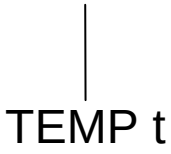
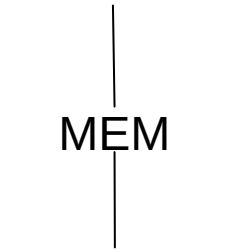
Translating to IR trees

Translating variables

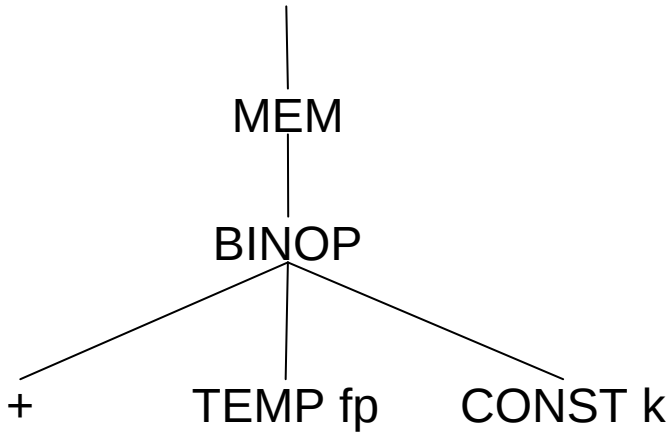
Temporary variable:



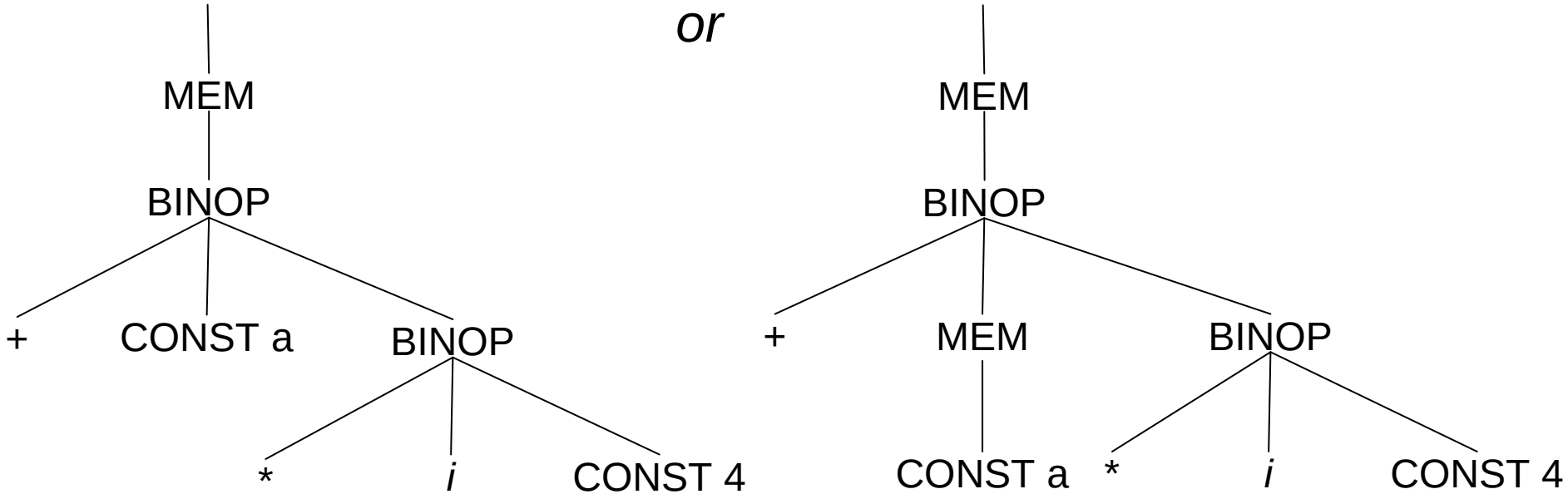
Variable in static memory location `m`:



Variable in memory location on stack, offset k:

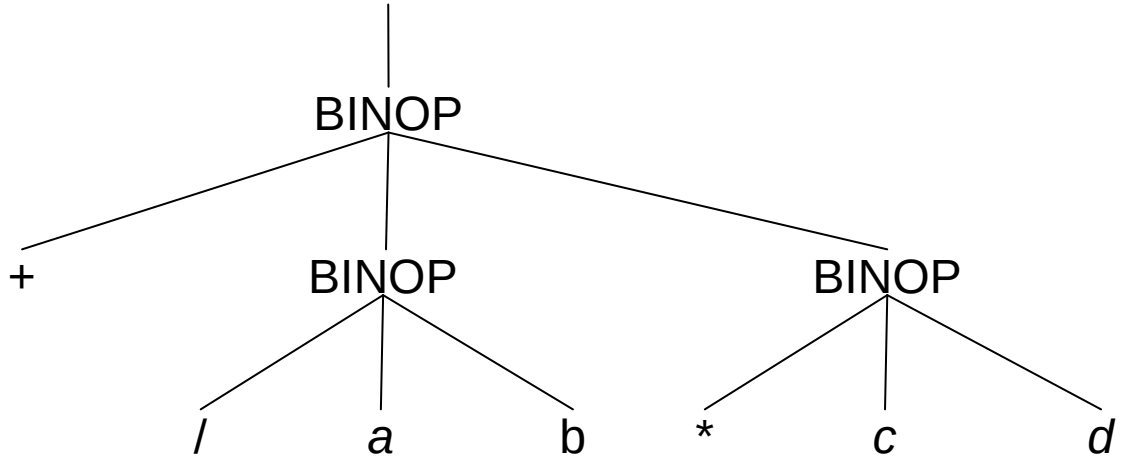


Array element `a[i]`:



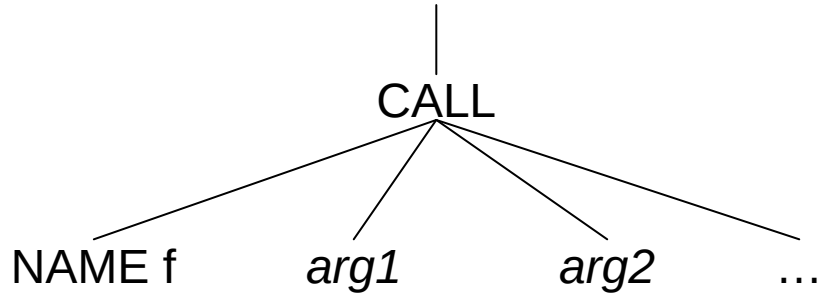
Translating expressions

Intermediate result = nonleaf node. E.g., `(a / b) + (c * d)`:



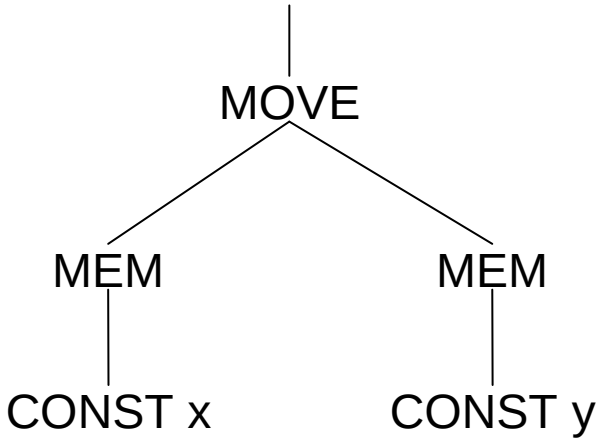
Translating function calls

`f(arg1, …, argn)`:



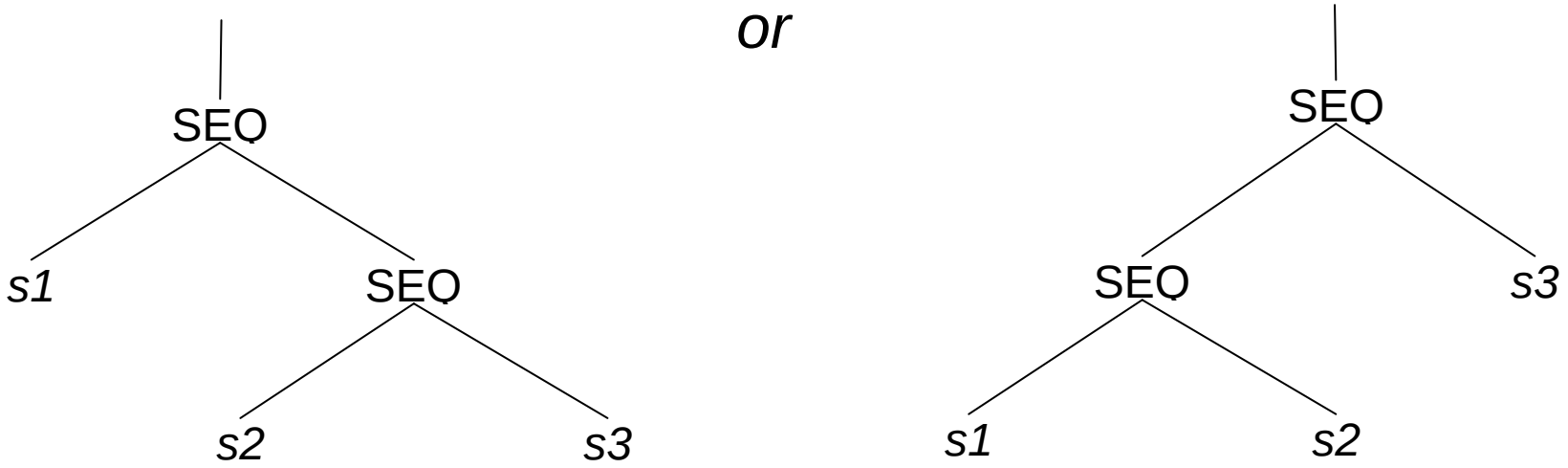
Translating assignment statements

Assignment `x = y`:



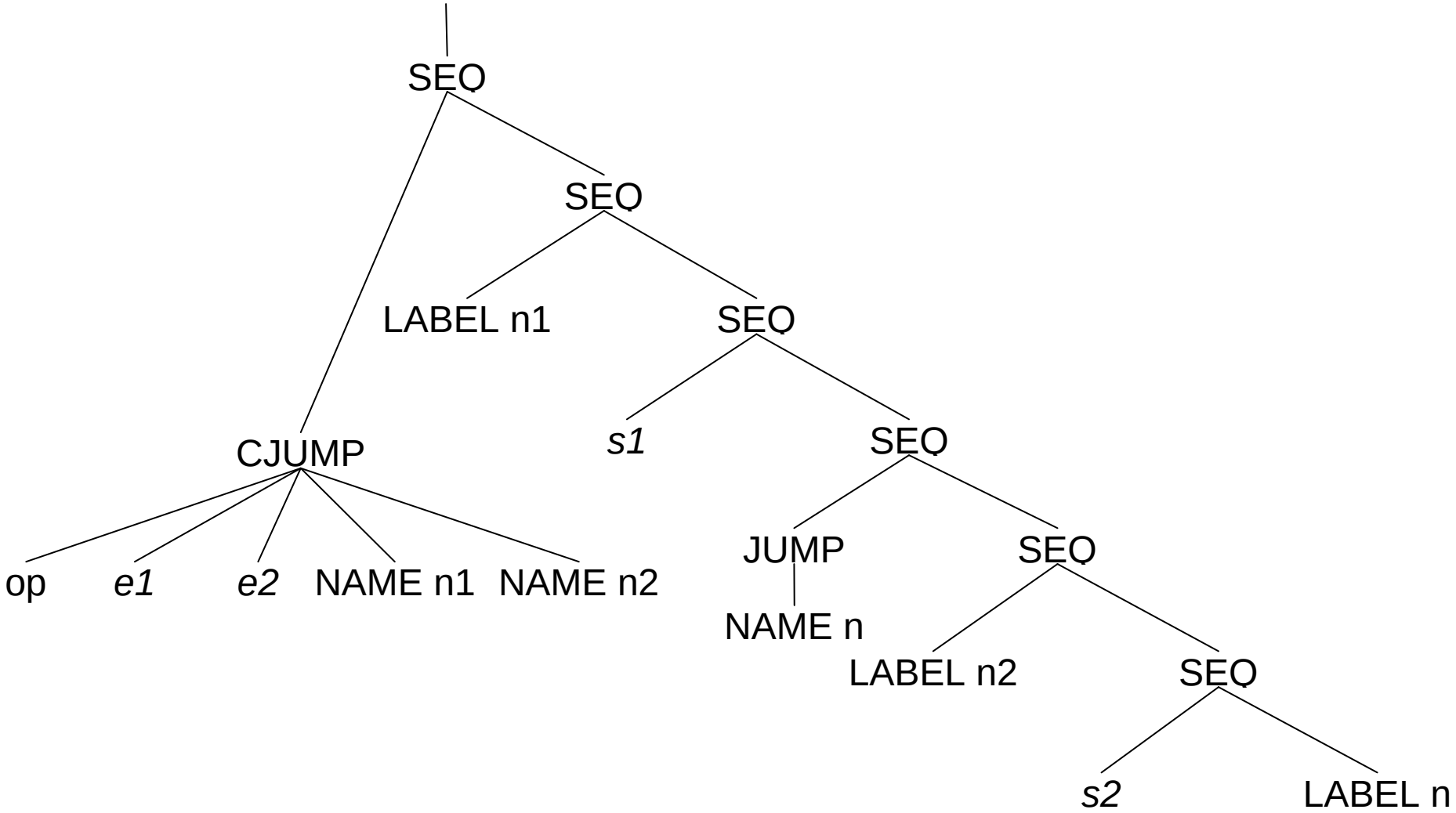
Translating sequence of statements

Sequence `s1; s2; s3`:



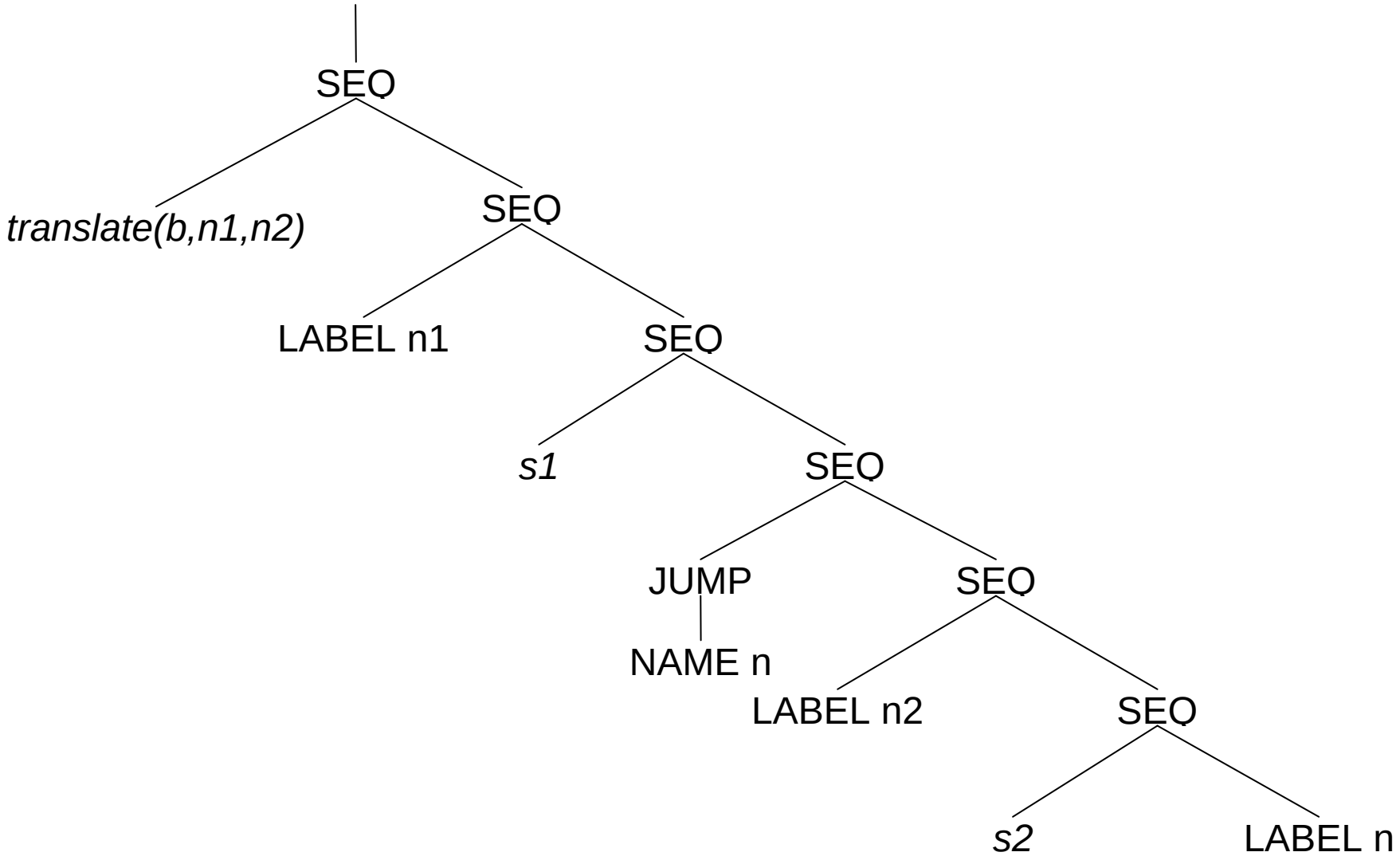
Translating conditionals

`if (e1 op e2) then s1 else s2`:



In general:

`if (b) then s1 else s2:`



where:

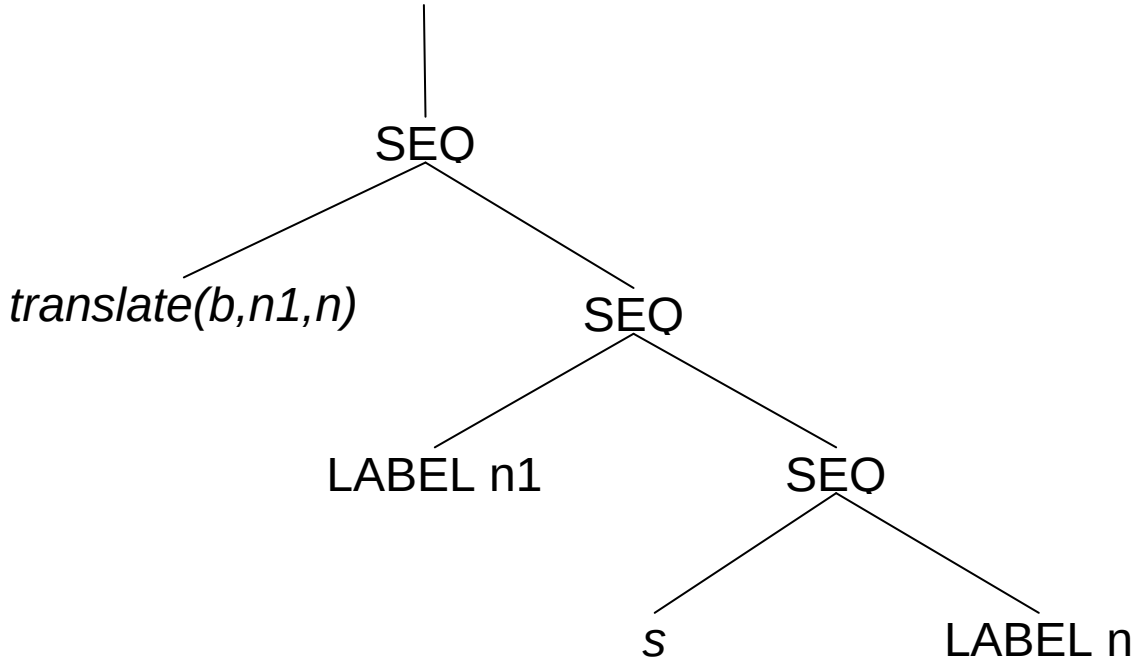
```
translate((b1 && b2), n1, n2) =>      translate(b1, next, n2)
                                next: translate(b2, n1, n2)
translate((b1 || b2), n1, n2) =>      translate(b1, n1, next)
                                next: translate(b2, n1, n2)
translate((e1 op e2), n1, n2) =>      CJUMP(op, e1, e2, NAME n1, NAME n2)
```

E.g.:

```
translate(((a<b && c<d) || e<f), n1, n2) =>
                                      CJUMP(<, a, b, next1, next)
                                next1: CJUMP(<, c, d, n1, next)
                                next:  CJUMP(<, e, f, n1, n2)
```

