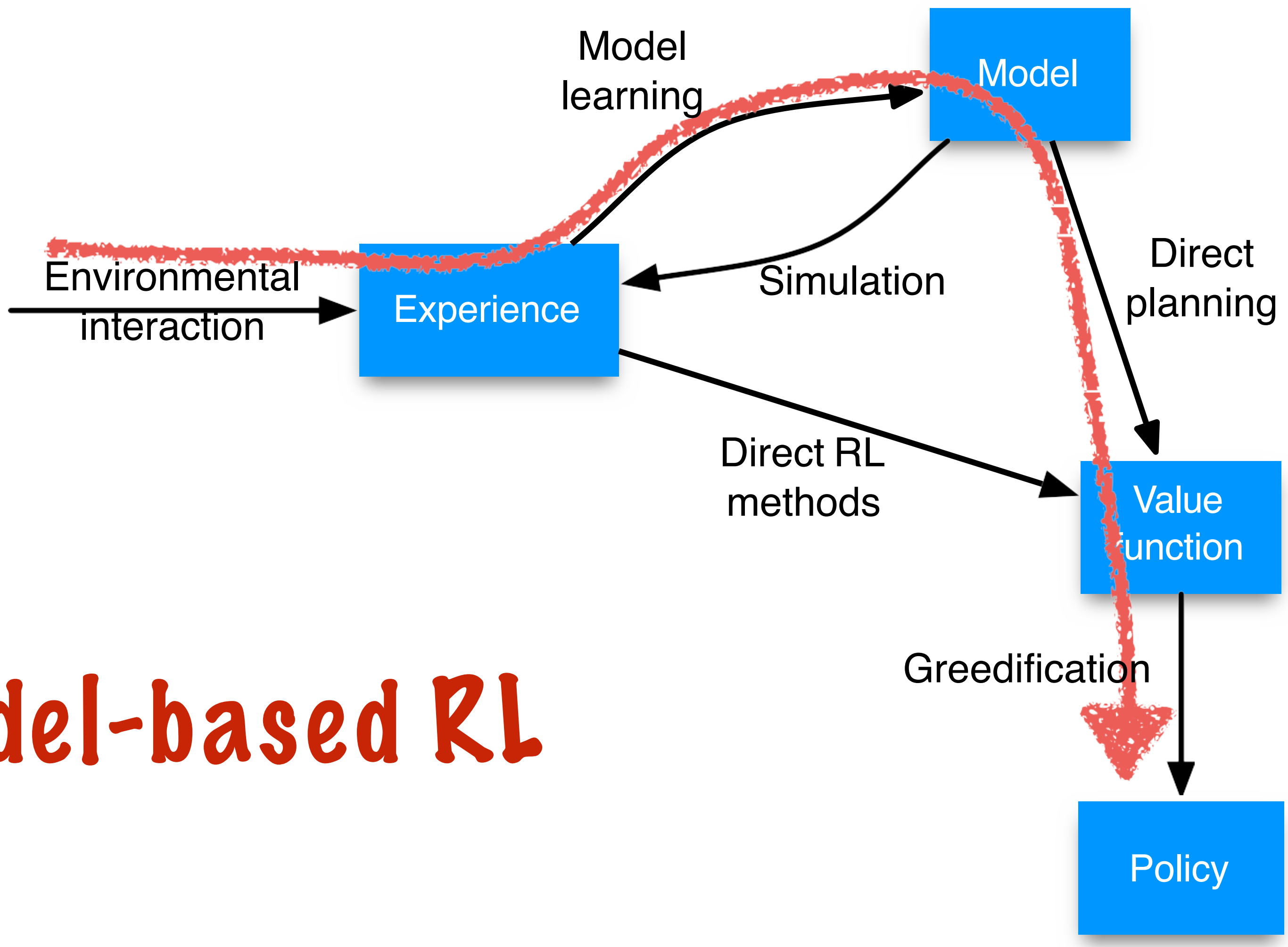
Chapter 8: Planning and Learning

Objectives of this chapter:

- To think more generally about uses of environment models
- Integration of (unifying) planning, learning, and execution
- “Model-based reinforcement learning”

Paths to a policy



Model-based RL

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., $\hat{p}(s', r | s, a)$ for all s, a, s', r
- Sample model, a.k.a. a simulation model
  - produces sample experiences for given s, a
  - allows reset, exploring starts
  - often much easier to come by
- Both types of models can be used to produce hypothetical experience

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., evolutionary method)
- 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

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

Planning Cont.

- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- Learning algorithms can be substituted for the key backup step of a planning method (e.g., a planning method based on Q-learning)

Random-Sample One-Step Tabular Q-Planning

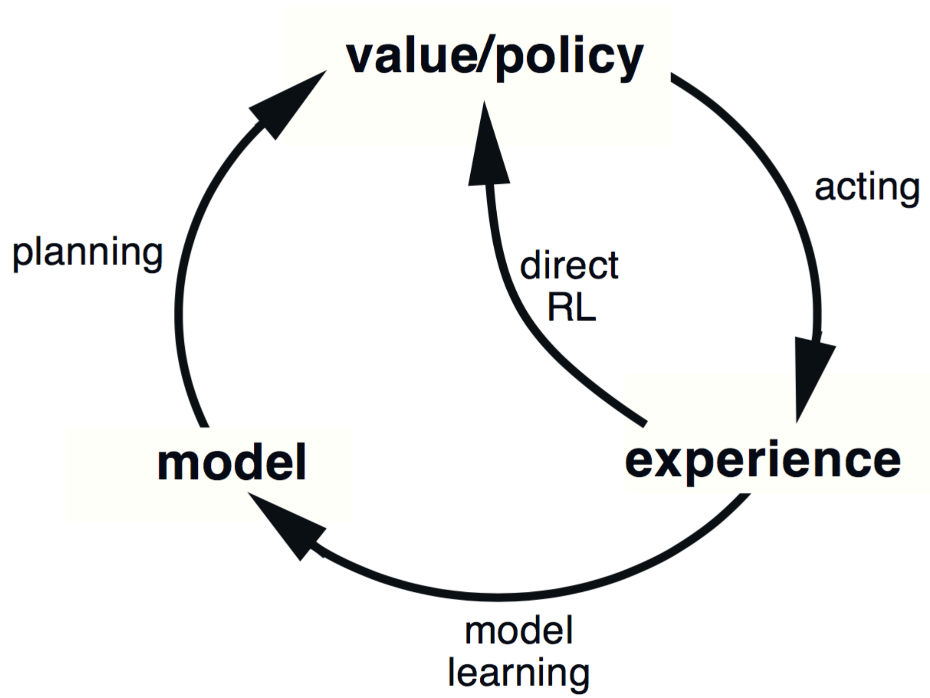
Do forever:

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 reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

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. Here, we call it planning.



