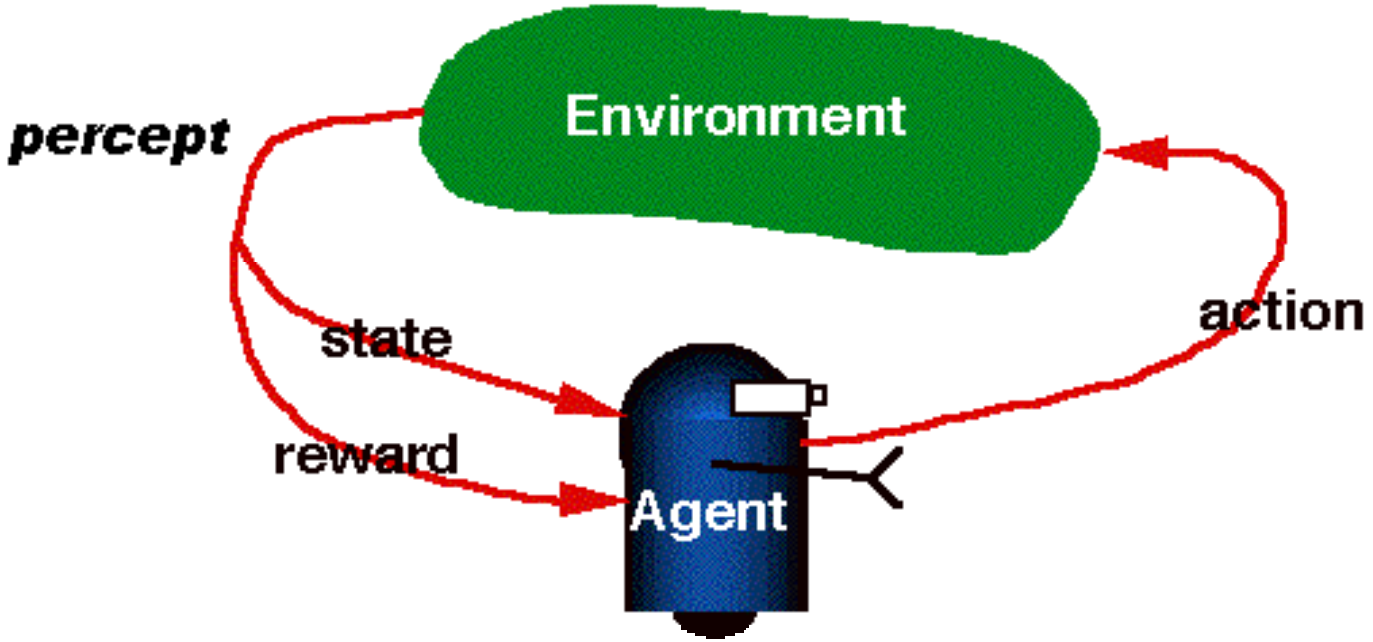
Markov decision process & Reinforcement Learning (I)

©Dr. Min Chi
mchi@ncsu.edu

AI Academy

RL is learning from interaction



Reinforcement Learning

In the case of the agent acts on its environment, it receives some evaluation of its action (reinforcement), but is not

told of which action is the correct one to achieve its goal

- Training data: (S, A, R). (State-Action-Reward)
- Develop an optimal policy (sequence of decision rules) for the learner so as to maximize its long-term reward.
- Robotics, Board game playing programs.

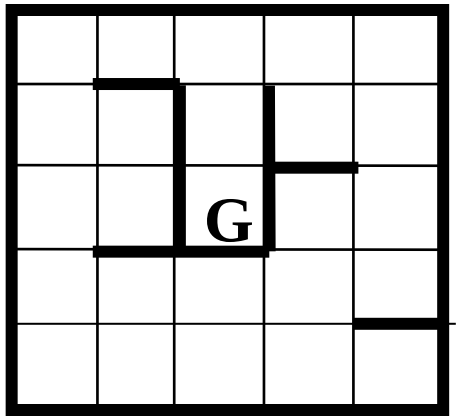
Gambling example

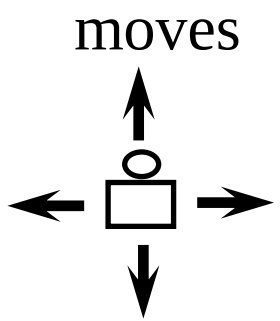
- **Game:** 3 different biased coins are tossed
  - The coin to be tossed is selected randomly from the three options and I always see which coin to play next
  - I make bets on head or tail and the bet always $1
  - If I win, I get $1, otherwise I lose my bet
- **RL model:**
  - **Input:** X – a coin chosen for the next toss,
  - **Action:** A – choice of head or tail,
  - **Reinforcements:** {1, -1}
- **A policy** $\pi : X \rightarrow A$

  Example: $\pi :$ $\left| \begin{array}{l} \textbf{Coin1} \rightarrow \textbf{\textit{head}} \\ \textbf{Coin2} \rightarrow \textbf{\textit{tail}} \\ \textbf{Coin3} \rightarrow \textbf{\textit{head}} \end{array} \right|$

Agent navigation example

- **The RL model:**
  - **Input:** X – position of an agent
  - **Output:** A –a move
  - **Reinforcements:** R
    - -1 for each move
    - +100 for reaching the goal
  - **A policy:** $\pi : X \rightarrow A$





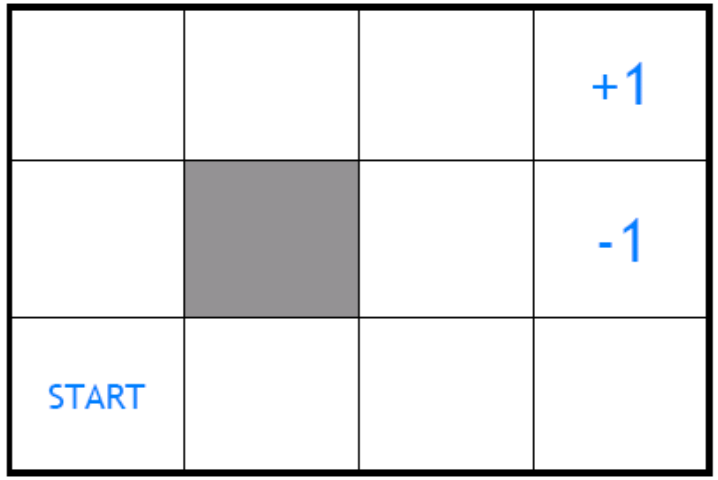
π :

Position 1 ⟶ ***right***
Position 2 ⟶ ***right***
…
Position 20 ⟶ ***left***

- **Goal: find the policy maximizing future expected rewards**

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were NOT deterministic?

What is special about RL?

- RL is learning how to map states to actions, so as to maximize a numerical reward over time.

- Unlike other forms of learning, it is a multi-stage decision-making process (often Markovian).

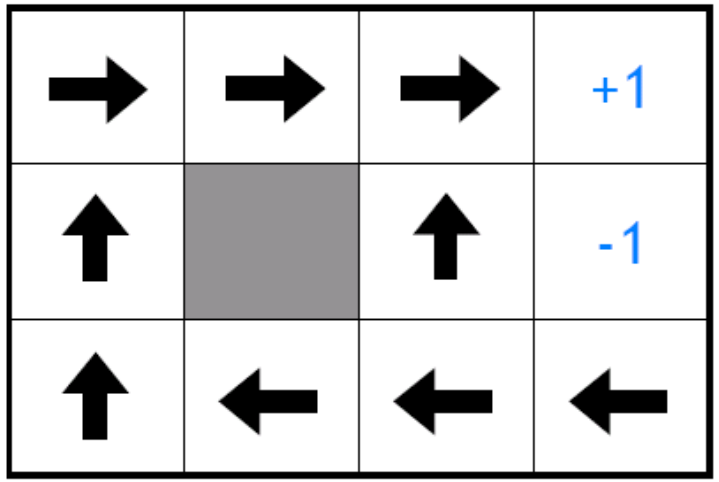
- An RL agent must learn by trial-and-error. (Not entirely supervised, but interactive)

- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect).

