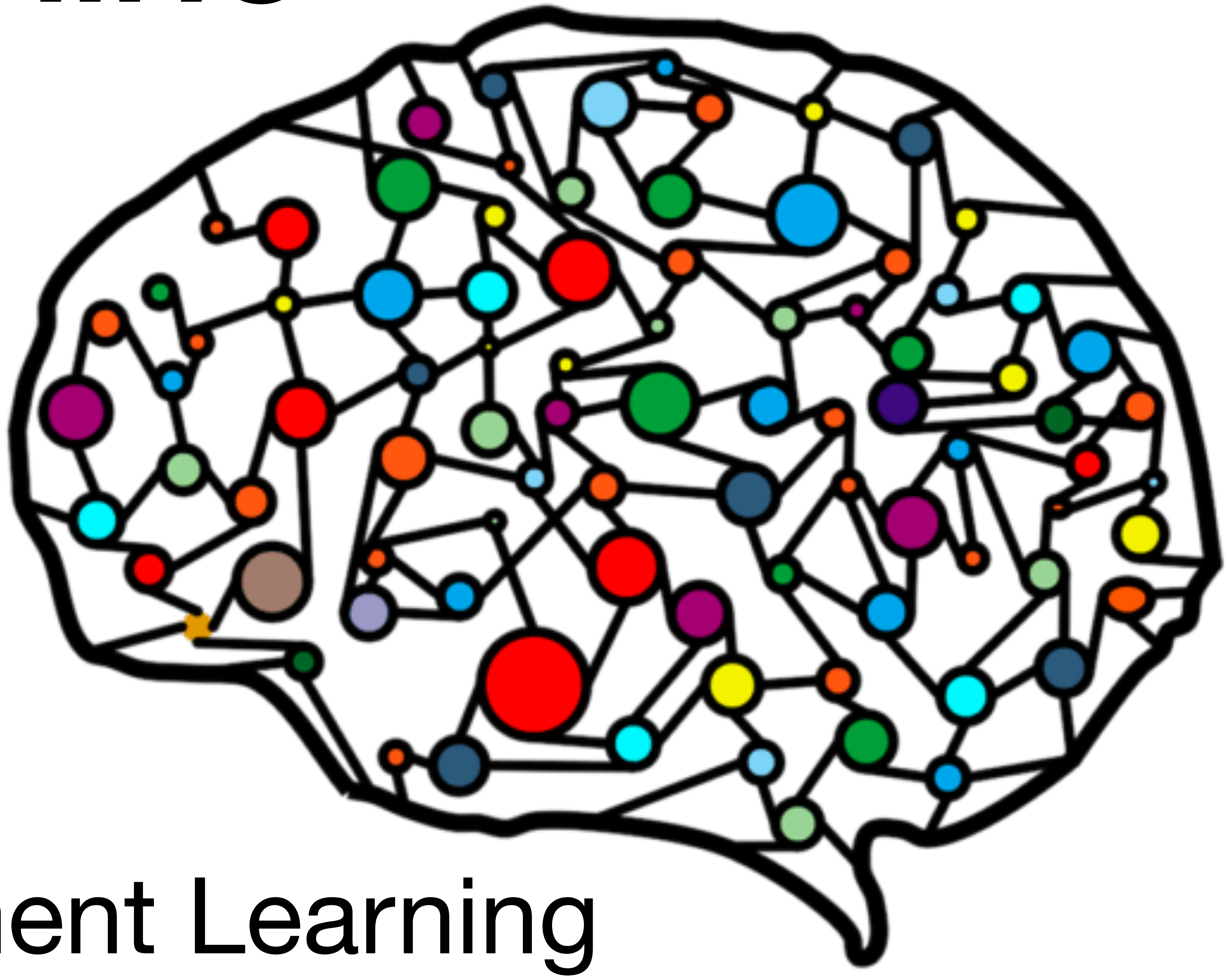
General Principles of Human and Machine Learning



Lecture 5: Advances in Reinforcement Learning

Dr. Charley Wu

https://hmc-lab.com/GPHML.html

Schedule

Week 6:	Guest lecturer: Alexandra Witt	Nov 19: Social learning	Nov 20	Alex	Witt et al., (2024)
Week 7:	Guest lecturer: Dr. David Nagy	Nov 26: Compression and resource constraints	Nov 27	David	Nagy et al,. (under review)
Week 8:		Dec 3: Concepts and Categories	Dec 4	Hanqi	Murphy (2023)
Week 9:		Dec 10: Supervised and Unsupervised learning	Dec 11	Hanqi	Bishop (Ch. 4)
	Holiday break				
Week 10:		Jan 14: Function learning	Jan 15	Alex	Wu, Meder, & Schulz (2024)
Week 11:		Jan 21: No Lecture	Jan 22: No Tutorial		
Week 12:		Jan 28: Language and semantics	Jan 29	TBD	Kamath et al., (2024)
Week 13:		Feb 4: General Principles	Feb 5	Charley	Gershman (2023)

Schedule

Week 6:	Guest lecturer: Alexandra Witt	Nov 19: Social learning	Nov 20	Alex	Witt et al., (2024)
Week 7:	Guest lecturer: Dr. David Nagy	Nov 26: Compression and resource constraints	Nov 27	David	Nagy et al,. (under review)
Week 8:		Dec 3: Concepts and Categories	Dec 4	Hanqi	Murphy (2023)
Week 9:		Dec 10: Supervised and Unsupervised learning	Dec 11	Hanqi	Bishop (Ch. 4)
	Holiday break				
Week 10:		Jan 14: Function learning	Jan 15	Alex	Wu, Meder, & Schulz (2024)
Week 11:		Jan 21: No Lecture	Jan 22: No Tutorial		
Week 12:		Jan 28: Language and semantics	Jan 29	TBD	Kamath et al., (2024)
Week 13:		Feb 4: General Principles	Feb 5	Charley	Gershman (2023)

Schedule

Week 6:	Guest lecturer: Alexandra Witt	Nov 19: Social learning	Nov 20	Alex	Witt et al., (2024)
Week 7:	Guest lecturer: Dr. David Nagy	Nov 26: Compression and resource constraints	Nov 27	David	Nagy et al,. (under review)
Week 8:		Dec 3: Concepts and Categories	Dec 4	Hanqi	Murphy (2023)
Week 9:		Dec 10: Supervised and Unsupervised learning	Dec 11	Hanqi	Bishop (Ch. 4)
	Holiday break				
Week 10:		Jan 14: Function learning	Jan 15	Alex	Wu, Meder, & Schulz (2024)
Week 11:		Jan 21: No Lecture	Jan 22: No Tutorial		
Week 12:		Jan 28: Language and semantics	Jan 29	TBD	Kamath et al., (2024)
Week 13:		Feb 4: General Principles	Feb 5	Charley	Gershman (2023)

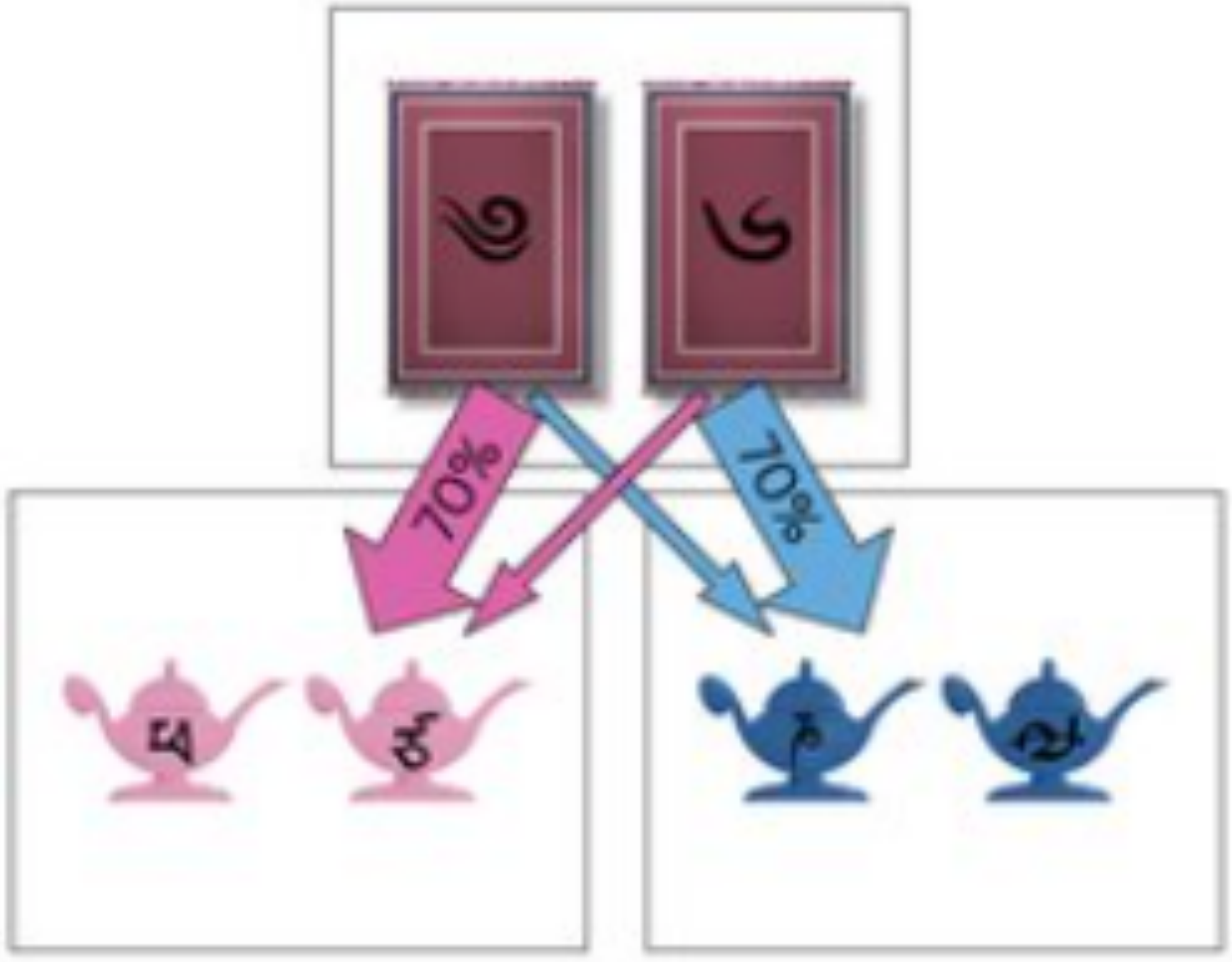
Swapped

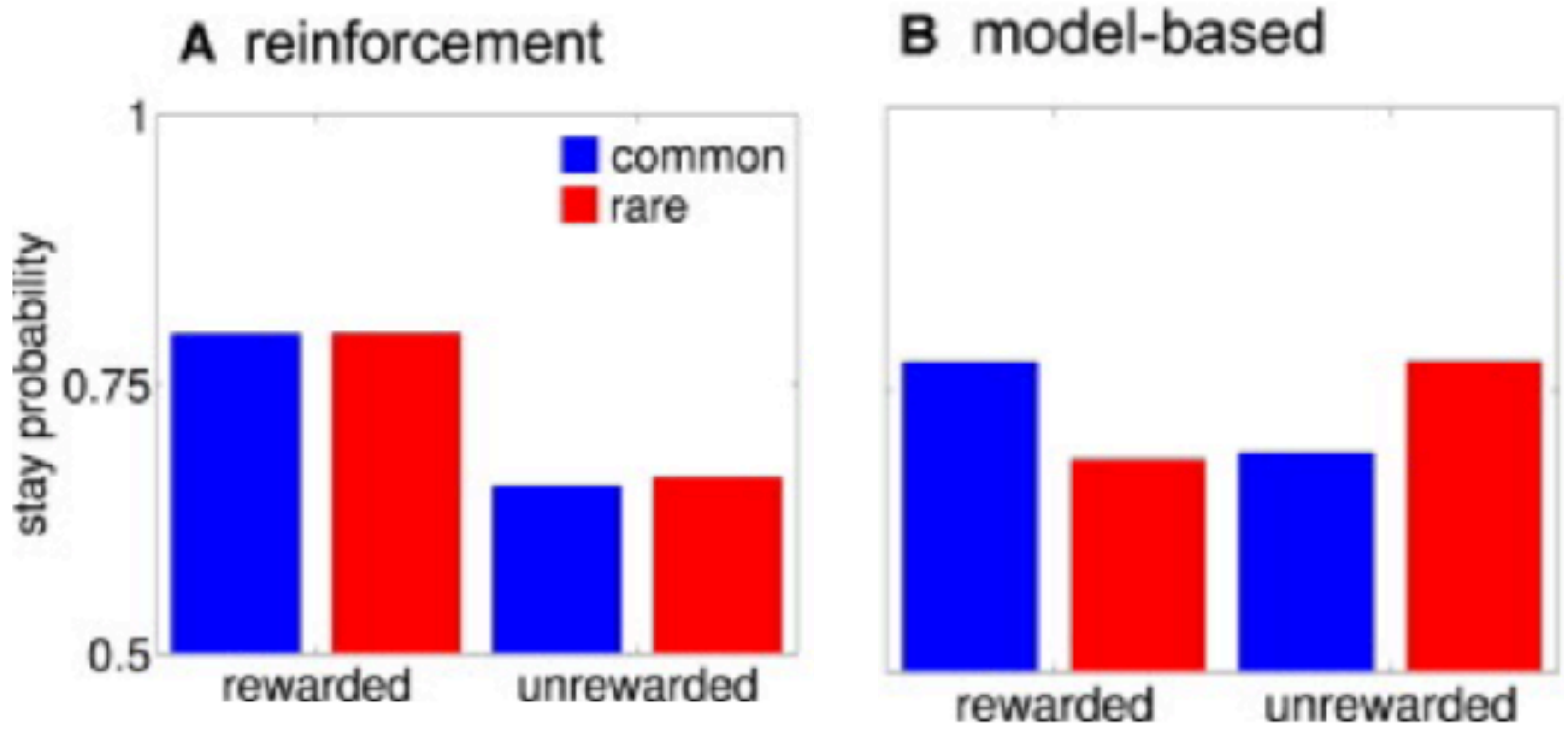
Exam times

Exam 1	13:00-15:00 21.02.2025 Hörsaal 1 F119 (SAND)
Exam 2	12:00-14:00 11.04.2025 (to be confirmed)

Clarification

- Two-step tasks
  - Transitions are a property of the environment and the participants choice
  - The participant chooses  or 
    - But then probabilistically (common=70% vs. rare = 30%), transitions to either pink or blue on the 2nd step
- The key takeaway is that MF vs MB have different responses to the same outcome
  - MF: If rewarded —> stay
    if not rewarded —> go
  - MB: depends on whether reward followed a common or rare transition… *you shouldn't expect a rare transition to occur again*
    if (common & reward | rare & no reward) —> stay
    if (common & no reward| rare & reward) —> stay





Last week…

Sutton and Barto (2018 [1998])

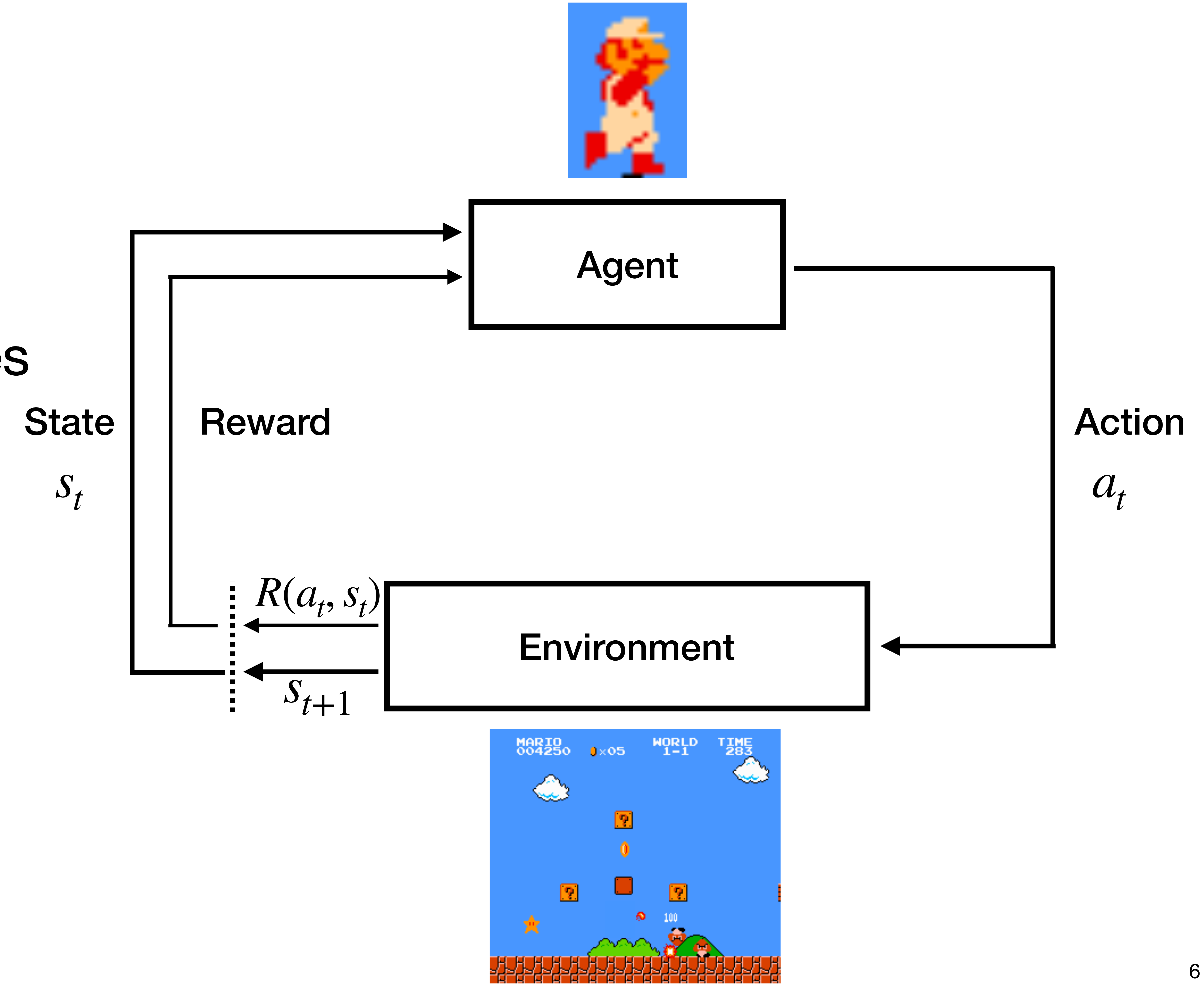
Reinforcement Learning

The Agent:

- Selects actions a_t
- Receives feedback from the environment in terms of new states s_{t+1} and rewards $R(a_t, s_t)$

The Environment:

- Governs the transition between states $s_t \rightarrow s_{t+1}$
- Provides rewards $R(a_t, s_t)$



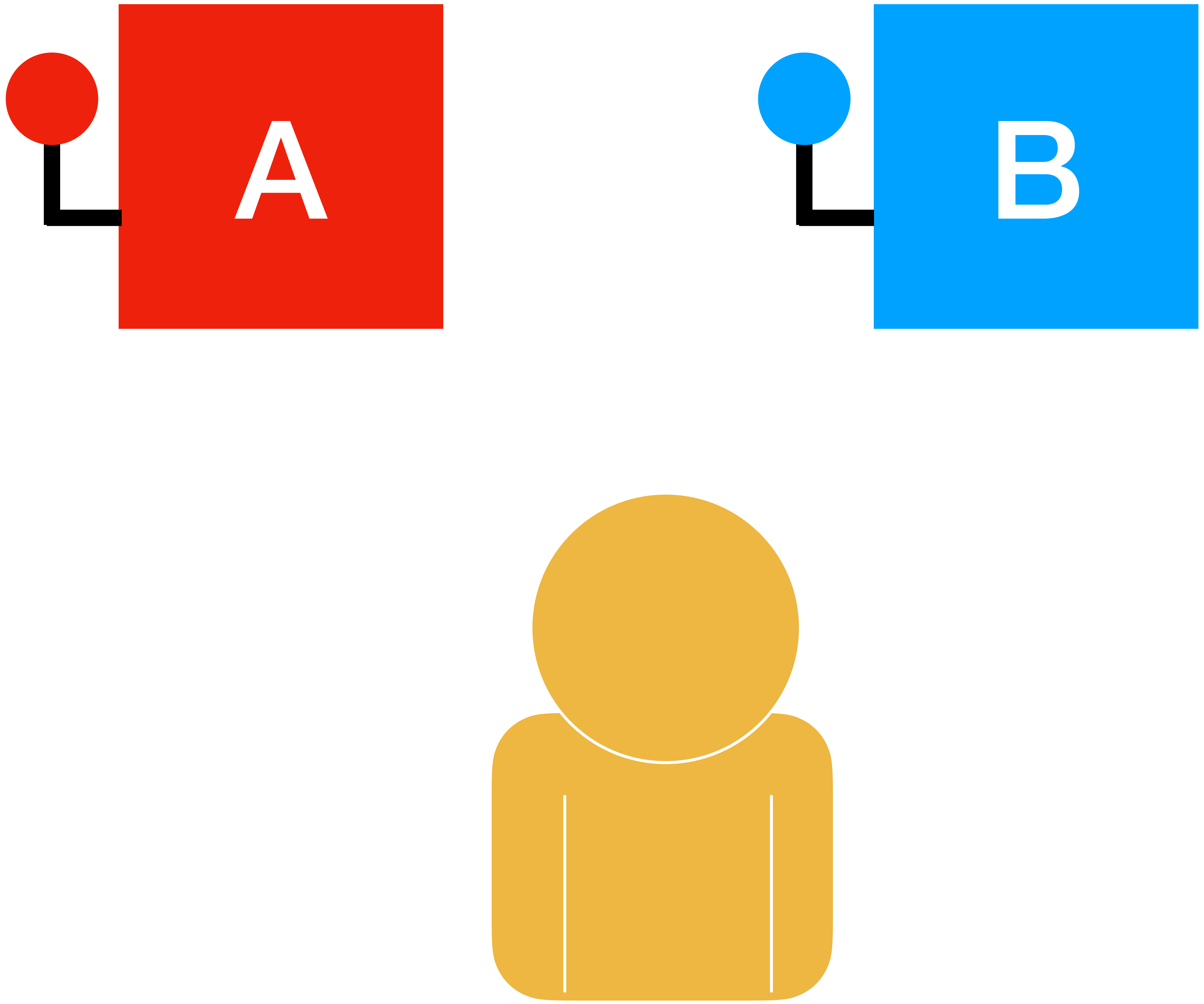
Q-Learning in a bandit task

Value learning

$$Q_t(a) \leftarrow Q_t(a) + \eta \left[r - Q_t(a) \right]$$

Policy

$$P(a) \propto \exp(Q_t(a)/\tau)$$



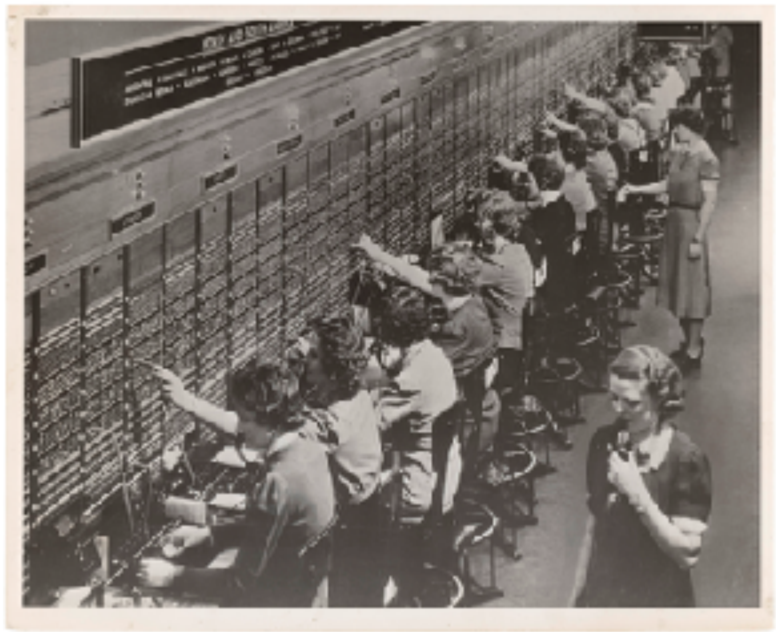
Model-free RL ⟷ Model-based RL

Model-free RL

- Habit
- Cheap
- $Q(s, a)$
- Myopically selecting actions that have been associated with reward

Model-based RL

- Goal-directed
- Computationally costly
- $P(s', r \mid s, a)$
- Planning and seeking of long term outcomes



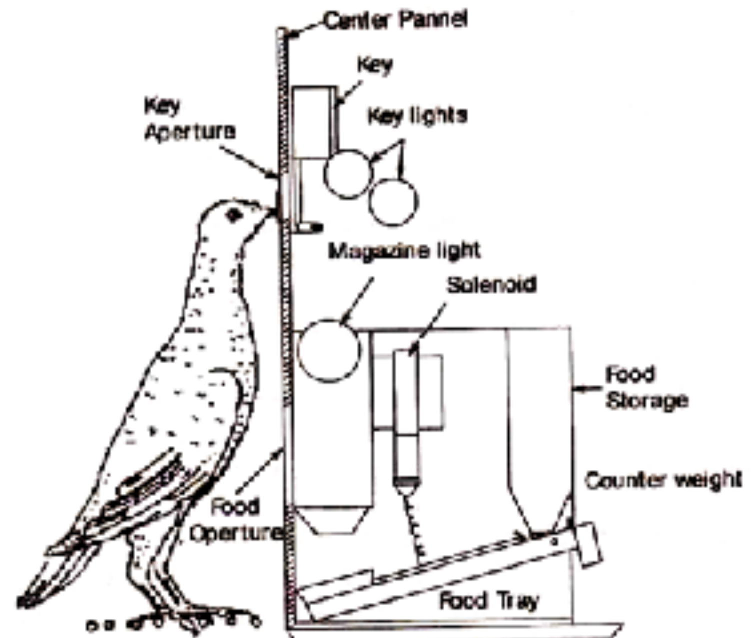
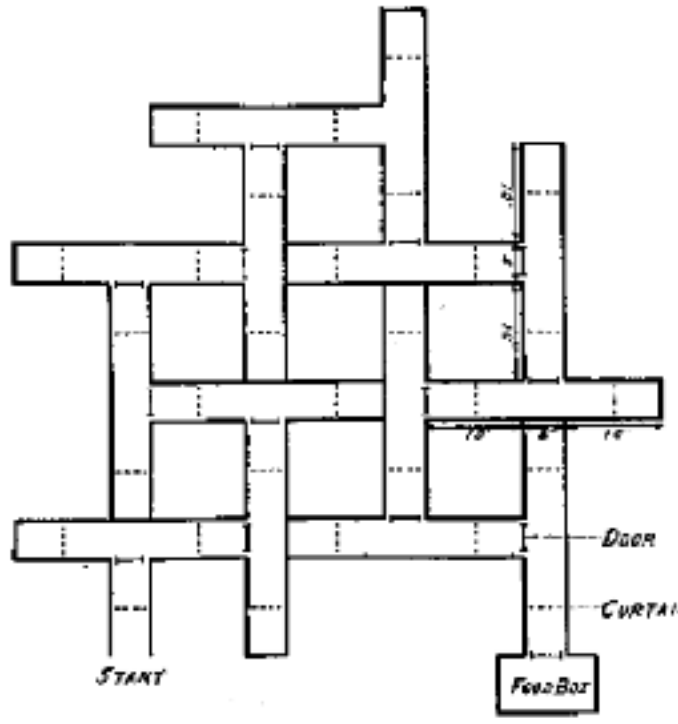
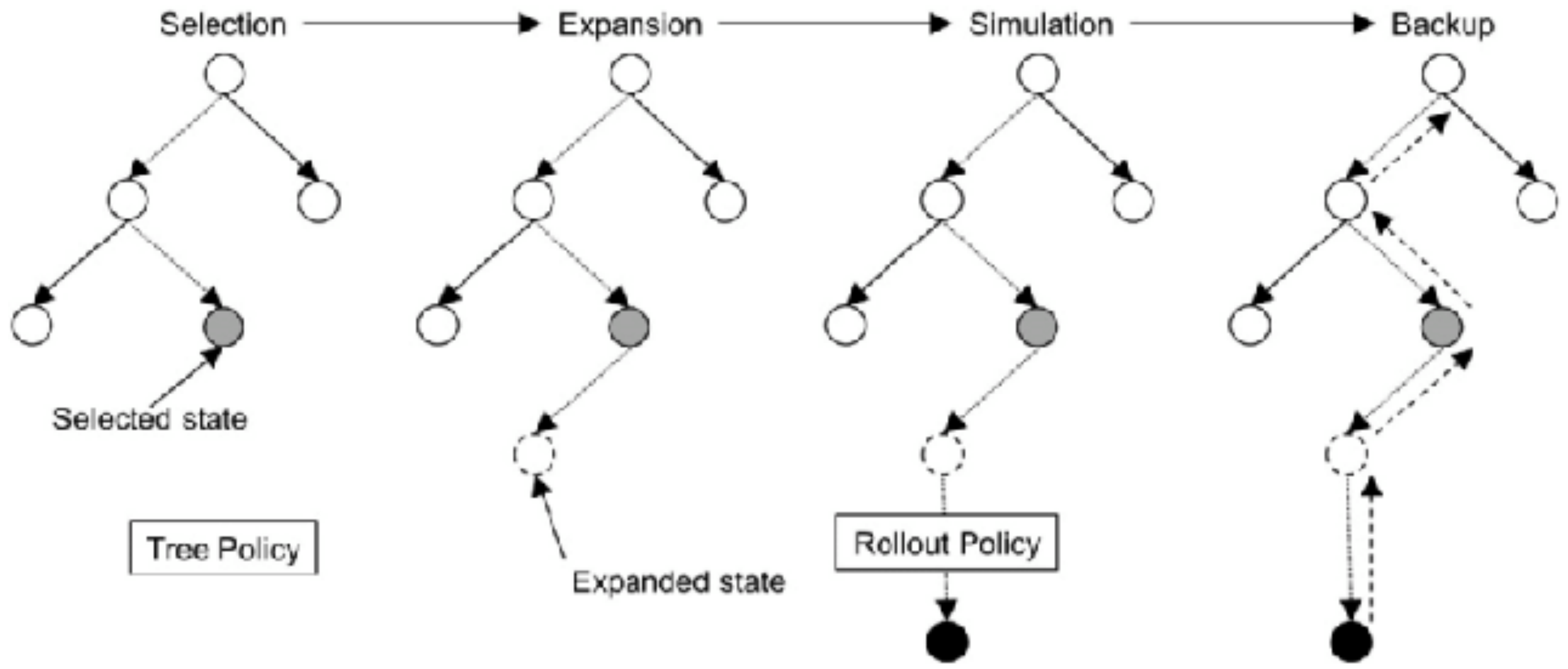


Illustration. Skinner box as adapted for the pigeon.



Monte carlo tree search

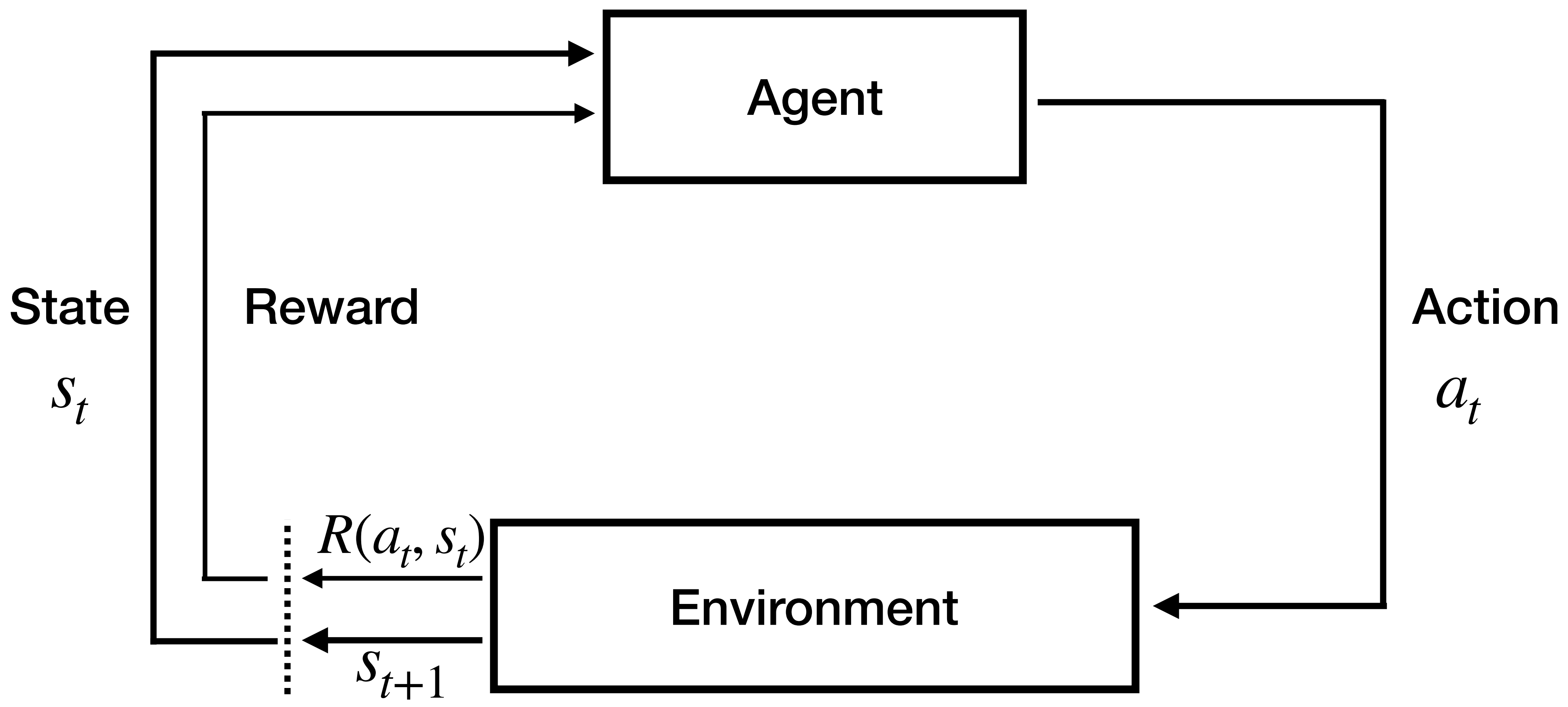


Duarte et al,. (2020)

Today's agenda

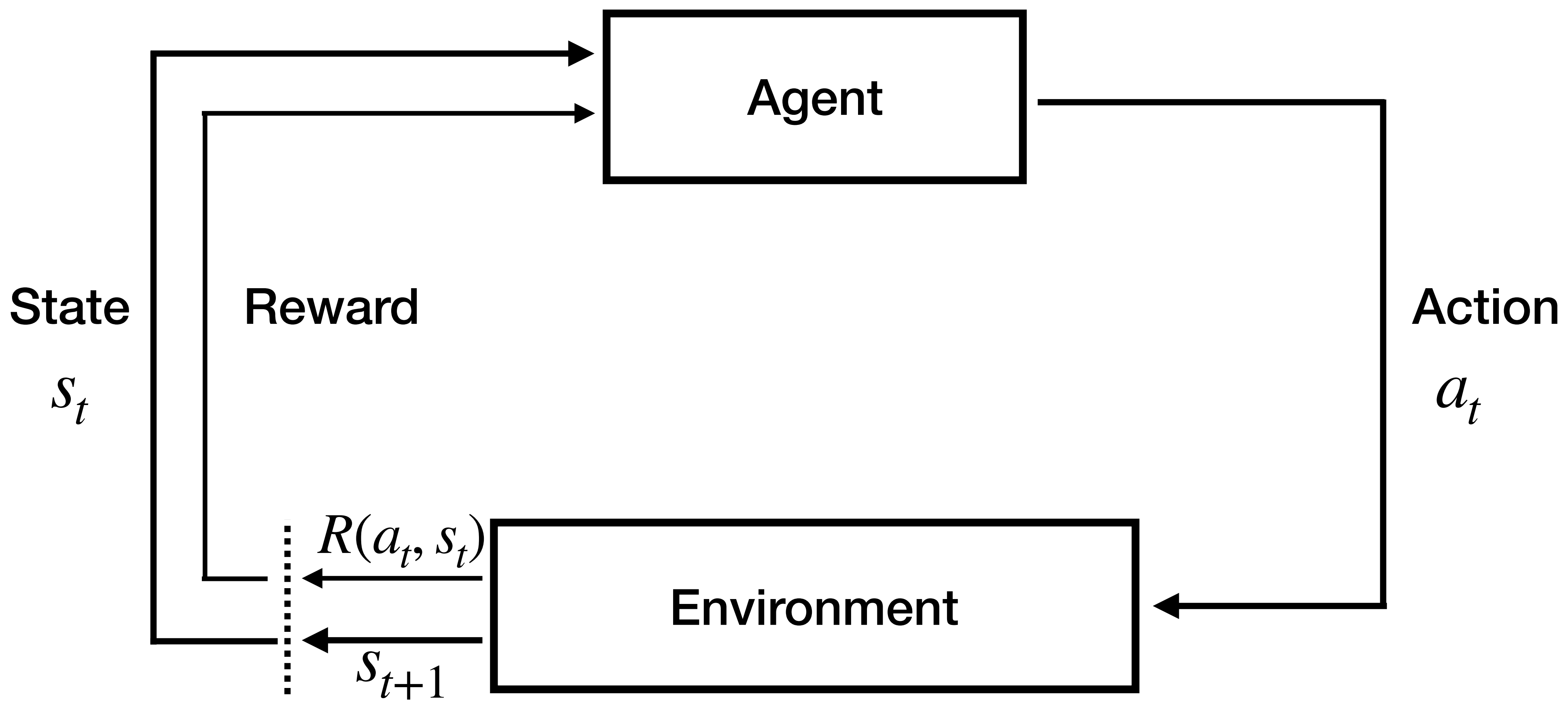
- Advances in …
  - **Model-free methods**
    - Deep Q-learning, policy gradient & Actor-Critic
  - **Model-based methods**
    - DYNA, World models, & Dreamer
  - **Something in between**
    - Successor representation

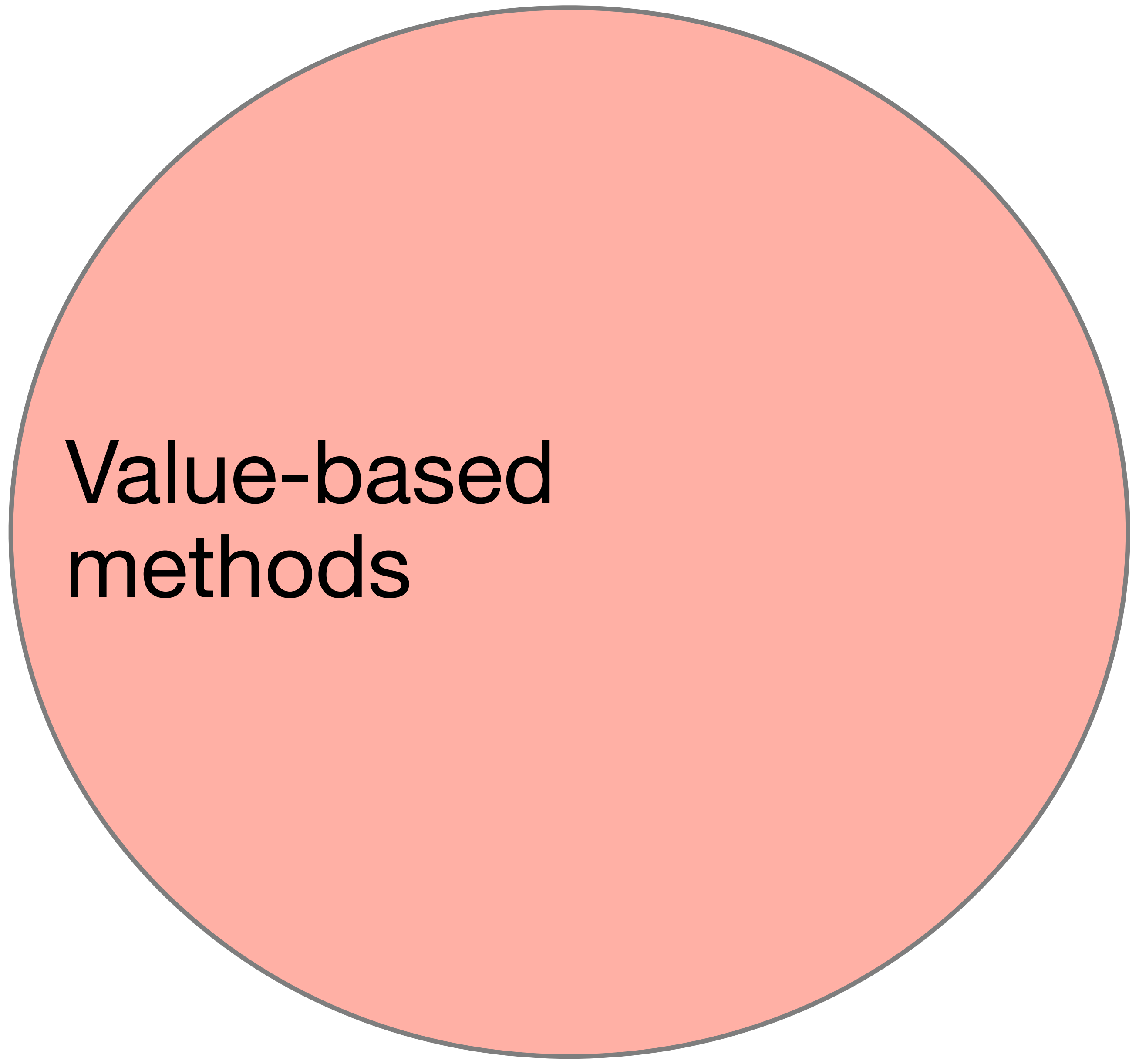
Advances in Model-free RL



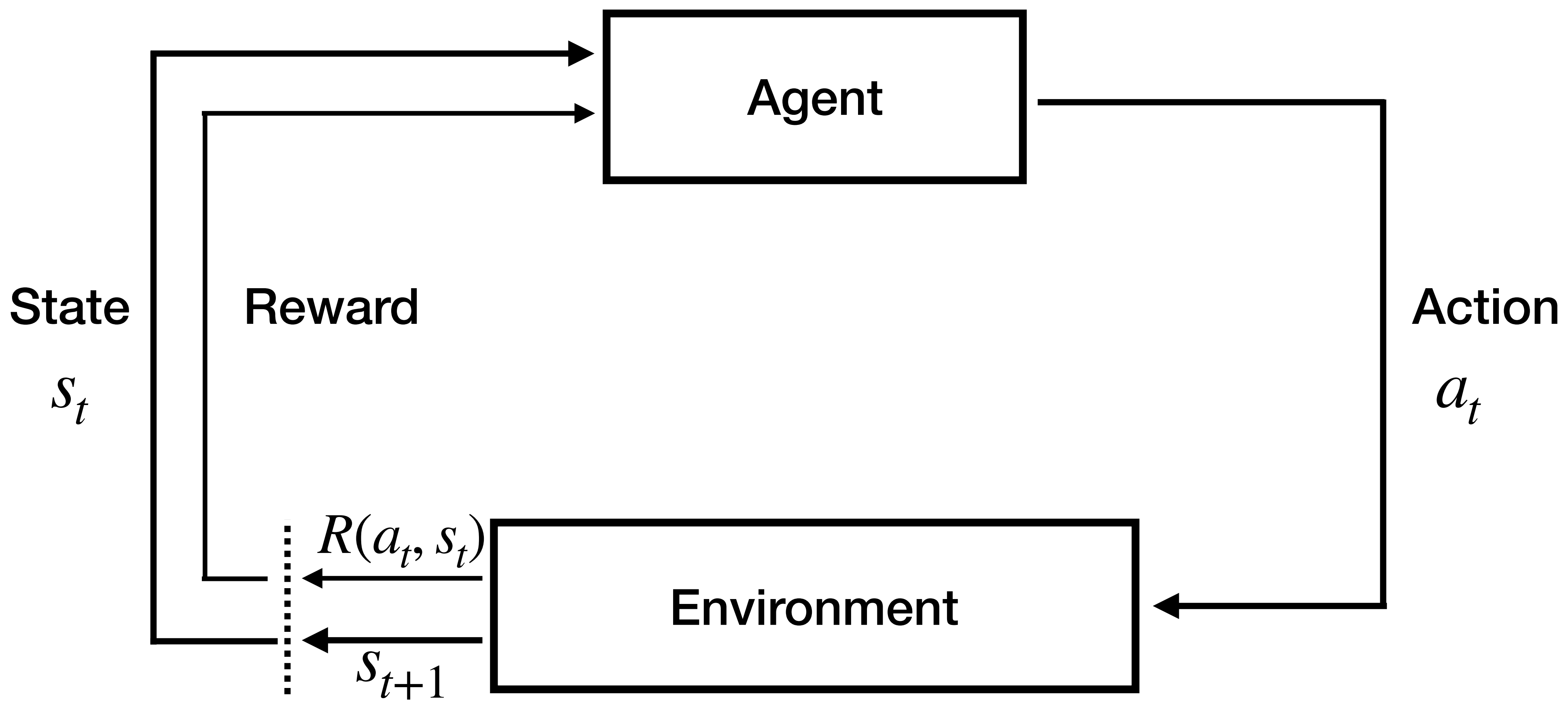
Advances in Model-free RL

- **Value-based methods**
  - Last week: Value iteration, Q-Learning & TD-learning
    - Problem: what if the state-space is too large to visit?
  - *Deep Q-Learning* for function approximation

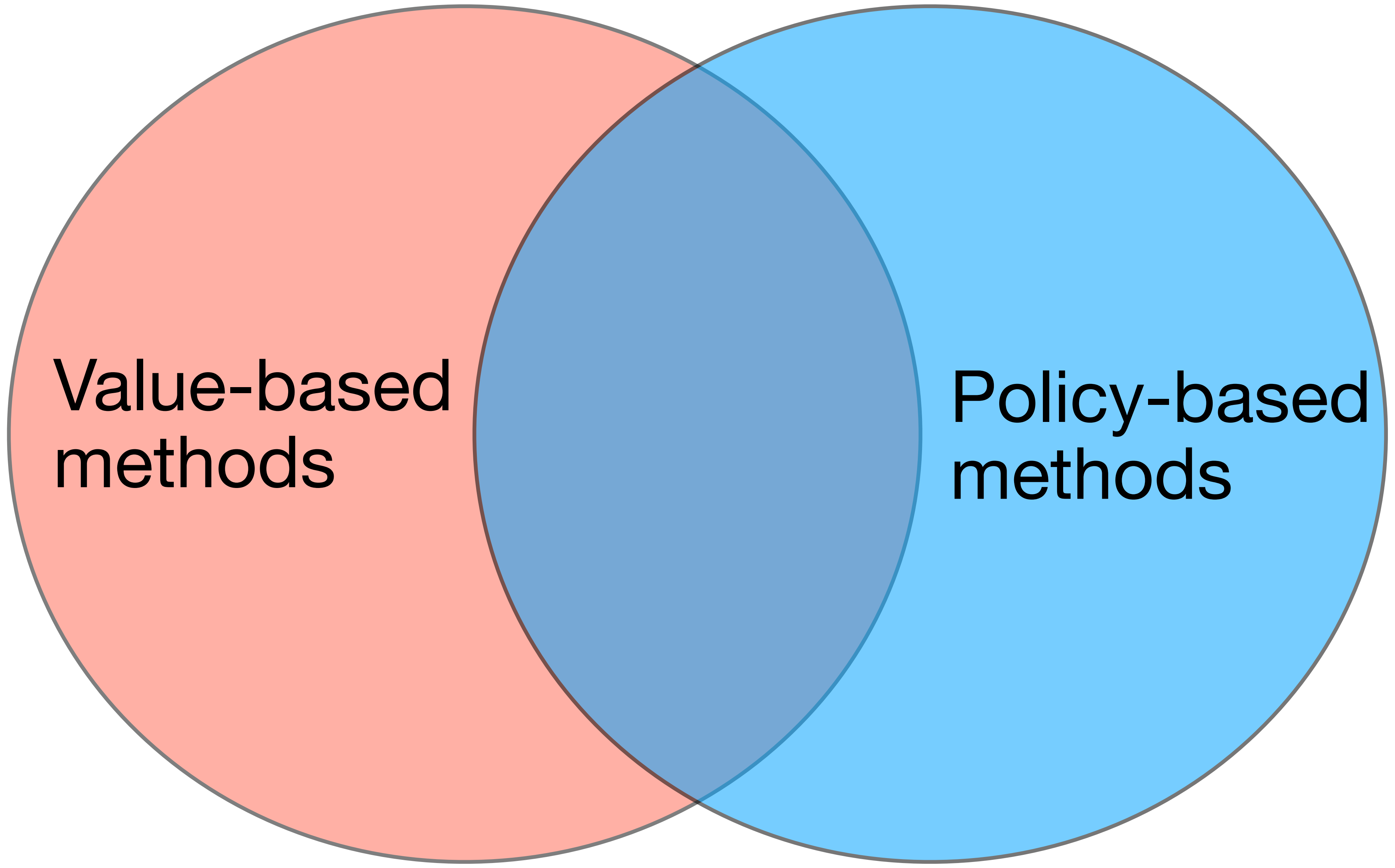




Advances in Model-free RL

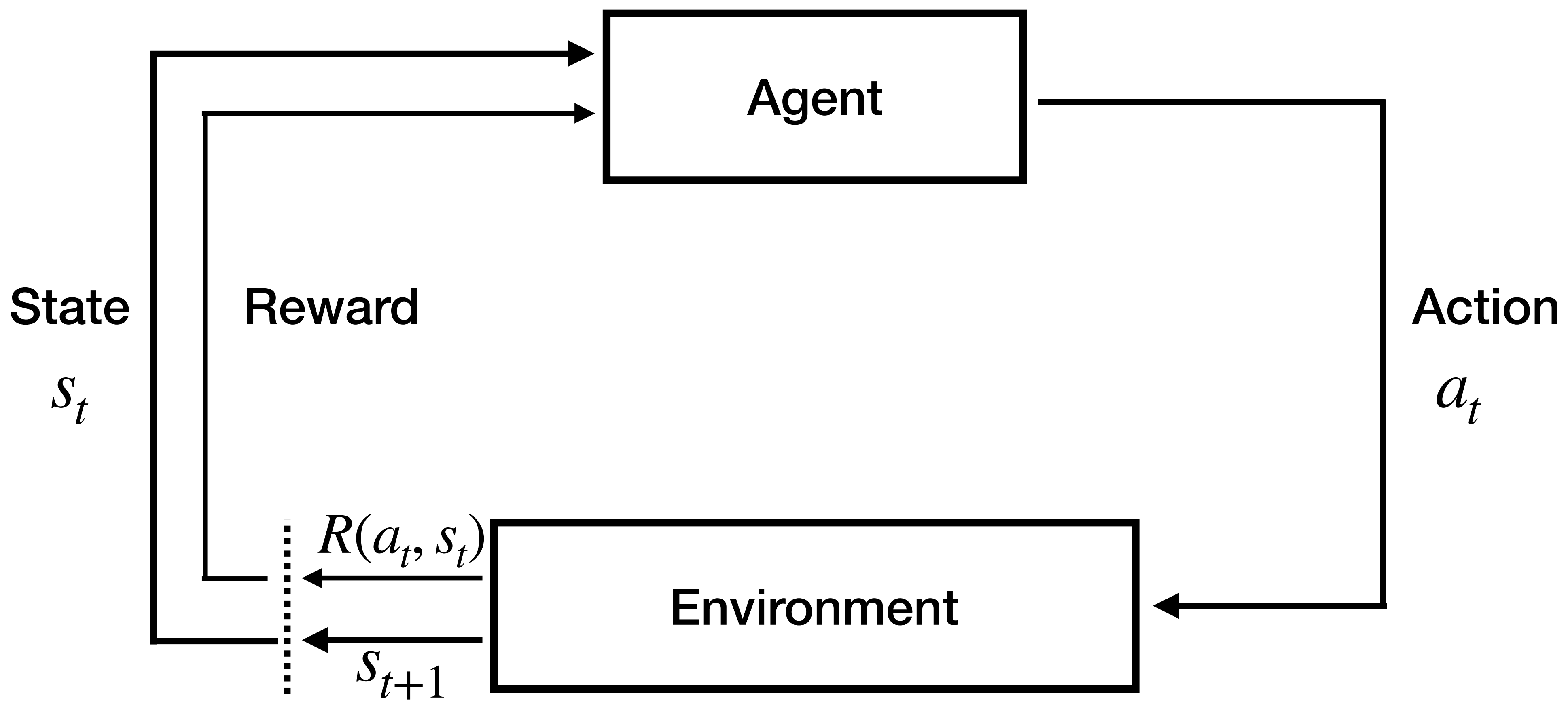


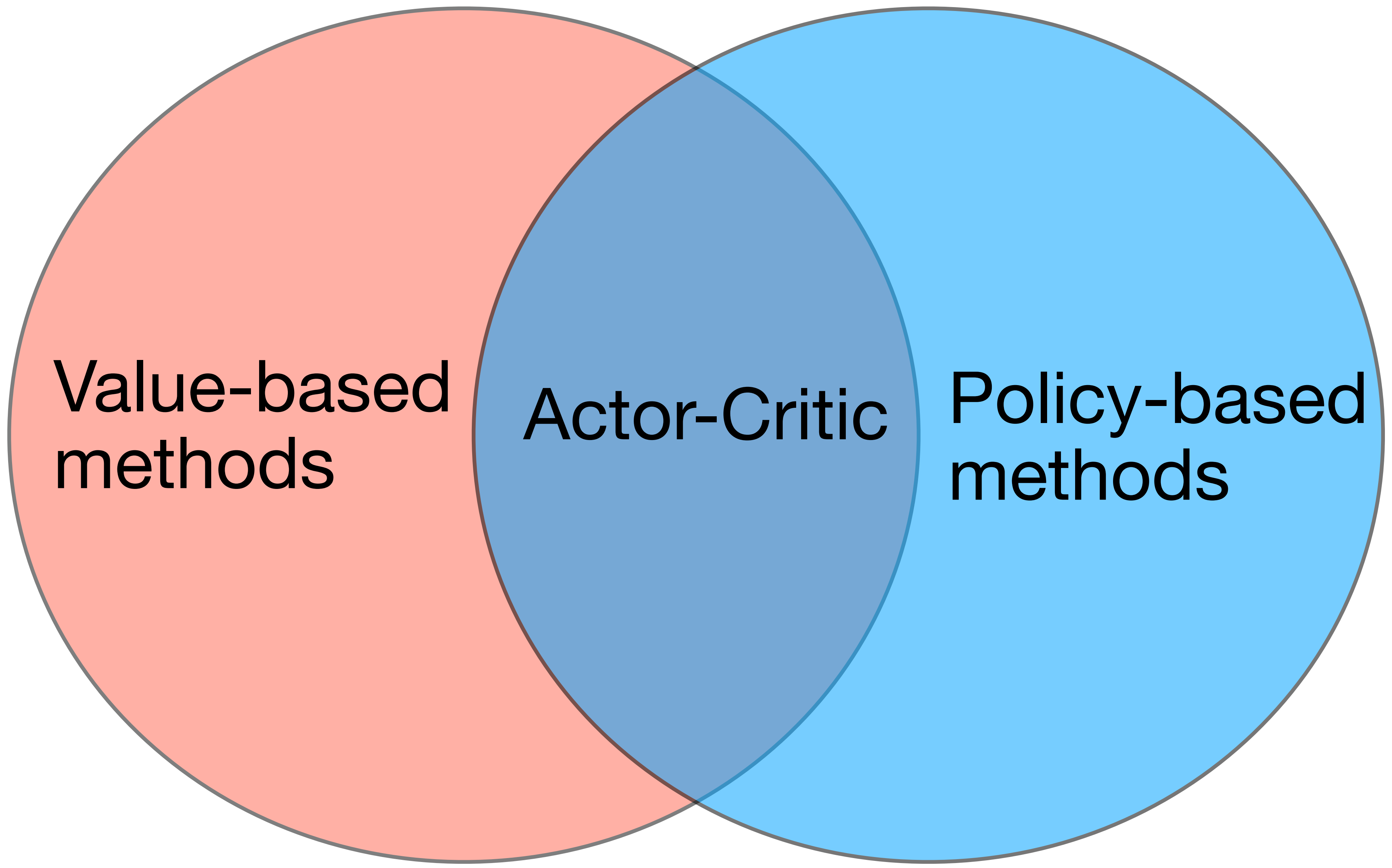
- **Value-based methods**
  - Last week: Value iteration, Q-Learning & TD-learning
    - Problem: what if the state-space is too large to visit?
  - *Deep Q-Learning* for function approximation
- **Policy-based methods**
  - Policy-gradient for directly optimizing a policy



Advances in Model-free RL

- **Value-based methods**
  - Last week: Value iteration, Q-Learning & TD-learning
    - Problem: what if the state-space is too large to visit?
  - *Deep Q-Learning* for function approximation
- **Policy-based methods**
  - Policy-gradient for directly optimizing a policy
- **Actor-Critic**
  - Modern version of Policy-iteration: Value ⟷ Policy

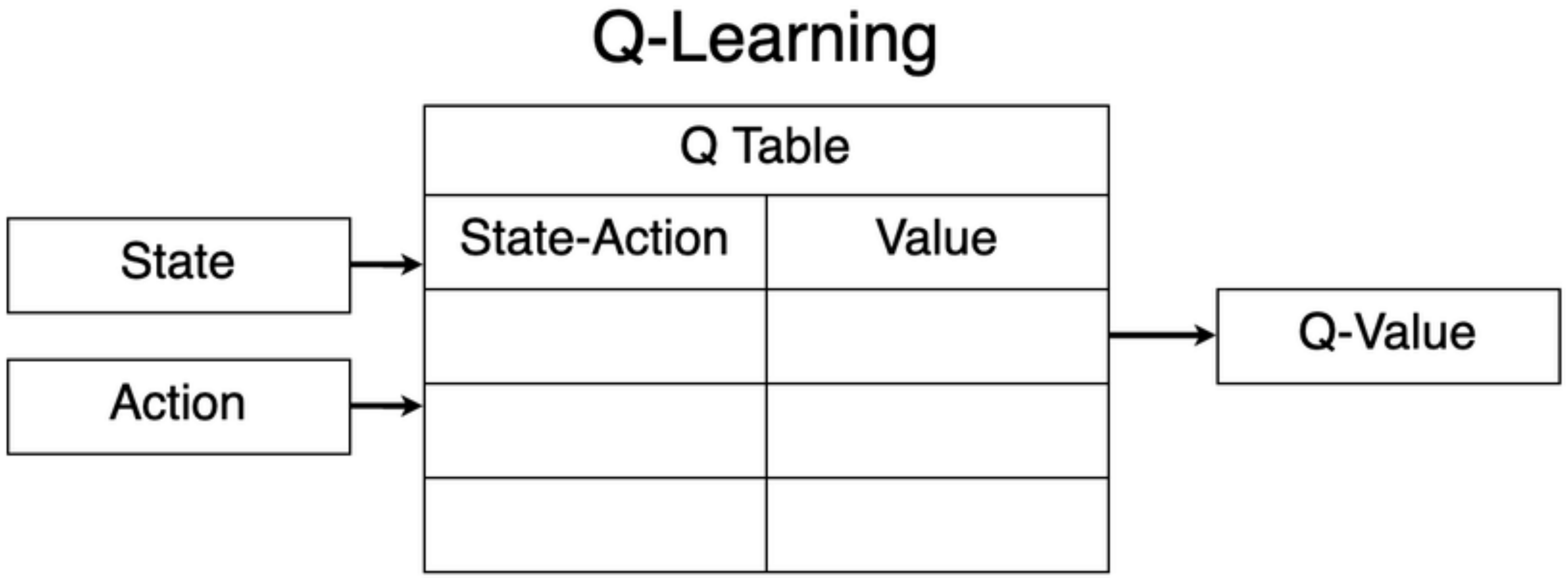




Deep Q-learning

Tabular methods:

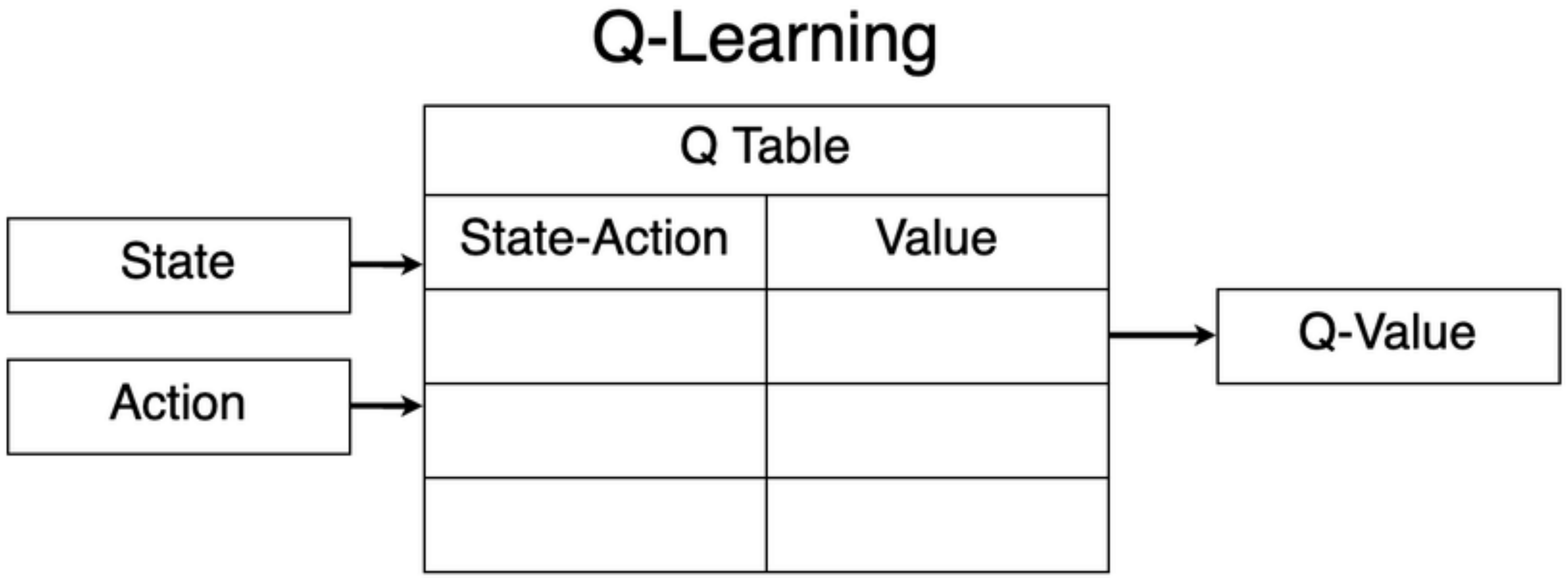
- Q-Learning: learn Q-values by updating a look-up table



Deep Q-learning

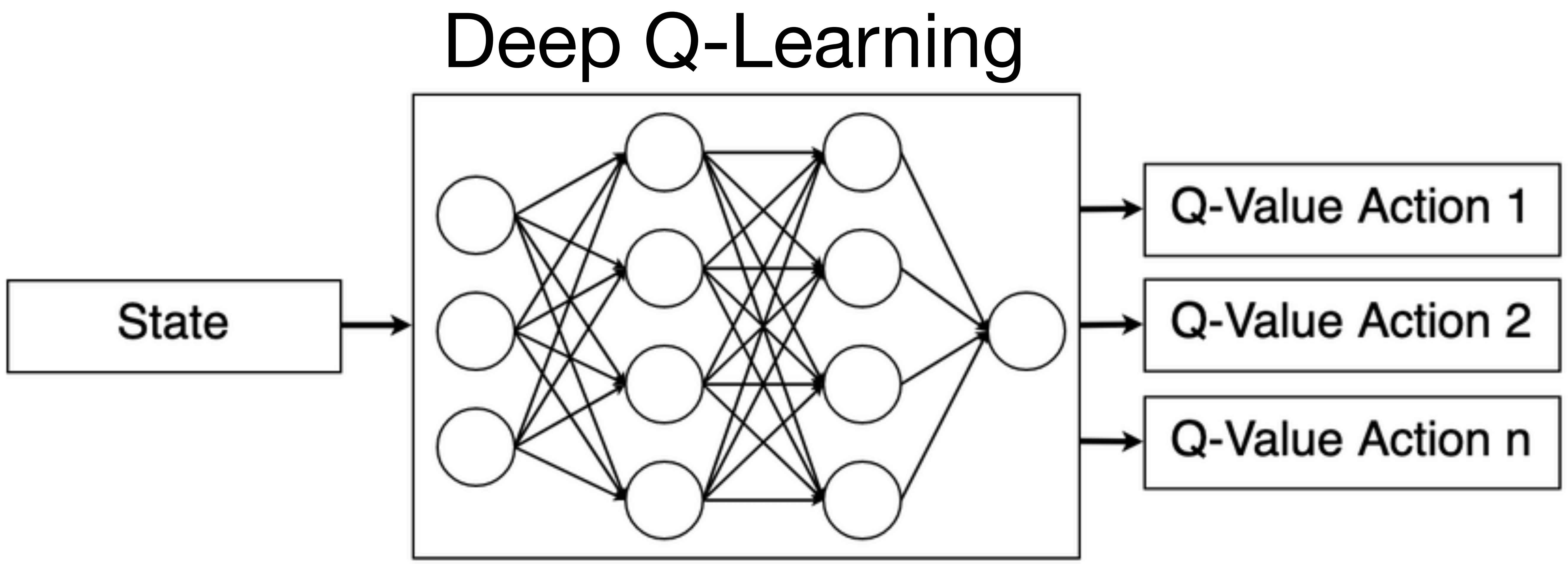
Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table



Function approximation:

- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$



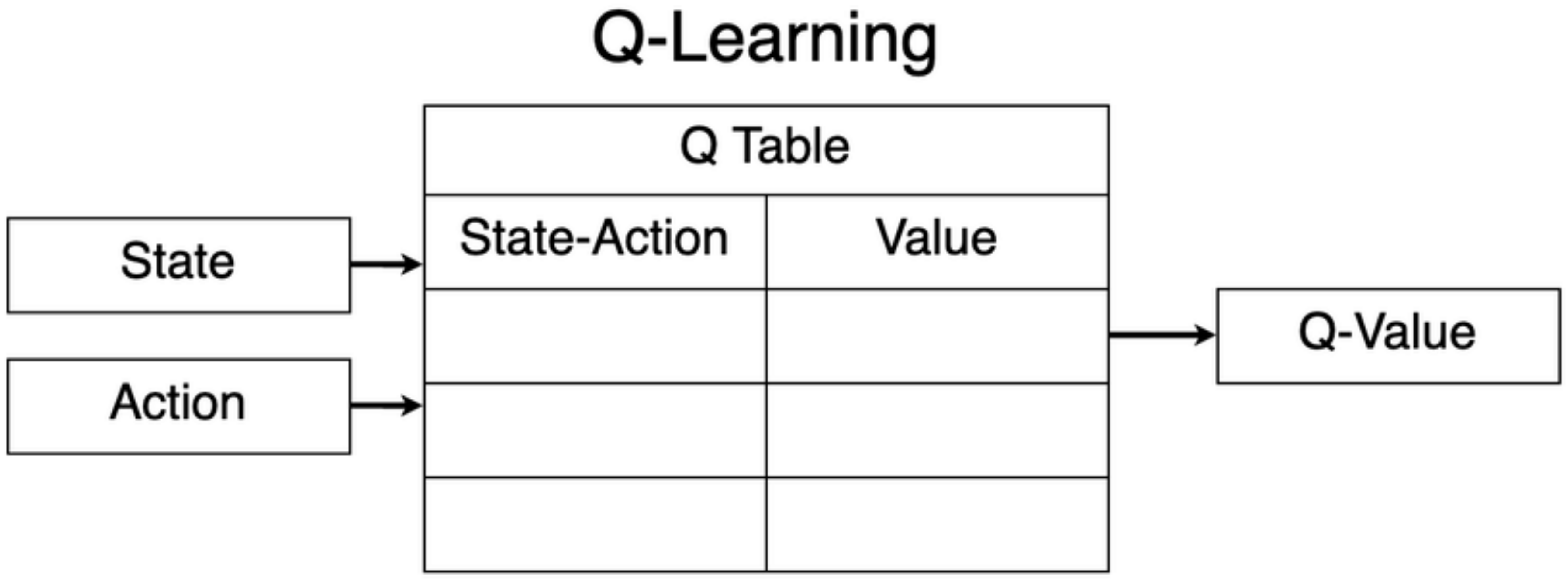
Deep Q-learning

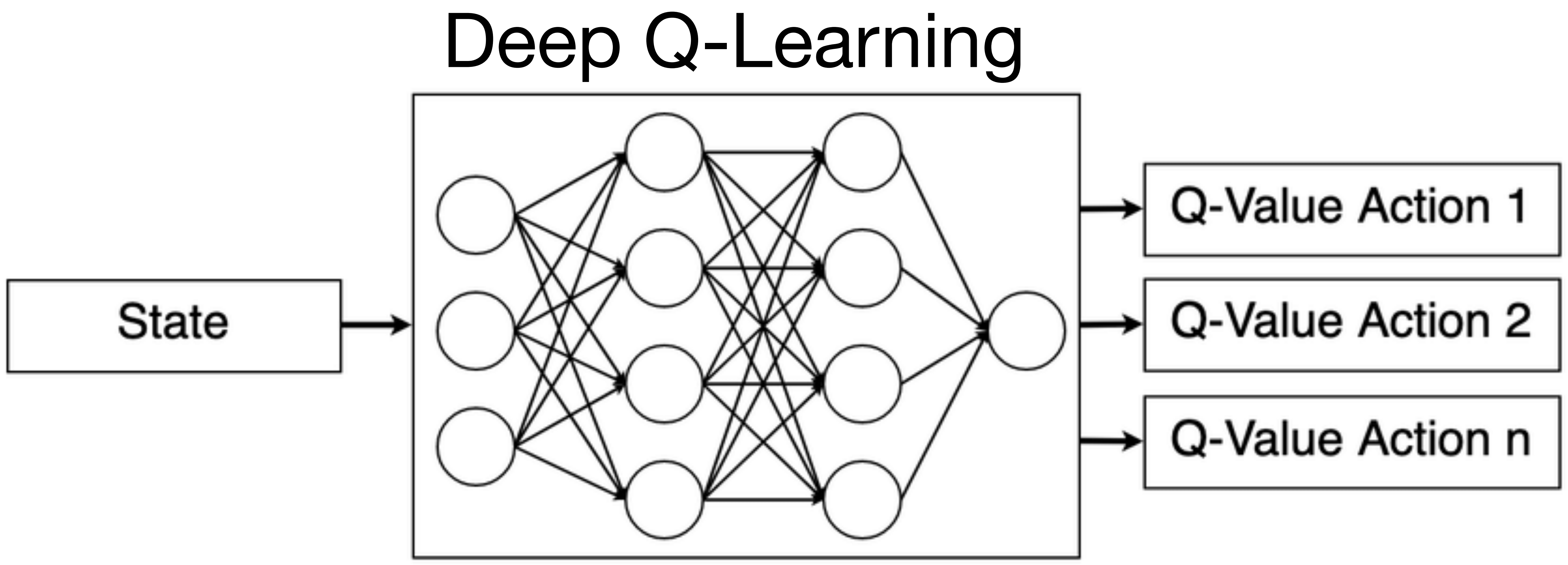
Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table

Function approximation:

- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$
- Weight updates:
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\delta\nabla_{\mathbf{w}}Q_{\mathbf{w}}(s, a)$

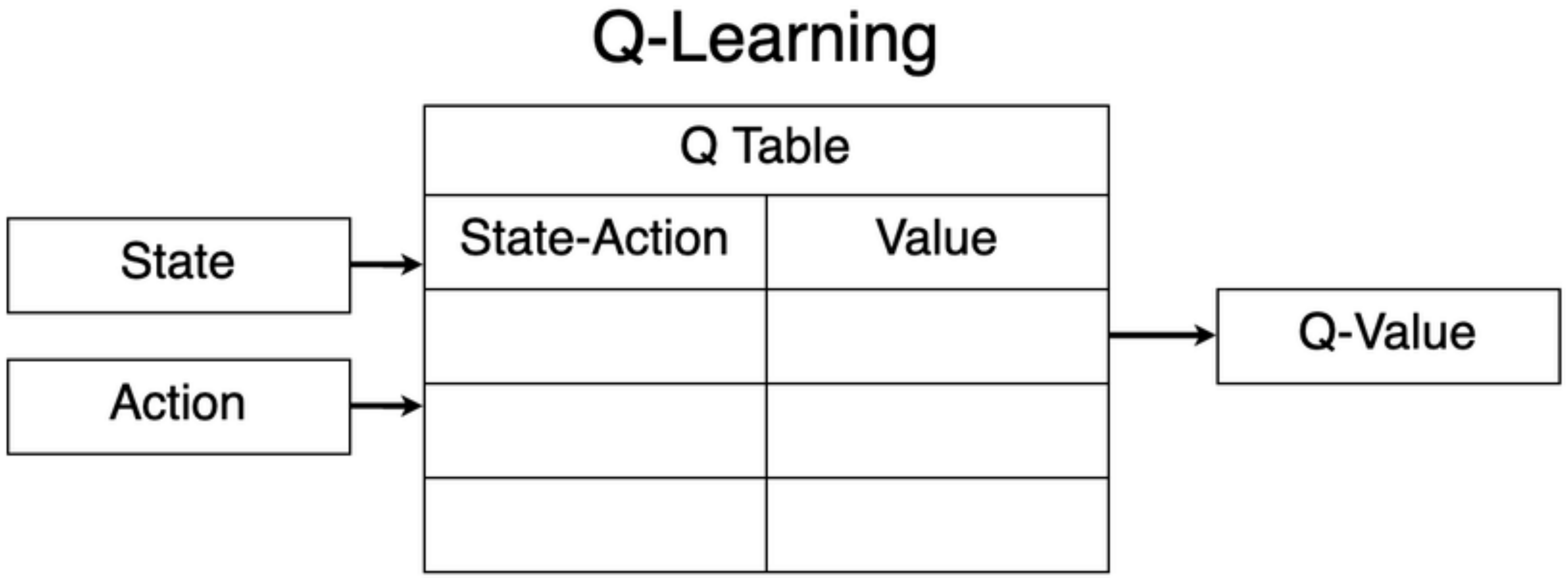




Deep Q-learning

Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table

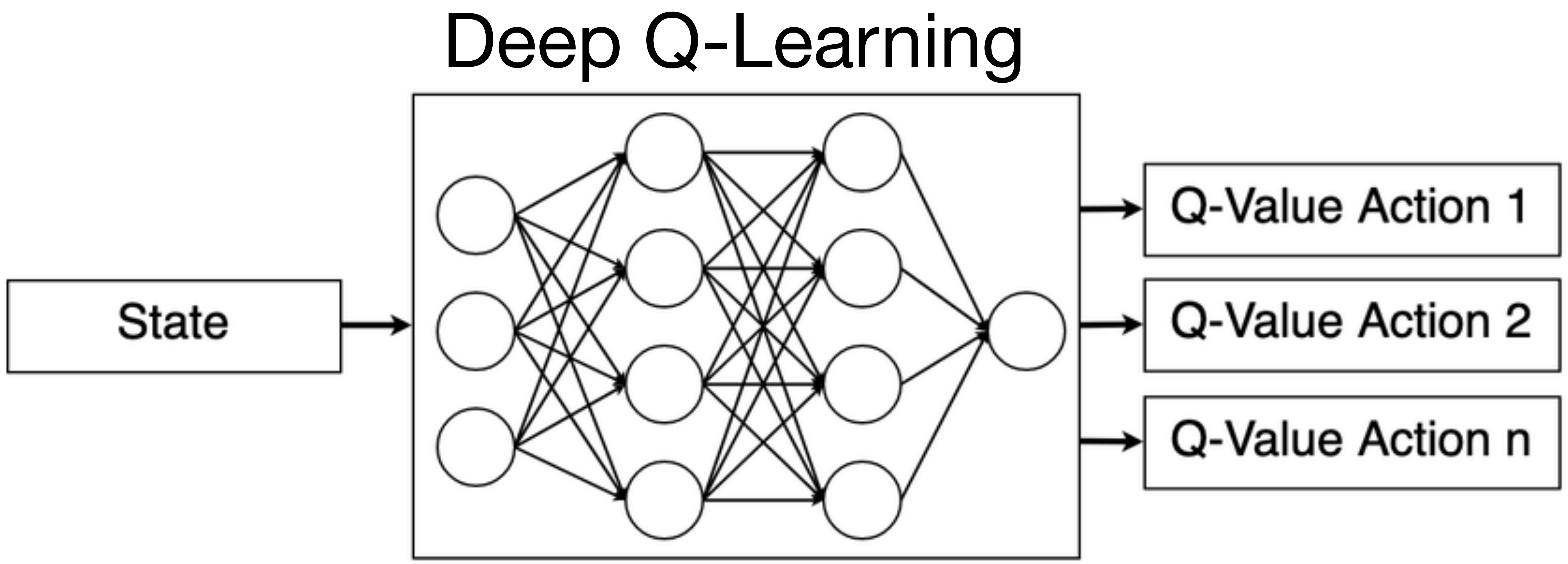


Function approximation:

- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$
- Weight updates:

learning rate

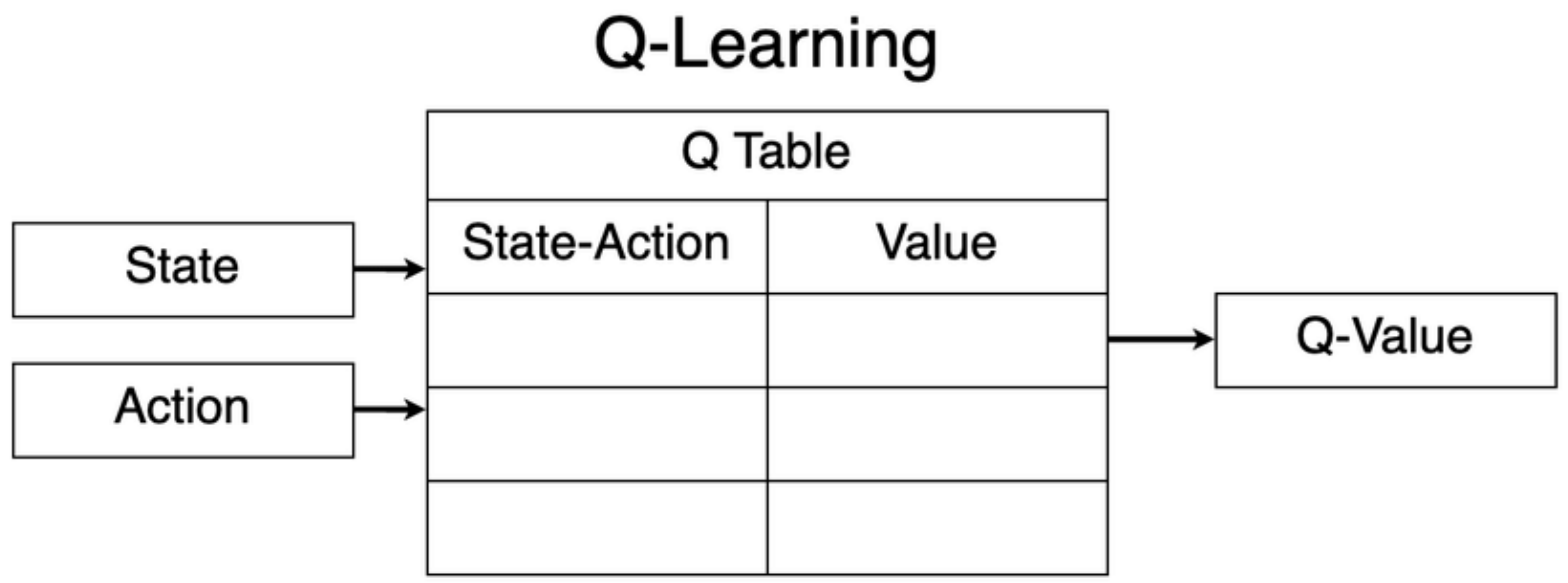
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$$



Deep Q-learning

Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table

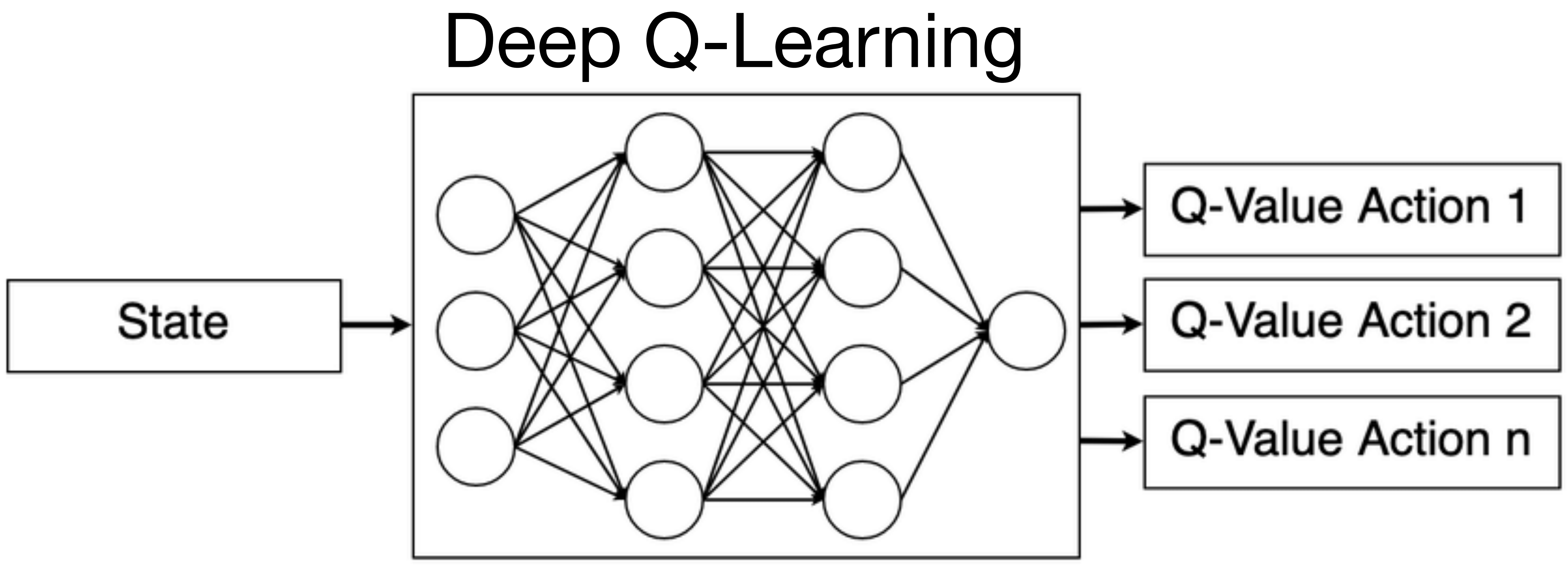


Function approximation:

- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$
- Weight updates: learning rate

  $$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$$
- **TD prediction error**

  $$\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$$



Deep Q-learning

Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table

Function approximation:

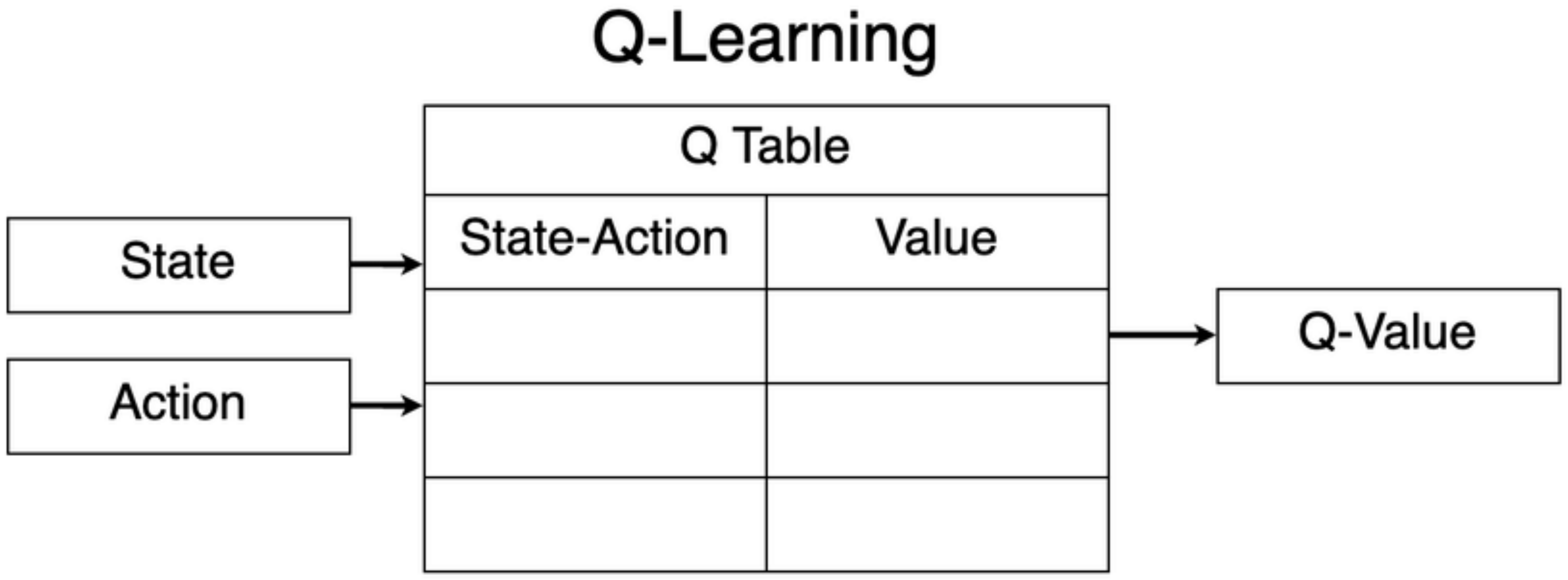
- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$
- Weight updates: learning rate

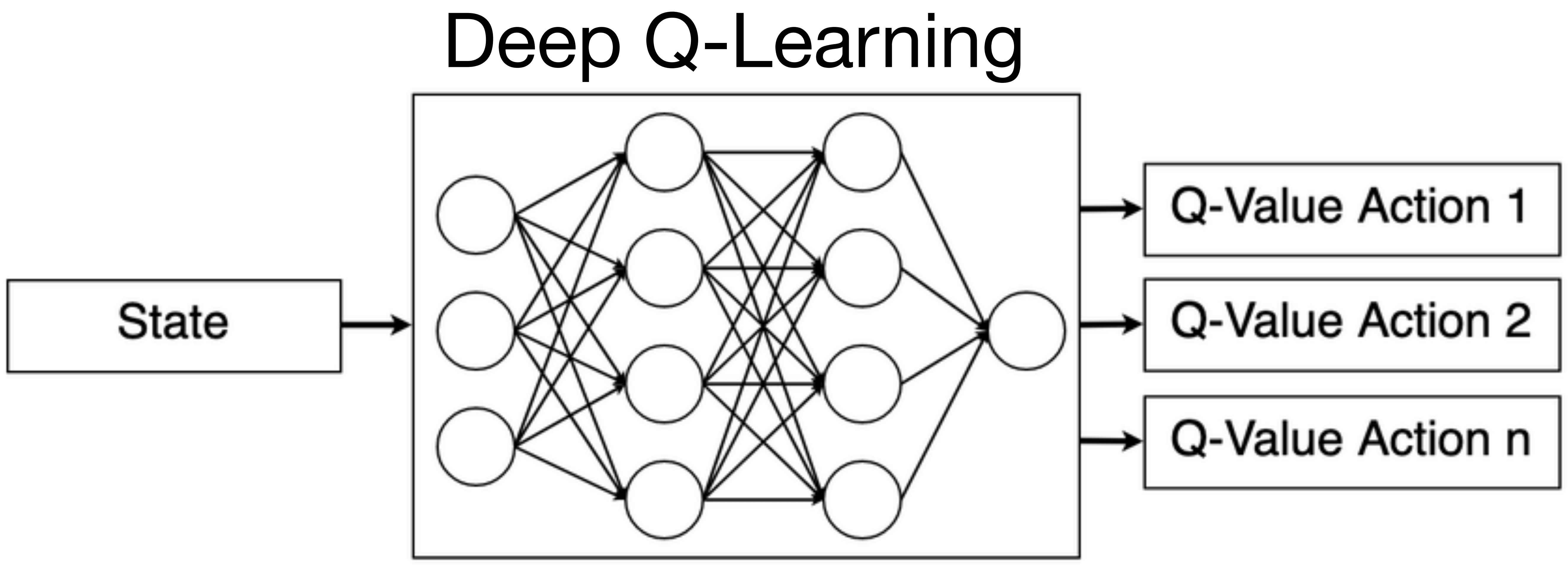
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$$

- **TD prediction error**

$$\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$$

- **Gradient** of Q-function w.r.t. to $\mathbf{w}$, trying to reduce prediction error!

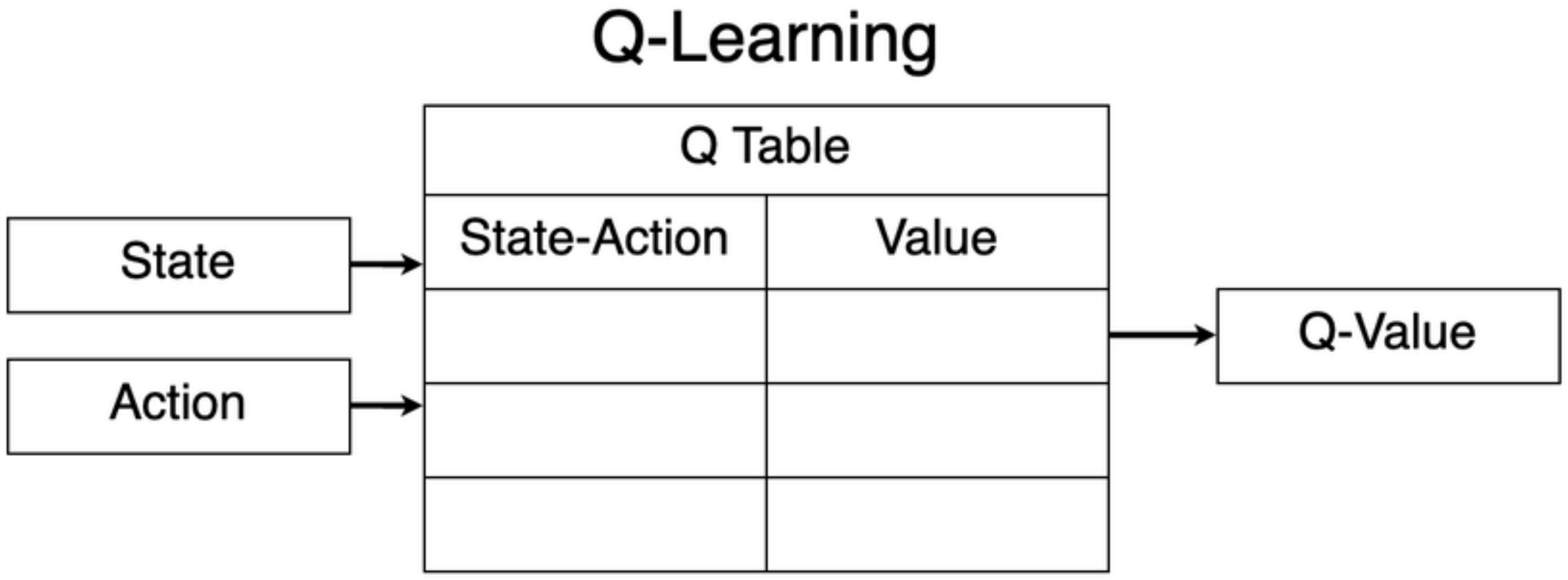




Deep Q-learning

Tabular methods:

- Q-Learning: learn Q-values by updating a look-up table



Function approximation:

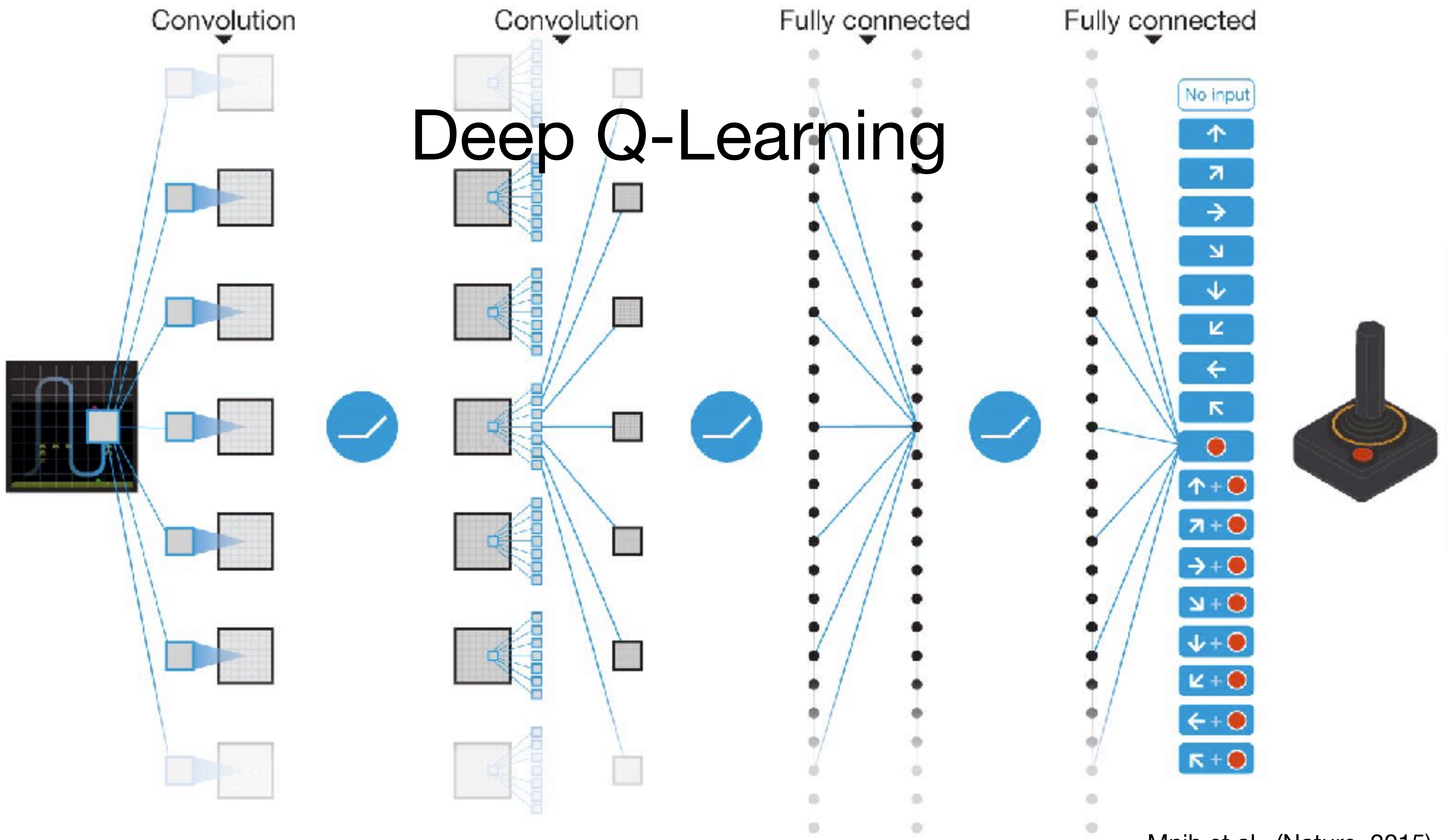
- Deep Q-Learning: use a deep ANN to learn $Q_{\mathbf{w}}(s, a)$
- Weight updates: learning rate

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$$

- **TD prediction error**

$$\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$$

- **Gradient** of Q-function w.r.t. to $\mathbf{w}$, trying to reduce prediction error!

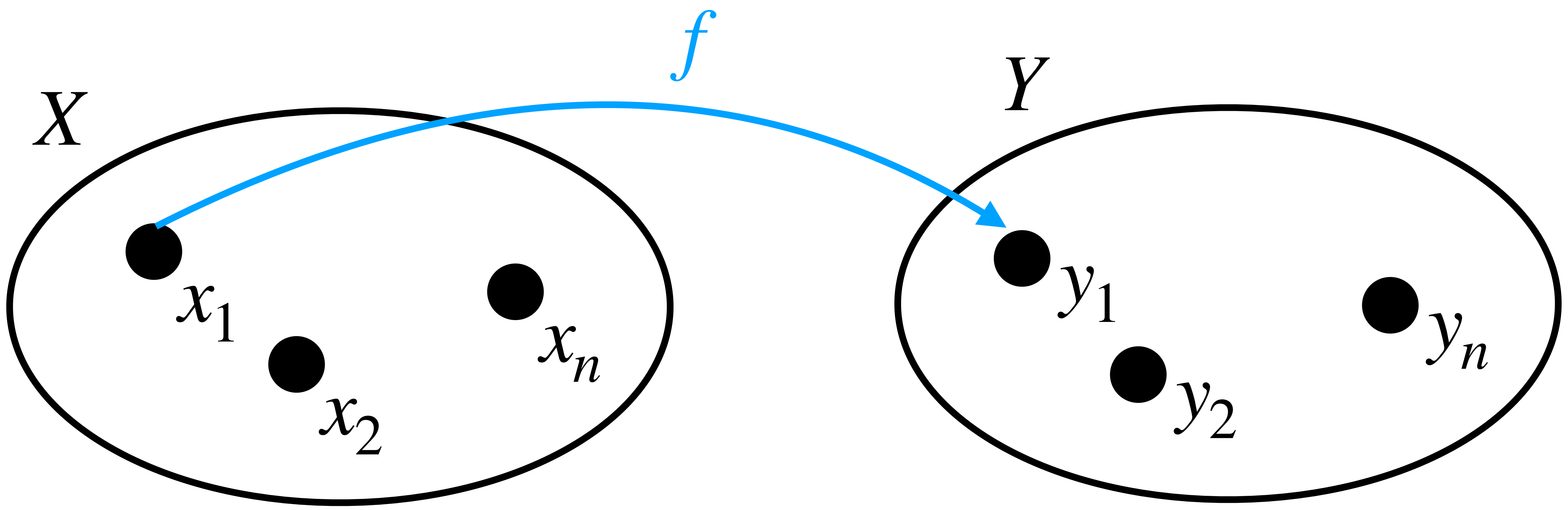


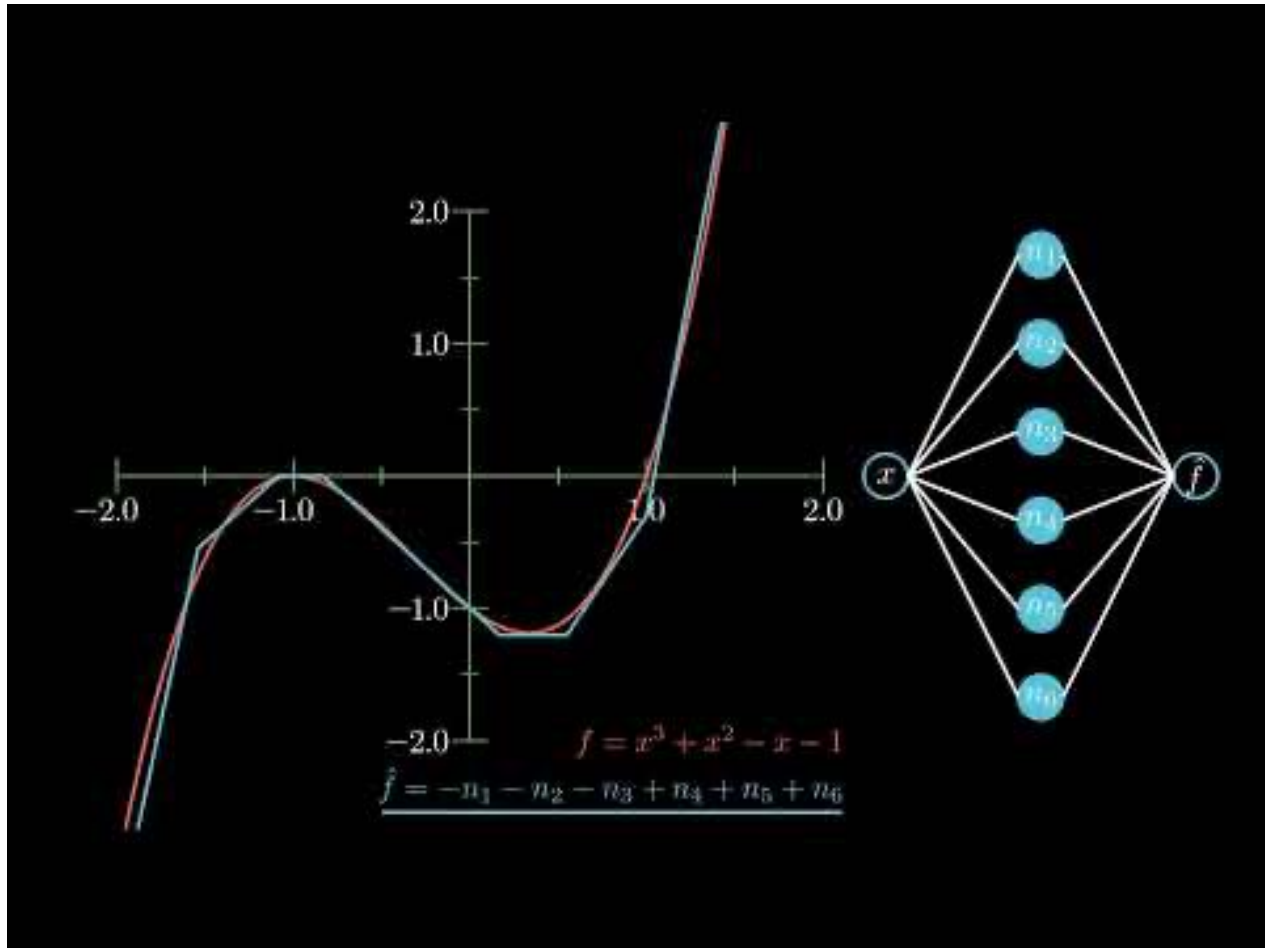
Mnih et al,. (Nature, 2015)

Universal Approximation Theorem

Cybenko (1989)

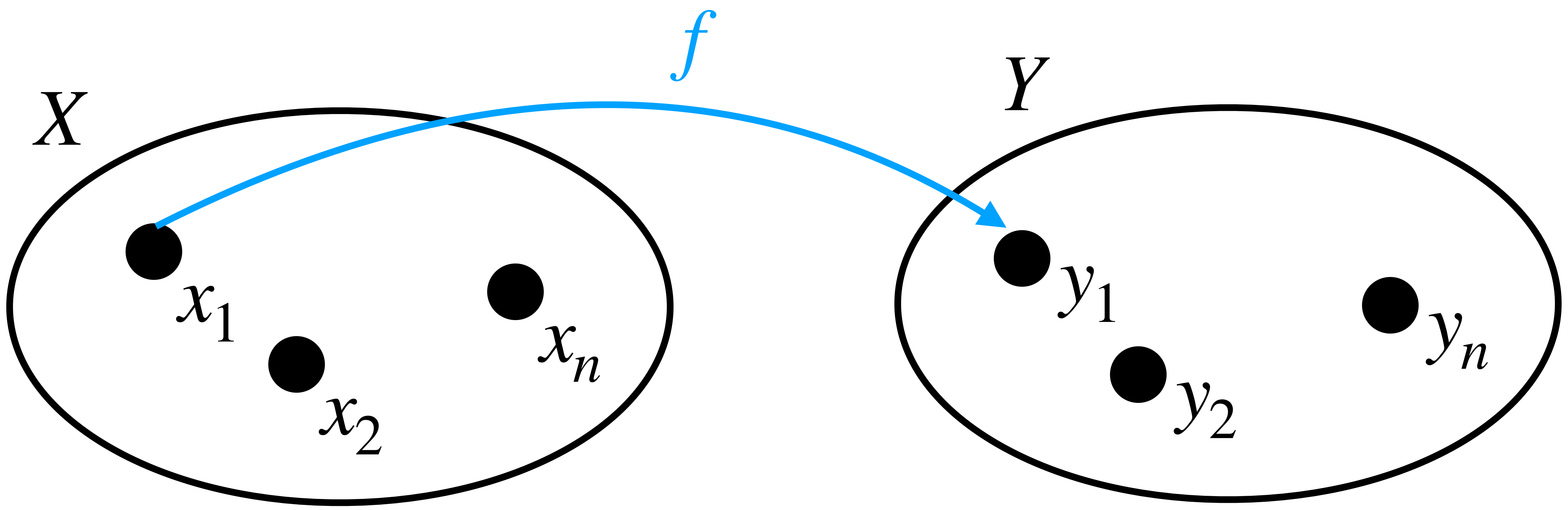
- What is a function?
    - $y = f(x)$
    - $f : X \to Y$
- ANNs are also functions
    - $g_{\mathbf{w}}(x) = \sigma(x)$ where $\mathbf{w}$ are the connection weights
- *At least one neural network exists that can approximate any continuous function with arbitrary precision*
    - $|g_{\mathbf{w}}(x) - f(x)| < \epsilon$

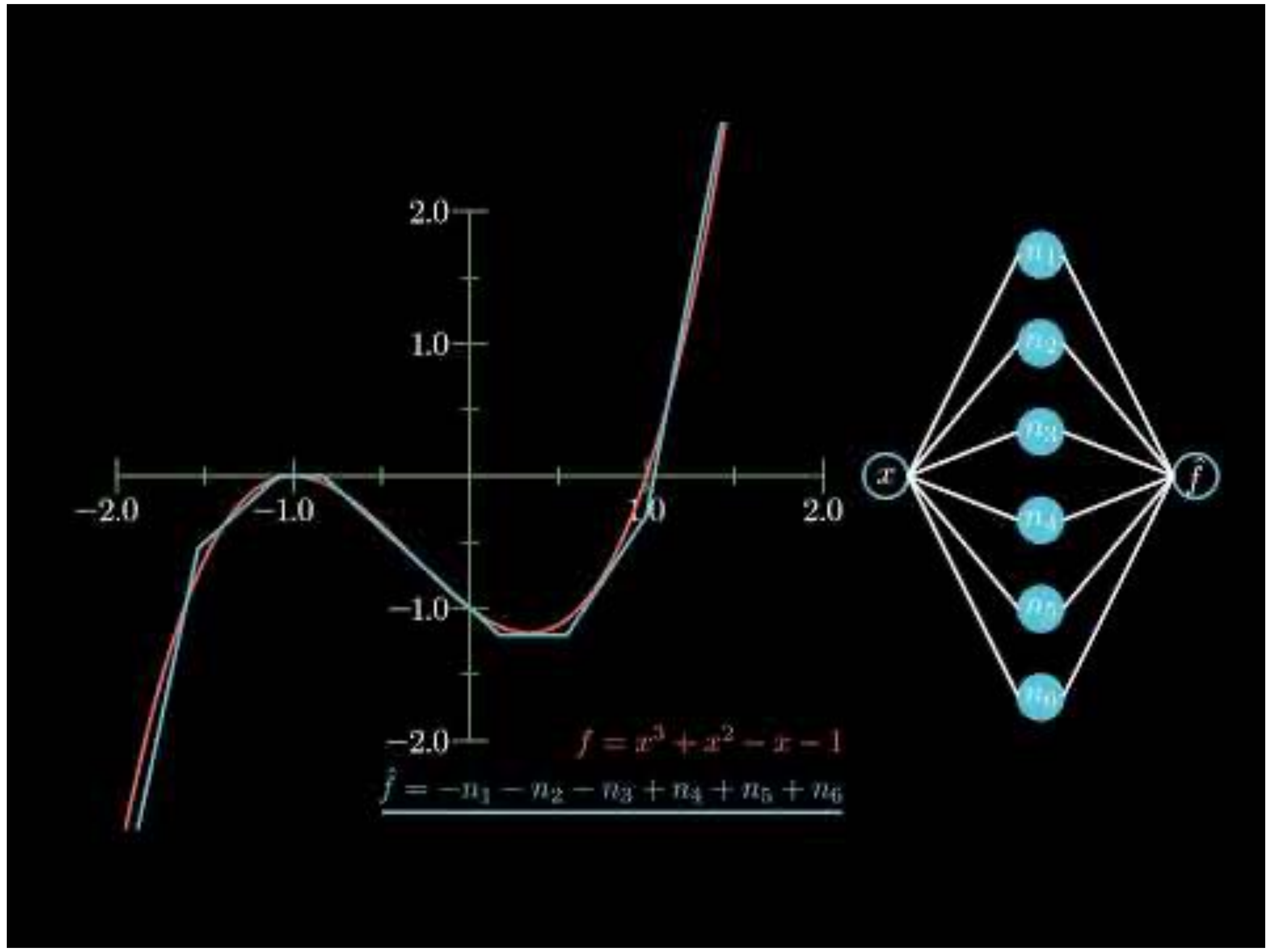




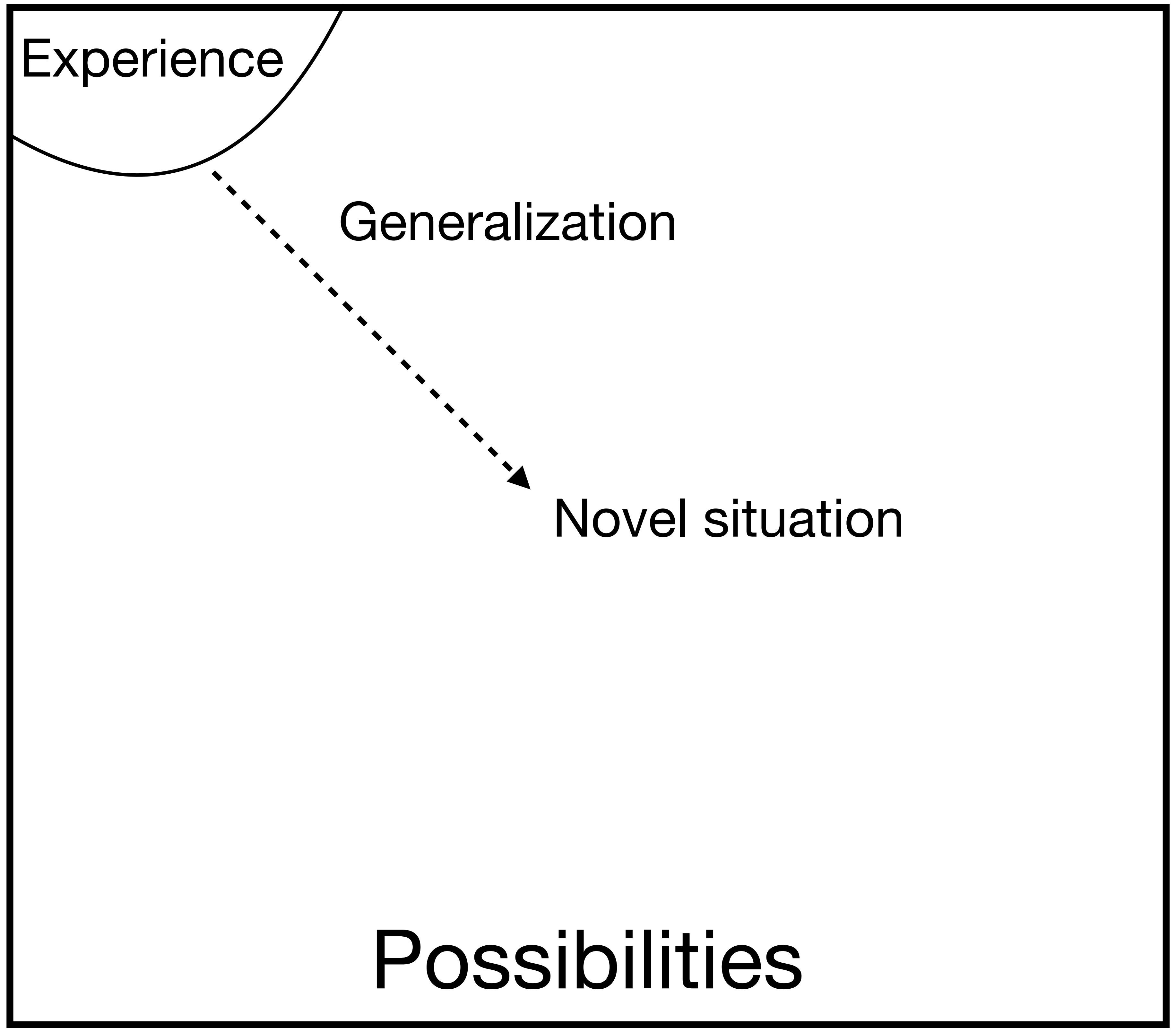
Universal Approximation Theorem Cybenko (1989)

- What is a function?
  - $y = f(x)$
  - $f : X \to Y$
- ANNs are also functions
  - $g_{\mathbf{w}}(x) = \sigma(x)$ where $\mathbf{w}$ are the connection weights
- *At least one neural network exists that can approximate any continuous function with arbitrary precision*
  - $|g_{\mathbf{w}}(x) - f(x)| < \epsilon$

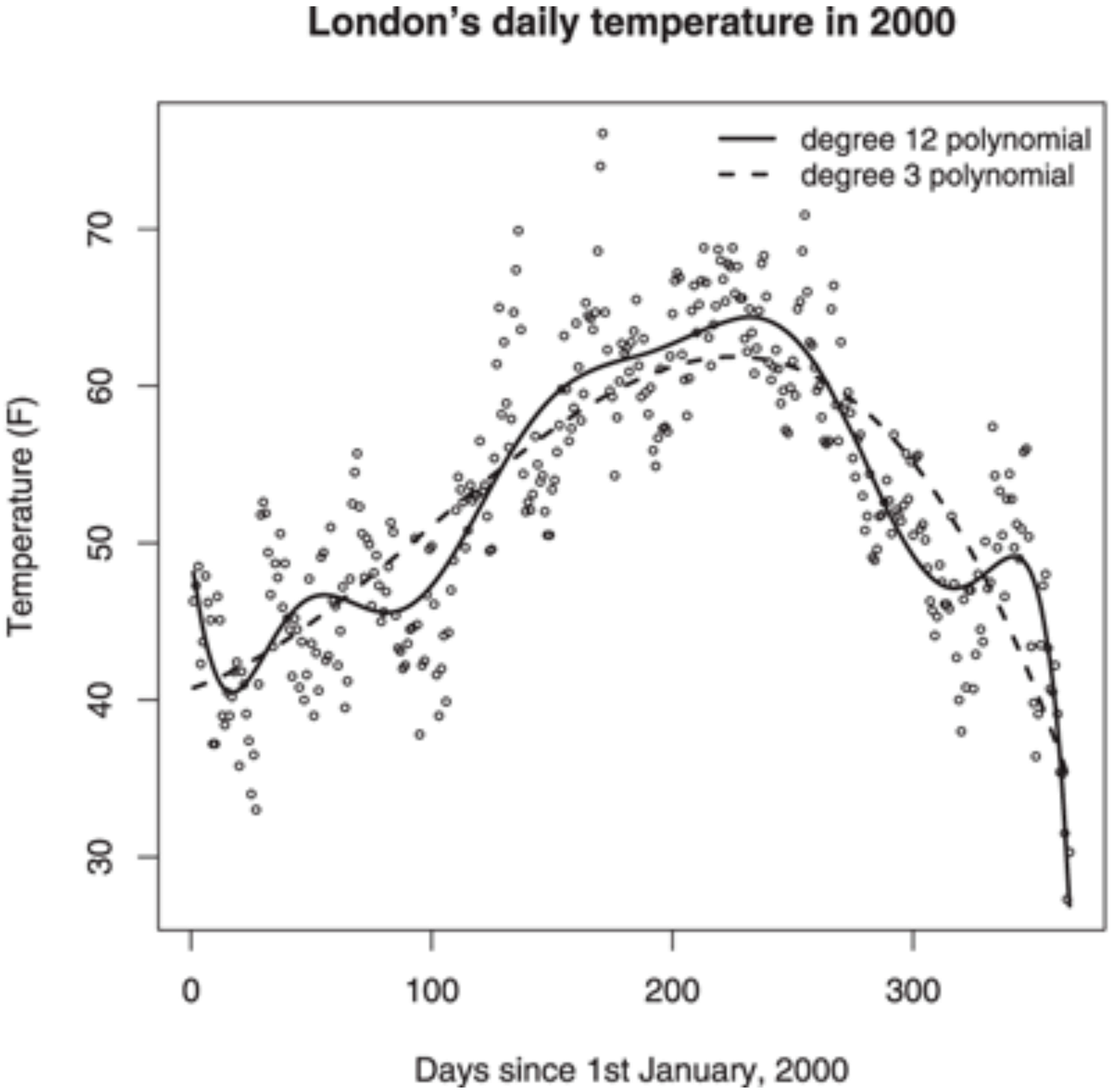




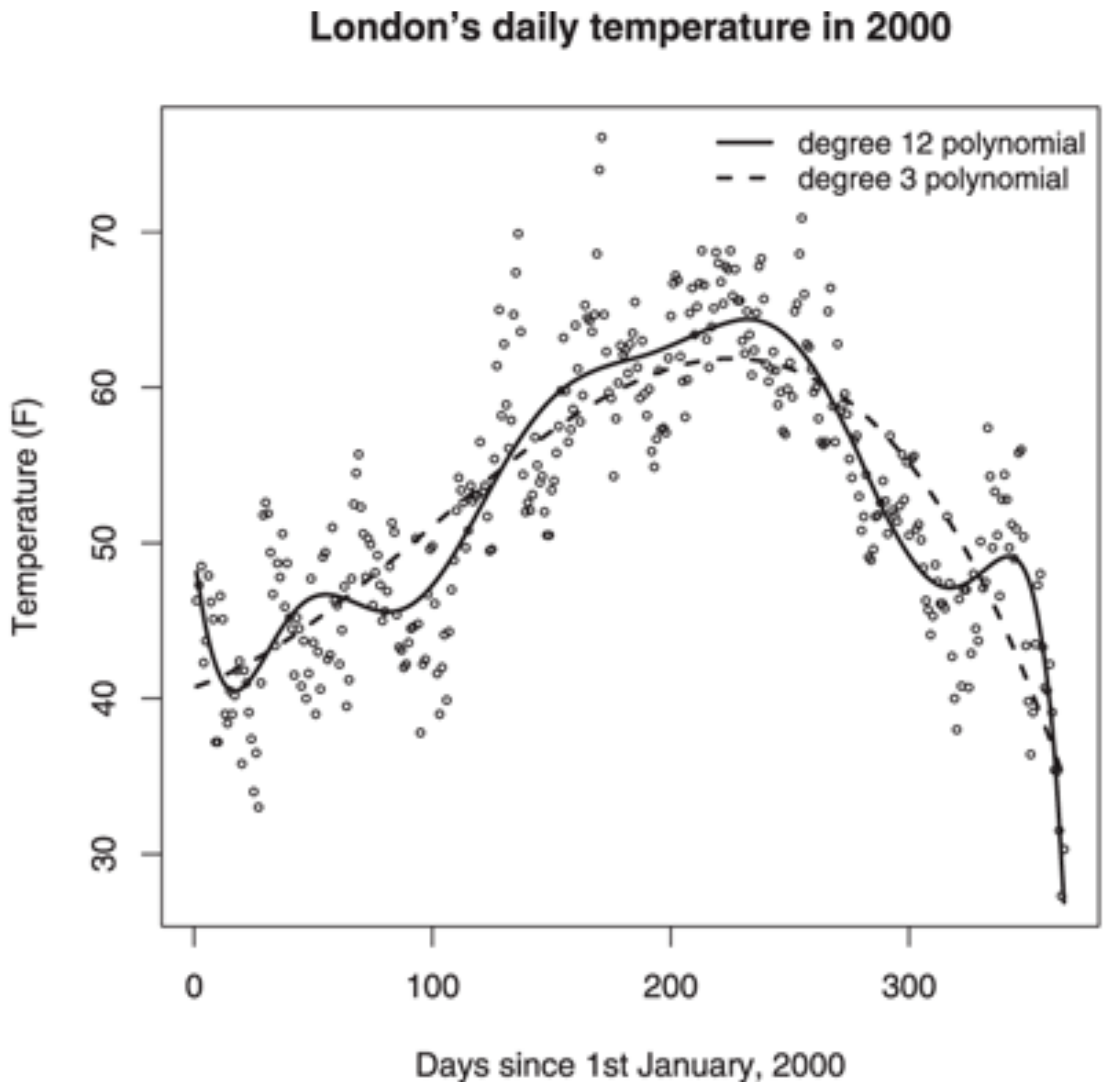
Caveat: Approximation does not guarantee generalization

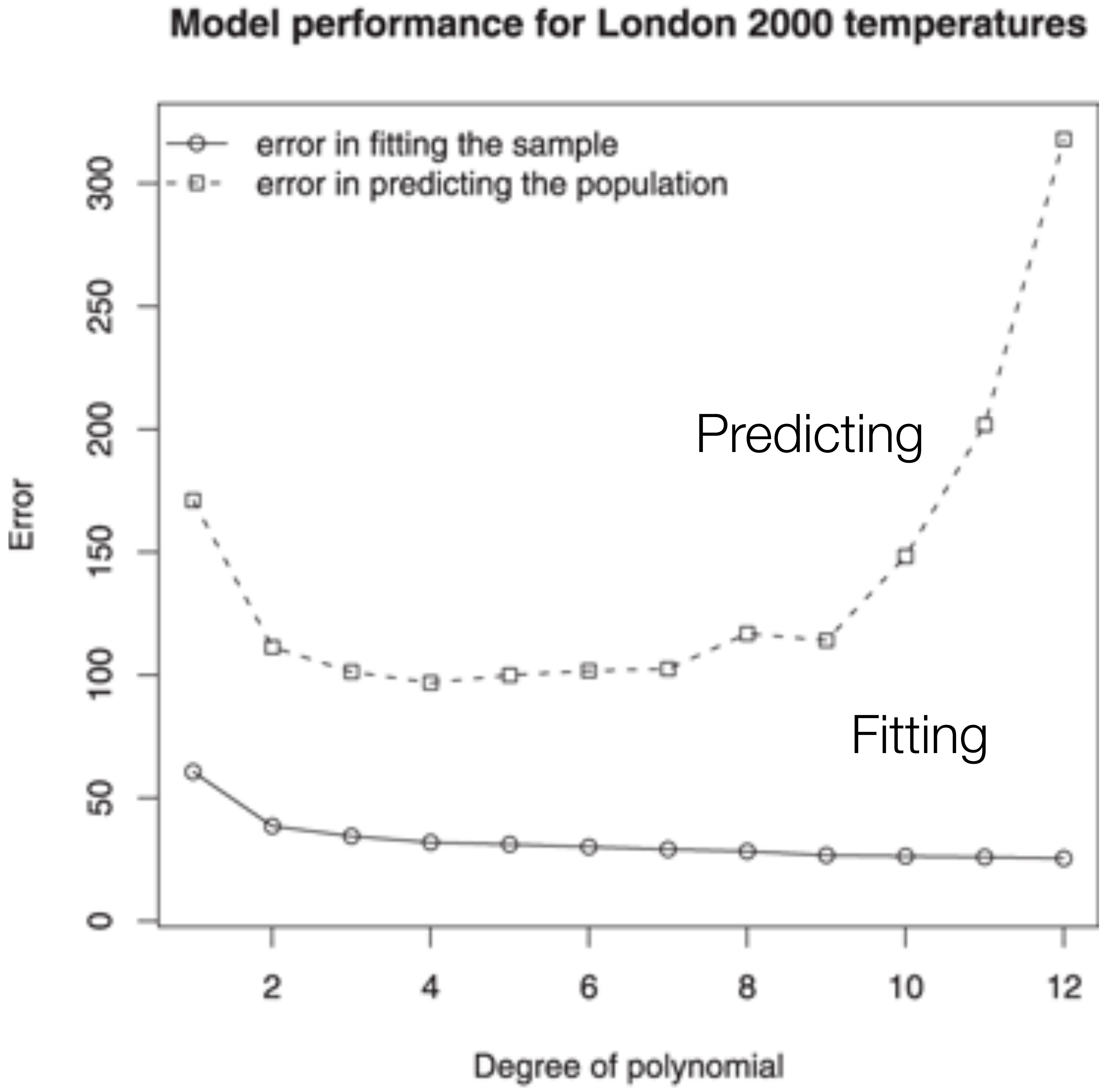


Caveat: Approximation does not guarantee generalization



Caveat: Approximation does not guarantee generalization

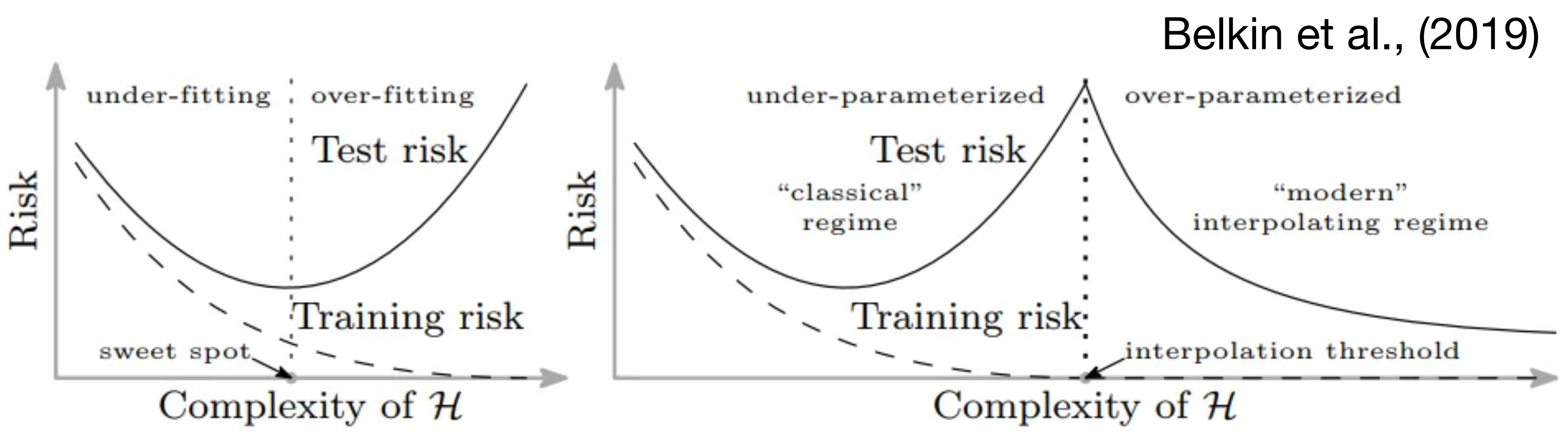




Yet why do large ANNs work so well?

Not on the exam but very cool

- Double descent phenomena
  - left: standard story
  - right: over-parameterized models start to reduce prediction error



(a) U-shaped "bias-variance" risk curve

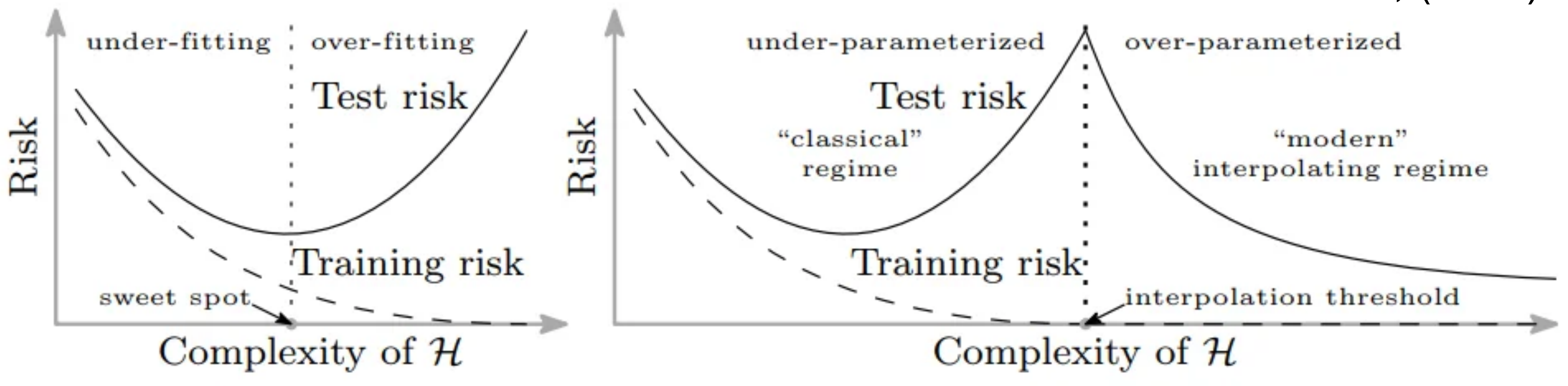
(b) "double descent" risk curve

Yet why do large ANNs work so well?

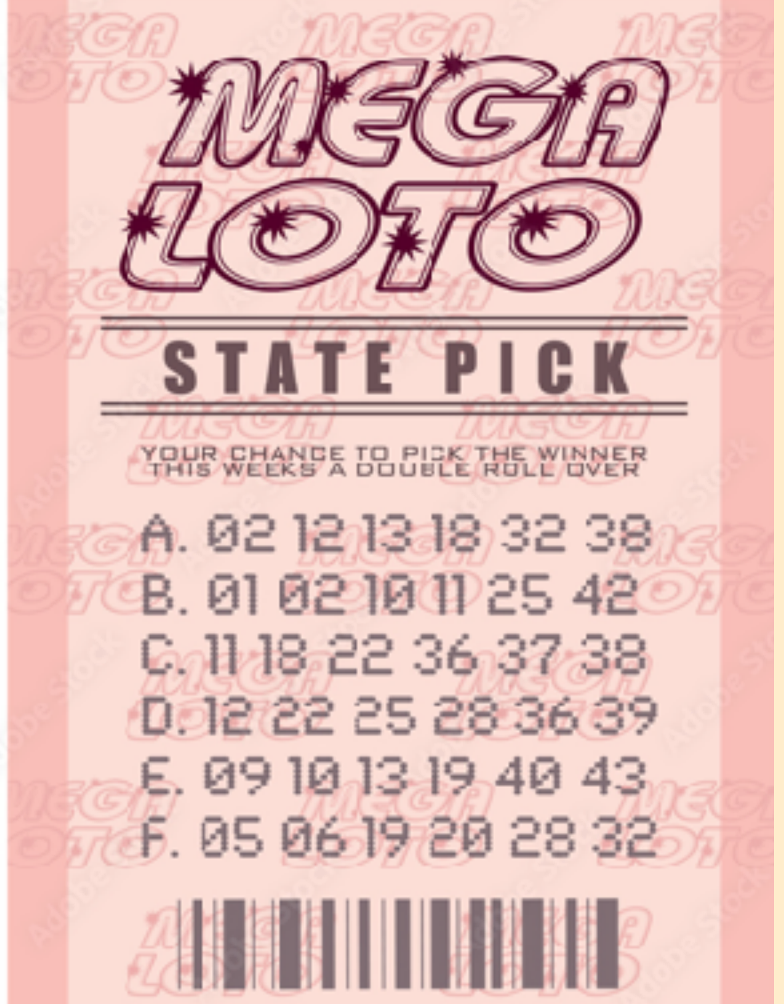
Not on the exam but very cool

- Double descent phenomena
  - left: standard story
  - right: over-parameterized models start to reduce prediction error
- Lottery Ticket conjecture
  - *if you buy enough lottery tickets, one is bound to be a winner**
  - Large “over-parameterized” ANNs have a bunch of different **subnetworks** that are randomly initialized (i.e., lottery tickets)
  - SGD focuses on training winning subnetworks
  - Pruning connections not part of the winning ticket can improve efficiency and even performance
  - The *effective* complexity $\neq |\theta|$

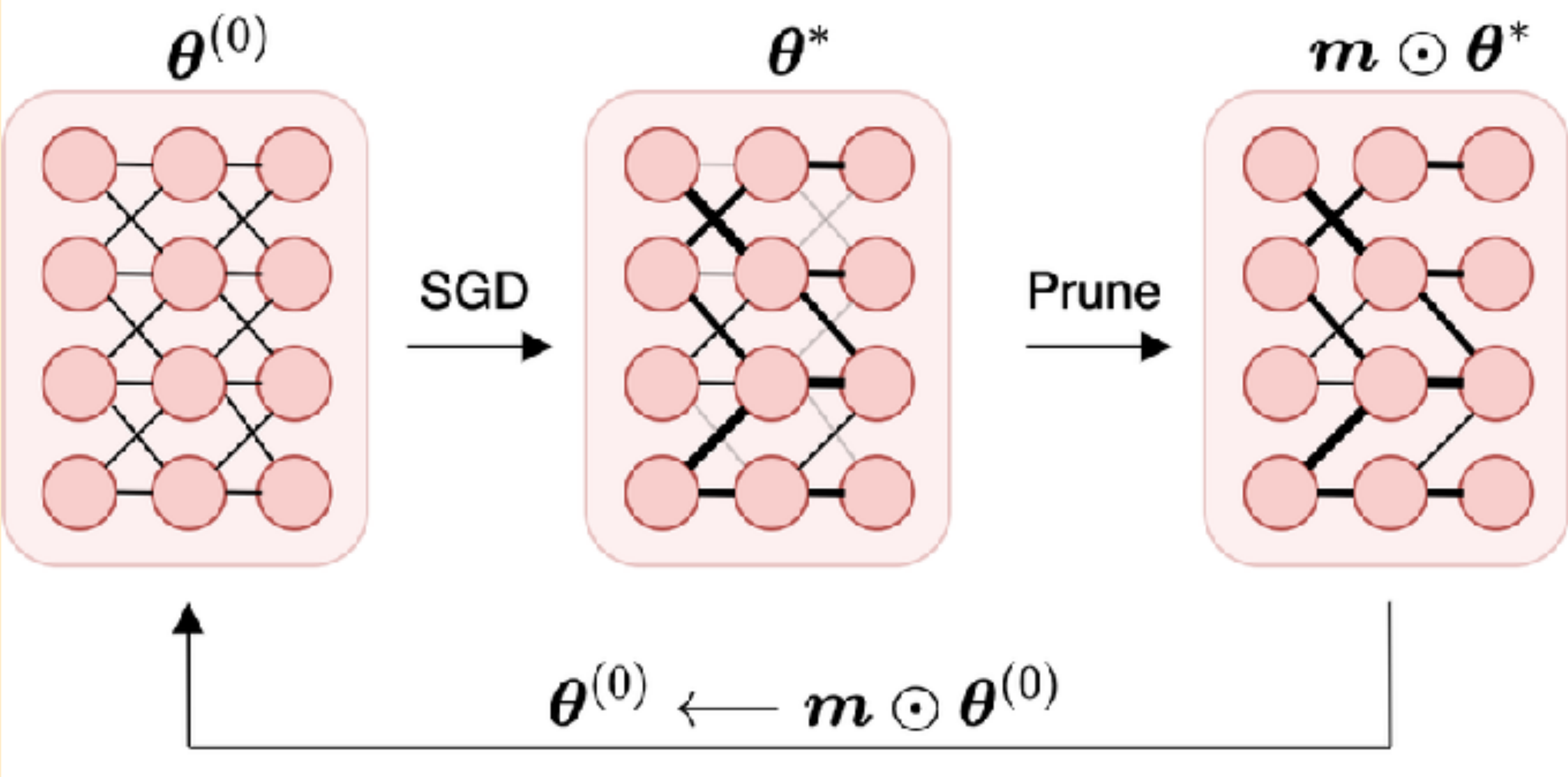
Belkin et al., (2019)



(a) U-shaped “bias-variance” risk curve
(b) “double descent” risk curve



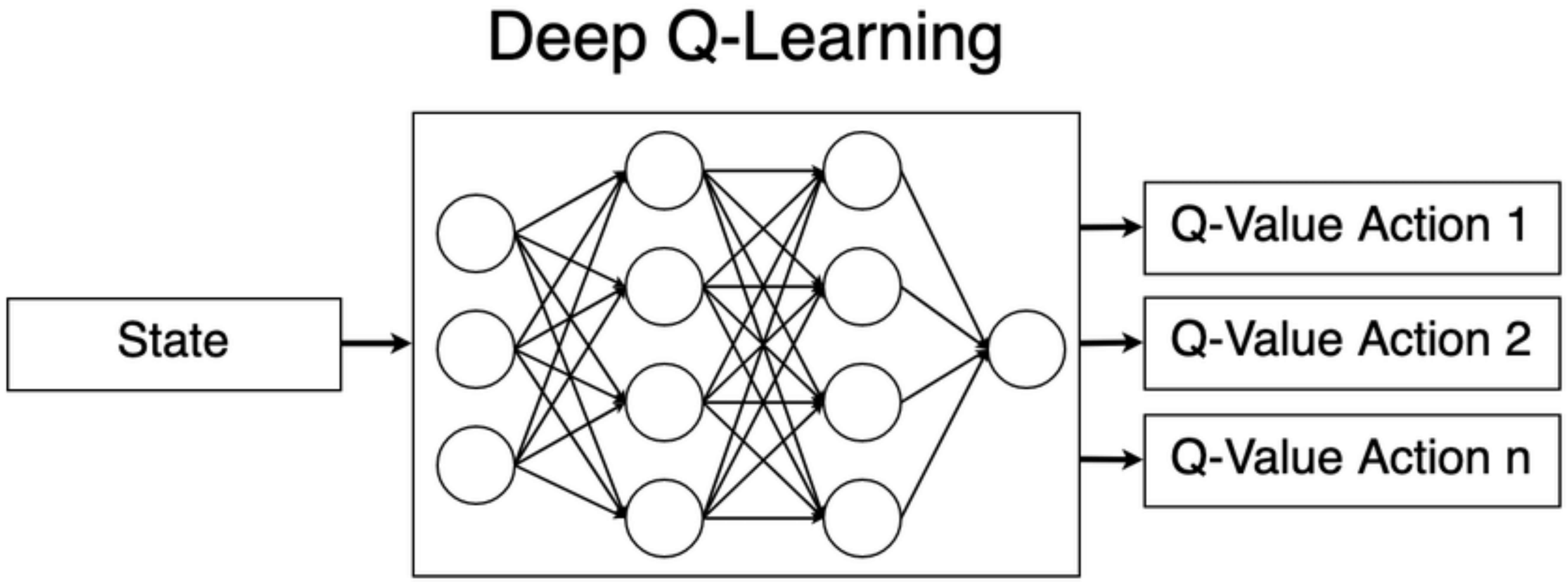
Frankle & Carbin (2019)



*not actually good financial advice

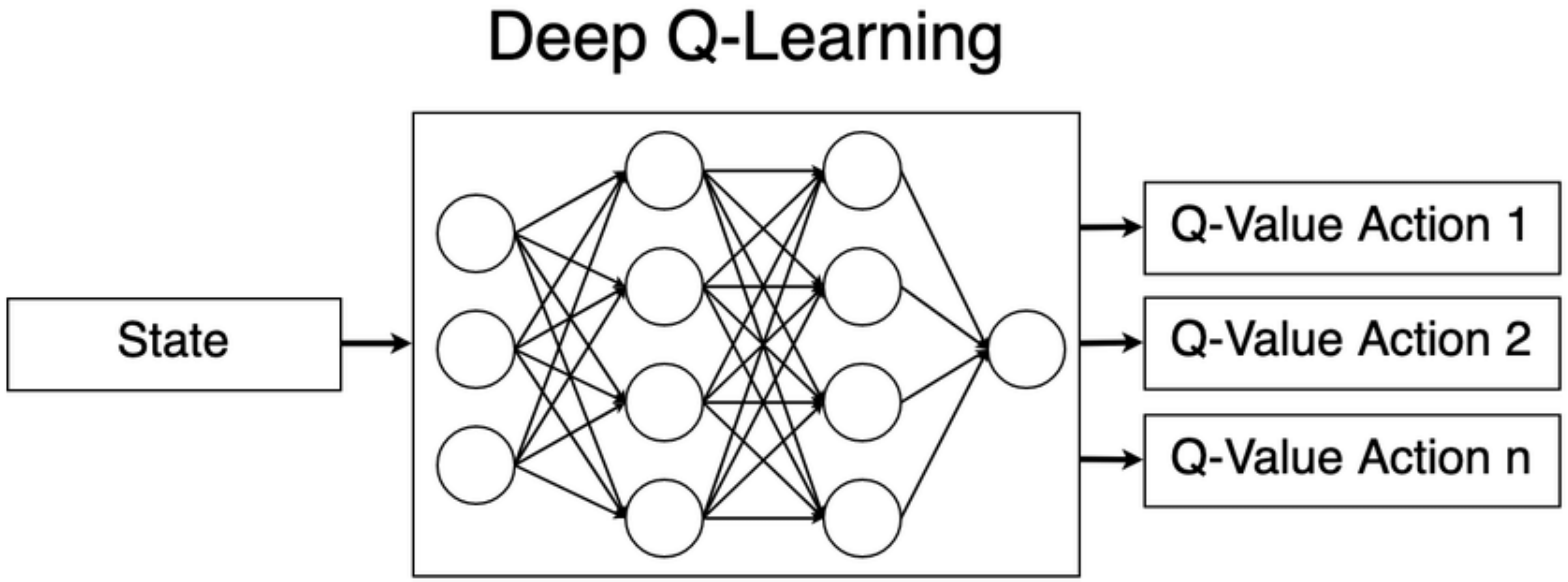
Policy Gradient

- **Deep Q-learning** uses an ANN to approximate the value function
  - the policy is implicit (e.g., a softmax over Q-values)



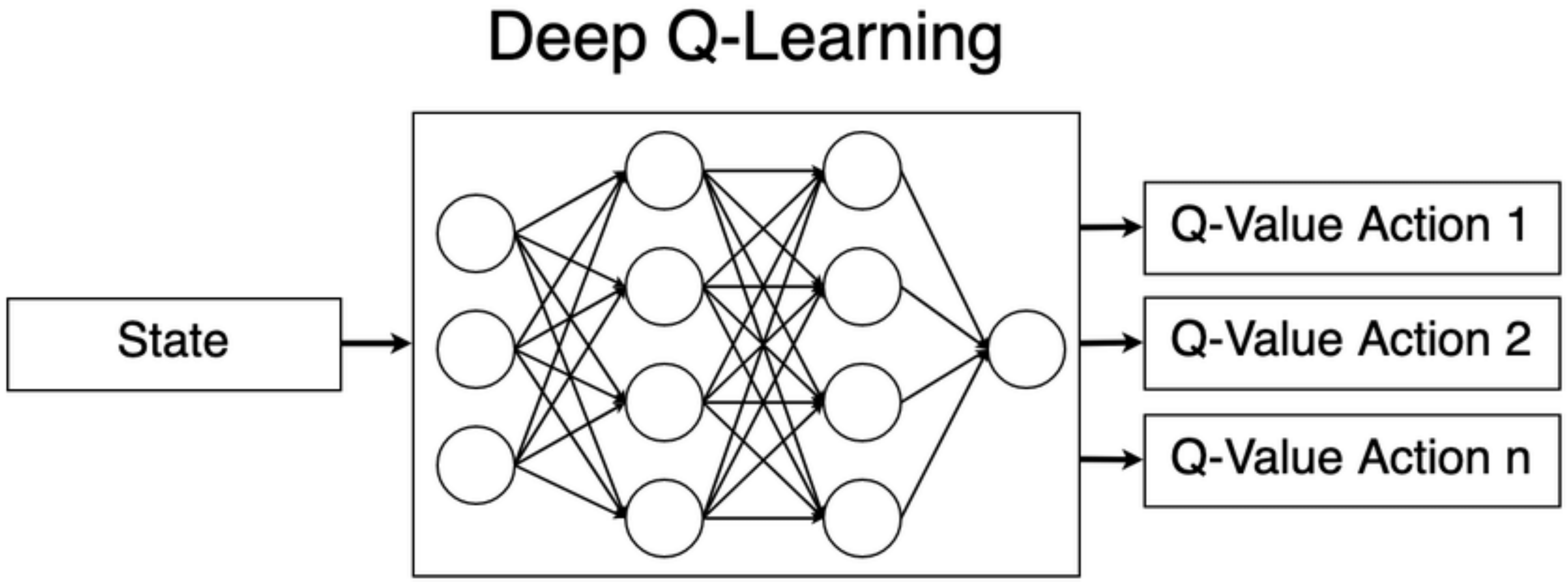
Policy Gradient

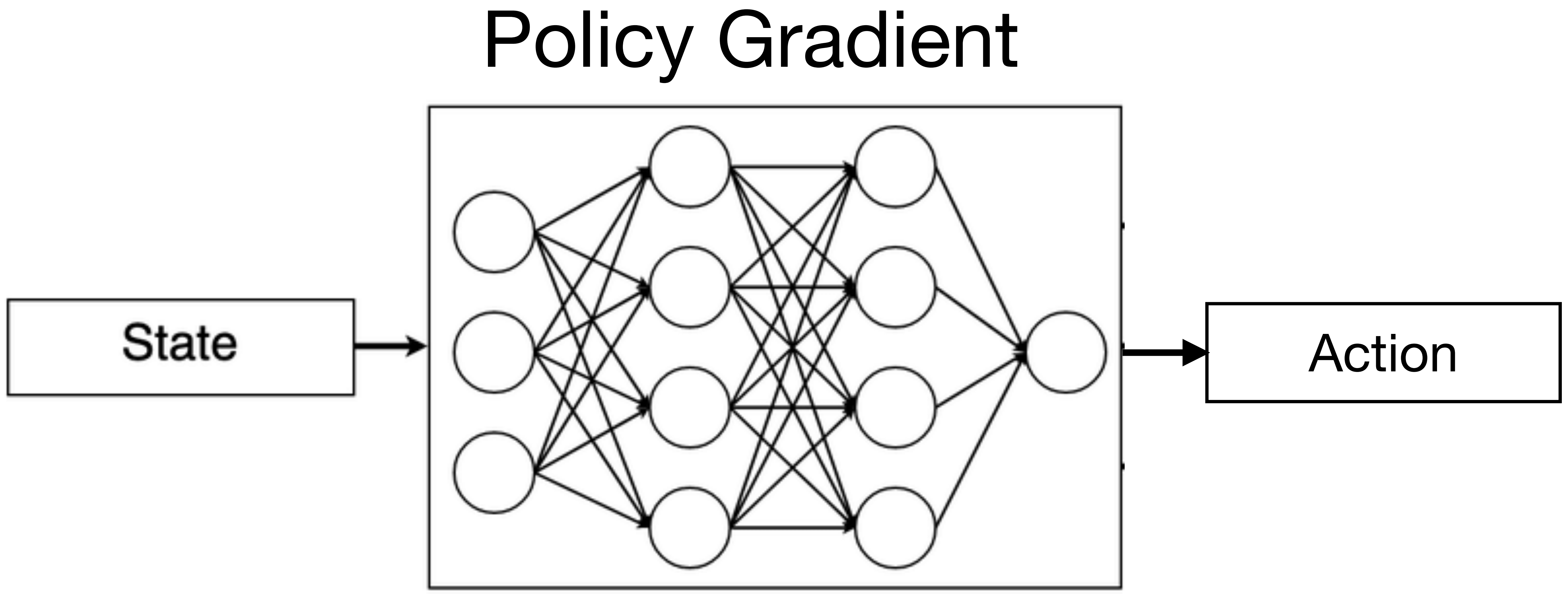
- **Deep Q-learning** uses an ANN to approximate the value function
  - the policy is implicit (e.g., a softmax over Q-values)
- **Policy gradient** uses a function to approximate the optimal policy
  - the value function is implicit



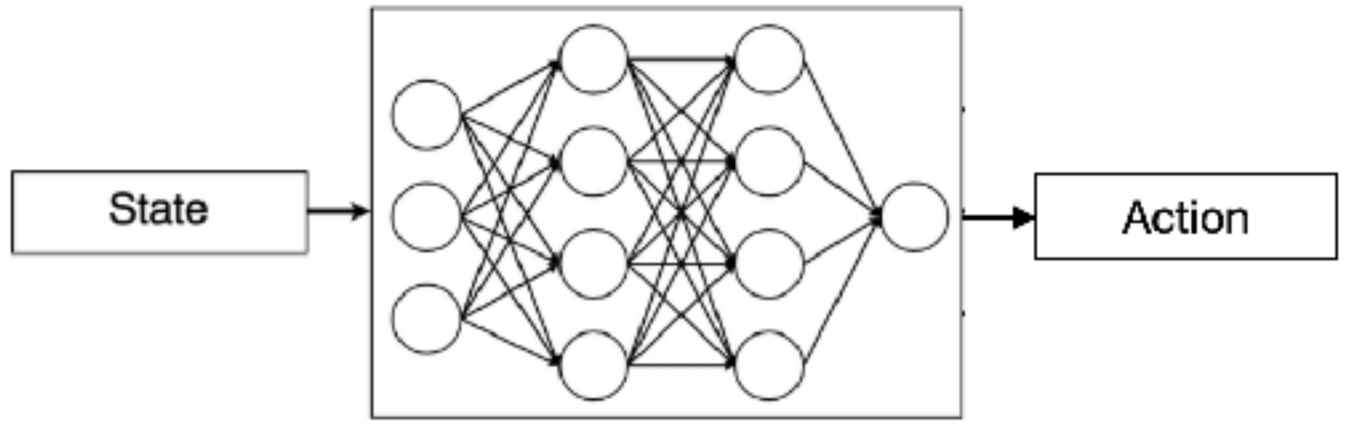
Policy Gradient

- **Deep Q-learning** uses an ANN to approximate the value function
  - the policy is implicit (e.g., a softmax over Q-values)
- **Policy gradient** uses a function to approximate the optimal policy
  - the value function is implicit





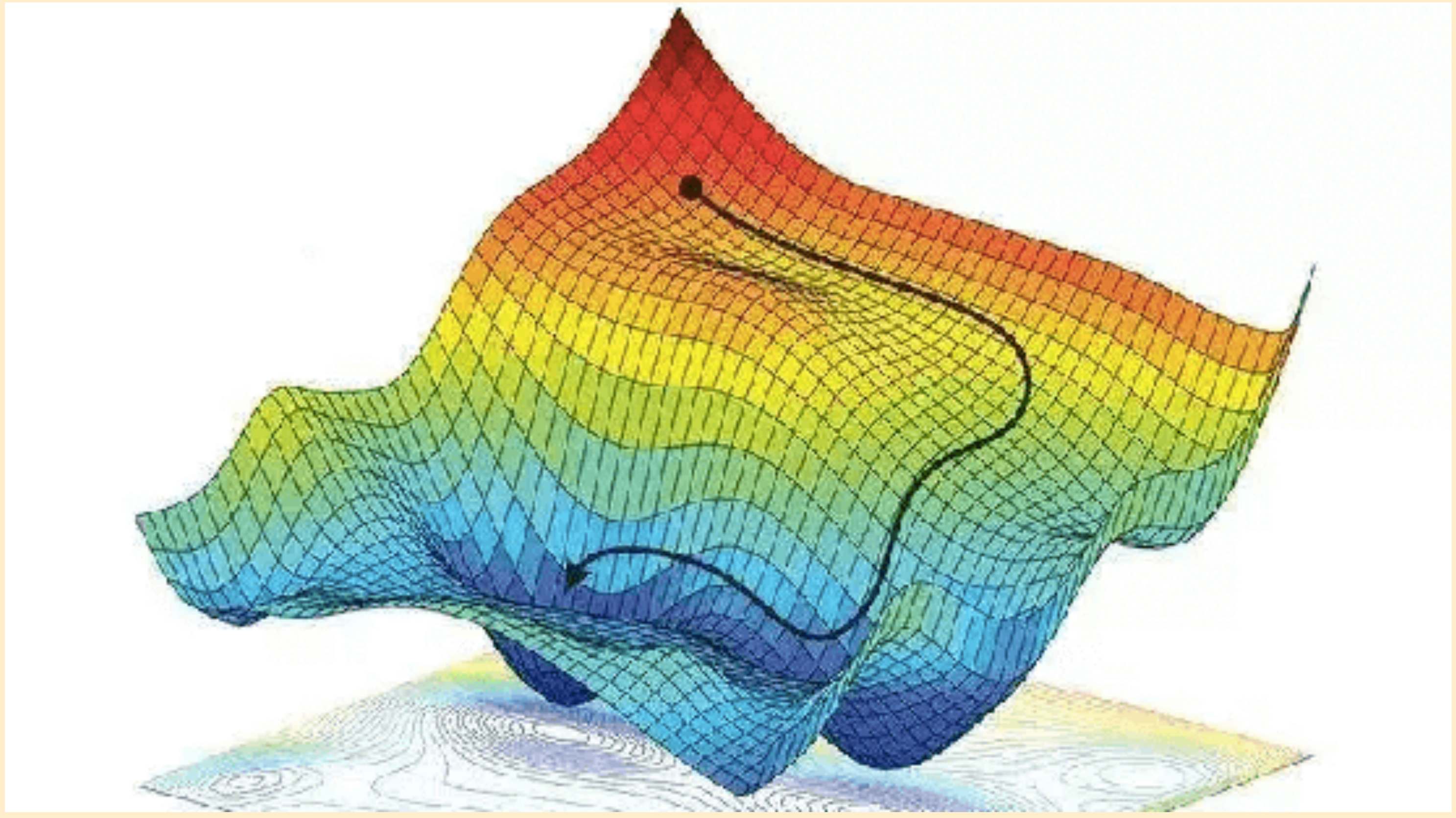
Policy Gradient



Formulas not on exam, but you should understand the general concept!

