

— IE437 · DATA-DRIVEN DECISION MAKING AND CONTROL — LECTURE 7

Markov Decision Processes & Dynamic Programming

A good plan stays good from wherever you land

Jinkyoo Park · KAIST

The deepest crossing — from choosing once, to choosing at every state your choices create

Where we are — the deepest crossing

PREVIOUS STEP	THIS CHAPTER'S QUESTION
Lecture 1 defined objectives and constraints; Lecture 3 added beliefs and utilities.	Now each action changes the next state and later opportunities.

A Markov state summarizes the information needed for future prediction. Begin with a known transition model and calculate an expected return.

Calculate a temperature-control backup, then repeat after changing a transition.

Learning route — one backup before a whole algorithm

Bring: Expectation, a discounted sum and finite tables.

FIRST PASS	WHAT TO DO
Follow the idea	State/action/return → Bellman backup → evaluate and improve → PI/VI
Work without the solution	Calculate a temperature-control backup, then repeat after changing a transition.
Return later	The contraction proof and detailed convergence assumptions are second-pass reading.

For the temperature thread: **predict** → **calculate** → **reveal and check** → **change one condition**. Complete the core calculation before reading the research extensions.

— ACT 1

The arena, and the value of a state

Q1. A greedy rule scores an action by its own reward. Build the object that scores a *situation* instead.

Why a sequence is not many single choices

from the deck's opening example

THE CHOICE, WITH NUMBERS

Standing in your final undergraduate year, three actions:

ACTION	REWARD NOW
get a job	\$4,000 / month
go to graduate school	\$500 / month
start a company	\$0 / month

A one-step-greedy rule takes the job and never looks up.

Actions are coupled **through the state**: what you do now changes the options and rewards you will have later. So an action cannot be scored by its own reward — only by **the whole future it unlocks**.

Which forces a new object. Instead of a number $f(x)$ attached to a decision, we need a number attached to a *situation* — the total future reward a good plan can extract from it — plus an equation tying each situation's number to its successors'. That equation is Bellman's, and it is the lecture.

WHY THAT IS THE WRONG READING

Graduate school pays 500 — and buys a **state**. From it a further action (publish X papers) opens a distribution over *researcher*, *professor*, *private company*, *officer*, *no job*.

The 500 was never the point. **The state it bought was.**

The arena — a Markov decision process

Q1 Why not greedy?

Q2 What must value satisfy?

Q3 How do we solve it?

Q4 Why does it converge?

A finite MDP is the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$:

- states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, both finite;
- a **transition function** $T(s, a, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a) = P(s' \mid s, a)$ — the source deck's own gloss: **also called the model, or the dynamics**;
- a **reward function** $R(s, a, s')$, or its expectation $\mathbb{E}[r_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$ if the reward is itself random;
- a **discount** $\gamma \in [0, 1)$, and a start state s_0 (plus terminals s_T for episodic tasks).

Stationary means T and R do not depend on t . That is what lets a chain of time slices collapse into a *single* recursive equation rather than one equation per step — and the two spellings $T(s, a, s')$ and $P(s' \mid s, a)$ name the same object. Lecture 8 will use the second, and delete it.

The goal is not a trajectory. It is a **policy** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that names the best action at every state.

The Markov property — and the licence it grants

$$P(S_{t+1} = s' \mid S_t, A_t, S_{t-1}, A_{t-1}, \dots, S_0, A_0) = P(S_{t+1} = s' \mid S_t, A_t)$$

Given the present state, the future and the past are independent. A state signal that retains all the relevant information is called Markov — the position and velocity of a projectile under known dynamics. A game state must also include the player to move and any history required by its rules.

The consequence is the one that matters, and it is not obvious:

$$\pi^*(s_t, s_{t-1}, \dots, s_0) = \pi^*(s_t)$$

For our finite discounted MDP, an optimal stationary policy can use only the present state. For a finite horizon, the best action can also depend on **how much time remains**: use $\pi_t(s)$ or include t in the state.

Without it, "the value of a state" would be meaningless — value would have to be indexed by every path that could have led there, and there are exponentially many. The Markov property is what makes a table with one row per state a sufficient answer. Lecture 3 arrived at the same condition from the other side: it is a missing-edge pattern in a Bayesian network over $S_1, \dots, S_T$.

The goal — return, and the discount

The agent maximises accumulated reward, not immediate reward. Two settings:

EPISODIC — THERE IS A LAST STEP

$$U_t = r_{t+1} + \dots + r_T = \sum_{k=0}^{T-t-1} r_{t+k+1}$$

A maze, a game of Go, a game of chess. The sum is finite because the episode is.

CONTINUING — THERE IS NOT

$$U_t = r_{t+1} + \gamma r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

With bounded rewards, $\gamma < 1$ makes the sum finite. Our convention throughout: take a_t at s_t , then receive r_{t+1} **and** s_{t+1} .

- $\gamma = 0$ — **myopic**: only the immediate reward counts, and we are back to the greedy rule that failed;
- $\gamma \rightarrow 1$ — **far-sighted**: distant reward counts almost as much as near reward.

A warning from the source deck worth repeating: **the reward signal communicates *what you want achieved*, never *how***. Reward a chess engine for taking pieces and it will take pieces; it will not win. Every reward specification is a modelling decision of the kind Lecture 1 taught, made under the extra hazard that the optimiser has many steps in which to find the loophole.

A policy is a rule, not a point

$$\pi : \mathcal{S} \rightarrow \mathcal{A}, \quad \pi^* = \arg \max_{\pi} \mathbb{E}[U^{\pi}(s)] \quad \text{for all } s$$

Those three words are the whole distance from Lecture 1. We do not want a policy that is best *on average over some start distribution*; we want one that is simultaneously best **from every state at once**. It is not obvious that such a policy exists. That it does is the first gift of the Bellman equation.

A policy may be deterministic, $a^* = \pi^*(s)$, or stochastic, $p(a \mid s) = \pi^*(s, a)$. For our finite, discounted, stationary MDP with bounded rewards, **a deterministic stationary optimal policy exists**, so we take π deterministic throughout. Stochastic policies return in Lecture 8, where randomness buys exploration, and again in **IE579**, where against an opponent it buys unpredictability.

