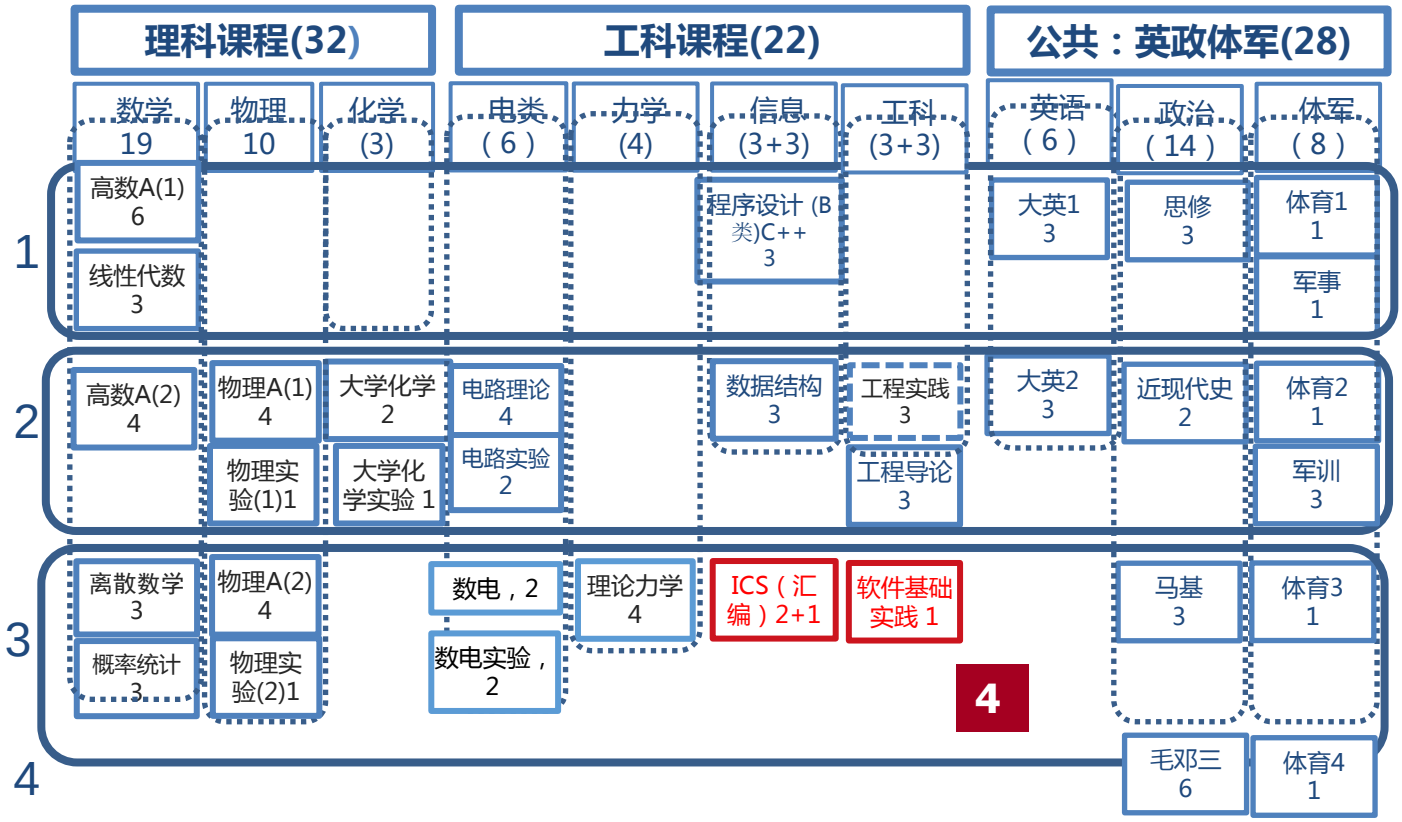
循序渐进多课程讲解
编译系统技术、原理和实现

上海交通大学软件学院
臧斌宇

不得不面对的教学改革



分类	学分
思政	14
军事	4
英语	6
体育	4
理科	32
工科	22
通识	12
个性化	6
小计	100
专业	58
交叉模块	6
总学分	164

核心课与方向课

核心课

- → 所有学生必修的课程
- → 分布第二与第三学年的四个学期

方向课

- → 学生必须选择一个方向
- → 学生必须修读该方向的所有课程
- → 目前分三个方向（系统软件、数字媒体、信息系统）

编译课程是方向课

- → 如何让所有学生都具备基础的编译知识

课程体系

学年	上学期	下学期
一	高等数学(1) 线性代数 程序设计思想与方法（C++）	高等数学(2) 数据结构
二	概率统计 离散数学 软件基础实践 计算机系统基础(汇编)	计算机系统基础(组成) 计算机系统基础(系统软件) 算法设计与实习 数据库原理与技术 互联网应用开发技术
三	计算机系统工程 软件工程 机器学习 编译原理与技术	软件测试 数字系统设计 操作系统 云计算系统设计与开发

软件基础实践

掌握基本的编译技术

自顶向下分析和表达式处理

软件基础实践

目标

- → 培养学生程序设计能力
  - mini-basic语言解释器（1200行C++代码）
  - 掌握OO的基本概念
  - 熟悉visual studio开发调试工具
- → 完成从大平台到专业的转变

学分

- → 1学分，2学时/周

软件基础实践

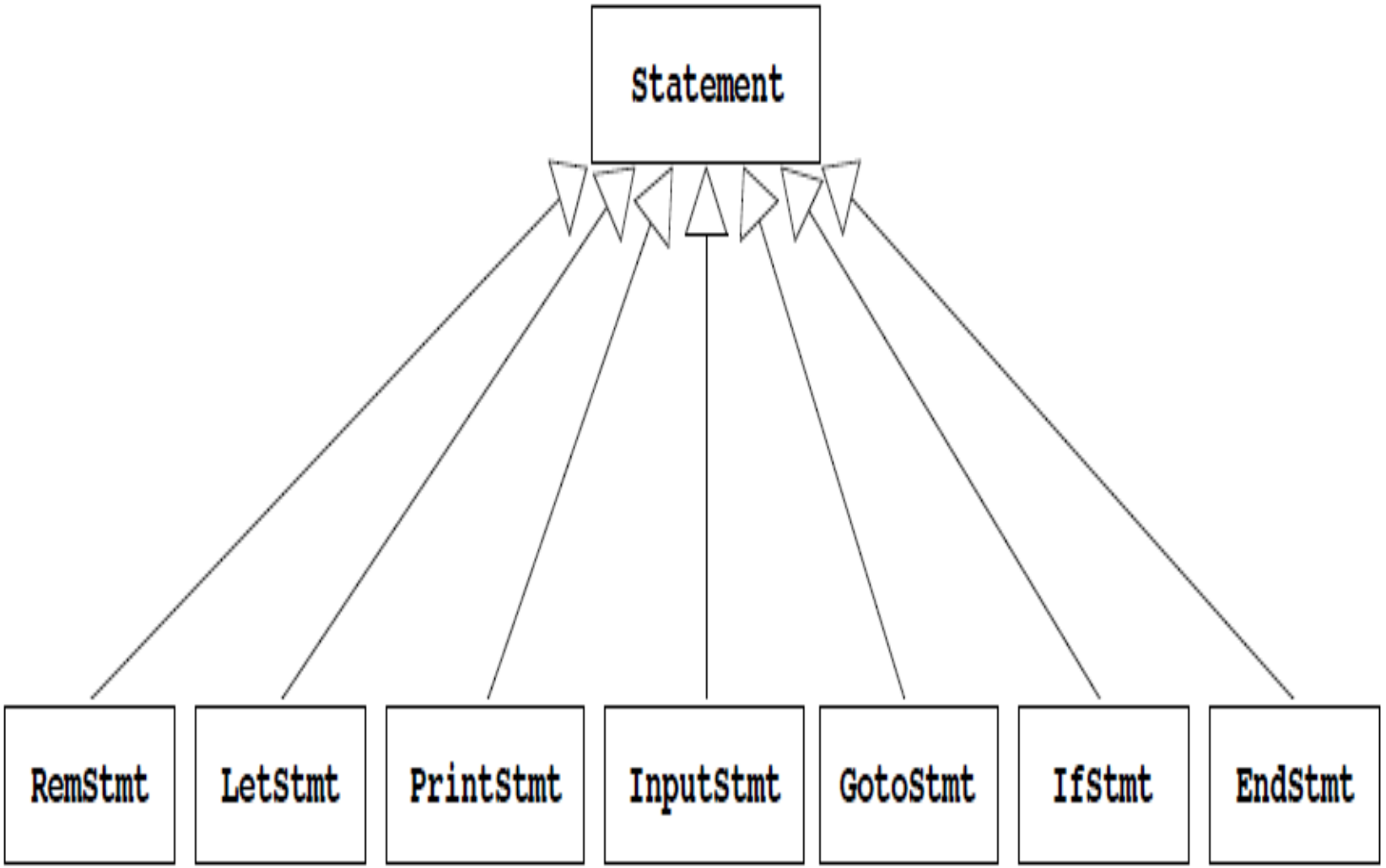
基本内容

- Basic编辑器（除了run之外的所有命令）
- 解释器
  - 语句自顶向下方法分析
  - 表达式（只支持+，-，*，/，以及括号），中缀变后缀处理

扩展内容

- 表达式支持右结合的幂次运算符↑、支持单目运算-
  - 今后在讲解shift/reduce冲突解决时，可以回头看
- 支持数组
- 支持函数调用
- 支持预定义函数调用

