中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

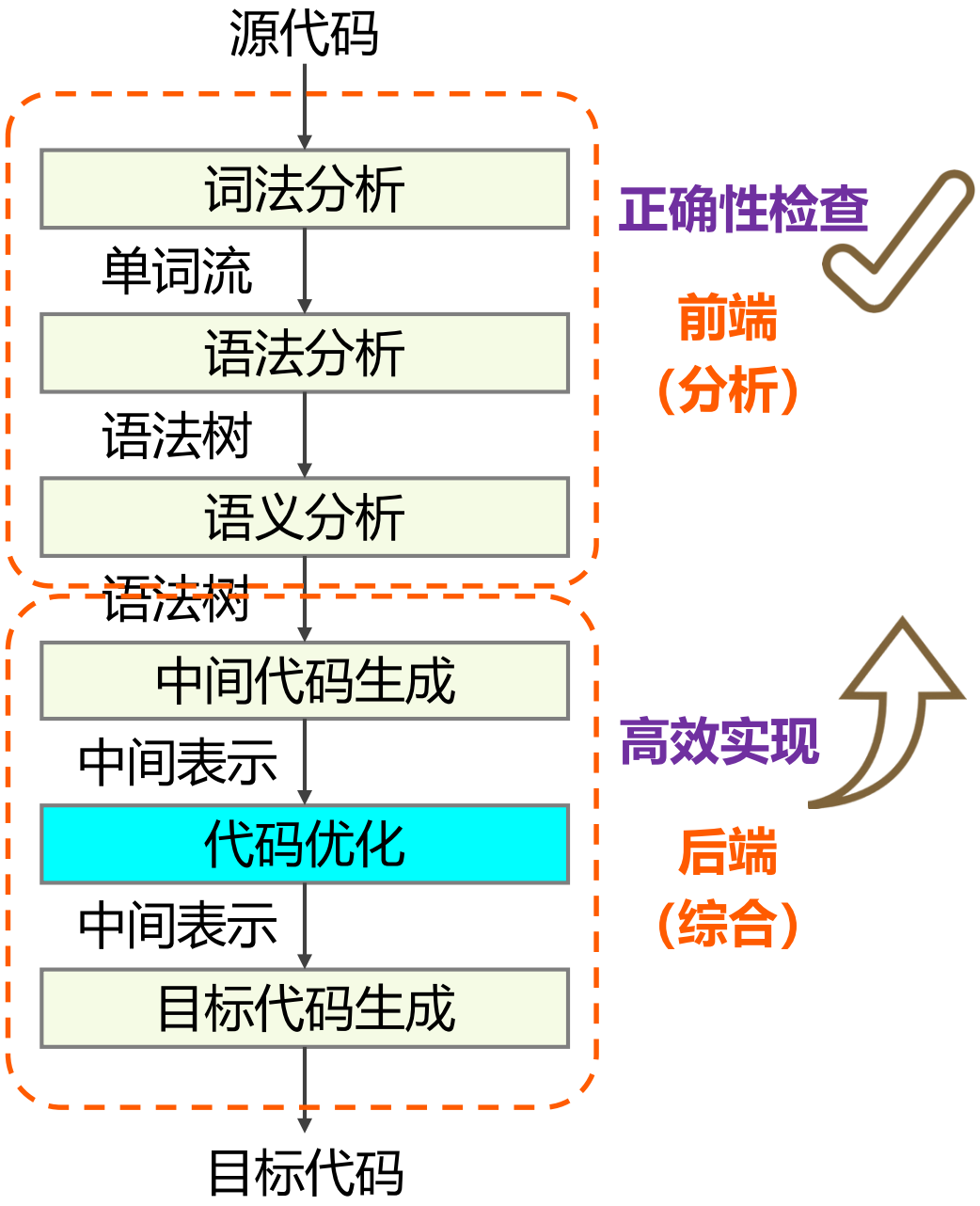
DCS290

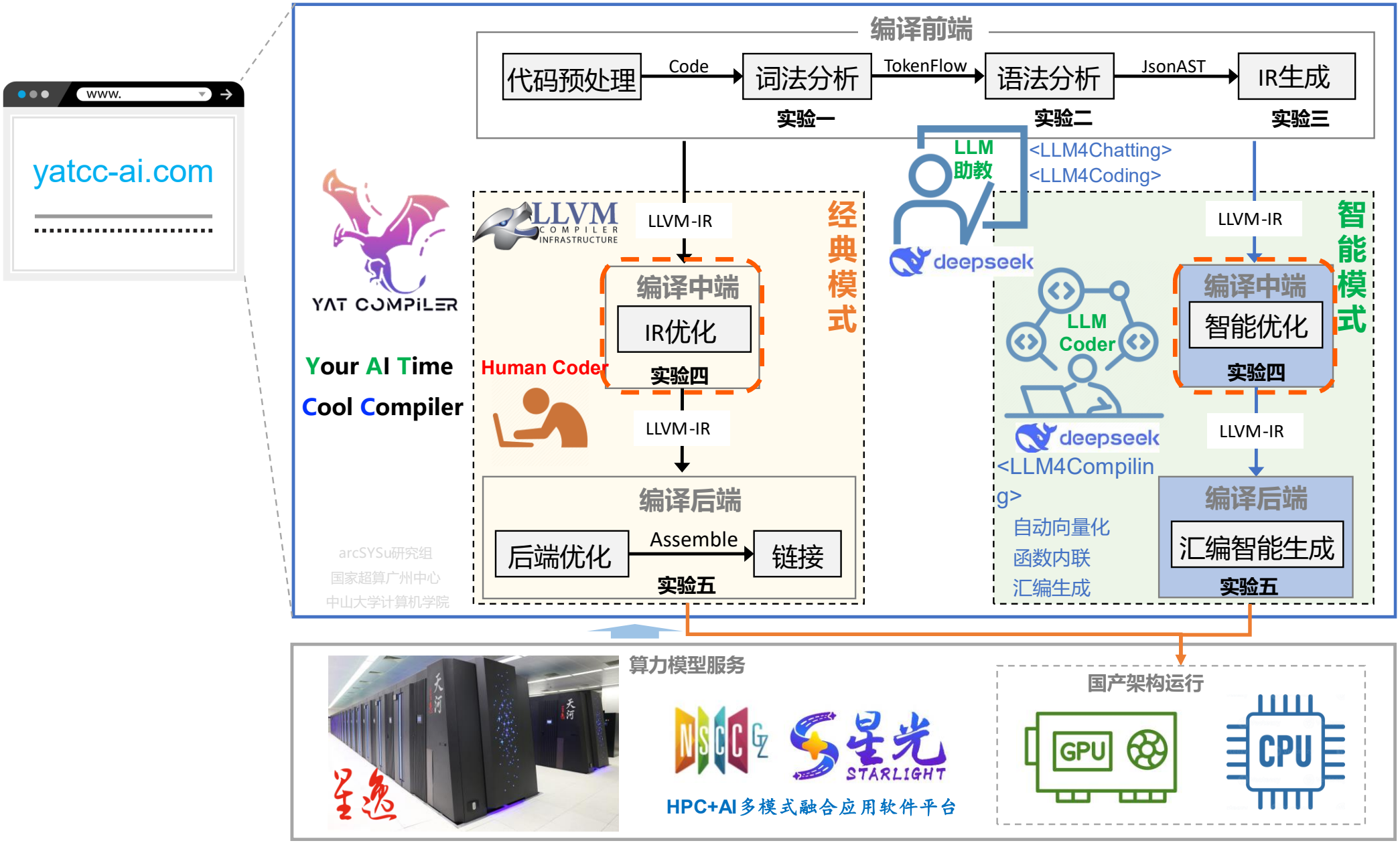
Compilation Principle

编 译 原 理

第九章 代码优化（1）

郑馥丹
zhengfd5@mail.sysu.edu.cn


源代码
词法分析
单词流
语法分析
语法树
语义分析
语法树
中间代码生成
中间表示
代码优化
中间表示
目标代码生成
目标代码
正确性检查
前端
（分析）
高效实现
后端
（综合）


www.
yatcc-ai.com
编译前端
代码预处理
Code
词法分析
TokenFlow
语法分析
JsonAST
IR生成
实验一
实验二
实验三
LLM
助教
<LLM4Chatting>
<LLM4Coding>
YAT COMPILER
Your AI Time
Cool Compiler
LLVM
COMPILER
INFRASTRUCTURE
LLVM-IR
经典模式
编译中端
IR优化
实验四
Human Coder
LLVM-IR
编译后端
后端优化
Assemble
链接
实验五
deepseek
LLM
Coder
deepseek
<LLM4Compiling>
自动向量化
函数内联
汇编生成
LLVM-IR
智能模式
编译中端
智能优化
实验四
LLVM-IR
编译后端
汇编智能生成
实验五
arcSYSu研究组
国家超算广州中心
中山大学计算机学院
算力模型服务
星光
STARLIGHT
HPC+AI多模式融合应用软件平台
国产架构运行
GPU
CPU

