Reinforcement Learning

Section 1: The Basics

A bit about me

Online Forum

- Google group for this course:

  spbau-rl@googlegroups.com
- Need to request access.
- Used for:
  - Announcements and posting exercises
  - Q&A
  - Anything else (comments, suggestions, etc)

Why study RL?

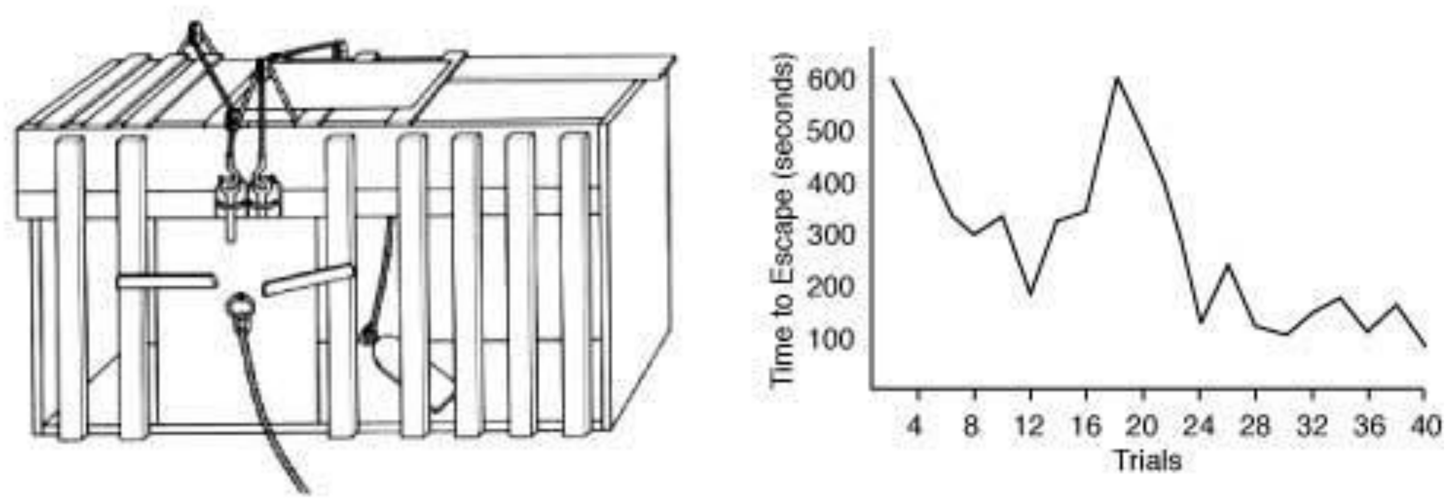
Motivation

- The psychology/neuroscience perspective.
- The engineering perspective.

1. The Psychology of RL

1911: Thorndike

Thorndike's puzzle box

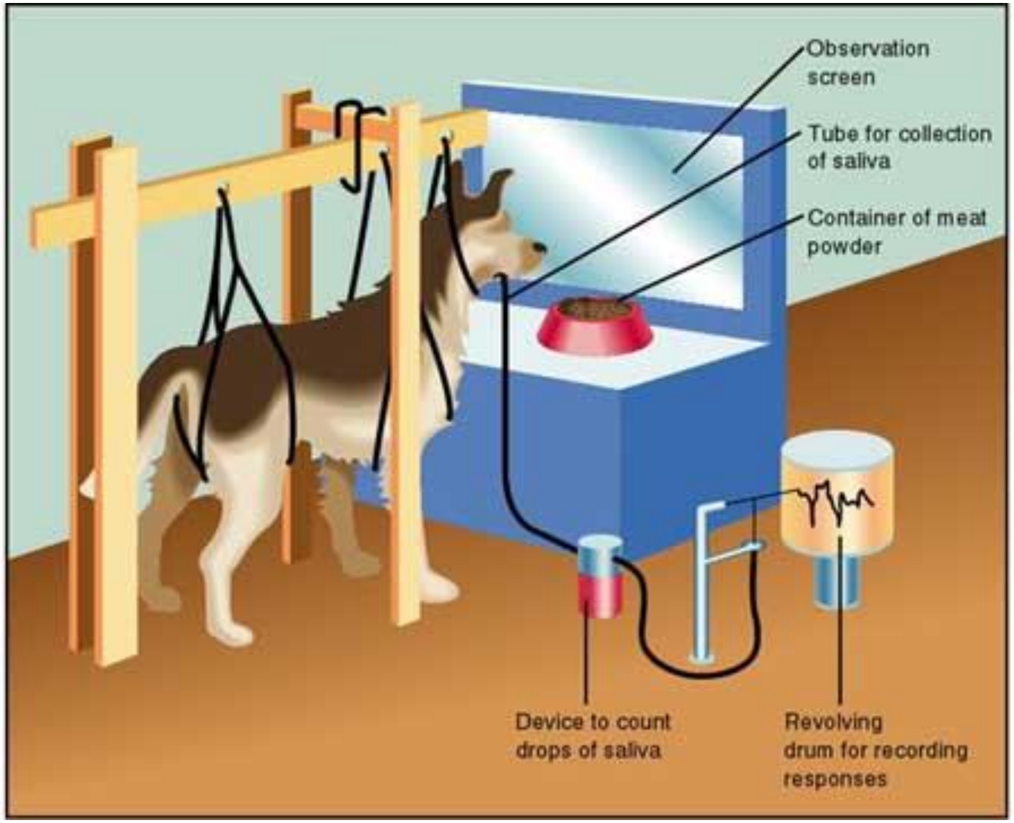


Adapted from Domjan, 1993 (modified from Thorndike, 1898 [left] and Imada & Imada, 1983 [right])

Law of effect: "Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation."