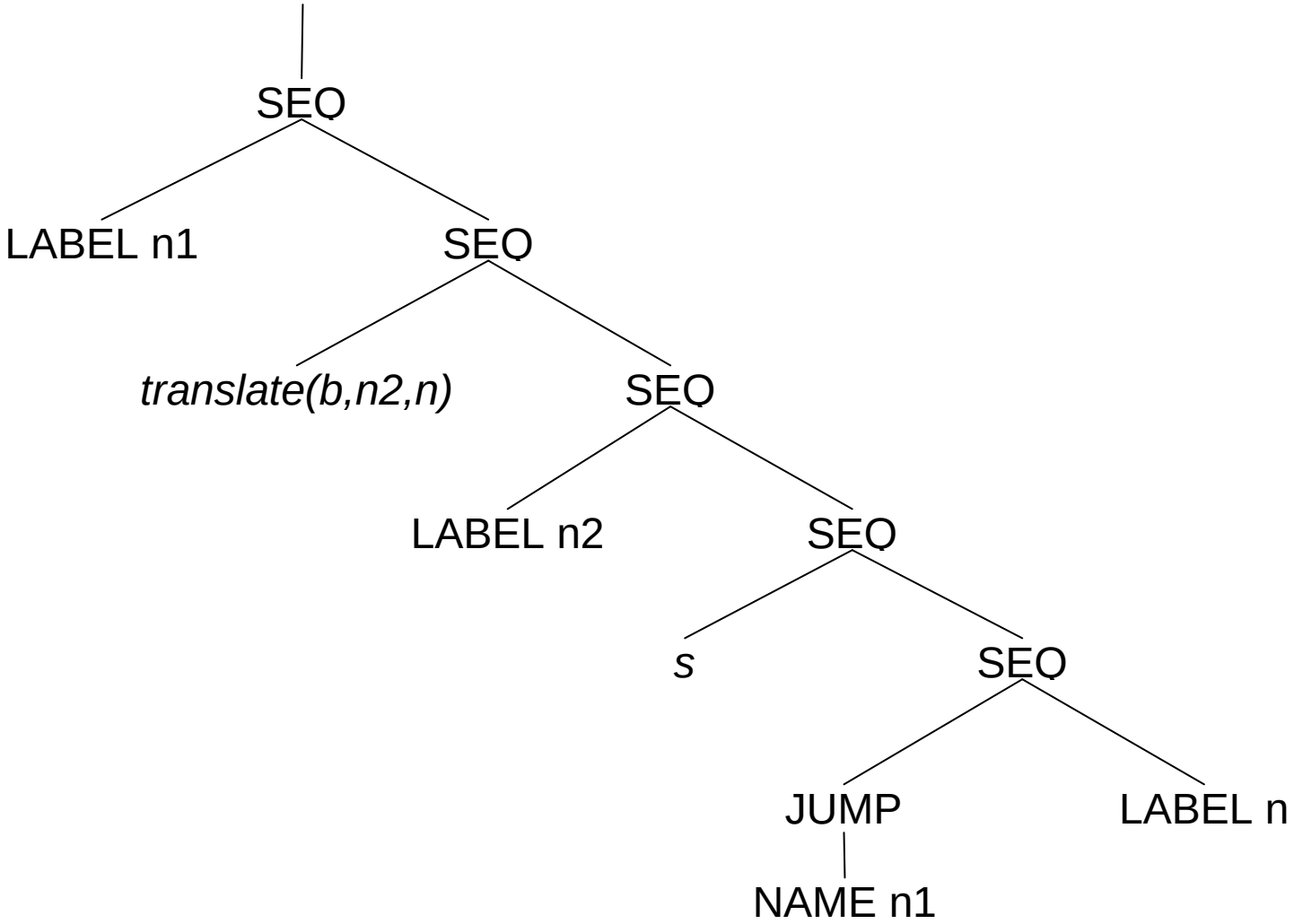
Translating conditionals (contd.)

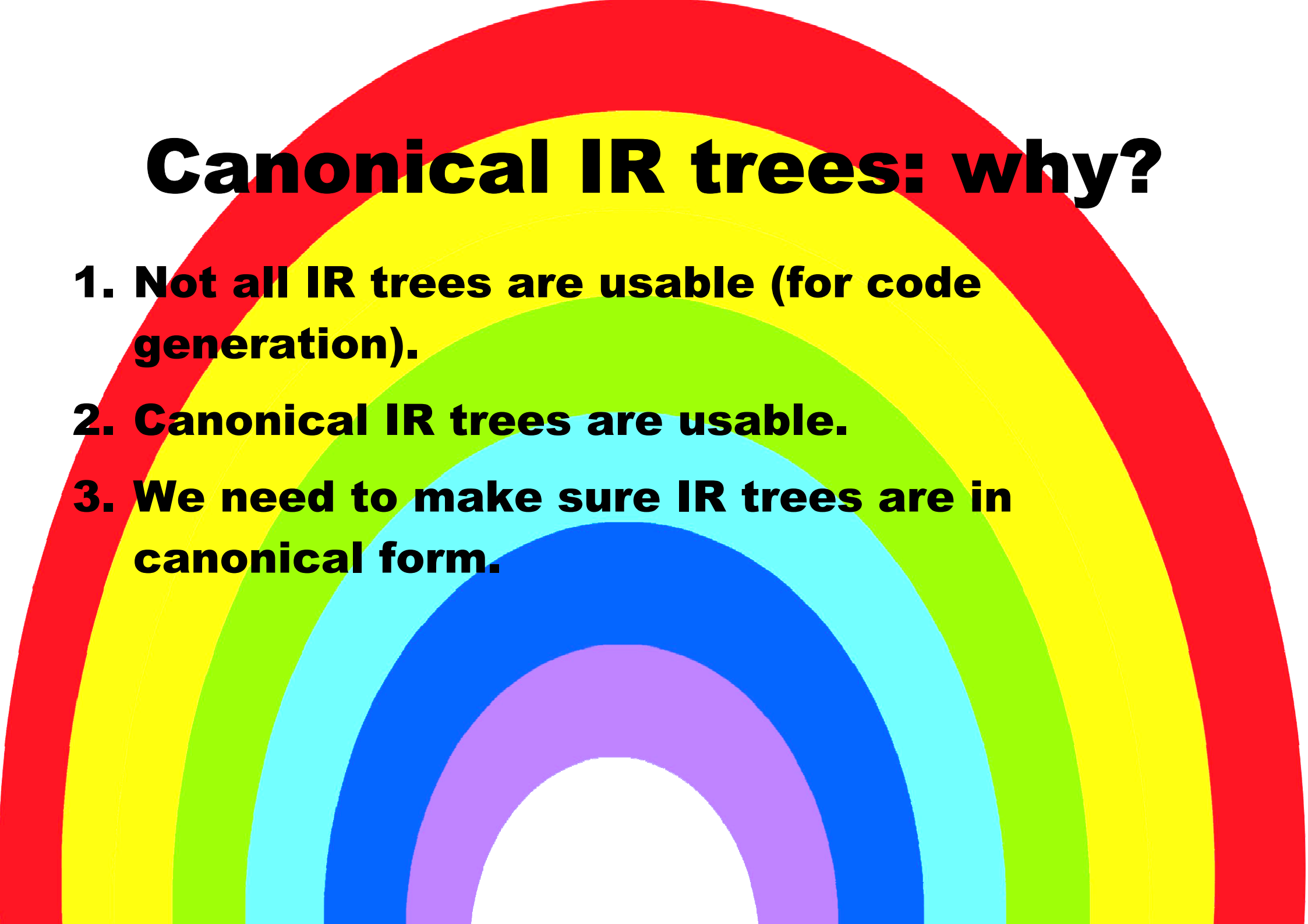
`if (b) then s`:



Translating while loops

```
while (b) s:
```





Canonical IR trees: why?