CONTENTS

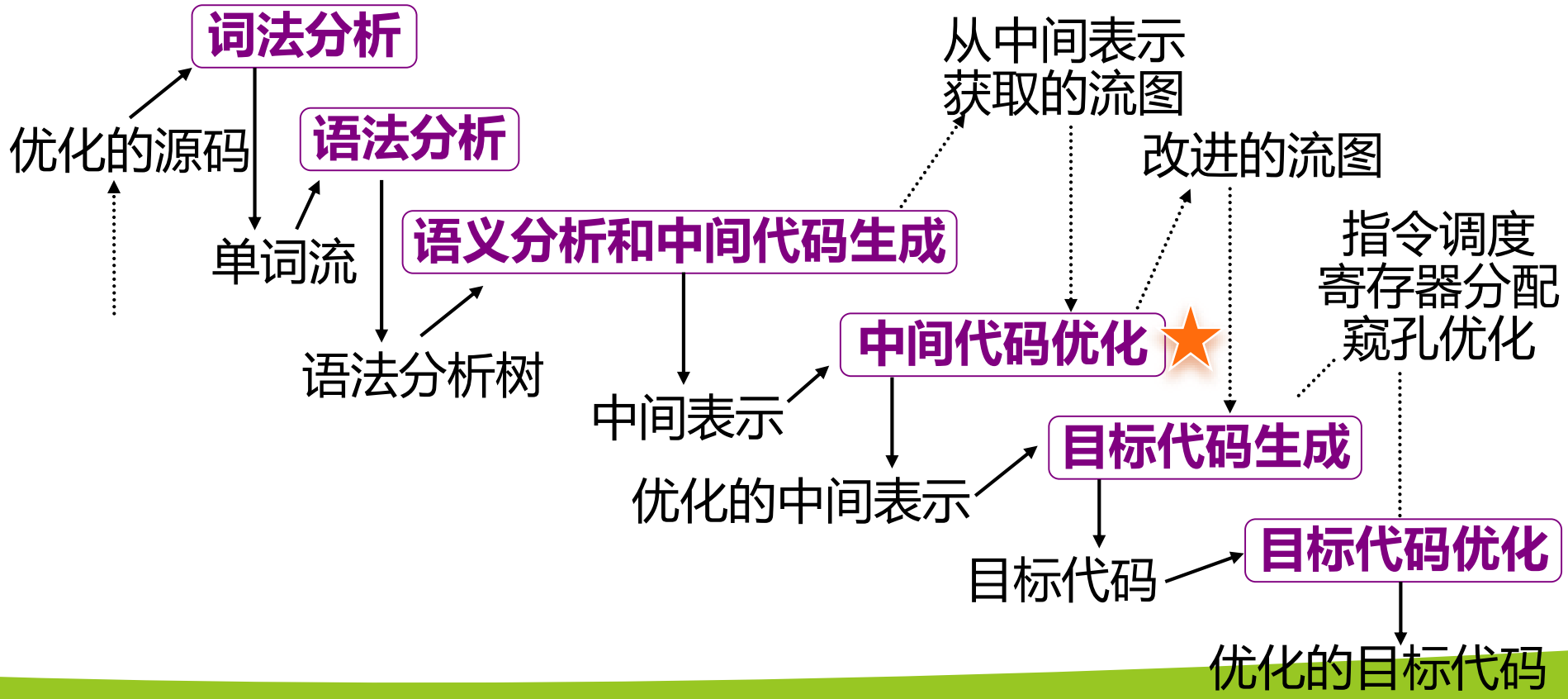
目 录

1. 定义

- 什么是代码优化[Code Optimization]
  - 所谓优化是对代码进行**等价变换**，使得变换后的代码的**效率更高**（节省运行时间、存储空间或两者兼而有之）
  - **语义保持转换**[Semantics-preserving Transformations]
  - 代码优化 vs. 代码改进[Code Improvement]
- 权衡[Trade-off]
  - **时间效率 vs.空间效率**
  - **编译器效率 vs.目标代码效率**
    - ✓ 高级别的优化→高效的目标代码 ↔ 更多的编译时间
    - ✓ 复杂的优化 ↔ 更多编译内存占用

2. 优化级别

- 三种优化级别
  - 源代码优化
  - **中间代码优化**（不依赖具体计算机）★
  - 目标代码优化（依赖于具体计算机）



3. 优化的分类

- 根据优化涉及的程序范围，分为：
  - **局部优化[Local Optimization]**
    - ✓ 在只有一个入口、一个出口的**基本块[basic block]**上进行优化
  - **循环优化[Loop Optimization]**
    - ✓ 对循环中的代码进行优化
    - ✓ 基于控制流分析
  - **全局优化[Global Optimization]**
    - ✓ 在整个程序范围内进行的优化
    - ✓ 基于数据流分析
  - 窥孔优化[Peephole Optimization]
    - ✓ 对一个滑动窗口内的代码进行优化

4. 通用技术

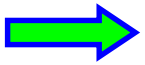
(1) **删除公共子表达式**[Common Subexpressions]

– 若子表达式E在前面计算过，且之后E中的**变量值都未改变**，则E的重复出现称为公共子表达式，可将其删除，避免重复计算

(1) T_1 :=4*I

(2) T_2 :=addr(A) - 4

(3) T_3 :=T_2[T_1]

(4) T_4 :=4*I  (4) T_4 :=T_1

(5) ……

4. 通用技术

(2) **常量折叠及复写传播**[Copy Propagation]

- 如果运算量都是已知量，则在编译时就算出它的值，称为**常量折叠（合并已知量）**。
- 若有A:=B，称为把B值复写到A。如果其后有引用A的地方，且其间A、B的值都未改变，则可把对A的引用改为对B引用，称为**复写传播**。

(1) I:=1

(2) T_1:=4*I ⇒ (2) T_1:=4 常量折叠

(3) T_4:=T_1 ⇒ (3) T_4:=4 常量传播

(4) T_6:=T_5[T_4] ⇒ (4) T_6:=T_5[T_1] 复写传播

4. 通用技术

(3) 消除死代码（删除无用赋值）

– 死代码/无用赋值：

- ✓ 变量被赋值，但在程序中从未被引用
- ✓ 变量赋值后未被引用又重新赋值，则前面赋值是无用的
- ✓ 变量的赋值仅被计算变量自己引用

(1) I:=1
(2) T_1:=4
(3) T_3:=T_2[T_1]
(4) T_4:=T_1
(5) I:=I+1
(6) T_1:=T_1+4
(7) if T_1≤80 goto (3)

(2) T_1:=4
(3) T_3:=T_2 [T_1]
(6) T_1:=T_1+4
(7) if T_1≤80 goto (3)

CONTENTS

目 录

1. 定义

- 局部优化是指**基本块内**的优化
  - **基本块**是指程序中一个顺序执行的语句序列，其中**只有一个入口语句和一个出口语句**。执行时只能从入口语句进入，从其出口语句退出。

- 局部优化的逻辑：

2. 基本块的划分

(1) 求基本块的入口语句，它们是：

– 程序的**第一个语句**；

– 条件转移或无条件转移语句的**转移目标语句**；

– 紧跟在**条件转移语句后面的语句**。

```
(1) read  X
(2) read  Y
(3) R:=X  mod  Y
(4) if  R=0   goto  (8)
(5) X:=Y
(6) Y:=R
(7) goto  (3)
(8) write  Y
(9) halt
```

入口语句：
(1)、(3)、(5)、(8)

2. 基本块的划分

(2) 对每一入口语句，构造其所属的基本块:

- 它是由该入口语句到**下一入口语句**(不包括下一入口语句);
- 或到**一转移语句**(包括该转移语句);
- 或到**一停止语句**(包括该停止语句)之间的语句序列组成的。

```
(1) read  X
(2) read  Y
(3) R:=X  mod  Y
(4) if  R=0   goto  (8)
(5) X:=Y
(6) Y:=R
(7) goto  (3)
(8) write  Y
(9) halt
```

入口语句:
(1)、(3)、(5)、(8)
分别构成基本块:
B1 {(1)、(2)}
B2 {(3)、(4)}
B3 {(5)、(6)、(7)}
B4 {(8)、(9)}

凡未被纳入某一基本块的语句，是不会被执行到的语句，可删除!

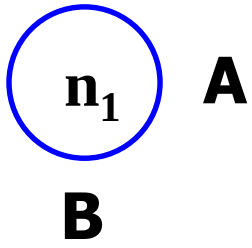
3. 基本块的DAG表示

- **DAG[Directed Acyclic Graph]**：有向无环图
- 基本块的DAG是一种结点带有标记或附加信息的DAG：
  - **叶子结点：标识符或常数**，代表变量或常数的值
  - **内部结点：运算符**，代表用该算符对其后继结点所代表的值进行运算的结果
  - 各结点都可以附加上一个或多个标识符，表示这些变量具有该结点所代表的值

