

Обучение с подкреплением

Часть 1
Введение

Виды машинного обучения

С учителем



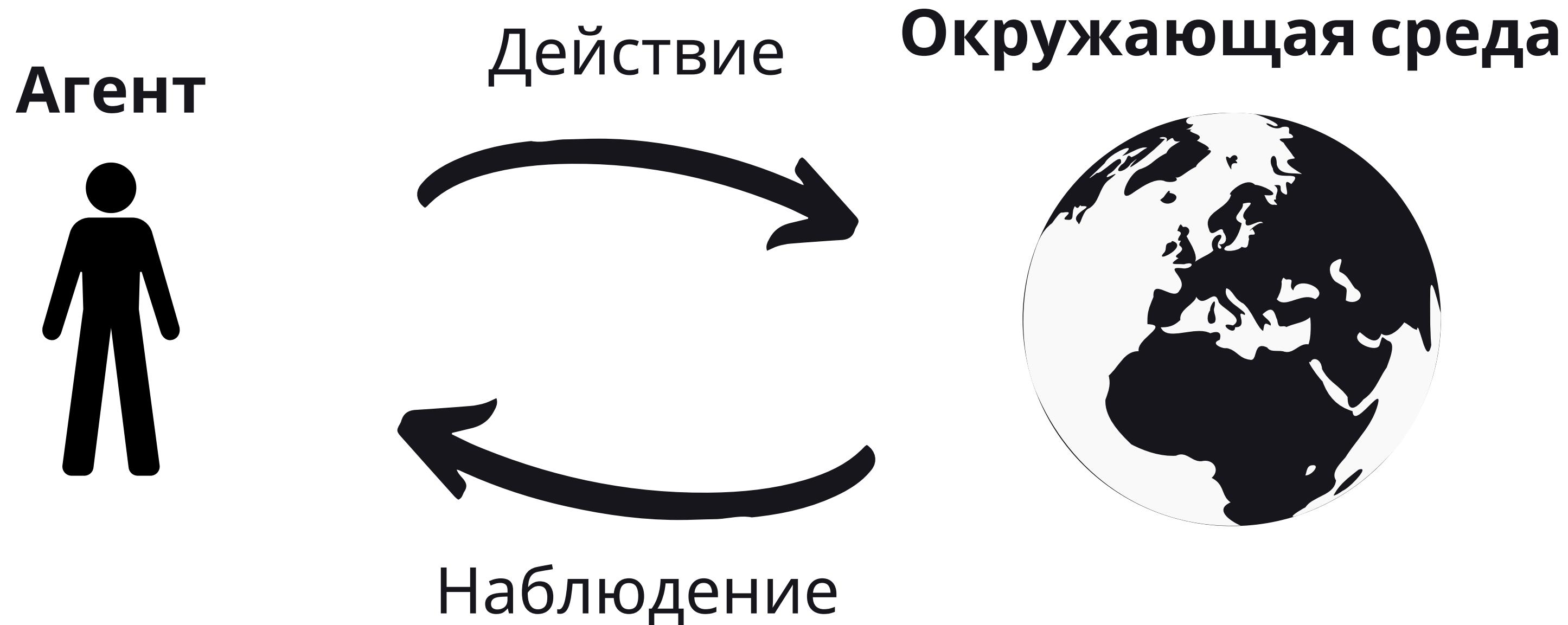
Без учителя



С подкреплением



Цикл взаимодействия



Цель: максимизировать сумму вознаграждений при серии взаимодействий

Гипотеза вознаграждения

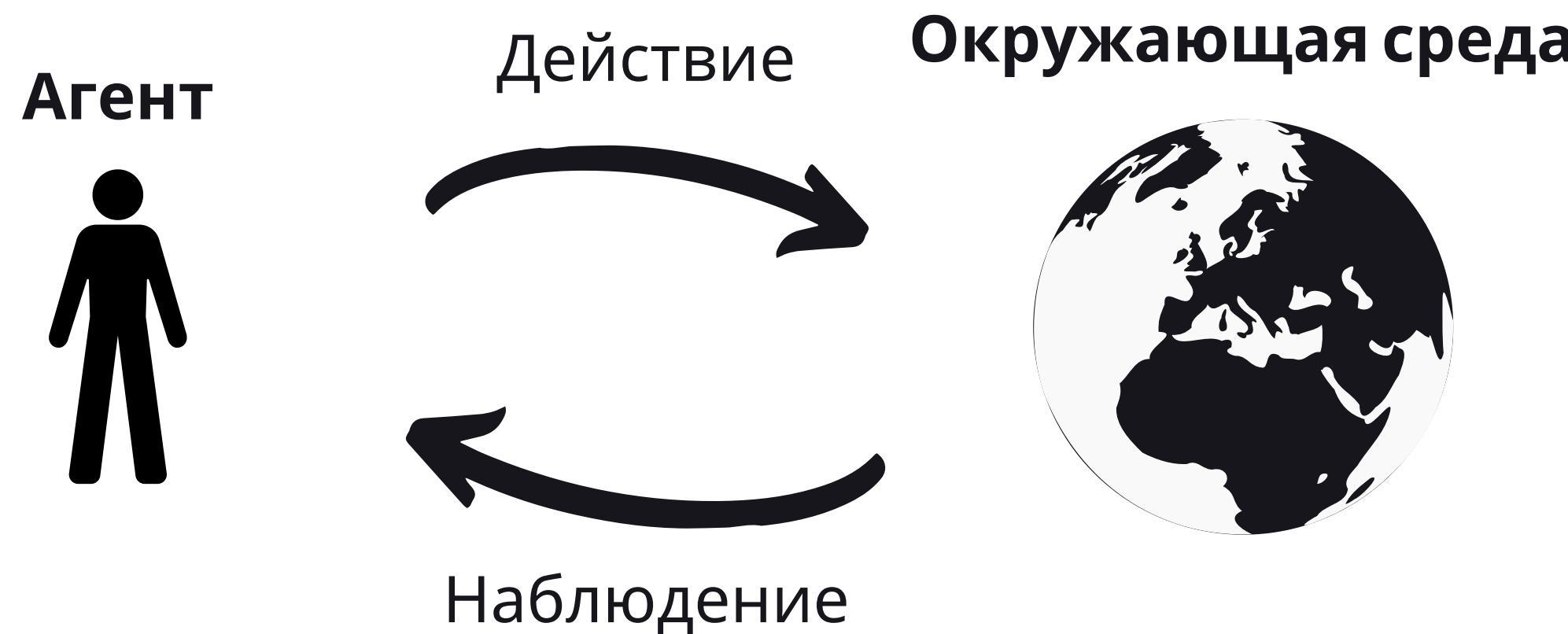
Обучение с подкреплением основано на **гипотезе вознаграждения**:
*Любую цель можно формализовать как результат
максимизации совокупного вознаграждения.*

Примеры

- Видеоигры или настольные игры:
 - **цель:** набрать наибольшее количество очков
- Походка робота:
 - **цель:** пройти максимальную дистанцию
- Управление инвестиционным портфелем:
 - **цель:** заработать наибольшее количество денег

Формальная часть

Агент и среда



- На каждом шаге t агент
 - получает наблюдение O_t и награду R_t
 - выполняет действие A_t
- На каждом шаге t среда
 - получает действие A_t
 - предоставляет наблюдение O_t и награду R_t

Награда

- Награда R_t – это обратная связь, с помощью которой измеряется успех или неудача действий агента
- Награда является скалярной величиной
- Задача агента максимизировать совокупное вознаграждение

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$$

Дисконтирование

- Величина дисконтирования $\gamma \in [0, 1]$
- Вводится для того чтобы дать больший вес краткосрочных наград
- Позволяет избежать бесконечного совокупного вознаграждения