Minimal Basic



计算机系统基础（汇编）

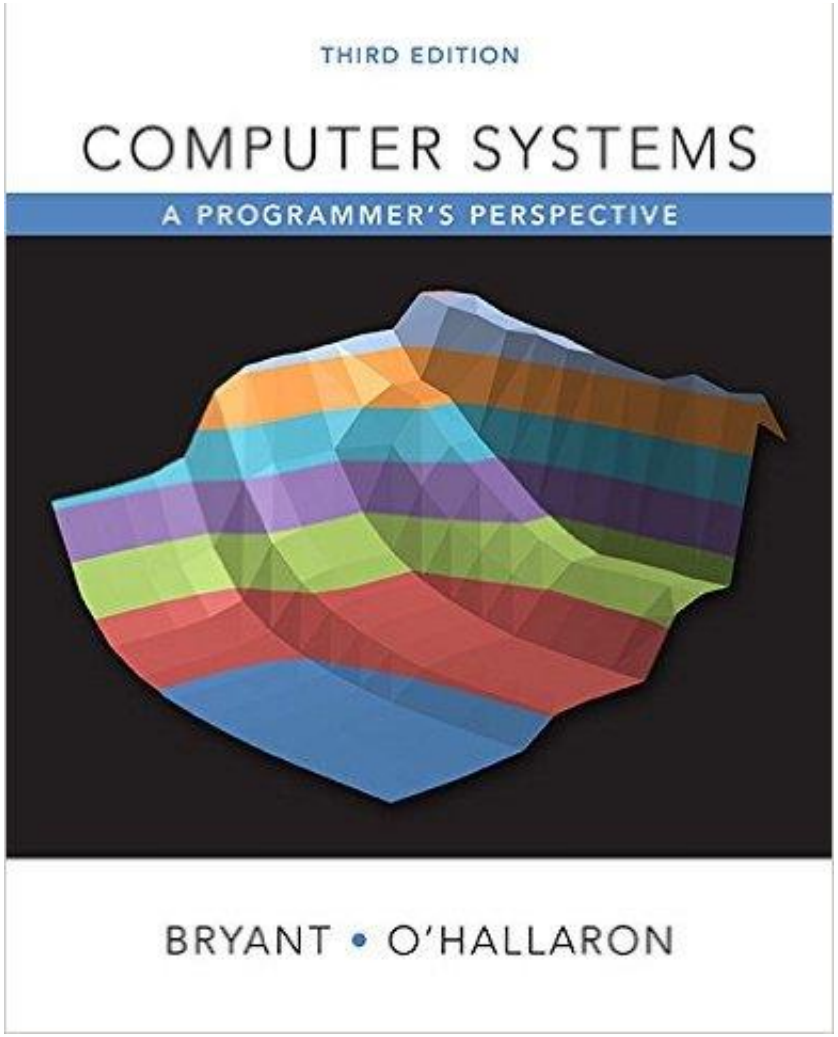
理解C程序和汇编程序的对应

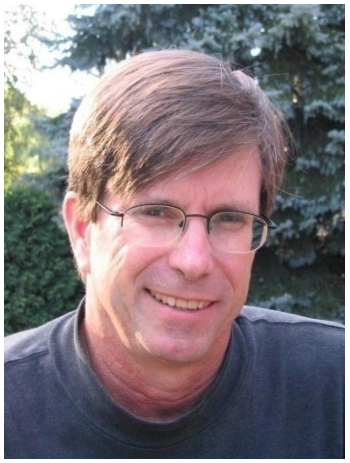
编译的输入和输出

系统类课程重构


计算机系统基础
计算机系统工程
从程序员的角度
从设计者的角度
计算机组成
计算机系统
计算机网络
专业内系统类
的通识教育
体系结构
汇编语言

CMU计算机学院教授
David R. O'Hallaron
Randal E. Bryant





- 1998年在CMU开设
- 2002年正式出版教材
- 涵盖了计算机系统领域的广泛内容
- 2011年第2版
- 2016年第3版

基本内容

学分：2+1（2课时上课/每周，2课时习题上机/双周）

数的表示、存储与运算

- → 原码、补码、浮点
- → 位运算、算术逻辑运算
- → 虚拟存储器（大、小端）

机器语言的执行模型

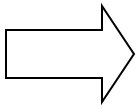
- → 汇编码和目标码
- → 预处理、编译、汇编、链接、加载
- → 指令的执行
  - ◆ 冯诺依曼结构
  - ◆ 指令的顺序执行

基本内容

- 从C 到汇编
  - → 数据移动
    - ◆ 寄存器
    - ◆ 寻址模式、mov指令、硬件栈与栈操作
    - ◆ 从指针到地址
  - → 算术逻辑运算指令
    - 从表达式到算术逻辑指令
  - → 控制结构
  - → 函数调用的实现
  - → Data Layout
    - ◆ 数组、结构与联合

如何翻译if语句

```
if ( test-expr )
    then-statement
else
    else-statement
```



```
    t = test-expr ;
    if ( t )
        goto true ;
    else-statement
    goto done
true:
    then-statement
done:
```

如何翻译if语句

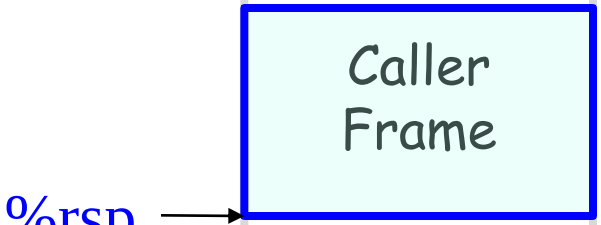
```
long lt_cnt = 0 ;
long ge_cnt = 0 ;
long absdiff_se(long x, long y)
{
    long result ;
    if  (x < y) {
    lt_cnt++
        return y – x;
    }
    else {
    ge_cnt++ ;
        return x – y;
    }
}
```

```
1.   long gotodiff_se(long x, long y)
2.   {
3.       long result ;
4.       if  (x >= y)
5.           goto x_ge_y ;
6.       lt_cnt++ ;
7.       result = y – x ;
8.       return result ;
9.    x_ge_y:
10.       ge_cnt++ ;
11.       result = y – x;
12.       return result;
13.  }
```

如何翻译if语句

```
1.  absdiff_se:
2.          cmpq %rsi, %rdi          Compare x : y
3.          jge      .L2             if  x >= y goto x_ge_y
4.          addq     $1, lt_cnt      lt_cnt++
5.          movq  %rsi, %rax         Copy y
6.          subq   %rdi, %rax        result = y - x
7.          ret                      Return
8.  .L2                           x_ge_y:
9.          addq     $1, gt_cnt      gt_cnt++
10.         movq  %rdi, %rax         Copy x
11.         subq    %rsi, %rax       result = x - y
12.         ret                      Return
```

如何处理函数调用



%rsp →

如何处理函数调用

1. Save caller-save registers

   (%rax, %rdx, %rcx, …)

如何处理函数调用

1. Save caller-save registers

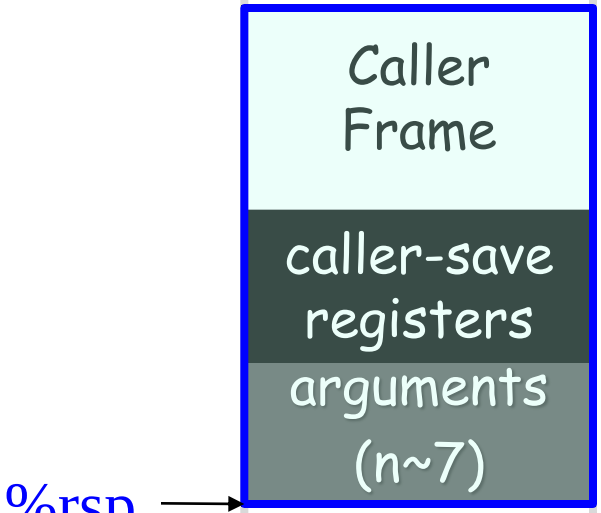
   (%rax, %rdx, %rcx, …)