Direct (model-free) vs. Indirect (model-based) RL

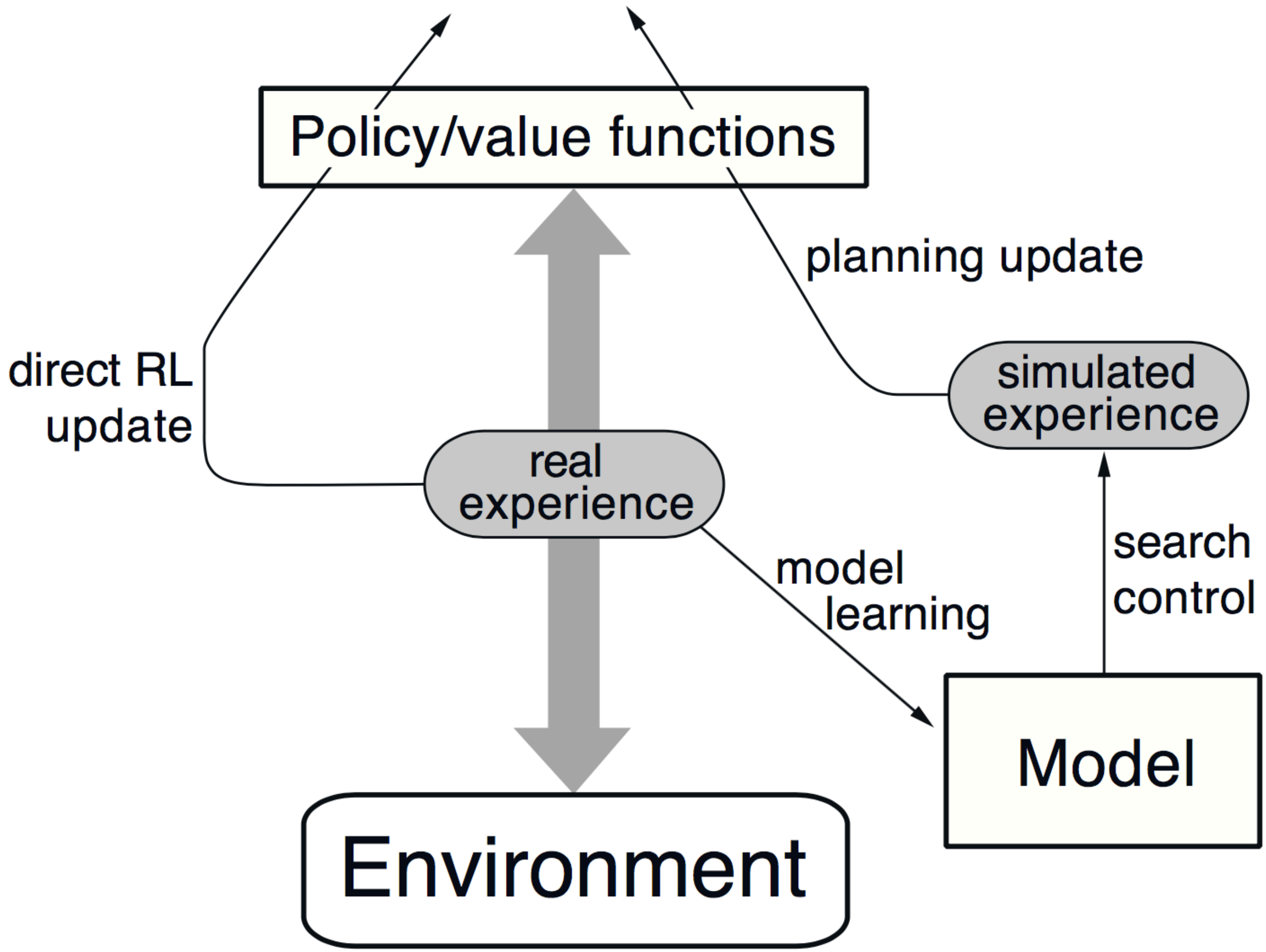
- Direct methods
  - simpler
  - not affected by bad models

- Indirect methods:
  - make fuller use of experience: get better policy with fewer environment interactions

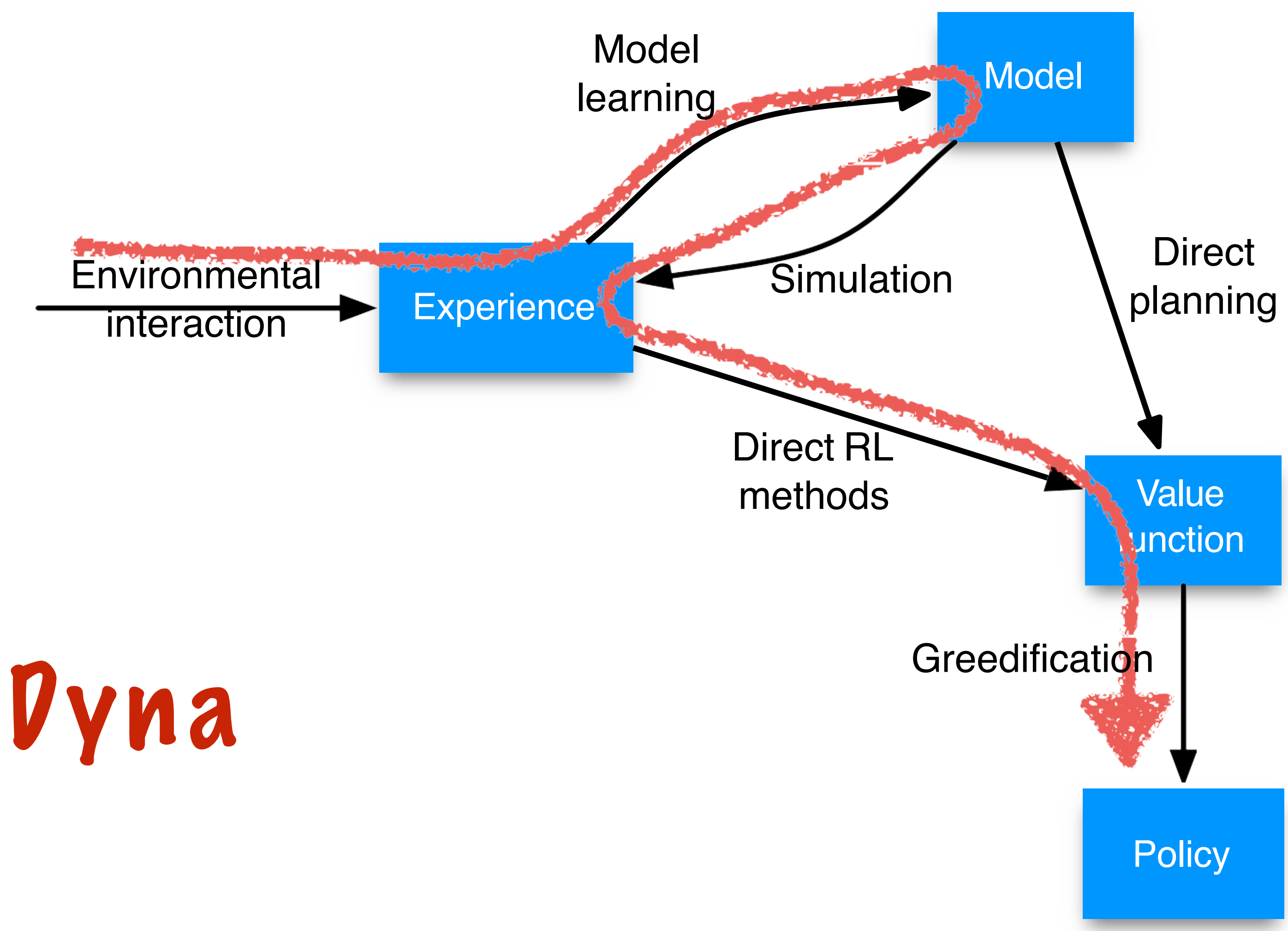
But they are very closely related and can be usefully combined:

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

The Dyna Architecture



Paths to a policy



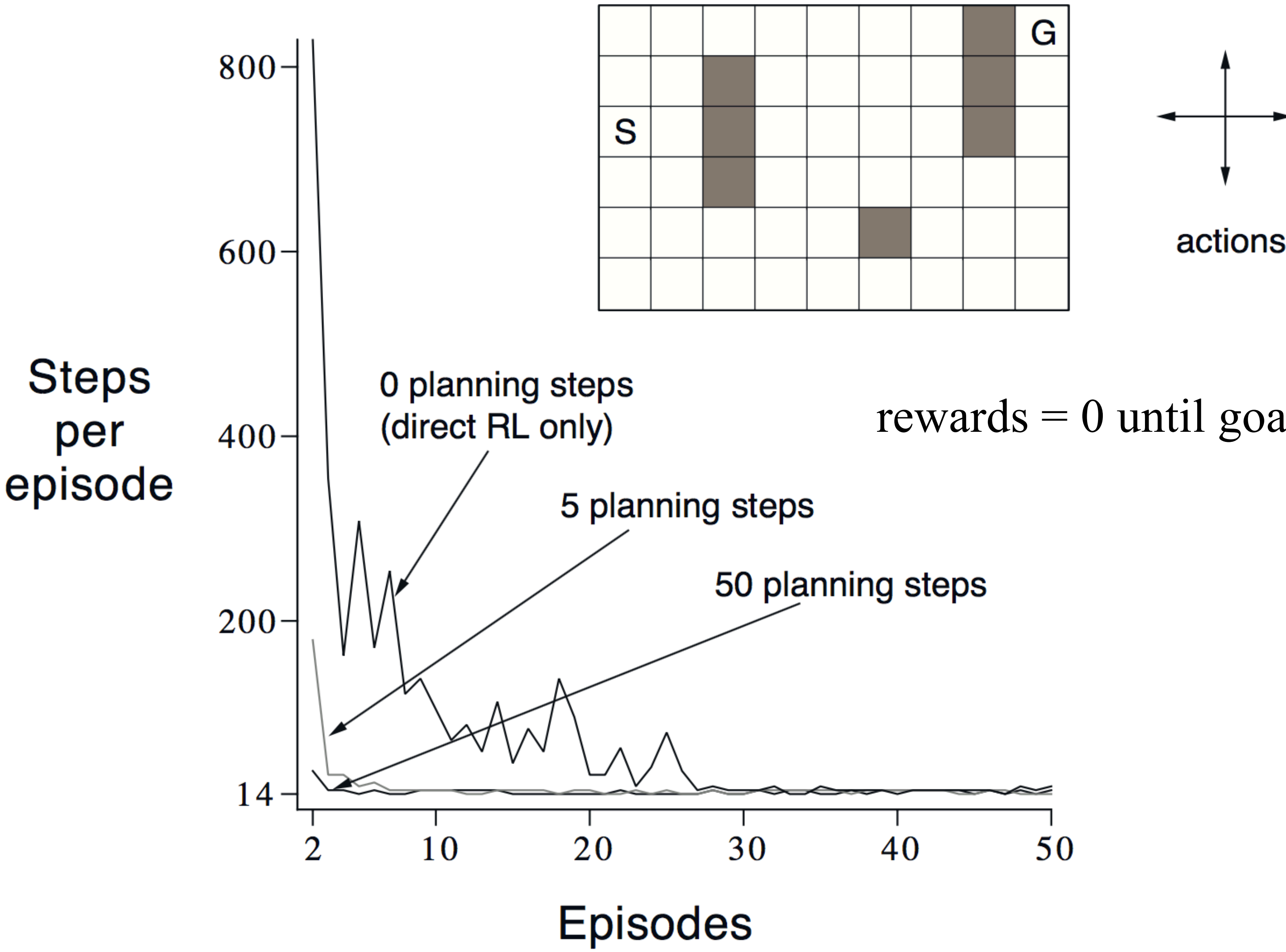
The Dyna-Q Algorithm

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) $A \leftarrow \epsilon$-greedy (S,Q)
c) Execute action A; observe resultant reward, R, and state, S'
d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$ direct RL
e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment) model learning
f) Repeat n times:
   $S \leftarrow$ random previously observed state
   $A \leftarrow$ random action previously taken in S
   $R, S' \leftarrow Model(S,A)$
   $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