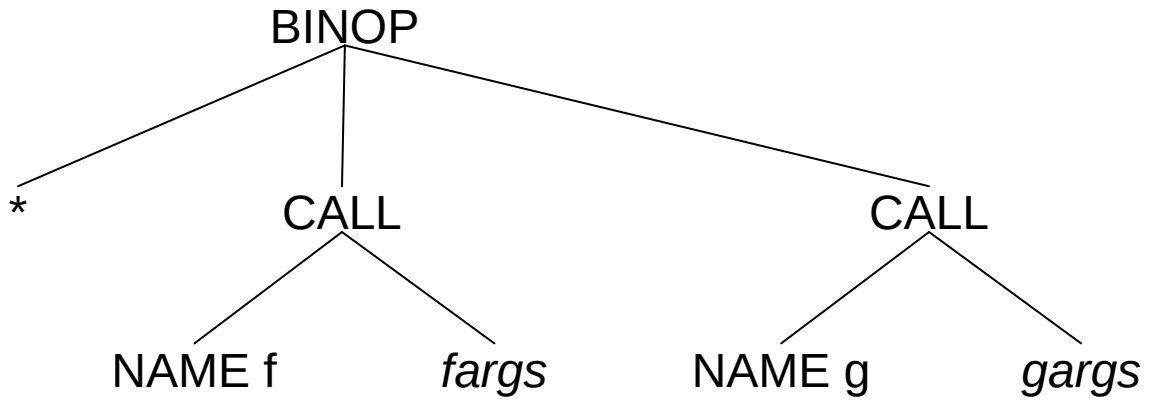
1. Not all IR trees are usable (for code generation).
2. Canonical IR trees are usable.
3. We need to make sure IR trees are in canonical form.

Canonical IR trees

IR trees produced may not be easy to implement:

- ESEQ and CALL nodes in expressions make order significant

- CALL nodes in expressions produce value in *same* register



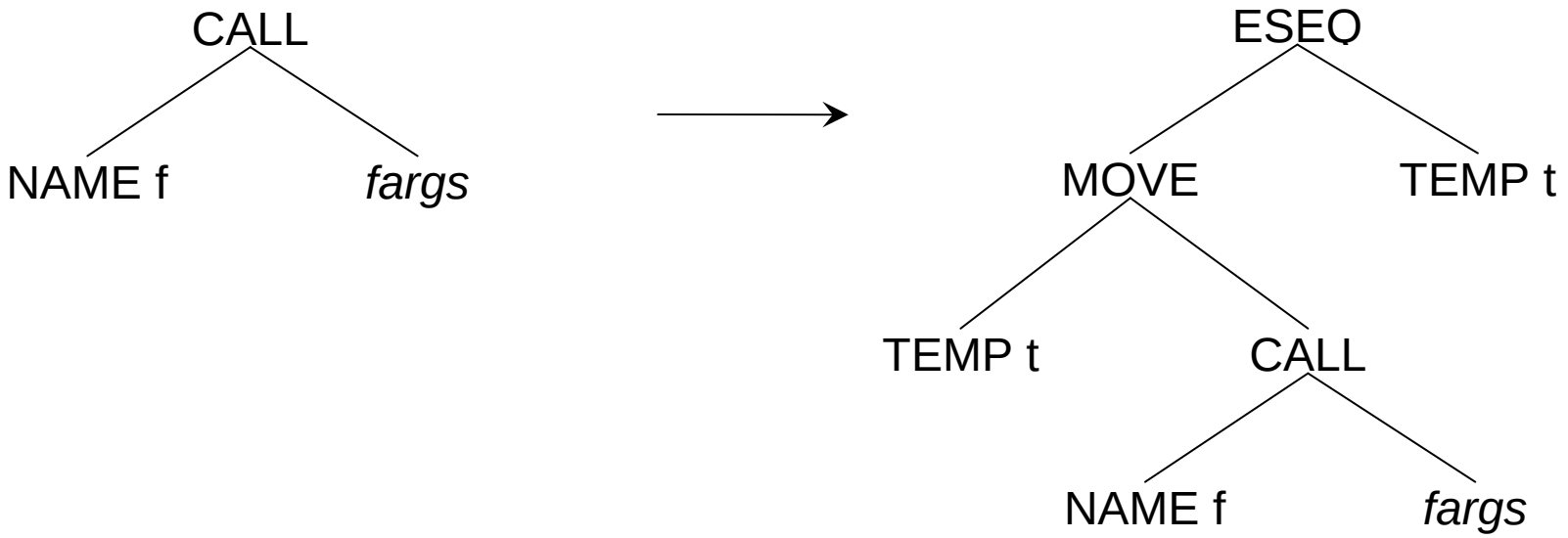
Canonical IR trees:

- CALL nodes occur only as right subtree of MOVE or EXP nodes
- ESEQ nodes do not occur

Canonicalizing IR trees: move CALL nodes

Allow CALL nodes to appear only on right of MOVE.

- if CALL node is not already right subtree of MOVE, replace it:



Canonicalizing IR trees: replace ESEQ by SEQ

Replace ESEQ in expressions by SEQ in statements.

