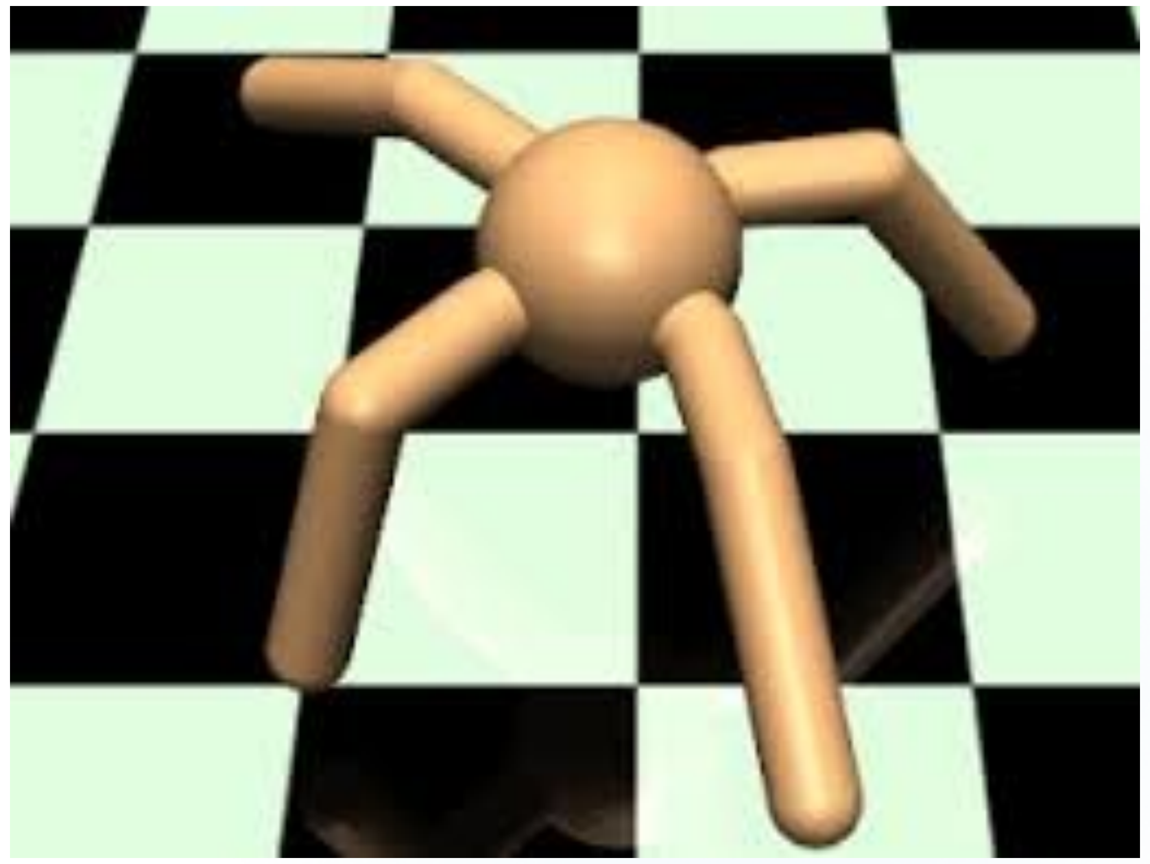


Reinforcement Learning

Motivation

Physics and Reinforcement Learning (RL)

Common ground:

- Both use simulators to train models.
- Both treat multi-step problems.
- Both assume the Markov property.

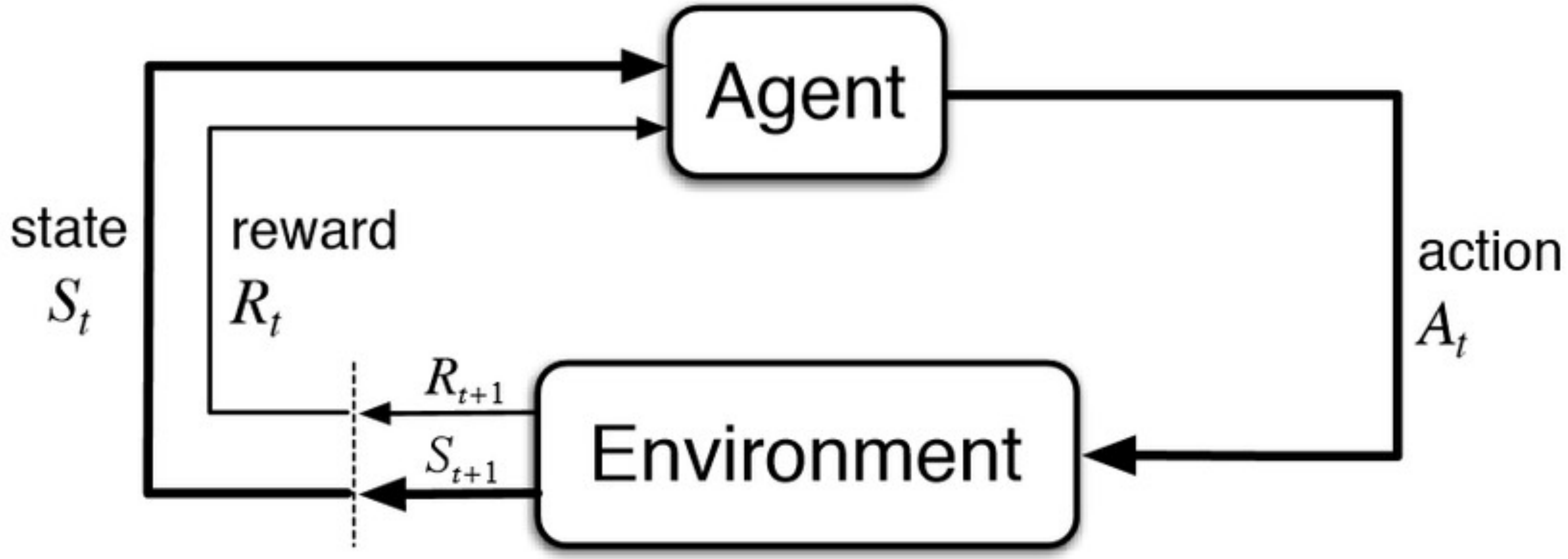
Part 1: RL Introduction

- RL Framework
- Value Functions
- RL Algorithms

Part 2: Ideas behind RL algorithms

RL Framework

RL Setup



Agent: learning and choosing actions.

Environment: responding by giving a reward and transitioning to a new state.

Episode: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots$

Policy: the rule followed by the agent to choose actions $\pi : S \rightarrow A$

Goal: Finding the policy that maximizes rewards

Rewards

Immediate Reward

+1 for not falling over (each step)

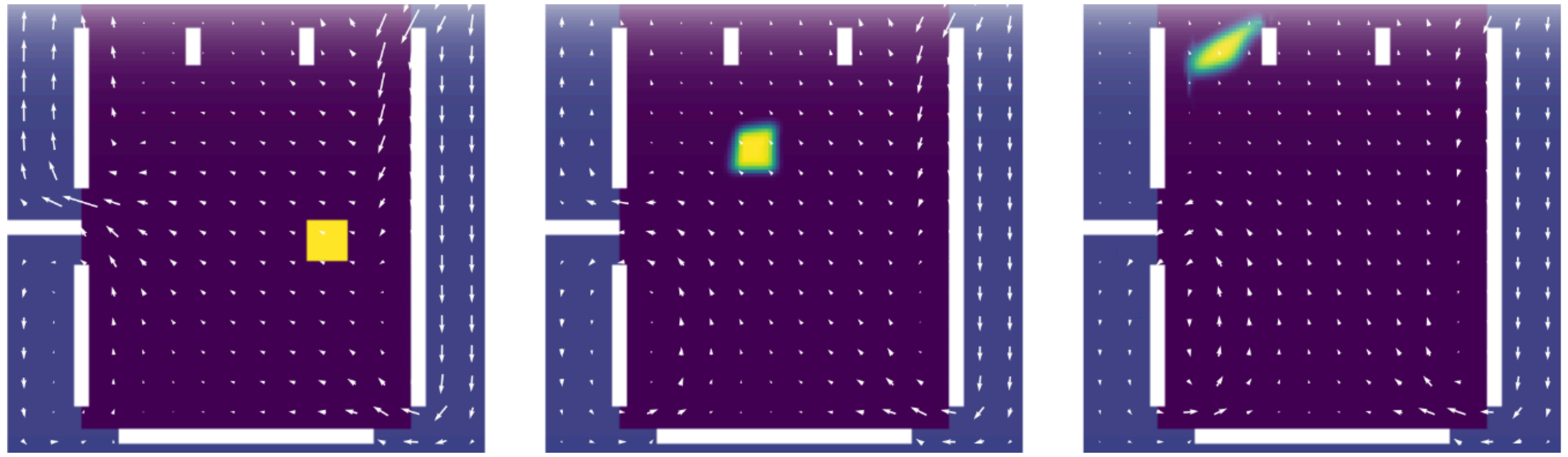
+d for the amount moved along the x-axis (each step)

Delayed Reward

+1/0/-1 for win/draw/lose (at the end)

Physics Tasks in the RL Framework

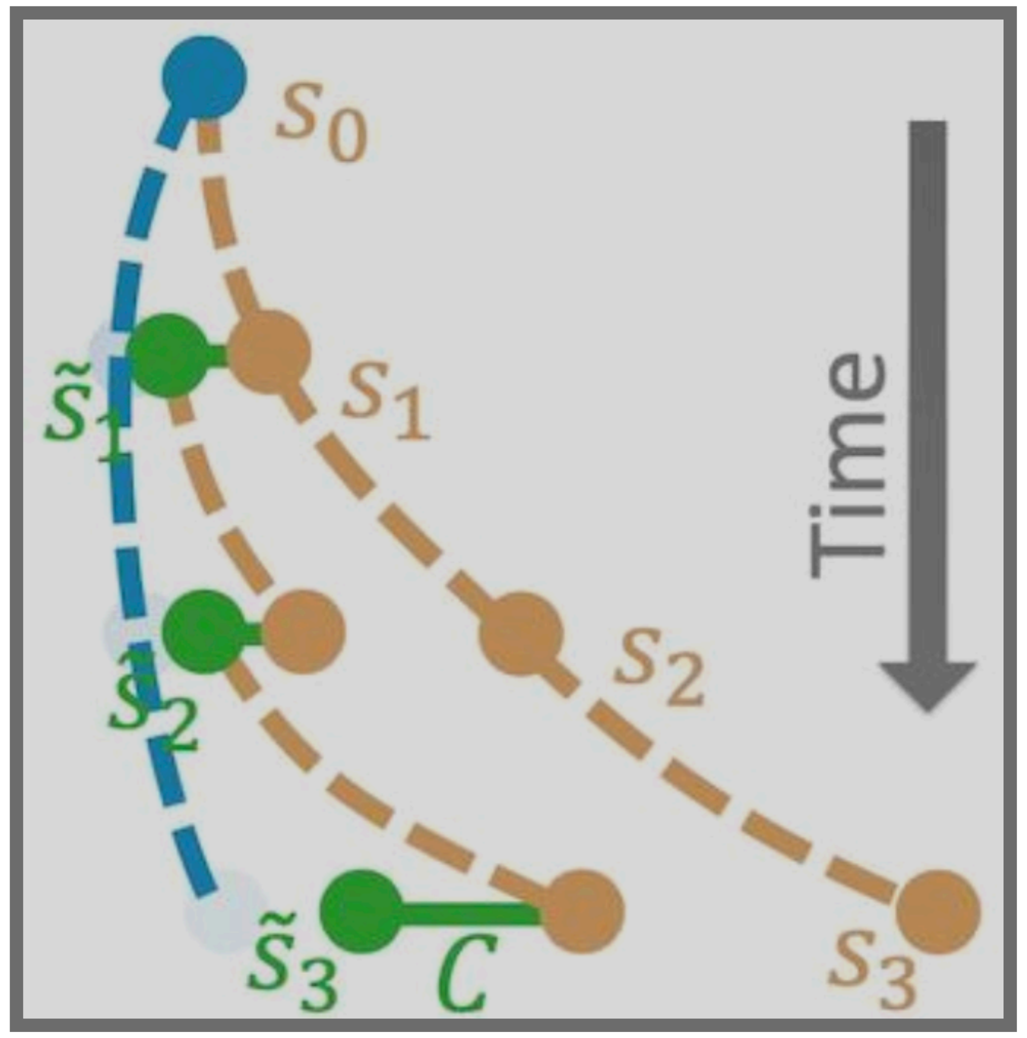
Fluid Control



Define a loss to measure if the object is moving through the left gate.

