上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 33: Policy Gradient Methods

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

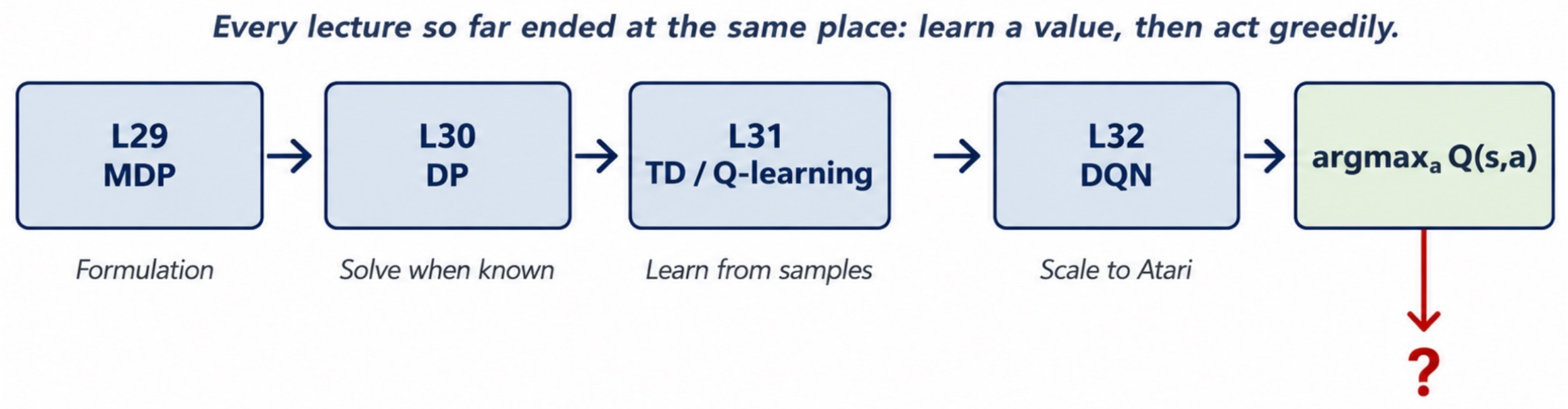
https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

Where We Left Off

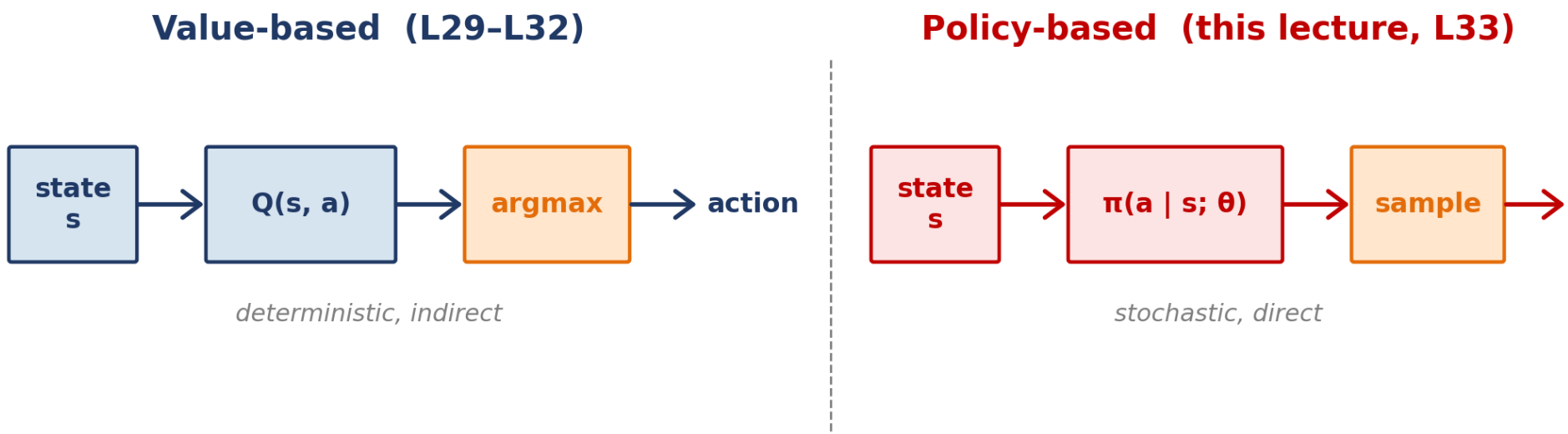
- L29: MDP. L30: DP. L31: TD / Q-learning. L32: DQN scaled it to Atari.



Every lecture so far ended at the same step. Today we branch off.

The Pivot

All value-based methods follow the same recipe:
estimate V or Q → derive policy by argmax.



What if we skip the value, and learn the policy directly? A different family entirely.

From scratch — to anything you can describe

OpenAI Dactyl solving Rubik's cube

Trained with PPO — a policy gradient method. This lecture: how that works, in three steps.