The value of a state, and the value of an action

V^π — HOW GOOD IS IT TO *BE* HERE

$$V^\pi(s) = \mathbb{E}_\pi[U_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k \geq 0} \gamma^k r_{t+k+1} \mid S_t = s\right]$$

The expected return from following π starting at s .

Q^π — HOW GOOD IS IT TO *DO* THIS HERE

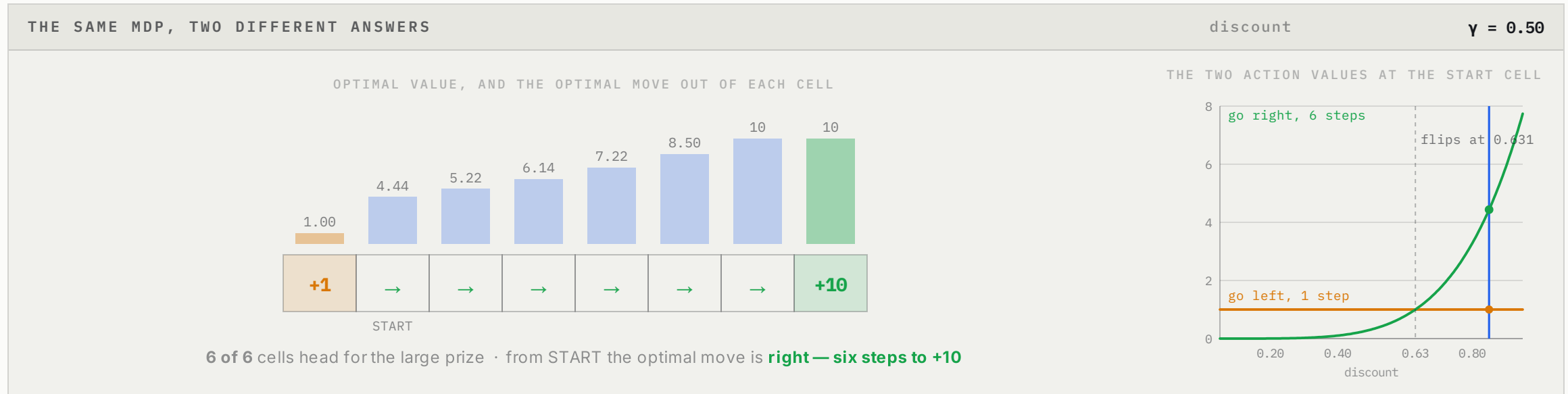
$$Q^\pi(s, a) = \mathbb{E}_\pi[U_t \mid S_t = s, A_t = a]$$

Take a now — whatever π would have said — then follow π forever after.

$\mathbb{E}_\pi$ is not an expectation over the policy; with π deterministic it is an expectation over **every stochastic transition the policy leads you through**. And both are defined *with respect to a policy*: there is no such thing as the value of a state, only the value of a state under a way of behaving.

This is exactly what the greedy rule lacked. A value scores a state by **the entire future a policy can extract from it**, not by one step's reward.

The discount is a decision, not bookkeeping



A corridor with a small prize one step to the left of START and a large prize six steps to the right. Turn γ and watch the arrows flip **one cell at a time, from the far end inwards**. The frontier is not a matter of taste — each cell has its own threshold, and they are exact: START switches at $\gamma = 10^{-1/5} \approx 0.631$, where $\gamma^5 \cdot 10$ overtakes 1; its neighbour at $10^{-1/3} \approx 0.464$; and the three cells nearest the large prize never point left at all. **Same MDP, same optimality principle, opposite answer.**

Check — why the state has to be Markov

The Markov property says the future depends on the past only through the current state. Why does the whole apparatus need it?

- ☒ A Because it lets the value of a state be written without reference to how you got there — which is what makes a recursion over states possible at all
- ☐ B To make the transition probabilities easier to estimate from data
- ☐ C To guarantee the reward is bounded
- ☐ D To ensure the optimal policy is deterministic

Without a sufficient state, the value of "here" can depend on earlier observations; one may need to include history or a belief state. The Markov property is what collapses an exponential tree of histories into a graph of states, and everything after it — the Bellman equation, dynamic programming, every algorithm in Lectures 8 to 12 — is built on that collapse. Choosing the state representation *is* choosing whether the assumption holds.

— ACT 2

The Bellman equation

Q2. The value of a state sums an infinite future. Turn that sum into a relation between neighbours.

Collapse the tree

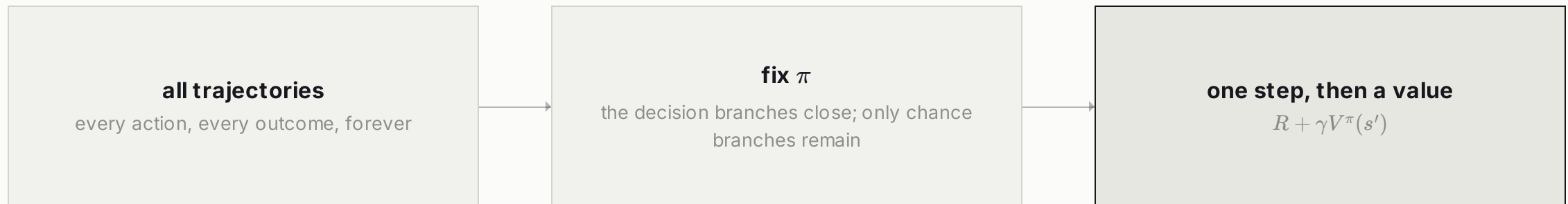
Q1 Why not greedy?

Q2 What must value satisfy?

Q3 How do we solve it?

Q4 Why does it converge?

Write out what $V^\pi(s)$ actually averages over. From s , every action; from each action, every successor; from each successor, every action again — a tree that never stops branching.



Fixing the policy prunes every decision branch. What remains is one layer of chance — and beyond it, a subtree that is *itself* a value function, because the Markov property says the future from s' depends on nothing but s' .

The infinite tree closes on itself. **That is the entire trick.**

The Bellman expectation equation

Peel one step off the return and let the Markov property close the loop:

$$V^\pi(s) = \mathbb{E}_\pi \left[\underbrace{r_{t+1}}_{\text{now}} + \gamma \underbrace{\sum_{k \geq 0} \gamma^k r_{t+2+k}}_{\text{the rest}} \mid S_t = s \right] = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

The value of a state = the expected immediate reward + the discounted value of where you land.

