Introduction to Reinforcement Learning
Lecture 2

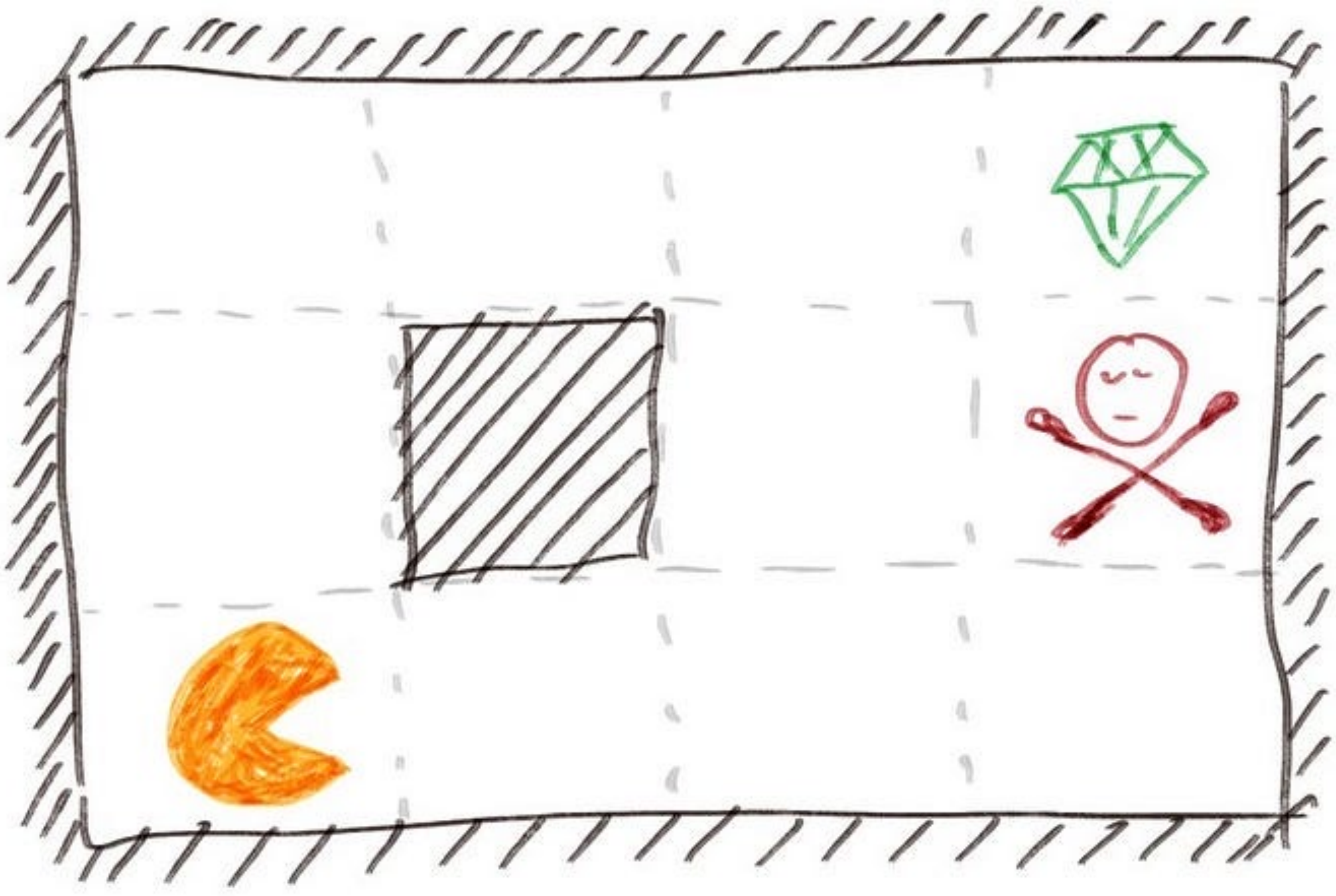
Dr. Apurva Narayan
http://anarayan.com

MARKOV DECISION PROCESSES

Markov Decision Processes

- Markov Processes
- Markov Reward Processes
- Markov Decision Processes
- Extension to MDPs

Introduction to MDPs



MDPs

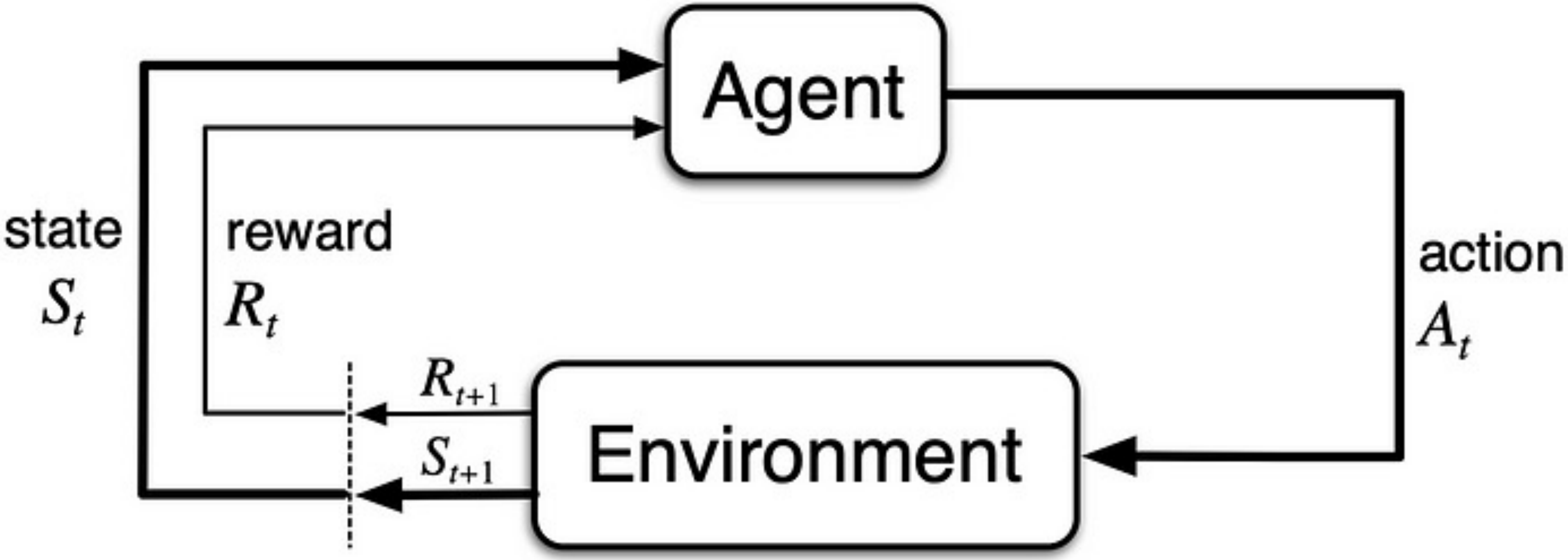
Markov Decision Process(MDP) is a mathematical framework for sequential decision and a dynamic optimization method in a stochastic discrete control process.

