上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 29: Markov Decision Processes

Tao Huang

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

https://taohuang.info/cs3317

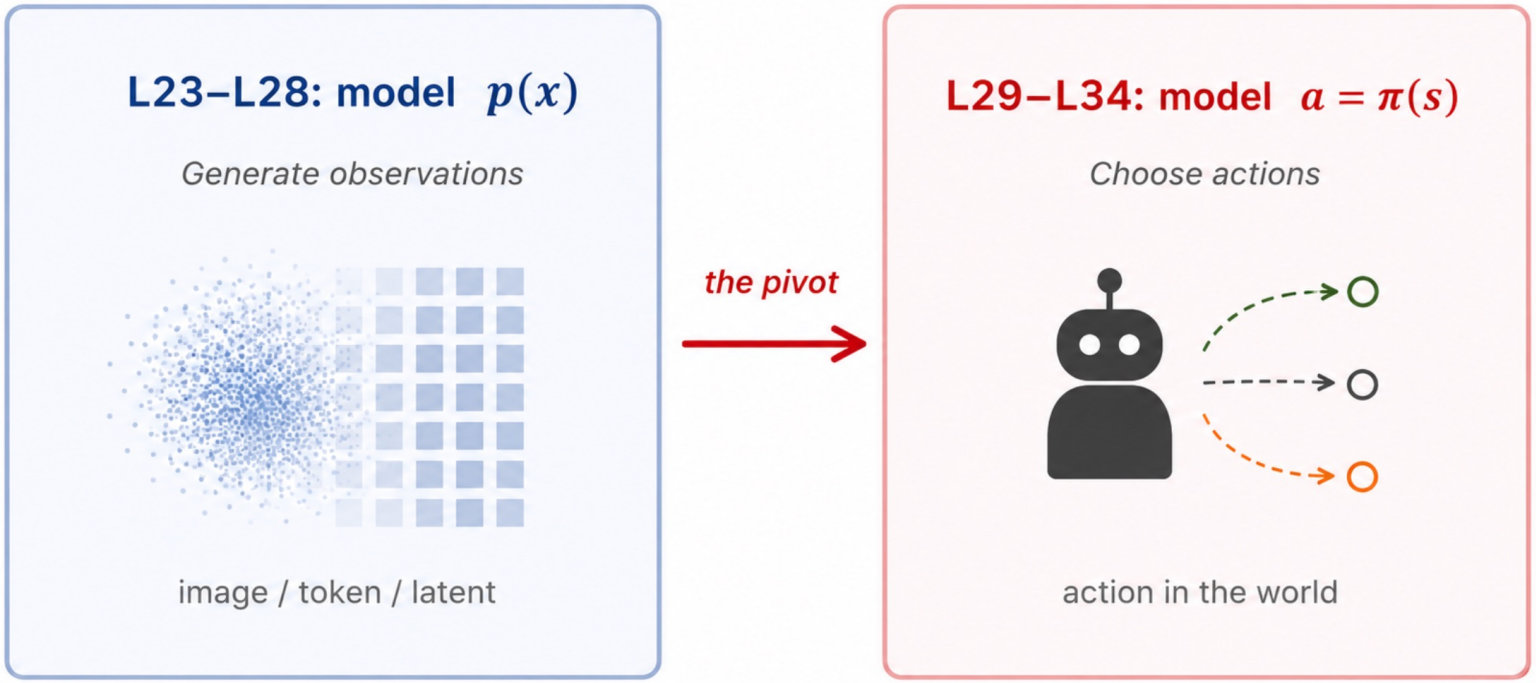
https://oc.sjtu.edu.cn/courses/89538

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've Been: A Pivot

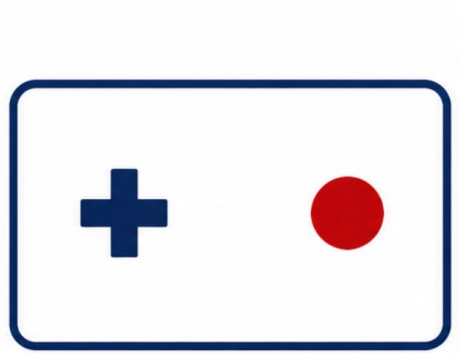
- L23–L28 was about generating data — images, language, multimodal.
- Models learned the structure of a static distribution p(x).

But intelligence isn't just generation — it's deciding what to do next.

