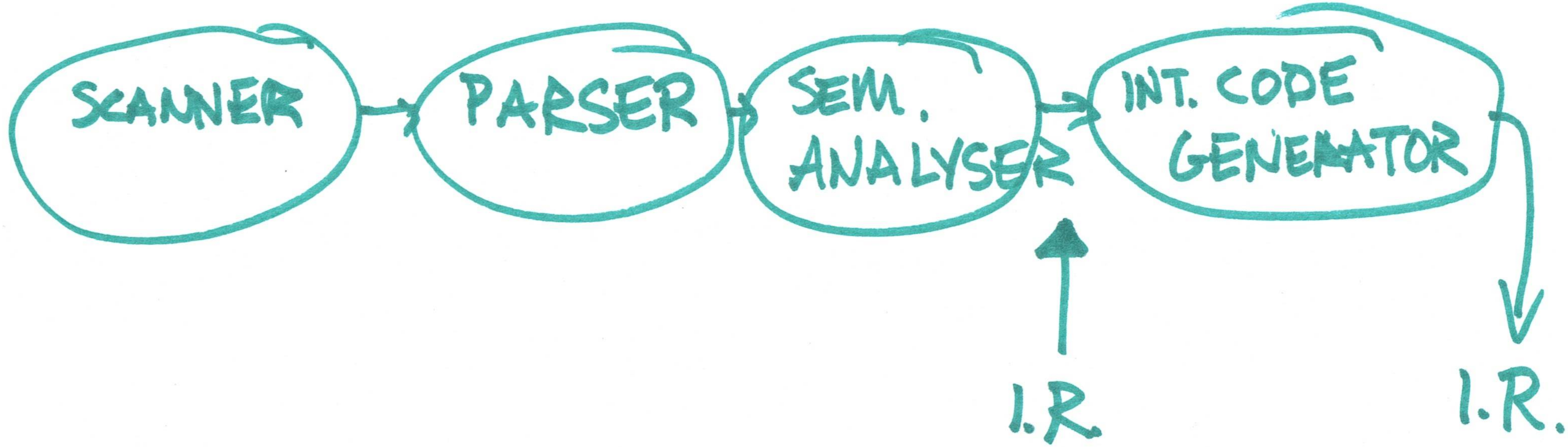
INTERMEDIATE REPRESENTATIONS

- GRAPHICAL
- LINEAR — 1-ADDRESS CODE (1AC)
  3-ADDRESS CODE (3AC)
  (BASIC BLOCKS)
- HYBRID
  - CONTROL-FLOW GRAPH (CFG)

EXPRESSION DAGS

STATIC SINGLE ASSIGNMENT FORM
(SSA)

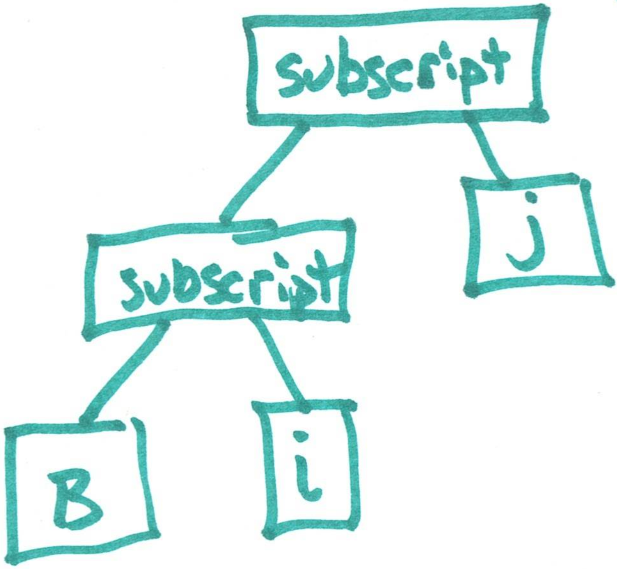


USE THE I.R. DURING SYNTHESIS AS THE REFERENCE FORM OF THE PROGRAM (THE SOURCE IS NOT CONSULTED)

MOST PHASES USE AN I.R. MANY COMPILERS USE MORE THAN ONE.

ABSTRACTION LEVEL OF AN I.R. CAN BE LOW OR HIGH. (OR ANYTHING INBETWEEN)

B[ ][ ] with 4-byte elements & subelements, both subscripts running from 1 to 10, and we wish to represent the array reference B[i][j]



