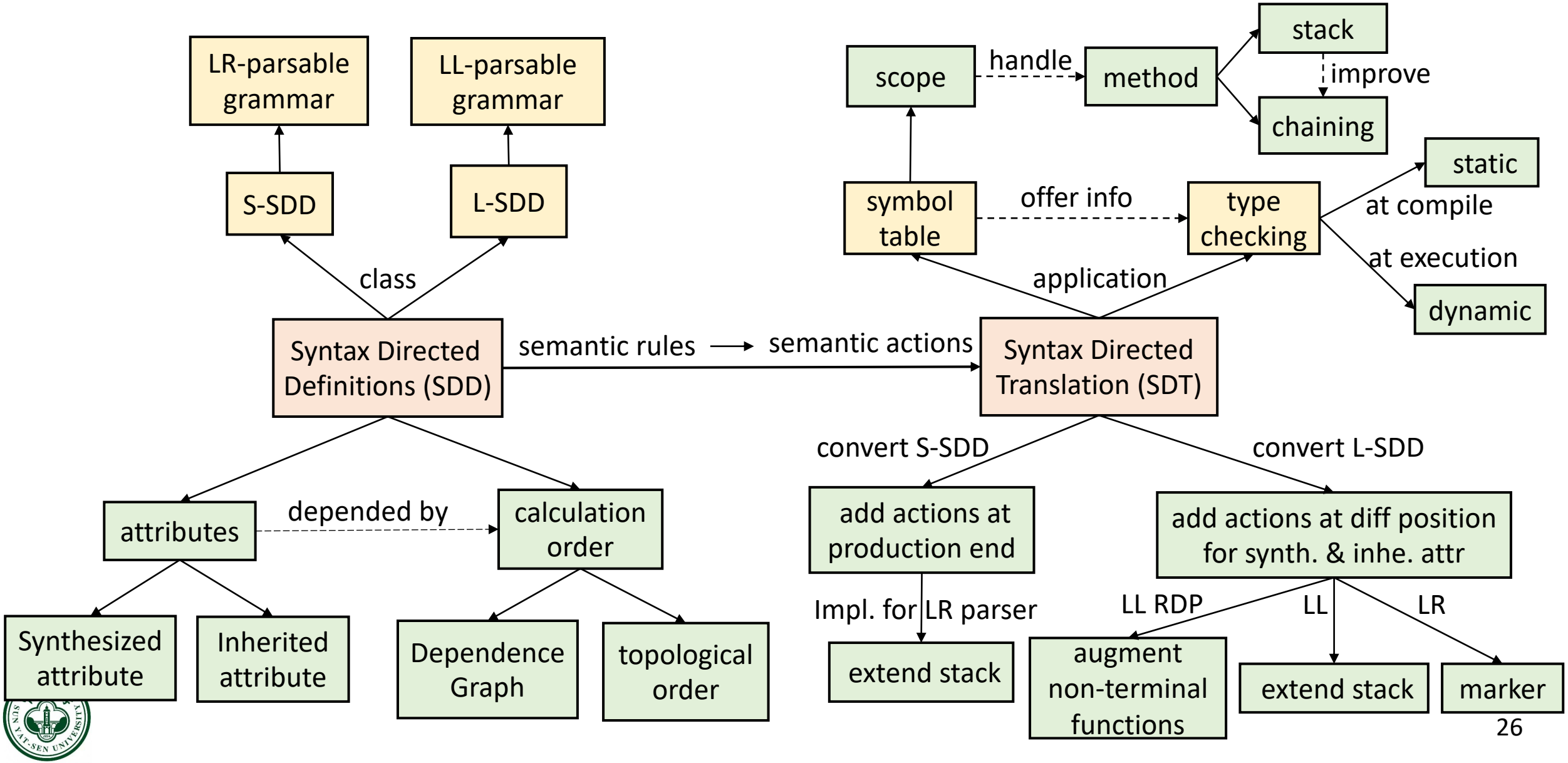


Abstract Syntax Tree (AST)
[抽象语法树]

Annotated AST/Decorated AST
[带标注的抽象语法树]

