中间代码生成
Part6：符号表与声明语句翻译

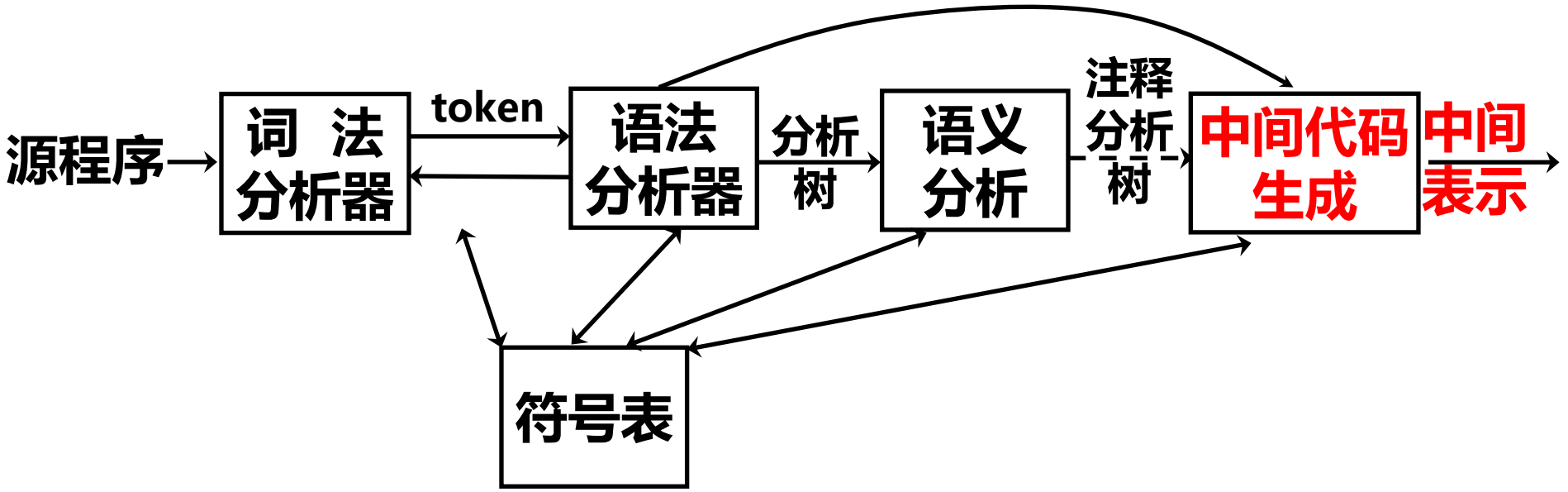
李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年10月30日

本节提纲



- **符号表的组织**
- **声明语句的翻译**
  - 存储空间计算、作用域、记录

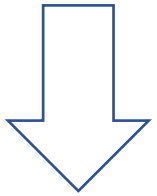
符号表 (Symbol table)

- **符号表的使用和修改伴随编译的全过程**
- **存储entity的各种信息**
  - 如variable names, function names, objects, classes, interfaces 等
  - 如类型信息、所占用内存空间、作用域
- **用于编译过程中的分析与合成**
  - 语义分析：如使用前声明检查、类型检查、确定作用域等
  - 合成：如类型表达式构造、内存空间分配等

符号表 (Symbol table)

代码片段:

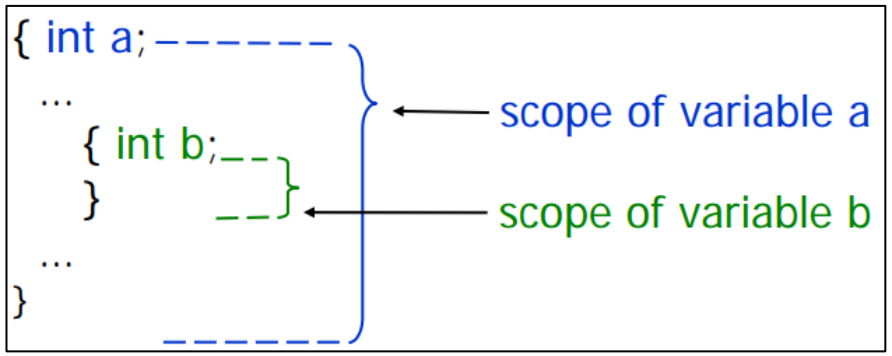
```
extern bool foo(auto int m, const int n);
const bool tmp;
```



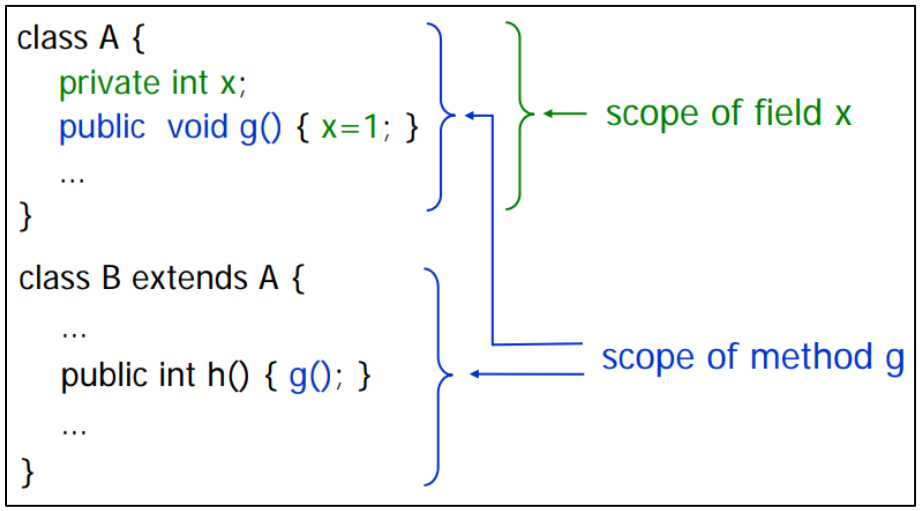
NAME	KIND	TYPE	OTHER
foo	fun	int x int → bool	extern
m	par	int	auto
n	par	int	const
tmp	var	bool	const