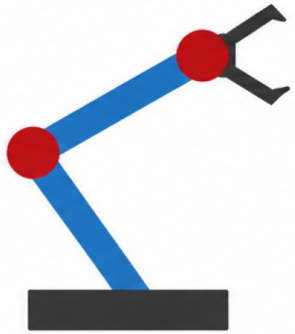


The Big Idea

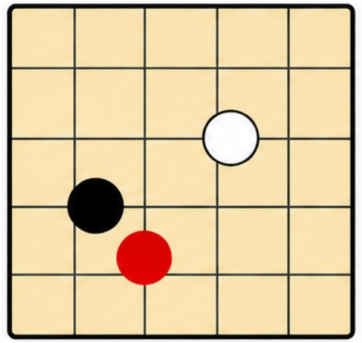
An agent in an environment,
choosing actions to maximize reward over time.



Atari

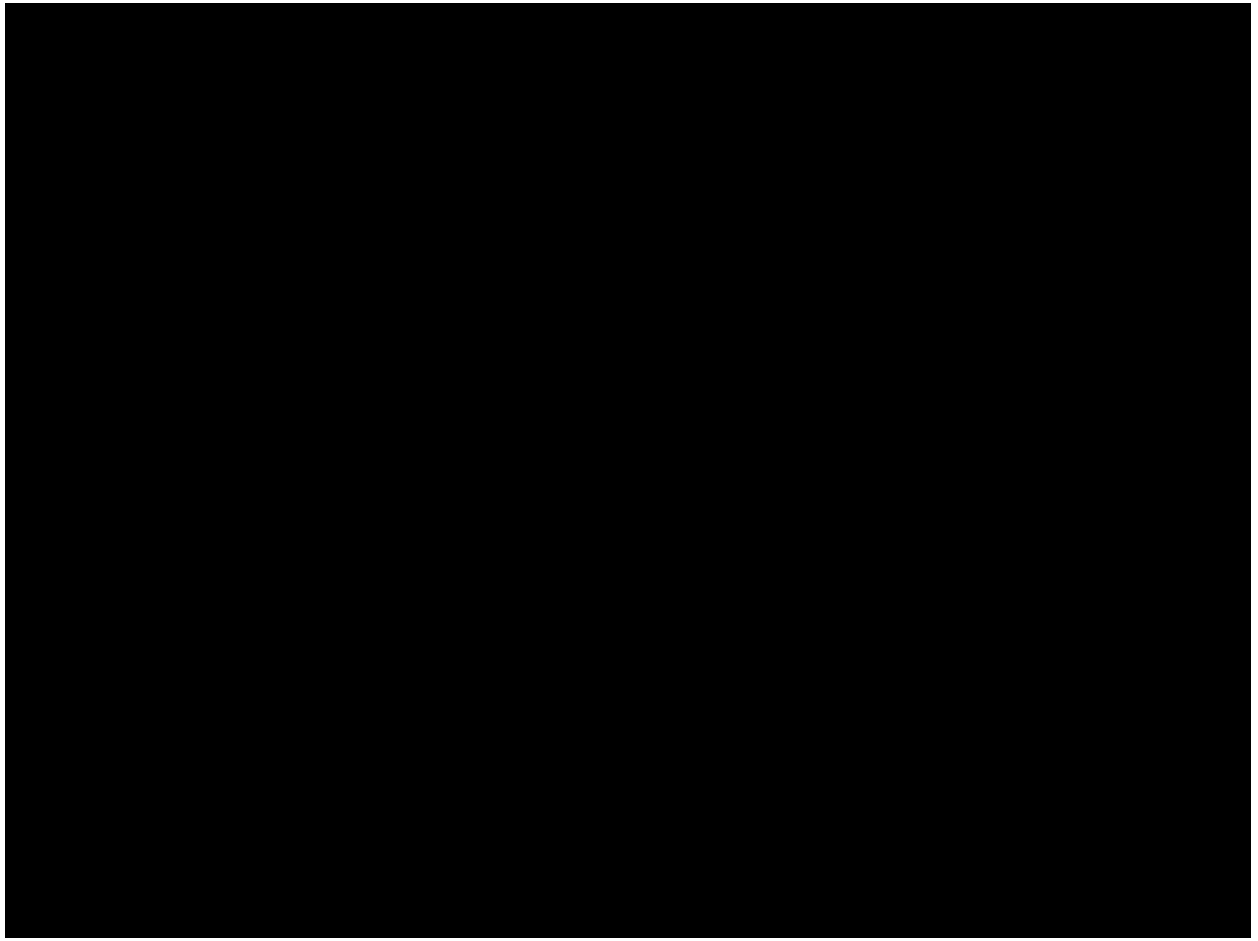


Robotics



Go

A 60-Second Hook



How does a neural net learn this — with no labels, just a score?