Objectives

By the end of this lecture, you should be able to:

- Explain three problems value-based methods can't solve.
- State the policy gradient theorem and identify the log-likelihood trick.
- Distinguish REINFORCE from actor-critic in one sentence.
- Explain why a baseline reduces variance without introducing bias.
- Sketch the RLHF pipeline and place the policy gradient step in it.

Outline

1 Part

Why we need policy gradient

Three problems value-based can't solve

2 Part

The policy gradient theorem

REINFORCE in one chain-rule step

3 Part

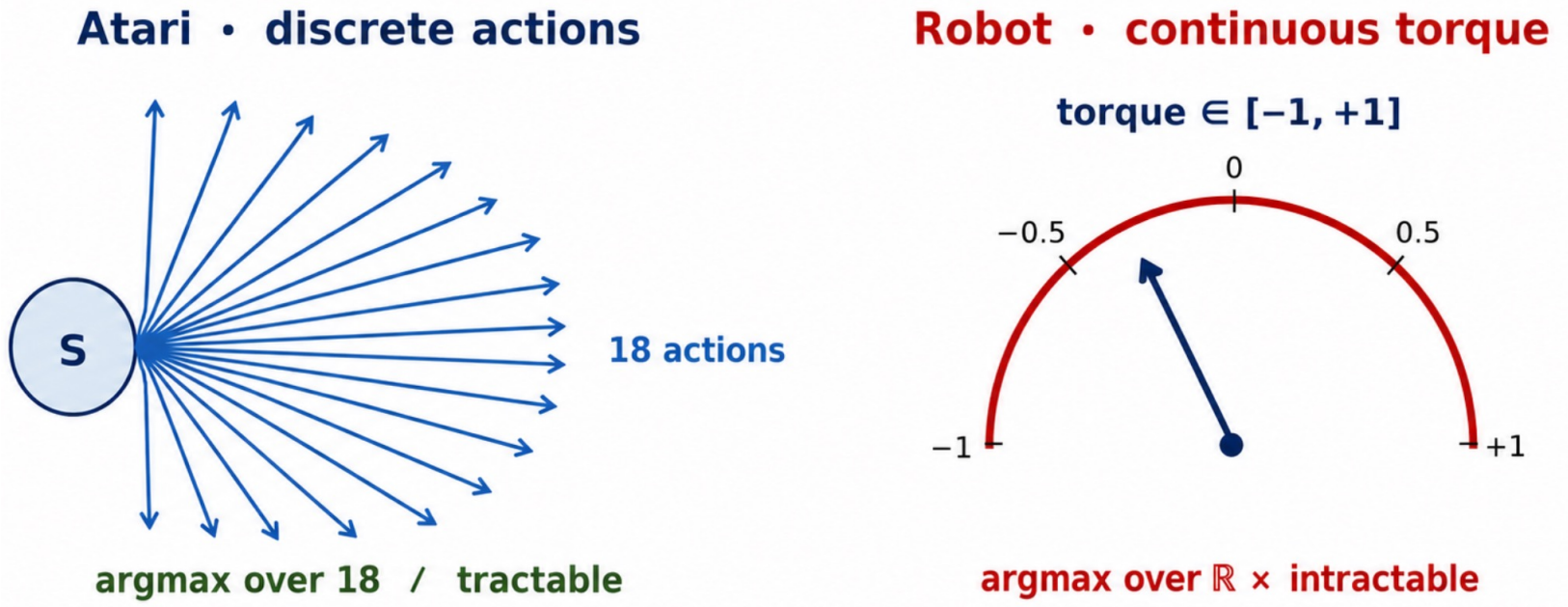
From REINFORCE to ChatGPT

Variance, baselines, PPO, RLHF

1. Why We Need Policy Gradient

Problem 1: Continuous Actions

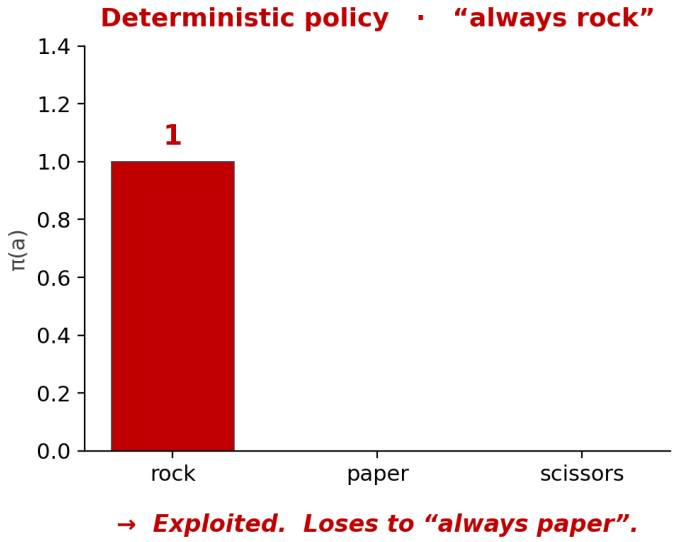
- **Situation:** DQN's policy is argmax_a Q(s, a).
- **Complication:** What if the action is "torque between −1 and +1"? The argmax is over an infinite set.
- **Question:** *How do you max over a continuum?*