符号表

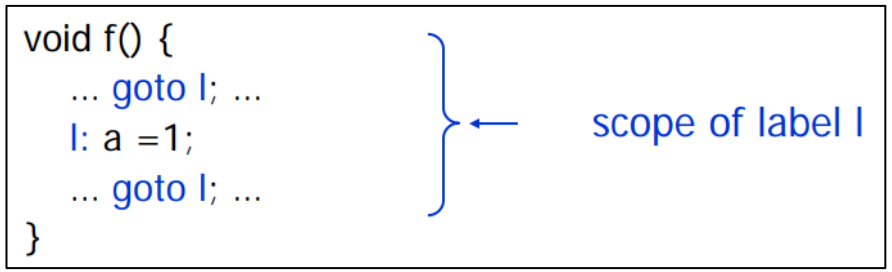
符号表——作用域



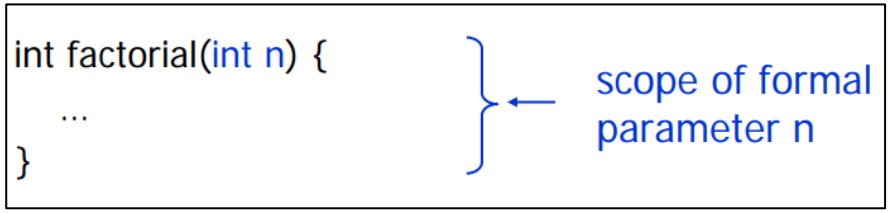
程序块中



对象中的field和methods



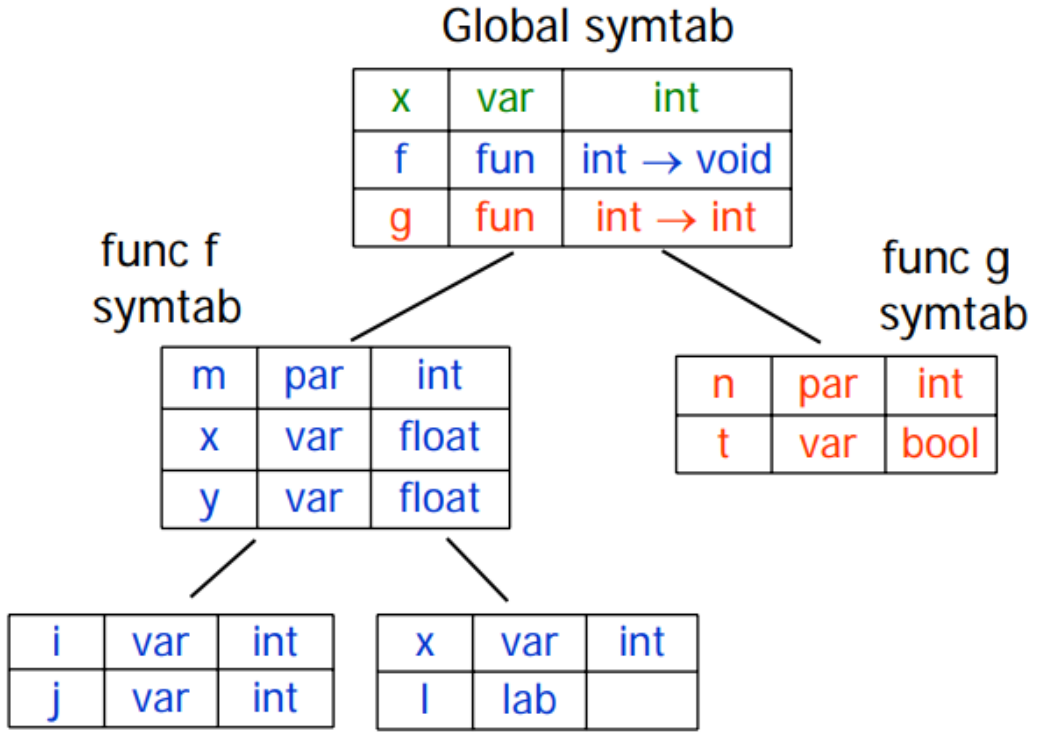
语句标号



过程或函数定义中的参数

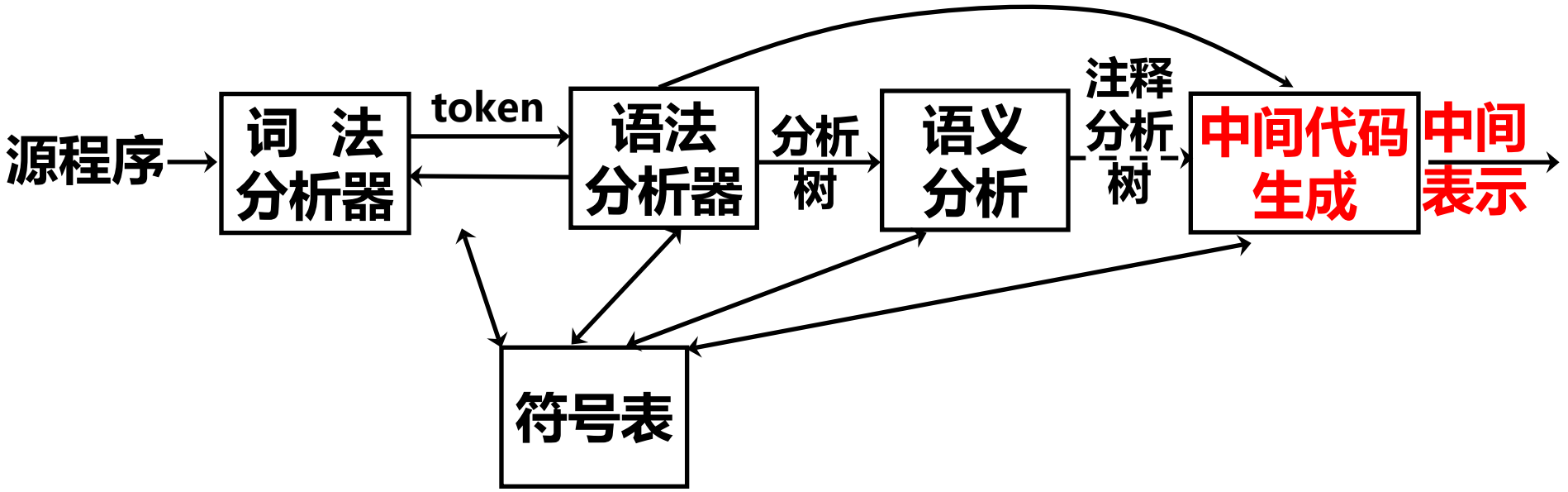
符号表 (Symbol table)

```
int x;

void f(int m) {
     float x, y;
     ...
    { int i, j; …; }
    { int x; l: …; }
}

int g(int n) {
    bool t;
    …;
}
```



注：l代表label

本节提纲



- 符号表的组织
- **声明语句的翻译**
  - 存储空间计算、作用域、记录

声明语句翻译的要点

- **分配存储单元**
  - 名字、类型、字宽、偏移
- **作用域的管理**
  - 过程调用
- **记录类型的管理**
- **不产生中间代码指令，但是要更新符号表**

声明语句的翻译

- 例：文法G_1如下：

  $P \rightarrow D ; S$

  $D \rightarrow D ; D$

  $D \rightarrow id : T$

  $T \rightarrow integer \mid real \mid array [num] of T_1 \mid \uparrow T_1$

声明语句的翻译

- 有关符号的属性

  T.type　　- 变量所具有的类型，如

  整型 INT

  实型　REAL

  数组类型 array（元素个数，元素类型）

  指针类型 pointer（所指对象类型）

  T.width　　- 该类型数据所占的字节数

  offset　- 变量的存储偏移地址

声明语句的翻译

T.type		T.width
整型	INT	4
实型	REAL	8
数组	array（num, T_1）	num.val * T_1.width
指针	pointer（T_1）	4
enter(name, type, offset)—将类型type和偏移offset填入符号表中name所在的表项。		

声明语句的翻译

计算被声明名字的类型和相对地址

P→ {*offset* = 0} D ; S

相对地址初始化为0

D→ D ; D

D→ id : T {*enter(* id.*lexeme, T.type, offset) ;*

offset = *offset* + *T.width* }

更新符号表信息

T→integer {*T.type* = *integer; T.width* = *4*}

T→real {*T.type* = *real; T.width* = *8*}

类型=>字宽

T→array [number] of T_1

{*T.type* = *array(num.val,* T_1*.type);*

T.width = *num.val* * T_1*.width*}

T→ ↑ T_1 {*T.type* = *pointer(*T_1*.type); T.width* = *4*}

声明语句翻译的要点

- 分配存储单元
  - 名字、类型、字宽、偏移
- **作用域的管理**
  - 过程调用
- **记录类型的管理**
- **不产生中间代码指令，但是要更新符号表**