***Andrei Markov*.**

MDPs

> *a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process, or MDP, and consists of a set of states (with an initial state s_0); a set ACTIONS(s) of actions in each state; a transition model $P(s' \mid s, a)$; and a reward function $R(s)$.*

MDPs

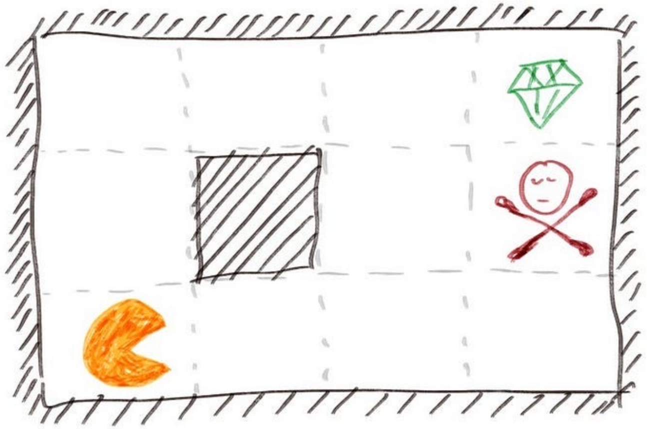


MDPs

$$(S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots)$$

The goal of the MDP process is for the Agent to maximize the total long-term reward over time from its Environment by choosing the right action for a specific state. MDP focuses on maximizing not immediate but cumulative reward in the long run.

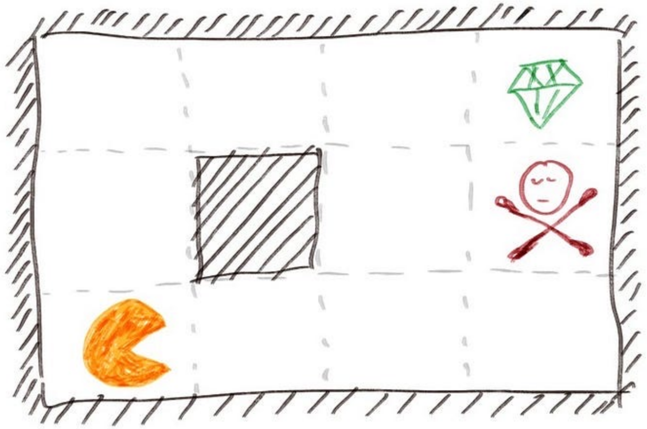
MDPs – Defining the problem



Environment refers to what the world looks like and the rules of the world. In our example, environment includes how big our world is, where the walls are placed, what happens if we bump into a wall, where is the diamond and if we reach there how much points we get, where is the poison and if we reach there how much points we lose, and so on.

Agent refers to the robot or the Pac-Man or whatever we put inside this world that looks around for possible moves, thinks about the pros and cons of possible moves, makes a decision of the next move, and executes the move.

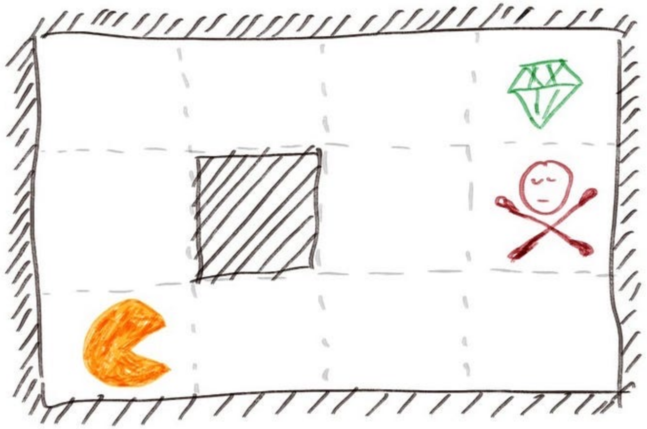
MDPs – Defining the problem



State in MDP refers to the state of our agent, not state of our environment. In MDP the environment is given and does not change. In contrast, our agent can change state from previous move to current move.

In our example, our agent has 12 states which represent the 12 squares that our agent can possibly be in. Technically our agent is not able to be in the black square, but for simplicity we still count the black square as a state. In MDP we use s to denote state and in our example we call them s = 0, 1, 2, …, 11.

