Markov Decision Processes

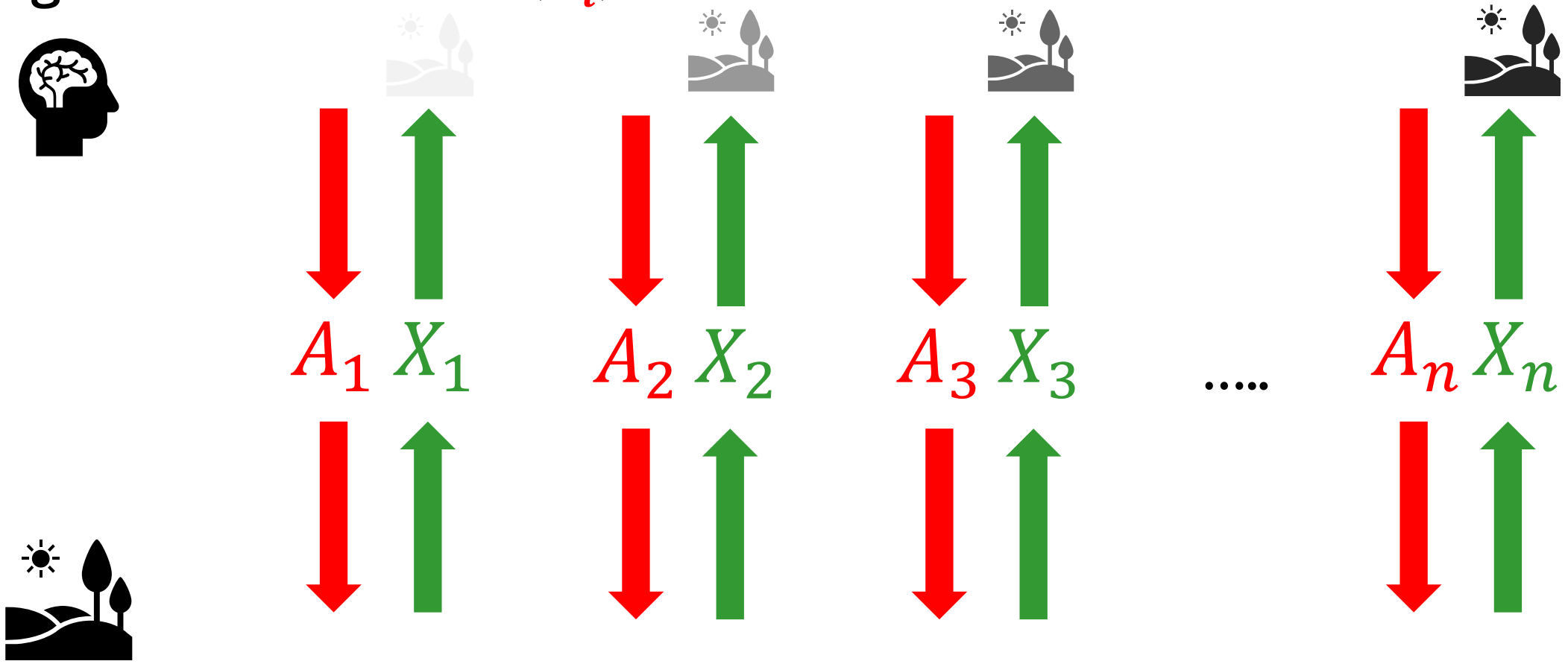
Principles of Online Decision-Making (CS-303)

Prof. Matthias Grossglauser

Information and Network Dynamics (INDY) lab
School of Computer and Communication Sciences (I&C)
EPFL

Stochastic bandits

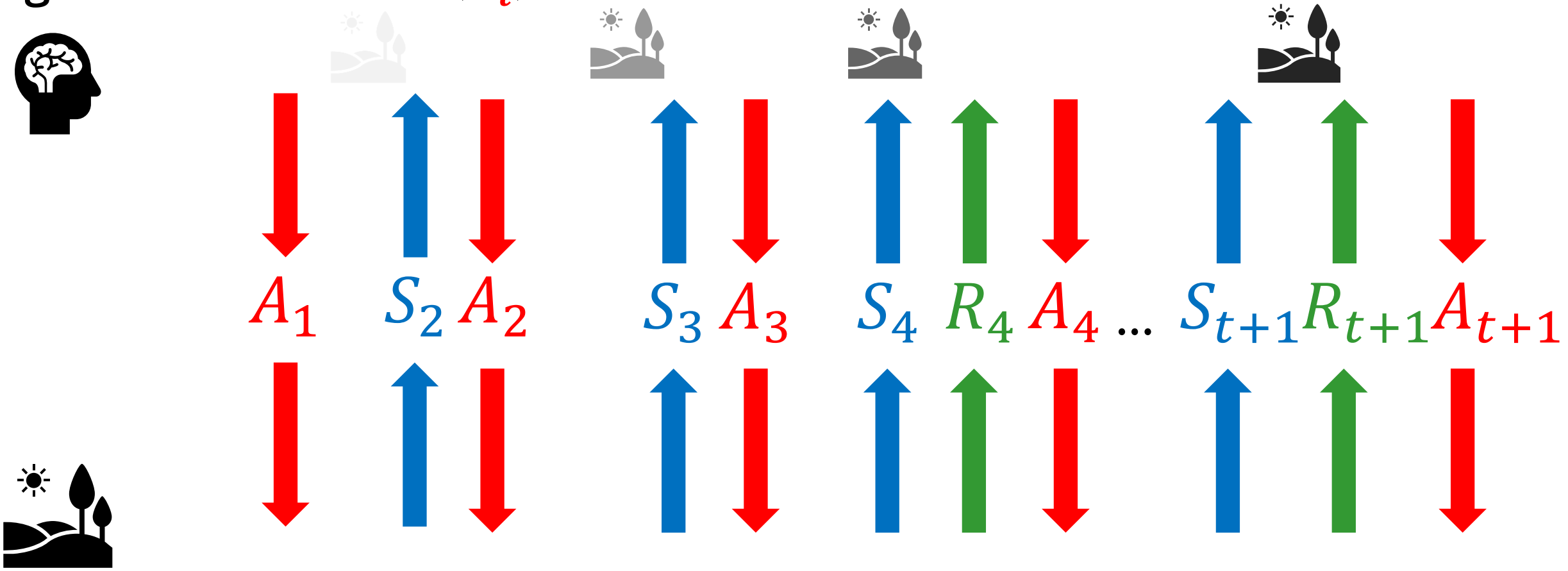
Agent: takes actions (A_i)



Environment: generates rewards (X_i)

Reinforcement learning (RL)

Agent: takes actions (A_t)



Environment: updates state (S_t), generates (occasional) rewards (R_t)

Bandits vs RL: the two key differences

Environment has state!

Delayed gratification!

Bandits vs reinforcement learning (RL)

Bandits	Reinforcement learning
Sequential interaction agent ↔ environment	
Environment is memoryless	Environment has memory (“state”)
Agent has memory (estimators)	Agent has memory (estimators, policy)
Dynamics: agent estimators	Dynamics: environment state + agent estimators
Reward distribution depends on action	Reward distribution depends on state and action
Objective: maximize reward over time	
Total reward	Discounted future reward
Explore-exploit tradeoffs	

Why the term “reinforcement learning”?