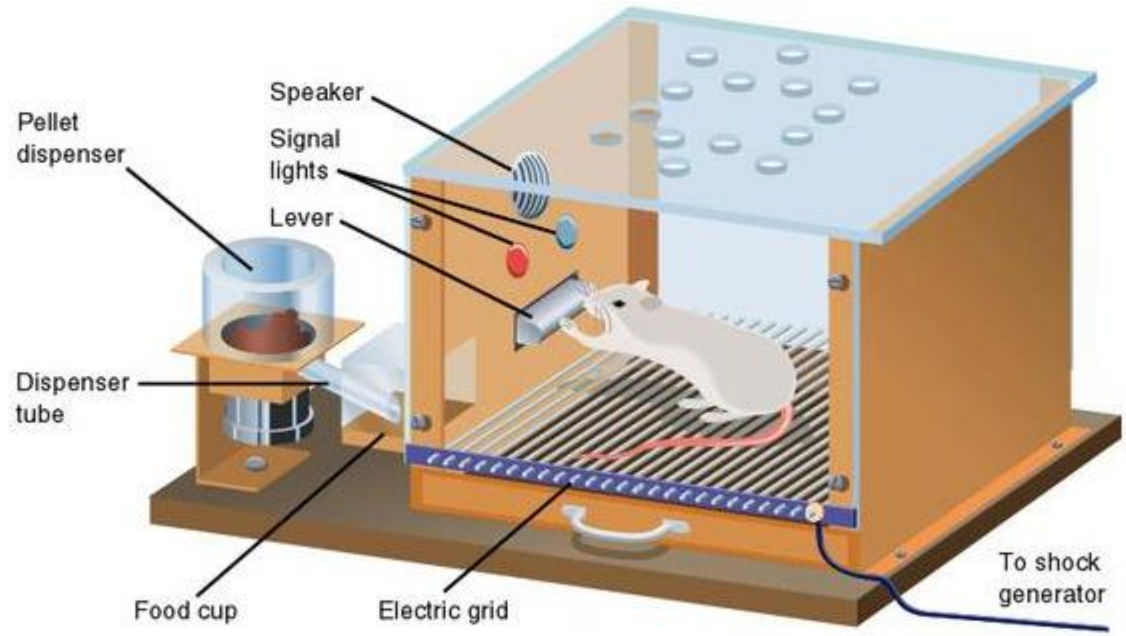
1927: Pavlov

Pavlov's dog



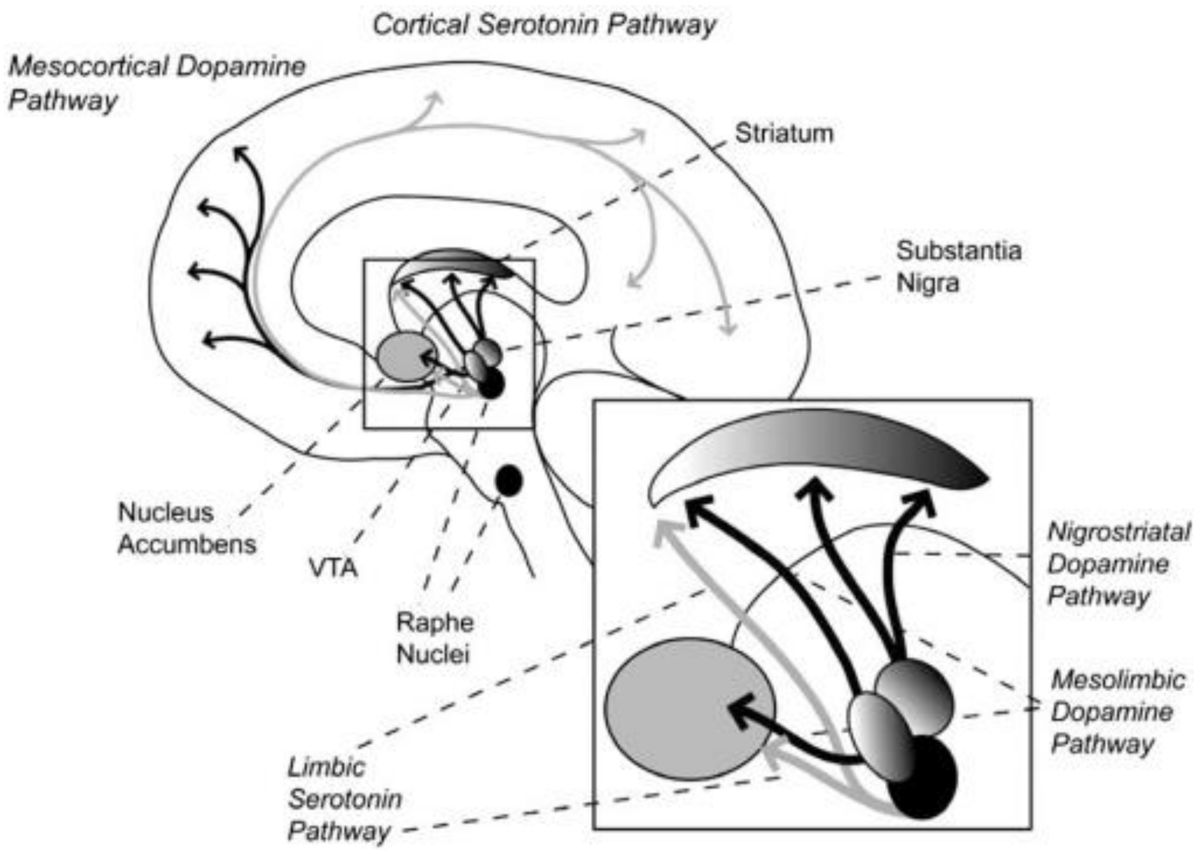
1948: Skinner

Skinner Box



- Positive reinforcement, negative reinforcement, punishment
- Observation of response and extinction rates.
- Psychotherapy: token economy, behaviour shaping

