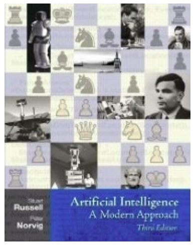
Outline

- Games of Chance
  - Expectiminimax
  - Monte-Carlo Evaluation
- Simulation Search
  - Monte-Carlo search
  - Bandits
  - UCT search
- Games of Imperfect Information

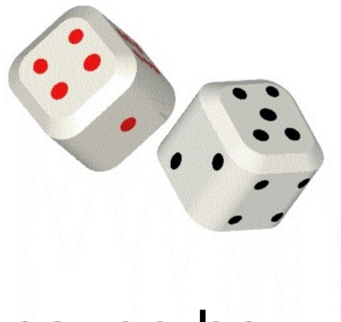


Many slides based on Russell & Norvig's slides
Artificial Intelligence: A Modern Approach

Additional slides on MCTS in Go by David Silver and DeepMind

Games of Chance

- Many games combine skill and chance
  - i.e., they contain a random element like the roll of dice
- This brings us closer to real-life
  - in real-life we often encounter unforeseen situations
- Examples
  - Backgammon, Monopoly, ...

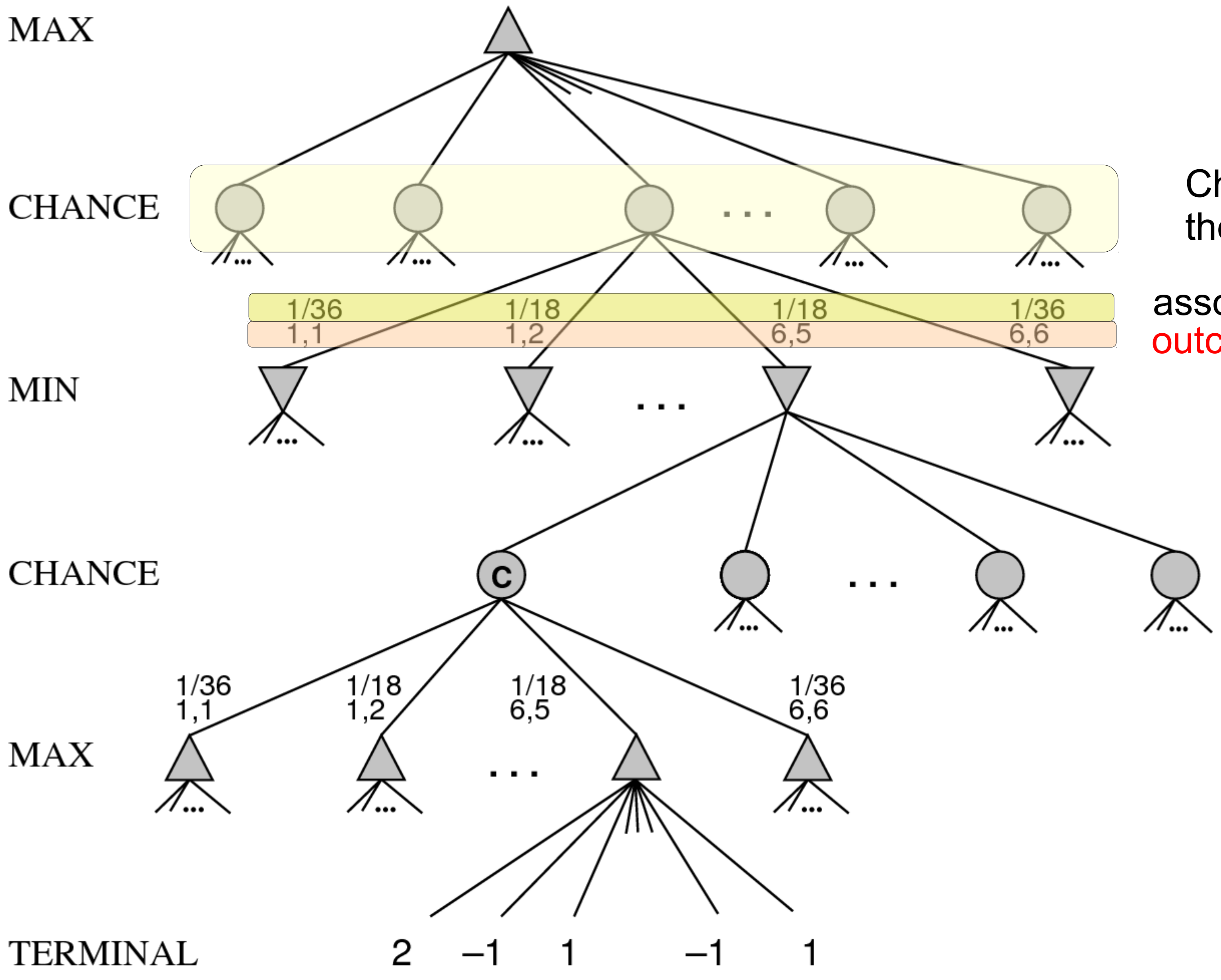


- Problem
  - Player MAX cannot directly maximize his gain because he does not know what MIN's legal actions will be
    - MIN makes a roll of the dice *after* MAX has completed his ply
  - and vice versa (MIN cannot minimize)

  → Minimax or Alpha-Beta no longer applicable

→ Standard game trees are extended with chance nodes

Game-Tree with Chance Nodes



Chance nodes for the roll of two dice

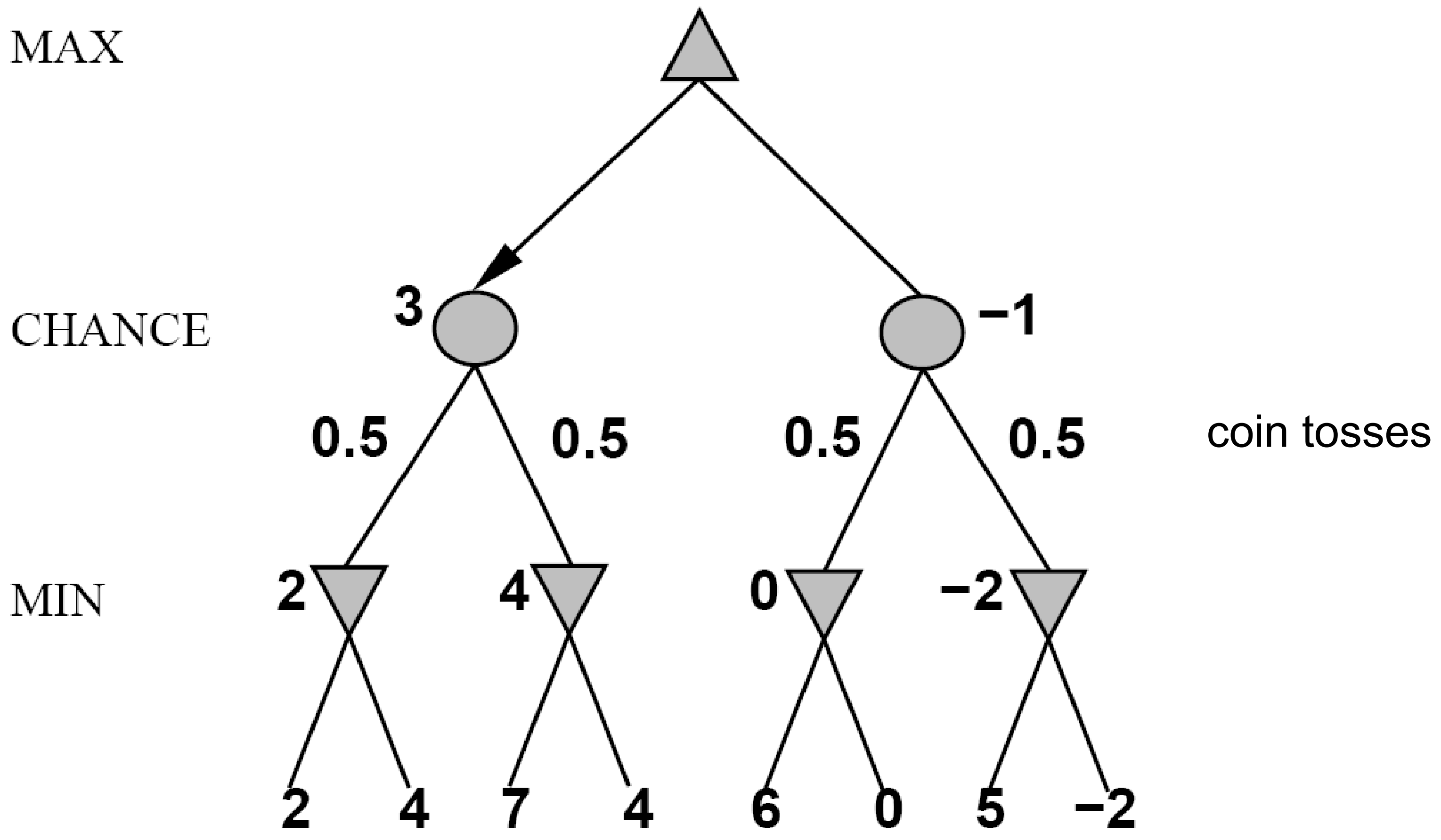
associated probability
outcome of the dice roll

Optimal Strategy with Chance Nodes

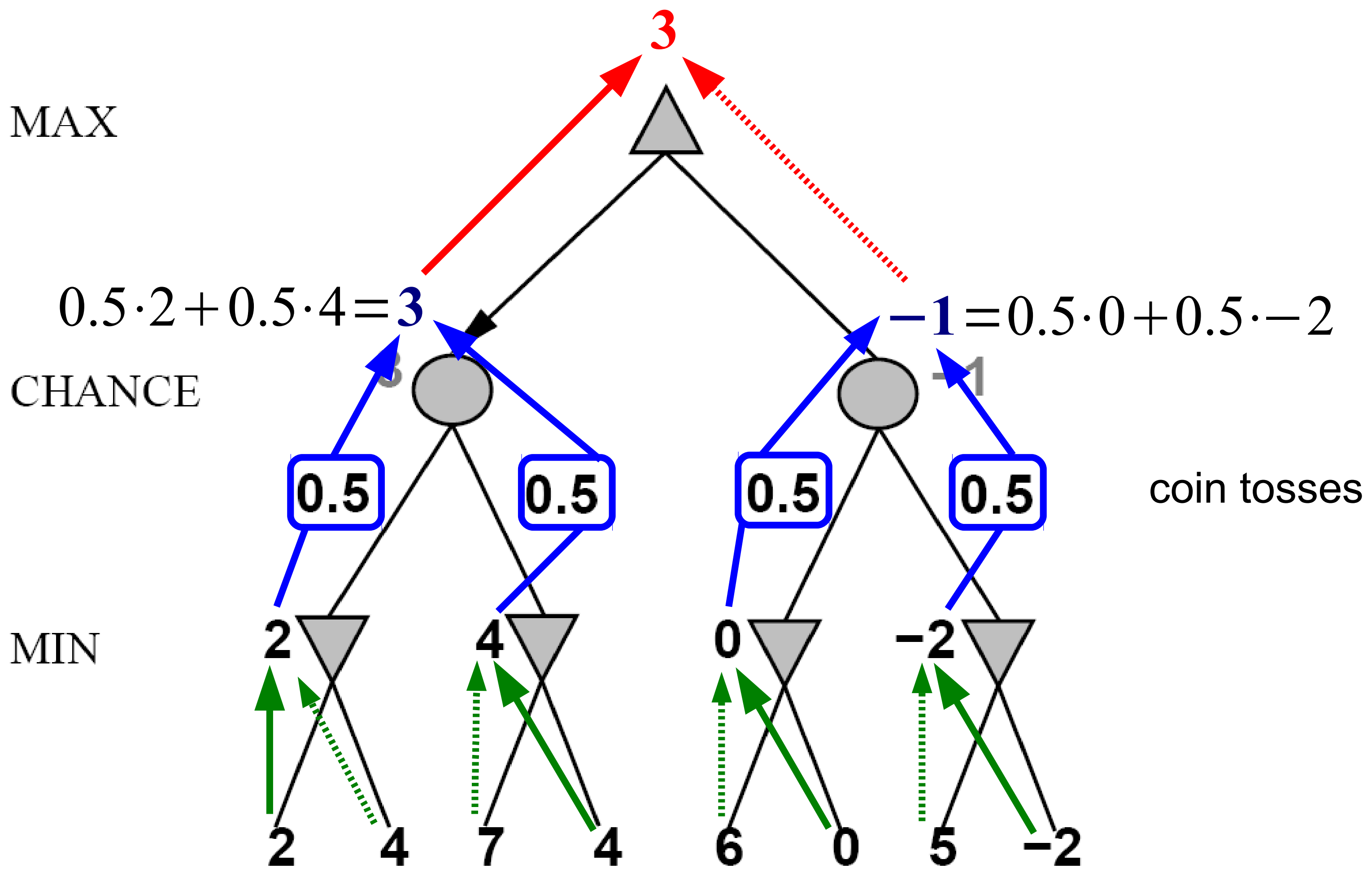
- MAX wants to play the move that maximizes his chances of winning
- Problem:
  - the exact outcome of a MAX-node cannot be computed because each MAX-node is followed by a chance node
  - analogously for MIN-nodes
- Expected Minimax value
  - compute the expected value of the outcome at each chance node

$$\text{E XPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in SUCCESSORS(n)} \text{E XPECTIMINIMAX} & \text{if } n \text{ is a MAX node} \\ \min_{s \in SUCCESSORS(n)} \text{E XPECTIMINIMAX} & \text{if } n \text{ is a MIN node} \\ \sum\limits_{s \in SUCCESSORS(n)} P(s) \cdot \text{E XPECTIMINIMAX} & \text{if } n \text{ is a chance node} \end{cases}$$