Reward is the negative loss.

Error Correction



Actions are the state change applied after each step

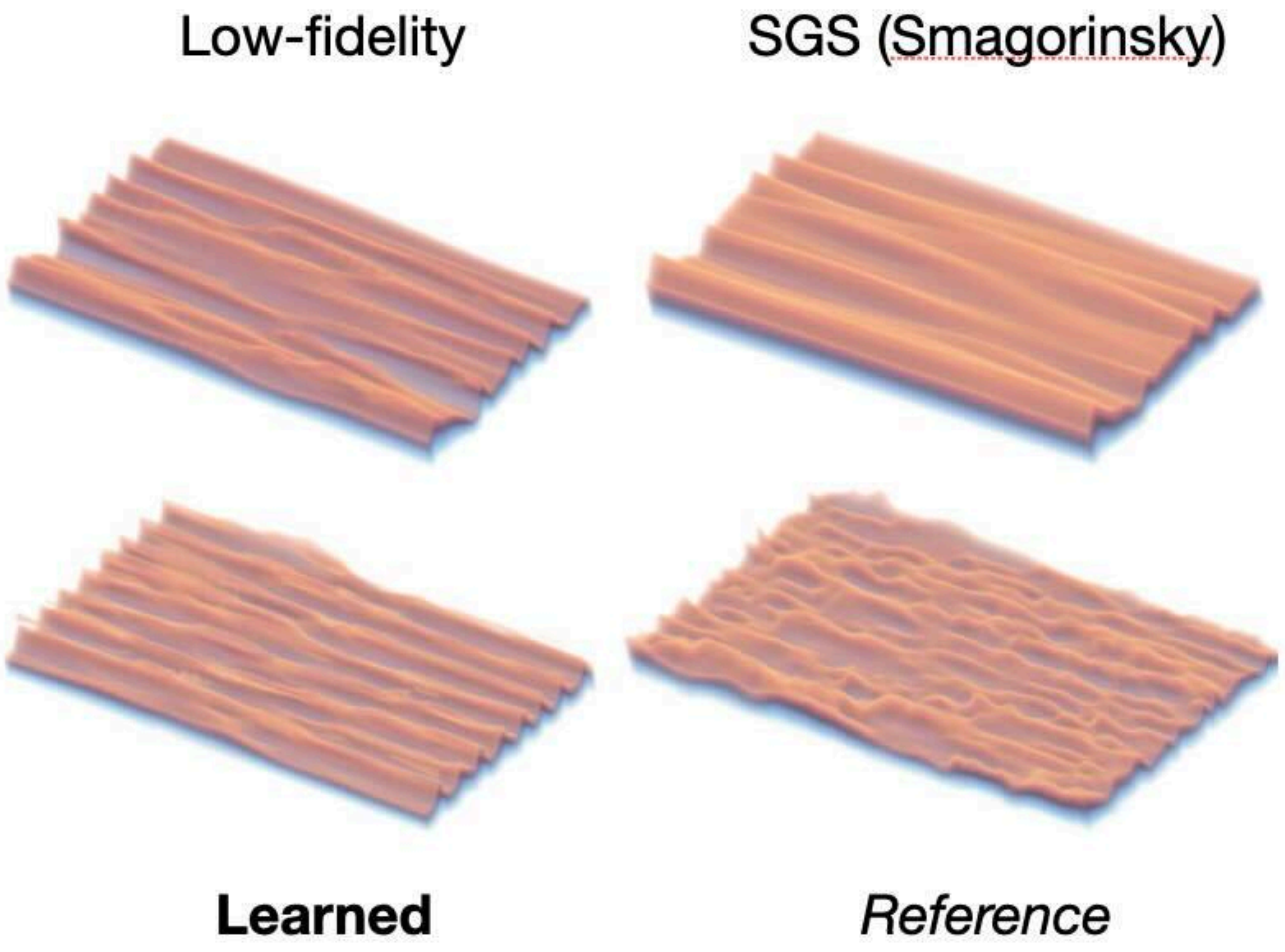
Reward is based on the similarity to the reference trajectory.

Physics Tasks in the RL Framework II

Learned Turbulence Models

Long term training signal via flow statistics

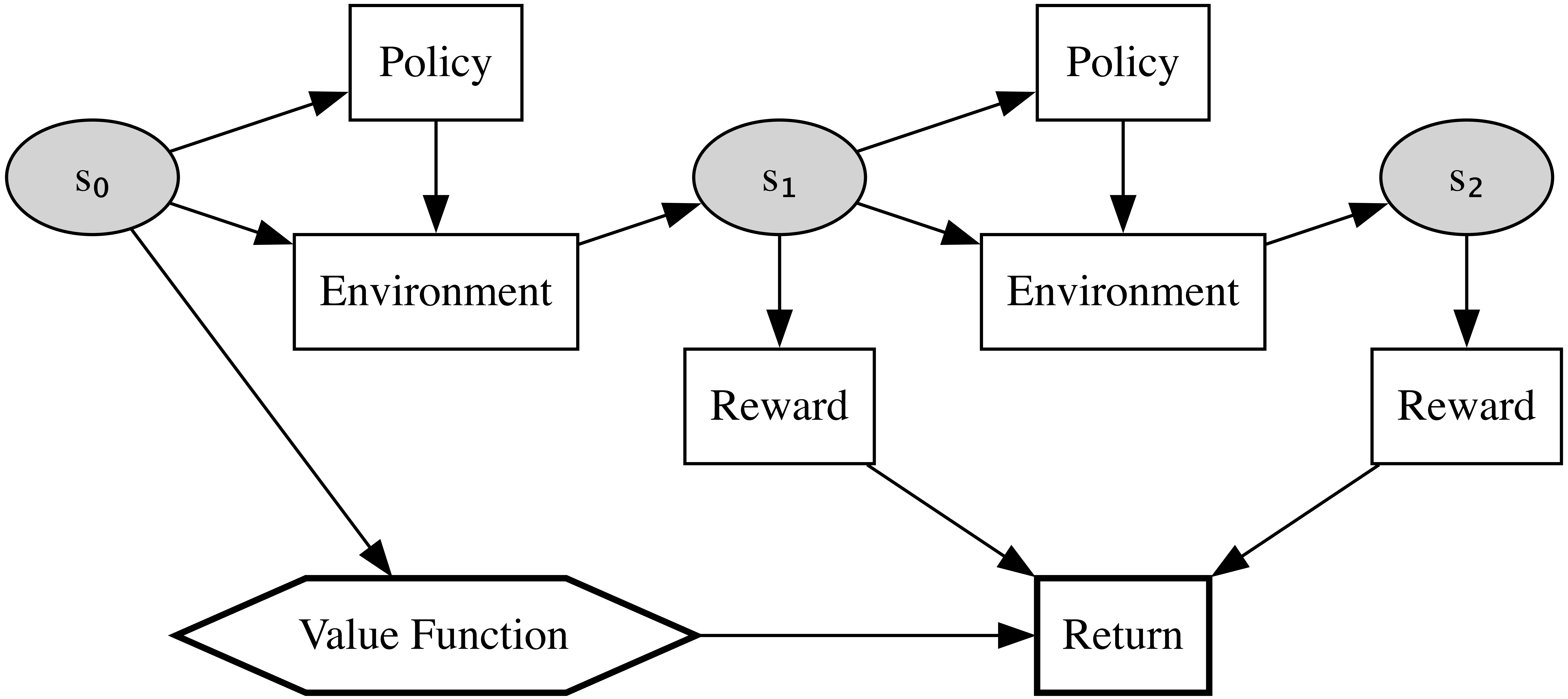
Fundamental topic: approximate influence of unresolved scales



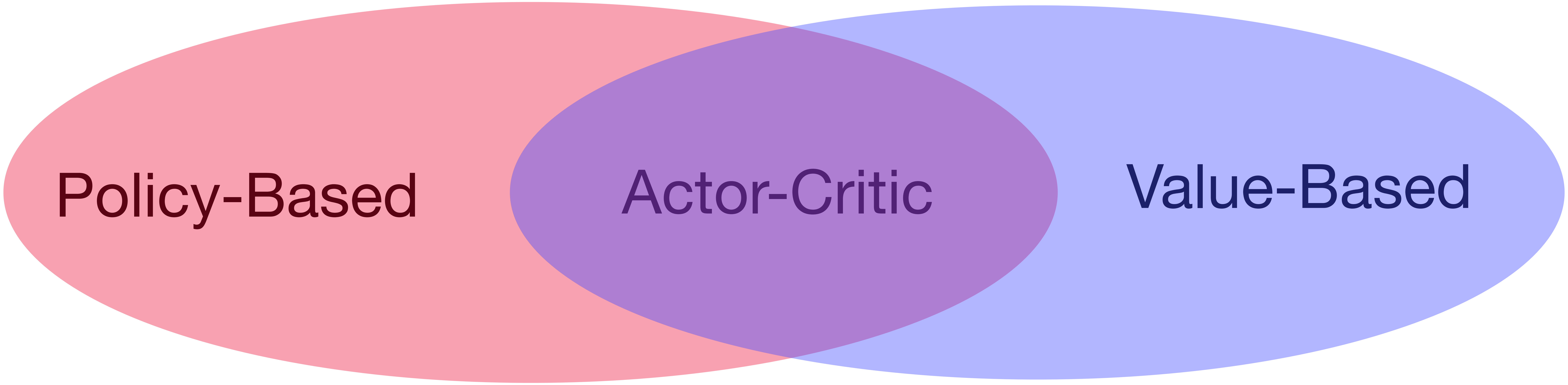
The Idea of a Critic

A central feature of RL:

Value functions instead of rollout-based estimates



Categories of RL methods



Policy-Based

- Policies are learned
- Similar to most physics setups.

Actor-Critic

- Values are learned
- Policies are learned.

Value-Based

- Values are learned
- Policies are derived implicitly

Value Functions

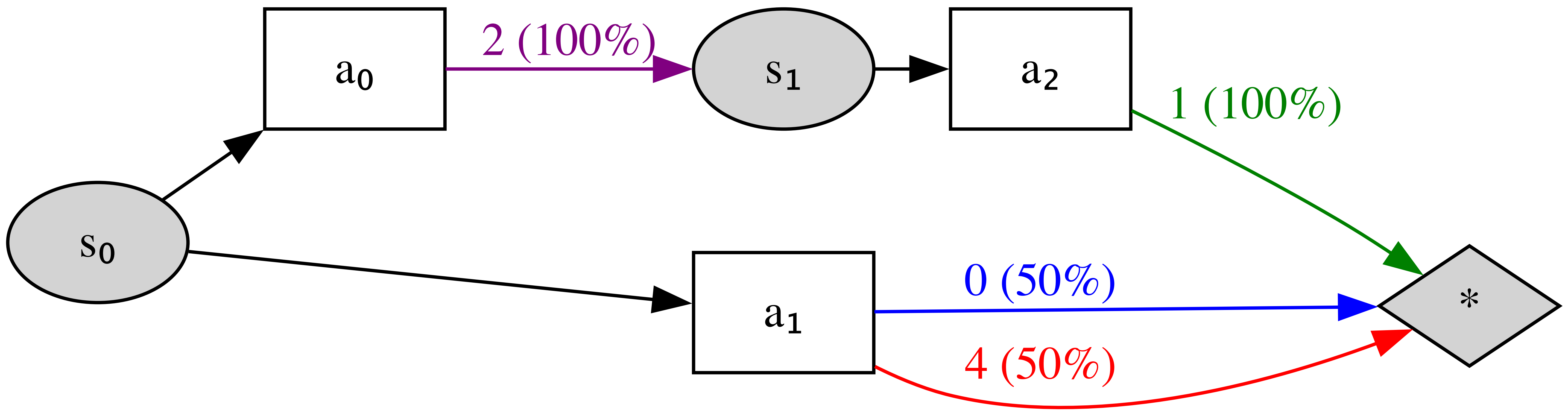
Markov Property

The future depends only on the present, not the past.

