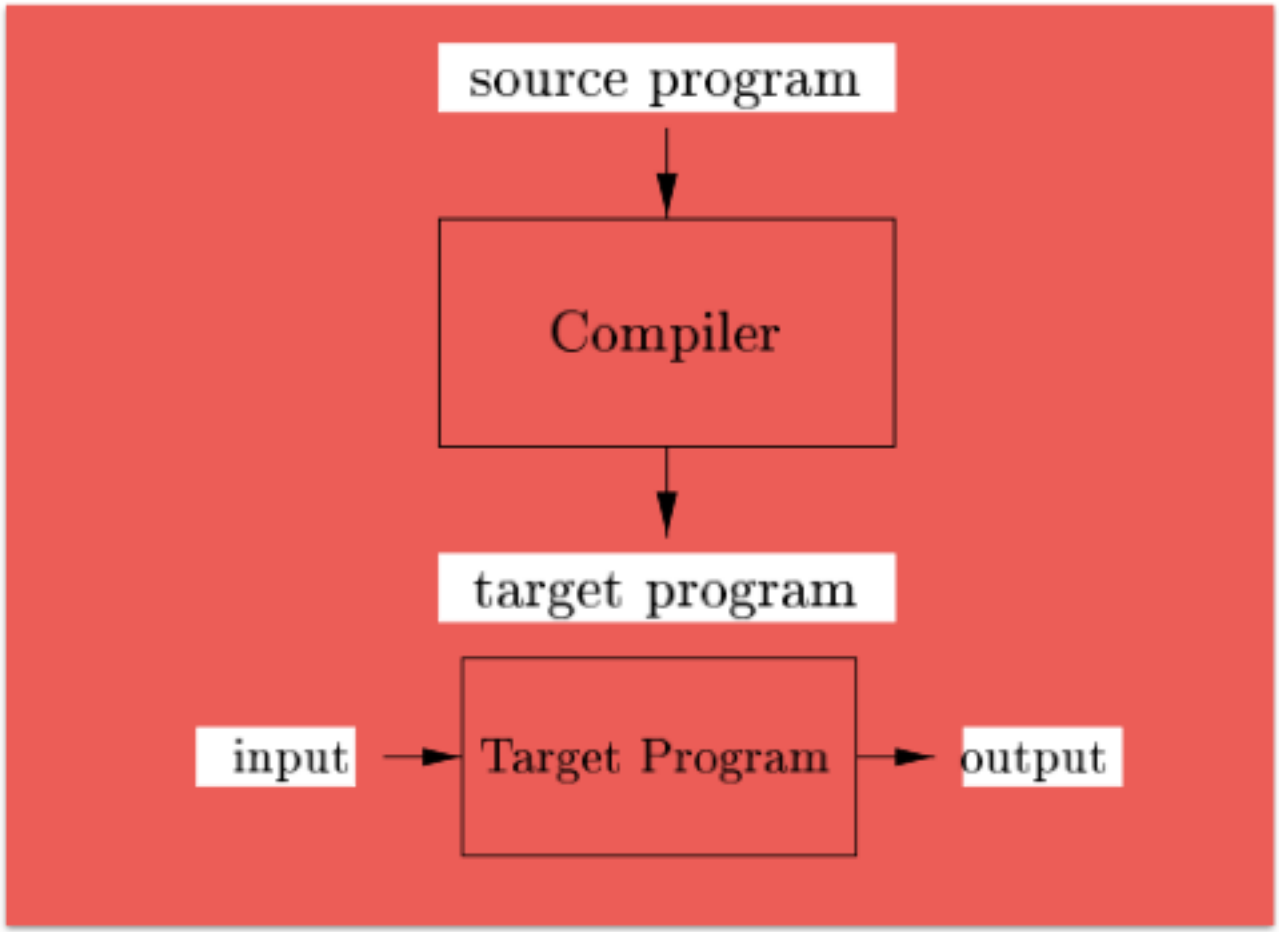
بسم الله الرحمن الرحيم

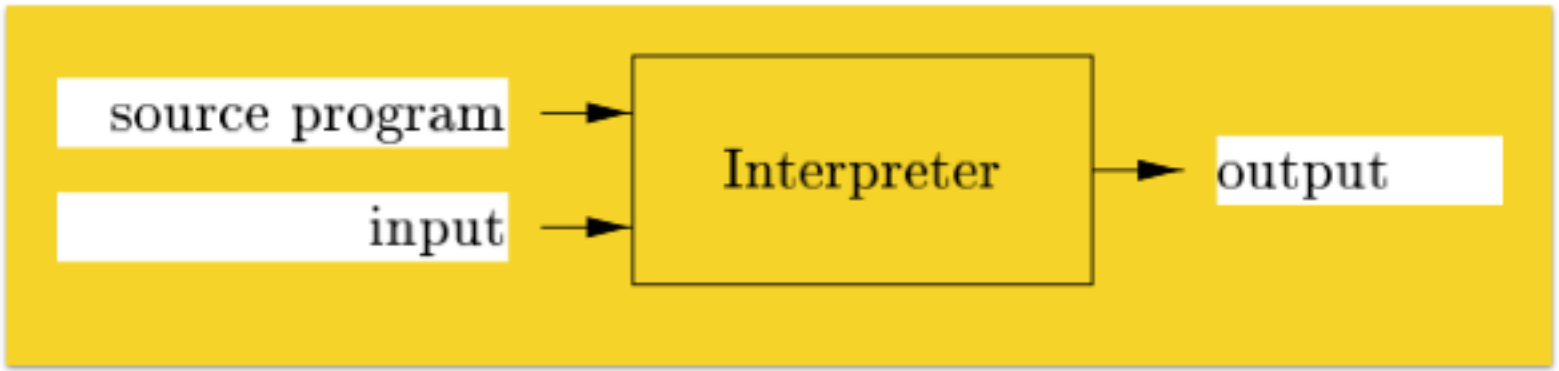
مقدمه

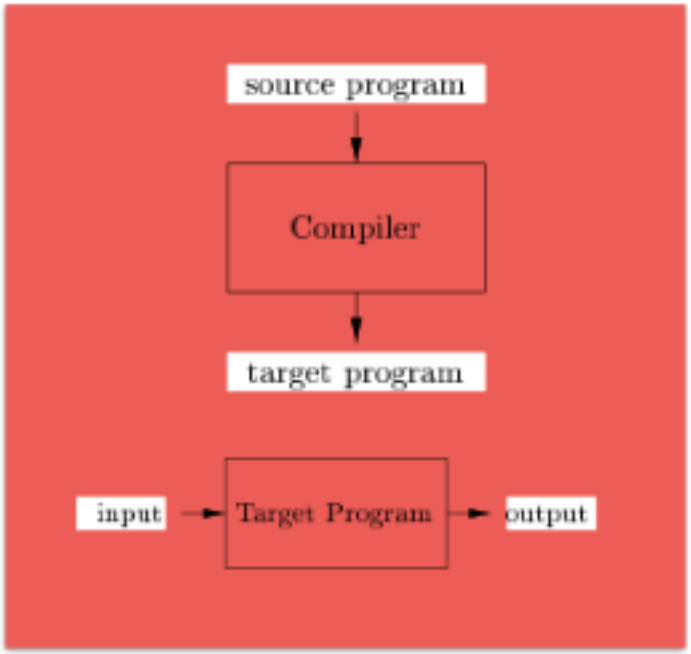
کامپایلر