- Use a neural network to parameterize a policy $\pi_\theta(a \mid s)$
- Objective: Maximize expected reward following a parameterized policy: $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)]$
- Method: using gradient ascent $\theta_{t+1} = \theta_t + \beta \nabla_\theta J(\theta_t)$

  learning rate



- Using the Markov principle, we can write the gradient as:

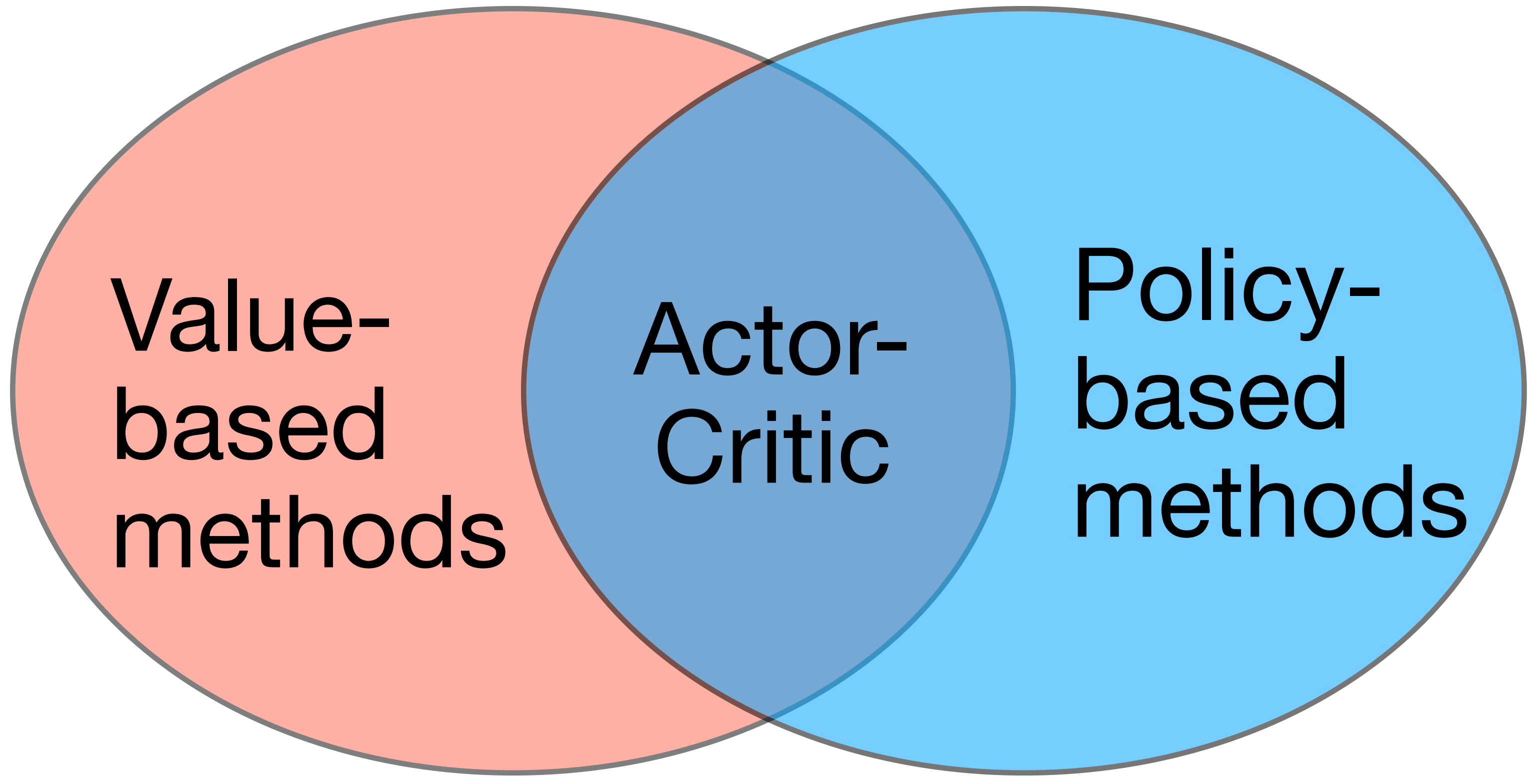
$$\nabla J(\theta_t) = \mathbb{E}_{\pi_\theta}\left[\sum_{t \in \tau}^{|\tau|} \nabla \log \pi_\theta(a_t \mid s_t) Q(a_t, s_t)\right]$$

- Updates to θ follow the **gradient** to increase the probabability of highly rewarding actions:

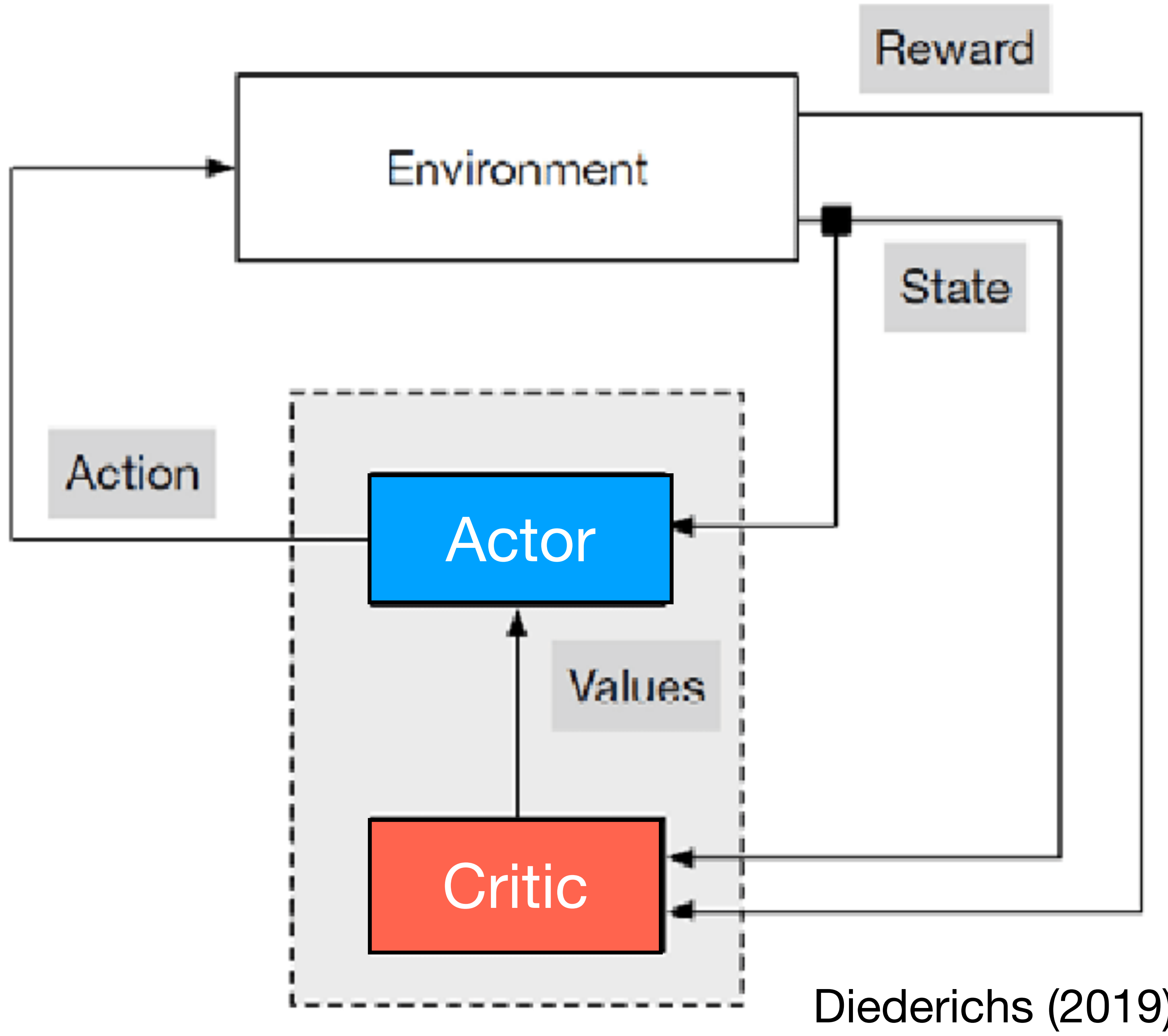
  $\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q(a_t, s_t)$

- Here, $Q(a_t, s_t)$ is usually estimated through Monte Carlo sampling

Actor-Critic

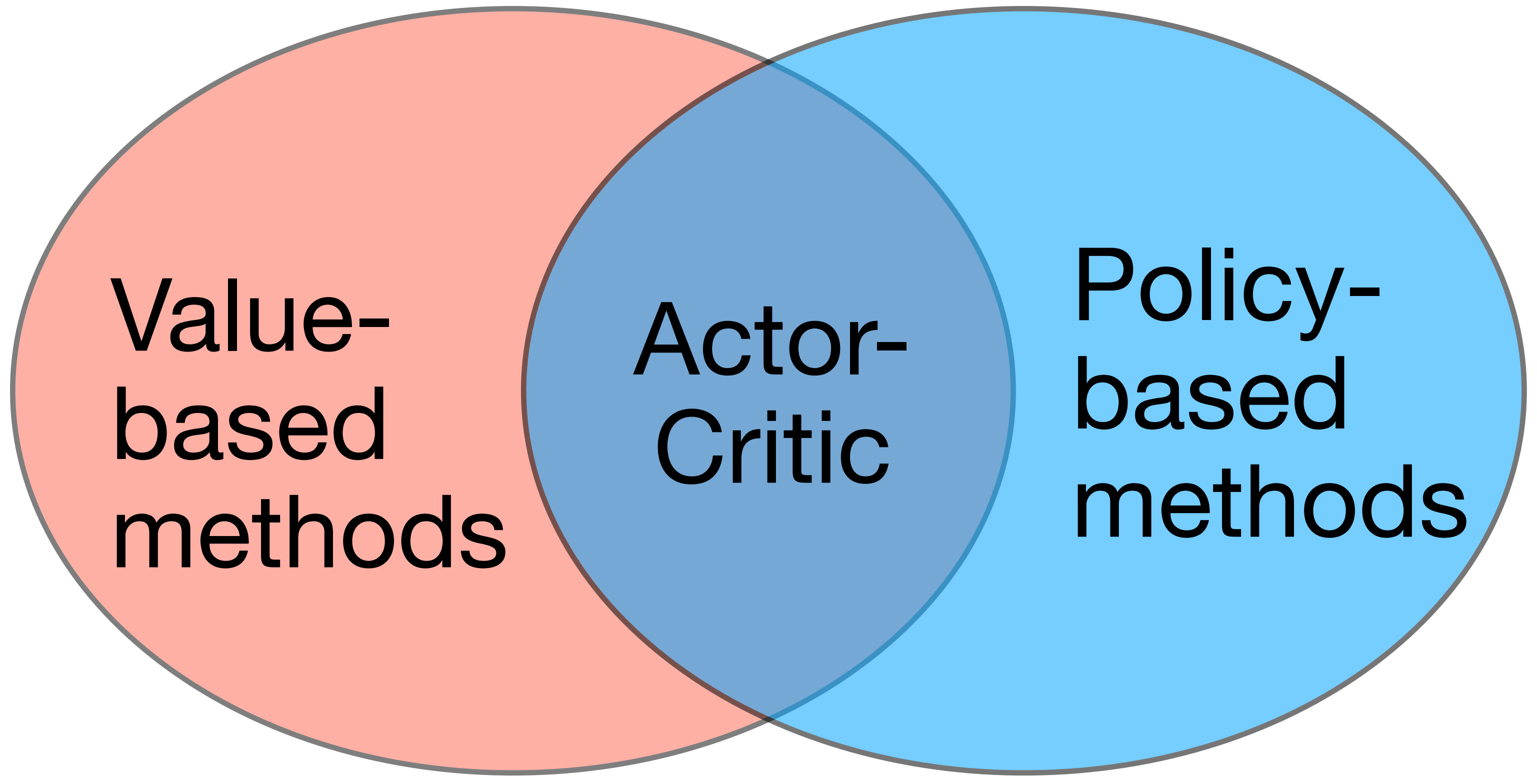


- Actor-critic combines **value-based** and **policy-based** methods and is a generalization of policy iteration
  - **Actor** provides the policy $\pi_\theta(a \mid s)$ parameterized by θ
  - **Critic** provides the value function $Q_{\mathbf{w}}(s, a)$ parameterized by $\mathbf{w}$

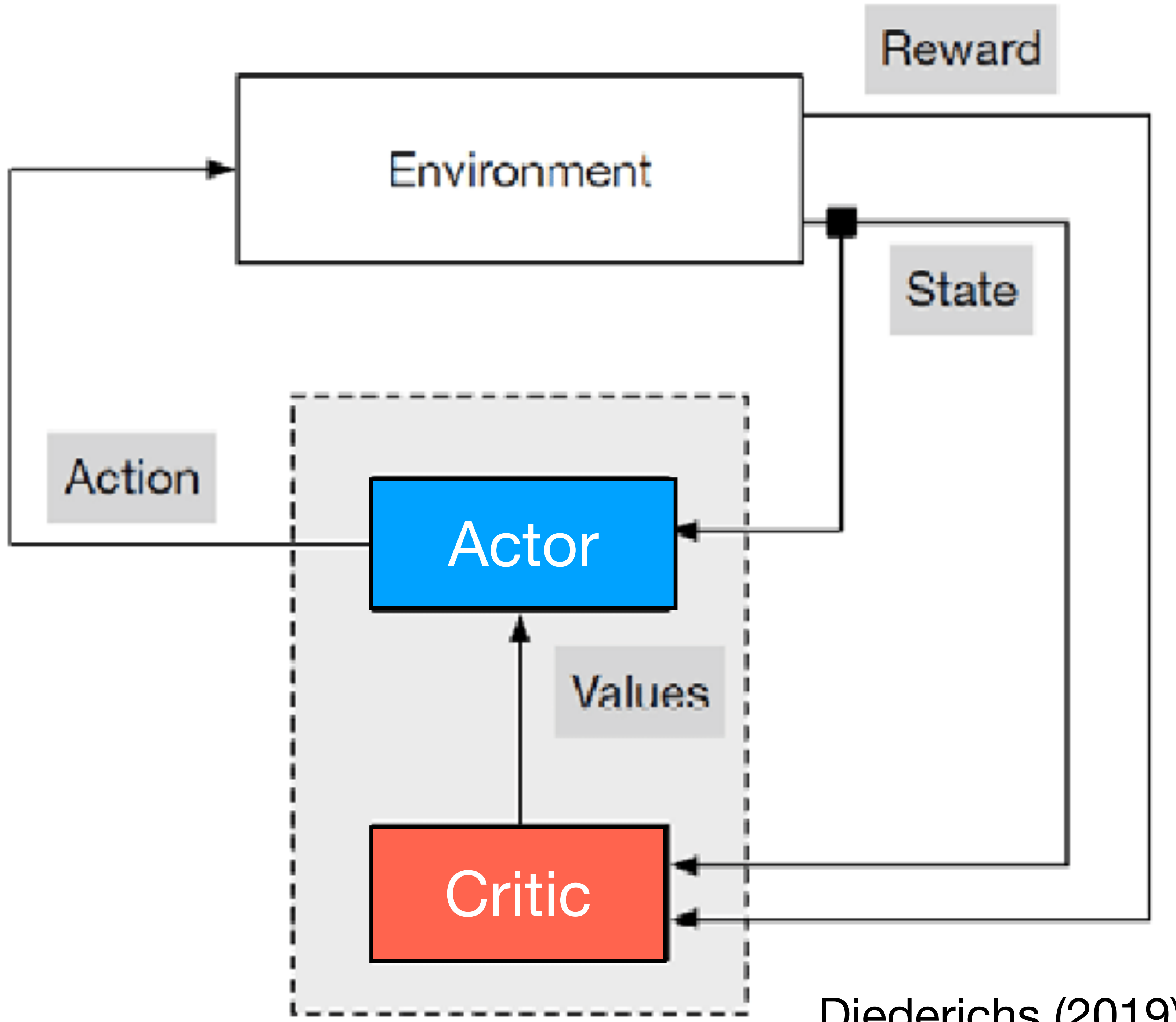


Diederichs (2019)

Actor-Critic

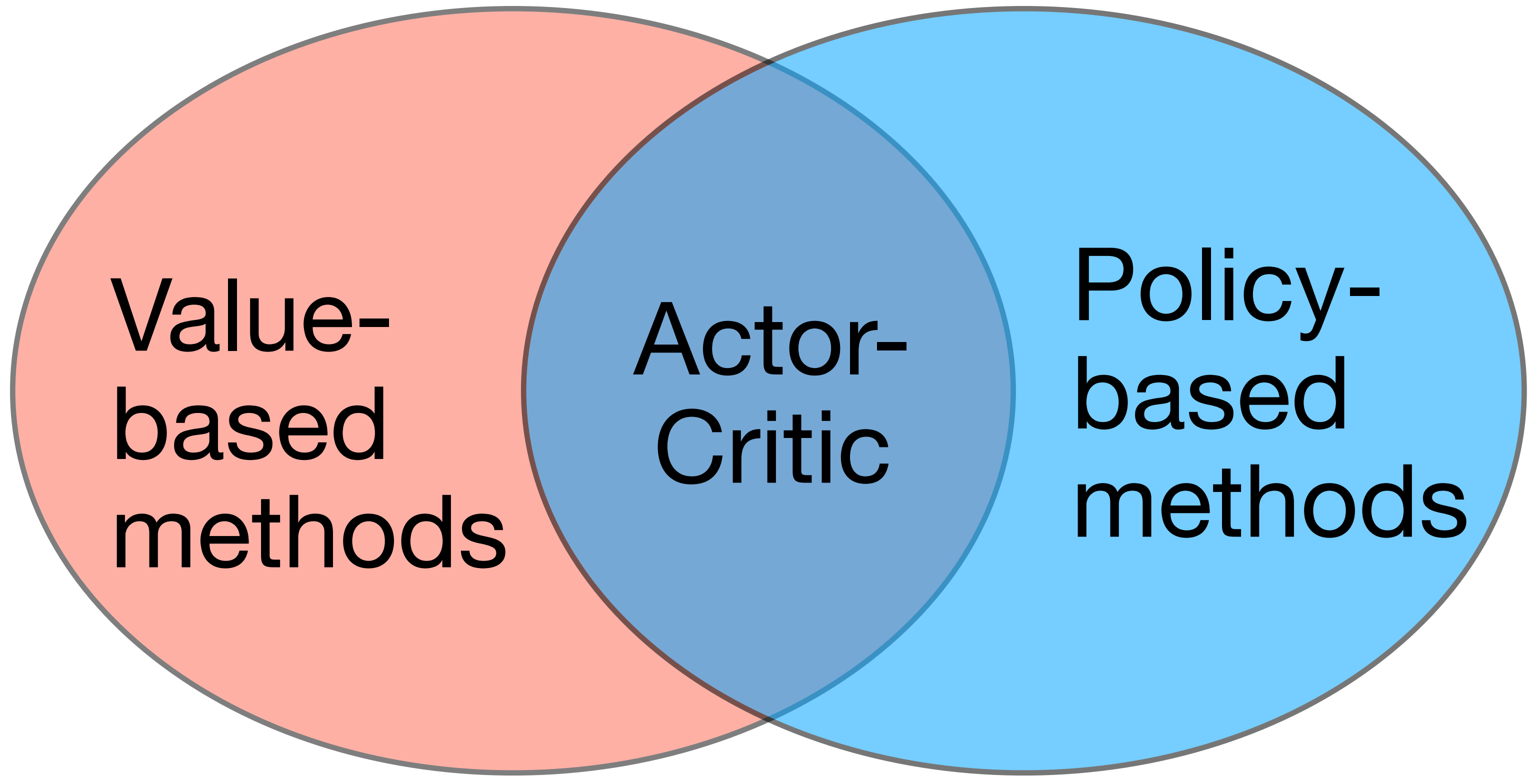


- Actor-critic combines **value-based** and **policy-based** methods and is a generalization of policy iteration
  - **Actor** provides the policy $\pi_\theta(a \mid s)$ parameterized by θ
  - **Critic** provides the value function $Q_{\mathbf{w}}(s, a)$ parameterized by $\mathbf{w}$
- Simulate trajectories $a \sim \pi_\theta(a \mid s)$ and compute TD error $\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$

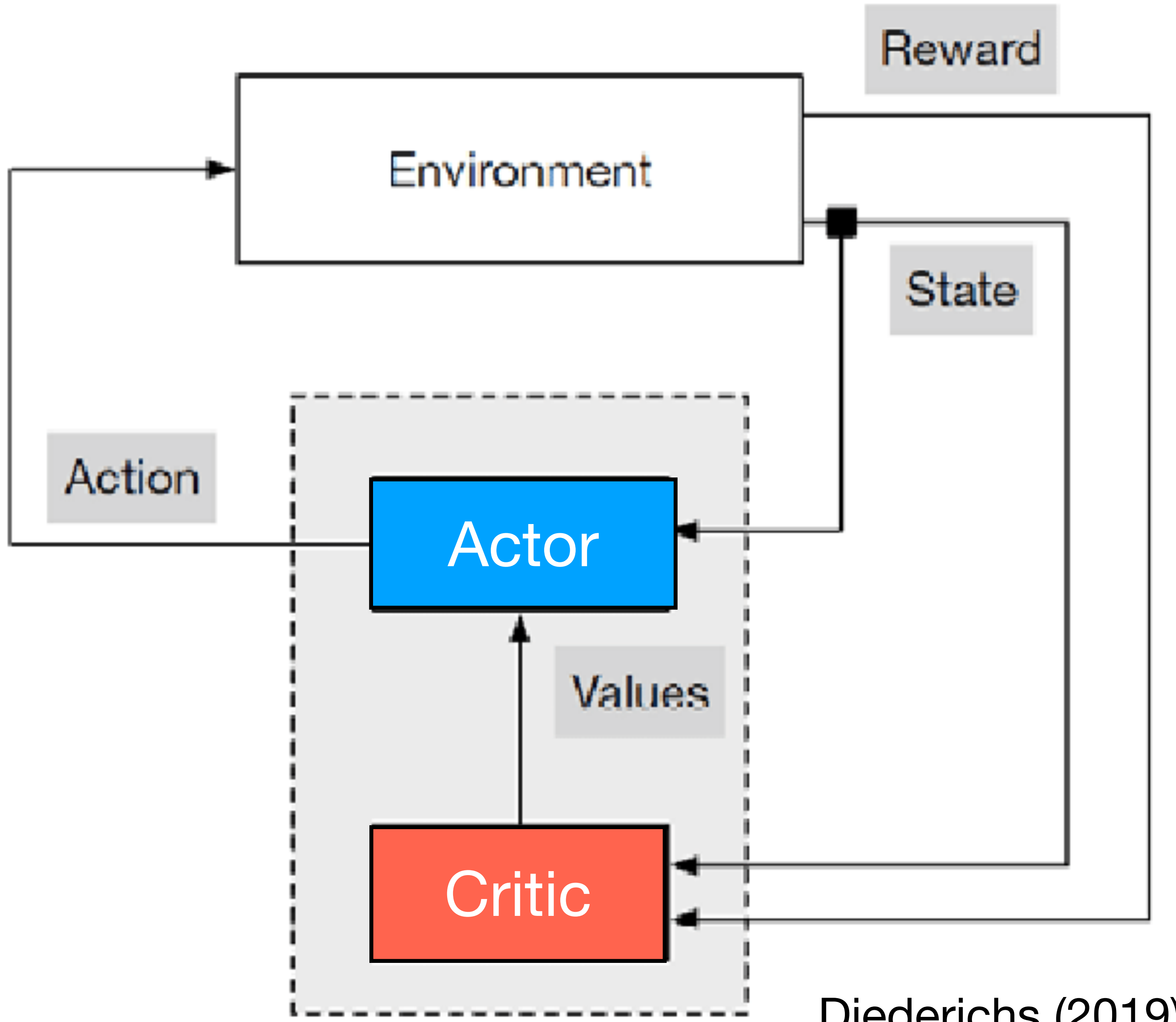


Diederichs (2019)

Actor-Critic

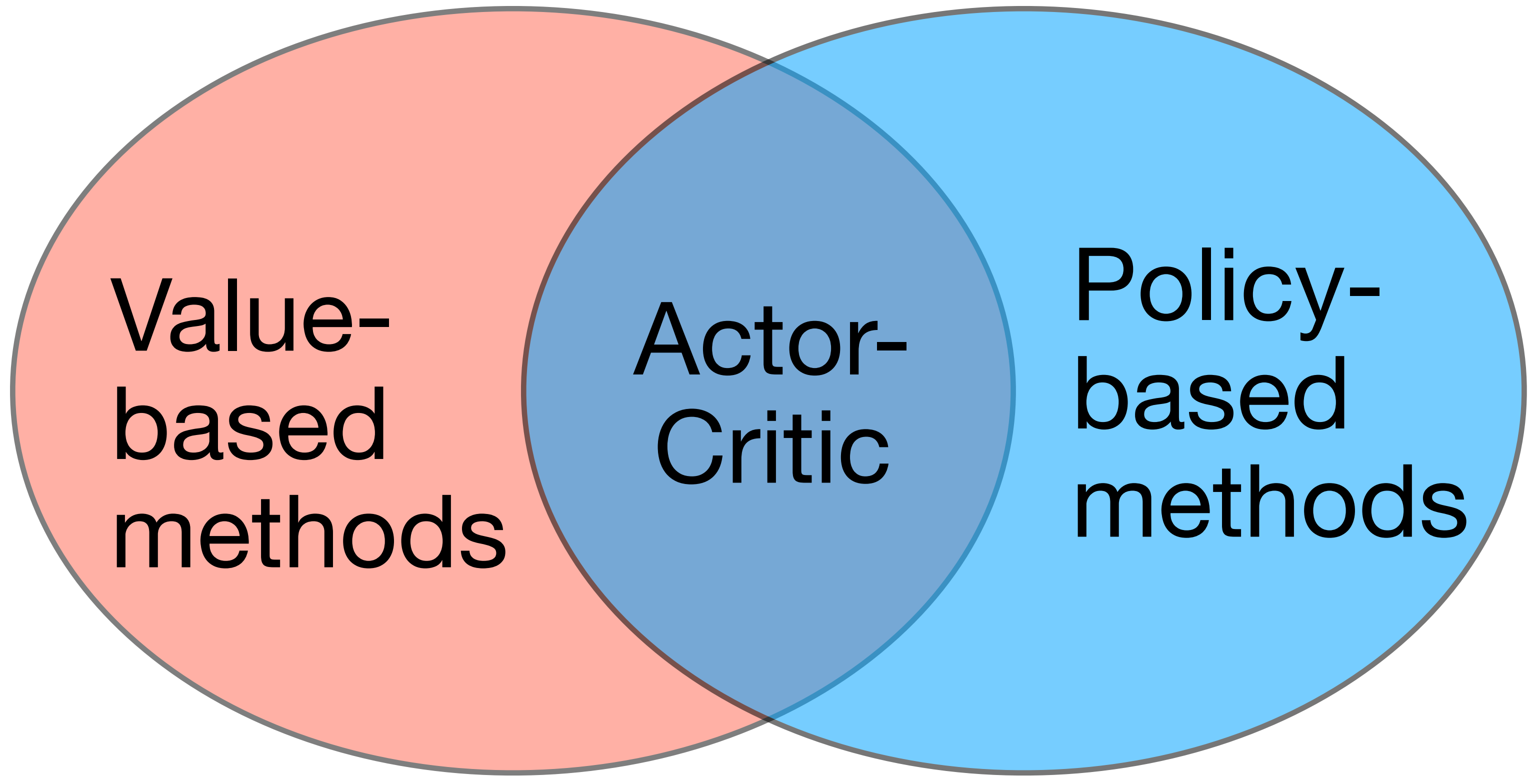


- Actor-critic combines **value-based** and **policy-based** methods and is a generalization of policy iteration
  - **Actor** provides the policy $\pi_\theta(a \mid s)$ parameterized by θ
  - **Critic** provides the value function $Q_{\mathbf{w}}(s, a)$ parameterized by $\mathbf{w}$
- Simulate trajectories $a \sim \pi_\theta(a \mid s)$ and compute TD error $\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$
- Iteratively update actor and critic

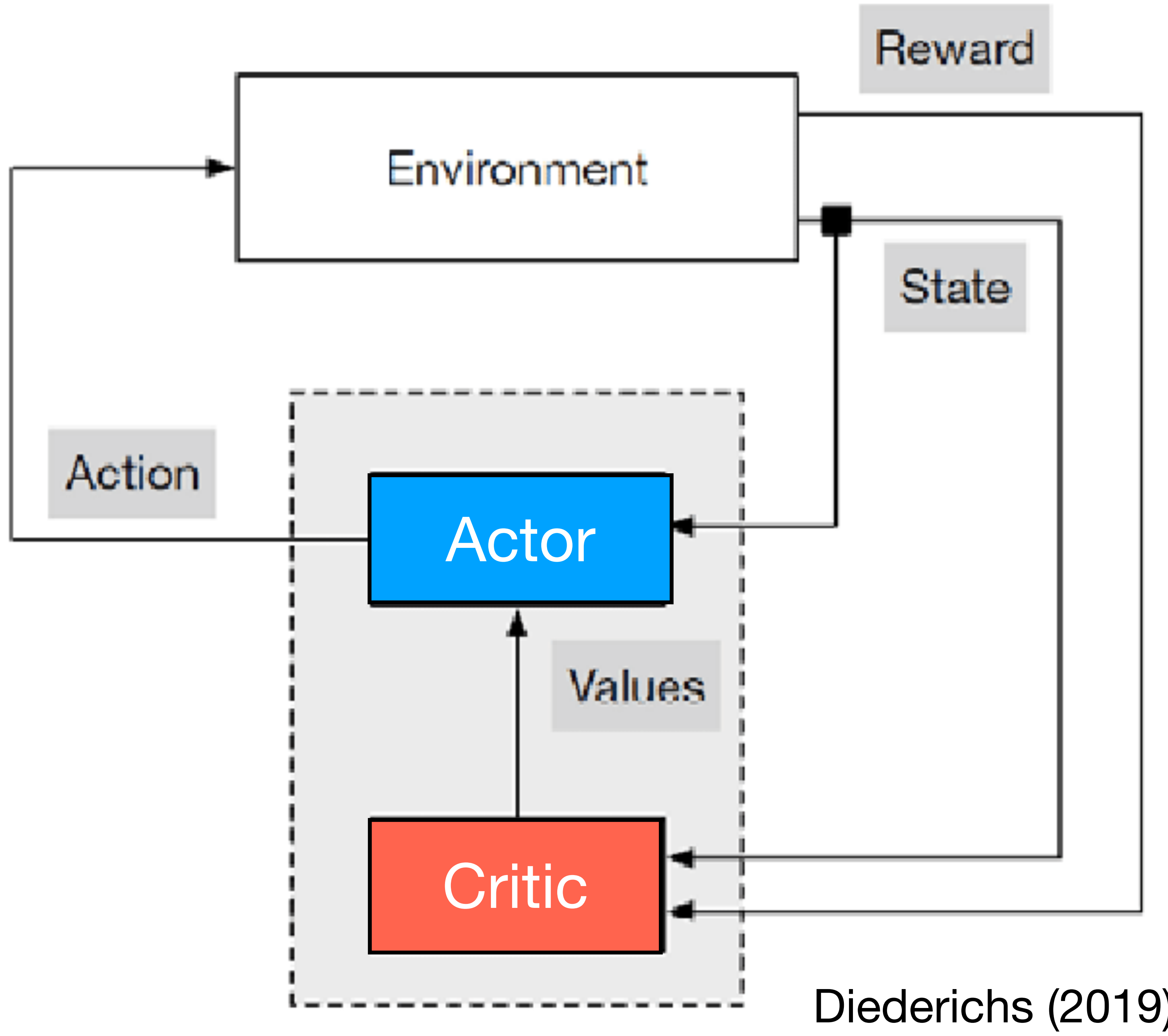


Diederichs (2019)

Actor-Critic

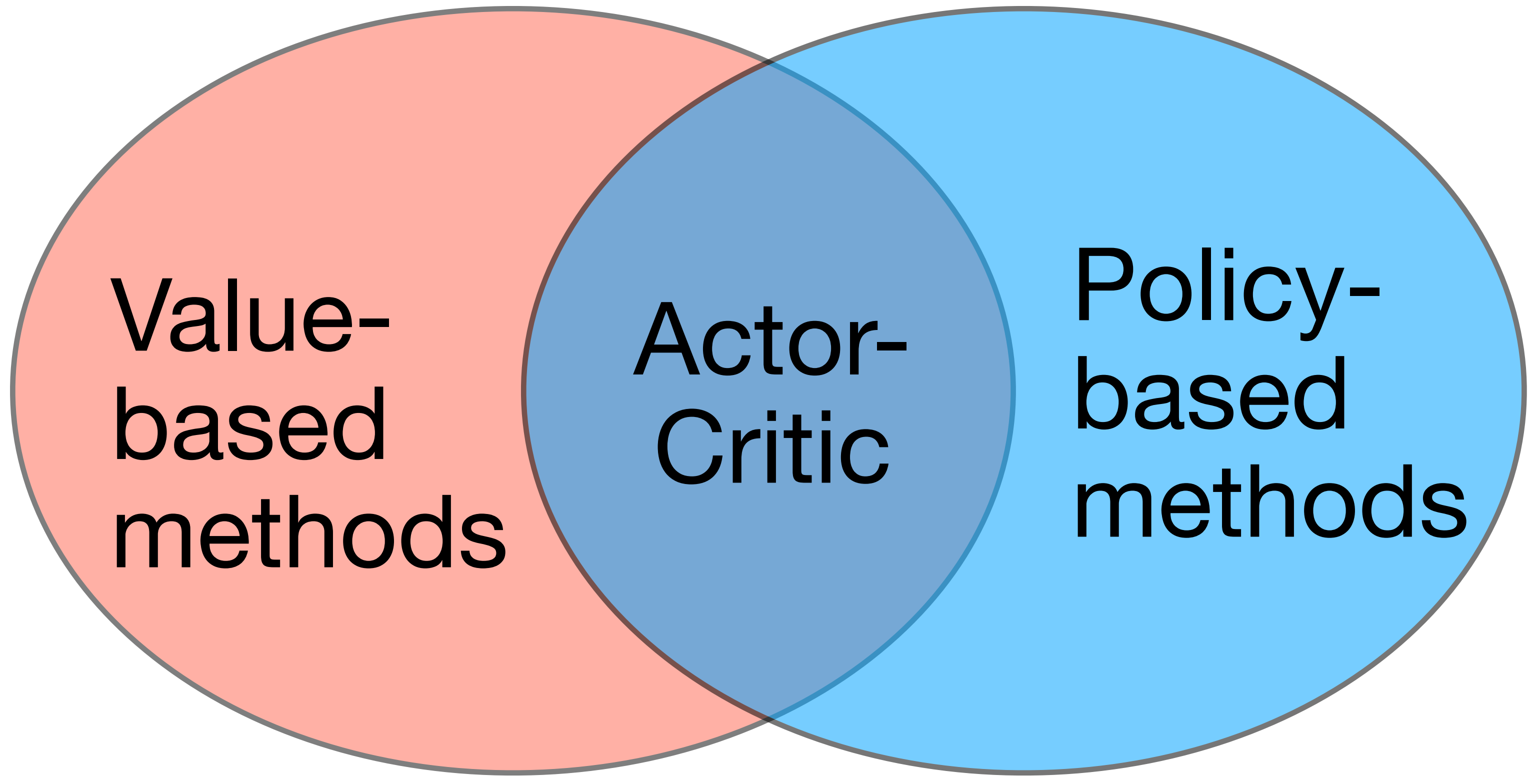


- Actor-critic combines **value-based** and **policy-based** methods and is a generalization of policy iteration
  - **Actor** provides the policy $\pi_\theta(a \mid s)$ parameterized by θ
  - **Critic** provides the value function $Q_{\mathbf{w}}(s, a)$ parameterized by $\mathbf{w}$
- Simulate trajectories $a \sim \pi_\theta(a \mid s)$ and compute TD error $\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$
- Iteratively update actor and critic
  - **Critic update**: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$
    reduce prediction error

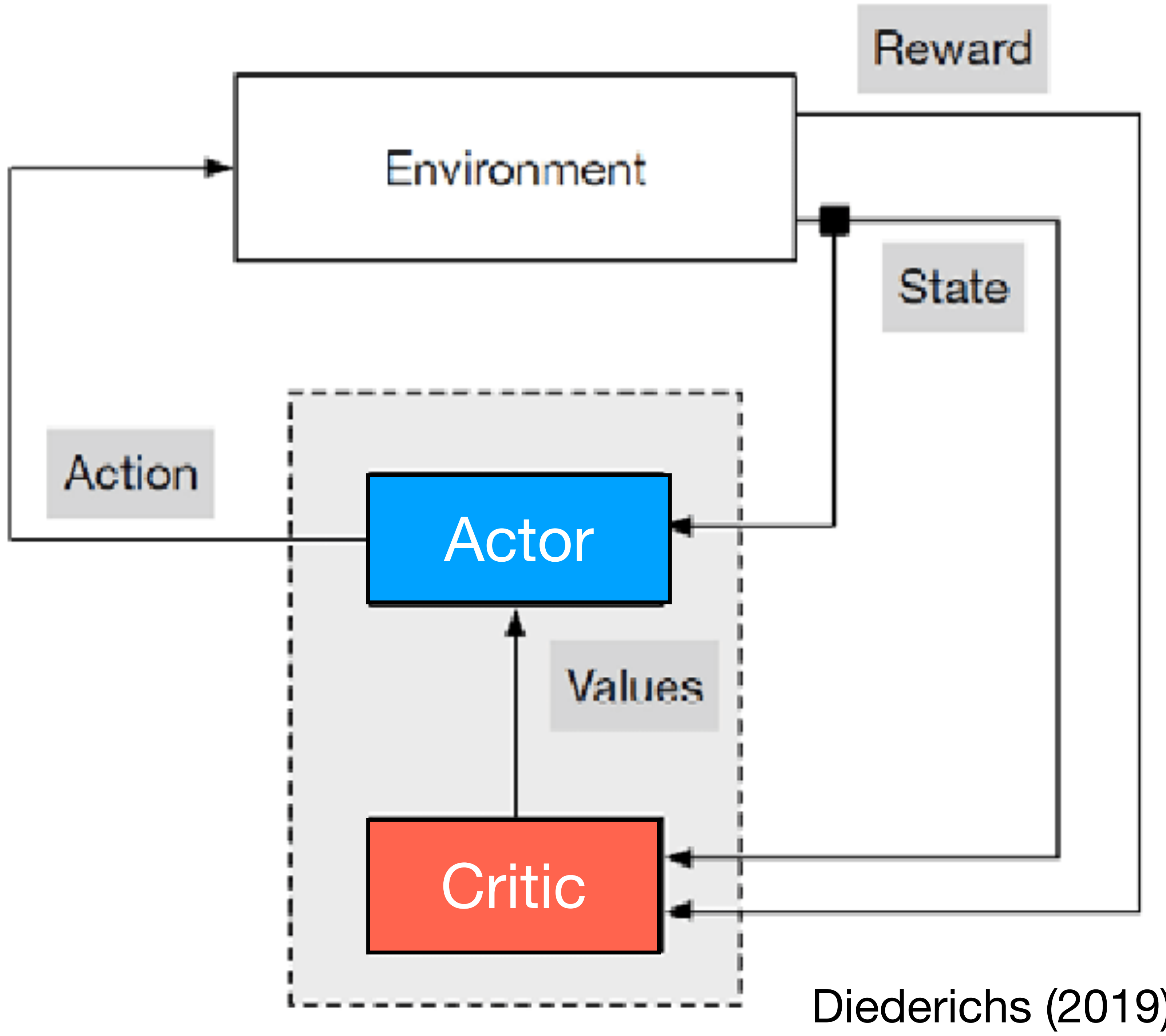


Diederichs (2019)

Actor-Critic



- Actor-critic combines **value-based** and **policy-based** methods and is a generalization of policy iteration
  - **Actor** provides the policy $\pi_\theta(a \mid s)$ parameterized by θ
  - **Critic** provides the value function $Q_{\mathbf{w}}(s, a)$ parameterized by $\mathbf{w}$
- Simulate trajectories $a \sim \pi_\theta(a \mid s)$ and compute TD error $\delta = r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$
- Iteratively update actor and critic
  - **Critic update**: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$
    reduce prediction error
  - **Actor update**: $\theta_{t+1} = \theta_t + \beta\delta \nabla_\theta \log \pi_\theta(a \mid s) Q_{\mathbf{w}}(s, a)$
    increase probability of highly rewarding actions



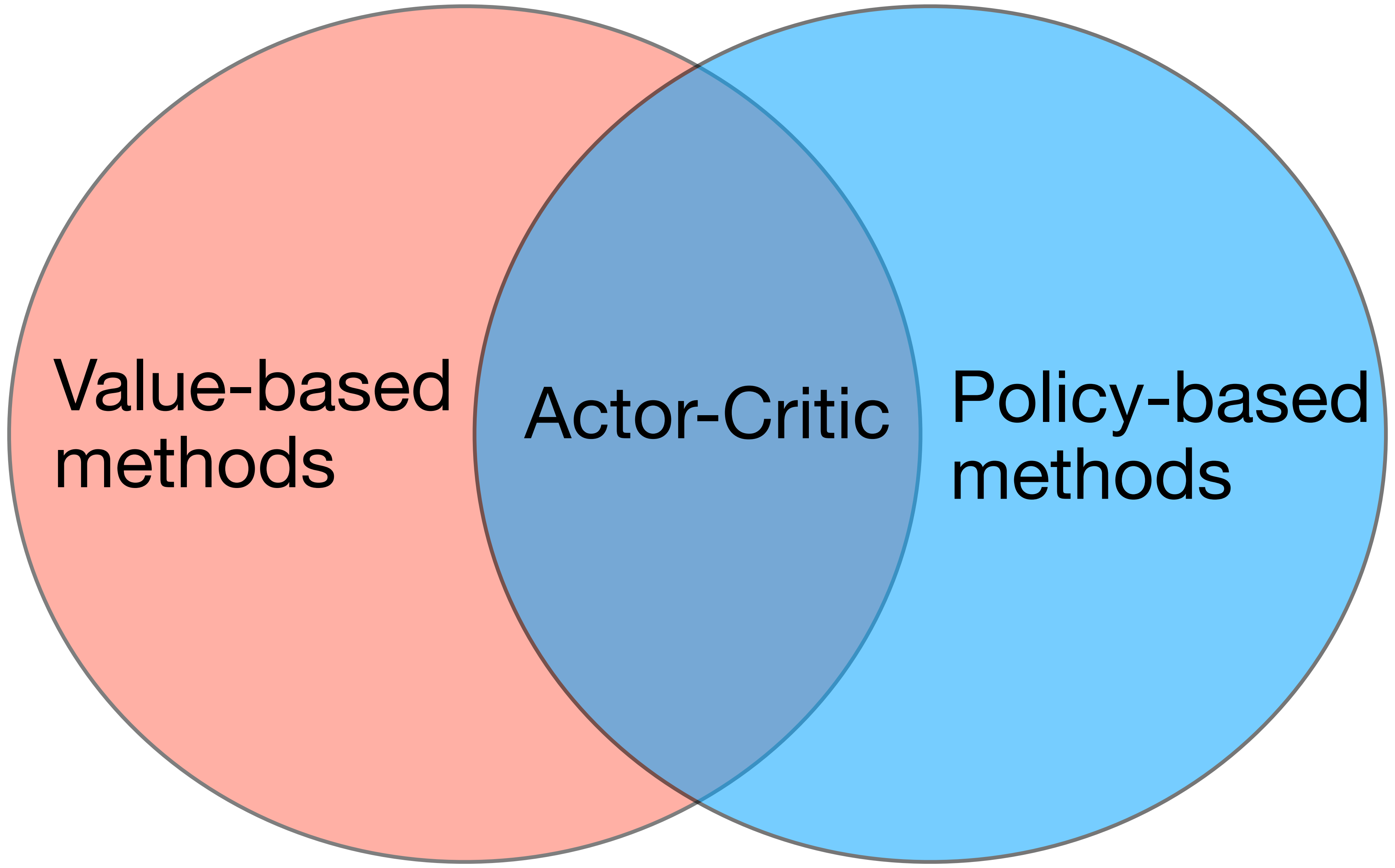
Diederichs (2019)

Model-free methods summary

- *Just put ANNs everywhere!*
- **Value-based** methods
  - Deep Q Learning
- **Policy-based** methods
  - Policy Gradient
- **Actor**-**Critic**
  - Integration of both value-based and policy-based methods

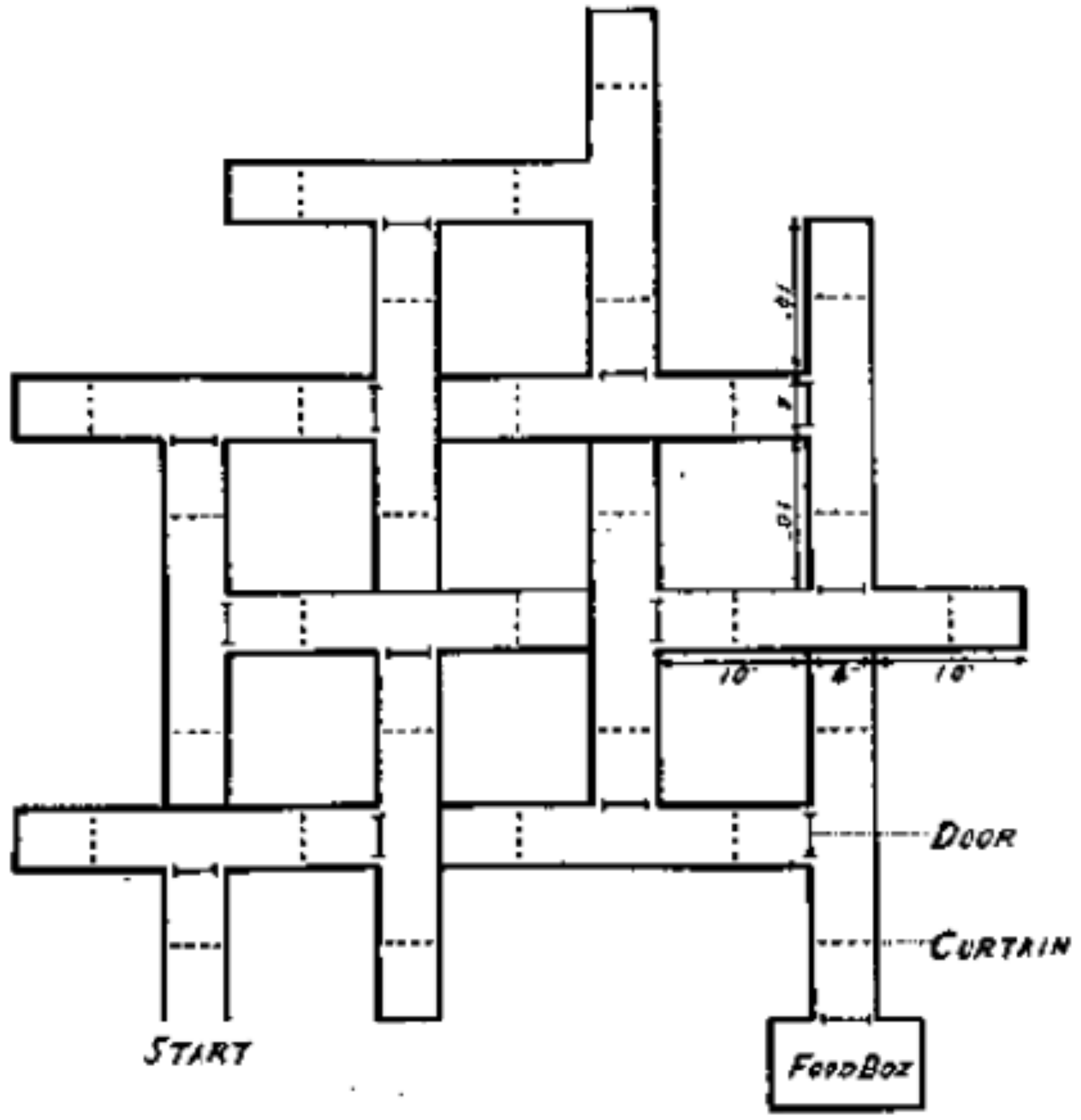
Model-free methods summary

- *Just put ANNs everywhere!*
- **Value-based** methods
  - Deep Q Learning
- **Policy-based** methods
  - Policy Gradient
- **Actor**-**Critic**
  - Integration of both value-based and policy-based methods

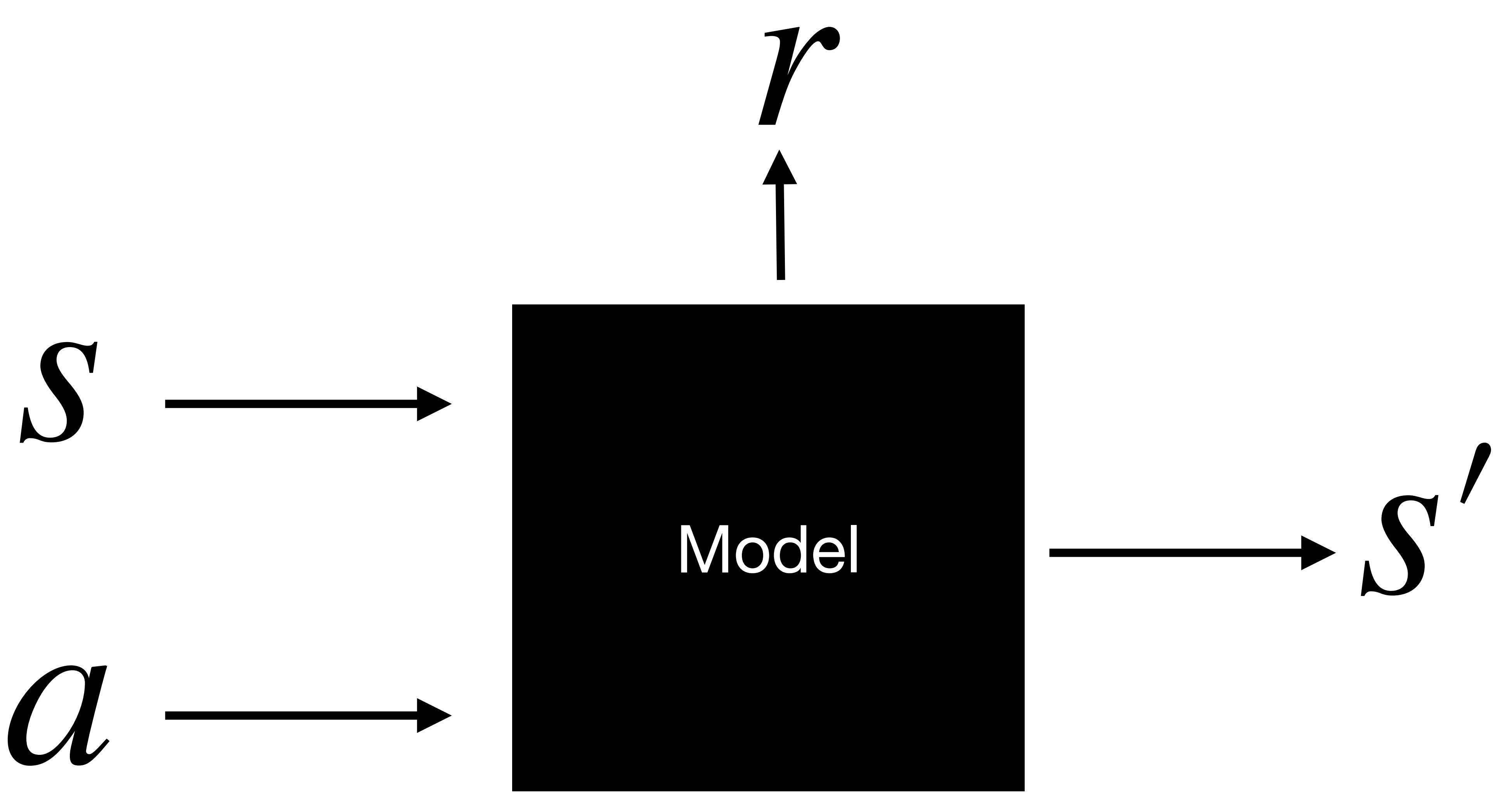


Model-based methods

- Learning a “field map of the environment” helps with planning and generalization
- But how is the model learned?
- And how is it used to in RL?
- We will discuss
  - Learning transitions via Delta-Rule
  - DYNA for simulating experiences
  - World Models
  - Dreamer V3



Plan of maze
14-Unit T-Alley Maze
Fig. 1
(From M. H. Elliott, The effect of change of reward on the maze performance of rats. *Univ. Calif. Publ. Psychol.*, 1928, 4, p. 20.)



Learning the model through experience

- Follow whatever policy (e.g., random) and update the transition matrix using delta-rule

$$T_{t+1}(s' \mid s, a) \leftarrow T_t(s' \mid s, a) + \alpha \left(\delta(s', s) - T_t(s' \mid s, a) \right)$$

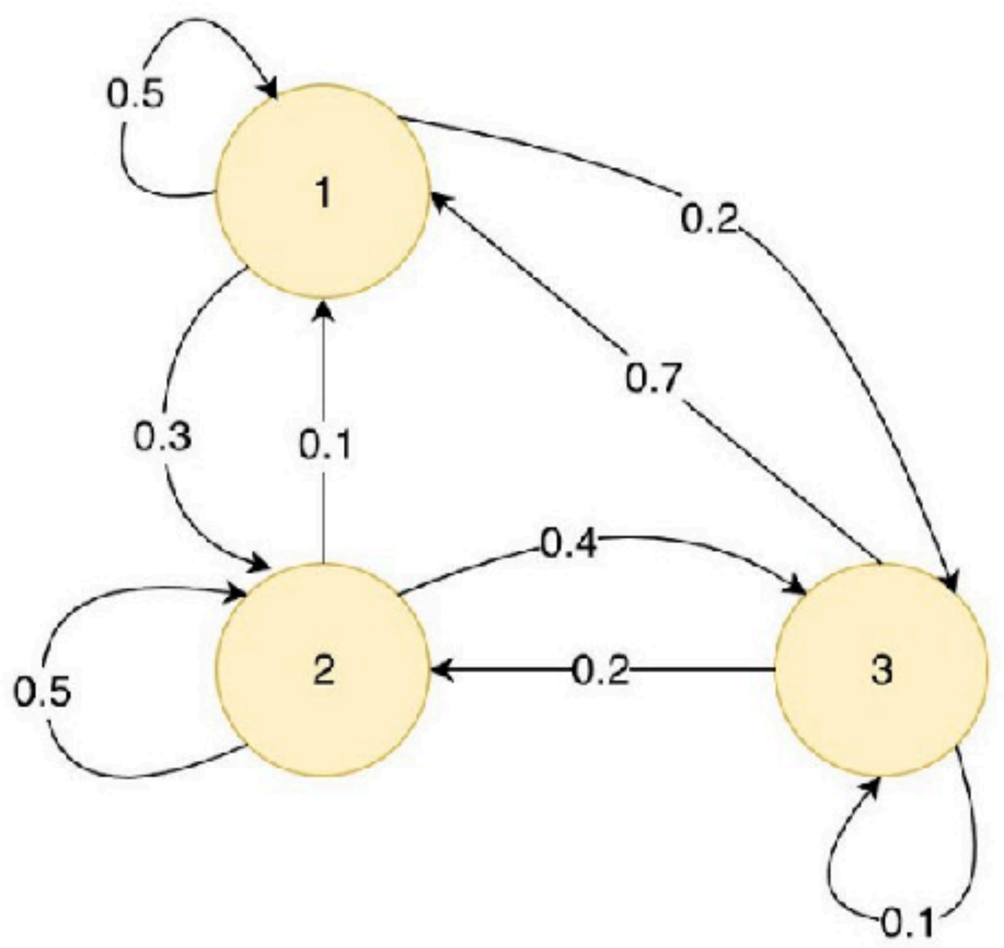
  - Kronecker delta $\delta(s', s') = 1$ when the transition occurs (i.e., $s \rightarrow s'$)

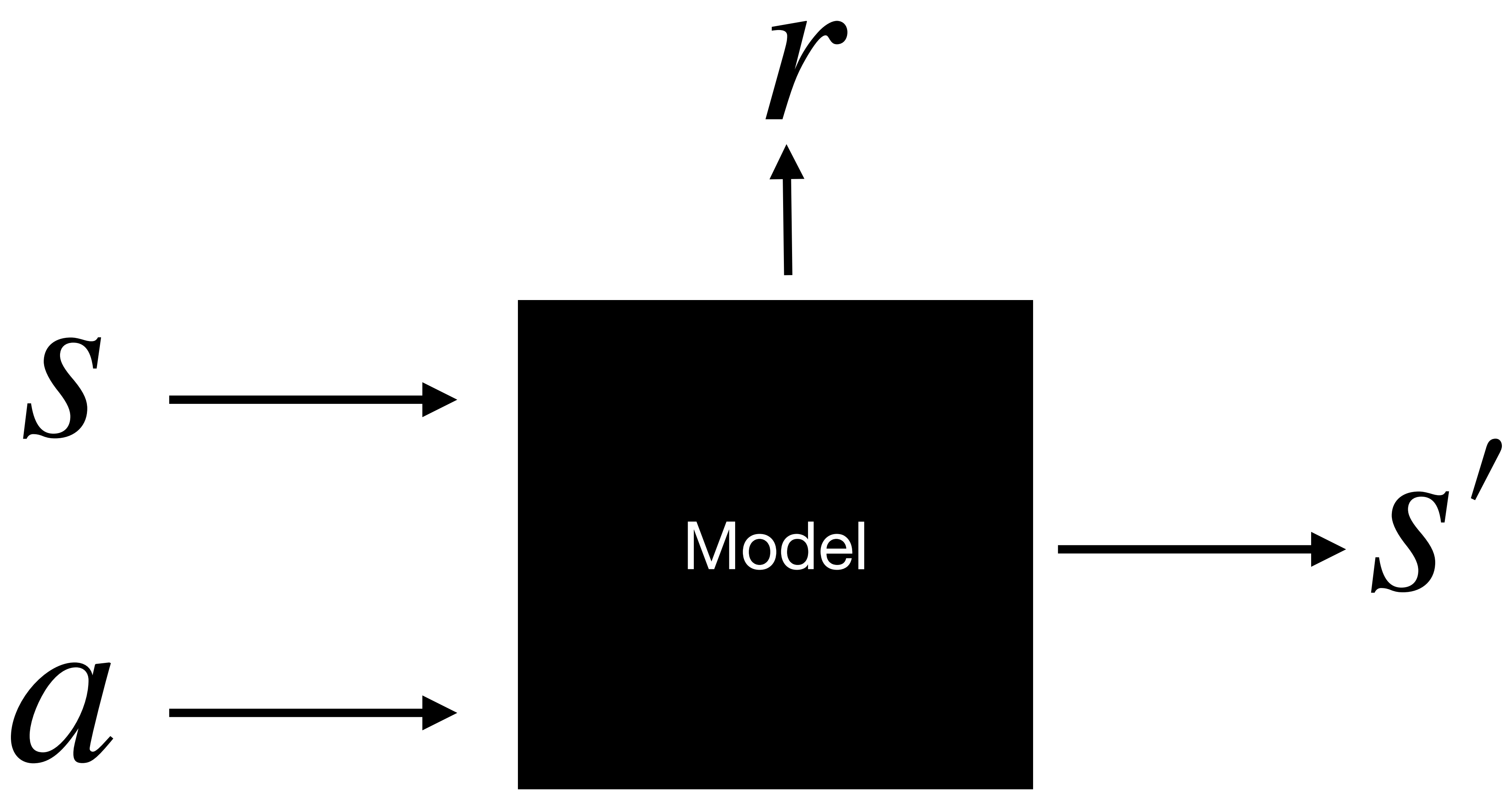
- $Model(s, a) \rightarrow [s', r]$ provided by learned transition matrix $T(s' \mid s, a)$ and value function $Q(s, a)$ or $V(s)$

Transition Matrix

$$\begin{array}{cc} & \begin{array}{ccc} s_1 & s_2 & s_3 \end{array} \\ \begin{array}{c} s_1 \\ s_2 \\ s_3 \end{array} & \begin{bmatrix} 0.5 & 0.1 & 0.7 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.4 & 0.1 \end{bmatrix} \end{array}$$

MDP





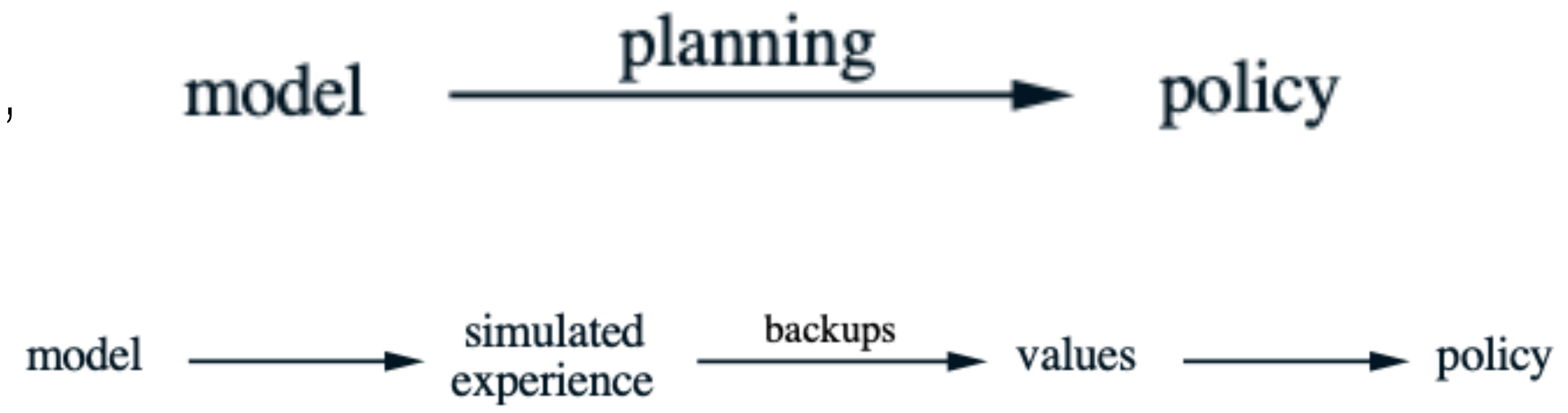
Simulating experiences with DYNA

- Models of the environment can be used for planning

$$\text{model} \xrightarrow{\text{planning}} \text{policy}$$

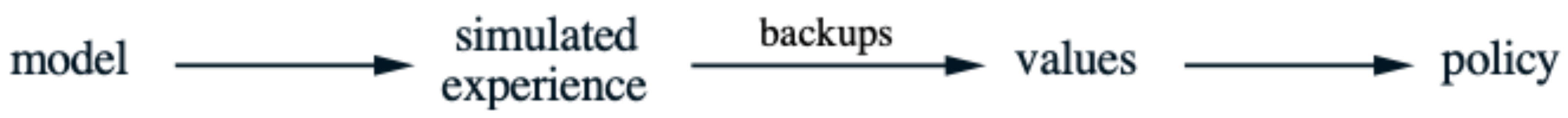
Simulating experiences with DYNA

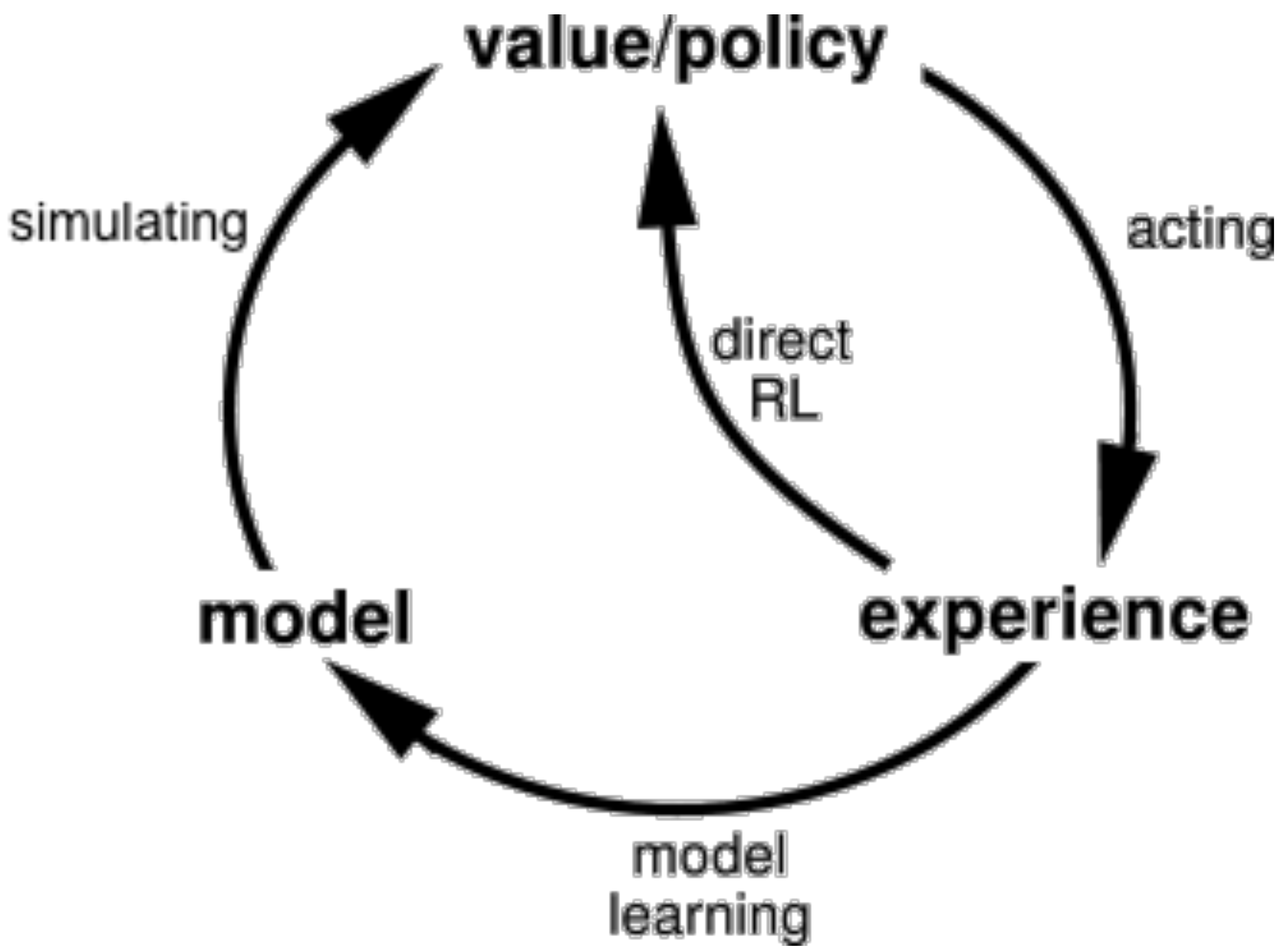
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences



Simulating experiences with DYNA

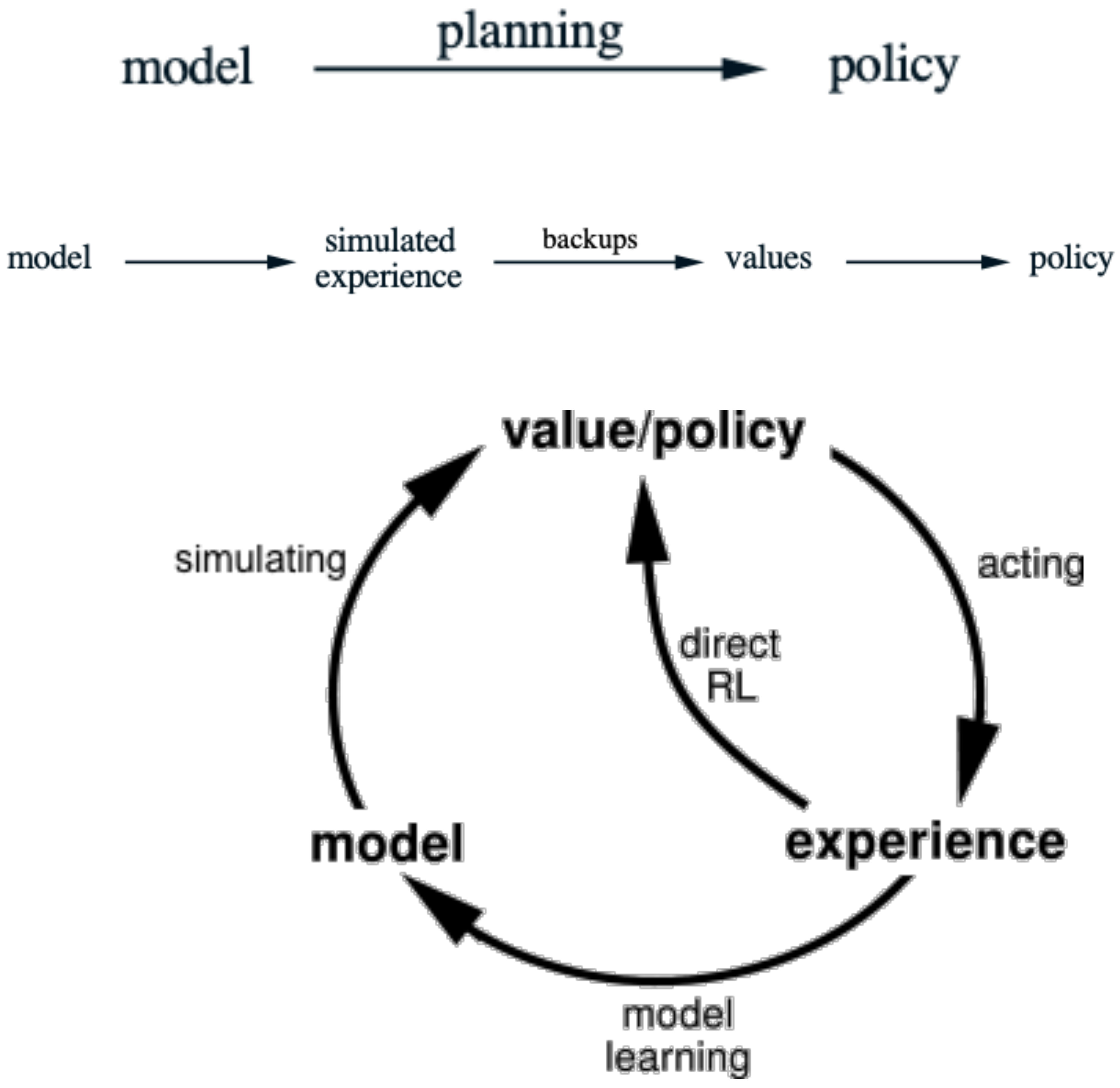
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences





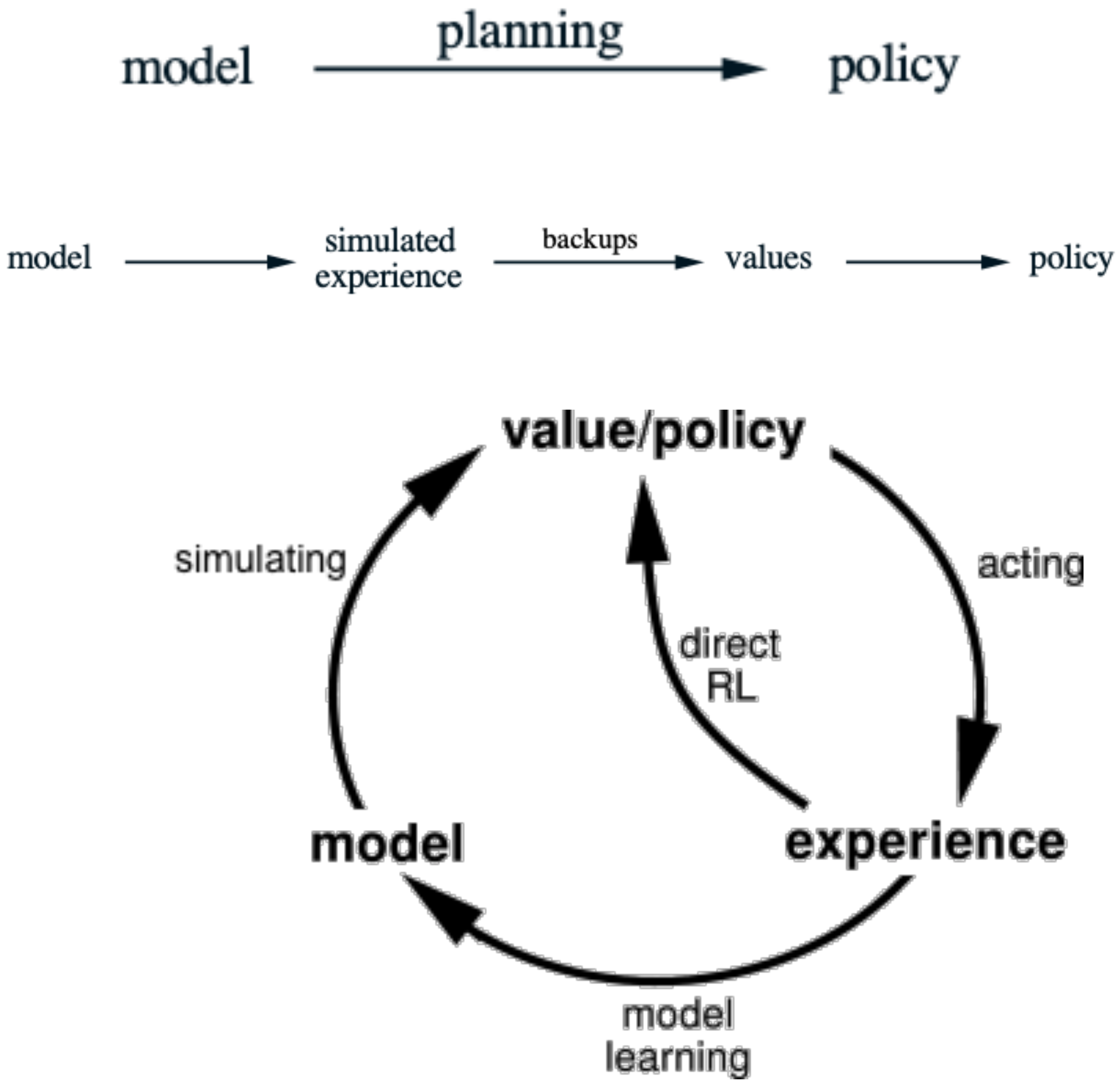
Simulating experiences with DYNA

- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
  1. **Direct RL**: Execute real actions and update value function
  $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$