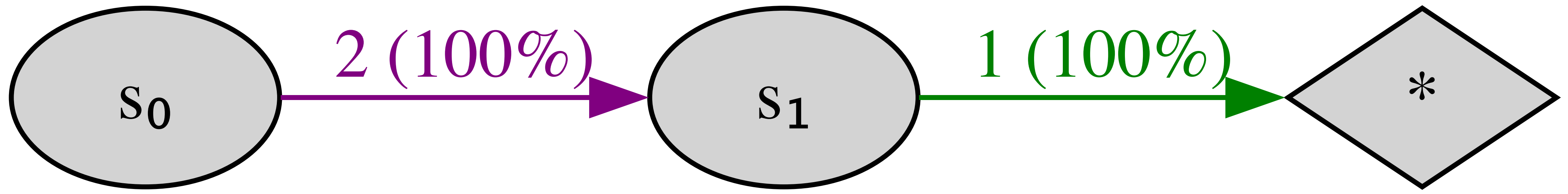
$$p(s_1, r_1 \mid s_0, a_0)$$

Visualization of Markov Processes

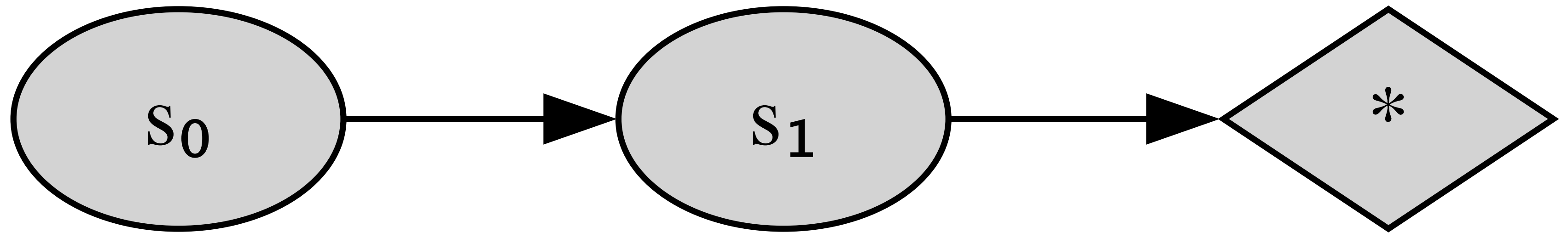
Markov Decision Process (MDP)



Markov Reward Process (MRP)



Markov Process (MP)



Value Functions

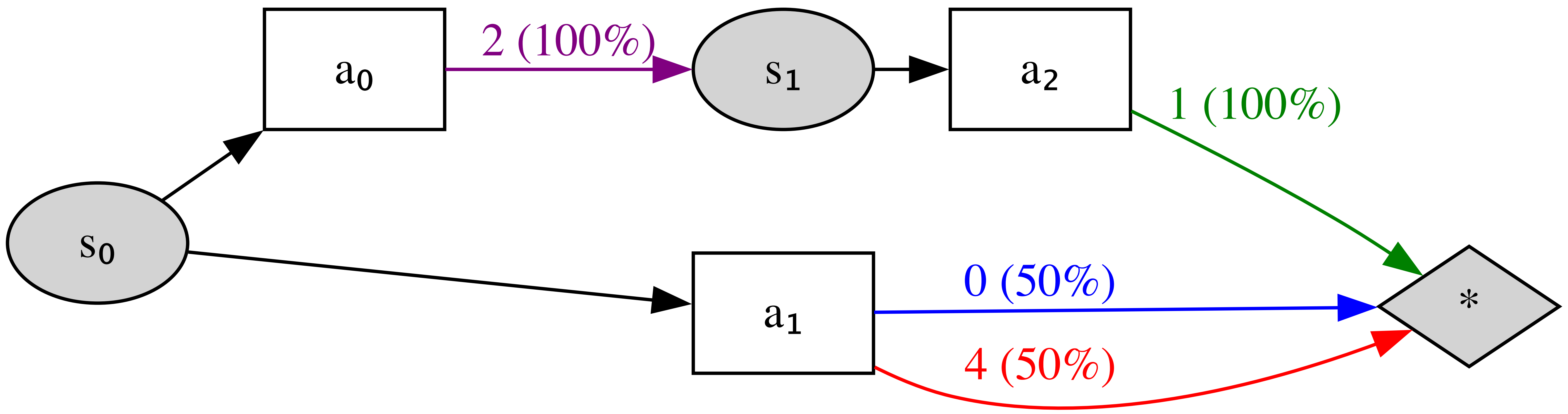
Value function: $v_\pi(s) = \mathbb{E}[\sum_{i>t} R_i \mid S_t = s]$

Bellman equation: consistency equation fulfilled by the true value function

(deterministic) $v_\pi(s_i) = v_\pi(s_{i+1}) + r$

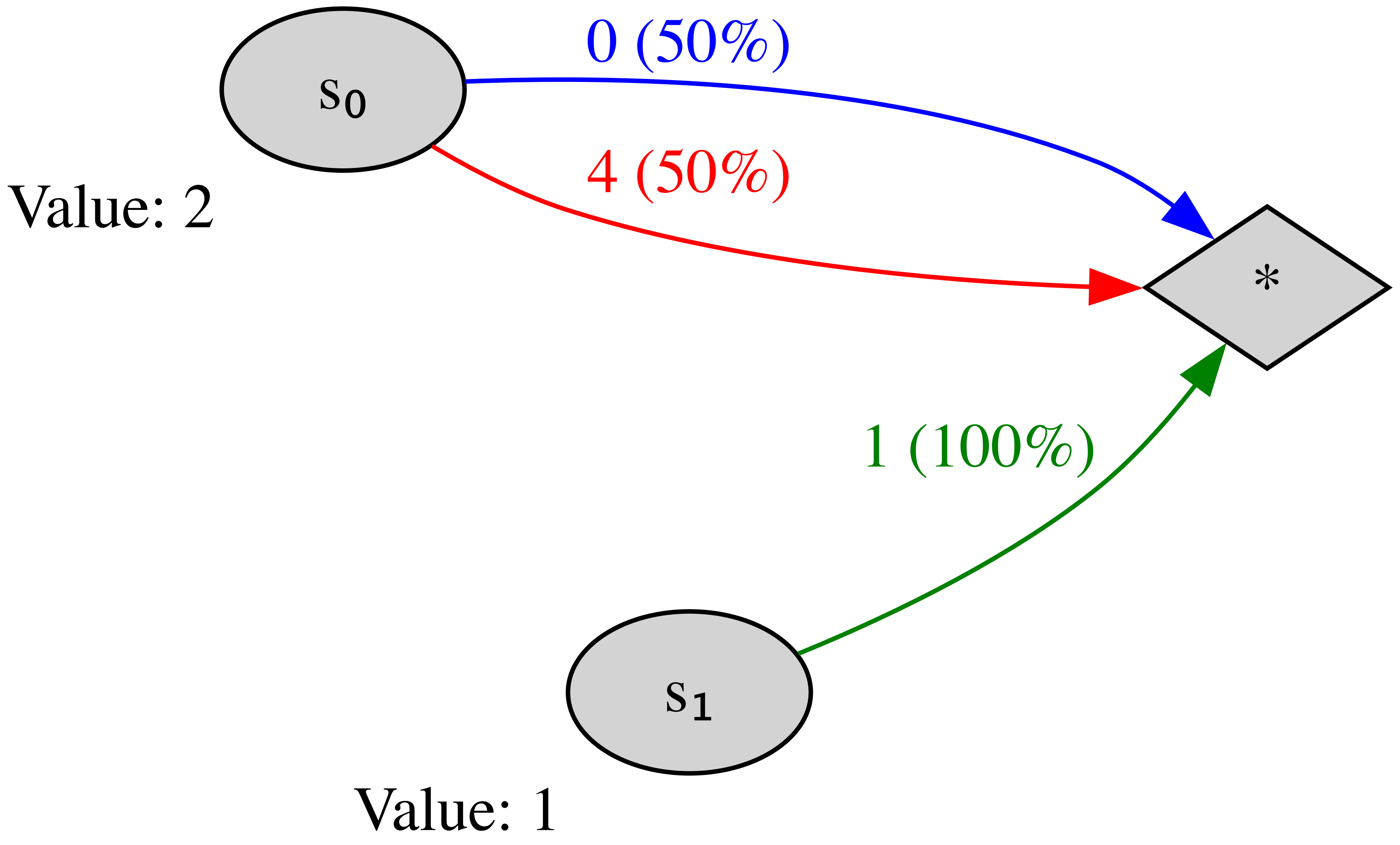
(stochastic) $v_\pi(s_i) = \sum_{s_{i+1},r,a_i} p(s_{i+1}, r \mid s_i, a_i) \cdot \pi(a_i \mid s_i) \cdot \left(v_\pi(s_{i+1}) + r\right)$

Example

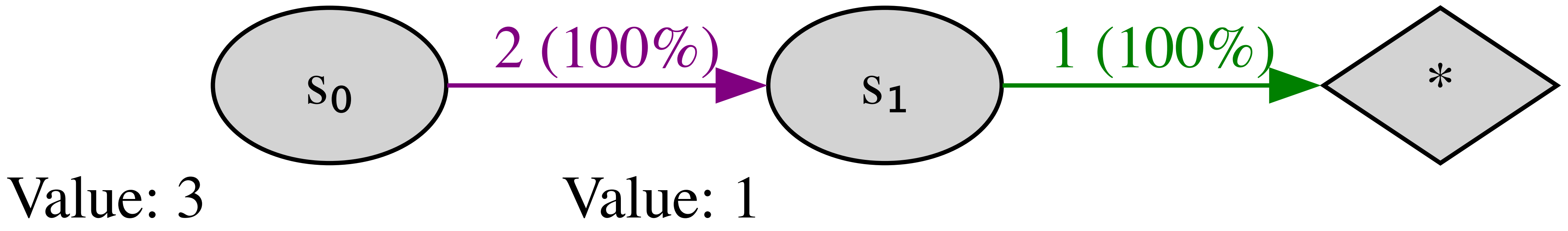


Policy 2:

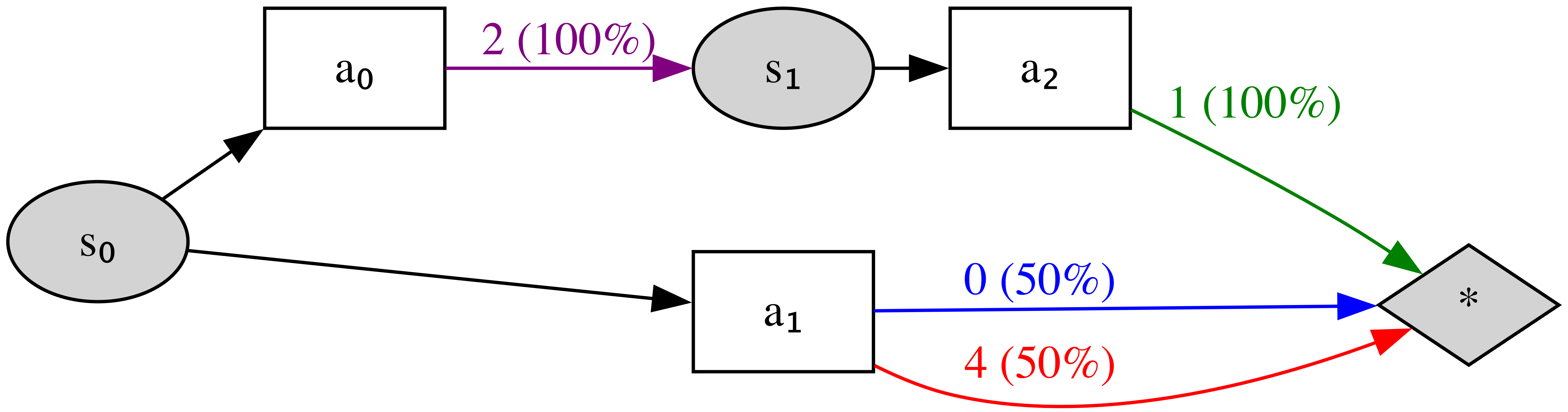
$$\pi(s_0) = a_1,\ \pi(s_1) = a_2$$



Policy 1: $\pi(s_0) = a_0,\ \pi(s_1) = a_2$



V- and Q-Values



V-Values (Policy 1): $v(s_0) = 3. \quad v(s_1) = 1$

Q-Values: $q_\pi(s, a) = \mathbb{E}[\sum_{i>t} R_i \,|\, S_t = s, A_t = a]$

$q(s_0, a_0) = 3 \qquad q(s_0, a_1) = 2 \qquad q(s_1, a_2) = 1$

RL Algorithms

Value Iteration

How to estimate value functions?

$$q(s, a) = \mathbb{E}[\sum_{i>t} R_i \,|\, S_t = s, A_t = a]$$

Monte-Carlo Loss

Generate a complete episode starting from state s_0 and action a_0 and collect all rewards r_i

$$L_{MC} = (q(s_0, a_0) - \sum_i r_i)^2$$

(Supervised approach)

Temporal Difference Loss

Generate a transition from state s_0 and action a_0 and collect reward r, next state s_1 and next action a_1

$$L_{TD} = (q(s_0, a_0) - q(s_1, a_1) - r)^2$$

(Bellmann-residual approach)

Policy Iteration

How to improve policies?

