上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 34: Model-Based Reinforcement Learning

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.

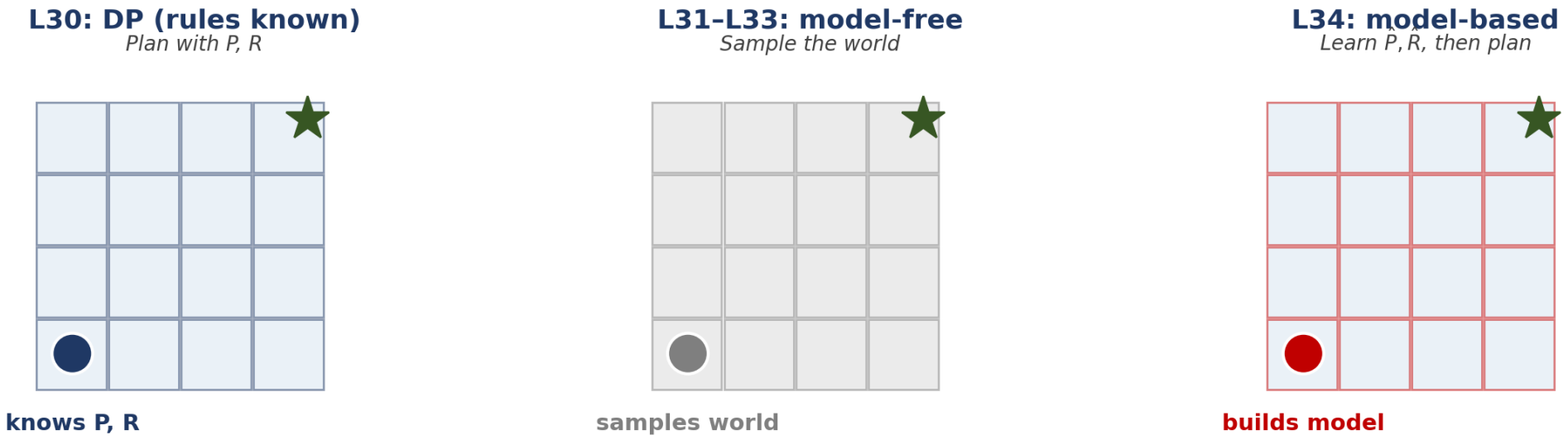
From Sampling Worlds to Building Them

L29–L33 learned to act in a world you can't see inside. L34 builds the world, then plans inside it.

L30 DP — knew the rules (P, R), solved exactly. Sample-free but unrealistic.

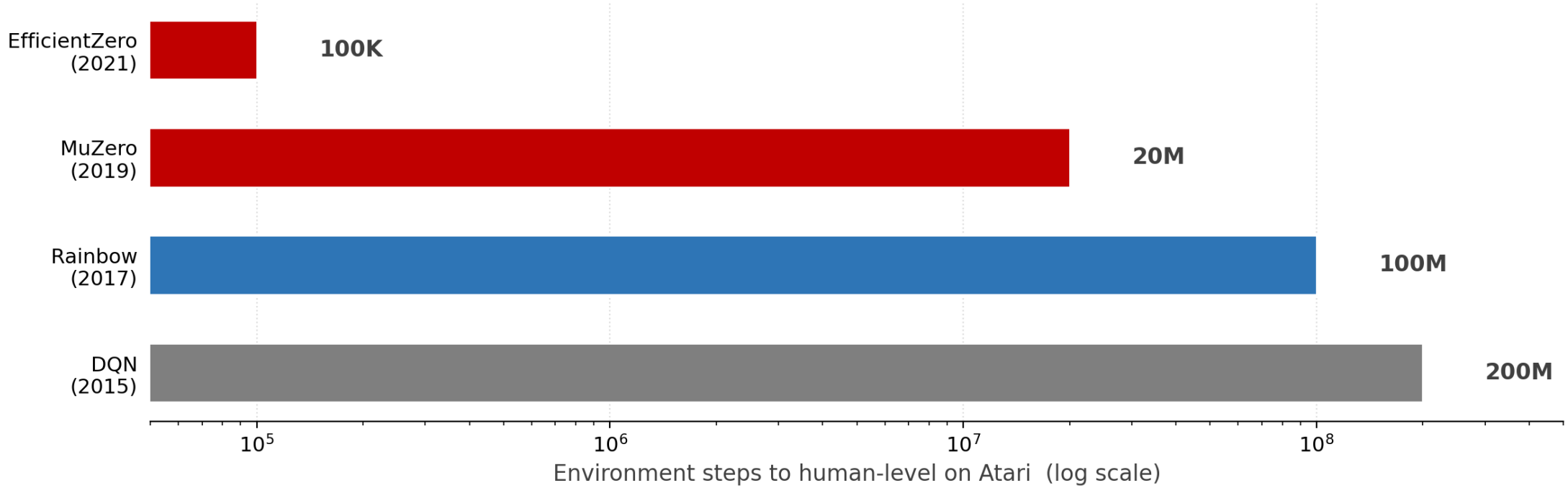
L31–L33 model-free — didn't know the rules, learned from experience. Realistic but sample-hungry.

L34 model-based — don't know the rules — learn them, then plan.



Two Questions L33 Left Us with

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



Two orders of magnitude. That's what a model buys you.

