# Lecture 3: Markov Decision Processes and Dynamic Programming

Optimal policies for Lagrangian turbulence – Dr. Robin Heinonen

Aqtivate workshop on data-driven and model-based tools for complex flows and complex fluids

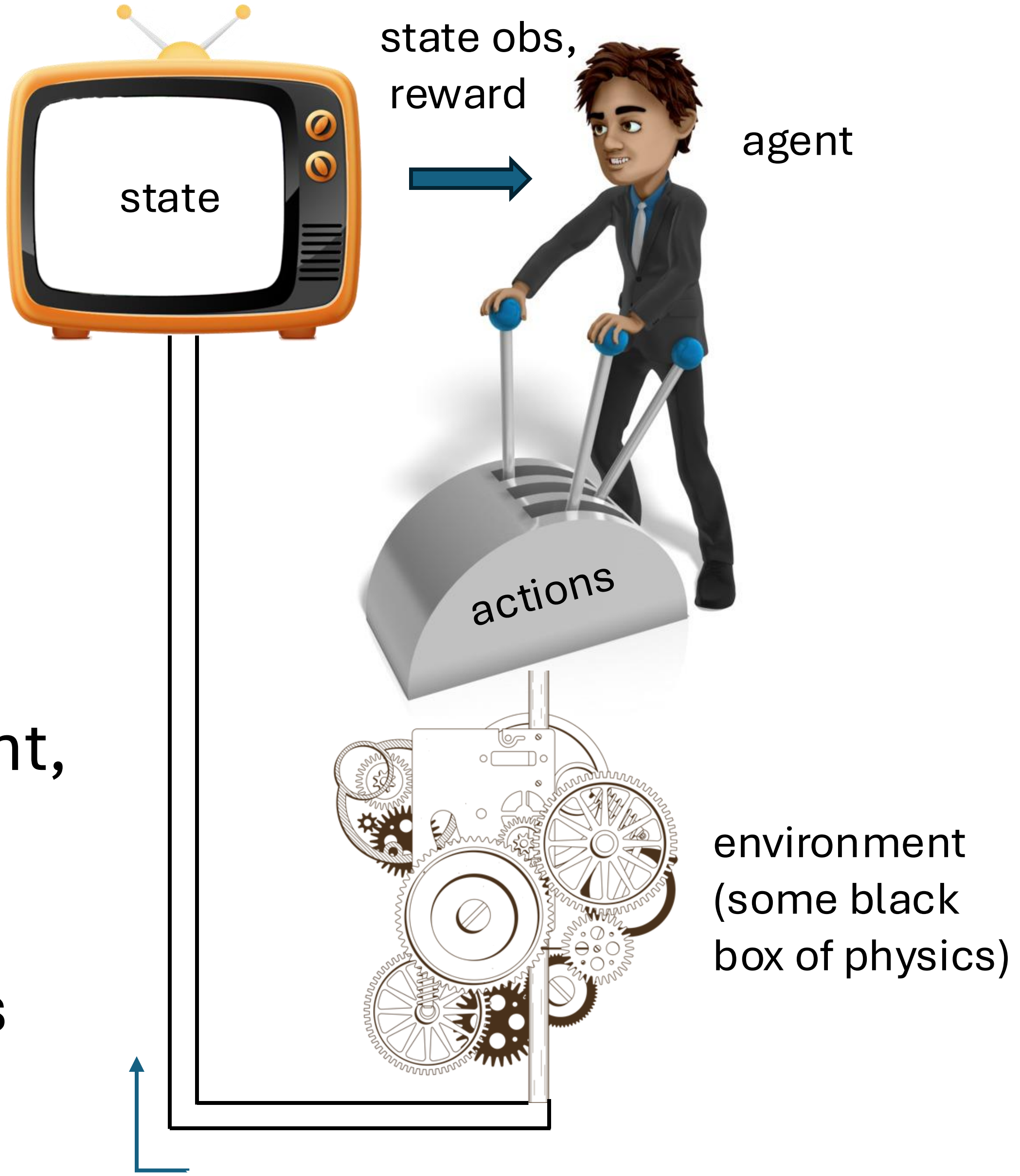
June 3-7

# Intro

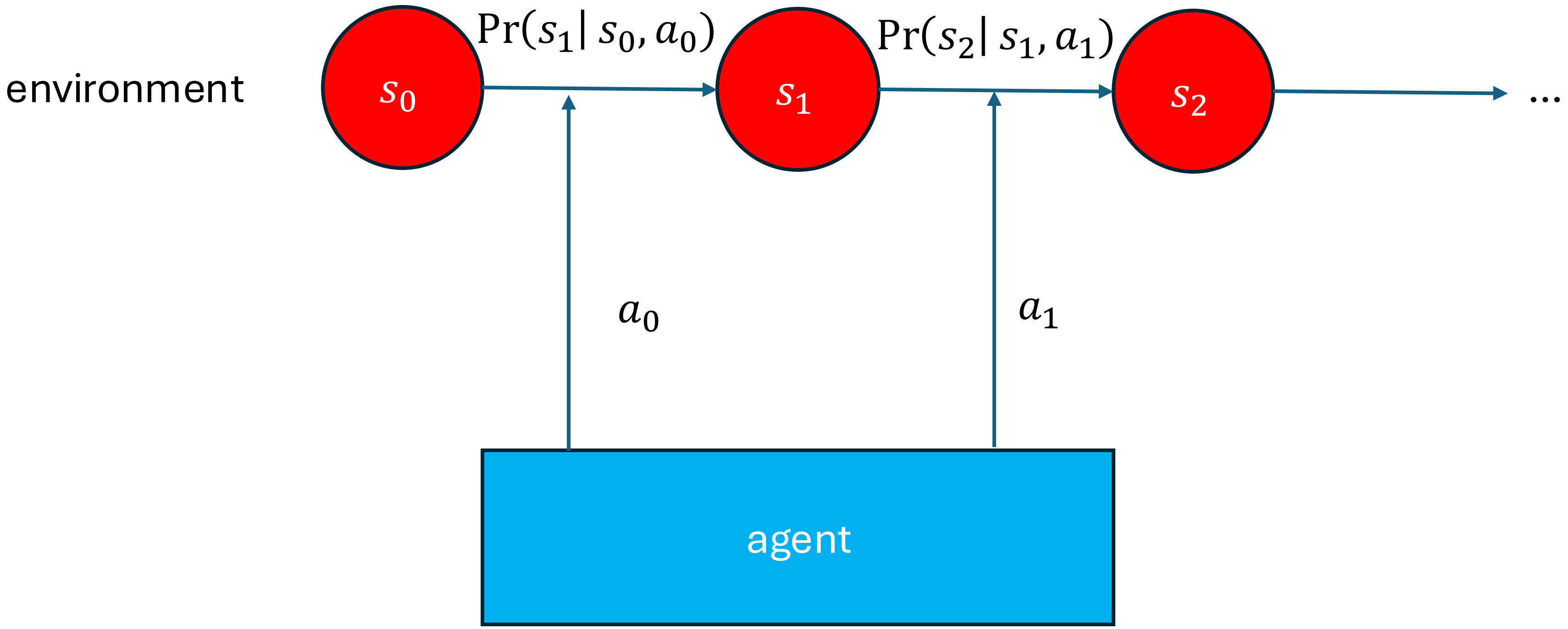
- Last time we noted that Pontryagin Maximum Principle is unsuitable for control of stochastic systems. **Turbulence is inherently stochastic**
- Need a framework for optimization and **decision making in the face of randomness**
- Appropriate framework is called a ***Markov decision process***. Can describe ***virtually all*** **decision problems**
- We will discretize time for this discussion, but it is possible to generalize to continuous time

