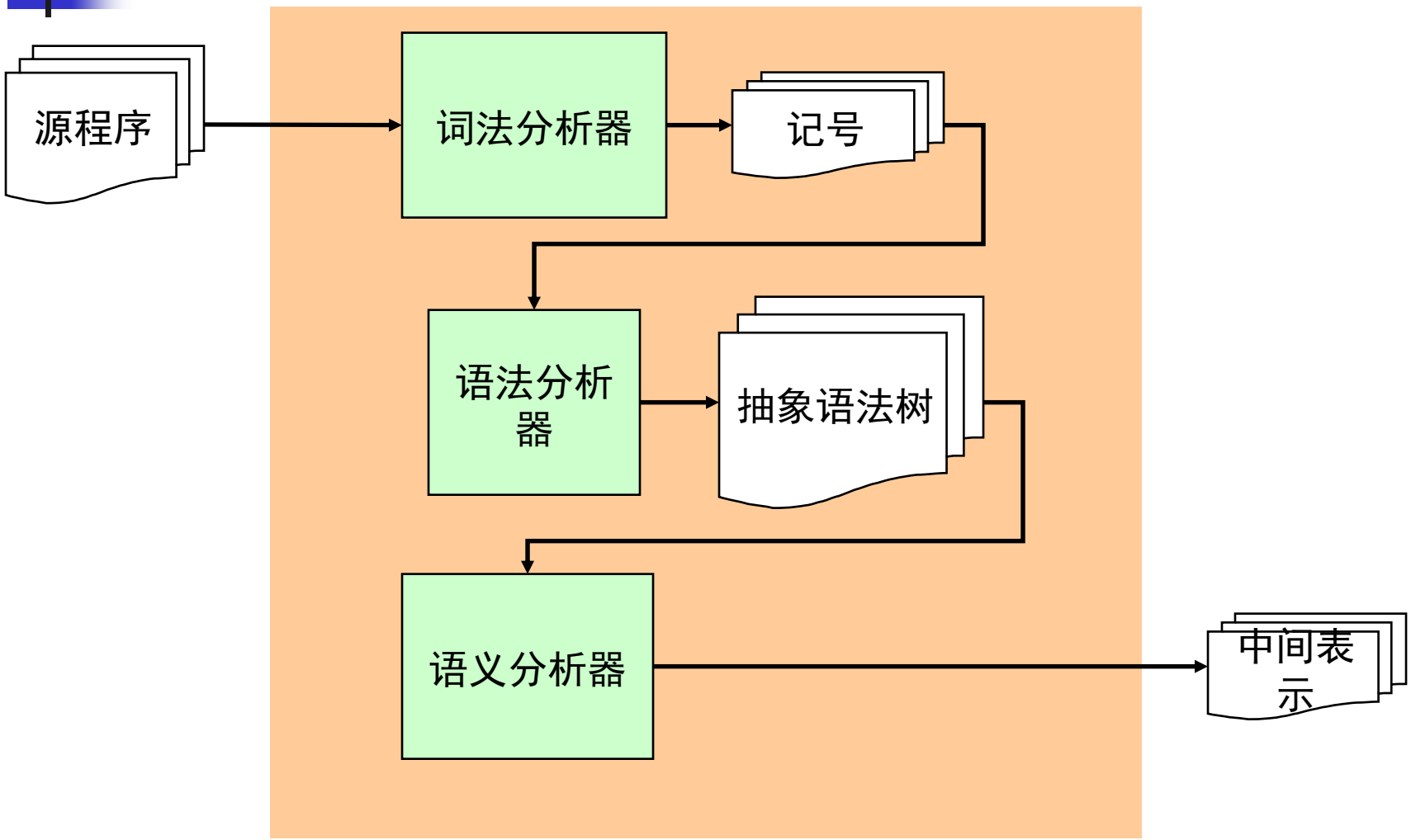
中间表示：控制流图

编译原理

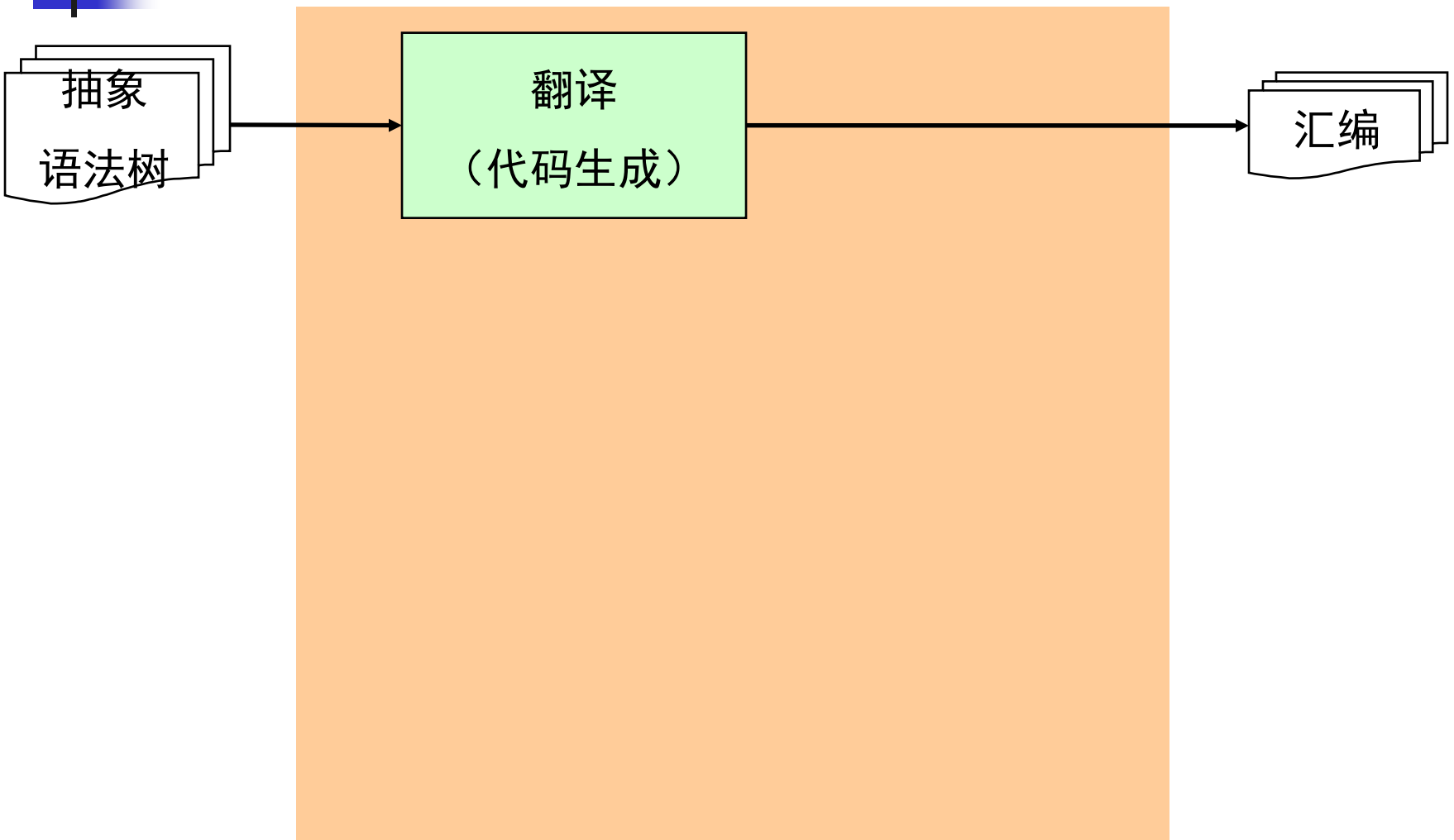
华保健

bjhua@ustc.edu.cn

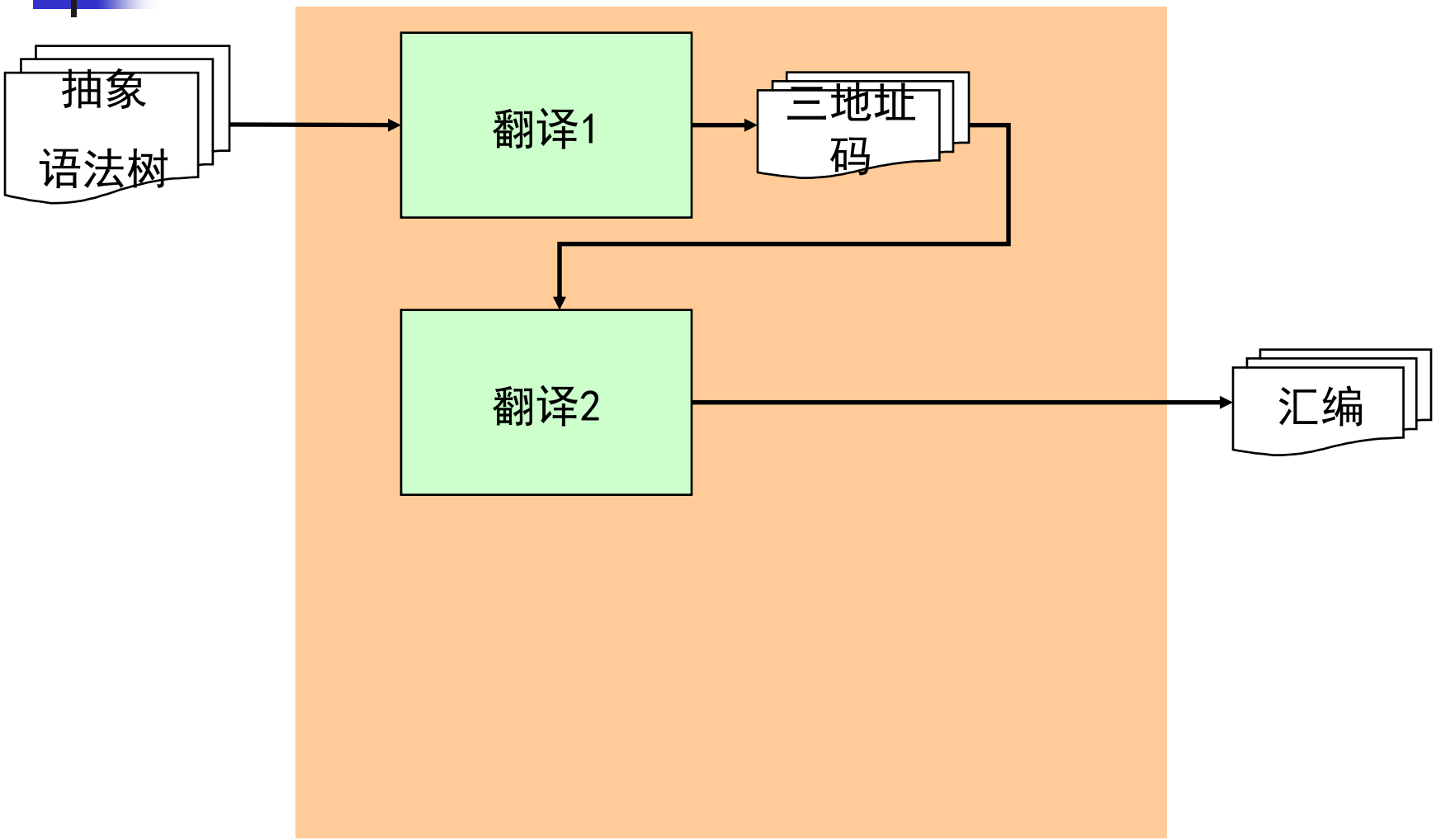
前端