Example



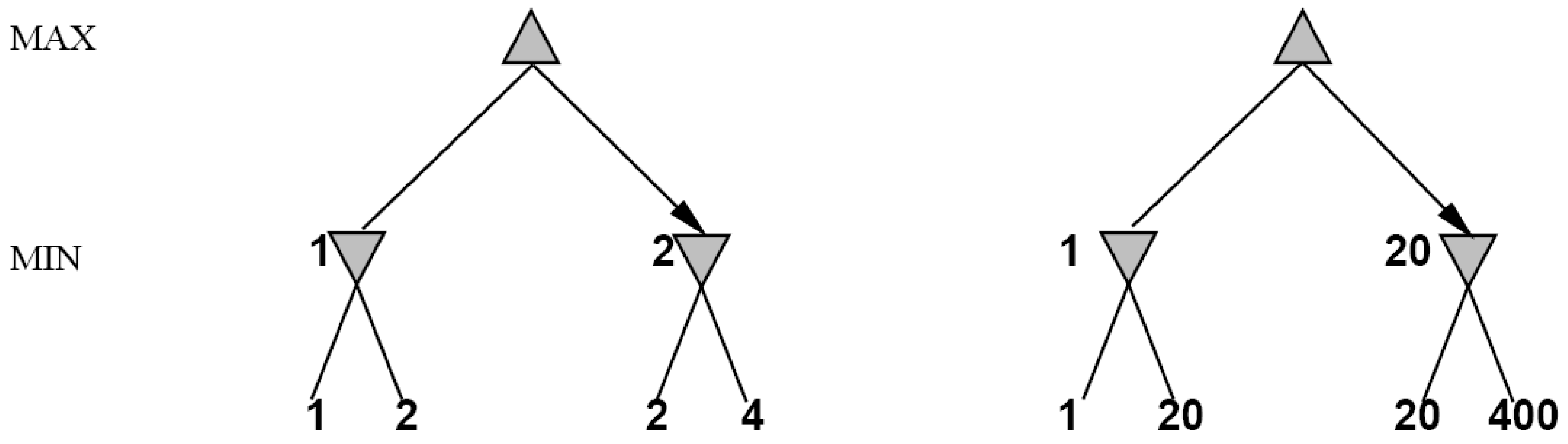
Example


3
MAX
CHANCE
MIN
$0.5 \cdot 2 + 0.5 \cdot 4 = 3$
$-1 = 0.5 \cdot 0 + 0.5 \cdot -2$
0.5
0.5
0.5
0.5
coin tosses
2
4
0
−2
2
4
7
4
6
0
5
−2

EXPECTIMINIMAX gives perfect play, like MINIMAX

Re-Scaling of Evaluation Functions

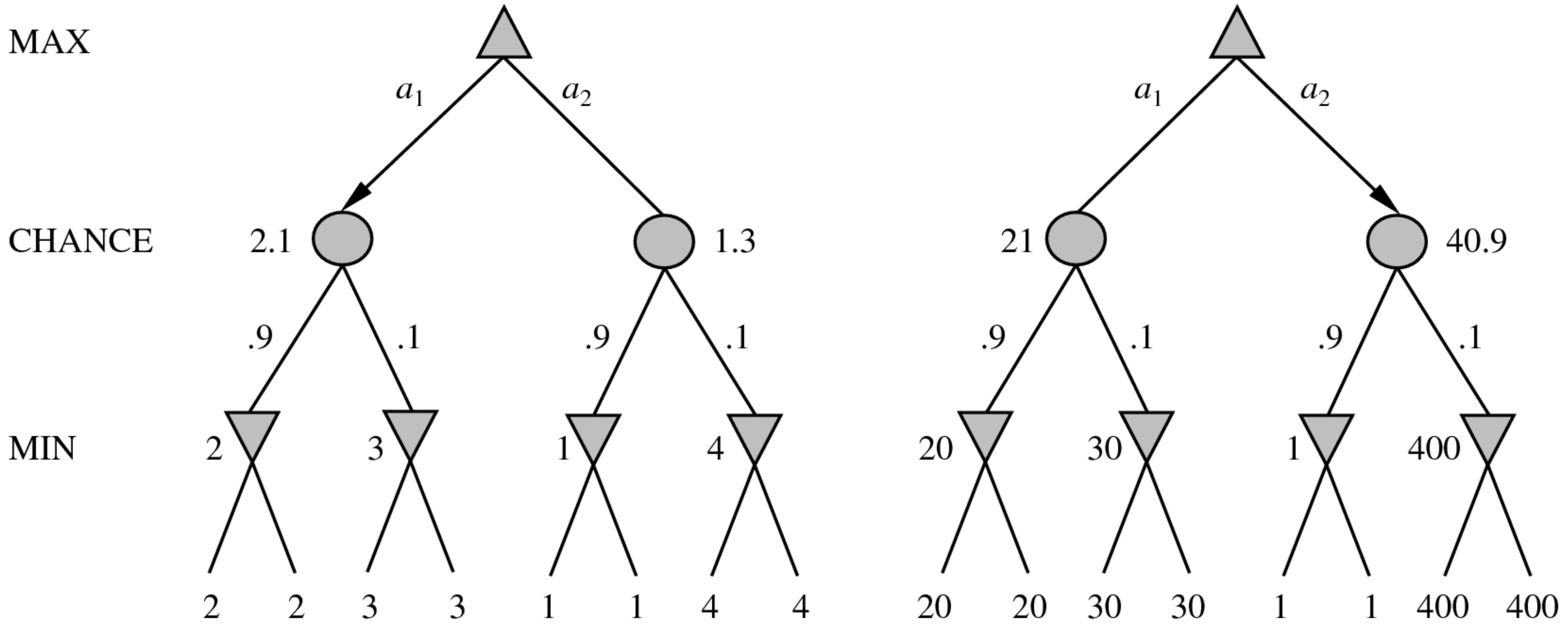
- Minimax:
  - no problem, as long as values are ordered in the same way (monotonic transformations)



MAX plays the same move in both cases

Re-Scaling of Evaluation Functions

- Expectiminimax:
  - Monotonic transformations may change the result



  - only positive *linear* transformations preserve behavior

→ EVAL should be proportional to the expected outcome!

Nondeterministic Games in Practice

- Complexity
  - In addition to the branching factor, the number of different outcomes c adds at each chance node to the complexity
  - Total complexity is $O(b^m c^m)$
    $\rightarrow$ deep look-ahead not feasible

$c = 2$	for coin flip
$c = 6$	for rolling one die
$c = 21$	for rolling two dice

- prob. of reaching a given node shrinks with increasing depth
  - forming plans is not that important
    $\rightarrow$ deep look-ahead is also not that valuable
  - Example:
    - TD-Gammon uses only 2-ply look-ahead + very good EVAL
- Alpha-Beta Pruning is also possible (but less effective)
  - at MIN and MAX nodes as usual
  - at chance nodes, expected values can be bounded before all nodes have been searched if the value range is bounded

Approximating Expected Values

- If exact probabilities for the outcomes at the chance nodes are not known, values can be sampled
  - we replace the computation of the expected value with the average outcome over a (large) number of N random games starting in the current node n

$$\sum_{s \in SUCCESSORS(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) \approx \frac{1}{N} \sum_{1..N} \text{Utility}(\text{RandomGame}(n))$$

  - the higher N, the better the approximation

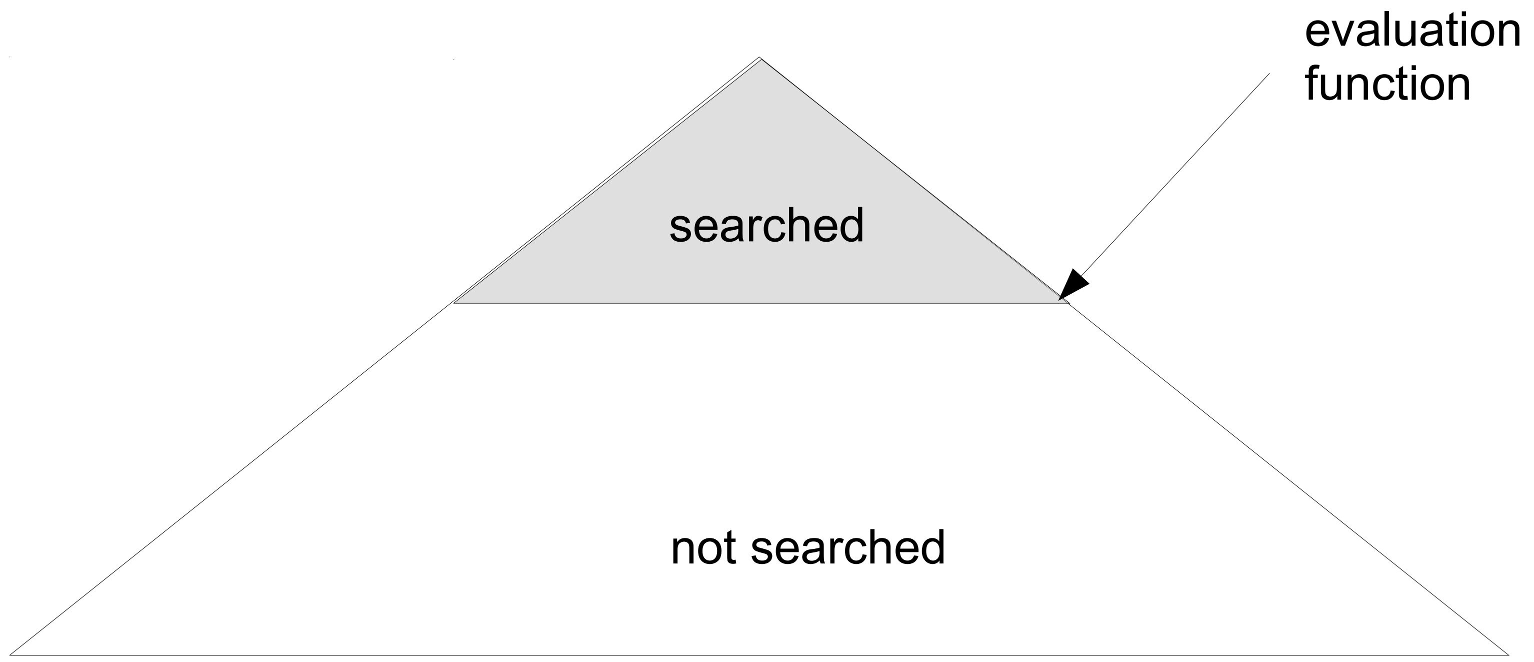
Monte-Carlo Sampling

Roll-outs

- Technique that has been used in Backgammon for estimating the outcome of a position
- Play the position many times, each time with a different (random) dice roll
- used to be done by hand, is now done by compute

