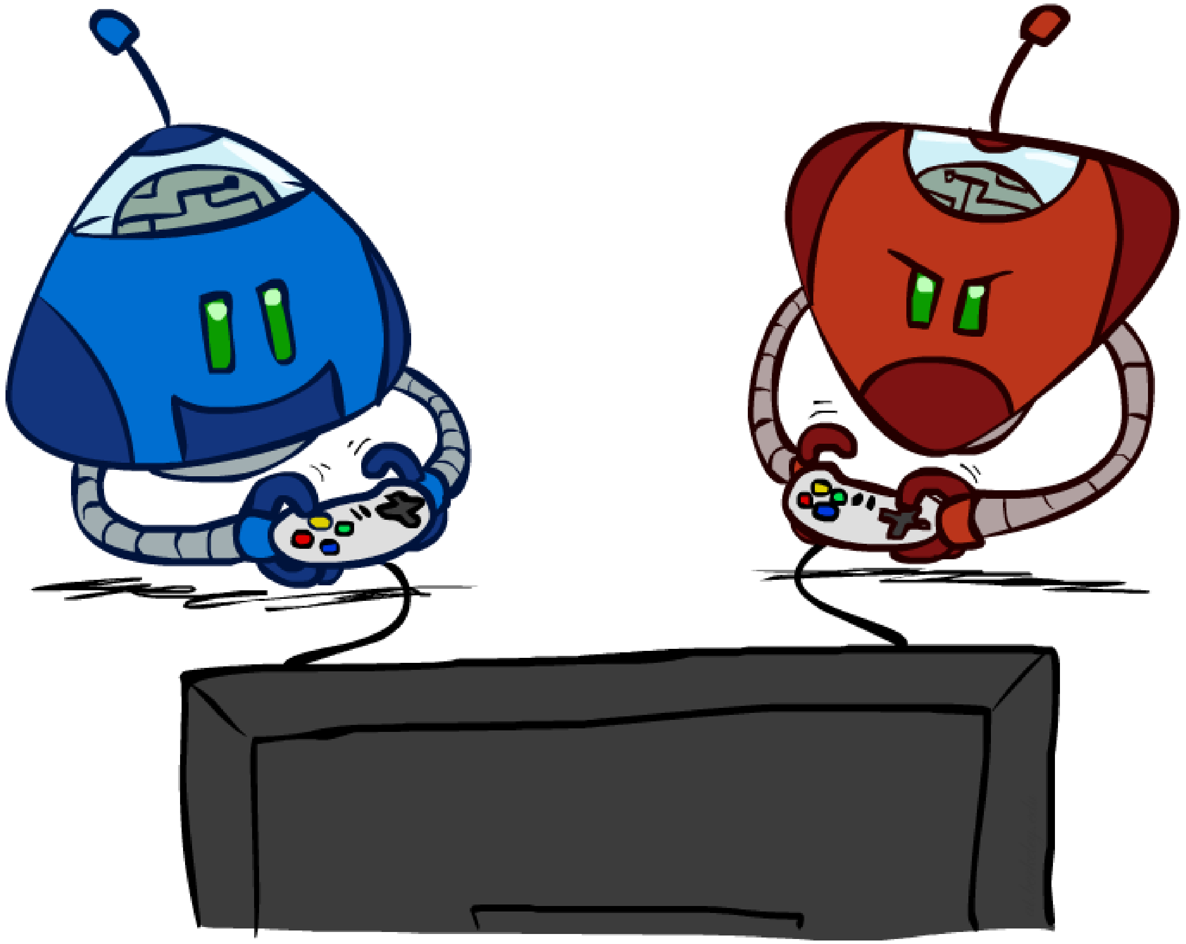
Adversarial Search

Dr. Emad Natsheh

Space of Search Strategies

- Blind Search
  - DFS, BFS, IDS, Uniform cost
- Informed Search
  - Systematic: BFS, A*
  - Stochastic: Hill climbing w/ random walk & restarts
- Adversary Search
  - Mini-max
  - Alpha-beta

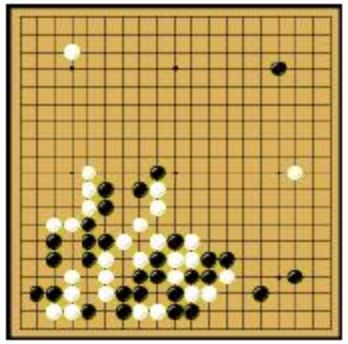
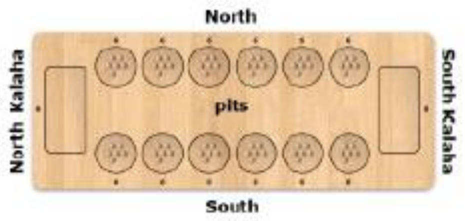


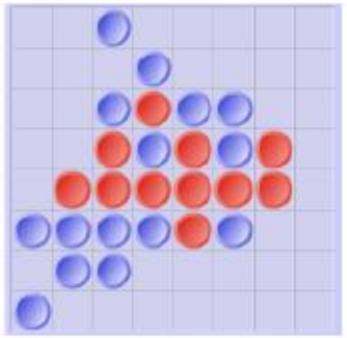
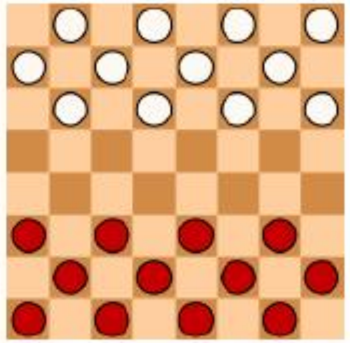
Adversarial Search

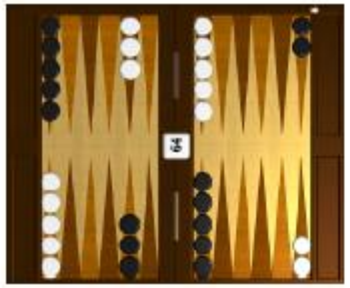
Types of Games

- Many different kinds of games
- Axes
  - Deterministic or stochastic?
  - One two or more player?
  - Zero sum?
  - Perfect information (you can see the state)

	deterministic	chance
perfect information	chess, checkers, kalaha go, othello	backgammon, monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble

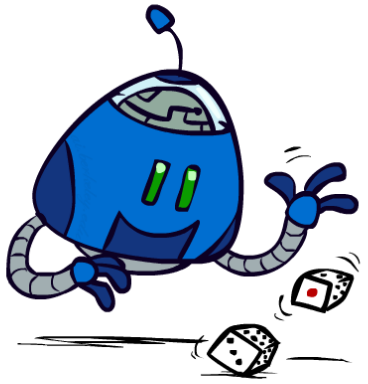



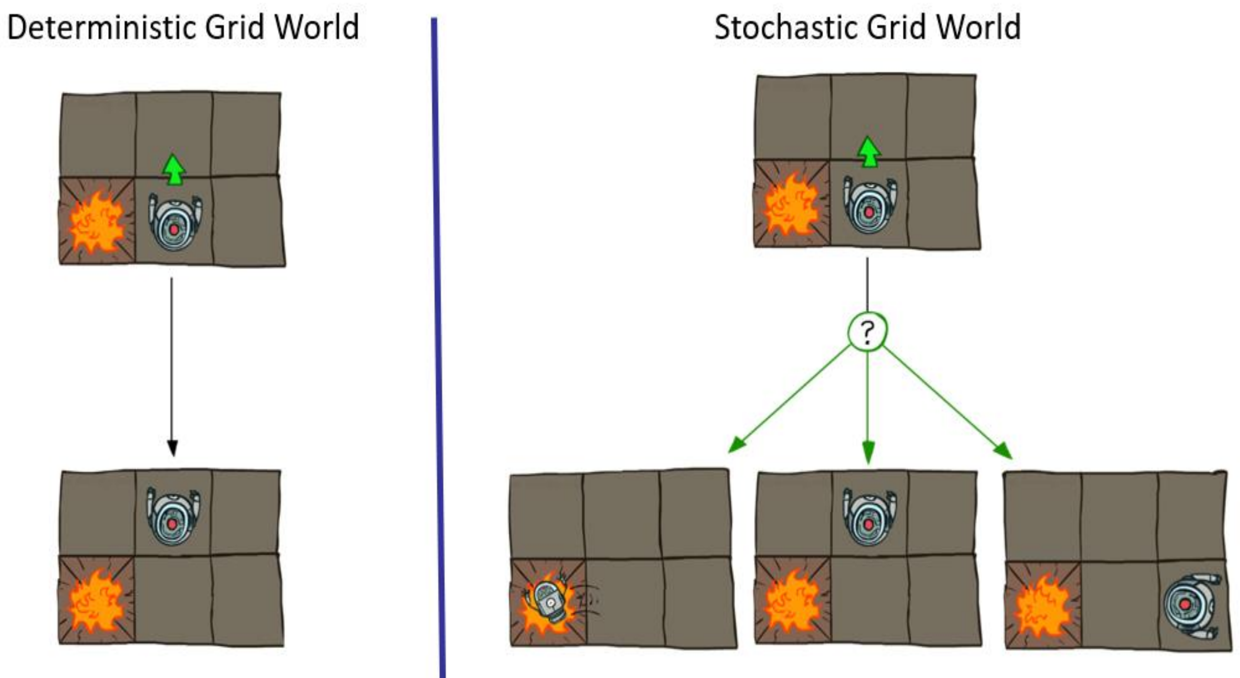





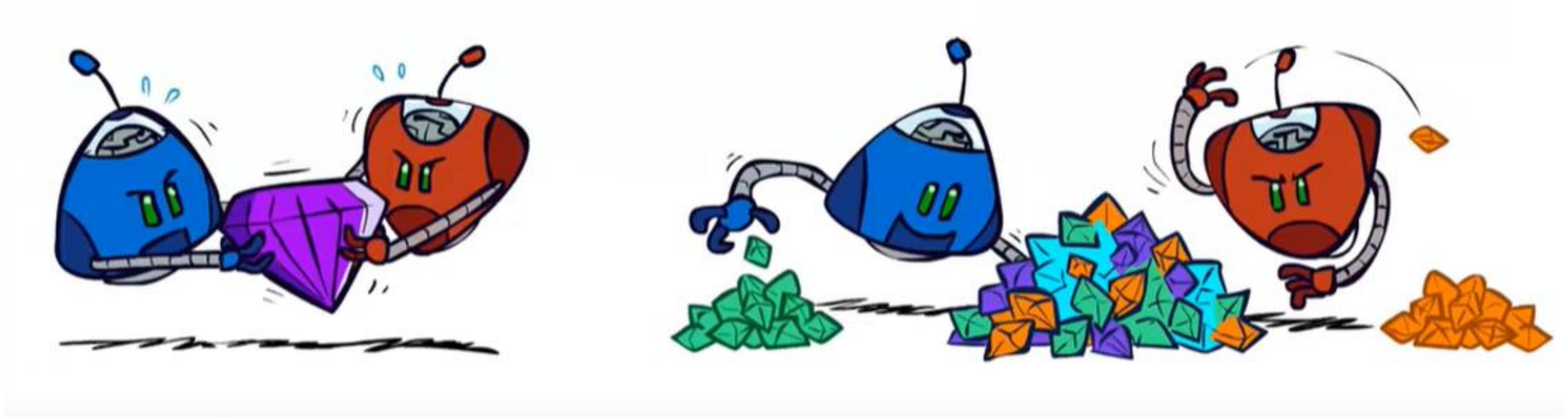
Deterministic and Perfect Information Games

- Perfect Information Games
  - States
  - Actions
  - Successor function
  - Terminal test
- Deterministic
- Non-Deterministic: You can look at Expectimax search, and Markov Decision Processes



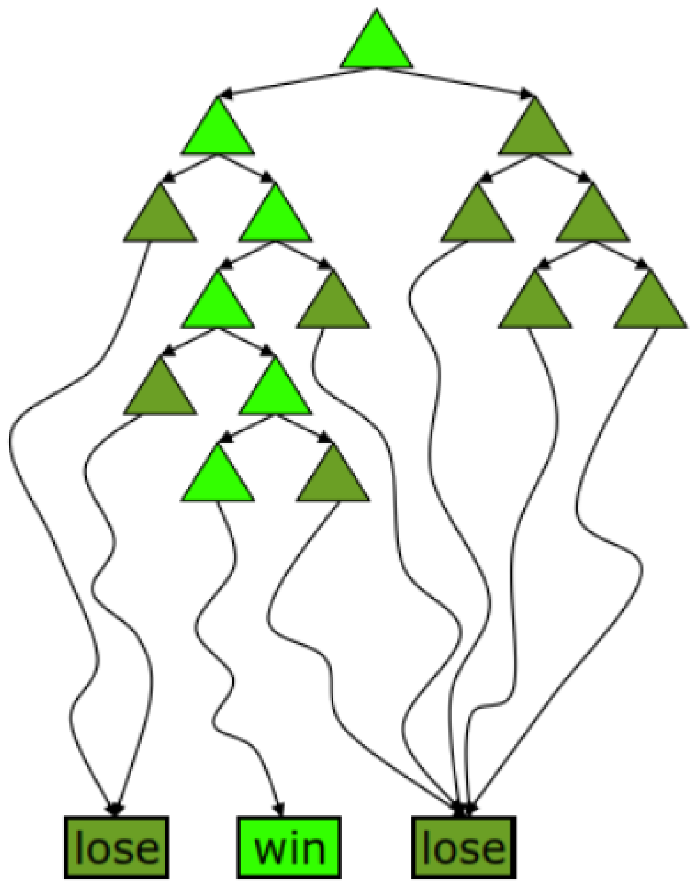


Zero Sum Games



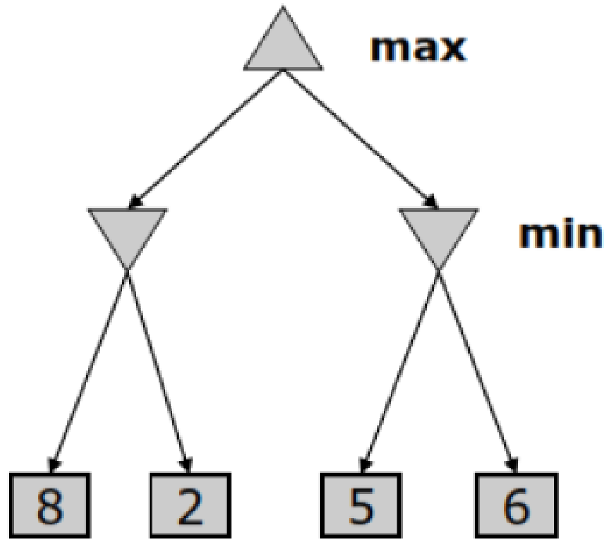
Deterministic Single-Player

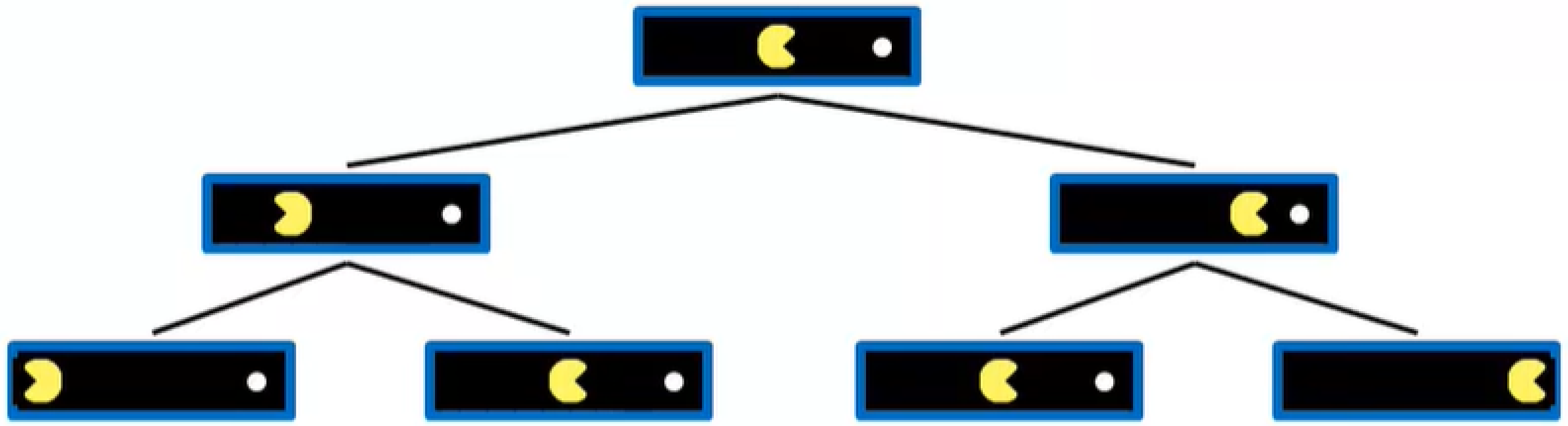
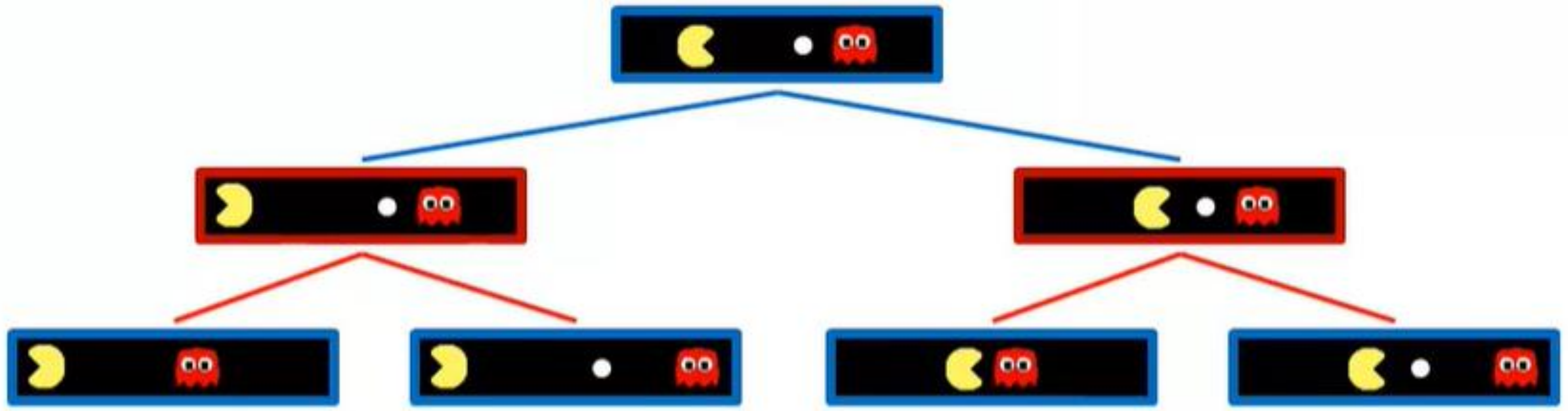
- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- **... it's just search!**
- After search, can pick move that leads to best node

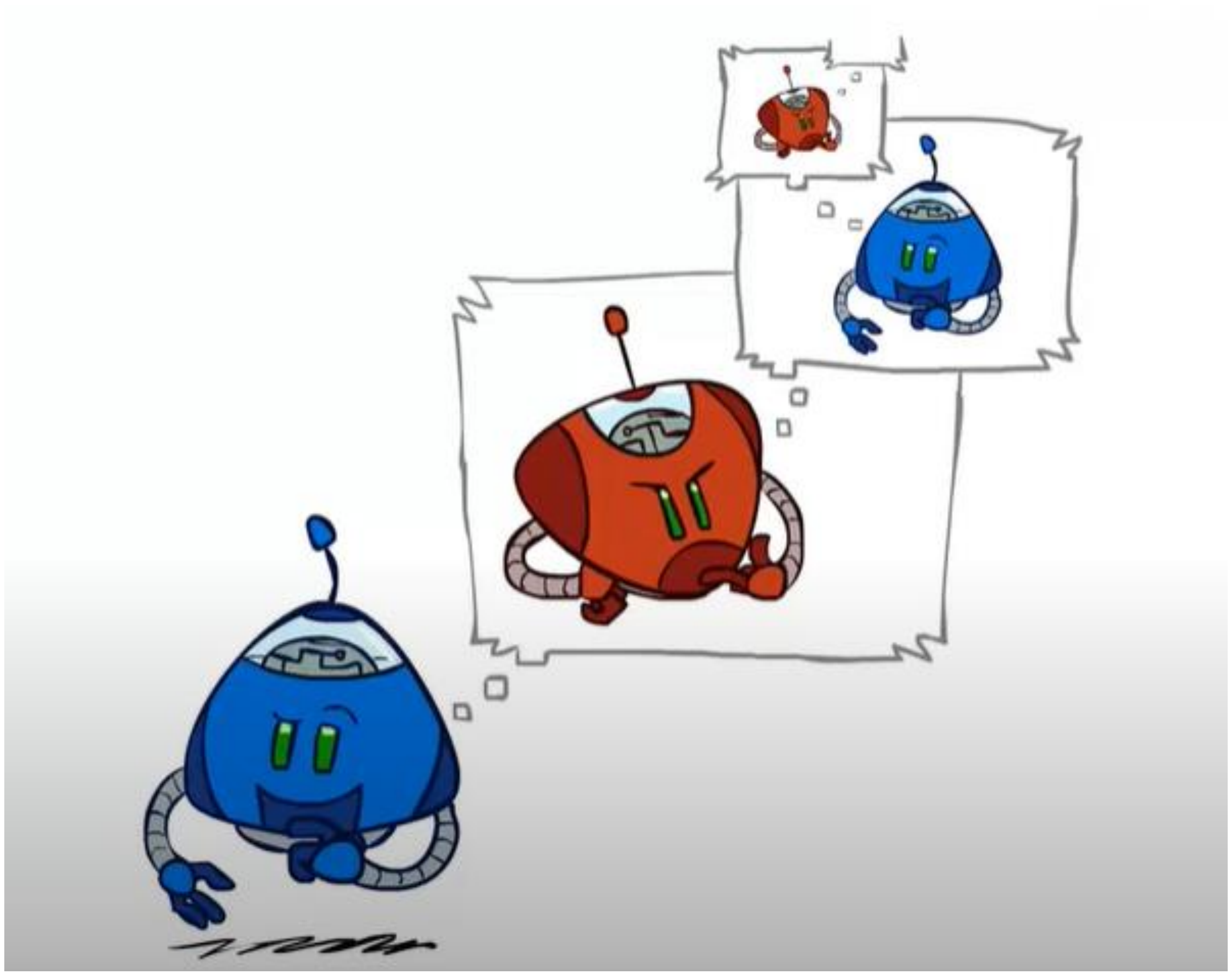


Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value = best achievable utility against best play





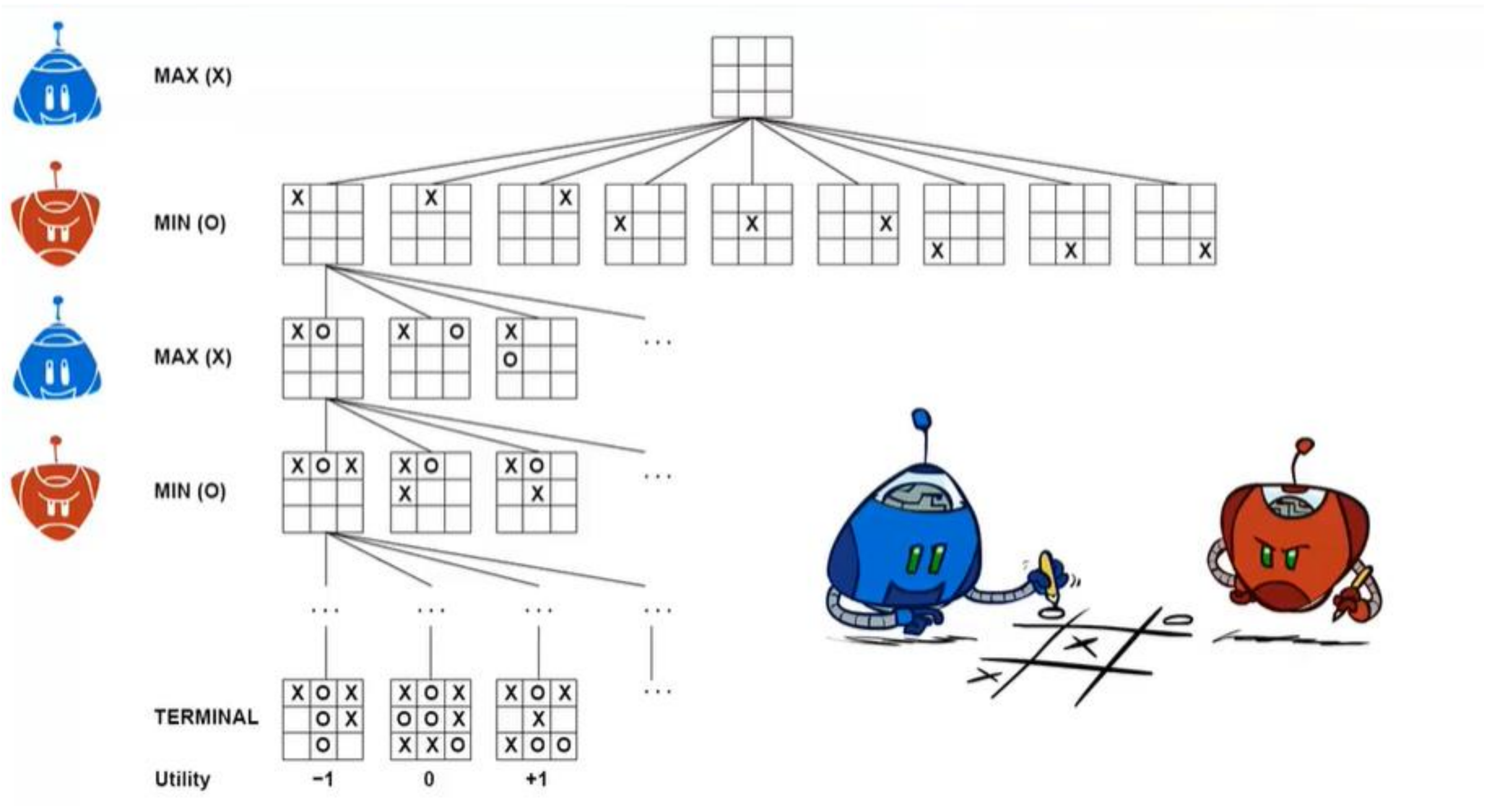
Game Theory (Strategy)

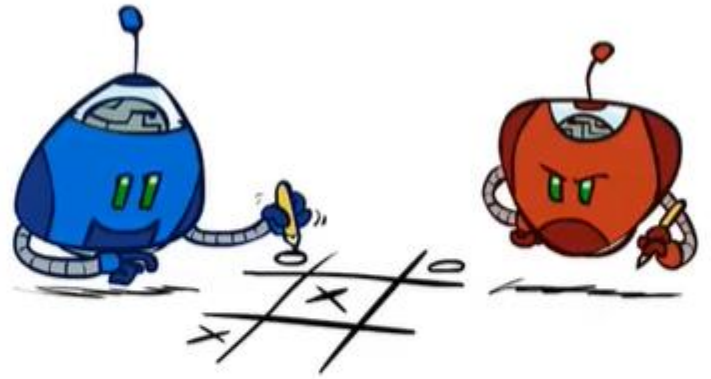
- Game theory does not prescribe a way or say how to play a game.
- Game theory is a set of ideas and techniques for analyzing conflict situations between two or more parties. The outcomes are determined by their decisions.
- **General game theorem** : In every two player, zero sum, perfect knowledge game, there exists a perfect strategy guaranteed to at least result in a tie game.
- The primary game theory is the **Mini-Max Theorem**. This theorem says :
  - **"If a Minimax of one player corresponds to a Maximin of the other player, then that outcome is the best both players can hope for."**

Mini-Max Algorithm

- Mini-max is a decision rule algorithm, which is represented as a game-tree. It has applications in decision-theory, statistics and philosophy.
- Two players: MAX and MIN
  - MAX moves first and they take turns until the game is over. Winner gets award, looser gets penalty.
- By agreement the root of the game-tree represents the max-player.
- It is assumed that each player aims to do the best move for himself and therefore the worst for his opponent in order to win the game.

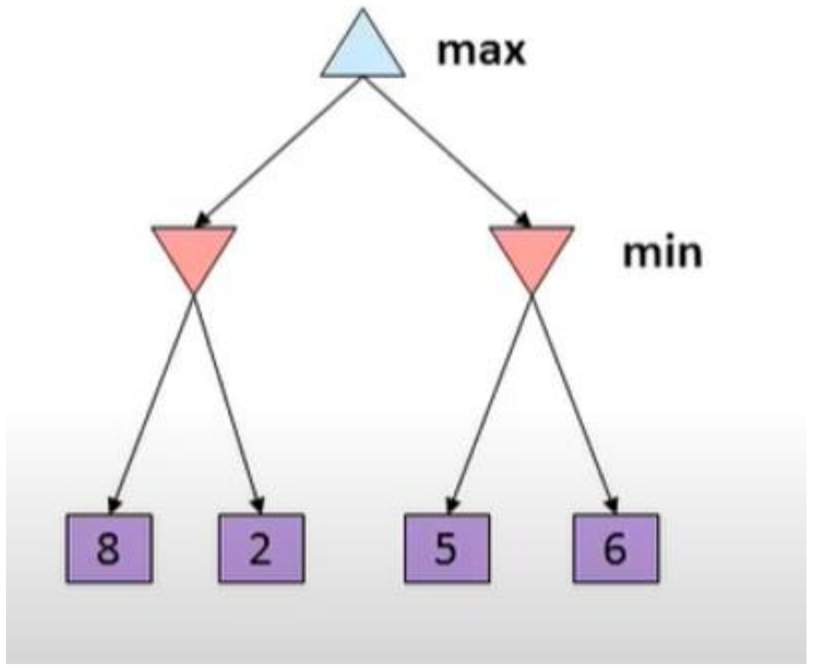
Game Tree





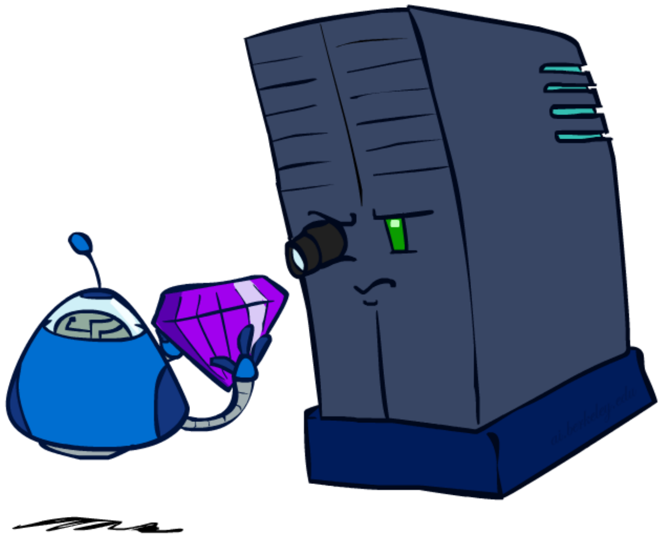
Mini-Max Algorithm

- Deterministic, zero-sum, perfect information, two player
  - Tic-Tac-Toe, chess
  - One player maximizes result
  - The other minimizes result



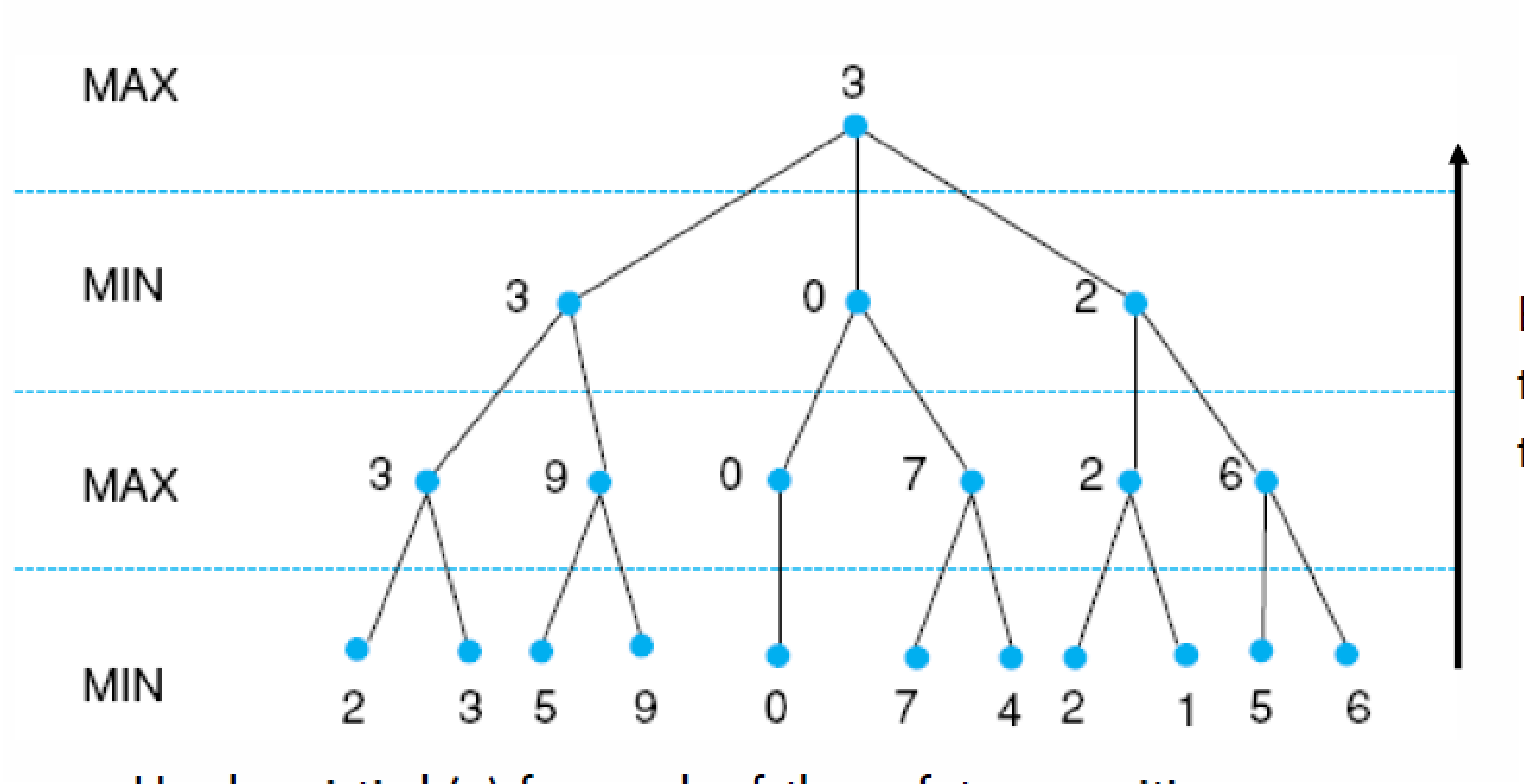
How Mini-Max work?

- Now, we’ll show how the algorithm works. The essence of it is an evaluation function.
- First the game-tree is fully developed to a given number of levels.
- Then an evaluation function is applied to each leaf node. These values represent how good each node is for the max player
- These values are then propagated upwards to get the value for the root node, including the next best mode for the max player.
- The success of the minimax algorithm depends on how accurate the evaluation function is.



Example

Look 3 plies ahead.



Back propagate the scores up the tree.

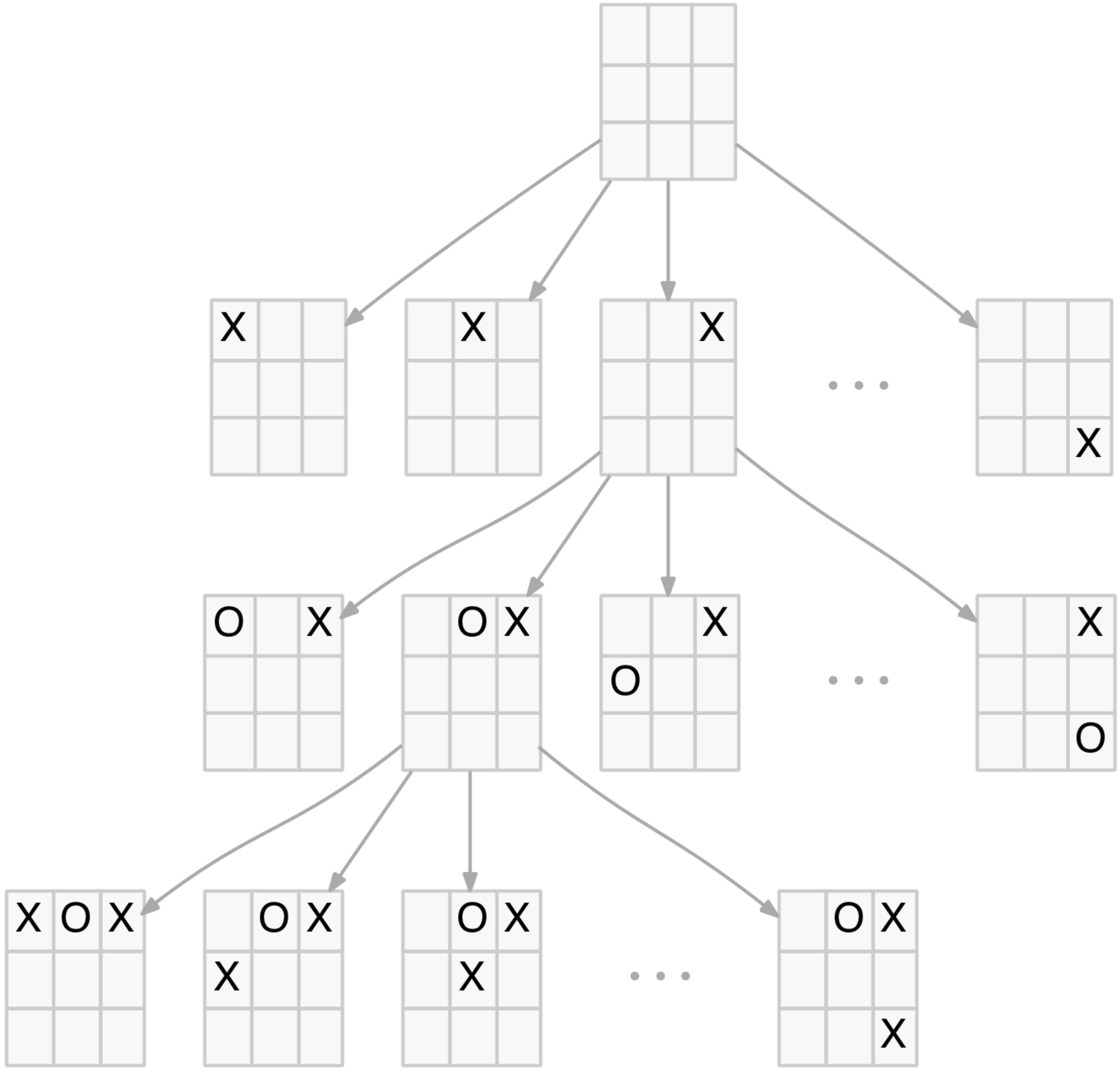
Use heuristic h(n) for each of these future positions.

Evaluation Function

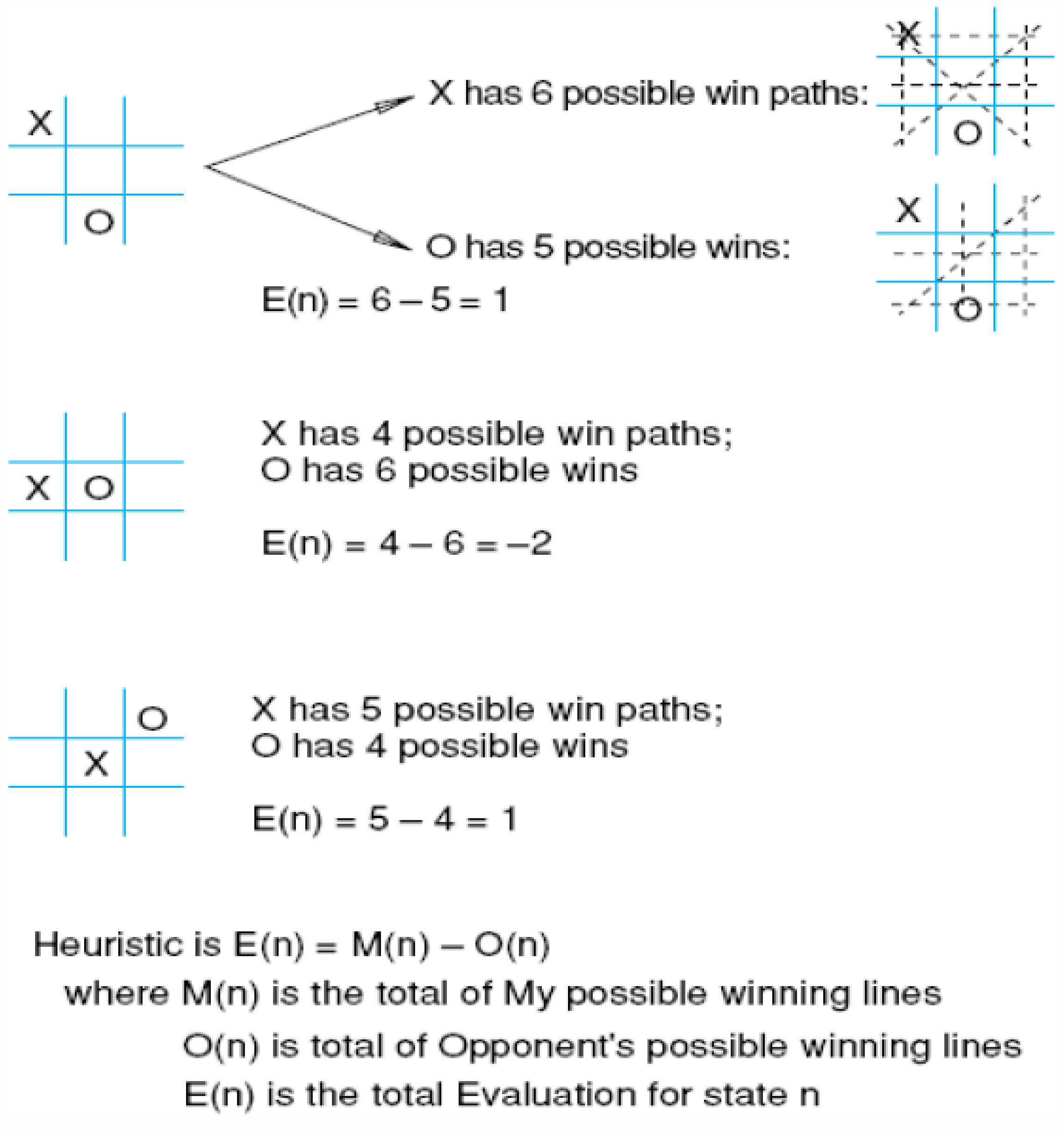
- Evaluations how good a ‘board position’ is
  - Based on static features of that board alone
- Zero-sum assumption lets us use one function to describe goodness for both players.
  - $f(n)>0$ if we are winning in position n
  - $f(n)=0$ if position n is tied
  - $f(n)<0$ if our opponent is winning in position n
- Build using expert knowledge (Heuristic)

Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters



Evaluation Function (Tic-Tac-Toe)



X has 6 possible win paths:

O has 5 possible wins:

$E(n) = 6 - 5 = 1$

X has 4 possible win paths;
O has 6 possible wins

$E(n) = 4 - 6 = -2$

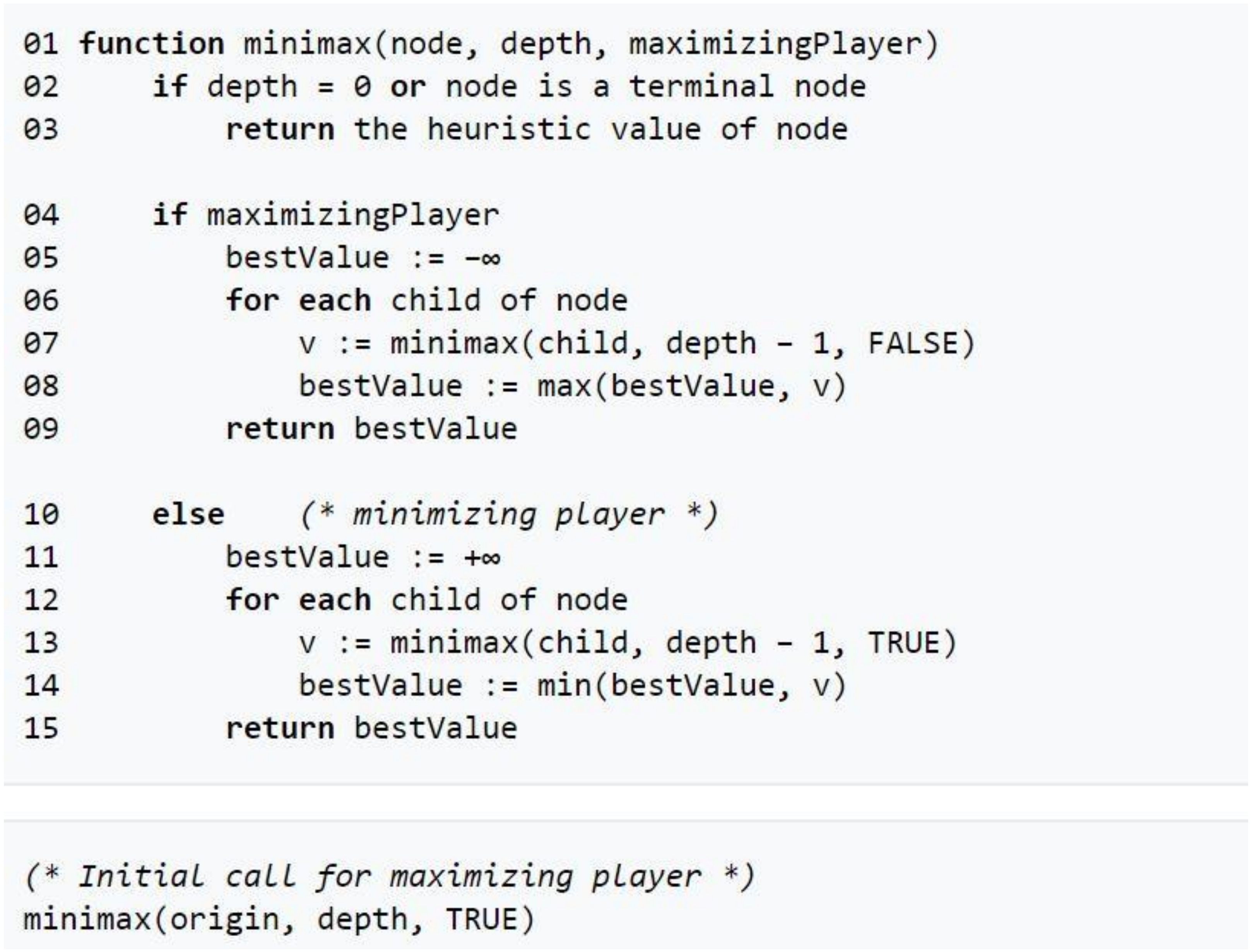
X has 5 possible win paths;
O has 4 possible wins

$E(n) = 5 - 4 = 1$

Heuristic is $E(n) = M(n) - O(n)$
where $M(n)$ is the total of My possible winning lines
$O(n)$ is total of Opponent's possible winning lines
$E(n)$ is the total Evaluation for state n

E(n) = 0 when my opponent and I have equal number of possibilities.

