ADVERSARIAL GAMES

Thomas Tiam-Lee, PhD

De La Salle University Manila
College of Computer Studies

Adversarial Games

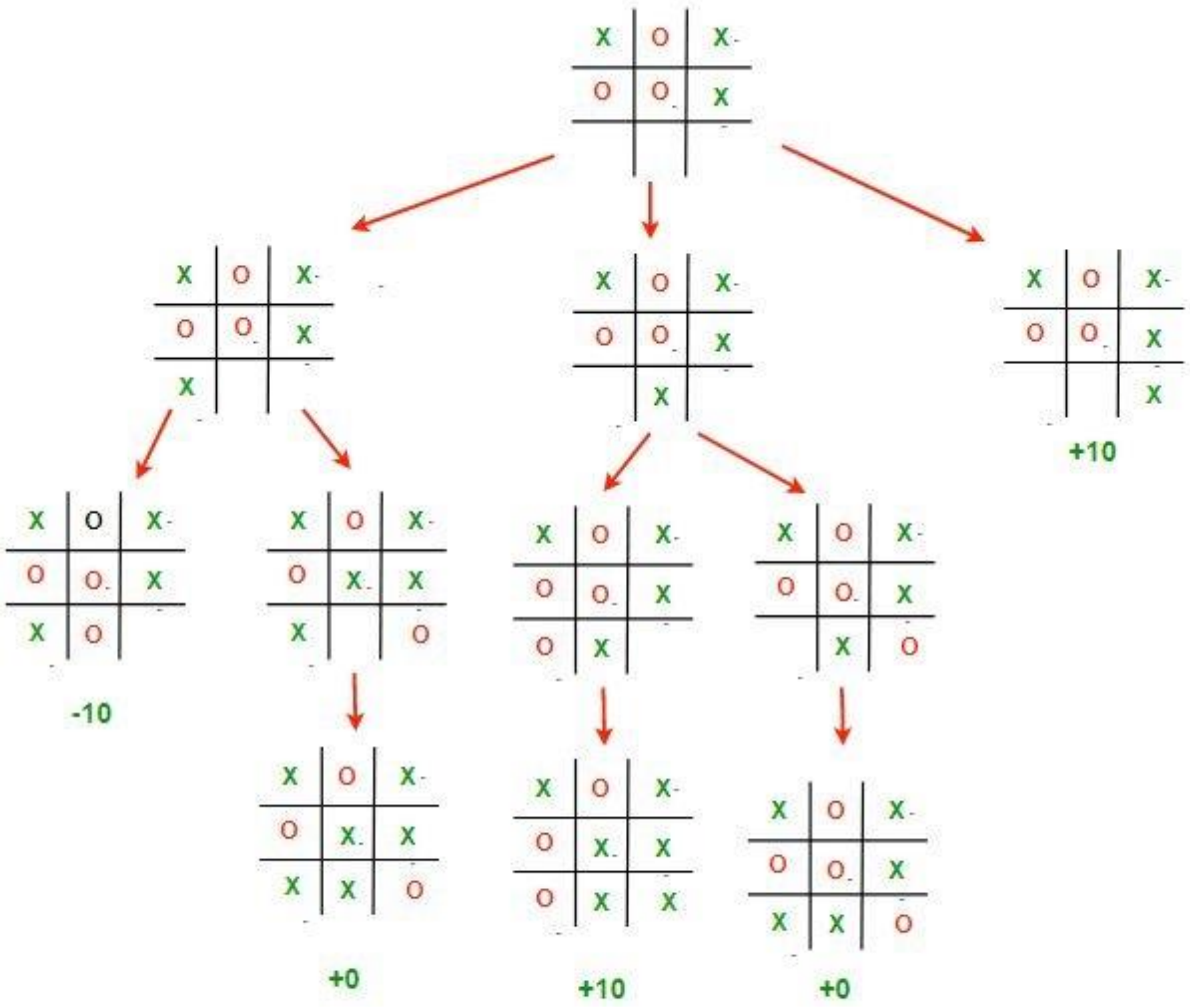
- In some real-world situations, the agent (you) is not the only one making decisions.
- Notable example: board games
  - There is an "opponent" that decides an action at certain points.
  - The opponent is often active trying to prevent you from your goal.

New Challenges

- Opponent can behave unpredictably, so we cannot plan a clear path from the beginning to the goal.
- Time limit: in some cases, decision must be made quickly, so may need to approximate.

Modeling: Game Tree

- Similar to a search tree
- Each node represents **a decision point** for either player
- Each path from root to leaf represents a unique run of the game

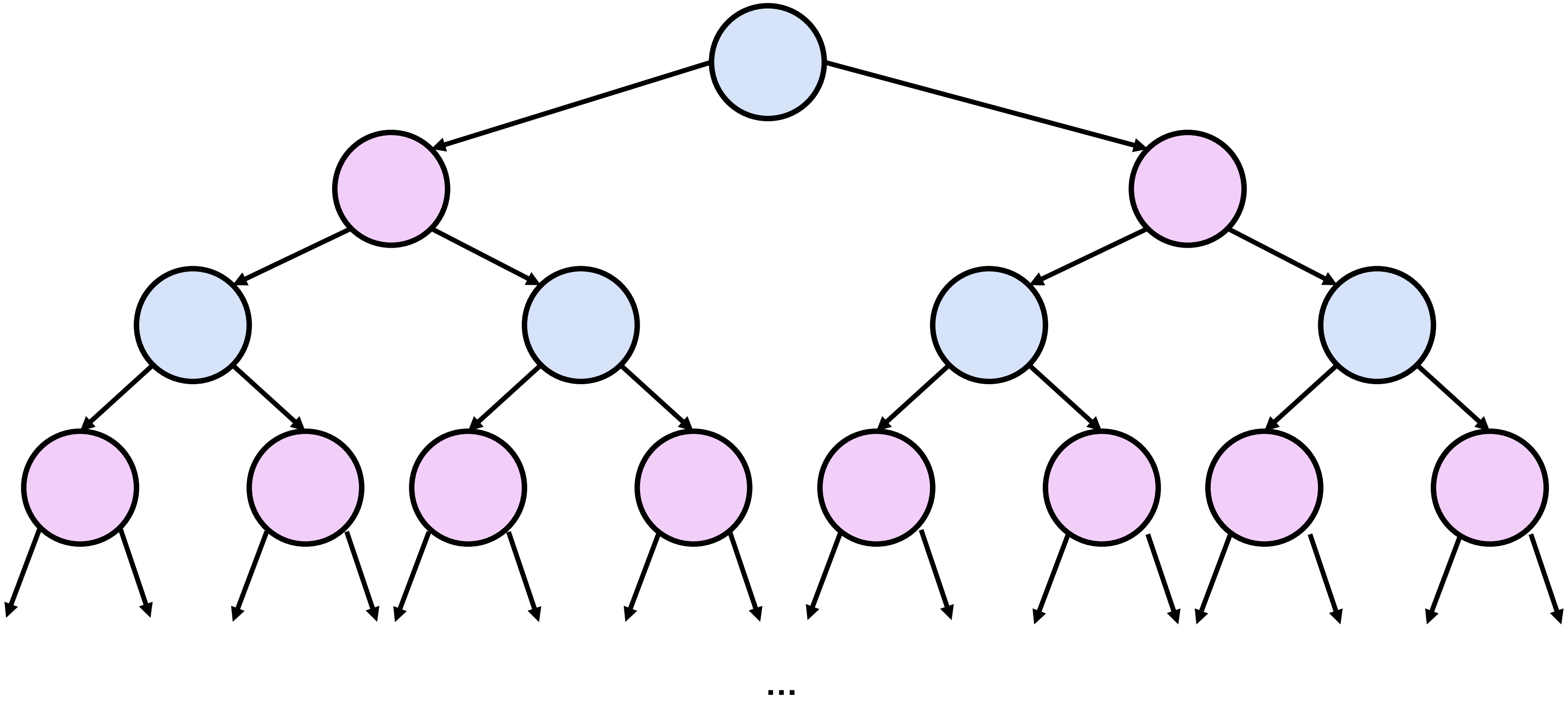


Source: geeksforgeeks.org

Terminologies

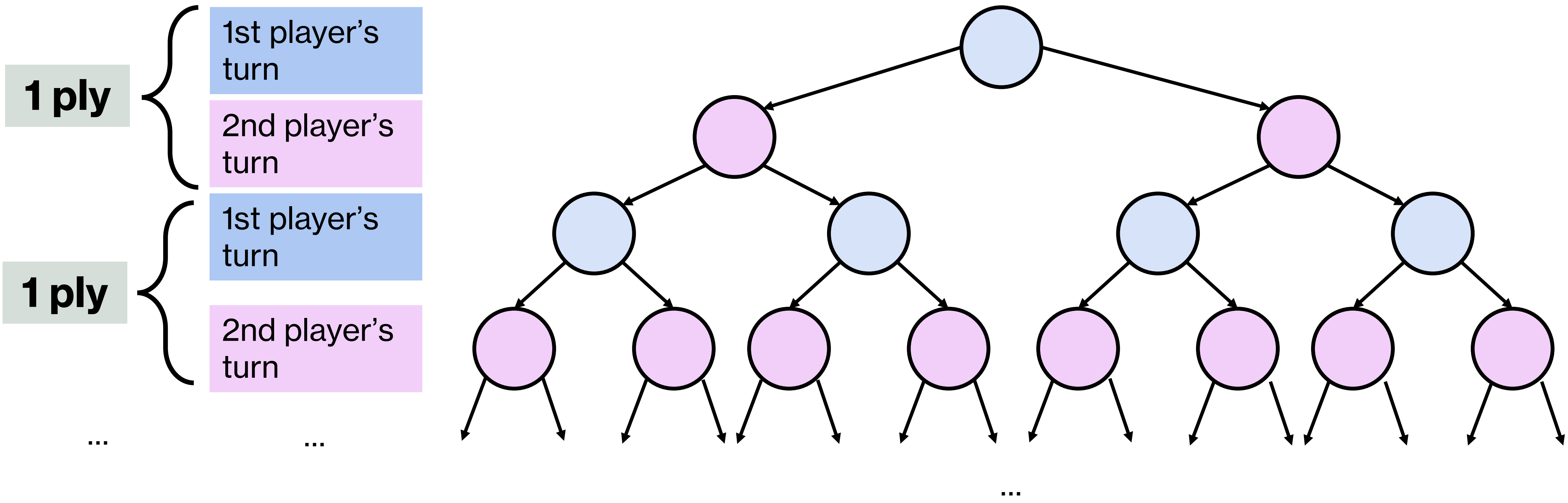
- **Depth:** the number of turns from the start state.

Depth	
0	1st player's turn
1	2nd player's turn
2	1st player's turn
3	2nd player's turn
…	…



Terminologies

- **Ply:** A single cycle of all player's turns.