允许自定义过程时的翻译

- **所讨论语言的文法**

  $P \rightarrow D; S$

  $D \rightarrow D\ ;\ D \mid \text{id} : T \mid$

  proc id ; D ; S

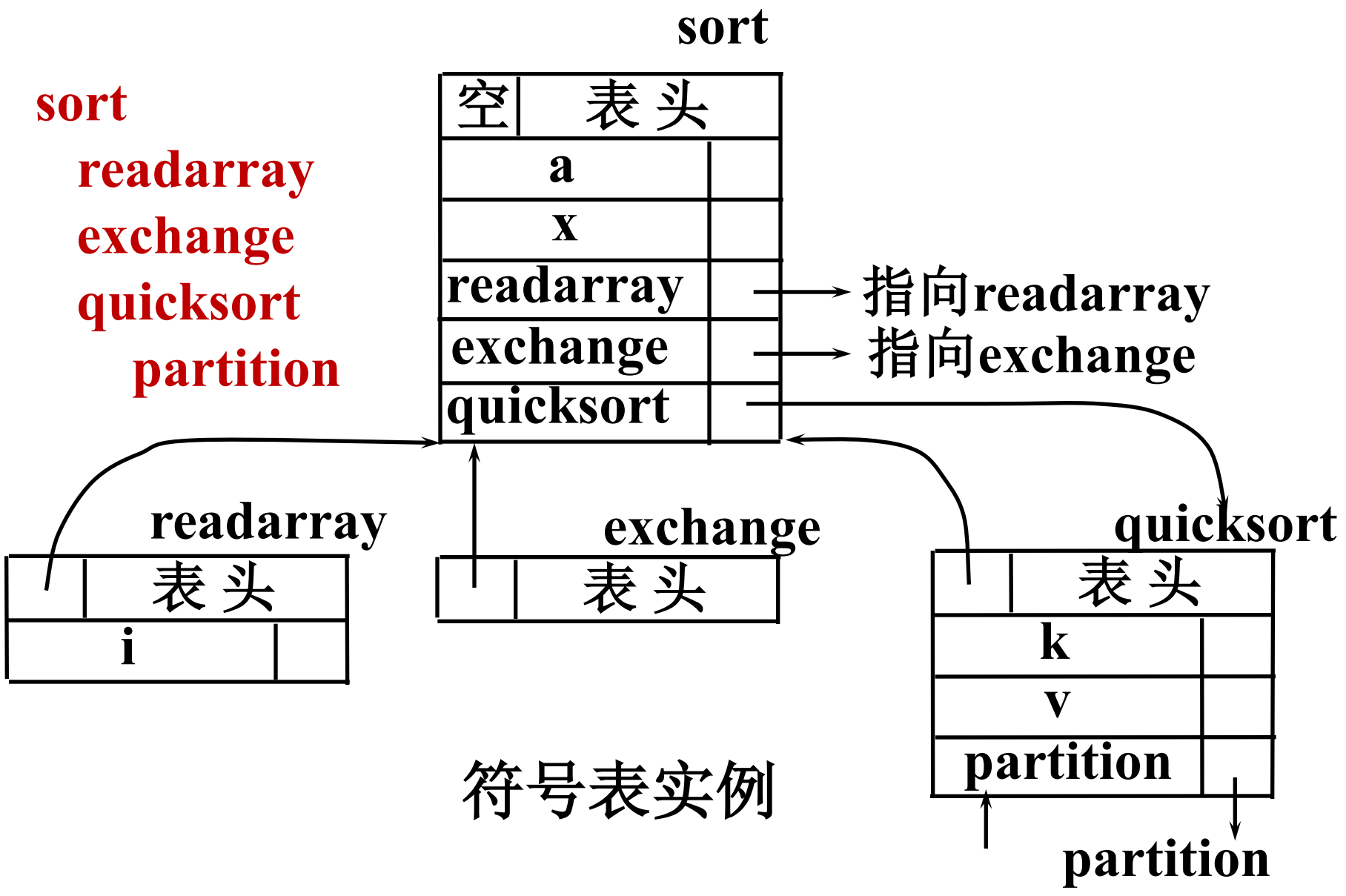
- **管理作用域(过程嵌套声明)**
  - 每个过程内声明的符号要置于该过程的符号表中
  - 方便地找到子过程和父过 程对应的符号

```
sort
    var a:…; x:…;
    readarray
        var i:…;
    exchange
    quicksort
        var k, v:…;
        partition
            var i, j:…;
```

教科书186页图6.14

过程参数被略去

各过程的符号表



符号表实例

符号表的组织与管理

- **符号表的特点及数据结构**
  - 各过程有各自的符号表：哈希表
  - 符号表之间有双向链
    - 父→子：过程中包含哪些子过程定义
    - 子→父：分析完子过程后继续分析父过程
  - 维护符号表栈(*tblptr*)和地址偏移量栈(*offset*)
    - 保存尚未完成的过程的符号表指针和相对地址

符号表的组织与管理

- **语义动作用到的函数**

/* 建立新的符号表，其表头指针指向父过程符号表*/

1. mkTable(parent-table)

/* 将所声明变量的类型、偏移填入当前符号表*/

2. enter(current-table, name, type, current-offset)

/* 在父过程符号表中建立子过程名的条目*/

3. enterProc(parent-table, sub-proc-name, sub-table)

/*在符号表首部添加变量累加宽度，可利用符号表栈tblptr和偏移栈offset（栈顶值分别表示当前分析的过程的符号表及可用变量偏移位置）*/

4. addWidth(table, width)

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow \text{proc id}\ ;\ N\ D_1;\ S$

$D \rightarrow \text{id}\ :T$

$N \rightarrow \varepsilon$

声明语句的处理

tblptr：符号表栈
offset：偏移量栈

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t = mkTable (nil);*
push(t, tblptr); push (0, offset) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id : T

$N \rightarrow \varepsilon$

建立主程序（最外围）的符号表偏移从0开始

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id : T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$

将变量name的有关属性填入当前符号表

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id : T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*));
push(*t*, *tblptr*); *push*(0, *offset*) }

建立子过程的符号表和偏移从0开始

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$ {*t* = *top*(*tblptr*);
addWidth(*t*, *top*(*offset*)); *pop*(*tblptr*); *pop*(*offset*);
enterProc(*top*(*tblptr*), id.*lexeme*, *t*) }

$D \rightarrow$ id : T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*));
push(*t*, *tblptr*); *push*(0, *offset*) }

保留当前过程声明的总空间；弹出符号表和偏移栈顶（露出父过程的符号表和偏移；在父过程符号表中填写子过程名有关条目

声明语句的处理

$P \rightarrow M\ D;\ S$ {*addWidth* (*top* (*tblptr*), *top* (*offset*));
pop(*tblptr*); *pop* (*offset*) }

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$ {*t* = *top*(*tblptr*);
addWidth(*t*, *top*(*offset*)); *pop*(*tblptr*); *pop*(*offset*);
enterProc(*top*(*tblptr*), id.*lexeme*, *t*) }

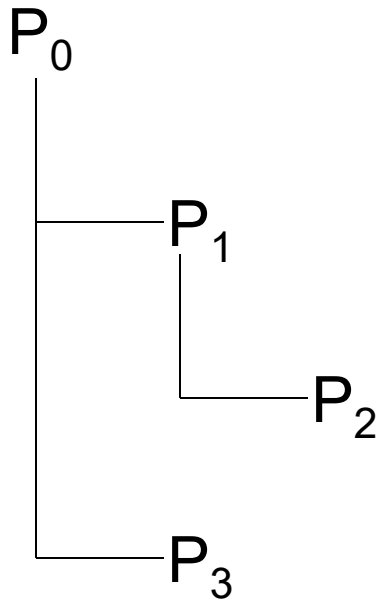
$D \rightarrow$ id :T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*));
push(*t*, *tblptr*); *push*(0, *offset*) }