1970s: Dopamine



Pavlov Revisited

2. The Engineering of RL

Reinforcement Learning

Application Example 1: Breakout

Reinforcement Learning

Application Example 1: Breakout

Reinforcement Learning

Application Example 2: Helicopter control

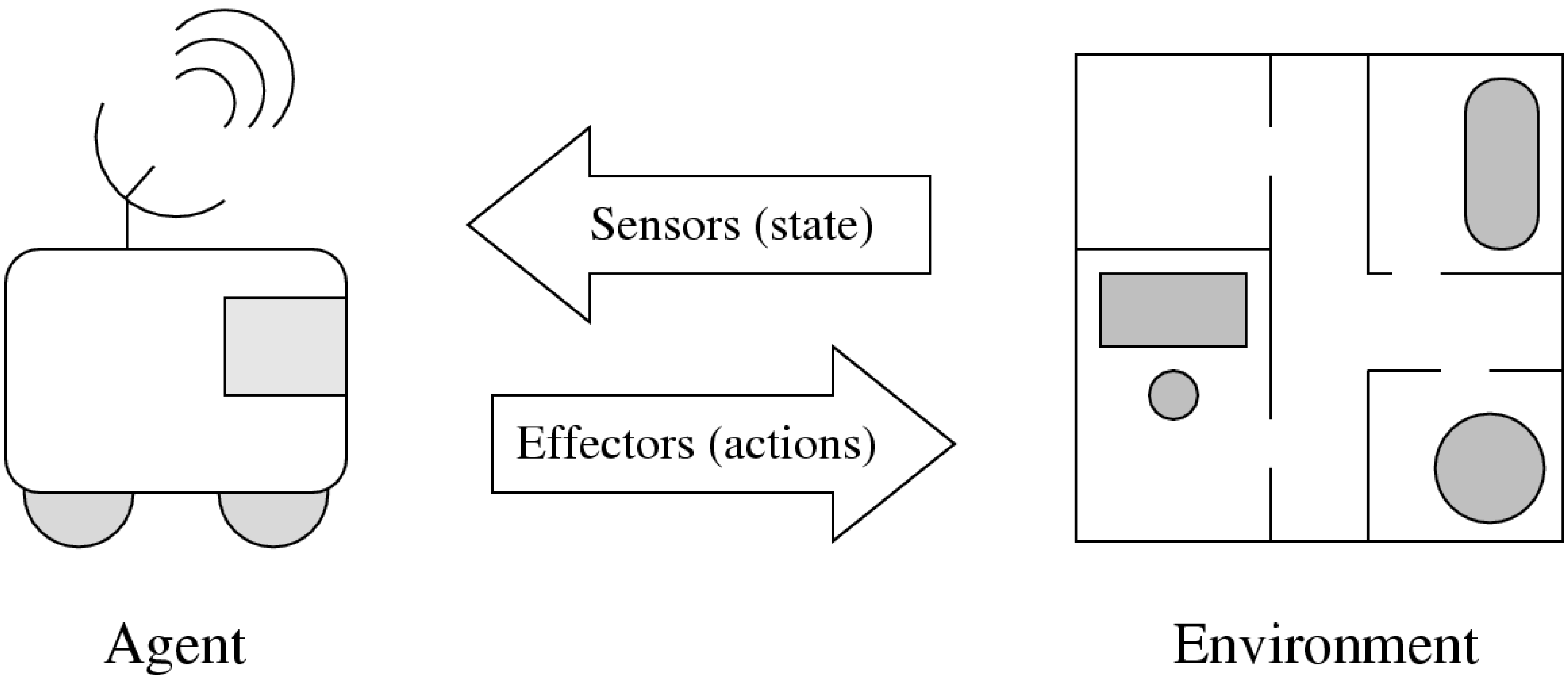
Real-world RL Problems

Beyond Toy Problems: sequential decision making

- Industrial plant control
- Investment portfolio management
- Robot movement
- Autonomous car driving