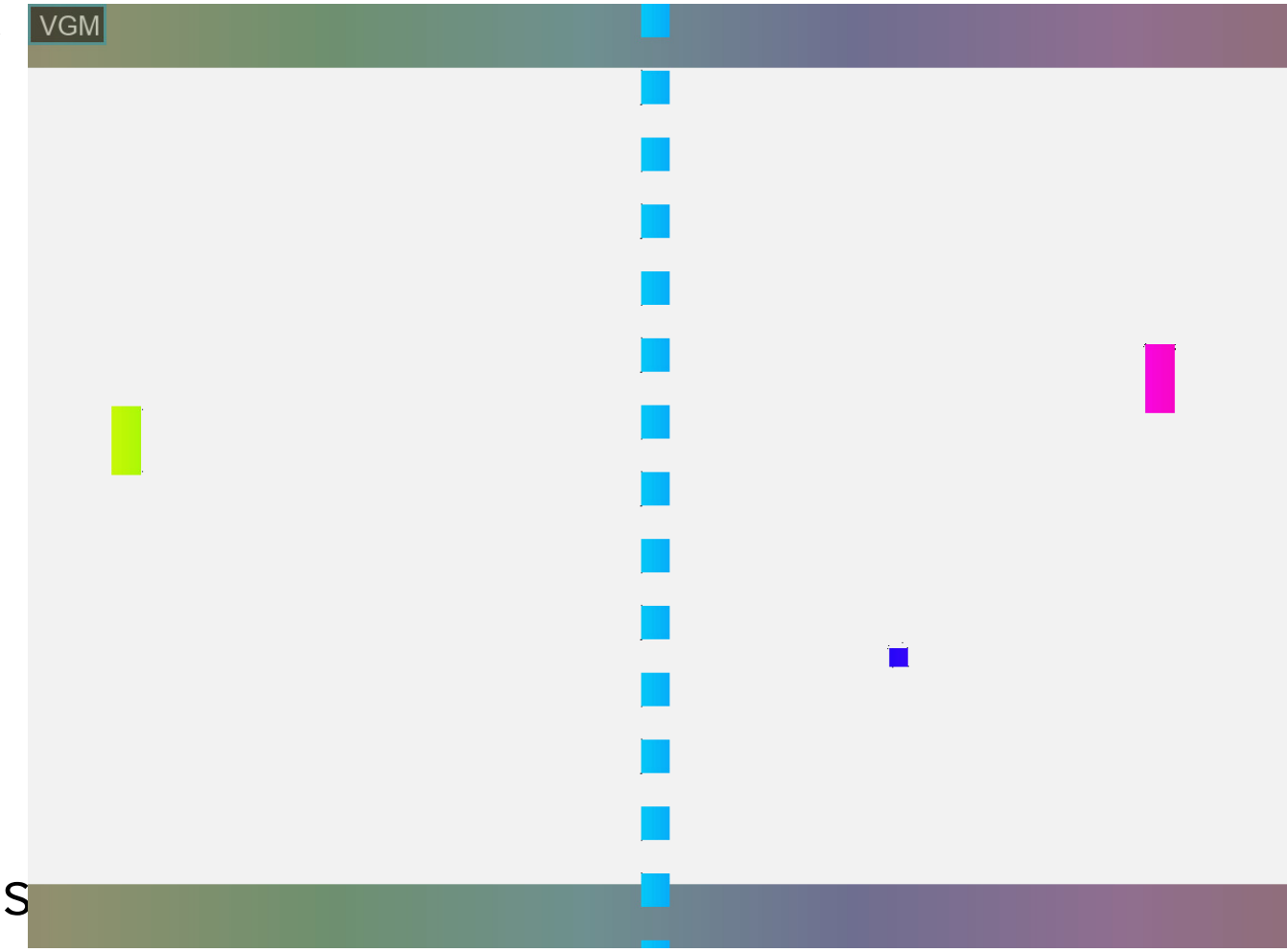
- Reinforcement learning refers to
  - Problem formulation
  - Class of solution methods that work well for these problem
  - And the field studying the two
- Closely related to optimal control
  - OC: known environment/system, achieve some dynamic goal
    - What we do this and next week is more OC than RL
  - RL: do this in a data-driven way: sampling, estimators, statistics,…
- Reinforce behaviors that are desirable, via reward signals
  - It is all about getting better over time
- Difference with supervised and unsupervised:
  - Supervised: needs labeled data a priori, and then does not improve at test time
  - Unsupervised: finds structure in data, but does not take into account signals accruing over time

Mapping a concrete problem to the RL framework

- What is the state?
  - A bit like feature engineering in supervised learning
  - More state variables: more environment signals that the RL algorithm can potentially exploit
  - But also: more state variables → (exponentially) larger search space!
- How to shape rewards?
  - Reward structure is a way to build prior expertise into the model
    - For example: small reward for bouncing the ball off at a steep angle → more likely to score
- This is not a math problem, but engineering+experience+a bit of flair!