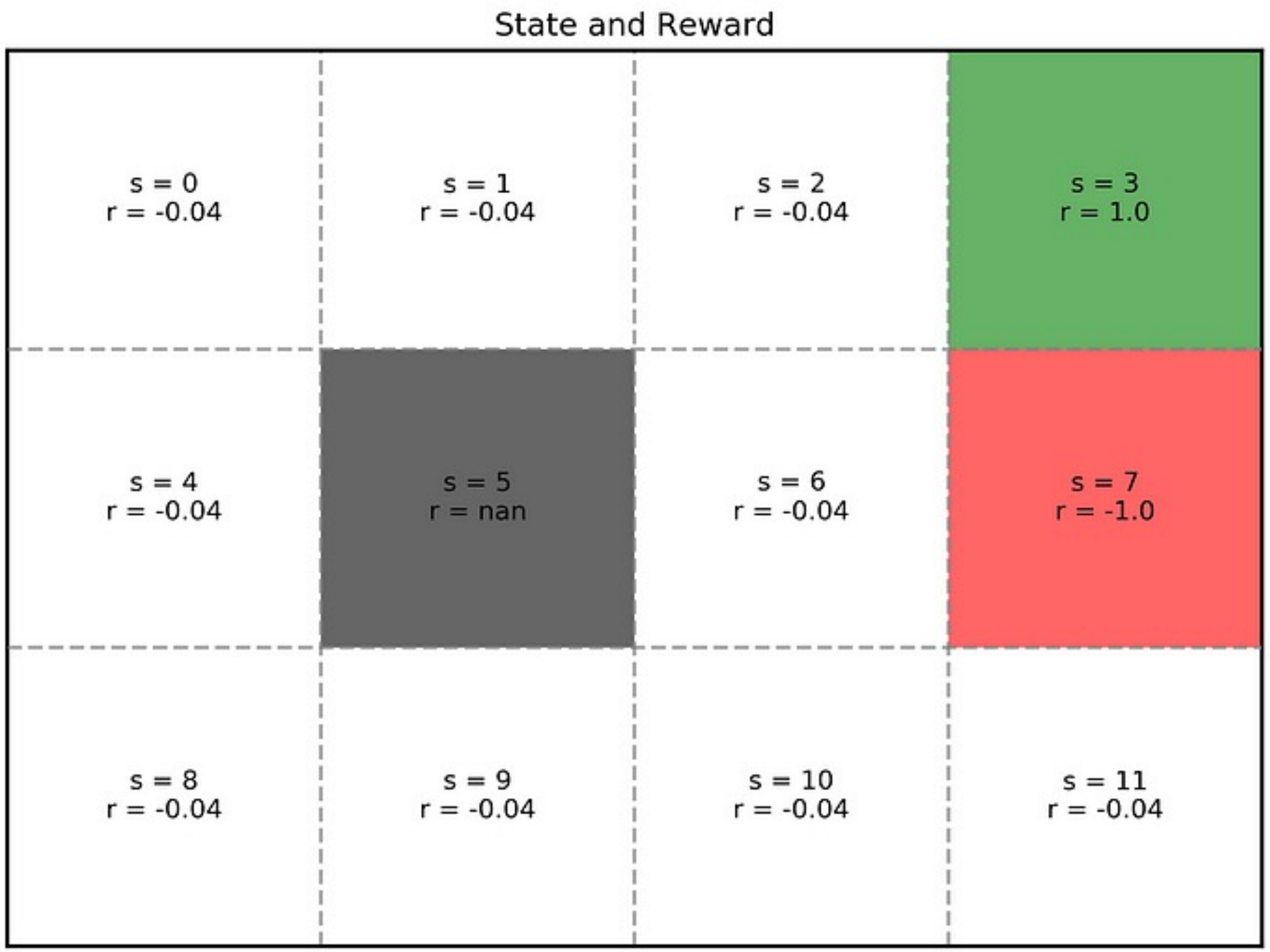
MDPs – Defining the problem



Action refers to the moves can be taken by our agent in each state. The set of possible actions is not necessarily the same for each state.

In our example, we have four possible actions $A(s)$ = {up, down, left, and right} for every state s. We use a to denote actions.

MDP



Rewards are additive!

Transition Model – Deterministic Case

$$P(s' = 0 | s = 10, a = \mathrm{UP}) = 0$$
$$P(s' = 1 | s = 10, a = \mathrm{UP}) = 0$$
$$P(s' = 2 | s = 10, a = \mathrm{UP}) = 0$$
$$\cdots$$
$$P(s' = 6 | s = 10, a = \mathrm{UP}) = 1$$
$$P(s' = 7 | s = 10, a = \mathrm{UP}) = 0$$
$$P(s' = 8 | s = 10, a = \mathrm{UP}) = 0$$
$$\cdots$$
$$P(s' = 9 | s = 10, a = \mathrm{UP}) = 0$$
$$P(s' = 10 | s = 10, a = \mathrm{UP}) = 0$$
$$P(s' = 11 | s = 10, a = \mathrm{UP}) = 0$$

As a probability distribution, the sum of $P(s' | s = 10, a = \mathrm{UP})$ of all 12 s' always equals to 1.

Transition Model – Stochastic Case

$$P(s' = 0 | s = 10, a = \text{UP}) = 0$$
$$P(s' = 1 | s = 10, a = \text{UP}) = 0$$
$$P(s' = 2 | s = 10, a = \text{UP}) = 0$$
$$\cdots$$
$$P(s' = 6 | s = 10, a = \text{UP}) = 0.8$$
$$P(s' = 7 | s = 10, a = \text{UP}) = 0$$
$$P(s' = 8 | s = 10, a = \text{UP}) = 0$$
$$\cdots$$
$$P(s' = 9 | s = 10, a = \text{UP}) = 0.1$$
$$P(s' = 10 | s = 10, a = \text{UP}) = 0$$
$$P(s' = 11 | s = 10, a = \text{UP}) = 0.1$$

This is the transition model when s = 10 and a = UP. Now for every pair of state s and action a, we can write out the probability of arriving at each new state like this.

Combining all these together, we get the transition model P.

We have 12 possible states and 4 possible actions. Therefore, the dimension of our transitional model P is 12×4×12, with a total of 576 probability values.

Fully Observable **Sequential** **Markovian**

Memoryless

Markov Property

"The future is independent of the past given the present"