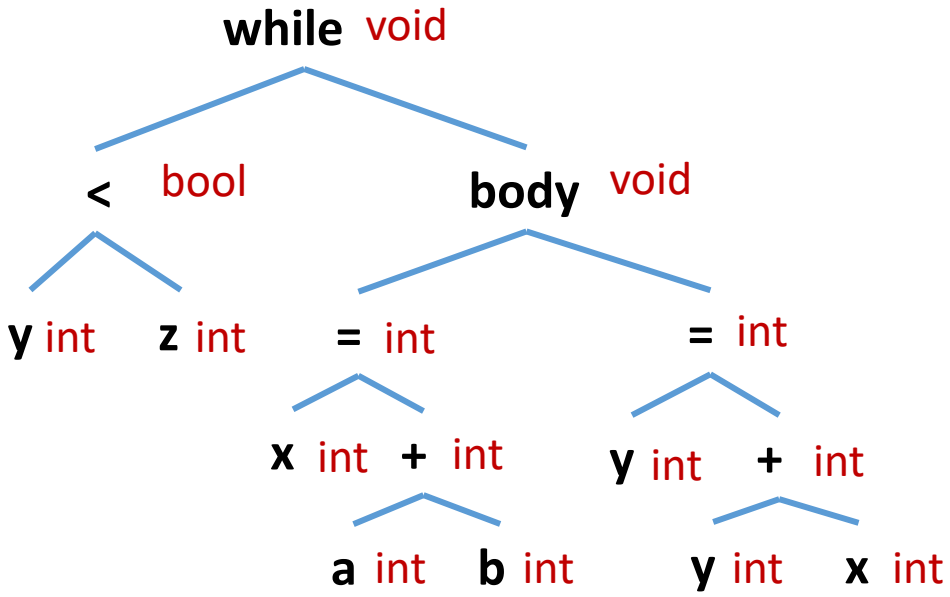


Semantic Analysis[语义分析]



Intermediate Code[中间代码生成]

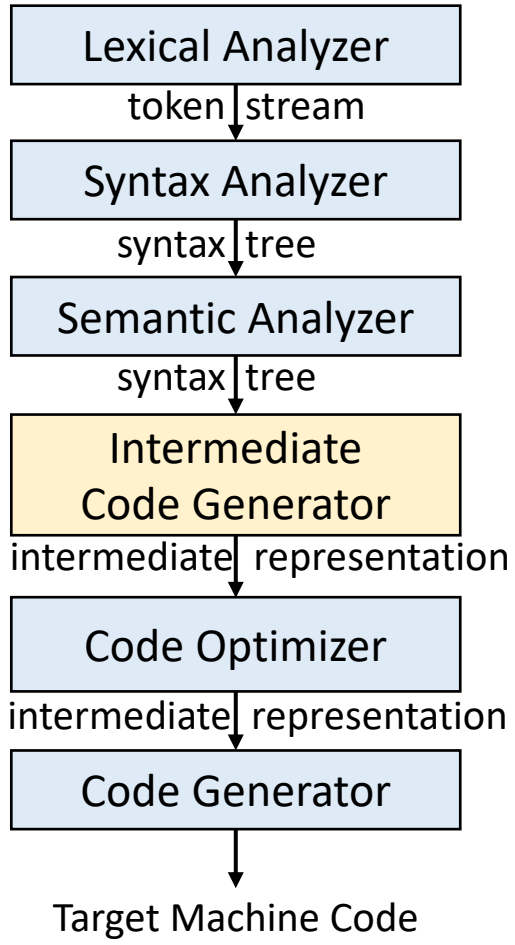
- 初步翻译，生成等价于源程序的中间表示（IR）：
  - 输入：语法树；输出：IR
  - 建立源和目标语言的桥梁，易于翻译过程的实现，利于实现某些优化算法
  - IR形式：例如三地址码（TAC, Three Address Code）