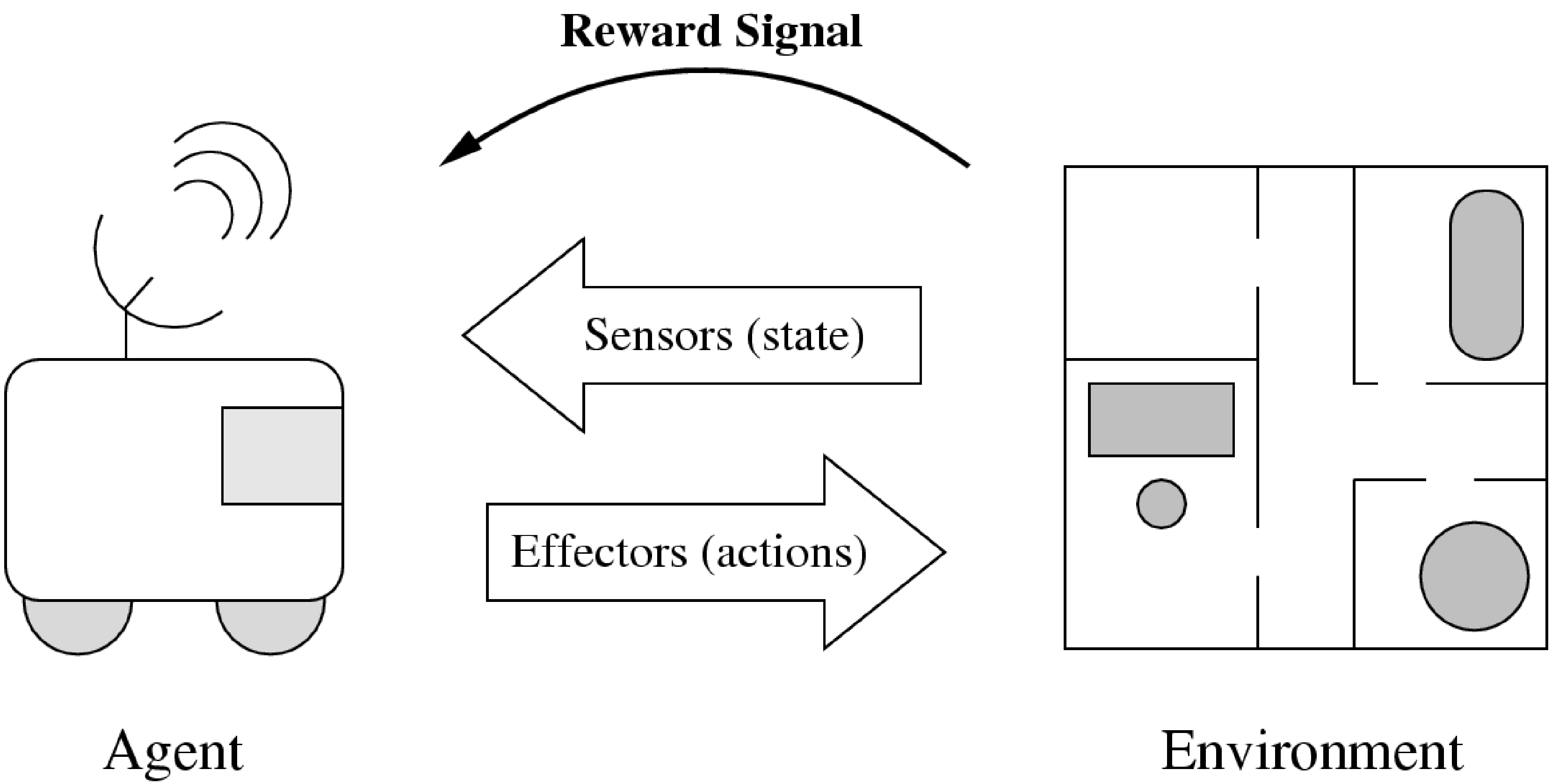
Reinforcement Learning

Agent Paradigm



Reinforcement Learning

Agent Paradigm



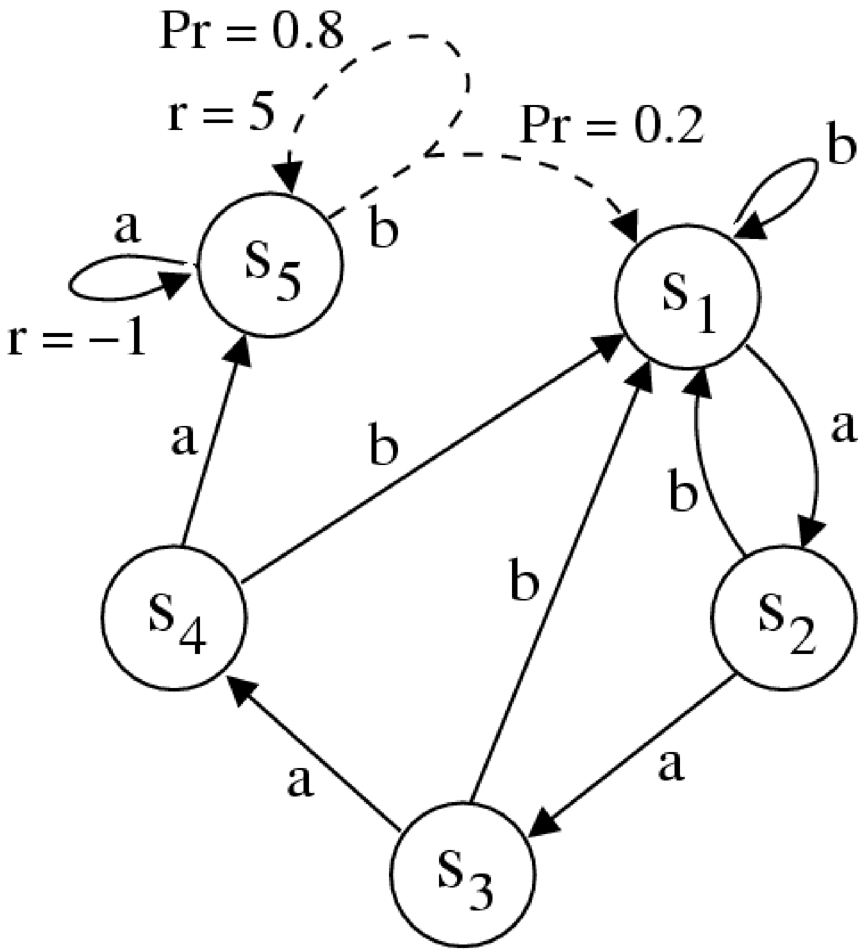
RL vs. Supervised Learning

What makes RL different?

- No supervisor feedback: learning from reward signal only.
- Feedback signal is often delayed, not instantaneous.
- Sequential aspect of decision making.
- Agent is influencing the selection of training experience.

