

Monte Carlo $\alpha - \beta$ (4.8.7)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Seminar aus Maschinellem Lernen

MC $\alpha \beta$

1. $\alpha\beta$ Algorithmus

2. MCTS + $\alpha\beta$

- MCTS als Evaluierungsfunktion für $\alpha\beta$
- $\alpha\beta$ als Default Policy für MCTS

3. MC-LOA _{$\alpha\beta$} als Anwendung in Lines of Action

Vorstellung des $\alpha\beta$ -Algorithmus

- $\alpha\beta$ = Modifizierter **MiniMax** Ansatz
- Intervall in jedem Knoten wird verwendet um Teilbäume abzuschneiden
 - => weniger Knoten müssen besucht werden
 - Ergebnis wird **nicht** verändert

$$[\alpha, \beta]$$

- α = Bisher bester Wert für **Max-Spieler**
- β = Bisher bester Wert für **Min-Spieler**

Vorstellung des $\alpha\beta$ -Algorithmus

- **Initialisierung:**

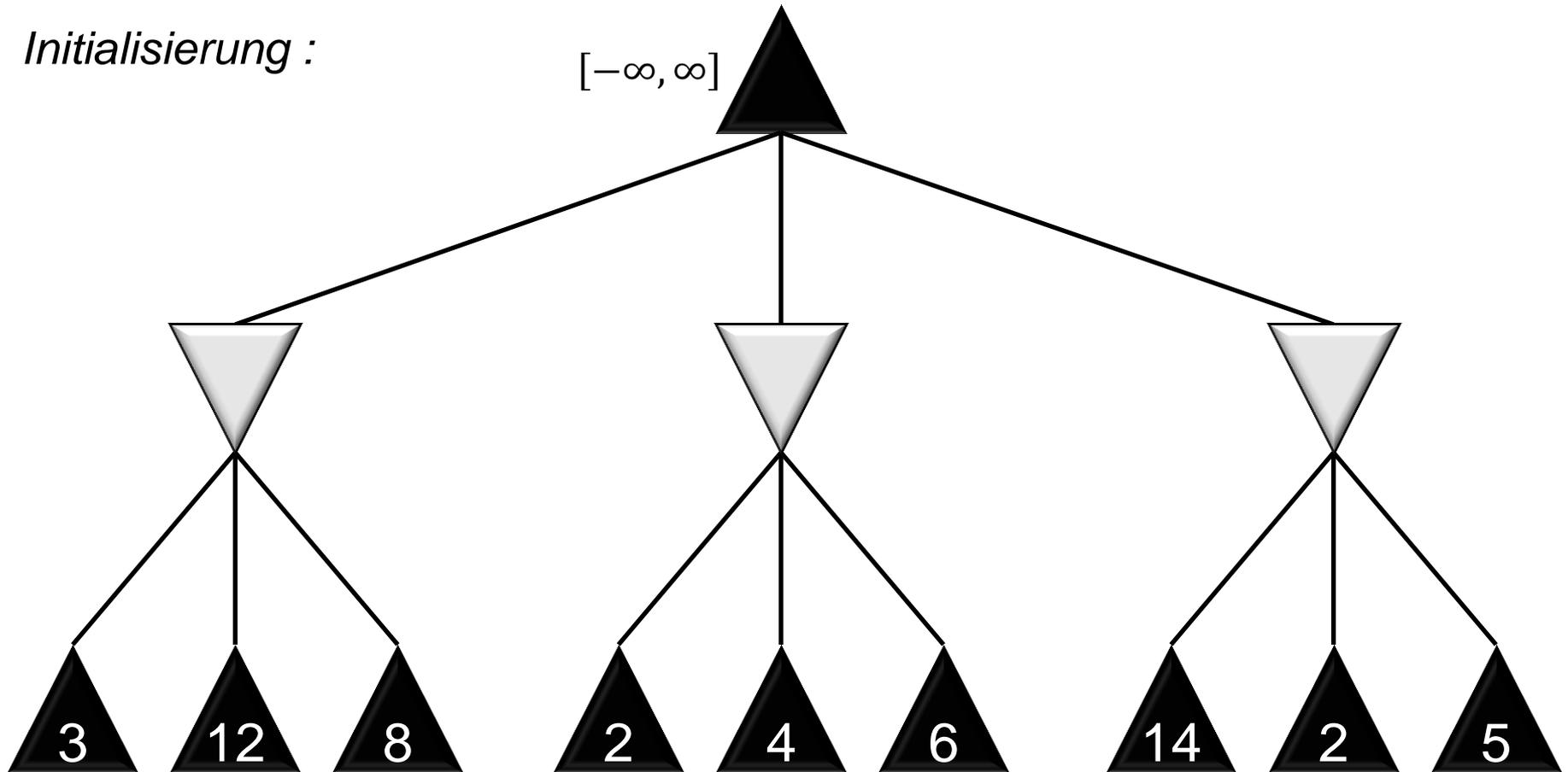
$$[\alpha, \beta] = [-\infty, \infty]$$

- **Suche:**

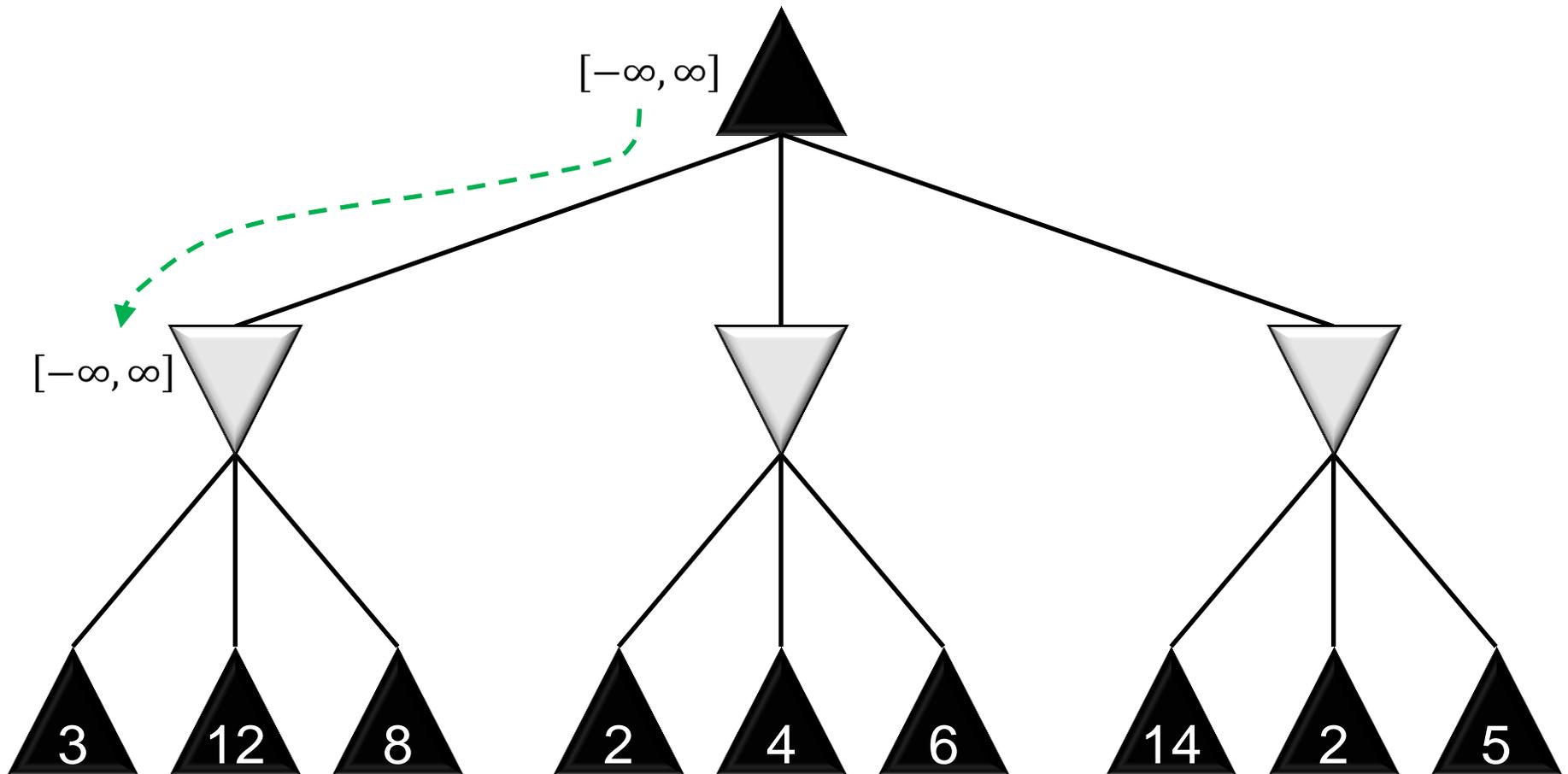
- $[\alpha, \beta]$ wird in rekursivem Aufruf übergeben
- Wenn während Suche besserer Wert für **Max** / **Min** gefunden wird, wird Wert in α / β eingetragen
- **MIN** Knoten: Cutoff wenn $value \leq \alpha$ gefunden (**Alpha Cutoff**)
- **MAX** Knoten: Cutoff wenn $value \geq \beta$ gefunden (**Beta Cutoff**)

$\alpha\beta$ -Beispiel

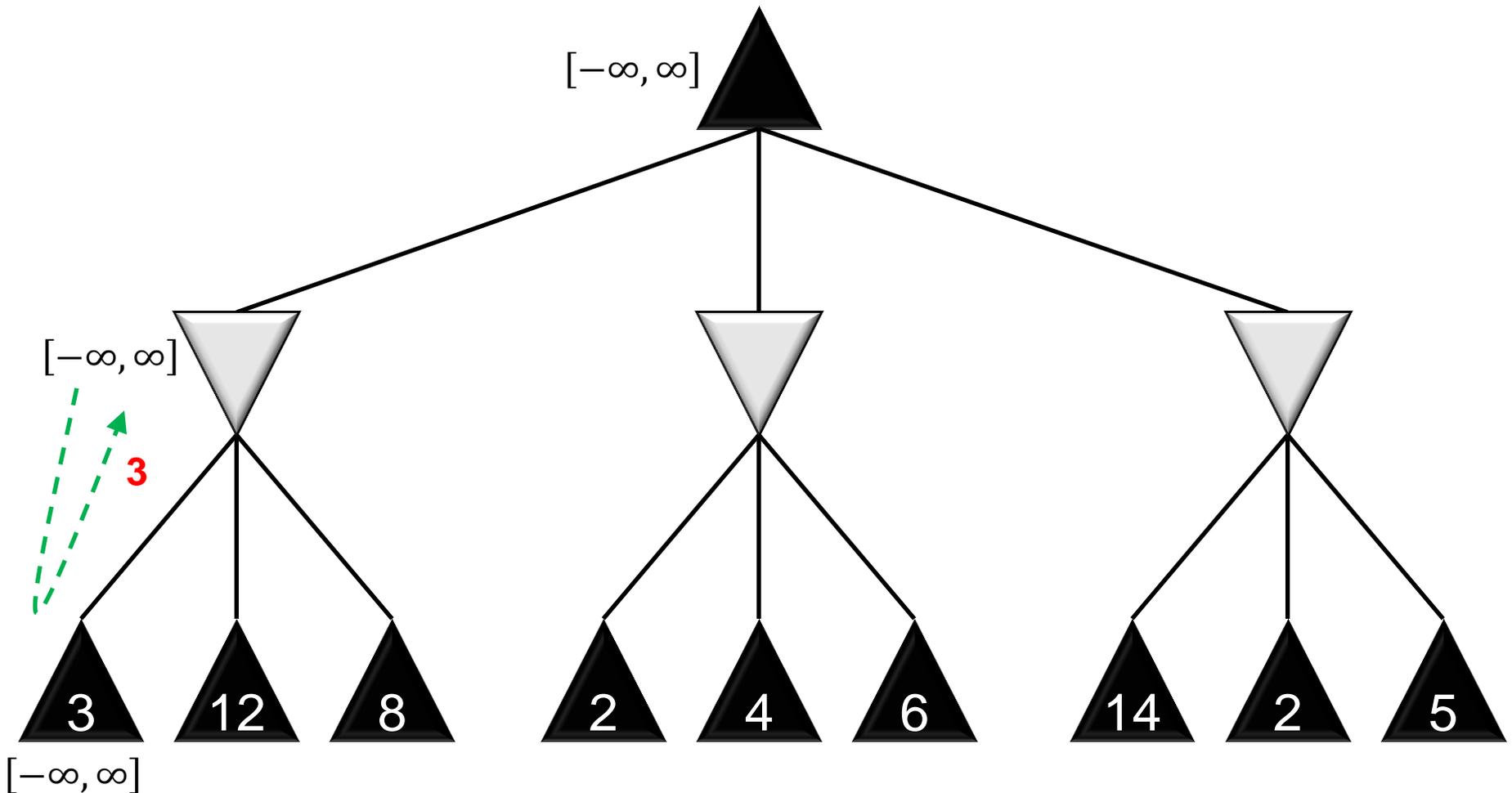
Initialisierung :