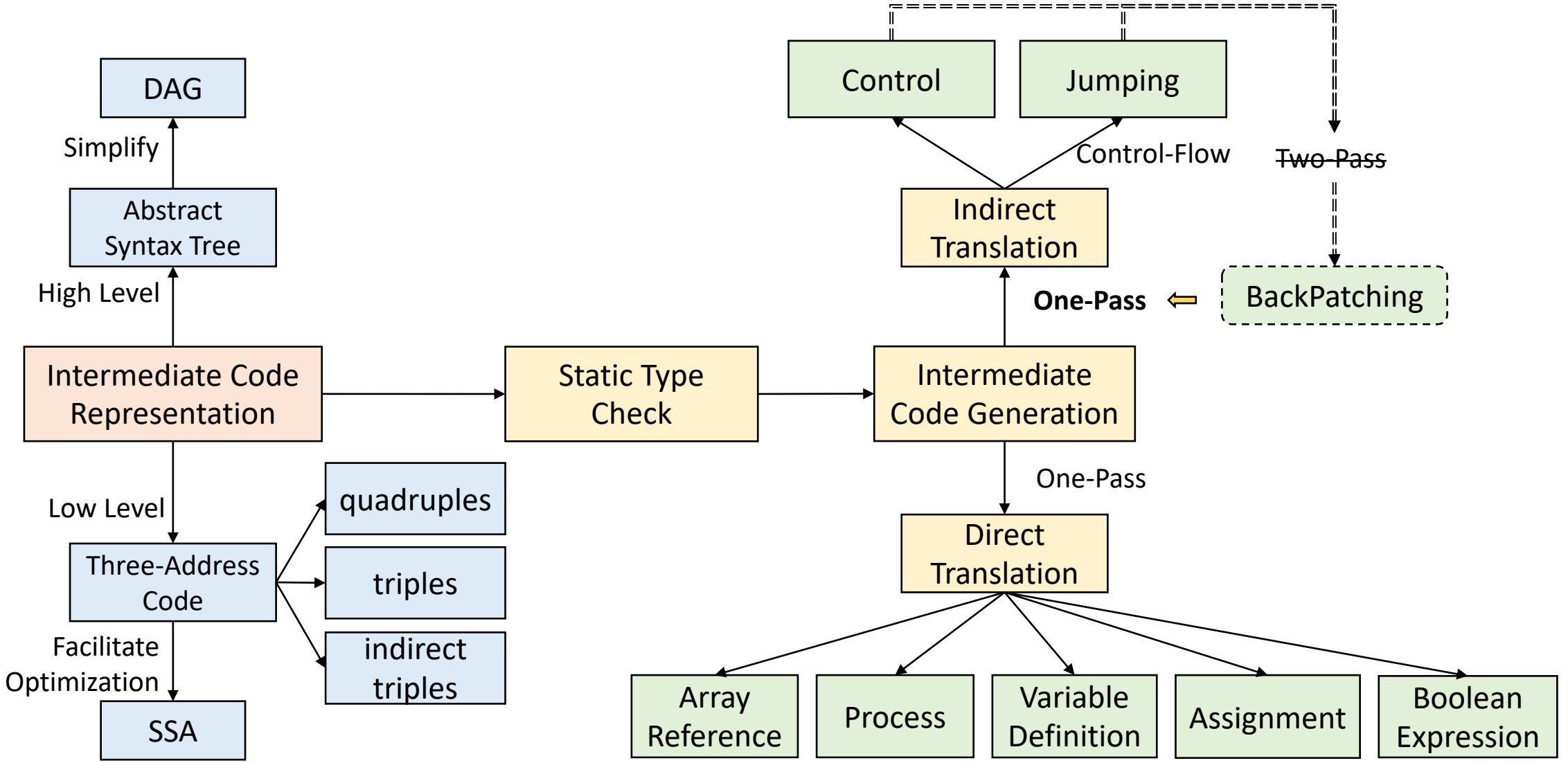
Annotated AST/Decorated AST

```
goto L1
L2:
      t1 := a + b
      x := t1
      t2 := y + x
      y := t2
L1:
      if y < z goto L2
```

Intermediate Representation



Intermediate Code[中间代码生成]



Code Optimization[代码优化]

