Lecture 30: Dynamic Programming

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

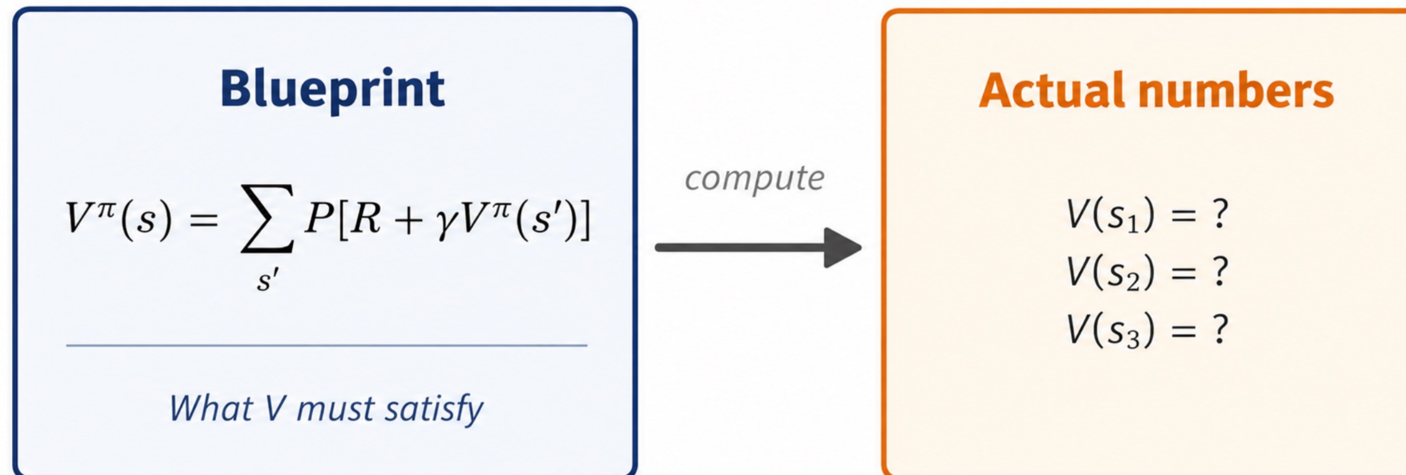
L29 gave us the vocabulary of RL:

states · actions · rewards · transitions · policies · values

And the Bellman equation — what V^π must satisfy.

But satisfying ≠ computing.

How do we actually find V^π and the optimal policy π*?



Big Idea

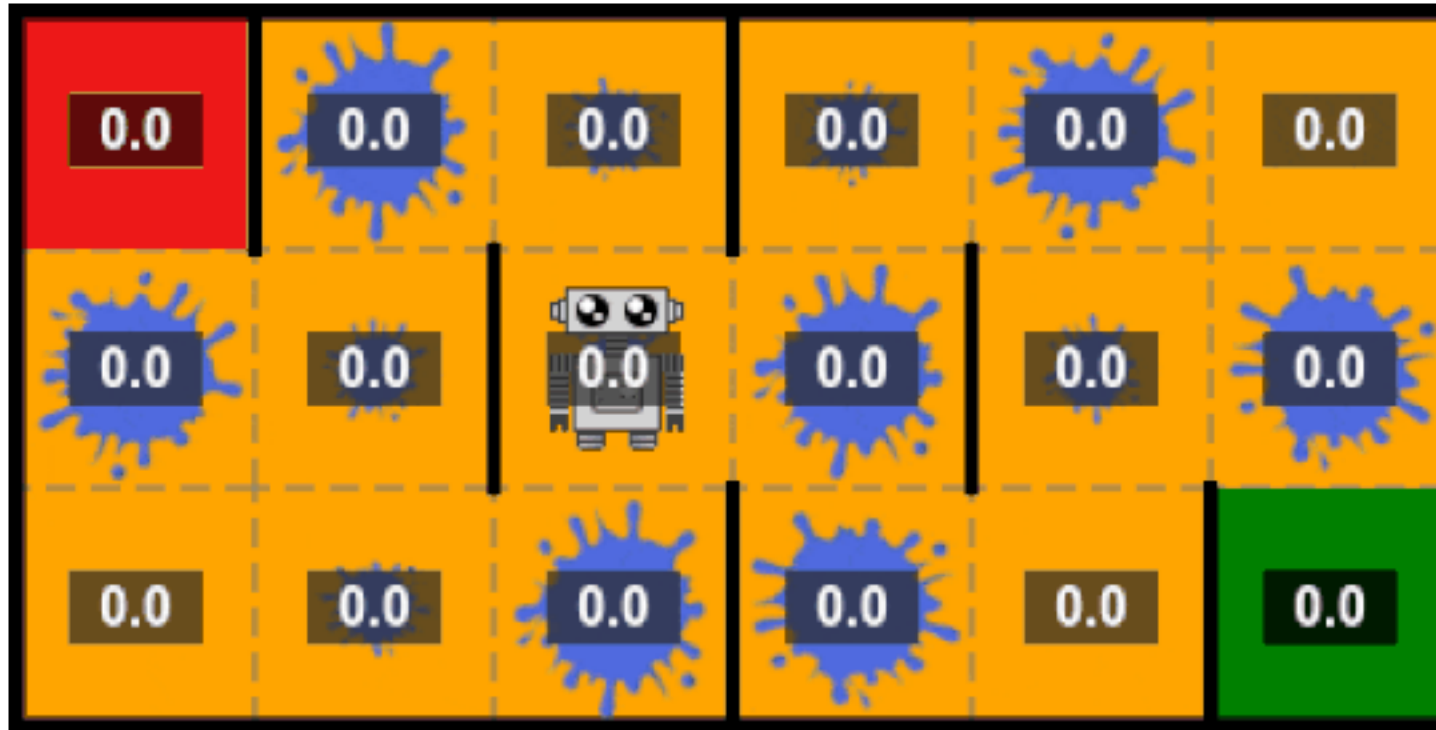
**When you know how the world works,
you can compute the optimal policy by repeated lookahead.**