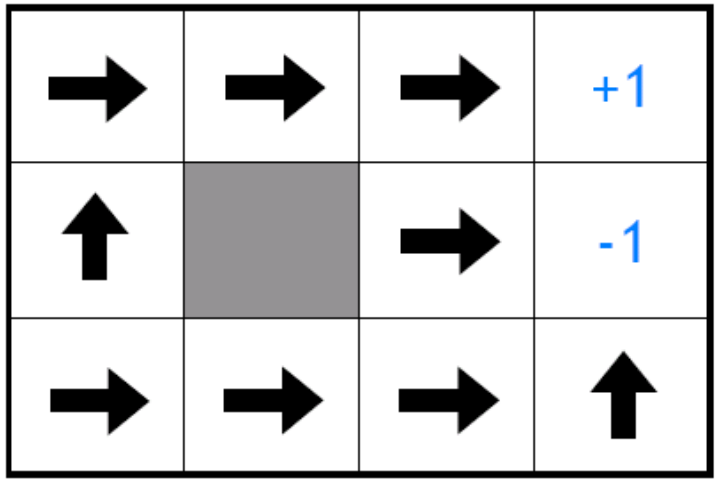
Elements of RL

- A **policy**
  - A map from state space to action space.
  - May be stochastic.
- A **reward function**
  - It maps each state (or, state-action pair) to a real number, called reward.
- A **value function**
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

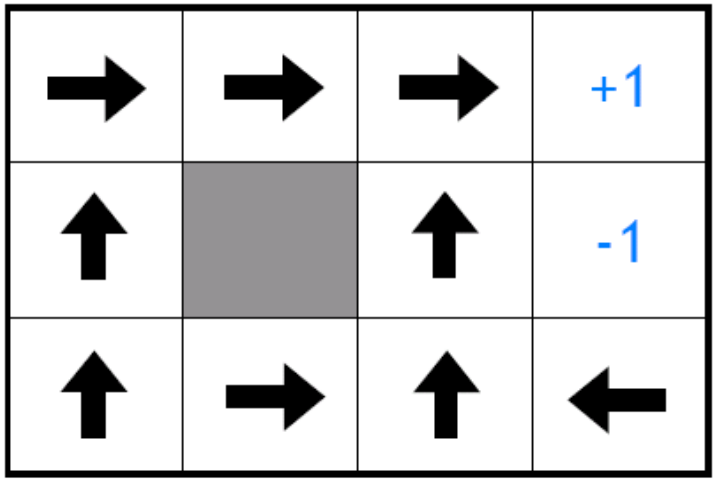
Policy



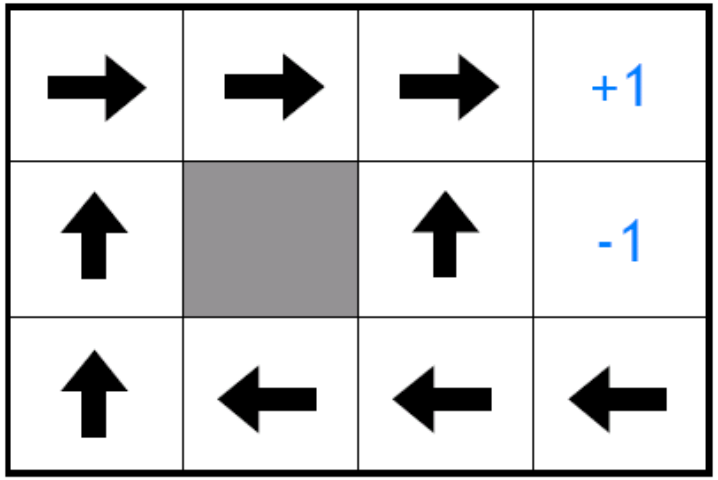
Reward for each step -2

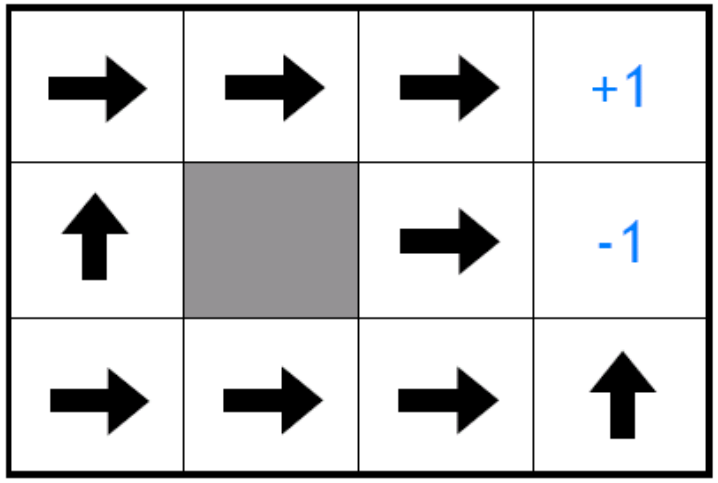


Reward for each step: -0.1

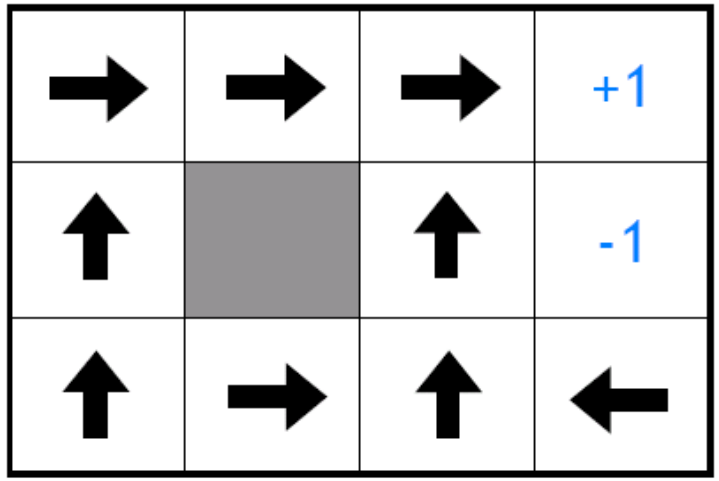


Reward for each step: -0.04

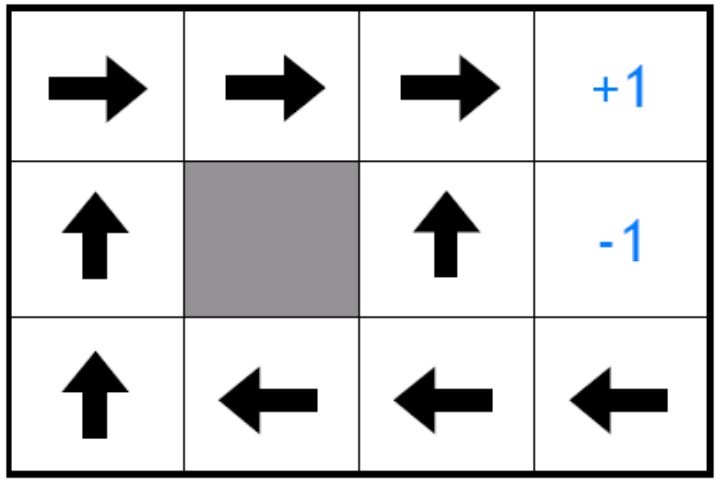




Each Step = -2



Each Step = -0.1



Each Step = -0.04

The Precise Goal

- To find a policy that maximizes the Value function.
  - transitions and rewards usually not available
- There are different approaches to achieve this goal in various situations.
- Value iteration and Policy iteration are two more classic approaches to this problem. But essentially both are dynamic programming.
- Q-learning is a more general approaches to this problem. Essentially it is a temporal-difference method.

