Institute of
Pervasive Computing

Monte Carlo Tree Search

Algorithms and Data Structures 2, 340300
Lecture – 2023W
Univ.-Prof. Dr. Alois Ferscha, teaching@pervasive.jku.at

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenberger Straße 69
4040 Linz, Austria
jku.at

Introduction

Search trees

- Search (no adversary), analyse traversing strategies, evaluation of cost

Game trees

- Games (adversary), game states (board configurations) have utility, tree defines decision process (**decision tree**), **find strategy**
  - Example: **2-Player Games:** Two players, **fully observable** environments, **deterministic**, **turn-taking, zero-sum games** of **perfect information** (e.g., go, chess, backgammonm, tic-tac-toe, etc.)

Consideration of a **Game** as a **Search Problem**:

- States = board configurations
- Operators = legal moves
- Initial State = current configuration
- Goal = find winning configuration
- payoff function (utility) = gives numerical value of outcome of the game

- **Two players, MIN** and **MAX taking turns**
- MIN/MAX use search tree to find next move

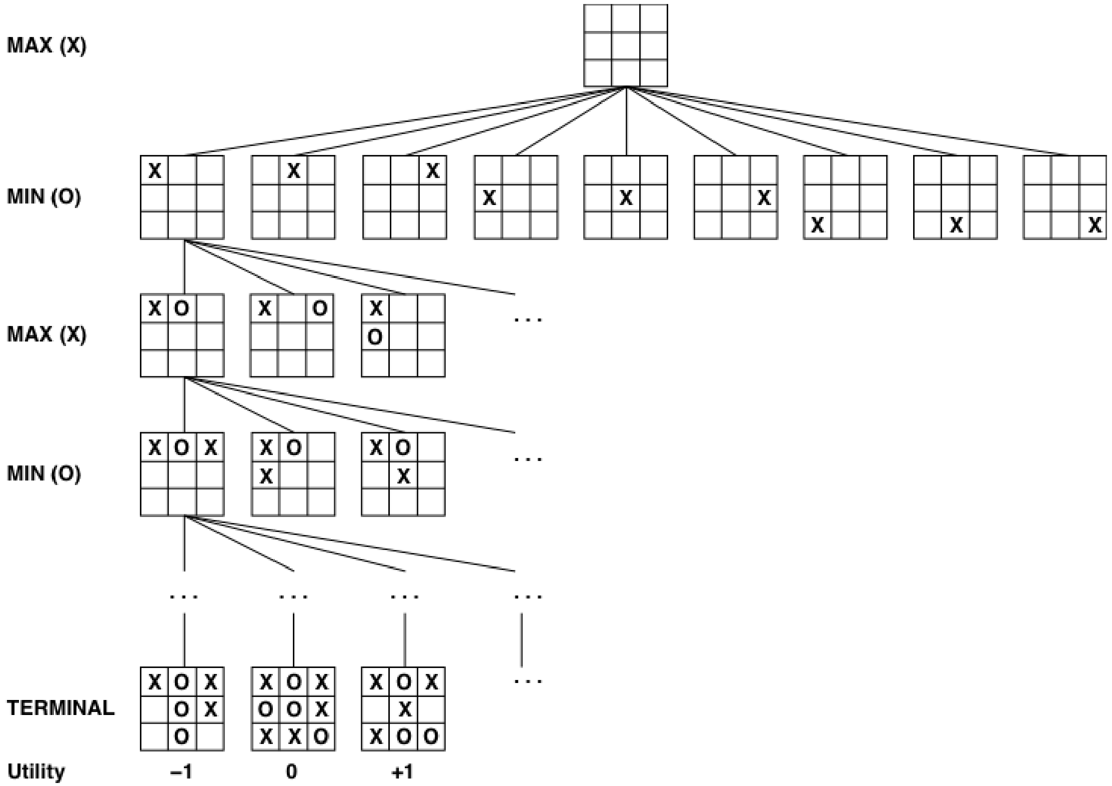
Game Trees and the Minimax Algorithm

How can MIN/MAX **determine which move** to pick to **win** the game?

We know for **each terminal state** the outcome of the game – this is called the **utility**.

In each turn, both players want to select a node which results in the best utility **for them.**

1. **Generate whole game tree** to leaves
2. Apply **utility function** to leaves
3. **Back up values** from leaves to root
   - MAX nodes compute maximum of children
   - MIN nodes compute minimum of children
4. When value **reaches root**: choose max value and the corresponding move



Deterministic, perfect information Tic-Tac-Toe game tree of 2 players (5,478 valid game states).

Games and Adversarial Search - Marco Chiarandini

Minimax Algorithm

Minimax value

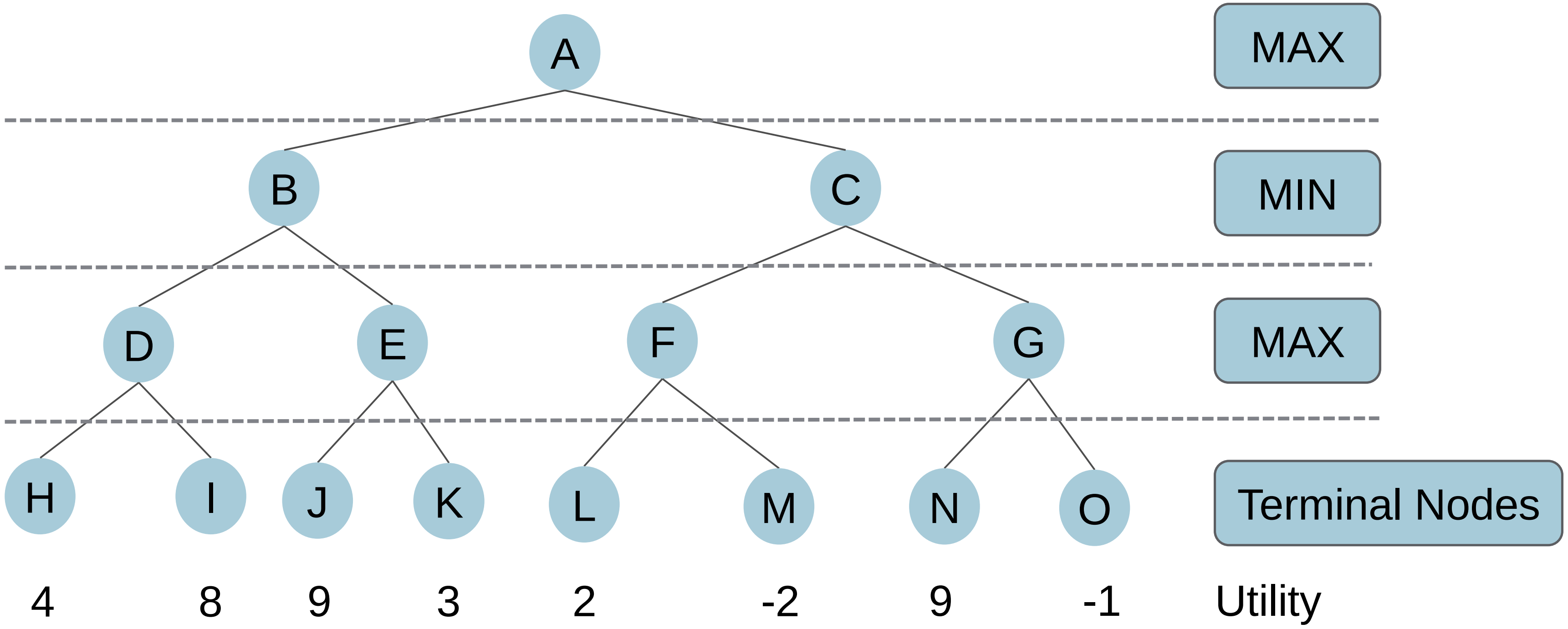
Is the **best utility** that can be **reached from** a current **node *n*** onwards, assuming that **both players play optimally** from *n* to the end of the game:

$$\text{MINIMAX-VALUE (n)}=\begin{cases} Utility(n) & if\ n\ is\ a\ terminal\ node \\ \min_{s \in Successor(n)} MINIMAX-VALUE(s) & if\ n\ is\ a\ MIN\ node \\ max_{s \in Successor(n)} MINIMAX-VALUE(s) & if\ n\ is\ a\ MAX\ node \end{cases}$$

MAX will try to move to states with maximum values.

MIN will try to move to states with minimum values.

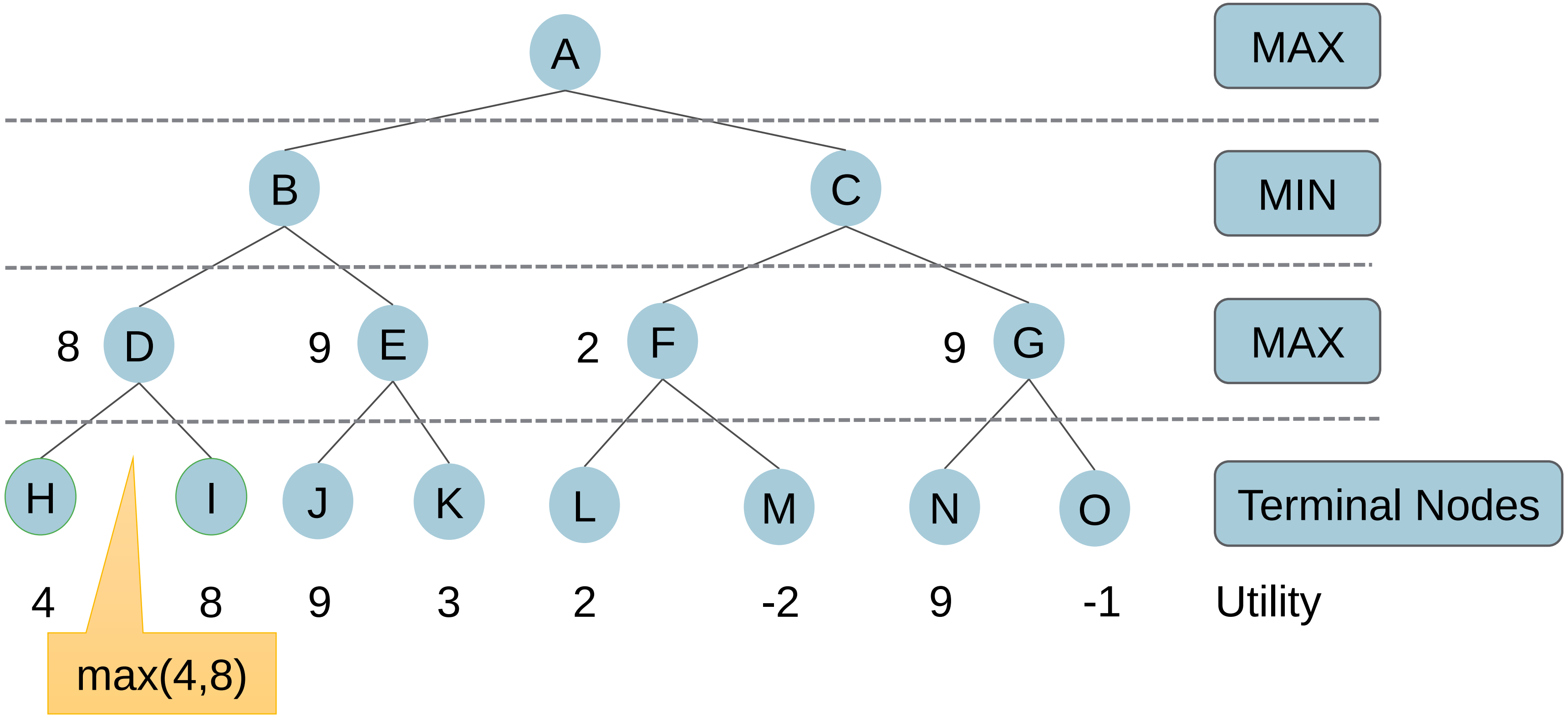
Minimax Algorithm :: Example



Step 1:

- The entire decision tree is generated (meaning we **expand every possible move**).
- The **utility function** is applied to get the terminal values for each node.

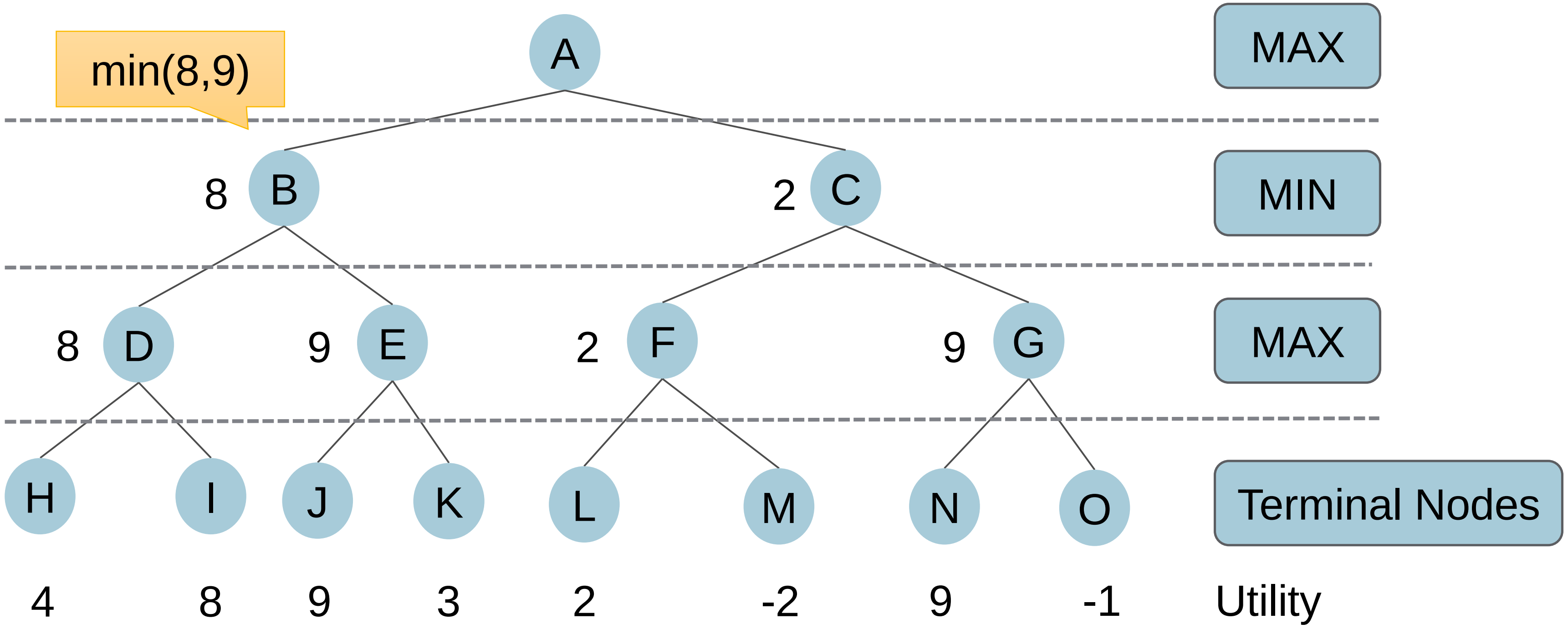
Minimax Algorithm :: Example



Steps 2-5:

- The first **minimax values** for **MAX** are determined.
- Node D: max(4, 8) = 8
- Node E: max(9, 3) = 9
- Node F: max(2, -2) = 2
- Node G: max(9, -1) = 9

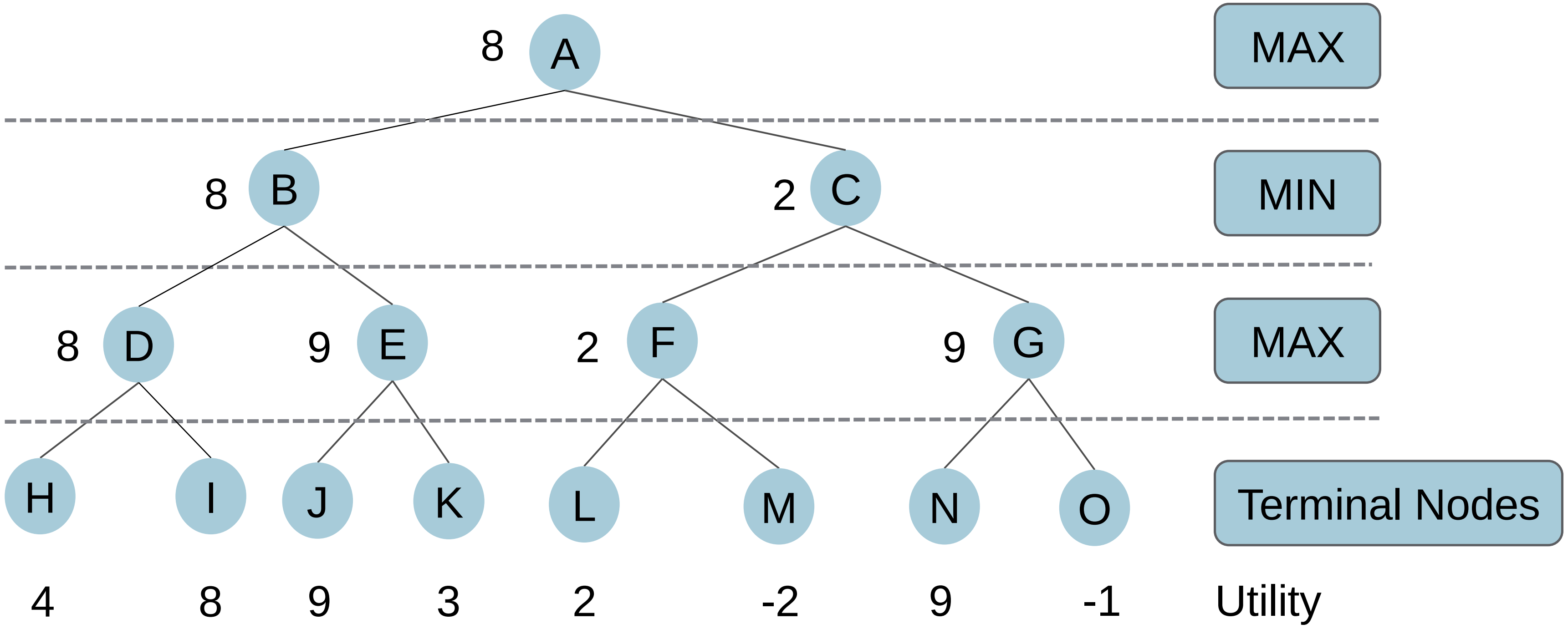
Minimax Algorithm :: Example



Steps 6-7:

- The minimax values for **MIN** are determined.
- Node B: min(8, 9) = 8
- Node C: min(2, 9) = 2

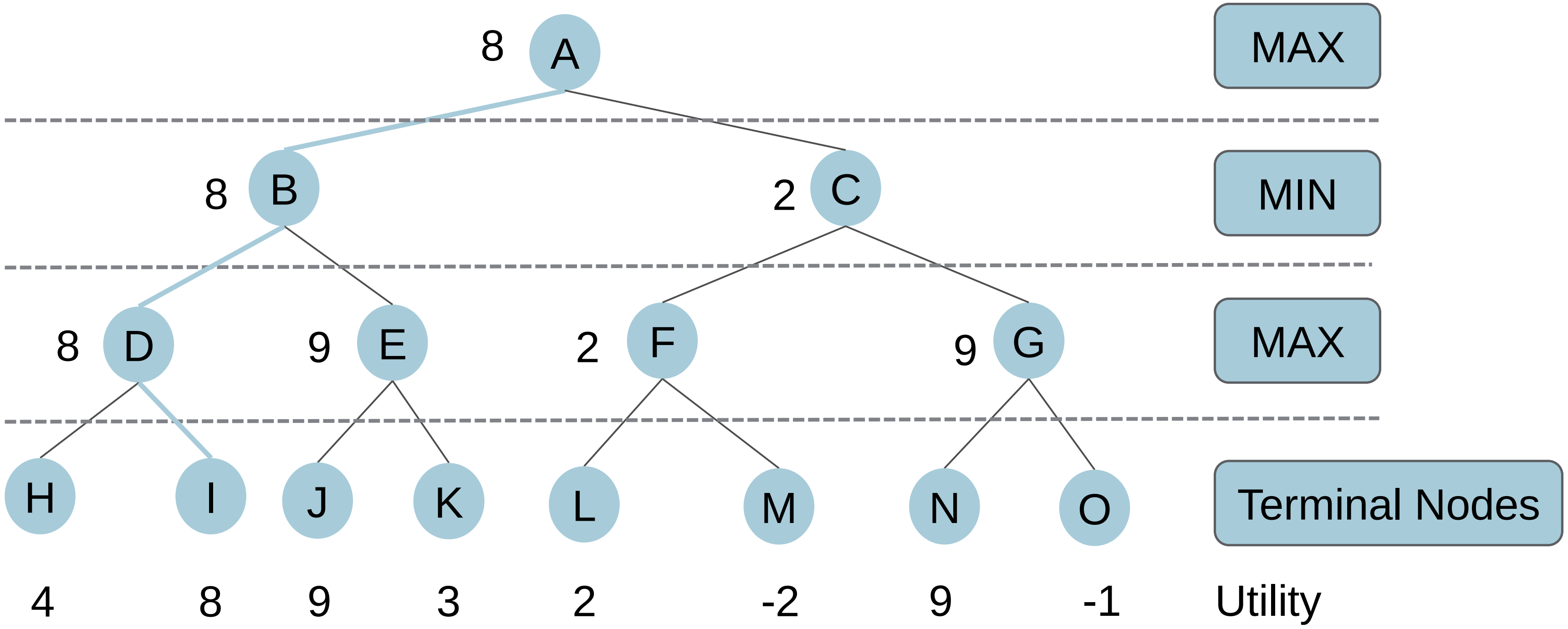
Minimax Algorithm :: Example



Step 8:

- The minimax value for **MAX** in the **root node** is determined.
- Node A: max(8, 2) = 8

Minimax Algorithm :: Example



Result

- With this we **found** our **optimal playing strategy**.
- MAX moves to node B.
- MIN Moves to node D.
- MAX moves to node I.

Minimax Algorithm

Properties of Minimax

Completeness:

- Minimax is **complete**, if the game tree is **finite**.

Optimality:

- Optimal if **opponent** also plays optimally.

Time Complexity:

- O(**b^m**)

Space Complexity:

- O(**bm**)

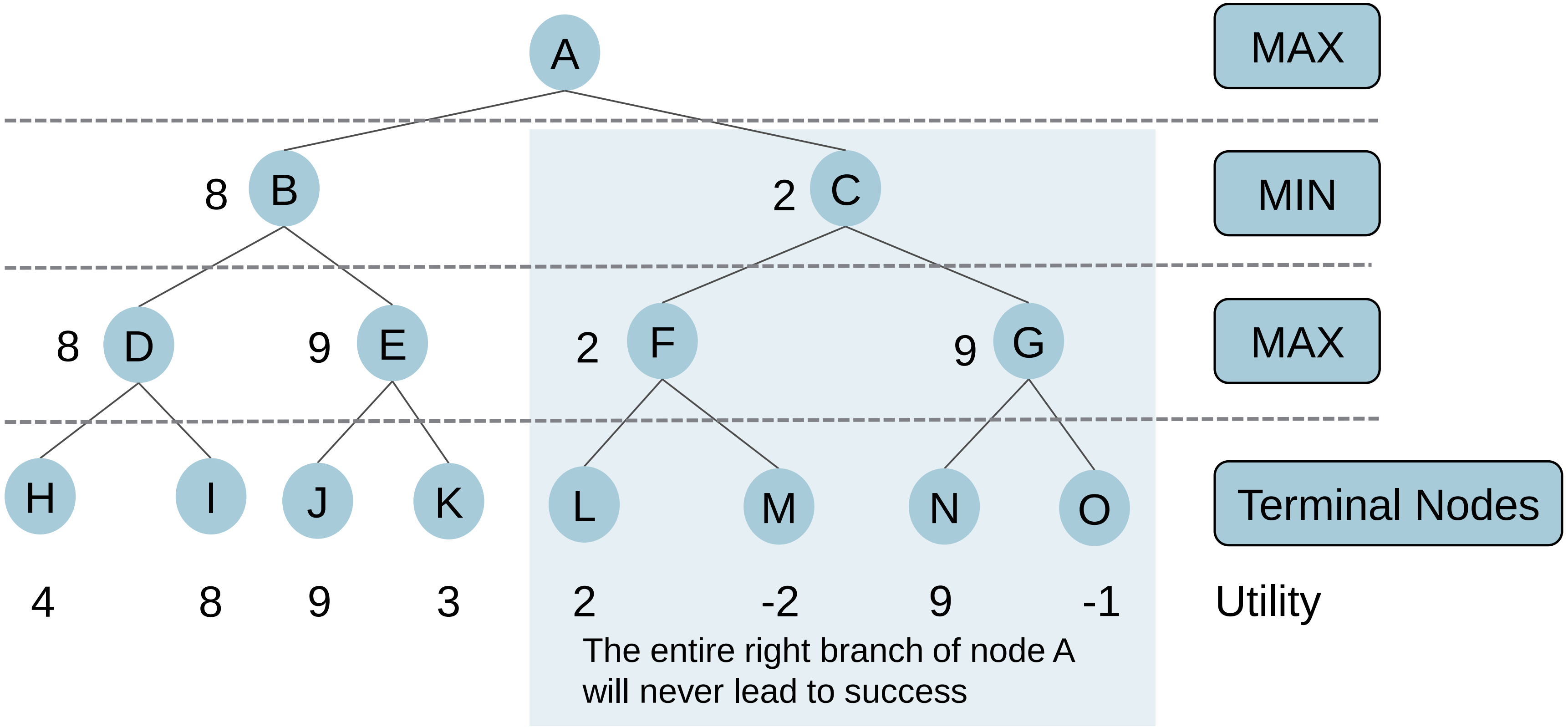
b ... **branching factor** (max. number of successors of any node).
m ... **maximum length** of **any path** in the state space (may be infinite).

Games and Adversarial Search - Marco Chiarandini

Alpha-Beta Pruning

Main disadvantage of Minimax

- Minimax has to **look into every node** of the game tree.



Alpha-Beta Pruning

Method

Propagate two parameters along the expansion of a path, and update them when backing up: [α, β].

- **α** ... best (**largest**) value found so far for **MAX**.
- **β** ... best (**smallest**) value found so far for **MIN**.

Pruning

- Whenever a Minimax **value** as a **child of a MIN node** is **less** than or equal to the current **α**:

  ➔ **ignore** remaining nodes (subtrees) **below** this **MIN** node.
- Whenever a Minimax **value** as a **child of a MAX node** is **greater** than or equal to the current **β**:

  ➔ **ignore** remaining nodes (subtrees) **below** this **MAX** node.

Games and Adversarial Search - Marco Chiarandini

Alpha-Beta Pruning

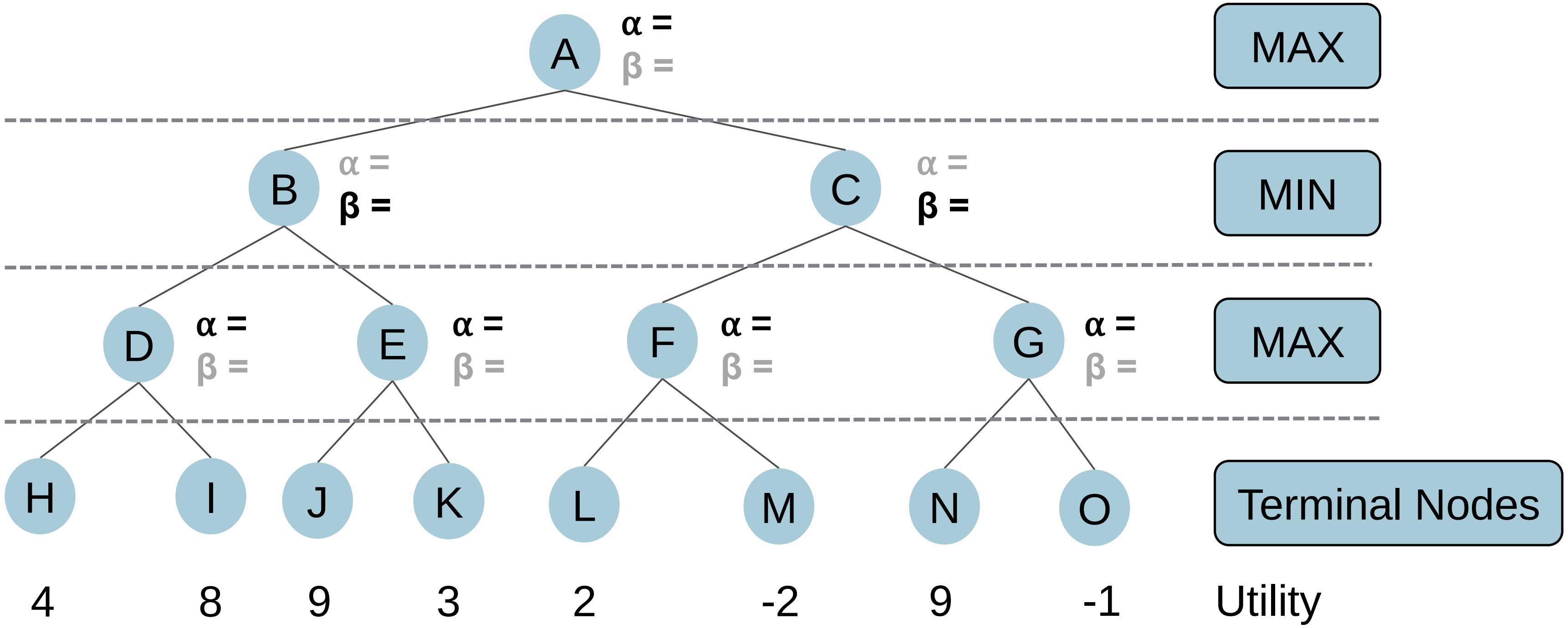
Basic algorithm outline

```
Max-Value(s, α, β):
  if terminal(s): return U(s)
  v = -∞
  for c in next-states(s):
    v’ = min-value(c, α, β)
    if v’ > v: v = v’
    if v’ ≥ β: return v
    if v’ > α: α = v’
  return v
```

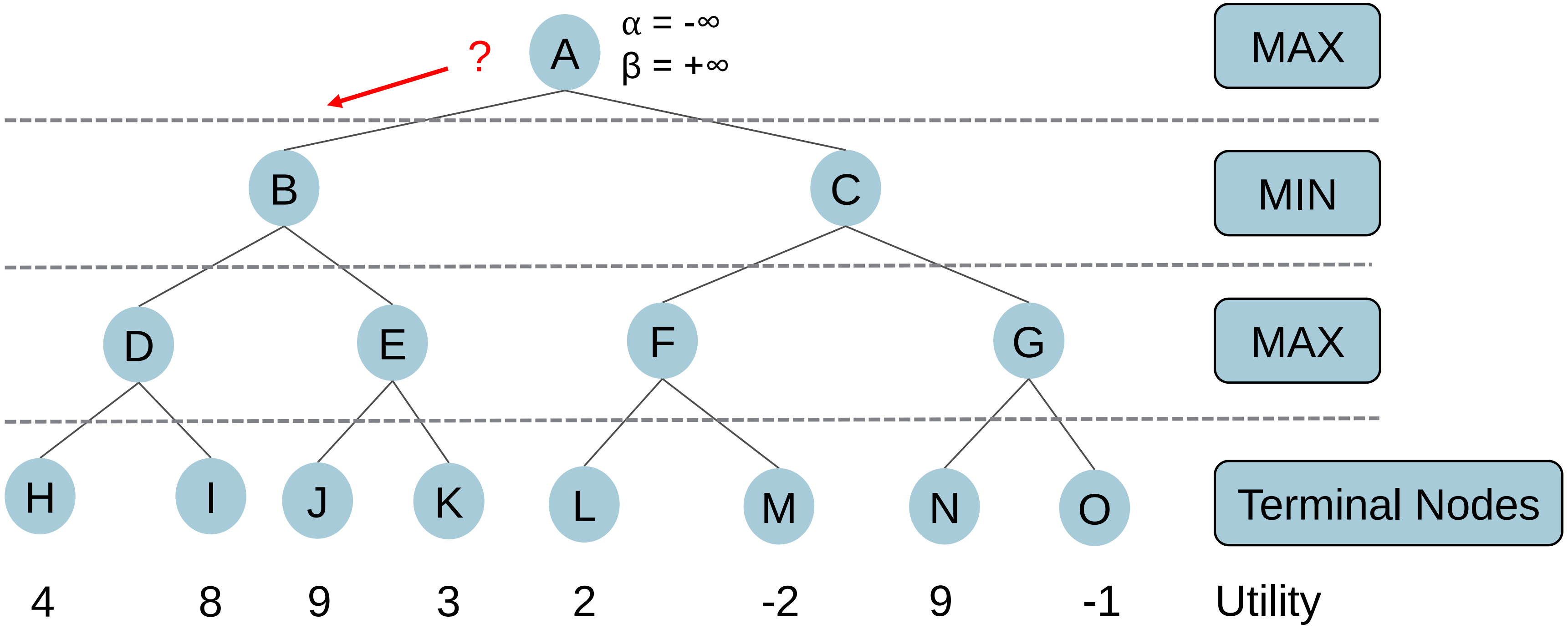
```
Min-Value(s, α, β):
  if terminal(s): return U(s)
  v = +∞
  for c in next-states(s):
    v’ = max-value(c, α, β)
    if v’ < v: v = v’
    if v’ ≤ α: return v
    if v’ < β: β = v’
  return v
```

