AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Model Free learning

Temporal Differencing

Q Learning

SARSA

Temporal Differencing (TD) Learning Combines Monte Carlo and Dynamic Programming

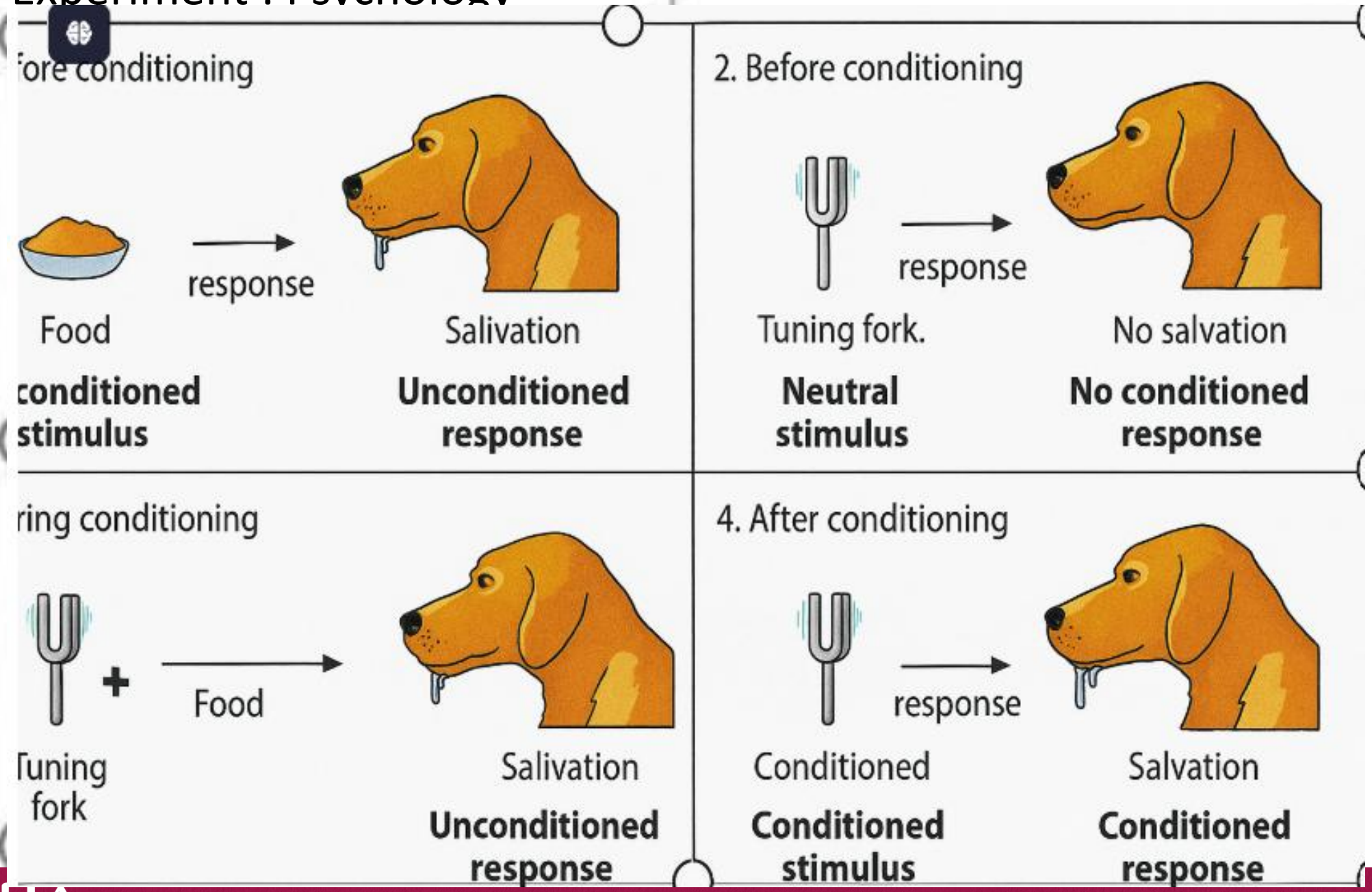
How Temporal Differencing (TD) Learning Combines Monte Carlo and Dynamic Programming

- **Monte Carlo Methods**:
  - Learn value estimates by **averaging returns after complete episodes**.
  - They require the **episode to finish** before updating values, which makes them unsuitable for continuing (non-episodic) tasks and can lead to high variance in updates.
- **Dynamic Programming (DP)**:
  - Uses a known model of the environment to **perform updates by bootstrapping** updating value estimates based on another estimate, not just actual returns.
  - DP requires full knowledge of transition and reward probabilities, which is often impractical.
- **TD Learning:** It uses both approaches
  - Like Monte Carlo, TD learning **does not need a model** of the environment (model-free).
  - Like DP, TD learning updates predictions based on other **estimates(bootstrapping**), not just actual returns.
  - TD methods can **learn after every step**, not just at episode end, making them suitable for non-episodic and sequential tasks.

Boot Strapping : Updating a value estimate **based on another estimate** — not waiting for the true outcome.

- The TD learning algorithm was introduced by the great Richard Sutton in 1988. The algorithm takes the benefits of both the Monte Carlo method and dynamic programming (DP) into account:
  - Like the DP method, it **doesn’t require model dynamics**, and
  - Like MC, it **doesn’t need to wait until the end of the episode** to make an estimate of the value function
  - Instead, temporal difference **learning approximates the current estimate based on the previously learned estimate**. This approach is also called bootstrapping.

Pavlov's Experiment : Psychology



What Pavlov's Experiment Shows

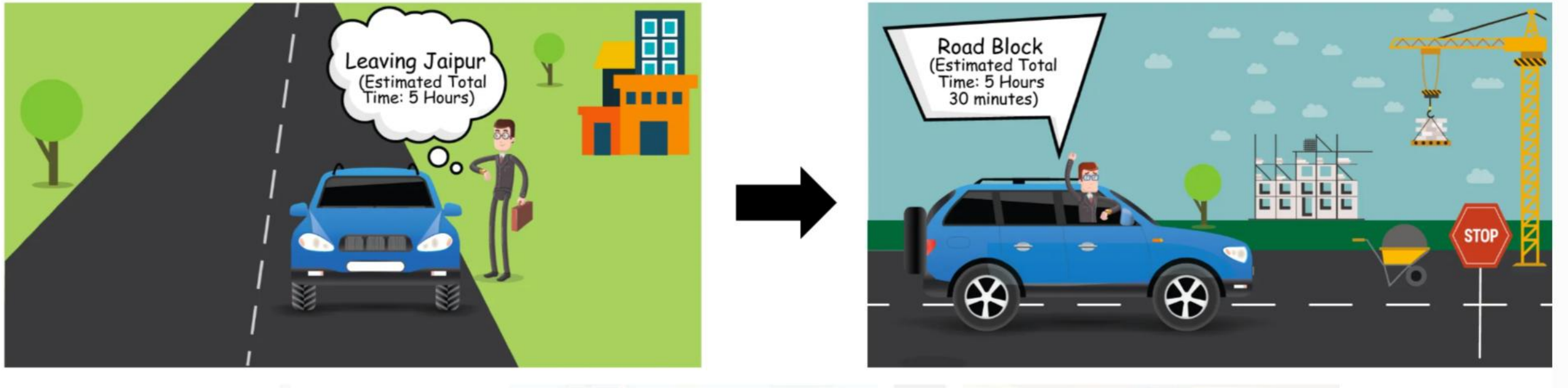
Phase	Stimuli & Response	Learning Meaning
Before Conditioning	Food → Dog salivates (natural response). Tuning fork → No response.	The food is the **unconditioned stimulus** (reward). The sound has **no predictive value** yet.
During Conditioning	Sound + Food repeatedly shown together.	Dog **learns to associate** the sound with the coming food.
After Conditioning	Sound → Dog salivates, even without food.	The sound now becomes a **predictor** of reward — it triggers the expectation of food.

