Reinforcement Learning

Policy Optimization and Planning
(Material not examinable)

Subramanian Ramamoorthy
School of Informatics

31 March, 2017

Plan for Lecture: Policies and Plans

- Policy Optimization
  - Policies can be optimized directly, without learning value functions
  - *Policy-gradient methods*
  - Special case: how could we learn with real-valued (continuous) actions
- Planning
  - Uses of “environment models”
  - Integration of planning, learning, and execution
  - “Model-based reinforcement learning”

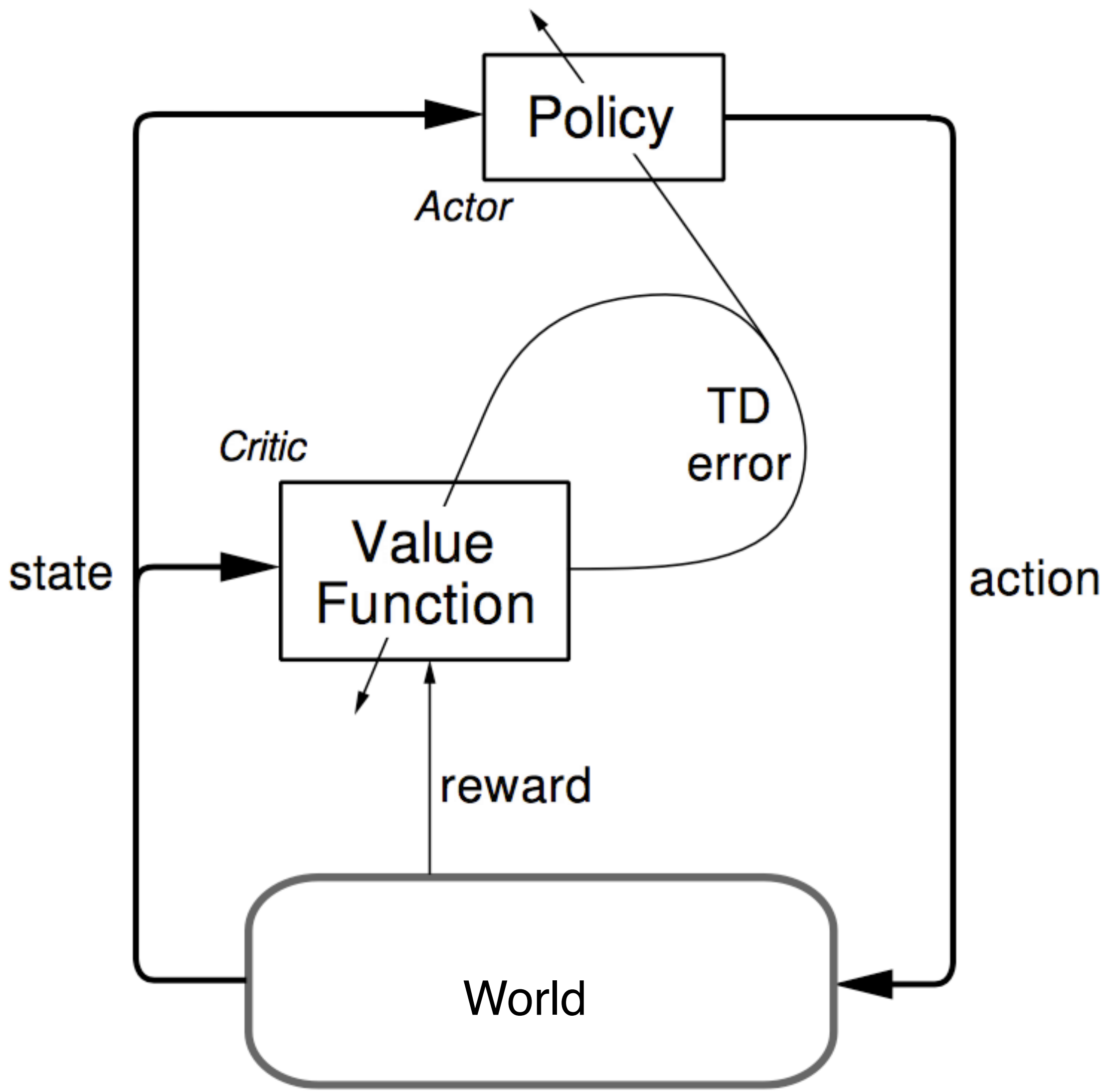
Policy-gradient methods

(*Note*: slightly different notation in this section, following 2nd ed. of S+B)

Approaches to control

1. Previous approach: *Action-value methods*:
   - learn the value of each (state-)action;
   - pick the max, usually
2. New approach: *Policy-gradient methods*:
   - learn the parameters of a stochastic policy
   - update by gradient ascent in performance
   - includes *actor-critic methods*, which learn *both* value and policy parameters

Actor-critic architecture



Why Approximate Policies rather than Values?

- In many problems, the policy is simpler to approximate than the value function
- In many problems, the optimal policy is stochastic
  - e.g., bluffing, POMDPs
- To enable smoother change in policies
- To avoid a search on every step (the max)
- To better relate to biology

Policy Approximation

- Policy = a function from state to action
  - How does the agent select actions?
  - In such a way that it can be affected by learning?
  - In such a way as to assure exploration?
- Approximation: there are too many states and/or actions to represent all policies
  - To handle large/continuous action spaces

Gradient Bandit Algorithm

- Store action preferences $H_t(a)$
  rather than action-value estimates $Q_t(a)$

- Instead of ε-greedy, pick actions by an exponential soft-max:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- Also store the sample average of rewards as $\bar{R}_t$

- Then update:

$$\frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t)$$

$$H_{t+1}(a) = H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbb{1}_{a=A_t} - \pi_t(a)\big)$$