Mini-Max Pseudocode

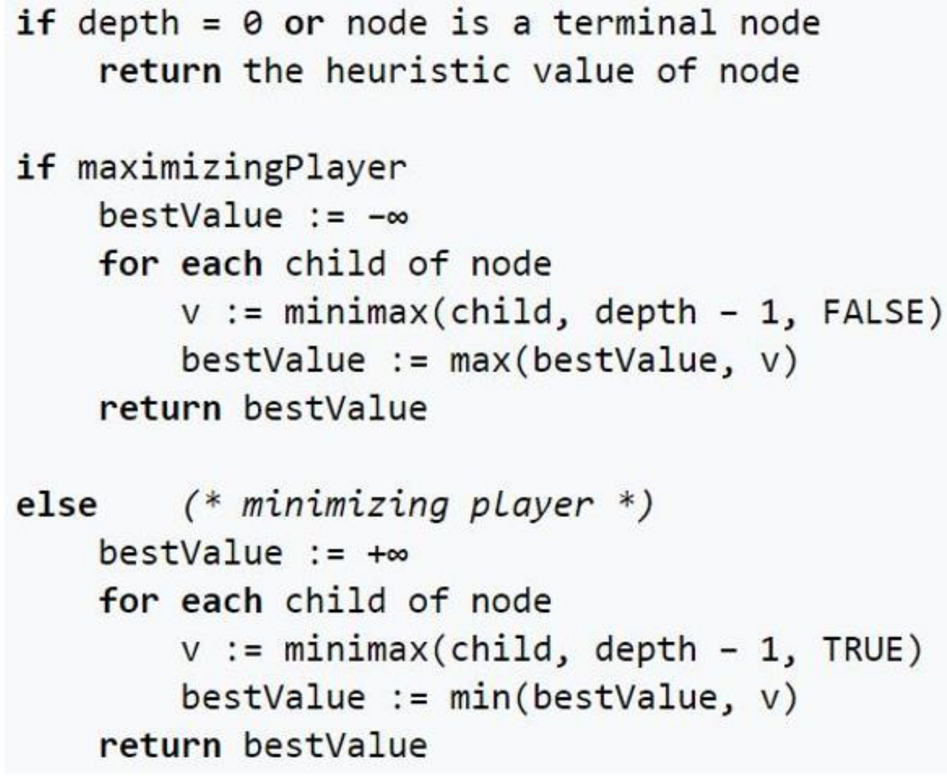


```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node

    if maximizingPlayer
        bestValue := −∞
        for each child of node
            v := minimax(child, depth − 1, FALSE)
            bestValue := max(bestValue, v)
        return bestValue

    else    (* minimizing player *)
        bestValue := +∞
        for each child of node
            v := minimax(child, depth − 1, TRUE)
            bestValue := min(bestValue, v)
        return bestValue
```

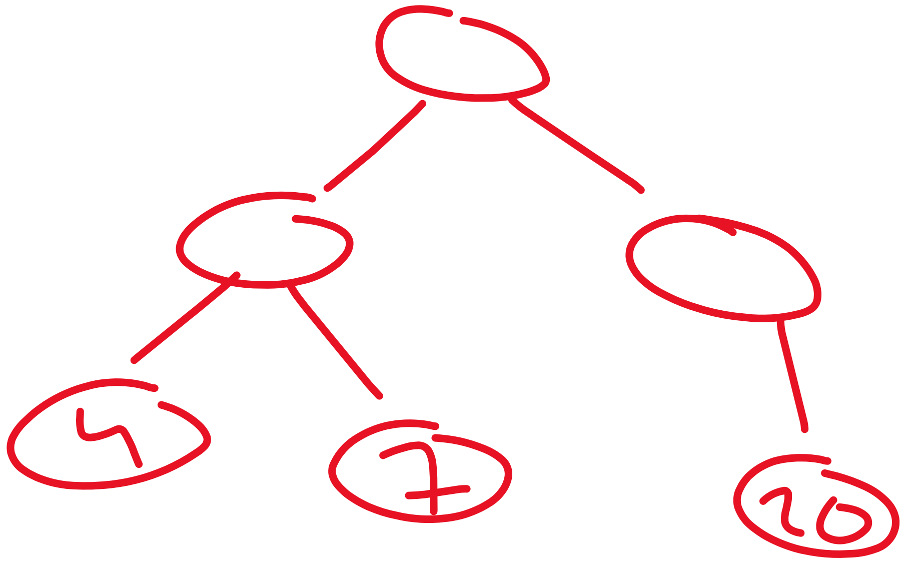
```
(* Initial call for maximizing player *)
minimax(origin, depth, TRUE)
```



```
if depth = 0 or node is a terminal node
    return the heuristic value of node

if maximizingPlayer
    bestValue := −∞
    for each child of node
        v := minimax(child, depth − 1, FALSE)
        bestValue := max(bestValue, v)
    return bestValue

else    (* minimizing player *)
    bestValue := +∞
    for each child of node
        v := minimax(child, depth − 1, TRUE)
        bestValue := min(bestValue, v)
    return bestValue
```

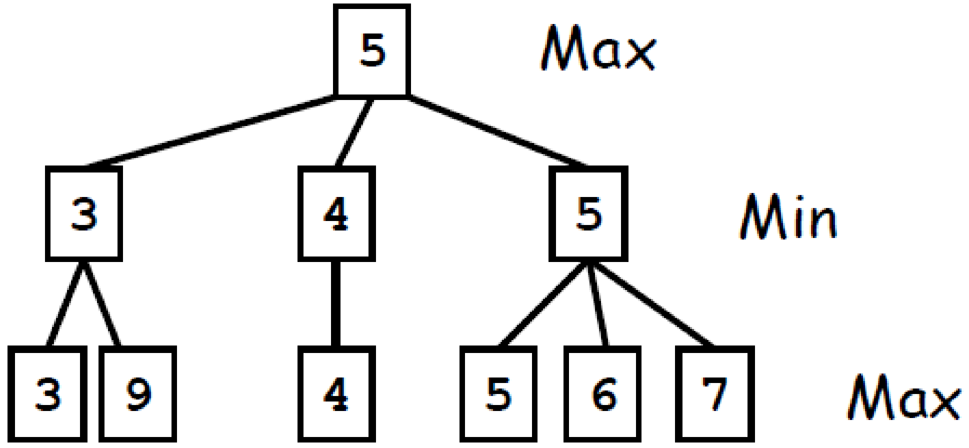


Example

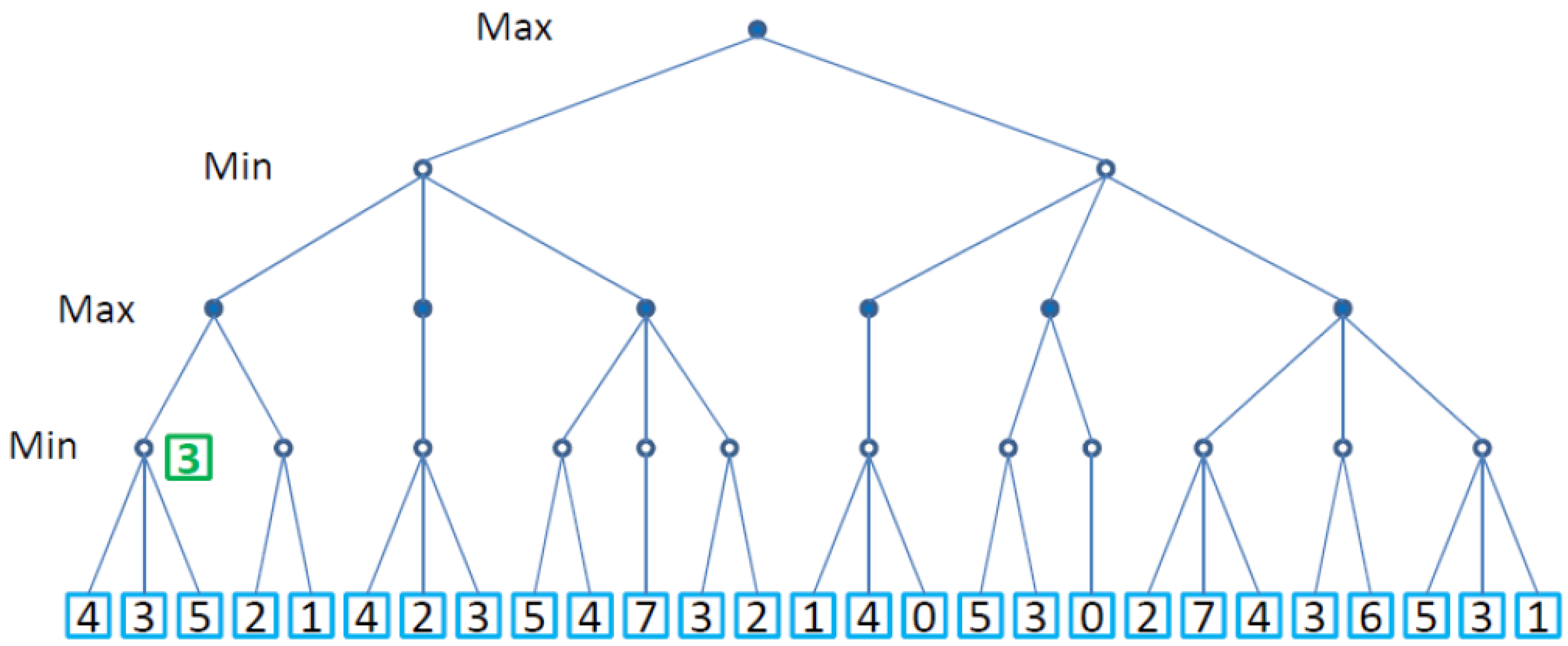
- Max's turn
- Would like the "9" points (the maximum)
- But if choose left branch, Min will choose move to get 3

  → left branch has a value of 3
- If choose right, Min can choose any one of 5, 6 or 7 (will choose 5, the minimum)

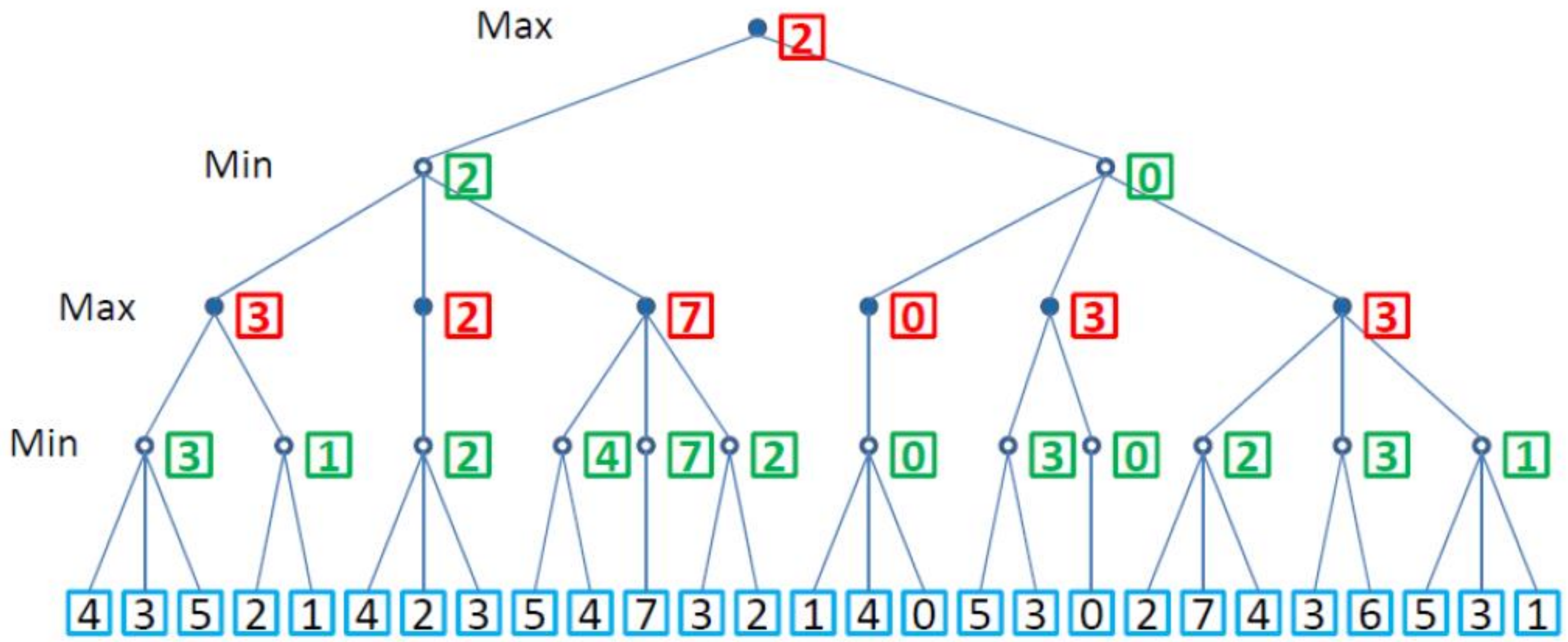
  → right branch has a value of 5
- Right branch is largest (the maximum) so choose that move



Example 2



Example 2


Max
2
Min
2
0
Max
3
2
7
0
3
3
Min
3
1
2
4
7
2
0
3
0
2
3
1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

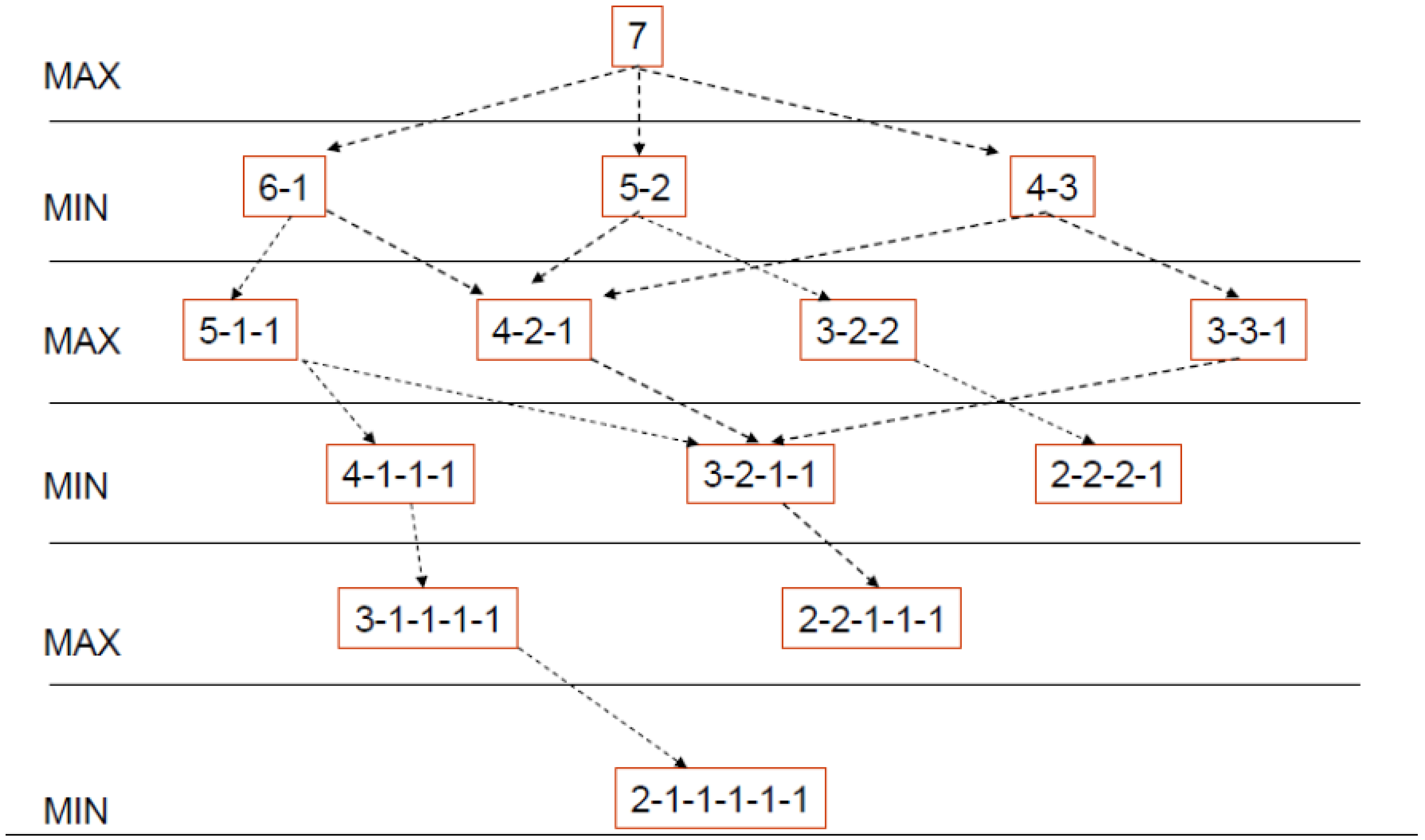
Game NIM

- Given N rods in n (n=1) heaps, two players, take turns
- At each turn, player has to select one heap and separate it into two different heaps having different number of rods
- A player looses the game if he/she can not select a heap for separation.

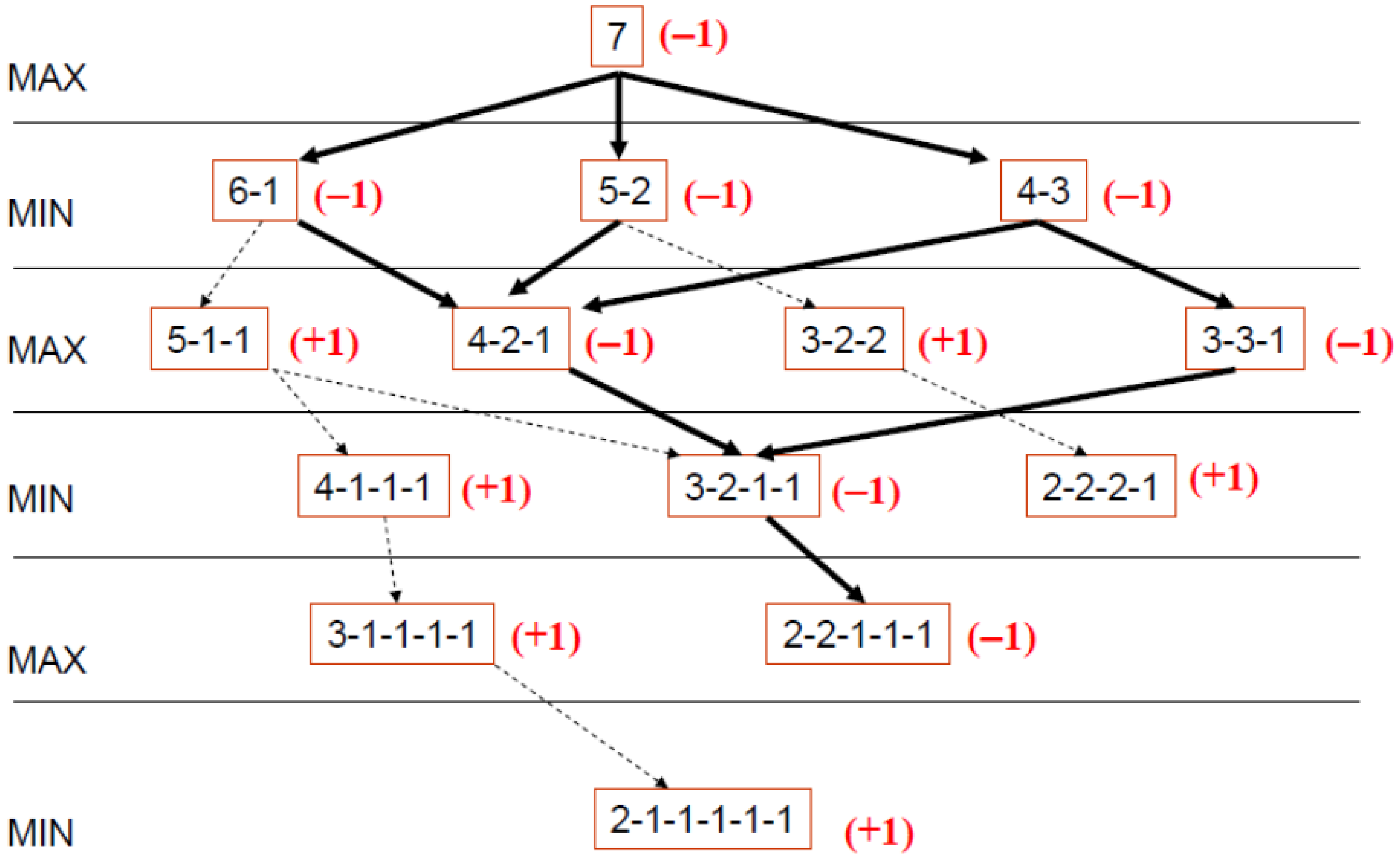
Game NIM

- Is this a zero sum game? Is this a game where each player has perfect information?
  - This is a zero-sum game since one player wins and the other loses. This is a perfect information game since both players always have all of the relevant knowledge of the game's state.

- Draw the complete game tree or graph when the initial pile has seven stones and using it, determine what the outcome is (e.g., win for player one, win for player two, draw, unknown) if both players play optimally.



MAX
MIN
MAX
MIN
MAX
MIN
7
6-1
5-2
4-3
5-1-1
4-2-1
3-2-2
3-3-1
4-1-1-1
3-2-1-1
2-2-2-1
3-1-1-1-1
2-2-1-1-1
2-1-1-1-1-1



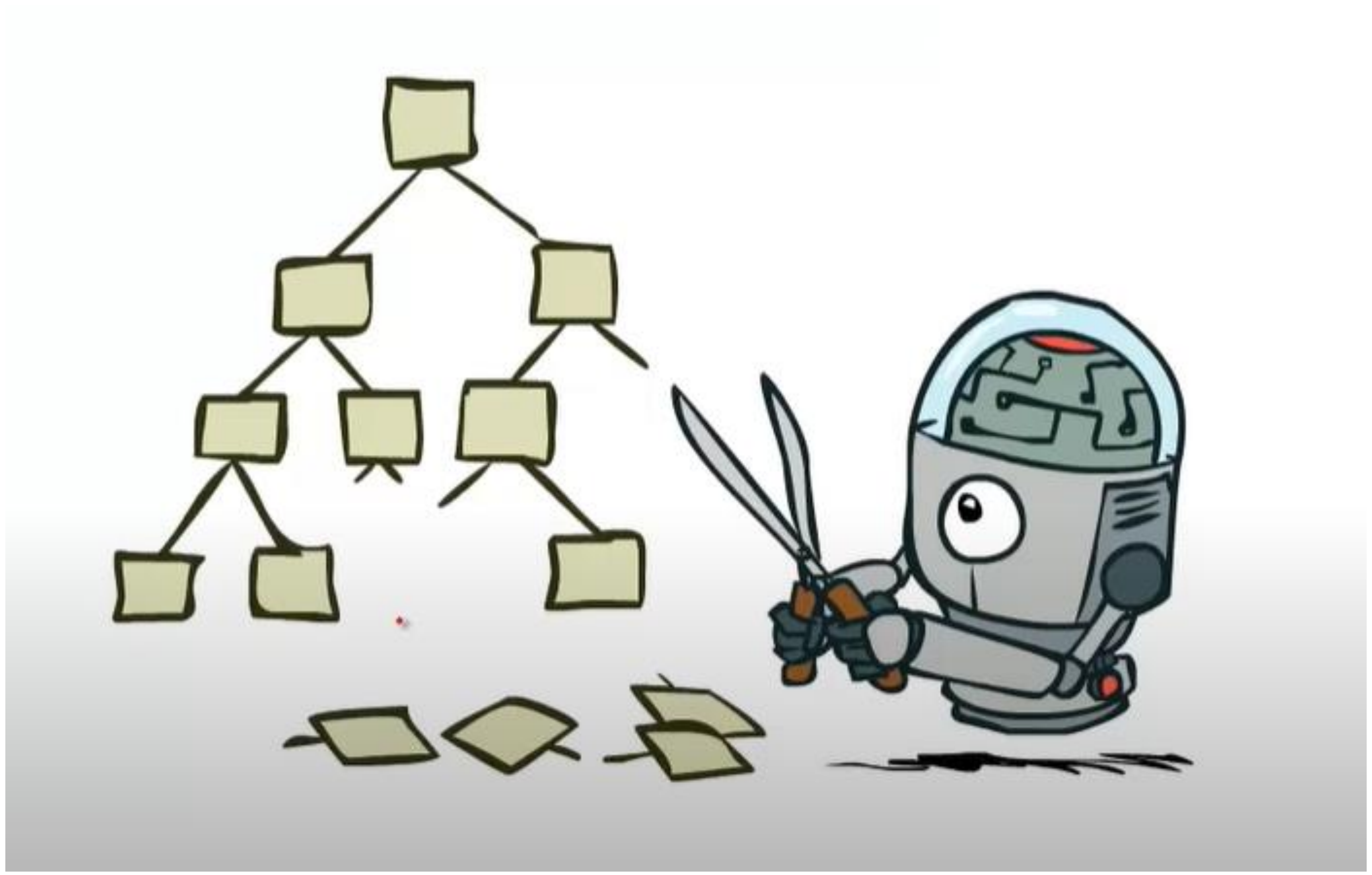
MAX
MIN
MAX
MIN
MAX
MIN
7 (–1)
6-1 (–1)
5-2 (–1)
4-3 (–1)
5-1-1 (+1)
4-2-1 (–1)
3-2-2 (+1)
3-3-1 (–1)
4-1-1-1 (+1)
3-2-1-1 (–1)
2-2-2-1 (+1)
3-1-1-1-1 (+1)
2-2-1-1-1 (–1)
2-1-1-1-1-1 (+1)

Mini-Max Analysis

- Time Complexity
- Space Complexity

Mini-Max Analysis

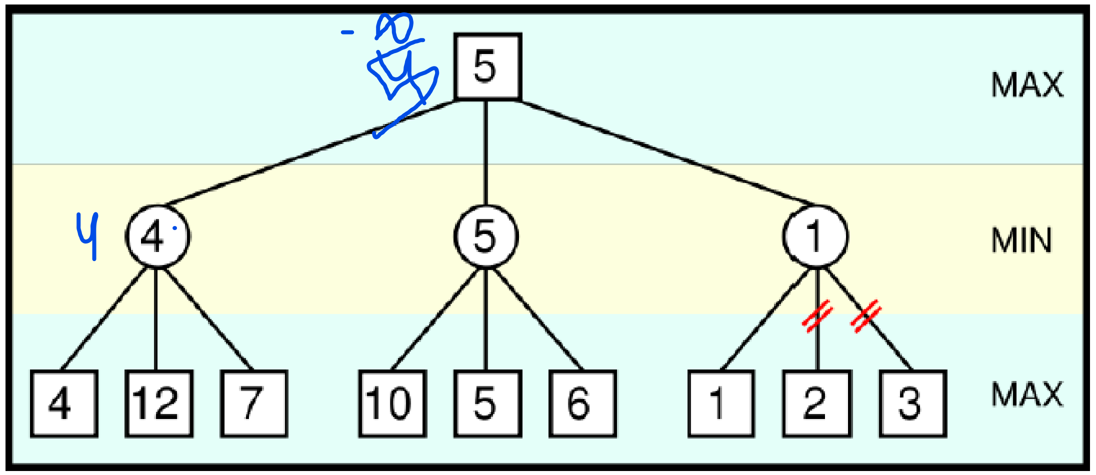
- Can we do betters? How?



Alpha-Beta Pruning

Alpha-Beta Pruning

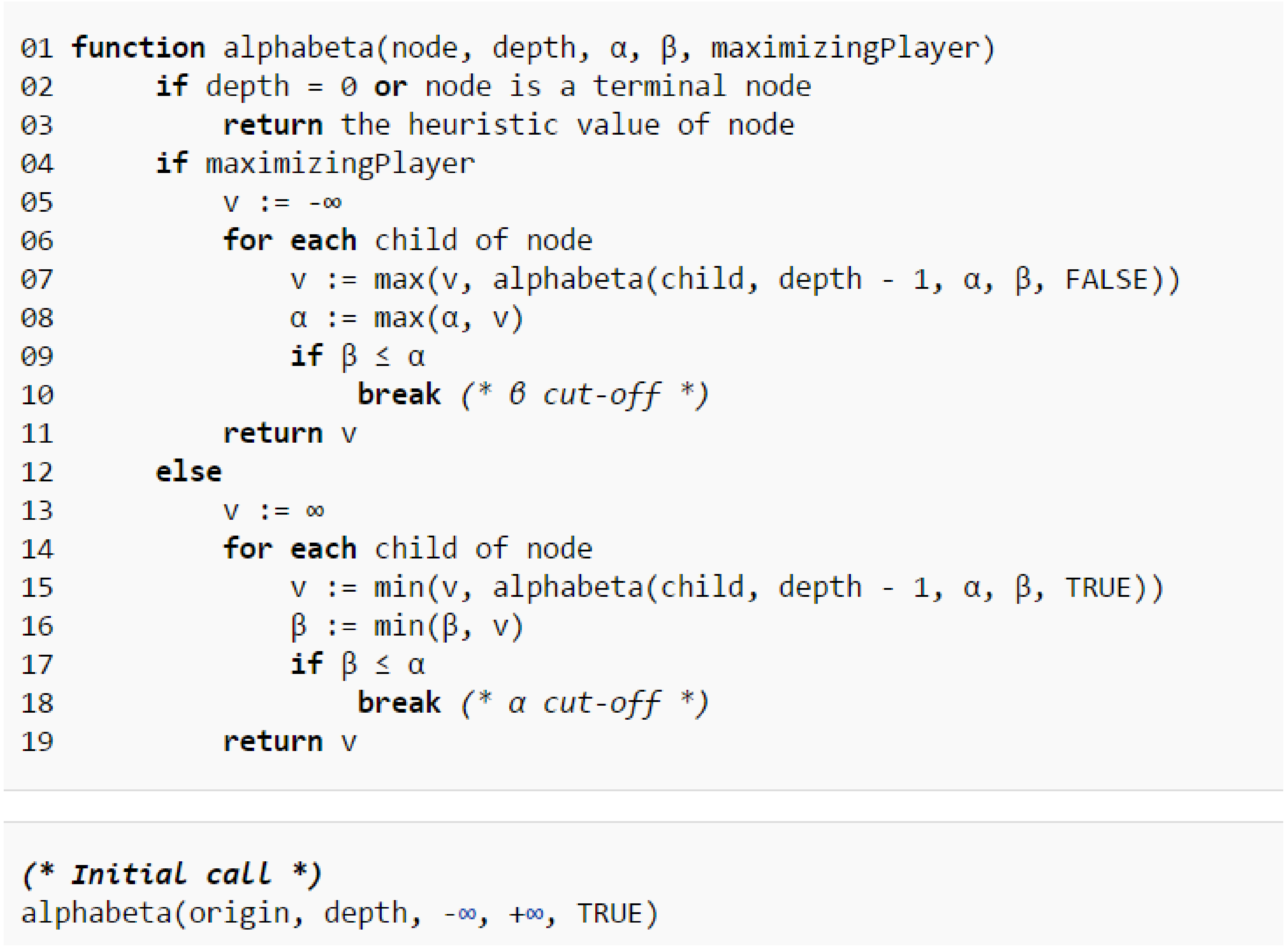
- A method that can often cut off a half the game tree.
- Based on the idea that if a move is clearly bad, there is no need to follow the consequences of it.
  - alpha – highest value we have found so far
  - beta – lowest value we have found so far



Alpha-Beta Pruning Pseudocode

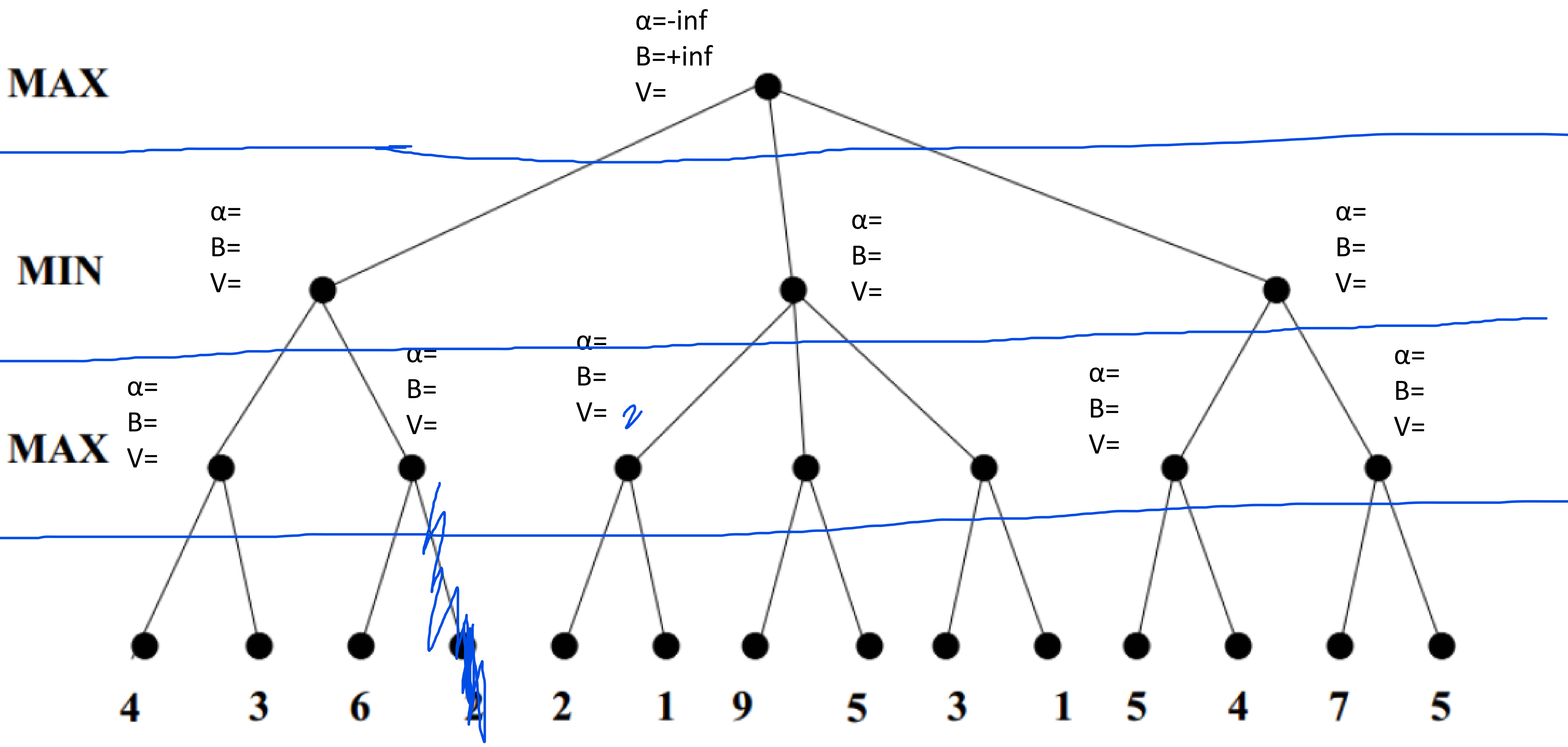
$\alpha = 4$

$\beta = 4$



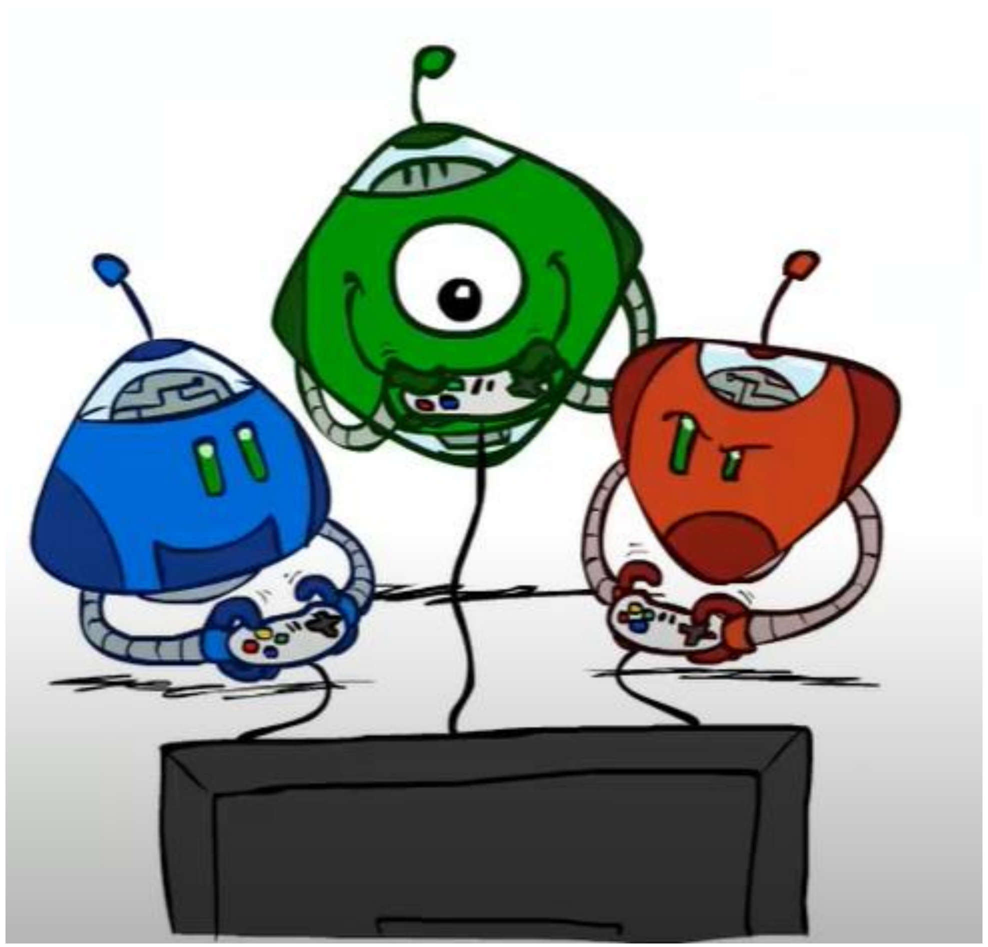
```
function alphabeta(node, depth, α, β, maximizingPlayer)
     if depth = 0 or node is a terminal node
         return the heuristic value of node
     if maximizingPlayer
         v := -∞
         for each child of node
             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
             α := max(α, v)
             if β ≤ α
                 break (* β cut-off *)
         return v
     else
         v := ∞
         for each child of node
             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
             β := min(β, v)
             if β ≤ α
                 break (* α cut-off *)
         return v
```

```
(* Initial call *)
alphabeta(origin, depth, -∞, +∞, TRUE)
```


MAX
MIN
MAX
α=-inf
B=+inf
V=
α=
B=
V=
α=
B=
V=
α=
B=
V=
α=
B=
V=
α=
B=
V=
α=
B=
V=
α=
B=
V=
α=
B=
V= 2
α=
B=
V=
α=
B=
V=
4
3
6
2
2
1
9
5
3
1
5
4
7
5

Alpha-Beta Analysis

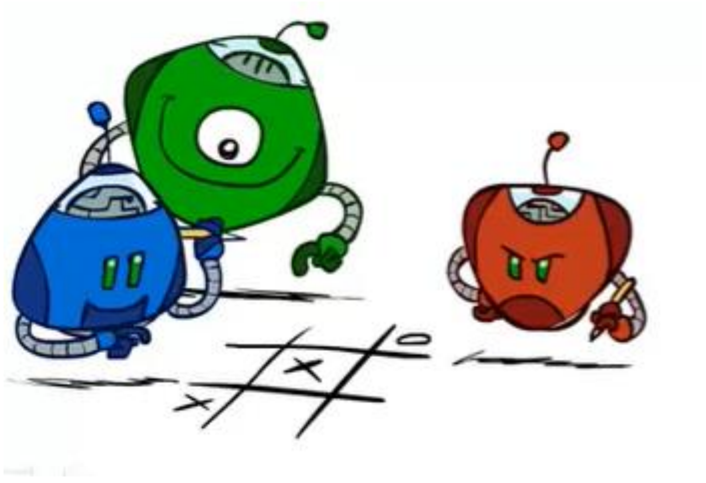
- Without pruning, need to examine $O(b^m)$ nodes
- With pruning, depends on which nodes we consider first
- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- If we manage to choose the best successor first, need to examine $O(b^{m/2})$ nodes
  - Practical heuristics for choosing next successor to consider get quite close to this

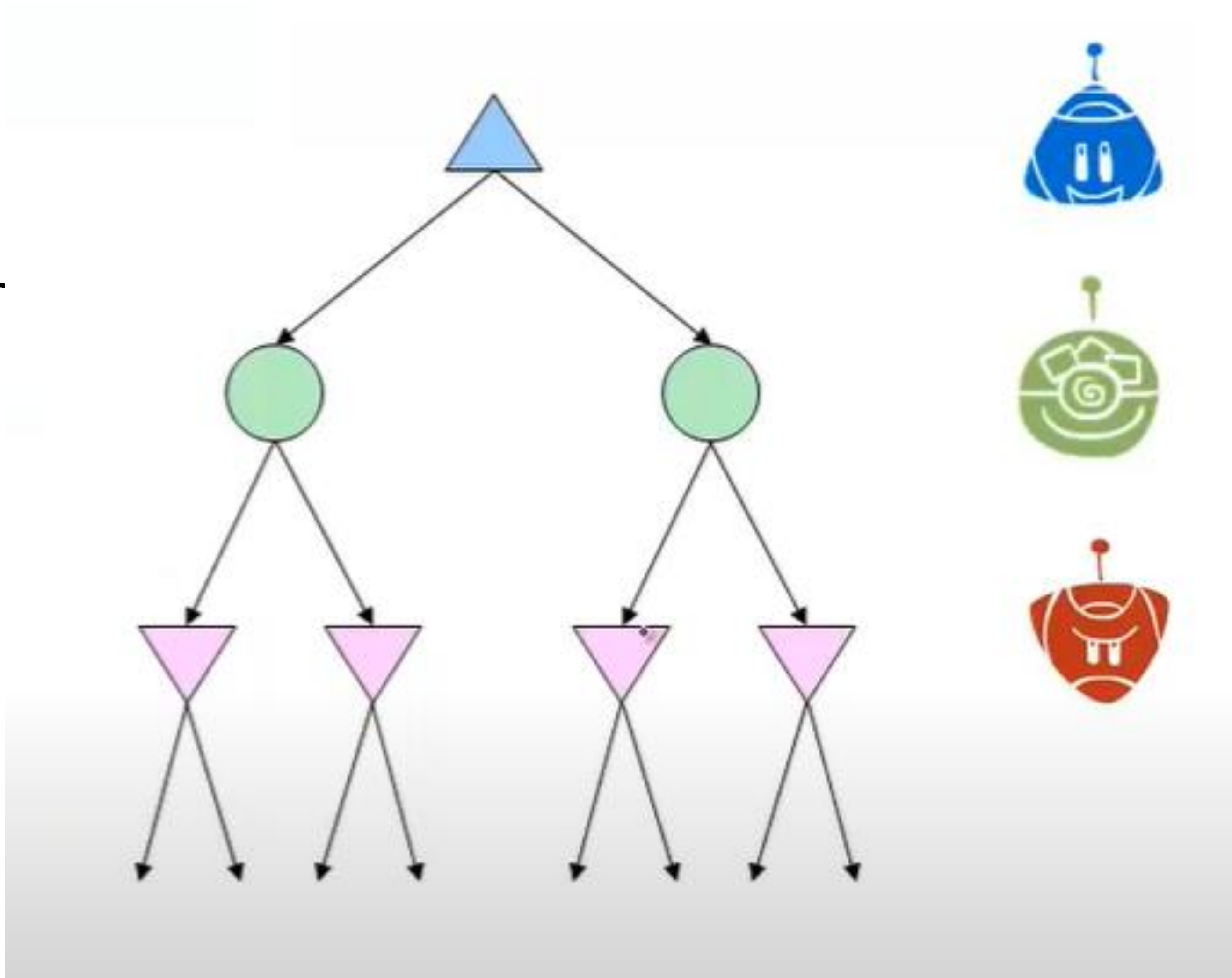


Multi-Agent Utilities

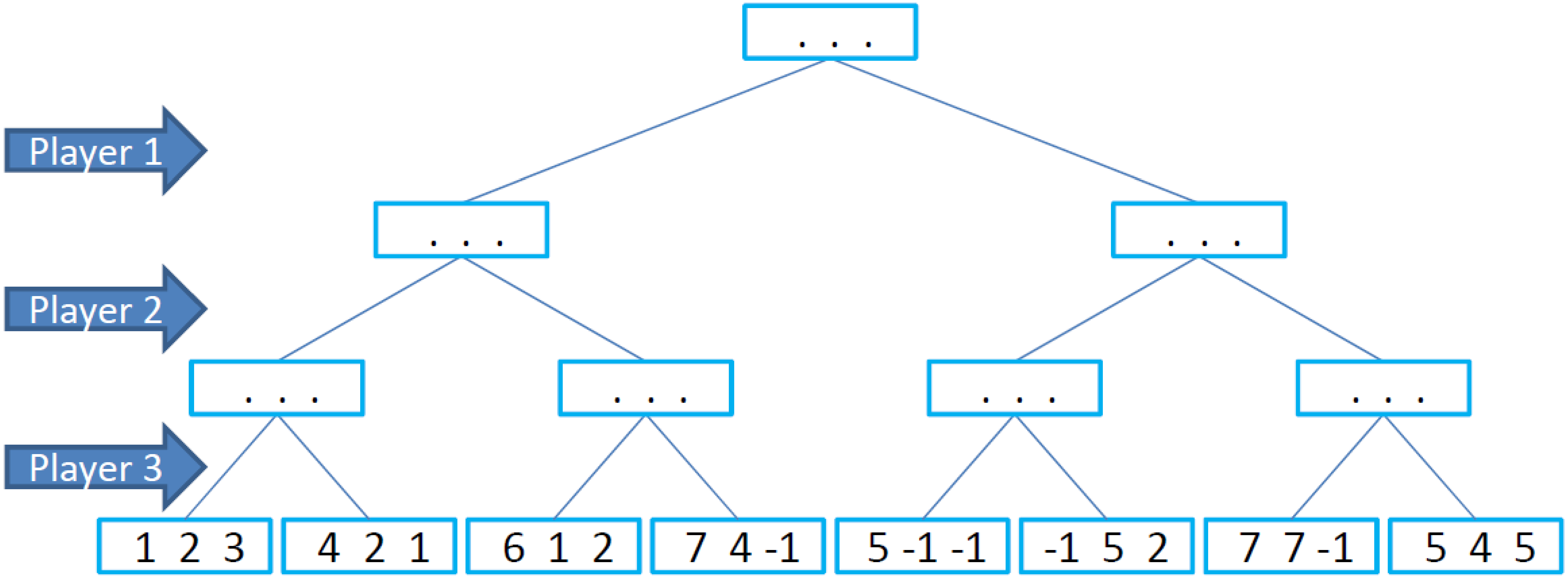
Multi-Agent Utilities

- What if the game has multiple player?
- Generalization of minimax:
  - Evaluation function given by vector
  - All players are Max

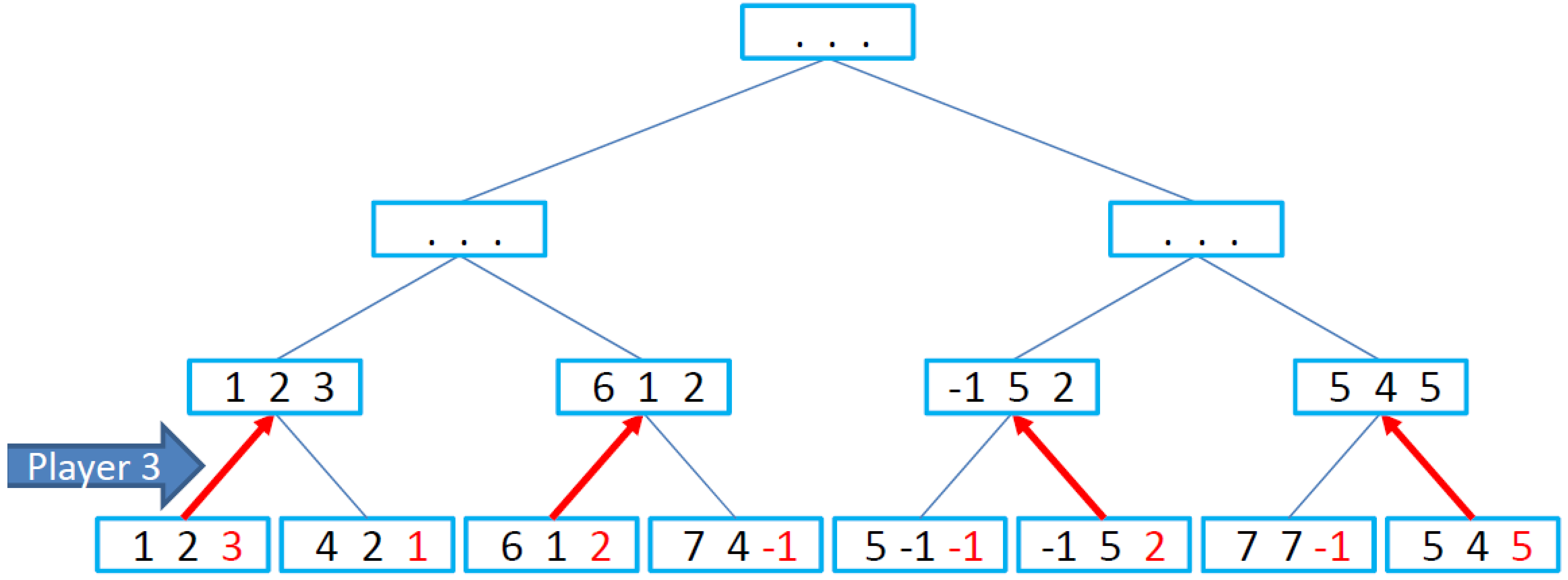




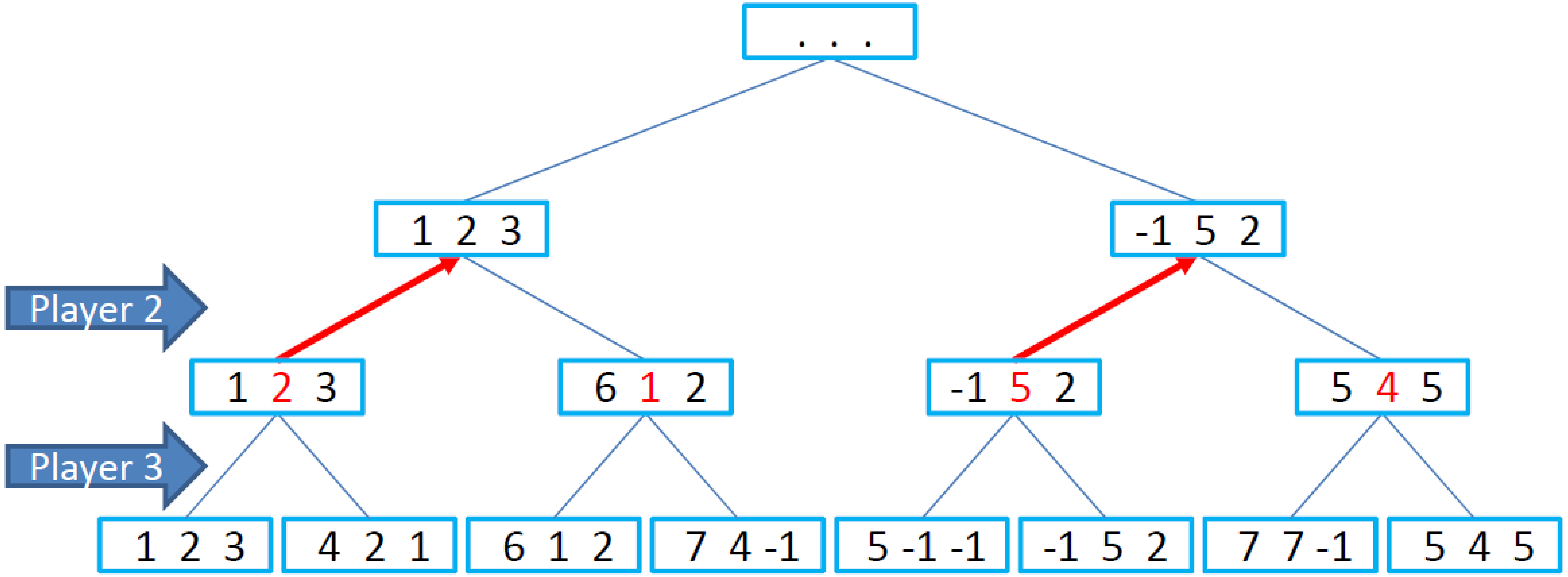
- Each layer assigned to 1 player
- Turn: every 3 layers



