ROB-GY 6323
reinforcement learning and optimal control for robotics

Lecture 13
Playing Go with Monte-Carlo Tree Search

Course material

All necessary material will be posted on Brightspace
Code will be posted on the Github site of the class

https://github.com/righetti/optlearningcontrol

Discussions/Forum with Slack

Contact

ludovic.righetti@nyu.edu
Office hours in person
Wednesday 3pm to 4pm
370 Jay street - room 801

Course Assistant

Armand Jordana
aj2988@nyu.edu
Office hours Monday 1pm to 2pm
Rogers Hall 515

any other time by appointment only

Schedule

Week	Lecture		Homework	Project
1	Intro	Lecture 1: introduction		
2	Trajectory optimization	Lecture 2: Basics of optimization	HW 1	
3		Lecture 3: QPs		
4		Lecture 4: Nonlinear optimal control		
5		Lecture 5: Model-predictive control		
6		Lecture 6: Sampling-based optimal control	HW 2	
7	Policy optimization	Lecture 7: Bellman's principle		
8		Lecture 8: Value iteration / policy iteration		Project 1
9		Lecture 9: Q-learning	HW 3	
10		Lecture 10: Deep Q learning		
11		Lecture 11: Actor-critic algorithms		
12		Lecture 12: Learning by demonstration	HW 4	Project 2
13		Lecture 13: Monte-Carlo Tree Search		
14		Lecture 14: Beyond the class		
15	Finals week			

HW4 is due December 8th

Project 2 is due December 19th

Paper report (due December 19th - no deadline extension)

Goal: read one scientific paper and understand it

Pick one paper from the list of papers posted on brightspace

Report (maximum 2 pages - IEEE format double column)
It should contain 3 sections:

1. A section that summarizes the paper:
   What was done? How was it done?
   Why was it worth doing? What are the results?
2. A section explaining how the paper relates to the algorithms seen in class.
   Which algorithms? What is different?
3. A section containing a critical discussion on the paper: pros and cons.
   What seems to work and what convinces you about the result What are the issues/limitations? What could be done better? What should be done next?

Do not copy equations or figures from the paper - keep your explanations to the point

Imitation learning

Learning optimal policies from demonstration

Behavioral cloning: learning <u>policies</u> from demonstrations

Provide a lot of demonstrations and learn a policy from it

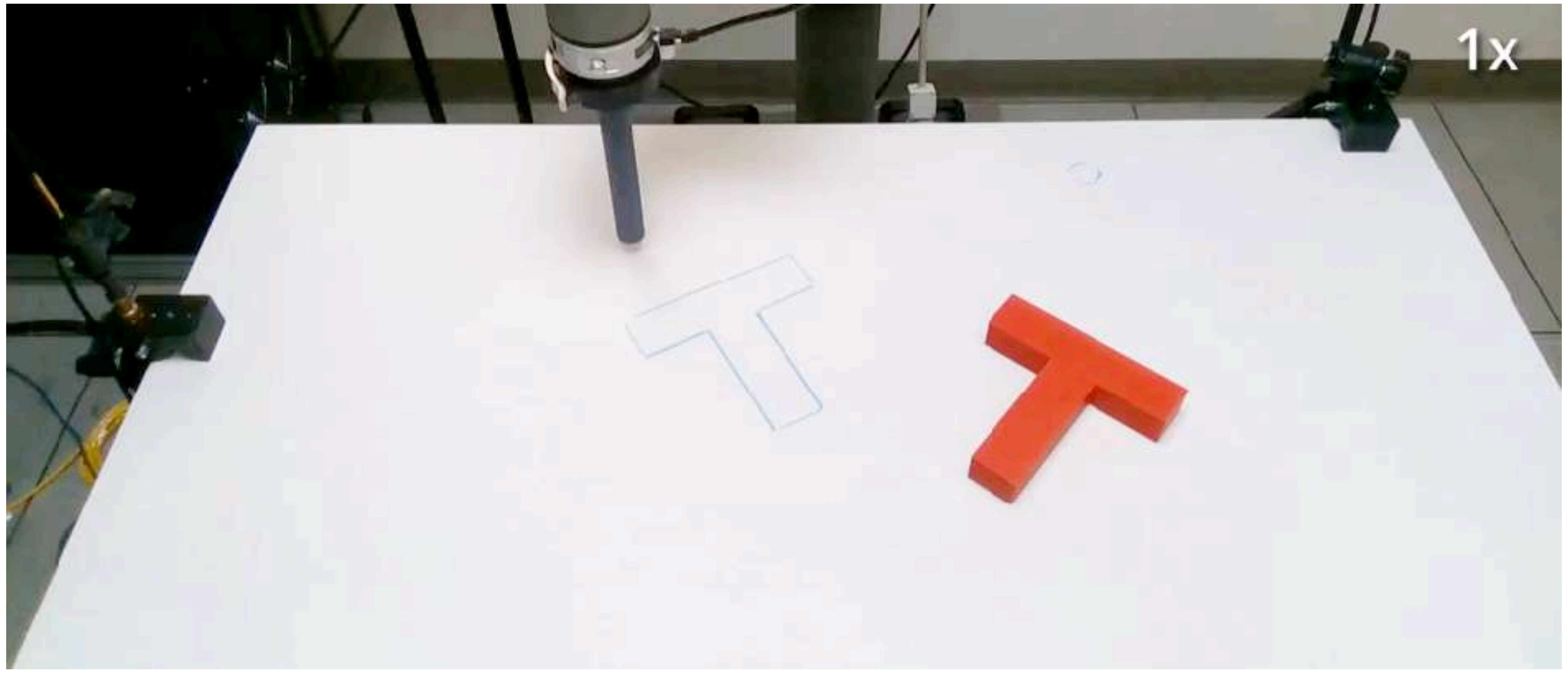
Input: a dataset of demonstrations $(x_0, u_0, x_1, u_1, \cdots, x_N, u_N)$