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Principle



Alpha-Beta Pruning :: Example

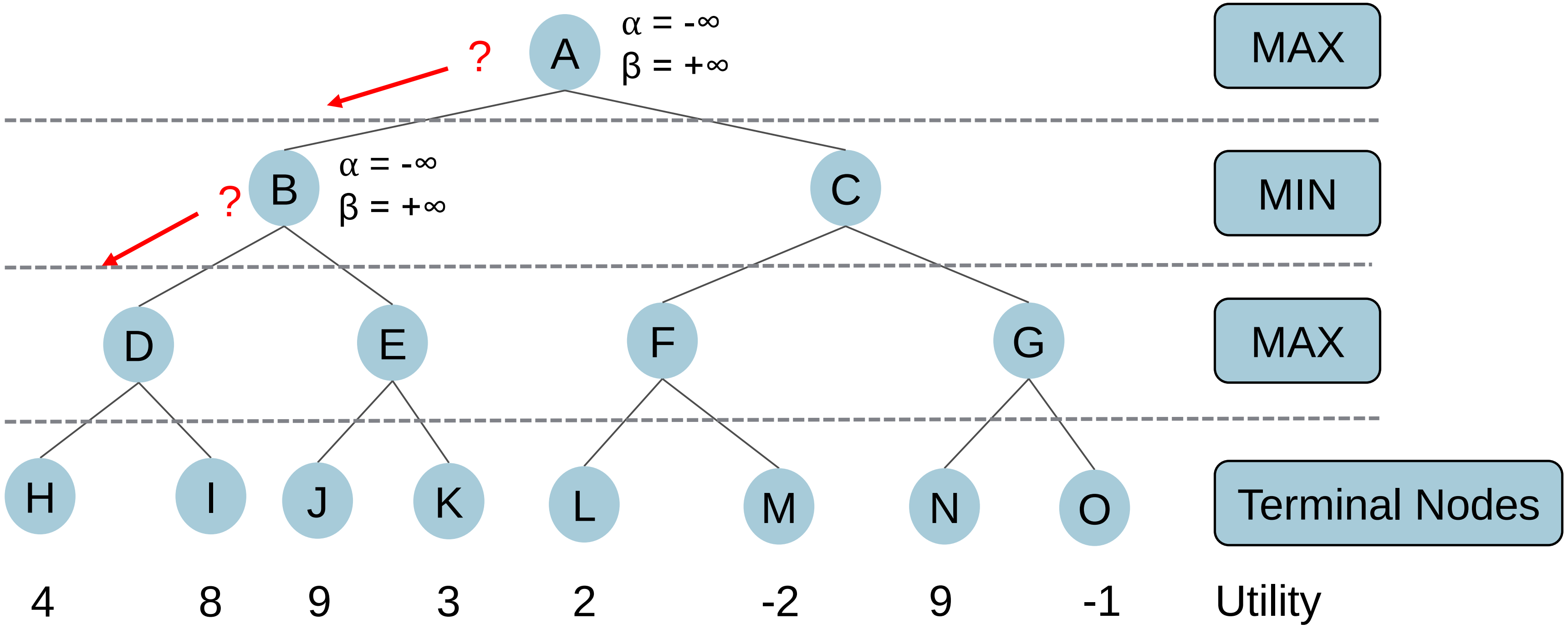


Step 1:

- The **entire decision tree** is generated.
- The **utility function** is applied to get the terminal values for each node.
- In node A **α is set to -∞** and **β is set to +∞** and propagated to node D.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

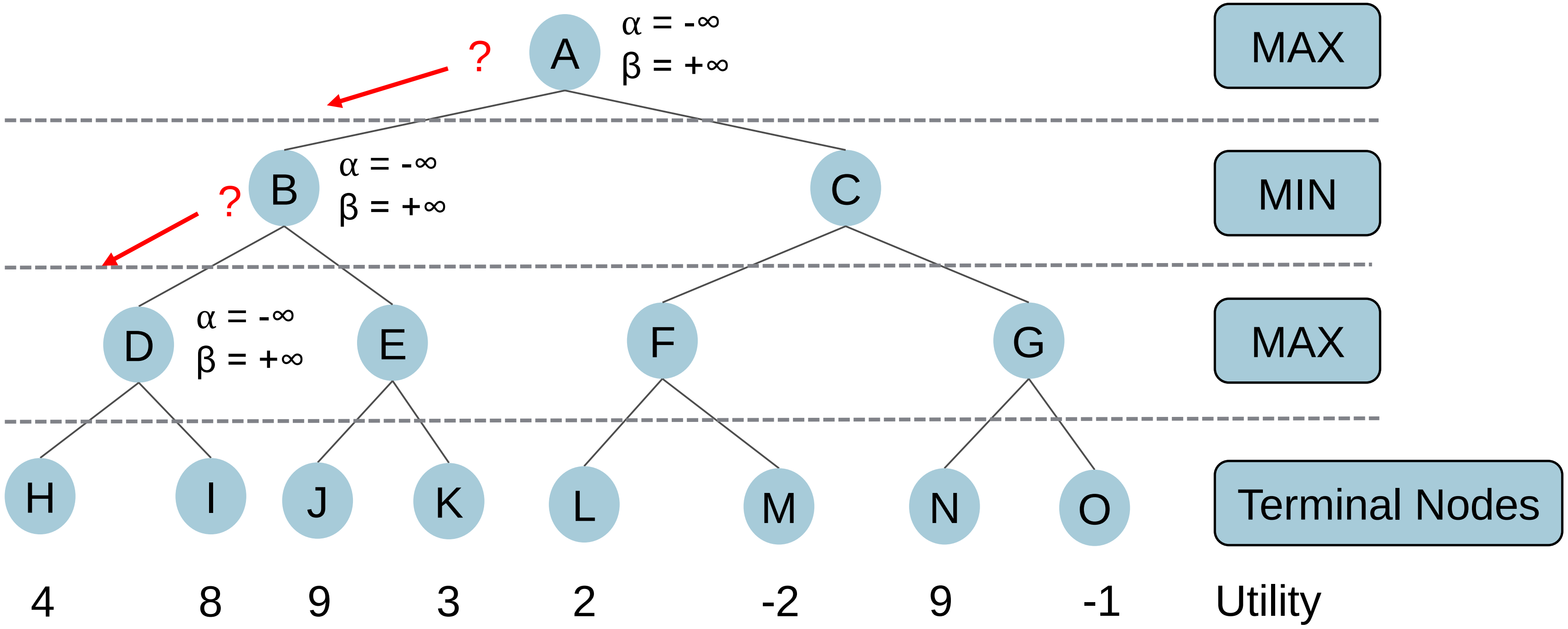


Step 1:

- The **entire decision tree** is generated.
- The **utility function** is applied to get the terminal values for each node.
- In node A **α is set to -∞** and **β is set to +∞** and propagated to node D.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

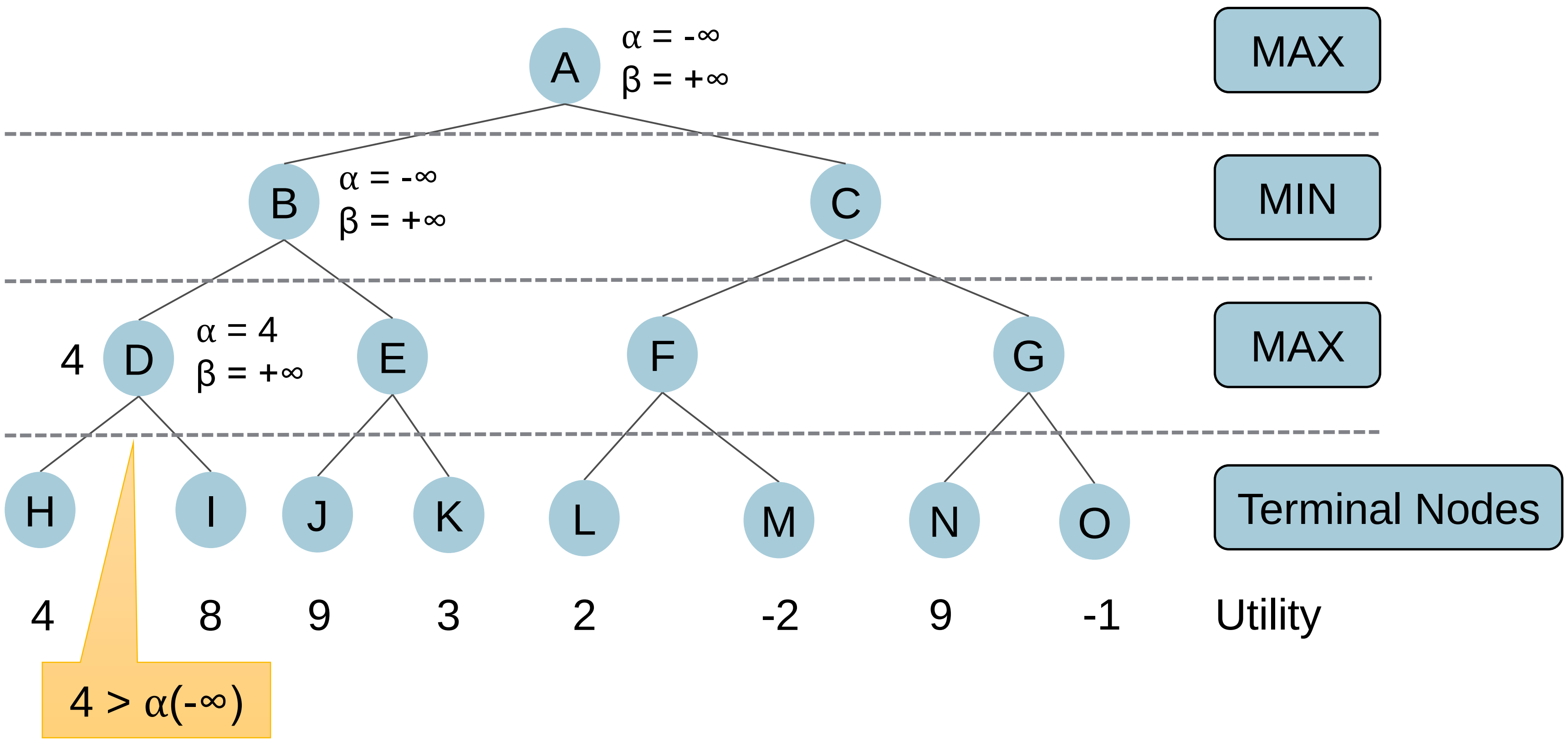


Step 1:

- The **entire decision tree** is generated.
- The **utility function** is applied to get the terminal values for each node.
- In node A **α is set to -∞** and **β is set to +∞** and propagated to node D.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

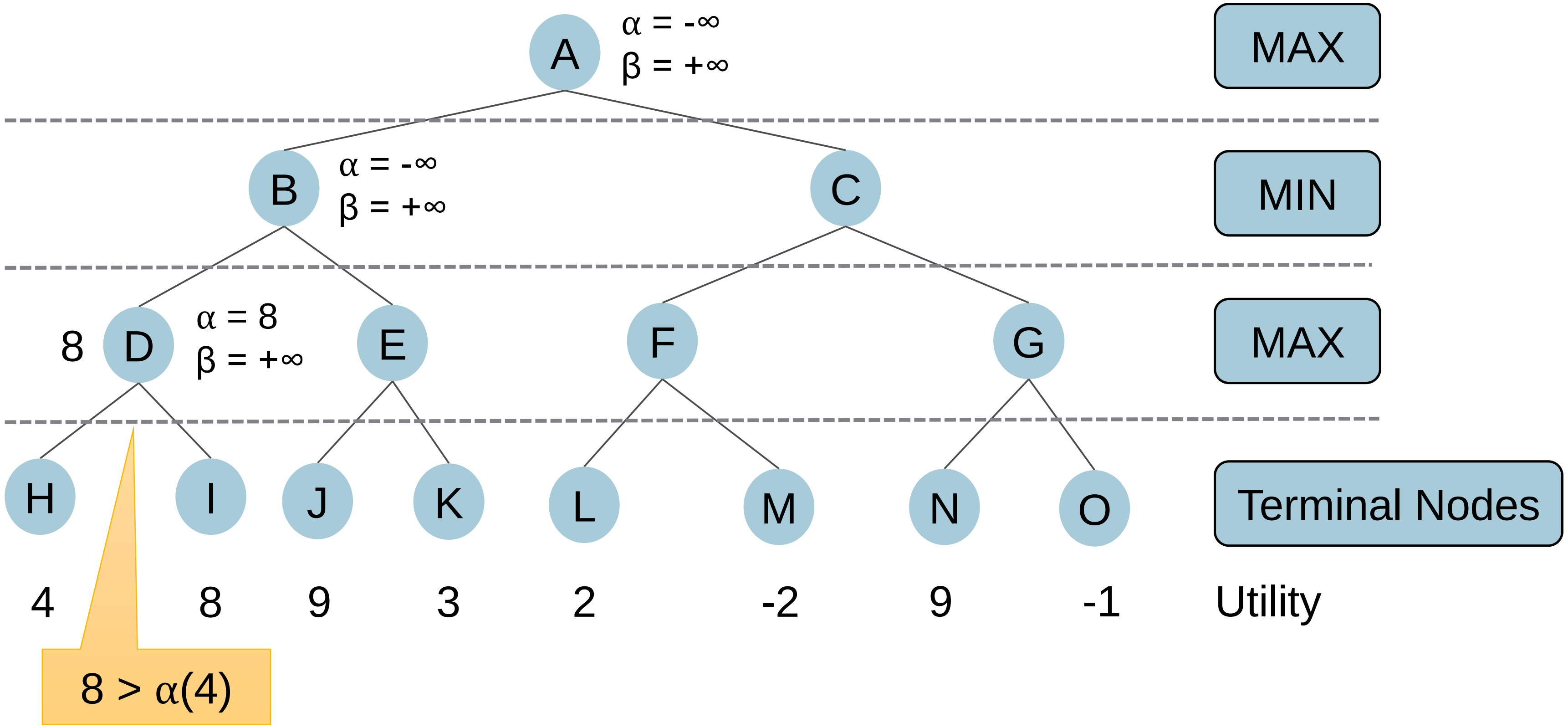


Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Step 2:

- In **node D**, **MAX** finds the value **4 of node H**.
- 4 > α(-∞): **α is updated** to 4 and the value of D is updated to 4.

Alpha-Beta Pruning :: Example

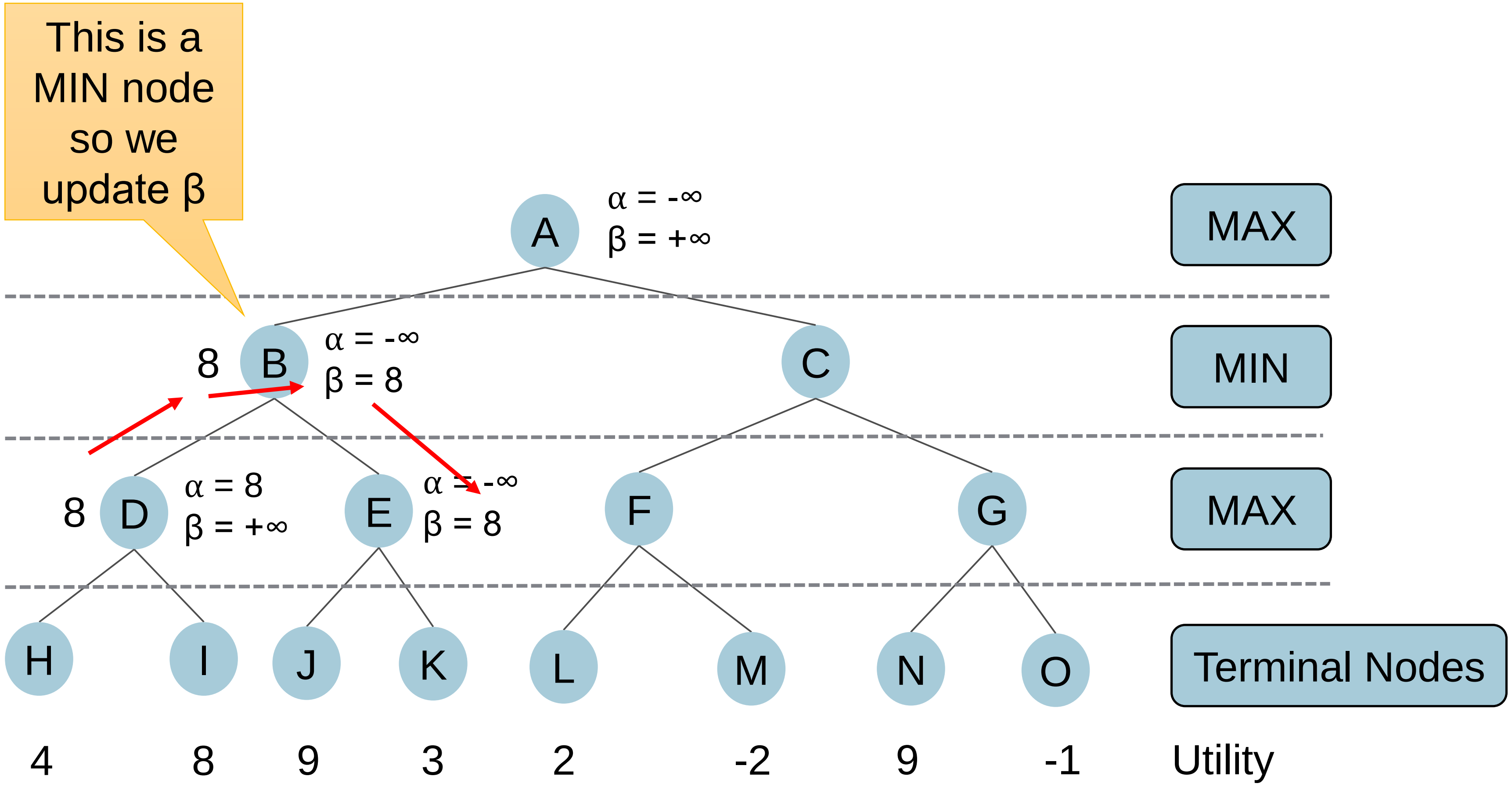


Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Step 3:

- In **node D** MAX finds the value **8 of node I**.
- 8 > α(4): α is updated to 8 and the value of D is updated to 8.