Psychology (Pavlov)	Temporal Difference Learning
Dog associates bell → food	Agent associates state → reward
Learning occurs gradually over trials	Value function updated incrementally via TD error
Salivation shifts from food to bell	The brain (and TD algorithms) shift their **prediction signal** earlier in time — from the actual reward event to the **cue that predicts it**.
Bell predicts food before it arrives	$V(S_t)$ predicts future cumulative reward

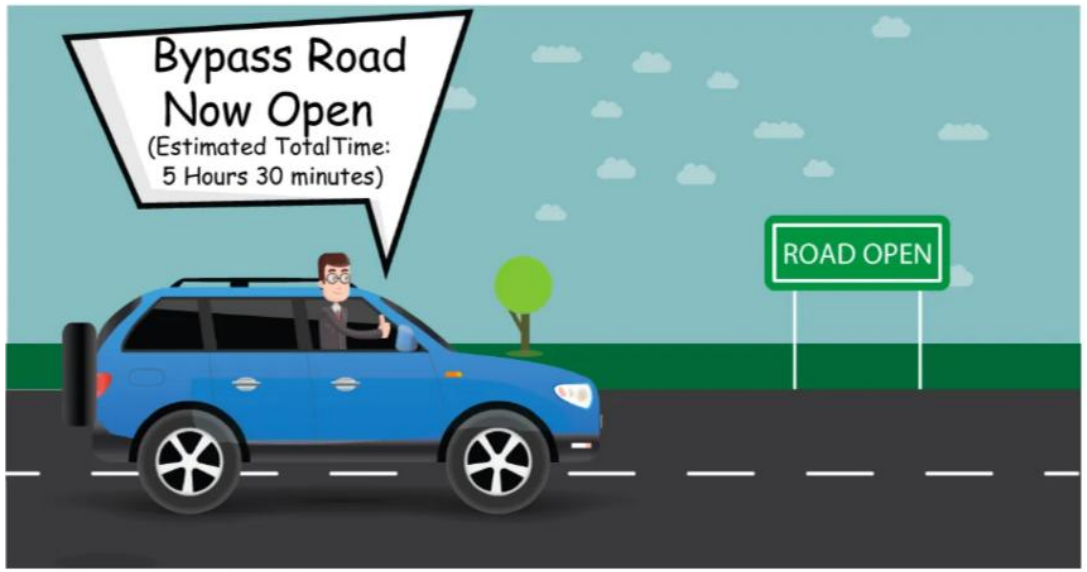
How TD Learning Mirrors Conditioning

In classical conditioning:
Learning happens by comparing *expected food* vs *actual food* at each time.
In TD learning:
Learning happens by comparing *predicted reward* vs *received reward + next prediction* — the **TD error**

Rajesh is planning to travel to Jaipur from Delhi in his car. A quick check on Google Maps shows him an estimated 5 hour journey. Unfortunately, there is an unexpected delay due to a roadblock (anyone who has taken a long journey can relate to this!). The estimated arrival time for Rajesh now jumps up to 5 hours 30 minutes.



Halfway through the journey, he finds a bypass that reduces his arrival time. So overall, his journey from Delhi to Jaipur takes 5 hours 10 minutes.



Did you notice how Rajesh's arrival time kept changing and updating based on different episodes? This, in a nutshell, illustrates the temporal difference learning concept

Temporal Differencing - Concept

Temporal Difference learning is all about **learning predictions by updating them over time**, using **partial information** before the final outcome is known.
Mathematically, the update rule for value prediction is:

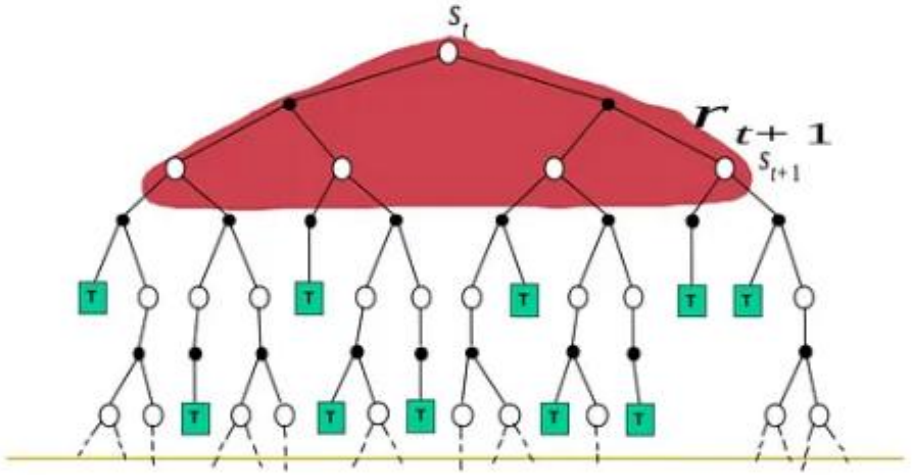
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Where:

- $V(S_t)$ =current estimate (e.g., expected remaining travel time at location S_t(
- R_{t+1} =immediate observation (e.g., delay or progress seen after next step)
- $V(S_{t+1})$ =new prediction based on updated information
- The **difference** $[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$is the **temporal difference error**

Dynamic Programming

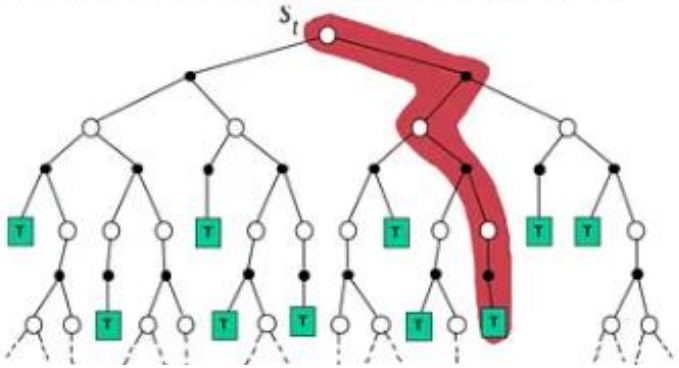
$$V(s_t) \leftarrow E_\pi\{r_{t+1} + \gamma V(s_t)\}$$



Monte Carlo Learning

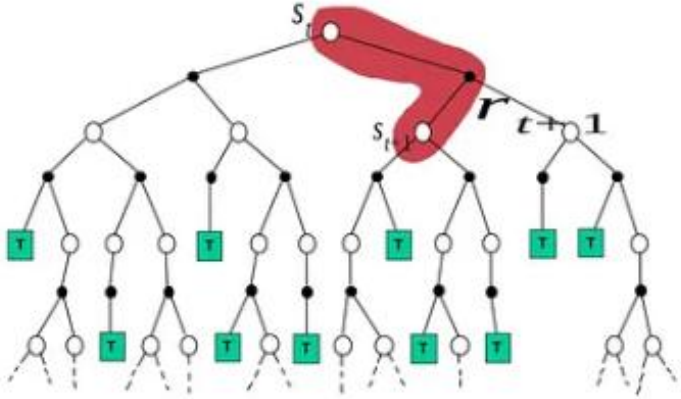
$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

where R_t is the actual return following state s_t.