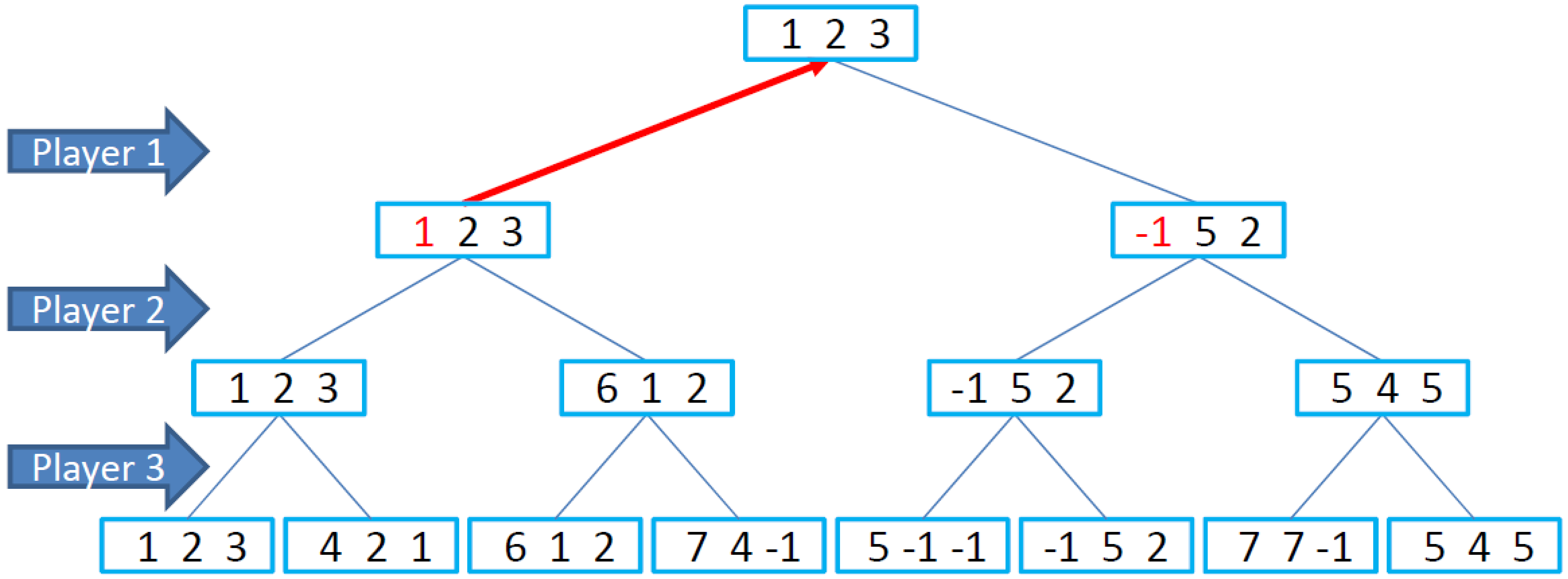
- MaxThirdPlayer([1,2,3],[4,2,1]) = [1,2,3]
- MaxThirdPlayer([6,1,2],[7,4,-1]) = [6,1,2]
- MaxThirdPlayer([5,-1,-1],[-1,5,2]) = [-1,5,2]
- MaxThirdPlayer([7,7,-1],[5,4,5]) = [5,4,5]



- MaxSecondPlayer([1,2,3],[6,1,2]) = [1,2,3]
- MaxSecondPlayer([-1,5,2],[5,4,5]) = [-1,5,2]



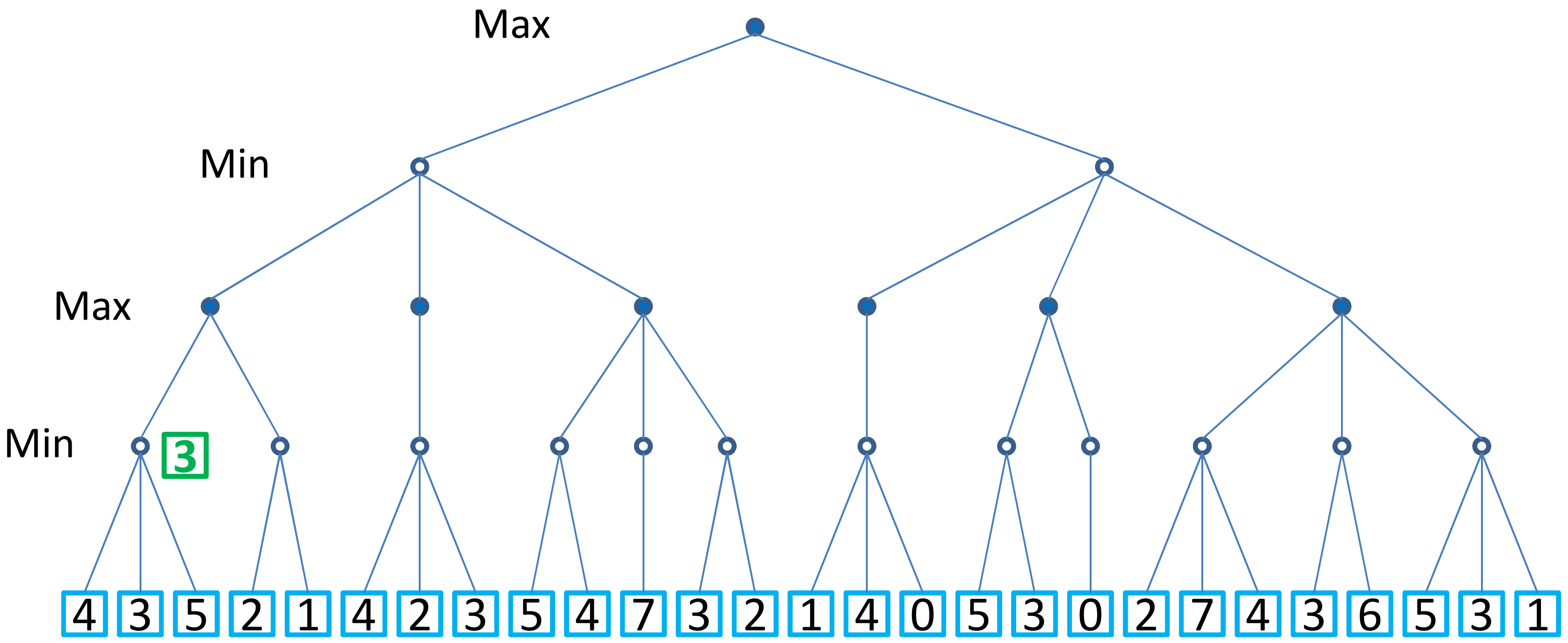
- MaxFirstPlayer([1,2,3],[-1,5,4]) = [1,2,3]



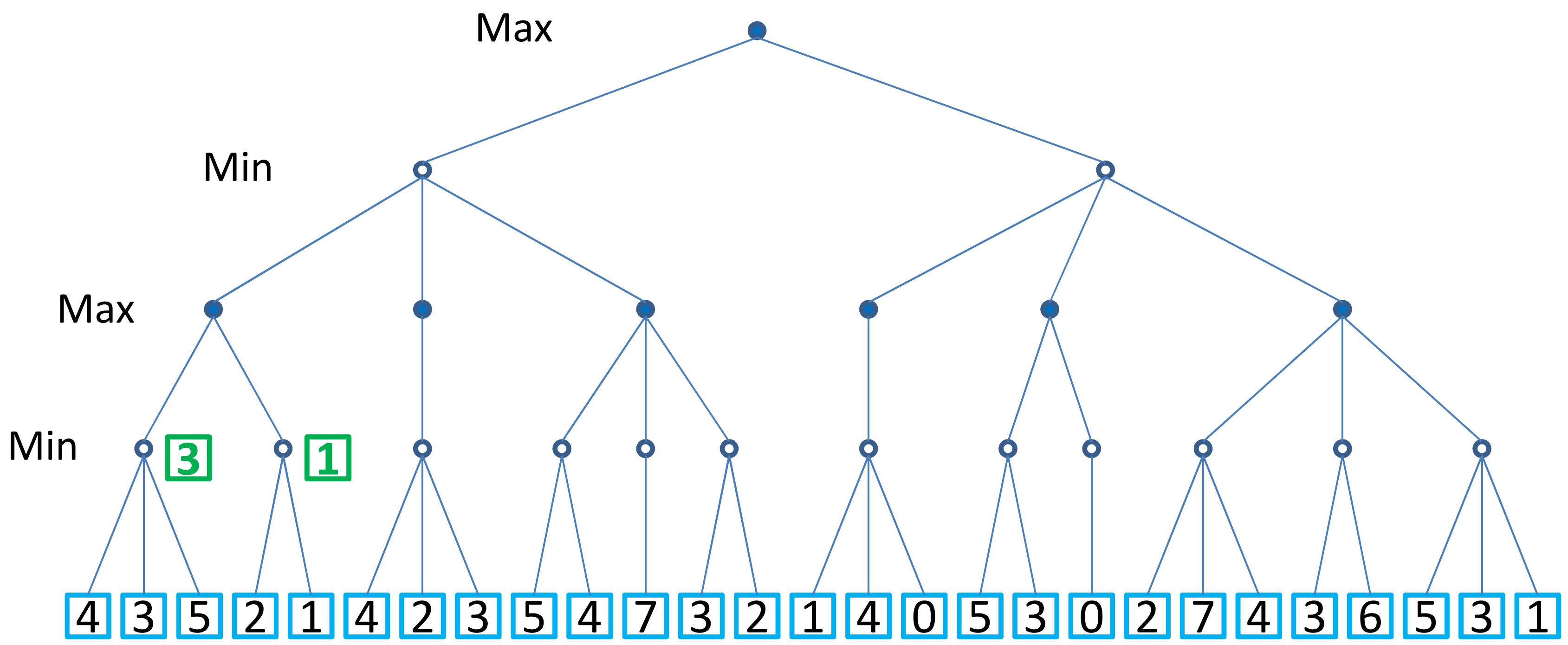
MiniMax & Constraint Processing: MiniMax Algorithm

MINIMAX WITHOUT $\alpha\beta$-PRUNING

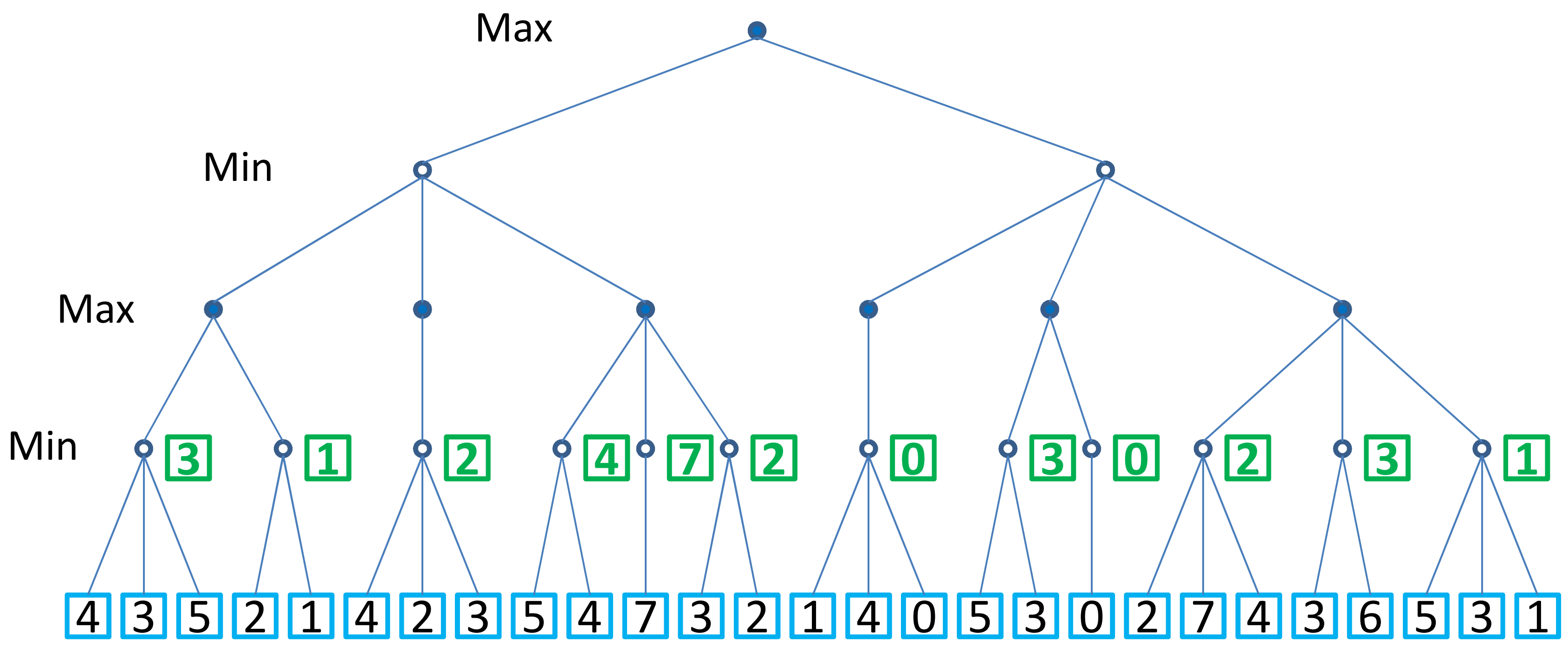
MiniMax without $\alpha\beta$-pruning



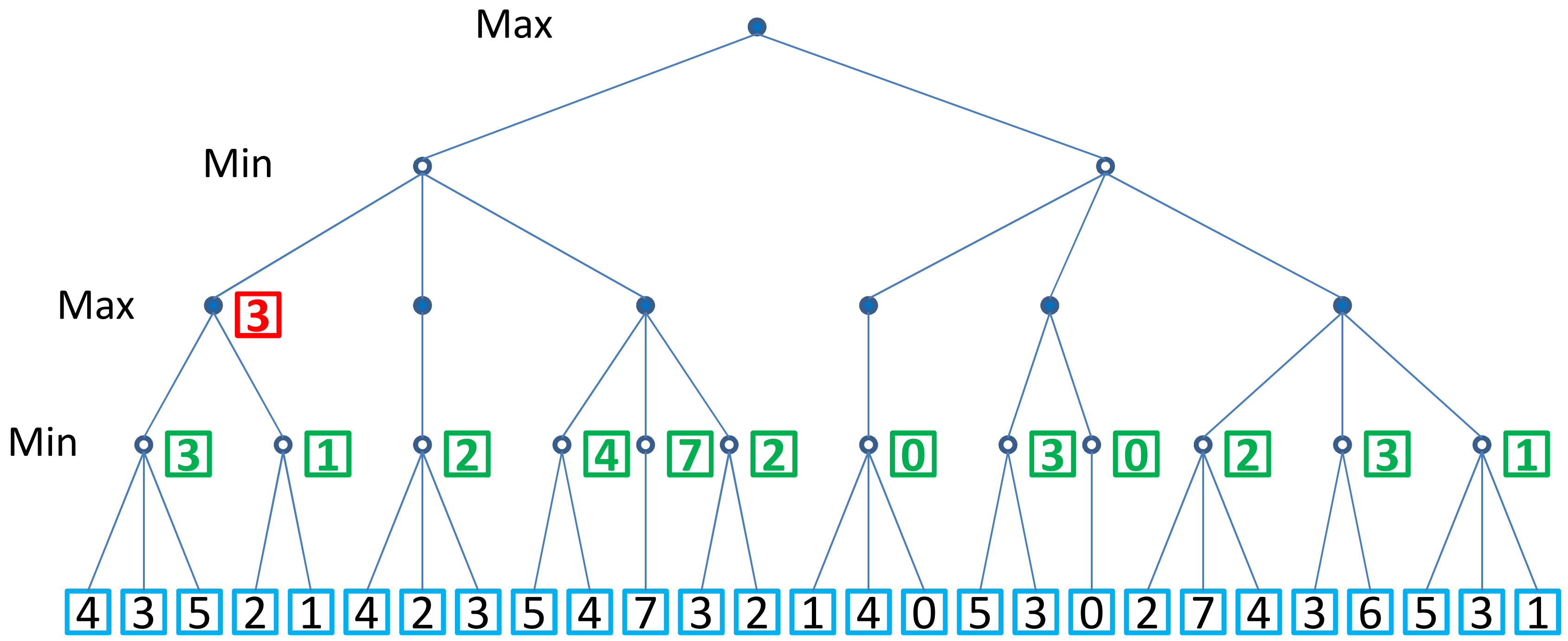
MiniMax without $\alpha\beta$-pruning



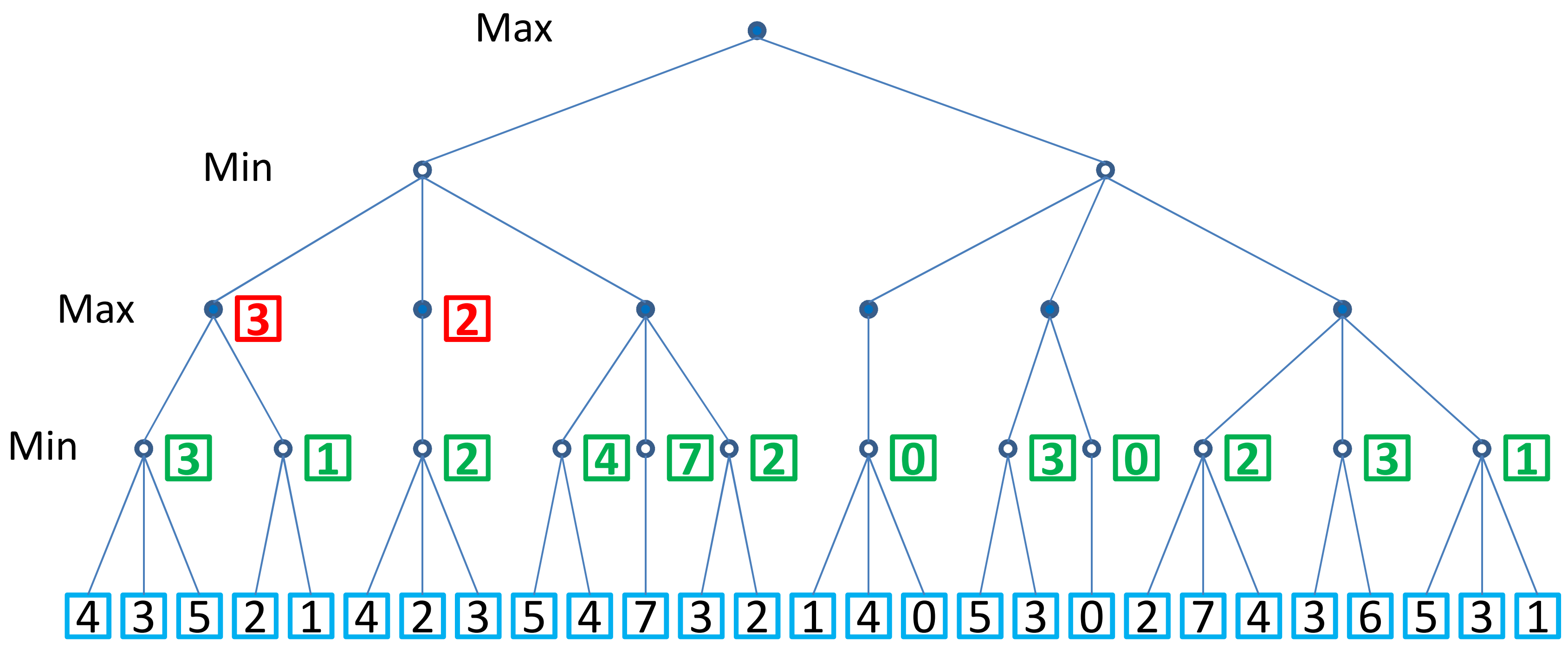
MiniMax without $\alpha\beta$-pruning



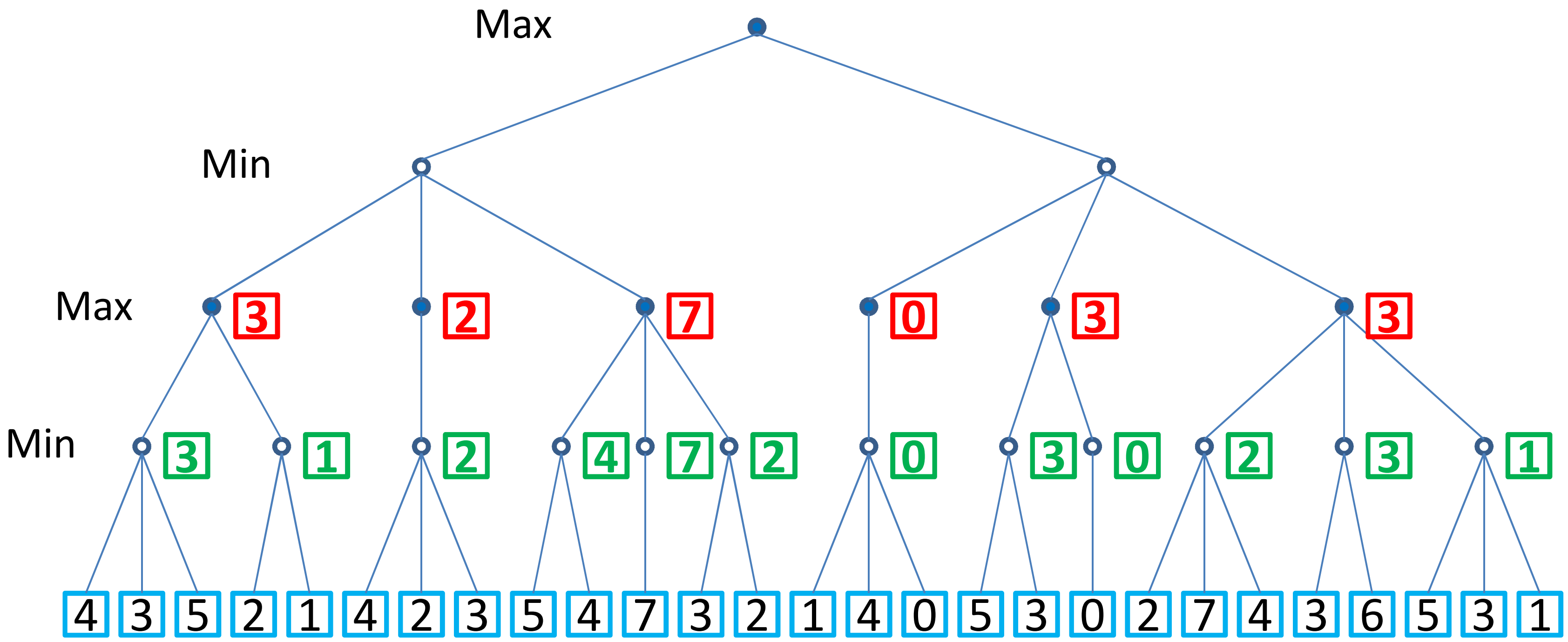
MiniMax without $\alpha\beta$-pruning



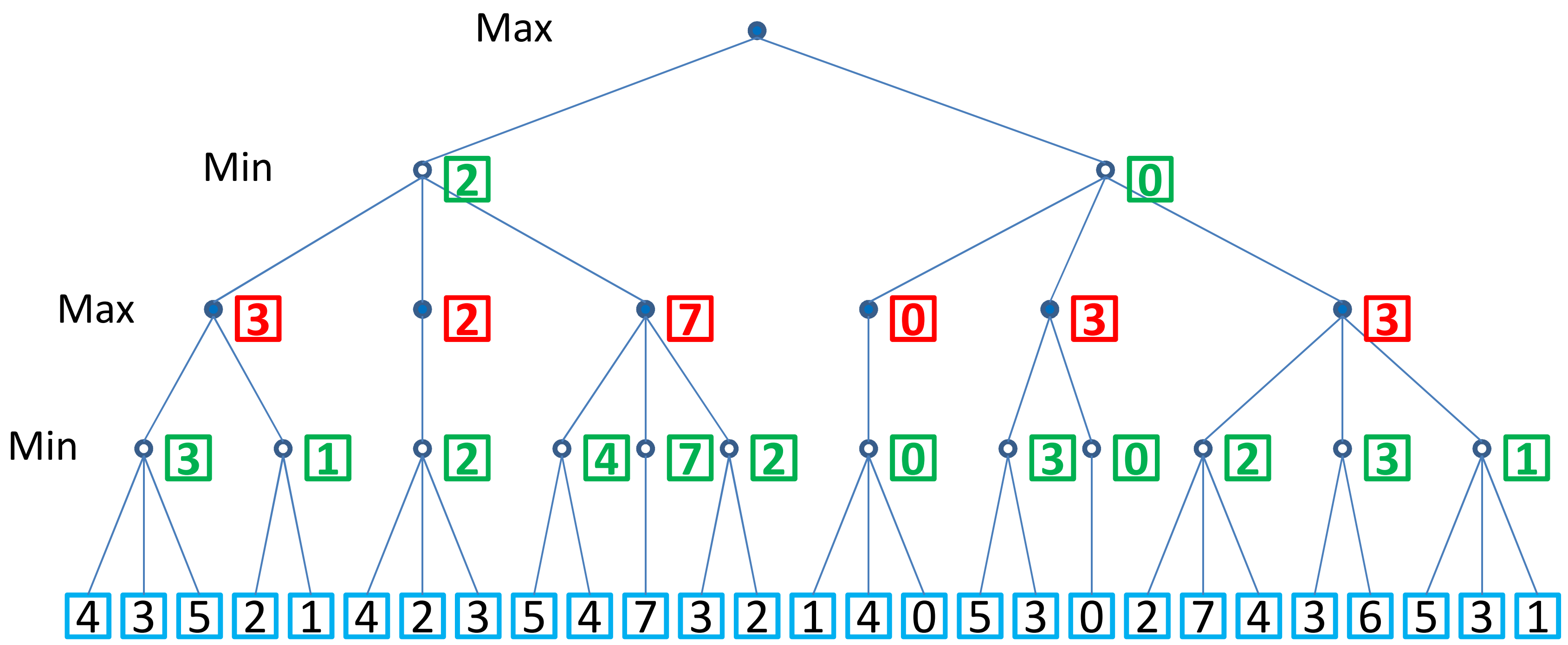
MiniMax without $\alpha\beta$-pruning



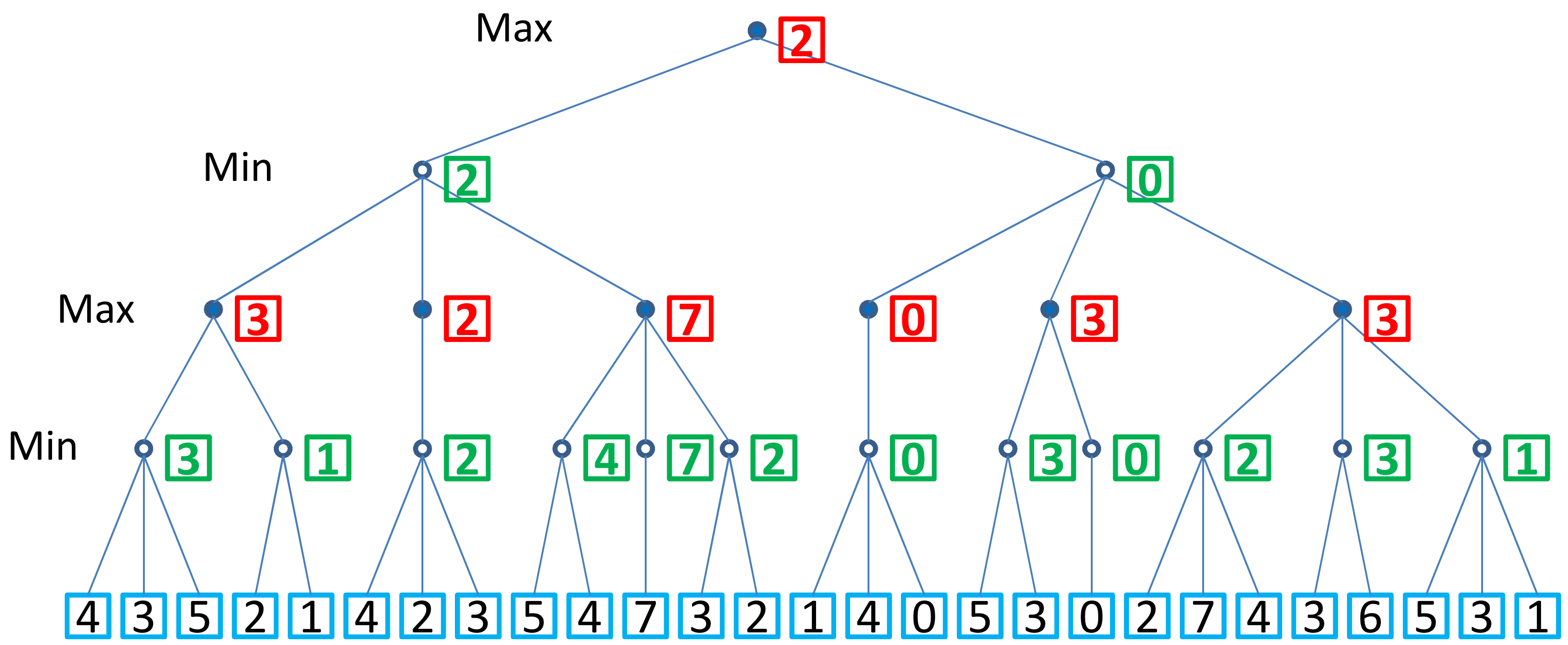
MiniMax without $\alpha\beta$-pruning



MiniMax without $\alpha\beta$-pruning



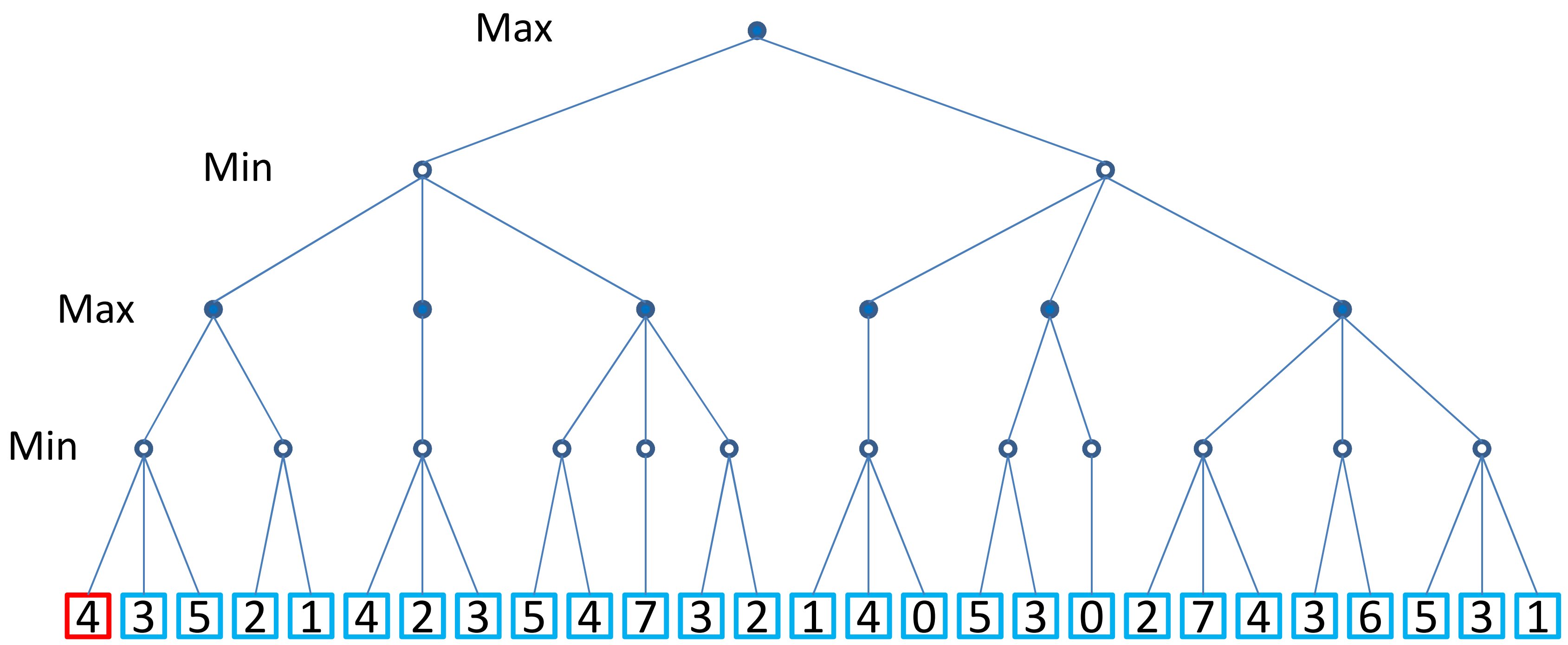
MiniMax without $\alpha\beta$-pruning



MiniMax & Constraint Processing: MiniMax Algorithm

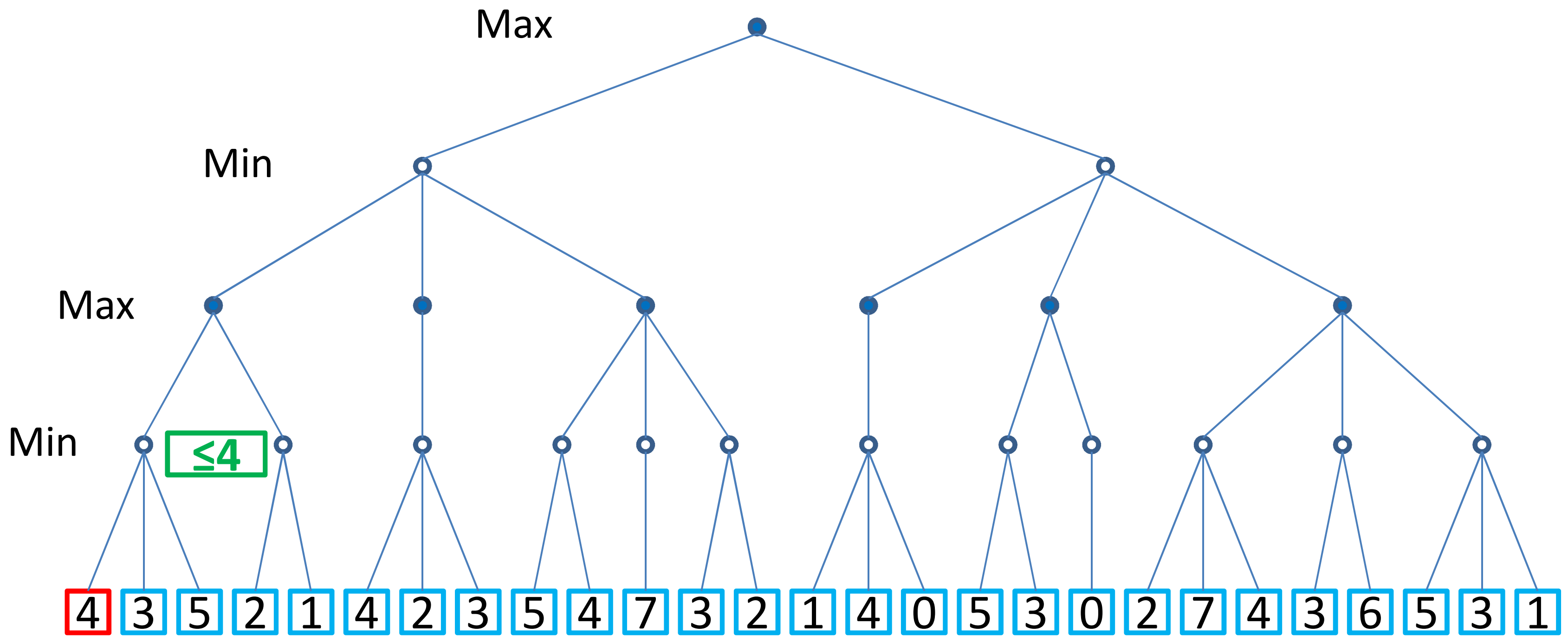
MINIMAX WITH $\alpha\beta$-PRUNING

MiniMax with $\alpha\beta$-pruning

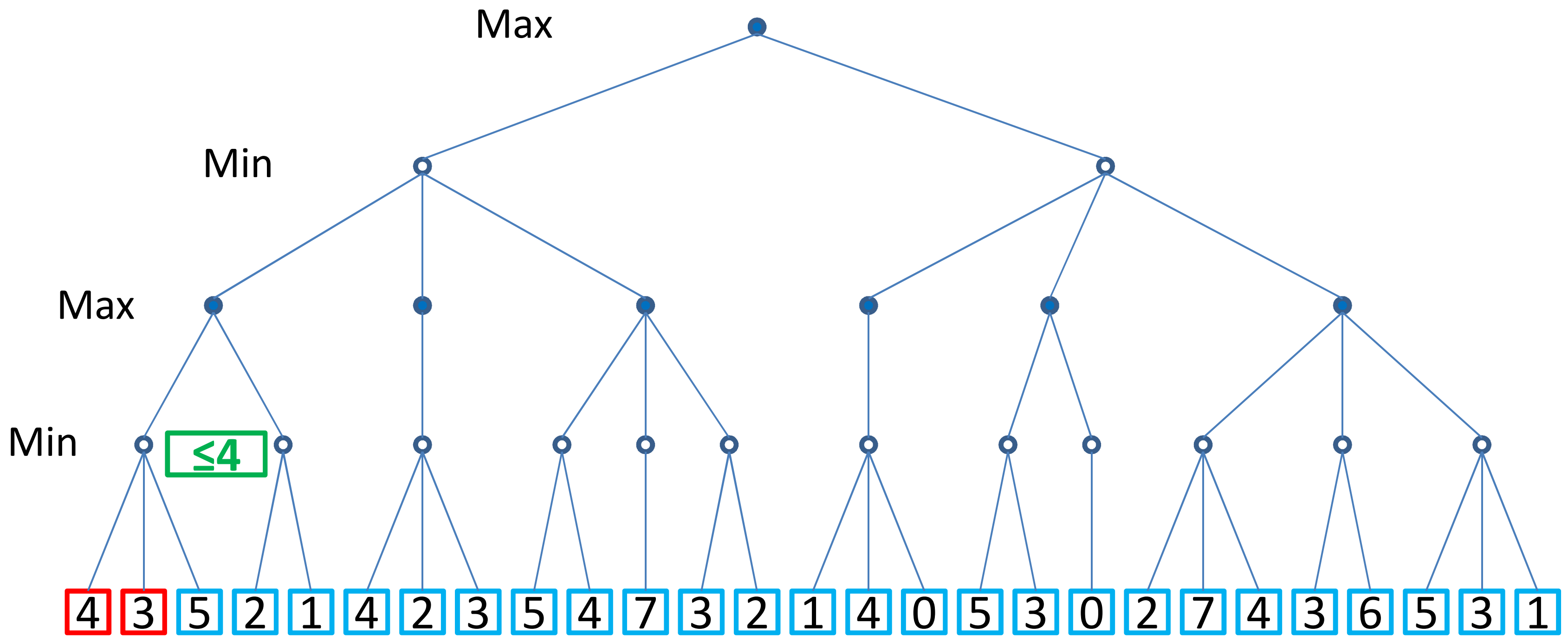


MiniMax with $\alpha\beta$-pruning

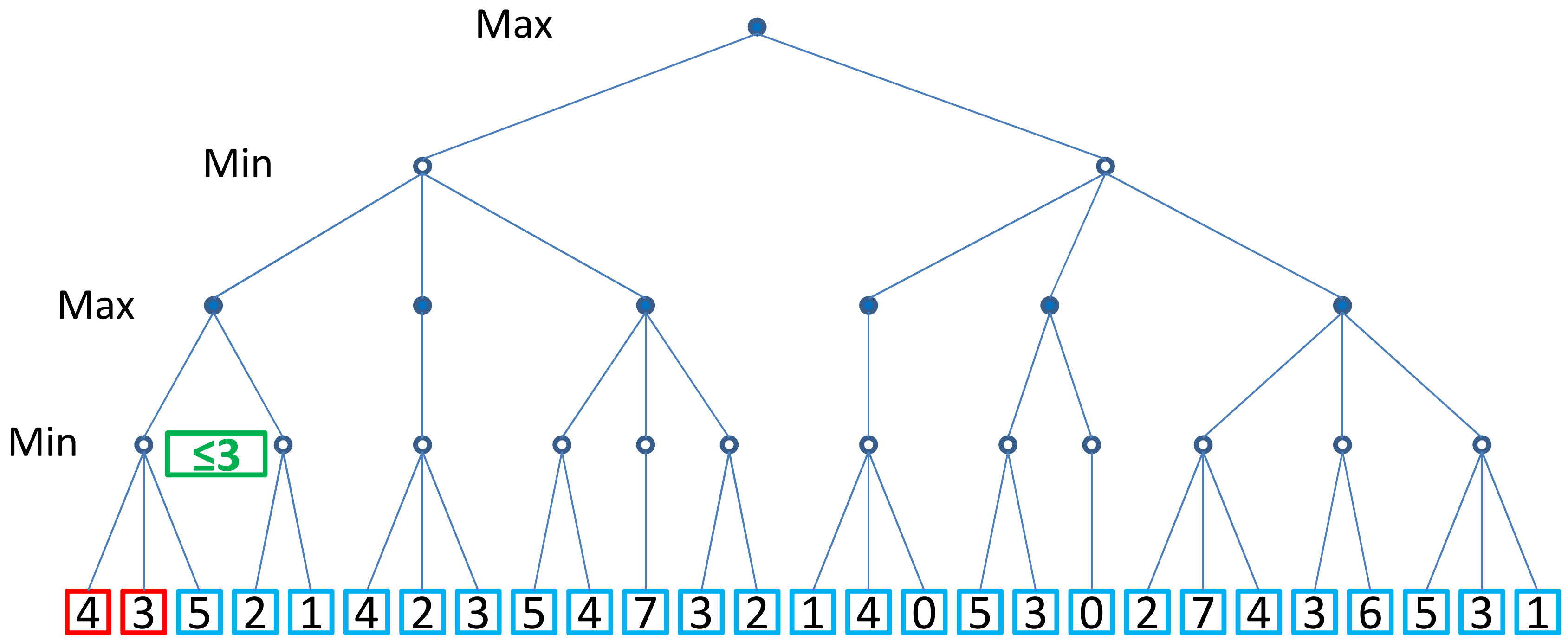
- **α-nodes**: Temporary values at MIN-nodes