Alpha-Beta Pruning :: Example

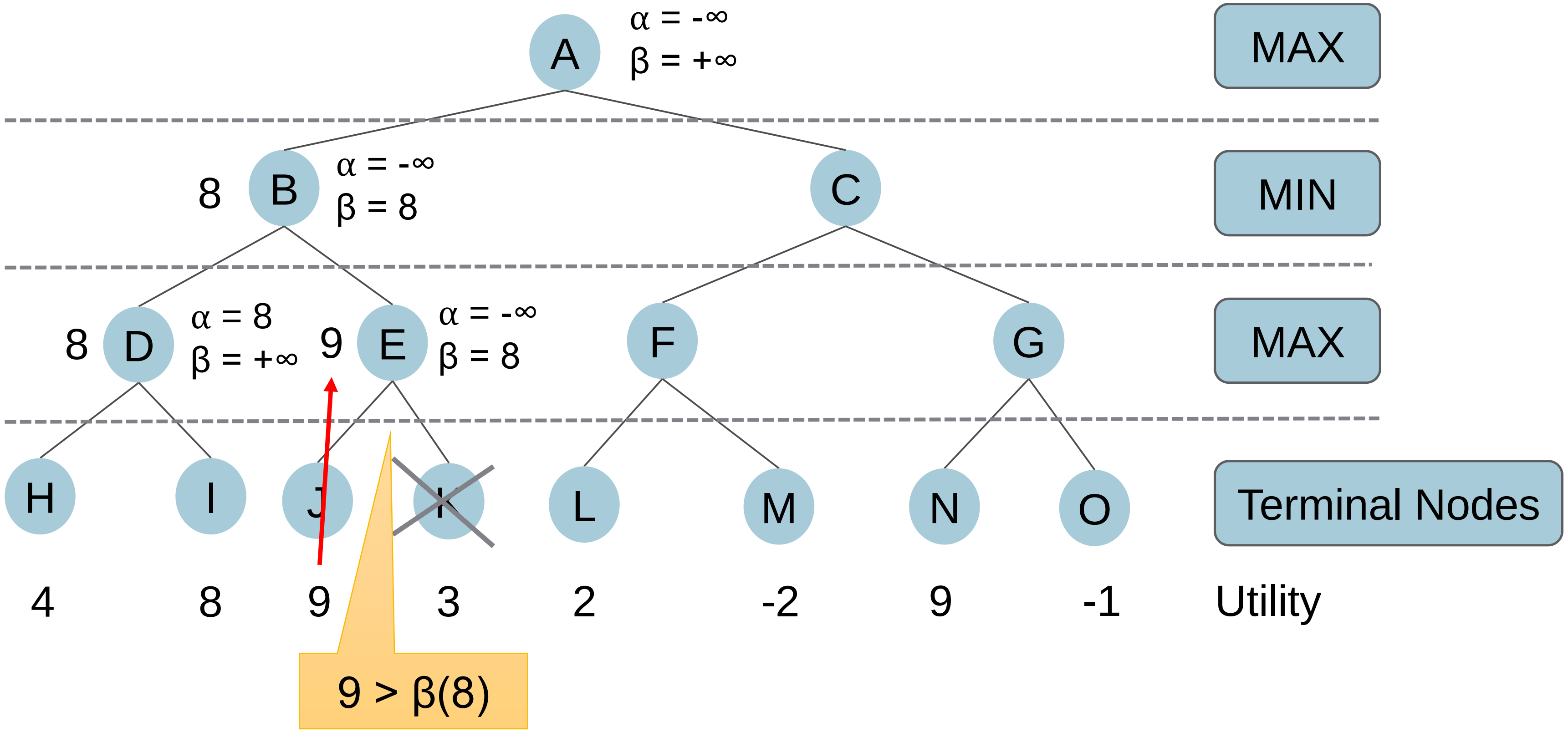


Step 4:

- In **node B** MIN finds the value **8 of node D**.
- 8 < β(+∞): **β is updated** to 8 and the value of B is updated to 8.
- **β is passed down** to node E.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

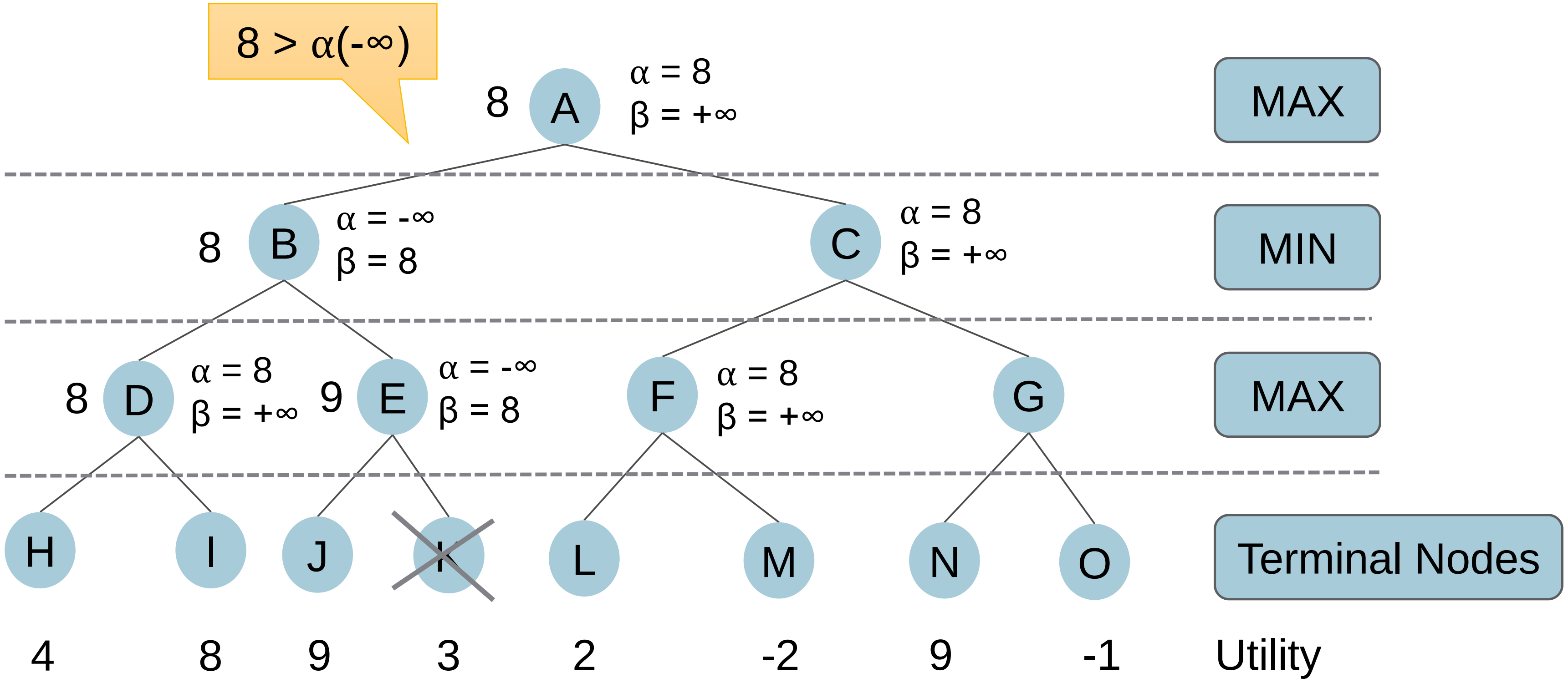


Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Step 5:

- In **node E** MAX finds the value **9 of node J**.
- 9 > β (8): the **remaining branches of E are pruned.**

Alpha-Beta Pruning :: Example

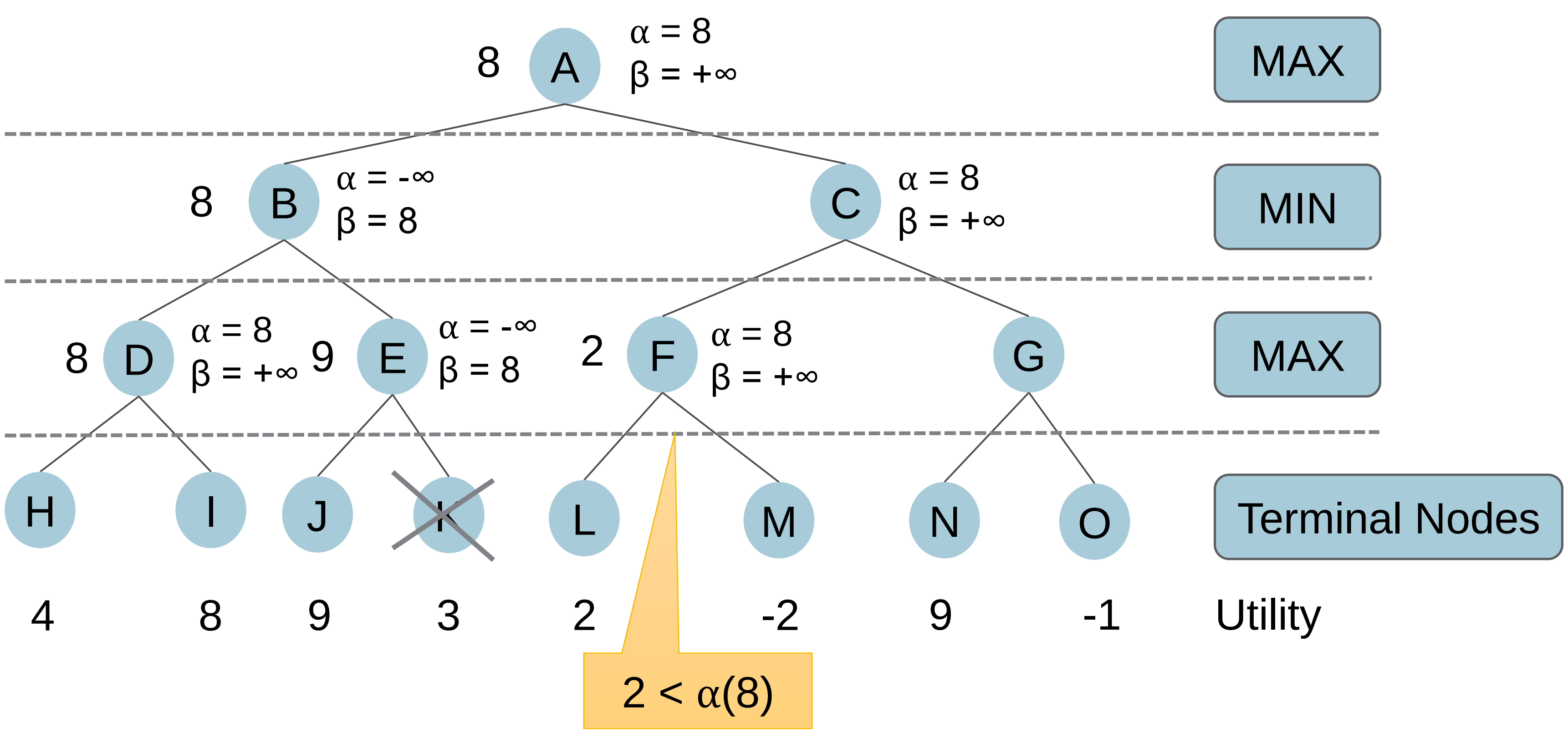


Step 6:

- In **node A** MAX finds the value **8 of node B**.
- 8 > α(-∞): **α is updated** to 8 and the value of node A is updated to 8.
- **α is down propagated** to node F.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

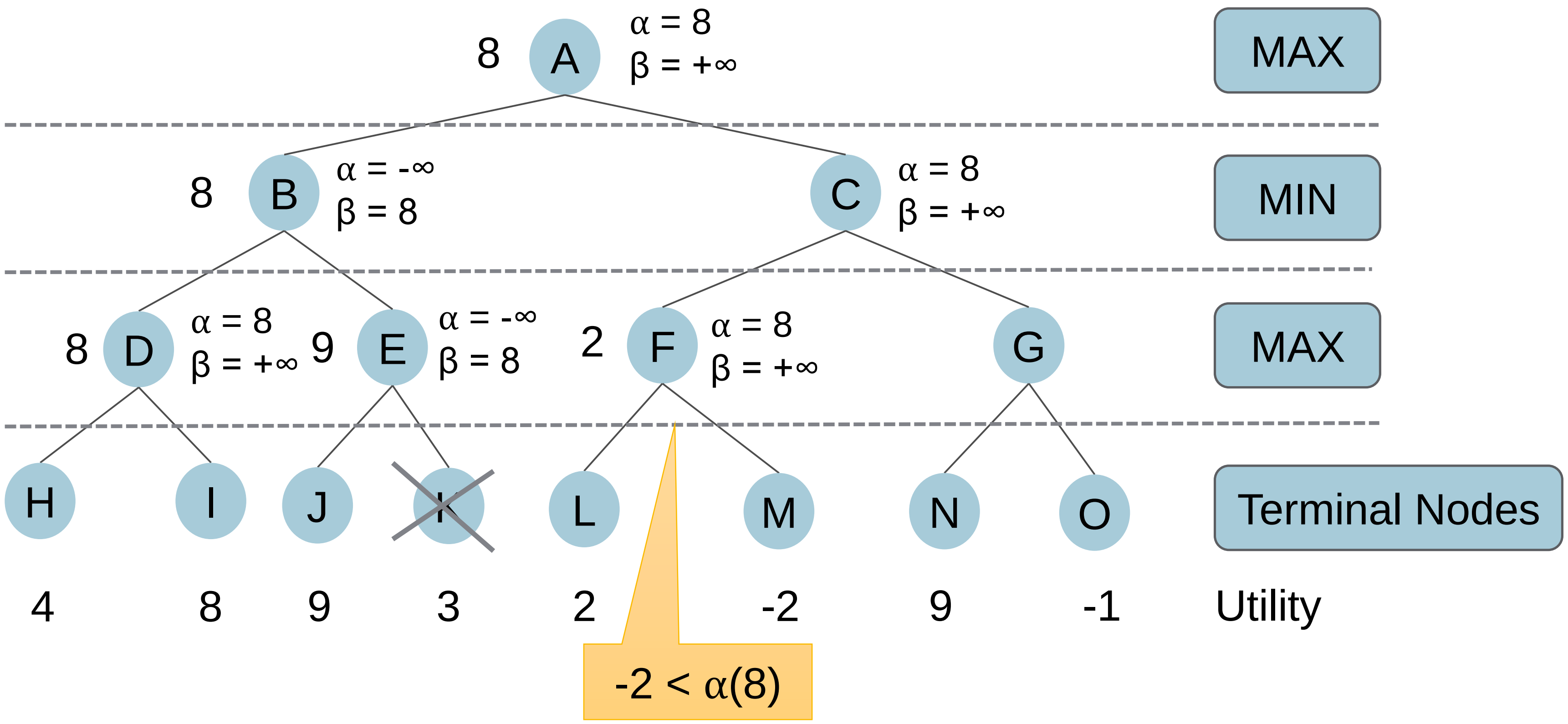


Step 7:

- In **node F** MAX finds the value **2 of node L**.
- 2 < α(8): **α is not updated.**

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

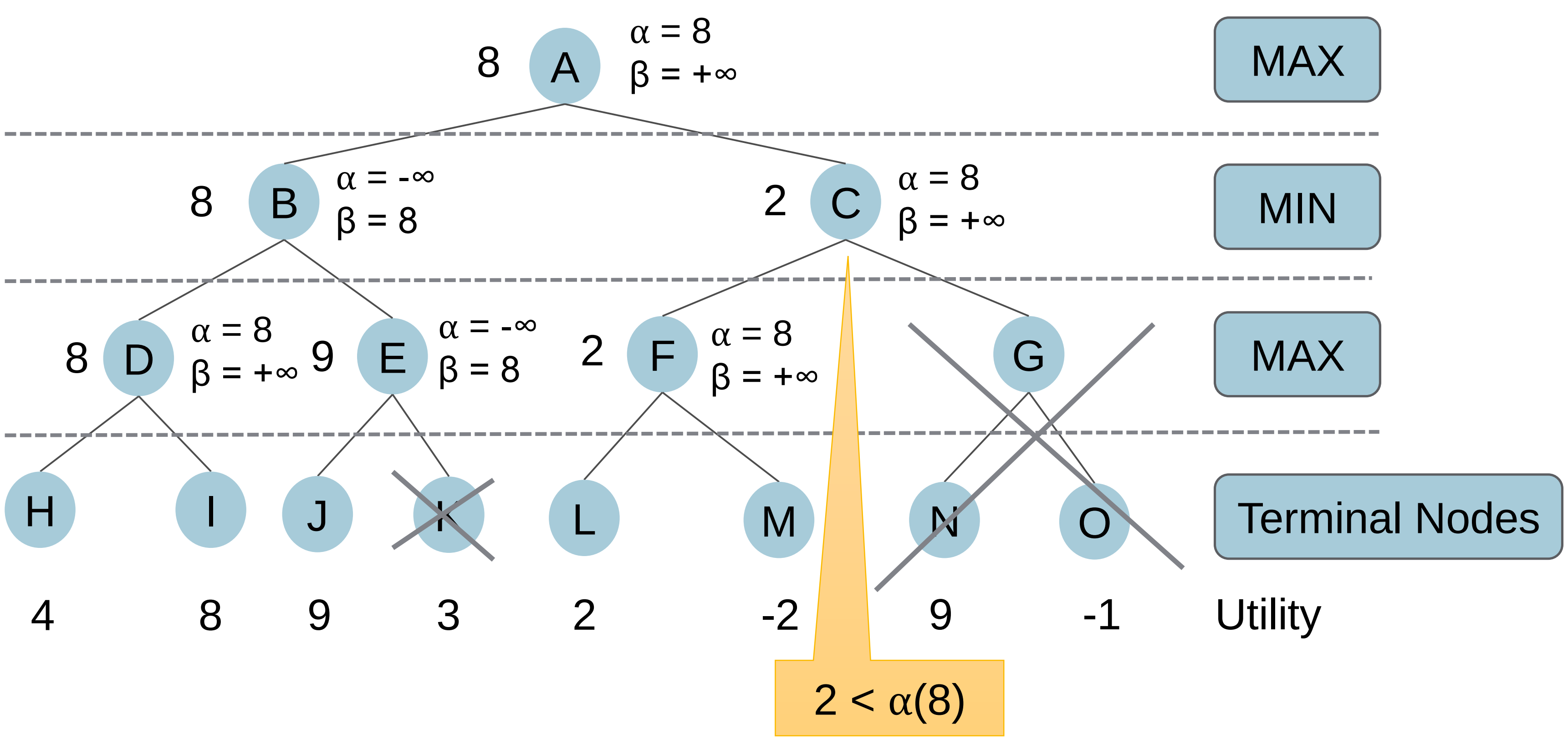


Step 8:

- Since we have not found a value >= α,
  MAX looks into **node M** to find a **value of -2.**
- -2 < α(8): α is not updated.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example

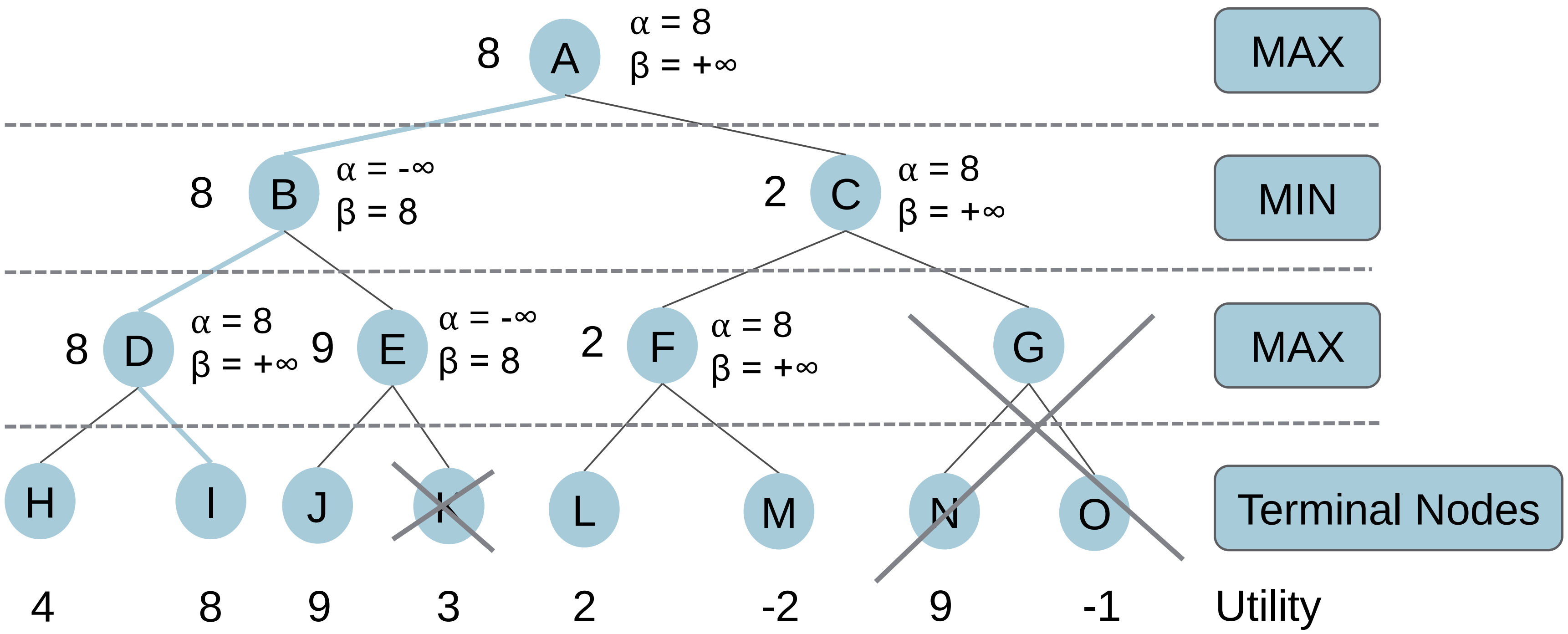


Step 9:

- In **node C** MIN finds the **value 2 of node F**.
- 2 < α(8): the **remaining branches** of **C** are **pruned**.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning :: Example



Result

- The resulting **path** is the **same** as in **Minimax,** but **fewer nodes** had to be analyzed.

Based on an example from John Levine (https://www.youtube.com/watch?v=zp3VMe0Jpf8)

Alpha-Beta Pruning

Properties of Alpha-Beta pruning

- **Does not affect** the **final result**.
- With perfect ordering, **time complexity** would be O($\mathbf{b^{m/2}}$).

Limitations of Minimax and Alpha-Beta Pruning

- Minimax traverses the **entire game tree**.
- Alpha-Beta pruning still has to **search all the way to terminal states** of **many nodes**.

Can we do better?

- While both algorithms have many applications,
  in certain scenarios they might **reach their limits**.
- This is where we can apply **Monte Carlo Tree Search**.

Games and Adversarial Search - Marco Chiarandini

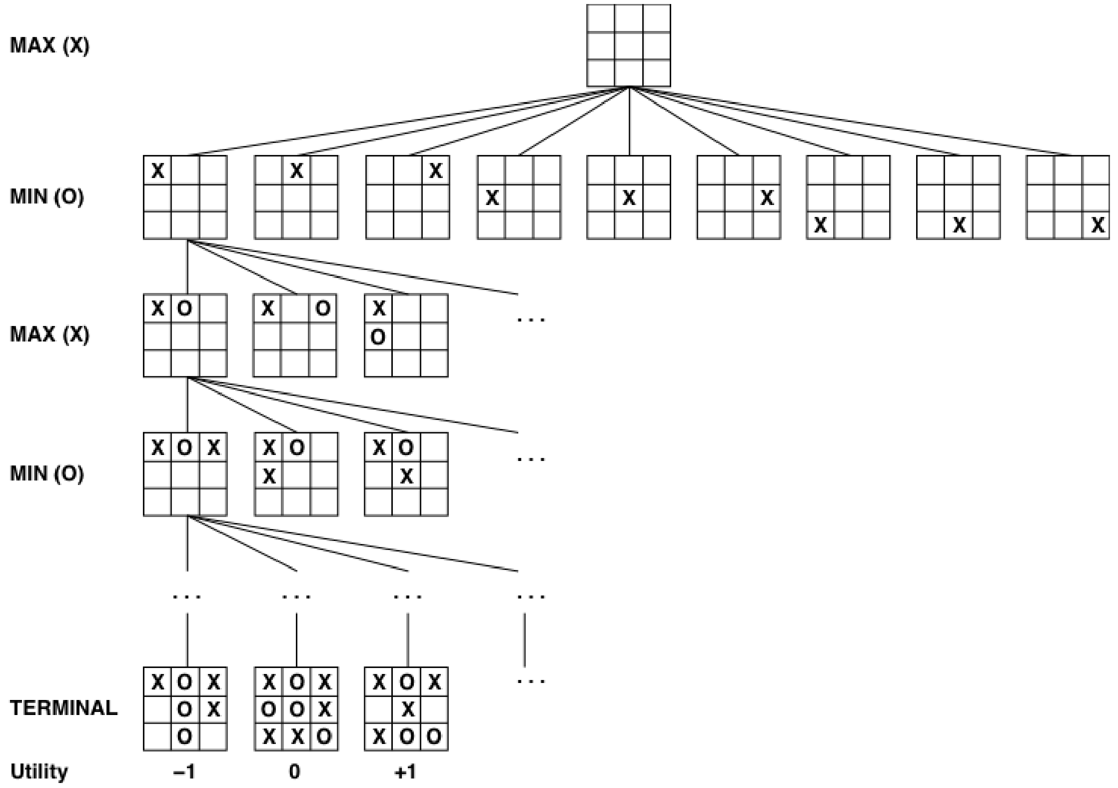
Game Trees and the Minimax Algorithm

How can MIN/MAX **determine which move** to pick to **win** the game?

We know for **each terminal state** the outcome of the game – this is called the **utility**.

In each turn, both players want to select a node which results in the best utility **for them.**

1. **Generate whole game tree** to leaves
2. Apply **utility function** to leaves
3. **Back up values** from leaves to root
   - MAX nodes compute maximum of children
   - MIN nodes compute minimum of children
4. When value **reaches root**: choose max value and the corresponding move



Deterministic, perfect information Tic-Tac-Toe game tree of 2 players (5,478 valid game states).

Monte Carlo Tree Search

Basic Algorithm Outline

The basic algorithm involves **iteratively building a search tree** until some **predefined computational budget** – typically a time, memory or **iteration constraint** – is **reached**.

At this point the search is **halted** and the **best performing root action** is returned.

Each **node** in the search tree represents a **state** of the domain.
Directed **links** to **child nodes** represent **actions** leading to **subsequent states**.

A Survey of Monte Carlo Tree Search Methods, Cameron Browne, 2012

Monte Carlo Tree Search

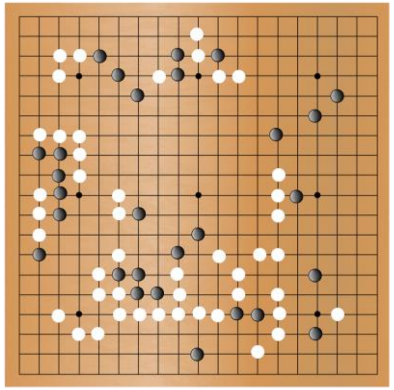
Motivation: the game of Go

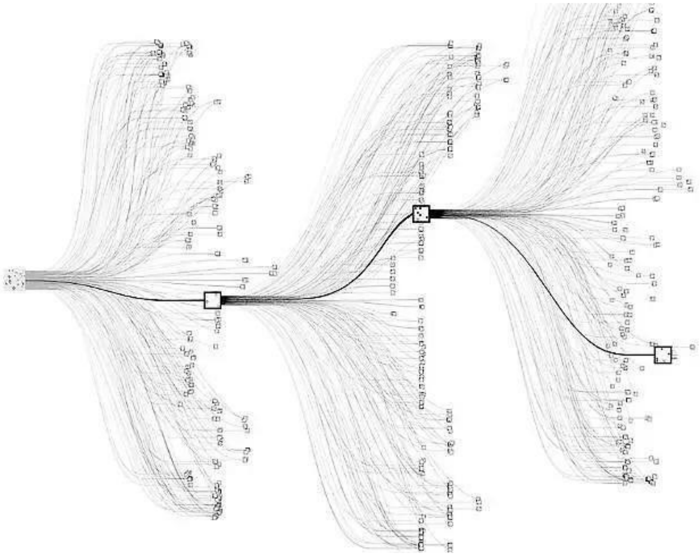
Played on a **19x19 board** by **two players** in **alternating moves** by placing **black** stones and **white** stones respectively on the board.

Opponent's stones can be **captured** once they are **fully surrounded** by own stones.

No move can lead to a game **state** that **has been present** in the move directly before (**no immediate repetitions**).

The **goal** is to **occupy** a **larger area** of the game board than the opponent.





There are **2.08*10^{170}** valid game sates in the game of Go.

Monte Carlo Tree Search

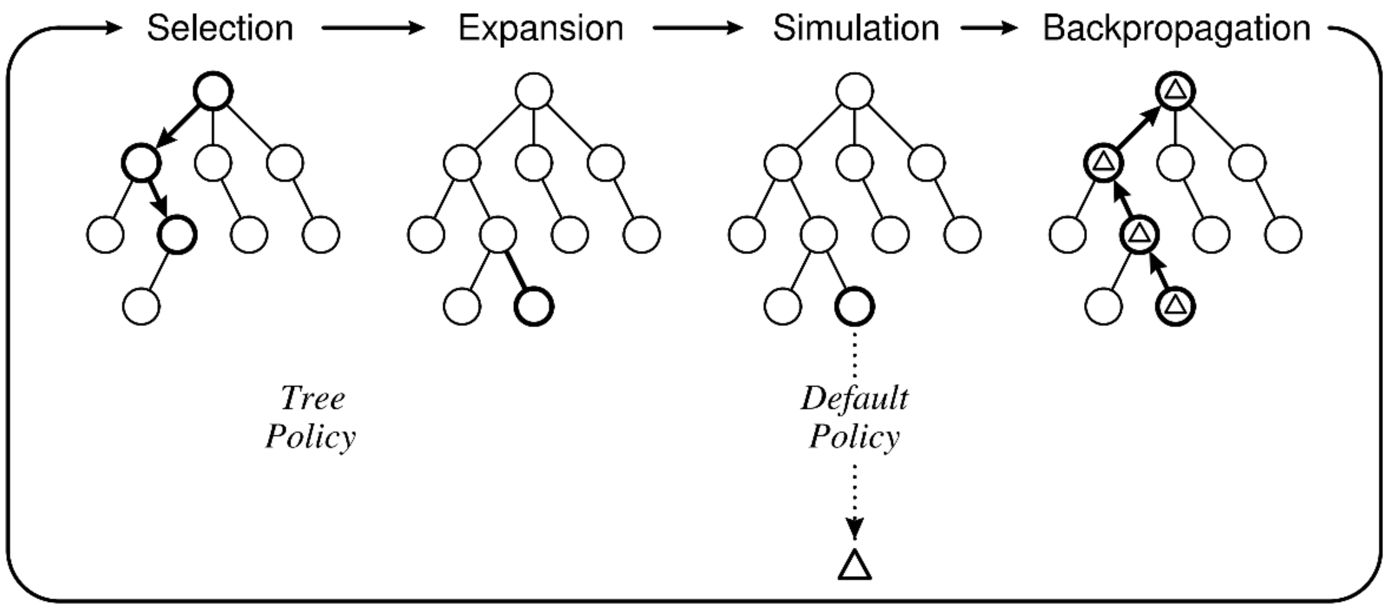
Basic algorithm Outline

Selection: **Find** a **leaf node** from which to traverse next.

Expansion: **Child nodes** are **added** to expand the tree.

Simulation: A **rollout** from the new node(s) to a **terminal** node is done.

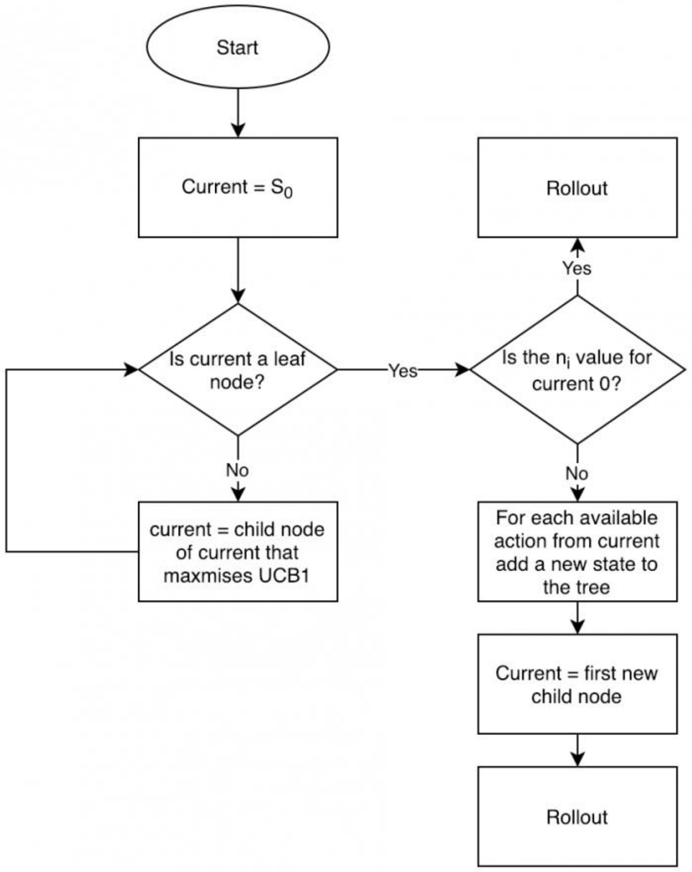
Backpropagation: The **simulation result** is "**backed up**" through the selected nodes to update their statistics.