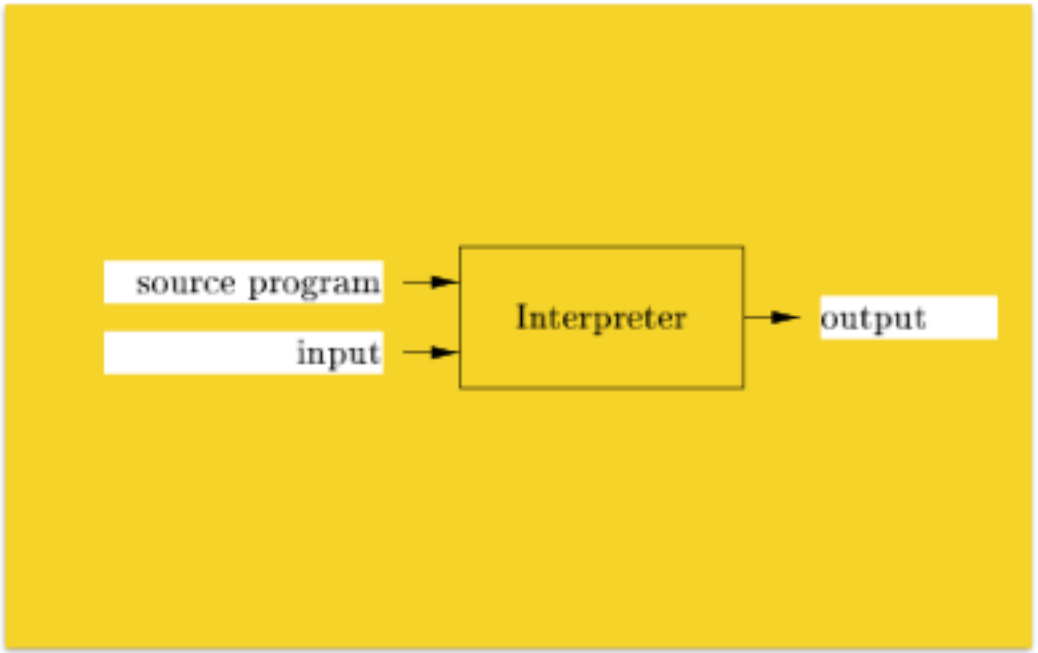
فهرست

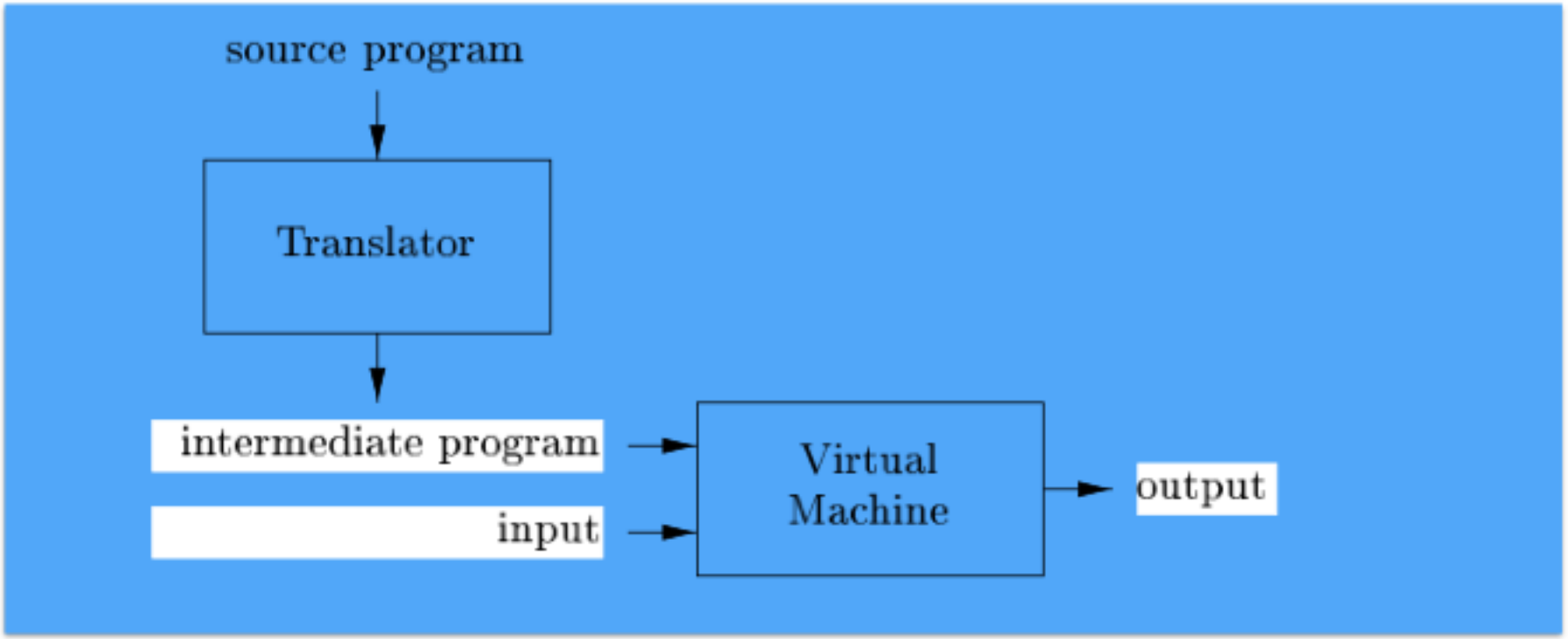
- ساختار کامپایلر
- تکامل زبان‌های برنامه‌سازی
- برخی مفاهیم زبان‌های برنامه‌سازی