Objectives

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

- Identify states, actions, rewards, and transitions in a new problem.
- Formulate a task as a Markov Decision Process.
- Distinguish between rewards, returns, and value functions.
- Write down the Bellman equation for V^{π} and explain its recursive form.
- Reason about why discounting is used — and what γ controls.

Outline

Part 1.

The RL Problem

what makes deciding different from predicting

Part 2.

MDPs

the formal model

Part 3.

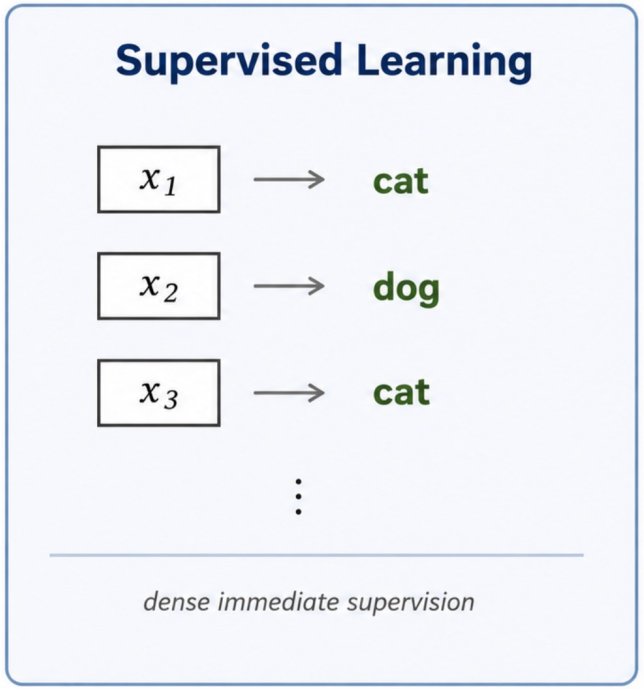
Values & Bellman

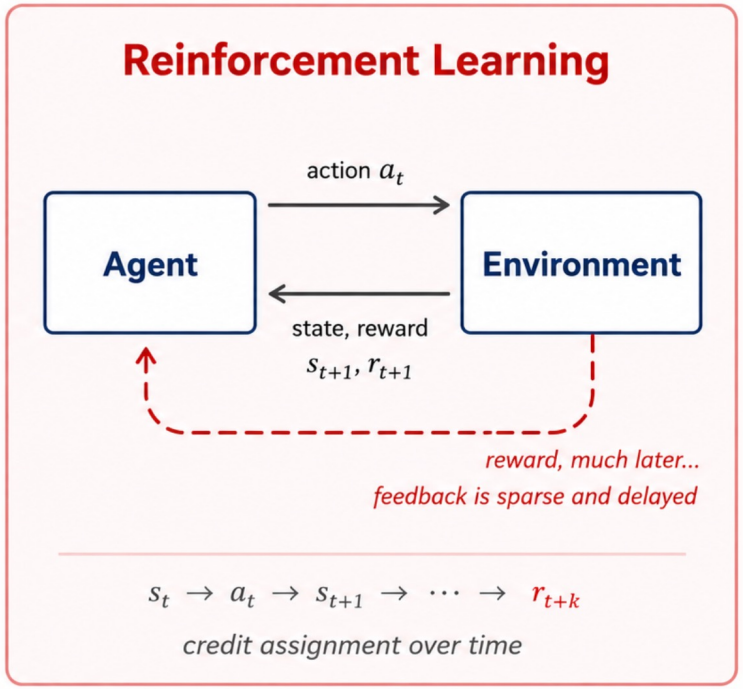
