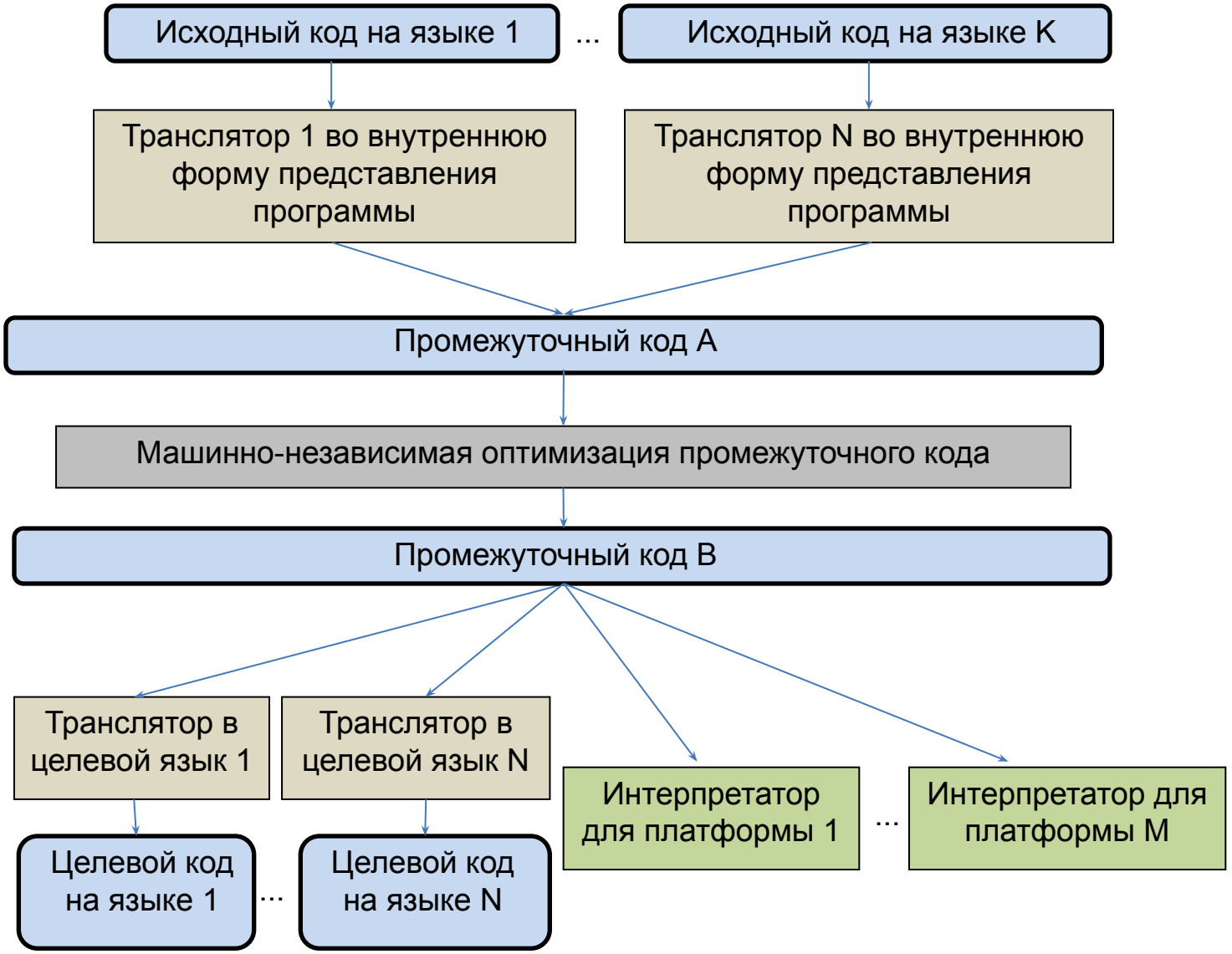
Внутренние формы представления программ

Разделение транслятора при использовании промежуточного языка

Построение трансляторов для различных языков и платформ



Требования к внутренним формам представления программ

- Простота трансляции с исходных языков во внутреннюю форму представления программы
- Простота трансляции внутренней формы представления программы в целевые языки или машинный код целевых платформ
- Удобство проведения машинно-независимой оптимизации по выбранной внутренней форме представления программы
- Эффективная интерпретация внутренней формы представления программы

Классификация внутренних форм представления программ

- **Высокоуровневые формы представления программ**
  - используются на ранних этапах трансляции
  - отражают конструкции языков высокого уровня
  - позволяют выполнять некоторую оптимизацию с учетом свойств исходного языка
  - часто преобразуются в формы представления более низкого уровня