source program
Compiler
target program
input
Target Program
output

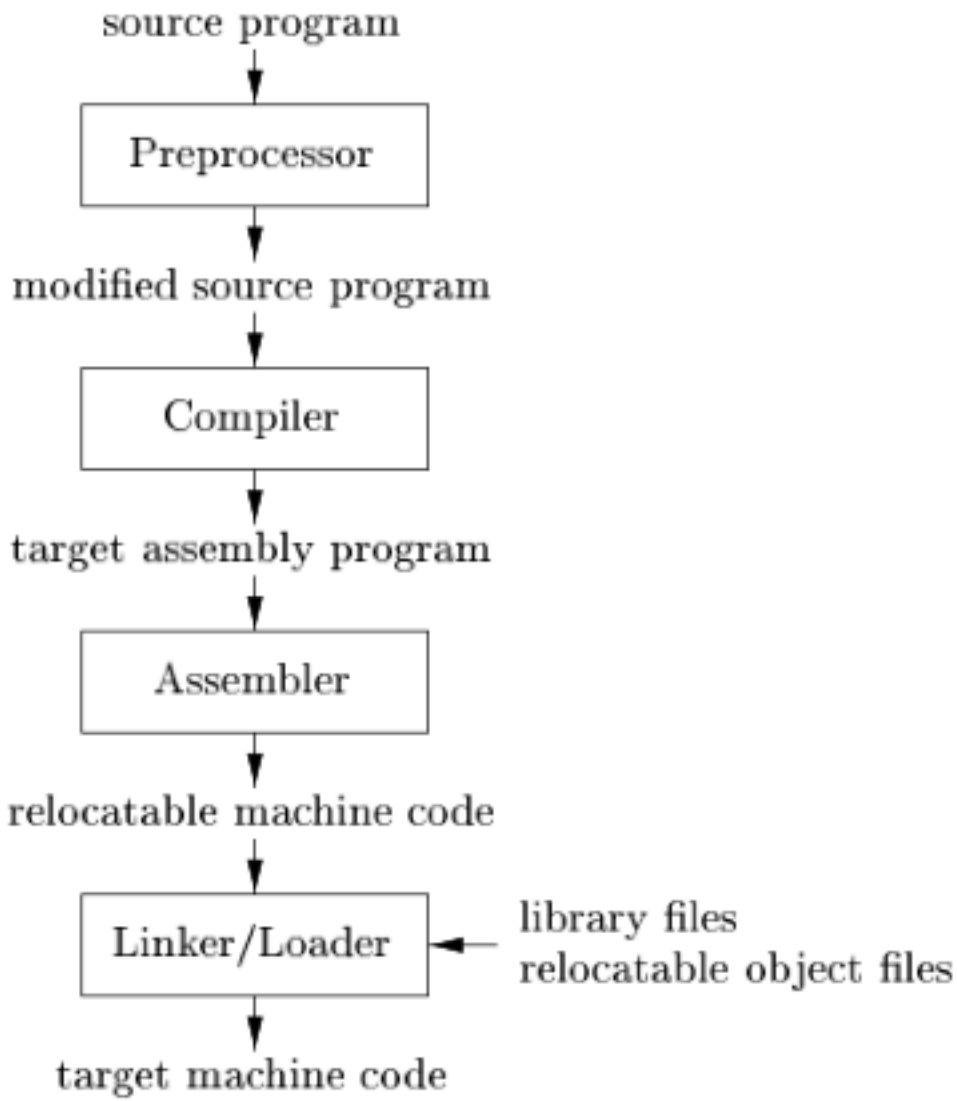

source program
input
Interpreter
output


source program
Compiler
target program
input
Target Program
output

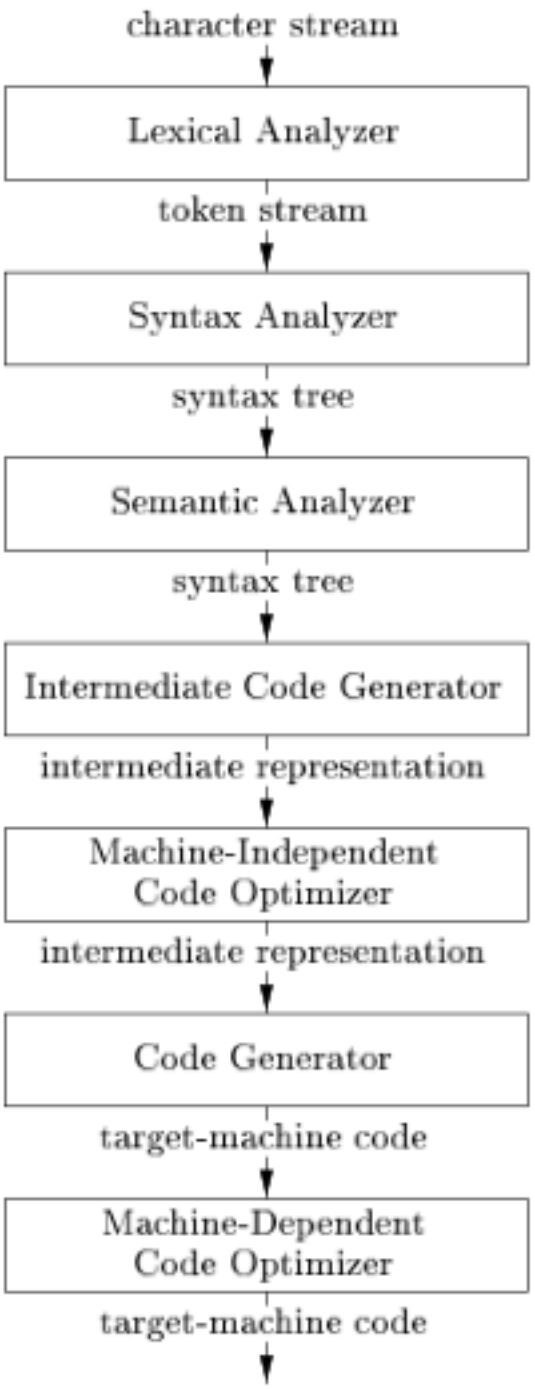

source program
input
Interpreter