Read as a picture: a **state node** s leads to a **chance node** $(s, \pi(s))$, whose branches carry the weights $T(s, \pi(s), s')$ and end at successors holding $V^\pi(s')$. Every equation in this lecture is that diagram with one thing changed.

For a finite $\mathcal{S}$ this is a **system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns** — solvable in principle by matrix inversion, in $O(|\mathcal{S}|^3)$. It has turned "sum an infinite future" into "relate each state to its neighbours". That is the trade dynamic programming lives on.

Calculate one backup — reward now plus value later

Take one action. Let $\gamma = 0.9$ and suppose our current successor values are:

POSSIBLE OUTCOME	PROBABILITY	REWARD	NEXT VALUE	REWARD + DISCOUNTED VALUE
reach A	0.75	1	4	$1 + 0.9(4) = 4.6$
reach B	0.25	-1	2	$-1 + 0.9(2) = 0.8$

$$Q(s, a) = 0.75(4.6) + 0.25(0.8) = \boxed{3.65}.$$

Compare this with another action that pays **3 and terminates**: its value is 3. Choose the first action using these successor values.

A backup is a weighted average of **reward now + estimated future**. It is not an average of rewards alone.

Q and V — one lookahead apart

$$Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')], \quad V^\pi(s) = Q^\pi(s, \pi(s))$$

Q is the more general object: it keeps the first action free, then defers to π . Force $a = \pi(s)$ and it collapses back to V .

WHY WE WILL END UP STORING Q , NOT V

the source deck's sentence, kept whole

"The Q function effectively caches the results of all one-step-ahead search."

That cache is worthless while the model is free — with T in hand you can recompute the lookahead whenever you like. It becomes the whole ballgame the moment T is gone, because **the lookahead is precisely the thing you can no longer perform**. Lecture 8's first structural decision follows from this line and nothing else.

Bellman optimality — choose now, then choose again later

At the current state, choose an action before the random transition:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')].$$

If the first action is already fixed, only the next decision remains:

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')].$$

The two maxima choose actions at **different times**. We observe s' before choosing a' . This is not permission to swap a maximum and an expectation arbitrarily.

A single optimal continuation policy works from every successor in our finite discounted MDP. That is why the remaining value can be written as $V^*(s')$.

Temperature thread — calculate a temperature-control backup

shared teaching example · predict before revealing the calculation

At $x = -2$, choose off ($u = 0$) or heat ($u = 1$). For now, $x' = x + u$ is known. Reward is $r = -[(x')^2 + u^2]$, discount is 0.9, and the current continuation estimates are $V(-2) = -3$, $V(-1) = -1$.

Predict: Compare the immediate cost and the continuation value before choosing the action.

Calculate and check.

ACTION	REWARD	ONE BELLMAN TARGET
Off	-4	$-4 + 0.9(-3) = -\mathbf{6.7}$
Heat	-2	$-2 + 0.9(-1) = -\mathbf{2.9}$

This backup selects **heat**. The supplied V values are inputs to this sweep, not a claim that convergence has occurred.

An action is evaluated by what happens now and the value of the state it reaches.

Try it — heating succeeds only 20 percent of the time

work independently · reveal only after writing an answer

Off behaves as before. Heat reaches $x' = -1$ with probability 0.2 and remains at $x' = -2$ with probability 0.8, while still paying effort $u^2 = 1$. Keep the same continuation estimates. Does the backup still choose heat?

Check your answer. A failed heating step has reward -5 and target $-5 + 0.9(-3) = -7.7$. The heat target is $0.2(-2.9) + 0.8(-7.7) = -6.74$. Off gives -6.7 , so this backup chooses **off**.

Changing a transition changes both next-state cost and continuation value. Take the expectation over complete outcomes.

The principle of optimality, and the policy read off

BELLMAN'S PRINCIPLE OF OPTIMALITY

An optimal policy has the property that, whatever the first action and the state it leads to, the **remaining decisions must themselves be optimal from that state**.

Which is this lecture's thesis in formal dress: *a good plan stays good from wherever you land*. And it is what makes a daunting search over plans tractable — because it licenses solving the problem **one state at a time**. The optimal policy then falls straight out:

$$\pi^*(s) = \arg \max_a Q^*(s, a) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

The source deck's gloss: *any* greedy policy with respect to V^* is optimal, **because V^* already accounts for the reward consequences of all possible future behaviour**. Greed was never the enemy. Greed applied to the *immediate reward* fails; greed applied to **the right quantity** is exactly optimal. Act 1's question is now answered: we were not too greedy, we were greedy about the wrong number.

Check the timing — guessing a coin before or after the toss

A fair coin will be tossed. Choose **heads** or **tails**; a correct guess pays 1.

DECIDE BEFORE OBSERVING

Either guess has expected reward 0.5.

$$\max_a \mathbb{E}[R(a, C)] = 0.5.$$

DECIDE AFTER OBSERVING

You can match every outcome and collect 1.

$$\mathbb{E}[\max_a R(a, C)] = 1.$$

Putting the current action's maximum inside the expectation grants extra information. In Bellman's equation, only the **next action** is chosen after the next state is observed.

Check — expectation and maximum, in that order

The Bellman optimality equation contains both a **max** over actions and an expectation over next states. Why can the two not be swapped?

- ☐ A They can be — the order is a matter of convention
- ☒ B Because you must commit to an action *before* seeing which next state occurs; maximising inside the expectation would let the agent choose after the fact
- ☐ C Swapping them is valid only for deterministic environments, which is the usual case
- ☐ D Because the expectation is over an infinite set and the maximum over a finite one

Moving the **max** inside the expectation gives $\mathbb{E}[\max \dots]$, which values the ability to pick an action **after** the dice are thrown. That quantity is an upper bound and it describes a clairvoyant, not an agent. The gap between the two is the value of information, and keeping them in the right order is what makes the equation describe a decision made under genuine uncertainty.

— ACT 3

Solving it with the model

Q3. The Bellman equations are conditions, not procedures. Turn them into iteration.

Two halves of one machine

Q1 Why not greedy?

Q2 What must value satisfy?

Q3 How do we solve it?

