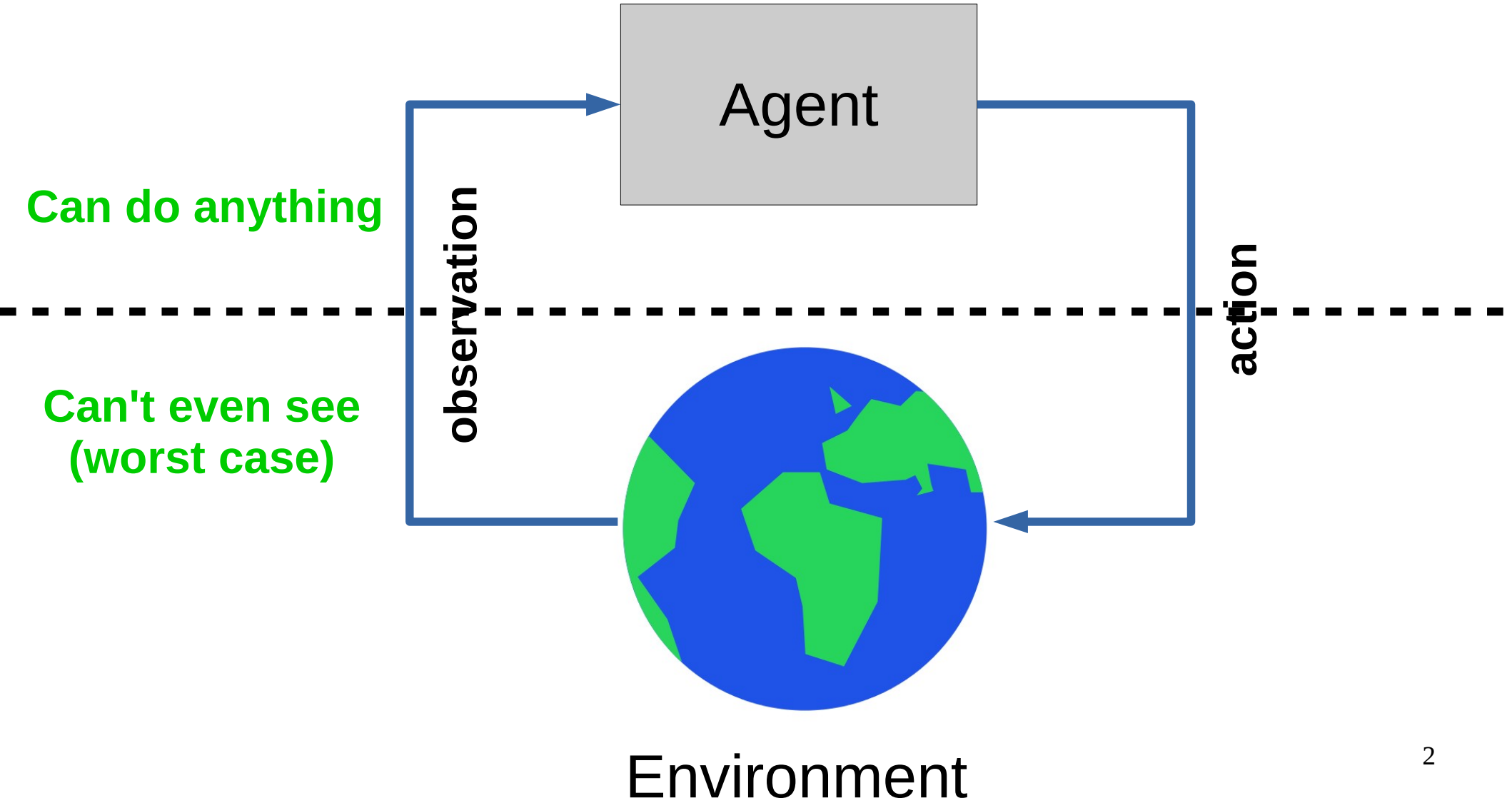
Reinforcement learning

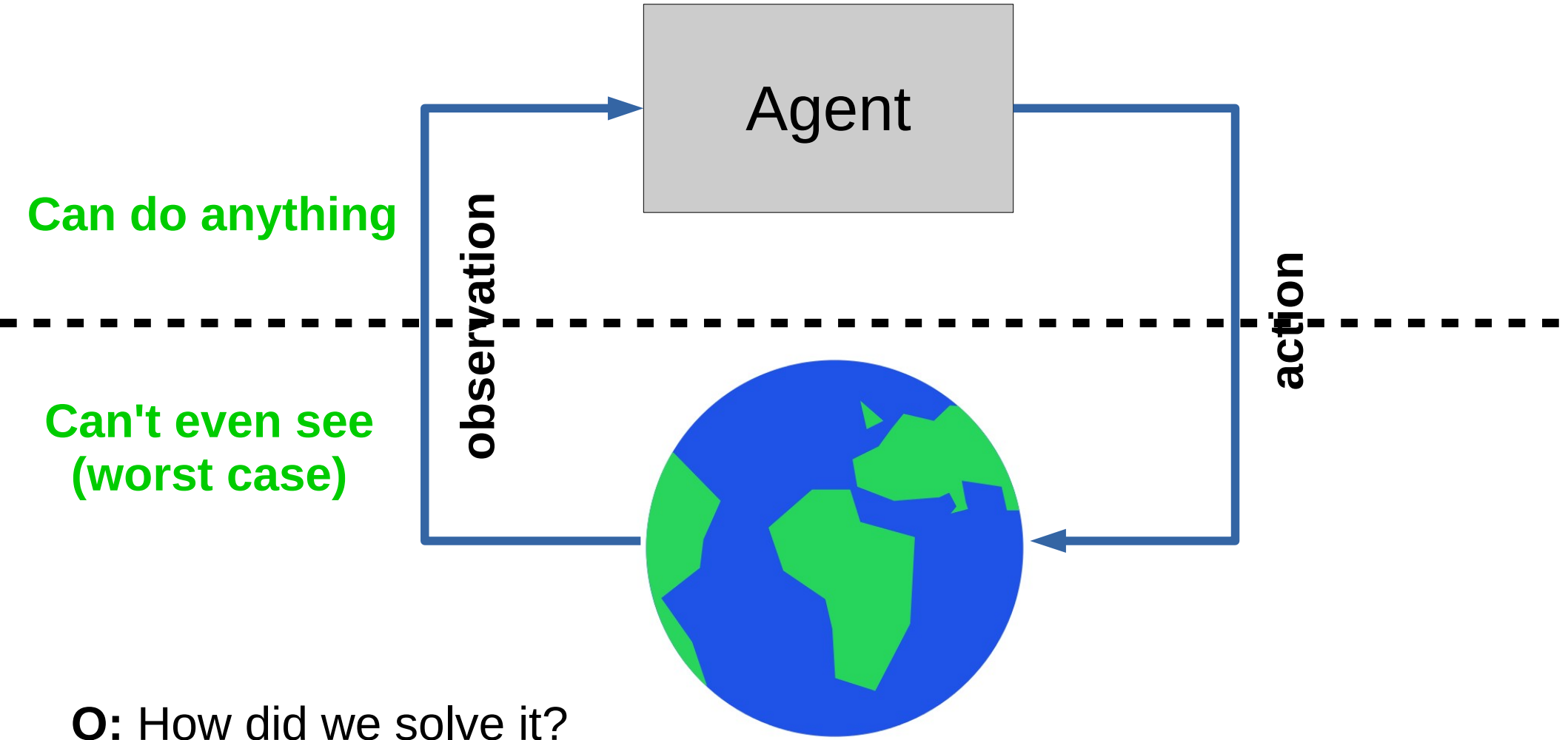
Episode 2

Temporal Difference

Recap: reinforcement learning



Recap: reinforcement learning



Q: How did we solve it?

Black box methods

- Metaheuristics (simulated annealing)
- Evolution strategies
- Crossentropy method

  …

Black box drawbacks

- Both need a full session to start learning
- Requires a lot of interaction
  - A lot of crashed robots / simulations

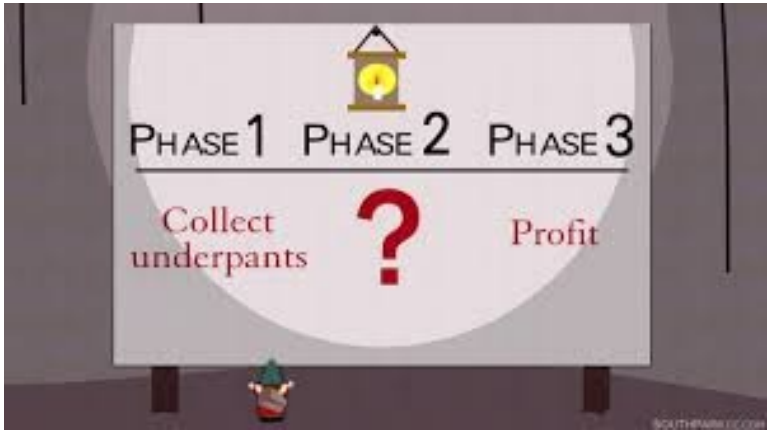
Today: value-based methods

- Core idea:
  1. You are at state **s**
  2. Compute expected reward for session if you take each possible action (a1,a2,a3,...)

  Then what? (e.g. chess)

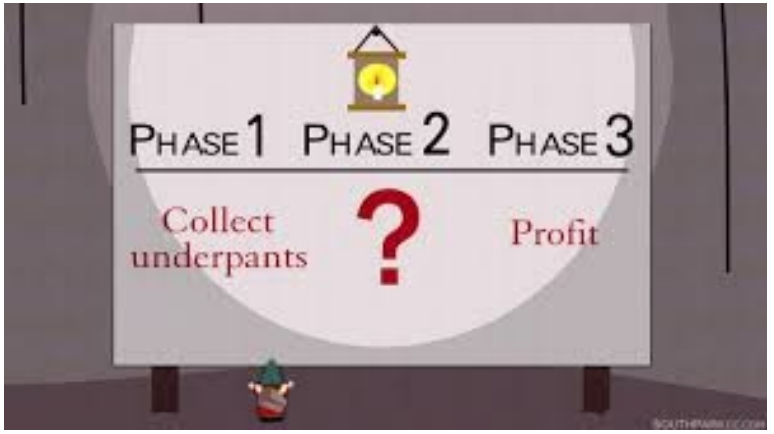
  You can compute expected win rate for each move.

  What do you do?

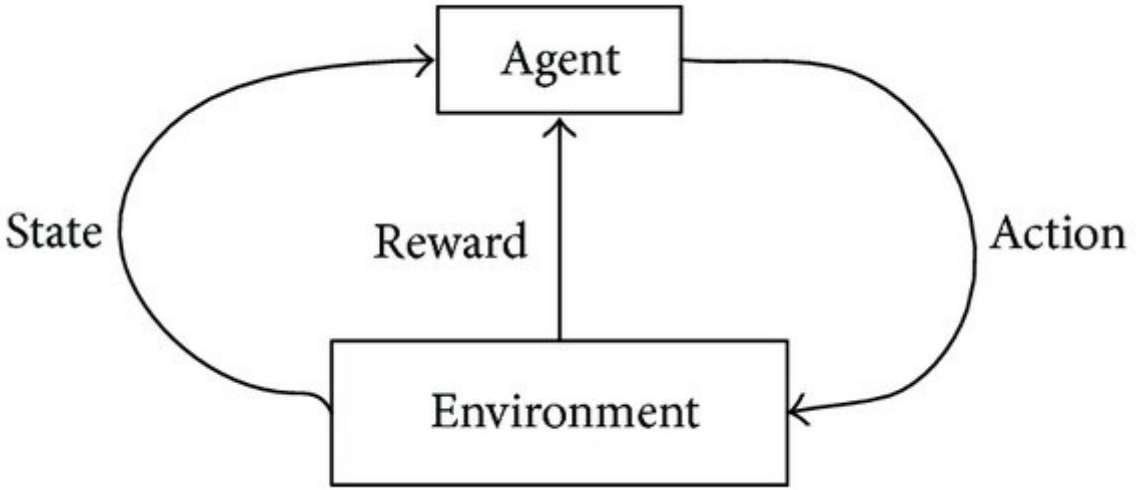


Today: value-based methods

- Core idea:
  1. You are at state **s**
  2. Compute expected reward for session if you take each possible action (a1,a2,a3,...)
  3. Take action with highest expected reward!



MDP formalism: reward on each tick



Classic MDP(Markov Decision Process)
Agent interacts with environment

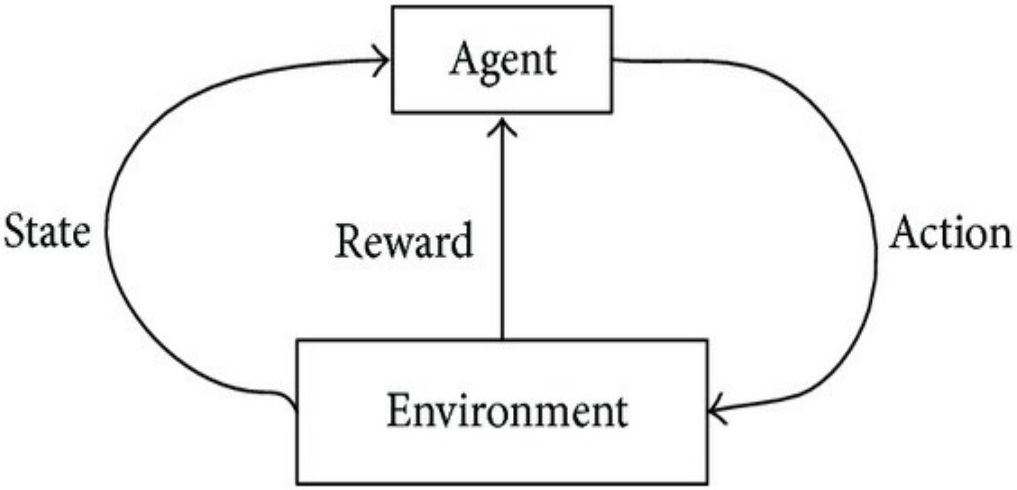
- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1}|s_t, a_t)$
- Reward: $r_t = r(s_t, a_t)$