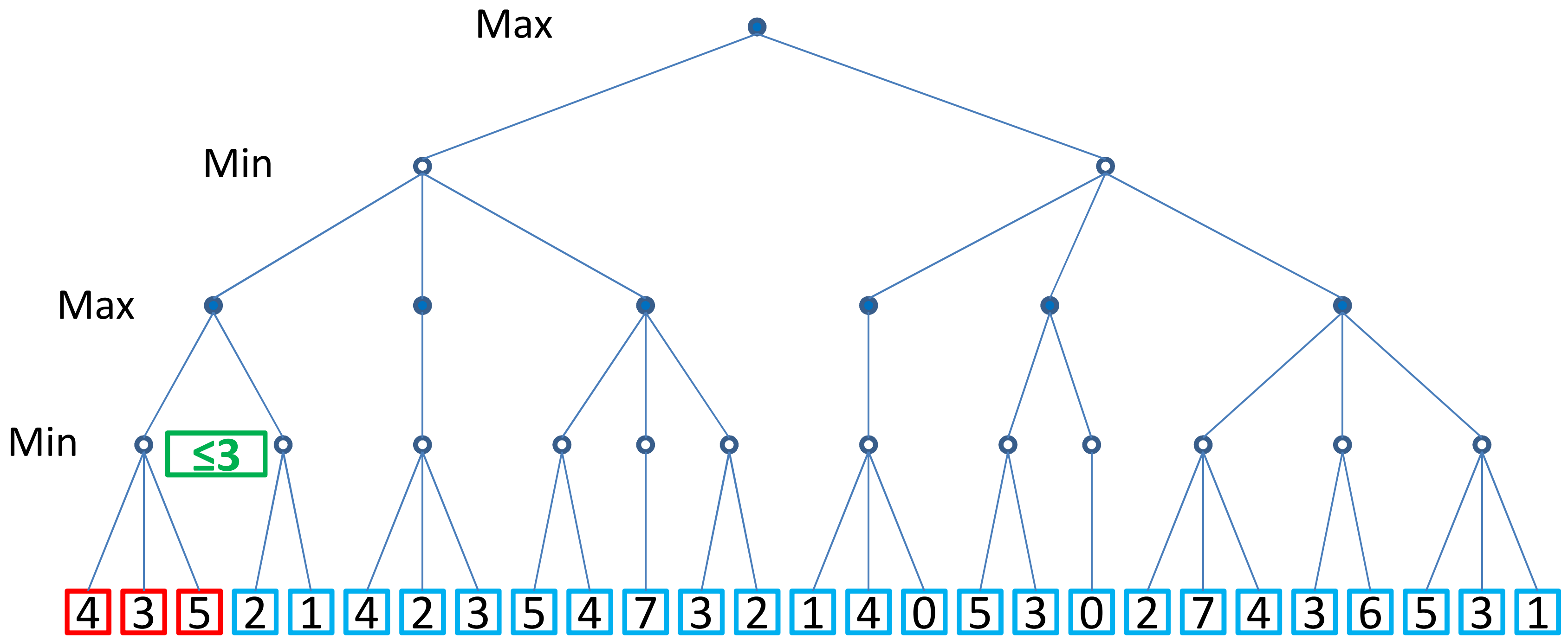
MiniMax with $\alpha\beta$-pruning



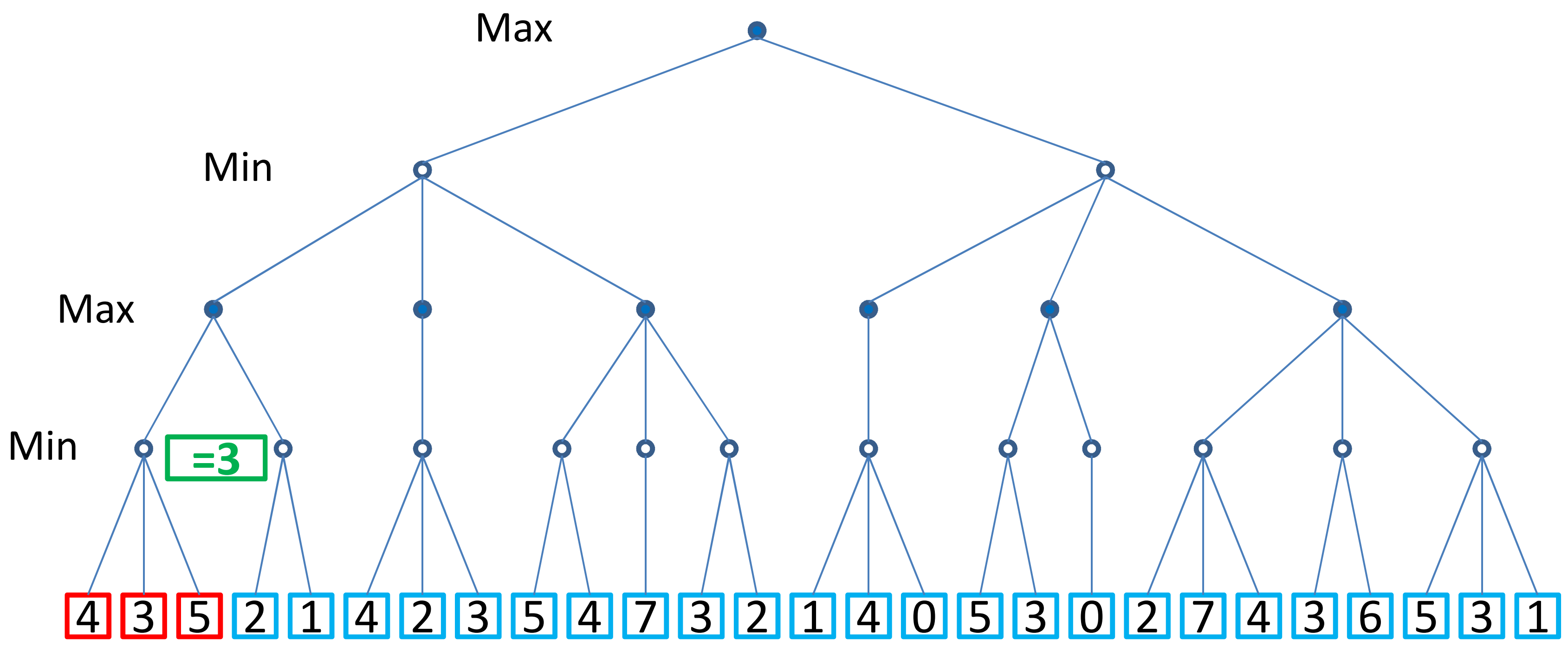
MiniMax with $\alpha\beta$-pruning



MiniMax with $\alpha\beta$-pruning

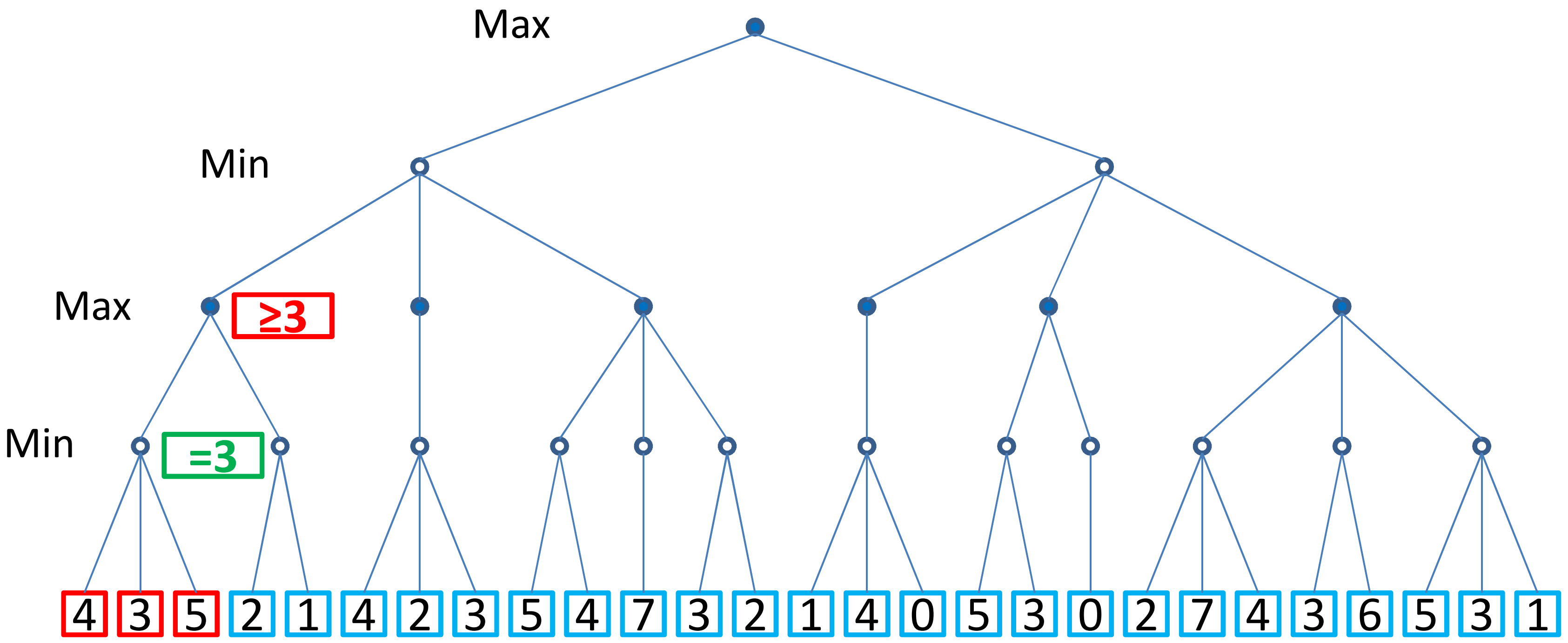


MiniMax with $\alpha\beta$-pruning

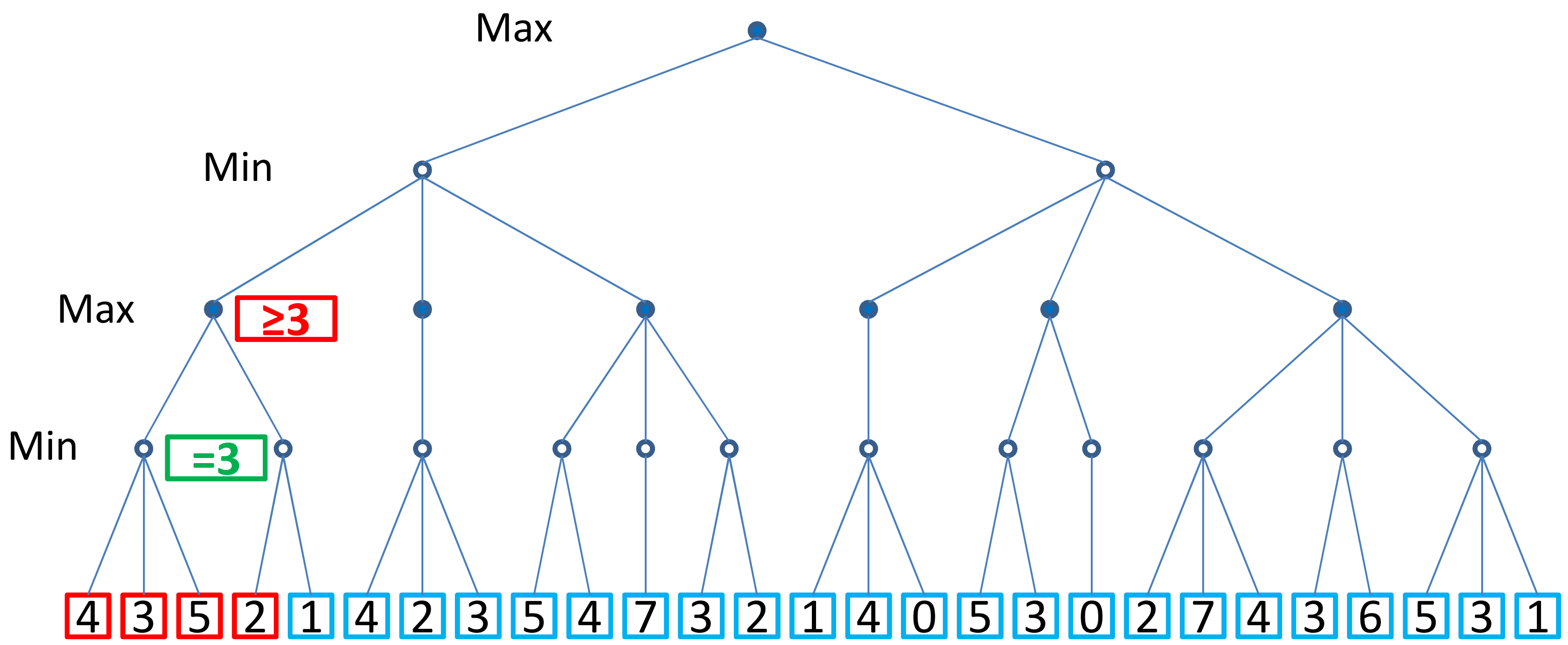


MiniMax with $\alpha\beta$-pruning

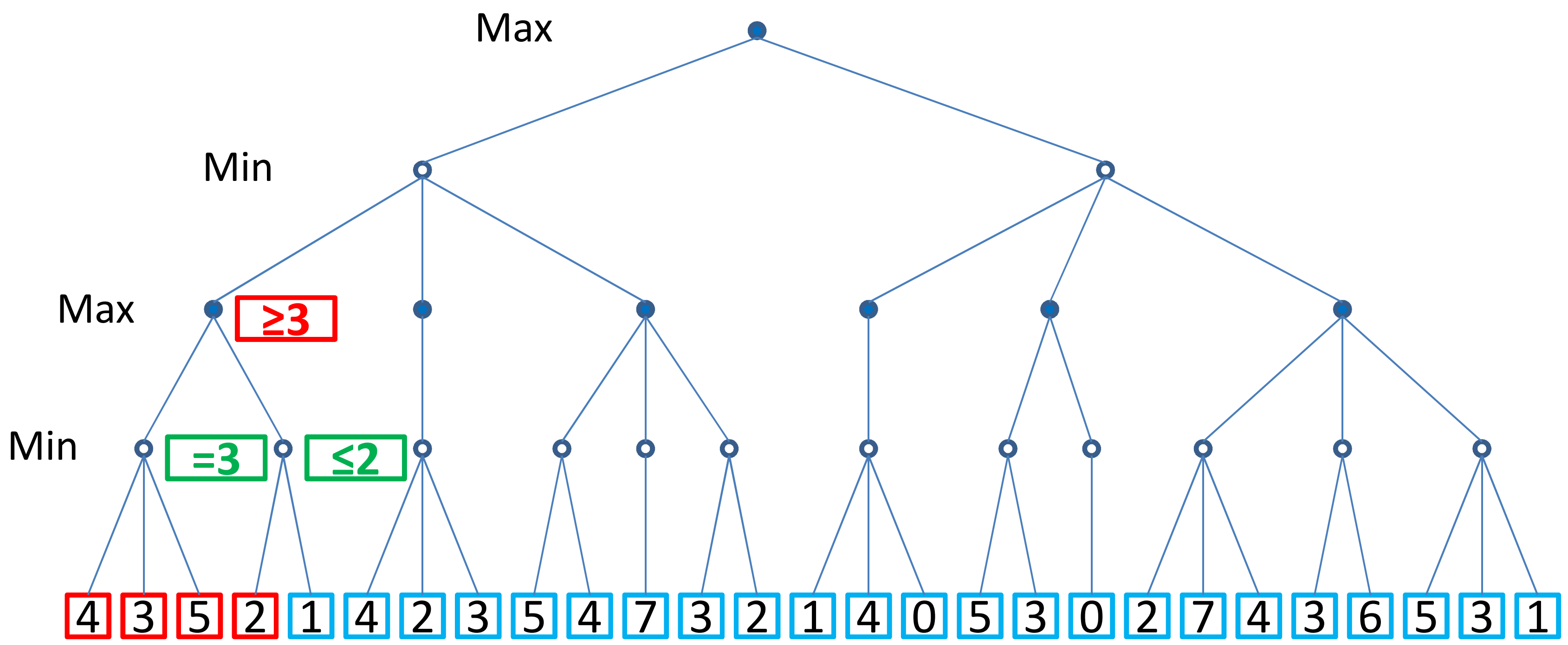
- **β-nodes**: Temporary values at MAX-nodes



MiniMax with $\alpha\beta$-pruning

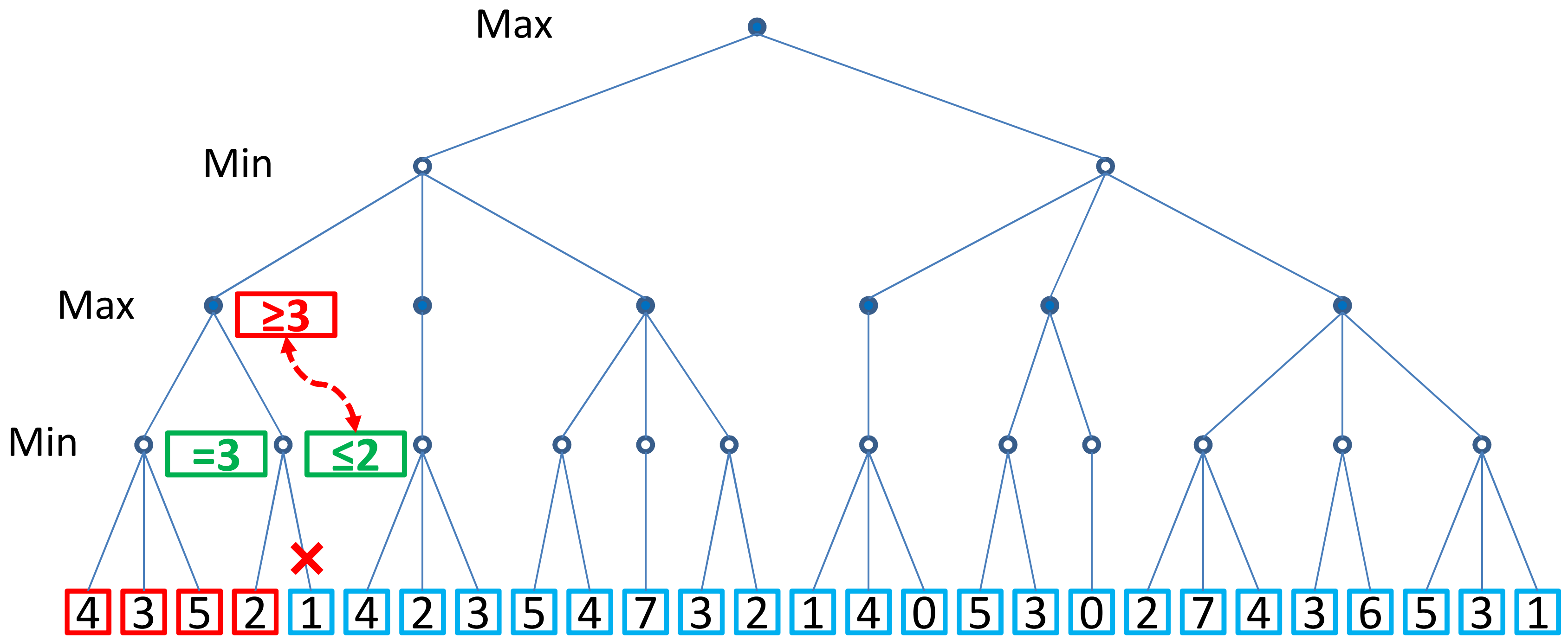


MiniMax with $\alpha\beta$-pruning

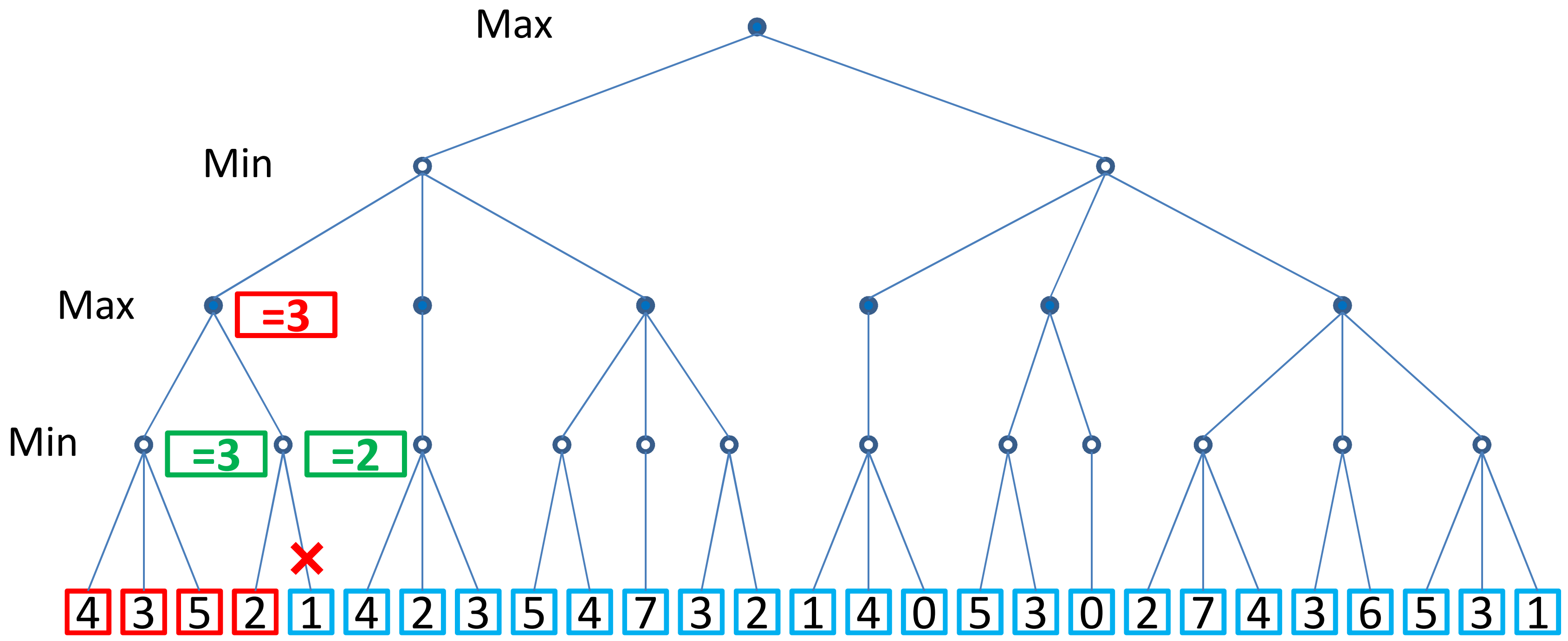


MiniMax with $\alpha\beta$-pruning

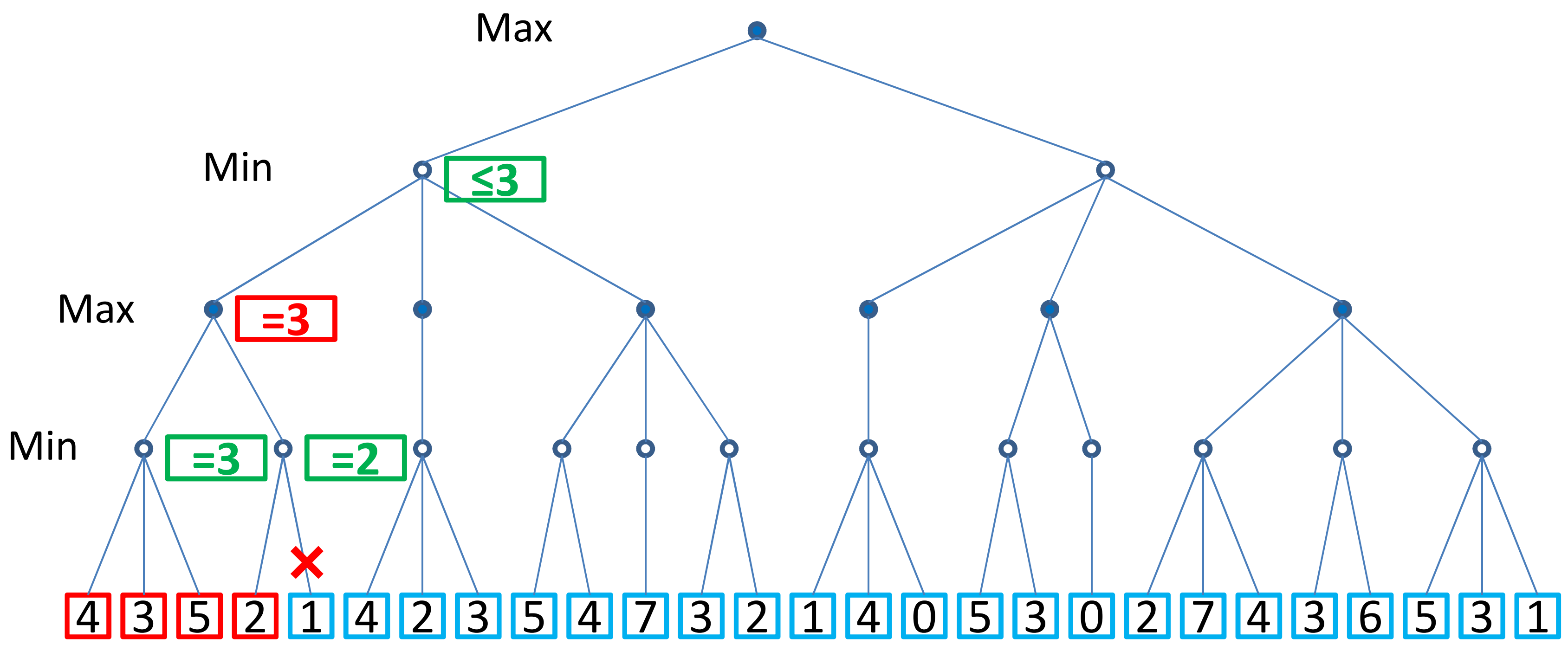
- **Prune: Parent β-node ≥ Child α-node**