Objectives

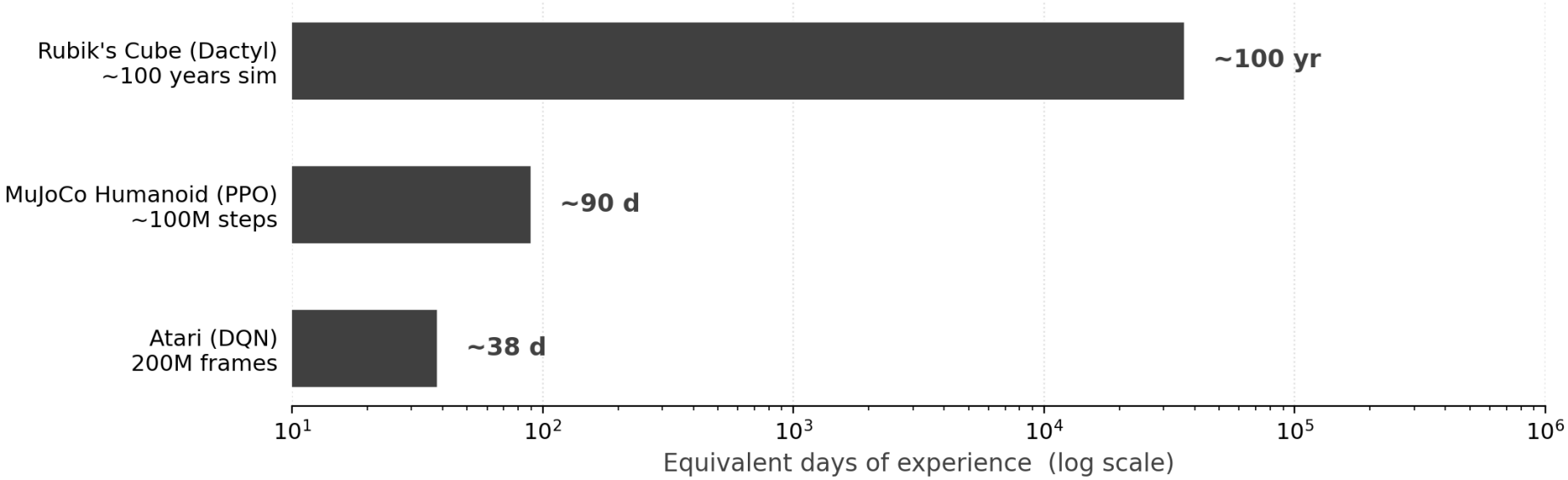
By the end of this lecture, you will be able to:

- Explain why model-based RL is the third family alongside value-based and policy-based.
- Derive the Dyna-Q update and identify which line is the “planning” step.
- Distinguish model-as-simulator from model-as-planner, with one example of each.
- Sketch the MuZero architecture and explain the role of its three networks.
- Evaluate when to reach for model-based RL — and when not to.
- Connect world models to the agentic systems that come next.

1. The Sample-Efficiency Problem

Model-free RL works. So what's wrong?

- **Situation:** DQN solved Atari. PPO solved locomotion. RLHF aligned ChatGPT.
- All three are *model-free* — sample the world, update the policy.
- **One brutal cost:** millions to billions of environment interactions.



In simulation, this is fine. On a real robot, this is fatal.

Complication: The Real-World Sample Cliff

In the real world, one interaction = one second of robot wear, one battery cycle, one shattered cup.

Simulator (cheap)

- 10^6 samples / hour
- Robot can fail freely
- ***Model-free works***

Real robot (expensive)

- 10^2 samples / hour
- Every sample costs hardware
- ***Model-free is hopeless***

Sample efficiency isn't a nice-to-have. It's the difference between RL on a screen and RL in the world.

Where Does The Budget Go?

What does the agent throw away after each interaction?

It throws away...

The transition.

(s, a) → s', r was observed once and used for a single TD update.

It throws away...

The structure.

The world has physics; the agent only saw a point sample.

It throws away...

The ability to think.

No simulation, no rollout, no “what if?” before acting.

A model is what you build when you want to keep all three.