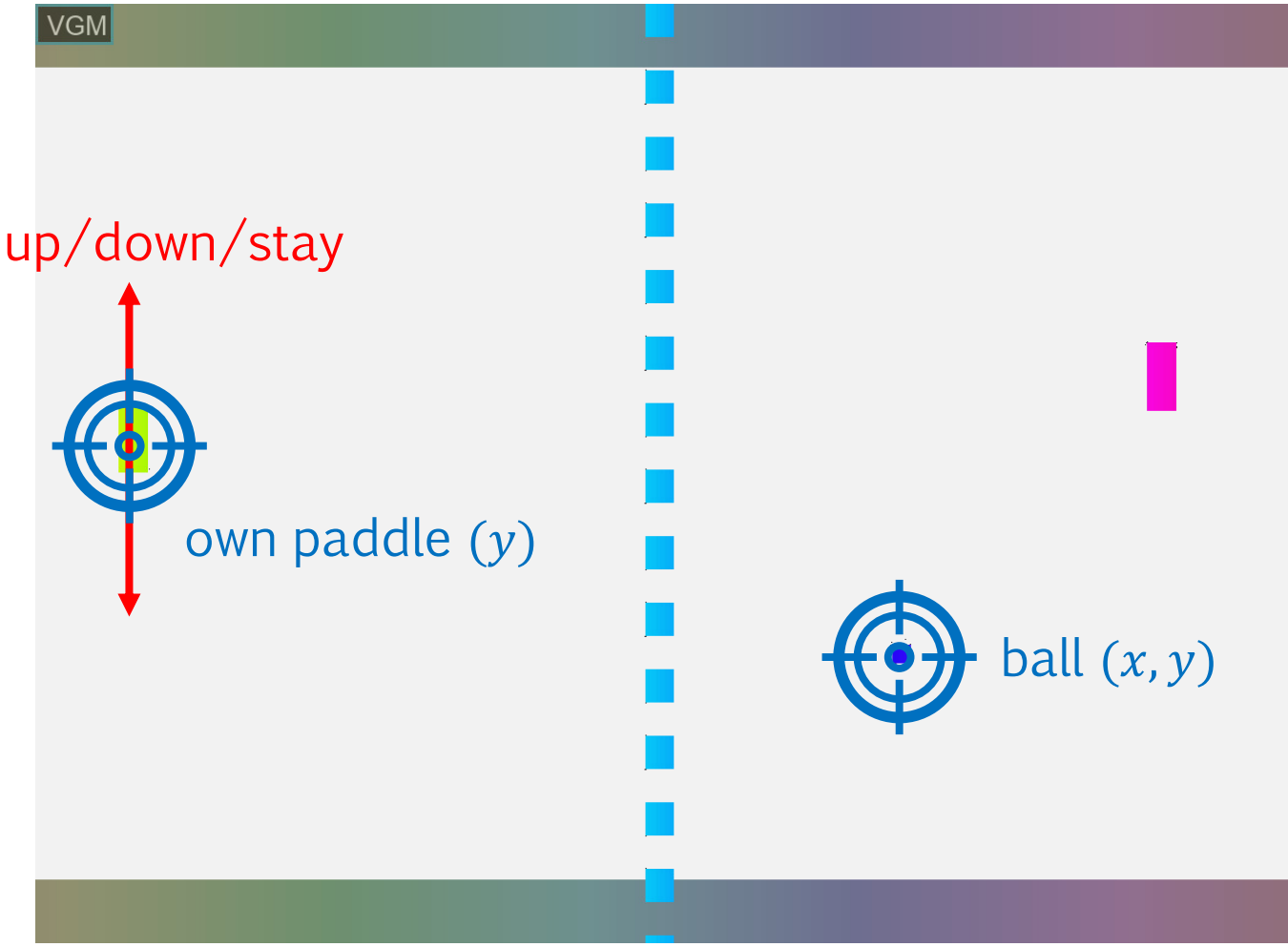
Reinforcement learning

- Example: learn to play a computer game (ATARI Pong)
  - We do not (or cannot) program the rules of the game and then find the optimal move (UP or DOWN)
  - Instead, an algorithm should observe and master the game as quickly as possible by playing and observing
- Feedback: reward signals
  - +1: I score
  - -1: adversary scores
- Reward is the result of a long sequence of environment evolutions and of agent actions



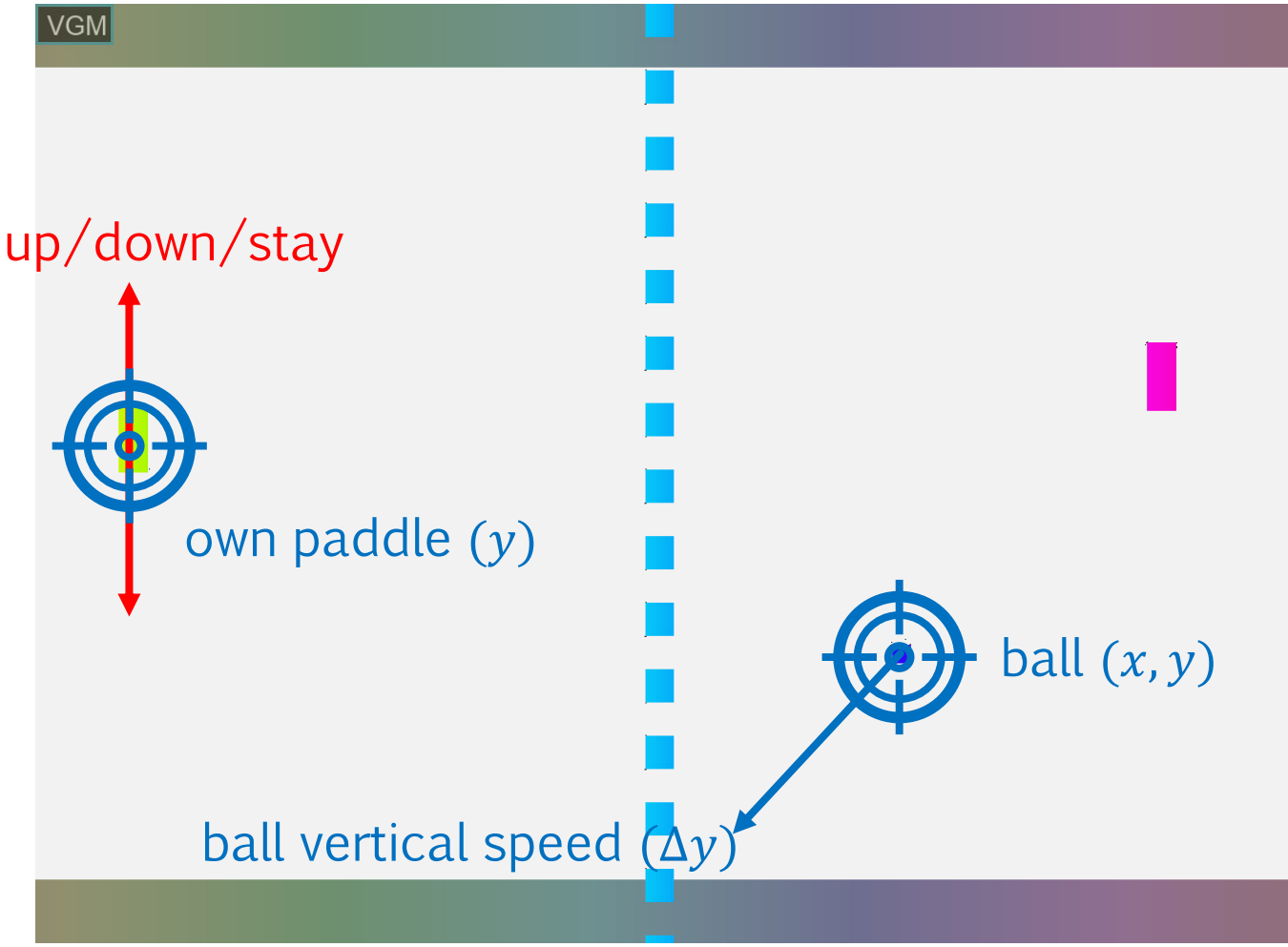
Reinforcement learning

- State space has 3 dimensions
- With this mapping:
  - No adaptation to the adversary
  - No “anticipation” of where the ball is going → if vertical speed of ball is > vertical speed of paddle, might miss