Temporal Difference Learning

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$




```
IF (s, a) not in visit:          # first-visit update
    N[(s, a)] ← N[(s, a)] + 1
    Q[s][a] ← Q[s][a] + (G – Q[s][a]) / N[(s, a)] #incremental average
    π [s] ← greedy_action(Q, s, actions)   # policy improvement
```

Dynamic Programming

$$V(s_t) \leftarrow \mathbb{E}_\pi \left[r_{t+1} + \gamma V(s_{t+1}) \right]$$

Monte Carlo

$$V(s) \leftarrow V(s) + \alpha[G_t - V(s)]$$

```
IF (s, a) not in visit:        # first-visit update
    N[(s, a)] ← N[(s, a)] + 1
    Q[s][a] ← Q[s][a] + (G – Q[s][a]) / N[(s, a)] #incremental average
    π [s] ← greedy_action(Q, s, actions)   # policy improvement
```

$$Q(s, a) \leftarrow Q(s, a) + \alpha[G_t - Q(s, a)]$$

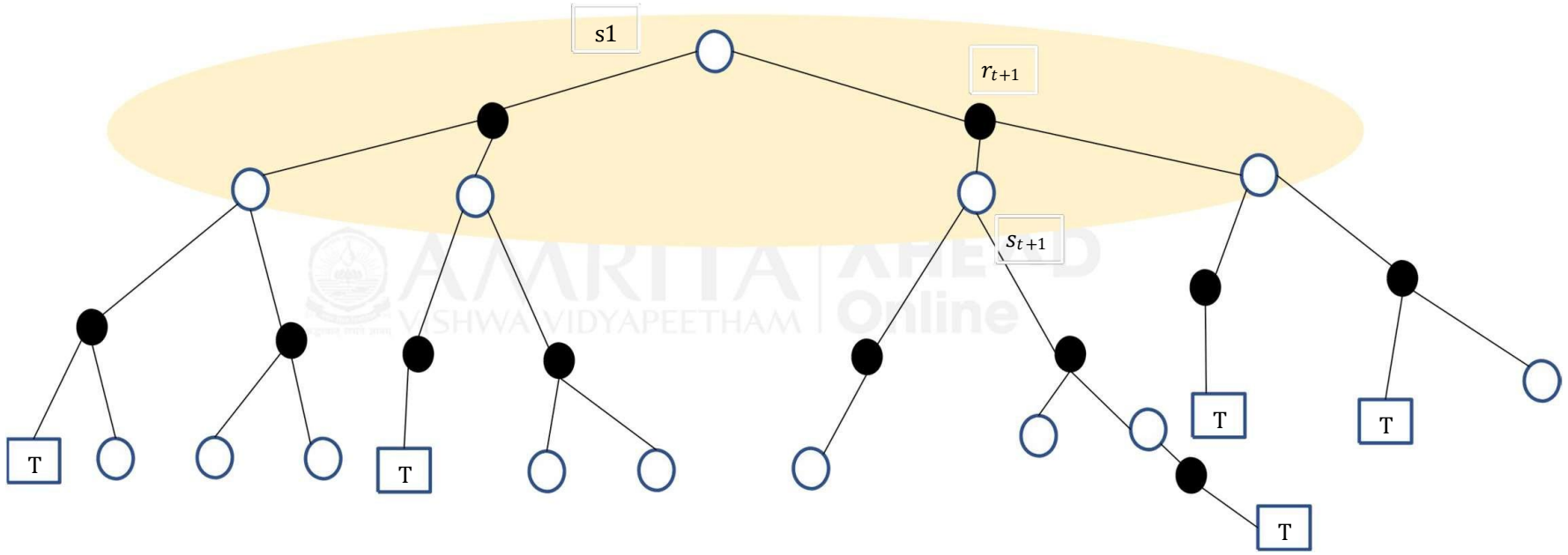
Temporal Differencing

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big]$$

Dynamic Programming

$$V(s_t) \leftarrow \mathbb{E}_\pi \left[r_{t+1} + \gamma V(s_{t+1}) \right]$$

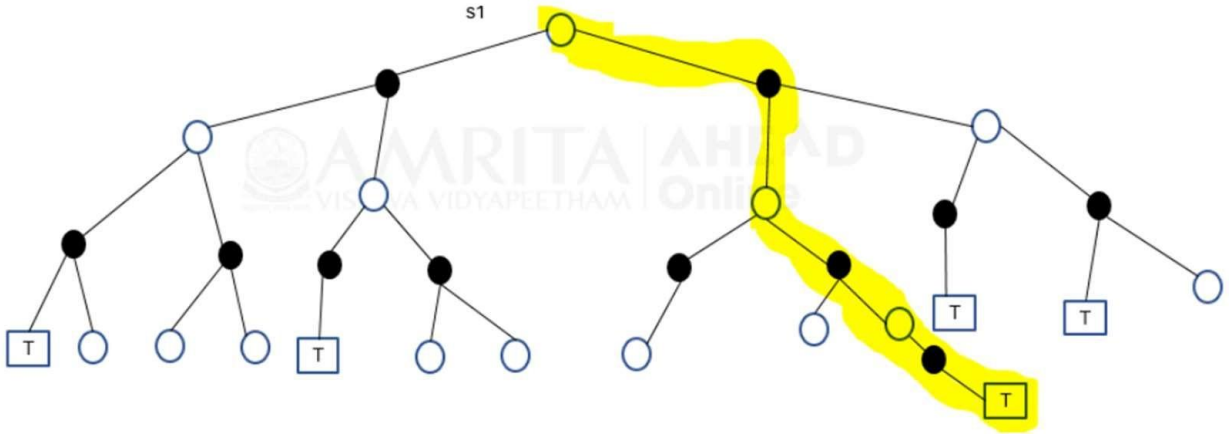
- $V(S_t) \leftarrow E\pi\{r_{t+1} - \gamma V(S_t)\}$



Simple Monte Carlo

$$Q(s,a) \leftarrow Q(s,a) + \alpha[G_t - Q(s,a)]$$

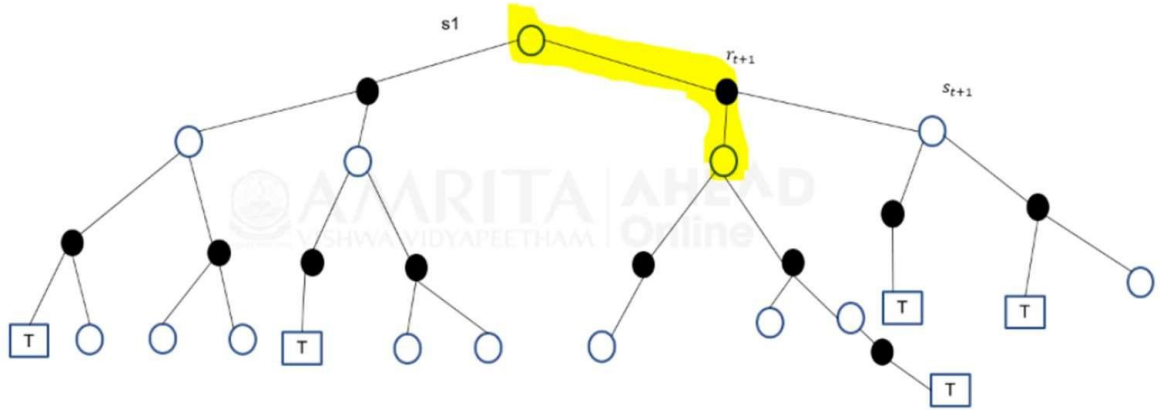
- $V(S_t) \leftarrow V(S_t) + \alpha[R_t - V(S_t)]$
- Where R_t is the actual return following state S_t



Monte Carlo use an estimate of the actual return

Simplest TD Method

- $$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$



TD samples the expected value and uses the current estimate of the value.