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

Ожидаемое вознаграждение

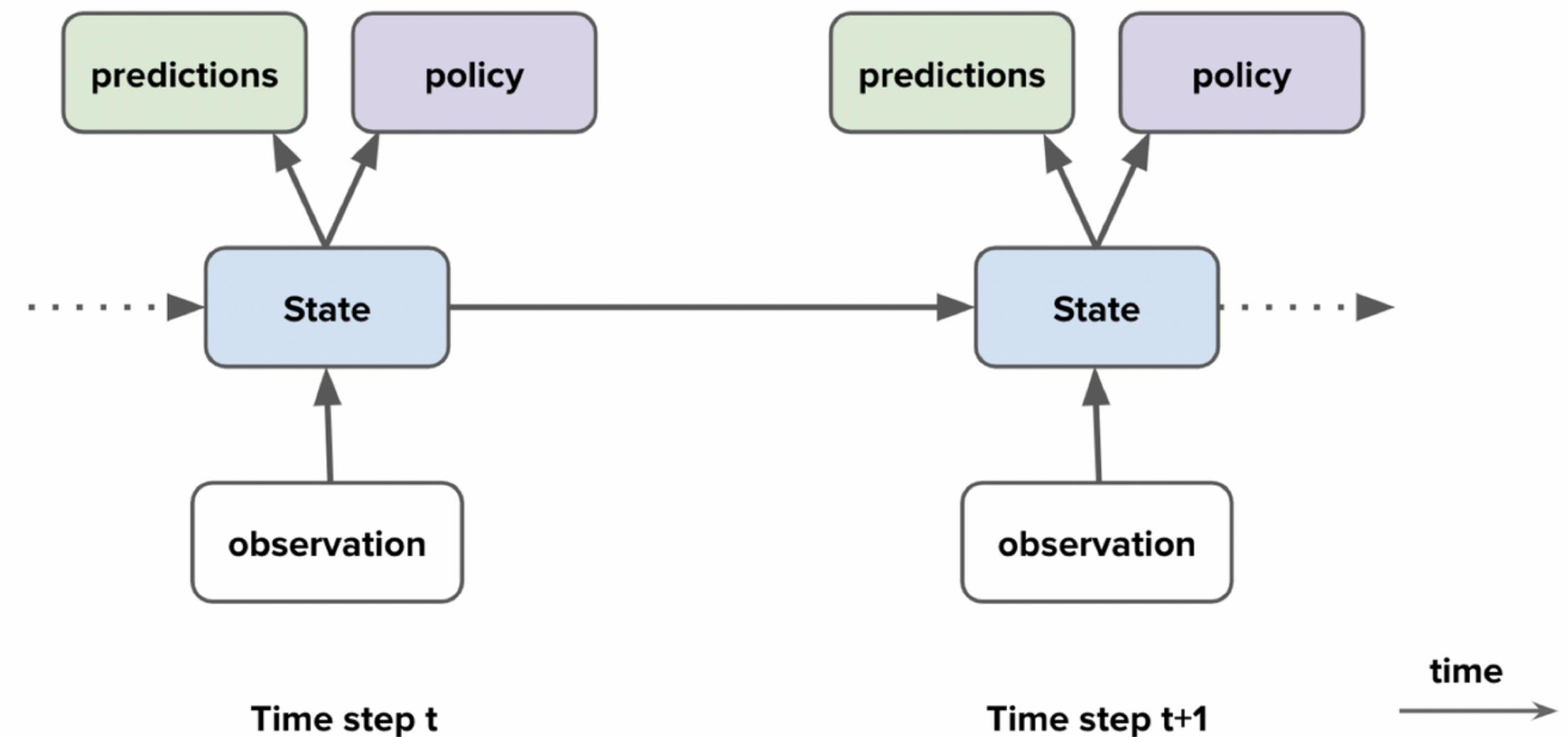
- Ожидаемое совокупное вознаграждение в состоянии S_t называется value
 - $v(s) = \mathbb{E}[G_t | S_t = s]$
- Цель: максимизировать value выбирая подходящие действия
- Можно вычислять рекурсивно
 - $G_t = R_{t+1} + \gamma G_{t+1}$
 - $v(s) = \mathbb{E}[R_{t+1} + v(S_{t+1}) | S_t = s]$

Поговорим об агенте



Компоненты агента

- Agent state
- Policy
- Value function
- Model



Agent state

- История всех наблюдений, действий и наград
 - $\mathcal{H}_t = O_0, A_0, R_1, O_1, \dots, O_{t-1}, A_{t-1}, R_t, O_t$
- Из истории конструируется состояние агента S_t

Марковские состояния

- Состояние S_t называется Марковским, когда выполнено
 - $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \dots, S_t]$
- При наличии состояния, нет необходимости хранить всю историю

Fully Observable Environments

Full observability

- Агент видит полное состояние среды
 - $S_t = O_t = \text{environment state}$
- Формально это называется Markov decision process (MDP)