Modelling the Environment

Markov Decision Process



Markov Decision Process (MDP)

Definition

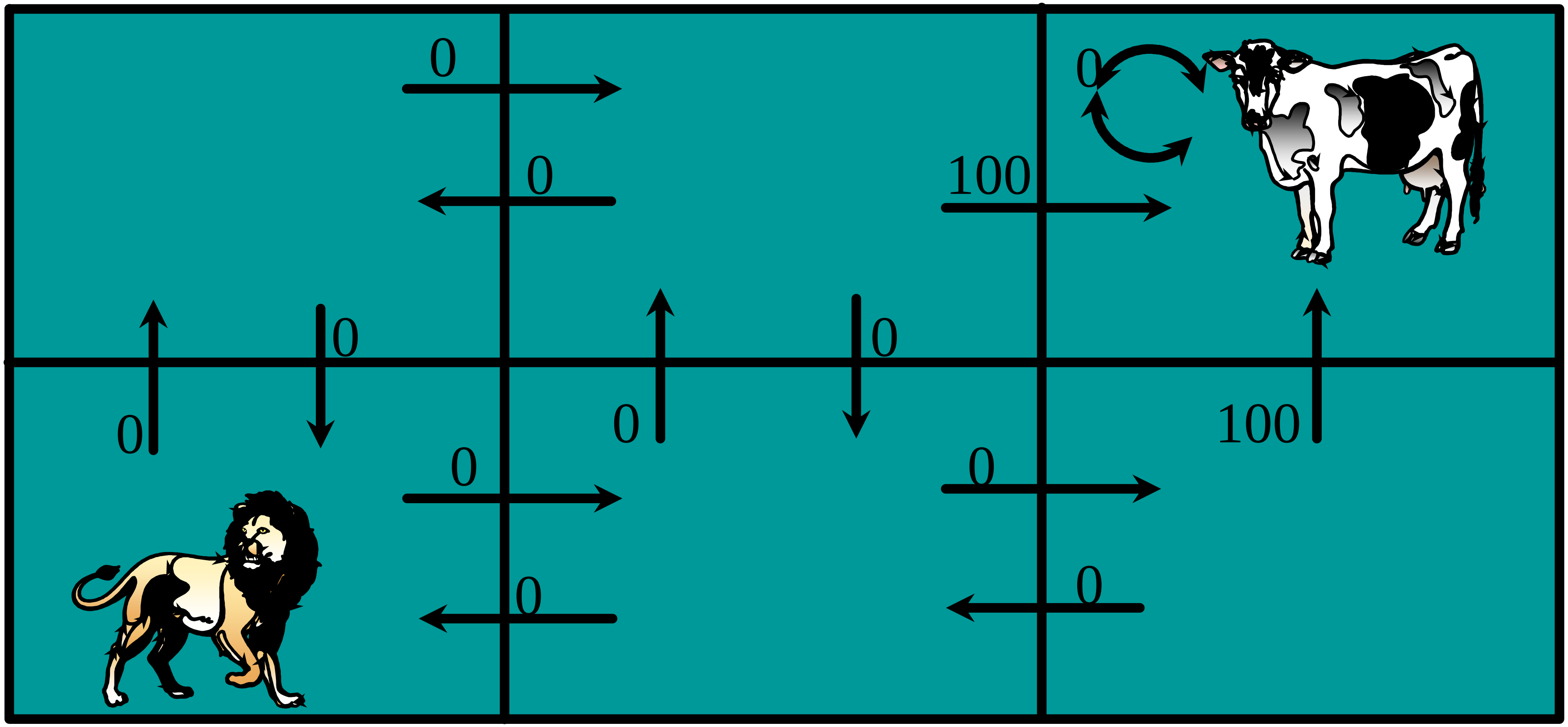
A (single-agent) MDP is a tuple (S,A,p,R) where:

- S is a set of states
- A is a set of actions
- p: $S \times A \times S \rightarrow [0,1]$ specifies the transition probabilities between states.
- R: $S \times A \rightarrow \Re$ specifies the reward for each state-action pair.

Note: deterministic transition function $\delta: S \times A \rightarrow S$

MDP

Example



Agent State vs. Environment State

Crucial Distinction

- Agents have their own representation of the environment state.
- Partial observability: agents may not be able to observe everything in the environment.
- Agent may want to focus in on relevant parts of the environment.
- Ideally: all information necessary to make an optimal decision is contained in the agent state.

Markov States

Sufficient Information in States

Def.: A state S_t is Markov if and only if:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \ldots, S_t)$$

In other words: S_t contains all relevant information to determine the next state.

Note: any state can be made Markov by incorporating the complete history.

RL Output

Optimal Policy

Goal: learn an *optimal* policy $\pi: S \rightarrow A$

Evaluation of policy via discounted cumulative reward: $V^{\pi}(s_t) = \sum_{i \geq 0} \gamma^i r_{t+i}$

where:

- $0 \leq \gamma < 1$ is a discount factor ($\gamma = 0$ means that only immediate reward is considered).
- r_{t+i} is the reward at time t+i determined by performing actions specified by policy π