The World Model, Formally

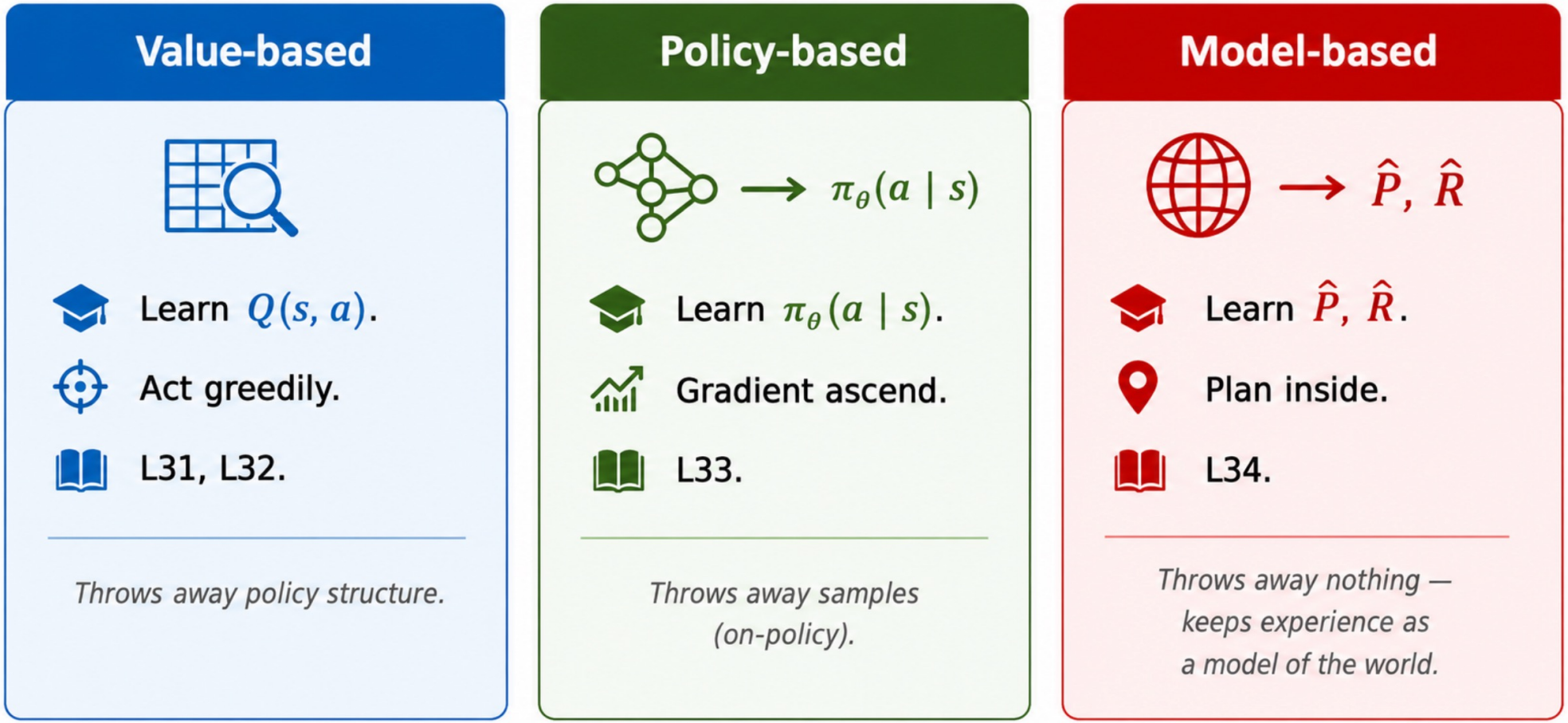
- A world model is a learned approximation of the MDP's dynamics:

$$\hat{P}_\phi(s' \mid s, a), \quad \hat{R}_\phi(s, a) \quad \approx \quad P(s' \mid s, a), \quad R(s, a)$$

Learned dynamics, parameterized by ϕ.

Once you have $\hat{P}$ and $\hat{R}$, you have a learned MDP. Now you can apply DP from L30 — inside your head, no environment needed.

The Three Families of RL



Think: Where to Use Model-based RL?

For each setting, would you choose model-free or model-based — and why?

Discuss:

(a) A simulated game environment (e.g., Atari).

(b) A surgical robot trained on a real patient mockup.

(c) An LLM being fine-tuned with RLHF on offline preference data.

2. Dyna-Q: The Gentle On-Ramp

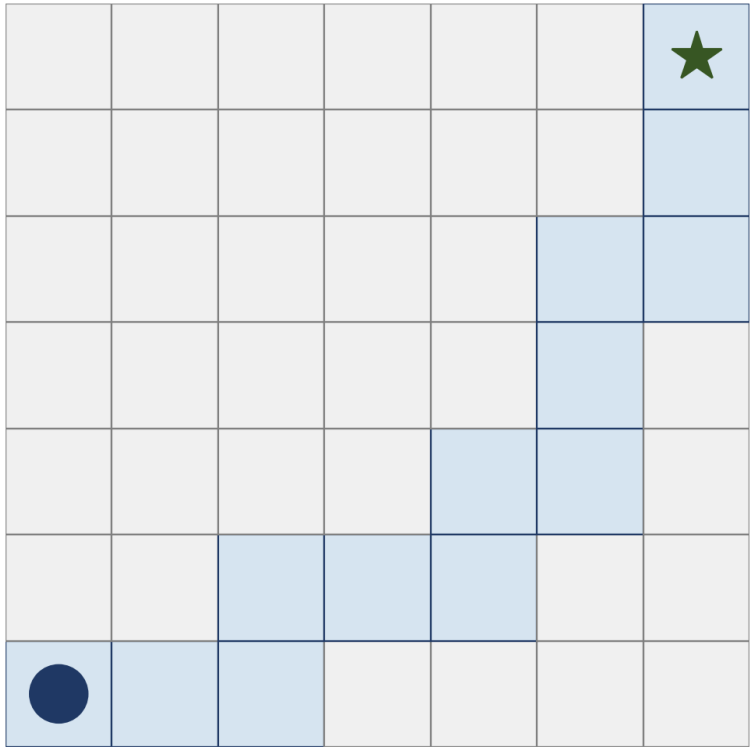
Q-Learning Uses Each Transition Once

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

One TD update per real transition.

The agent learns slowly because each interaction touches Q at one (s, a) pair.

We have all this experience — *and we used it once.*



Only path cells get TD updates.
80% of the grid is invisible to learning.

Why Throw Experience Away?

You watched a transition. Why not replay it — and replay imagined variations of it?

Option 1. Remember every (s, a, r, s') tuple and replay them.

→ that's *experience replay* (DQN, L32).

Option 2. Summarize them into a learned $\hat{P}$, $\hat{R}$.

→ now you can sample *new transitions you've never seen* — by querying the model.