Three settings where this works:

Gridworld maze	**Simple board games**	**Robot in known room**
P, R known	*P, R known*	*P, R known*

This is the easy setting. L31 drops the assumption — and the real RL story begins.

Value Iteration in Motion



https://towardsdatascience.com/policy-and-value-iteration-78501afb41d2/

Each pass updates every cell's value — until nothing changes.

Objectives

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

- **Explain** why the Bellman equation can't usually be solved by linear algebra.
- **Execute** one iteration of value iteration by hand on a small MDP.
- **Distinguish** value iteration from policy iteration.
- **Reason** about convergence — why repeated updates approach the truth.
- **Identify** when DP applies — and when it doesn't (preview of L31).

Outline

Part 1.

The Scaling Problem

why we can't just solve it

Part 2.

Value Iteration

fixed-point as algorithm

Part 3.

Policy Iteration

alternating evaluation and improvement

1. Why Can't We Just Solve It?

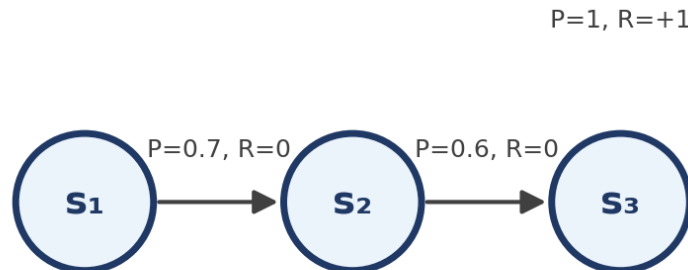
Situation: Bellman as a Linear System

From L29 — for a fixed policy π, the Bellman equation says:

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

For a fixed π, the right-hand side is linear in V.