- 加工变换中间代码使其更好（代码更短、性能更高、内存使用更少）
  - 输入：IR；输出：(优化的）IR
  - 机器无关（machine independent）
  - 例如：重复运算识别；运算操作替换

```
goto L1
L2:
      t1 := a + b
      x := t1
      t2 := y + x
      y := t2
L1:
      if y < z goto L2
```

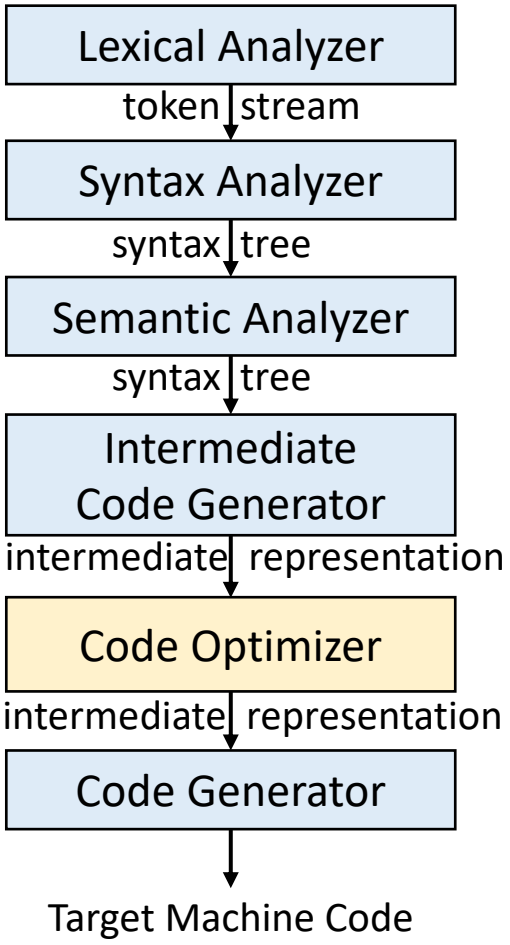
Intermediate Representation

Quiz: 这里做了什么优化?

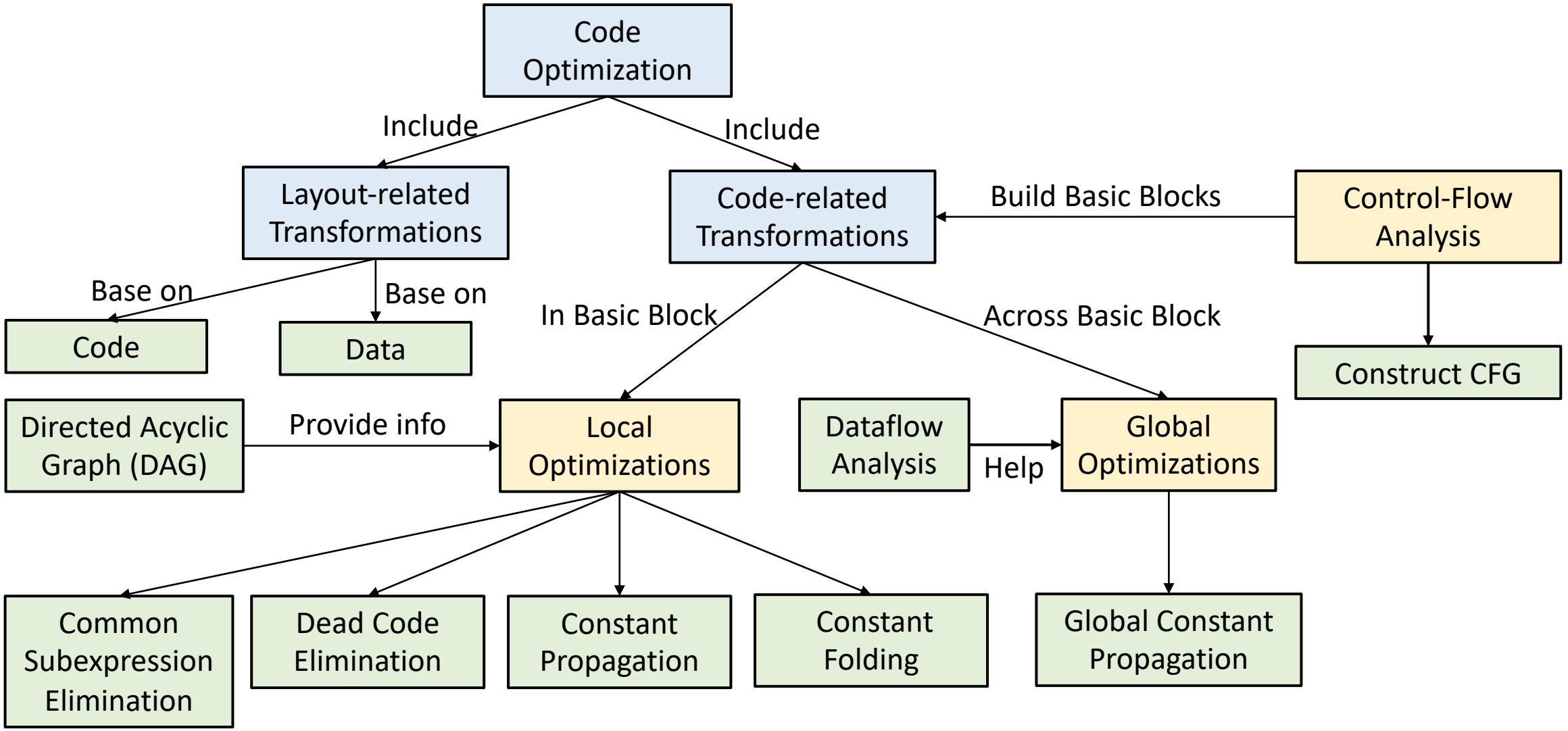
⇨

```
t1 = a + b
goto L1
L2:
      t2 := y + t1
      y := t2
L1:
      if y < z goto L2
```