Model-based setup

What we know

- State transitions

$P(s_{next}|s,a)$ or $s_{next}=T(s,a)$



- Rewards $r(s,a)$

Model-based setup

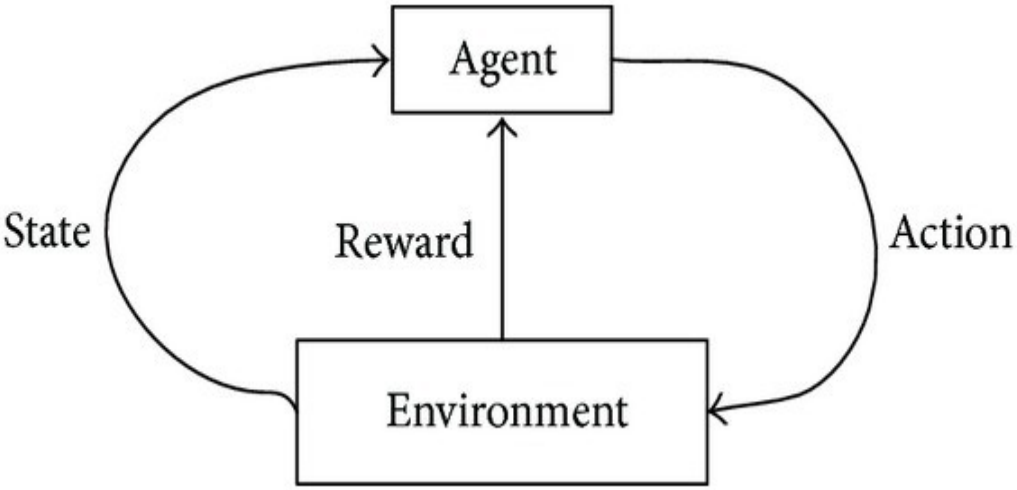
What we know

- State transitions

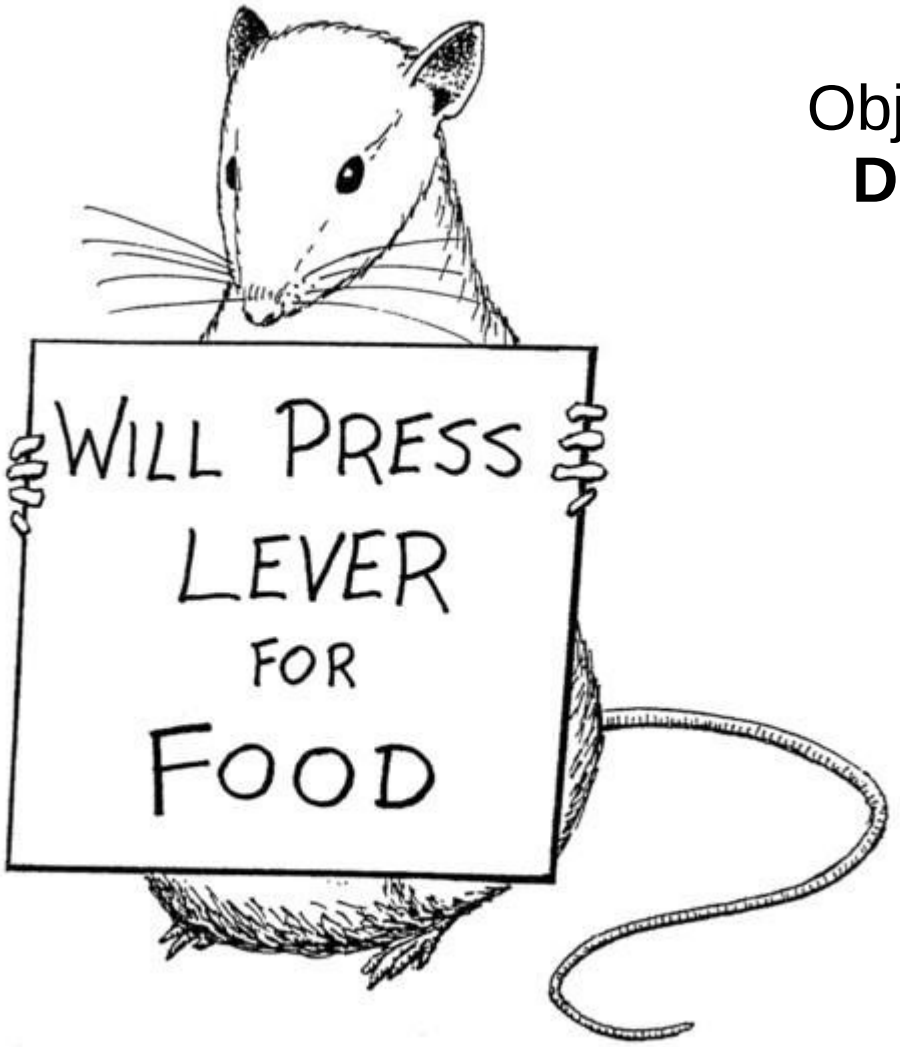
$P(s_{next}|s,a)$ or $s_{next}=T(s,a)$

Weaker version: we can only sample from P(s'|s,a)

- Rewards $r(s,a)$



Discounted reward MDP



Objective:

Discounted return G

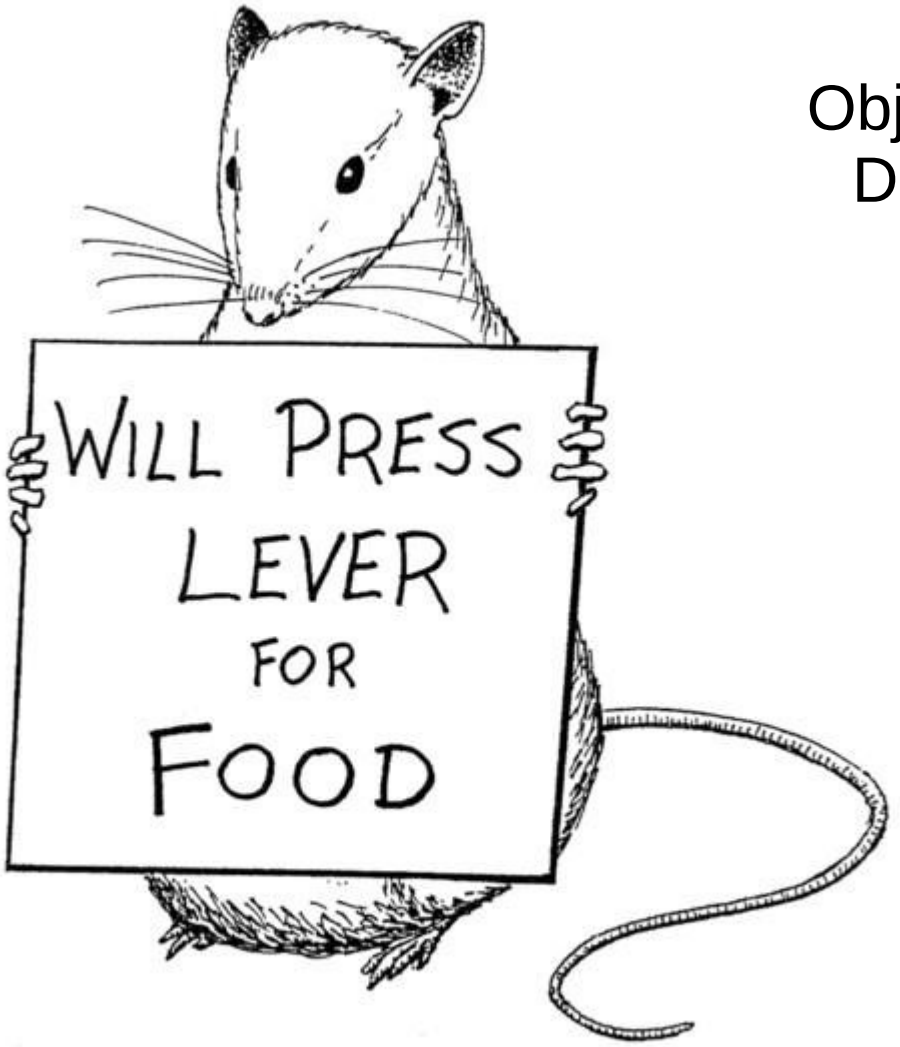
$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

γ ~ patience

Cake tomorrow is γ as good as now

Discounted reward MDP



Objective:
Discounted return G

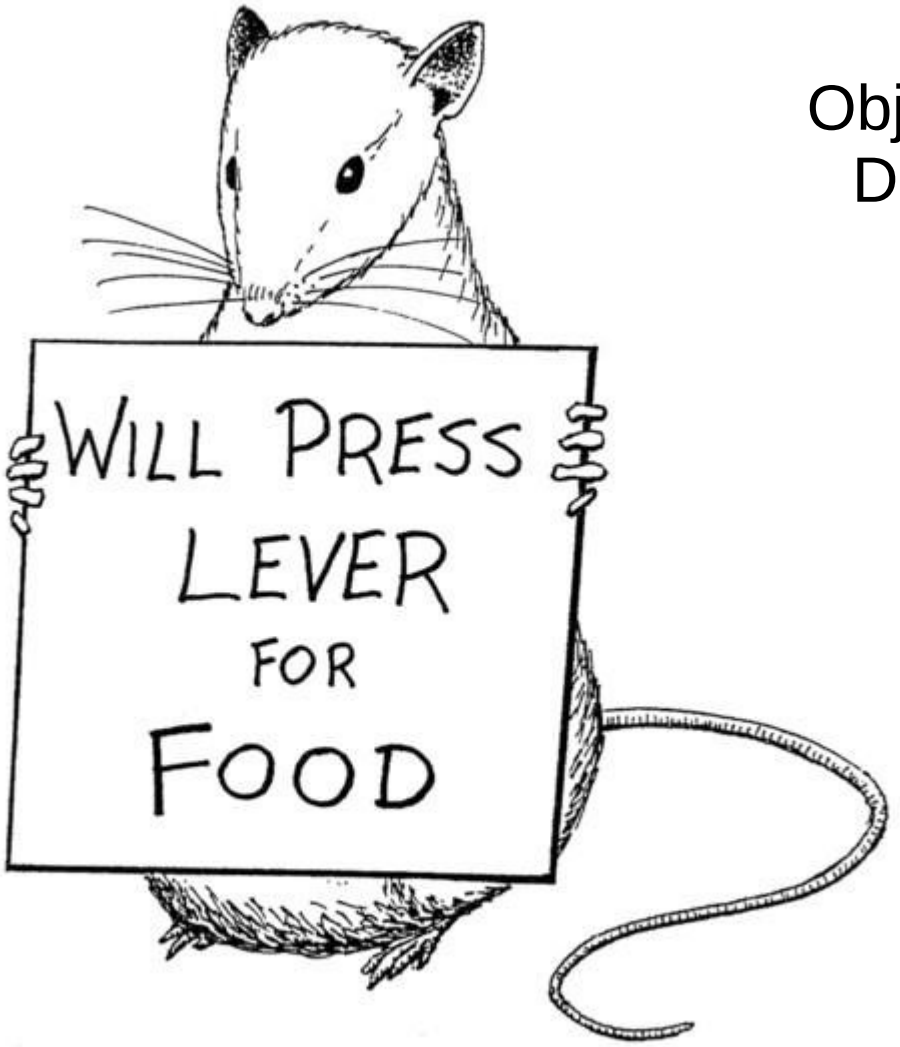
$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

γ ~ patience
Cake tomorrow is γ as good as now

Q1: which γ corresponds to "only current reward matters"?
Q2: with which γ G is just a sum of rewards?

Discounted reward MDP



Objective:
Discounted return G

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

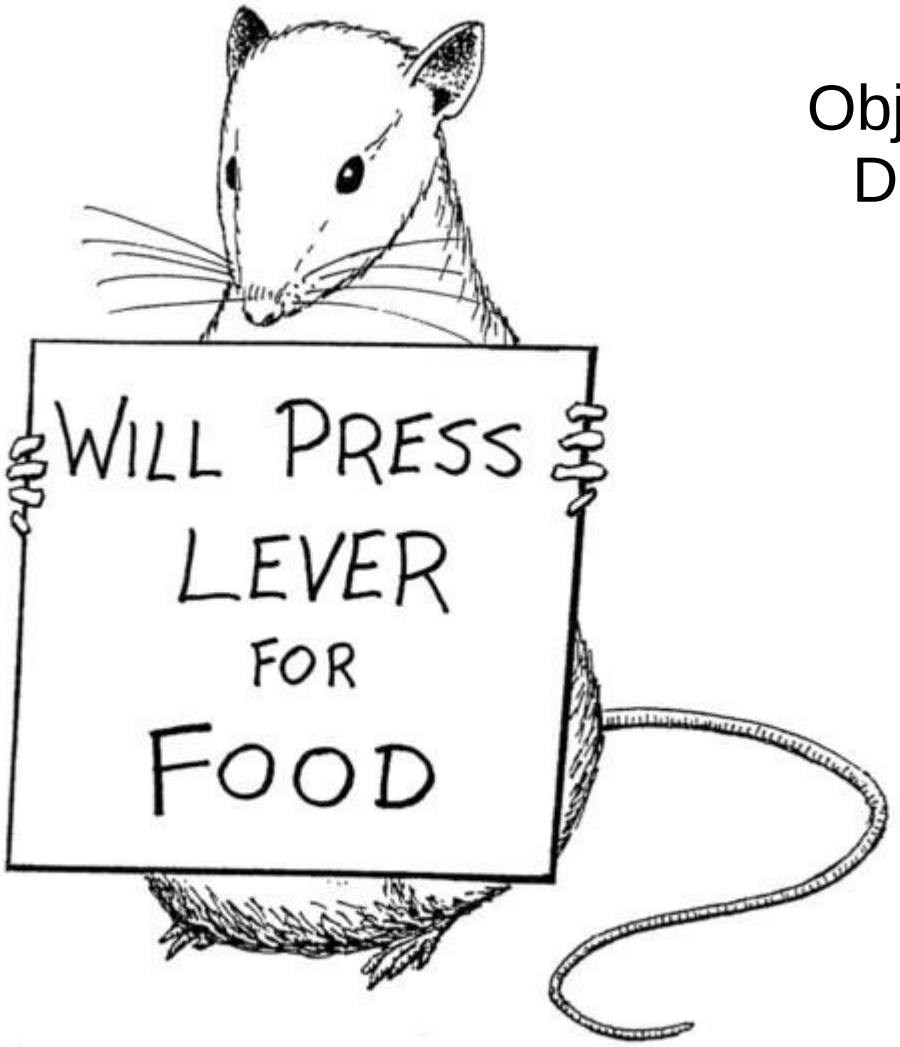
$$G_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) \, const$$