修改变量分配空间大小并清空符号表和偏移栈

举例：过程嵌套声明

```
i : int;  j : int ;
PROC P1 ;
   k : int;  f : real ;
   PROC P2;
         l : int ;
        a1 ;
   a2;
PROC P3;
   temp : int ;  max : int ;
   a3;
```



过程声明层次图

举例：过程嵌套声明

- 初始：M→ε

null	总偏移：

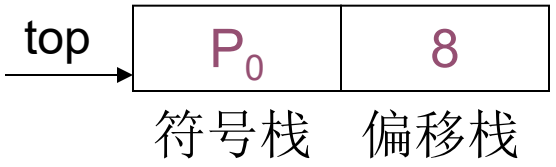
P_0

举例：过程嵌套声明

- **i : int ; j : int ;**

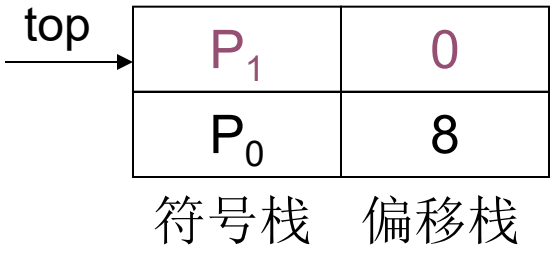
null	总偏移：	
i	INT	0
j	INT	4

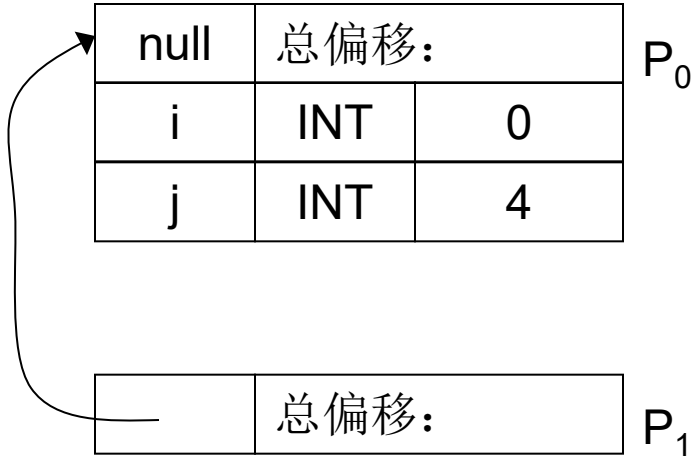
P_0



举例：过程嵌套声明

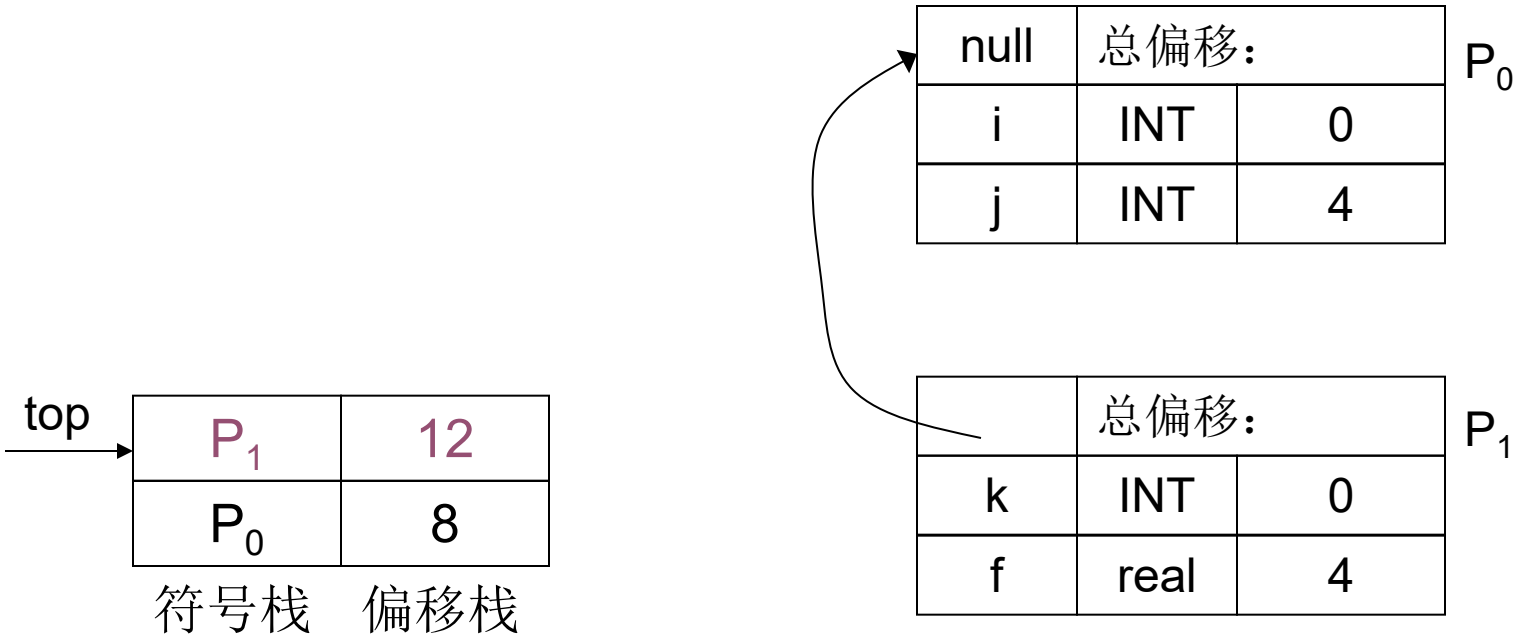
- PROC P_1；（$N \rightarrow \varepsilon$）