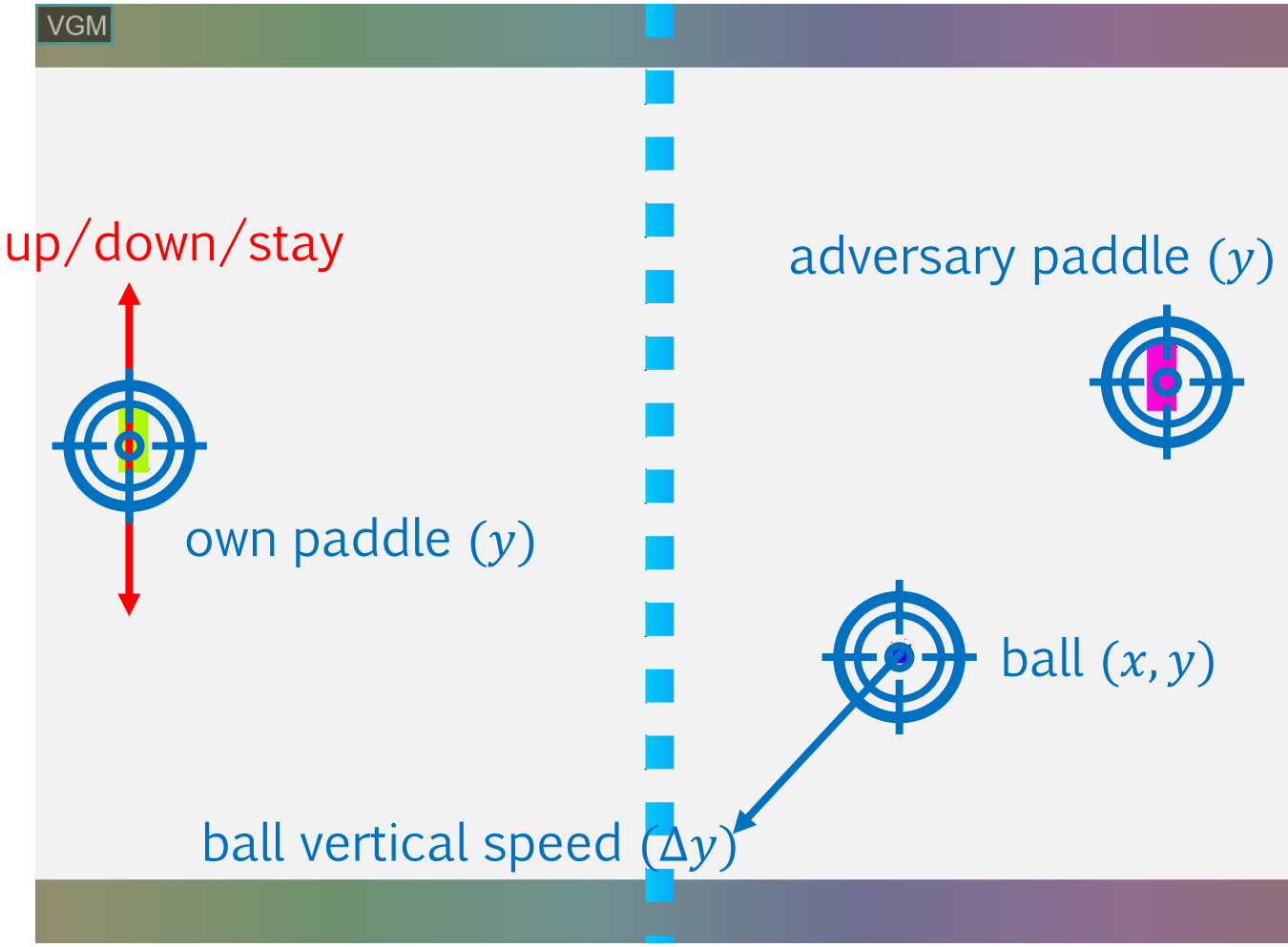
Reinforcement learning

- State space has 4 dimensions
- With this mapping:
  - No adaptation to the adversary
  - Can learn strategies that anticipate where the ball is going to intersect paddle plane



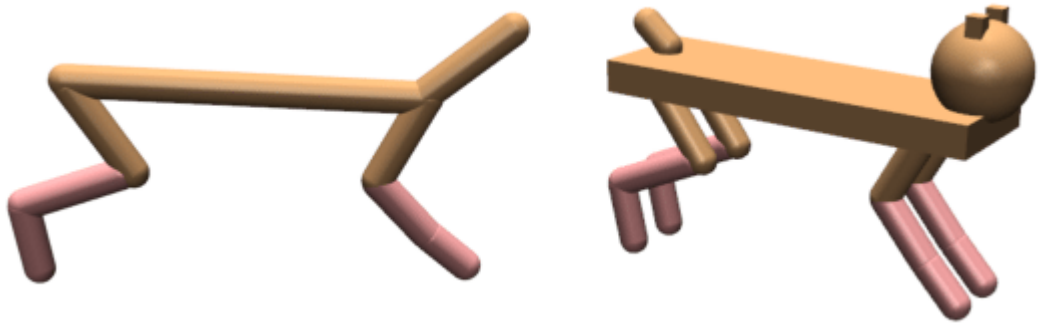
Reinforcement learning

- State space has 5 dimensions
- With this mapping:
  - Adaptation to the adversary possible: could learn strategies that exploit "a gap left in the goal"
  - Can learn strategies that anticipate where the ball is going to intersect own and adversary paddle plane
- If we discretize with 20 levels:
  - $20^5 = 3'200'000$ states
- For this simple game, probably easy to come up with good policy "manually" → RL helps if we cannot



Example: learning to walk

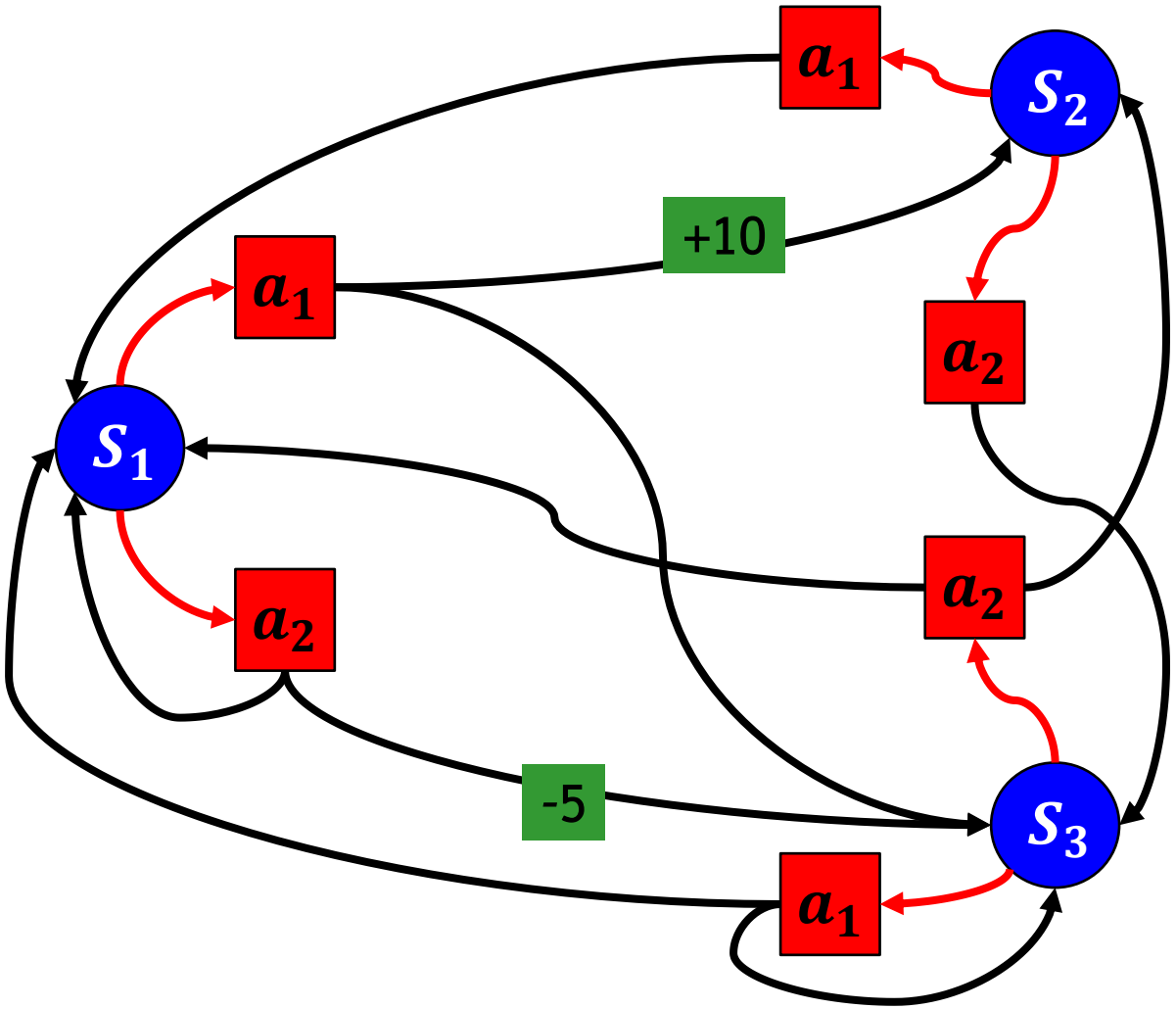
- Model: physics of limbs, gravitation, friction with ground
- Actions: move various joints with variable forces
- Rewards: getting to some place