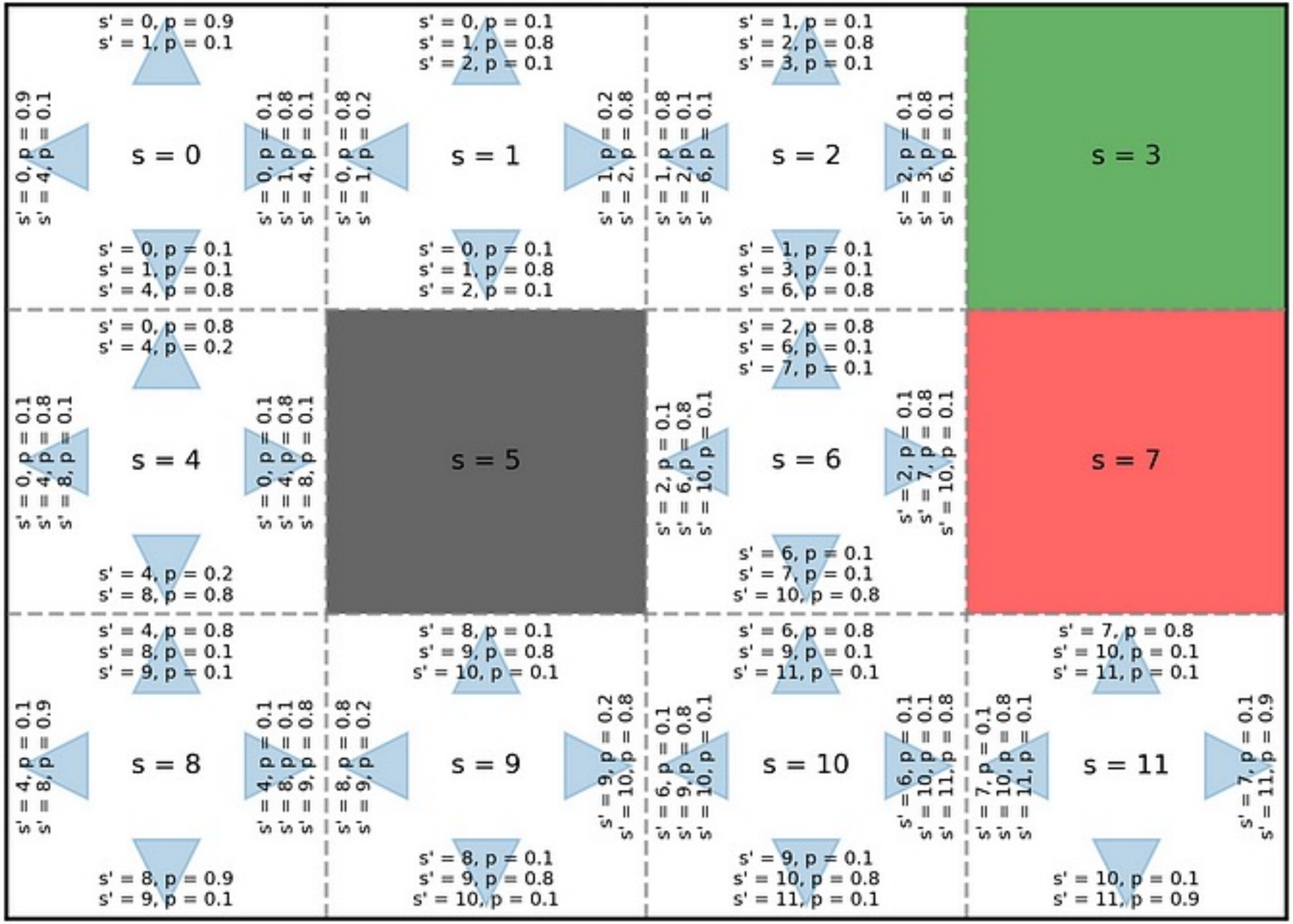
Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

Transitional model



State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} \quad = \textit{from} \quad \begin{array}{c} \textit{to} \\ \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

where each row of the matrix sums to 1.

Markov Process

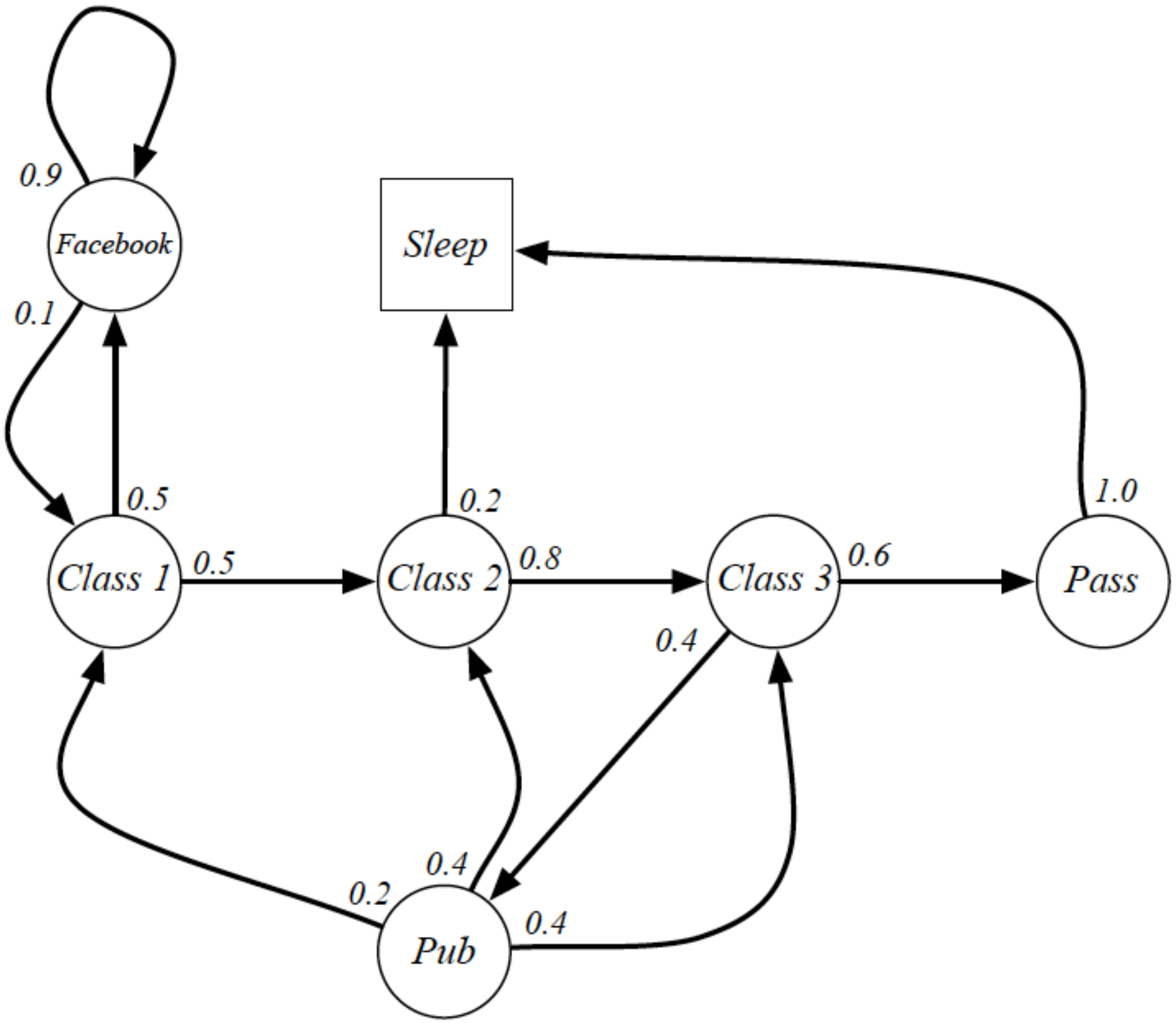
A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

