PRG-w4a:
ゲーム戦略といくつかの賢い枠組み

脇田建

2019.10.18

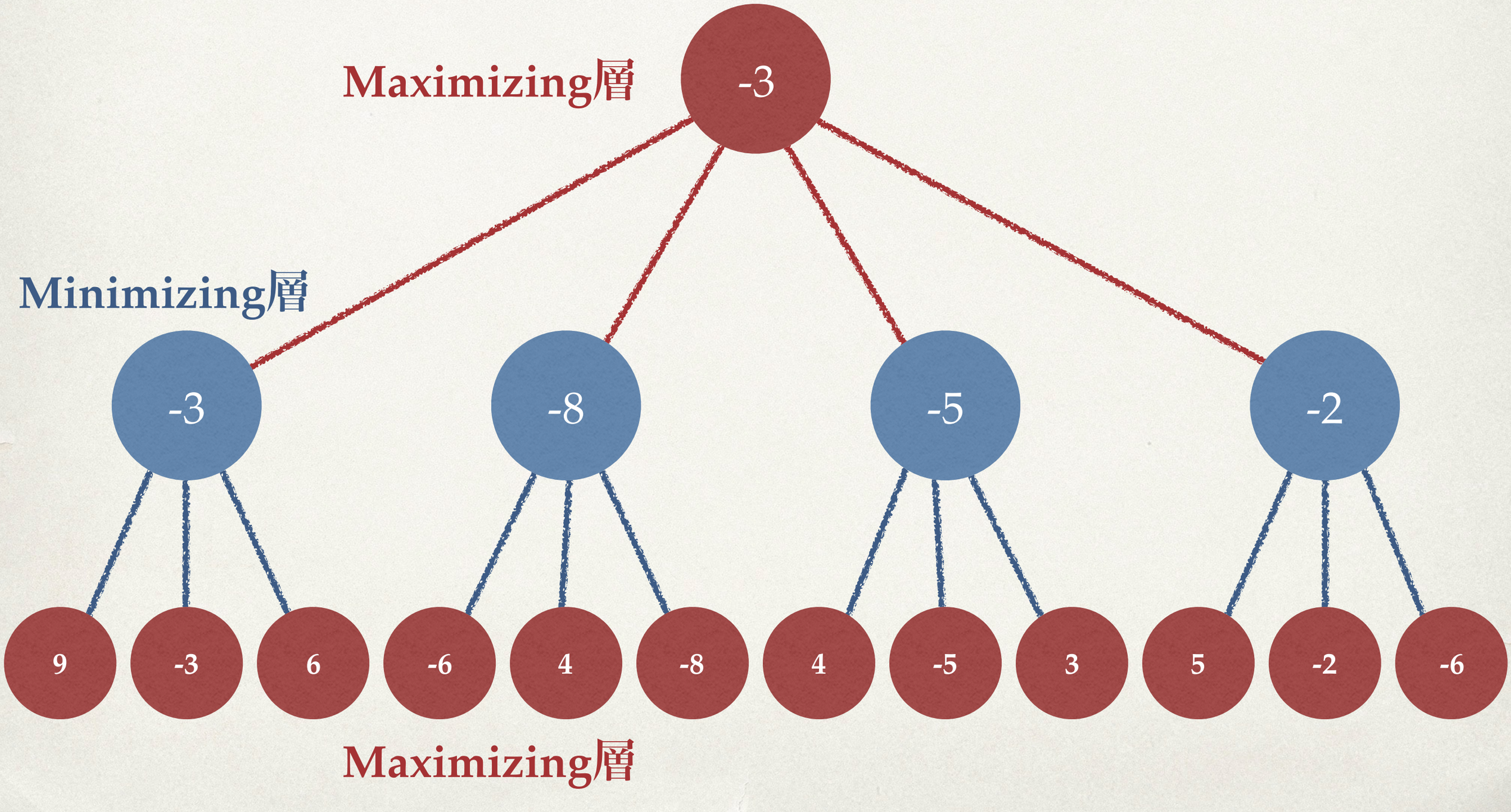
Alpha Beta法

Minimaxアルゴリズム

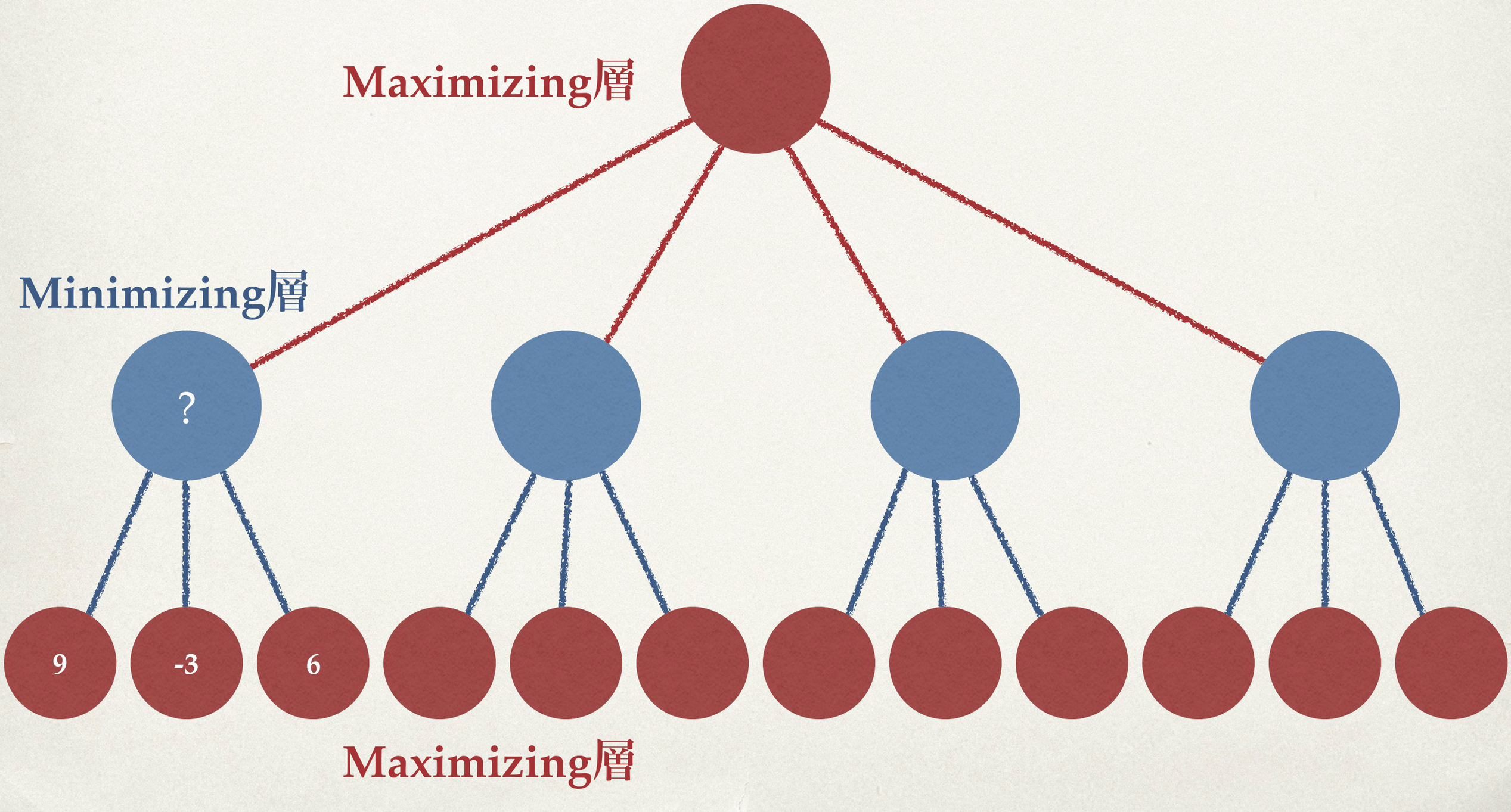
- def minimax(頂点, 深さ, プレーヤ): Double = {

```
    if (深さ == 0) v(頂点)
    プレーヤ match {
      case 自分 =>
        頂点.子頂点たち.foldLeft(-∞).((v, 子頂点) =>
          max(v, minimax(子頂点, 深さ-1, 相手))
      case 相手 => { 評価関数を最小化しようとする }
    }
}
```