what we actually want to compute

1. The RL Problem

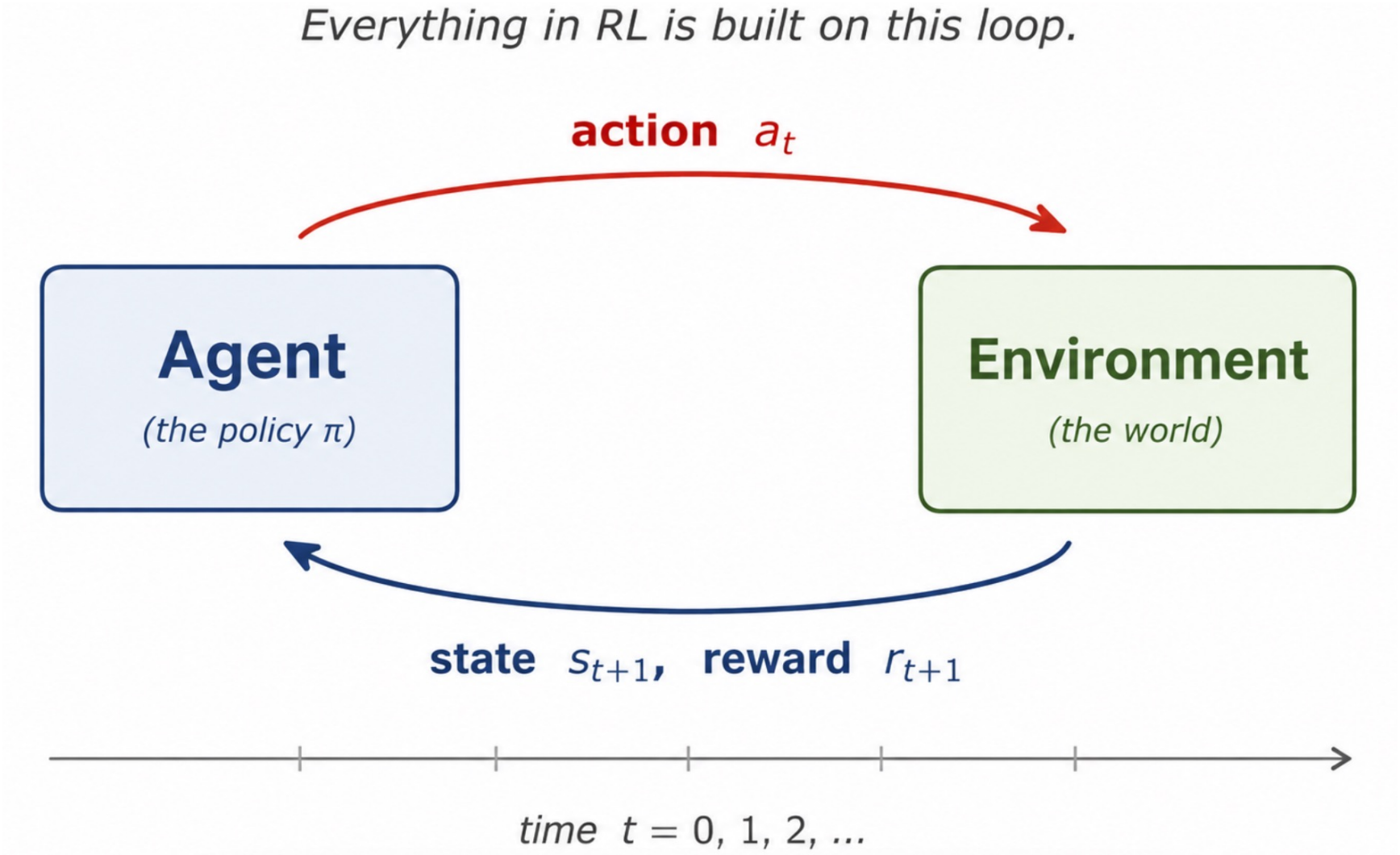
Supervised vs. Reinforcement

- **Situation:** Supervised learning gives the model the answer for every input.
- **Complication:** In Breakout, there's no "correct action" for each frame. Only a score — and it comes much later.
- **Question:** *How do you learn when feedback is delayed, sparse, and your own choices shape the data?*





The Agent–Environment Loop



Three Things That Make RL Hard

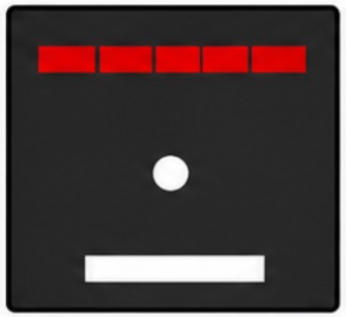
1. Delayed reward. The action that loses the game might happen 50 moves before “Game Over”.