A 3-state MDP under fixed policy π



Linear system: 3 equations, 3 unknowns

$$V(s_1) = 0.7\,[0 + 0.9\,V(s_2)] + 0.3\,[\cdot]$$

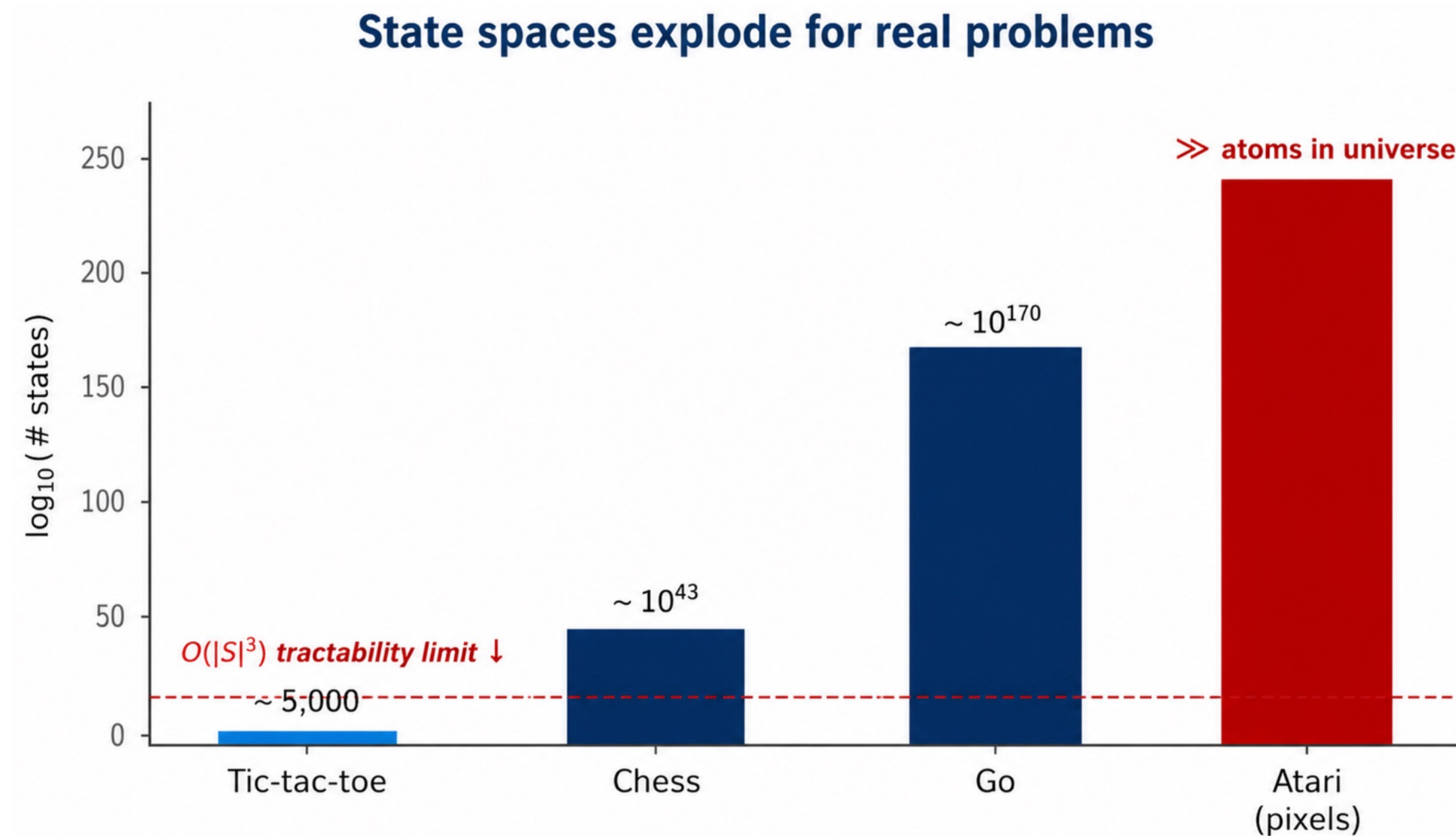
$$V(s_2) = 0.6\,[0 + 0.9\,V(s_3)] + 0.4\,[\cdot]$$

$$V(s_3) = 1.0\,[1 + 0.9\,V(s_3)]$$

Solve by matrix inversion: $O(|S|^3)$

(works when |S| is small)

Complication: States Explode



$O(|S|^3)$ matrix inversion dies long before chess.

Question

Can we find V^{π} without solving a giant linear system?

Idea preview —

instead of solving it once,
use the Bellman equation as an update rule, not an equation.

Answer: From Equation to Algorithm

Equation

$$V(s) = \sum_{s'} P\,[R + \gamma\, V(s')]$$

Must hold simultaneously for all s.

Update

$$V_{k+1}(s) \leftarrow \sum_{s'} P\,[R + \gamma\, V_k(s')]$$

Apply repeatedly, one pass at a time.

Same right-hand side. But now it's a program, not algebra.

We'll see this idea twice — first for a fixed policy, then for the optimal one.