举例：过程嵌套声明

- **k : int ; f : real ;**



举例：过程嵌套声明

- PROC P_2；（N→ε）

top

P_2	0
P_1	12
P_0	8

符号栈　偏移栈

null	总偏移：	
i	INT	0
j	INT	4

P_0

	总偏移：	
k	INT	0
f	real	4

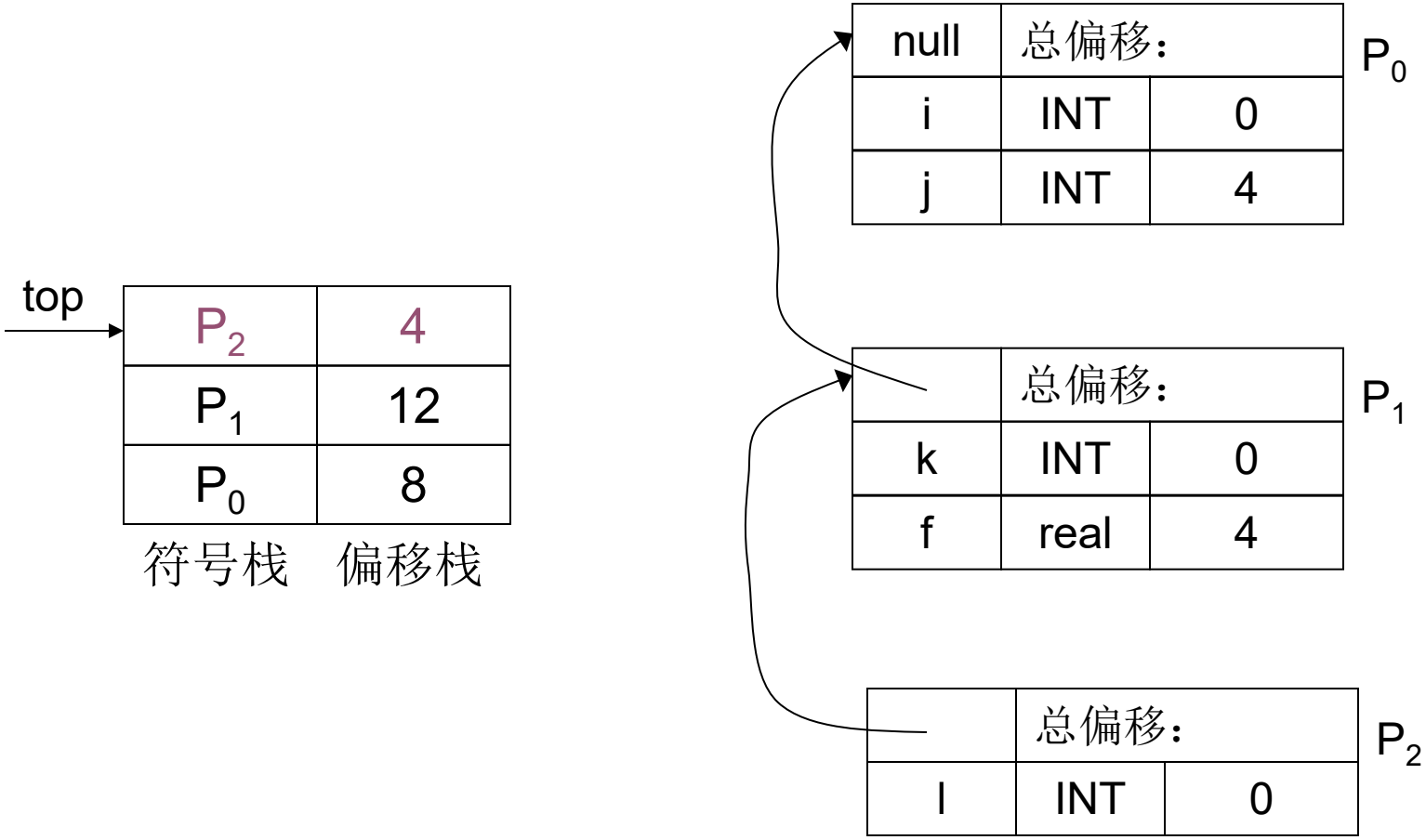
P_1

	总偏移：

P_2

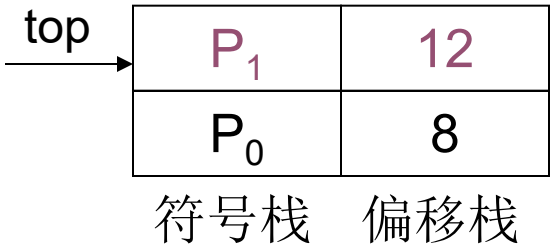
举例：过程嵌套声明

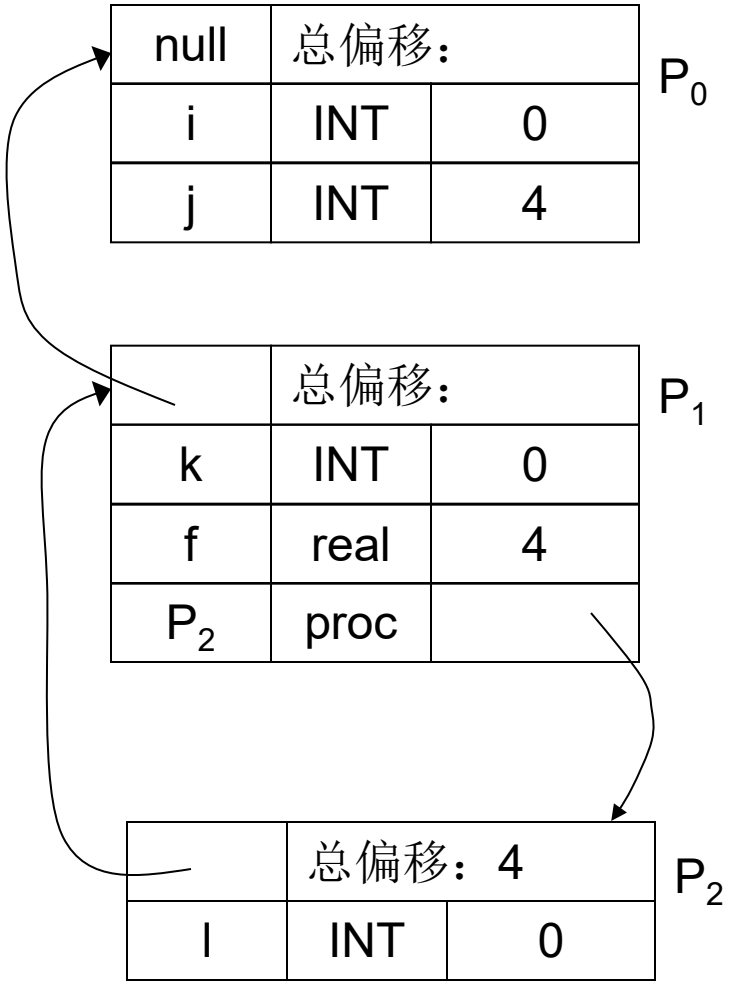
- l : int ;