2-Player Zero Sum Game

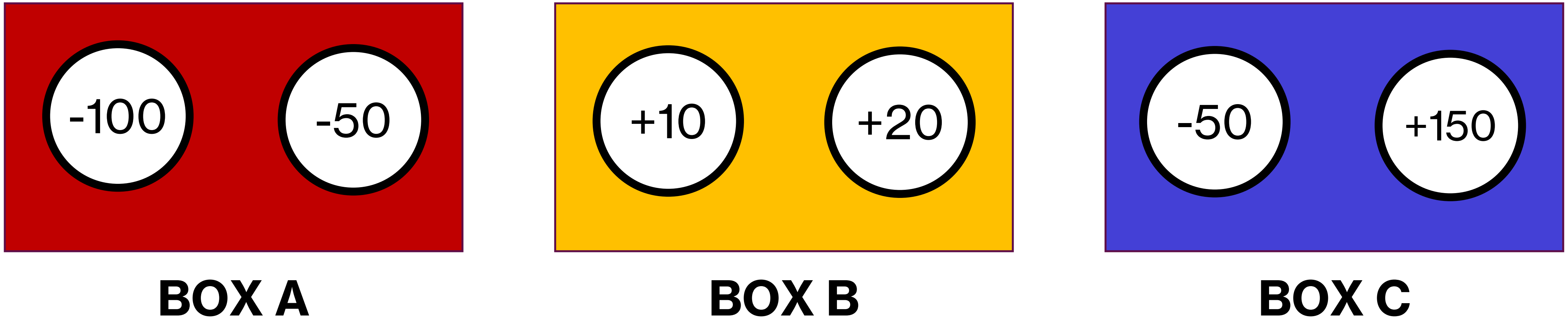
- Game played with 2 players (you and an opponent)
- One player's gain is equivalent to the other player's loss
- Games where if one person wins, the other person loses (by the same degree)
- Examples:
  - Tic Tac Toe
  - Chess
  - Checkers

Zero Sum Game Formulation

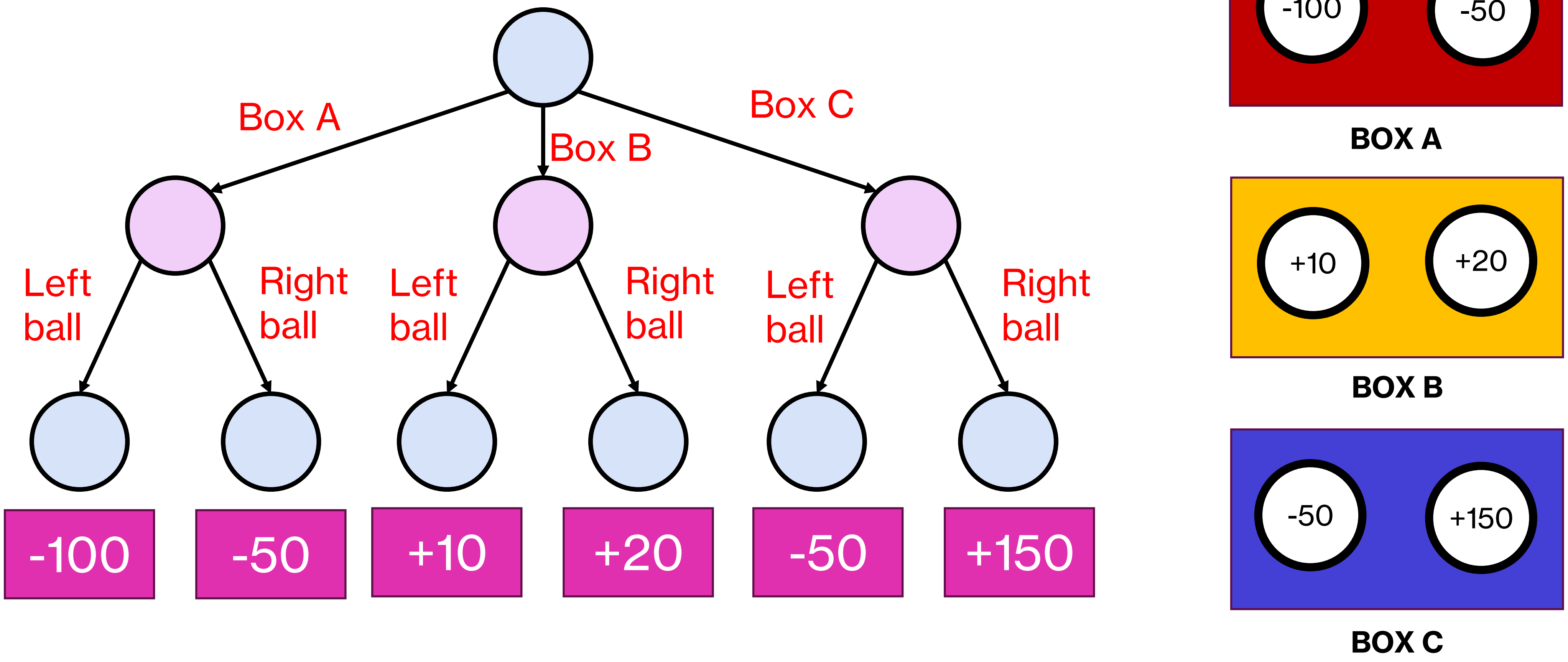
- $Players = \{agent, opp\}$
- s_{start}: the start state
- $Actions(s)$: possible actions from state s
- $Succ(s, a)$: resulting state when choosing action a from state s
- $IsEnd(s)$: true if s is an end state (game is finished) or false otherwise
- $Utility(s)$: agent's utility at an end state s
- $Player(s)$: whose turn it is at state s (agent or opponent)

Example: Money Game

- Player 1 (you) picks a box (A, B, or C)
- Player 2 (the opponent) picks a ball from the box that was chosen.
- If positive, player 2 must pay that amount to player 1. If negative, player 1 must pay that amount to player 2.



Game Tree for Money Game



Modeling the Opponent Behavior

- **Policy:** a mathematical model that defines how the opponent behaves at a given action point.

Case 1: “Dumb” Opponent

- **Opponent Policy:** the opponent picks a random move with equal probability.
    - $\pi_{opp}(s, a) = \frac{1}{2}$ for $a \in Actions(s)$

Case 2: Stochastic Opponent

- **Opponent Policy:** the opponent picks a move based on some probability distribution.

- $\pi_{opp}(s, a) \in [0,1]$

Expectimax

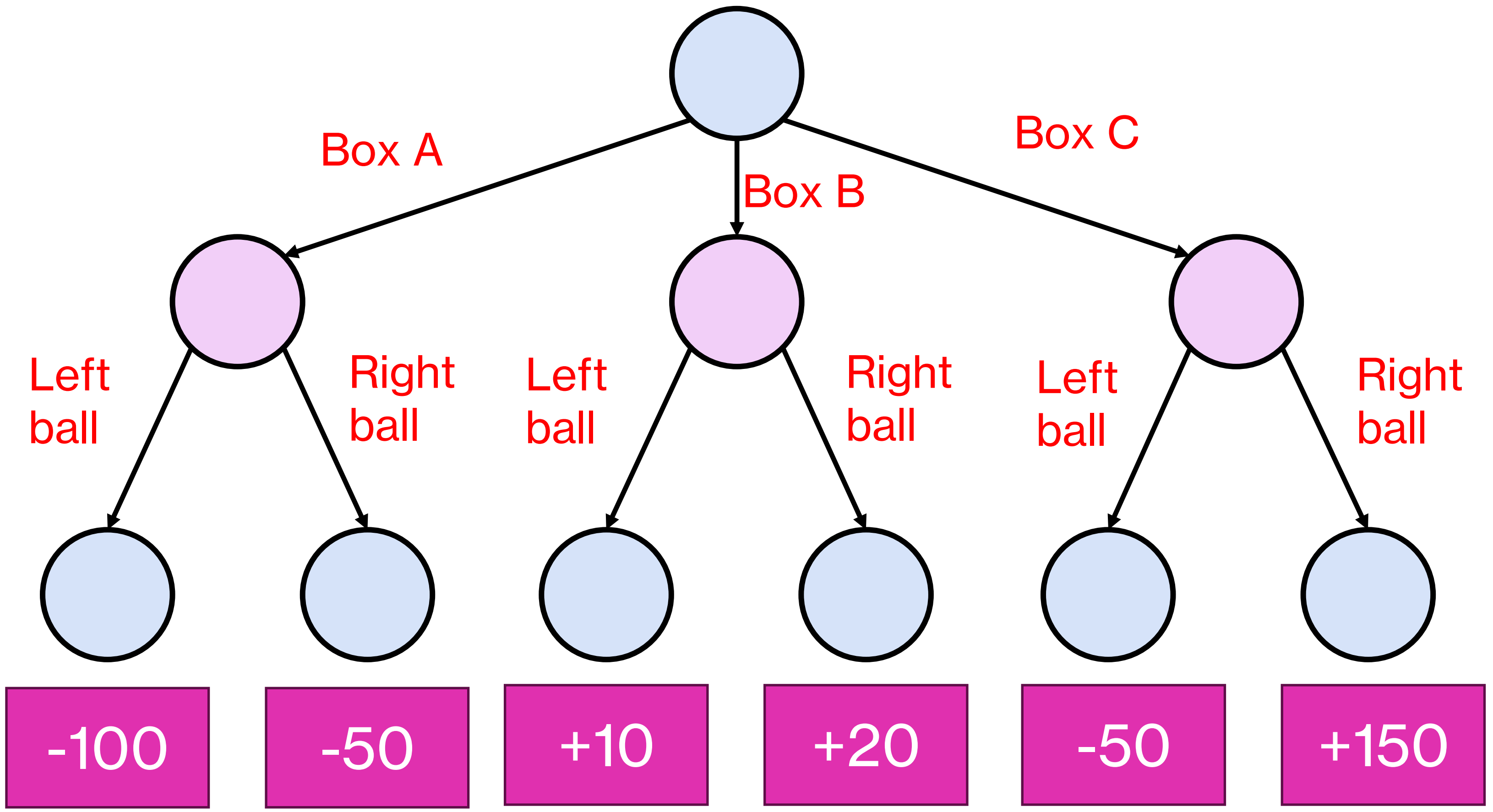
- **Goal:** pick the best move for the agent assuming a stochastic opponent policy.
  - Each state is assigned a “value”, representing how good that state is for the agent.
  - The value is computed recursively (dynamic programming) from the bottom of the tree upwards.

Expectimax Formulation

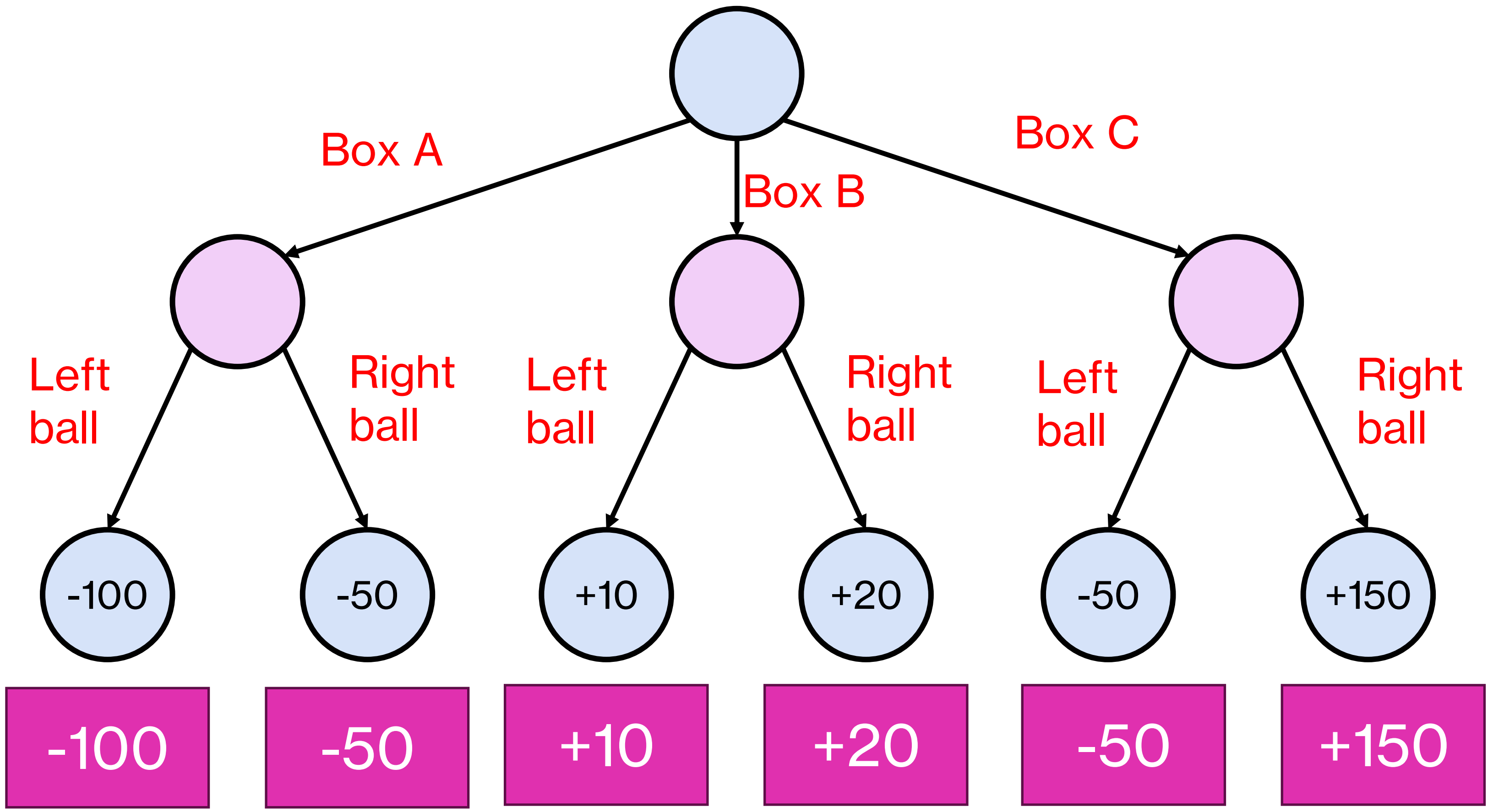
- $V_{eval}(s)$: the expected value of state s.