Optimal Policy

Definition

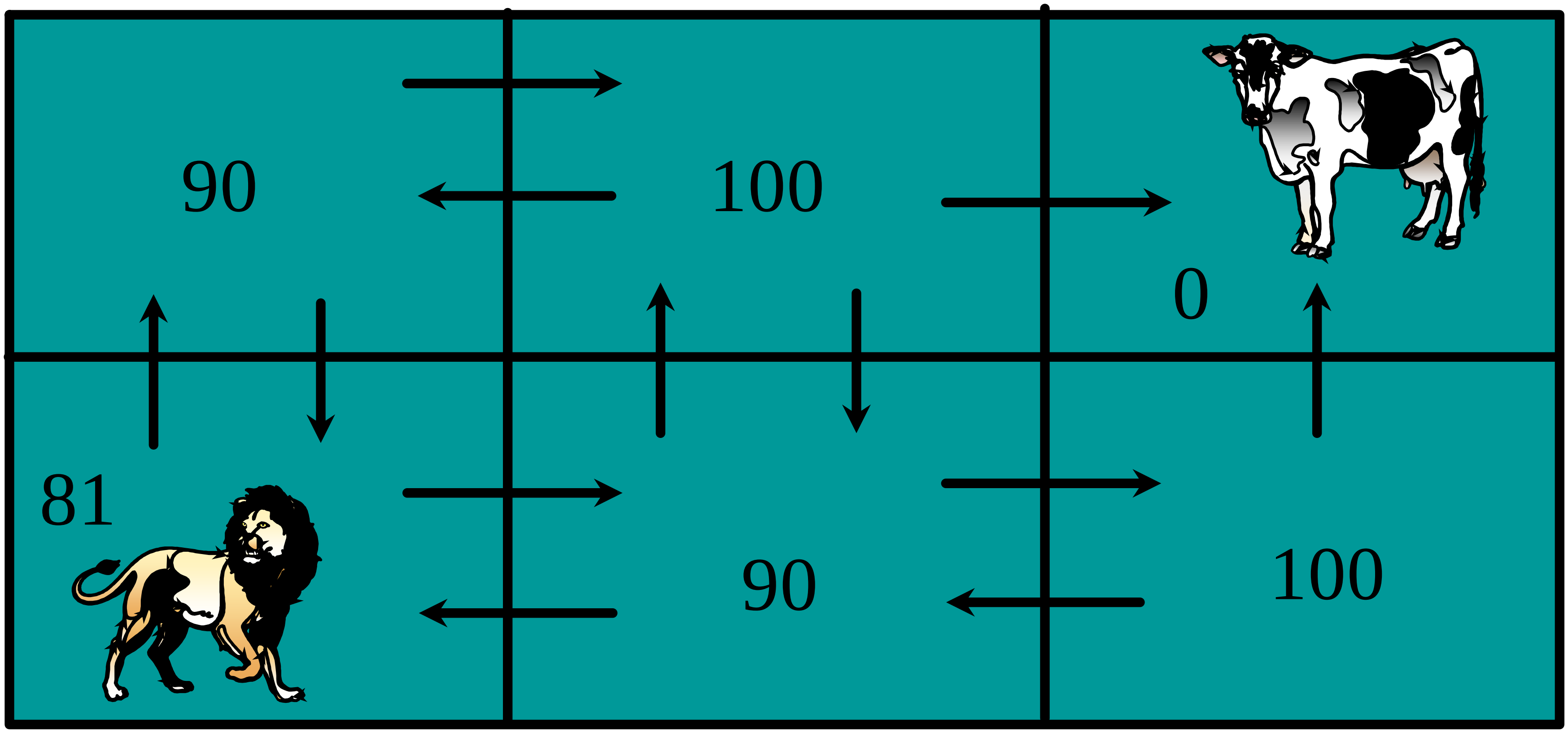
Goal: Agent learns policy π that maximizes $V^{\pi}(s)$ for all states s.

Optimal policy $\pi^* = \text{argmax}_{\pi}\ V^{\pi}(s)$ for all s.

Notation: $V^{\pi^*}(s) = V^*(s)$

State Values

Example: V* values for γ=0.9



Optimal Policy

Computation

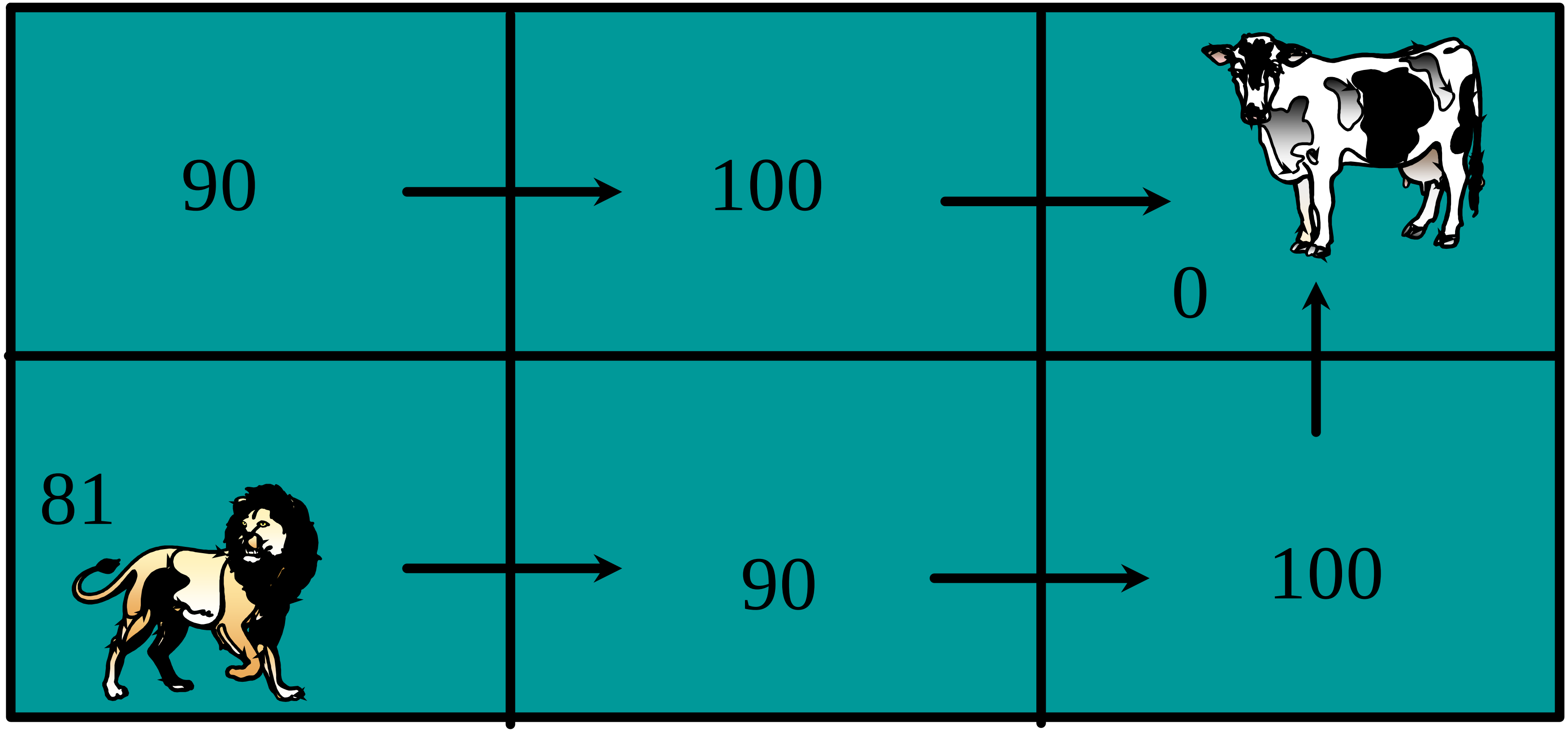
The optimal action in state s is the action a that maximizes the sum of the immediate reward r(s,a) plus the value V* of the immediate successor state, discounted by γ:

$\pi^*(s) = \text{argmax}_a\ [r(s,a) + \gamma V^*(\delta(s,a))]$

If functions r and δ are known then the agent can acquire π^* by computing V* (more details later).

Optimal Policy

Example



Bellman Equation

Computing V*

$$V^*(s) := r(s) + \gamma \max_a \sum_{s'} T(s,a,s')\, V^*(s')$$

For n states: n equations with n unknowns

But: n equations are non-linear (“max” operator).

Value Iteration

Algorithm to Compute V*

1. For all states s: $V_0(s) := 0$

2. t := 0;

3. Update values of all states s based on successor states:

   $V_{t+1}(s) := r(s) + \gamma \max_a V_t(\delta(s,a))$

4. t := t+1;

5. Repeat steps 3 and 4 until convergence (or had enough)

Model-based vs model-free

What if δ and r are unknown?

Solution 1: Learn δ and r from experience (model-based).

Solution 2: Learn quality function Q directly (model-free).

$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$

Optimal policy computation: $\pi^*(s) = \mathrm{argmax}_a\ Q(s,a)$

It is possible to learn Q even if δ and r are unknown!

Q Learning

Algorithm

- $V^*(s) = \max_{a'} Q(s,a')$
- $Q(s,a) = r(s,a) + \gamma\ (\max_{a'} Q(\delta(s,a),a')$
- How to learn Q?
- Observe reward and update estimate Q' accordingly.

Q Learning

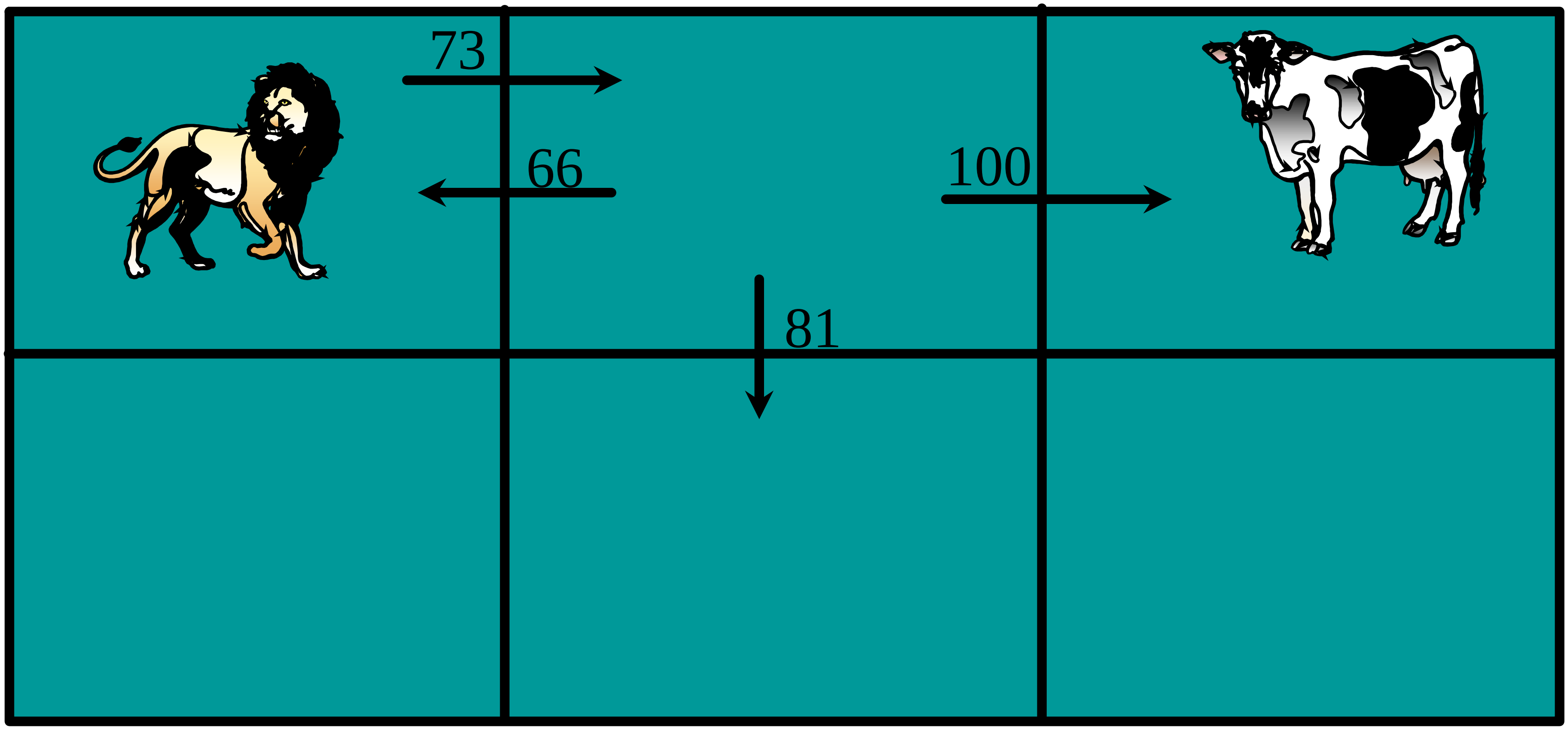
Algorithm

Compute estimate Q' of Q:

For each state s and action a do {
    Q'(s,a) = 0;
}
Do forever {
  Select action a and execute it in current state s;
  r = reward received;
  s' = new state;
  $Q'(s,a) = (1-\alpha)\, Q'(s,a) + \alpha\, (r + \gamma \max_{a'} Q'(s',a'))$
  s = s';
}

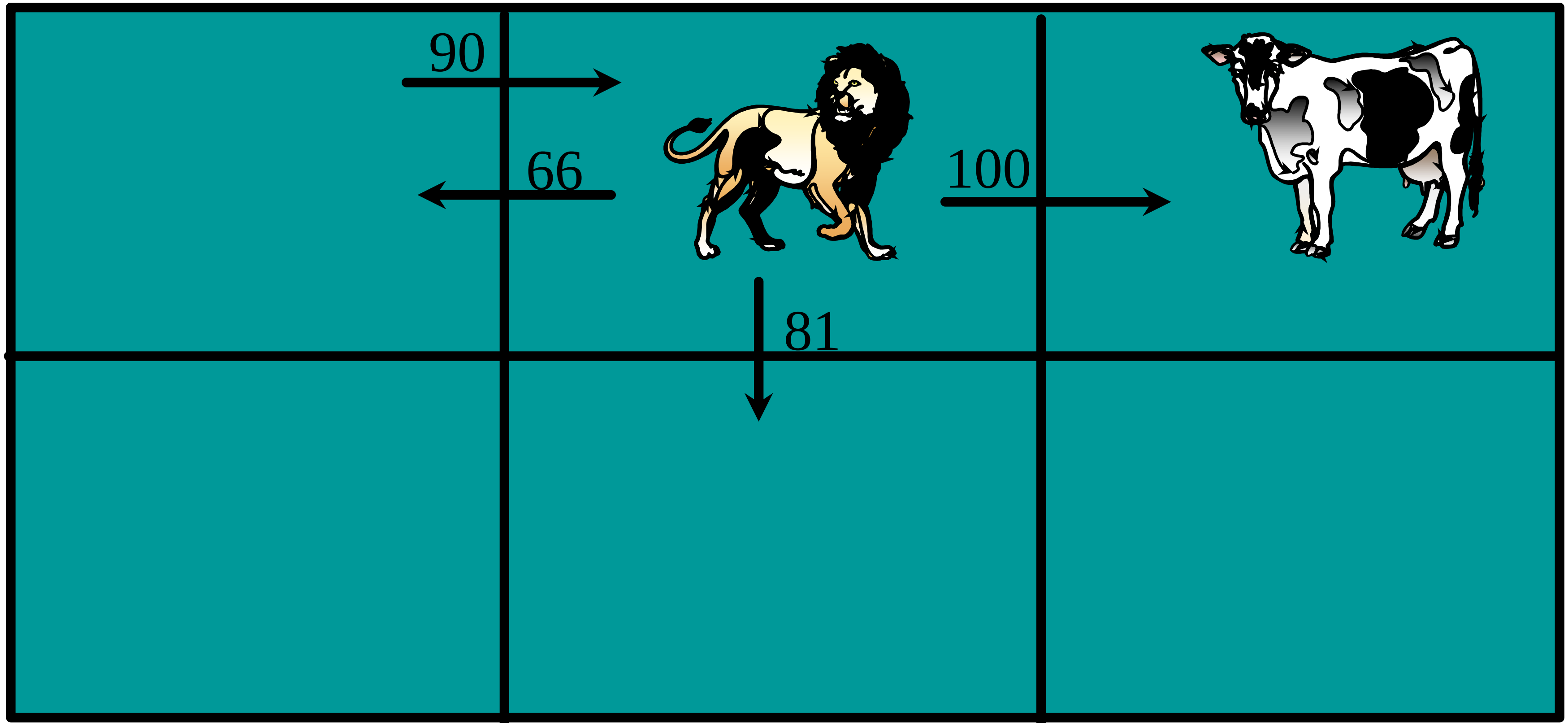
Q Learning

Example: $\gamma = 0.9$, $\alpha = 1$



Q Learning

Example (Cont.)



Q Learning

Convergence

Theorem: Q' will converge to Q, if:

- Reward values have an upper bound.
- Agent visits every possible state-action pair infinitely often.
- $0 \leq \alpha_t < 1, \sum_0^\infty \alpha_t = \infty \; and \sum_0^\infty \alpha_t^2 < \infty$