2. Exploration vs. exploitation. To learn, you must try unknown actions. To score, you must use known good ones.

3. Non-i.i.d. data. Your current policy decides which states you visit next.

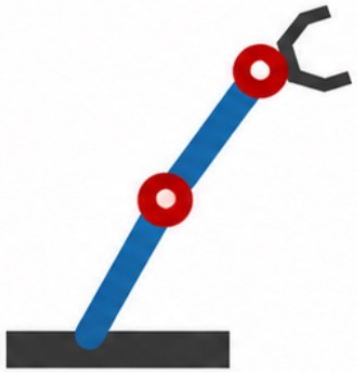
RL Is Everywhere

Game playing



AlphaGo, Atari, StarCraft

Robotics



manipulation,
locomotion

LLM post-training

"helpful, harmless, honest"

RLHF, RLAIF
(ChatGPT)

2. Markov Decision Processes

Why We Need a Formal Model

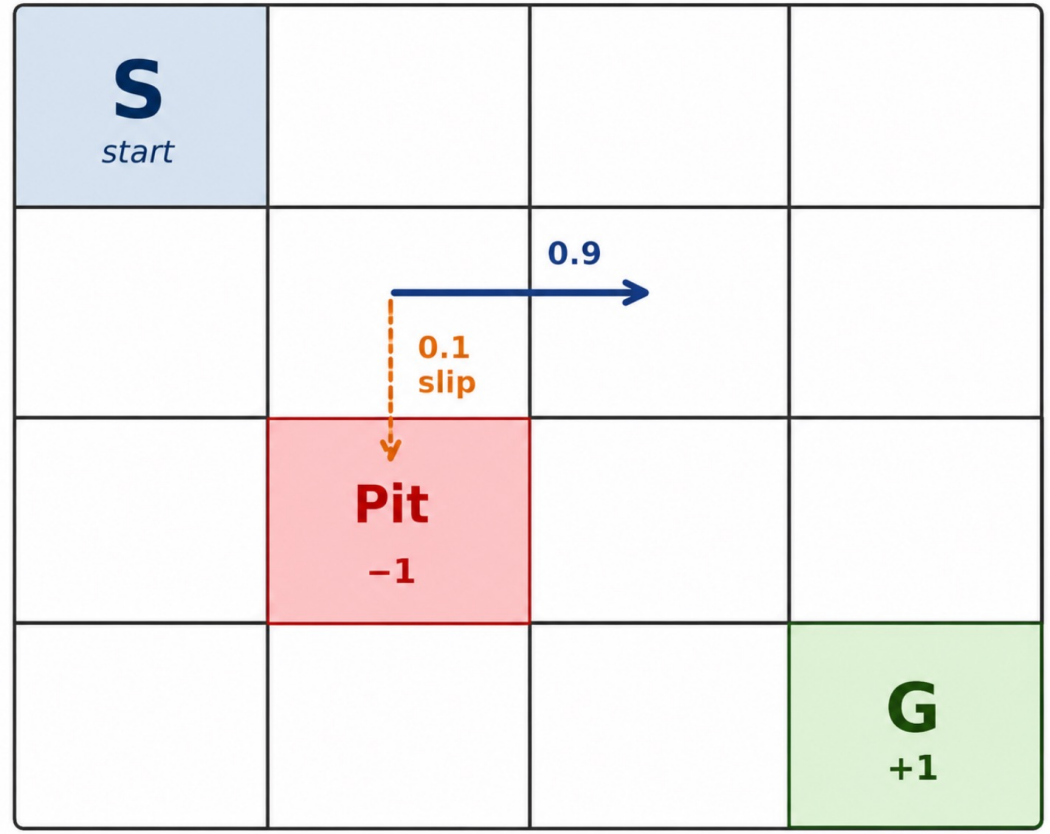
- **Situation:** We have a loop with states, actions, rewards.
- **Complication:** "Loop with rewards" is too vague to build algorithms on.
- **Question:** *What assumptions does the agent actually need?*

- **Answer:** A Markov Decision Process — five things.

$$\text{MDP} = (\mathcal{S},\ \mathcal{A},\ P,\ R,\ \gamma)$$

The Five Pieces

- $\mathcal{S}$ states
  — *the 16 cells*
- $\mathcal{A}$ actions
  — *{up, down, left, right}*
- **P(s' | s, a)** transition prob.
  — *10% slip sideways*
- **R(s, a, s')** reward
  — *+1 goal, −1 pit, 0 else*
- **γ ∈ [0, 1)** discount factor



action '**right**' from this cell, with slip

The Markov Property

The future depends only on the present state — not the history.

$$\Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = \Pr(s_{t+1} \mid s_t, a_t)$$

This is a modeling choice, not a fact about the world.