high level tree

low-level linear

$r_1 = \text{load } B$

$r_2 = i - 1$

$r_3 = r_2 * 4$

$r_4 = r_1 + r_3$

$r_5 = \text{load } r_4$

$r_6 = j - 1$

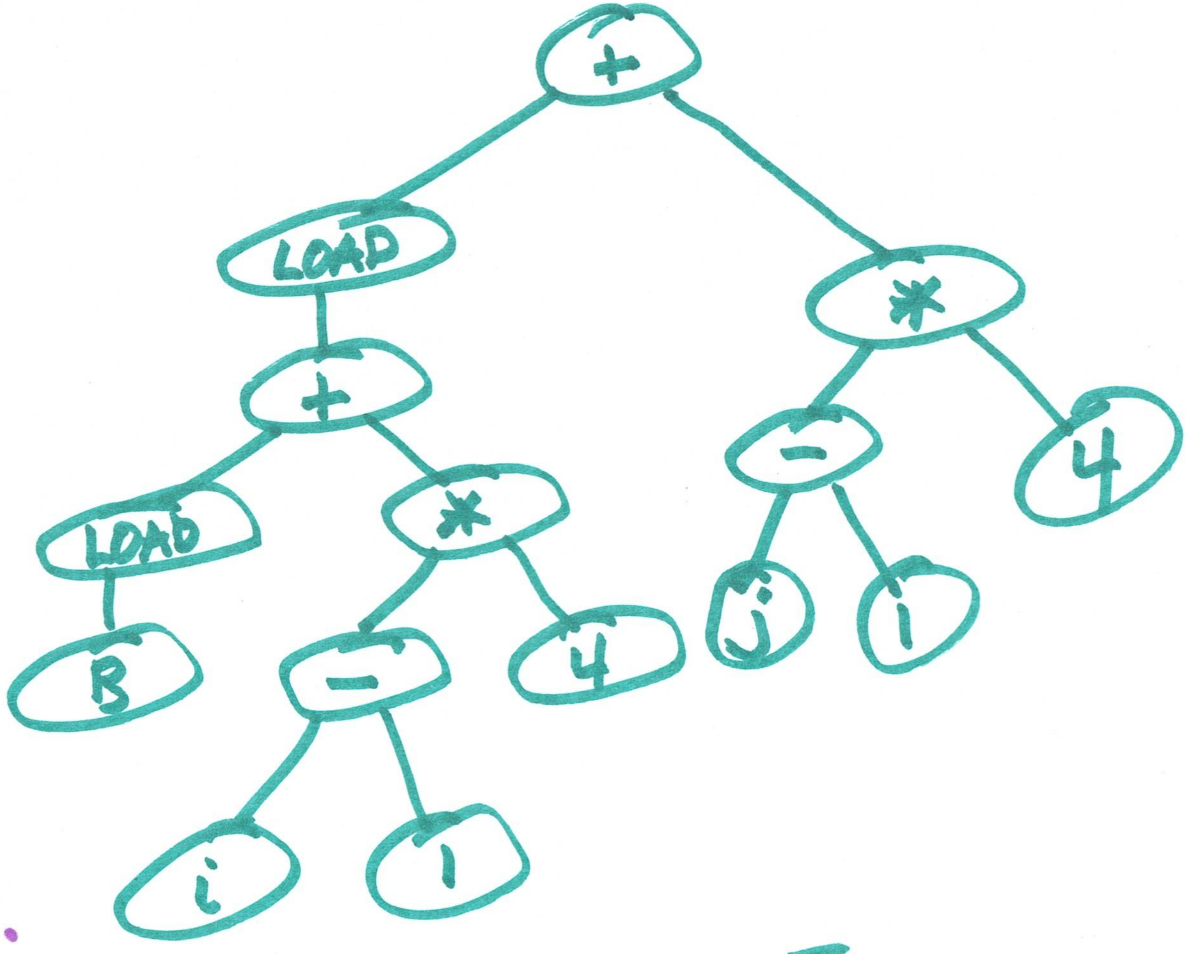
$r_7 = r_6 * 4$

$r_8 = r_5 + r_7$

HIGH-LEVEL LINEAR

$r_1 = B[i]$

$r_2 = r_1[j]$



LOW-LEVEL TREE.

examples:

HIGH-LEVEL

ALIAS ANALYSIS.

LOW-LEVEL

LOOP INVARIANT EXPRESSIONS.

IRs: NAMING OF VALUES

$t_1 = b$
$t_2 = 2 * t_1$
$t_3 = a$
$t_4 = t_3 - t_2$

(no name re-use)

more common +
more flexible

$t_1 = b$
$t_1 = 2 * t_1$
$t_2 = a$
$t_1 = t_2 - t_1$

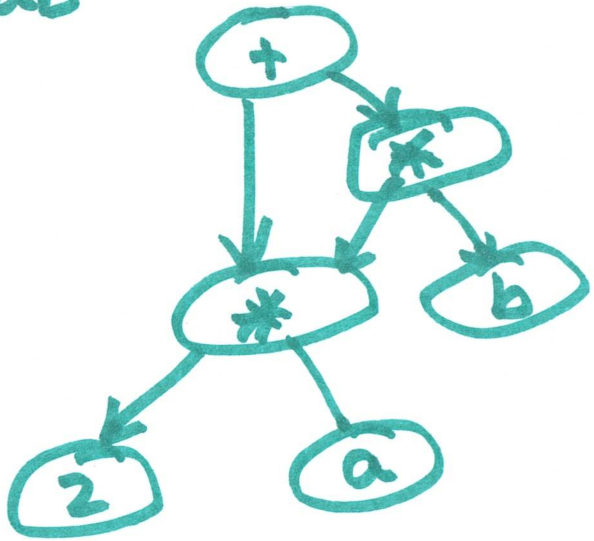
(re-use names)