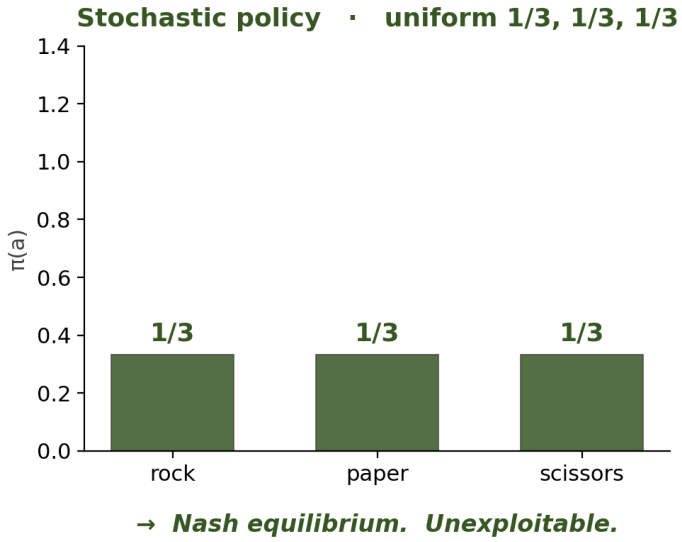


Discretizing to 100 bins per joint × 20 joints = 10^{40} actions. Worse than the table.

Problem 2: Stochastic Optimal Policies

- **Situation:** Value-based methods always extract a deterministic policy.
- **Complication:** In paper–rock–scissors, the optimal strategy is uniform random. A deterministic policy gets exploited.
- **Question:** *How do you learn to be intentionally random?*

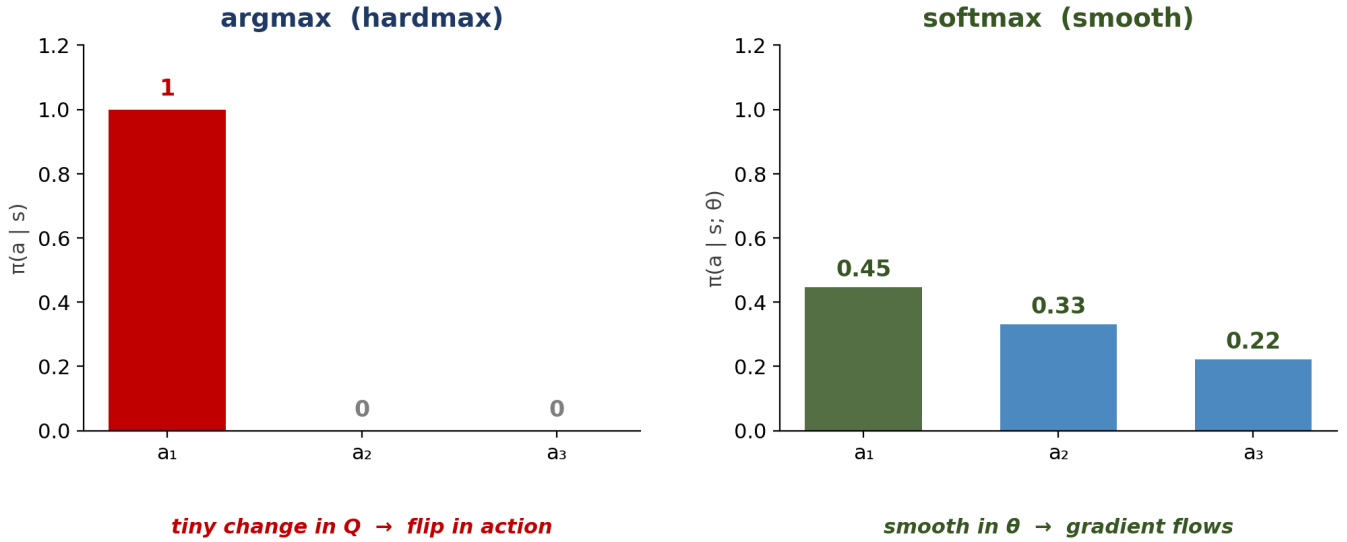




Callback to L29: the deterministic optimal-policy theorem holds for fully-observable single-agent MDPs. Multi-agent or partial-obs breaks it.