- $V_{eval}(s) = \begin{cases} Utility(s), & IsEnd(s) \\ \max\left(V_{eval}(Succ(s,a))\right), & Player(s) = agent \\ \sum_{a \in Actions(s)} \pi_{opp}(s,a) V_{eval}(Succ(s,a)), & Player(s) = opp \end{cases}$

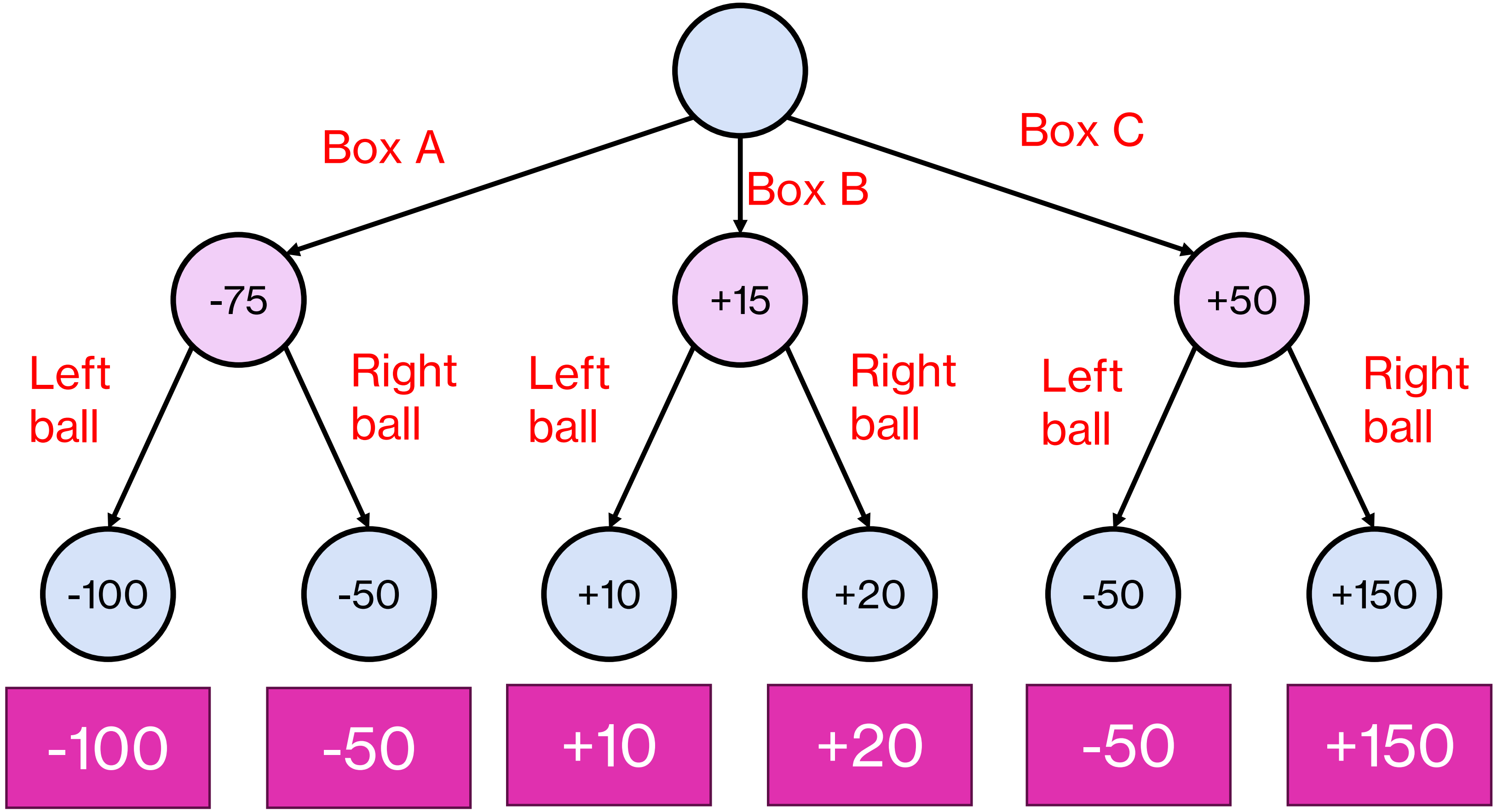
Expectimax: Value Evaluation



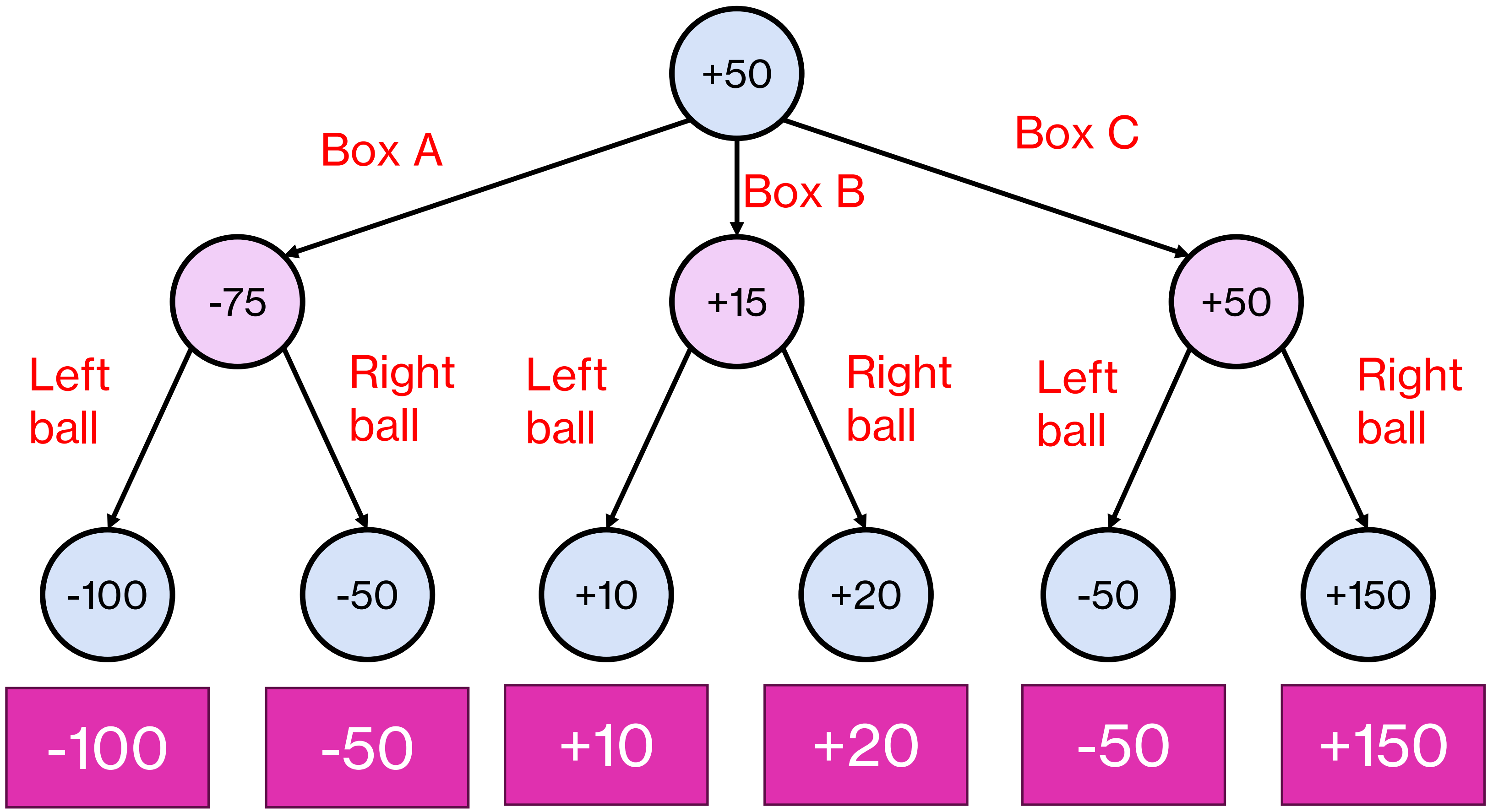
Expectimax: Value Evaluation



Expectimax: Value Evaluation



Expectimax: Value Evaluation



Case 3: Optimally Smart Opponent

- **Opponent Policy:** the opponent always picks the move that will minimize the agent’s expected winnings.

Minimax

- **Goal:** pick the best move for the agent assuming an optimally smart opponent.
  - Each state is assigned a “value”, representing how good that state is for the agent.
  - The value is computed recursively (dynamic programming) from the bottom of the tree upwards.

Minimax Formulation

- $V_{eval}(s)$: the expected value of state s.
- $$V_{eval}(s) = \begin{cases} Utility(s), & IsEnd(s) \\ \max\left(V_{eval}\left(Succ(s,a)\right)\right), & Player(s) = agent \\ \min\left(V_{eval}\left(Succ(s,a)\right)\right), & Player(s) = opp \end{cases}$$

Characteristics of Minimax

- If the game tree is finite, it is **complete**.
- It is **optimal**, assuming the opponent also behaves optimally.
- Time complexity: $\boldsymbol{O\left(b^d\right)}$ (depth-first search)
- Space complexity: $\boldsymbol{O(bD)}$ (depth-first search)

Characteristics of Minimax

- If the game tree is finite, it is **complete**.
- It is **optimal**, assuming the opponent also behaves optimally.
- Time complexity: $\boldsymbol{O(b^d)}$ (depth-first search)
- Space complexity: $\boldsymbol{O(bD)}$ (depth-first search)

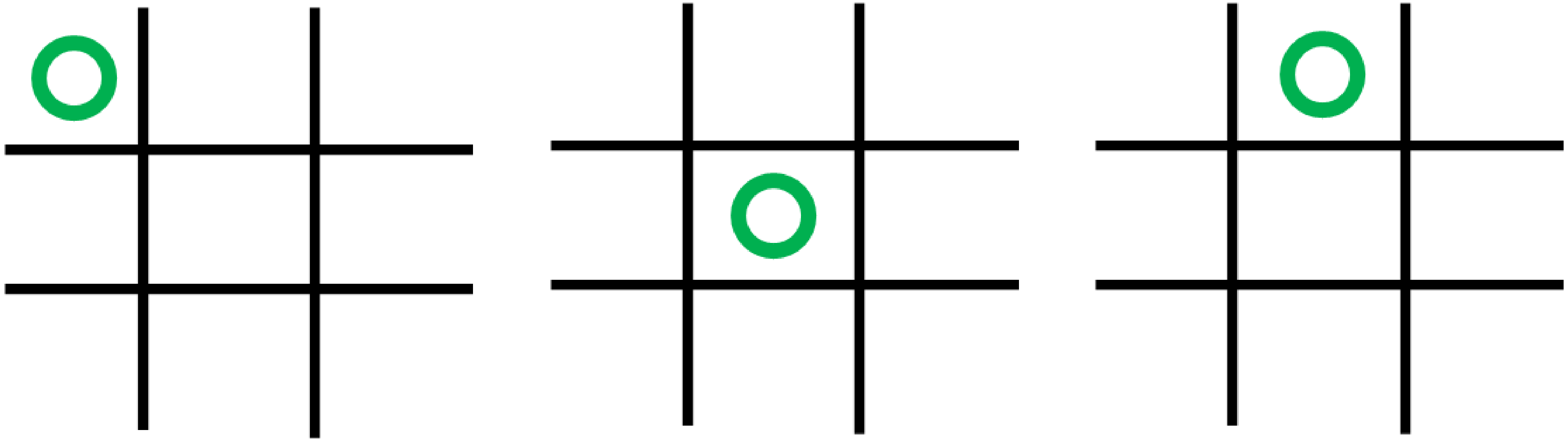
- For many games, b is too high. Therefore, Minimax is infeasible!
  - Checkers: $b \approx 6.4$
  - Chess: $b \approx 35$
  - Go: $b \approx 250$

Speeding Up Techniques

- Use domain knowledge to **prune redundant states**
- Depth-limited search with **evaluation functions**
- **$\alpha\beta$ (alpha-beta) pruning**