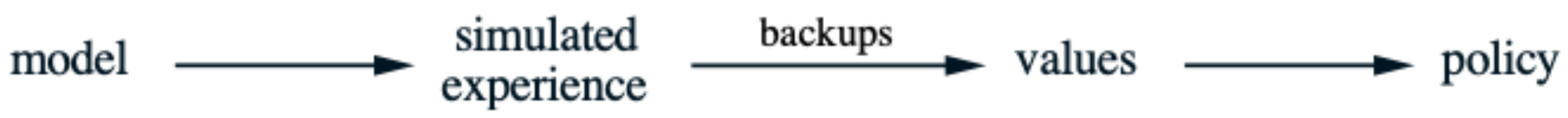
Simulating experiences with DYNA

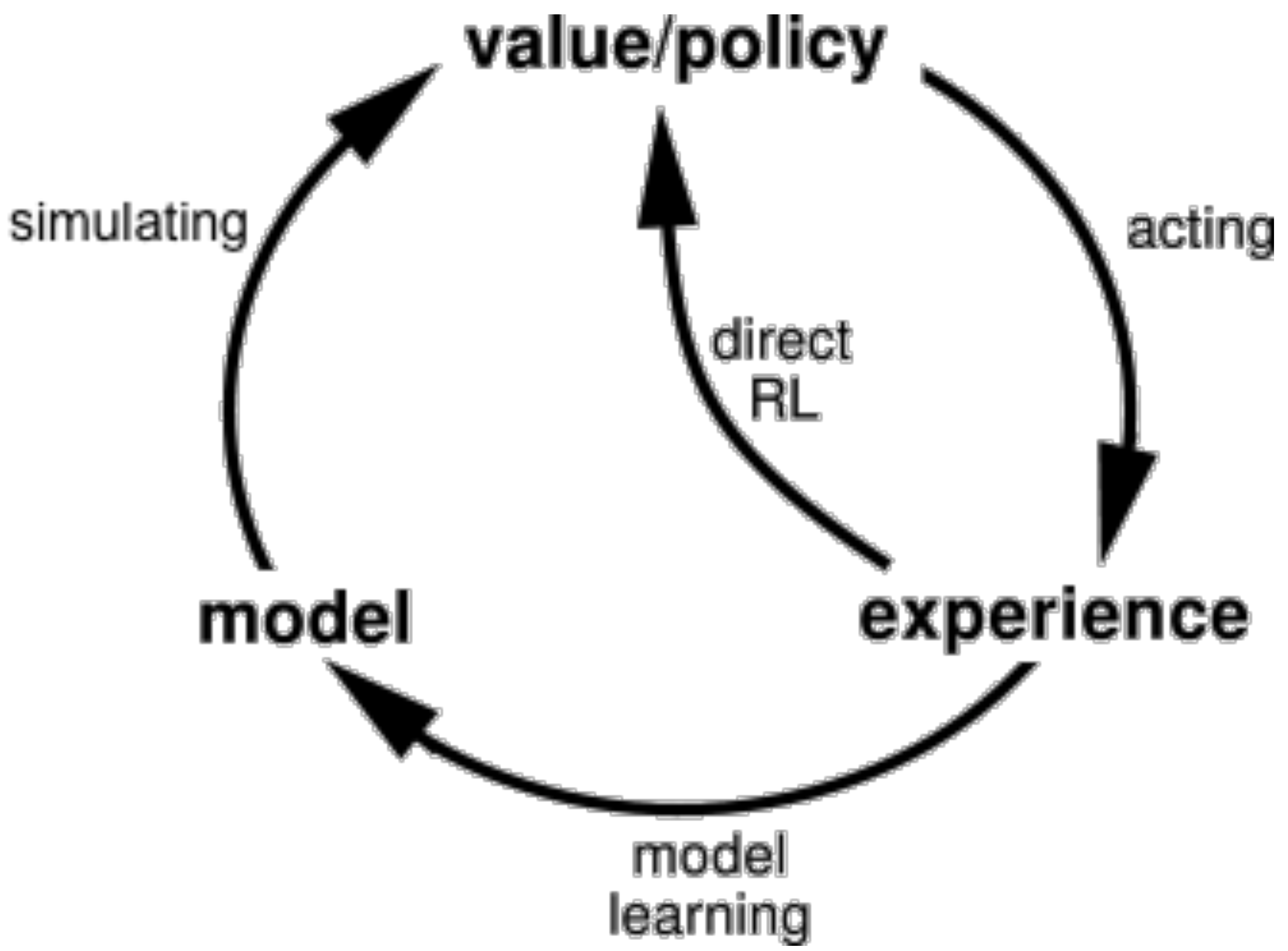
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
    1. **Direct RL**: Execute real actions and update value function
    $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
    2. **Model learning**:



Simulating experiences with DYNA

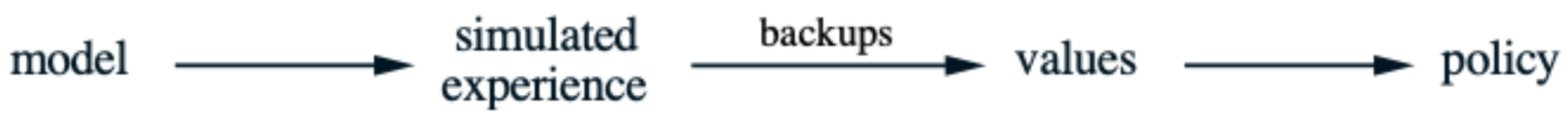
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
  1. **Direct RL**: Execute real actions and update value function
  $$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$
  2. **Model learning**:
     a. Update model of the environment $Model(s, a)$

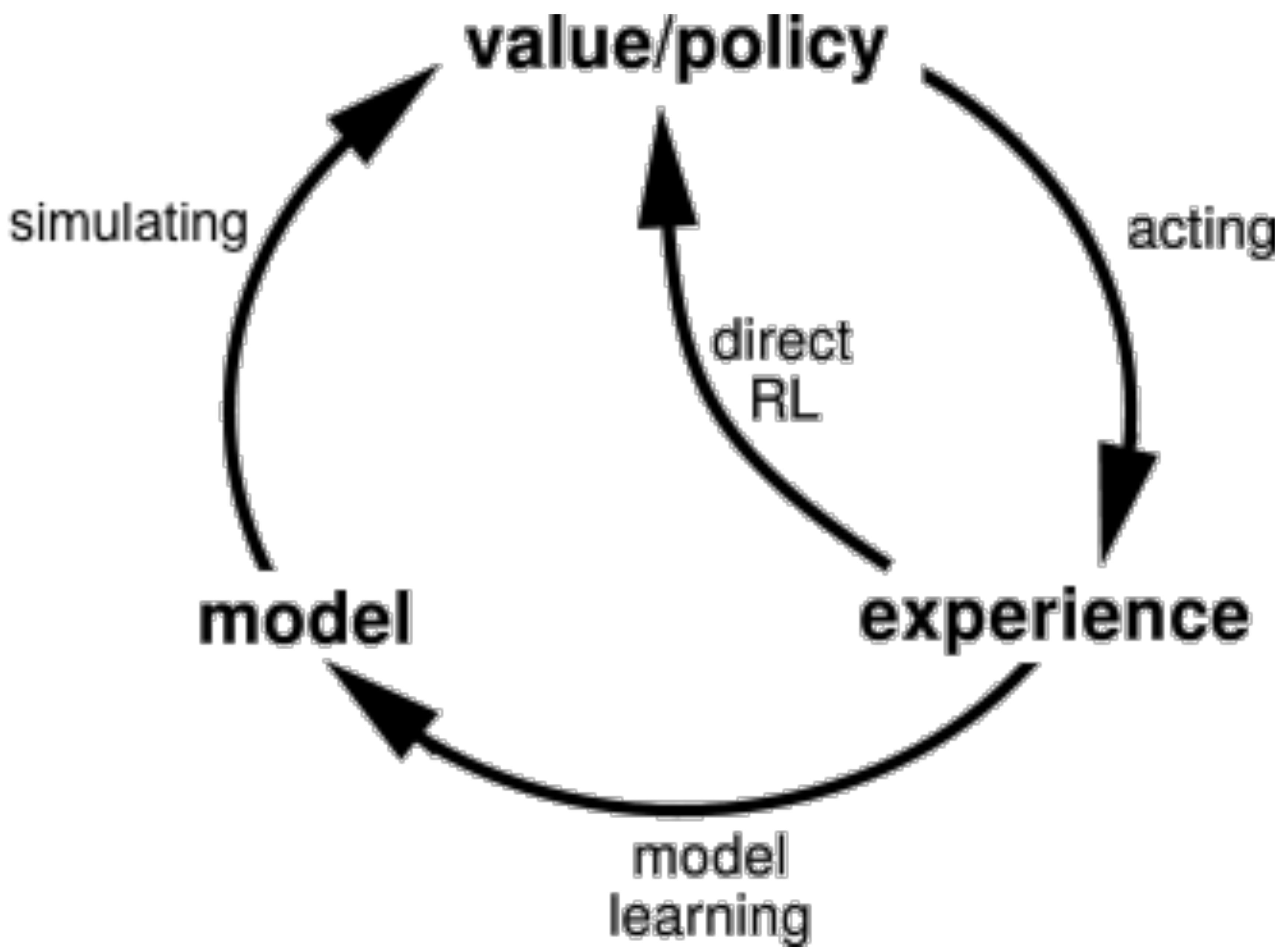




Simulating experiences with DYNA

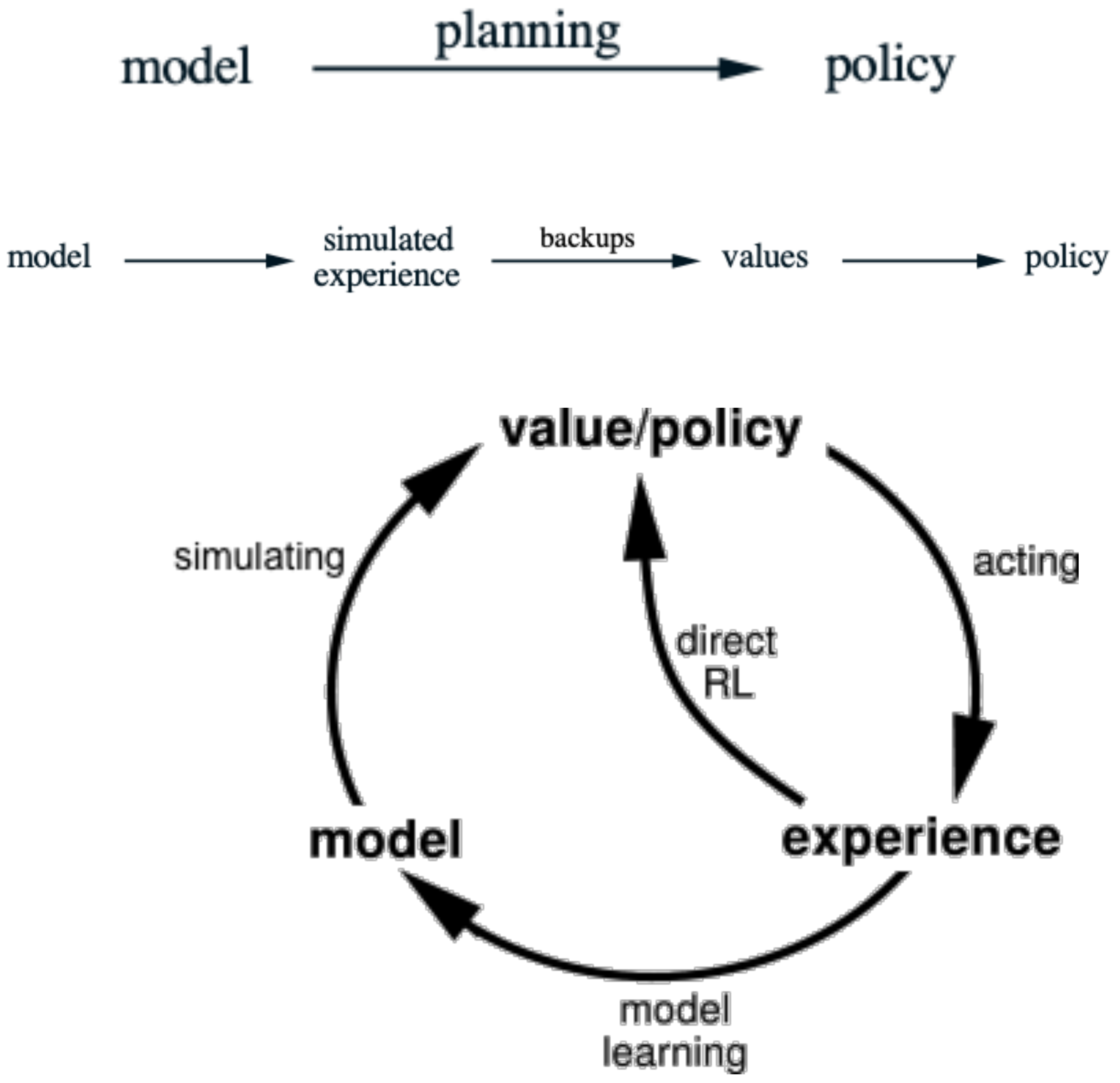
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
  1. **Direct RL**: Execute real actions and update value function
     $$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$
  2. **Model learning**:
     a. Update model of the environment $Model(s, a)$
     b. Simulate experiences:





Simulating experiences with DYNA

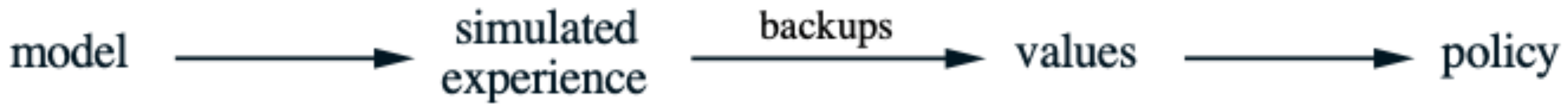
- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
  1. **Direct RL**: Execute real actions and update value function
     $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
  2. **Model learning**:
     a. Update model of the environment $Model(s,a)$
     b. Simulate experiences:
        - Give previously observed states and actions to model
          $[s,a] \rightarrow Model(s,a) \rightarrow [s',r]$

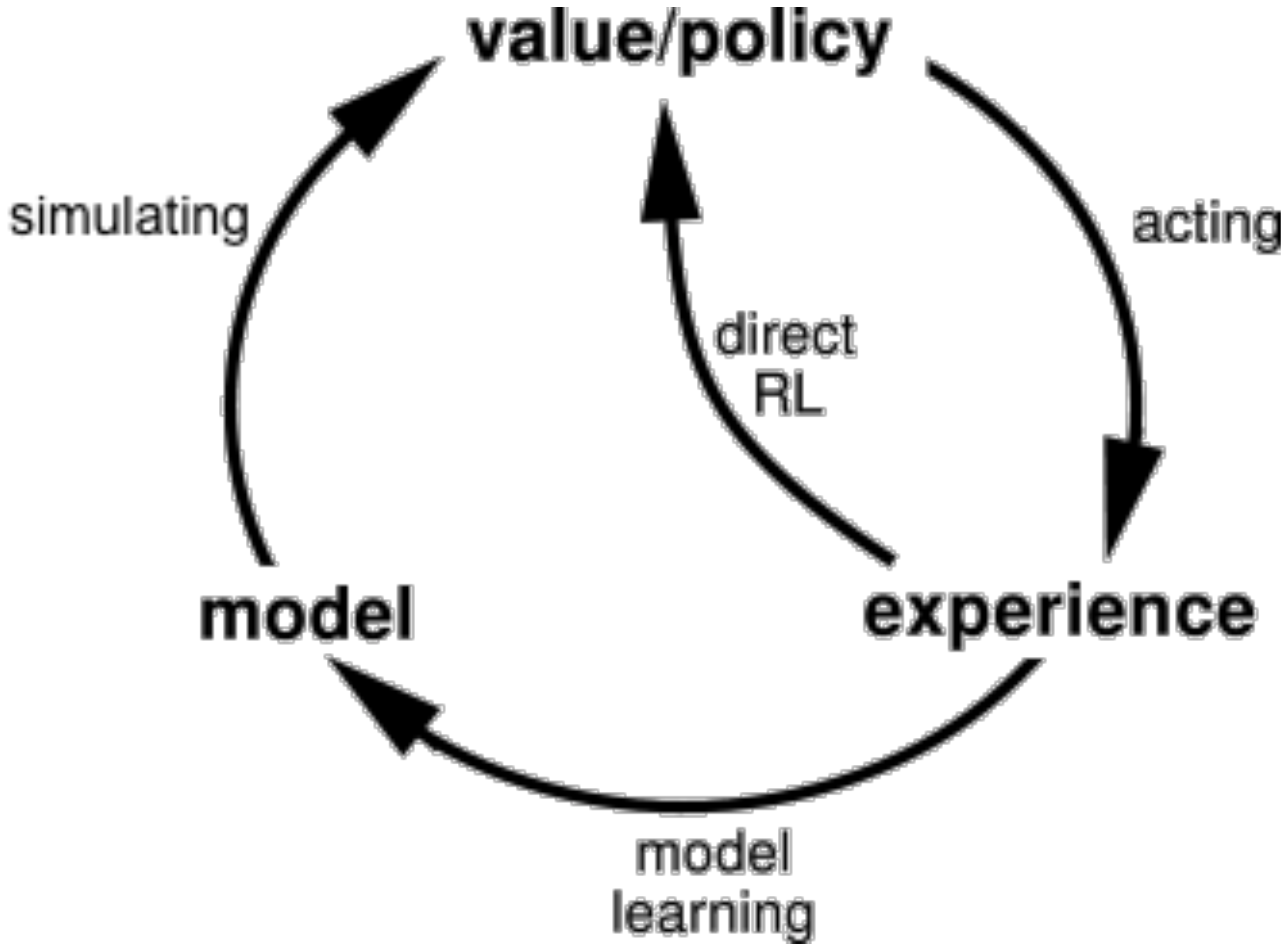


Simulating experiences with DYNA

- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences

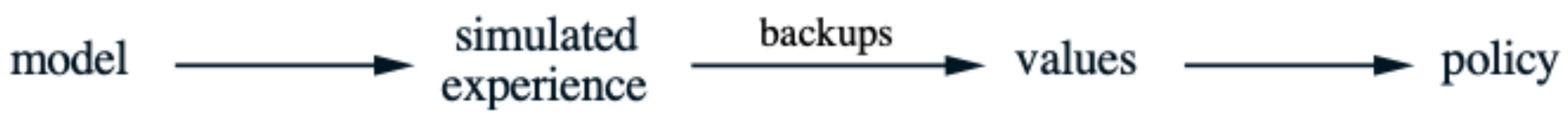
1. **Direct RL**: Execute real actions and update value function
$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
2. **Model learning**:
    a. Update model of the environment $Model(s,a)$
    b. Simulate experiences:
        - Give previously observed states and actions to model
        $$[s,a] \rightarrow Model(s,a) \rightarrow [s',r]$$
        - Update value function with simulated experiences
        $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

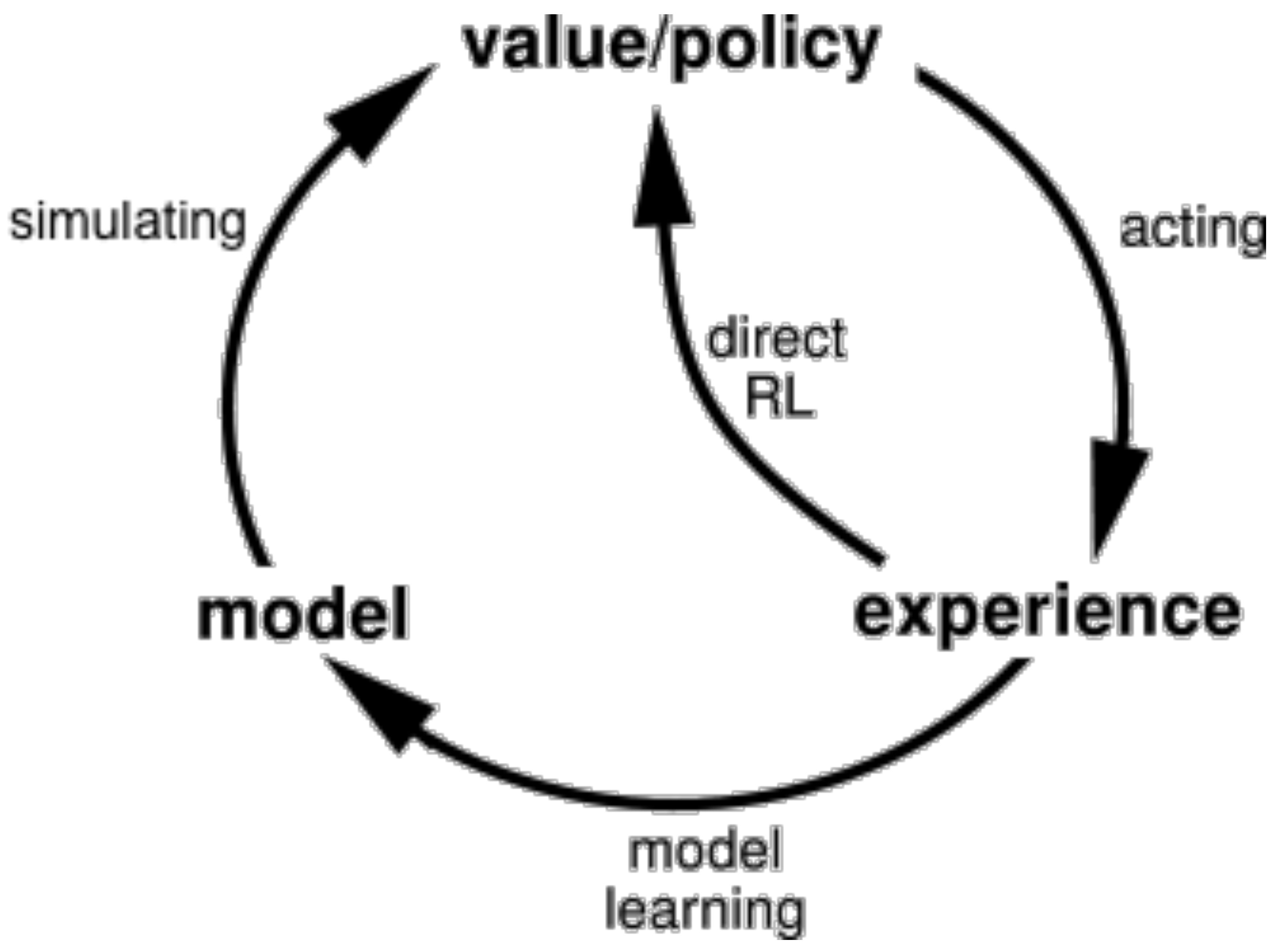




Simulating experiences with DYNA

- Models of the environment can be used for planning
- DYNA uses simulated experiences to update policy/value functions, just like real experiences
  1. **Direct RL**: Execute real actions and update value function
  $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
  2. **Model learning**:
     a. Update model of the environment $Model(s,a)$
     b. Simulate experiences:
        - Give previously observed states and actions to model
        $$[s,a] \rightarrow Model(s,a) \rightarrow [s',r]$$
        - Update value function with simulated experiences
        $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
- These simulations can be controlled for better efficiency (e.g., prioritized sweeps of reward-relevant state visitations; Moore & Atkeson, 1993)

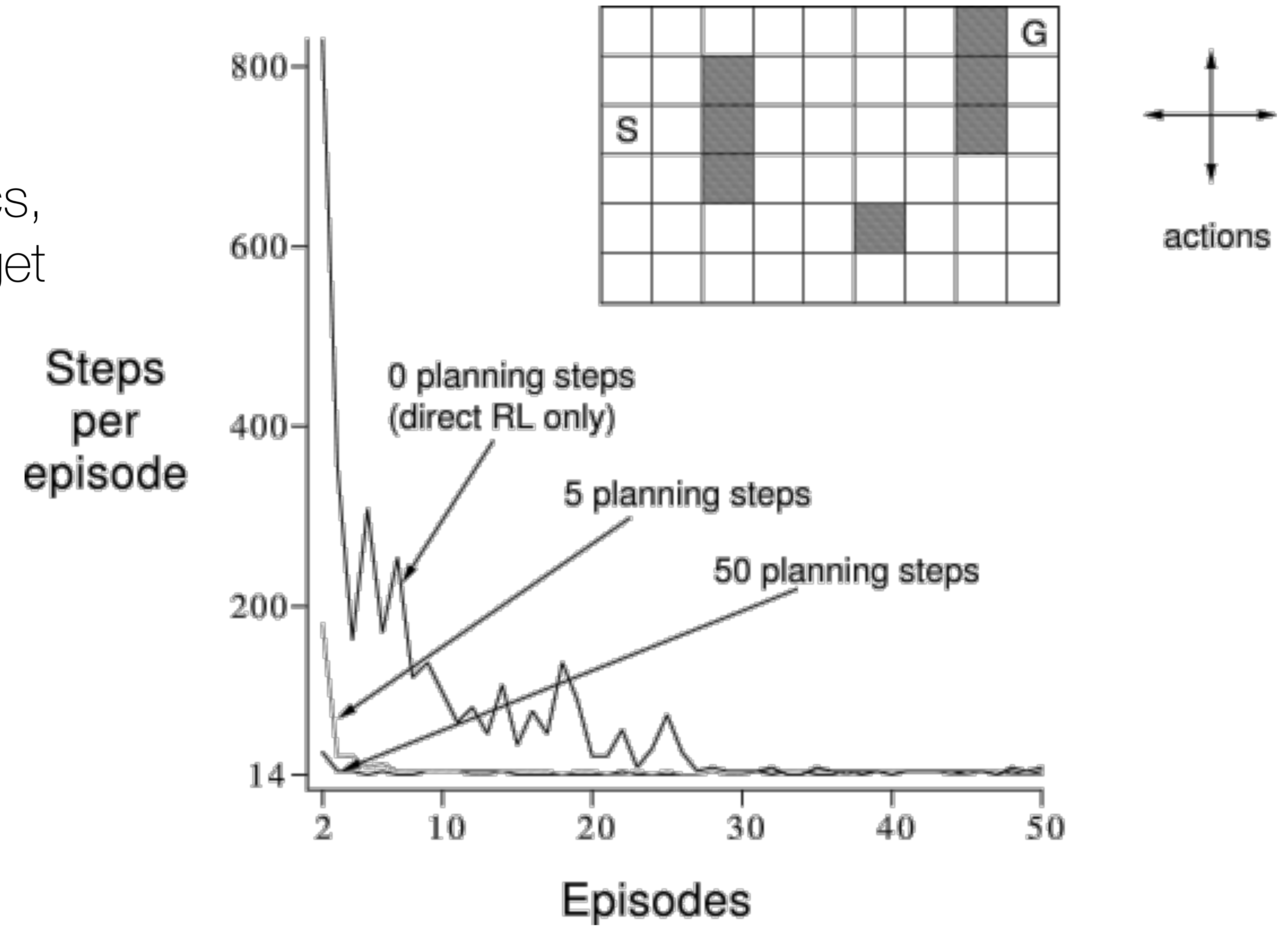




Benefits of Model-based planning

- Model-based planning needs far fewer real interactions with the environment (episodes) to learn better policies
  - Consider settings like self-driving cars, robotics, financial systems… where it is very costly to get real interaction data

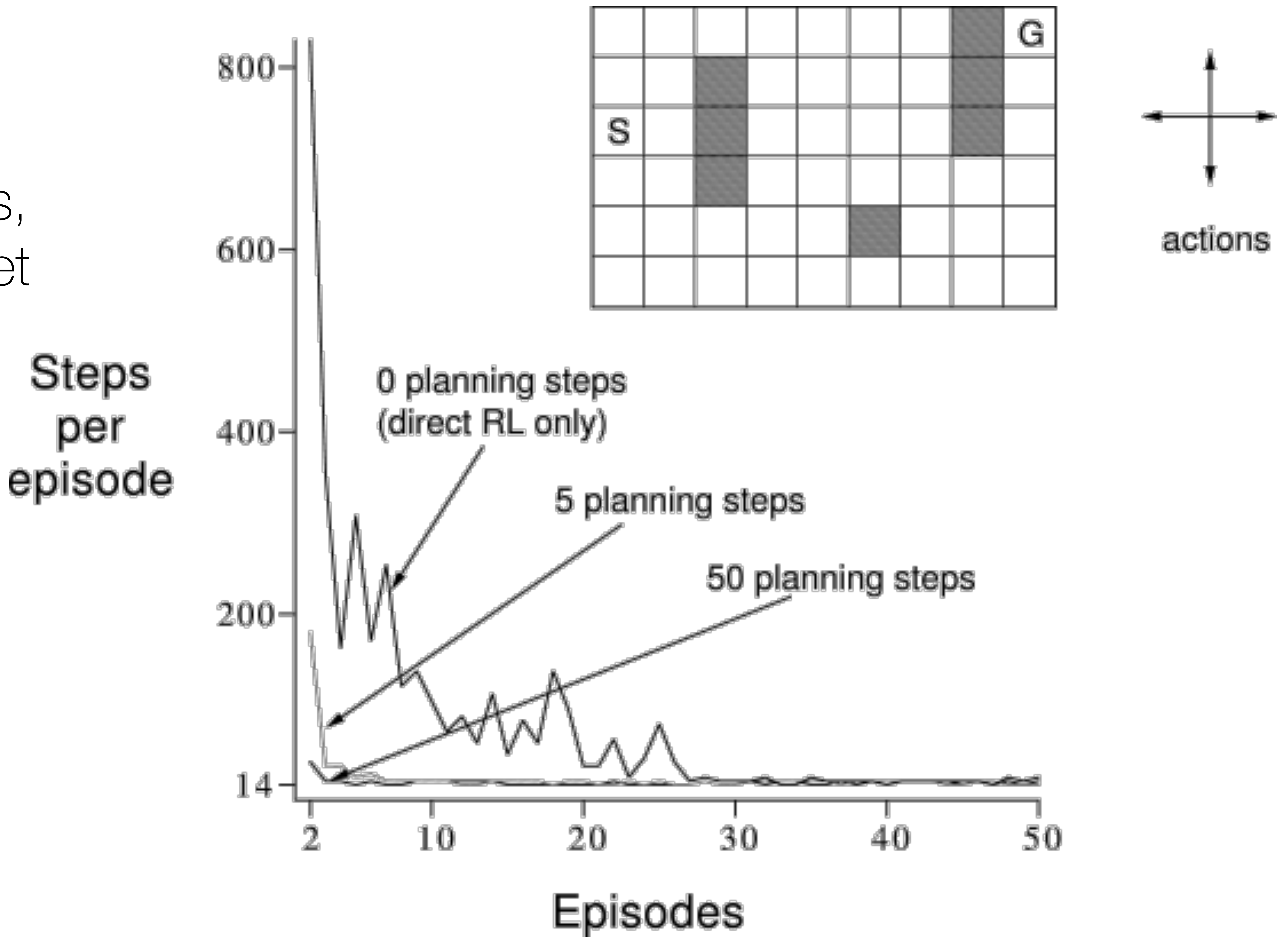
Sutton & Barto Fig. 9.5

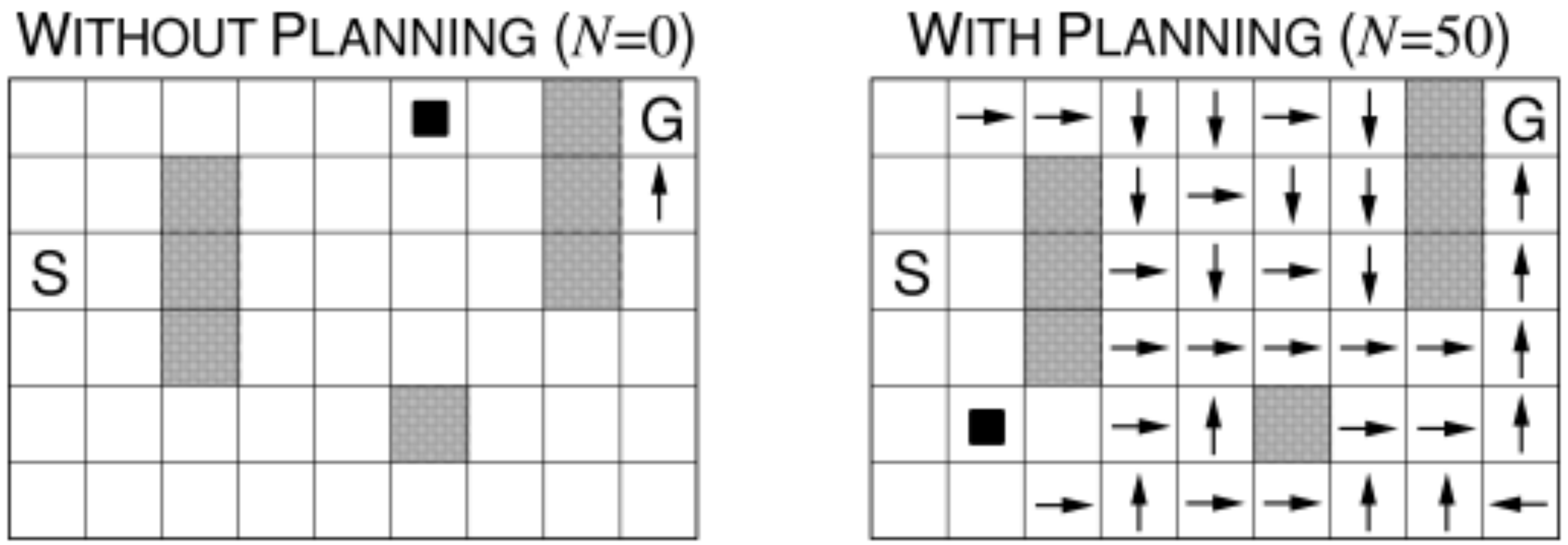


Benefits of Model-based planning

- Model-based planning needs far fewer real interactions with the environment (episodes) to learn better policies
    - Consider settings like self-driving cars, robotics, financial systems… where it is very costly to get real interaction data
- Halfway through only the 2nd episode…
    - Arrows show greedy action in each state and no arrow if all actions are equal

Sutton & Barto Fig. 9.5





Sutton & Barto Fig. 9.6

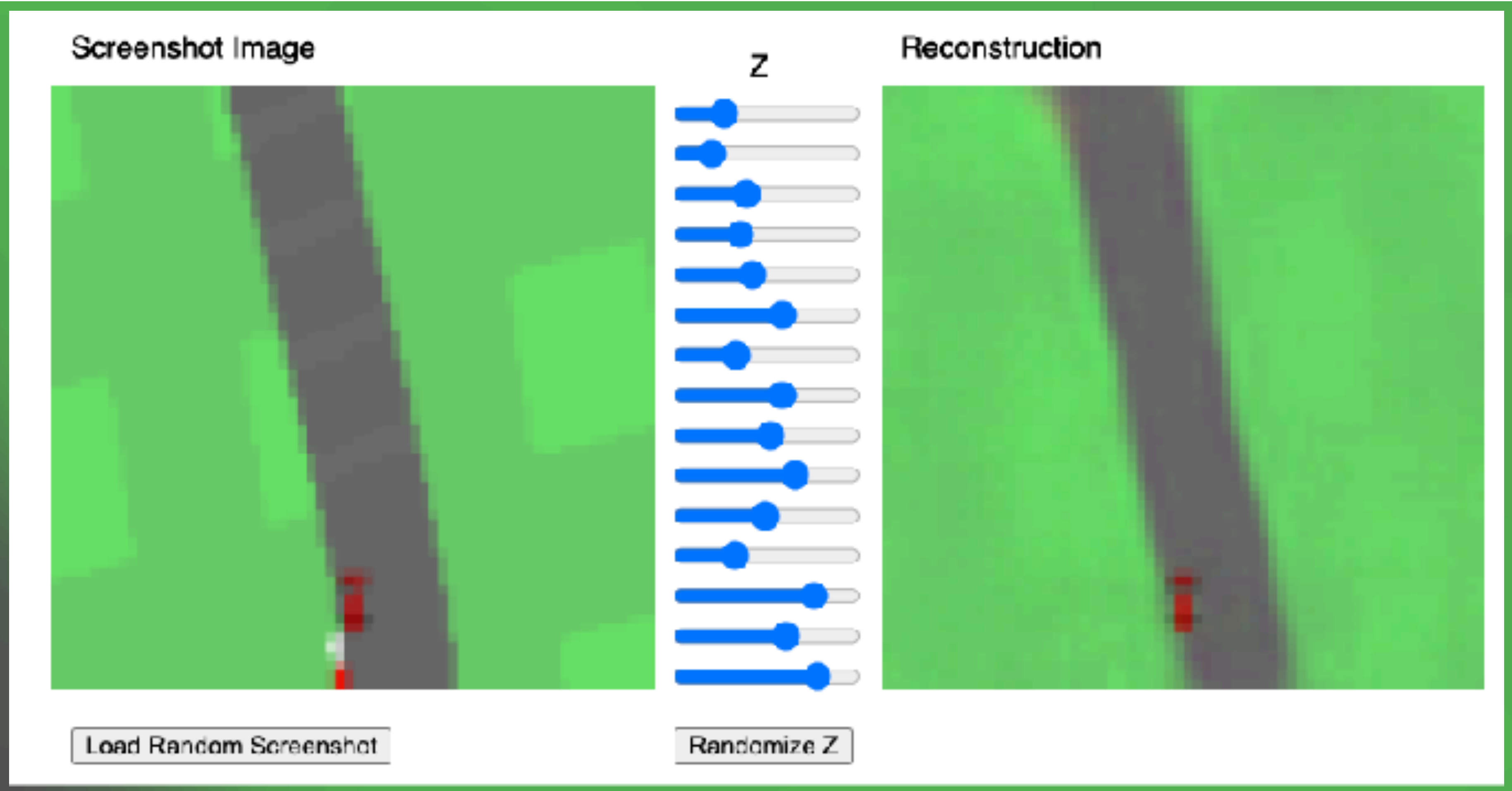
Recurrent World Models Facilitate Policy Evolution

NIPS 2018
Oral Presentation

Thirty-Second Annual Conference on Neural Information Processing Systems
Montréal, Canada

Interactive demo. Tap screen or use arrow keys to override the agent's decisions.

Recurrent World Models Facilitate Policy Evolution



Check out worldmodels.github.io/ for iteractive demos and more details!

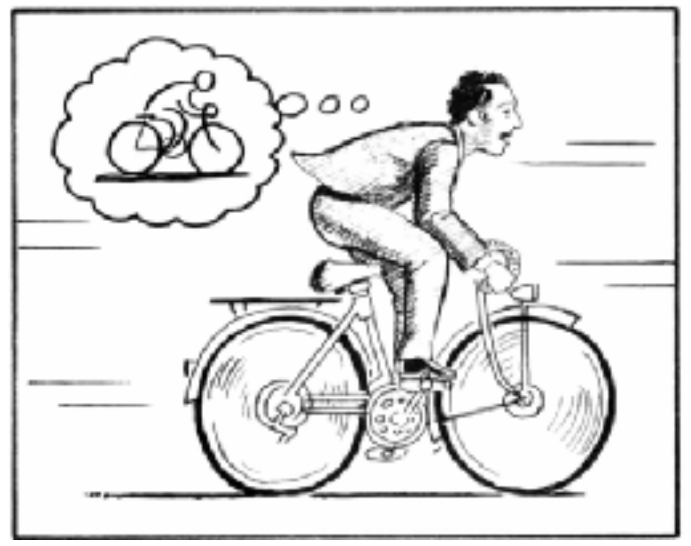
NIPS 2018
Oral Presentation

Thirty-Second Annual Conference on
Neural Information Processing Systems
Montréal, Canada

Interactive demo. Tap screen or use arrow keys to override the agent's decisions.

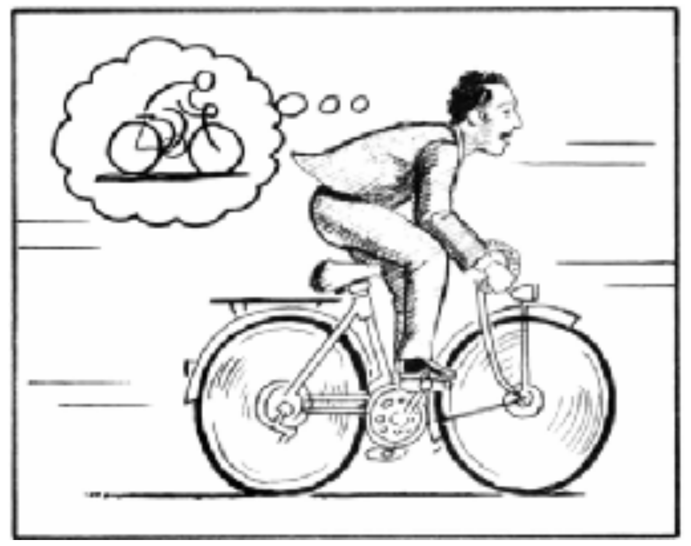
Ha & Schmidhuber (2018)

World models

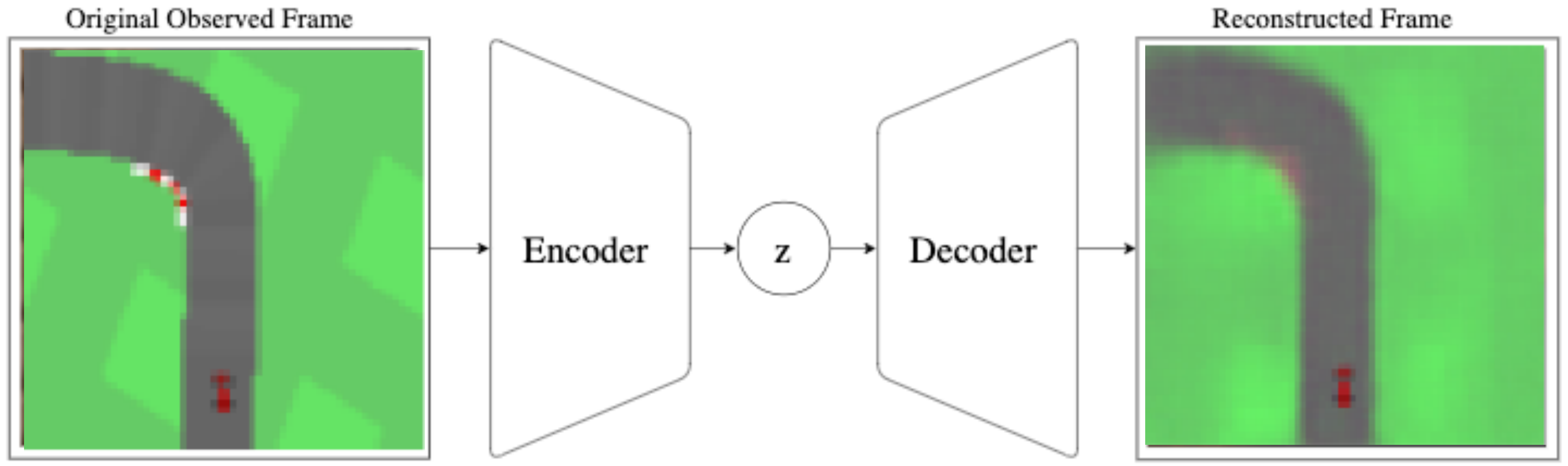


Ha & Schmidhuber (2018)

World models



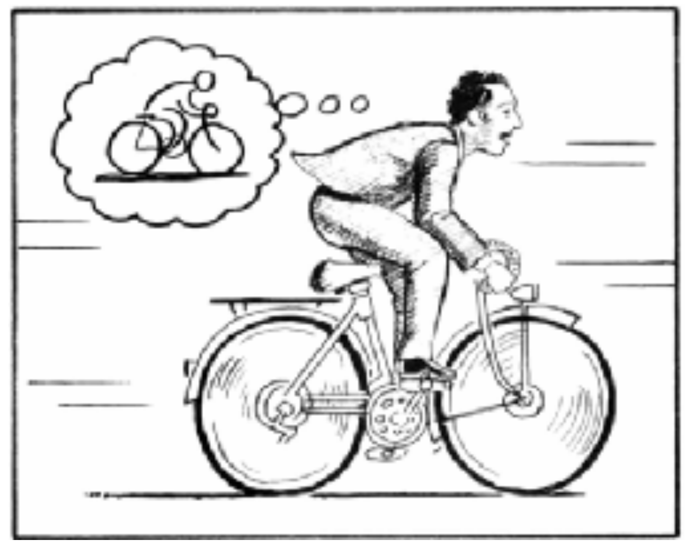
Variational Autoencoder (VAE)



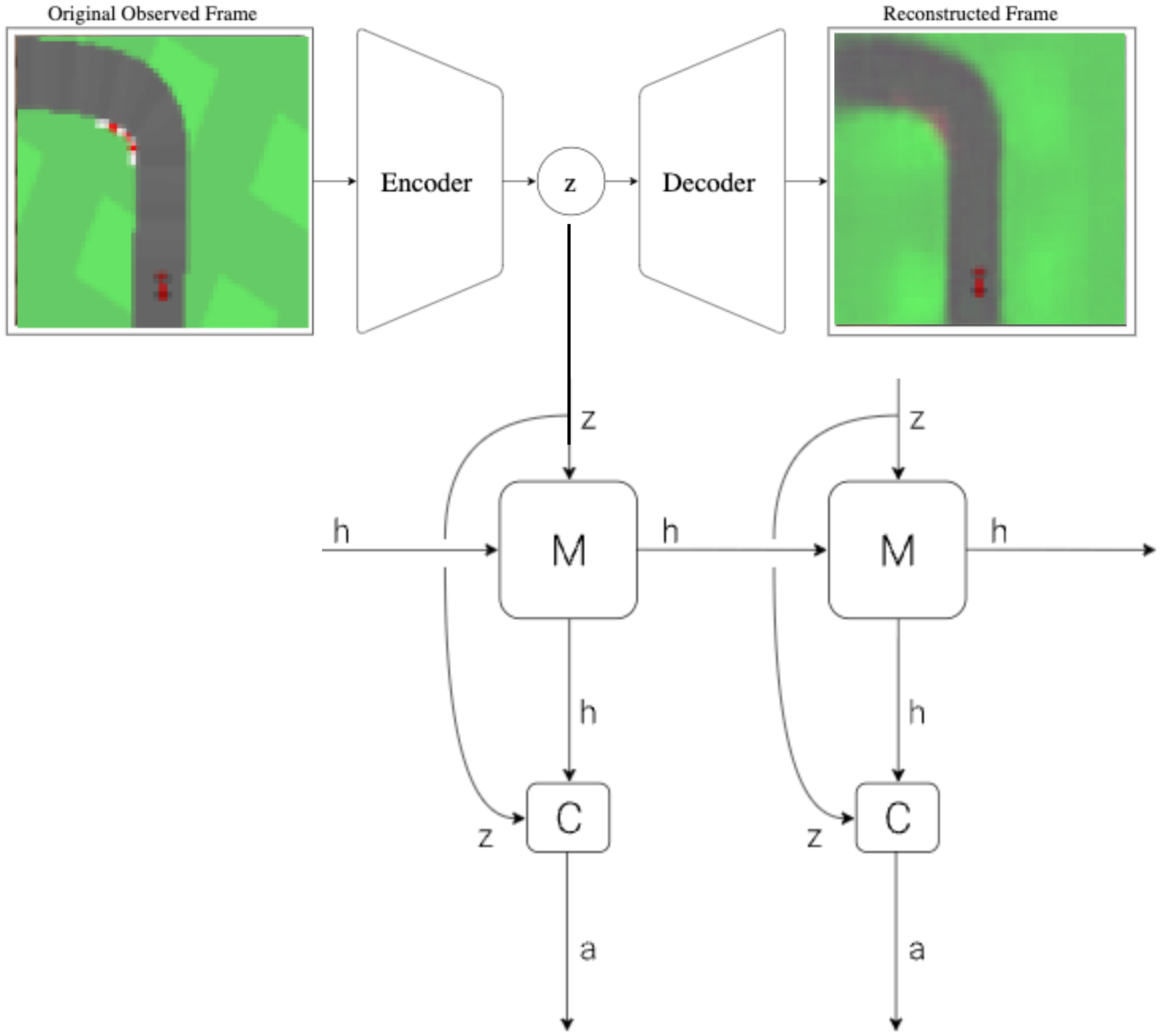
- **Vision Model (V)** encodes high-dimensional visual data into a low-dimensional latent vector z

Ha & Schmidhuber (2018)

World models



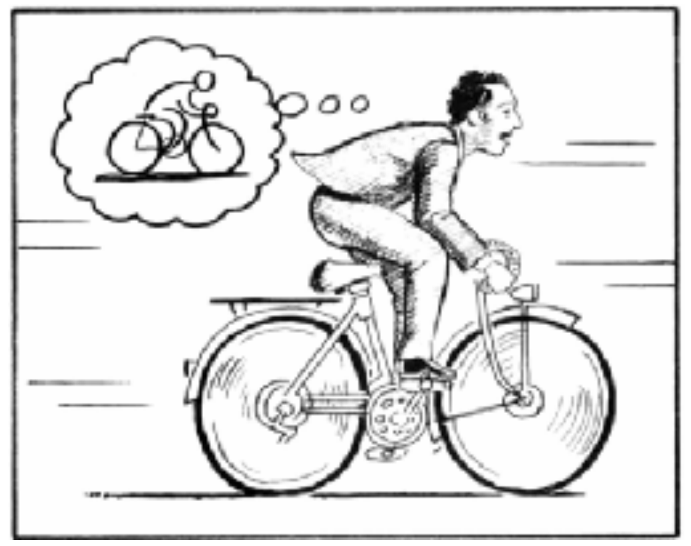
Variational Autoencoder (VAE)



- **Vision Model (V)** encodes high-dimensional visual data into a low-dimensional latent vector z
- **Memory RNN (M)** learns the temporal dynamics and predicts future states $P(z_{t+1} \mid a_t, z_t, h_t)$ where h is a hidden state vector capturing dynamics

Ha & Schmidhuber (2018)

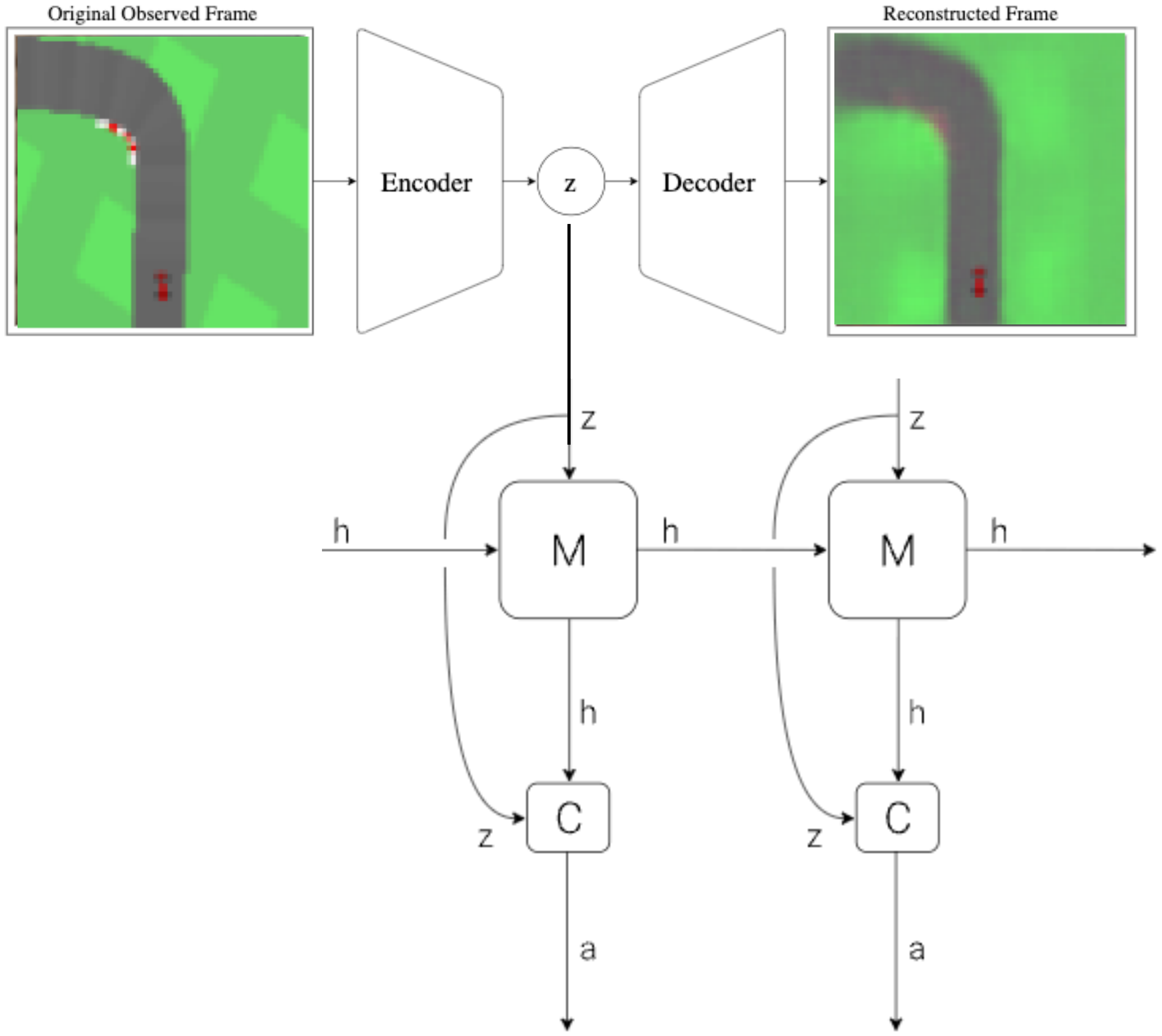
World models

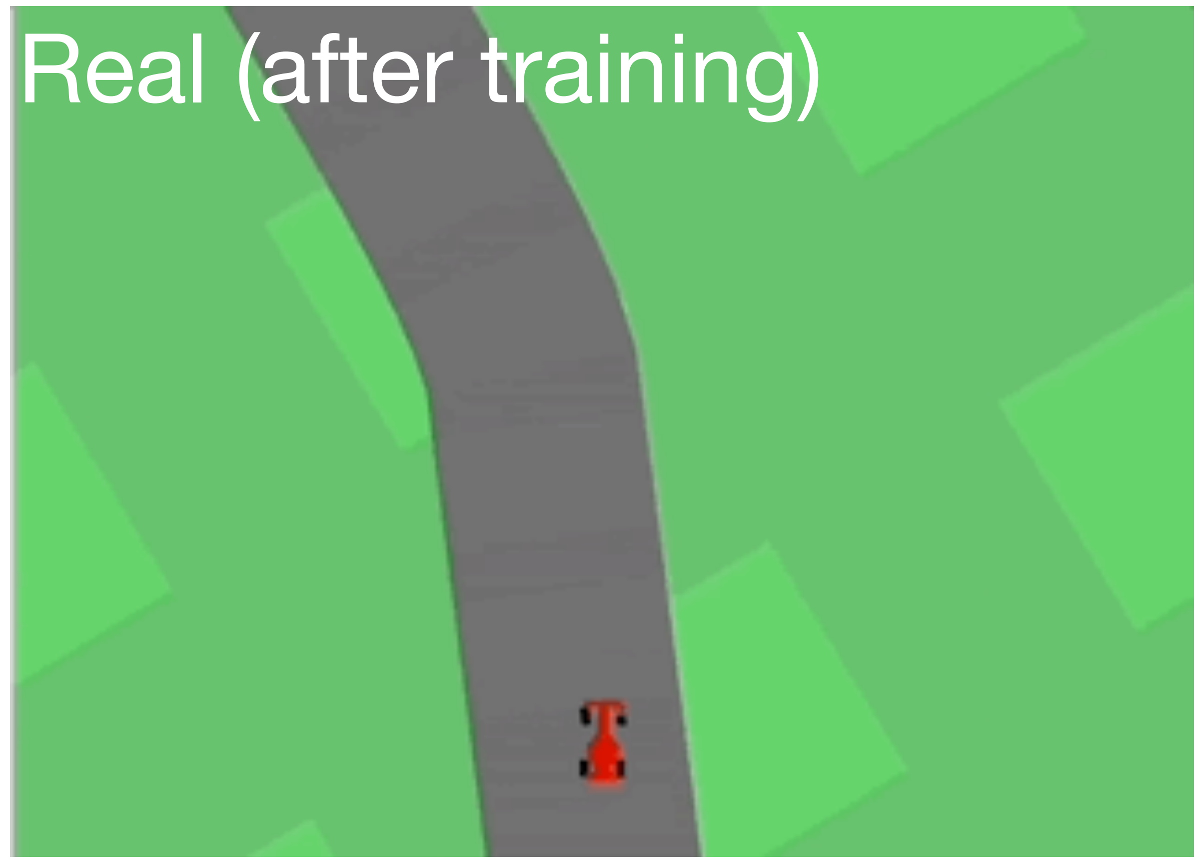


- **Vision Model (V)** encodes high-dimensional visual data into a low-dimensional latent vector z
- **Memory RNN (M)** learns the temporal dynamics and predicts future states $P(z_{t+1} \mid a_t, z_t, h_t)$ where h is a hidden state vector capturing dynamics
- **Linear controller (C)** selects actions as a linear function of z_t and h_t

  $a_t = W_c[z_t\, h_t] + b_c$ where W_c and b_c are weights/bias

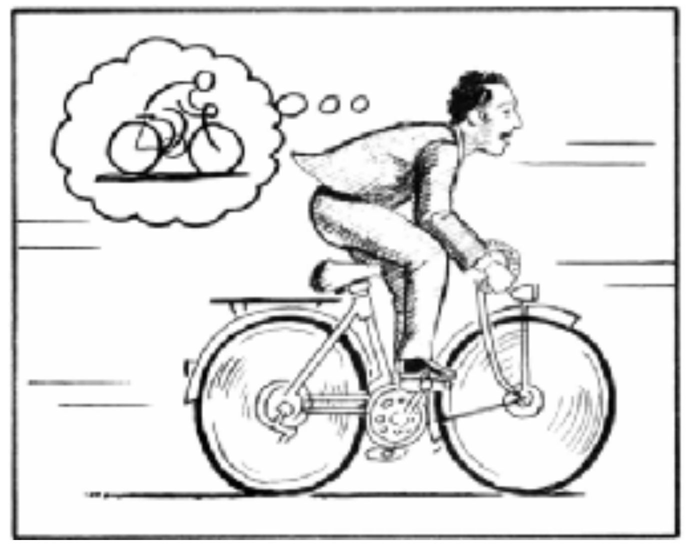
Variational Autoencoder (VAE)





Ha & Schmidhuber (2018)

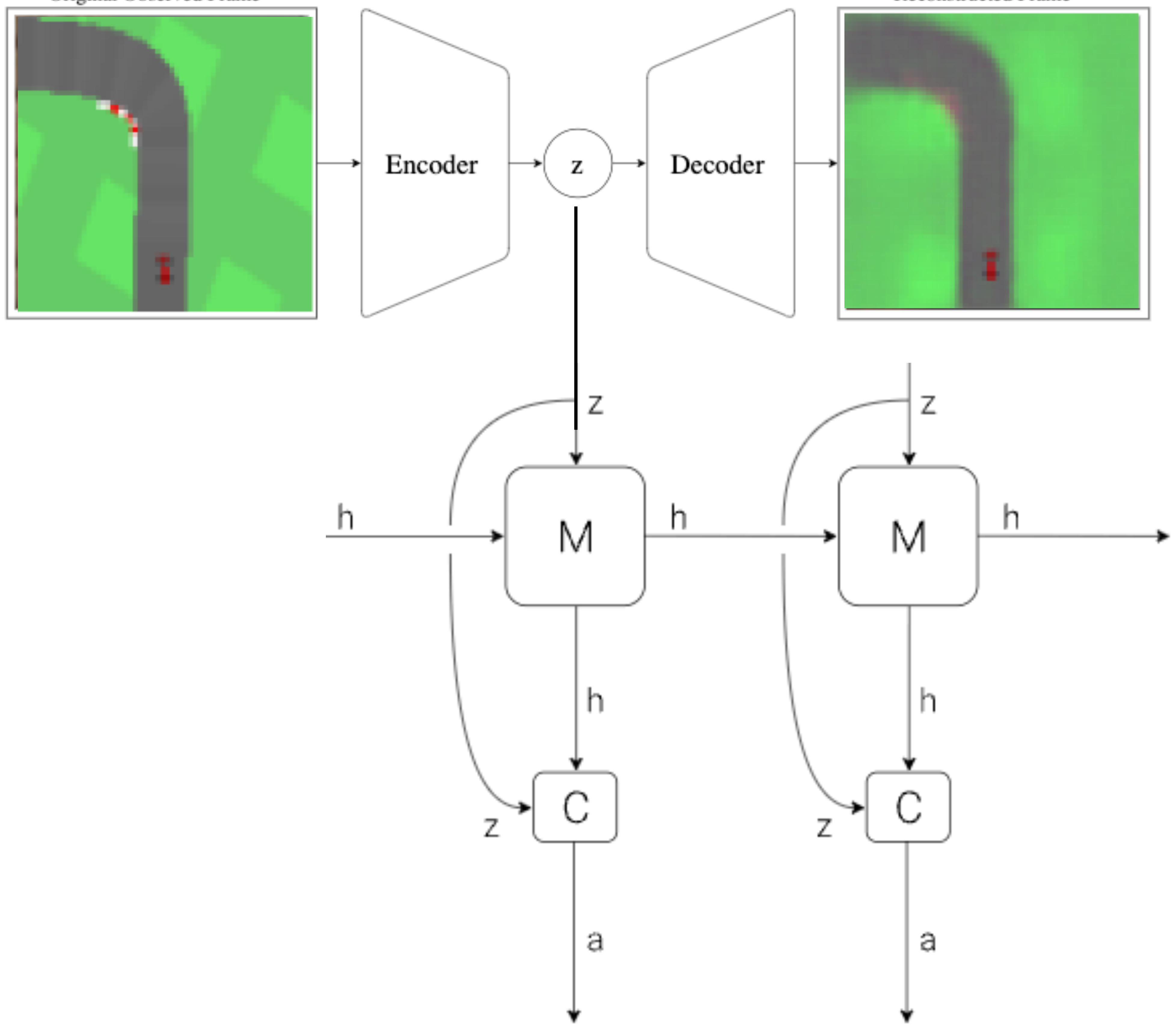
World models

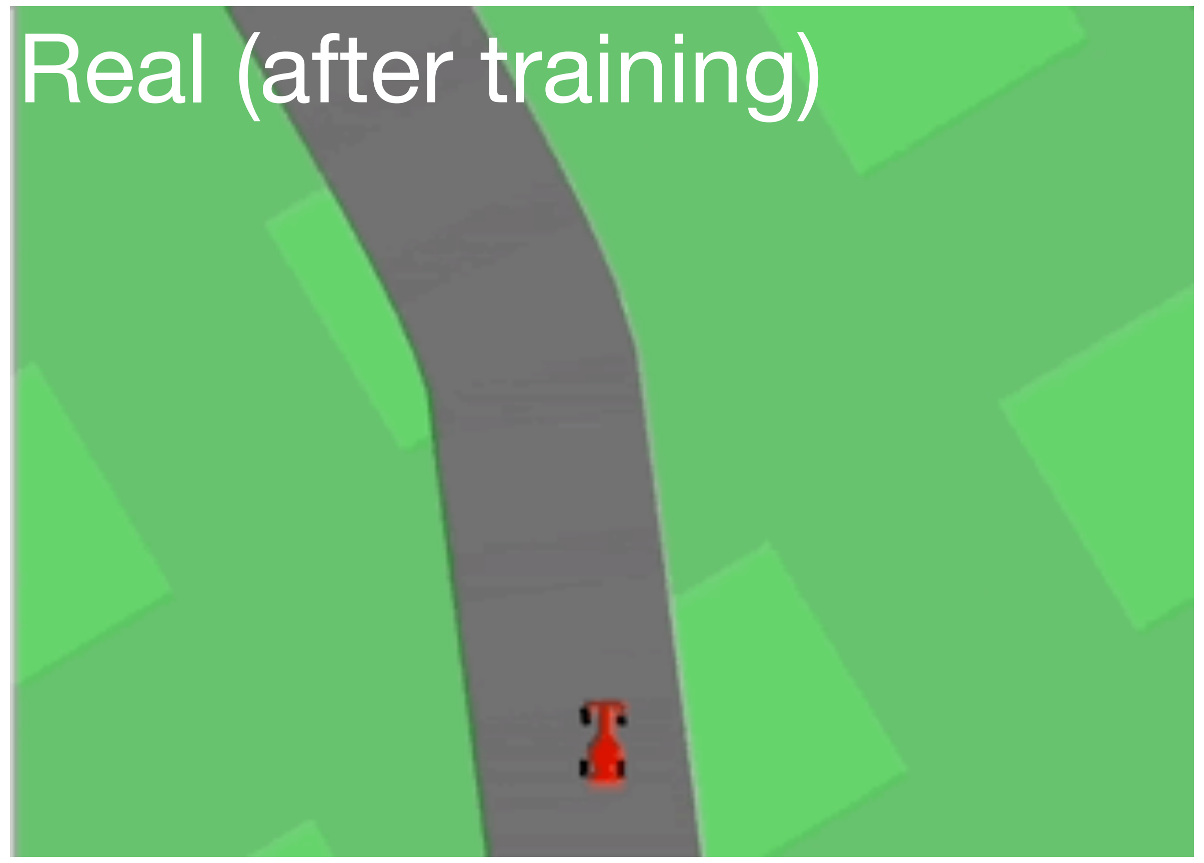


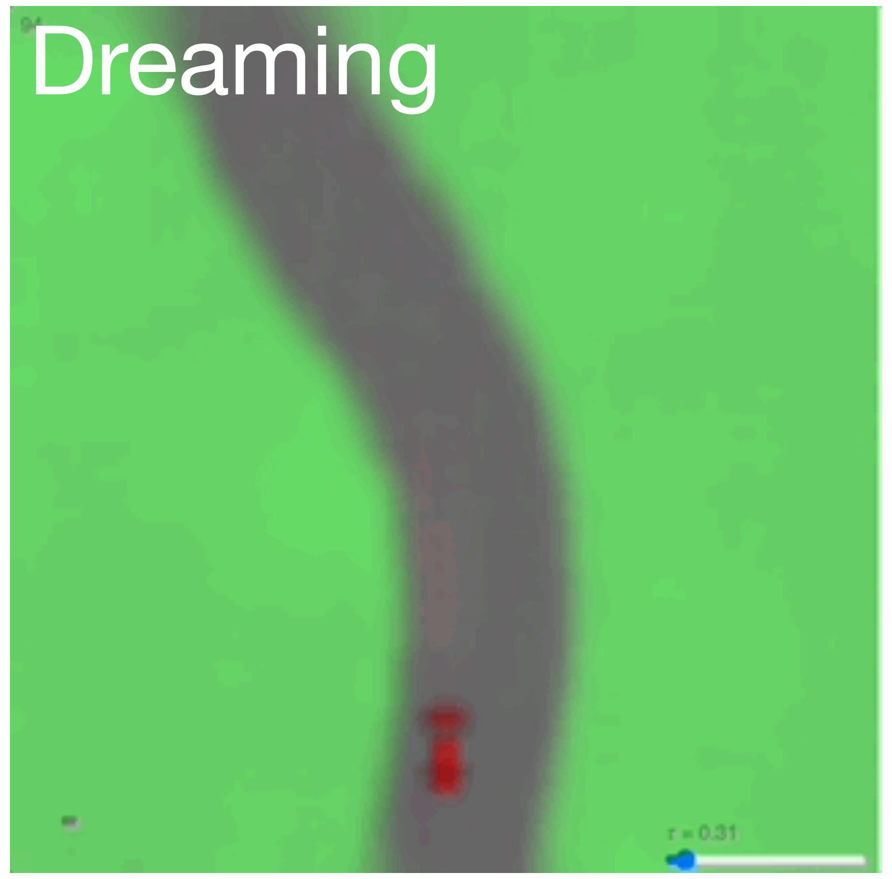
- **Vision Model (V)** encodes high-dimensional visual data into a low-dimensional latent vector z
- **Memory RNN (M)** learns the temporal dynamics and predicts future states $P(z_{t+1} \mid a_t, z_t, h_t)$ where h is a hidden state vector capturing dynamics
- **Linear controller (C)** selects actions as a linear function of z_t and h_t

  $a_t = W_c[z_t\, h_t] + b_c$ where W_c and b_c are weights/bias
- Training on dreams by simulating future states and treating them as real
  - Used to update controller weights

Variational Autoencoder (VAE)



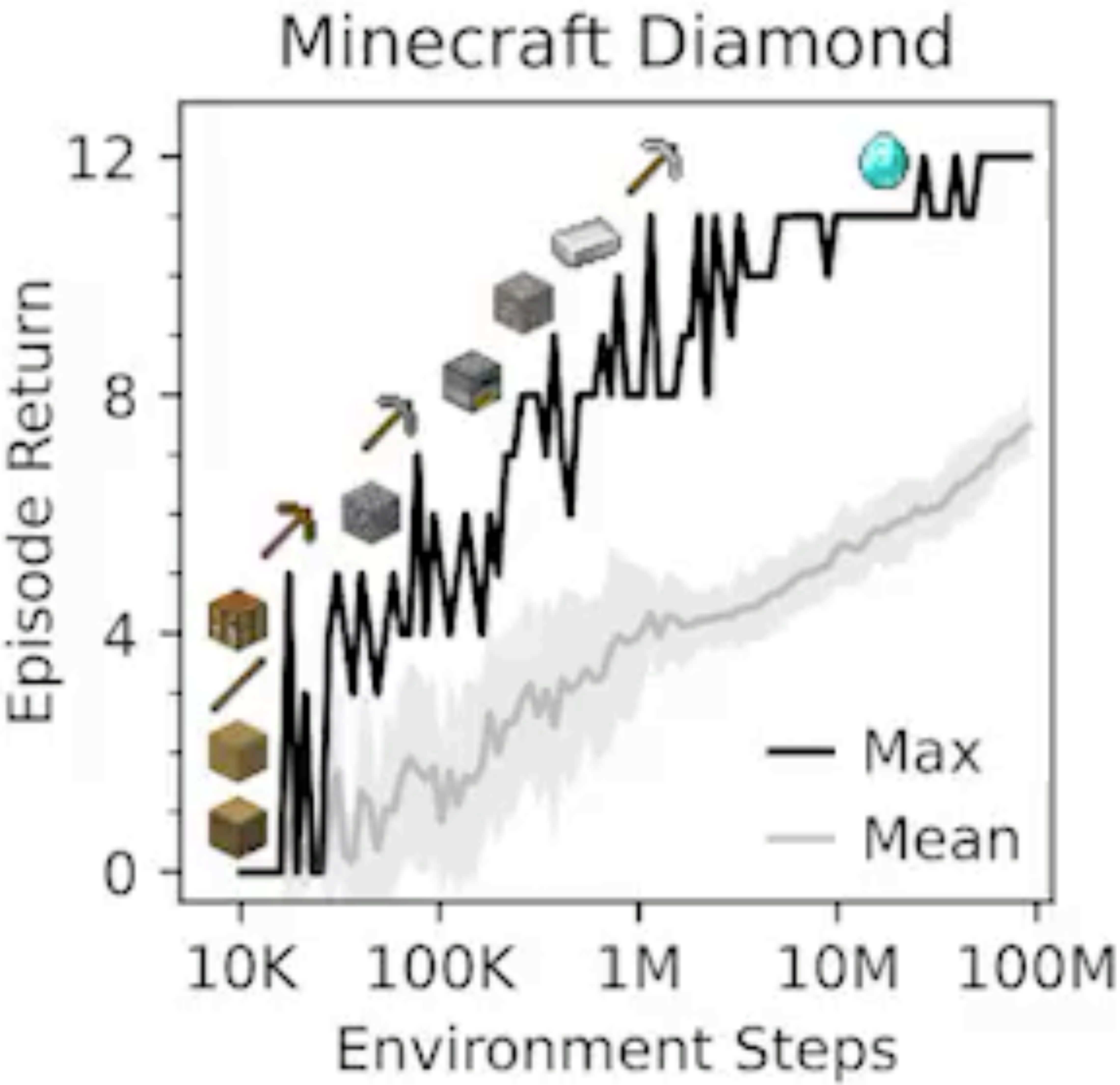


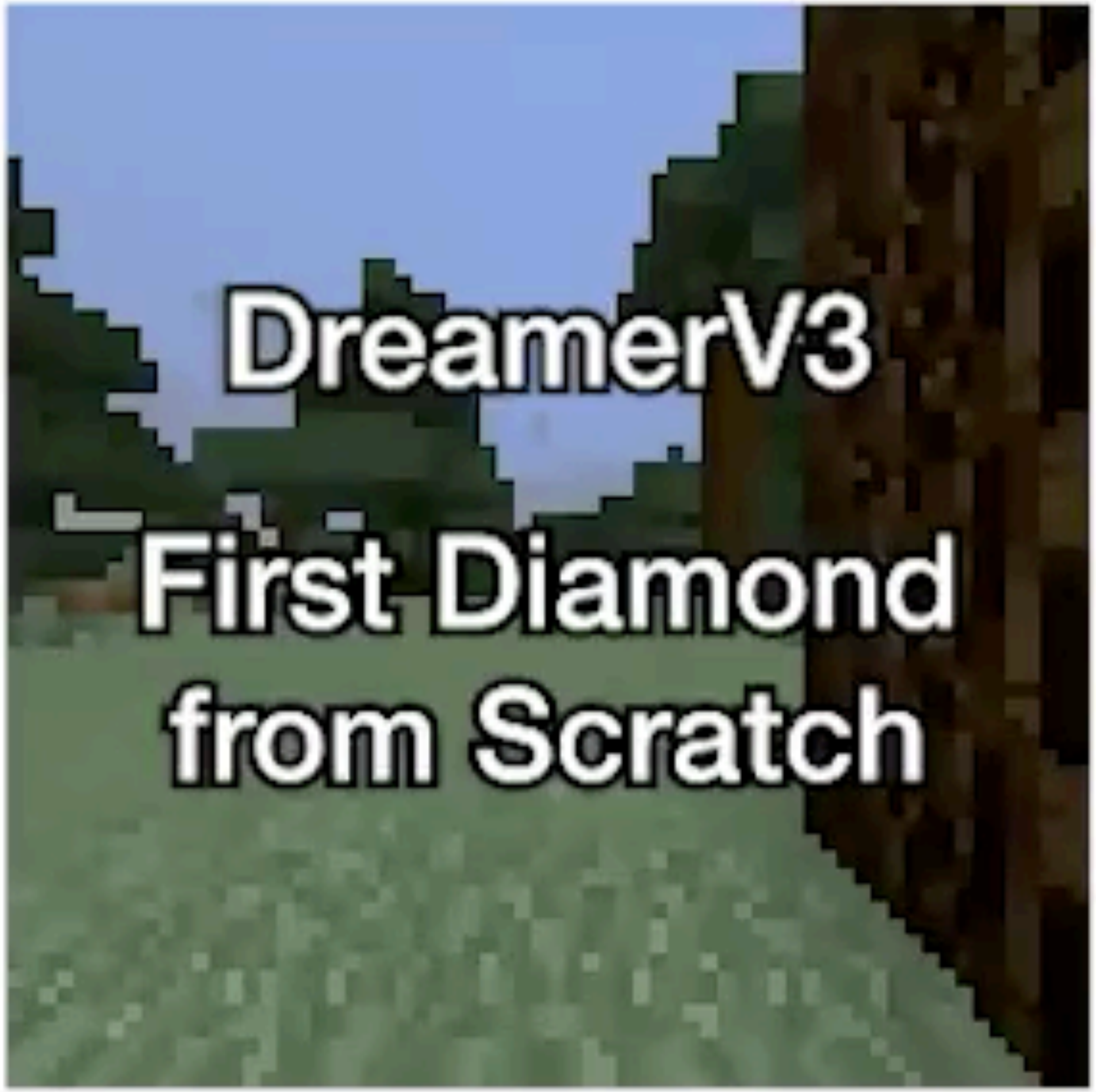



DreamerV3
Hafner et al., (2023)