2. Value Iteration

The Optimal Bellman Equation

- The version we'll iterate — for the optimal value function V*:

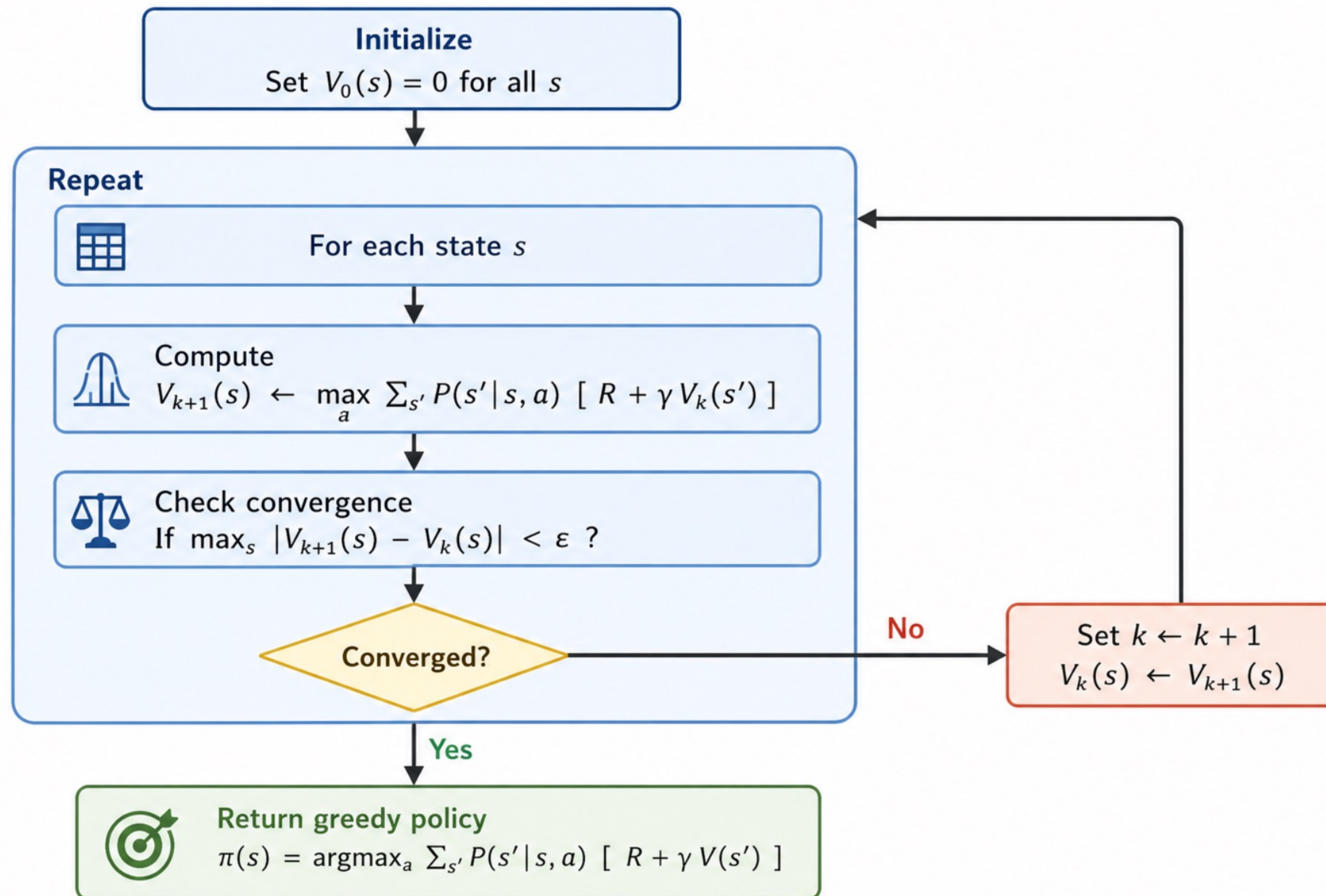
$$V^*(s) \;=\; \max_a \sum_{s'} P(s' \mid s, a)\big[\, R(s, a, s') + \gamma\, V^*(s') \,\big]$$

The only change from L29's V^π: pick the best action instead of averaging over π.