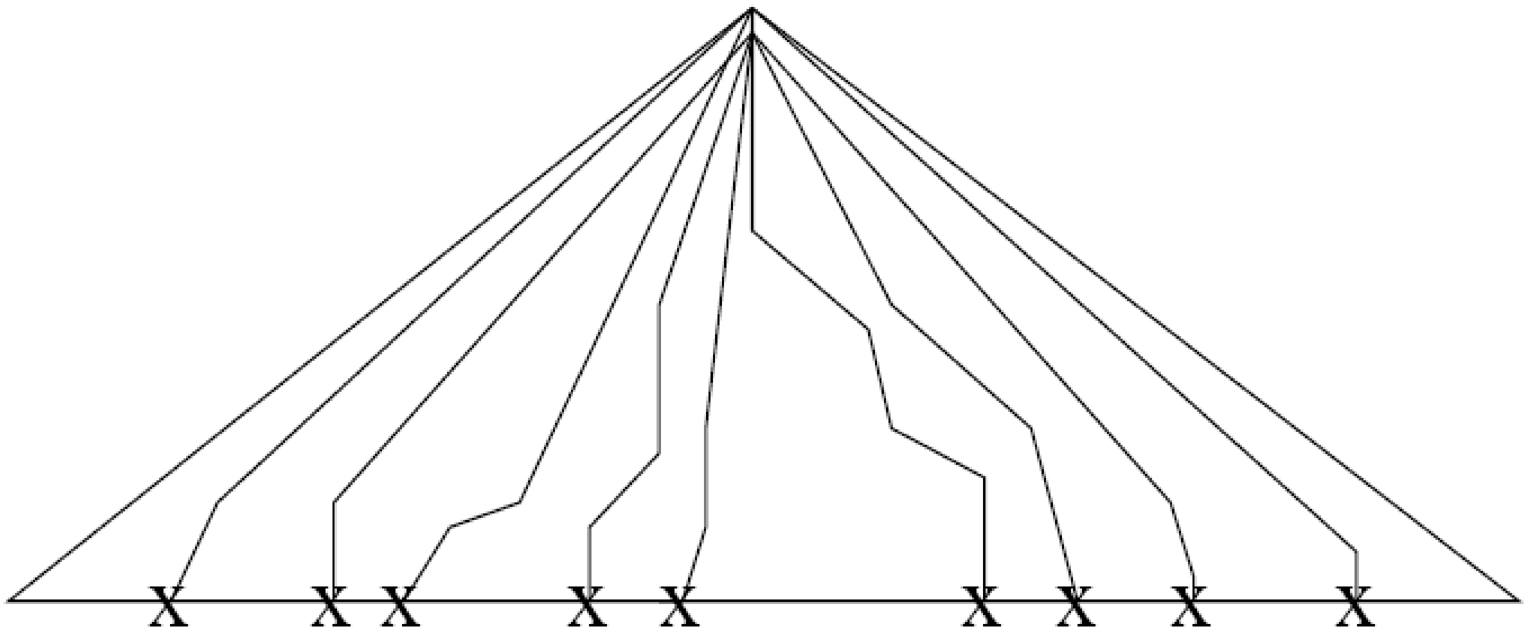
Simulation Search – Key Idea

- The complete tree is not searchable
  - thus minimax/alpha-beta limit the depth of the search tree
    - search all variations to a certain depth



Simulation Search – Key Idea

- The complete tree is not searchable
  - thus minimax/alpha-beta limit the depth of the search tree
    - search all variations to a certain depth
  - alternatively, we can limit the breadth of the search tree
    - sample some lines to the full depth



Simulation Search

- Algorithm Sketch:
  - estimate the expected value of each move by counting the number of wins in a series of complete games
  - at each chance node select one of the options at random (according to the probabilities)
  - at MAX and MIN nodes make moves (e.g., guided by a fast evaluation function)
- Examples:
  - roll-out analysis in Backgammon
    - play a large number of games from the same position
    - each game has different dice rolls
  - in Scrabble:
    - different draws of the remaining tiles from the bag
  - in card games (e.g., GIB in Bridge)
    - different distributions of the opponents' cards

Simulation Search

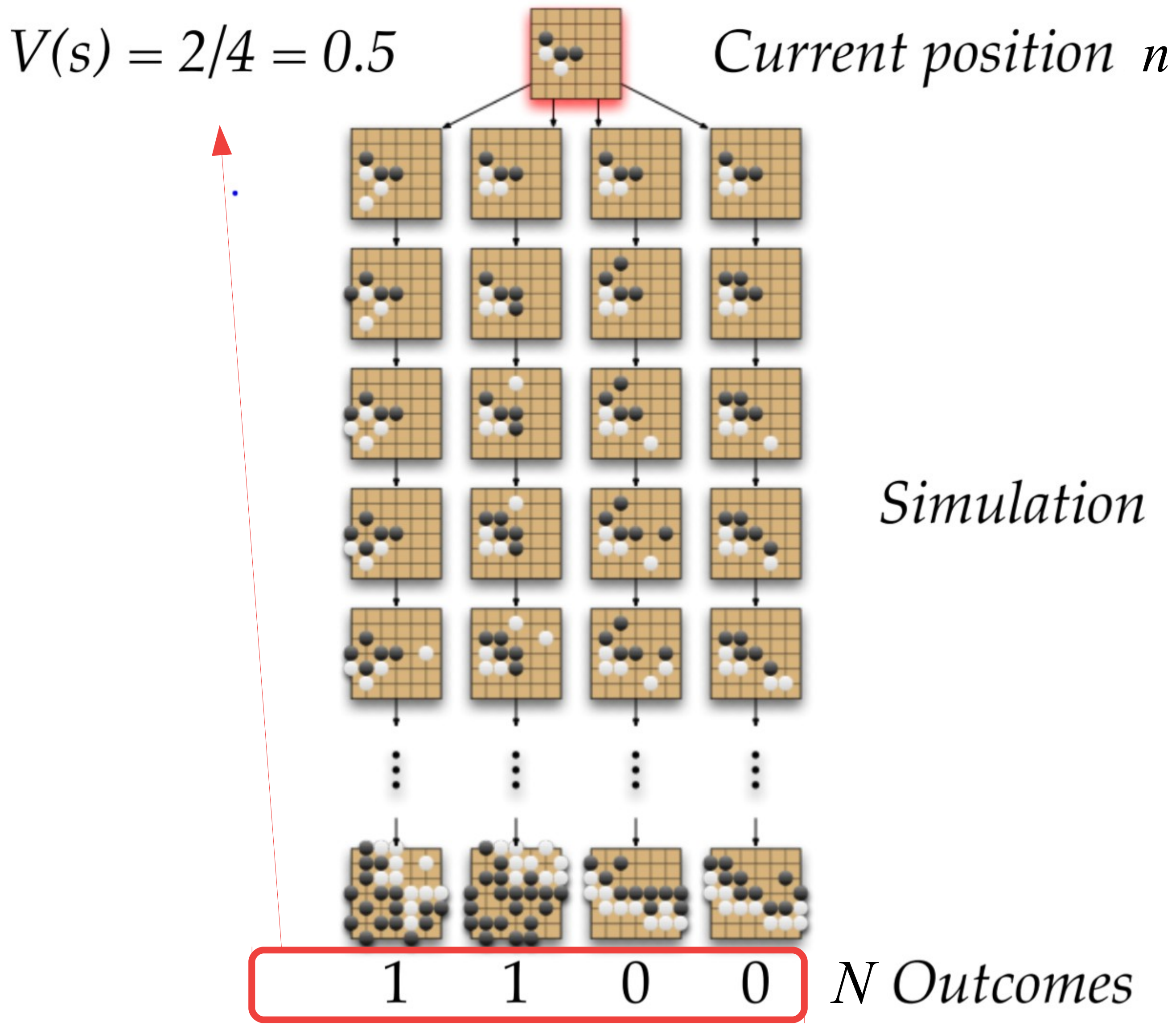
- Algorithm Sketch:
  - estimate the expected value of each move by counting the number of wins in a series of complete games
  - at each chance node select one of the options at random (according to the probabilities)
  - at MAX and MIN nodes make moves (e.g., guided by a fast evaluation function)
- Properties:
  - We need a fast algorithm for making the decisions at each MAX and each MIN node
    - the program plays both sides, of course
  - Often works well even if the program is not that strong

    → fast is possible
  - Easily parallelizable

Monte-Carlo Search

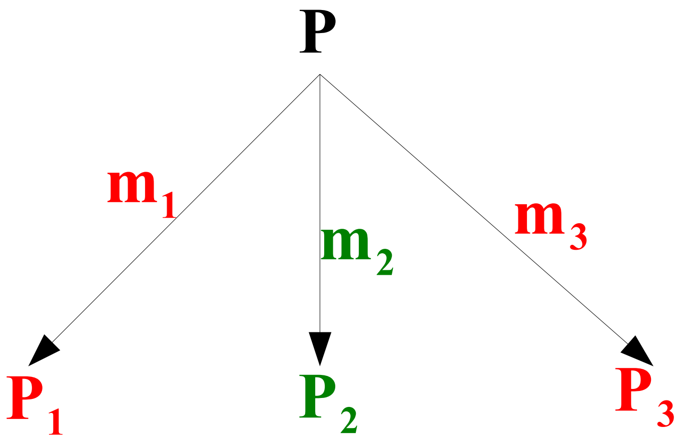
- Extreme case of Simulation search:
  - play a large number of games where both players make their moves randomly
  - average the scores of these games
  - make the move that has the highest average score
- Can also be used in deterministic games
  - Has been used with some success in Go
    - e.g., Bruegmann 1993

Monte-Carlo Sampling in Go



Sampling the best move

- Basic Setting
  - The player $\mathbf{P}$ is faced with a choice of moves $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$
  - leading to positions $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$
  - the move that leads to the best position is preferred



Learning to make this choice may be:

Utility-based:

$$u(\mathbf{P}_1)=u_1$$
$$u(\mathbf{P}_2)=u_2$$
$$u(\mathbf{P}_3)=u_3$$

Preference-based:

$$\mathbf{P}_2 \succ \mathbf{P}_1$$
$$\mathbf{P}_2 \succ \mathbf{P}_3$$

classical approaches using reinforcement learning (e.g., TD-Gammon)

currrent work in our group

Multi-Armed Bandit Problems

(Robbins 1952)

One-armed bandit:

- One action (pull the lever)
- Fixed long-term expected reward (or loss)

Multi-Armed Bandit Problems

(Robbins 1952)

One-armed bandit:

- One action (pull the lever)
- Fixed long-term expected reward (or loss)

Multi-armed bandit

Multi-Armed Bandit Problems

(Robbins 1952)

One-armed bandit:

- One action (pull the lever)
- Fixed long-term expected reward (or loss)

Multi-armed bandit

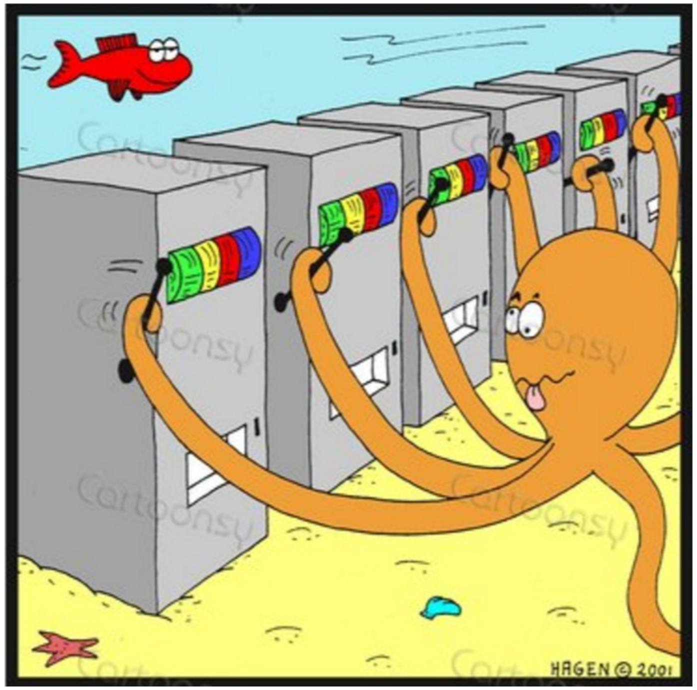
- k actions (moves) $\mathbf{m}_\mathrm{i}$
- associated with different fixed but unknown long-term expected rewards / utilities u_i

$u_1 = 150\,\$$ $u_2 = 100\,\$$ $u_3 = 300\,\$$ $u_4 = 200\,\$$

Learning Problem

Task: **Find a strategy (policy) to maximize your gain**

- Find the arm (move) with the highest long-term utility from observations of immediate utilities
- with as little regret as possible
- and then keep playing it



Multi-Armed Bandits – Learning Problem

Which arm should I play next (at time t)?

$u_1 = ?$

$u_2 = ?$

$u_3 = ?$

$u_4 = ?$

Winning $w_i^{(t)} = 50\,\$$

And which arm should I play next (at time $t+1$)?