Prune Redundant States

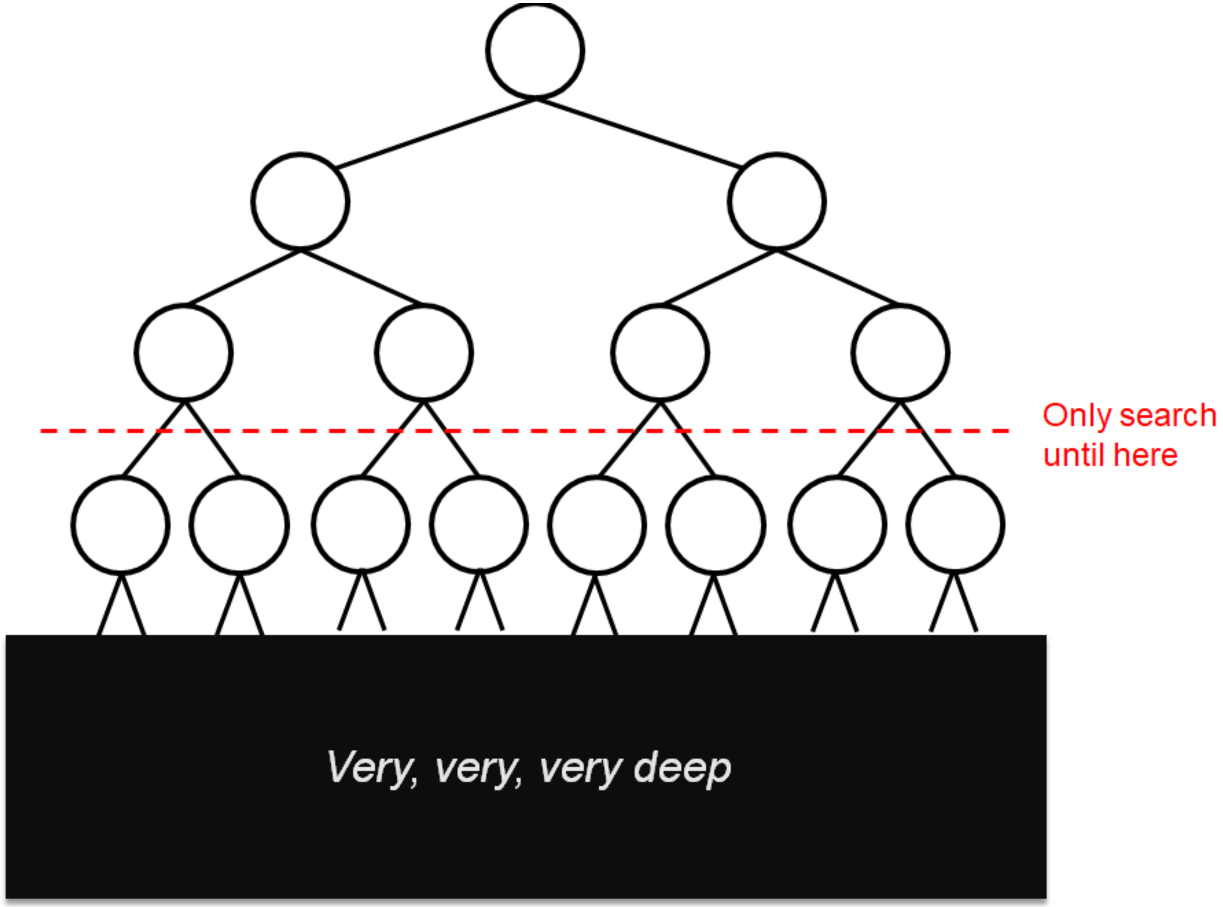
- **Key idea:** Some states may be redundant based on the rules of the game – don’t explore them more than once!



It may seem that Tic-Tac-Toe has 9 possible initial moves, but these are really the only three that matter!

Depth-Limited Search with Evaluation Functions

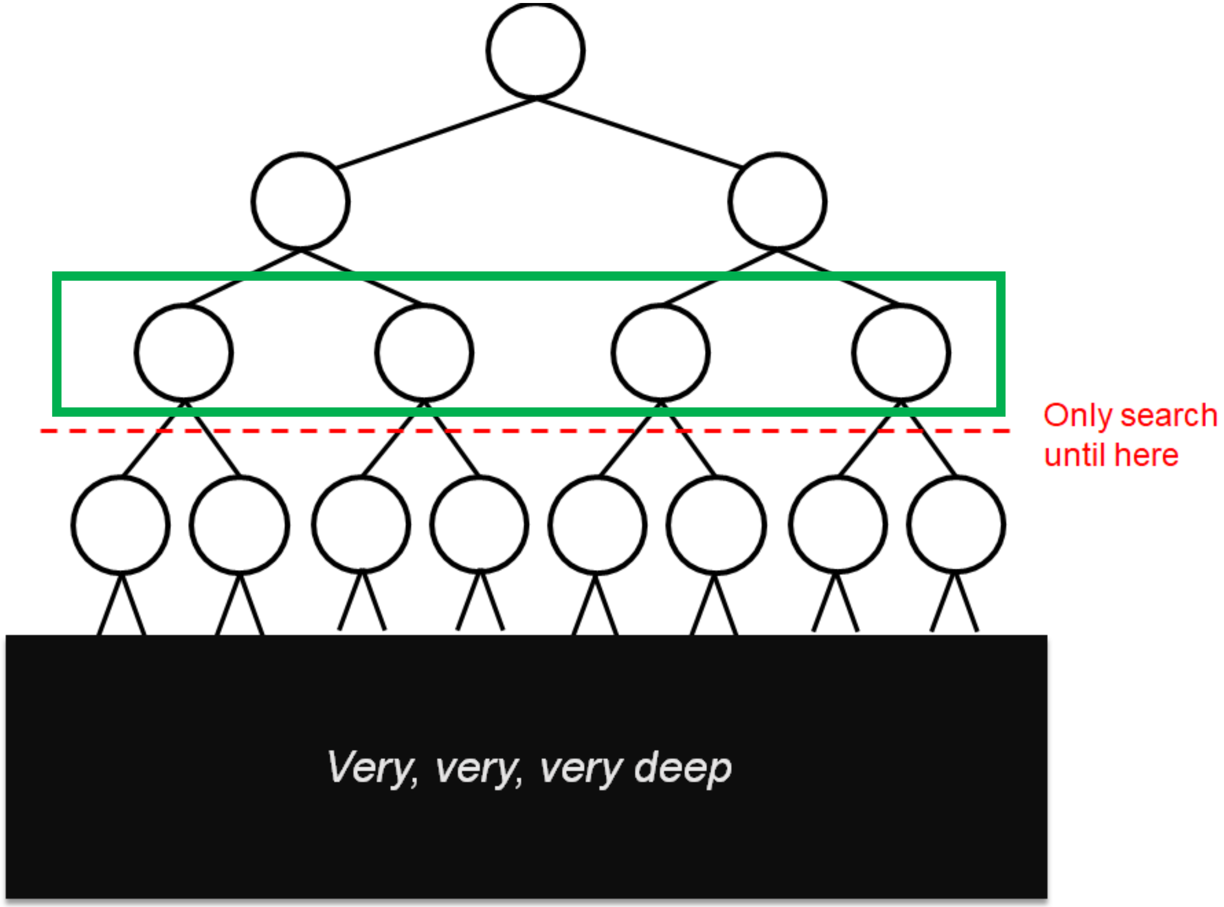
- **Key idea:** Since the game tree is too deep, set a limit to the depth to be explored!



Depth-Limited Search with Evaluation Functions

- **Key idea:** Since the game tree is too deep, set a limit to the depth to be explored!

- **But how we do the expected value of these states?**



Evaluation Function

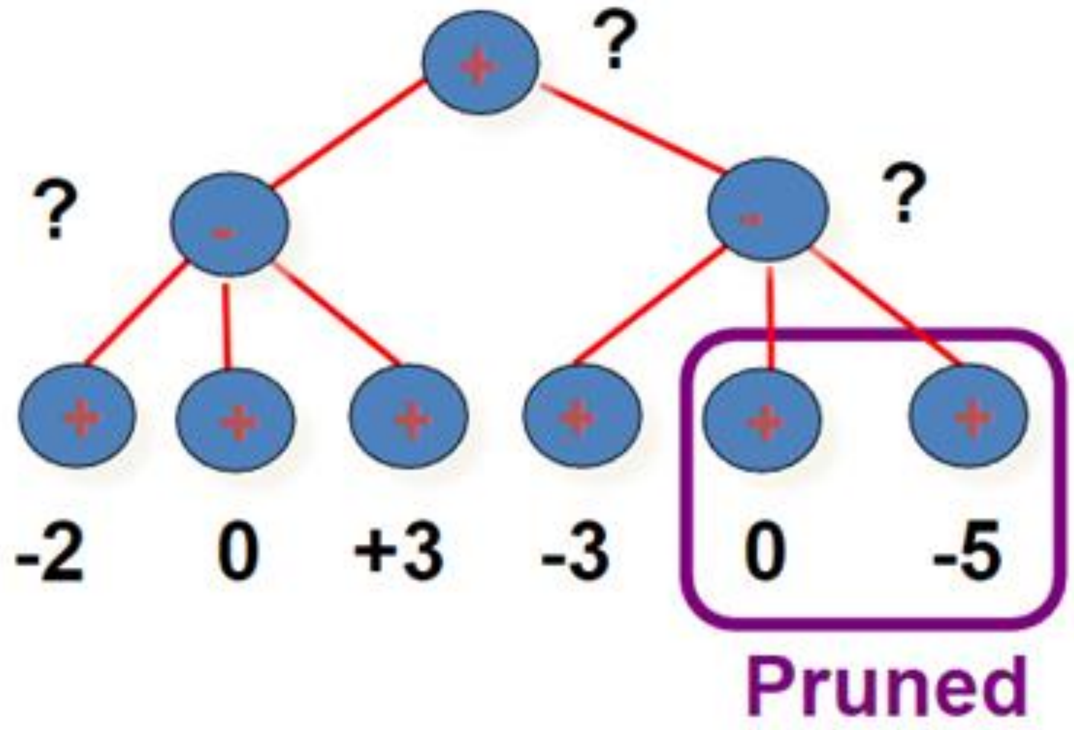
- An estimation of how "good" the state is for the agent.
- Based on the domain knowledge about the game.
- Example: in chess, the total value of the remaining pieces

Pieces and Point Value		
Pawn	♟	1
Knight	♞	3
Bishop	♝	3
Rook	♜	5
Queen	♛	9
King	♚	priceless