output

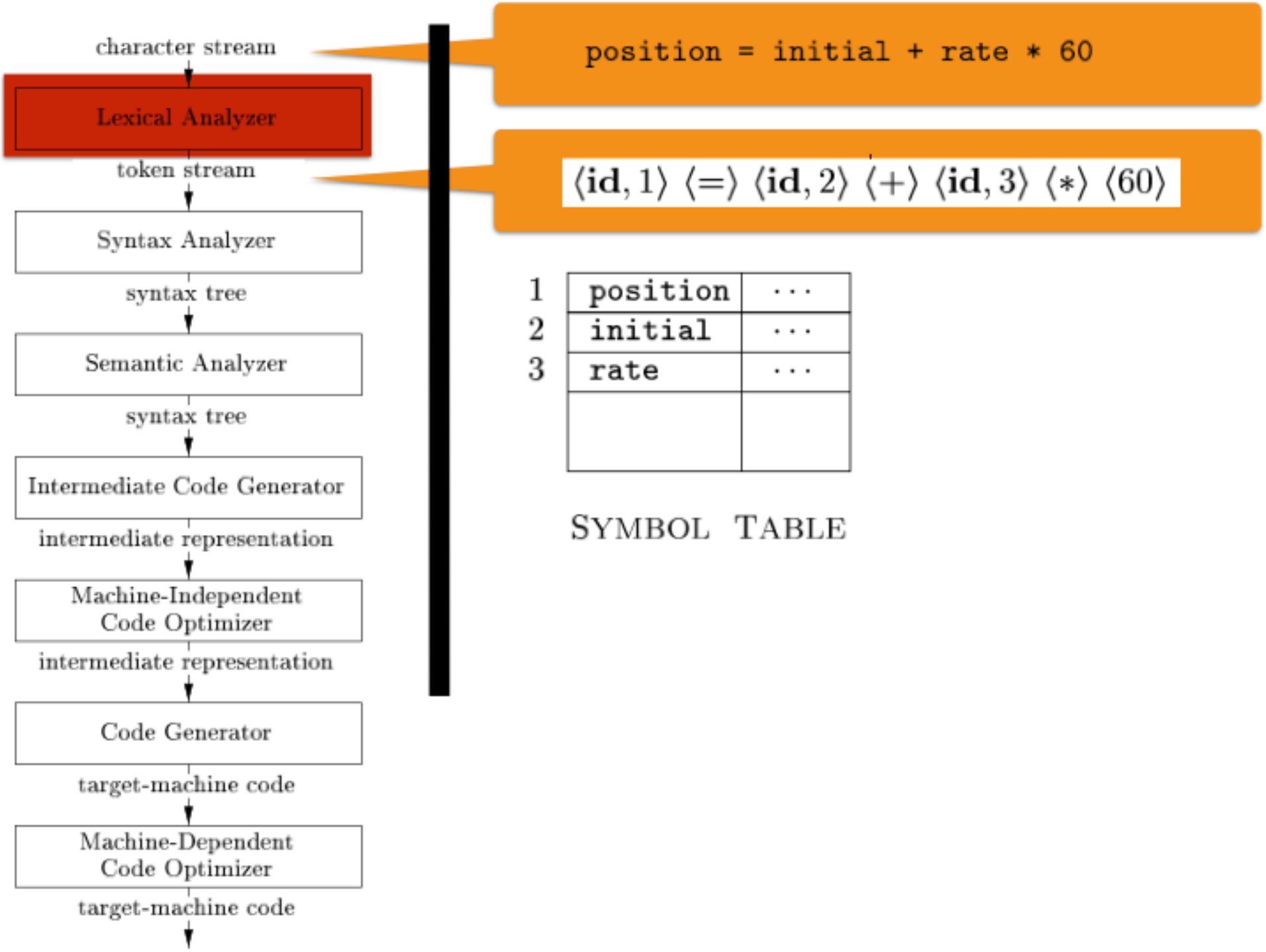

source program
Translator
intermediate program
input
Virtual
Machine
output

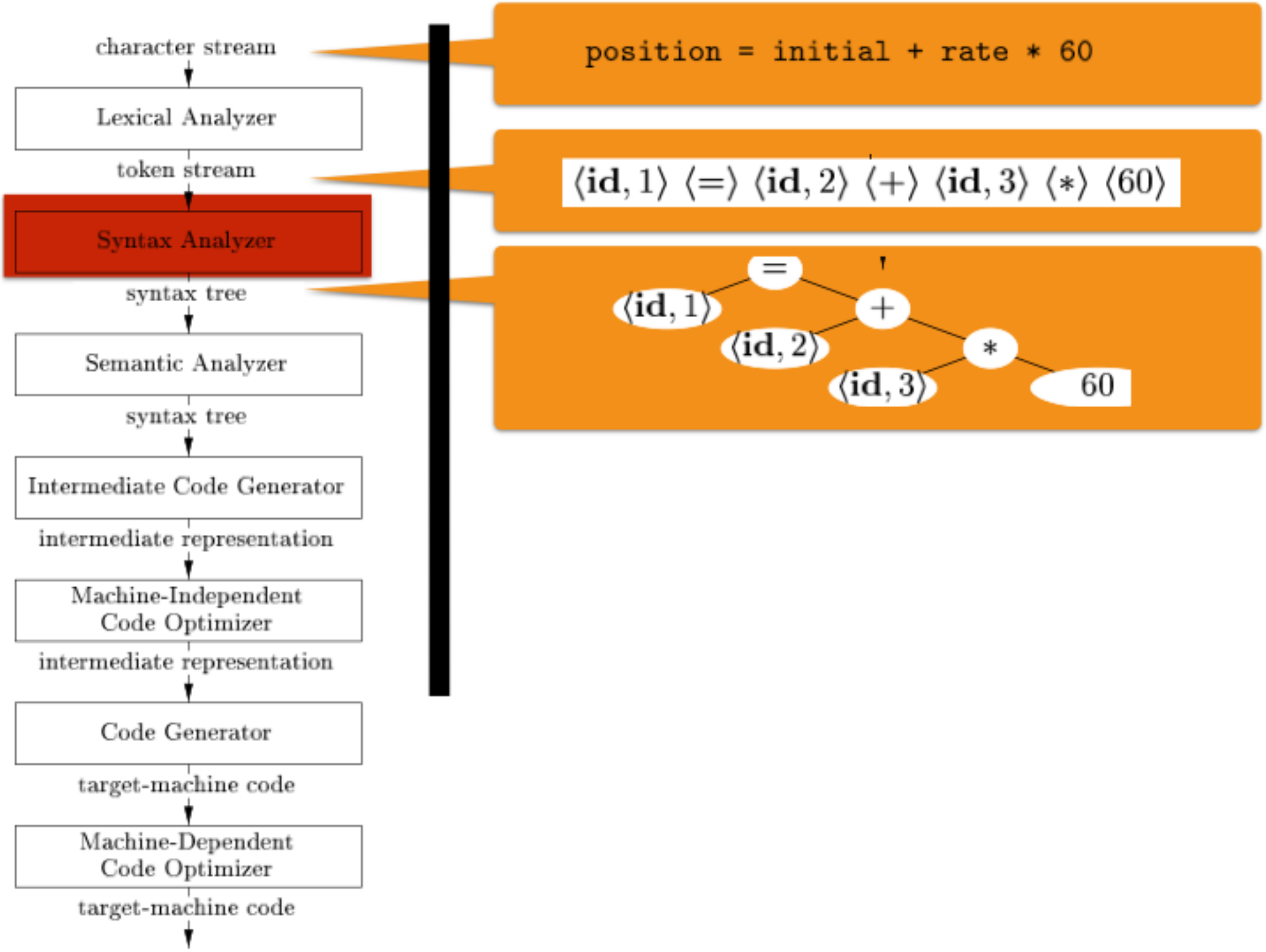
سرنوشت یک برنامه

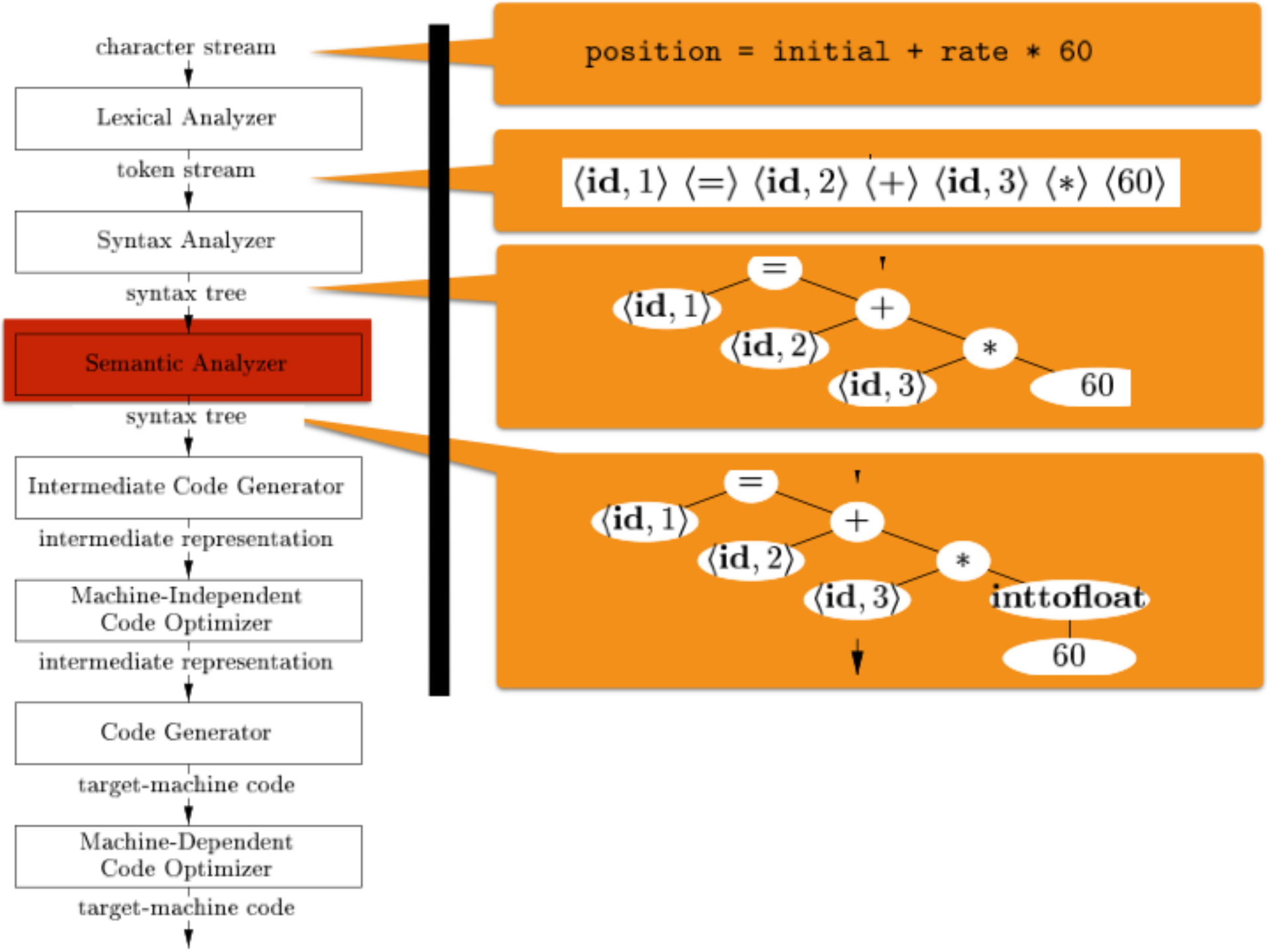


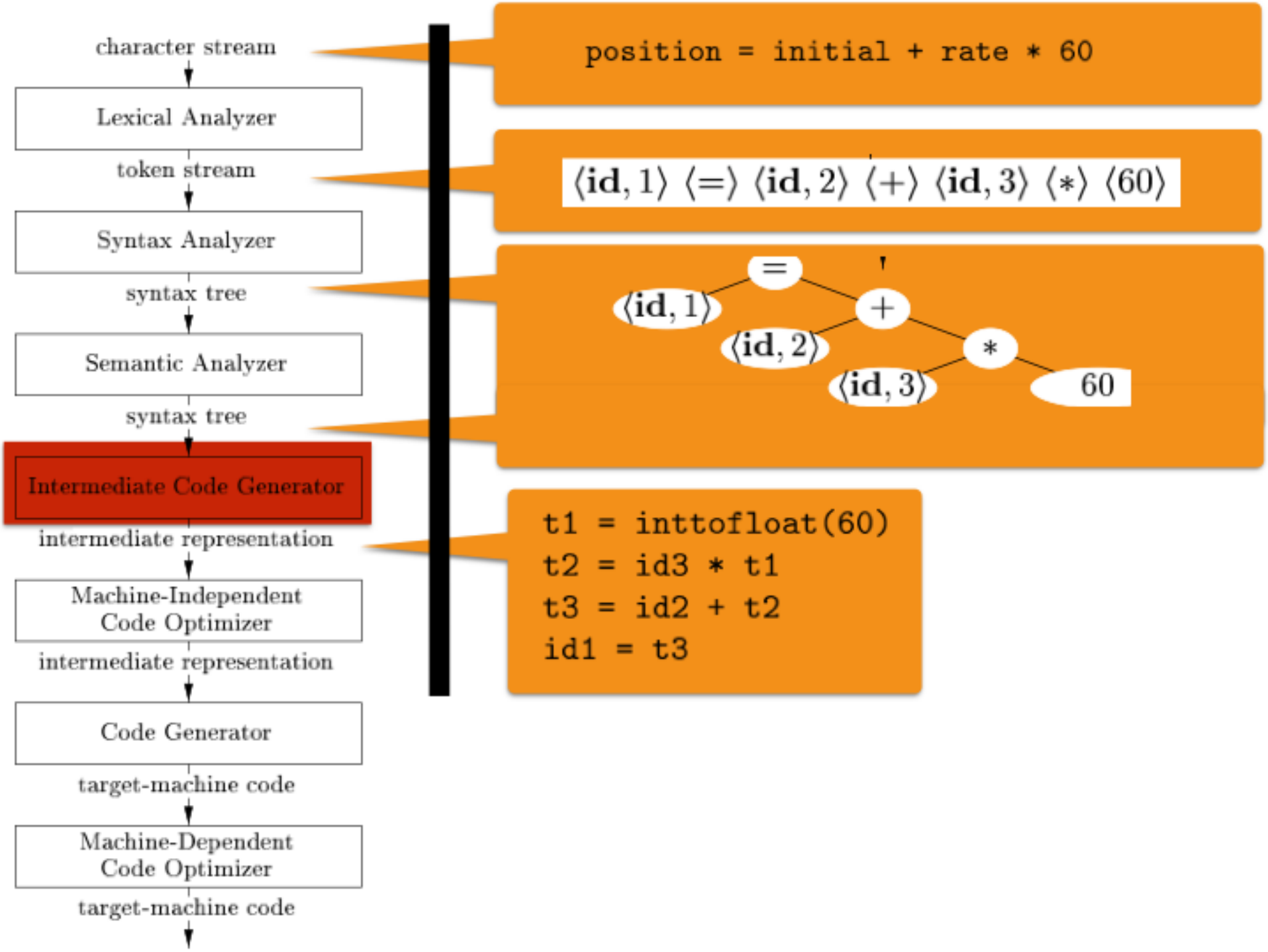
ساختار یک کامپایلر