RL with immediate rewards

- **Problem:** In the RL framework we do not know $R(\mathbf{x}, a)$
  - The expected reward for performing action a at input $\mathbf{x}$
- **Solution:**
  - For each input $\mathbf{x}$ try different actions a
  - Estimate $R(\mathbf{x}, a)$ using the average of observed rewards

$$\tilde{R}(\mathbf{x}, a) = \frac{1}{N_{x,a}} \sum_{i=1}^{N_{x,a}} r_i^{x,a}$$

  - Action choice $\pi(\mathbf{x}) = \arg\max_a \tilde{R}(\mathbf{x}, a)$

RL with immediate rewards

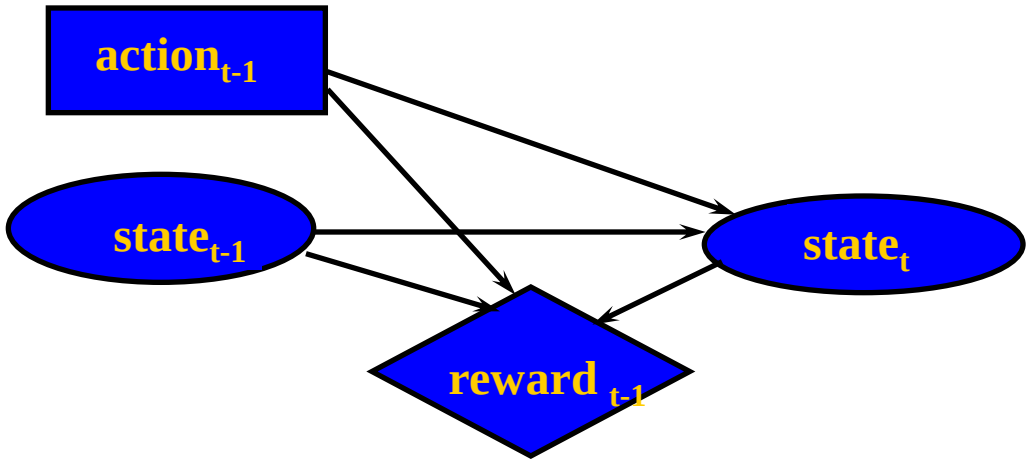
- **On-line (stochastic approximation)**
  - An alternative way to estimate $R(\mathbf{x}, a)$
- **Idea:**
  - choose action a for input $\mathbf{x}$ and observe a reward $r^{x,a}$
  - Update an estimate

$$\tilde{R}(\mathbf{x}, a) \leftarrow (1-\alpha)\tilde{R}(\mathbf{x}, a) + \alpha r^{x,a}$$

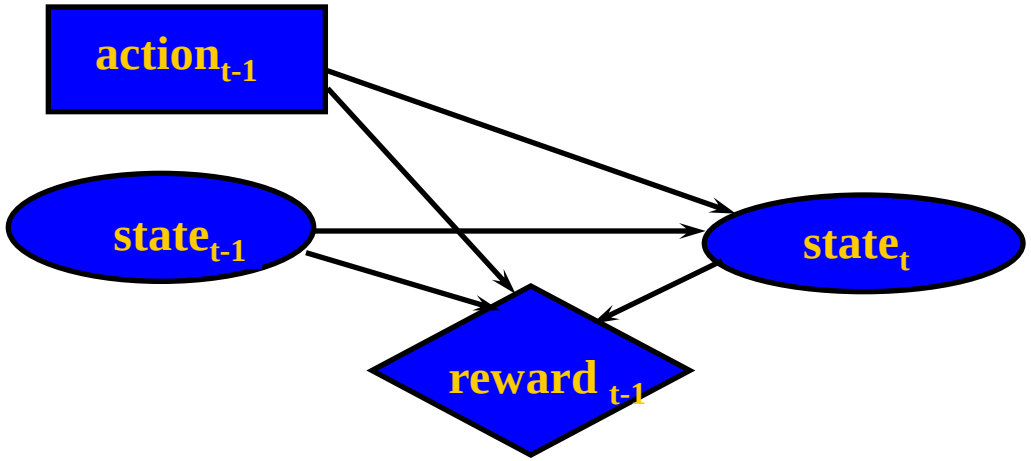
α - a learning rate

Learning with delayed rewards

- Actions, in addition to immediate rewards affect the next state of the environment and thus indirectly also future rewards
- We need a model to represent environment changes
- The model we use is called **Markov decision process (MDP)**
  - Frequently used in AI, OR, control theory
  - **Markov assumption**: next state depends on the previous state and action, and not states (actions) in the past



Markov decision process



Formal definition: 4-tuple (S, A, T, R)

• **A set of states** S (X)	locations of a robot
• **A set of actions** A	move actions
• **Transition model** $S \times A \times S \rightarrow [0,1]$	where can I get with different moves
• **Reward model** $S \times A \times S \rightarrow \Re$	reward/cost for a transition

The dynamics of an MDP

- We start in some state s_0, and get to choose some action $a_0 \in A$
- As a result of our choice, the state of the MDP randomly transitions to some successor state s_1, drawn according to $s_1 \sim P_{s0a0}$
- Then, we get to pick another action a_1
- …

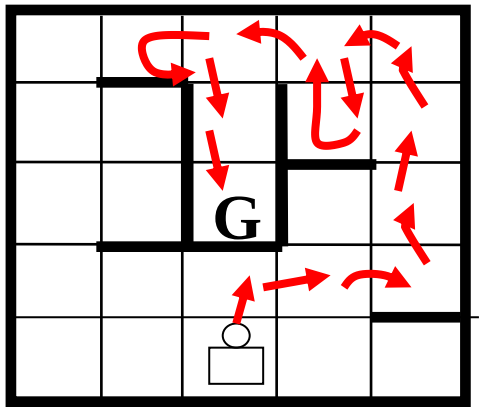
$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

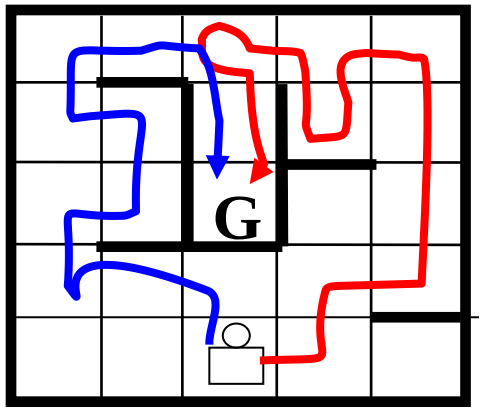
MDP problem

- We want to find the best policy $\pi^* : S \rightarrow A$
  - **Value function** (V) for a policy, quantifies the goodness of a policy through, e.g. infinite horizon, discounted model

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

It: 1. combines future rewards over a trajectory
2. combines rewards for multiple trajectories (through expectation-based measures)





Value of a policy for MDP

- Assume a fixed policy $\pi : S \rightarrow A$
- How to compute the value of a policy under infinite horizon discounted model?