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow max$$

Discounted reward MDP



Objective:
Discounted return G

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

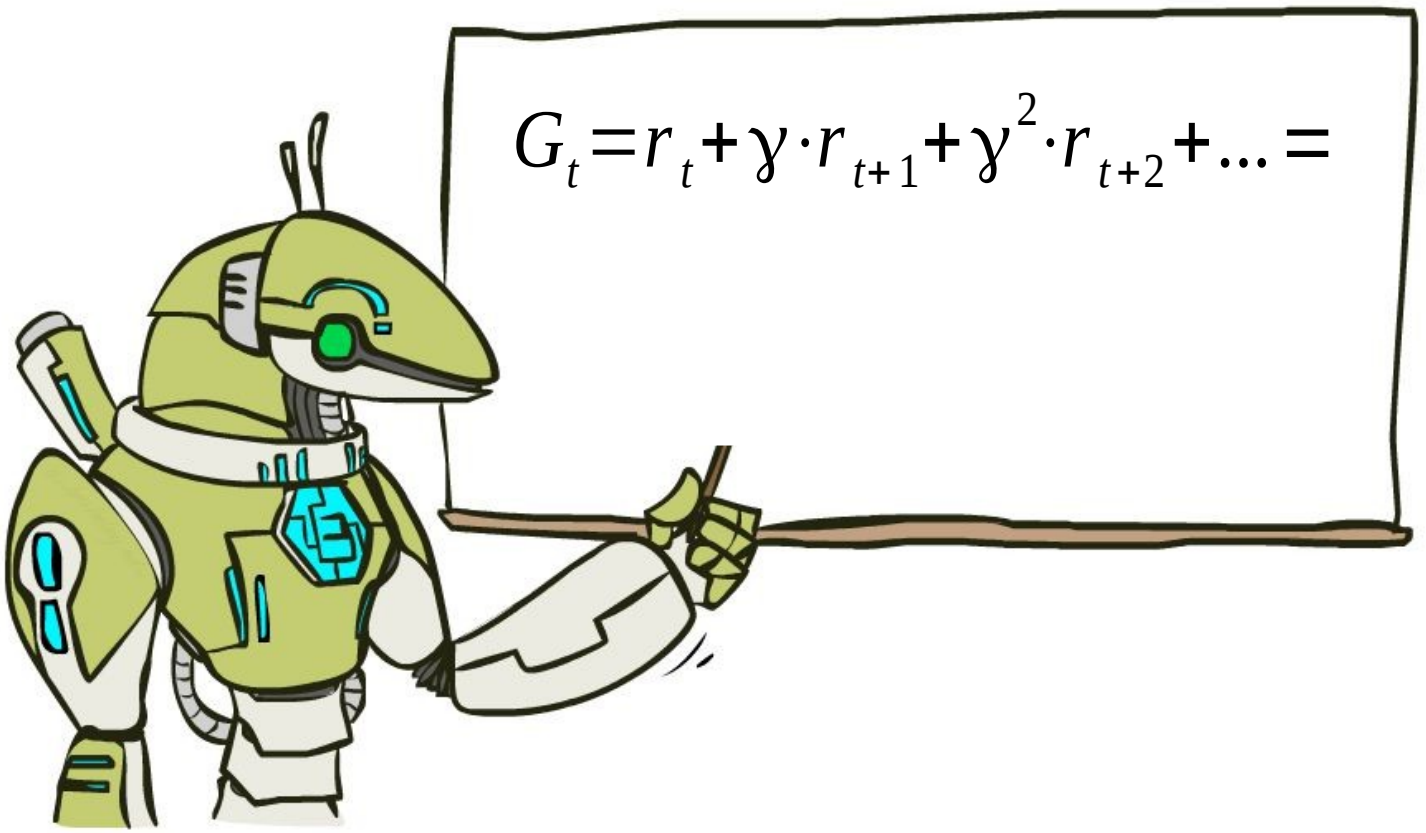
Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow max$$

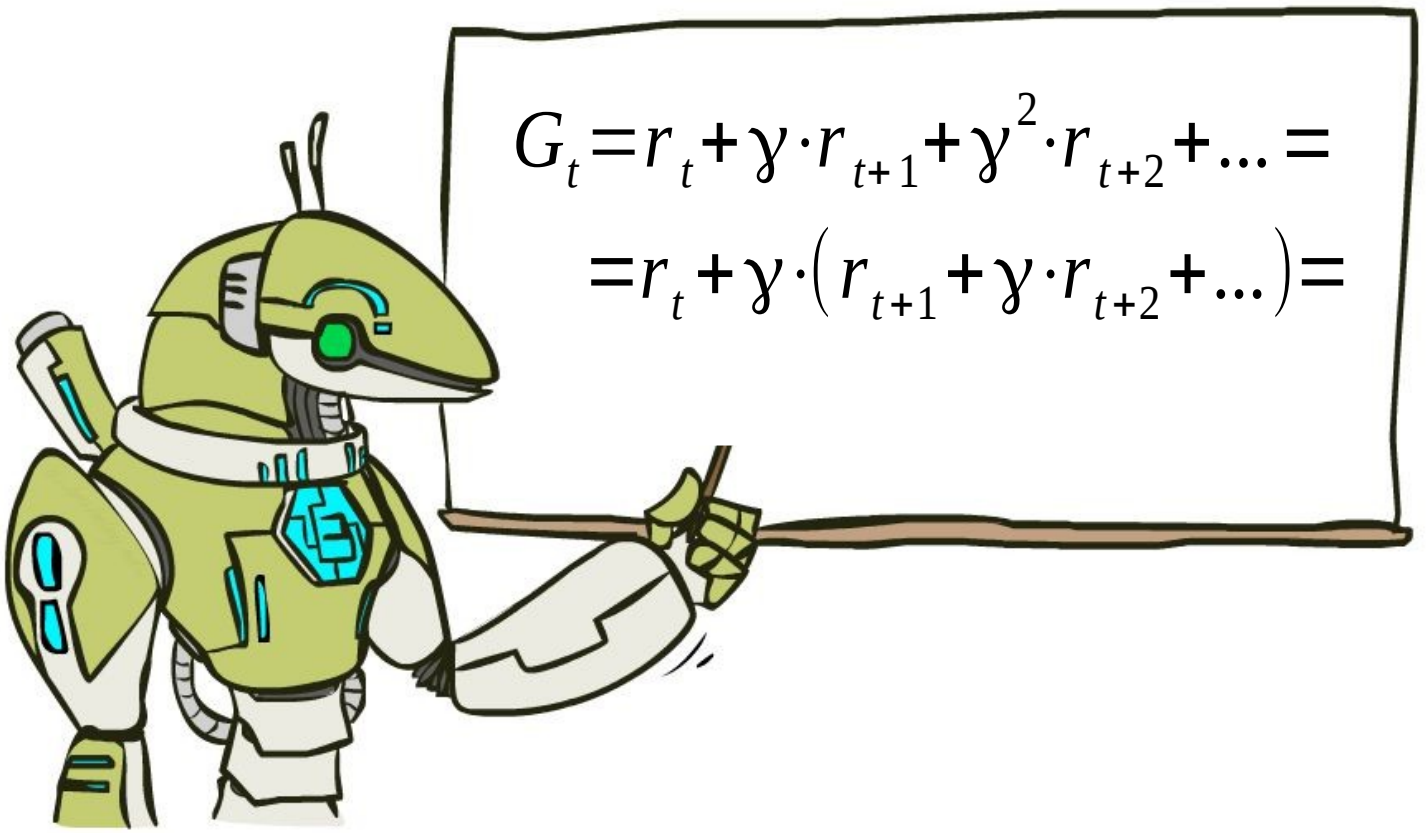
Does not maximize sum of rewards
Unless γ = 1

Dealing with G



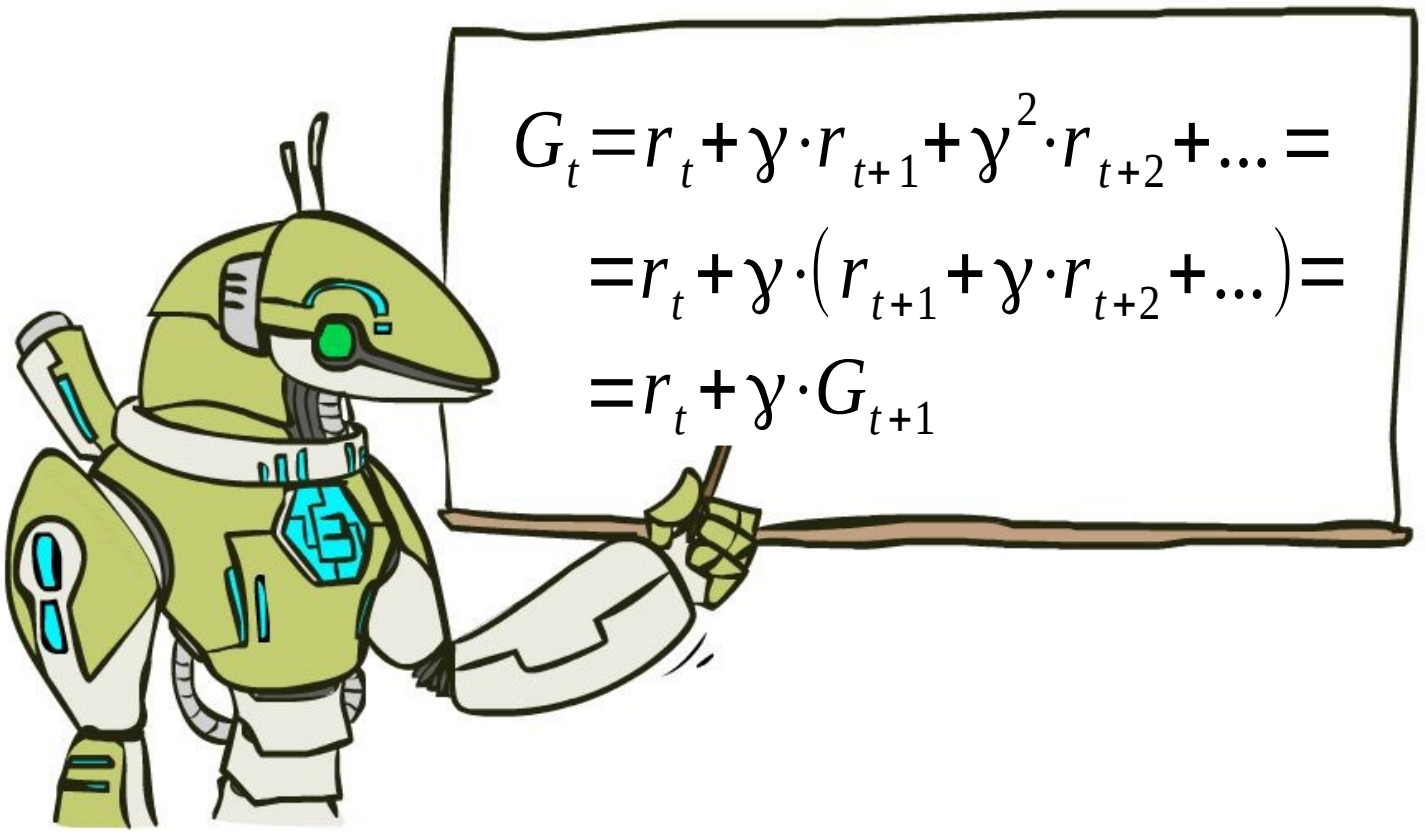
We rewrite G with sheer power of math!

Dealing with G



We rewrite G with sheer power of math!

Dealing with G



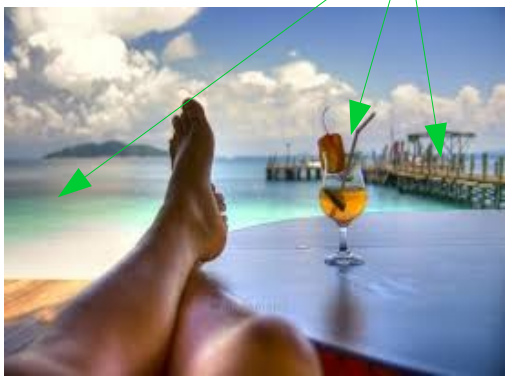
We rewrite G with sheer power of math!

More new letters!

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

large potential rewards



vs

small potential rewards

More new letters!

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \underset{a \sim \pi(a|s)}{E} \ldots$$

State value

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \underset{a \sim \pi(a|s)}{E} \quad \underset{s', r \sim P(s'|s,a)}{E} \quad \ldots$$

State value

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \underset{a\sim\pi(a|s)}{E} \; \underset{s',r\sim P(s'|s,a)}{E} \; \underset{a',r',s'',\ldots}{E} \; \ldots$$

State value

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \underset{a\sim\pi(a|s)}{E} \; \underset{s',r\sim P(s'|s,a)}{E} \; \underset{a',r',s'',\ldots}{E} \; r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

State value

State values : $V_\pi(s)$

- **Definition:** $V_\pi(s)$ – expected total reward G that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \mathop{E}_{\tau \sim p(\tau|...)} G(\tau)$$

trajectory **τ** = (s,a,r,s',a',r',...)

Bellman equation

State values : $V_\pi(s)$

Definition: $V_\pi(s)$ – expected total reward R that can be obtained starting from state s and following policy π.

$$V_\pi(s) = \mathop{E}_{a\sim\pi(a|s)} \mathop{E}_{s',r\sim P(s',r|s,a)} [r + \gamma \cdot V_\pi(s')]$$

Bellman equation

State values : $V_\pi(s)$

Definition: $V_\pi(s)$ – expected total reward R that can be obtained starting from state s and following policy π.

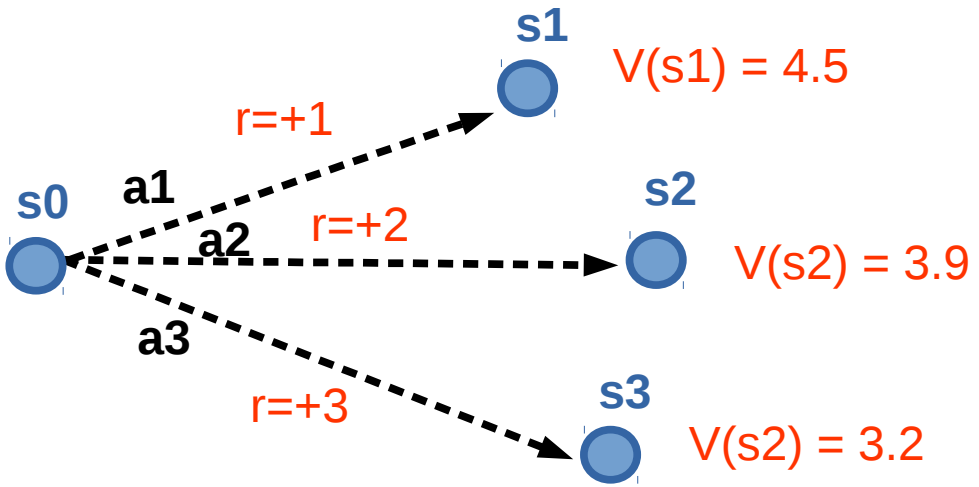
$$V_\pi(s) = \underset{a\sim\pi(a|s)}{E} \; \underset{s',r\sim P(s',r|s,a)}{E} \left[r + \gamma \cdot V_\pi(s') \right]$$

how I like **s** = what **r** I can get right now + γ * how I like **s'**

Optimal policy

Imagine we know exact value of any $V_\pi(s)$.
(also P(s',r|s,a))

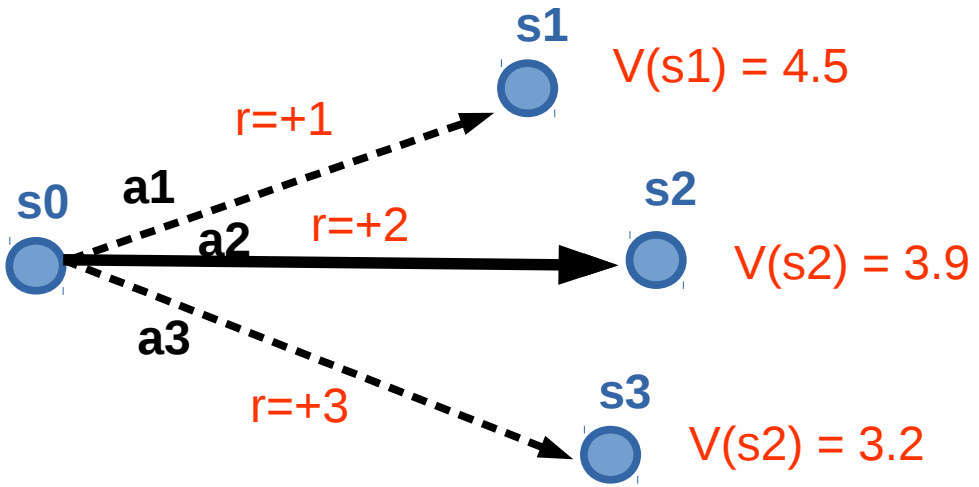
Q: how do we pick optimal action?