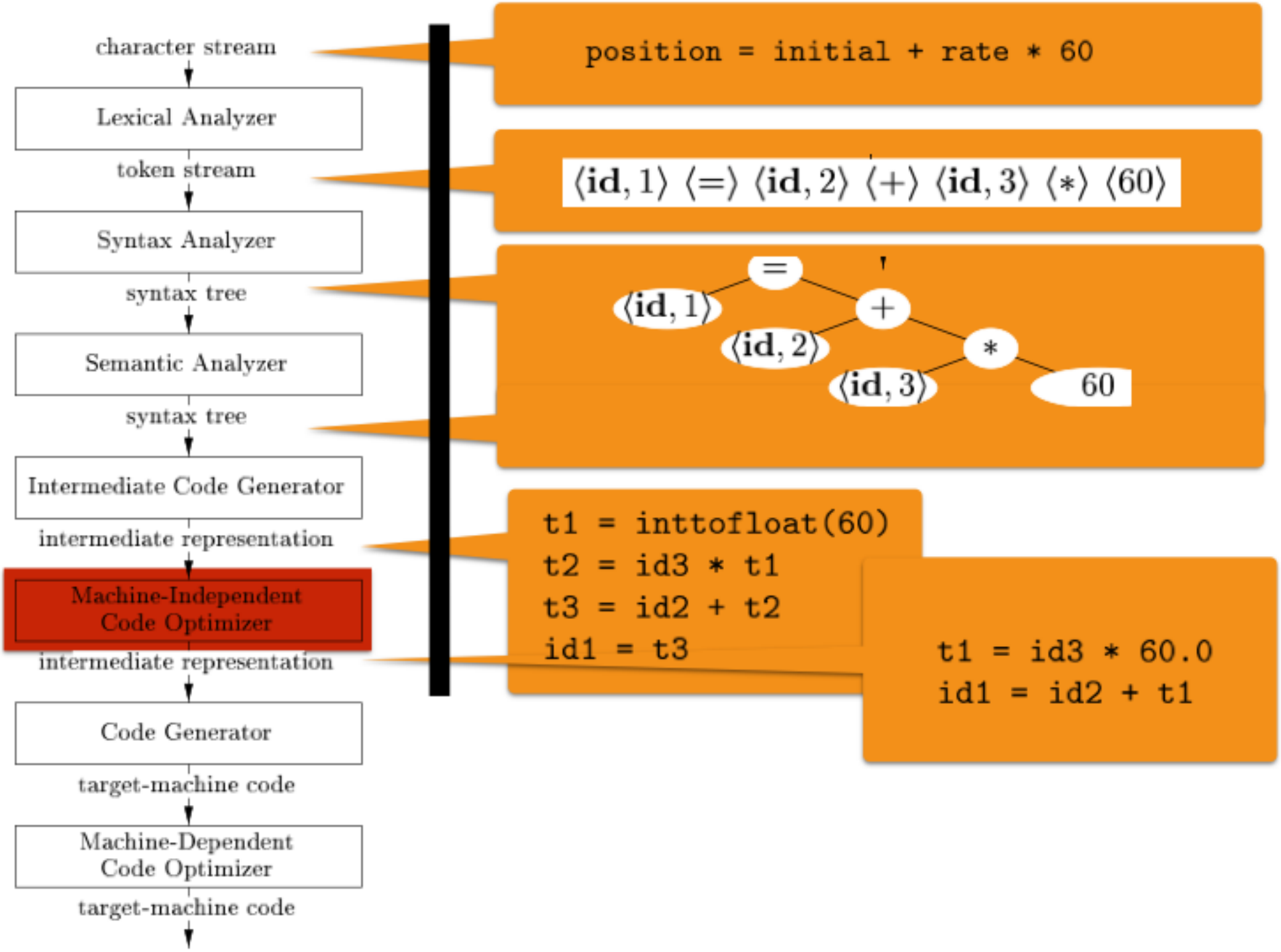


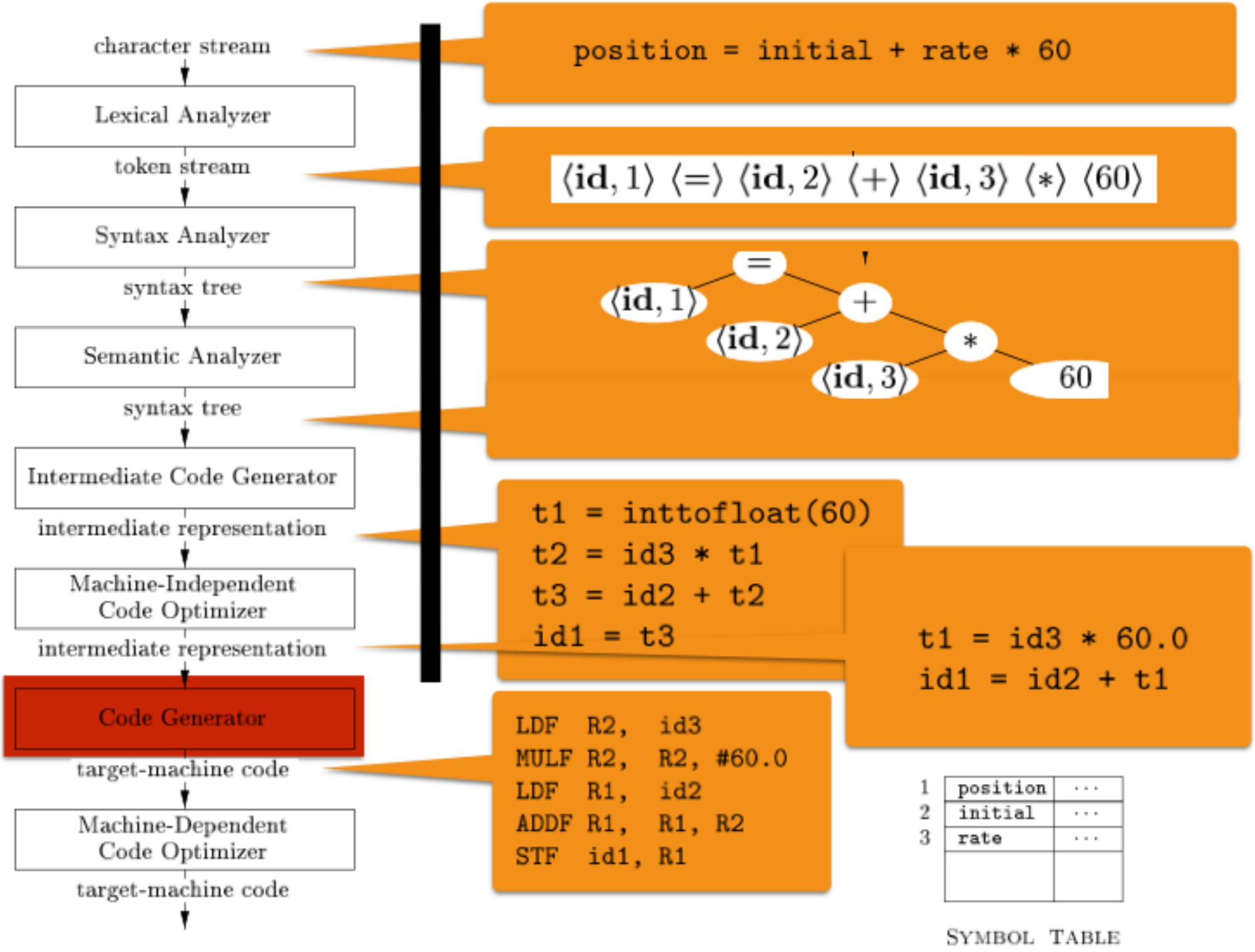

character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨∗⟩ ⟨60⟩
1 position ···
2 initial ···
3 rate ···
SYMBOL TABLE

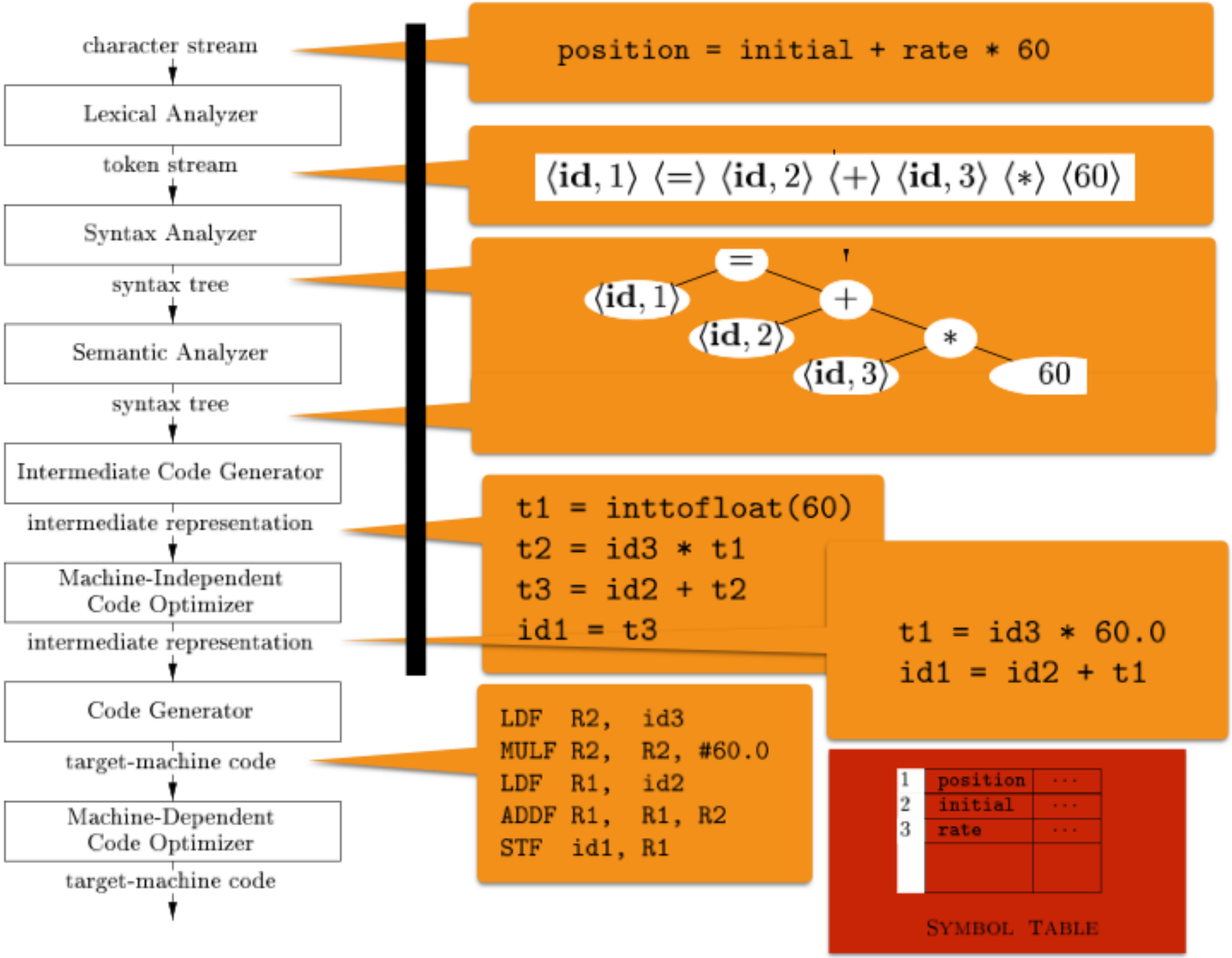

character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60