Fixed point equation:

$$V^{\pi}(s) = \underbrace{R(s, \pi(s))}_{\text{expected one step reward for the first action}} + \underbrace{\gamma \sum_{s' \in S} P(s' | s, \pi(s)) V^{\pi}(s')}_{\text{expected discounted reward for following the policy for the rest of the steps}}$$

$$\mathbf{v} = \mathbf{r} + \mathbf{U}\mathbf{v} \quad \Longrightarrow \quad \mathbf{v} = (\mathbf{I} - \mathbf{U})^{-1}\mathbf{r}$$

– For a finite state space– we get a set of linear equations

Optimal policy

- **The value of the State**

$$V^*(s) = \max_{a \in A} \Big[\underbrace{R(s,a)}_{\text{expected one step reward for the first action}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s,a) V^*(s')}_{\text{expected discounted reward for following the opt. policy for the rest of the steps}} \Big]$$

- **The optimal policy:** $\pi^* : S \rightarrow A$

$$\pi^*(s) = \arg\max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \Big]$$

Computing optimal policy

Dynamic programming. Value iteration:

- computes the optimal value function first then the policy
- iterative approximation
- converges to the optimal value function

Value iteration (ε)

initialize $\mathbf{V}$ **;;** V is vector of values for all states

repeat

set $\mathbf{V}' \leftarrow \mathbf{V}$

set $\mathbf{V} \leftarrow \mathbf{HV}$

until $\|\mathbf{V}' - \mathbf{V}\|_\infty \leq \varepsilon$

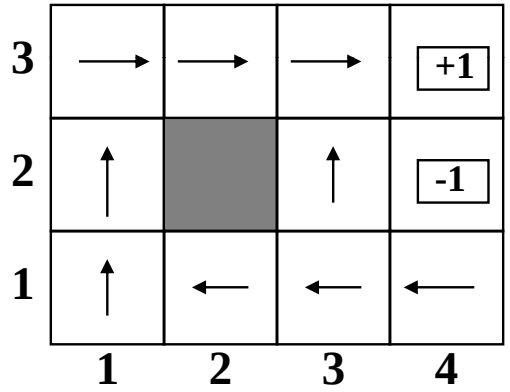
output $\pi^*(s) = \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s') \right]$

The Grid World

M = 0.8 in direction you want to go
0.2 in perpendicular — **0.1 left**, **0.1 right**

Policy: mapping from states to actions

An optimal policy for the stochastic environment:



utilities of states:

3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Environment — **Observable (accessible): percept identifies the state**
Partially observable

***Markov property*: Transition probabilities depend on state only, not on the path to the state.**
Markov decision problem (MDP).
Partially observable MDP (POMDP): percepts does not have enough info to identify transition probabilities.

Algorithm 2: Policy Iteration

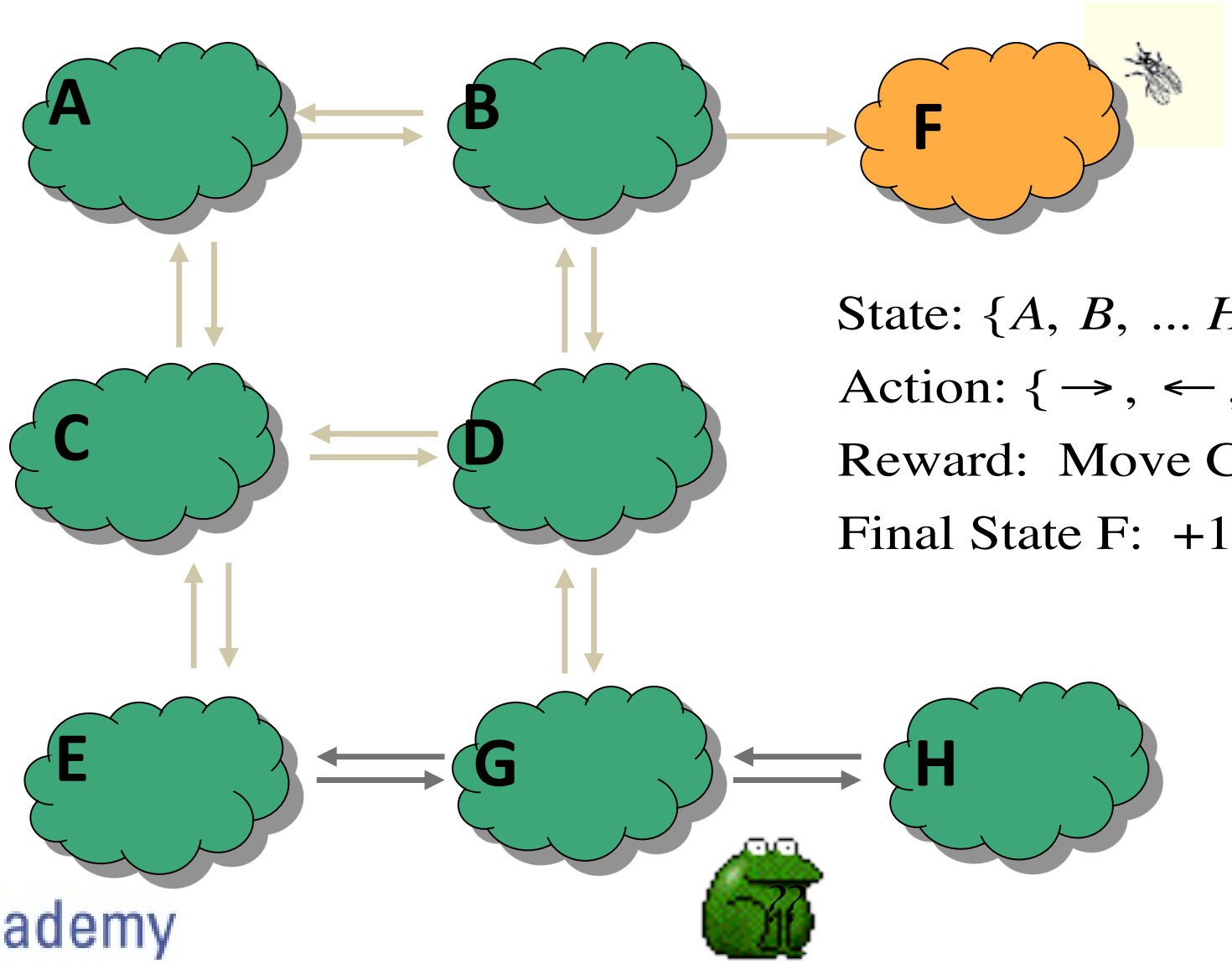
1. $\pi_0(s) = random(a \in \mathrm{A})$
2. Update the value of each state using neighboring states:

$$V^{\pi}(s) = \underbrace{R(s, \pi(s))}_{\text{expected one step reward for the first action}} + \underbrace{\gamma \sum_{s' \in S} P(s' | s, \pi(s)) V^{\pi}(s')}_{\text{expected discounted reward for following the policy for the rest of the steps}}$$

3. Update the policy:

$$\pi^*(s) = \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right]$$

What's the best policy to get the fly?