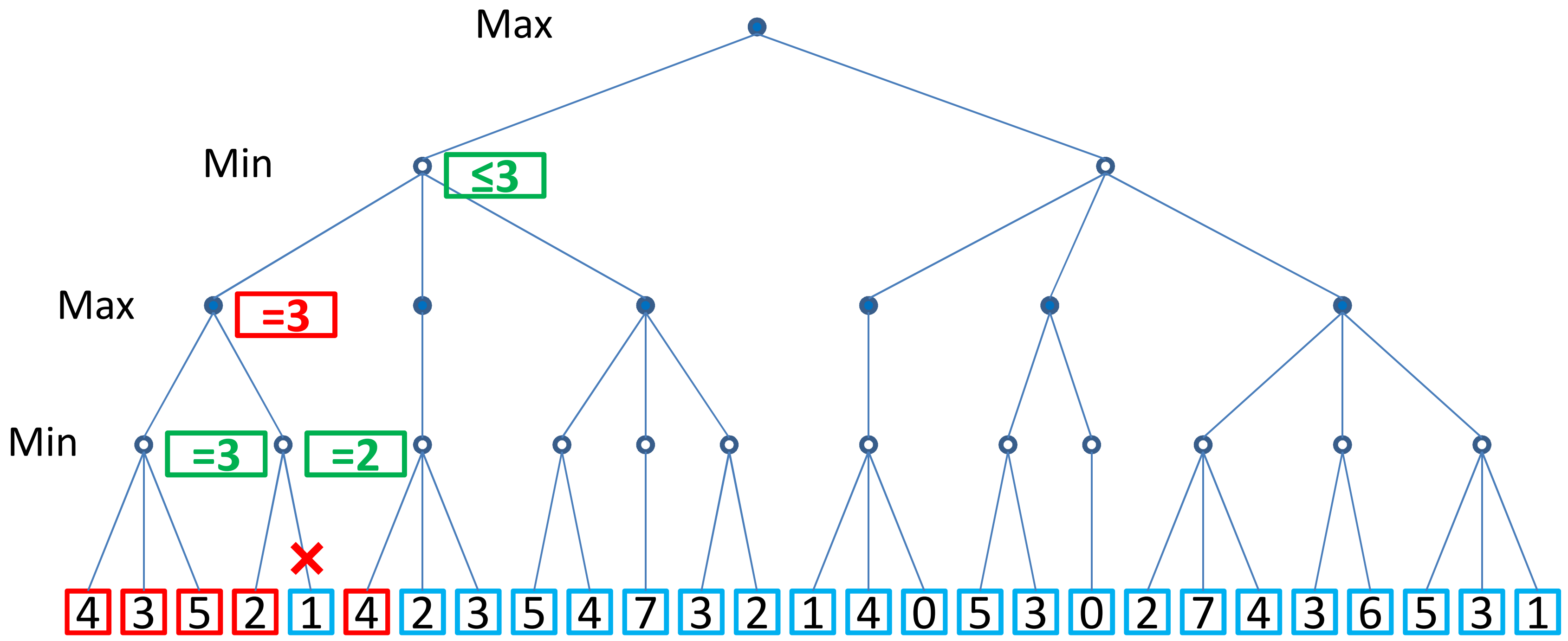
MiniMax with $\alpha\beta$-pruning



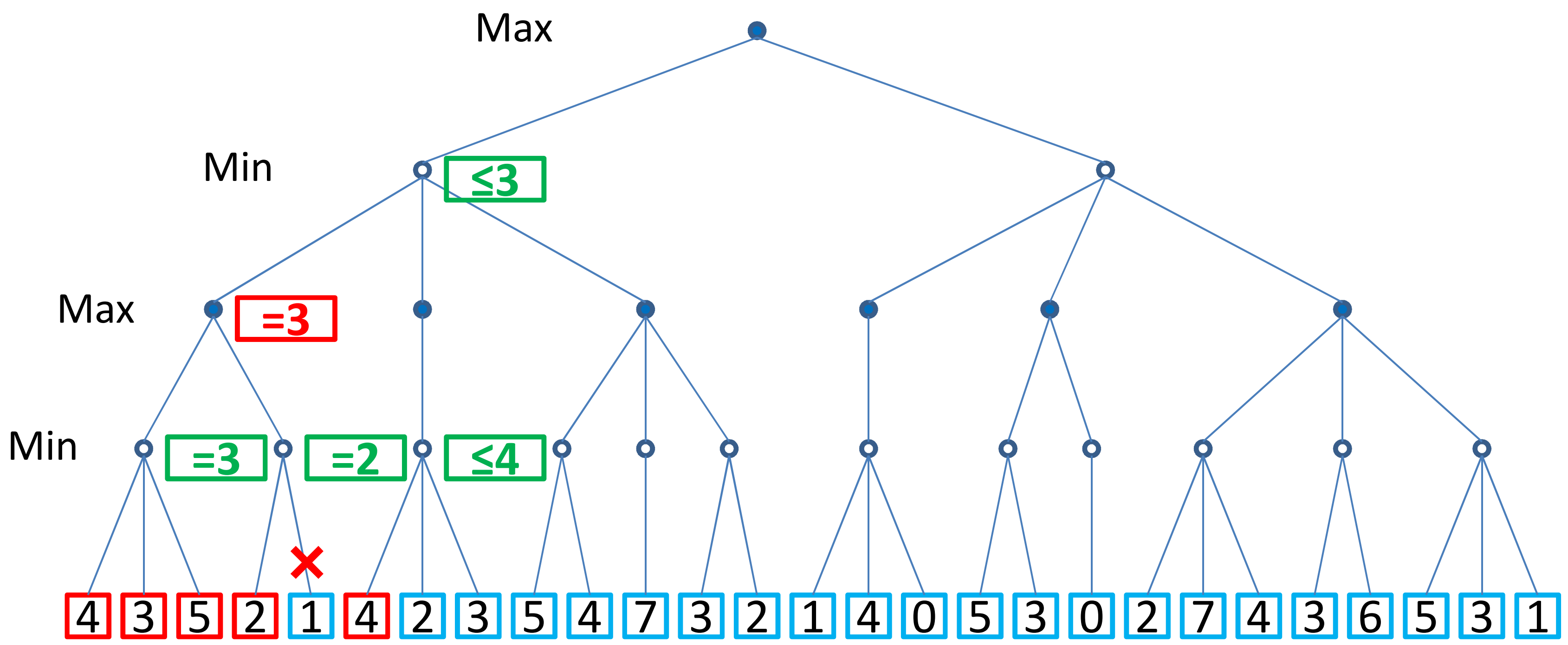
MiniMax with $\alpha\beta$-pruning



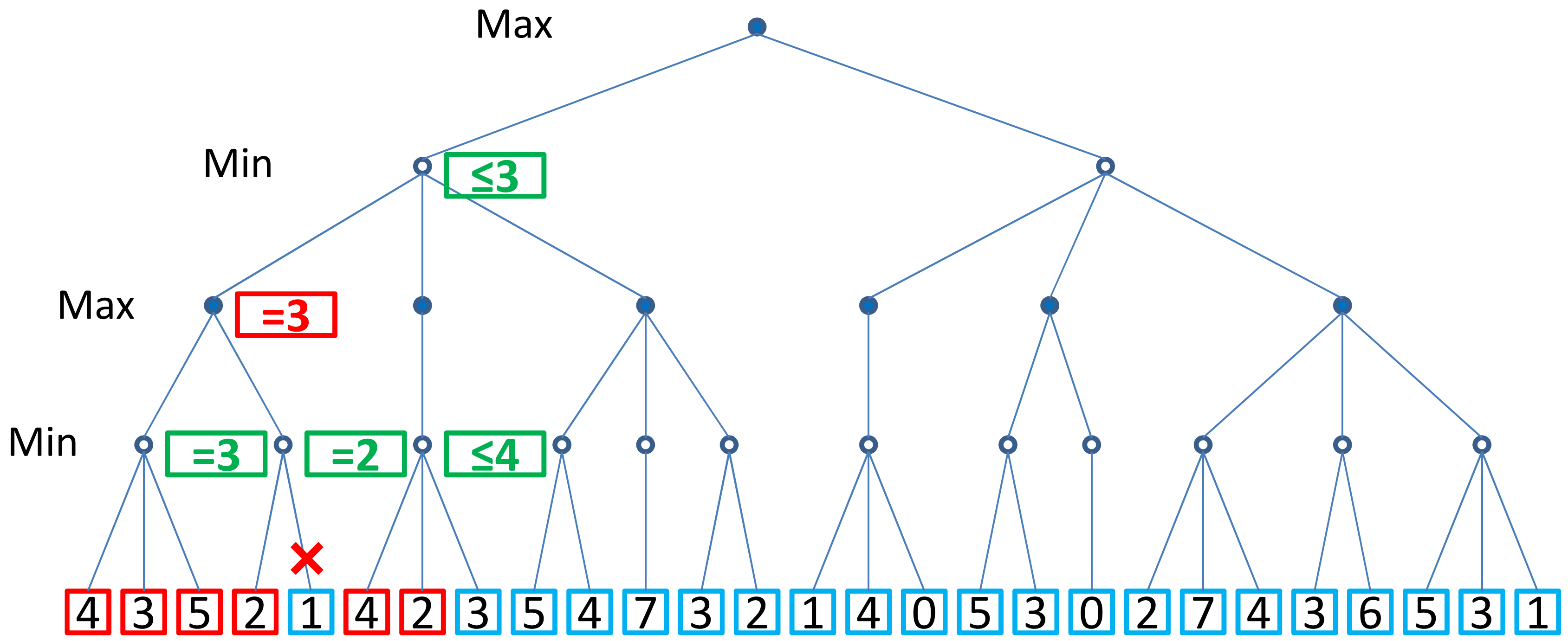
MiniMax with $\alpha\beta$-pruning



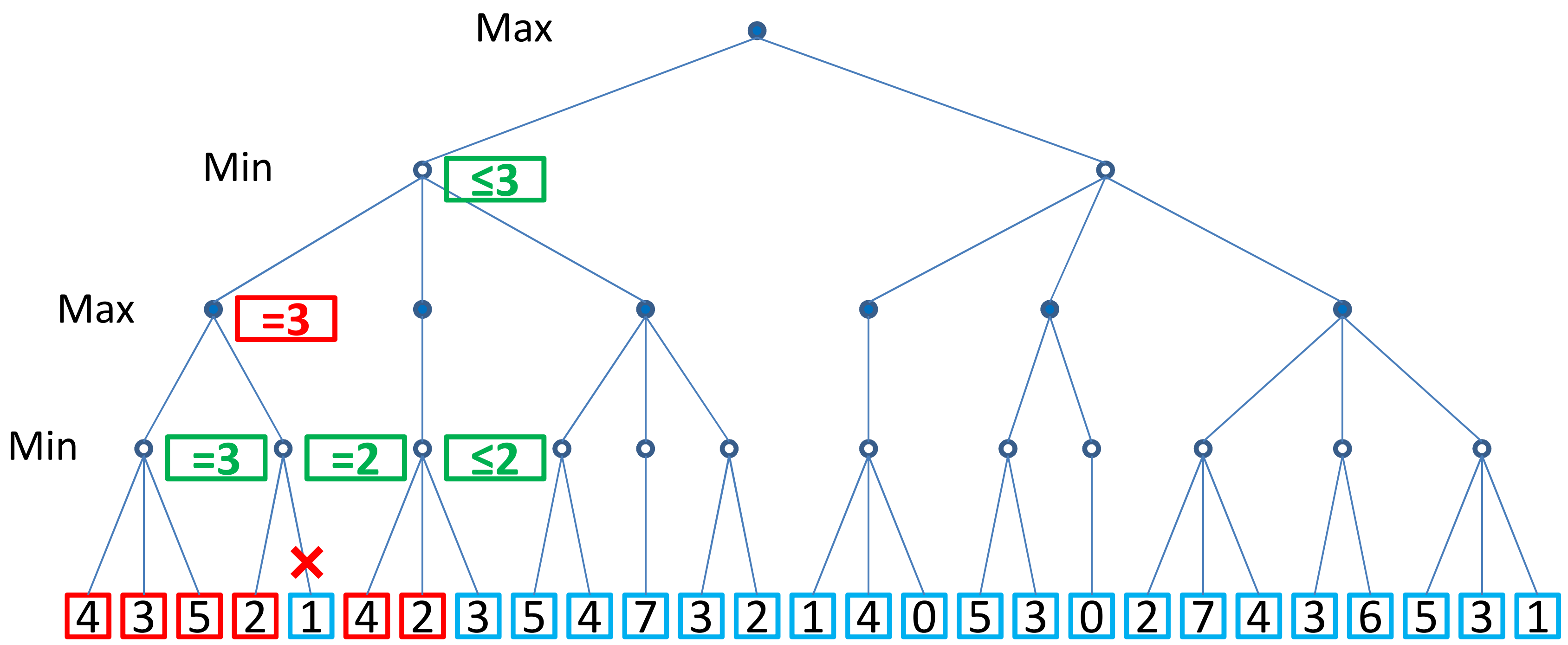
MiniMax with $\alpha\beta$-pruning



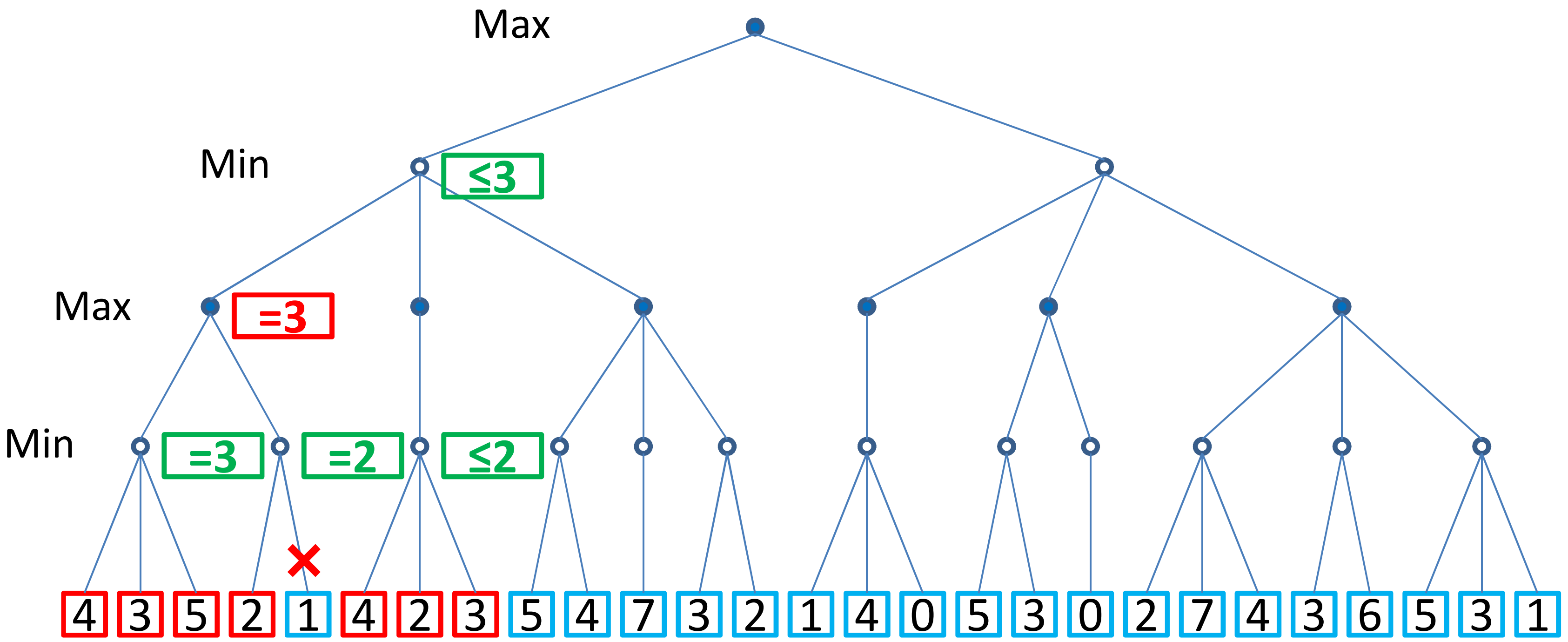
MiniMax with $\alpha\beta$-pruning



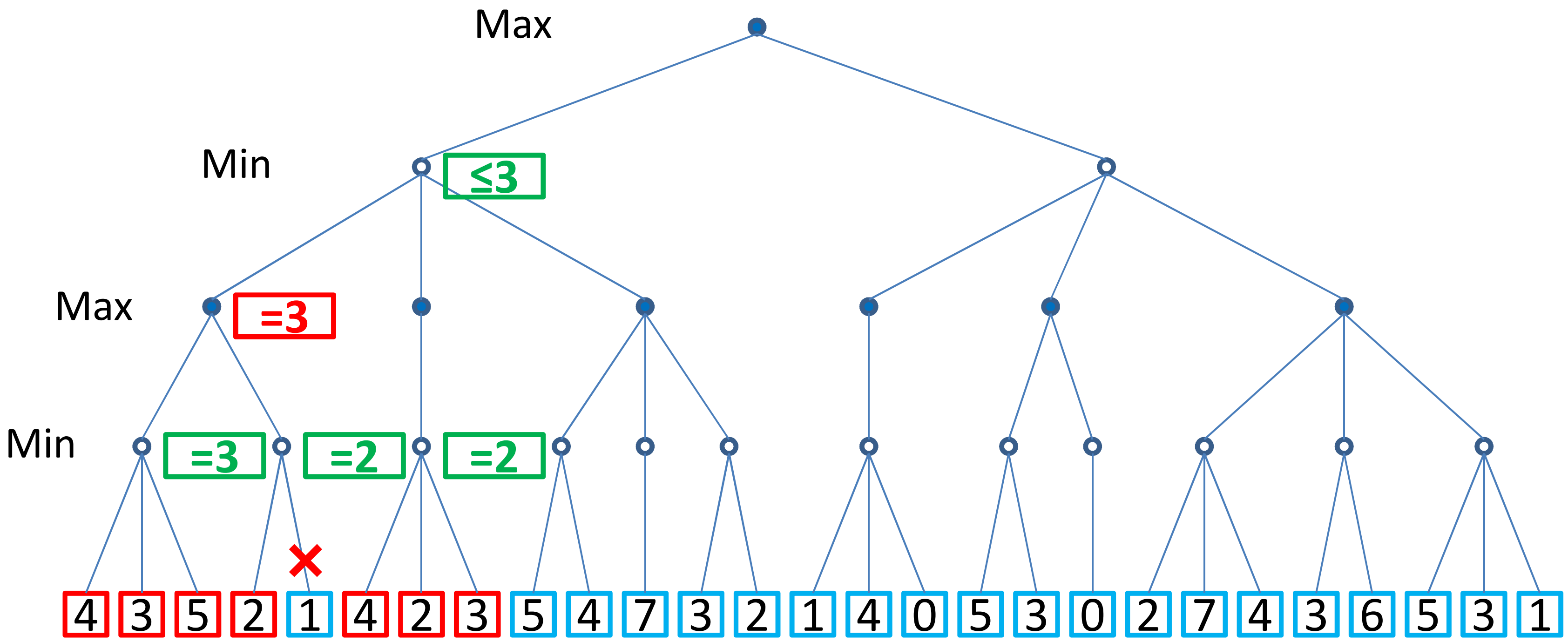
MiniMax with $\alpha\beta$-pruning



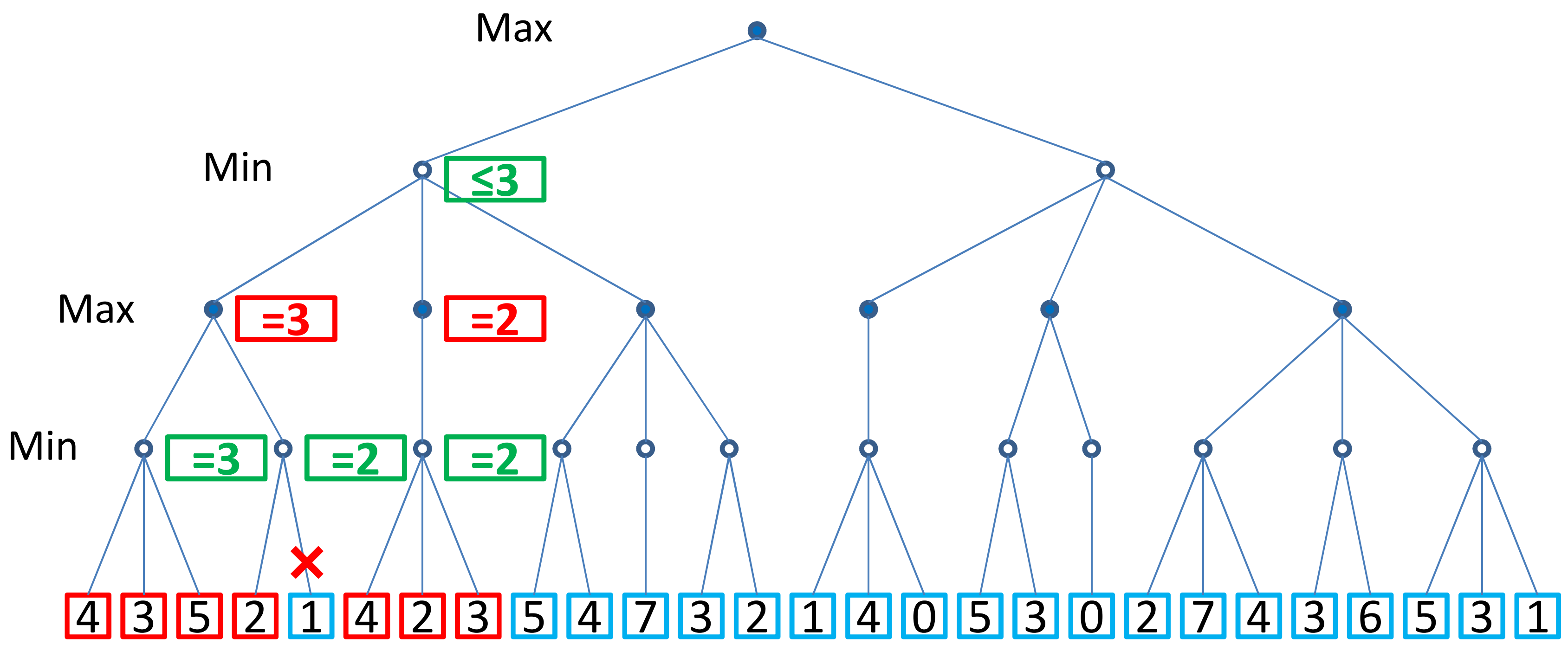
MiniMax with $\alpha\beta$-pruning



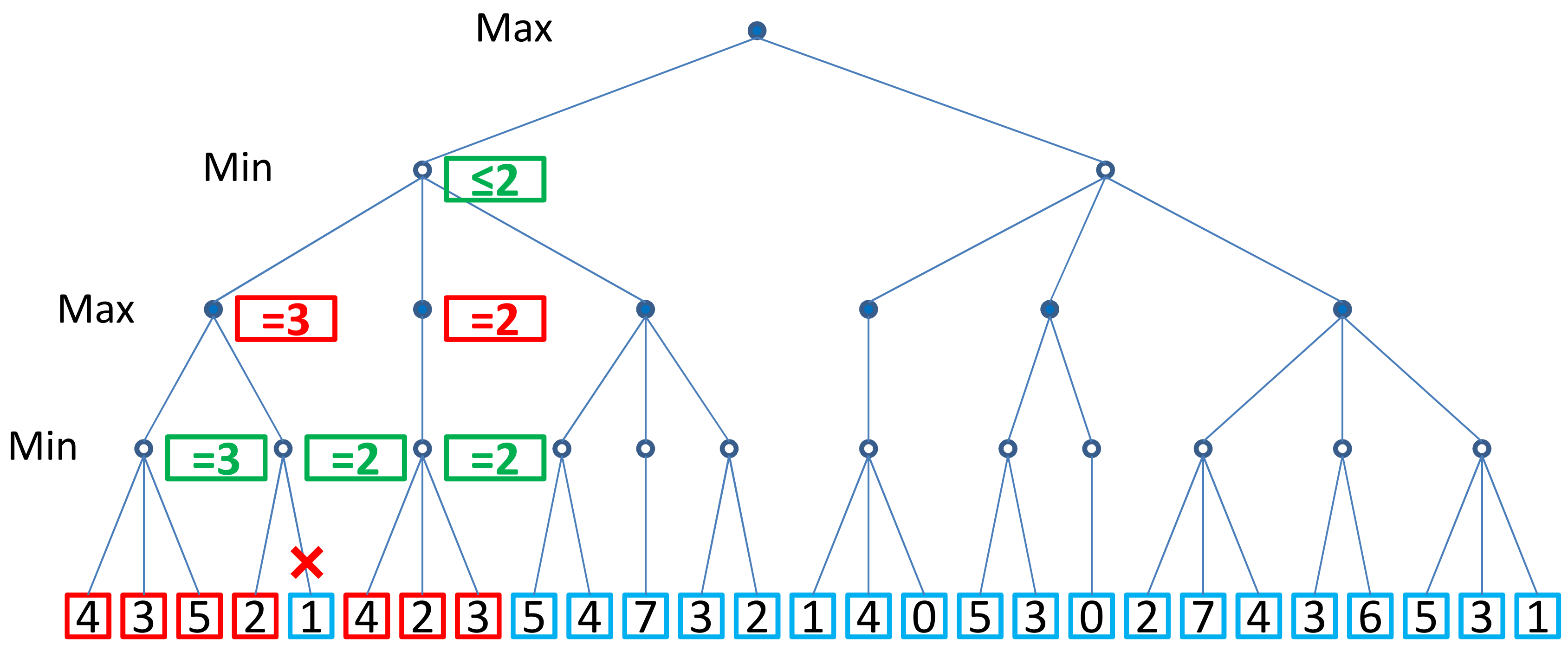
MiniMax with $\alpha\beta$-pruning



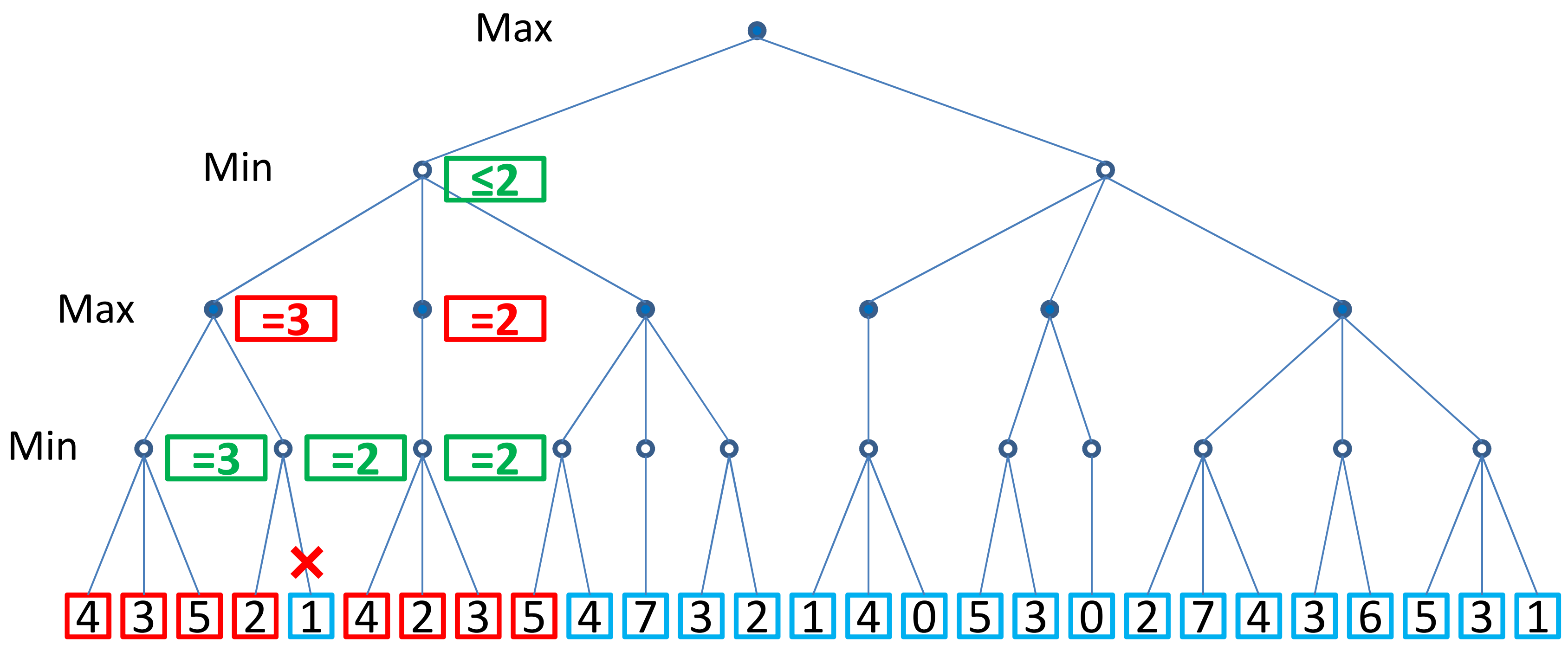
MiniMax with $\alpha\beta$-pruning



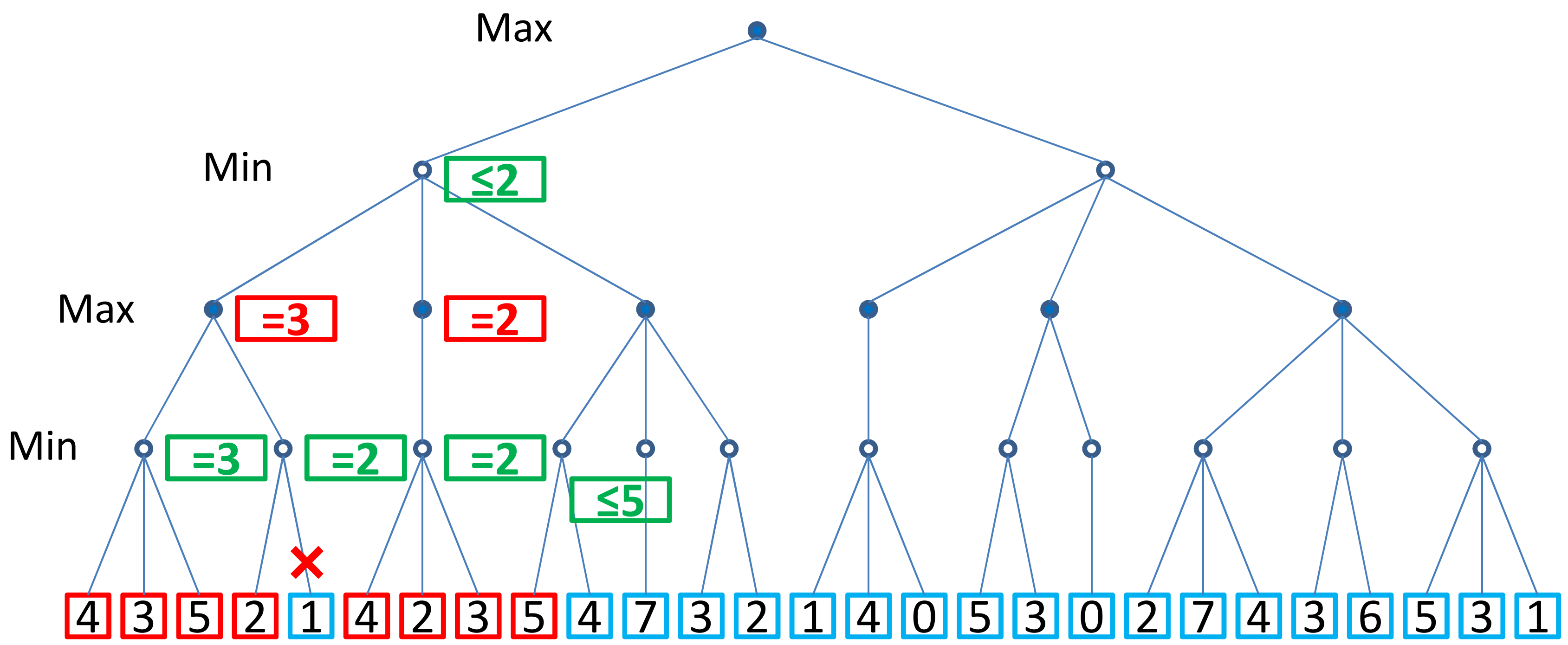
MiniMax with $\alpha\beta$-pruning



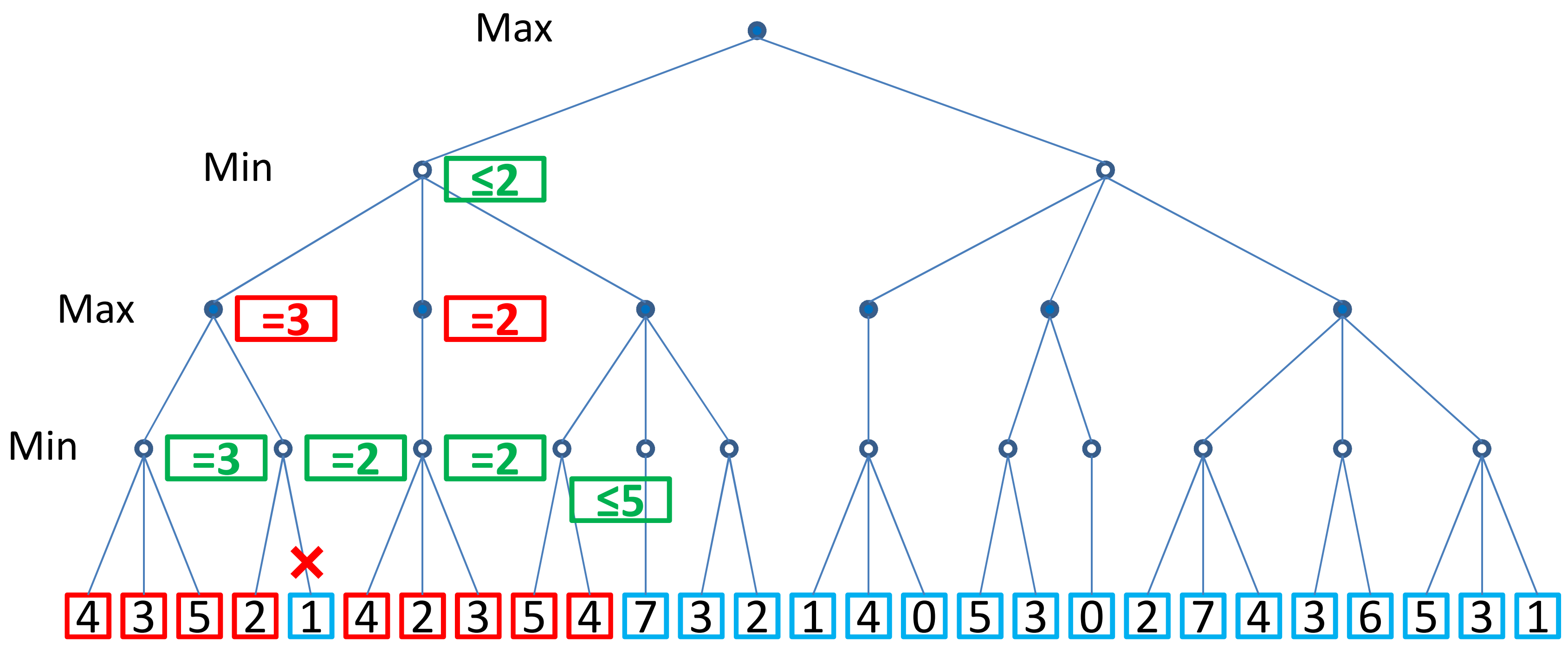
MiniMax with $\alpha\beta$-pruning



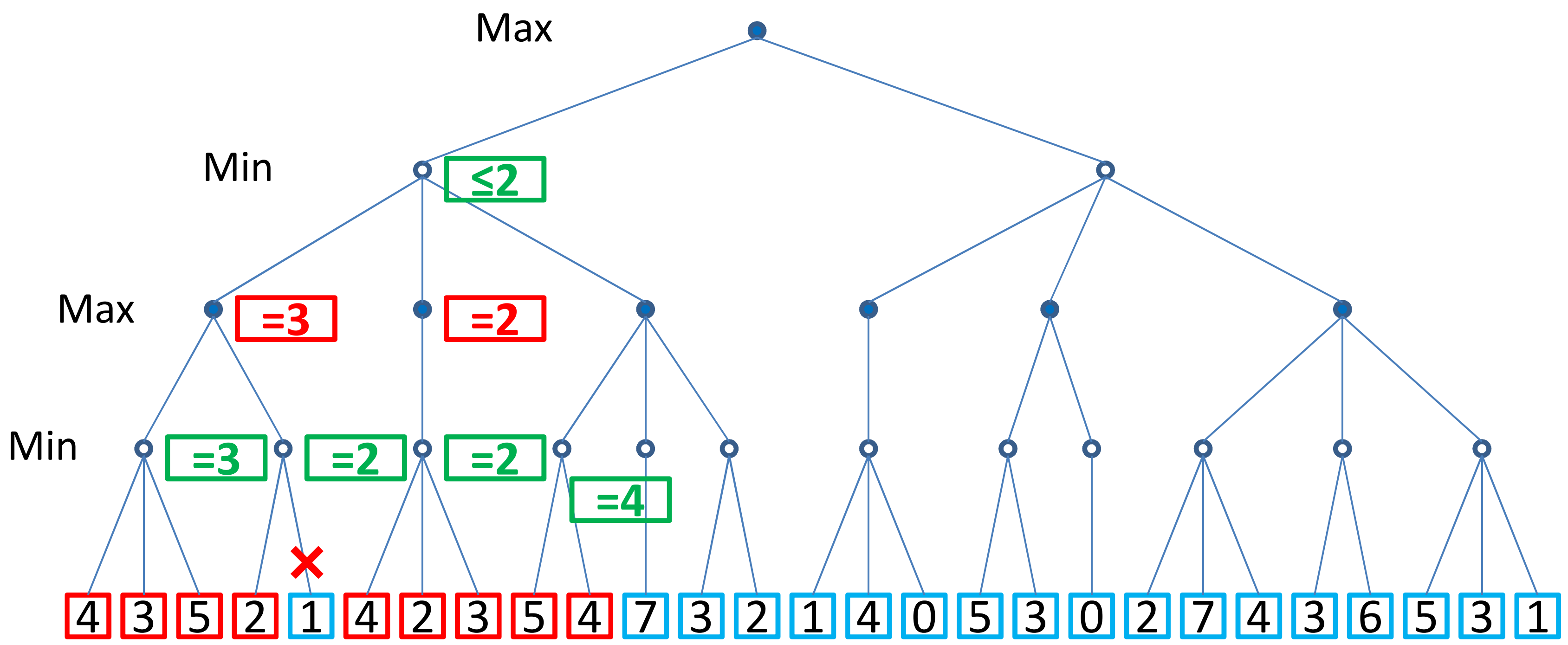
MiniMax with $\alpha\beta$-pruning



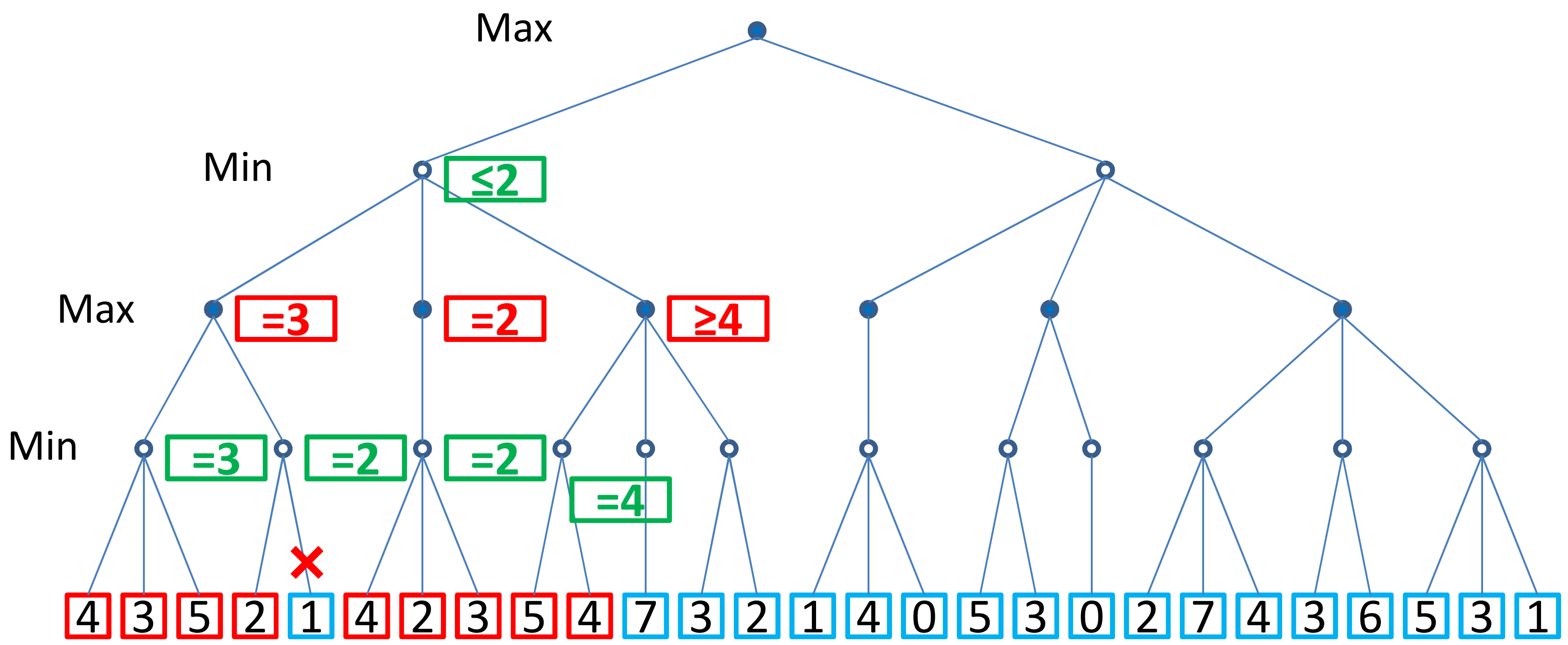
MiniMax with $\alpha\beta$-pruning



MiniMax with $\alpha\beta$-pruning

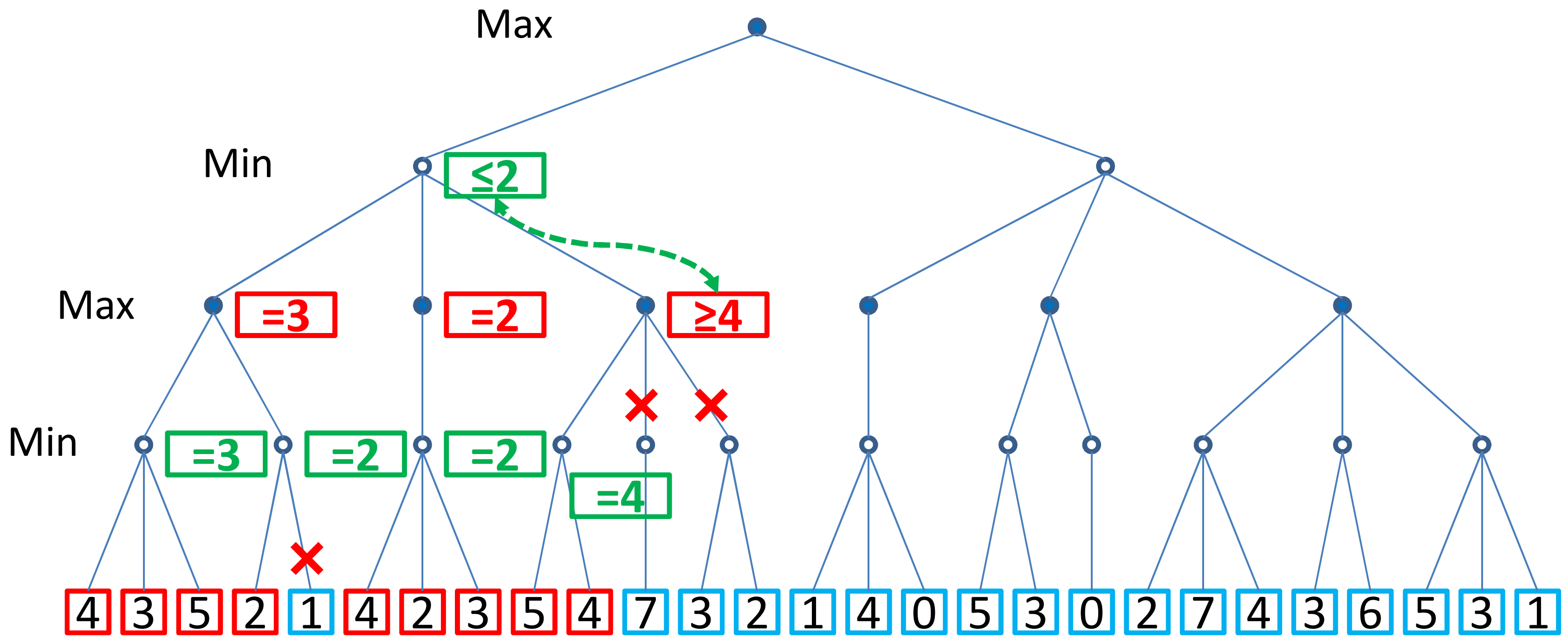


MiniMax with $\alpha\beta$-pruning

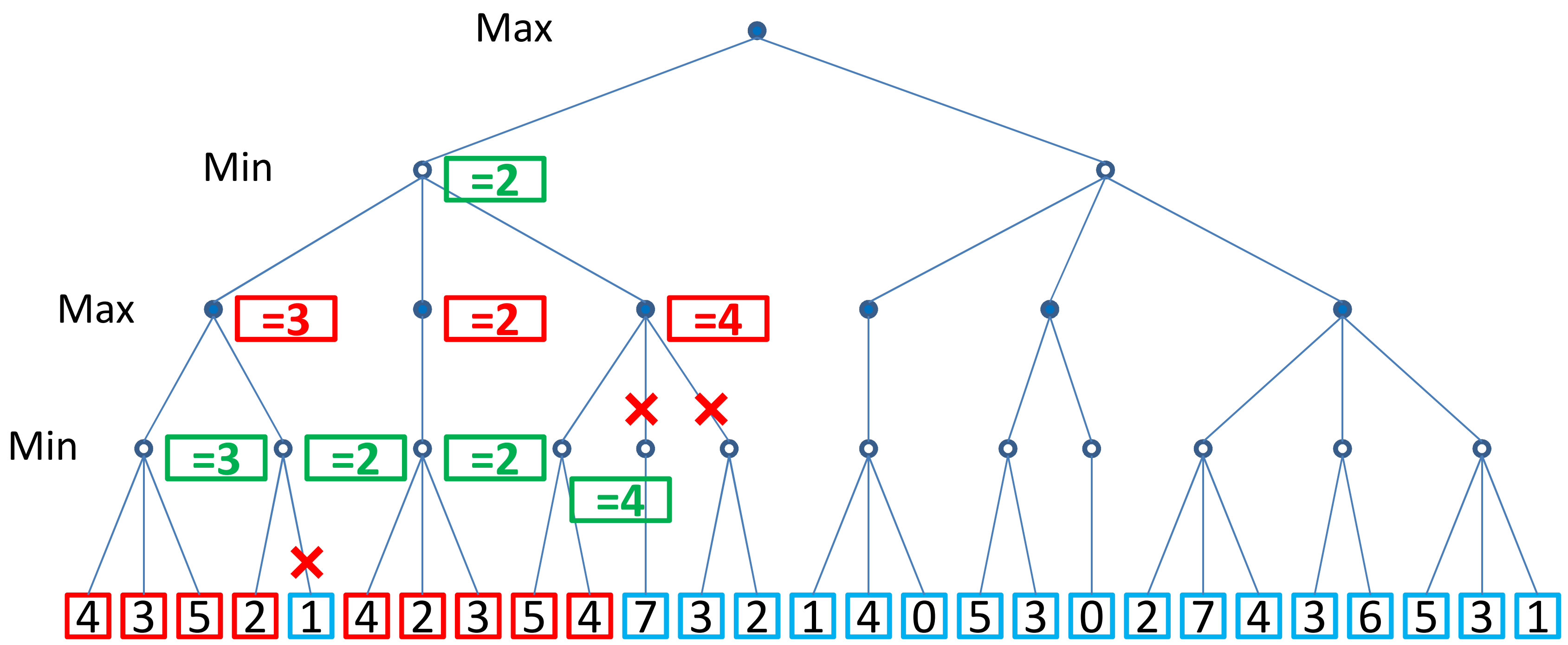


MiniMax with $\alpha\beta$-pruning

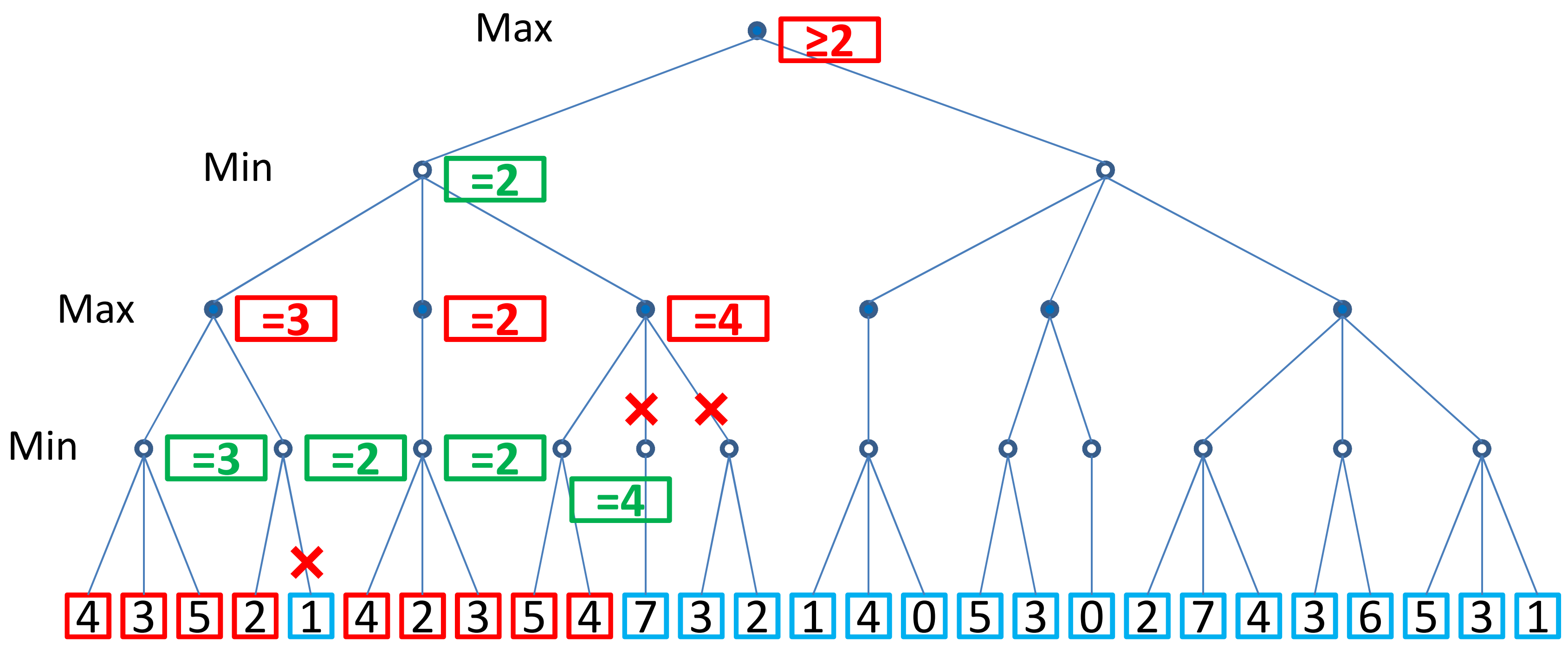
- **Prune: Parent α-node ≤ Child β-node**



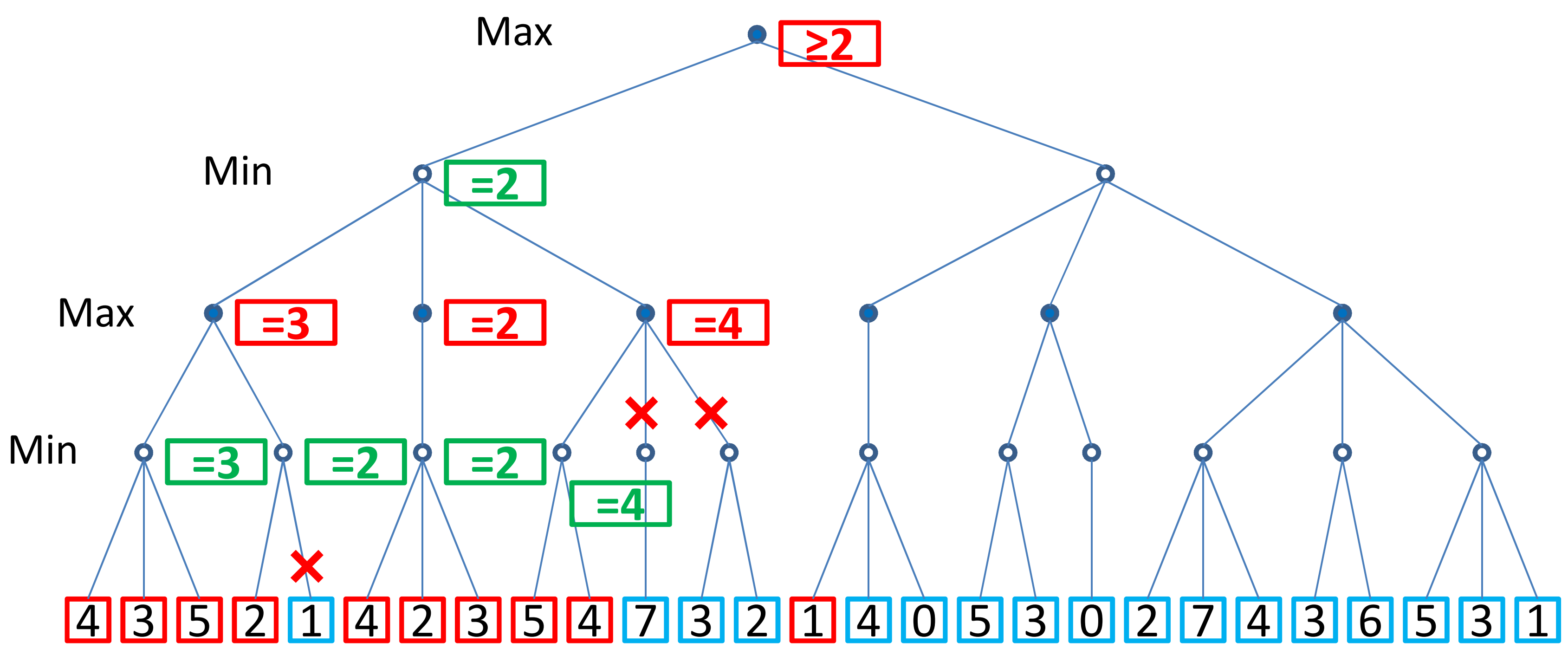
MiniMax with $\alpha\beta$-pruning



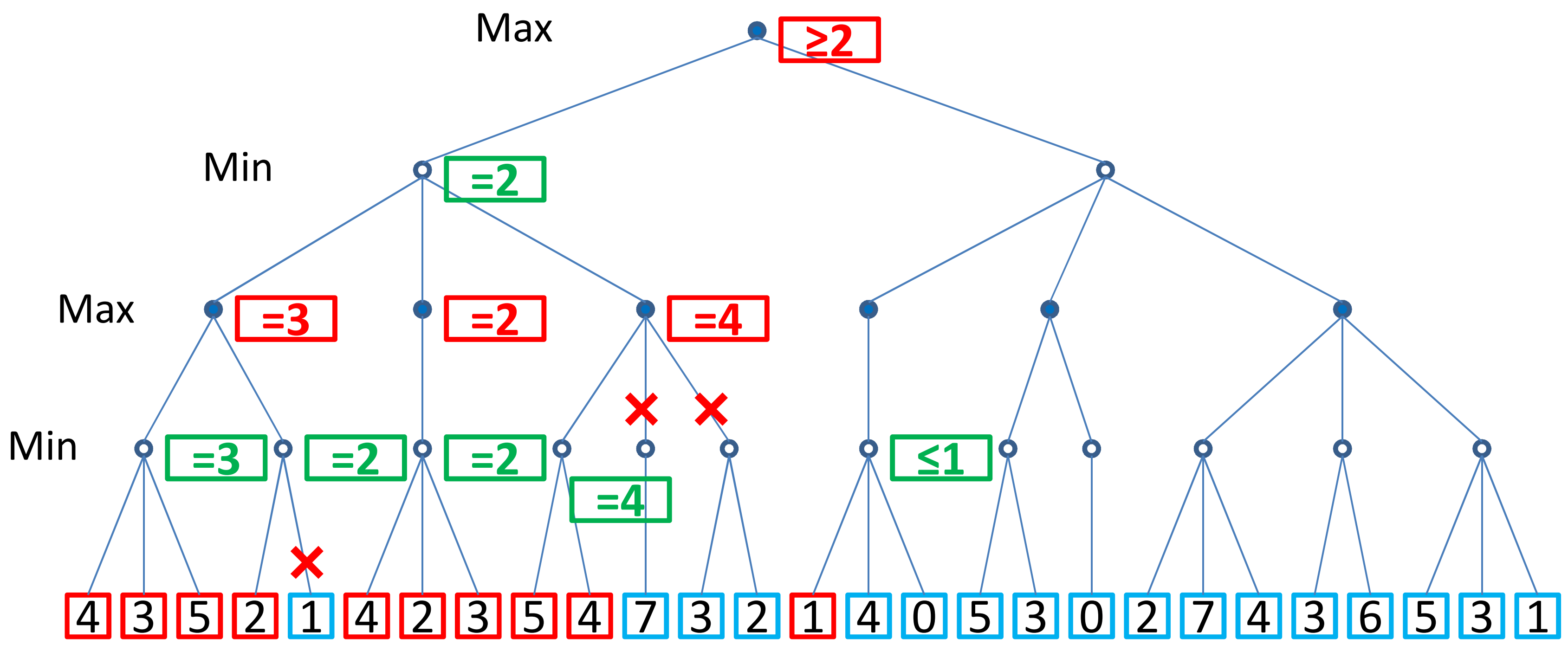
MiniMax with $\alpha\beta$-pruning



MiniMax with $\alpha\beta$-pruning

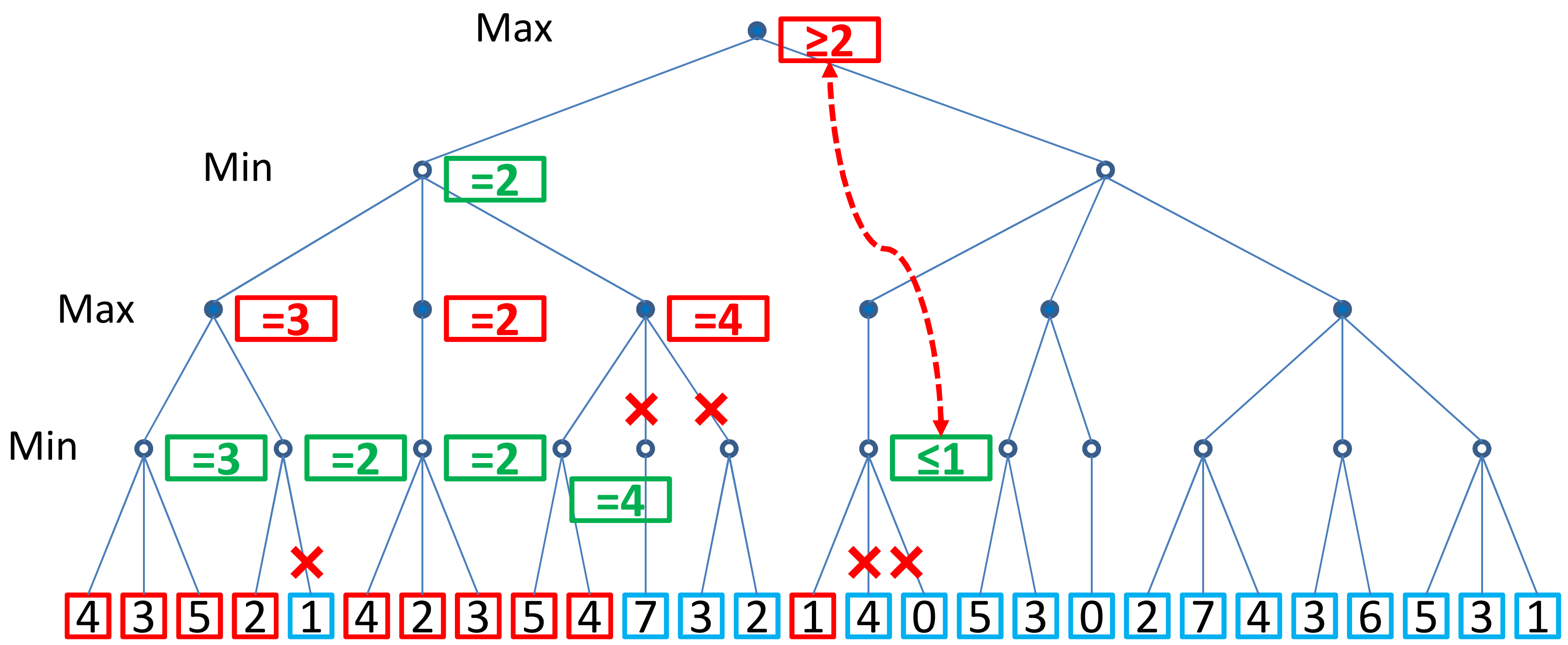


MiniMax with $\alpha\beta$-pruning

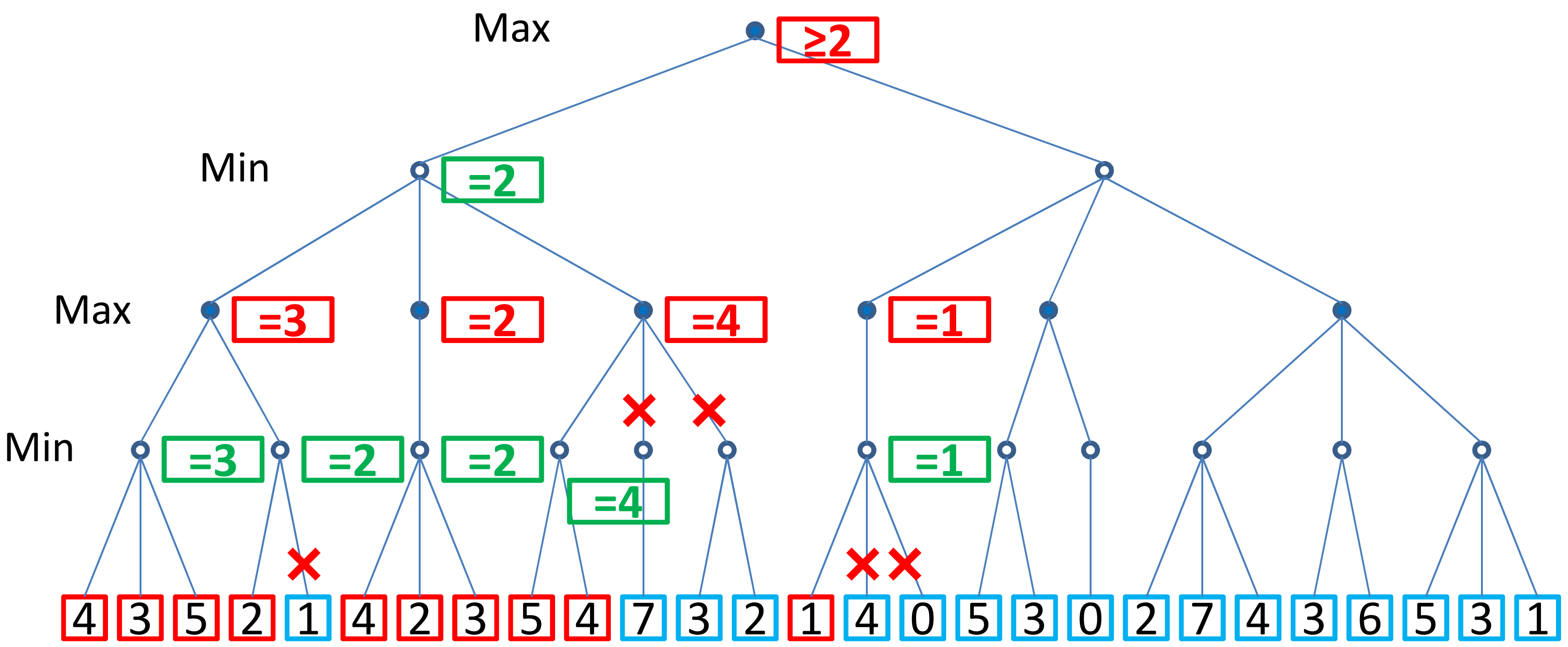


MiniMax with $\alpha\beta$-pruning

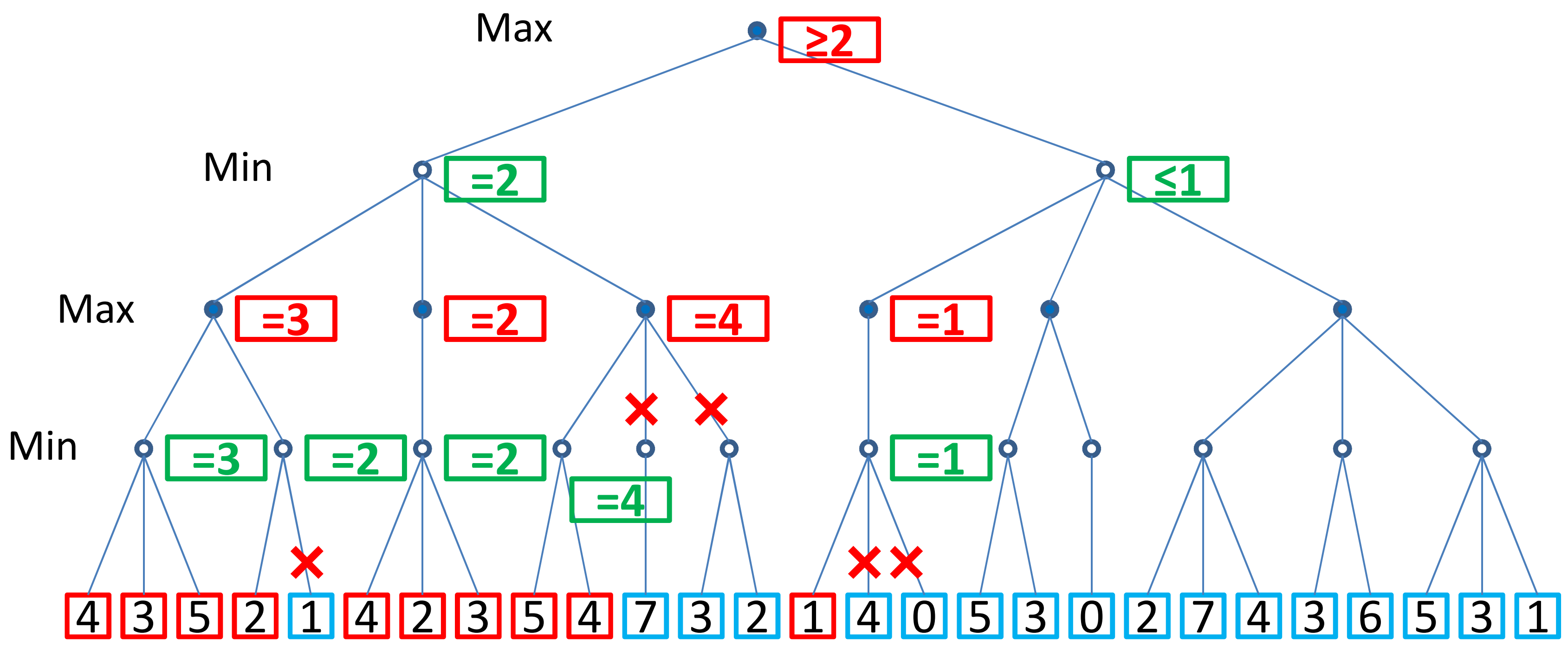
- **"Deep" cut-off: Ancestor β-node ≥ α-node**



MiniMax with $\alpha\beta$-pruning

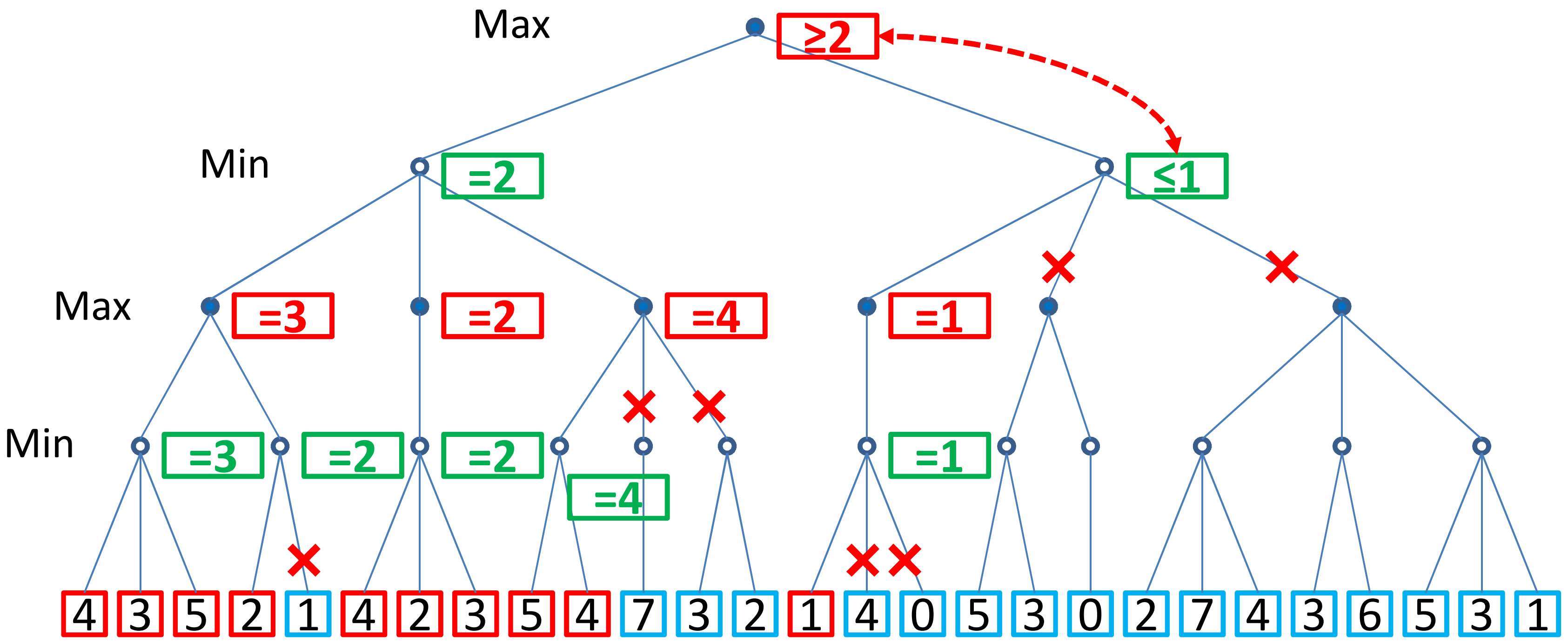


MiniMax with $\alpha\beta$-pruning



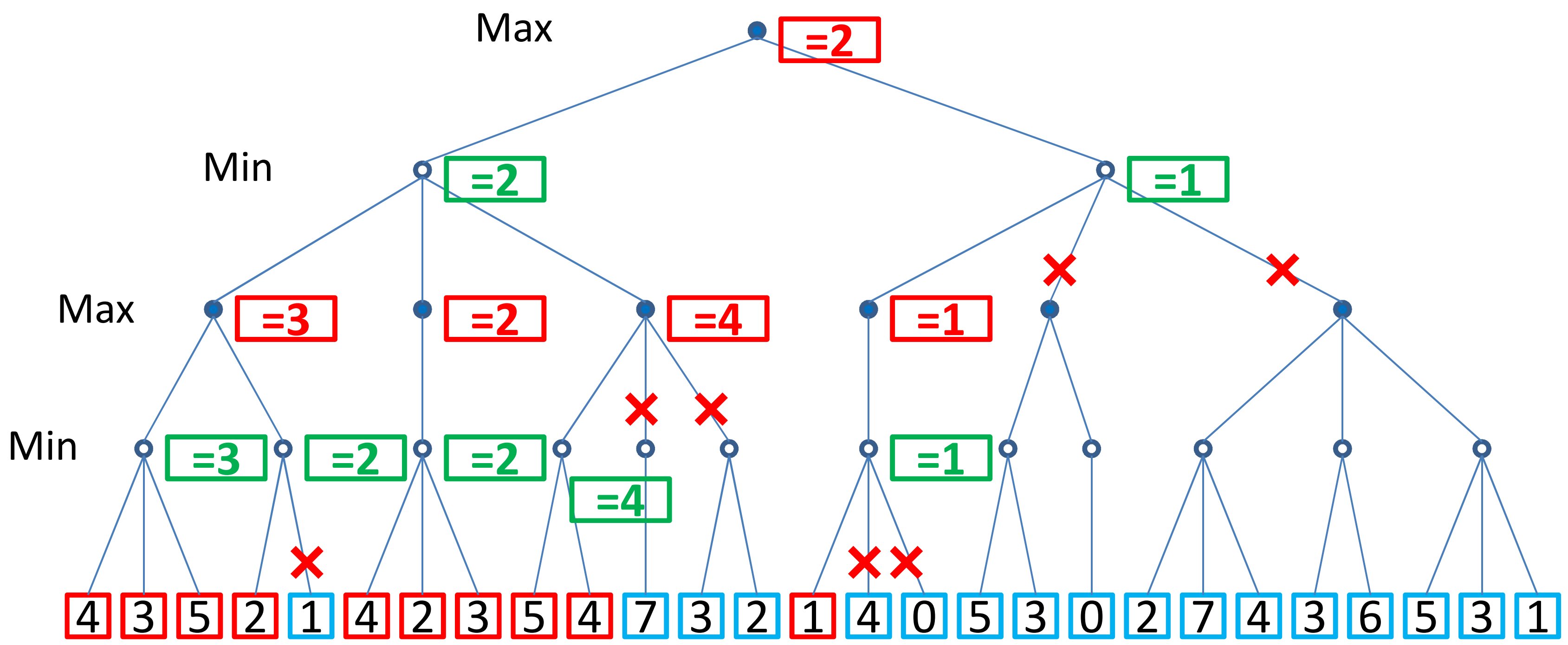
MiniMax with $\alpha\beta$-pruning

- **Prune: Parent β-node ≥ Child α-node**



MiniMax with $\alpha\beta$-pruning

- **17 static evaluations saved**

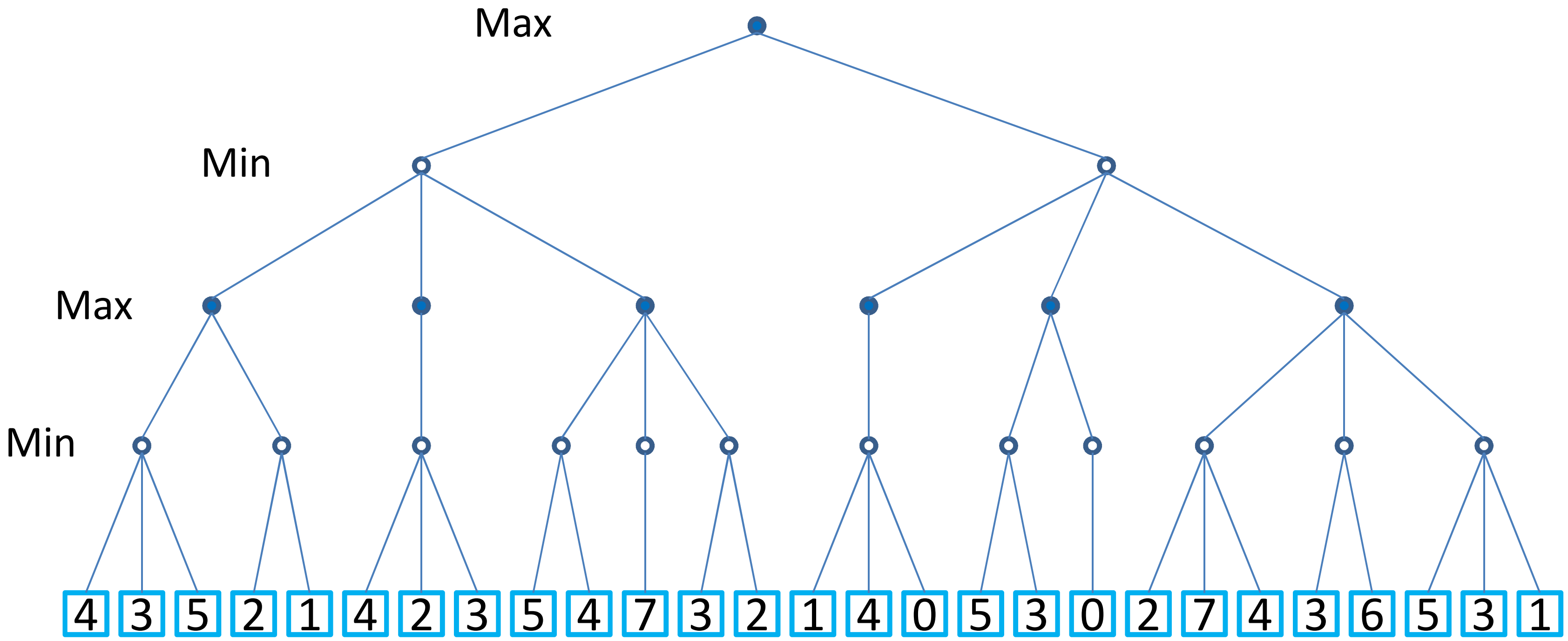


MiniMax & Constraint Processing: MiniMax Algorithm

PROBLEM 2

Problem 2

- Can the nodes be ordered in such a way that $\alpha\beta$-pruning can cut off more branches?

