Bandits and MCTS

UCSD CSE257
Sicun Gao

Bandits: Applications

- A/B testing, advertising and recommendation
- Network routing
- Dynamic pricing, resource allocation
- Tree search in games (MCTS)
- Microsoft “Decision Service”

Choosing a good coin

Sequential bet with two unfair coins. Assume the payoff in each round is always in [0,1].

μ_1

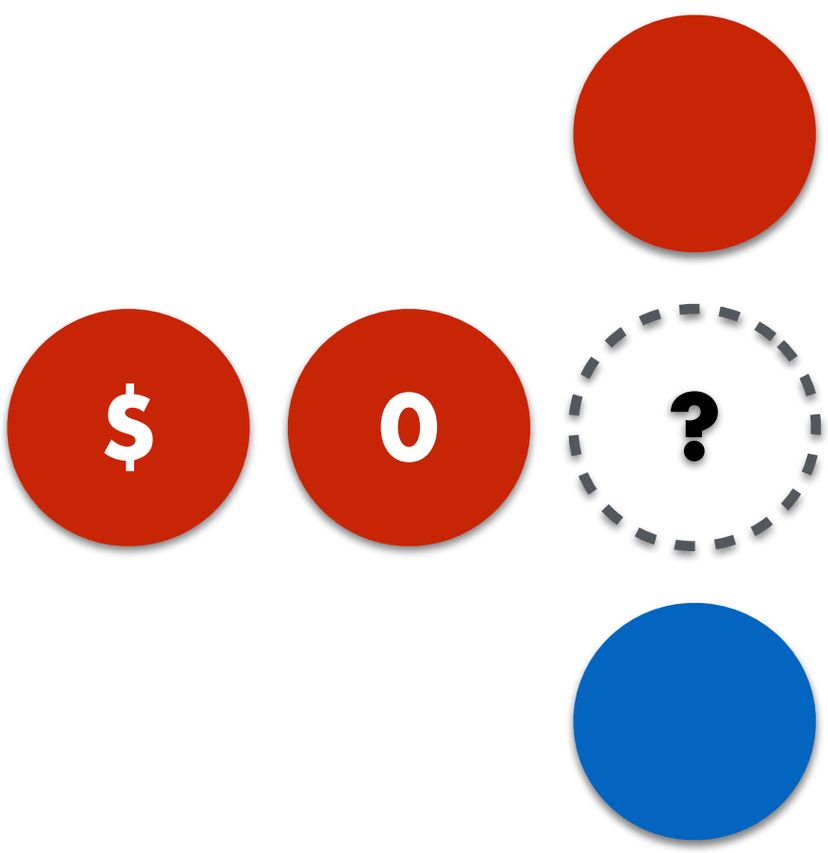
μ_2

Choosing a good coin

Two random variables, a choice at each step

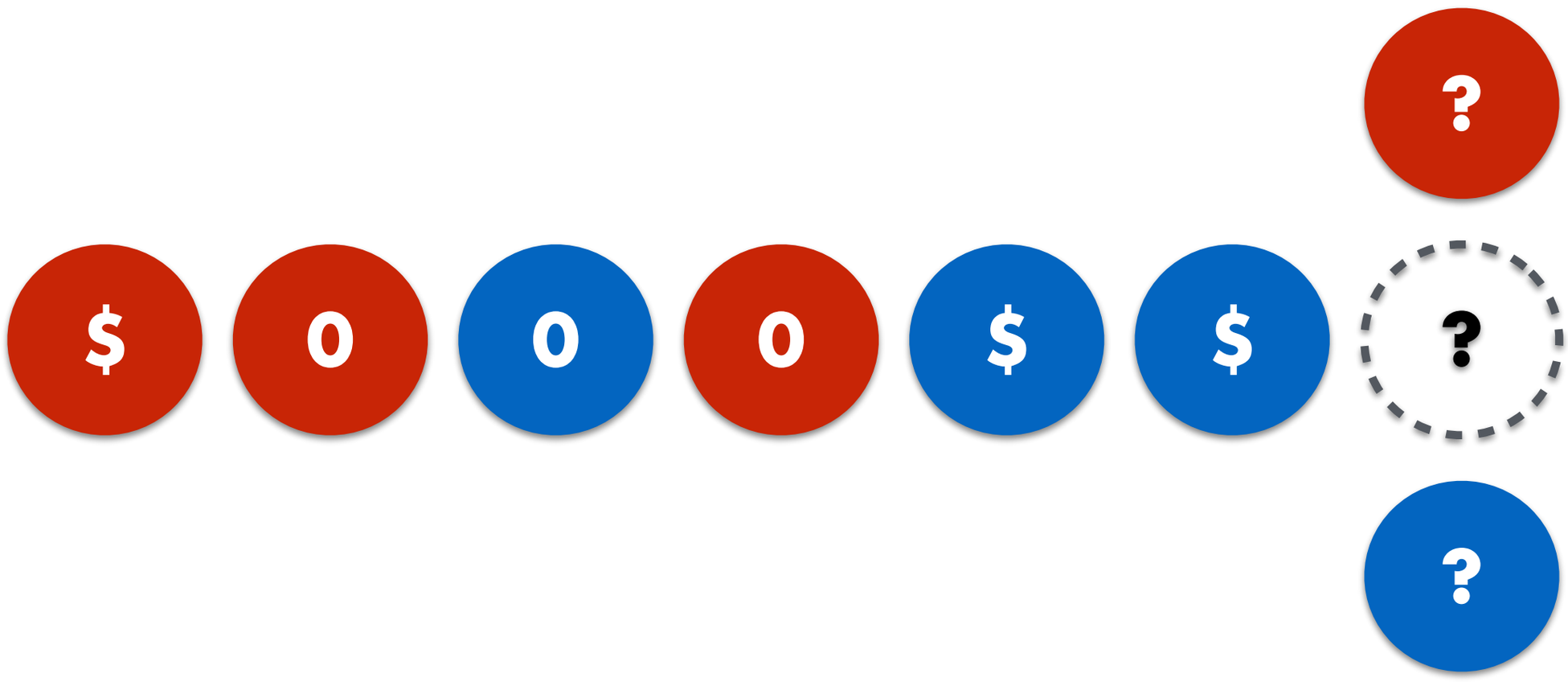
Choosing a good coin

Two random variables, one choice at each step

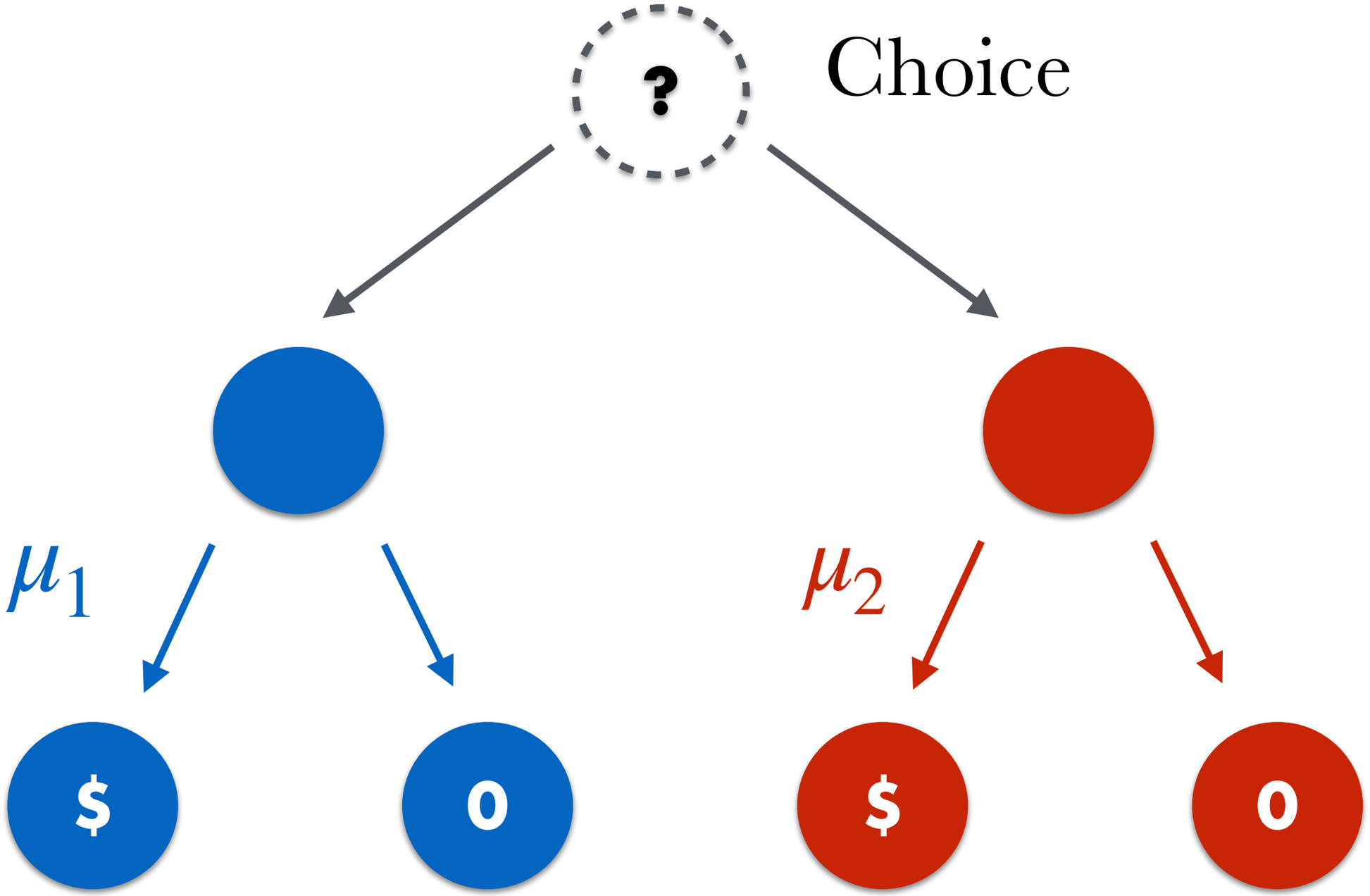


Choosing a good coin

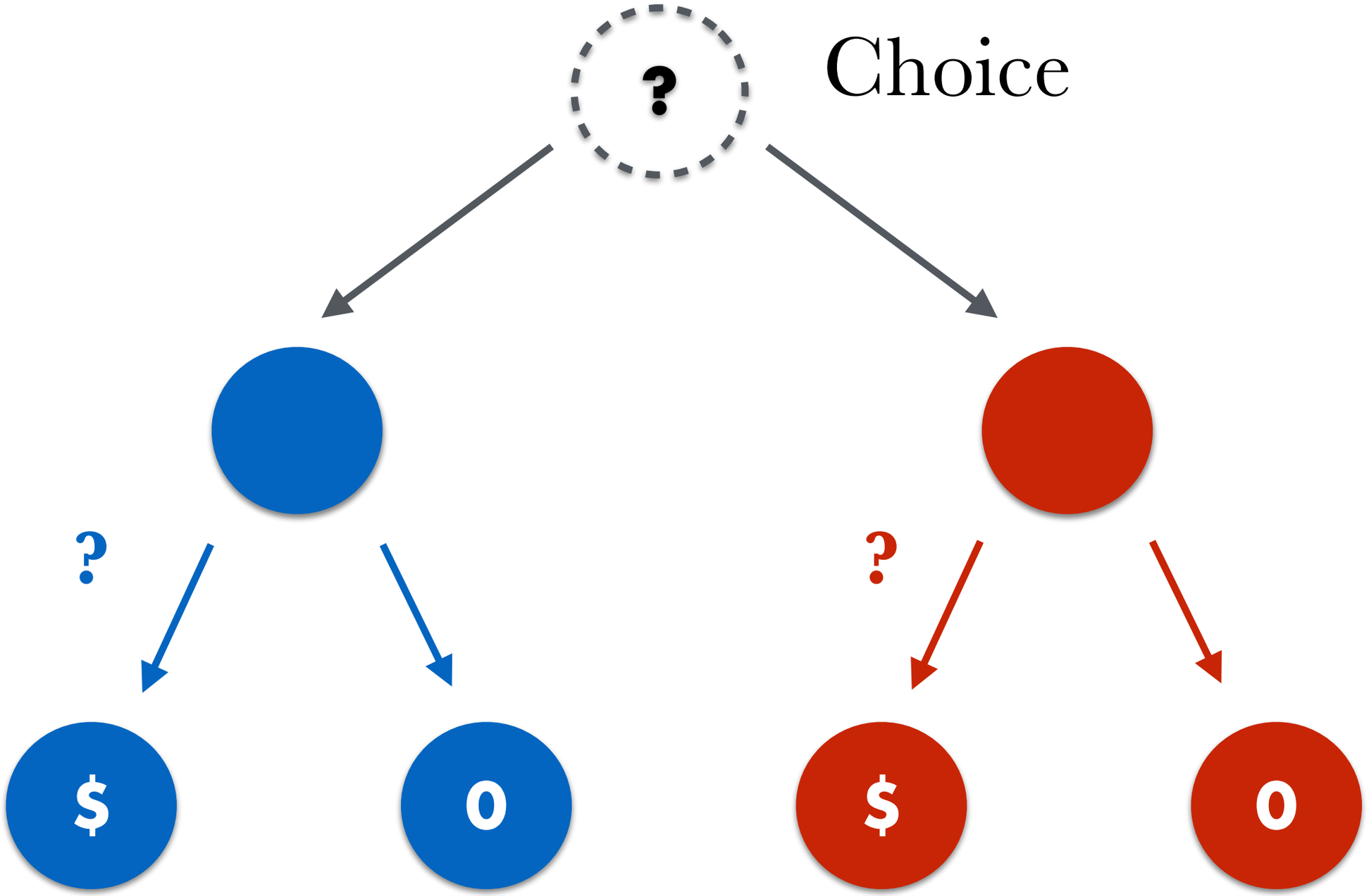
Two random variables, one choice at each step



Relate to Expectimax

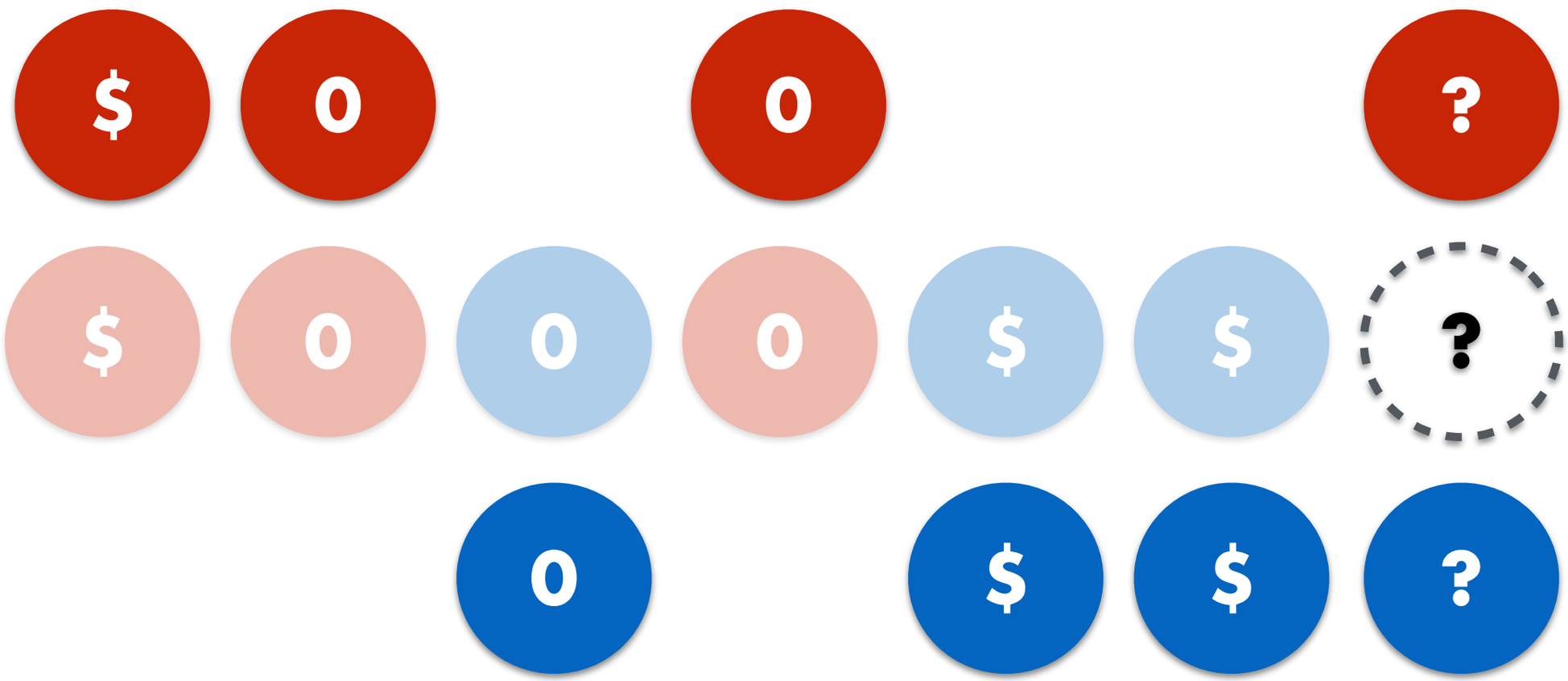


Relate to Expectimax



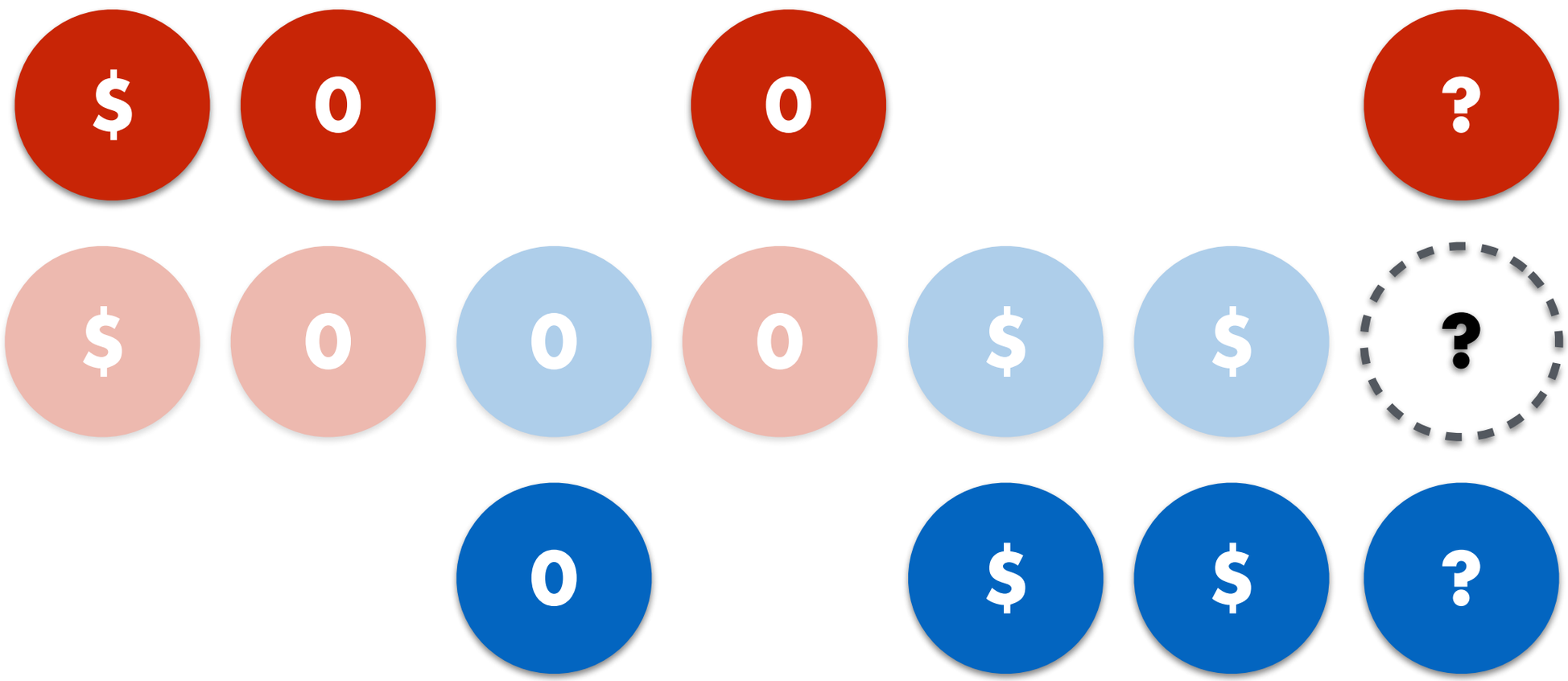
Choosing a good coin

Use experience to estimate which coin is good. Want as few bad choices as possible.



Choosing a good coin

Trade-off: better estimates vs. better reward



Choosing a good coin

- Can you expect to earn $ in every round?
- How good is it to choose at random?
- How good is it to blindly stick with one?
- What is the best possible outcome?
- How to measure good or bad?

Choosing a good coin

- Regret over time of a strategy σ: expected loss over time compared to the best possible outcome (of always betting on the best coin)

$$R_\sigma(t) = \mathbb{E}_\sigma\left(\sum_{i=0}^{t} X_{c*} - \sum_{i=0}^{t} X_{c_i}\right)$$

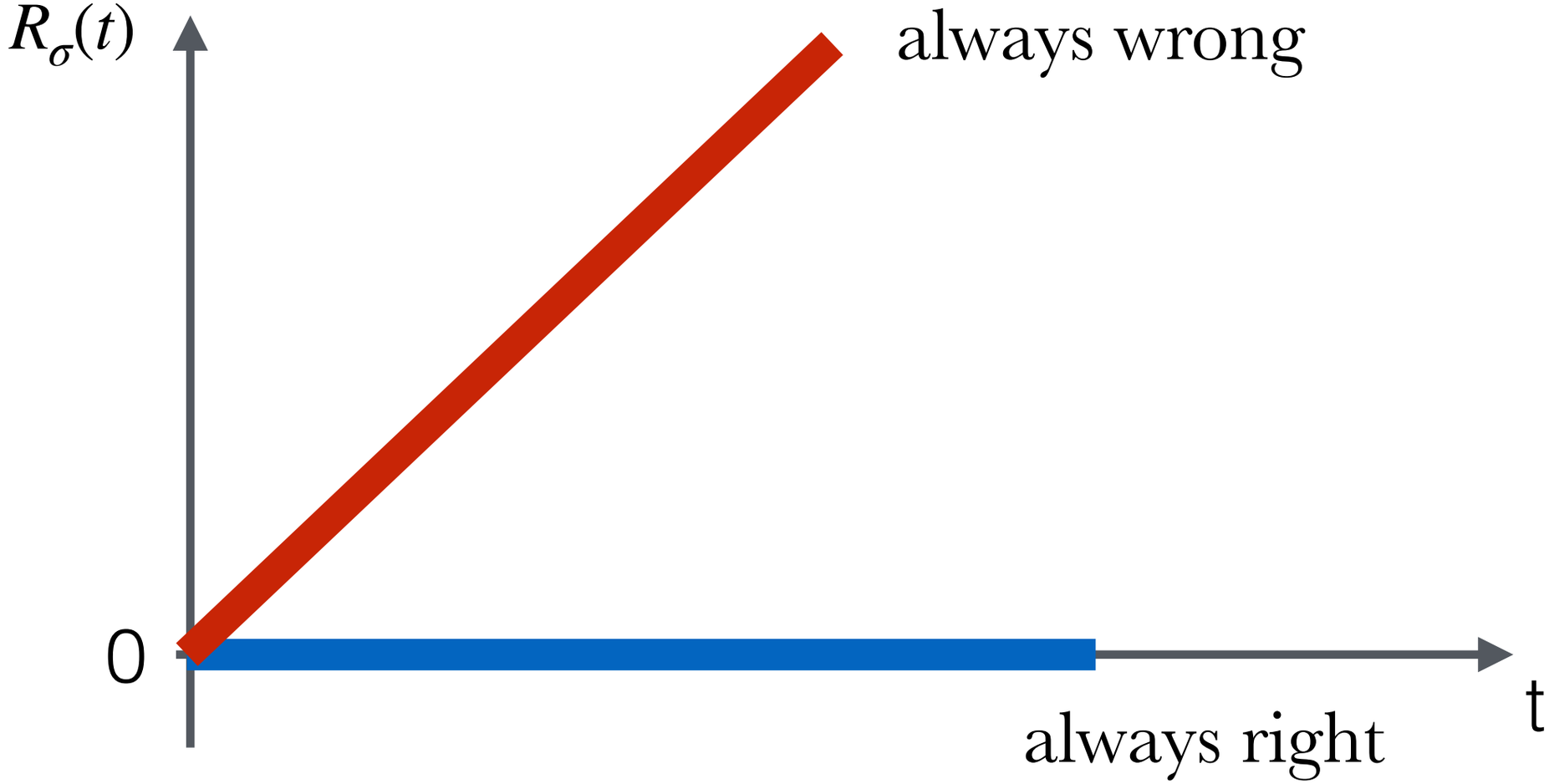
Choosing a good coin

$$R_\sigma(t) = \mathbb{E}_\sigma(\sum_{i=0}^{t} X_{c*} - \sum_{i=0}^{t} X_{c_i}) = \mu^* t - \mathbb{E}_\sigma(\sum_{i=0}^{t} \mu(c_i))$$

- We aim to design strategy σ that minimizes $R_\sigma(t)$.