TD(0)

- After every step value function is updated with the value of the next state and along the way reward obtained.

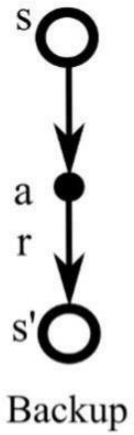
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1})]$$

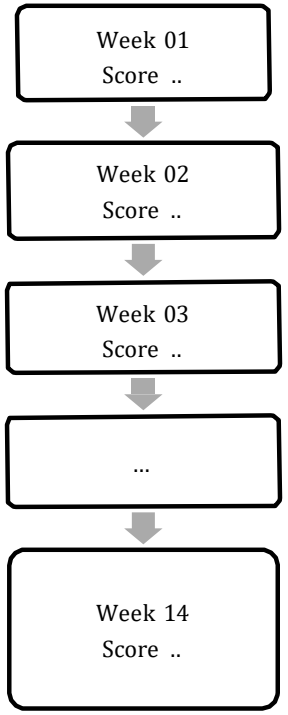
TD target = $R_{t+1} + \gamma V(S_{t+1})$

TD error $(\delta_t) = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

Instead of using the accumulated sum of discounted rewards (Gt) use only the immediate reward (Rt+1), plus the discount of the estimated value of only 1 step ahead (V(St+1))



Backup



Driving Home Example from Sutton-

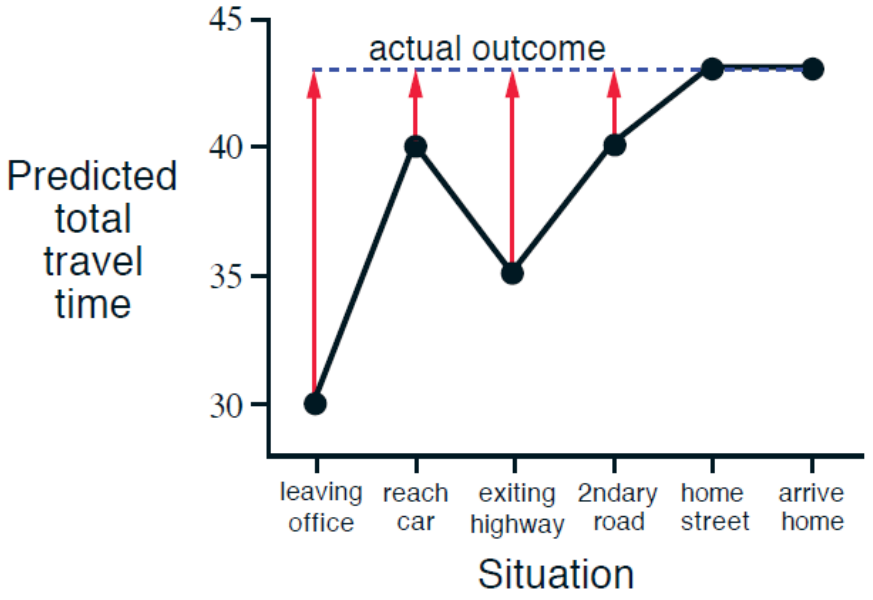
Sutton and Barto, Reinforcement Learning: An Introduction, The MIT Press Cambridge

Example 6.1: Driving Home Each day as you drive home from work, you try to predict how long it will take to get home. When you leave your office, you note the time, the day of week, the weather, and anything else that might be relevant. Say on this Friday you are leaving at exactly 6 o'clock, and you estimate that it will take 30 minutes to get home. As you reach your car it is 6:05, and you notice it is starting to rain. Traffic is often slower in the rain, so you reestimate that it will take 35 minutes from then, or a total of 40 minutes. Fifteen minutes later you have completed the highway portion of your journey in good time. As you exit onto a secondary road you cut your estimate of total travel time to 35 minutes. Unfortunately, at this point you get stuck behind a slow truck, and the road is too narrow to pass. You end up having to follow the truck until you turn onto the side street where you live at 6:40. Three minutes later you are home. The sequence of states, times, and predictions is thus as follows:

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

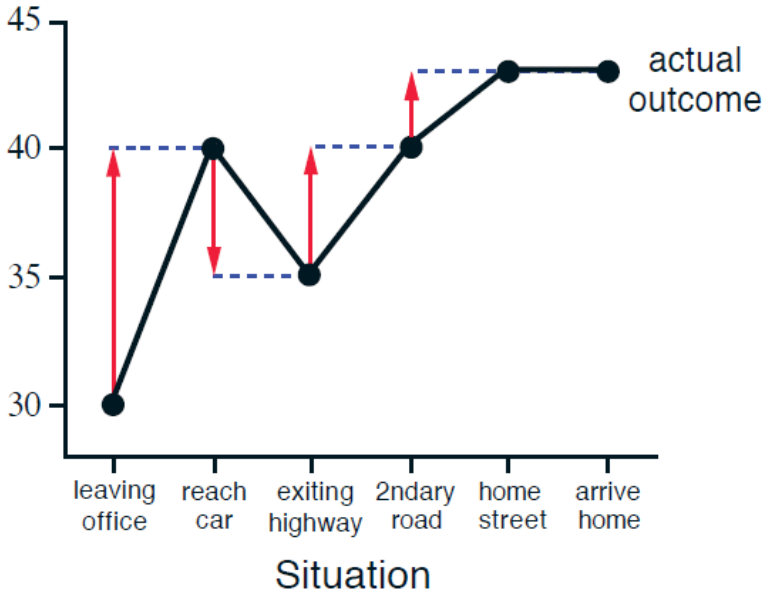
Driving Home Example

Changes recommended by Monte Carlo methods(α=1)



$$V(S_t) \leftarrow V(S_t) + \alpha[R_t - V(S_t)]$$

Changes recommended by TD methods(α=1)



$$V(S_t) \leftarrow V(S_t) + \alpha[r_{t+1} + \gamma V(S_{t+1})] - V(S_t)]$$

MC must wait the update until the final outcome

Sutton and Barto, Reinforcement Learning: An Introduction, The MIT Press Cambridge

Is it necessary to wait until the final outcome is known before learning can begin?

Observation

- TD updates **immediately** after each observation (e.g., when you see it's raining, or when you leave the highway).
- It uses the **difference between consecutive predictions** to adjust $V(S_t)$.
- This allows **continuous learning during the episode** — you don't have to wait until the trip is over.

