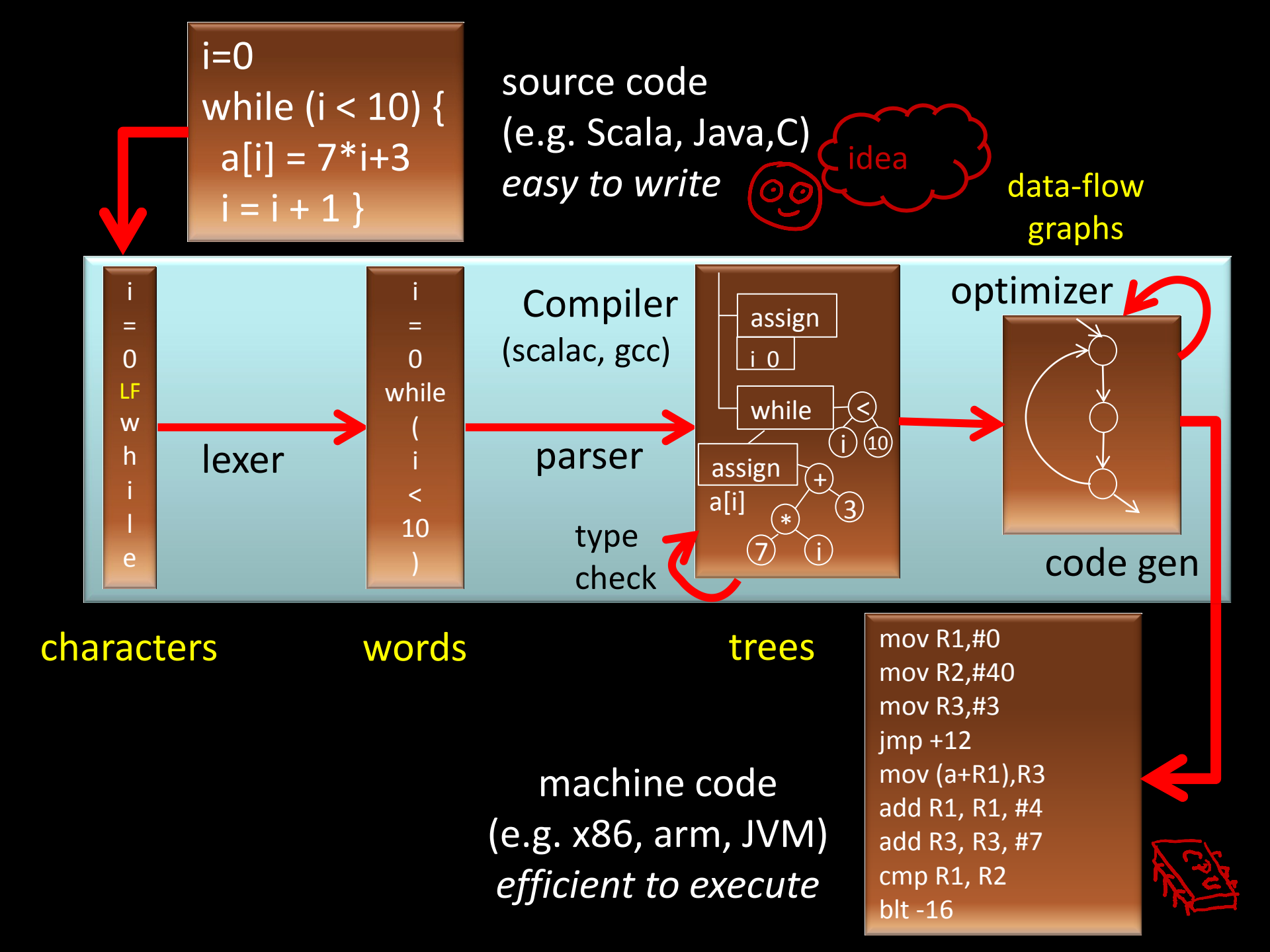
Compiler Construction 2010, Lecture 9

Code Generation


i=0
while (i < 10) {
a[i] = 7*i+3
i = i + 1 }
source code
(e.g. Scala, Java,C)
easy to write
idea
data-flow
graphs
i
=
0
LF
w
h
i
l
e
lexer
i
=
0
while
(
i
<
10
)
Compiler
(scalac, gcc)
parser
assign
i 0
while
<
i
10
assign
a[i]
+
3
*
7
i
type
check
optimizer
code gen
characters
words
trees
mov R1,#0
mov R2,#40
mov R3,#3
jmp +12
mov (a+R1),R3
add R1, R1, #4
add R3, R3, #7
cmp R1, R2
blt -16
machine code
(e.g. x86, arm, JVM)
efficient to execute
CPU