# Environments and agents

- Environment: system evolving stochastically in time
- Environment characterized by its ***state*** $s_t \in S$
- An ***agent*** interacts with the environment, observing $s$ and taking ***actions*** $a_t \in A$
- State transitions are **Markovian**. Depends on previous state and agent's action via $\Pr(s_{t+1} | s_t, a_t)$



# Environments and agents

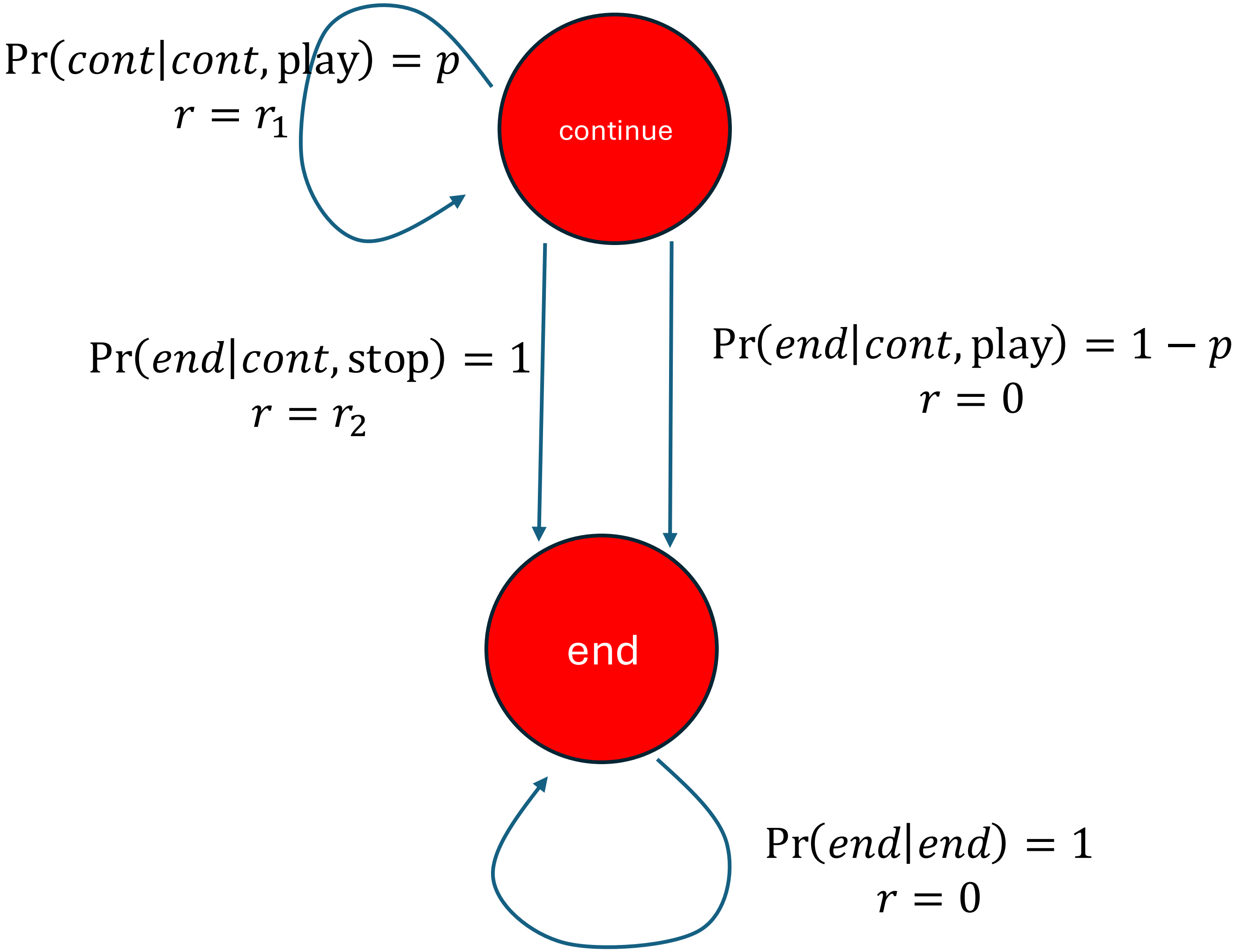


# Rewards and policies

- At each timestep, agent incurs ***reward*** $r$ according to $\Pr(r|s,a)$
- N.B. reward will typically be deterministic for our purposes
- Goal: **maximize (on average) total *discounted* rewards**
$$R = \sum_{t=0}^{\infty} \gamma^t r_t.$$
- $0 < \gamma \leq 1$ problem parameter called ***discount factor***, regularizes $R$. $(\log 1/\gamma)^{-1}$ is characteristic time over which rewards are important ("horizon")
- $\gamma \to 0$: agent myopic/greedy. Only cares about immediate rewards
- Need to **craft *policy*** $\pi(a|s)$ which selects actions based on state (in general draw from a state-dependent probability distro)

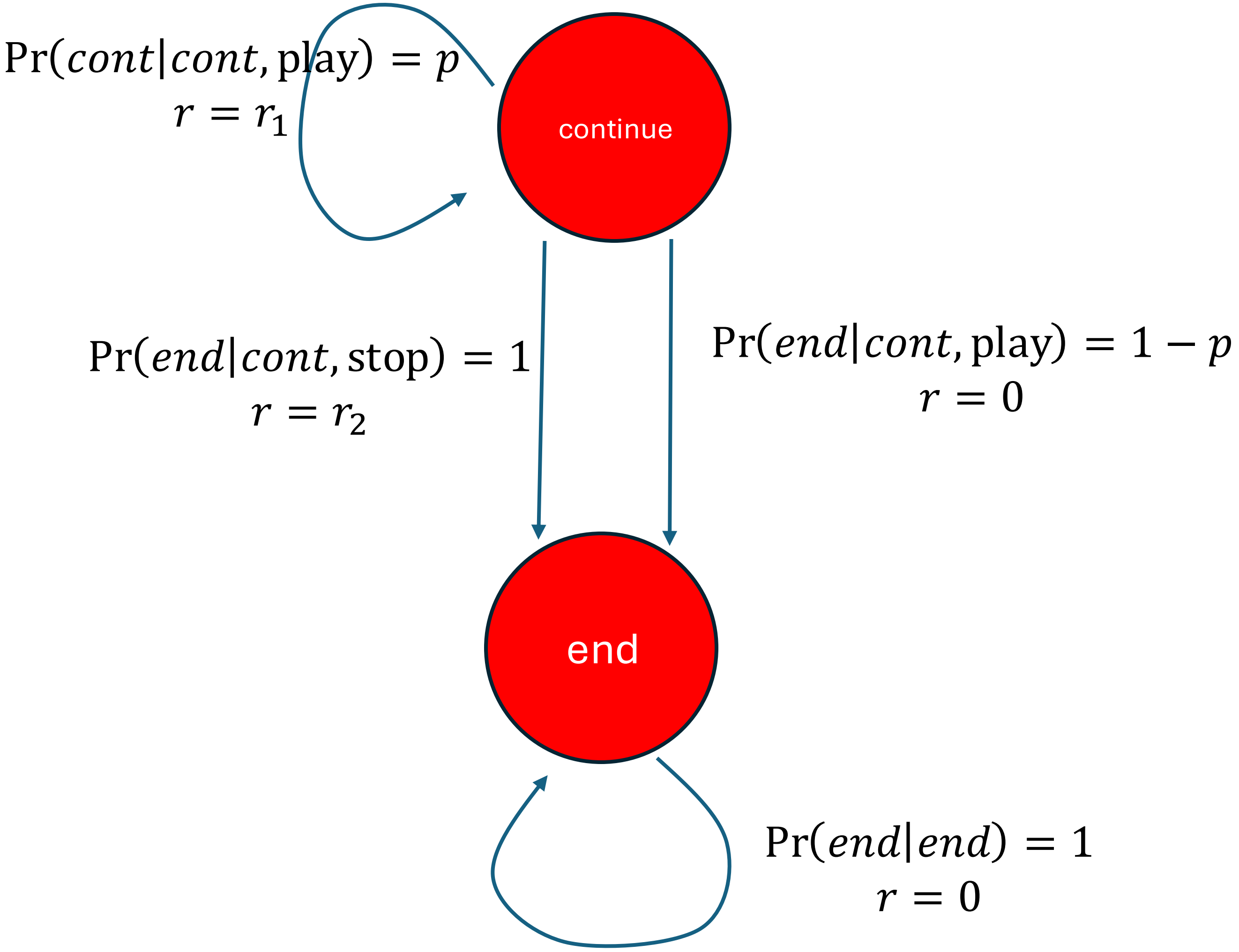
# Example 1: a coin flipping game

- A (not necessarily fair) coin is flipped every *t*. Heads with probability *p*
- Agent can either flip the coin or end the game
- Game also ends if tails
- Receives $r_1$ euros for heads, $r_2$ euros for ending the game
- MDP model: **2 states**, "continue" and "end." **2 actions**, "play" and "stop"