2. Put first actual arguments

   to fixed registers

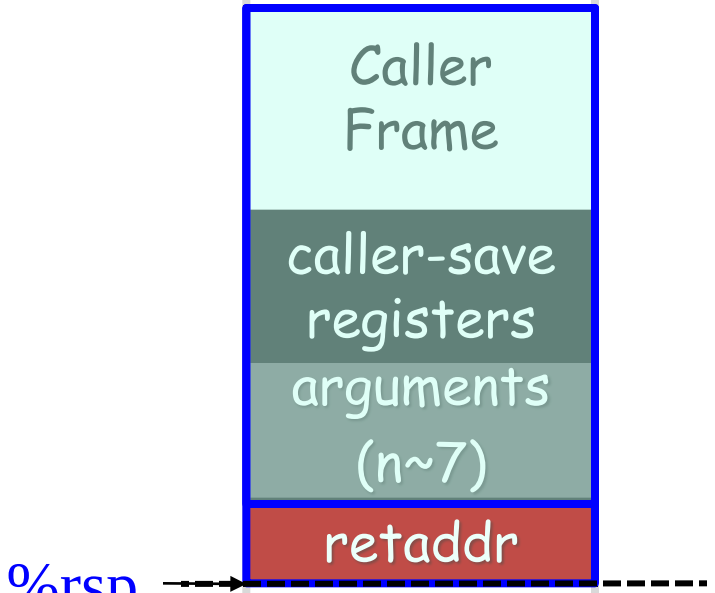
   Push rest actual arguments

   to stack from right to left



如何处理函数调用

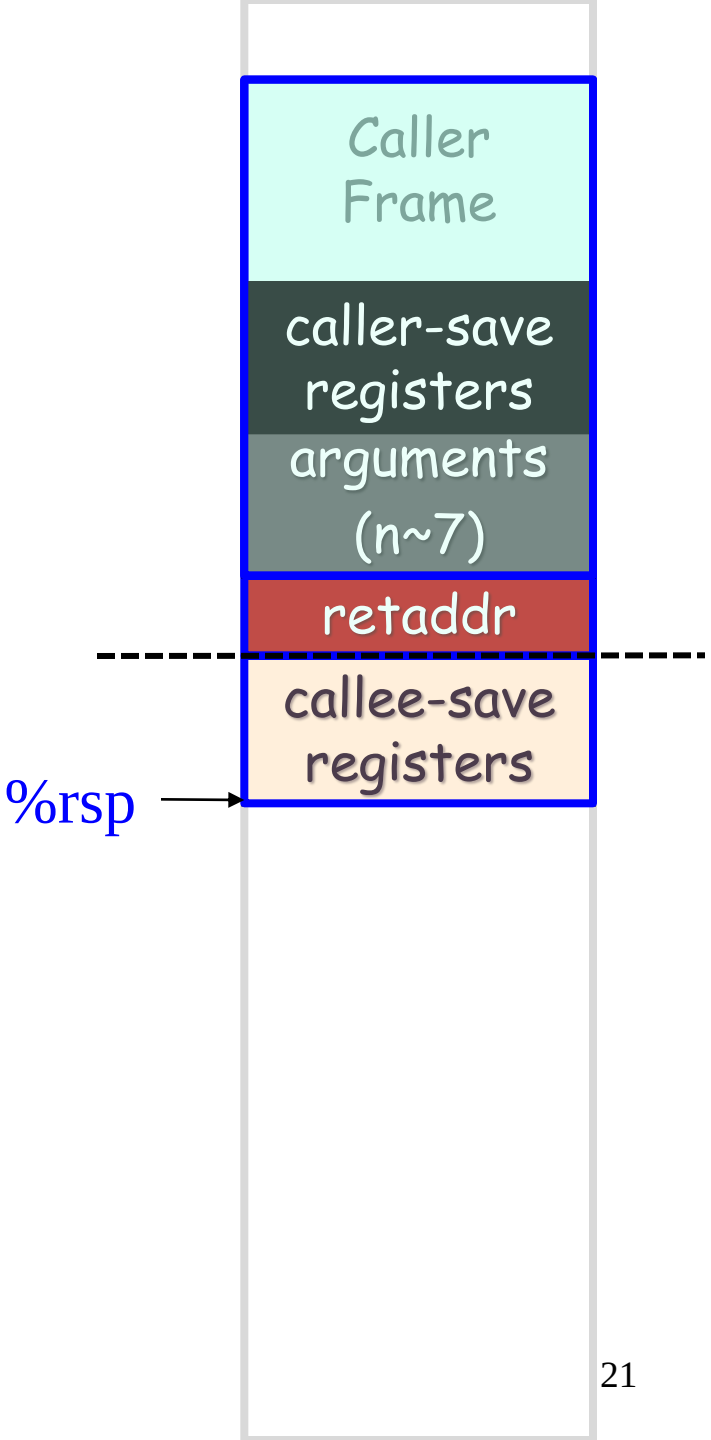
3. Call instruction
   - Save return address
   - Transfer control to callee



如何处理函数调用

4. Save callee-save registers

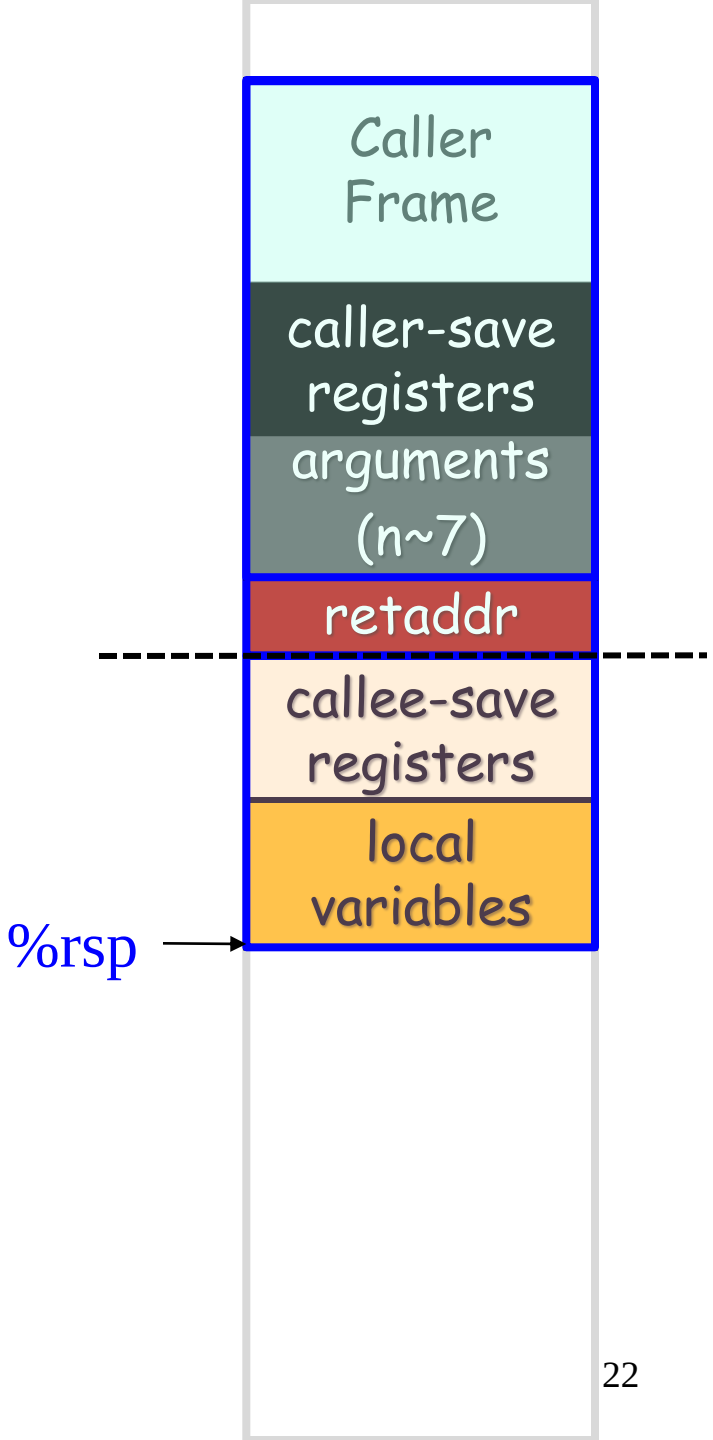
(%rbx, %rbp, . . .)



如何处理函数调用

4. Save callee-save registers (%rbx, %rbp, . . .)

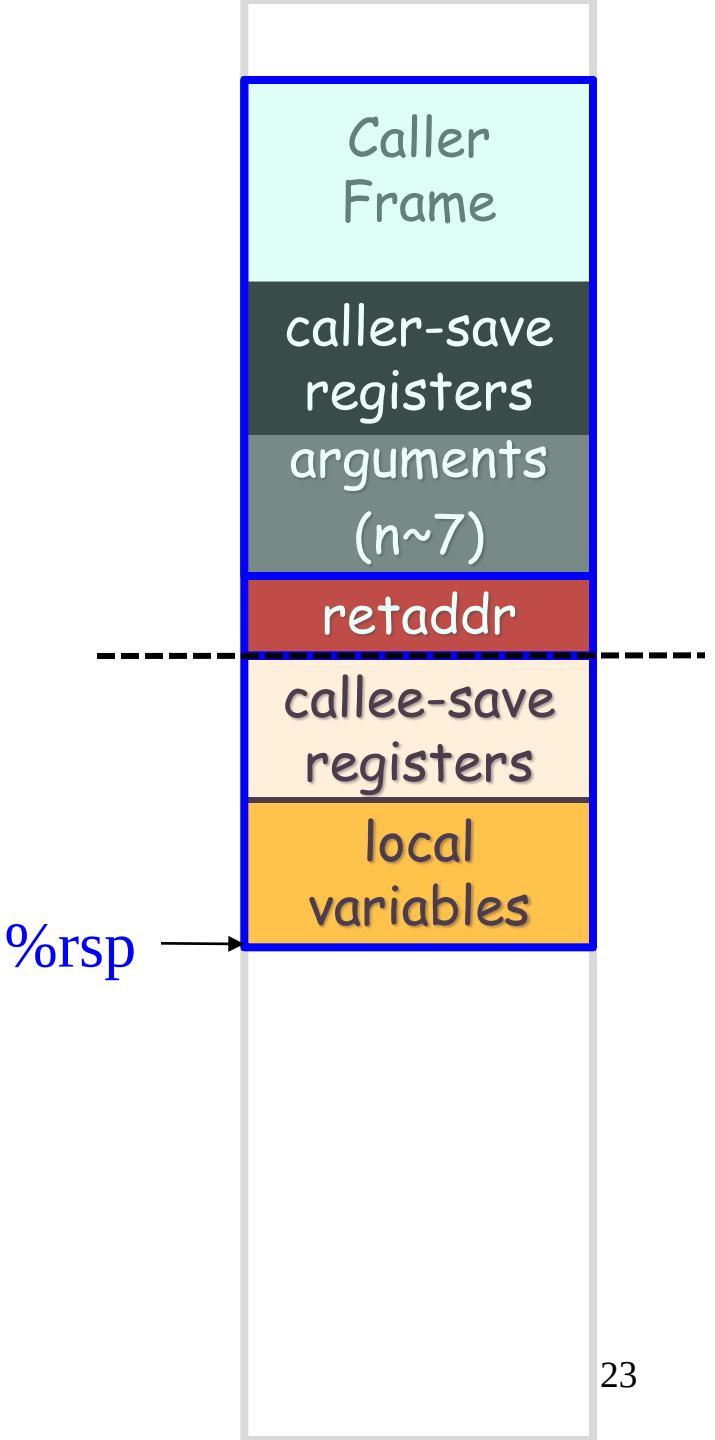
5. Allocate space for local variables



如何处理函数调用

. . .