  Примеры: синтаксические деревья, абстрактные синтаксические деревья – AST, направленные ациклические графы - DAG

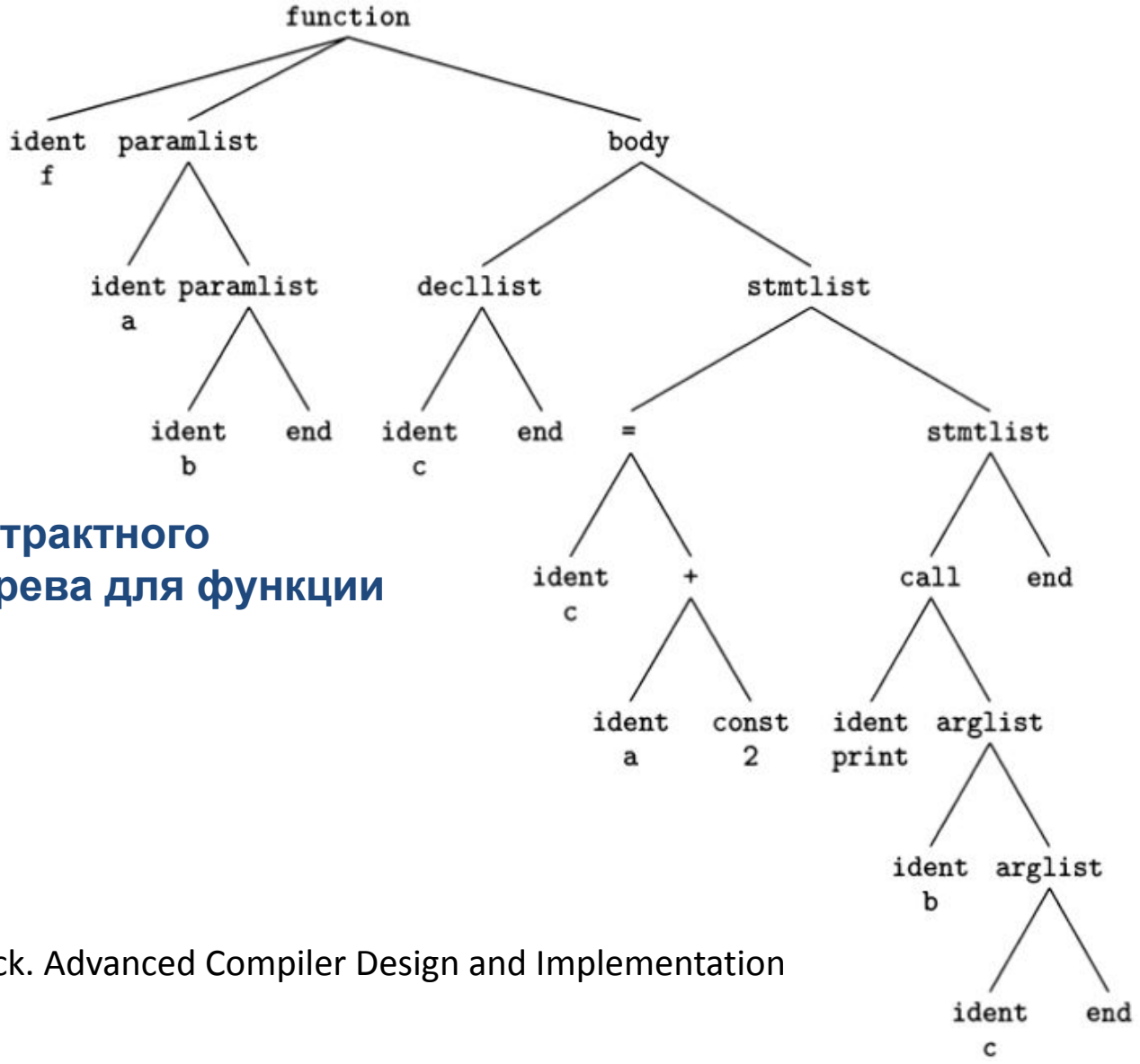
- **Формы представления программ среднего уровня**
  - достаточно независимы от входного и целевого языков
  - позволяют генерировать достаточно эффективный целевой код

  Примеры: тетрады и триады

- **Низкоуровневые формы представления программ**
  - машинно-зависимая форма представления
  - целевой код генерируется очевидным образом

  Пример: язык RTL компилятора gcc

Абстрактные синтаксические деревья



Пример абстрактного синтаксического дерева для функции

```
int f(int a, int b)
{
        int c;
        c = a + 2;
        print(b, c);
}
```

Пример из Steven Muchnick. Advanced Compiler Design and Implementation / Morgan Kaufmann. 1997.