举例：过程嵌套声明

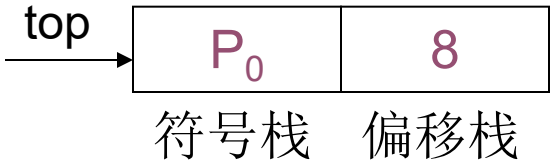
- a_1；

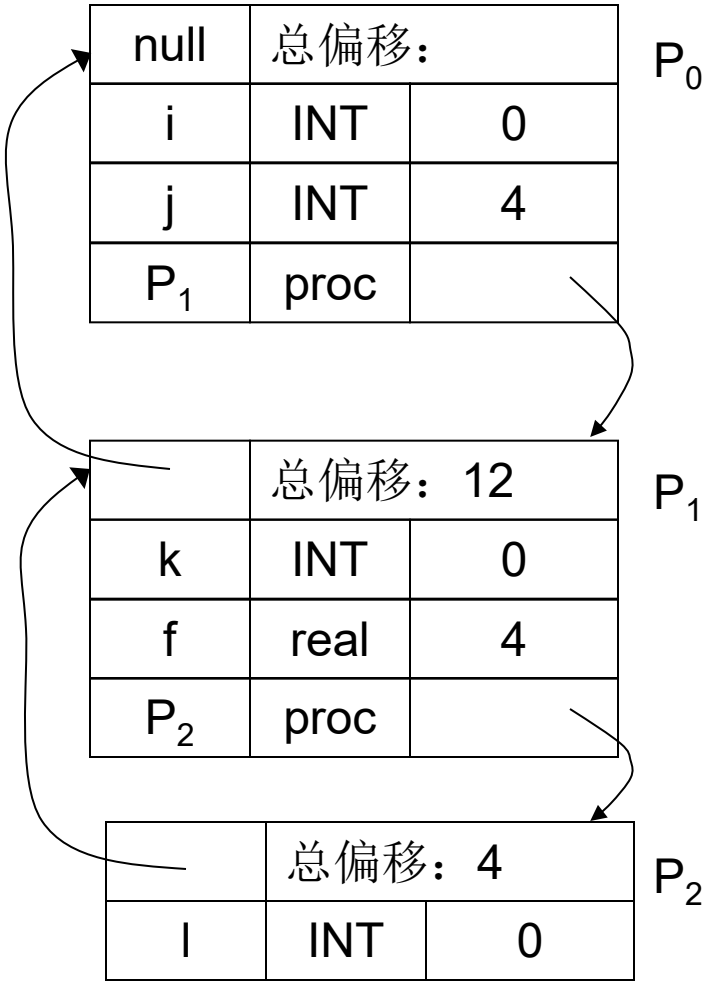

top
P1
12
P0
8
符号栈
偏移栈


null
总偏移：
i
INT
0
j
INT
4
P0
总偏移：
k
INT
0
f
real
4
P2
proc
P1
总偏移：4
l
INT
0
P2

举例：过程嵌套声明

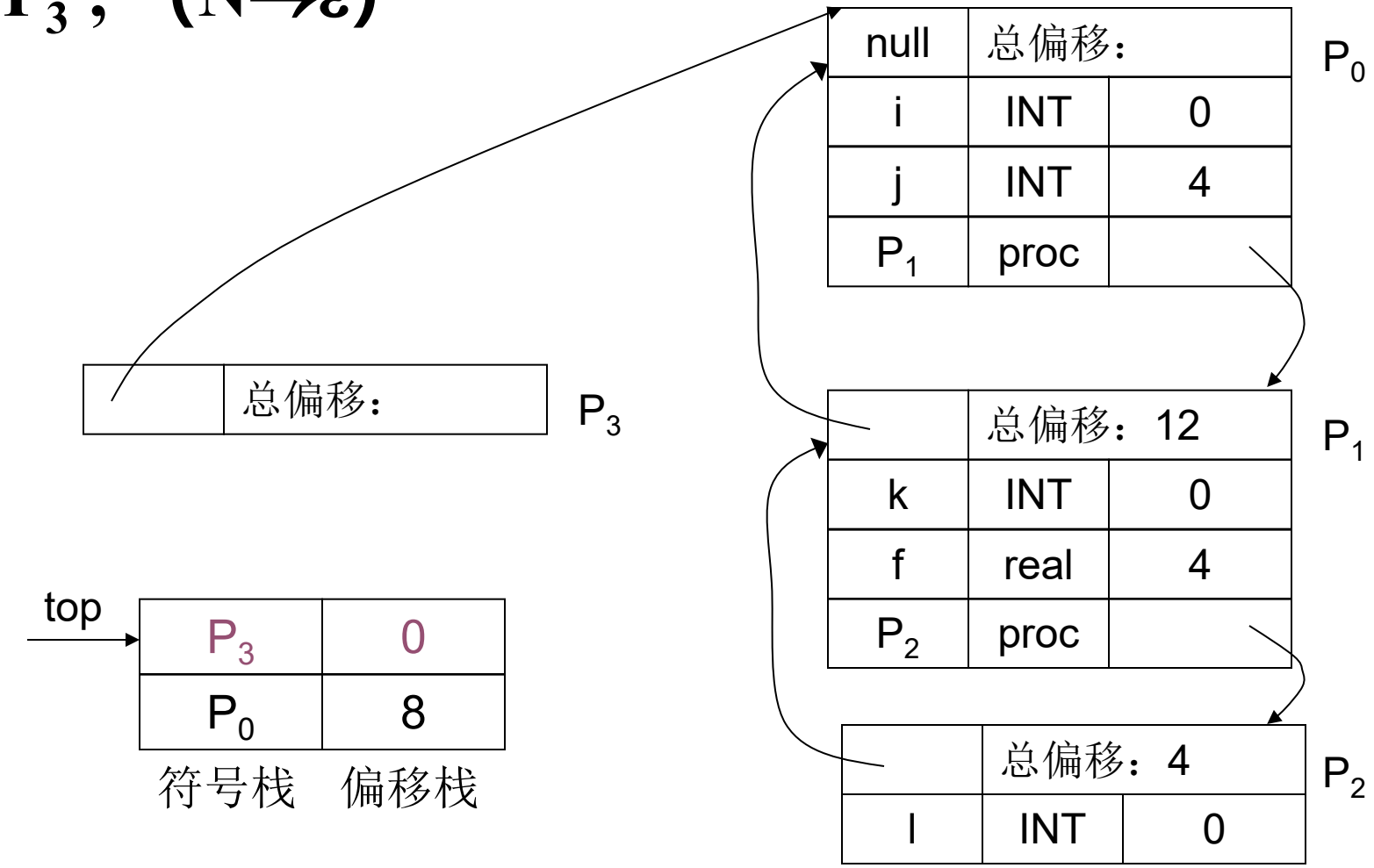
- a_2 ;


top
P0
8
符号栈
偏移栈


null
总偏移：
i
INT
0
j
INT
4
P1
proc
P0
总偏移：12
k
INT
0
f
real
4
P2
proc
P1
总偏移：4
l
INT
0
P2