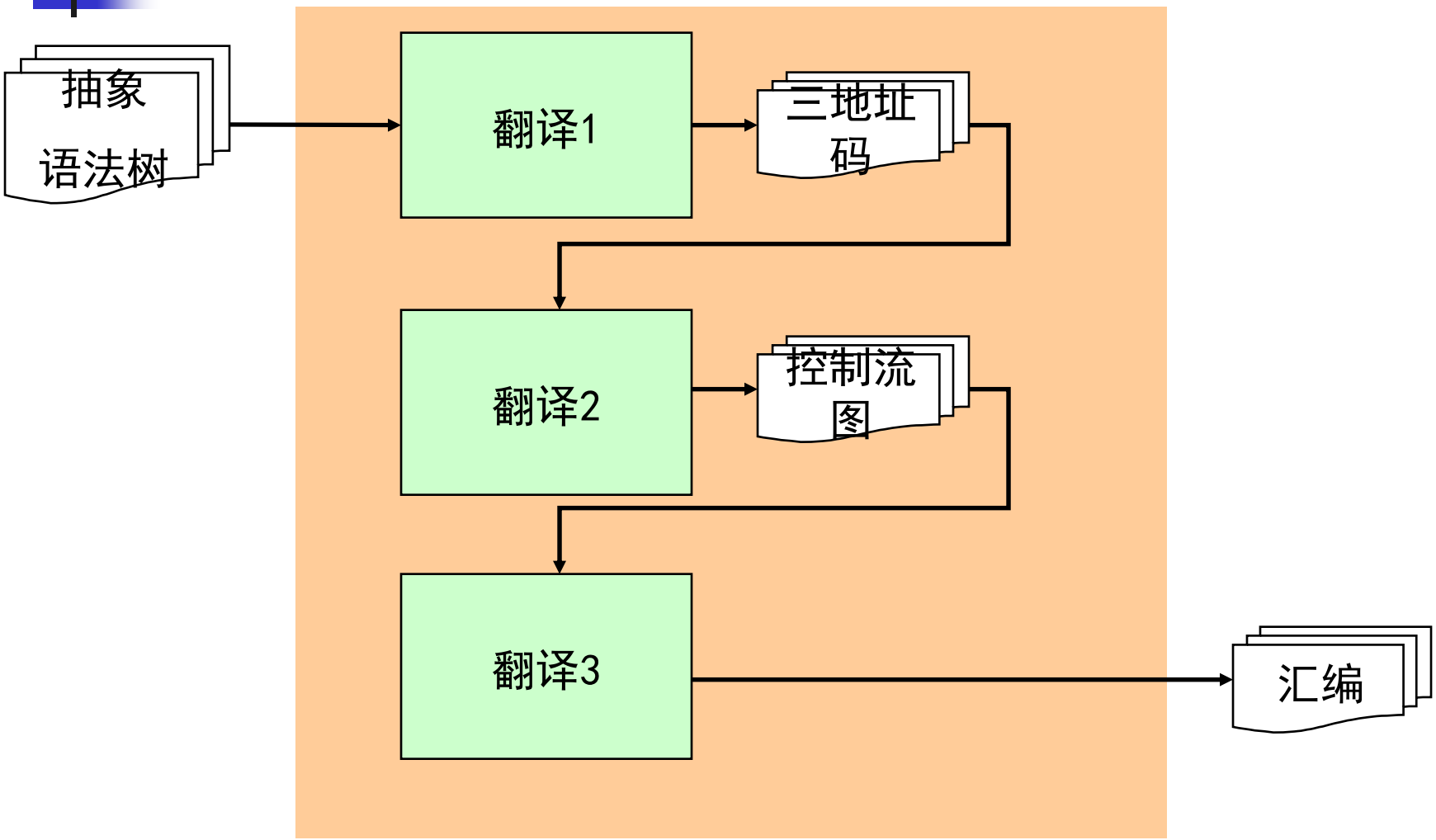
最简单的结构



使用三地址码的编译器结构

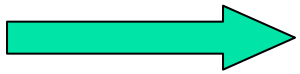


使用控制流图的编译器结构



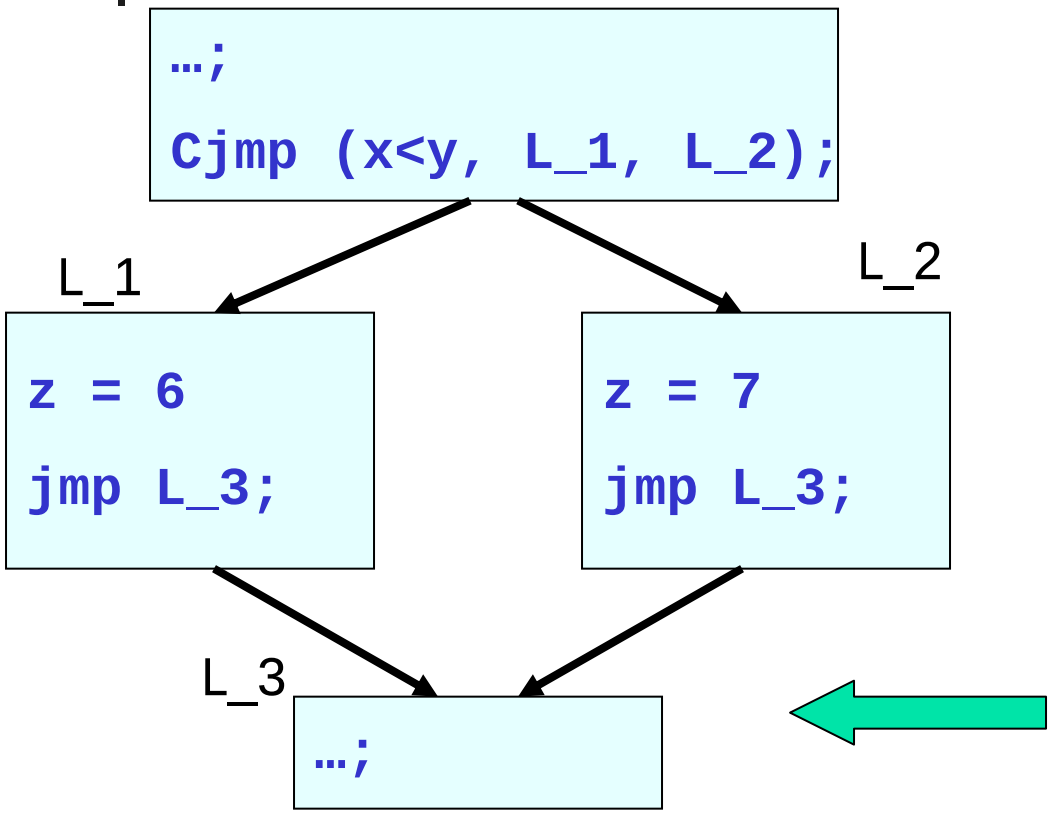
三地址码的不足

```
a = 3 + 4 * 5;




if (x < y)
  z = 6;
else
  z = 7;
```

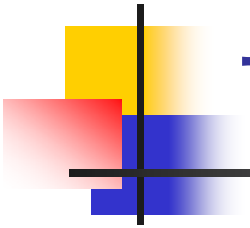


```
x_1 = 4;
x_2 = 5;
x_3 = x_1 * x_2;
x_4 = 3;
x_5 = x_4 + x_3;
a   = x_5;
Cjmp (x<y, L_1, L_2);
L_1:
    z = 6;
    jmp L_3;
L_2:
    z = 7;
    jmp L_3;
L_3:
    …
```

控制结构

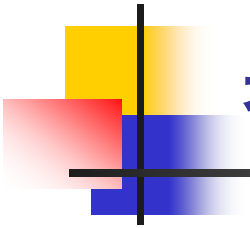



```
x_1 = 4;
x_2 = 5;
x_3 = x_1 * x_2;
x_4 = 3;
x_5 = x_4 + x_3;
a   = x_5;
Cjmp (x<y, L_1, L_2);
L_1:
    z = 6;
    jmp L_3;
L_2:
    z = 7;
    jmp L_3;
L_3:
    …
```