Dyna-Q (Sutton, 1990) — The Algorithm

$$\textbf{(1) Act:} \quad a \sim \pi(\cdot \mid s), \quad \text{observe } r,\ s'$$

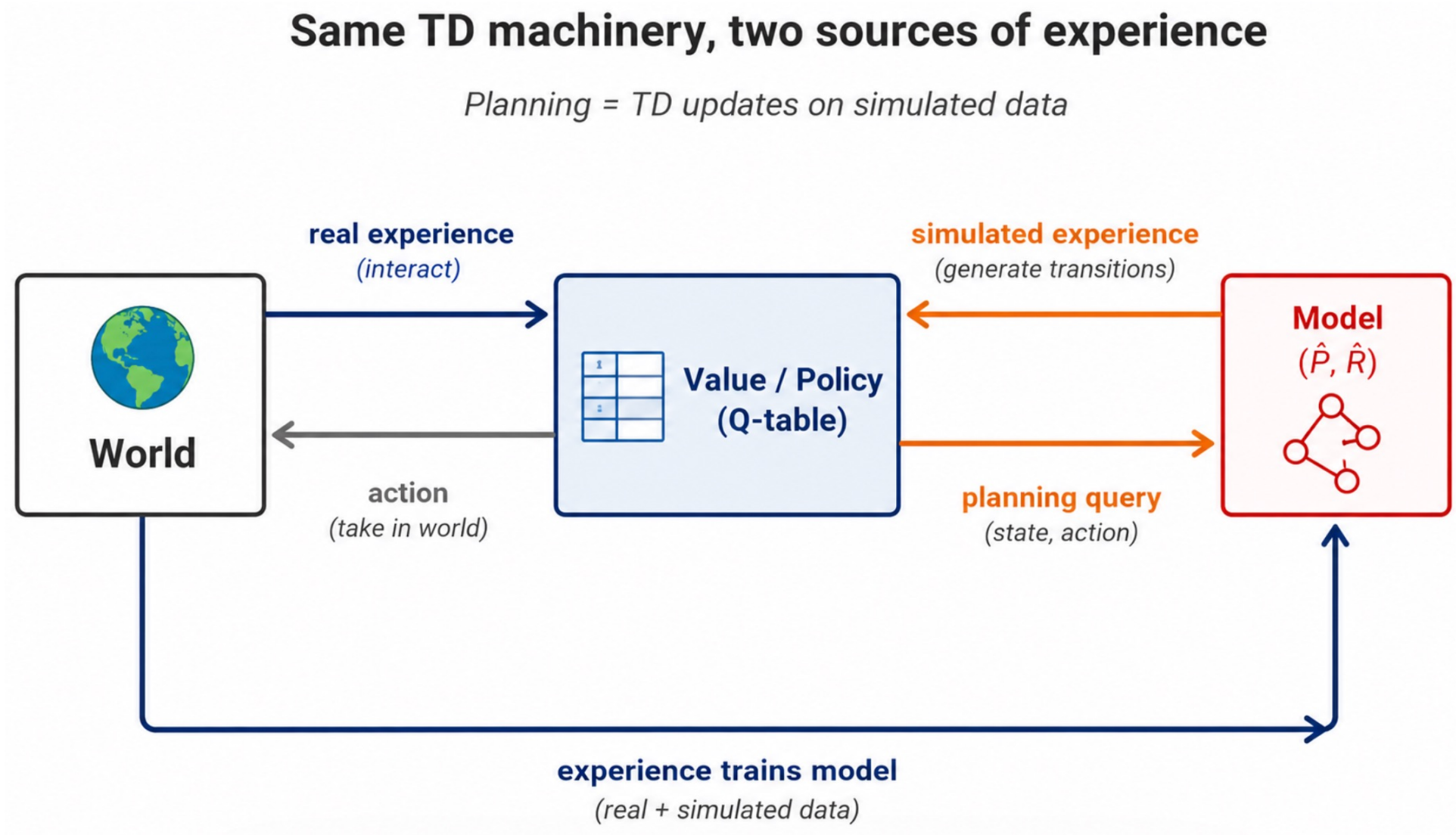
$$\textbf{(2) Direct RL:} \quad Q(s,a) \mathrel{+}= \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

$$\textbf{(3) Model learning:} \quad \hat{P}(s' \mid s,a) \leftarrow s', \quad \hat{R}(s,a) \leftarrow r$$

$$\textbf{(4) Planning} \times n\textbf{:} \quad (\tilde{s},\tilde{a}) \sim \text{seen};\ (\tilde{r},\tilde{s}') \sim \hat{P},\hat{R};\ Q(\tilde{s},\tilde{a}) \mathrel{+}= \alpha[\cdots]$$