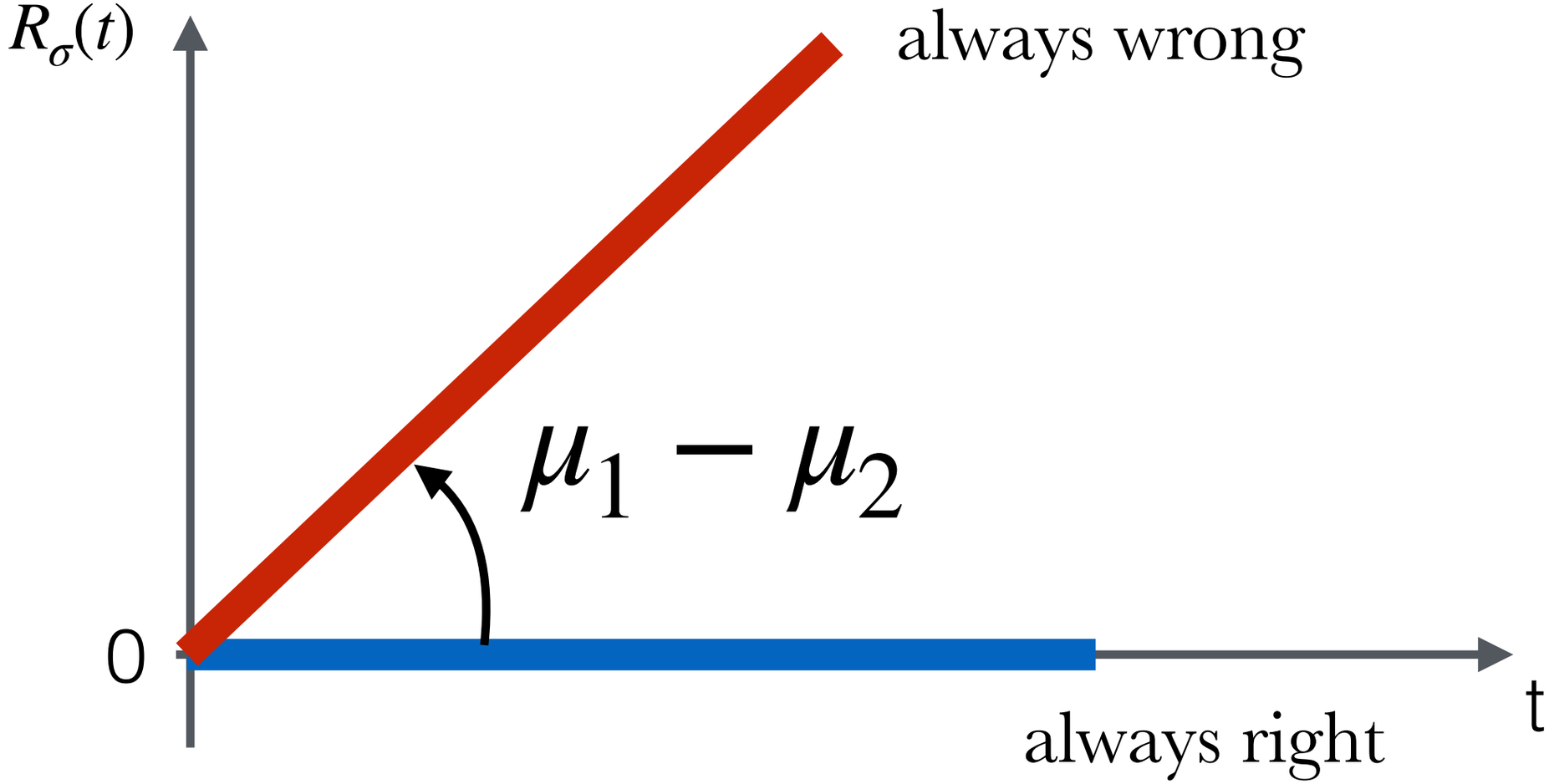
Choosing a good coin

$$R_\sigma(t) = \mathbb{E}_{c_i \sim \sigma}[\mu^* t - \sum_{i=0}^{t} \mu(c_i)]$$



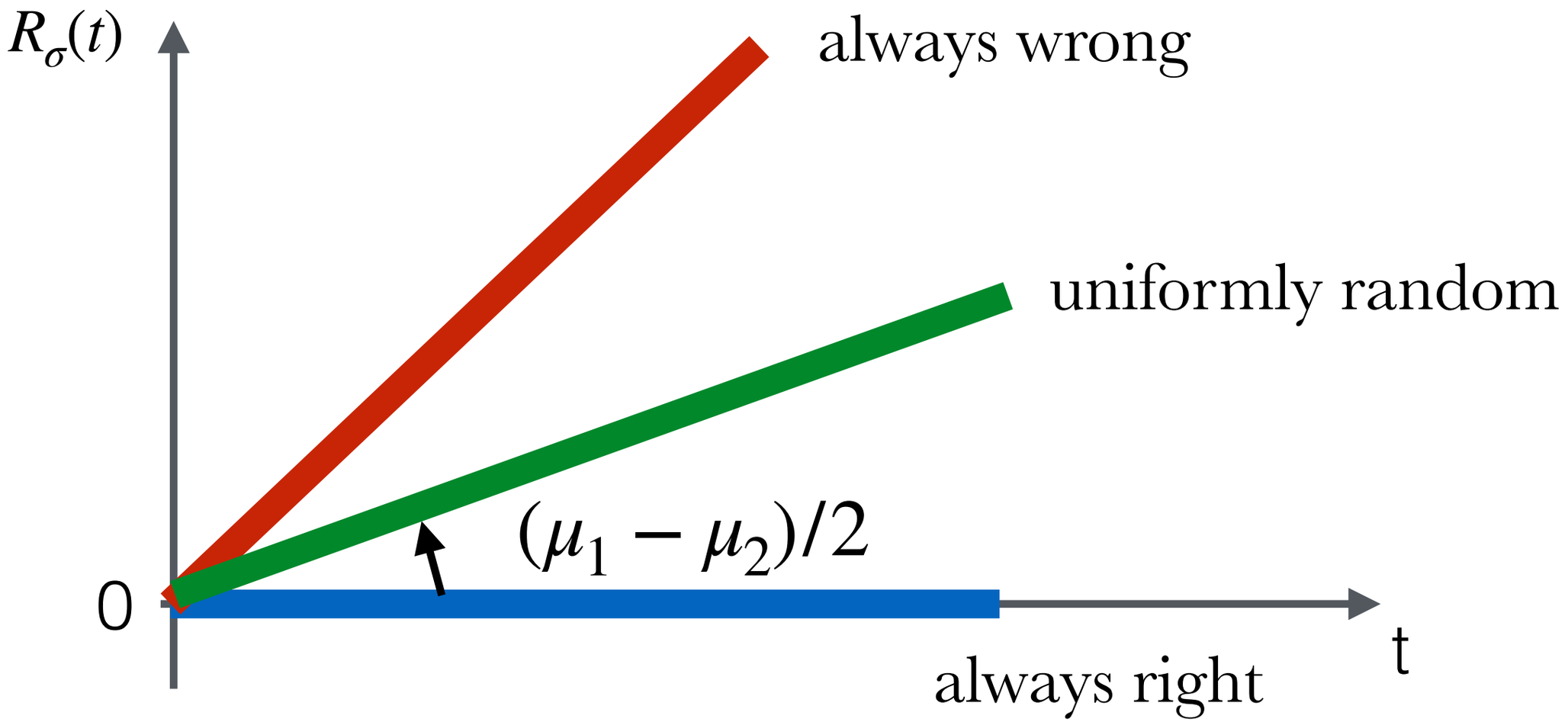
Choosing a good coin

$$R_\sigma(t) = \mathbb{E}_{c_i \sim \sigma}[\mu^* t - \sum_{i=0}^{t} \mu(c_i)]$$

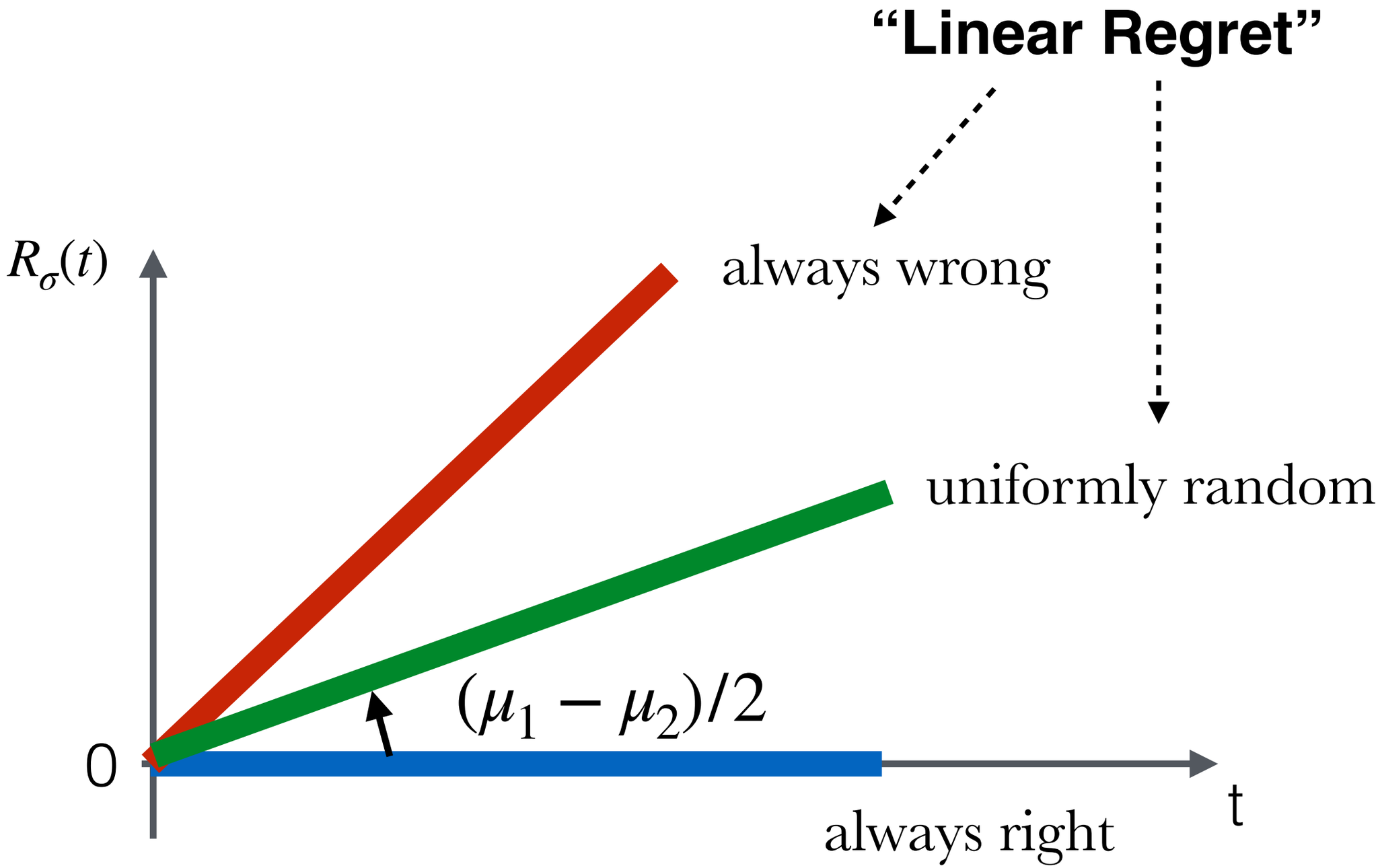


Choosing a good coin

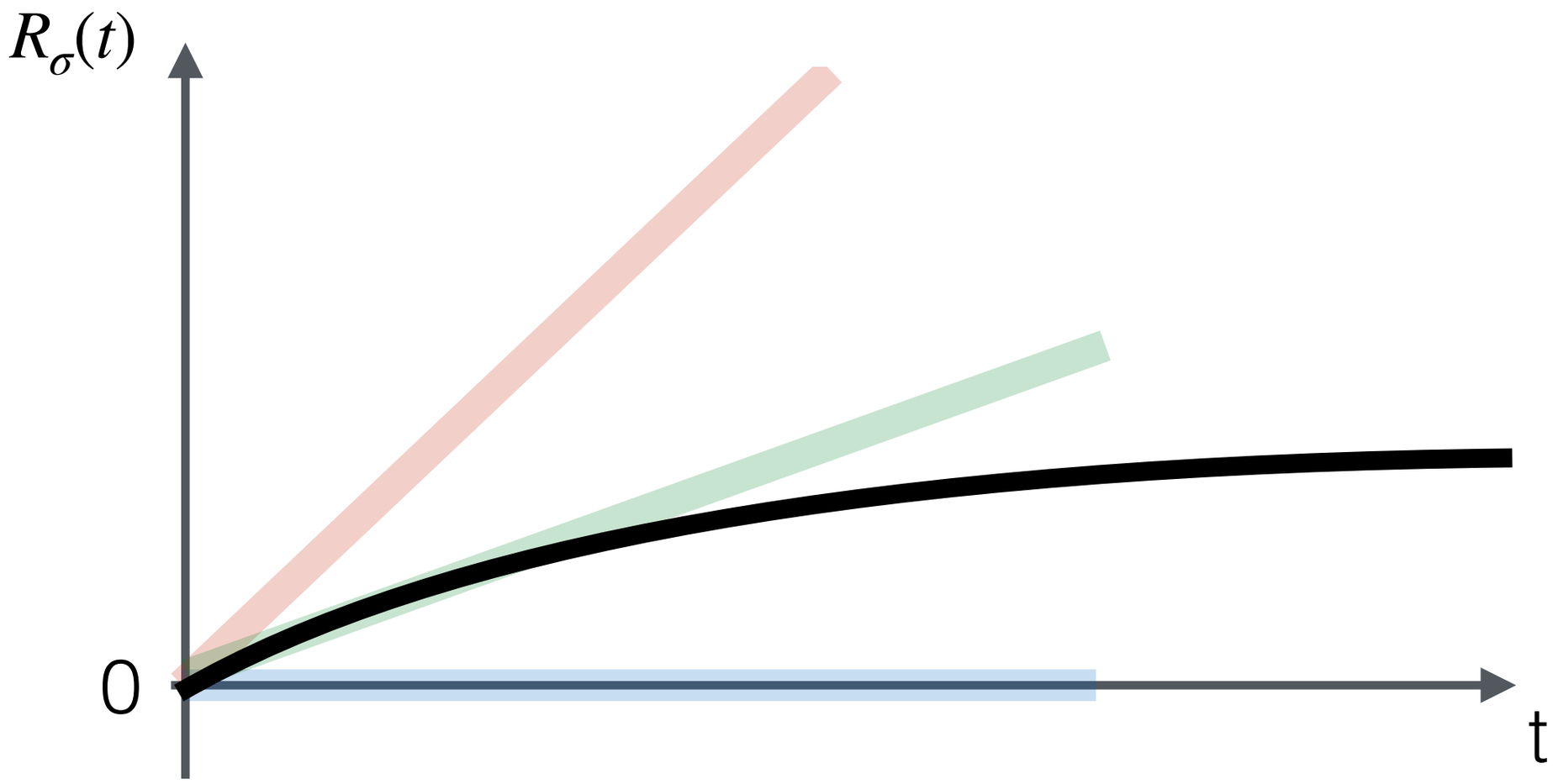
$$R_\sigma(t) = \mathbb{E}_{c_i \sim \sigma}[\mu^* t - \sum_{i=0}^{t} \mu(c_i)]$$



Choosing a good coin



Choosing a good coin



Is it possible to achieve sublinear regret?

Our Goal: Understand "Upper Confidence Bound"

For any round t:

- Compute the UCB value of each arm $i \in [K]$

$$B(i, t, n_i(t)) = \bar{X}_i + \sqrt{\frac{2 \log t}{n_i(t)}}$$

- Play the arm with $\max_i B(i, t, n_i(t))$

Simple strategy: explore-then-commit

Suppose we play T rounds in total:

- First, play each coin N times ($N \leq T/2$)
- Compare the average reward $\bar{\mu}_1$ and $\bar{\mu}_2$
- Afterwards play the coin with higher $\bar{\mu}_i$

Simple strategy: explore-then-commit

Suppose we play T rounds in total:

- First, play each coin N times ($N \leq T/2$)
- Compare the average reward $\bar{\mu}_1$ and $\bar{\mu}_2$
- Afterwards play the coin with higher $\bar{\mu}_i$

We will see this result: using $N \sim cT^{\frac{2}{3}} \log T^{\frac{1}{3}}$

$$R(T) \sim O\left(T^{\frac{2}{3}}(\log T)^{\frac{1}{3}}\right)$$

Simple strategy: explore-then-commit

Suppose we play T rounds in total:

- First, play each coin N times ($N \leq T/2$)
- Compare the average reward $\bar{\mu}_1$ and $\bar{\mu}_2$
- Afterwards play the coin with higher $\bar{\mu}_i$

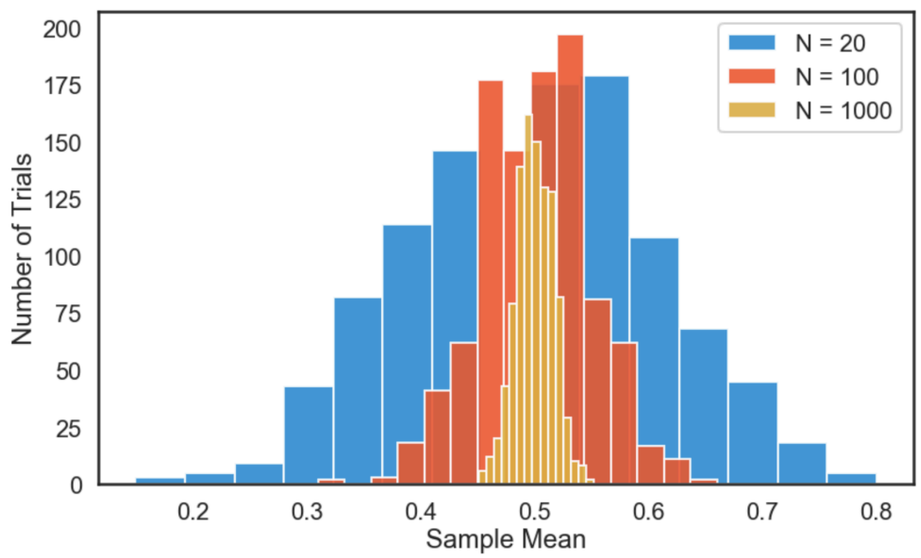
What if N is very large? What if N is very small?

Concentration bounds

Take N i.i.d. samples $X_1, \ldots, X_N \sim P_\mu$ in $[0,1]$

- Hoeffding's inequality (1963)

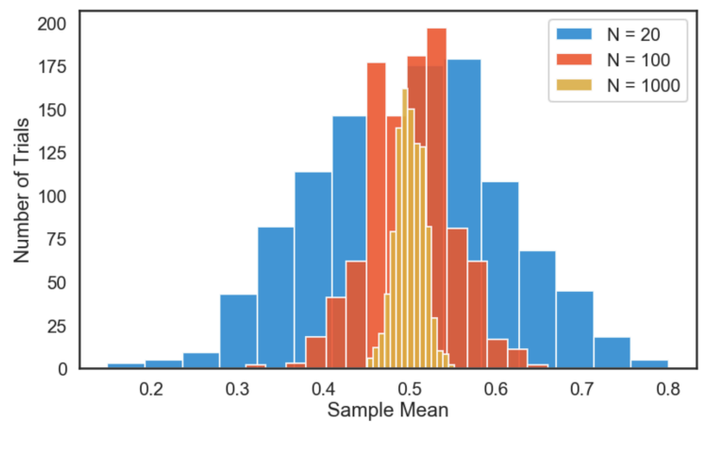
$$P(|\bar{X} - \mu| \geq \varepsilon) \leq 2e^{-2N\varepsilon^2}$$



Concentration bounds

$$P(|\bar{X} - \mu| \geq \varepsilon) \leq 2e^{-2N\varepsilon^2}$$

confidence radius



- That is, we can choose ε to make the tail ("not concentrating") arbitrarily small.
- In fact, we can let ε depend on N.

Concentration bounds

- For instance, by choosing

$$\varepsilon = \sqrt{\frac{c \log T}{N}}$$

We have the following concentration bound, where c can easily control how quickly it converges

$$P(|\bar{X} - \mu| \geq \varepsilon) \leq \frac{2}{T^{2c}}$$

Analysis of regret for explore-then-commit

Play 2N rounds of estimation and then commit to one coin for T-2N rounds of exploitation:

- Gap: $\Delta = \mu^* - \mu_i$
- At the first 2N rounds, we always trivially accumulate $N\Delta$ in regret (not knowing the value).
- Now we need to estimate how much regret we can accumulate in the exploit phase.

Analysis of regret for explore-then-commit

Estimate how much regret we can accumulate in the exploit phase. Two cases:

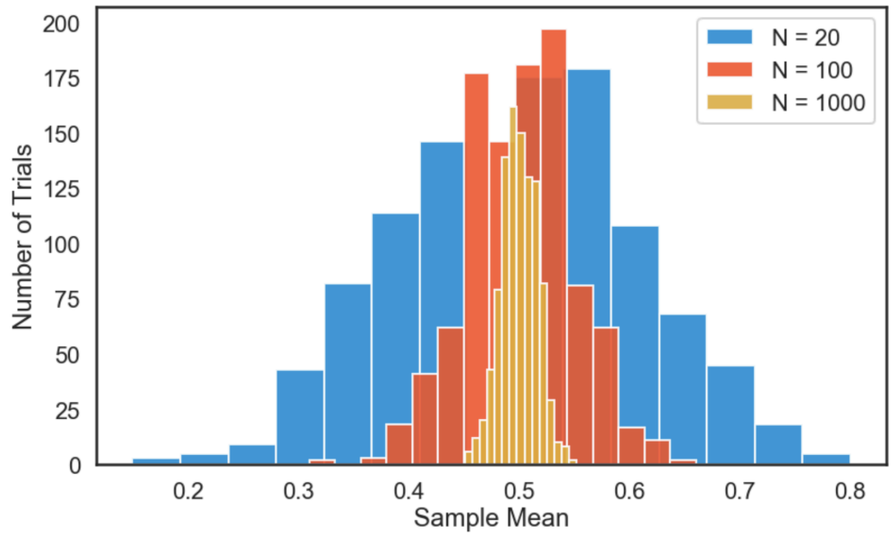
- The normal case: the estimates on the two coins both lie in the the confidence radius (with high probability this happens)

Analysis of regret for explore-then-commit

Estimate how much regret we can accumulate in the exploit phase. Two cases:

- The normal case: the estimates on the two coins both lie in the the confidence radius (with high probability this happens)
- The unlikely case: the estimates are not both in the confidence radius (we will choose parameters such that this happens with negligible probability)

Overall regret



$$R_\sigma(T) = \mathbb{E}_\sigma[R(T)\,|\,\text{concentrate}] \cdot P[\text{concentrate}]$$

the normal case: high probability

$$+\mathbb{E}_\sigma[R(T)\,|\,\neg\text{concentrate}] \cdot P[\neg\text{concentrate}]$$

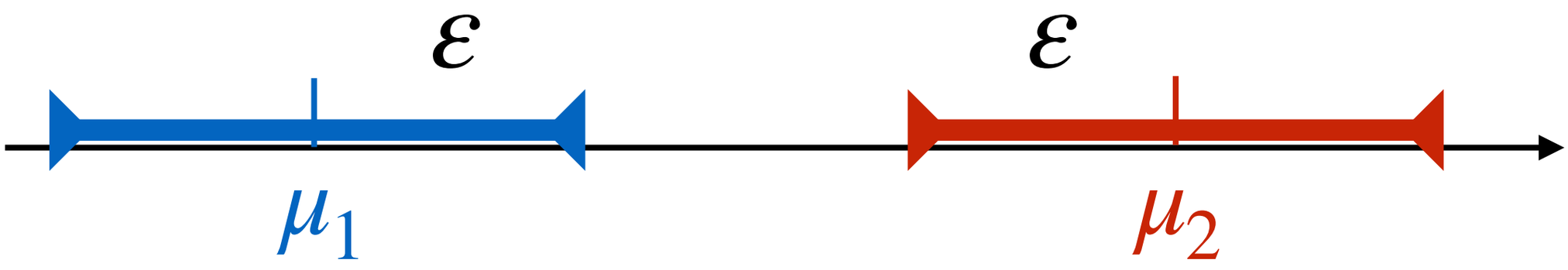
the weird case: low probability

Analysis of regret in the normal case

Suppose after N rounds, the estimates have concentrated properly.

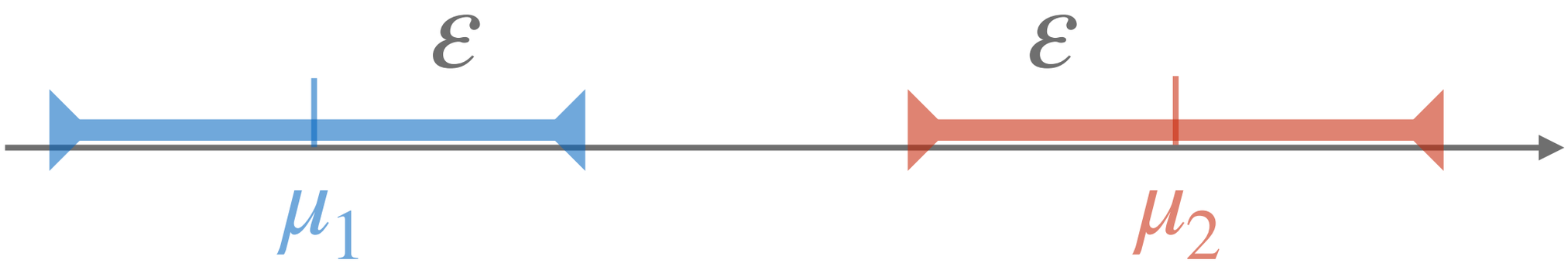
- If the two real means are far apart, because of concentration, we will be able to identify the better one and have no regret.
- We can only accumulate regret when the two means are close to each other, so that we mis-identify. But in this case the gap should be small.

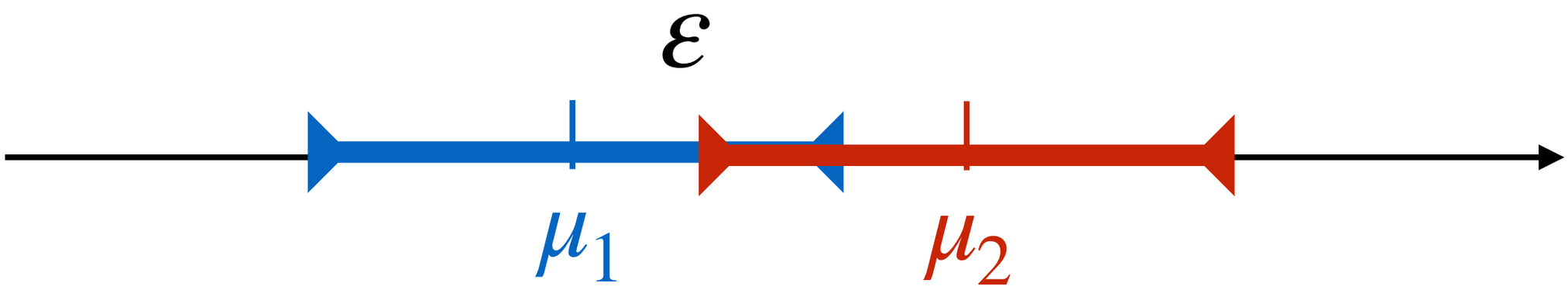
Analysis of regret in the normal case



will identify, no regret

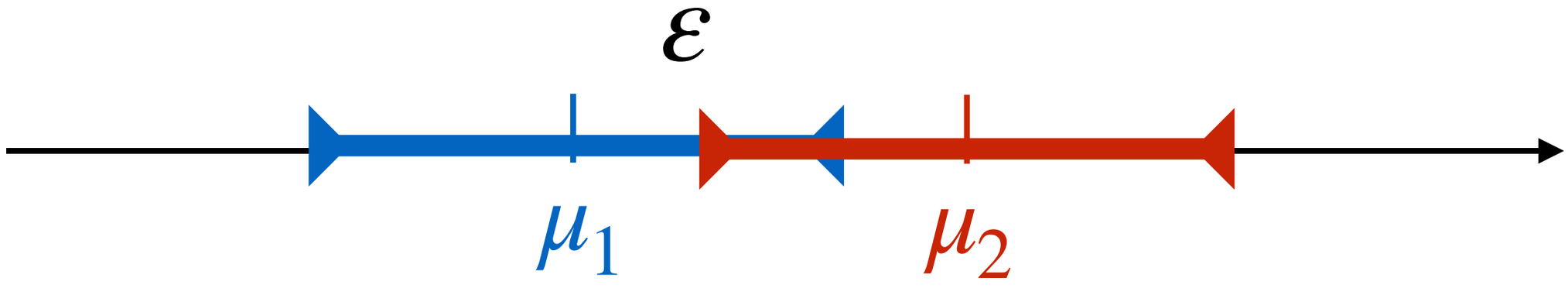
Analysis of regret in the normal case





can mis-identify, bounded regret

Analysis of regret in the normal case



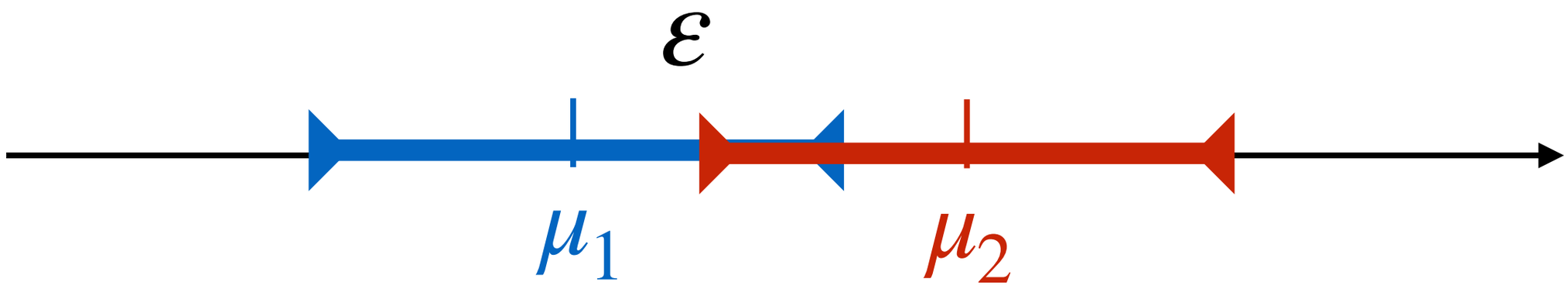
Misidentification can only happen when $\Delta = \mu_2 - \mu_1 \leq 2\varepsilon$