GRAPHICAL IRs

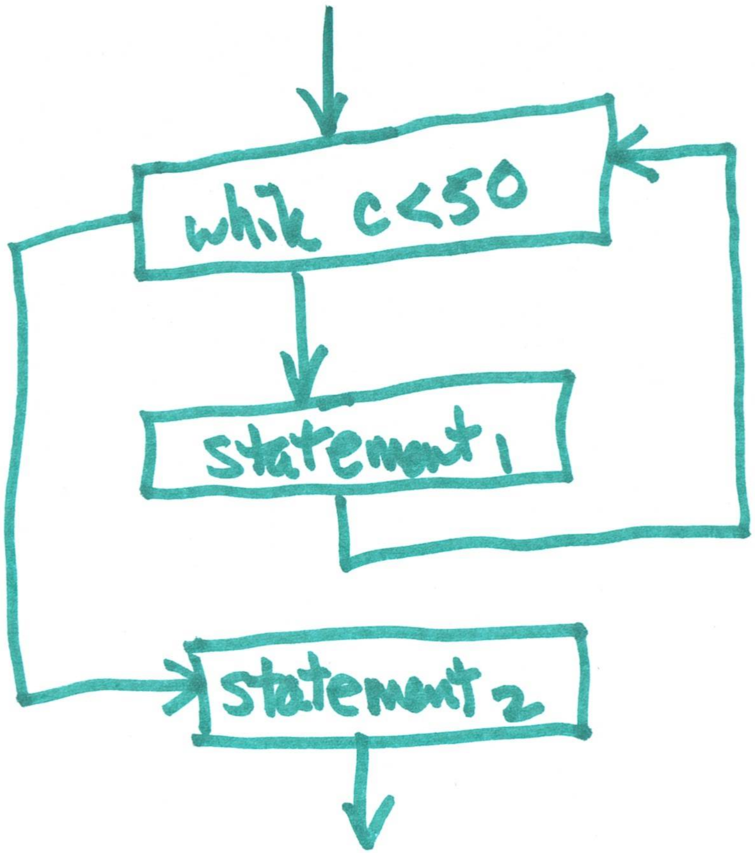
- AST — high level / low level

- EXPRESSION DAG

2a+2ab



- CONTROL-FLOW GRAPHS

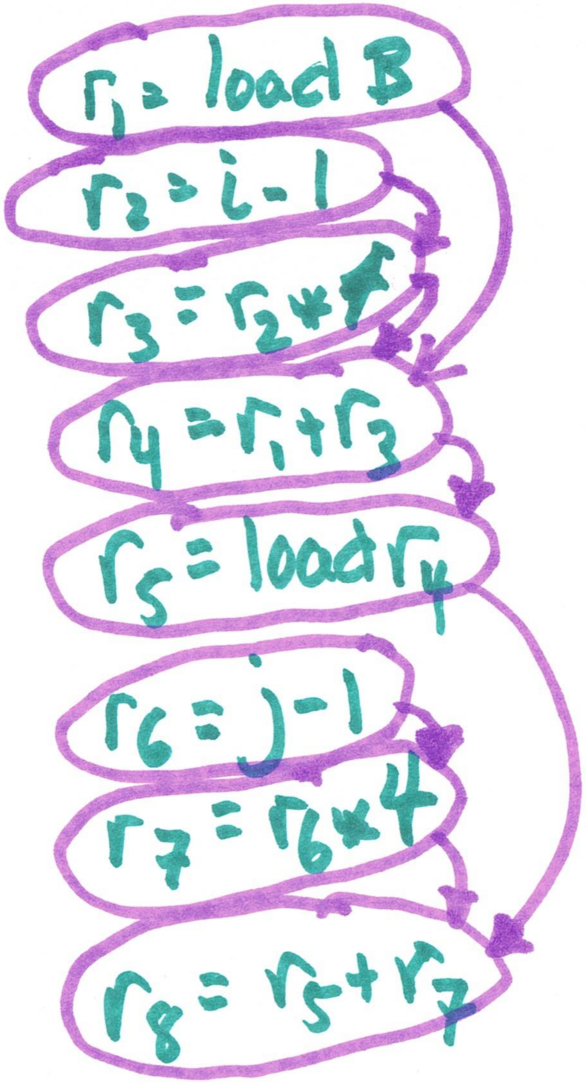


- CALL GRAPHS

A graph of which subroutines call which other subroutines.

- DEPENDENCE GRAPHS

which statements depend on values of which other statements



LINEAR IRS

- mostly resemble assembly code for some abstract machine.
- control-flow modelled after real machine - jumps & branches.

(1) one-address: stack machines
smalltalk, java, scala compiles to 1AC

(2) two-address: machine with destructive operations
(two operands & overwrite one of them
with the result). $t_2 = t_2 * t_1$

(3) three-address: machine has most operations with 2 operand
and a seperate place for the result.

$$\underline{t_3} = \underline{t_2} * \underline{t_1}$$