Optimal policy

Imagine we know exact value of any $V_\pi(s)$.
(also P(s',r|s,a))

Q: how do we pick optimal action?



$$\pi^*(a|s) = \underset{a}{argmax} \underset{s',r \sim P(s',r|s,a)}{E} [r + \gamma \cdot V_{\pi^*}(s')]$$

V*(s) and π*(a|s)

Optimal policy : $\pi^*(a|s)$

$$\pi^*(a|s) = \underset{a}{argmax} \underset{s',r \sim P(s',r|s,a)}{E} [r + \gamma \cdot V_{\pi^*}(s')]$$

Optimal state value : $V^*(s)$

$$V^*(s) = V_{\pi^*}(s) = \underset{a}{max} \underset{s',r \sim P(s',r|s,a)}{E} [r + \gamma \cdot V(s')]$$

Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$

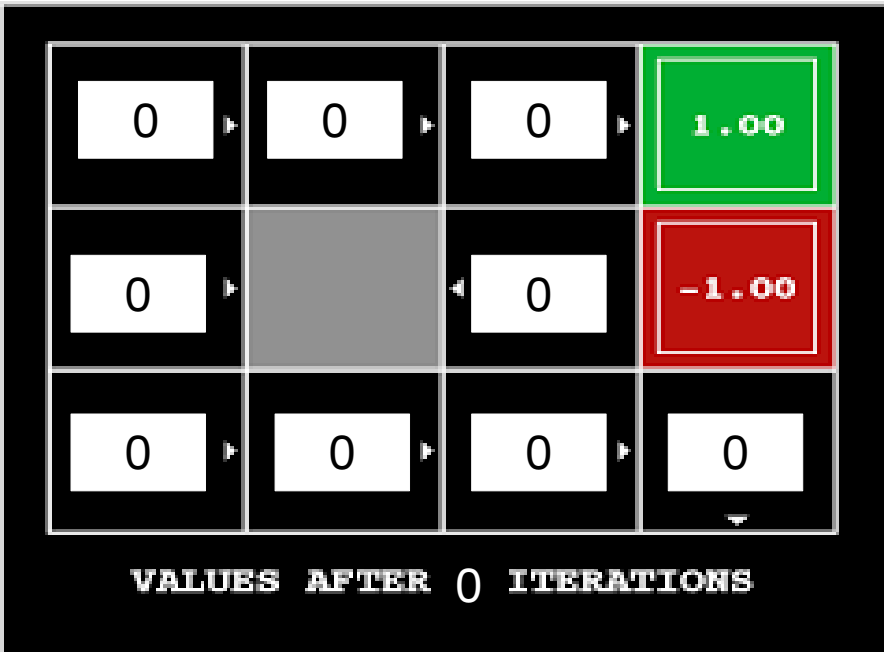
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



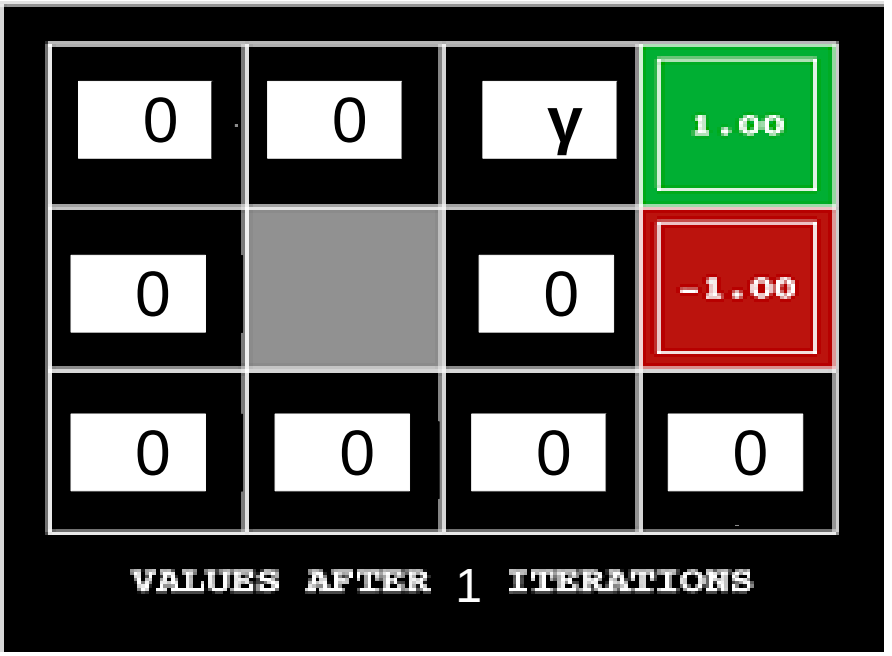
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



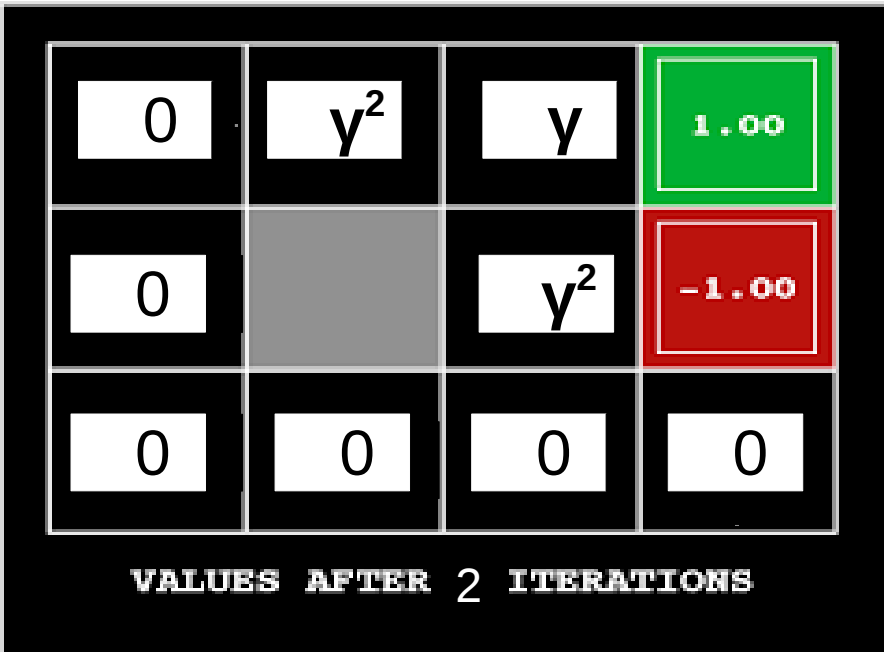
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



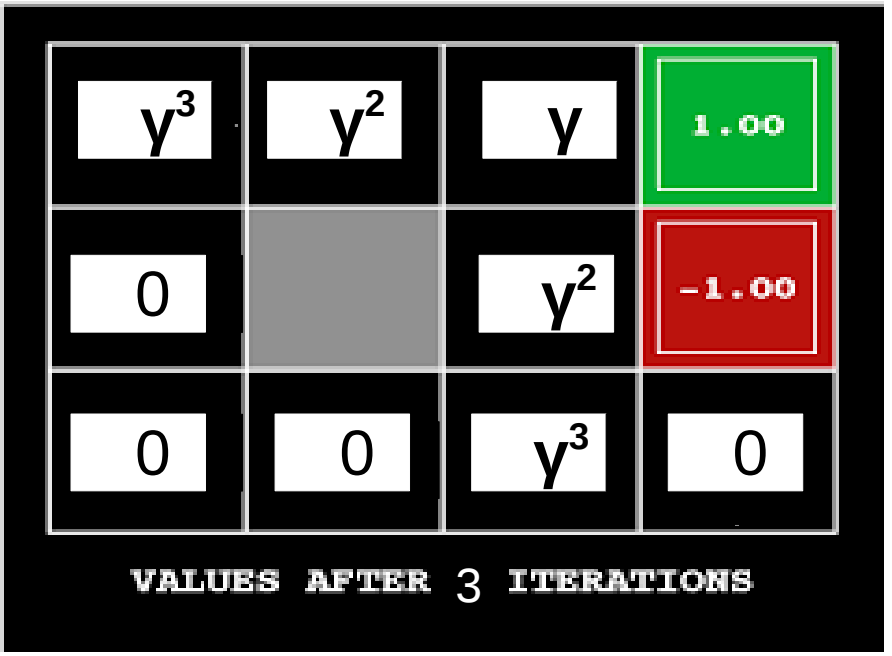
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



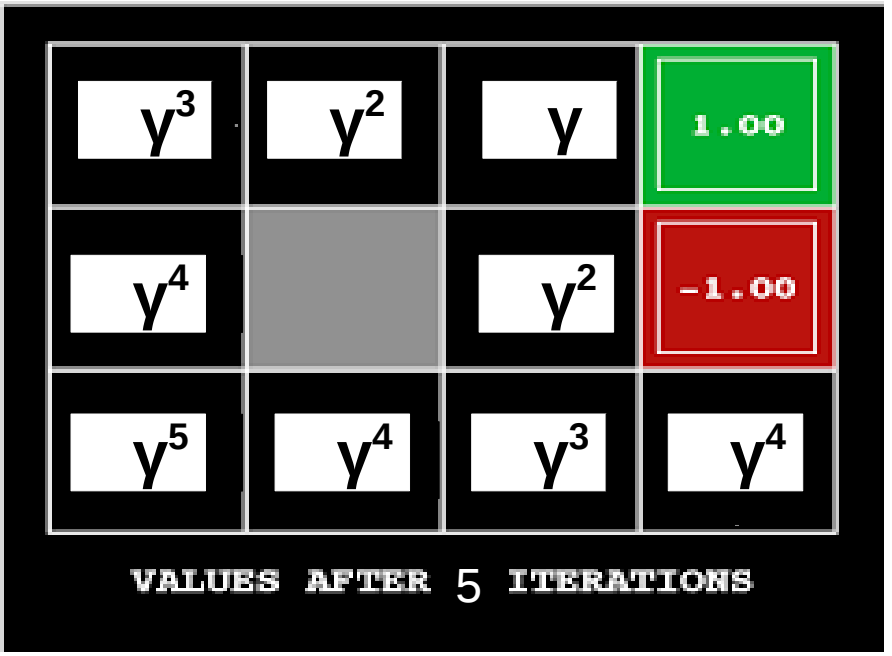
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



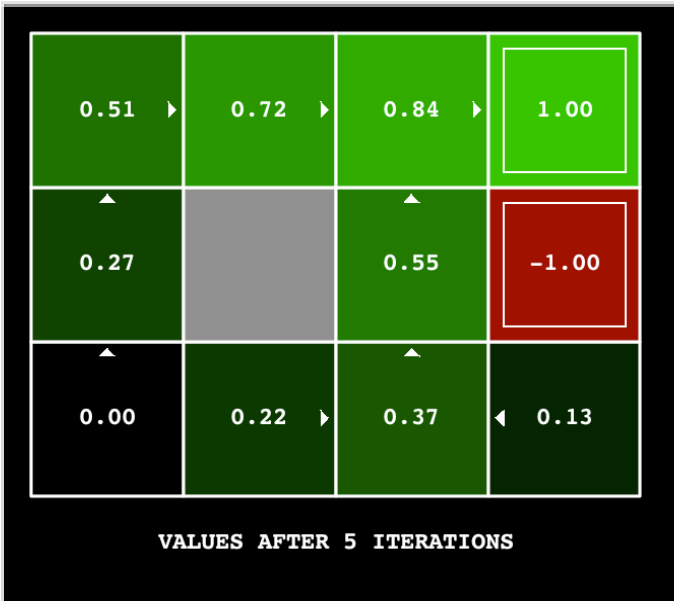
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



At scale

Q: What to do if we have large/continuous state space?

We can no longer explore all states!

At scale

Q: What to do if we have large/continuous state space?

We can no longer explore all states!

Idea: we can only search some neighboring states!

Adversarial setup

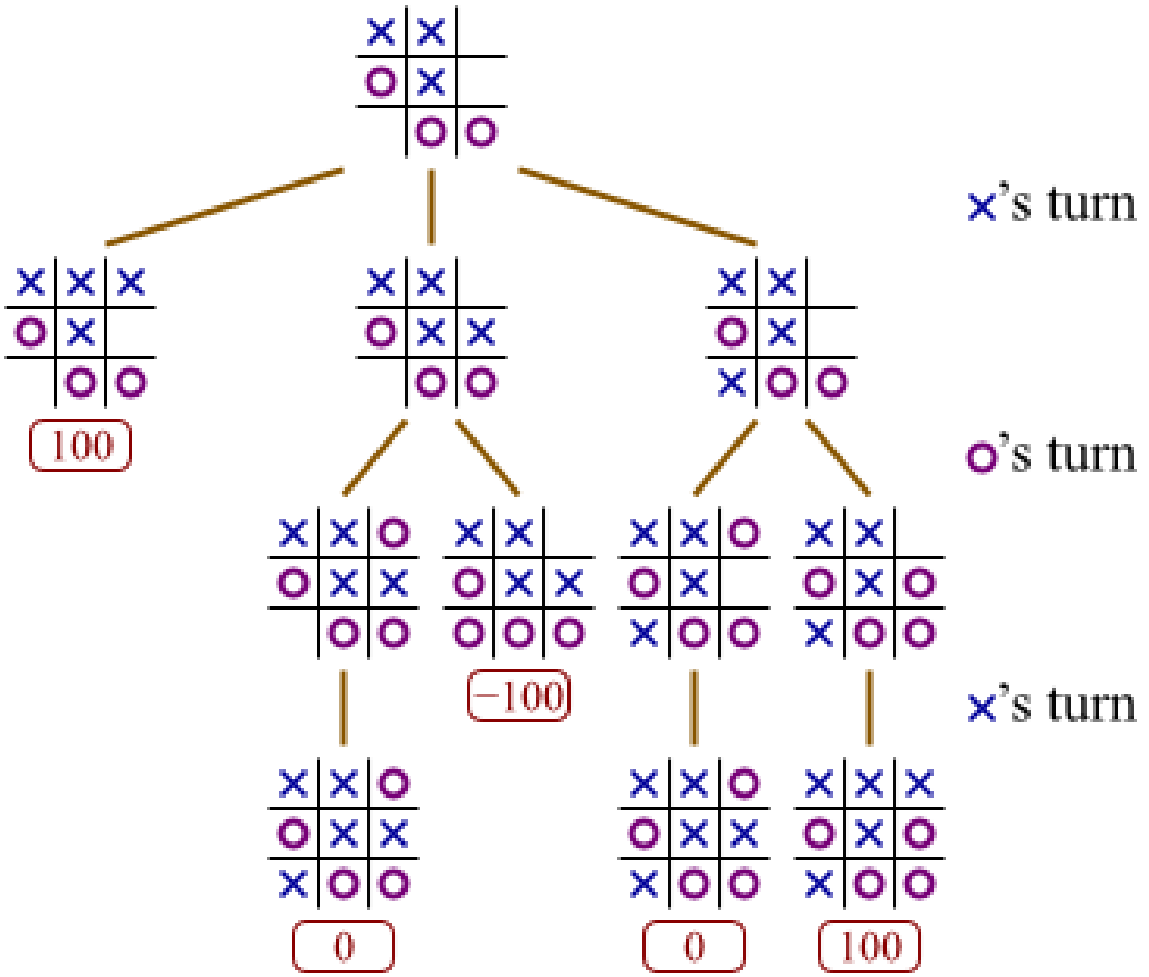
Your agent plays a deterministic game with another agent in it.