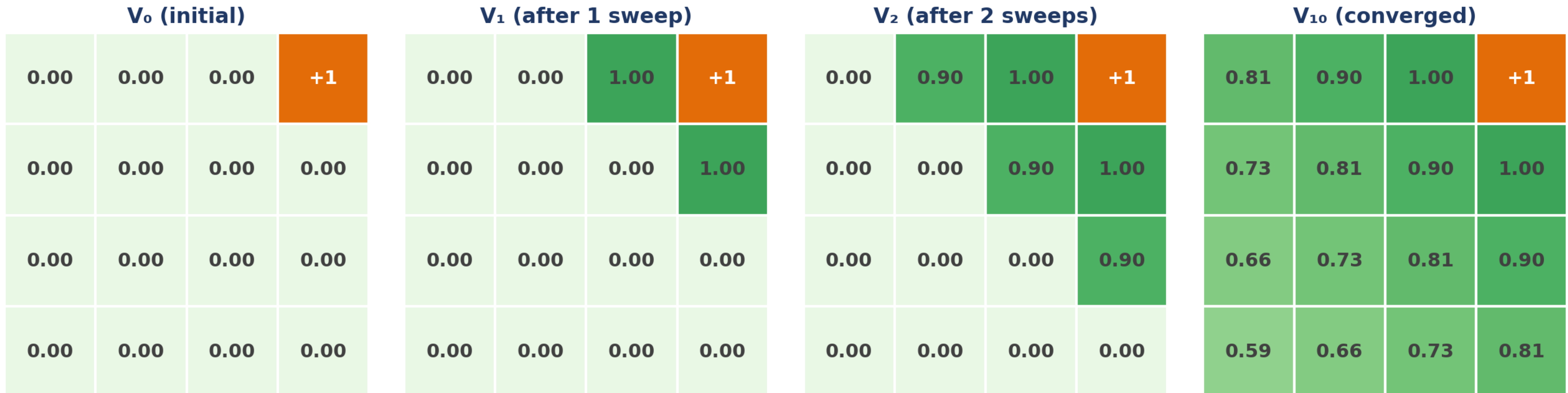
Two flavors of Bellman:

V^π *(policy evaluation)* *vs.* V^* *(optimal control — today)*

Value Iteration



Visual: One Sweep on a 4×4 Gridworld



Value propagates outward from the goal — one step per sweep.

Notice: V_1 knows only direct neighbors of the goal. V_2 knows two steps away.
Each sweep extends the "reachable horizon" by one more step.

Think: One Step of Value Iteration

Setup:

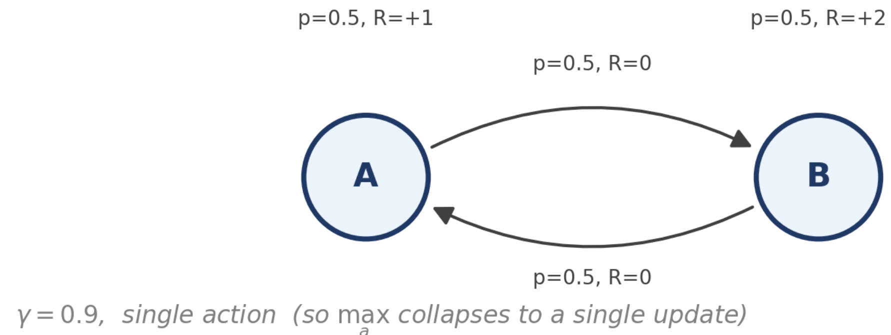
Deterministic transitions. $\gamma = 0.9$. Starting from $V_0 = (0, 0, 0)$.

Question: After one sweep of value iteration, what is V_1?

Discuss with your neighbor — 2 minutes.

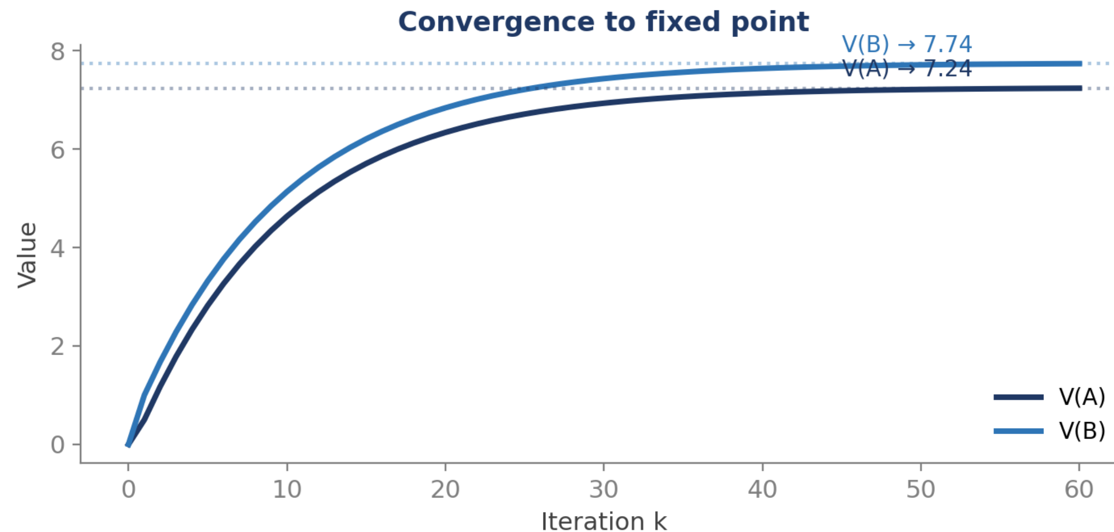
Hint: For each state, look one step ahead. What's the best you can do?