planning

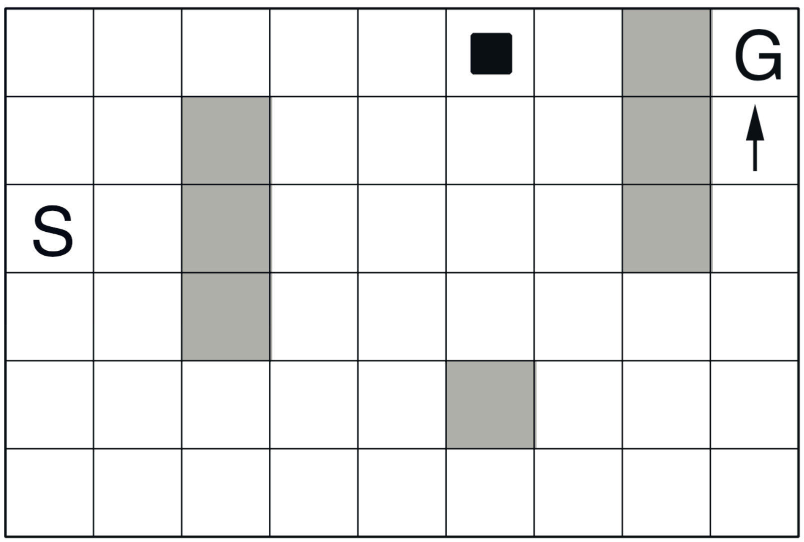
Dyna-Q on a Simple Maze



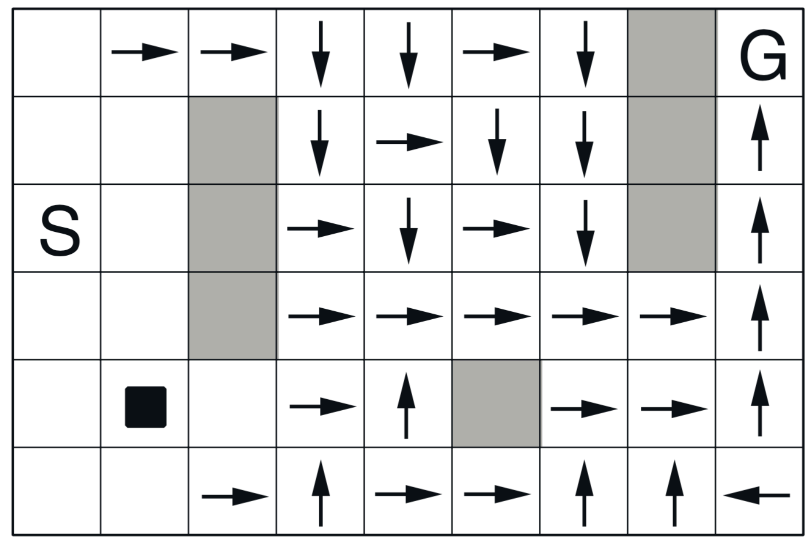
rewards = 0 until goal, when =1

Dyna-Q Snapshots: Midway in 2nd Episode

WITHOUT PLANNING (n=0)

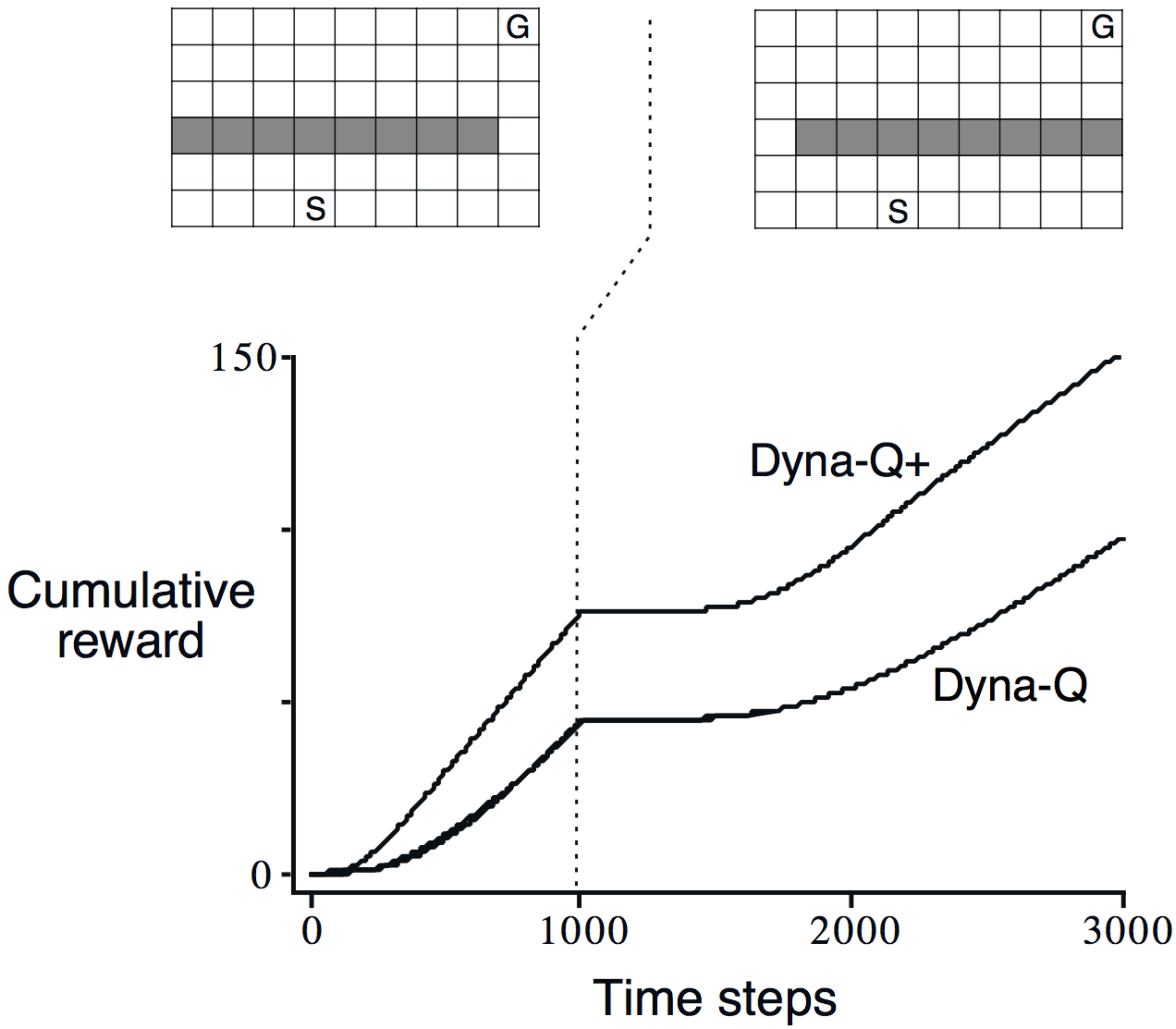


WITH PLANNING (n=50)