n-3. save return value in %rax

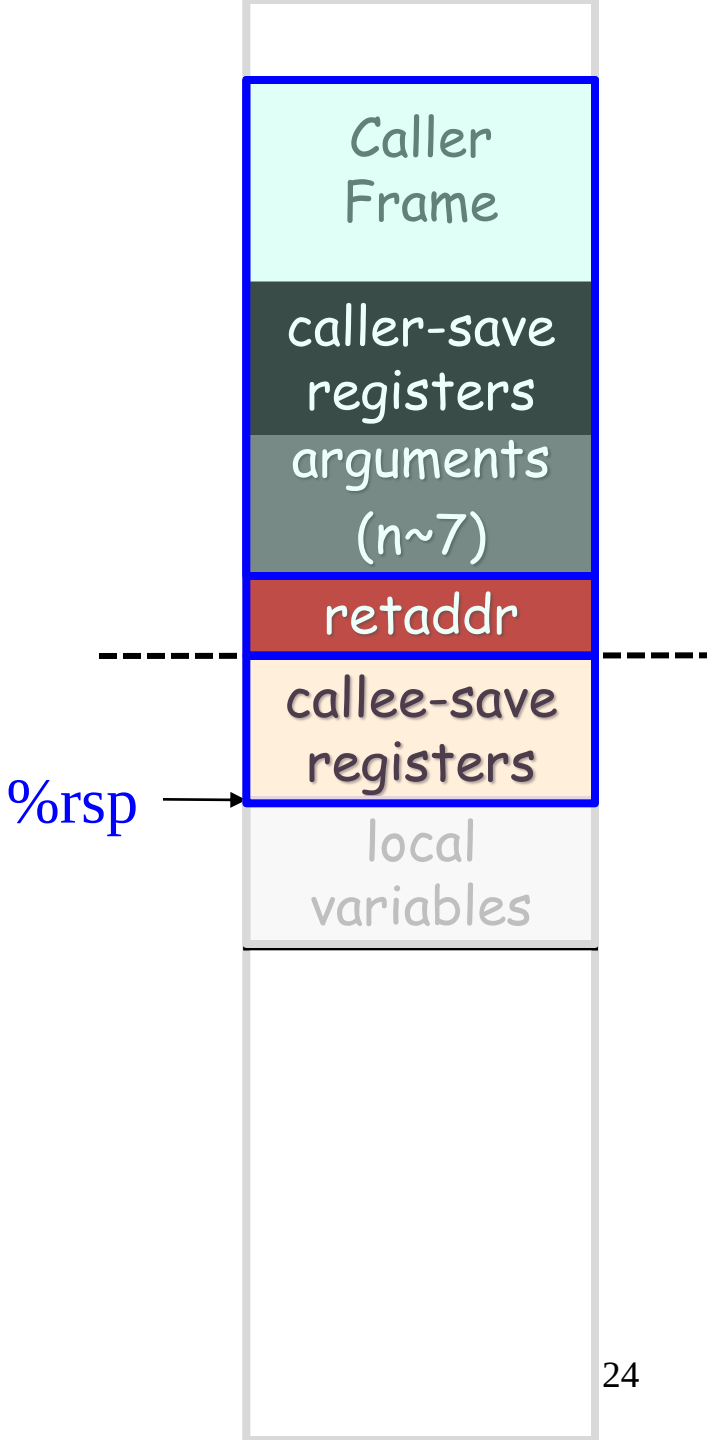


如何处理函数调用

. . .

n-3. save return value in %rax

n-2. de-allocate local variable



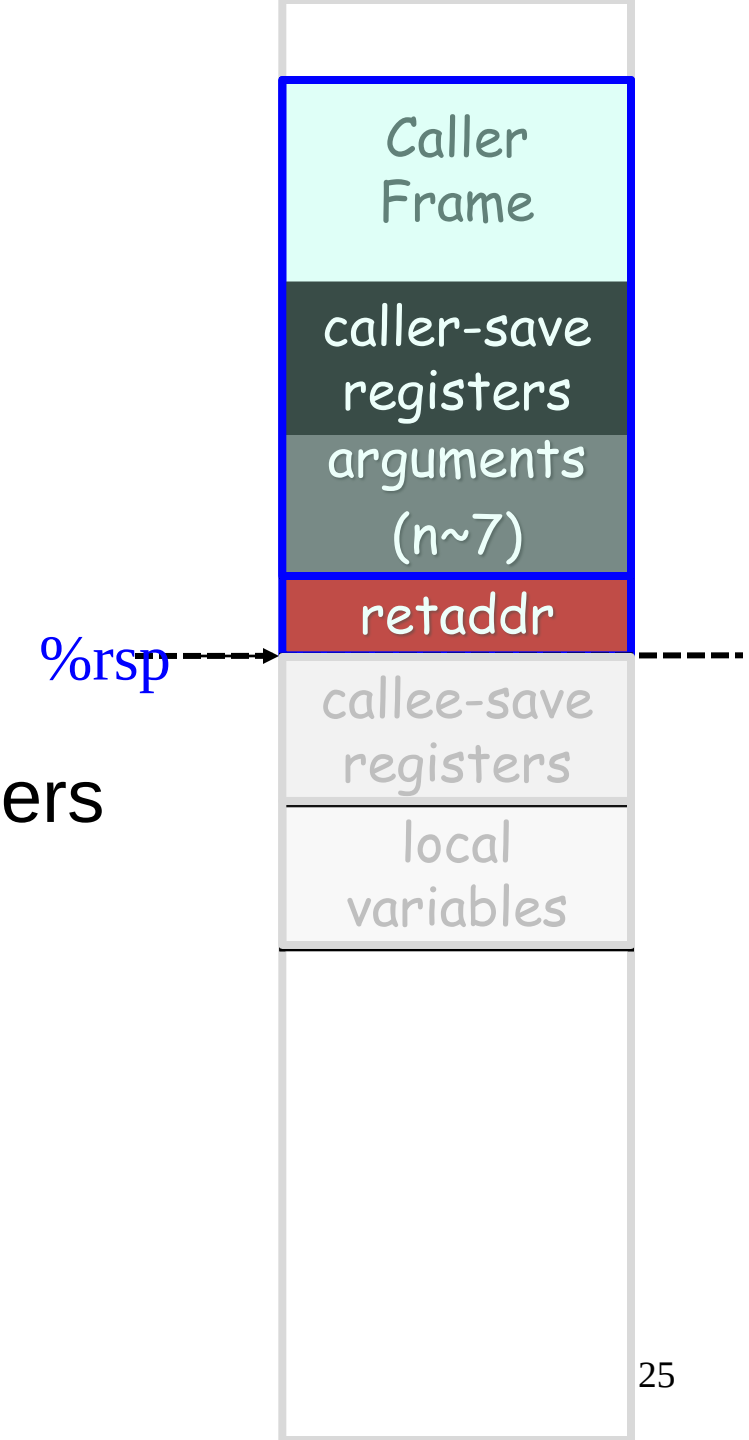
如何处理函数调用

. . .

n-3. save return value in %rax

n-2. de-allocate local variable

n-1. Restore callee-save registers



如何处理函数调用

. . .