Problem 3: Smoothness of the Optimization

- **Situation:** A small change in Q can flip argmax. The induced policy is a step function.
- **Complication:** Step functions are hard to optimize end-to-end — no gradient.
- **Question:** *Can we make the policy a smooth function of θ, so gradients flow?*



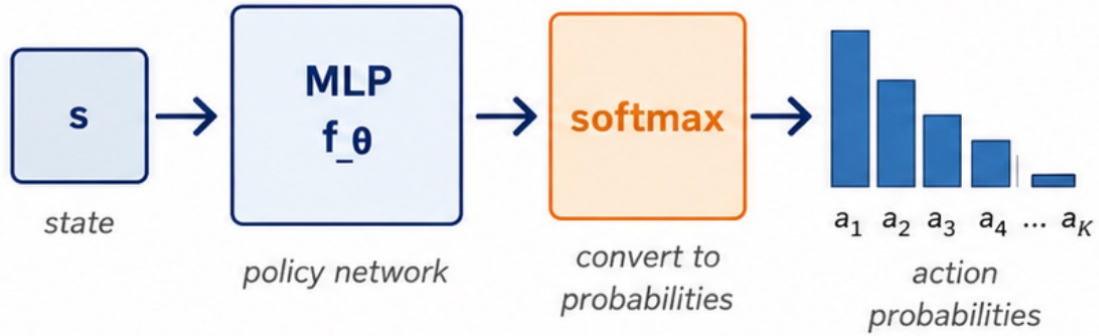
Backpropagation needs differentiable everything. Policy gradient gives us that.

The Answer: Parameterize the Policy

The network outputs a distribution, not a value.

Discrete actions → softmax:

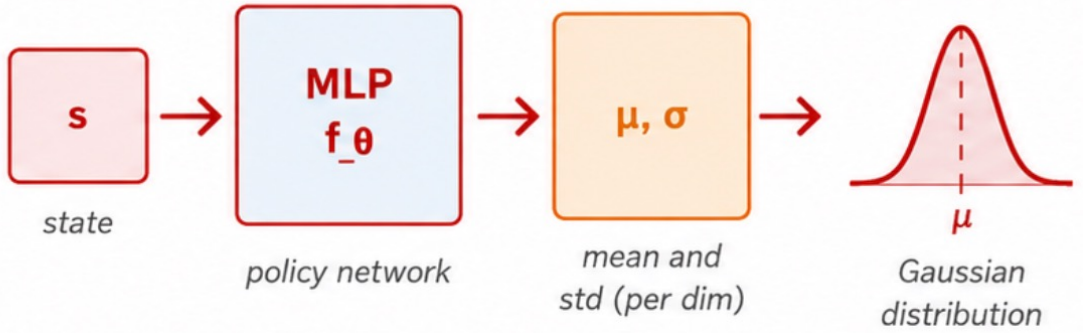
$$\pi_\theta(a \mid s) = \text{softmax}(f_\theta(s))$$



$$\pi(a \mid s; \theta) = \text{softmax}(f_\theta(s))$$

Continuous actions → Gaussian:

$$\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$$



$$\pi(a \mid s; \theta) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$$

Now we sample actions. To learn, we need to follow ∇θ E[return].

2. The Policy Gradient Theorem

The Goal in One Line

Maximize expected return as a function of policy parameters:

$$J(\theta) \;=\; \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]\;, \quad R(\tau) \;=\; \sum_t r_t$$

Then do gradient ascent:

$$\theta \;\leftarrow\; \theta \;+\; \alpha\,\nabla_\theta J(\theta)$$

Sounds easy. How do you take the gradient of an expectation over trajectories drawn from π_θ ?

Think: Spot the Problem

You want $\nabla_\theta J(\theta)$. The expectation is over trajectories drawn from π_θ. The thing whose distribution depends on θ is also what you want to differentiate w.r.t. θ.

Discuss — three prompts:

(a) If you sample N trajectories and average their returns, can you backprop through that?

(b) The action a was sampled — is a a differentiable function of θ?

(c) Why doesn't the reparameterization trick (VAEs, L24) work directly here?

The Log-Likelihood Trick

One algebraic step that fixes everything:

$$\nabla_\theta \, \mathbb{E}_\pi[f(\tau)] \;=\; \mathbb{E}_\pi[\, f(\tau) \cdot \nabla_\theta \log \pi(\tau)\,]$$

Why? One-line identity:

$$\nabla_\theta \, p \;=\; p \cdot \nabla_\theta \log p$$

The gradient is itself an expectation we can estimate by sampling.
No backprop through the environment needed
— sample-and-average an estimator of the gradient.

The Policy Gradient Theorem

Plug into the trajectory return:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \;\cdot\; R(\tau)\right]$$

In words: increase log-probability of actions that led to high returns; decrease it for actions that led to low returns.

REINFORCE - Williams (1992). *One of the most-cited results in modern RL.*

No value function. No Bellman equation. Just sample, score, multiply, average.

[1] Williams et al., Simple statistical gradient-following algorithms for connectionist reinforcement learning, 1992.

