

Developments in Model-Based Reinforcement Learning

Alan Fern & Anurag Koul

Oregon State University

ICAPS Online Summer School 2020

Outline

Intro to Model-Based Reinforcement Learning (MBRL)

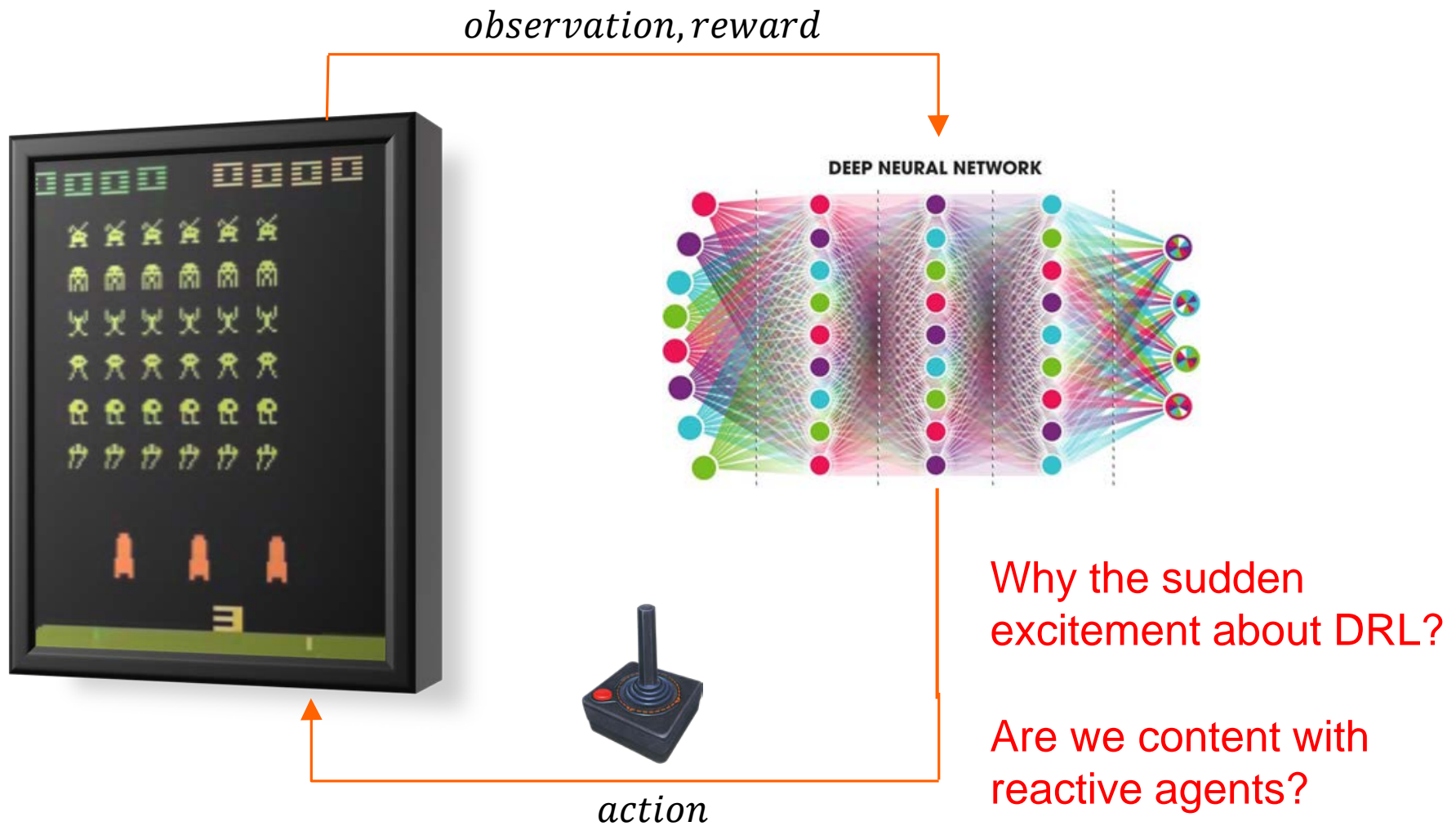
- Brief Introduction to Reinforcement Learning
- MBRL Choices – Types of Models
- MBRL Choices – Types of Planners

Class of Algorithms

- Class 1: Tabular Models with Optimal Planning
- Class 2: Simulation Models with Search-Based Planning
- Class 3: Simulation Models for Model-Free “Planning”

Research Directions from Planning Perspective

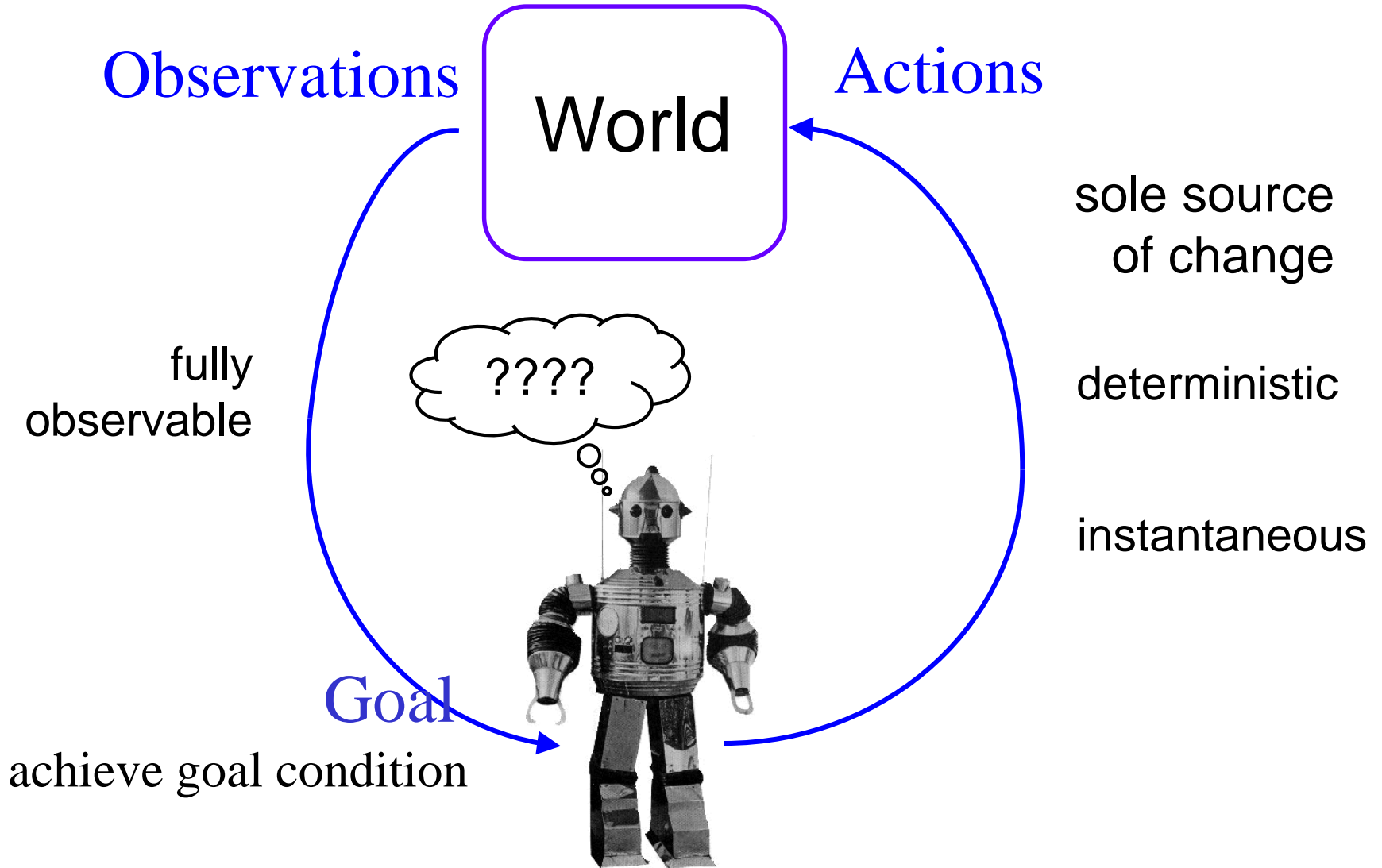
Deep Reinforcement Learning (DRL)



Objective : learn to take action to maximize cumulative future reward

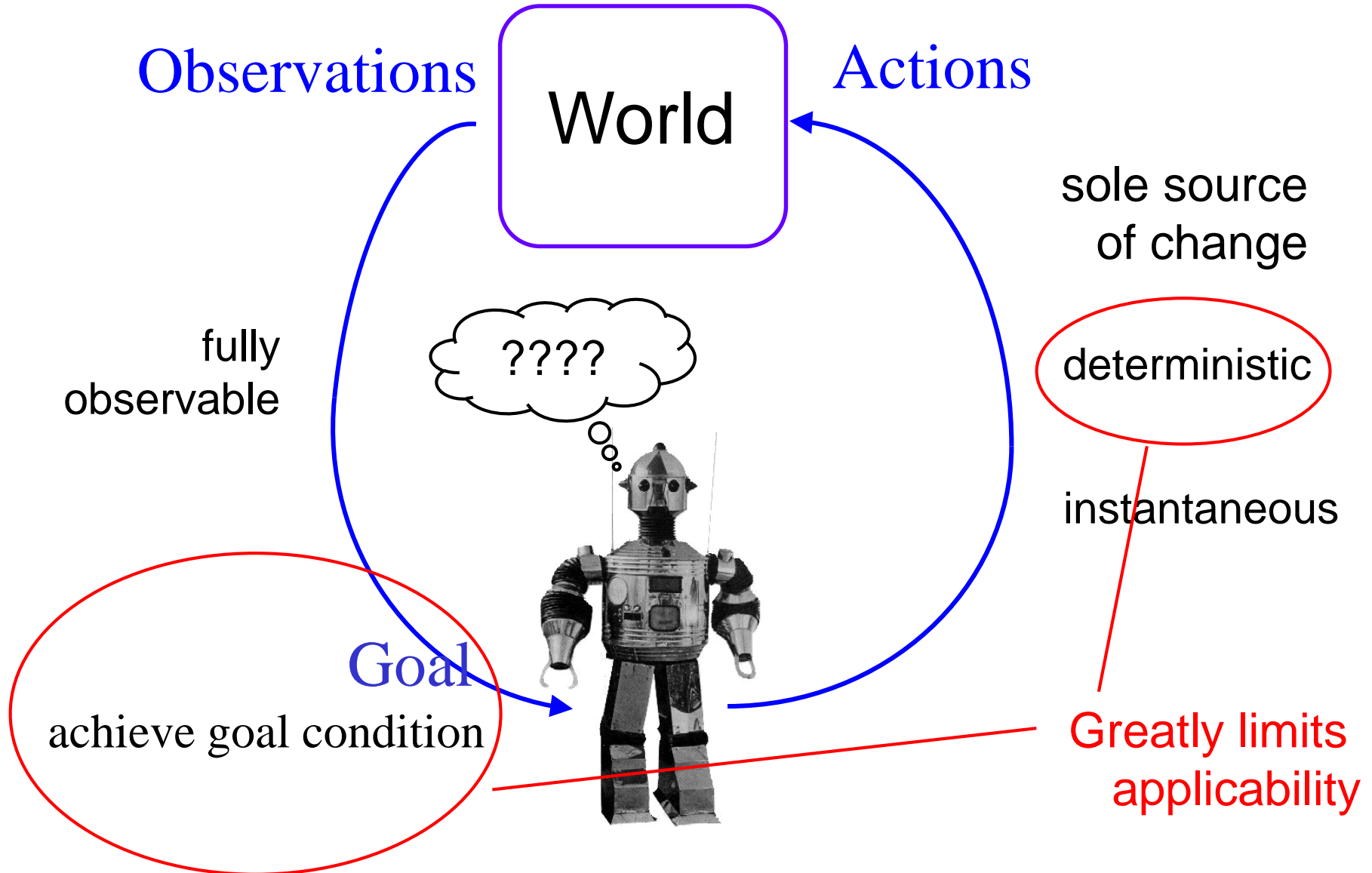
Classical Planning Assumptions

(primary focus of AI planning until early 90's)

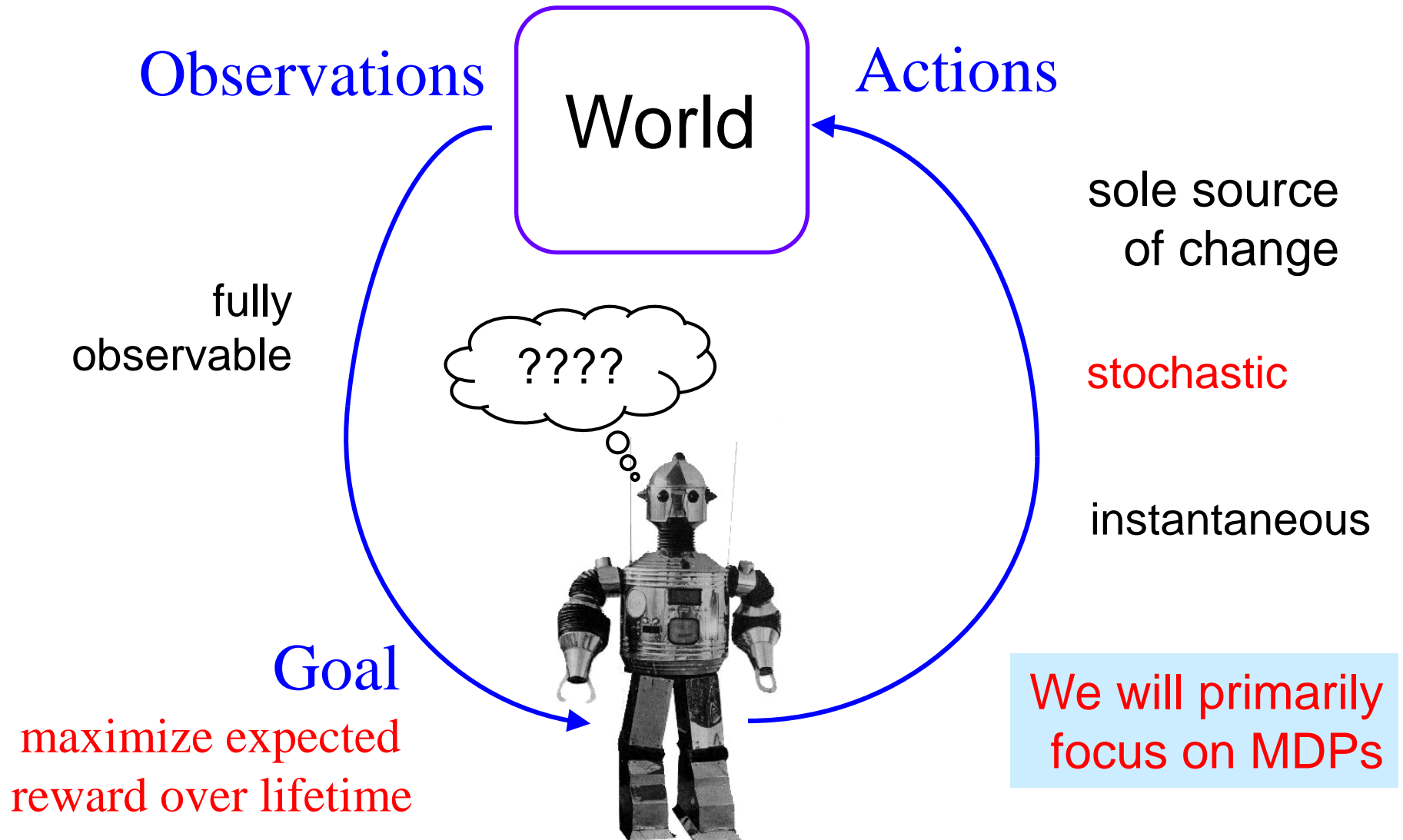


Classical Planning Assumptions

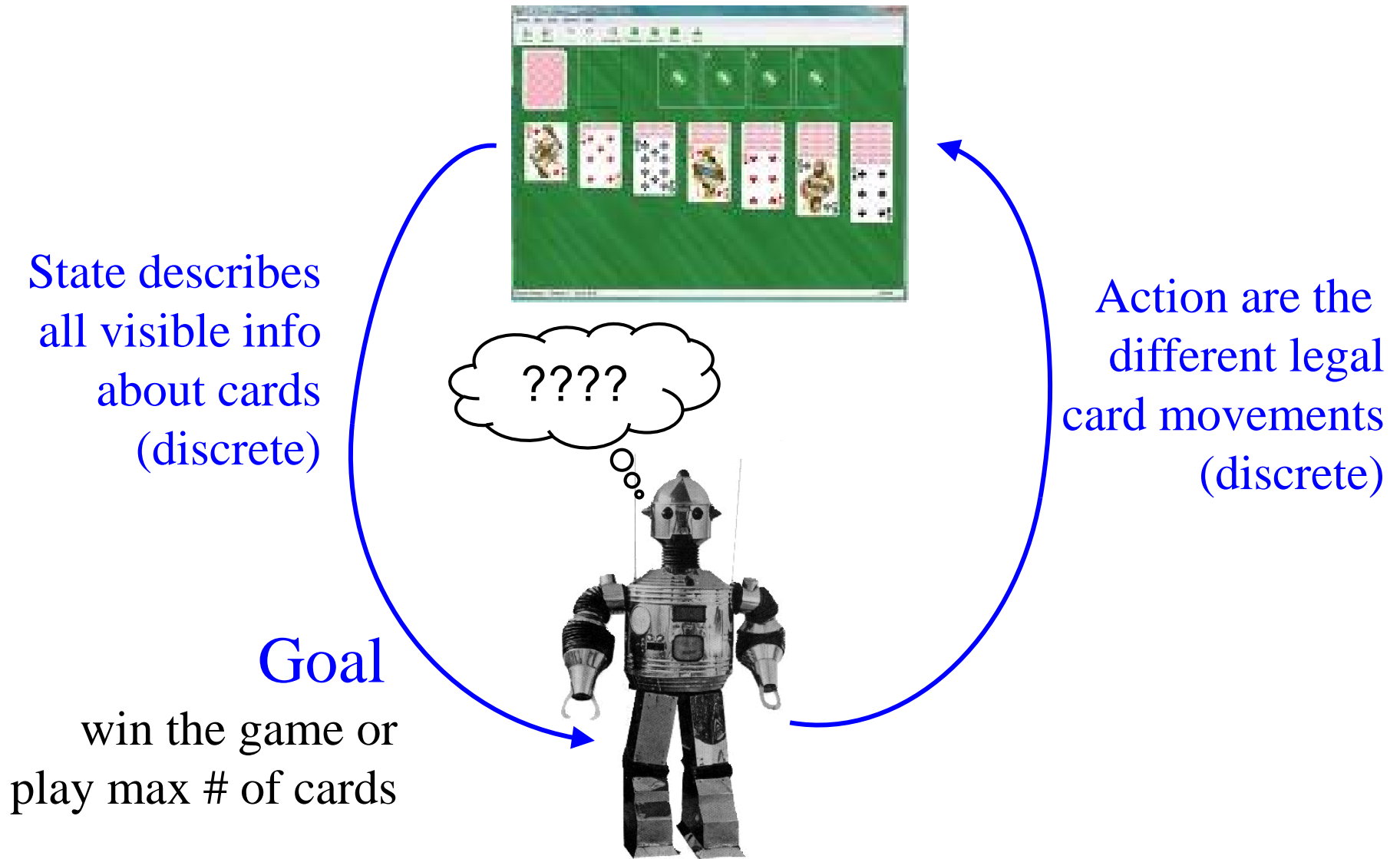
(primary focus of AI planning until mid 90's)



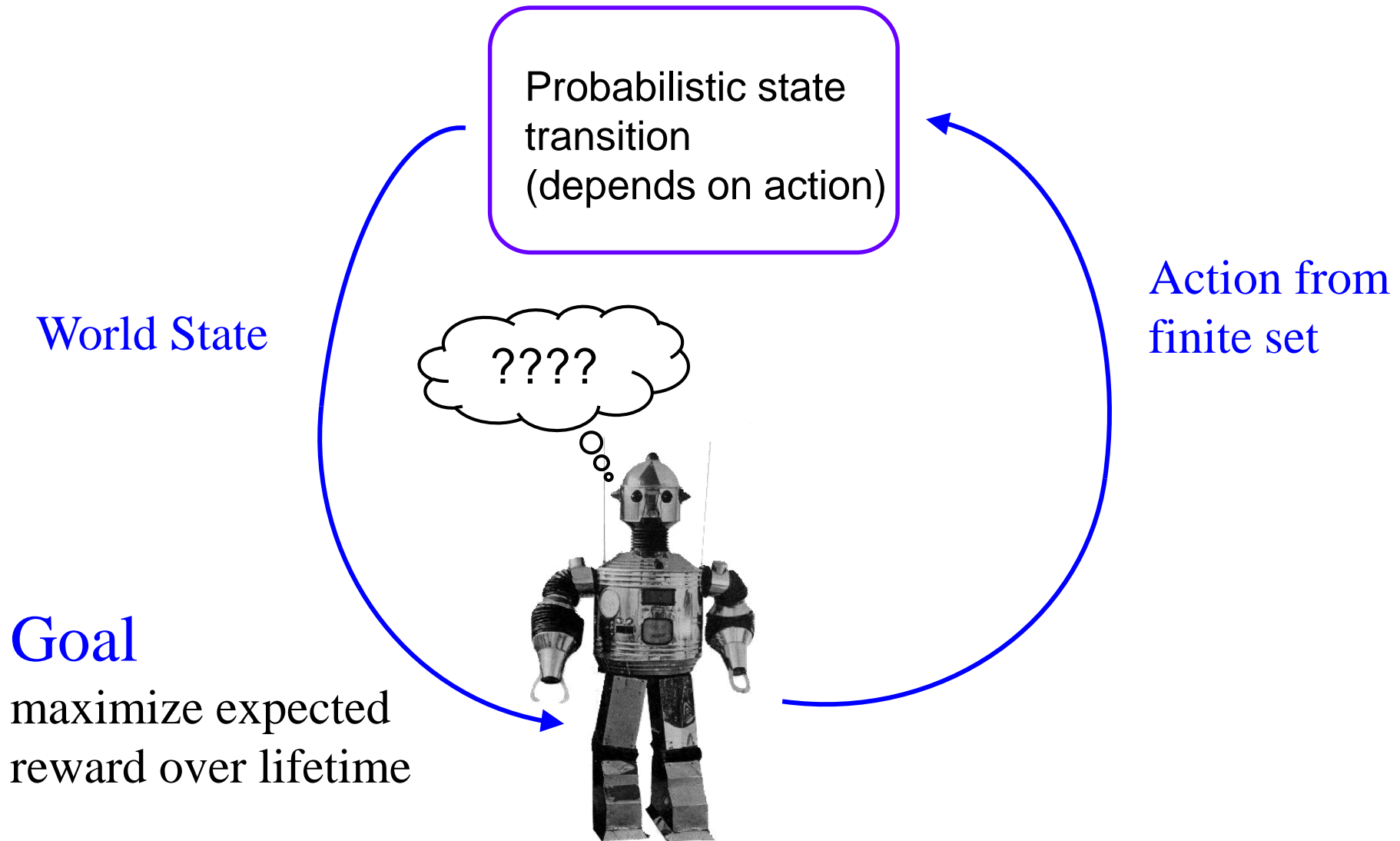
Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model



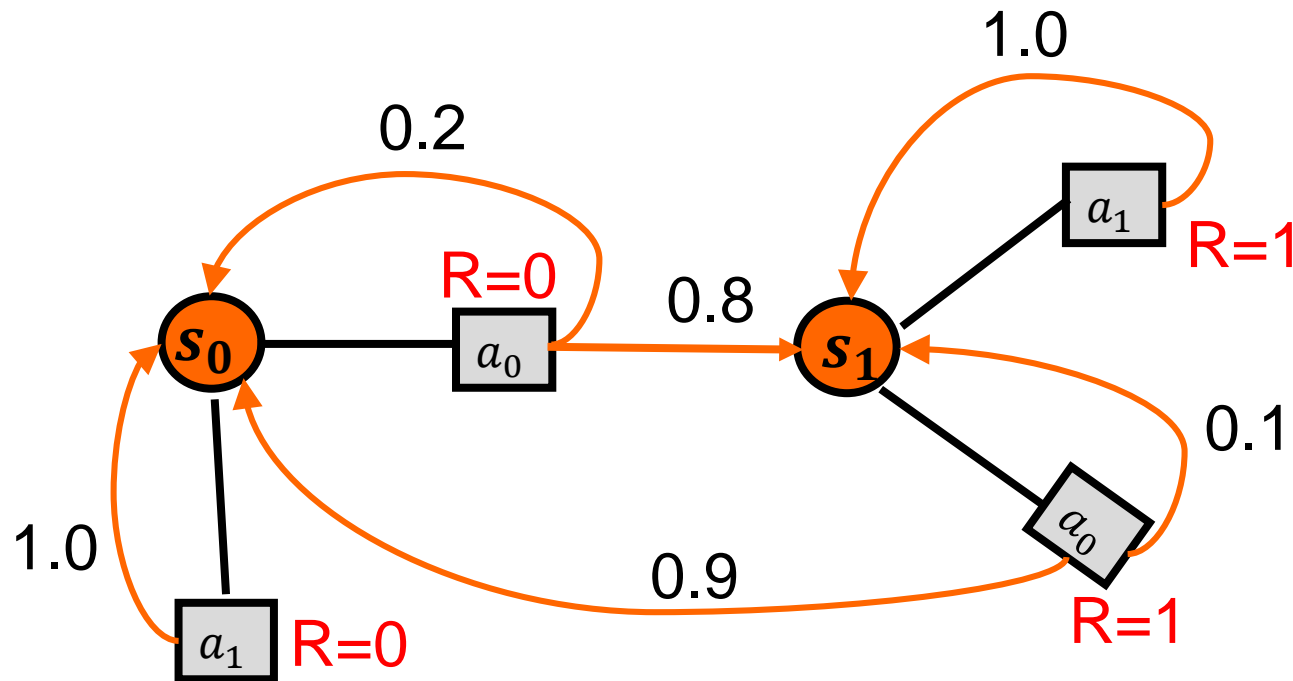
Example MDP



Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model



Markov Decision Processes



$$T(s_0, a_0, s_0) = \Pr(s_0 | s_0, a_0) = 0.2$$

$$T(s_0, a_0, s_1) = \Pr(s_1 | s_0, a_0) = 0.8$$

$$T(s_0, a_1, s_0) = \Pr(s_0 | s_0, a_1) = 1.0$$

$$T(s_0, a_1, s_1) = \Pr(s_1 | s_0, a_1) = 0.0$$

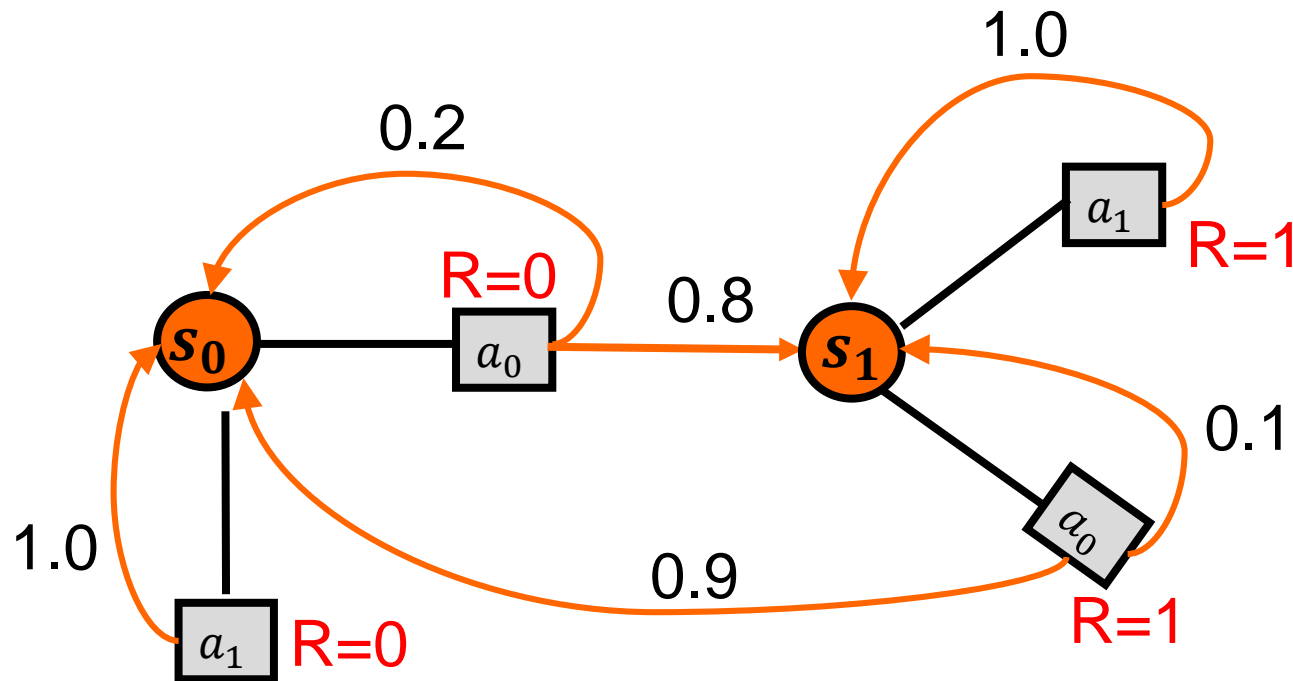
$$T(s_1, a_0, s_0) = \Pr(s_0 | s_1, a_0) = 0.9$$

$$R(s_0, a_0) = R(s_0, a_1) = 0$$

$$R(s_1, a_0) = R(s_1, a_1) = 1$$

....