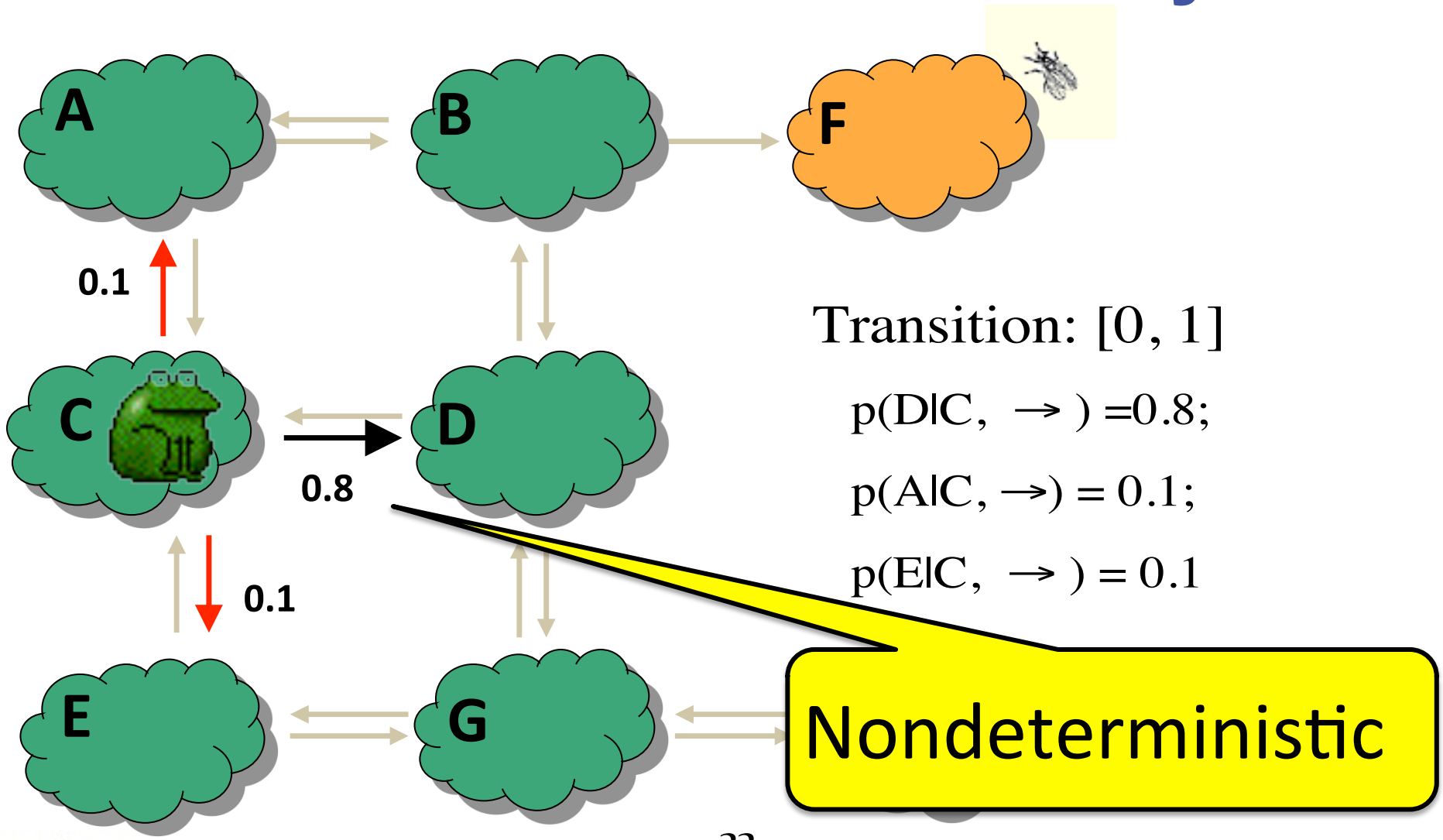
State: $\{A, B, \ldots H\}$

Action: $\{\rightarrow, \leftarrow, \uparrow, \downarrow\}$

Reward: Move Cost = -1;

Final State F: +100

Transition Probability



Transition: [0, 1]