Q4 Why does it converge?

Dynamic programming is, in the source deck's words, "a collection of algorithms that compute optimal policies **given a perfect model of the environment as a Markov decision process**". Its key idea is to use value functions to organise the search over policies. It has exactly two moves:

POLICY EVALUATION — MAKE V AGREE WITH π

$$V_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R + \gamma V_k(s')]$$

Sweep the expectation backup until it stops changing.

POLICY IMPROVEMENT — MAKE π GREEDY ON V

$$\pi'(s) = \arg \max_a \sum_{s'} T(s, a, s') [R + \gamma V^\pi(s')]$$

One arg max over a finite set, per state.

Set the right-hand box beside Lecture 1's $x^* = \arg \max_x f(x)$ — the source deck does exactly that, on one slide. Policy improvement *is* Lecture 1's arg max, executed once per state over $|\mathcal{A}|$ candidates instead of over a continuum. What is new is not the optimisation. It is **the quantity being optimised**.

Policy evaluation — sweeping the backup

```

Initialise  $V_0(s) \leftarrow 0$  for every  $s \in S$ 
Repeat for  $k = 0, 1, 2, \dots$ 
  for each state  $s$ :
     $V_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [ R(s, \pi(s), s') + \gamma V_k(s') ]$ 
until  $\max_s | V_{k+1}(s) - V_k(s) | \leq \epsilon$ 

```

This is a **full backup**: each sweep backs up the value of **every state, from every successor**, once. Nothing is sampled, nothing is skipped. Hold that phrase — Lecture 8's central move is to replace it with a *sample* backup from one successor.

TWO-ARRAY UPDATE

Read from V_k , write into V_{k+1} . The mathematically clean version: the sweep order cannot matter.

IN-PLACE UPDATE

One array. A state updated early in the sweep is read by states updated later, so **fresh information travels within a single sweep**. Usually faster, and half the memory.

The in-place variant is the first crack in the idea that DP must proceed in lockstep. Act 4 widens it into a hole.

Two sweeps — how a delayed reward changes the action

Let $\gamma = 0.9$. At **A**, stop for 2 or continue for 0 to B. At **B**, the only action pays 4 and terminates. Terminal value is 0.

SYNCHRONOUS SWEEP	VALUE AT A	VALUE AT B	GREEDY ACTION AT A USING THESE VALUES
initial	0	0	stop: $2 > 0$
first	$\max(2, 0 + 0.9 \cdot 0) = 2$	4	continue: $3.6 > 2$
second	$\max(2, 0 + 0.9 \cdot 4) = 3.6$	4	continue

The first sweep discovers B's reward. The next sweep carries it back to A. **Each new value uses the previous sweep's values.**

Try the same calculation at $\gamma = 0.4$: continuing is worth only 1.6, so stopping is optimal.

The gridworld — the source deck's worked example

4×4, undiscounted, reward -1 on every move

THE MDP

- $\mathcal{S} = \{1, \dots, 14\}$ plus two shaded **terminal** corners;
- $\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$, deterministic;
- a move off the grid **leaves the state unchanged** — so $T(7, \rightarrow, 7) = 1$;
- $R = -1$ on *every* transition, and $\gamma = 1$.

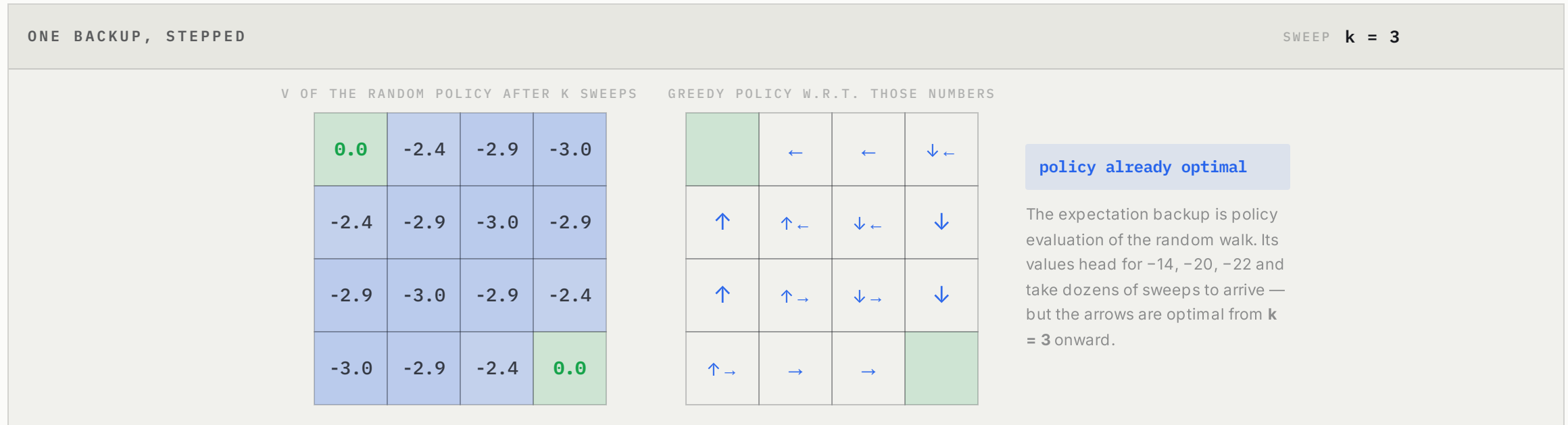
THE QUESTION

Under the **equiprobable random policy** — all four actions equally likely — what is V^π ?

Reward -1 everywhere means $-V^\pi(s)$ is simply **the expected number of steps a random walker takes to fall into a corner.**

$\gamma = 1$ is admissible here only because the **random policy being evaluated** reaches a terminal corner with finite expected hitting time; with a policy that never terminates, undiscounted evaluation simply runs to $-\infty$. Both corners are terminal, so value propagates inward from *two* sources at once. Watch the next slide's $k = 1$ frame: after one sweep every non-terminal state reads exactly -1, because a step costs 1 and everything it can reach is still worth 0. The information has not moved yet; after that it moves one ring per sweep.