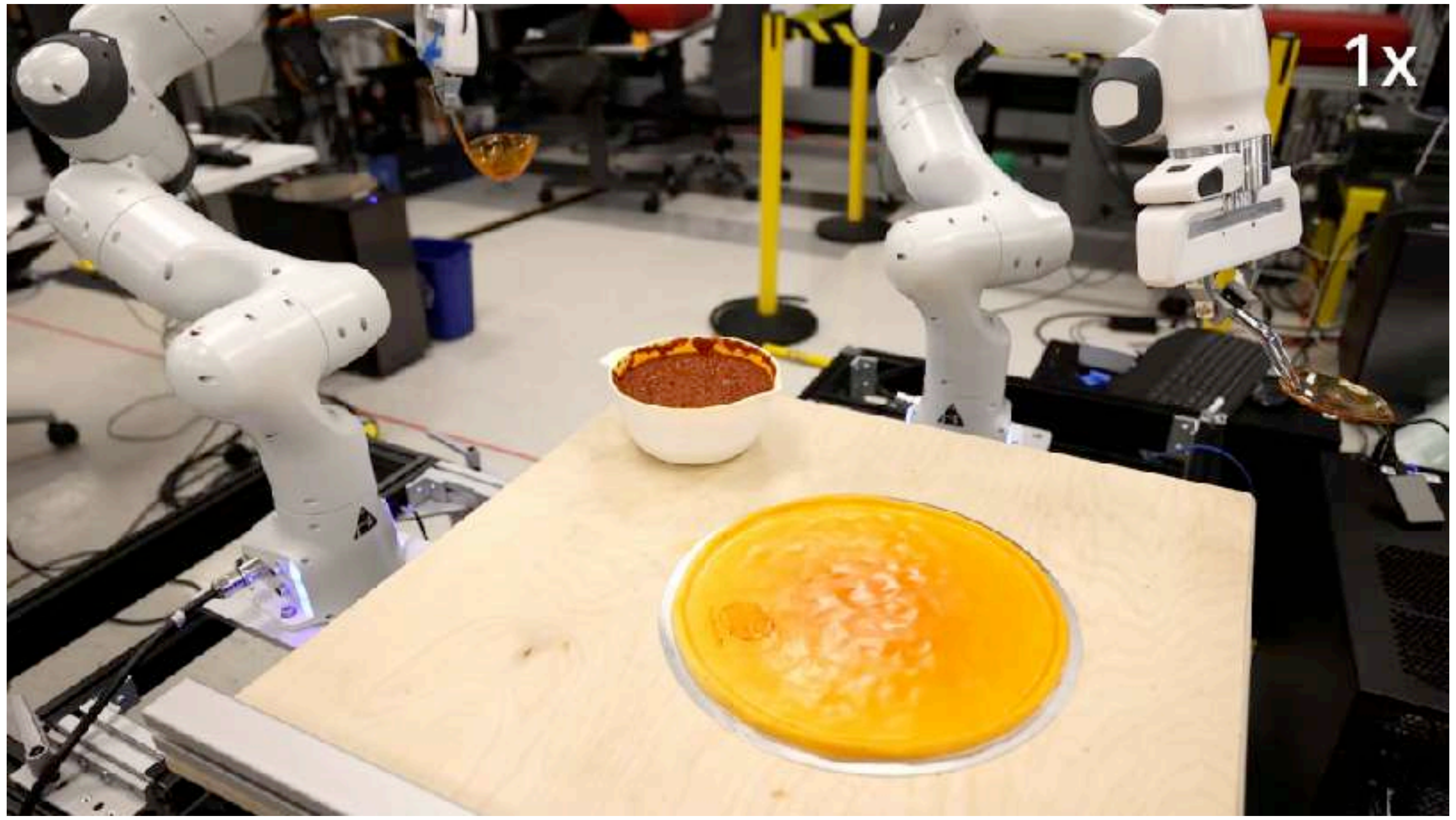
Output: a policy $u_n = \pi(x_n)$

A supervised learning problem!

$$\min_\theta \sum_N (\pi_\theta(x_n) - u_n)^2$$

Can learn very complex behaviors

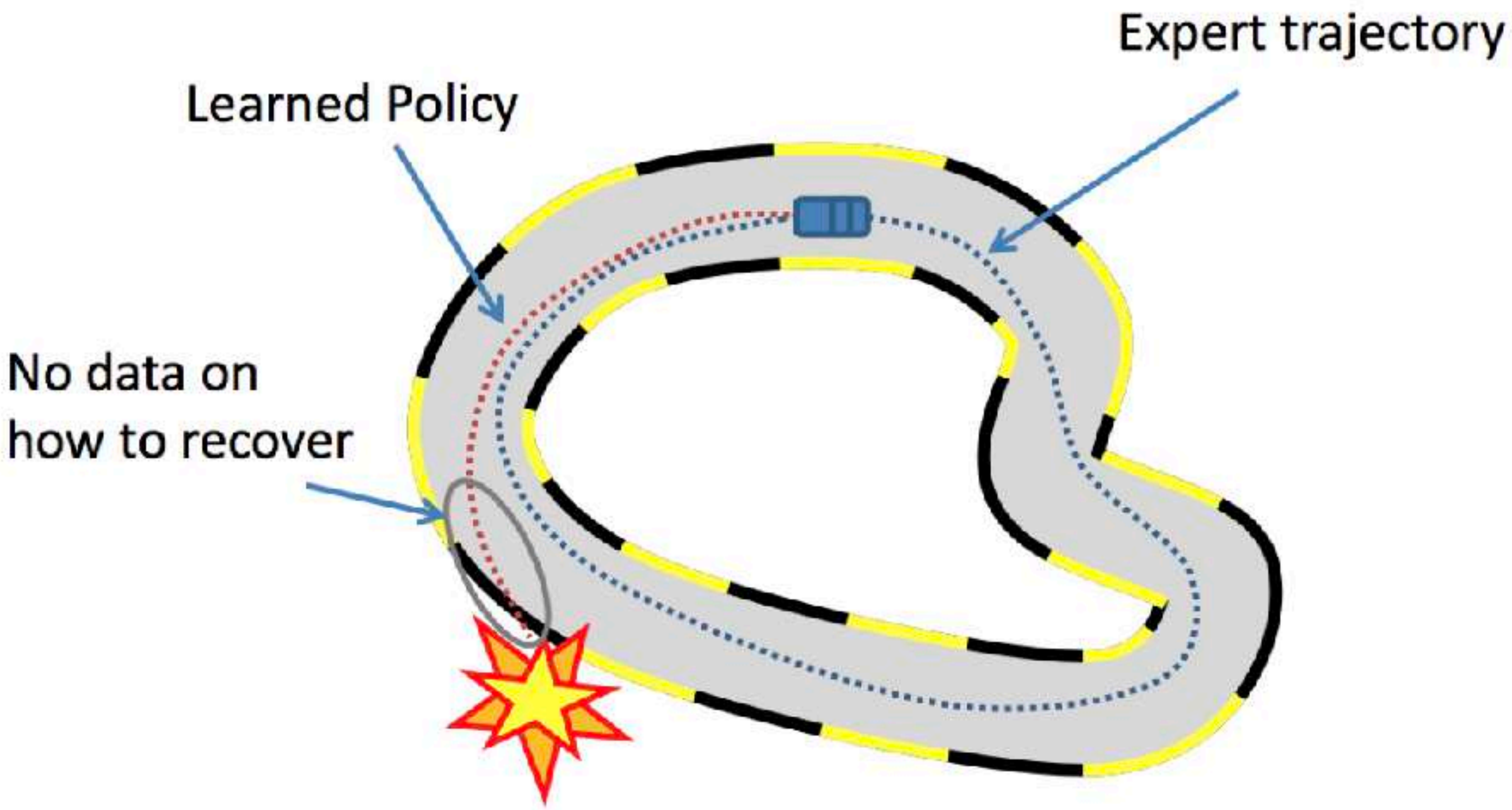




[Chi et al. 2024]