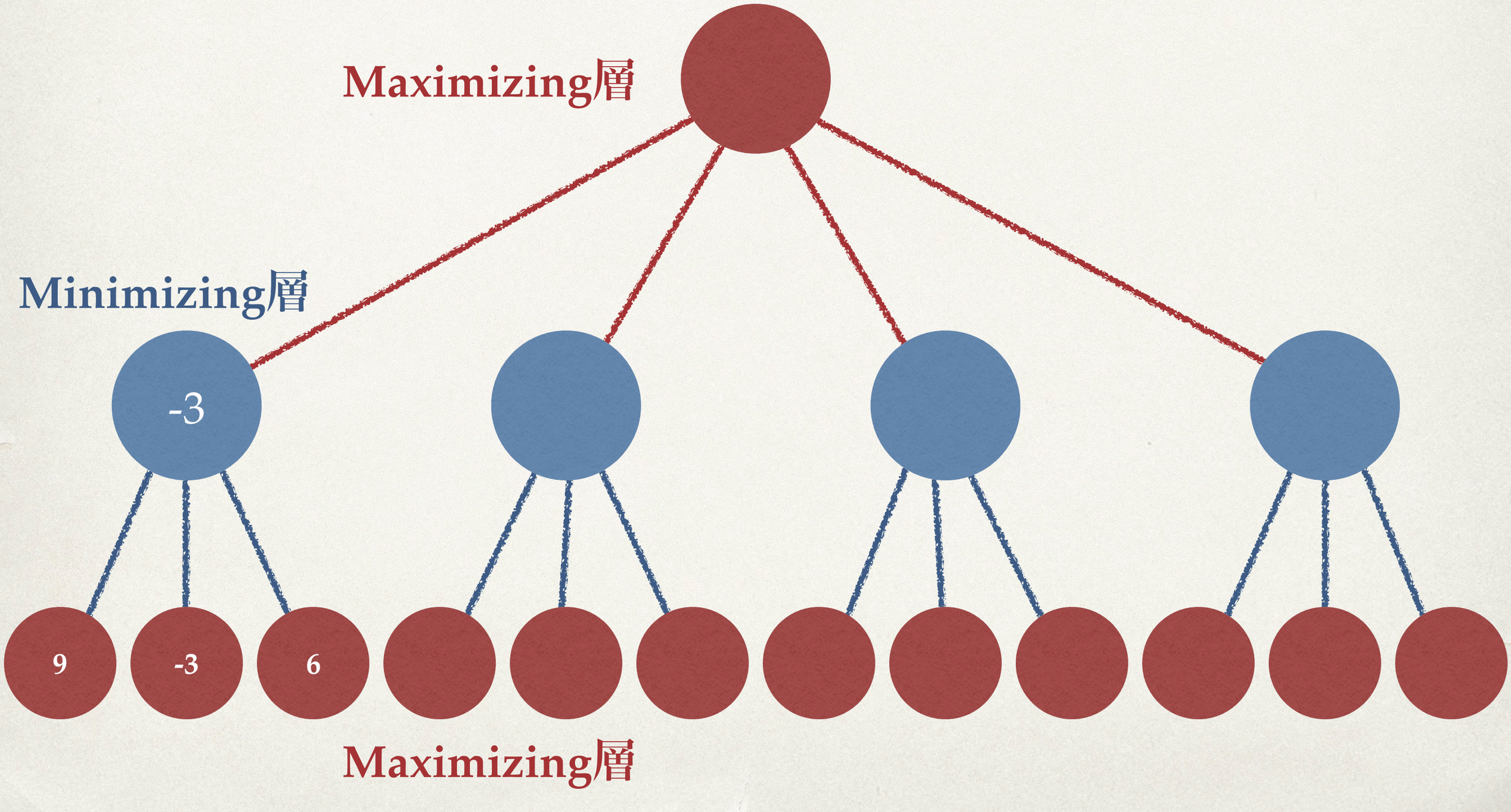
Minimax計算の無駄



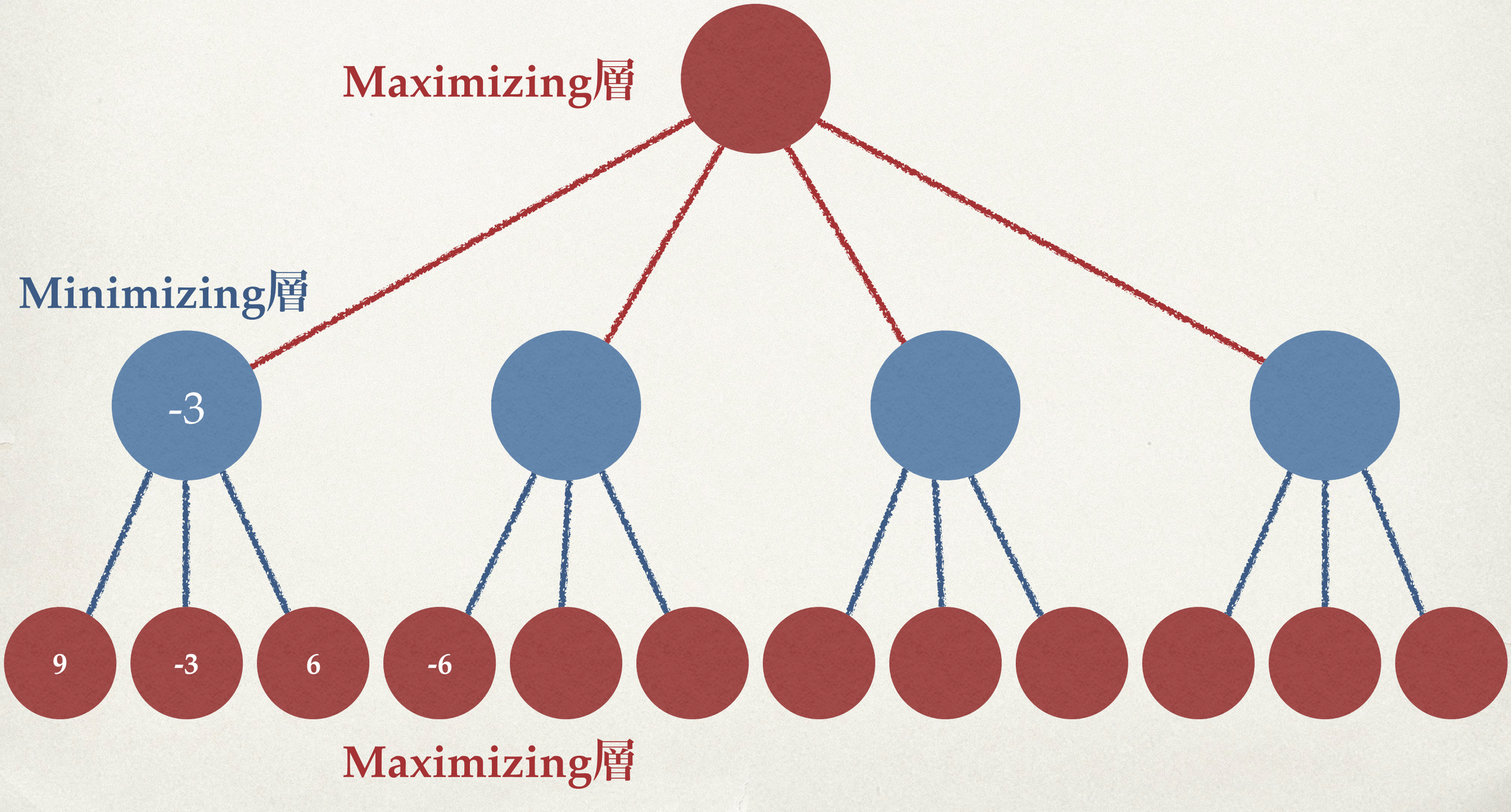
Minimax計算の無駄



Minimax計算の無駄



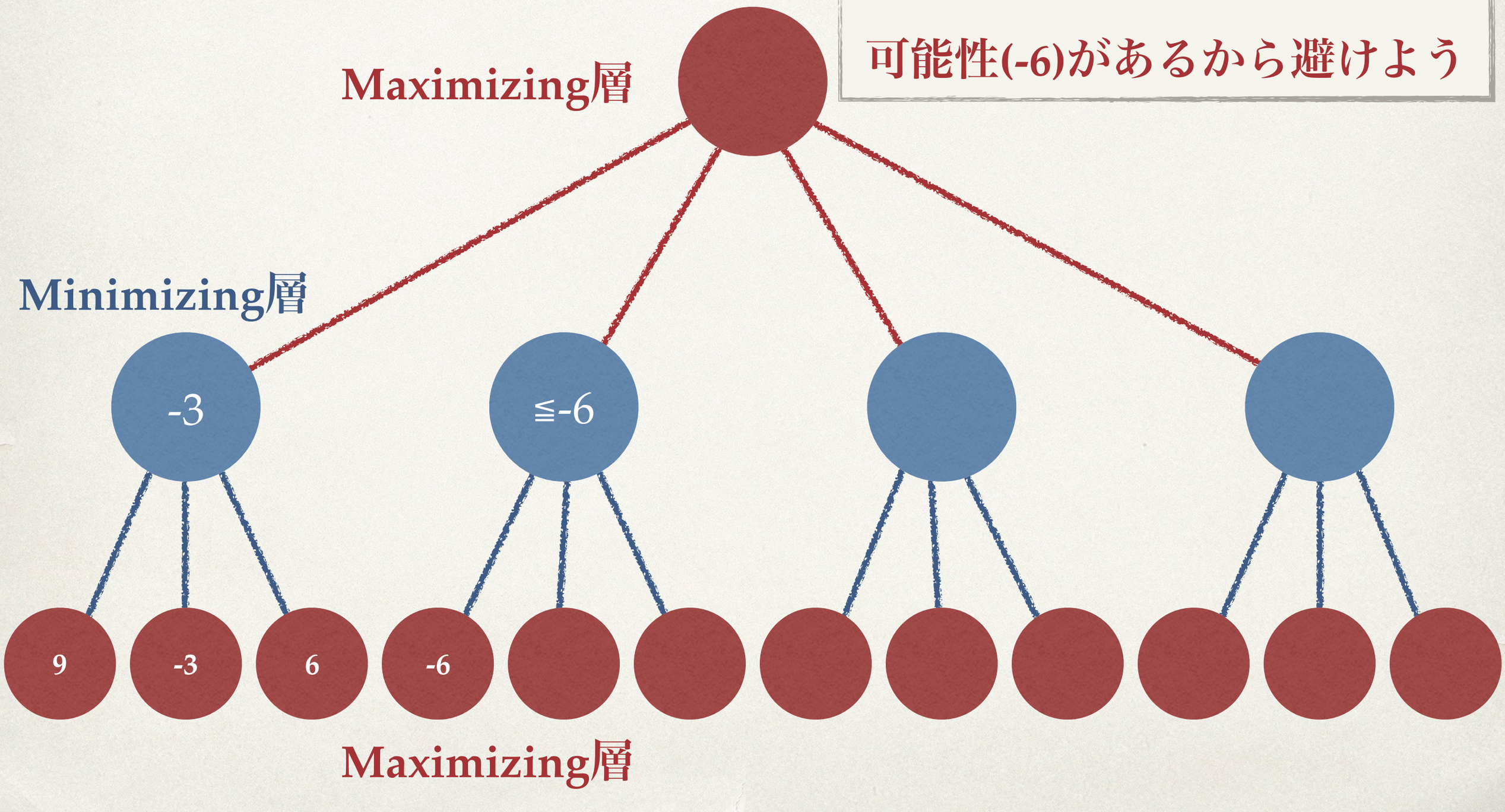
Minimax計算の無駄



Minimax計算の無駄

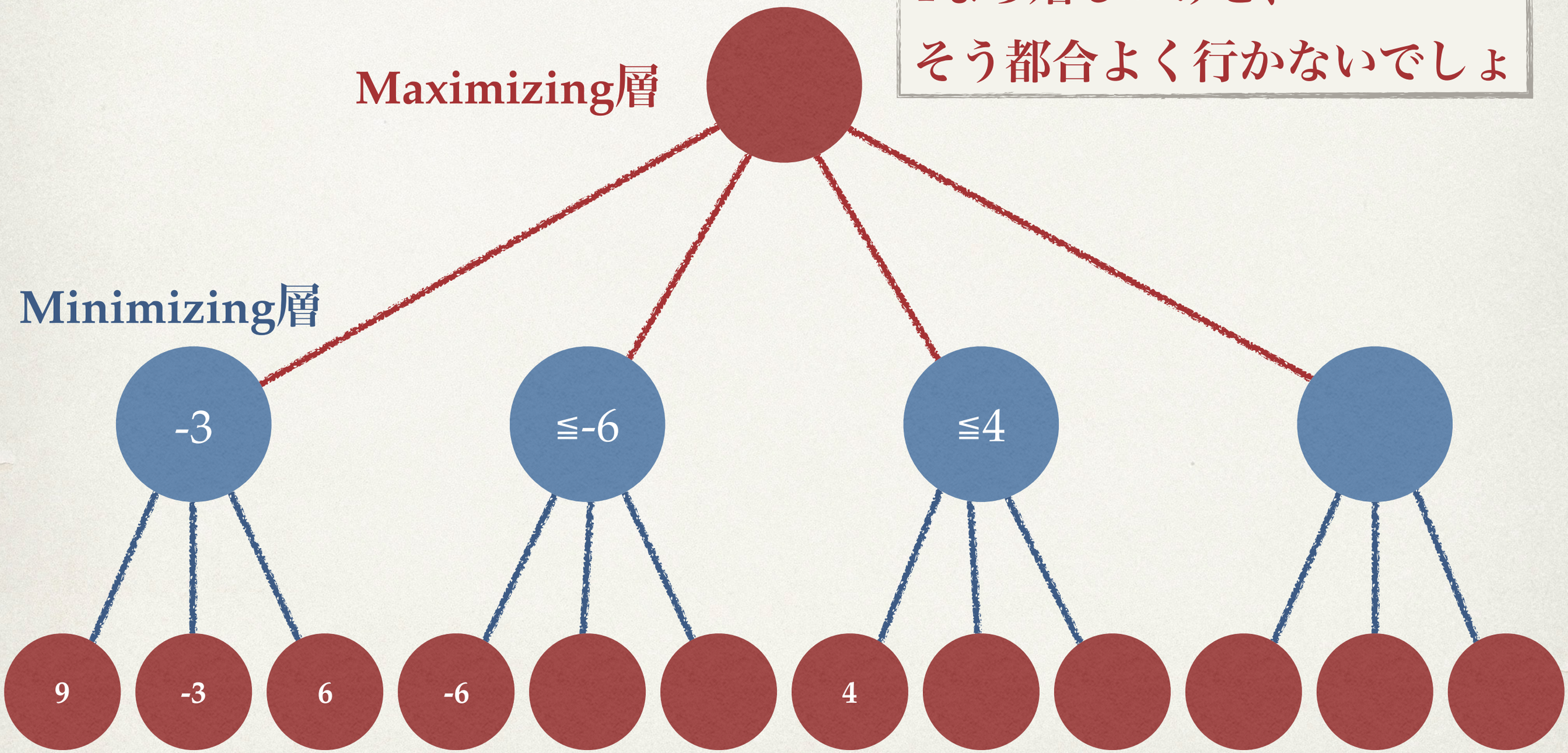