- End is **terminal state** ("absorbing" state)



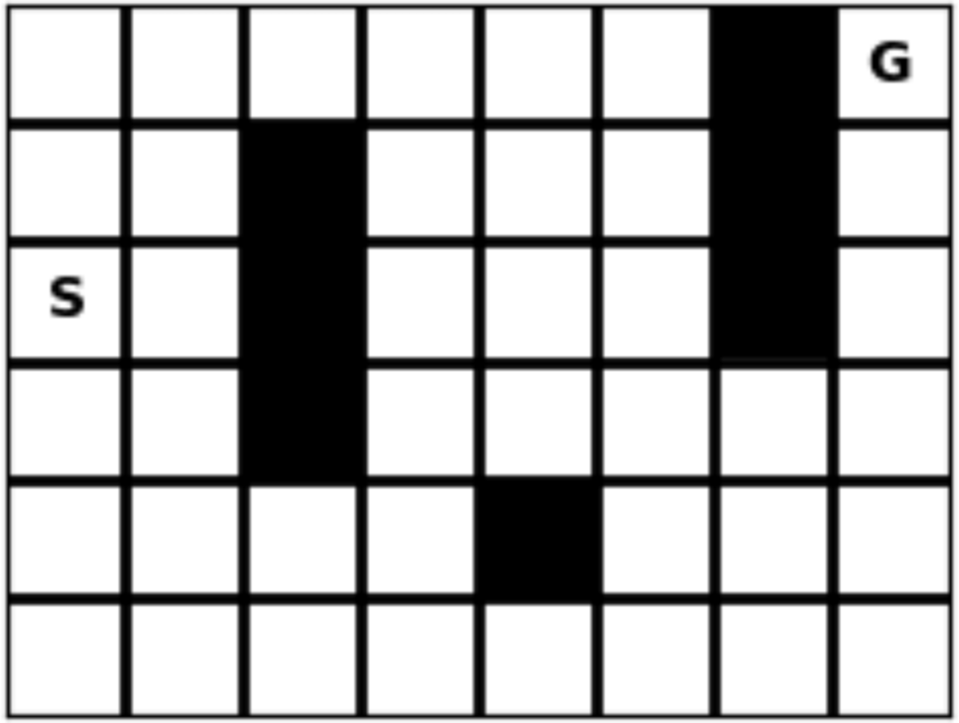
# Example 1: a coin flipping game (cont'd)

- Take $\gamma = 1$. Total expected reward for always playing is
$$E[R] = pr_1 + p^2 r_1 + p^3 r_1 + \cdots$$
$$= pr_1/(1-p)$$
- Total expected reward for stopping is
$$E[R] = r_2$$
- Intuitive that agent should only play if $r_1 > r_2(p^{-1} - 1)$