Behavioral cloning: learning <u>policies</u> from demonstrations

Problem: compounding errors leads the robot out of demonstration distribution



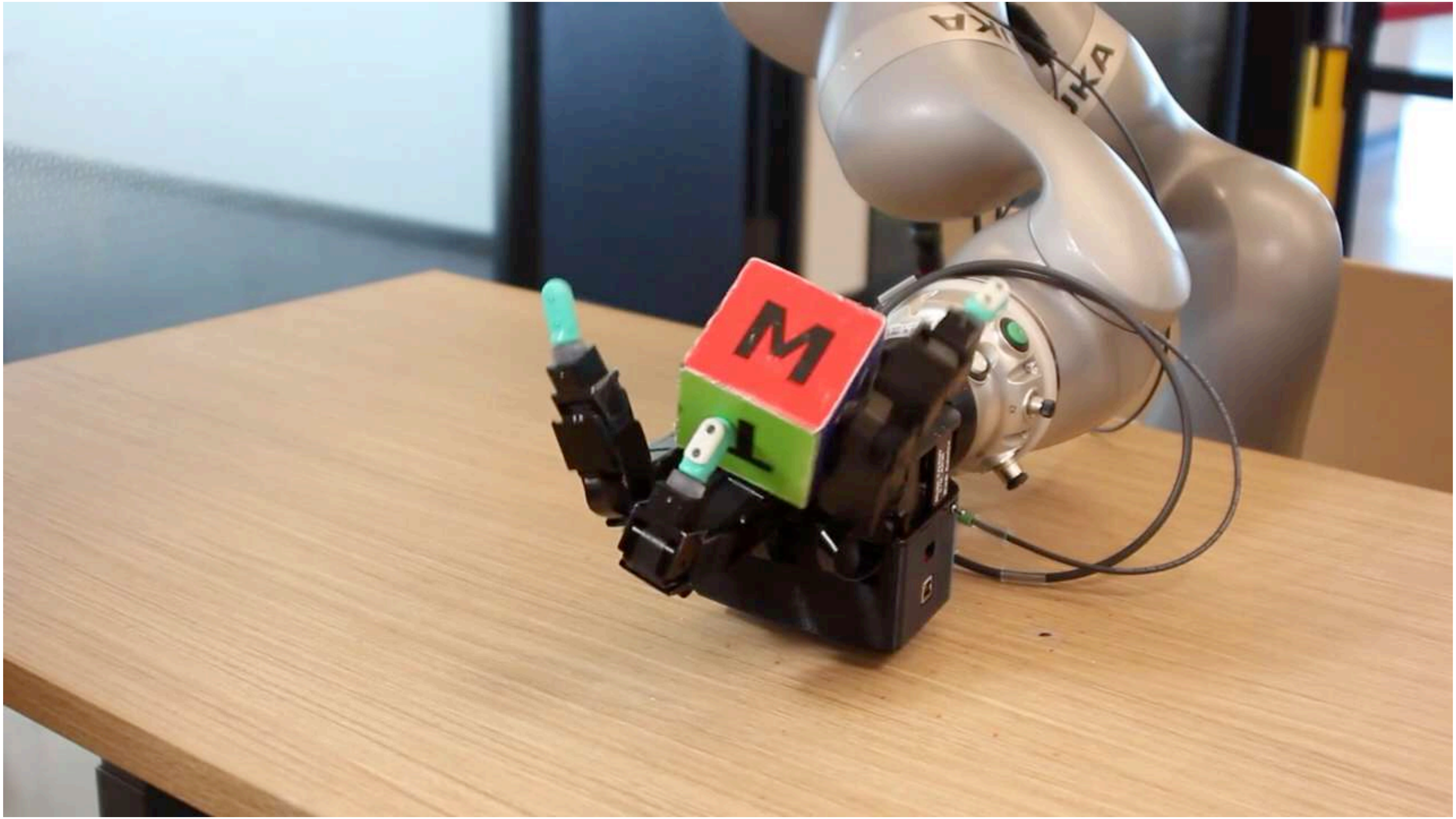
Dataset aggregation DAGGER

Idea: as the robot does into "unseen territory" collect data and ask an "expert" to provide the correct control (In effect we relabel the data the robot is collecting)

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.

[Ross et al. 2011]



[Handa et al. 2022]