Partial Observable Environments

Partial observability

- Агент не видит полного состояние среды, например
 - робот с видеокамерой, но не знающий своей локации
 - игрок покера, который видит только свои карты
- Формально это называется partially observable Markov decision process (POMDP)

Policy

- Это стратегия, которую агент использует для определения следующего действия на основе текущего состояния
- Два вида
 - детерминированный
 - $a = \pi(s)$
 - случайный
 - $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Value function

- Функция предсказания будущего вознаграждения
- Используется для оценки хороших и плохих состояний
- Может использоваться для выбора действия
 - $v_\pi(s) = \mathbb{E}[G_t | S_t = s, \pi] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, \pi]$

Model

- Модель предсказывает что среда сделает далее
- \mathcal{P} предсказывает следующее состояние
- \mathcal{R} предсказывает следующую награду

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Виды агентов

Категории агентов (1)

- Value Based
 - ~~Policy~~
 - Value Function
- Policy Based
 - Policy
 - ~~Value Function~~
- Actor Critic
 - Policy
 - Value Function

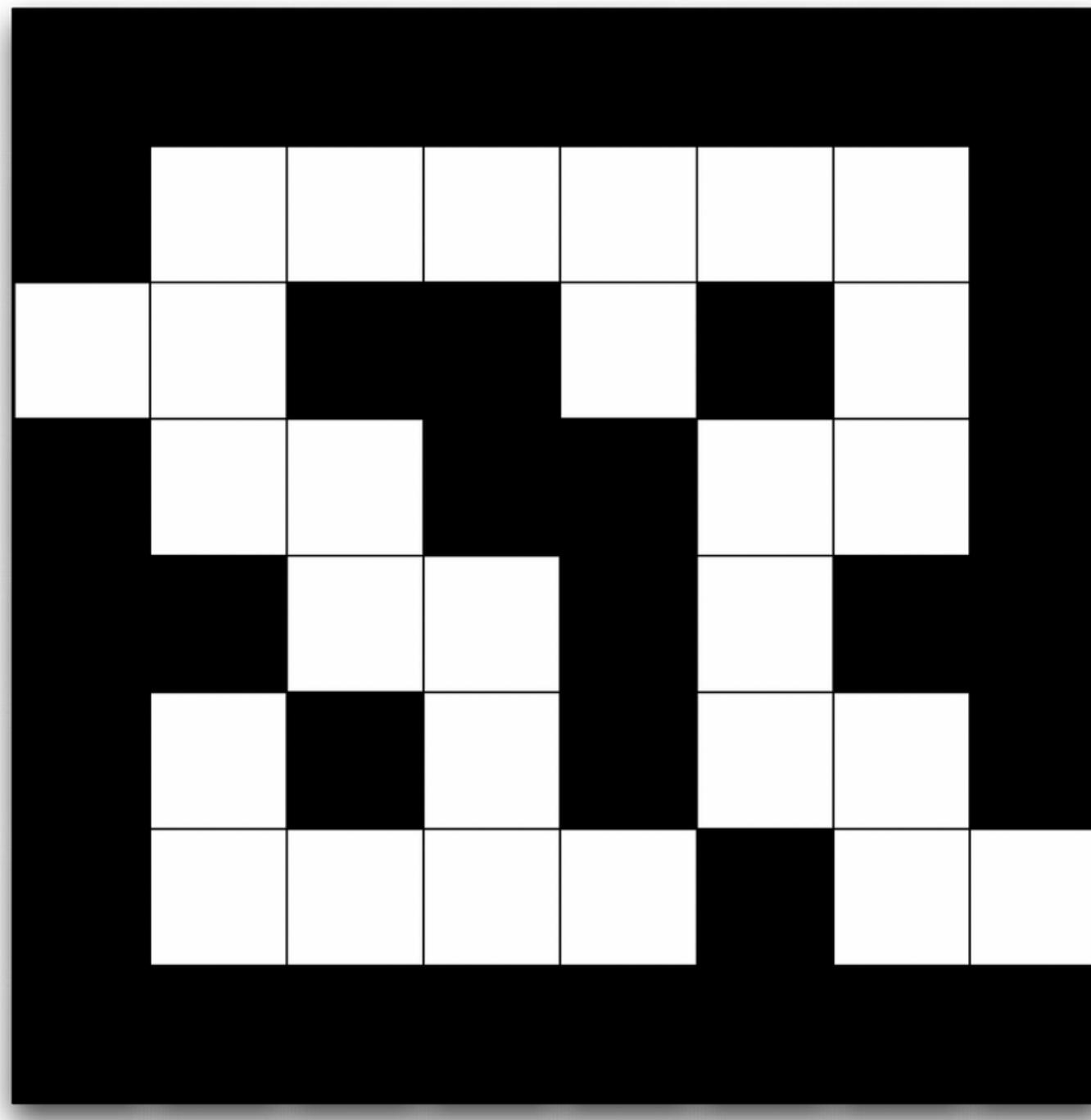
Категории агентов (2)