评论

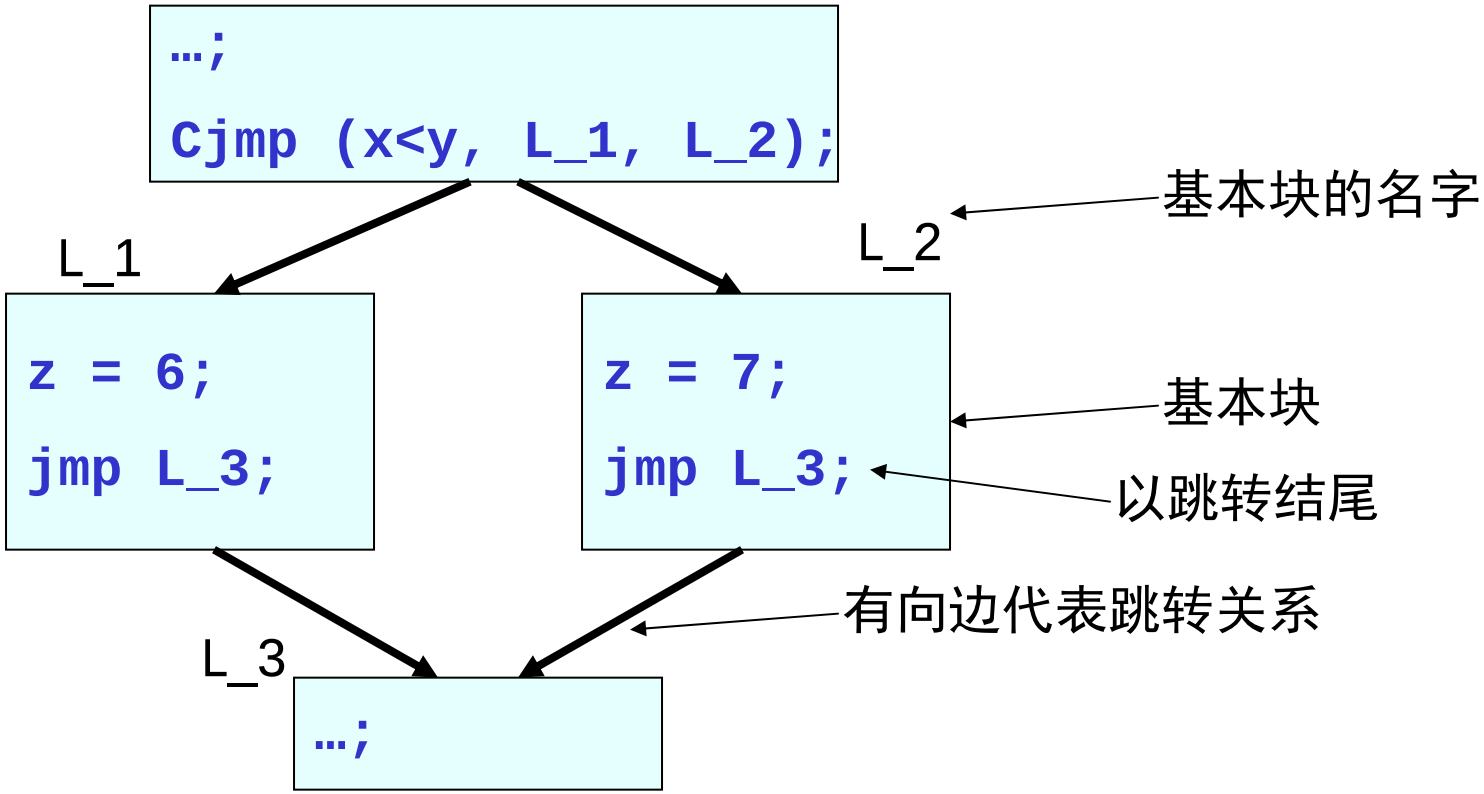
- 程序的控制流图表示带来很多好处:
  - 控制流分析:
    - 对很多程序分析来说，程序的内部结构很重要
      - 典型的问题：“程序中是否存在循环？”
  - 可以进一步进行其他分析:
    - 例如数据流分析
      - 典型的问题：“程序第5行的变量x可能的值是什么？”
- 现代编译器的早期阶段就会倾向做控制流分析
  - 方便后续阶段的分析



基本概念

- 基本块：是语句的一个序列，从第一条执行到最后一条
  - 不能从中间进入
  - 不能从中间退出
    - 即跳转指令只能出现在最后
- 控制流图：控制流图是一个有向图G=(V, E)
  - 节点V：是基本块
  - 边E：是基本块之间的跳转关系

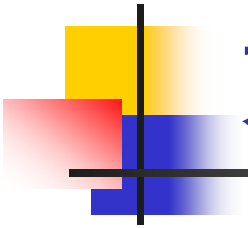
示例


…;
Cjmp (x<y, L_1, L_2);
L_1
L_2
z = 6;
jmp L_3;
z = 7;
jmp L_3;
L_3
…;
基本块的名字
基本块
以跳转结尾
有向边代表跳转关系

控制流图的定义

```
// 是更精细的三地址码
S -> x = n
   | x = y + z
   | x = y
   | x = f (x1, x2, …, xn)
J -> jmp L
   | cjmp (x, L1, L2)
   | return x
B -> Label L;
     S1; S2; …; Sn
     J
F -> x() { B1, …, Bn }
P -> F1, …, Fn
```