この手を打つとひどい目に会う

可能性(-6)があるから避けよう

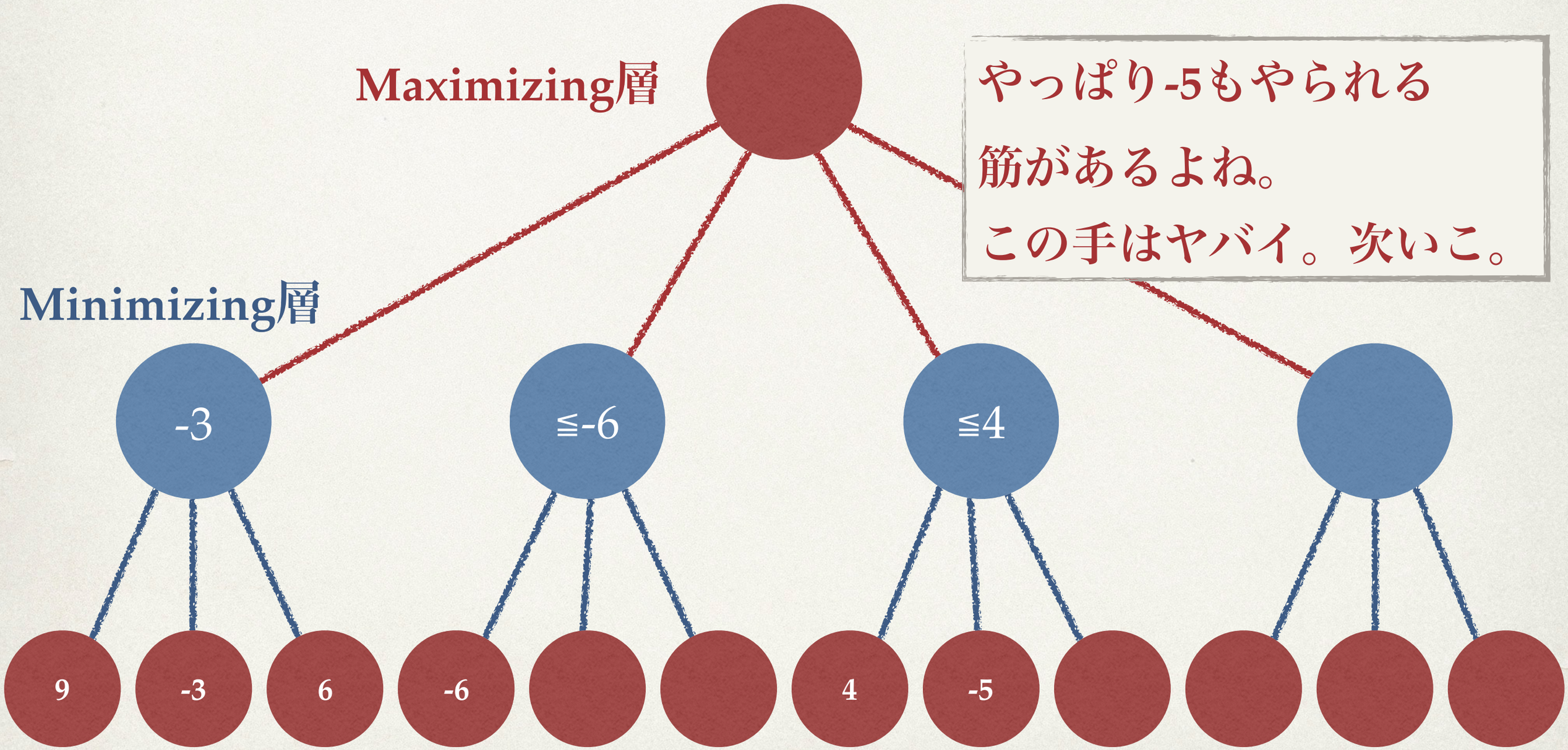


Minimax計算の無駄

4なら嬉しいけど、
そう都合よく行かないでしょ



Minimax計算の無駄

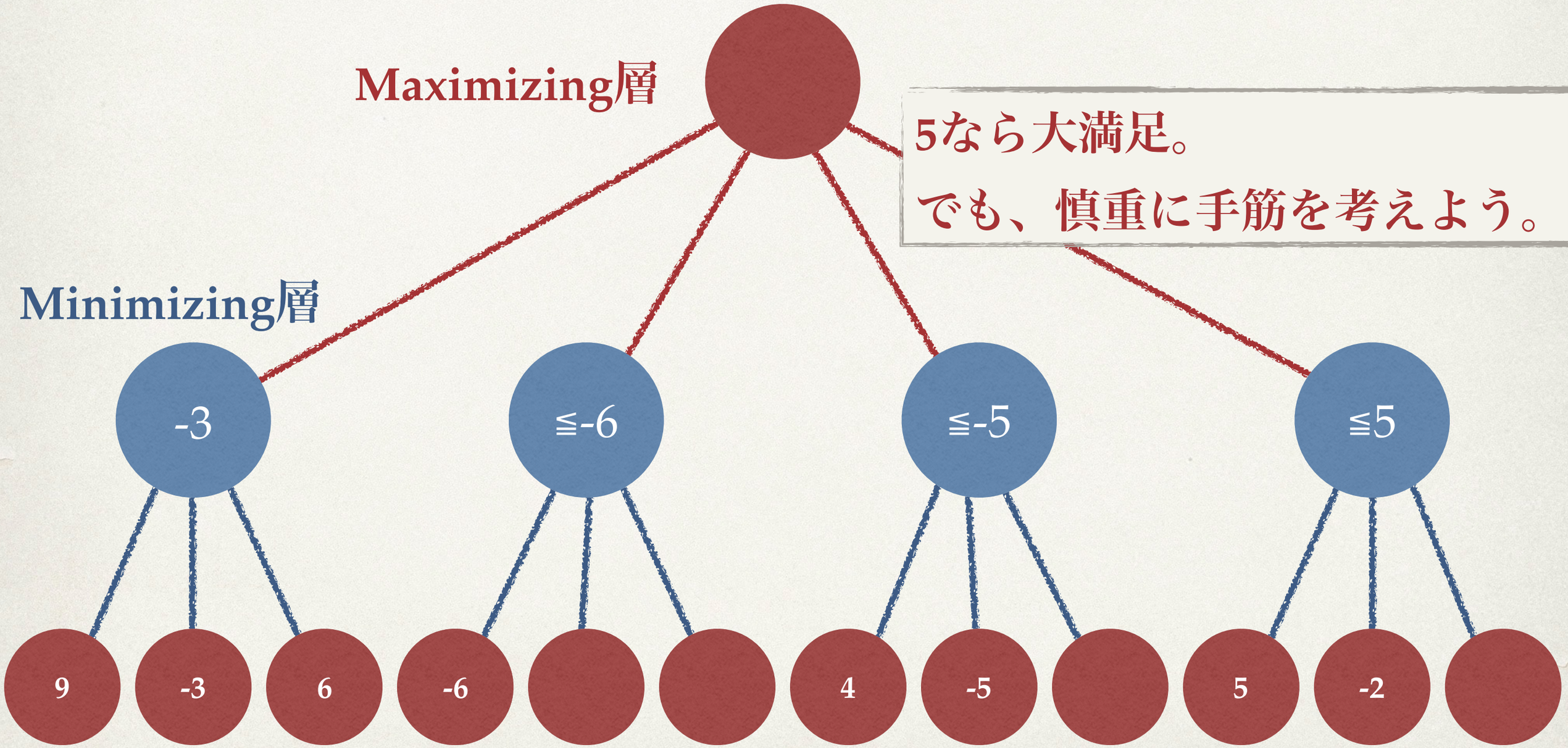


やっぱり-5もやられる