- Model Free
 - Policy and/or Value Function
 - ~~Model~~
- Model Based
 - Optionally Policy and/or Value Function
 - Model

Пример

Лабиринт

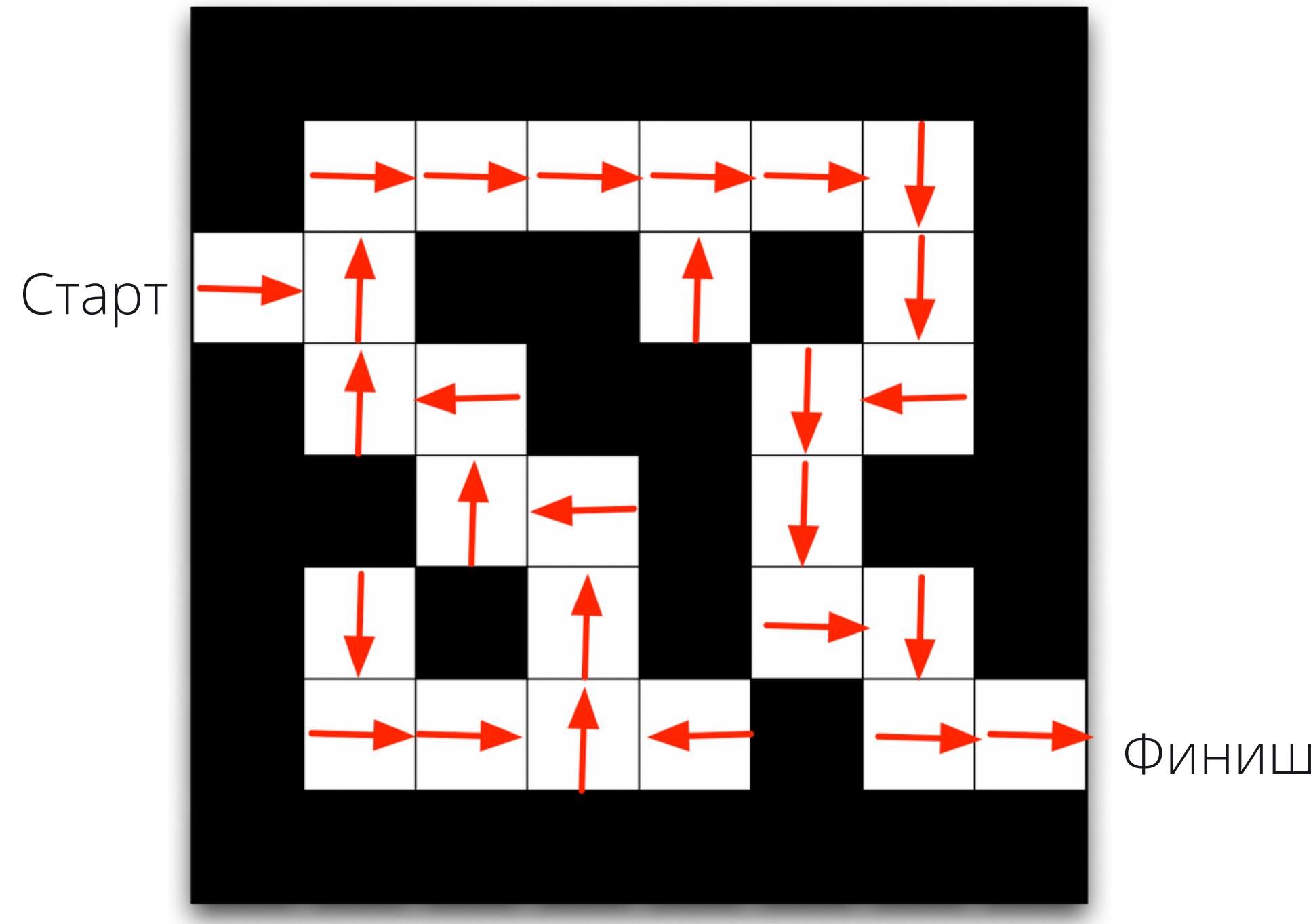