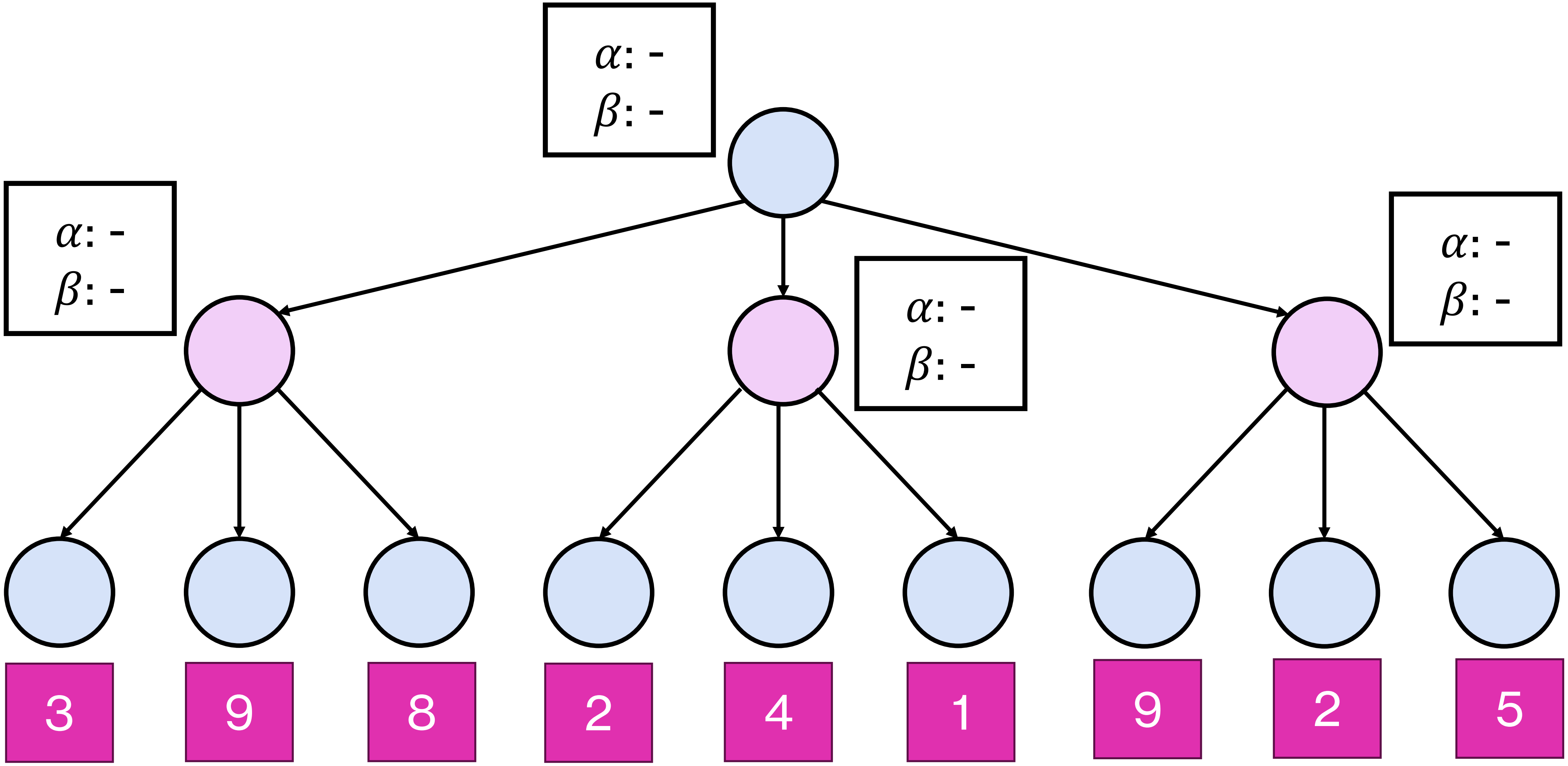
$\alpha\beta$-Pruning

- **Key Idea:** Don't explore a branch if we're sure that a better path already exists!

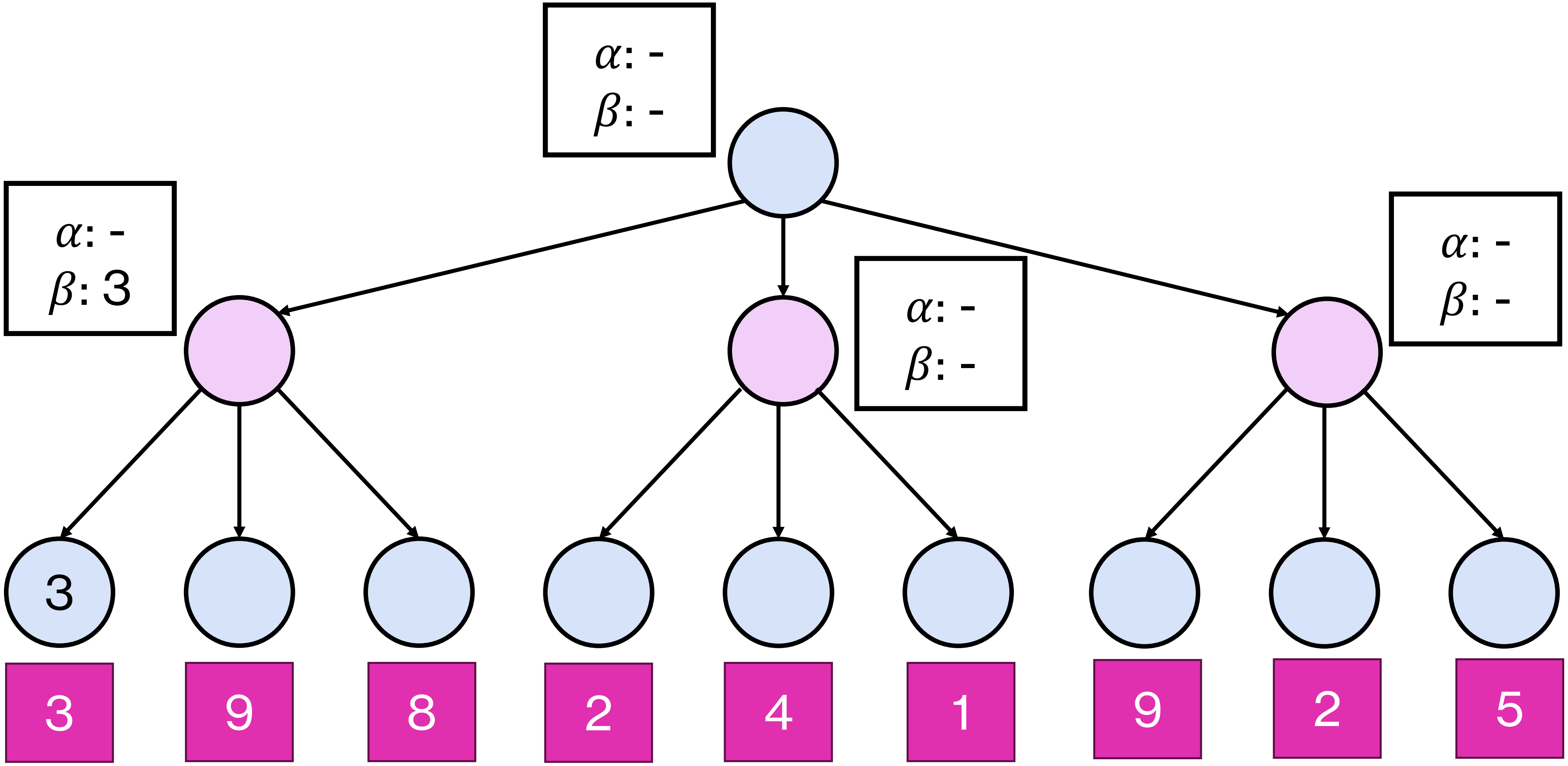
- **Goal:** Search space may become significantly smaller