$$R(T) = N\Delta + (T - 2N)\Delta$$

$$\leq N + 2\varepsilon(T - 2N)$$

Now, recall the choice of $\varepsilon = \sqrt{\frac{c \log T}{N}}$

Analysis of regret in the normal case



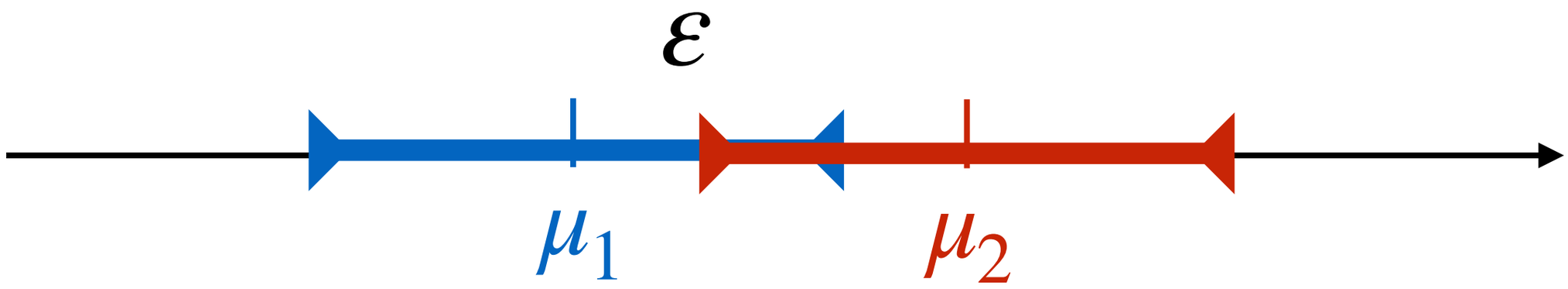
$$R(T) \leq N + 2\varepsilon(T - 2N)$$

$$= N + 2\sqrt{\frac{c \log T}{N}}(T - 2N)$$

$$\varepsilon = \sqrt{\frac{c \log T}{N}}$$

Goal: choose a good N to make R(T) small

Analysis of regret in the normal case



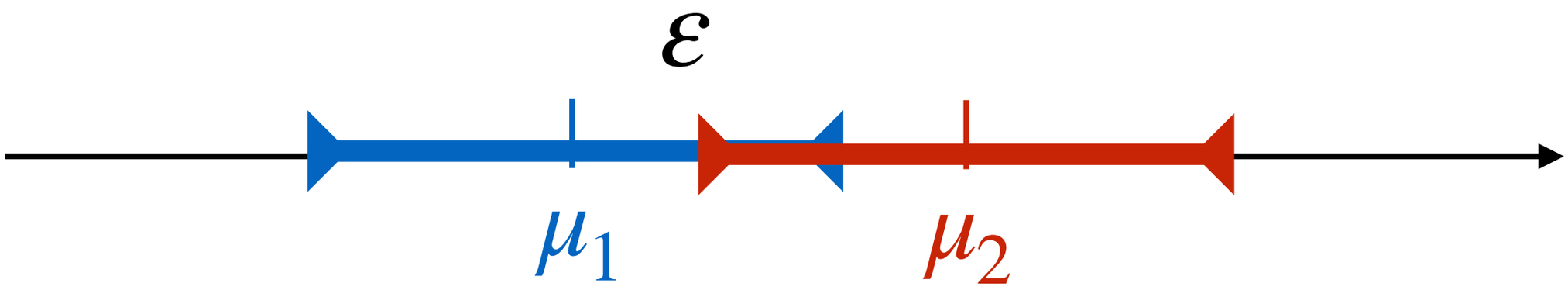
$$R(T) \leq N + 2\varepsilon(T - 2N)$$

$$= N + 2\sqrt{\frac{c \log T}{N}}(T - 2N)$$

$$\leq N + c'\sqrt{\frac{\log T}{N}} \cdot T$$

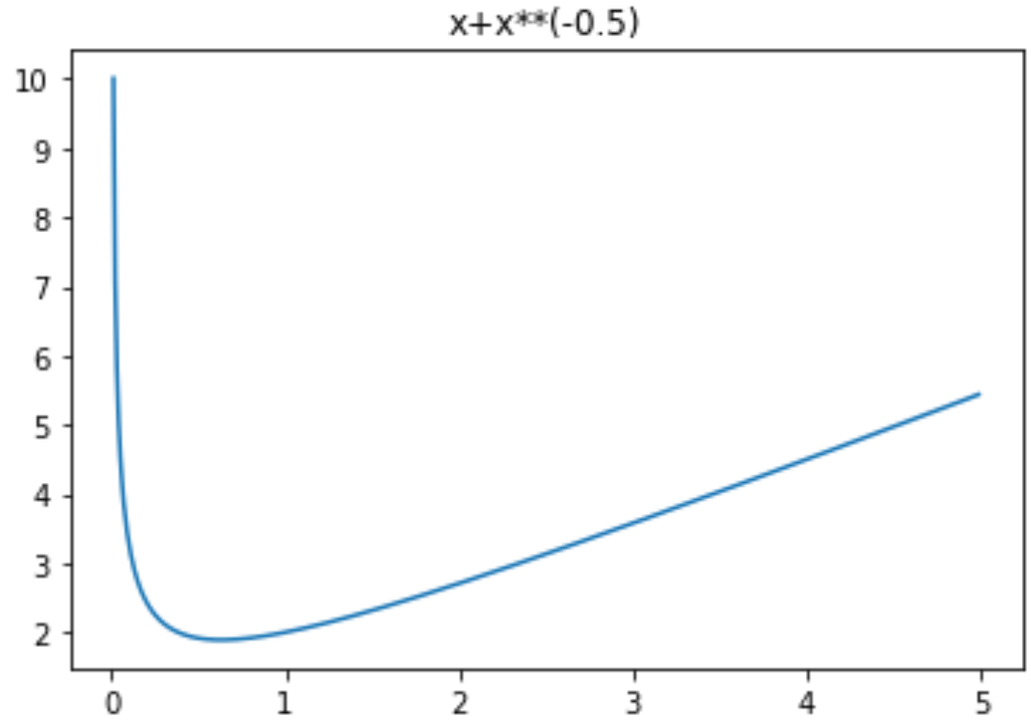
$$\varepsilon = \sqrt{\frac{c \log T}{N}}$$

Analysis of regret in the normal case

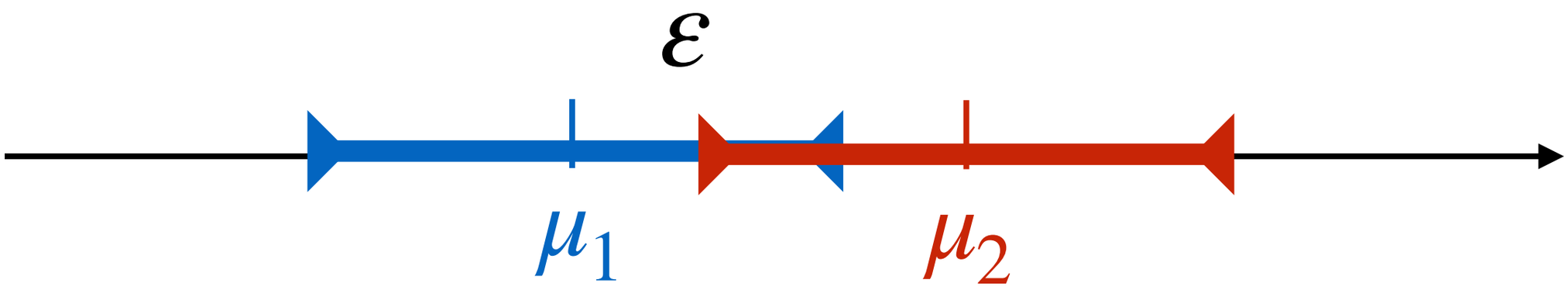


$$R(T) \leq N + c' \sqrt{\frac{\log T}{N}} \cdot T$$

How should we choose N that minimizes this bound?



Analysis of regret in the normal case

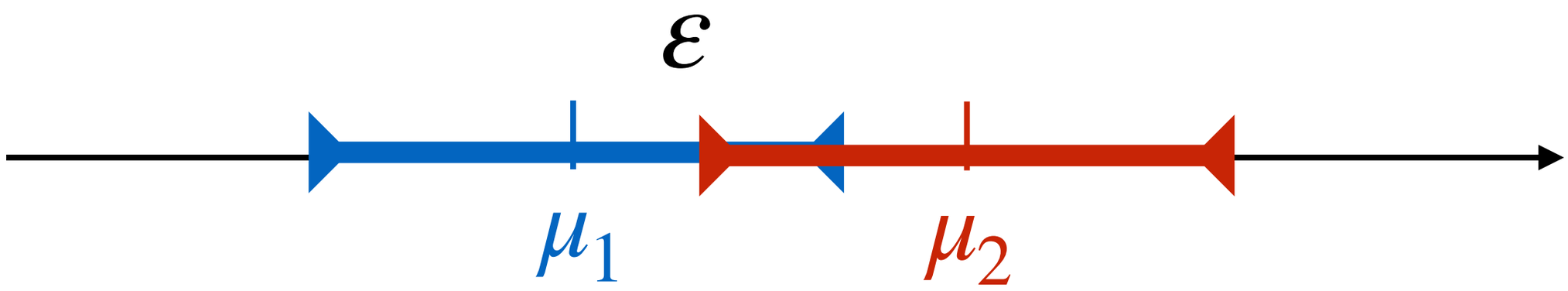


$$R(T) \leq N + c' \sqrt{\frac{\log T}{N}} \cdot T$$

Take the following N that minimizes R(T)

$$N = c'' \cdot T^{\frac{2}{3}} \log T^{\frac{1}{3}}$$

Analysis of regret in the normal case



$$R(T) \leq N + c' \sqrt{\frac{\log T}{N}} \cdot T$$

Using $N = c'' \cdot T^{\frac{2}{3}} \log T^{\frac{1}{3}}$, we have

$$R(T) \sim O\left(T^{\frac{2}{3}} (\log T)^{\frac{1}{3}}\right)$$

Overall regret

$$R_\sigma(T) = \mathbb{E}[R(T)\,|\,\text{concentrate}] \cdot P[\text{concentrate}]$$

$$+\mathbb{E}[R(T)\,|\,\neg\text{concentrate}] \cdot P[\,\neg\text{concentrate}]$$

$$\leq \mathbb{E}[R(T)\,|\,\text{concentrate, mis-identify}] + \frac{1}{T^{2c}} \cdot T$$

$$\sim O\left(T^{\frac{2}{3}}(\log T)^{\frac{1}{3}}\right)$$

choose, say, $c = 2$

For tighter results, check "On Explore-Then-Commit Strategies" NIPS'17

Overall regret

$$R_\sigma(T) = \mathbb{E}[R(T)\,|\,\text{concentrate}] \cdot P[\text{concentrate}]$$

$$+\mathbb{E}[R(T)\,|\,\neg\text{concentrate}] \cdot P[\,\neg\text{concentrate}]$$

$$\leq \mathbb{E}[R(T)\,|\,\text{concentrate, mis-identify}] + \frac{1}{T^{2c}} \cdot T$$

$$\sim O\left(T^{\frac{2}{3}}(\log T)^{\frac{1}{3}}\right)$$

choose, say, $c = 2$

In the case of K arms, similar reasoning gives $O\left(T^{\frac{2}{3}}(K \log T)^{\frac{1}{3}}\right)$

For tighter results, check "On Explore-Then-Commit Strategies" NIPS'17

Simple strategy: epsilon-greedy

For any round t:

- Compare the average reward $\bar{\mu}_1$ and $\bar{\mu}_2$
- With probability $1 - \varepsilon$, play the one with better empirical mean. With ε, play the other one.

Simple strategy: epsilon-greedy

- For epsilon-greedy, the analysis is the same as explore-then-commit, where the bound

$$R(T) \leq N + c\sqrt{\frac{\log T}{N}} \cdot T$$

is changed to

$$R(t) \leq \varepsilon t + c\sqrt{\frac{\log t}{\varepsilon t}} \cdot t$$

Simple strategy: epsilon-greedy

- Following similar analysis, we know

$$R(t) \leq \varepsilon t + c\sqrt{\frac{\log t}{\varepsilon t}} \cdot t$$

achieves $O(t^{\frac{2}{3}}(\log t)^{\frac{1}{3}})$ when $\varepsilon = c \cdot t^{-\frac{1}{3}}(\log t)^{\frac{1}{3}}$

- So in general epsilon-greedy achieves regret

$$R_\sigma(t) \sim O(t^{\frac{2}{3}}(K \log t)^{\frac{1}{3}})$$

Upper Confidence Bound

For any round t:

- Compute the UCB value of each arm $i \in [K]$

$$B(i, t, n_i(t)) = \bar{X}_i + \sqrt{\frac{2 \log t}{n_i(t)}}$$

- Play the arm with $\max_i B(i, t, n_i(t))$

What happens in the first K rounds?

Recap of key definitions

- K arms, each with a reward distribution in [0,1]
- We look for a strategy that chooses the next arm to play based on past plays and rewards
- Expected regret after t rounds

$$R_\sigma(t) = \mu_{i*}t - \mu_i \sum_{i=1}^{K} \mathbb{E}_\sigma[n_i(t)]$$

where $\mu_{i*} = \max_{i \in [k]} \mu_i$ and $n_i(t)$ is the #play of arm i

- Gap $\Delta_i = \mu_{i*} - \mu_i$

Upper Confidence Bound

- Explore-then-commit and epsilon-greedy are not efficient enough because each arm is played for at least a fixed number of times.
- UCB makes it possible to adapt the number of plays for each arm based on actual performance.
- In this way a lot of regret can be avoided. How much, precisely?

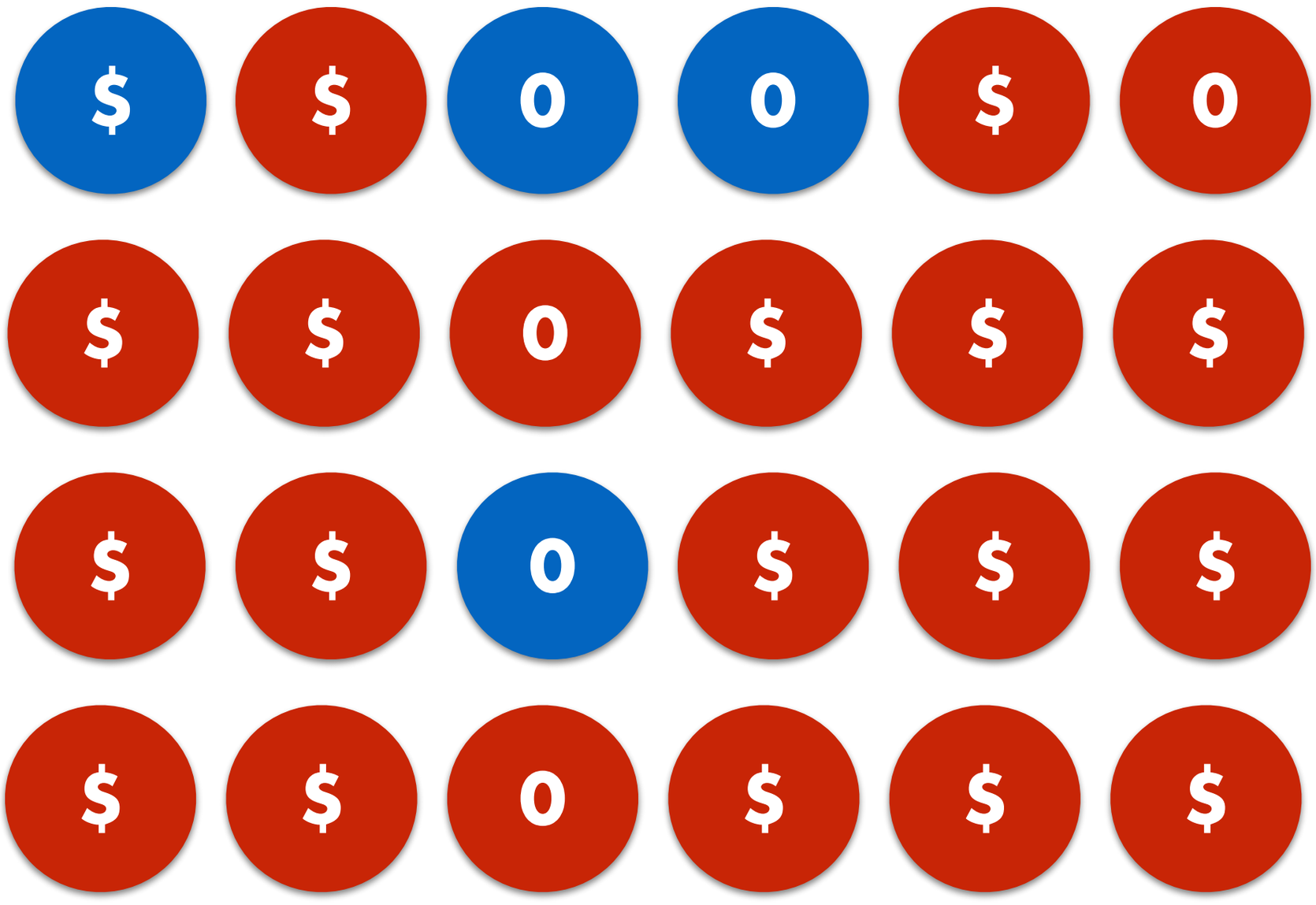
Exponentially less bad plays

- Theorem: Using the UCB formula, in t rounds (t is reasonably large), each suboptimal arm i is played, in expectation:

$$\mathbb{E}[n_i(t)] \leq \frac{8 \log t}{\Delta_i^2} + O(1)$$

- That is, each suboptimal arm should be played exponentially less than the optimal arm.

Exponentially less bad plays

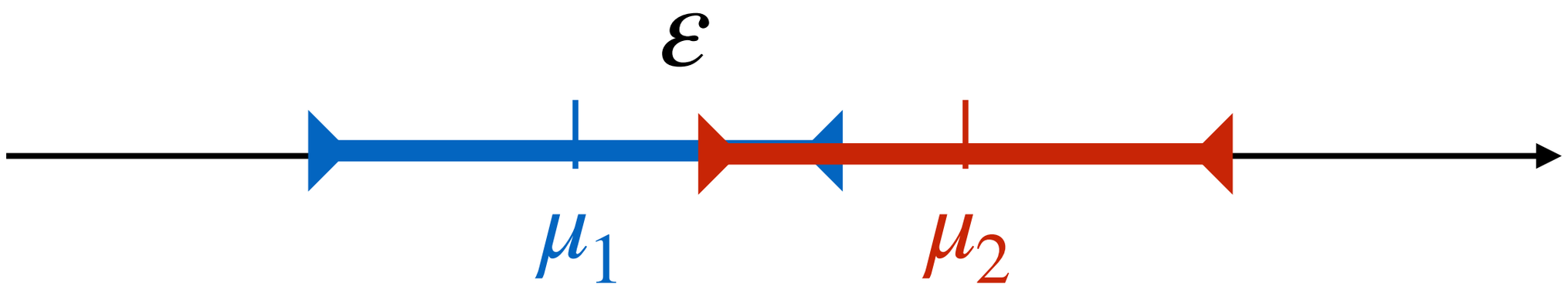


Exponentially less bad plays

- Where does this bound come from?

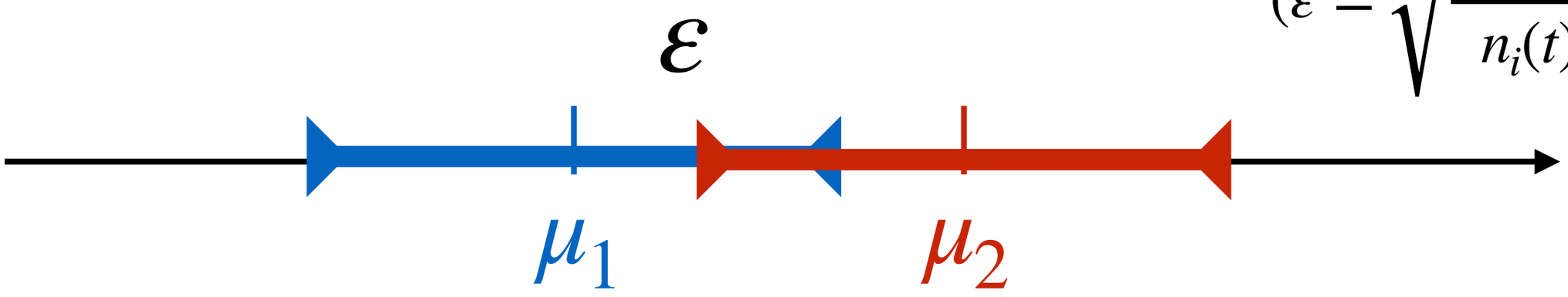
$$\mathbb{E}[n_i(t)] \leq \frac{8 \log t}{\Delta_i^2} + O(1)$$

- Recall the condition for identifying a bad arm:



Exponentially less bad plays

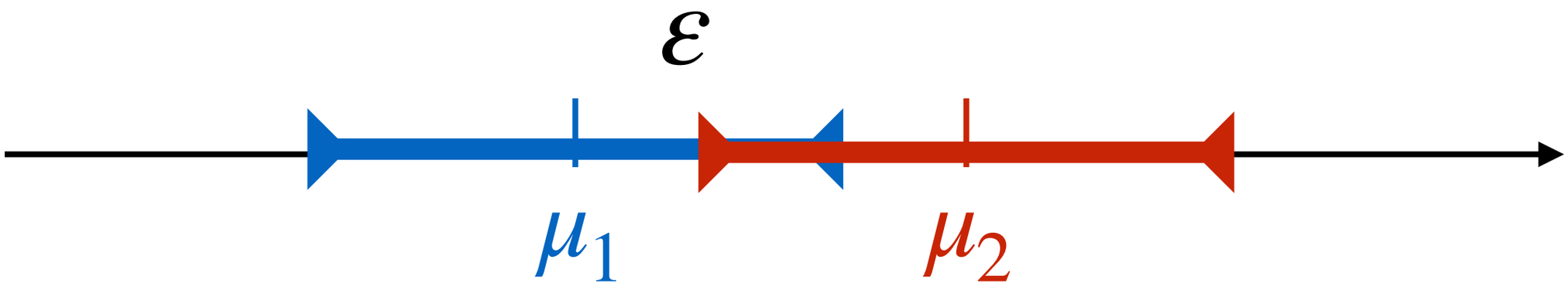
$$(\varepsilon = \sqrt{\frac{2\log t}{n_i(t)}})$$



- Consider normal concentration. Following the UCB formula, to play suboptimal i at round t, we need to see

$$\bar{\mu}_{i,n_i(t)} + \sqrt{\frac{2\log t}{n_i(t)}} \geq \bar{\mu}_{i*,n_{i*}(t)} + \sqrt{\frac{2\log t}{n_{i*}(t)}}$$

Exponentially less bad plays

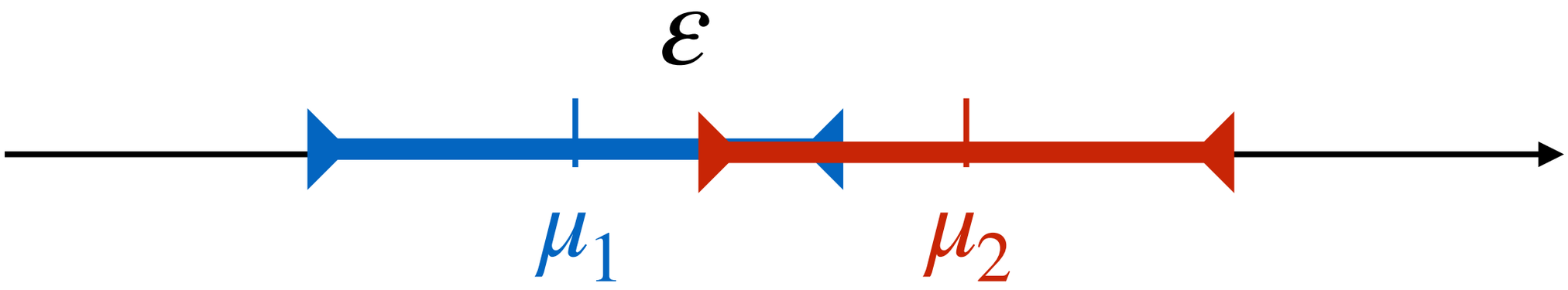


- Consider normal concentration. Following the UCB formula, to play suboptimal i at round t, we need to see

$$\bar{\mu}_{i,n_i(t)} + \sqrt{\frac{2\log t}{n_i(t)}} \geq \bar{\mu}_{i*,n_{i*}(t)} + \sqrt{\frac{2\log t}{n_{i*}(t)}}$$

$$\leq \mu_i + \sqrt{\frac{2\log t}{n_i(t)}} \qquad \geq \mu_{i*}$$

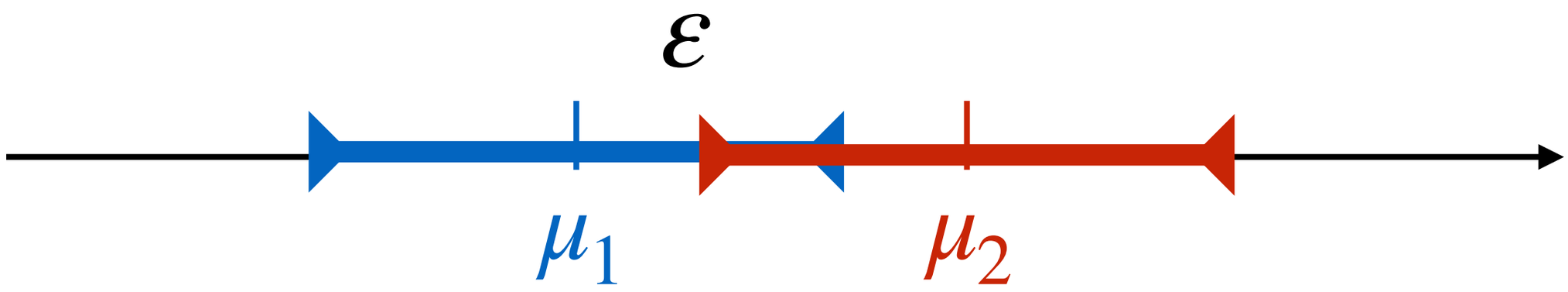
Exponentially less bad plays



- Consider normal concentration. Following the UCB formula, to play suboptimal i at round t, we need to see

$$\mu_i + 2\sqrt{\frac{2\log t}{n_i(t)}} \geq \mu_{i*}$$

Exponentially less bad plays



- So we need to see

$$\mu_i + 2\sqrt{\frac{2\log t}{n_i(t)}} \geq \mu_{i*}$$

which can happen only when

$$n_i(t) \leq \frac{8\log t}{(\mu_{i*} - \mu_i)^2} = \frac{8\log t}{\Delta_i^2}$$

Exponentially less bad plays

- So the argument is, after playing arm i for

$$n_i(t) \geq \frac{8 \log t}{\Delta_i^2}$$

The gap between i and i^* can be identified, and it should be very unlikely that arm i can be played again. Formally,

$$\mathbb{E}(n_i(t)) \leq \frac{8 \log t}{\Delta_i^2} + \sum_{s=\frac{8 \log t}{\Delta_i^2}+1}^{t} \mathbb{E}[\mathbf{1}\{\text{mis-identify}\}] \cdot P(\neg\text{concentrate})$$

"Finite-time Analysis of the Multiarmed Bandit Problem" [Auer 2002]

Upper Confidence Bound

- Since we can bound the suboptimal visits with

$$\mathbb{E}[n_i(t)] \leq \frac{8 \log t}{\Delta_i^2} + O(1)$$

The overall expected regret has bound

$$R_{UCB}(t) = \sum_{i=1}^{K} \Delta_i \cdot \mathbb{E}[n_i(t)] \leq \sum_{i=1}^{K} \frac{8 \log t}{\Delta_i} + O(1)$$

Note: This bound is problem-dependent on Δ_i .