Tabular Representation



N states and M actions

Transition Function $T(s, a, s')$ stored as one $N \times N$ table for each action

$O(MN^2)$ space

Reward Function stored as $O(MN)$ sized table

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)

Output: ????

What is a solution to an MDP?

MDP Planning Problem:

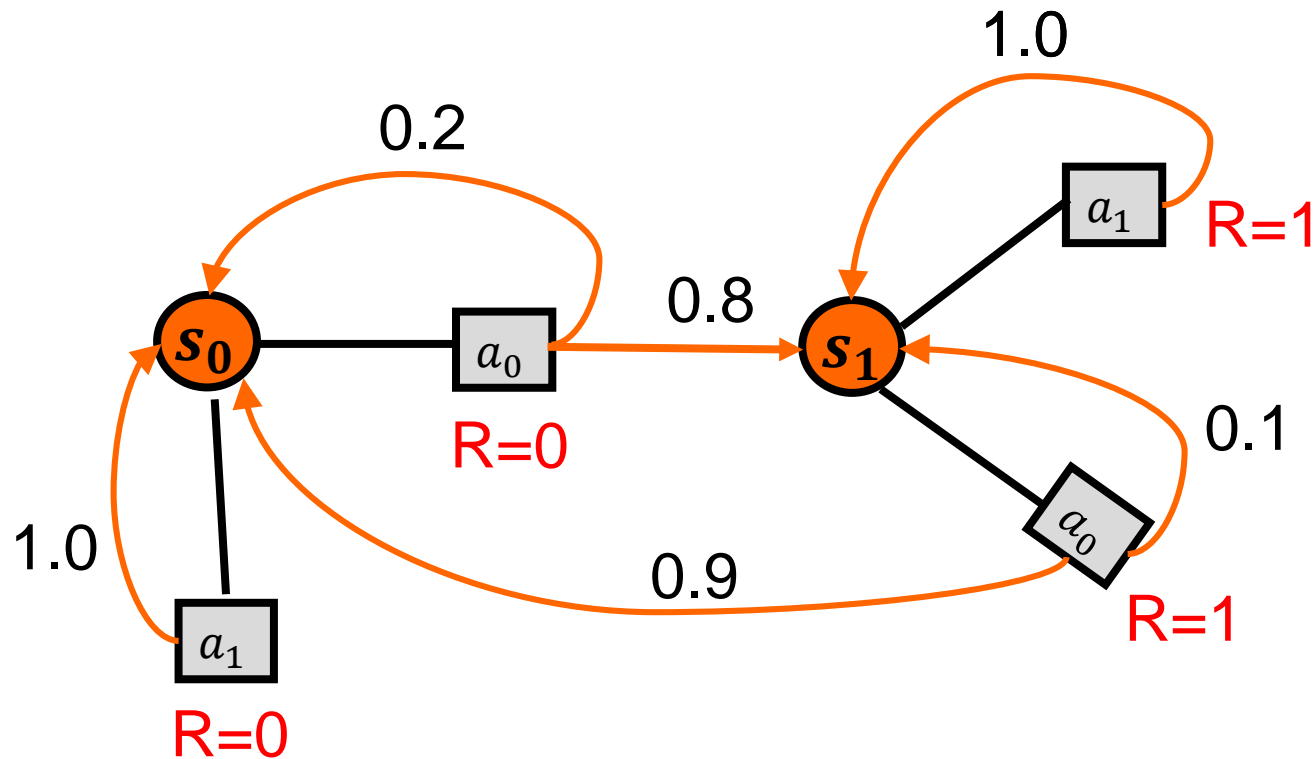
Input: an MDP (S,A,R,T)

Output: ????

- **One Answer:** Suppose we are given an initial starting state s_0 .
 - ▲ An **open-loop plan** from s_0 is a sequence of actions (a_1, a_2, a_0, \dots) that will be executed by the agent
- Should the solution to an MDP be open-loop plan in general?
 - ▲ Consider a single player card game like Blackjack/Solitaire.

No. When there is randomness we need to open our eyes to see where we end up to select good actions.

Stationary Policies

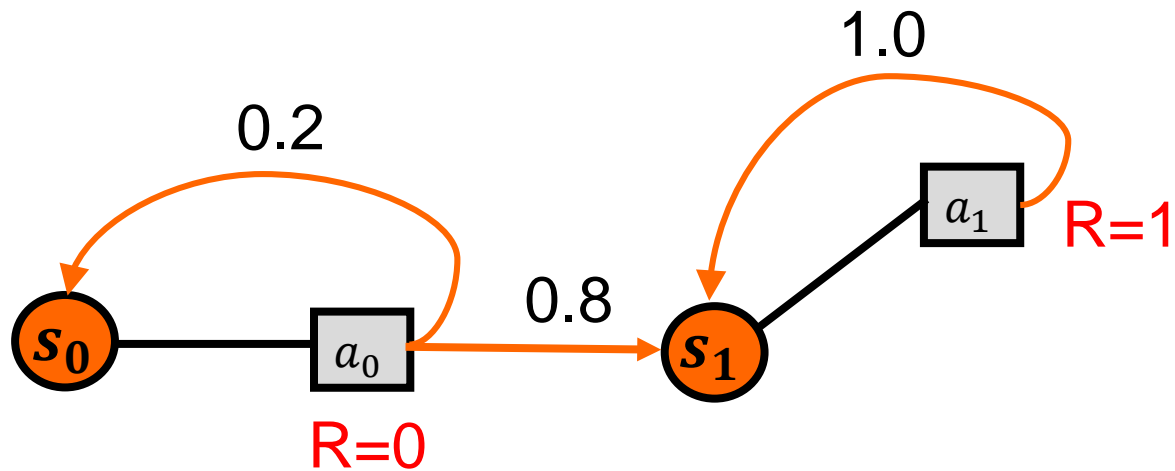


A **stationary policy** π selects an action for each state.

$$\pi : S \rightarrow A$$

Can view as a sub-graph of the MDP.

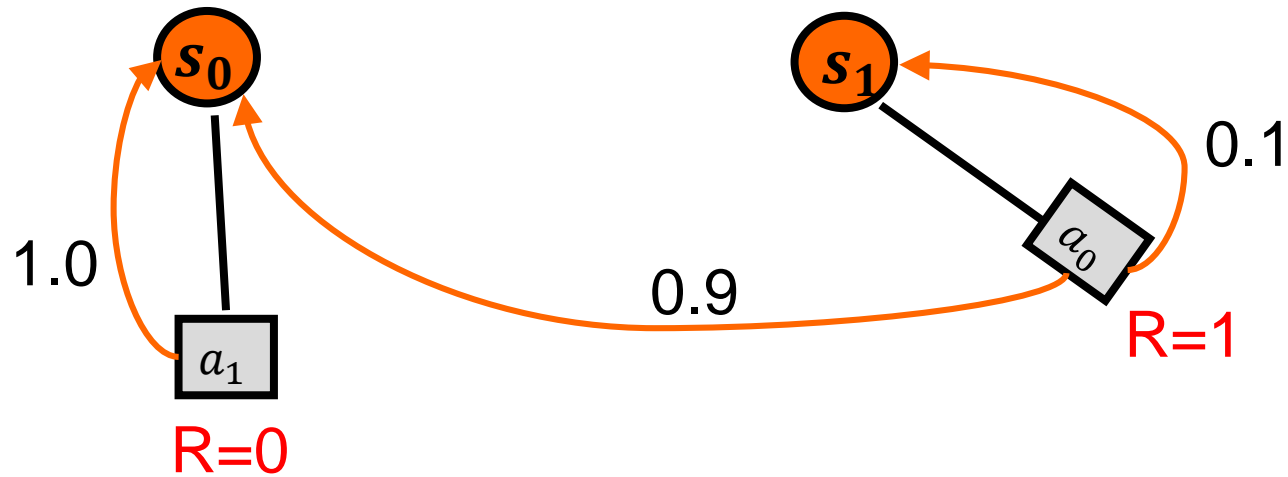
Stationary Policies



Example policy #1

$$\pi(s_0) = a_0 \quad \pi(s_1) = a_1$$

Stationary Policies



Example policy #2

$$\pi(s_0) = a_1 \quad \pi(s_1) = a_0$$

What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S, A, R, T)

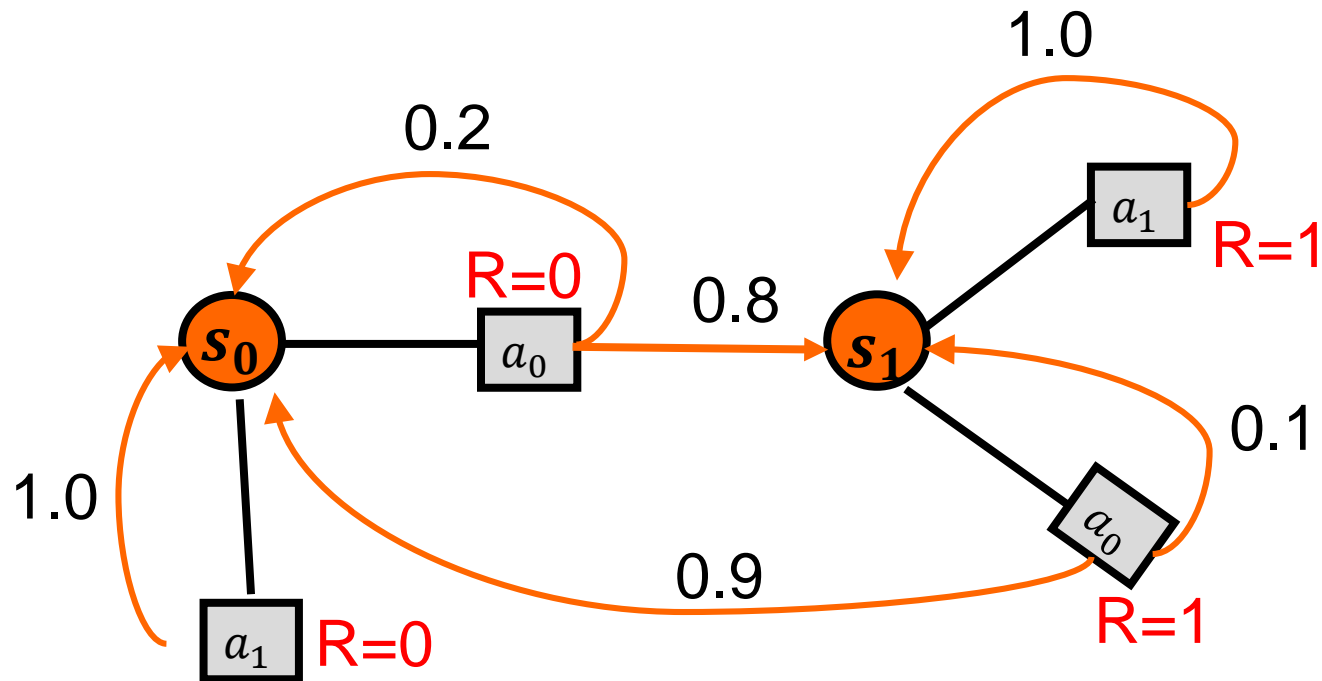
Output: a policy such that ????

- We don't want to output just any policy
- We want to output a “good” policy
- One that accumulates a lot of reward

Value of a Policy

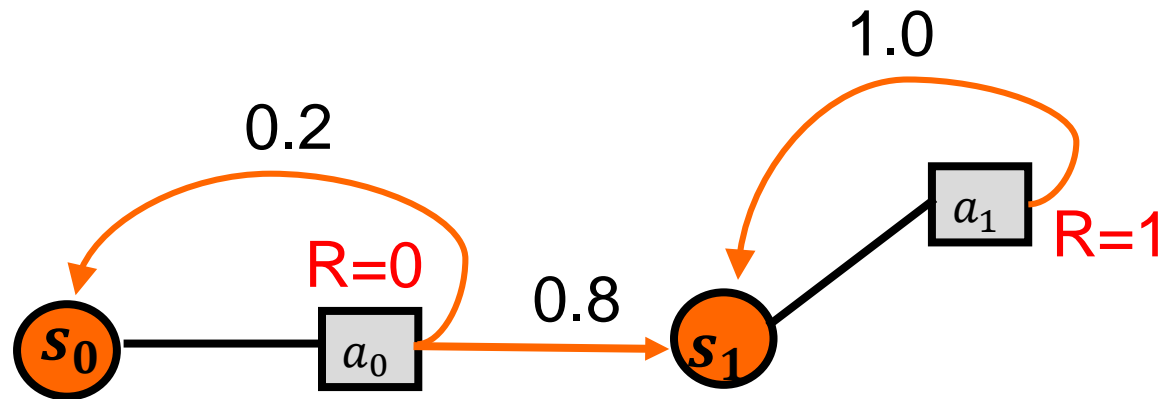
- How good is a policy π ?
 - ▲ How do we measure reward “accumulated” by π ?
- **Value function** $V: S \rightarrow \mathbb{R}$ associates value with each state (or each state and time for non-stationary π)
- $V_\pi(s)$ denotes **value** of policy π at state s
 - ▲ Depends on immediate reward, but also what you achieve subsequently by following π
 - ▲ An **optimal policy** is one that is no worse than any other policy at any state
- The goal of MDP planning is to compute an optimal policy

Infinite Horizon Value



Consider accumulating reward over an infinite horizon?

Infinite Horizon



Example policy: $\pi(s_0) = a_0$, $\pi(s_1) = a_1$

Do we have any problems here for infinite horizon?

Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons $r_0 + r_1 + r_2 + r_3 + \dots$
 - ▶ many or all policies have infinite expected reward
 - ▶ some MDPs are ok (e.g., zero-cost absorbing states)
- Why is this bad?
 - ▶ Consider π_1 that gets $R=1$ per step and π_2 that gets $R=2$ per step
 - ▶ π_2 is clearly better, but infinite total reward can't distinguish between them (both get infinite value)
- “Trick”: introduce discount factor $0 \leq \beta < 1$
 - ▶ future rewards discounted by β per time step
 - ▶ $r_0 + \beta r_1 + \beta^2 r_2 + \beta^3 r_3 + \dots$


Discounted Infinite Horizon MDPs

- Expected infinite horizon discounted reward

$$V_{\pi}(s) = E \left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s \right]$$

- We avoid infinite values (consider getting max absolute reward each step)

Maximum absolute reward


$$V_{\pi}(s) \leq E \left[\sum_{t=0}^{\infty} \beta^t R^{\max} \right] = \frac{1}{1-\beta} R^{\max}$$

21


- Motivation: economic? prob of death? convenience?

Notes: Discounted Infinite Horizon

- Optimal policies guaranteed to exist (Howard, 1960)
 - ▲ I.e. there is a policy that maximizes value at each state
- Furthermore there is always an optimal stationary policy
 - ▲ Intuition: why would we change action at s at a new time when there is always forever ahead
- We define $V^*(s)$ to be the optimal value function.
 - ▲ That is, $V^*(s) = V_{\pi}(s)$ for some optimal stationary π

Computing an Optimal Value Function

- **Bellman equation** for optimal value function

$$V^*(s) = \max_a R(s, a) + \beta \sum_{s'} T(s, a, s') \cdot V^*(s')$$


immediate reward

discounted expected value
of best action assuming we
we get optimal value in future

- Bellman proved this is always true for an optimal value function

Computing an Optimal Value Function

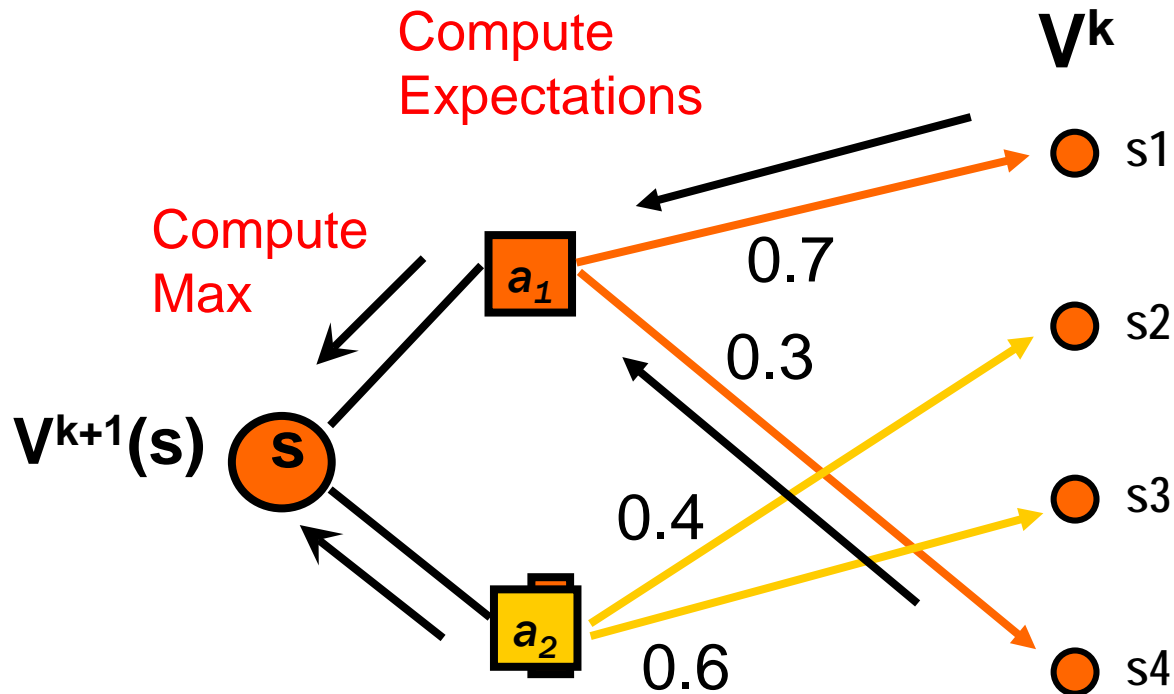
- **Bellman equation** for optimal value function

$$V^*(s) = \max_a R(s, a) + \beta \sum_{s'} T(s, a, s') \cdot V^*(s')$$

- How can we solve this equation for V^* ?
 - ▶ The MAX operator makes the system non-linear
- There are several classic optimal algorithms for tabular representations
 - ▶ Value Iteration
 - ▶ Policy Iteration
 - ▶ Reduction to Linear Programming

Value Iteration

Computes a sequence of value functions using Bellman Backup Operator



$$V^{k+1}(s) = \max_a R(s, a) + \beta \sum_{s'} T(s, a, s') \cdot V^k(s')$$

Value Iteration

$$V^0(s) = 0$$

$$V^k(s) = \max_a R(s, a) + \beta \sum_{s'} T(s, a, s') \cdot V^{k-1}(s')$$

- Each iteration requires $O(MN^2)$ time.
- Converges to optimal value function

$$\lim_{k \rightarrow \infty} V^k = V^*$$

How to Act

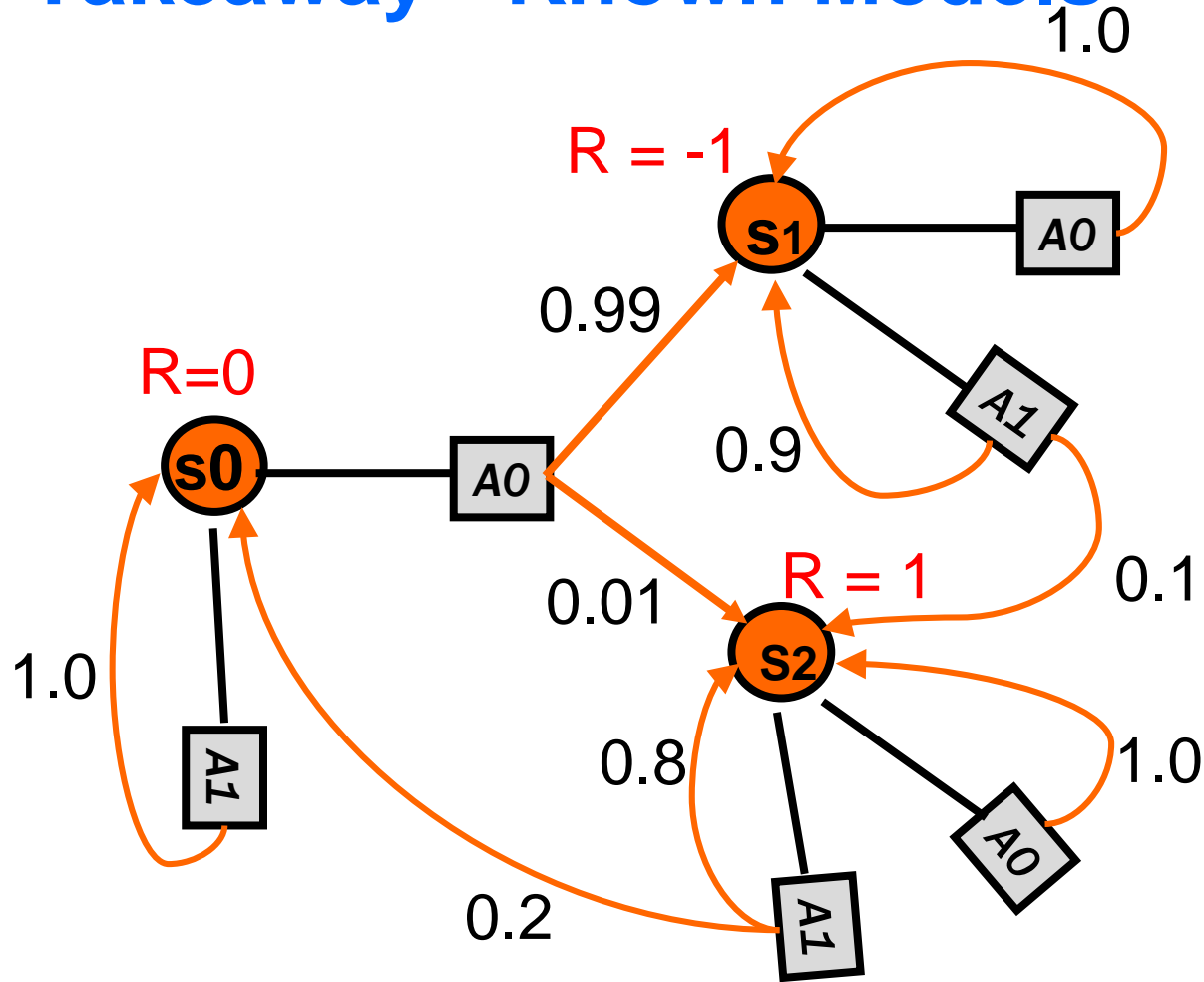
- Given a V from value iteration that closely approximates V^* , what should we use as our policy?
- Use V to define the Q-value function over states and actions (one step look ahead)

$$Q(s, a) = R(s, a) + \beta \sum_{s'} T(s, a, s') \cdot V(s')$$

- Use *greedy* policy: (max action value)

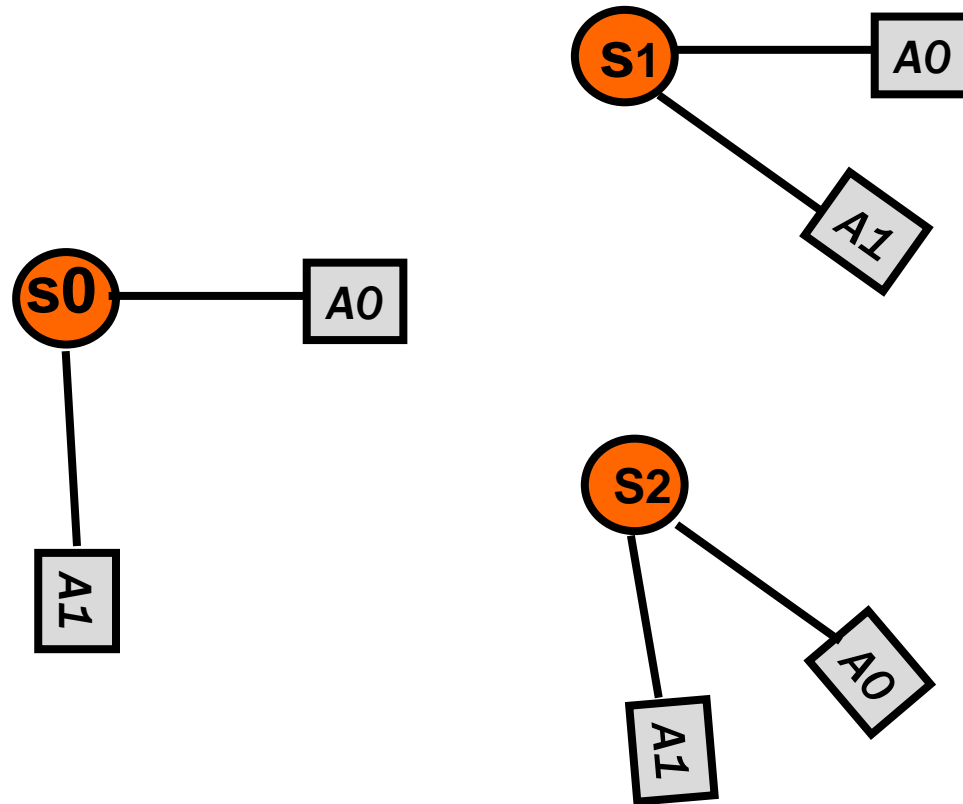
$$\pi(s) = \arg \max_a Q(s, a)$$

Main Takeaway - Known Models



Given a **moderately-sized** MDP model (up to millions of states), we can compute optimal policies.