Effect on the plot (right side):

- Predictions gradually improve at every stage.
- Each correction depends on the *next prediction* rather than waiting for the final outcome.
- This is **bootstrapping** — learning from partial information.

Conclusion – Driving Example

Aspect	Monte Carlo (MC)	Temporal Difference (TD)
When updates occur	After full trip (episode ends)	After every step (online)
What is used as target	Actual final return	One-step lookahead $r + \gamma V(S')$
Learning efficiency	Slow (needs complete episodes)	Fast (updates continuously)
Data requirement	Needs many full trips	Can learn from ongoing experience
Computation	Non-bootstrapped	Bootstrapped
Best for	Episodic problems with known final outcomes	Continuous/online environments

Advantages of TD

- TD methods do not require a model of the environment, only experience
- TD methods can be fully incremental
- can learn before knowing the final outcome
    - Less memory
    - Less computation
- can learn without the final outcome
    - From incomplete sequences

Numerical Example

THE PROBLEM: A SIMPLE LINE WORLD

An agent learns to predict the value of each state.

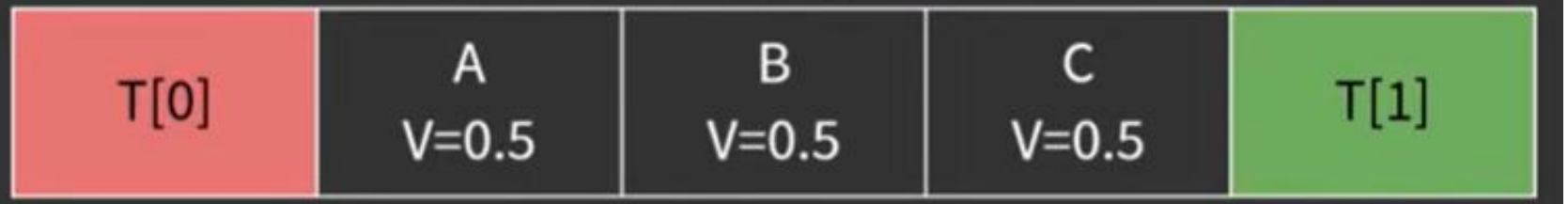
Rules:

- Start in state **B**.
- Policy: 50% chance Left, 50% Right.
- Rewards: **+1** for right terminal, **0** otherwise.
- Parameters: $\gamma = 1, \alpha = 0.1$.

Initial Brain (V-Values):

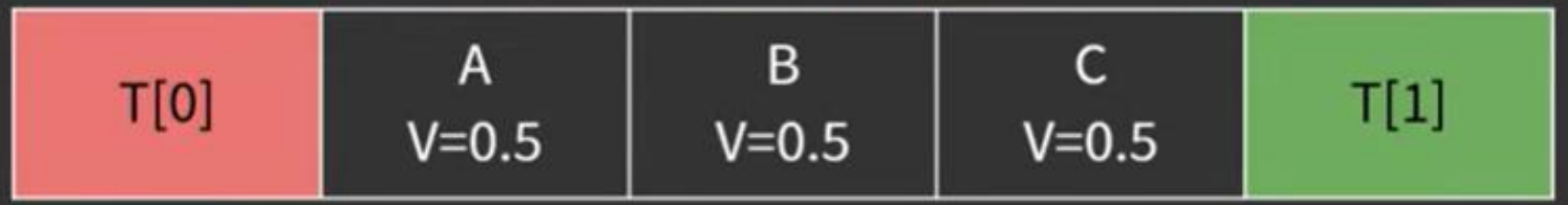
- `V(A) = 0.5`
- `V(B) = 0.5`
- `V(C) = 0.5`

EPISODE 1: A SHORT, UNLUCKY WALK



TRAJECTORY: B → A → Term(0)

EPISODE 1: A SHORT, UNLUCKY WALK



Step 1: B → A (Reward R=0)

TD Error: $\delta = [R + \gamma V(A)] - V(B)$

$$\delta = [0 + 1 \cdot 0.5] - 0.5 = \mathbf{0}$$

No surprise, no update. The agent thinks A and B are equally good.

EPISODE 1: A SHORT, UNLUCKY WALK

Step 2: A → Term(0) (Reward R=0, V(Term)=0)

TD Error: $\delta = [R + \gamma V(\text{Term})] - V(A)$

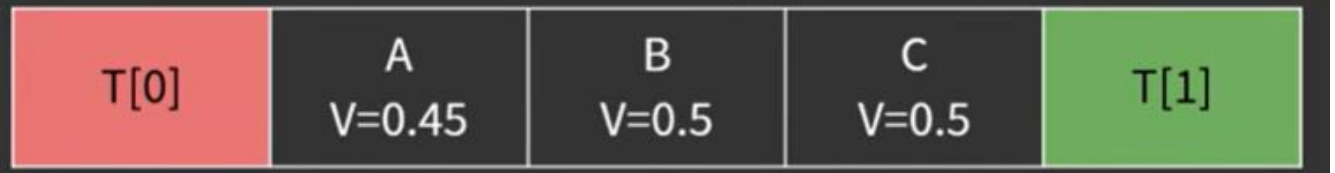
$$\delta = [0 + 1 \cdot 0] - 0.5 = \mathbf{-0.5}$$

Aha! A surprise! The agent was too optimistic.

Update V(A): $V(A) \leftarrow V(A) + \alpha \cdot \delta$

$$V(A) \leftarrow 0.5 + 0.1 \cdot (-0.5) = \mathbf{0.45}$$

EPISODE 2: A SHORT, LUCKY WALK

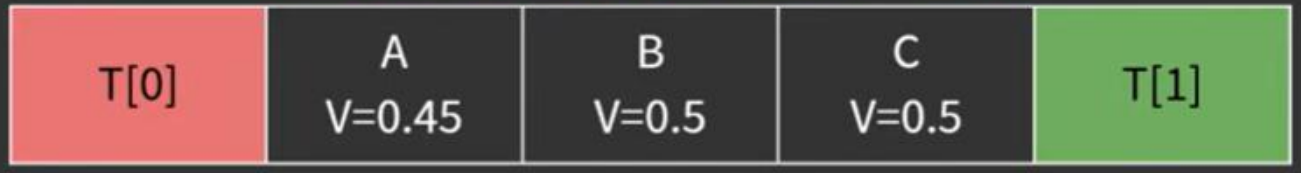


TRAJECTORY: B → C → Term(+1)

Step 1: B → C (Reward R=0)

TD Error: $\delta = [0 + 1 \cdot 0.5] - 0.5 = \mathbf{0}$. No surprise... yet.

EPISODE 2: A SHORT, LUCKY WALK



Step 2: C → Term(+1) (Reward R=+1, V(Term)=0)

TD Error: $\delta = [R + \gamma V(\text{Term})] - V(C)$

$$\delta = [1 + 1 \cdot 0] - 0.5 = \mathbf{0.5}$$

Another surprise! The agent was too pessimistic.

Update V(C): $V(C) \leftarrow V(C) + \alpha \cdot \delta$

$$V(C) \leftarrow 0.5 + 0.1 \cdot (0.5) = \mathbf{0.55}$$

KNOWLEDGE AFTER EPISODE 2

T[0]	A V=0.45	B V=0.5	C V=0.55	T[1]

So far, so good. But notice, its opinion of the starting state, B, hasn't changed at all.