Pseudocode of REINFORCE

```
for iteration = 1, 2, ... :
    sample trajectory  τ ~ π_θ
    compute return  G = Σ_t r_t
    for each step t in τ :
        θ ← θ + α · G · ∇_θ log π_θ(a_t|s_t)
```

Three things happen each iteration:

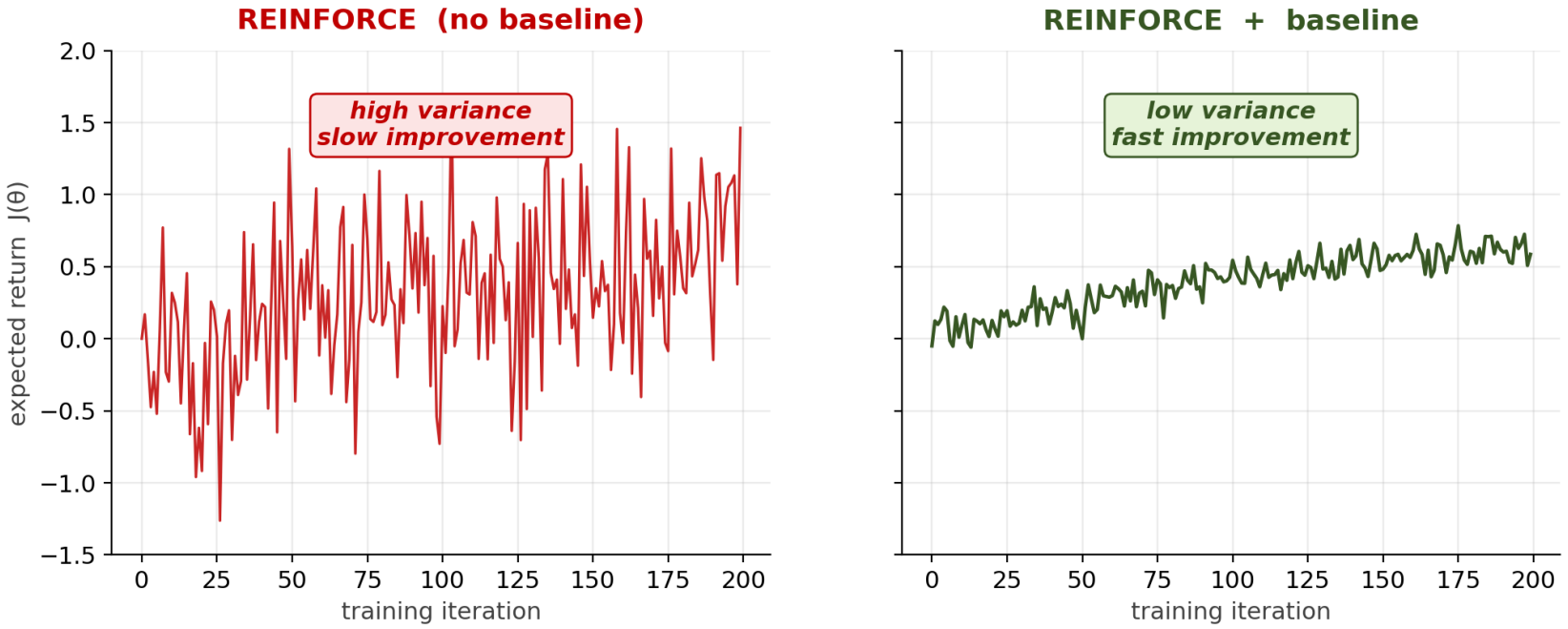
1. **roll out** — sample a trajectory from the current policy.
2. **score** — compute its return G.
3. **credit-assign** — push log π up where G is high, down where G is low.

That's it. Now where does it break?

3. From REINFORCE to ChatGPT

REINFORCE's Problem: Variance

- **Situation:** REINFORCE is unbiased — the estimator is correct on average.
- **Complication:** It's also catastrophically high-variance. G can swing wildly between rollouts.
- **Question:** *Can we reduce variance without introducing bias?*



The Baseline Trick

Subtract any function b(s) that doesn't depend on a :

$$\nabla_\theta J \;=\; \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \;\cdot\; \big(G_t - b(s_t)\big)\right]$$