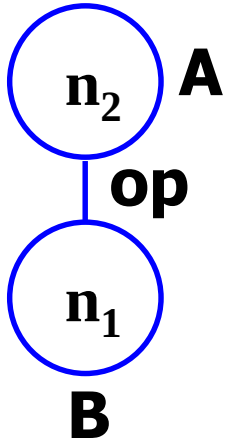
3. 基本块的DAG表示

- 具体结点形式

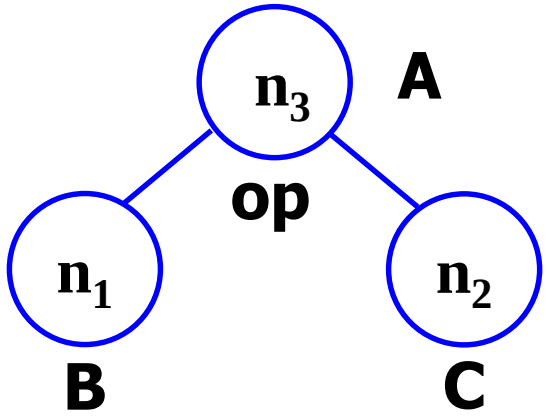
0型： A:=B　　(:=, B, －,A)



1型：
A:=op B　(op , B, －,A)

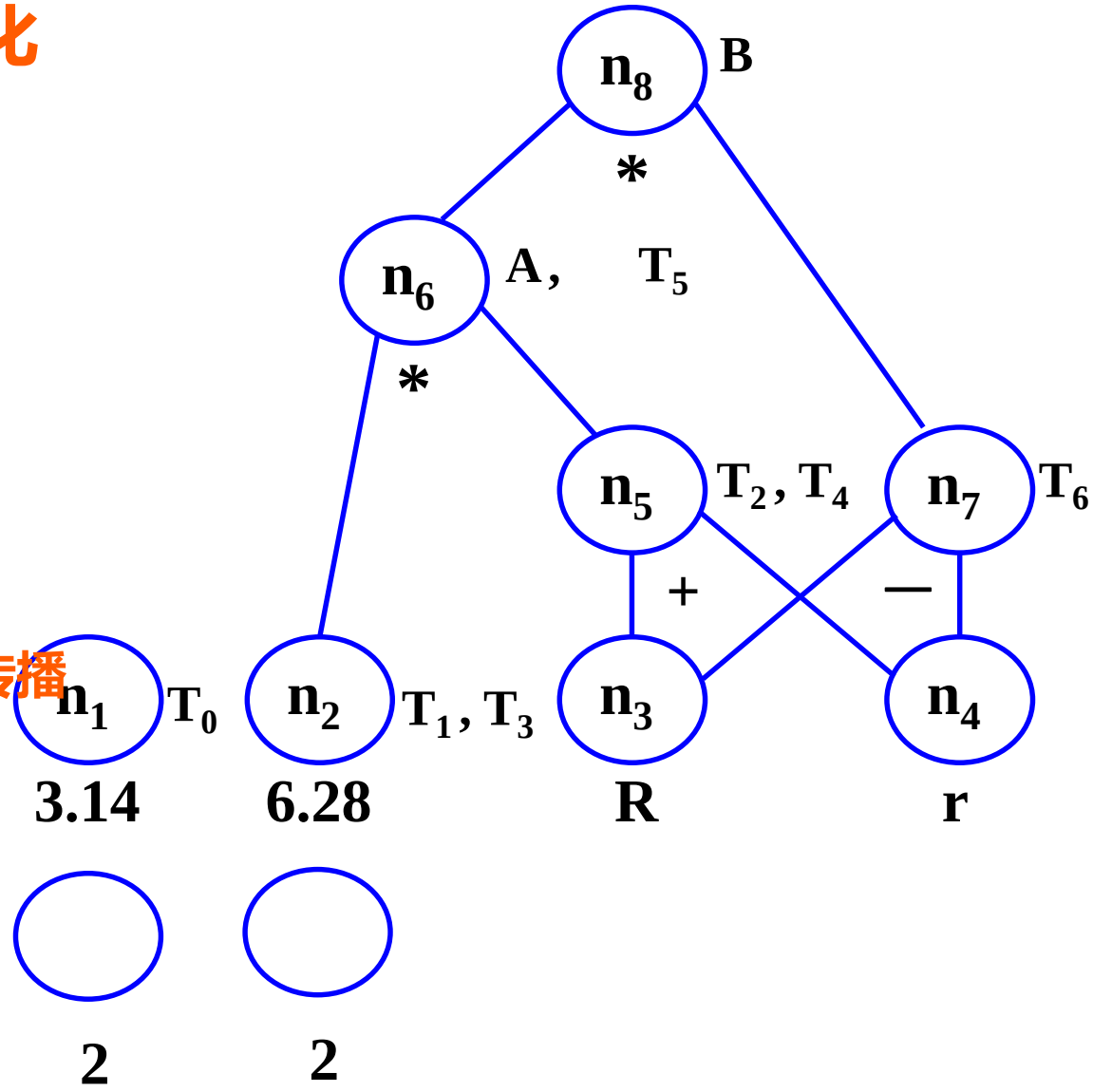


2型：
A:=B op C　(op , B , C , A)



4. DAG的应用

- **例：构造以下基本块的DAG并进行优化**
  - (1) T_0:=3.14
  - ~~(2) T_1:=2*T_0~~ T_1 :=6.28 常量折叠
  - (3) T_2:=R+r
  - (4) A:=T_1*T_2
  - ~~(5) B:=A~~ 消除死代码
  - ~~(6) T_3:=2*T_0~~ T_3 :=6.28 常量折叠+常量传播
  - ~~(7) T_4:=R+r~~ T_4 := T_2 删除公共子表达式
  - ~~(8) T_5:=T_3*T_4~~ T_5 := A 删除公共子表达式
  - (9) T_6:=R—r
  - ~~(10) B:=T_5*T_6~~ B := A* T_6 复写传播