Example: 2-State MDP



Value Iteration Updates

k	V(A)	V(B)
0	0.000	0.000
1	0.500	1.000
2	1.175	1.675
3	1.783	2.283
5	2.821	3.321
10	4.635	5.135
20	6.338	6.838
50	7.211	7.711

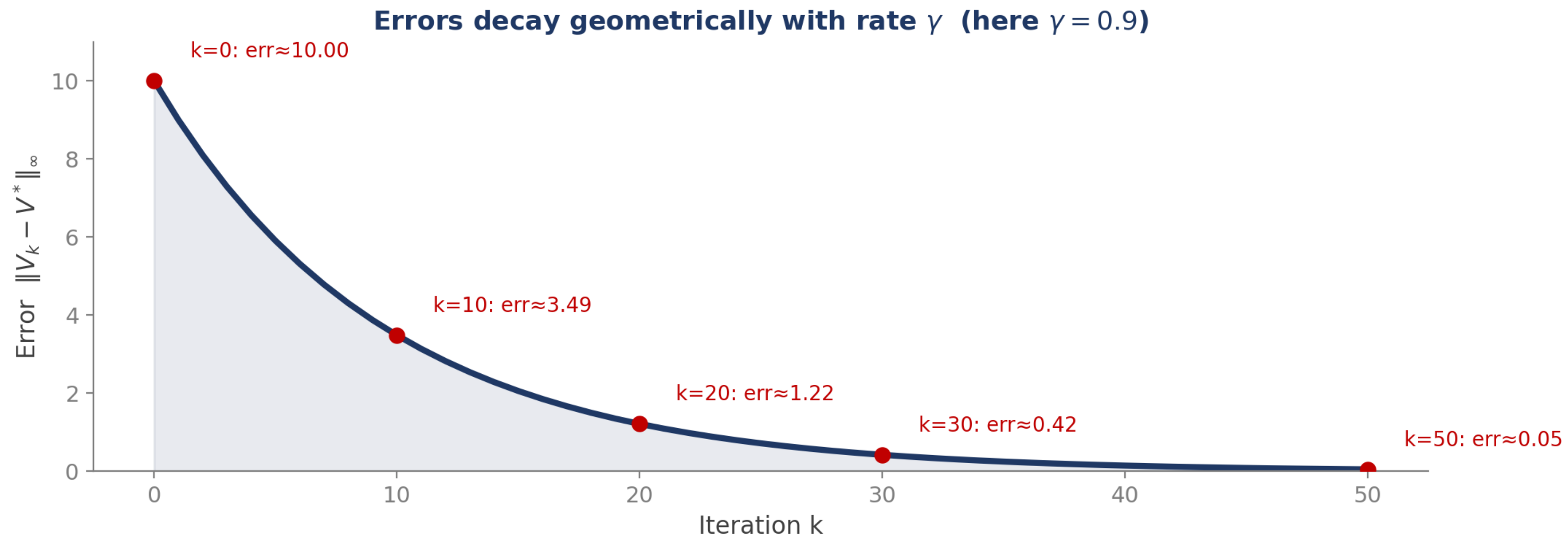


Same fixed point we'd get by solving the 2×2 linear system — without inverting a matrix.

Why Does It Converge?

Intuition (not a proof):

1. Each sweep brings V "closer" to V* by a factor of γ < 1.
2. γ acts as a shrink factor on errors — errors decay geometrically.
3. After enough sweeps, V ≈ V*.



Formal proof uses Banach's fixed-point theorem — beyond this course. The picture is what matters.

Honest Aside

Value iteration is just fixed-point iteration on the Bellman operator.

Everything that follows — Q-learning, DQN, PPO, AlphaGo's value net — is a way to do this when you don't have P and R written down.