Unknown Models



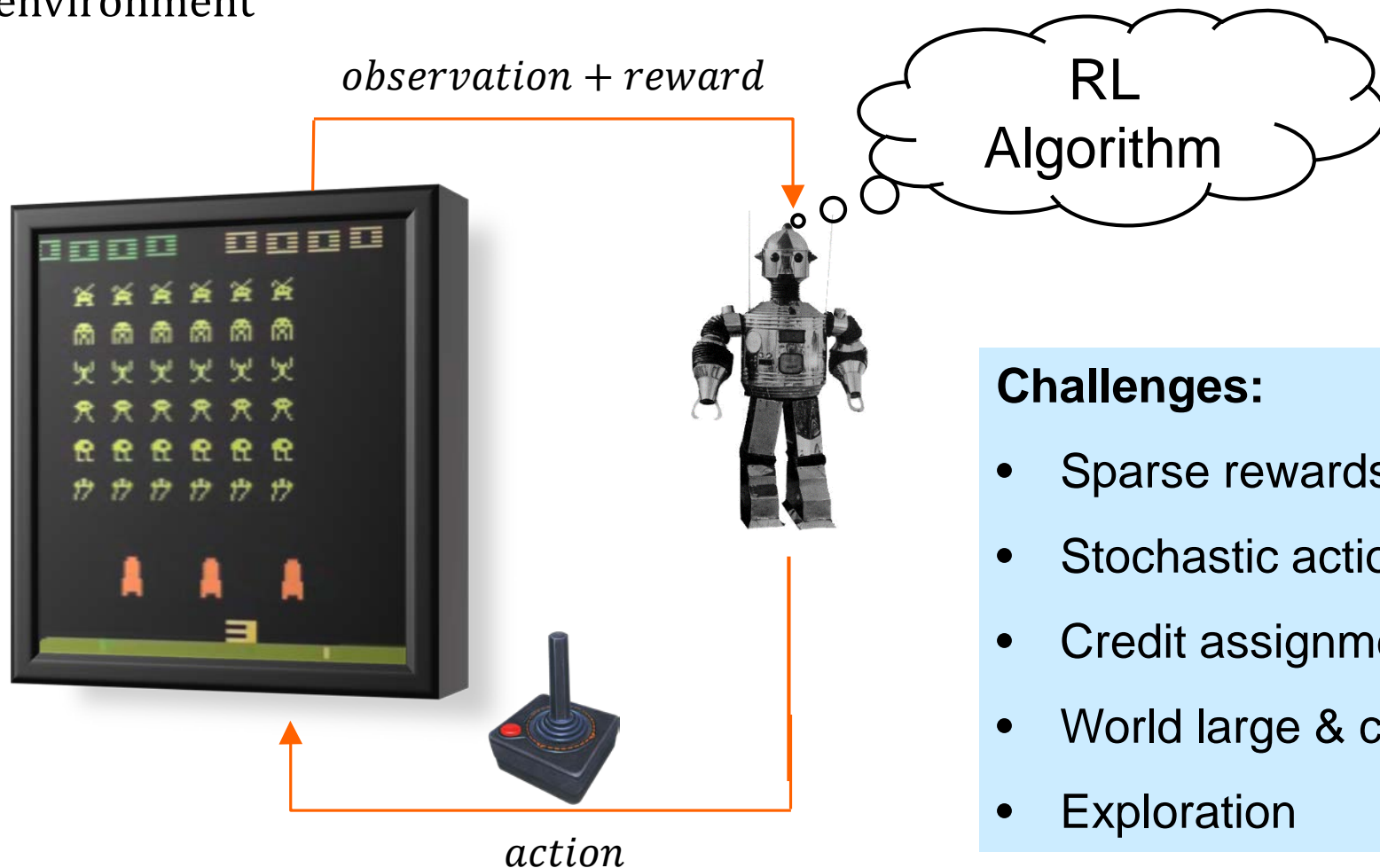
What if we don't know the reward and transition functions?
(Like in many real-world domains.)

But we can take actions and observe their effects.

Reinforcement Learning

Agent starts with no knowledge of environment.

Objective: learn optimal (or good enough) policy through interaction with environment



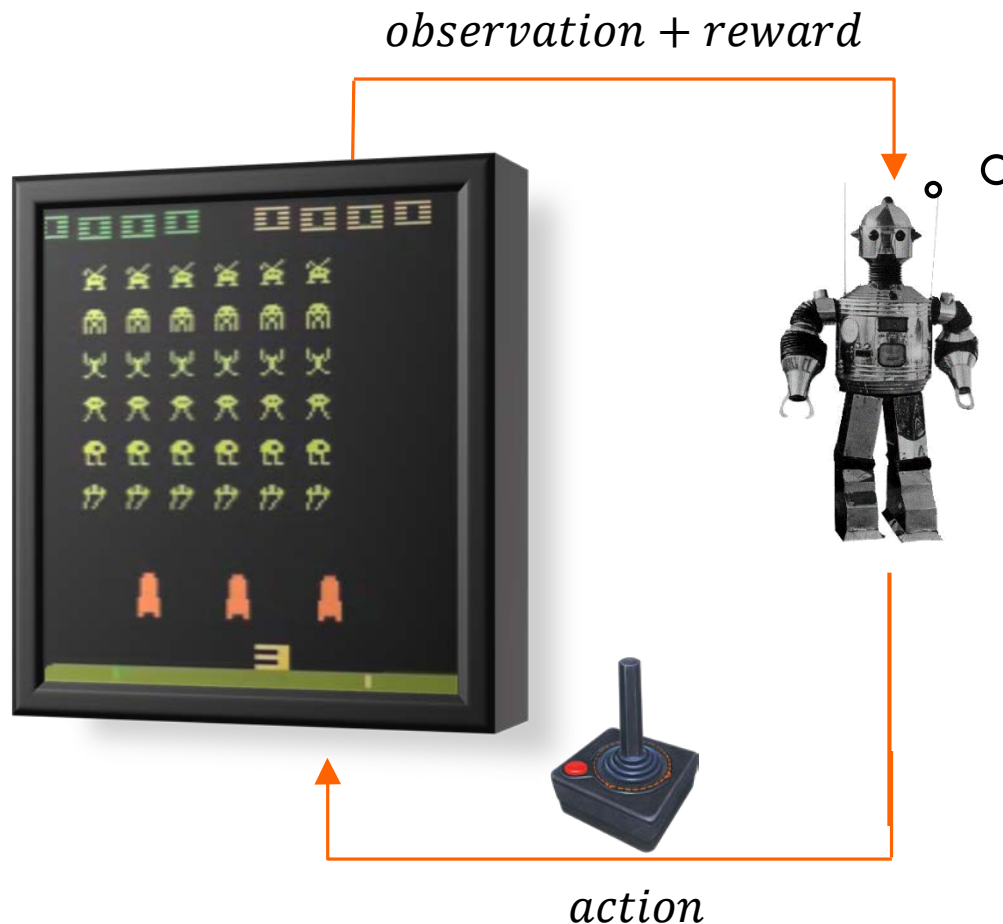
Challenges:

- Sparse rewards
- Stochastic actions
- Credit assignment
- World large & complex
- Exploration

Reinforcement Learning

Agent starts with no knowledge of environment.

Objective: learn optimal (or good enough) policy through interaction with environment



Model-Free vs.
Model-Based

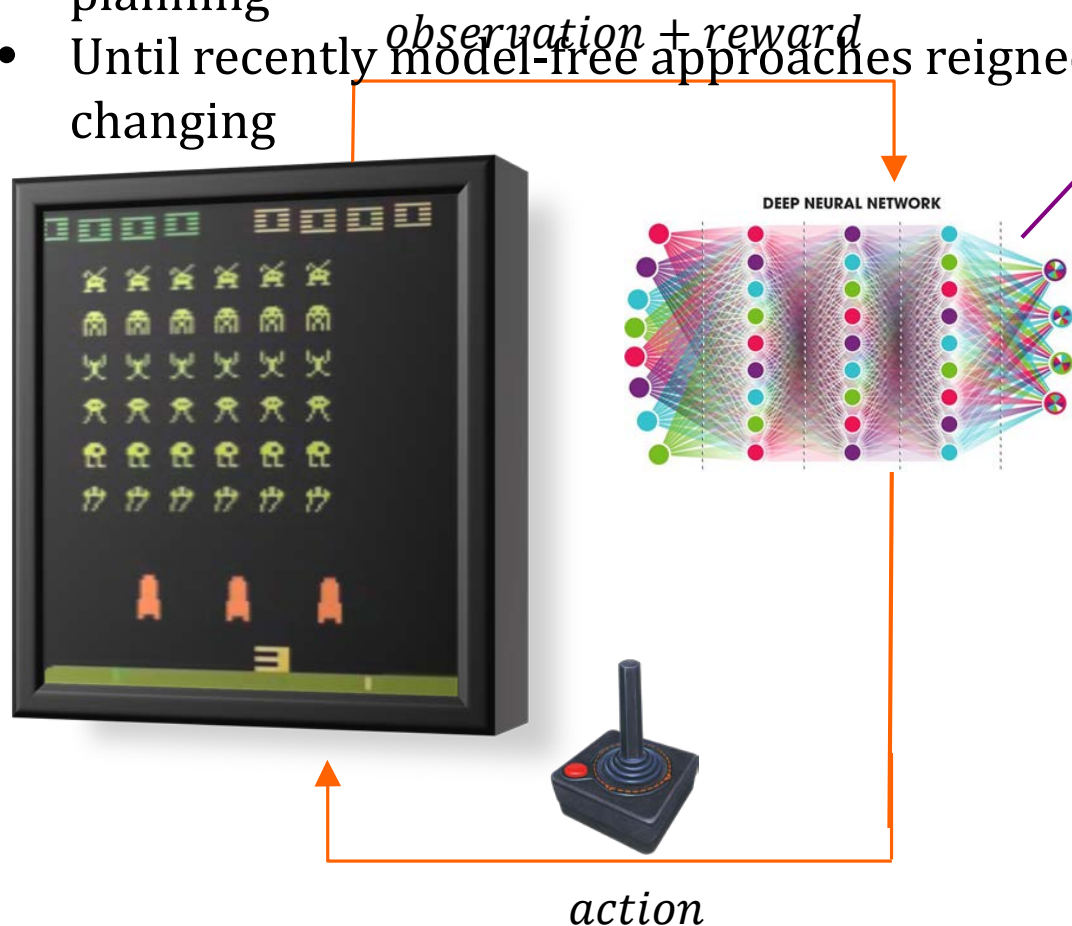
Challenges:

- Sparse rewards
- Stochastic actions
- Credit assignment
- World large & complex
- Exploration

Model-Free Reinforcement Learning

Model free approach to RL:

- Directly learn a value function or policy without explicitly learning a model
- Useful when model is difficult to represent or learn, or too large for planning
- Until recently model-free approaches reigned supreme in practice – that is changing



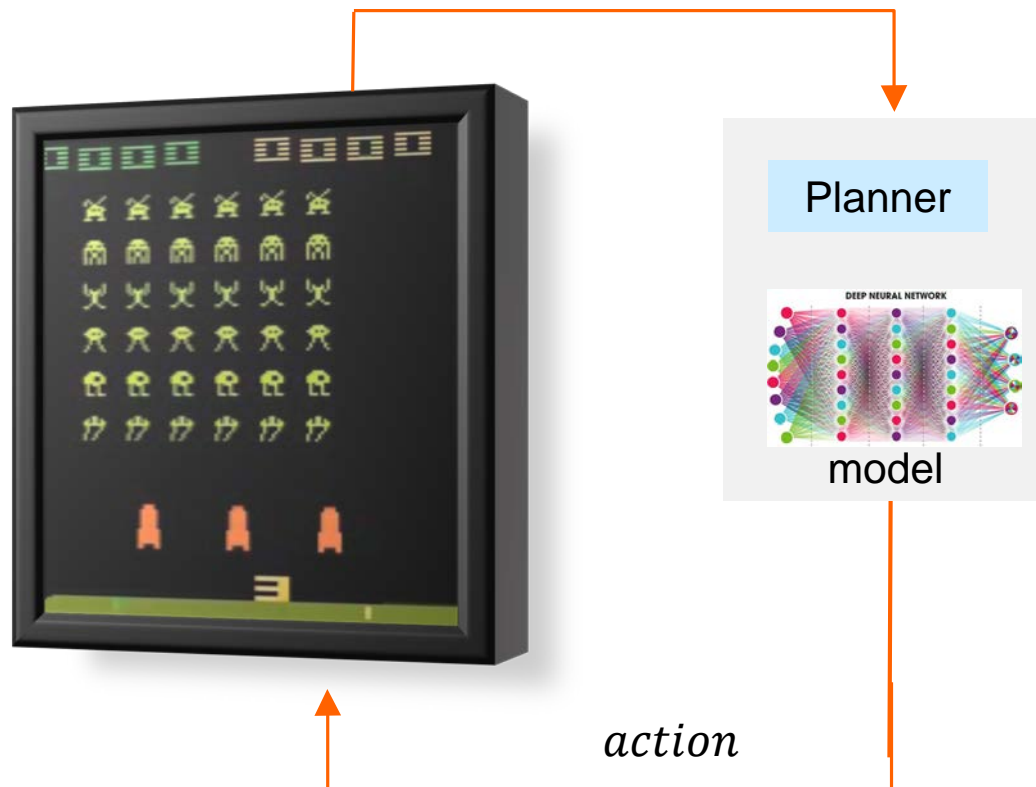
For this lecture just know:

- Many model-free algorithms -- often effective given “enough” experience
- “enough” in practice can be 10^6 to 10^7 transitions!
- Q-learning, PolicyGradient, Actor-Critic (e.g. DQN, PPO, DDPG, TD3,...)

Model-Based Reinforcement Learning

Model-based approach to RL:

- continually learn an (approximate) MDP model from experience
- use model to somehow improve “experience efficiency”
- Intuition: fitting a model to experience is supervised learning– easier than general RL
- Intuition: can leverage “planning” if we have a model
observation + reward



MBRL Approaches differ in:

- The **type of model** learned
- The **type of planning** done
- Some methods don't do anything that looks like planning
(e.g. model-learning is just “auxillary learning task” for model-free learning)
- We focus on those w/ planning

Models: to use or not to use

- **Potential Advantages**

- ▶ use to “imagine” arbitrary amounts and types of experience
- ▶ leverage planning algorithms to identify good actions (either for exploration or performance)
- ▶ learned dynamics can be independent of specific tasks (i.e. reward functions) so generalize across tasks

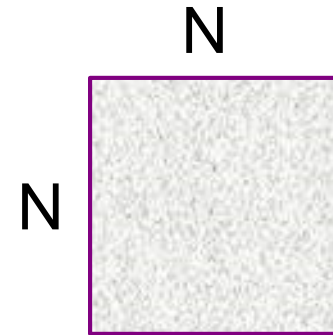
- **Potential Disadvantages**

- ▶ if model is inaccurate, performance could be hurt (optimize for the wrong world)
- ▶ planning can be computationally expensive (trade-off: computational vs. experience efficiency)
- ▶ for very complex environments a model may be harder to learn than the policy or value function

Model Choices: Representation

- **Tabular Models**

- ▶ much RL theory is in this setting
- ▶ challenges for enormous state-action spaces
- ▶ recent integrations with deep learning!



- **Structured/Symbolic Models**

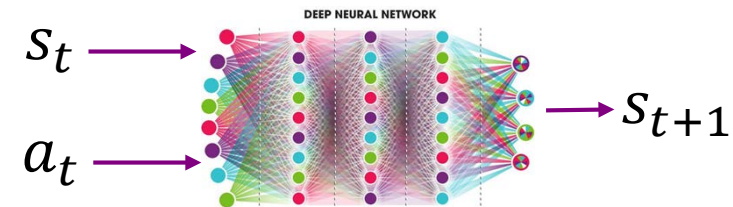
- ▶ learn a model representation in a language that can be “reasoned about” by planner
- ▶ E.g. PDDL/STRIPS, PPDDL, RDDL,
- ▶ What if language isn’t well-matched to world?
- ▶ Where do symbols come from?

PICK-UP(A,B)

PRE: clear(A), on(A,B)
ADD: holding(A), clear(B)
DEL: on(A,B), clear(A)

- **Black Box Simulators**

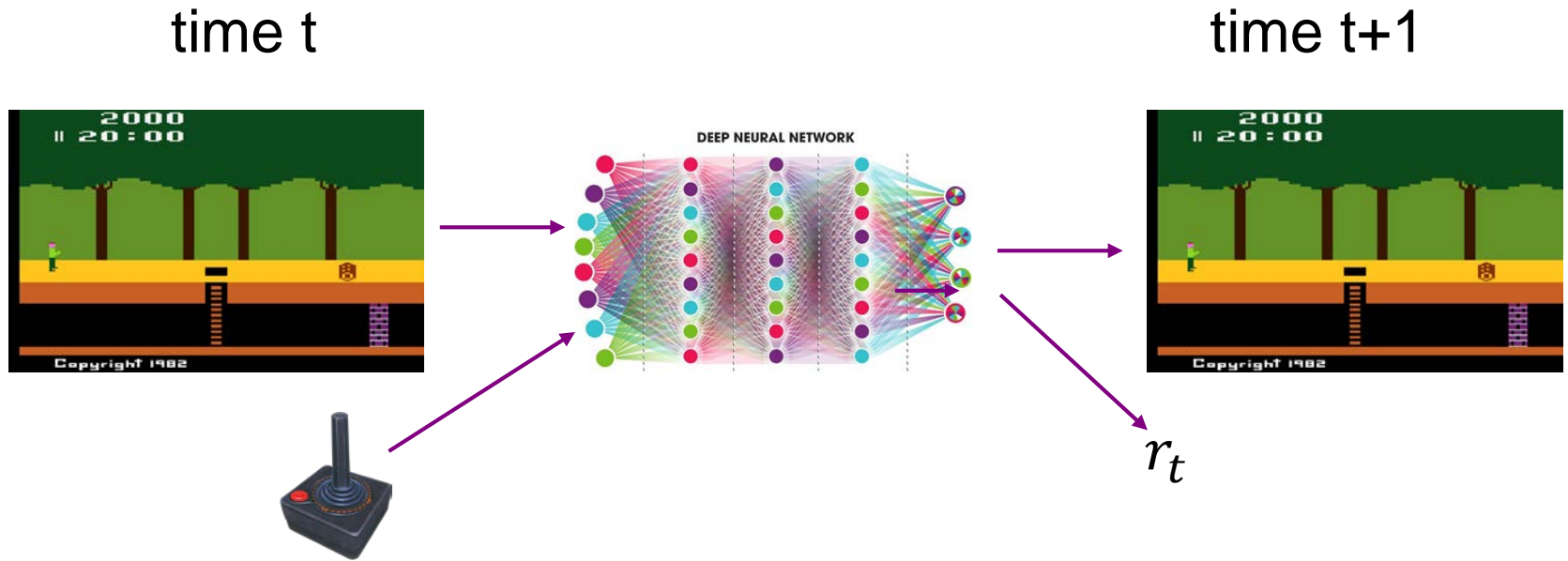
- ▶ learn a black-box function that can simulate transitions and rewards
- ▶ E.g. represent via neural network
- ▶ the most common approach today



Model Choices: Observational vs. Latent

- **Observational Models**

- ▶ maps raw observations and actions to predicted next observation & reward
- ▶ can be difficult with complex observations (e.g. images)
- ▶ many parts of observations are irrelevant to actions (e.g. dynamic backgrounds)

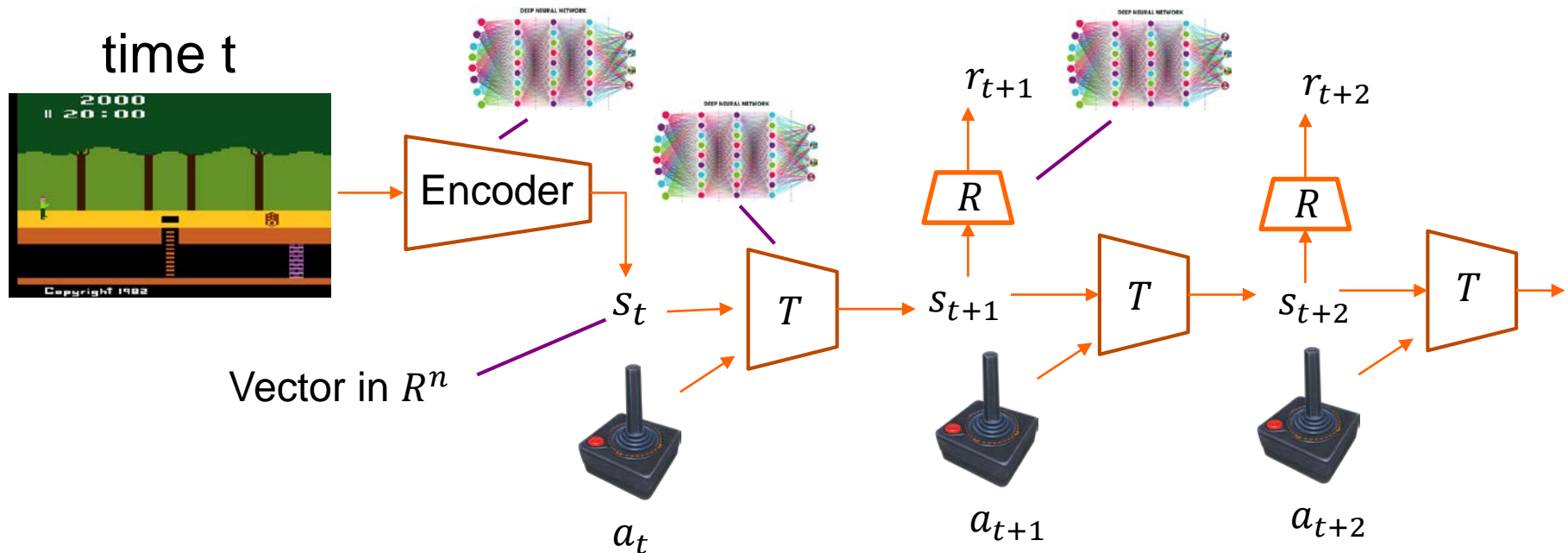


Limited success for MBRL

Model Choices: Observational vs. Latent

- **Latent State Models**

- ▶ Learn to encode raw observations into internal latent state representation
- ▶ Learn dynamics and reward in latent space
- ▶ Latent representation may ignore unimportant details of observations

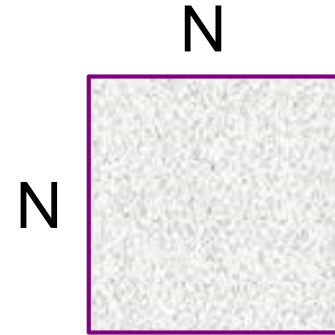


- Train so latent model can accurately predict future rewards
- Use of latent models is one key to recent MBRL successes

MBRL Planner Types

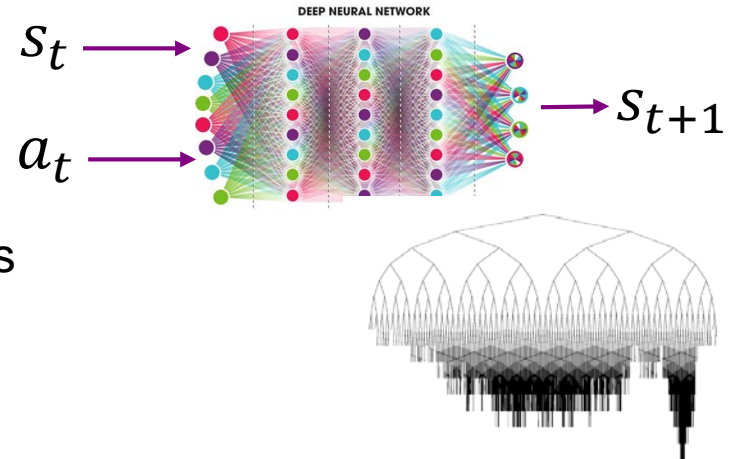
- **(Near) Optimal Tabular Planning**

- ▶ E.g. value iteration
- ▶ basis for much of RL theory
- ▶ recent uses w/ deep latent representations



- **Monte-Carlo Planning**

- ▶ only requires black-box simulator
- ▶ E.g. Monte-Carlo Tree Search (MCTS)
- ▶ effectiveness can be limited in some cases where RL is hard (e.g. sparse rewards)



- **Symbolic Planning**

- ▶ requires structured/symbolic model representation
- ▶ Sadly, so far there has been very little MBRL work in this direction
- ▶ **Happily, this may be an opportunity!**

PICK-UP(A,B)