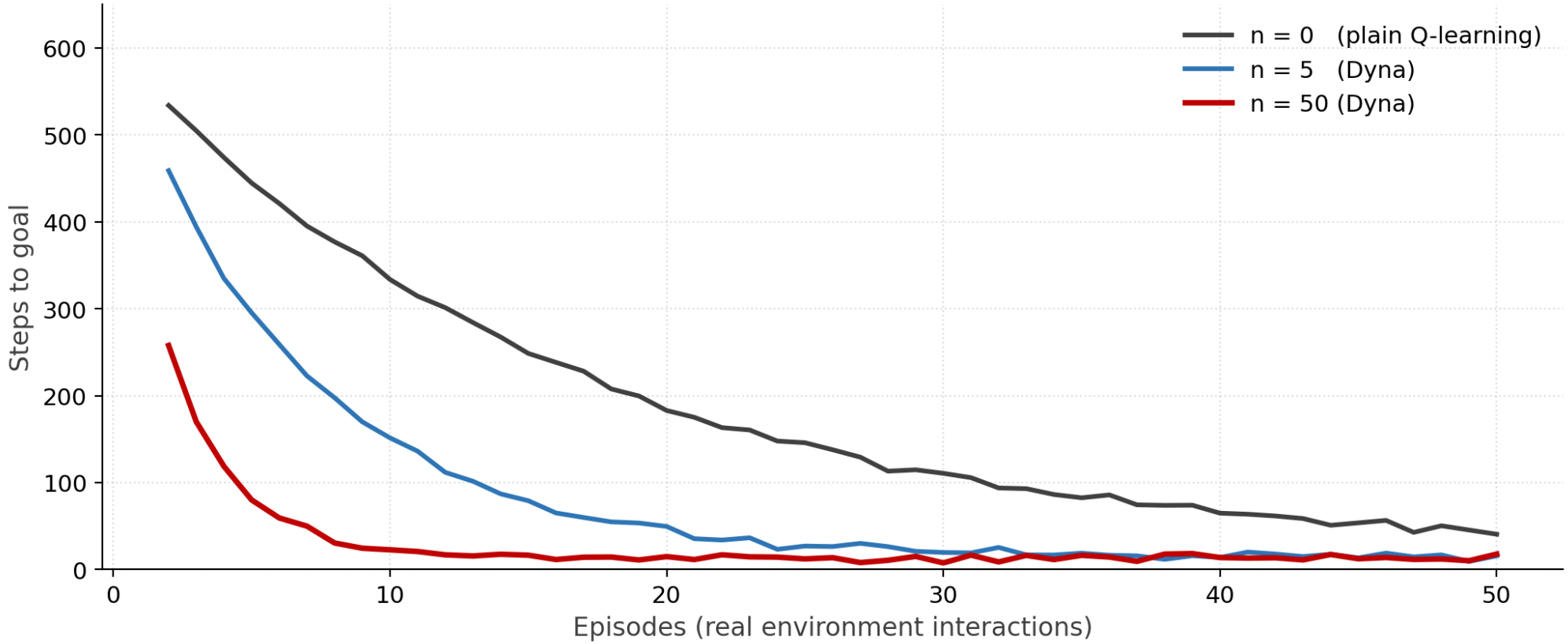
Lines (2) and (4) are the same TD update. The only difference is whether the transition came from the world — or the model.

Dyna-Q, Visualized



Planning Sweeps Win

Same real budget, ~10× faster to the goal.



n = 0 is plain Q-learning. n = 50 reaches the goal ~10× faster — with the same number of real interactions.

The two faces of a model

There are two distinct ways to use $\hat{P}$, $\hat{R}$.

Model as simulator

- Generate synthetic transitions.
- Run model-free RL on them.

Dyna-Q lives here.
Also: Dreamer.

Model as planner

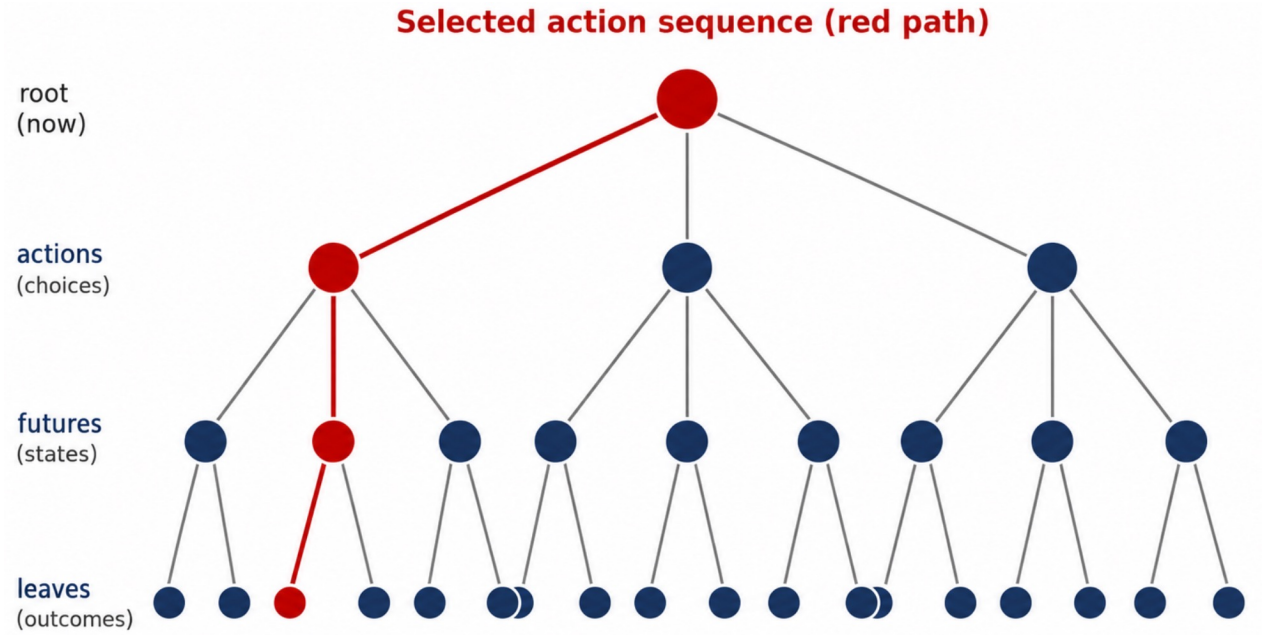
- Don't update a separate policy.
- Search the model directly for the best action.