$\alpha\beta$ -Beispiel

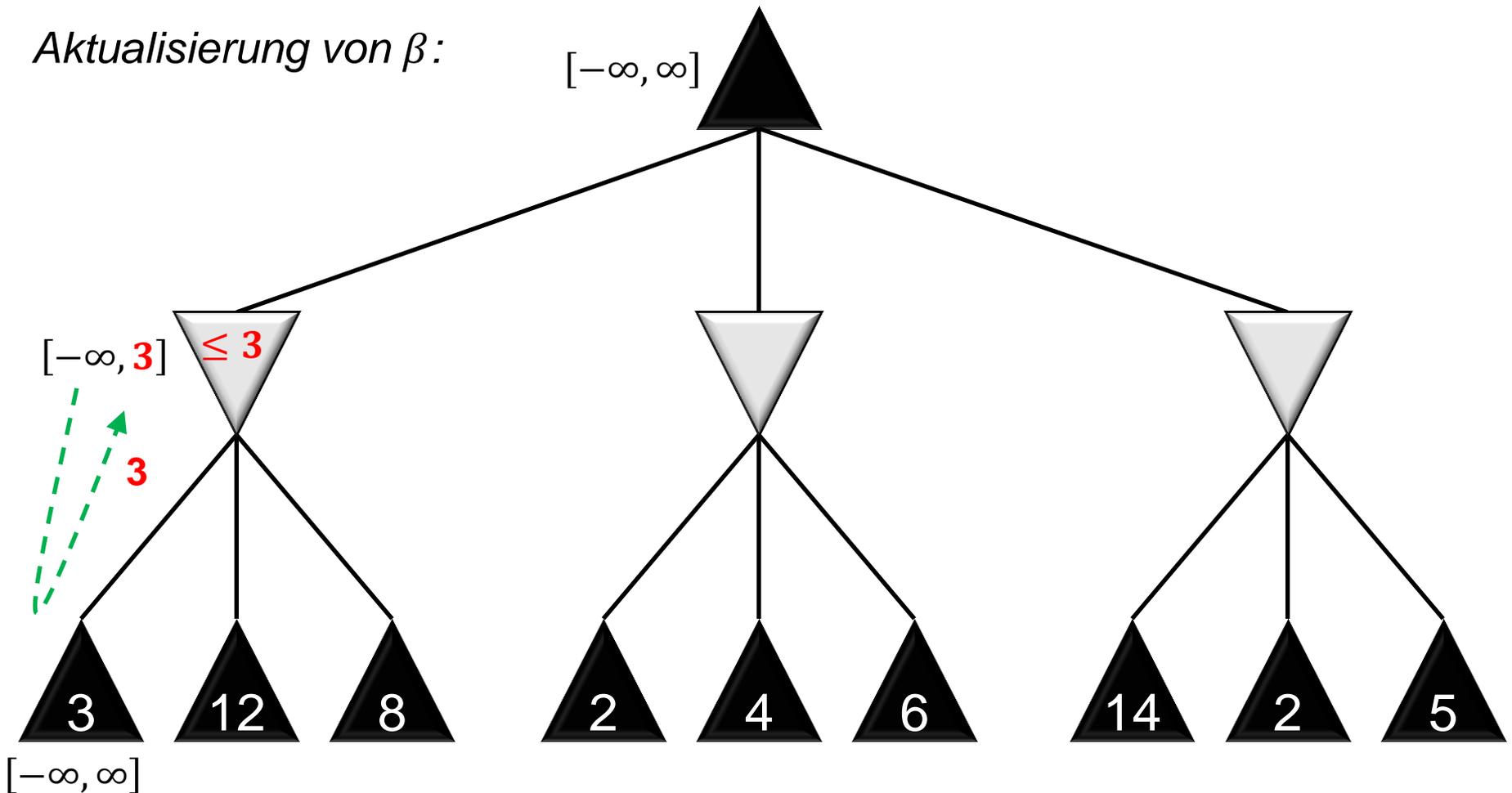


$\alpha\beta$ -Beispiel

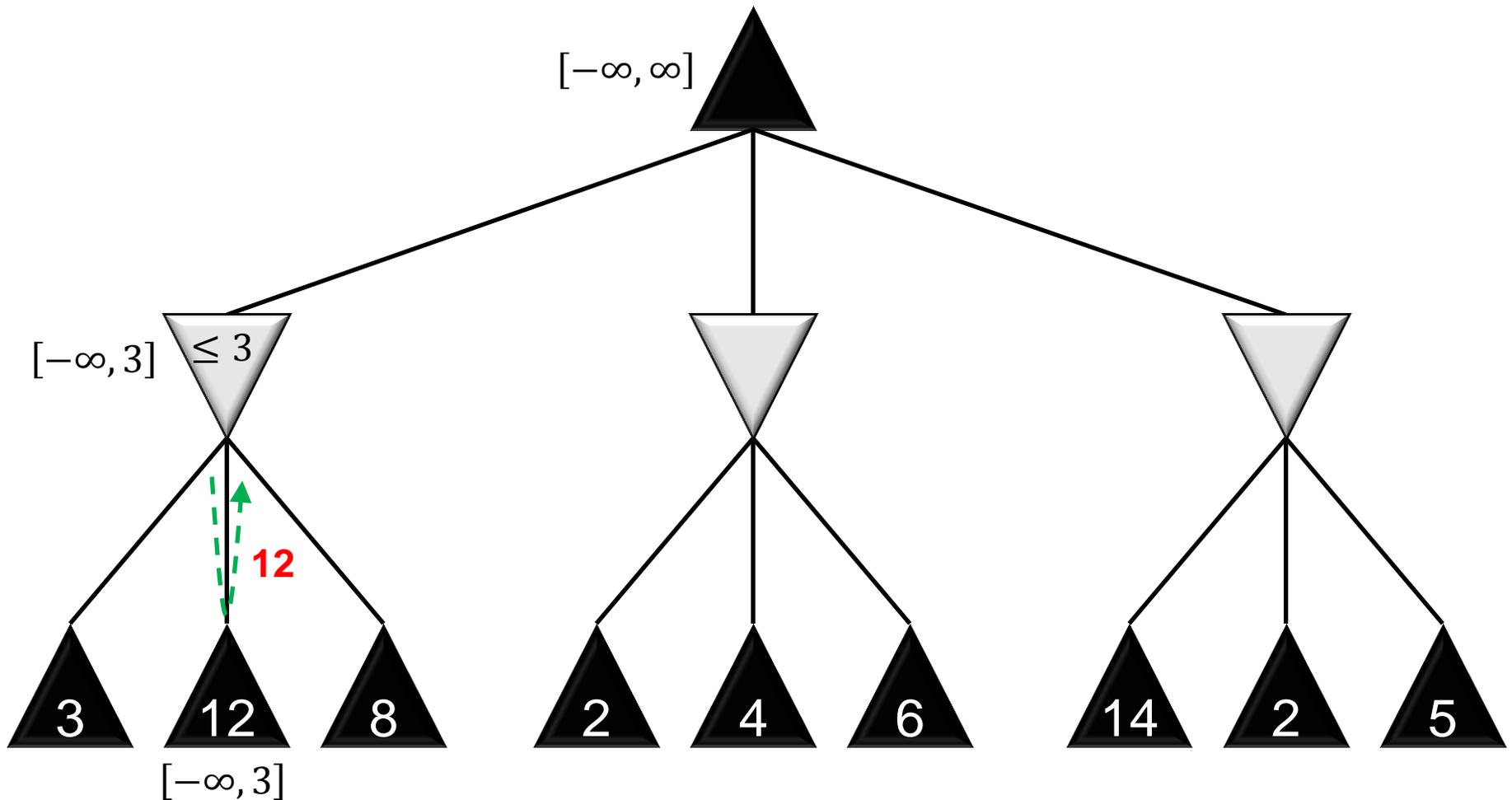


$\alpha\beta$ -Beispiel

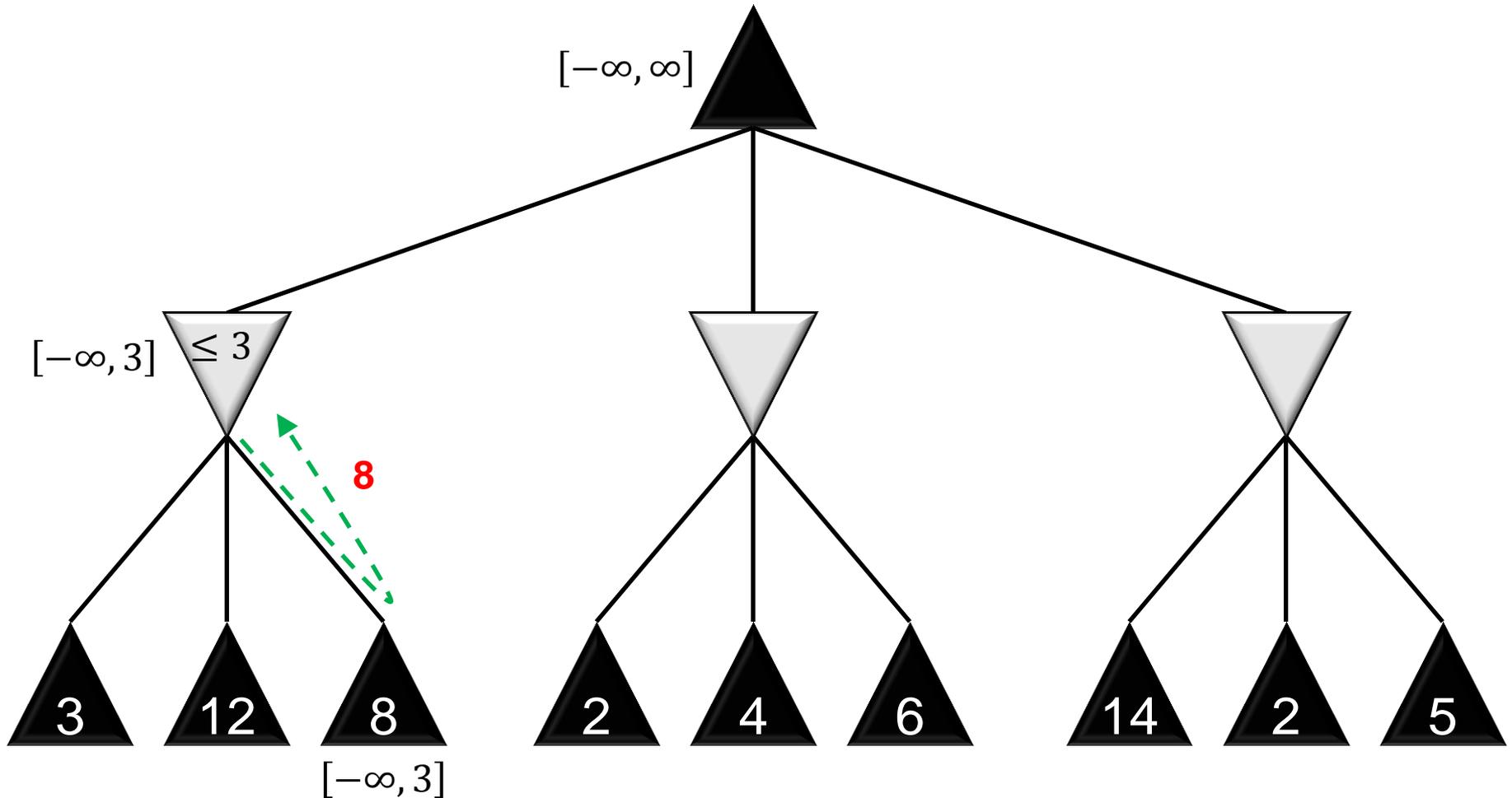
Aktualisierung von β :