Learning cost functions from demonstrations

Inverse RL and apprenticeship learning

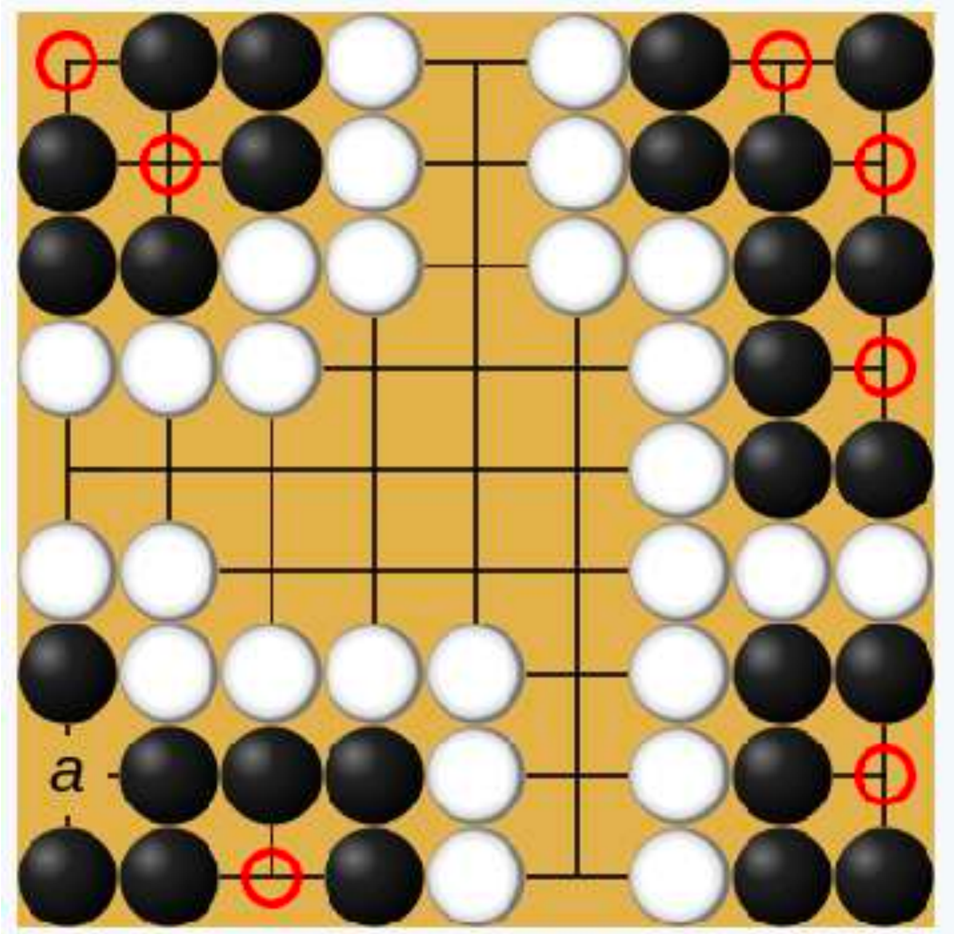
Inverse RL / inverse OC

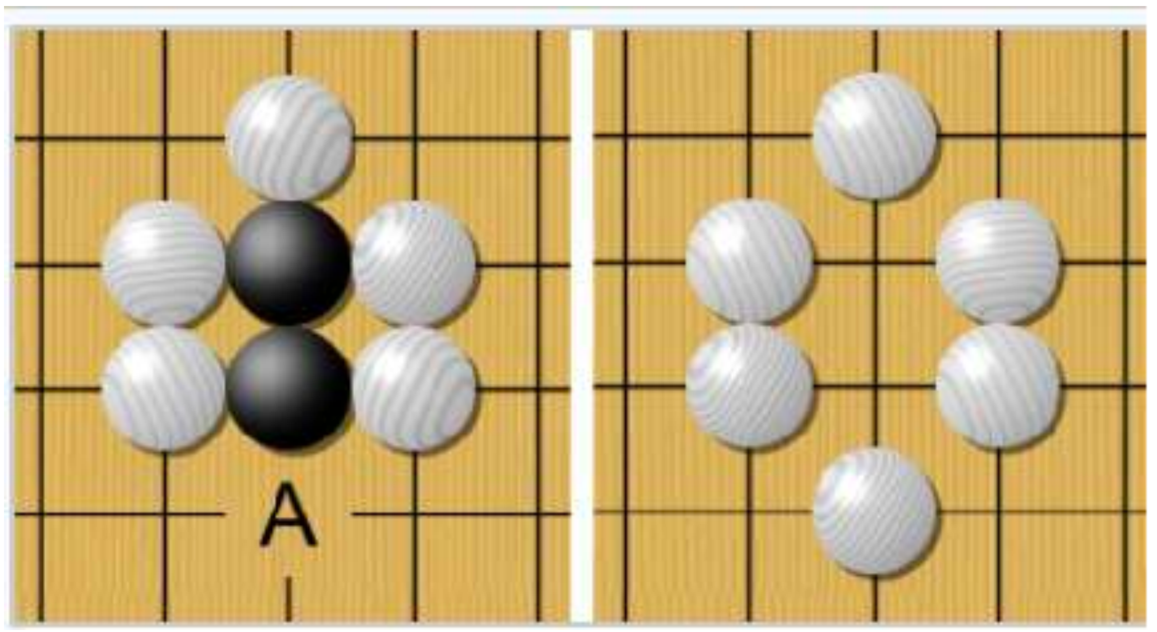
Can we infer the cost function from a demonstration?

Useful for:

- Learning from demonstrations
- Apprenticeship learning
- Transferring skills across robots
- Also… analyzing human behavior

Playing the game of Go
A mixture of imitation learning, OC and RL


a


A

Deciding how to play with tree search

Go branching factor

For a game typically b^d number of moves to test
b is "breadth", i.e. number of legal moves at each turn
d is the "depth", i.e. the game length

For Go b~250 and d~150

Speeding up search: Monte-Carlo Tree Search

Invented by R. Coulom in 2006 (not new!)

4 steps to be repeated N times

1. Selection
2. Expansion
3. Simulation
4. Backup

Exploration vs. Exploitation (UCT)

AlphaGo 2016

Reduce the "breadth" and the "depth" of the search using MCTS
In addition, improve the sampling efficiency by:

- Learning a policy
- Learning a value function

Defining the states

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

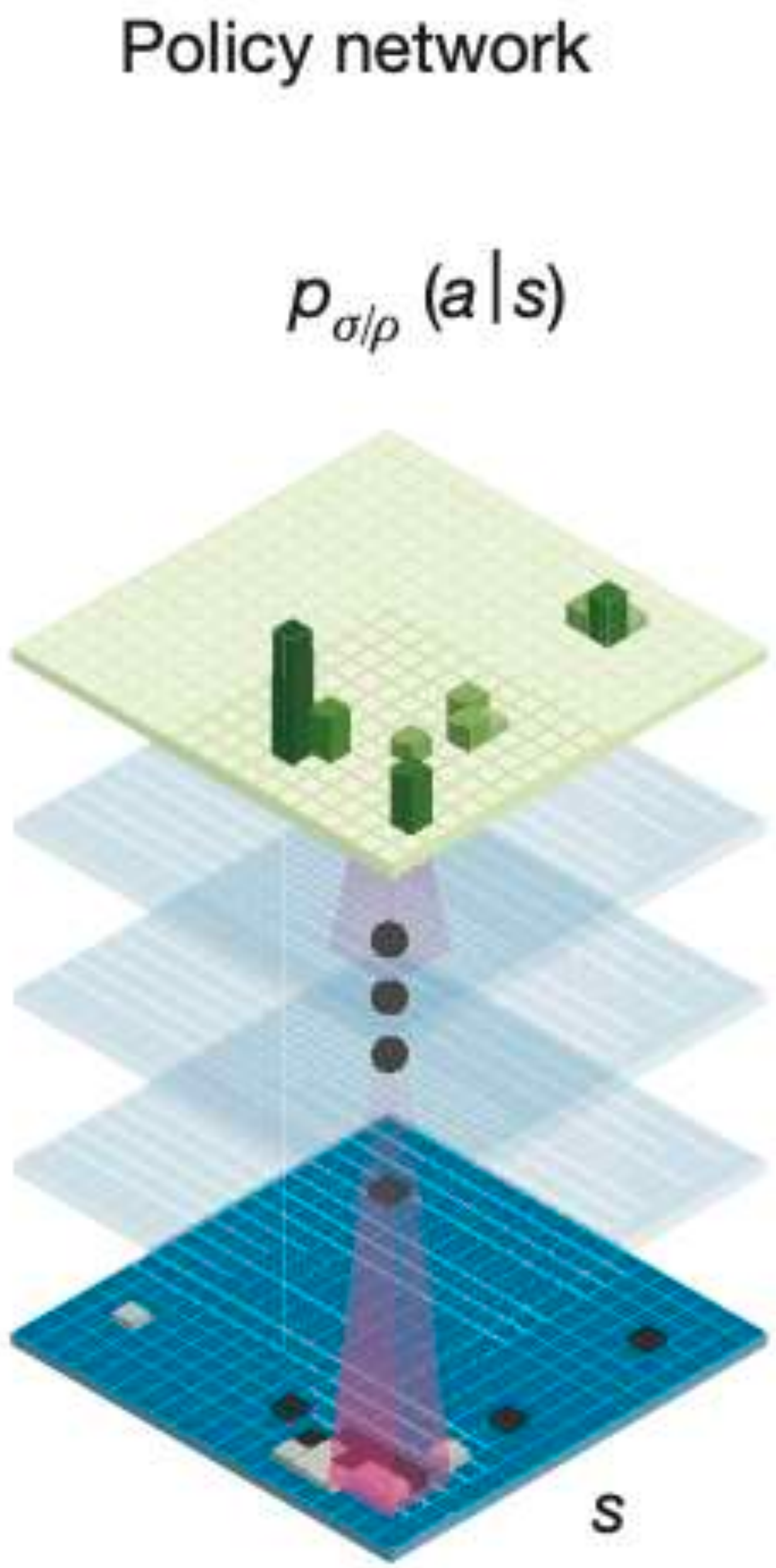
Step 1: Use supervised learning using human plays to learn a policy