A Survey of Monte Carlo Tree Search Methods, Cameron Browne, 2012

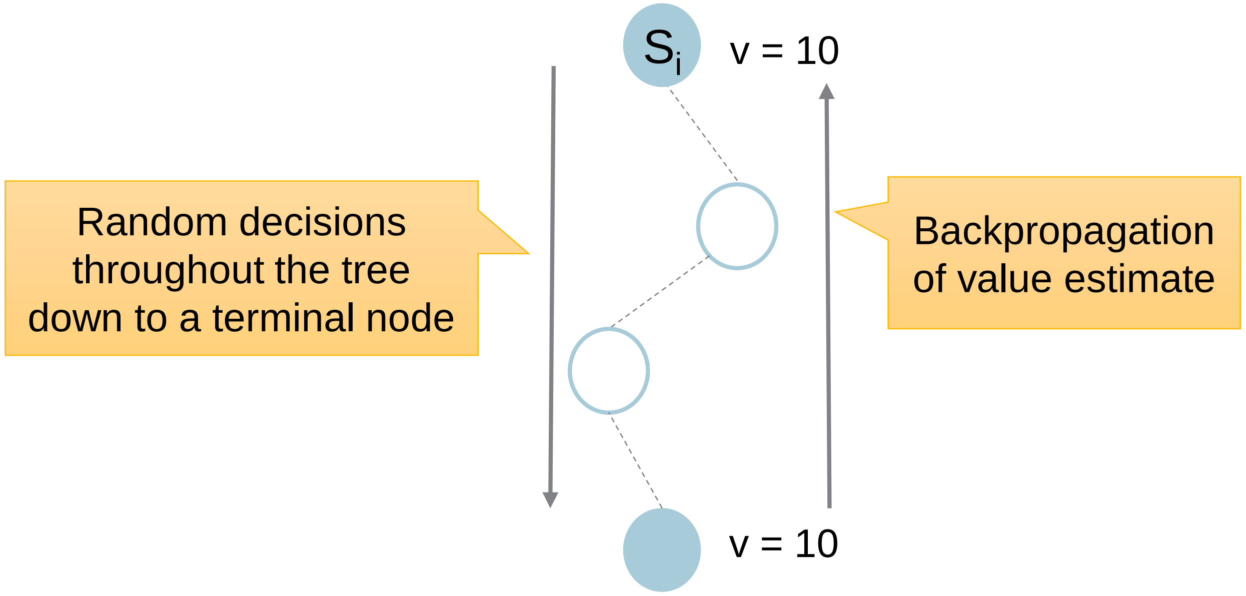
Monte Carlo Tree Search

Basic algorithm Outline



A Survey of Monte Carlo Tree Search Methods, Cameron Browne, 2012

```
Rollout(Sᵢ):
  loop forever:
    if Sᵢ is a terminal state:
      return value(Sᵢ)
    Aᵢ = random(available_actions(Sᵢ))
    Sᵢ = simulate(Aᵢ, Sᵢ)
```



Monte Carlo Tree Search

How to select a leaf node?

Similarly to Minimax, we need to find some value that gives the **branch** a **score**, determining the **most promising path**.

Tree Policy

In MCTS the most widely used utility function is called **Upper Confidence Bound** (UCB1).

$$\text{UCB1} = \underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(\underbrace{N}_{\text{total number of trials}})}{\underbrace{n_i}_{\text{num trials for arm i}}}}$$

A value of 2 for the tunable parameter C has been used in the past to yield promising results.

A Survey of Monte Carlo Tree Search Methods, Cameron Browne, 2012

Monte Carlo Tree Search

How to choose which route to follow in the expanded node?

Once we **decided which branch to expand**, we need to **find some strategy** to **traverse** through that branch down to its terminal node.

Default Policy

Play out the domain from a given non-terminal state to produce a **value estimate** (simulation). In the **simplest** case these are **just random moves**.

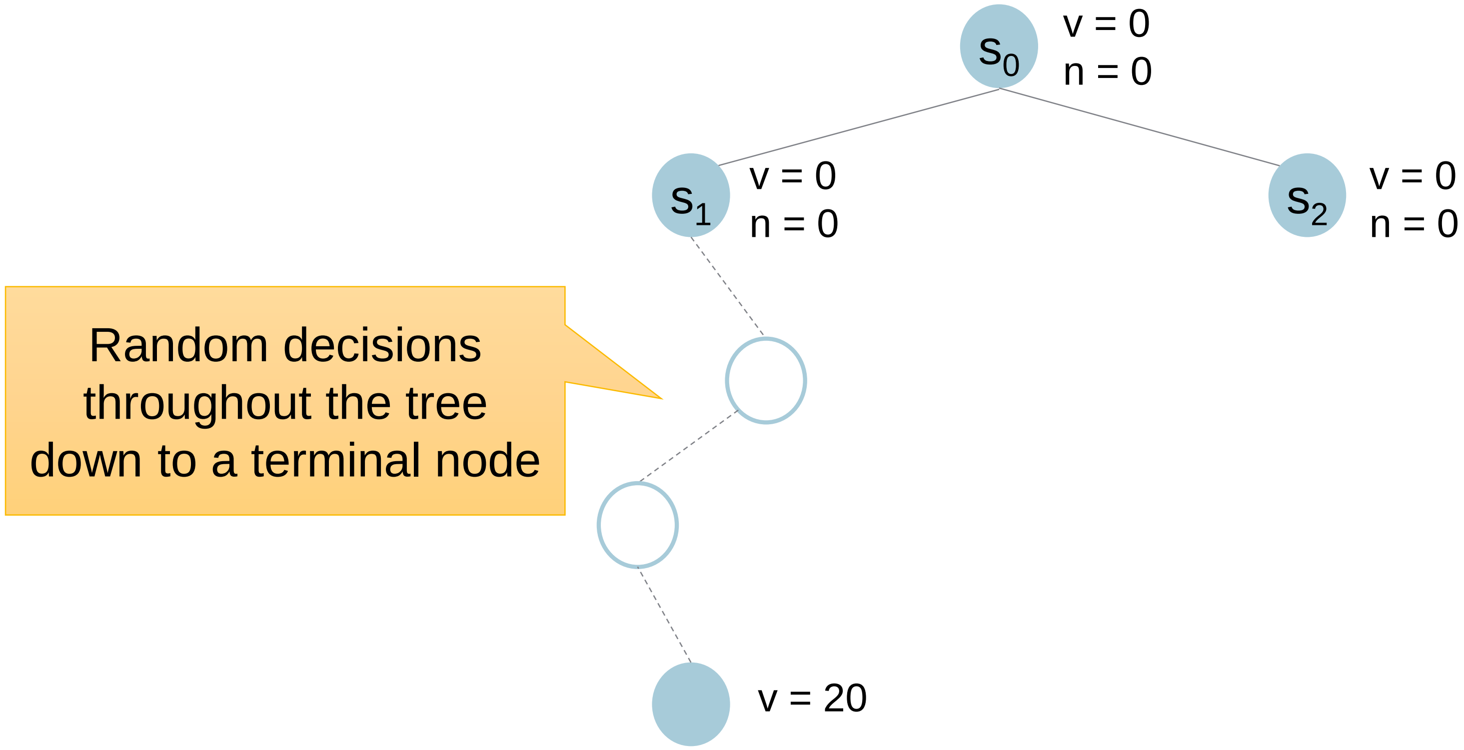
A Survey of Monte Carlo Tree Search Methods, Cameron Browne, 2012

Monte Carlo Tree Search :: Example

Iteration 1 (Simulation Phase)

Since s_1 is a **leaf node** that has **not** been **visited yet**, we **perform** a **rollout** to a terminal state.

Via simulation we get a value estimate of 20. Remember, this is the result when **playing out** this **branch to** the **end**.



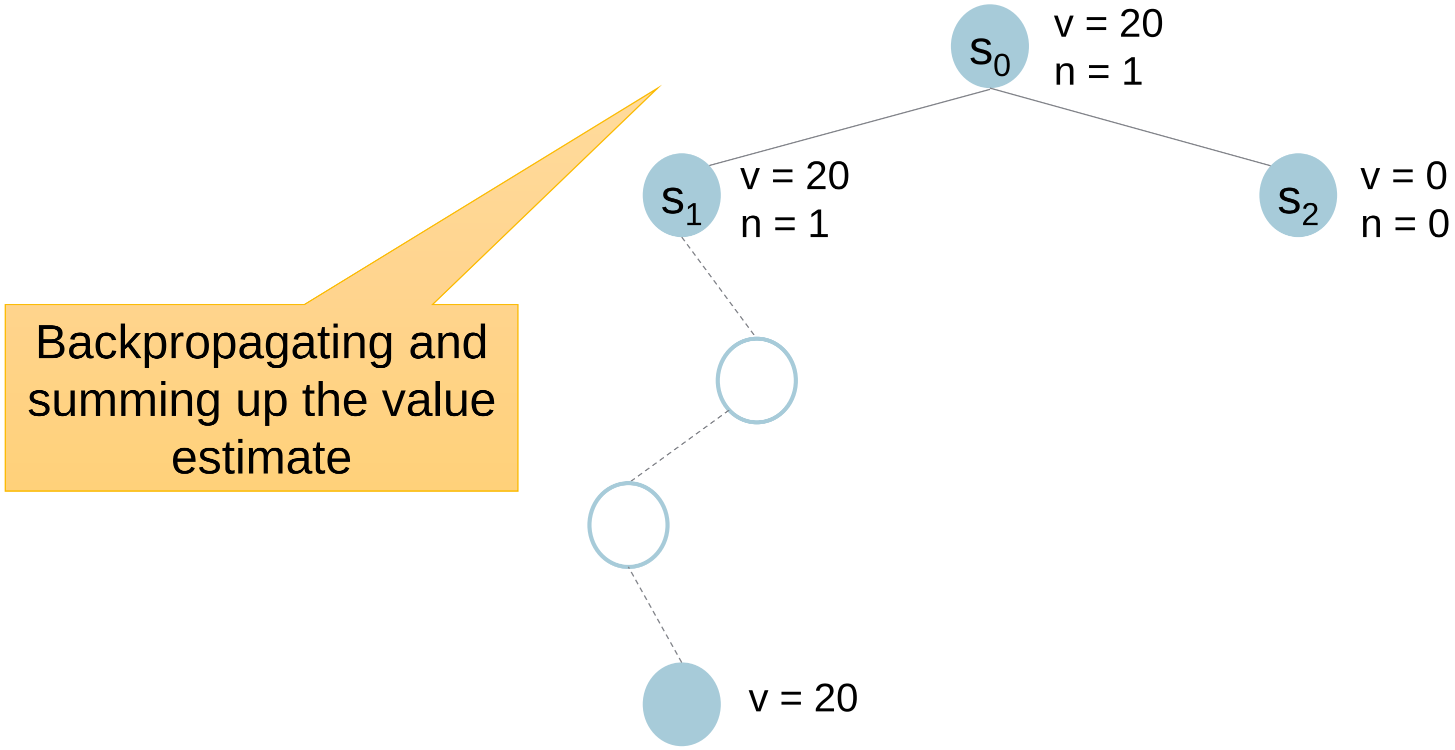
Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 1 (Backpropagation Phase)

The value estimate is **backpropagated** up to the **root node**.

This **concludes** the **first iteration**.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte carlo tree search :: Example

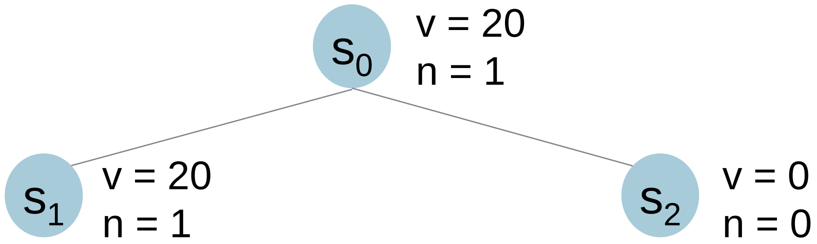
Iteration 2 (Selection Phase)

In the second iteration, again **starting** in $\mathbf{s_0}$, we calculate the **UCB1** (Upper Confidence Bound) scores for $\mathbf{s_1}$ and $\mathbf{s_2}$.

$$\text{UCB1}(s_1) = 20 + 2\sqrt{\frac{\ln(1)}{1}} = 20$$

UCB1(s_2) is still infinite.

Since the UCB1 **score** of $\mathbf{s_2}$ is **higher** we select this branch.



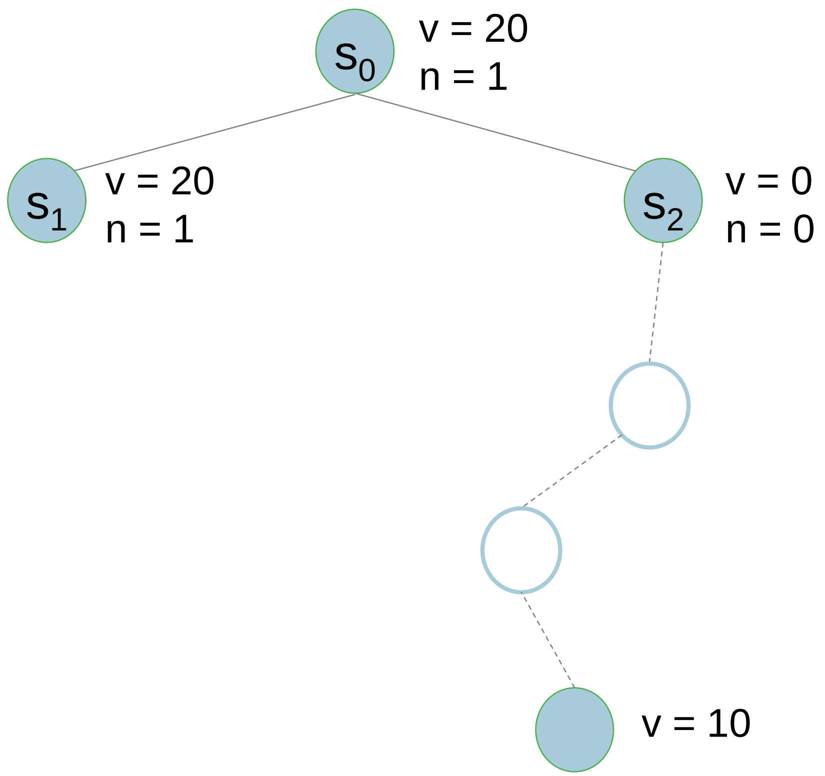
Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 2 (Simulation Phase)

Since $\mathbf{s_2}$ is a leaf node which has **not been** visited yet, we perform a **rollout to** a **terminal state**.

Via simulation we get a value estimate of 10.



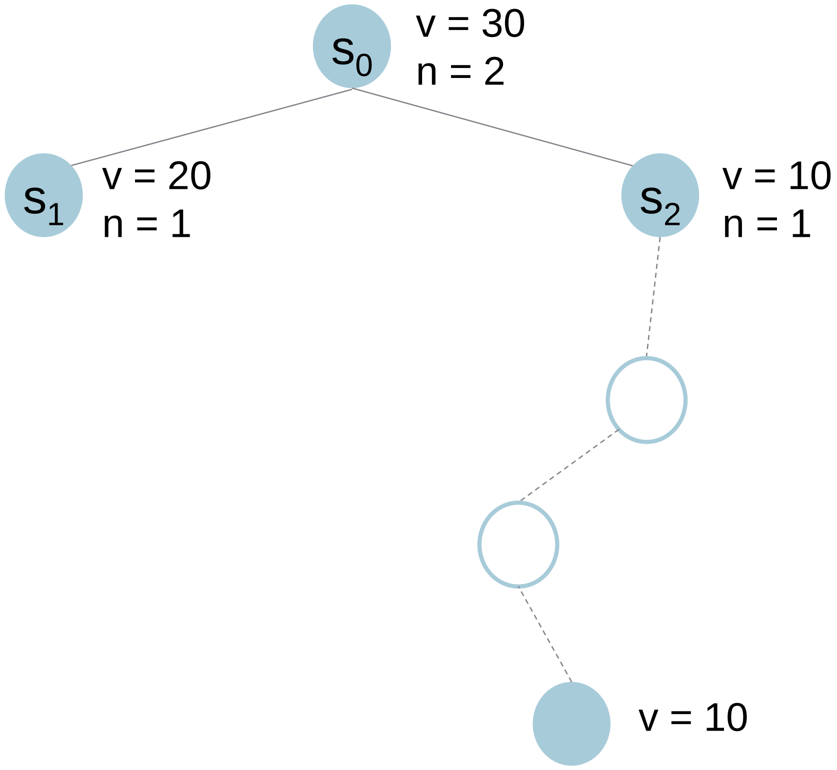
Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 2 (Backpropagation Phase)

The **value estimate** is **backpropagated** up to the **root node**.

This concludes the second iteration.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

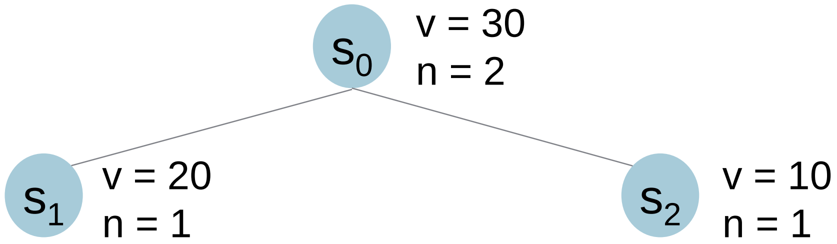
Iteration 3 (Selection Phase)

In the third iteration, again **starting** in $\mathbf{s_0}$, we **calculate** the **UCB1** scores for $\mathbf{s_1}$ and $\mathbf{s_2}$.

UCB1($\mathbf{s_1}$) $= 20 + 2\sqrt{\frac{\ln(2)}{1}} = \mathbf{21.67}$

UCB1($\mathbf{s_2}$) $= 10 + 2\sqrt{\frac{\ln(2)}{1}} = \mathbf{11.67}$

Since the UCB1 score of $\mathbf{s_1}$ is **higher** we **expand** this branch.