Watching the value propagate



The deck's own figure, stepped. **Expectation backup** reproduces its printed table — -1.0, then -1.75 (which the printed figure truncates to -1.7), then -2.4, and at last -14, -20, -22. Switch to the **max backup** and the same sweep becomes value iteration: it settles in three sweeps at V^* = minus the distance to the nearest corner. The arrows are the greedy policy with respect to the numbers beside them, **recomputed every sweep**.

The policy is optimal long before the value is

The source deck draws a blue cross through the last two rows of its own figure. It is the most useful thing on the slide.

WHAT THE NUMBERS DO

At $k = 3$ the values read $-2.4, -2.9, -3.0$. They are heading for $-14, -20, -22$ and will take dozens more sweeps to get there.

WHAT THE POLICY DOES

At $k = 3$ the greedy policy is **already optimal, at every state**, and never changes again. At $k = 2$ it is not.

Evaluating to convergence was **wasted work**. The $\arg \max$ only needs the values *ordered* correctly, not converged.

This is the whole licence for truncating the inner loop, and everything in the rest of this act follows from it. It is also a warning: "the policy stopped changing" is a weaker certificate than "the value converged", and only the second one bounds how suboptimal you might still be.

Policy improvement — greedy on your own value

Given V^π , ask at each state whether a *single-step* deviation would pay:

$$\pi'(s) = \arg \max_a Q^\pi(s, a) \implies Q^\pi(s, \pi'(s)) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

That inequality is cheap — it says *one* deviation, then back to π , is no worse. The theorem is the leap from there to *always*:

POLICY IMPROVEMENT THEOREM

If $\pi'(s) = \arg \max_a Q^\pi(s, a)$ then $V^{\pi'}(s) \geq V^\pi(s)$ for every s , with equality **at all states** only when π is already greedy with respect to its own value — **which is exactly the Bellman optimality condition**.

The proof (Appendix, Backup 3) is a telescope: substitute the inequality, expand one step, substitute again. Each pass converts one more step from π to π' and the inequality never turns round. Two consequences. Improvement **can never hurt**; and since a finite MDP has finitely many deterministic policies and each nonterminal round strictly improves at least one state (keep the old action when it ties for best), **policy iteration terminates exactly**.

Policy iteration, and its bottleneck

$$\pi_0 \xrightarrow{\text{PE}} V^{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} V^{\pi_1} \xrightarrow{\text{PI}} \pi_2 \longrightarrow \dots \longrightarrow \pi^*$$

Alternate the two moves until the policy stops changing. Each round gives an *exact* value for an *exact* policy, and finite termination follows with consistent tie handling; the number of rounds need not be small.

THE BOTTLENECK

Every arrow marked PE hides a whole iterative solve — many sweeps over the entire state set. The source deck flags it bluntly: **policy evaluation is slow to converge.**

THE MITIGATION

Each evaluation **warm-starts from the previous policy's value**, and consecutive policies are similar, so later solves are far cheaper than the first.

And the previous slide has already told us the deeper answer: since the greedy step only needs the values *ranked*, running the inner loop to convergence is buying precision we throw away.

Value iteration — one optimality backup per sweep

Policy iteration evaluates a fixed policy before improving it. Value iteration directly applies the **optimality** backup:

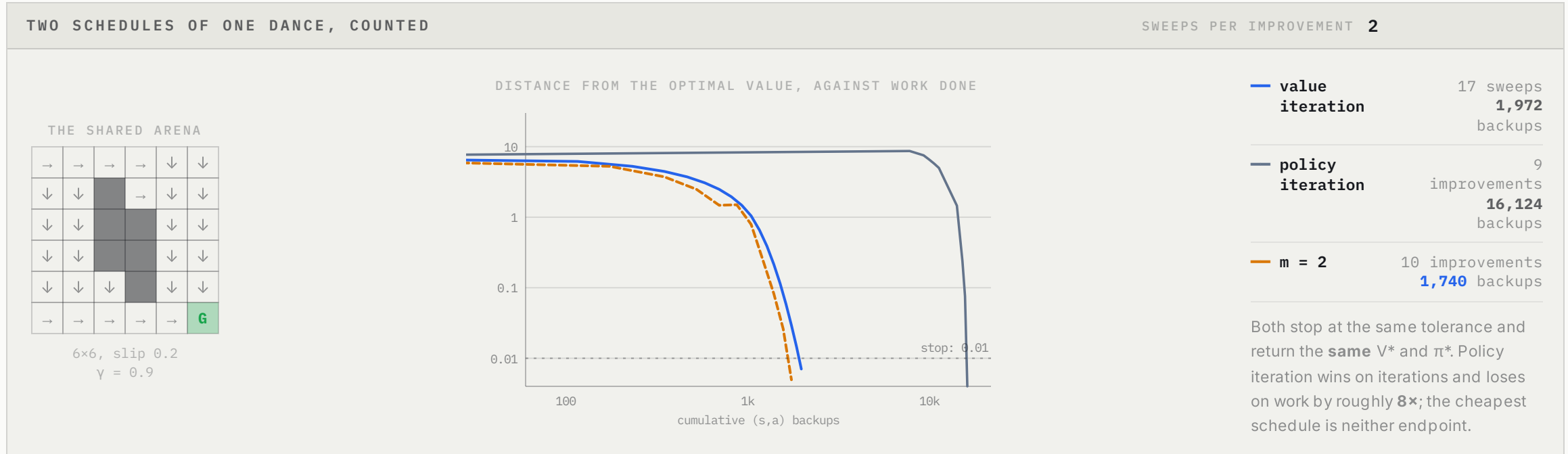
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')].$$

1. Use V_k to score every action with a one-step lookahead.
2. Store the largest score as $V_{k+1}(s)$.
3. Repeat; then read off a greedy policy from the final values.

This is the Bellman optimality equation used as an **update rule**. The two-state example just performed it twice.

Lecture 8 replaces the model-weighted expectation with sampled transitions and incremental updates. A neural network is a later extension; tabular Q-learning comes first.