Definition

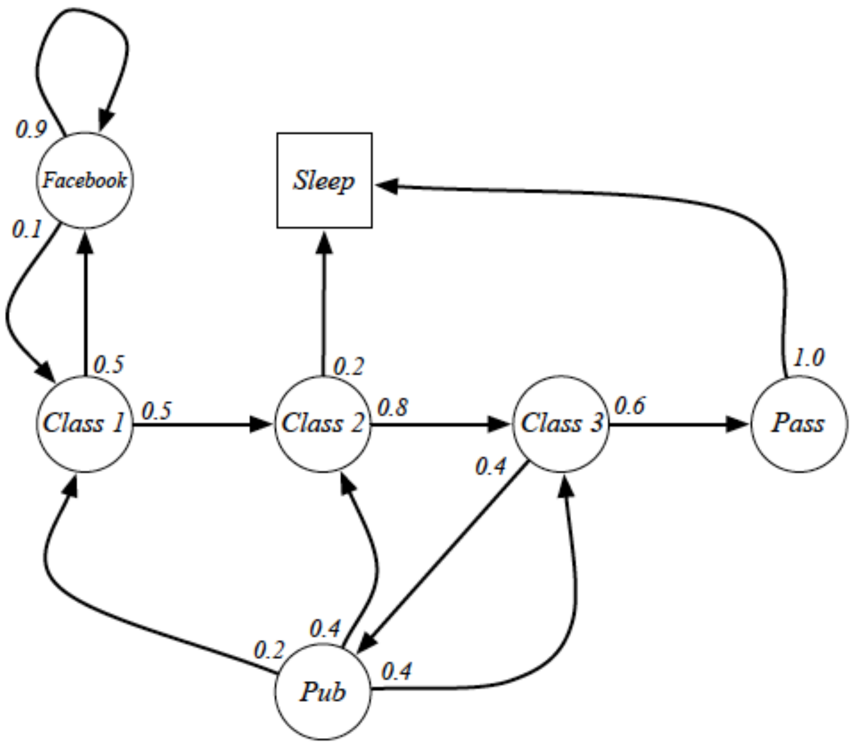
A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

Markov Process Example



Markov Process Example

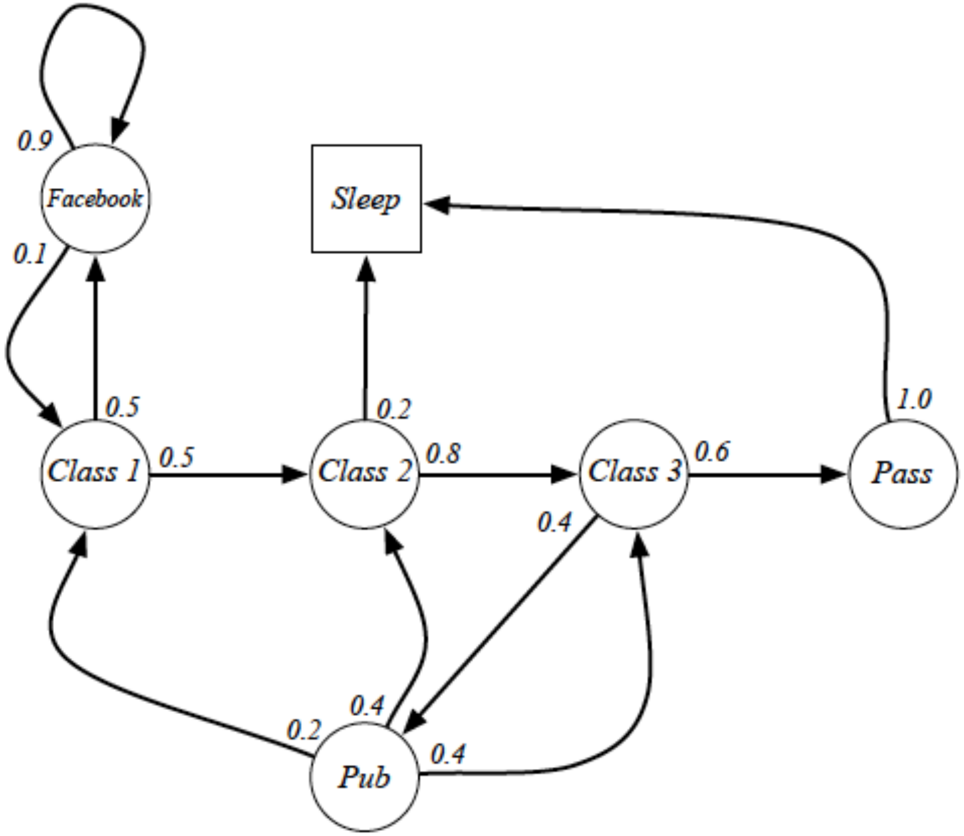


Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, \ldots, S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Markov Process Example



$$\mathcal{P} = \begin{array}{cc} & \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{array}{c} C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} & \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Markov Reward Process