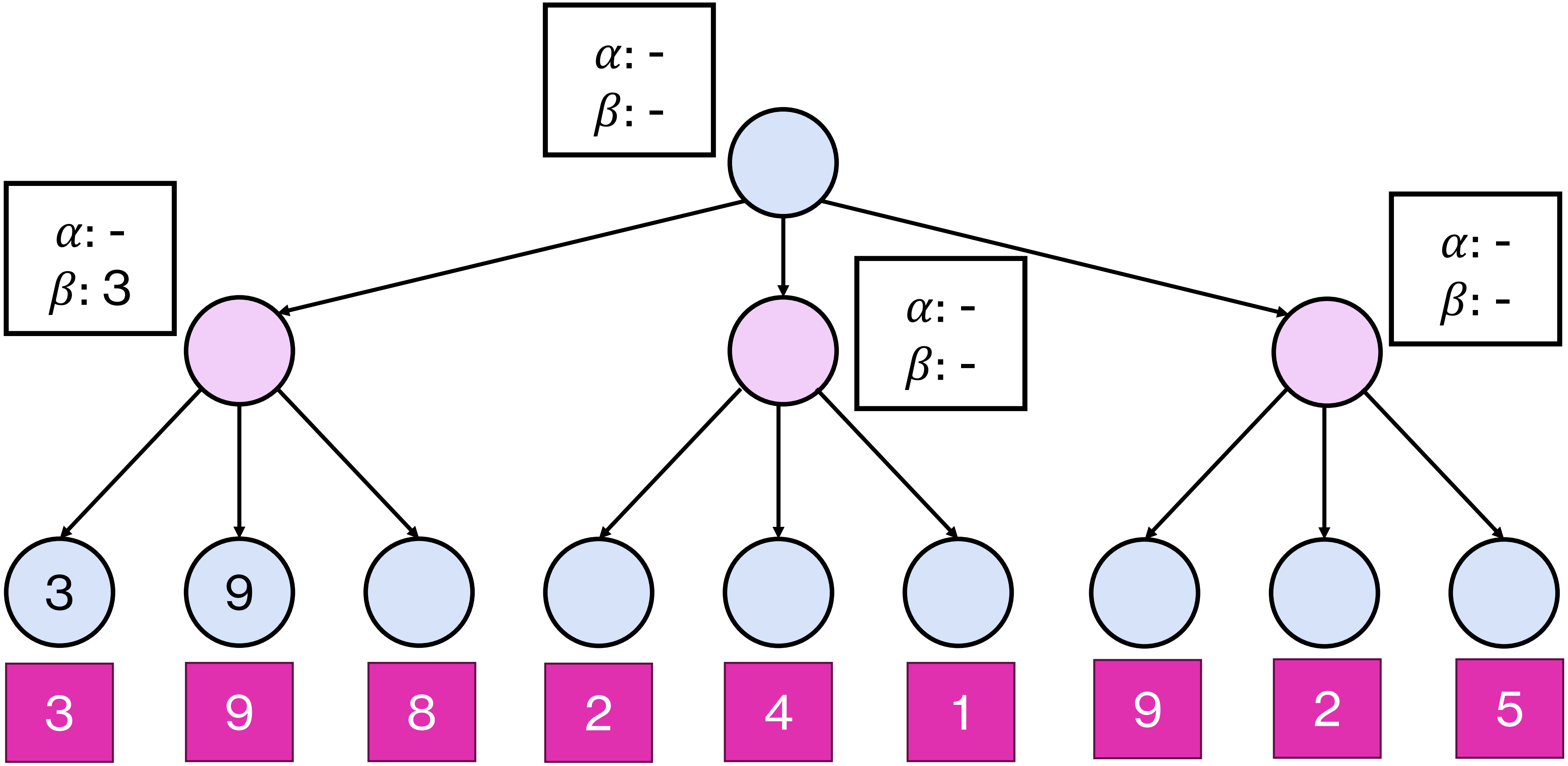
$\alpha\beta$-Pruning Example



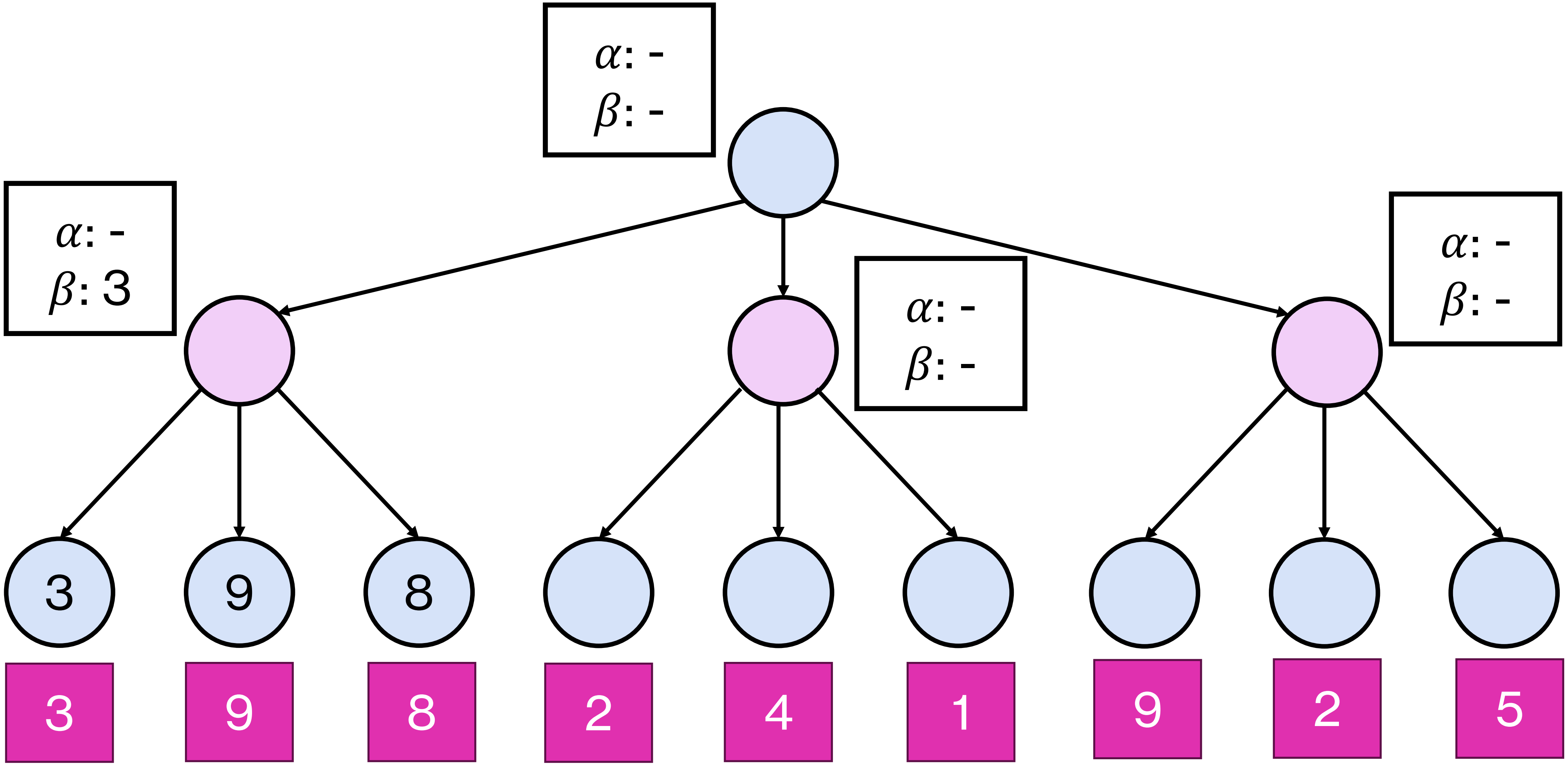
$\alpha\beta$-Pruning Example



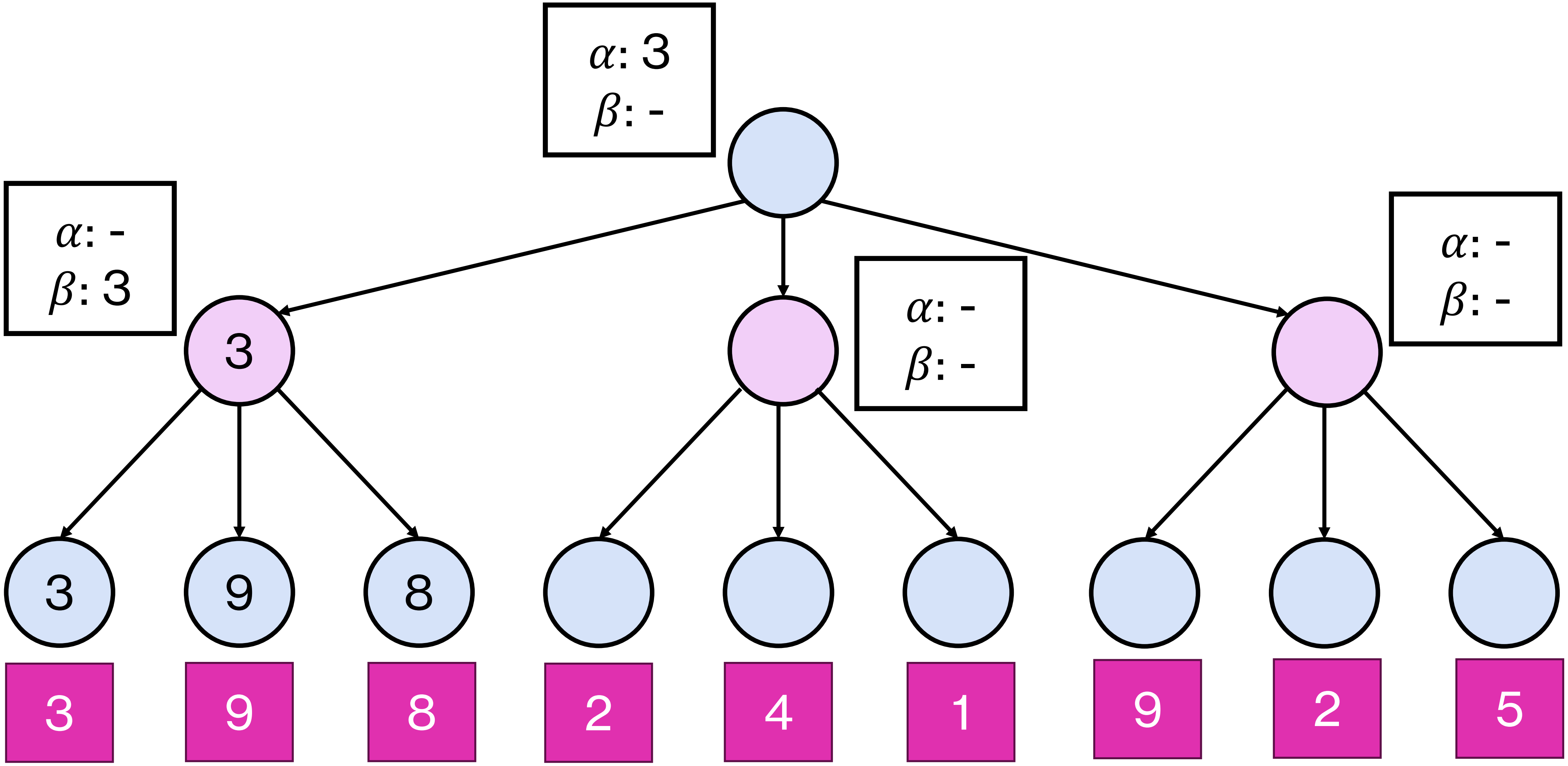
$\alpha\beta$-Pruning Example



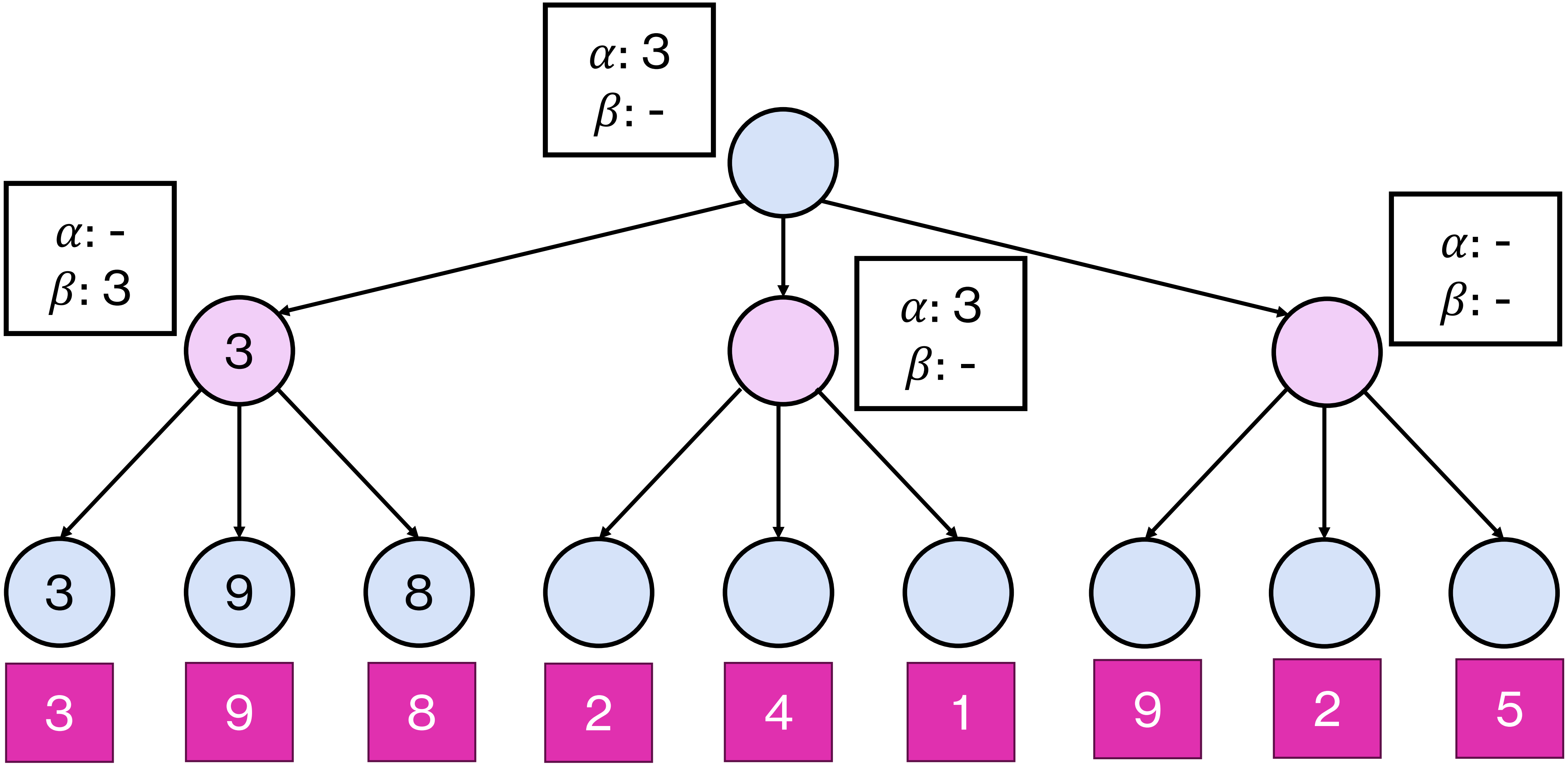
$\alpha\beta$-Pruning Example



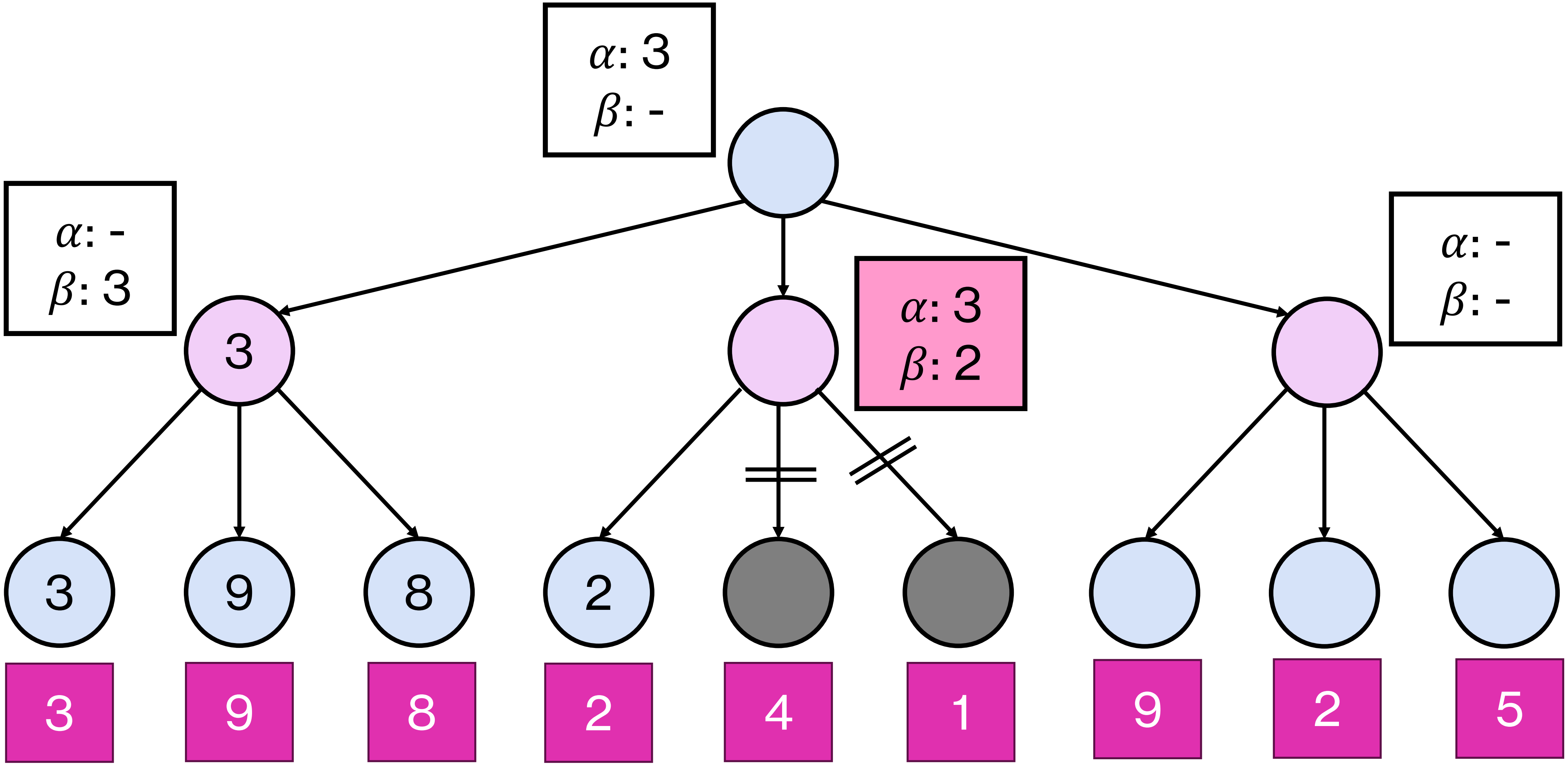
$\alpha\beta$-Pruning Example



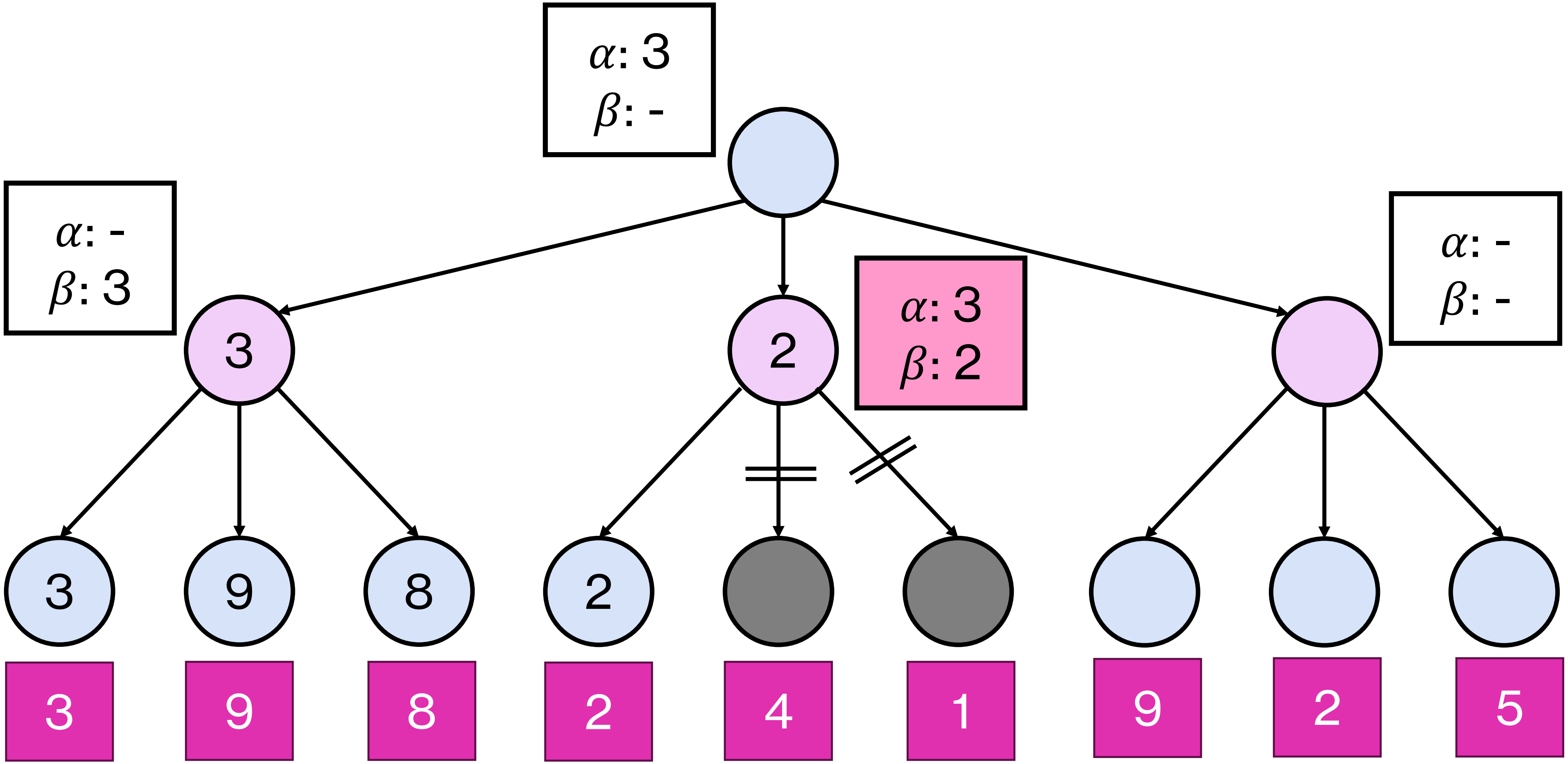
$\alpha\beta$-Pruning Example



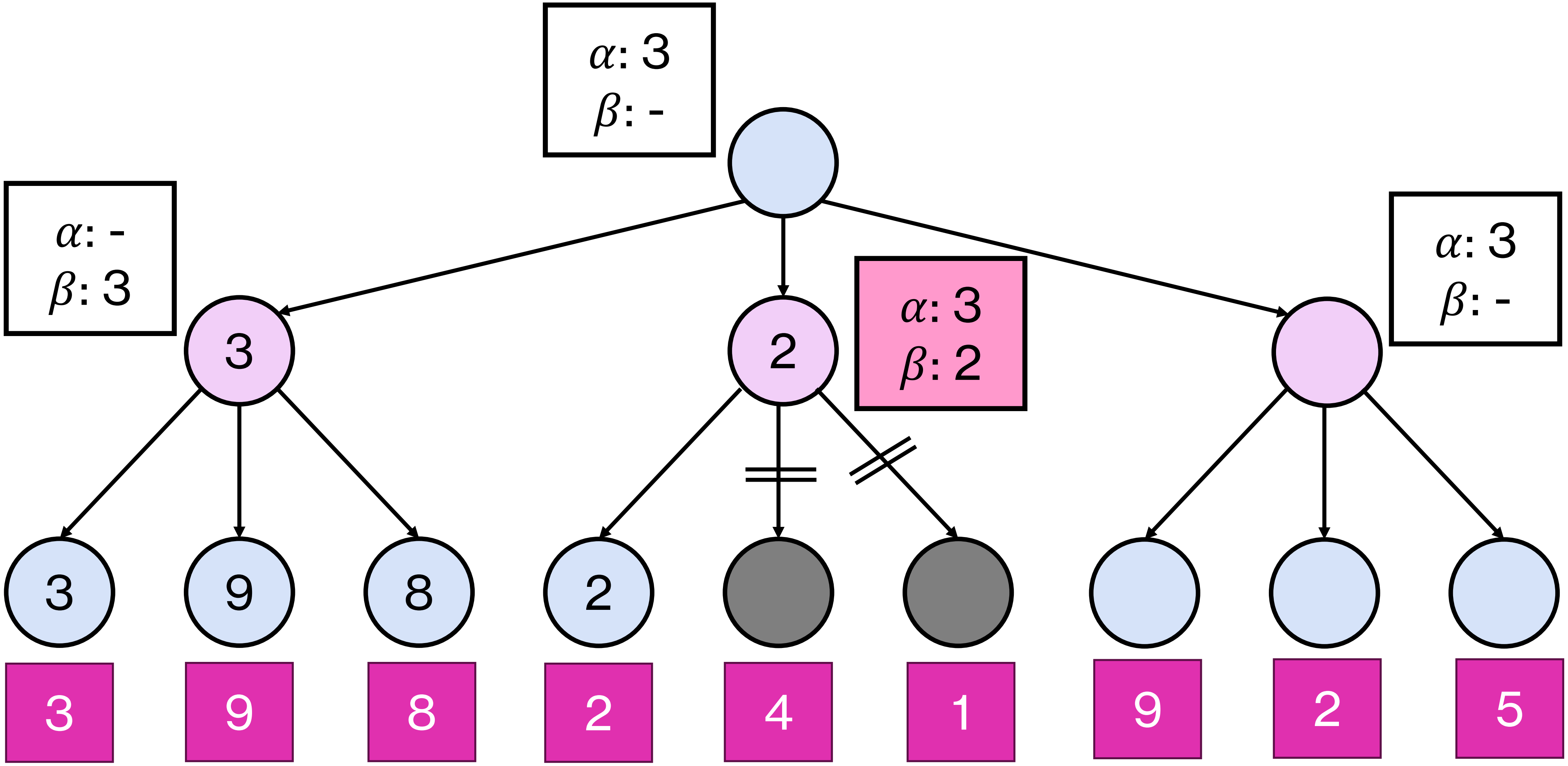