[medium.com]

- Key challenges: large continuous state space, continuous state space and action space; complex dynamics

Markov Decision Processes (MDP)



- Policy π:
  - Maps a state to an action
  - **Random**
- Goal: max discounted reward over a long time horizon
- Note: reward is per (state, action)
  - If we want to e.g. avoid or encourage a whole state → penalize/reward every transition into that state

Formal definition of MDP

- Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$
- Environment state space: $S_t \in \mathcal{S}$
- Action space: $A_t \in \mathcal{A}(s)$ (often we assume $\mathcal{A}(s) = \mathcal{A}$)
- Reward: $R_t \in \mathcal{R}$
- Model of the **environment**: $p(s', r|s, a) = \mathbb{P}[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a]$
  - This function defines the dynamics of the MDP
- MDP is Markovian: next state and reward is conditionally independent of the past, given the preceding state and action
  - Under this assumption, $p(.,.|.,.)$ fully describes the environment model
- Time steps: need not correspond to wall clock – step = whenever the agent needs to do something, and/or we get fresh information

Expected reward

- Expected reward when taking action a in state s:
  - $r(s,a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$

$$= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

- Expected reward when going from state s to state s' with action a:
  - $r(s,a,s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$

$$= \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s' | s, a)}$$

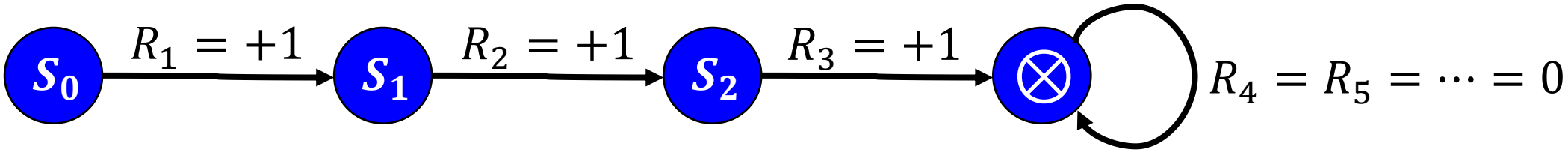
- Our goal will be to find a policy π that leads to large **cumulative** reward

Episodes vs continuing tasks

- Episodic tasks: some situations have a natural beginning and end
  - Examples:
    - Gaming: playing until checkmate/"game over"
    - Roomba: vacuuming the house once
    - Robotics: assembling one car
  - Special absorbing end state in MDP: $\otimes$
  - Stopping time T when MDP first hits $\otimes$
  - Data: typically, many independent episodes
- Continuing tasks: some situations are naturally continuing "forever"
  - Examples:
    - Running a whole factory
    - Adjust appearance of a website or mobile app
    - Self-driving car
  - Total reward may be $\pm\infty$ → we use discounting: later reward counts less today
  - Data: may be small set of long trajectories

Expected return

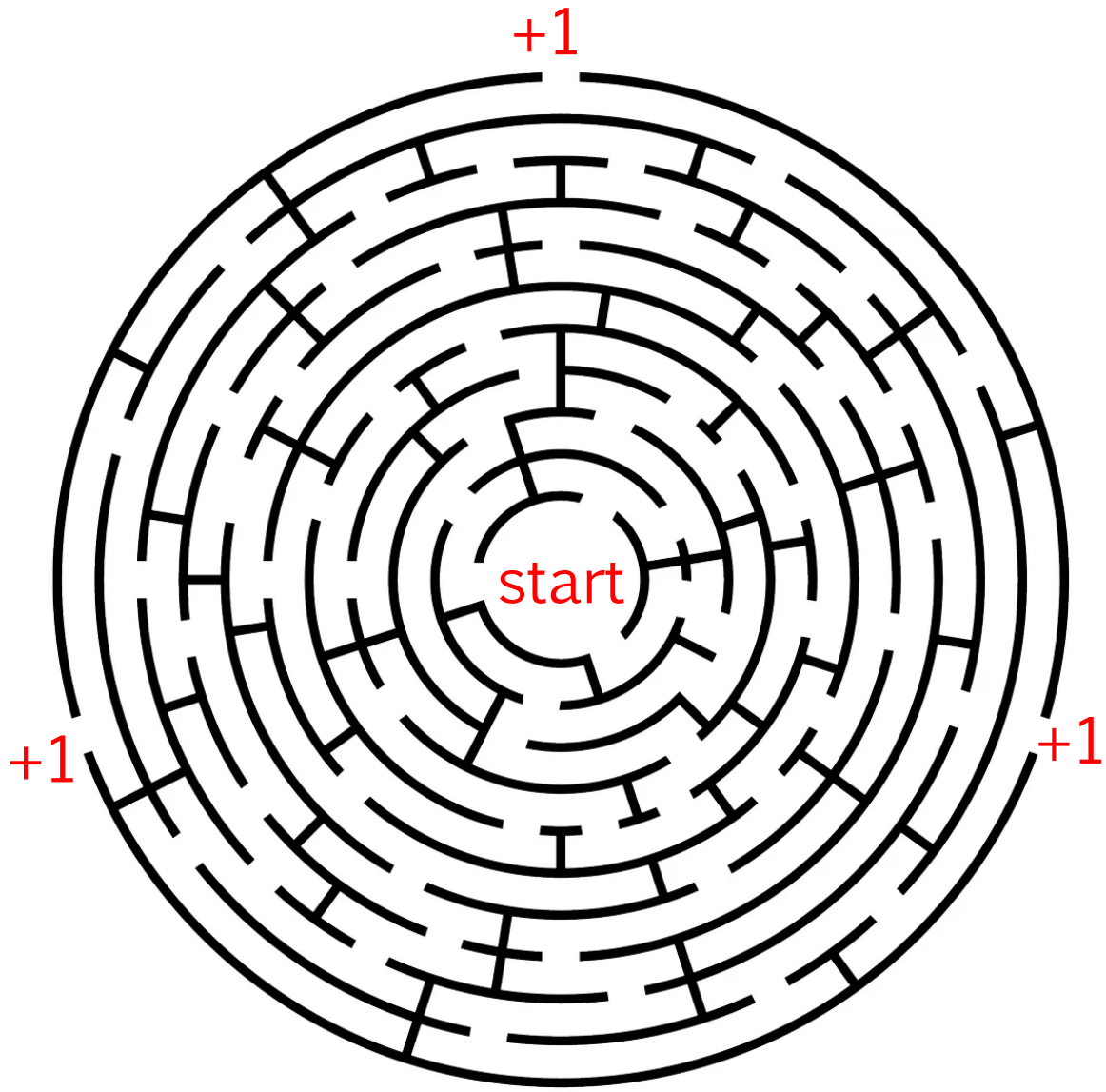
- Episodic: $G_t \stackrel{\text{def}}{=} R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$
  - $\gamma = 1$
- Continuing: $G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
  - Present value of future rewards
  - $0 \leq \gamma \leq 1$: discount rate
  - Avoids diverging returns (as long as $\mathcal{R}$ itself is bounded)
- Recursive relationship: $$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$
  - Note: this also works for episodic for all $t < T$, but need to set $G_T = 0$



Incentive design is important

- Example: we design a robot to escape from a maze
- Natural to make it episodic: one successful escape = one episode
- Rewards:
  - +1 once we reach the outside
  - 0 otherwise

- ...and then we observe that our robot does not learn and plods around randomly! Why not?
- Solution?