1 or 0, depending on whether
the predicate (subscript) is true

Core Principle: Policy Gradient Methods

- Parameterized policy selects actions without consulting a value function
- VF can still be used to **learn** the policy weights
  - But not needed for action selection
- Gradient ascent on a performance measure $\eta(\theta)$ with respect to policy weights

$$\theta_{t+1} = \theta_t + \underline{\alpha \widehat{\nabla \eta(\theta_t)}}$$

Expectation approximates
the gradient (hence "policy gradient")

Linear-exponential policies (discrete actions)

- The "preference" for action a in state s is linear in θ and a state-action feature vector $\phi(s,a)$

- The probability of action a in state s is exponential in its preference

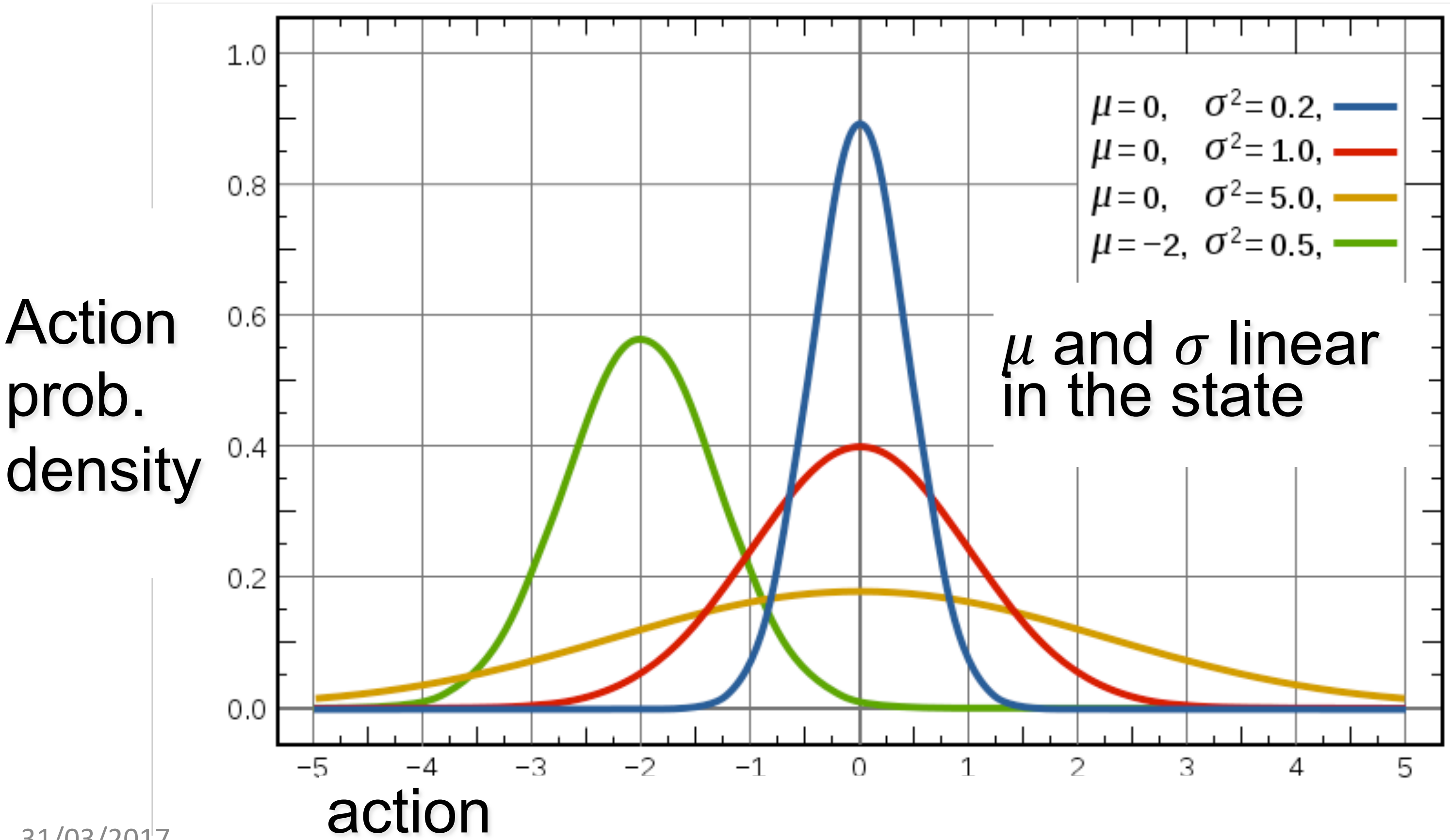
$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a))}{\sum_b \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,b))}$$

- Corresponding *eligibility function:*

$$\frac{\nabla \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \boldsymbol{\phi}(s,a) - \sum_b \pi(b|s,\boldsymbol{\theta})\boldsymbol{\phi}(s,b)$$

Factor to modulate TD update, going beyond TD(0) to TD(λ)

eg, linear-gaussian policies (continuous actions)



eg, linear-gaussian policies (continuous actions)

- The mean and std. dev. for the action taken in state s are linear and linear-exponential in

$$\boldsymbol{\theta} \doteq (\boldsymbol{\theta}_\mu^\top; \boldsymbol{\theta}_\sigma^\top)^\top \qquad \mu(s) \doteq \boldsymbol{\theta}_\mu^\top \boldsymbol{\phi}(s) \qquad \sigma(s) \doteq \exp(\boldsymbol{\theta}_\sigma^\top \boldsymbol{\phi}(s))$$

- The probability density function for the action taken in state s is gaussian

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{1}{\sigma(s)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s))^2}{2\sigma(s)^2}\right)$$

Gaussian eligibility functions

$$\frac{\nabla_{\boldsymbol{\theta}_\mu} \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \frac{1}{\sigma(s)^2}(a - \mu(s))\boldsymbol{\phi}_\mu(s)$$

$$\frac{\nabla_{\boldsymbol{\theta}_\sigma} \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \left(\frac{(a - \mu(s))^2}{\sigma(s)^2} - 1\right)\boldsymbol{\phi}_\sigma(s)$$