Optimized Intermediate Representation



Code Optimization[代码优化]



Target Code[目标代码生成]

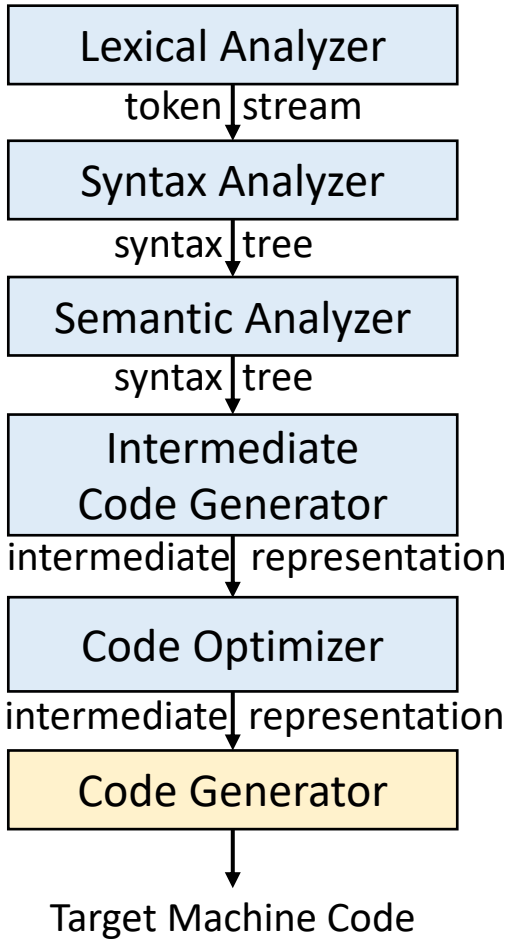
- 为特定机器产生目标代码（e.g., 汇编）
  - 输入：(优化的) IR；输出：目标代码
  - 寄存器分配：放置频繁访问数据
  - 指令选取：确定机器指令实现IR操作
  - 进一步的机器有关优化
    - 例如：寄存器及访存优化

```
t1 = a + b
goto L1
L2:
      t2 := y + t1
      y := t2
L1:
      if y < z goto L2
```