[popular science]

Policy π

- That's the thing we want to learn: what action to take given the state of the system!
- Deterministic policy: always take the same action in the same state
  - $a = \pi(s)$
- Random policy: prescribes the probabilities for different actions given the state
  - $\pi(a|s) =$ probability of selecting action a when in state s
  - This allows for interpolating between exploration (trying different actions in a state) and exploitation (make one action more likely than another)
- Note: for fixed π, the MDP is a Markov chain

Value functions $v_\pi(s)$ and $q_\pi(s)$

- How good is state s, i.e., what is the expected return (cumulative discounted reward)?
  - This depends both on the model of the environment ($p(.,.|.,.)$) and the policy ($\pi(.|.)$)
- $v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$:
  state-value function for policy π

- How good is an action a when we are in state s?
- $q_\pi(s,a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\,, A_t = a]$:
  action-value function for policy π

Bellman equation

- Recursive consistency condition on values of all states

- $v_\pi(s)$

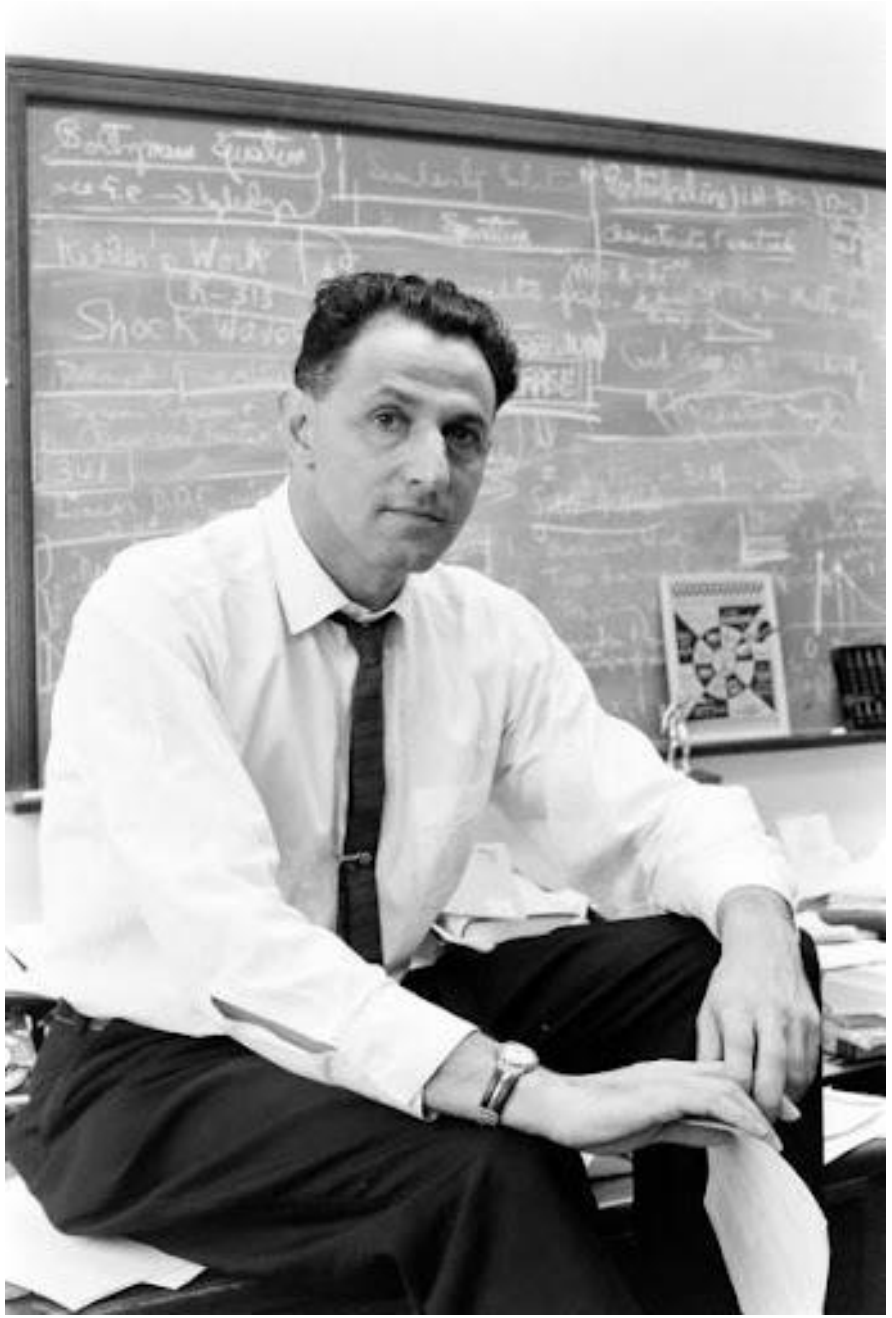
$$\begin{aligned}
&\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}$$

- Linear system of $|\mathcal{S}|$ equations in $|\mathcal{S}|$ unknowns

- The value function $v_\pi(.)$ is the unique solution of this system

Richard E. Bellman (1920-1984)

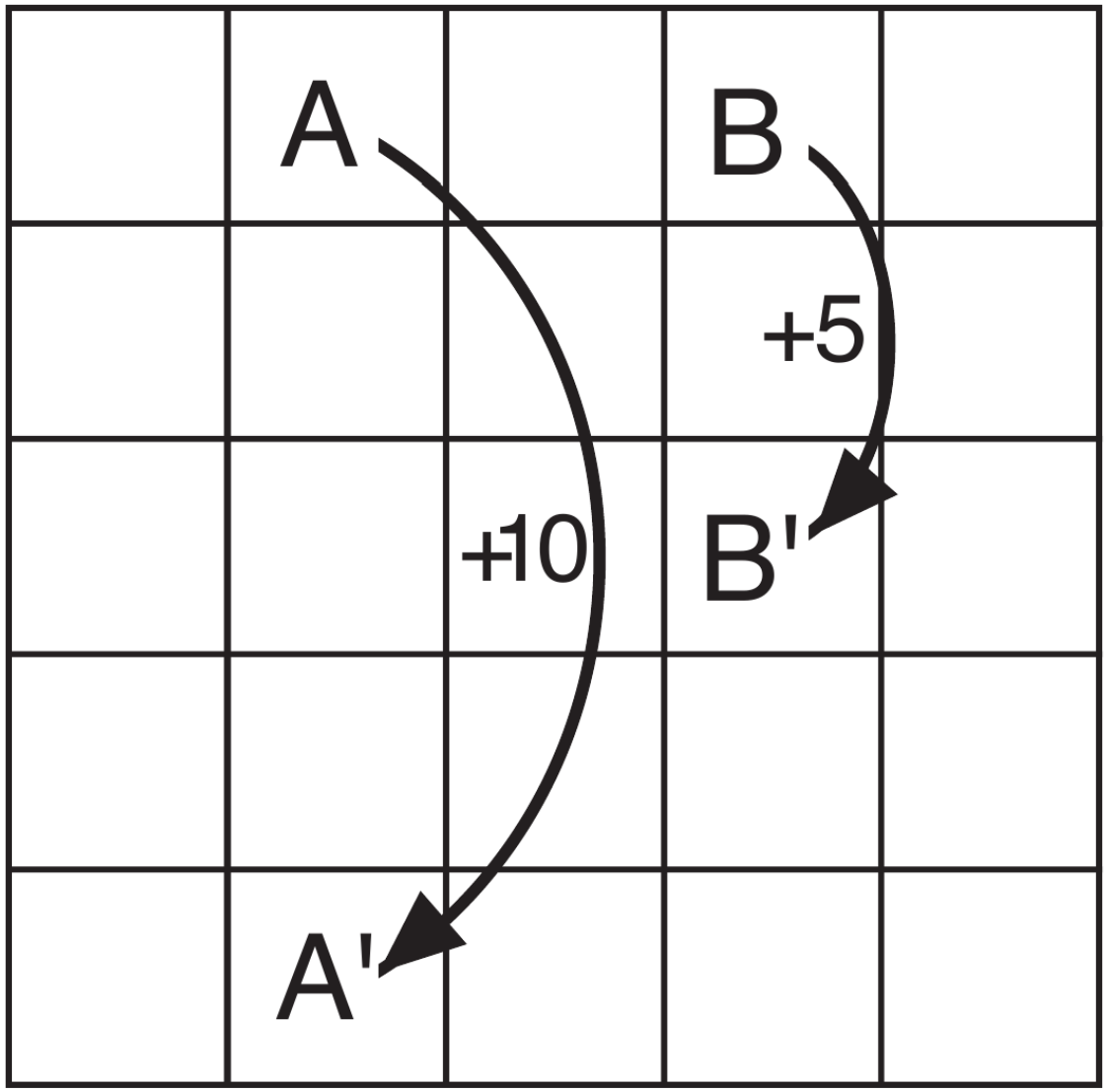
- 1920: Born in NYC (Brooklyn), studied math
- 1944-1946: worked on the Manhattan project in Los Alamos (theoretical physics)
- 1946: Ph.D. in mathematics, Princeton
- 1946-1948: Assistant prof at Princeton
- 1948: left to Stanford U, soon returned to Princeton to work on H-bomb
- 1952: left Stanford to RAND corporation
- 1965: moved to USC
- Father of “dynamic programming” (next week)
- Coined the famous term “curse of dimensionality”