If history matters — fold it into the state.

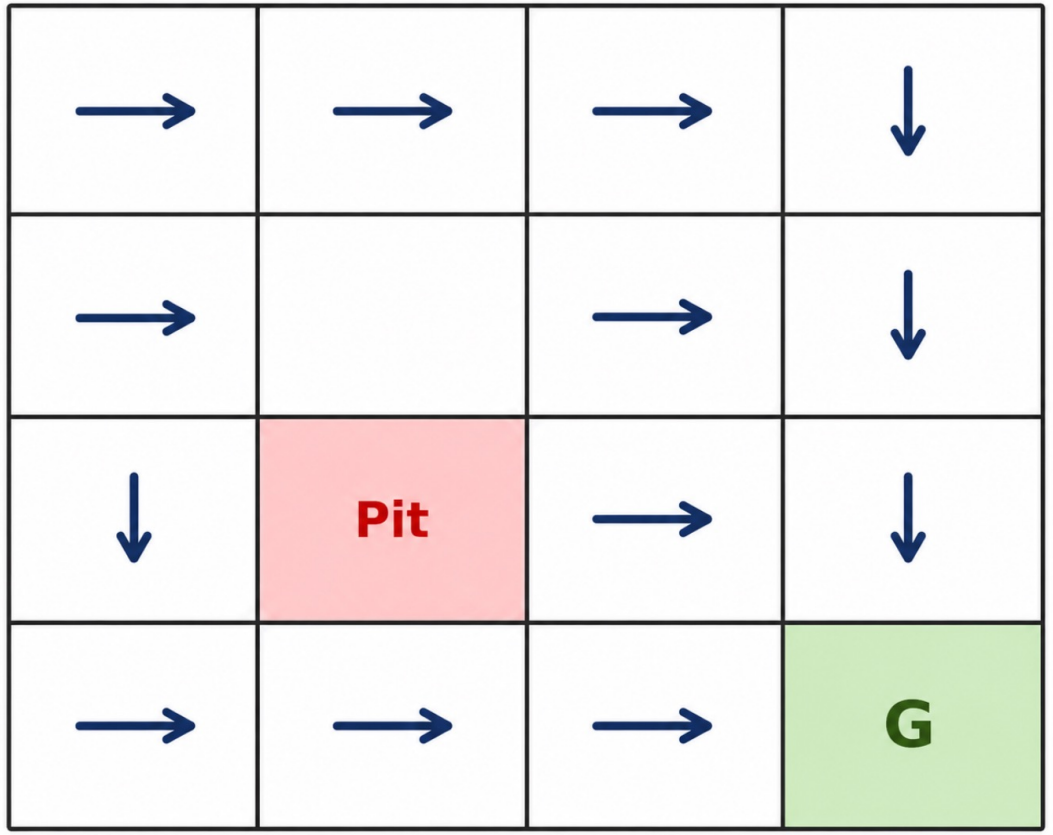
Policies

A **policy** π is a rule for choosing actions from states.

Two flavors:

- **Deterministic:** π(s) = a
- **Stochastic:** π(a | s) — a distribution over actions

Goal of RL: *find π★ that maximizes expected return.*



a deterministic policy $\pi(s)$ — one action per state

Think: Identify the MDP

A self-driving car following GPS directions. What are $\mathcal{S}$, $\mathcal{A}$, R?

Discussion prompts:

1. Is the state Markov if you only use the current GPS coordinates?

2. What's the reward — distance, time, comfort, or something else?

3. Do collisions get reward 0 or reward $-\infty$? Why does that matter?

3. Values and the Bellman Equation

Reward vs. Return

Situation: The agent gets a reward r_t at each step.

Complication: A single reward says nothing about the long run.