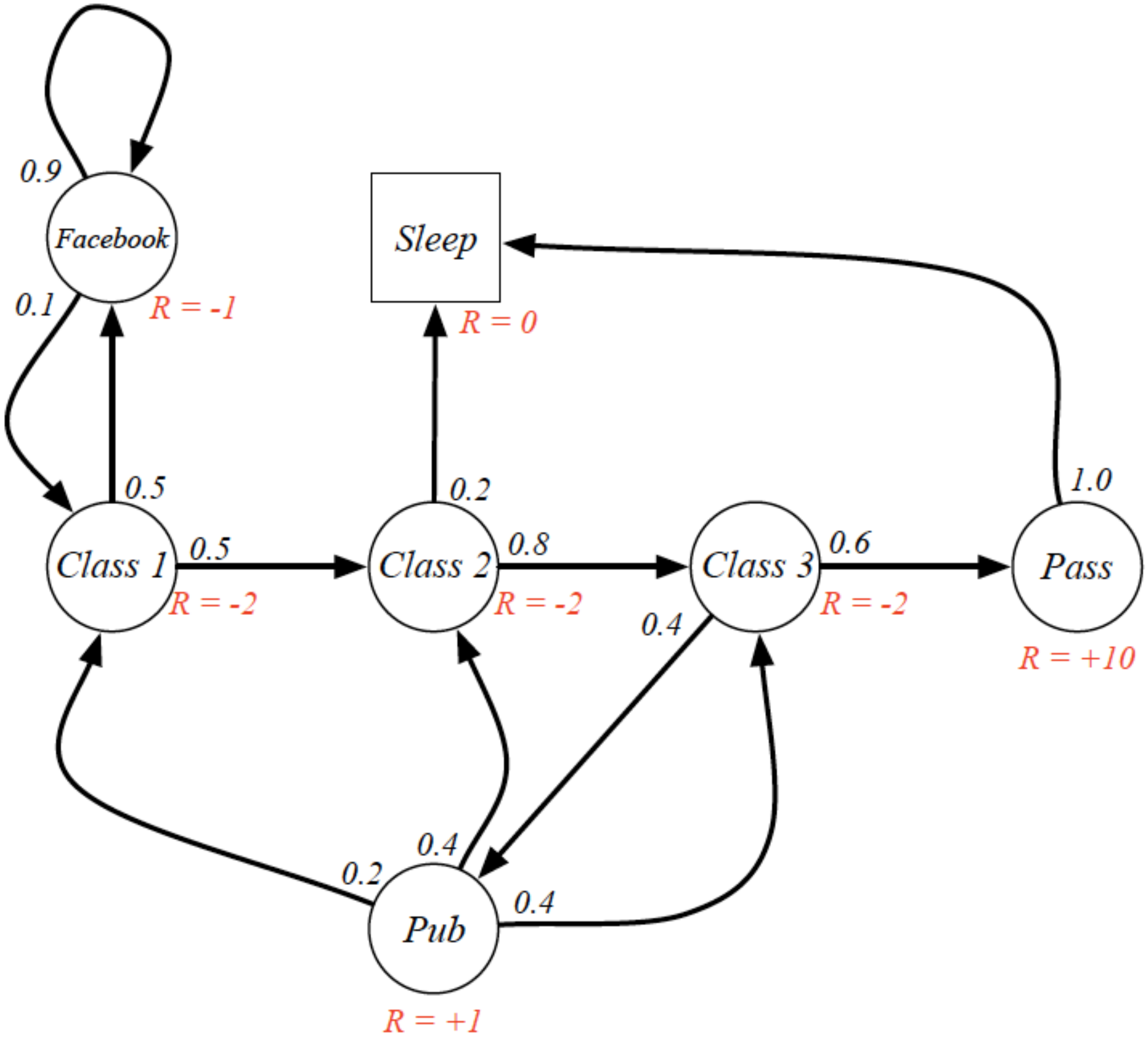
A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Reward Process



Return

Definition

The *return* G_t is the total discounted reward from time-step t.

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Value Function

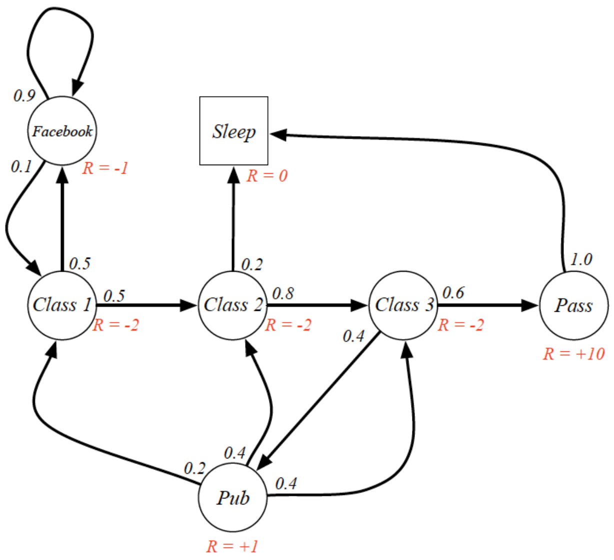
The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Sample Student Returns in MRP



Sample returns for Student MRP:
Starting from $S_1 = C1$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

Bellman’s Equation

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

Bellman's Equation

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Bellman’s Equation Representation

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman's Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1}\, \mathcal{R}
\end{aligned}
$$

Solving the Bellman's Equation

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Rewards

- **Rewards**: $r_t \in \Re$

- **Reward function**: $R(s_t, a_t) = r_t$
  mapping from state-action pairs to rewards

- Common assumption: stationary reward function
  - $R(s_t, a_t)$ is the same $\forall t$
- Exception: terminal reward function often different
  - E.g., in a game: 0 reward at each turn and +1/-1 at the end for winning/losing

- Goal: **maximize sum of rewards** $\boldsymbol{\sum_t R(s_t, a_t)}$

Markov Decision Process

- Definition
  - Set of states: S
  - Set of actions: A
  - Transition model: $\Pr(s_t | s_{t-1}, a_{t-1})$
  - Reward model: $R(s_t, a_t)$
  - Discount factor: $0 \leq \gamma \leq 1$
    - discounted: $\gamma < 1$ undiscounted: $\gamma = 1$
  - Horizon (i.e., # of time steps): h
    - Finite horizon: $h \in \mathbb{N}$ infinite horizon: $h = \infty$

- Goal: find optimal policy

Policy

- Choice of action at each time step

- Formally:
  - Mapping from states to actions
  - i.e., $\pi(s_t) = a_t$
  - Assumption: fully observable states
    - Allows a_t to be chosen only based on current state s_t

How to find an optimal policy ?

- Policy Iteration
- Value Iteration

Policy Optimization

- Policy evaluation:
  - Compute expected utility

    $V^{\pi}(s_0) = \sum_{t=0}^{h} \gamma^t \sum_{s_t} \Pr(s_t | s_0, \pi) \, R(s_t, \pi(s_t))$