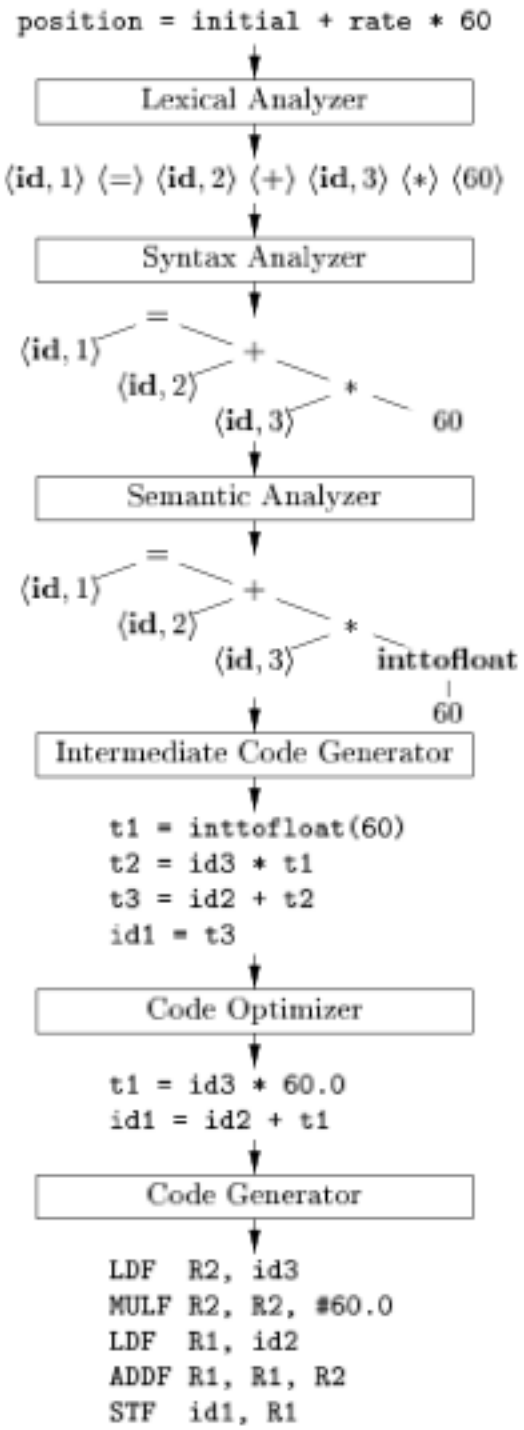

character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
inttofloat
60


character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3


character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
t1 = id3 * 60.0
id1 = id2 + t1

