The recent switch lowering improvements

Hans Wennborg
hwennborg@google.com

A Switch

C:

```
switch (x) {
case 0:
  // foo
case 1:
  // bar
...
default:
  // baz
}
```

LLVM IR:

```
switch i32 %x, label %baz [
    i32 0, label %foo
    i32 1, label %bar
    ...
  ]
```

A Switch

C:

```
if (x == 0) {
  // foo
} else if (x == 1) {
  // bar
} else {
  // baz
}
```

LLVM IR:

```
switch i32 %x, label %baz [
    i32 0, label %foo
    i32 1, label %bar
    ...
  ]
```

Lowering

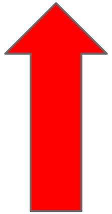
LowerSwitch

SelectionDAGBuilder::visitSwitch

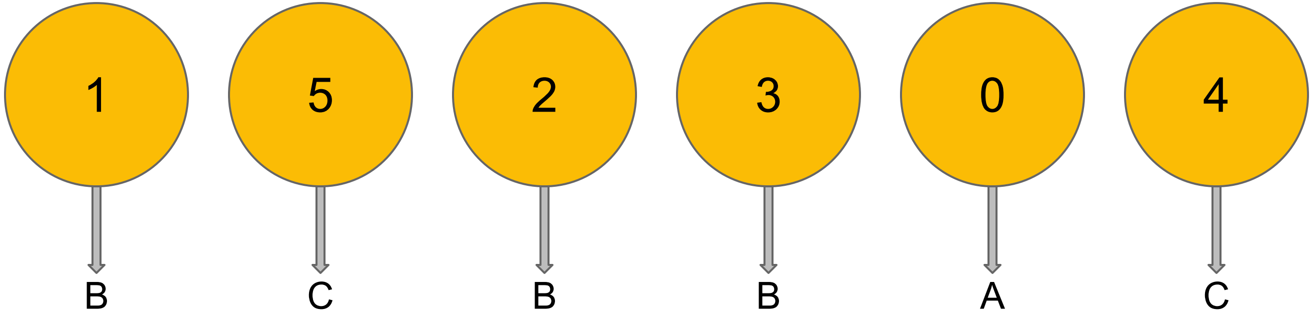
Lowering

LowerSwitch

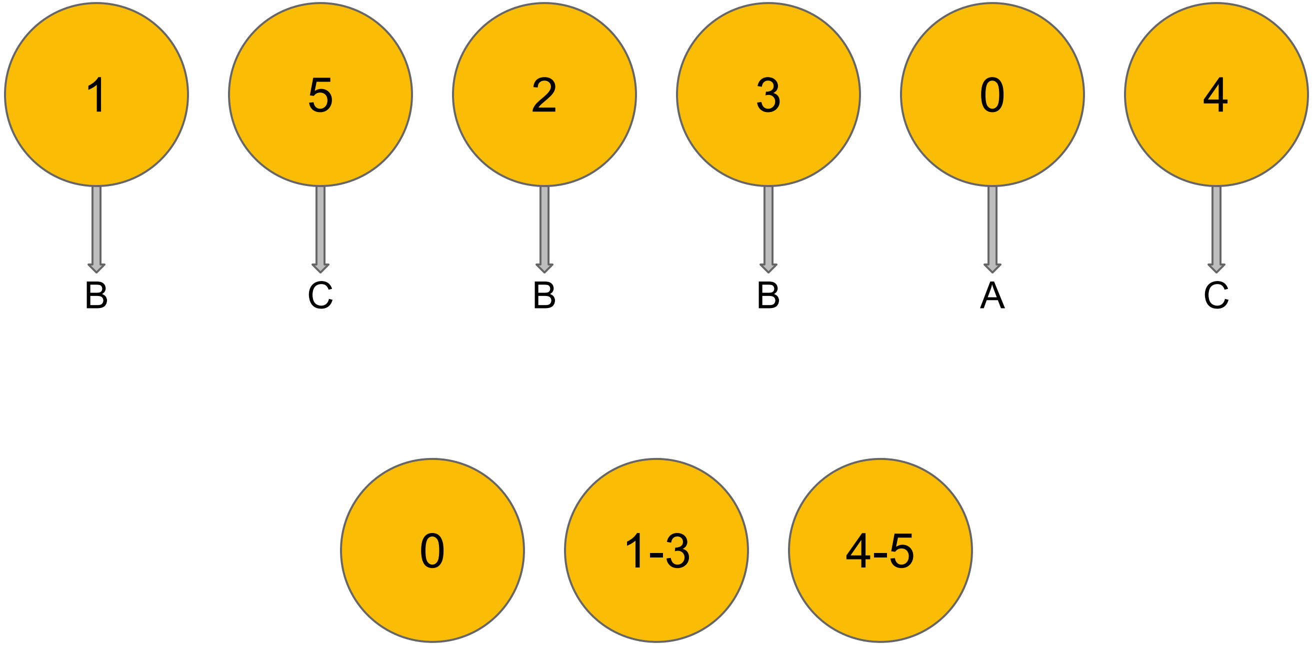
SelectionDAGBuilder::visitSwitch



Step 0: Cluster adjacent cases



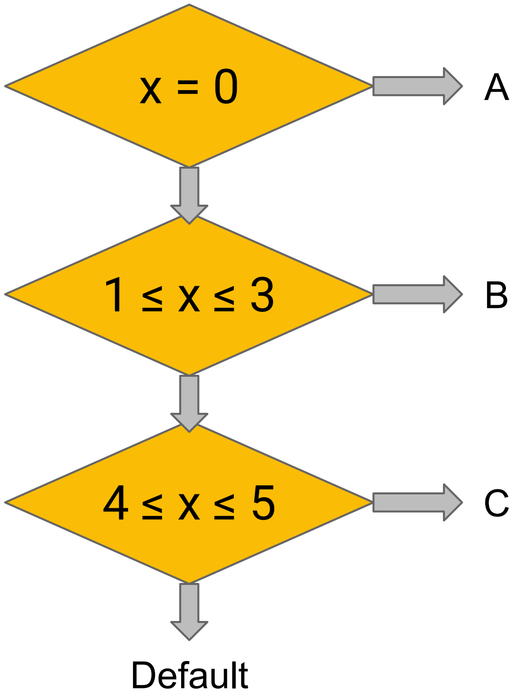
Step 0: Cluster adjacent cases



Lowering strategies

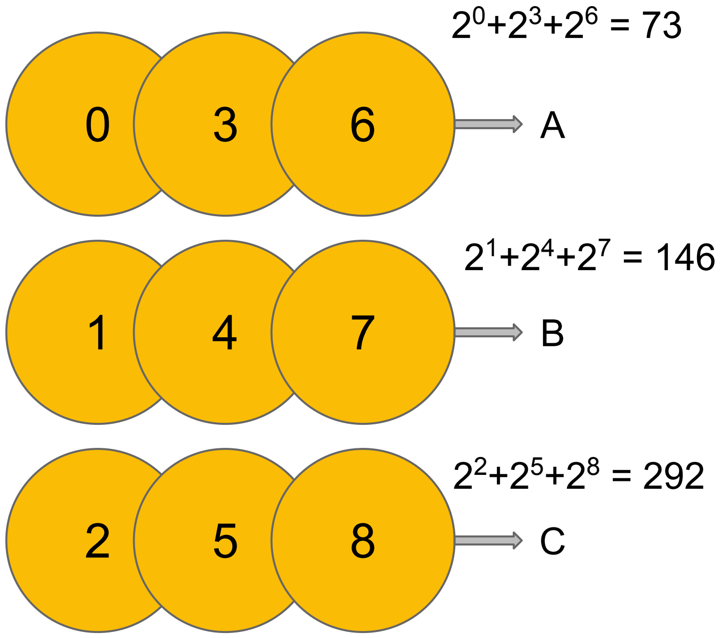
1. Straight comparisons
2. Jump tables
3. Bit tests
4. Binary search tree