Policy Gradient Setup

Given a policy parameterization:

$$\pi(a|s,\boldsymbol{\theta}) \qquad \frac{\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta})$$

And objective:

$$\eta(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(S_0)$$ (or average reward)

Approximate stochastic gradient ascent:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \widehat{\nabla \eta(\boldsymbol{\theta}_t)}$$

Typically, based on the Policy-Gradient Theorem:

$$\nabla \eta(\boldsymbol{\theta}) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

REINFORCE: Monte-Carlo Policy Gradient, from Policy Gradient Theorem

$$
\begin{aligned}
\nabla\eta(\boldsymbol{\theta}) &= \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}), \\
&= \mathbb{E}_\pi\left[\gamma^t \sum_a q_\pi(S_t,a) \nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta})\right] \\
&= \mathbb{E}_\pi\left[\gamma^t \sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_\pi\left[\gamma^t q_\pi(S_t,A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right] \qquad \text{(replacing } a \text{ by the sample } A_t \sim \pi) \\
&= \mathbb{E}_\pi\left[\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right] \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t))
\end{aligned}
$$

Thus

$$
\boldsymbol{\theta}_{t+1} \triangleq \boldsymbol{\theta}_t + \alpha \widehat{\nabla\eta(\boldsymbol{\theta}_t)} \triangleq \boldsymbol{\theta}_t + \alpha\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}
$$

The generality of the policy-gradient strategy

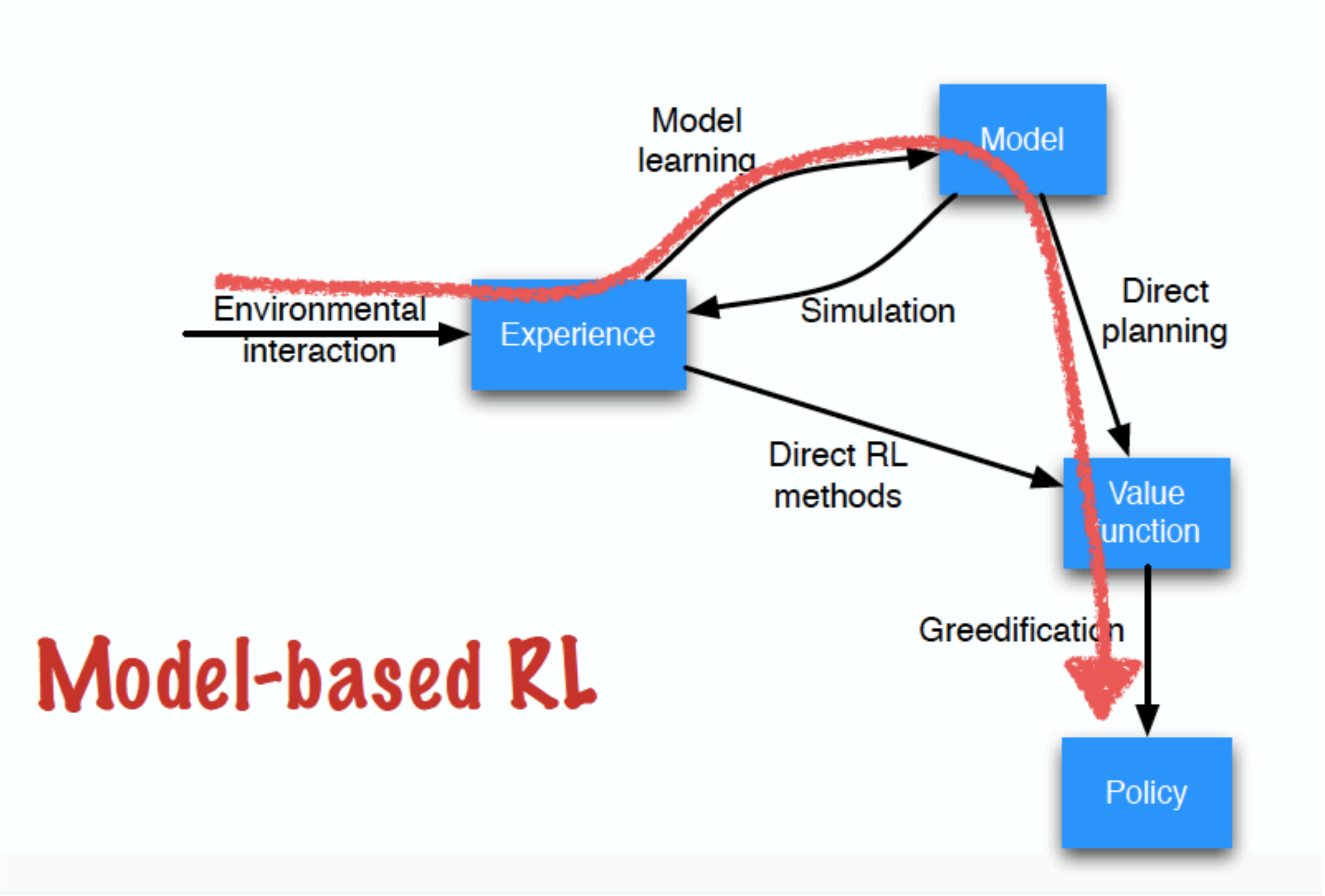
- Can be applied whenever we can compute the effect of parameter changes on the action probabilities, $\nabla\pi(A_t|S_t, \boldsymbol{\theta})$