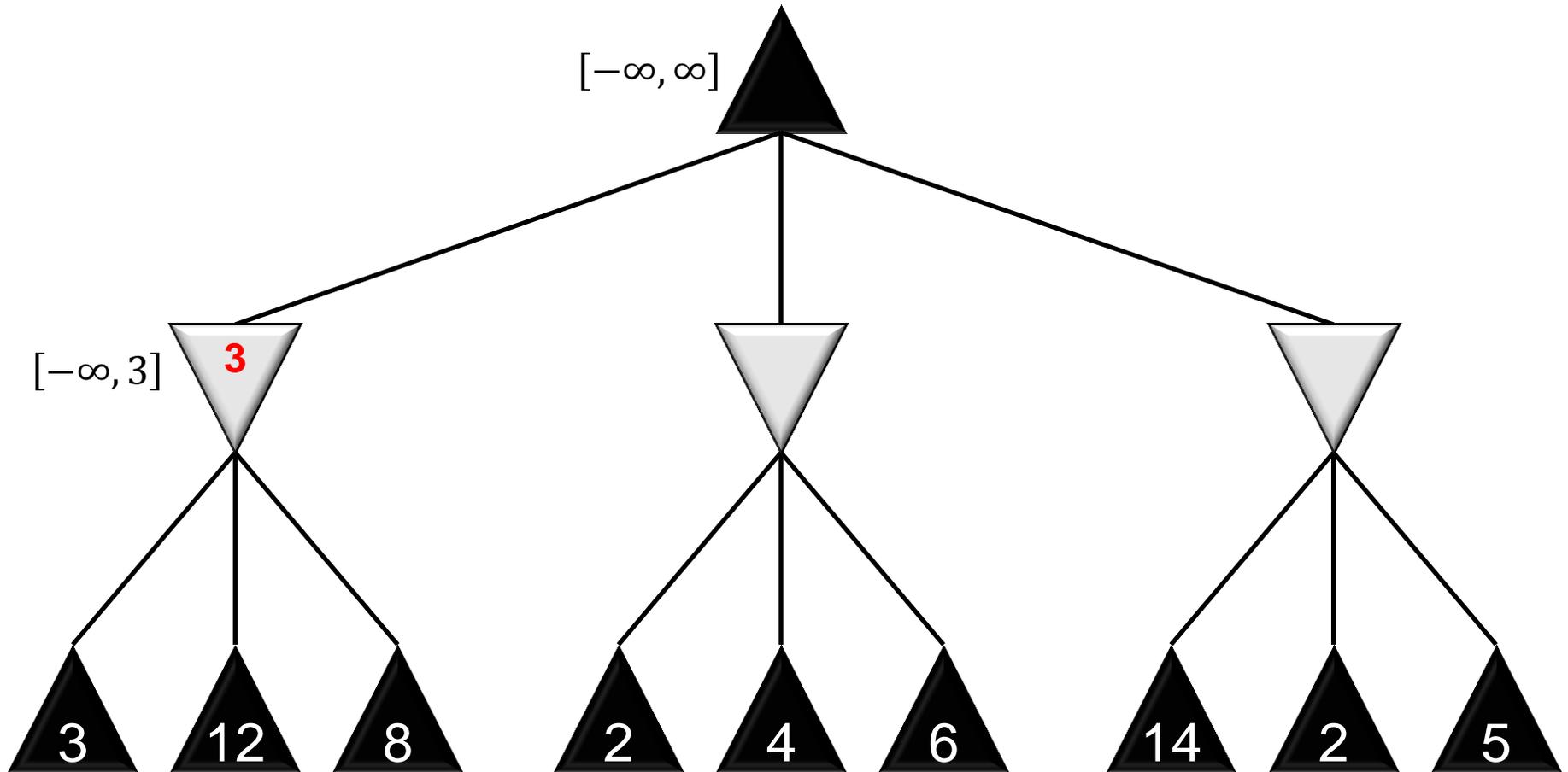
$\alpha\beta$ -Beispiel



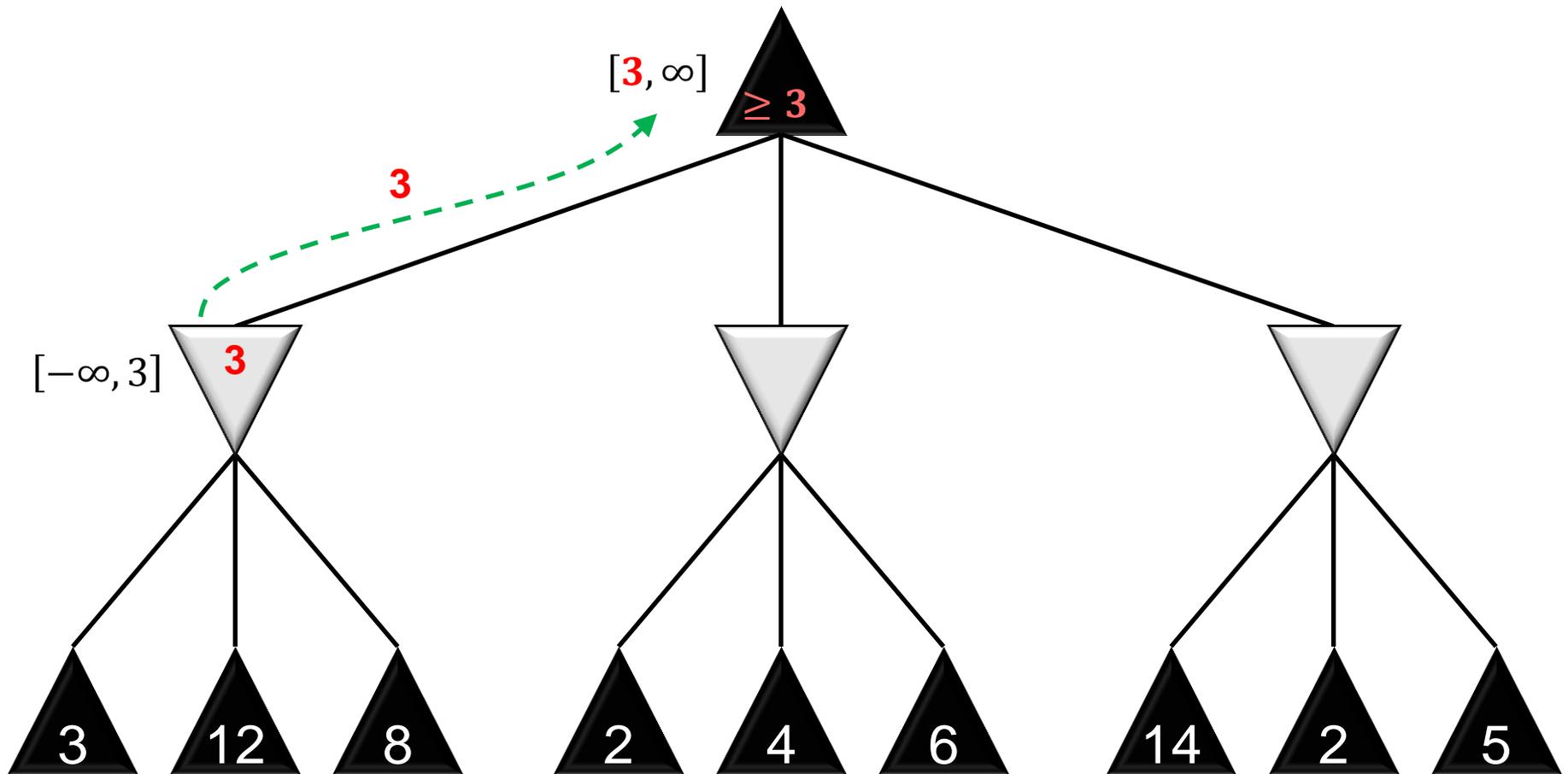
$\alpha\beta$ -Beispiel



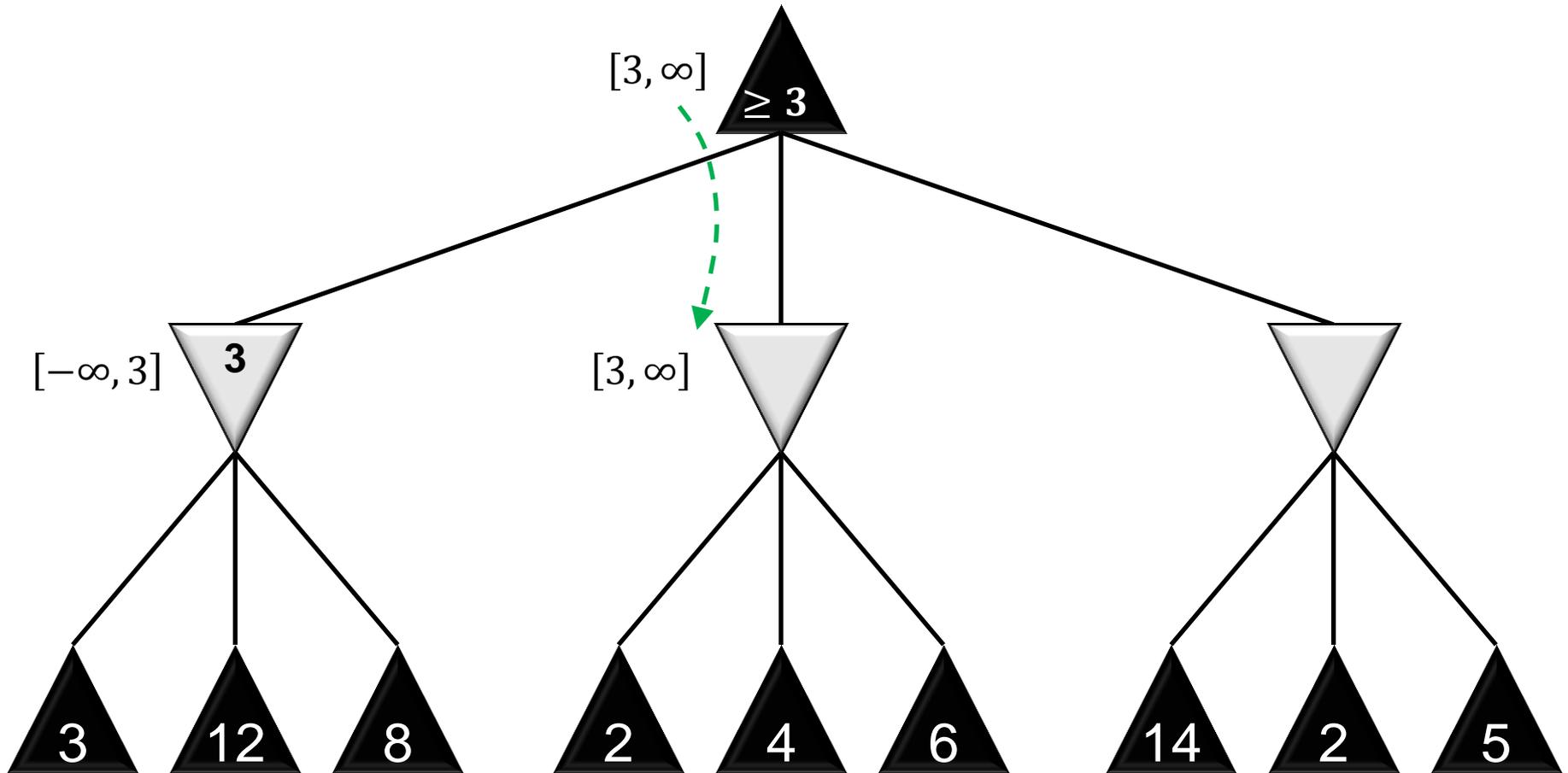
$\alpha\beta$ -Beispiel



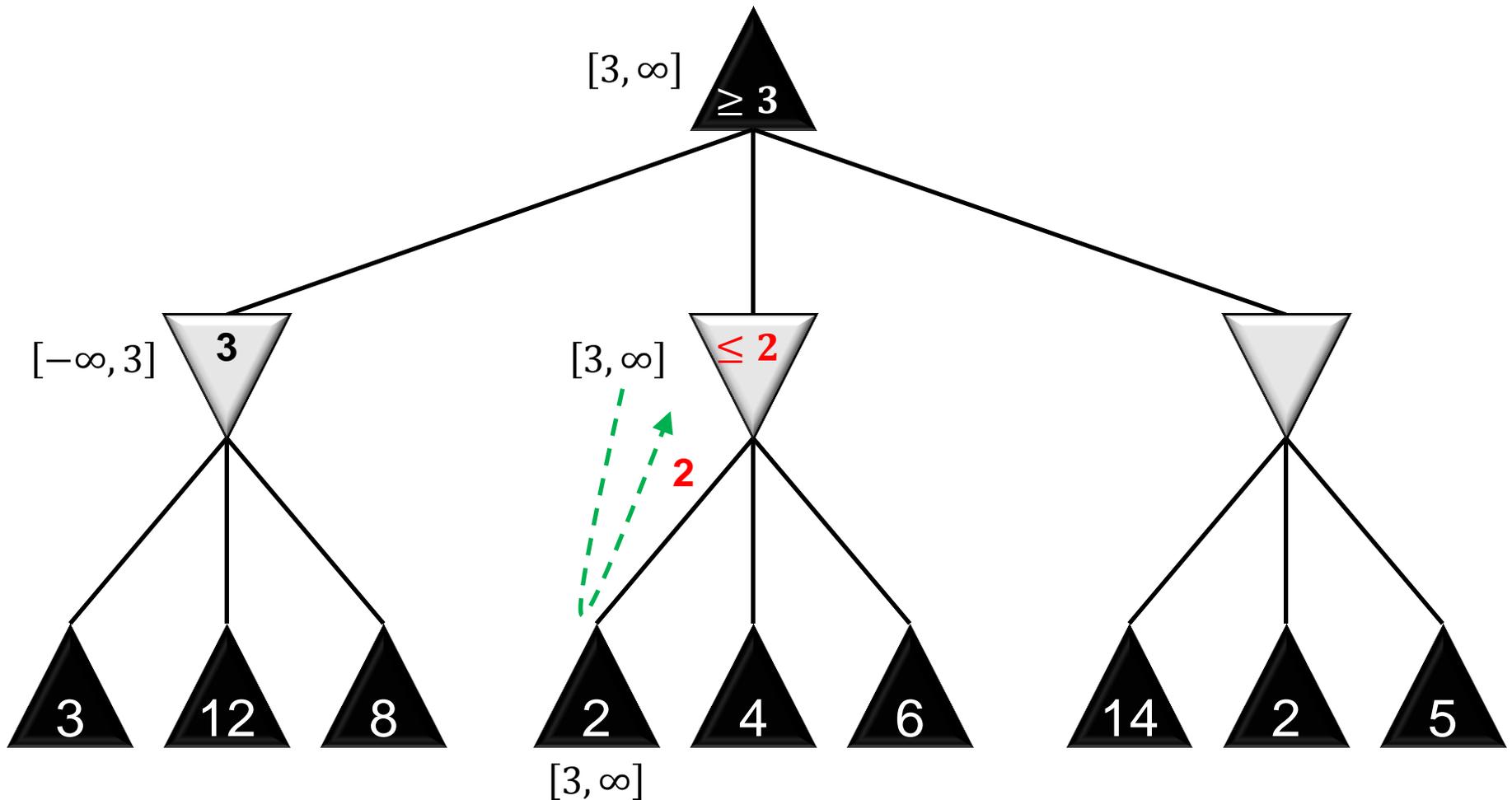