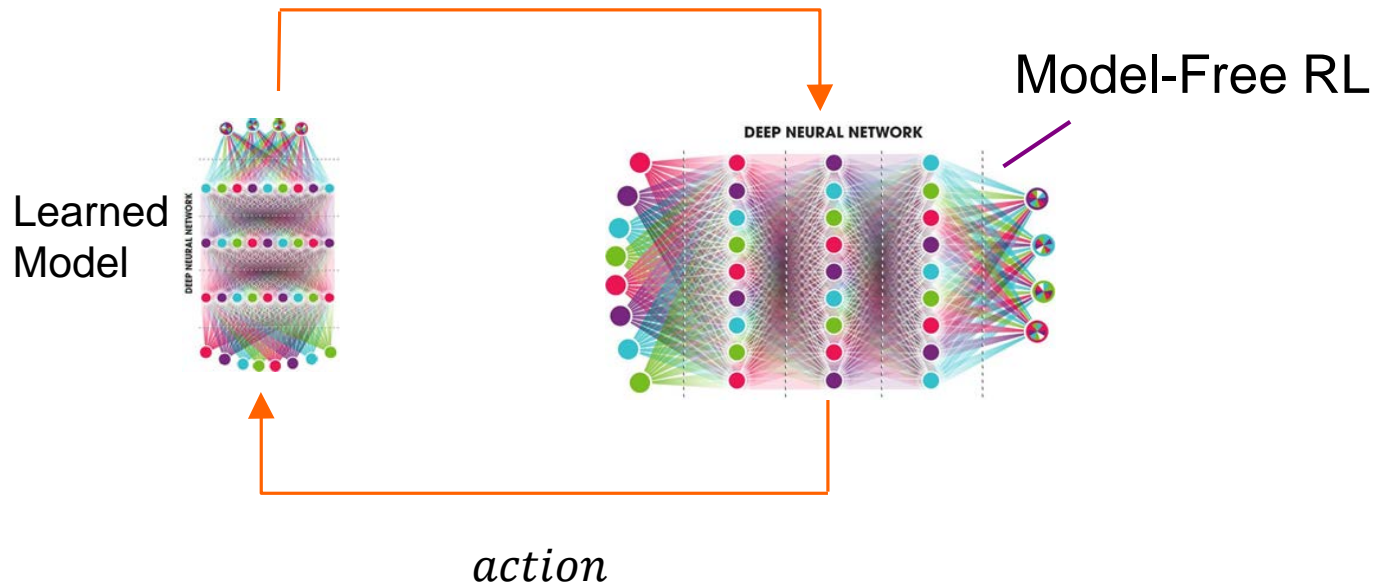
PRE: clear(A), on(A,B)

ADD: holding(A), clear(B)

DEL: on(A,B), clear(A)

MBRL Planner Types

- **Use Model-Free Reinforcement Learning as “Planner”**
 - ▶ Model-Free RL algorithm interacts with learned model to learn a policy
 - ▶ Only requires black-box simulation model
 - ▶ Can provide orders of magnitude more experience than real world
 - ▶ By definition suffers in domains difficult for model-free RL (e.g. sparse reward)
 - ▶ **This is currently the most common type of MBRL approach!**



Outline

Intro to Model-Based Reinforcement Learning (MBRL)

- Brief Introduction to Reinforcement Learning
- MBRL Choices – Types of Models
- MBRL Choices – Types of Planners

Class of Algorithms

- **Class 1: Tabular Models with Optimal Planning**
- Class 2: Simulation Models with Search-Based Planning
- Class 3: Simulation Models for Model-Free “Planning”

Research Directions from Planning Perspective

Learning a Tabular Model from Data

- Suppose we have a data set that contains many sample transitions of taking actions in an MDP M
- Each sample is a tuple (s,a,r,s') --- after taking a in s we got reward r and reached the next state s' .
- How can we estimate the MDP model \hat{M} ?
 - ▲ Reward Function
 - $R(s, a) = r$ if we have a tuple (s, a, r, s') , otherwise
 - $R(s, a)$ is an arbitrary value such as 0
 - ▲ Approximate Transition Function
 - $T(s, a, s') = \frac{\# \text{ of } (s,a,s') \text{ transitions in data}}{\# \text{ of } (s,a) \text{ in data}}$
 - Various ways to handle zero counts

Naïve Model-Based Approach

1. Act Randomly for a (long) time and collect (s,a,r,s') tuples
2. Learn
 - ▶ Transition function
 - ▶ Reward function
3. Apply value/policy iteration to model to get policy
4. Follow resulting policy thereafter.

Will this work? (assume no dead-ends)

Yes, if we do step 1 long enough so learned model is uniformly close enough to true model.

But data collection can be arbitrarily inefficient!

Idea: Use Planning for Efficient Exploration

1. Estimate model with currently available data
 - ▲ Transition function
 - ▲ Reward function
2. Use planning to derive an “exploration policy” (should lead agent to unexplored parts of MDP)
3. Follow exploration resulting policy and collect new transition data
4. Goto step 1

Yes, if we do step 1 long enough so learned model is uniformly close enough to true model.

Idea: Use Planning for Efficient Exploration

- There is a class of MBRL algorithms based on the idea of **optimism in the face of uncertainty**.
- Basically, if the agent has not explored a state “enough”, then it pretends that state gives maximum reward when planning
 - ▲ So planner will try to reach such states
- Many of the theoretical results are based on this idea
 - ▲ We'll only touch on the theory

Optimistic Exploration: Rmax Algorithm

1. Start with an **optimistic model**
(assign largest possible reward to “unexplored states”)
(actions from “unexplored states” only self transition)
2. Solve for optimal policy in optimistic model (standard VI)
3. Take **greedy** action according to policy
4. Update optimistic estimated model
(if a state becomes “known” then use its true statistics)
5. Goto 2

Agent always acts greedily according to a model that assumes all “unexplored” states are maximally rewarding

Rmax: Optimistic Model

- Let $N(s, a)$ be the number of times that action a has been tried in state s
 - ▶ A state-action pair (s, a) is “unexplored” if $N(s, a) < N_e$
 - ▶ A state s is unexplored if for some action a , (s, a) is unexplored
- **Optimistic Model Construction (only in the agents head):**
 - ▶ If $N(s, a) < N_e$ then $T(s, a, s) = 1$ and $R(s, a) = R_{\max}$
 - ▶ Unexplored states have max reward self loops
 - ▶ If $N(s, a) \geq N_e$ then $T(s, a, s')$ and $R(s, a)$ are estimated from the N_e experiences
 - ▶ Explored states are estimated from observed data
- For large enough N_e the explored states will have accurate models

Optimistic Exploration

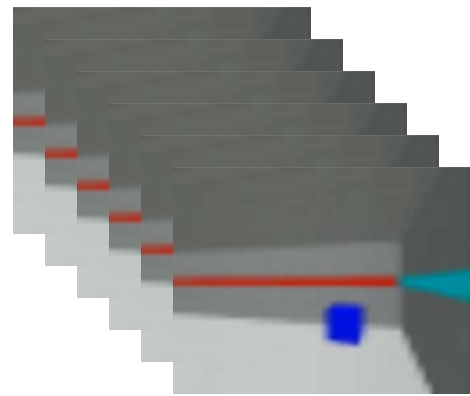
- Is Rmax provably efficient?
 - ▶ If the model is ever completely learned (i.e. $N(s,a) > N_e$, for all (s,a)), then the policy will be near optimal.
 - ▶ Theoretical results show that this will happen “quickly”
- **Theoretical Guarantee (Roughly speaking):**
There is a value of N_e (depending on n, m , and R_{\max}), such that with high probability the Rmax algorithm will select at most a polynomial number of actions with value less than ϵ of optimal.
- RL can be solved in poly-time in n , m , and R_{\max} !

Thoughts on Model-Based RL

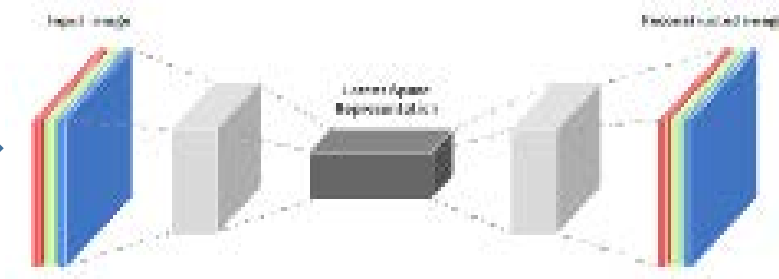
- Model-based methods like RMAX have established some fundamental theory about **finite-time convergence** to near optimal policies
- But in practice they are difficult to use for large MDPs
 - ▲ Require storing and solving an estimated MDP model
- How could we use these methods for enormous MDPs?

Tabular MBRL for Enormous Worlds

Prior Experience = $\{(s,a,r,s')\}$



Latent State Representation Learning



Compilation to Tabular
Markov Decision Process (MDP)

Tabular Model



Value Iteration
(maybe on GPU)

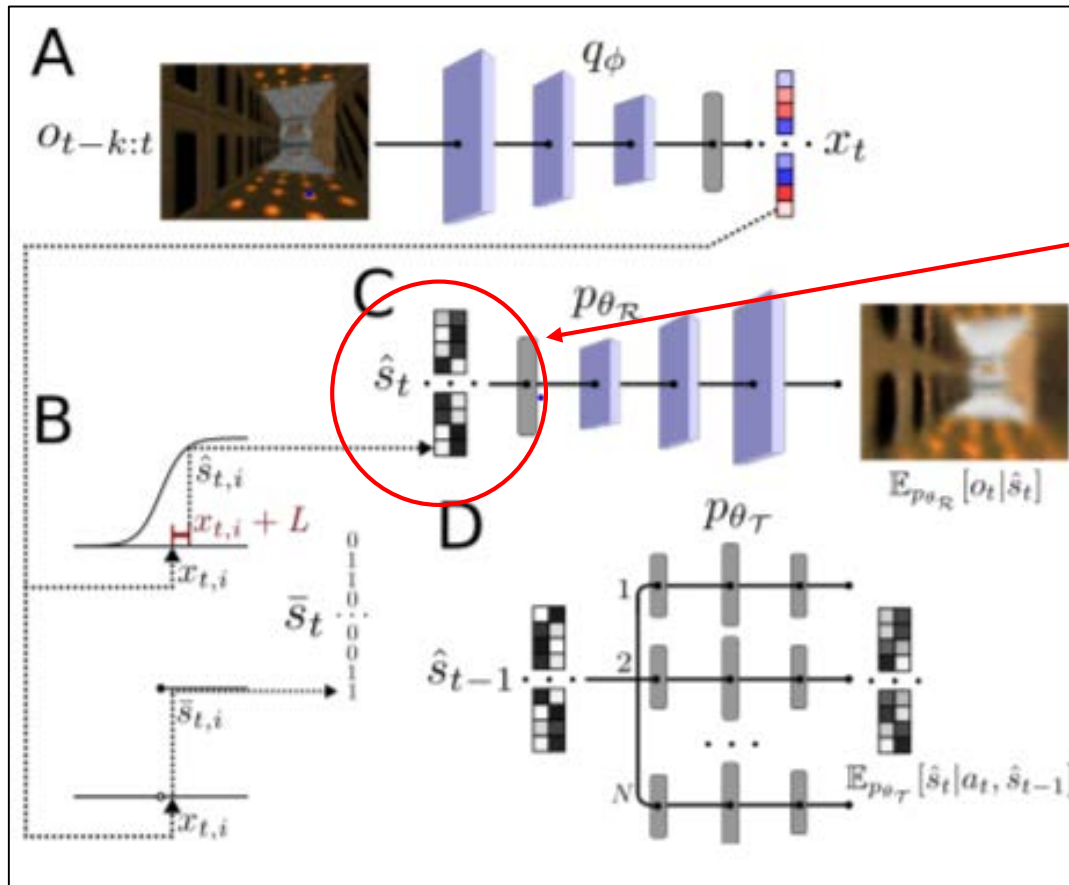
Policy

- Maps new observations to nearest (or k nearest) tabular states
- “nearest” is based on learned latent representation

Tabular MBRL for Enormous Worlds

Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation. Dane Corneil, Wulfram Gerstner, Johanni Brea. ICML 2018.

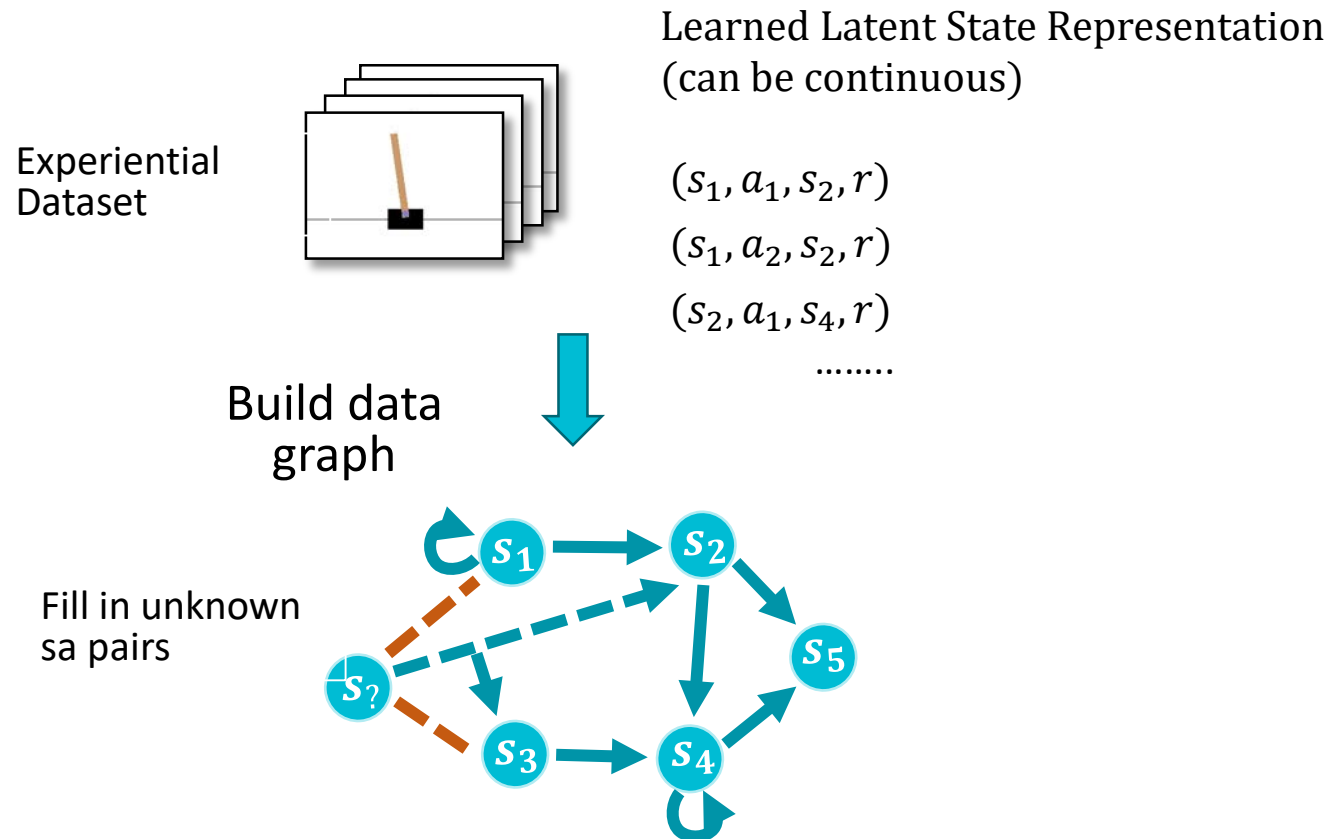
Variational Auto Encoder



- Learn variational auto encoder over observations with **Discrete Bottleneck**
- Use as discrete state space
- Promising results in 3D navigation domain
- Has not yet shown robustness and scaling to Atari.

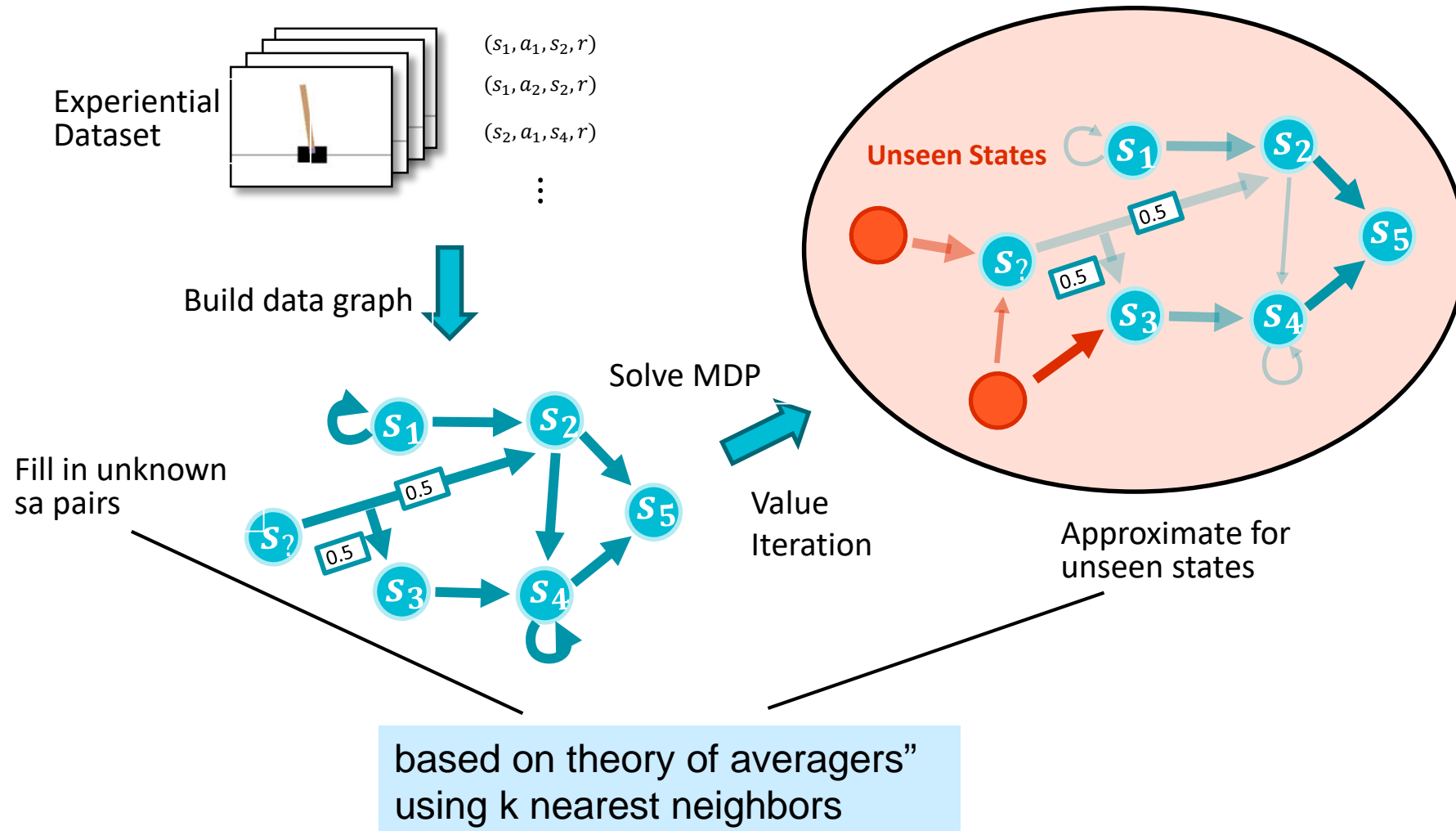
Tabular MBRL for Enormous Worlds

DeepAveragers (soon to be on arxiv) also see my invited talk at ICAPS WS on Bridging the Gap Between AI Planning and Reinforcement Learning



Tabular MBRL for Enormous Worlds

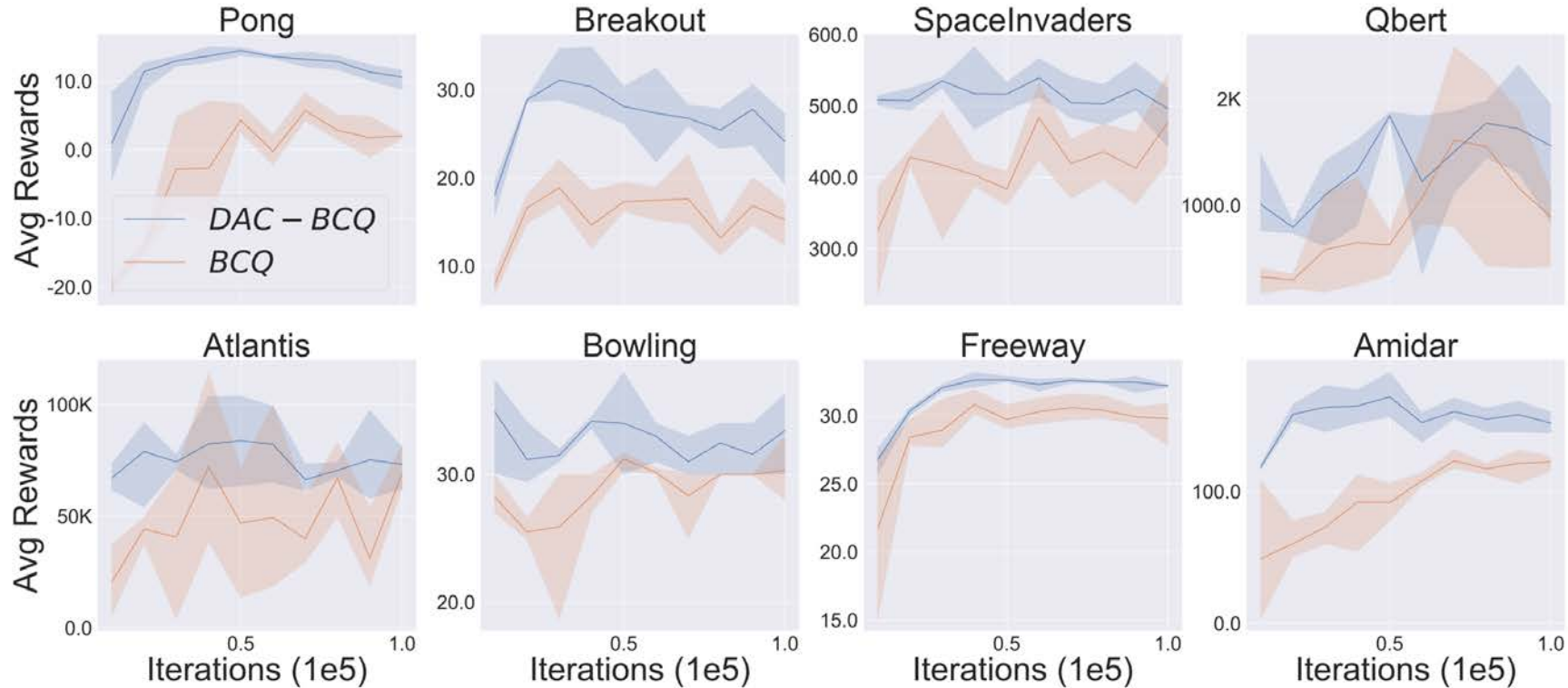
DeepAveragers (soon to be on arxiv) also see my invited talk at ICAPS WS on Bridging the Gap Between AI Planning and Reinforcement Learning



Atari Results (only 100K transitions)

BCQ = model-free algorithm

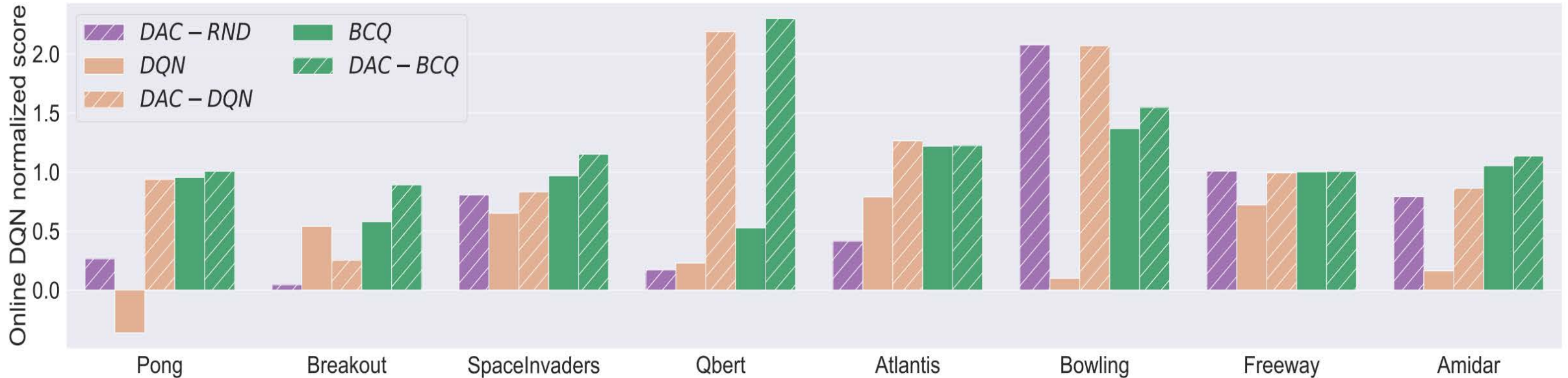
DAC-BCQ = Value Iteration Policy using BCQ Representation



GPU implementation of VI easily scales to 3M states

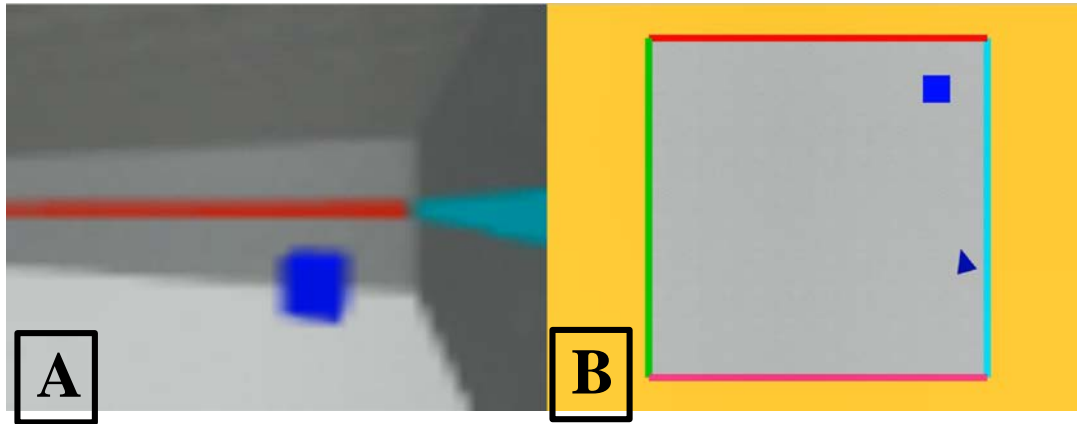
Atari Results (only 2.5M transitions)

Basic GPU VI implementation easily scales to 3M states
(currently limited to 10s of actions – memory constraints)

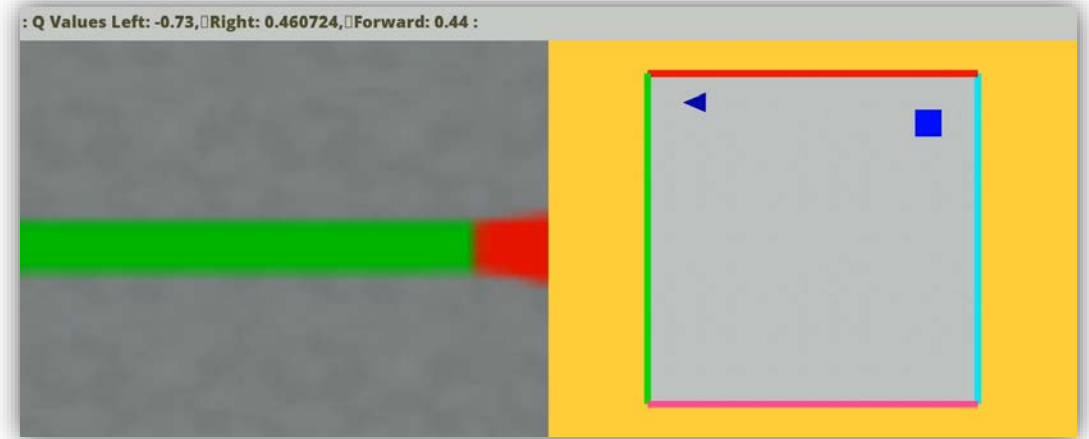


BCQ and DQN = model-free algorithm
DAC-BCQ and DAC-DQN = model-based

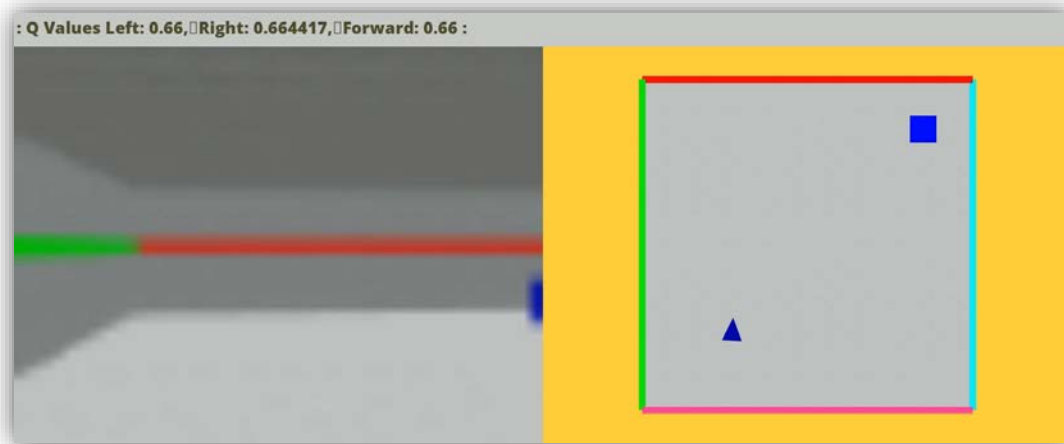
MBRL Use Case (“zero-shot transfer”)



(A) Agent View of the world. (B) Top view of the world. Agent receives a reward of +1 on reaching near the box. -1 for bumping into a wall. Episode ends once goal is reached. Actions Available: Forward, Right, Left



Point Goal Navigation Task. Left action penalized



[Generated] Optimal policy for Point Goal Navigation Task.