And he's playing against us!

We want highest expected reward.
He wants to minimize our expected reward.

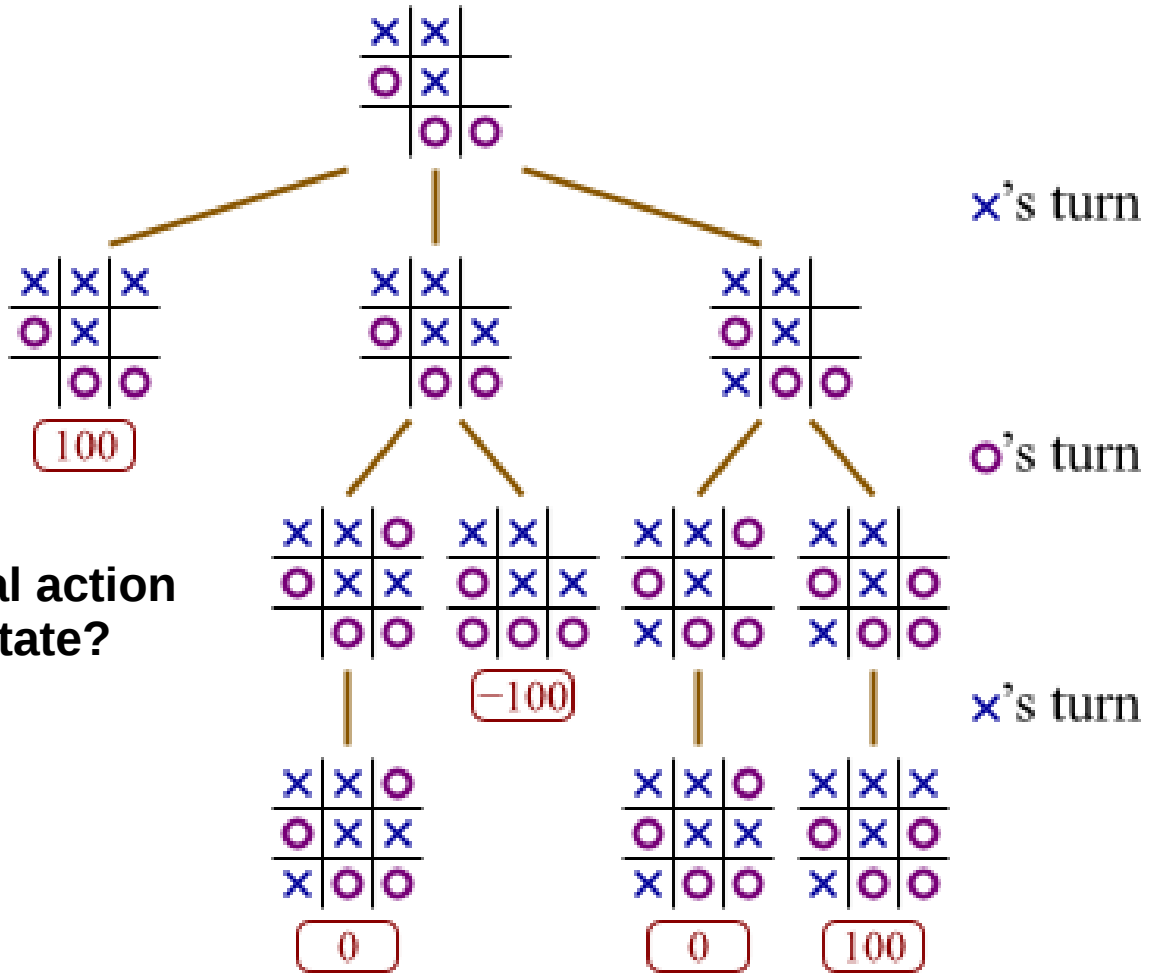
Examples:
- Any board game: chess, checkers, go
- Pong :)

Adversarial search trees

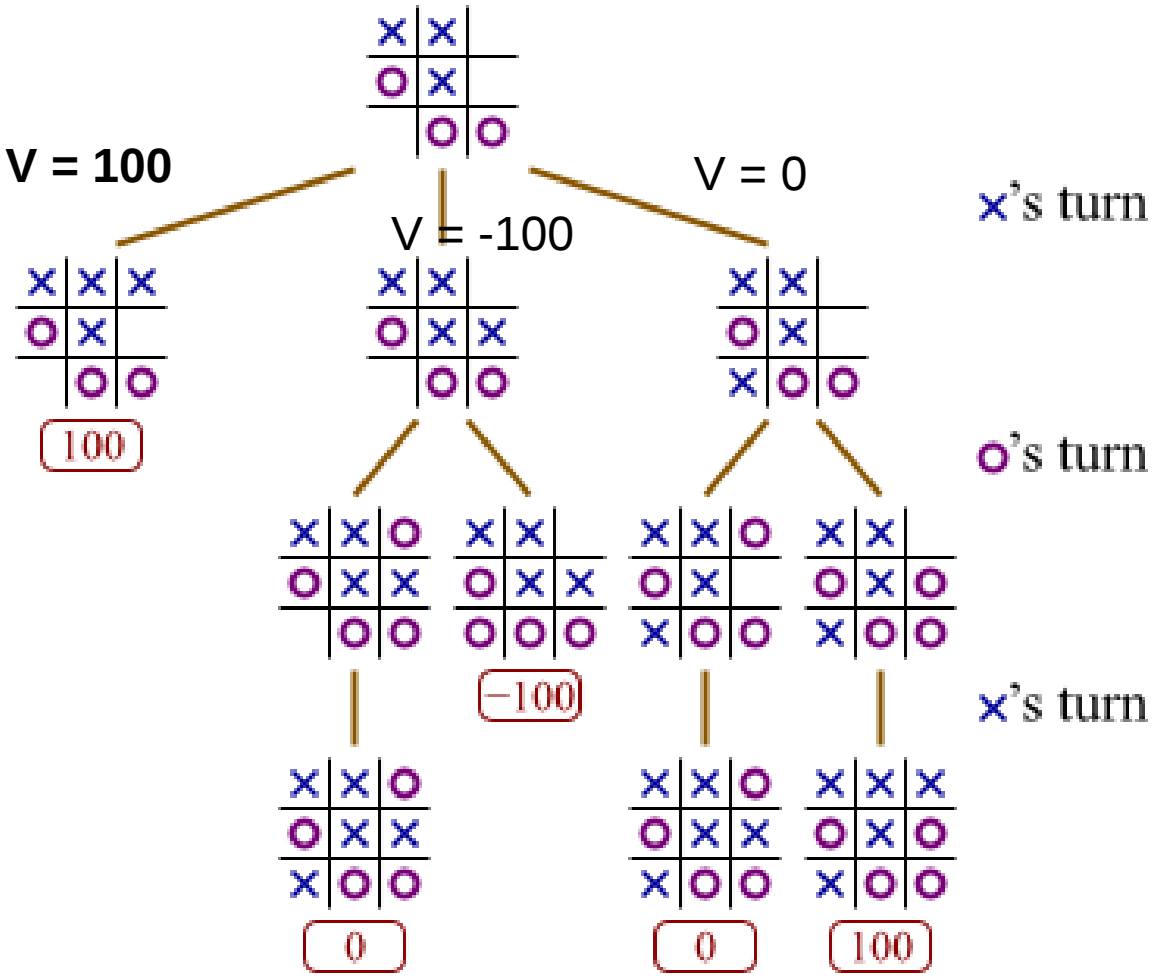


Adversarial search trees

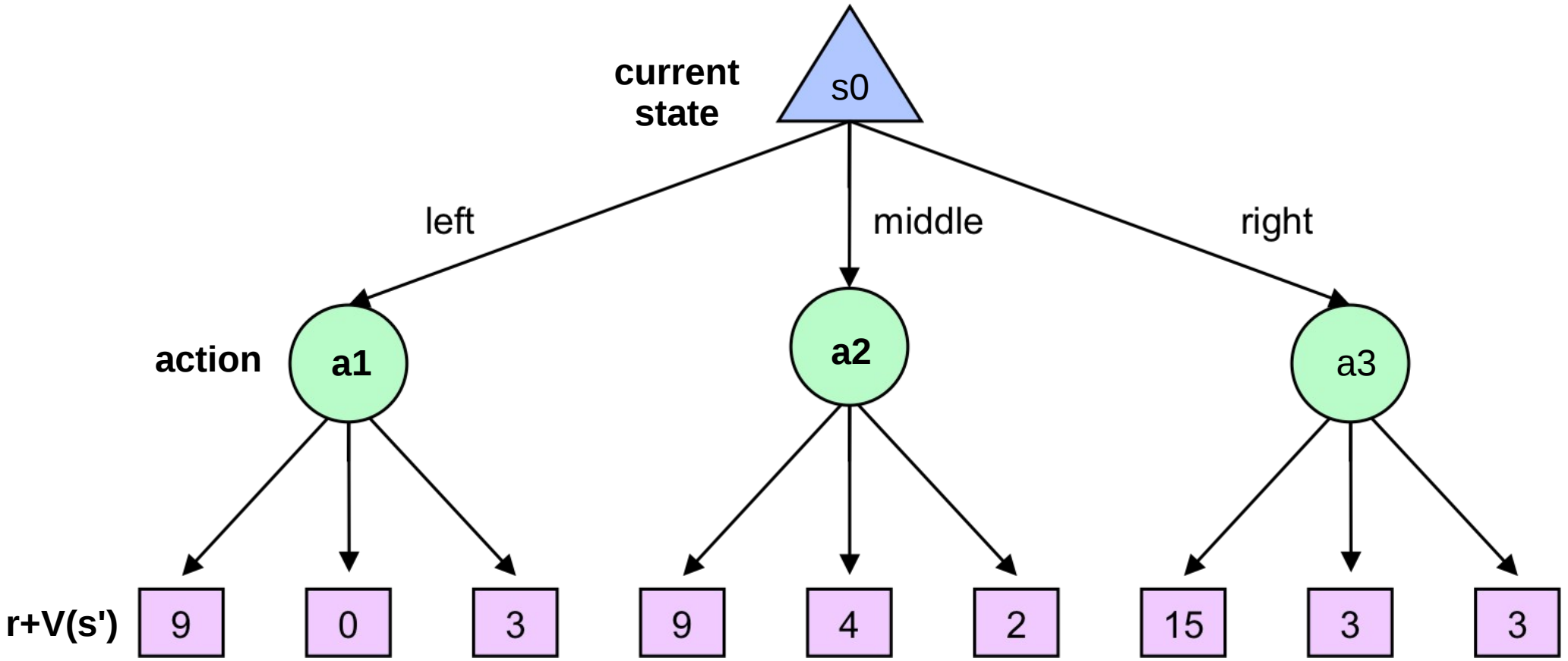
What's optimal action from top state?



Adversarial search trees



Stochastic search trees



How to evaluate action value?

Large/continuous state space

We can't explore all the nodes.

Need to pick most interesting ones!

Examples:

- ~any practical use case :)
- Atari

Count-based exploration

UCB-1 for bandits

Idea:

Prioritize actions with uncertain outcomes!

Less times visited = more uncertain.

Math: add upper confidence bond to reward.

Count-based exploration

UCB-1 for bandits

Take actions in in proportion to $\tilde{v}_a$

$$\tilde{v}_a = v_a + \sqrt{\frac{2 \log N}{n_a}}$$

Upper conf. bound for r in [0,1]