- Optimal policy:
  - Policy with highest expected utility

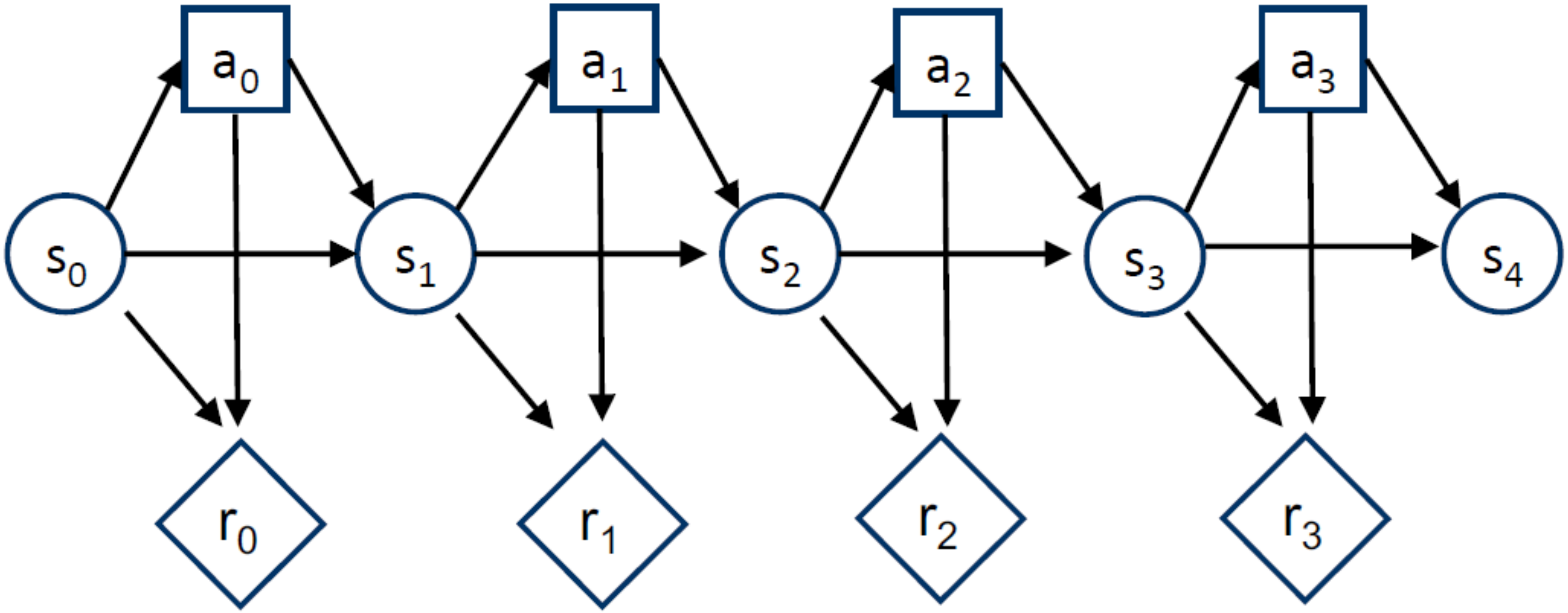
    $V^{\pi^*}(s_0) \geq V^{\pi}(s_0) \;\; \forall \pi$

Policy Optimization

- Several classes of algorithms:
  - Value iteration
  - Policy iteration
  - Linear Programming
  - Search techniques
- Computation may be done
  - Offline: before the process starts
  - Online: as the process evolves

Value Iteration

- Performs dynamic programming
- Optimizes decisions in reverse order



Value Iteration

- Value when no time left:

$$V(s_h) = \max_{a_h} R(s_h, a_h)$$

- Value with one time step left:

$$V(s_{h-1}) = \max_{a_{h-1}} R(s_{h-1}, a_{h-1}) + \gamma \sum_{s_h} \Pr(s_h | s_{h-1}, a_{h-1}) V(s_h)$$

- Value with two time steps left:

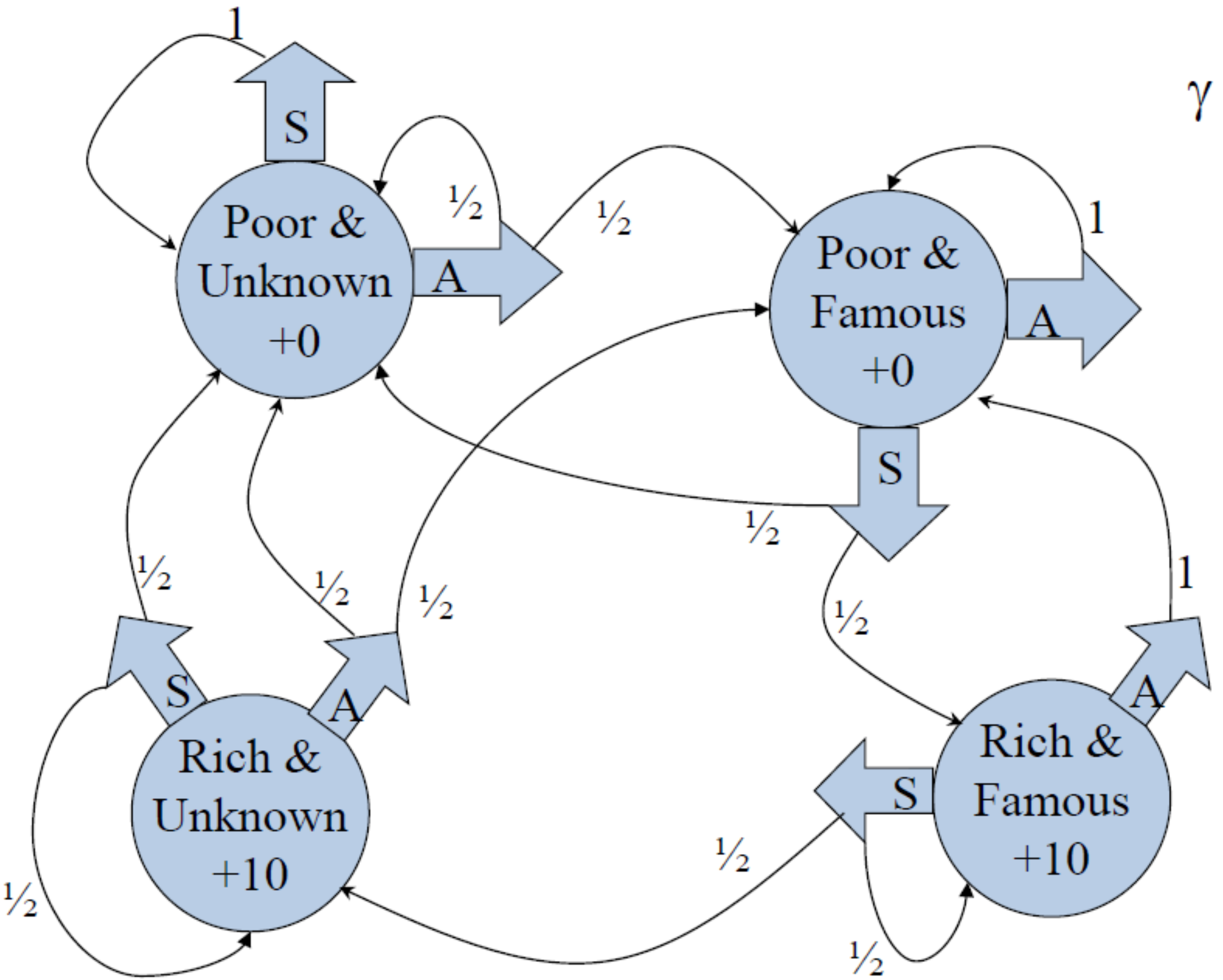
$$V(s_{h-2}) = \max_{a_{h-2}} R(s_{h-2}, a_{h-2}) + \gamma \sum_{s_{h-1}} \Pr(s_{h-1} | s_{h-2}, a_{h-2}) V(s_{h-1})$$