Summary: Upper Confidence Bound

For any round t:

- Compute the UCB value of each arm $i \in [K]$

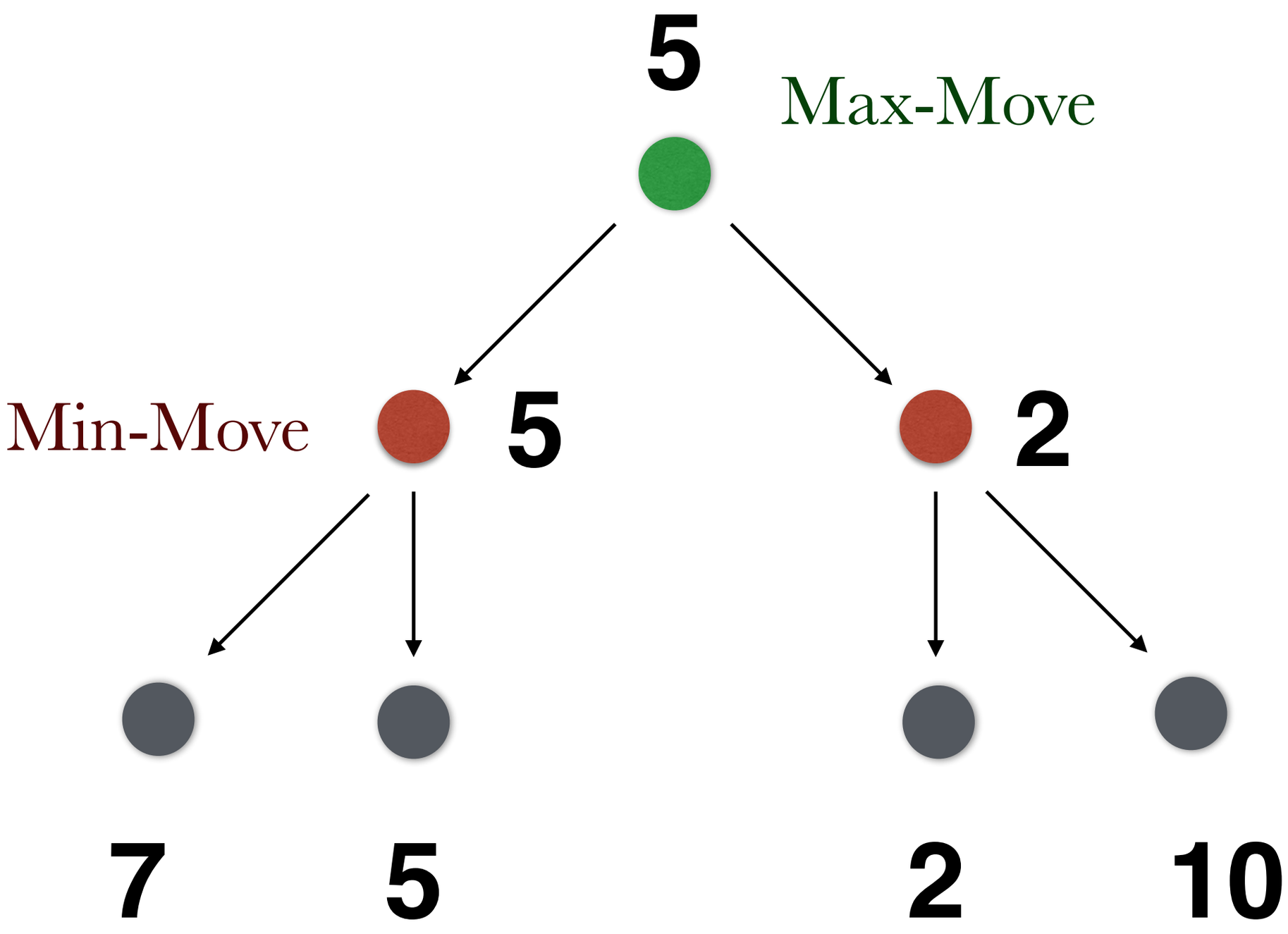
$$B(i, t, n_i(t)) = \bar{X}_i + \sqrt{\frac{2 \log t}{n_i(t)}}$$

- Play the arm with $\max_i B(i, t, n_i(t))$
- We then achieve regret $O(\log t)$

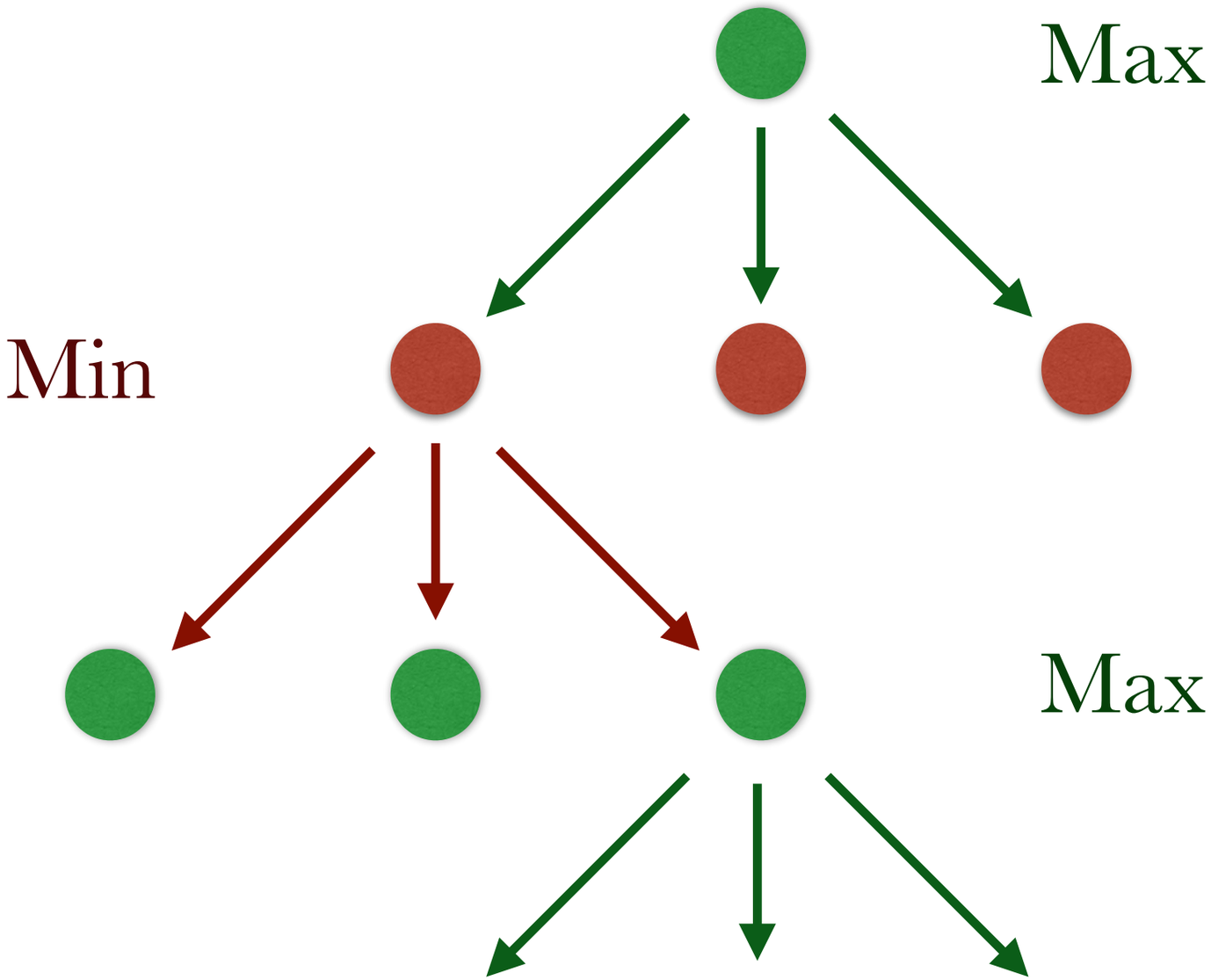
Limitations and extensions

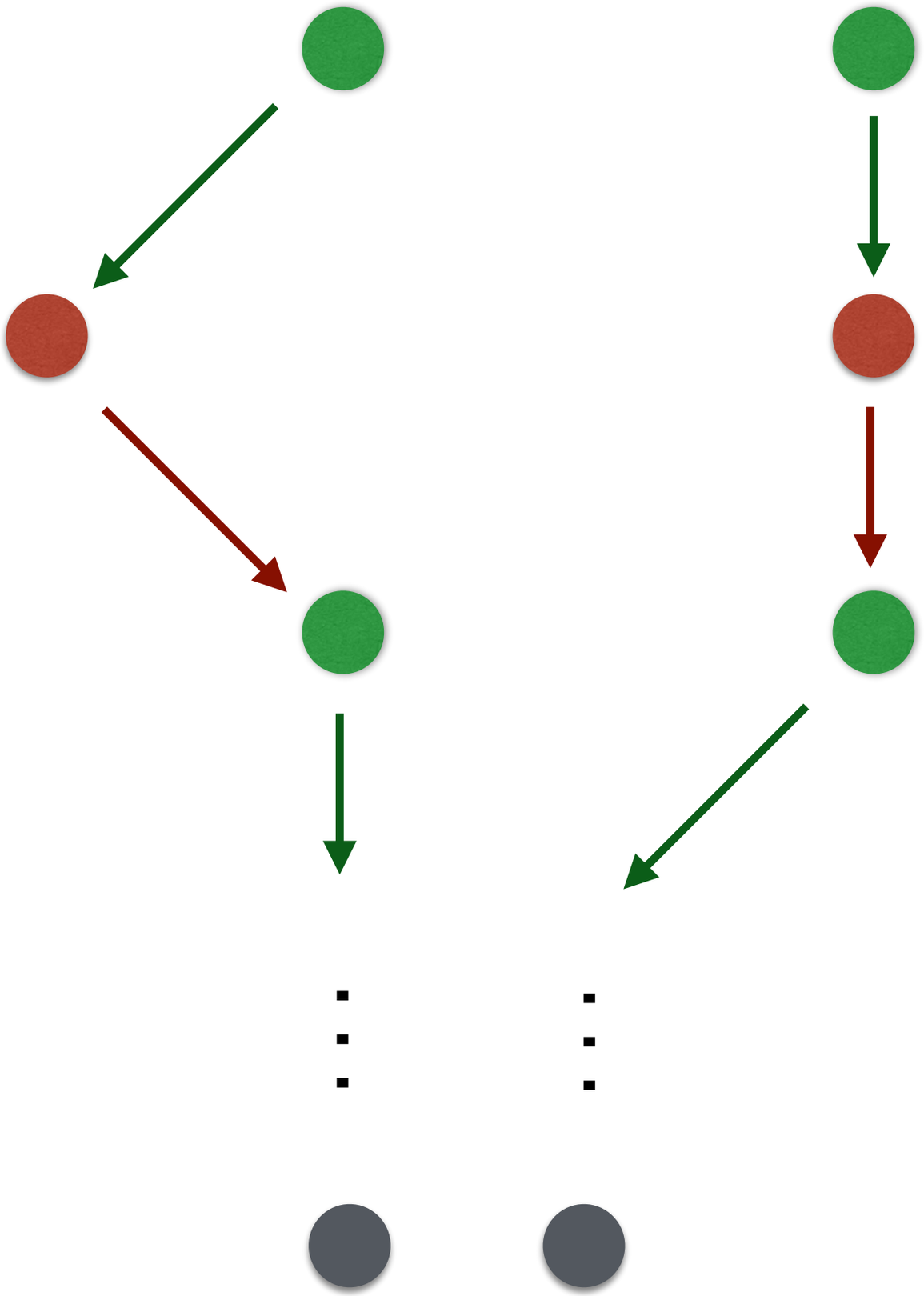
- Stationary distribution
- Contextual bandits
- No long term rewards
- Full observability

Back in adversarial search



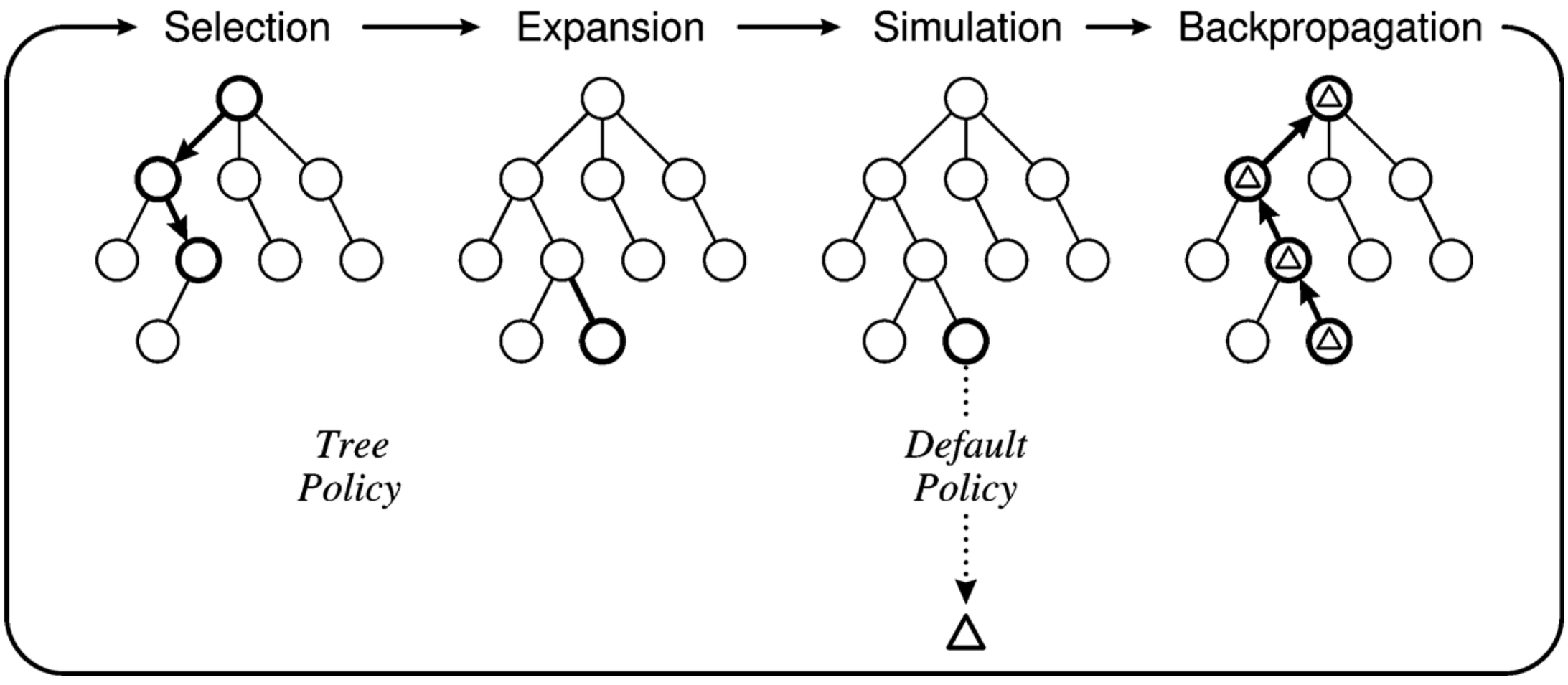
MCTS: How do we grow trees cleverly?





Can we approximate the Minimax values through running a lot of simulations?