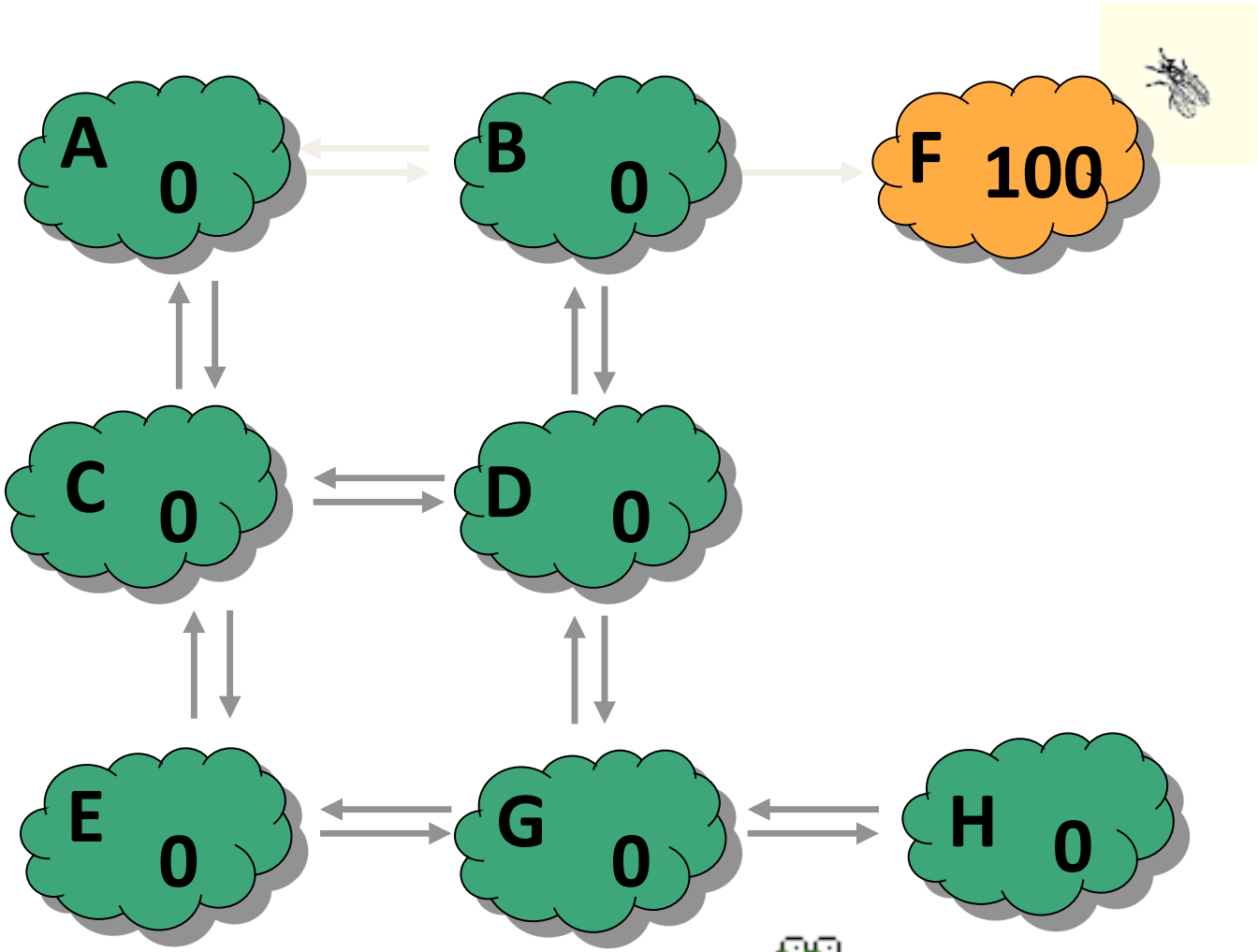
$p(D|C, \rightarrow) = 0.8;$

$p(A|C, \rightarrow) = 0.1;$

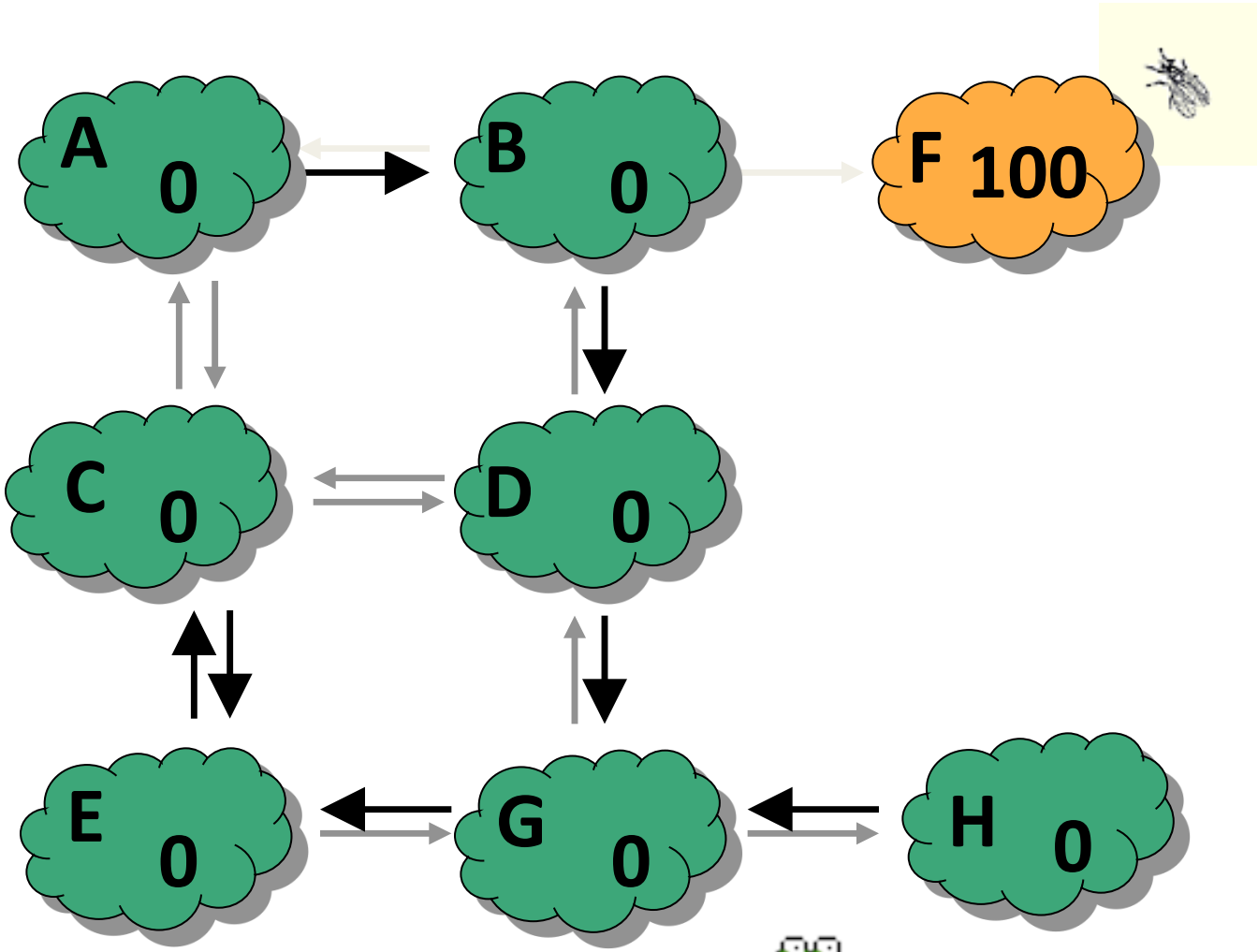
$p(E|C, \rightarrow) = 0.1$

Nondeterministic

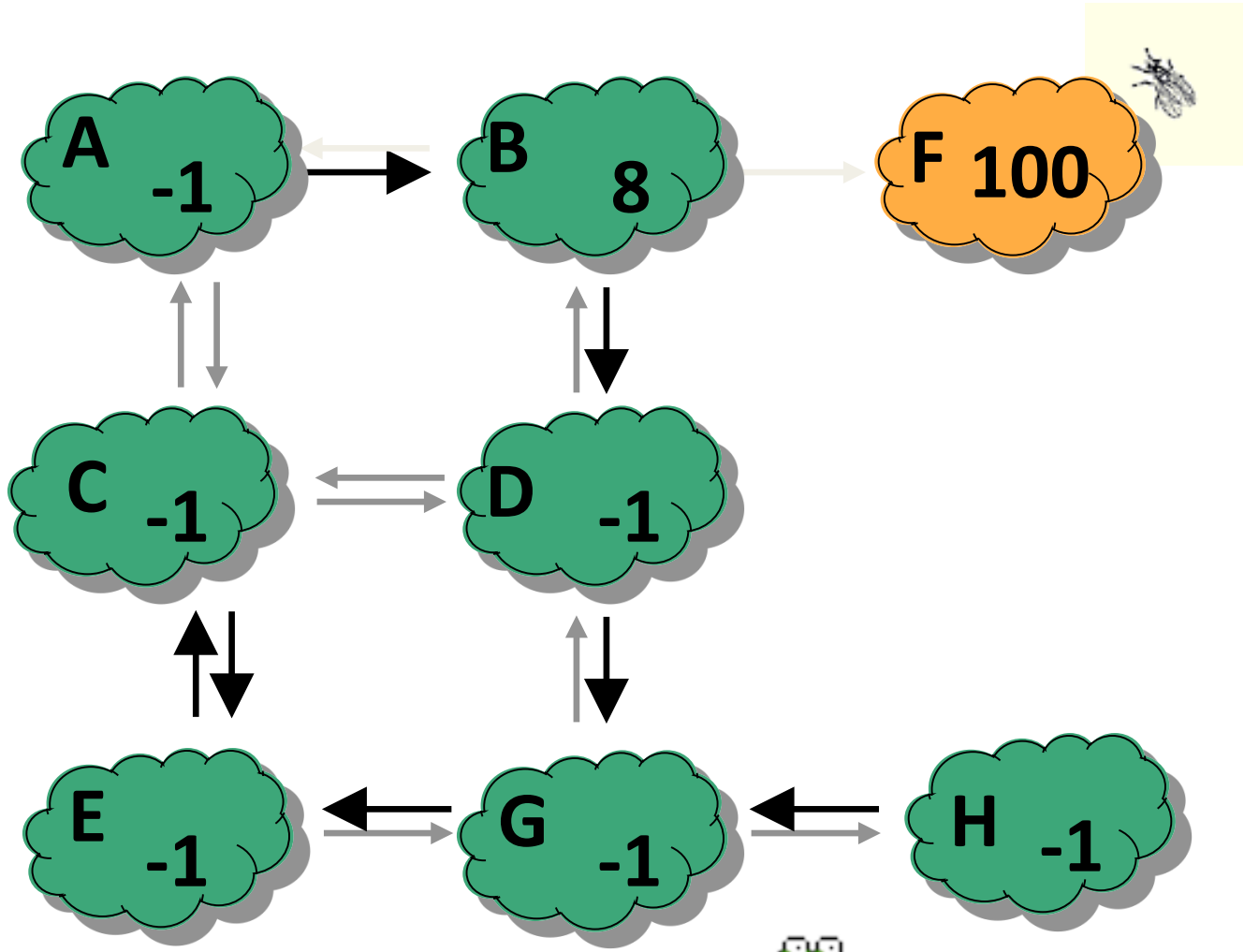