Step 2: Use policy gradients to improve policy (using self-play)

Step 3: Use RL to compute value function of policy

Step 4: Monte-Carlo Tree Search using previously learned policy and value function to direct exploration

Stage 1: learn a policy from Human players



$p_\sigma(a|s)$

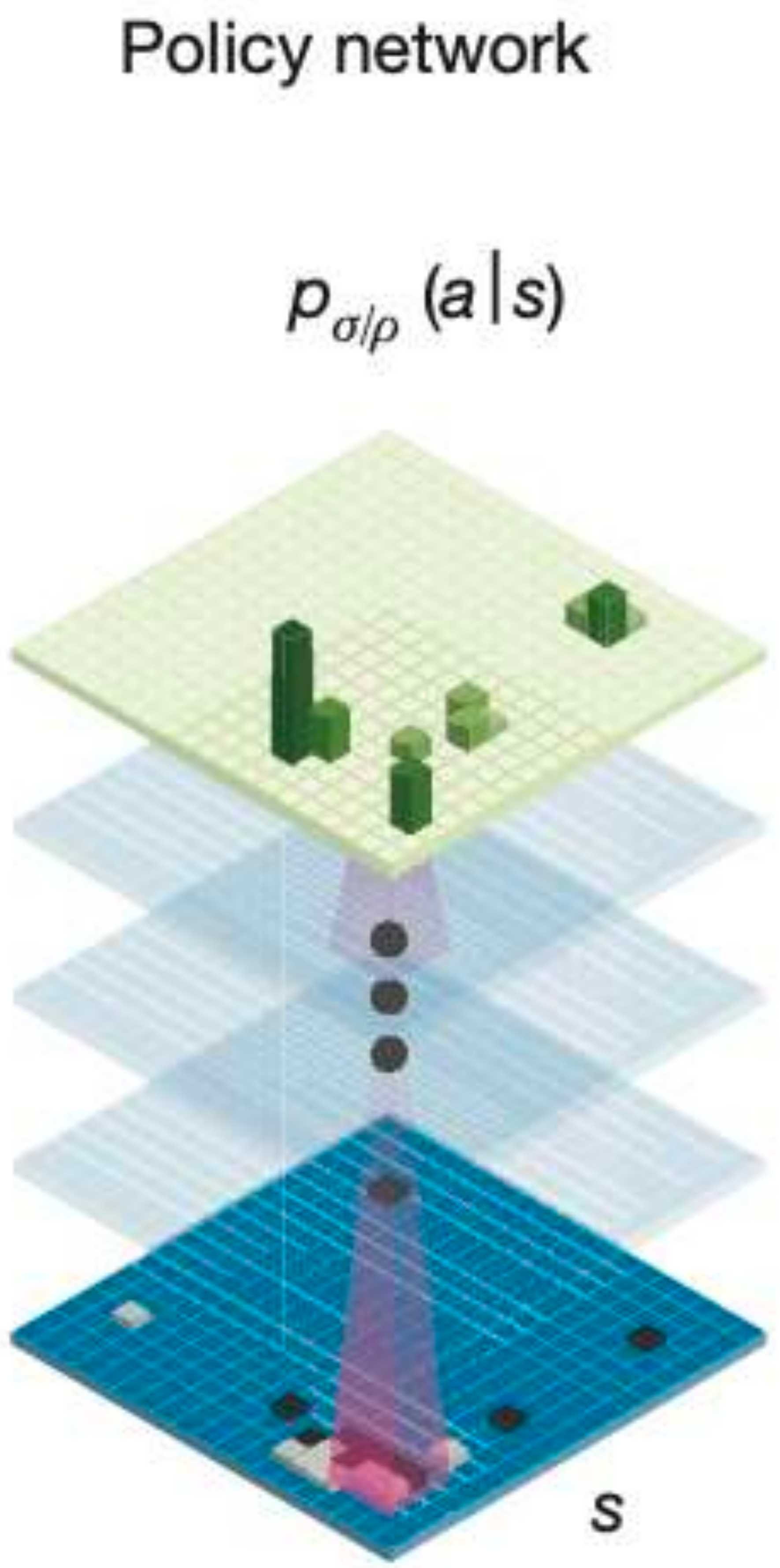
Policy learned with supervised learning SL-policy

13 layers neural network - accurate (57% / 55%) but slow to evaluate (3ms)

$p_\pi(a|s)$

Policy with smaller network - less accurate (24%) but fast to evaluate (2us)