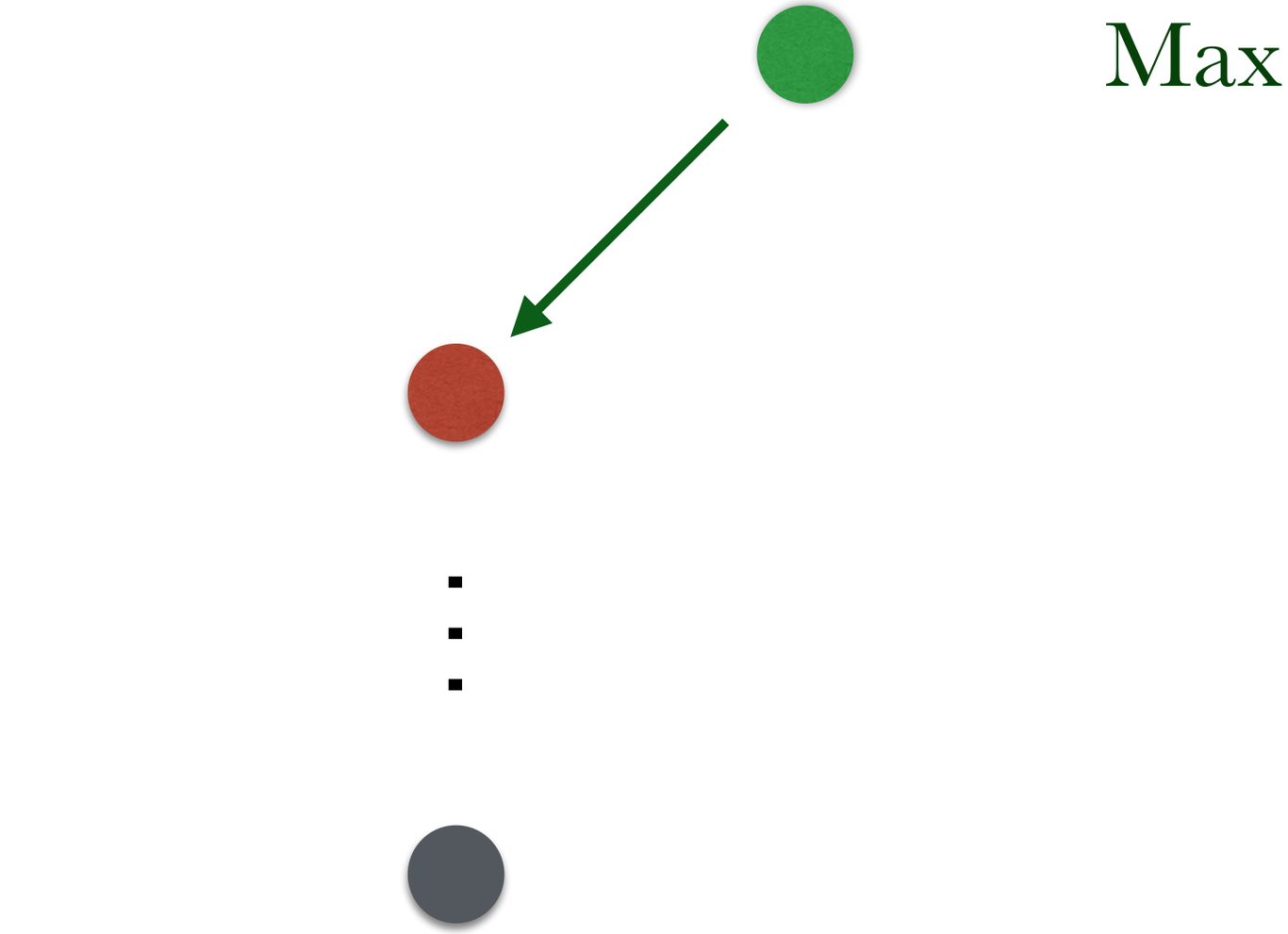
Overall Idea of MCTS



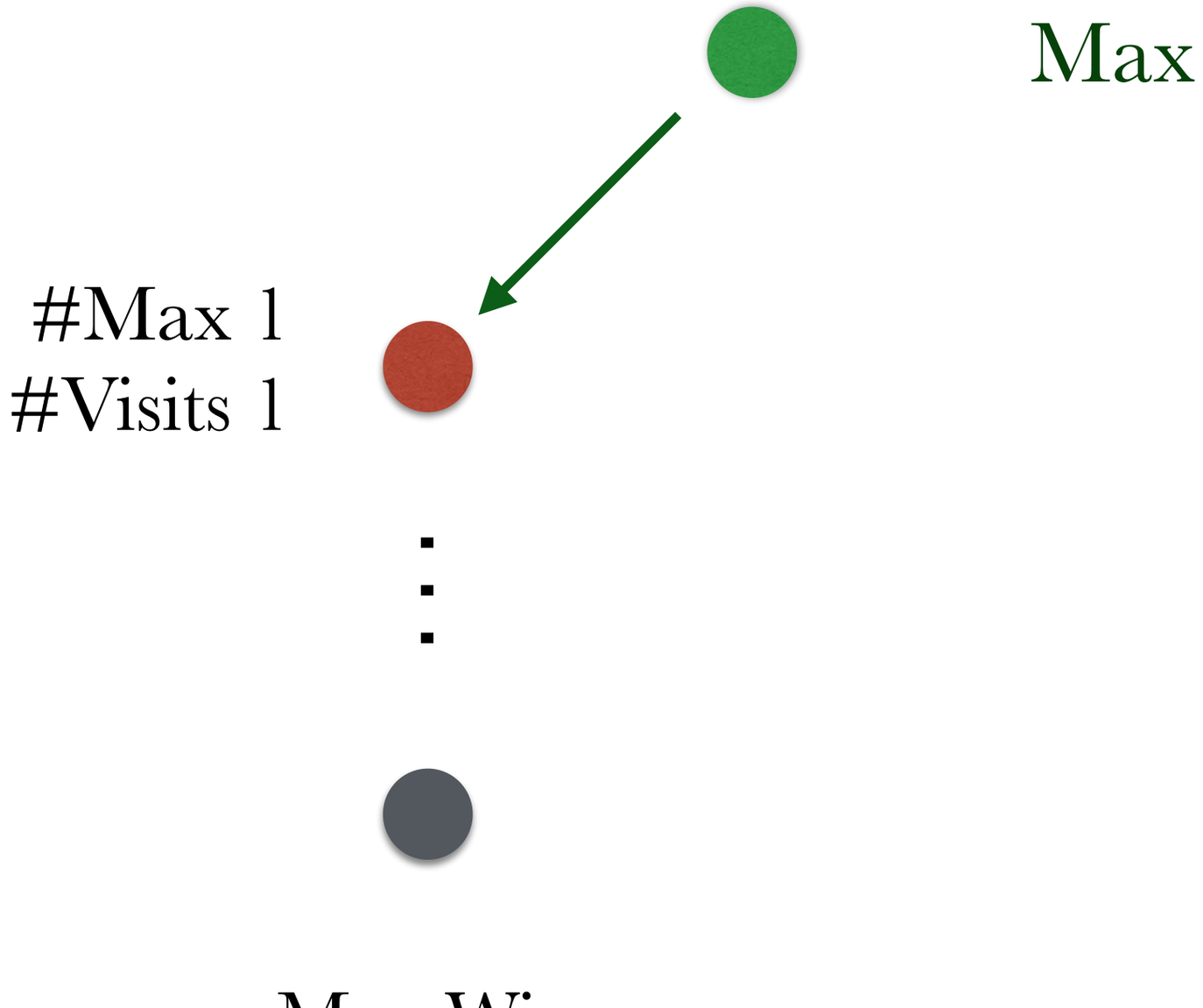
Overall Idea of MCTS

Max

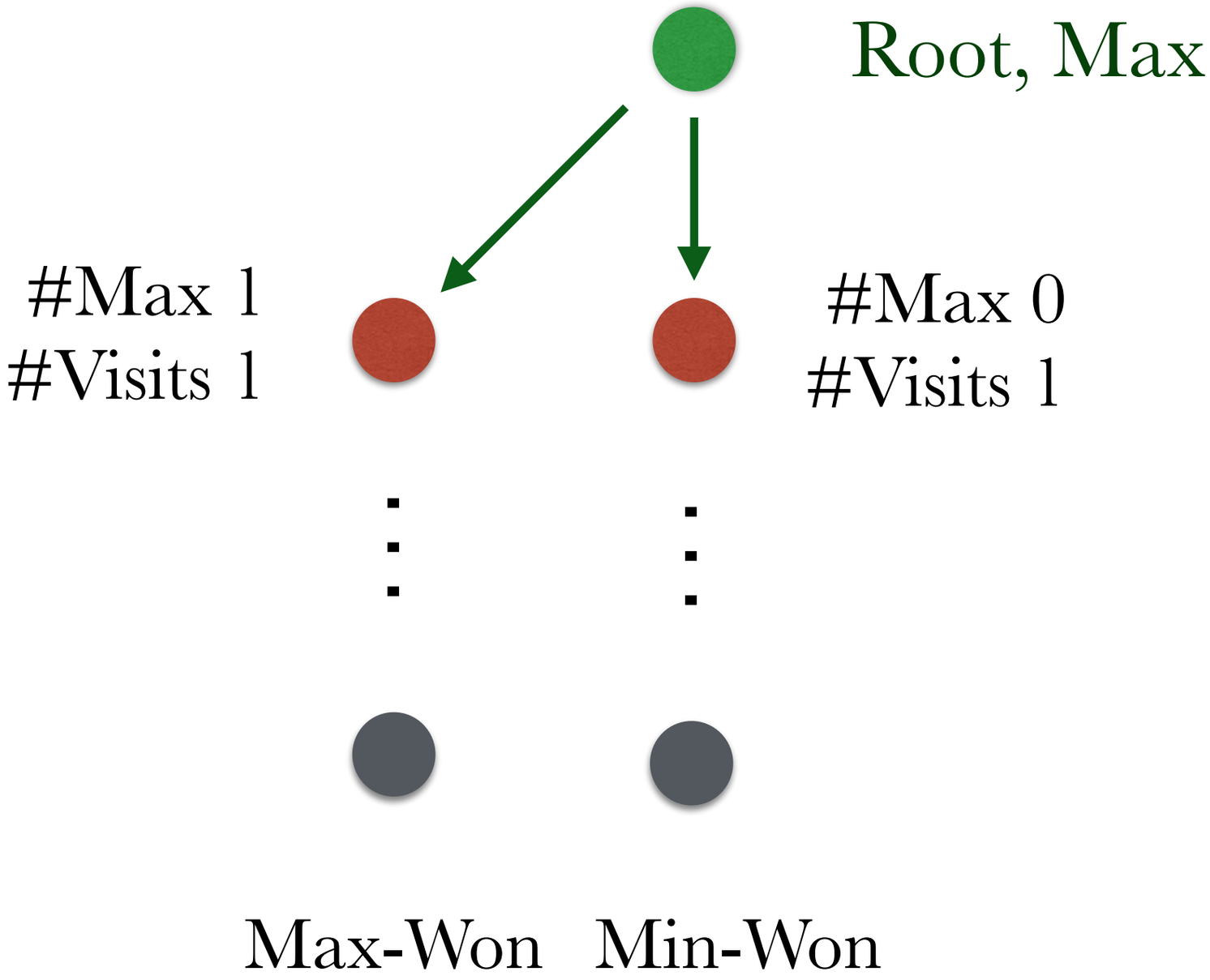
Overall Idea of MCTS



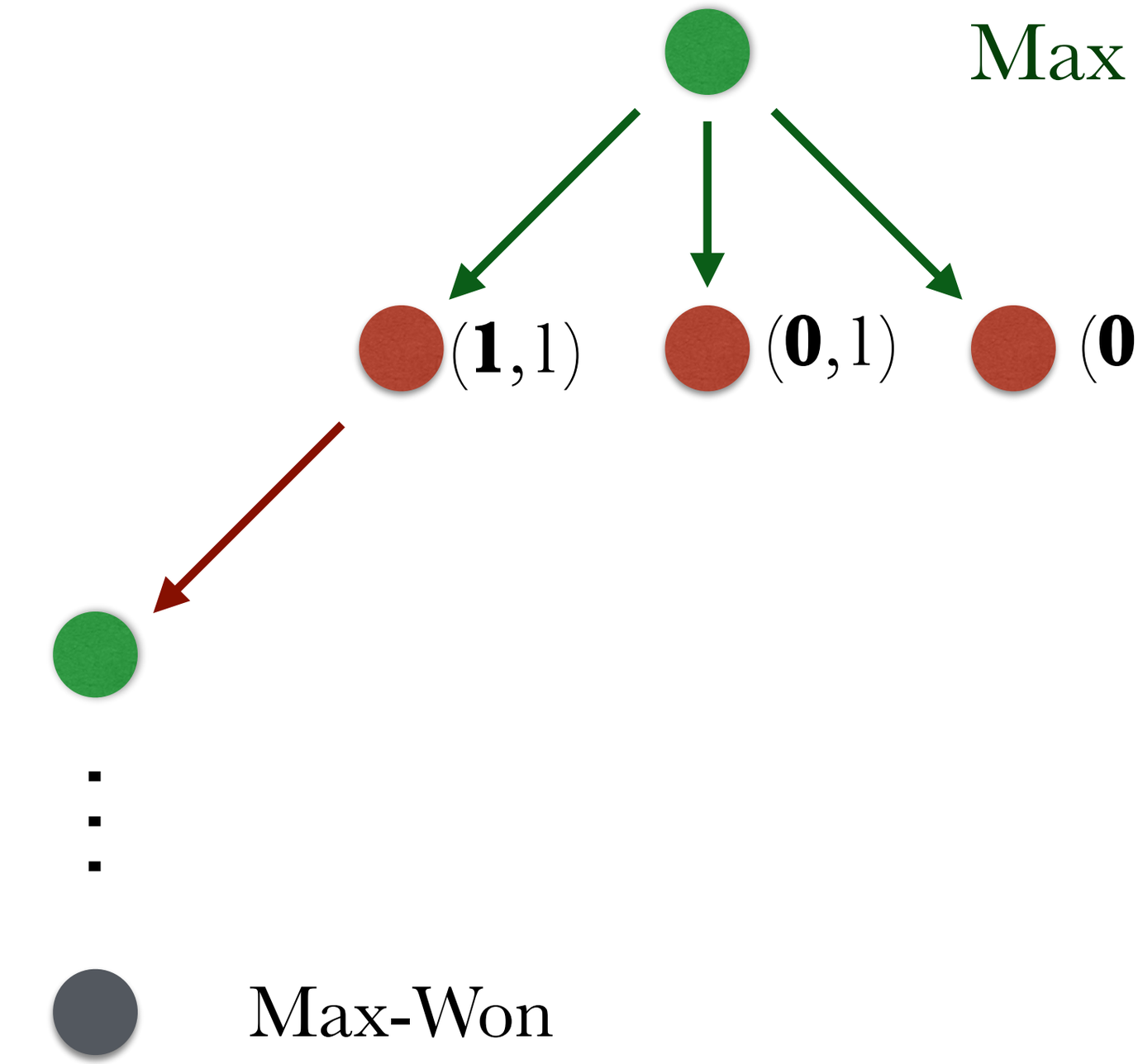
Overall Idea of MCTS



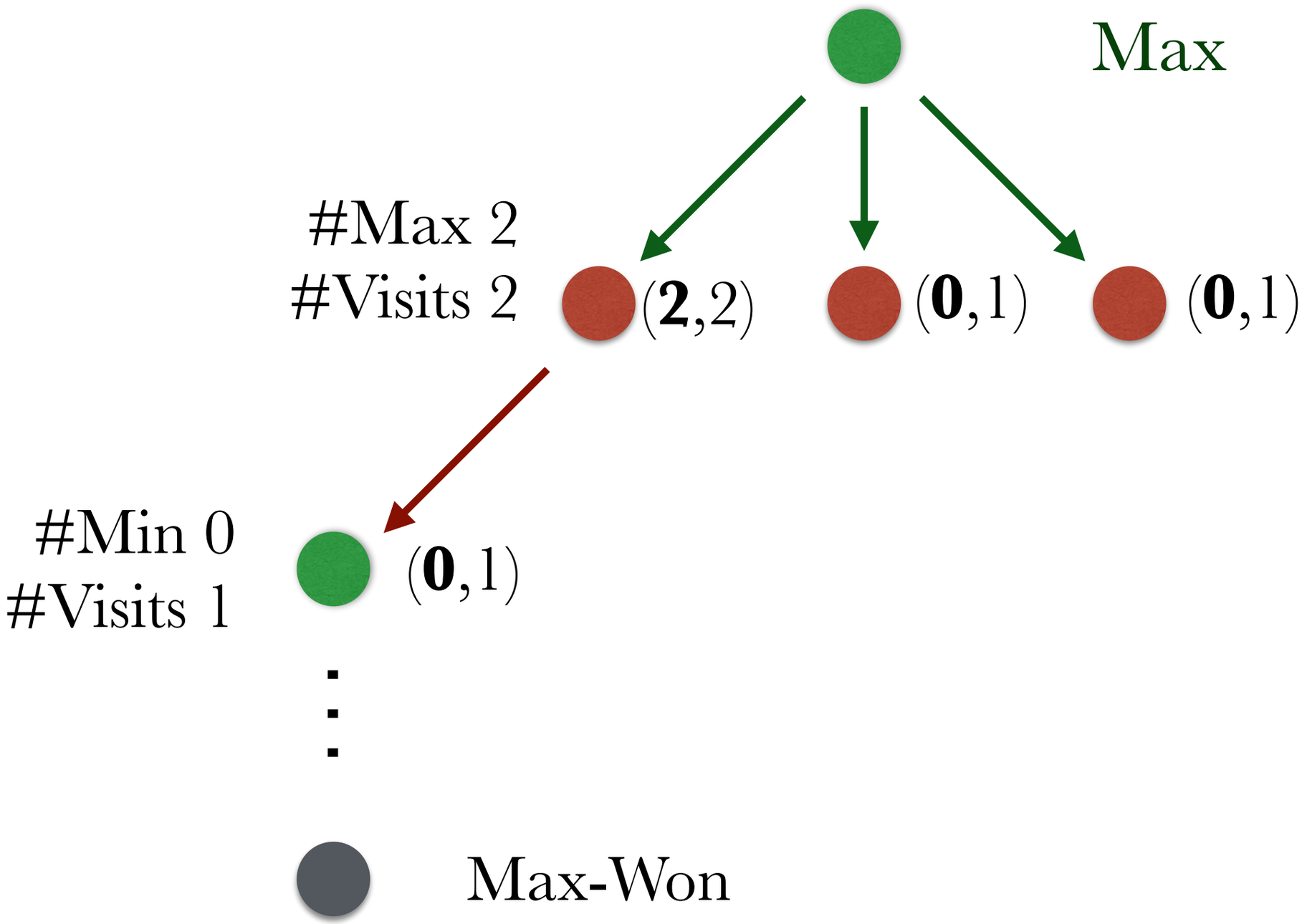
Overall Idea of MCTS



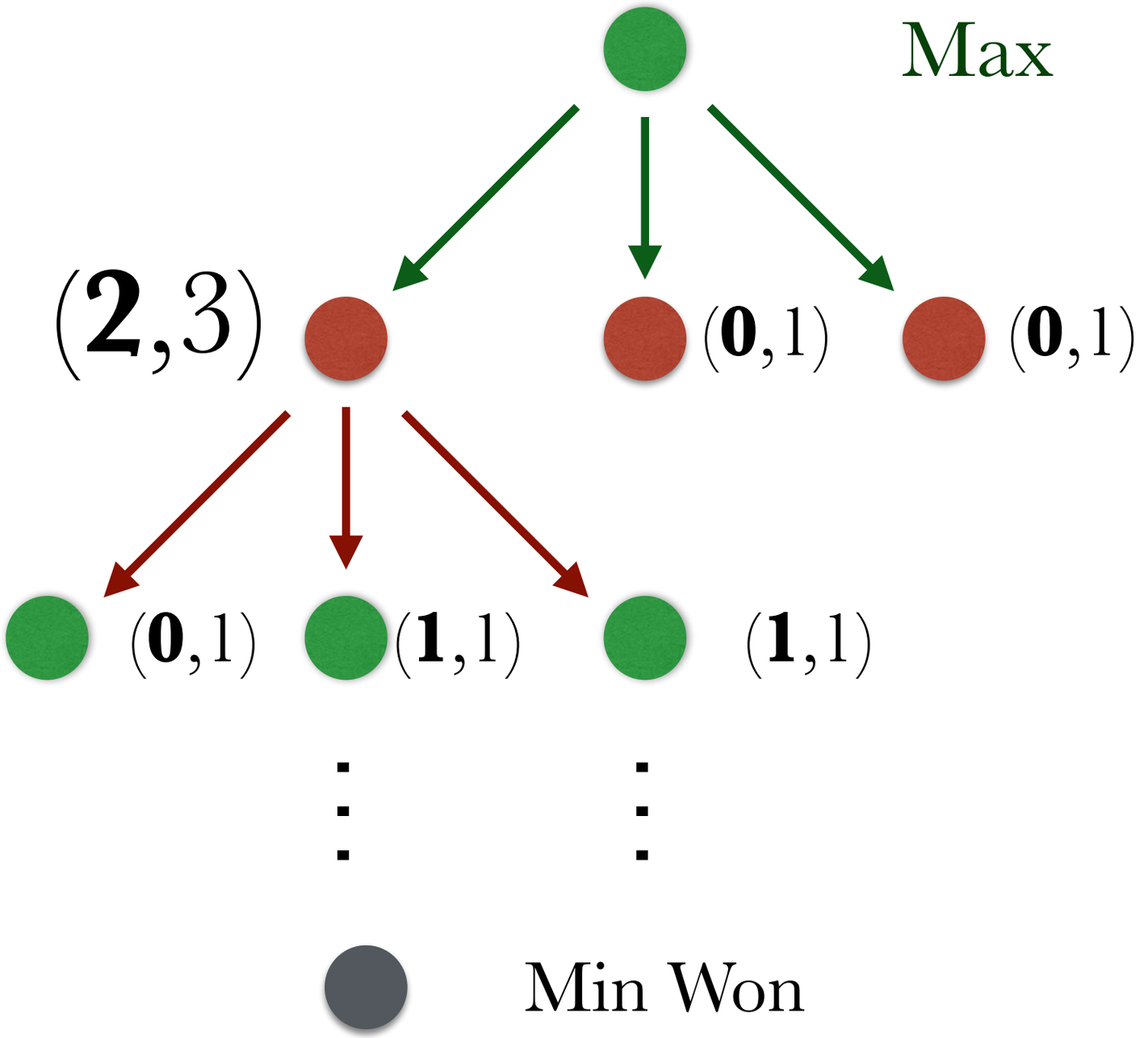
Overall Idea of MCTS



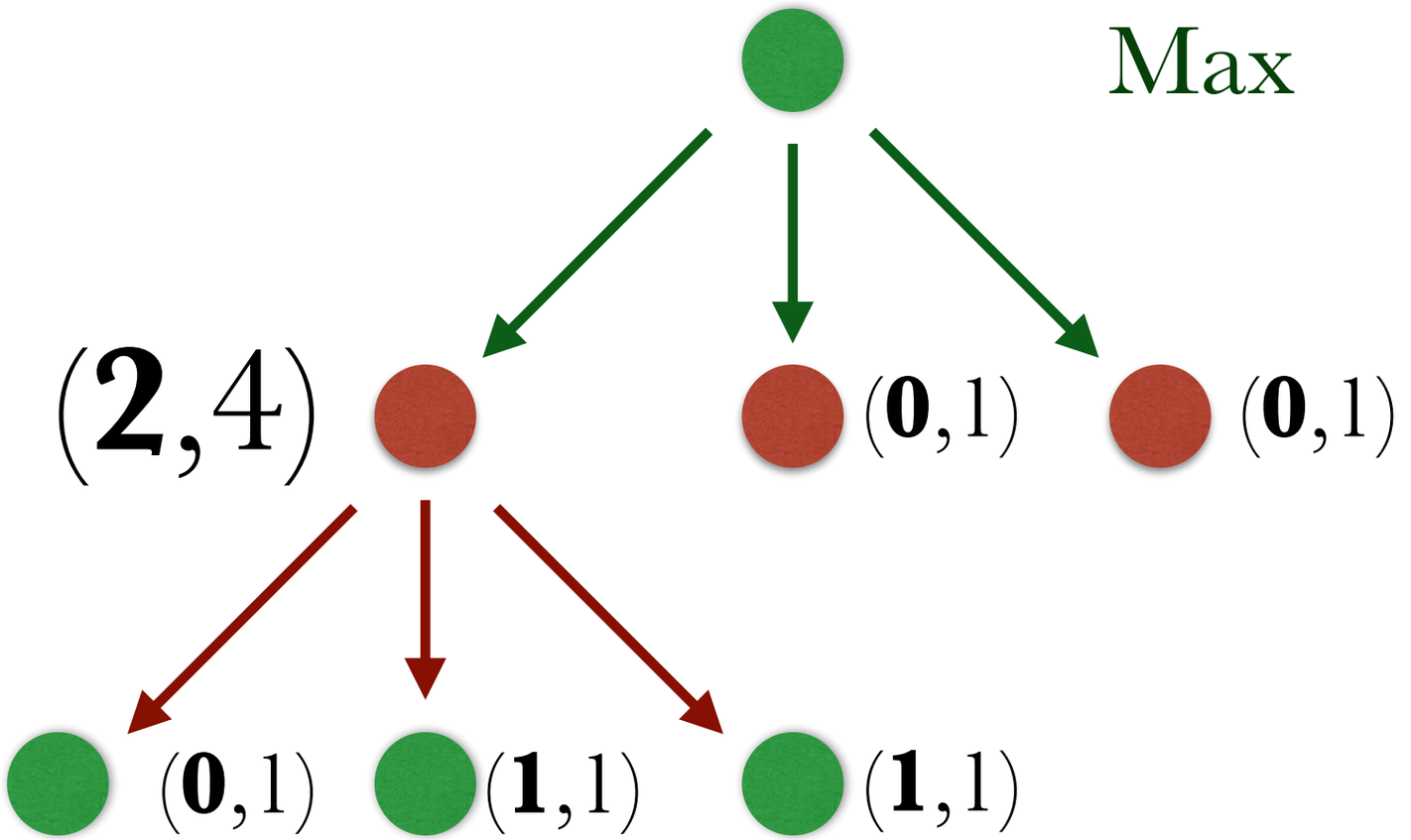
Overall Idea of MCTS



Overall Idea of MCTS

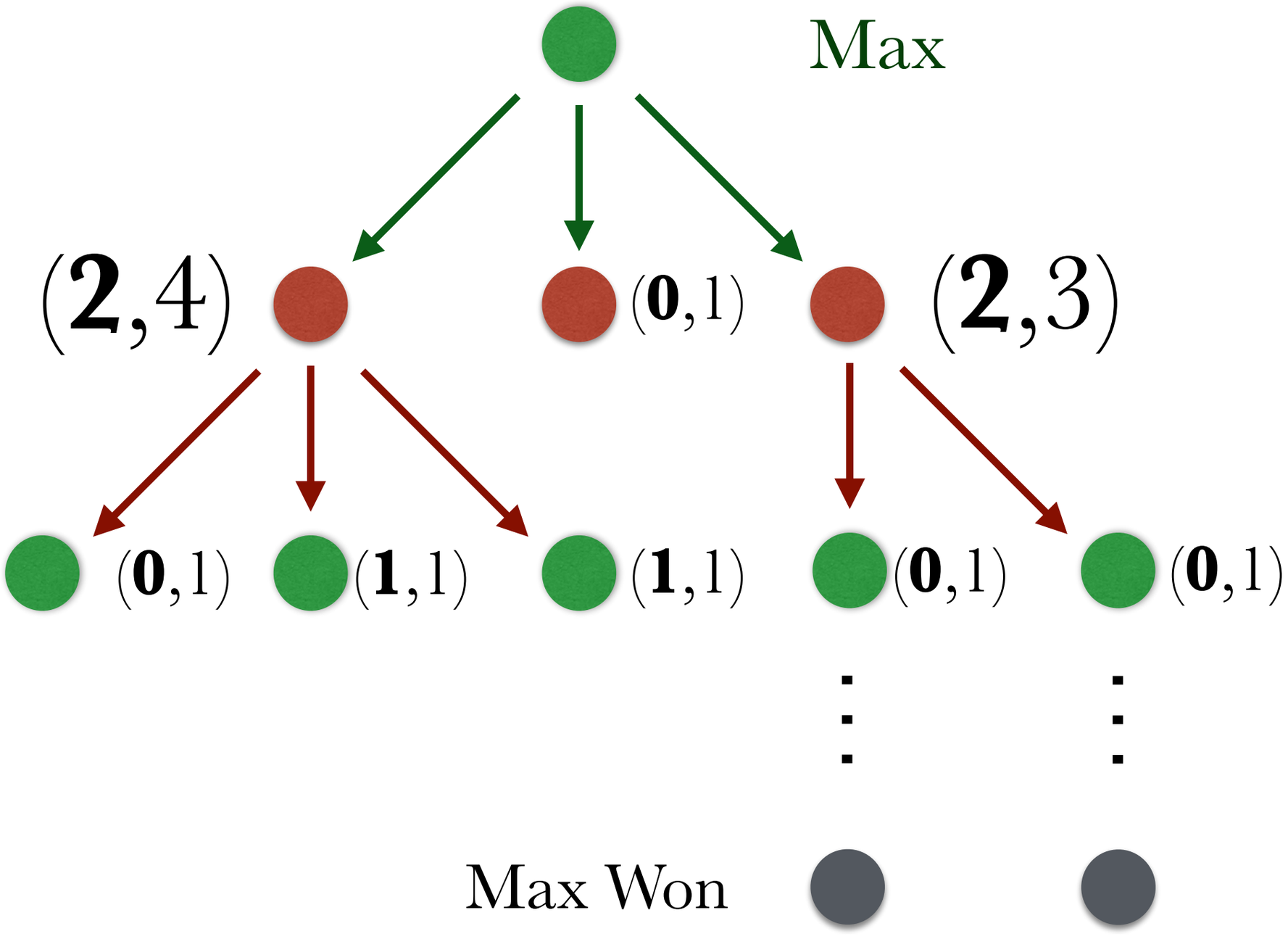


Overall Idea of MCTS



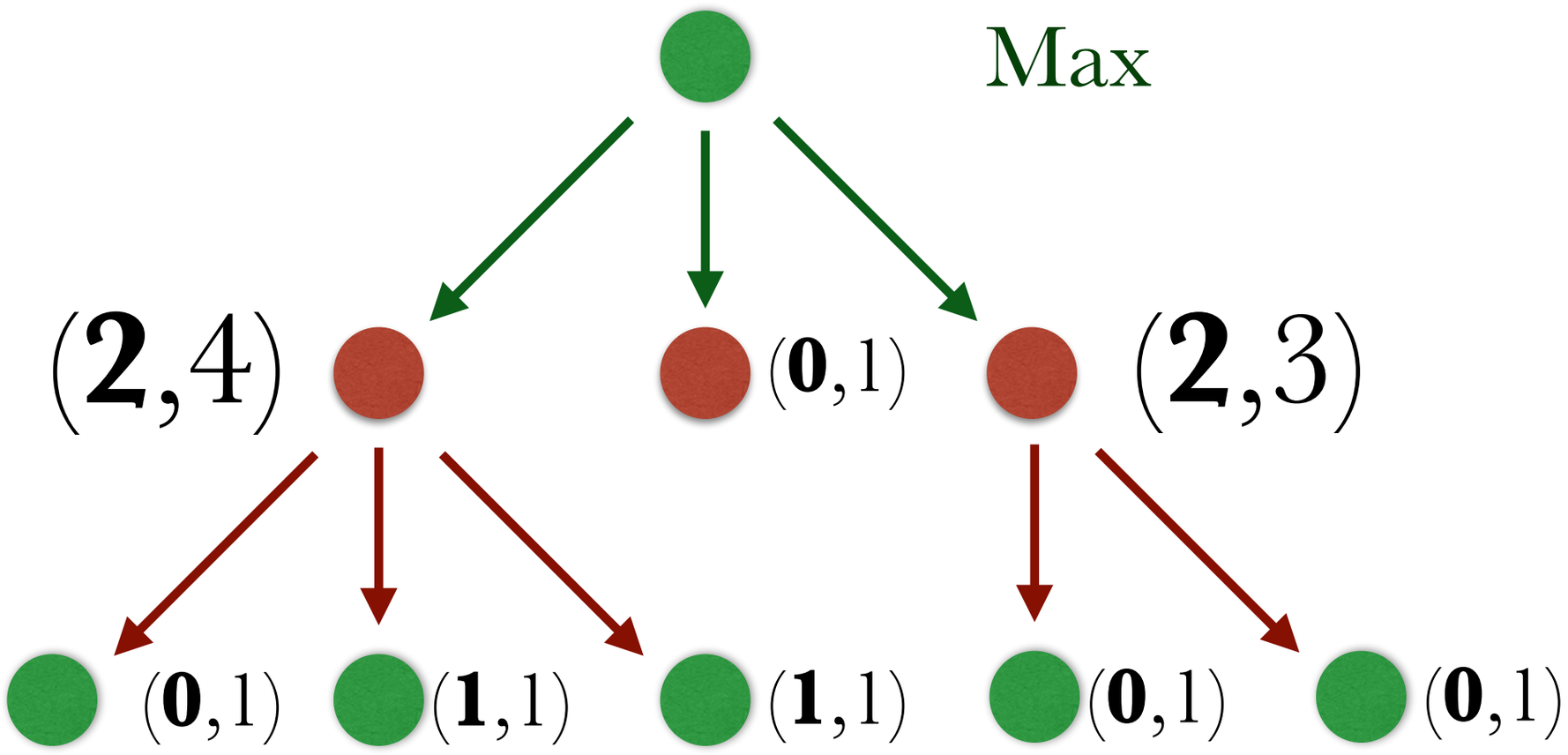
Now it seems worthwhile to look at other options of max?

Overall Idea of MCTS



Exploration vs Exploitation

- How do we decide whether to keep going down into a subtree, or change focus to another one?



Upper Confidence Bound

For any round t:

- Compute the UCB value of each arm k

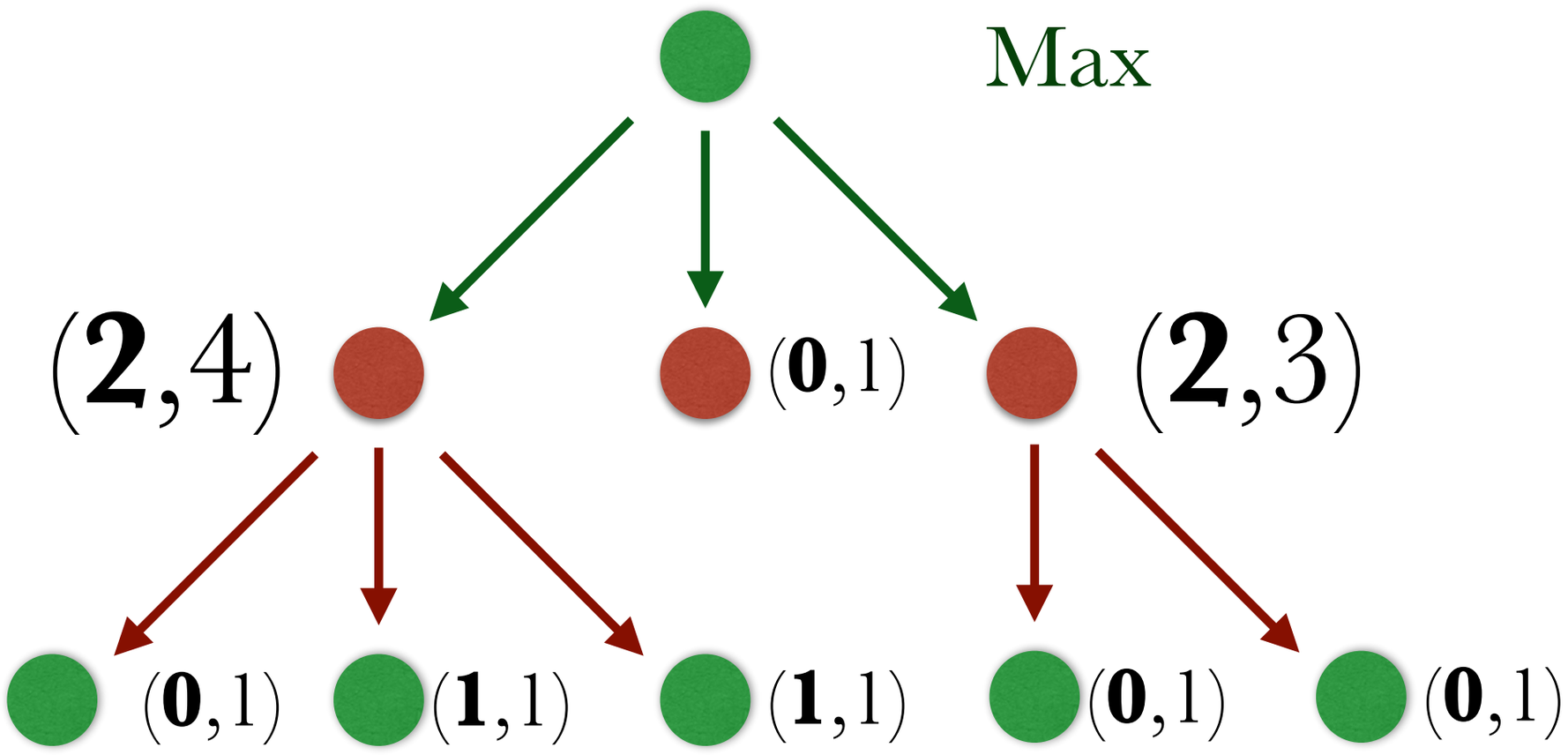
$$B(k, t, n_k(t)) = \bar{X}(k) + \sqrt{\frac{2 \log t}{n_k(t)}}$$

- Play the arm with $\max_k B(k, t, n_k(t))$

What happens in the first K rounds?

Back to MCTS

- How do we decide whether to keep going down into a subtree, or change focus to another one?



Back to MCTS

- How do we decide whether to keep going down into a subtree (exploit), or change our focus to another one? (explore)

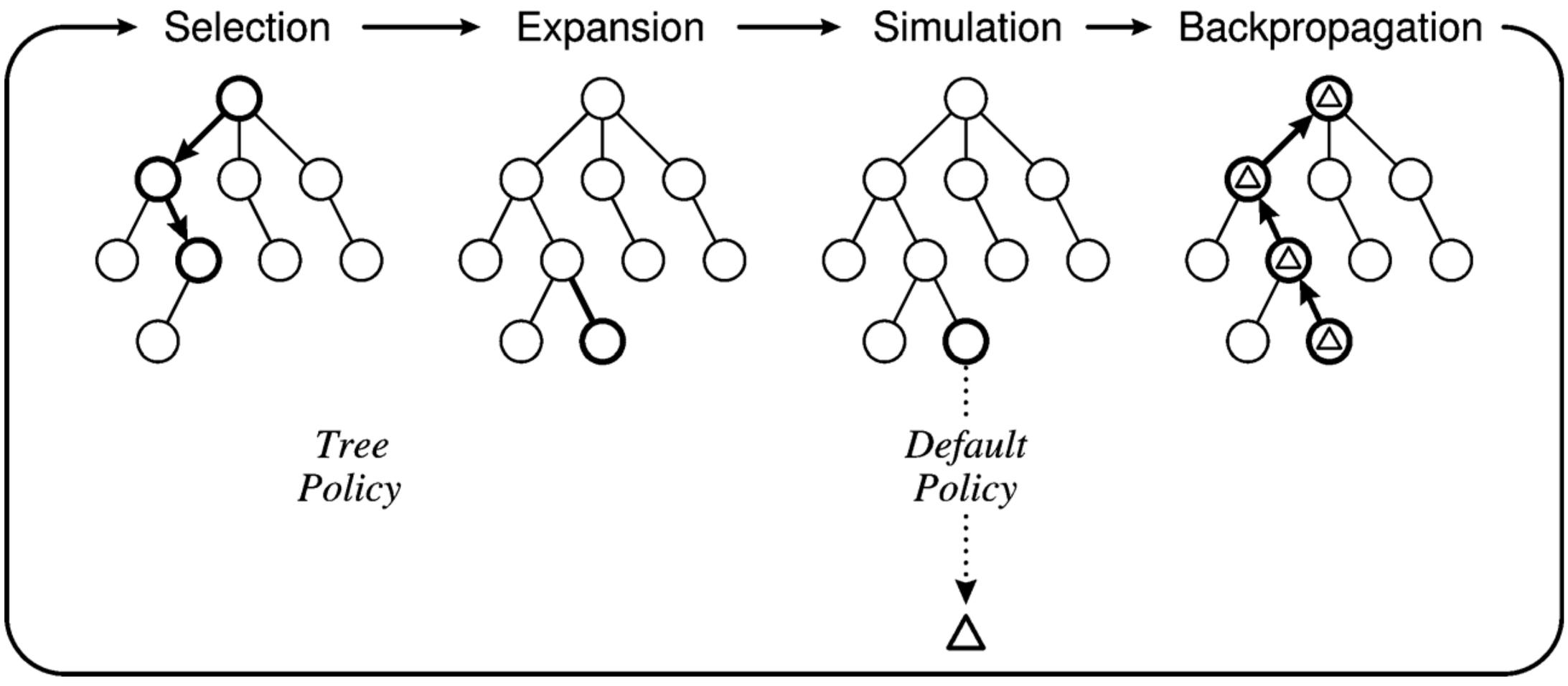
- Use the UCB formula (with a new parameter c)

function BESTCHILD(s, c)