Two schedules, counted



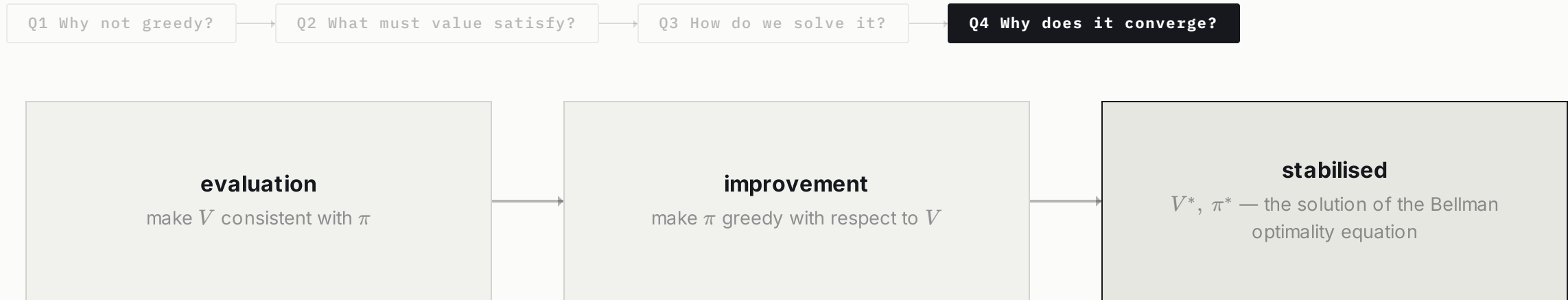
One MDP, one stopping rule, both algorithms — and a dial for the number of evaluation sweeps per improvement. At $m = \infty$ the schedule *is* policy iteration; drive m down and it slides continuously toward value iteration. Policy iteration reaches π^* in **9 improvements** but spends **16,124 backups** getting there; value iteration needs **17 sweeps** and only **1,972**. Same V^* , same π^* , an eightfold difference in work — and the cheapest schedule, at $m = 2$, is **neither endpoint**.

— ACT 4

Why it works

Q4. Two processes that keep undoing each other, and an answer they nevertheless agree on.

Generalised policy iteration — the dance behind both



Each move breaks the other. Making π greedy makes V wrong for the new π ; re-evaluating makes π no longer greedy. The source deck draws this as two lines in the plane of (V, π) — the locus $V = V^\pi$ and the locus $\pi = \text{greedy}(V)$ — with the algorithm [zigzagging between them in ever shorter hops](#).

The optimal **value function** is unique. Several optimal policies can share it when actions tie. Policy iteration and value iteration are different algorithms organised by the same evaluation–improvement idea.

Why the meeting point is unique — the contraction

The **Bellman optimality operator** $\mathcal{T}$:

$$(\mathcal{T}V)(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$$

Value iteration is nothing but $V_{k+1} = \mathcal{T}V_k$. And $\mathcal{T}$ is a **γ -contraction in the sup-norm**:

$$\|\mathcal{T}V - \mathcal{T}V'\|_\infty \leq \gamma \|V - V'\|_\infty$$

Two value functions disagreeing by δ are pushed to within $\gamma\delta$. By Banach's fixed-point theorem, $\gamma < 1$ buys three things at once:

- a **unique** fixed point V^* — one answer, not a family;
- **convergence from any** V_0 , with no condition on the initialisation;
- a **rate**: $\|V_k - V^*\|_\infty \leq \gamma^k \|V_0 - V^*\|_\infty$.

γ is not a modelling afterthought bolted on to keep a sum finite. It is **the number that makes the whole machine well-posed**, and it is also its convergence rate.

What the contraction bound actually says

Suppose $\gamma = 0.8$ and the initial maximum value error is at most 10.

$$\|V_k - V^*\|_\infty \leq 10(0.8)^k.$$

SWEEPS	GUARANTEED ERROR AT MOST
1	8
5	3.277
10	1.074
21	0.0923

This is an **upper bound**, not a prediction that every observed error follows the same curve. With $\gamma = 1$, this contraction argument no longer applies.

A practical certificate uses the Bellman residual: $\|V - V^*\|_\infty \leq \|\mathcal{T}V - V\|_\infty / (1 - \gamma)$. A small update matters relative to $1 - \gamma$.

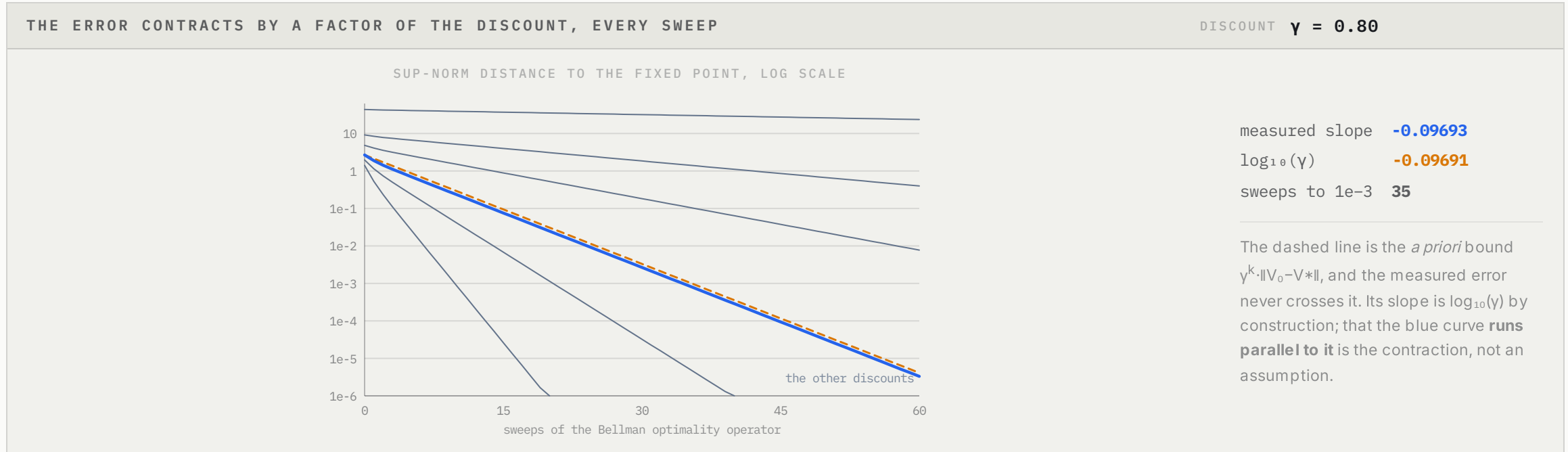
Check — what value iteration is really doing

For a finite MDP with bounded rewards and $0 \leq \gamma < 1$, value iteration converges from any finite V_0 . What guarantees that?