  e.g., has been applied to spiking neuron models

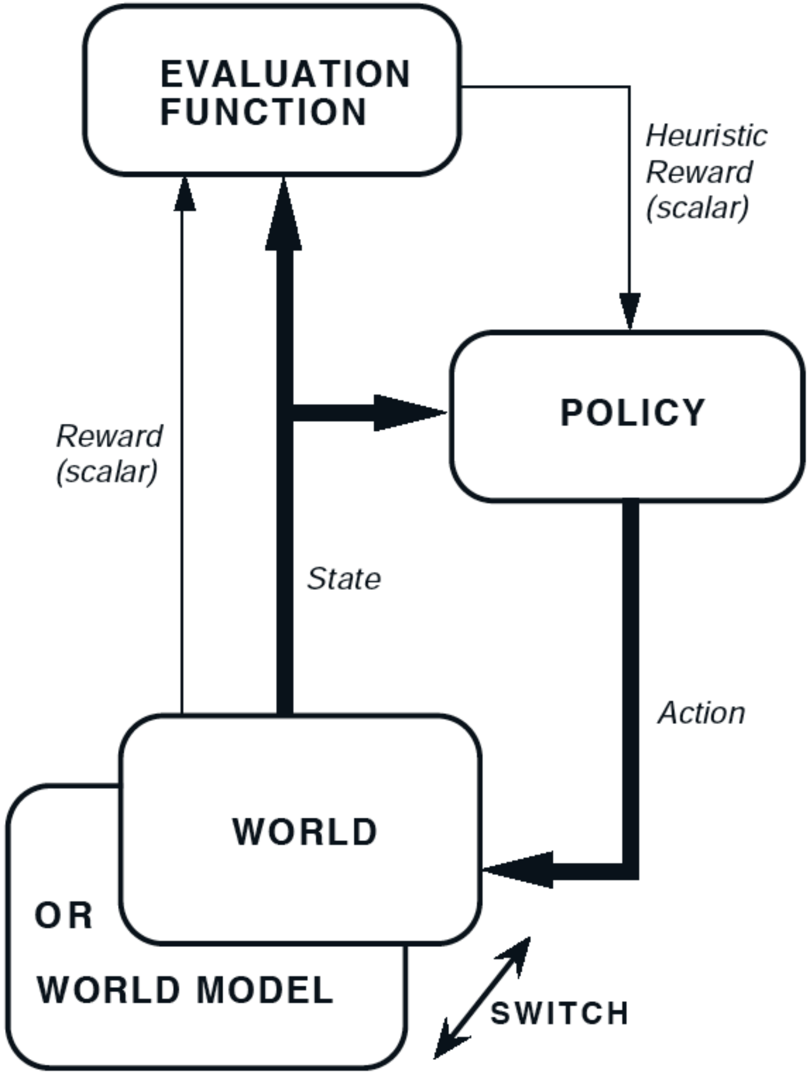
- There are many possibilities other than linear-exponential and linear-gaussian
  - e.g., mixture of random, argmax, and fixed-width gaussian; learn the mixing weights, drift/diffusion models

Planning

Paths to a Policy



Schematic



EVALUATION FUNCTION
Heuristic Reward (scalar)
POLICY
Reward (scalar)
State
Action
WORLD
OR
WORLD MODEL
SWITCH

Models

- Model: anything the agent can use to predict how the environment will respond to its actions
- Distribution model: description of all possibilities and their probabilities
  - e.g., $P^a_{ss'}$ and $R^a_{ss'}$ for all $s,\ s',$ and $a \in A(s)$
- Sample model: produces sample experiences
  - e.g., a simulation model
- Both types of models can be used to produce simulated experience
- Often sample models are much easier to come by

Planning

- Planning: any computational process that uses a model to create or improve a policy

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

- Planning in AI:
  - state-space planning
  - plan-space planning (e.g., partial-order planner)
- We take the following (unusual) view:
  - all state-space planning methods involve computing value functions, either explicitly or implicitly
  - they all apply backups to simulated experience

$$\text{model} \longrightarrow \begin{array}{c}\text{simulated}\\ \text{experience}\end{array} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

Planning Cont.

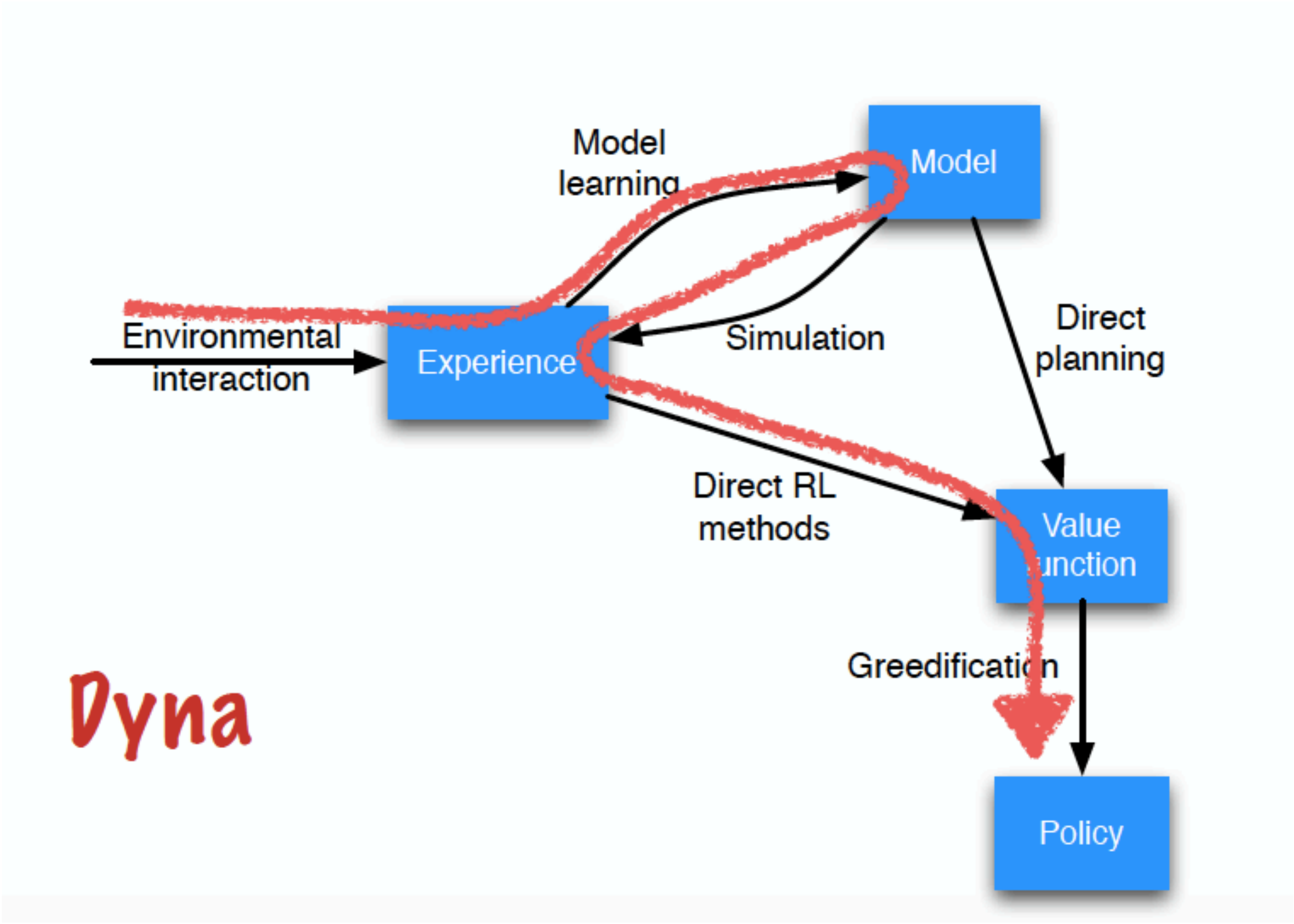
- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- A planning method based on Q-learning:

Do forever:

1. Select a state, $s \in \mathcal{S}$, and an action, $a \in \mathcal{A}(s)$, at random
2. Send s, a to a sample model, and obtain
   a sample next state, s', and a sample next reward, r
3. Apply one-step tabular Q-learning to s, a, s', r:
   $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

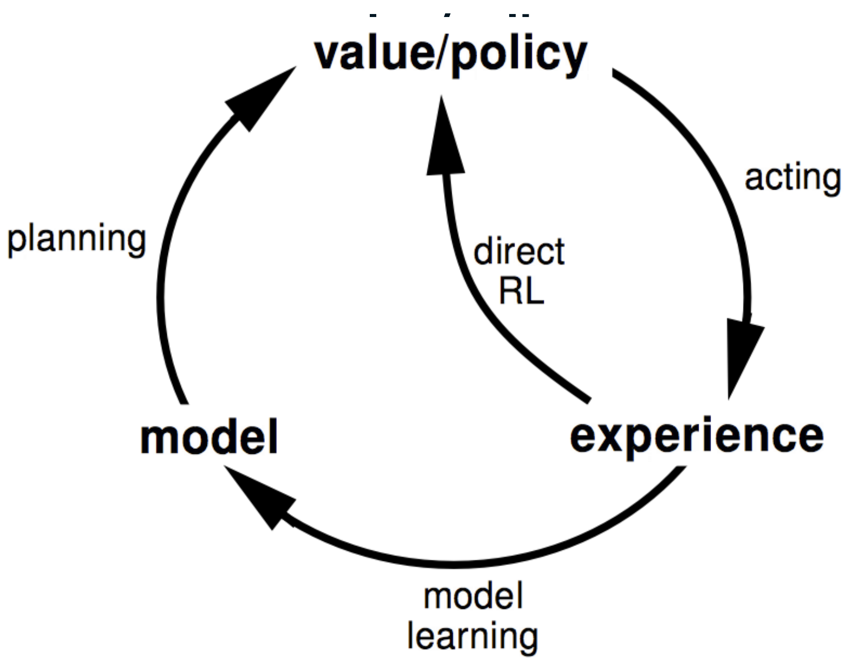
Random-Sample One-Step Tabular Q-Planning

Paths to a Policy: Dyna



Learning, Planning, and Acting

- Two uses of real experience:
  - model learning: to improve the model
  - direct RL: to directly improve the value function and policy
- Improving value function and/or policy via a model is sometimes called indirect RL or model-based RL. Here, we call it planning.