Stage 2: improve policy using RL policy gradient



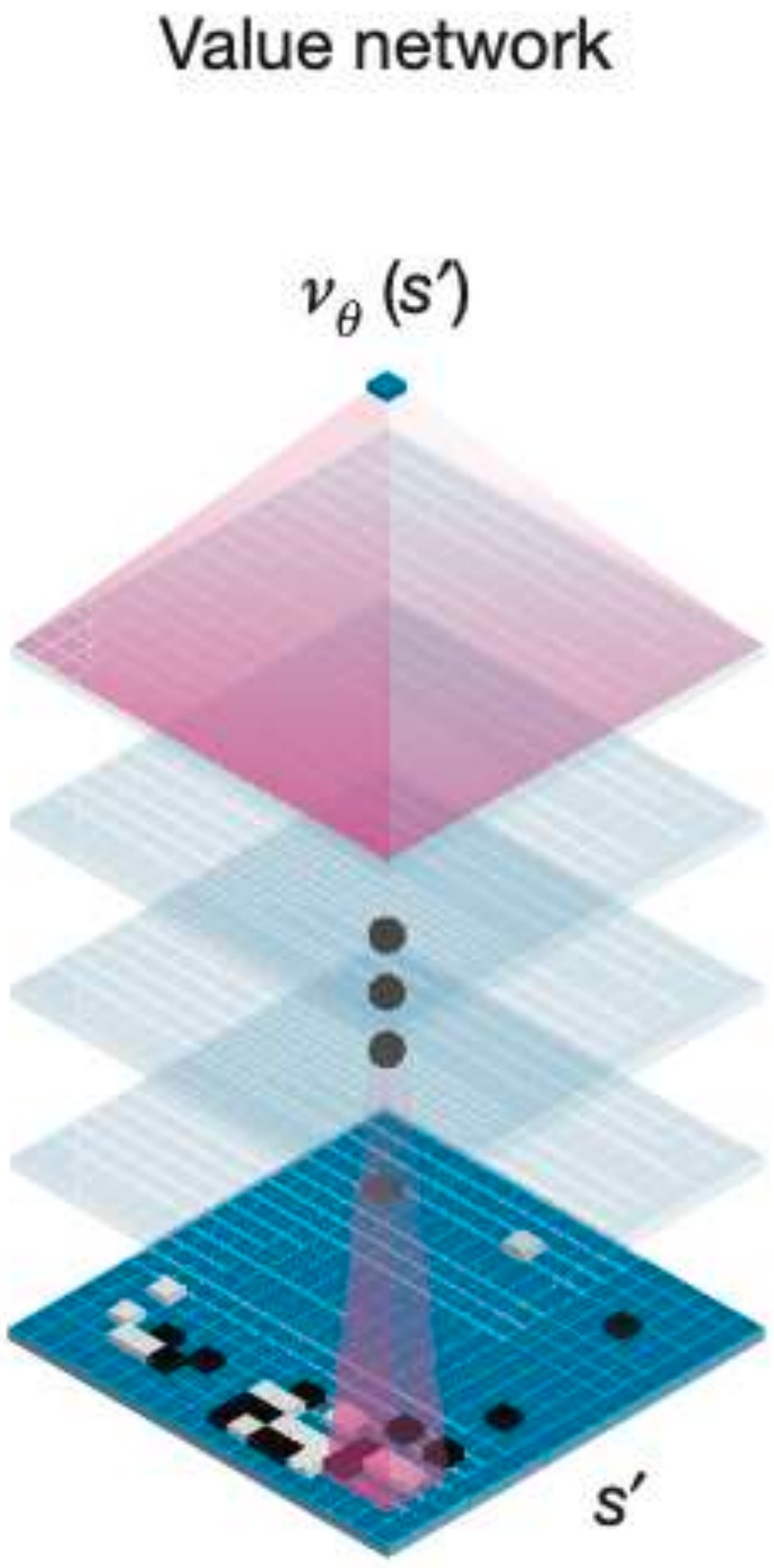
p_ρ Policy learned using RL

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

$$z_t = \pm r(s_T)$$

RL policy won 80% games agains SL policy

Stage 3: learning a value function



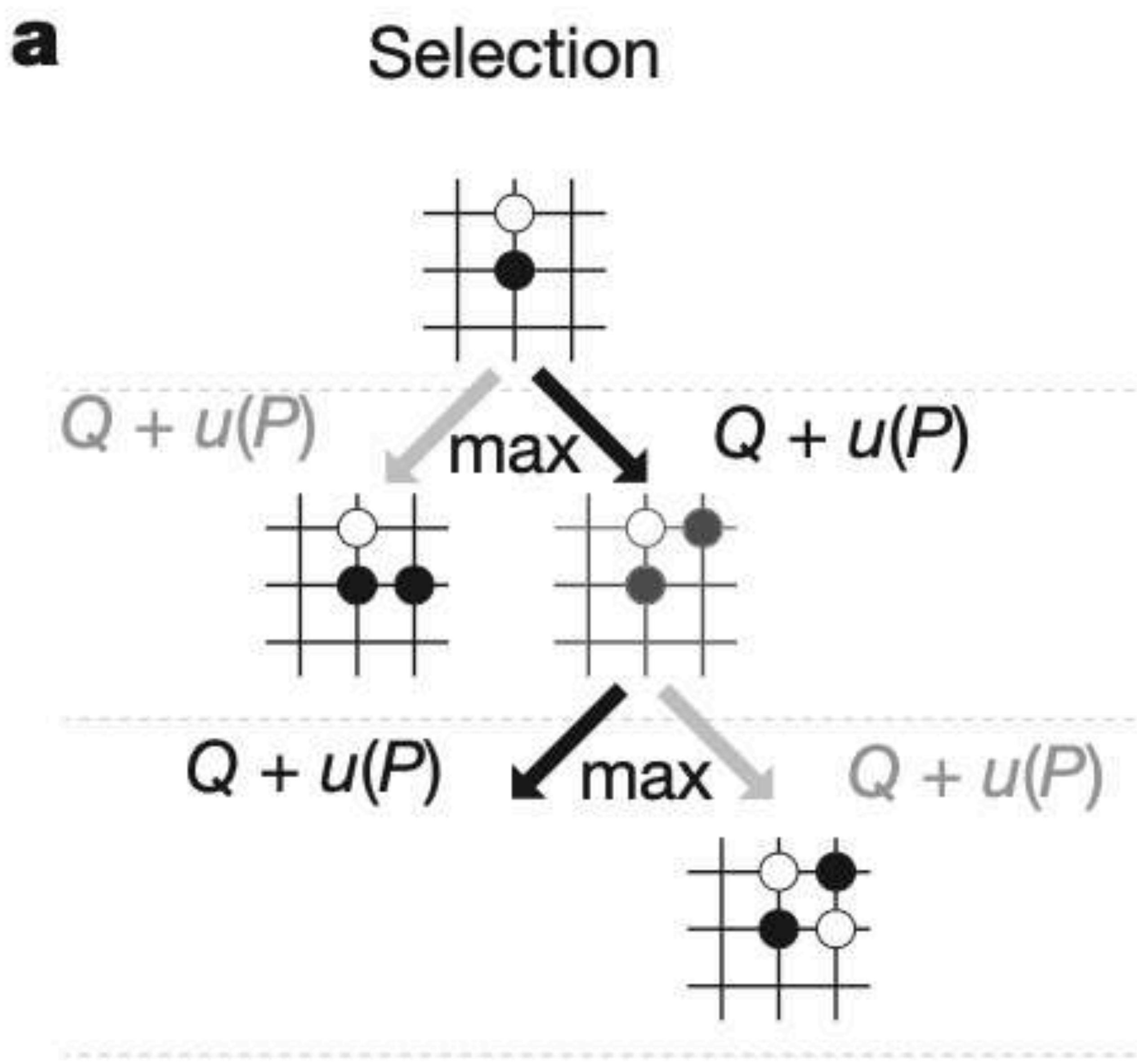
Self-play and randomization

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t\ldots T} \sim p]$$

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))$$

Stage 4: (modified) Monte-Carlo Tree Search

Each edge of the tree (action/state pair) stores an action value Q(s,a), visit count N(s,a) and prior probability P(s,a)