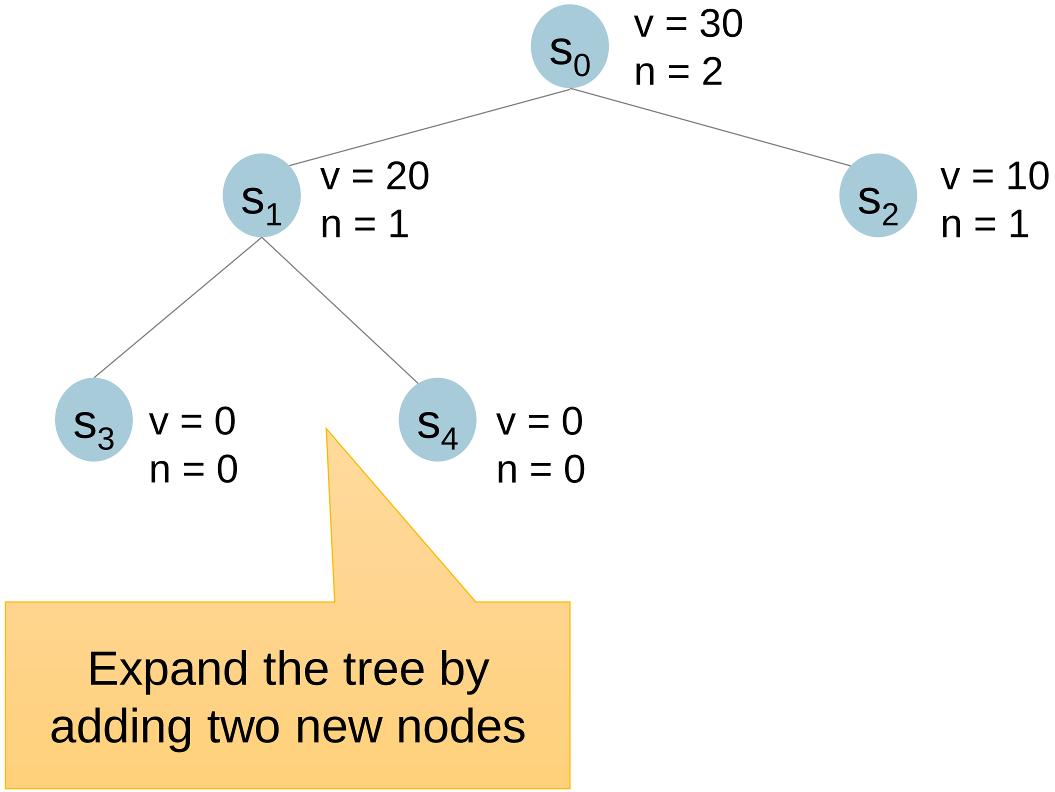


Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 3 (Expansion Phase)

Since $\mathbf{s_1}$ is a **leaf** node but has **already** been **visited**, we expand the tree.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

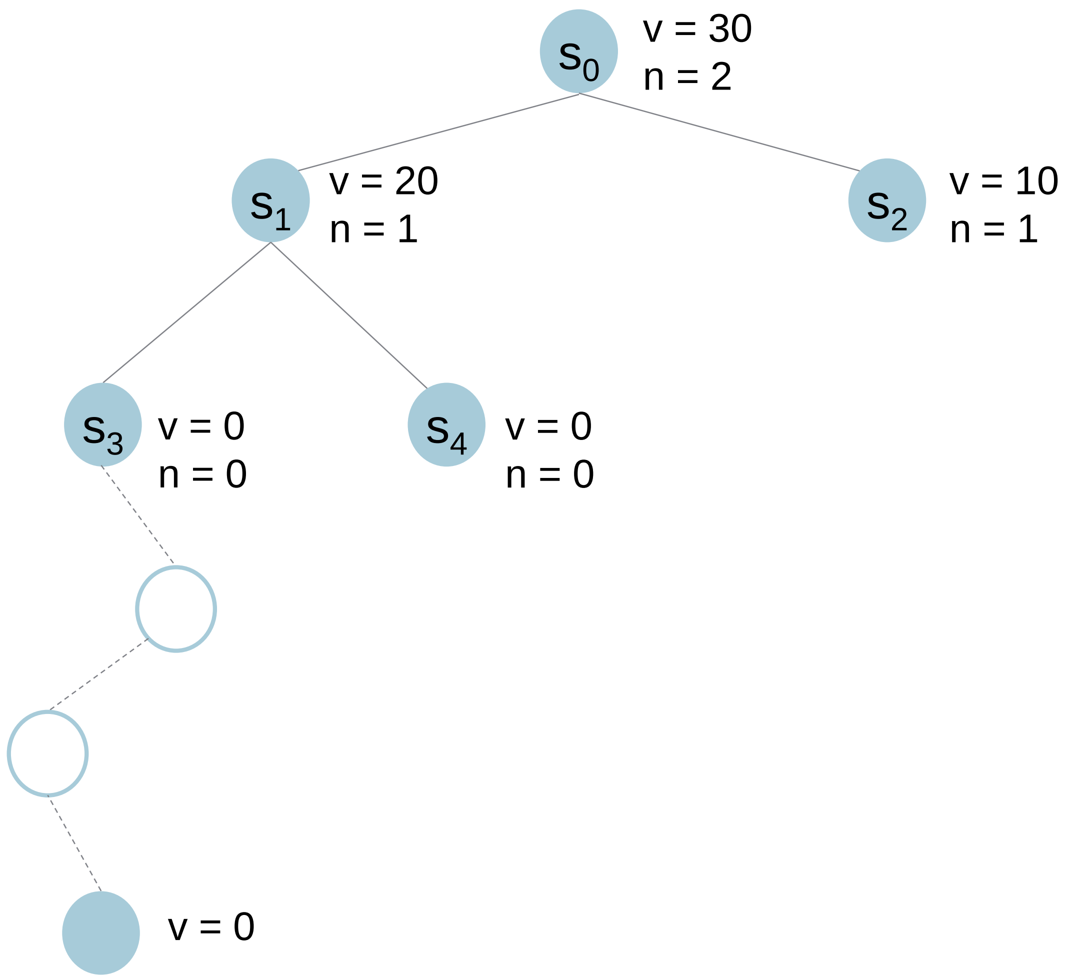
Monte Carlo Tree Search :: Example

Iteration 3 (Simulation Phase)

We **calculate** the **UCB1** score for **both** nodes, $\mathbf{s_3}$ **and** s_4.

Since the UCB1 score for both children **infinite** (no one has been visited before) we can again **choose** to start with the leftmost child which is s_3**.**

We perform the **rollout** and via **simulation** get a value estimate of **0**.



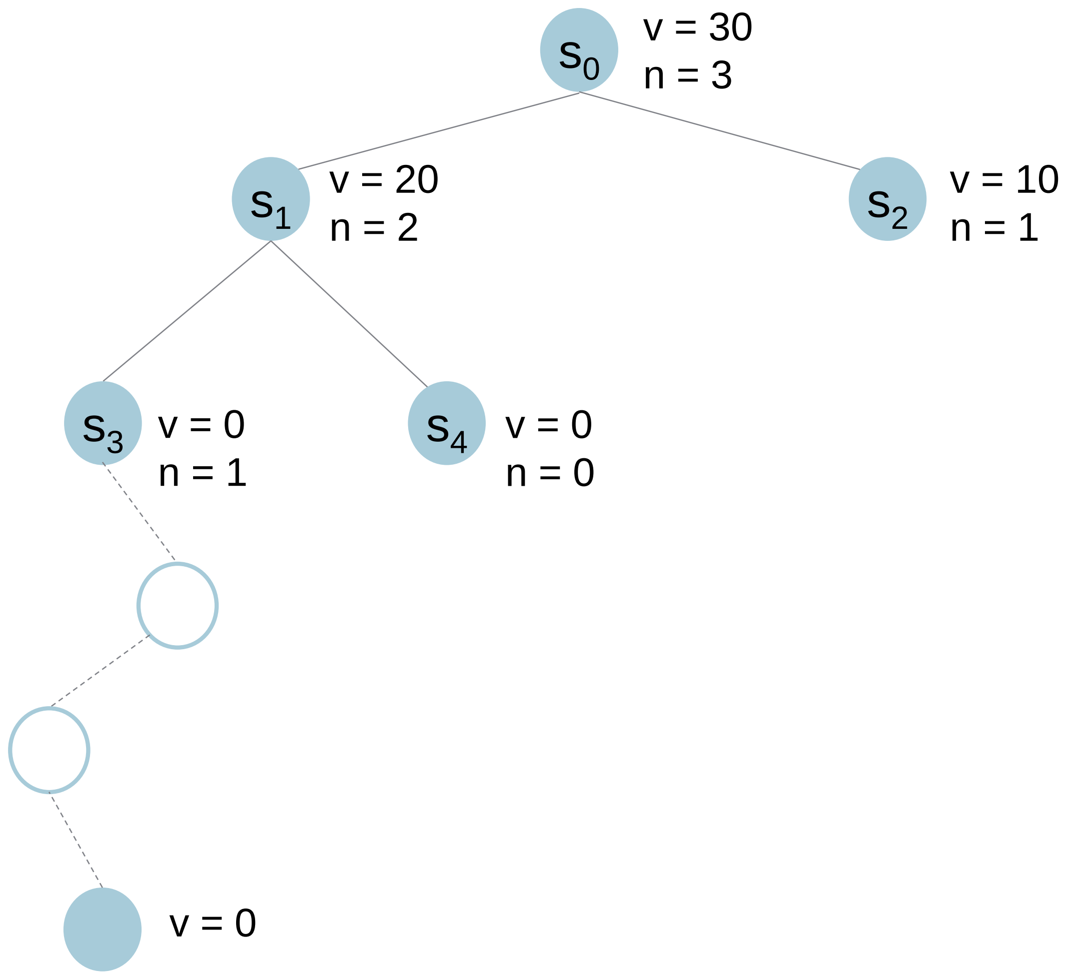
Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 3 (Backpropagation Phase)

The **value estimate** is now **backpropagated** up to the **root** node.

This concluses the third iteration.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

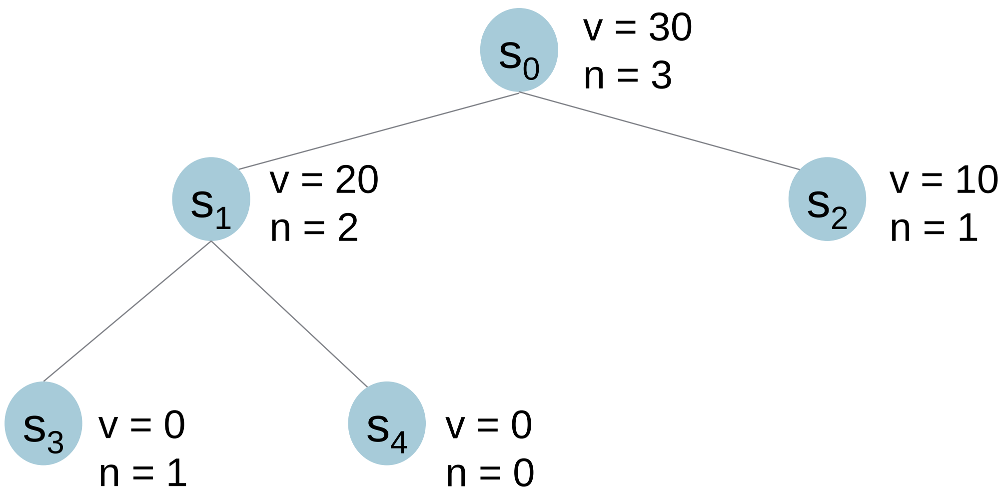
Iteration 4 (Selection Phase)

In the fourth iteration, again starting in s_0, we **calculate** the **UCB1** scores for $\mathbf{s_1}$ and $\mathbf{s_2}$.

UCB1(s_1) = 20 +2$\sqrt{\frac{\ln(3)}{2}}$ = **11.48**

UCB1(s_2) = 10 +2$\sqrt{\frac{\ln(3)}{1}}$ = **12.10**

Since the UCB1 score of $\mathbf{s_2}$ is **higher** we explore this branch.

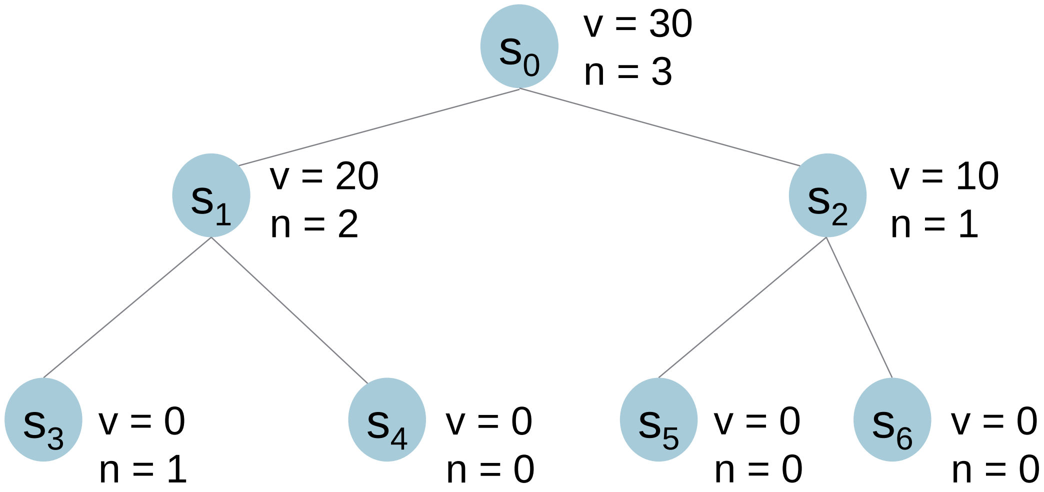


Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 4 (Expansion Phase)

Since s_2 is a leaf node but has **already** been **visited**, we **expand** the tree. (s_5, s_6)



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

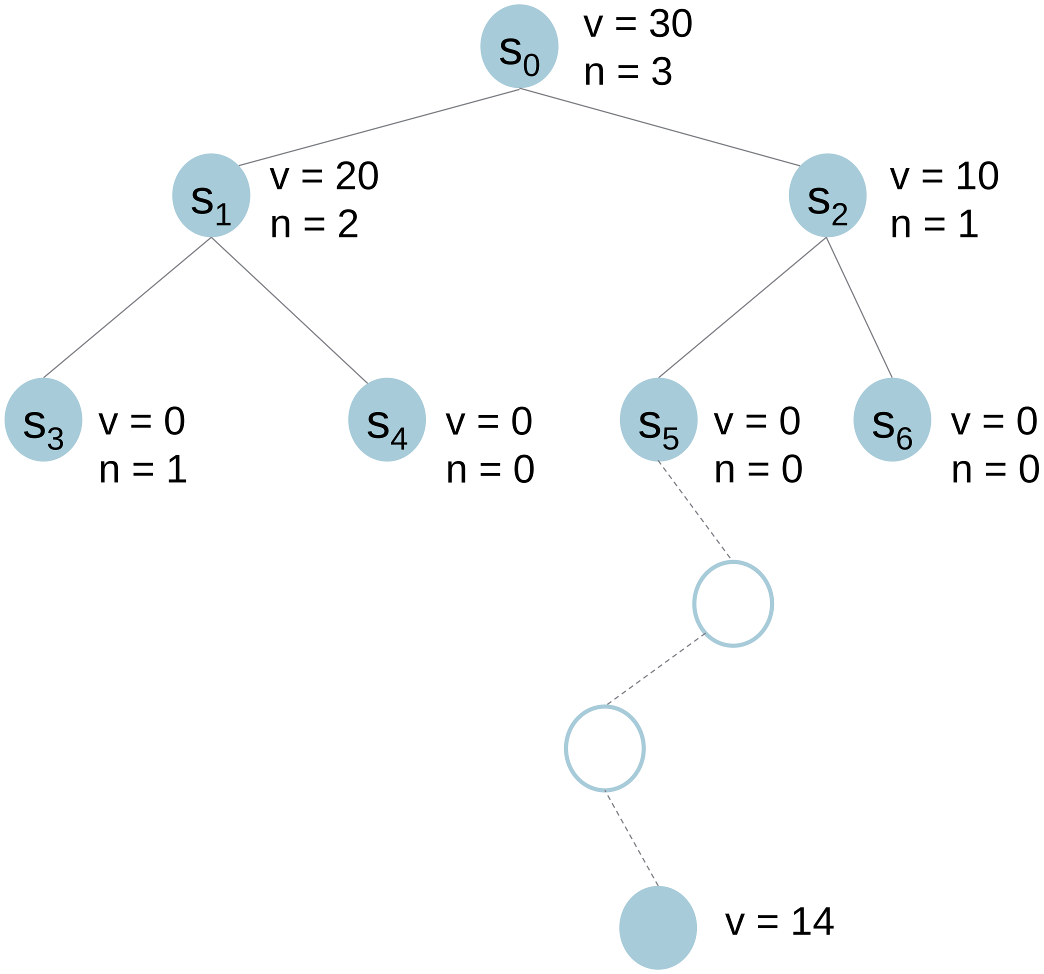
Monte Carlo Tree Search :: Example

Iteration 4 (Simulation Phase)

We **calculate** the **UCB1** score for both nodes, **s_5** and **s_6**.

Since the **UCB1 score** for **both** children is infinite (both have not been visited before) we can again **choose** to start with the **leftmost** node which is s_5.

We **perform** a **rollout** and via **simulation** get a value estimate of **14**.