Point Goal Navigation Task. Right action penalized

Outline

Intro to Model-Based Reinforcement Learning (MBRL)

- Brief Introduction to Reinforcement Learning
- MBRL Choices – Types of Models
- MBRL Choices – Types of Planners

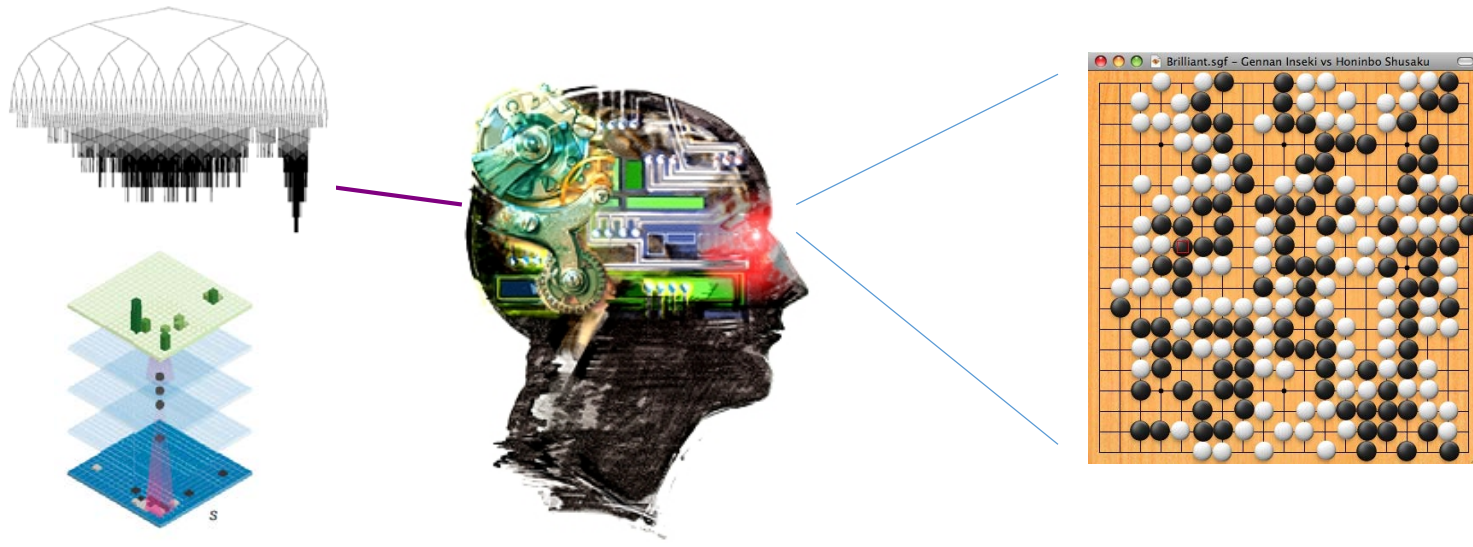
Class of Algorithms

- Class 1: Tabular Models with Optimal Planning
- **Class 2: Simulation Models with Search-Based Planning**
- Class 3: Simulation Models for Model-Free “Planning”

Research Directions from Planning Perspective

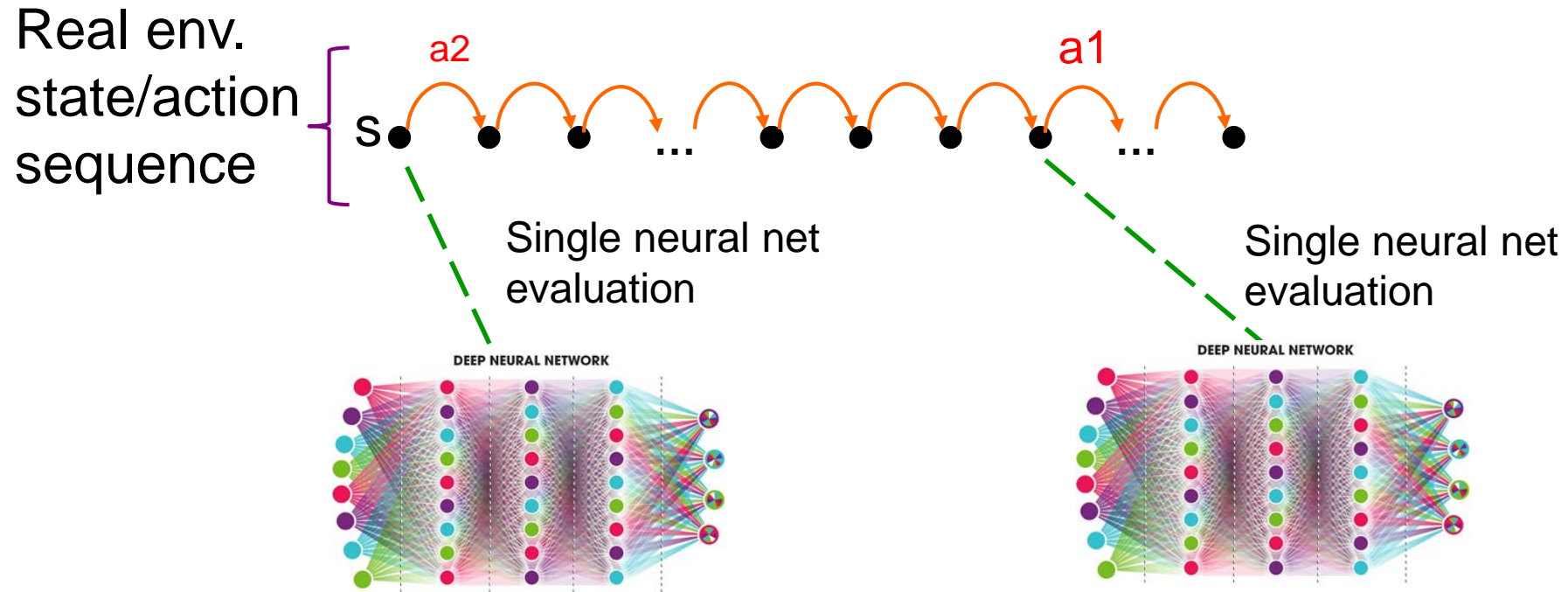
Most Famous Examples

AlphaGo → AlphaZero → MuZero



Reactive Policies vs. Deliberative Search

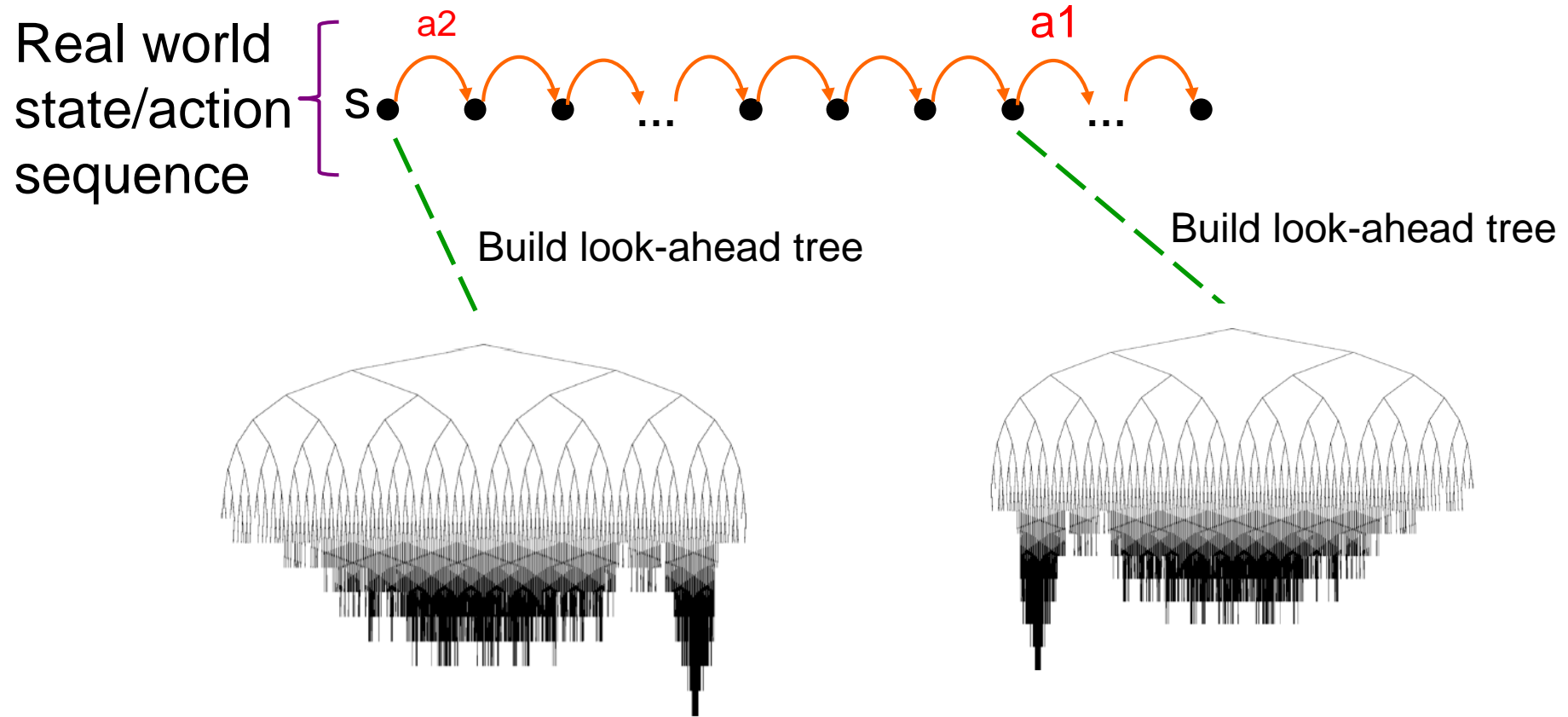
So far the output of RL has been a reactive policy.



But some environment seem like they require
“deliberation” at each decision step (e.g. Chess, Go)

Reactive Policies vs. Deliberative Search

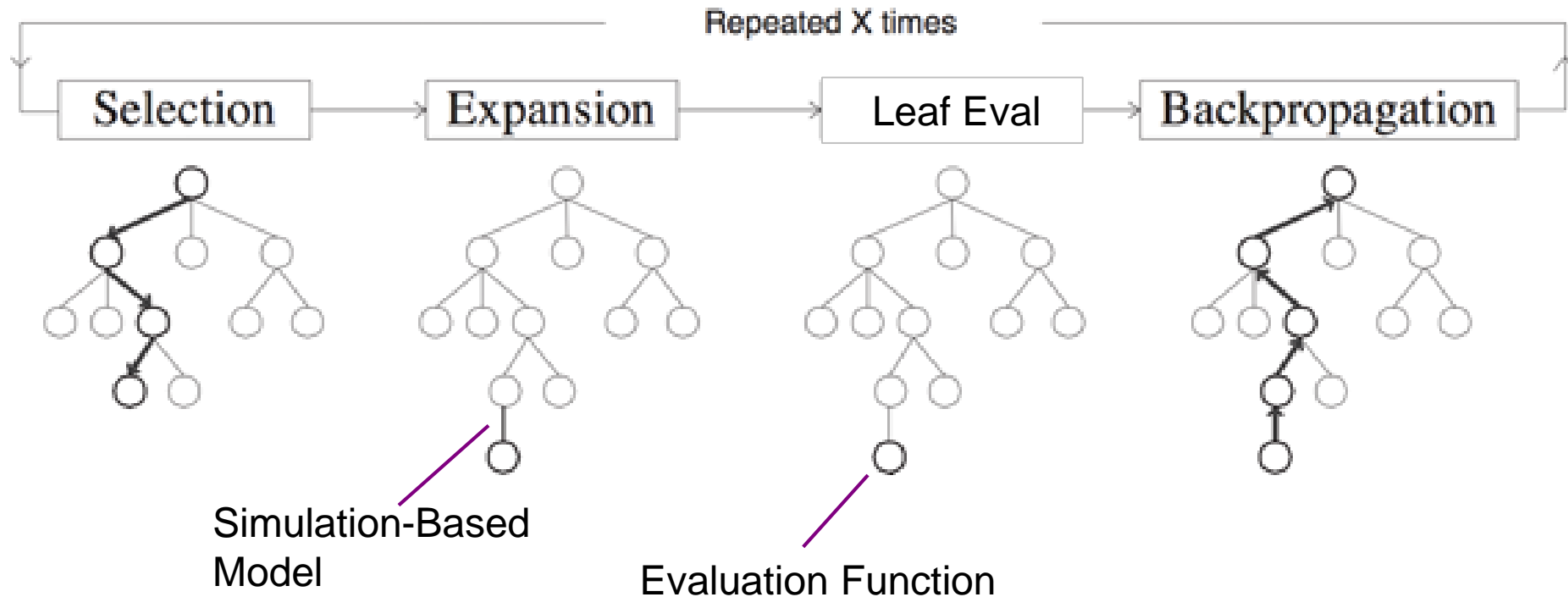
Let's consider MBRL that uses tree search to select actions.



Let's start with AlphaGo and AlphaZero.

Monte Carlo Tree Search

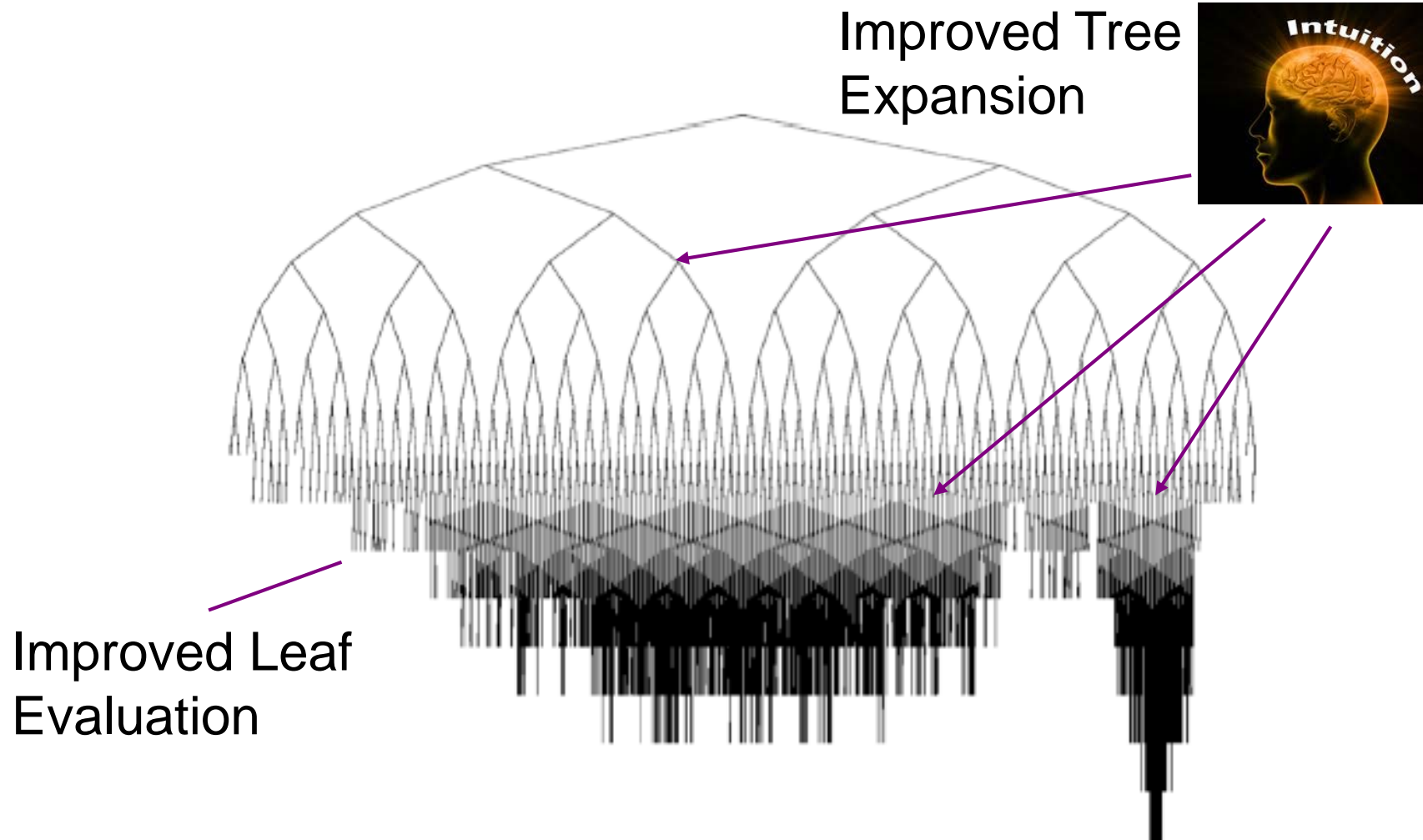
Both AlphaGo and AlphaZero use Monte-Carlo Tree Search



Both assume a perfect simulation-based model is given!

Should this still be considered MBRL?

Monte Carlo Tree Search



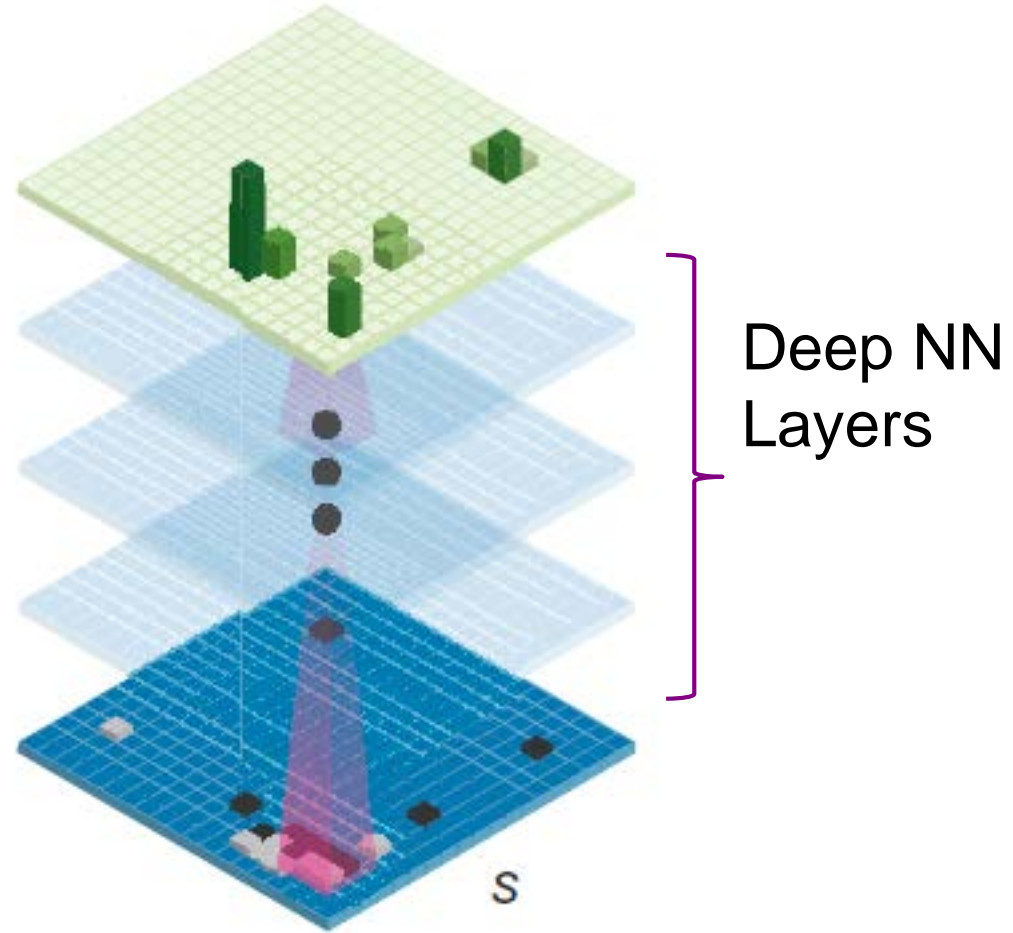
If we don't need to learn a model,
then what is left to learn?

Improving Tree Expansion

Idea: learn neural network that predicts probability that each action will lead to a win

That is, learn a probabilistic policy.

Output: probability of each move



Input: Board Position

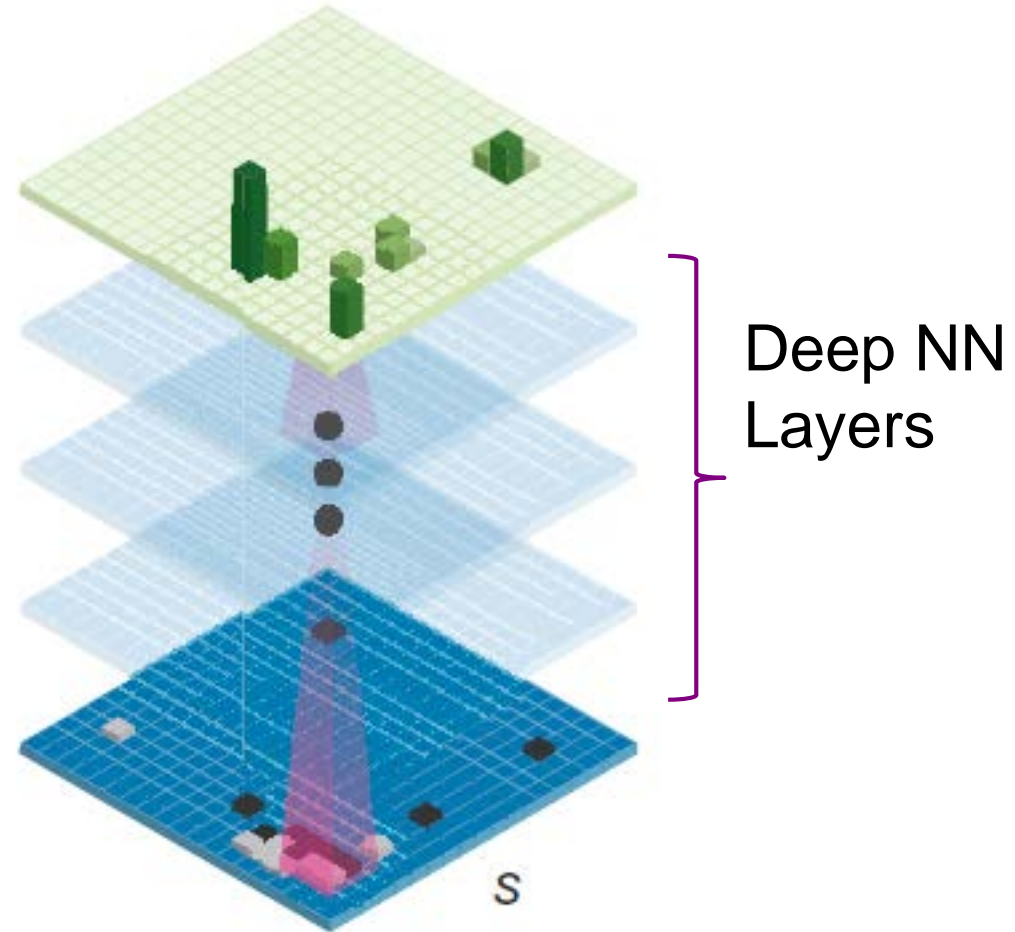
Improving Leaf Evaluation

Idea: learn neural network that predicts probability of win for a board position

That is, learn a value function.