Given q-values, update policy by setting:

$$\pi(s) = \text{argmax}_a \, q(s, a)$$

Value-based RL algorithms iterate between value estimation and policy improvement

Monte Carlo Learning

Repeat:

Generate episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T$ *following policy* π

$g \leftarrow 0$

For $t = T - 1, \ldots, 0$ *repeat:*

$g \leftarrow g + r_{t+1}$

Append g *to* $\mathrm{Returns}(s_t, a_t)$

$Q(s_t, a_t) \leftarrow \mathrm{average}(\mathrm{Returns}(s_t, a_t))$

$\pi(s_t) \leftarrow \mathrm{argmax}_a\, Q(s_t, a)$

Temporal Difference Learning

Q-Learning

Repeat:

$t \leftarrow 0$

Initialize s_0

Repeat until episode terminates:

Generate next step a_t, r_{t+1}, s_{t+1} *by following policy* π

$\Delta \leftarrow r_{t+1} + \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$

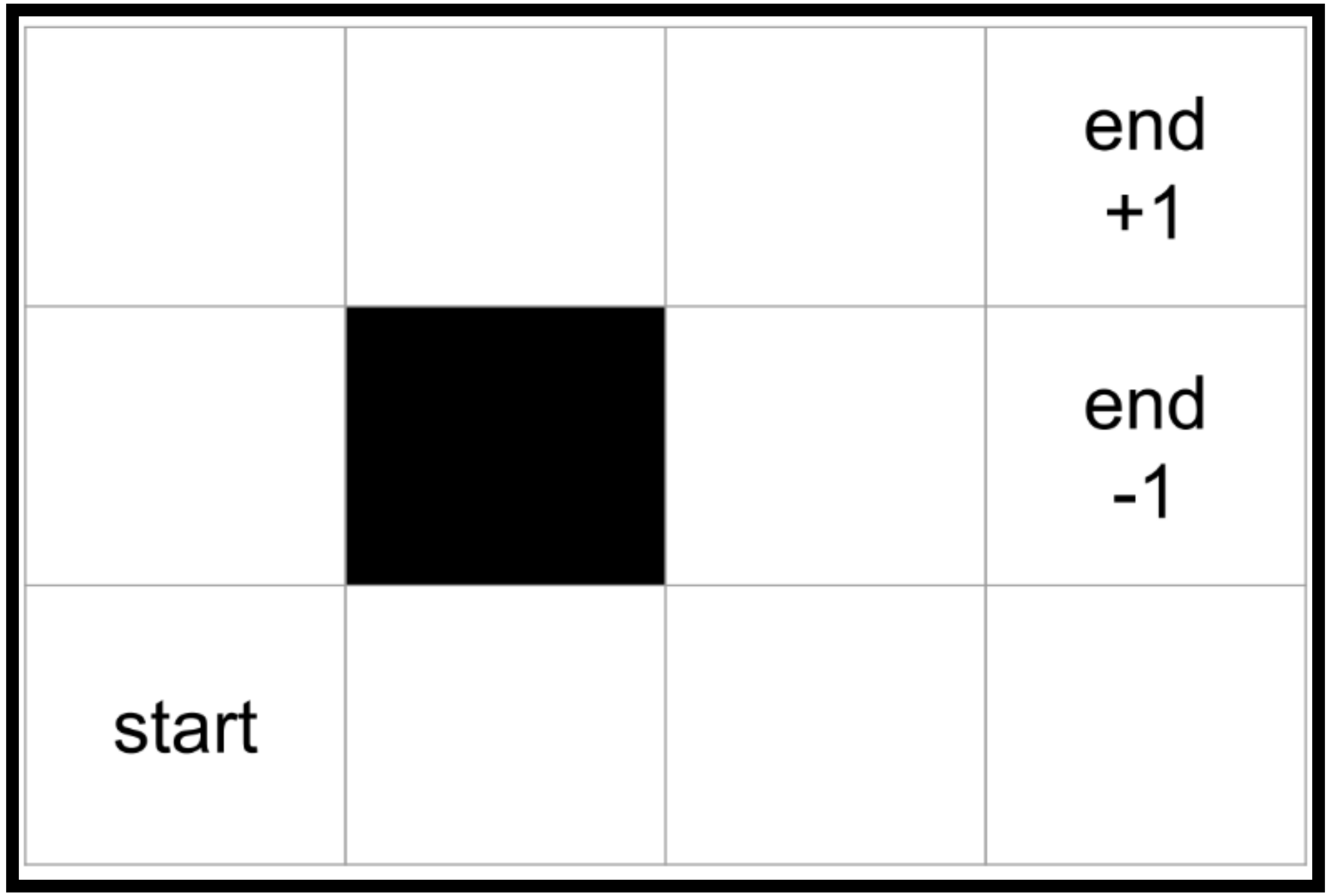
$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Delta$

$\pi(s_t) \leftarrow \operatorname{argmax}_a Q(s_t, a)$

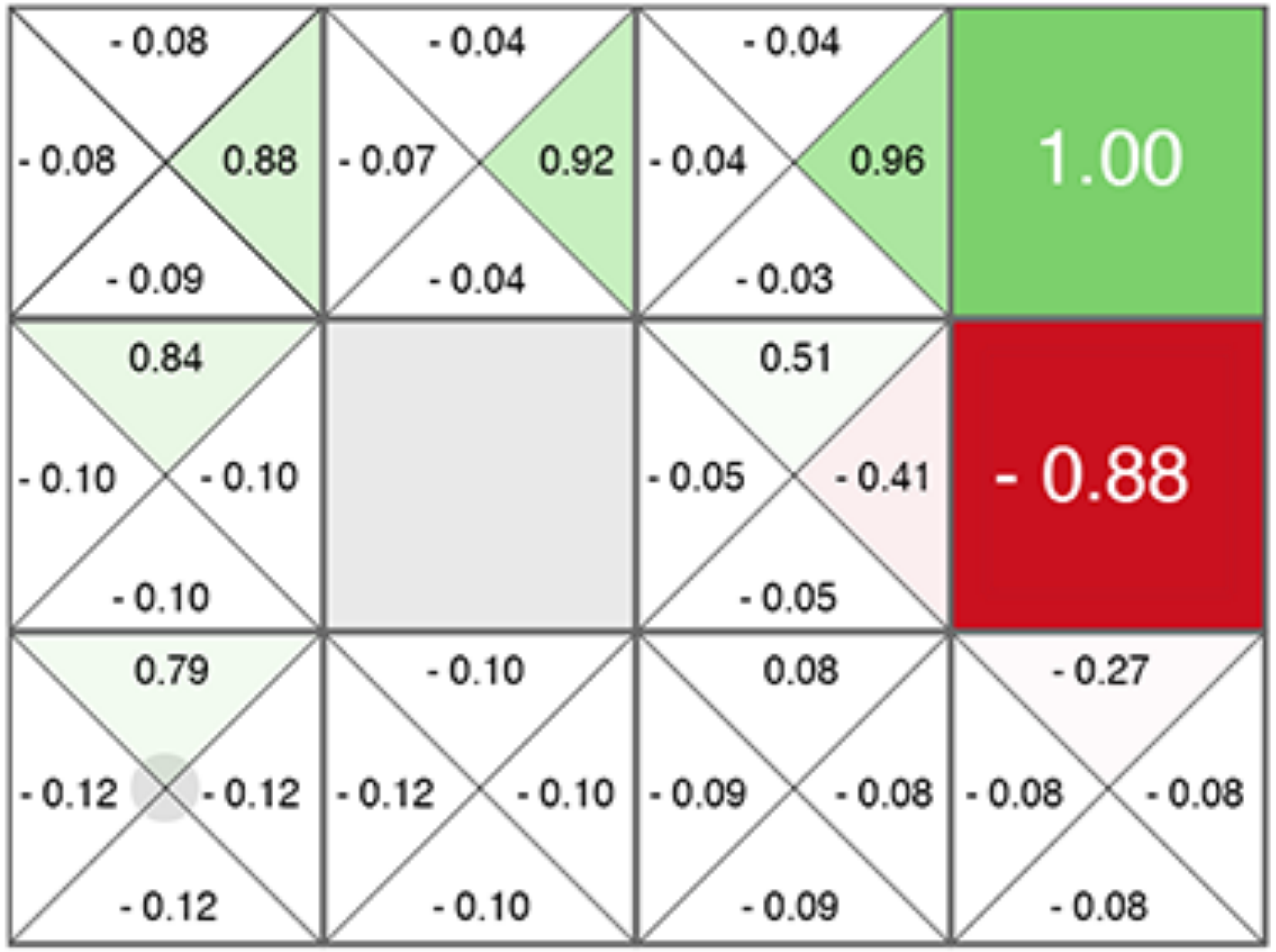
$t \leftarrow t + 1$

Grid World

Environment



Q-Values



Function Approximation in RL

Function approximation is needed if the number of states or actions is too large or even continuous.

Approximating values with neural networks parametrized by θ:

$$v(s) \rightarrow v_\theta(s)$$

Training through backpropagation of value errors:

$$\Delta\theta = \frac{\partial v}{\partial \theta} \Delta v$$

DDPG: An actor-critic method

Deep deterministic policy gradient

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**

Data generation

   Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
   Execute action a_t and observe reward r_t and observe new state s_{t+1}

Replay buffer

   Store transition (s_t, a_t, r_t, s_{t+1}) in R
   Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Value iteration

   Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
   Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Policy iteration

   Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Target networks

   Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

Part 2: Why RL Needs Specialized Techniques

Components of a learning system: one-step -> multi-step methods

- Model: autoregressive models -> value functions
- Loss: supervised -> residual-based
- Optimization: gradient methods-> non-gradient methods

Stochasticity is a key property to motivate these changes.

Autoregressive Models vs Value Functions

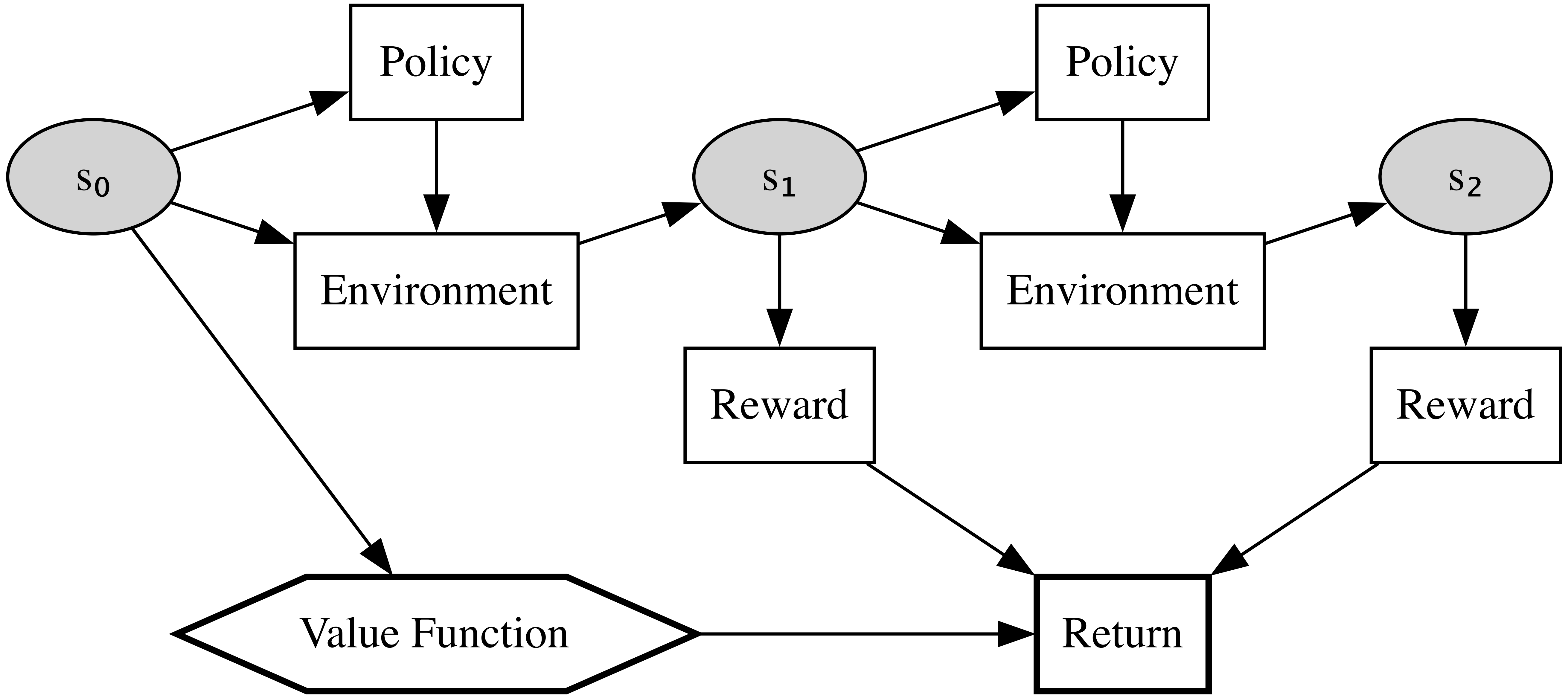
Prediction: One-Step and Multi-Step Methods

Autoregressive model $s_t = F(s_{t-1})$

- Perfect a model on one-step predictions
- Apply it recursively to receive a multi-step prediction

Value function $\sum_{t>t} r_i = F(s_t)$

- A multi-step method



Multi-Step Predictors on Differential Equations

Differential equation and initial condition:

$$\partial_t u(t) = P(t, u, \ldots) \quad \text{with} \quad u(0) = u_0$$

How to learn the solution of this differential equation?