Upper Confidence Bound (UCB) Algorithm

(Auer et al. 2002)

Problem: **Exploitation vs. Exploration**

- *Exploitation:* Pull the best arm in order to maximize your earnings
- *Exploration:* Try if other arms are more promising

UCB algorithm

- Always Play the Arm with the highest Upper-Confidence Bound

$$UCB1 = \frac{\sum_t w_i^{(t)}}{n_i} + \alpha \sqrt{\frac{\log(N)}{n_i}}$$

$\sum_t w_i^{(t)}$	total win for $\mathbf{m}_i$
n_i	#games with $\mathbf{m}_i$
$N = \sum_i n_i$	total #games

Exploitation Term

Empirical estimate of r_i from n_i observed earnings $w_i^{(t)}$

Exploration Term

Inverse percentage of trials of arm i

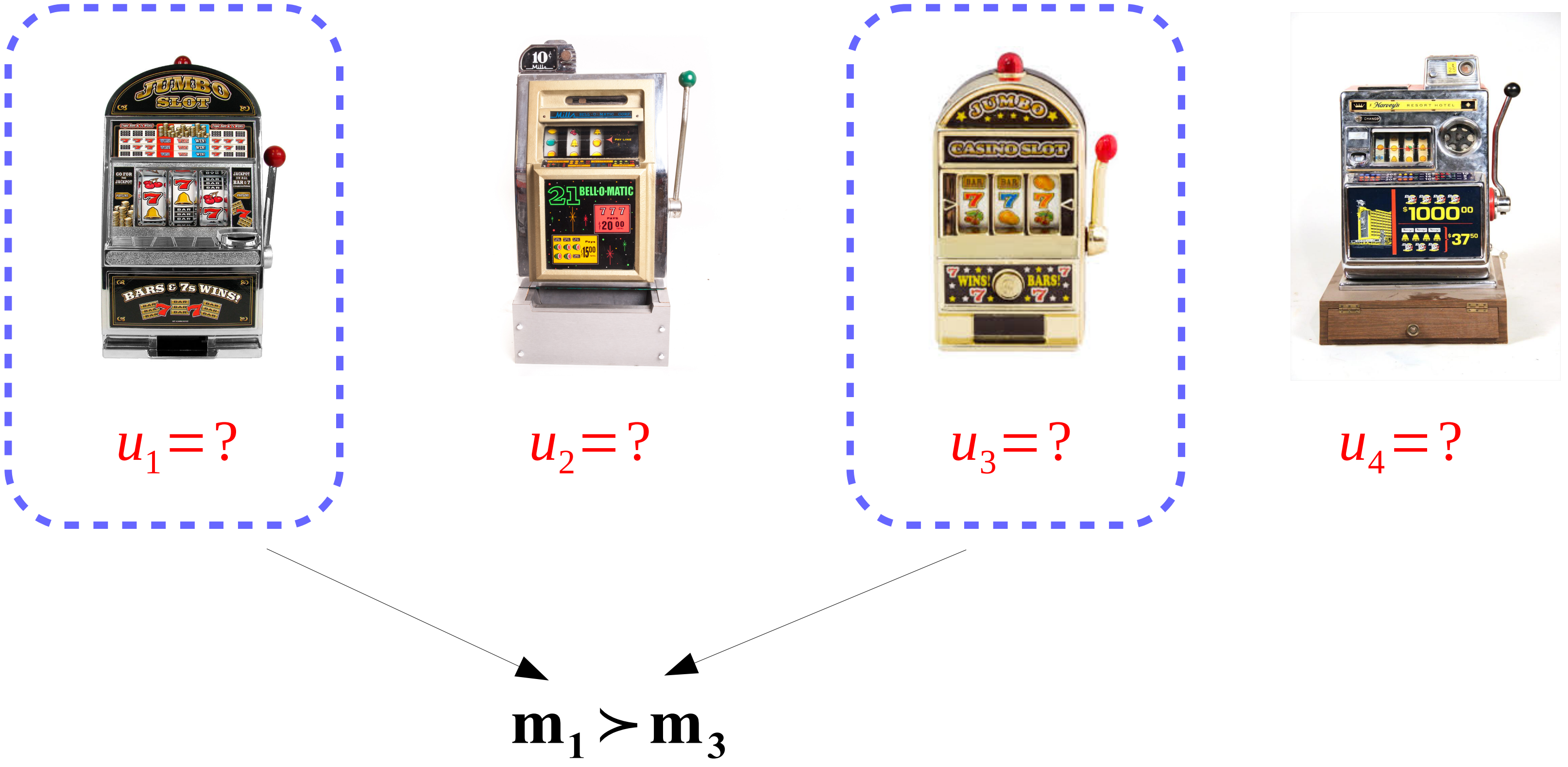
Dueling Bandits

(Yue et al. 2012)
(Busa-Fekete et al. 2014)

- One can also assume that **feedback** is not numeric but **preference-based**
  - The player pulls not one but two arms, and observes which one of them is better (will yield the higher long-term reward
- **Long-Term Goal** remains the same
  - Identify the machine which promises the highest average reward
  - Main difference is that the exact amount of winning cannot be estimated from the qualitative feedback
- **Applications** in areas where qualitative judgement is necessary
  - search result A is better than search result B
  - dish A tastes better than dish B
  - move A is better than move B

Dueling Bandits – Learning Problem

Which two arms should I play next (at time t)?



And which two arms should I play next (at time $t+1$)?

Relative UCB (slightly simplified)

(Zoghi et al. 2014)

Maintain a UCB-value for each action pair

$$u_{ij} = \frac{w_{ij}}{n_{ij}} + \alpha \sqrt{\frac{\log(N)}{n_{ij}}}$$

$w_{ij} = \sum_t w_{ij}^{(t)}$	#wins of $\mathbf{m}_i$ over $\mathbf{m}_j$
$n_{ij} = w_{ij} + w_{ji}$	#games $\mathbf{m}_i$ vs. $\mathbf{m}_j$
$N = \sum_{ij} n_{ij}$	total #games

1st Action Selection

- If exists, select an action $\mathbf{m}_i$ that dominates all others
- If not pick a random action $\mathbf{m}_i$

$$(\mathbf{m}_i | \forall j : u_{ij} \geq \tfrac{1}{2})$$

2nd Action Selection

- choose the strongest opponent $\mathbf{m}_j$ for the action $\mathbf{m}_i$

$$j = \arg\max_k u_{ki}$$

More realistic Games

Bandits are a very simple "game"

- only one game state
- fixed stochastic reward distribution independent of adversary

More realistic games require

- **Game States:** Most games have different states (positions) described with features, and possibly different move sets
  - utility will depend on the state and the features used for describing it
- **Delayed Reward:** Feedback will not be available right after the move played, but after a **sequence of moves** (often entire game)
- **Adversary:** Obtained feedback is not only for the own quality of play but also about the opponent's play
- **Search:** Look-ahead is often more important than good utility estimates

Monte-Carlo Tree Search

- Monte-Carlo Search can be integrated with conventional game-tree search algorithms

Key ideas:

- incrementally build up a search tree
- evaluate the leaf nodes via roll-outs
- evalaute promising moves more often than bad moves
  - but give bad moves also a chance

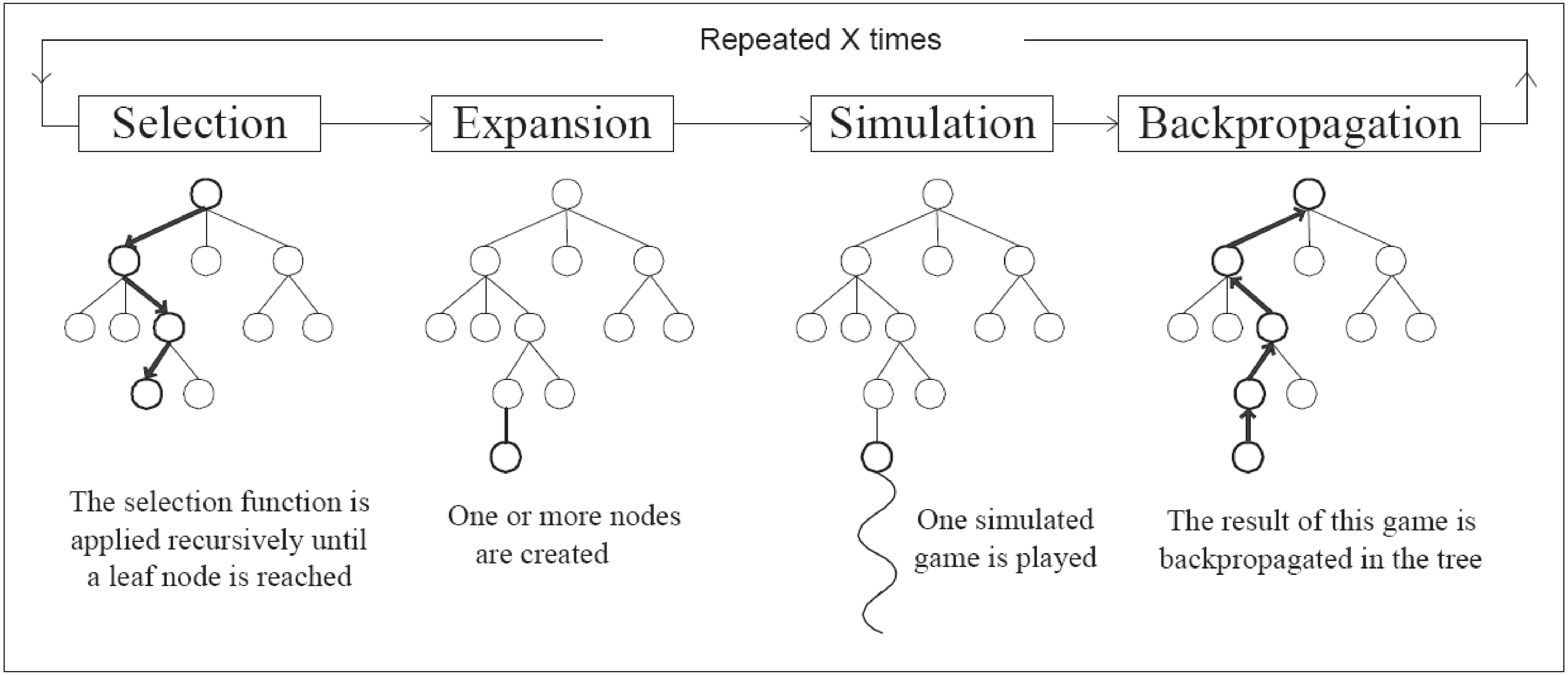
MCTS essentially uses two policies

- **Tree Policy:**
  - How is the tree traversed in order to find the best leaf to be expanded
- **Rollout Policy** (or **Default Policy**):
  - How are the moves in the roll-out games selected

Policy: A strategy for selecting actions (→ Reinforcement Learning)

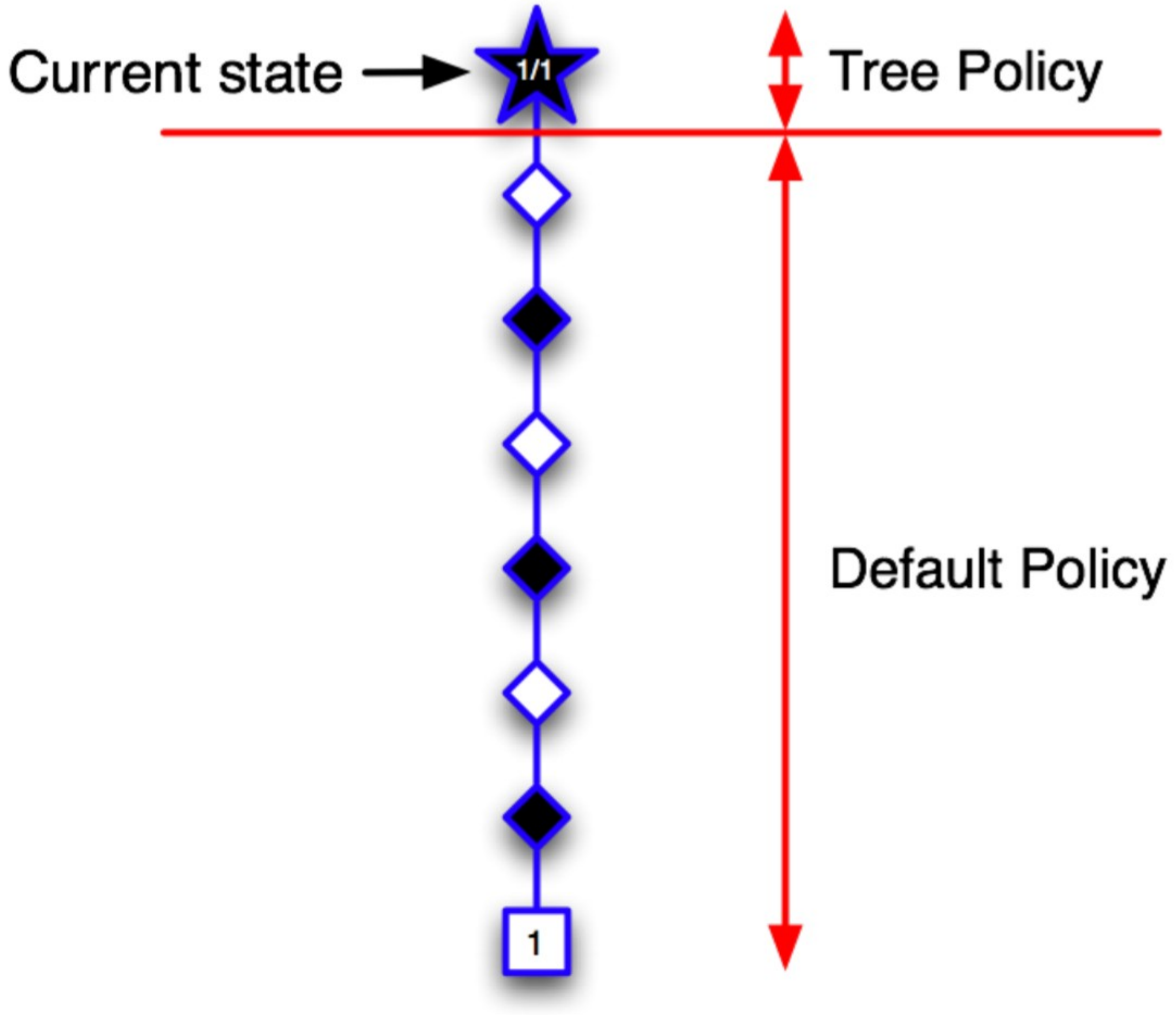
Monte-Carlo Tree Search

- Monte-Carlo Search can be integrated with conventional game-tree search algorithms