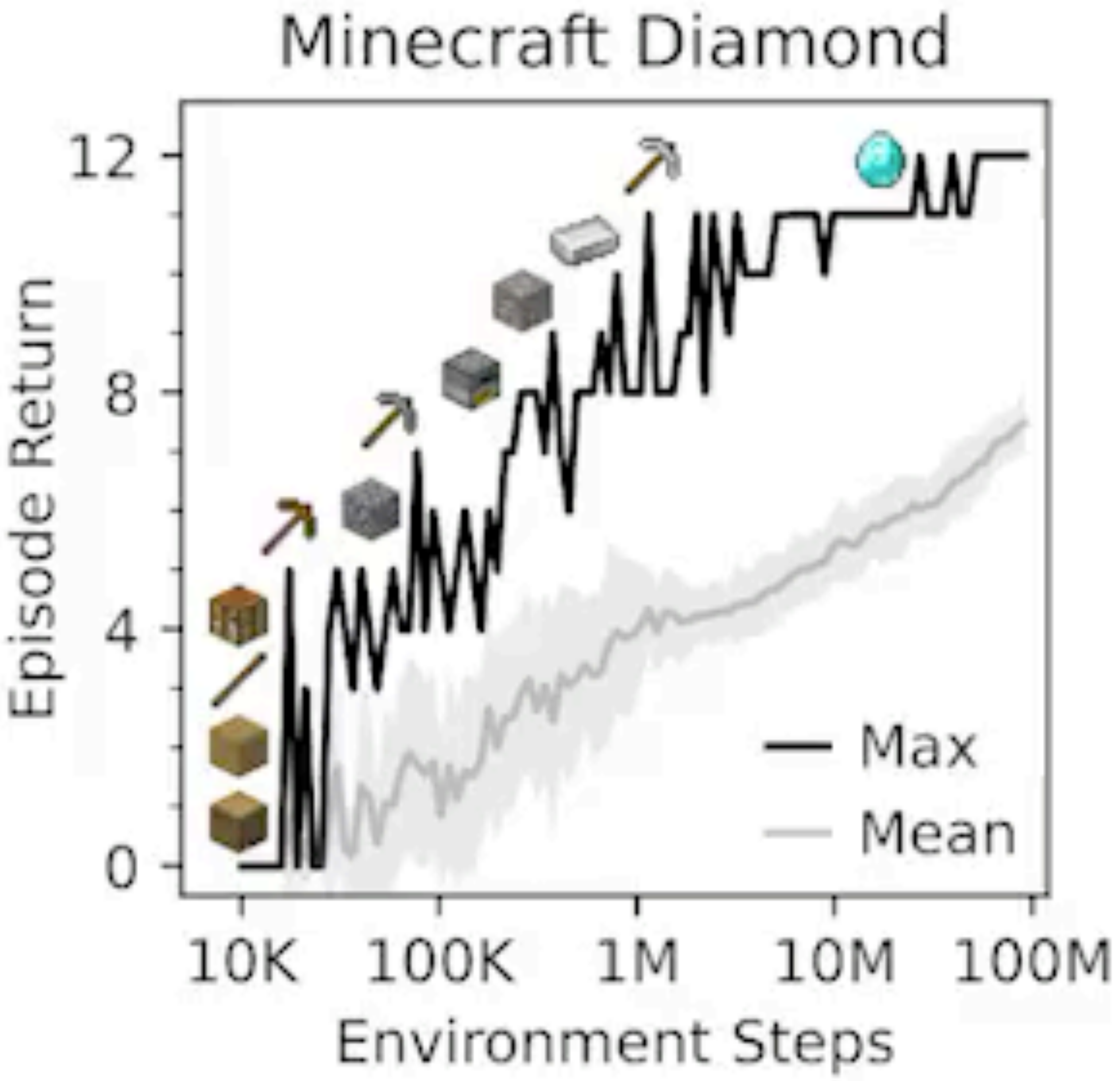

DreamerV3
Hafner et al., (2023)

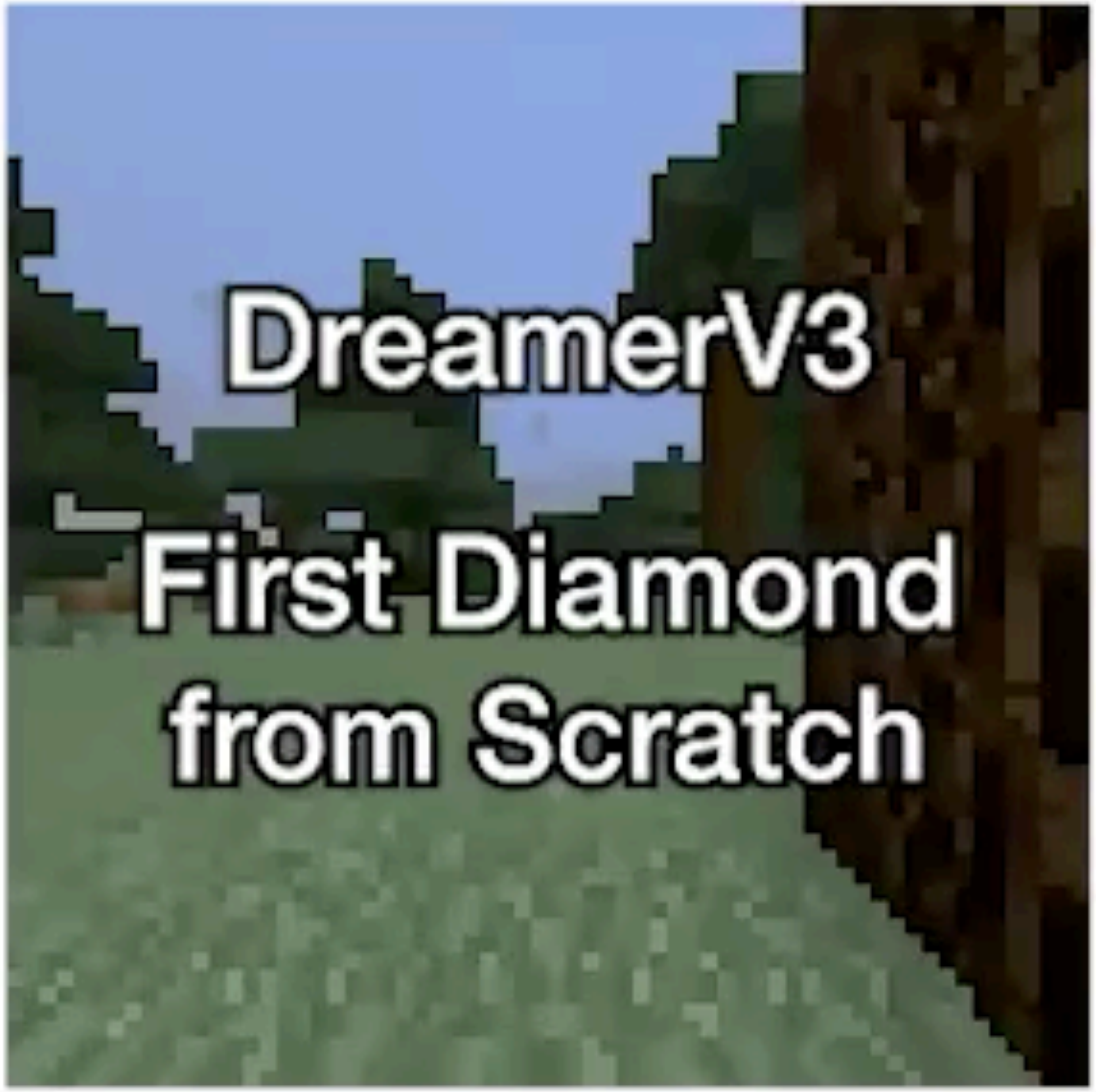
First algorithm to collect diamonds in Minecraft without human data or training curricula





First algorithm to collect diamonds in Minecraft without human data or training curricula





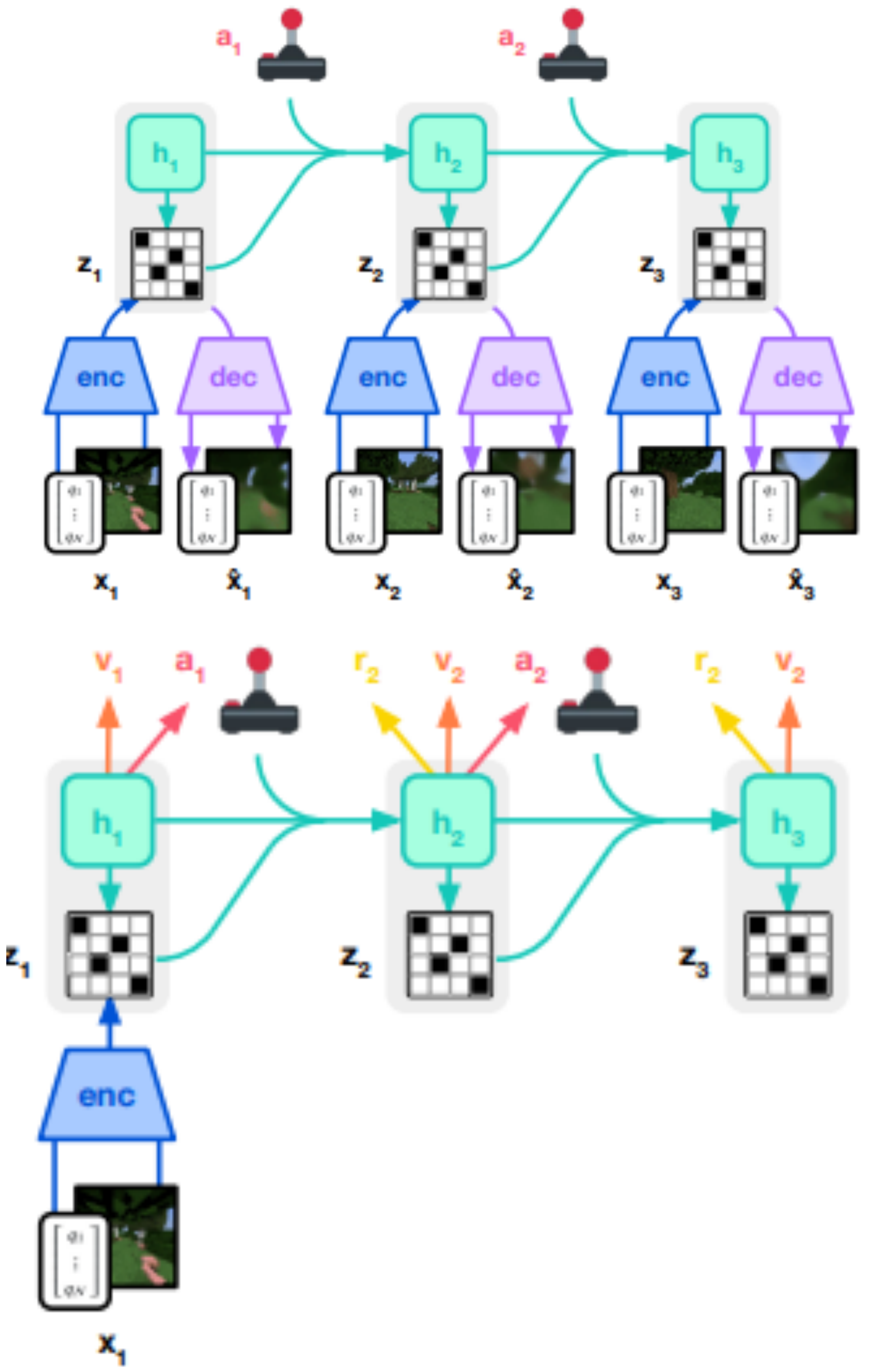
Dreamer Hafner et al., (2023)

Similar concept to World Models

- **Encoder** $z_t \sim q_\phi(z_t \mid h_t, x_t)$ given hidden state h_t and observations x_t
- **Sequence model** $h_{t+1} = f_\phi(h_t, z_t, a_t)$ and **dynamics predictor** $\hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t)$

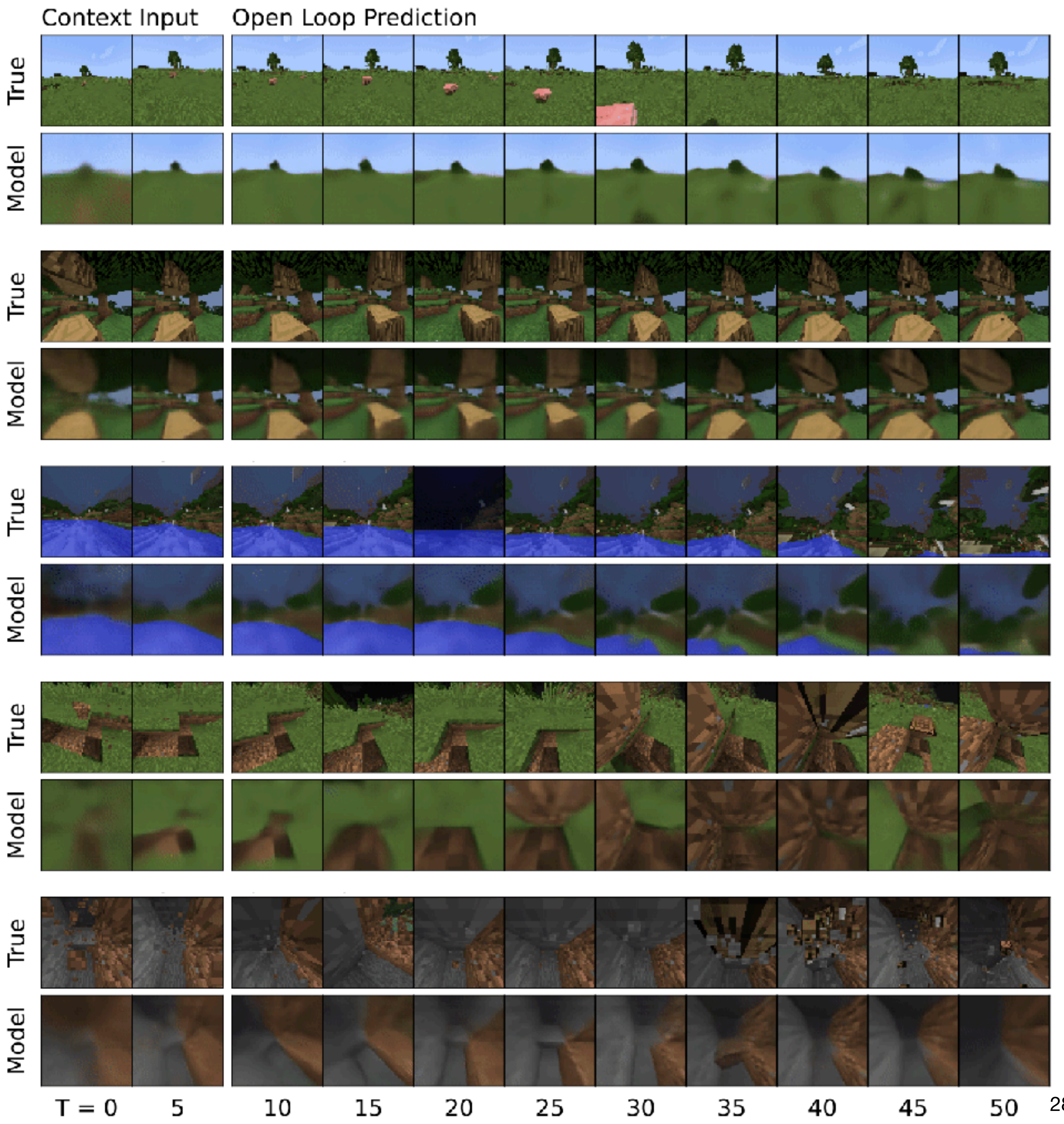
Actor-critic architecture

- **Actor** $a_t \sim \pi_\theta(a_t \mid s_t)$ where $s_t = \{h_t, z_t\}$
- **Critic** $v_\psi(R_t \mid s_t)$



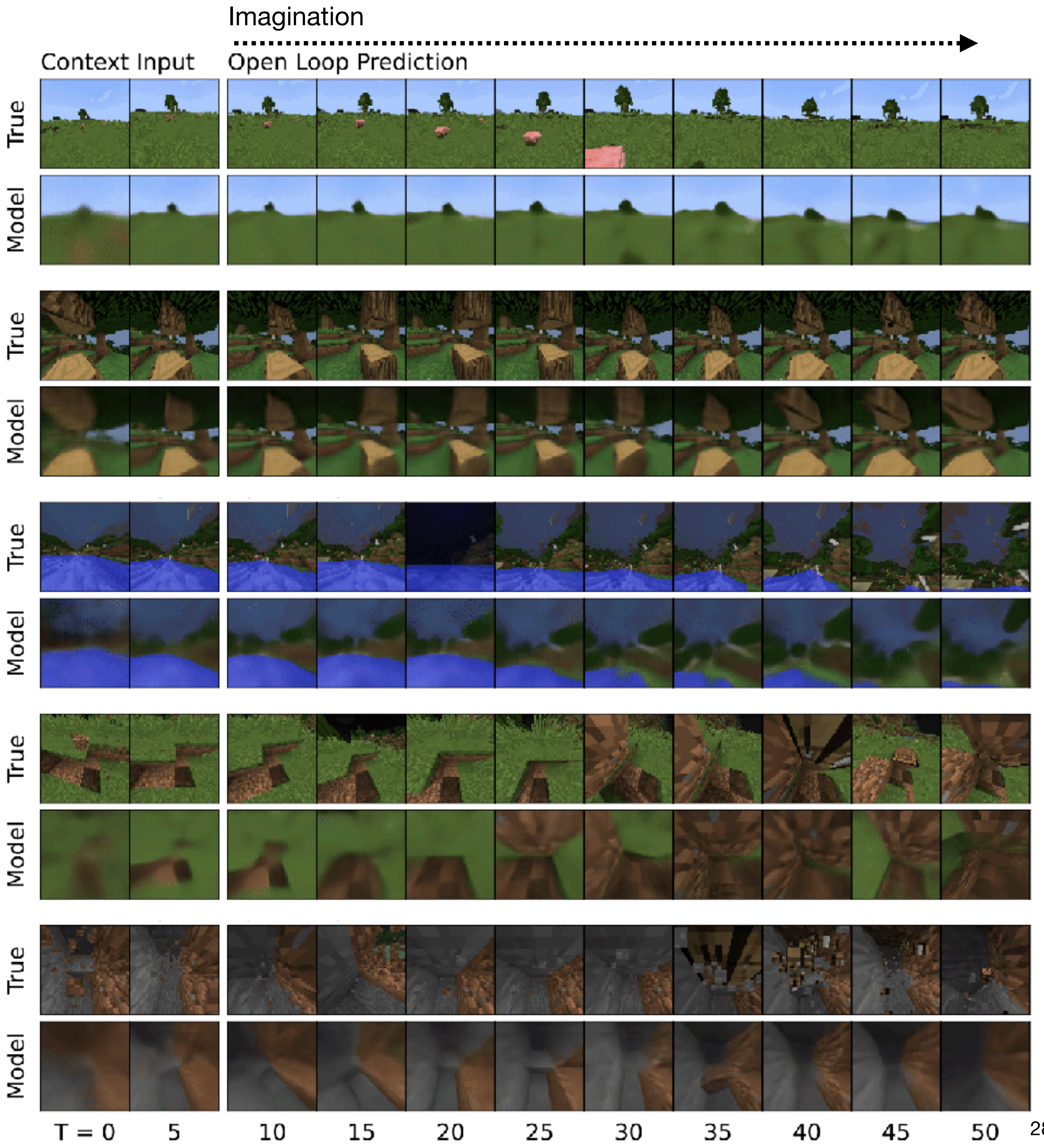
Model-based planning via simulating the future

- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences



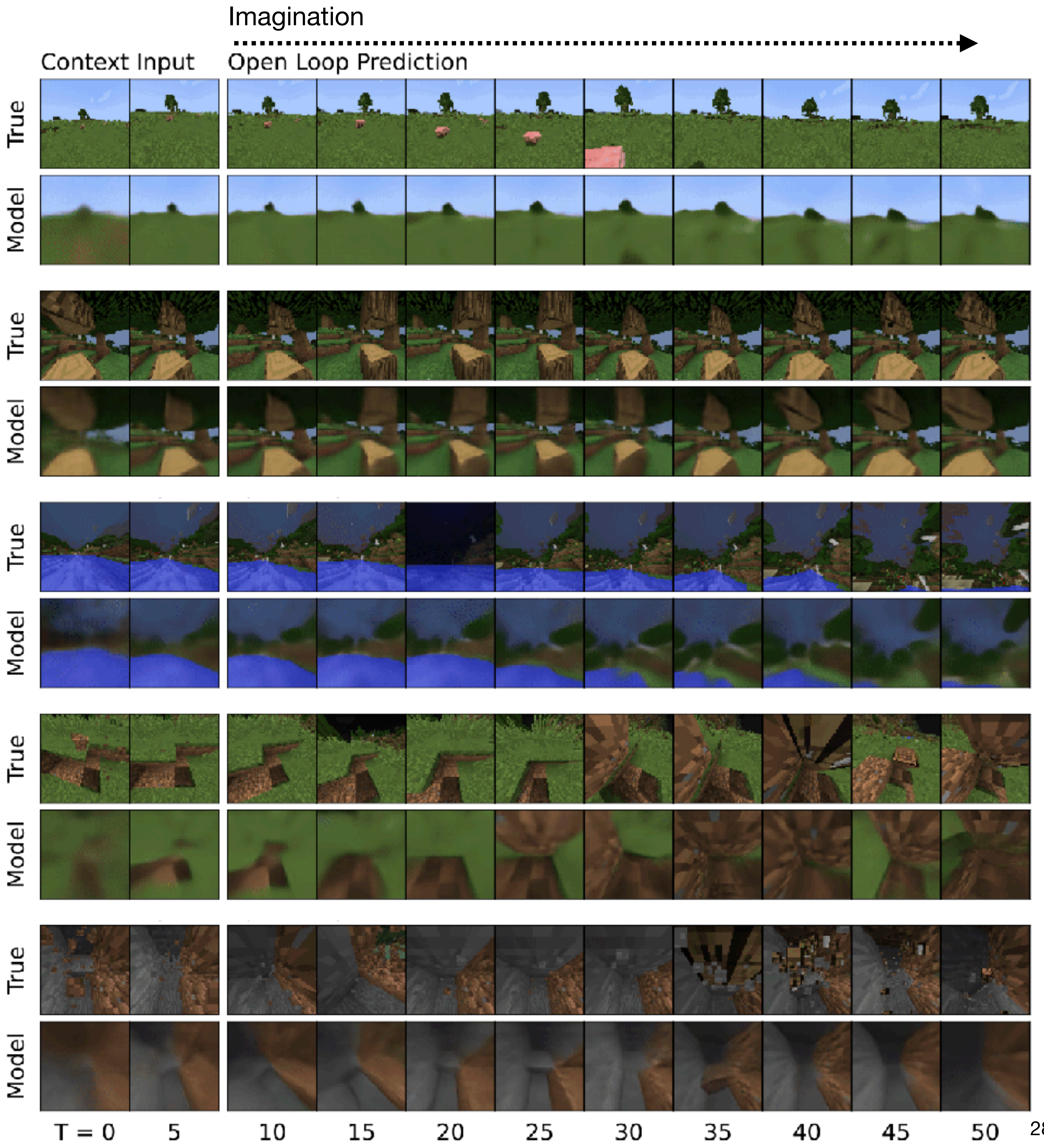
Model-based planning via simulating the future

- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences



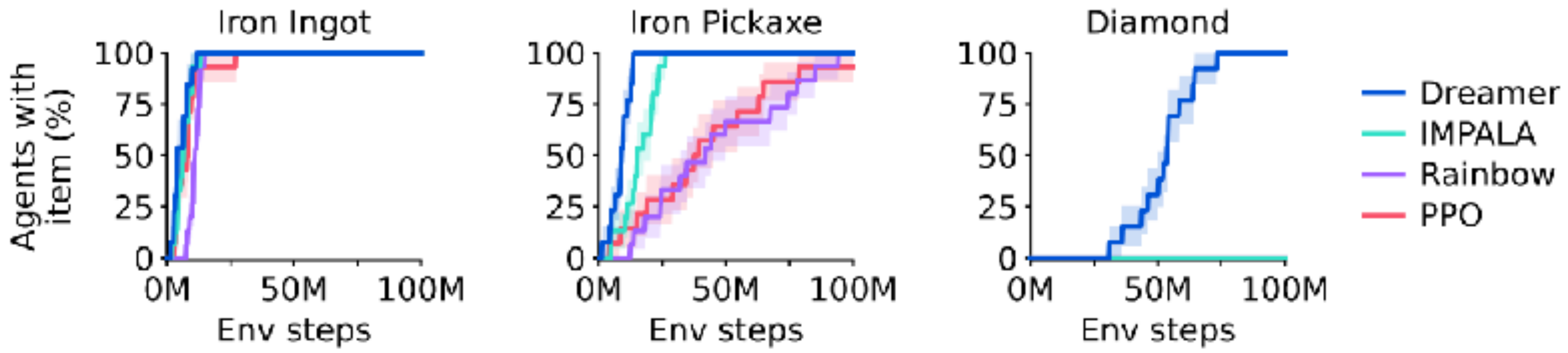
Model-based planning via simulating the future

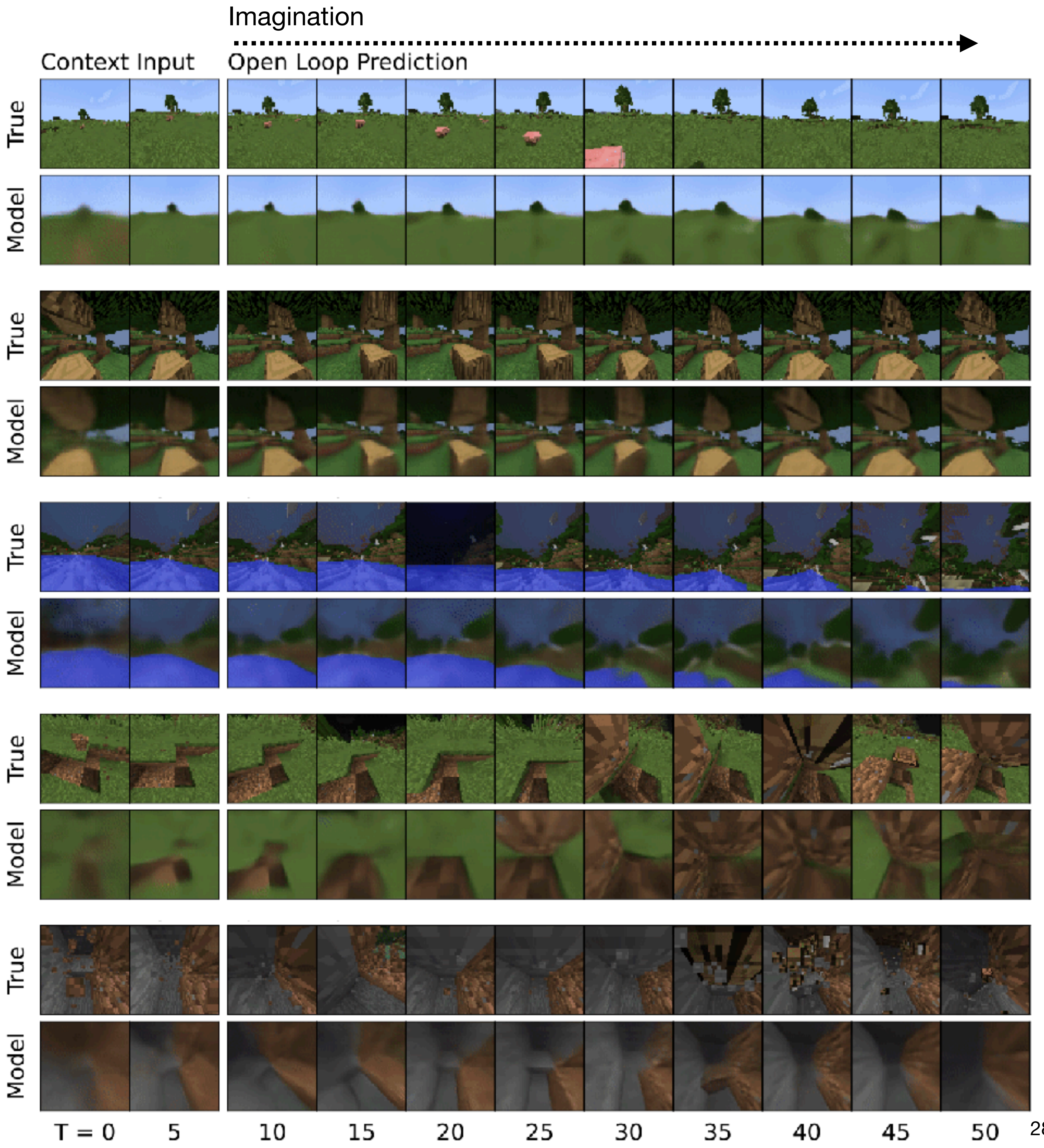
- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences
- It's ok that the model predictions don't perfectly match the true visual state of the world
    - only needs to be useful for training the actor and critic



Model-based planning via simulating the future

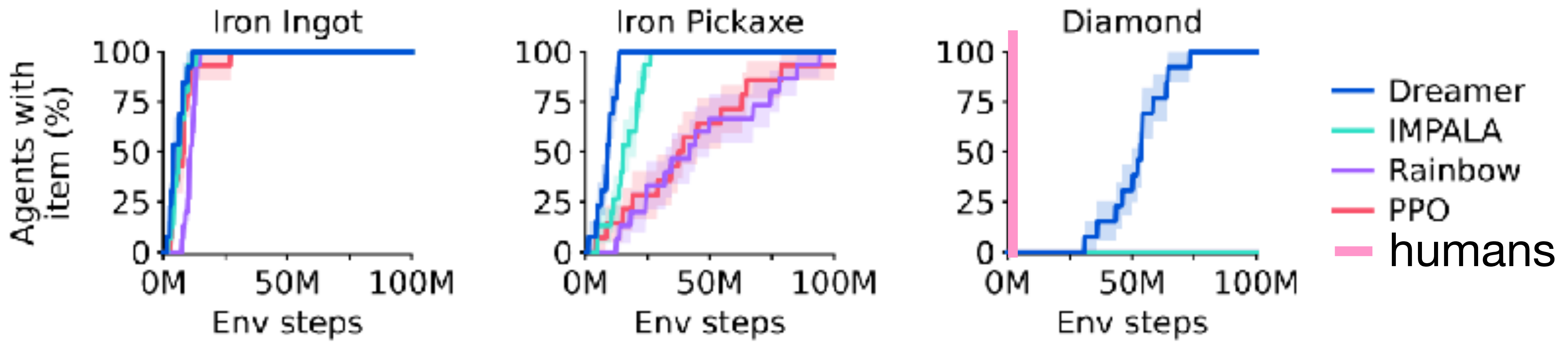
- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences
- It's ok that the model predictions don't perfectly match the true visual state of the world
  - only needs to be useful for training the actor and critic
- While similar performance for easier goals (iron ingot + pickaxe), first RL model to reliably find diamonds
  - But still very short of human performance

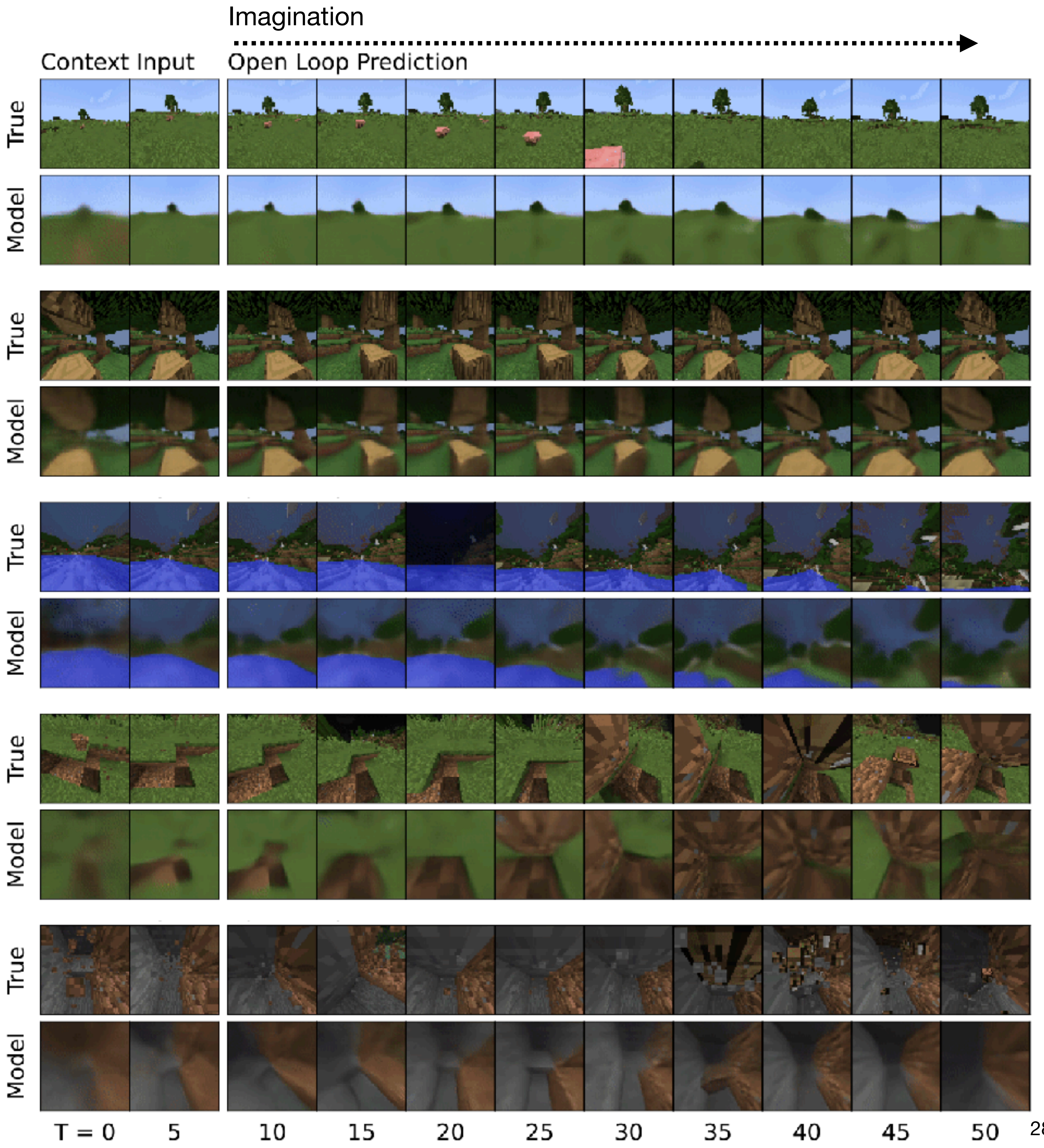




Model-based planning via simulating the future

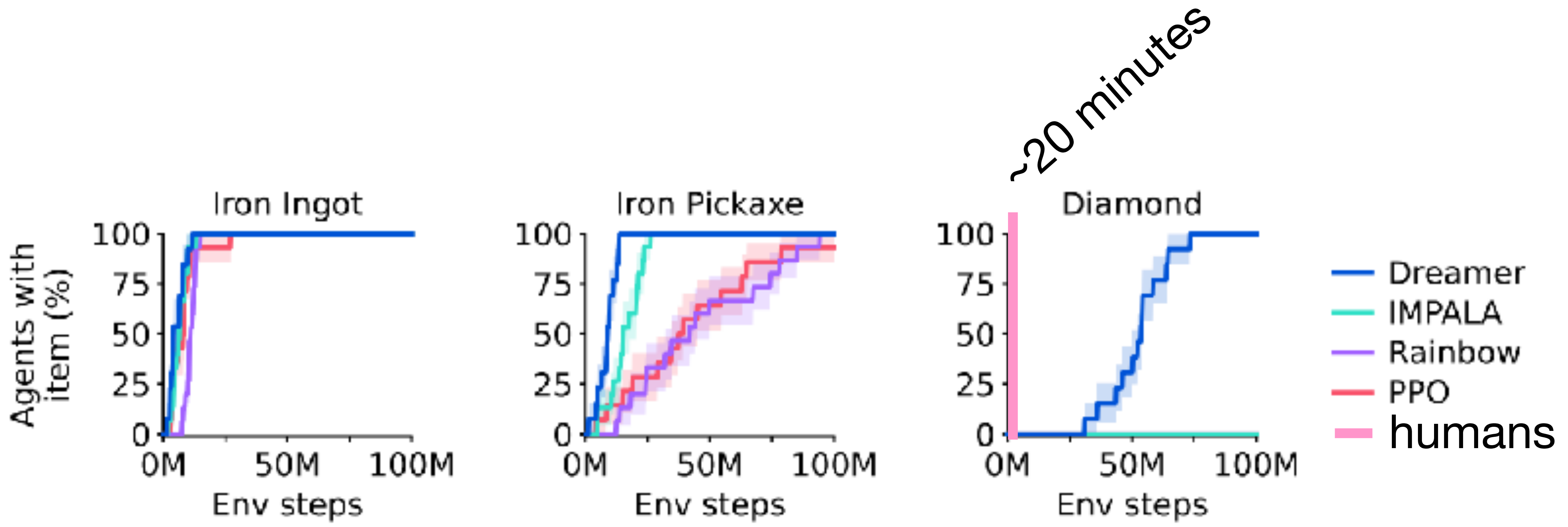
- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences
- It's ok that the model predictions don't perfectly match the true visual state of the world
  - only needs to be useful for training the actor and critic
- While similar performance for easier goals (iron ingot + pickaxe), first RL model to reliably find diamonds
  - But still very short of human performance

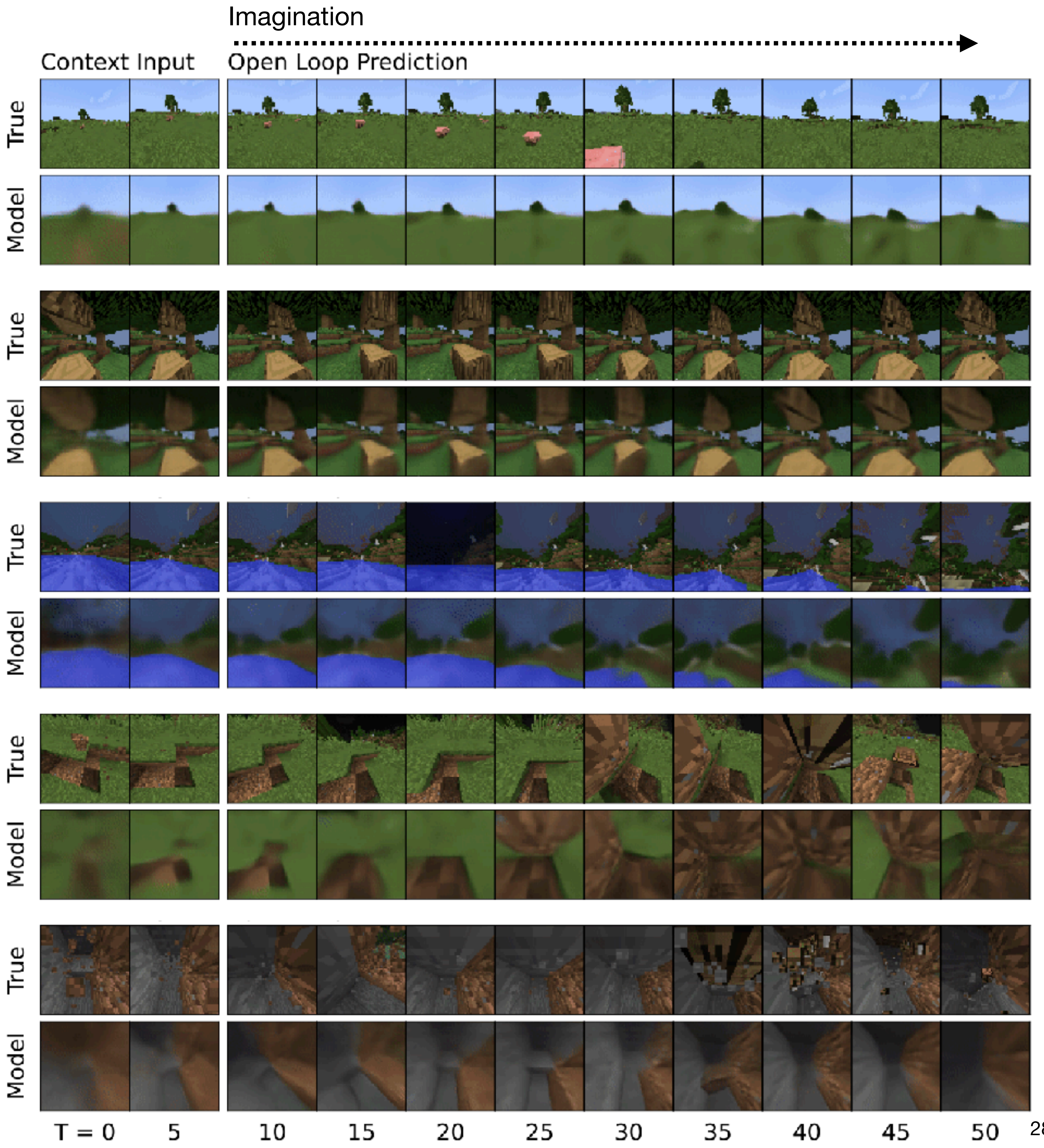




Model-based planning via simulating the future

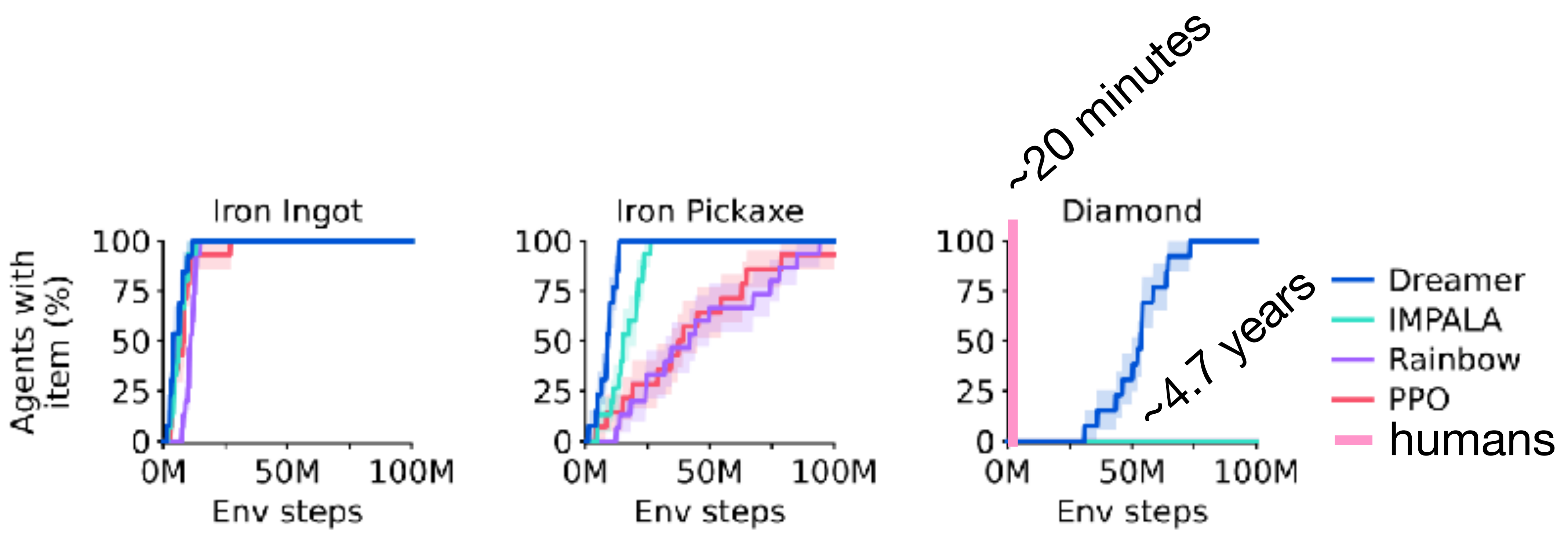
- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences
- It's ok that the model predictions don't perfectly match the true visual state of the world
  - only needs to be useful for training the actor and critic
- While similar performance for easier goals (iron ingot + pickaxe), first RL model to reliably find diamonds
  - But still very short of human performance

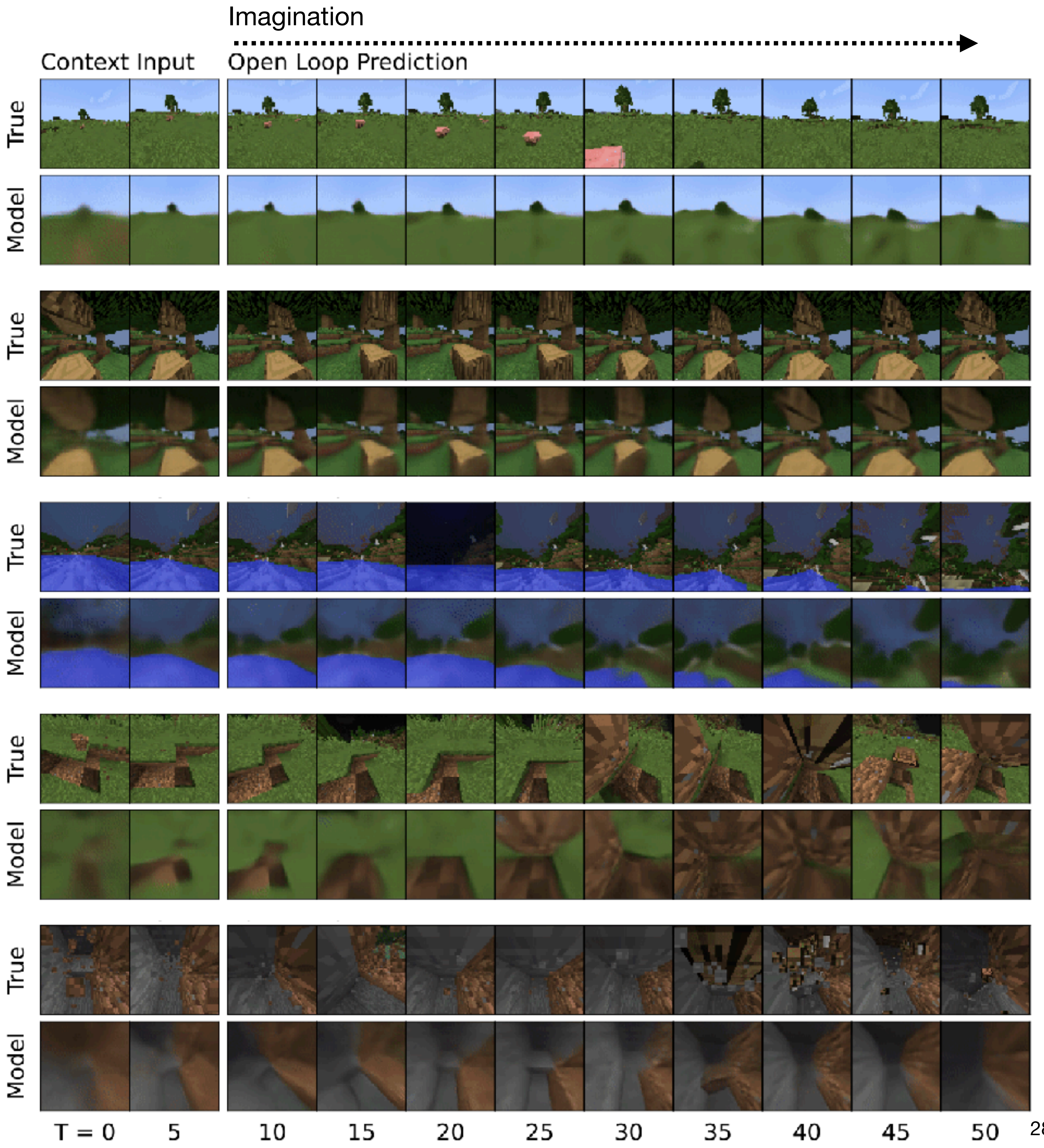




Model-based planning via simulating the future

- Main purpose of the model is to supplement real training experiences (direct RL) by simulating (imagining) future experiences
- It's ok that the model predictions don't perfectly match the true visual state of the world
    - only needs to be useful for training the actor and critic
- While similar performance for easier goals (iron ingot + pickaxe), first RL model to reliably find diamonds
    - But still very short of human performance





Model-based methods summary

How is the model learned?

- Through trial-and-error learning using delta-rule updates
- With modern ML techniques
  - Encode high-dimensional stimuli into a low-dimensional representation z
  - Learn the temporal dynamics $P(z_{t+1} | z_t)$

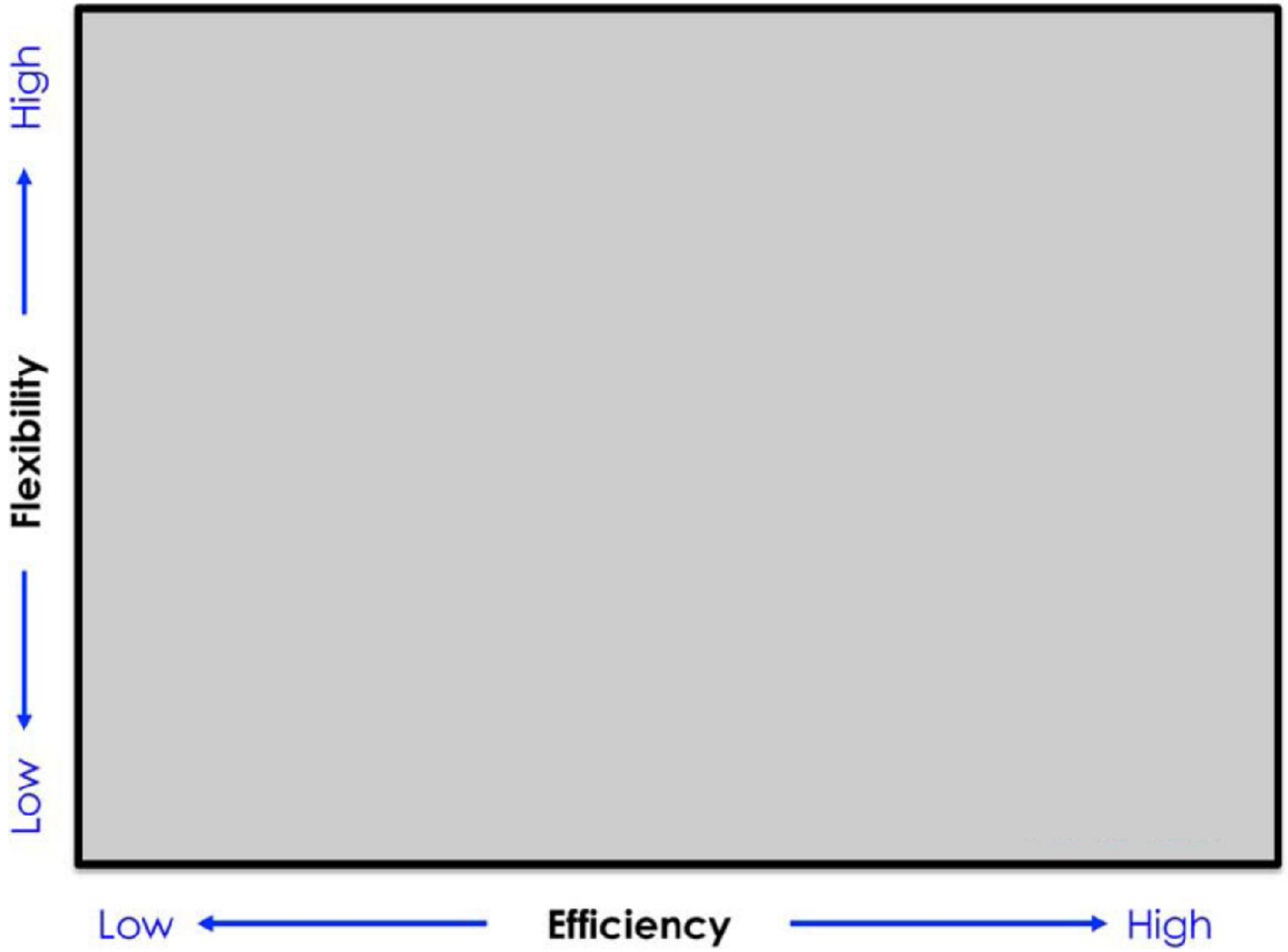
How is the model used?

- Use simulated experiences to augment direct RL (i.e., learning from real experiences)
- Model-free methods (e.g., actor-critic) can also be combined with model-based learning to great effect (Dreamer)

5 minute break

Balancing flexibility and efficiency

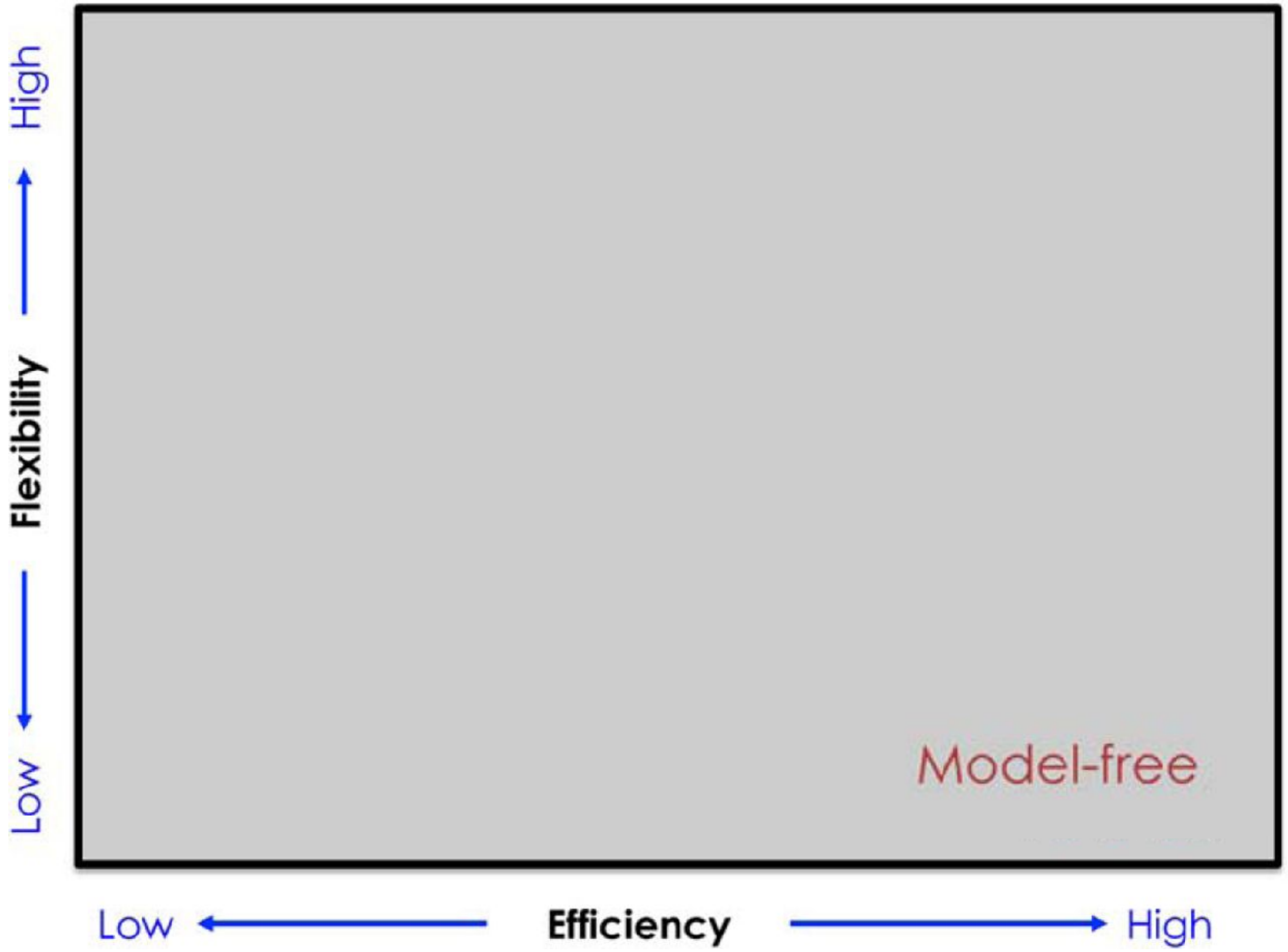
- Model-free methods are more **computationally efficient**
    - But lack flexibility to changes in the environment
- Model-based methods are highly flexible (local changes in environment lead to local changes in model)
    - But **computationally costly when it comes to performing simulations**
- Is there nothing in between?



Gershman (2018)

Balancing flexibility and efficiency

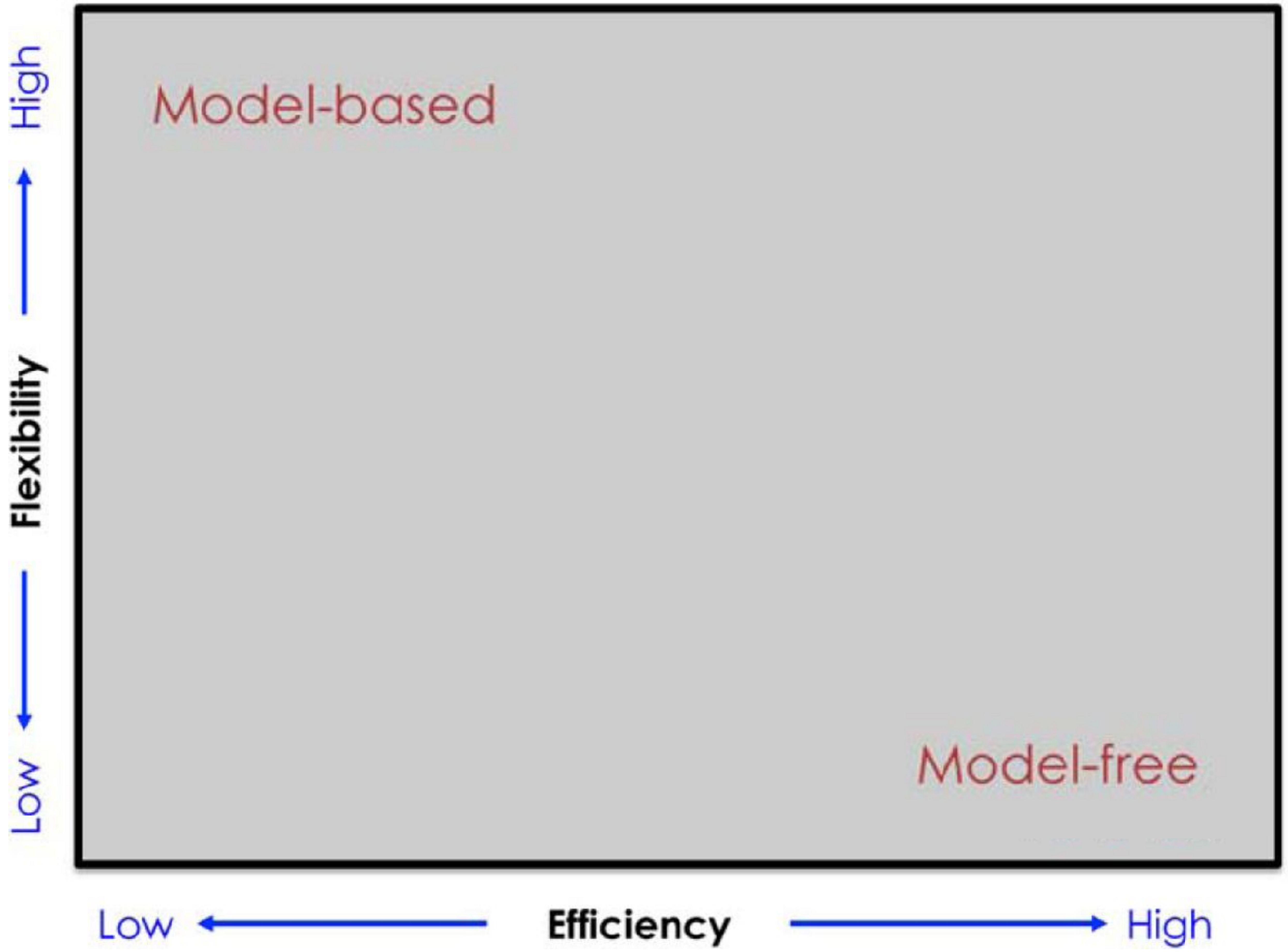
- Model-free methods are more **computationally efficient**
  - But lack flexibility to changes in the environment
- Model-based methods are highly flexible (local changes in environment lead to local changes in model)
  - But **computationally costly when it comes to performing simulations**
- Is there nothing in between?



Gershman (2018)

Balancing flexibility and efficiency

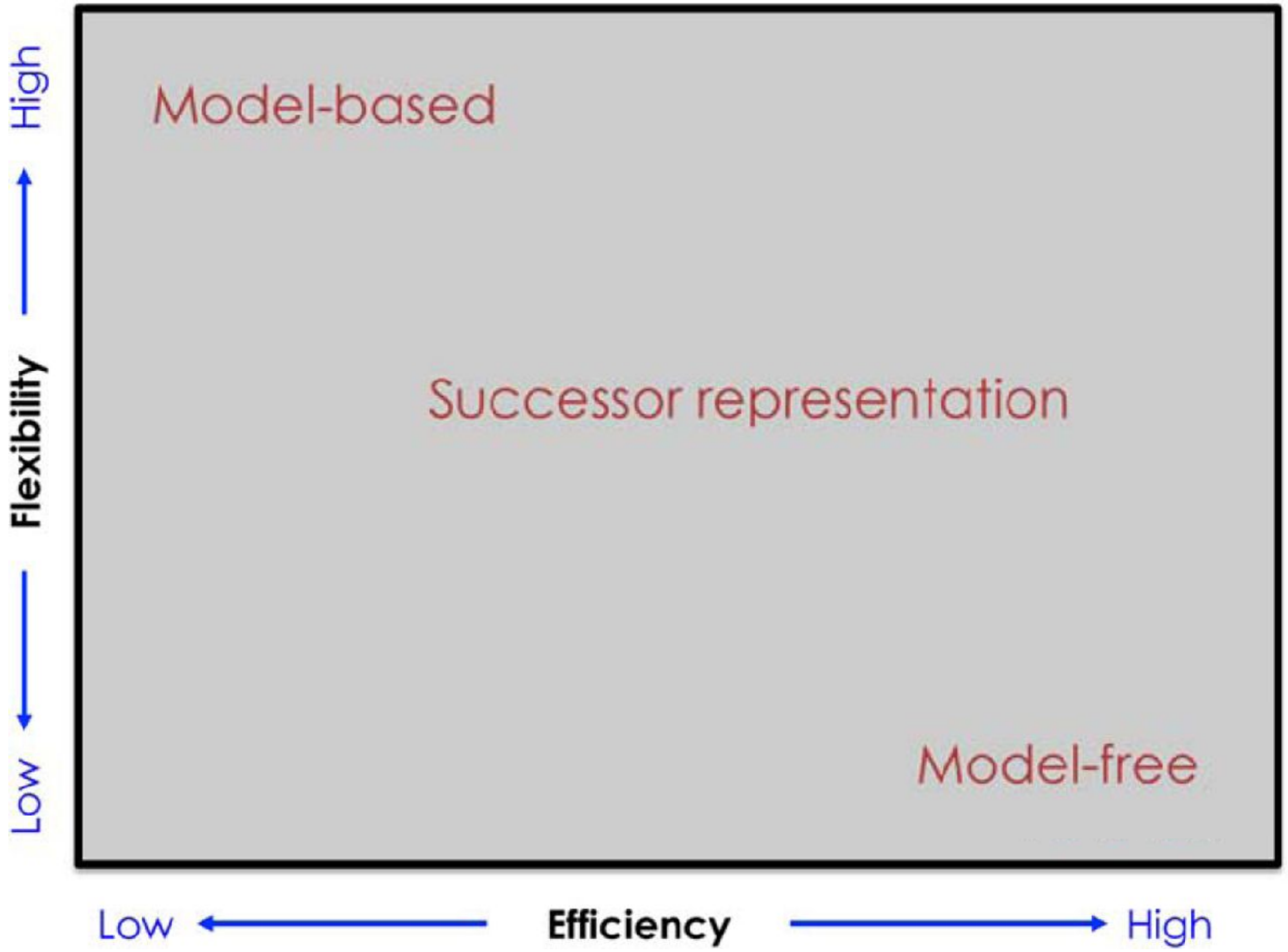
- Model-free methods are more **computationally efficient**
    - But lack flexibility to changes in the environment
- Model-based methods are highly flexible (local changes in environment lead to local changes in model)
    - But **computationally costly when it comes to performing simulations**
- Is there nothing in between?



Gershman (2018)

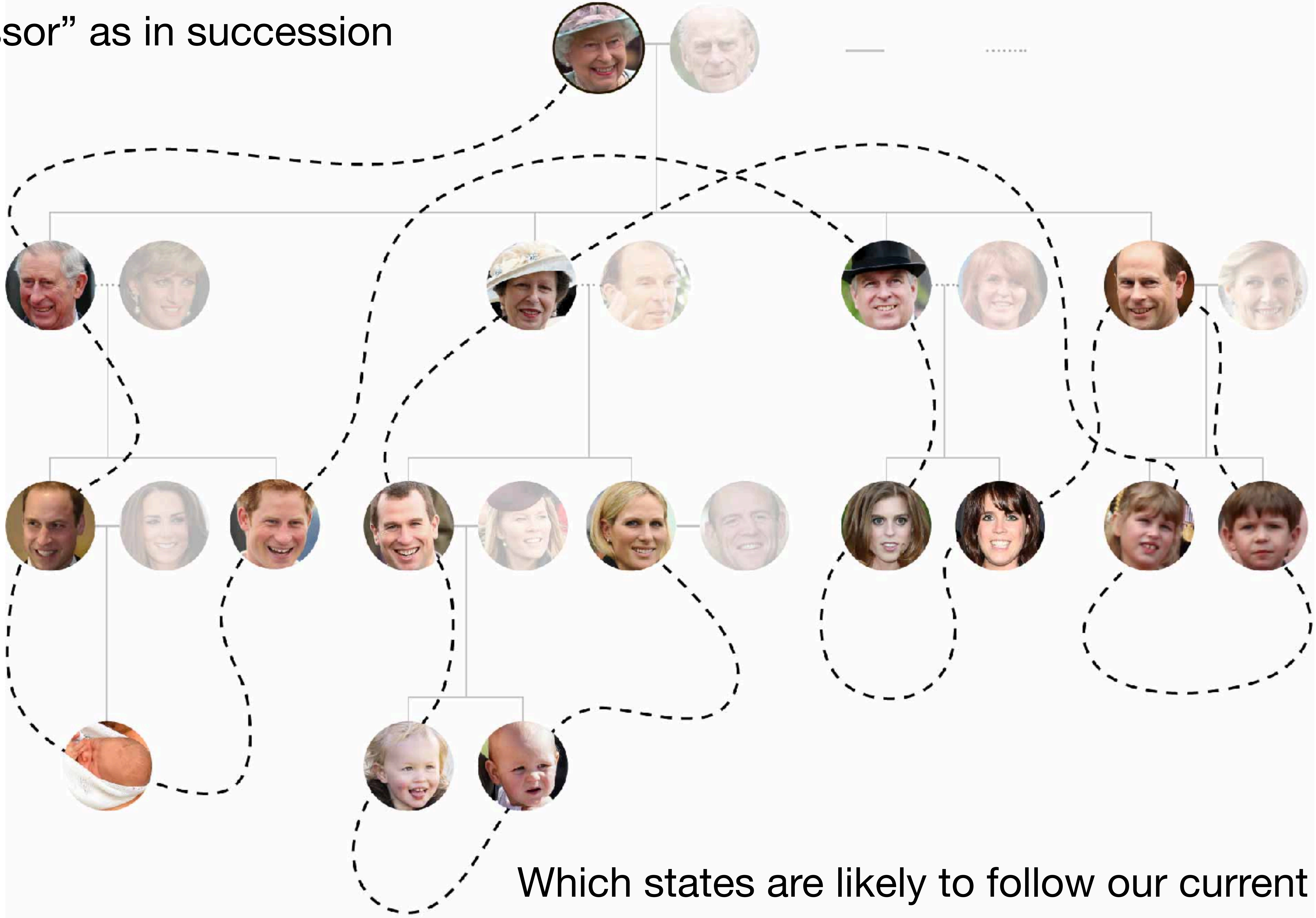
Balancing flexibility and efficiency

- Model-free methods are more **computationally efficient**
    - But lack flexibility to changes in the environment
- Model-based methods are highly flexible (local changes in environment lead to local changes in model)
    - But **computationally costly when it comes to performing simulations**
- Is there nothing in between?



Gershman (2018)

“Successor” as in succession



Which states are likely to follow our current state?