Discretize on a grid $u_i = u(t_i)$

- (One-step approach) Autoregressive model: $u_{i+1} = F(u_i)$
- (Multi-step approach) Directly parametrize the solution $u_i = F(t_i)$.

Disadvantages of Autoregressive Models

Autoregression

- Requires N model calls for an N-step prediction
- Learns the entire state trajectory
  (in stochastic settings, all occurring state sequences)
- Exponential error growth over time

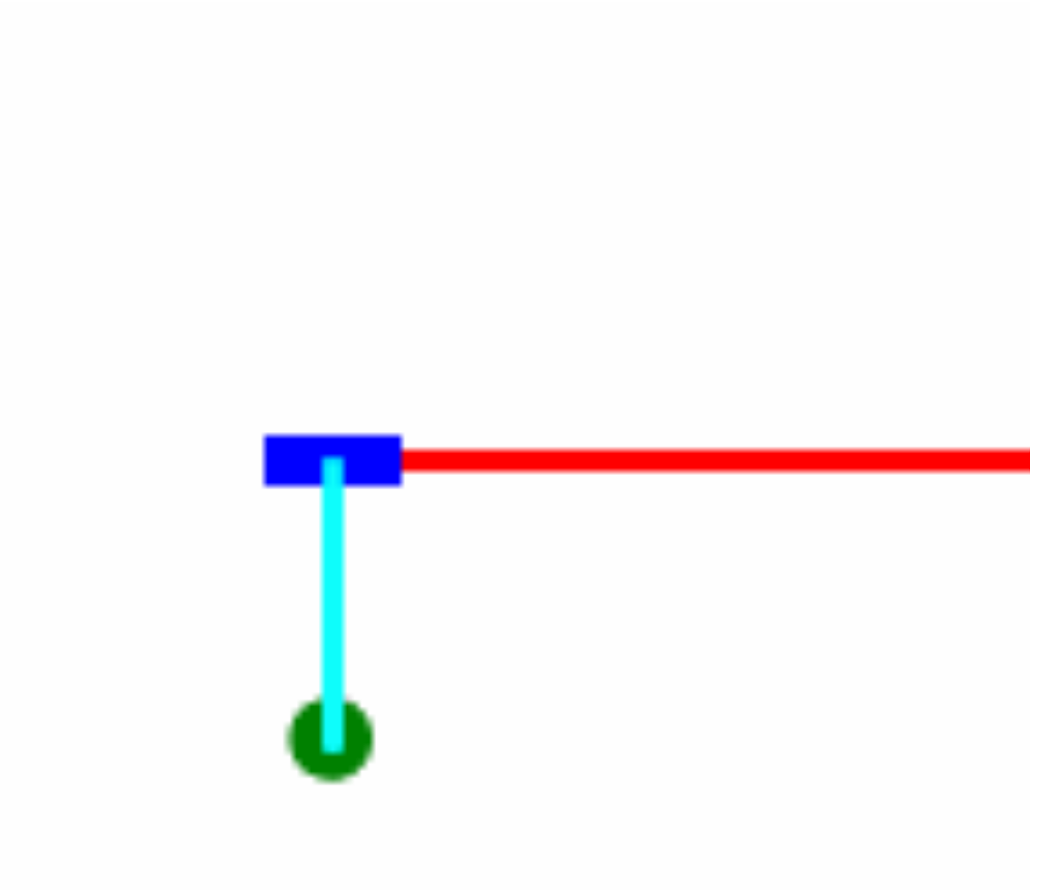
Example: Ising Model

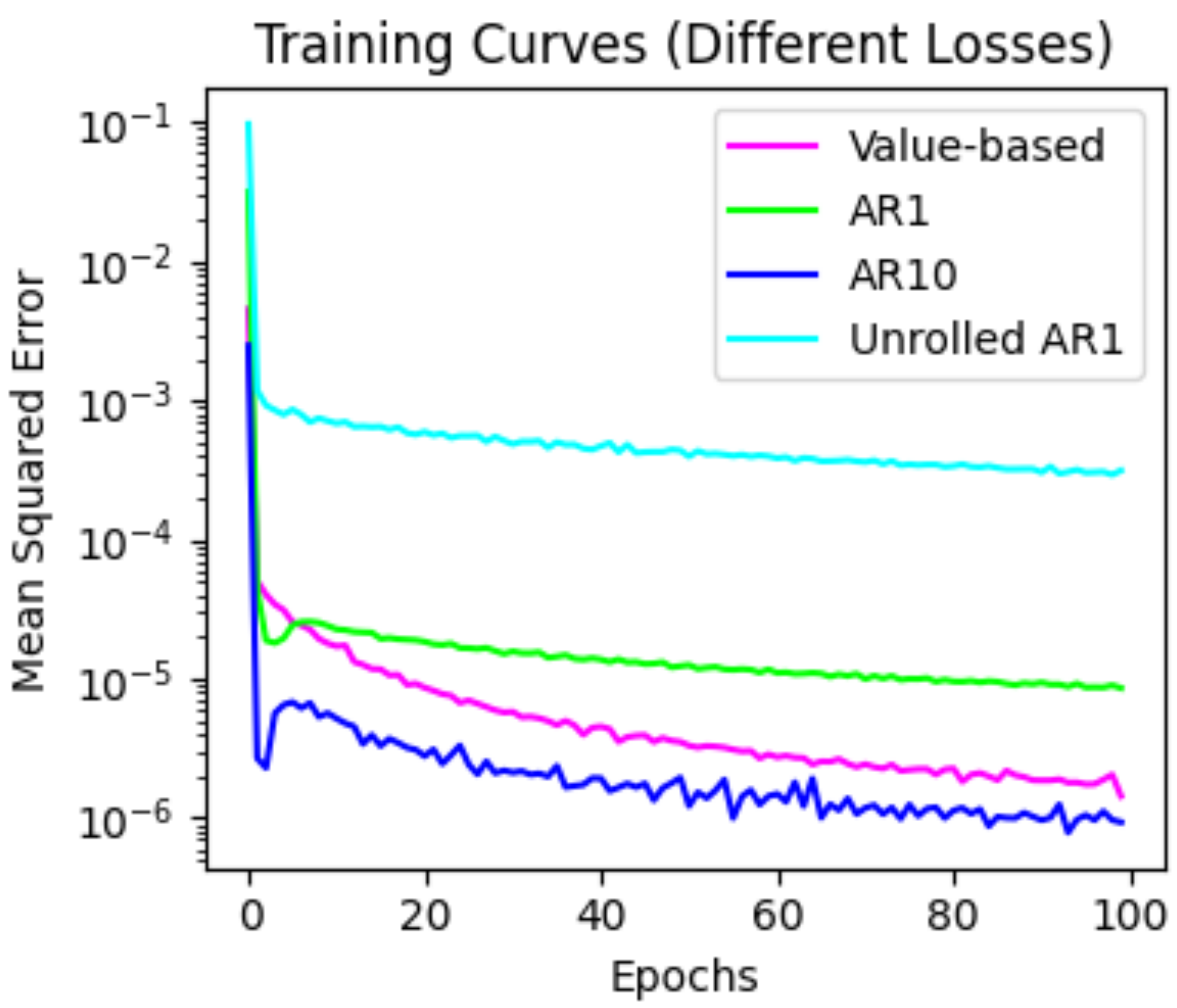
Goal: learn to predict who wins

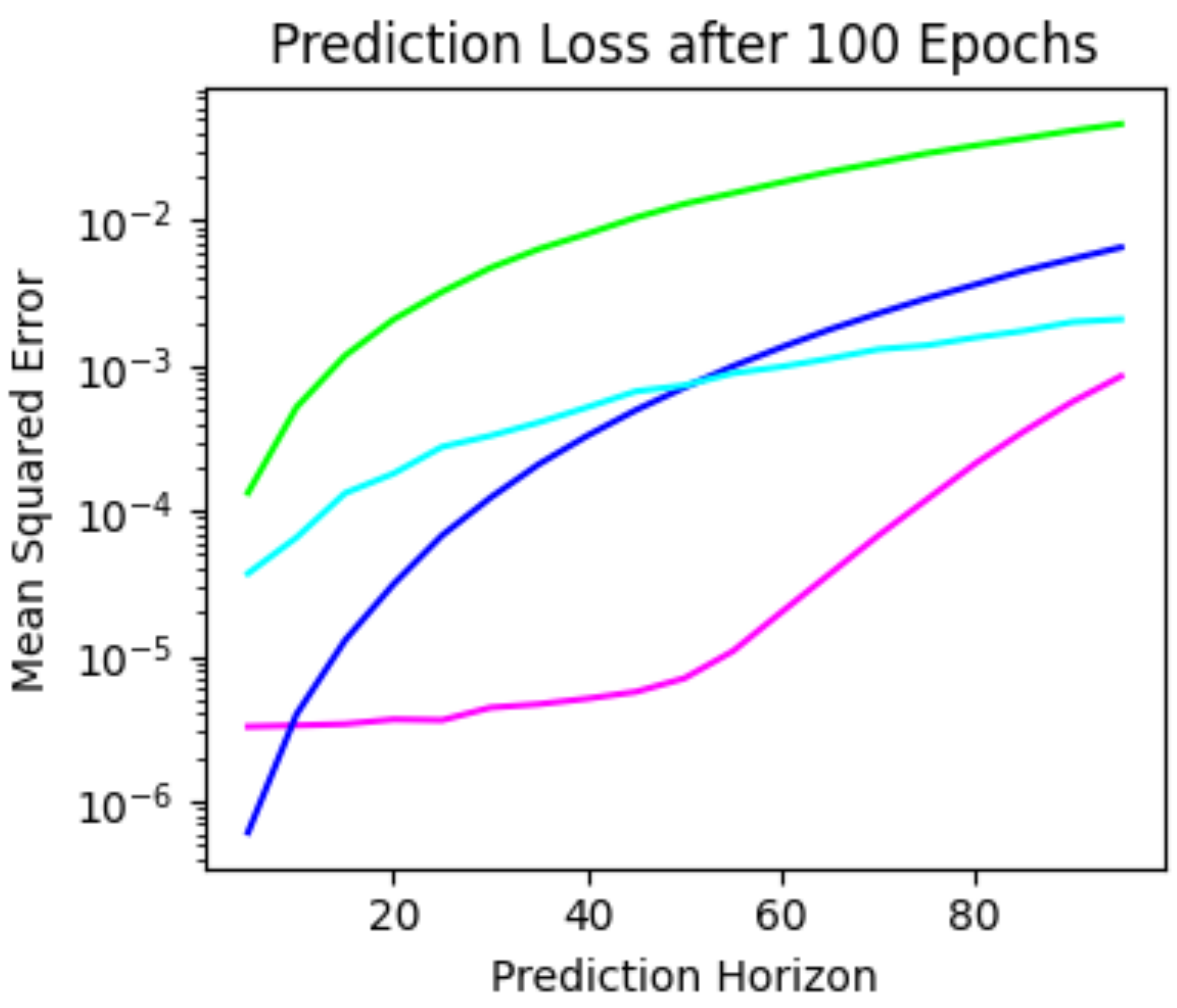
Autoregressive models learn the complete environment dynamics.