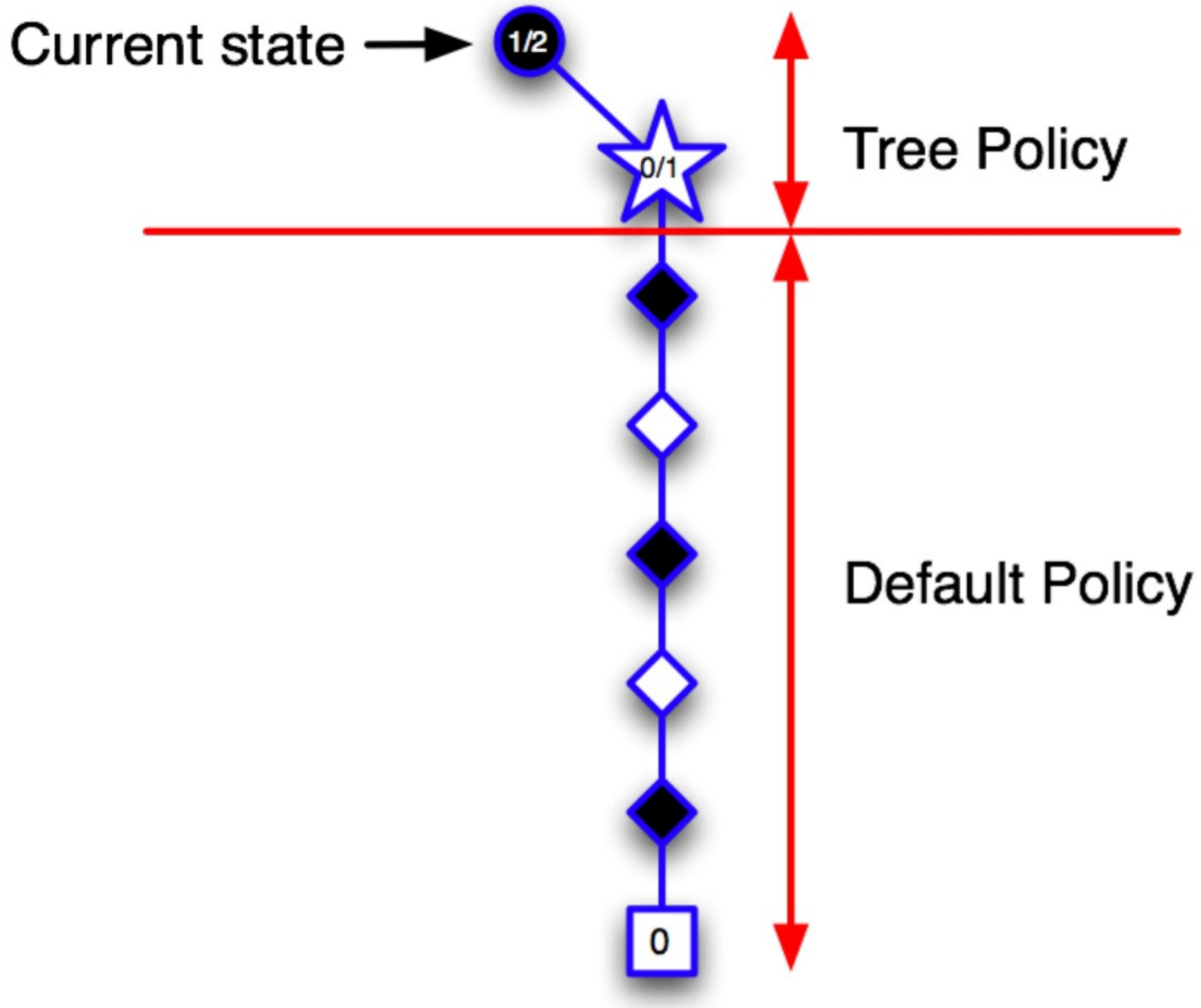

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

G.M.J-B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4(3), 2008.