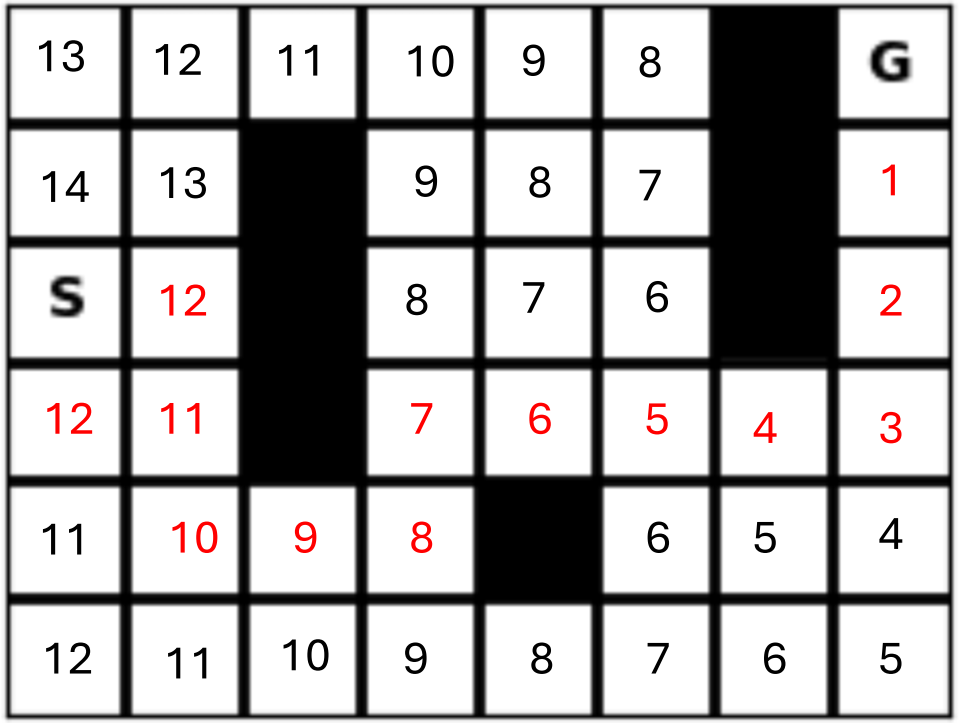
# Example 2: a maze

- State is agent's position (grid cell)
- Actions: agent moves to adjacent grid cell
- State transitions simply determined by actions
- Agent starts at $S$ and tries to get to $G$ in minimal time
- Appropriate reward?



# Example 2: a maze (cont'd)

- Notice that **problem can be solved by counting backwards from goal**
- Optimal policy: always move to square shortest distance from goal
- This idea of **exploiting recursive structure of problem** can be generalized to *all* MDPs!
- Leads to solution technique called "**dynamic programming**"



# The value function and $Q$

- Suppose agent starts in state $s$. Want to find $\pi$ to **maximize expected reward**

$$V_\pi(s) = E[\textstyle\sum_{t=0}^{\infty} \gamma^t r_t \,|a_t \sim \pi(a_t|s_t), s_0 = s] \equiv E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|s_0 = s\right]$$

- $V_\pi(s)$ called "**value function**." Expected reward starting from s under $\pi$
- Another function (will be very useful later): the *action-value* or *Q-function*

$$Q_\pi(s,a) = E_\pi\left[\textstyle\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right]$$

- Important property:

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a)$$

# The Bellman equation

- Dynamic programming works because $V_\pi$ enjoys recursive relation as follows
- Note that

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|s_0 = s\right] = \sum_{t=0}^{\infty} \gamma^t \, E_\pi[r_t|s_0 = s]$$

$$= E_\pi[r_0|s_0 = s] + \gamma \sum_{t=0}^{\infty} \gamma^t \, E_\pi[r_{t+1}|s_0 = s]$$

$$= \sum_{a \in \mathrm{A}} \pi(a|s) \sum_{s',r} \Pr(s', r|s, a) \left( r + \gamma \sum_{t=0}^{\infty} \gamma^t \, E_\pi[r_{t+1}|s_1 = s'] \right)$$

$$= \sum_{a \in \mathrm{A}} \pi(a|s) \sum_{s',r} \Pr(s', r|s, a) \, (r + \gamma V_\pi(s'))$$