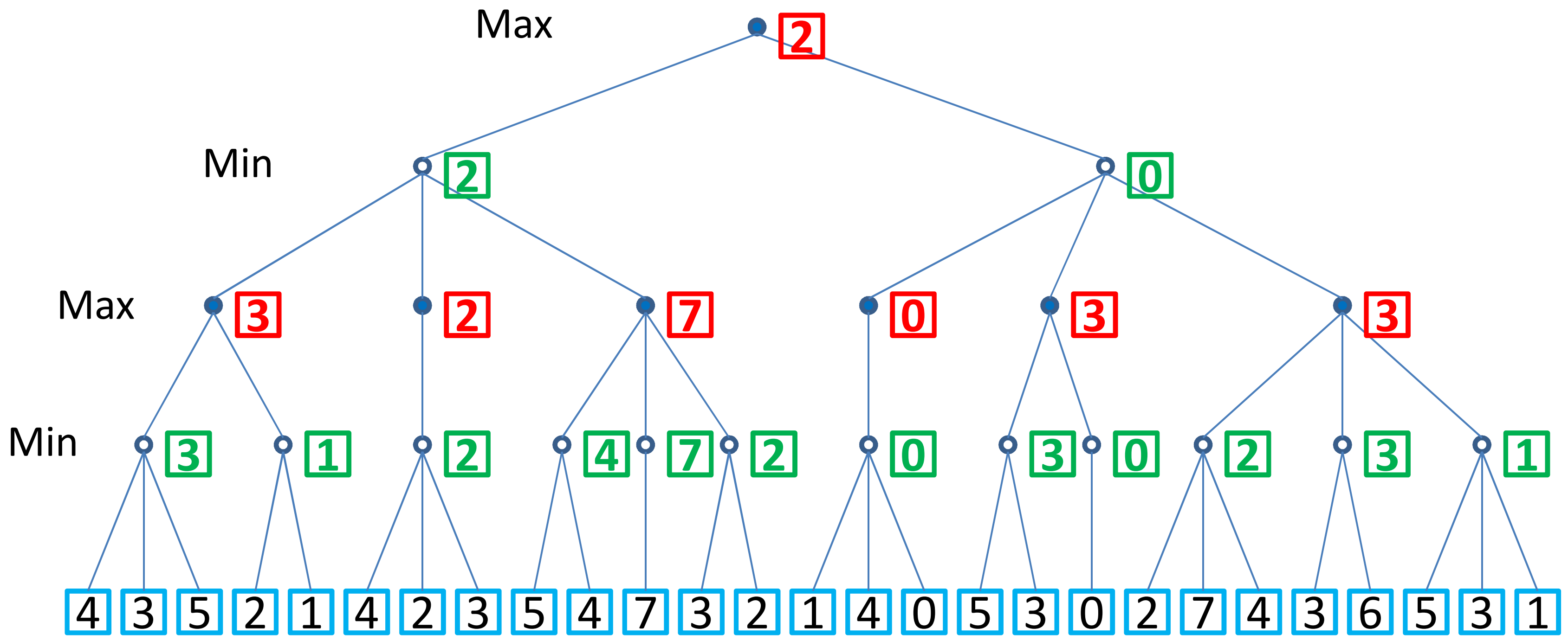


MiniMax & Constraint Processing: MiniMax Algorithm

OPTIMIZING $\alpha\beta$-PRUNING

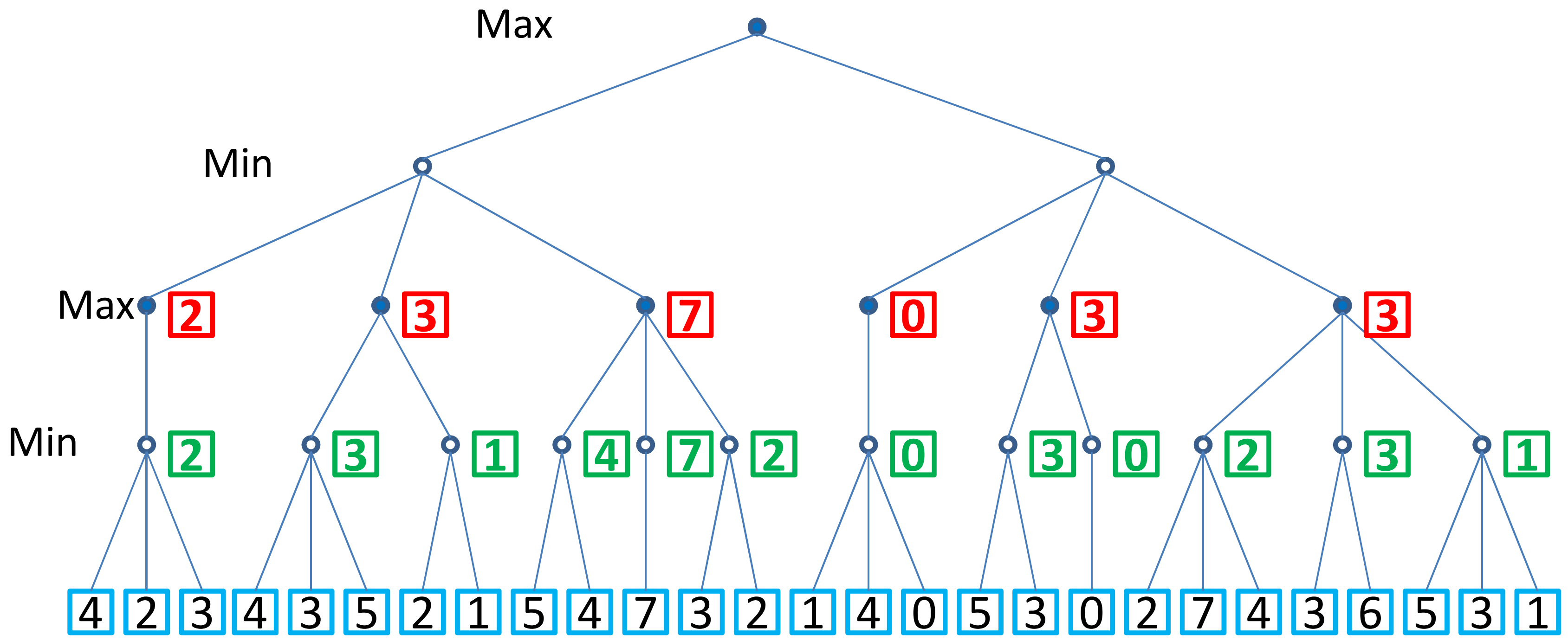
Optimizing $\alpha\beta$-Pruning

- **Best case:** Each layer best node left-to-right



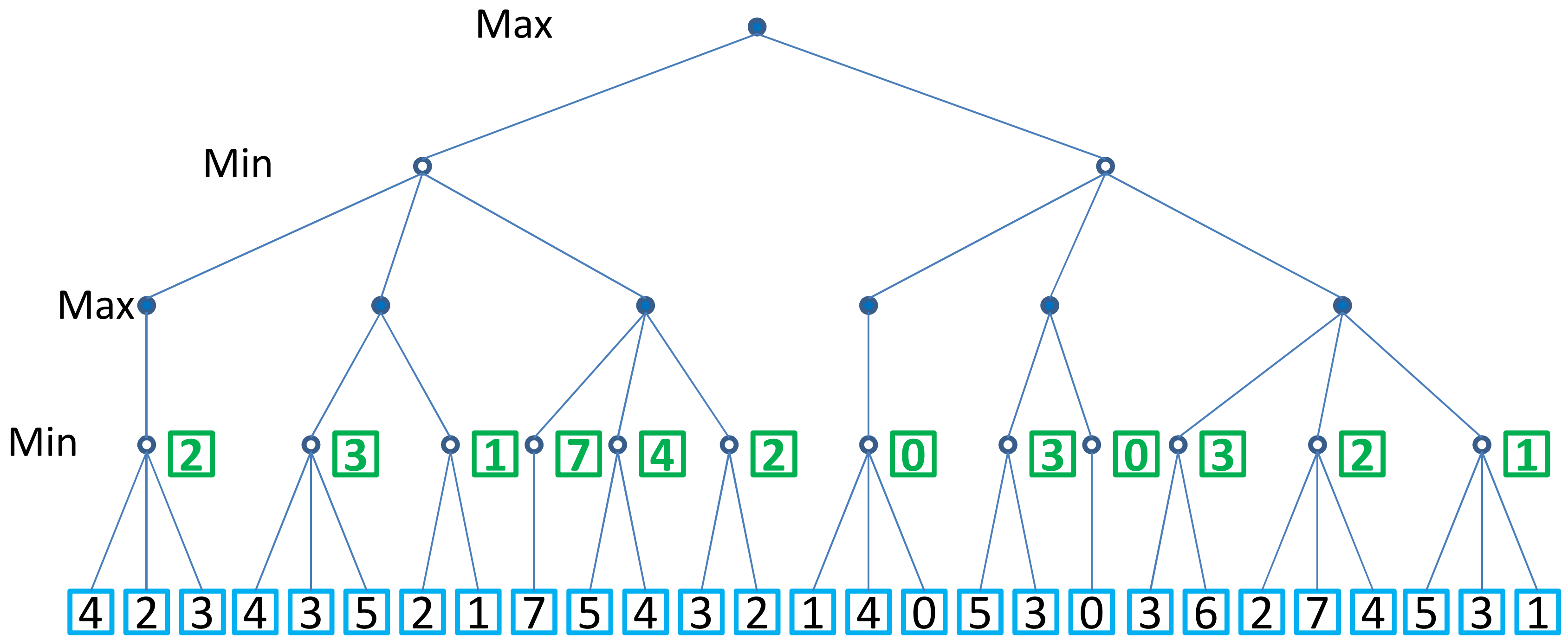
Optimizing $\alpha\beta$-Pruning

- **Best case:** Each layer best node left-to-right



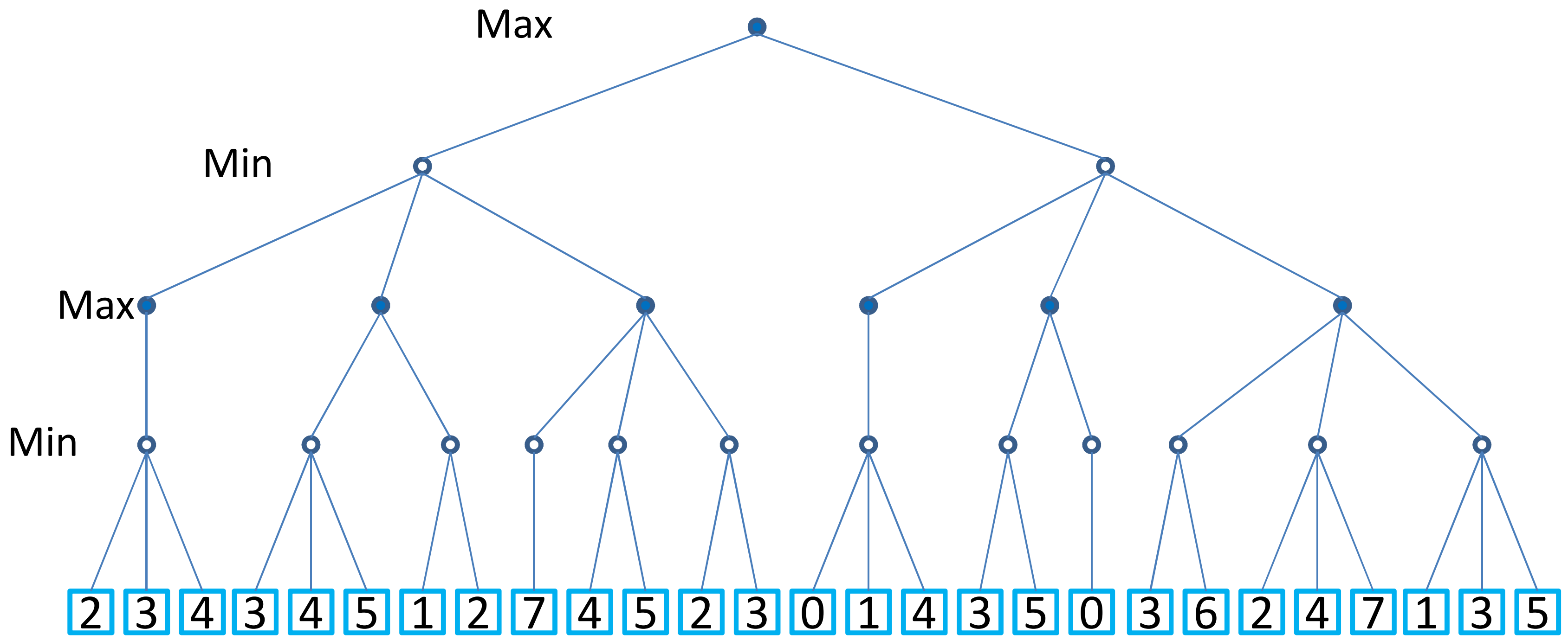
Optimizing $\alpha\beta$-Pruning

- **Best case:** Each layer best node left-to-right



Optimizing $\alpha\beta$-Pruning

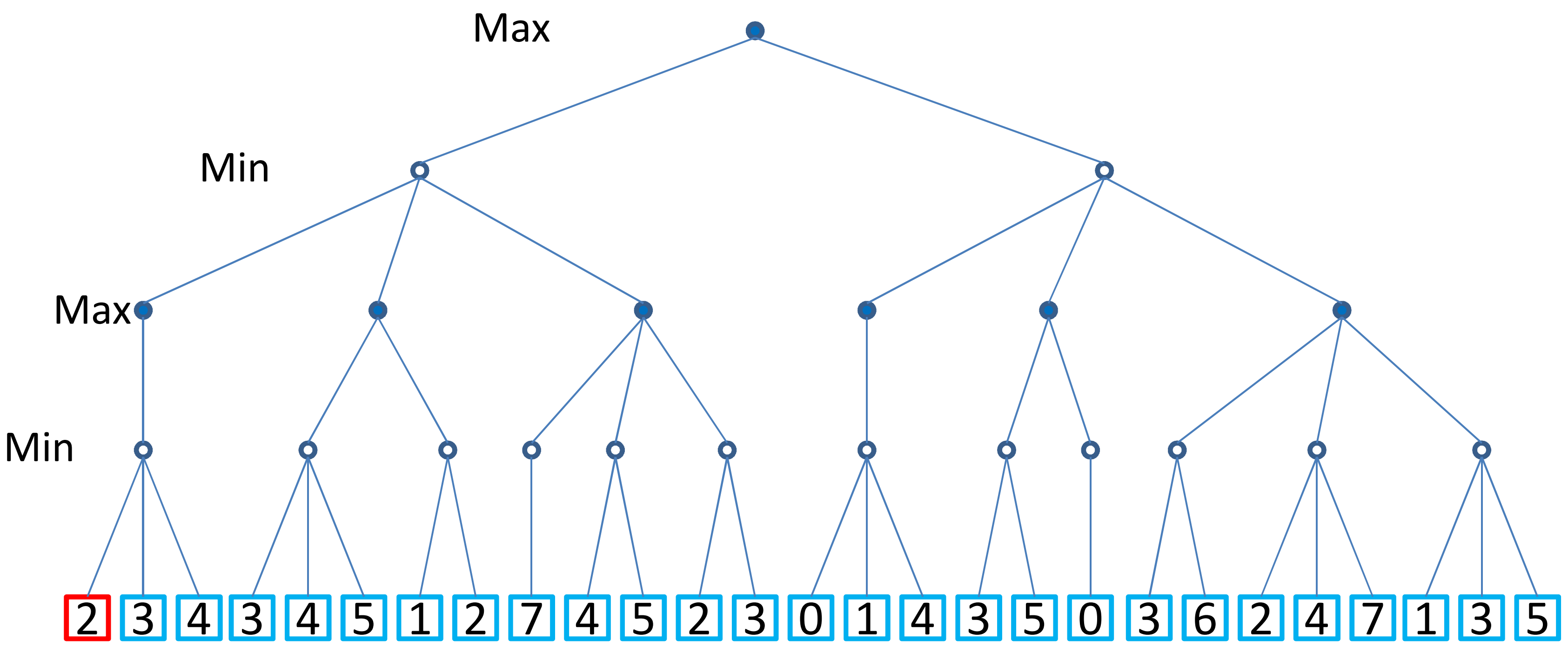
- **Best case:** Each layer best node left-to-right



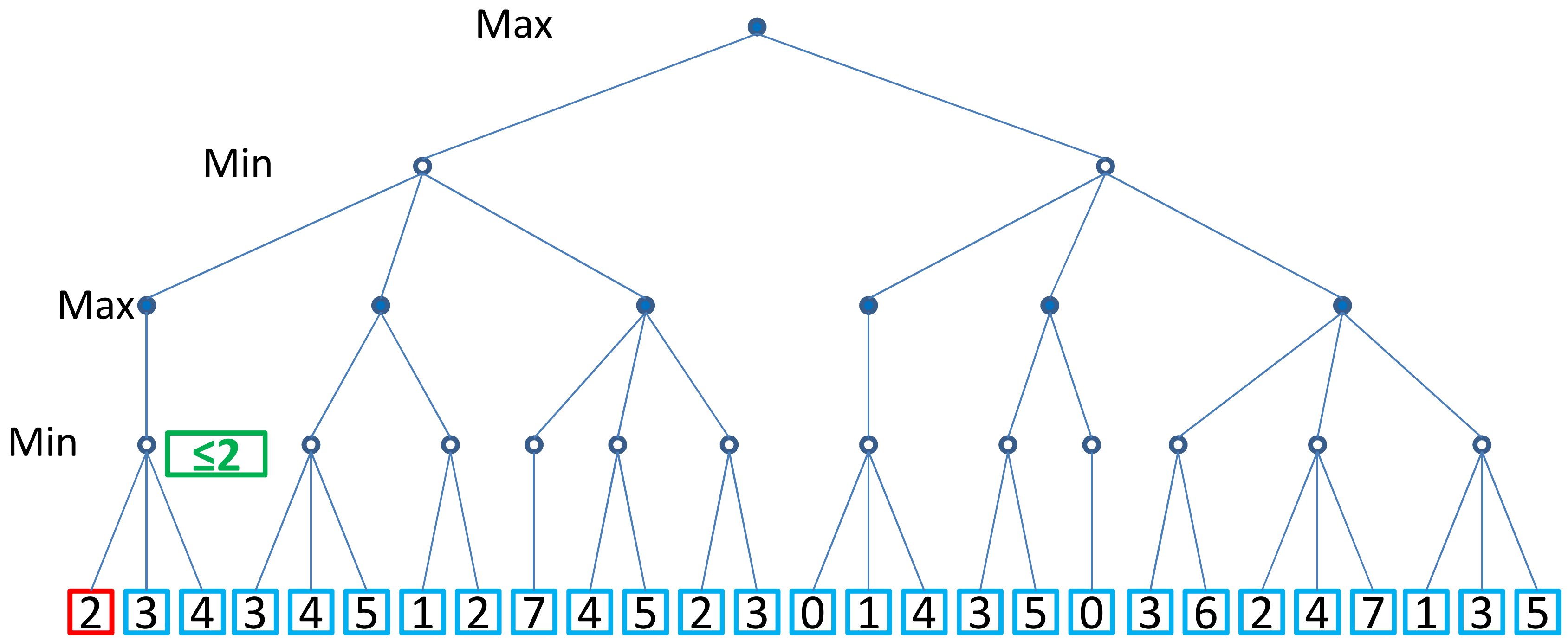
MiniMax & Constraint Processing: MiniMax Algorithm