n-3. save return value in %rax

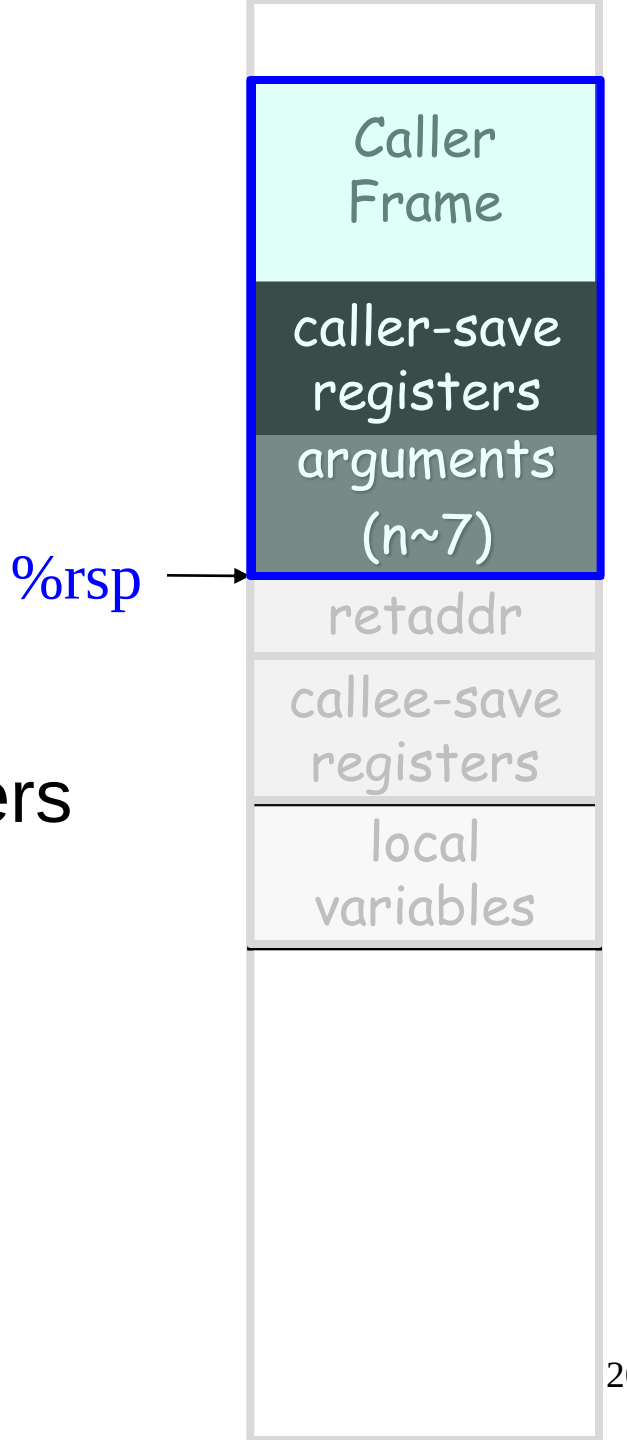
n-2. de-allocate local variable

n-1. Restore callee-save registers

n. Ret instruction

– pop return address

– Transfer control to caller

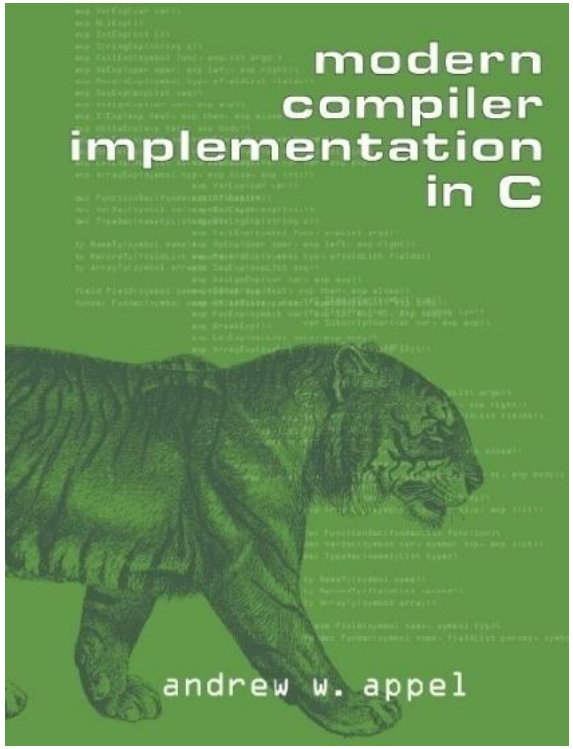


编译原理与技术

开发编译系统

教材

书名：Modern Compiler Implementation in C
作者：Andrew W. Appel
大学：Princeton



课程基本情况

方向课
- → 部分学生修读（30%-60%）

目标
- → 完成tiger语言编译器（4000+C代码）
- → 生成IA-64/Linux环境下的汇编代码

授课
- → 48学时，12周*4学时/周

实验
- → 实现tiger语言

成绩
- → 实验60%+期中20%+期末20%

编译大纲

引论（2）

- → 简介
- → Tiger语言与树结构

前端（20）

- → 词法分析（4）
- → CFG(2)
- → Top-down(2）
- → LR(4)
- → Yacc、出错处理（2）
- → AST、属性文法（2）
- → 符号表（2）
- → 语义分析（2）

中端（14）

- → 活跃记录（4）
- → 中间代码（4）
- → 基本块和轨迹（2）
- → 指令选择（4）

后端（8）

- → 活跃分析（2）
- → 寄存器分配（6）
- → 汇编代码生成（2）

理解和设计
语言特征

Tiger Language

类函数式语言

- 结构体：指针
- 库函数
- 函数嵌套定义

Merge.tig

```
let    type any = {any : int}
       var buffer := getchar()
 function readint(any: any) : int  =
  let    var i := 0
     function isdigit(s: string) : int =
          ord(buffer)>=ord(“0”) & ord(buffer)<=ord(“9”)
  in while buffer=“ “ | buffer=“\n” do buffer := getchar()
     any.any := isdigit(buffer);
     while isdigit(buffer)
        do ( i := i*10+ord(buffer)-ord(“0”);
             buffer := getchar() )
     i
 end
```

Tiger Language

类函数式语言

- 空指针
- 结构体递归定义
- 数组：指针
- **函数递归调用**

Merge.tig

```
    type list = {first: int, rest: list}

    ……
    function printlist(l: list) =
      if   l=nil then print("\n")
      else (pintint((l.first);  print(" ");  printlist(l.rest))

    /* BODY OF MAIN PROGRAM */
    in printlist(merge(readlist(), readlist()))
  end

  var N := 8
  type intArray = array of int
  var row := intArray [N] of 0
```

As simple as possible, but not simpler

Only base type of int and string

Only one dimensional array

Nested functions

- → Escape analysis

Structured l-value

- → No record or array variables
- → Only pointers to heap

形式语言和自动机

仅仅是工具的一部分

词法分析需要分割输入并分类

R = Keyword | Identifier | Number | …

= R_1 | R_2 | R_3 | …

If $s \in L(R)$ then s is a lexeme

→ Furthermore $s \in L(R)$ for some "R_i"

→ This "R_i" determines the token that is reported

分类产生不确定性

R = Whitespace | Integer | Identifier | '+'

Parse "foo+3"

- → "f" matches R, more precisely Identifier
- → But also "fo" matches R, and "foo", but not "foo+"

How much input is used? What if

- → $x_1 \ldots x_i \in L(R)$ and also $x_1 \ldots x_K \in L(R)$

"最长匹配" 规则:

- → Pick the longest possible substring that matches R

最长匹配需要回溯

Last Final	Current State	Current Input	Accept Action
0	1	⌶if --not-a-com	
2	2	⌶i⌷f --not-a-com	
3	3	⌶if⌷ --not-a-com	
3	0	⌶if⌷ --not-a-com	*return* IF
0	1	⌶if --not-a-com	
12	12	⌶if⌷ --not-a-com	
12	0	⌶if⌷ -not-a-com	*found white space; resume*
0	1	⌶if -not-a-com	
9	9	⌶if -⌷not-a-com	
9	10	⌶if -⌷n⌷ot-a-com	
9	10	⌶if -⌷no⌷t-a-com	
9	10	⌶if -⌷not⌷-a-com	
9	10	⌶if -⌷not-⌷a-com	
9	0	⌶if -⌷not-a⌷-com	*error, illegal token '-'; resume*
0	1	⌶if -not-a-com	
9	9	⌶if -⌷not-a-com	
9	0	⌶if -⌷not-a-com	*error, illegal token '-'; resume*

分类产生不确定性

R = Whitespace | ‘new’ | Integer | Identifier

Parse “new foo”

→ “new” matches R, more precisely ‘new’

→ but also Identifier, which one do we pick?

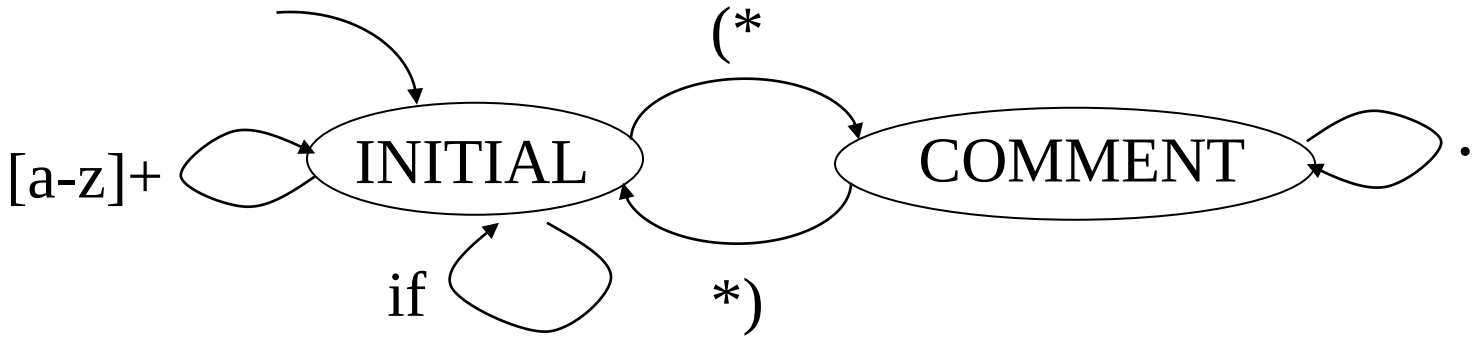
In general, if $x_1 \ldots x_i \in L(R_j)$ and $x_1 \ldots x_i \in L(R_k)$

“优先级”规则：

→ We must list ‘new’ before Identifier

Lex可以支持嵌套注解

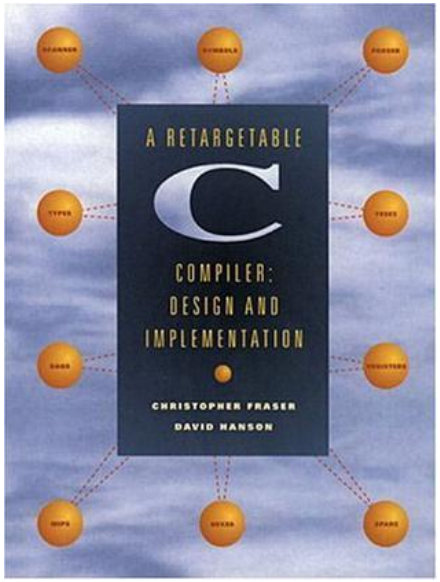
```
%Start INITIAL COMMENT
%%
<INITIAL>if              {ADJ; return IF;}
<INITIAL>[a-z]+          {ADJ; yylval.sval=String(yytext); return ID;}
<INITIAL>”(*”            {ADJ; BEGIN COMMENT;}
<INITIAL>.               {ADJ; EM_error(“illegal character”);}
<COMMENT>”*)”            {ADJ; BEGIN INITIAL;}
<COMMENT>.               {ADJ;}
.                        {BEGIN INITIAL; yyless(1);}
```



Lex可以支持嵌套注解

- Comments
  - May appear between any two tokens
  - Start with /* and end with */
  - **May be nested**