- …
- Bellman's equation:

$$V(s_t) = \max_{a_t} R(s_t, a_t) + \gamma \sum_{s_{t+1}} \Pr(s_{t+1} | s_t, a_t) V(s_{t+1})$$

$$a_t^* = \operatorname{argmax}_{a_t} R(s_t, a_t) + \gamma \sum_{s_{t+1}} \Pr(s_{t+1} | s_t, a_t) V(s_{t+1})$$

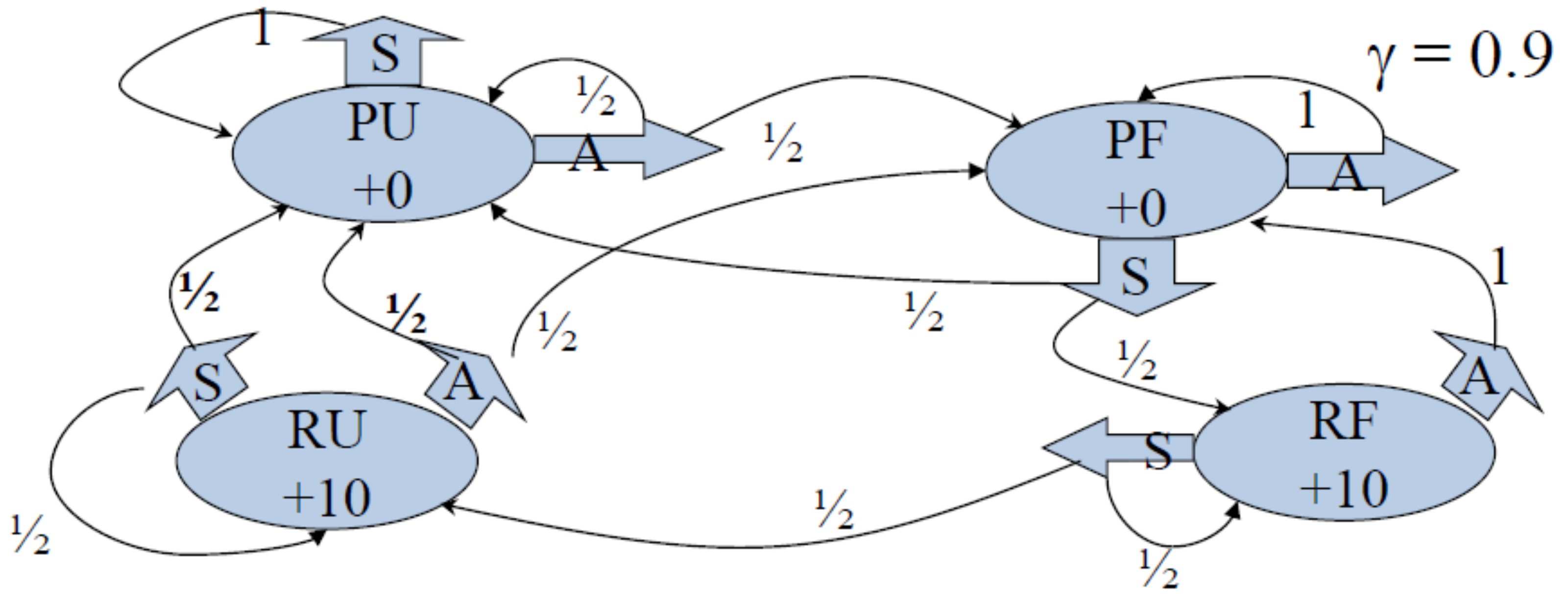
A Markov Decision Process



$\gamma = 0.9$

You own a company

In every state you must choose between **S**aving money or **A**dvertising



t	$V(PU)$	$\pi(PU)$	$V(PF)$	$\pi(PF)$	$V(RU)$	$\pi(RU)$	$V(RF)$	$\pi(RF)$
h	0	A,S	0	A,S	10	A,S	10	A,S
$h-1$	0	A,S	4.5	S	14.5	S	19	S
$h-2$	2.03	A	8.55	S	16.53	S	25.08	S
$h-3$	4.76	A	12.20	S	18.35	S	28.72	S
$h-4$	7.63	A	15.07	S	20.40	S	31.18	S
$h-5$	10.21	A	17.46	S	22.61	S	33.21	S

Finite Horizon

- When h is finite,
- Non-stationary optimal policy
- Best action different at each time step
- Intuition: best action varies with the amount of time left

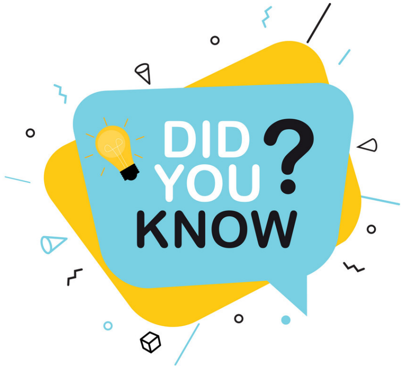
Infinite Horizon

- When h is infinite,
- Stationary optimal policy
- Same best action at each time step
- Intuition: same (infinite) amount of time left at each time step, hence same best action

- Problem: value iteration does an infinite number of iterations…

Infinite Horizon

- Assuming a discount factor γ, after n time steps, rewards are scaled down by γ^n
- For large enough n, rewards become insignificant since $\gamma^n \to 0$
- Solution:
  - pick large enough n
  - run value iteration for n steps
  - Execute policy found at the n^{th} iteration



Google’s PageRank developed by Sergey Brin and Larry Page is based on a Markov Decision Process(MDP) utlizing the Markov chains making it the most used applications of a MDP.