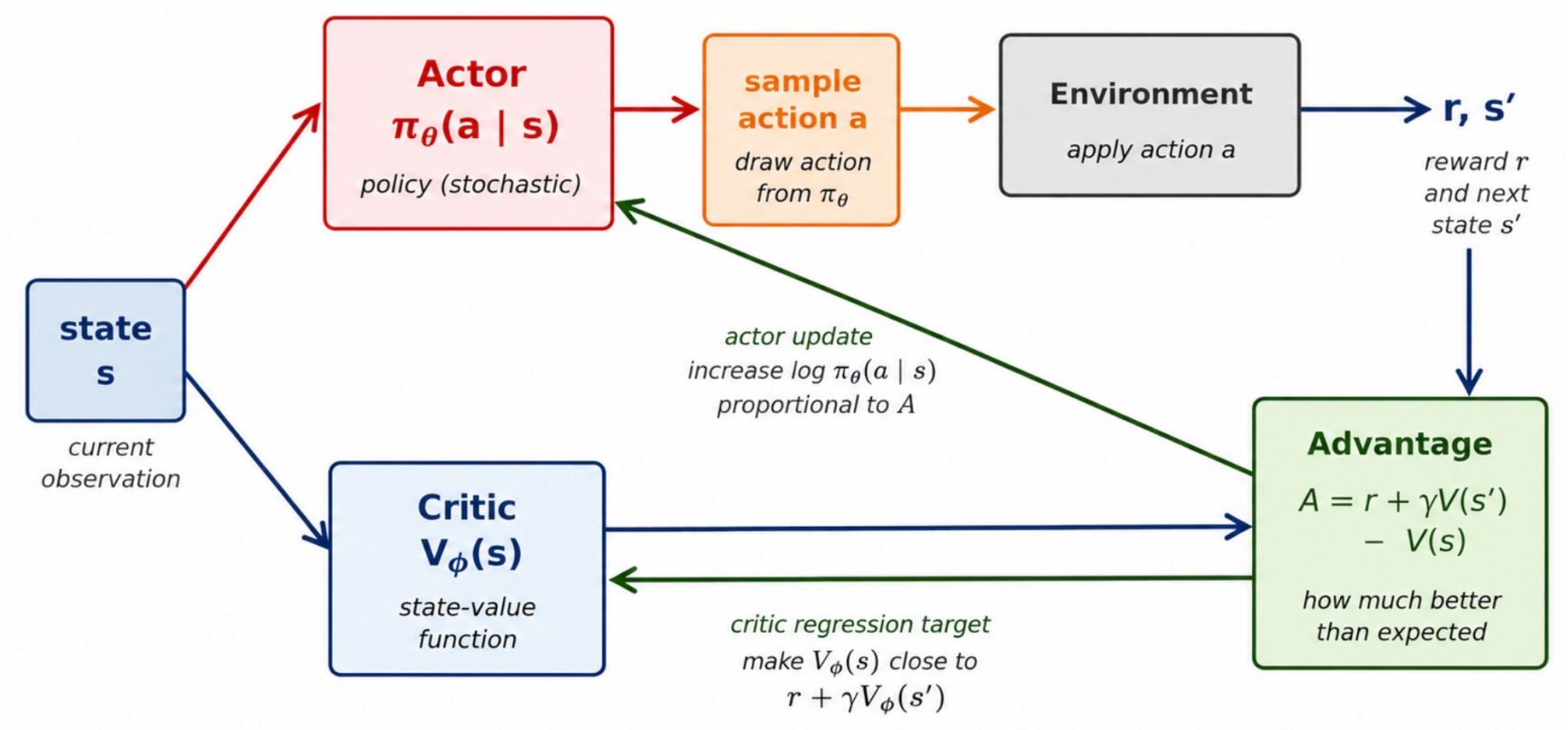
Claim: leaves the gradient unbiased, but can dramatically reduce variance.
Why unbiased: $E[\,\nabla \log \pi \;\cdot\; b(s)\,] \;=\; b(s) \;\cdot\; \nabla\, E[1] \;=\; 0.$

Best choice of baseline: b(s) = V(s) — the expected return from s .

Now $G_t - V(s_t)$ is the advantage A_t — "how much better was this action than average?"

Actor-Critic: The Two-Network View

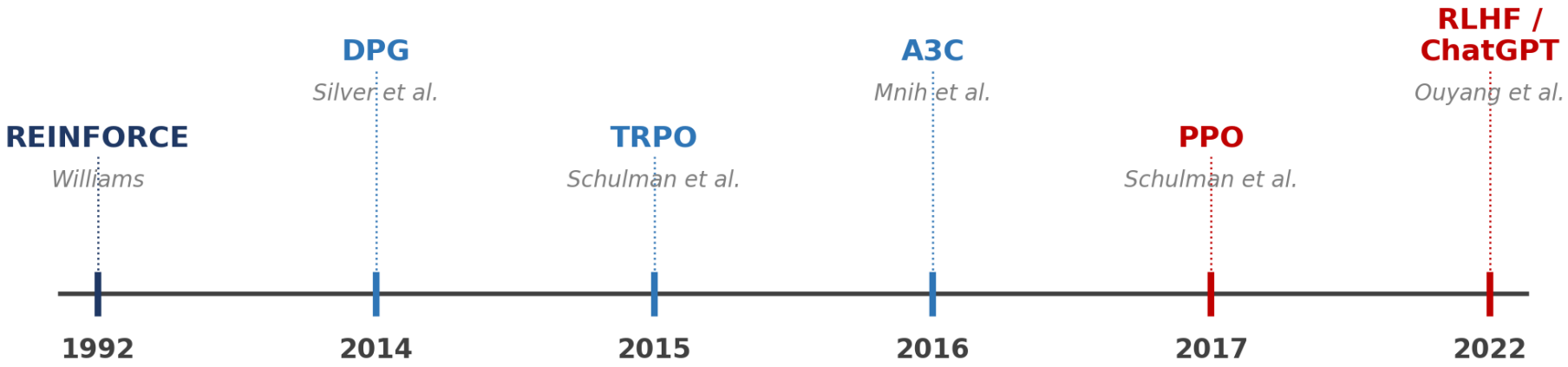
- **Actor:** $\pi_\theta(a \mid s)$ — the policy. Tells you what to do.
- **Critic:** $V_\varphi(s)$ — the value baseline. Tells you what to expect.



This is the algorithmic skeleton of every modern policy gradient method.