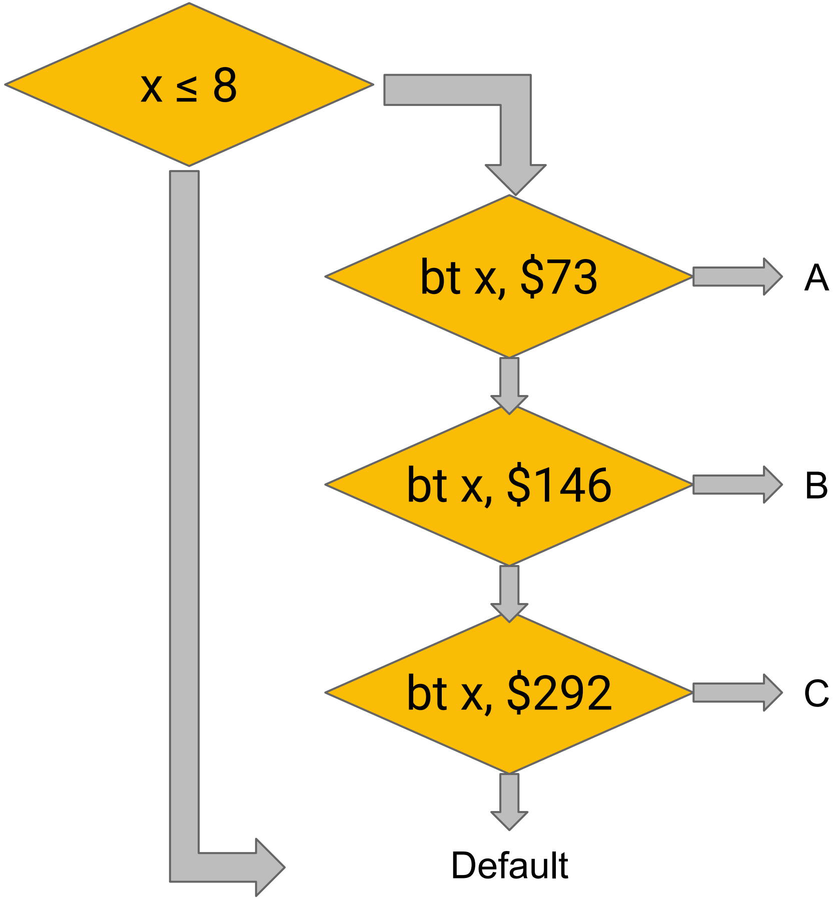
1. Straight comparisons



- Number of clusters ≤ 3

2. Bit tests





- Number of destinations ≤ 3
- Range fits in machine word

3. Jump table

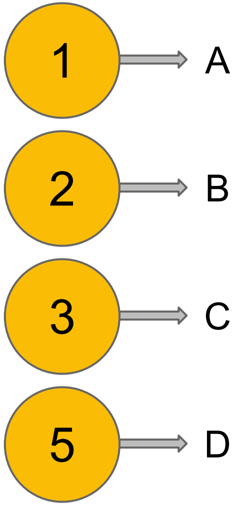
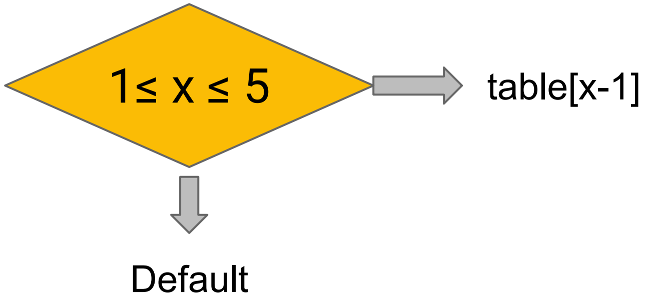


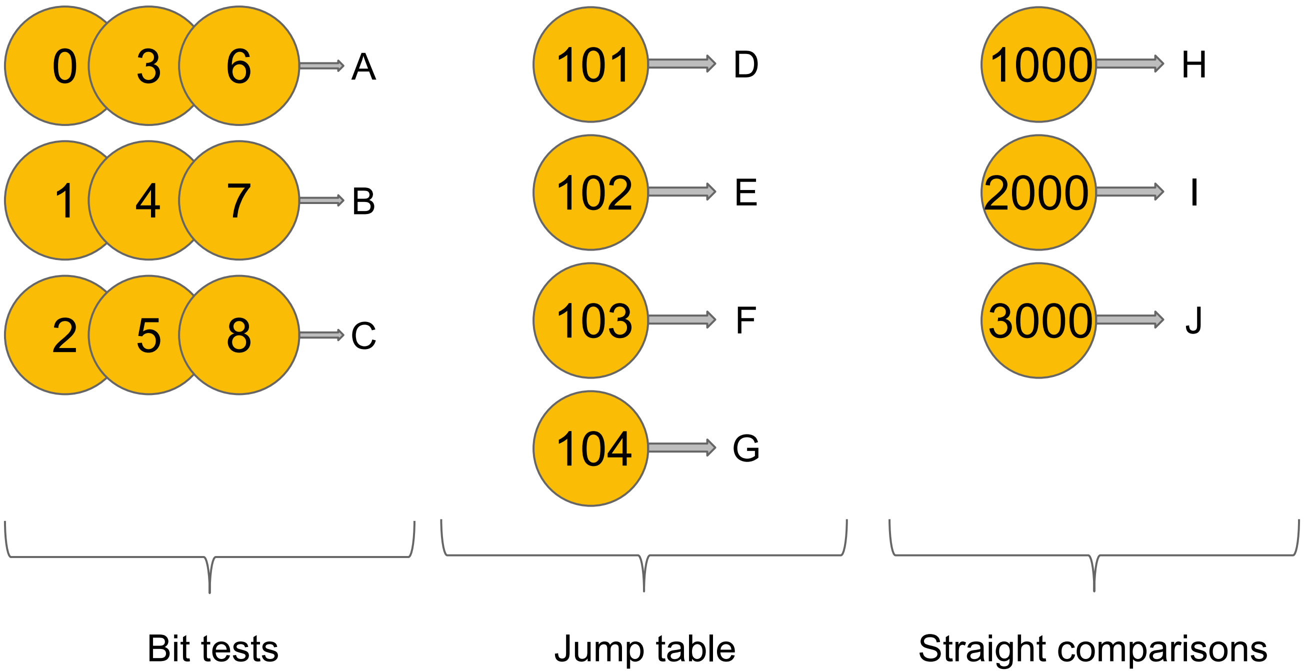
table:

0	A
1	B
2	C
3	Default
4	D

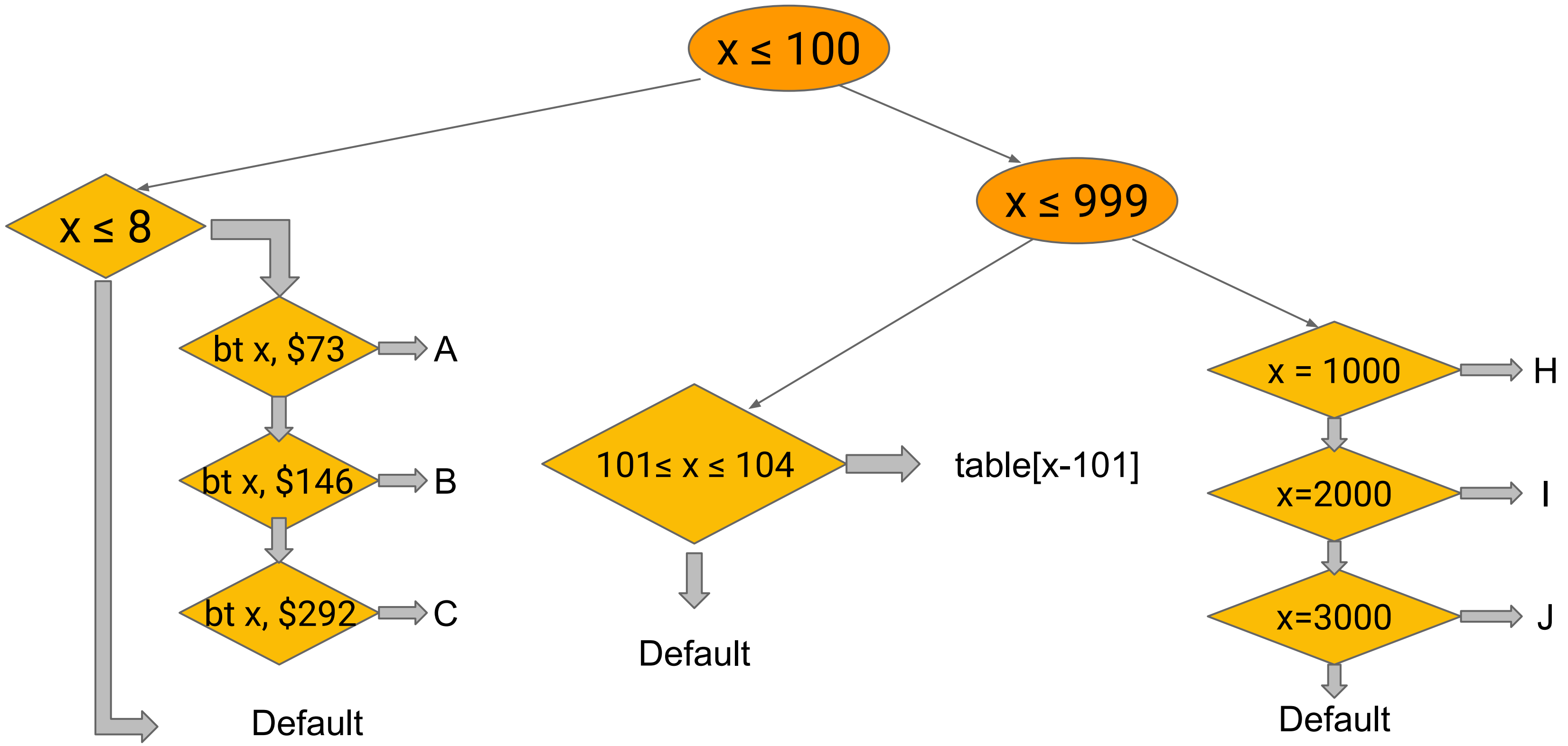


- Number of clusters ≥ 4
- Table density ≥ 40%

4. Binary search tree



4. Binary search tree

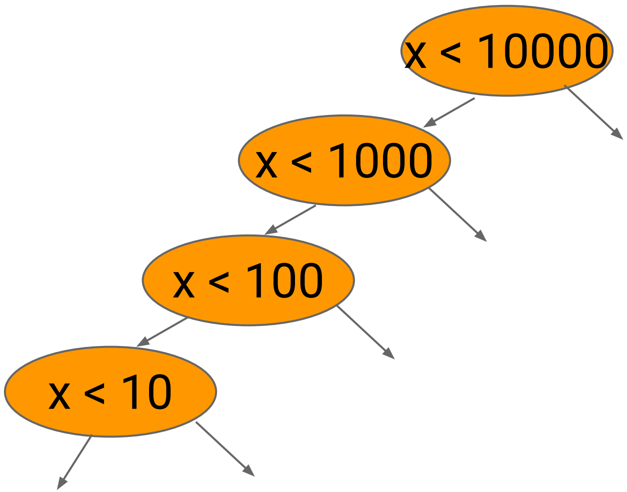


What changed?

Old algorithm: top-down

- Consider the range of cases
- Lower by cmps, bit tests or jump table? If yes, done
- Split the range in two*, creating BST
- Repeat for both sides

Old algorithm: pivot selection is hard



* Pivot heuristic: maximize gap size

and sum density of LHS and RHS.

Heuristic helps find jump tables

But trees might not be balanced

(PR22262)

New algorithm: bottom-up

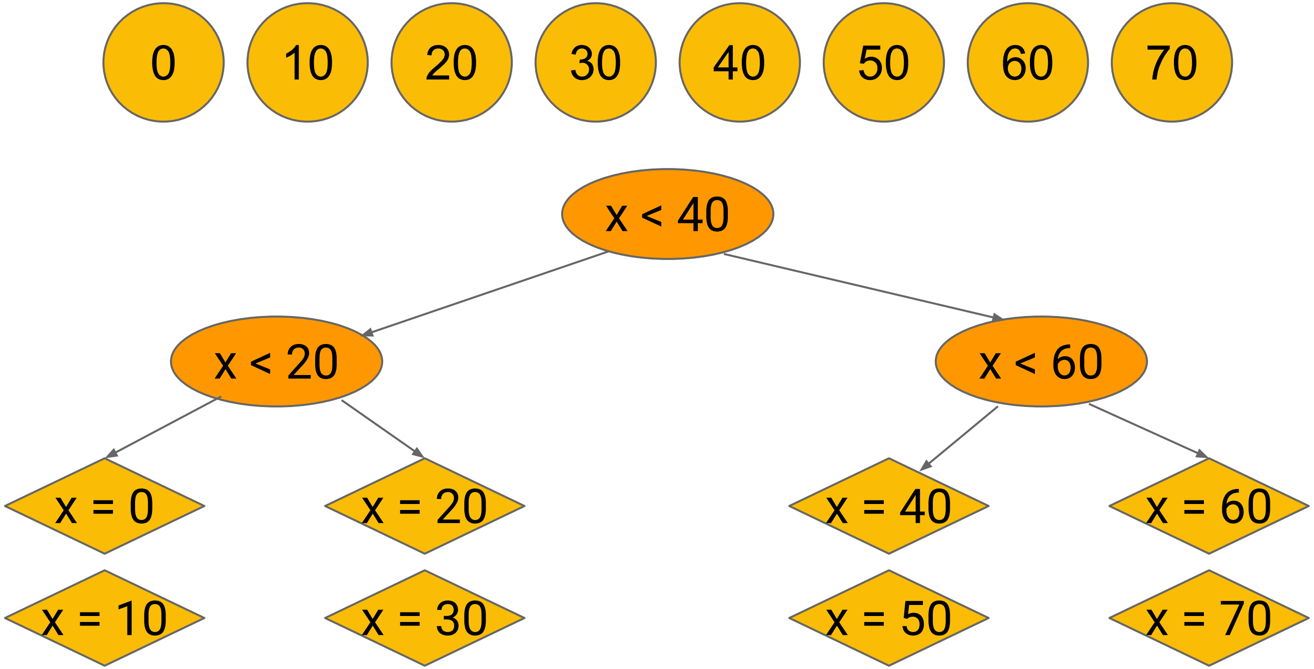
- Consider the whole range of cases
- Find case clusters suitable for bit tests
- Find case clusters suitable for jump tables
- Build binary search tree

New algorithm: benefits

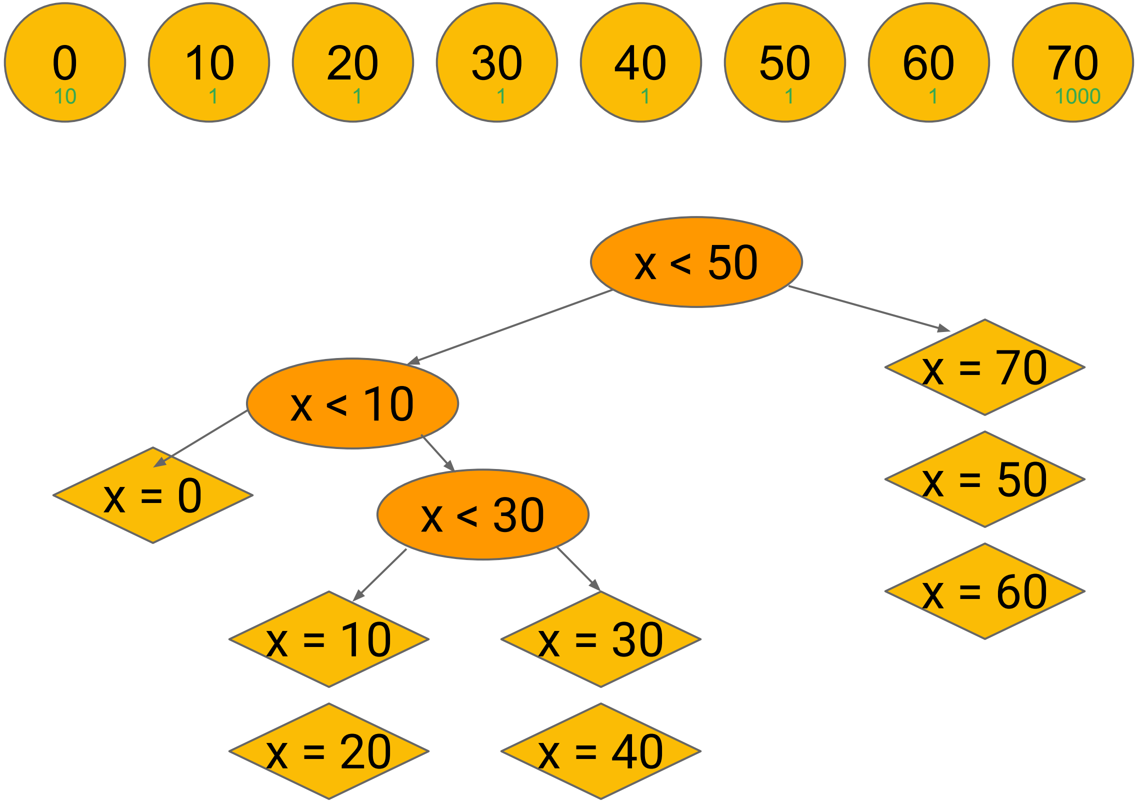
- Lowering strategies decoupled
  a. Code is easier to follow
  b. Can do less work at -O0
- Jump table extraction is optimal*
- BST will be balanced**

* For our size and density criteria
** Next slide!

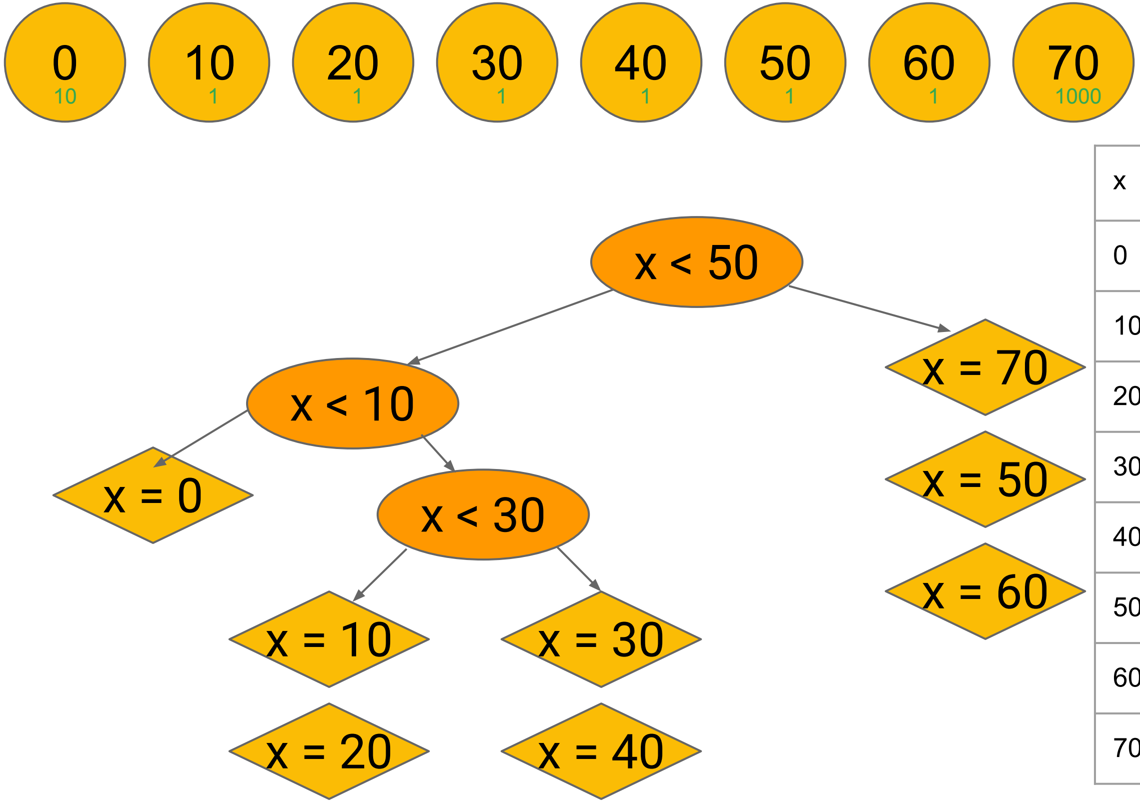
Balanced by node count



Balanced by node weight



Balanced by node weight



x	Branches	x weight
0	3	30
10	4	4
20	5	5
30	4	4
40	5	5
50	3	3
60	4	4
70	2	2000

(Without weight balancing: 3052) Sum: 2055

Summary

- Trees are balanced
- Jump tables are found
- Uses profile info