SR as a decomposition of the TD value function

Dayan (1993)

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}r_{t}\right]$$

Value function from TD Learning

SR as a decomposition of the TD value function

Dayan (1993)

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Value function from TD Learning

$$V^{\pi}(s) = \sum_{s'} M(s, s') r(s')$$

SR decomposition

SR as a decomposition of the TD value function

Dayan (1993)

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

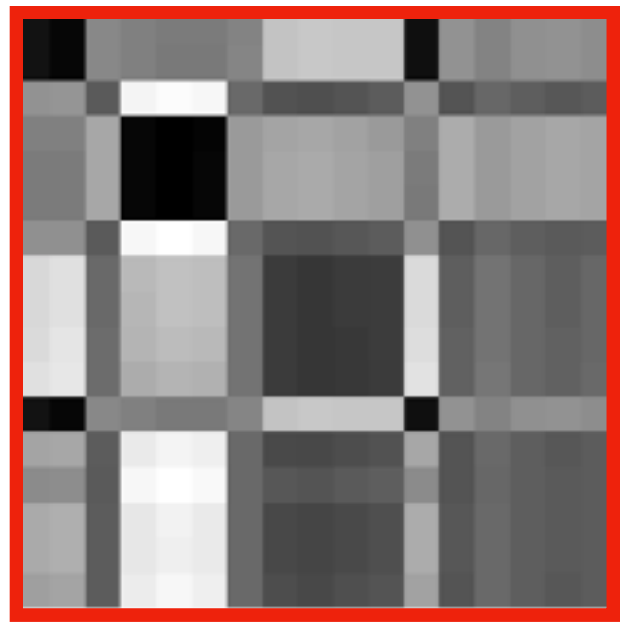
Value function from TD Learning

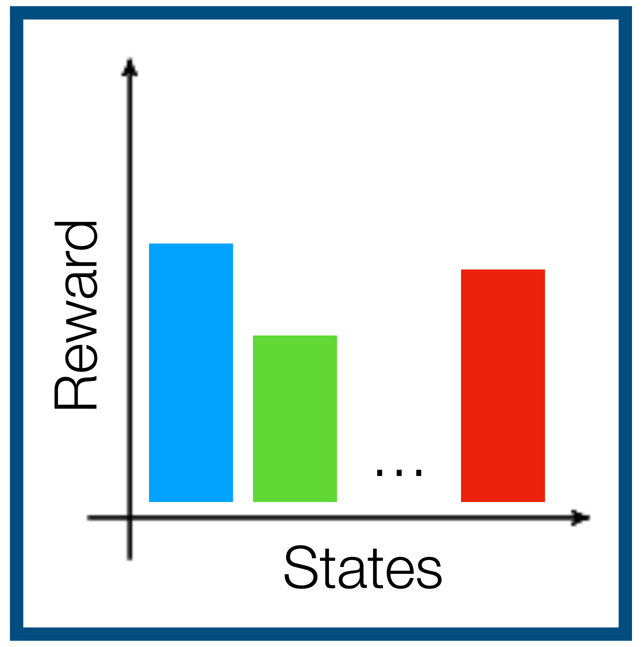
$$V^{\pi}(s) = \sum_{s'} M(s,s')r(s')$$

SR decomposition

Successor Representation

S by S matrix of future discounted state occupancies



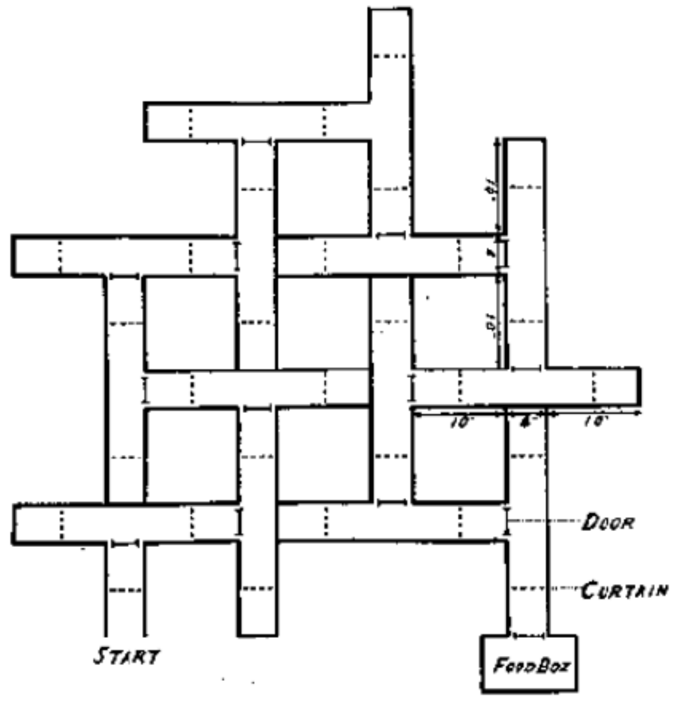


Reward Values

vector of singular rewards for each state

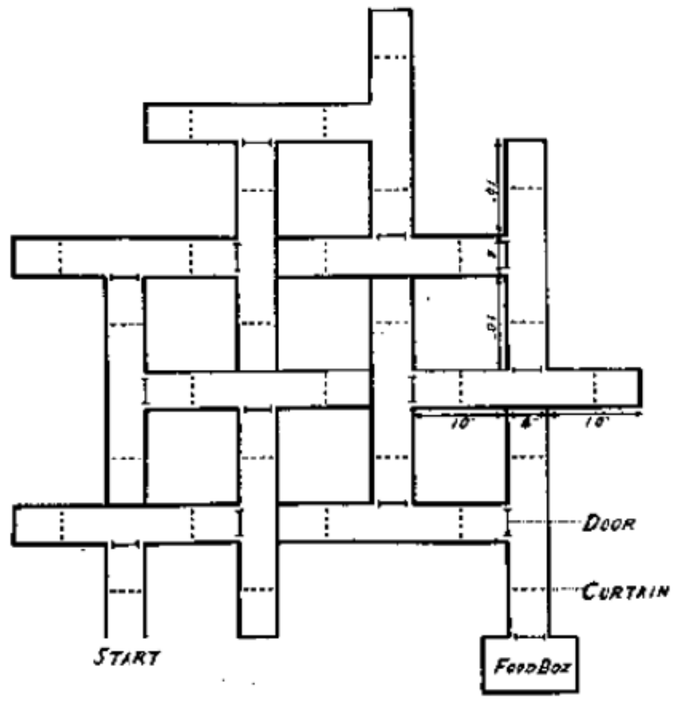
Successor Representation

Not just a map…

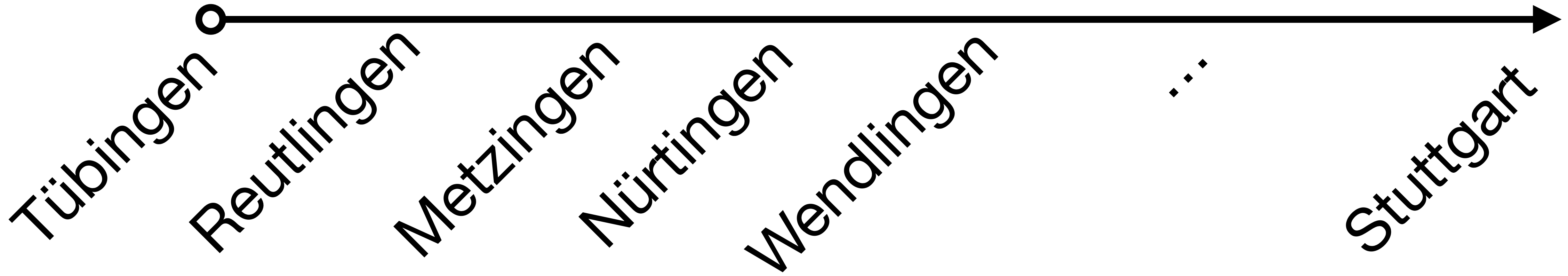


Successor Representation

Not just a map…

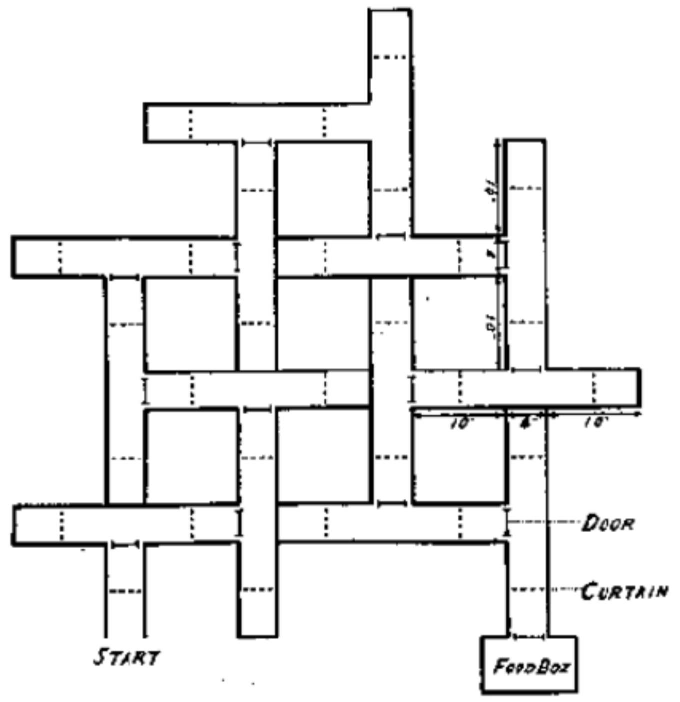


… but a goal-directed representation about which states are likely to encountered given a policy

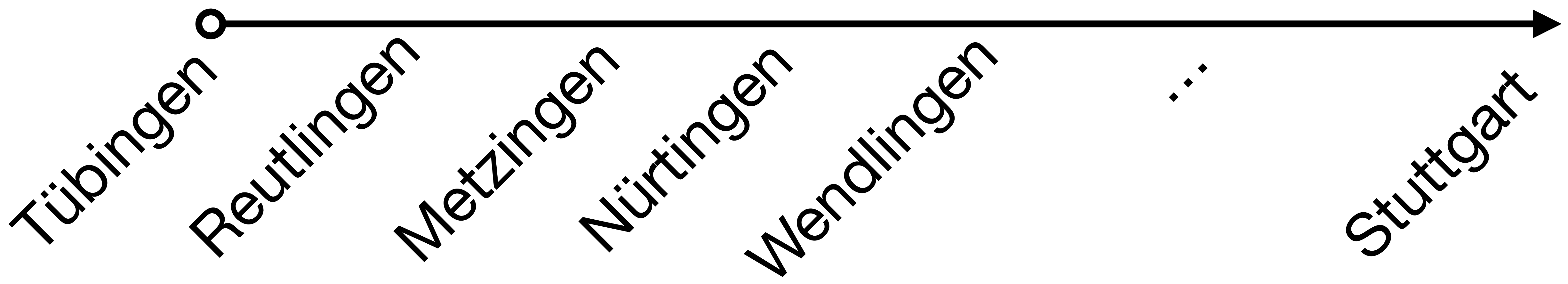


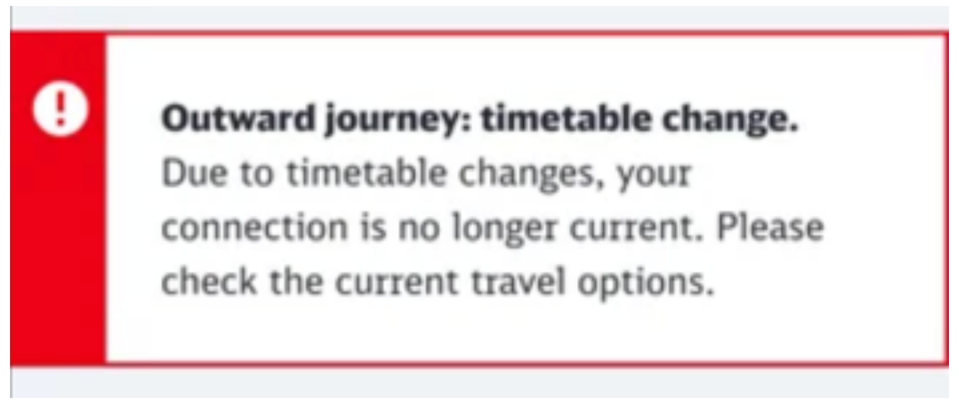
Successor Representation

Not just a map…



… but a goal-directed representation about which states are likely to encountered given a policy

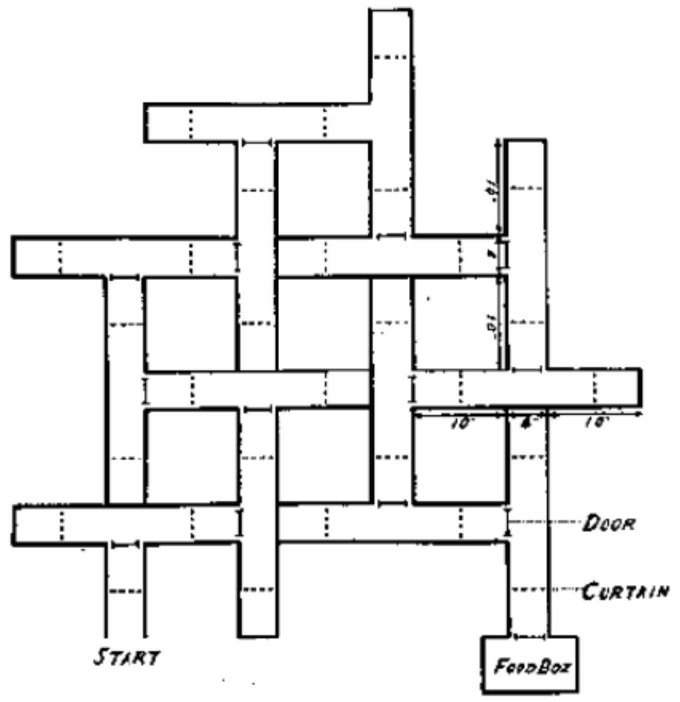




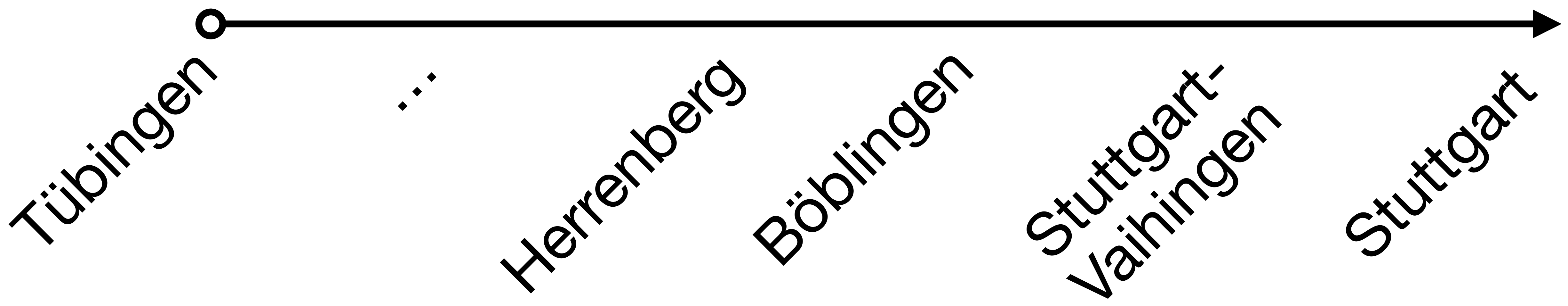
Outward journey: timetable change.
Due to timetable changes, your connection is no longer current. Please check the current travel options.

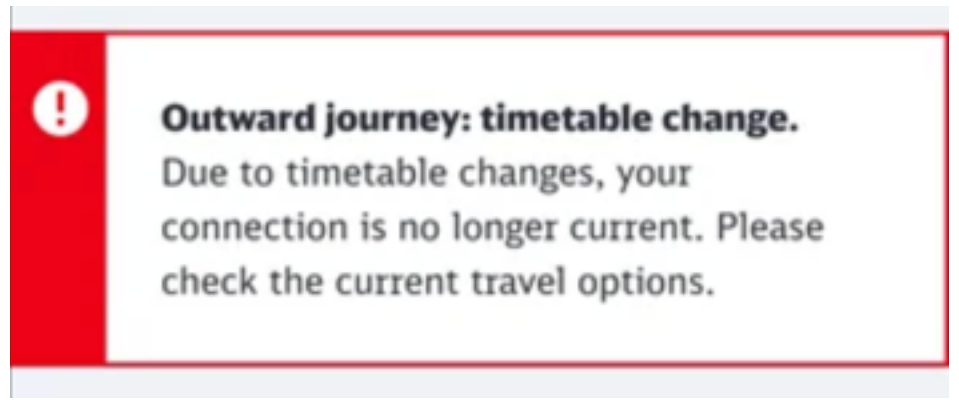
Successor Representation

Not just a map…



… but a goal-directed representation about which states are likely to encountered given a policy

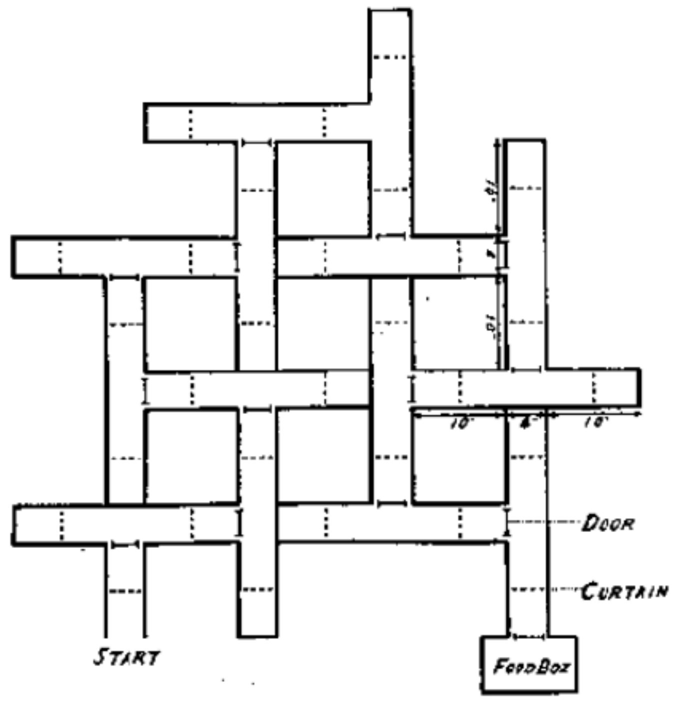




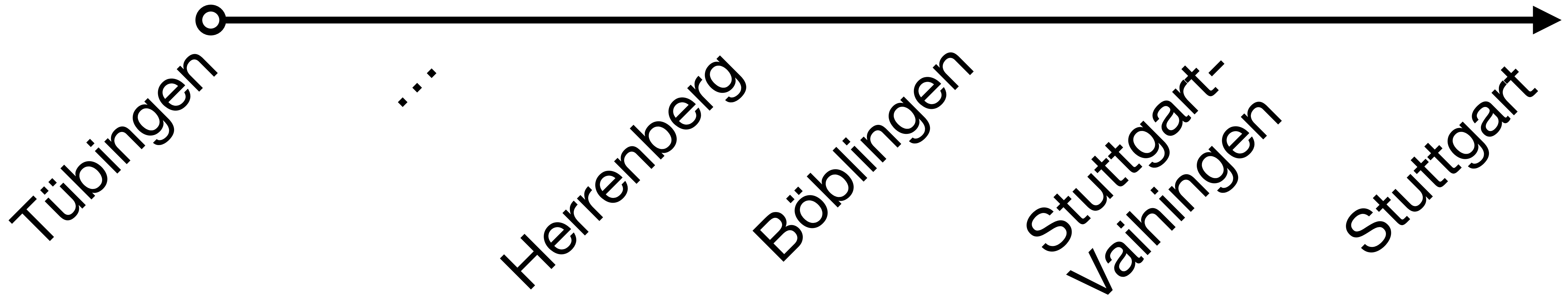
Outward journey: timetable change.
Due to timetable changes, your connection is no longer current. Please check the current travel options.

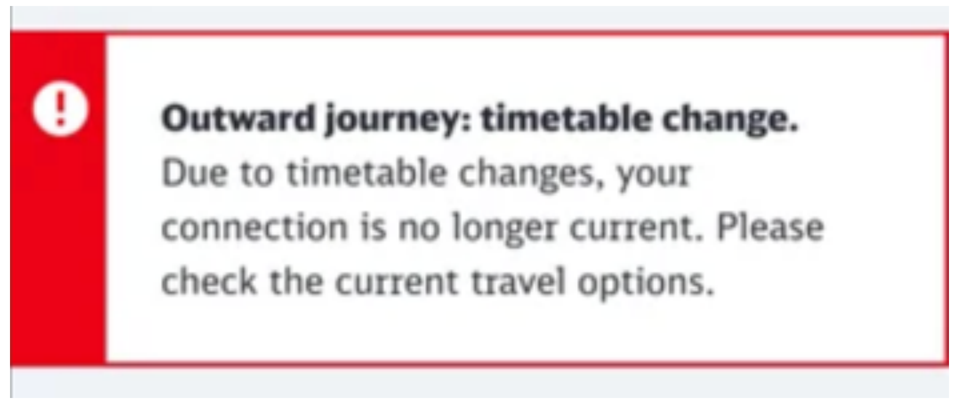
Successor Representation

Not just a map…



… but a goal-directed representation about which states are likely to encountered given a policy



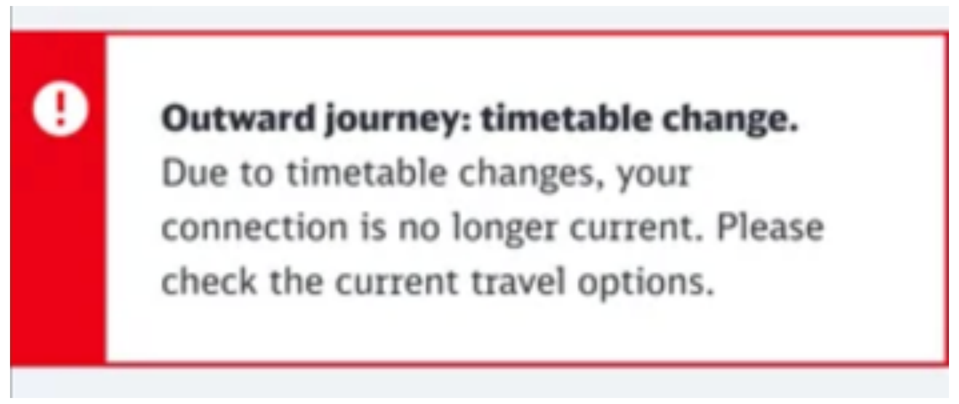


From a trajectory initiated in state s, the SR encodes the expected discounted future occupancy of state s':

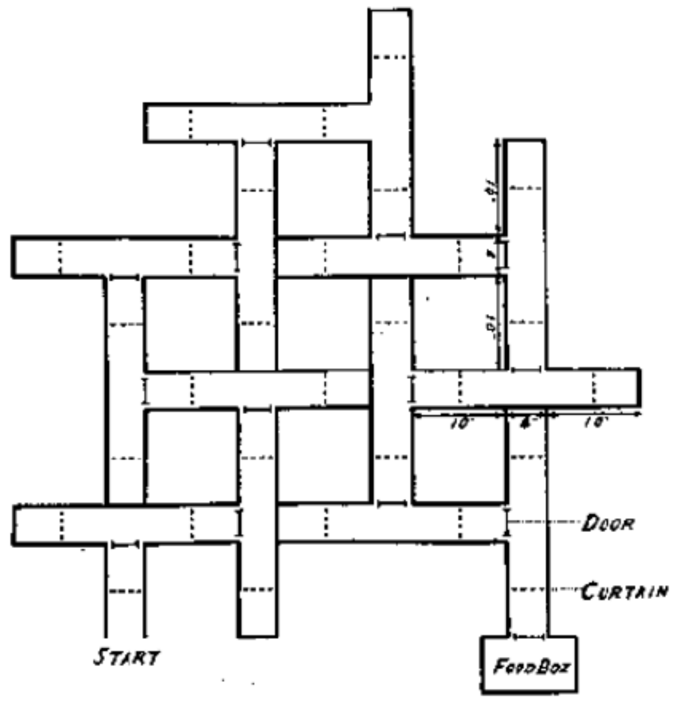
$$M(s, s') = \mathbb{E}_\pi \left[\sum_{t=0} \gamma^t \delta(s_t = s') \,|\, s_0 = s \right]$$

where $\delta(*)$ is the Kronecker delta and equal to 1 when the argument is true, and 0 otherwise

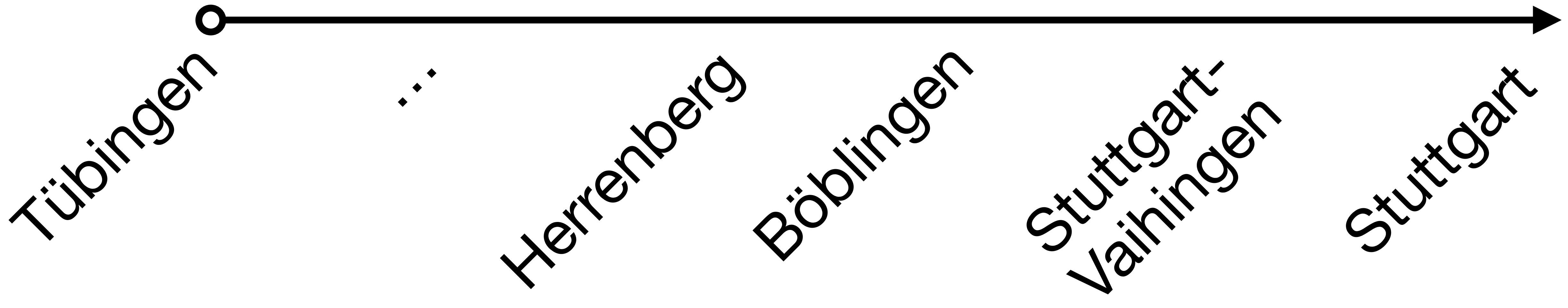
Successor Representation


Outward journey: timetable change.
Due to timetable changes, your connection is no longer current. Please check the current travel options.

Not just a map…

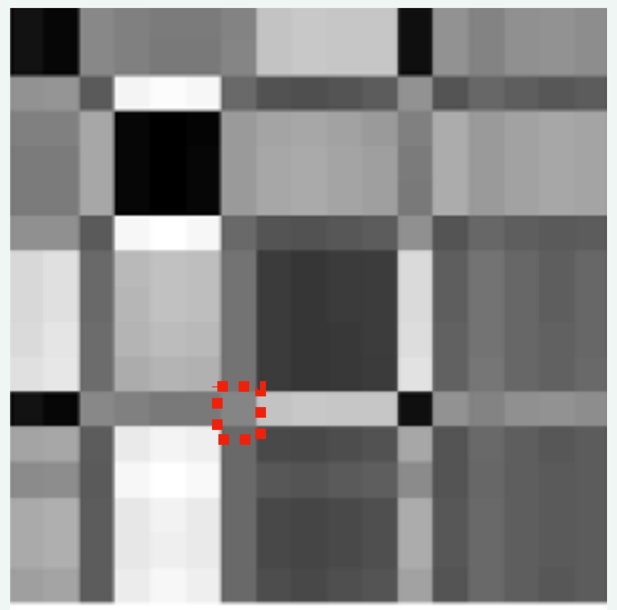


… but a goal-directed representation about which states are likely to encountered given a policy



From a trajectory initiated in state s, the SR encodes the expected discounted future occupancy of state s':

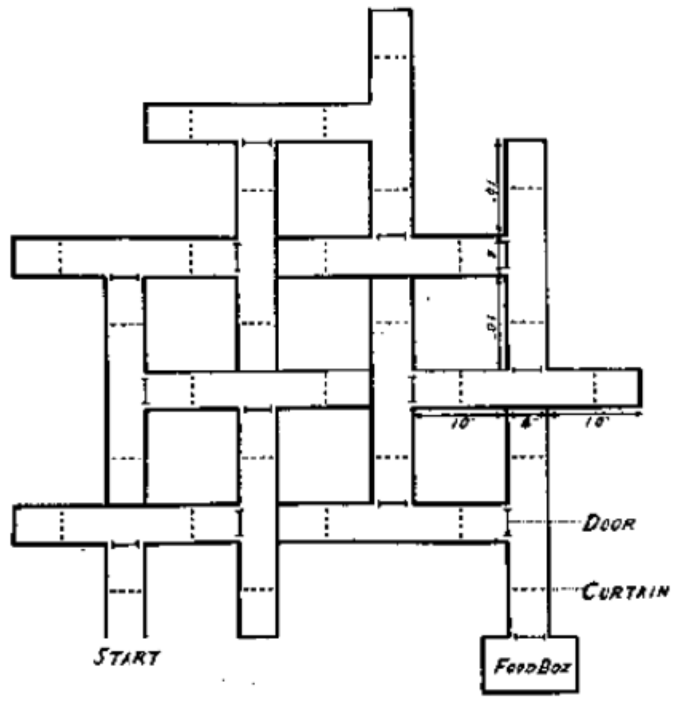
$$M(s, s') = \mathbb{E}_\pi \left[\sum_{t=0} \gamma^t \delta(s_t = s') \,|\, s_0 = s \right]$$



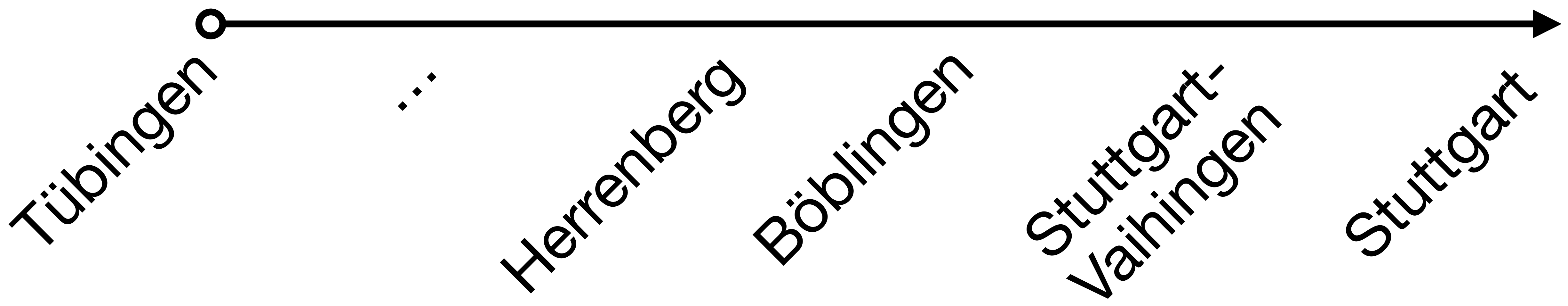
where δ(*) is the Kronecker delta and equal to 1 when the argument is true, and 0 otherwise

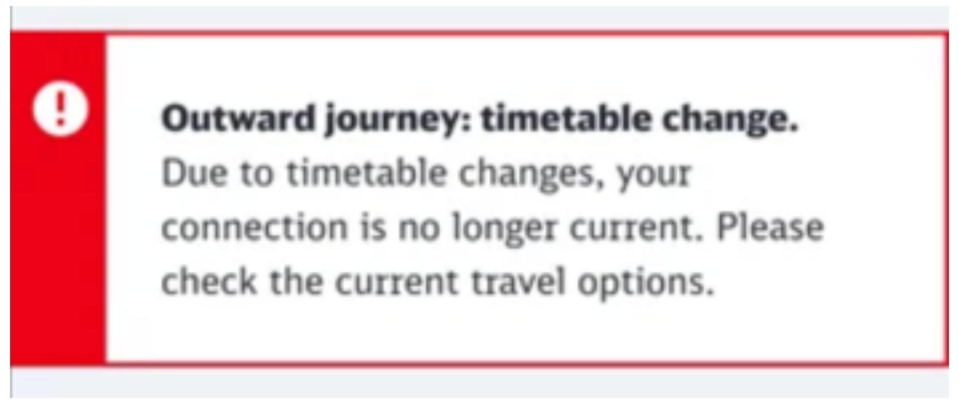
Successor Representation

Not just a map…



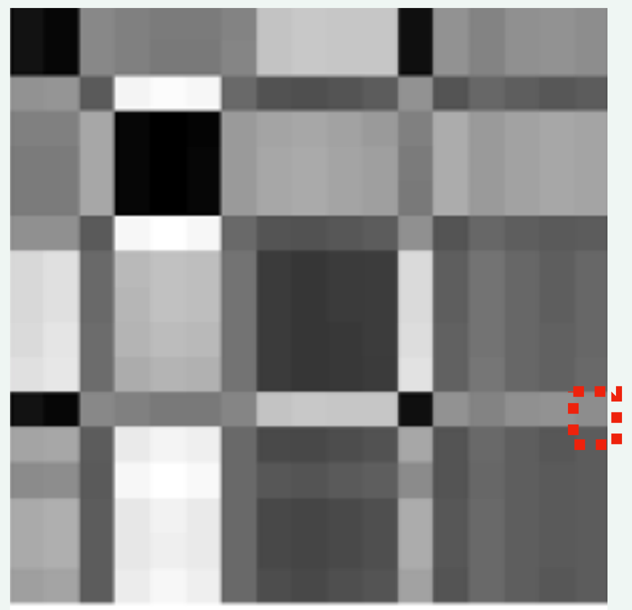
… but a goal-directed representation about which states are likely to encountered given a policy




Outward journey: timetable change.
Due to timetable changes, your connection is no longer current. Please check the current travel options.

From a trajectory initiated in state s, the SR encodes the expected discounted future occupancy of state s':

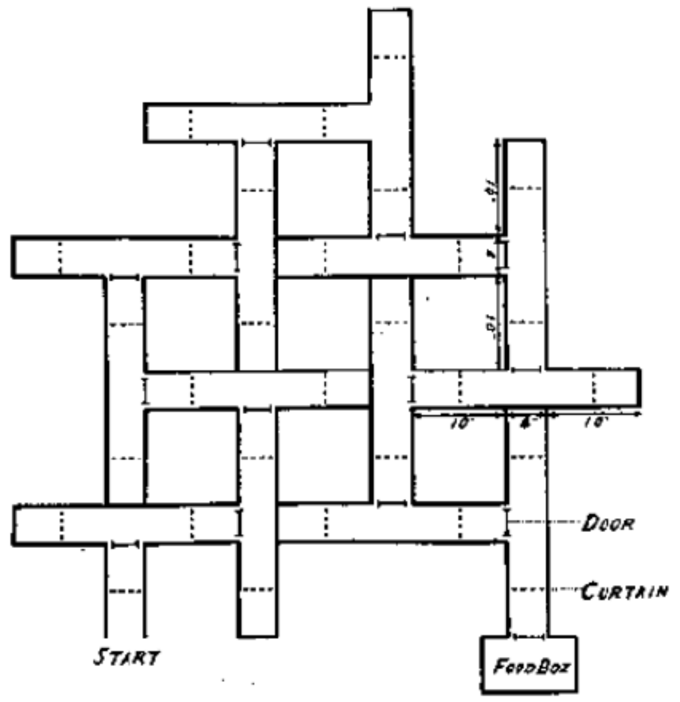
$$M(s, s') = \mathbb{E}_\pi \left[\sum_{t=0} \gamma^t \delta(s_t = s') \,|\, s_0 = s \right]$$



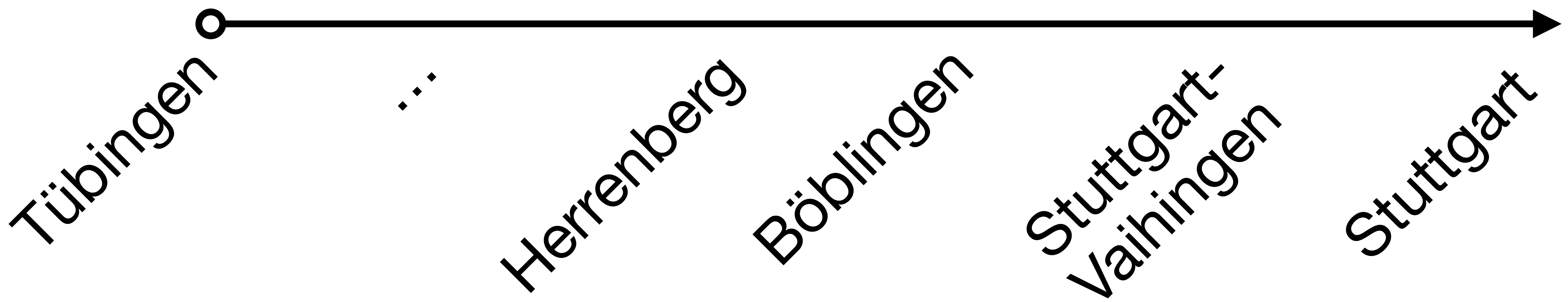
where $\delta(*)$ is the Kronecker delta and equal to 1 when the argument is true, and 0 otherwise

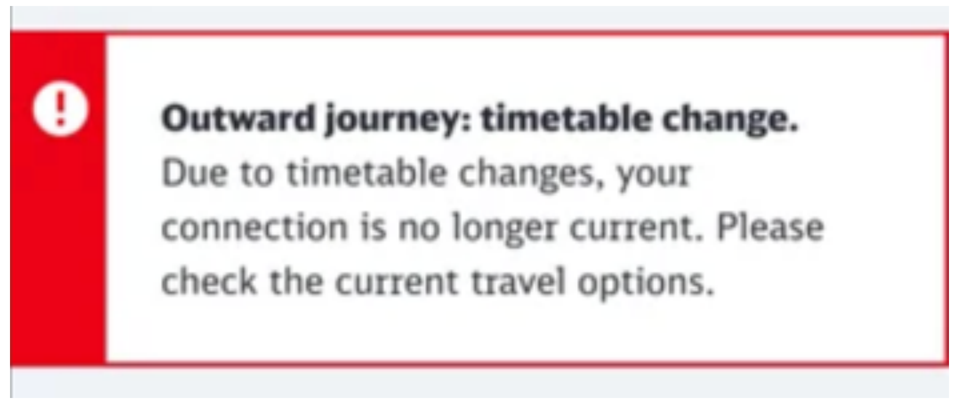
Successor Representation

Not just a map…



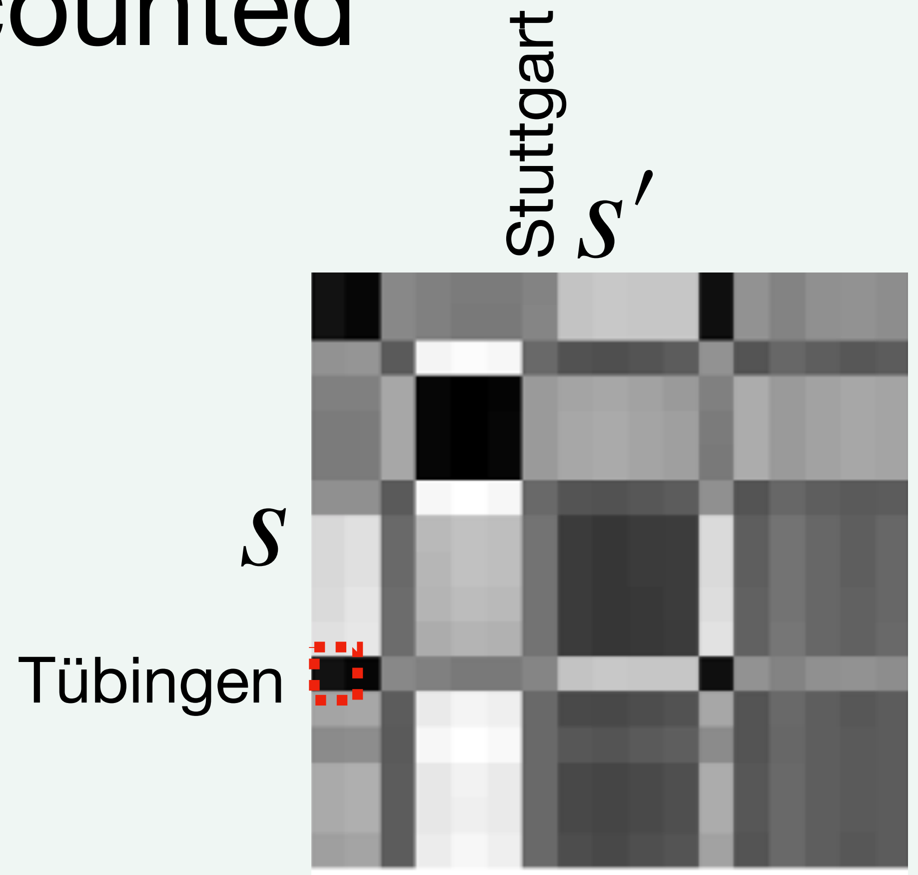
… but a goal-directed representation about which states are likely to encountered given a policy





From a trajectory initiated in state s, the SR encodes the expected discounted future occupancy of state s':

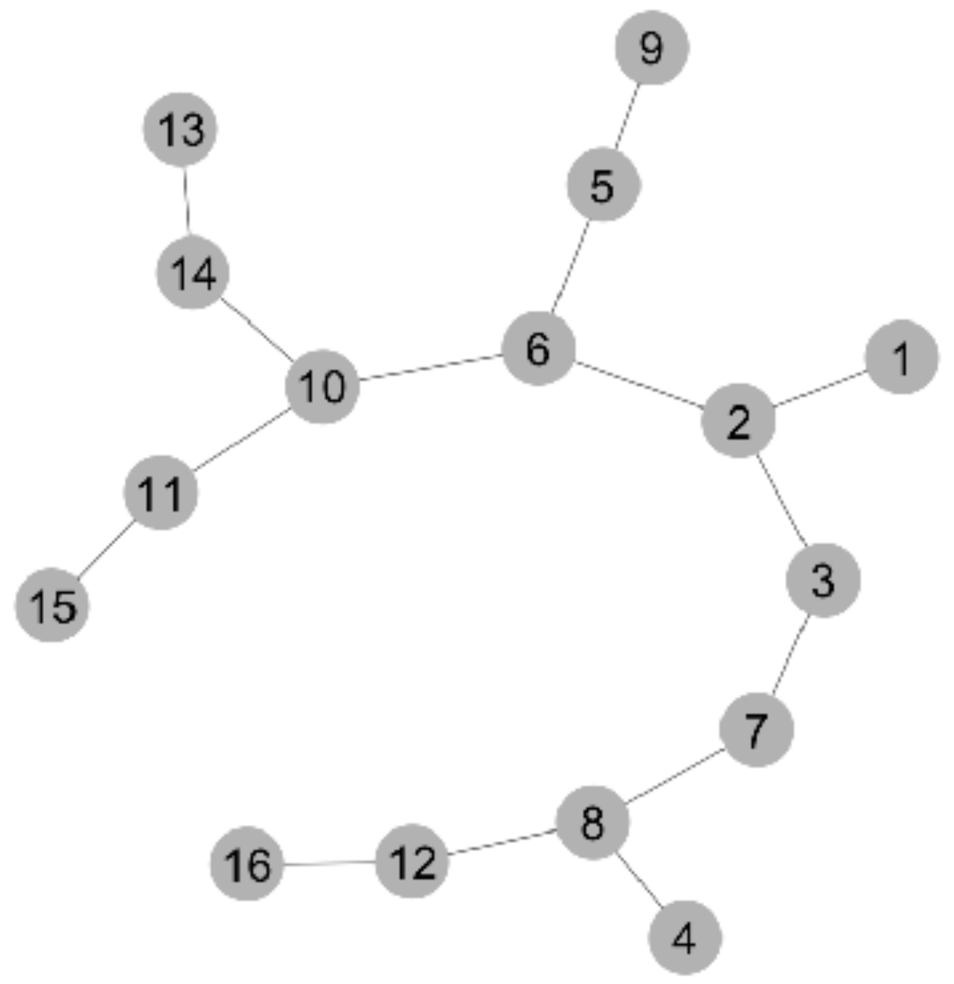
$$M(s, s') = \mathbb{E}_\pi \left[\sum_{t=0} \gamma^t \delta(s_t = s') \,|\, s_0 = s \right]$$



where $\delta(*)$ is the Kronecker delta and equal to 1 when the argument is true, and 0 otherwise

Computing the Successor Representation (off-policy)

If the state space is fully known, we can compute the SR in closed form:



$$M(s, s') = \sum_{t=0}^{\infty} \gamma^t T^t = (I - \gamma T)^{-1}$$

I is the identity matrix, γ is the temporal discount factor

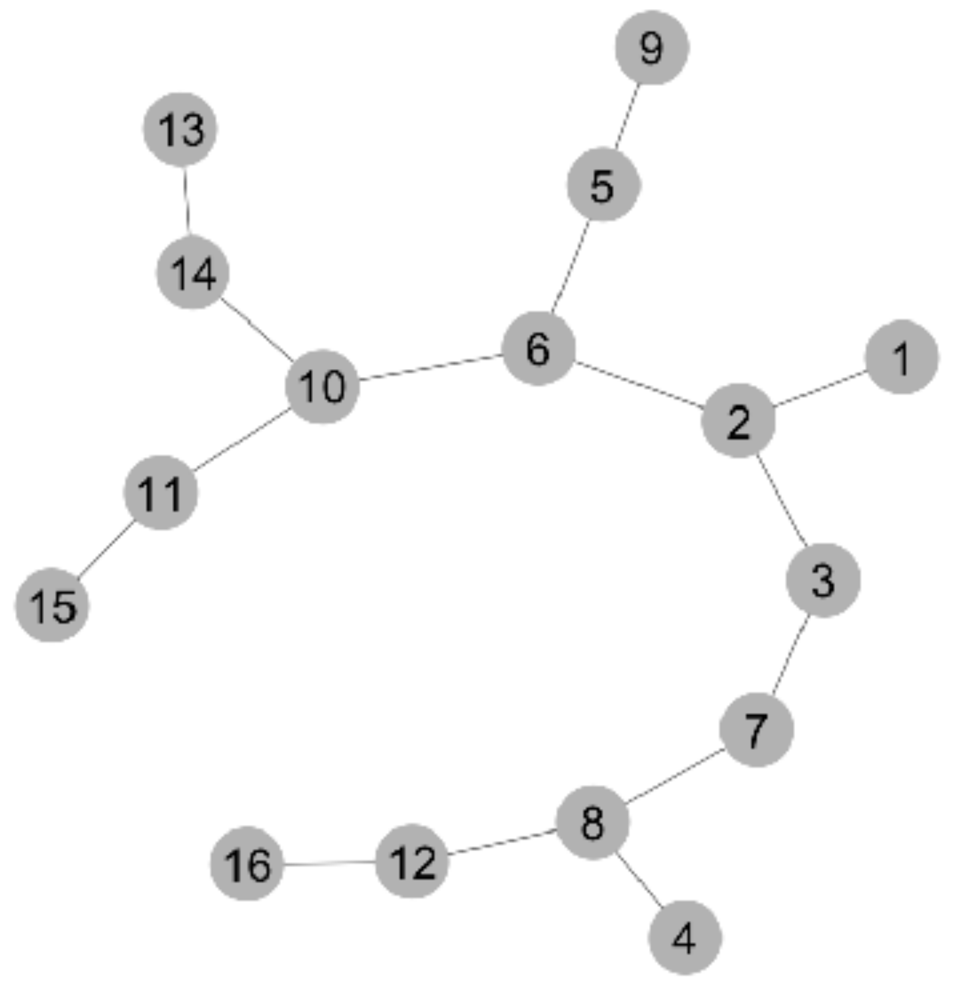
T is the transition matrix under a policy: $T(s, s') = \sum_a \pi(a \mid s) P(s' \mid s, a)$

A further simplifcation that is often used is to assume a random policy, allowsing us to define T using the degree (D) and adjacency (A) matrices

$$T = D^{-1} A$$

Computing the Successor Representation (off-policy)

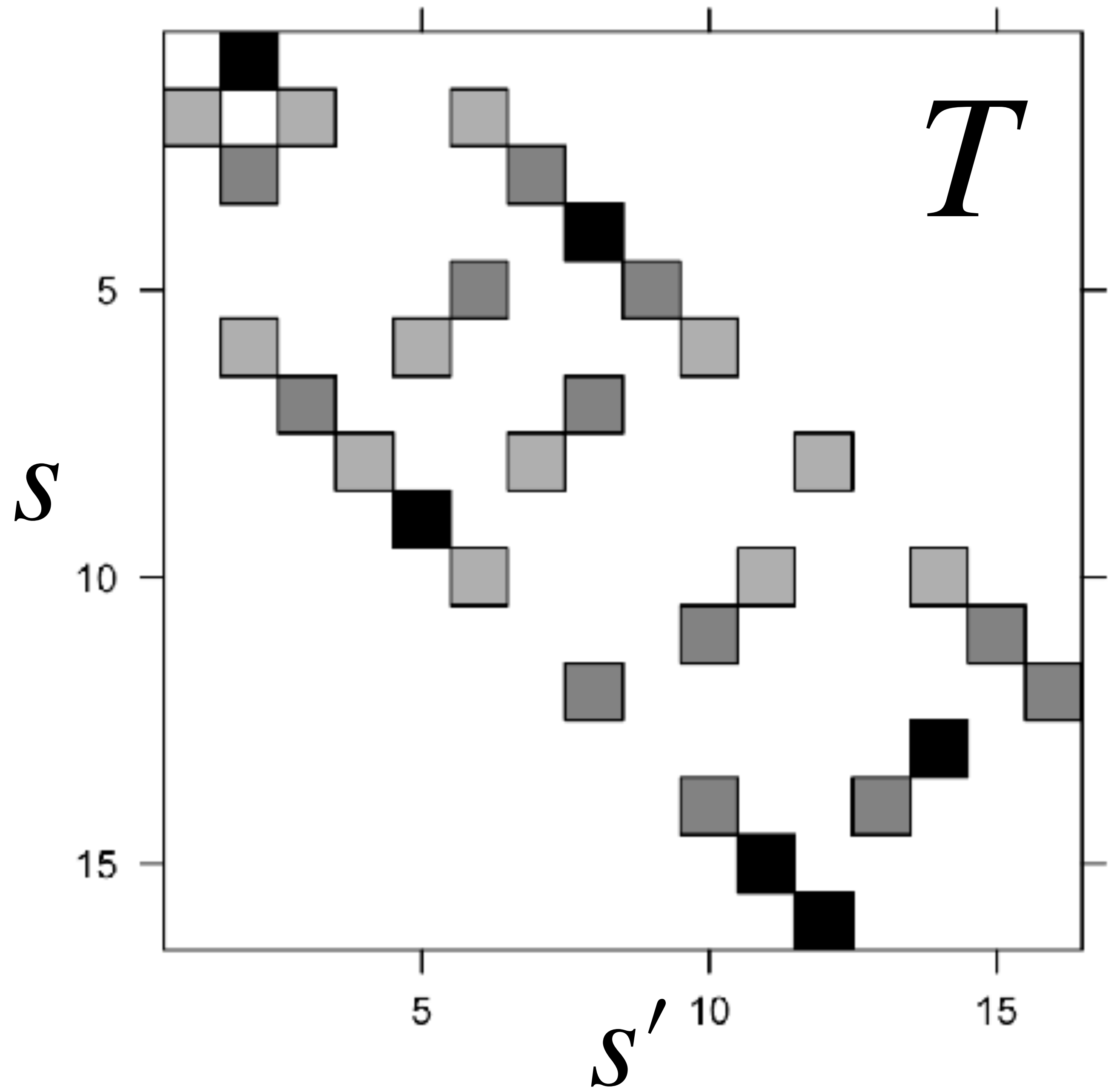
If the state space is fully known, we can compute the SR in closed form:



$$M(s, s') = \sum_{t=0}^{\infty} \gamma^t T^t = (I - \gamma T)^{-1}$$

I is the identity matrix, γ is the temporal discount factor

T is the transition matrix under a policy: $T(s, s') = \sum_{a} \pi(a \mid s) P(s' \mid s, a)$



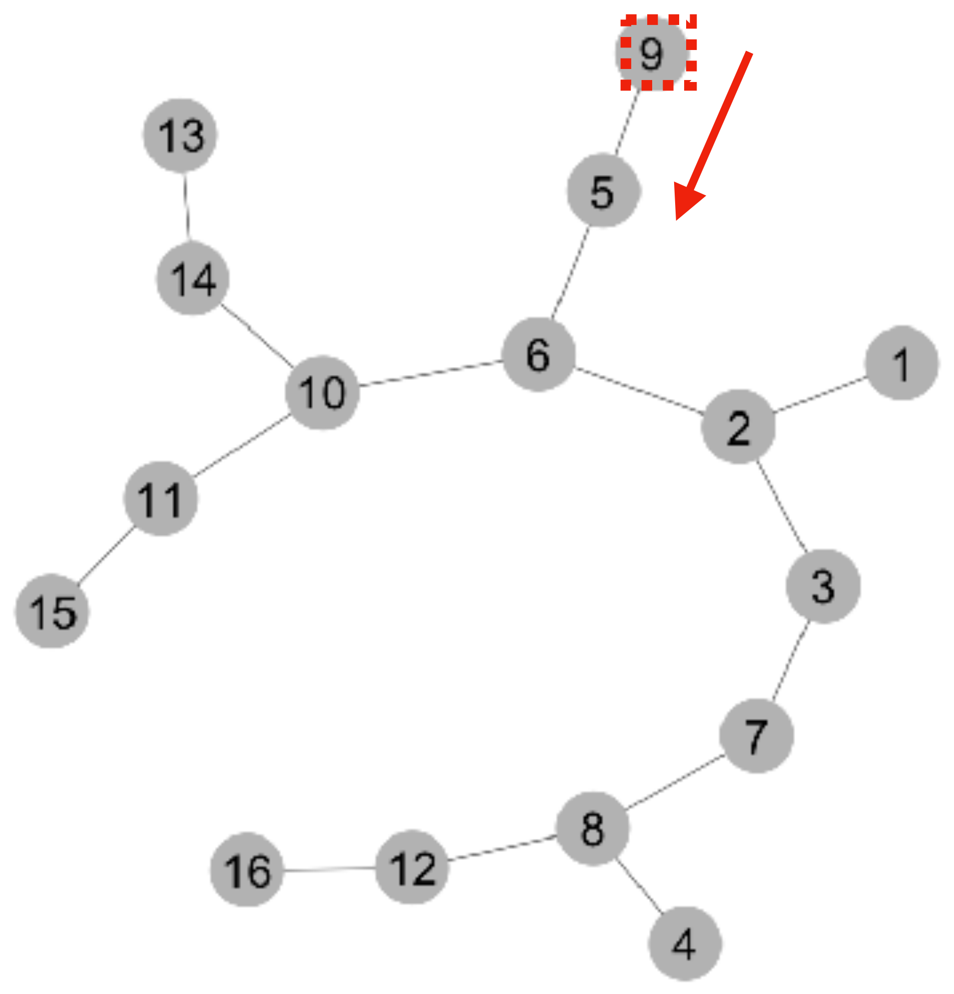
A further simplifcation that is often used is to assume a random policy, allowsing us to define T using the degree (D) and adjacency (A) matrices

$$T = D^{-1} A$$

Computing the Successor Representation (off-policy)

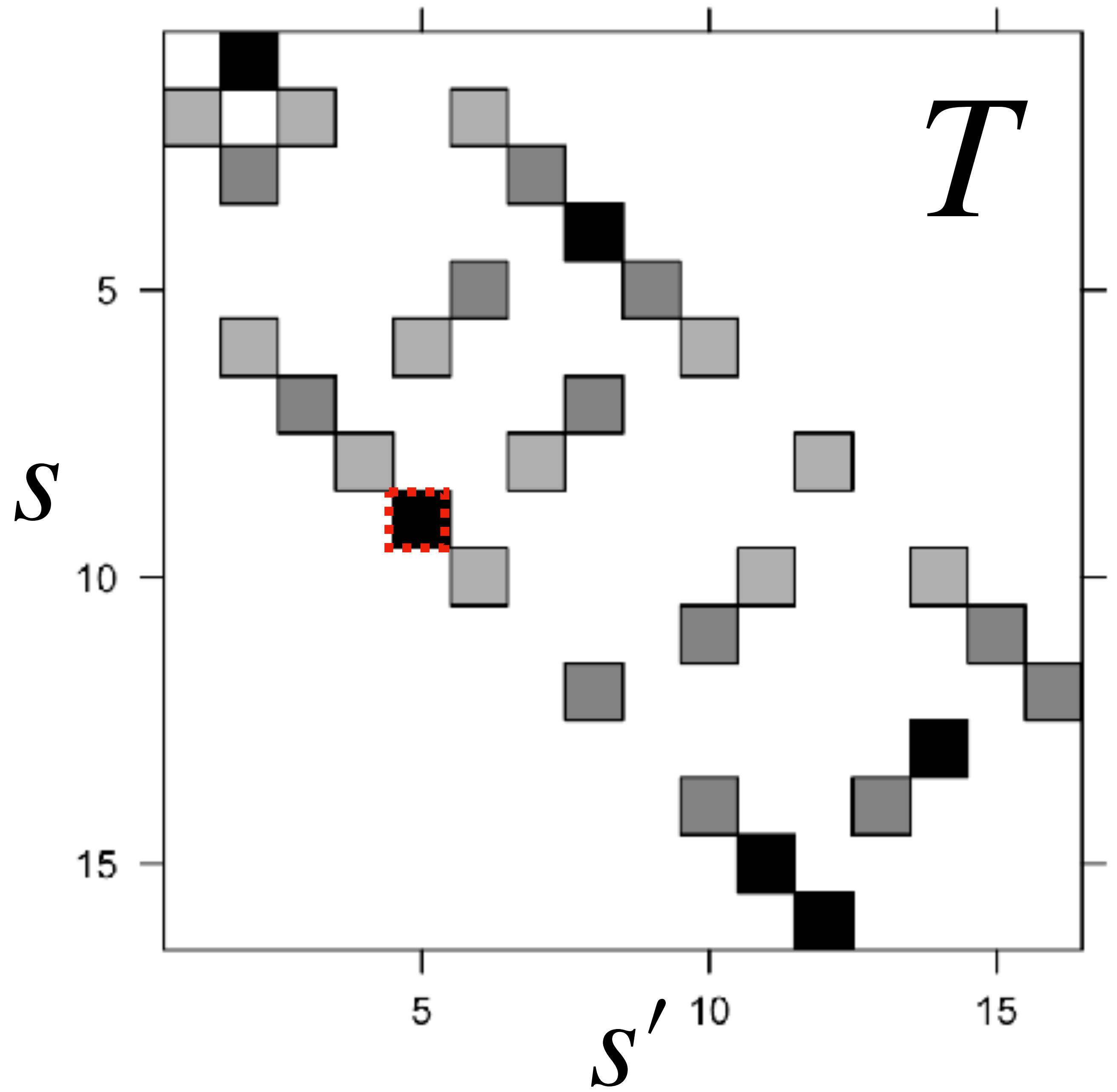
If the state space is fully known, we can compute the SR in closed form:

$$M(s, s') = \sum_{t=0}^{\infty} \gamma^t T^t = (I - \gamma T)^{-1}$$



I is the identity matrix, γ is the temporal discount factor

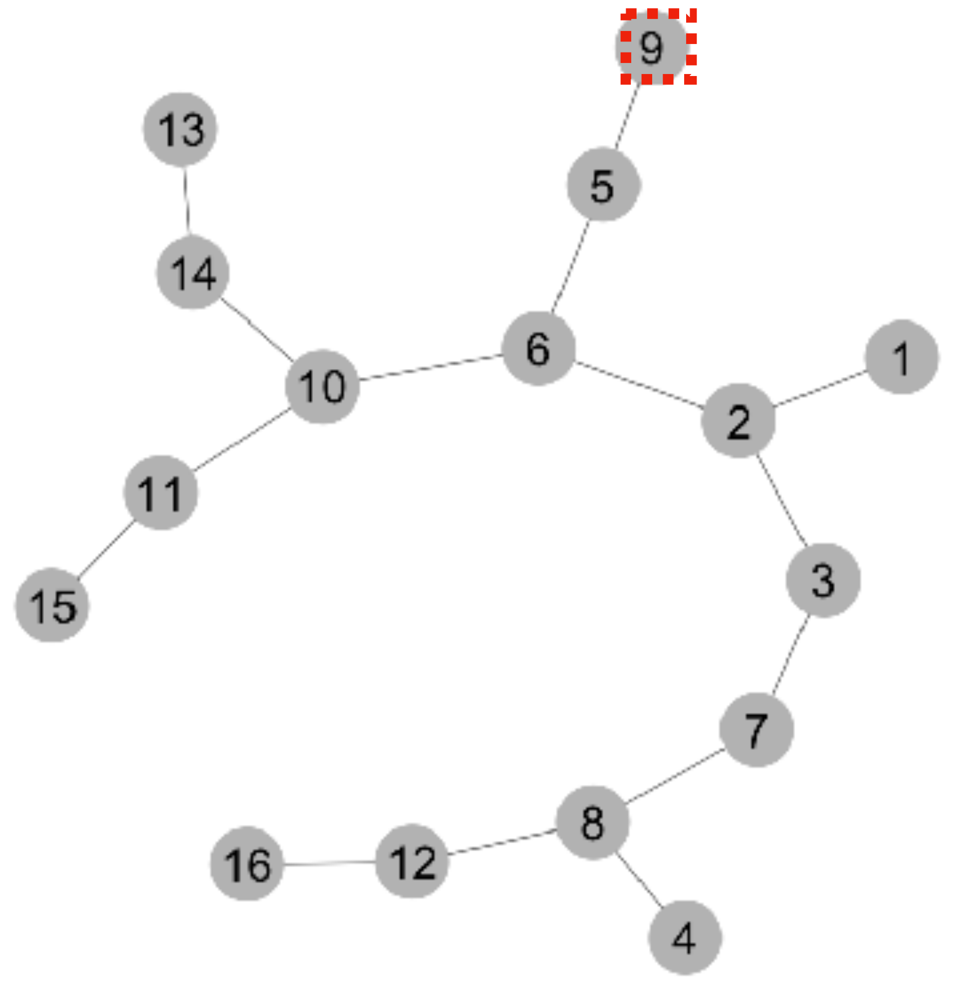
T is the transition matrix under a policy: $T(s, s') = \sum_a \pi(a \mid s) P(s' \mid s, a)$



A further simplifcation that is often used is to assume a random policy, allowsing us to define T using the degree (D) and adjacency (A) matrices

$$T = D^{-1} A$$

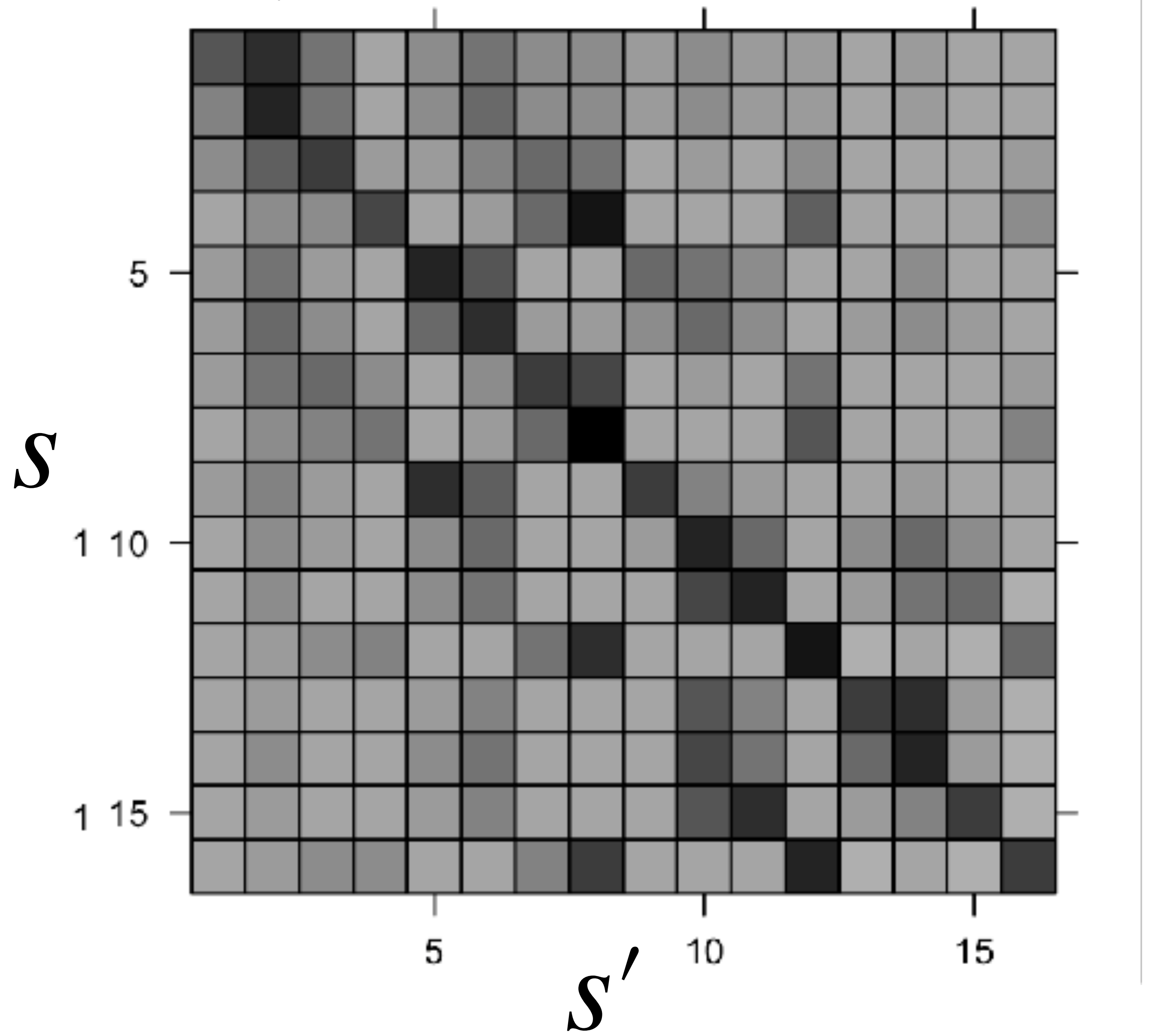
Computing the Successor Representation (off-policy)



If the state space is fully known, we can compute the SR in closed form:

$$M(s, s') = \sum_{t=0}^{\infty} \gamma^t T^t = (I - \gamma T)^{-1}$$

$M_{\gamma=.9}(s, s')$



I is the identity matrix, γ is the temporal discount factor

T is the transition matrix under a policy: $T(s, s') = \sum_{a} \pi(a \mid s) P(s' \mid s, a)$

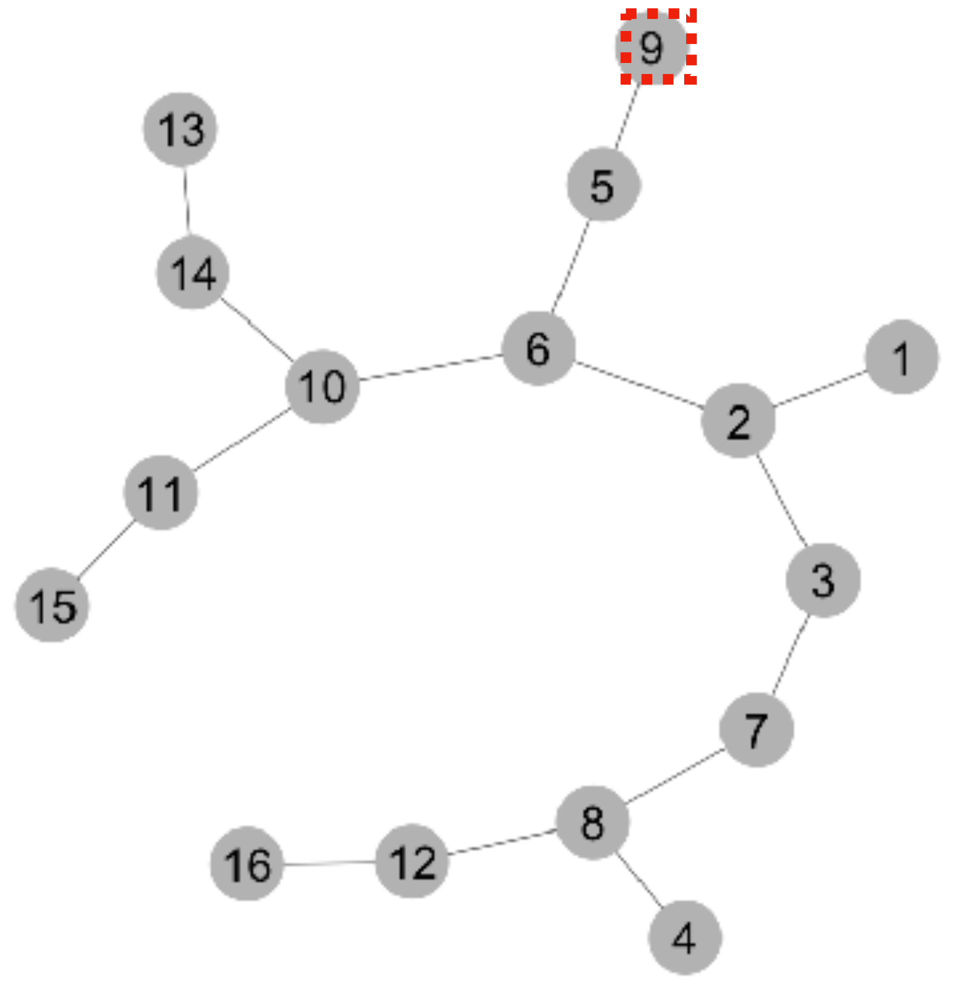
A further simplifcation that is often used is to assume a random policy, allowsing us to define T using the degree (D) and adjacency (A) matrices

$$T = D^{-1} A$$

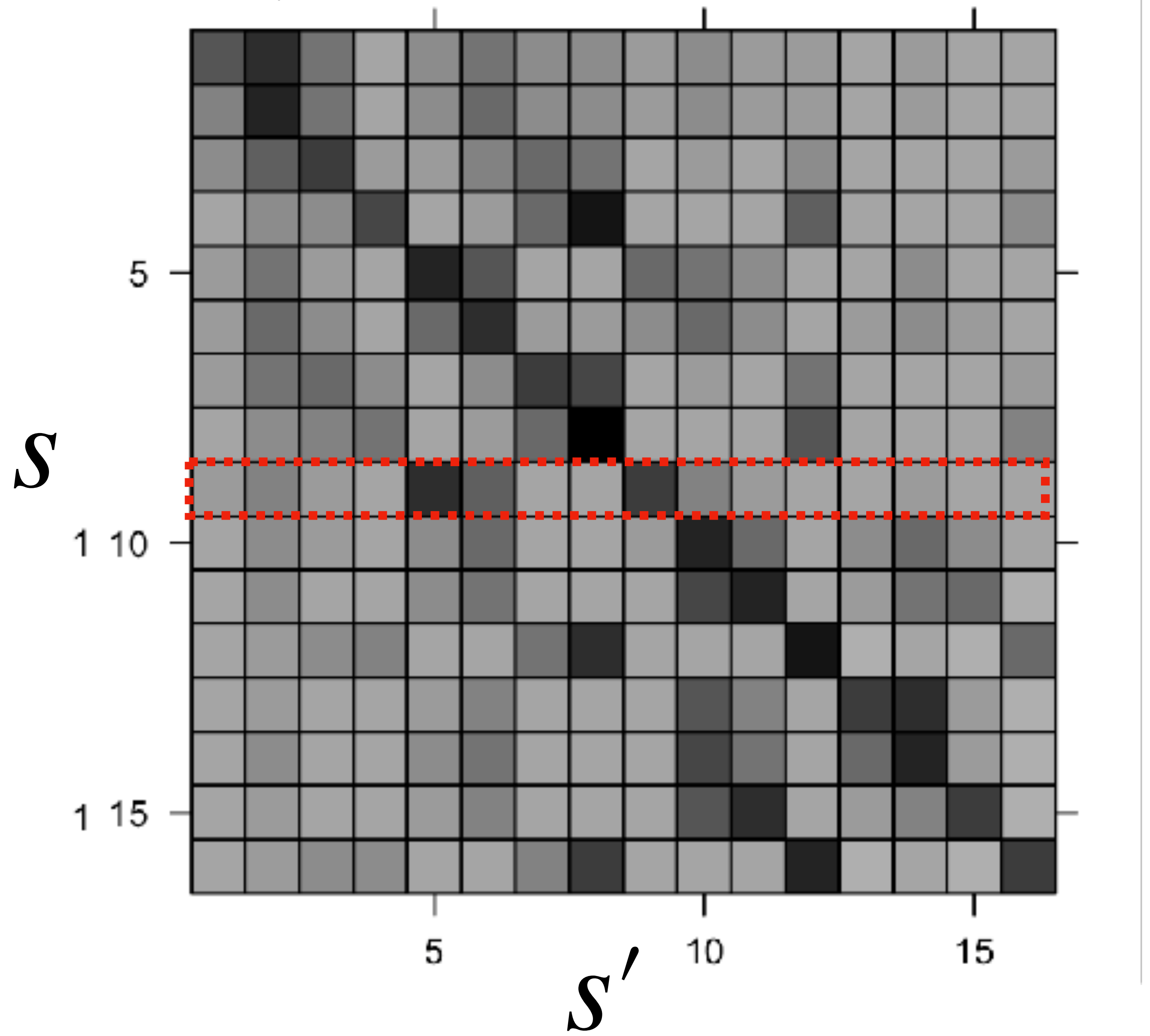
Computing the Successor Representation (off-policy)

If the state space is fully known, we can compute the SR in closed form:

$$M(s,s') = \sum_{t=0}^{\infty} \gamma^t T^t = (I - \gamma T)^{-1}$$



$M_{\gamma=.9}(s,s')$



I is the identity matrix, γ is the temporal discount factor

T is the transition matrix under a policy: $T(s,s') = \sum_a \pi(a \mid s) P(s' \mid s, a)$

A further simplifcation that is often used is to assume a random policy, allowsing us to define T using the degree (D) and adjacency (A) matrices

$$T = D^{-1}A$$

Computing the Successor Representation (on policy)

If the state space is not known, we can compute the SR using the delta-rule:

$$\hat{M}_{t+1}(s_t, s') = \hat{M}_t(s_t, s') + \alpha \left[\delta(s_t = s') + \gamma \hat{M}_{t+1}(s_t, s') - \hat{M}_t(s_t, s')\right]$$

where α is the learning rate and δ is the kronecker delta $\delta = 1$ when true, 0 otherwise

- This update is identical to the temporal difference learning rule for value functions

- The successor representation is updated based on the *successor* prediction error instead of the *reward* prediction error

Computing the Successor Representation (on policy)

Russek et al., (2017)

If the state space is not known, we can compute the SR using the delta-rule:

$$\hat{M}_{t+1}(s_t, s') = \hat{M}_t(s_t, s') + \alpha \left[\delta(s_t = s') + \gamma \hat{M}_{t+1}(s_t, s') - \hat{M}_t(s_t, s')\right]$$

where α is the learning rate and δ is the kronecker delta $\delta = 1$ when true, 0 otherwise

- This update is identical to the temporal difference learning rule for value functions
- The successor representation is updated based on the *successor* prediction error instead of the *reward* prediction error

Computing the Successor Representation (on policy)

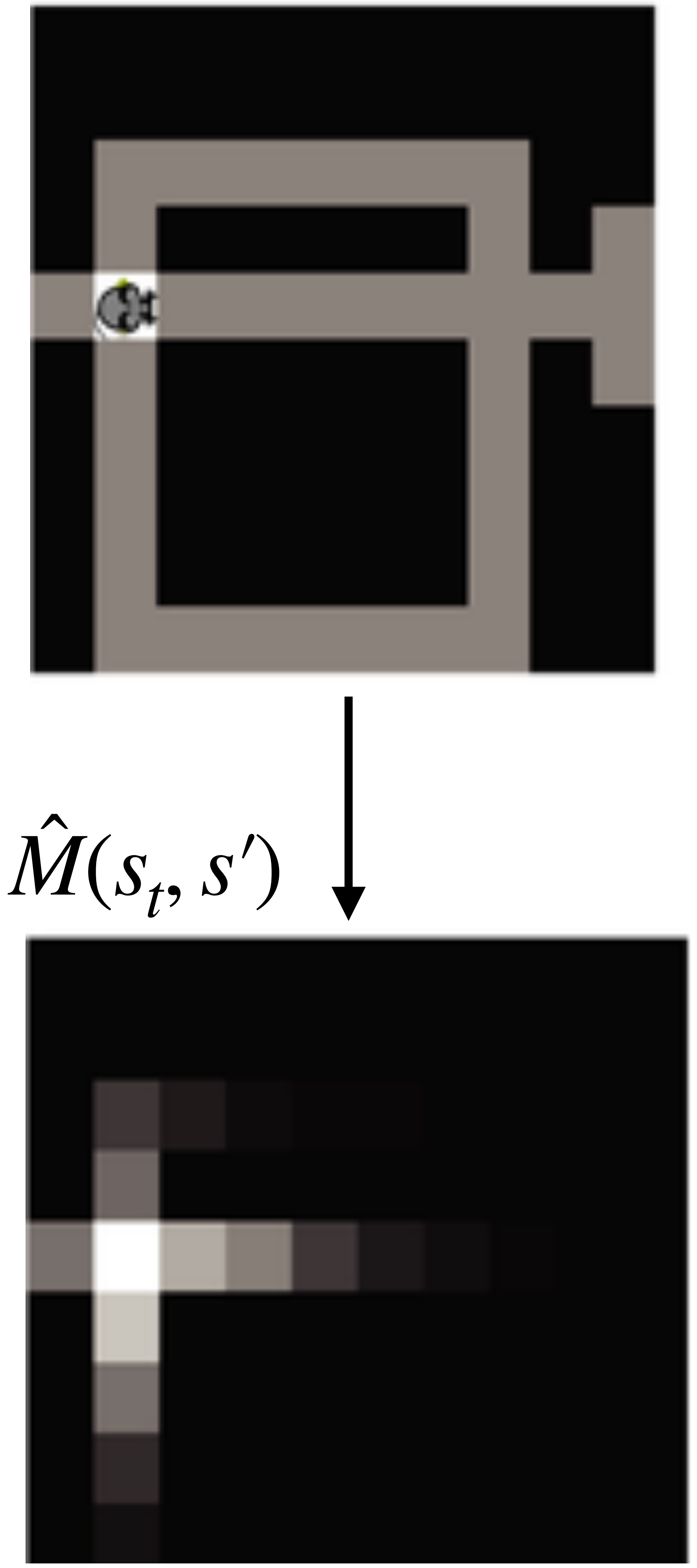
Russek et al., (2017)

If the state space is not known, we can compute the SR using the delta-rule:

$$\hat{M}_{t+1}(s_t, s') = \hat{M}_t(s_t, s') + \alpha \left[\delta(s_t = s') + \gamma \hat{M}_{t+1}(s_t, s') - \hat{M}_t(s_t, s')\right]$$

where α is the learning rate and δ is the kronecker delta $\delta = 1$ when true, 0 otherwise

- This update is identical to the temporal difference learning rule for value functions
- The successor representation is updated based on the *successor* prediction error instead of the *reward* prediction error



Computing the Successor Representation (on policy)

Russek et al., (2017)

$\hat{M}(s_t, s')$

If the state space is not known, we can compute the SR using the delta-rule:

$$\hat{M}_{t+1}(s_t, s') = \hat{M}_t(s_t, s') + \alpha \left[\delta(s_t = s') + \gamma \hat{M}_{t+1}(s_t, s') - \hat{M}_t(s_t, s') \right]$$

where α is the learning rate and δ is the kronecker delta $\delta = 1$ when true, 0 otherwise

- This update is identical to the temporal difference learning rule for value functions

- The successor representation is updated based on the *successor* prediction error instead of the *reward* prediction error