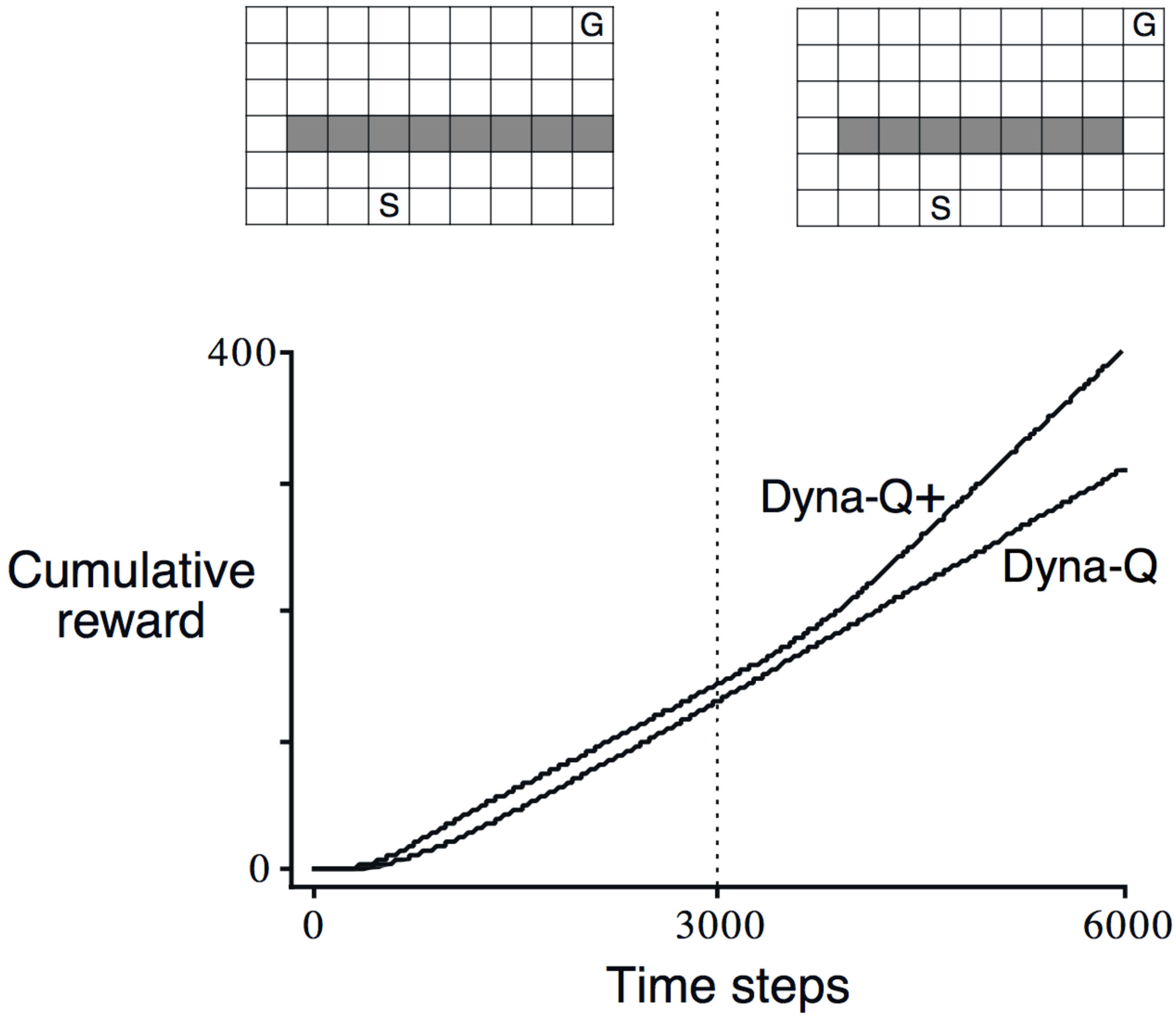
When the Model is Wrong: Blocking Maze

The changed environment is harder



When the Model is Wrong: 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

$$R + \kappa\sqrt{\tau}$$

time since last visiting the state-action pair

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

Prioritized Sweeping

- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
  - Maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
  - When a new backup occurs, insert predecessors according to their priorities
  - Always perform backups from first in queue
- Moore & Atkeson 1993; Peng & Williams 1993
- Improved by McMahan & Gordon 2005; Van Seijen 2013