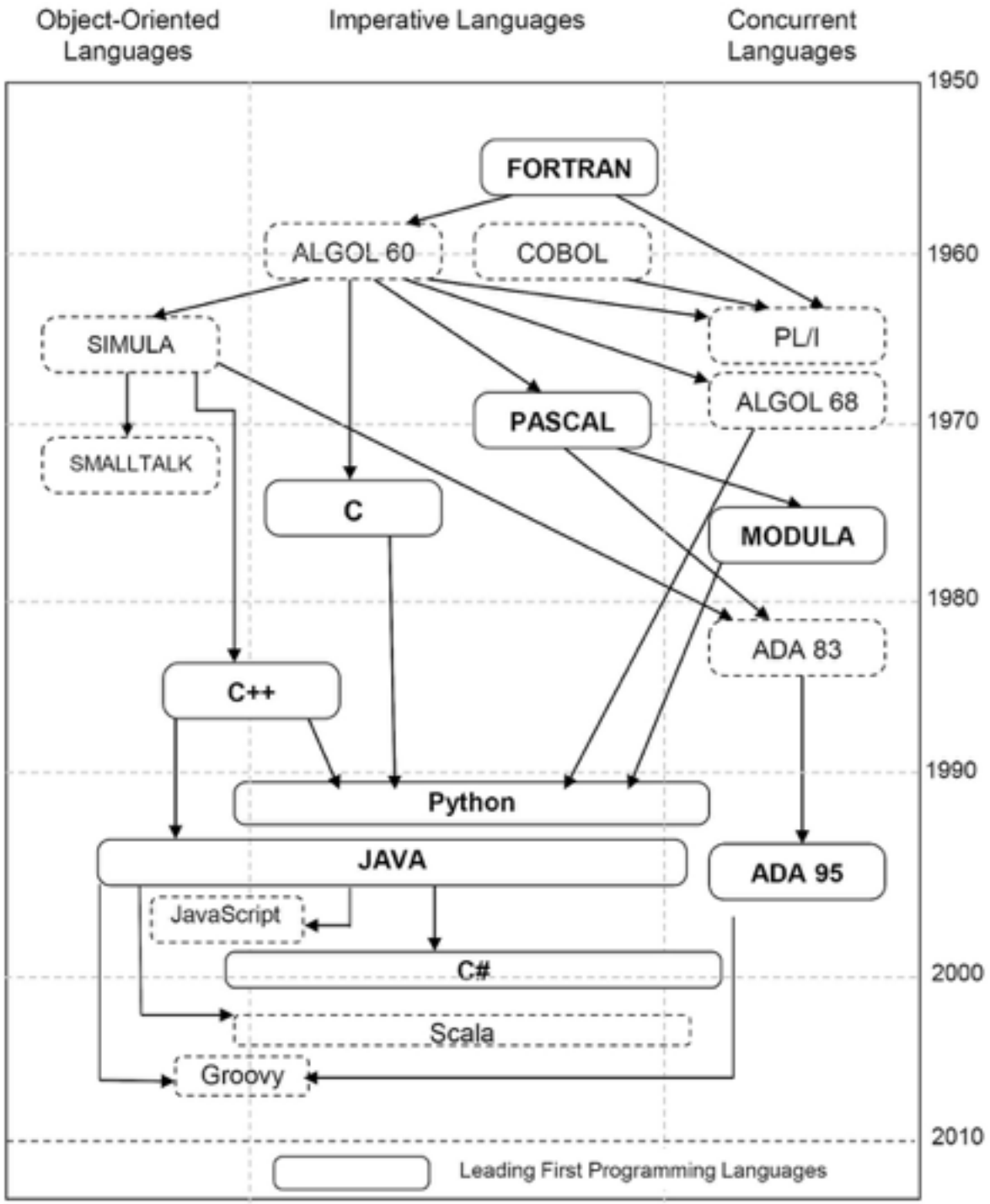

character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
t1 = id3 * 60.0
id1 = id2 + t1
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
1 position ···
2 initial ···
3 rate ···
SYMBOL TABLE

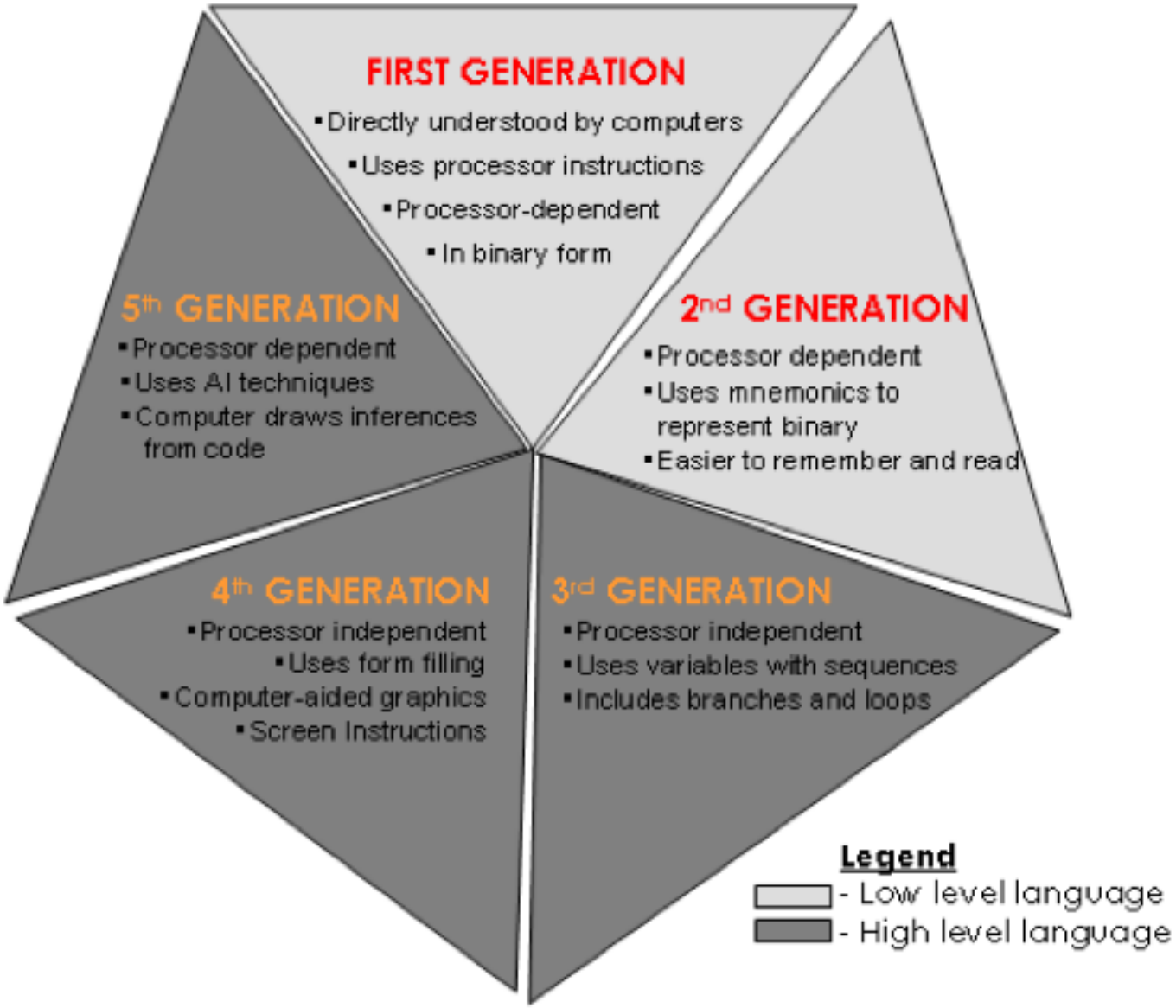

character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
position = initial + rate * 60
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
t1 = id3 * 60.0
id1 = id2 + t1
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
1 position ...
2 initial ...
3 rate ...
SYMBOL TABLE