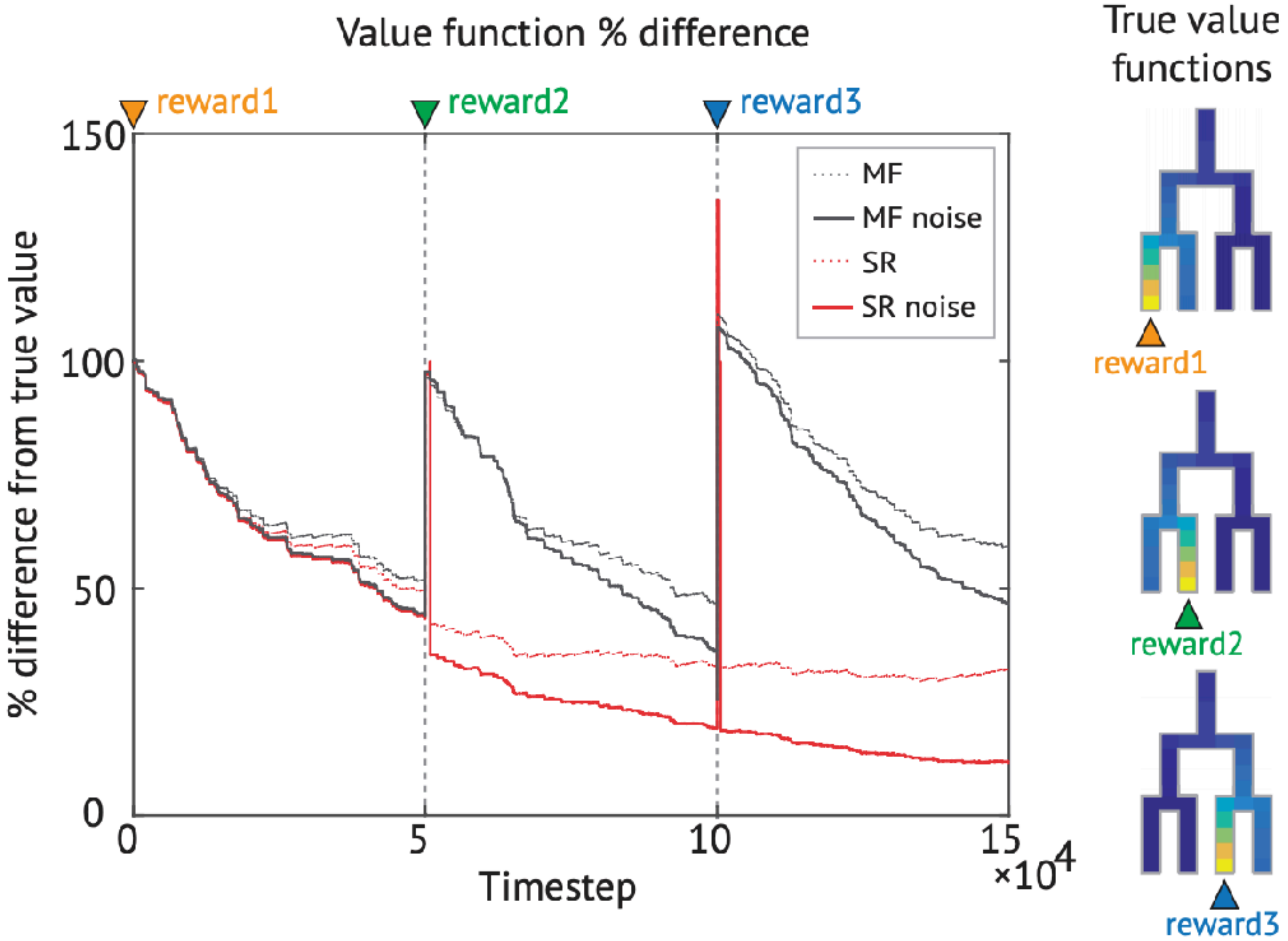
SR generalizes for changes in reward

- If the location of rewards change, only the $r(s')$ part needs to be re-learned while $M(s, s')$ remains the same

$$V^{\pi}(s) = \sum_{s'} M(s, s') r(s')$$

- This leads to faster generalization to changes in the environment

Model-Free and SR value computations with changing reward and noise



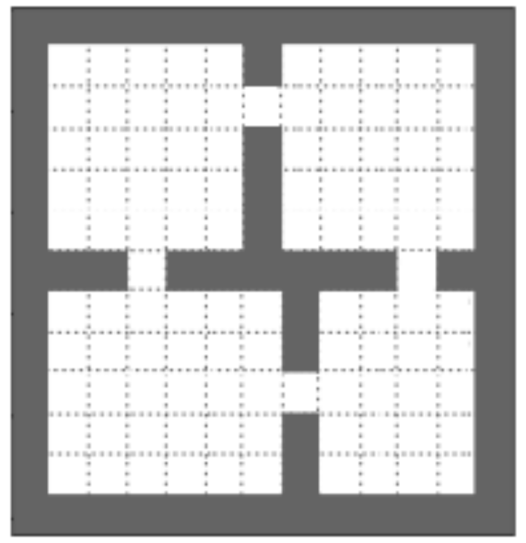
Stachenfeld, Botvinick & Gershman (2017)

The Eigenvalues of the SR

- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues

Environment

Machado et al., (2023)

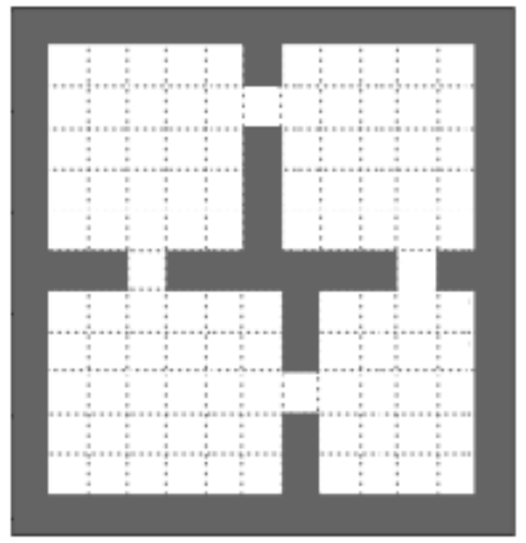


The Eigenvalues of the SR

- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues

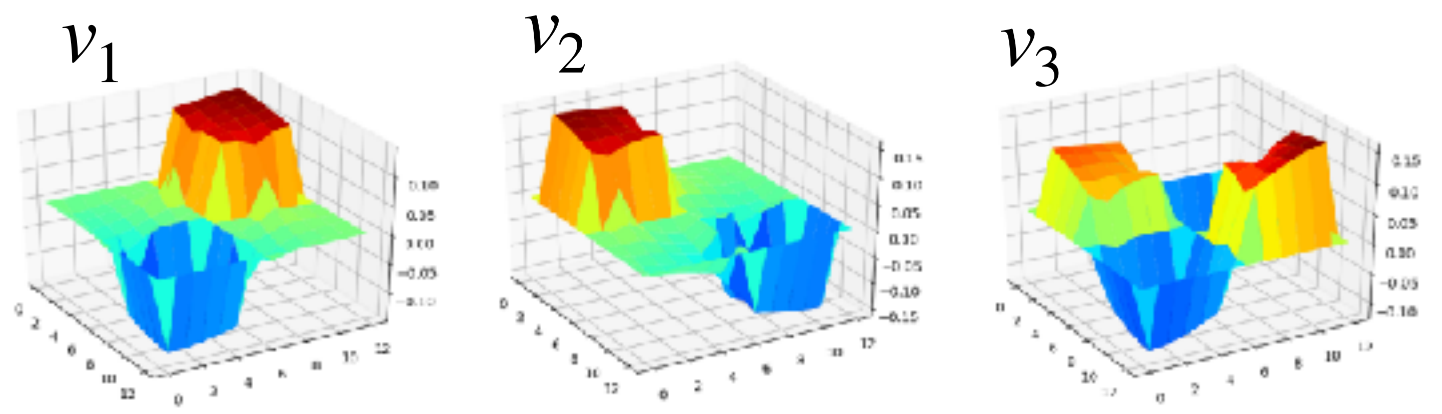
The Eigenvalues of the SR

Environment



- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues

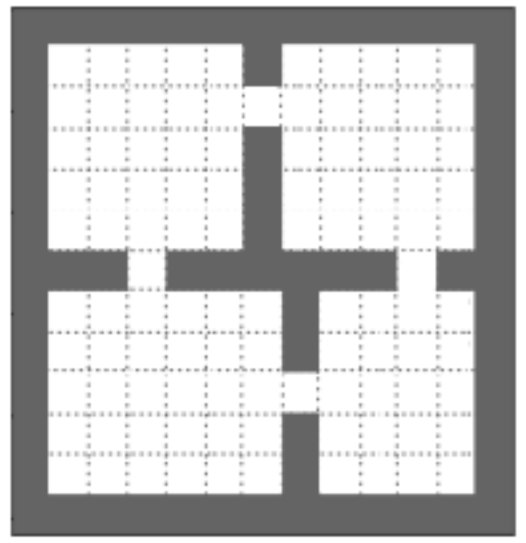
Eigenvectors

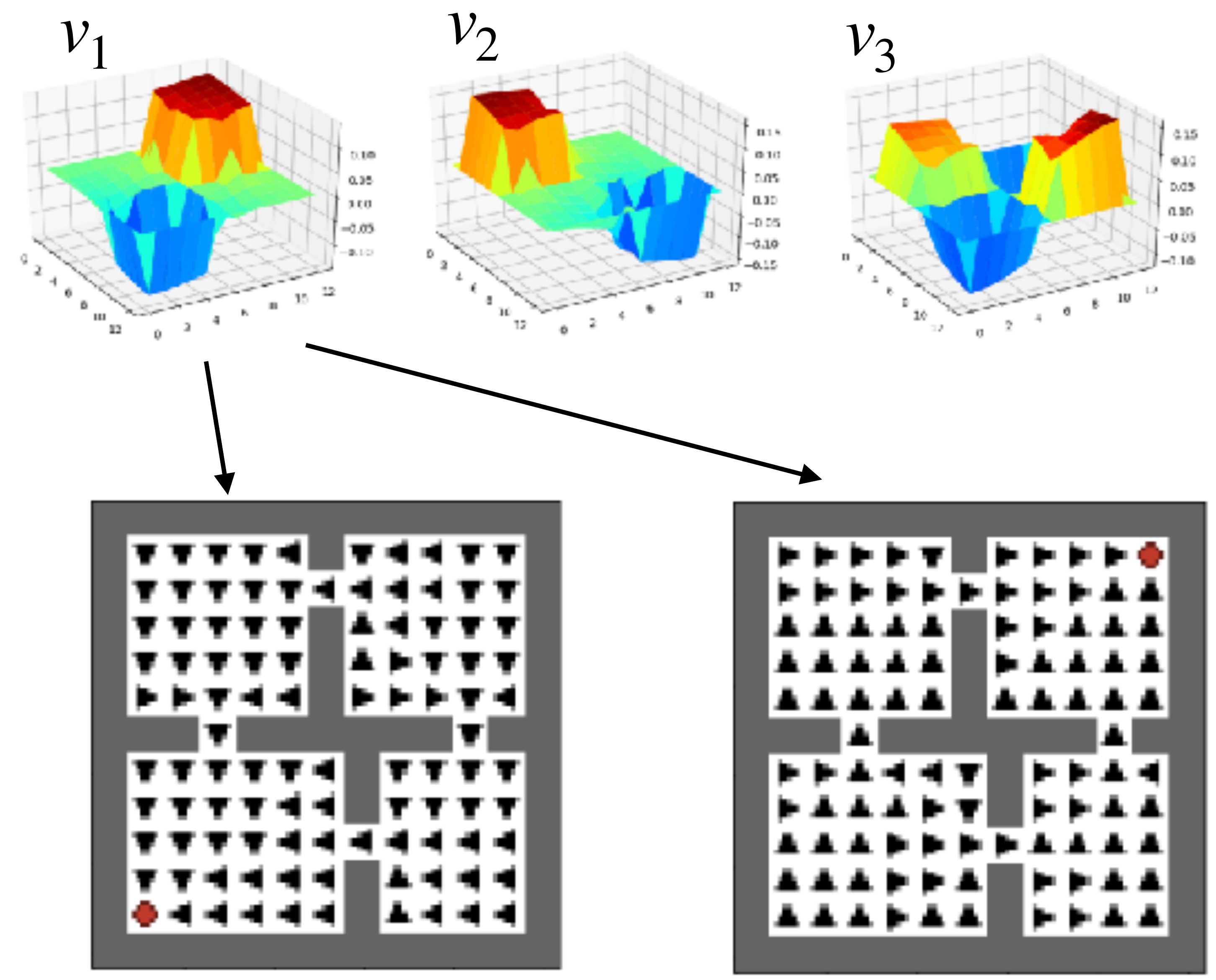


The Eigenvalues of the SR

- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues
- For the SR, the different Eigenvectors capture different orthogonal patterns of state visitation

Machado et al., (2023)



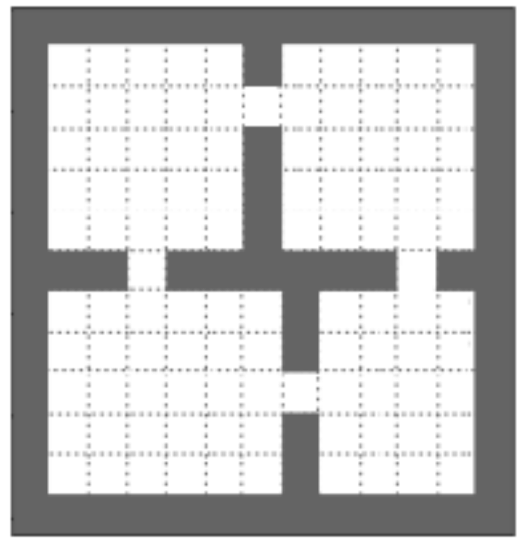


The Eigenvalues of the SR

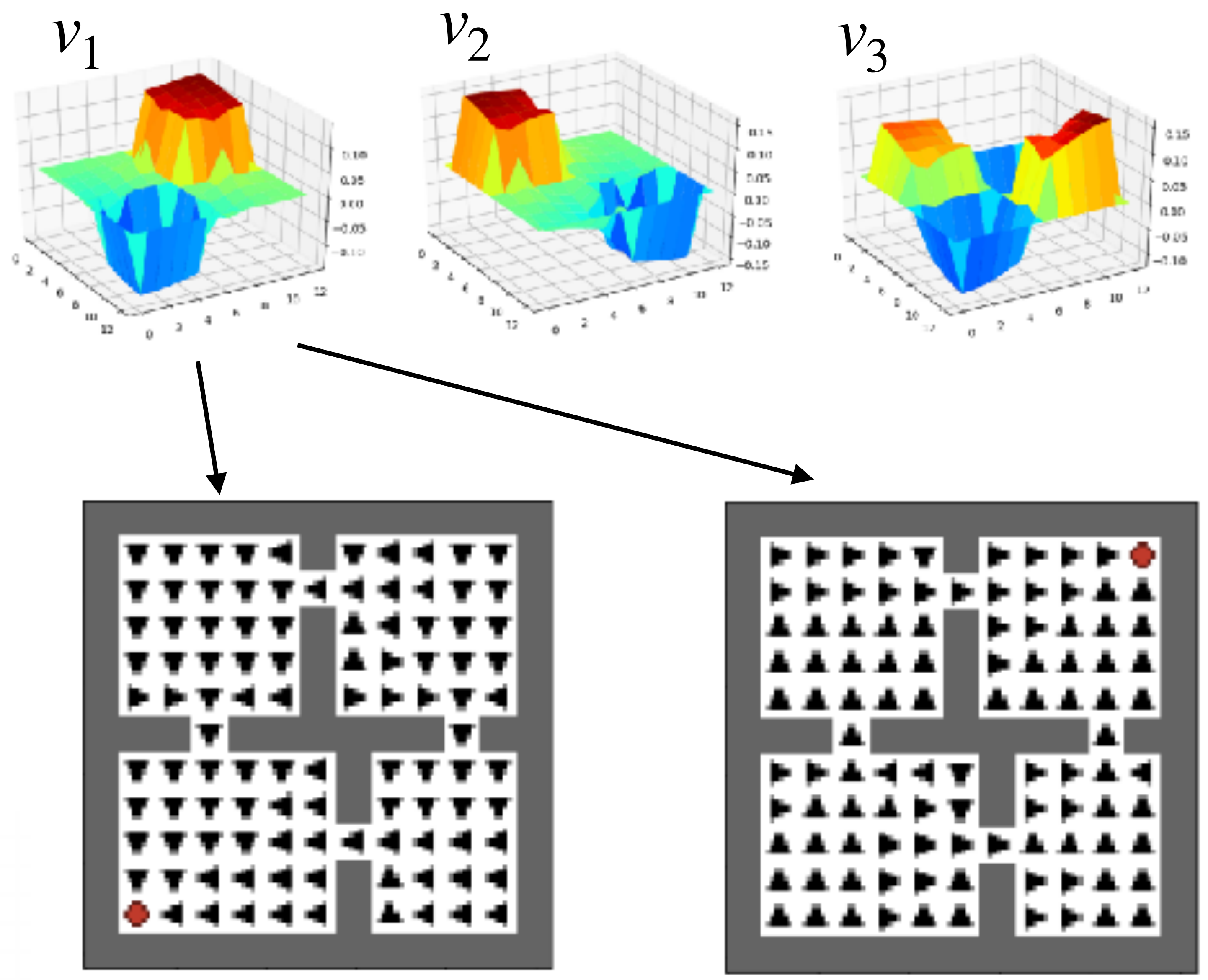
- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues
- For the SR, the different Eigenvectors capture different orthogonal patterns of state visitation
- In more regular environments, the Eigenvectors of the SR are gridded*, just like grid cells in the Entorhinal Cortex

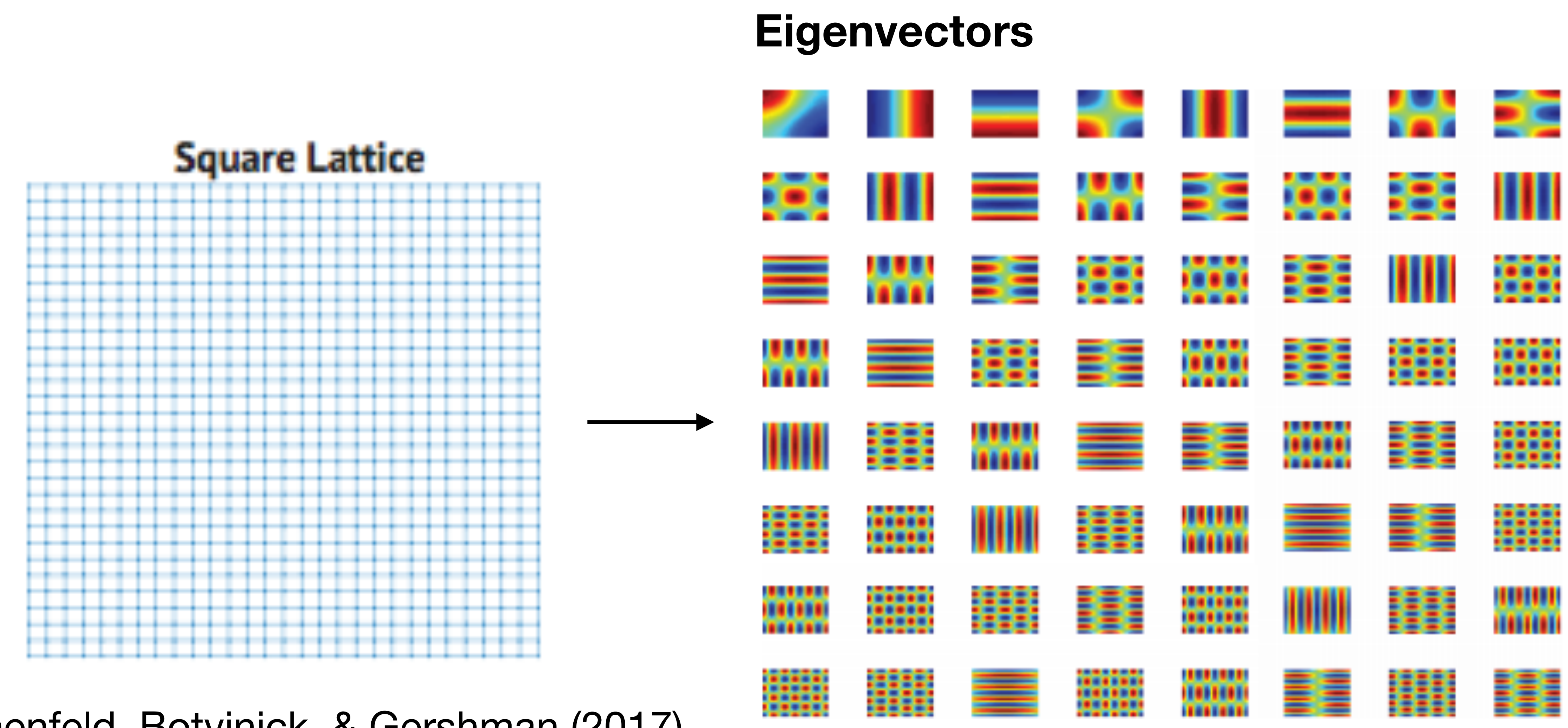
Environment

Machado et al., (2023)

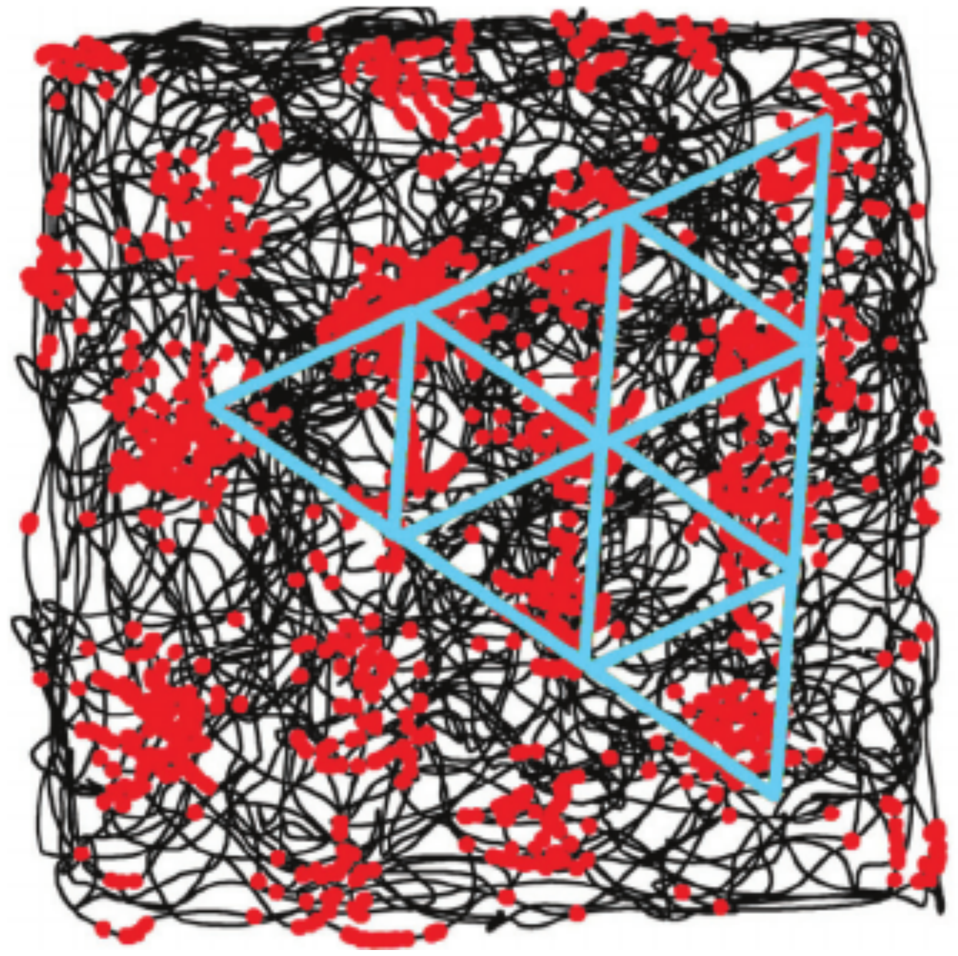


Eigenvectors



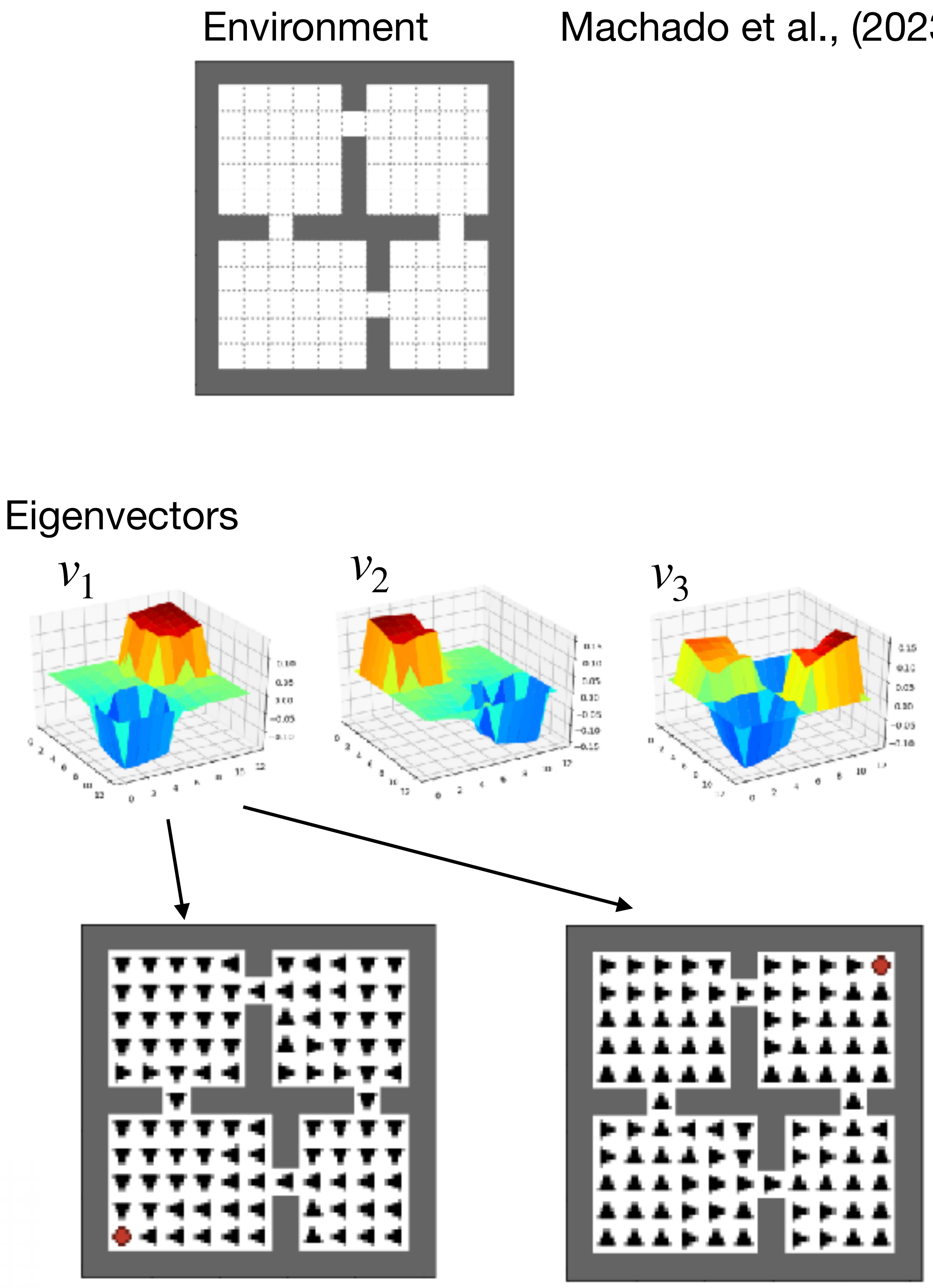


Stachenfeld, Botvinick, & Gershman (2017)

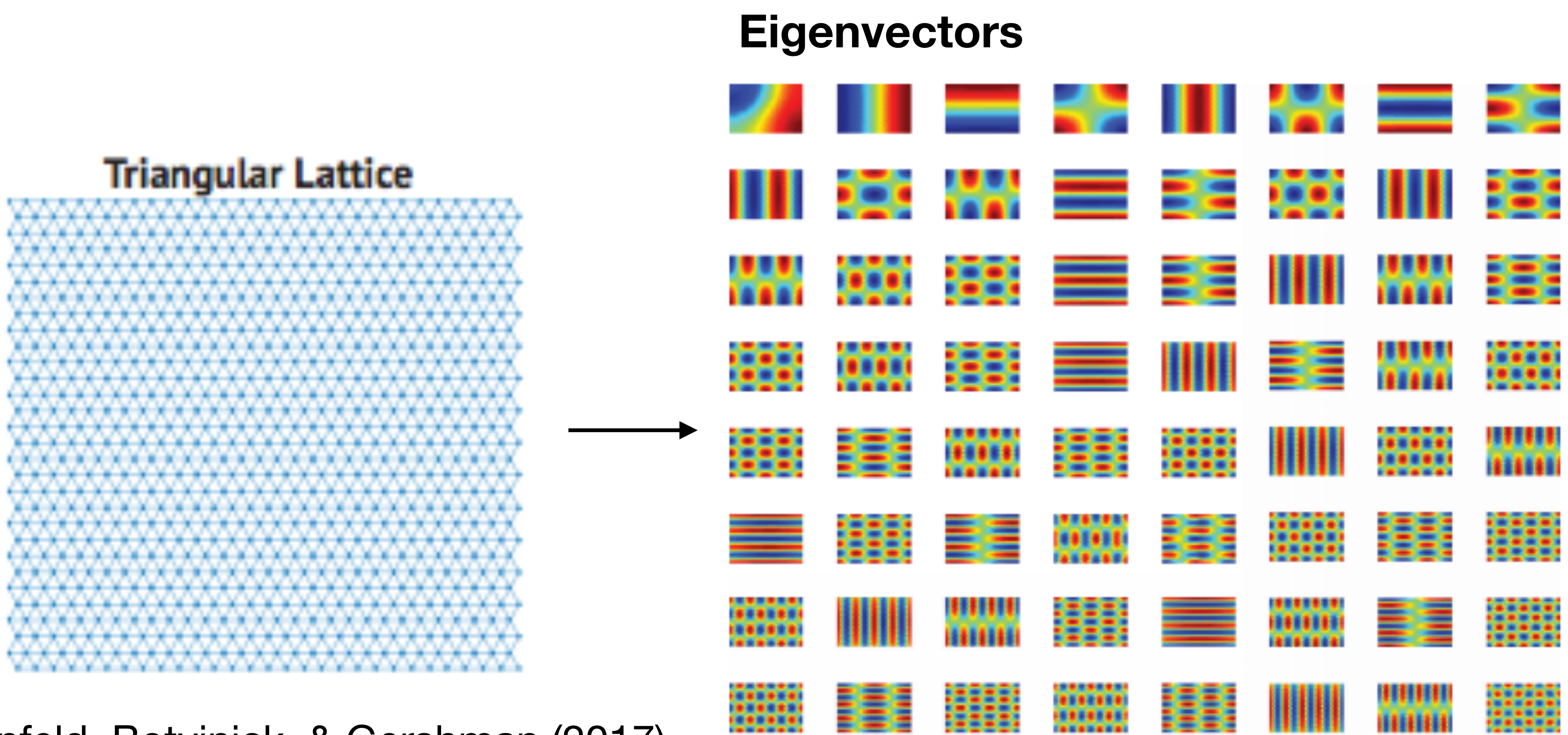


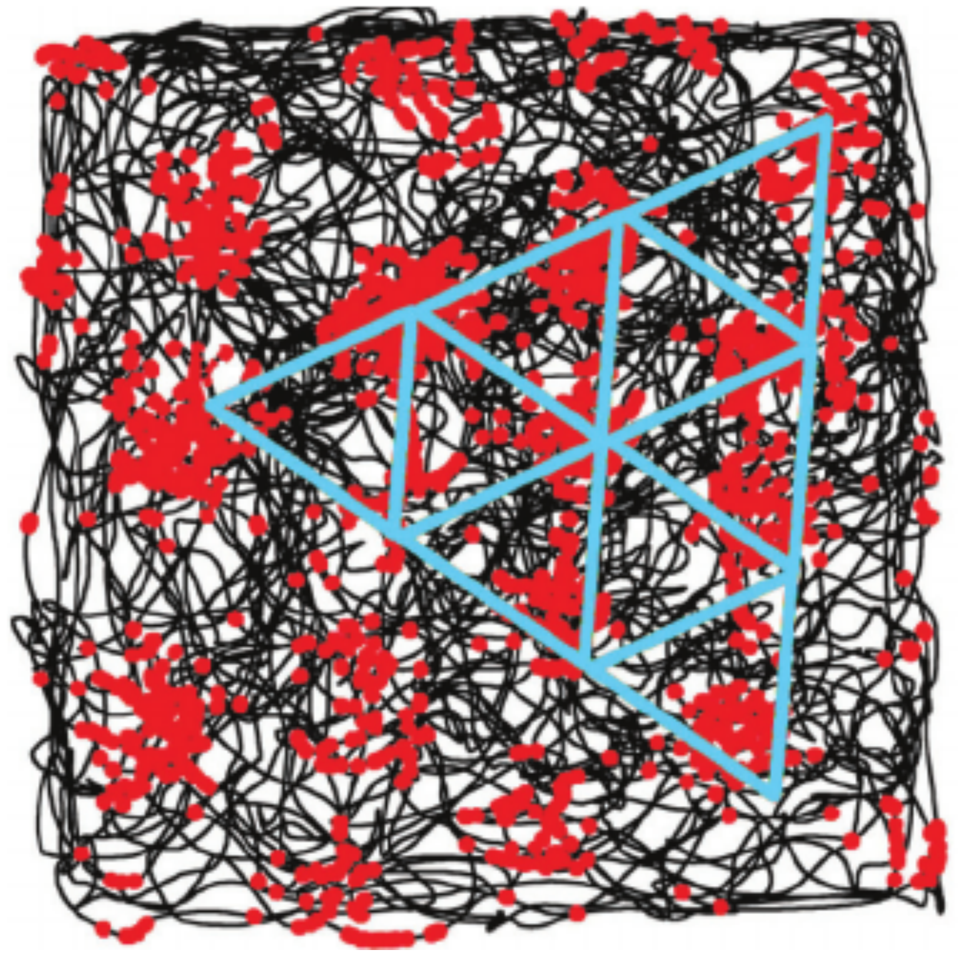
Machado et al., (2023)

The Eigenvalues of the SR



- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues
- For the SR, the different Eigenvectors capture different orthogonal patterns of state visitation
- In more regular environments, the Eigenvectors of the SR are gridded*, just like grid cells in the Entorhinal Cortex





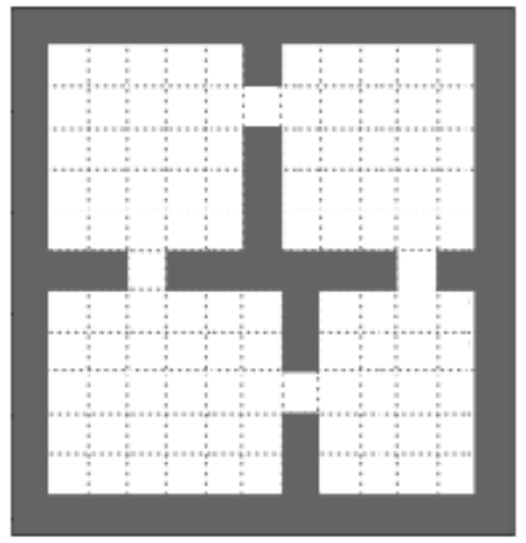
Stachenfeld, Botvinick, & Gershman (2017)

The Eigenvalues of the SR

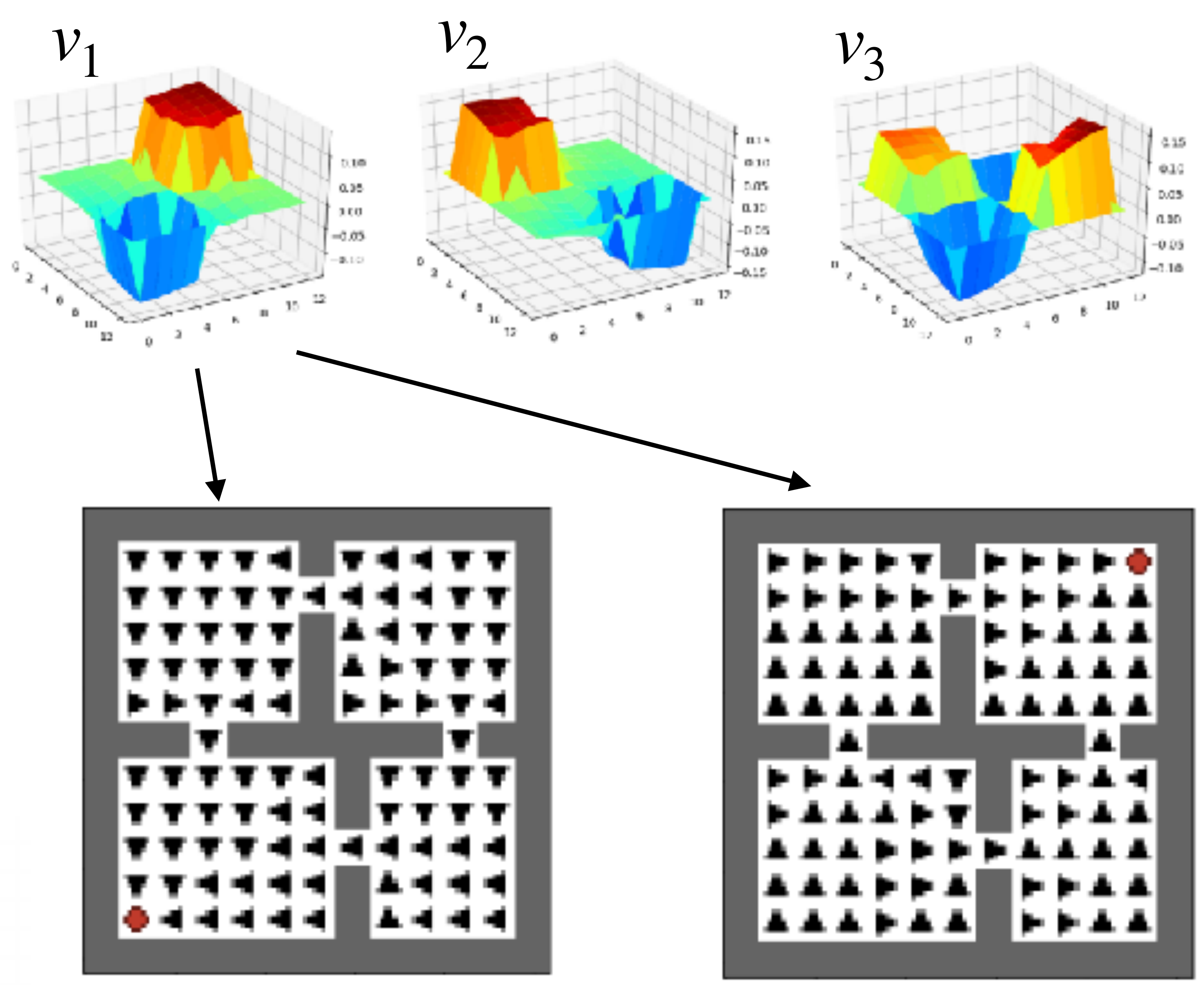
- Eigenvectors capture different dimensions of variability
  - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues
- For the SR, the different Eigenvectors capture different orthogonal patterns of state visitation
- In more regular environments, the Eigenvectors of the SR are gridded*, just like grid cells in the Entorhinal Cortex

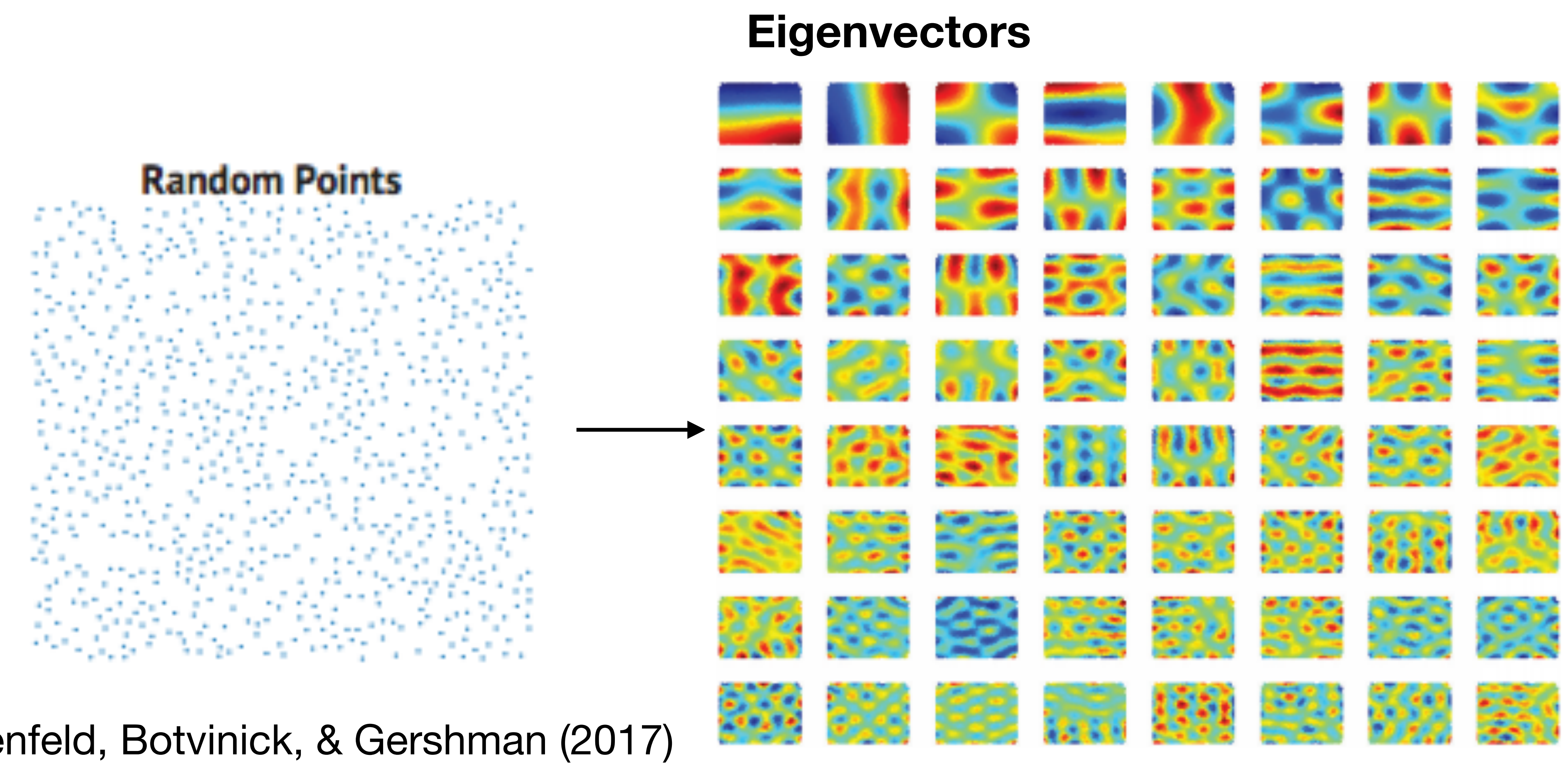
Environment

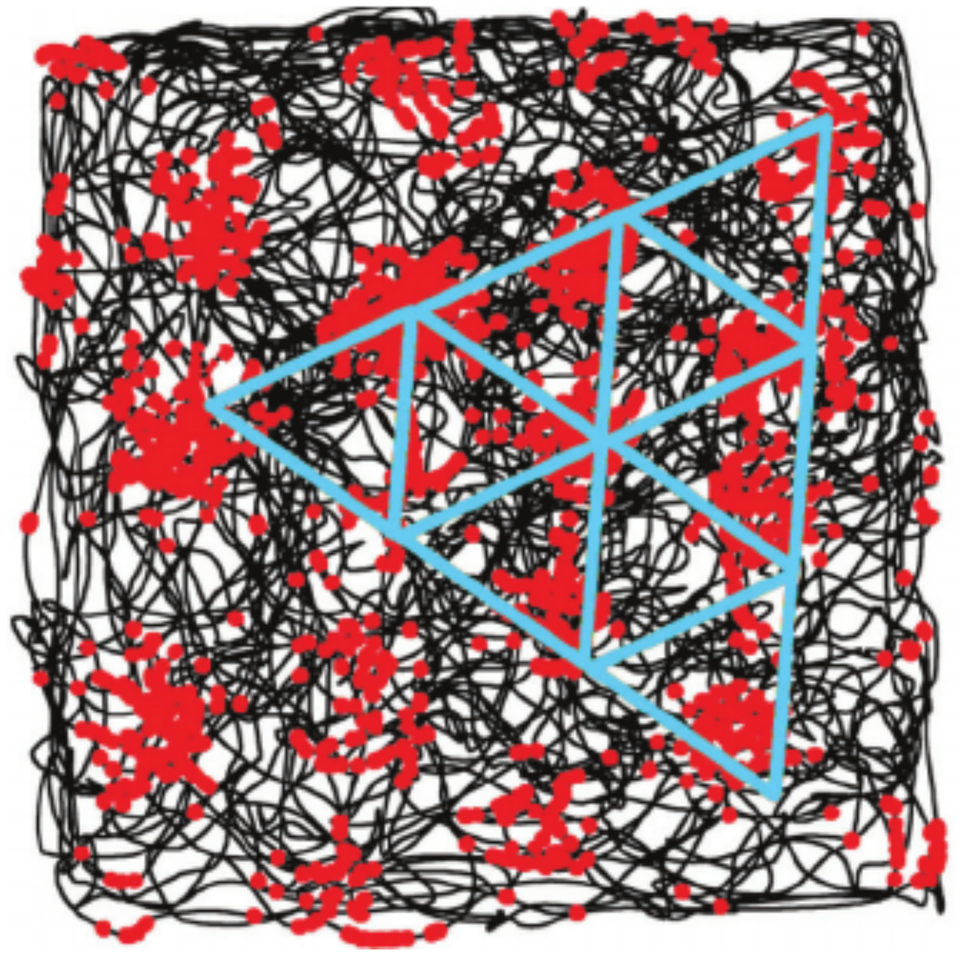
Machado et al., (2023)



Eigenvectors



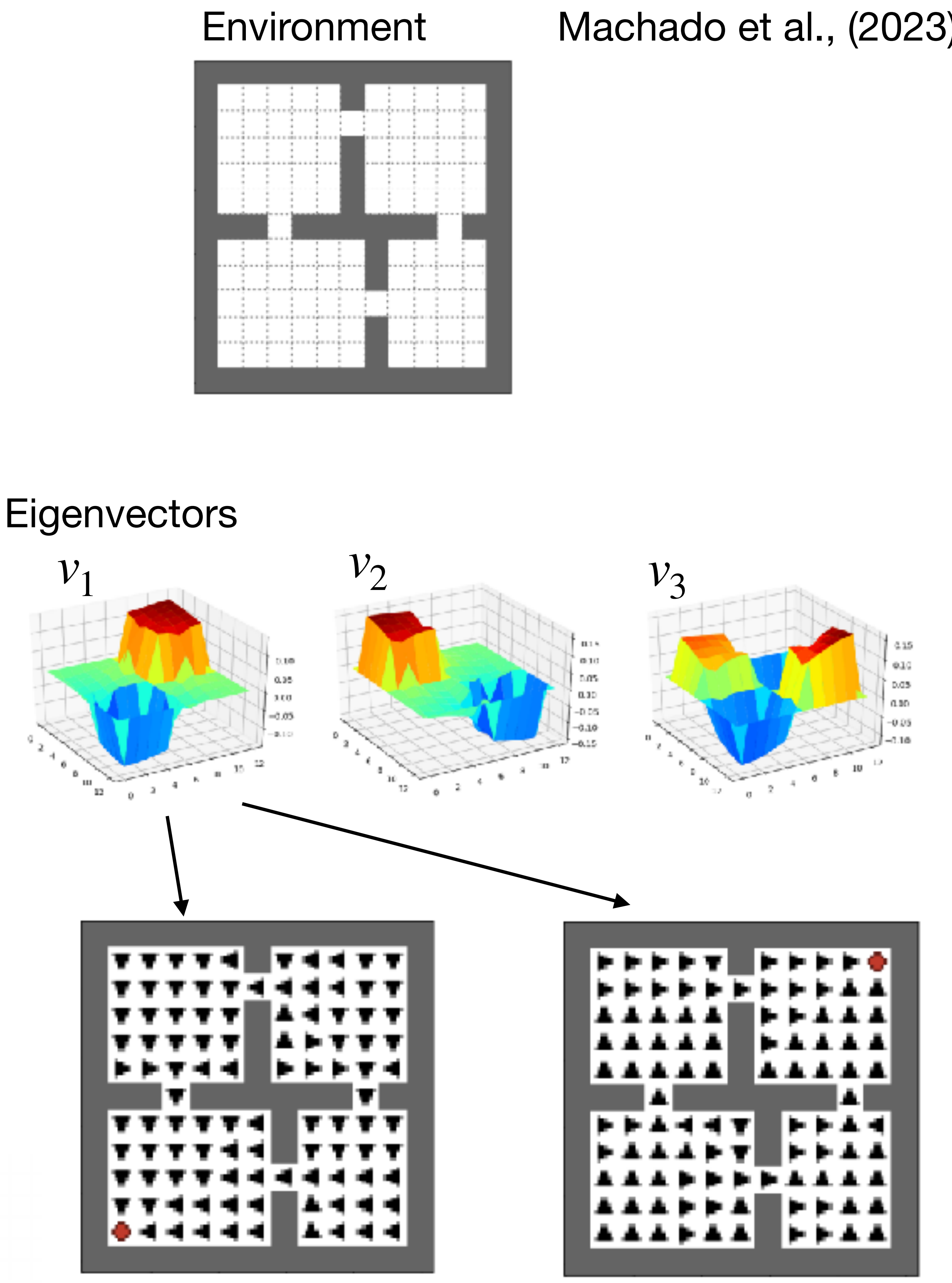




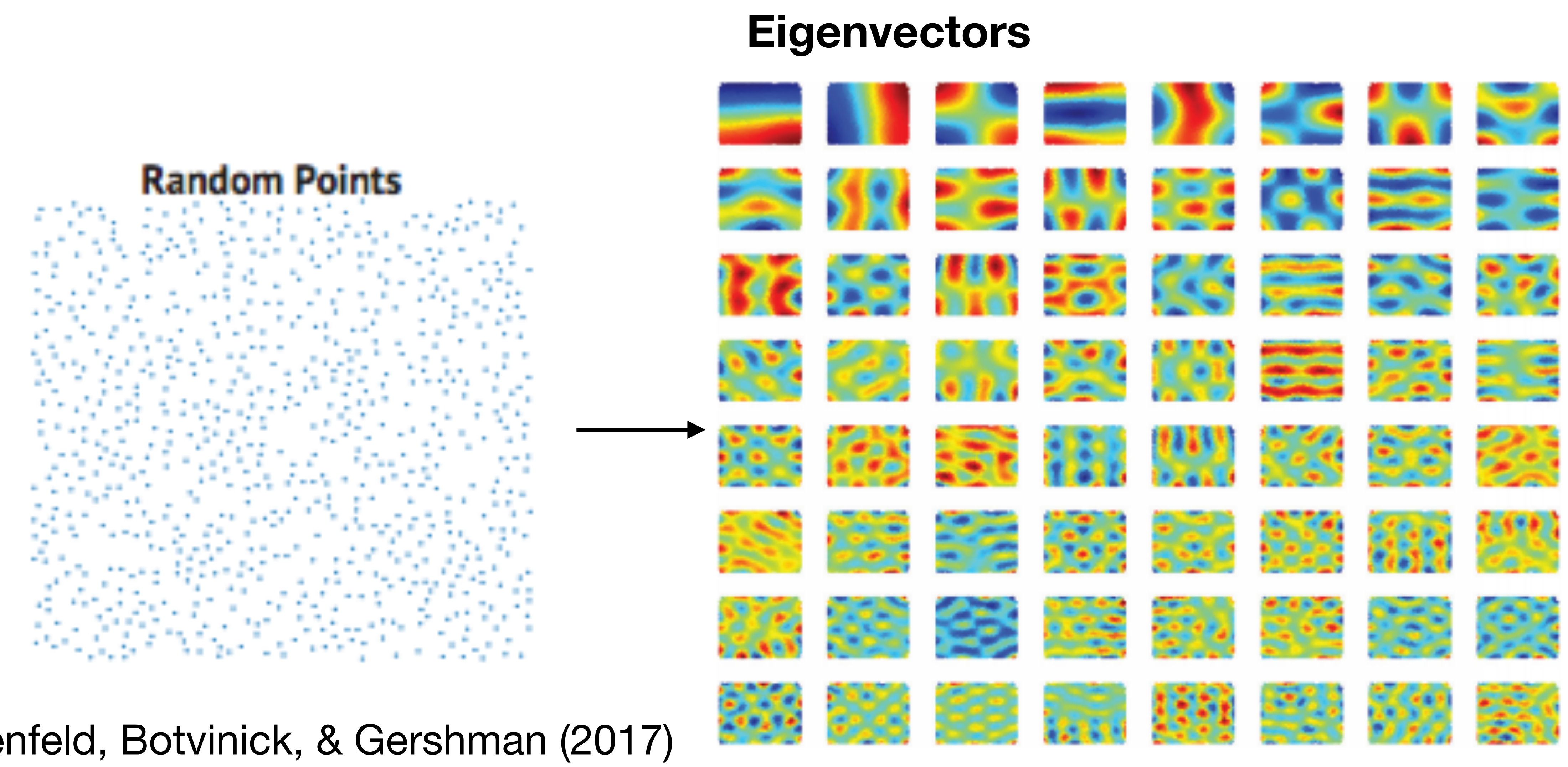
Stachenfeld, Botvinick, & Gershman (2017)

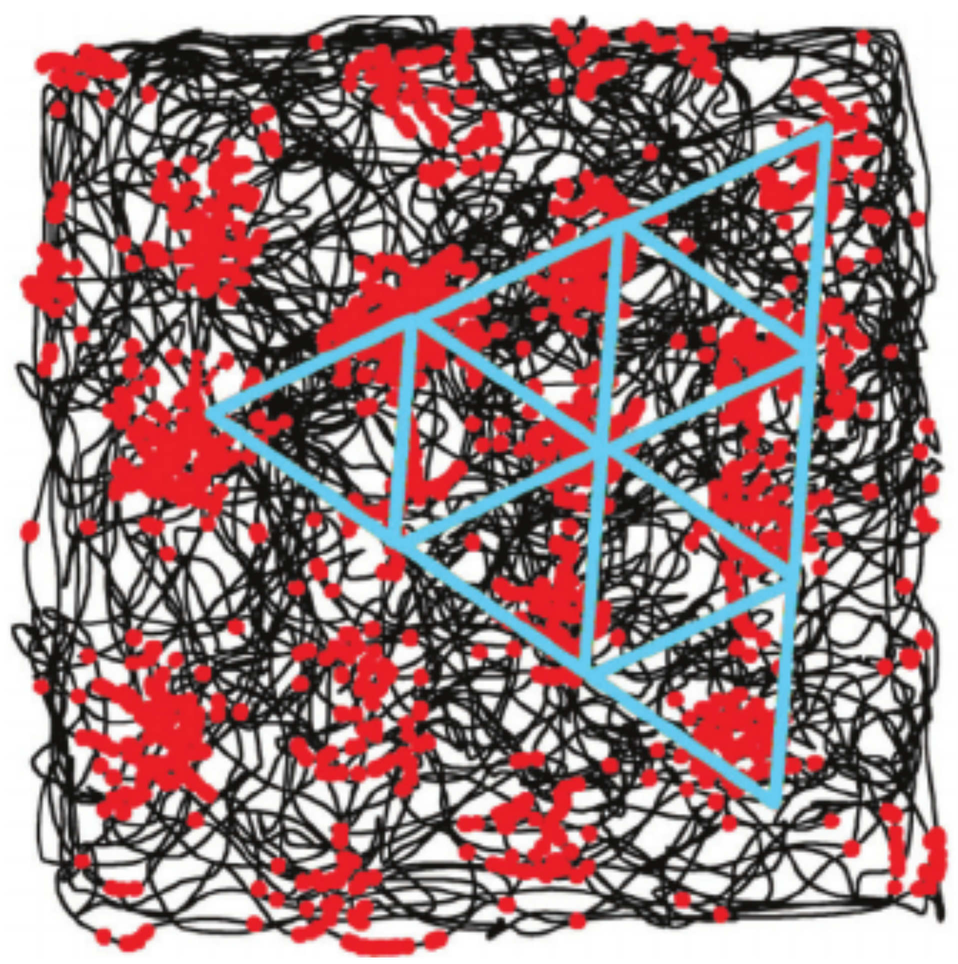
The Eigenvalues of the SR

- Eigenvectors capture different dimensions of variability
    - $M = V\Lambda V^{-1}$ where $v_i \in V$ are Eigenvectors and $\lambda_i \in \Lambda$ are the Eigenvalues
- For the SR, the different Eigenvectors capture different orthogonal patterns of state visitation
- In more regular environments, the Eigenvectors of the SR are gridded*, just like grid cells in the Entorhinal Cortex



Machado et al., (2023)





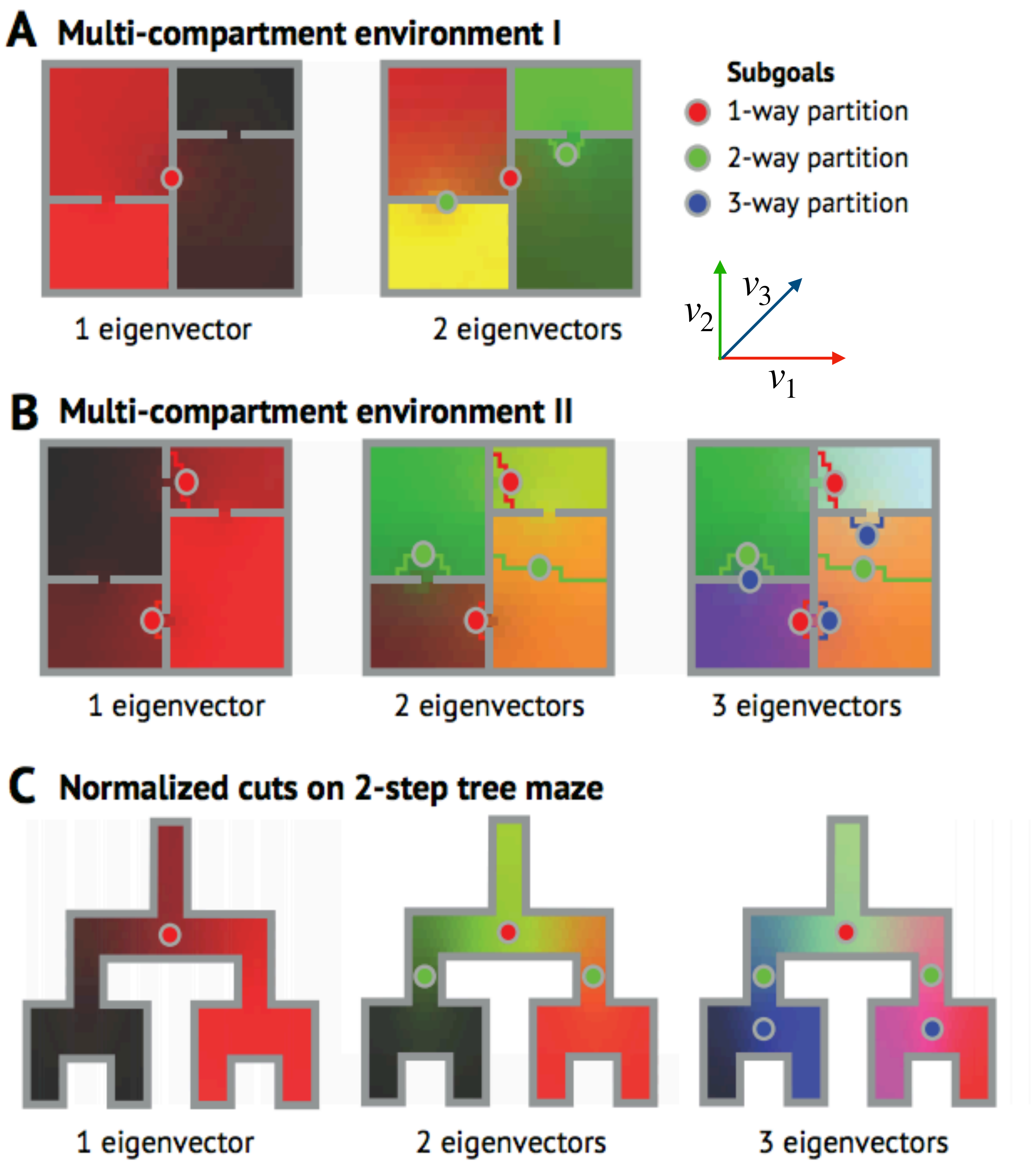
** Not unique to the SR, but any similarity metric that captures transition structure*

Stachenfeld, Botvinick, & Gershman (2017)

SR naturally identifies subgoals

- Eigenvectors capture subgoals (i.e., compartments in the environment)

Stachenfeld, Botvinick, & Gershman (*NatNeuro* 2017)

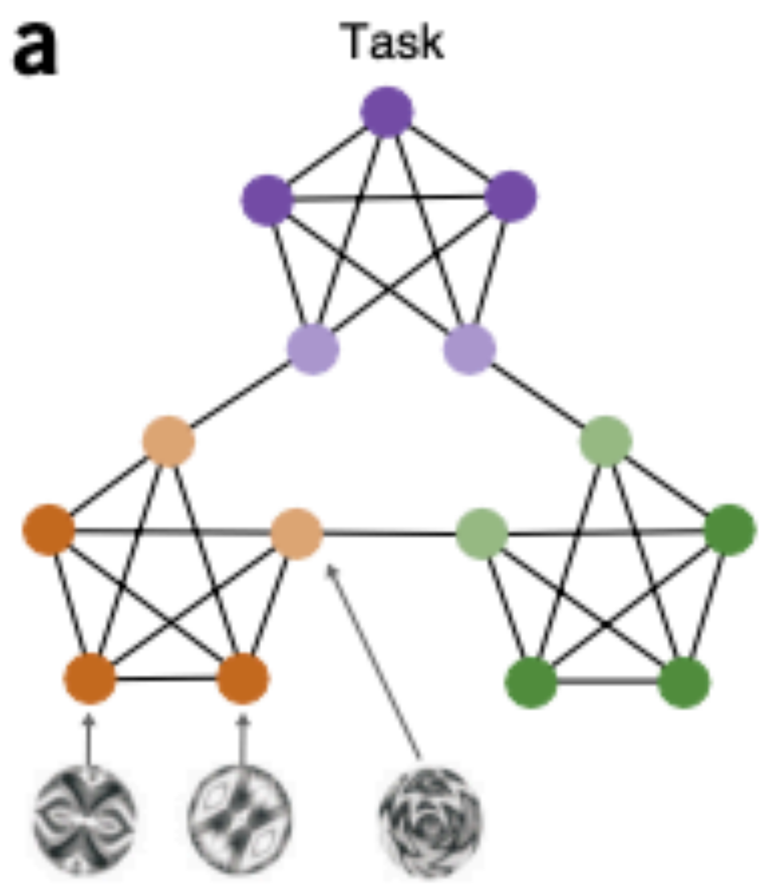


SR naturally identifies subgoals

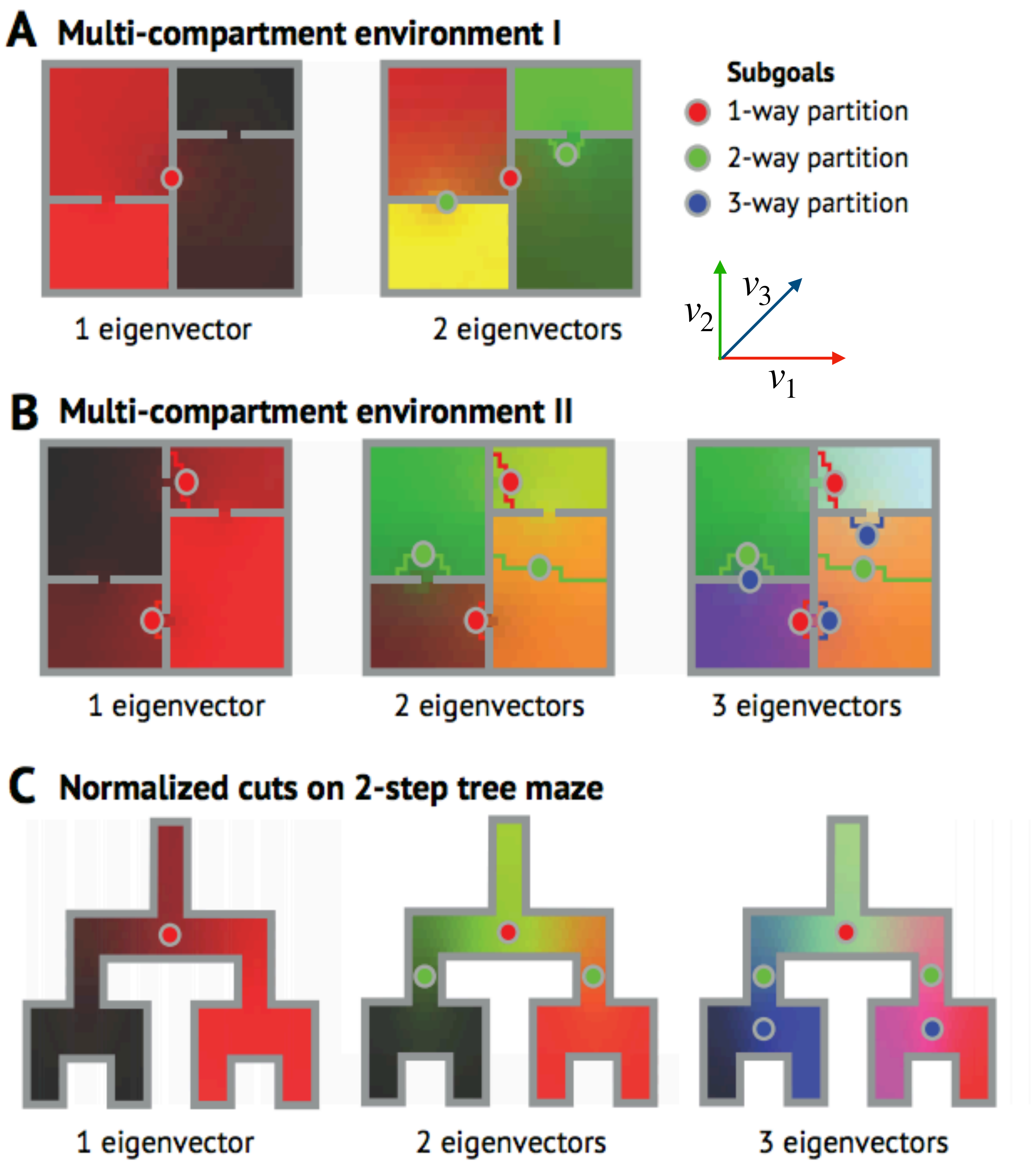
- Eigenvectors capture subgoals (i.e., compartments in the environment)

- These connectivity-based representations correspond to Hippocampal activity found in Schapiro et al. (2015) and Garvert et al. (2017)

Schapiro *et al.* (2015)



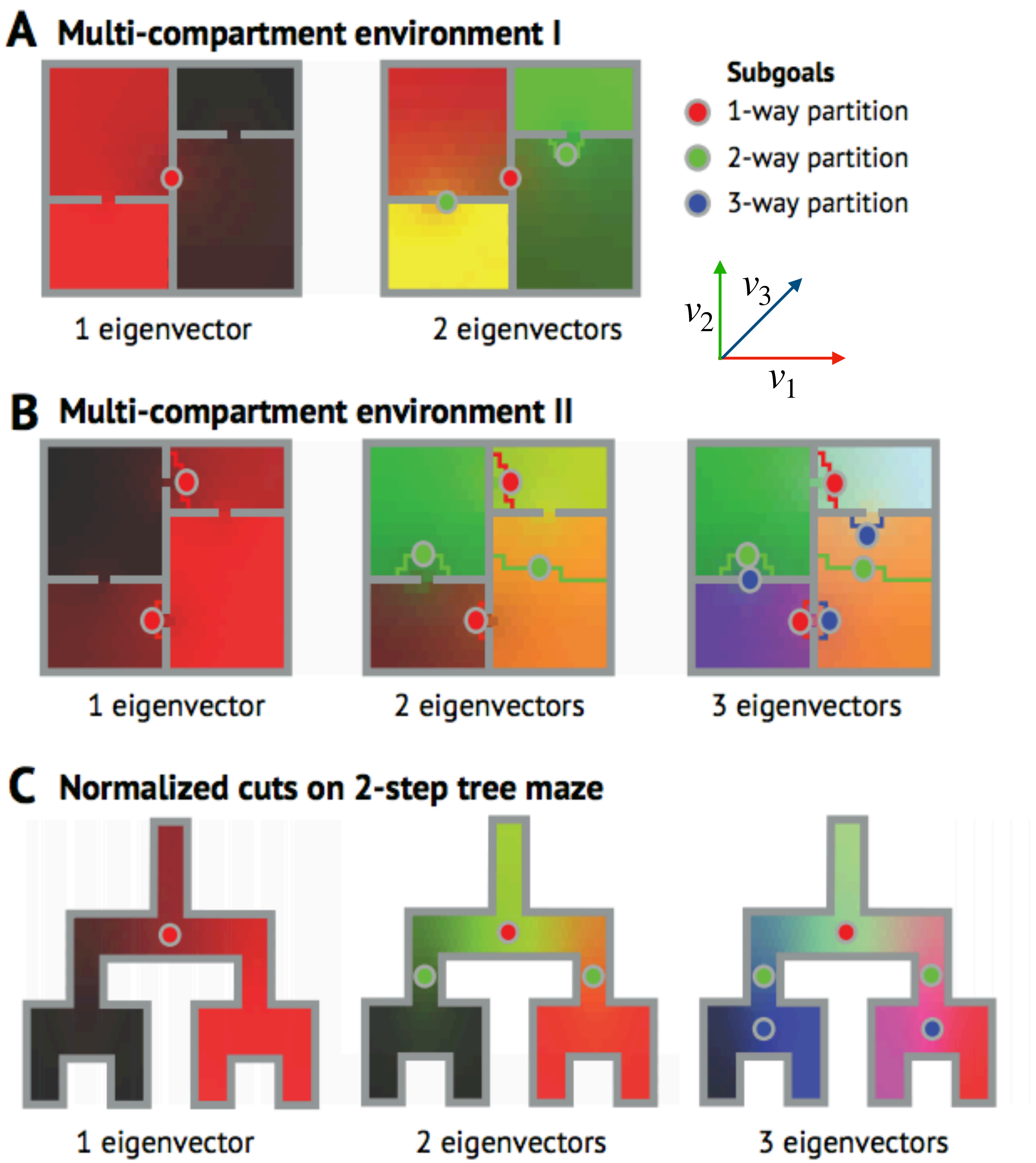
Stachenfeld, Botvinick, & Gershman (*NatNeuro* 2017)

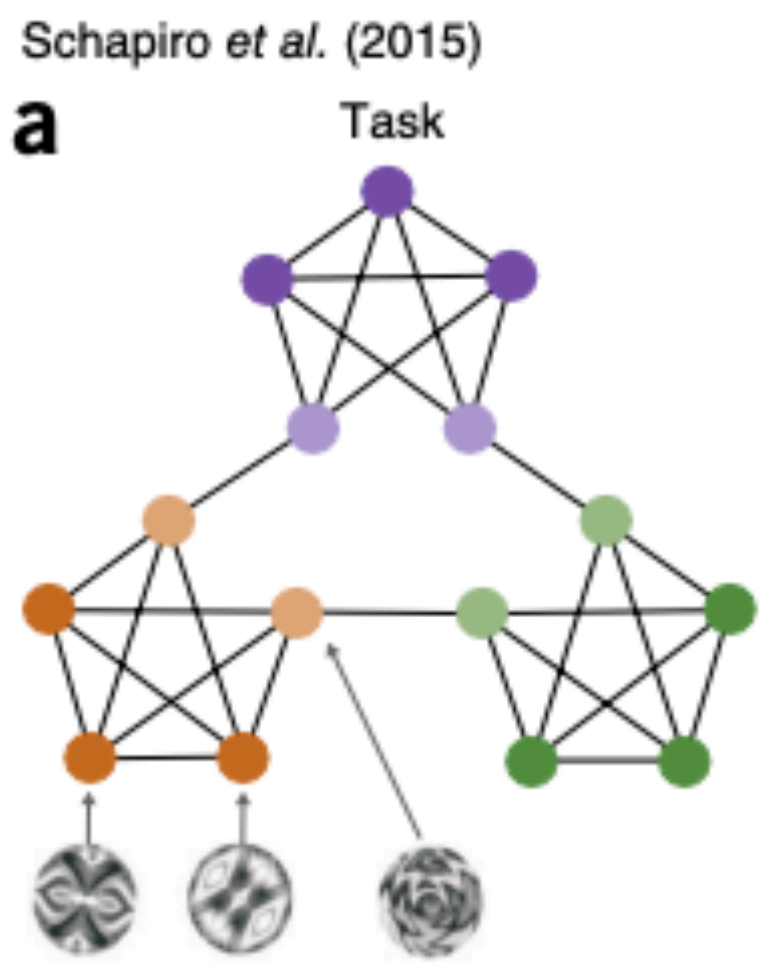


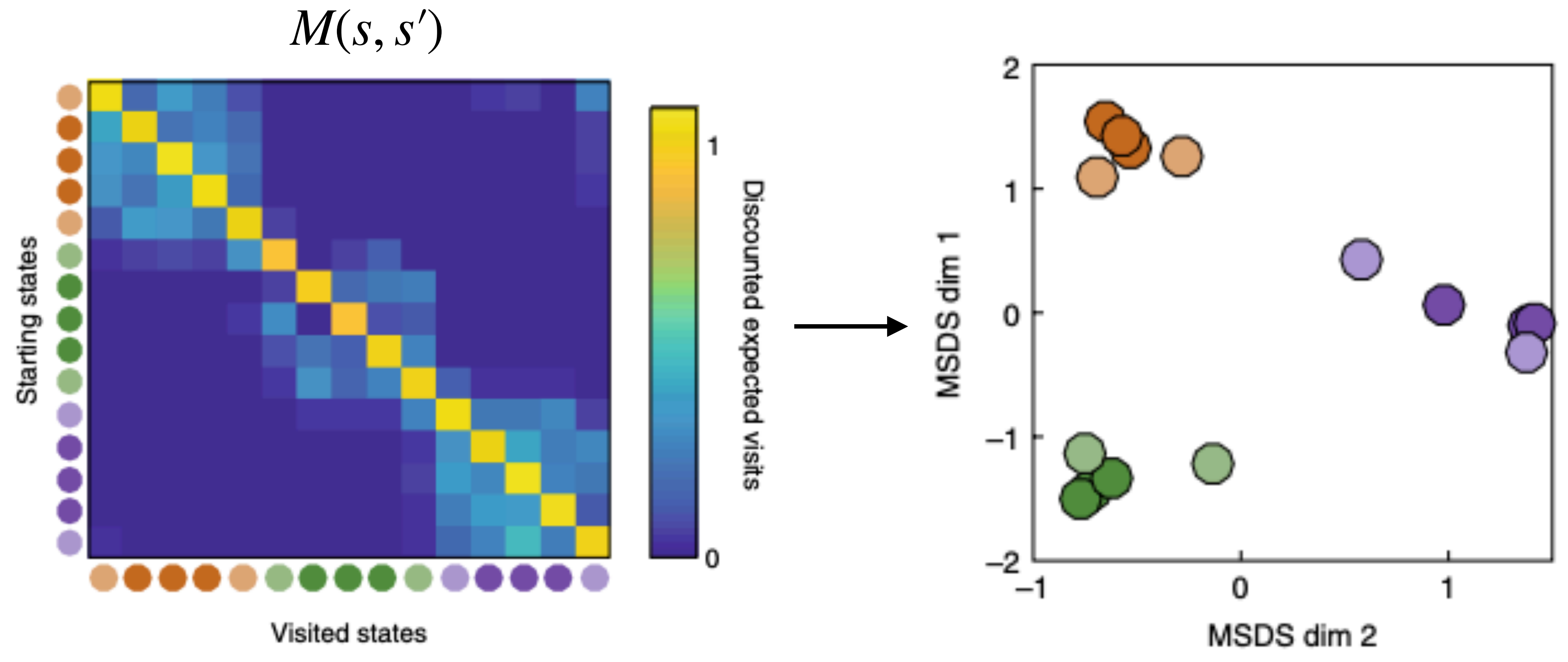
SR naturally identifies subgoals

- Eigenvectors capture subgoals (i.e., compartments in the environment)
- These connectivity-based representations correspond to Hippocampal activity found in Schapiro et al. (2015) and Garvert et al. (2017)