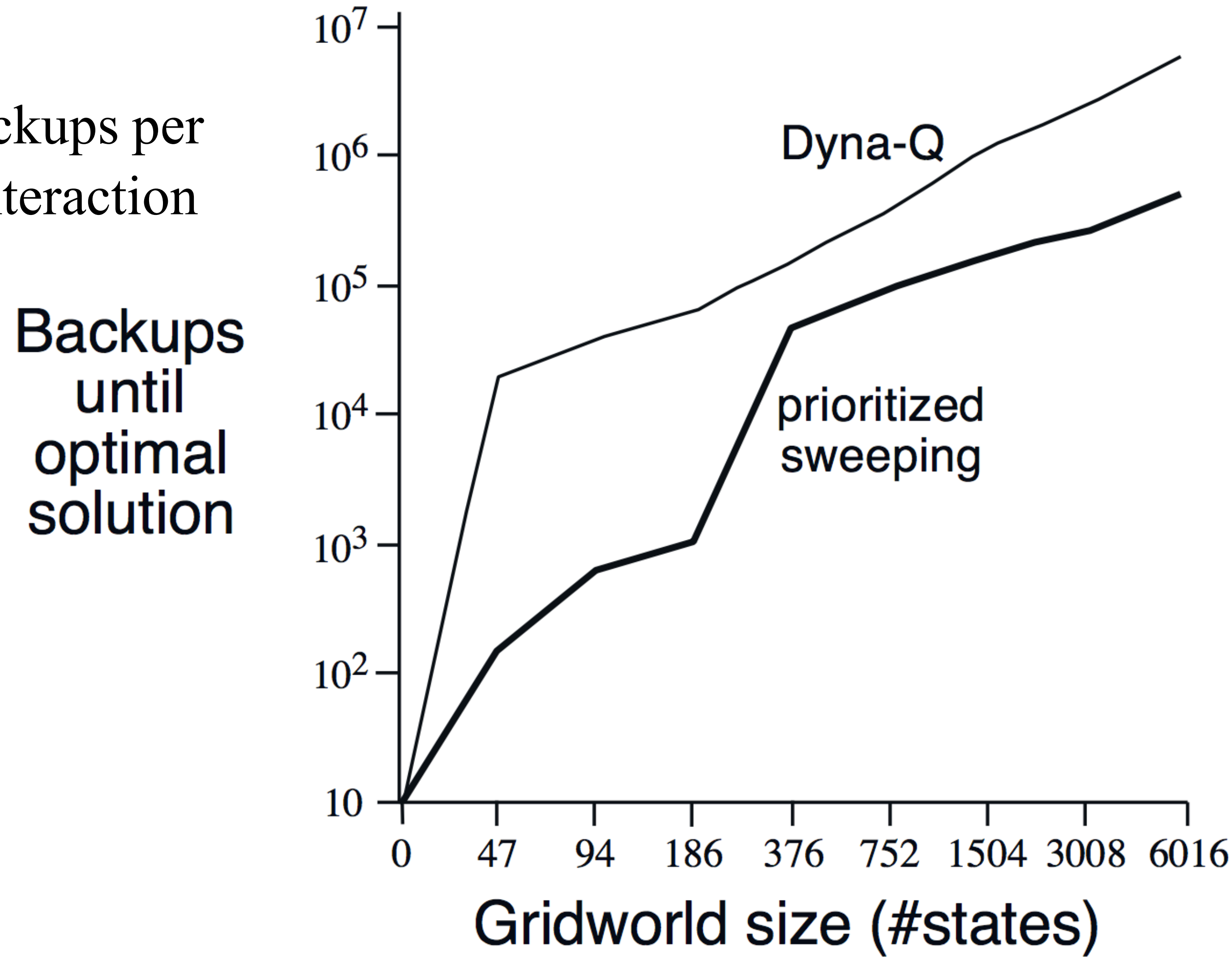
Prioritized Sweeping

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:

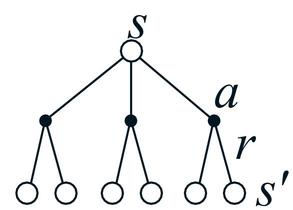
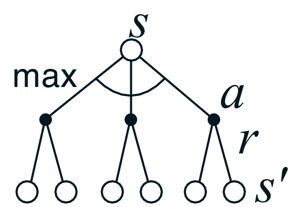
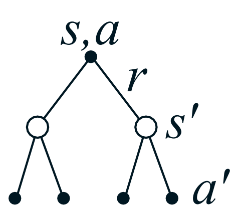
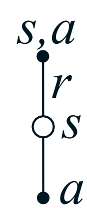
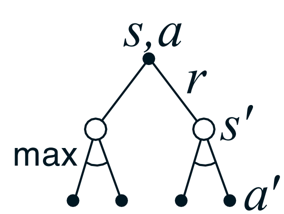
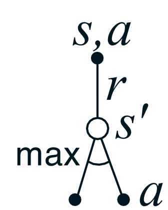
a) $S \leftarrow$ current (nonterminal) state
b) $A \leftarrow policy(S,Q)$
c) Execute action A; observe resultant reward, R, and state, S'
d) $Model(S,A) \leftarrow R, S'$
e) $P \leftarrow \left| R + \gamma \max_a Q(S',a) - Q(S,A) \right|$
f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
g) Repeat n times, while $PQueue$ is not empty:
   - $S,A \leftarrow first(PQueue)$
   - $R, S' \leftarrow Model(S,A)$
   - $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A) \right]$
   - Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
     - $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
     - $P \leftarrow \left| \bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A}) \right|$
     - if $P > \theta$, then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

Prioritized Sweeping vs. Dyna-Q

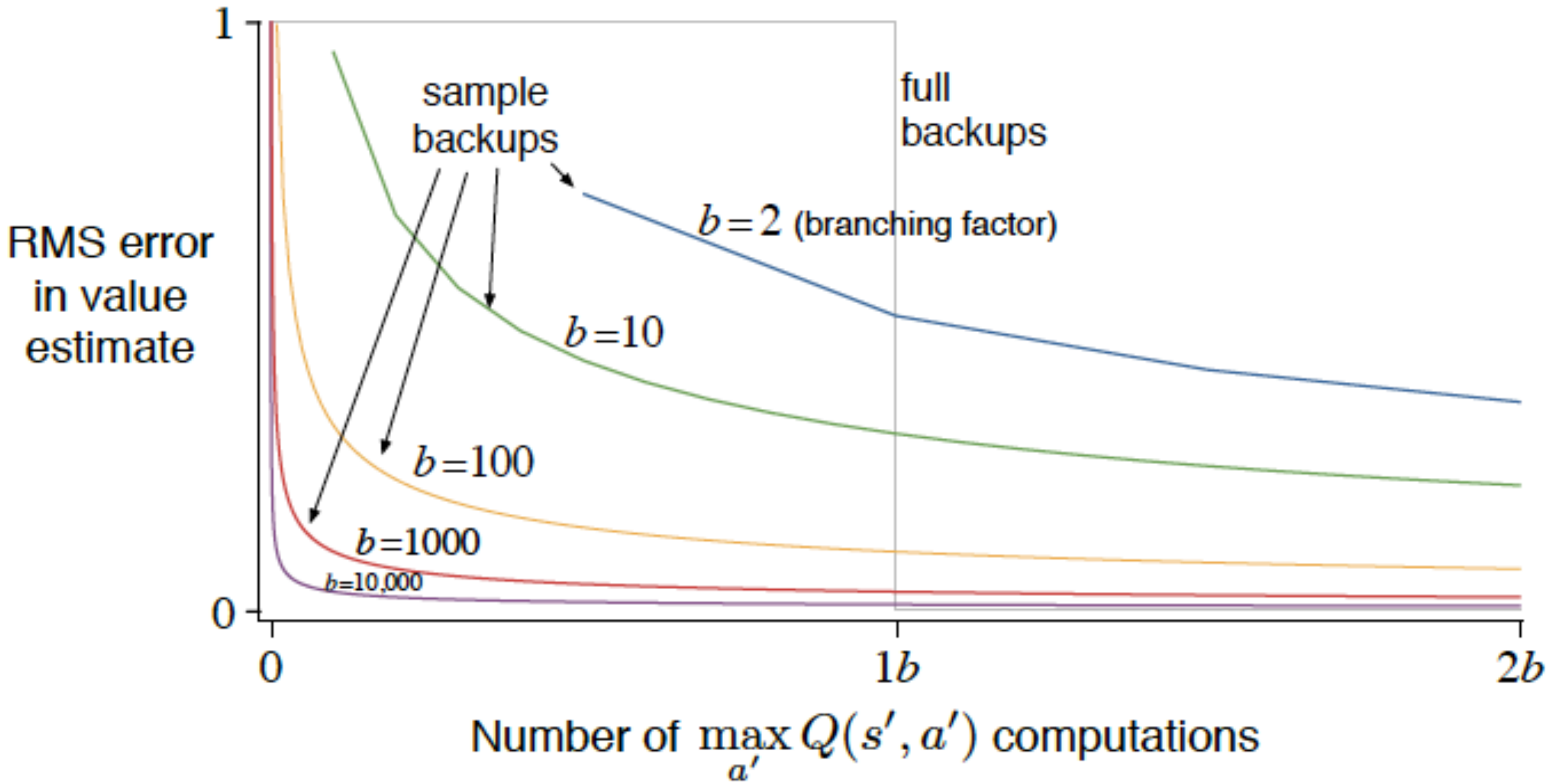
Both use n=5 backups per environmental interaction