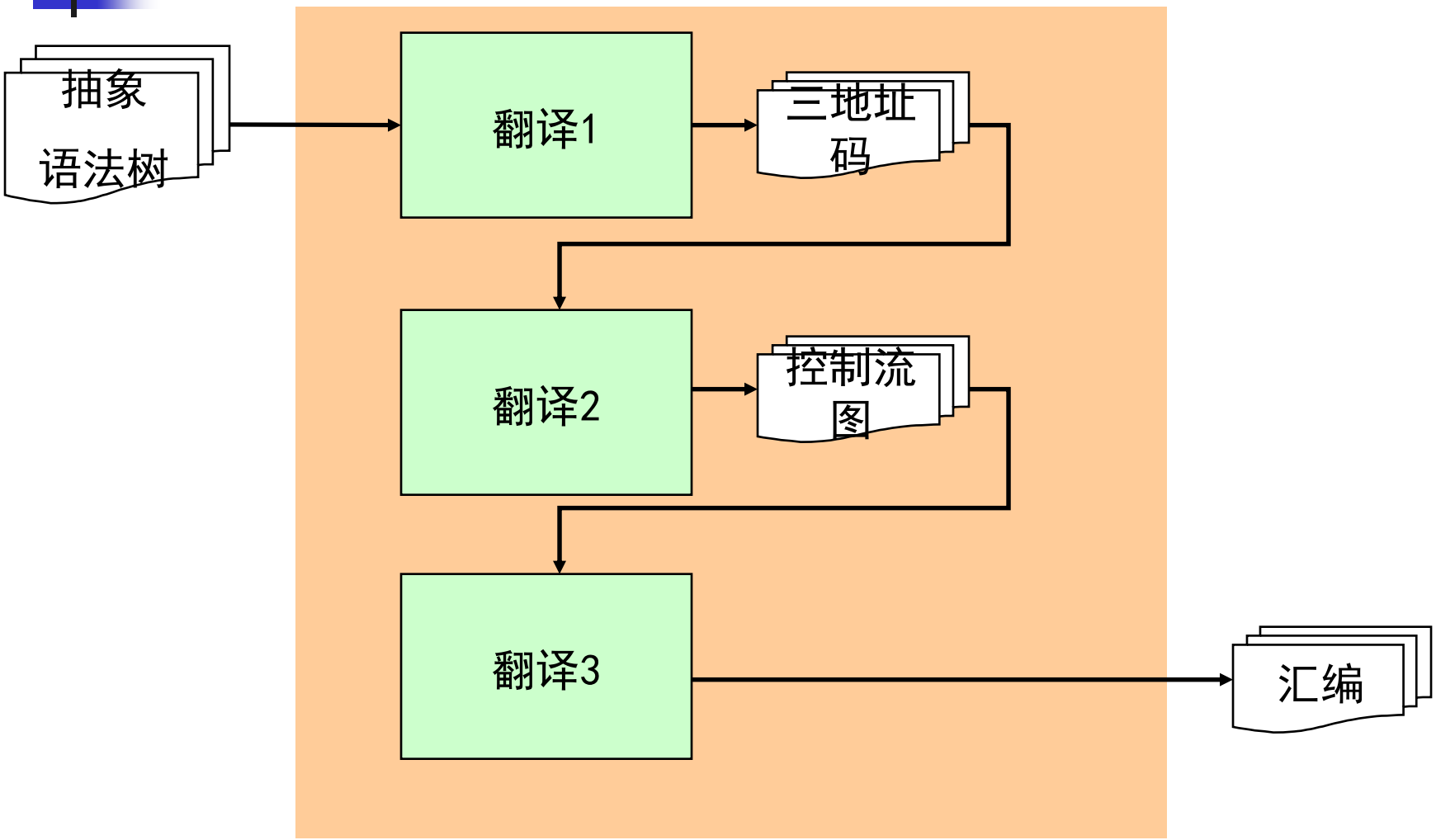
```
// 数据结构定义（以B为例）
struct Block{
  Label_t label;
  List_t stms;
  Jump_t j;
};
```



如何生成控制流图？

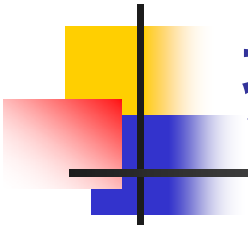
- 可以直接从抽象语法树生成:
  - 如果高层语言具有特别规整控制流结构的话较容易
- 也可以先生成三地址码，然后继续生成控制流图:
  - 对于像C这样的语言更合适
    - 包含像goto这样的非结构化的控制流语句
  - 更加通用（阶段划分！）
- 接下来，我们重点讨论第二种