Value functions learn possible shortcuts (e.g. number of white cells minus number of black cells)

Example: Cart Pole Prediction







Method	AR1	Unrolled AR1	AR10	Value-based
Parameters	9000	9000	9000	9000
Training Time [sec]	300	800	300	300
Prediction Time [sec]	270	270	295	6

Supervised vs Residual-Based Losses

Update Equations

Supervised Learning

Monte Carlo (MC):

$$\Delta v(s_t) = \alpha \cdot \left(\sum_{i>t} r_i - v(s_t) \right)$$

Trajectory Learning

$$\Delta s_t = \alpha \cdot \left(P^t(s_0) - s_t \right)$$

Residual-Based Learning

Temporal Difference (TD):

$$\Delta v(s_t) = \alpha \cdot \left(r_t + v(s_{t+1}) - v(s_t) \right)$$

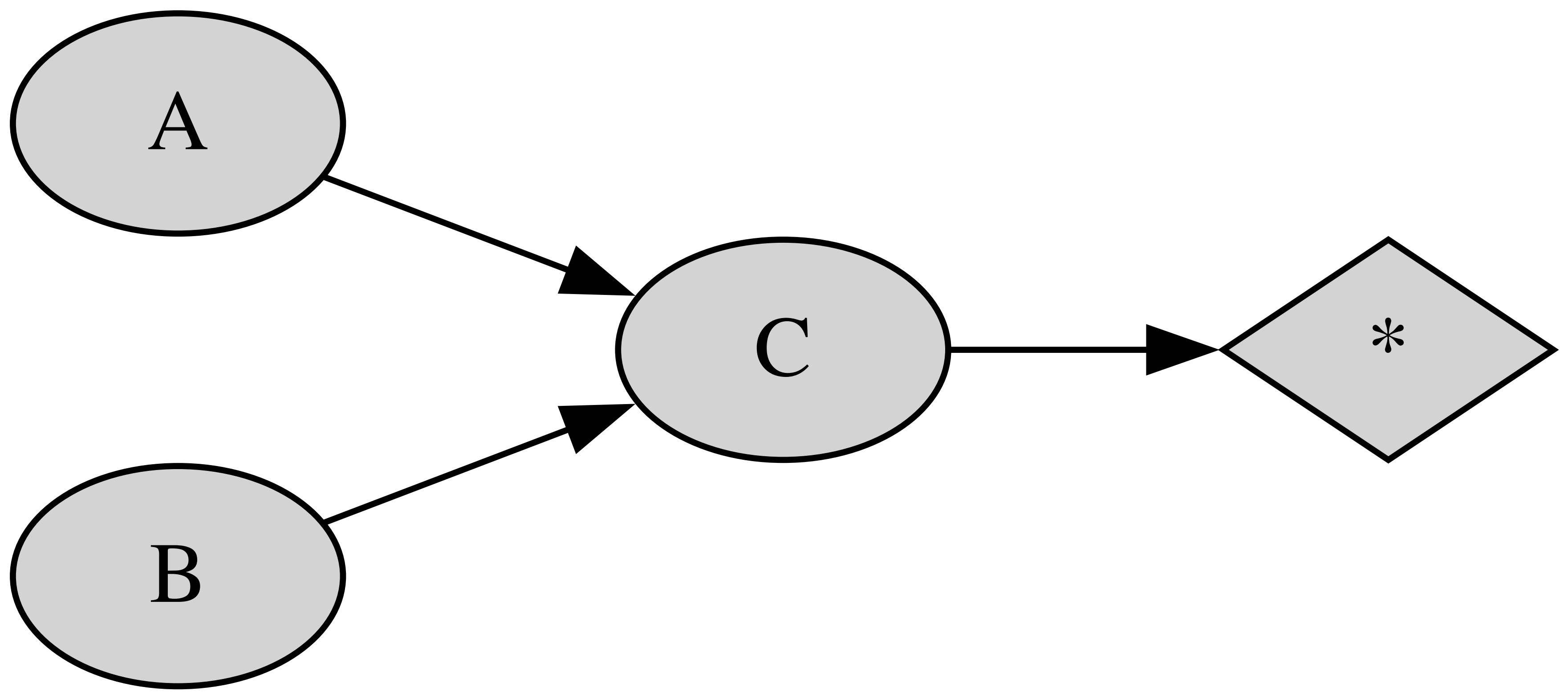
Physics-Informed Training

$$\Delta s_{t+1} = \alpha \cdot \left(P(s_t) - s_{t+1} \right)$$

v value s_t state at time t r_t reward at time t α learning rate P physics operator

Example: 3-State Value Estimation

How would you estimate the values in this Markov reward process based on these observed episodes?



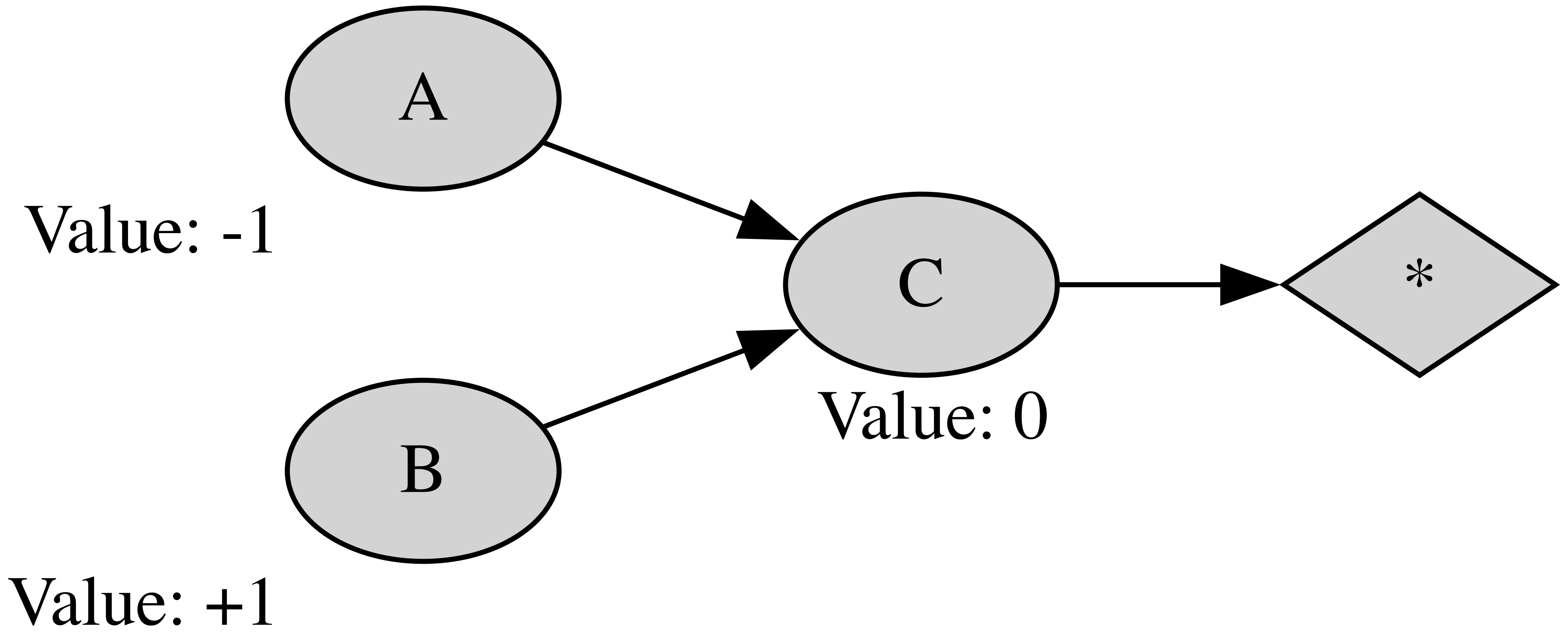
Episode 1: state A, reward 0, state C, reward -1

