# Machine Learning

## Imperial College London

**Nuri Cingillioglu**

### Part 7: Agency & Evolution

---

Do or do not, there is no try. Or is there?

# Agency

Real biological intelligence does not passively categorise static image datasets.

An organism **acts**, **probes**, and **navigates** through an uncertain environment:

| Attribute | Supervised Learning | Unsupervised Learning | Reinforcement Learning |
|---|---|---|---|
| **Guidance** | Corrective ($\hat{y}$ vs. $y$) | None (Density / Geometry) | Evaluative (Scalar reward $r$) |
| **Feedback** | Instantaneous | Passive | Delayed across long horizons |
| **Data Source** | Static i.i.d. dataset | Static unlabelled dataset | **Active, non-i.i.d. interaction** |
| **Agent Influence** | None (Observer) | None (Observer) | **Actions change future observations** |

# Animal Conditioning

## Classical Conditioning (Ivan Pavlov, 1904)

- **Passive associative prediction:** An animal learns correlations between neutral cues (metronome / bell) and unconditioned stimuli (food).

$$\text{Cue} \longrightarrow \text{Anticipation of Outcome}$$



# Animal Conditioning

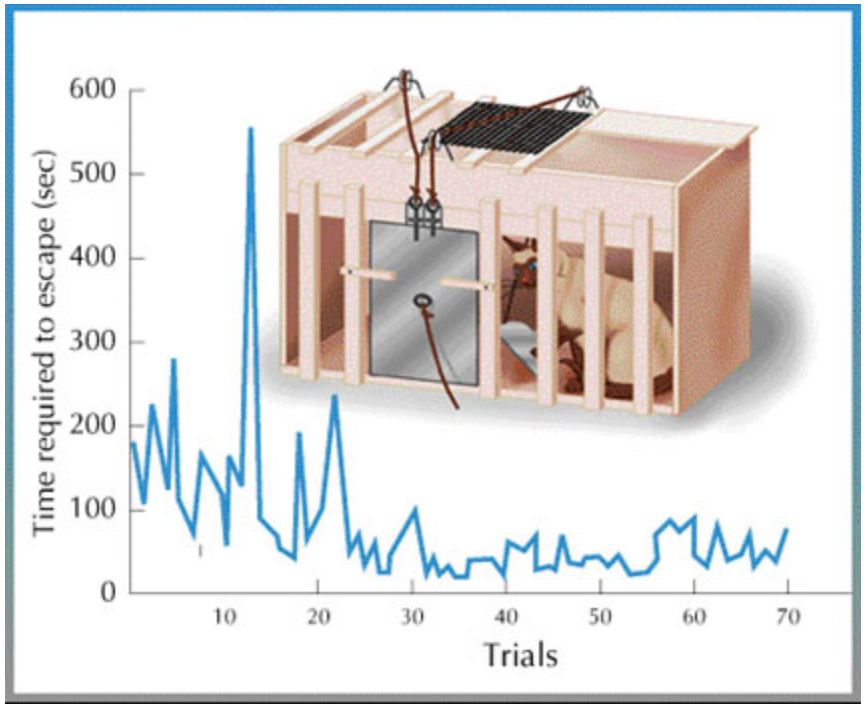
## Instrumental Conditioning & The Law of Effect (Edward Thorndike, 1898)

- **Active trial-and-error discovery:** A cat trapped inside a puzzle box claws haphazardly until accidentally tripping the latch.
- **The Law of Effect:** Actions followed by satisfying states are stamped into memory; actions followed by discomfort are extinguished.

> My 2 year old niece randomly touches everything in the house until she gets hurt...



# Thorndike's Puzzle Box (1898)




> The PhD student randomly tries different things until something works. Over repeated trials, they learn to publish more papers.

# Why is RL Difficult?

**1. The Credit Assignment Problem**

When an agent wins a game of chess after 60 moves, *which specific move was responsible for victory?* The decisive move may have occurred 40 steps earlier.

**2. Non-Stationary, Non-i.i.d. Data**

In supervised learning, samples $\mathbf{x}_i \sim P(\mathbf{X})$ are identically distributed. In RL, **the agent's policy directly determines the data it collects**:

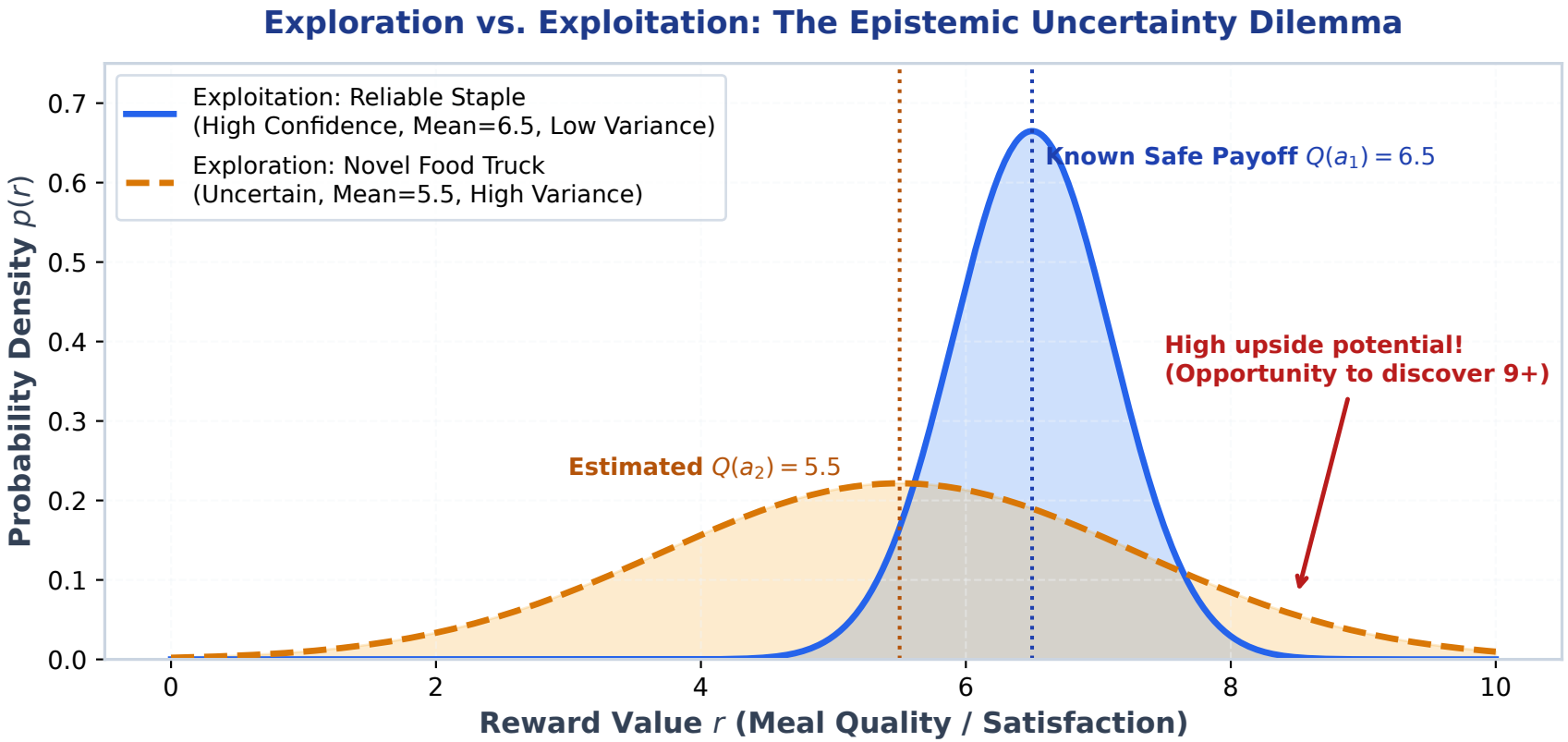
$$\pi_\theta \implies \text{Trajectory } (S_0, A_0, S_1, A_1, \ldots) \sim P_{\pi_\theta}$$

Change the policy, and the entire training distribution shifts...

**3. The Exploration vs. Exploitation Trade-off**

To discover high rewards, the agent must venture into the unknown. But to maximise cumulative score, it must exploit what it already knows works.

# Food at Imperial




- **Exploit:** Visit your dependable college café. You are guaranteed a ~~solid~~ lunch ($r \approx 6.5$).
- **Explore:** Try a new outlet. It might be awful ($r \approx 2.0$), or it might be magnificent ($r \approx 9.5$).

# The Multi-Armed Bandit

To investigate exploration vs. exploitation, we define the $K$**-Armed Bandit Problem**:

Imagine standing before $K$ slot machines (arms) $a \in \{1, \ldots, K\}$:

1. At each step $t = 1, 2, \ldots, T$, the agent pulls an arm $a$.
2. The arm delivers a numerical reward $R_t$ with an unknown true average payoff:

$$\mu_a \triangleq \mathbb{E}[R \mid \text{pull arm } a]$$

3. The true averages $\{\mu_1, \ldots, \mu_K\}$ are completely hidden from the agent!

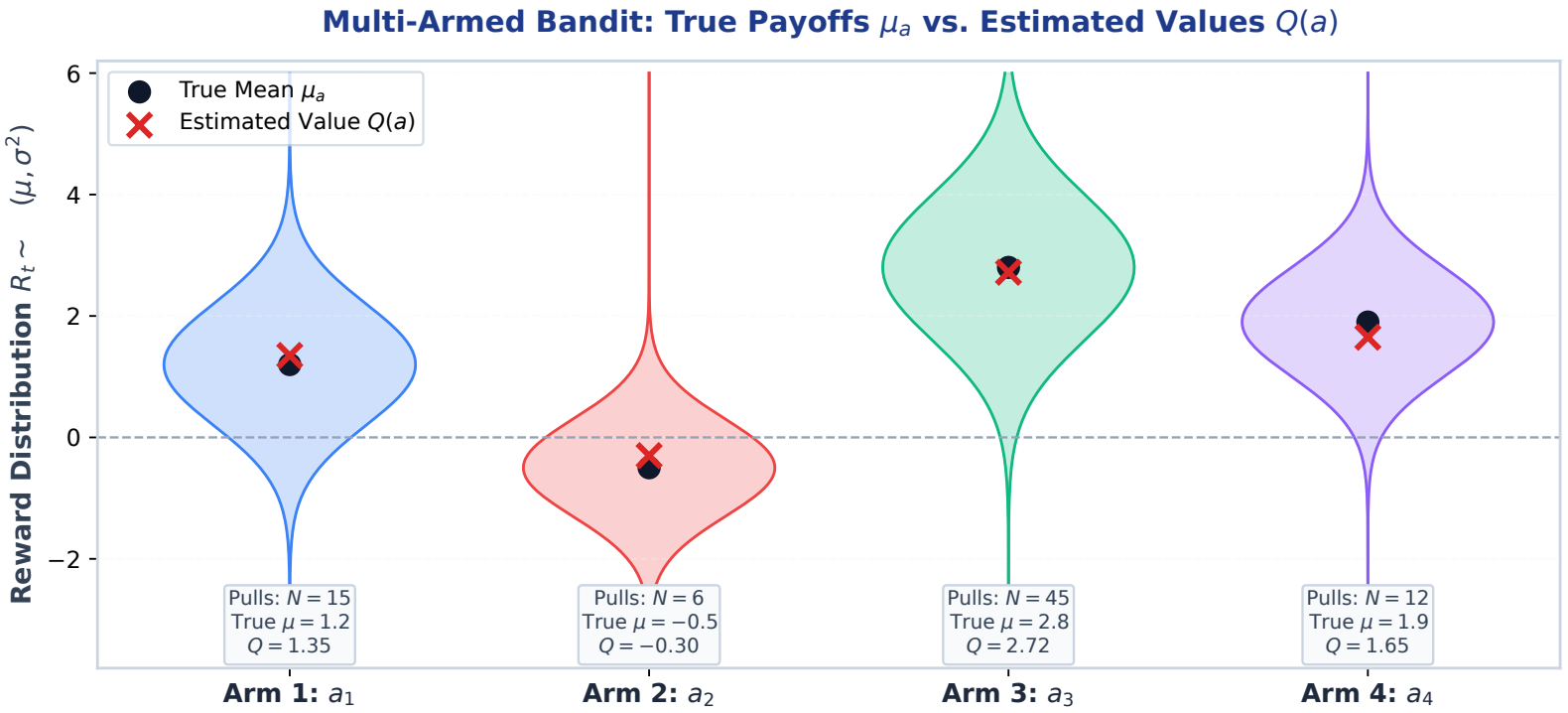
The objective is simply to **maximise total cumulative reward** over $T$ steps:

$$\max \sum_{t=1}^{T} R_t$$

> **The Dilemma:** You must explore unfamiliar arms to discover which $\mu_a$ is highest, but exploit the current best arm to accumulate reward 🤔



# Payoff Distributions




We estimate the **Value (Quality)** $Q(a)$ of each arm by taking the sample average of the rewards received when pulling that arm:

$$Q(a) \triangleq \frac{\text{sum of rewards received from arm } a}{\text{number of times arm } a \text{ was pulled}} = \frac{1}{N(a)} \sum_{i=1}^{N(a)} R_i$$

# Incremental Value Updates

Computing $Q(a)$ by summing all past rewards requires storing every historical pull in memory.

Instead, let's derive an **incremental update rule** for an arm after receiving its $n$-th reward $R_n$:

$$
\begin{aligned}
Q_n &= \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}(R_n + (n-1)Q_{n-1}) = \frac{1}{n}(R_n + nQ_{n-1} - Q_{n-1}) \\
Q_n &= \underbrace{Q_{n-1}}_{\text{Old Estimate}} + \underbrace{\frac{1}{n}}_{\text{Step Size } \alpha}\Big[\underbrace{R_n}_{\text{Target}} - \underbrace{Q_{n-1}}_{\text{Old Estimate}}\Big]
\end{aligned}
$$

WHOA! This is our usual **update rule** where the bracketed term is the **prediction error**:

$$\text{Error} = R_n - Q_{n-1}$$

# Why this works?

The key term you are looking for is **the law of large numbers**. As $n \to \infty$, the estimate $Q_n$ converges to the true mean $\mu_a$.

$$
\begin{aligned}
Q_n &= \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \underbrace{Q_{n-1}}_{\text{Old Estimate}} + \underbrace{\frac{1}{n}}_{\text{Step Size } \alpha}\left[\underbrace{R_n}_{\text{Target}} - \underbrace{Q_{n-1}}_{\text{Old Estimate}}\right]
\end{aligned}
$$

$$
\lim_{n\to\infty} Q_n = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} R_i = \mathbb{E}[R] = \mu_a
$$

> Which means you *eventually* learn, this is the trial and error process.