EPISODE 3: THE MAGIC TRICKLE-BACK

T[0]	A V=0.45	B V=0.5	C V=0.55	T[1]

Trajectory: `B → C → B → A → Term(0)`

Step 1: B → C (Reward R=0)

TD Error: $\delta = [R + \gamma V(C)] - V(B)$

$$\delta = [0 + 1 \cdot \mathbf{0.55}] - 0.5 = \mathbf{0.05}$$

THIS IS THE KEY MOMENT!

Update V(B): $V(B) \leftarrow 0.5 + 0.1 \cdot (0.05) = \mathbf{0.505}$

It used its new guess for C to improve its guess for B. This is bootstrapping!

EPISODE 3: THE LEARNING CASCADE

Step 2: C → B

$$\delta = [0 + 1 \cdot 0.505] - 0.55 = -0.045$$

Update V(C): $\rightarrow 0.5455$

Step 3: B → A

$$\delta = [0 + 1 \cdot 0.45] - 0.505 = -0.055$$

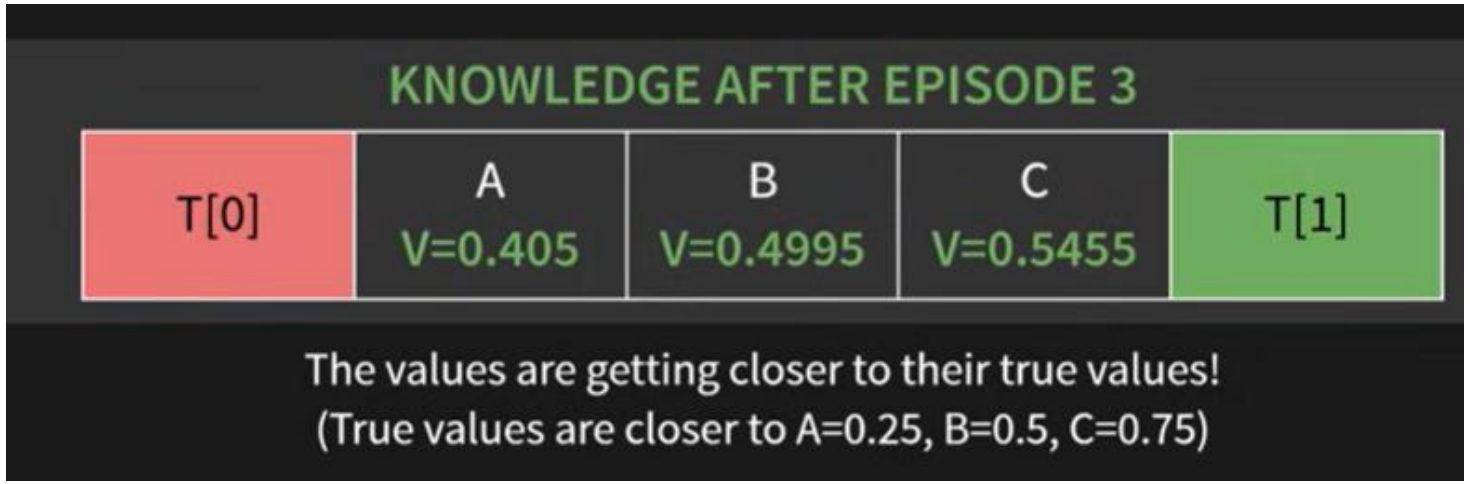
Update V(B): $\rightarrow 0.4995$

Step 4: A → Term(0)

$$\delta = [0 + 1 \cdot 0] - 0.45 = -0.45$$

Update V(A): $\rightarrow 0.405$

Prediction problem



This is great in predicting how good a state is But

It does not say to our Agent what to do?

So……Move from Knowing to Doing

This leads to SARSA

State Action Reward State Action

Knowing the Value of a state is prediction
But to make decisions it needs to know the value of an action

We need to move from V(S) to Q(S)

Prediction problem → Control Problem

SARSA – TD Control Algorithm

1. You start in a **State (S)**
2. You take an **Action (A)**
3. You receive a **Reward (R)**
4. You land in a new **State (S')**
5. You choose your next **Action (A')**

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Swap Qs for Vs

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

$$R + \gamma Q(S', A')$$

The SARSA TD Target

We update our *current* action `Q(S, A)` based on the reward we got, plus the discounted value of the *next action A'* that we **actually decide to take**.

SARSA Algorithm (ON-POLICY TD control)

1. Initialize Q(s, a) for all states and actions
2. Repeat (for each episode):
    1. Initialize state S
    2. Choose action A from S using an ε-greedy policy

    3. Repeat (for each step):
        a. Take action A, observe reward R and next state S'
        b. Choose the NEXT action A' from S' using the ε-greedy policy
        c. Update the Q-value for the PREVIOUS step:
            $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$
        d. Update for the next loop: $S \leftarrow S'$; $A \leftarrow A'$

    Until S is terminal

SARSA is a CAUTIOUS REALIST

SARSA follows ON-POLICY

Why SARSA is said as a CAUTIOUS REALIST?

To update its Q-value, SARSA needs to know `A'`, the *actual action* it's going to take next.

It learns the value of its actions while accounting for its own imperfect, exploratory behavior.

Why you say SARSA follows ON-POLICY

The Q-values SARSA learns are directly tied to the policy it's currently following.

It knows there's a small chance its own policy will randomly decide to jump off a cliff.

This Realism Has DOWNSIDES

It can be overly Cautious

What if we don't want to account for our exploration

What if we want our agent to learn the perfect Optimal Path even while we are stumbling around exploration?

Can we separate Exploration from Learning?

Can our agent learn cautiously but Greedly?

YES

Q-Learning

Q Learning is an OFF POLICY Algorithm

This means the policy it learns about (the optimal one)…
…is different from the policy it uses to act (the exploratory one).

Q-Learning

It’s like a student who gets a C on a practice test…

…but updates their belief about their potential by looking at the answer key for an A+ student.

The difference- SARSA, Q-LEARNING

Algorithm snippet

SARSA (The Cautious Realist)

ON- POLICY

Repeat (for each step):
- a. Take action A, observe R and S'
- b. Choose A' from S' using ε-greedy policy
- c. Update:
  - $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$
- d. $S \leftarrow S'$; $A \leftarrow A'$

SARSA's Target

$$R + \gamma Q(S', A')$$

It uses the Q-value of `A'`, the **actual** next action its exploratory policy chose.

Perfect Greedy Policy

Q-Learning (The Bold Optimist)

OFF POLICY

Repeat (for each step):
- a. Choose A from S using ε-greedy policy
- b. Take action A, observe R and S'
- c. Update:
  - $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma$ **max_a** $Q(S', a) - Q(S, A)]$
- d. $S \leftarrow S'$

Q-Learning's Target

$$R + \gamma \max_a Q(S', a)$$

It **ignores** the next action and finds the Q-value of the absolute **best** one.