Старт



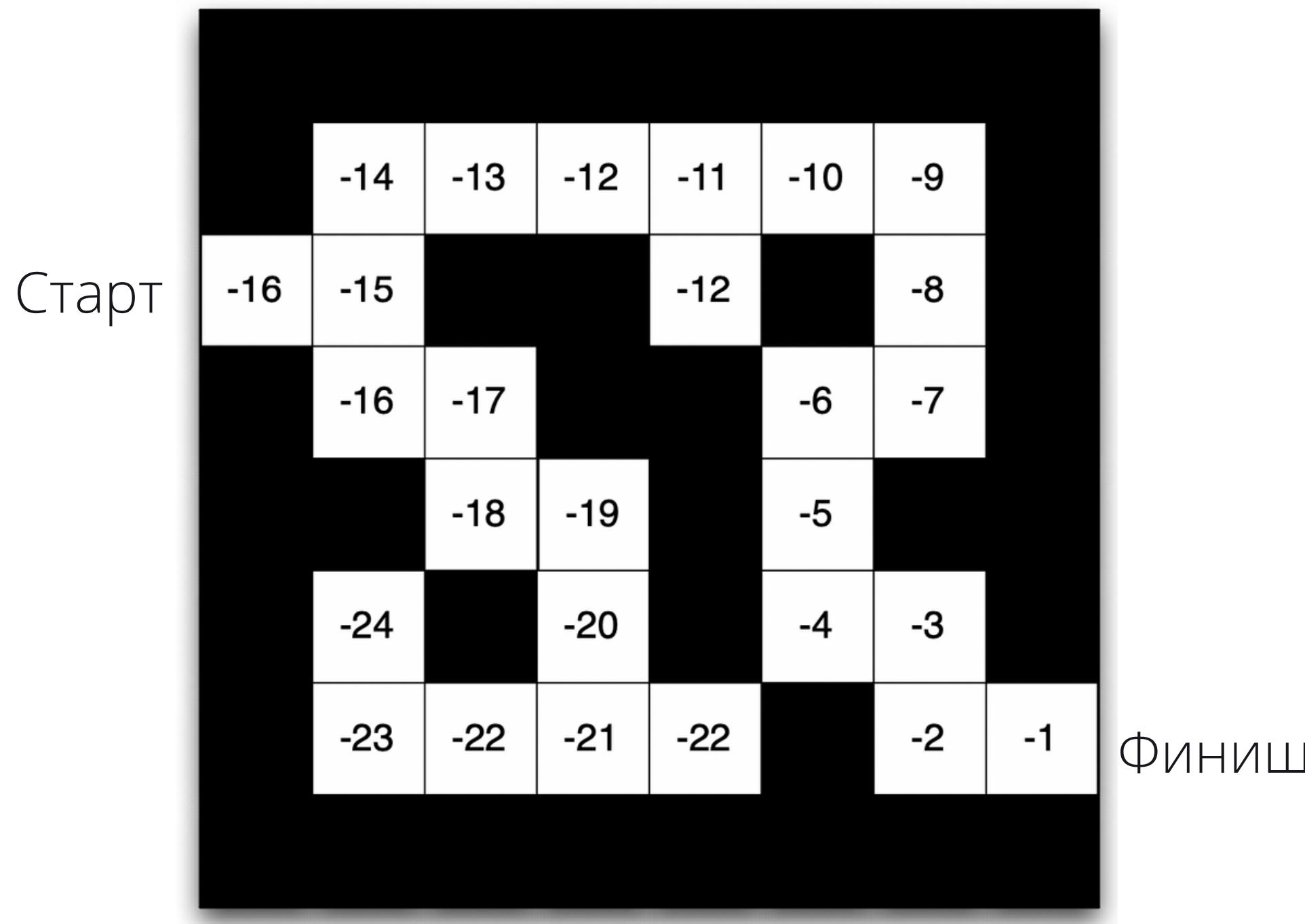
Финиш

- Награда: -1 за шаг
- Действия: $\leftarrow \uparrow \rightarrow \downarrow$
- Состояние: позиция в лабиринте

