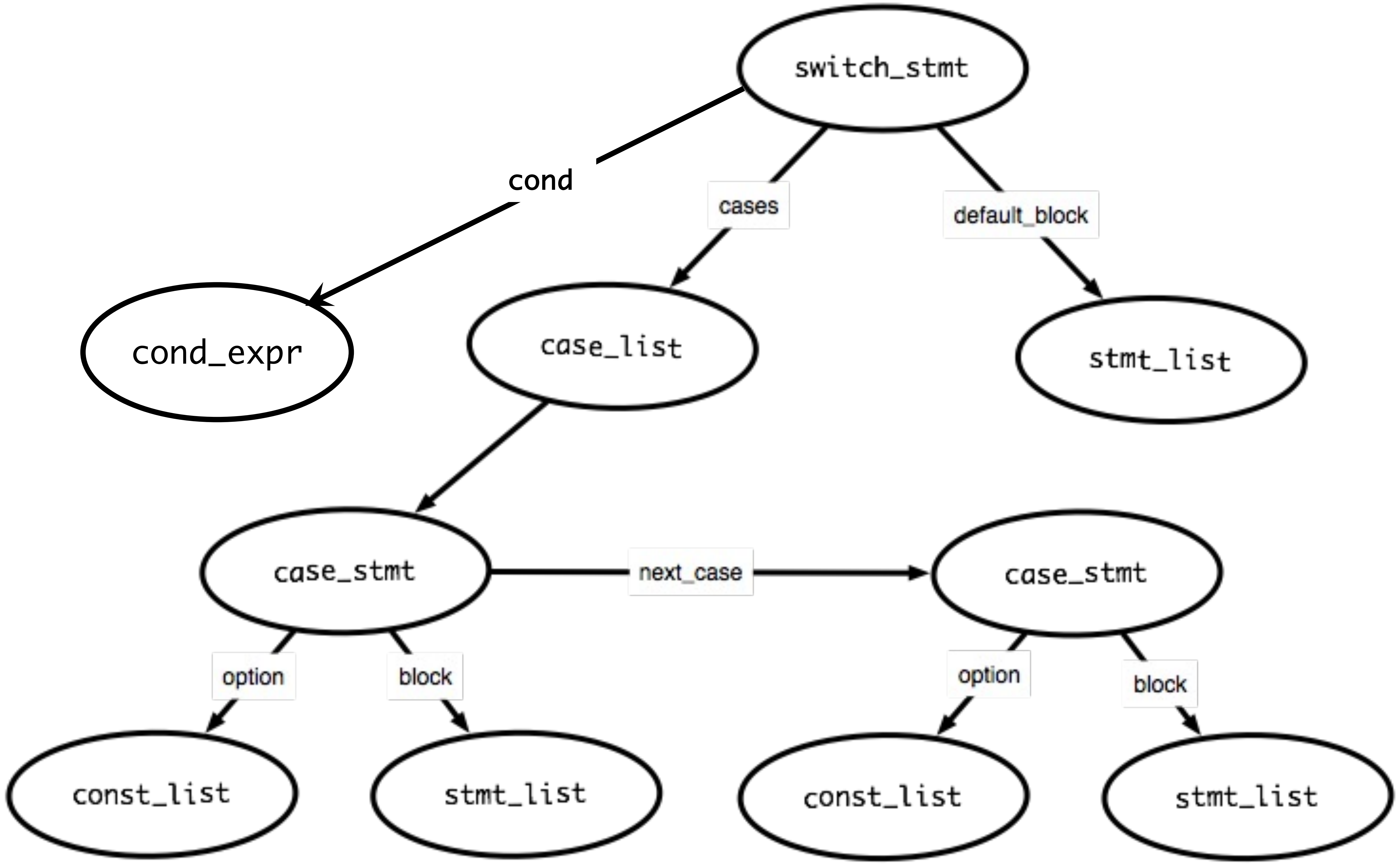
Switch Statements

Switch statements

```
switch (<expr>)
 case <const_list>: <stmt_list>
 case <const_list>: <stmt_list>
 ...
 default: <stmt_list>
end
```



Switch statements

- Generated code for <expr> then check all the cases to see which matches the result
- Key issues:
  - Where to jump?
  - Multiple cases lead to the same code
  - Many different cases --- potentially dozens or hundreds

```
switch (<expr>)
 case <const_list>: <stmt_list>
 case <const_list>: <stmt_list>
 ...
 default: <stmt_list>
end
```

jump tables

- Problem: do not know *which label* to jump to until switch expression is evaluated
- Use a **jump table**: an array indexed by case values, contains address to jump to
    - If table is not full (i.e., some possible values are skipped), can point to a default clause
        - If default clause does not exist, this can point to error code
    - Problems
        - If table is sparse, wastes a lot of space
        - If many choices, table will be very large

```
switch (<expr>)
 case <const_list>: <stmt_list>
 case <const_list>: <stmt_list>
 ...
 default: <stmt_list>
end
```

Jump table example

Consider the code:
((xxxx) is address of code)

```
Case x is
(0010) When 0: stmts
(0017) When 1: stmts
(0192) When 2: stmts
(0198) When 3 stmts;
(1000) When 5 stmts;
(1050) Else stmts;
```

Jump table has 6 entries:

0	JUMP 0010
1	JUMP 0017
2	JUMP 0192
3	JUMP 0198
4	JUMP 1050
5	JUMP 1000

Table only has one Unnecessary row (for choice 4)

Jump table example

Consider the code:
((xxxx) Is address of code)

```
Case x is
(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3 stmts3
(1000) When 987 stmts4
(1050) When others stmts5
```

Jump table has 988 entries:

0	JUMP 0010
1	JUMP 0017
2	JUMP 0192
3	JUMP 0198
4	JUMP 1050
. . .	JUMP 1050
986	JUMP 1050
987	JUMP 1000

Table has 983 unnecessary rows. Doesn't appear to be the right thing to do! ***NOTE: table size is proportional to range of choice clauses, not number of clauses!***

Do a binary search

Jump table has 5 entries:

Consider the code:
((xxxx) Is address of code)

```
Case x is
(0010) When 0: stmts0
(0017) When 1: stmts1
(0192) When 2: stmts2
(0198) When 3 stmts3
(1000) When 987 stmts4
(1050) When others stmts5
```

0	JUMP 0010
1	JUMP 0017
2	JUMP 0192
3	JUMP 0198
987	JUMP 1000

Perform a binary search on the table. If the entry is found, then jump to that offset. If the entry isn't found, jump to others clause. *O(log n)* time, n is the size of the table, for each jump.

Linear search example

Consider the code:
((xxxx) Is address of code)

```
Case x is
(0010) When 0: stmts1
(0017) When 1: stmts2
(0192) When 2: stmts3
(1050) When others stmts4
```

If there are a small number of choices, then do an in-line linear search. A straightforward way to do this is generate code analogous to an IF THEN ELSE.

```
If (x == 0) then stmts1;
Elseif (x = 1) then stmts2;
Elseif (x = 2) then stmts3;
Else stmts4;
```

O(n) time, n is the size of the table, for each jump.

Dealing with jump tables

```
switch (<expr>)
 case <const_list>: <stmt_list>
 case <const_list>: <stmt_list>
 ...
 default: <stmt_list>
end
```

```
 <expr>
 <code for jump table>
LABEL0:
 <stmt_list>
LABEL1:
 <stmt_list>
...
DEFAULT:
 <stmt_list>
OUT:
```

- Generate labels, code, then build jump table
  - Put jump table after generated code
- Why do we need the OUT label?
  - In case of `break` statements