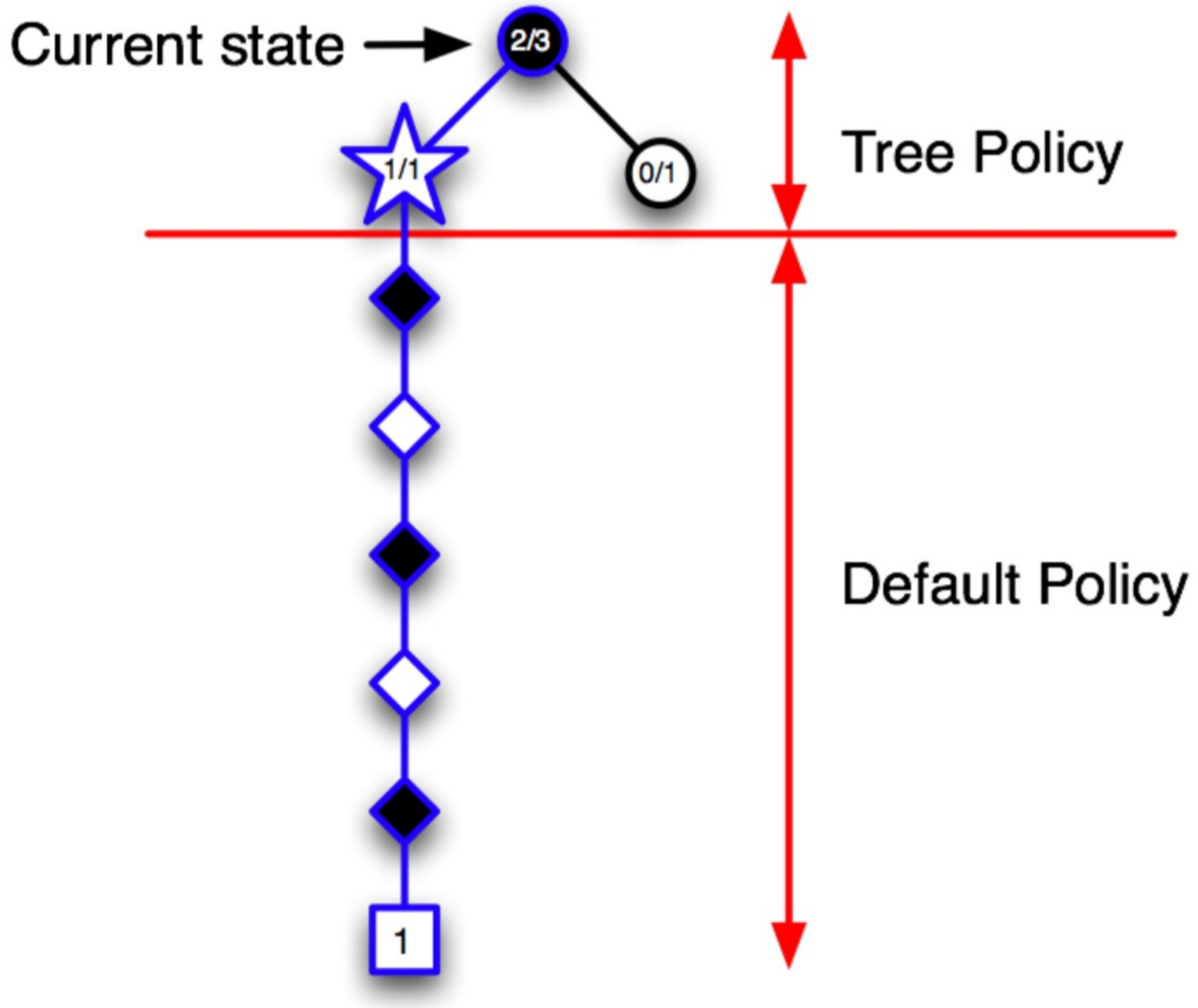
Applying MCTS



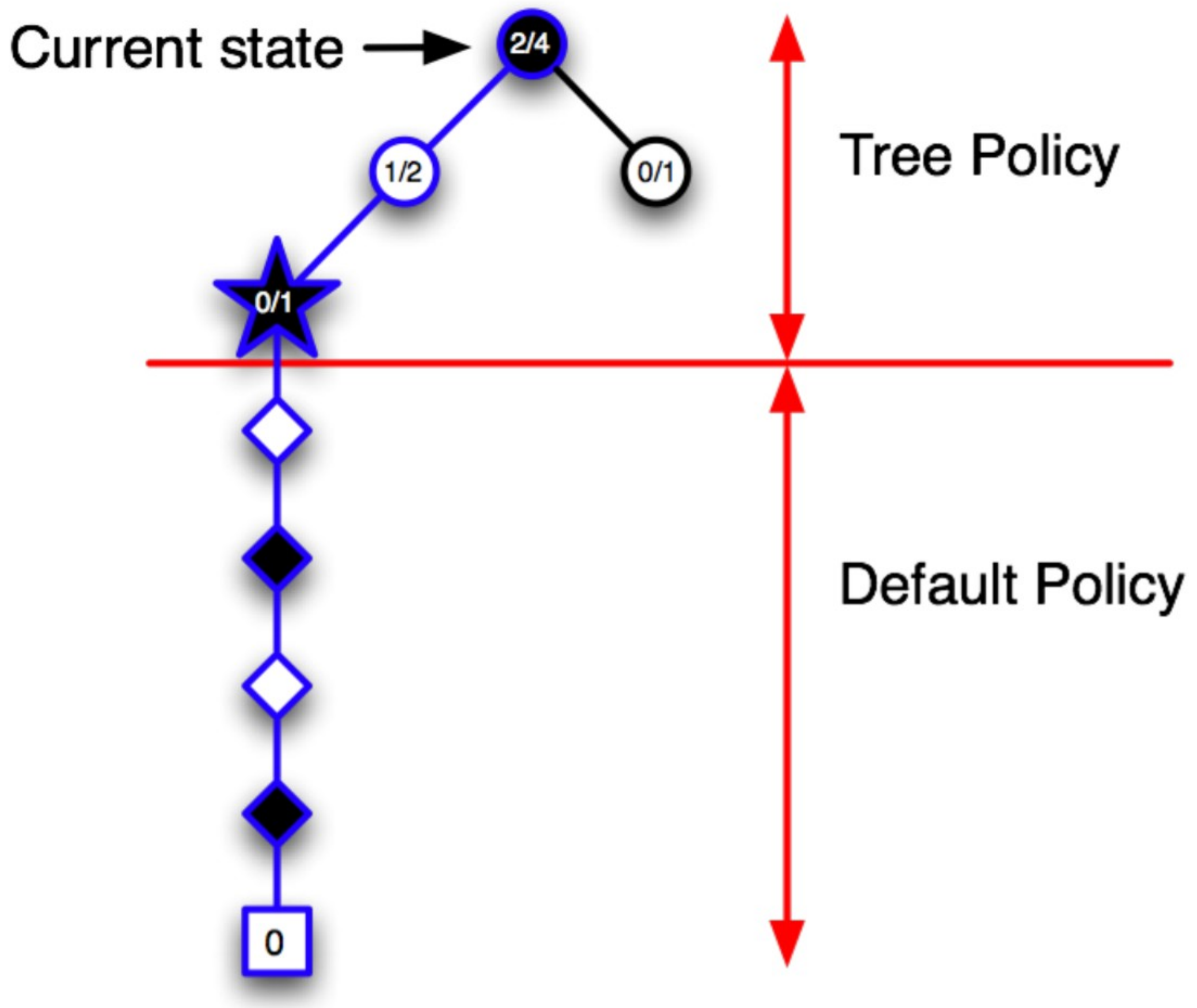
Applying MCTS



Applying MCTS

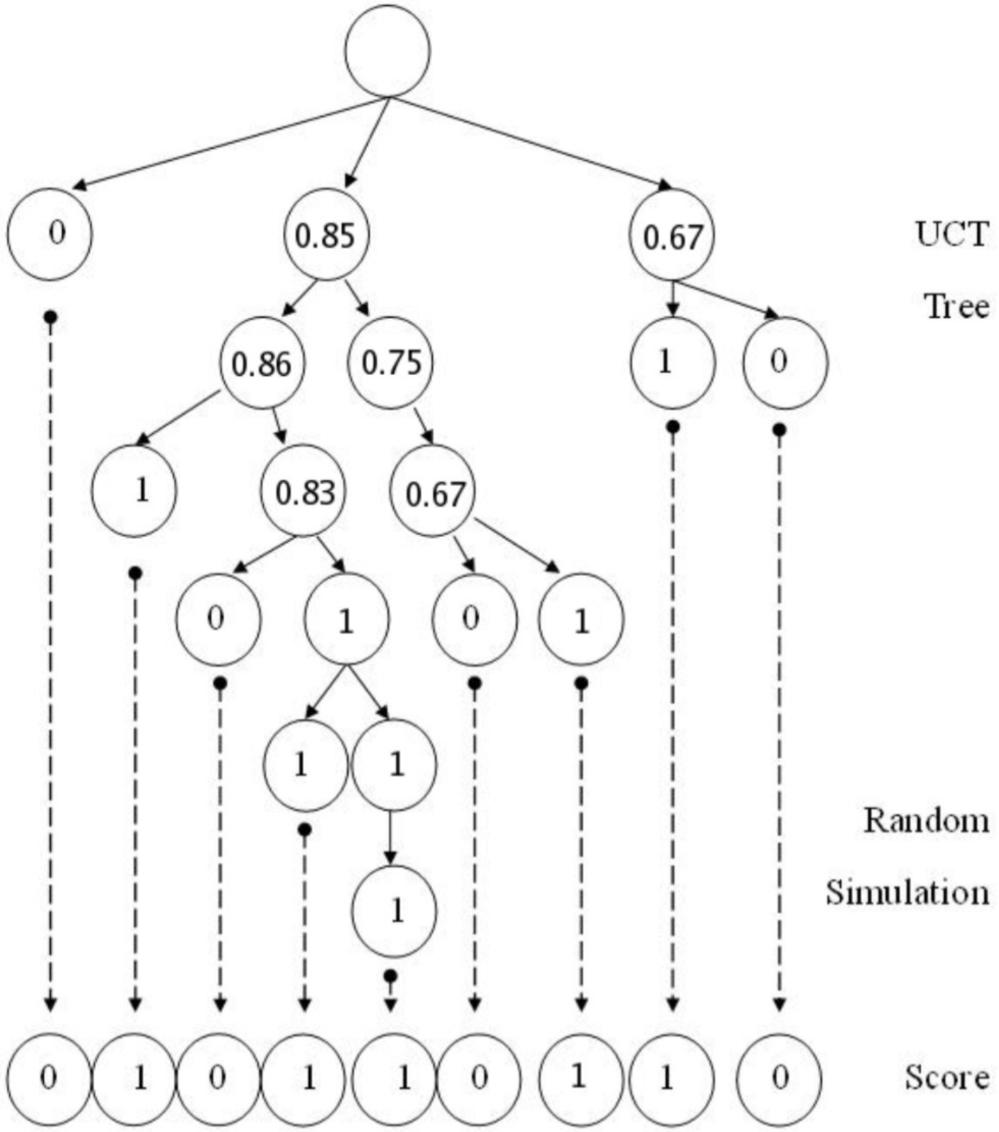


Applying MCTS



Selective Search in MCTS

- Typically, the tree policy is chosen in a way that prefers good alternatives over bad alternatives
  - → the tree grows deeper in regions of good moves



UCT Search
(Kocsis & Szepesvari, 2006)

- UCT is the best-known formulation of MCTS
  - it combines a UCB-based tree policy with random roll-outs

- Selection:
  - Select the node $s_{max} = \arg\max_{s \in \mathrm{Successors}(n)} \mathrm{value}(s) + C \cdot \sqrt{\frac{\ln \#\mathrm{visits}(n)}{\#\mathrm{visits}(s)}}$
  - Parameter C trades off between
    - **Exploitation**: Try to play the best possible move
      - maximize $\mathrm{value}(s)$
    - **Exploration**: Try new moves to learn something new
      - s gets a high value when the number of visits in the node is low
        - in relation to the number of visits in the parent node n
  - Sometimes:
    - only use UCT if the node has been visited at least *T* times
      - frequently used value $T = 30$

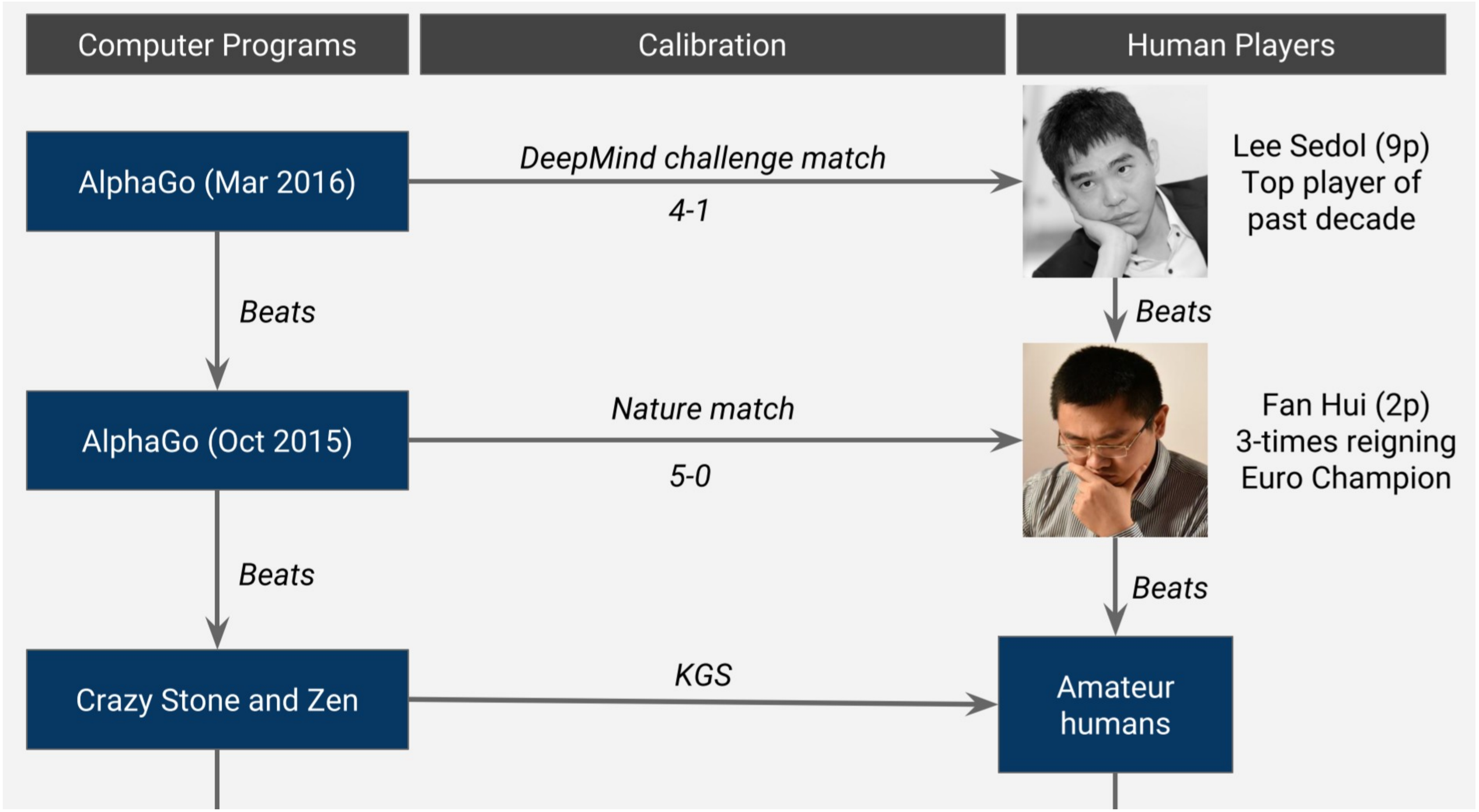
UCT Search
(Kocsis & Szepesvari, 2006)

- Expansion
  - add a randomly selected node to the game tree
- Simulation
  - perform one iteration of a Monte-Carlo search starting from the selected node
- Backpropagation
  - adapt $\text{value}(n)$ for each node n in the partial game tree
  - the value is just the average result of all games that pass through this node
- Move Choice
  - make the move that has been visited most often (reliability)
  - not necessarily the one with the highest value (high variance)

- UCT caused a breakthrough in Computer Go Research
  - e.g., MoGo (Gelly, Wang, Munos, Teytaud, 2006)