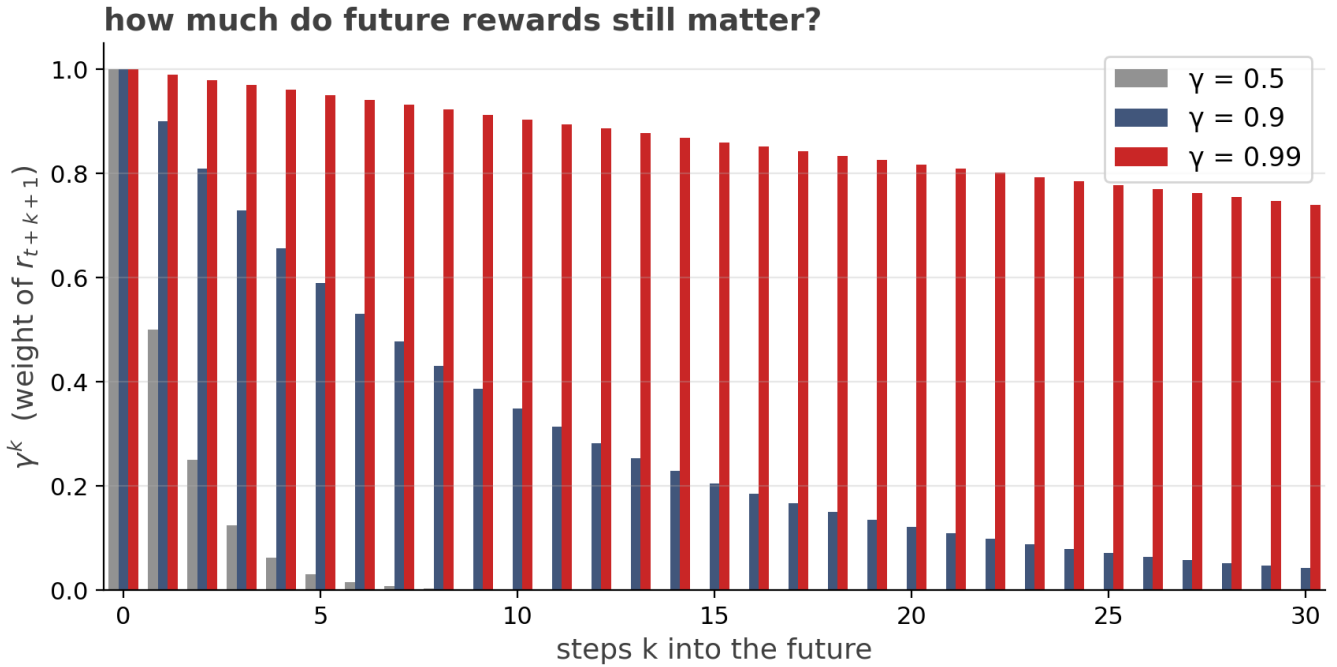
Question: *How do we score a whole trajectory, not a single step?*

Answer: the **return** G_t — cumulative future reward.

$$G_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}$$

Why Discount? (γ explained)

- **Mathematical:** without γ < 1, infinite sums diverge.
- **Practical:** near rewards matter more (uncertainty grows with horizon).
- **Behavioral:** γ controls how far-sighted the agent is. *γ = 0.9 ≈ 10-step horizon; γ = 0.99 ≈ 100-step.*



The Value Function $V^{\pi}(s)$

- The value of state s under policy π is the expected return if you start in s and follow π forever.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \,\middle|\, s_t = s \right]$$

A scalar per state.

The policy's credit score for being there.

Visualizing V^{π} on GridWorld

$V^{\pi}(s)$ — value of being in each state

- **Each cell:** the value of standing there under this policy.
- **High (green)** near the goal — many paths lead there with few steps.
- **Low (red)** near the pit — falling in costs −1.
- **Gradient in between** = the field of 'how good is here?'

If we knew V^{π}, we could almost act optimally — pick the action leading to the highest-value next state.

The Action-Value Function $Q^{\pi}(s, a)$

Expected return from taking action a in state s, then following π.

$$Q^{\pi}(s,a) \;=\; \mathbb{E}_{\pi}\Big[\, G_t \,\Big|\, s_t = s,\; a_t = a \,\Big]$$

Relationship to V:

$$V^{\pi}(s) \;=\; \sum_{a} \pi(a \mid s)\, Q^{\pi}(s,a)$$

Q separates the action from the policy — you can compare actions directly.

The Bellman Equation

Value here = reward you get + (discounted) value of where you go next.

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^{\pi}(s') \right]$$

policy: which action?

π(a | s)

world: where do we land?

P(s' | s, a)

now + later

R + γ Vπ(s')

Same shape as a recursion: define value in terms of next-step value.