Initial Values



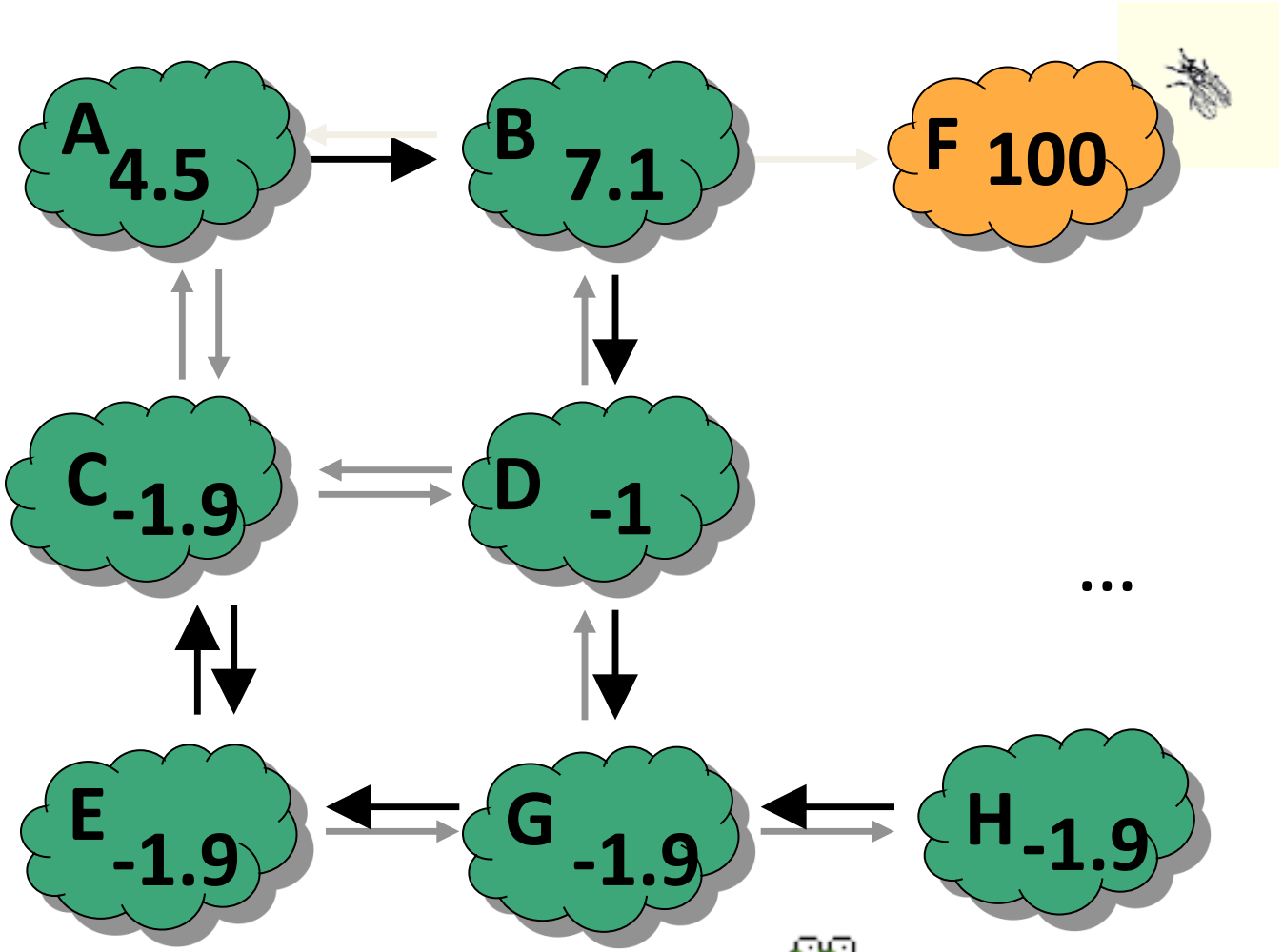
Step 1: Random Initial Policy $\pi(0)$



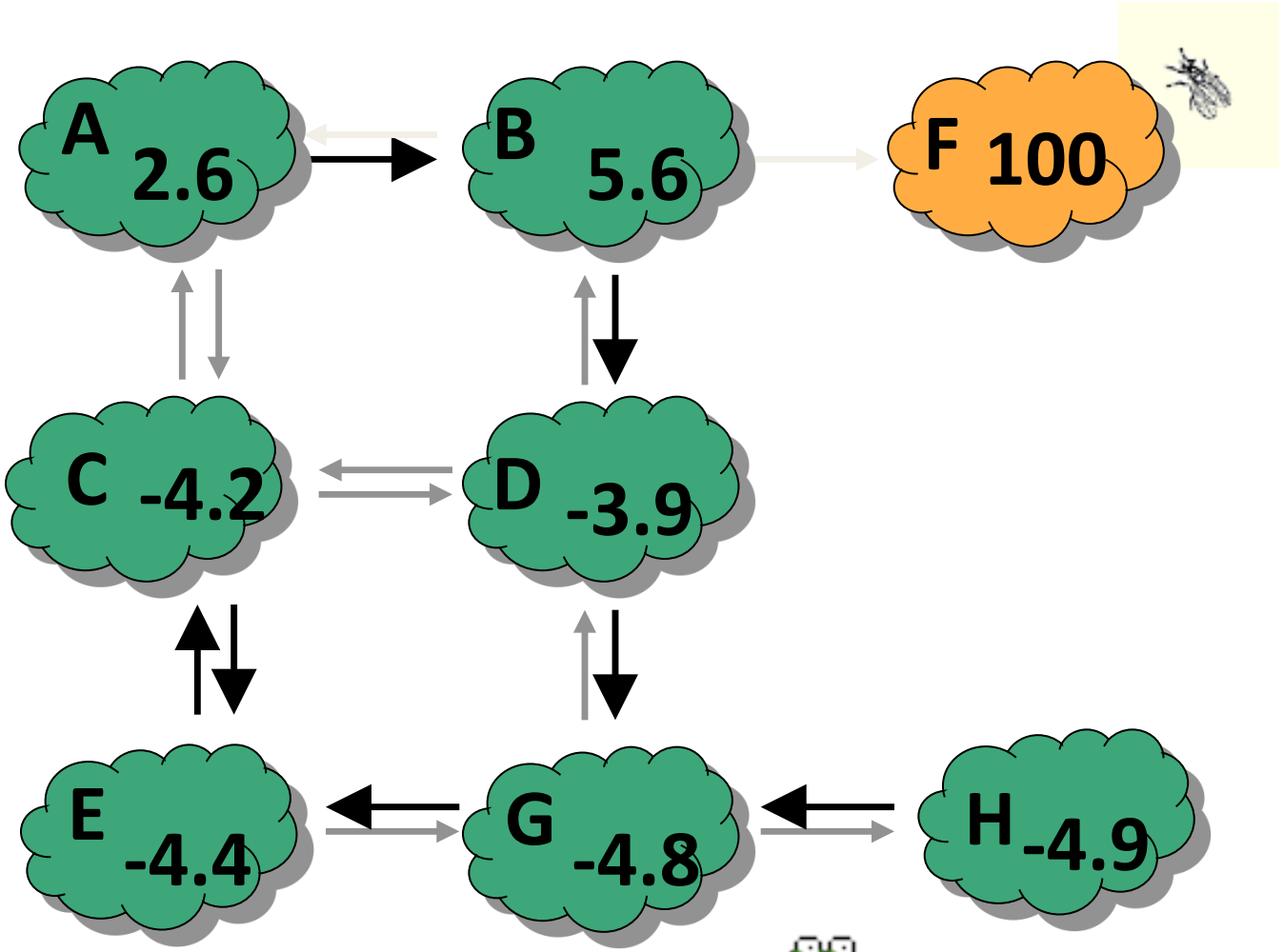
Step 2: updating V(S) (Rd 1)



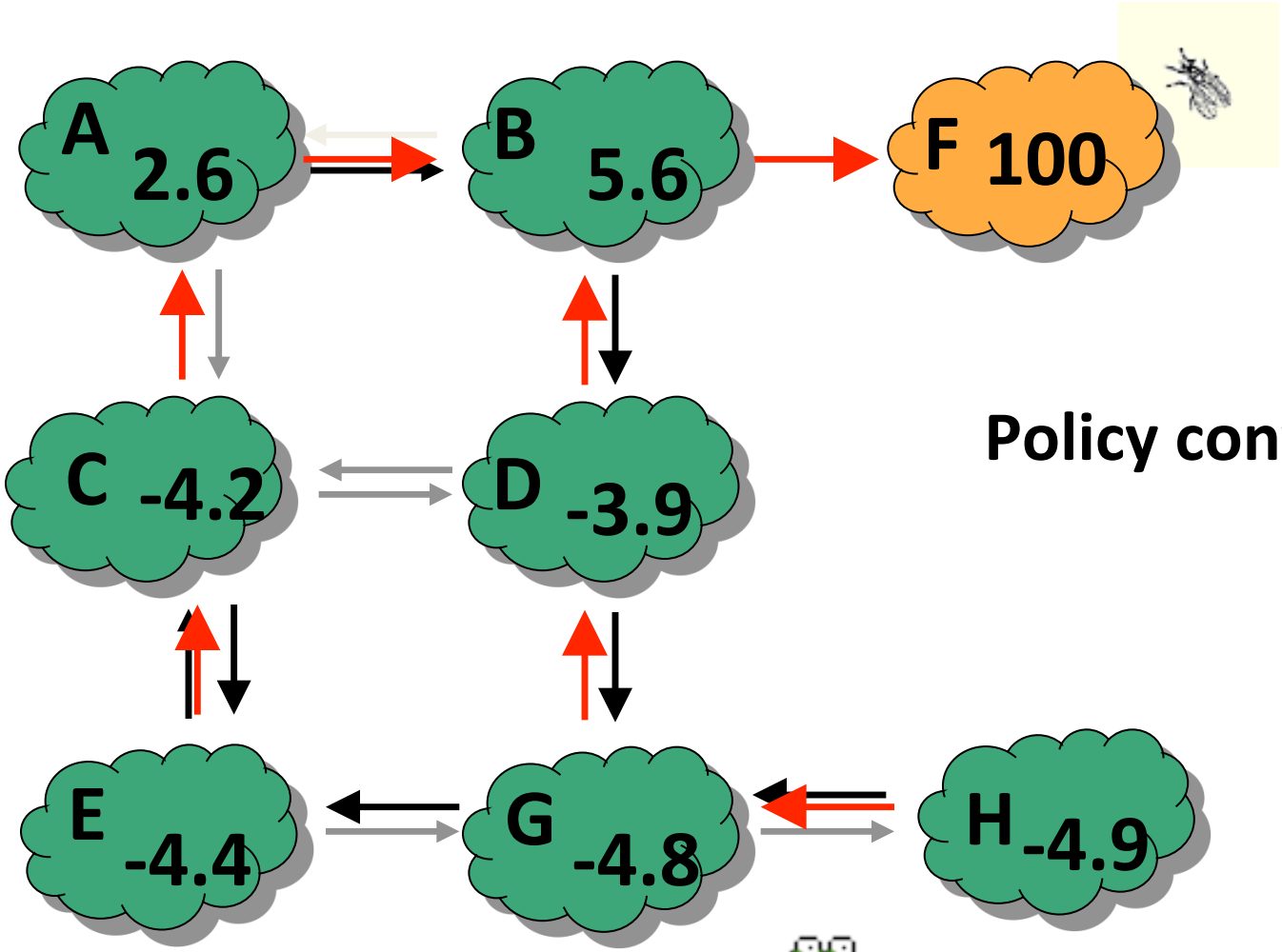
Step 2: updating V(S) (Rd 2)