- N number of time-steps so far
- n_a times action **a** is taken

Count-based exploration

UCB-1 for bandits

Take actions in in proportion to $\tilde{v}_a$

$$\tilde{v}_a = v_a + \sqrt{\frac{2 \log N}{n_a}}$$

- N number of time-steps so far
- n_a times action **a** is taken

Count-based exploration

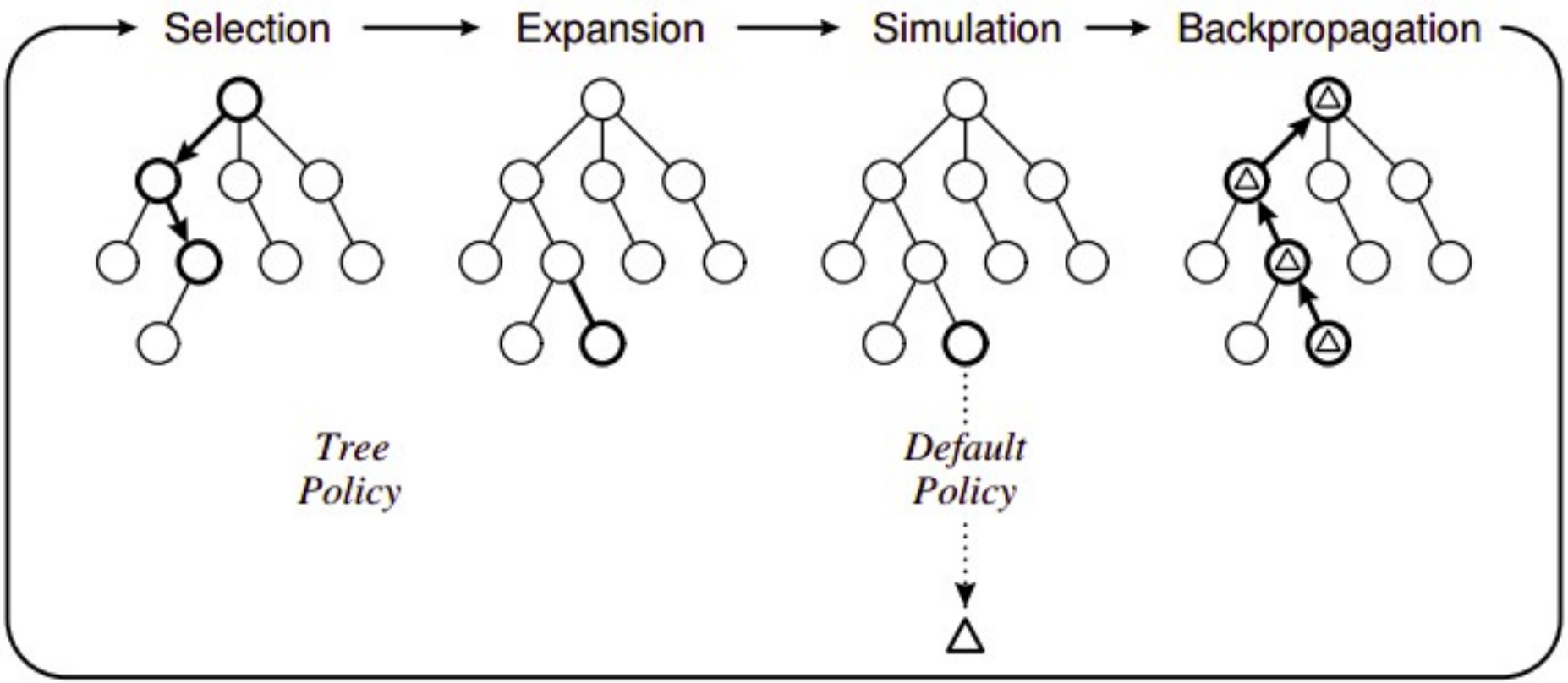
UCB generalized for multiple states

$$\tilde{Q}(s,a) = Q(s,a) + \alpha \cdot \sqrt{\frac{2 \log N_s}{n_{s,a}}}$$

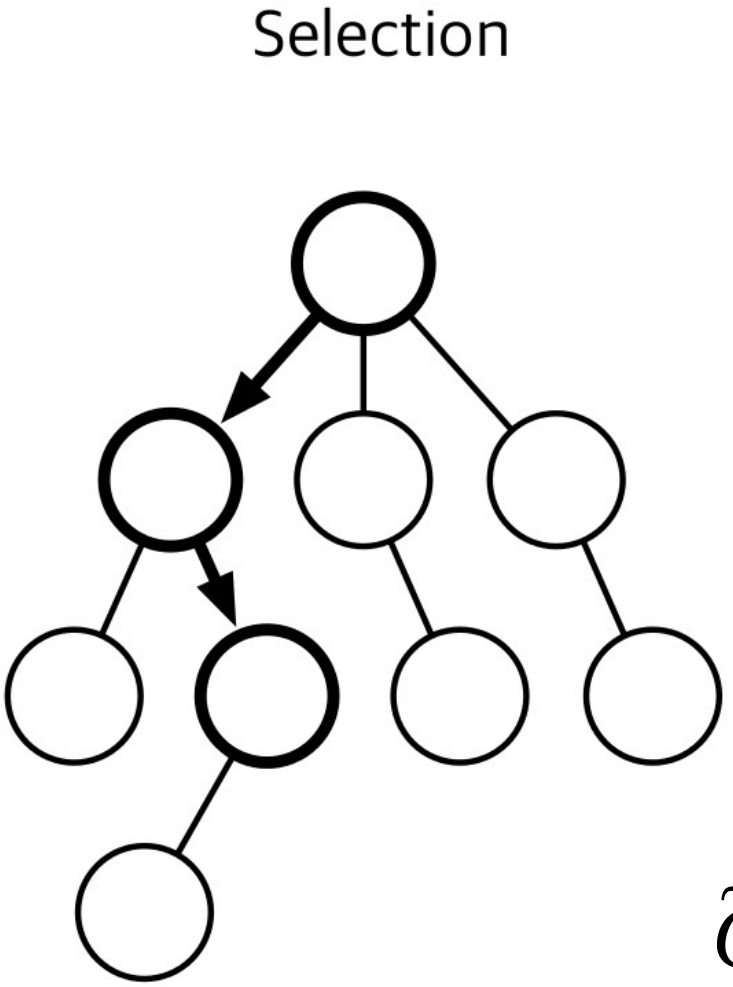
where

- N_s visits to state **s**
- $n_{s,a}$ times action **a** is taken from state **s**

MCTS


Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

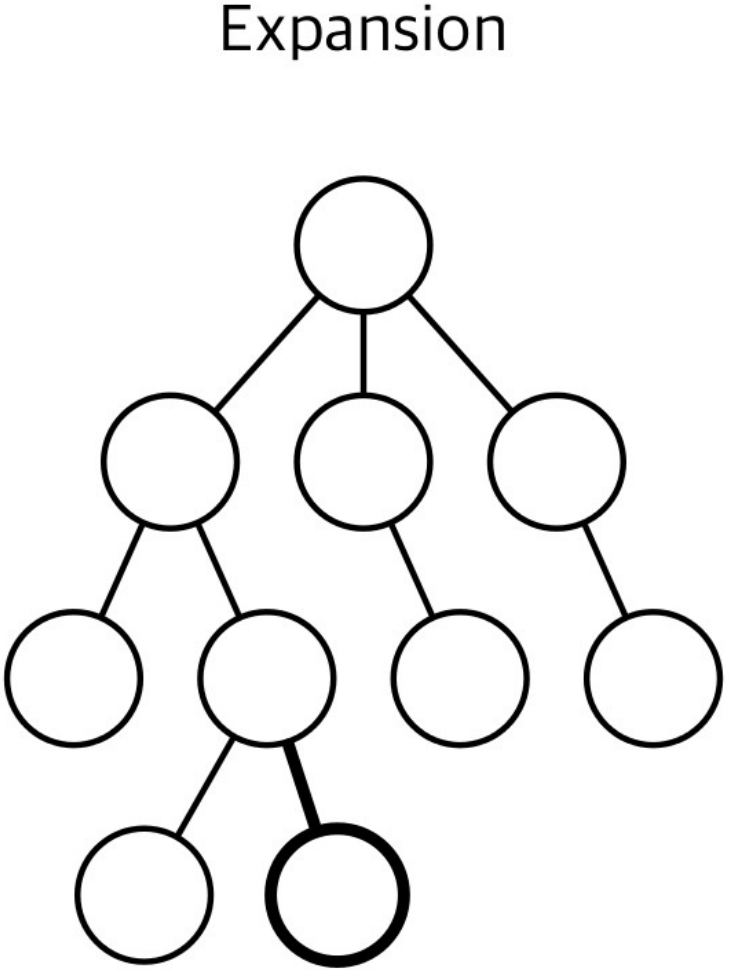
MCTS: selection



Starting from the root, recursively select node with highest ucb-1 score

$$\tilde{Q}(s,a) = Q(s,a) + \alpha \cdot \sqrt{\frac{2 \log N_s}{n_{s,a}}}$$

MCTS: Expansion

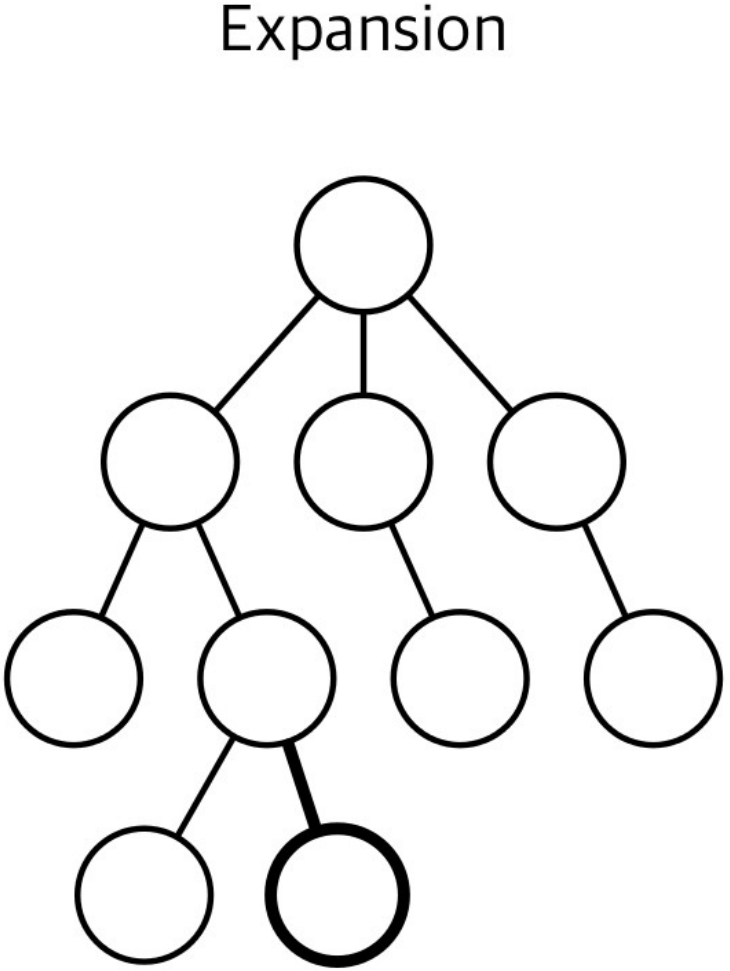


Add one or more children from the chosen node.

Each child is a one-step simulation **s** → **s', a, r**

Simple case: add one node per action.

MCTS: Expansion



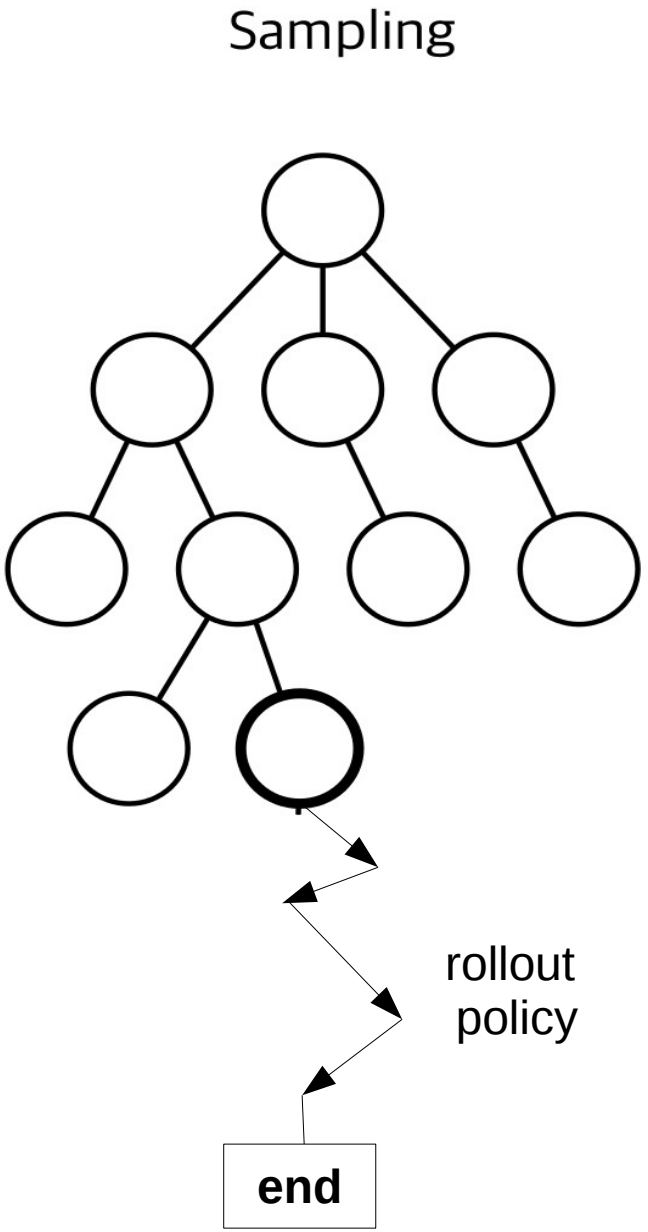
Add one or more children from the chosen node.

Each child is a one-step simulation **s → s', a, r**

Simple case: add one node per action.

Any ideas when this is won't work?

MCTS: Rollout (sampling)

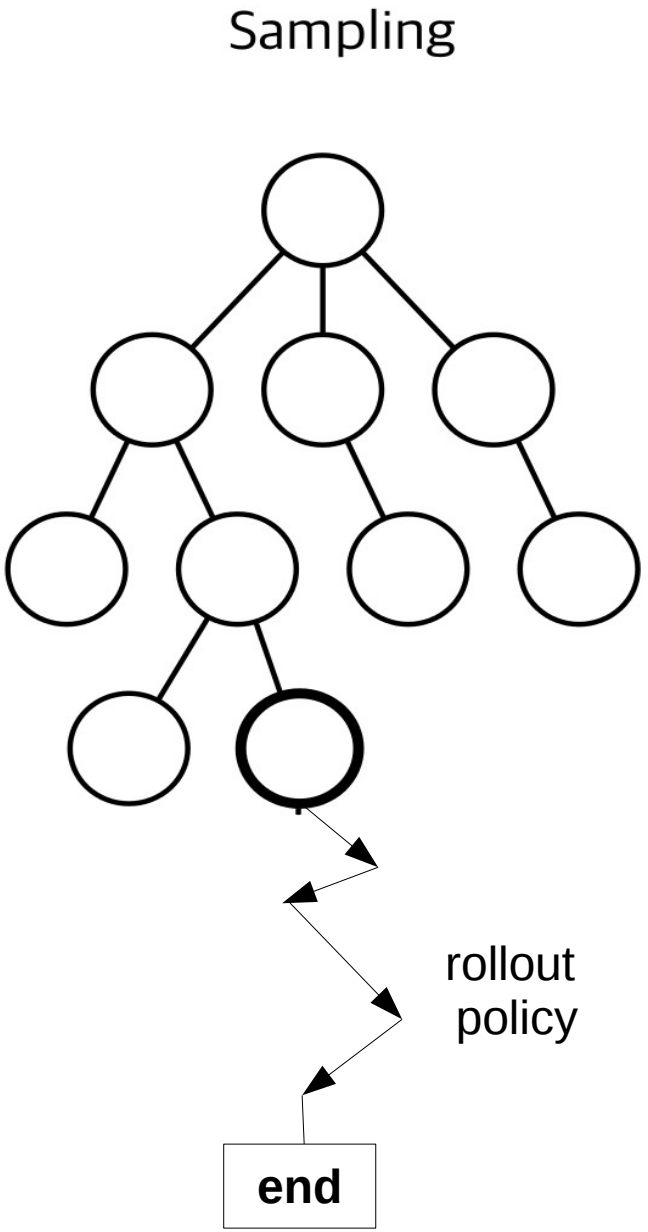


Estimate node value by playing game from that state till the end with simple policy.

e.g. random actions

Remember total reward.