Could just use max probability over actions.

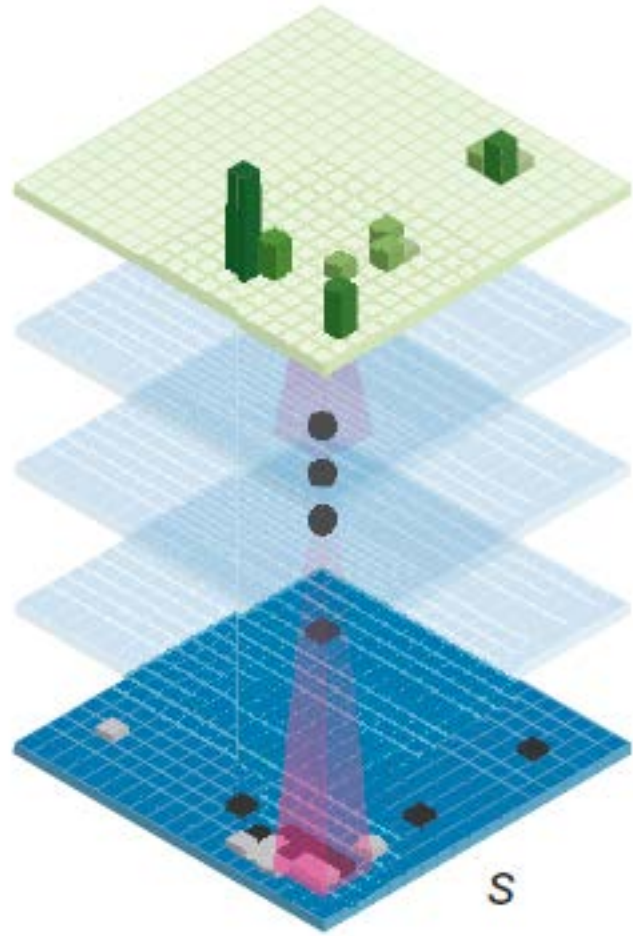
Output: probability of each move



Input: Board Position

AlphaGo – Bootstrapping via Supervised Learning

Output: probability of each move



Deep NN Internal
Layers

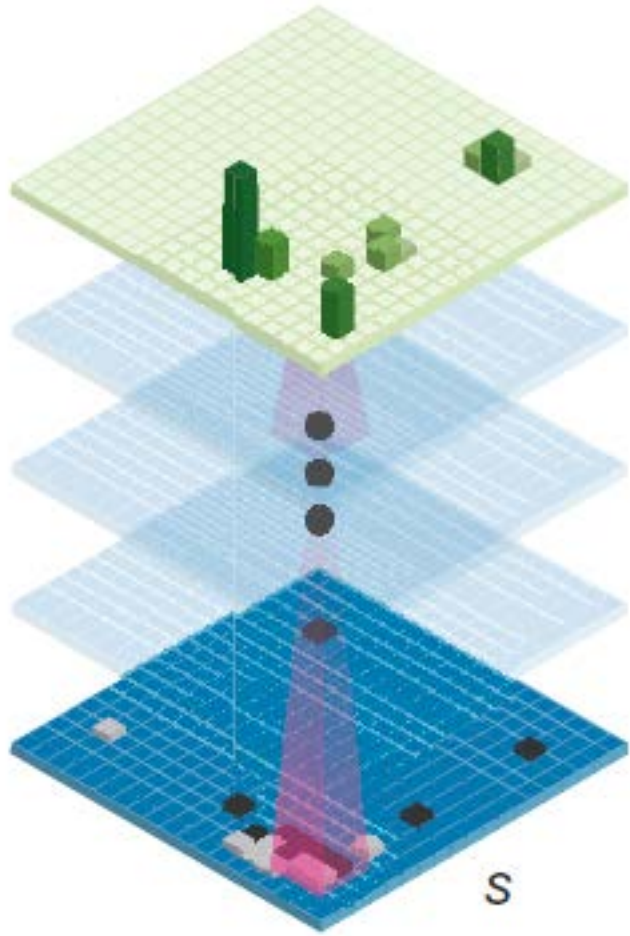
Trained for 3 weeks on 30 million
expert moves

- 57% prediction accuracy!

Input: Board Position

Supervised Learning for Go

Output: probability of each move ~~being played by an expert~~
leading to a win



Input: Board Position

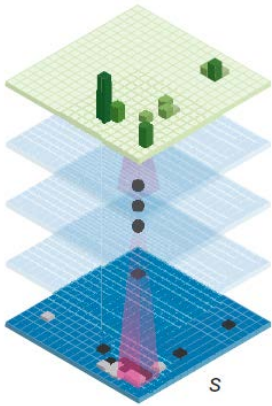
AlphaGo has still not played a game of Go!

Could it improve further by playing?

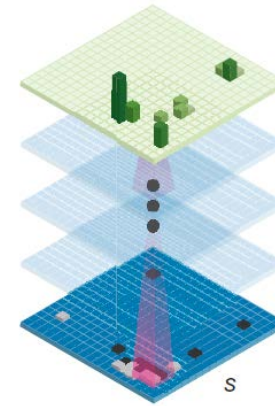
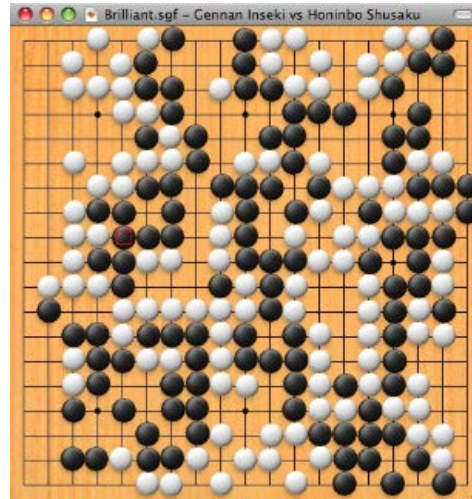
Use model-free RL!

AlphaGo - Learning from Self Play

Play 1000 Games



Learning Network

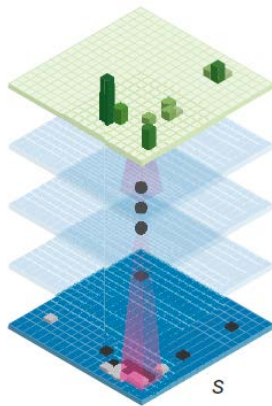


Frozen Network
ver 0

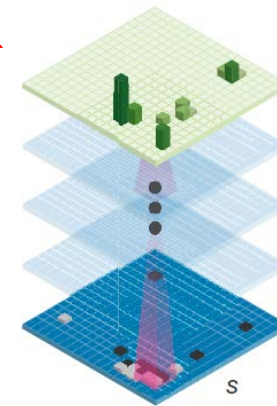
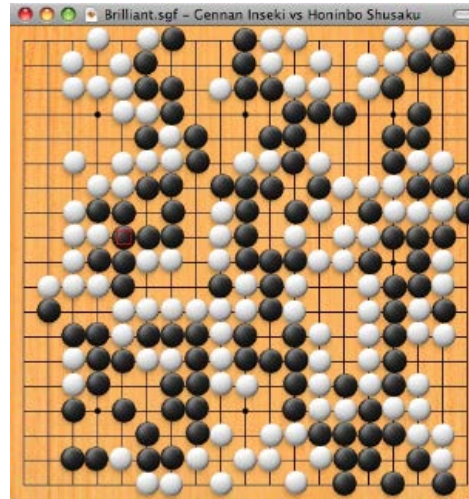
Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Transfer Improved Network
(brain transplant)



Learning Network

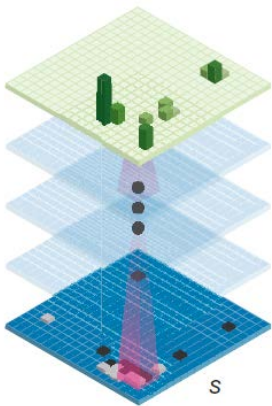


Frozen Network
ver 0

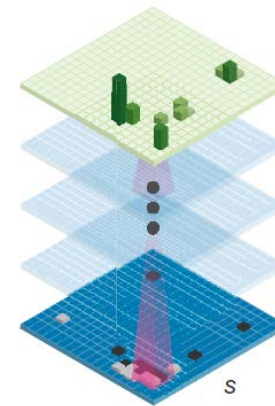
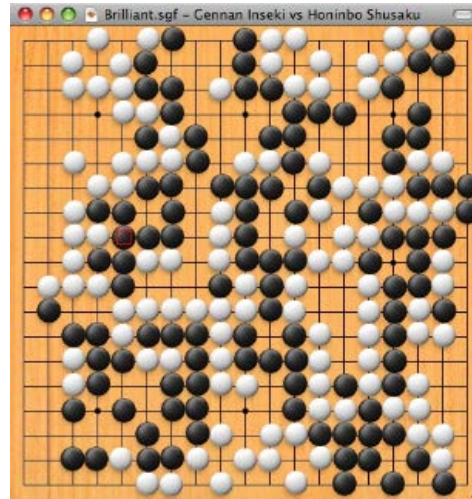
Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Play 1000 Games



Learning Network

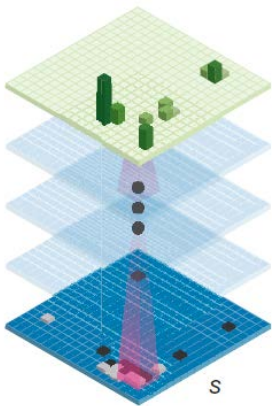


Frozen Network
ver 1

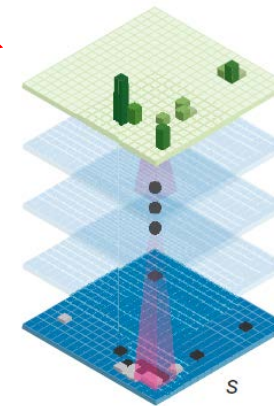
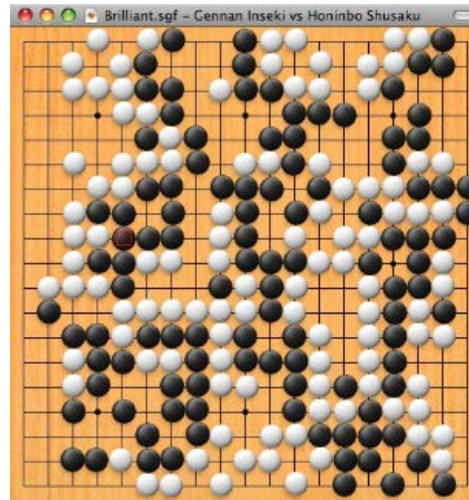
Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Transfer Improved Network
(brain transplant)



Learning Network

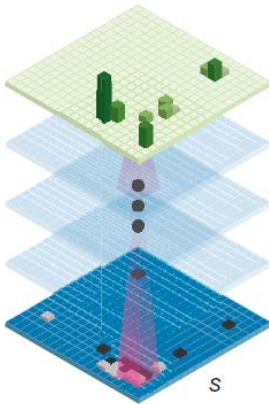


Frozen Network
ver 1

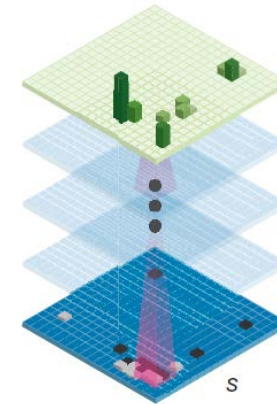
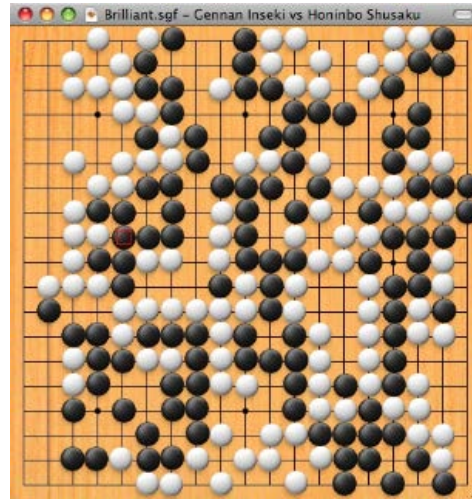
Reinforcement Learning : learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Play 1000 Games



Learning Network

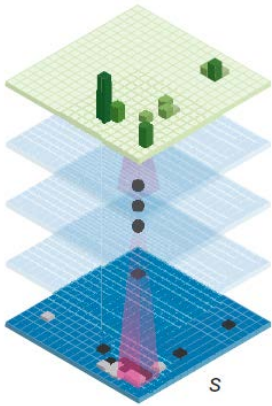


Frozen Network
ver 2

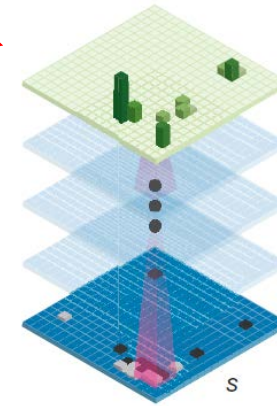
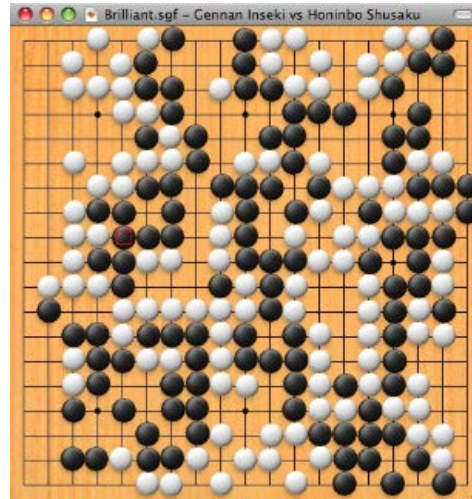
Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Transfer Improved Network
(brain transplant)



Learning Network

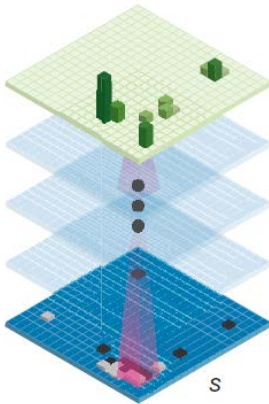


Frozen Network
ver 2

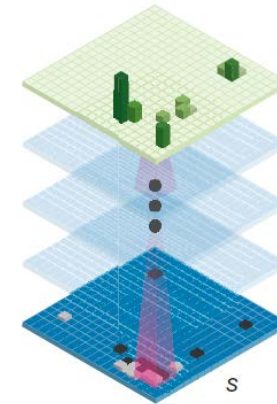
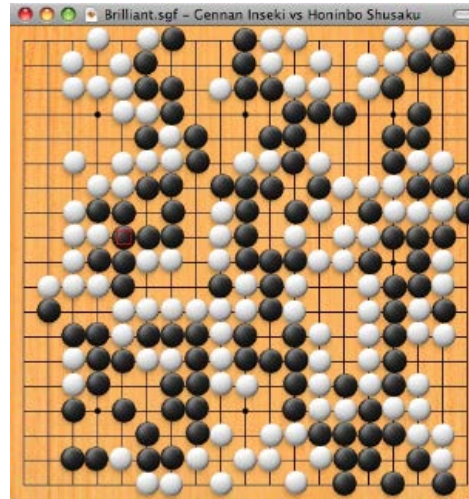
Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Learning from Self Play

Eventually learns to play much better!



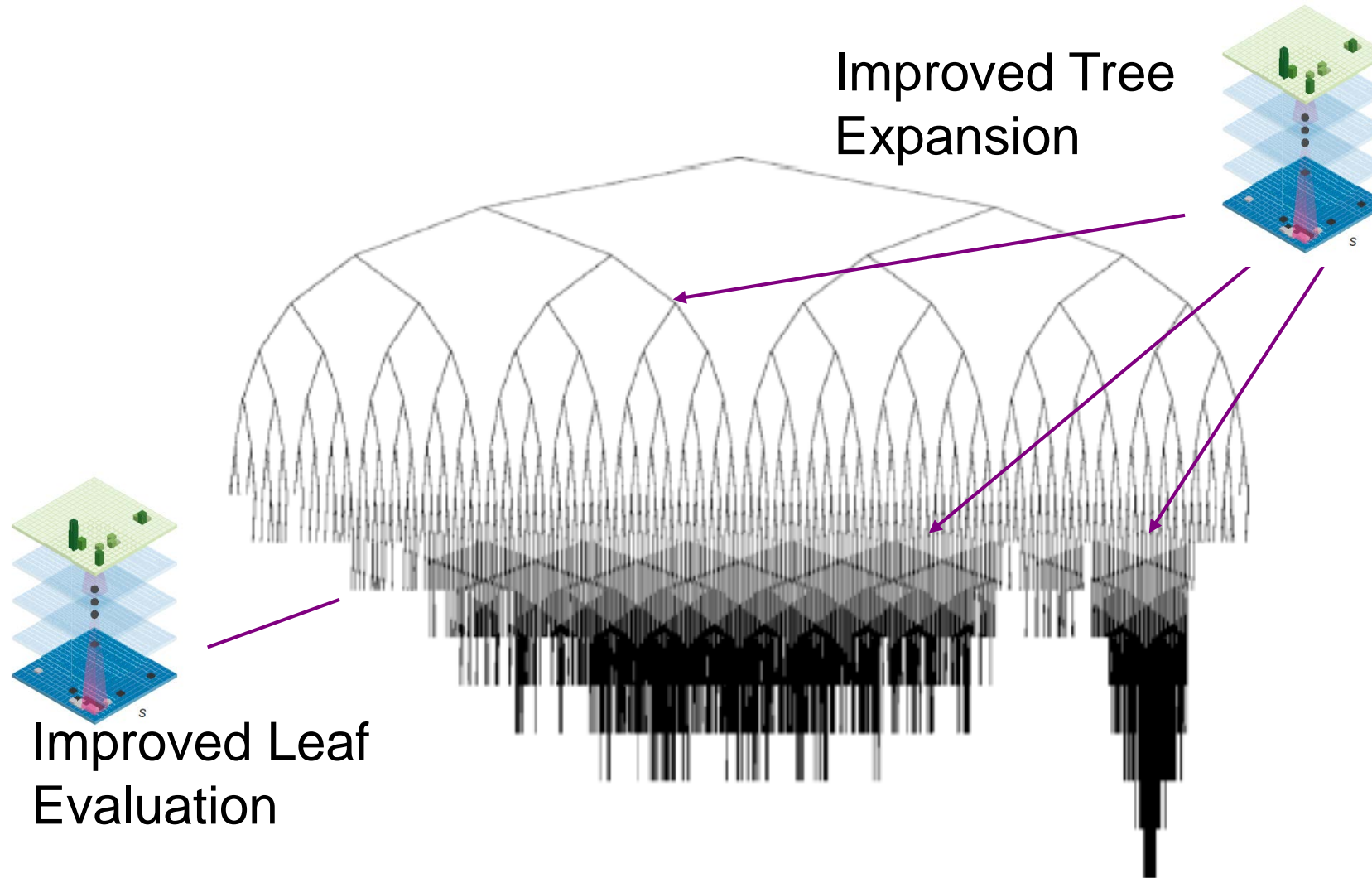
Learning Network



Frozen Network
ver 1,000,000

Model-Free RL: learn from positive and negative rewards (win = +1 and loss = 0 in Go)

Monte Carlo Tree Search



There are 1001 ways to incorporate networks into tree search. They tuned search approach via lots of experimentation.

AlphaGo

- Deep Learning + Monte Carlo Tree Search + HPC
- Learn from 30 million expert moves and self play
- Highly parallel search implementation



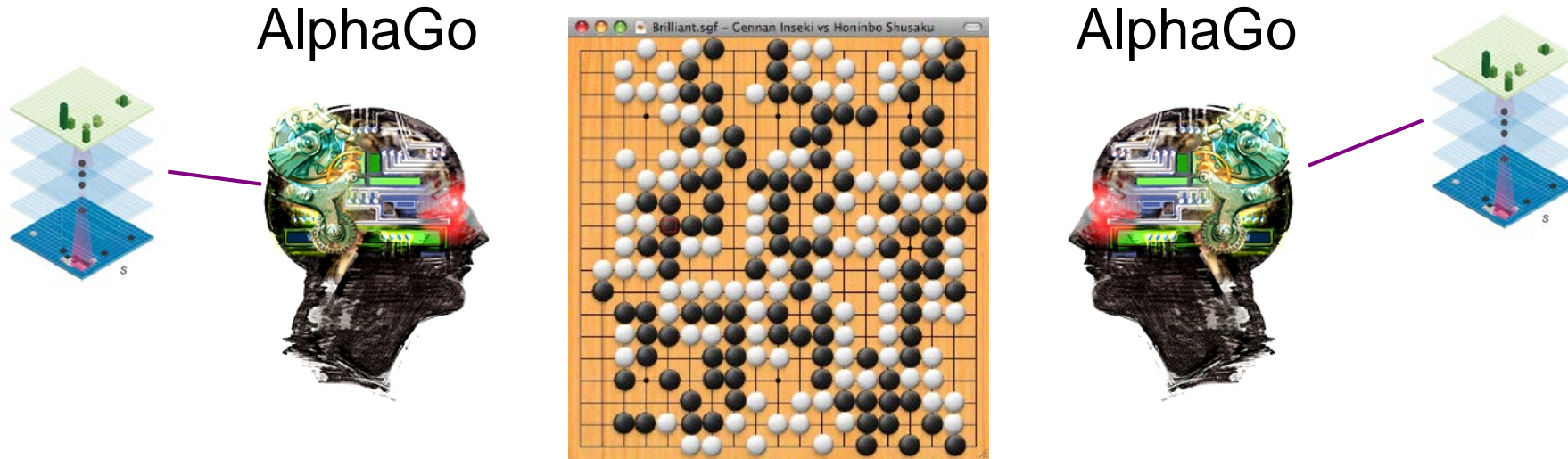
Image: AP/ Lee Jin-man

March 2016 :
AlphaGo beats Lee Sedol 4-1

But AlphaGo required
bootstrapping from human
experts.

We'd prefer to learn from scratch!

Learning from Self Play: AlphaGo

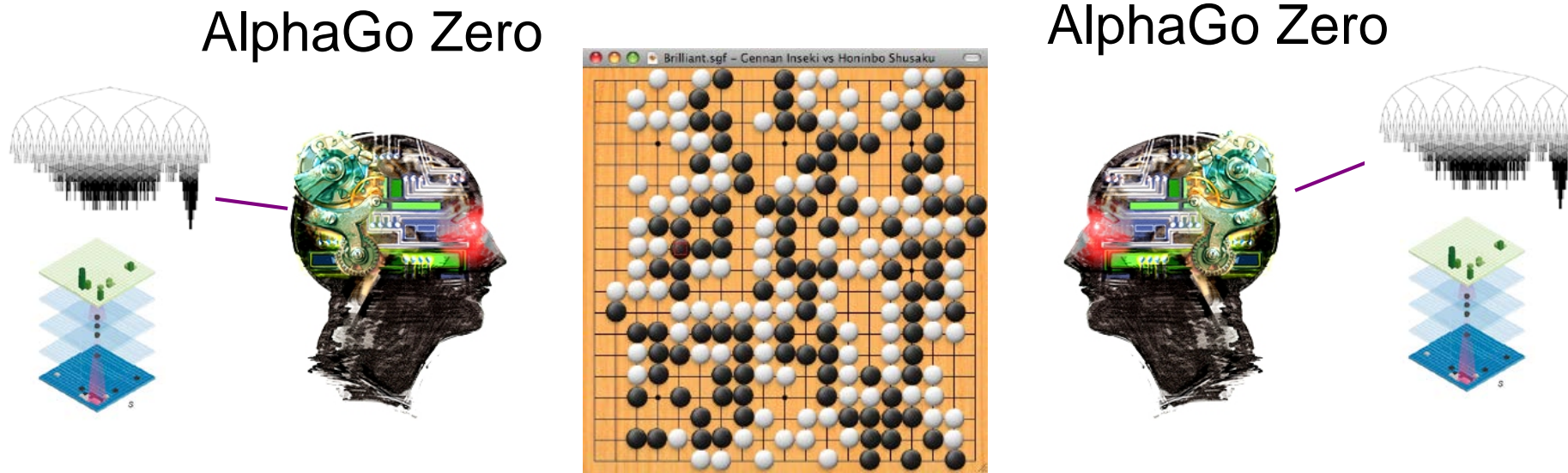


AlphaGo used just the neural network to make decisions during self-play.

Then inserted neural network into search for real games.

There seems to be a limit on the performance of a single neural network with no search!

Learning from Self Play: AlphaZero

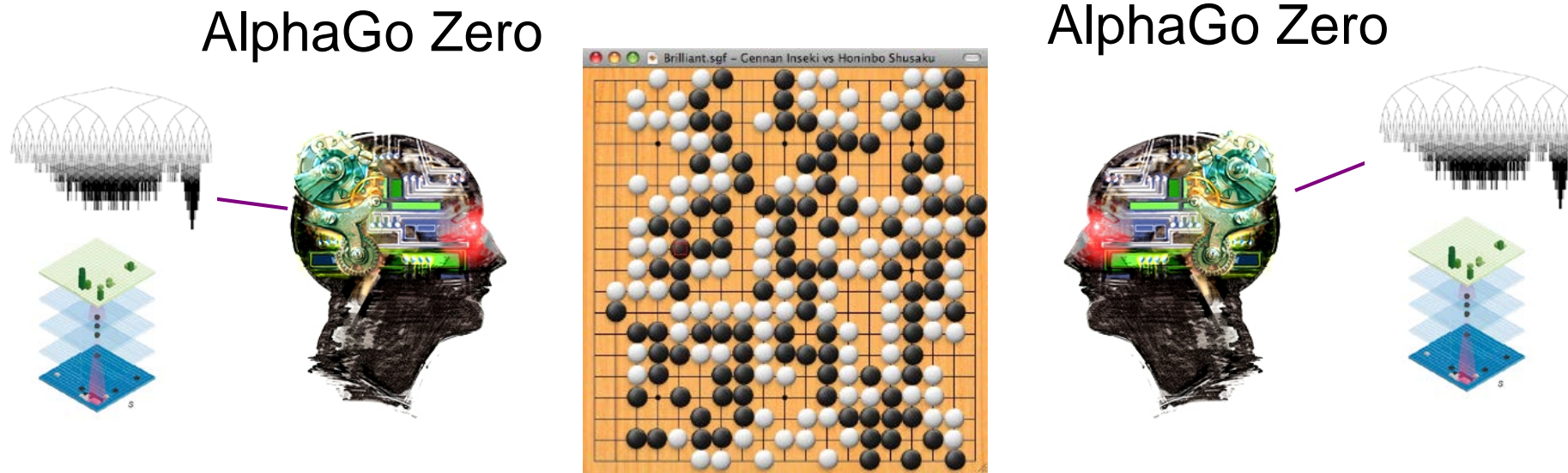


AlphaGo Zero uses Search + Neural Net during self-play.