Don't be impressed by the algorithm.
Be impressed by the equation.

2. Policy Iteration

A Different Question

Value Iteration asks:

What are the best values?

Policy Iteration asks:

Which actions should I take in each state?

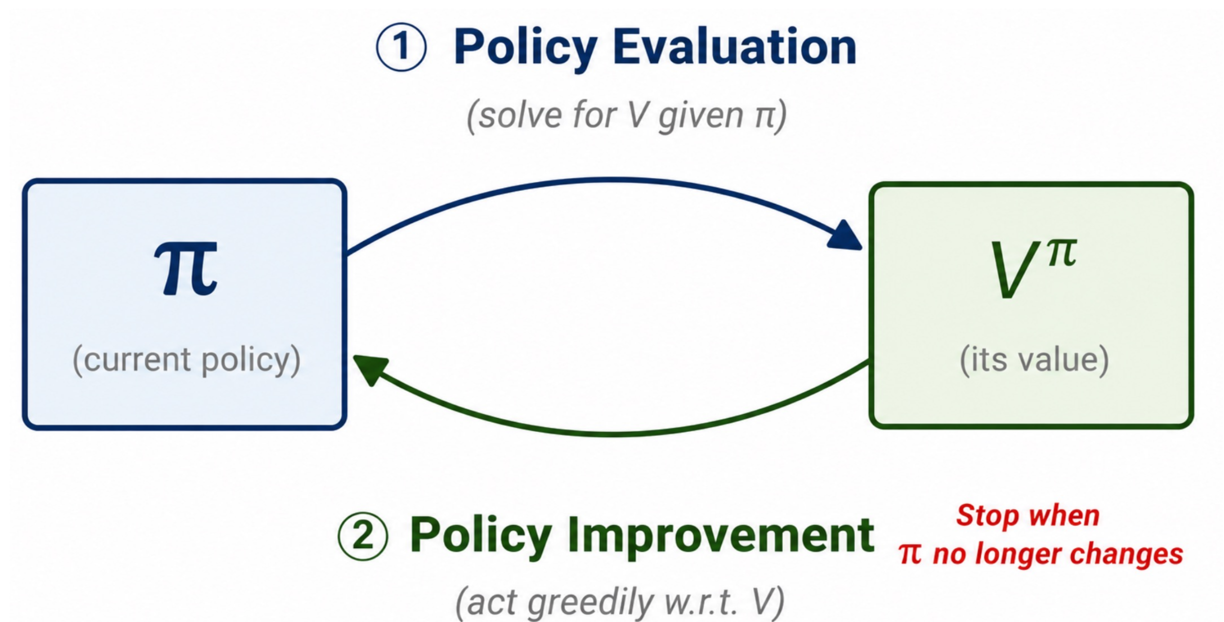
Both arrive at π^* — but via different routes.

The Two-Step Cycle

```
Initialize policy π₀ arbitrarily

repeat:
  ① Policy Evaluation:
       solve  V π  for the current π
  ② Policy Improvement:
       π′(s) ← argmaxₐ Σ P [R + γ V π(s′)]

  if  π′ = π:  stop.   else  π ← π′
```



Why Does It Terminate?

1. Each improvement step produces a strictly better policy.
2. There are only finitely many deterministic policies.
3. So the cycle must stop.

In practice: policy iteration usually converges in fewer than 10 iterations.

Value Iteration vs. Policy Iteration

	Value Iteration	Policy Iteration
Per-iteration cost	Cheap ($O(\|S\|^2\|A\|)$)	Expensive (solve linear system)
# iterations to converge	Many (until ε-converged)	Few (often < 10)
When to use	Large |S|, can't afford eval	Small |S|, want exact intermediate values

Two paths up the same mountain. In practice: mix both.

"Modified policy iteration" runs only a few sweeps of evaluation per cycle — a hybrid.

Think: How Close Are We?

Setup:

- You're running value iteration on a problem with 1 million states.
- After 50 sweeps, $\|V_{50} - V_{49}\|_\infty = 0.001$.
- $\gamma = 0.9$.