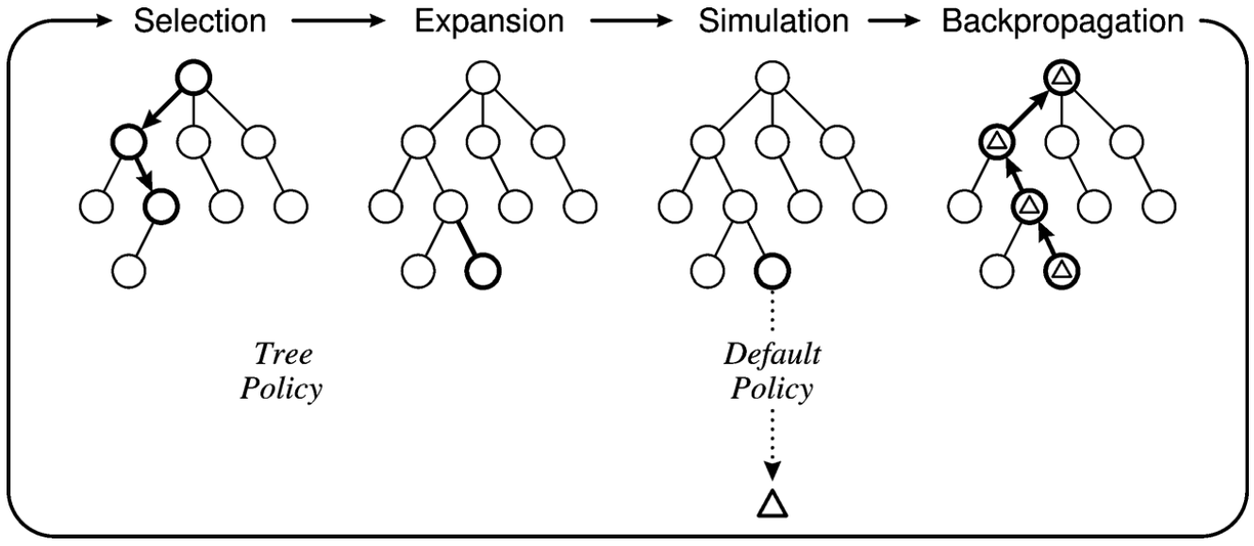
return $\text{argmax}_{s' \in \text{children}(s)} \left(\frac{Q(s')}{N(s')} + c\sqrt{\frac{2 \ln N(s)}{N(s')}} \right)$

end function

MCTS with UCT

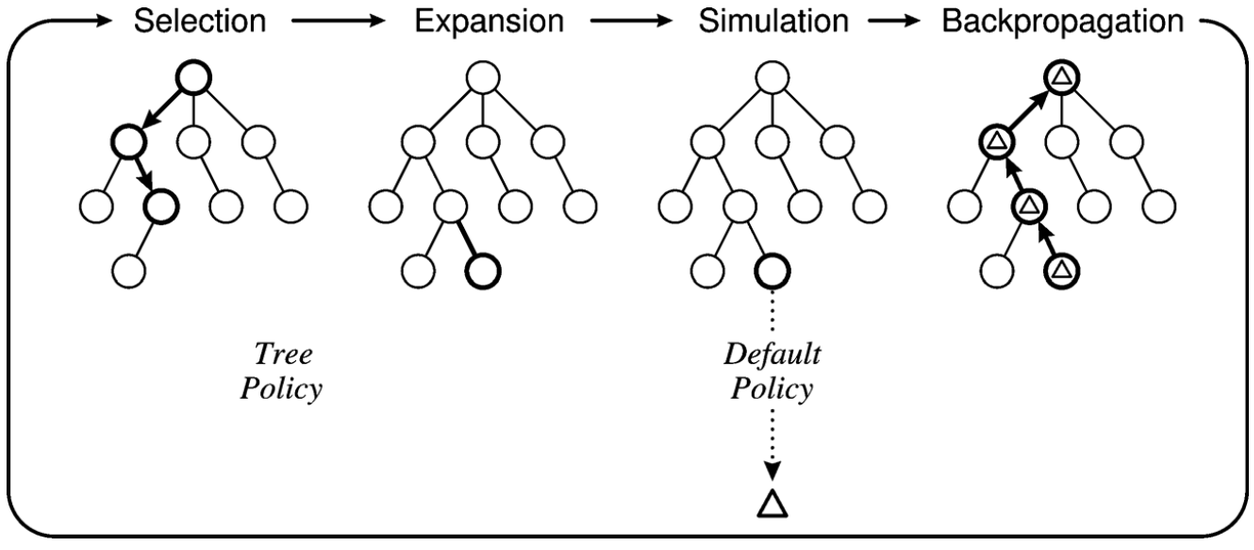


MCTS with UCT



function MCTS(s_{root})
  while within computational budget **do**
    $s \leftarrow$TreePolicy(s_{root})
    winner$\leftarrow$DefaultPolicy(s)
    Backup(s, winner)
  end while
  return Action(argmax$_{s' \in \text{children}(s_{root})} \frac{Q(s')}{N(s')}$)
end function

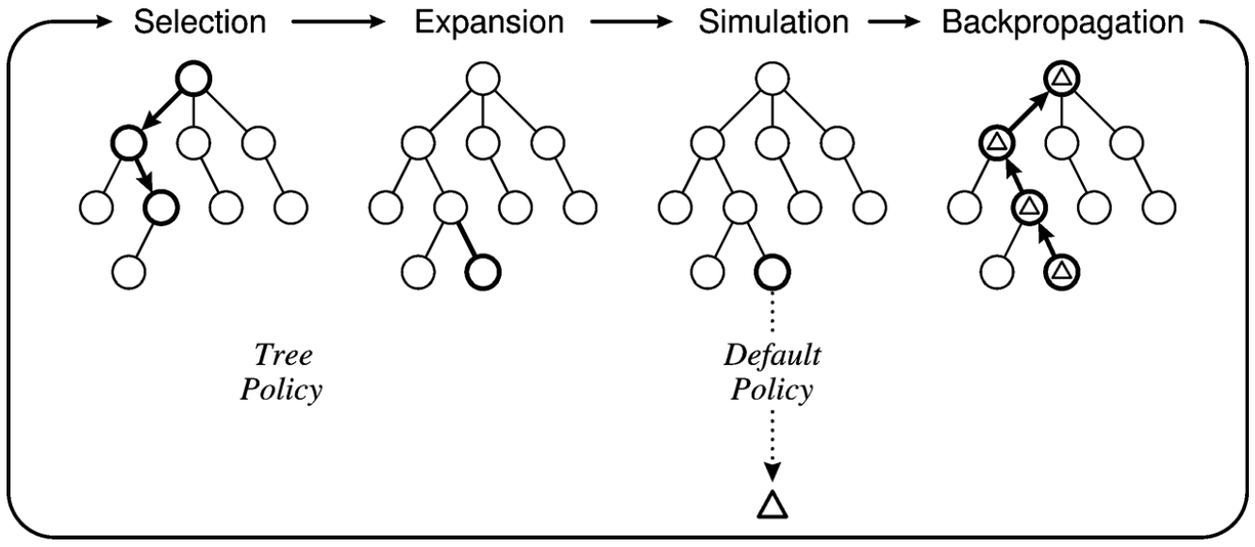
MCTS with UCT



function TreePolicy(s)
  while s is not terminal **do**
    if s is not fully expanded **then**
      return Expand(s)
    else
      $s \leftarrow$ BestChild(s, c)
    end if
  end while
  return s
end function

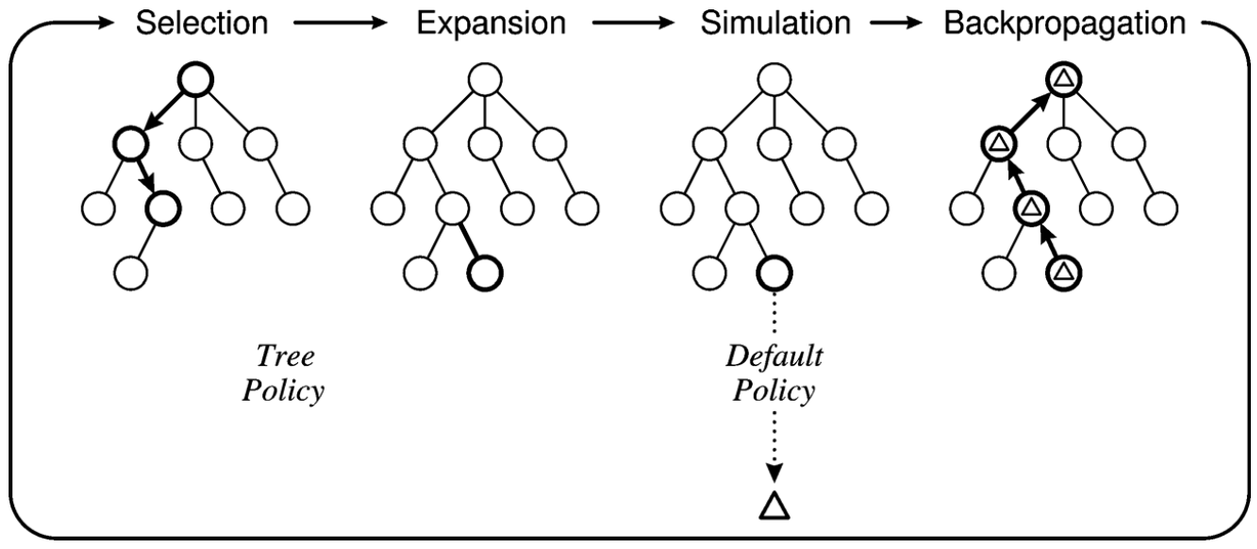
function BestChild(s, c)
  return $\text{argmax}_{s' \in \text{children}(s)} \left(\frac{Q(s')}{N(s')} + c\sqrt{\frac{2 \ln N(s)}{N(s')}} \right)$
end function

MCTS with UCT



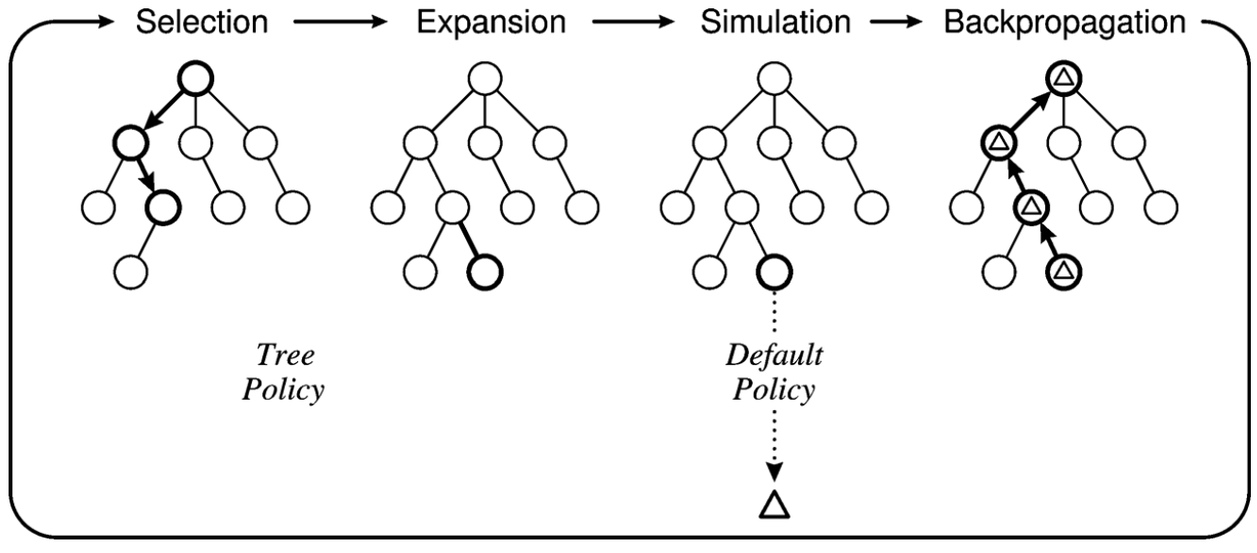
function EXPAND(s)
    child $\leftarrow$ previously unexpanded child of s
    Update the tree with $(s, child)$
    return child
end function

MCTS with UCT



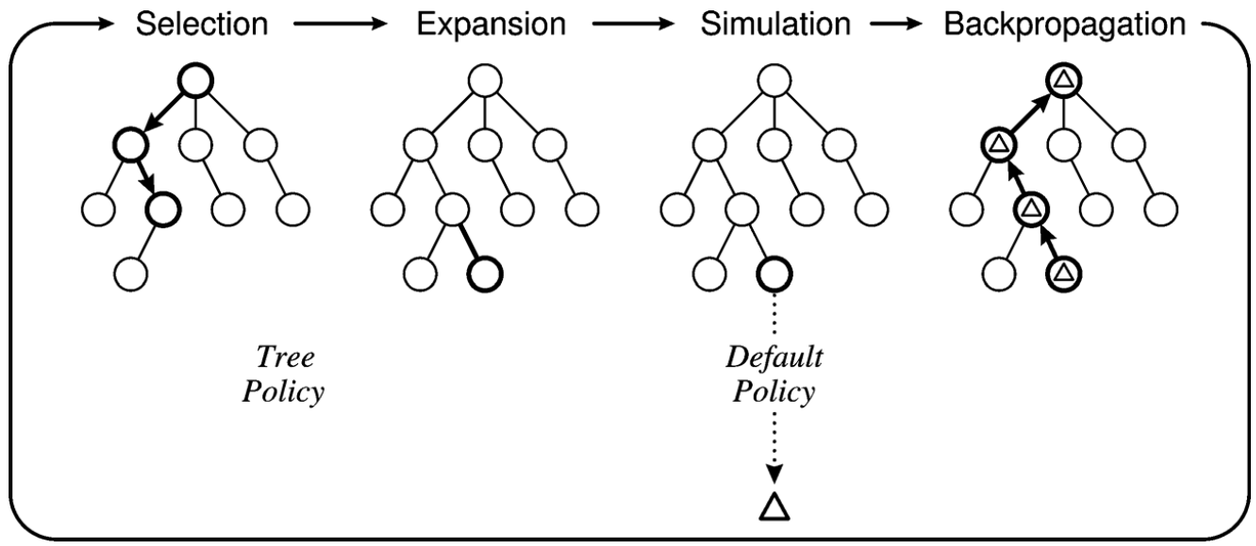
function DefaultPolicy(s)
    while s is not terminal **do**
        $s \leftarrow$ random child of s
    end while
    return winner
end function

MCTS with UCT



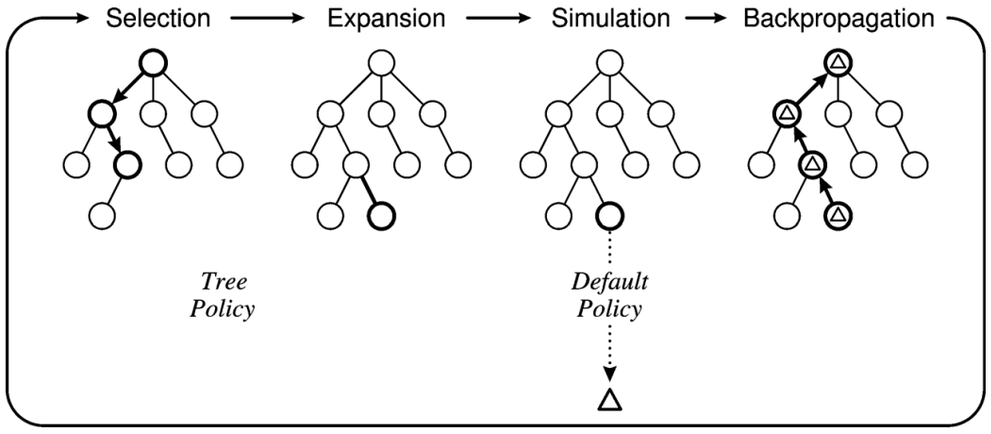
function BACKUP(s, winner)
  while s is not Null **do**
    $N(s) \leftarrow N(s) + 1$
    $Q(s) \leftarrow Q(s) + \Delta(s, winner)$
    $s \leftarrow$parent(s)
  end while
end function

MCTS with UCT

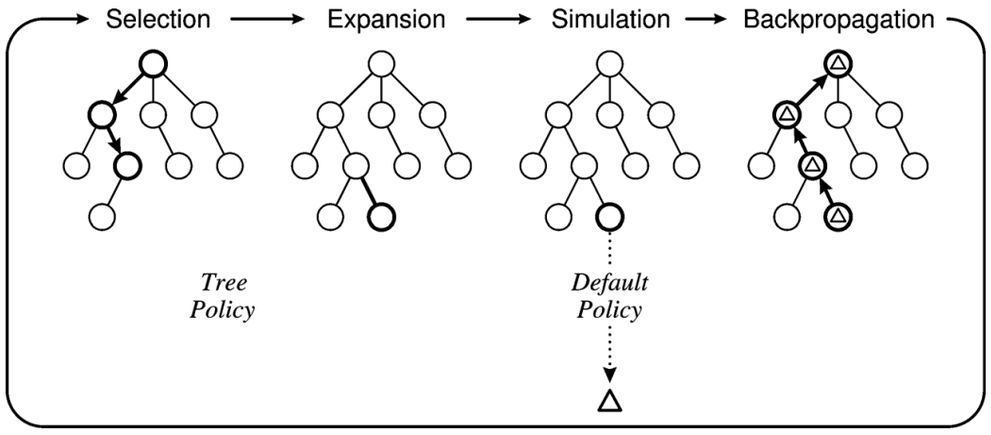


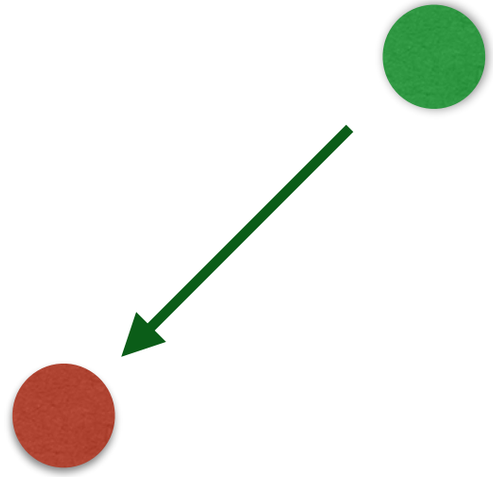
function BACKUP(s, winner)
  while s is not Null **do**
    $N(s) \leftarrow N(s) + 1$
    $Q(s) \leftarrow Q(s) + \Delta(s, winner)$
    $s \leftarrow$parent(s)
  end while
end function

Important:
$\Delta(s, \text{winner}) = 1$ if s is **controlled** by the winner, and 0 otherwise

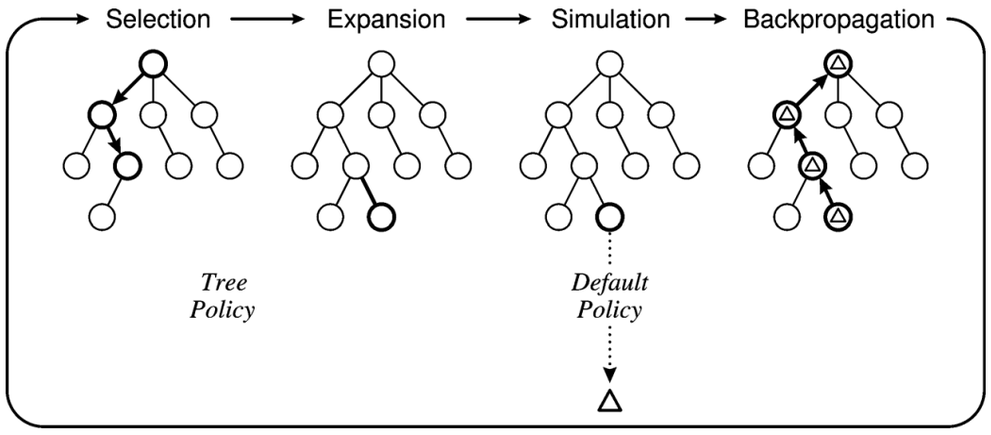

Selection
Expansion
Simulation
Backpropagation
Tree
Policy
Default
Policy

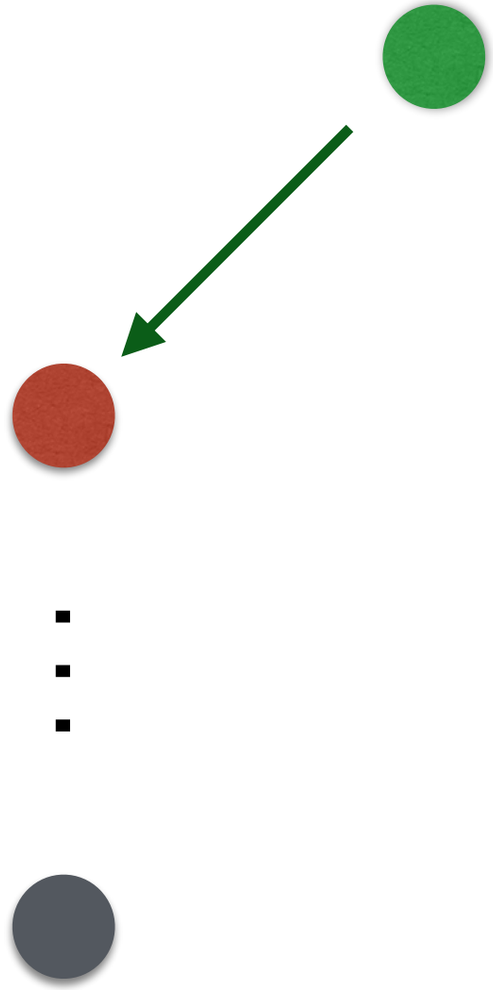
Max


Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy



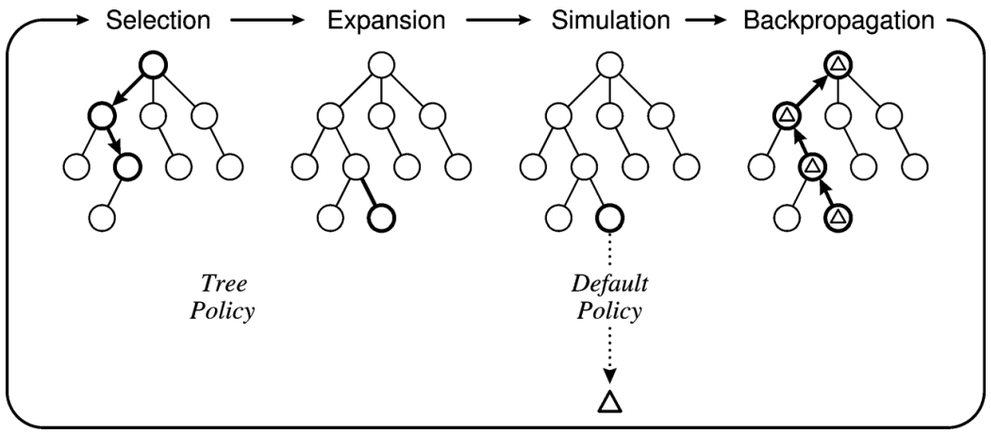
Max

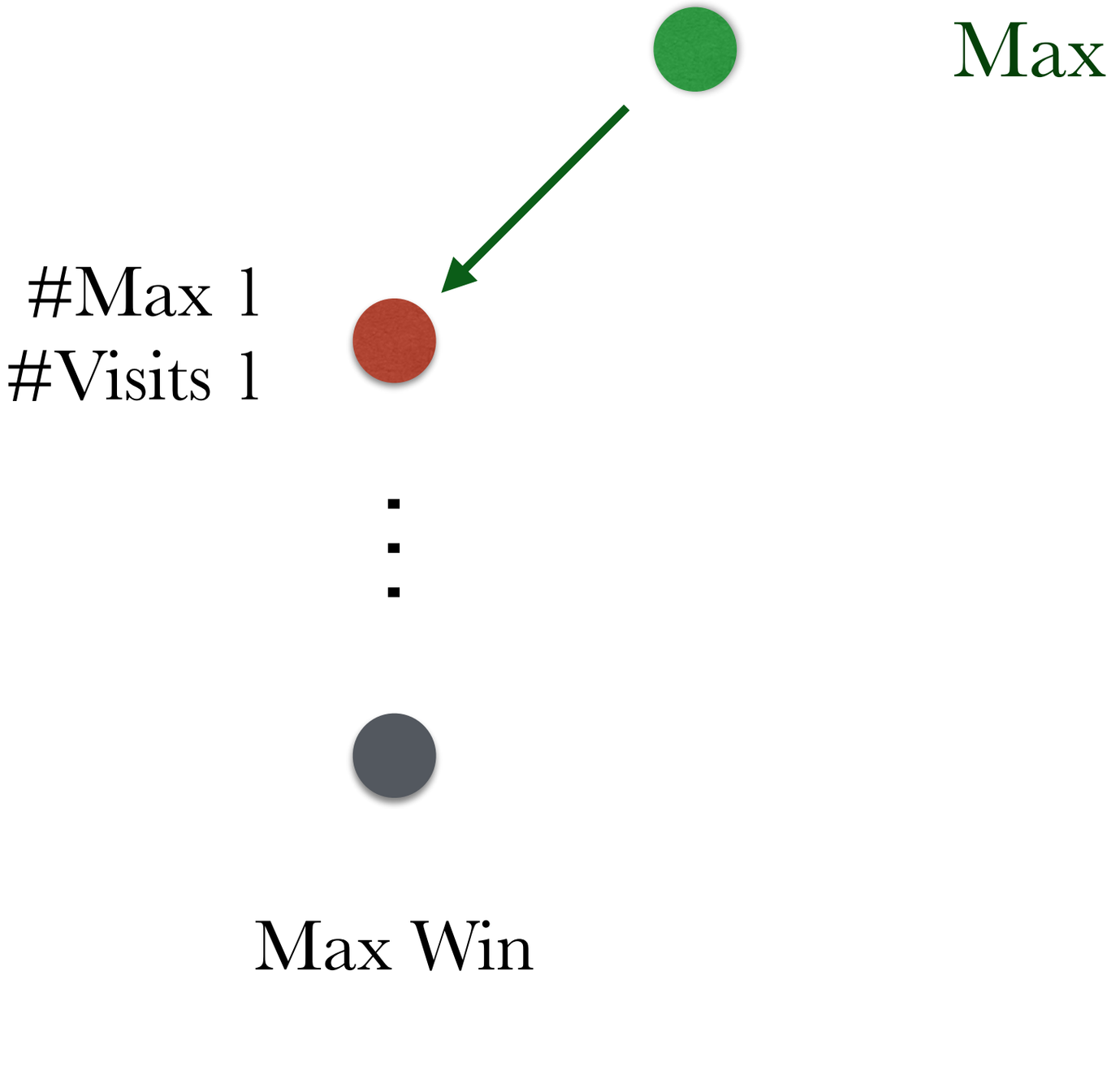

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

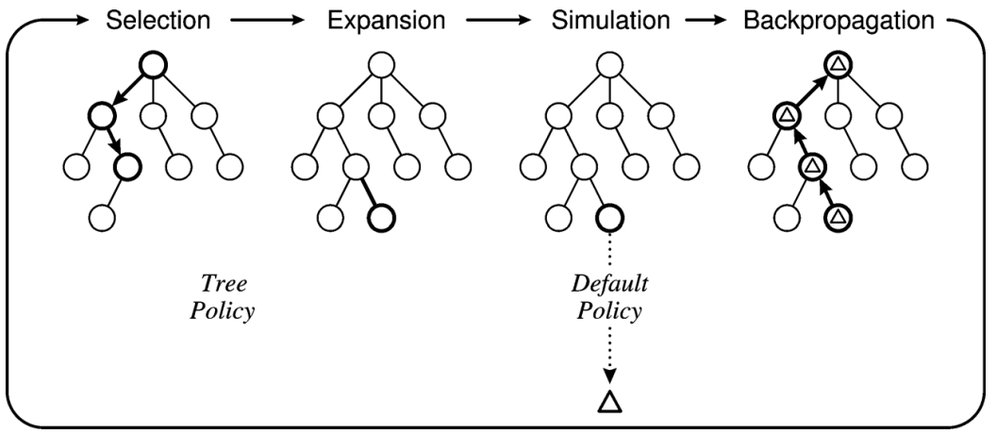


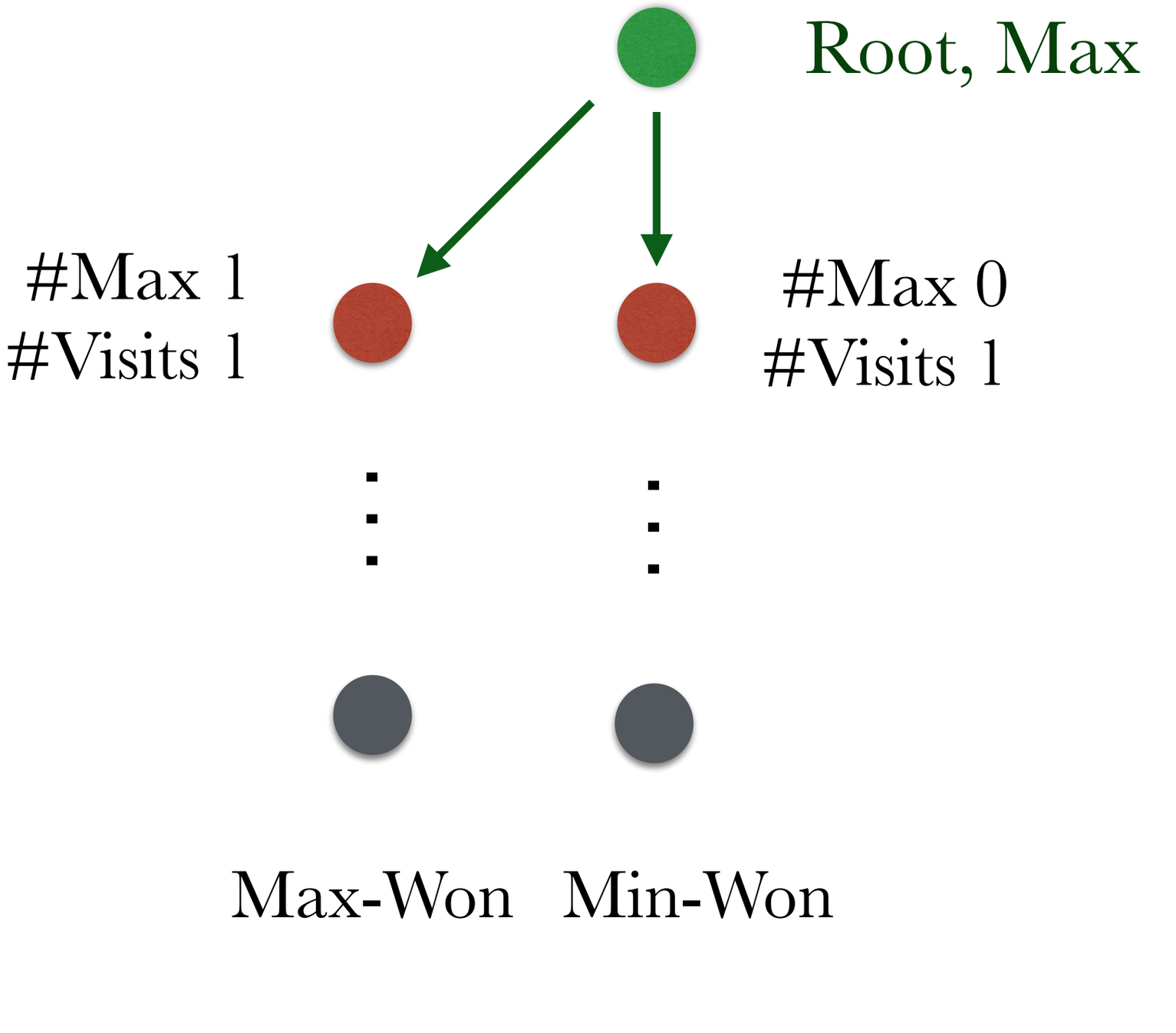
Max

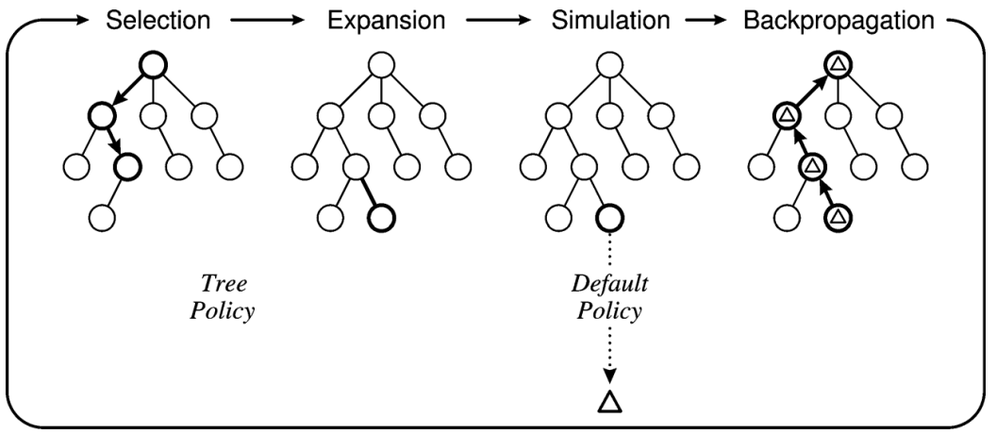
Win? Lose?