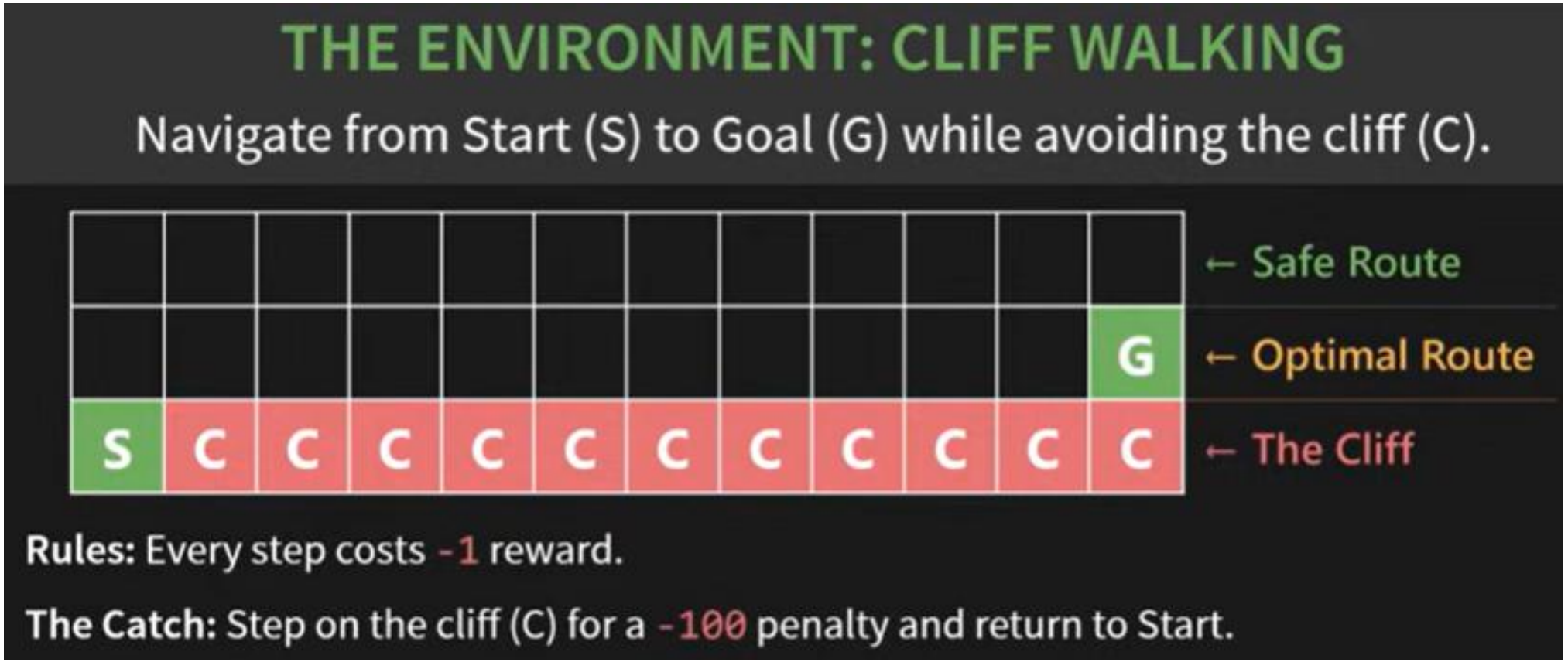
Which approach is actually better?

The **cautious realist** that learns from its own mistakes?

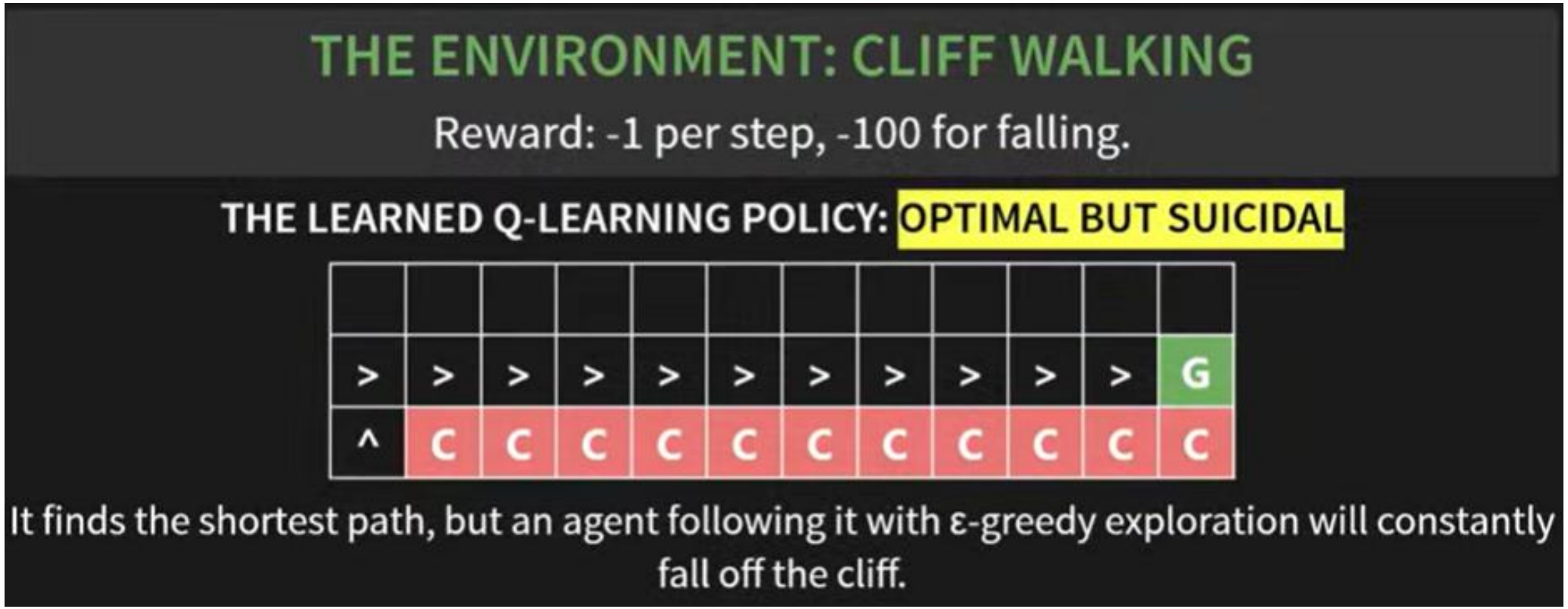
Or the **bold optimist** that learns from a perfect ideal?