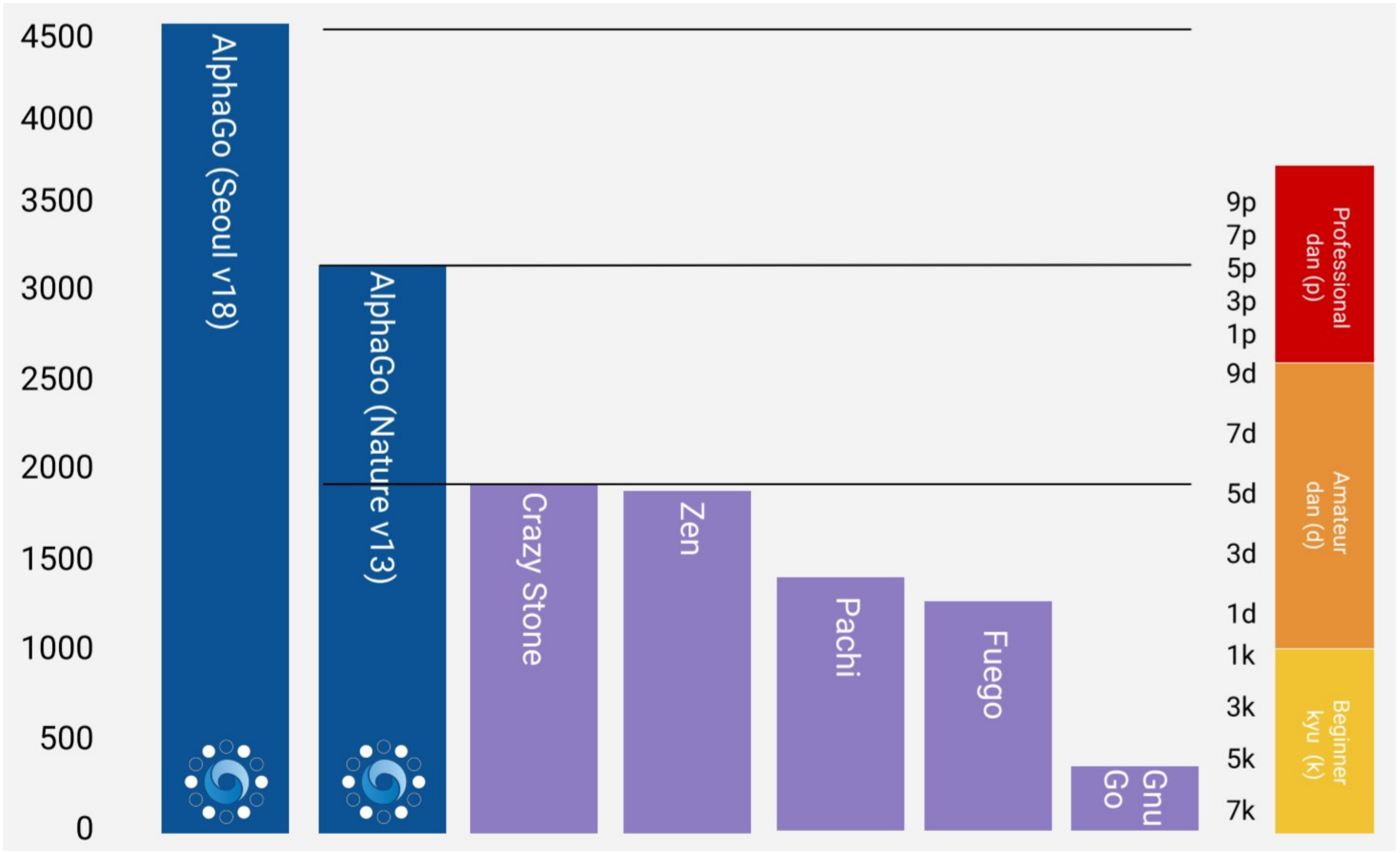
AlphaGo

- AlphaGo was the first Go Program to defeat a human champion Go player



AlphaGo

- AlphaGo reached unprecedented playing strength in computer (and human) Go



AlphaGo

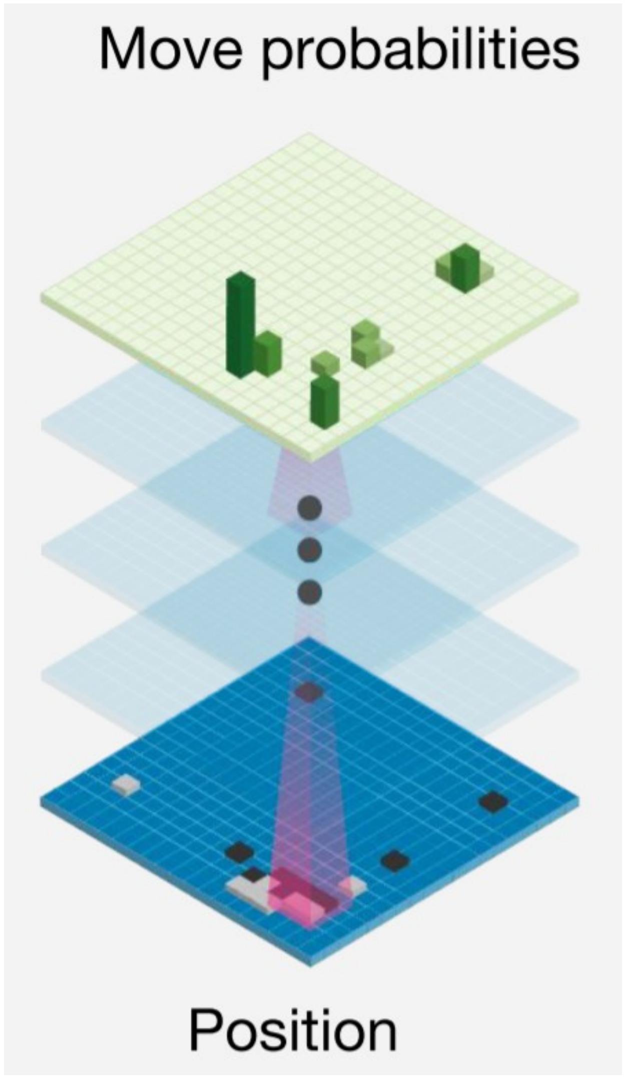
- AlphaGo combines MCTS with deep learning and reinforcement learning from self-play
  → these will be covered later…
  - which produced a large jump in playing strength

Key components of MCTS in AlphaGo:

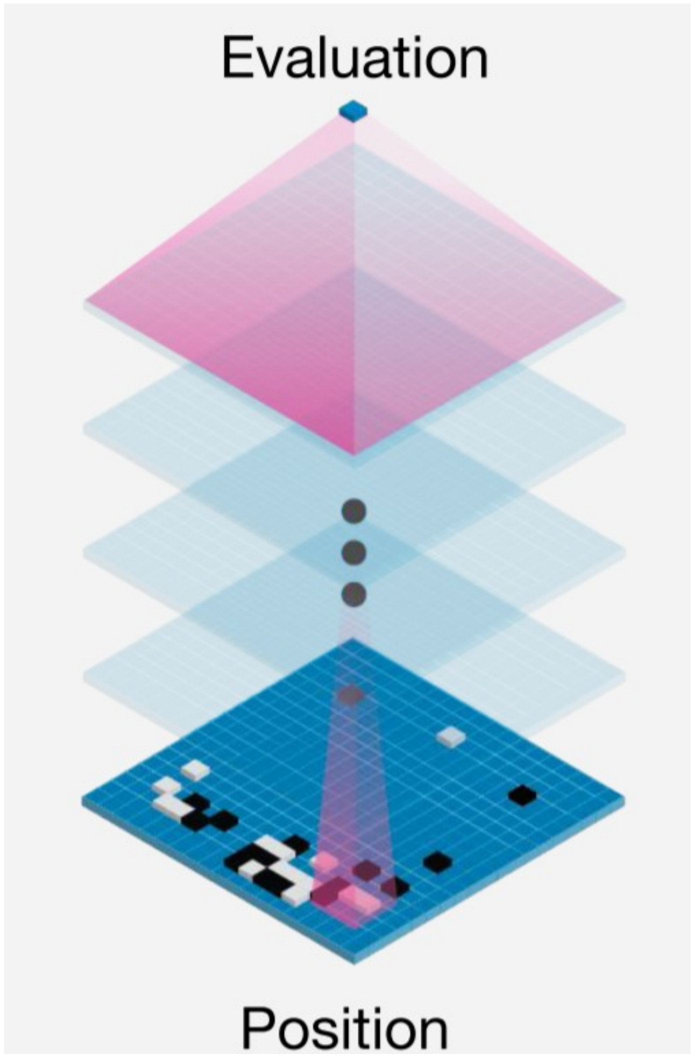
- use a (fast) **learned roll-out policy** instead of random sampling
- use a depth-limit in MCTS where a **learned evaluation function** is used instead of real game outcomes
  - similar to conventional search techniques

Two types of learned networks

- Policy Networks



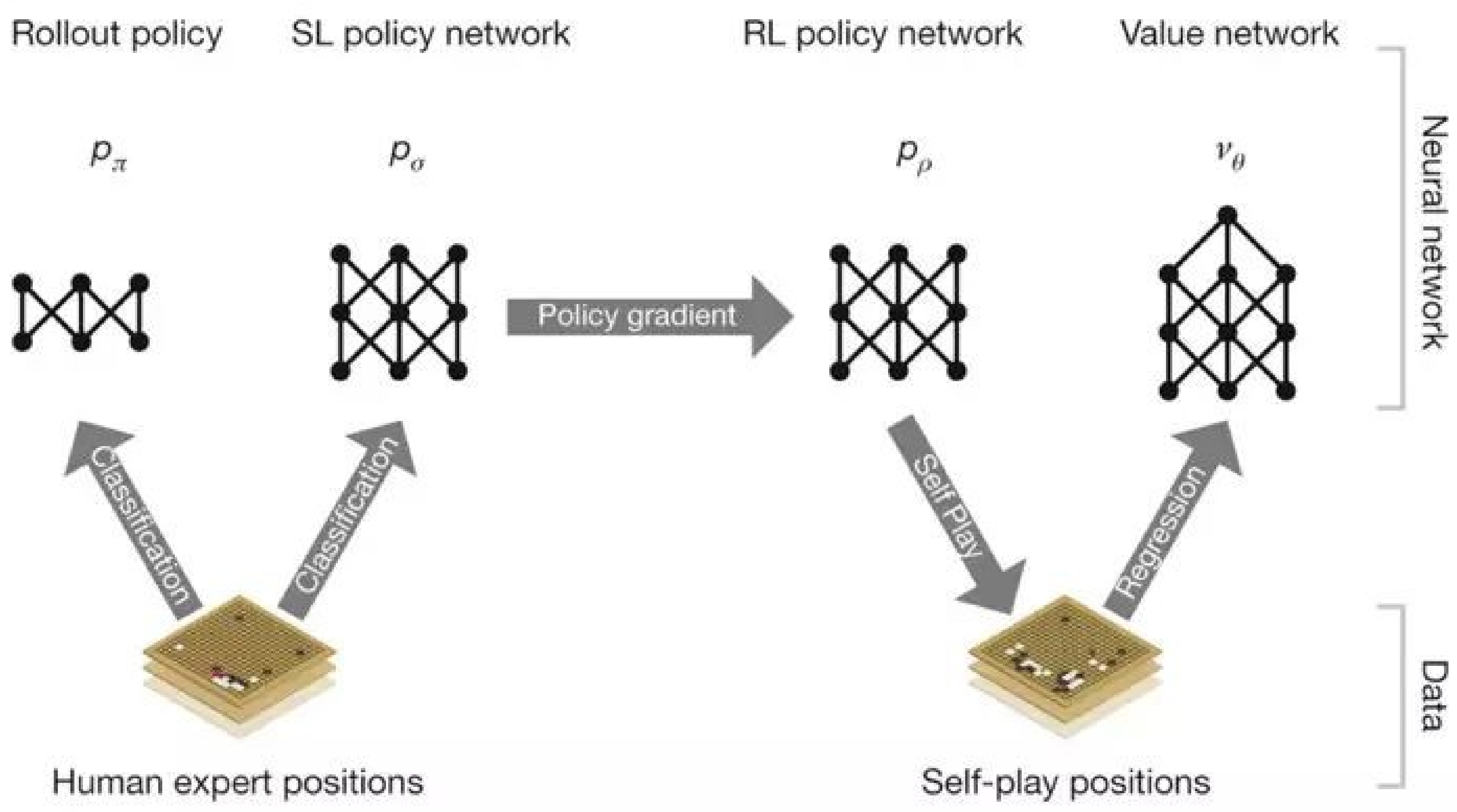
- Value Networks



AlphaGo – The four key learning steps

- learn from expert games a fast but inaccurate roll-out policy $p_\pi(a|s)$ for guiding the roll-outs in an MCTS algorithm
- learn from expert games an accurate expert policy $p_\sigma(a|s)$ to be used a prior probability in newly expanded nodes at the MCTS fringe
- refine the expert policy into a more accurate selection policy $p_\rho(a|s)$ using policy gradient search from self-play
- use self-play from the selection policy to train a utility function $v_\theta(s)$ for evaluating a given game position, which (at the MCTS fringe nodes) will be averaged with the final game evaluation using a trade-off parameter λ

AlphaGo – The four key learning steps

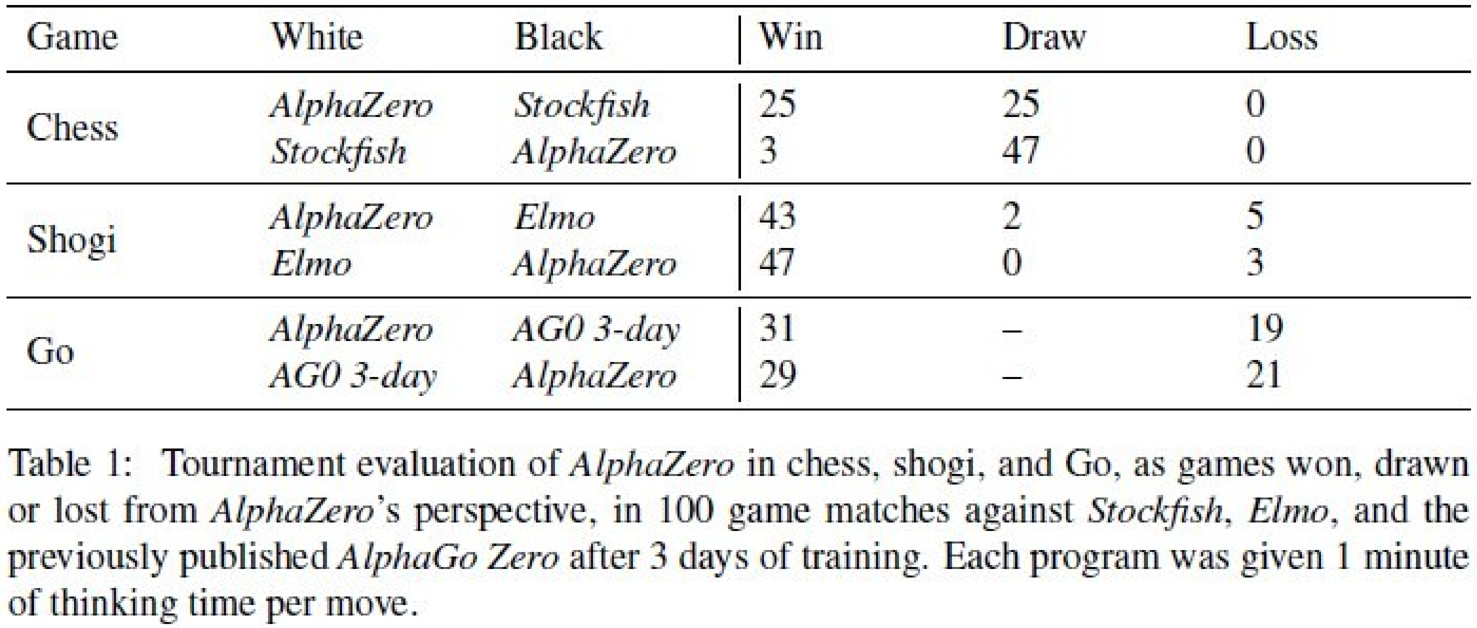


AlphaZero

- AlphaGo Zero
  - improved version that learned to play only from self-play
  - beat AlphaGo 100-0 using much less training data

  https://www.youtube.com/watch?time_continue=36&v=tXIM99xPQC8

- AlphaZero furtherimprovement in Go, Chess, Shogi



Game	White	Black	Win	Draw	Loss
Chess	*AlphaZero*	*Stockfish*	25	25	0
	Stockfish	*AlphaZero*	3	47	0
Shogi	*AlphaZero*	*Elmo*	43	2	5
	Elmo	*AlphaZero*	47	0	3
Go	*AlphaZero*	*AG0 3-day*	31	–	19
	AG0 3-day	*AlphaZero*	29	–	21

Table 1: Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in 100 game matches against *Stockfish*, *Elmo*, and the previously published *AlphaGo Zero* after 3 days of training. Each program was given 1 minute of thinking time per move.