The Modern Family: PPO and Friends

- **A2C / A3C (2016):** parallel actor-critic — the mainstream of the deep-RL era.
- **TRPO (2015):** don't take a step that changes the policy too much (controlled via KL divergence).
- **PPO (2017):** TRPO's idea, simplified — clip the update ratio. The most-used policy gradient method today.

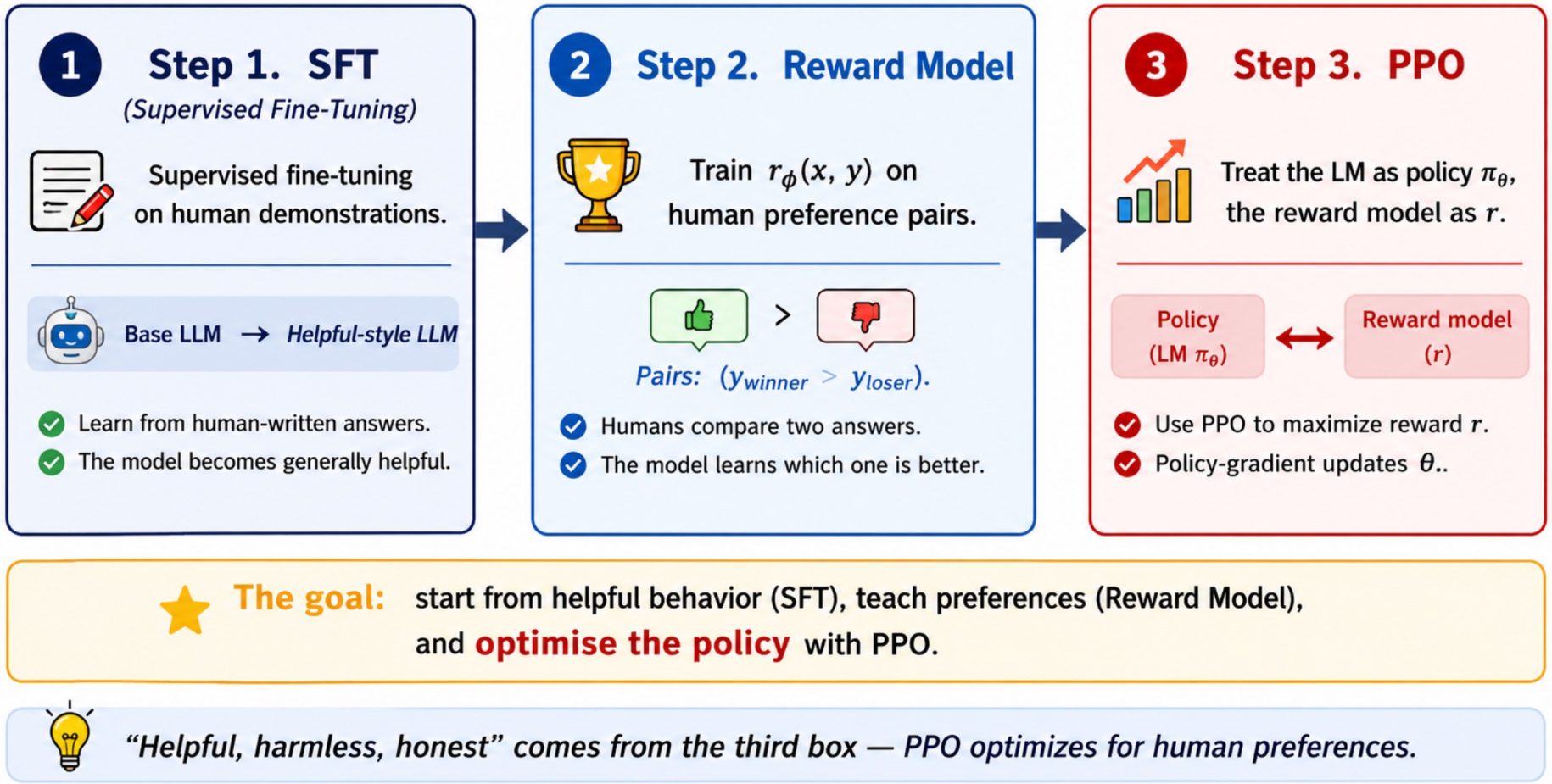


REINFORCE was the seed. PPO is the production-grade descendant. RLHF brought it to language.