Episode 2: state B, reward 0, state C, reward +1

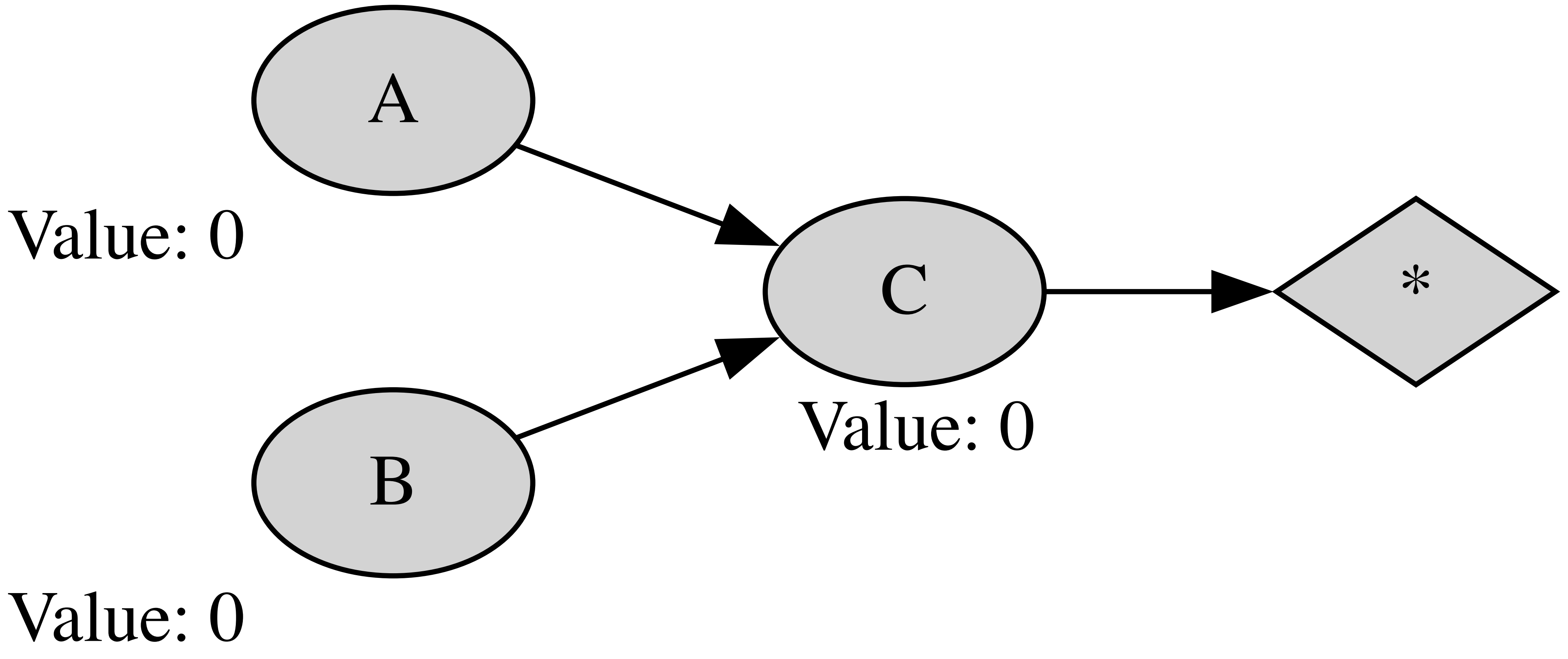
Example: 3-State Value Estimation

Episode 1: state A, reward 0, state C, reward -1

Episode 2: state B, reward 0, state C, reward +1



Monte Carlo



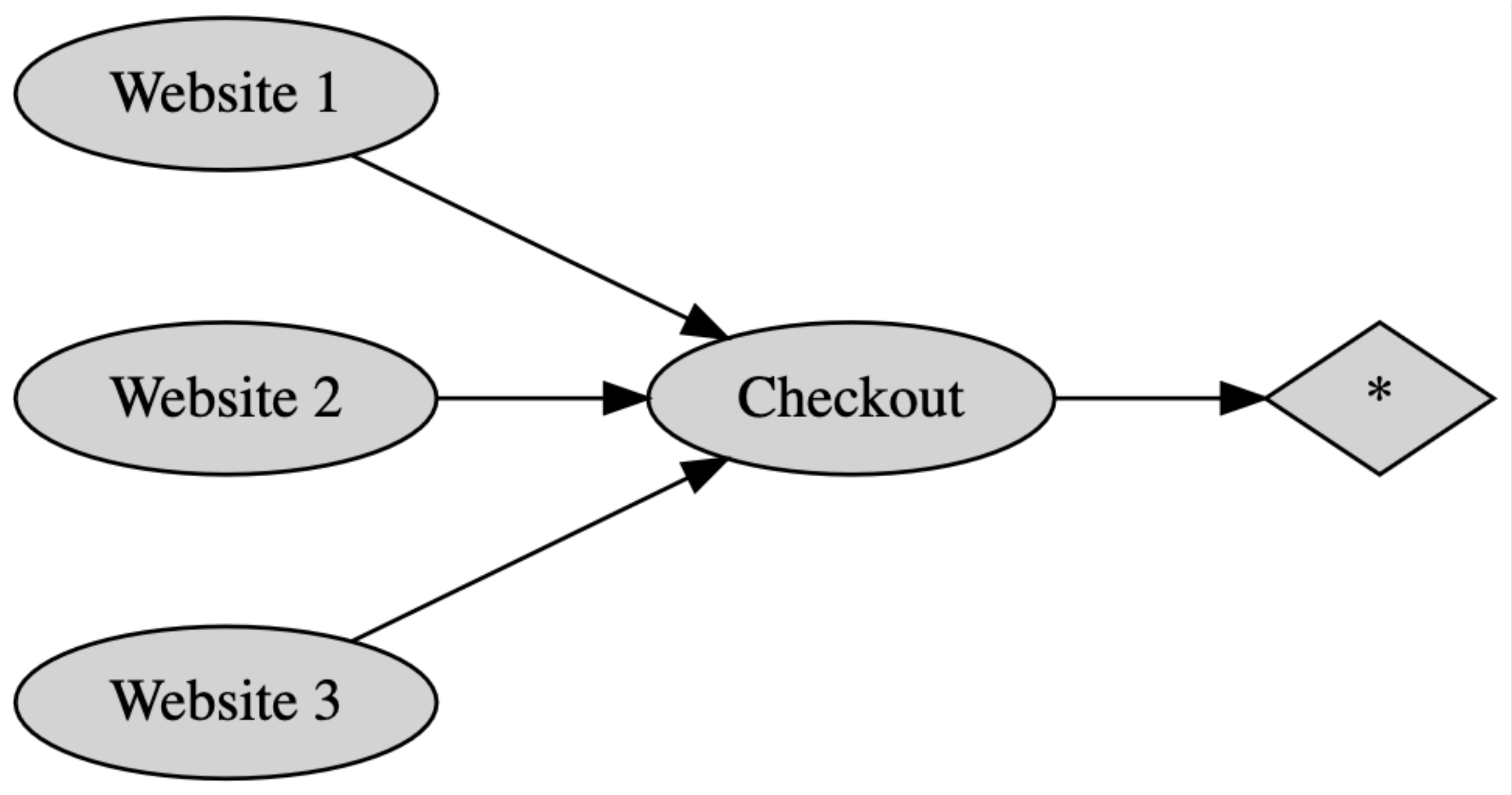
Temporal Difference

MC minimizes the error on past data.

TD minimizes the error on future data (through the Markov property).

Example: Website Design

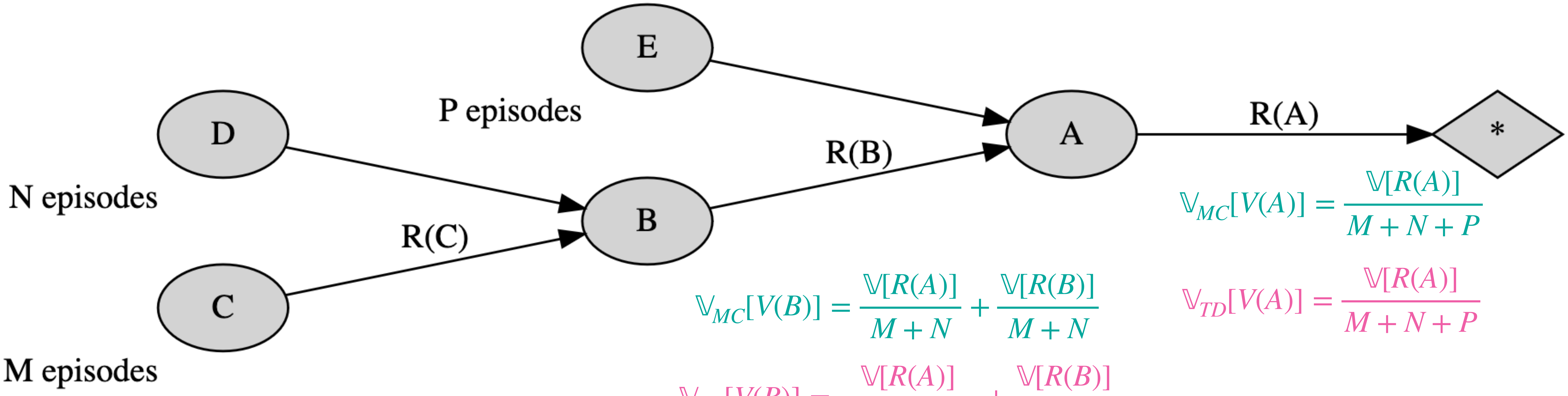
Task: Select website design that leads to the largest sale rate.



Which one would you choose based on this data?

	Website 1	Website 2	Website 3
Visitors	10	8	7
Proceeded to Checkout	6	5	4
Purchase	4	2	2

Variance of MC and TD Estimators



$$\mathbb{V}_{MC}[V(A)] = \frac{\mathbb{V}[R(A)]}{M+N+P}$$

$$\mathbb{V}_{TD}[V(A)] = \frac{\mathbb{V}[R(A)]}{M+N+P}$$

$$\mathbb{V}_{MC}[V(B)] = \frac{\mathbb{V}[R(A)]}{M+N} + \frac{\mathbb{V}[R(B)]}{M+N}$$

$$\mathbb{V}_{TD}[V(B)] = \frac{\mathbb{V}[R(A)]}{M+N+P} + \frac{\mathbb{V}[R(B)]}{M+N}$$

$$\mathbb{V}_{MC}[V(C)] = \frac{\mathbb{V}[R(A)]}{M} + \frac{\mathbb{V}[R(B)]}{M} + \frac{\mathbb{V}[R(C)]}{M}$$

$$\mathbb{V}_{TD}[V(C)] = \frac{\mathbb{V}[R(A)]}{M+N+P} + \frac{\mathbb{V}[R(B)]}{M+N} + \frac{\mathbb{V}[R(C)]}{M}$$

TD estimators have lower variance.

Example: Going Home

	Current prediction	New observation	MC	TD
University	40 min left		+	0
		took 5 min		
University subway station	35 min left		+	-
		took 25 min		
Home subway station	5 min left		+	+
		took 15 min		
Home	0 min left			

Gradient vs Non-Gradient Methods

Optimization Methods

Classical Methods

- Newton's Method
- Gradient Descent
- Momentum

Our Interest

- First-order
- Not Gradient Descent
- Not gradient-based

TD Updates are Non-Gradient Updates

TD loss for step t

$$L_t = \left(r_t + v(s_{t+1}) - v(s_t)\right)^2$$

Non-gradient update

$$\Delta v(s_t) = \alpha \cdot \left(r_t + v(s_{t+1}) - v(s_t)\right)$$

Gradient update

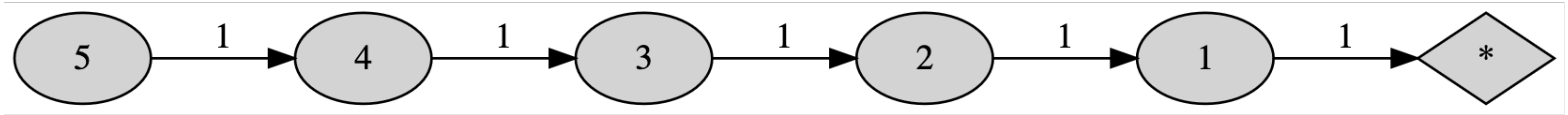
$$\Delta v(s_t) = \alpha \cdot \left(r_t + v(s_{t+1}) - v(s_t)\right)$$

$$\Delta v(s_{t+1}) = -\alpha \cdot \left(r_t + v(s_{t+1}) - v(s_t)\right)$$

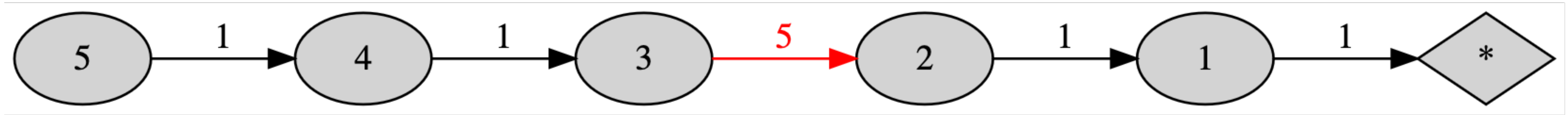
v value s_t state at time t r_t reward at time t α learning rate

Information Flow

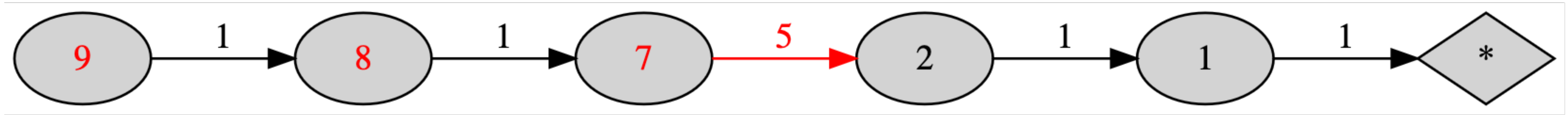
Rewards (edges) and corresponding values (nodes)



New observation



Earlier values have to be changed, not later values.

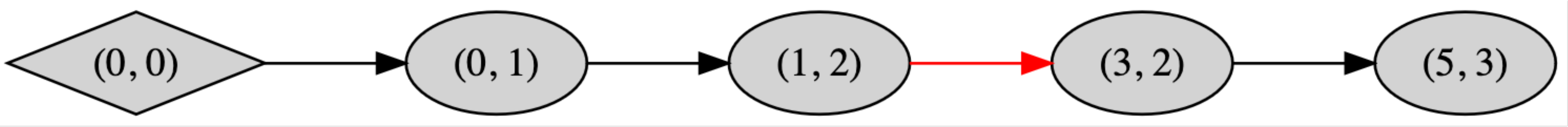


Information Flow II

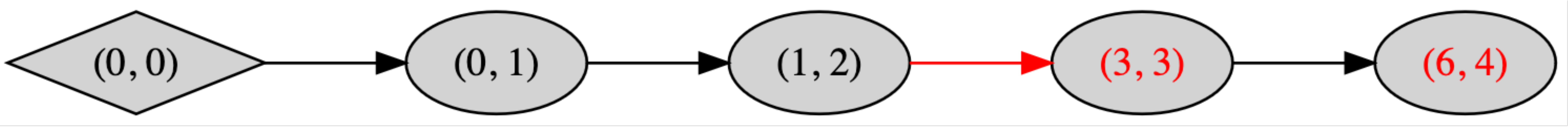
One particle in a uniform force field: $\ddot{x} = F$ with $x(0) = \dot{x}(0) = 0$

$$(x_{i+1}, p_{i+1}) = (x_i + p_i \cdot dt, p_i + F \cdot dt) \approx (x_i + p_i, p_i + 1)$$