$\alpha\beta$ -Beispiel



$\alpha\beta$ -Beispiel

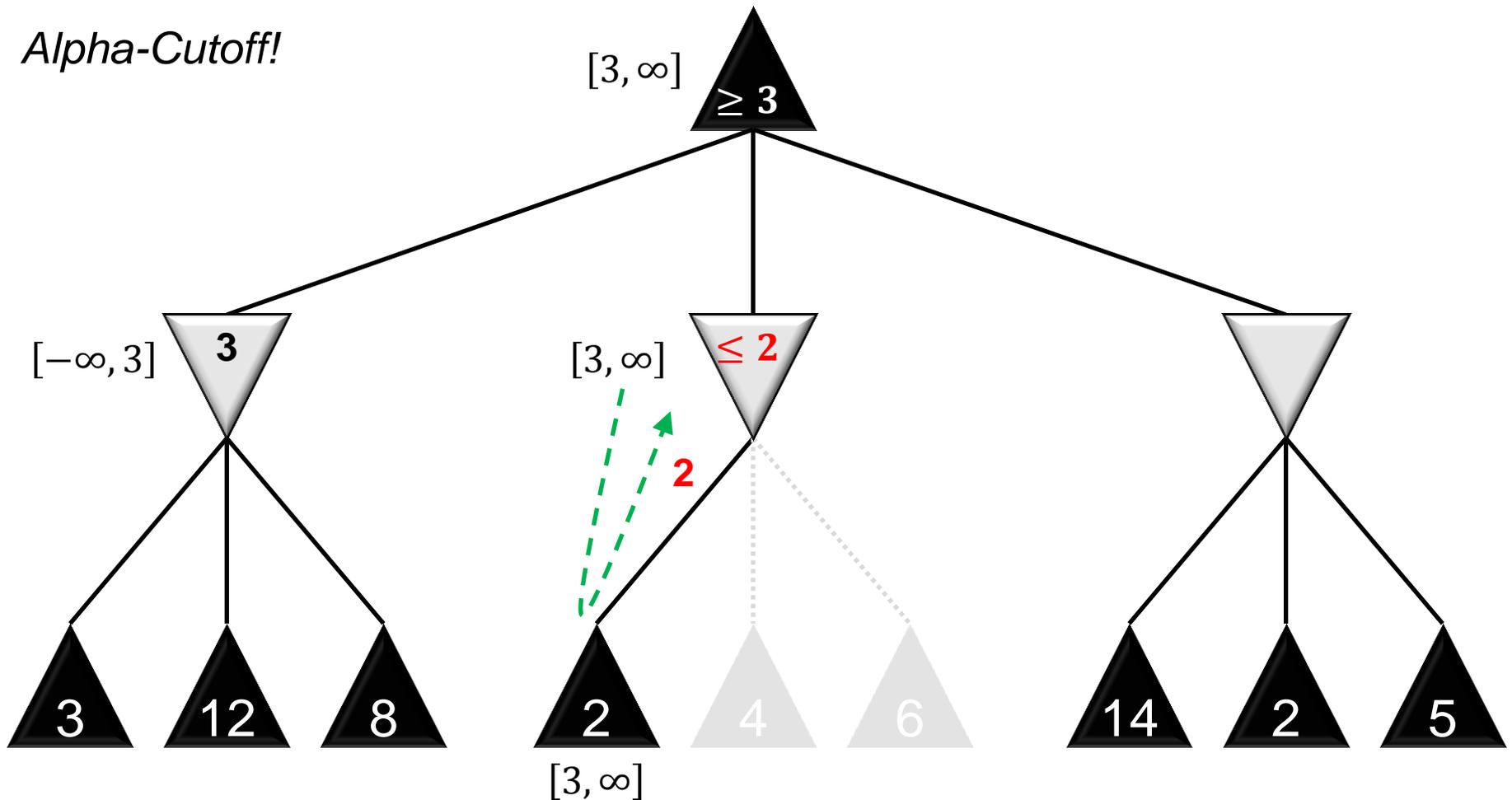


$\alpha\beta$ -Beispiel

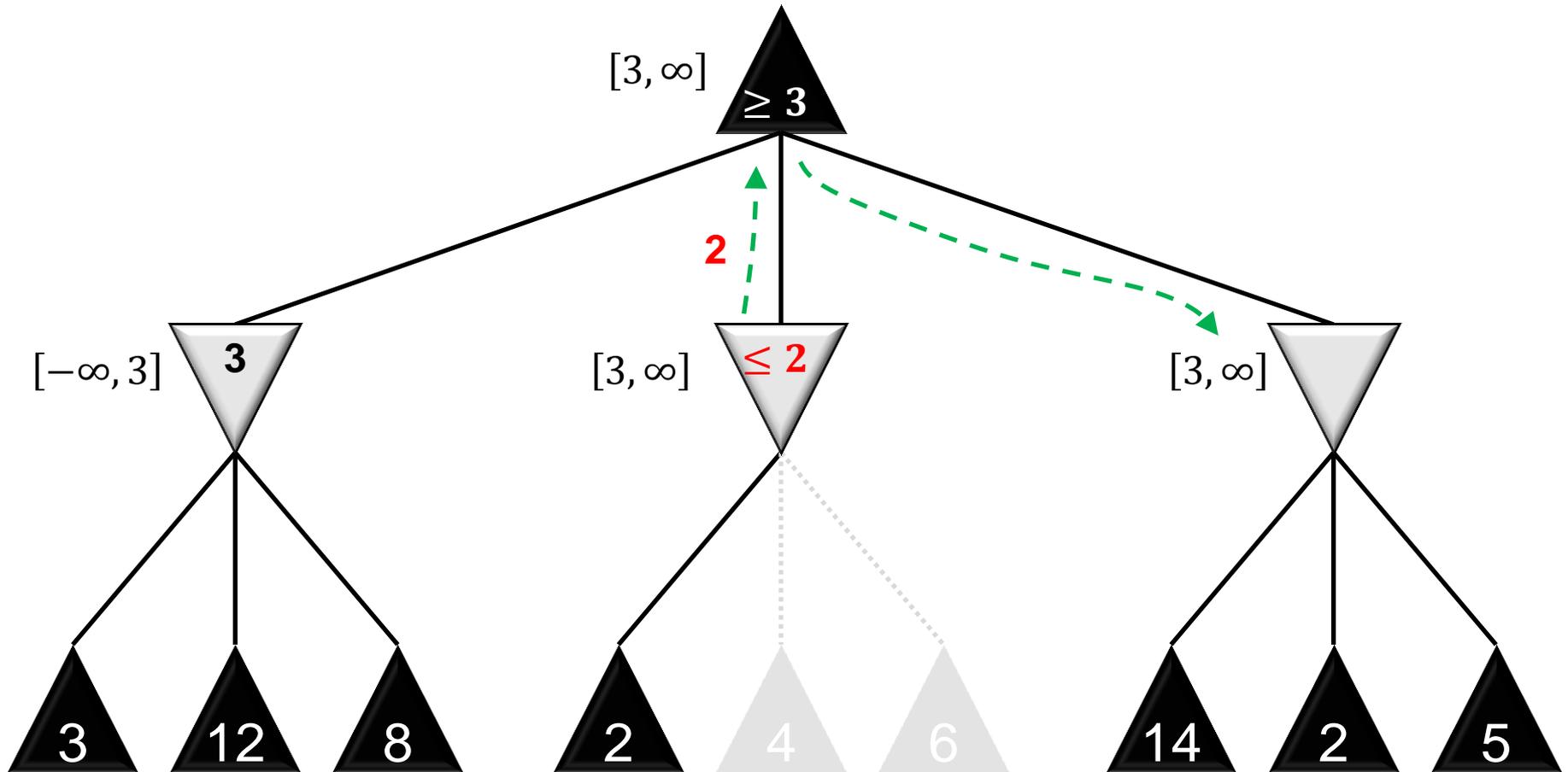


$\alpha\beta$ -Beispiel

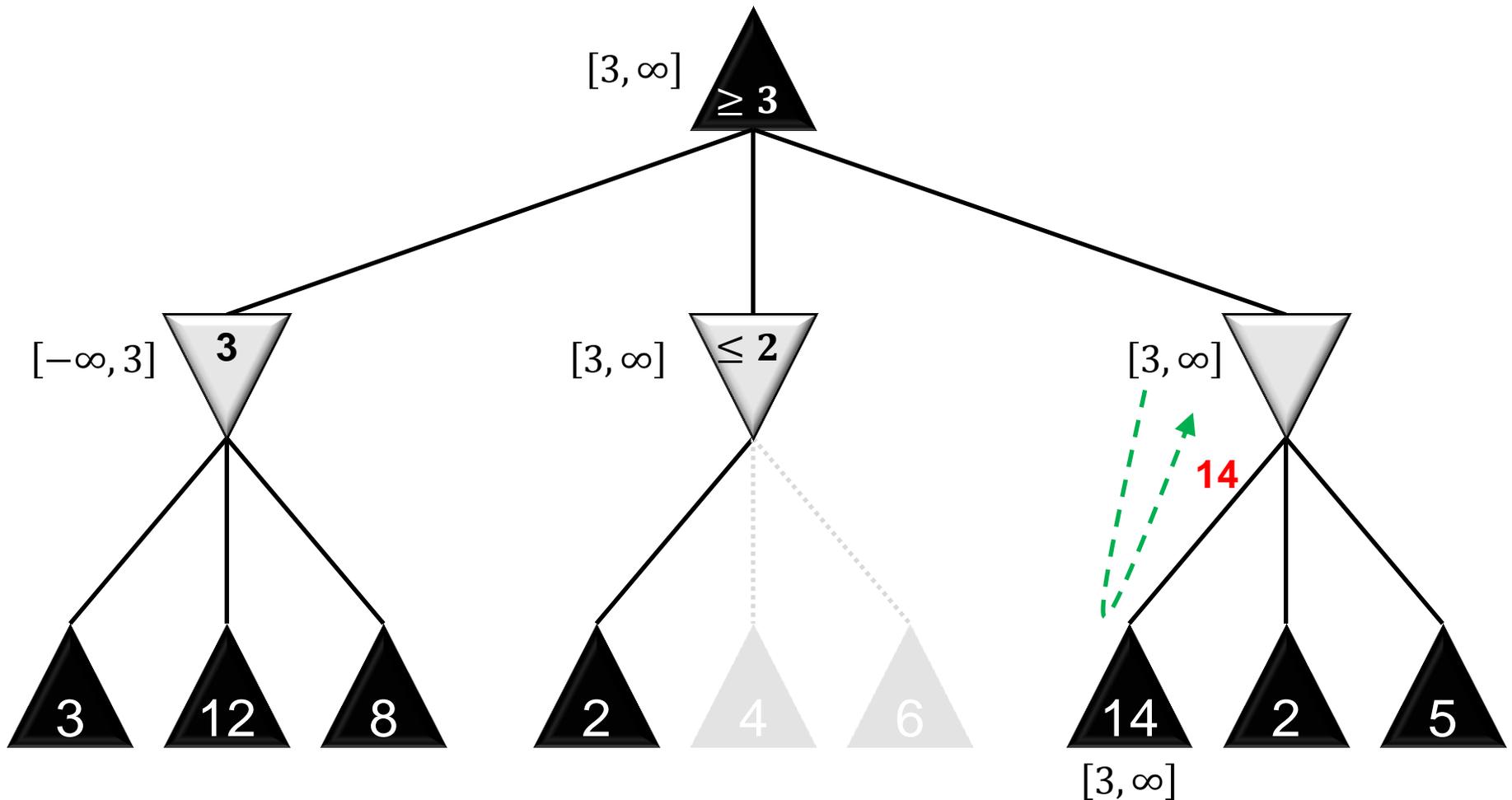
Alpha-Cutoff!