Roughly how close are you to V?*

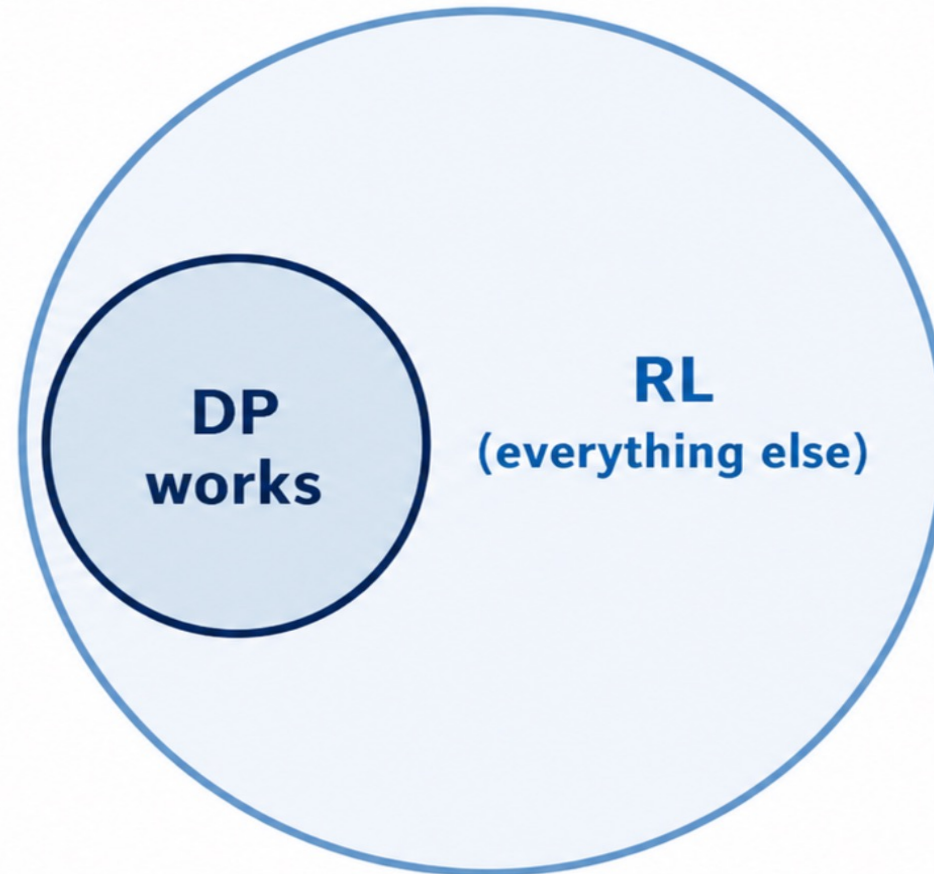
Hint: think about how errors shrink between consecutive iterations.

Where DP Lives — and Where It Doesn't

DP works when:

- P and R are known
- |S| is small enough

Examples:
gridworlds,
inventory,
simple games



DP fails when:

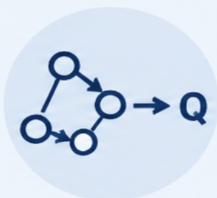
- P, R unknown (real robot)
- |S| huge (Atari, Go)

Examples:
Atari, Go,
robotics from pixels

L31 + L32: how RL handles the rest

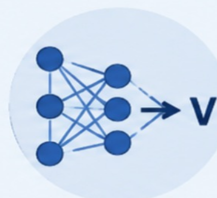
DP's Legacy: The Structure Everyone Reuses

Even when you can't run DP, its logic is everywhere.



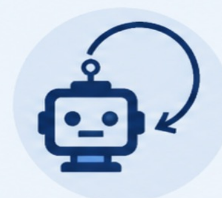
Q-learning (L31)

1-sample approximation of the Bellman update.



Deep Q-Networks

Neural-net approximation of value iteration.



AlphaGo's value net

Same idea, with self-play to estimate V.

DP is the grammar of value-based RL. The rest is dialects.

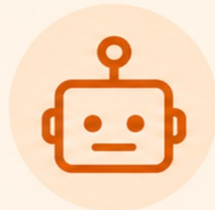
Summary

- The Bellman equation *defines* V* — but solving it directly takes $O(|S|^3)$.
- **Value iteration** treats Bellman as an update rule. Converges geometrically.
- **Policy iteration** alternates evaluation and improvement. Finishes in few iterations.
- Both require **known P and R** — the next lecture removes that assumption.

What If We Don't Know P and R?

Today

DP assumes
P, R
are written down.



But...

Reality:
a robot has never seen
a transition table
for the real world.

Next

L31: Temporal
Difference Learning

Learn the Bellman update
from samples.

This is the moment RL becomes learning.