Сравнение уровней внутренних форм представления программ

Цикл со счетчиком	Обращение к элементу массива
Высокоуровневая форма представления	

`for x a to b`
`  <операторы>`
`end`
	`y:=x[i,j]`

Средний уровень	

`  x := a`
`  t1 := b`
`L1: if (x>t1) goto L2`
`  <операторы>`
`  x:=x+1`
`  goto L1`
`L2:`
	`t1:=i*10`
	`t2:=t1+j`
	`t3:=addr(x)`
	`t4:=t3+t2`
	`y:=*t4`

Низкий уровень

```
r1:=[fp-1]
r2:=[fp-2]
r3:=r1*10
r4:=r3+r2
r5:=fp-102
r6:=r5+r4
r7:=[r6]
```

Выбор внутренней формы представления с учетом модели вычислительной машины

Регистровые вычислительные машины или машины с произвольным доступом к памяти:

- Операции производятся с регистрами ЭВМ или непосредственно с памятью с произвольным доступом

Внутренние формы представления: Тетрады и триады

Стековые вычислительные машины:

- Операции производятся с содержимым стека - операнды извлекаются из вершины стека, и на их место помещается результат

Внутренние формы представления: Польская инверсная запись (постфиксная нотация)

Трехадресный код. Тетрады и триады

Тетрада

(<оператор>,<операнд1>,<операнд2>,<результат>)

Исходный код

```
s = 0;
for (i = 0; i < 10; ++i) s += ar[i];
```

Последовательность тетрад

```
      (:=, 0, , s)// s = 0
      (:=, 0, , i) // i = 0
L1:   (<, i, 10, t0)   // t0 = i<10
      (JZ, t0, L2,)    // if (t0 == 0) goto L2
      (+, ar, i, t1)   // t1 = ar+i
      (**, t1, , t2)   // t2=*t1, разыменование
      (+, s, t2, s)    // s=s+t2
      (+, i, 1, i) // i=i+1
      (JMP, L1, ,)     // goto L1
L2:
```

Основной недостаток тетрад - большое количество временных переменных

Триада

(<оператор>,<операнд1>,<операнд2>)

Исходный код

```
a*b+c*d
```

Последовательность триад

```
1:    (*, a, b)
2:    (*, c, d)
3:    (+,(1),(2))
```

Достоинства тетрад и триад - простота оптимизации внутренней формы представления программы

Польская инверсная запись

Формы записи арифметических выражений

Пример записи арифметических выражений в ПОЛИЗе

Исходное выражение:
(2+3)*7+4/2

Польская инверсная запись:
2 3 + 7 * 4 2 / +

Вычисление выражения вручную

2 3 + 7 * 4 2 / +
5 7 * 4 2 / +
35 **4 2 /** +
35 2 +
37

Примеры записи управляющих конструкций в ПОЛИЗе

Условный оператор

```
if (<условие>) <оператор1> else <оператор2>;
```

ПОЛИЗ

```
<условие> <адр7> JZ   <оператор1> <адр2> JMP  <оператор2> …
адр1      адр2   адр3 адр4        адр5   адр6 адр7
```

Оператор цикла с предусловием

```
while (<условие>) <оператор1>;
```

ПОЛИЗ

```
<условие> <адр7> JZ   <оператор1> <адр1> JMP  …
адр1      адр2   адр3 адр4        адр5   адр6 адр7
```

Оператор цикла с постусловием

```
do <оператор1> while (<условие>);
```

ПОЛИЗ

```
<оператор1> <условие> NOT  <адр1> JZ   …
адр1        адр2      адр3 адр4   адр5 адр6
```

Операция обращения по индексу

Для одномерных массивов: **<имя_массива> <индекс> [**

Для многомерных массивов : **<имя_массива><индекс1>…<индексN> N [**

Пример записи фрагмента программы в ПОЛИЗе

Фрагмент исходной программы

```
for (i=0; i<10; i++) {
    if (i<5)&&(j>0)  a[i] = j;
    else  a[i]=0;
}
```

ПОЛИЗ

```
i 0 := i 10 < 36 JZ i 5 <  j  0  >  AND 24 JZ a  i  [  j  :=
0 1 2  3 4  5 6  7  8 9 10 11 12 13 14  15 16 17 18 19 20 21

29 JMP a  i  [  0  := i  i  1  +  := 3  JMP
22 23  24 25 26 27 28 29 30 31 32 33 34 35  36
```