SARSA vs Q-Learning
ON-POLICY vs OFF-POLICY

CLIFF WALKING PROBLEM

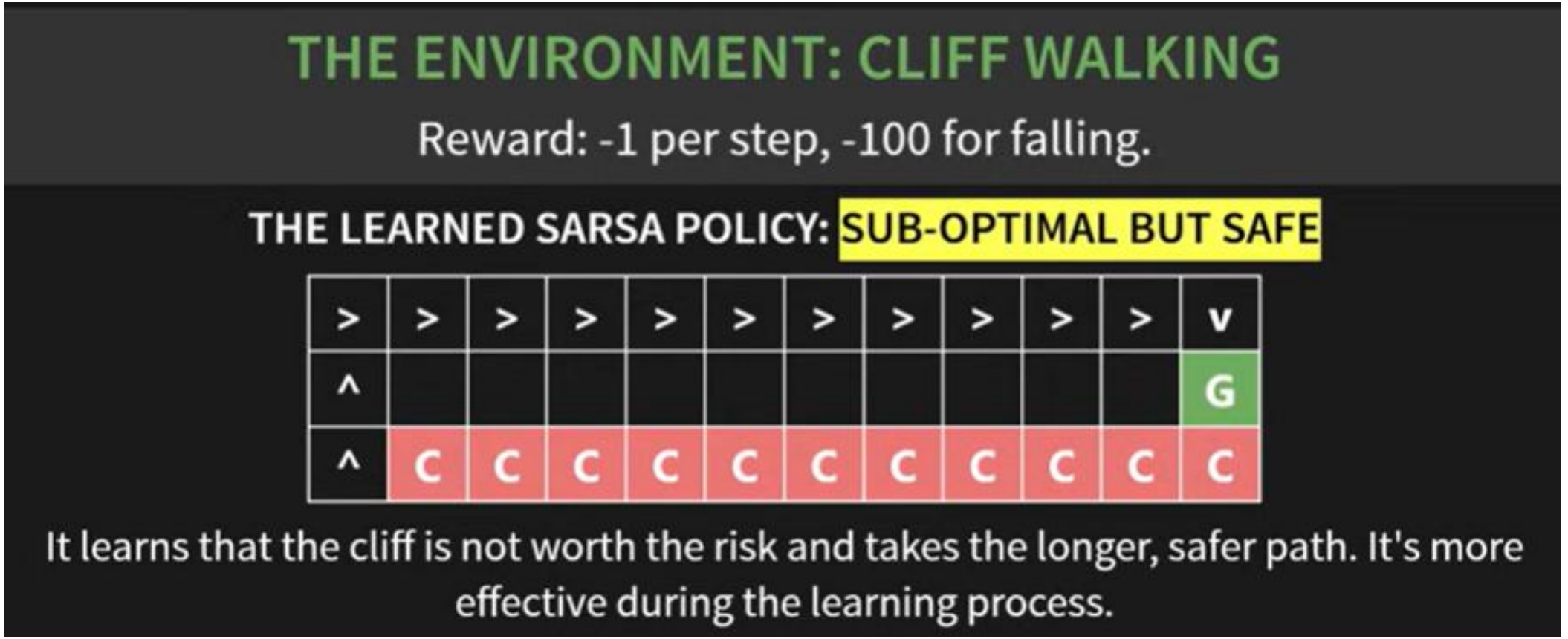


Q-Learning, The BOLD Optimist



Q-Learning is forced to learn the best possible future completely ignoring the risks of its own exploration

SARSA- The CAUTIOUS REALIST



Q-Learning (Off-Policy)

Answers: *"What is the best path, assuming I execute it perfectly?"*

Learns the optimal policy, but the learning process can be unstable.

SARSA (On-Policy)

Answers: *"What is the best path for me, a clumsy agent still exploring?"*

Learns a safer policy that accounts for its own imperfections.

Neither one is strictly "better."

They are two different **tools** for two different **jobs.**

Autonomous drone or robot navigation in a crowded, dynamic environment

SARSA is better:- The Cautious Realist

-The agent's policy is ε-greedy, meaning it still explores (sometimes making risky moves).

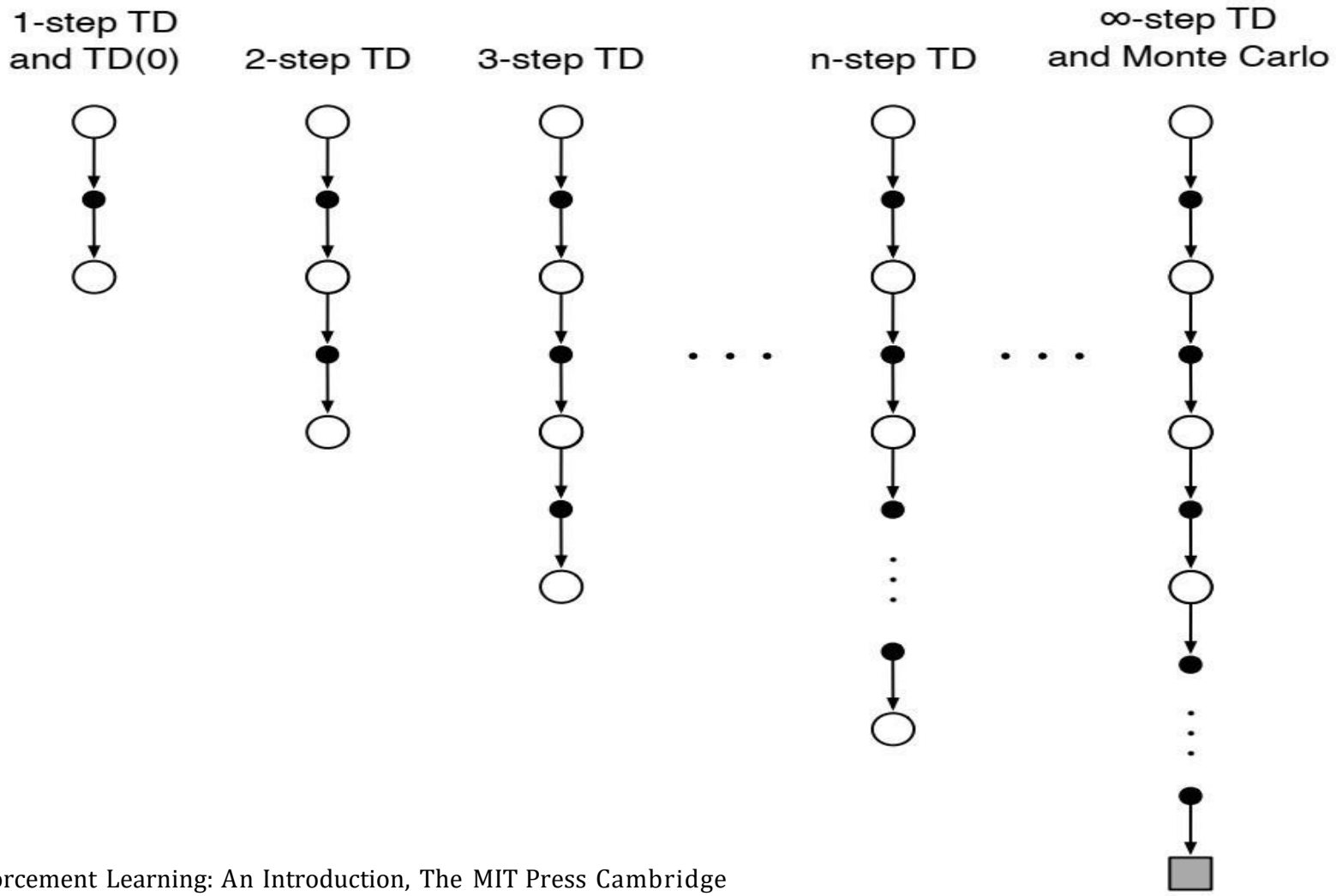
- SARSA updates using the actual next action A′ — it learns with awareness of its own exploration errors.
- This produces safer, more conservative policies.

Game-playing agent or simulation-based training (e.g., Atari, chess, or gridworld)

Q-Learning Fits Best — "The Bold Optimist"

- The agent can afford to take risks — failure doesn't cause physical harm.
- Q-learning updates using the max_a Q(S′, a) — it assumes the next action will be the best possible one.
- This makes it more aggressive and usually converges faster to the optimal policy, but can be unstable in noisy settings.

n-Step Prediction



Sutton and Barto, Reinforcement Learning: An Introduction, The MIT Press Cambridge