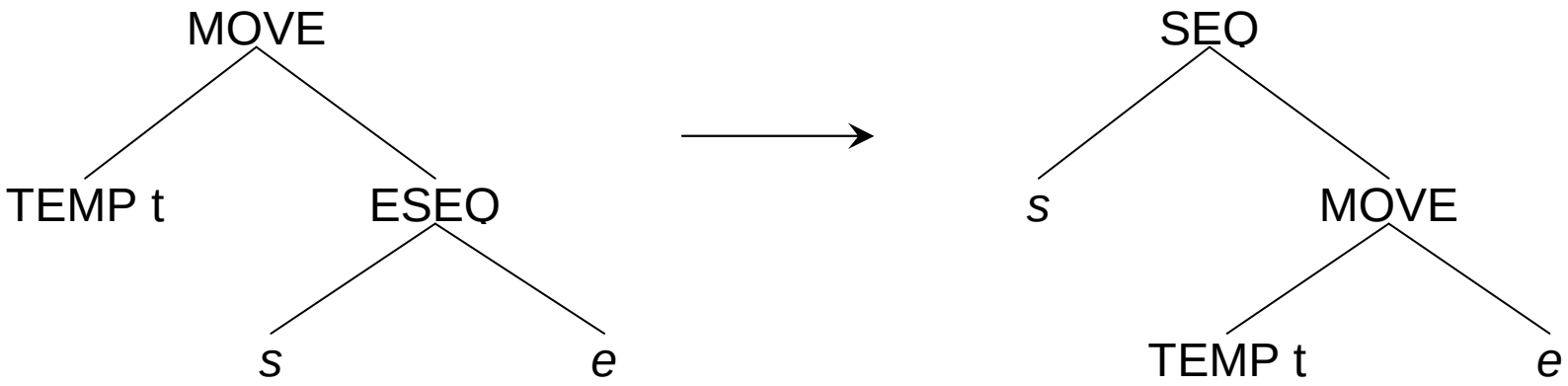
Final IR tree has only SEQ nodes, at top level:

statements contain expression subtrees but ***not*** vice versa.

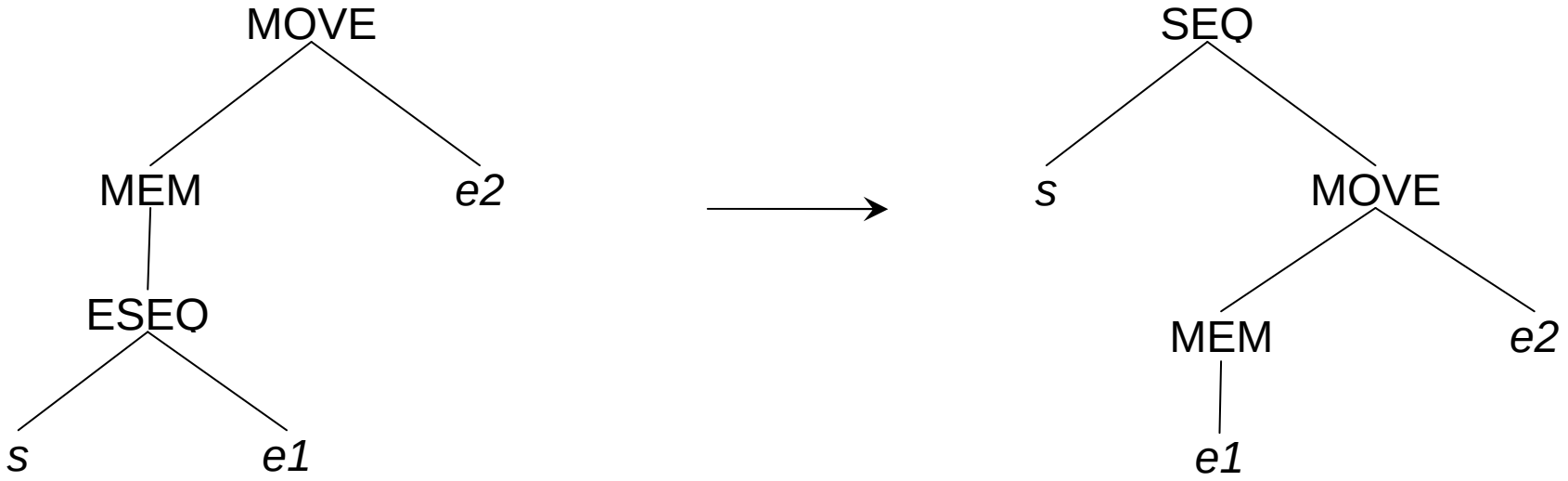
One rewrite rule for each place where an ESEQ node can appear.

E.g., ESEQ as subtree of MOVE:

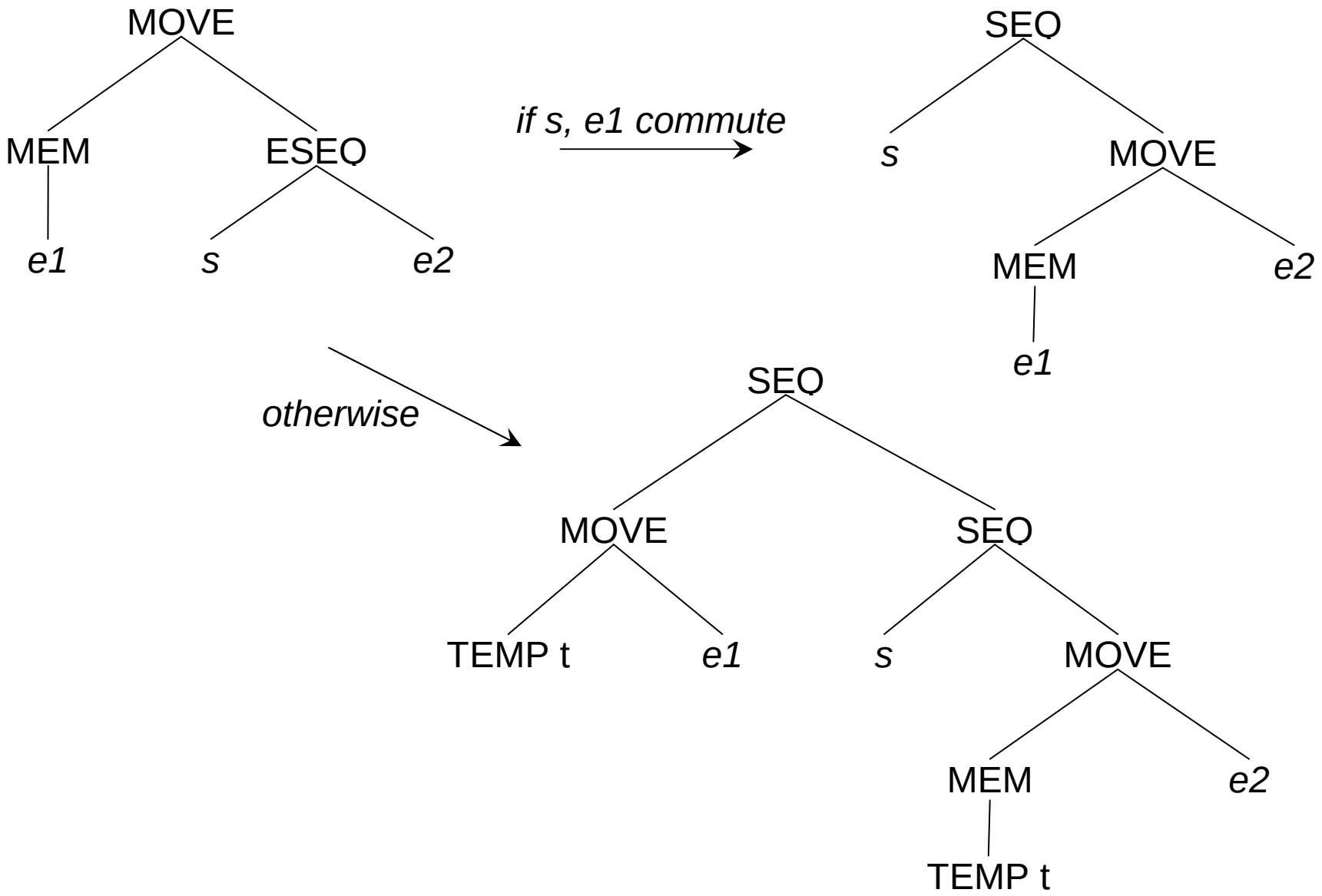
- Rule 1:


MOVE
TEMP t
ESEQ
s
e
SEQ
s
MOVE
TEMP t
e

- Rule 2:


MOVE
MEM
e2
ESEQ
s
e1
SEQ
s
MOVE
MEM
e2
e1

- Rule 3:

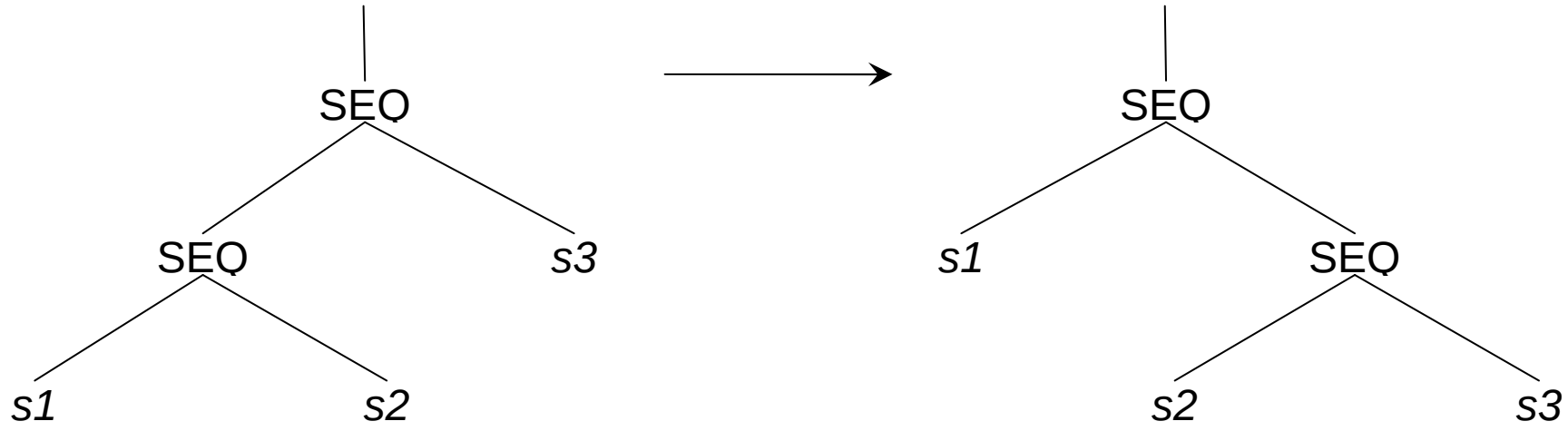

MOVE
MEM
ESEQ
e1
s
e2
if s, e1 commute
SEQ
s
MOVE
MEM
e2
e1
otherwise
SEQ
MOVE
SEQ
TEMP t
e1
s
MOVE
MEM
e2
TEMP t

s and *e* commute if *s* does not assign to variables used by *e*.

Linearizing IR trees

SEQ should not have SEQ as left subtree.

Repeatedly apply rule:



Removing redundant jumps

Removing jumps to next statement: