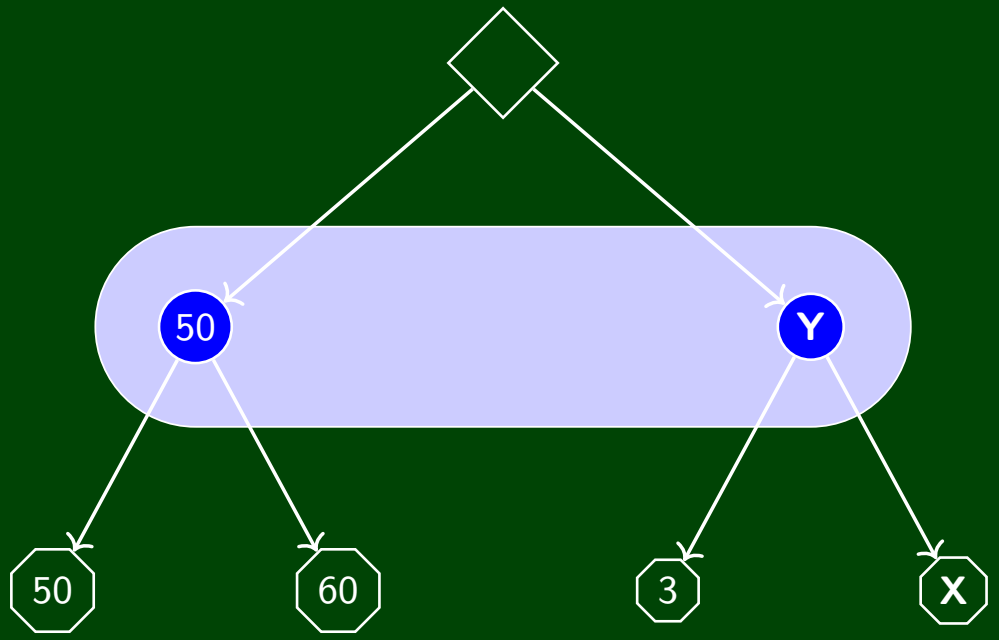
Introduction to Programming Methods

Alphabeta Pruning and Transposition Tables

Max's Turn

Min's Turn

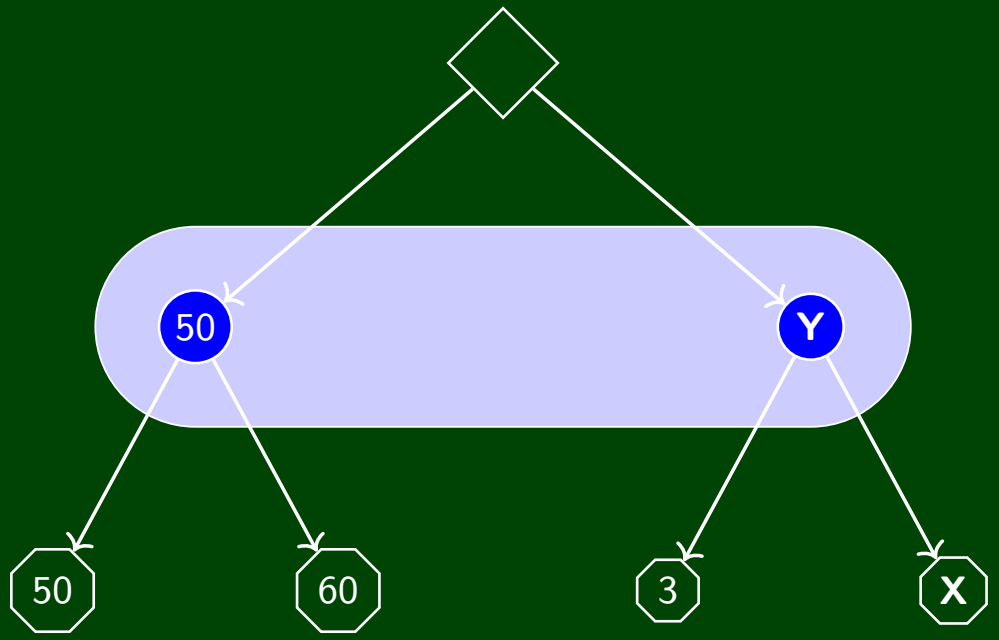
Max's Turn



Max's Turn

Min's Turn

Max's Turn

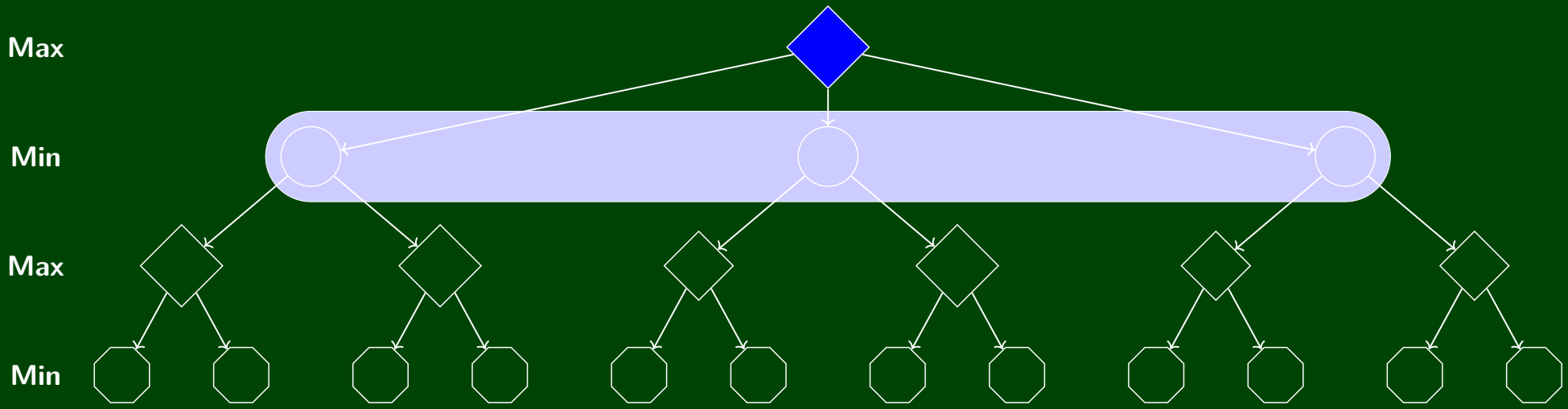


To fill in Y, **MIN** will take $\min(3, X)$. So, there are two cases:

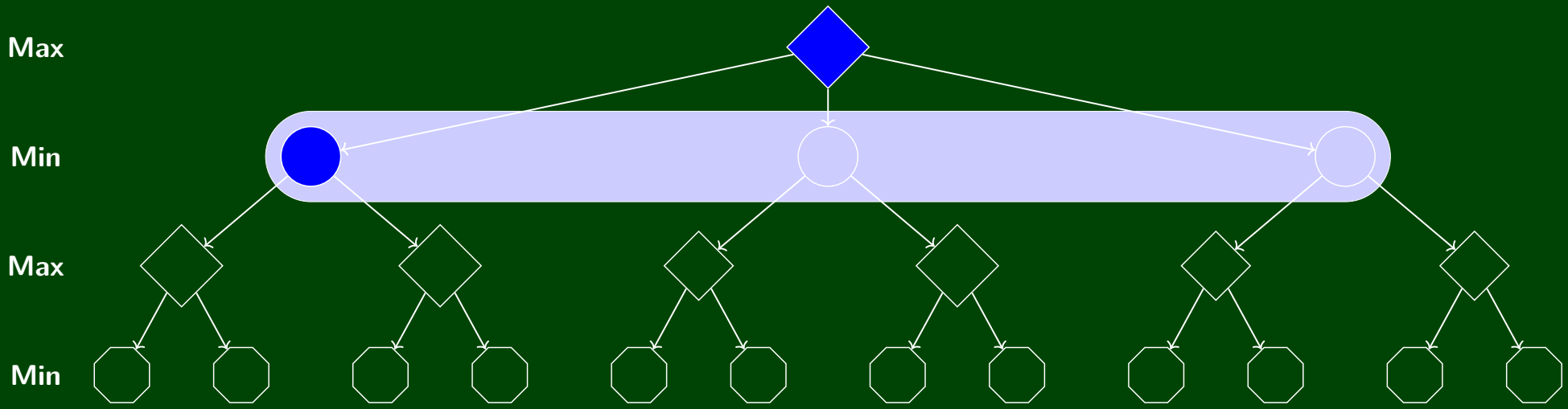
- $4 = X > 3$. Then, $Y = \min(3,4) = 3$. So, the box is 50.
- $2 = X < 3$. Then, $Y = \min(3,2) = 2$. So, the box is 50.

The values of X and Y don't matter! Don't calculate them!

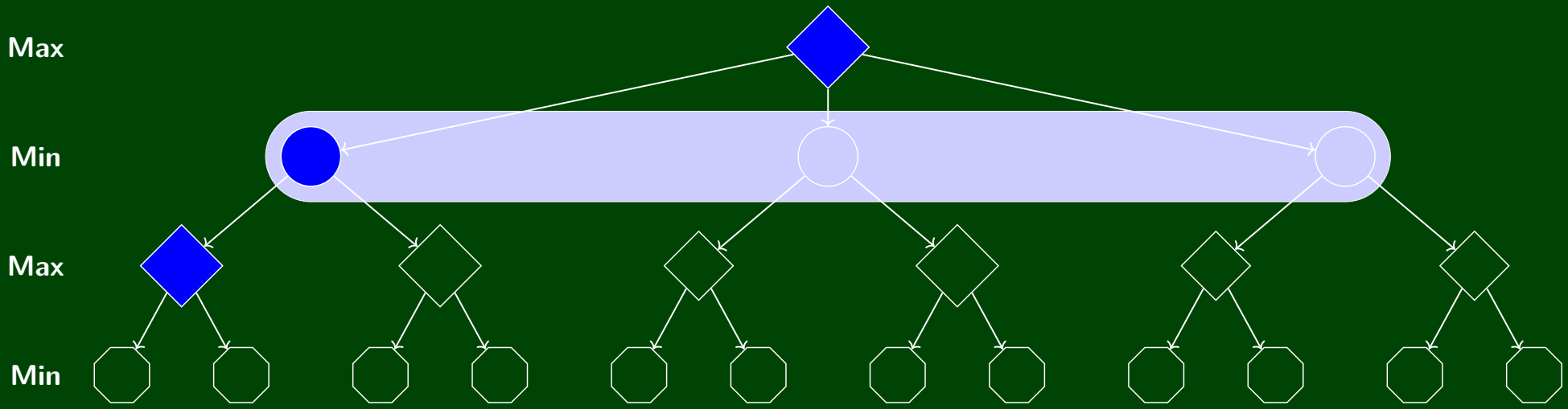
Pruning



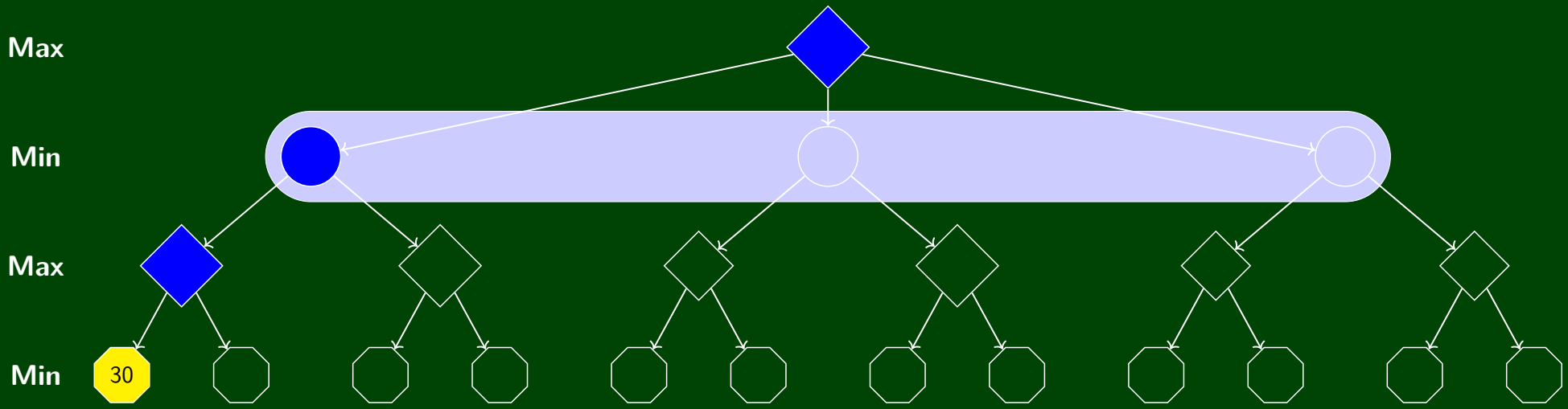
Pruning



Pruning



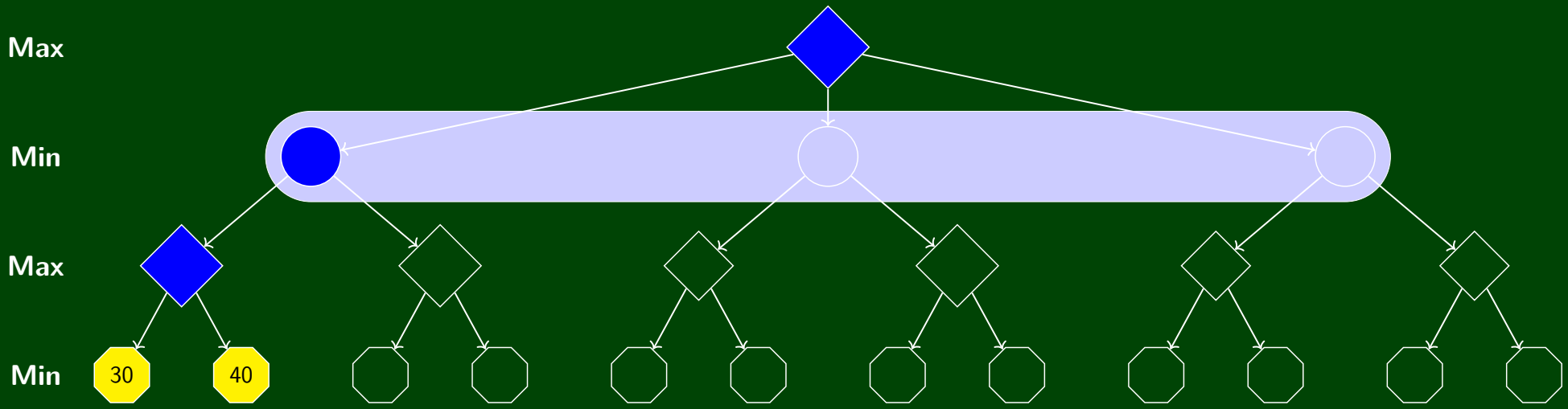
Pruning



Do we check the next node?

We currently have no information. So, yes!

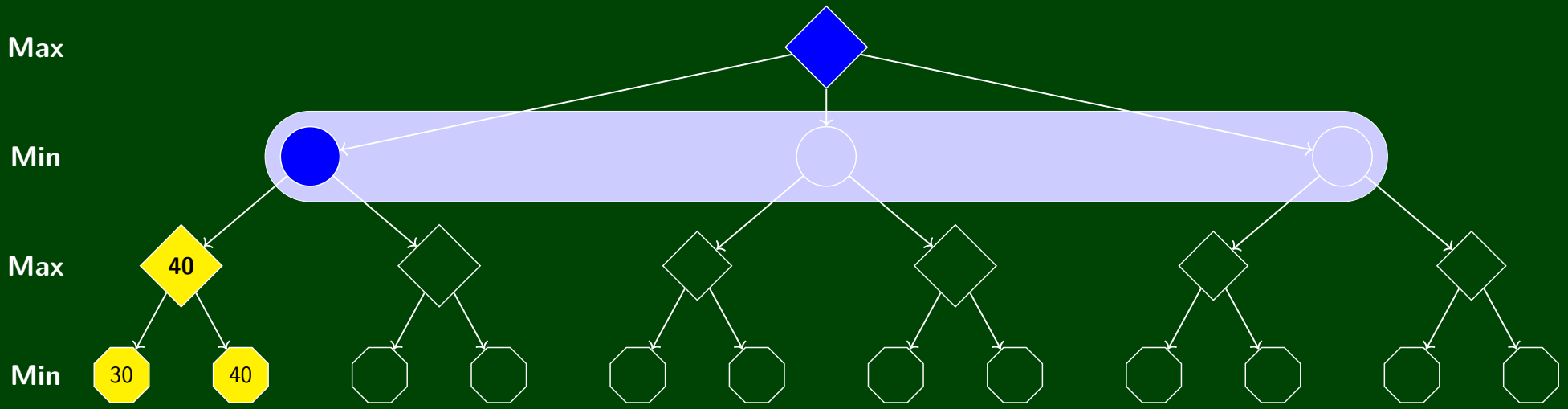
Pruning



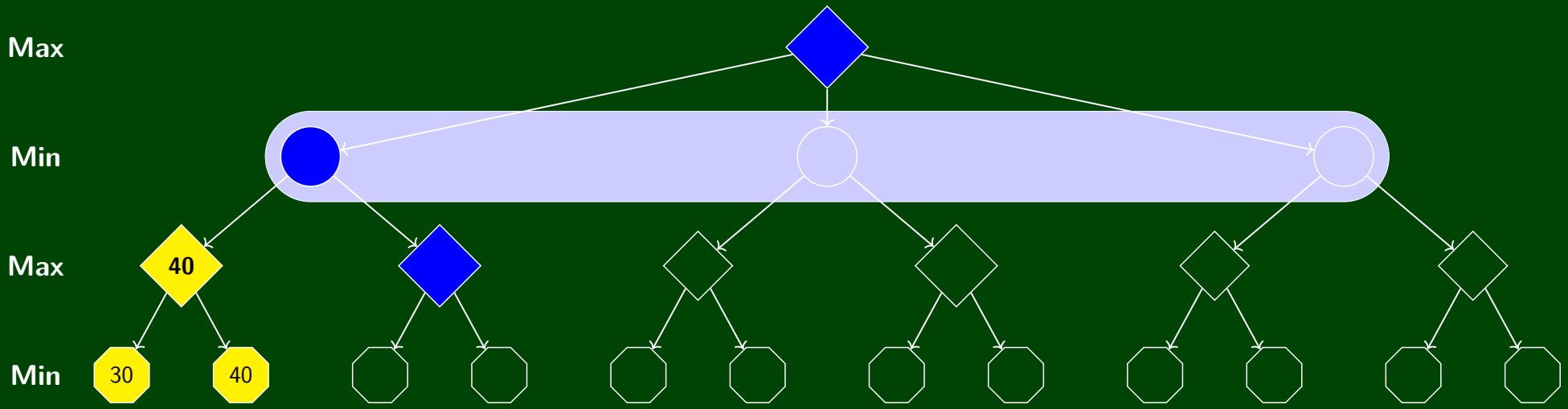
Do we check the next node?

We currently have no information. So, yes!

Pruning

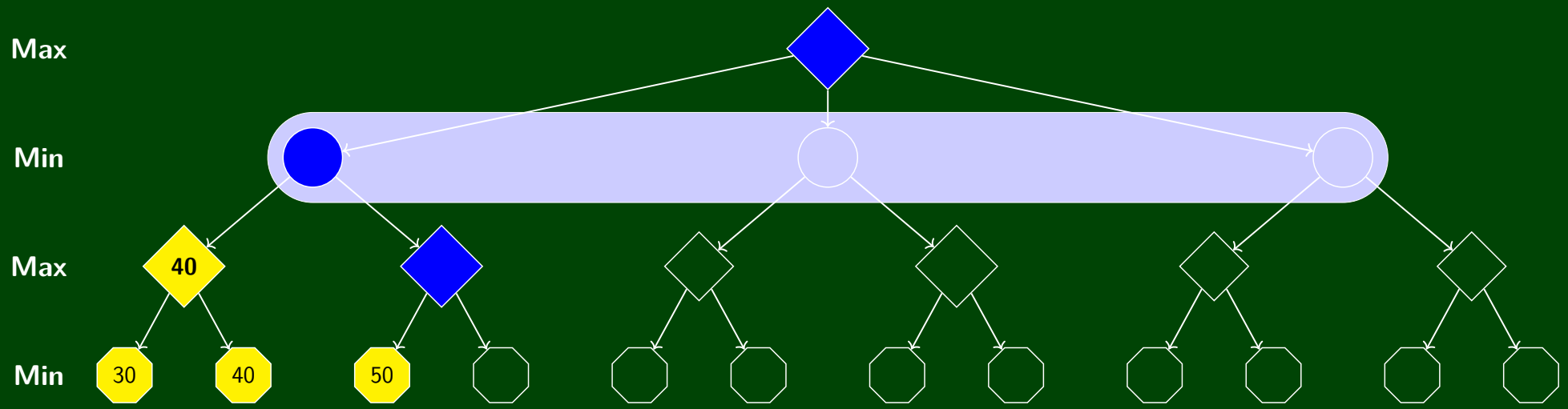


Pruning



Do we check the next node?

The current bounds are $[-\infty, 40]$. So, we **might** do better!

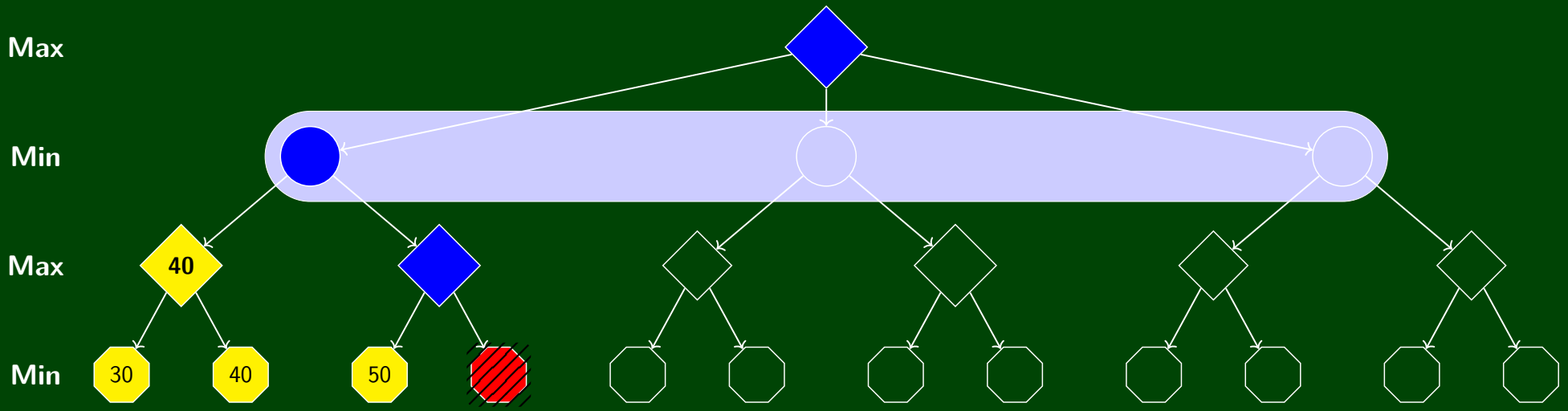


Do we check the next node?

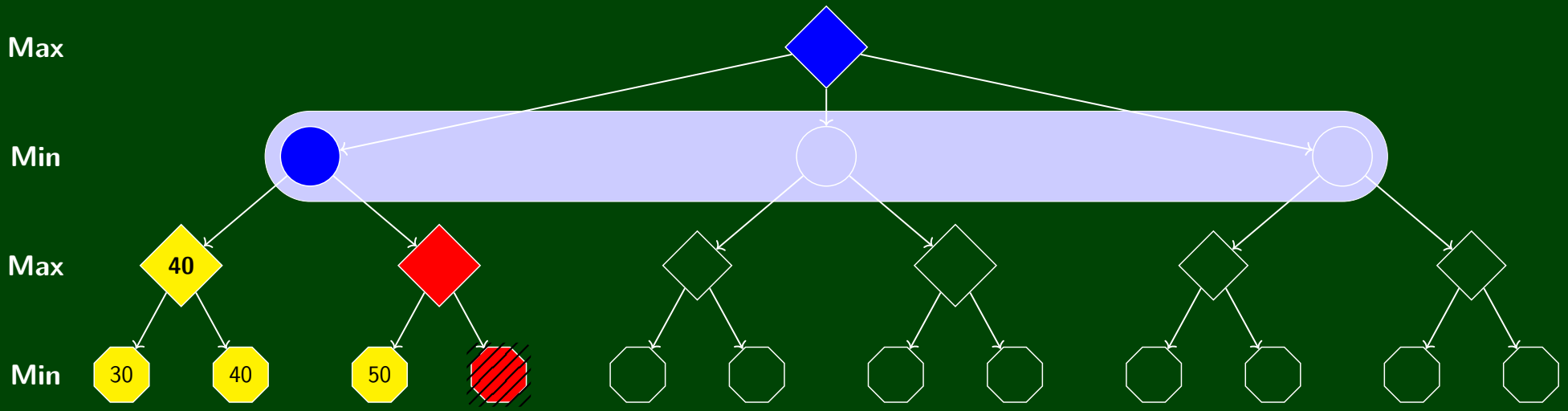
Max will choose $x \geq 50$ which is already worse than the 40.

The current bounds are $[50, 40]$. Don't bother.

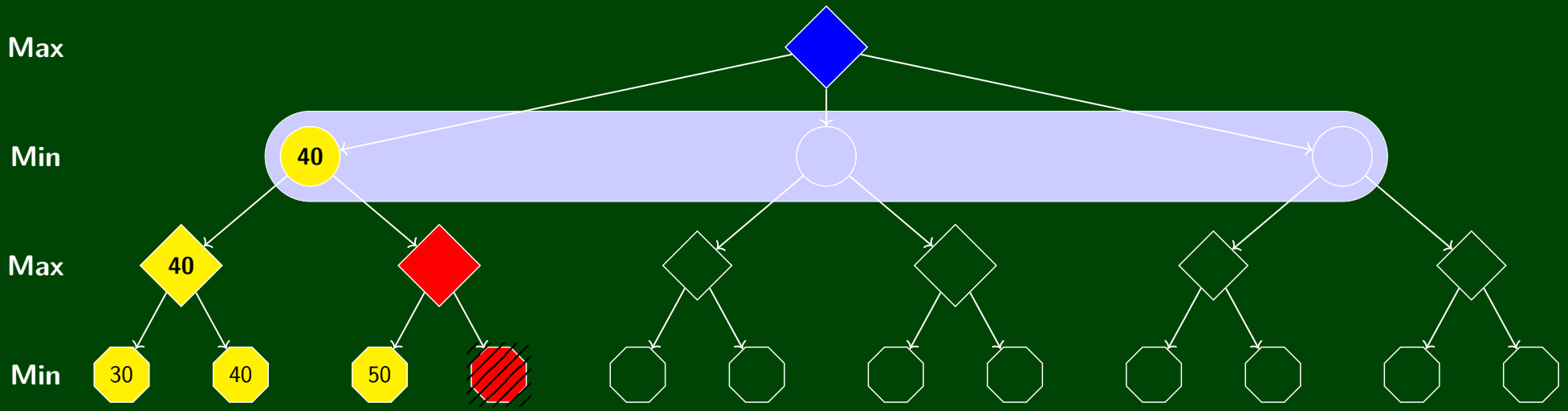
Pruning



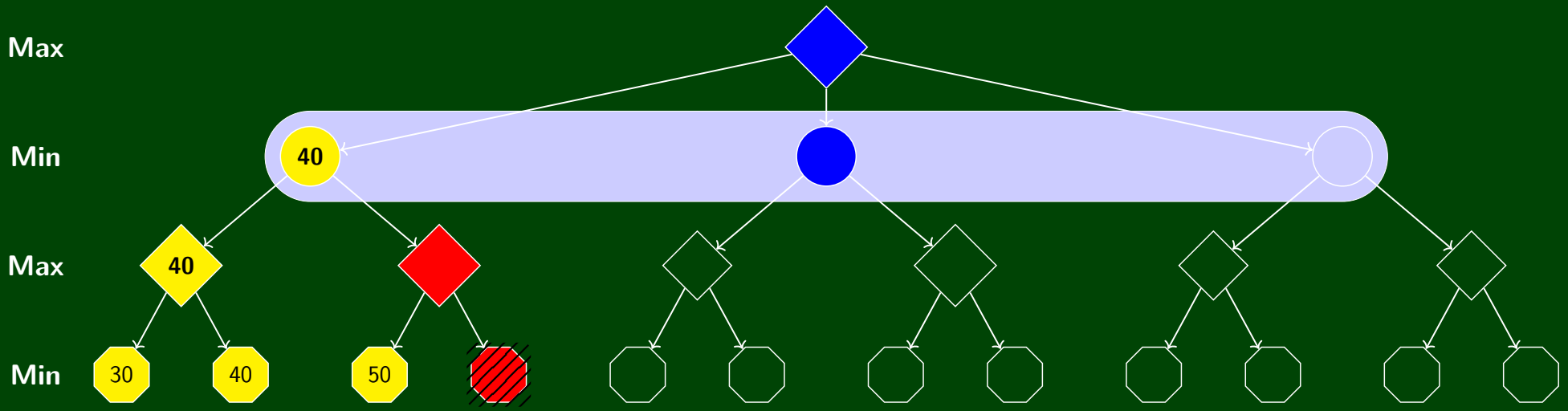
Pruning



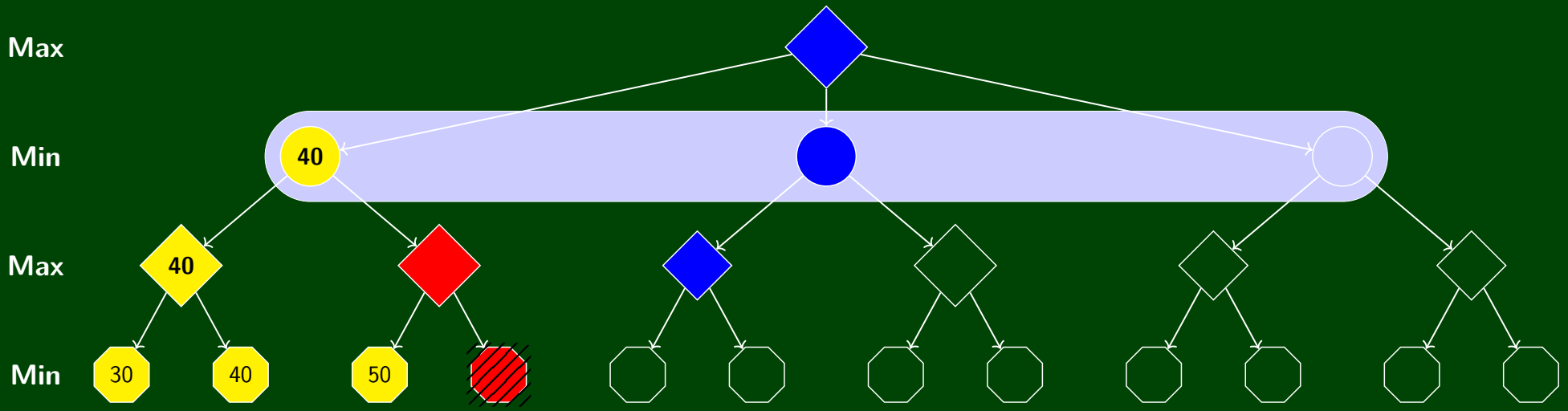
Pruning



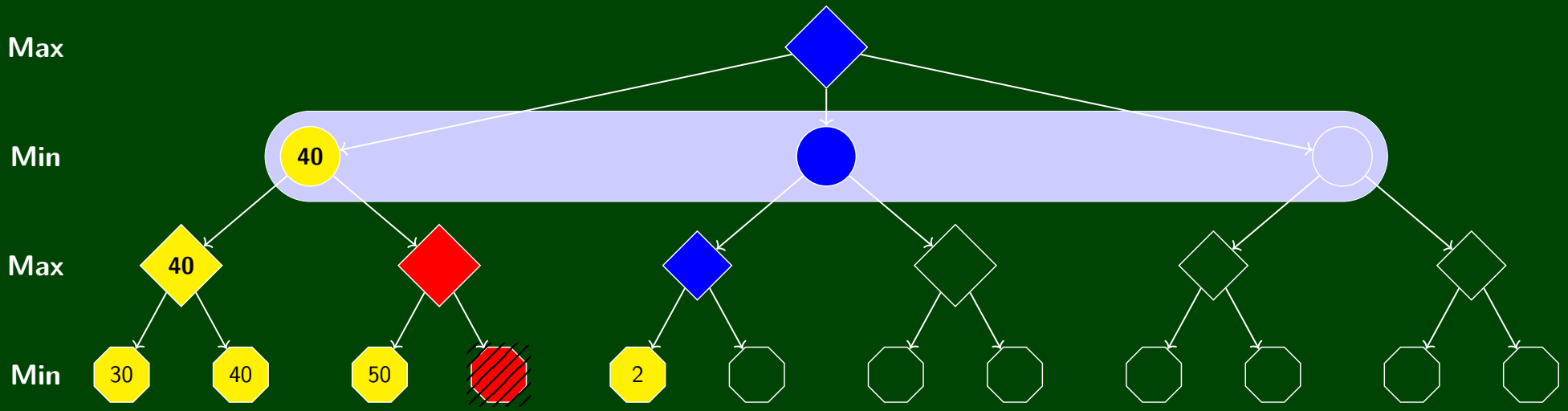
Pruning



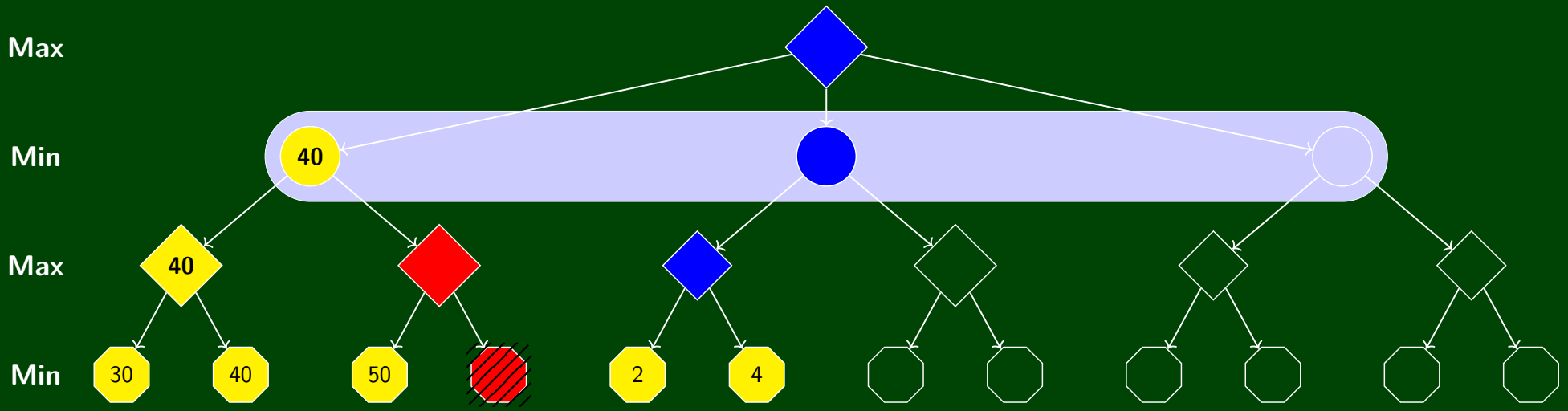
Pruning



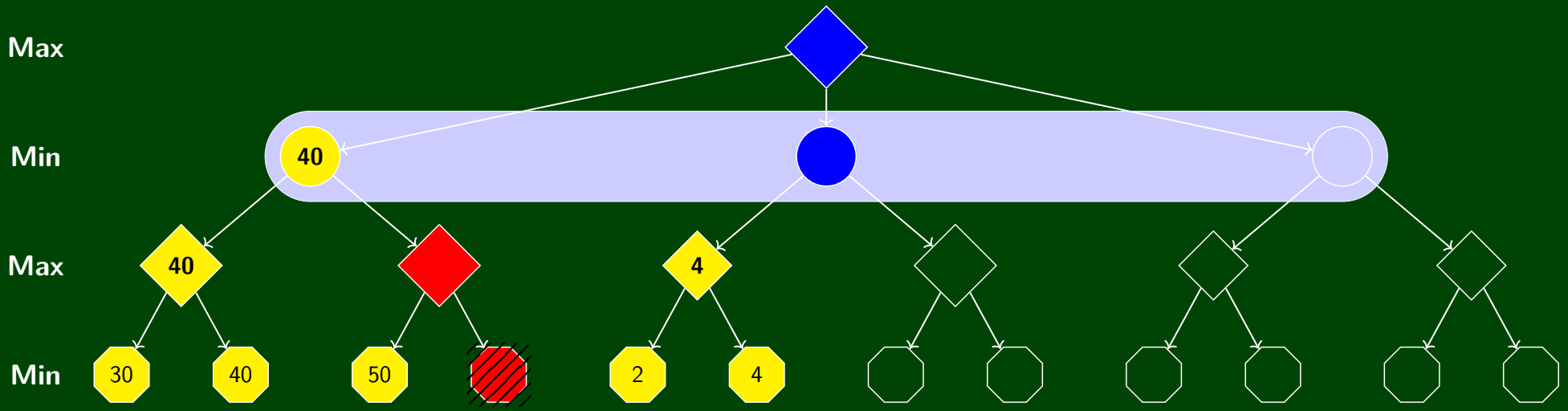
Pruning



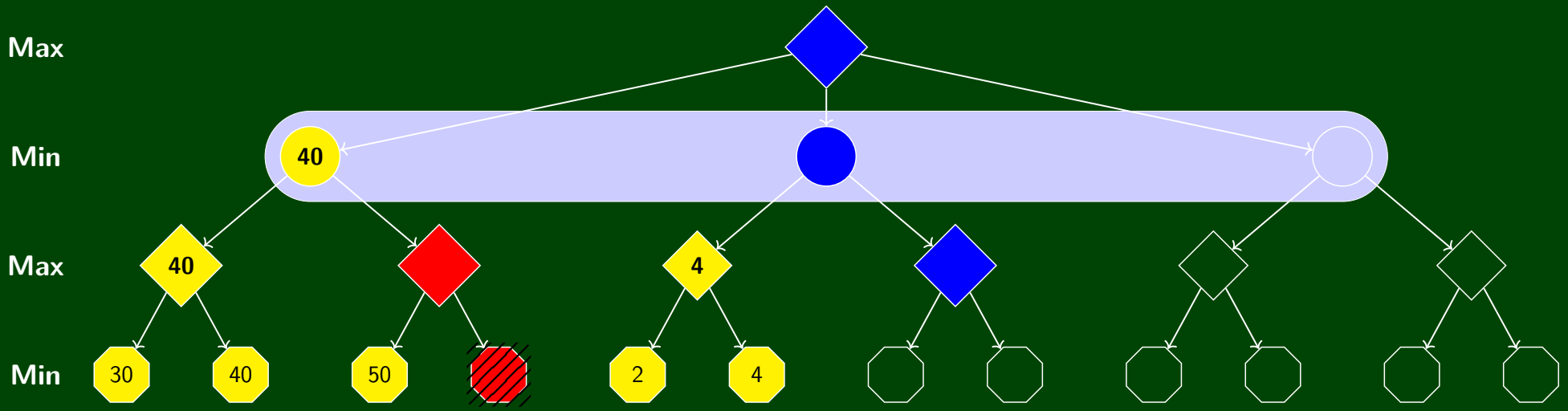
Pruning



Pruning



Pruning

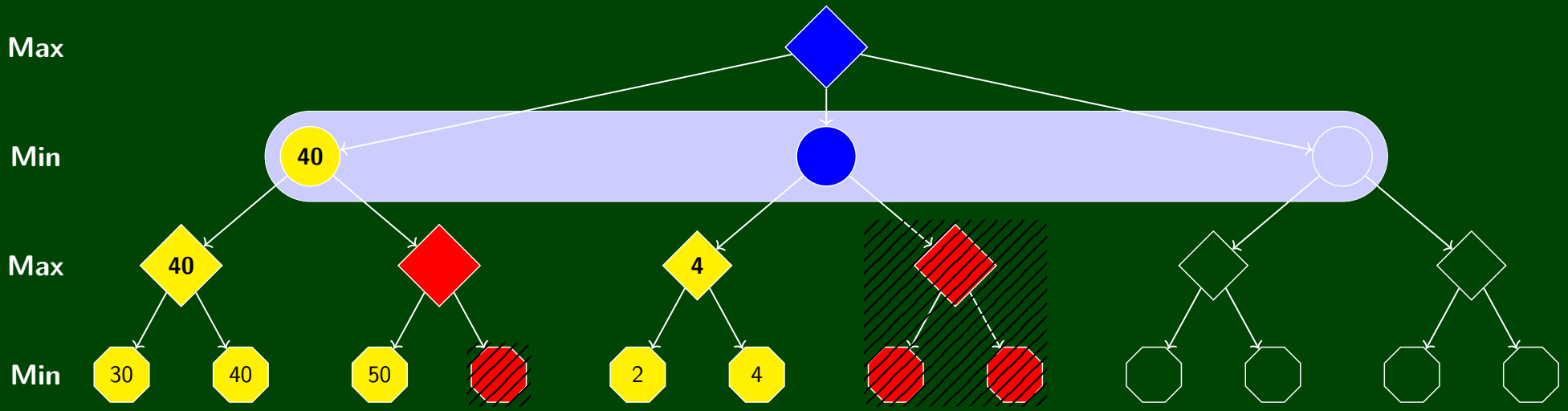


Do we check the next node?

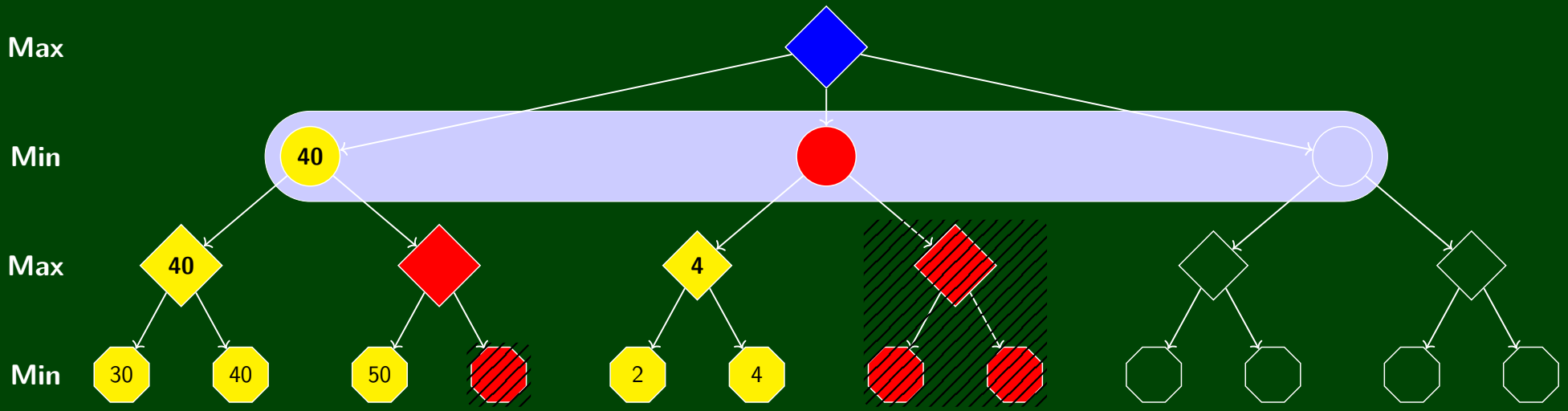
Min will choose $x \leq 4$ which is already worse than the 40.

The current bounds are $[40, 4]$. Don't bother.

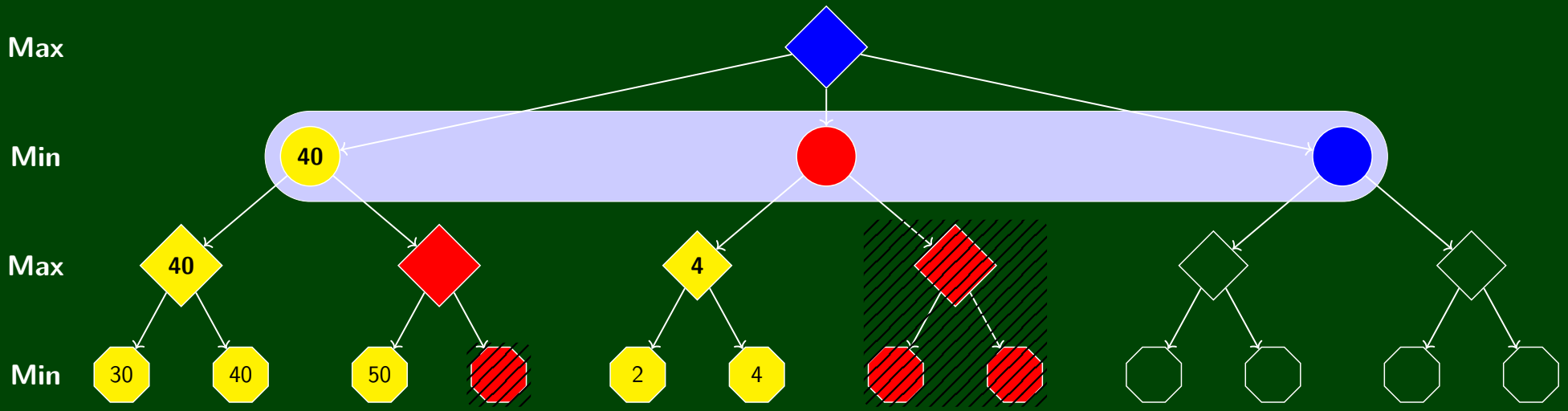
Pruning



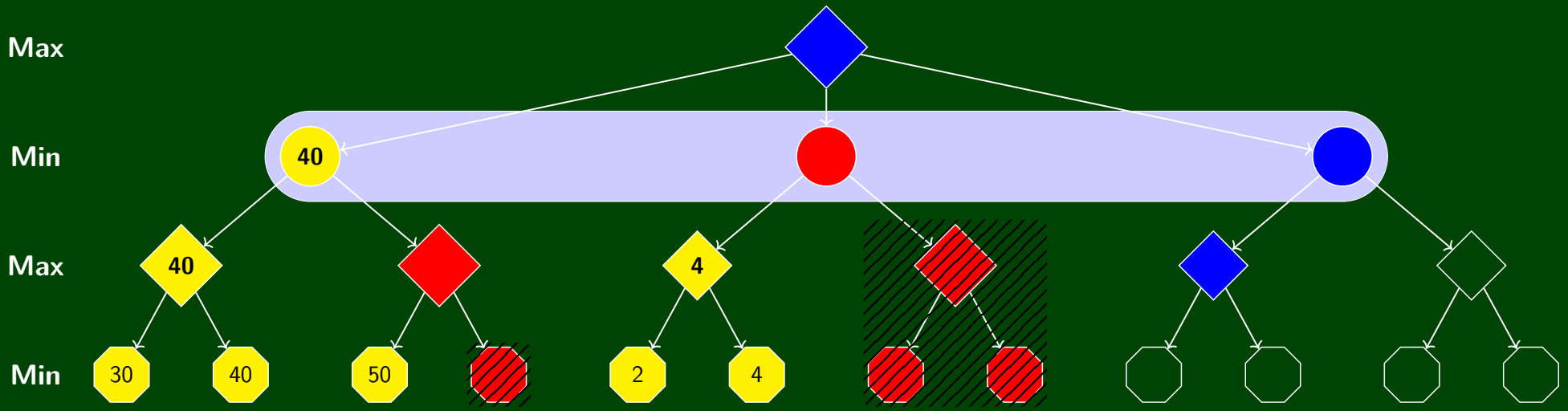
Pruning



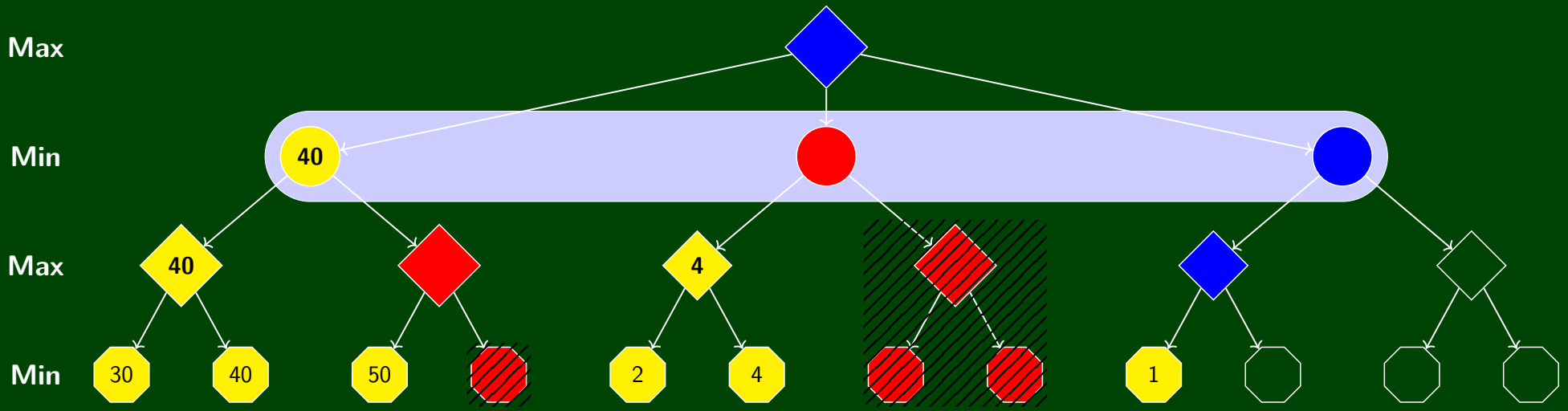
Pruning



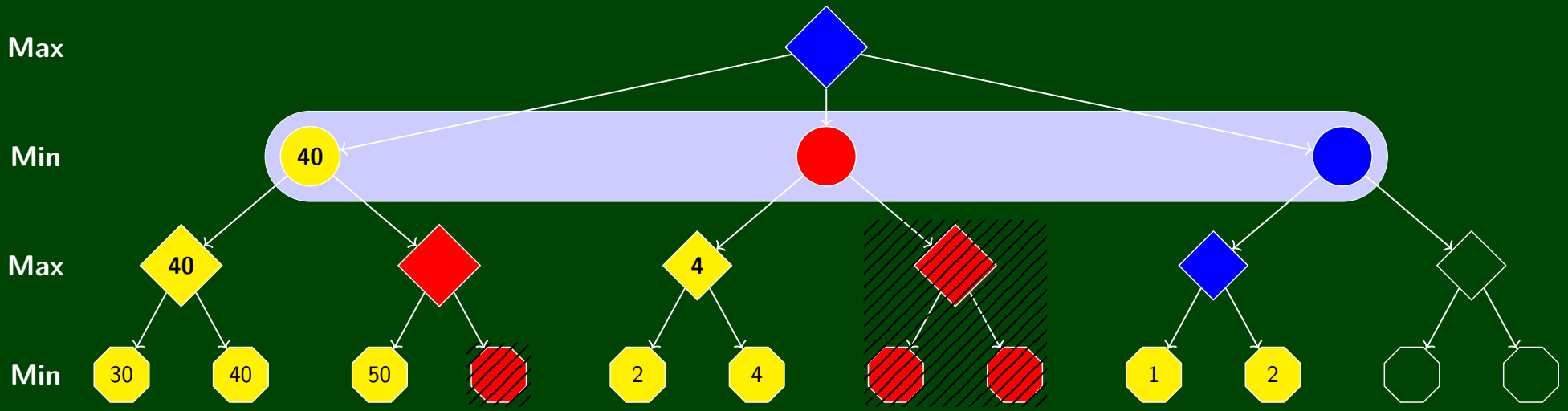
Pruning



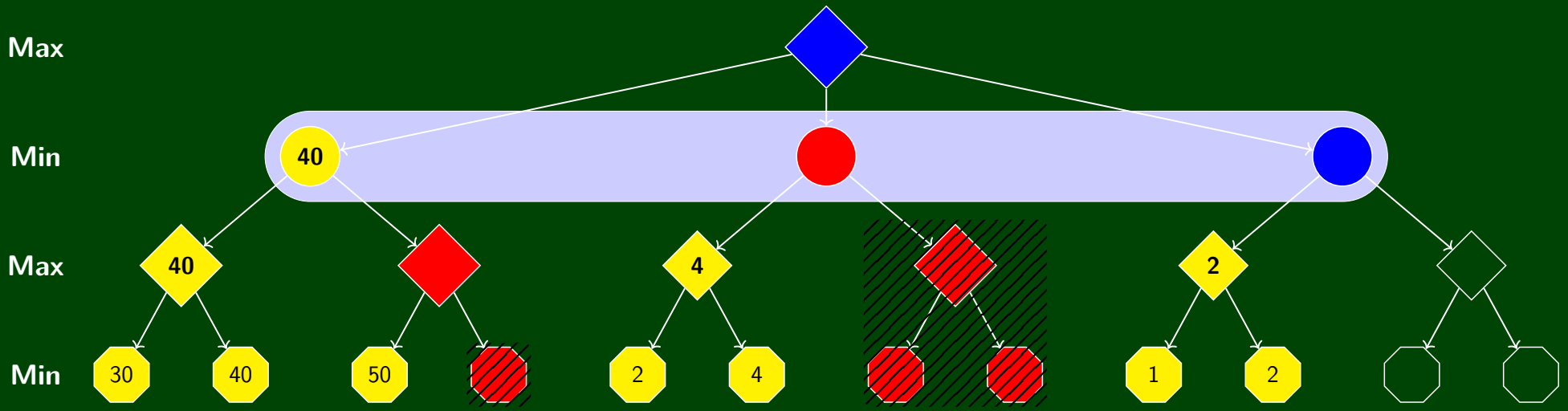
Pruning



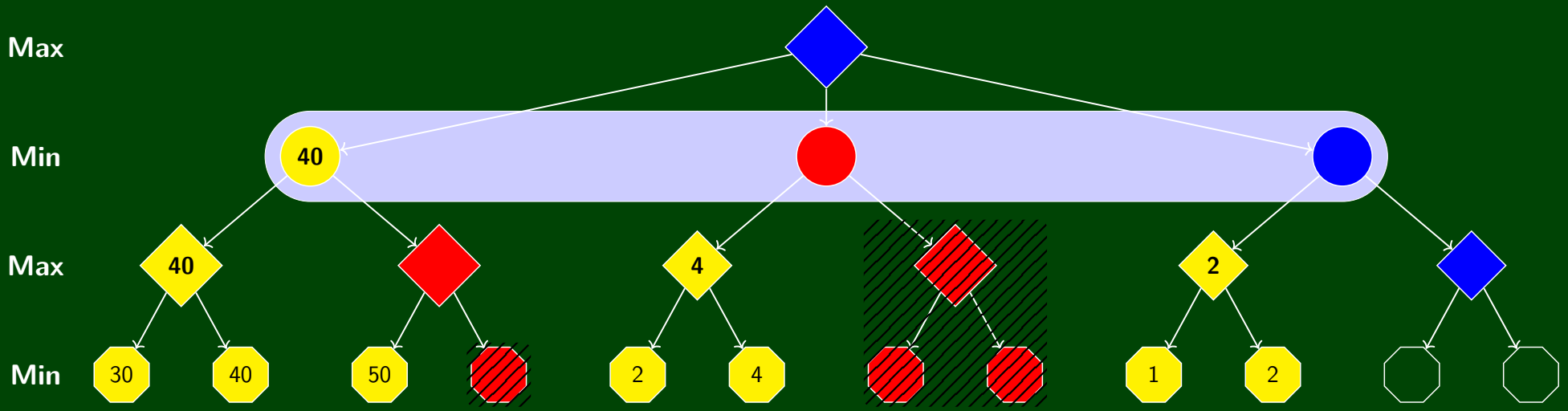
Pruning



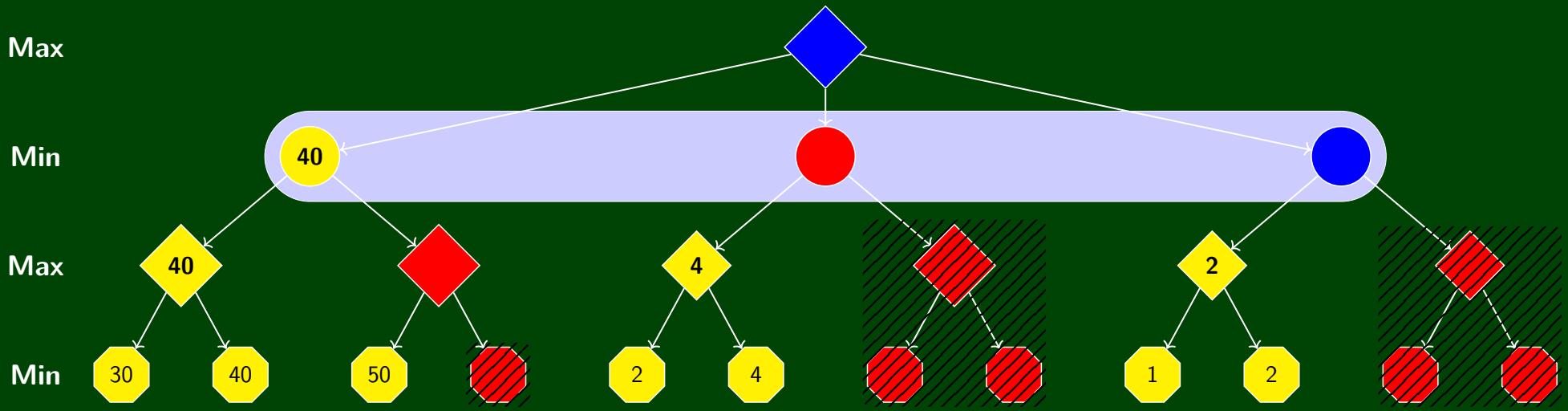
Pruning



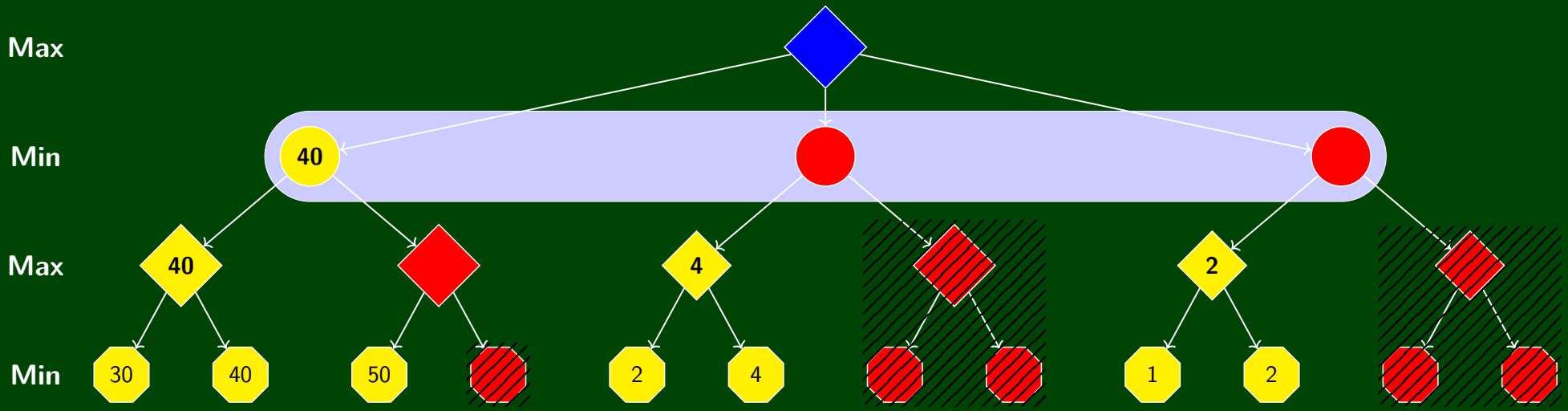
Pruning



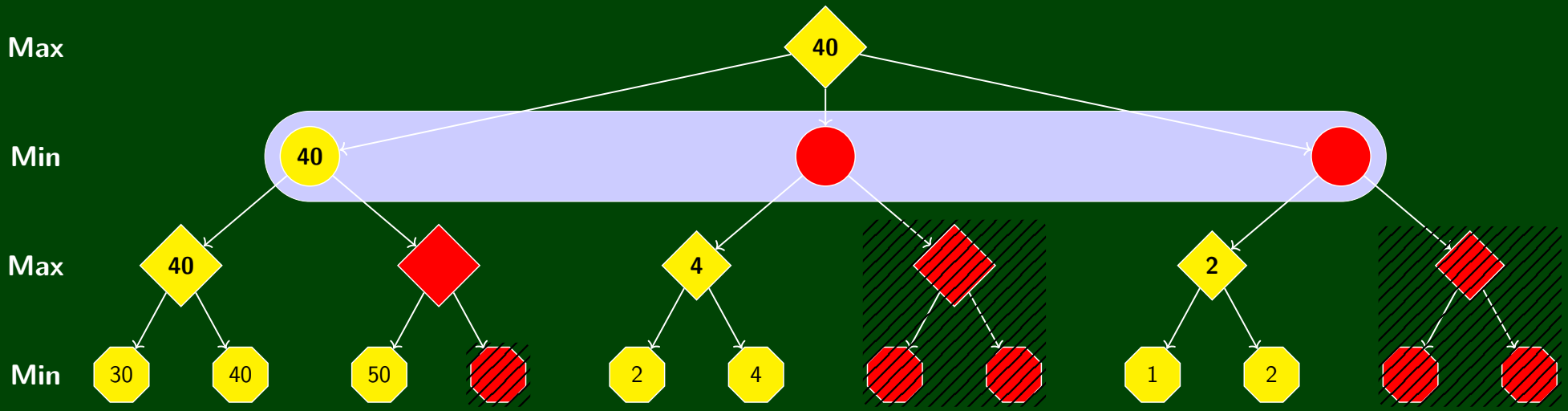
Pruning



Pruning



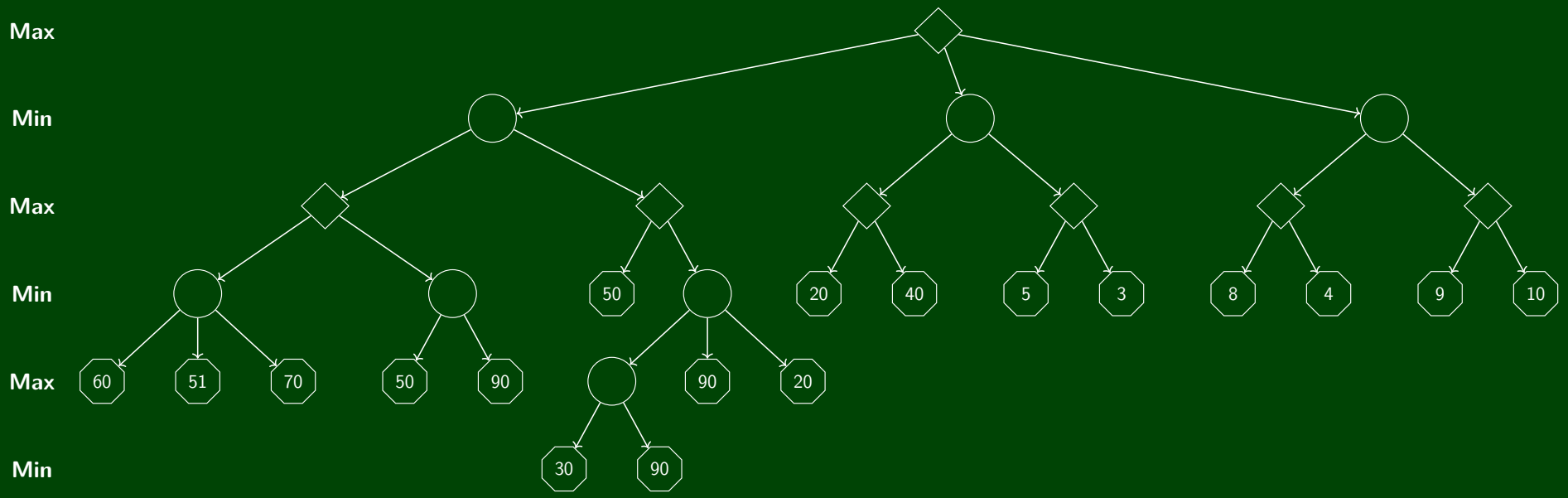
Pruning



The algorithm we just ran is called **AlphaBeta**.

α is the lower bound; β is the upper bound

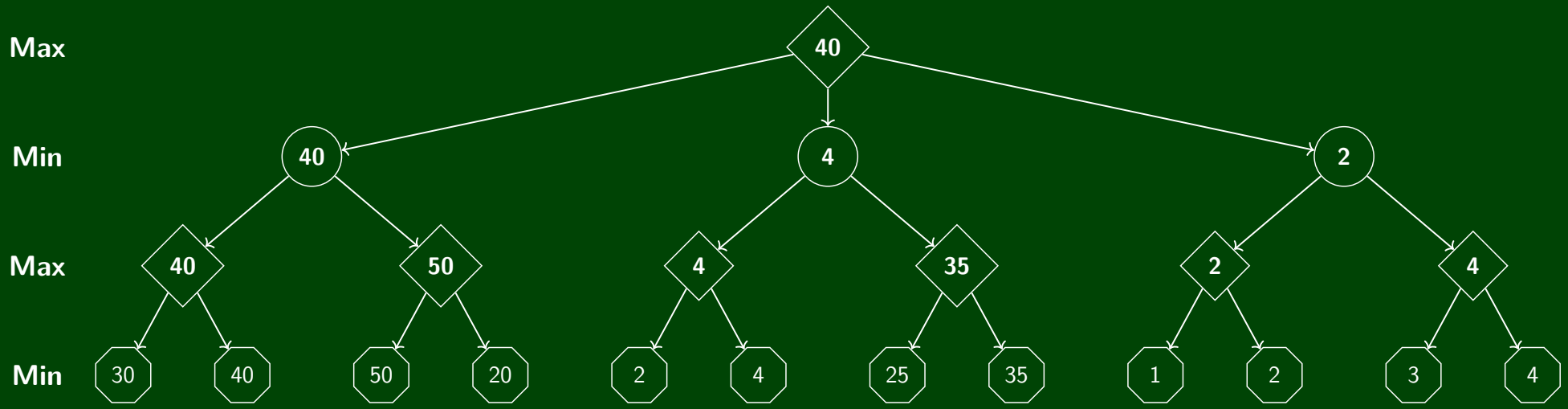
Another AlphaBeta Example

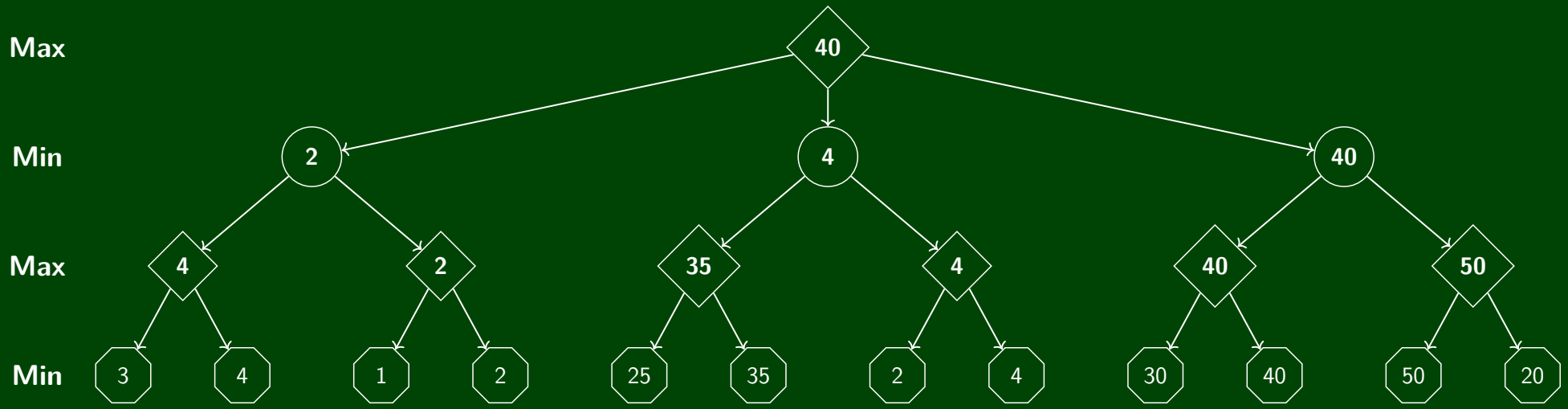


AlphaBeta Psuedocode


```
 1 int alphabeta(Position p, int alpha, int beta) {
 2    if (p.gameOver()) {
 3       return p.evaluate();
 4    }
 5
 6    for (Move m : p.getMoves()) {
 7       int value = -alphabeta(m, -beta, -alpha);
 8
 9       // If value is between alpha and beta, we've
10       // found a new lower bound
11       if (value > alpha) {
12          alpha = value;
13       }
14
15       // If the value is bigger than beta, we won't
16       // actually be able to get this move
17       if (alpha >= beta) {
18          return alpha;
19       }
20    }
21
22    // Return the best achievable value
23    return alpha;
24 }
```

Move Ordering





A **branching factor** is how many times a node splits at each level. In Othello, for a random position, the average branching factor is:

10

The average Othello game lasts about

58 Moves

If we wanted to evaluate the whole game, we would be evaluating 10^{58} **leaves**. If we were able to evaluate **1 trillion** leaves a second, we would need 10^{46} seconds.

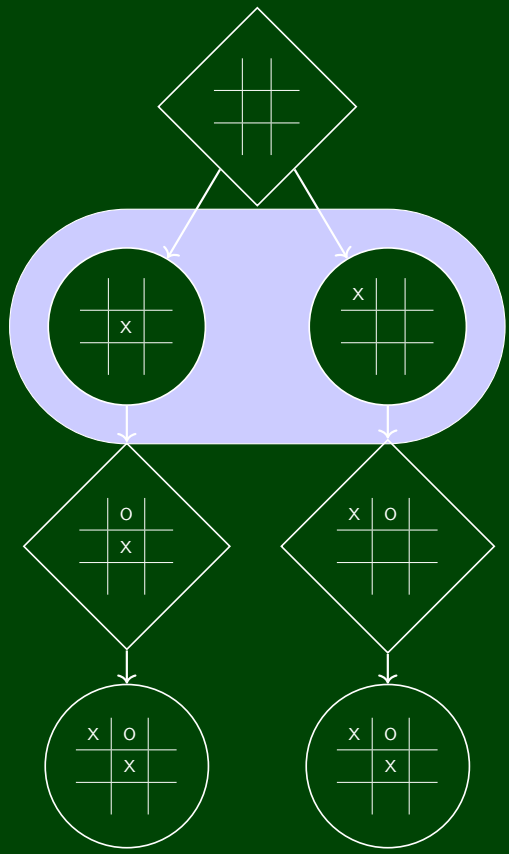
Depth-Limited Search

X's Turn

O's Turn

X's Turn

O's Turn



Zobrist Hashing