Bellman equation

- $\boldsymbol{V}_\pi = \boldsymbol{R}_\pi + \gamma \boldsymbol{P}_\pi \boldsymbol{V}_\pi$
  - $\boldsymbol{V}_\pi \in \mathbb{R}^{|\mathcal{S}|} = (v_\pi(1), v_\pi(2), \dots, v_\pi(|\mathcal{S}|))$
  - $\boldsymbol{R}_\pi \in \mathbb{R}^{|\mathcal{S}|}$: $R_\pi(s) = \sum_a \pi(a|s) r(s,a)$
  - $\boldsymbol{P}_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$: $P_\pi(s) = \sum_a \pi(a|s) p(s'|s,a)$
- Solution: $\boldsymbol{V_\pi = (I - \gamma P_\pi)^{-1} R_\pi}$
  - For $\gamma < 1$, $(\boldsymbol{I - \gamma P_\pi})$ is always full-rank so the inverse exists → unique solution
- Computationally costly:
  - Computing $\boldsymbol{P}_\pi$: $O(|\mathcal{S}|^2 |\mathcal{A}|)$
  - Inverting $(\boldsymbol{I - \gamma P_\pi})$: $O(|\mathcal{S}|^3)$
- We will develop iterative approximation methods for large systems → dynamic programming, next week

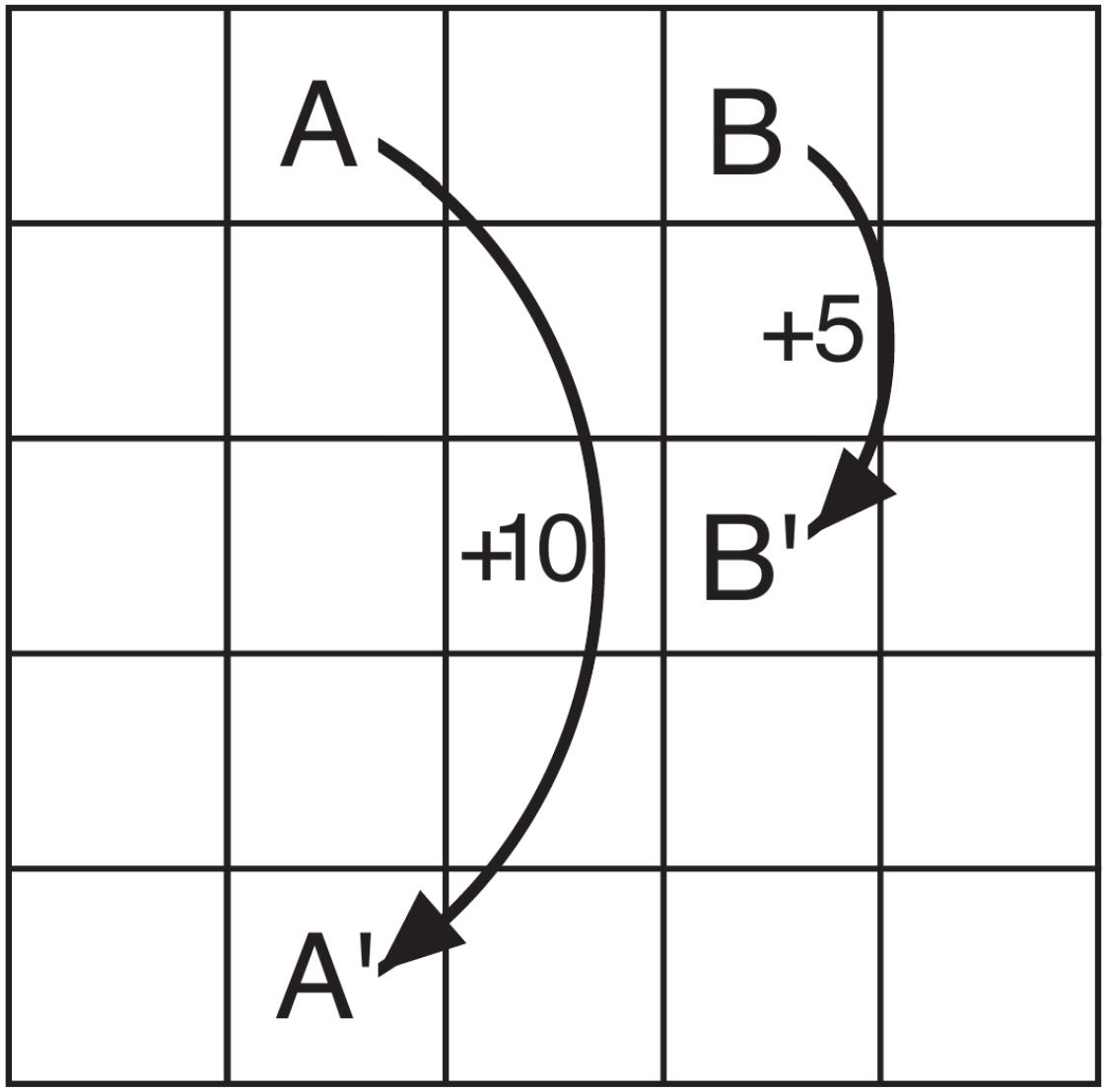
Gridworld example



- Possible actions: N,S,E,W
- All actions have reward 0, except for walls and special transitions
- Special transition:
  - From A, any action jumps to A', +10
  - From B, any action jumps to B', +5
- Borders: reward -1 if "bumping against the wall"