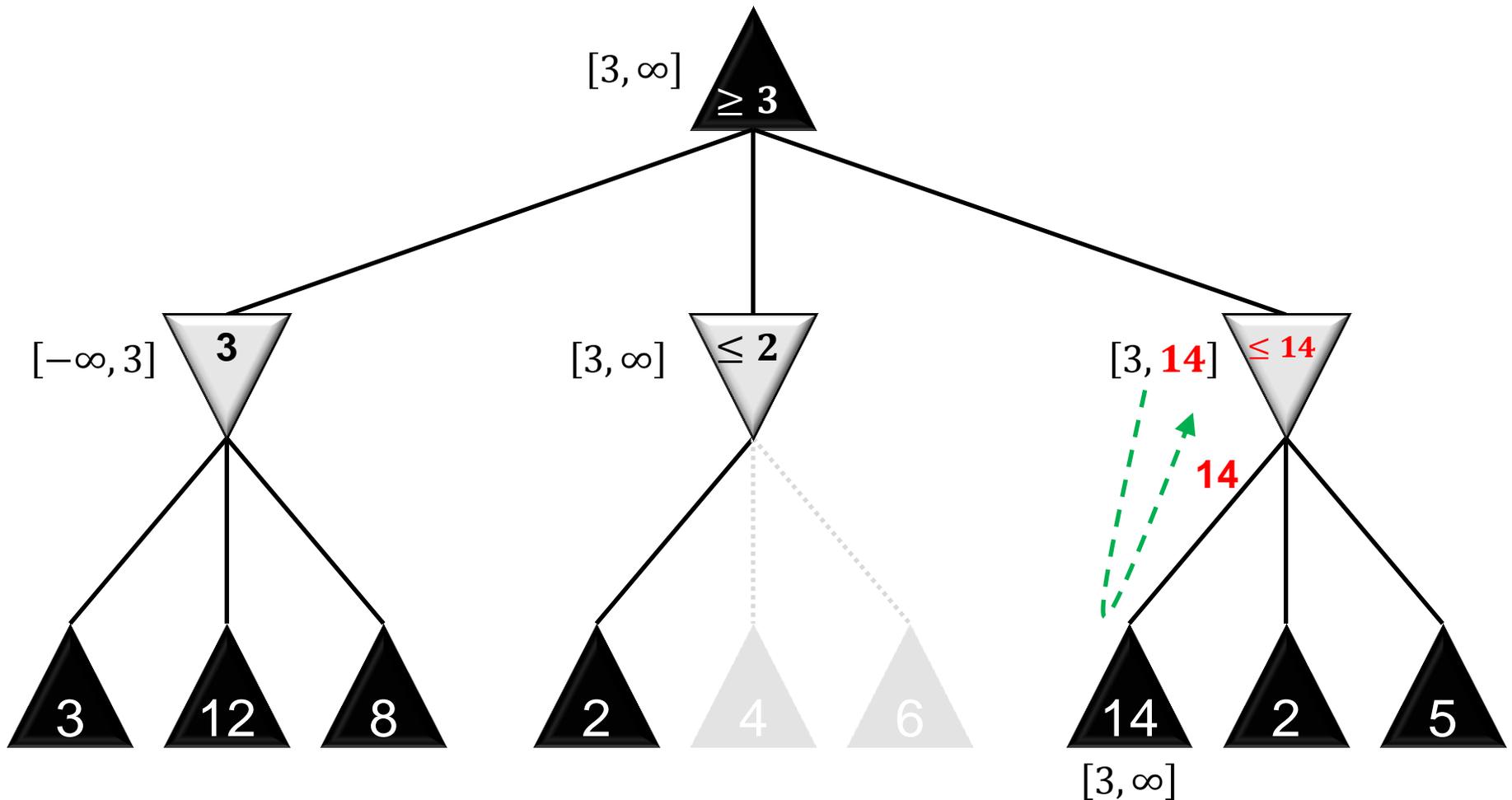
$\alpha\beta$ -Beispiel



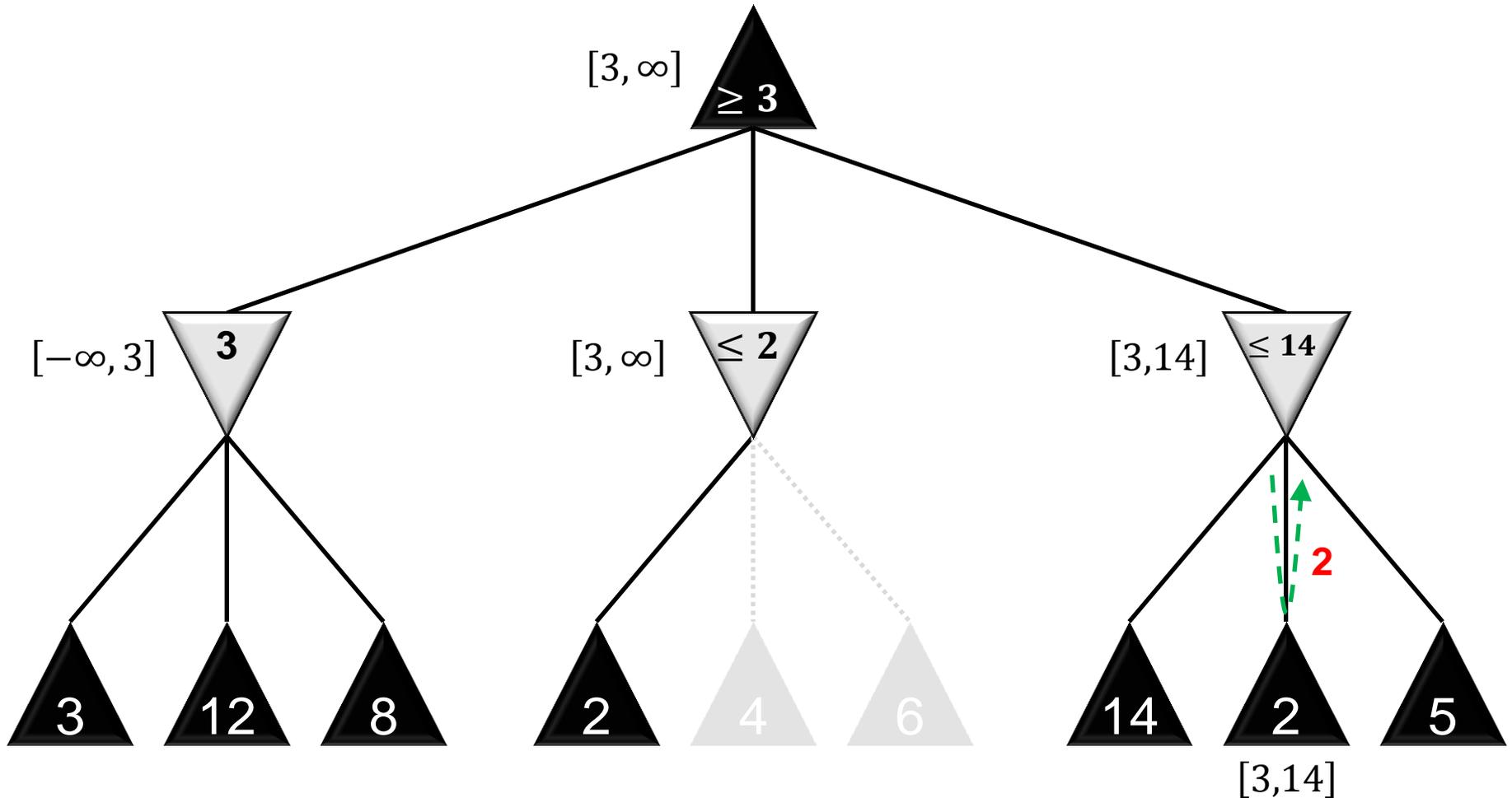
$\alpha\beta$ -Beispiel



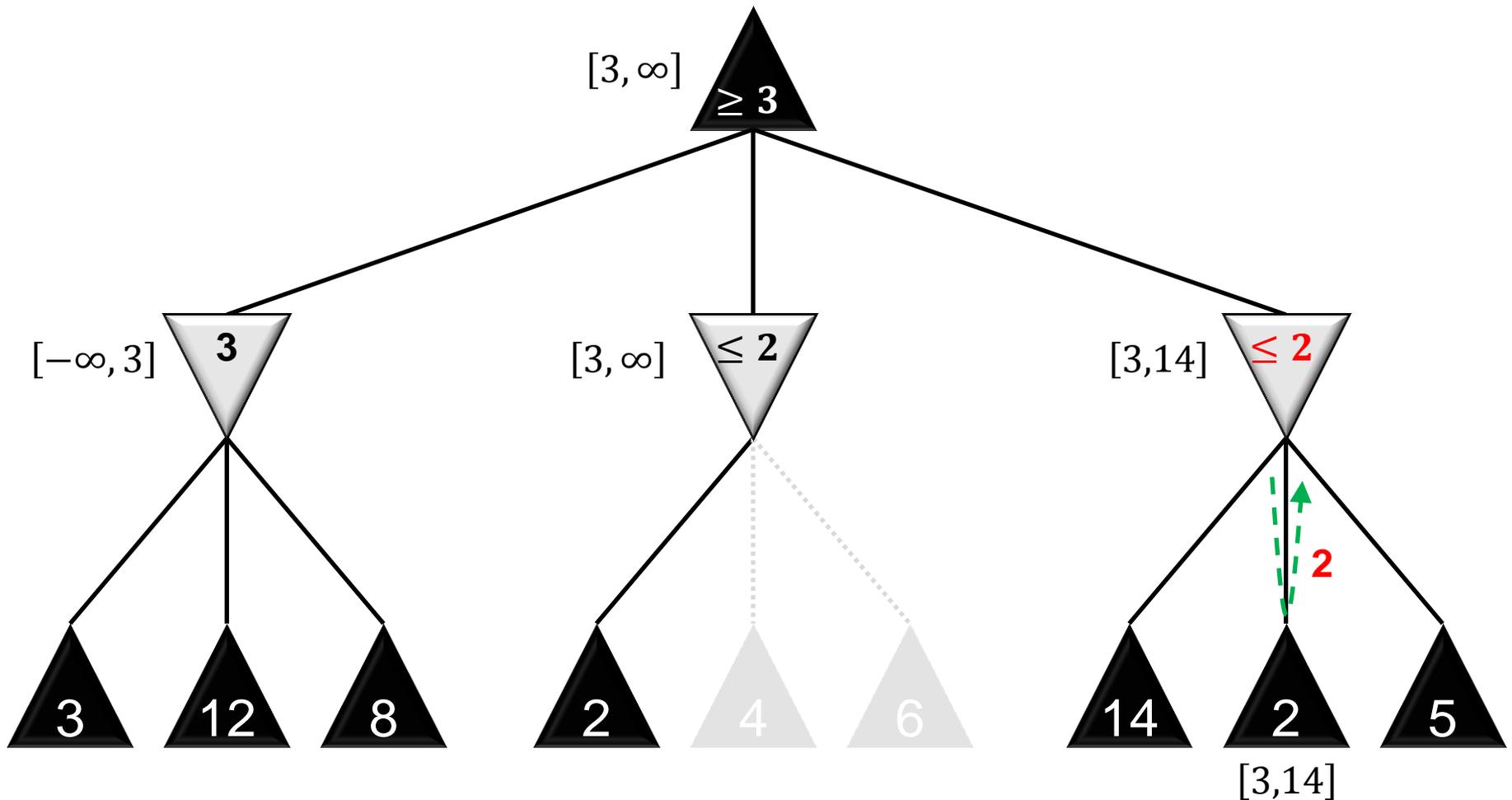
$\alpha\beta$ -Beispiel



$\alpha\beta$ -Beispiel

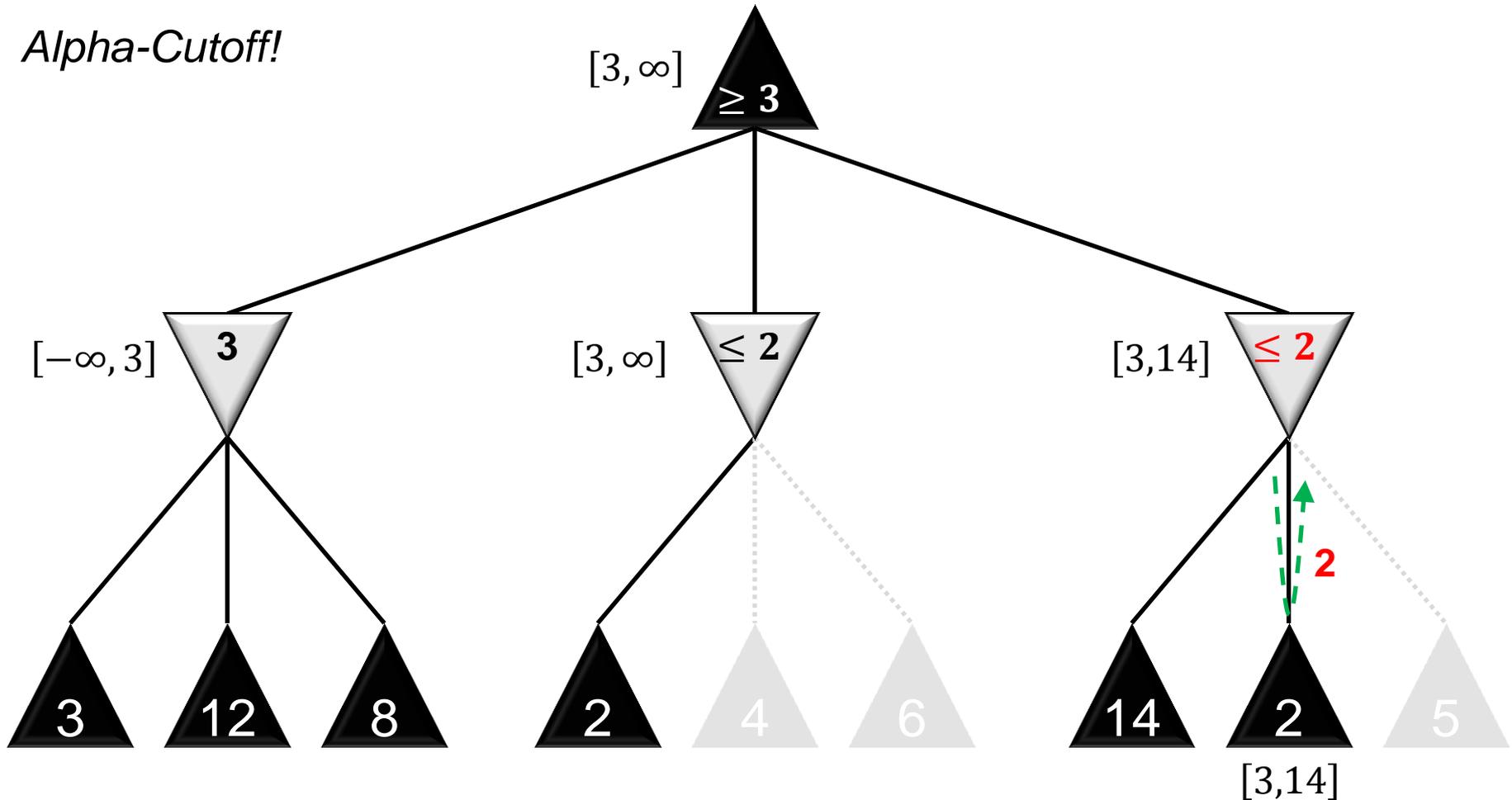


$\alpha\beta$ -Beispiel

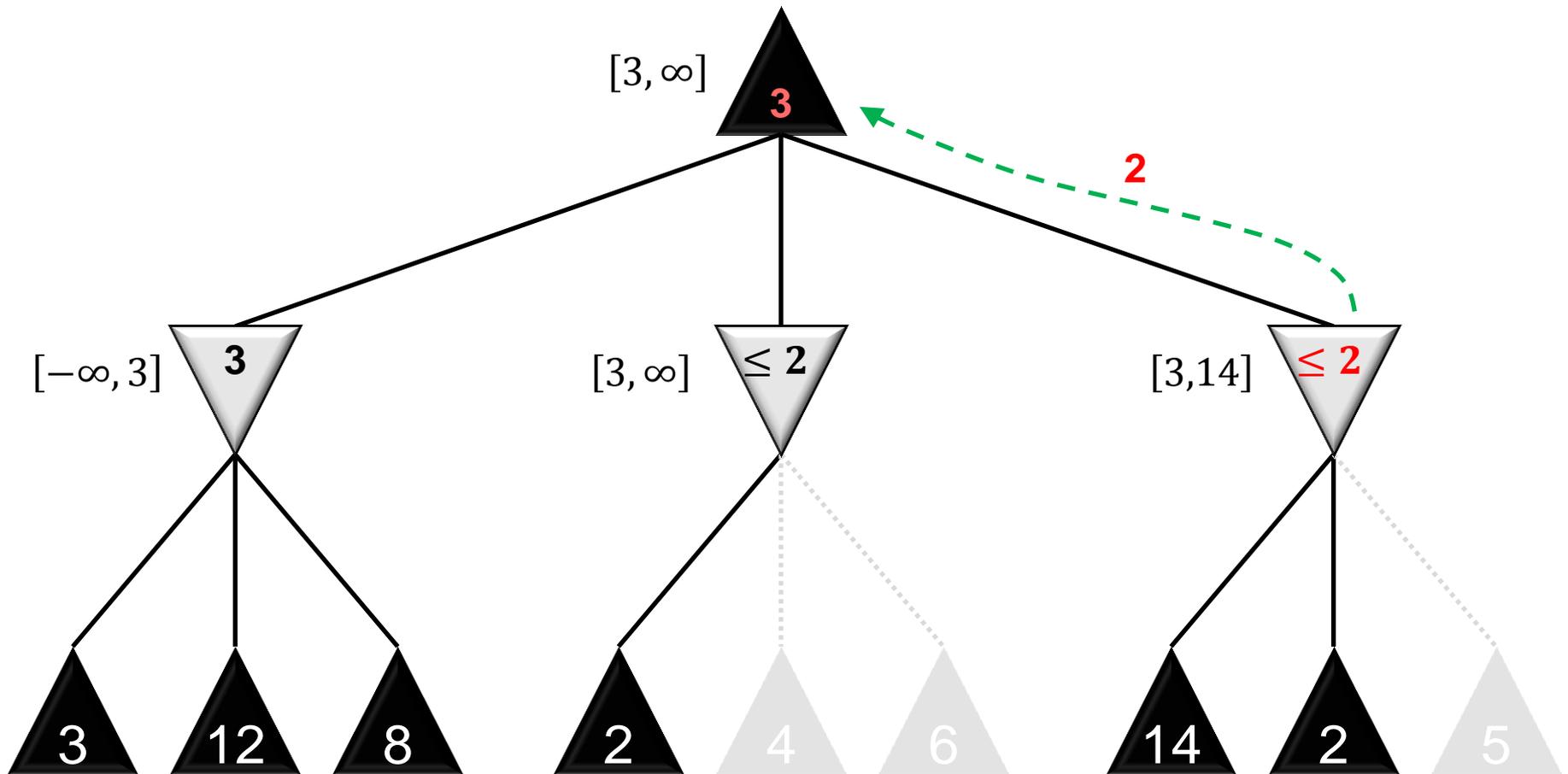


$\alpha\beta$ -Beispiel

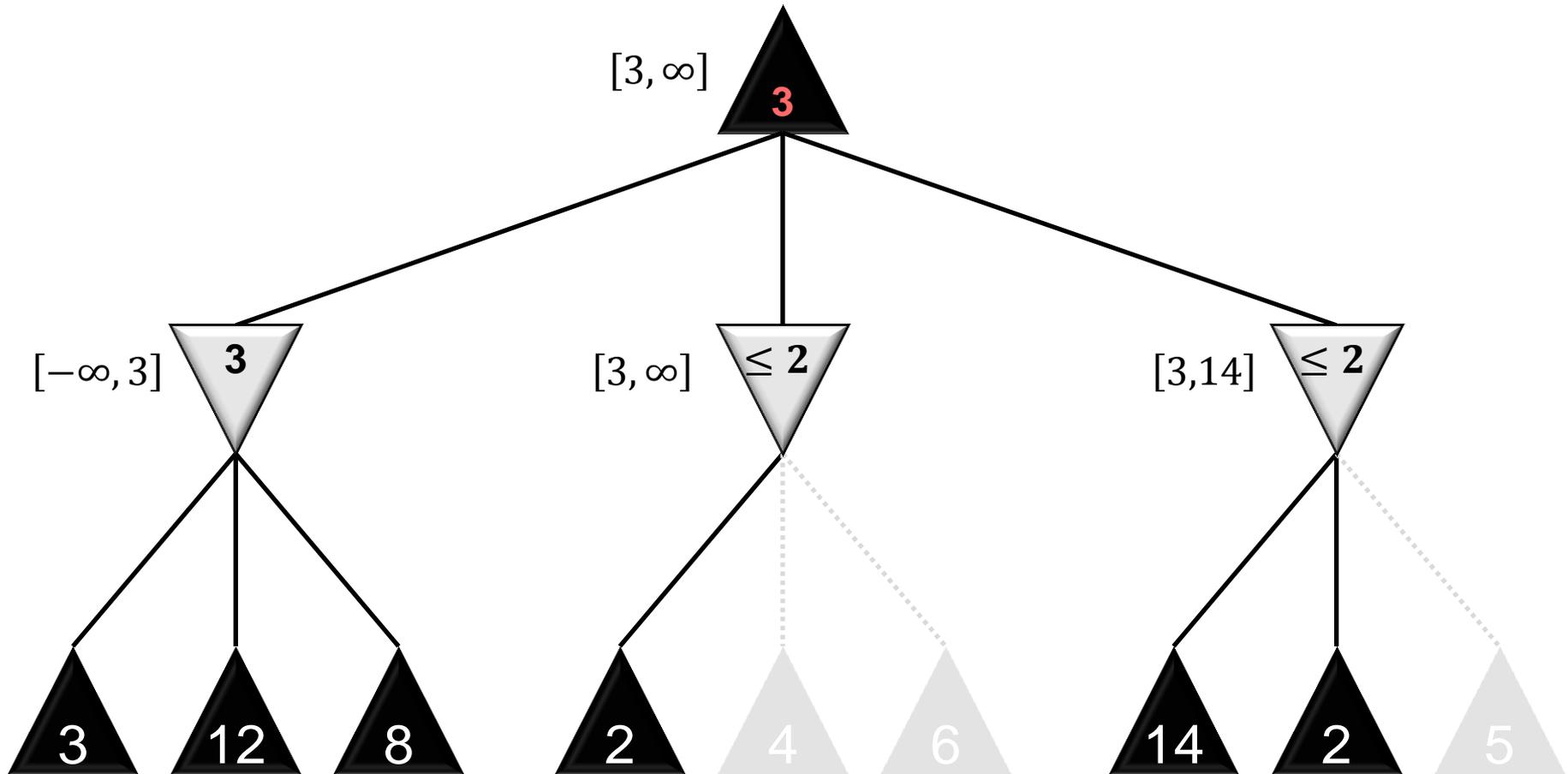
Alpha-Cutoff!



$\alpha\beta$ -Beispiel



$\alpha\beta$ -Beispiel

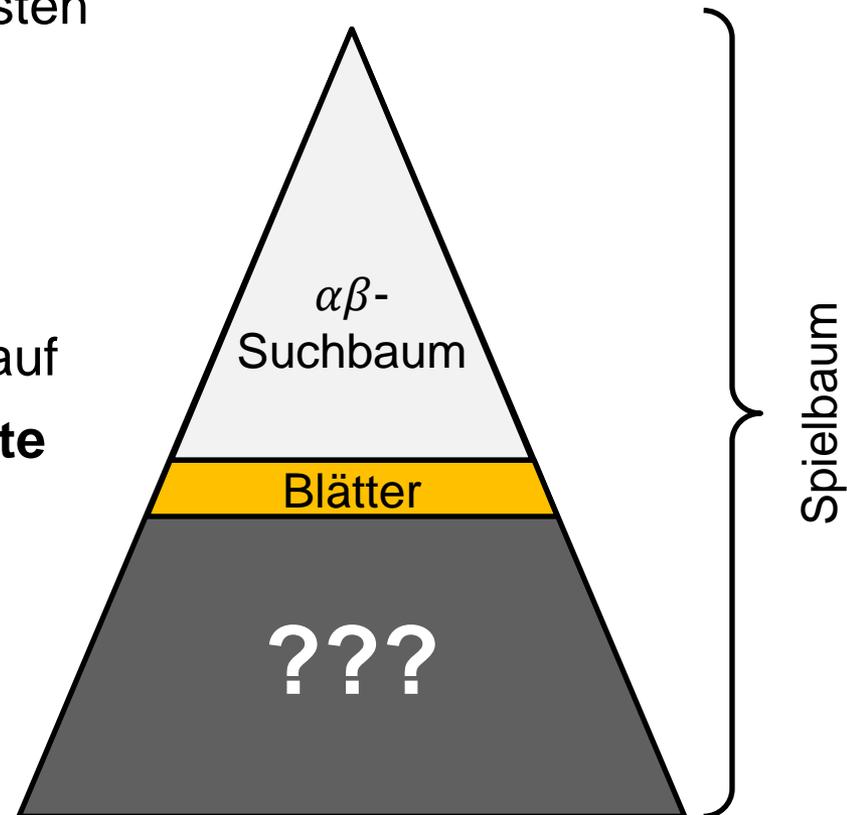


$\alpha\beta$ + Monte Carlo Tree Search

- Es gibt im Wesentlichen 2 Möglichkeiten, $\alpha\beta$ und MCTS zu kombinieren:
 1. **MCTS** als Evaluierungsfunktion für $\alpha\beta$