# Action Selection

How should the agent choose which arm to pull given its current estimates $Q(a)$?

**1. $\epsilon$-Greedy Selection**

With probability $1 - \epsilon$, exploit the best arm; with probability $\epsilon$, explore uniformly at random:

$$a = \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \epsilon \\ \text{Random arm} & \text{with probability } \epsilon \end{cases}$$

**2. Softmax (Boltzmann) Selection**

Convert value estimates into smooth selection probabilities via temperature $\tau > 0$:

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^{K} e^{Q(b)/\tau}}$$

> Notice that we have a toolkit like softmax that keeps on appearing in different areas of AI.

# Interactive Multi-Armed Bandit Playground

🎰 **Multi-Armed Bandit Testbed** Stateless Exploration

ε-Greedy | Softmax (Boltzmann) | ▶ Step 1 | ▶▶ Run 50 | ↺ Reset

### ACTION SELECTION STRATEGY

Exploration Rate (ε): 0.10

Probability of choosing a random arm

### PERFORMANCE METRICS

Total Steps (T)
**0**

Total Cumulative Reward
**0.0**

Cumulative Regret
**0.0**

% Optimal Arm Chosen
**0.0%**

💡 Click any arm card directly to manually pull it!

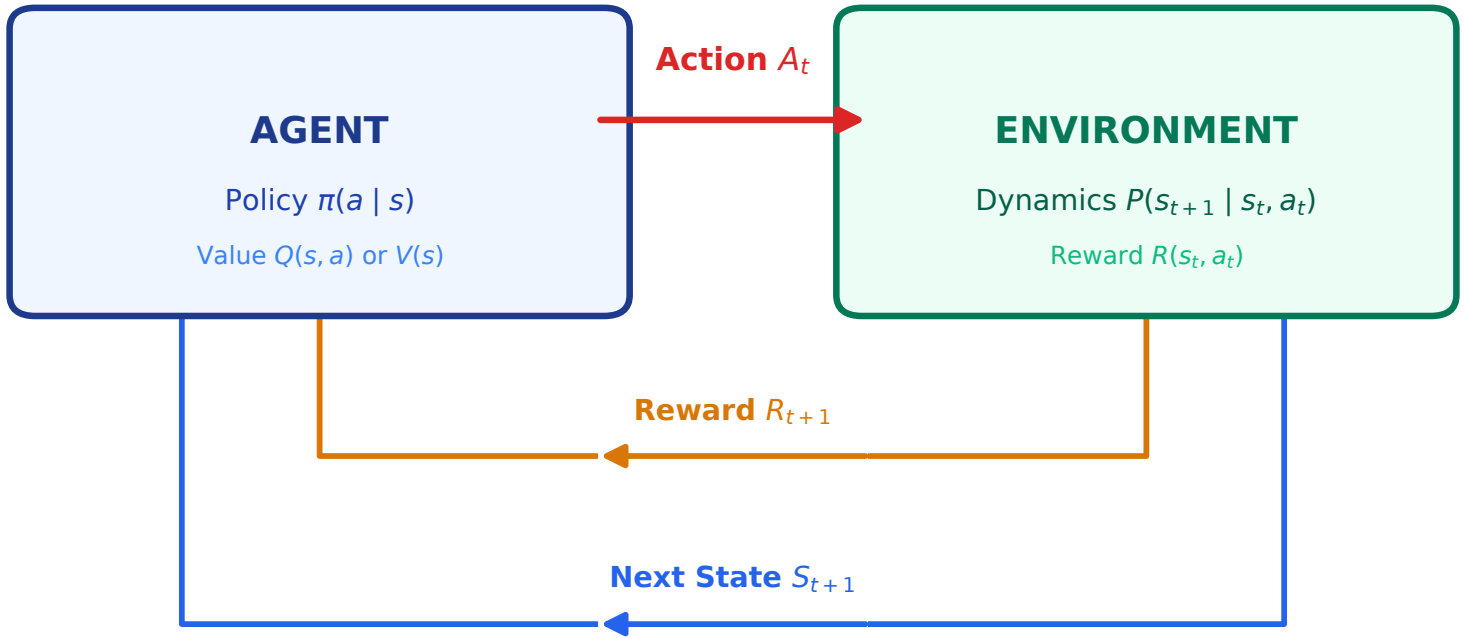
| Arm 1 | Arm 2 | Arm 3 (★ Best) | Arm 4 |
|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 |
| 0 pulls (0%) | 0 pulls (0%) | 0 pulls (0%) | 0 pulls (0%) |
| Pull Arm 1 | Pull Arm 2 | Pull Arm 3 | Pull Arm 4 |

**Cumulative Regret Growth Curve: R(T) = Σ (q* - E[R_t])**

Pull arms to accumulate regret curve...

### LIVE DECISION STREAM

```
Ready. Press Step or Run.
```




*Closed-loop feedback: The agent's actions actively change its future state distribution!*

- **State $S_t \in \mathcal{S}$:** The agent's current sensory representation.
- **Action $A_t \in \mathcal{A}$:** The agent's selected motor command.
- **Reward $R_{t+1} \in \mathbb{R}$:** Scalar evaluative feedback emitted by the environment.
- **Next State $S_{t+1} \in \mathcal{S}$:** The environment transitions into a new state.

# Markov Decision Processes (MDP)

Bandits are stateless. In real tasks, **actions change your state in the world**.

A discrete **Markov Decision Process (MDP)** is formalised as a 4-tuple:

$$M = \langle \mathcal{S}, \mathcal{A}, p, \gamma \rangle$$

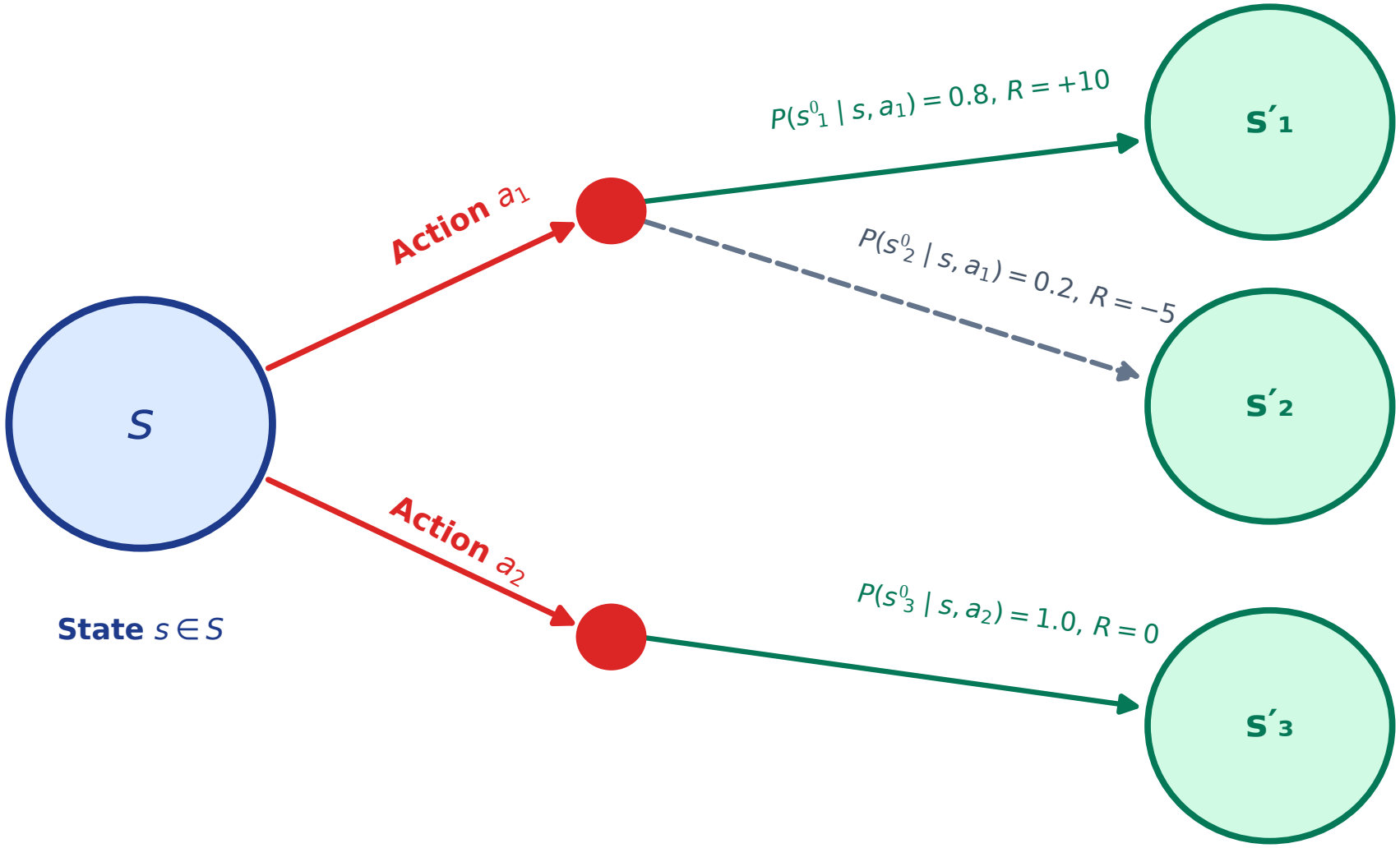
- $\mathcal{S}$: The set of all valid environmental states.
- $\mathcal{A}$: The set of all valid agent actions.
- $p$: Joint transition-reward dynamics:

  $$p(s', r \mid s, a) \triangleq P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

  Given state $s$ and action $a$, the environment determines the next state $s'$ and reward $r$.
- $\gamma \in [0, 1)$: Temporal discount factor.

> Some gymnastics, what is $p(s'|s, a)$ and $\mathbb{E}[R|s, a]$?

# Markov Decision Process (MDP): State → Action → Transition Dynamics



# The Markov Property

> *"The future is conditionally independent of the past given the present."*

Mathematically, the transition probability distribution satisfies:

$$P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a, S_{t-1}, A_{t-1}, R_t, \ldots, S_0, A_0) = p(s', r \mid s, a)$$

**What does this mean intuitively?**

- The current state $S_t$ is **sufficient** to predict the future.
- You do not need to record the full history of past moves to predict the future.
- If an environment violates this (e.g., poker with hidden cards or a robot lacking velocity sensors), it is a **Partially Observable MDP (POMDP)**.

# Randomness in RL

Reinforcement learning has **two distinct random actors** interacting in a closed loop:

1. **The Agent (Policy $\pi$):** Selects an action with probability:

$$\pi(a \mid s) \triangleq P(A_t = a \mid S_t = s)$$

2. **The Environment (Dynamics $p$):** Simultaneously determines where you land and what reward you receive:

$$p(s', r \mid s, a) \triangleq P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

> Everything in RL is about these: $\pi$ controls the agent, while $p$ governs the *world*.

# From Bandits to Sequential Decisions

The bandit update moves an estimate towards the observed reward:

$$Q(a) \leftarrow Q(a) + \alpha \left[ \underbrace{r}_{\text{target}} - Q(a) \right]$$

In a MDP, the action also takes us to $s'$. **What is the target?**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ \underbrace{???}_{\text{target}} - Q(s,a) \right]$$

# Tabular Q-Learning (Watkins, 1989)

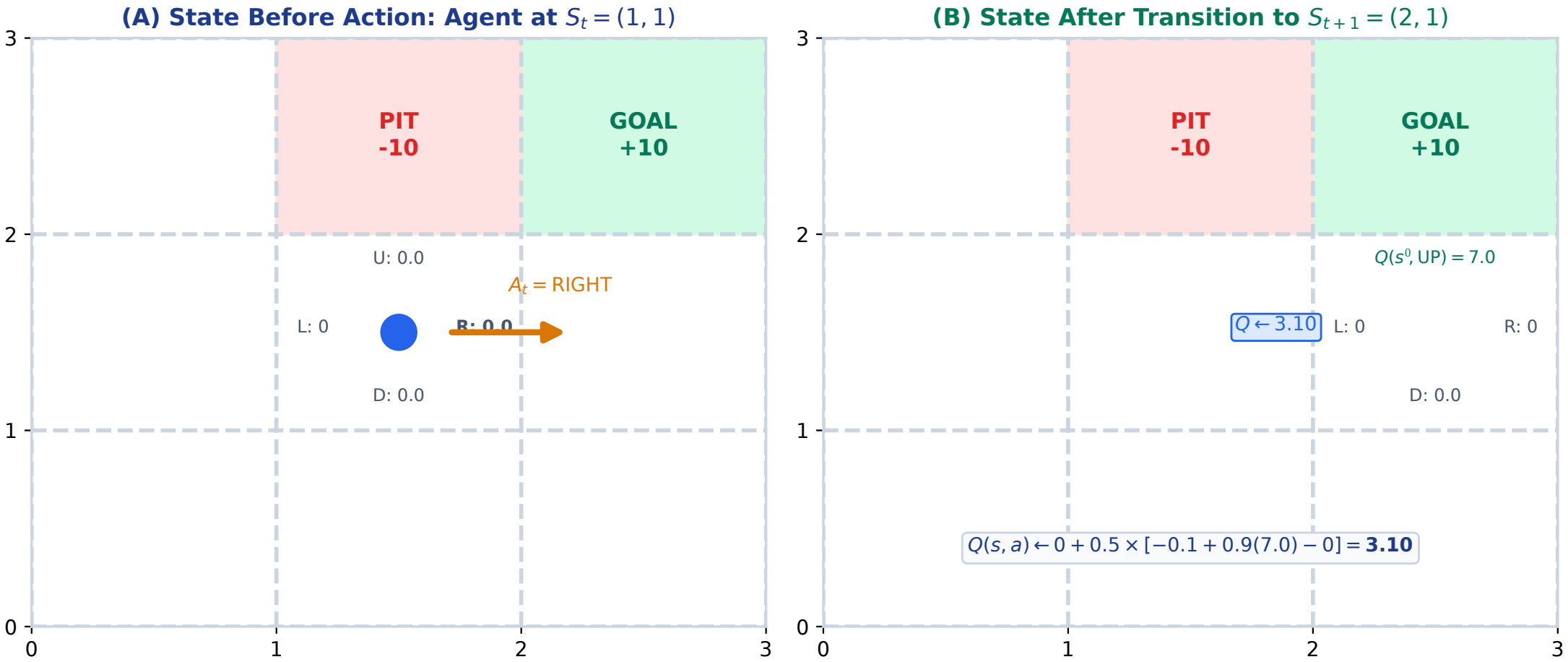
Christopher Watkins formulated $Q$**-Learning**, an off-policy model-free TD control algorithm. When the agent takes action $a$ in state $s$, receives reward $r$, and lands in state $s'$:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \left[ \underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Target}} - Q(s,a) \right]$$

- **Off-Policy:** The agent acts using an exploratory policy (e.g. $\epsilon$-greedy), but updates $Q$ towards the greedy maximum $\max_{a'} Q(s',a')$.
- **Model-Free:** The agent does not need to know the transition dynamics $p(s', r \mid s, a)$ in advance. It learns purely from experience.

# Step-by-Step Manual Trace on a Toy Grid

Let's manually compute a single Q-learning update step on a $3 \times 3$ grid:



1. **Prior Values:** Agent is at $S_t = (1, 1)$ with $Q((1, 1), \text{RIGHT}) = 0.0$.
2. **Action & Outcome:** Agent moves RIGHT to $S_{t+1} = (2, 1)$, receives step cost $R_{t+1} = -0.1$.
3. **Target Estimation:** In state $(2, 1)$, the best action is UP towards the goal, with $Q((2, 1), \text{UP}) = 7.0$.
4. **TD Arithmetic ($\alpha = 0.5, \gamma = 0.9$):**

$$\text{TD Target} = -0.1 + 0.9 \times \max(7.0, 0, 0, 0) = -0.1 + 6.30 = 6.20$$

$$\delta_t = 6.20 - 0.0 = 6.20$$

$$Q((1, 1), \text{RIGHT}) \longleftarrow 0.0 + 0.5 \times (6.20) = \mathbf{3.10}$$

# Q-Learning Playground

## 🗺️ GridWorld Tabular Q-Learning

Model-Free TD Control

▶ Step 1 | ▶ Run 1 Ep | ▶▶ Train 50 Ep | ↺ Reset

### HYPERPARAMETERS

Learning Rate (α): 0.20

Discount Factor (γ): 0.90

Exploration (ε): 0.15

### EPISODE STATS

Episodes Completed
**0**

Total Steps Taken
**0**

Last Episode Return
**0.0**




### LIVE BELLMAN UPDATE

**Q-Learning Rule:**

$Q(s,a) \leftarrow Q + \alpha [r + \gamma \max Q(s',a') - Q]$

Waiting for step...

### LEARNING CURVE (RETURN / EP)

Run episodes to view return curve...

🟩 Goal: +10 | 🟥 Pit: -10 | Step cost: -0.1
Blue arrows indicate greedy policy π(s).

# Show me the code: Q-learning

```python
import gymnasium as gym
import numpy as np

env = gym.make("FrozenLake-v1", is_slippery=False)
Q = np.zeros((env.observation_space.n, env.action_space.n))

alpha, gamma, epsilon = 0.2, 0.95, 0.15
episodes = 500

for ep in range(episodes):
    state, _ = env.reset()
    done = False

    while not done:
        # Epsilon-greedy action selection
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])

        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated

        # Bellman TD error update
        td_target = reward + gamma * np.max(Q[next_state]) * (not done)
        td_error = td_target - Q[state, action]
        Q[state, action] += alpha * td_error

        state = next_state
```

# Where does this target come from???

In the multi-armed bandit, the target $R$ was intuitive:

$$\text{Target} = R \quad \longleftarrow \text{ a single sample of the expected payoff } \mathbb{E}[R \mid a]$$

In $Q$-learning, the target looks oddly specific:

$$\text{Target} = r + \gamma \max_{a'} Q(s', a')$$

**Why this combination?**

- Why the immediate reward $r$?
- Why discounted by $\gamma$?
- Why the maximum over next actions $\max_{a'} Q(s', a')$?

**Just like in bandits, this target is a single sample of an underlying expectation.**

# Cumulative Discounted Return

We want to maximise the **cumulative discounted return** over time:

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

where $\gamma \in [0, 1)$ ensures convergence for bounded reward sequences, see Appendix.

- $\gamma = 0$ (**Short-sighted**): Only cares about the very next reward ($R_{t+1}$).
- $\gamma \to 1$ (**Far-sighted**): Weighs future rewards almost as heavily as immediate ones.

Our goal is to find an **optimal policy** $\pi^*$ that maximises expected return from every state:

$$\pi^* \triangleq \arg\max_{\pi} \mathbb{E}[G_t \mid S_t = s] \quad \forall s \in \mathcal{S}$$

# Constructing the Expectation: $V^\pi$ and $Q^\pi$

**State-Value Function $V^\pi(s)$**

The expected return starting in state $s$ and following policy $\pi$ thereafter:

$$V^\pi(s) \triangleq \mathbb{E}\left[G_t \mid S_t = s\right]$$

**Action-Value Function $Q^\pi(s, a)$**

The expected return starting in state $s$, taking action $a$, and following $\pi$ thereafter:

$$Q^\pi(s, a) \triangleq \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right]$$

Using the environment's joint distribution $p(s', r \mid s, a)$, they are connected by:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) Q^\pi(s, a)$$

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \left[r + \gamma V^\pi(s')\right]$$

# The Bellman Equation

Richard Bellman (1957) formulated the recursive definition:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Substituting the recursive definition into $V^\pi(s)$ gives the **Bellman Equation**:

$$V^\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma V^\pi(S_{t+1}) \mid S_t = s\right]$$
$$Q^\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

> **Bootstrapping:** The value of where you are today equals the expected immediate reward, plus the discounted value of where you land next. See appendix for derivation.



# Bellman Optimality

Our goal is to find an **optimal policy** $\pi^*$ that maximises expected return:

$$V^*(s) \triangleq \max_\pi V^\pi(s) \quad \text{and} \quad Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a)$$

1. **Greedy choice now:** The optimal value of a state is achieved by taking the best available action:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$

2. **Optimal future:** Take action $a$ now, then act optimally from $S_{t+1}$ onward:

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a\right]$$

# Bellman Optimality Equation for $Q^*$

Substituting $V^*(S_{t+1}) = \max_{a'} Q^*(S_{t+1}, a')$ into identity (2) yields:

$$Q^*(s,a) = \mathbb{E}\left[ R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a \right]$$

In explicit transition probability form:

$$Q^*(s,a) = \sum_{s',r} p(s', r \mid s, a) \left[ r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

# From Expectation to Sample Target

Look at the right-hand side of the Bellman Optimality Equation:

$$\sum_{s',r} p(s', r \mid s, a) \left[ r + \gamma \max_{a'} Q^*(s', a') \right] = \mathbb{E}_{(S_{t+1}, R_{t+1}) \sim p(\cdot \mid s, a)} \left[ R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \right]$$

**This expectation** averages over all possible next states $s'$ and rewards $r$ weighted by $p(s', r \mid s, a)$.

- In real life, an agent does **not** know the transition dynamics $p(s', r \mid s, a)$ of the universe!

# From Expectation to Sample Target

- **So we sample:** The agent executes action $a$ in state $s$, and the environment returns **one single sample**:

$$(S_{t+1}, R_{t+1}) = (s', r) \sim p(s', r \mid s, a)$$

- Replacing the expectation with this single sample (and using current $Q$ for unknown $Q^*$):

$$\text{Sample Target} = r + \gamma \max_{a'} Q(s', a')$$

| Problem | Goal: True Expectation | Real-World Sample | Update Target |
|---|---|---|---|
| **Bandit** | $\mathbb{E}[R \mid a] = \sum_r r\, p(r \mid a)$ | $r \sim p(r \mid a)$ | $r$ |
| **Q-Learning** | $\mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid s, a\right]$ | $(s', r) \sim p(s', r \mid s, a)$ | $r + \gamma \max_{a'} Q(s', a')$ |

# Temporal Difference (TD) Learning

Updating an estimate towards a target formed by taking a **1-step sample** and **bootstrapping** from existing estimates is called **Temporal Difference (TD) Learning**.

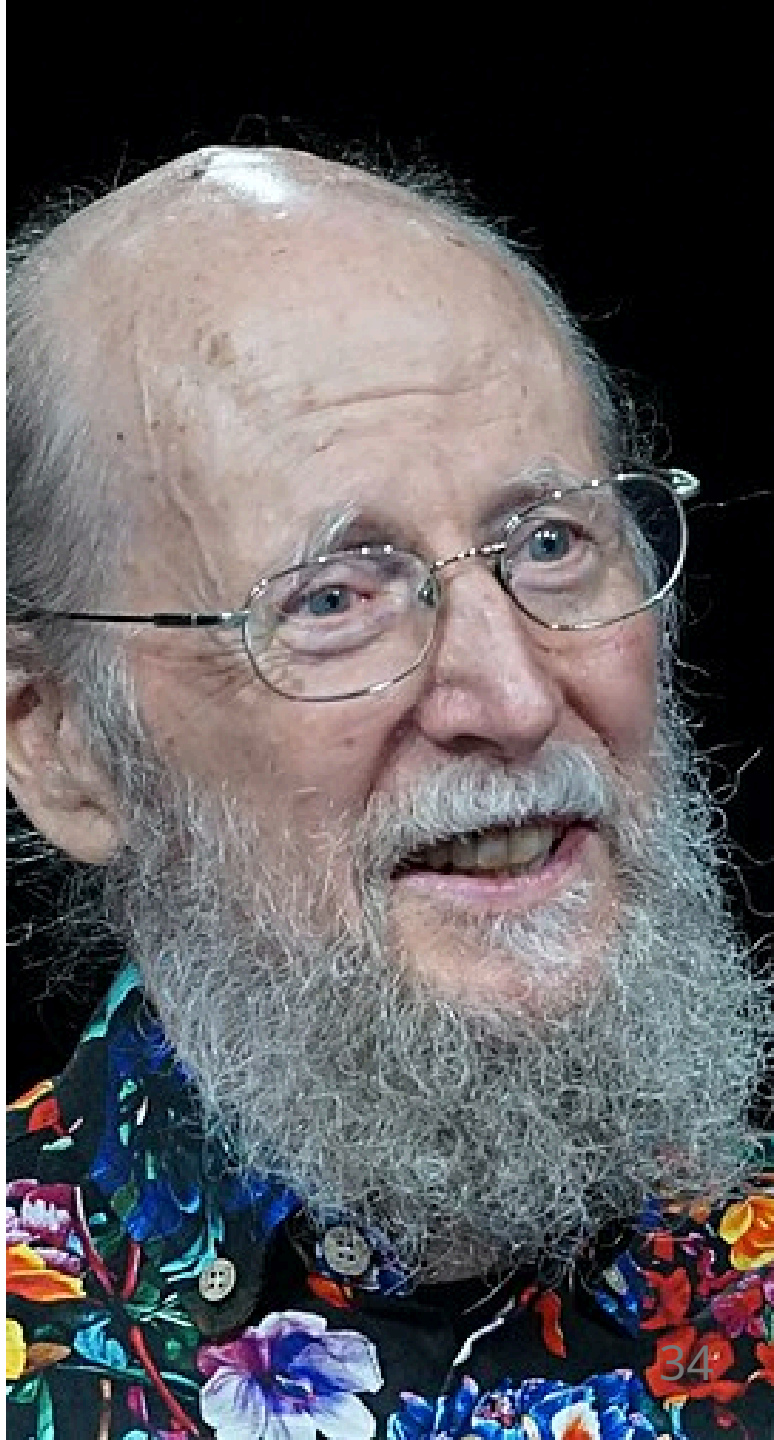
Formulated by Richard Sutton (1988), the update rule is driven by the discrepancy between expectation and reality:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha\,\delta_t$$

$$\delta_t \triangleq \underbrace{\left(r + \gamma \max_{a'} Q(s',a')\right)}_{\text{TD Target (Sample Reality)}} - \underbrace{Q(s,a)}_{\text{Current Expectation}}$$

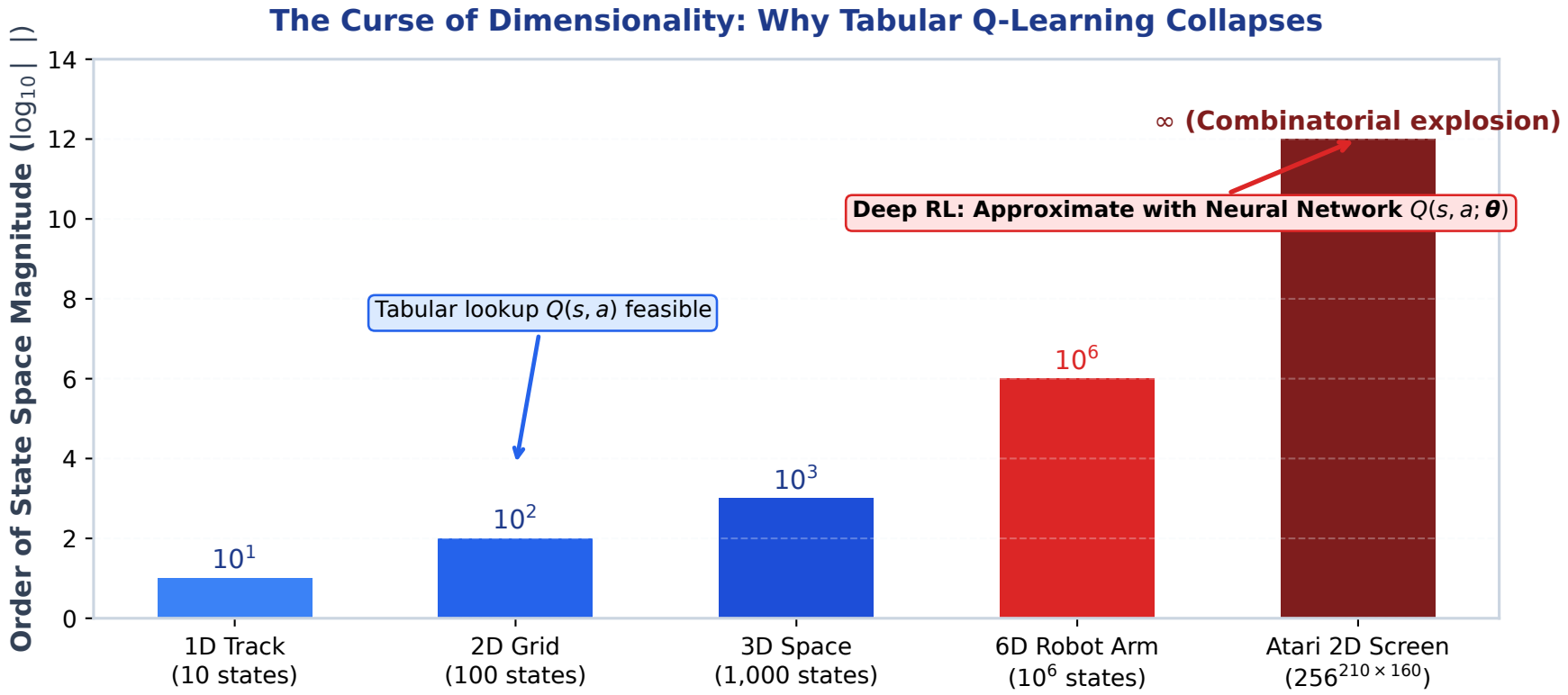
Here $\delta_t$ is the **Temporal Difference Error**:

$$\delta_t = \text{Outcome} - \text{Expectation}$$



# The Curse of Dimensionality

**So why can't we use tabular Q-learning to drive an autonomous car?**




- An Atari 2600 screen has $210 \times 160$ pixels with 256 colour values $\implies$ $256^{33,600}$ states!

# Bridge to Deep RL: Function Approximation

To scale beyond discrete grids, we replace the lookup table with a **differentiable parametric function**:

$$Q(s, a) \approx Q(s, a; \boldsymbol{\theta}) \quad (\text{a Deep Neural Network with weights } \boldsymbol{\theta})$$

Instead of cell updates, we optimise mean squared error via gradient descent:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^{-})}_{\text{Target}} - Q(s, a; \boldsymbol{\theta})\right)^2\right]$$

where the expectation is over transition tuples $(s, a, r, s')$ sampled from the replay buffer $\mathcal{D}$.

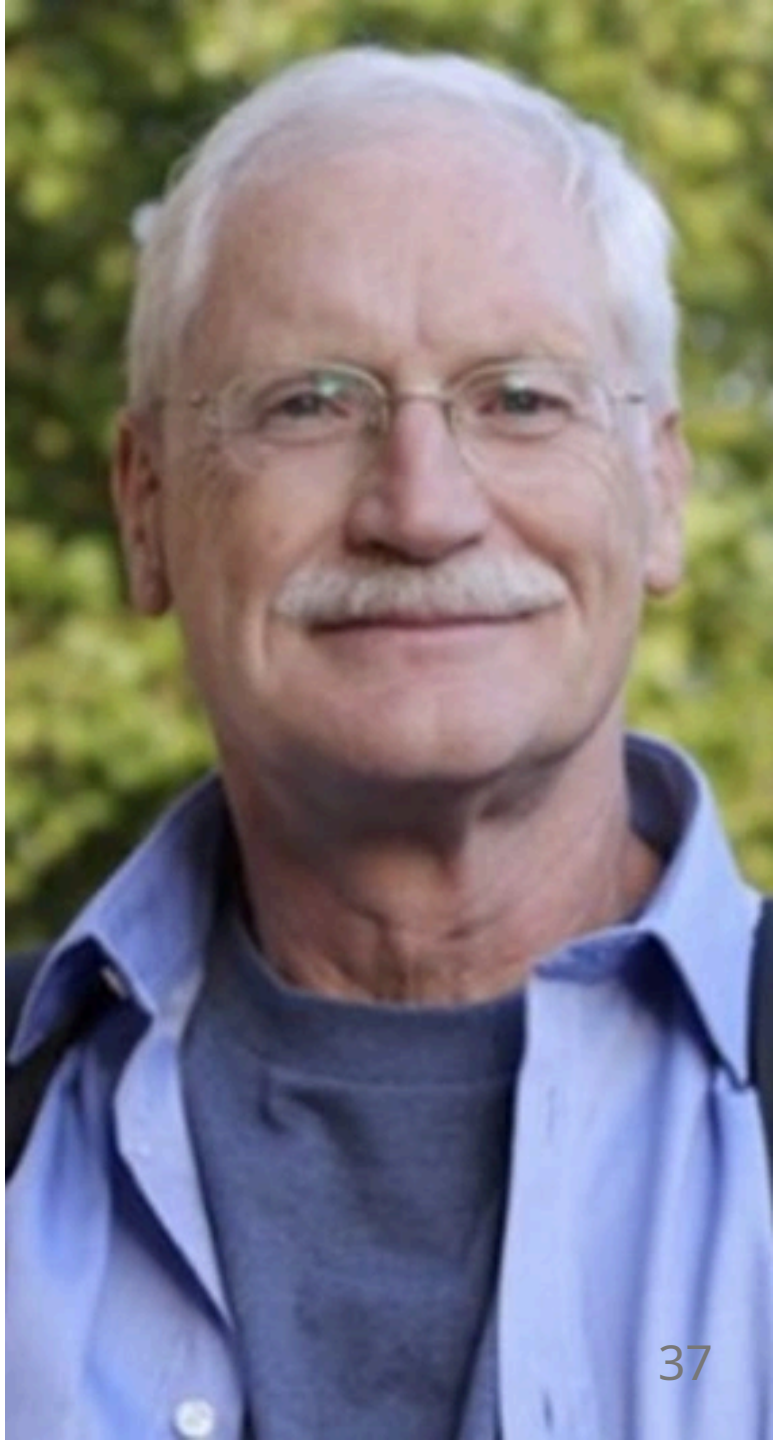
**DQN (Mnih et al., DeepMind 2015):** Played 49 Atari games directly from raw pixels.
**AlphaGo / MuZero (Silver et al., 2016–2020):** Mastered Go, Chess, and Shogi defeating human world champions through policy iteration + Monte Carlo Tree Search.

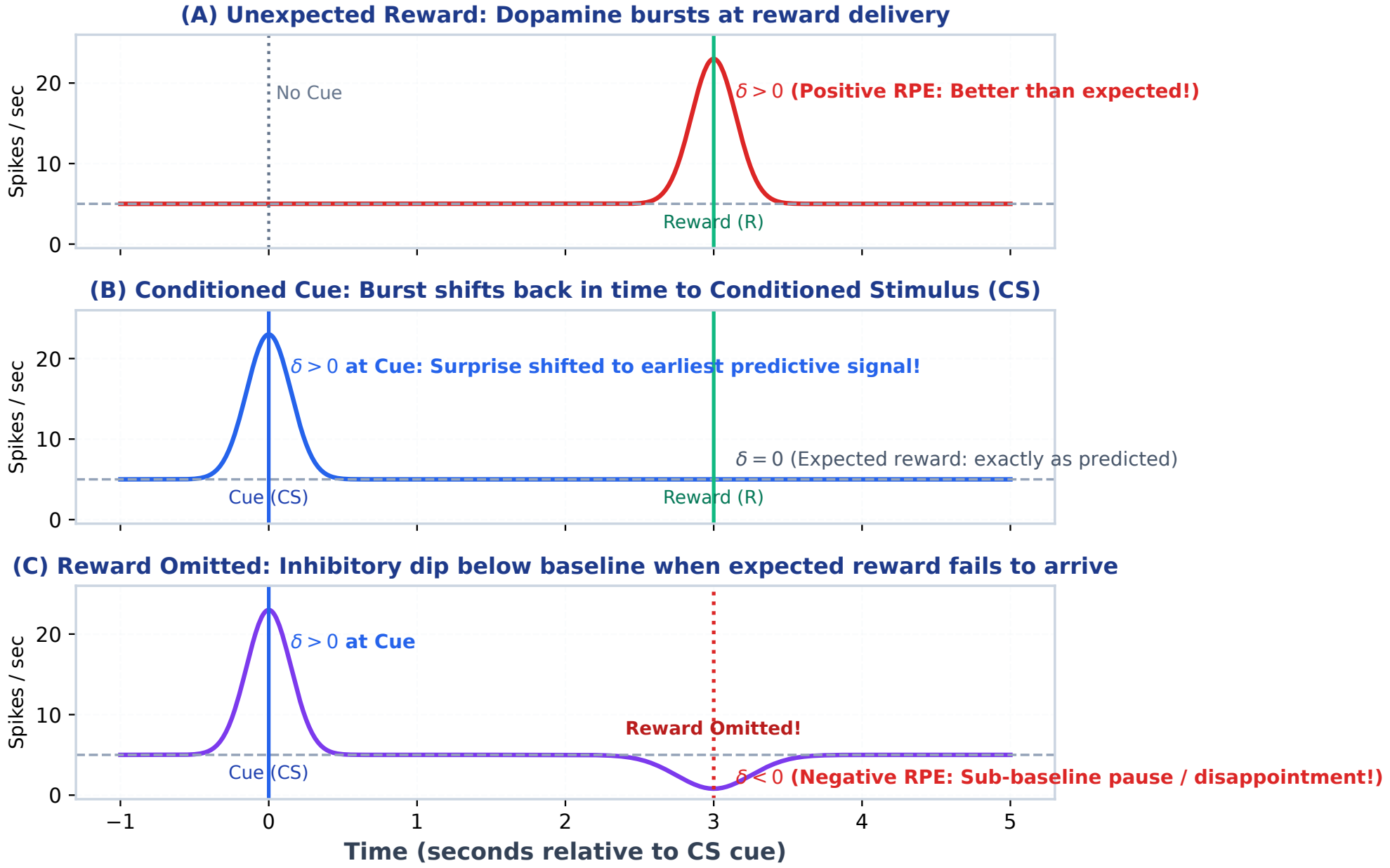
# Dopamine as Temporal Difference Error

In 1997, Wolfram Schultz, Peter Dayan, and Read Montague discovered that:

**Dopamine neurons do NOT just encode reward itself.**

**In many circuits, they encode the discrepancy between expected and actual reward $\rightarrow$ the TD Error!**




(A) Unexpected Reward: Dopamine bursts at reward delivery
Spikes / sec
20
10
0
No Cue
$\delta > 0$ (Positive RPE: Better than expected!)
Reward (R)
(B) Conditioned Cue: Burst shifts back in time to Conditioned Stimulus (CS)
Spikes / sec
20
10
0
$\delta > 0$ at Cue: Surprise shifted to earliest predictive signal!
$\delta = 0$ (Expected reward: exactly as predicted)
Cue (CS)
Reward (R)
(C) Reward Omitted: Inhibitory dip below baseline when expected reward fails to arrive
Spikes / sec
20
10
0
$\delta > 0$ at Cue
Reward Omitted!
Cue (CS)
$\delta < 0$ (Negative RPE: Sub-baseline pause / disappointment!)
−1
0
1
2
3
4
5
Time (seconds relative to CS cue)

# The Dopamine Response

Schultz recorded dopamine firing in macaque monkeys across three distinct regimes:

**1. Unexpected Reward ($\delta_t > 0$)**

- A juice drop is delivered without prior warning.
- $V(s_t) = 0, r_{t+1} = +1 \implies \delta_t = +1 > 0$. Remember, $\delta_t = \text{Outcome} - \text{Expectation}$.
- Dopamine neurons fire a burst of action potentials.

**2. Conditioned Cue Learned ($\delta_t = 0$ at Reward)**

- A light turns on, 2 seconds later juice arrives.
- As the monkey learns, the positive burst **shifts backward in time** to the earliest predictive cue 🔦 matching the prediction.
- At the moment the juice arrives, expectation matches reality ( $r_{t+1} + \gamma V(s_{t+1}) = V(s_t) \implies \delta_t = 0$). Firing stays at baseline.

### 3. Reward Omitted ($\delta_t < 0$)

- The light turns on, but juice is withheld.
- $V(s_t) = 1, r_{t+1} = 0 \implies \delta_t = -1 < 0.$
- At the exact millisecond juice was anticipated, dopamine firing drops to an **inhibitory silence below baseline**. Utter disappointment...

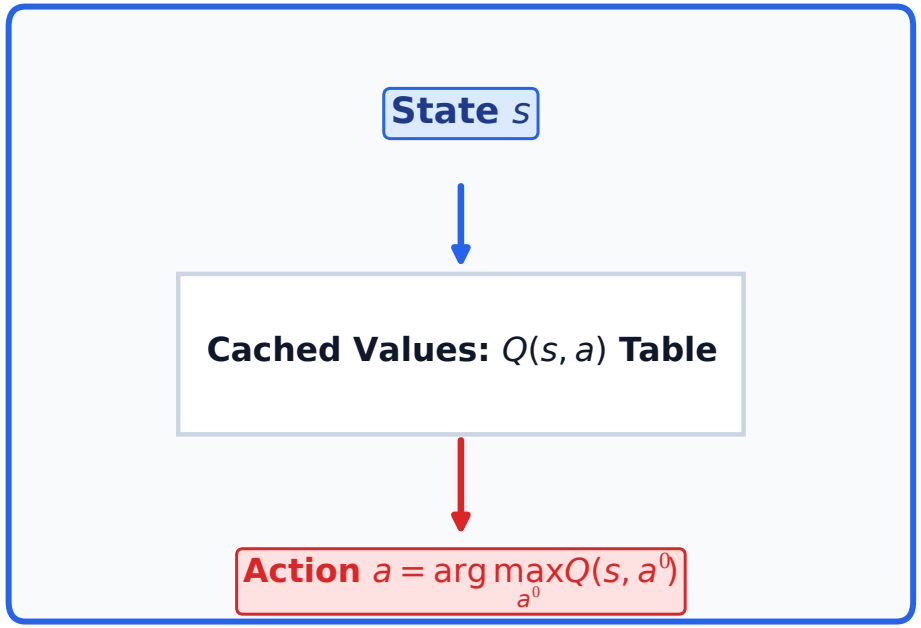


> Now imagine all the things we don't know about how our brain works and the amazing natural world 🌍

# Two Cognitive Modes in the Brain

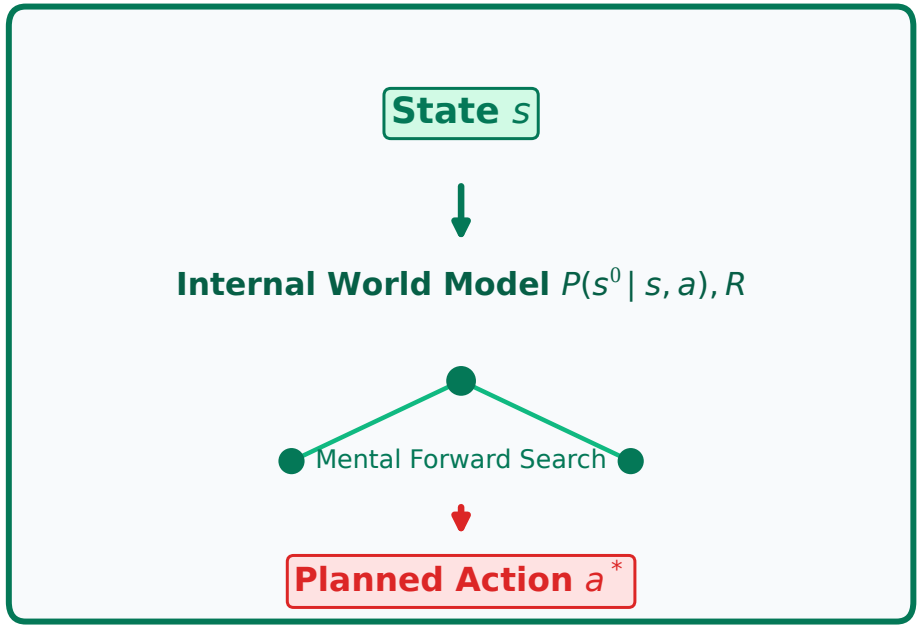
Neuroscience and artificial intelligence converge on two parallel control loops:

**Model-Free RL: Reflexive & Cached**
**(Dorsolateral Striatum / Basal Ganglia)**




- Computational Cost: $O(1)$ fast lookup
- Brittle: Slow to adapt to sudden reward devaluation

**Model-Based RL: Deliberative Planning**
**(Prefrontal Cortex & Hippocampus)**




- Computational Cost: High (tree traversal / rollouts)
- Flexible: Adapts immediately to environmental changes

# Questions

# Transition: The Limits of Lifetime Learning

Reinforcement Learning provides a powerful model of **lifetime learning (ontogeny)**.

However, consider how biological brains actually operate:

- A newborn gazelle can stand and run within 30 minutes of birth.
- It did not initialise its synaptic weights to random Gaussian noise $\mathcal{N}(0, 1)$ and run millions of backpropagation gradient steps!
- Its motor architecture, reflexes, and body plan were sculpted across millions of years by **Evolution**.

> *"What we learn during a lifetime is constrained and enabled by what natural selection optimised across eons."*

# Evolution & Derivative-Free Search

## Two Clocks of Intelligence: Phylogeny vs. Ontogeny

### PHYLOGENY (Evolutionary Optimisation)

Time Scale: Millions of Generations across Evolutionary History




↑ **Optimizes: Architecture, inductive biases, gene encoding, reward function** ↑

### ONTOGENY (Lifetime Plasticity & Learning)

Time Scale: Seconds to Decades within a Single Individual's Lifetime




↑ **Optimizes: Synaptic weights w, policy $\pi_\theta$, value functions** $Q(s, a)$ ↑

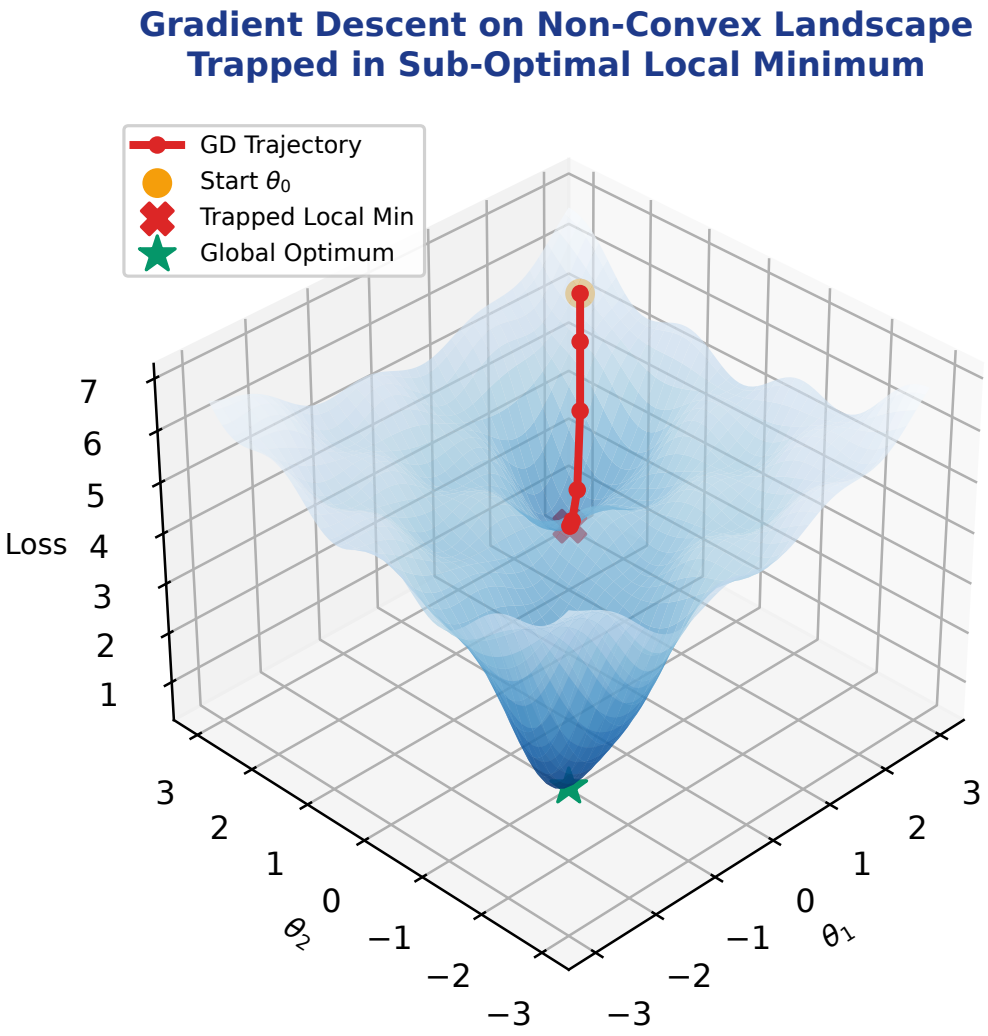
# The Non-Differentiable World

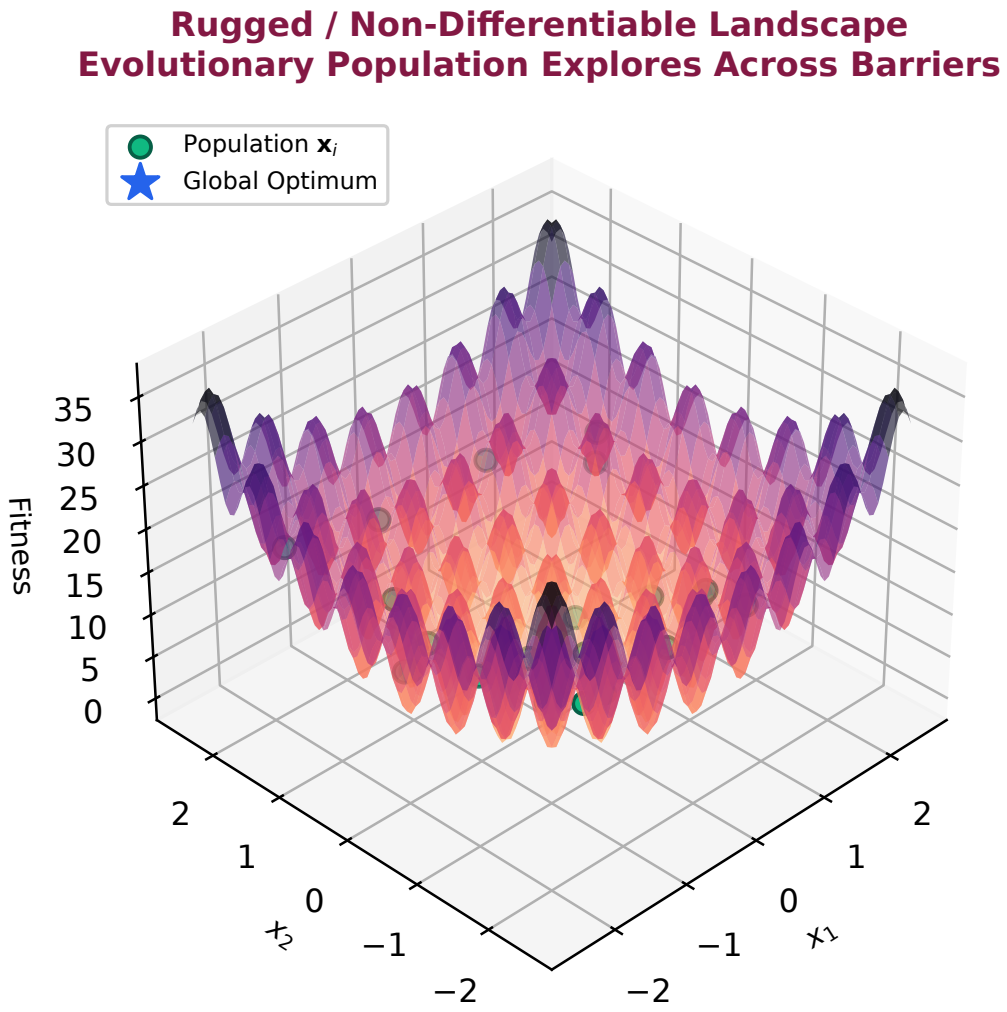
Backpropagation requires computing analytical gradients $\nabla_{\theta}\mathcal{L}$.

What happens when gradients do not exist?

1. **Non-Differentiable Functions:** Step functions, discrete hardware switches, branching code execution.
2. **Black-Box Environments:** Proprietary physics simulators, economic markets, wet-lab biological assays.
3. **Deceptive & Rugged Surfaces:** Surfaces riddled with sharp local traps where gradients point directly away from the global optimum.

# Gradient Descent vs. Evolution

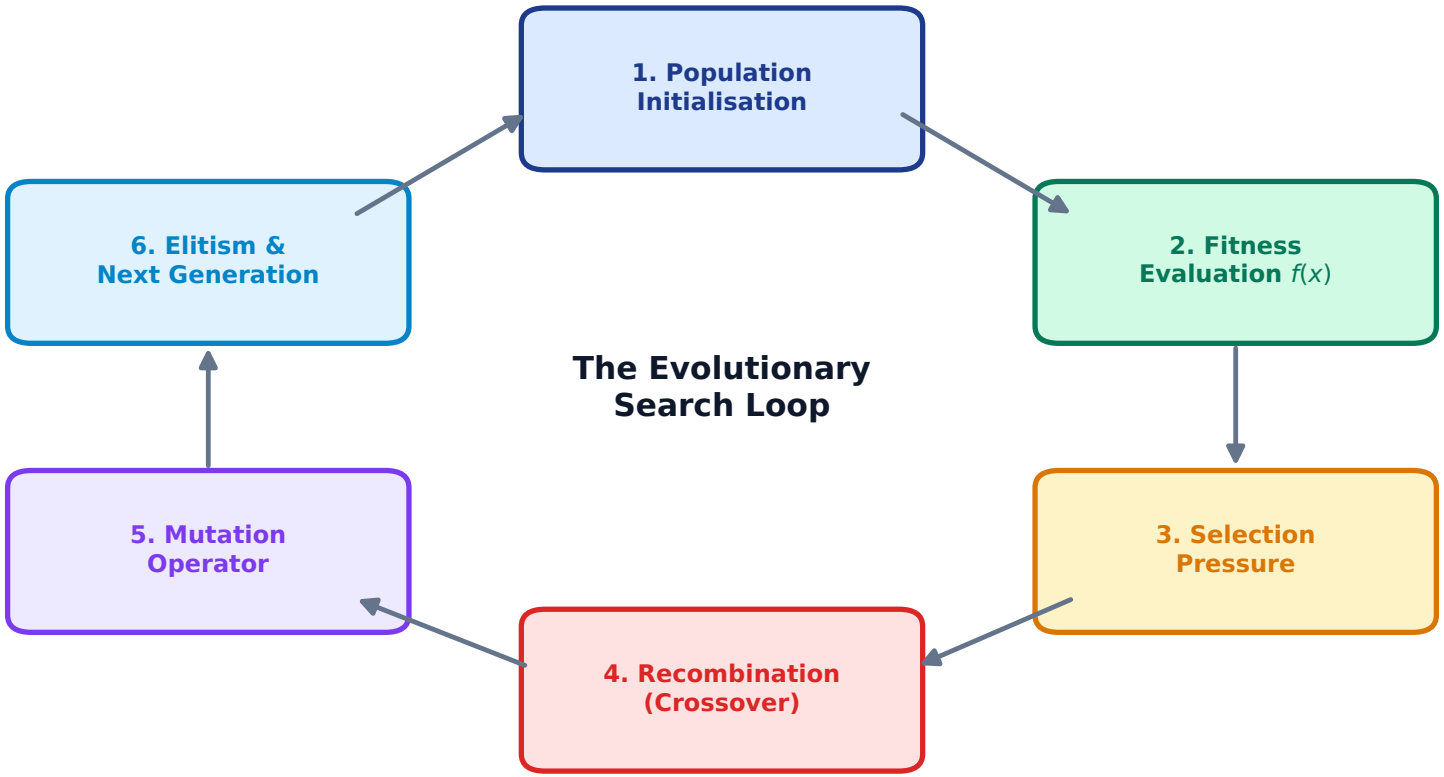


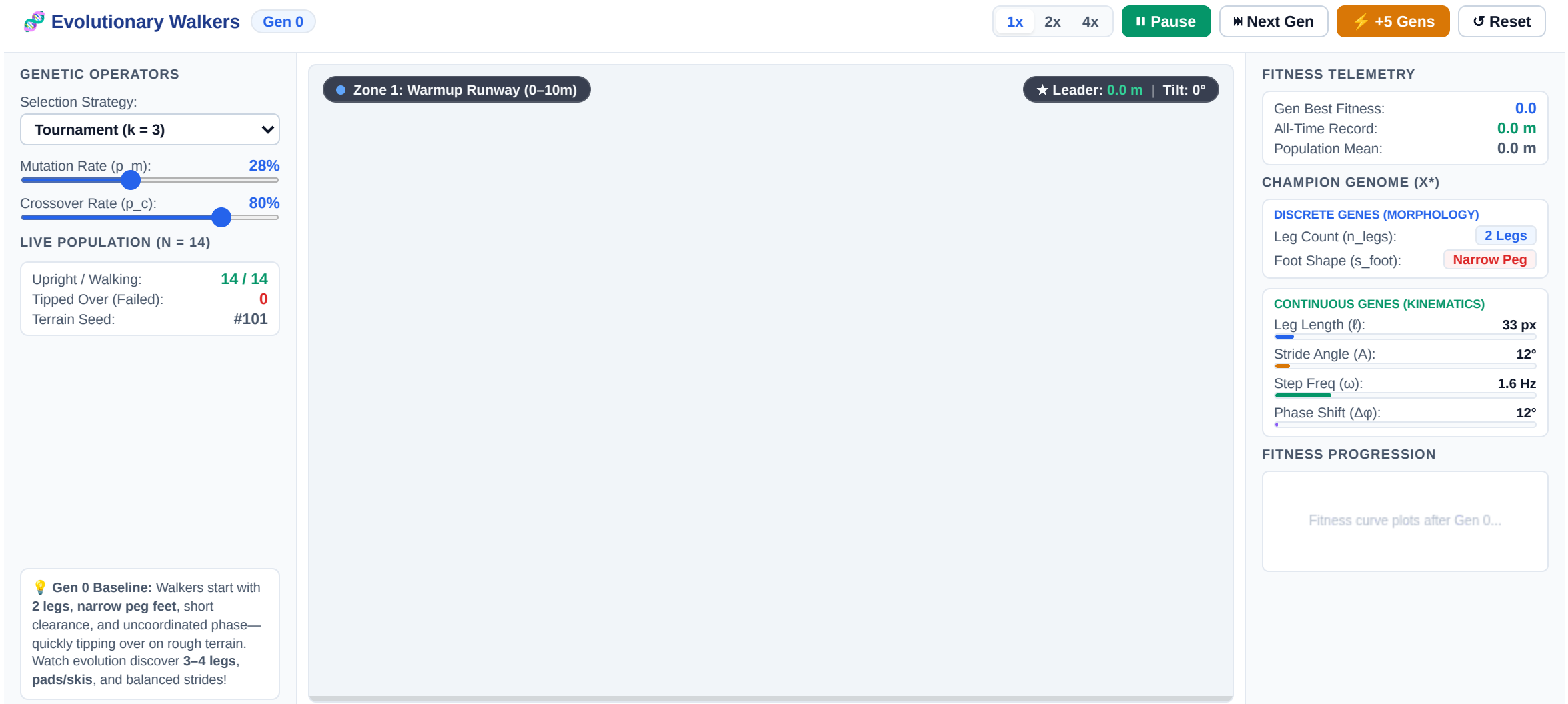
# Evolution

Evolutionary Algorithms treat the task as a **derivative-free black-box optimisation**:

$$\max_{\mathbf{x}} f(\mathbf{x}) \quad \text{where } f : \mathcal{X} \to \mathbb{R} \text{ is a fitness function}$$



# Interactive Evolutionary Walkers



# 1. Population

A population $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is a set of $N$ candidate solutions in search space $\mathcal{X}$.

**A walker:**
Each individual $\mathbf{x} = (n_{\text{legs}}, s_{\text{foot}}, \ell, A, \omega, \Delta\phi)$ encodes both morphology and gait:

- **Discrete** Genes (Body Morphology):
  - **Leg Count** $n_{\text{legs}} \in \{2, 3, 4\}$ number of two-jointed limbs attached to the chassis (starts at $2$).
  - **Foot Shape** $s_{\text{foot}} \in \{\text{Peg}, \text{Pad}, \text{Ski}\}$ ankle geometry (`Narrow Peg`, `Flat Pad`, or `Curved Ski`).
- **Continuous** Genes (Kinematics & Proportions):
  - **Leg Length** $\ell \in [30, 66]$ px limb ground clearance over uneven rocks.
  - **Stride Angle** $A \in [10^\circ, 40^\circ]$ & **Step Frequency** $\omega \in [1.0, 3.6]$ Hz.
  - **Phase Shift** $\Delta\phi \in [10^\circ, 180^\circ]$ inter-leg phase offset driving.

# 2. Fitness Function

The fitness function $f(\mathbf{x})$ is a **scalar-valued evaluation** of how well a candidate solution $\mathbf{x}$ performs in the environment:

**Evaluating the Walker $\mathbf{x}$:**

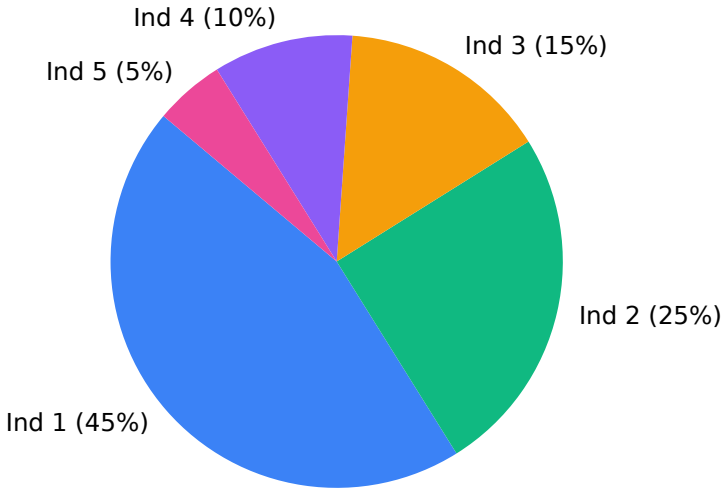
1. **Tip-Over Failure:** If torso tilt exceeds $|\theta_t| > \theta_{\max}$ ($36^\circ$) or the body crashes into the rocks, the walker tips over and immediately fails at tick $t_{\text{stop}}$.
2. **Scalar Fitness:** Rewards forward distance $d(\mathbf{x}) = \max(0, x_{\text{stop}} - x_0)$ and upright balance $b(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{t_{\text{stop}}}\left(1 - \frac{|\theta_t|}{\theta_{\max}}\right)$.

> What is the fitness function of nature?

# 3. Selection

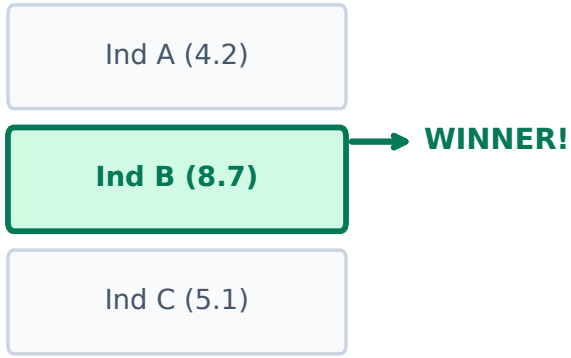
How do we decide which individuals reproduce?

**Fitness-Proportionate (Roulette Wheel)**



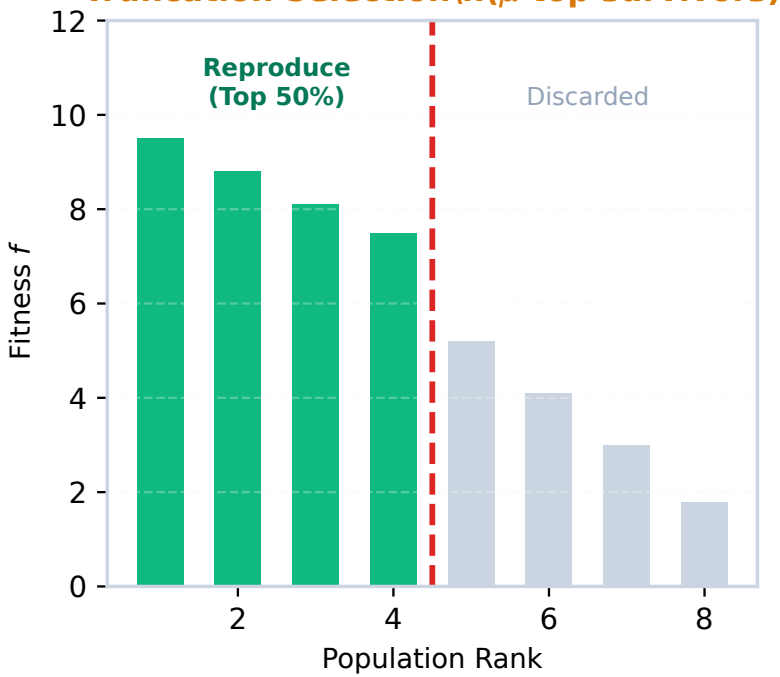

Prone to super-individual premature takeover

**Tournament Selection (Size $k = 3$)**




Robust against scale; Tunable pressure via $k$

**Truncation Selection\n($\mu$ top survivors)**




High selection pressure; Fast but loses diversity

**1. Fitness-Proportionate (Roulette Wheel)**

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

But, if one "super-individual" has $5\times$ higher fitness early on, it takes over the gene pool immediately $\rightarrow$ *local optima*.

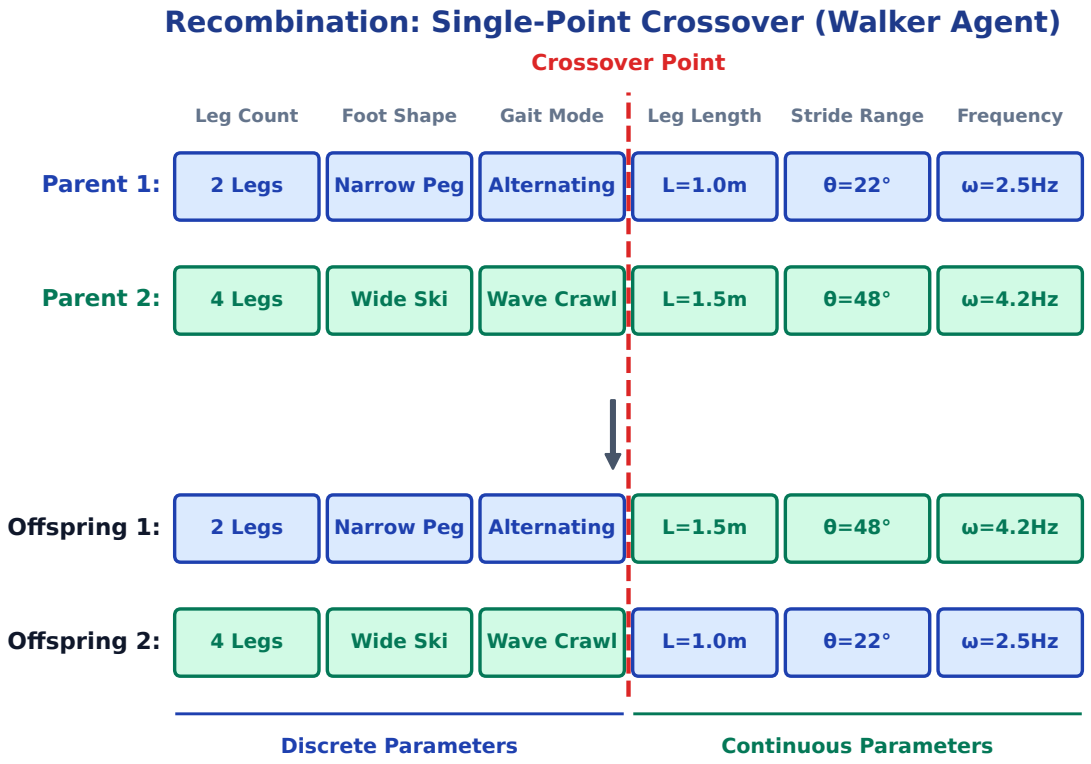
**2. Tournament Selection ($k$-Tournament)**

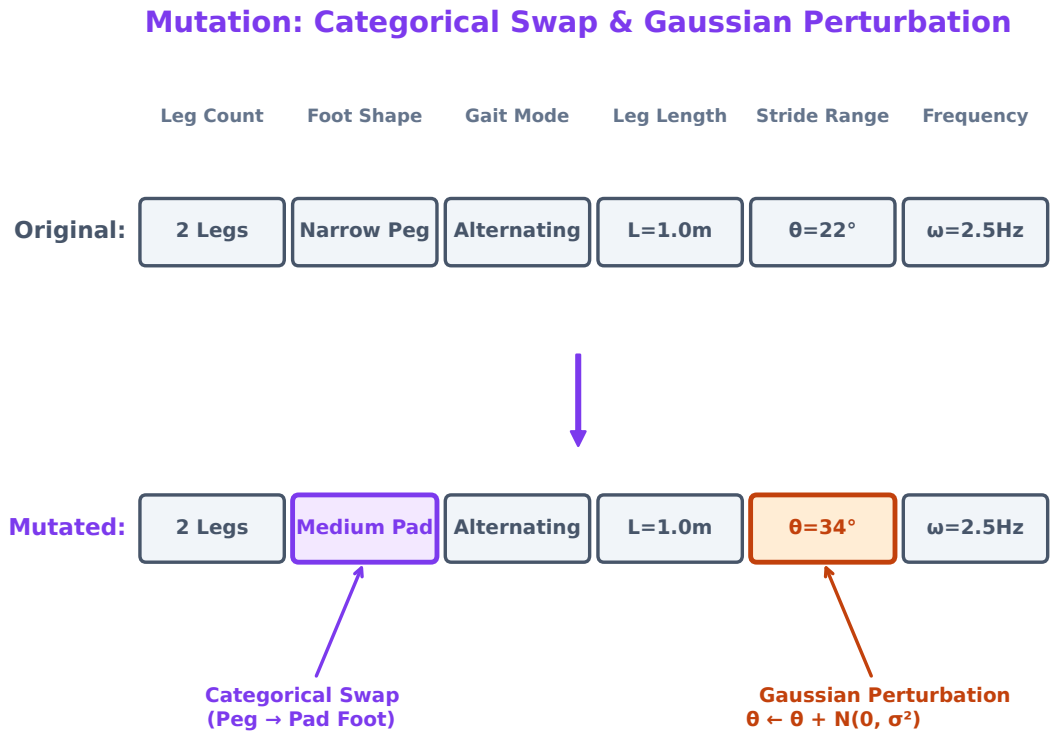
Sample $k$ random individuals; the highest-fitness among the $k$ wins reproduction rights.

**3. Truncation Selection**

Keep top $k$ percent of population and discard the rest.

# 4. Genetic Operators: Crossover & Mutation

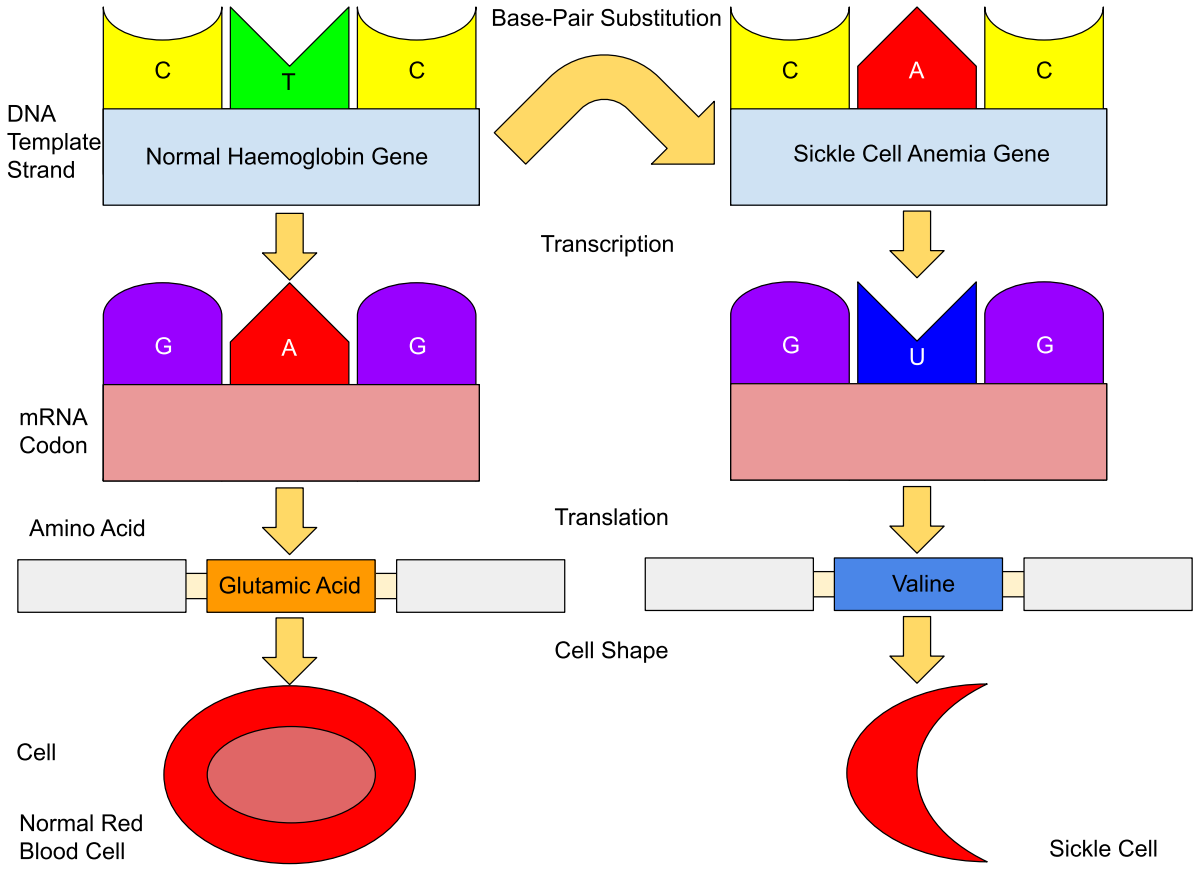



We also keep the best individual (solution) after each generation (**elitism**).

# The real evolution

In biological systems, genetic operators act directly on chromosomes and DNA sequences:



# Minimal Genetic Algorithm in Python

```python
import numpy as np

def fitness(x):
    # Inverted Rastrigin function (optimum at x = [0, 0])
    return -(10 * len(x) + np.sum(x**2 - 10 * np.cos(2 * np.pi * x)))

pop_size, d, gens = 50, 2, 60
pop = np.random.uniform(-3.0, 3.0, size=(pop_size, d))

for gen in range(gens):
    scores = np.array([fitness(ind) for ind in pop])
    best_idx = np.argmax(scores)

    # Elitism: carry over the best individual
    new_pop = [pop[best_idx].copy()]

    while len(new_pop) < pop_size:
        # Tournament selection (k = 3)
        cand_idx = np.random.choice(pop_size, size=3, replace=False)
        parent1 = pop[cand_idx[np.argmax(scores[cand_idx])]]
        cand_idx = np.random.choice(pop_size, size=3, replace=False)
        parent2 = pop[cand_idx[np.argmax(scores[cand_idx])]]

        # Arithmetic crossover
        alpha = np.random.rand()
        child = alpha * parent1 + (1 - alpha) * parent2

        # Gaussian mutation
        if np.random.rand() < 0.2:
            child += np.random.normal(0, 0.3, size=d)

        new_pop.append(child)
    pop = np.array(new_pop)
```

# Evolution Strategies (ES)

Instead of discrete chromosomes, **ES** optimise continuous real-valued parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ without gradients:

1. **Perturbation (Candidate Sampling):** Sample a population of $N$ mutant vectors by adding isotropic Gaussian noise to the current parameter estimate $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_i = \boldsymbol{\theta} + \sigma \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad i = 1, \ldots, N$$

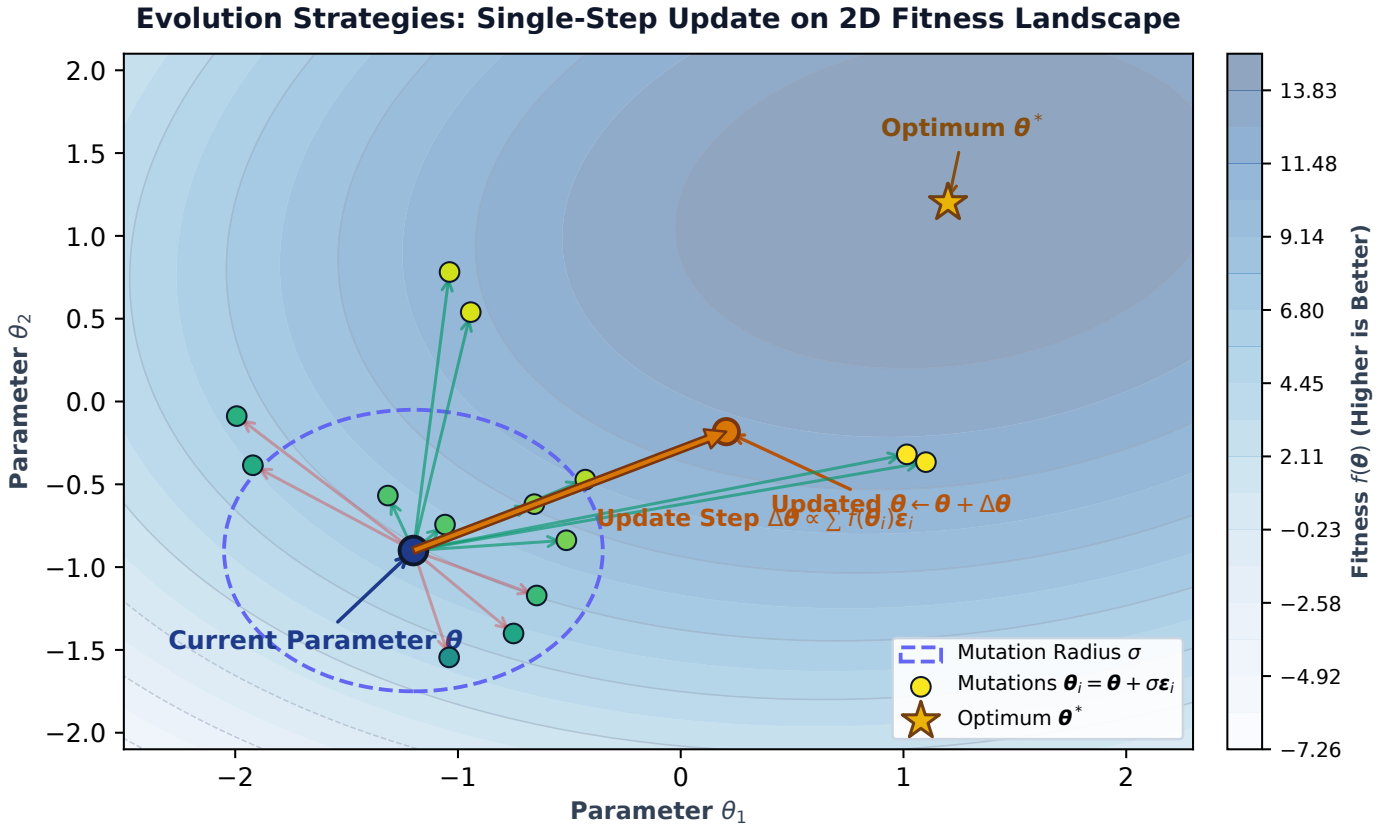
   where $\sigma > 0$ is the mutation step size (search radius).

2. **Evaluation:** Evaluate fitness $f(\boldsymbol{\theta}_i)$ for each candidate in the environment.

3. **Recombine:** Update $\boldsymbol{\theta}$ in proportion to candidate fitness:

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \alpha \frac{1}{N\sigma} \sum_{i=1}^{N} f(\boldsymbol{\theta}_i)\boldsymbol{\epsilon}_i$$

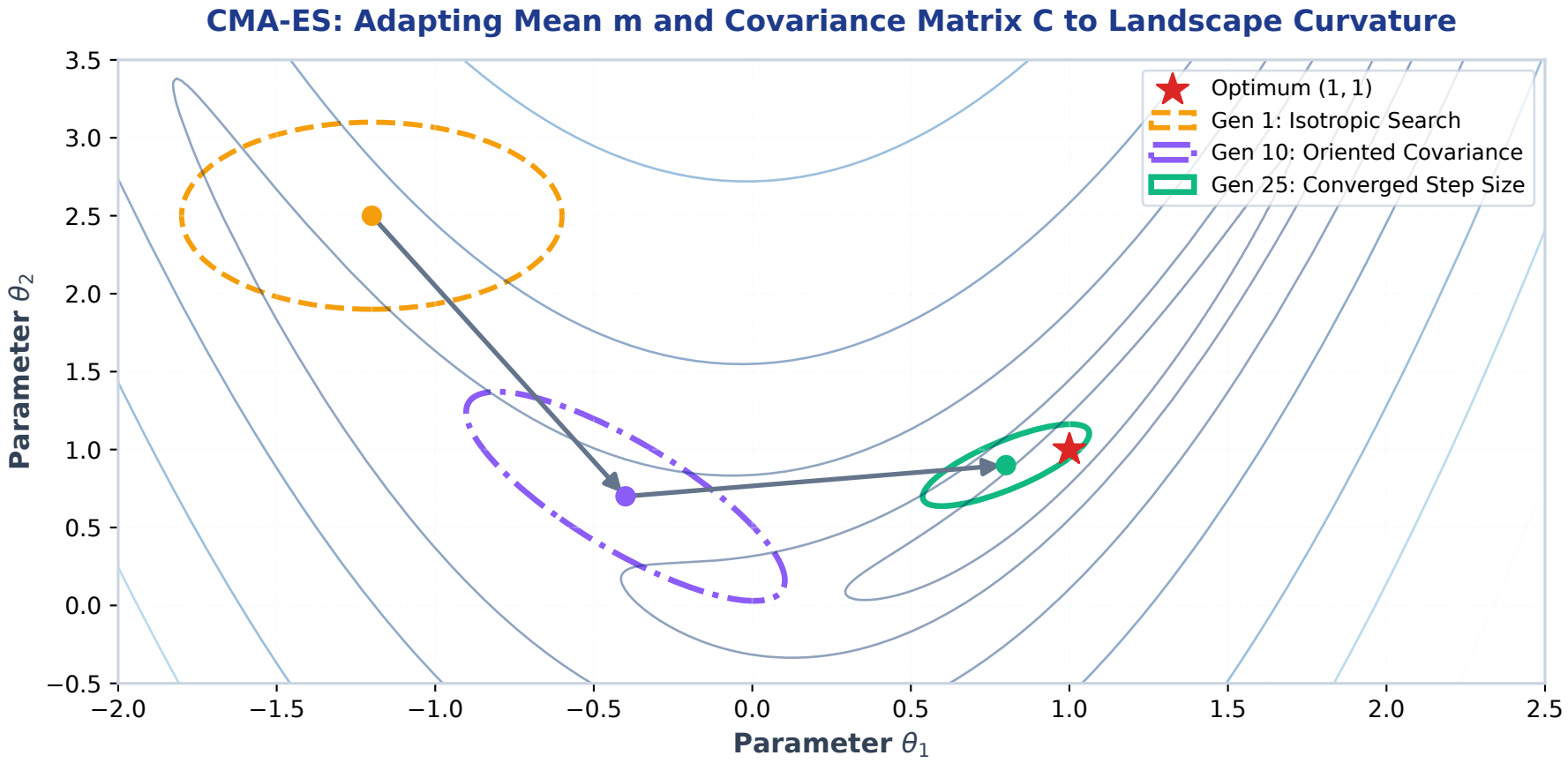
where $\alpha$ is the learning rate. Mutations yielding higher fitness pull $\boldsymbol{\theta}$ towards them.



# Covariance Matrix Adaptation (CMA-ES)

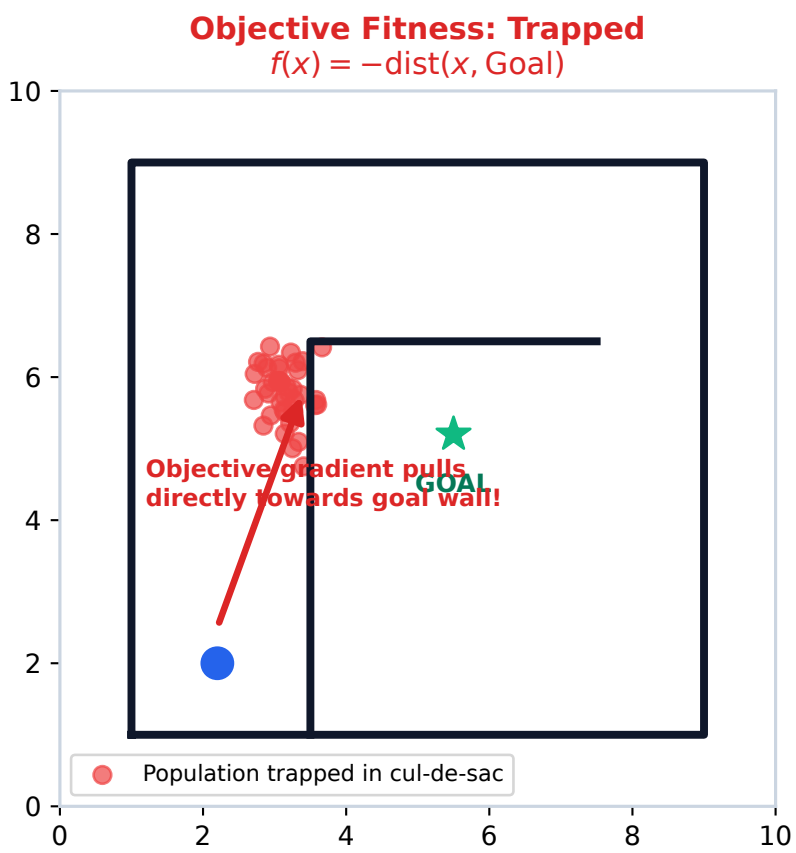
**CMA-ES** (Hansen, 2001) adapts both the mean $\mathbf{m}$ and covariance $\mathbf{C}$:

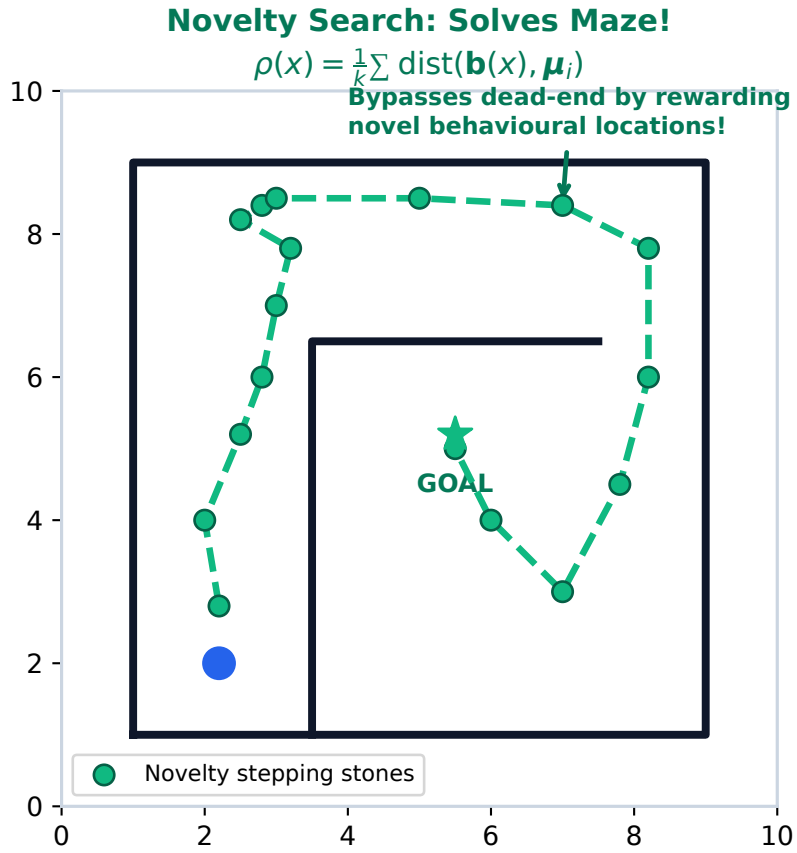
$$\mathbf{x} \sim \mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})$$



# The Problem with Objectives

So far, we define an objective function $f(\mathbf{x})$ and climb the gradient, but this can be deceptive:



# Novelty Search

Lehman & Stanley proposed an audacious paradigm: **Novelty Search**.

> *"To find the highest mountain, don't look up—explore the terrain."*

1. **Discard the Objective Function:** The algorithm ignores performance completely 🤔
2. **Define a Behavioural Characterisation $\mathbf{b}(\mathbf{x})$:** A descriptor of how the agent behaves (e.g. final $(x, y)$ coordinate reached in the maze).
3. **Reward Behavioural Sparseness $\rho(\mathbf{x})$:**

$$\rho(\mathbf{x}) \triangleq \frac{1}{k} \sum_{i=1}^{k} \|\mathbf{b}(\mathbf{x}) - \boldsymbol{\mu}_i\|$$

where $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ are the $k$-nearest neighbours in an archive of behaviours.

> Think of having a team of diverse people versus all VIM users.

# Deceptive Maze: A Toy Example

**The Robot & Genome:**
Starts at $\mathbf{s}_0 = (x_0, y_0)$ and executes a sequence of steering commands $\mathbf{x} = (\Delta\theta_1, \ldots, \Delta\theta_T)$.
Traces a continuous trajectory through the maze, stopping if colliding with walls.

**Behavioural Characterisation $\mathbf{b}(\mathbf{x})$:**
We define behaviour simply as the robot's **final landing coordinate**: $\mathbf{b}(\mathbf{x}) = (x_{\text{final}}, y_{\text{final}})$.
The genome is *how* it steered; the behaviour is *where* it ended up.

**Objective Fitness vs. Novelty Search:**

- **Objective ($f = -\|\mathbf{b} - \mathbf{x}_{\text{goal}}\|$):** Pulls the swarm directly against the wall closest to the goal, trapping all individuals in the cul-de-sac.
- **Novelty ($\rho = \frac{1}{k}\sum \|\mathbf{b} - \boldsymbol{\mu}_i\|$):** Once the dead end fills up, it is no longer novel. The swarm is pushed outward to explore unvisited corridors—bypassing the trap!

# Deceptive Maze & Novelty Search

🧭 **Lehman & Stanley Deceptive Maze** Objective vs Novelty Search

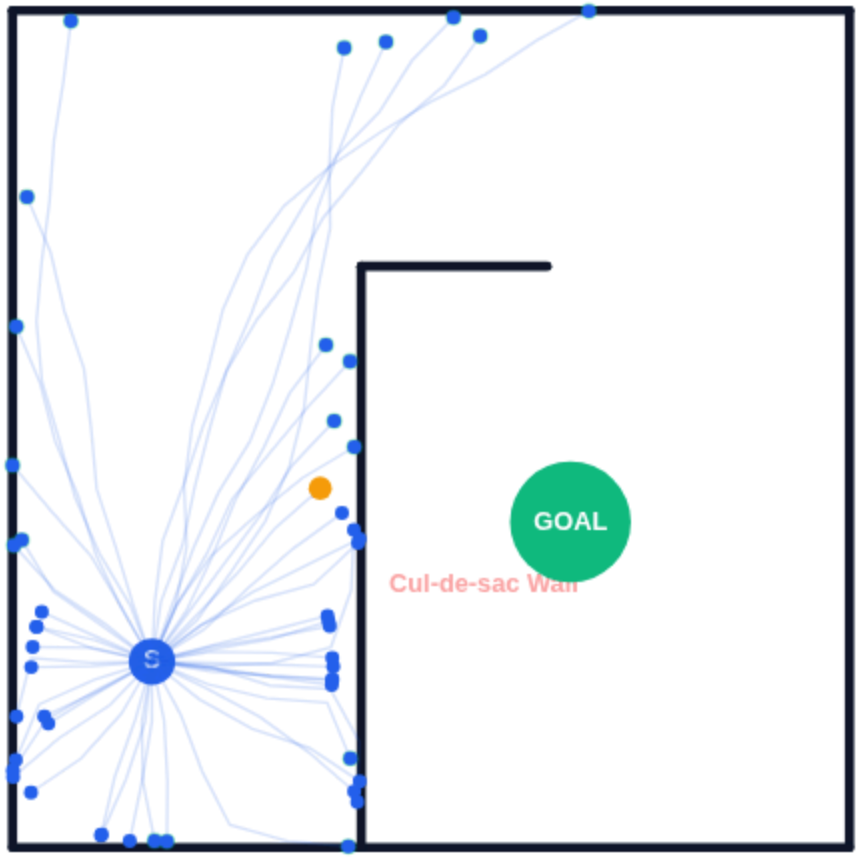
Objective Fitness | Novelty Search | ▶ Gen +1 | ▶▶ Run 50 Gens | ↺ Reset

### SEARCH CONFIGURATION

Archive k-NN (k): 5

Novelty Add Threshold: 22

**Novelty Search:** Ignoring the goal objective entirely! Agents are rewarded purely for reaching novel, unvisited coordinates.




### SEARCH STATUS

Generation
**0**

Goal Reached?
**NO**

Novelty Archive Size
**20**

Min Dist to Goal
**91 px**

🟢 Green dots: Novelty archive stepping stones
🔴 Red dots: Current population endpoints
⭐ Gold star: Maze exit goal

💡 Notice how **Objective Fitness** drives the swarm directly into the cul-de-sac wall because it looks geometrically closer to the goal.

# Minimal Novelty Search in Python

```python
import numpy as np

def simulate(genes, start=(1.0, 2.0)):
    x, y, heading = start[0], start[1], 0.0
    for steer in genes:
        heading += steer
        nx, ny = x + 0.8 * np.cos(heading), y + 0.8 * np.sin(heading)
        # Cul-de-sac wall blocking direct path (x in [3.8, 4.2], y < 6.0)
        if not (3.8 < nx < 4.2 and ny < 6.0) and (0 < nx < 10) and (0 < ny < 10):
            x, y = nx, ny
    return np.array([x, y])  # Behaviour b(x) = terminal landing coordinate

def novelty(b, archive, k=5):
    if len(archive) < k:
        return 1.0
    dists = np.linalg.norm(archive - b, axis=1)
    return np.mean(np.sort(dists)[:k])

pop_size, num_steps, gens = 40, 15, 30
pop = np.random.uniform(-0.5, 0.5, size=(pop_size, num_steps))
archive = []

for gen in range(gens):
    behaviors = np.array([simulate(ind) for ind in pop])
    scores = np.array([novelty(b, np.array(archive)) for b in behaviors])

    for b, s in zip(behaviors, scores):
        if s > 1.2 or len(archive) < 5:
            archive.append(b)

    new_pop = [pop[np.argmax(scores)].copy()]
    while len(new_pop) < pop_size:
        cands = np.random.choice(pop_size, size=3, replace=False)
        parent = pop[cands[np.argmax(scores[cands])]]
        child = parent + (np.random.rand(num_steps) < 0.25) * np.random.normal(0, 0.4, num_steps)
        new_pop.append(child)
    pop = np.array(new_pop)
```

# Quality-Diversity (QD) Algorithms

Can we combine the exploratory power of Novelty with high performance?

- **Objective Search:** Maximises fitness $f(\mathbf{x})$, but gets trapped in deceptive local optima.
- **Novelty Search:** Discovers diverse behaviours $\mathbf{b}(\mathbf{x})$, but ignores quality.

## Quality-Diversity

Instead of finding a single optimum, find an entire **repertoire of diverse, high-performing solutions**:

$$\text{For every behavioural niche } \mathbf{b} \in \mathcal{B}, \quad \text{maximise quality } f(\mathbf{x})$$

- **MAP-Elites (Mouret & Clune, 2015):** Offspring mutate and compete *locally within their behavioural niche* and each niche has its own *elite* individual.
- **Robotic Resilience (Cully et al., Nature 2015):** A 6-legged robot suffered a broken leg, queried its pre-computed MAP-Elites repertoire, and discovered an effective compensatory limp in $< 2$ minutes.

# Modern Neuroevolution

Can evolutionary algorithms compete with deep backpropagation on complex tasks?

## 1. OpenAI Evolution Strategies (Salimans et al., 2017)

- Evolved continuous neural network controllers for Atari 2600 and 3D humanoid locomotion.
- Evaluated hundreds of mutated policy vectors in parallel across **1,440 CPU** cores.
- Achieved linear speedup with zero backpropagation communication overhead.

## 2. Evolutionary Neural Architecture Search (NAS)

- Algorithms like NEAT (Stanley & Miikkulainen, 2002) and AmoebaNet (Real et al., Google 2019) evolve both network topology and layer connectivity.
- Discovered vision backbones that surpassed hand-designed human architectures.

# Where is my artificial brain?

I think we faired better this time. At least the numbers game was inspired by or connected to the real thing.

## AI → ML → Numbers game

---

**Next Part:** The ethics of it all, should we be doing this, and what are the consequences of our actions?

# Appendix

# Why $\gamma \in [0, 1)$?

**1. Mathematical Guarantee of Convergence**

In infinite-horizon tasks where rewards are bounded $|R_t| \leq R_{\max}$, return would diverge to $\infty$. With $\gamma < 1$, geometric series summation guarantees:

$$|G_t| \leq \sum_{k=0}^{\infty} \gamma^k R_{\max} = \frac{R_{\max}}{1-\gamma} < \infty$$

**2. The Agent's Effective Planning Horizon**

The effective temporal horizon over which rewards meaningfully matter is:

$$H_{\text{eff}} \approx \frac{1}{1-\gamma}$$

- $\gamma = 0.5 \implies H_{\text{eff}} \approx 2$ steps (myopic, immediate gratification).
- $\gamma = 0.9 \implies H_{\text{eff}} \approx 10$ steps.
- $\gamma = 0.99 \implies H_{\text{eff}} \approx 100$ steps (long-sighted strategic foresight).

# Deriving the Bellman Equation for $V^{\pi}(s)$

Starting from the definition of the state-value function under policy $\pi$:

$$
\begin{aligned}
V^{\pi}(s) &\triangleq \mathbb{E}_{\pi}\left[G_t \mid S_t = s\right] && \text{(Definition of } V^{\pi}(s)\text{)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \text{(Recursive definition of return } G_t\text{)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{\pi}\left[G_{t+1} \mid S_t = s\right] && \text{(Linearity of expectation)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{\pi}\left[\mathbb{E}_{\pi}\left[G_{t+1} \mid S_t = s, S_{t+1}\right] \mid S_t = s\right] && \text{(Law of total expectation / Tower property)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{\pi}\left[\mathbb{E}_{\pi}\left[G_{t+1} \mid S_{t+1}\right] \mid S_t = s\right] && \text{(Markov property: } G_{t+1} \perp S_t \mid S_{t+1}\text{)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{\pi}\left[V^{\pi}(S_{t+1}) \mid S_t = s\right] && \text{(Definition of } V^{\pi}(S_{t+1})\text{)} \\
&= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma V^{\pi}(S_{t+1}) \mid S_t = s\right] && \text{(Linearity of expectation)}
\end{aligned}
$$

> This is normally how it is written. You'll see this in RL courses or uglier versions of it.

# Deriving the Bellman Equation for $V^\pi(s)$

Using explicit trajectory notation $\tau_{t:\infty} \triangleq (A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, \ldots) \sim (\pi, p)$:

$$
\begin{aligned}
V^\pi(s) &\triangleq \mathbb{E}_{\tau_{t:\infty}\sim(\pi,p)}\left[G_t \mid S_t = s\right] && \text{(Definition of } V^\pi(s)\text{)} \\
&= \mathbb{E}_{\tau_{t:\infty}\sim(\pi,p)}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \text{(Recursive definition of return } G_t\text{)} \\
&= \mathbb{E}_{A_t\sim\pi,\, R_{t+1}\sim p}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{\tau_{t:\infty}\sim(\pi,p)}\left[G_{t+1} \mid S_t = s\right] && \text{(Linearity of expectation)} \\
&= \mathbb{E}_{A_t\sim\pi,\, R_{t+1}\sim p}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{A_t\sim\pi,\, S_{t+1}\sim p}\left[\mathbb{E}_{\tau_{t+1:\infty}\sim(\pi,p)}\left[G_{t+1} \mid S_t = s, A_t, S_{t+1}\right] \,\middle|\, S_t = s\right] && \text{(Law of total expectation)} \\
&= \mathbb{E}_{A_t\sim\pi,\, R_{t+1}\sim p}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{A_t\sim\pi,\, S_{t+1}\sim p}\left[\mathbb{E}_{\tau_{t+1:\infty}\sim(\pi,p)}\left[G_{t+1} \mid S_{t+1}\right] \,\middle|\, S_t = s\right] && \text{(Markov property: } G_{t+1} \perp S_t, A_t \mid S_{t+1}\text{)} \\
&= \mathbb{E}_{A_t\sim\pi,\, R_{t+1}\sim p}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}_{A_t\sim\pi,\, S_{t+1}\sim p}\left[V^\pi(S_{t+1}) \mid S_t = s\right] && \text{(Definition of } V^\pi(S_{t+1})\text{)} \\
&= \mathbb{E}_{A_t\sim\pi(\cdot|S_t),\, (S_{t+1},R_{t+1})\sim p(\cdot,\cdot|S_t,A_t)}\left[R_{t+1} + \gamma V^\pi(S_{t+1}) \mid S_t = s\right] && \text{(Linearity of expectation)}
\end{aligned}
$$