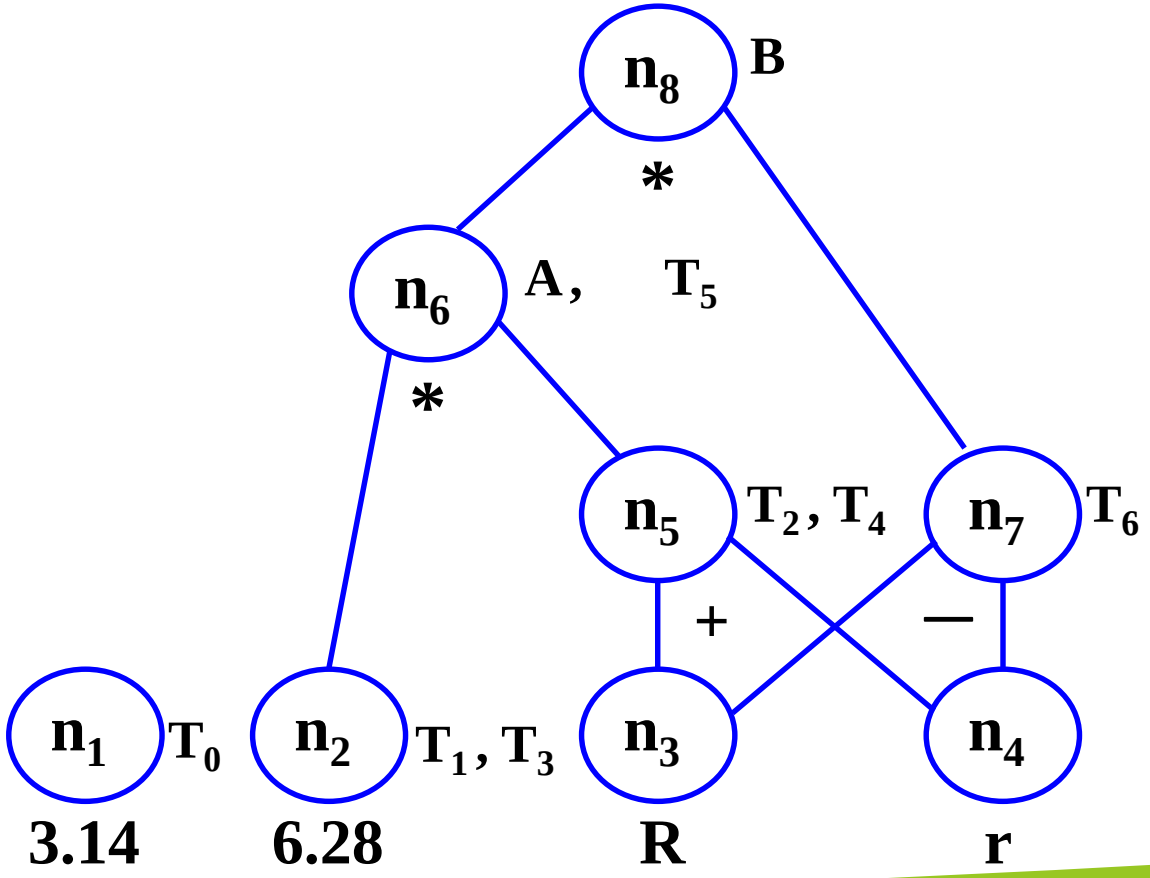


4. DAG的应用

- 利用DAG进行优化
  - 删除在基本块后不被引用变量的赋值

假如T_0，T_1，…，T_6在基本块后都不被引用，则这些符号可在DAG附加标识符中删去

(1) T_0 :=3.14
(2) T_1 :=6.28
(3) T_3 :=6.28
(4) T_2 :=R+r
(5) T_4 :=T_2
(6) A :=6.28*T_2
(7) T_5 :=A
(8) T_6 :=R−r
(9) B :=A*T_6