Stachenfeld, Botvinick, & Gershman (*NatNeuro* 2017)





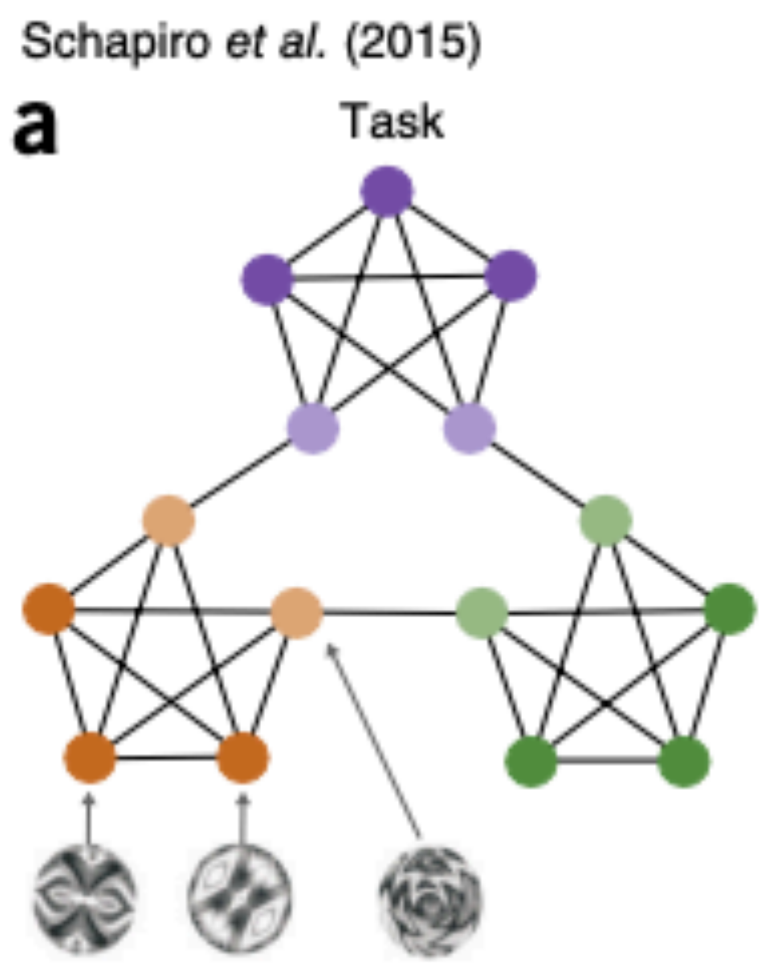


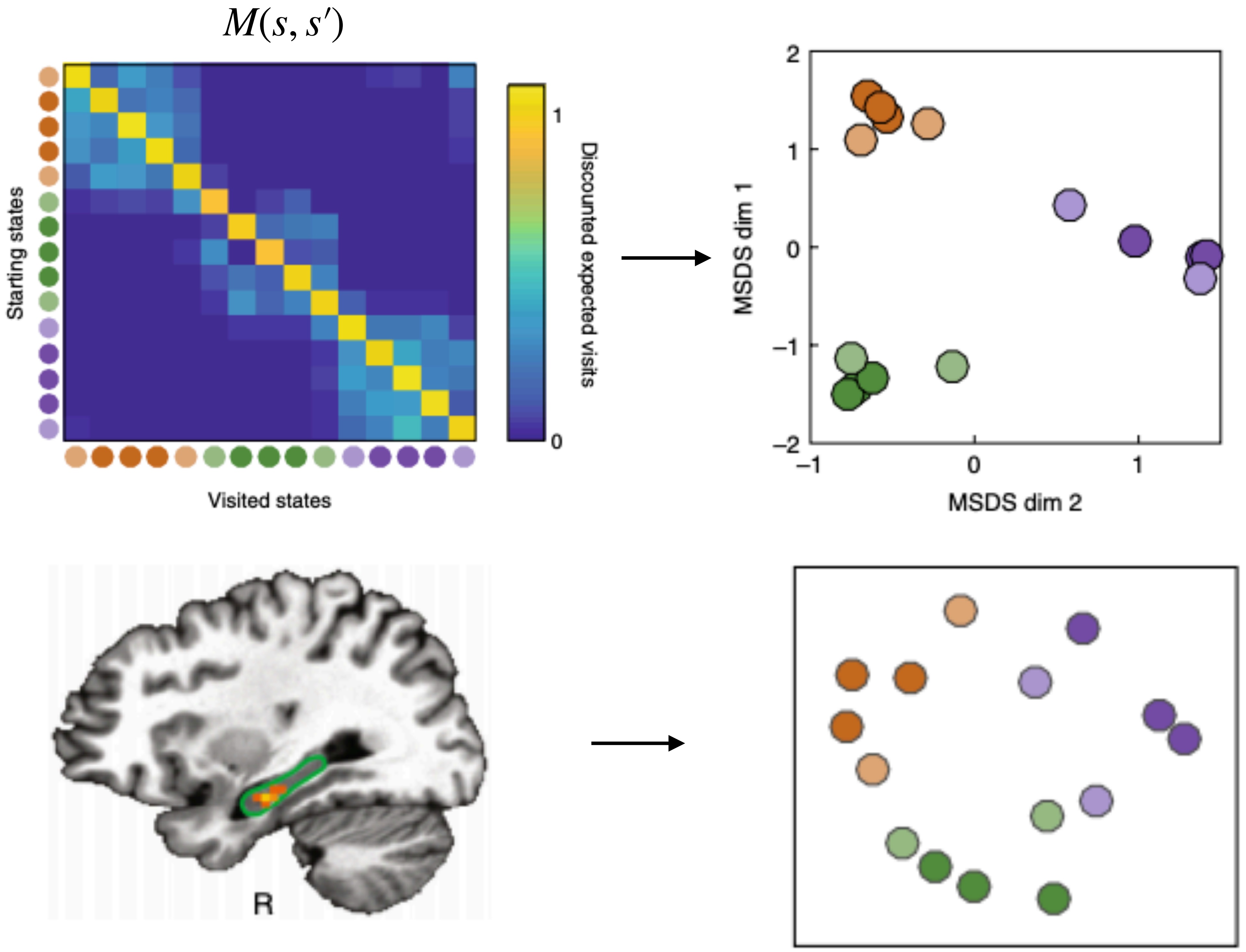
SR naturally identifies subgoals

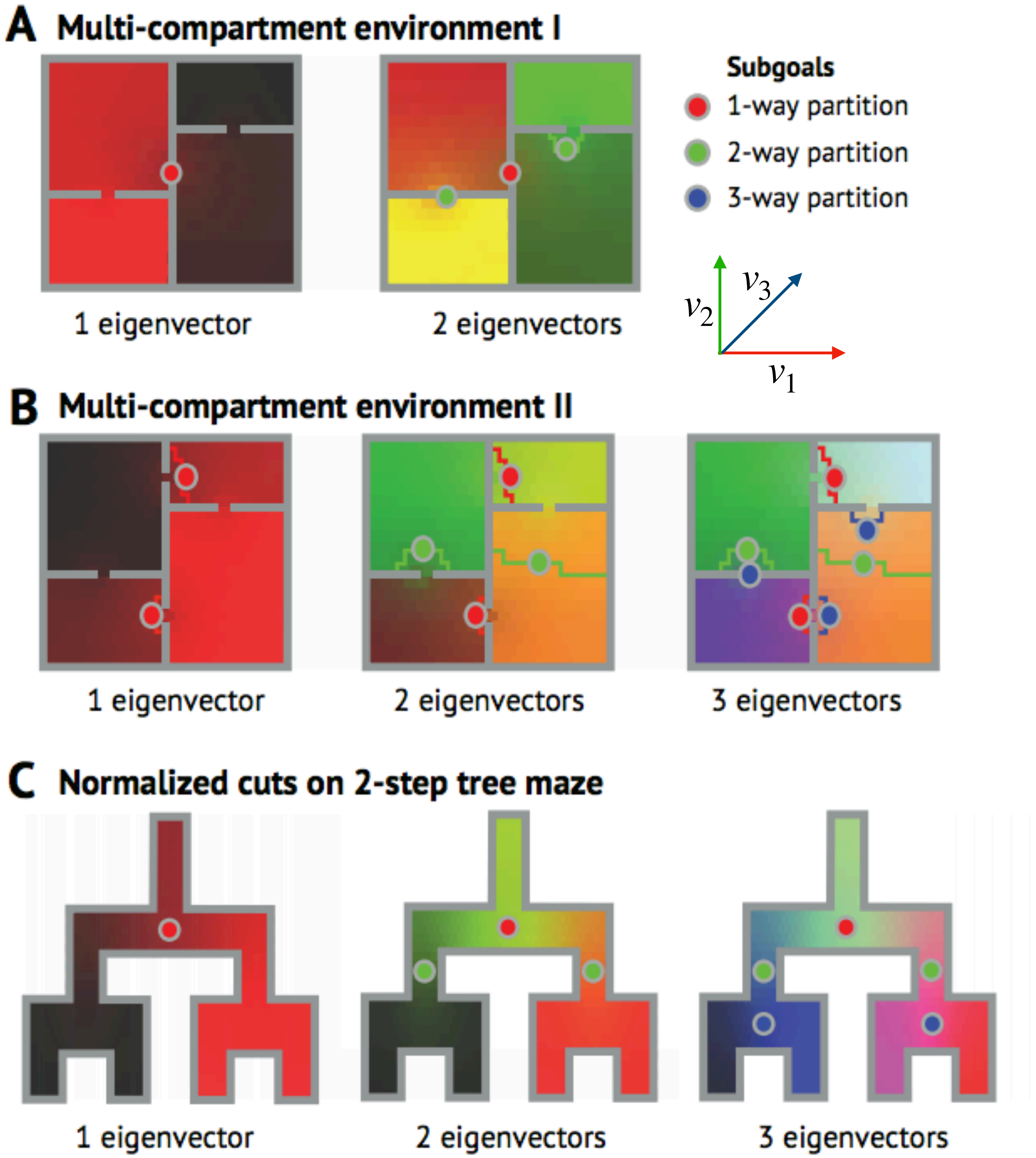
- Eigenvectors capture subgoals (i.e., compartments in the environment)

- These connectivity-based representations correspond to Hippocampal activity found in Schapiro et al. (2015) and Garvert et al. (2017)

Stachenfeld, Botvinick, & Gershman (*NatNeuro* 2017)

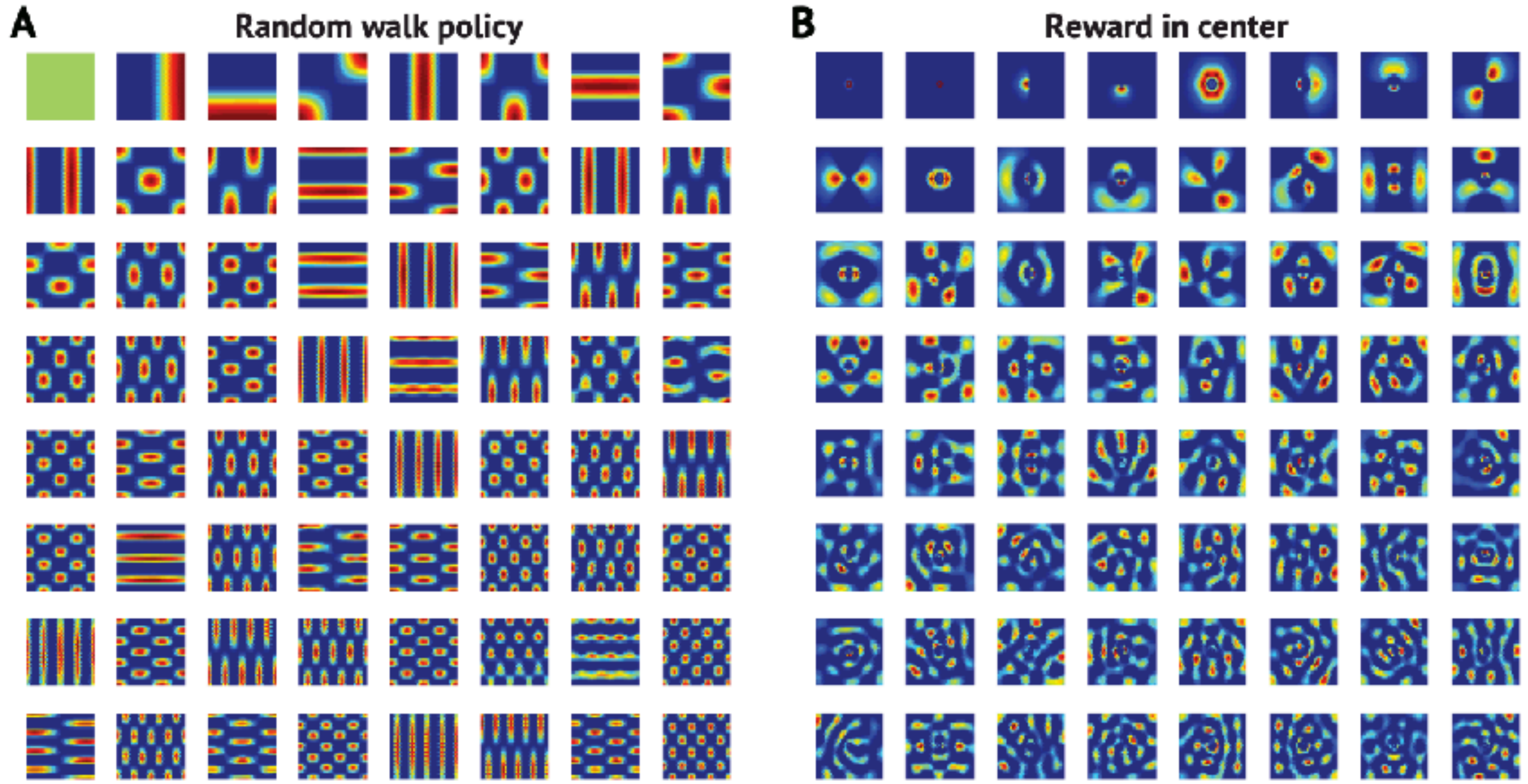




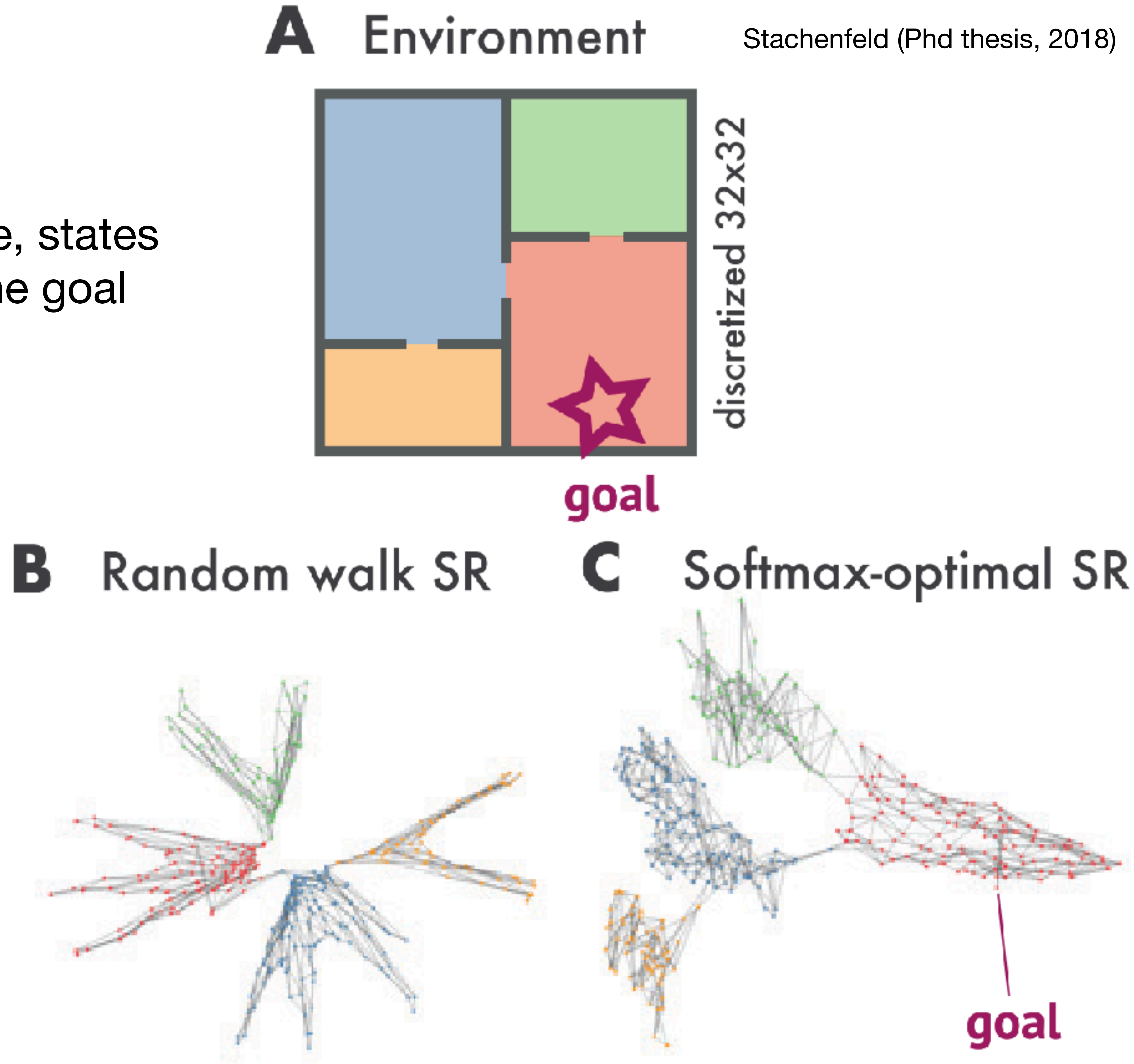


SR is sensitive to policy

- SR learned under different policies learn different representations
- Under a softmax-optimal policy seeking the goal state, states become organized according to their distance from the goal
- But it also makes them less grid-like



Stachenfeld et al., (2017)



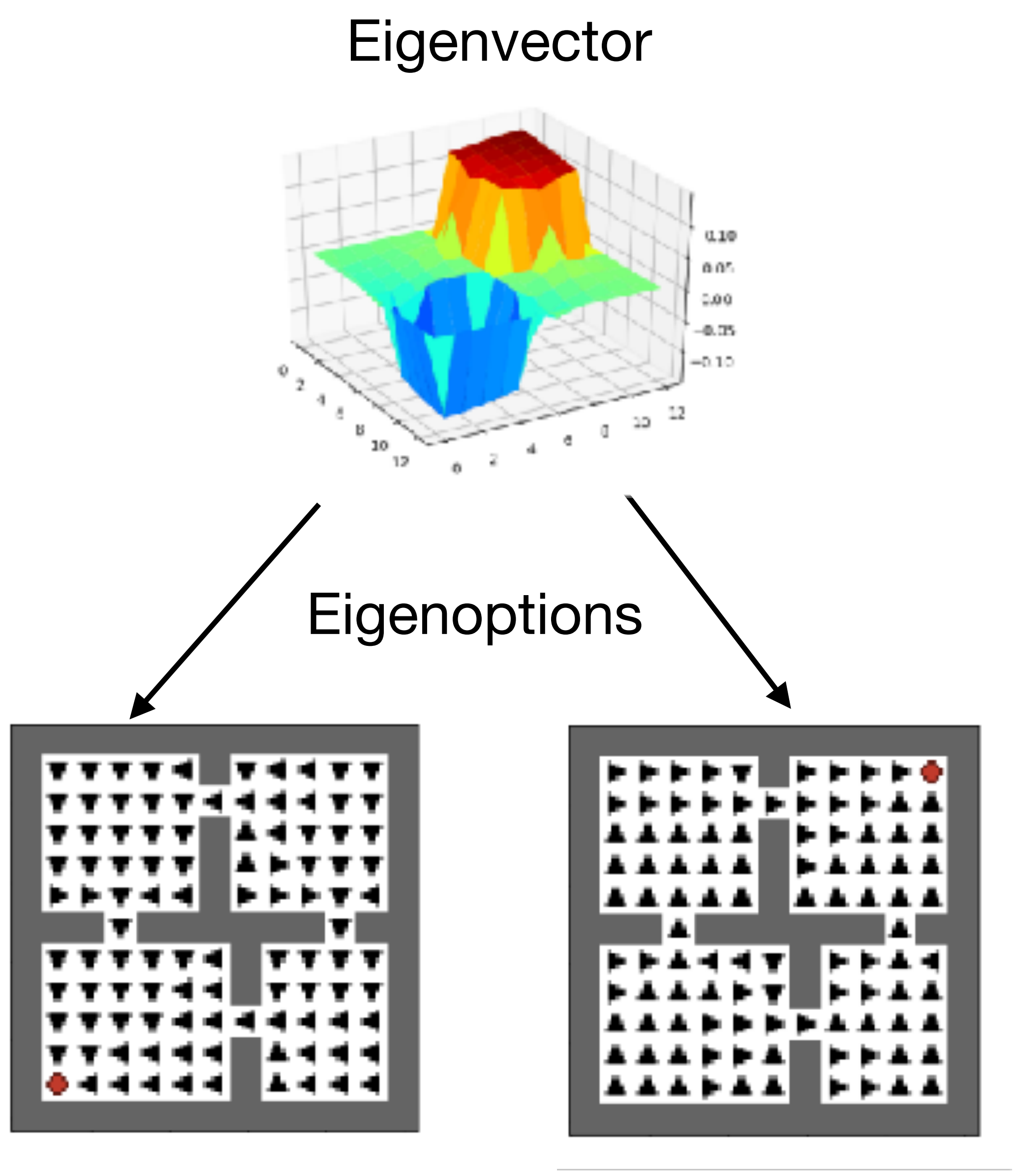
Stachenfeld (Phd thesis, 2018)

SR for option discovery

- “Options framework” in RL corresponds to learning extended sequences of actions instead of only a single action at a time
  - Options: Make coffee vs. make tea
  - Actions: move left leg, move right leg, …. move wrist 34 degrees
- SR naturally discovers *Eigenoptions*
- More recent work has identified functional sequences of actions this way

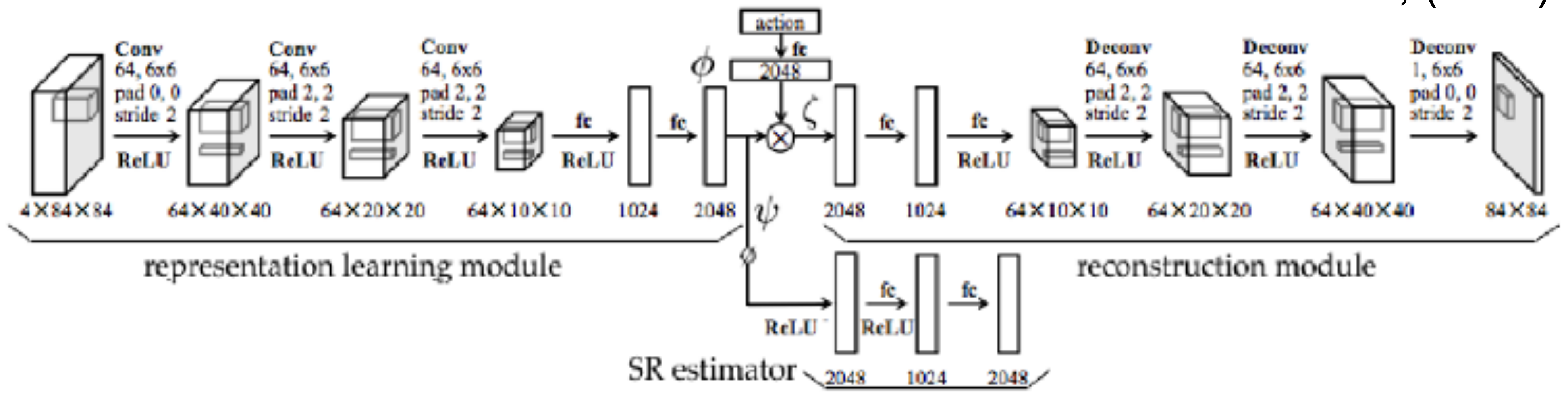
SR for option discovery

- "Options framework" in RL corresponds to learning extended sequences of actions instead of only a single action at a time
  - Options: Make coffee vs. make tea
  - Actions: move left leg, move right leg, …. move wrist 34 degrees
- SR naturally discovers *Eigenoptions*
- More recent work has identified functional sequences of actions this way



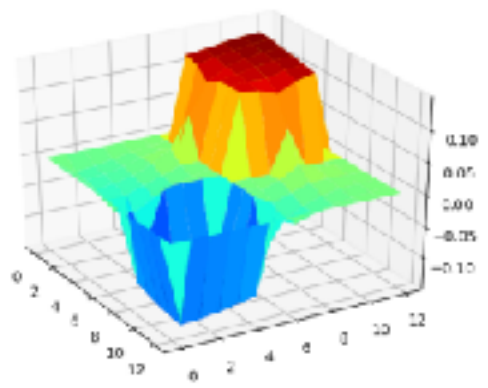
SR for option discovery

Machado et al., (2018)

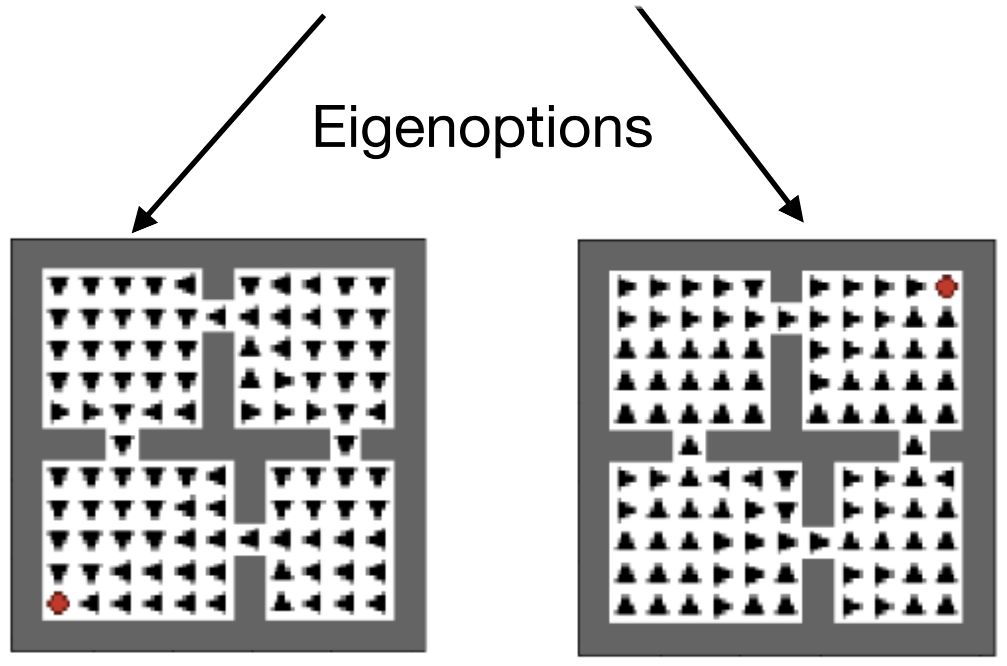


- "Options framework" in RL corresponds to learning extended sequences of actions instead of only a single action at a time
  - Options: Make coffee vs. make tea
  - Actions: move left leg, move right leg, …. move wrist 34 degrees
- SR naturally discovers *Eigenoptions*
- More recent work has identified functional sequences of actions this way

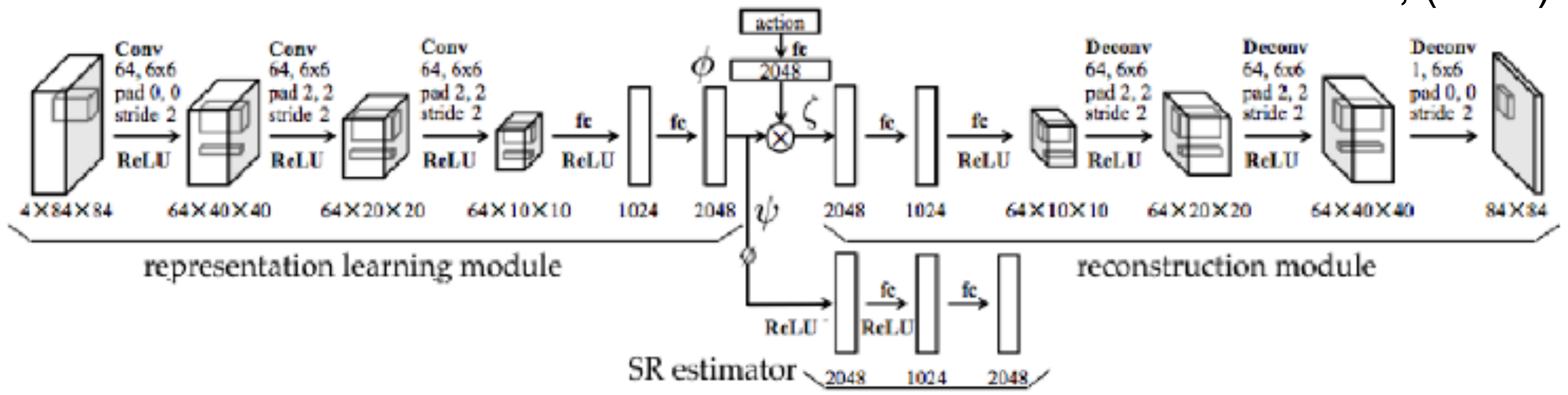
Eigenvector



Eigenoptions



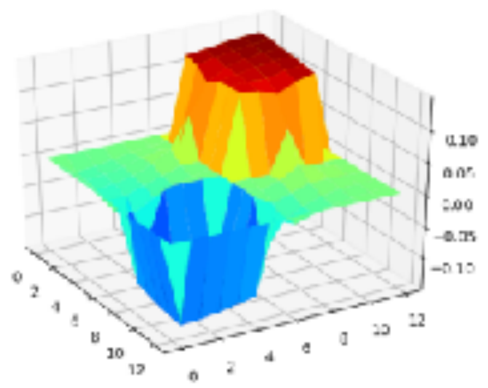
Machado et al., (2018)



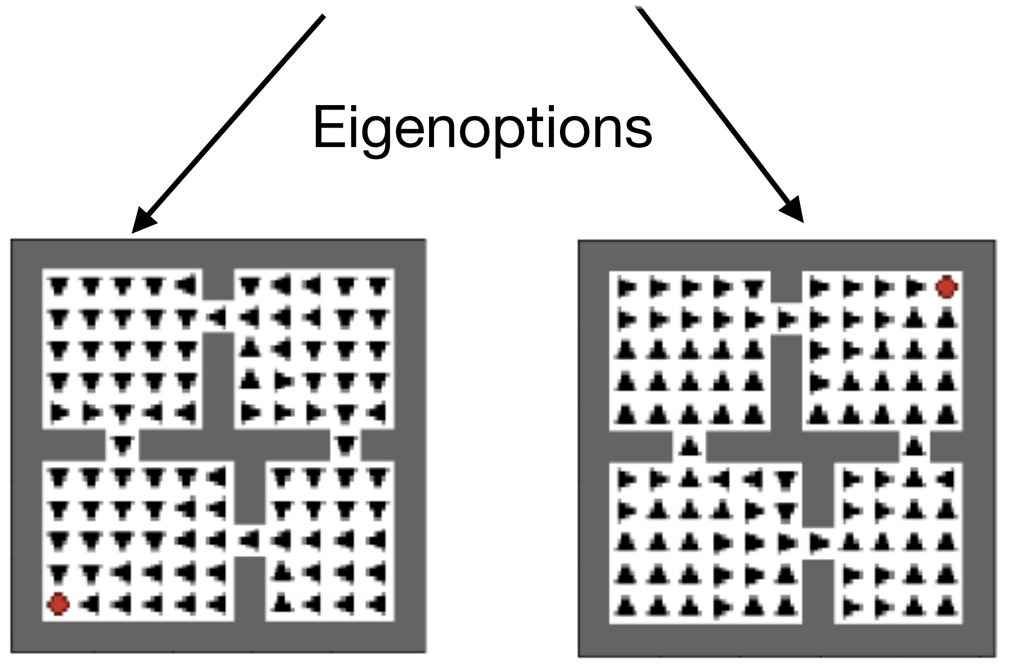
SR for option discovery

- “Options framework” in RL corresponds to learning extended sequences of actions instead of only a single action at a time
  - Options: Make coffee vs. make tea
  - Actions: move left leg, move right leg, …. move wrist 34 degrees
- SR naturally discovers *Eigenoptions*
- More recent work has identified functional sequences of actions this way

Eigenvector

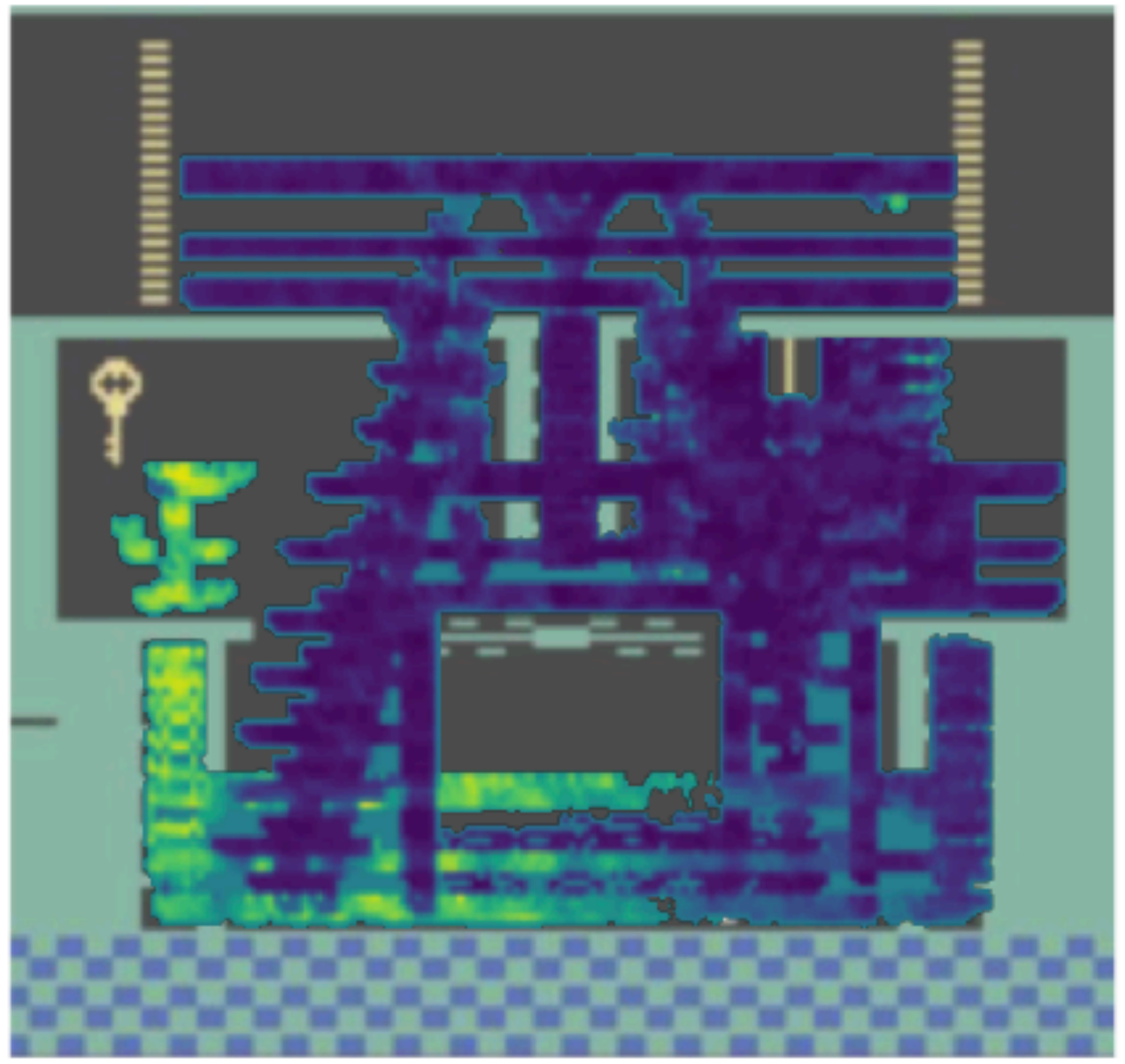


Eigenoptions

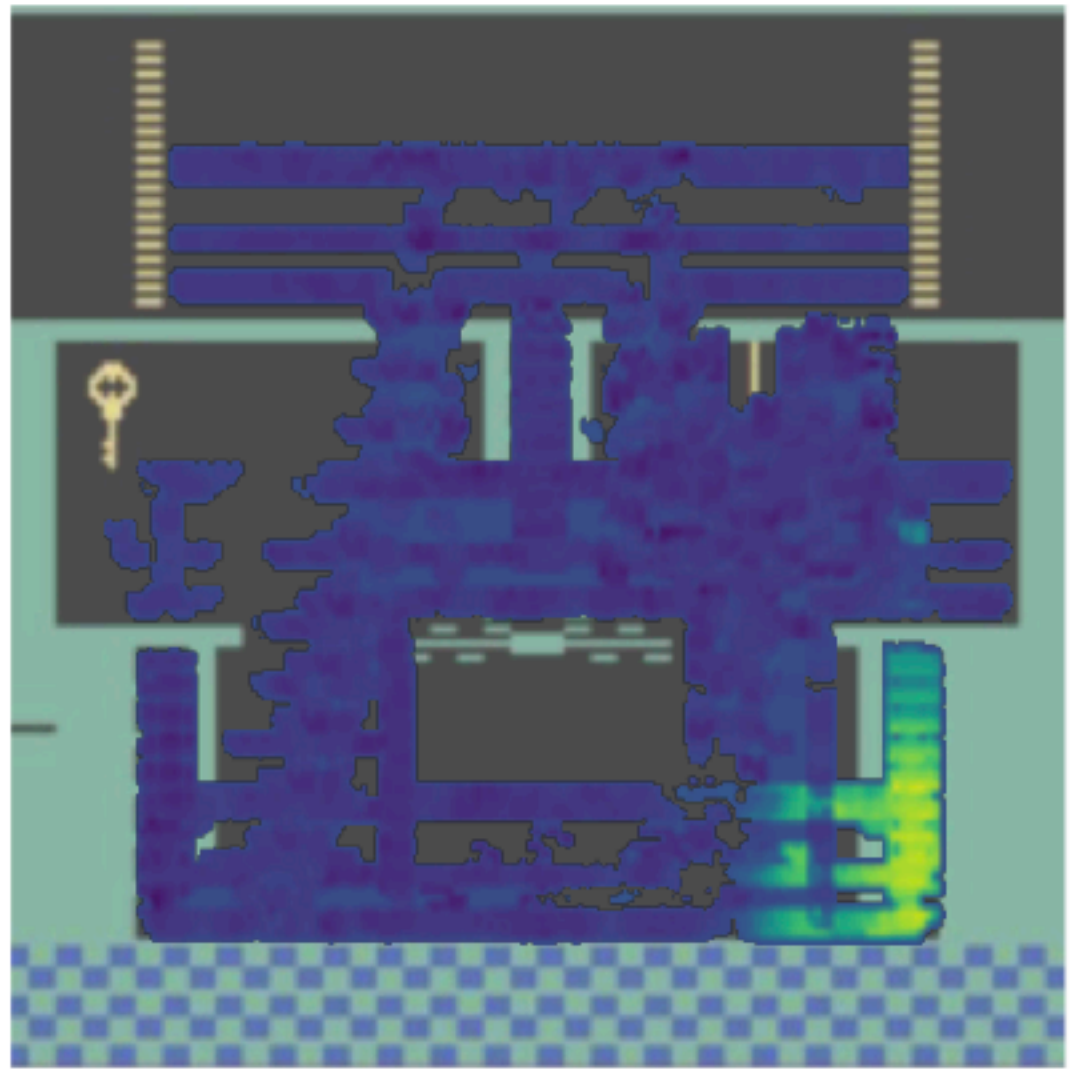


Montezuma’s Revenge

Eigenfunction #1



Eigenfunction #2



Klissarov & Machado (2023)

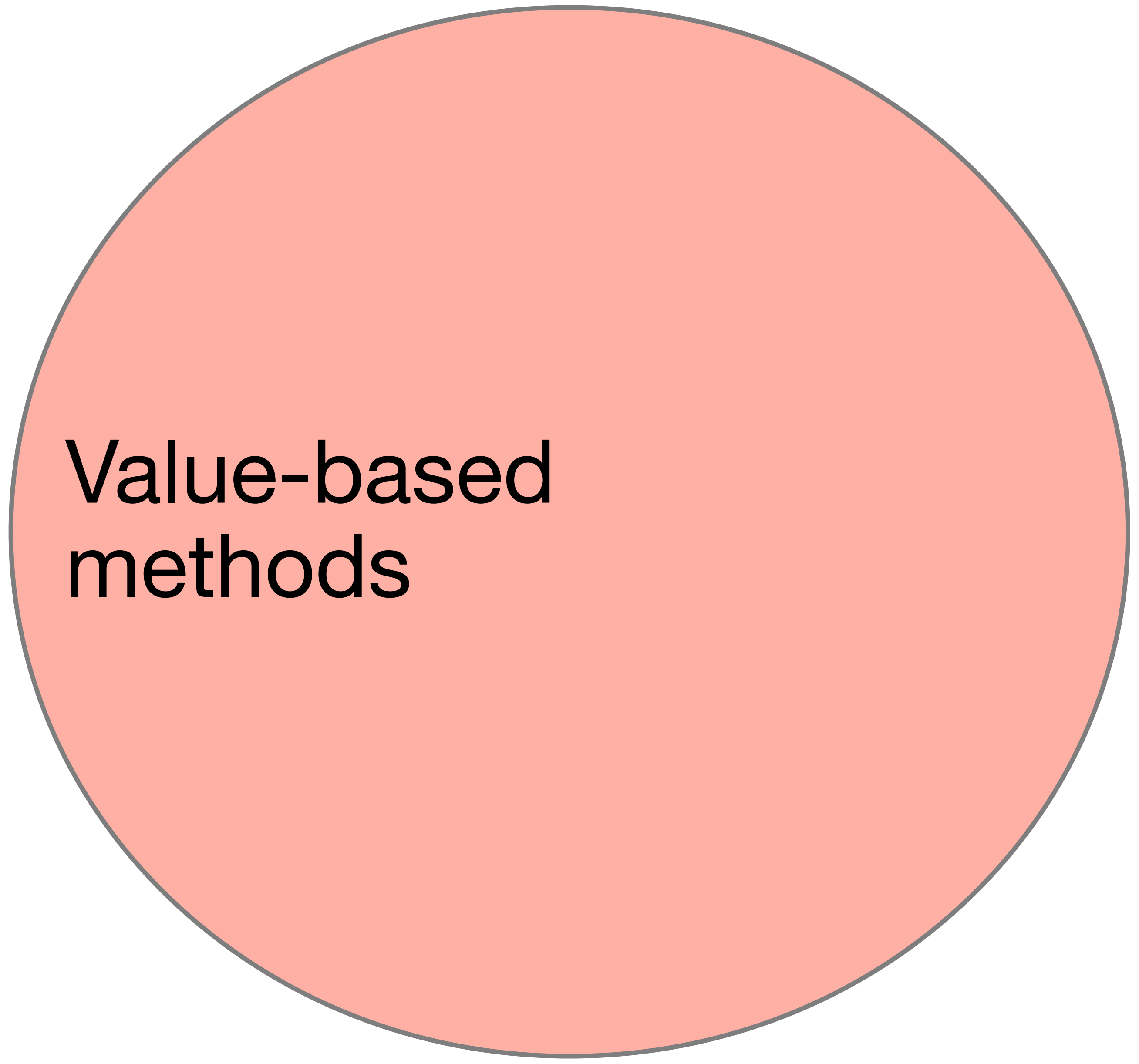
Summary

Summary

- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor**-**Critic**

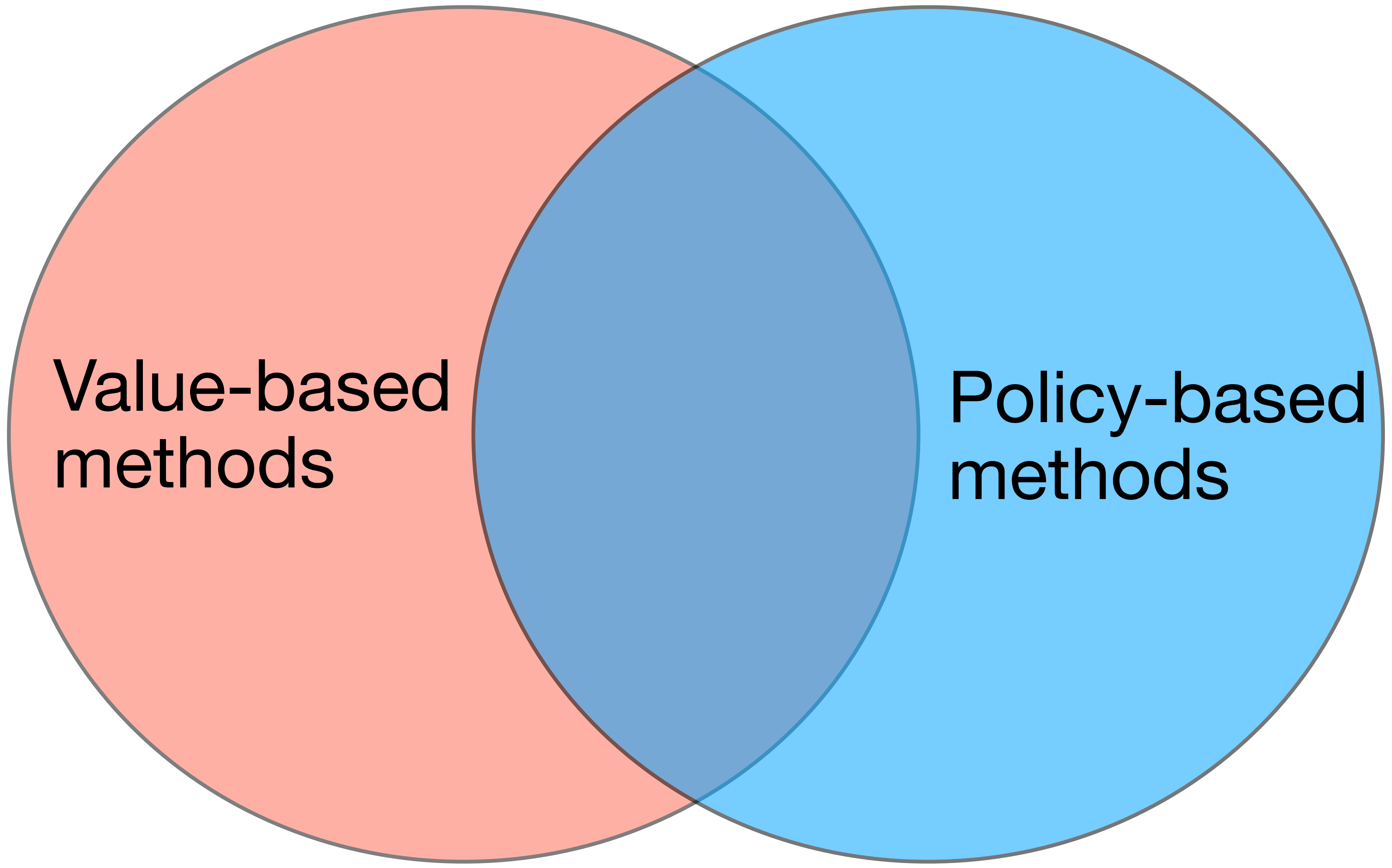
Summary

- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor**-**Critic**



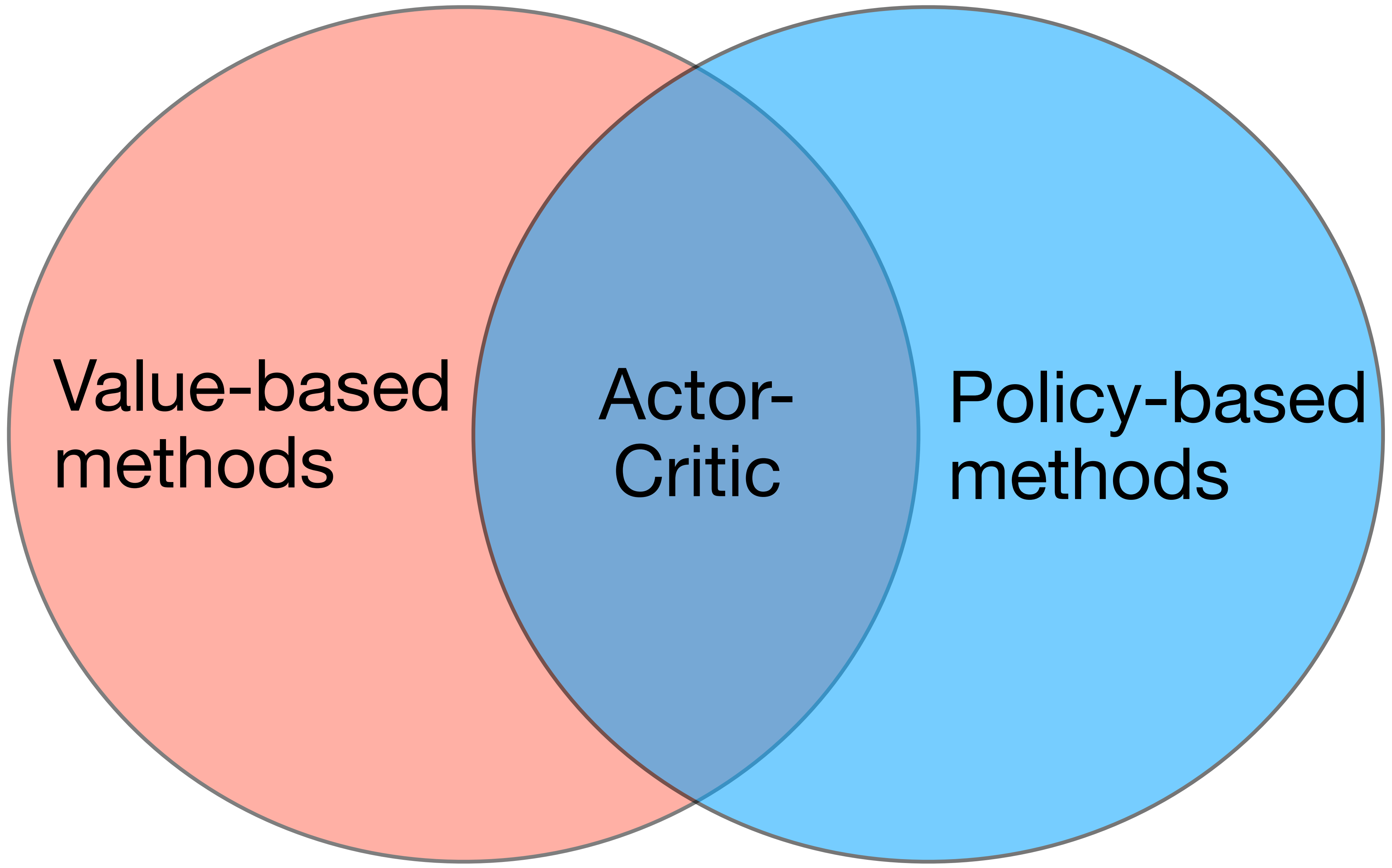
Summary

- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor**-**Critic**



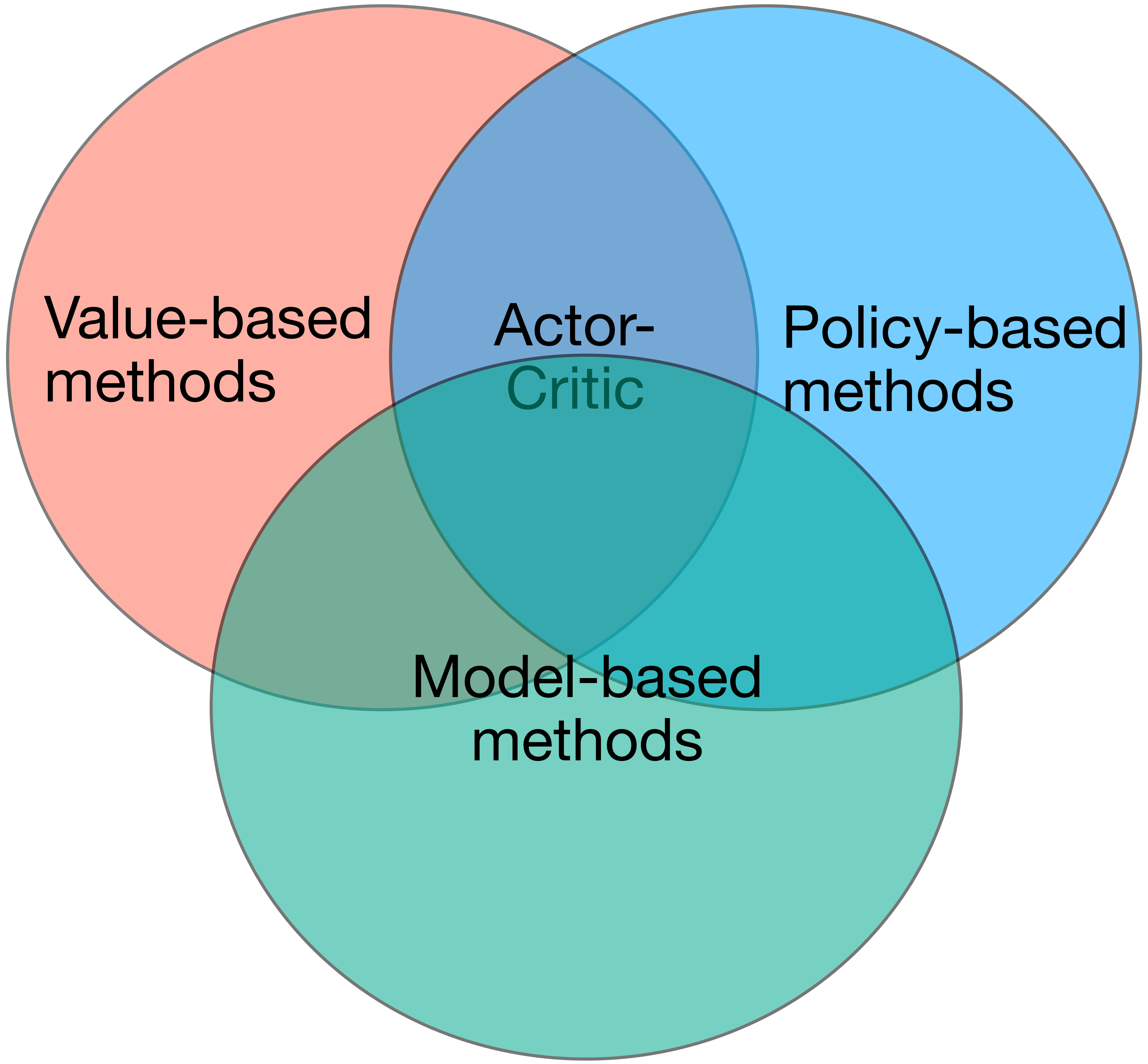
Summary

- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor**-**Critic**



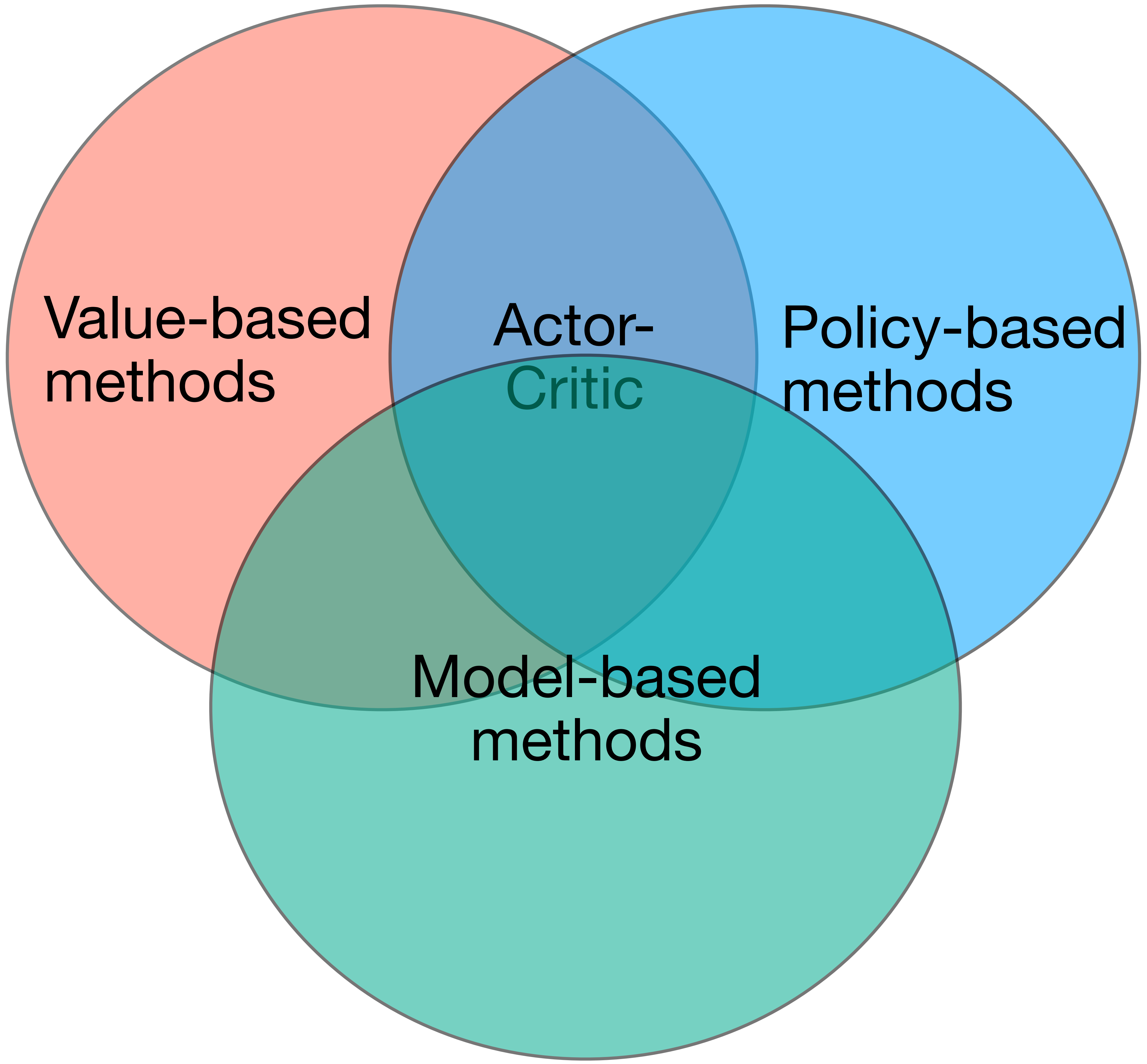
Summary

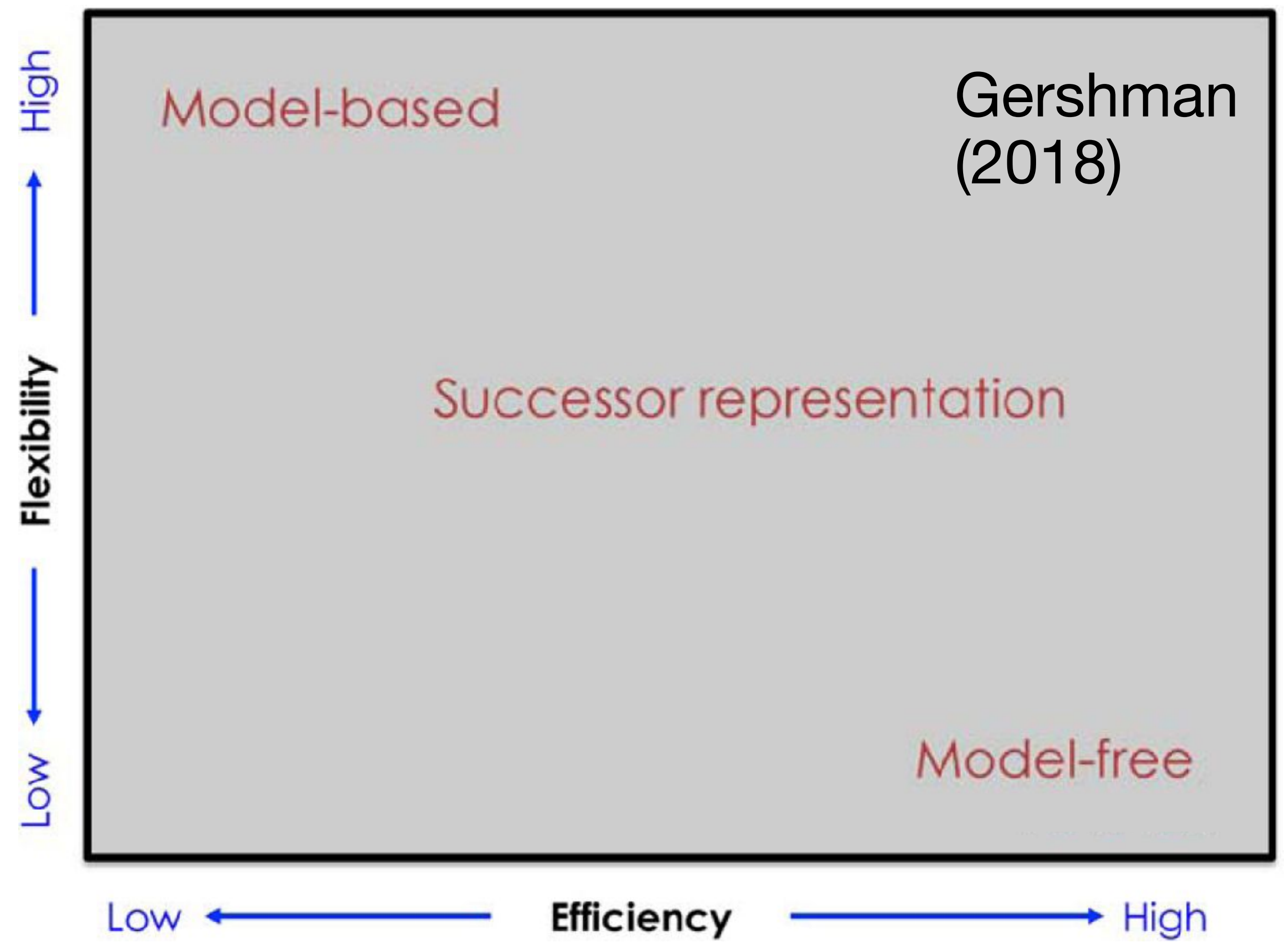
- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor**-**Critic**
- Model-based methods
  - DYNA (**Model** & **Value**)
  - World Models (**Model** & **Policy**)
  - Dreamer (**Model** & **Actor**-**Critic**)



Summary

- Model-free methods
  - **Value-based** Deep Q-Learning
  - **Policy-based** Policy Gradient
  - **Actor-Critic**
- Model-based methods
  - DYNA (**Model** & **Value**)
  - World Models (**Model** & **Policy**)
  - Dreamer (**Model** & **Actor-Critic**)
- Successor representation
  - Balancing flexibility and efficiency



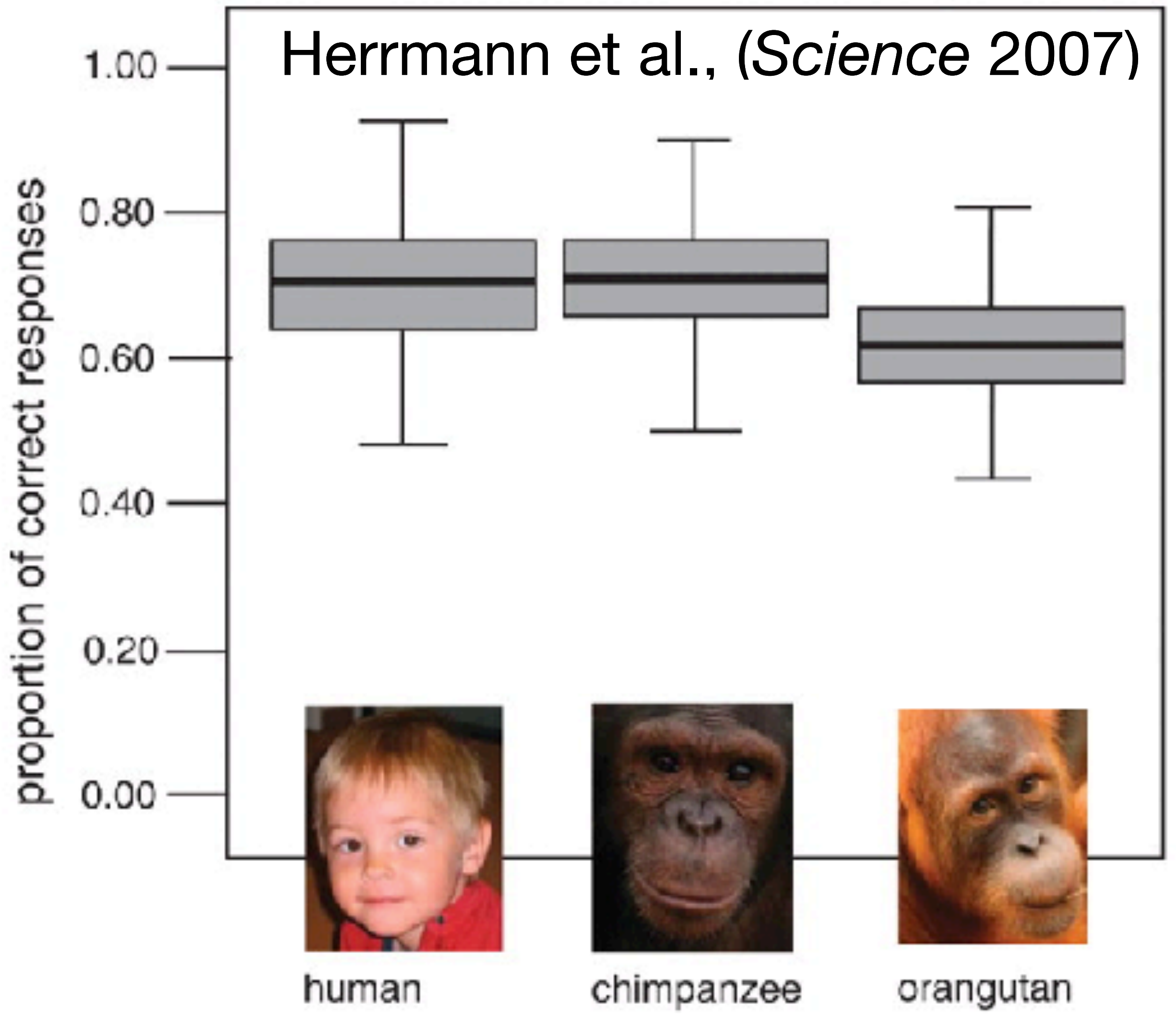


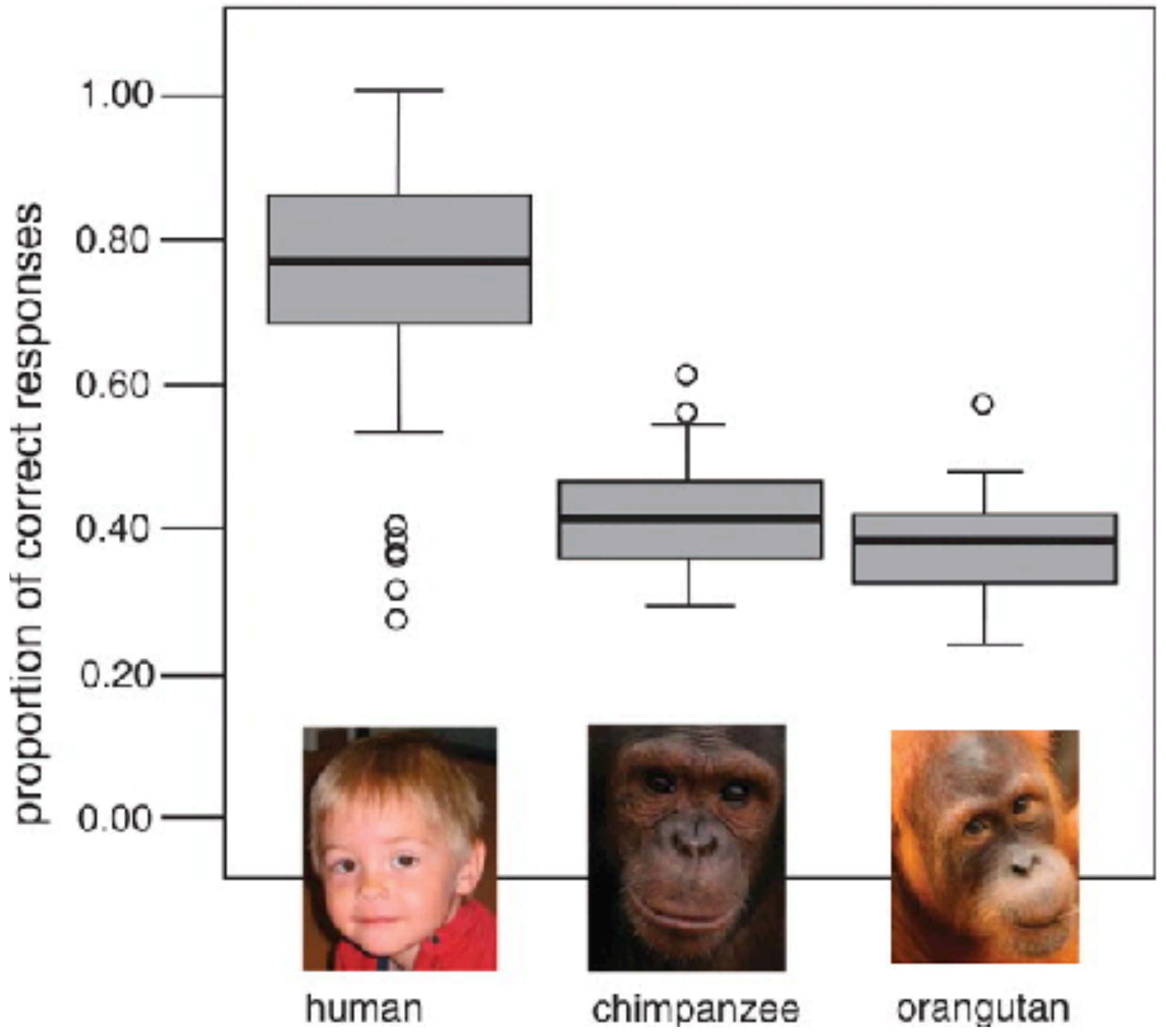
Next week: Social learning

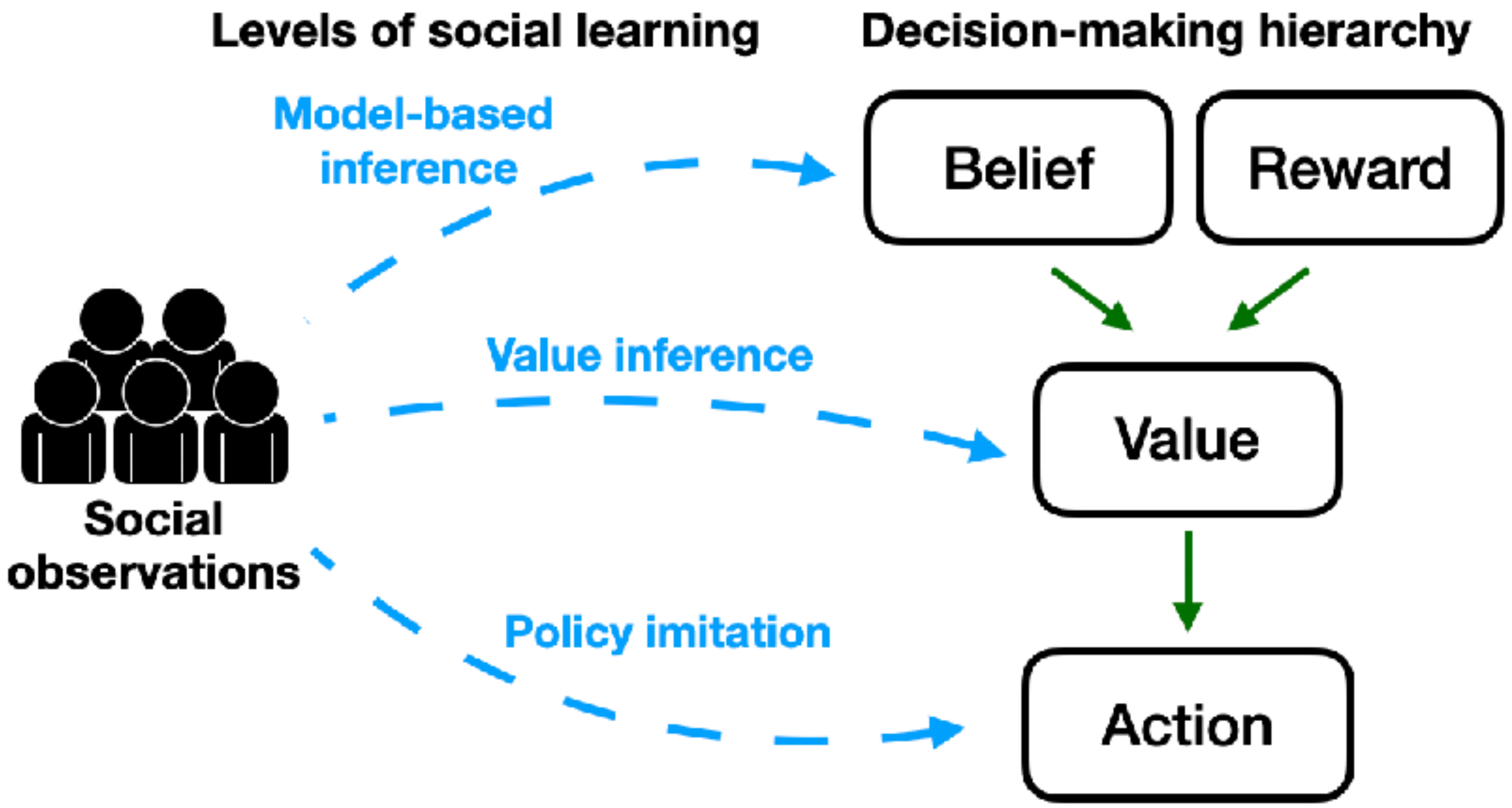
Alexandra Witt

Physical

Social







Wu et al., (2022)

Witt et al., (*PNAS* 2024)