使用控制流图的编译器结构



由三地址码生成控制流图算法

```
List_t stms;                        // 三地址码中所有语句
List_t blocks = {};                 // 控制流图中的所有基本块
Block_t b = Block_fresh();          // 一个初始的空的基本块
scan_stms ()
  foreach(s ∈ stms)
    if (s is “Label L”)             // s是标号
      b.label = L;
    else (s is some jump)           // s是跳转
      b.j = s;
      blocks ∪= {b};
      b = Block_fresh ();
    else                            // s是普通指令
      b.stms ∪= {s};
```

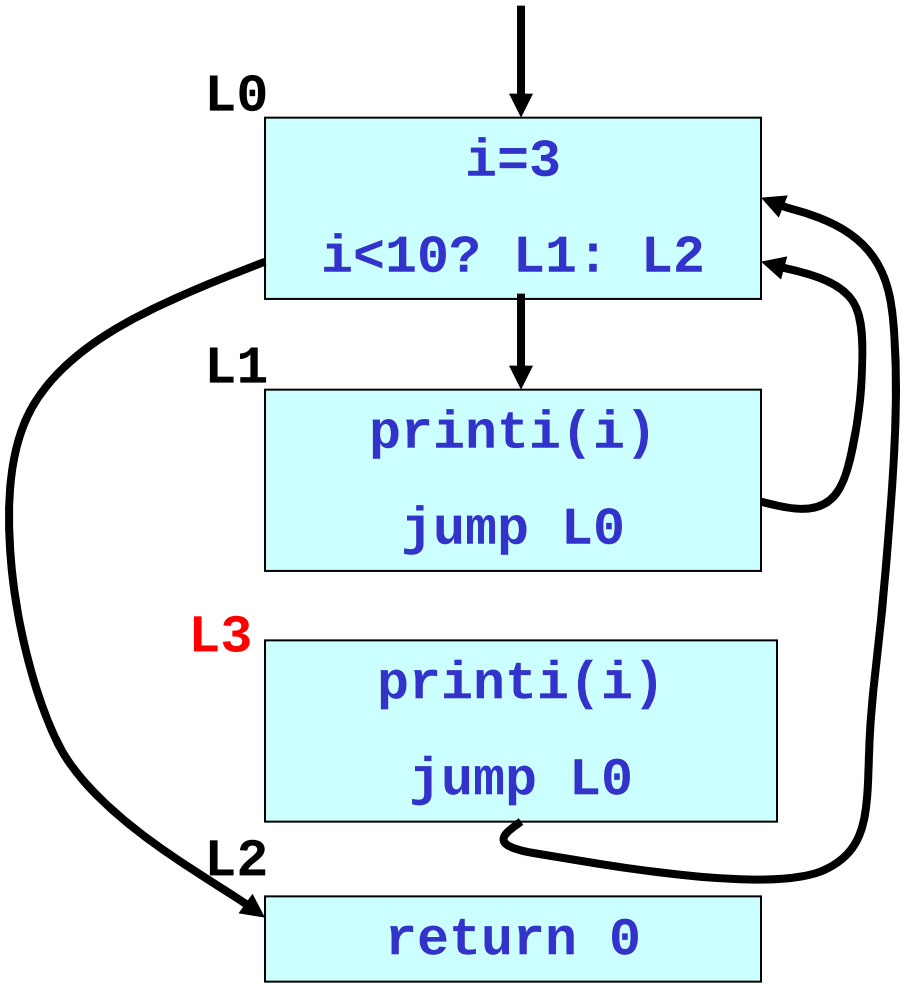


控制流图的基本操作

- 标准的图论算法都可以用在控制流图的操作上:
  - 各种遍历算法、生成树、必经节点结构、等等
- 图节点的顺序有重要的应用:
  - 拓扑序、逆拓扑序、近似拓扑序、等等
- 这里我们不打算重复算法课的内容，而是通过研究一个具体的例子来展示基本图算法的应用:
  - 死基本块删除优化

死基本块删除优化的示例

```
int f ()
{
  int i = 3;
  while (i<10){
    i = i+1;
    printi(i);
    continue;
    printi(i);
  }
  return 0;
}
```



死基本块删除的算法

```
// 输入：控制流图g
// 输出：经过死基本块删除
//       后的控制流图
dead_blocks_elim (g)
  dfs (g);
  for (each node n in g)
    if (!visited(n))
      delete (n);
```