Full and Sample (One-Step) Backups

Value estimated	Full backups (DP)	Sample backups (one-step TD)
$v_\pi(s)$	 policy evaluation	 TD(0)
$v_*(s)$	 value iteration	
$q_\pi(a, s)$	 Q-policy evaluation	 Sarsa
$q_*(a, s)$	 Q-value iteration	 Q-learning

Full vs. Sample Backups

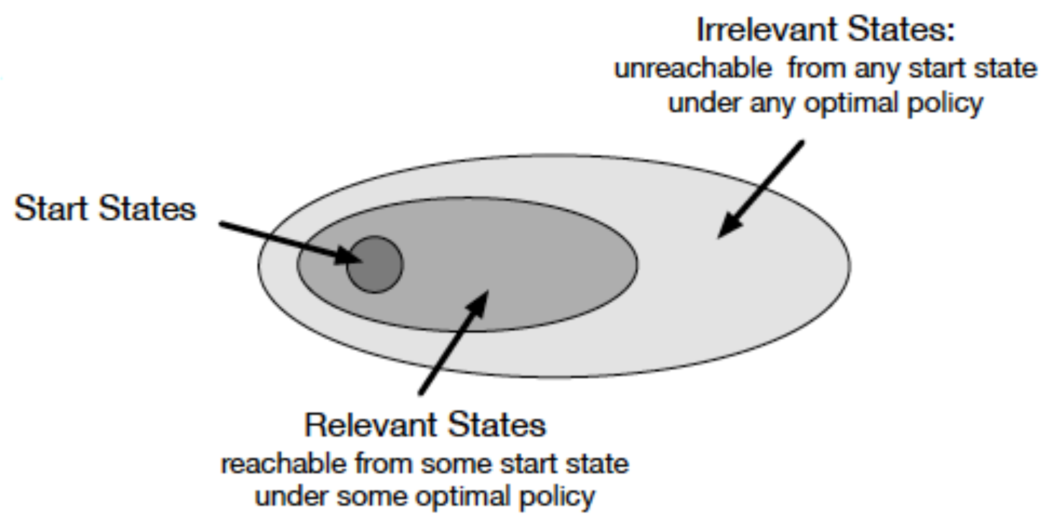


Values backed up from the "successor" states are more accurate;
Sample backups work better with more branches and states

Trajectory Sampling

- Two ways of distributing backups
  - Based on probability distribution to sweep through all state-action pairs
  - Based on some other distribution such as on-policy distribution
    - State transitions and rewards generated by the model; actions by following the current policy
    - i.e., simulate individual trajectories along the way

- Trajectory sampling: Real-time DP (RTDP)
  - Only back up the states on the sampled trajectory
  - With exploring starts, RTDP converges under (infinite-visit) conditions
  - For undiscounted episodic MDP, converges without the need for infinite-visit conditions for all the states



DP vs. RTDP

	DP	RTDP
Average computation to convergence	28 sweeps	4000 episodes
Average number of backups to convergence	252,784	127,600
Average number of backups per episode	—	31.9
% of states backed up $\leq$ 100 times	—	98.45
% of states backed up $\leq$ 10 times	—	80.51
% of states backed up 0 times	—	3.18

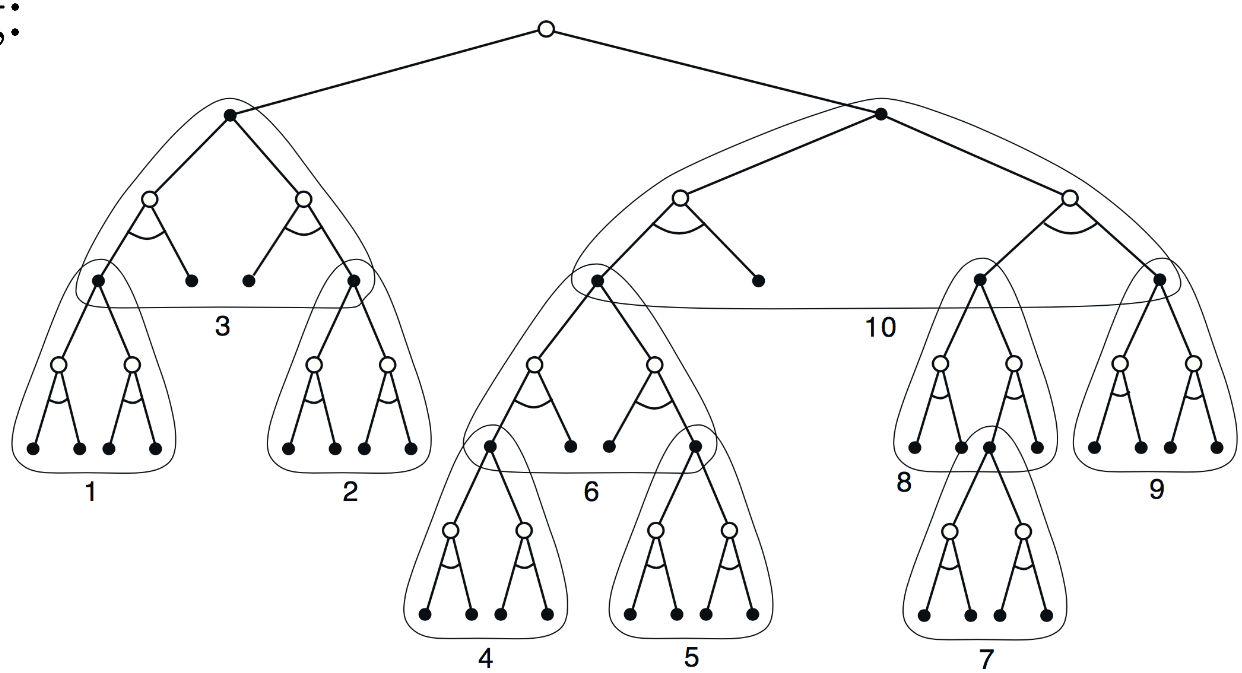
- Racetrack problem (Exercise 5.7)
- 9115 states
- 599 relevant states (by counting over optimal policy execution)