MCTS: Rollout (sampling)



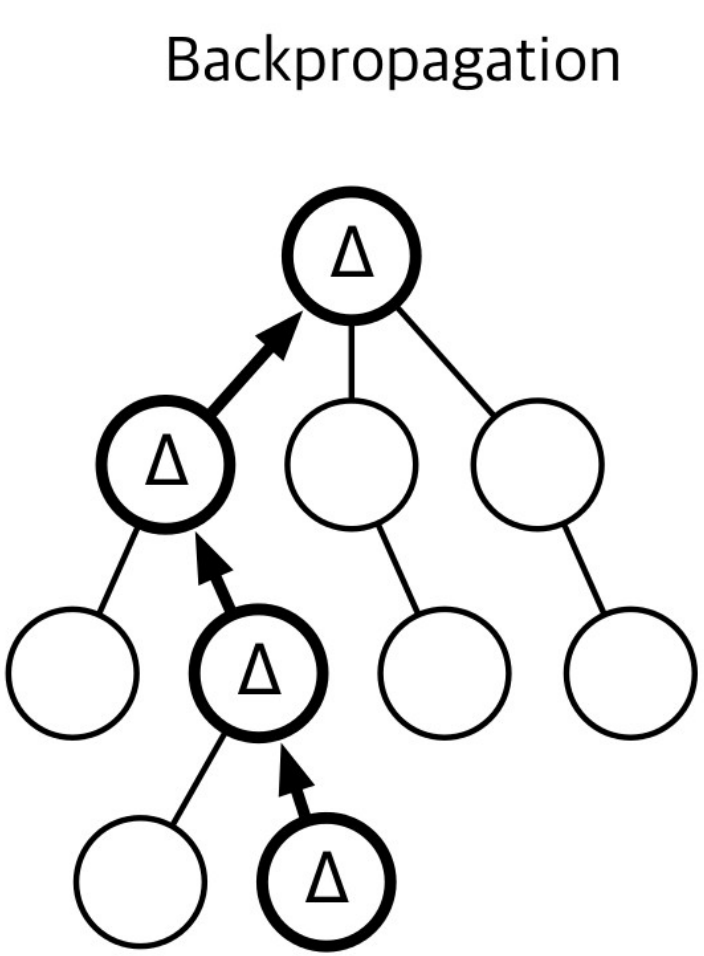
Estimate node value by playing game from that state till the end with simple policy.

e.g. random actions

Remember total reward.

Can we do better than random?

MCTS: Backprop

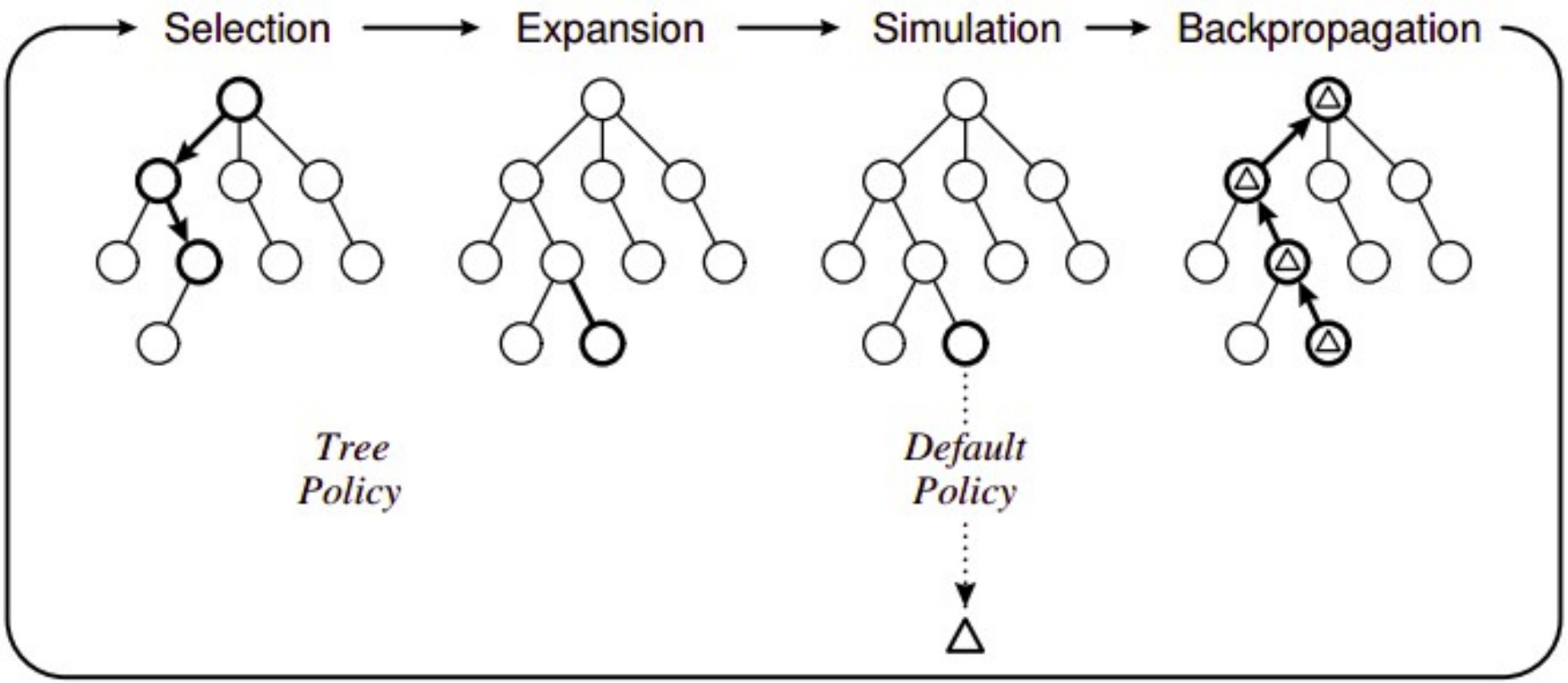


Given rollout reward, update value of leaf and all it's parents.

$$V(parent) = r + \gamma \cdot V(child)$$

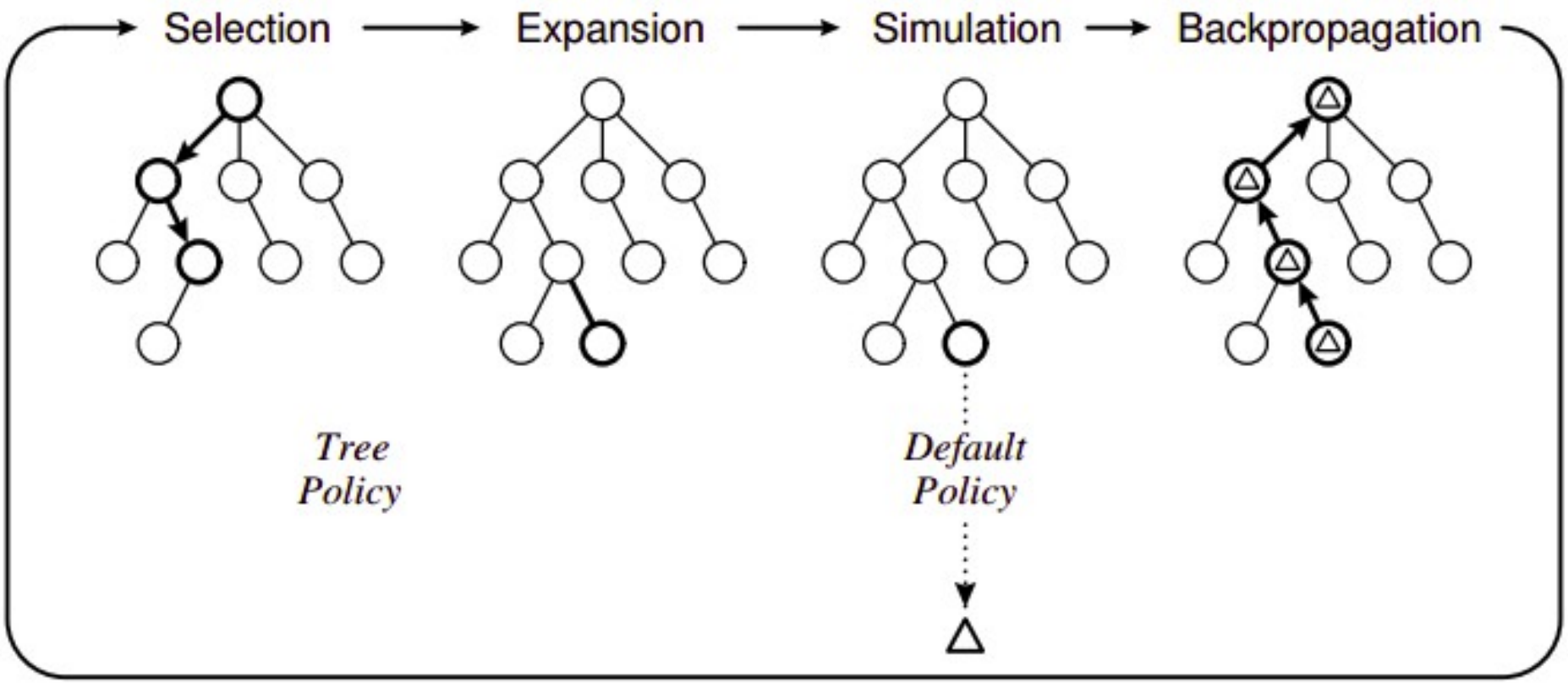
Also increment visit counts (N and n_a for ucb-1)

MCTS



How do we pick action from root?

MCTS



Pick action to maximize expected reward!

Recap

What did we learn?

Recap

What did we learn?

- Discounted rewards, G
- State values,V(s) V*(s)
- Value iteration
- Planning trees, MCTS

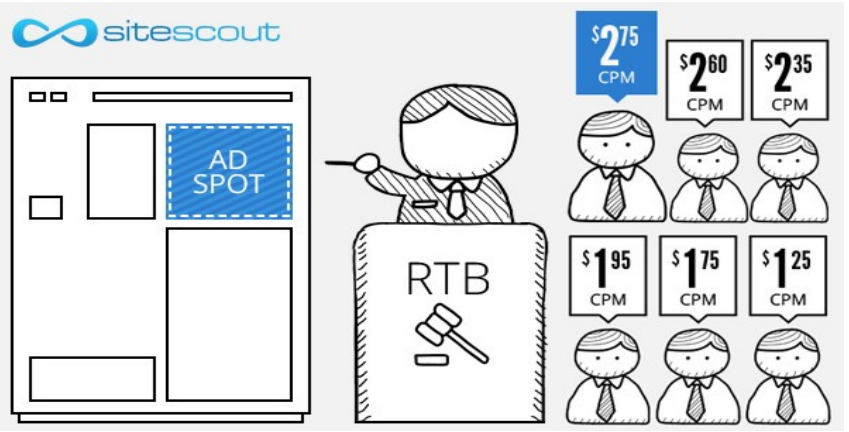
Voila! We've solved the reinforcement learning!

Or have we?

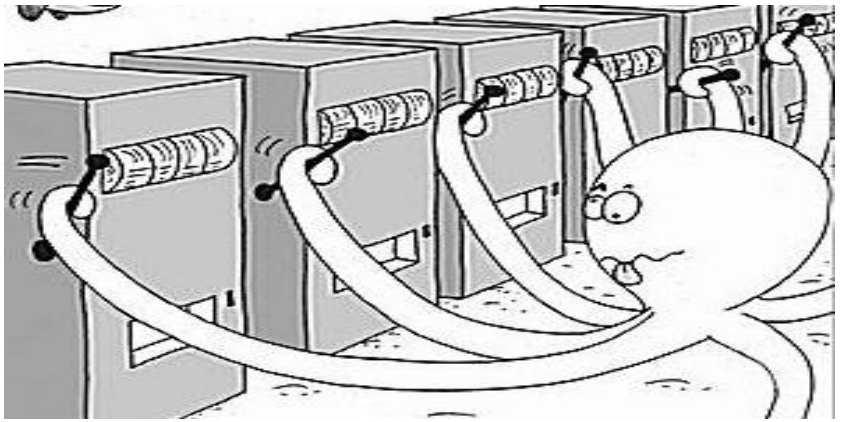
What happens if we apply it to real world problems?

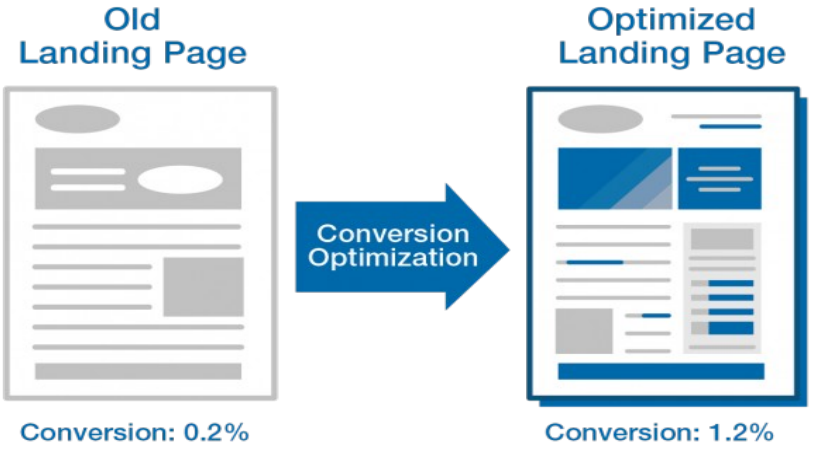
Reality check: web

- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend items to users

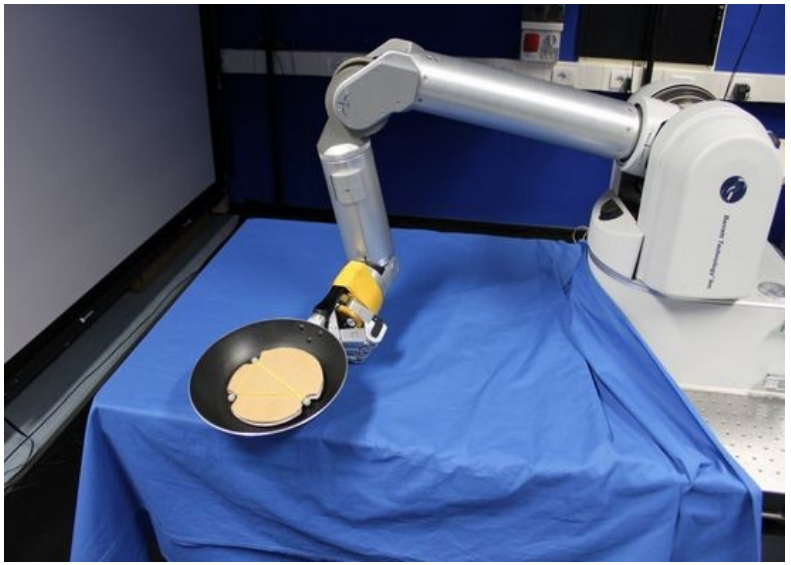


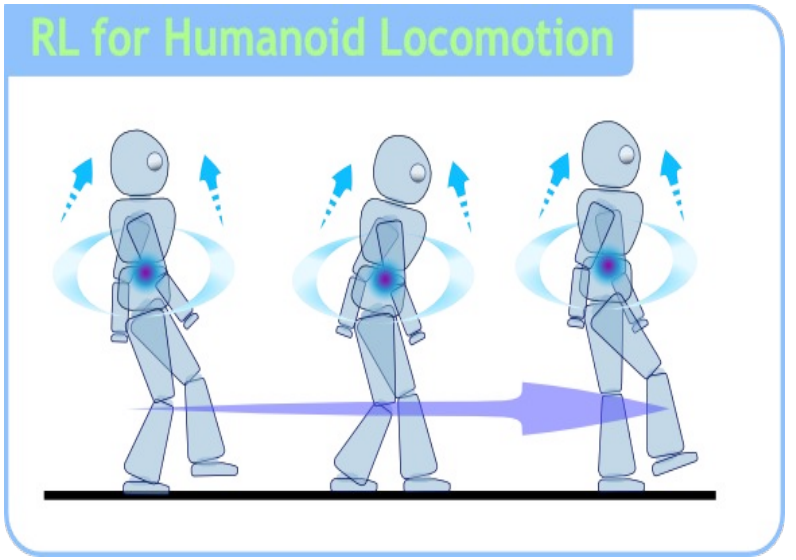
- **Common traits:**
  - Independent states
  - Large action space