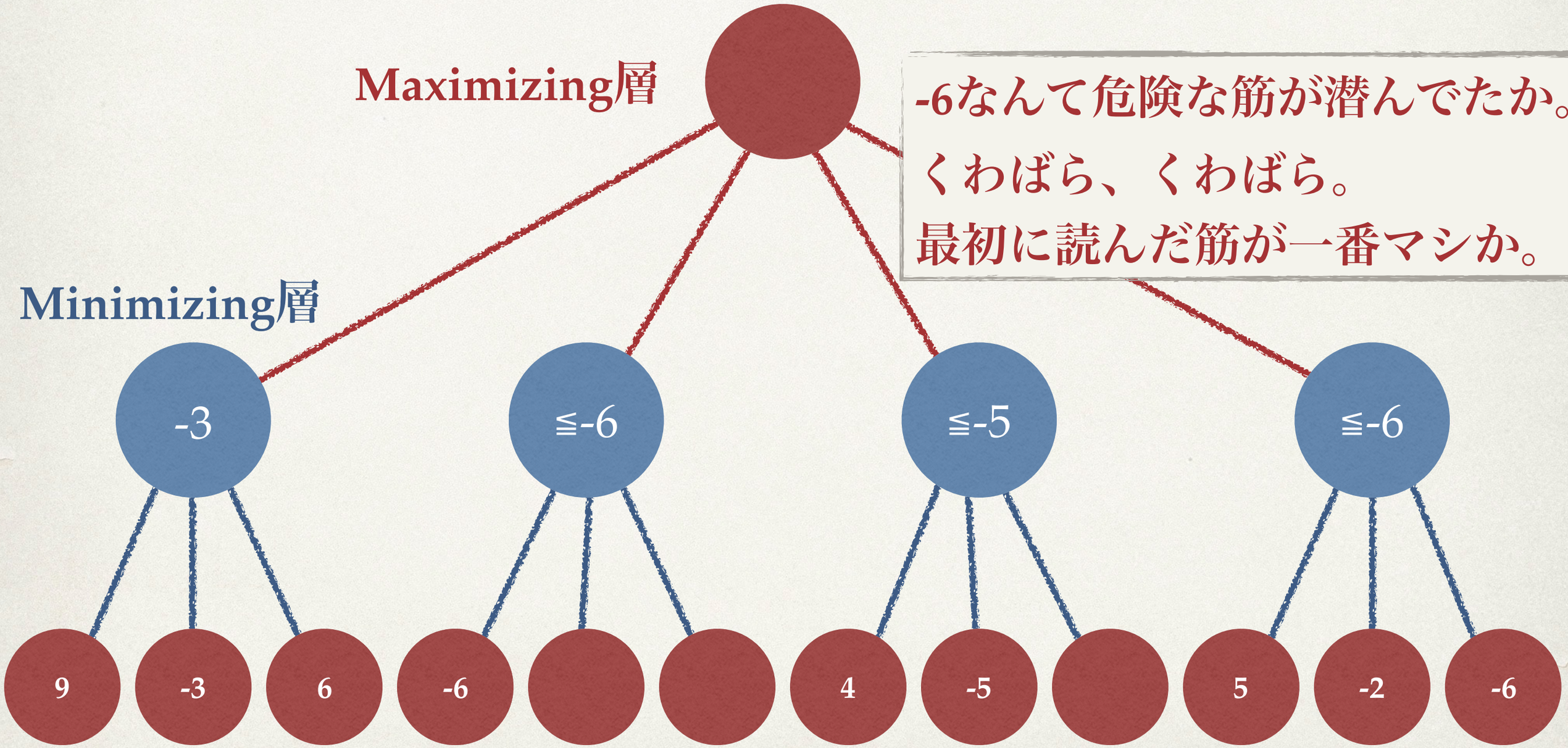
筋があるよね。

この手はヤバイ。次いこ。

Minimax計算の無駄



Minimax計算の無駄



-6なんて危険な筋が潜んでたか。

くわばら、くわばら。

最初に読んだ筋が一番マシか。

AlphaBeta法

- AlphaBeta(node, depth, α, β, player):
  if depth = 0 or node が葉: value(node)