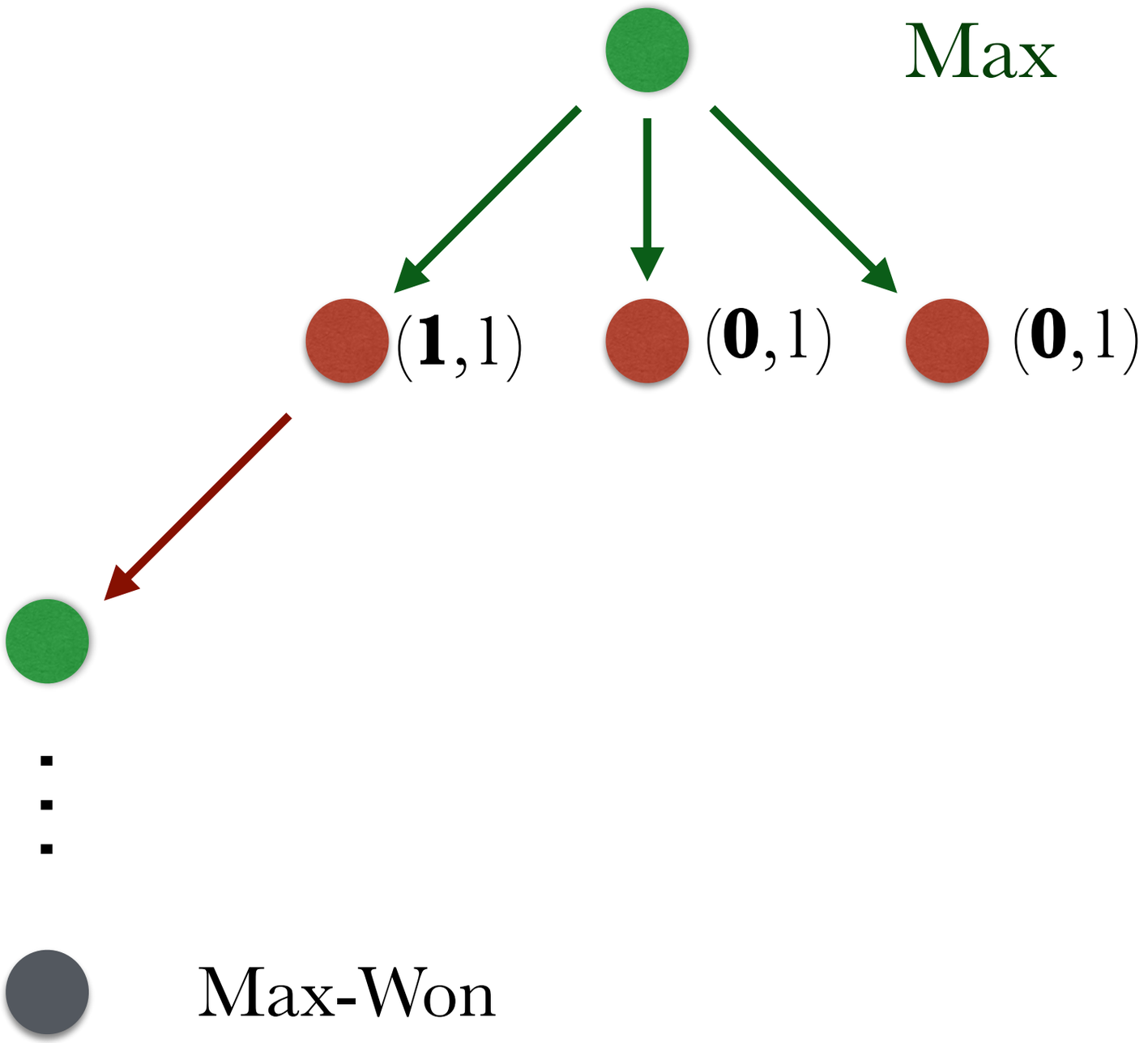

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

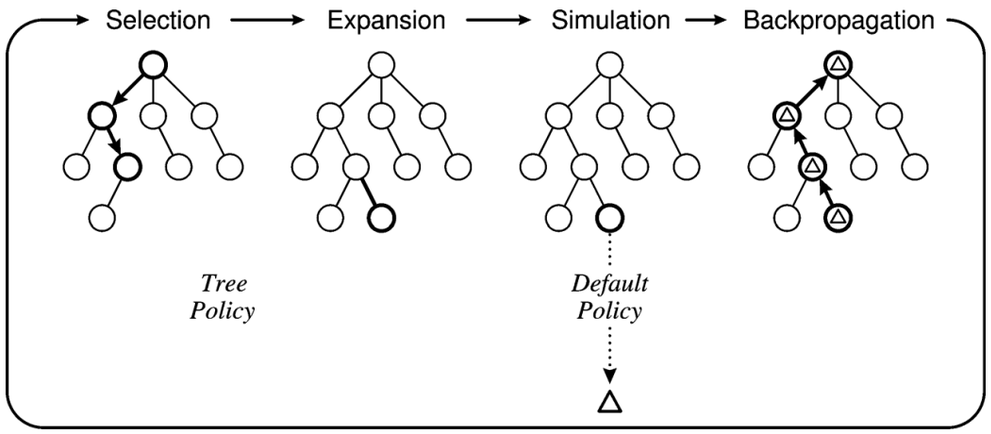

Max
#Max 1
#Visits 1
Max Win

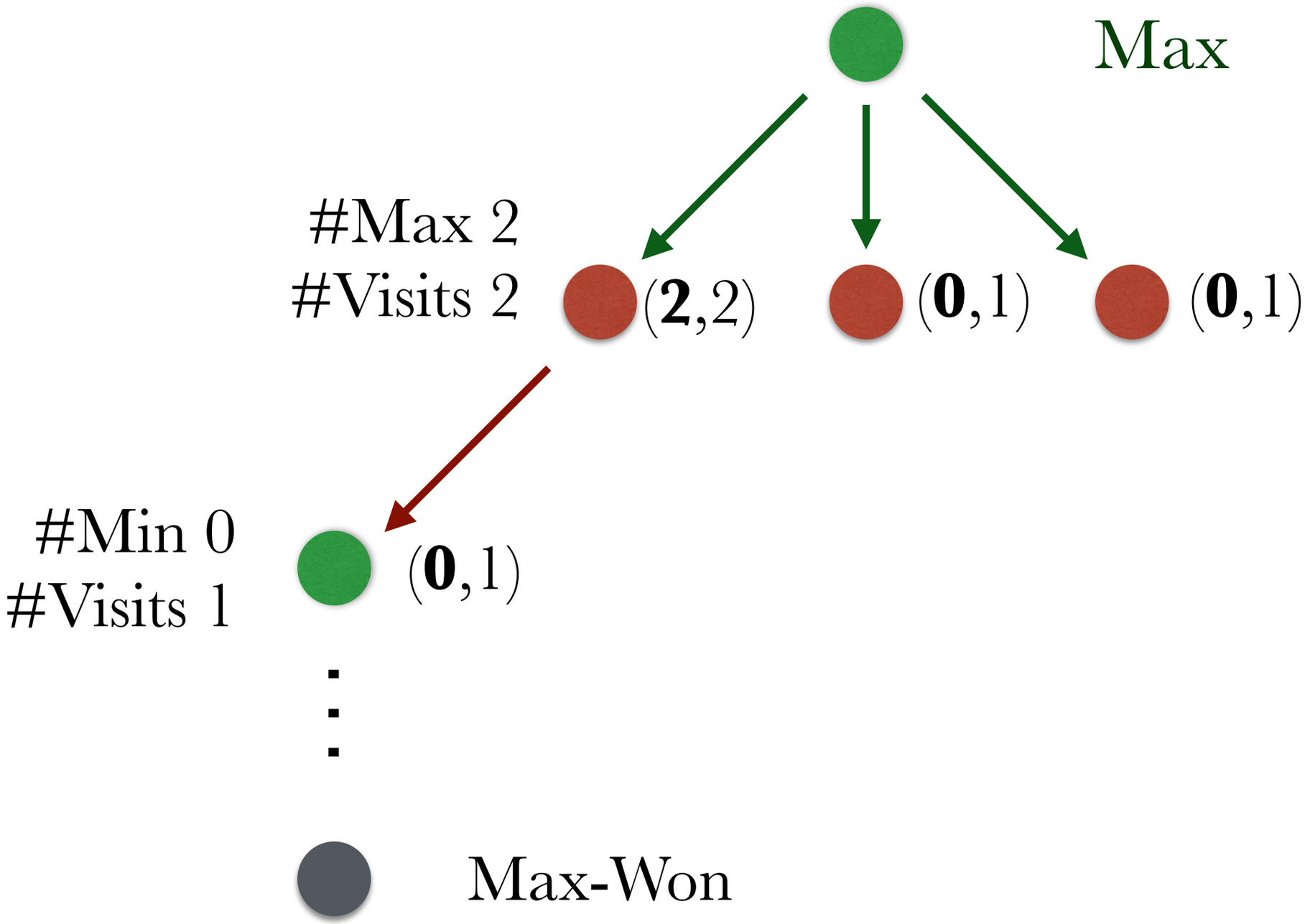

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

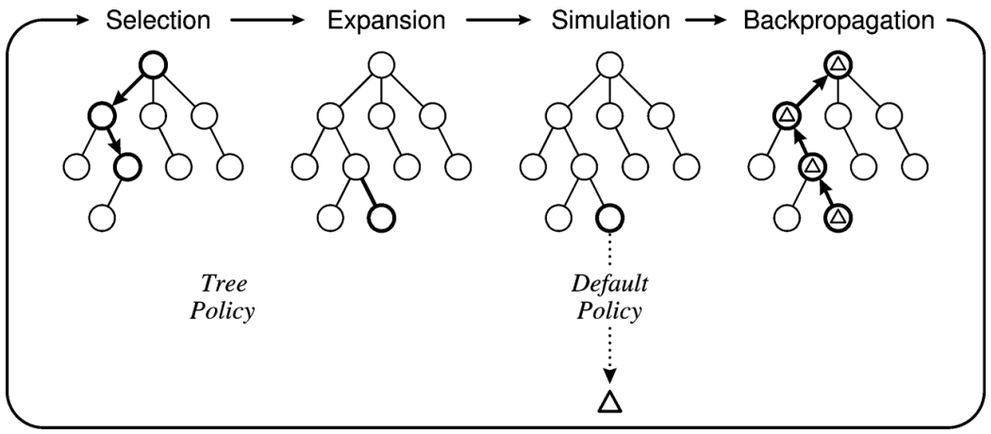

Root, Max
#Max 1
#Visits 1
#Max 0
#Visits 1
Max-Won
Min-Won

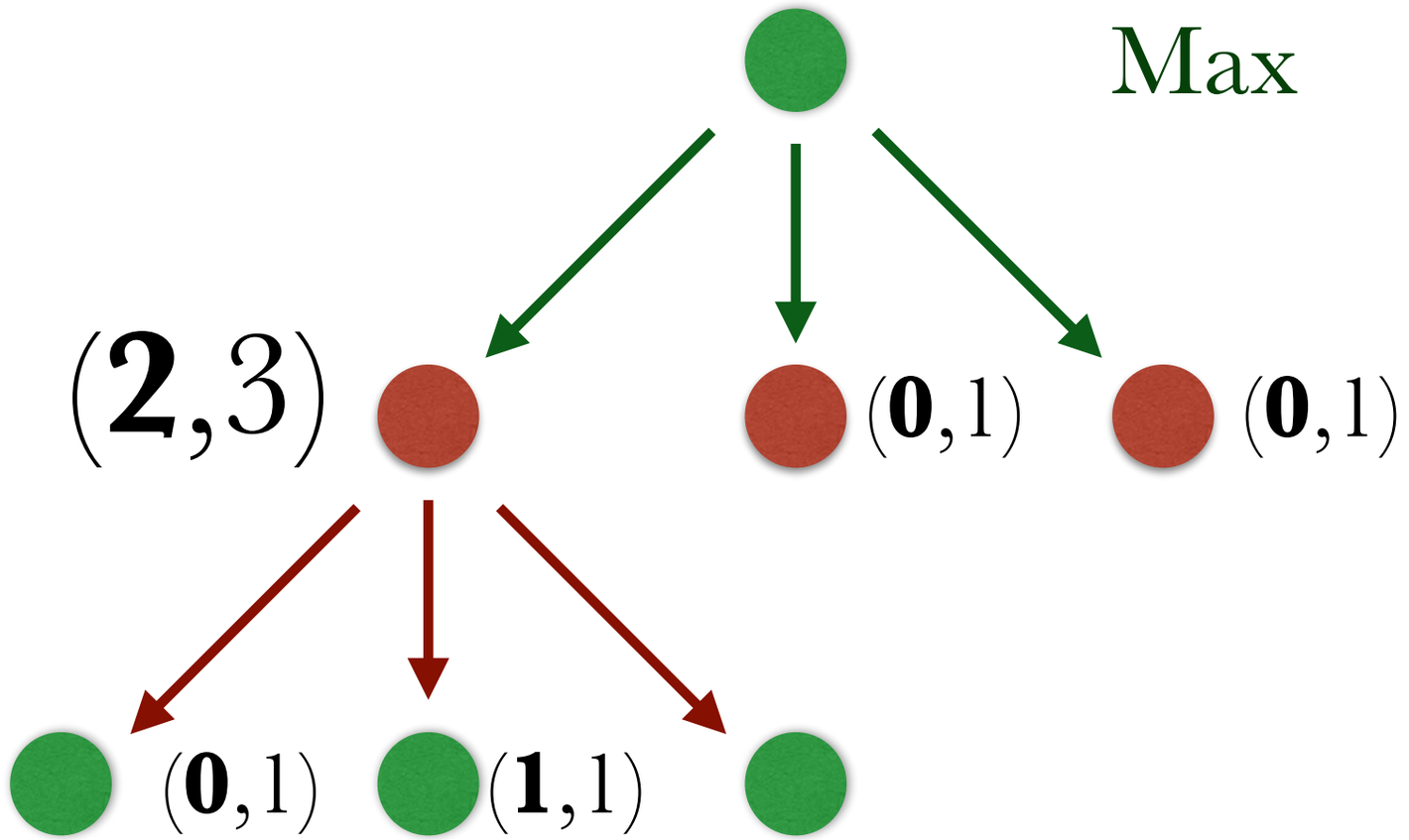

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

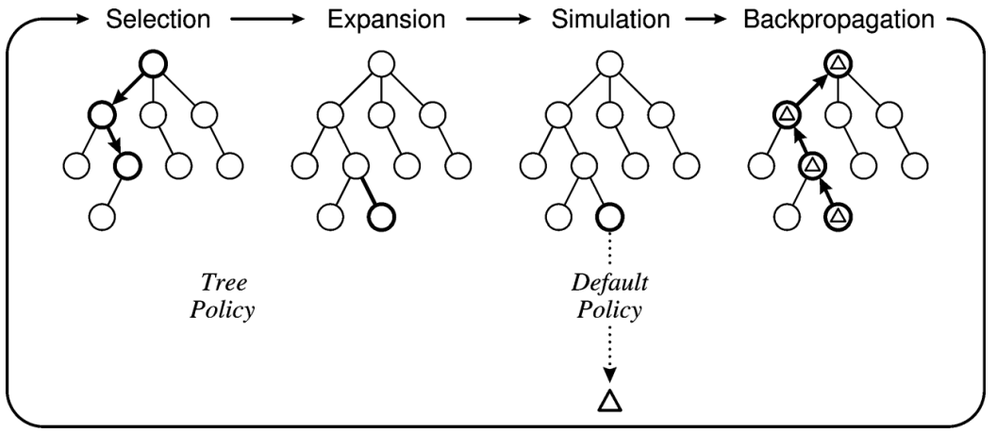

Max
(1,1)
(0,1)
(0,1)
Max-Won

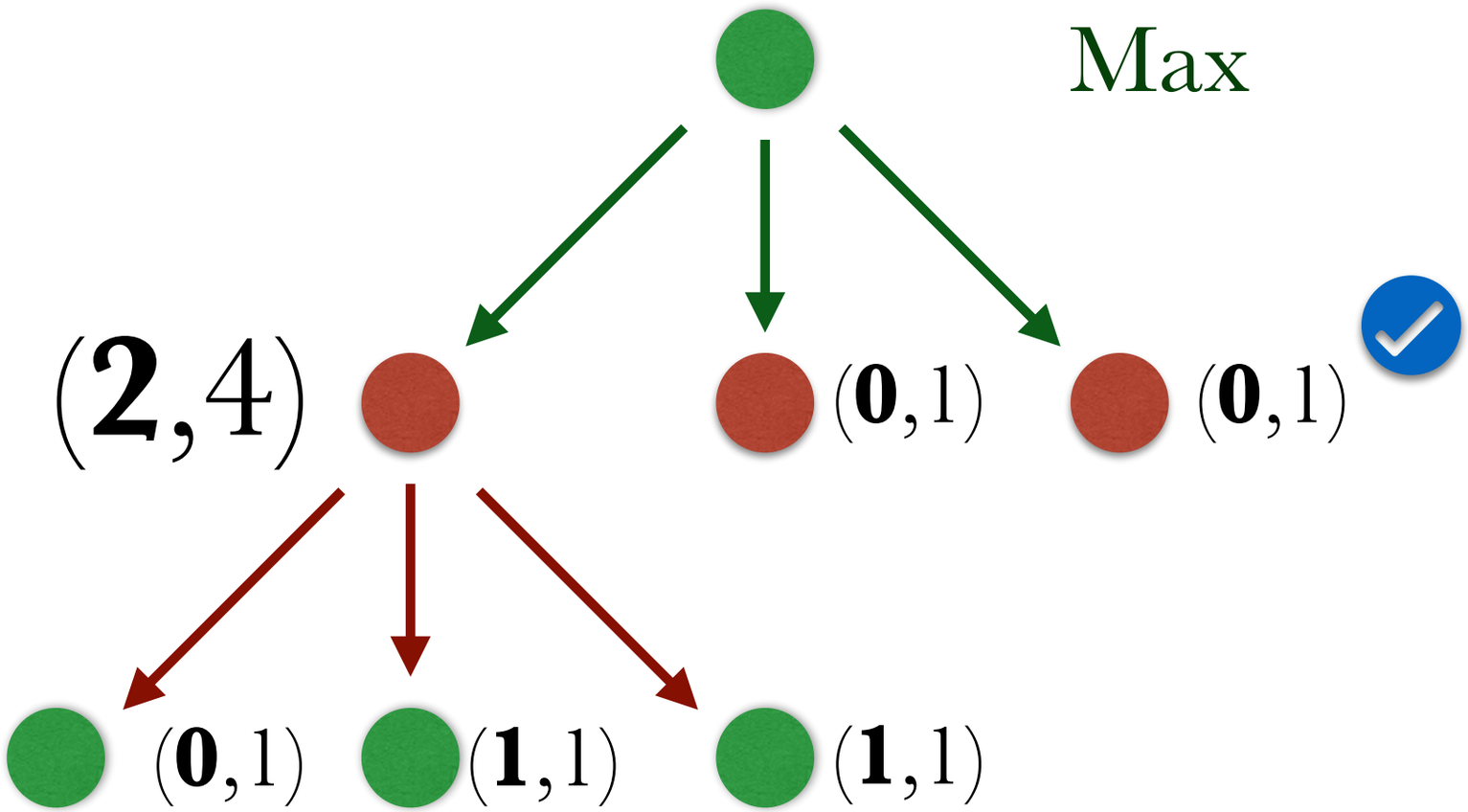

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy


Max
#Max 2
#Visits 2
(2,2)
(0,1)
(0,1)
#Min 0
#Visits 1
(0,1)
Max-Won


Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

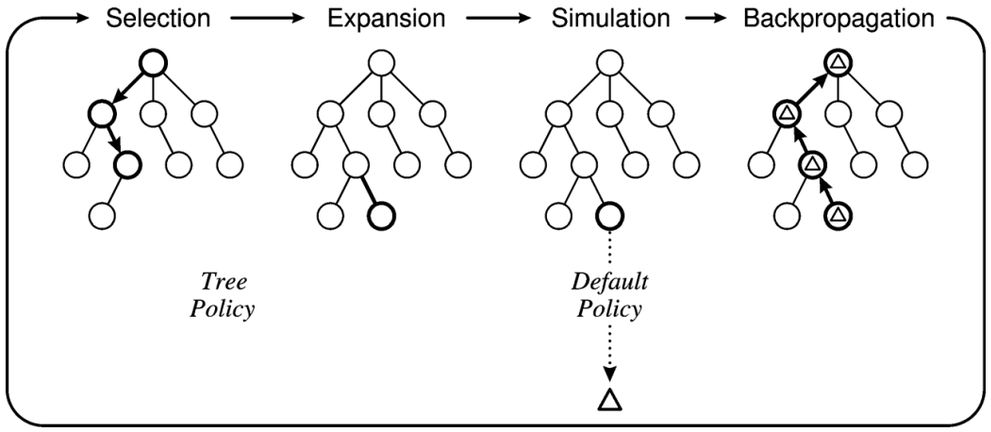

Max
(2,3)
(0,1)
(0,1)
(0,1)
(1,1)

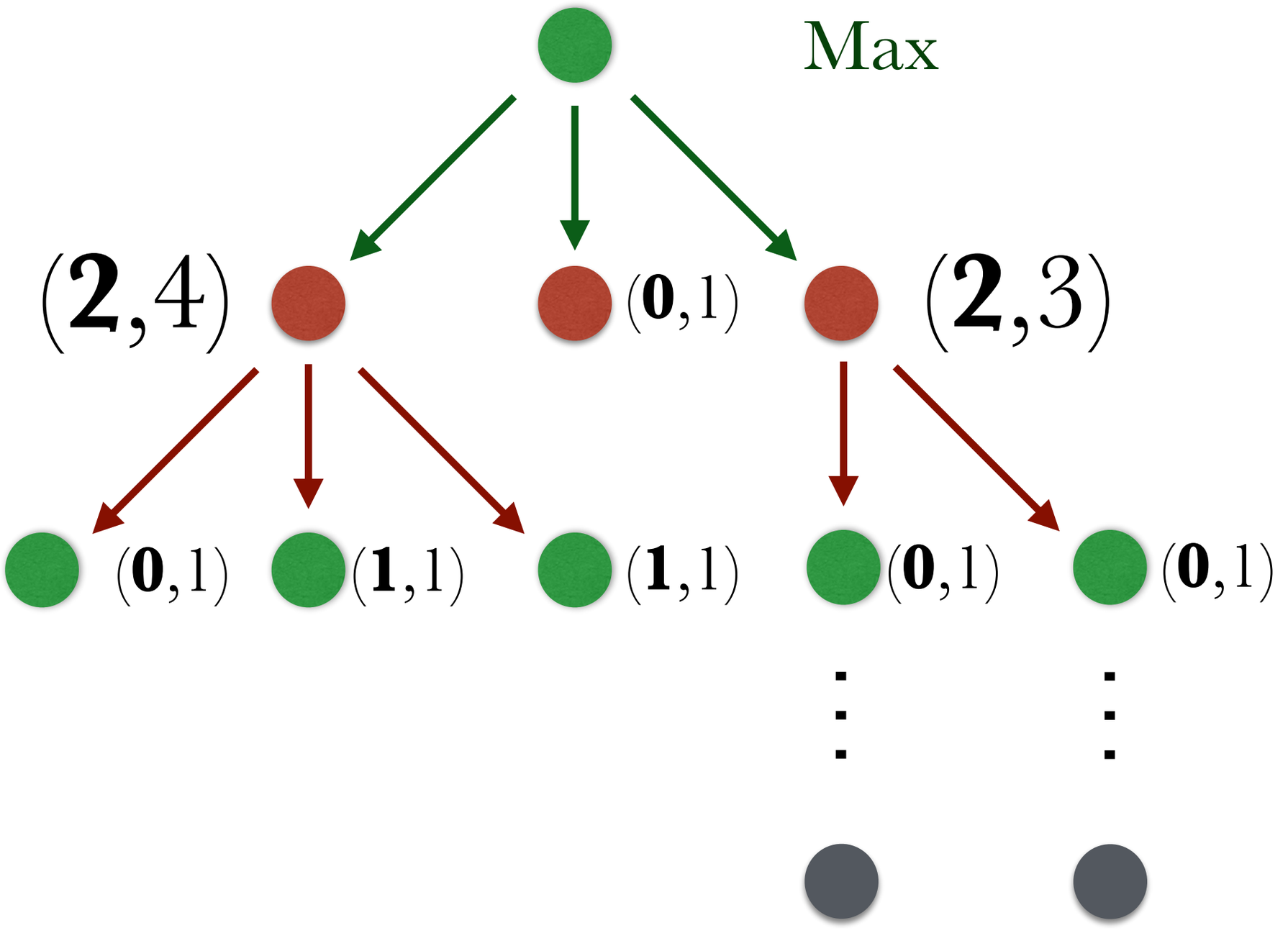




function BESTCHILD(s)

return $\text{argmax}_{s' \in \text{children}(s)} \left(\frac{Q(s')}{N(s')} + c\sqrt{\frac{\ln N(s)}{N(s')}} \right)$

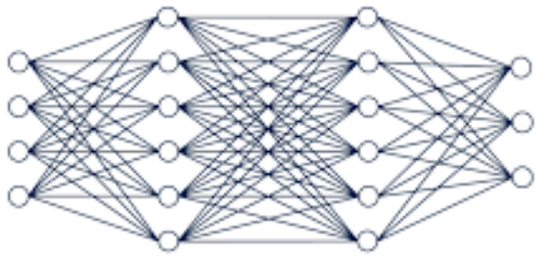

Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy


Max
(2,4)
(0,1)
(2,3)
(0,1)
(1,1)
(1,1)
(0,1)
(0,1)