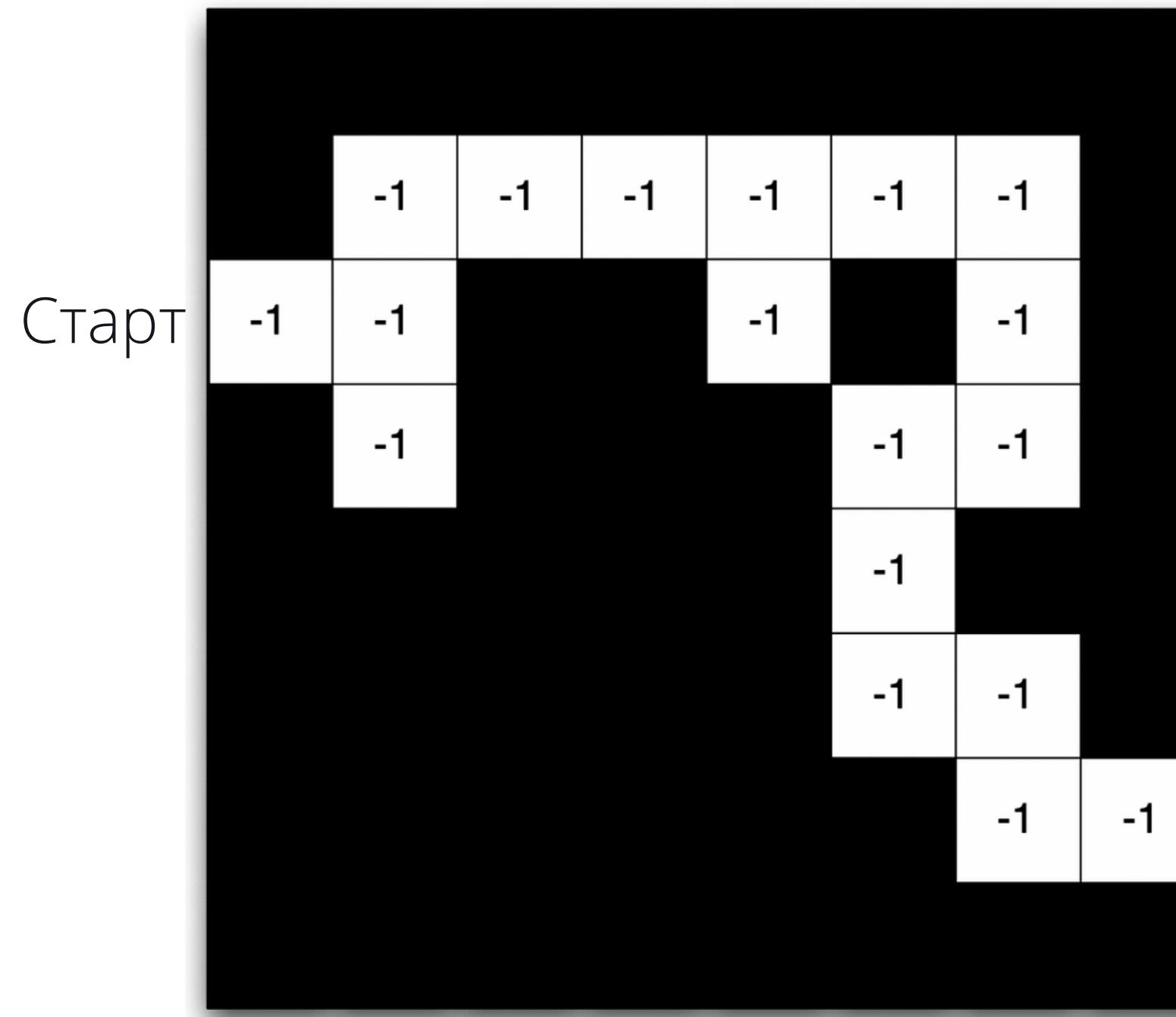
Policy



Value function



Model



Старт

Финиш

- Схема отражает модель частичного перехода $\mathcal{P}_{ss'}^a$,
- Числам соответствует награда при переходе из каждого состояния $\mathcal{R}_{ss'}^a$,