Example: gcc

```
#include <stdio.h>
int main() {
  int i = 0;
  int j = 0;
  while (i < 10) {
    printf("%d\n", j);
    i = i + 1;
    j = j + 2*i+1;
  }
}
```

gcc test.c -S

```
        jmp .L2
.L3:    movl -8(%ebp), %eax
        movl %eax, 4(%esp)
        movl $.LC0, (%esp)
        call printf
        addl $1, -12(%ebp)
        movl -12(%ebp), %eax
        addl %eax, %eax
        addl -8(%ebp), %eax
        addl $1, %eax
        movl %eax, -8(%ebp)
.L2:
        cmpl $9, -12(%ebp)
        jle .L3
```

Amusing Question

maximum optimization

- What does this produce (with GCC 4.2.4)

  gcc -O3 test.c

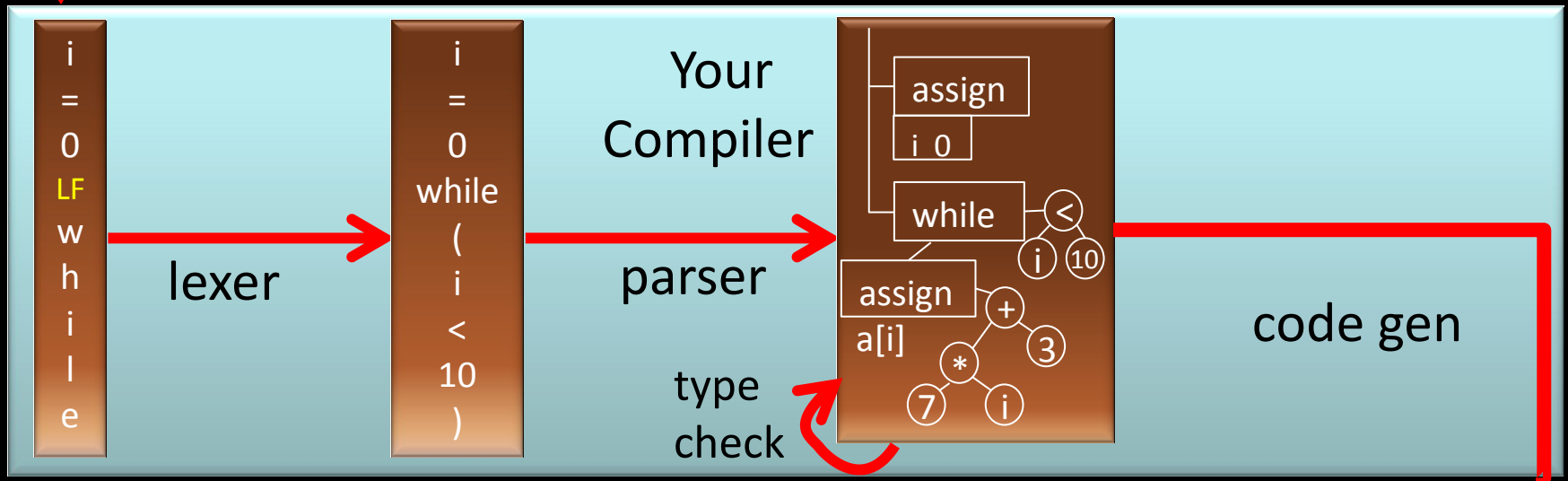
- No loop at all, a sequence corresponding to

  printf("...", 0)
  printf("...", 3)
  printf("...", 8)
  printf("...", 15)
  printf("...", 24)
  printf("...", 35)
  printf("...", 48)
  printf("...", 63)
  printf("...", 80)
  printf("...", 99)

A Simple Compiler

```
i=0
while (i < 10) {
  a[i] = 7*i+3
  i = i + 1 }
```

source code
simplified Java-like
language



characters words trees

JVM
Code

```
21: iload_2
22: iconst_2
23: iload_1
24: imul
25: iadd
26: iconst_1
27: iadd
28: istore_2
```

javac example

```
while (i < 10) {
  System.out.println(j);
  i = i + 1;
  j = j + 2*i+1;
}
```

javac Test.java
javap –c Test

```
4: iload_1
5: bipush 10
7: if_icmpge 32
10: getstatic #2; //System.out
13: iload_2
14: invokevirtual #3; //println
17: iload_1
18: iconst_1
19: iadd
20: istore_1
21: iload_2
22: iconst_2
23: iload_1
24: imul
25: iadd
26: iconst_1
27: iadd
28: istore_2
29: goto 4
32: return
```

You will build a compiler that generates such code