不花太多时间在LL上



LL适用于手工编写编译器

- → 在mini-basic解释器项目中已经练习

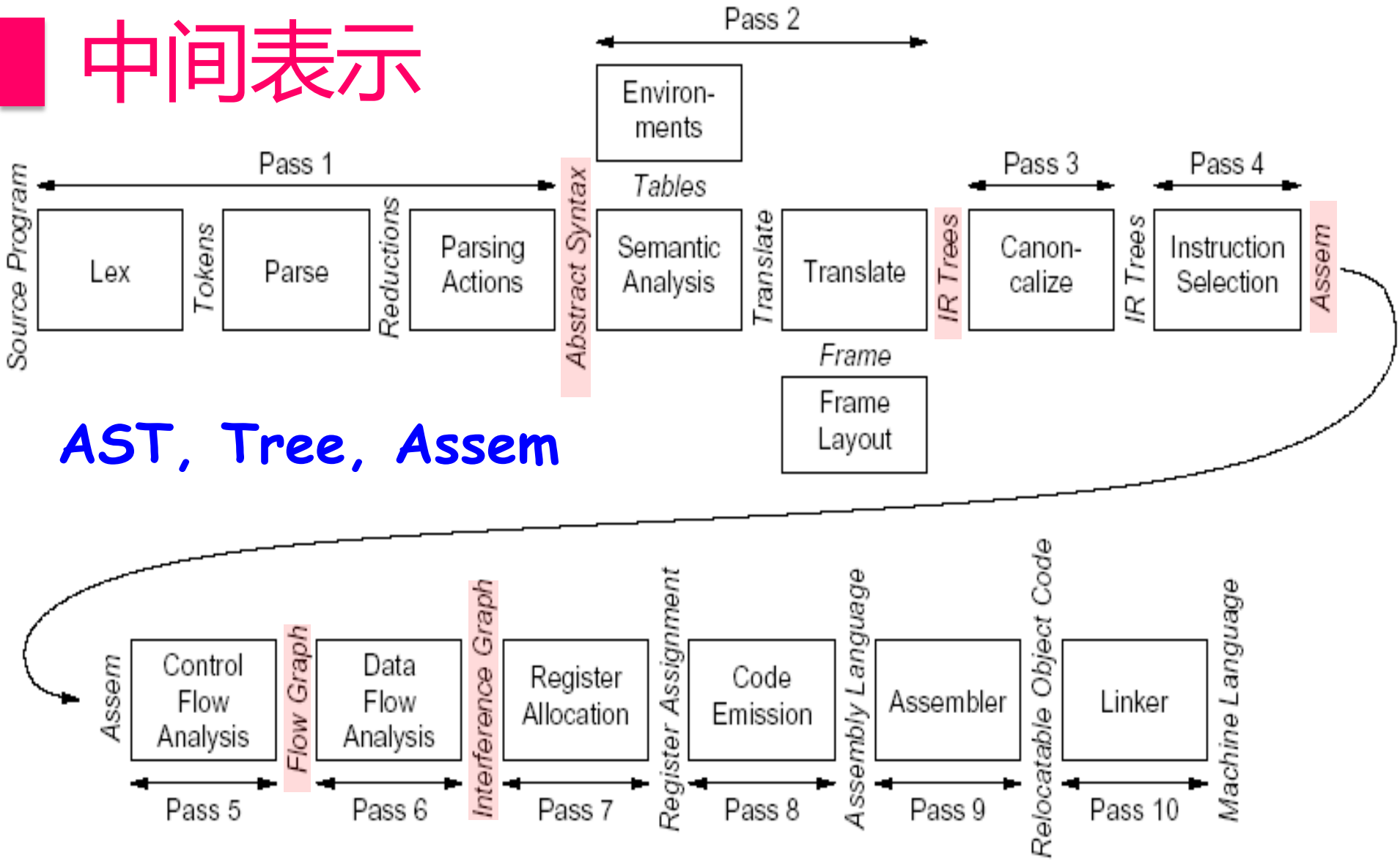
一般程序设计语言

- → 大部分语句以关键字开头（直接递归下降）
- → 表达式可以用算符优先
- → 消除左递归没有实际意义
- → A Retargetable C Compiler Design and Implementation

中间表示

很重要

中间表示


Pass 2
Environ-
ments
Pass 1
Tables
Pass 3
Pass 4
Source Program
Lex
Tokens
Parse
Reductions
Parsing
Actions
Abstract Syntax
Semantic
Analysis
Translate
Translate
IR Trees
Canon-
calize
IR Trees
Instruction
Selection
Assem
Frame
Frame
Layout
AST, Tree, Assem
Assem
Control
Flow
Analysis
Flow Graph
Data
Flow
Analysis
Interference Graph
Register
Allocation
Register Assignment
Code
Emission
Assembly Language
Assembler
Relocatable Object Code
Linker
Machine Language
Pass 5
Pass 6
Pass 7
Pass 8
Pass 9
Pass 10

与ICS不同的内容

View Shift：

→ 实参在寄存器中、虚参在内存中

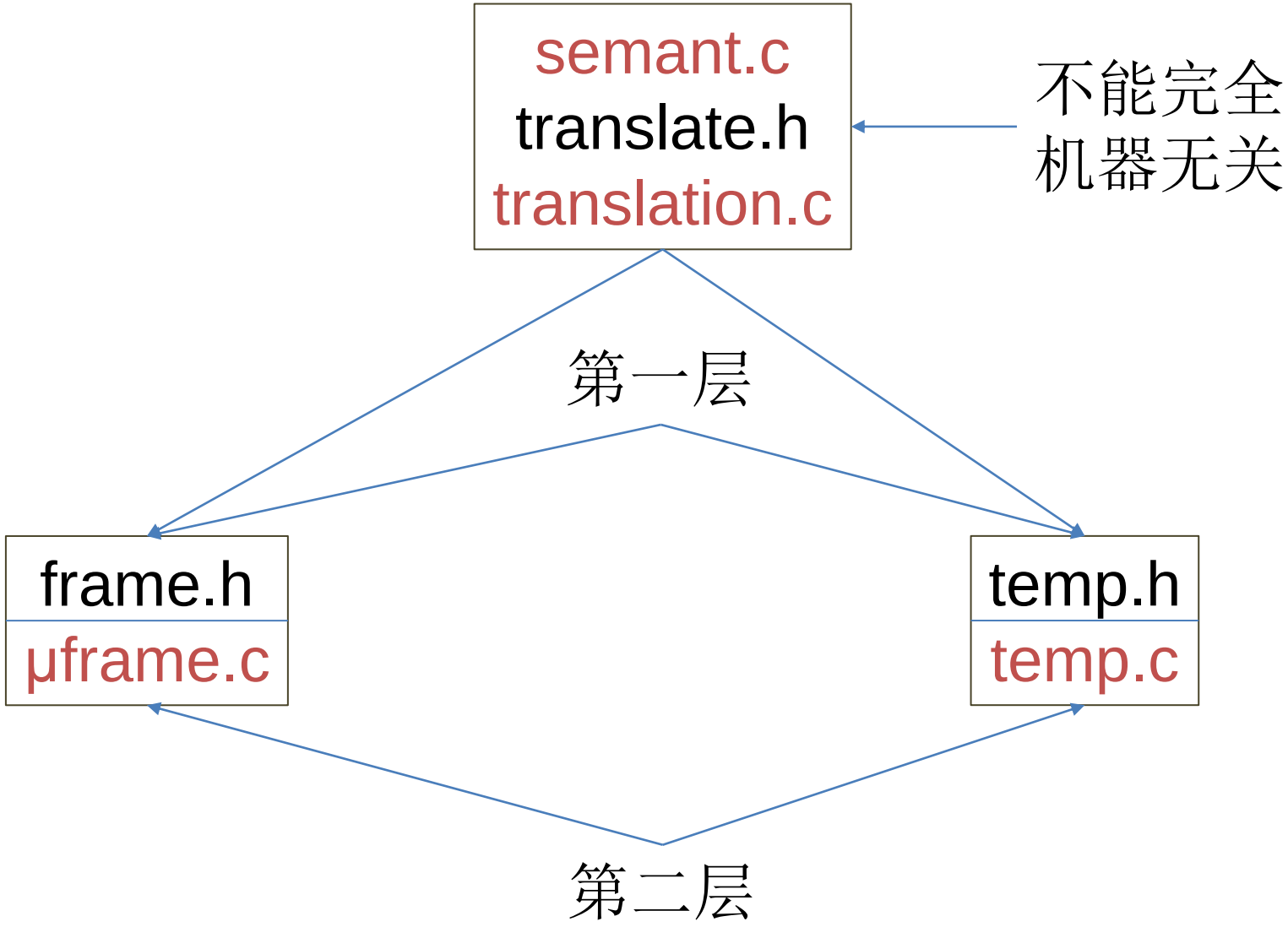
```
void caller()
{
    ……
    f(2) ;
    ……
}
```

```
void f(int x)
{
    ……
    p = &x ;
    ……
}
```