  player match {
  case Max =>
  var v = -∞; var alpha = α
  for (child <- node.children):
  v = max(value, AlphaBeta(child, depth-1, alpha, β, Min)
  alpha = max(alpha, v)
  if alpha ≥ β: **return** v // 打ちきり = 枝刈り

  case Min => // Max と双対(-∞ → ∞、α → β、max → min)

強化学習

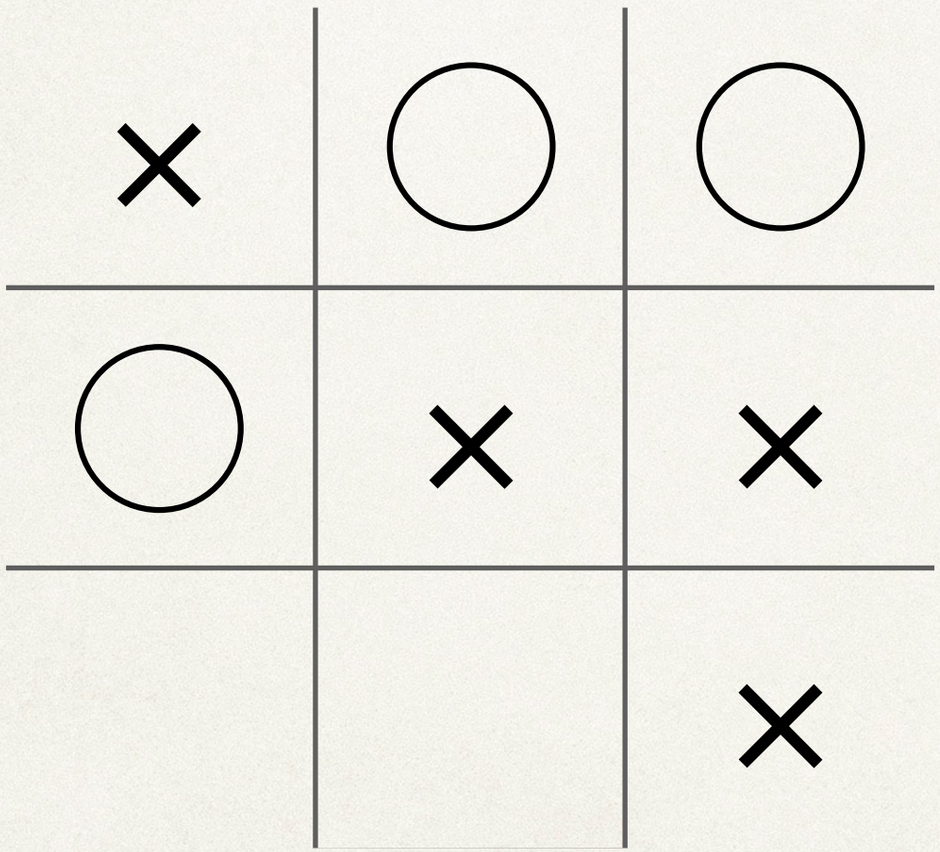
強化学習 (reinforcement learning)

- 試行錯誤から学んでいく学習方法
- 学習の枠組み
  - ポリシー: 与えられた局面下での行為選択
  - 報酬関数: 各局面の好ましさ
  - 価値関数: 長期的視点からの各局面の好ましさ