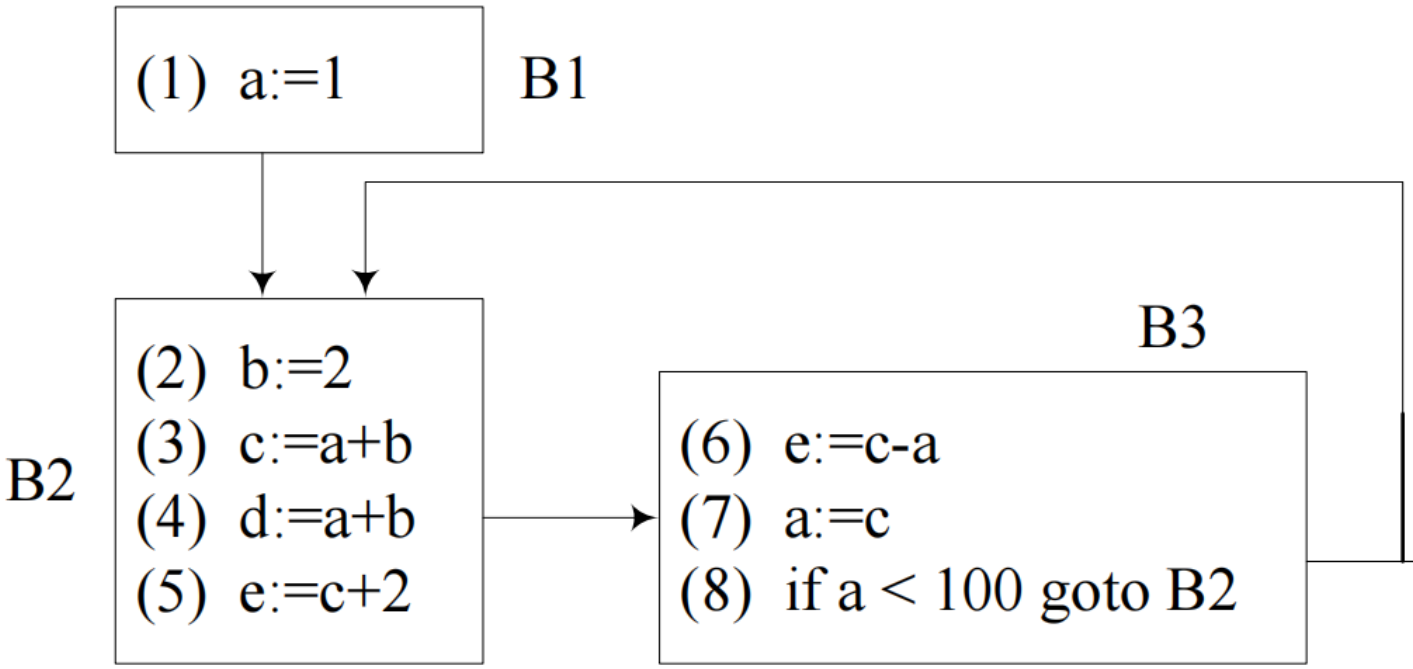


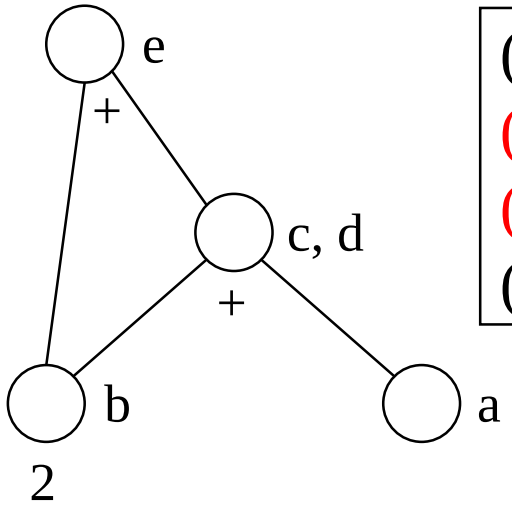
随堂练习（1）

对于以下流图：

(1) 为基本块B2构造DAG图；

(2) 根据DAG图优化基本块B2。





(2) b:=2
(3) c:=a+2
(4) d:=c
(5) e:=c+2

CONTENTS

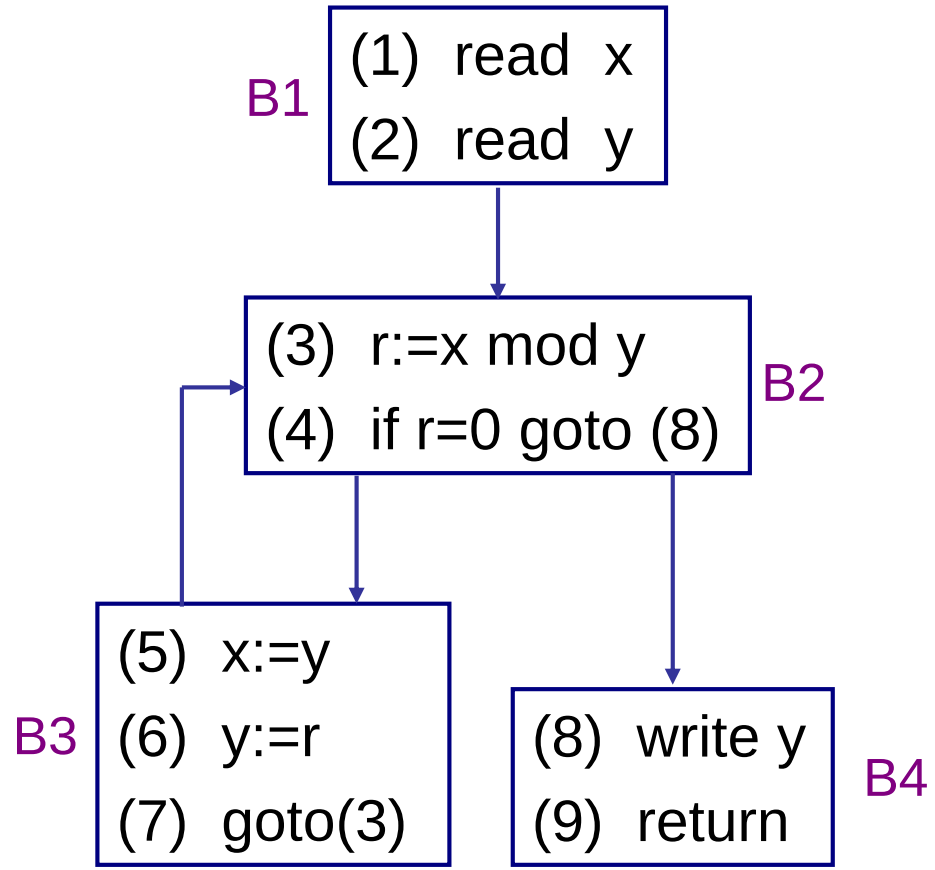
目 录

1. 简介

- **循环优化对提高目标代码的效率作用显著**
- 基于**流图[Flow Graphs]**进行
  - 把**控制流的信息**加到基本块集合上构成的**有向图**称为表示程序的流图，简称流图。
  - 流图的构造方法：
    - ✓ 点集：以基本块为结点，含程序第一条语句的结点为首结点。
    - ✓ 边集：从基本块B_i向基本块B_j引有向边，仅当
      - **B_j在程序中的位置紧跟在B_i之后**，且B_i的出口语句不是无条件转移语句或停止语句。或者
      - **B_i的出口**是转移语句（goto（s)或if...goto（s)），并且**转移目标(s)是B_j的入口**语句。

1. 简介

```
*(1)  read  x
  (2)  read y
*(3)  r:=x mod y
  (4)  if r=0 goto (8)
*(5)  x:=y
  (6)  y:=r
  (7)  goto(3)
*(8)  write y
  (9)  return
```