$\alpha\beta$-Pruning Example



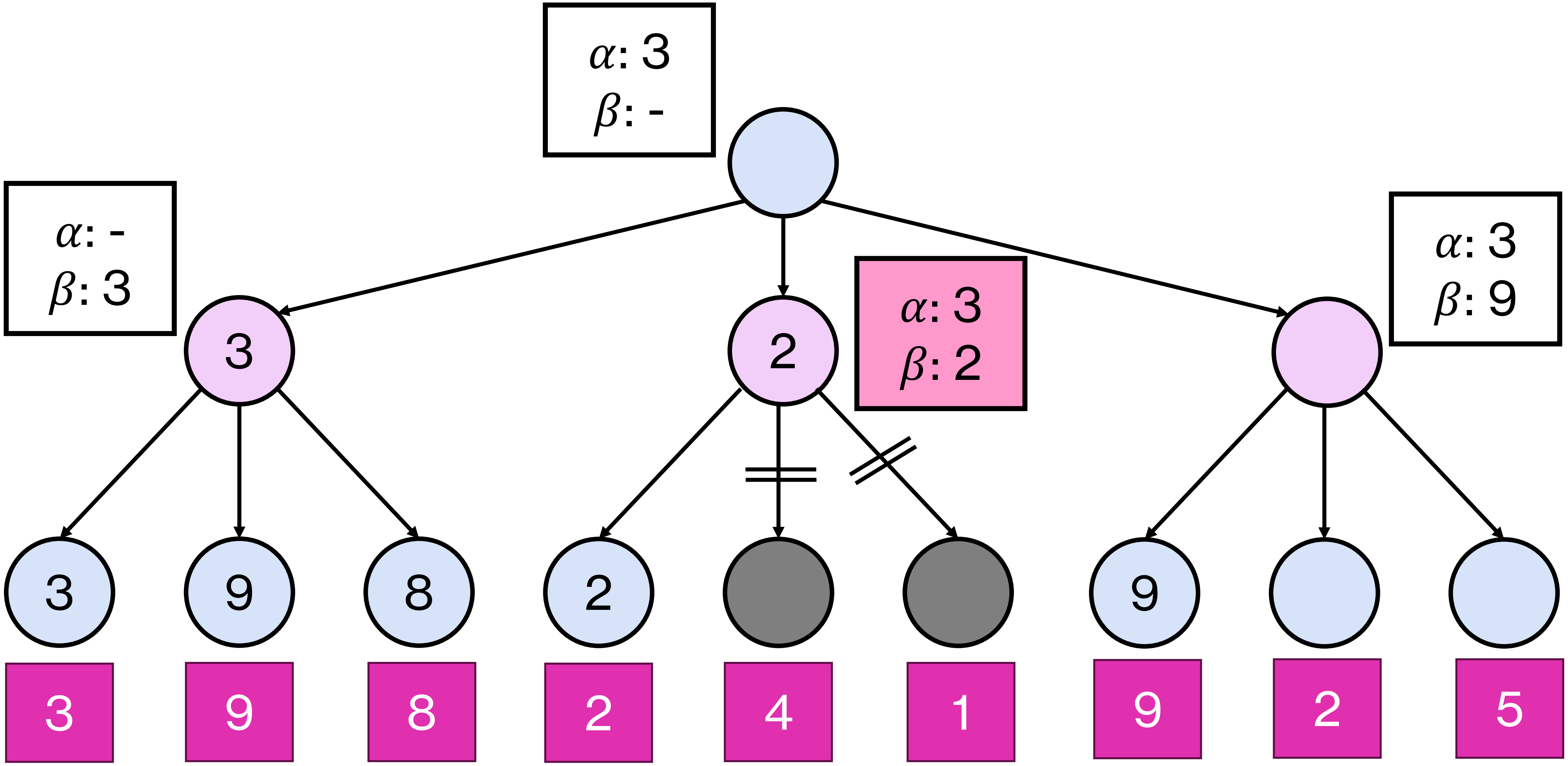
$\alpha\beta$-Pruning Example



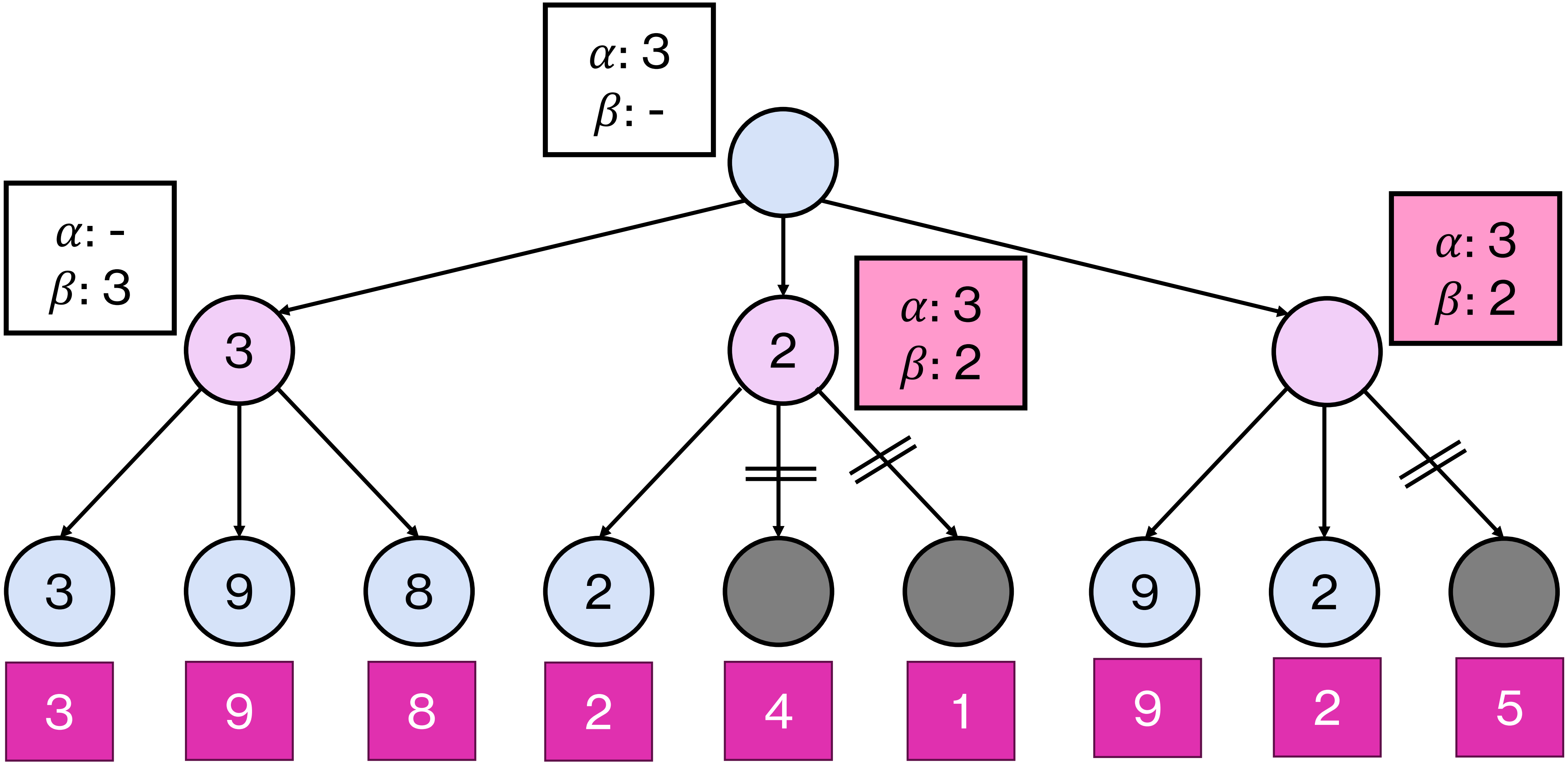
$\alpha\beta$-Pruning Example



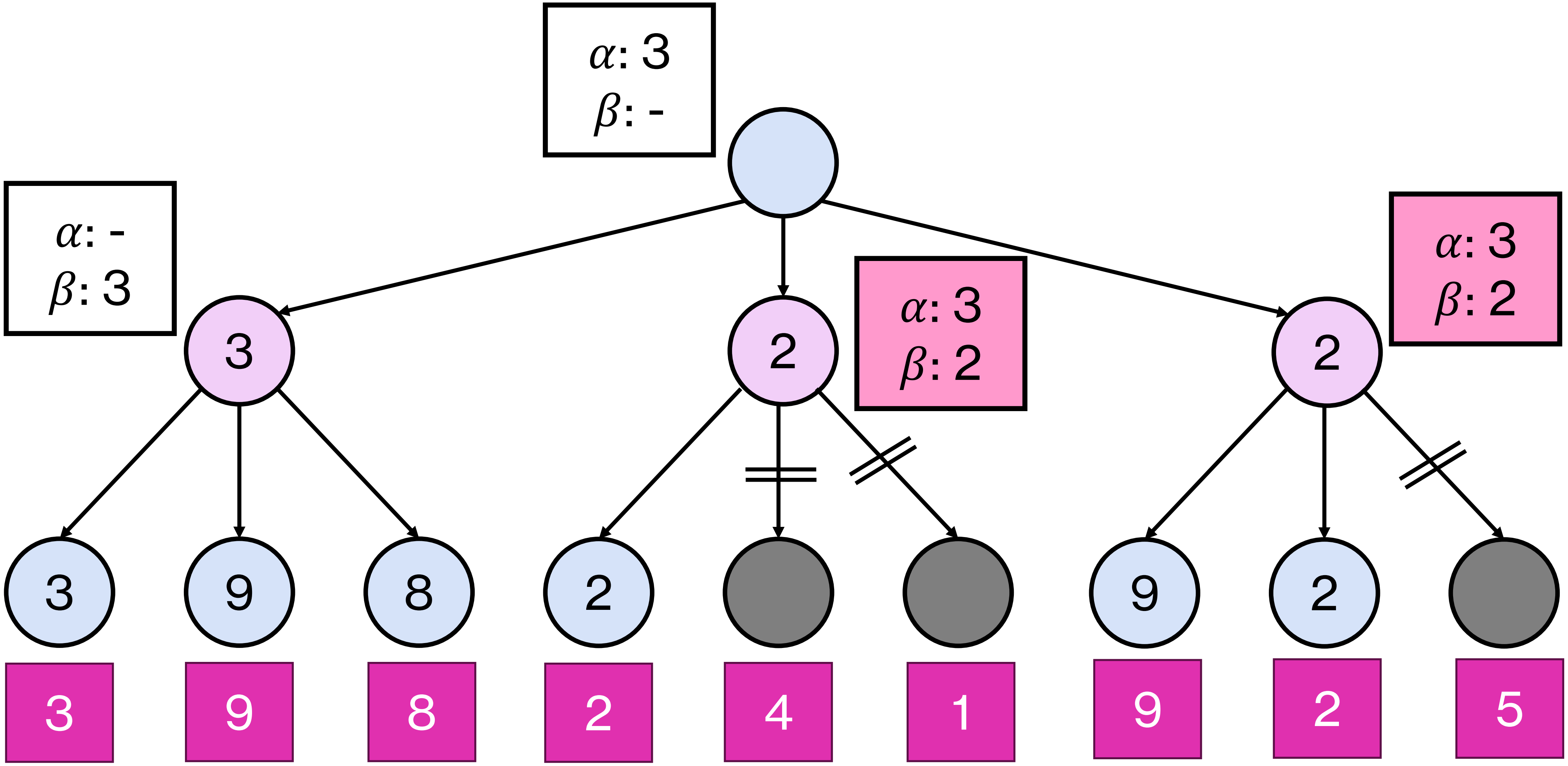
$\alpha\beta$-Pruning Example



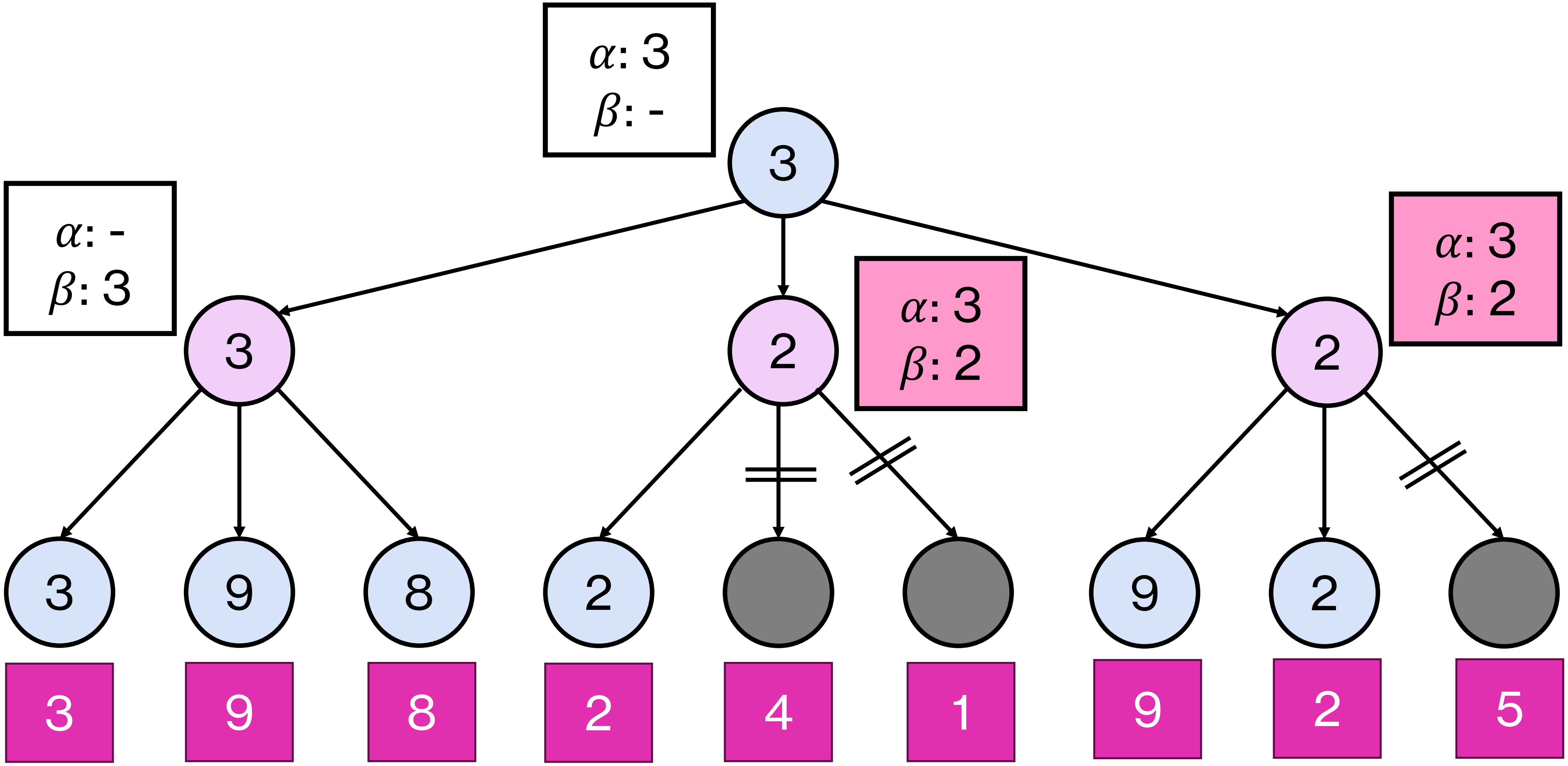
$\alpha\beta$-Pruning Example



$\alpha\beta$-Pruning Example



$\alpha\beta$-Pruning Example



Characteristics of $\alpha\beta$-Pruning

- Pruning does not affect the result of Minimax.
- Entire subtrees can be pruned.
- Good move ordering influences the performance gains of pruning.
- Ordering moves from “best” to “worst” generally results to better pruning.

Acknowledgments

- Stanford University CS221 Autumn 2021 course. Available online at: https://stanford-cs221.github.io/autumn2021
- Previous CSINTSY slides by the following instructors:
  - Raymund Sison, PhD
  - Judith Azcarraga, PhD
  - Merlin Suarez, PhD
  - Joanna Pauline Rivera