 2. $\alpha\beta$ als Default Policy für **MCTS**

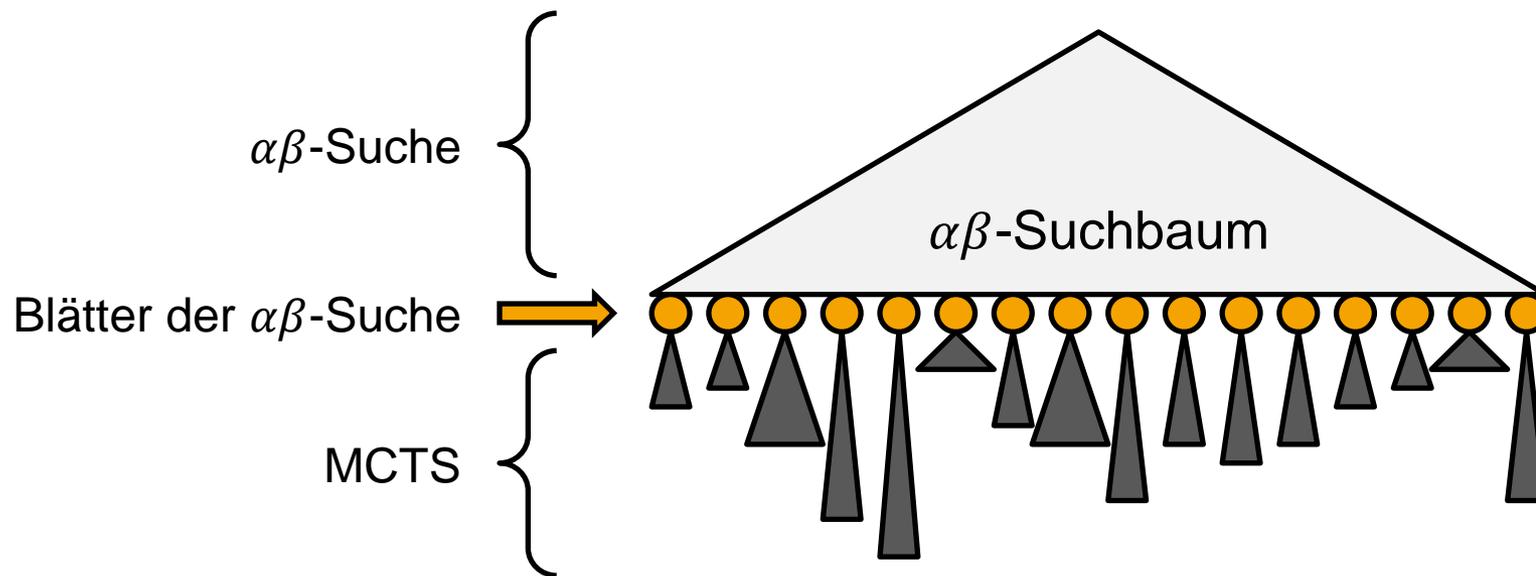
MCTS als Evaluierungsfunktion für $\alpha\beta$

- **Hauptproblem von $\alpha\beta$:** die meisten Probleme sind **zu groß**, um sie vollständig zu traversieren
- **Lösung:** Limitierung der Suche auf **maximale Suchtiefe / Suchbreite**
 - **Problem:** Evaluierungswert der „Blätter“ muss ermittelt werden



MCTS als Evaluierungsfunktion für $\alpha\beta$

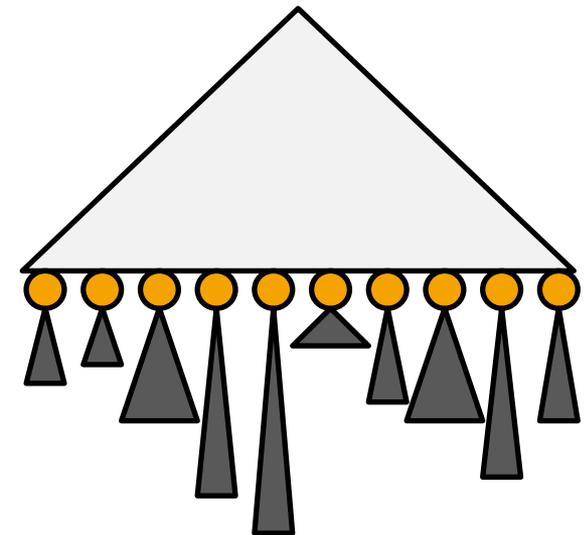
- = „Klassische“ Kombination
- MCTS ermittelt Evaluierungswert der Blätter für $\alpha\beta$ per Simulation
- Gut, wenn keine Evaluierungsfunktion bekannt ist