MINIMAX WITH $\alpha\beta$-PRUNING

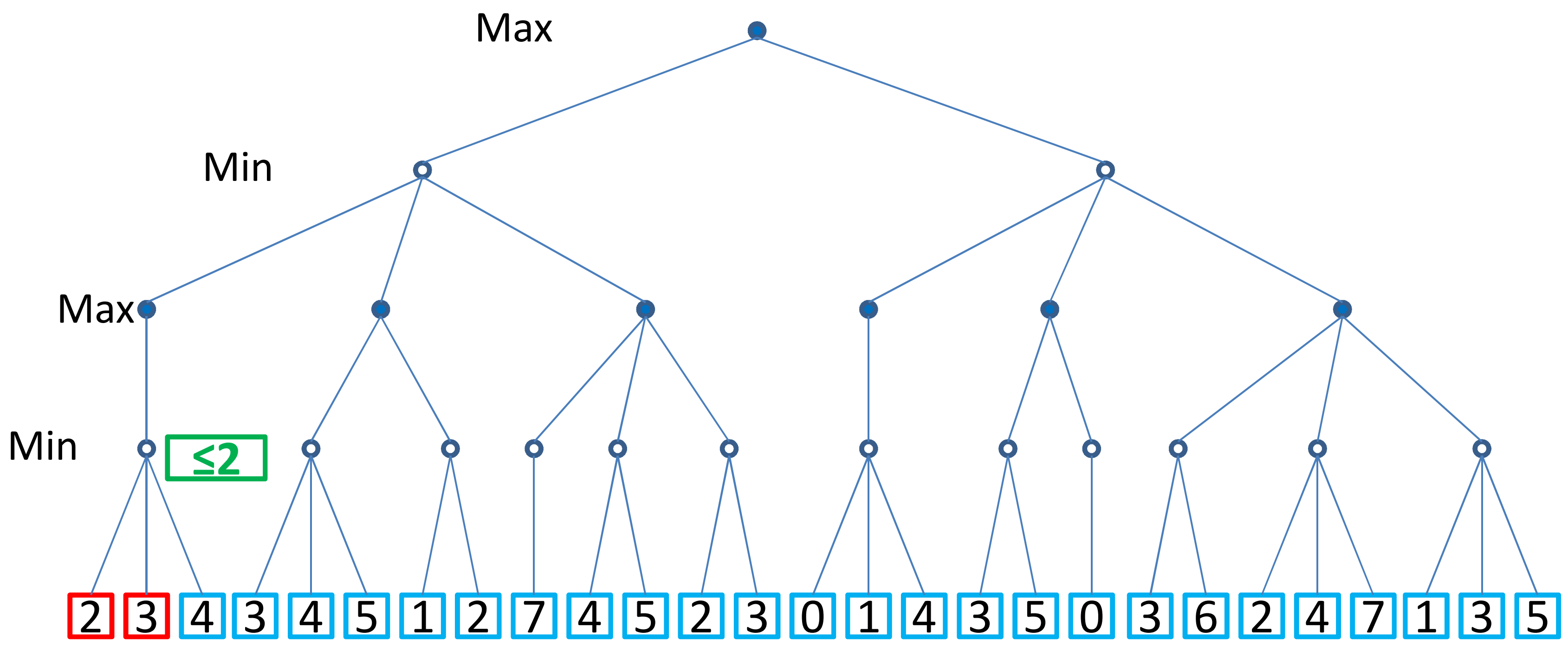
Minimax with $\alpha\beta$-Pruning



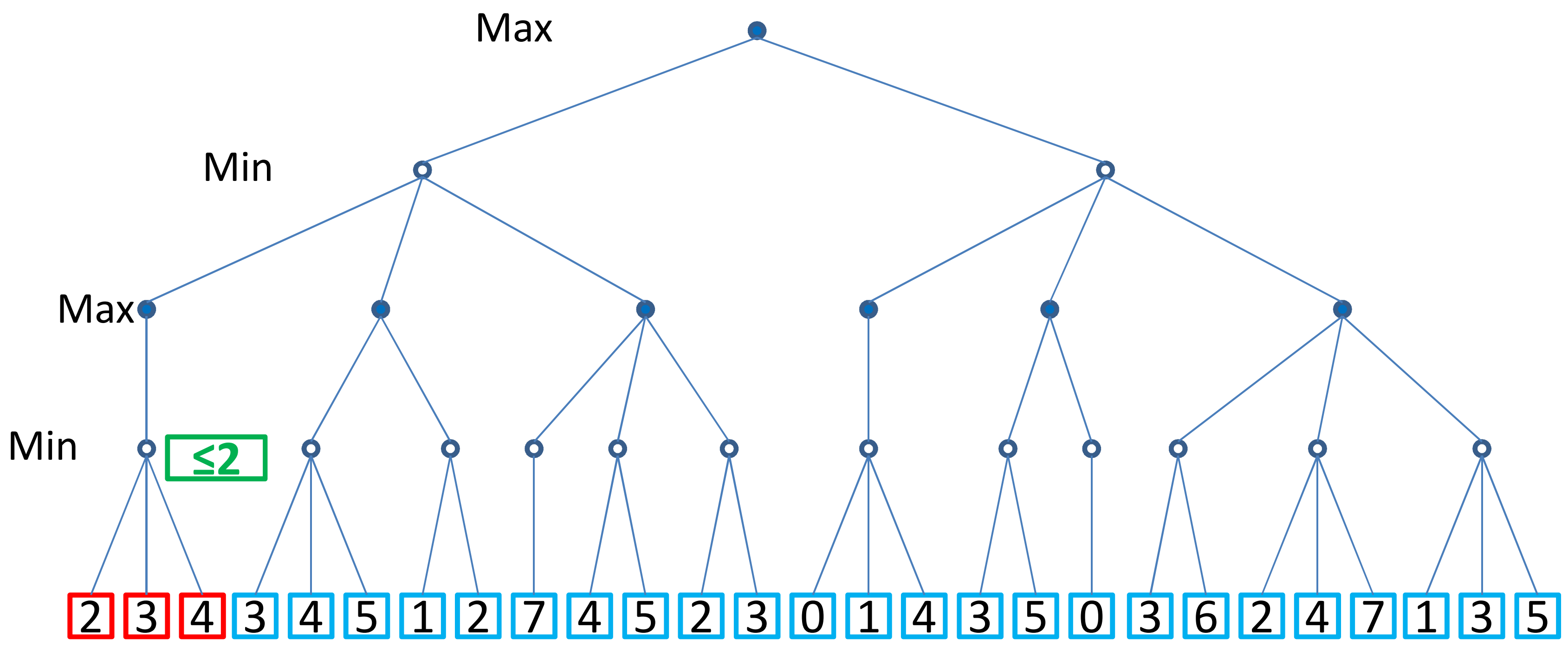
Minimax with $\alpha\beta$-Pruning



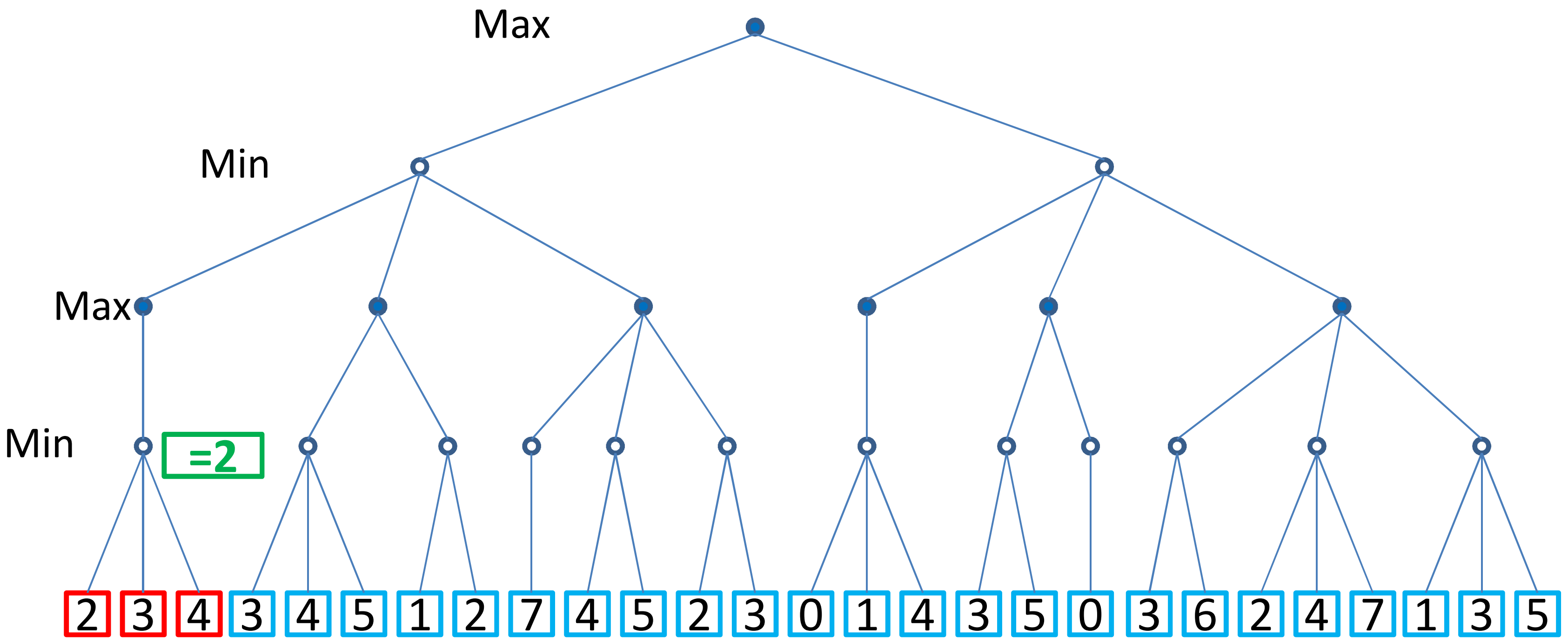
Minimax with $\alpha\beta$-Pruning



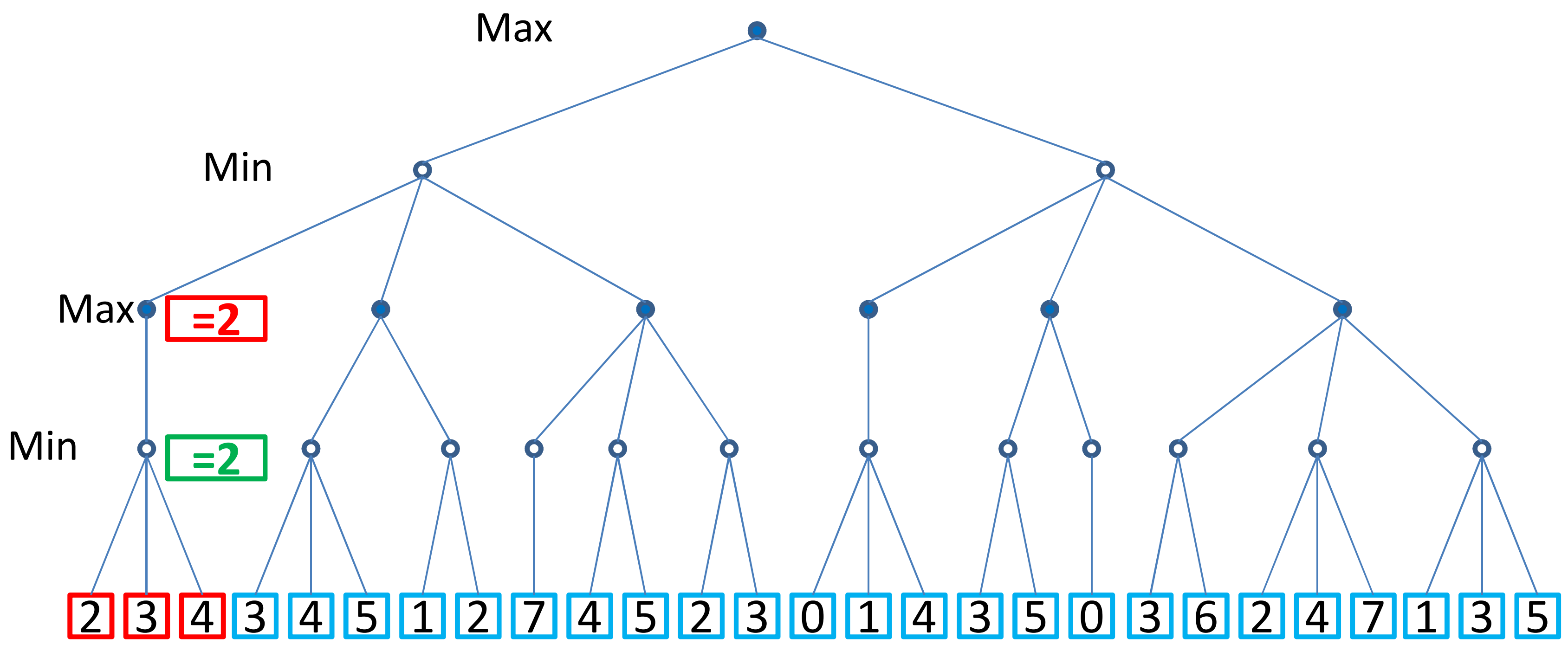
Minimax with $\alpha\beta$-Pruning



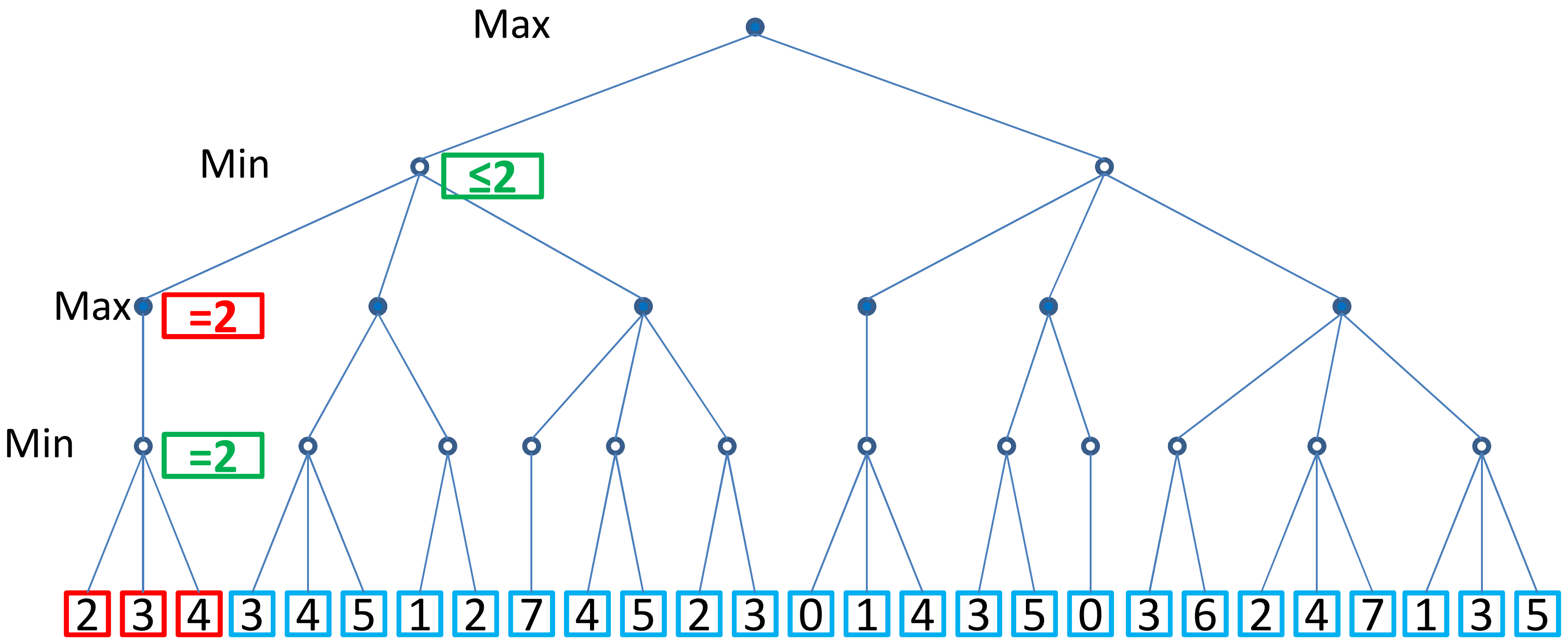
Minimax with $\alpha\beta$-Pruning



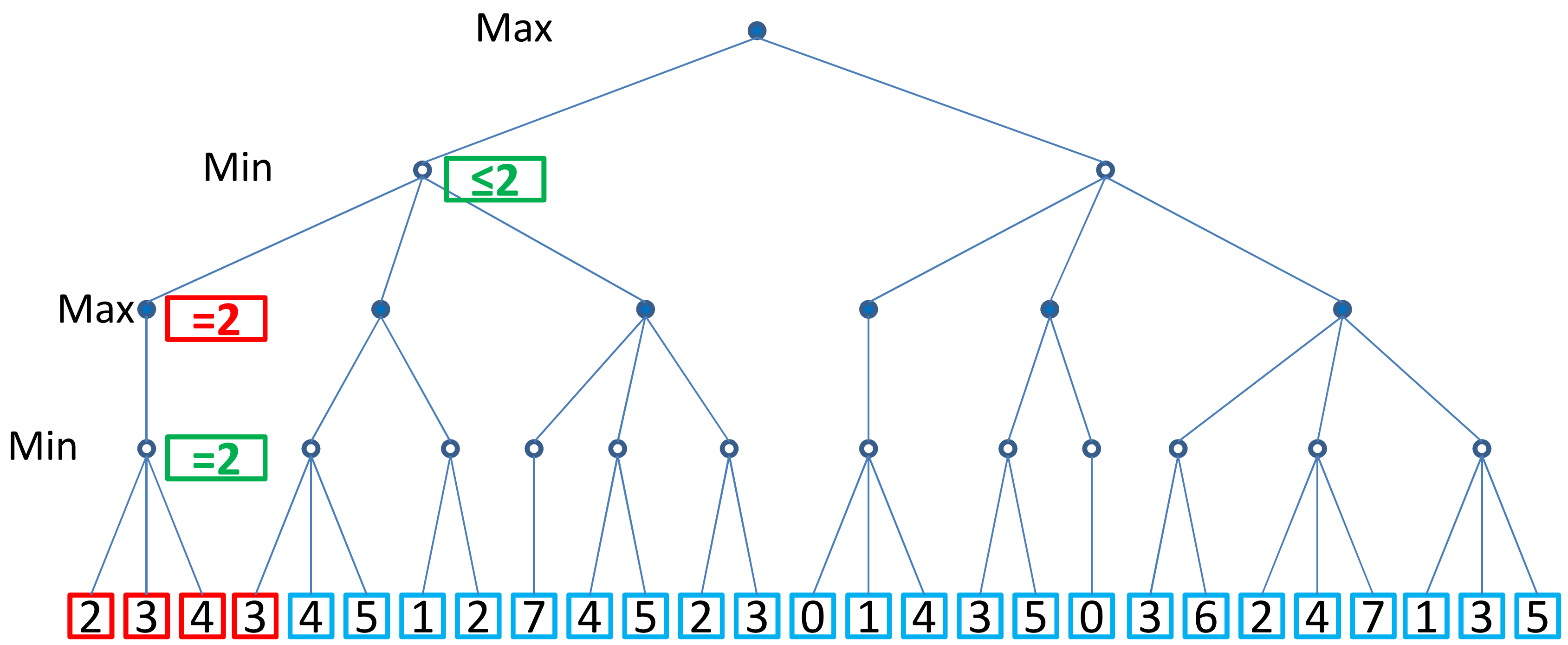
Minimax with $\alpha\beta$-Pruning



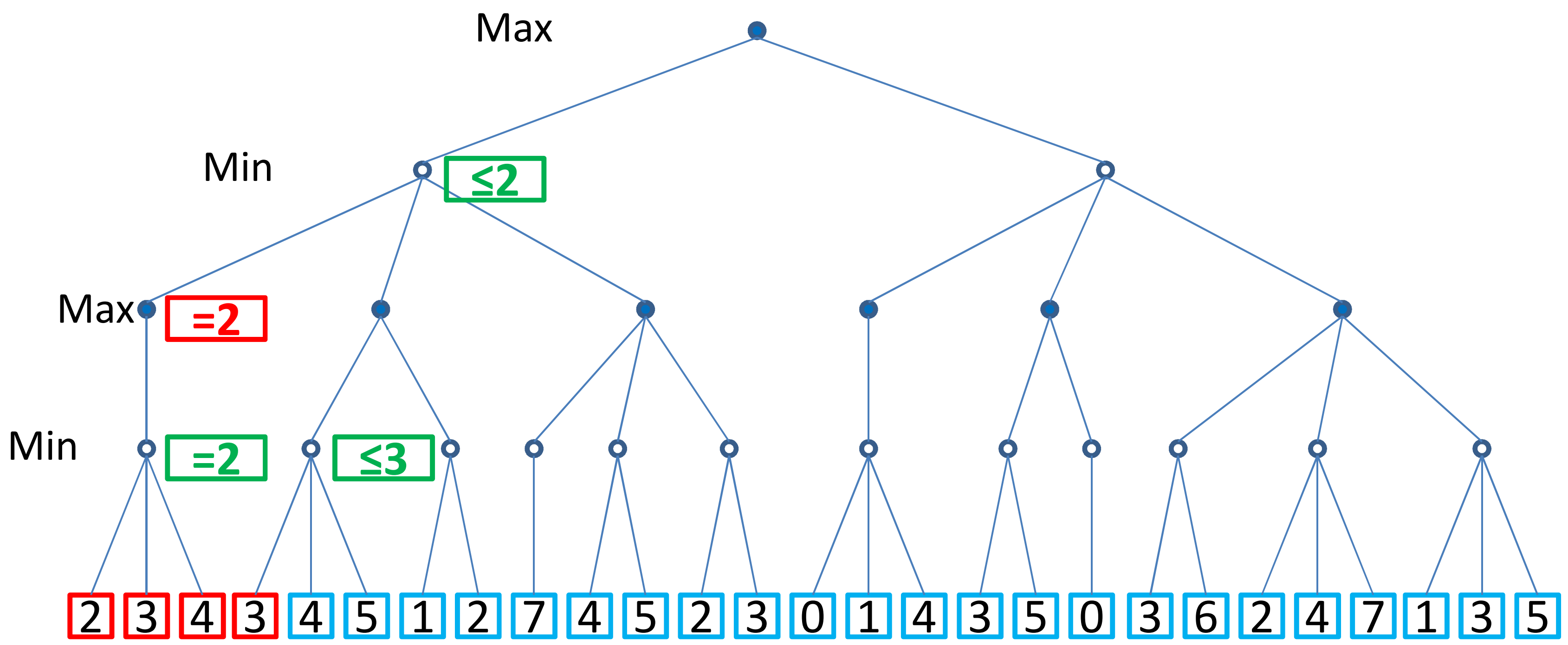
Minimax with $\alpha\beta$-Pruning



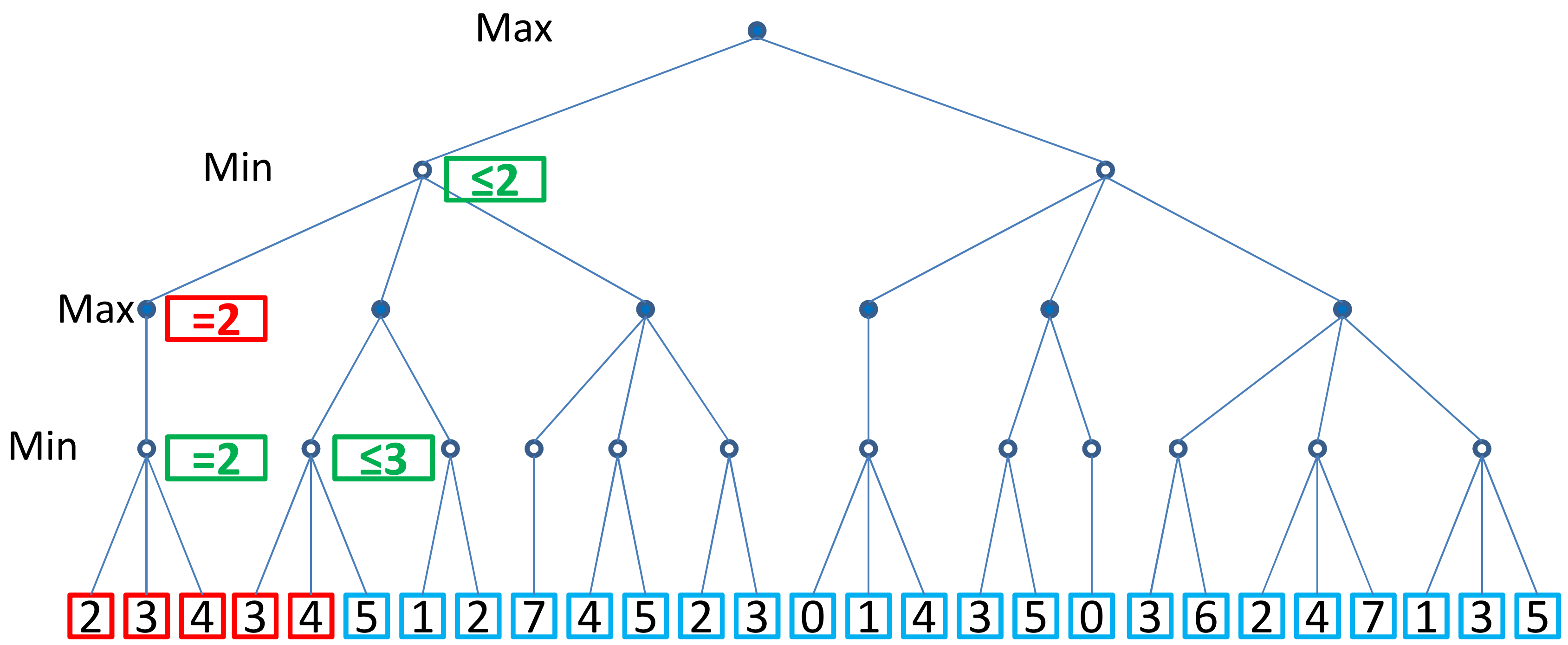
Minimax with $\alpha\beta$-Pruning



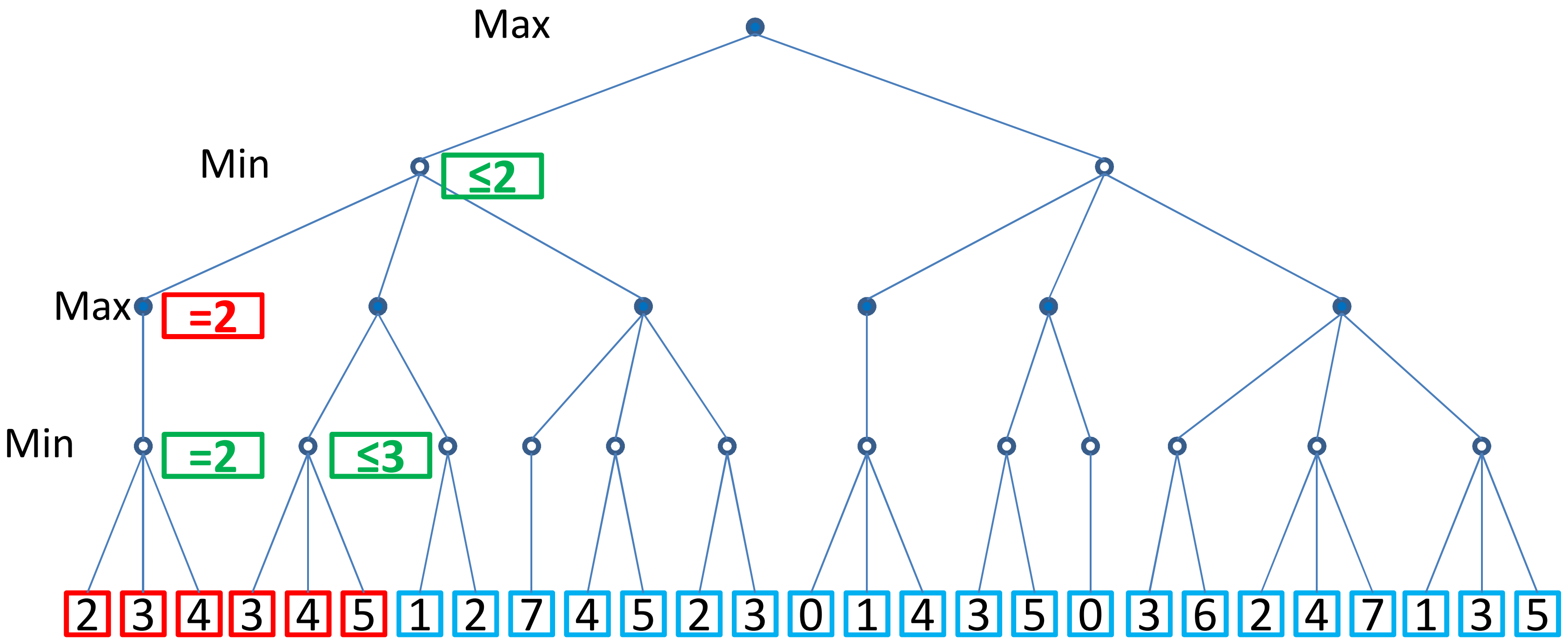
Minimax with $\alpha\beta$-Pruning



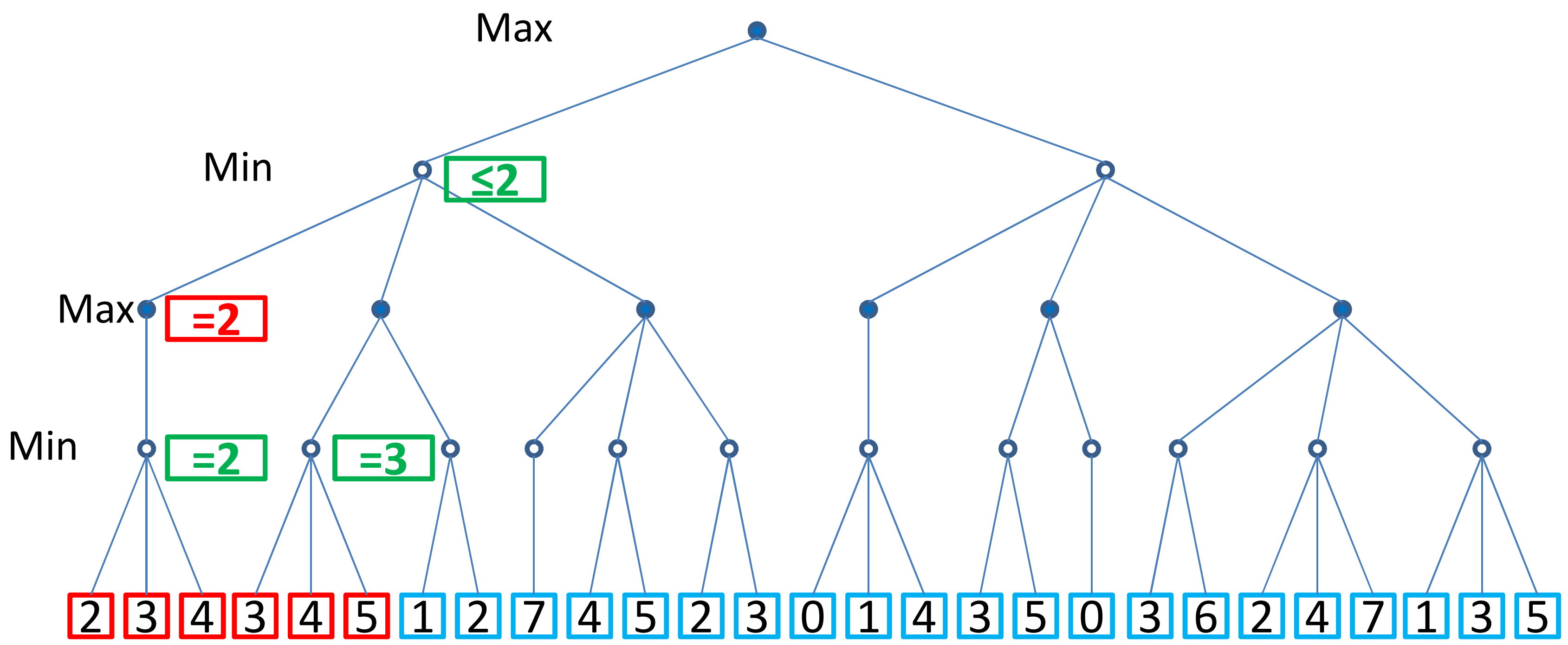
Minimax with $\alpha\beta$-Pruning



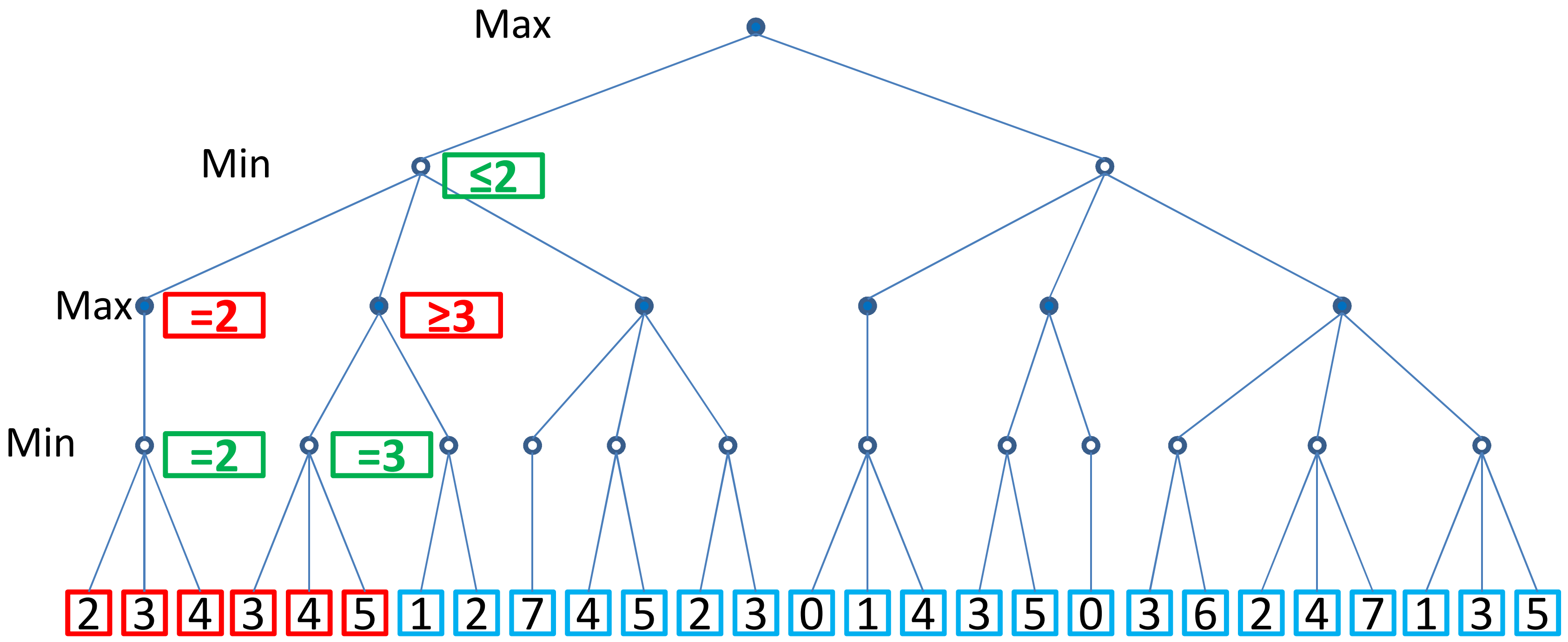
Minimax with $\alpha\beta$-Pruning



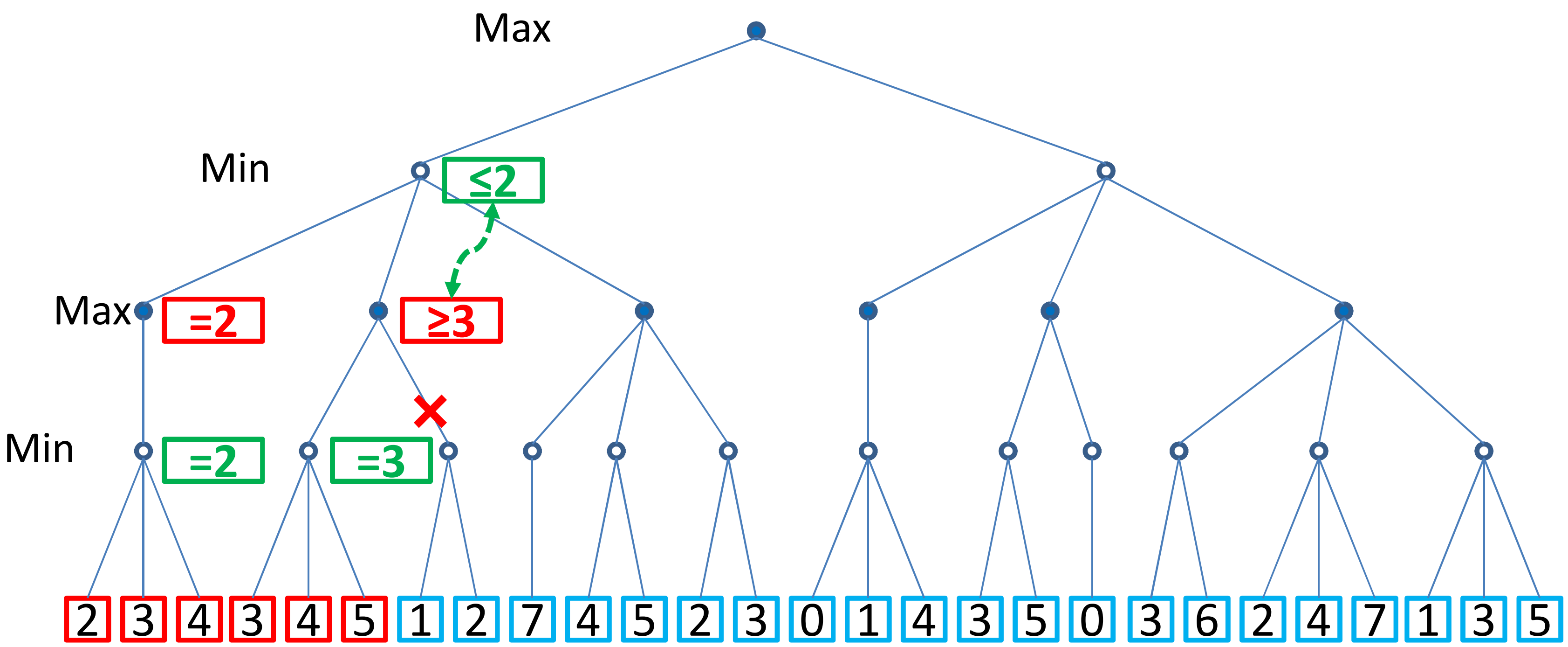
Minimax with $\alpha\beta$-Pruning



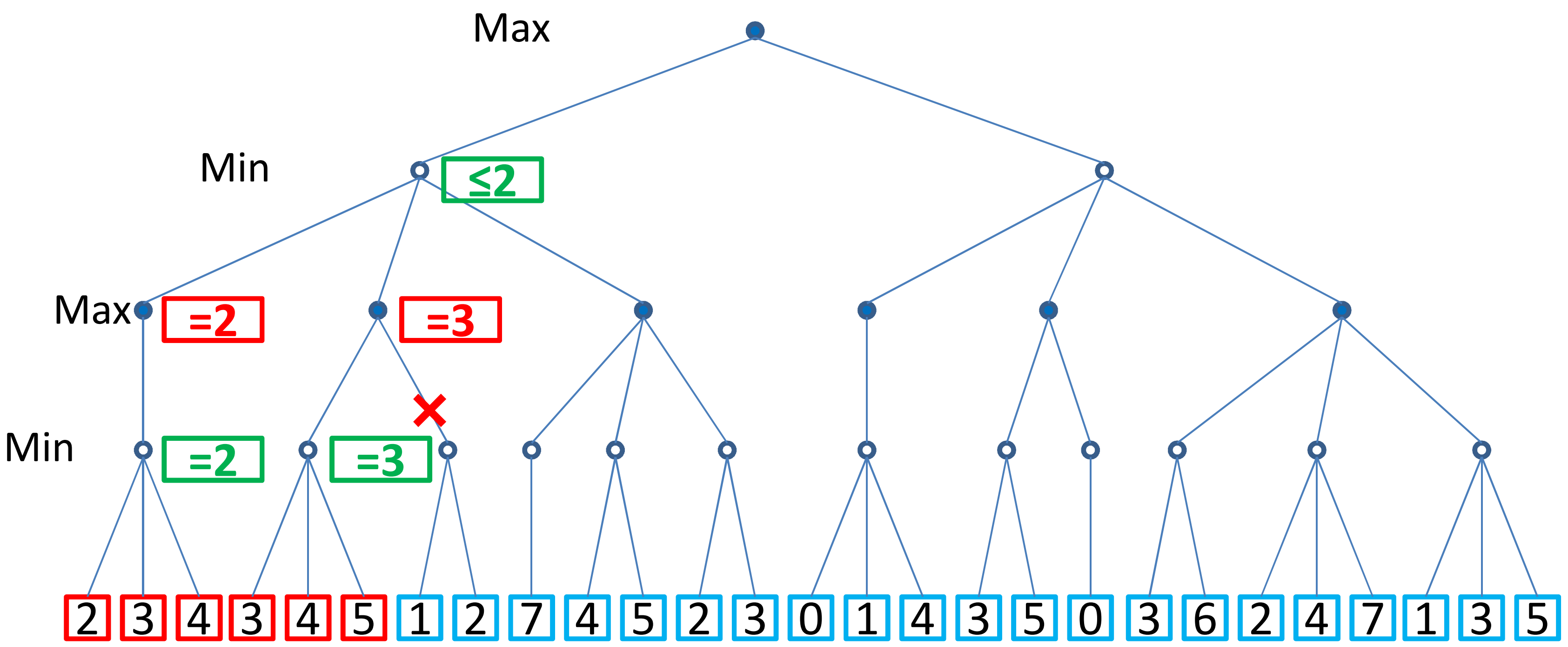
Minimax with $\alpha\beta$-Pruning



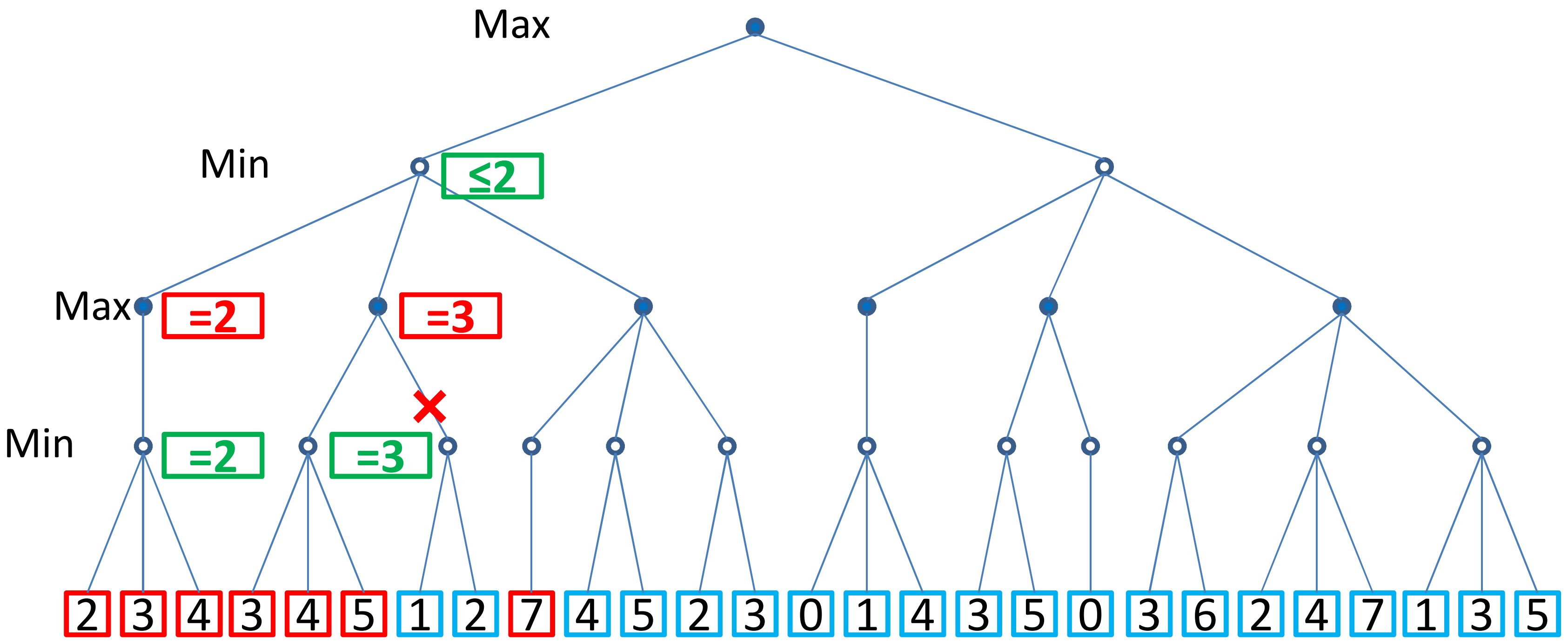
Minimax with $\alpha\beta$-Pruning



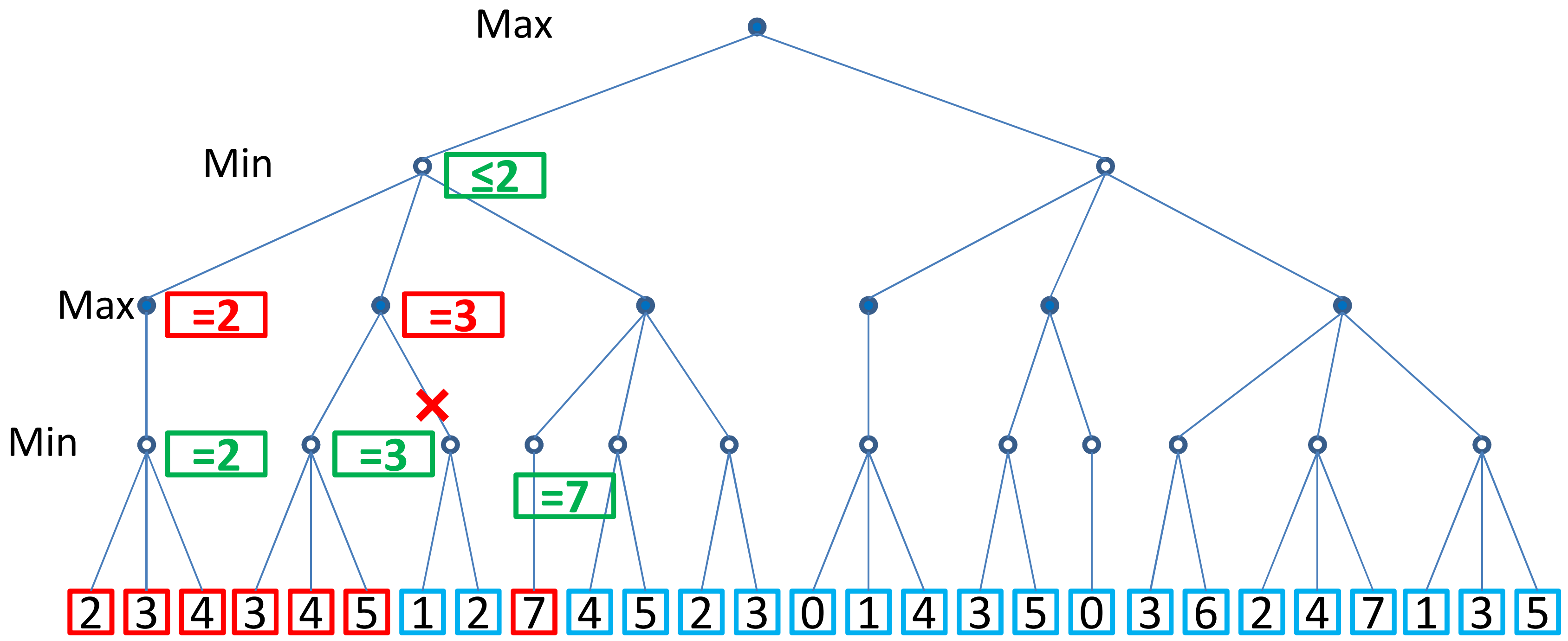
Minimax with $\alpha\beta$-Pruning



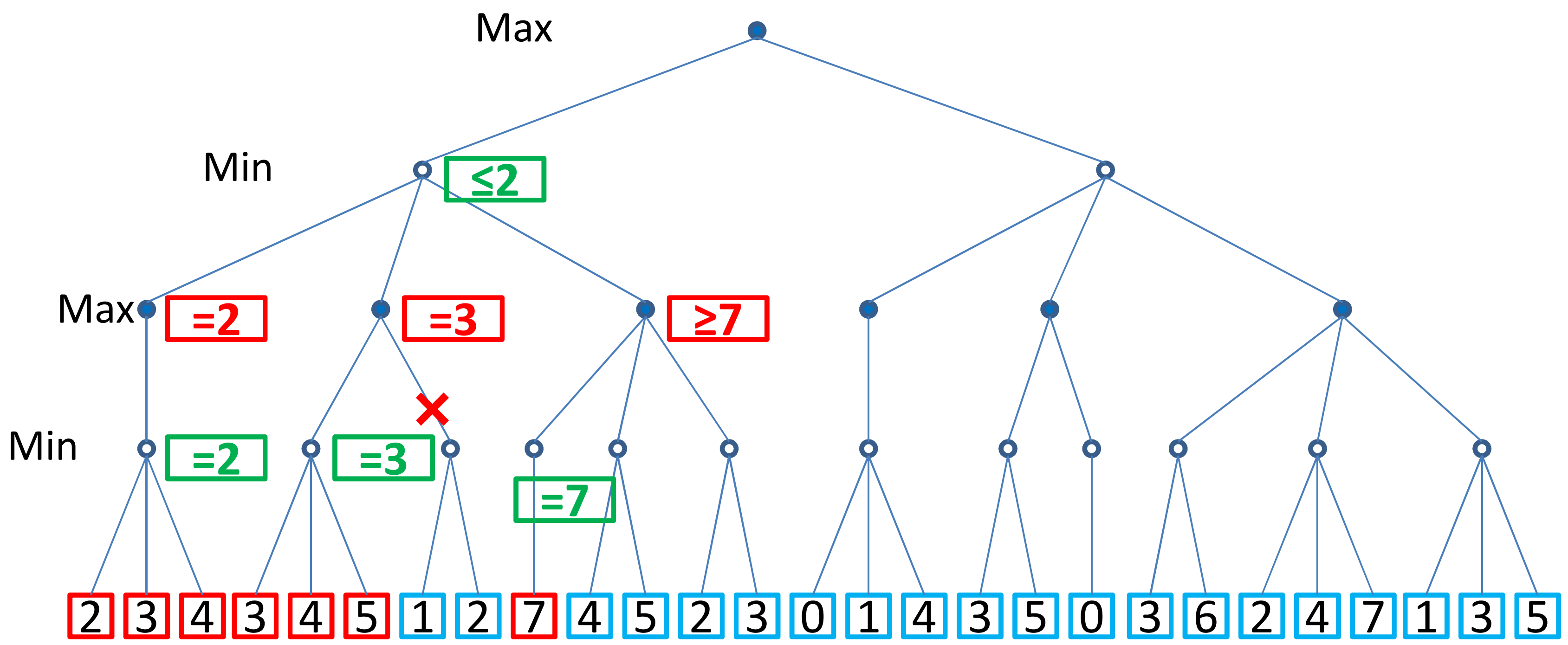
Minimax with $\alpha\beta$-Pruning



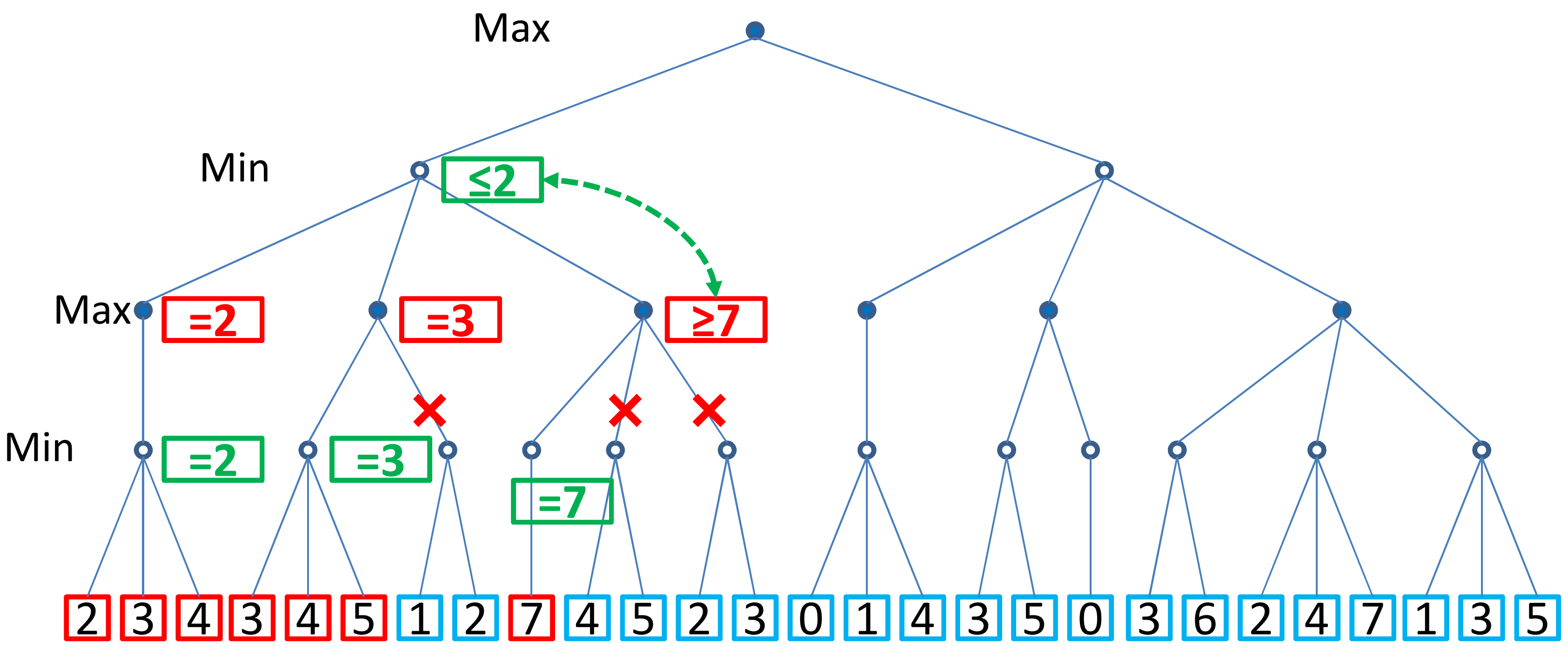
Minimax with $\alpha\beta$-Pruning



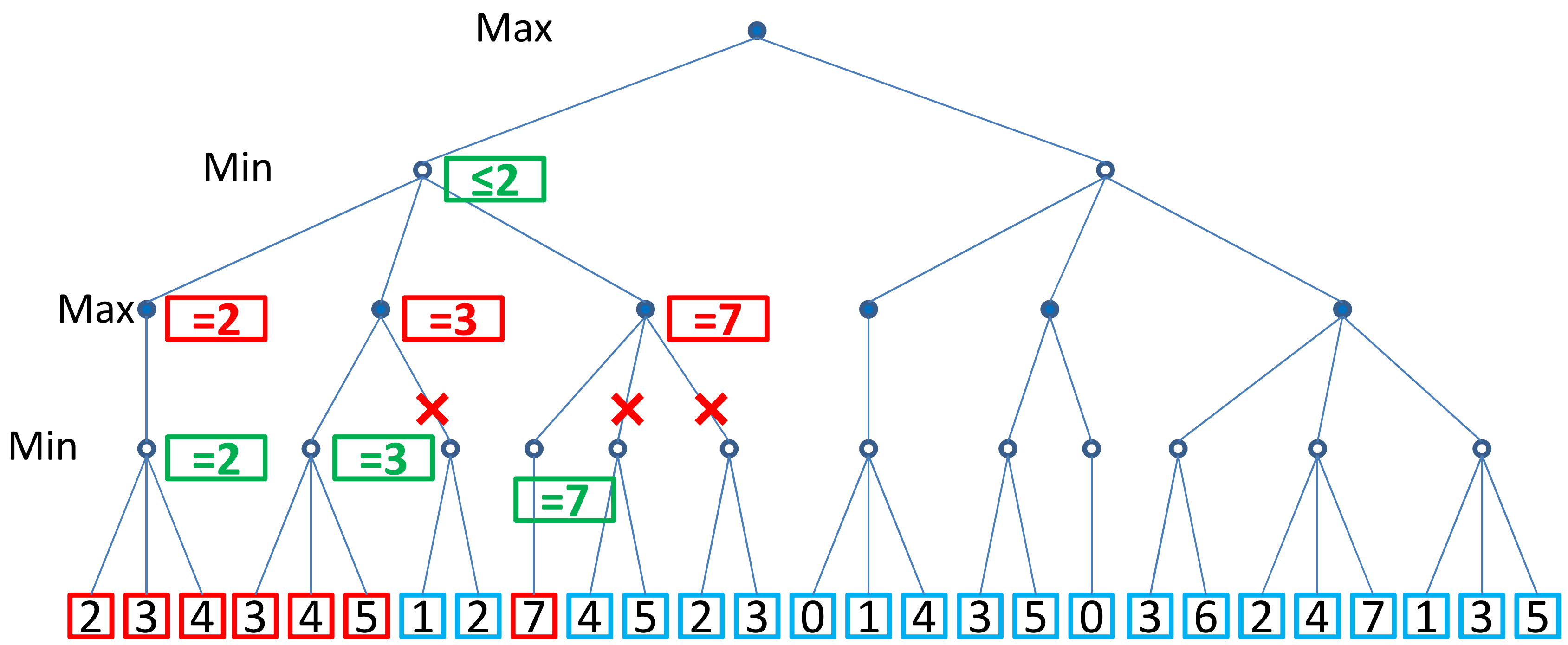
Minimax with $\alpha\beta$-Pruning



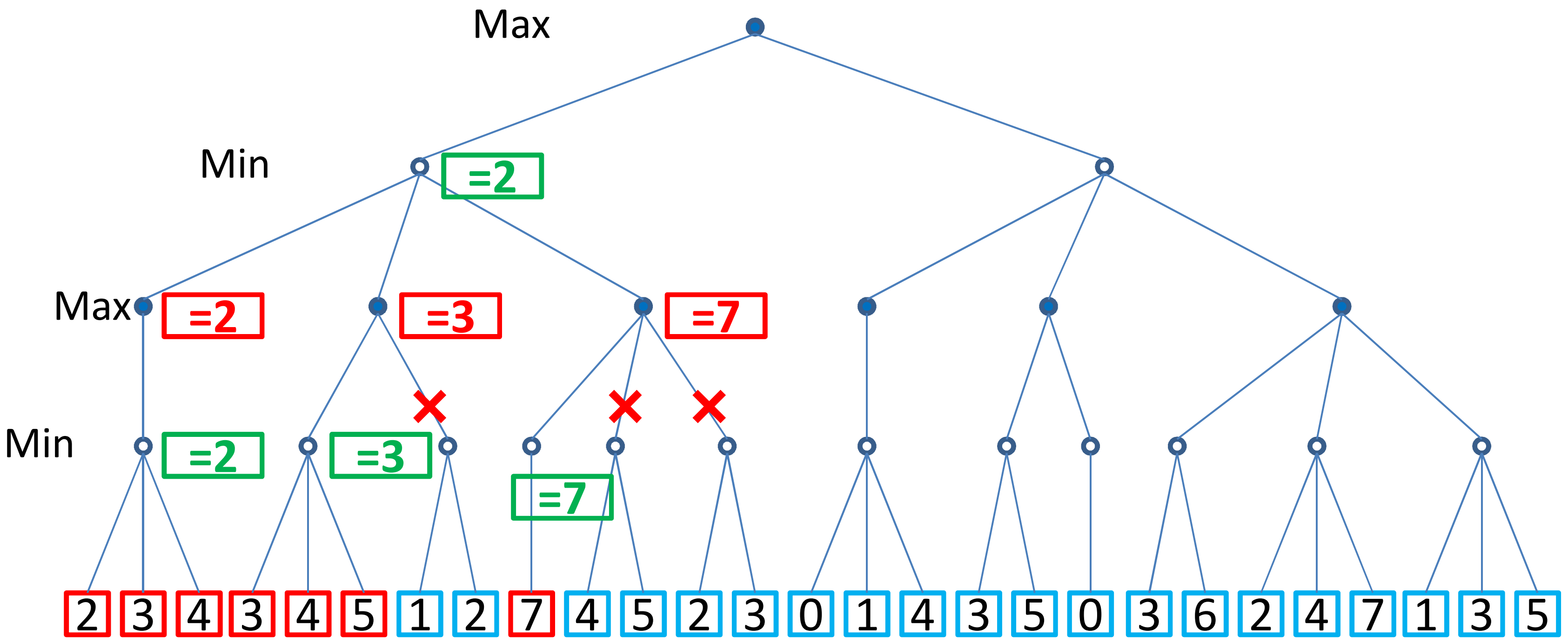
Minimax with $\alpha\beta$-Pruning



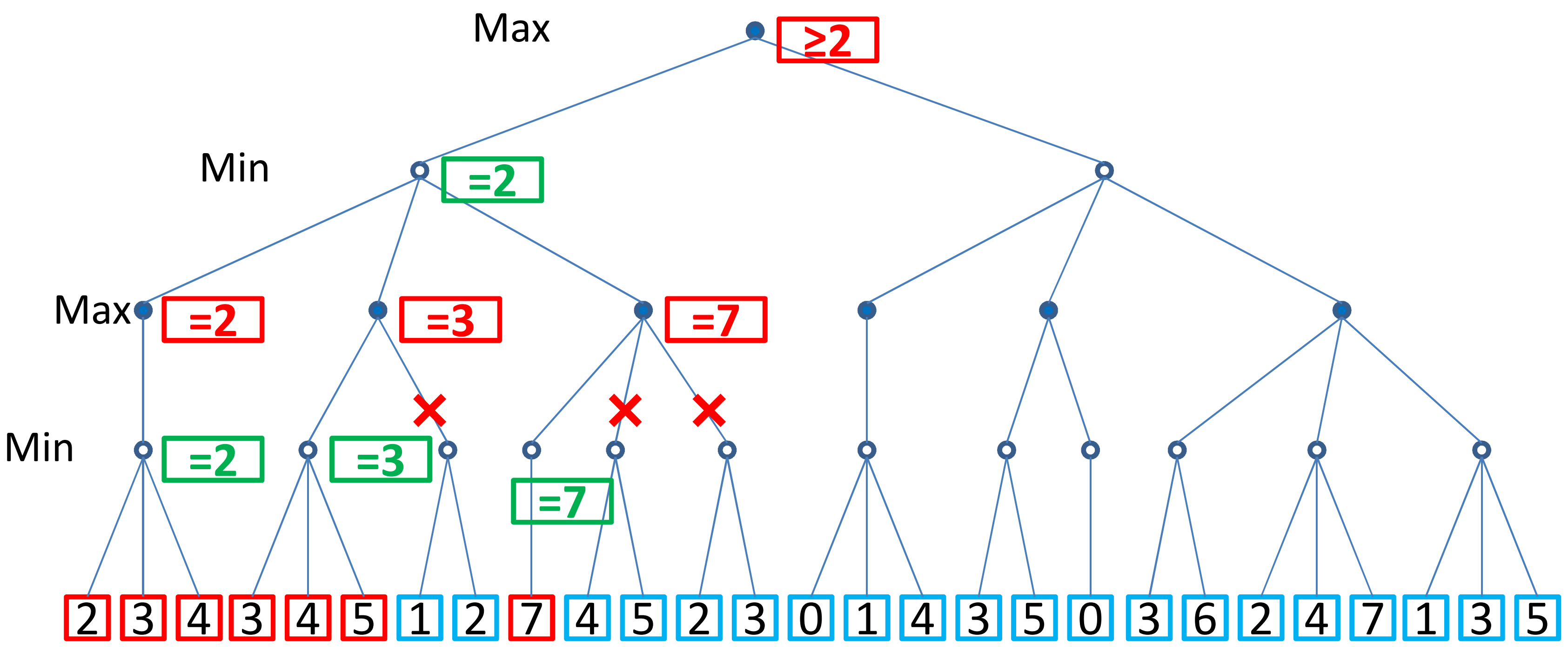
Minimax with $\alpha\beta$-Pruning



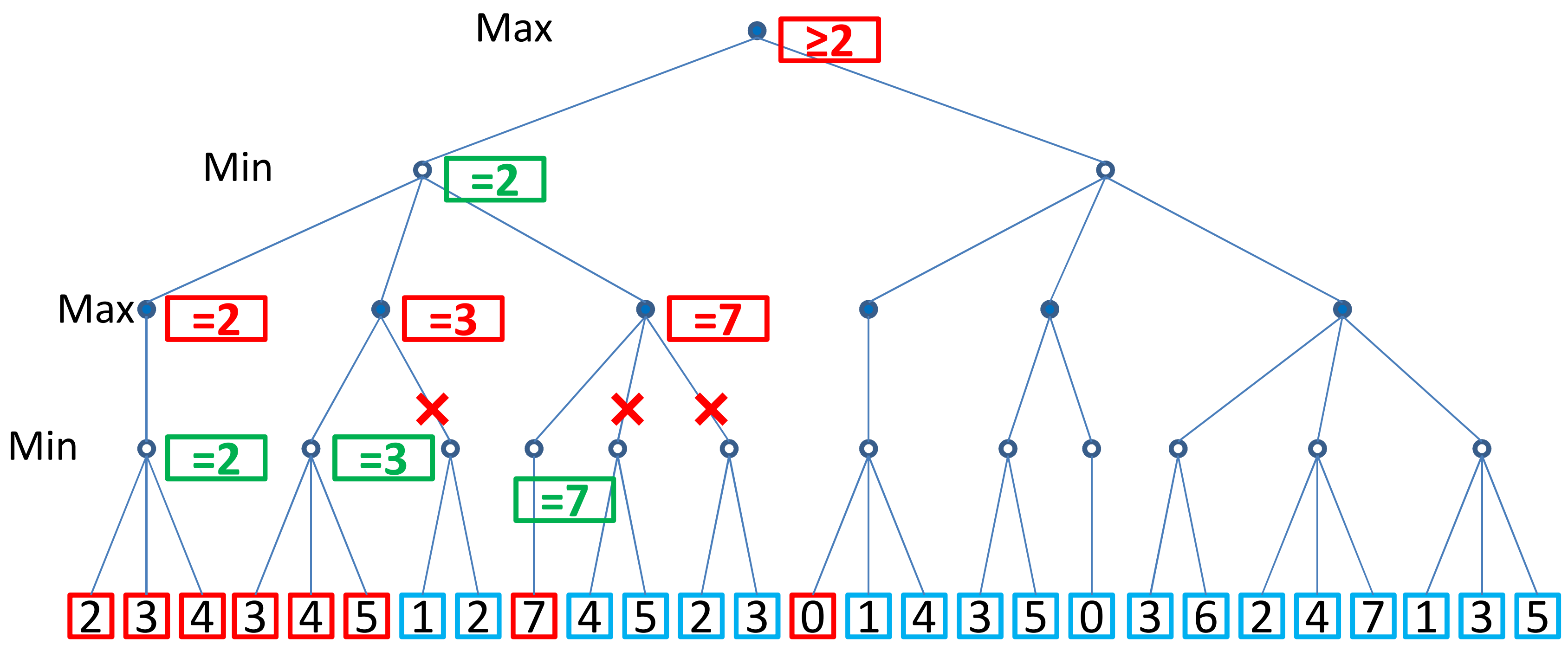
Minimax with $\alpha\beta$-Pruning



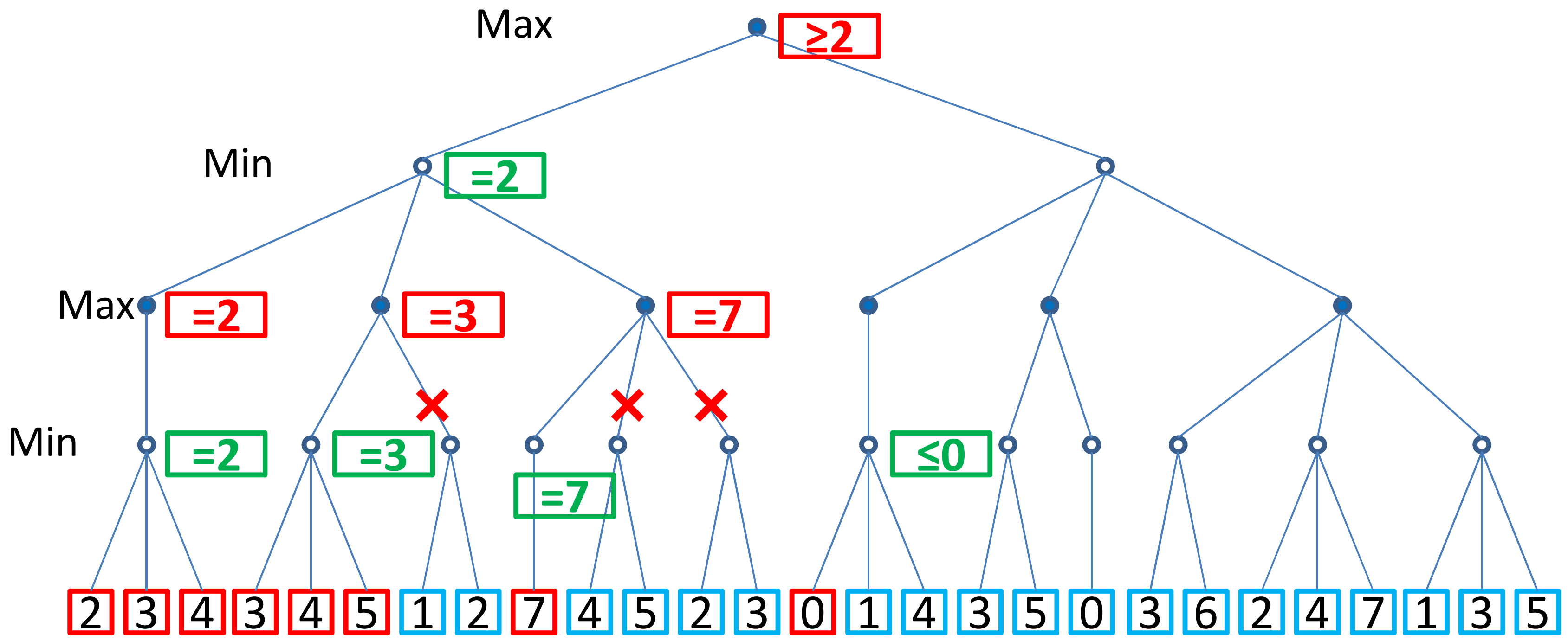
Minimax with $\alpha\beta$-Pruning



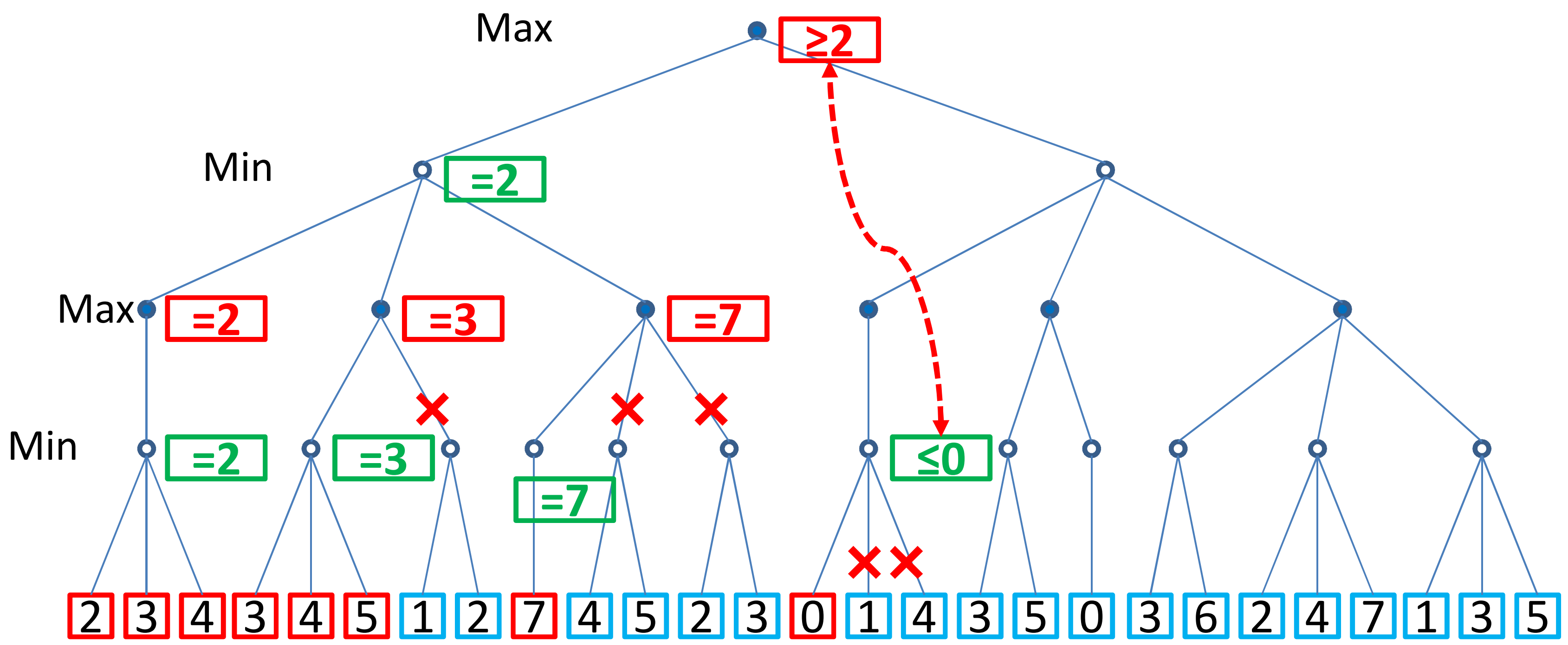
Minimax with $\alpha\beta$-Pruning



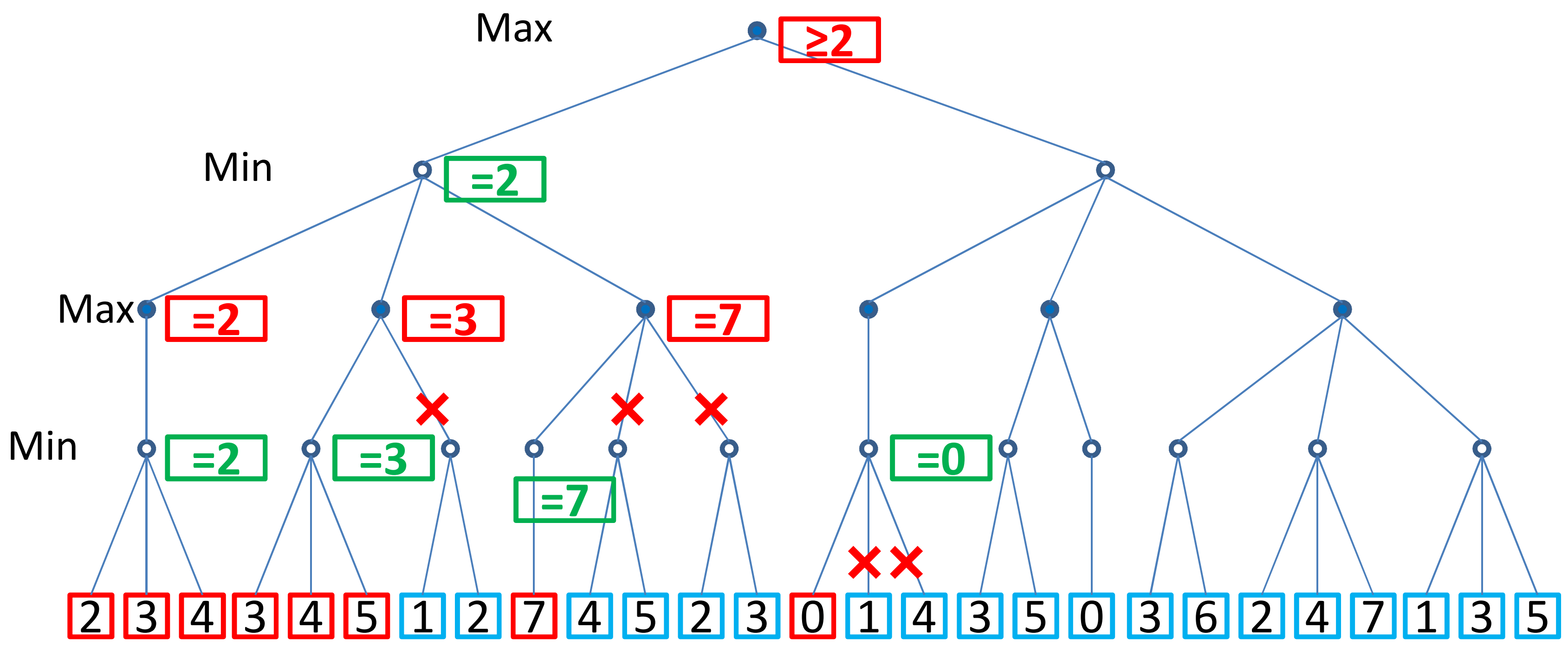
Minimax with $\alpha\beta$-Pruning



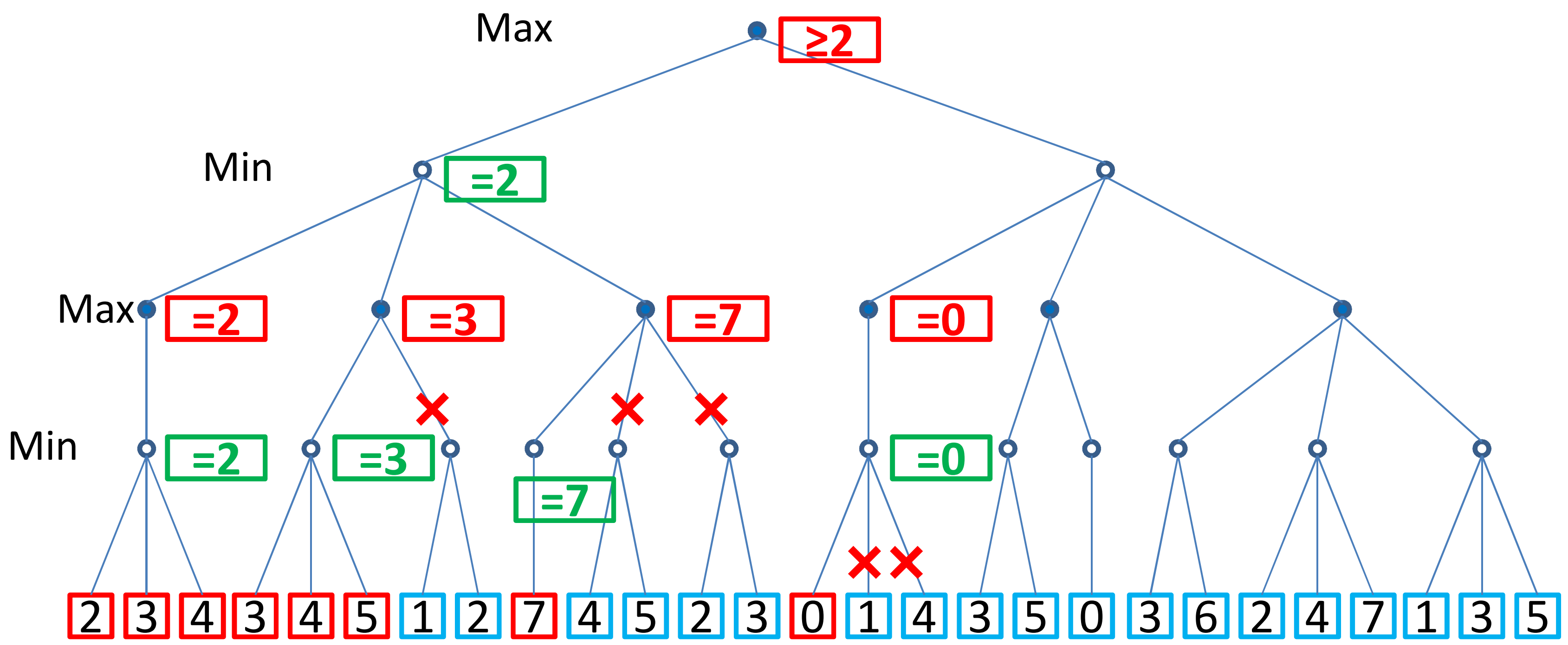
Minimax with $\alpha\beta$-Pruning



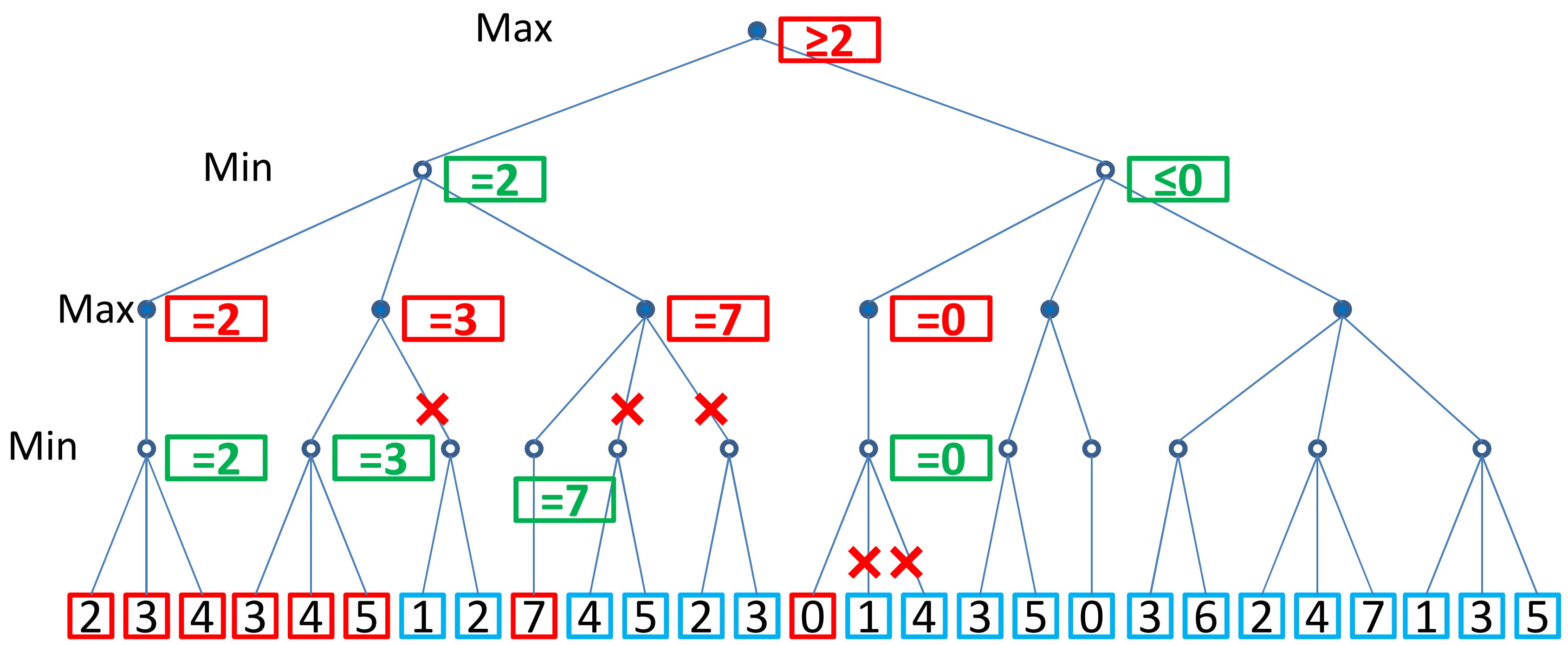
Minimax with $\alpha\beta$-Pruning