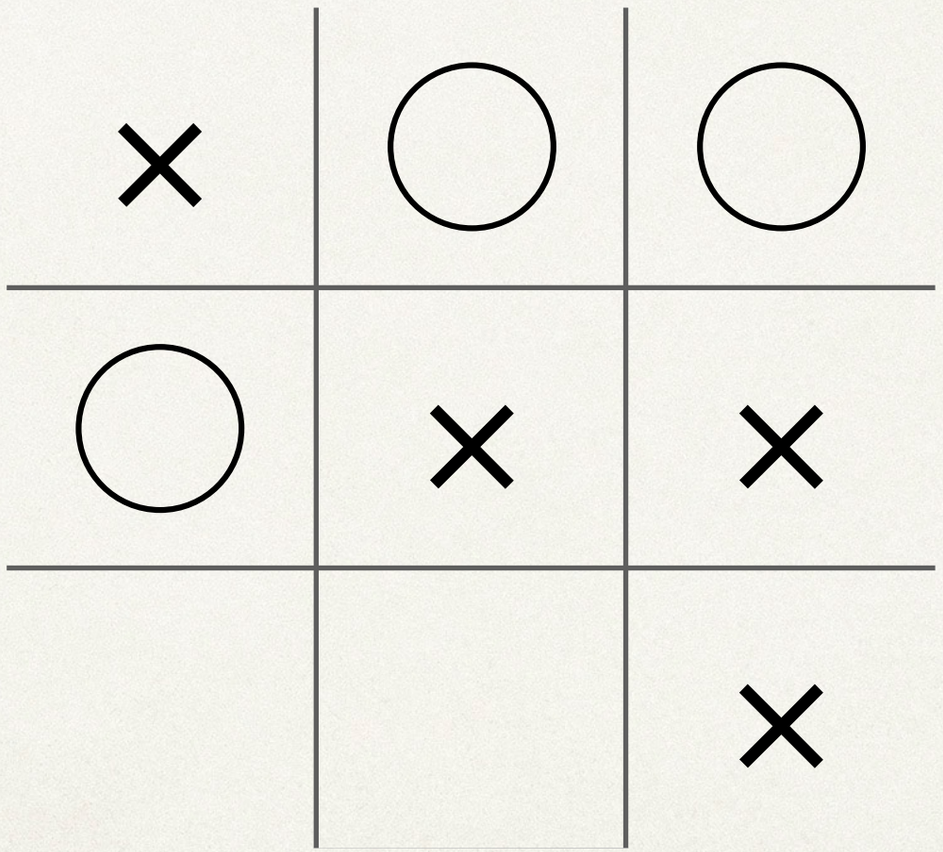
学習のジレンマ

- ポリシー
- 大きな報酬を得る方法を学んだら、それに従いたい
- でも、すでに学んだポリシーを逸脱してより大きい報酬を得られるかもしれない

○×で考える



○×で考える



初期価値関数

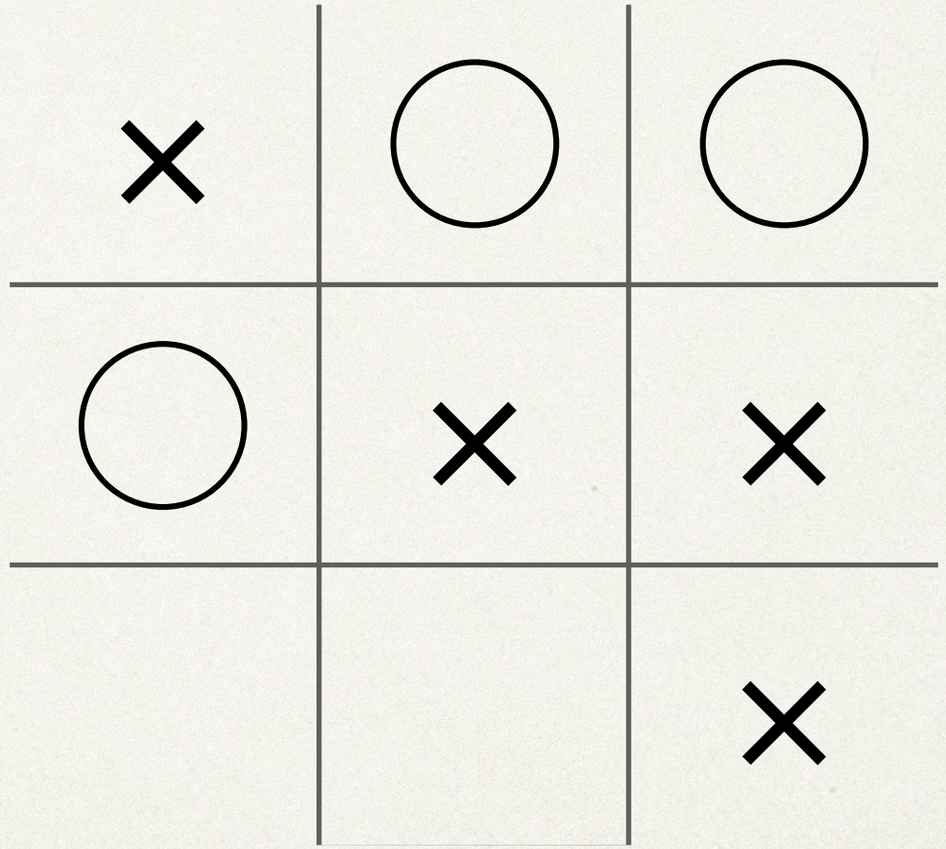
- 価値関数(局面) =

  この局面で:

  ×が勝ち => 1

  ○が勝ち => 0

  その他 => 0.5



ポリシー

- 次の行為の選択にあたって、

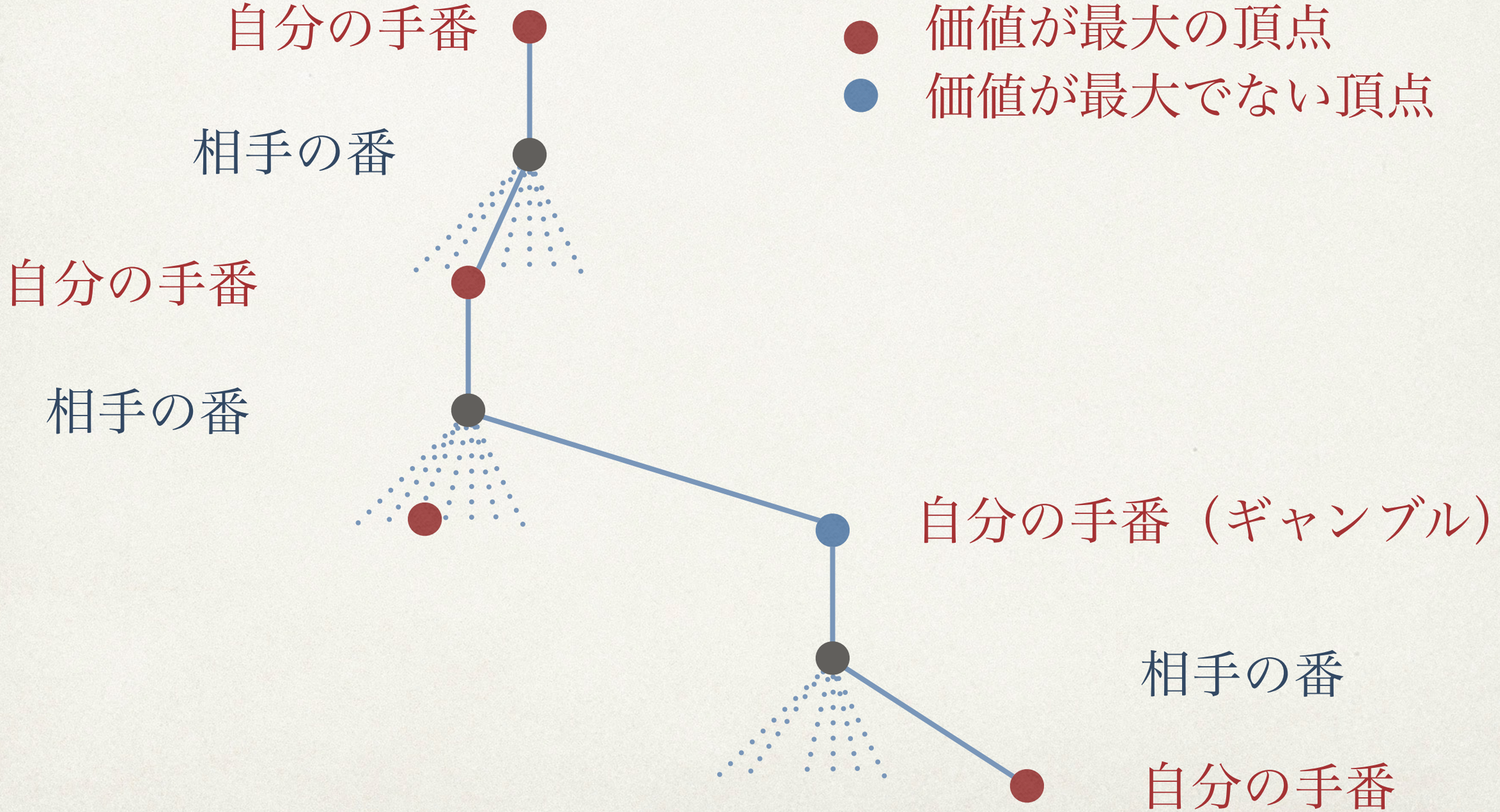
- 基本的には貪欲な選択

  すでに学んだ知識に沿って行動

- ときどき探索的な選択

  失敗しそうな冒険の旅（無作為な選択）

強化学習の概要



- 価値が最大の頂点
- 価値が最大でない頂点

貪欲 vs ギャンブル

- p in [0, 1]: はかなり小さな確率値

- random() > p:

  $\max_{c \in s.\text{children}}$ (V[s]) となる c を選択

  else: s.children から無作為に選択

学習

- 局面 s の遷移先 s.children から s' を選択したとき、状況が改善・悪化を価値関数に反映したい。

- $V[s] \mathrel{+}= \alpha[V[s] - V[s']]$
  α: step wise parameter

- 学習の進展とともに α を低減することで学習を収束させる

- α を0に下げなければ、相手が戦略を変更したときにも対応できる

価値関数の表現

- Tic-Tac-Toe の学習空間の大きさ： $3^9 < 20{,}000$ なら V: 配列
- もう少し、学習空間が大きい場合 → V: Hash表
- もっと学習空間が大きい場合
  - Vの Neural-Network を用いたモデル化

    (G. Tesauro 1994) TD-Gammon, a self-teaching backgammon program, achieves master-level play, Neural computation 6(2): 215-219. – 10^{20} 状態 (オセロは10^{28})

学習の仕方

- 人間とのプレイで鍛える
- 自分自身と対戦

気になっていること

- **ビットボード表現**（高効率な表現）
  - オセロの8x8のマス: 無 | 黒 | 白
  - 64bit整数 × 3 (empty, black, white)
  - 3 words, 24 bytes しか消費しない（2 words でもいいかも）
  - Hash値は3つの整数の排他的論理和で簡単

    hash = empty ^ black ^ white

気になっていること

- **ビットボード表現**（超絶的反転計算？）

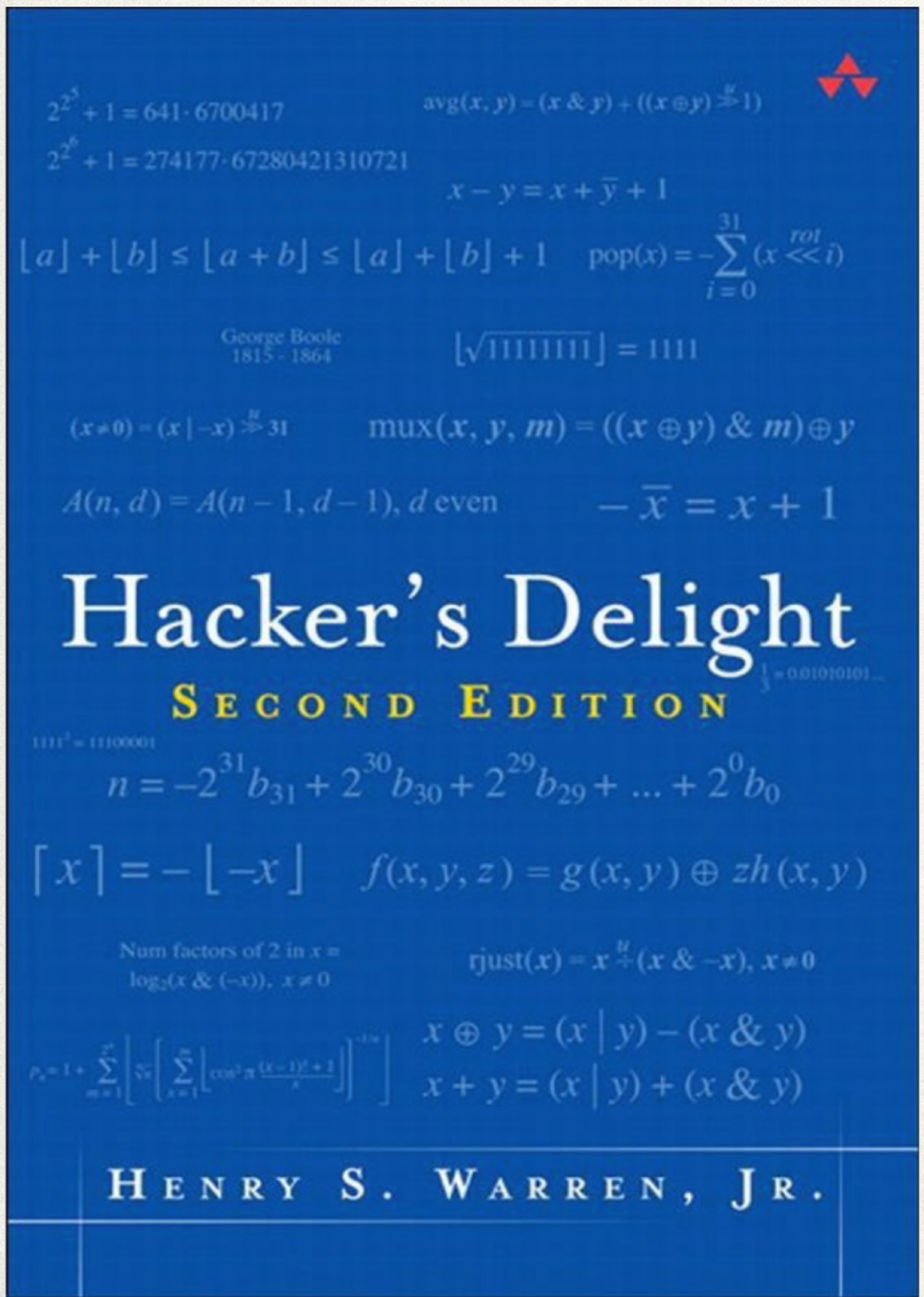
Use the following formula to turn on the rightmost 0-bit in a word, producing all 1's if none (e.g., 10100111 ⇒ 10101111):

$$x \mid (x + 1)$$

Use the following formula to turn off the trailing 1's in a word, producing x if none (e.g., 10100111 ⇒ 10100000):

$$x \,\&\, (x + 1)$$

Hacker's Delight 2nd ed



- 日本語版は「ハッカーの楽しみ、SiB access」