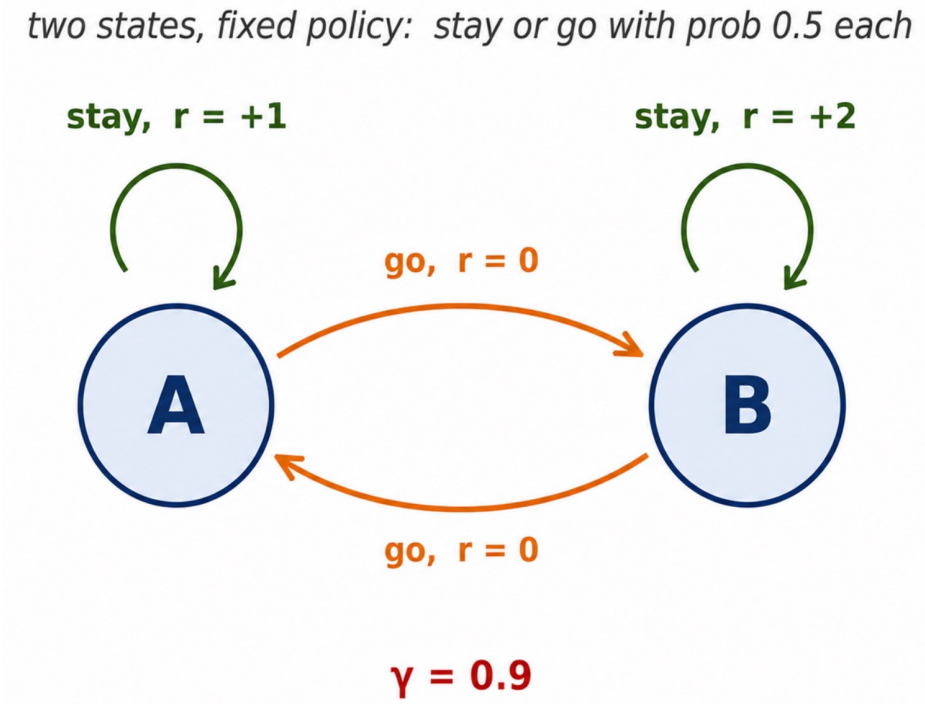
Why the Bellman Equation Matters

1. It's a **fixed-point equation**: V^π is the unique solution to $V = T^\pi V$.

2. It turns a **long-horizon problem** into a **one-step problem**.
   — exploited by value iteration (L30).

3. Every value-based RL algorithm is built on this equation.
   Q-learning, DQN, AlphaGo's value net, …

Example: 2-State MDP



$$V(A) = 0.5\big[1 + \gamma V(A)\big] + 0.5\big[0 + \gamma V(B)\big] \qquad V(B) = 0.5\big[2 + \gamma V(B)\big] + 0.5\big[0 + \gamma V(A)\big]$$

Solve the 2×2 linear system →

$$V^{\pi}(A) \approx 5.26 \qquad V^{\pi}(B) \approx 5.79$$

Optimal Value & Optimal Policy

Define the best achievable value, and the policy that achieves it:

$$V^\star(s) = \max_\pi V^\pi(s) \qquad \pi^\star(s) = \arg\max_a Q^\star(s,a)$$

For finite MDPs, an optimal deterministic policy always exists.

→ *Searching over deterministic policies is enough.*

Think: Designing the Reward

You're training a robot to walk. Which reward signal will it actually learn from?

(a) r = +1 when standing, 0 otherwise.

(b) r = +1 for every meter walked forward, −10 for falling.

(c) r = velocity of center of mass.

Which gives the densest learning signal?
Which exploits a loophole?
Which actually learns walking — vs. exploiting the simulator?

Summary

- RL = an agent maximizing expected return through interaction.
- MDP = ($\mathcal{S}$, $\mathcal{A}$, P, R, γ) — formalizes the loop with the Markov assumption.
- Vπ and Qπ measure how good a state (or state–action) is under a policy.
- The Bellman equation makes value recursive — that's what lets us compute it.
- Reward design is the hardest part of an RL problem. The math doesn't help you here.

Frontier: Where MDPs Stop

- The MDP formalism is a starting point. It breaks where:

Partial observability	you don't see the full state	*POMDPs*
Continuous state / action	sums → integrals; function approximation	*deep RL (L32)*
Multi-agent	others' policies are part of yours	*game theory (L37)*
Unknown P, R	the topic from L31 onward	*learning, not planning*

Bridge to L30

We know what we want — V^{π}.

L30: Dynamic Programming. Two algorithms — value iteration and policy iteration — that solve the Bellman equation when P, R are known.

After that — L31 drops the 'P, R known' assumption, and the real RL story begins.