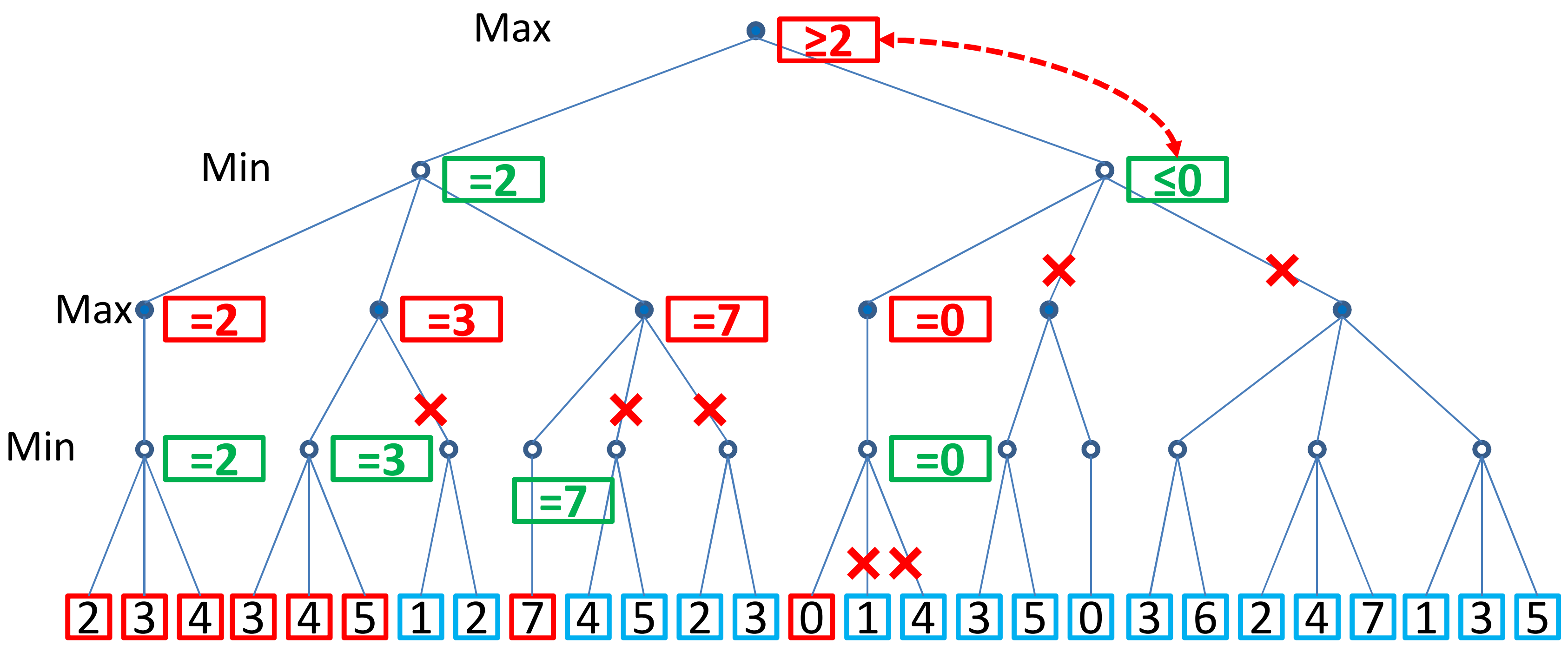
Minimax with $\alpha\beta$-Pruning



Minimax with $\alpha\beta$-Pruning

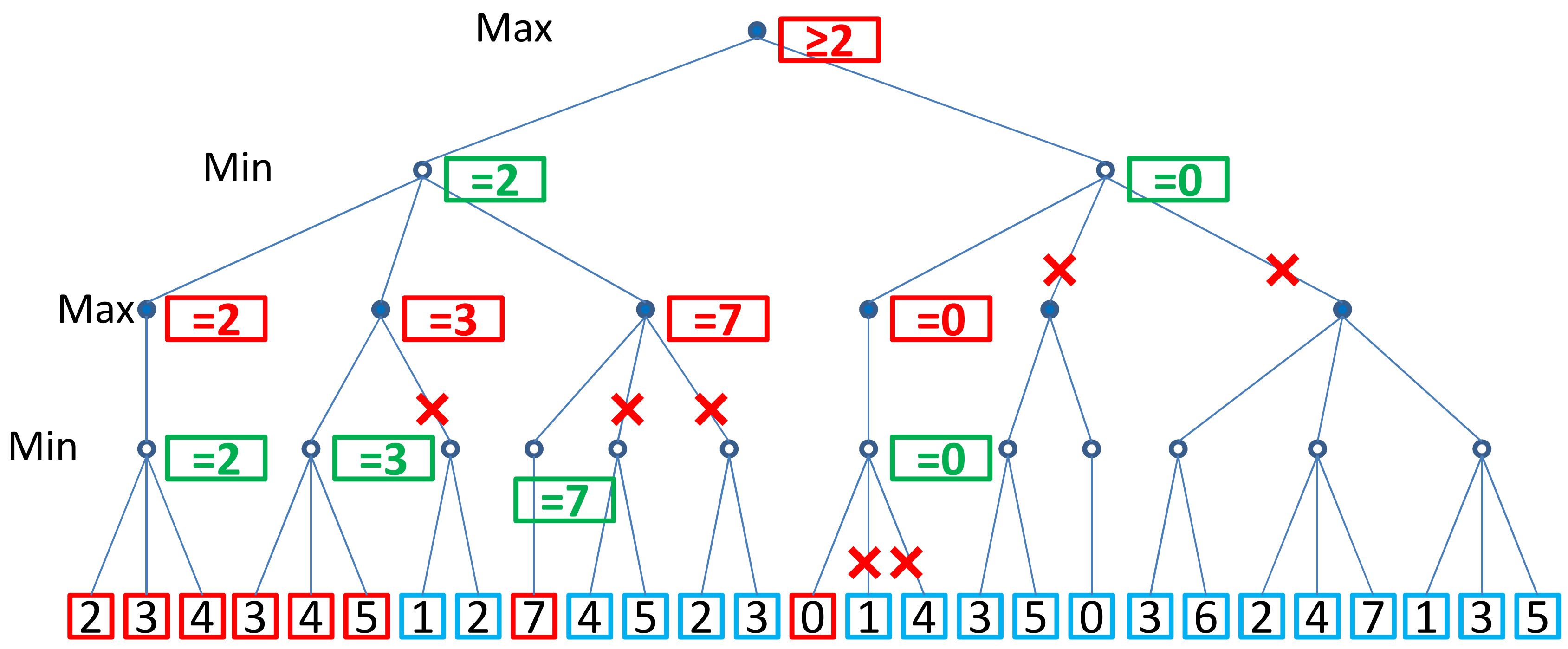


Minimax with $\alpha\beta$-Pruning



Minimax with $\alpha\beta$-Pruning

- **19 static evaluations saved**



Exercises: Artificial Intelligence

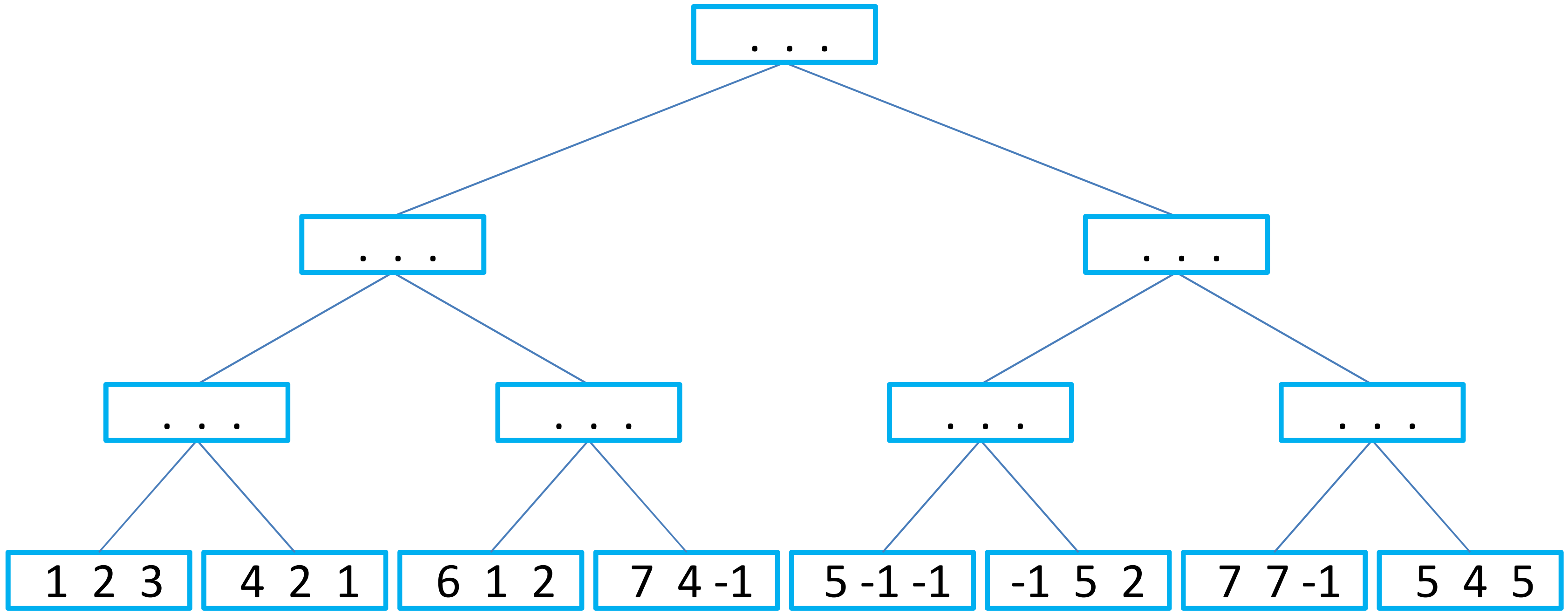
MiniMax & Constraint Processing:
MiniMax Algorithm for 3 Players

MiniMax & Constraint Processing: MiniMax Algorithm for 3 Players

PROBLEM

Problem

- Come up with a MiniMax algorithm for 3 players and apply on the figure below.

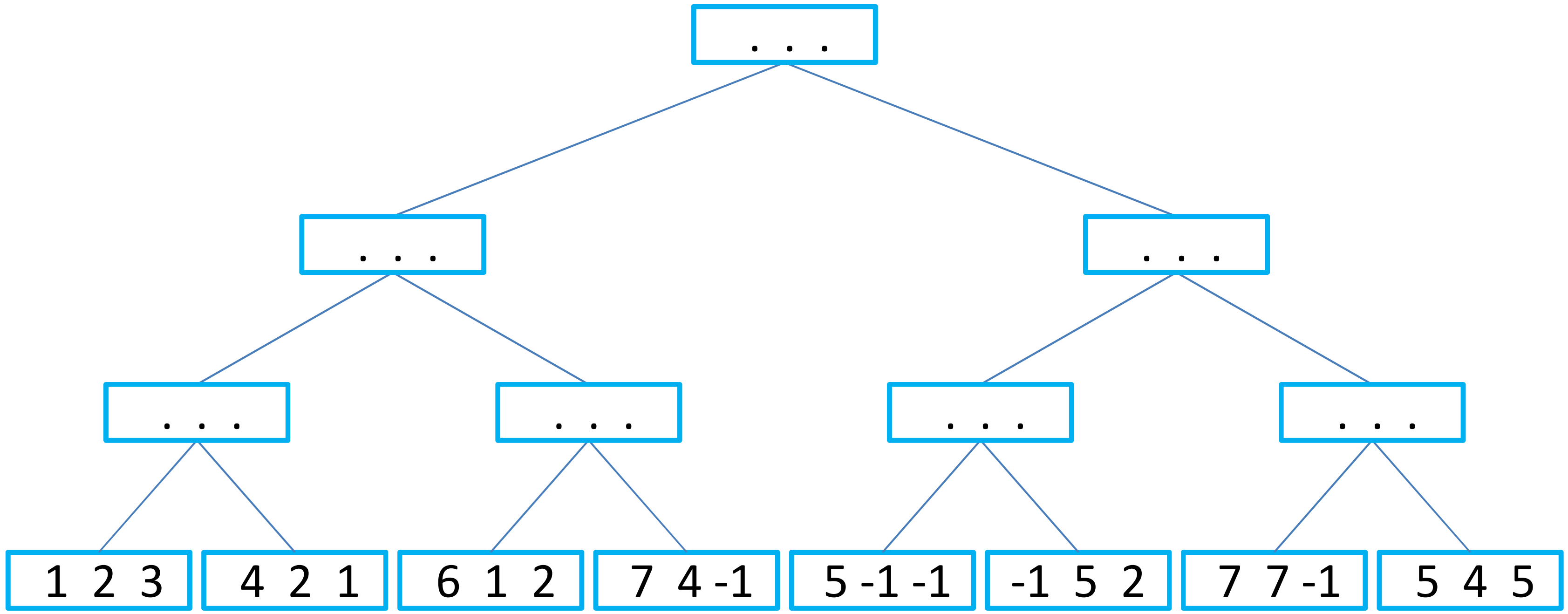


MiniMax & Constraint Processing: MiniMax Algorithm

MINIMAX FOR 3 PLAYERS

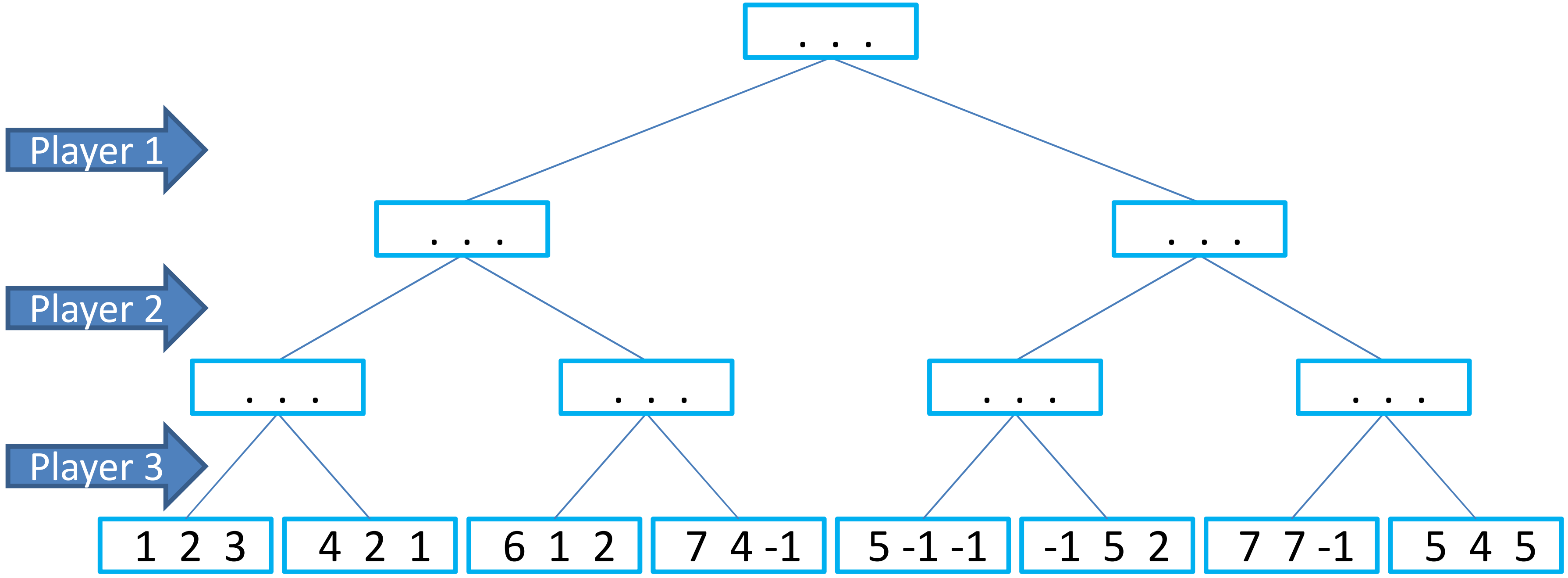
MiniMax For 3 Players

- All players are Max
- Evaluation function given by vector



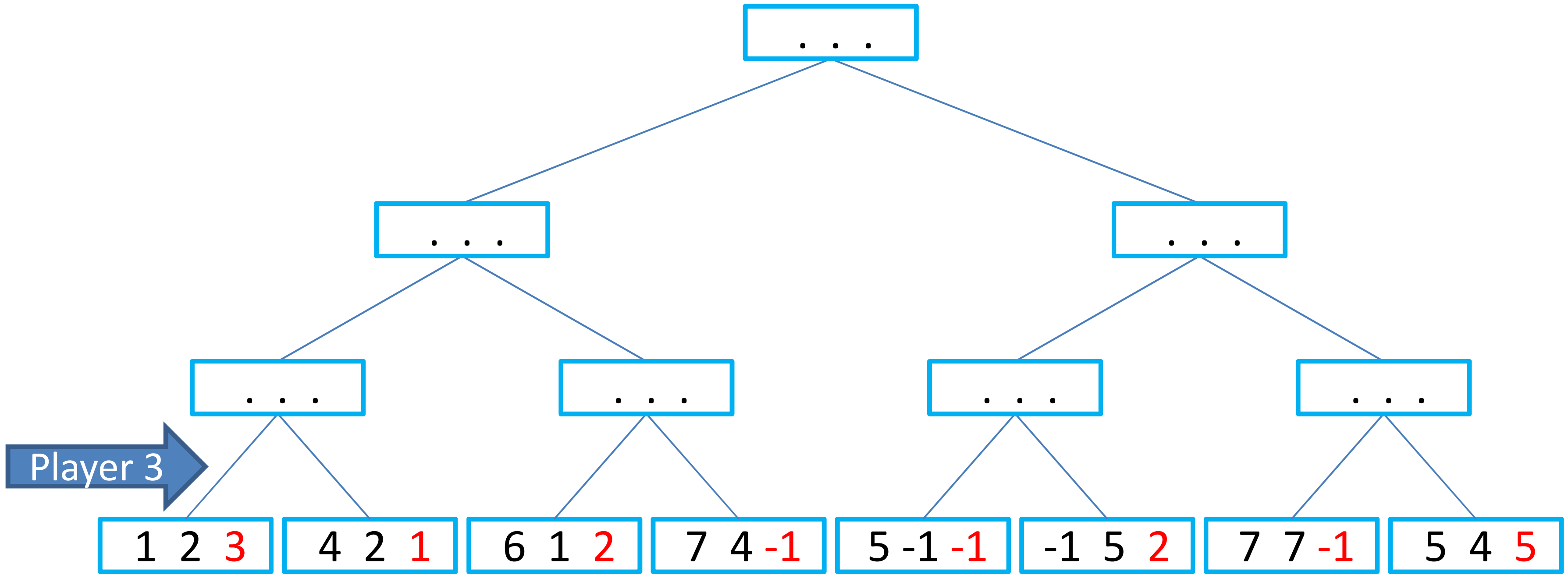
MiniMax For 3 Players

- Each layer assigned to 1 player
- Turn: every 3 layers



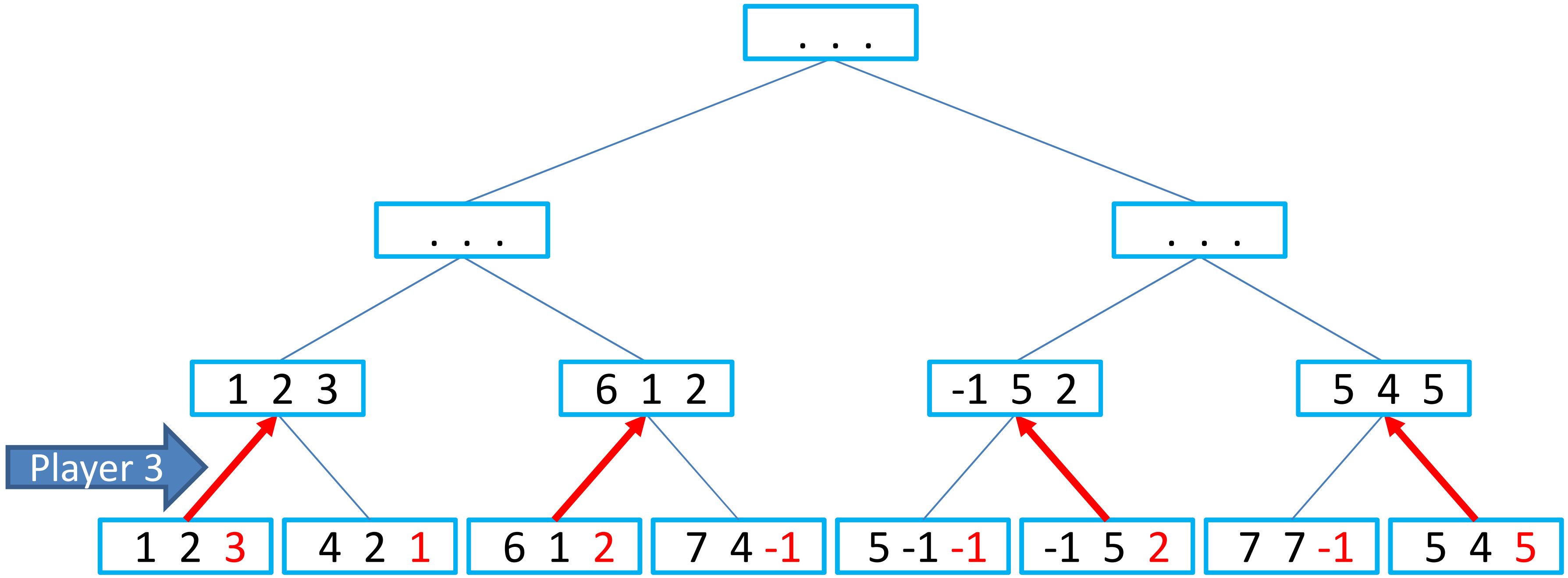
MiniMax For 3 Players

- Max third player: third position of vector



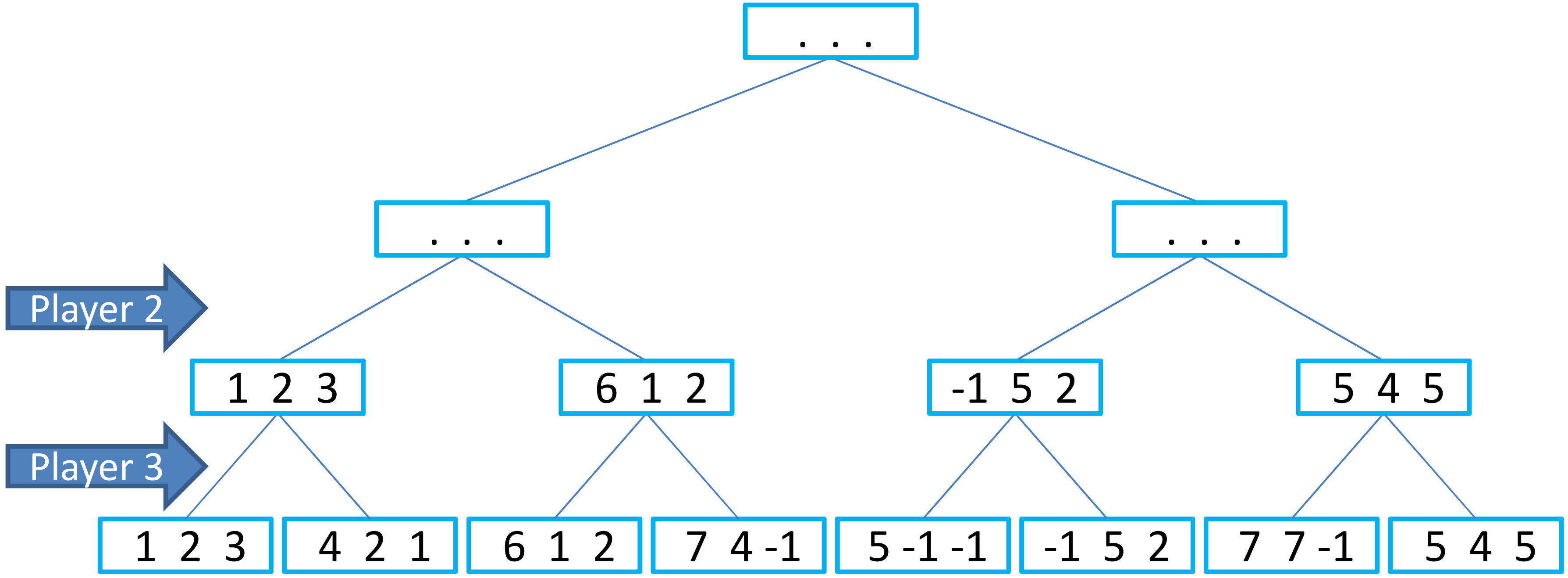
MiniMax For 3 Players

- MaxThirdPlayer([1,2,3],[4,2,1]) = [1,2,3]
- MaxThirdPlayer([6,1,2],[7,4,-1]) = [6,1,2]
- MaxThirdPlayer([5,-1,-1],[-1,5,2]) = [-1,5,2]
- MaxThirdPlayer([7,7,-1],[5,4,5]) = [5,4,5]



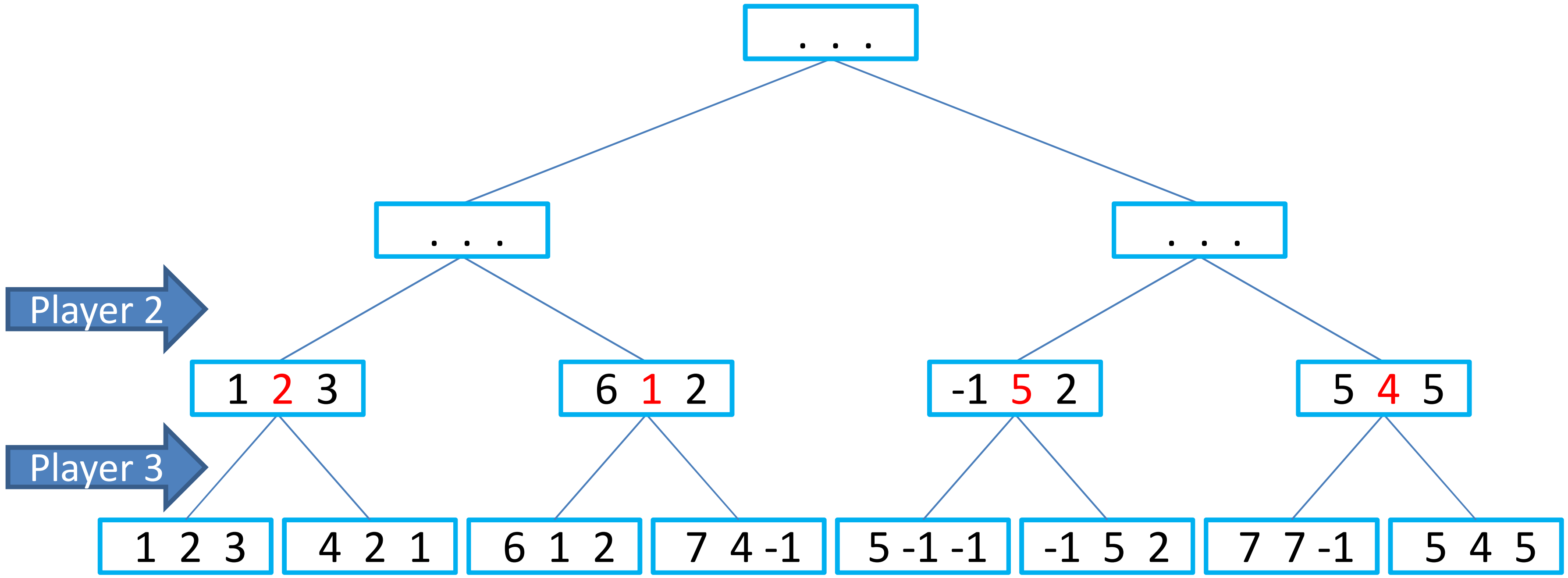
MiniMax For 3 Players

- Second player's move



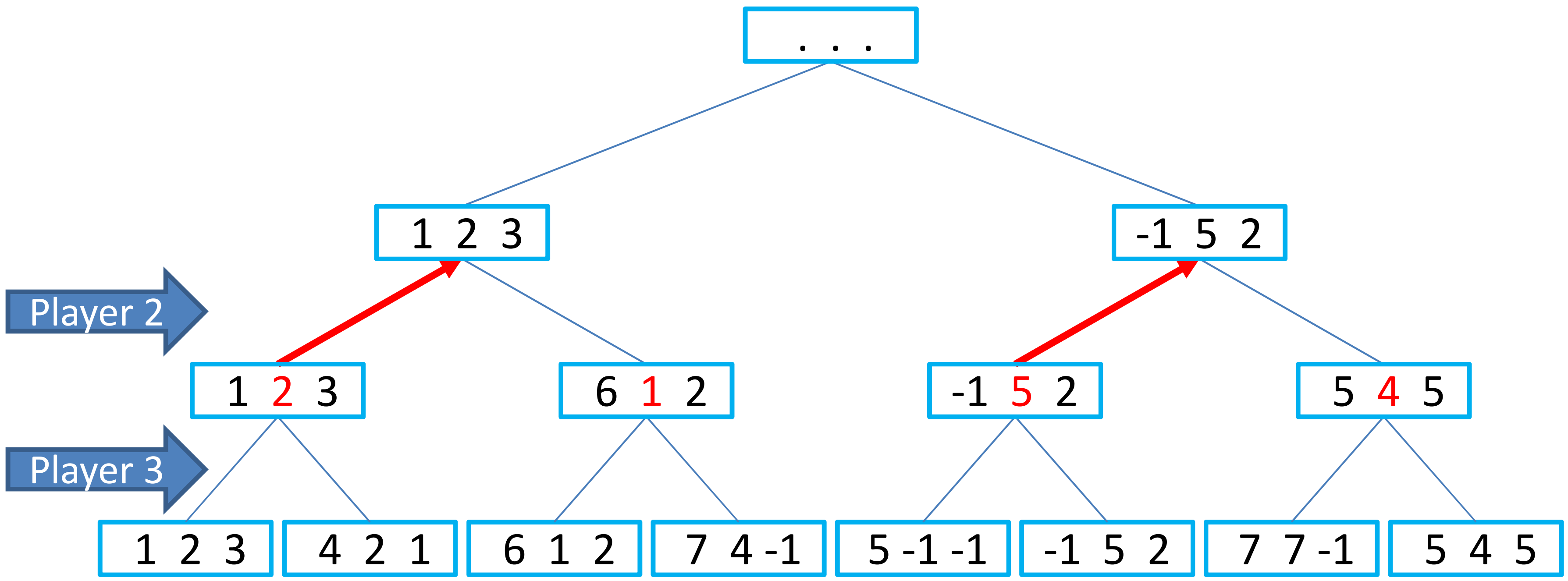
MiniMax For 3 Players

- Max second player: second position of vector



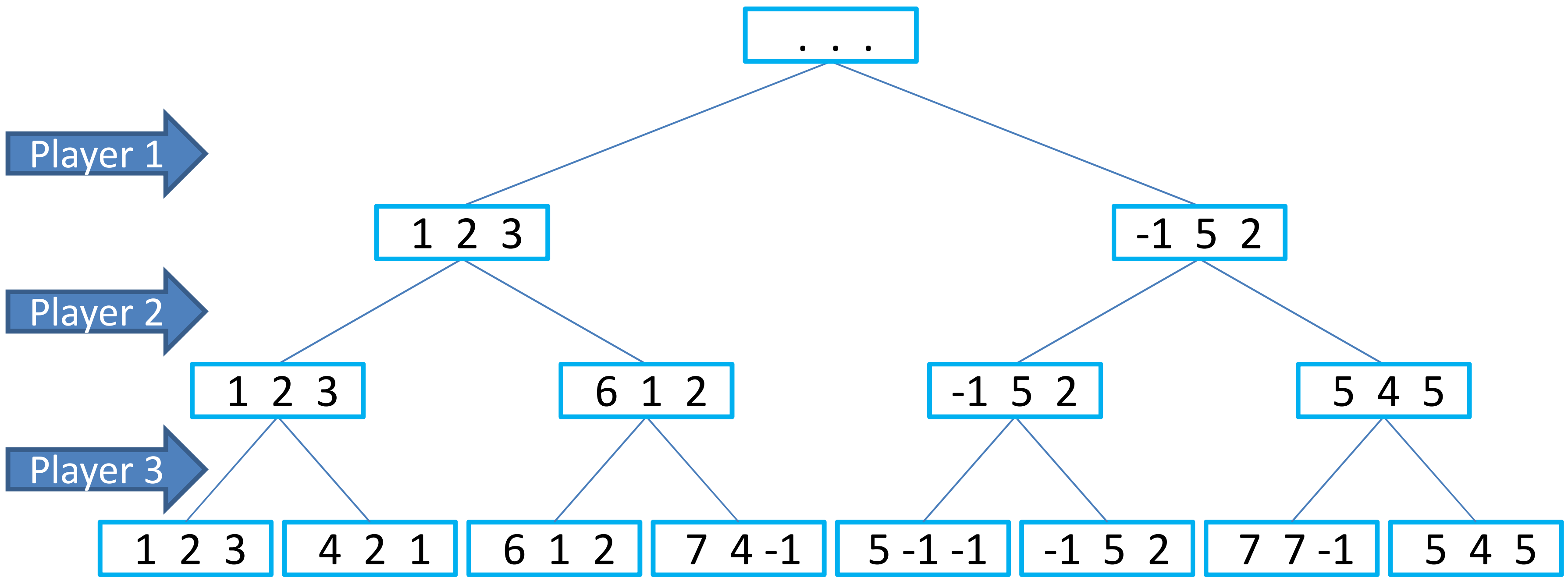
MiniMax For 3 Players

- MaxSecondPlayer([1,2,3],[6,1,2]) = [1,2,3]
- MaxSecondPlayer([-1,5,2],[5,4,5]) = [-1,5,2]



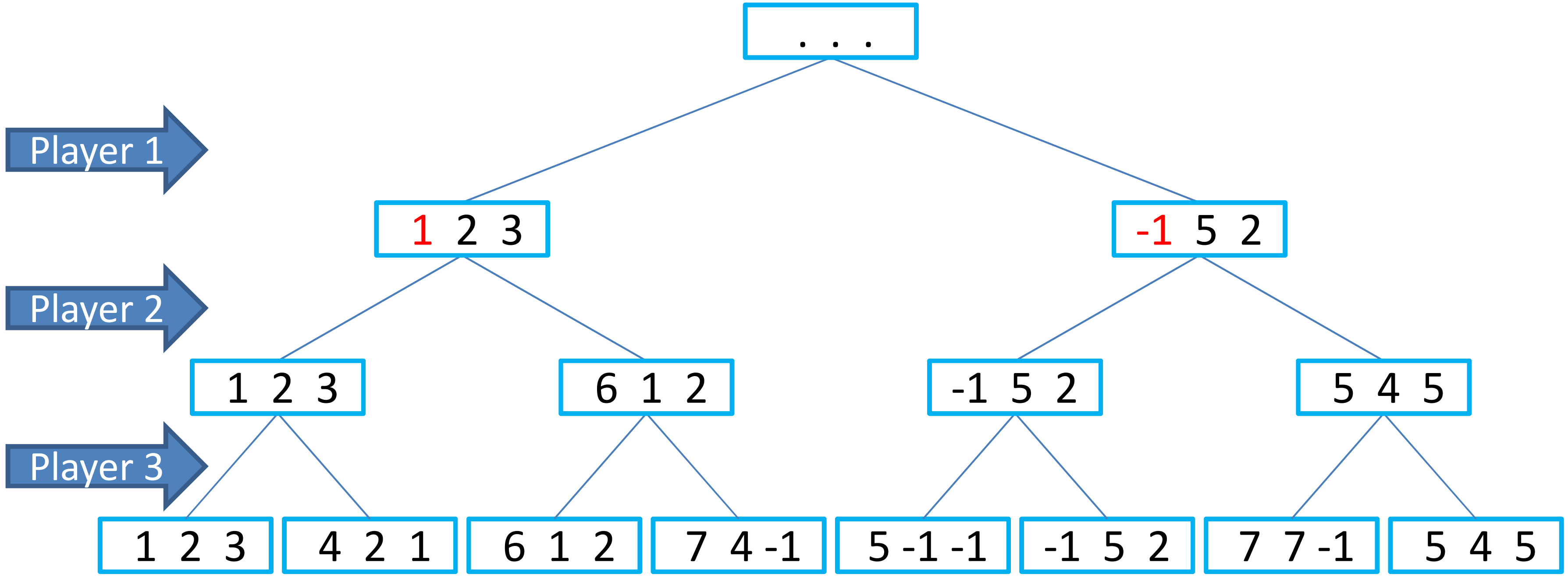
MiniMax For 3 Players

- First player’s move



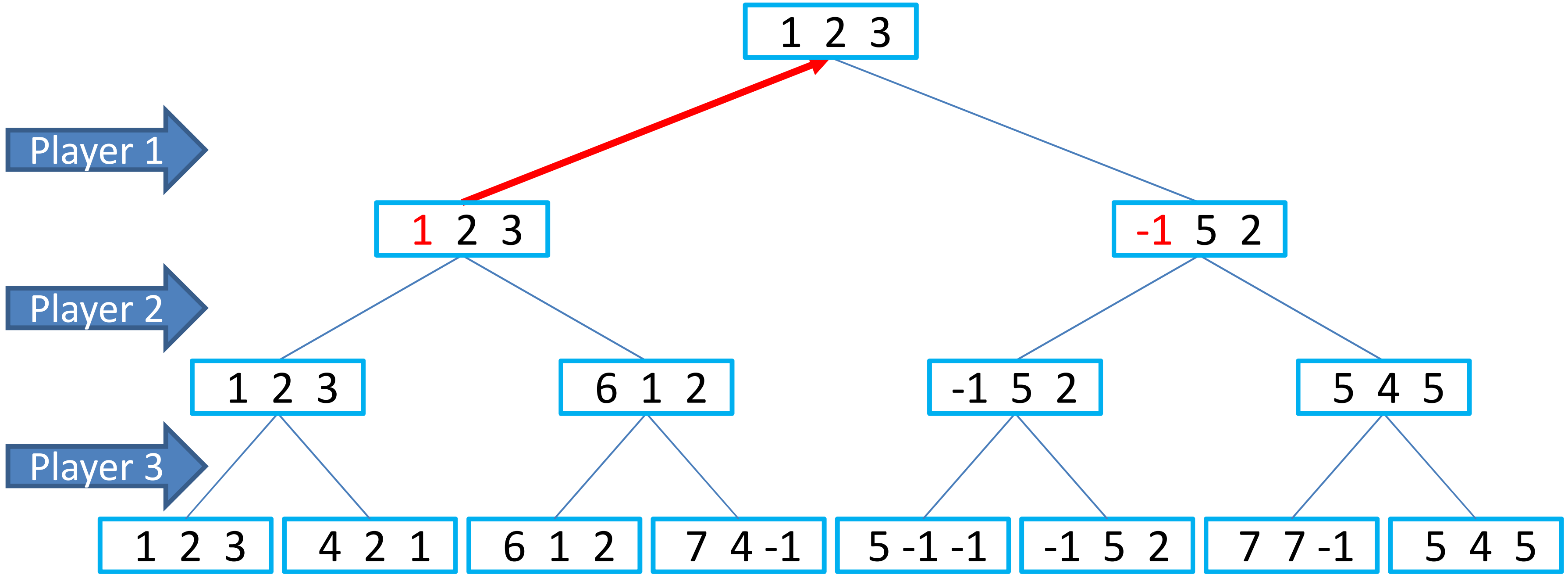
MiniMax For 3 Players

- Max first player: first position of vector



MiniMax For 3 Players

- MaxFirstPlayer([1,2,3],[-1,5,4]) = [1,2,3]


1 2 3
Player 1
1 2 3
-1 5 2
Player 2
1 2 3
6 1 2
-1 5 2
5 4 5
Player 3
1 2 3
4 2 1
6 1 2
7 4 -1
5 -1 -1
-1 5 2
7 7 -1
5 4 5