MCTS lives here.
Also: MuZero, AlphaZero.

3. MuZero: Planning in a Learned Latent World

AlphaGo's Secret Was The Rules

- **L32** ended on AlphaGo. Its real magic wasn't DQN-like — it was ***Monte Carlo Tree Search*** through the *known* rules of Go.
- AlphaGo Zero / AlphaZero generalized this to chess, shogi, Go — *all games whose rules were given*.



What if the rules aren't given? Can the agent learn its own model and search inside that?

The Rules of Atari are Not Given

Three problems with running AlphaZero-style MCTS on Atari:

① The “rules” are millions of pixels and a hidden game state.

② You can't query P(s' | s, a) — you only get to play forward and observe.

③ Tree search over raw pixels would be intractable even if you could.

The fix: don't model the world in pixel space.
Model only what matters for planning.

MuZero (Schrittwieser et al., 2019)

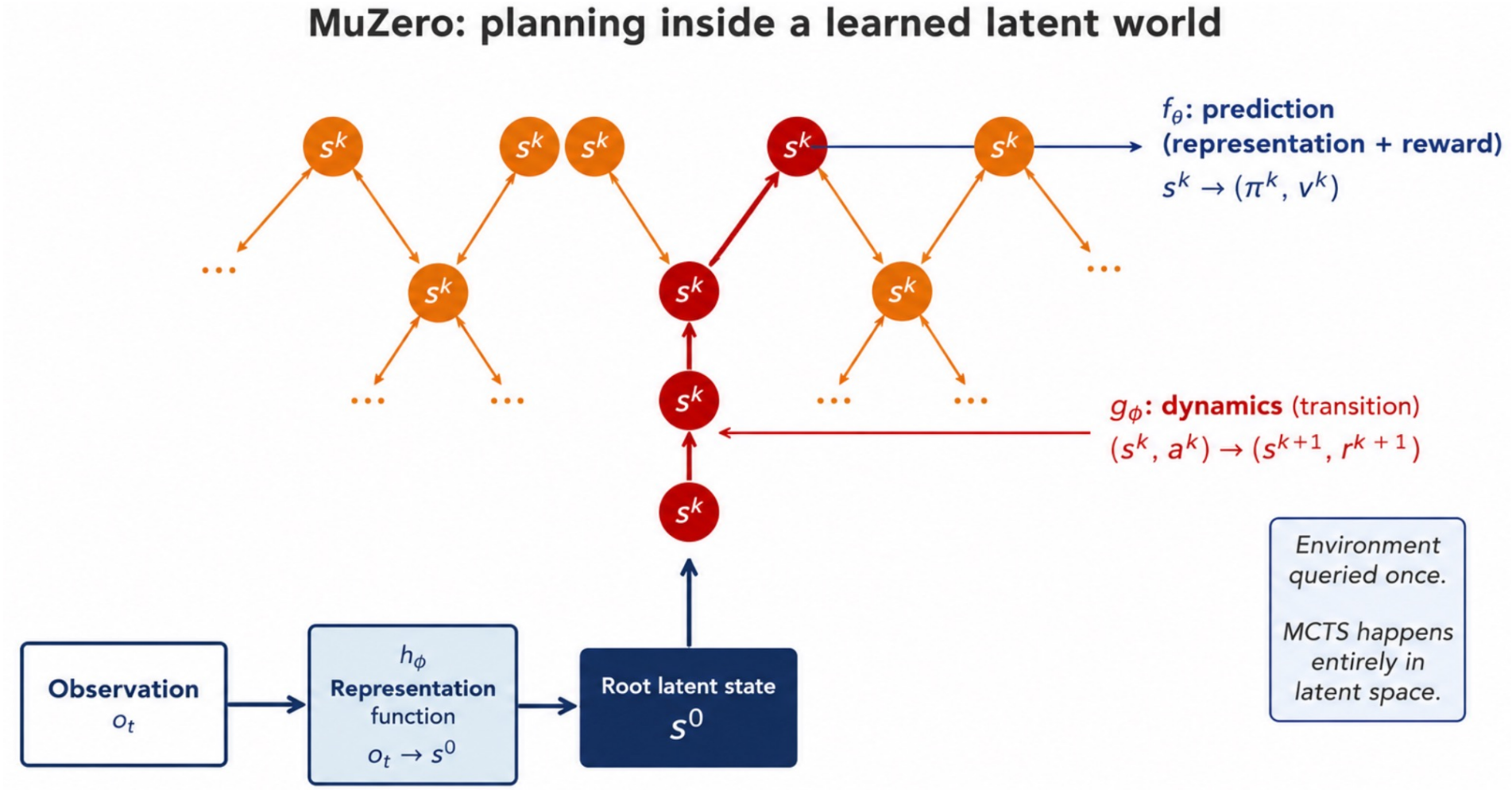
MuZero learns a model that doesn't predict the future — it predicts the things you need to know to plan.

Three networks, each with a deliberately mnemonic role:

- **Representation *h*** : real observation → latent state s^0
- **Dynamics *g*** : latent state + action → next latent state + immediate reward
- **Prediction *f*** : latent state → policy prior π + value v

Nothing predicts pixels. Everything predicts what MCTS needs.