Static link：

→ 支持函数嵌套

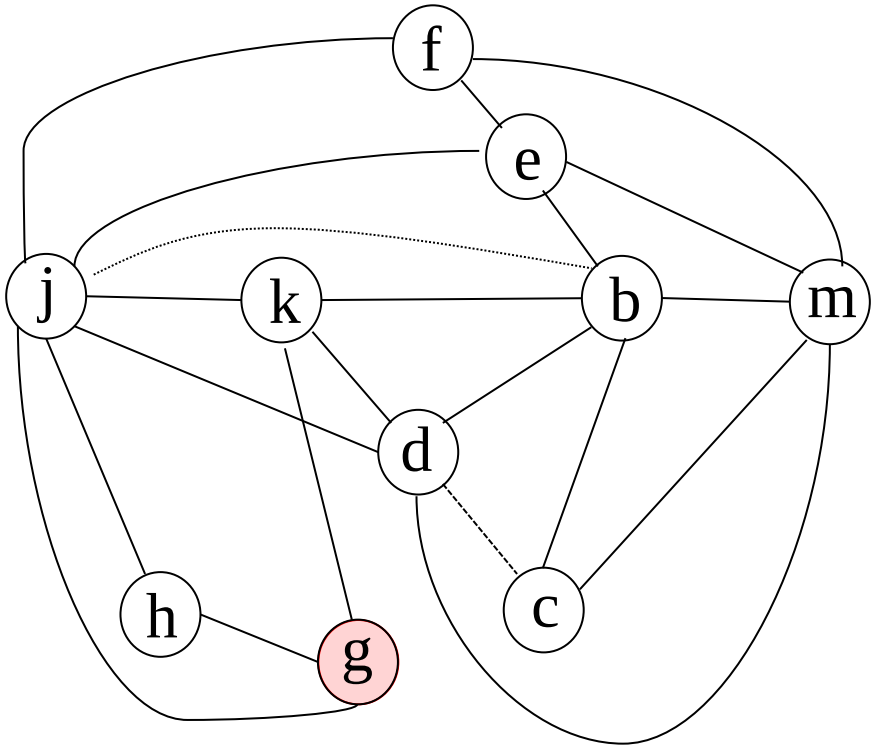
两层抽象支持多目标平台



寄存器分配
必须实现的优化

图着色（4色）

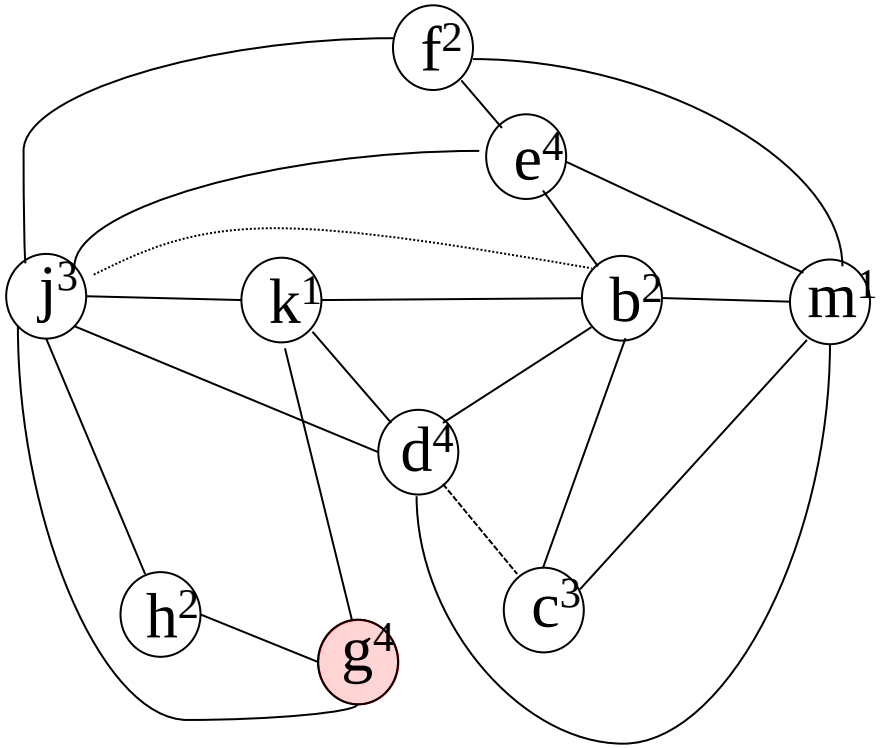
```
Live in: k  j
        g := mem[j+12]
        h := k-1
        f := g*h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e+8
        d := c
        k := m+4
        j := b
Live out d k j
```



图着色（4色）

Stack

m 1
c 3
b 3
e 4
d 4
j 3
f 2
k 1
h 2
g 4

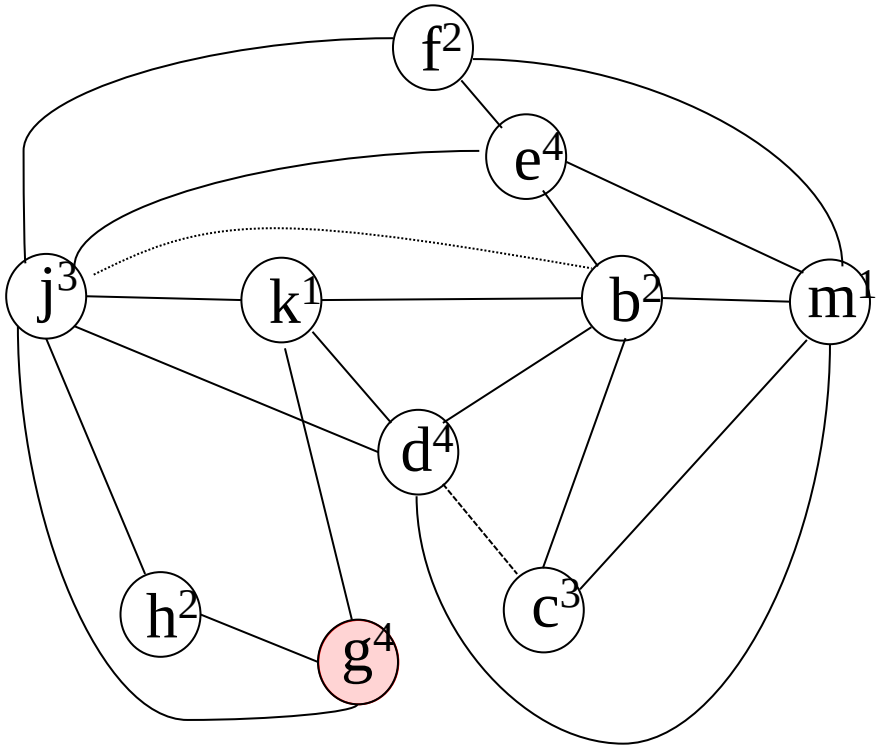


图着色（4色）

Live in: k j

```
        r4 := mem[r3+12]
        r2 := r1-1
        r2 := r4*r2
        r4 := mem[r3+8]
        r1 := mem[r3+16]
        r2 := mem[r2]
        r3 := r4+8
        r4 := r3
        r1 := r1+4
        r3 := r2
```

Live out d k j



其它问题

寄存器合并

→ Briggs, George

寄存器溢出

预着色寄存器

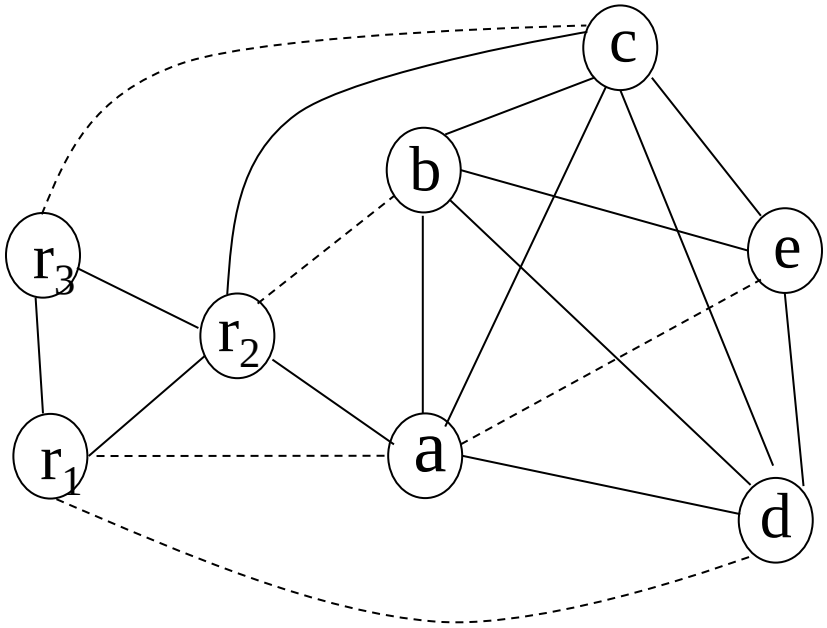
实例（K=3）


```
int f(int a, int b)
{
    int d=0;
    int e=a;
    do {
        d=d+b;
        e=e-1;
    } while (e>0) ;
    return d;
}
```

```
enter:  c ← r3
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop:   d ← d+b
        e ← e-1
        if e>0 goto loop
        r1 ← d
        r3 ← c
        return   (r1, r3 live out)
```

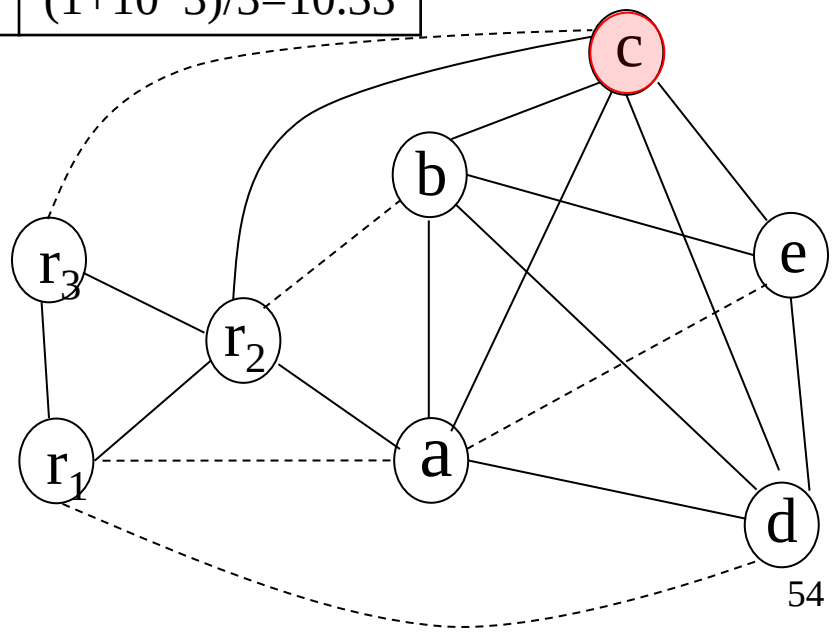
实例（K=3）

- No way to simplify or freeze
  - All non-precolored nodes have degree $\geq K$
- No way to coalesce
  - No less than K significant-degree nodes for coalesced nodes
- The only way is spilling