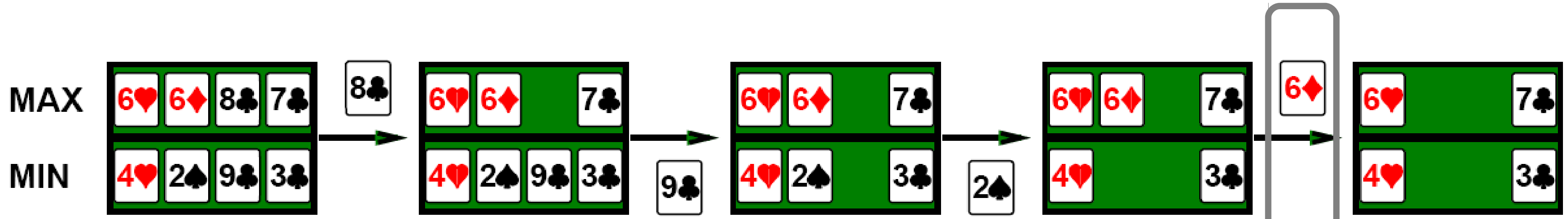
Games of Imperfect Information

- The players do not have access to the entire world state
  - e.g., card games, when opponent's initial cards are unknown
- We can calculate a probability for each possible deal
  - seems just like one big dice roll at the beginning of the game
- Intuitive Idea:
  - compute the minimax value of each action in each deal
  - choose the action with the highest expected value over all deals
- Main problem:
  - too many possible deals to do this efficiently

  → take a sample of all possible deals
- Example:
  - GIB (a very good Bridge program) generates 100 deals consistent with bidding information (this also restricts!)
  - picks the move that wins the most tricks on average

Imperfect Information - Example

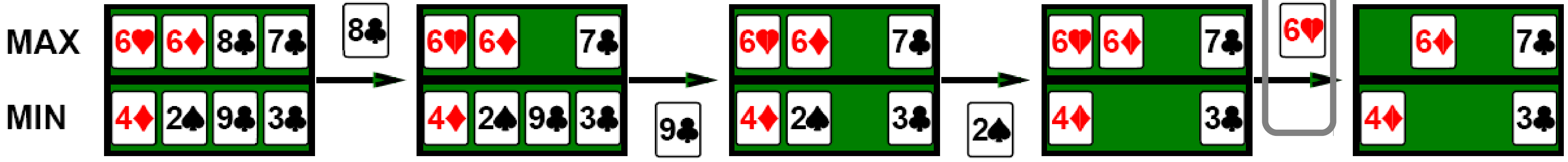
Scenario a) MIN has 4♥



→ both players will make two tricks

MAX can play optimally if MAX knows MIN's cards

Scenario b) MIN has 4♦

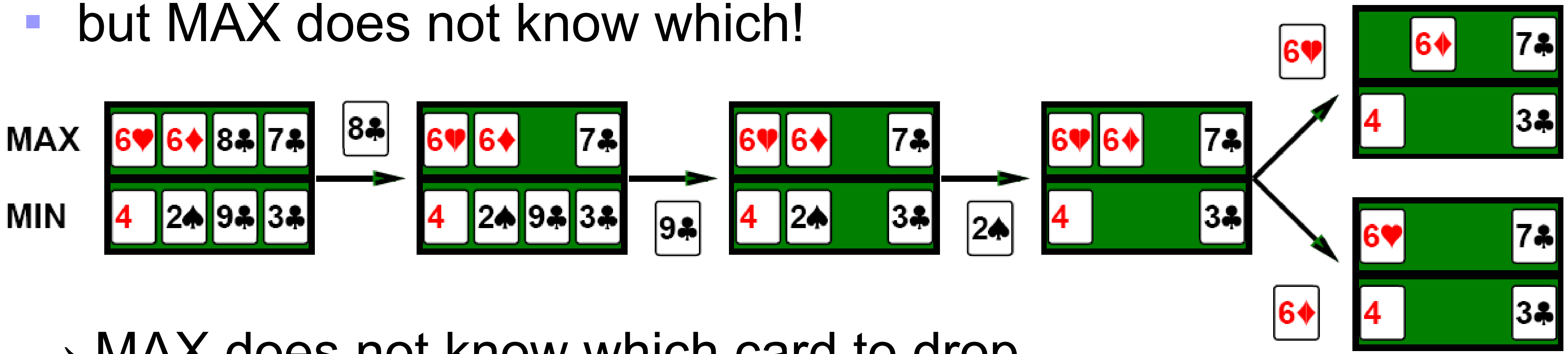


→ both players will make two tricks

Imperfect Information - Example

Scenario c) MIN has either 4♥ or 4♦

- but MAX does not know which!

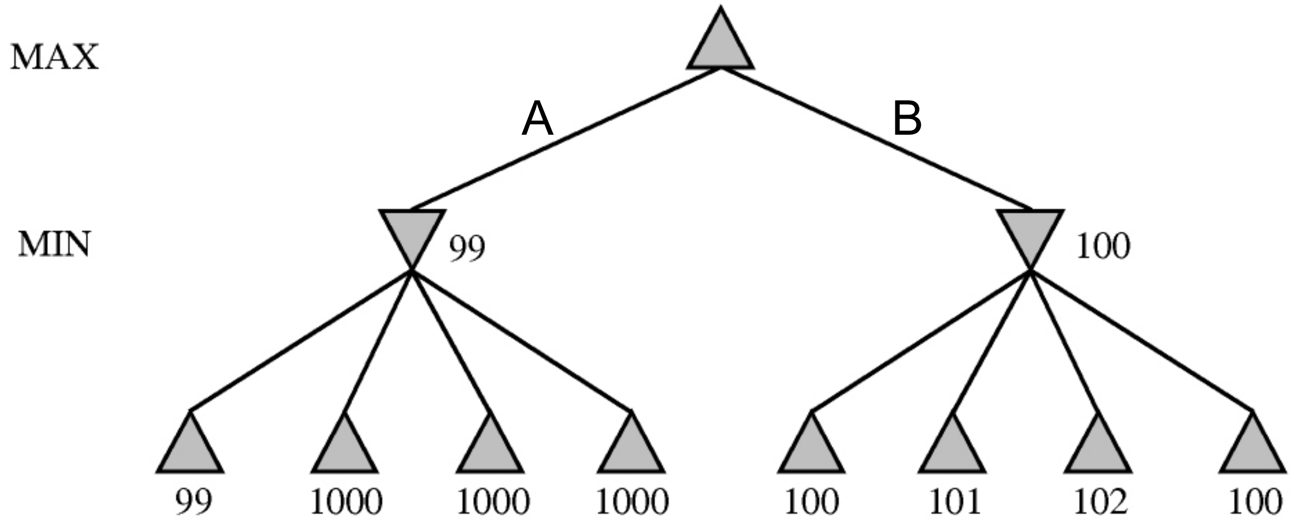


→ MAX does not know which card to drop and has a 50% chance of losing the game!

- Lesson:
  - The intuition that the value of an action is the average of its value in all actual states is *wrong*!
  - the value of an action also depends on the agents' belief state
    - if I know that it is more probable that he has 4♥, the expected value should be adjusted accordingly
  - may lead to information-gathering or information-disclosing actions (e.g., signalling bids or unpredictable (random) play)

Belief States in Minimax Search

- Minimax always assumes that the opponent plays its best response (it is said to be *conservative)*
- This may be a bad idea:



  - MAX will play move B
  - If there is a small chance that MIN does not play according to MAX's evaluation
    - because the evaluation is wrong or MIN makes a mistake

  then A would be the better choice!

Opponent Modeling

Somewhat off-topic, but see also:
http://www.youtube.com/watch?v=3nxjjztQKtY

- For simple games we know optimal solutions
  - Complete search through Minimax tree
  - Game-Theory: Nash-Equilibrium
- Optimal solutions are not Maximal!
  - Example: Roshambo (Rock/Paper/Scissors)
    - Optimal Solution: Pick a random move
    - clearly suboptimal against a player that always plays rock!

  → Roshambo Computer Tournament (1999, 2000)
- Opponent Modeling
  - try to predict the opponent's next move
  - try to predict what move the opponent predicts that your next move will be, ...
- For some games, opponent modeling is part of the game
  - e.g., bluffing and calling a bluff in Poker

Perspective on Games: Pro

“Saying Deep Blue doesn’t really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings”

Drew McDermott

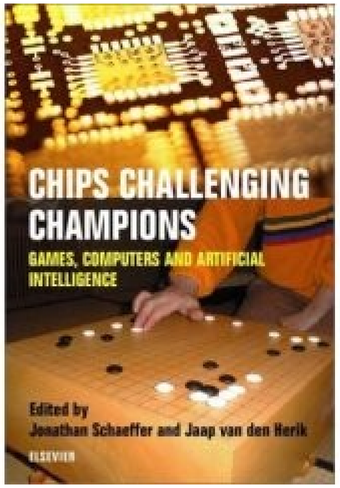
Perspective on Games: Con

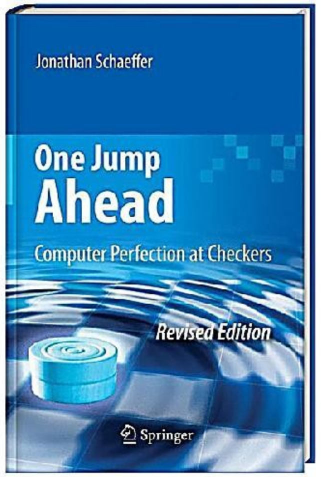
“Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.”

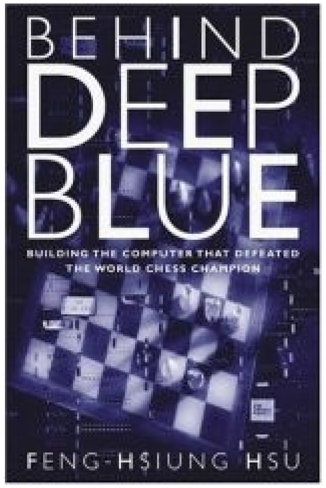
John McCarthy

Additional Reading

- Jonathan Schaeffer. The Games Computers (and People) Play, *Advances in Computers 50* , Marvin Zelkowitz (ed.) Academic Press, pp. 189-266, 2000. http://www.cs.ualberta.ca/~jonathan/Papers/Papers/advances.ps
  - excellent survey paper
- Jonathan Schaeffer and Jaap van den Herik (eds.) *Chips Challenging Champions: Games, Computers and Artificial Intelligence*, North-Holland 2002.
  - very good collection of papers
- Jonathan Schaeffer: *One Jump Ahead: Challenging Human Supremacy in Checkers*, Springer 1998, revised 2009.
  - non-technical first-hand account on the Chinook project
- Feng-Hsiung Hsu: *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*, Princeton 2002
  - non-technical first-hand account on Deep Blue


CHIPS CHALLENGING CHAMPIONS
GAMES, COMPUTERS AND ARTIFICIAL INTELLIGENCE
Edited by Jonathan Schaeffer and Jaap van den Herik
ELSEVIER




BEHIND DEEP BLUE
BUILDING THE COMPUTER THAT DEFEATED THE WORLD CHESS CHAMPION
FENG-HSIUNG HSU

Additional Reading AlphaGo

- Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M., A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis. arXiv 2017
- Mastering the Game of Go without Human Knowledge. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel & D. Hassabis. *Nature* 2017.
- Mastering the Game of Go with Deep Neural Networks and Tree Search. D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis. *Nature* 2016.

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
SONGBIRDS À LA CARTE
SAFEGUARD TRANSPARENCY
WHEN GENES GOT 'SELFISH'