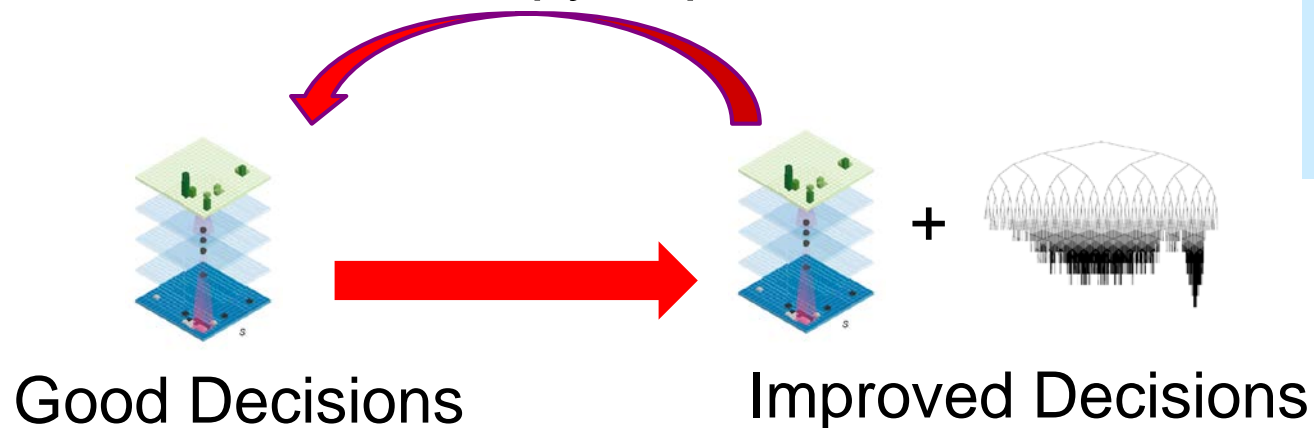
Trains neural nets on decisions found by Search + Neural Net

No bootstrapping from human experts!

Learning from Self Play: AlphaZero

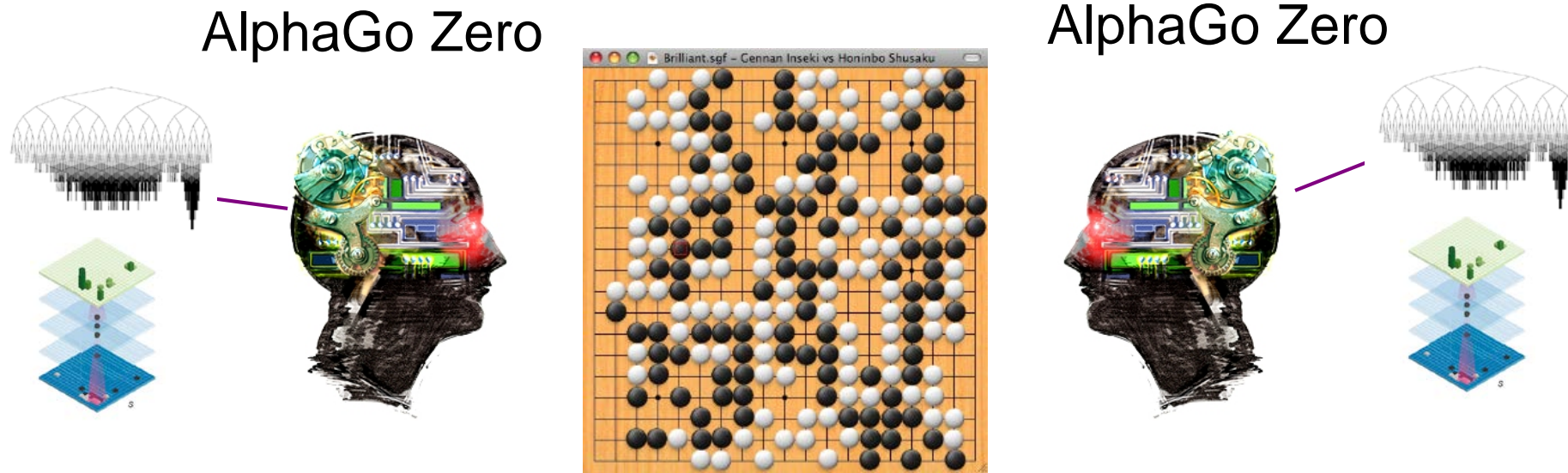


Train network to copy improved decisions



Bootstrap via
search rather than
human experts!

Learning from Self Play: AlphaZero



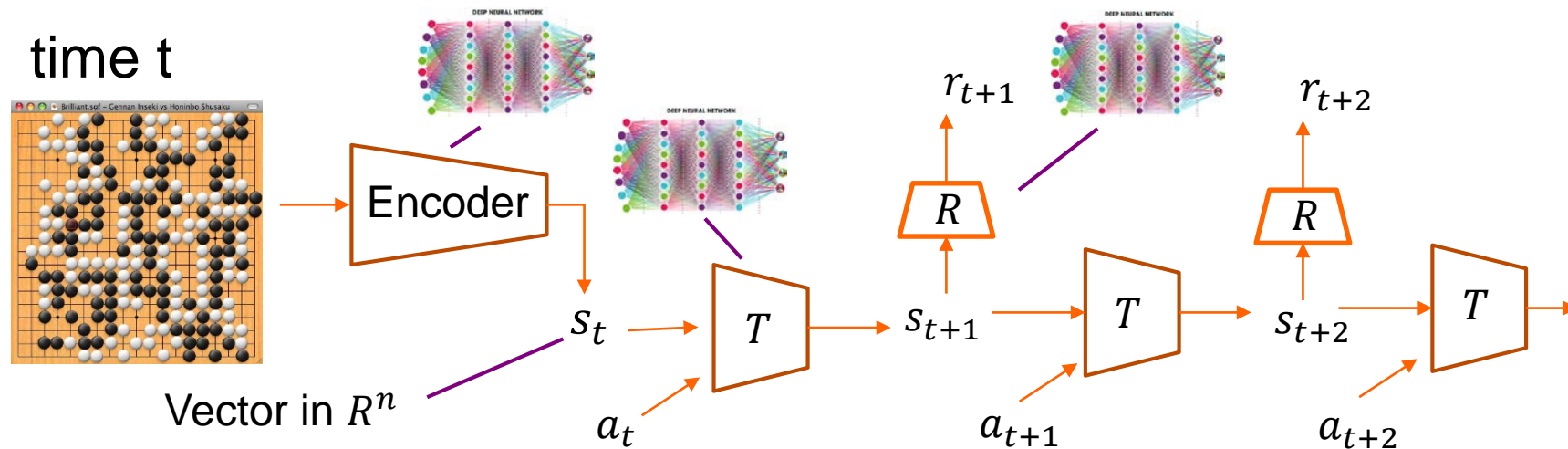
But AlphaGo and AlphaZero both assume access to a perfect world model!

End the trilogy with MuZero – learns latent space simulation model + knowledge to control tree-search in latent space

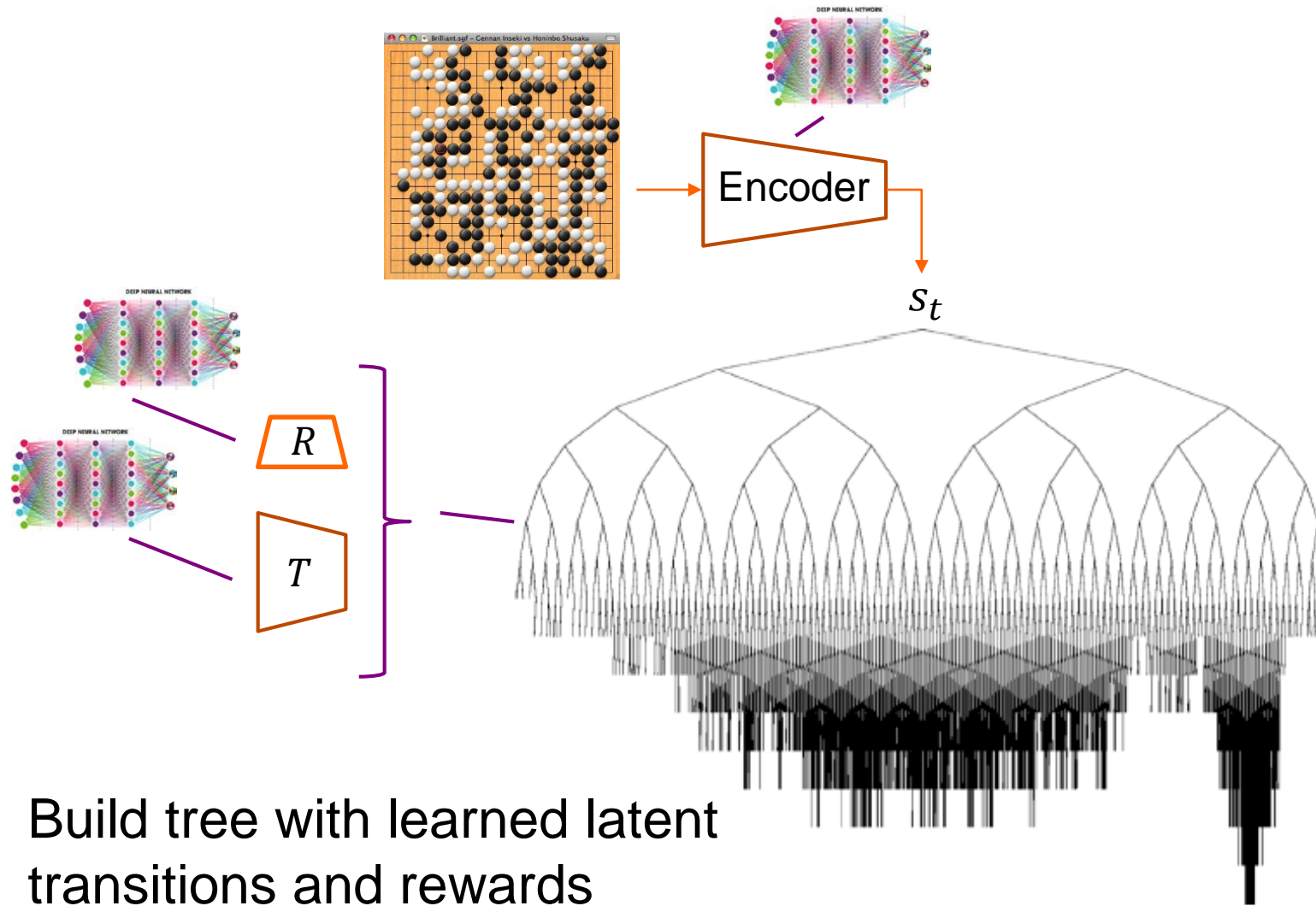
MuZero – Learn Latent State Model

- **Latent State Model**

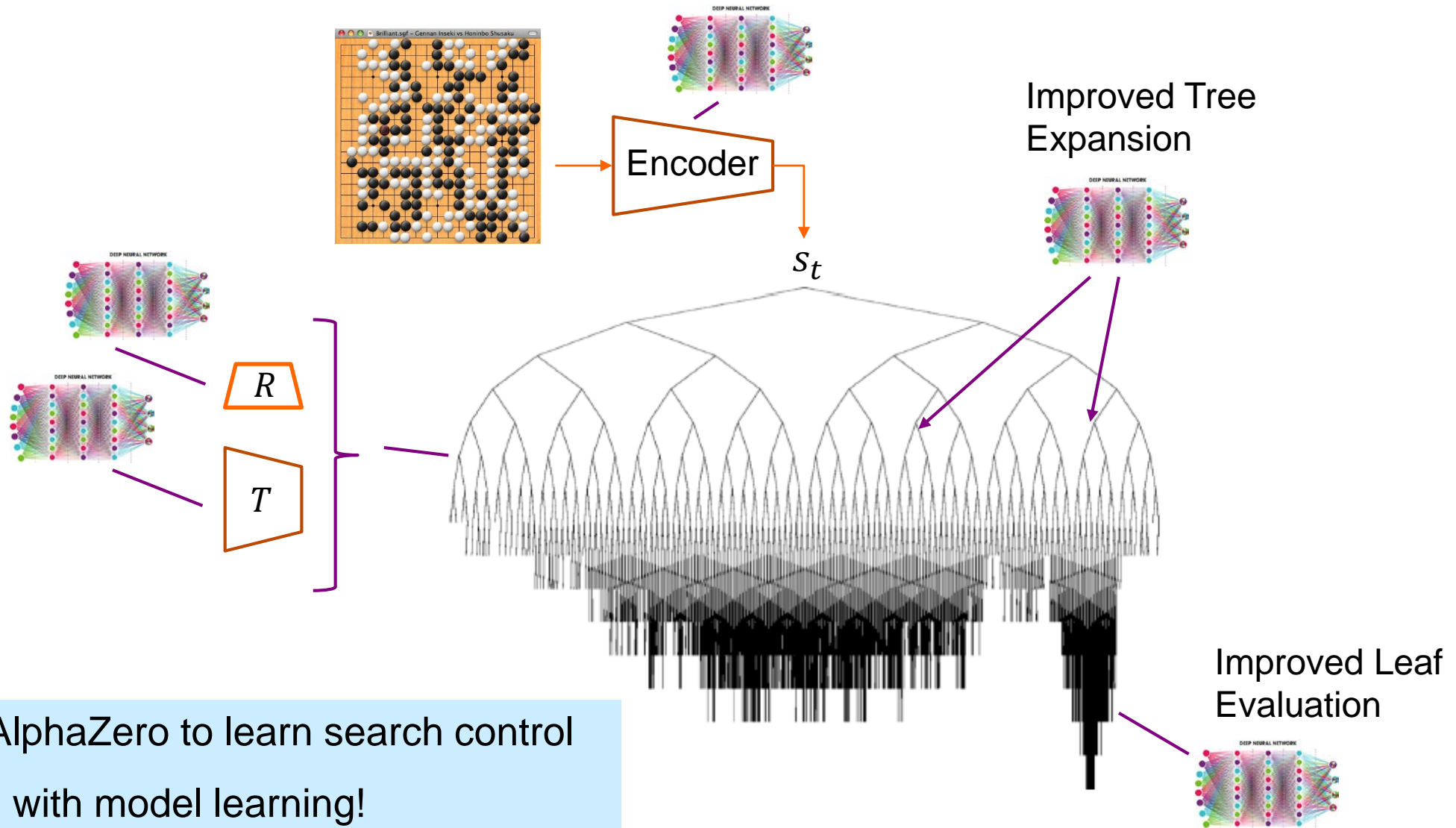
- ▶ Learns to encode observations to a latent state space from self-play
- ▶ Learns dynamics and reward function in latent space



MuZero – Tree Search in Latent Space

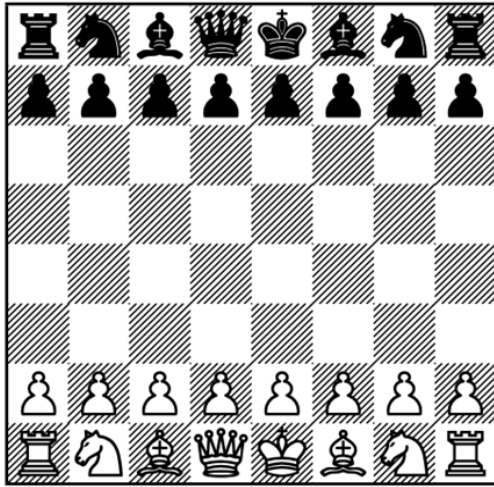


MuZero – Tree Search in Latent Space

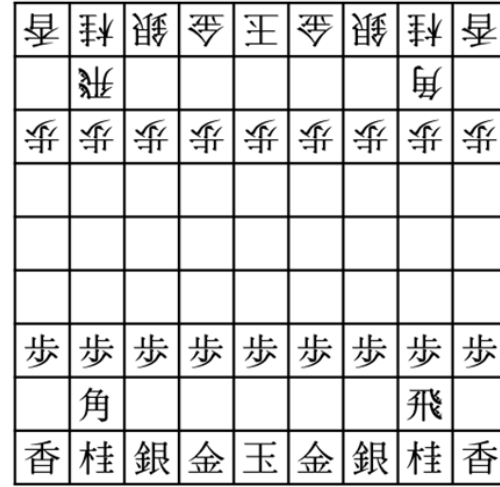


Results

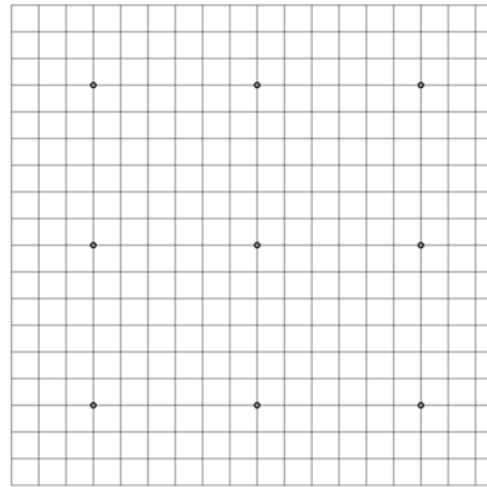
Chess



Shogi

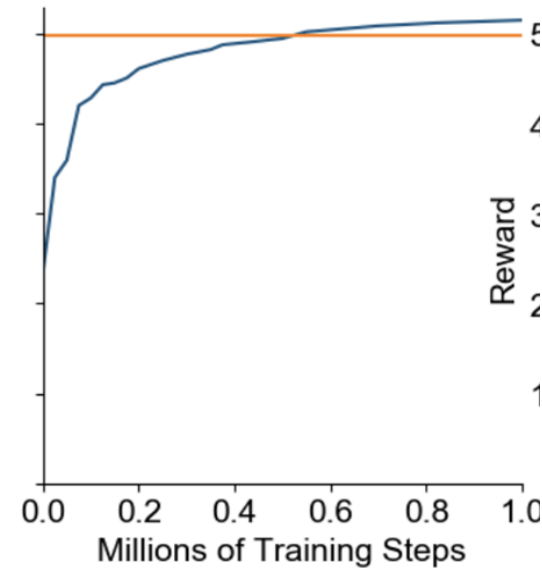
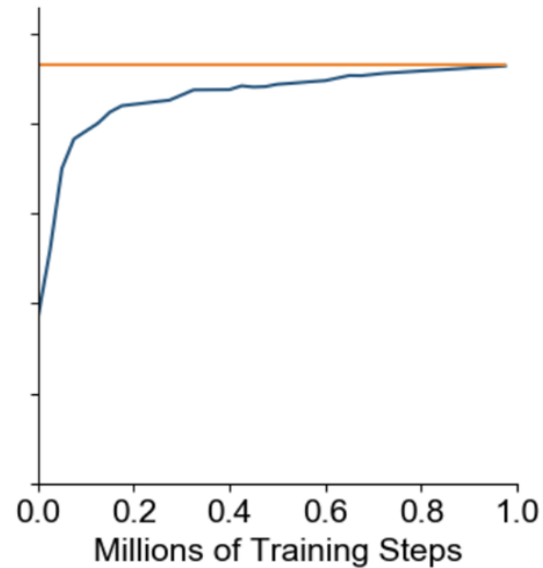
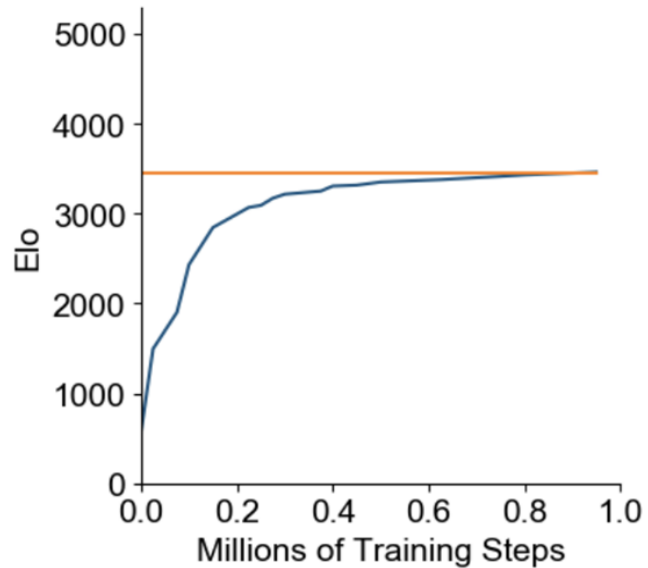


Go



— AlphaZero
— MuZero

Muzero gets better performance despite having smaller architecture than AlphaZero.



Outline

Intro to Model-Based Reinforcement Learning (MBRL)

- Brief Introduction to Reinforcement Learning
- MBRL Choices – Types of Models
- MBRL Choices – Types of Planners

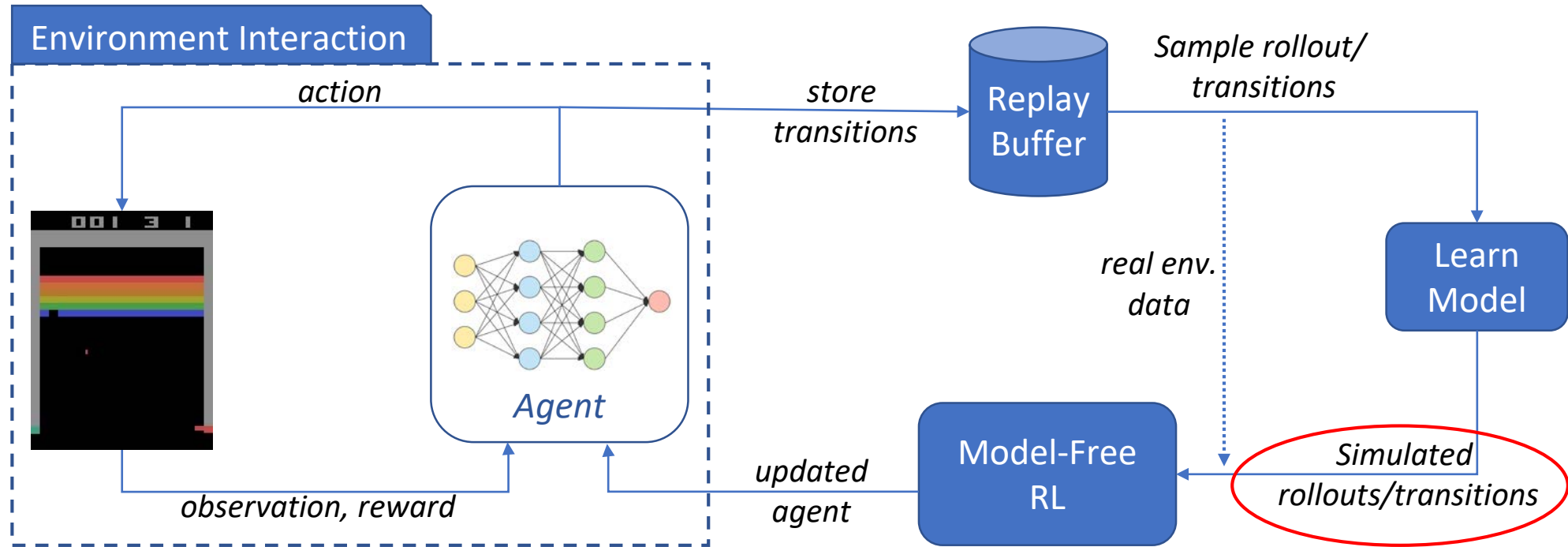
Class of Algorithms

- Class 1: Tabular Models with Optimal Planning
- Class 2: Simulation Models with Search-Based Planning
- **Class 3: Simulation Models for Model-Free “Planning”**

Research Directions from Planning Perspective

Basic Template

1001 variations of this idea
..... and growing every day



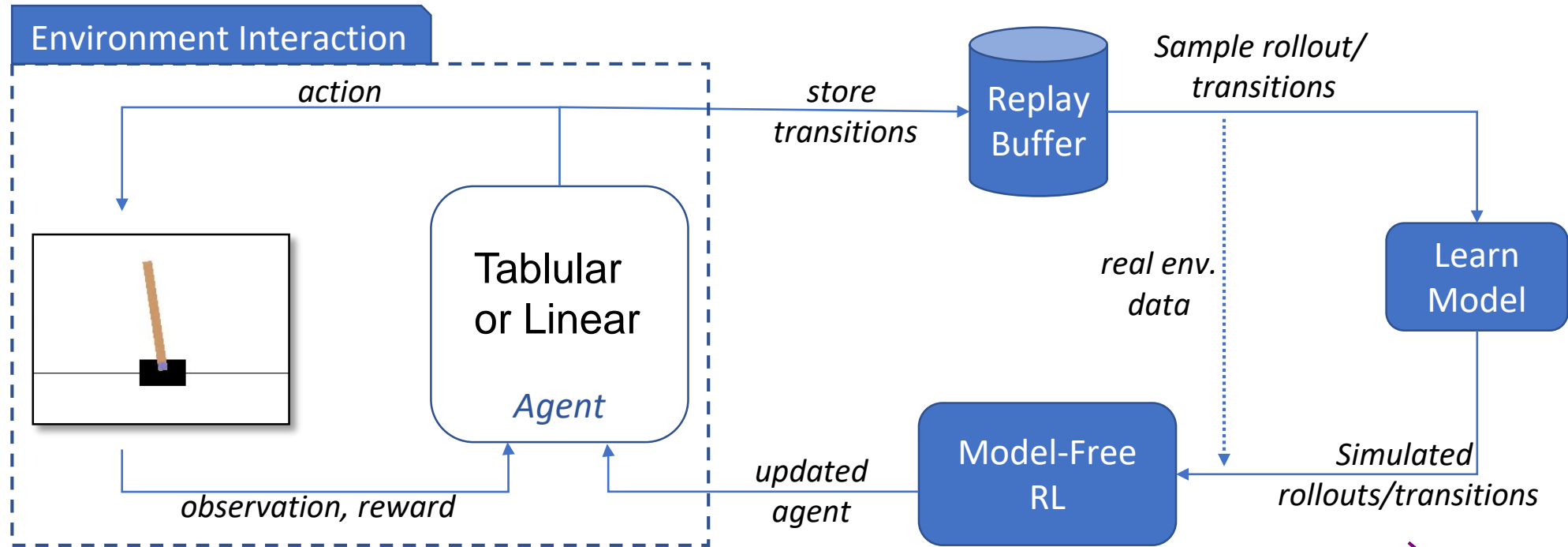
Advantage: can handle continuous action domains by using appropriate model-free RL agent!