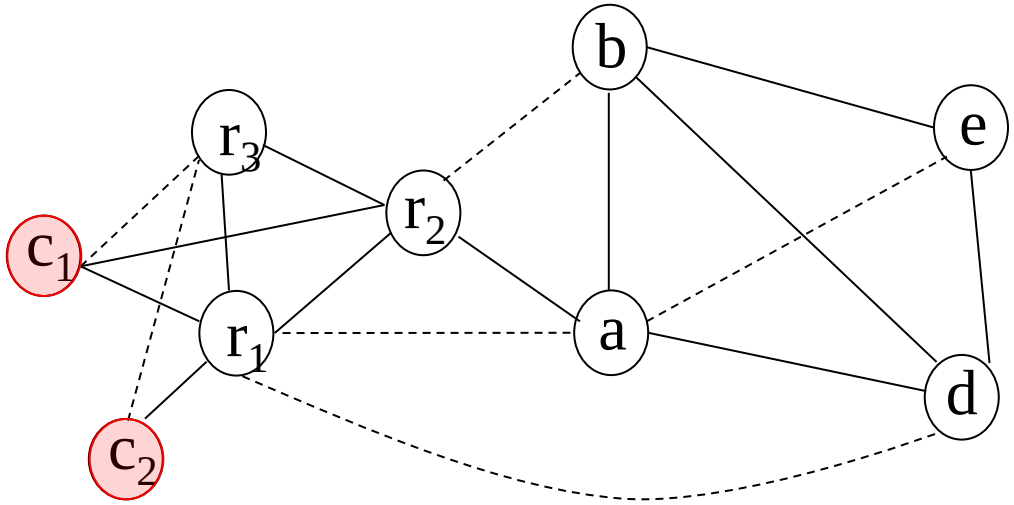
实例（K=3）

Node	Use+Def Outside loop	Use+Def inside loop	Degree	Spill priority
a	2	0	4	(2+10*0)/4=0.5
b	1	1	4	(1+10*1)/4=2.75
c	2	0	6	(2+10*0)/6=0.33
d	2	2	4	(2+10*2)/4=5.5
e	1	3	3	(1+10*3)/3=10.33



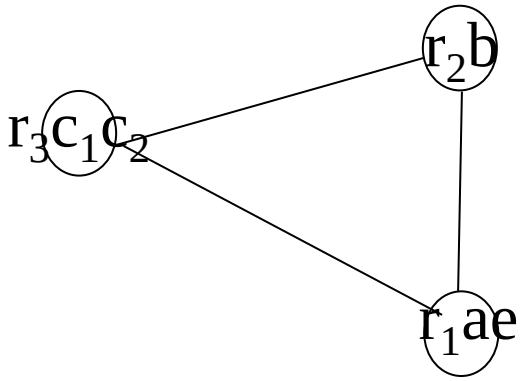
实例（K=3）

```
enter:  c1 ← r3
        M[c_loc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
        loop:   d ← d+b
        e ← e-1
        if e>0 goto loop
        r1 ← d
        c2 ← M[c_loc]
        r3 ← c2
        return (r1, r3 live out)
```



实例（K=3）

```
d        r3
a        r1
e        r1
b        r2
c        r3
enter:   M[c_loc] ← r3
         r3 ← 0
         loop:   r3 ← r3+r2
         r1 ← r1-1
         if r1>0 goto loop
         r1 ← r3
         r3 ← M[c_loc]
         return
```



高级特征
选讲

高级特征

垃圾回收
面向对象语言
函数式语言
多态类型
数据流分析
循环优化
静态单赋值
~~流水和调度~~
~~存储层次~~

考虑减少课程
增加实践课时
减少实现部分的讲解
增加高级特征的讲解

进一步的
改革

百家争鸣阶段

七十年代中期至八十年代中期

计算机技术和理论均不成熟

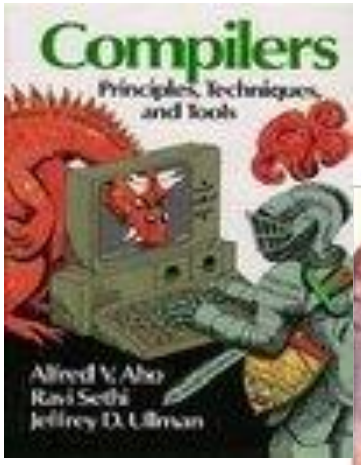
百科全书式的教学内容

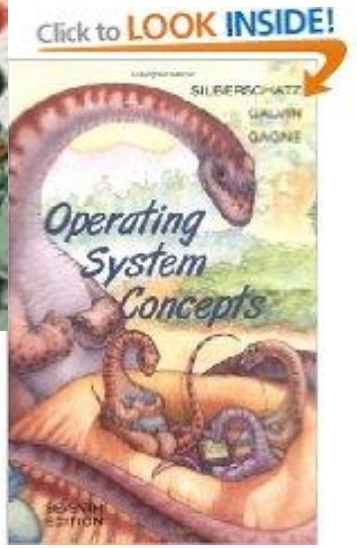
理论、概念为主、实践不足

典型教材

→ 龙书、恐龙书

中国目前的课程体系脱胎于此



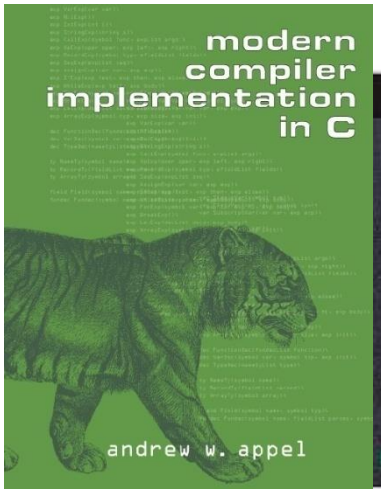


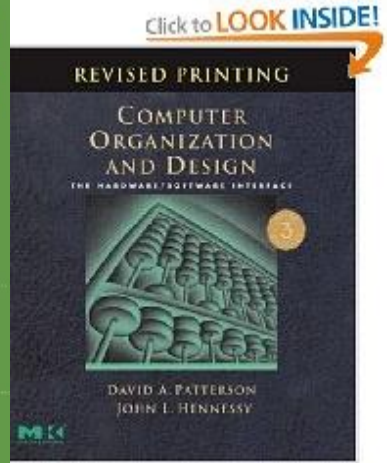
独尊儒术阶段

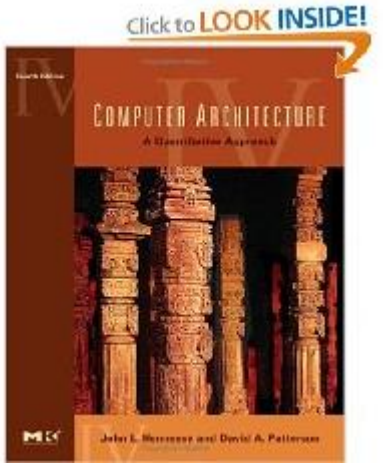
八十年代中期至上世纪末
单机技术逐渐成熟
有了主流的理论和平台
典型教材

- → H/P软硬件接口、量化方法
- → 虎书
- → 以实际系统为主线的教科书

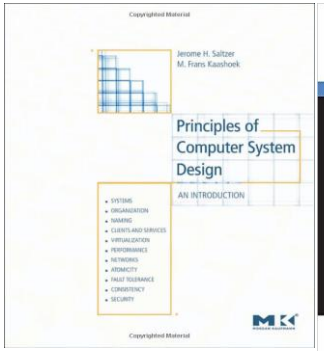
理论教学深和实验强度大

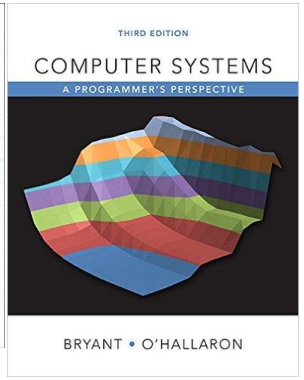






互联网阶段





始于本世纪

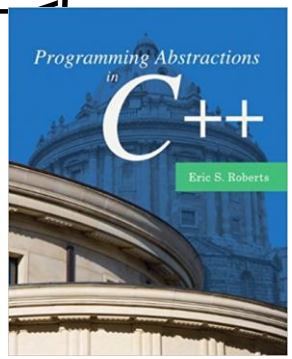
CMU、Stanford、MIT开始了课程重构

这个阶段还在不断演化

→ 互联网、云计算机、基于大数据分析的机器学习

三个抽象

→ 问题抽象、系统抽象、数据抽象



典型教材

→ Principles of Computer System Design

→ Programming Abstractions using C++

→ Computer Systems: A Programmer’s Perspective

→ 经典课程不再为单一系统，向更加综合方向发展

思考

虎书出版至今已二十年

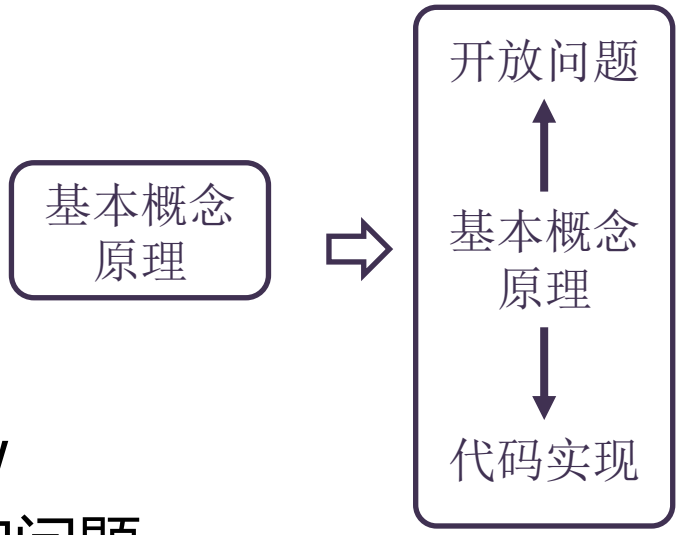
→ 第二阶段的产物、技术过于成熟

新时代编译应该教什么

从基础出发向上下延伸

→ 向下：落到实际的代码实现

→ 向上：接触更多开放问题



带着问题去思考

→ 从What，到Why，再到How

→ 从解决已知问题，到解决未知问题

谢谢！

byzang@sjtu.edu.cn