2. 优化技术

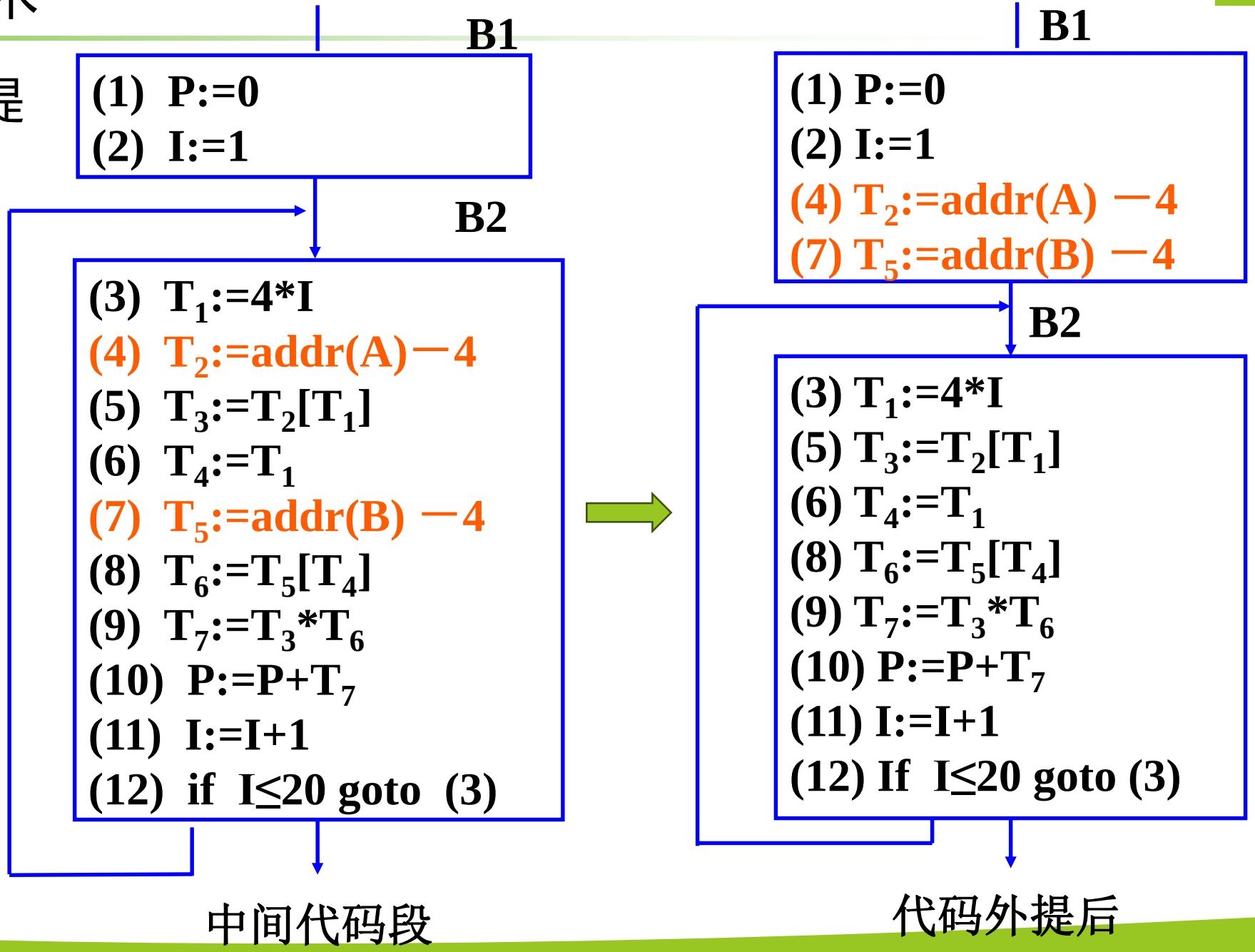
(1) 代码外提[Code Motion]

– 把**循环不变运算**提到循环的前面，使之只在循环外计算一次。

– 循环不变运算：其结果独立于循环执行次数的表达式；运算量为常量或在循环外定值，**每次循环时其值不变的运算**。

2. 优化技术

(1) 代码外提


B1
(1) P:=0
(2) I:=1
B2
(3) T1:=4*I
(4) T2:=addr(A)－4
(5) T3:=T2[T1]
(6) T4:=T1
(7) T5:=addr(B) －4
(8) T6:=T5[T4]
(9) T7:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(12) if I≤20 goto (3)
中间代码段
B1
(1) P:=0
(2) I:=1
(4) T2:=addr(A) －4
(7) T5:=addr(B) －4
B2
(3) T1:=4*I
(5) T3:=T2[T1]
(6) T4:=T1
(8) T6:=T5[T4]
(9) T7:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(12) If I≤20 goto (3)
代码外提后