The MuZero Loop

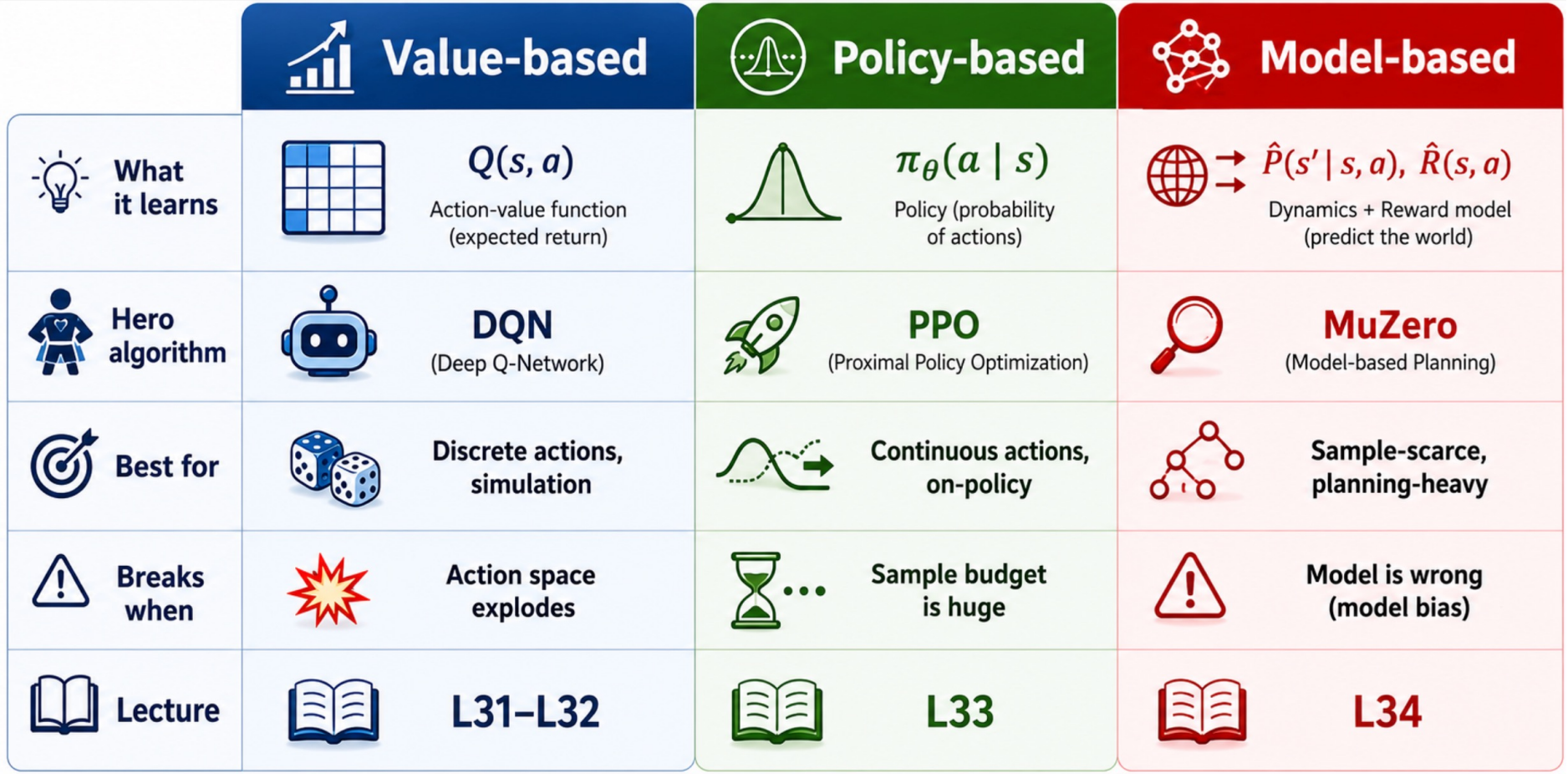


Why MuZero Works

- No pixel reconstruction loss.
  h, g, f are trained end-to-end on *policy targets, value targets, and reward targets* — never on next-observation reconstruction.
- The latent state is a learned, task-specific abstraction.
  It is whatever the network finds useful for predicting reward and value.
- Planner-shaped, not simulator-shaped.
  Compare Dreamer, which *does* try to reconstruct pixels — and uses its model as a simulator.