جمع بندی مراحل کامپایلر

تکامل زبان‌های برنامه‌سازی


Object-Oriented Languages
Imperative Languages
Concurrent Languages
1950
1960
1970
1980
1990
2000
2010
FORTRAN
ALGOL 60
COBOL
SIMULA
PL/I
ALGOL 68
PASCAL
SMALLTALK
C
MODULA
ADA 83
C++
Python
JAVA
ADA 95
JavaScript
C#
Scala
Groovy
Leading First Programming Languages


FIRST GENERATION
▪ Directly understood by computers
▪ Uses processor instructions
▪ Processor-dependent
▪ In binary form
2nd GENERATION
▪ Processor dependent
▪ Uses mnemonics to represent binary
▪ Easier to remember and read
3rd GENERATION
▪ Processor independent
▪ Uses variables with sequences
▪ Includes branches and loops
4th GENERATION
▪ Processor independent
▪ Uses form filling
▪ Computer-aided graphics
▪ Screen Instructions
5th GENERATION
▪ Processor dependent
▪ Uses AI techniques
▪ Computer draws inferences from code
Legend
- Low level language
- High level language

زبان‌های شیئ‌گرا

Simula, Smalltalk
C++ C# Java Ruby

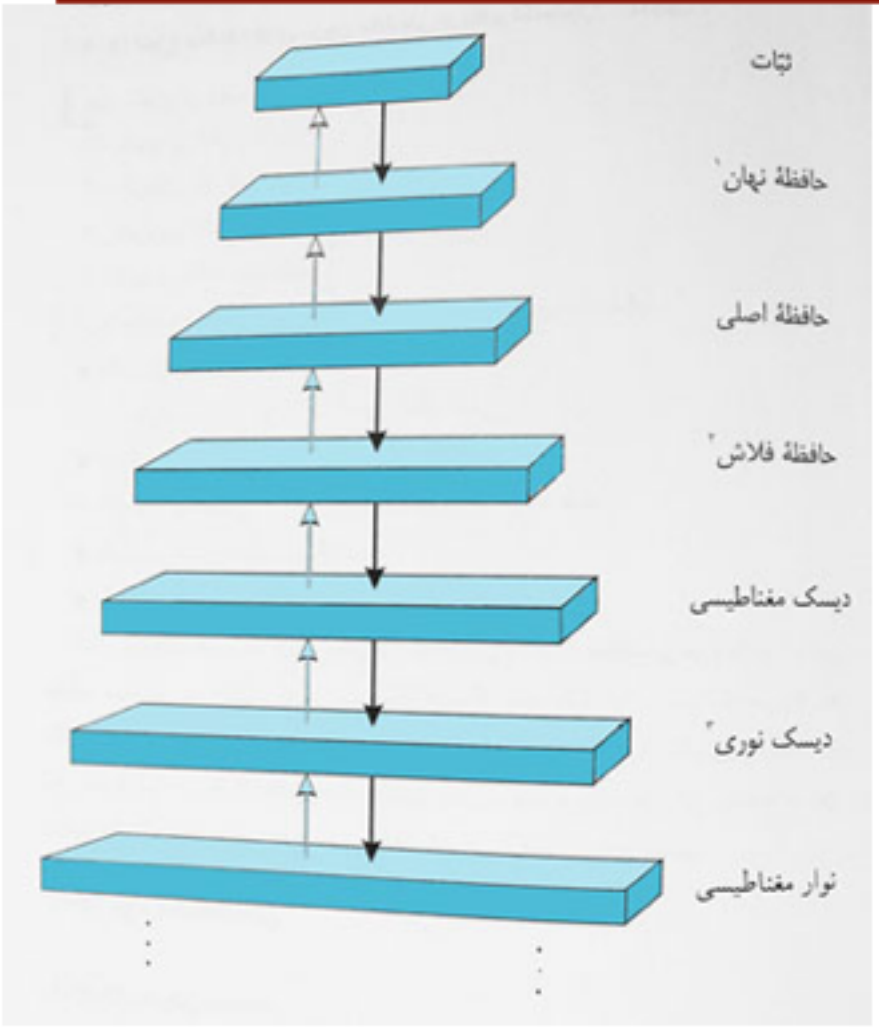
اسکریپت

Awk JavaScript Perl PHP Python Ruby Tcl

زبان سطح بالا

- چک کردن نوع
- چک آرایه
- فارغ از جزئیات

بهینه‌سازی برای پردازنده



- موازی‌سازی: خط لوله، دستورات موازی، چندریسمانی
- سلسله‌مراتب حافظه:
- ساختار پردازنده: RISC و CISC و GPU

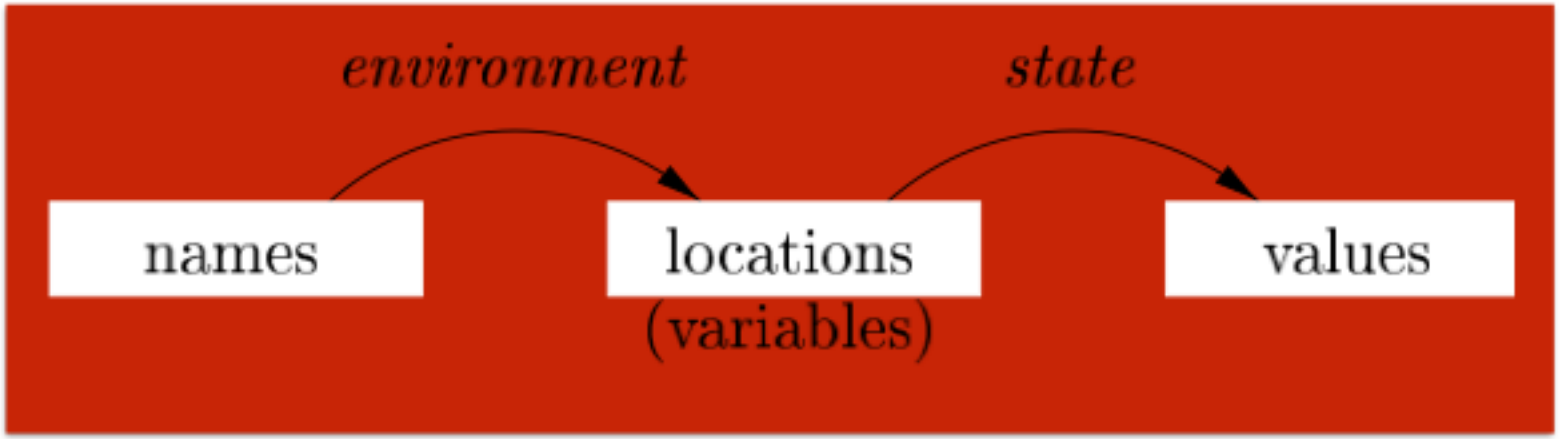
برخی مفاهیم زبان‌های برنامه‌نویسی

ایستا و پویا

- زمان کامپایل / زمان اجرا
- حیطه ایستا و حیطه پویا

محیط و حالت

```
...
int i;                    /* global i       */
...
void f(···) {
      int i;              /* local i        */
      ...
      i = 3;              /* use of local i */
      ...
}
...
      x = i + 1;          /* use of global i */
```



محیط و حالت (ادامه)

- نسبت ایستا و پویای نام به محیط (متغیر محلی و سراسری)
- نسبت ایستا و پویای مقدار به متغیر

حیطه ایستا

```
main() {
    int a = 1;                          B1
    int b = 1;
    {
        int b = 2;                      B2
        {
            int a = 3;                  B3
            cout << a << b;
        }
        {
            int b = 4;                  B4
            cout << a << b;
        }
        cout << a << b;
    }
    cout << a << b;
}
```

کنترل دسترسی

- public, private, protected
- مانند حیطه
- برخی تابع‌های کلاس
- برخی تابع‌های بیرون کلاس که جزو کلاس هستند

حیطه پویا (استثناء)

نمی‌توان آدرس را در زمان کامپایل به دست آورد

```
#define a (x+1)
int x = 2;
void b() { int x = 1; printf("%d\n", a); }
void c() { printf("%d\n", a); }
void main() { b(); c(); }
```

وراثت چندگانه: متغیر به کدام کلاس اشاره می‌کند

روش‌های انتقال متغیر

- با مقدار
- با ارجاع
- با اسم

نام مستعار

- function q(x,y) {}
- q(x,x);

با تشکر