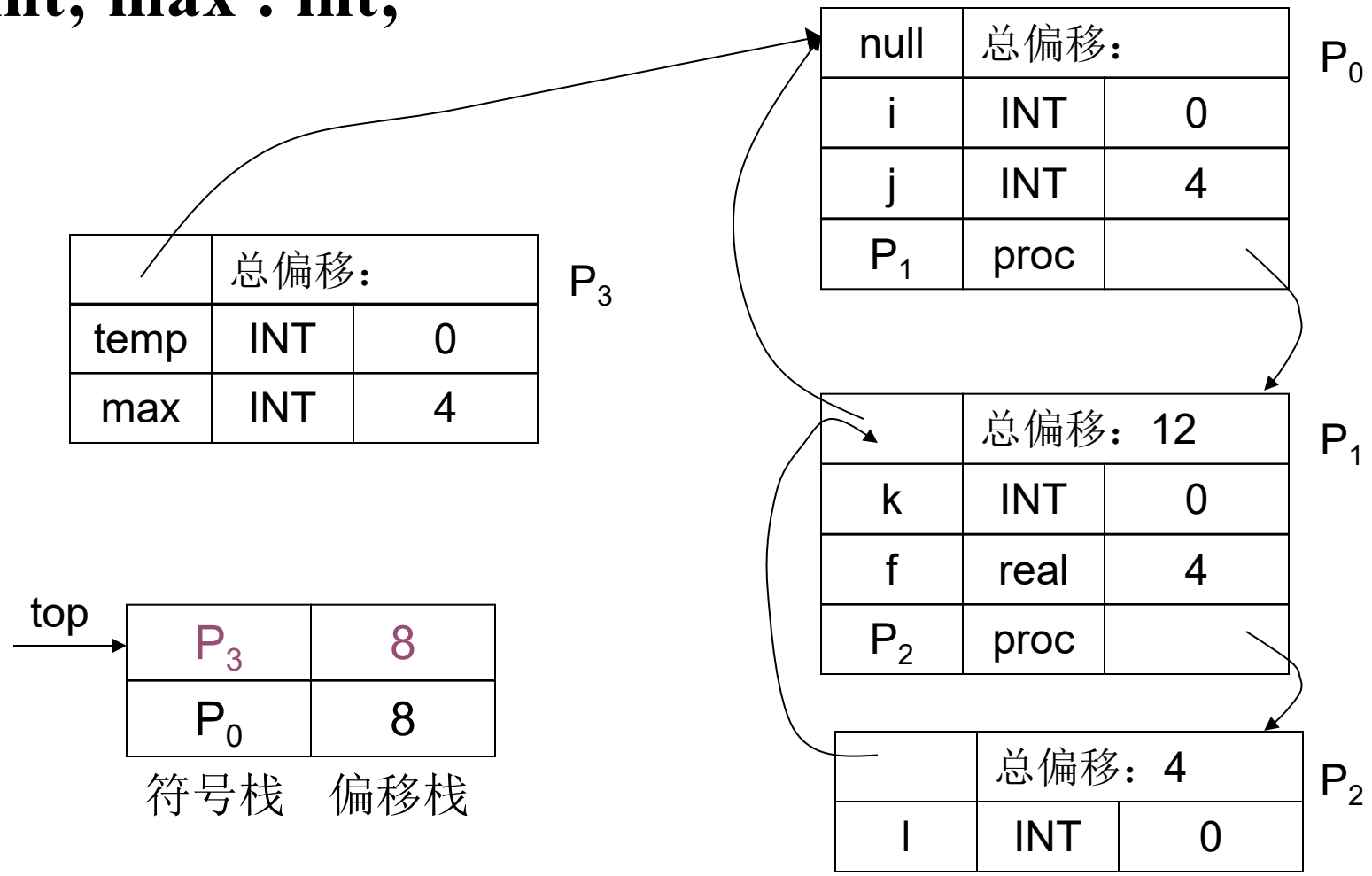
举例：过程嵌套声明

- PROC P_3 ； （N→ε）



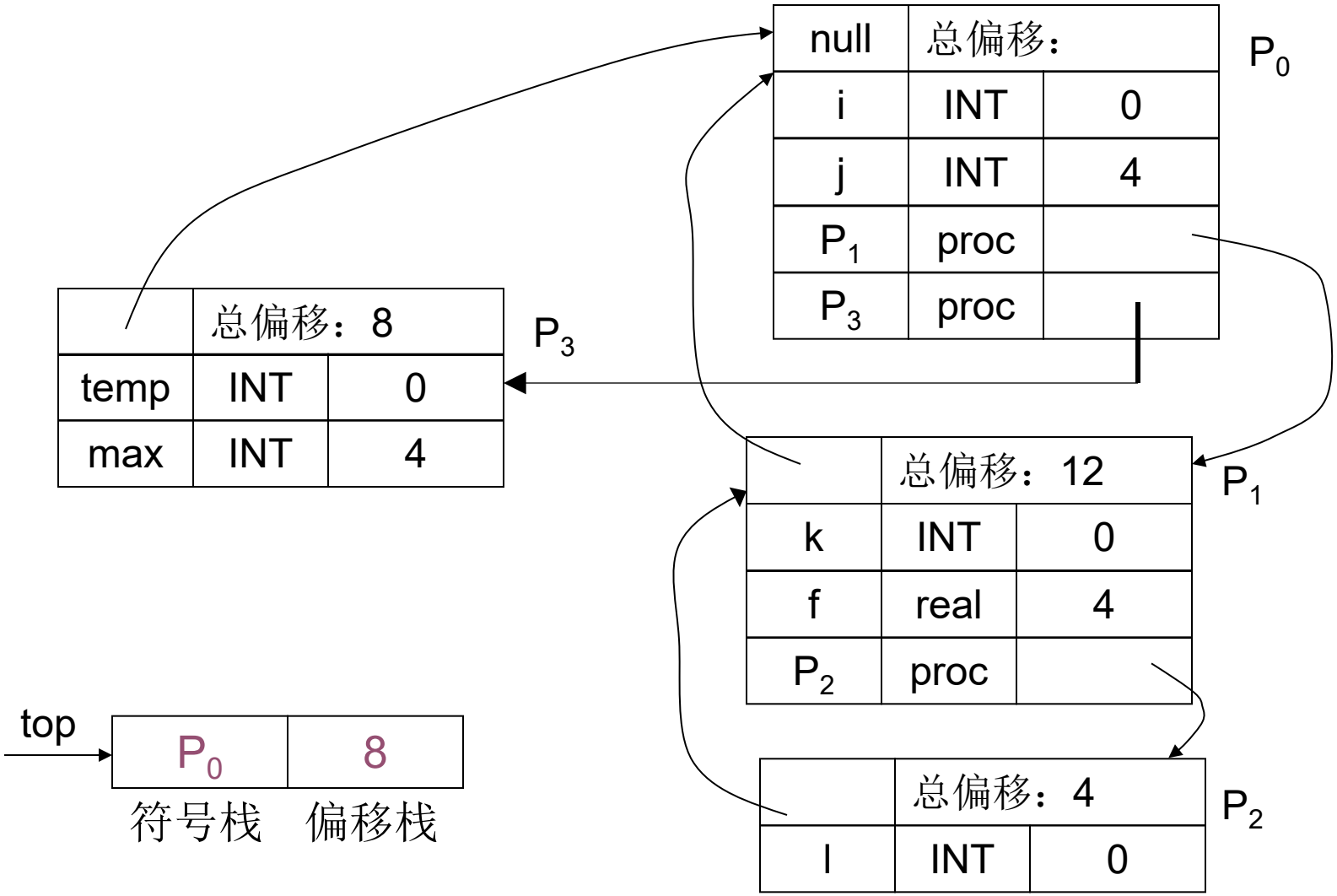
举例：过程嵌套声明

- **temp : int; max : int;**



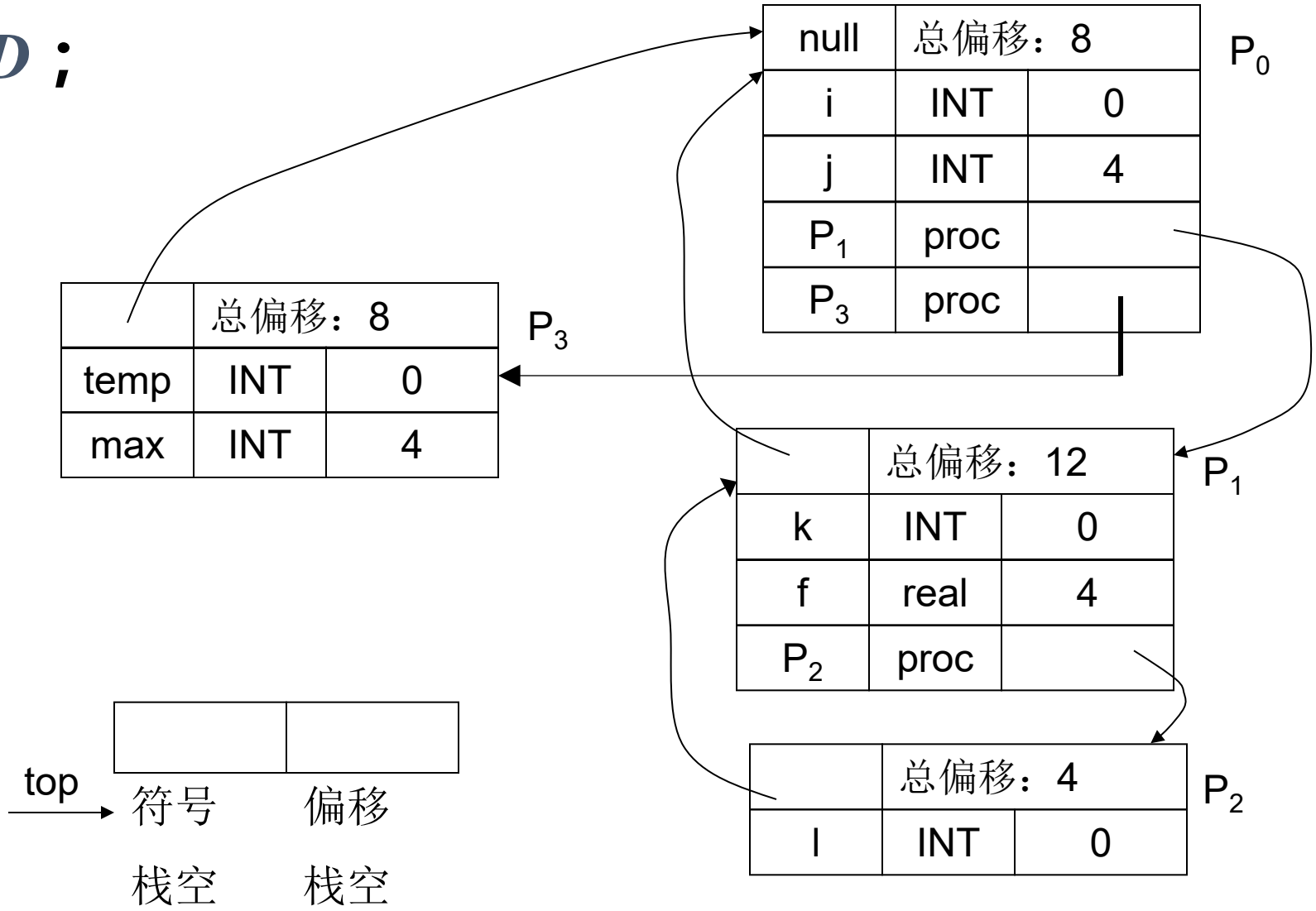
举例：过程嵌套声明

- a_3;


null
总偏移：
i INT 0
j INT 4
P1 proc
P3 proc
P0
总偏移：8
temp INT 0
max INT 4
P3
总偏移：12
k INT 0
f real 4
P2 proc
P1
总偏移：4
l INT 0
P2
top
P0 8
符号栈 偏移栈