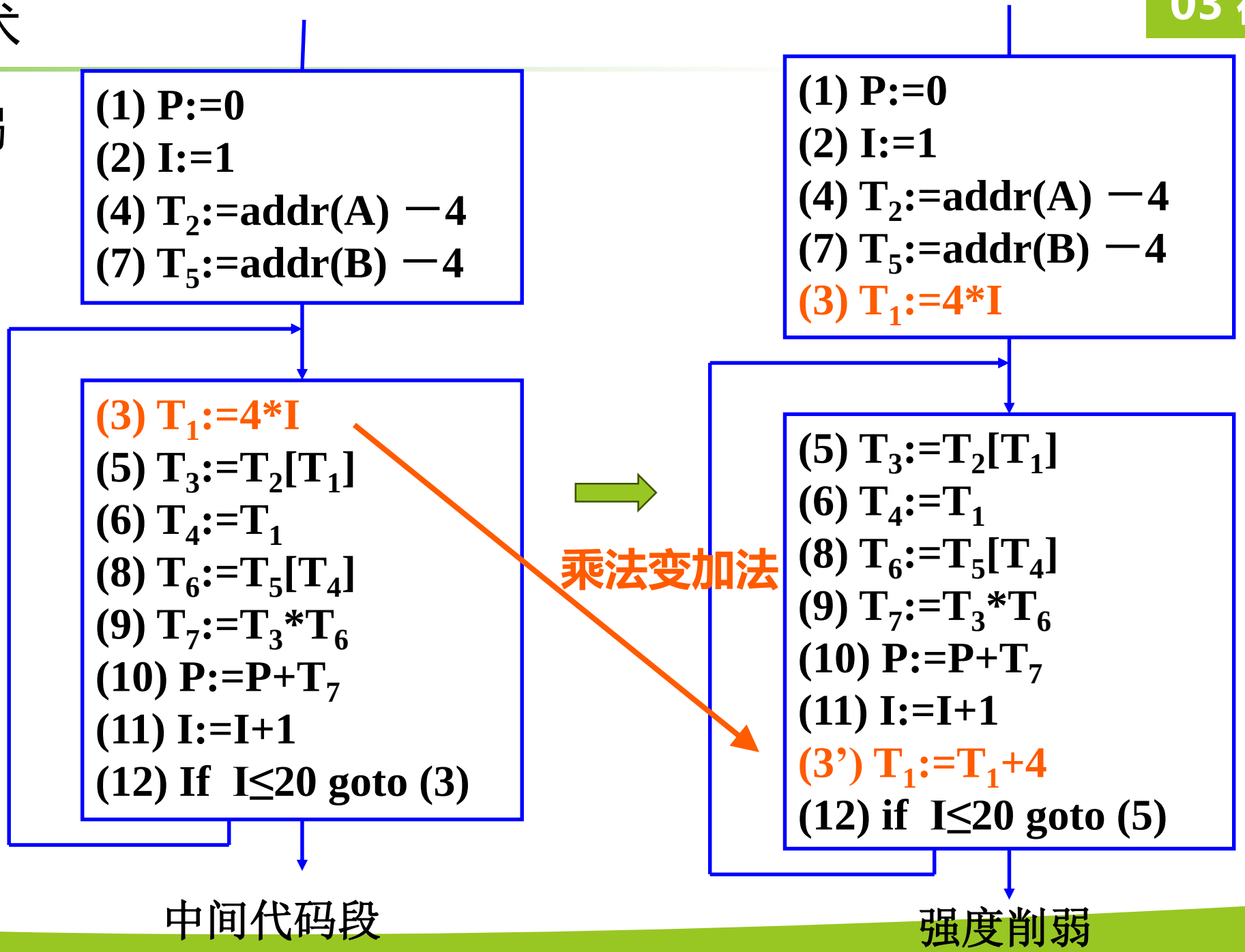
2. 优化技术

(2) 强度削弱[Reduction in Strength]

– 把程序中**强度大的运算替换成强度小**的运算。例如把乘法运算换成加法运算等。

2. 优化技术

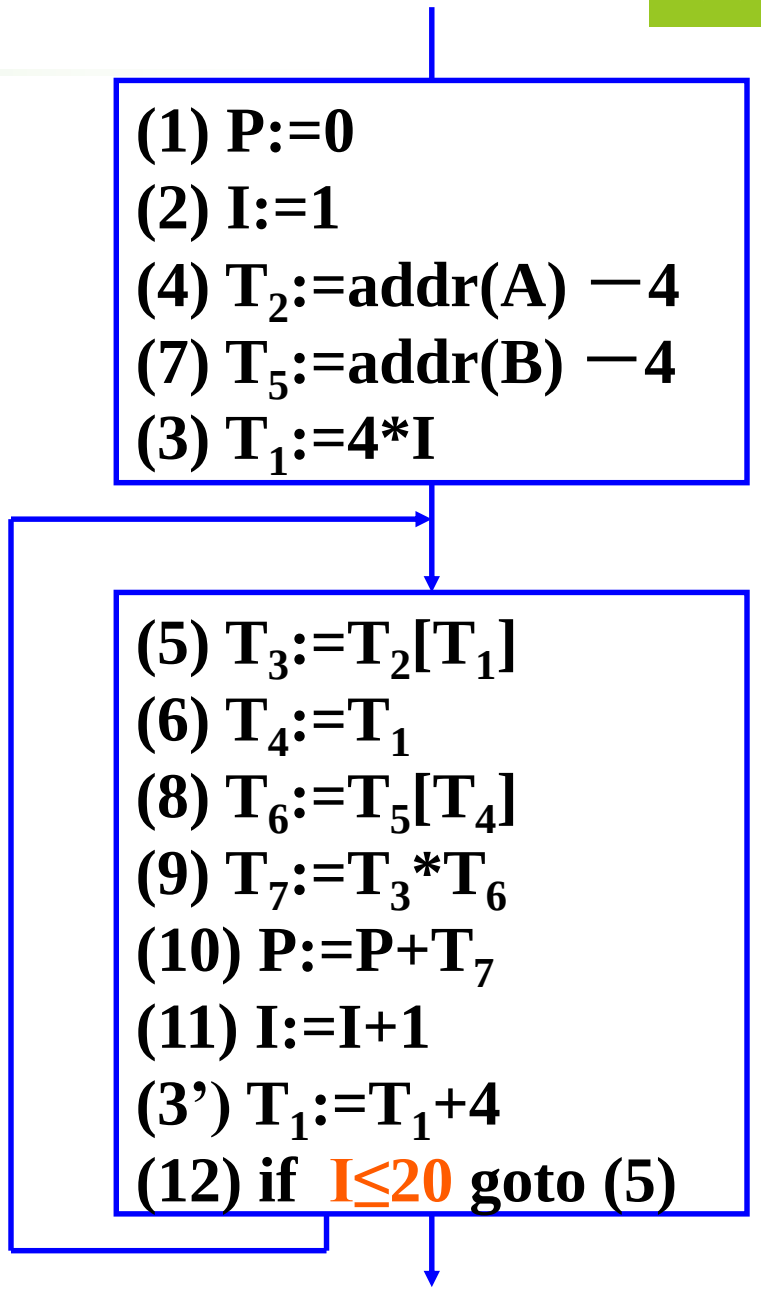
(2) 强度削弱


(1) P:=0
(2) I:=1
(4) T2:=addr(A) －4
(7) T5:=addr(B) －4
(3) T1:=4*I
(5) T3:=T2[T1]
(6) T4:=T1
(8) T6:=T5[T4]
(9) T7:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(12) If I≤20 goto (3)
中间代码段
乘法变加法
(1) P:=0
(2) I:=1
(4) T2:=addr(A) －4
(7) T5:=addr(B) －4
(3) T1:=4*I
(5) T3:=T2[T1]
(6) T4:=T1
(8) T6:=T5[T4]
(9) T7:=T3*T6
(10) P:=P+T7
(11) I:=I+1
(3') T1:=T1+4
(12) if I≤20 goto (5)
强度削弱

2. 优化技术

(3) 变换循环控制条件[Loop Condition Transformation]

- 将循环控制条件更换为**等价**的另一个循环控制条件
- **变换后可以删减代码或减少执行次数**
  - ✓ I和T1始终保持T1:=4*I的线性关系
  - ✓ 这样把(12)的循环控制条件 I≤20变换成T1≤80，程序的运行结果不变



2. 优化技术

(3) 变换循环控制条件