Step 2: updating V(S) (Rd 7)

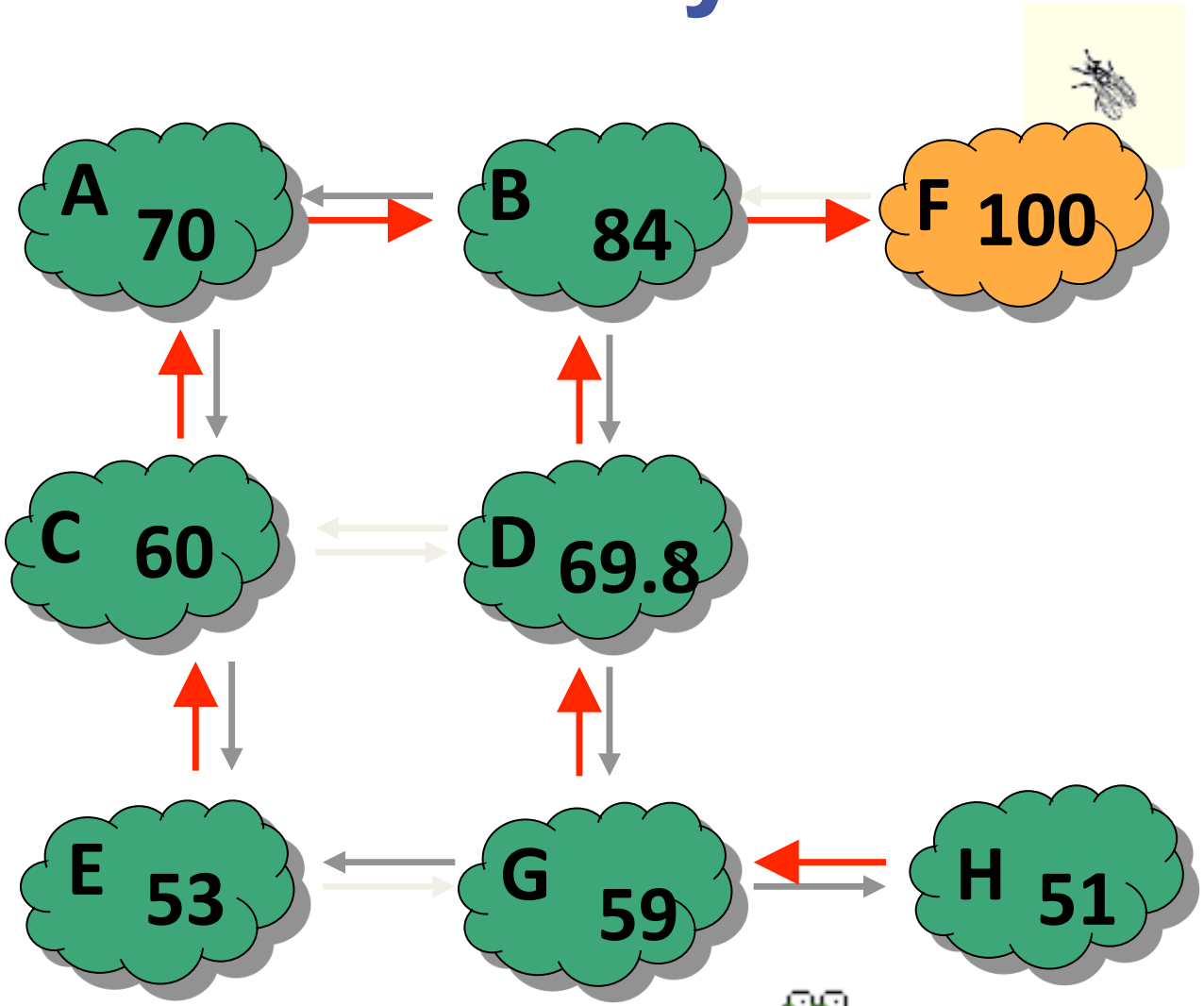


Step 3: updating π(s)

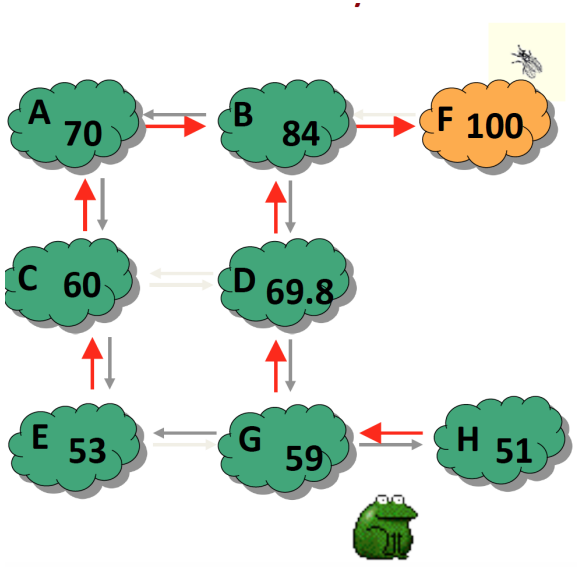


Policy converged.

Final Policy and V-values



ECR for Evaluating a Policy



Expected Culmulative Reward (ECR)

$$ECR_\pi = \sum_{i=1}^{n} \frac{N_i}{N_1 + \cdots + N_n} \times V(s_i)$$

** Where $\mathbf{S_1, \cdots, S_n}$ is the set of all **starting states** and $V(S_i)$ is the V-values for state S_i ;

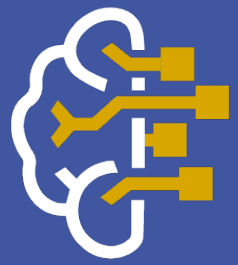
** $\mathbf{N_i}$ is the number of times that S_i appears **as a start state in the model** and it is normalized.

Stay Connected

Dr. Min Chi

Associate Professor
mchi@ncsu.edu
(919) 515-7825

AI Academy



AI Academy

go.ncsu.edu/aiacademy

NC STATE UNIVERSITY