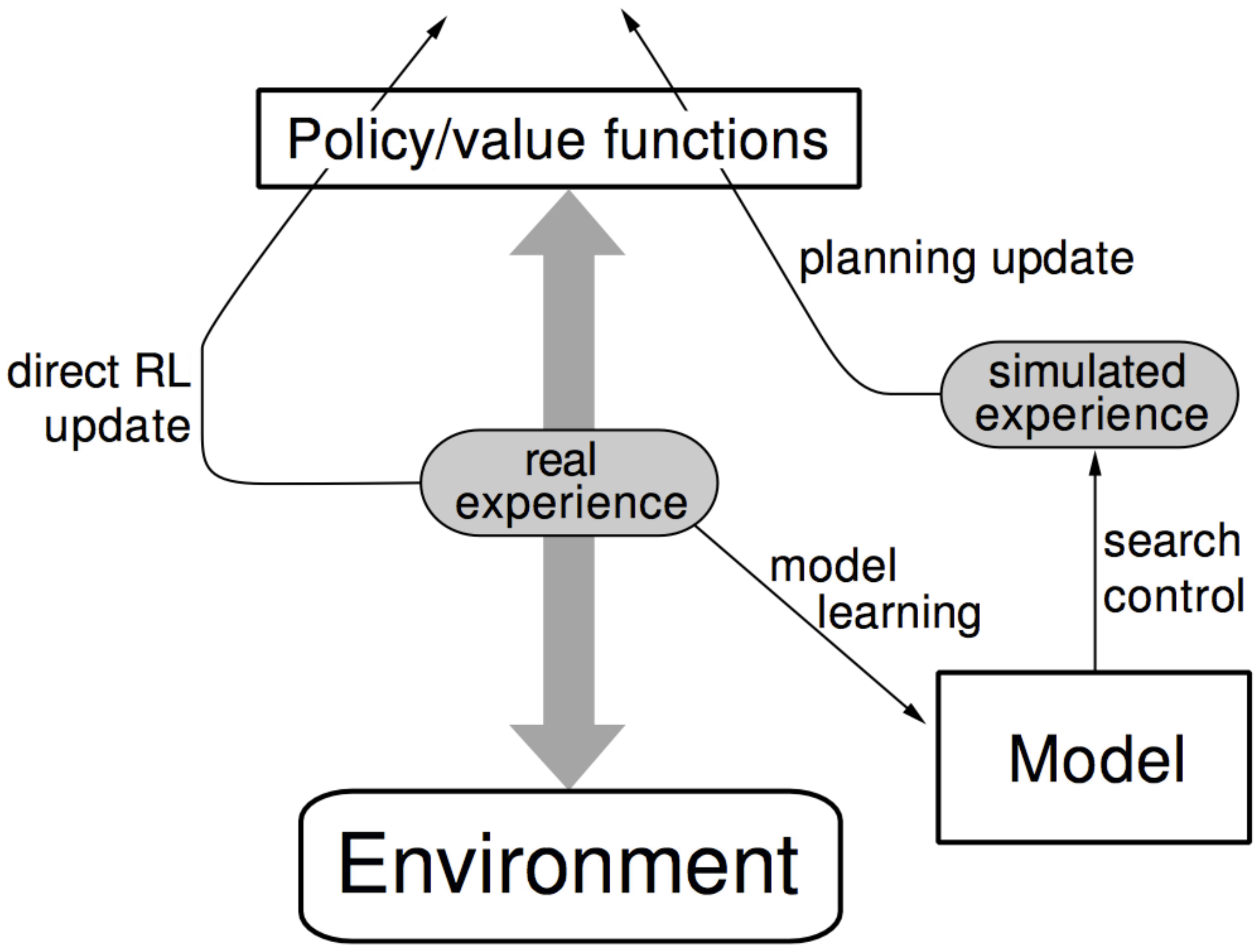
Direct vs. Indirect RL

- Indirect methods:
  - make fuller use of experience: get better policy with fewer environment interactions
- Direct methods
  - simpler
  - not affected by bad models

But they are very closely related and can be usefully combined:

planning, acting, model learning, and direct RL can occur simultaneously and in parallel

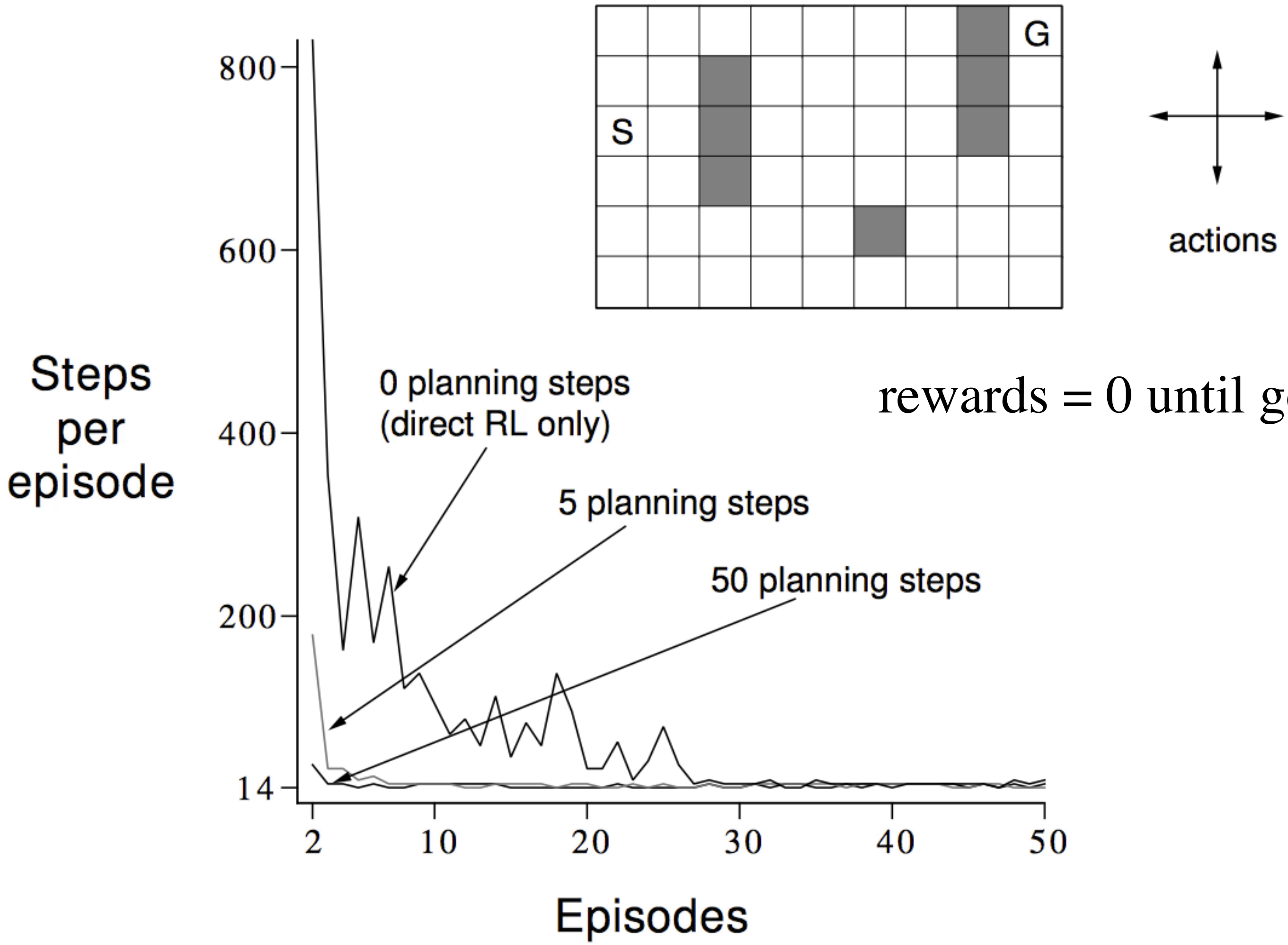
The Dyna Architecture (Sutton 1990)



The Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy(s, Q)
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ **direct RL**
(e) $Model(s, a) \leftarrow s', r$ (assuming deterministic environment) **model learning**
(f) Repeat N times: **planning**
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$s', r \leftarrow Model(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

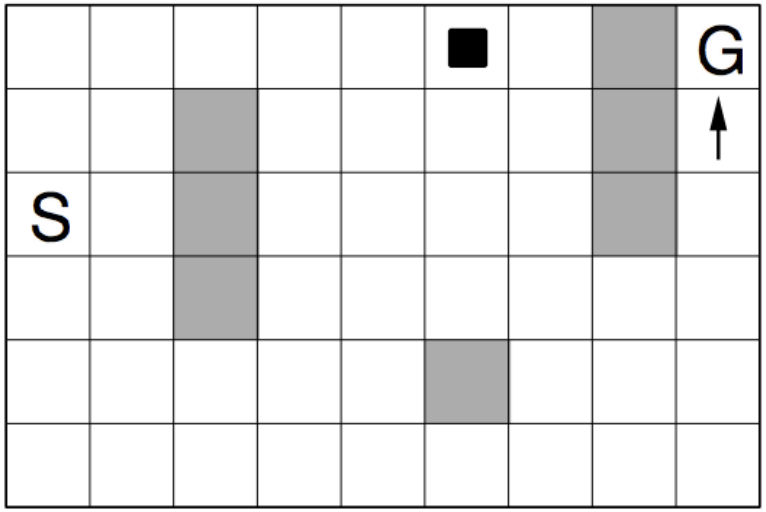
Dyna-Q on a Simple Maze



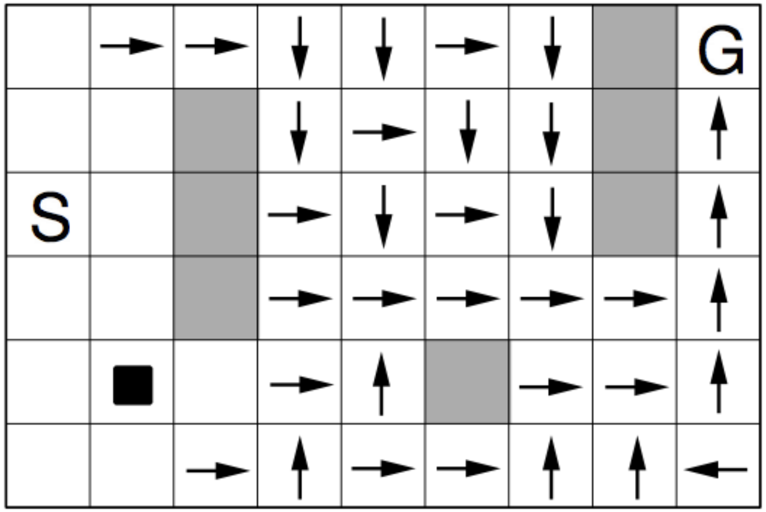
rewards = 0 until goal, when =1

Dyna-Q Snapshots: Midway in 2nd Episode

WITHOUT PLANNING (N=0)