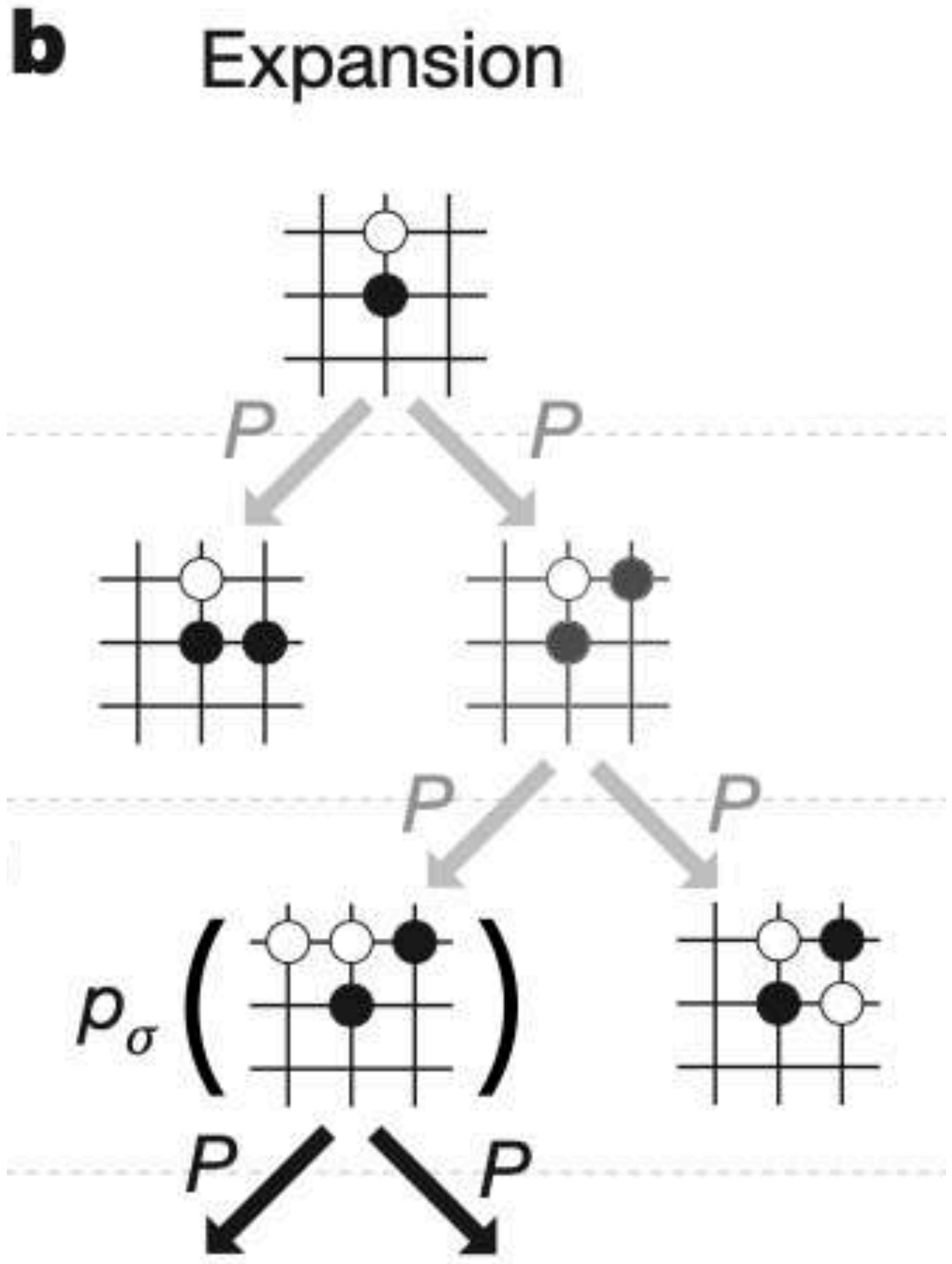


Go down the (partial) tree using

$$a_t = \operatorname{argmax}(Q(s_t, a) + u(s_t, a))$$

$$u(s,a) \propto \frac{P(s,a)}{1 + N(s,a)}$$

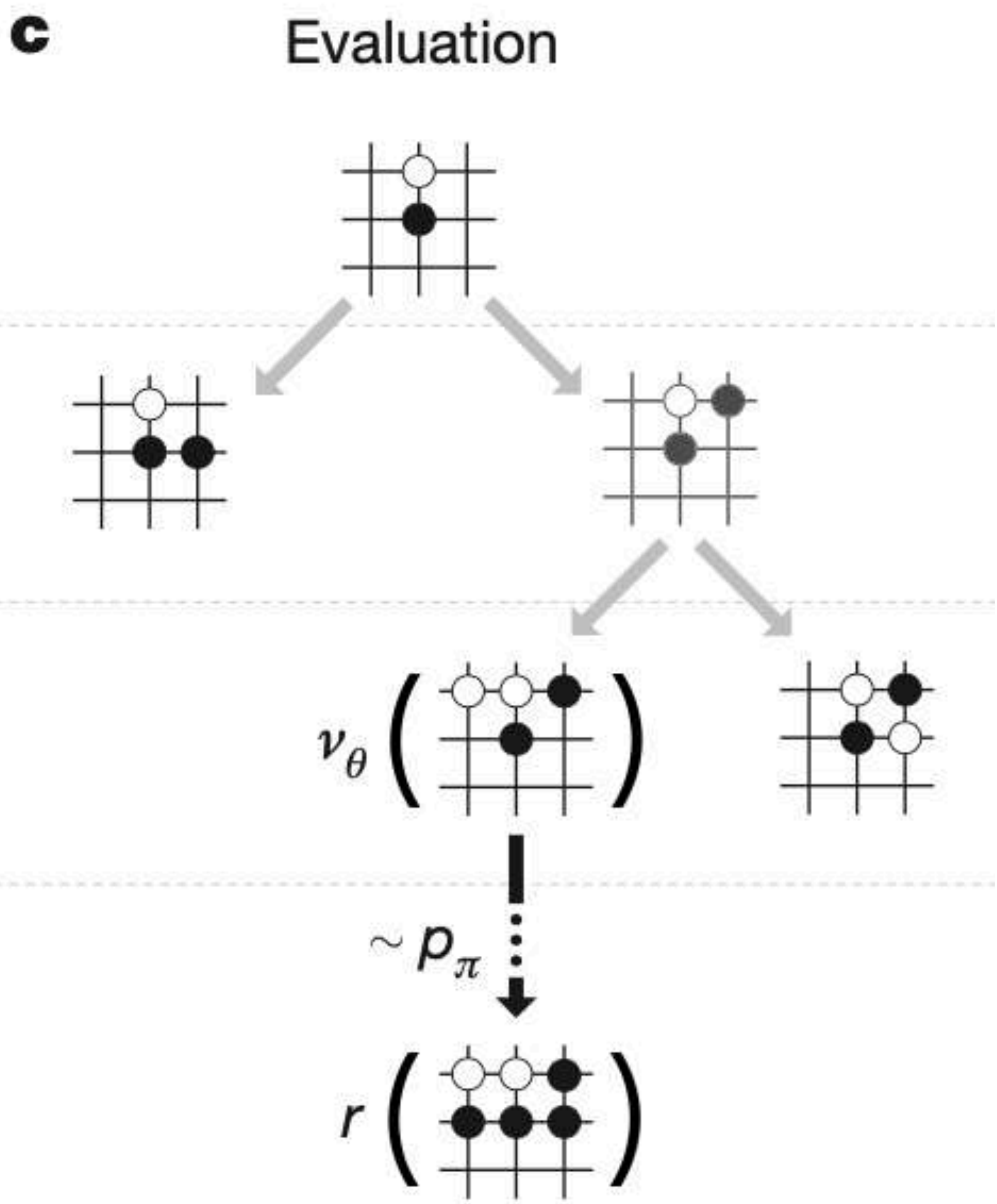
Stage 4: (modified) Monte-Carlo Tree Search