- ☐ A The objective is convex in V
- ☐ B The state space is finite
- ☒ C The Bellman operator is a γ -contraction in the sup-norm, so each sweep shrinks the distance to the unique fixed point by a factor γ
- ☐ D The rewards are bounded, so V cannot diverge

The Banach fixed-point theorem does the whole job. Each application of the operator brings any two value functions closer by a factor no greater than γ , which forces both a **unique** fixed point and convergence to it from anywhere. The error after k sweeps decays like γ^k — which is also why a γ near 1 is not free, and why Lecture 8 has to worry when the operator is no longer a contraction.

The rate is the discount



$\|V_k - V^*\|_\infty$ against sweep number, on a log axis. In this demonstration, the measured decay is close to a straight line of slope $\log_{10} \gamma$. The theorem guarantees only that the curve remains below the dashed *a priori* bound $\gamma^k \|V_0 - V^*\|$. Move the dial and watch it tilt. Reaching an error of 10^{-3} takes 10 sweeps at $\gamma = 0.5$ and more than 60 at $\gamma = 0.9$:
far-sightedness is not free, and γ is the invoice.

Order does not matter either — asynchronous DP

Every sweep so far updated all states in lockstep. Nothing required that.

- **In-place:** overwrite $V(s)$ immediately, and reuse it within the same sweep.
- **Asynchronous:** back states up in **any order whatsoever**, using the latest available values (or delayed values whose information does not remain permanently stale) — repeating some, skipping others. Provided every state is updated **infinitely often**, $V \rightarrow V^*$ still holds.

The motivation is brute size. Backgammon has about 10^{20} states and Go has $3^{19 \times 19}$; a single full sweep over either will not finish, and even one complete sweep is prohibitively expensive. Partial backups can still provide useful information.

THE SOURCE DECK'S OWN ARROW FORWARD, EXCLAMATION MARKS INCLUDED

"Make it easier to intermix computation with real-time interaction: to solve a given MDP, we can run the iterative DP algorithm at the same time that an agent is actually experiencing the MDP **(Reinforcement Learning !!!!)**"

Sampled RL is asynchronous DP driven by whichever states the trajectory happens to visit. That is not an analogy — the "visit every state infinitely often" condition in Lecture 8's convergence theorem **builds on this slide's condition**, and also needs suitable learning rates and coverage of every state–action pair.

Three things we have not paid for

The machine is complete and provably correct. Every part of it borrows something.

WHAT VALUE ITERATION DOES	WHAT IT NEEDS	WHAT HAPPENS WITHOUT IT
$\sum_{s'} T(s, a, s')[\dots]$	the transition model T	the expectation cannot be evaluated
$\arg \max_a \sum_{s'} T[R + \gamma V(s')]$	T again, for the lookahead	V alone cannot name an action
one entry $V(s)$ per state	a table the size of $\mathcal{S}$	no storage, and no generalisation to unseen states

Three debts, and the next lecture defaults on all three. Notice they are not independent: the first two are the same debt — the model — appearing once in evaluation and once in improvement. The third is why the last act of Lecture 8 exists at all.

Check — what is still being assumed

Value iteration and policy iteration both solve the MDP exactly. What do they need that the next lecture will not have?

- ☐ A A finite horizon
- ☐ B A deterministic policy
- ☐ C A discount factor strictly less than 1
- ☒ D The model — the transition probabilities P and the reward function R , needed to compute the expectation

Every backup in this lecture evaluates $\sum_{s'} P(s' | s, a)[\dots]$, which you can only write down if you own P and R . Lecture 8 assumes the full model is unavailable and uses observed transitions, so the question becomes: how do you take an expectation you cannot compute? Sampling the backup is the starting point for the value-based methods in Lecture 8.

— CLOSING

Closing

The equation is proved, the solvers are exact, and everything rested on owning the world.

Where we are — one parent established

	LINEAGE A • OR / DYNAMIC PROGRAMMING	LINEAGE B • CONTROL THEORY
MODEL-BASED (ORIGIN)	MDP & Dynamic Programming LECTURE 7	Optimal Control / Planning LECTURE 9
DATA-DRIVEN (EXTENSION)	Value-Based RL LECTURE 8	Policy-Based RL LECTURE 10

We hold the Bellman equation, two exact solvers, and a convergence guarantee — and every line of it leaned on **knowing T and R** .

Two roads lead out, and the course takes both.

DOWN — LECTURE 8

Keep the equation, **delete the model**. Replace the expectation with a sample, V with Q , the table with a network. Value-based reinforcement learning: this lecture with (T, R) removed.

RIGHT — LECTURE 9

The *other* parent. Control theory solves the same dynamic problem through **continuous-control examples with dynamics f** , instead of today's finite tables — and Lecture 10 will delete f exactly as Lecture 8 deletes (T, R) .

What Lecture 8 receives

This lecture hands on [the Bellman equation, which still needs the model](#). In the form the next lecture opens with:

$$Q^*(s, a) = \sum_{s'} P(s' | s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

And the objects that solve it, each with a sampled successor waiting:

WHAT WE HAND OVER	BUILT IN	WHAT LECTURE 8 DOES TO IT
the expectation $\sum_{s'} P(s' s, a)[\dots]$	Act 2	one sampled transition (s, a, r, s')
policy evaluation, solved with T, R	Act 3	MC returns, or a TD bootstrap
greedy on V : $\arg \max_a \sum_{s'} P[\dots]$	Acts 2–3	greedy on Q — the cache, now essential
value iteration	Act 3	Q-learning: the same $\max$, sampled
the full backup, all successors	Act 3	a sample backup, one successor
a table, exact, one entry per state	Acts 3–4	a function $\hat{Q}(s, a; w)$

Column three is the whole of Lecture 8. Not one row changes the *principle* — only [what we may look up](#).

A sequence of decisions seemed to need a plan for
every possible future.

Bellman showed it needs only a **value for every state** — because an optimal plan, by its nature, stays optimal from wherever you happen to land.

Questions?

Value, defined recursively, turns an intractable search over plans into local backups over states. It is the founding move of operations-research decision making, the equation Lecture 8 keeps after throwing the model away, and — read one column to the right — the problem Lecture 9 will solve all over again in a different language.