Current Approximation

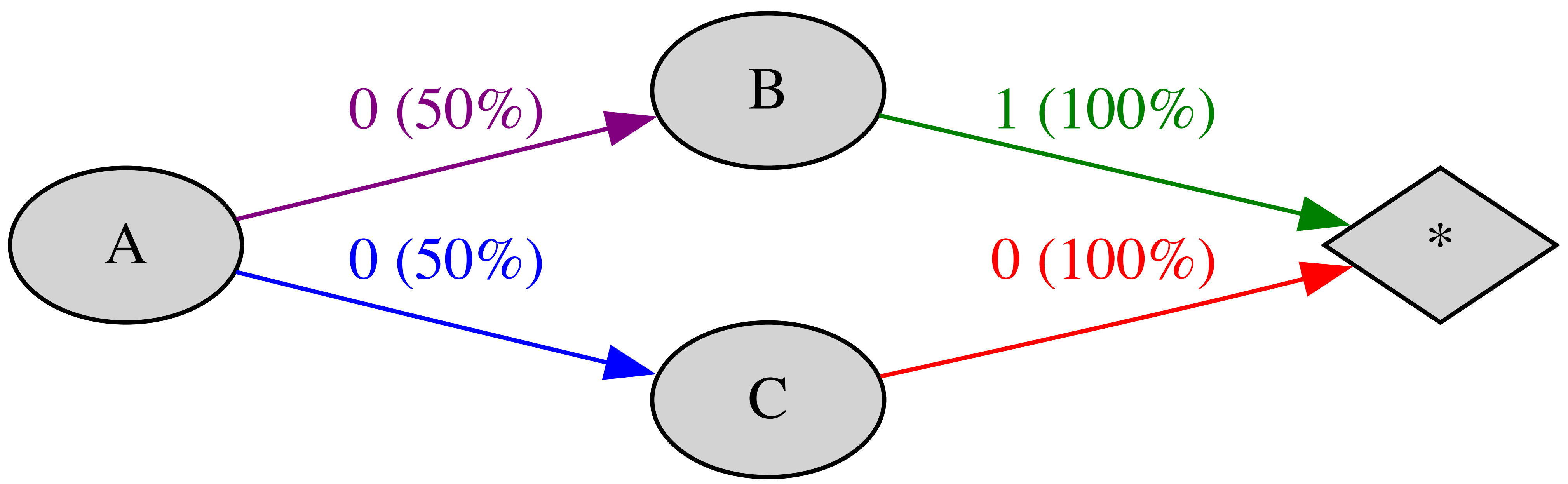


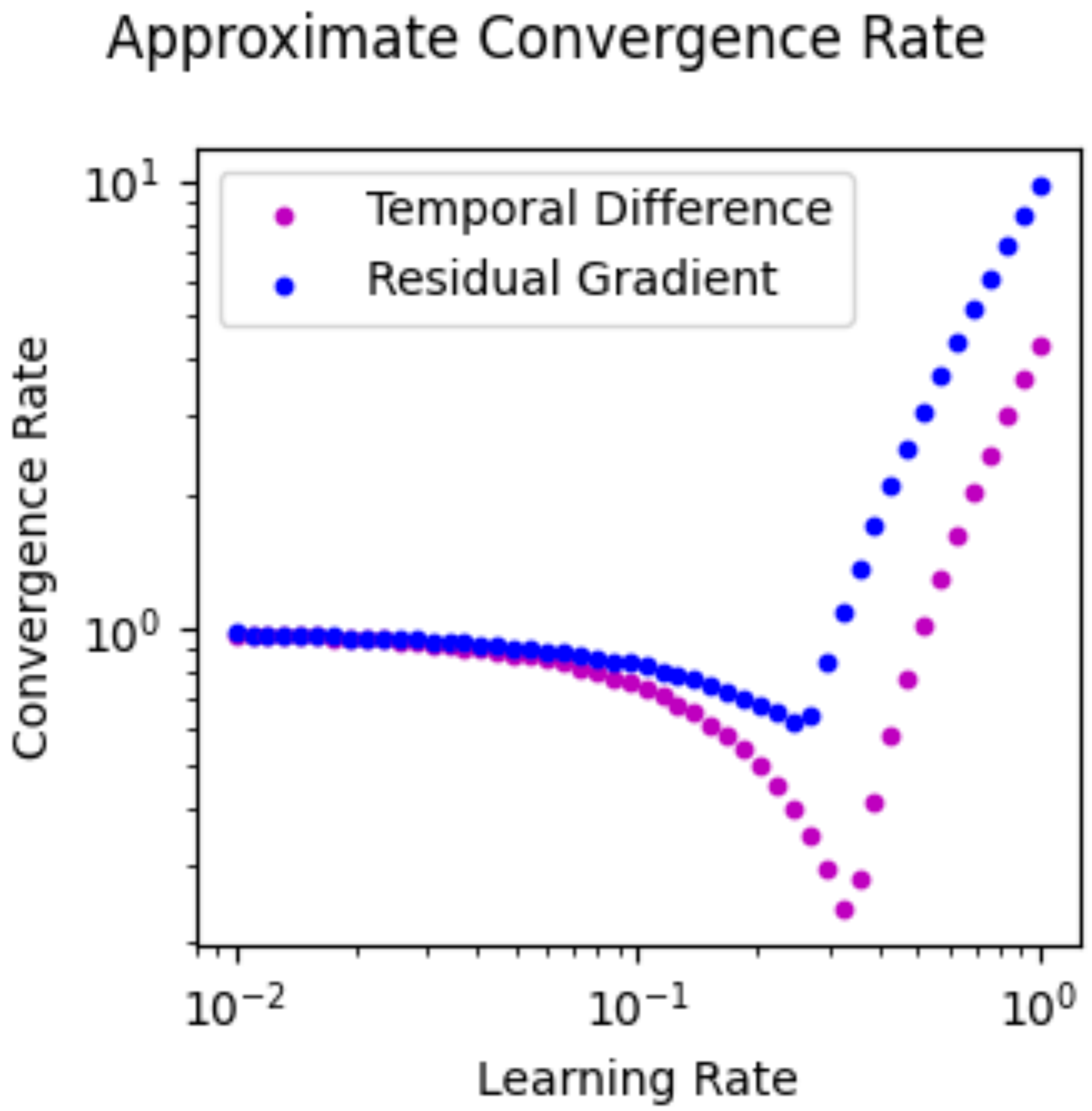
Later predictions have to be changed, not earlier predictions.



Convergence Rate

Value prediction on this
Markov reward process





Function Approximation

TD loss for step t

$$L_t = \left(r_t + v_\theta(s_{t+1}) - v_\theta(s_t)\right)^2$$

Gradient update

$$\Delta\theta = \alpha \cdot \left(r_t + v_\theta(s_{t+1}) - v_\theta(s_t)\right) \cdot \left(\partial_\theta v_\theta(s_t) - \partial_\theta v_\theta(s_{t+1})\right)$$

Non-gradient update

$$\Delta\theta = \alpha \cdot \left(r_t + v_\theta(s_{t+1}) - v_\theta(s_t)\right) \cdot \partial_\theta v_\theta(s_t)$$

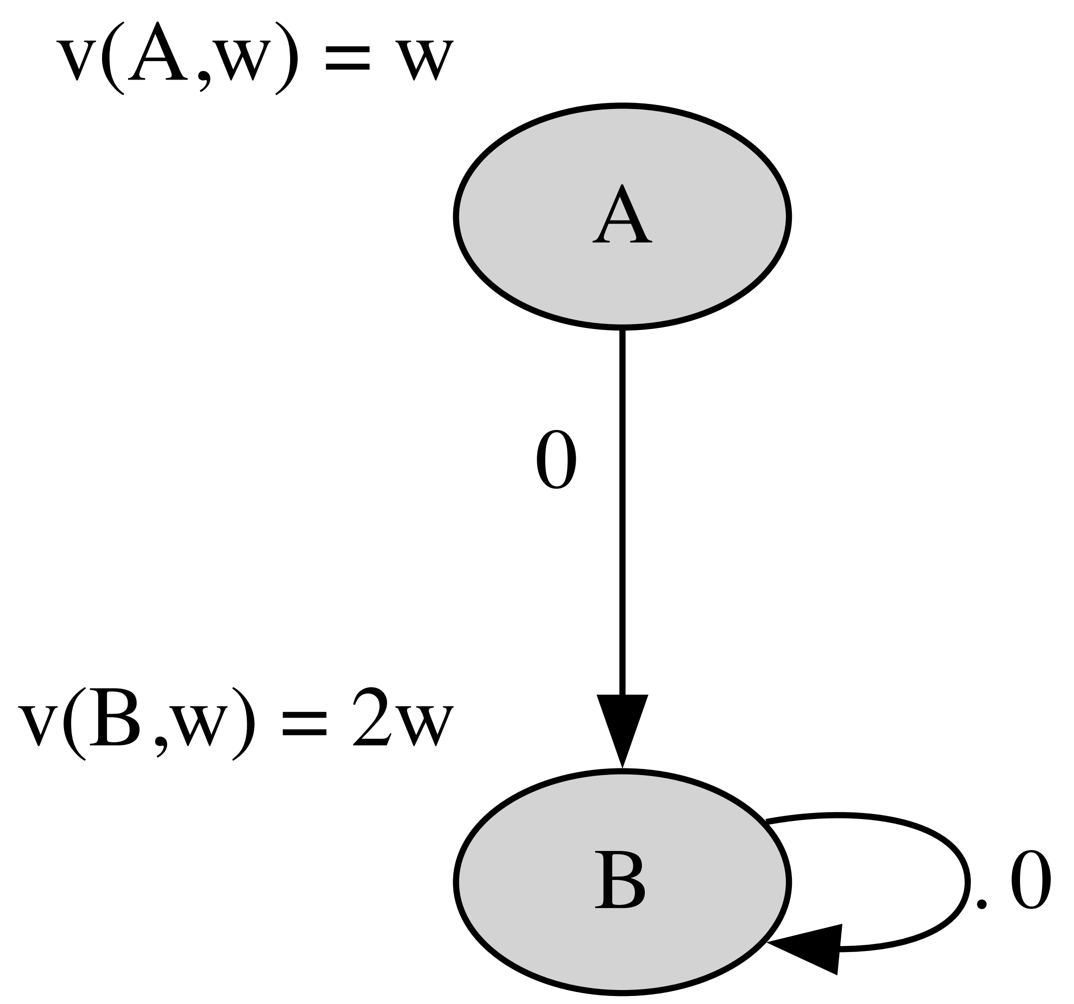
v value $\quad$ s_t state at time t $\quad$ r_t reward at time t $\quad$ α learning rate $\quad$ θ parameters

Convergence Properties

Divergence of non-gradient TD

Solution: $w = 0$

Non-gradient update step: $\Delta w = \alpha w$



Convergence theorem for non-gradient TD

Linear function approximation

On-Policy Sampling

Fixed Points

Linear function approximation with quadratic loss

$$L(\theta) = (F\theta - c)^2$$

Gradient update

$$\partial_\theta L = F^\dagger (F\theta - c)$$

Gradient fixed-point

$$\theta_{Gradient} = (F^\dagger F)^{-1} F^\dagger c$$

Non-gradient update

$$B(F\theta - c)$$

Non-gradient fixed-point

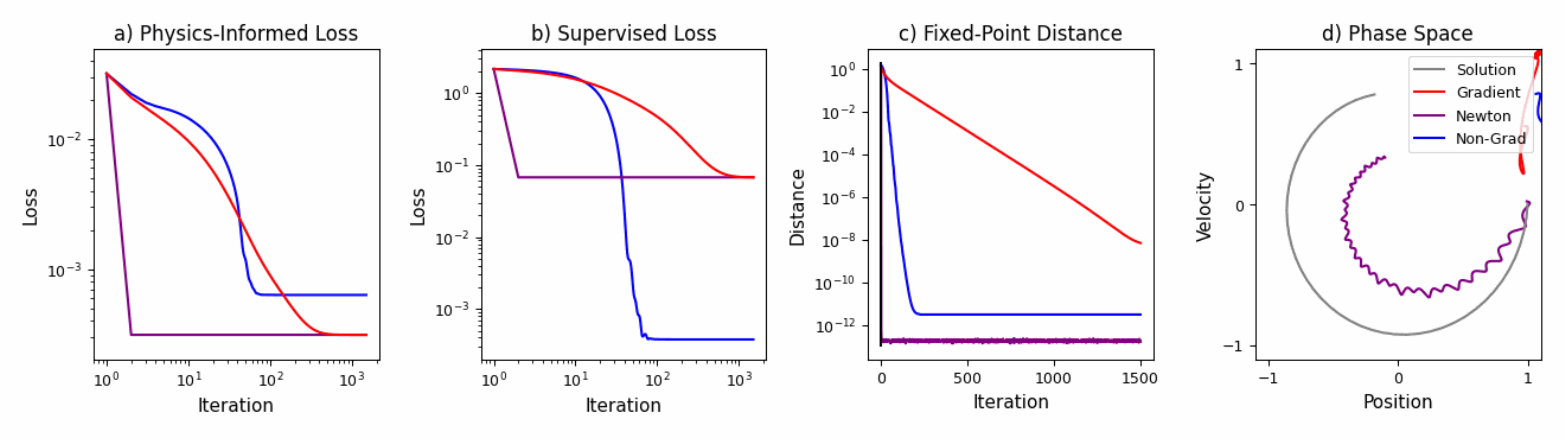
$$\theta_{Non-Gradient} = (BF)^{-1} Bc$$

The Problem with Gradients on Residual Objectives

- Even though we don't use it for training, our goal is to minimize a supervised loss.
- We are interested in a residual loss for training only because of its statistical benefits.
- Classical optimization methods attempt to find a minimum of the loss they are applied to.

Supervised objectives + Gradient methods	Residual objectives + Non-gradient methods

Harmonic Oscillator, Physics-Informed Training



Summary

Reinforcement Learning

- explores alternative approaches for multi-step tasks (value functions, temporal difference learning, non-gradient methods).
- shares deep analogies with physical learning techniques.
- adresses key computational, statistical and optimization bottlenecks.

References

- R. S. Sutton, Andrew G. Barto: ‘Reinforcement Learning, An Introduction’
- R. S. Sutton’s personal homepage: http://www.incompleteideas.net/
- L. C. Baird: ‘Reinforcement Learning through Gradient Descent'
- T. P. Lillicrap et al.: ‘Continuous Control with Deep Reinforcement Learning’
- D. Cheikhi & D. Russo: ‘On the Statistical Benefits of Temporal Difference Learning’
- P. Schnell et al.: ‘Temporal Difference Learning: Why It Can Be Fast and How It Will Be Faster’