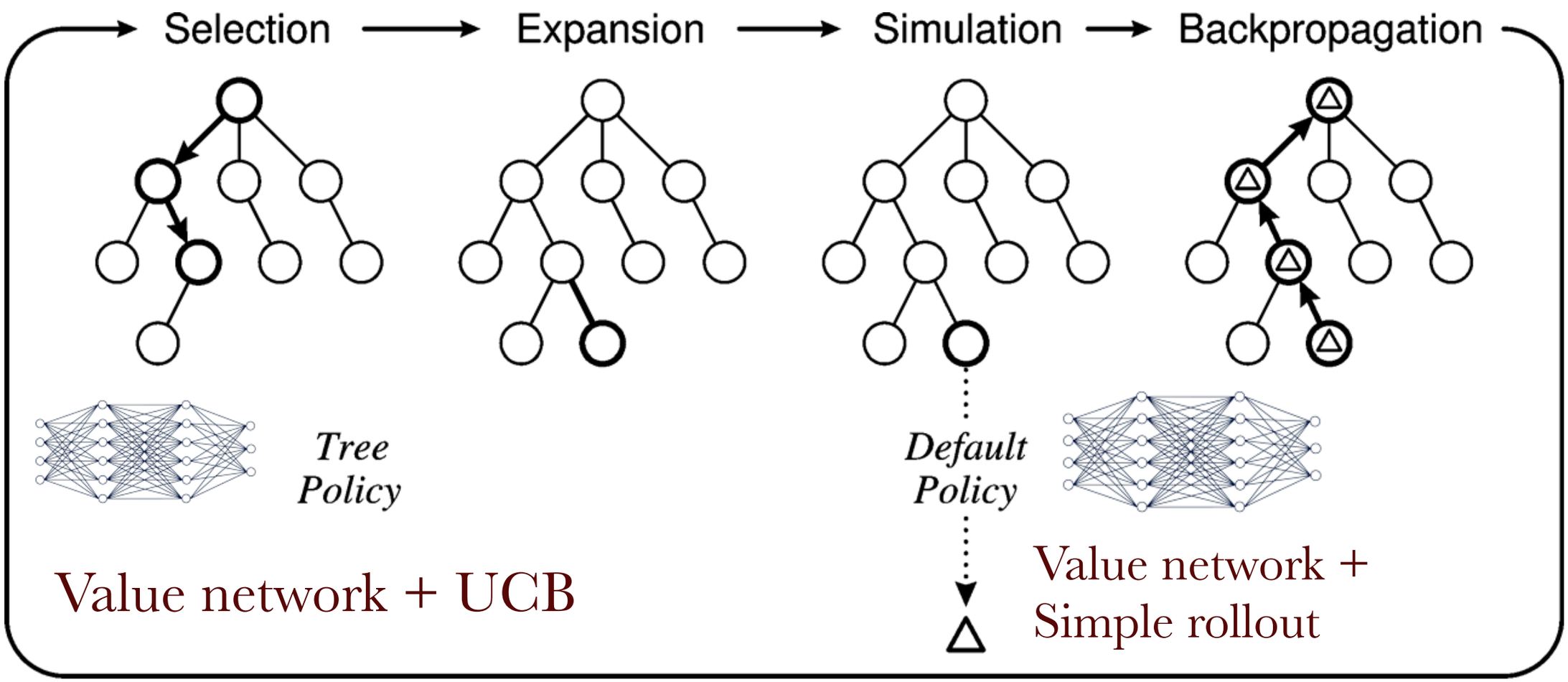
Article | Published: 27 January 2016

Mastering the game of Go with deep neural networks and tree search

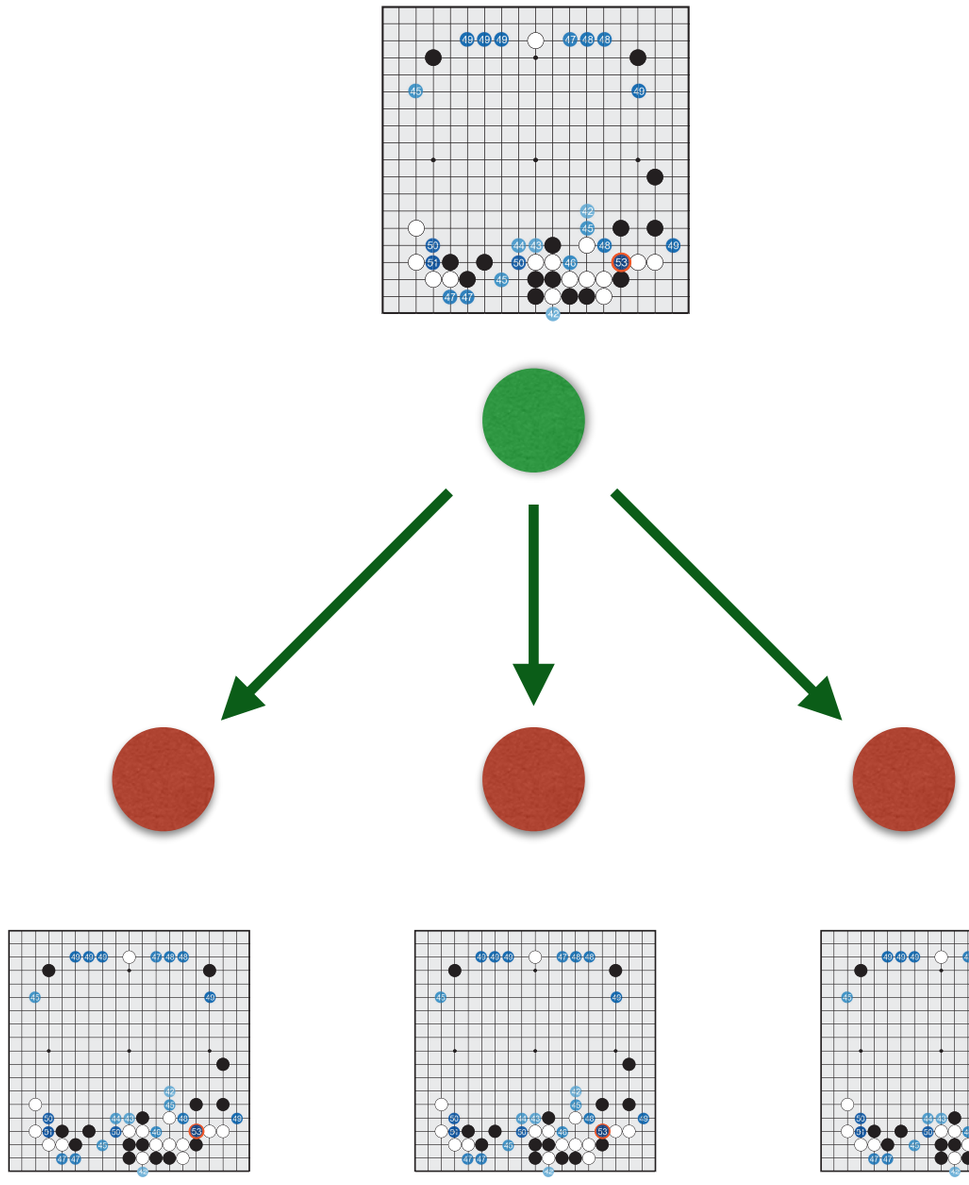
Deep RL + MCTS



Policy network



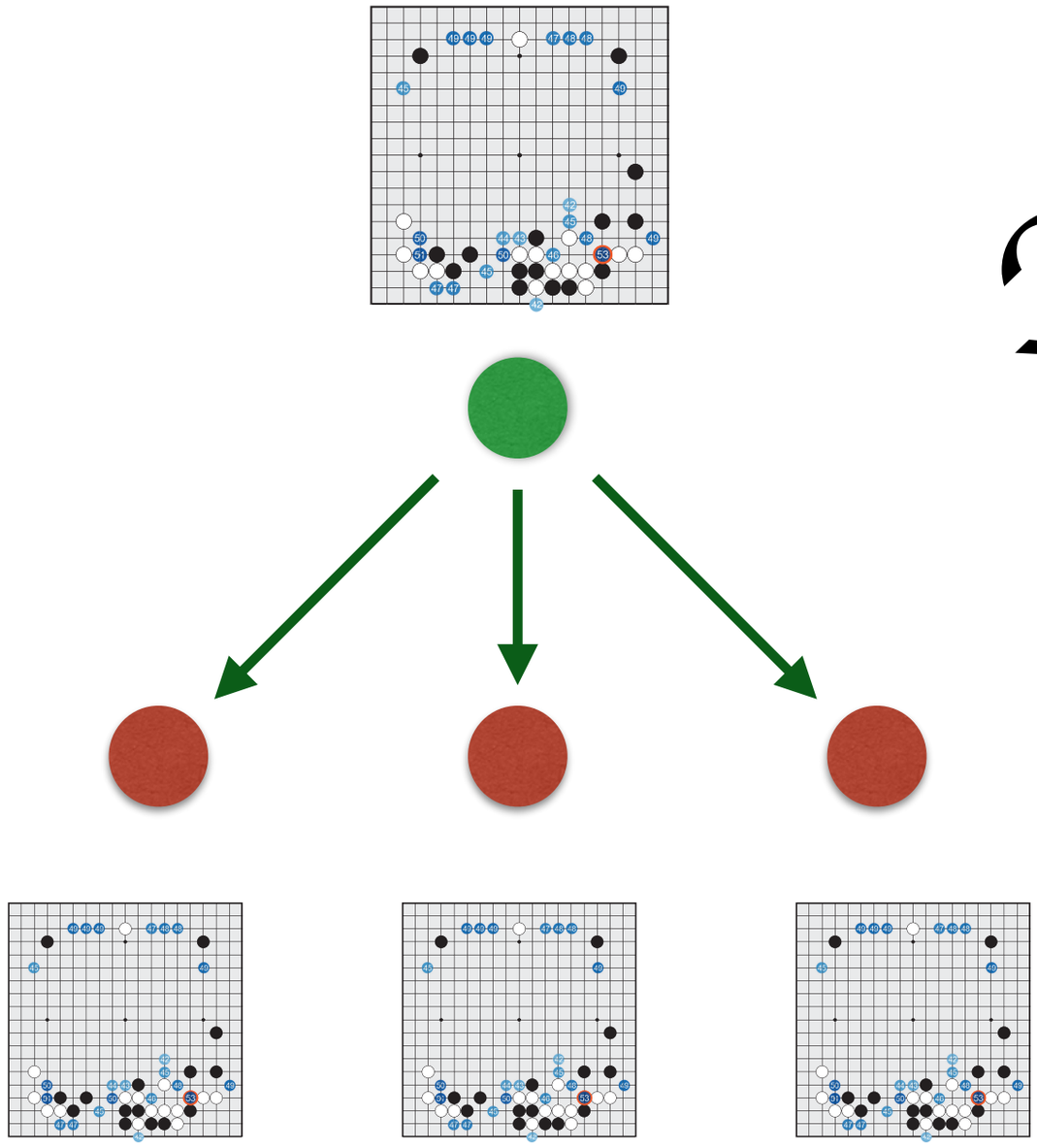
Learn from Human Moves



$$\mathcal{N}_h(\,\cdot\,) = P(a \mid s)$$

Huge database (30 million) of (s,a) examples

Learn from Human Moves

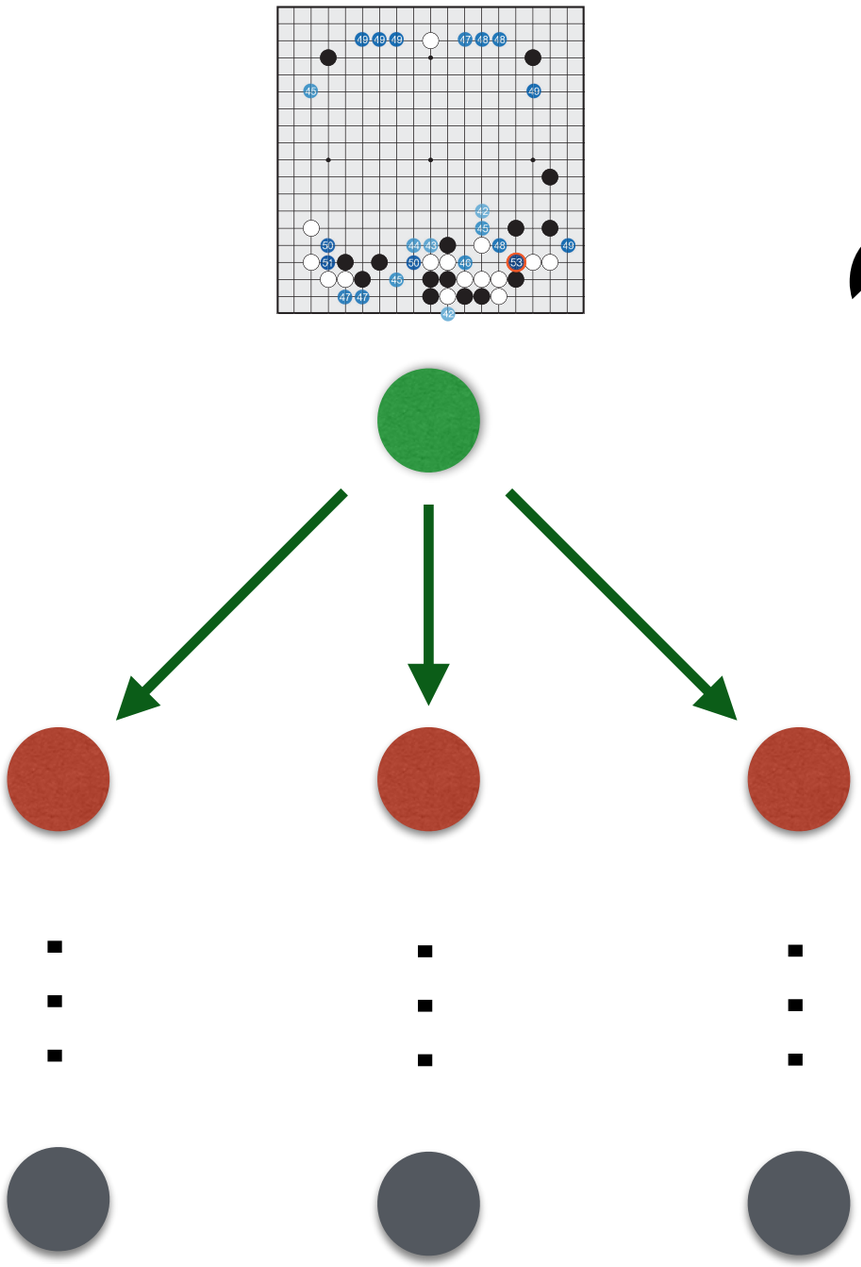


$$\mathcal{N}_h(\,\cdot\,) = P(a \mid s)$$

Huge database (30 million) of (s,a) examples

Enough? No, 57% accuracy.

Learn from Self-Play (Policy Gradient)

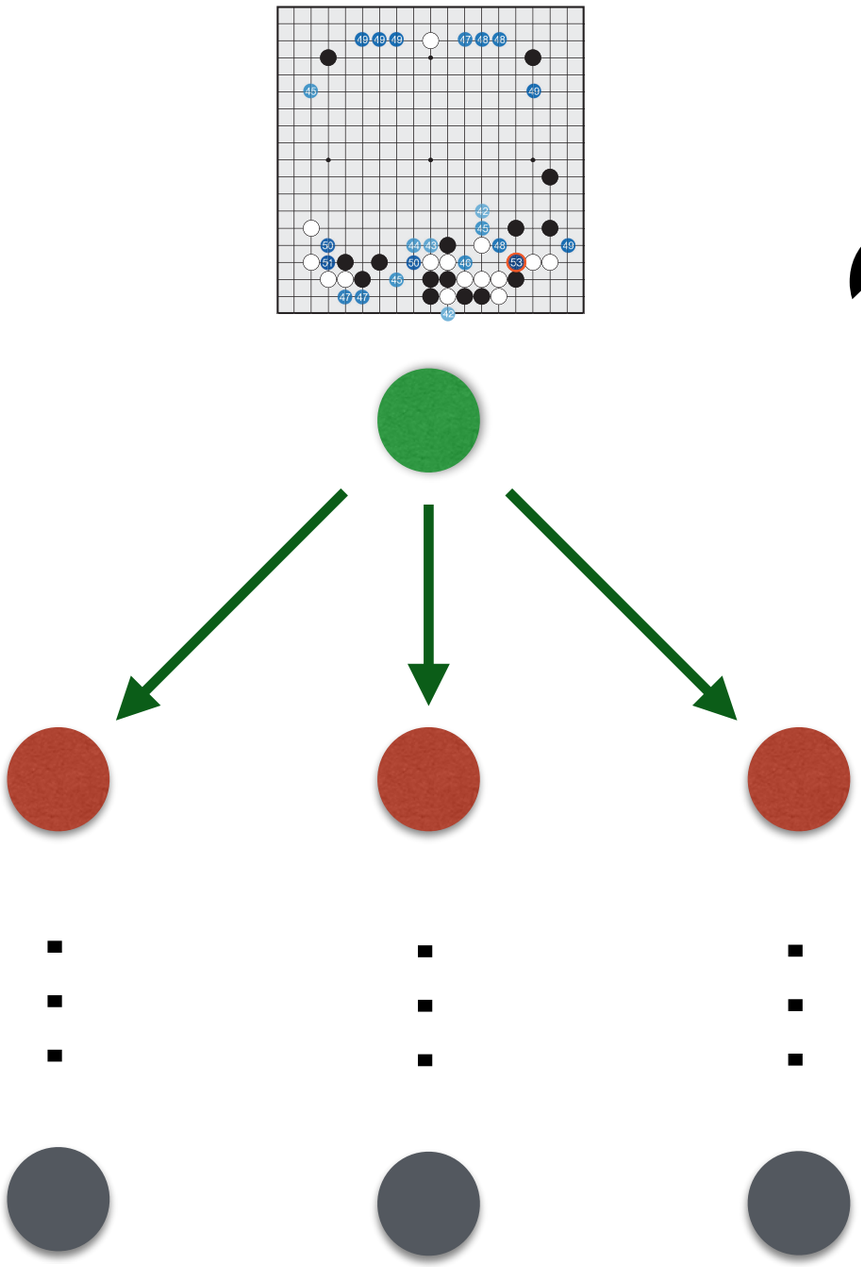


$$\mathcal{N}_{RL}(\quad) = P(a \mid s)$$

Don’t randomly rollout, but use the policy each time and improve.

Win 80% over previous.
85% against best pure MCTS.

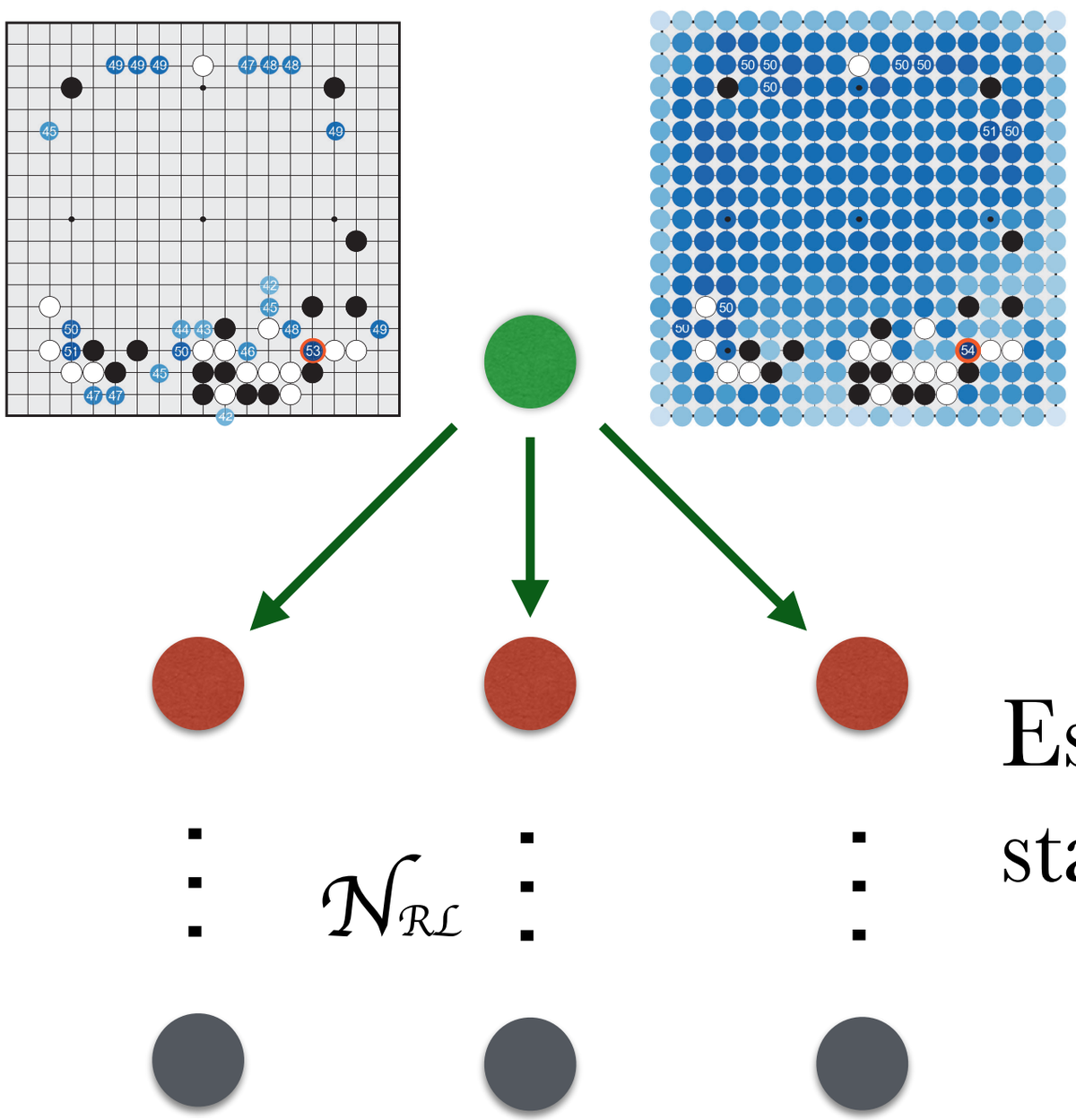
Learn from Self-Play (Policy Gradient)

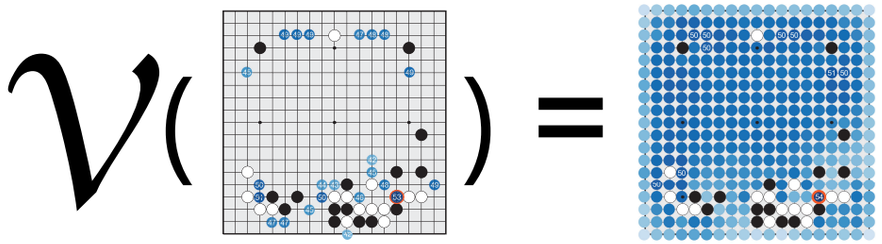


$$\mathcal{N}_{RL}(\quad) = P(a \mid s)$$

Such direct policies are too expensive, and can't capture full consequences in actual games.

Train Value Predictions

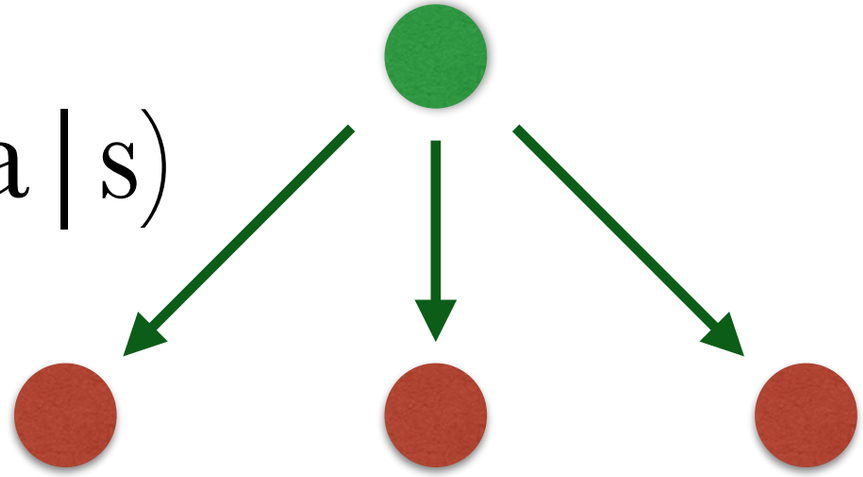




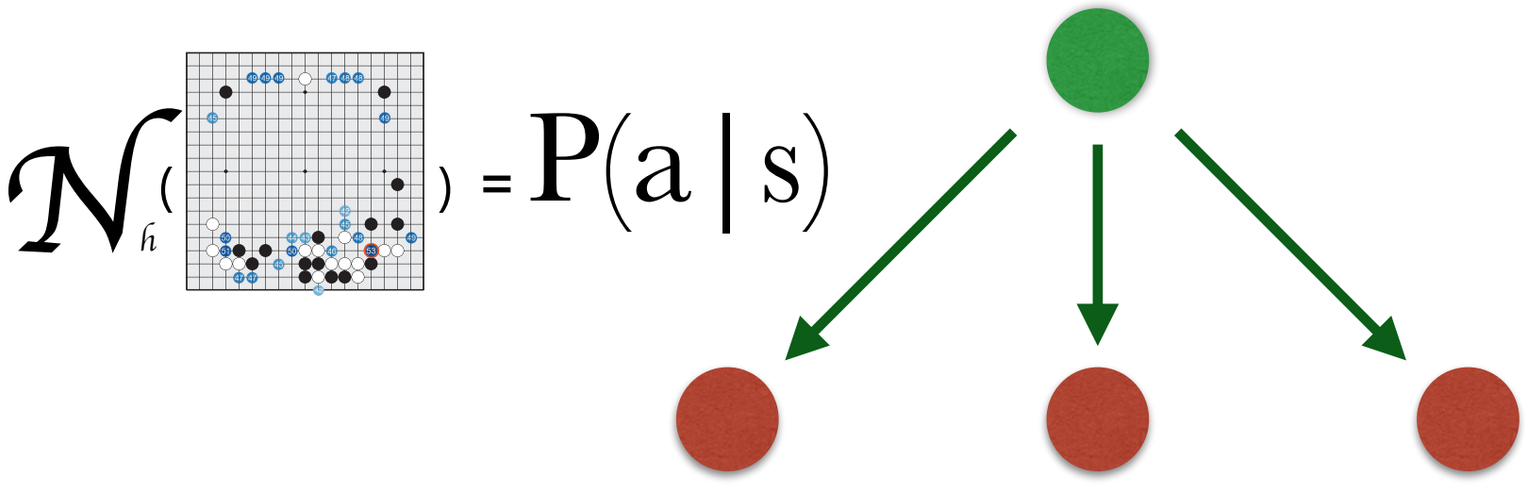
Estimate the value of each state under the best policy.

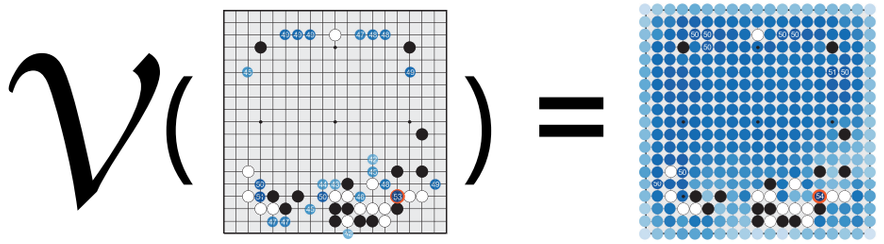
Put Things back in MCTS

$$\mathcal{N}_h\left(\quad\right) = P(a \mid s)$$



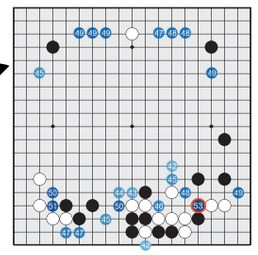
Put Things back in MCTS





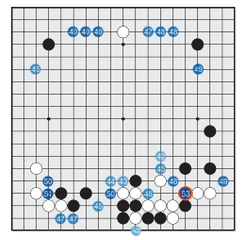
Put Things back in MCTS

$$\mathcal{N}_h(\ \ ) = P(a \mid s)$$

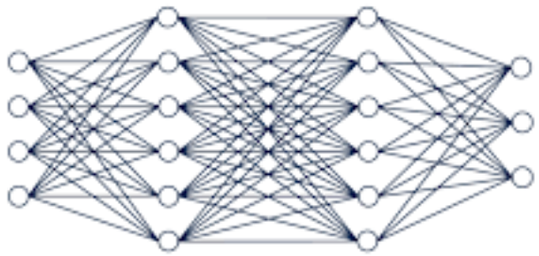


MCTS, cheap rollout

$$\mathcal{V}(\ \ ) + \mathrm{UCT}$$



Deep RL + MCTS



Policy network