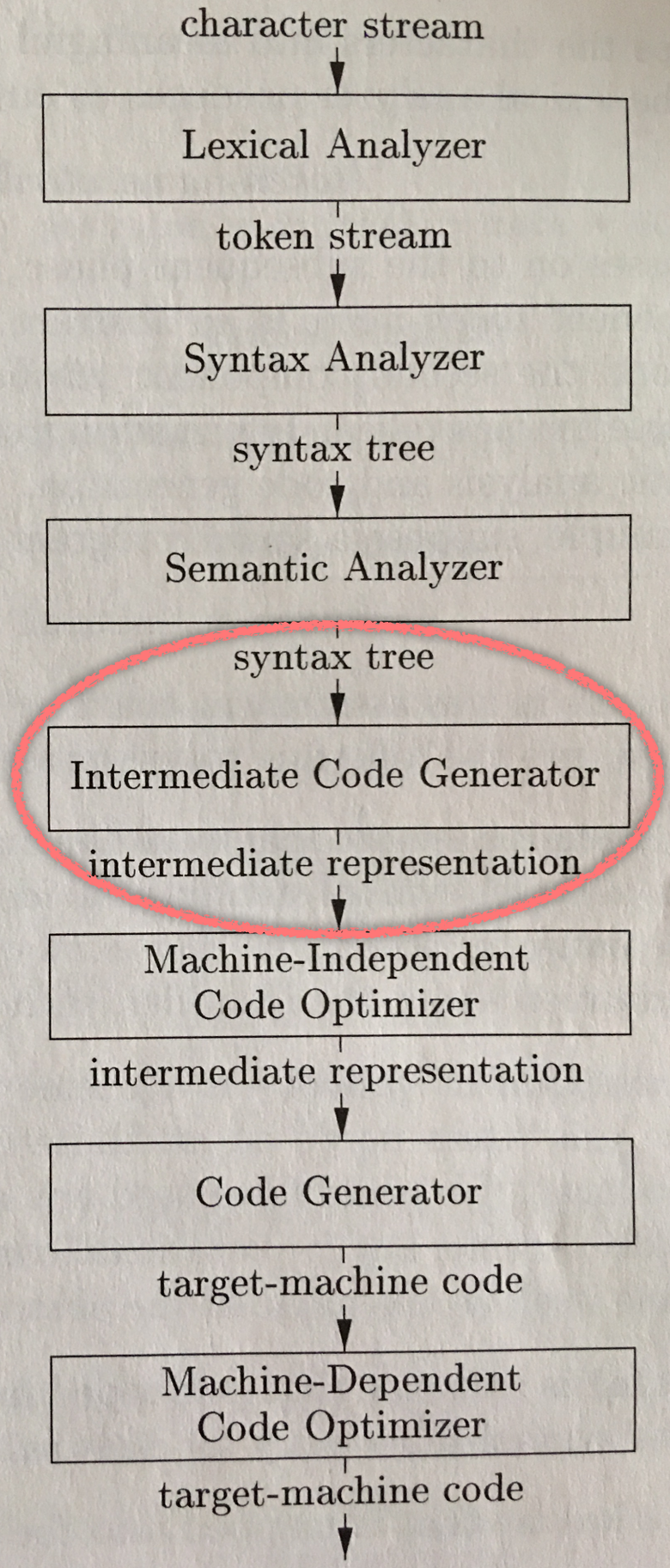
CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text



backpatching while

6.7.3 Backpatching Flow-of-Control statements

The end-of-rule actions for a **while statement** are shown on the next slide.

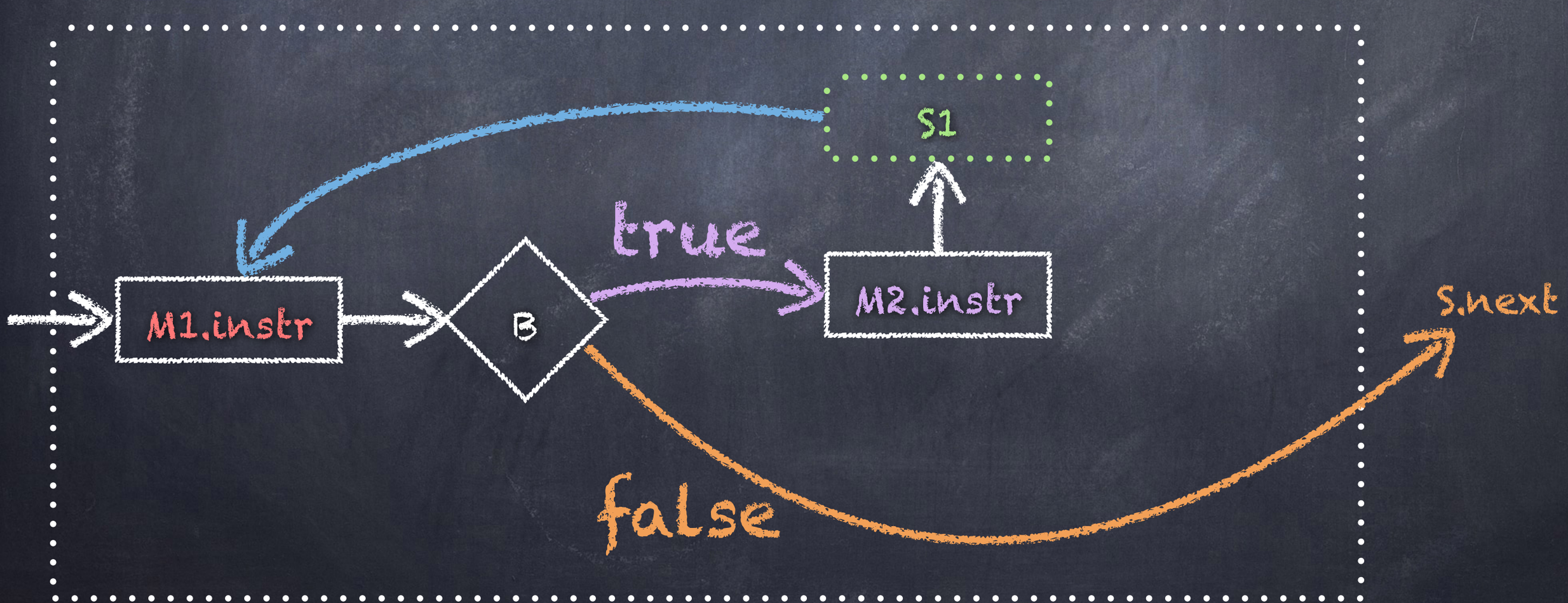
Exercise:
Extend example 6.24 as a while statement where the body of the while requires 5 instructions.

```
while (x < 100 || x > 200 && x != y) S1
```

Show how backpatching works in the instruction array.

6.7.3 Backpatching Flow-of-Control statements

S -> while M1 (B) M2 S1	backpatch(S1.nextlist, M1.instr) backpatch(B.truelist, M2.instr) S.nextlist = B.falselist gen('goto' M1.instr)
M -> ε	M.instr = nextinstr



6.7.3 Backpatching Flow-of-Control statements

Here's what I came up with...

Example 6.24 - extended

while (x < 100 || x > 200 && x != y) S1

```
100: if x < 100 goto 106
101: goto 102
102: if x > 200 goto 104
103: goto ____
104: if x != y goto 106
105: goto ____
106: instruction for S1
107: instruction for S1
108: instruction for S1
109: instruction for S1
110: instruction for S1
111: goto 100
```

B.truelist = {100,104}

S.nextlist = B.falselist = {103,105}

Notice that we backpatch only those instructions whose targets are within the (while) instruction's code block.

backpatching for

6.7.3 Backpatching Flow-of-Control statements

Exercise: show how to translate a generic for statement

`for ( S1 ; B ; S2 ) S3`

and give the translation of this one in particular:

`for ( S1 ; x < 100 || x > 200 && x != y ; S2 ) S3`

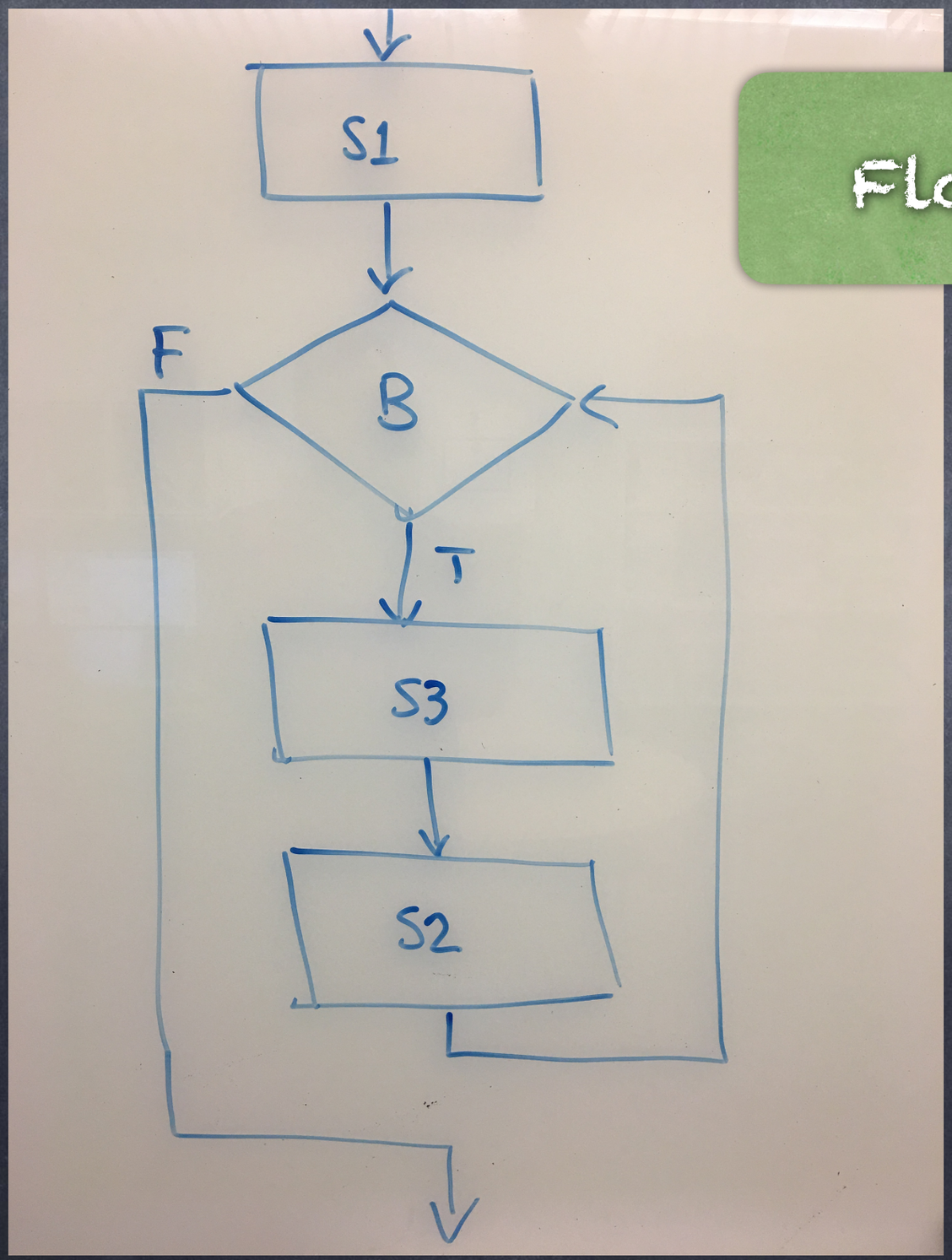
Exercise: show how to translate
for (S1 ; B ; S2) S3

First show how to translate a generic for statement

for (S1 ; B ; S2) S3

Exercise: show how to translate

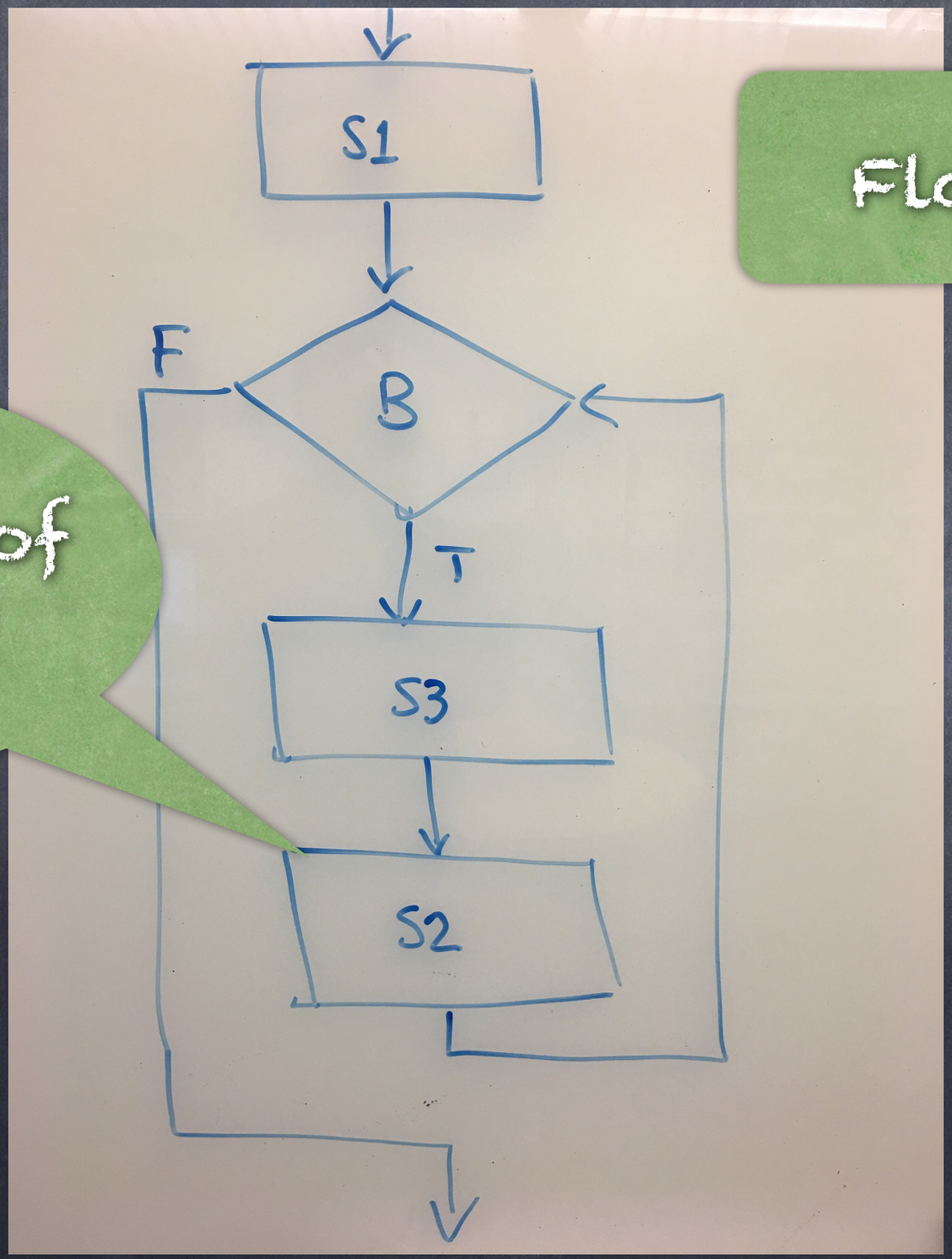
for (S1 ; B ; S2) S3



Flowchart

Exercise: show how to translate

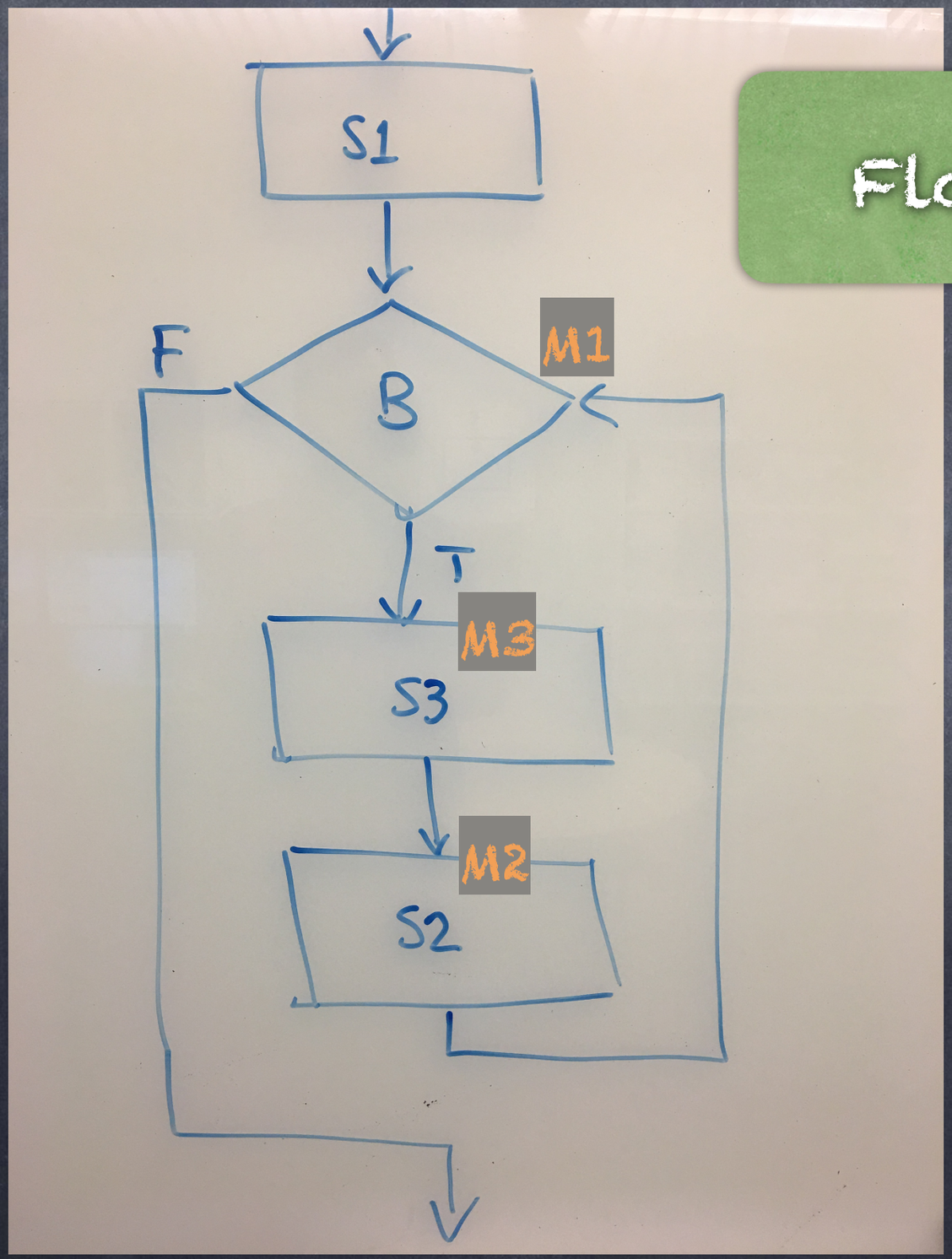
for (S1 ; B ; S2) S3



Flowchart

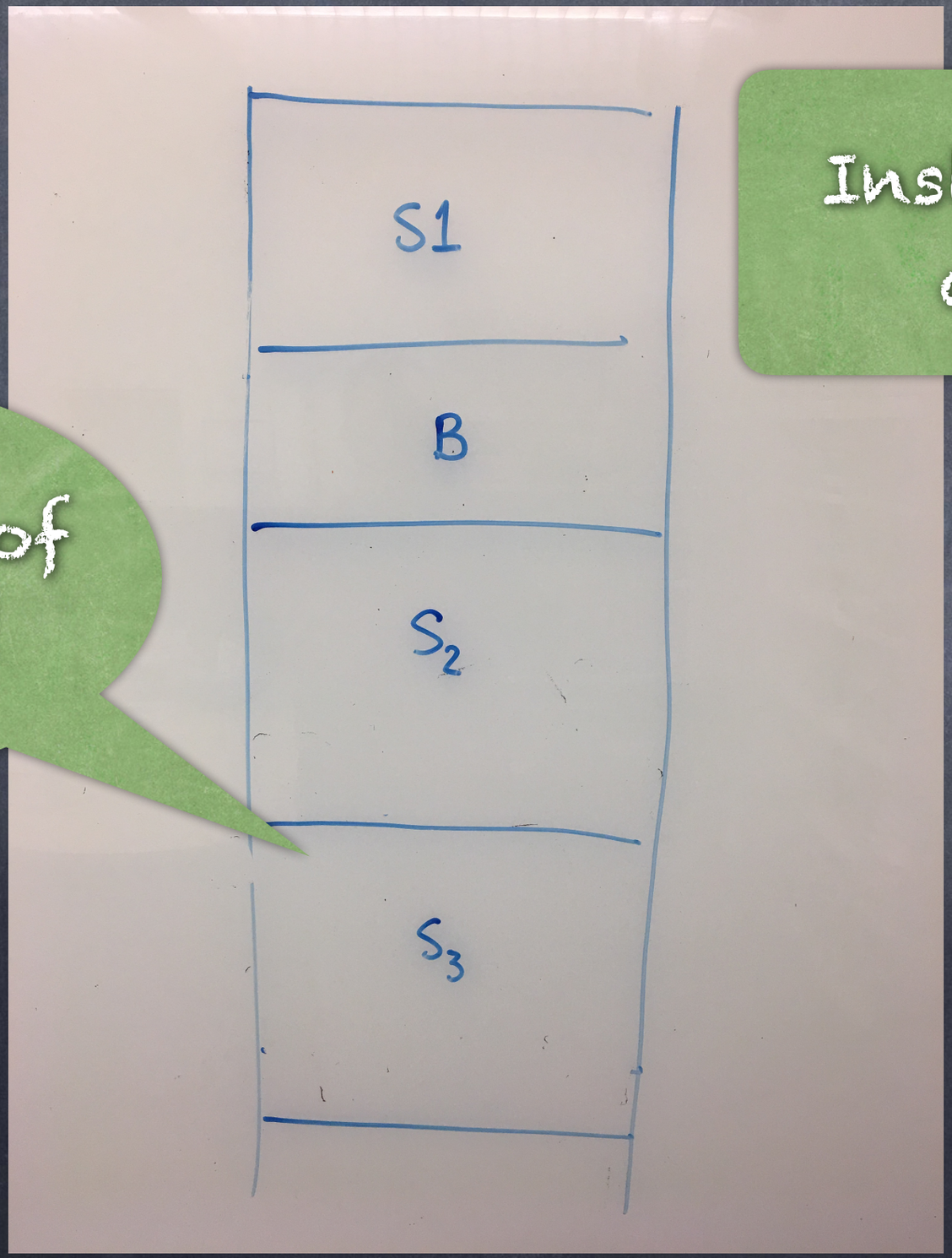
Note order of S3 and S2!

for (S1 ; M1 B ; M2 S2) M3 S3



Flowchart

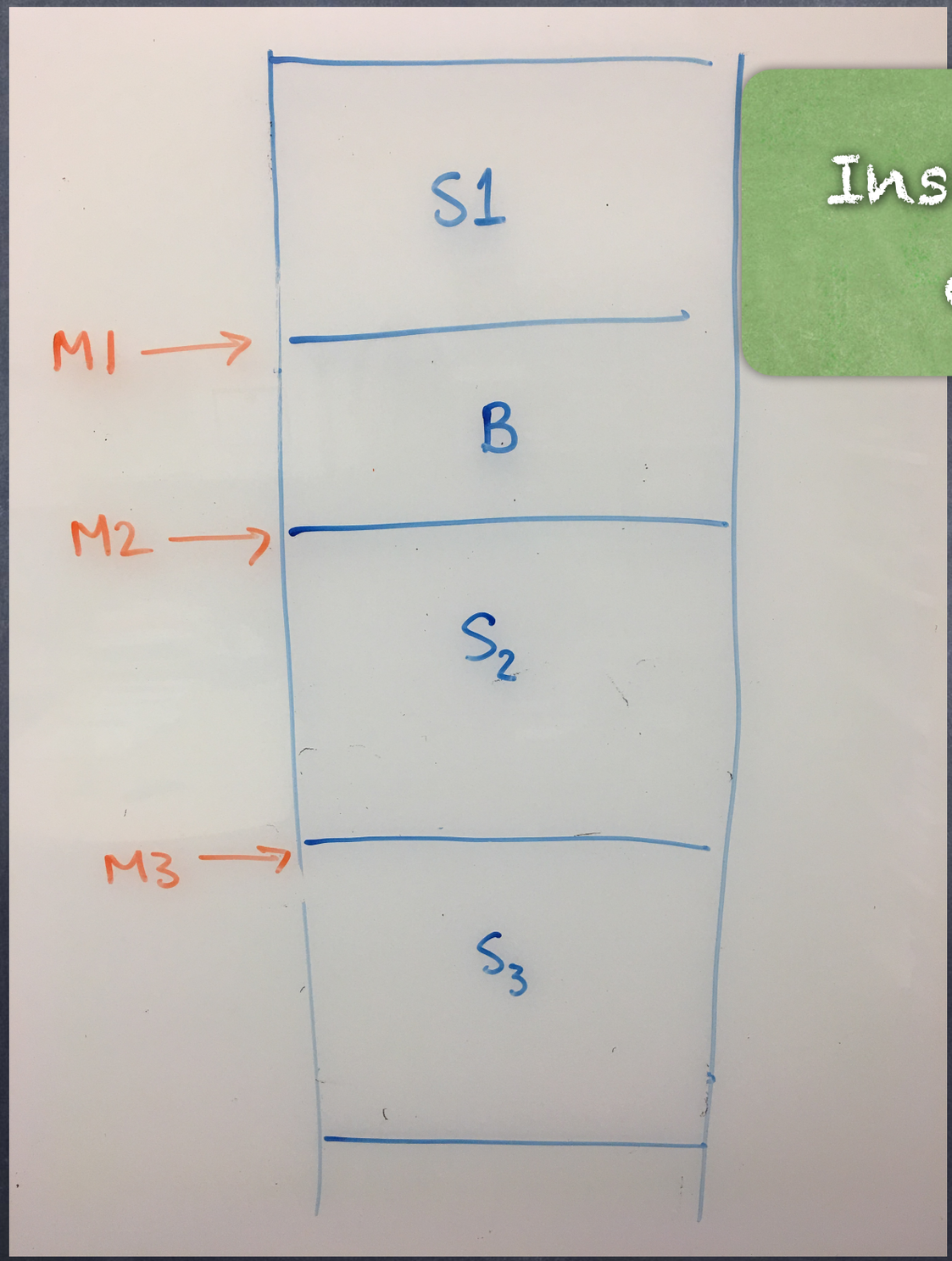
How is code generated?



Instruction array

Note order of S3 and S2!

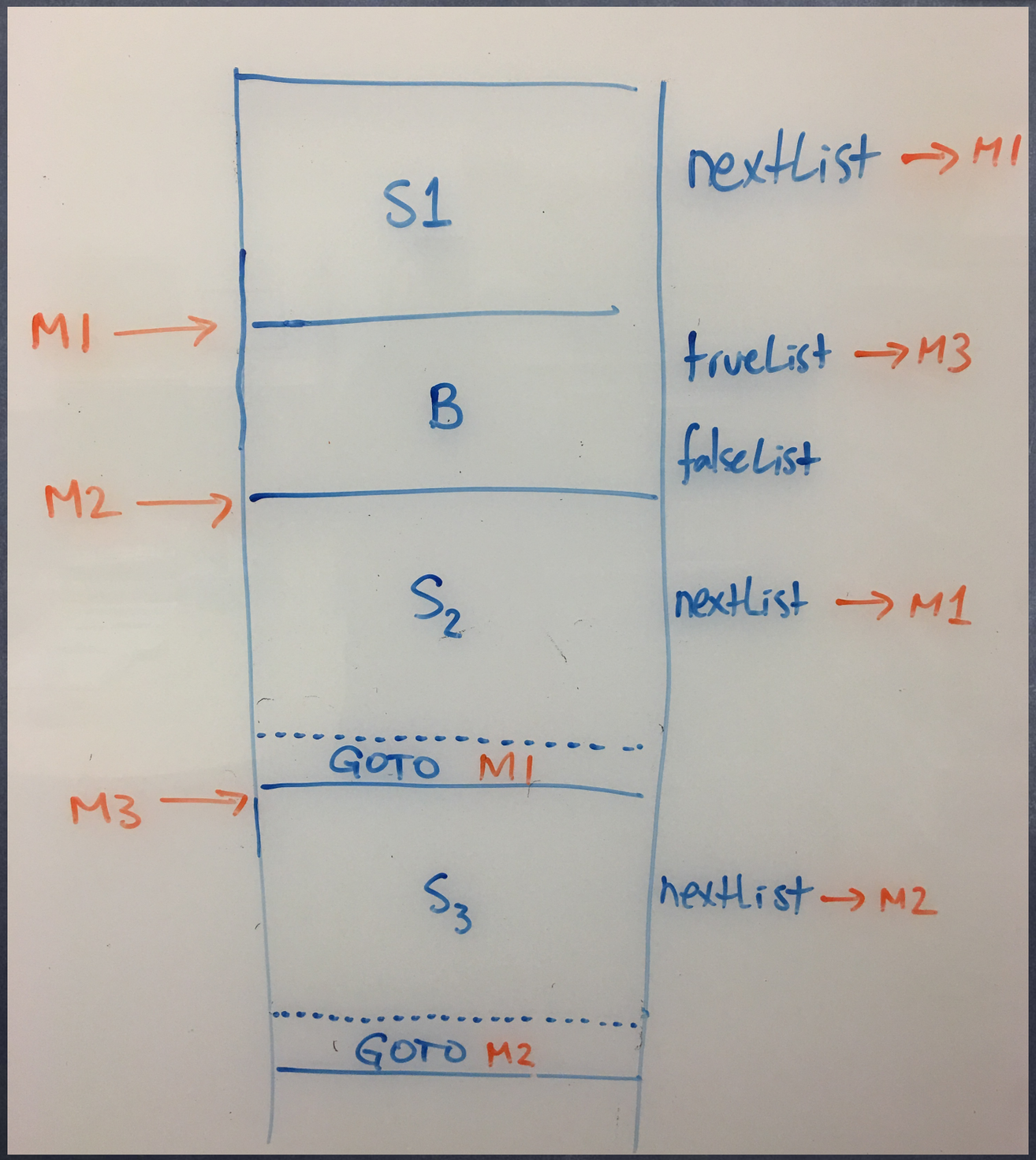
Where are jump targets?



Instruction array

Jumps

Instruction array



Jumps

Instruction array

for (S1 ; M_1 B ; M_2 S2) M_3 S3

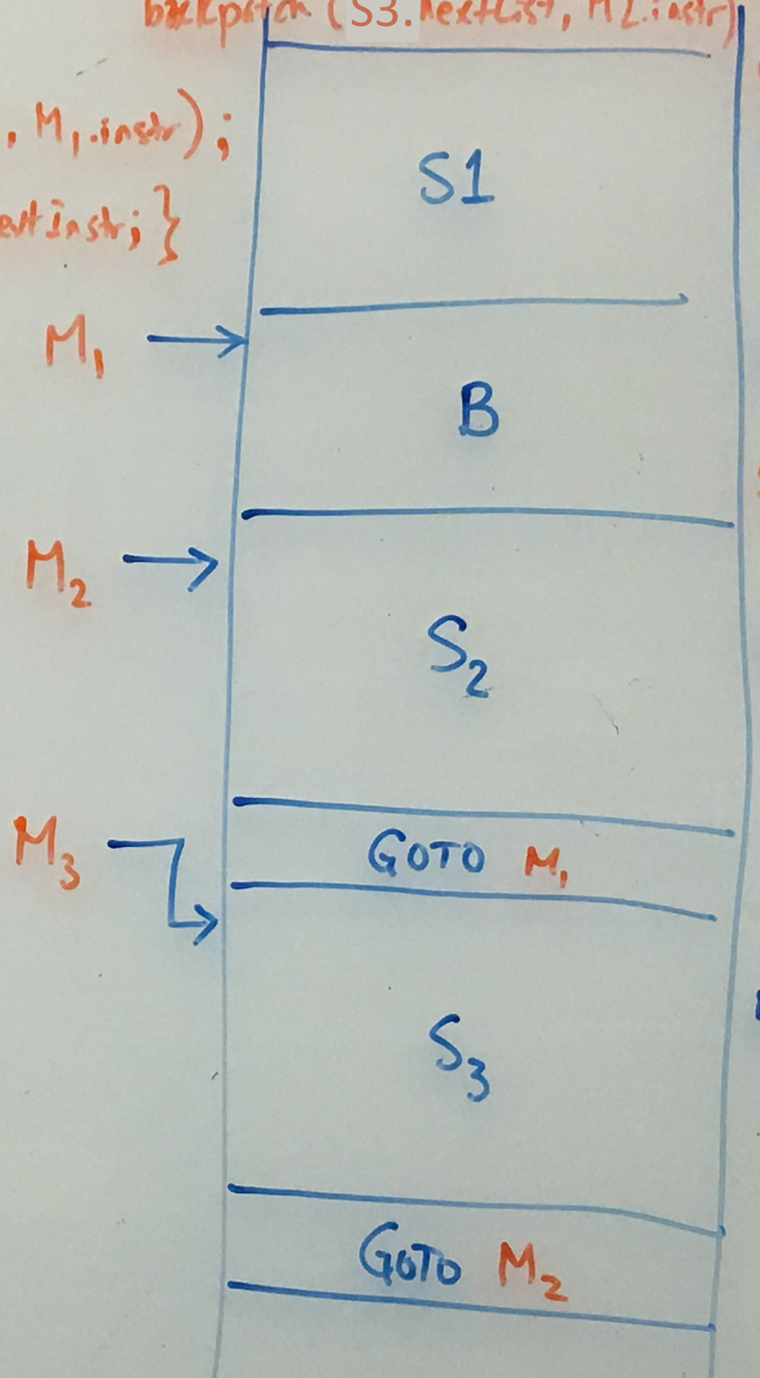
{ M_1.instr = nextInstr; }

{ M_2.instr = nextInstr; }

{ gen('Goto', M_1.instr); M_3.instr = nextInstr; }

Quartet

{ t = mergeList(S1.nextList, S2.nextList)
backpatch(t, M1.instr) backpatch(B.trueList, M_3.instr)
backpatch(S3.nextList, M2.instr) S.nextList = B.falseList
gen(Goto, M_2.instr)



nextList → M_1 ✓

trueList → M_3 ✓

S.nextList = B.falseList

nextList → M_1 ✓

nextList → M_2 ✓

Jumps

S -> for (S1 ; M1 B ; M2 S2) M3 S3	t = mergeList(S1.nextList, S2.nextList) backpatch(t, M1.instr) backpatch(B.trueList, M3.instr) backpatch(S3.nextList, M2.instr) S.nextList = B.falseList gen('goto', M2.instr)
M1 -> ε	M1.instr = nextinstr
M2 -> ε	M2.instr = nextinstr
M3 -> ε	gen('goto', M1.instr) M3.instr = nextinstr

Exercise: show how to translate
for (S1 ; B ; S2) S3

Second give the translation of this one in particular:

for (S1 ; x < 100 || x > 200 && x != y ; S2) S3

Example 6.24 - extended

for (S1; x < 100 || x > 200 && x != y; S2) S3

```
097: ...S1 instruction...
098: ...S1 instruction...
099: ...S1 instruction...
100: if x < 100 goto 109
101: goto 102
102: if x > 200 goto 104
103: goto ____
104: if x != y goto 109
105: goto ____
106: ...S2 instruction...
107: ...S2 instruction...
108: goto 100
109: ...S3 instruction...
110: ...S3 instruction...
111: ...S3 instruction...
112: goto 106
```

B.truelist = {100,104}

S.nextlist = B.falselist = {103,105}

Notice that we backpatch only those instructions whose targets are within the (for) instruction's code block.

backpatching switch

6.7.3 Backpatching Flow-of-Control statements

Exercise: show how to translate a generic switch statement:

```
switch (E) {
        case C1 : sblock1
        case C2 : sblock2
        ...
        case Cn-1 : sblockn-1
        otherwise : sblockn
}
```

6.7.3 Backpatching Flow-of-Control statements

Exercise: show how to translate a generic **switch** statement:

```
switch (E) {
        case C1 : sblock1
        case C2 : sblock2
        ...
        case Cn-1 : sblockn-1
        otherwise : sblockn
}
```

See also the discussion on pages 420 and 421 in the textbook.

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code
	goto test

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code
	goto test
L_1	$sblock_1$.code
	goto next

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code
	goto test
L_1	$sblock_1$.code
	goto next
L_2	$sblock_2$.code
	goto next

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code
	goto test
L_1	$sblock_1$.code
	goto next
L_2	$sblock_2$.code
	goto next
	...
L_{n-1}	$sblock_{n-1}$.code
	goto next
L_n	$sblock_n$.code
	goto next

Switch (6.8.2)
[read p.420-421]

```
switch (E) {
    case C1 : sblock1
    case C2 : sblock2
    ...
    case Cn-1 : sblockn-1
    otherwise : sblockn
}
```

start	E.code
	goto test
L_1	$sblock_1$.code
	goto next
L_2	$sblock_2$.code
	goto next
	...
L_{n-1}	$sblock_{n-1}$.code
	goto next
L_n	$sblock_n$.code
	goto next
test	if $t=C_1$ goto L_1
	if $t=C_2$ goto L_2
	...
	if $t=C_{n-1}$ goto L_{n-1}
	goto L_n
next	