# The Bellman equation

$$V_\pi(s) = \sum_{a \in \mathrm{A}} \pi(a|s) \sum_{s',r} \Pr(s', r|s, a)\,(r + \gamma V_\pi(s'))$$

- This is called the *Bellman equation* for $V_\pi$
- Let $V^*(s) = \max_\pi V_\pi(s)$ , $Q^*(s,a) = \max_\pi Q_\pi(s,a)$. Called *optimal (action-) value function*.
- $V^*$ and $Q^*$ are the (action-)value functions for optimal policy $\pi^*$ and
$$V^*(s) = \max_a Q^*(s,a)$$
- $V^*$ satisfies *Bellman optimality equation*
$$V^*(s) = \max_a \sum_{s',r} \Pr(s', r|s, a)\,(r + \gamma V^*(s'))$$
- Exercise: derive BOE. (Hint: first derive Bellman equation for $Q_\pi$)

# Aside: Hamilton-Jacobi-Bellman equation

- Consider again continuous optimal control problem

$$\dot{x} = F(x, \alpha),$$

define $V(x,t) = \min_{\alpha}\left(\phi\big(x(T)\big) + \int_0^T dt\, L(x,\alpha)\right)$

- Alternative to PMP: (theorem) $V$ is solution to

$$\partial_t V + \min_{\alpha}[\partial_x V \cdot F(x,\alpha) + L(x,\alpha)] = 0\,, V(x(T),T) = \phi(x(T))$$

- Called “Hamilton-Jacobi-Bellman” eq., **continuous analog of Bellman**
- Has natural extension to stochastic problems, whereas PMP does not
- Closely related to Hamilton-Jacobi formulation of mechanics. $V$ plays role of action functional/Hamilton’s principal function

# Dynamic programming

$$\underbrace{V^*(s)}_{\text{max expected total reward}} = \max_a \underbrace{\sum_{s',r} \Pr(s',r|s,a)}_{\text{expectation of}} (\underbrace{r}_{\text{next reward}} + \underbrace{\gamma V^*(s')}_{\text{all future expected rewards}})$$

- Can prove: unique solution if finite number of states
- Idea: somehow **solve for $V^*$. Then $\boldsymbol{\pi}^*$ is easy to compute**.
- $\pi^*$ is greedy with respect to $V^*$: take action which saturates the RHS maximum

# Value iteration

- One way to solve Bellman equation for $V^*$. Another is "policy improvement" (see Sutton and Barto 4.1-4.3)
- Theorem: if $\gamma < 1$, RHS of BOE (call it $HV^*$) is contraction operator: $\|HU - HV\|_\infty \leq \gamma \|U - V\|_\infty$ for bounded $U, V$
- This implies **iterative application of $H$ converges** to a unique fixed point $V^*$
- Algorithm: iterate for each $s$ until converged

$$V^{(n+1)}(s) = \max_a \sum_{s',r} \Pr(s', r | s, a) \left(r + \gamma V^{(n)}(s')\right)$$

- **Very expensive if state space is large**

# Can we solve Zermelo problem now?

- System state: $\mathbf{x}(t), \mathbf{u}_L(t)$
- $\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t(\mathbf{u}_L(t) + v\hat{n})$
- What about $\mathbf{u}_L(t)$ transitions?

**Case 1: stationary flow**

- Transitions are easy but…
- Back to same problem as before (instability)

**Case 2: dynamic flow**

- Need to timestep Navier-Stokes + Poisson pressure solver to evolve $\mathbf{u}_E$
- Likely *still* suffer from instability!

**And in both cases…**

- need to know flow at every position
- Have to solve the dynamic programming problem (good luck!)

N.B. MDP solution by stochastic HJB could be good approach for control of diffusing particle!

# What about olfactory search?

- What is the state of the system?
- Does the agent **have access to state**?
- Need notion of **partial observability**