- Suppose π = uniform(N,S,E,W)

Gridworld example



$v_*(s)$

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Gridworld example

- $v_\pi(s)$ for all $s \in [5]^2$ for $\gamma = 0.9$
- Observations:
  - Bottom-left and bottom-right: strongly negative because of high chance of bumping into wall soon
  - Top-left and top-right: this is tempered by likelihood of soon moving to A or B, high reward
  - $v_\pi(A)$ is < 10 because after A→B, high likelihood of soon bumping into the wall
  - $v_\pi(B)$ is > 5 because less likely to soon bump into the wall

$v_*(s)$

3.3	A 8.8	4.4	B 5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	B' 0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	A' -1.3	-1.2	-1.4	-2.0

Bellman equation for action-value function

- We can also characterize the action-value function $q_\pi(s,a)$ via the Bellman equations:

- $q_\pi(s,a) \quad \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$$

$$= \sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s')\, \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', A_{t+1} = a']\right]$$

$$= \sum_{s'}\sum_{r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s')\, q_\pi(s', a')\right]$$

Optimal policies and value functions

- Recall: reinforcement learning is about finding good policies!

- Def: a policy π is better (or equal) to another policy π' if its expected return is greater than or equal to the expected return of π' in all states
  - $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s)$ for all states $s \in \mathcal{S}$
  - Defines a partial order over policies

- There always exists at least one policy π_* (optimal policy) that is $\pi_* \geq \pi$ for all π
- Optimal state-value function: $v_*(s) \stackrel{\text{def}}{=} \max_\pi v_\pi(s)$ for all states $s \in \mathcal{S}$
- Optimal action-value function: $q_*(s,a) \stackrel{\text{def}}{=} \max_\pi q_\pi(s,a)$ for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$
  - Can be written as $q_*(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$

Bellman optimality equation for v_* and q_*

- Write Bellman condition for v_*:

- $v_*(s) \stackrel{\text{def}}{=} \max_a q_{\pi_*}(s,a)$

$$= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a]$$
$$= \max_a \mathbb{E}_{\pi_*}[R_t + \gamma G_{t+1} | S_t = s, A_t = a]$$
$$= \max_a \mathbb{E}_{\pi_*}[R_t + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')]$$

- $q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right]$

$$= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')]$$

- Note: this is not a linear system of equations any more

$\pi \rightarrow *$: Sum → max

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

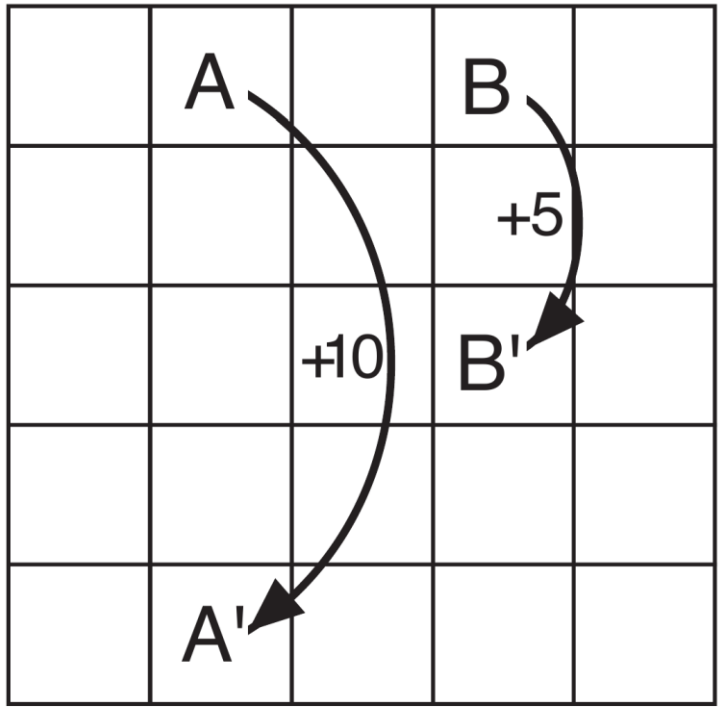
$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

Note: among all policies π_*, there is always at least one deterministic policy:
for each s, set $\pi_*(s) =$ one of the actions a that maximizes $q_*(s,a)$

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a') \right]$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right]$$

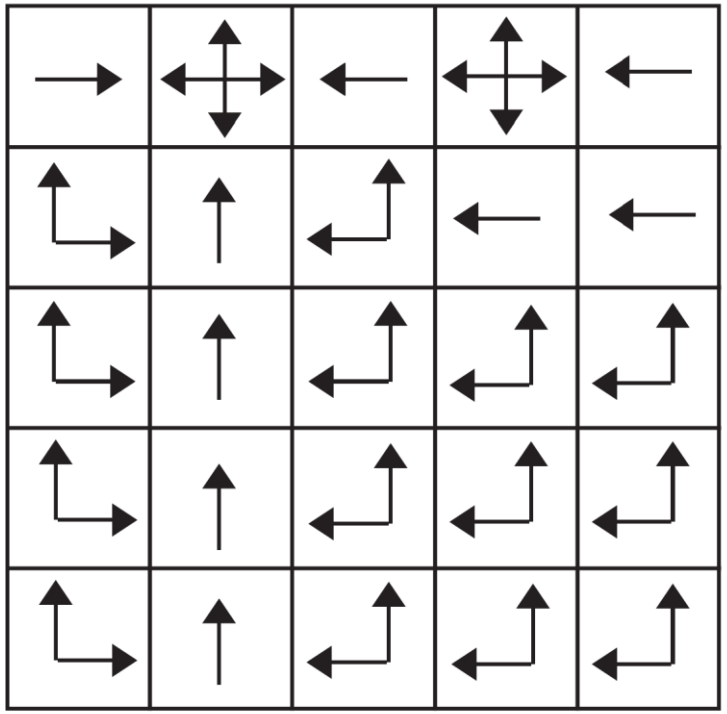
Solving the Gridworld



$v_*(s)$

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

$q_*(s,a)$



All states s with multiple arrows: any probability distribution gives π_* (including constant action → deterministic policy)

Optimal policy π_* by solving Bellman optimality equations

- We rely so far on strong assumptions:
  - (1) We accurately know the dynamics of the environment ($p(s', t|s, a)$) → we will use learning methods to deal with unknown environments
  - (2) Sufficient computational power to solve nonlinear problem → approximation, dimensionality reduction
  - (3) Markov property → might need large state space
- Methods enumerating and computing explicitly for all states: **tabular**
- Example: game of backgammon: (1) and (3) are reasonable assumptions, (2): $|\mathcal{S}| \approx 10^{20}$
  - But: with large $|\mathcal{S}|$, many states may be so unlikely as to be safely ignored
  - Goal: figure this out from agent-environment interaction, focus samples on important states → "reinforcement"
- Approaches:
  - Approximation, search heuristics, dimensionality reduction

Summary

- Keep in mind: so far, we have not dealt with the learning problem (i.e., adapting to unknown environment by collecting statistics) - we pretend we know the environment perfectly (access to $p(s', r|s, a)$), and we want to find a policy maximizing expected cumulative return – "solving the MDP"
- We have so far simply ascertained what the optimal policy looks like when we know the system dynamics perfectly
- MDP: environment has state, agent chooses actions
- Bellman equations: consistency relationship for value- and action-functions for a specific policy π (linear), and for the optimal policy π_* (non-linear)
- Reading assignment:
  - [Sutton&Barto]: 3.1-3.6, 3.8