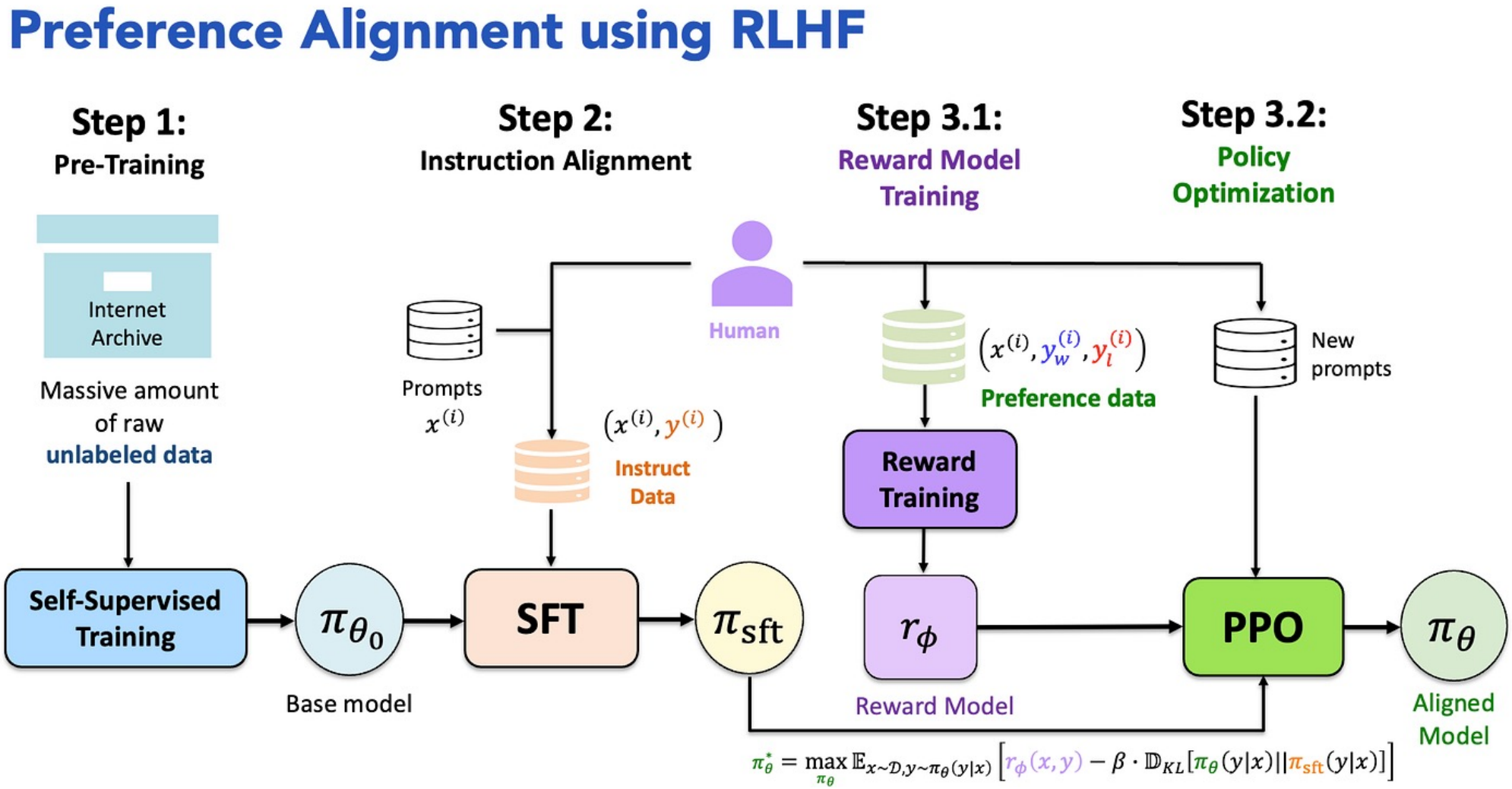
PPO is the algorithm that trained ChatGPT.

RLHF: The Payoff

- Pretrained LMs are great at next-token prediction, but bad at "be a helpful assistant". RLHF fixes that:



ChatGPT — RLHF in Production



Three years from PPO paper (2017) to ChatGPT (2022). The bridge was scale + RLHF.

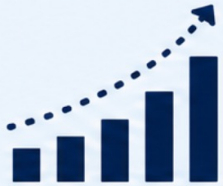
Summary

- Value-based methods break on continuous actions, stochastic-optimal policies, and non-smooth optimization.
- Policy gradient parameterizes π directly and does gradient ascent on expected return.
- The log-likelihood trick turns ∇ of an expectation into an expectation of ∇.
- REINFORCE: unbiased but high-variance.
- Baseline / advantage reduces variance — and gives us actor-critic.
- PPO is the production-grade actor-critic — what trained ChatGPT.

Frontier: Where Policy Gradient Stops

- Three binding constraints — each is its own active research area:

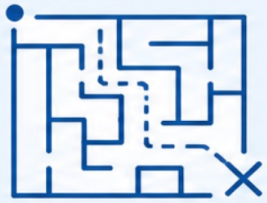
Sample efficiency



PPO needs millions of rollouts.

→ *off-policy methods, SAC*

Exploration



Gradient ascent gets stuck in local optima.

→ *entropy bonuses, intrinsic curiosity*

Models of the world

We sample the environment — we don't learn it.

→ *L34: Model-Based RL*

Bridge to L34: Model-Based RL

- *Five lectures (L29–L33), one assumption — the environment is a black box you sample.*

What if you learn a model of it instead?

Two questions for L34:

1. Can a learned world model do everything DP did — without knowing the rules?
2. Why is sample efficiency the holy grail of RL, and how does a model help?

L34: Model-Based RL — the last lecture of the RL module.