举例：过程嵌套声明

- **P→ M *D* ;**



声明语句翻译的要点

- 分配存储单元
  - 名字、类型、字宽、偏移
- 作用域的管理
  - 过程调用
- 记录类型的管理
- 不产生中间代码指令，但是要更新符号表

记录的域名管理

- **描述记录的文法**

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

$L \rightarrow \varepsilon$

```
record
    a :...;
    r : record
            i :...;
            . . .
        end;
    k : ...;
end
```

记录的域名管理

- 描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

$L \rightarrow \varepsilon$ { *t* = *mkTable*(*nil*);
push(*t*, *tblptr*); *push*(0, *offset*) }

```
record
    a :...;
    r : record
            i :...;
            . . .
        end;
    k : ...;
end
```

建立符号表，进入作用域

记录的域名管理

- **描述记录的文法**

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

{*T.type = record (top(tblptr));*

T.width = top(offset);

pop(tblptr); pop(offset) }

$L \rightarrow \varepsilon$ { *t = mkTable(nil);*

push(t, tblptr); push(0, offset) }

```
record
    a :...;
    r : record
            i :...;
            . . .
        end;
     k : ...;
end
```

设置记录的类型表达式和宽度，退出作用域

记录的域名管理

- 描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

{*T.type* = *record* (*top*(*tblptr*));

T.width = *top*(*offset*);

pop(*tblptr*); *pop*(*offset*) }

$L \rightarrow \varepsilon$ { *t* = *mkTable*(*nil*);

push(*t*, *tblptr*); *push*(0, *offset*) }

D的翻译同前

```
record
    a :...;
    r : record
            i :...;
            . . .
        end;
    k : ...;
end
```

举例：记录域的偏移

- 有2个C语言的结构定义如下：

```
struct A {
    char c1;
    char c2;
    long l;
    double d;
} S1;
```

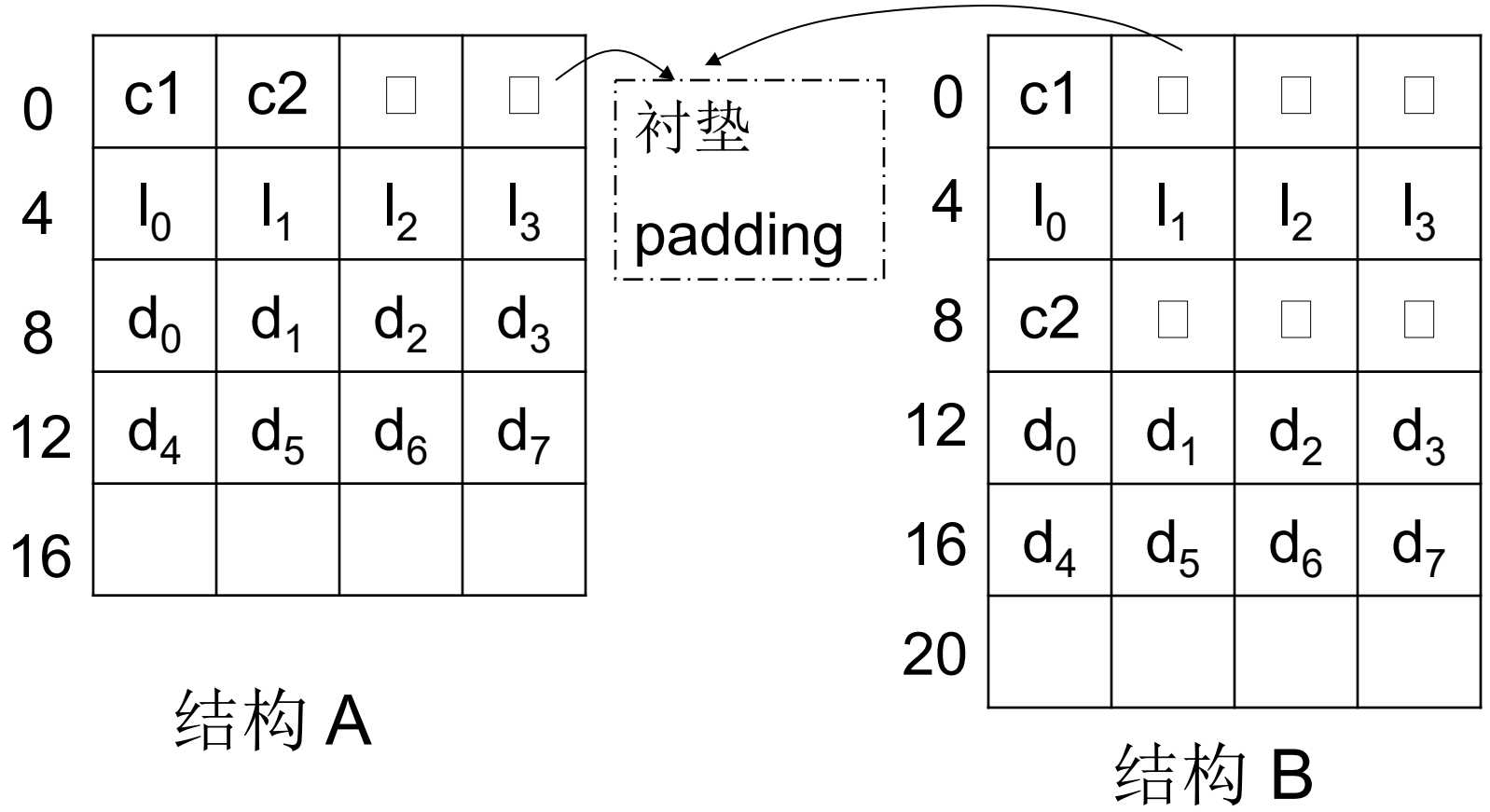
```
struct B {
char c1;
long l;
        char c2;
        double d;
} S2;
```

举例：记录域的偏移

- **数据（类型）的对齐 - alignment**
- **在 X86-Linux下：**
  - char： 对齐1，起始地址可分配在任意地址
  - int，long，double： 对齐4，即从被4整除的地址开始分配
- **注*： 其它类型机器，double可能对齐到8**
  - 如sun－SPARC

举例：记录域的偏移

- 结构A 和 B的大小分别为16和20字节(Linux)



举例：记录域的偏移

- 2个结构中域变量的偏移如下：

```
struct A {
   char c1;   0
   char c2;   1
   long l;    4
   double d;  8
} S1;
```

```
struct B {
    char c1;    0
    long l;     4
    char c2;    8
    double d;  12
} S2;
```

一起努力
打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年10月30日