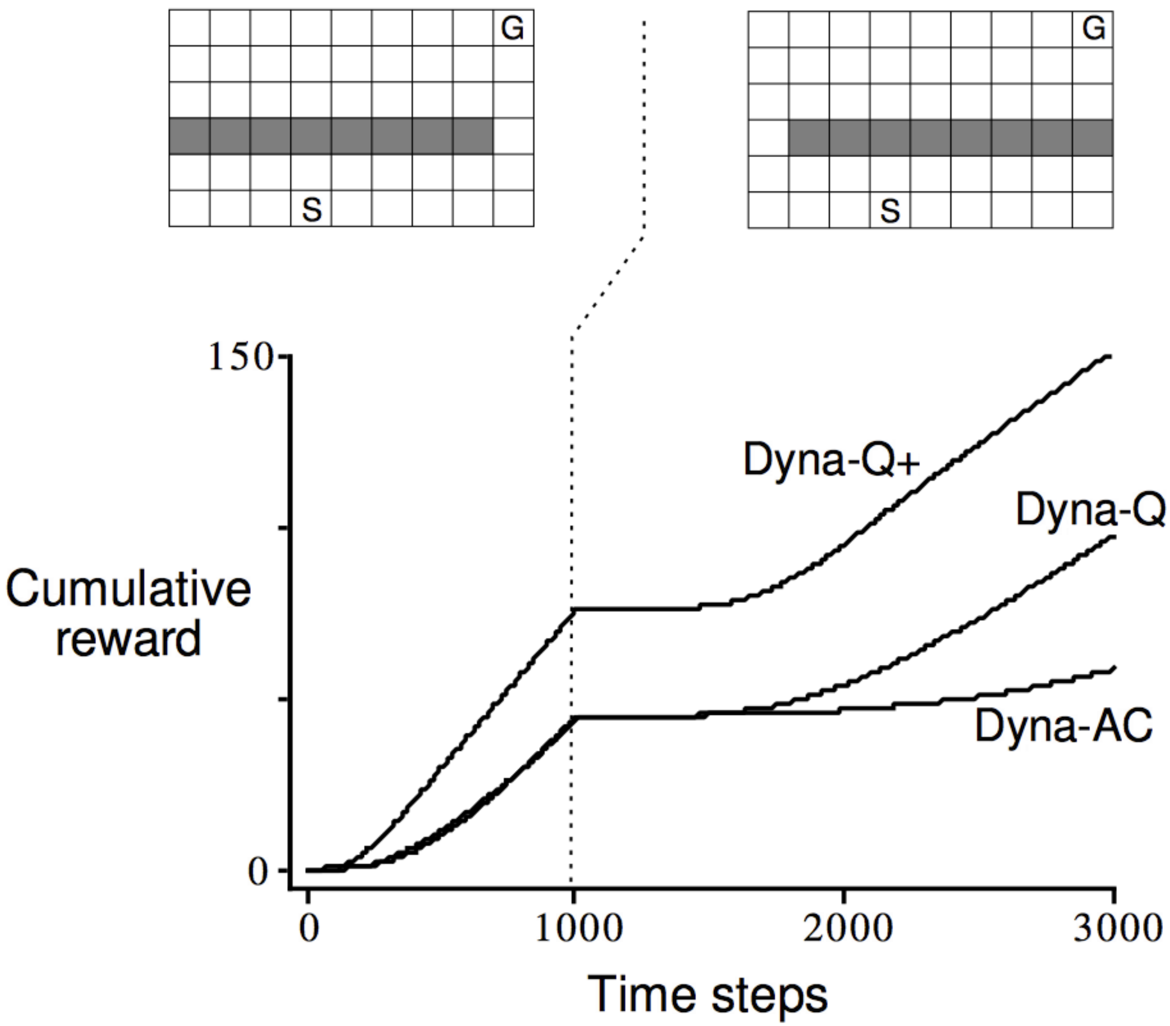


WITH PLANNING (N=50)



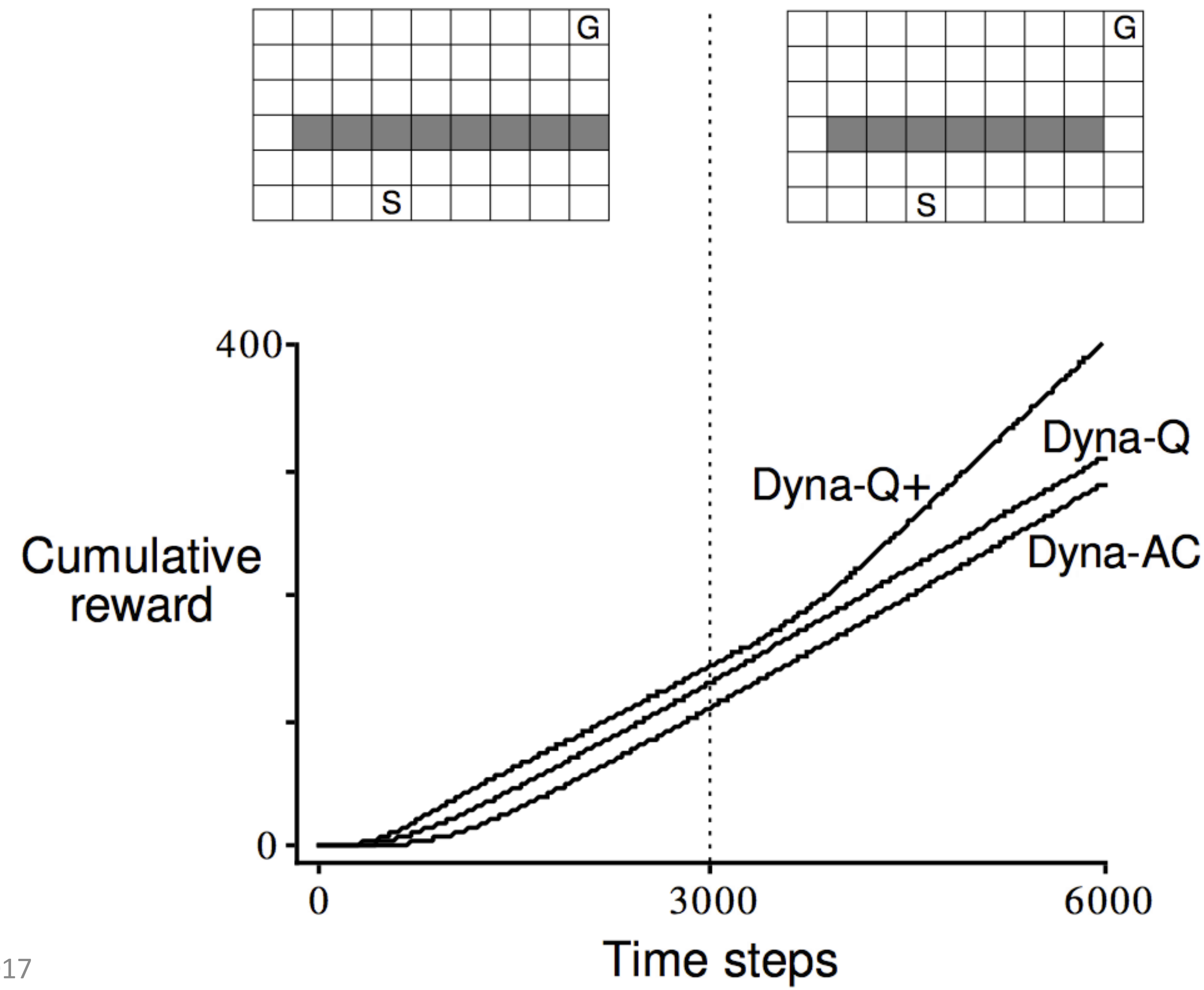
When the Model is Wrong: Blocking Maze

The changed envirnoment is harder



Shortcut Maze

The changed environment is easier



What is Dyna-Q^+ ?

- Uses an "exploration bonus":
  - Keeps track of time since each state-action pair was tried for real
  - An extra reward is added for transitions caused by state-action pairs related to how long ago they were tried: the longer unvisited, the more reward for visiting

$$r + \kappa\sqrt{n}$$

  - The agent actually "plans" how to visit long unvisited states