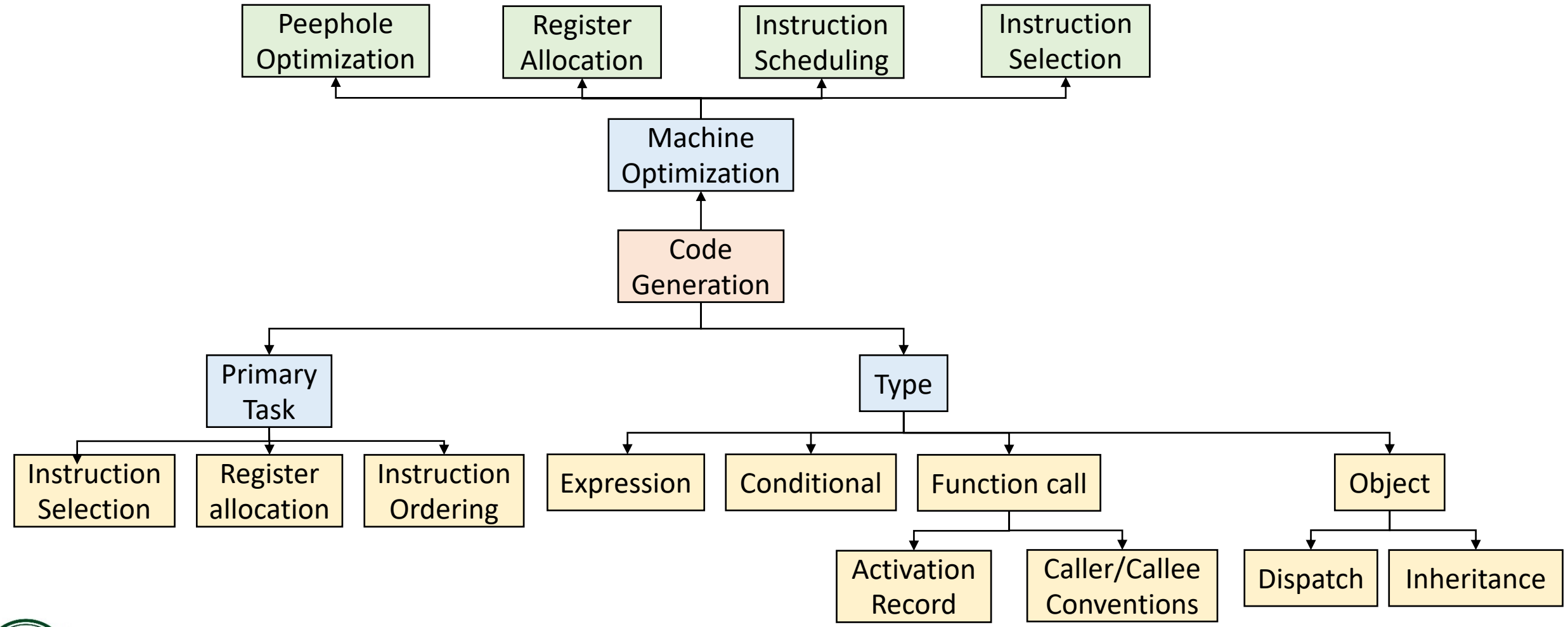
Optimized Intermediate Representation

```
   8b 55 f4    mov   edx, DWORD PTR [rbp-12]
   8b 45 f0    mov   eax, DWORD PTR [rbp-16]
   01 d0       add   eax, edx
   89 45 ec    mov   DWORD PTR [rbp-20], eax
   eb 06       jmp   L2
L3:8b 45 ec    mov   eax, DWORD PTR [rbp-20]
   01 45 fc    add   DWORD PTR [rbp-4], eax
L2:8b 45 fc    mov   eax, DWORD PTR [rbp-4]
   3b 45 f8    cmp   eax, DWORD PTR [rbp-8]
   7c f2       jl    L3
```

Target Machine Code



Target Code[目标代码生成]



Enjoy the Journey!