MCTS als Evaluierungsfunktion für $\alpha\beta$

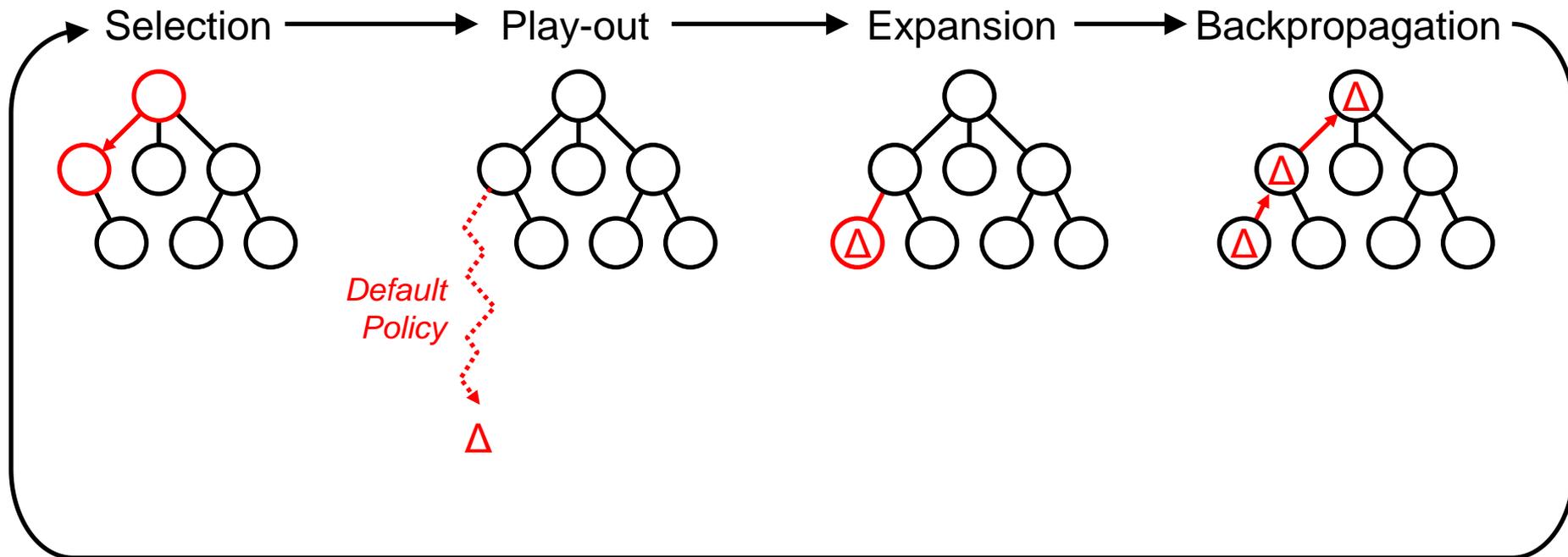
Eigenschaften:

- Knotenevaluierung ohne Evaluierungsfunktion
- Policy für MCTS notwendig
 - *Random Policy möglich*
- Ggf. falsche / schlechte Evaluierungswerte



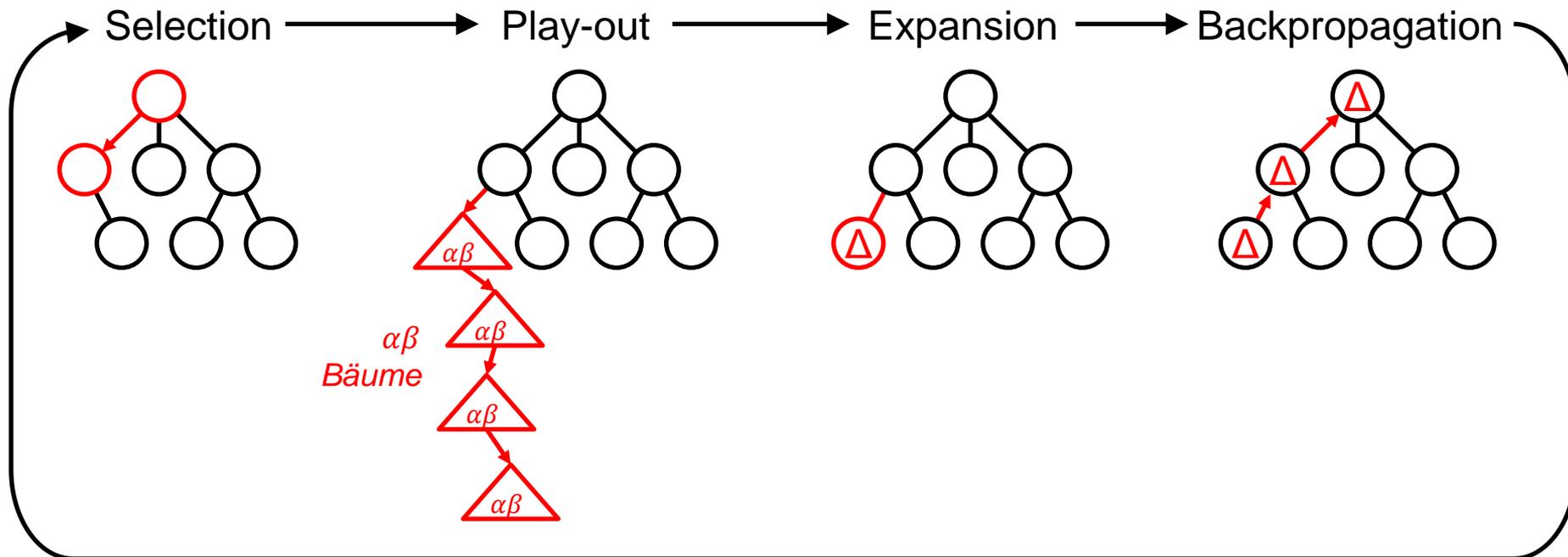
$\alpha\beta$ als Default Policy für MCTS

- Standard MCTS: Verwendet „Default Policy“



$\alpha\beta$ als Default Policy für MCTS

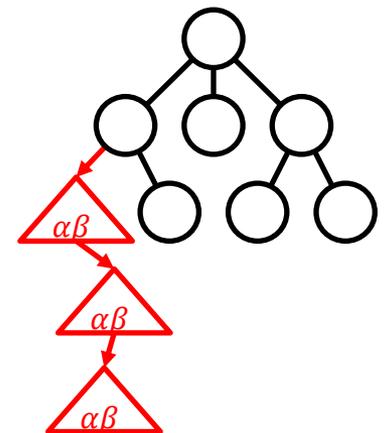
- $MC_{\alpha\beta}$ ersetzt Default Policy durch $\alpha\beta$ Suche
 - $\Rightarrow \alpha\beta$ verbessert schwache Evaluierungsfunktion



$\alpha\beta$ als Default Policy für MCTS

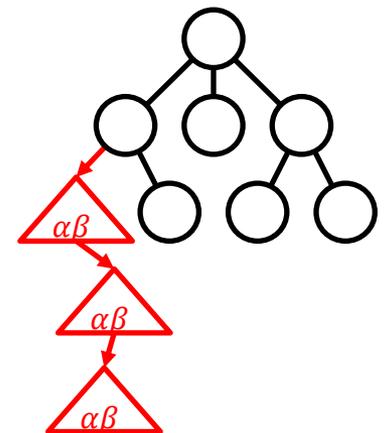
■ Probleme:

- Suchbreite- und Tiefe muss beschränkt werden
 - => Evaluierungsfunktion für $\alpha\beta$ Suche benötig
- **Performance** hängt von vielen **Heuristiken** ab:
 - Selection-Strategie von MCTS
 - Verhältnis aus Exploration und Exploitation
 - Sortierung / Selection-Strategie in $\alpha\beta$
 - Suchbreite von $\alpha\beta$
 - Suchtiefe $\alpha\beta$
 - Evaluierungsfunktion von $\alpha\beta$



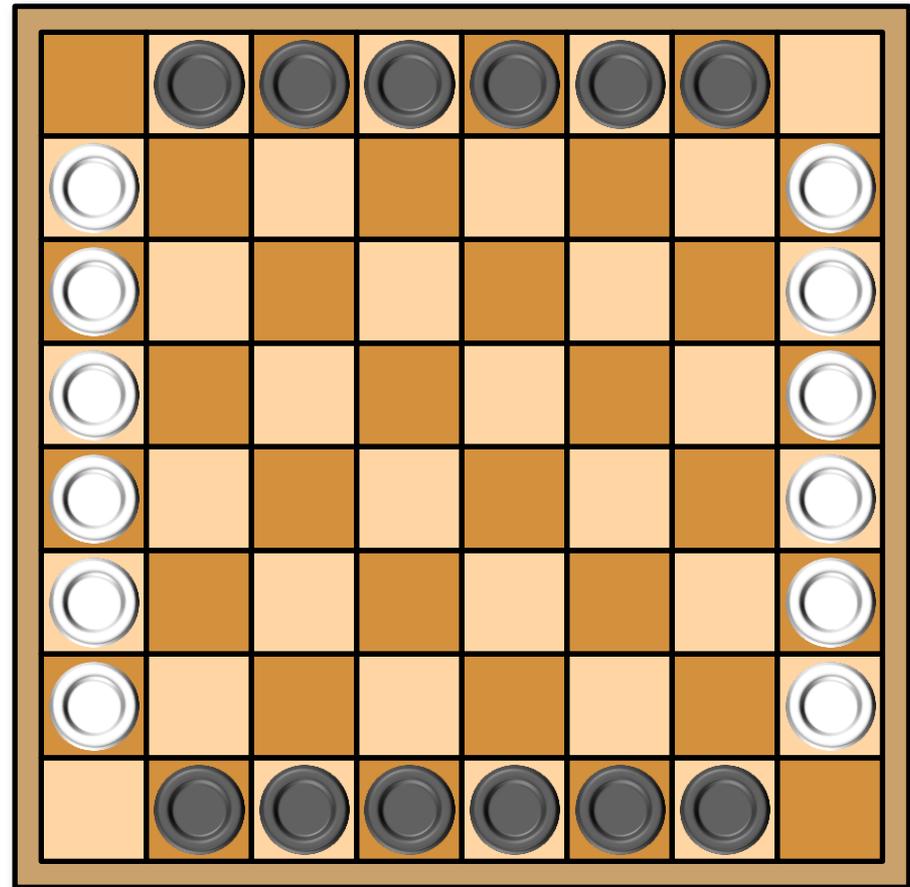
$\alpha\beta$ als Default Policy für MCTS

- Durch viele Heuristiken wird $MC_{\alpha\beta}$ **sehr problemspezifisch**
 - Parameter / Heuristiken müssen i.d.R. **per Hand getuned** werden
 - **Tradeoff:**
 - Großer $\alpha\beta$ Baum => langsam
 - Kleiner $\alpha\beta$ Baum => schlechte Evaluierung
- $MC_{\alpha\beta}$ wird trotzdem in $MC-LOA_{\alpha\beta}$ verwendet!



Lines of Action

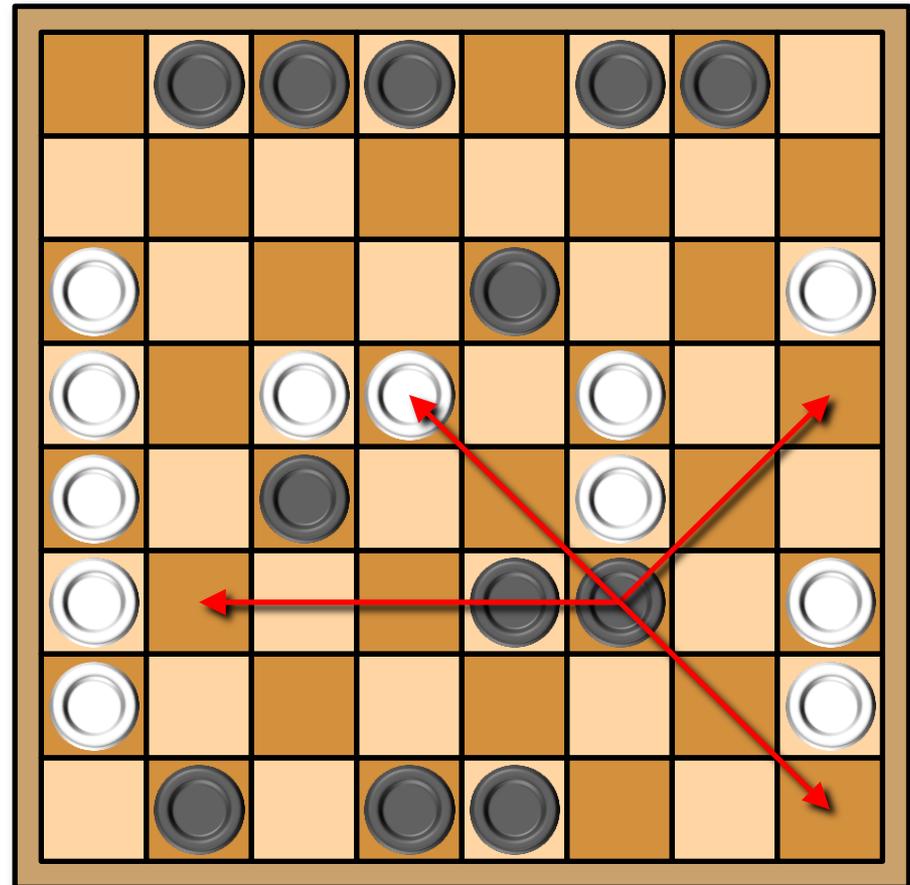
- Spiel für 2 Spieler
- 8x8 Spielfeld
- Pro Spieler 12 Spielsteine
- Abwechselnde Züge, schwarz beginnt