Reality check: dynamic systems



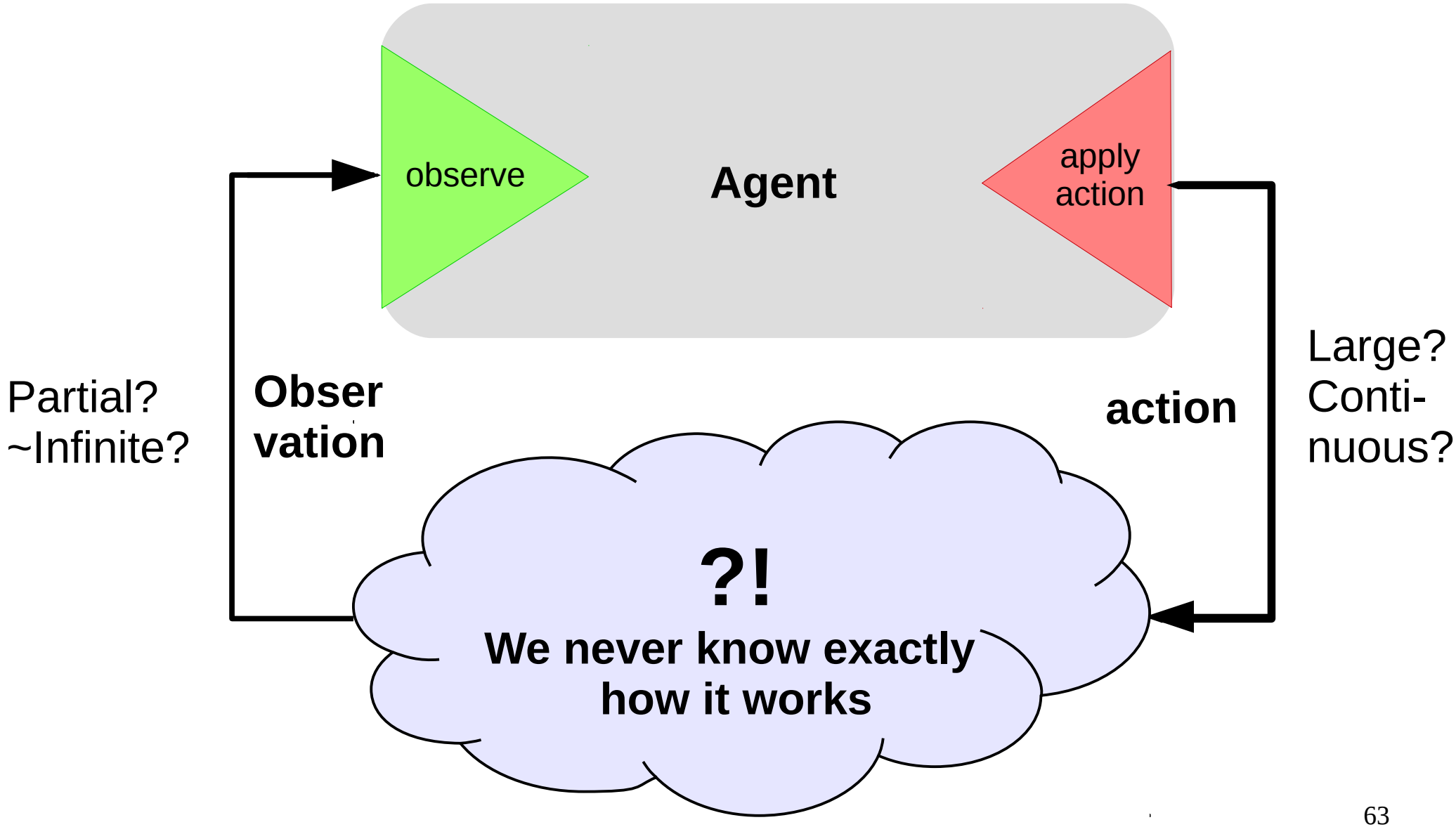


Reality check: MOAR

- **Cases:**
  - Robots
  - Self-driving vehicles
  - Pilot assistant
  - More robots!

- **Common traits:**
  - Continuous state space
  - Continuous action space
  - Partially-observable environment
  - LONG sessions

Real world


observe
Agent
apply action
Partial?
~Infinite?
Obser
vation
action
Large?
Conti-
nuous?
?!
We never know exactly
how it works

Next on practical_rl

How to deal with unknown P(s'|s,a)?

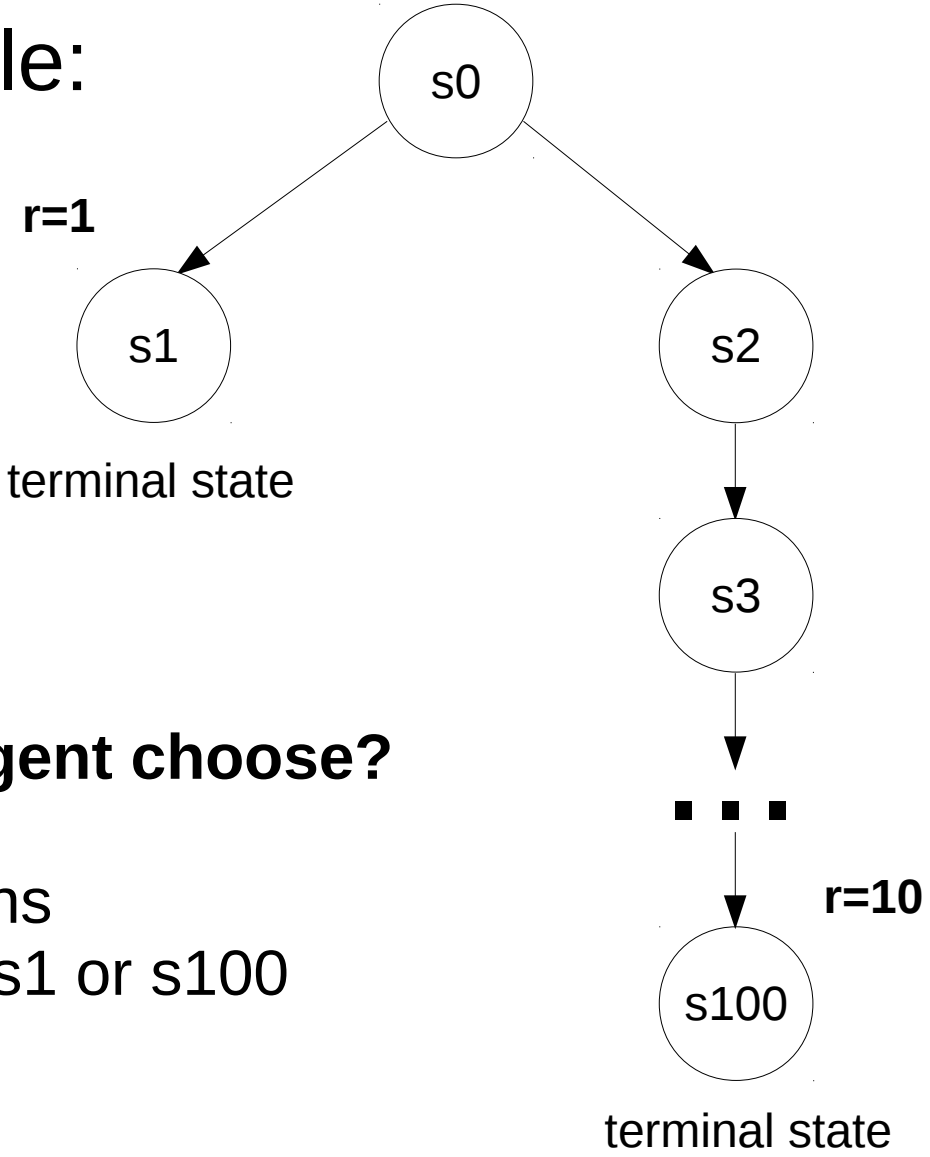
How to learn from sampled trajectories?

How to learn from other agent's experience?

Remember discounted rewards?

Discounted reward fails #1

Trivial example:

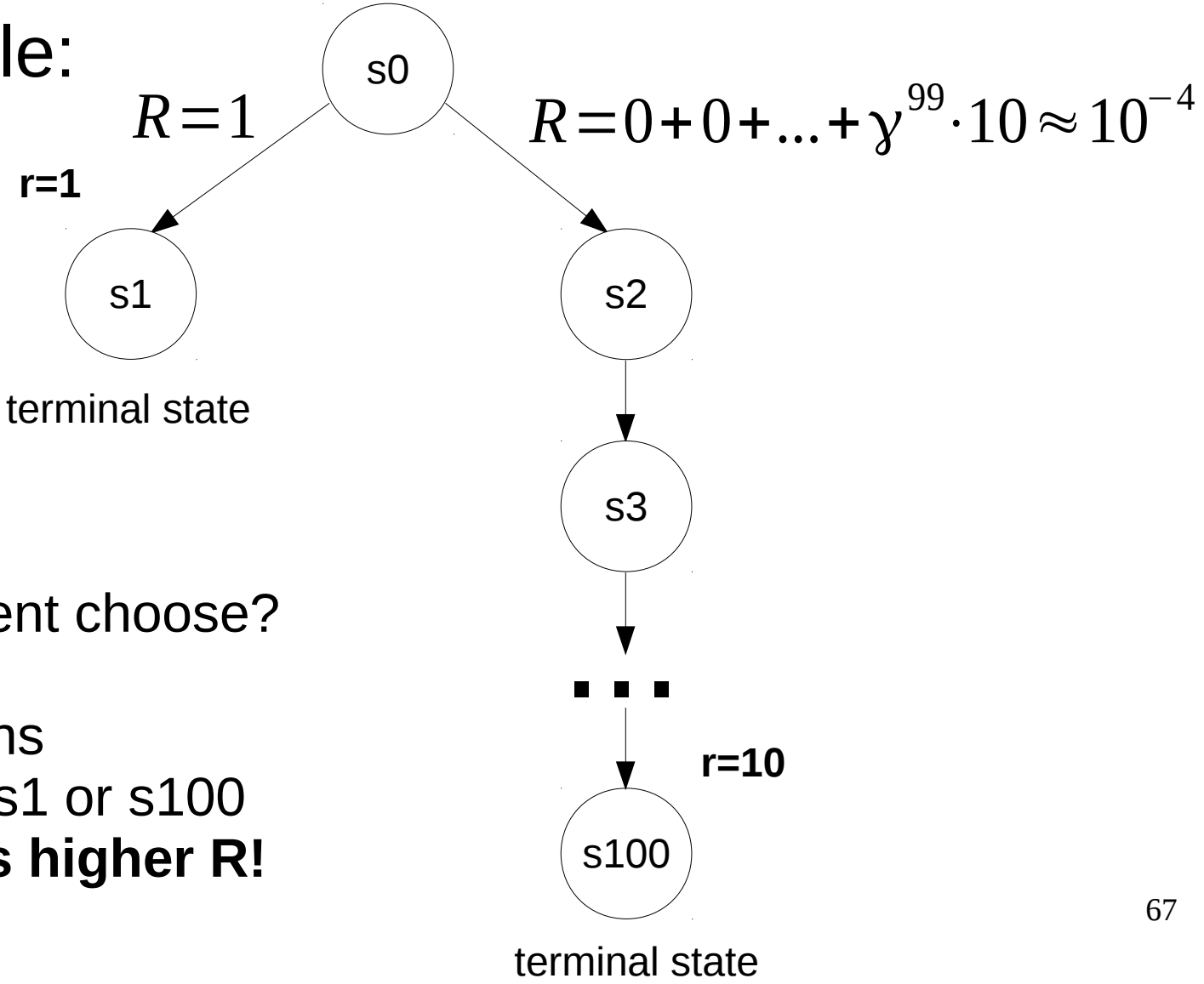


What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100

Discounted reward fails #1

Trivial example:



What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100
- **left action has higher R!**

Discounted reward fails #2

Deephack'17 qualification round, Atari Skiing

- You steer the red guy
- Session lasts ~5k steps

- You get -3~-7 reward each tick
  (faster game = better score)

- At the end of session, you get up to r=-30k
  (based on passing gates, etc.)

- Q-learning with gamma=0.99 fails
  it doesn't learn to pass gates

What's the problem?

http://rl.deephack.me/

Discounted reward fails #2

Deephack'17 qualification round, Atari Skiing

- You steer the red guy
- Session lasts ~5k steps

- You get -3~-7 reward each tick
  (faster game = better score)

- At the end of session, you get up to r=-30k
  (based on passing gates, etc.)

  - Q-learning with gamma=0.99 fails

http://rl.deephack.me/

Discounted reward fails #3

CoastRunner7 experiment (openAI)

- You control the boat
- Rewards for getting to checkpoints
- Rewards for collecting bonuses
- What could possibly go wrong?
- “Optimal” policy video:
  https://www.youtube.com/watch?v=tlOIHko8ySg

https://openai.com/blog/faulty-reward-functions/

Nuts and bolts: MC vs TD

Monte-carlo

- Ignores intermediate rewards
  doesn't need **γ** (discount)
- Needs full episode to learn
  Infinite MDP are a problem
- Doesn't use Markov property
  Works with non-markov envs

Temporal Difference

- Uses intermediate rewards
  trains faster under right **γ**
- Learns from incomplete episode
  Works with infinite MDP
- Requires markov property
  Non-markov env is a problem

Nuts and bolts: discount

- Effective horizon $1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

Typical values:
- γ=0.9, 10 turns
- γ=0.95, 20 turns
- γ=0.99, 100 turns
- γ=1, infinitely long

Higher γ = less stable algorithm.
γ=1 only works for episodic MDP (finite amount of turns).

Nuts and bolts: discount

- Effective horizon $1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$

  Heuristic: your agent stops giving a damn in *this many* turns.

- Atari Skiing, reward was delayed by in 5k steps
- γ=0.99 is not enough
- γ=1 and a few hacks works better
- Or use a better reward function

Let's write some code!