Learns from both real and imagined transitions

Sometimes called
"imagined" transitions

Dyna-Q

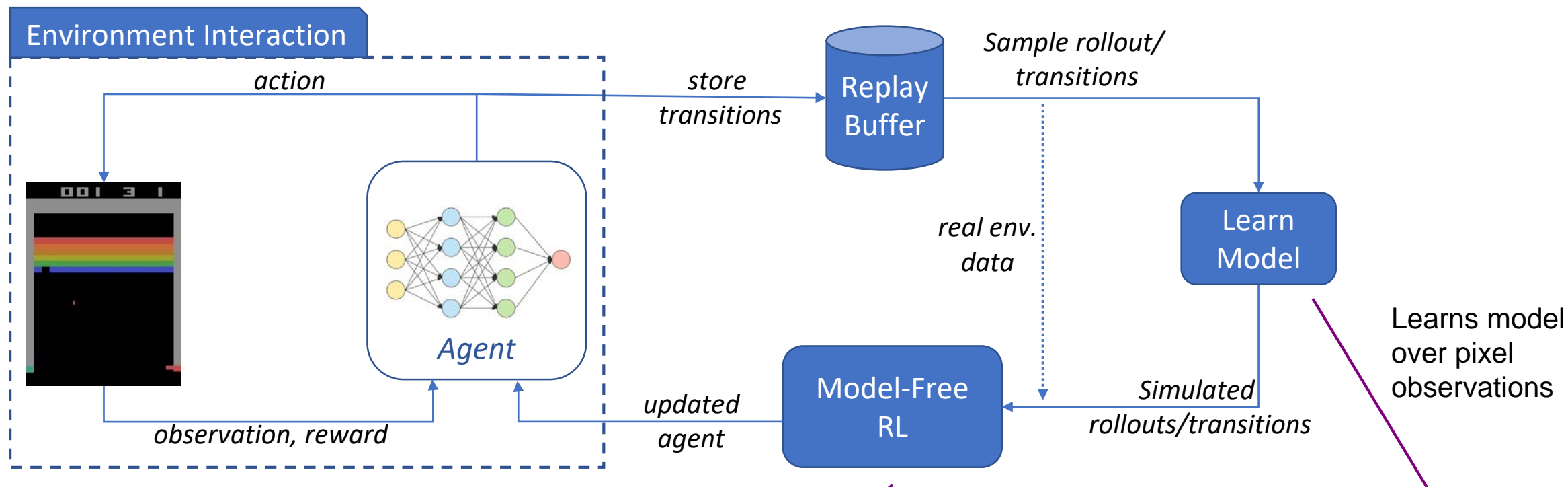
Richard S. Sutton. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*.



Long before the era of Deep RL
Many deep adaptations

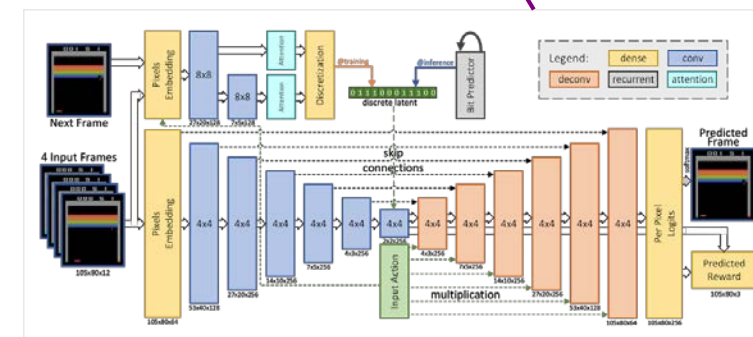
Simulated Policy Learning (SimPle)

Kaiser et al. (2020). Model-Based RL for Atari.
International Conference on Learning Representations.



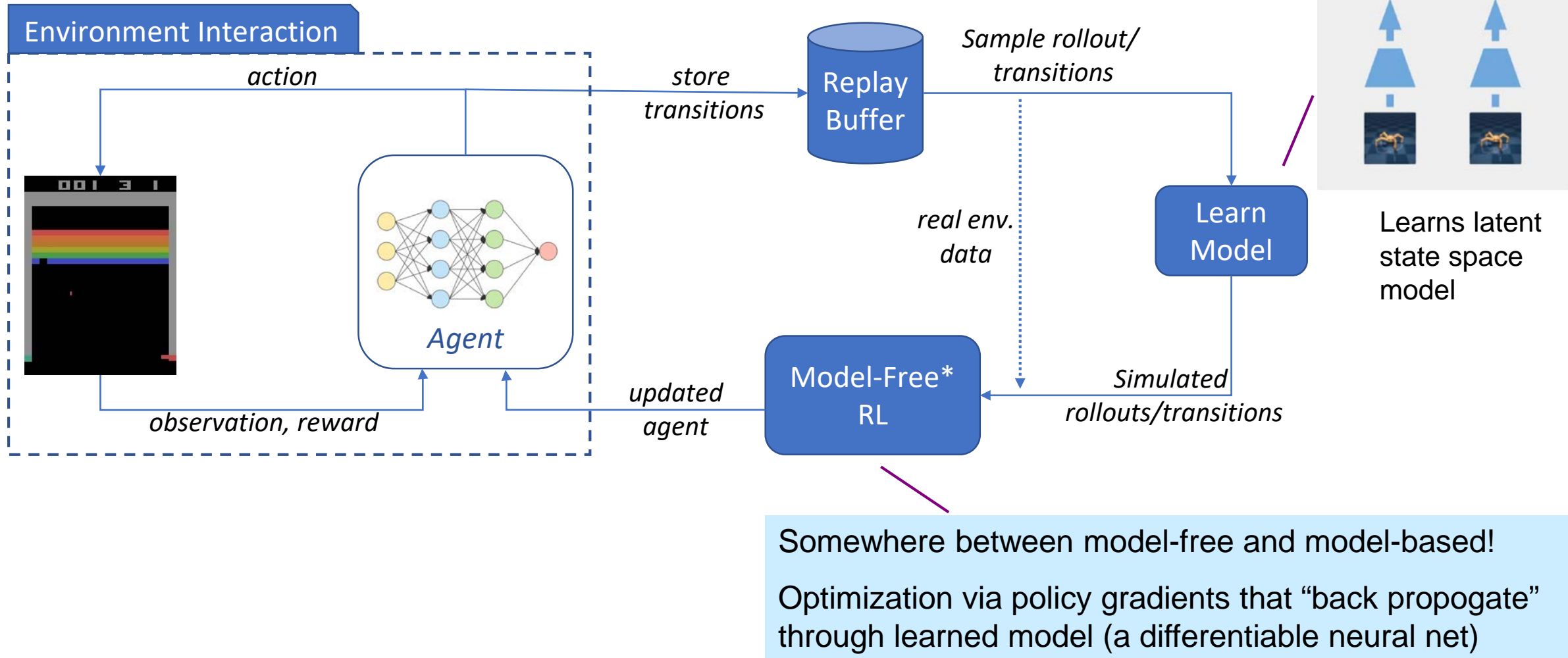
Result: showed improvement over model-free, but fails on really hard games (poor reconstructions)

Proximal Policy Optimization (PPO)

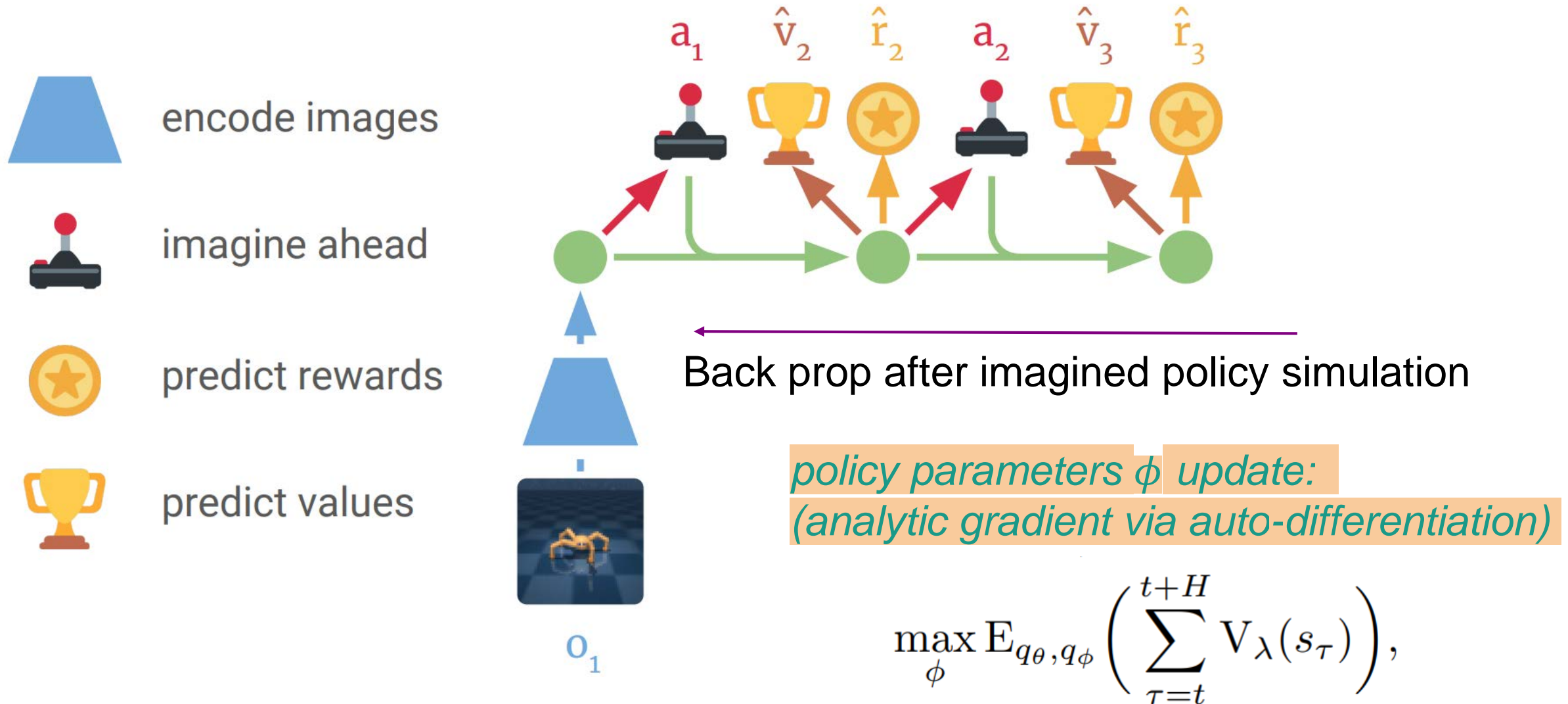


Dream to Control (Dreamer)

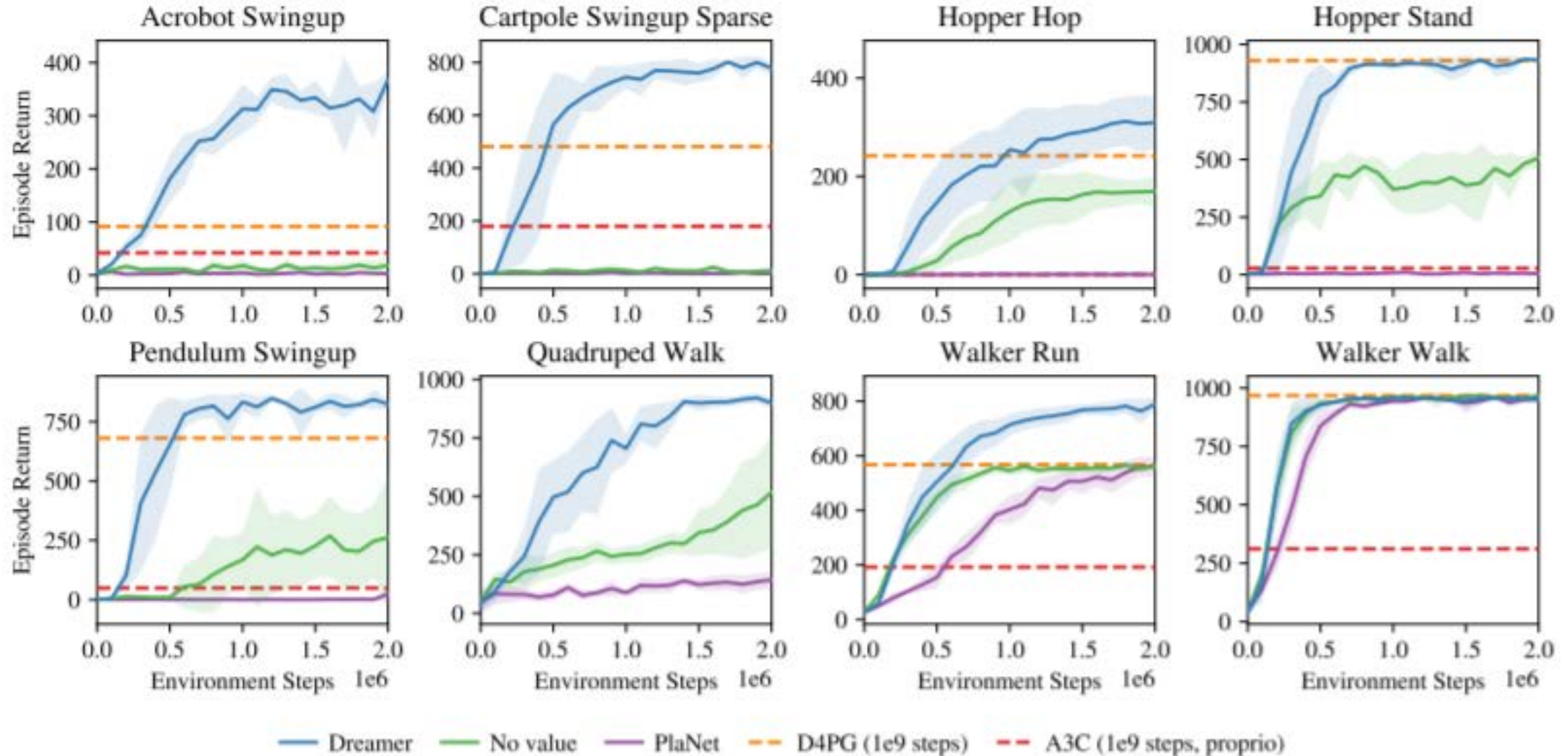
Hafner et al. (2020). Dream to Control: Learning Behaviors by Latent Imagination. *International Conference on Learning Representations*.



Policy Optimization – Not Quite Model-Free



Dreamer Results: Focused on Continuous Control (a current state-of-the-art approach)



Outline

Intro to Model-Based Reinforcement Learning (MBRL)

- Brief Introduction to Reinforcement Learning
- MBRL Choices – Types of Models
- MBRL Choices – Types of Planners

Class of Algorithms

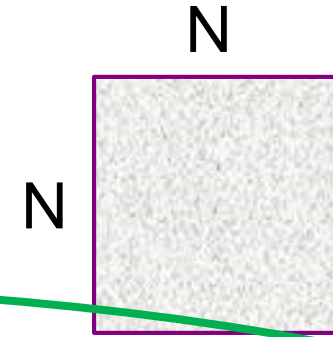
- Class 1: Tabular Models with Optimal Planning
- Class 2: Simulation Models with Search-Based Planning
- Class 3: Simulation Models for Model-Free “Planning”

Research Directions from Planning Perspective

Model Choices: Representation

- **Tabular Models**

- ▶ much RL theory is in this setting
- ▶ challenges for enormous state-action spaces
- ▶ recent integrations with deep learning!



- **Structured/Symbolic Models**

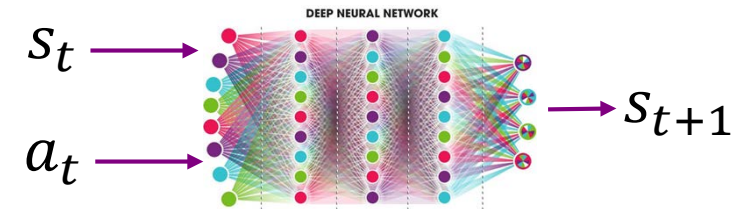
- ▶ learn a model representation in a language that can be “reasoned about” by planner
- ▶ E.g. PDDL/STRIPS, PPDDL, RDDL,
- ▶ What if language isn’t well-matched to world?
- ▶ Where do symbols come from?

PICK-UP(A,B)

PRE: clear(A), on(A,B)
ADD: holding(A), clear(B)
DEL: on(A,B), clear(A)

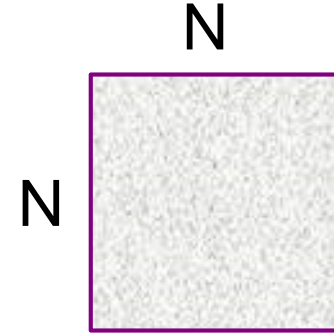
- **Black Box Simulators**

- ▶ learn a black-box function that can simulate transitions and rewards
- ▶ E.g. represent via neural network
- ▶ the most common approach today

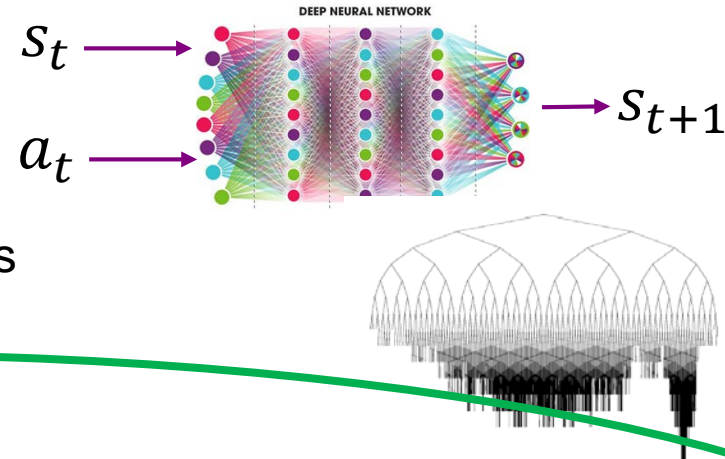


MBRL Planner Types

- **(Near) Optimal Tabular Planning**
 - ▶ E.g. value iteration
 - ▶ basis for much of RL theory
 - ▶ recent uses w/ deep latent representations



- **Monte-Carlo Planning**
 - ▶ only requires black-box simulator
 - ▶ E.g. Monte-Carlo Tree Search (MCTS)
 - ▶ effectiveness can be limited in some cases where RL is hard (e.g. sparse rewards)



- **Symbolic Planning**
 - ▶ requires structured/symbolic model representation
 - ▶ Sadly, so far there has been very little MBRL work in this direction
 - ▶ **Happily, this may be an opportunity!**

PICK-UP(A,B)

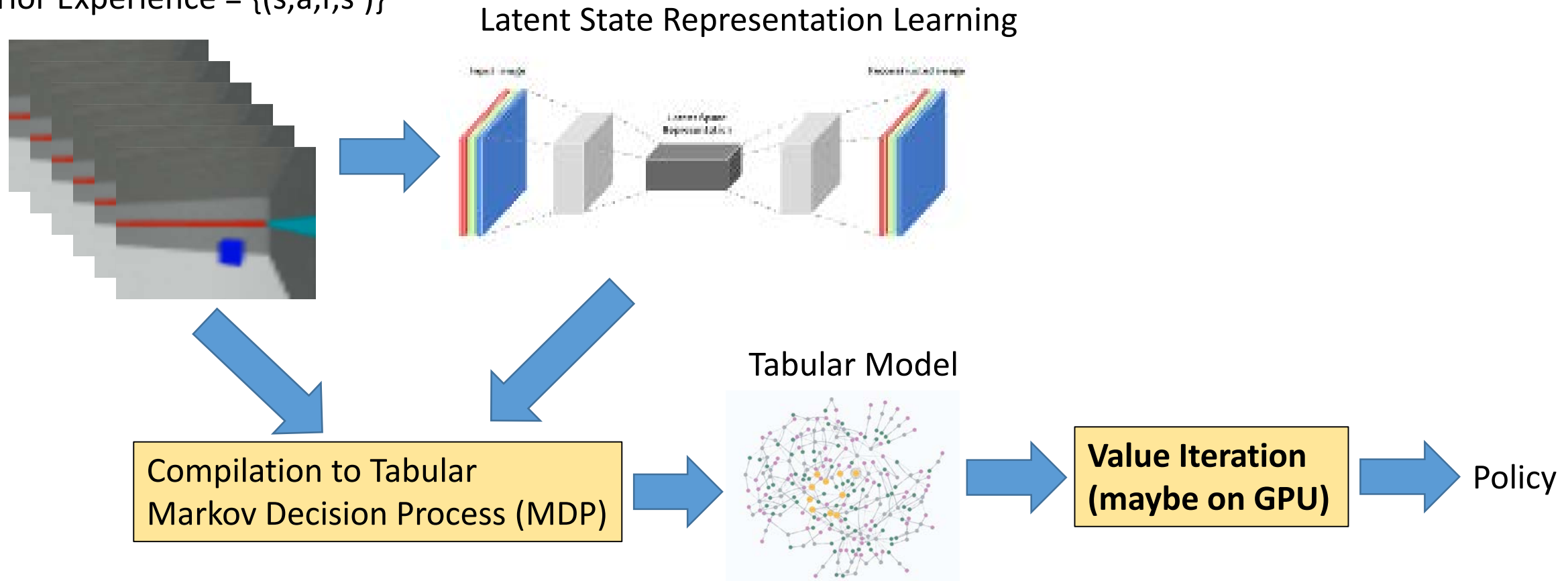
PRE: clear(A), on(A,B)

ADD: holding(A), clear(B)

DEL: on(A,B), clear(A)

Tabular MDPs are Trivially Symbolic – Move Beyond This

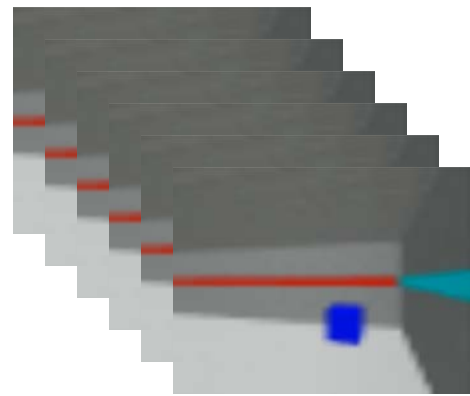
Prior Experience = $\{(s,a,r,s')\}$



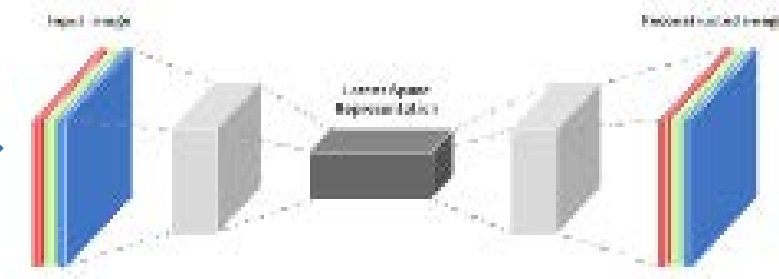
- Maps new observations to nearest (or k nearest) tabular states
- “nearest” is based on learned latent representation

Latent Space SAT Encodings?

Prior Experience = $\{(s,a,r,s')\}$



Latent State Representation Learning



Compilation to Satisfiability Problem

Sat Problem



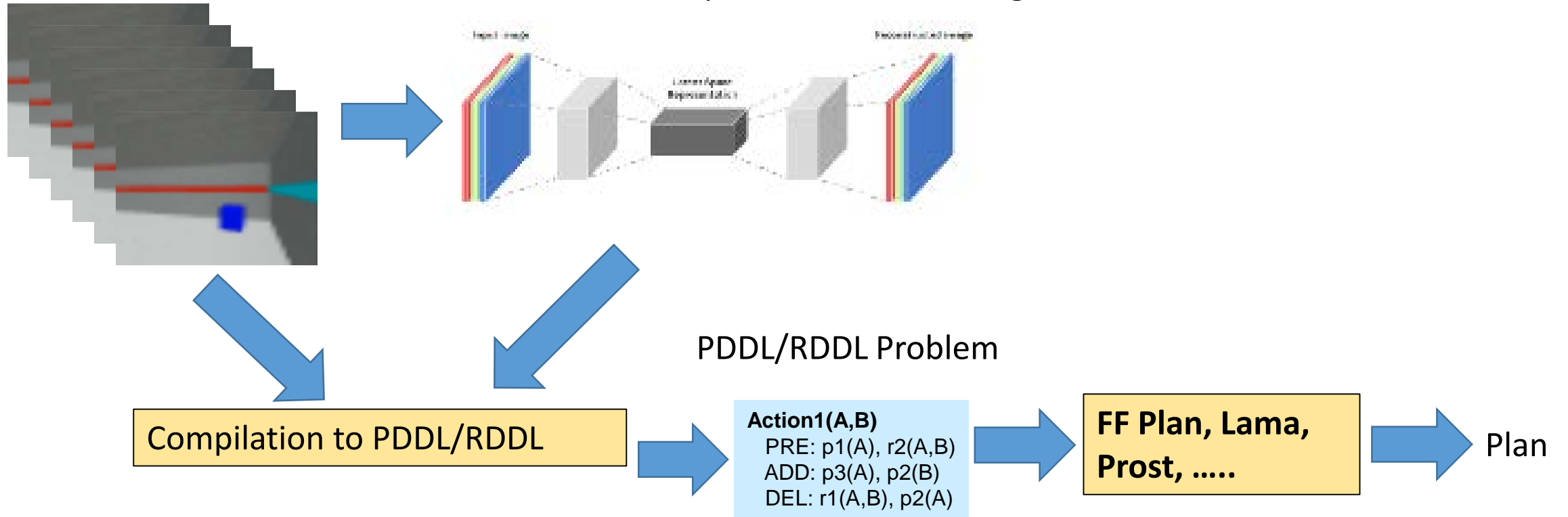
SAT Solver

Plan

Latent Space PDDL/RDDL Encodings?

Prior Experience = $\{(s,a,r,s')\}$

Latent State Representation Learning



Leveraging Symbolic Structure/Solvers for Deep MBRL

Where can we demonstrate the value of integrating ICAPS style planning and Deep Representation Learning?

Hope to see Ph.D. Theses coming out on this soon!

Questions?