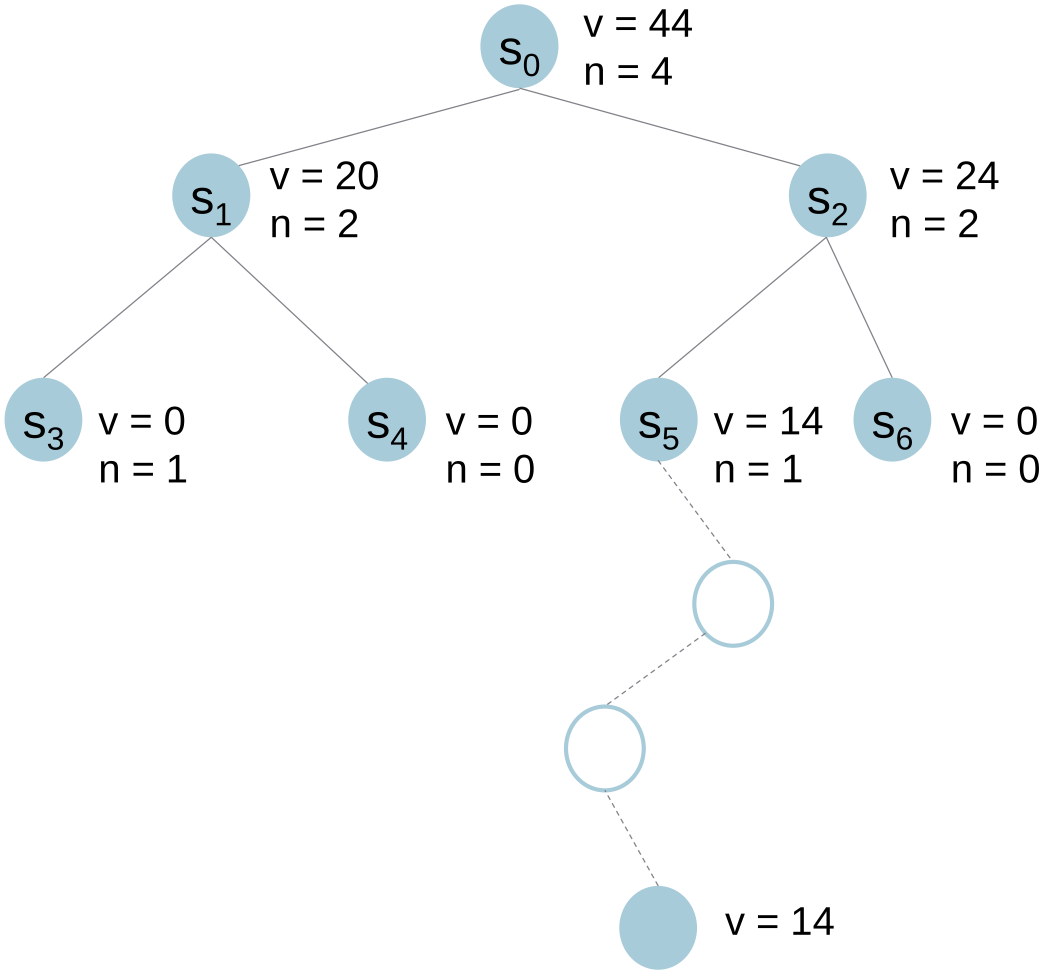
Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

Monte Carlo Tree Search :: Example

Iteration 4 (Backpropagation Phase)

The **value estimate** is **backpropagated** up to the **root** node.

This concludes the fourth iteration.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)

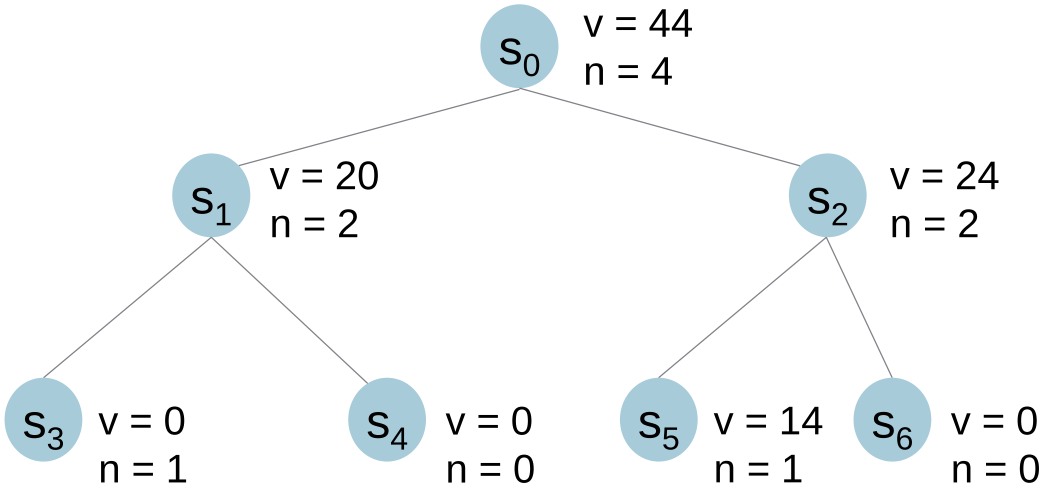
Monte Carlo Tree Search :: Example

Result

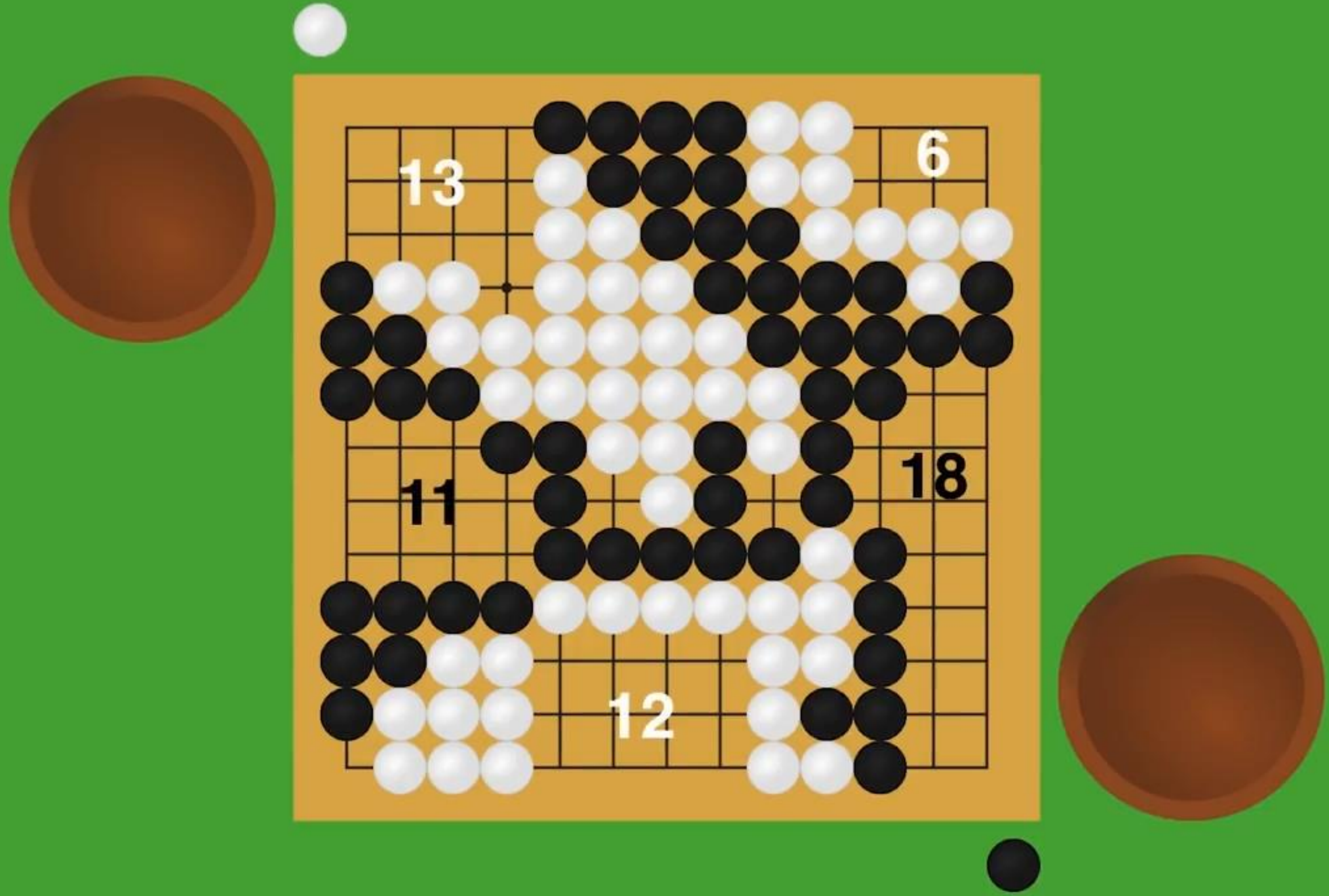
Following the **design** of the **MCTS algorithm** we could **do as many iterations** as we want.

However, if we were to **stop** now, the **branches** with the **highest total scores** would be optimal to choose (which is s_2 followed by s_5).

More iterations often **improve results**.



Based on an example from John Levine (https://www.youtube.com/watch?v=UXW2yZndl7U)


13
6
11
18
12

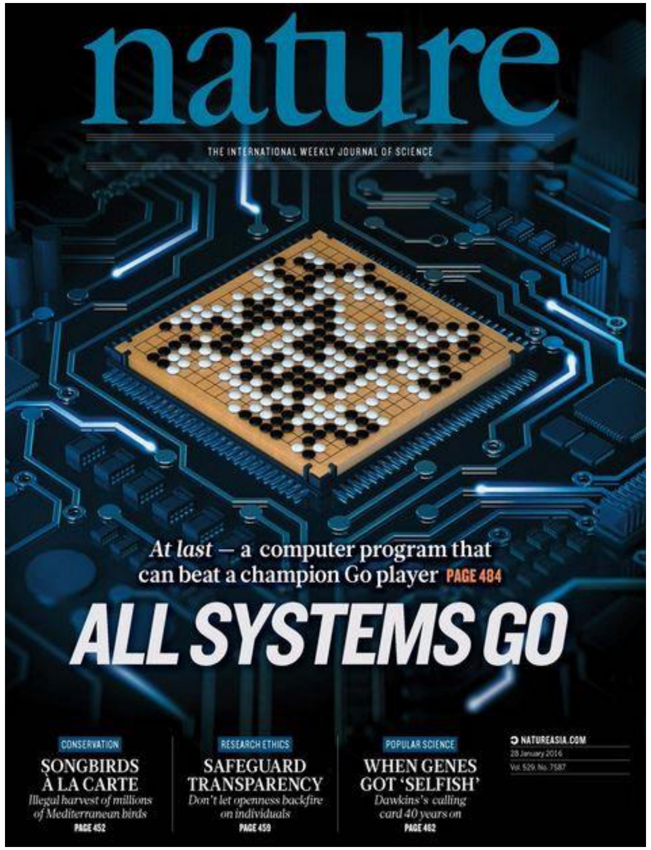
Solving the Game of Go

Background

- Remember, in a 19x19 Go board there are **$2.08*10^{170}$** valid game states.
- While boards with the size of 5x5 have successfully been solved in 2002, **19x19** boards have **long been assumed unsolvable**.

Approach by DeepMind via AlphaGo

- In **2015 DeepMind** realized their idea of solving **Go via machine learning** and **MCTS**.
- They **combine two approaches** in their implementation:
  - ***Value networks*** to **evaluate board positions** and ***policy networks*** to **select moves**.
  - A **search algorithm** that combines Monte Carlo simulation with **value** and **policy networks**.



Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

Solving the Game of Go

Training pipeline of AlphaGo

Rollout policy and Supervised Learning (SL) policy

- A **rollout policy** trained on **8 million human expert moves** (accuracy of 24.2% in just 2μs).
- A **13-layer convolutional neural network** trained on **30 million moves** of human experts (accuracy of 57% while best result of other research groups was 44.4%).

Reinforcement Learning (RL) policy

- Aims to improve the **Supervised Learning** (SL-)policy **through self-play** by having the same architecture as the SL-policy but initializing it with the final RL-weights.
- This adjusts the policy towards the **correct goal of winning games** rather than maximizing predictive accuracy.

Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

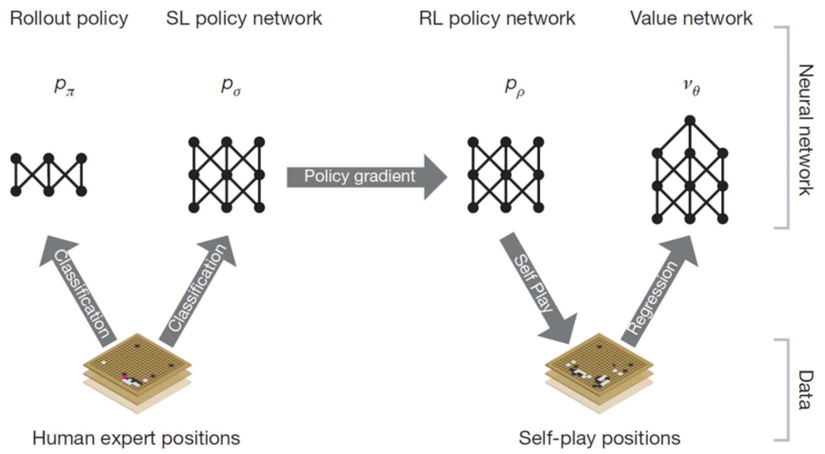
Solving the Game of Go

Training pipeline of AlphaGo (cont'd)

Value Networks

- A **value network** approximates the **optimal value function**.
- Trained by 30 million moves sampled from distinct games of self-play from RL-policy.
- While the **policy networks** reveal which **moves** are **promising**,
  the **value network** determines how **good** a **board position** is.

Network overview



Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

Solving the Game of Go

MCTS during live play

- Up until now the **models are trained** but **still have to be processed**.
- Right now the **network does not play any better than** any **state-of-the-art MCTS algorithm**.
- The key factor is combing the neural networks with MCTS in what is called *asynchronous policy and value MCTS (APV-MCTS).*

Evaluation of terminal nodes

(1) by the value network $v_\theta(s_L)$.

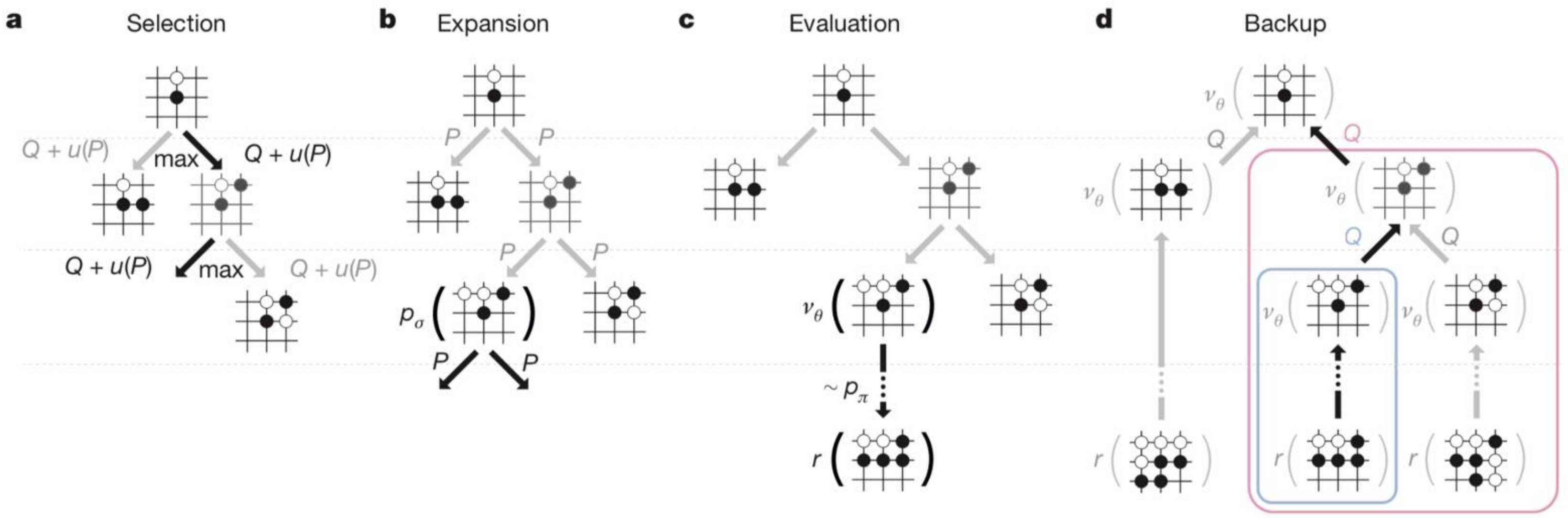
(2) by the outcome of MCTS simulations z_L.

These evaluations are combined into a terminal node evaluation using a mixing parameter λ tuned to 0.5 : $V(s_L) = (1- \lambda)v_\theta(s_L) + \lambda z_L$

Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

Solving the Game of Go

Evaluation of terminal nodes



Play strength of AlphaGo

When released in 2015, AlphaGo won against European Go champion ***Fan Hui*** followed by winning 4 out of 5 matches against 18 times world champion ***Lee Se-dol***.

Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

Institute of
Pervasive Computing

Monte Carlo Tree Search

Algorithms and Data Structures 2, 340300
Lecture – 2023W
Univ.-Prof. Dr. Alois Ferscha, teaching@pervasive.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at