Planning at Decision Time

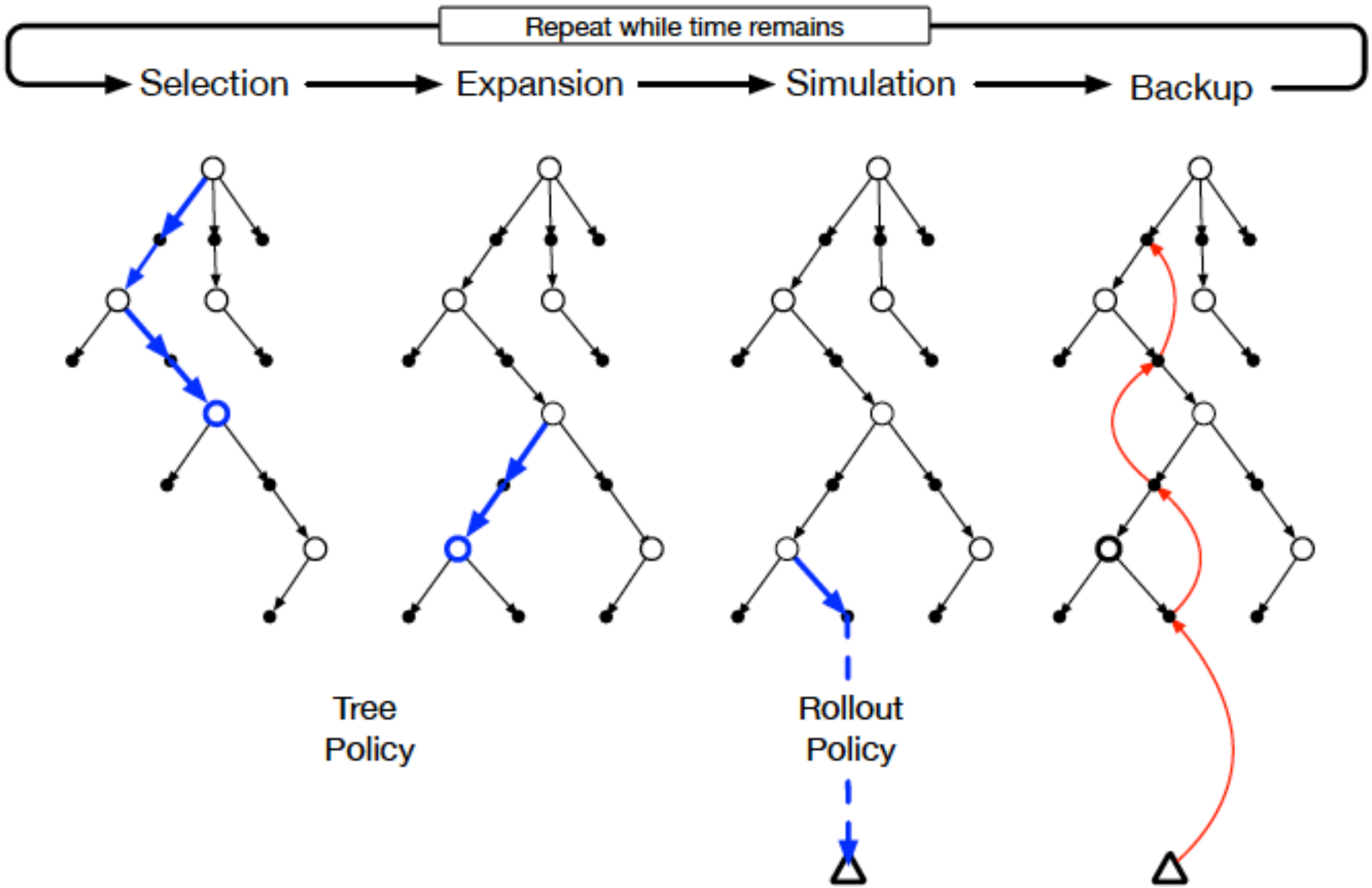
- Two ways of using planning
  - ***Background planning***: Used to improve value estimates, which improves the policy
  - ***Decision-time planning***: Used to just select an action at the current state, then discarded
    - Useful and effective when fast (relatively speaking) responses are not required: e.g., game of Go (slow response in the eyes of computers)
- Decision-time planning
  - Heuristic search: backup from a large tree of future transitions (computationally costy)
  - Rollout algorithms: backup from a tree of trajectories by following a rollout policy
  - Monte Carlo Tree Search (MCTS)
    - An efficient and powerful rollout algorithm
    - With enhancement to accumulating value estimates (a subset from the previous state decision making period)
    - Tree policy (inside the current tree) vs. rollout policy (outside the current tree)

Heuristic Search

- Used for action selection, not for changing a value function (= heuristic evaluation function)
- Backed-up values are computed, but typically discarded
- Extension of the idea of a greedy policy - only deeper
- Also suggests ways to select states to backup: smart focusing:



MCTS



The key is to generate, maintain, and utilize the ***initial segments*** of high-yielding trajectories.

Summary

- Emphasized close relationship between planning and learning
- Important distinction between distribution models and sample models
- Looked at some ways to integrate planning and learning
  - synergy among planning, acting, model learning
- Distribution of backups: focus of the computation
  - prioritized sweeping
  - small backups
  - sample backups
  - trajectory sampling: backup along trajectories
  - Planing at decision time: heuristic search/MCTS
- Size of backups: full/sample/small; deep/shallow