— APPENDIX

Backup slides

The derivations, kept out of the narrative.

Reading guide — understand the backup before the algorithm

one main idea to explain, one comparison, one application

ROLE	READ OR REVISIT	QUESTION TO ANSWER
Core	Sutton & Barto, <i>Reinforcement Learning: An Introduction</i>, 2nd ed. (2018), Chapters 3–4	How does one expected return become a Bellman backup?
Compare	Policy iteration versus value iteration on the same MDP	Which operation evaluates a policy, and which improves it?
Apply	The original gridworld, with the same transition and reward conventions	How many backups are needed before the preferred action changes?

Calculate one backup and two sweeps before reading the convergence proof. The figures and widgets use the same mathematical objects; a newer research paper is not needed to replace this foundation.

Backup 1—the Bellman equation, derived

Start from the definition and peel off one step, using linearity of expectation and the Markov property:

$$\begin{aligned}
 V^\pi(s) &= \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k r_{t+k+1} \mid S_t = s \right] = \mathbb{E}_\pi \left[r_{t+1} + \gamma \sum_{k \geq 0} \gamma^k r_{t+2+k} \mid S_t = s \right] \\
 &= \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k r_{t+2+k} \mid S_{t+1} = s' \right] \right] \\
 &= \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right] \quad \blacksquare
 \end{aligned}$$

The third line is where the Markov property is spent: the inner expectation depends on the whole history only through s' , so it *is* $V^\pi(s')$.

Optimality form. Maximise over the first action and use the principle of optimality for the remainder: $\pi(s)$ becomes $\max_a$ and V^π becomes V^* . **Q form.**

Condition on (s, a) before the transition instead of after it; the $\max$ then lands at the successor, $\max_{a'} Q^*(s', a')$, which is why V^* maxes before the expectation and Q^* after.

Backup 2 — the contraction proof

Claim. $\|\mathcal{T}V - \mathcal{T}V'\|_\infty \leq \gamma\|V - V'\|_\infty$, where $(\mathcal{T}V)(s) = \max_a \sum_{s'} T(s, a, s') [R + \gamma V(s')]$.

For any state s , using $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$:

$$\begin{aligned} |(\mathcal{T}V)(s) - (\mathcal{T}V')(s)| &\leq \max_a \left| \sum_{s'} T(s, a, s') \gamma [V(s') - V'(s')] \right| \\ &\leq \gamma \max_a \sum_{s'} T(s, a, s') |V(s') - V'(s')| \leq \gamma \|V - V'\|_\infty \end{aligned}$$

since $\sum_{s'} T(s, a, s') = 1$. Taking the maximum over s gives the claim. ■

Consequence (Banach). For $\gamma < 1$, $\mathcal{T}$ has a unique fixed point V^* , and $V_{k+1} = \mathcal{T}V_k$ converges to it from any V_0 with $\|V_k - V^*\|_\infty \leq \gamma^k \|V_0 - V^*\|_\infty$. The max inequality is the only step that needs care. Finiteness ensures maxima are attained; analogous suprema also work under suitable continuous-space assumptions — Lecture 9 works in a setting where that maximum is over a continuum. Q-learning combines this contraction with stochastic-approximation, coverage and step-size conditions (Lecture 8, Backup 2).

Backup 3 — policy improvement never hurts

Theorem. Let $\pi'(s) = \arg \max_a Q^\pi(s, a)$. Then $V^{\pi'}(s) \geq V^\pi(s)$ for all s .

By construction $Q^\pi(s, \pi'(s)) = \max_a Q^\pi(s, a) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$. Now telescope, substituting that same inequality at each successor:

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s] \\ &\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s] = \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = s] \\ &\leq \cdots \leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s] = V^{\pi'}(s) \quad \blacksquare \end{aligned}$$

Each pass converts one more step of the trajectory from π to π' , and the inequality never turns round. **Why policy iteration terminates.** A finite MDP has finitely many deterministic policies, and each improvement is strict unless π is already greedy with respect to its own value — which is precisely the Bellman optimality condition. So π^* is reached in finitely many rounds, not merely in the limit.

Backup 4 — asynchronous and in-place DP

In-place value iteration. Keep one array; overwrite $V(s)$ immediately and let later states in the same sweep read the new number. It converges, and usually faster, because fresh information propagates within a sweep instead of waiting for the next one. The sweep order now matters for *speed* — never for the answer.

Asynchronous DP. Back states up in any order, repeating some and skipping others. As long as every state is updated infinitely often, $V \rightarrow V^*$. This decouples the computation from any rigid sweep and is what lets DP be *prioritised* — order the backups by how large their Bellman residual was last time (prioritised sweeping), and effort follows the states where the value is still wrong.

Why it matters downstream. Sampled RL is asynchronous DP whose update order is chosen for it by the environment: Q-learning backs up whatever (s, a) the trajectory delivers, in whatever order experience delivers them. The "every state infinitely often" condition here is the model-based ancestor of the exploration requirement in Lecture 8 — and the reason ε must not be allowed to reach zero too soon.

Where the two lineages split

the source deck's own taxonomy, pp. 4–5

MODEL BASED – T AND R GIVEN

	FINITE ACTIONS	INFINITE ACTIONS
discrete time	MDP (Lec 7)	dynamic system (Lec 9)
continuous time	continuous-time MDP	continuous-time system

$$P(s_{t+1} \mid s_t, a_t) \cdot x_{t+1} = f(x_t, u_t), \dot{x}_t = f(x_t, u_t)$$

These columns describe **the examples chosen for this course**: finite-action MDPs in Lecture 7 and continuous-control systems in Lecture 9. MDPs can also have continuous spaces; policy methods can also handle discrete actions. The common question is how to choose actions over time.

Lecture 9 is the mirror of this lecture, not its sequel. Lecture 8 sits directly below it; Lecture 10 directly below Lecture 9.

MODEL FREE – LEARNED FROM EXPERIENCE

	FINITE ACTIONS	INFINITE ACTIONS
discrete time	value-based RL (Lec 8)	policy-based RL (Lec 10)
continuous time	—	—

The empty cells are outside this course’s scope. Continuous-time reinforcement learning also exists.