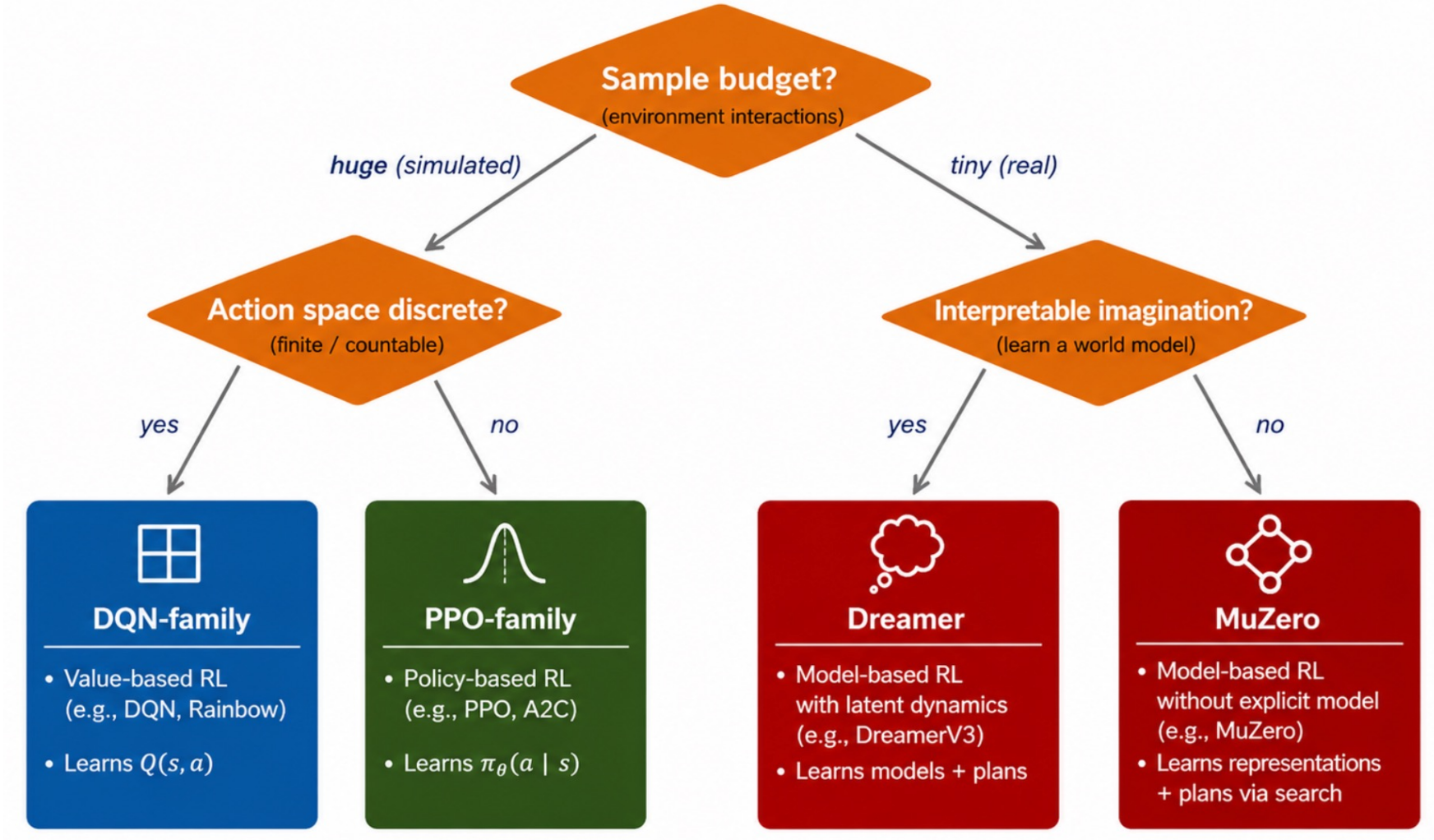
4. The Three Families

The RL Family Tree



One algorithm — TD updates — applied to one of three things you can learn.

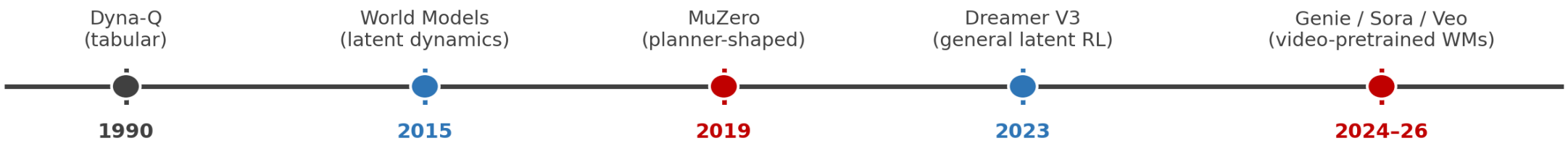
When to Reach for Which Family



No silver bullet. The constraint that binds tells you the family.

Frontier: World Models are Leaving RL

- Sora, Veo, Genie are world models in the model-based-RL sense — predict the future given an action — but trained on internet video, not on RL rollouts.
- Genie 3 can be controlled action-by-action and generates *playable worlds*.
- The MuZero question — *what should the latent state predict?* — is now the question of every modern video world model.



World models leave RL and meet generative video

RL Module Summary

- **L29 MDPs** — formalized the agent–environment loop.
- **L30 DP** — solved it exactly, *when you knew the rules*.
- **L31 TD / Q-learning** — dropped the rules but kept the value function.
- **L32 DQN** — scaled value functions to pixel-state worlds.
- **L33 Policy Gradient** — swapped Q for π, and led to RLHF.
- **L34 Model-based RL** — brought the rules back, learned this time.

RL is one algorithm — TD updates — applied to one of three things you can learn: a value, a policy, or a model.

Next: When The Model is an LLM

Week 13 — AI Agents. The model-based RL ideas in three new clothes:

- **Tool use** ≈ querying a learned environment model.
  (LLM + tools = Dyna-Q with a language-shaped model.)
- **Long-horizon reasoning** ≈ MCTS-style planning, written in language instead of latent vectors.
- **Autonomous systems** ≈ the full Dyna loop: imagine, act, observe, revise.

L34: we learned a model and planned inside it.
L35: the model is GPT — and "planning" is called "reasoning."