When a leaf is reached it can be expanded using the SL policy network

$$P(s,a) = p_\sigma(a|s)$$

Stage 4: (modified) Monte-Carlo Tree Search

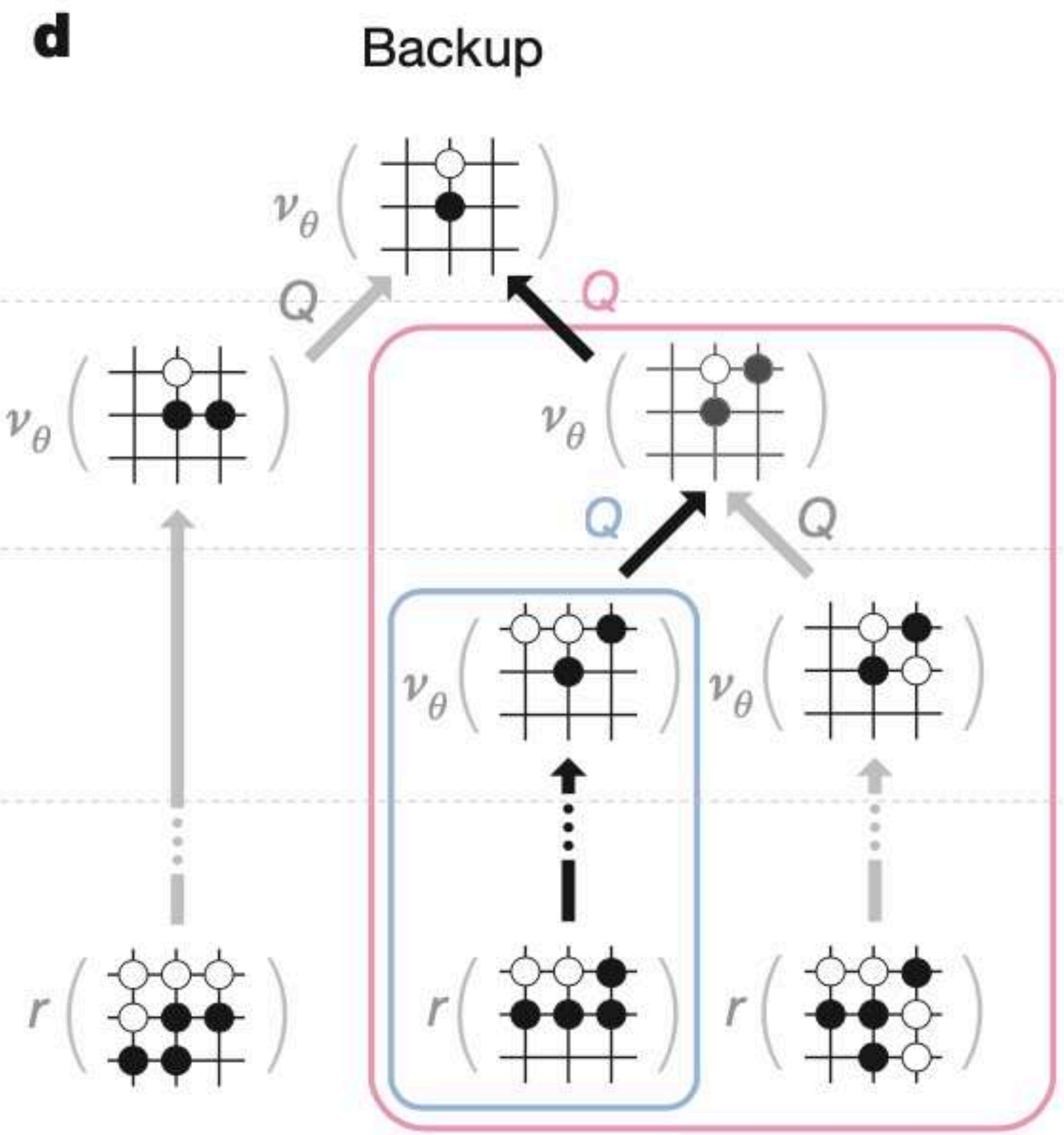


Evaluate the leaf node V(s) using:

1. The learned value function $v^p(s)$
2. The outcome of a random simulated "play"

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

Stage 4: (modified) Monte-Carlo Tree Search



$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

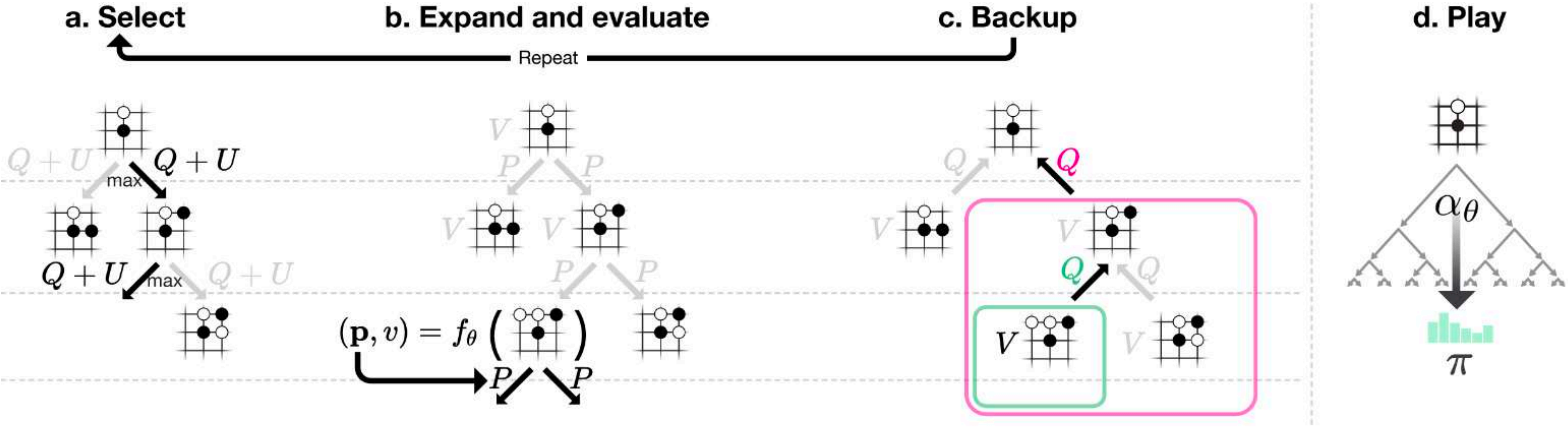
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

AlphaGoZero (2017)

Use self-play to learn a policy and value function (no SL)
Input features are only black and white stones (no other features)
Single NN for both policy and value
Simpler tree search - no evaluation through simulated play

AlphaGoZero (2017)

Use self-play to learn a policy and value function (no SL)
Input features are only black and white stones (no other features)
Single NN for both policy and value
Simpler tree search - no evaluation through simulated play



AlphaZero (2017)

Similar to AlphaGoZero but to play also Chess and Shogi

MuZero (2019)

Similar to AlphaGoZero but also learns the game model