Startpositionen

Lines of Action

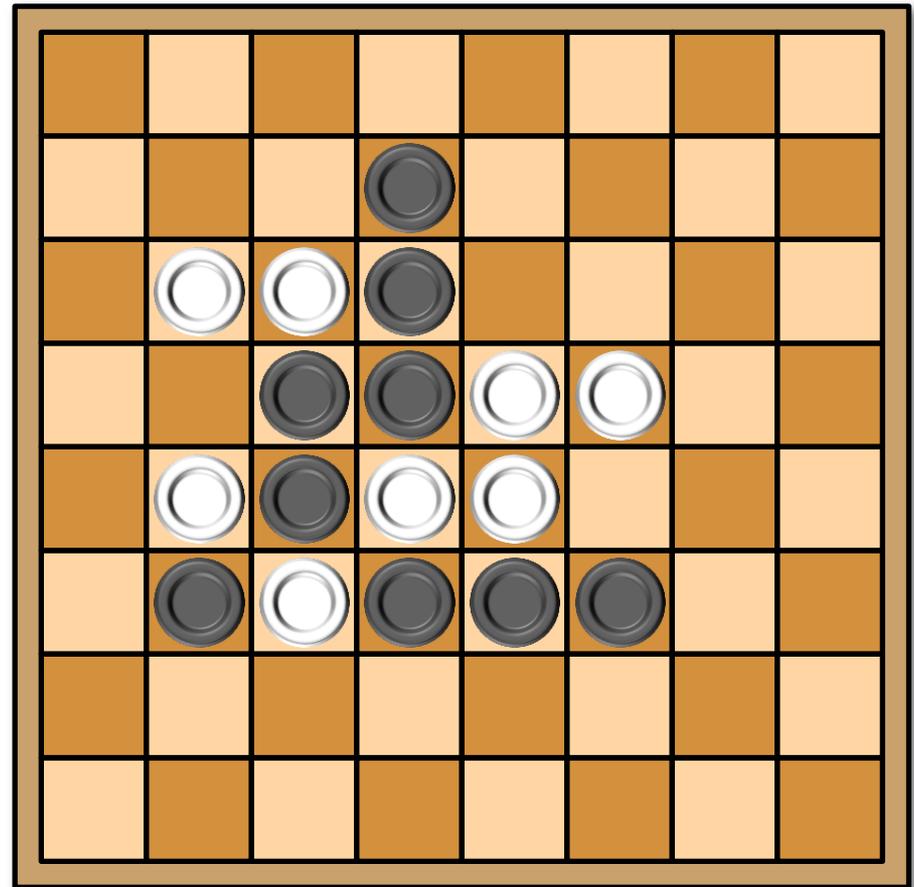
- **Züge:**
 - Gerade oder schräg
 - Zu ziehende Felder = Anzahl Steine auf der Zuglinie (Line of Action)
 - Gegnerische Steine dürfen geschlagen, nicht übersprungen werden
 - Eigene Steine dürfen übersprungen werden



Beispielzüge

Lines of Action

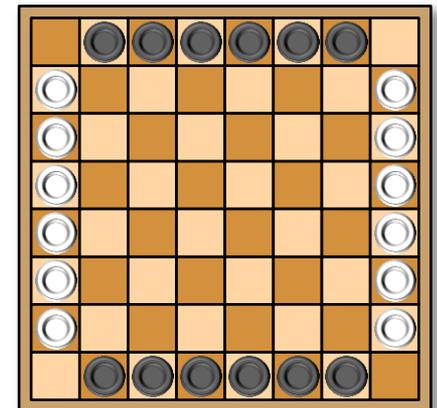
- **Ziel:**
 - Alle Steine in beliebiger Formation (gerade oder schräg) zu verbinden



Schwarz gewinnt

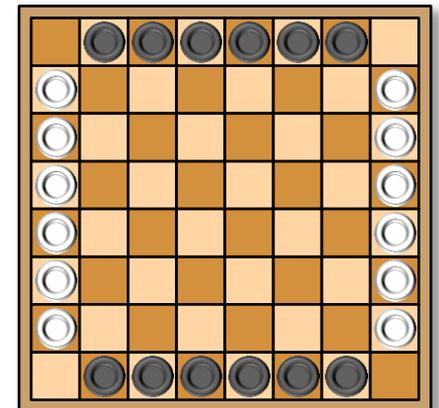
LOA mit $\alpha\beta$

- Bis 2010: reiner $\alpha\beta$ Ansatz als bestes Programm (MIA)
 - 2001 spielt MIA bereits auf Weltmeisterniveau
- Verhältnismäßig kleiner Suchbaum
 - Nur 8x8 Feld (Go: 19x19)
 - Wenig mögliche Züge
 - => Verzweigungsfaktor durchschnittlich 30
 - (Go: 250)
- Gute Evaluierungsfunktion existiert



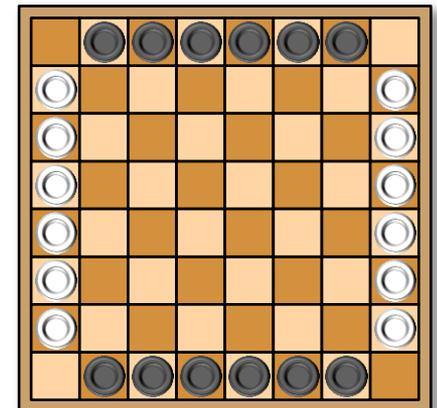
LOA mit MCTS

- 2010: MC-LOA erreicht 50% Gewinnrate gegen MIA
 - *Reiner Monte-Carlo Tree Search Ansatz*
- 2011: MC-LOA _{$\alpha\beta$} erreicht 60% Gewinnrate gegen MIA
 - *MCTS + $\alpha\beta$ als default Policy*



MC-LOA _{$\alpha\beta$} Heuristiken

- MCTS Selection:
 - ≤ 5 Mal besucht: Corrective Strategy (versuche, Situation zu verbessern)
 - > 5 Mal besucht: UCT mit Progressive Bias (= Domänenwissen)
 - Test auf Spielende in einem Zug (gefunden \Rightarrow Kein Playout mehr)
- $\alpha\beta$ Selection / Reihenfolge
 - Durch Einteilung in gewichtete Move Categories
 - Verwendung von Killer Moves aus vorherigen Zügen
- $\alpha\beta$ Tiefe / Breite:
 - Maximale Tiefe = 2
 - Maximal Breite: 7 in Ebene 1, 5 in Ebene 2
- Test auf Spielende in einem Zug in Ebene 1

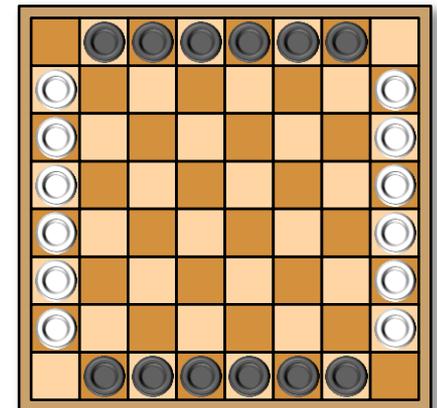


Evaluierung

- MIA gegen MC-LOA _{$\alpha\beta$}
- 1000 Spiele auf 2,2 Ghz AMD Opteron
- Siege von MC-LOA _{$\alpha\beta$} in Abhängigkeit von

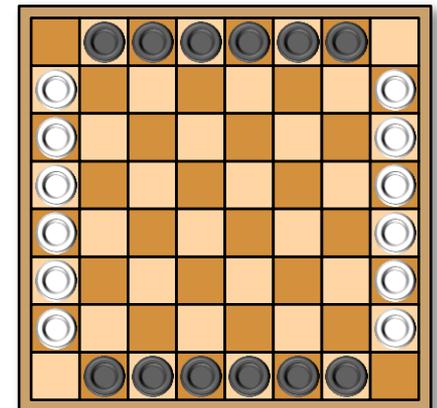
Zugzeit:

- 1 Sekunde: 44,8%
- 5 Sekunden: 57,6%
- 30 Sekunden: 59,85%



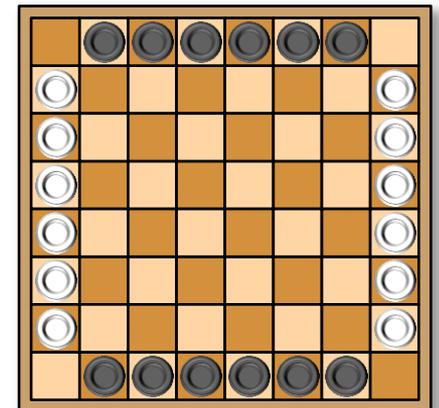
Taktische Stärke:

- Taktische Phase liegt bei LOA am Ende des Spiels
 - Konterzug auf Angriff muss gefunden werden
- MC-LOA _{$\alpha\beta$} benötigt für Lösen von Endspielpositionen 20% mehr Zeit und muss 5% mehr Knoten untersuchen
 - => bessere Performance von MC-LOA _{$\alpha\beta$} entsteht durch Herausspielen von besseren Positionen am Anfang



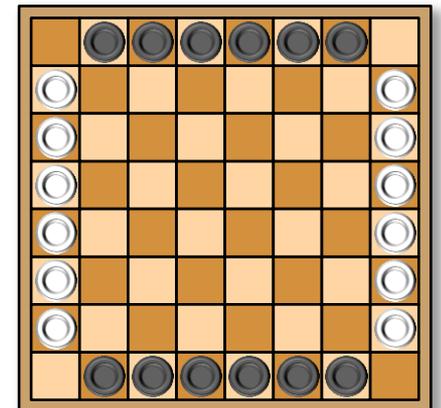
Corrective Strategy

- Minimiert Risiko, die Situation zu verschlechtern
 - **Evaluierungsfunktion** wird genutzt um Qualität der aktuellen und der Folgesituation zu bestimmen
 - Schlechtere Züge => Gewicht nahe 0
 - Gute Züge => Gewicht aus Evaluierungsfunktion
- Zug wird zufällig gewählt per Gewicht



Corrective Strategy

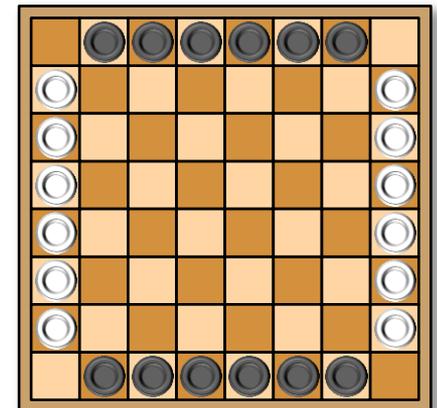
```
defaultValue = evaluate(board);  
  
foreach (Move m in moveList) {  
    value = evaluate(board, m);  
    if (value > bound)  
        return m;  
    else if (value <= defaultValue)  
        m.score = Epsilon;  
    else  
        m.score = m.getMoveCategoryWeight(board);  
    scoreSum += m.score;  
}
```



UCT in MC-LOA _{$\alpha\beta$}

- Für Selektionsschritt
- Kein „reines UCT“
- Progressive Bias fügt Domänenwissen hinzu (durch Evaluierungsfunktion)

$$k \in \operatorname{argmax}_{i \in I} (UCT + \text{ProgressiveBias})$$



$$k \in \operatorname{argmax}_{i \in I} \left(\underbrace{v_i + \sqrt{\frac{C \cdot \ln(n_p)}{n_i}}}_{UCT} + \underbrace{\frac{W \cdot P_{mc}}{\sqrt{l_i + 1}}}_{\text{Progressive Bias}} \right)$$

I = Menge der direkten Kindknoten

p = aktueller Knoten

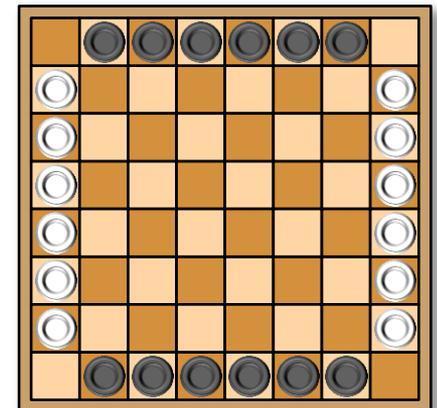
v_i = Wert des Knotens i

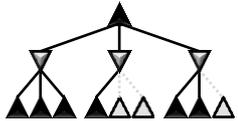
W, C = Konstanten (handgetuned)

P_{mc} = Wahrscheinlichkeit für Move-category mc

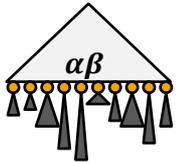
n_i = Anzahl Besuche von Knoten i

l_i = Anzahl der Niederlagen von Knoten i

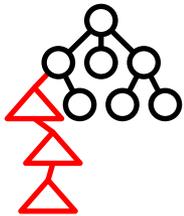




- $\alpha\beta$ Algorithmus



- MCTS als Evaluierungsfunktion für $\alpha\beta$



- $\alpha\beta$ als Default Policy für MCTS



- $MC-LOA_{\alpha\beta}$ für Anwendung in Lines of Action

Billings, D. & Björnsson, Y., 2004. Search and knowledge in Lines of Action. In: *Advances in Computer Games*. s.l.:Springer, pp. 231-248.

Browne, C. B. et al., 2012. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1), pp. 1-43.

Chen, K.-H., Du, D. & Zhang, P., 2009. Monte-Carlo tree search and computer Go. In: *Advances in Information and Intelligent Systems*. s.l.:Springer, pp. 201-225.

Winands, M. H. & Björnsson, Y., 2010. Evaluation function based monte-carlo LOA. In: *Advances in Computer Games*. s.l.:Springer, pp. 33-44.

Winands, M. H. & Björnsson, Y., 2011. $\alpha\beta$ -based play-outs in monte-carlo tree search. s.l., s.n., pp. 110-117.