Обучение с подкреплением

Часть 2

Markov Decision Process

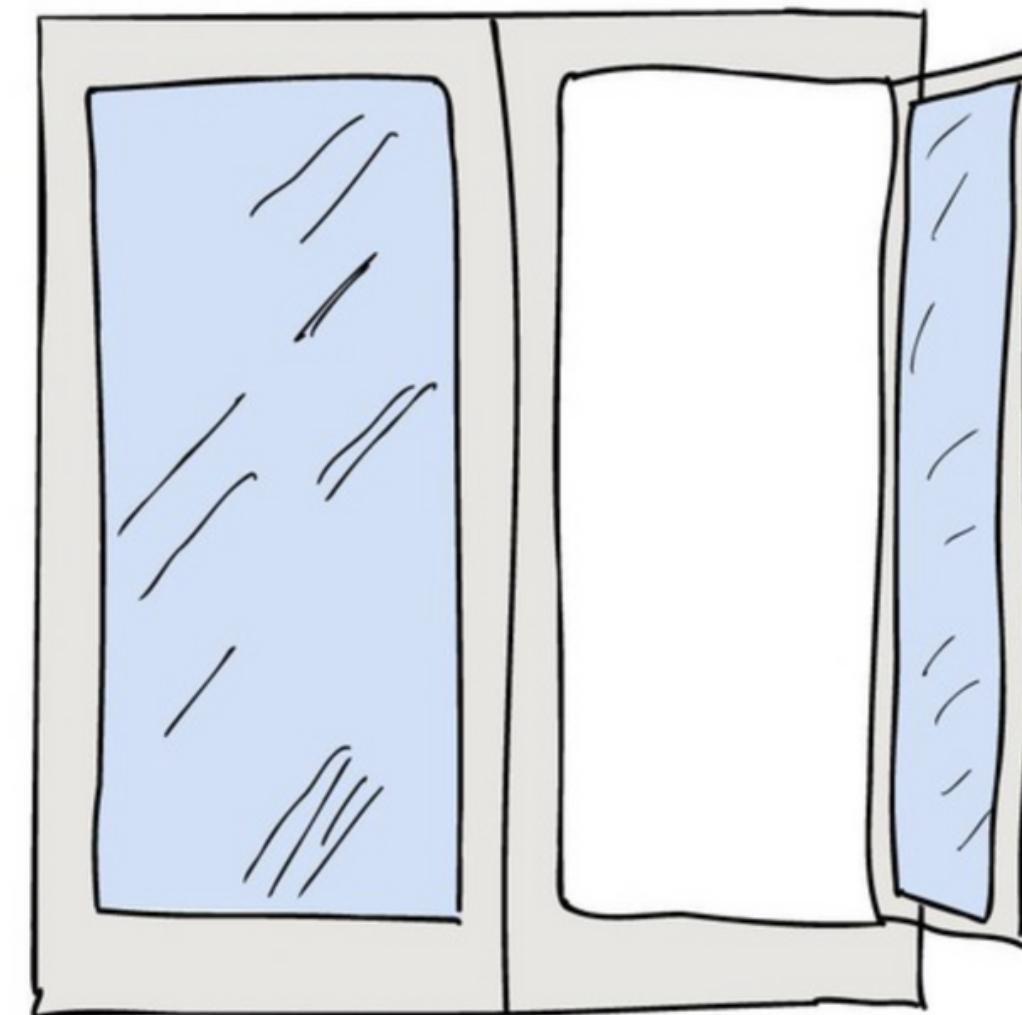
Что это и зачем оно

MDP - это формальное описание среды (environment) в RL.

Будем говорить про полностью наблюдаемый (fully observable)
случай для простоты,
т.е. когда агент полностью видит состояние среды.

Практически все задачи в RL могут быть формализованы как MDP.

Сейчас будет немножко душно



Свойство марковости

Def. Состояние S_t называется *Марковским* если и только если

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

то есть состояние содержит всю необходимую информацию.

Другими словами, если состояние известно, то можно выкинуть историю состояний.

Матрица переходов

Для пары состояний s, s' вероятность перехода определяется как

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

Матрица переходов \mathcal{P} определяет вероятности перехода между всеми парами состояний s, s'

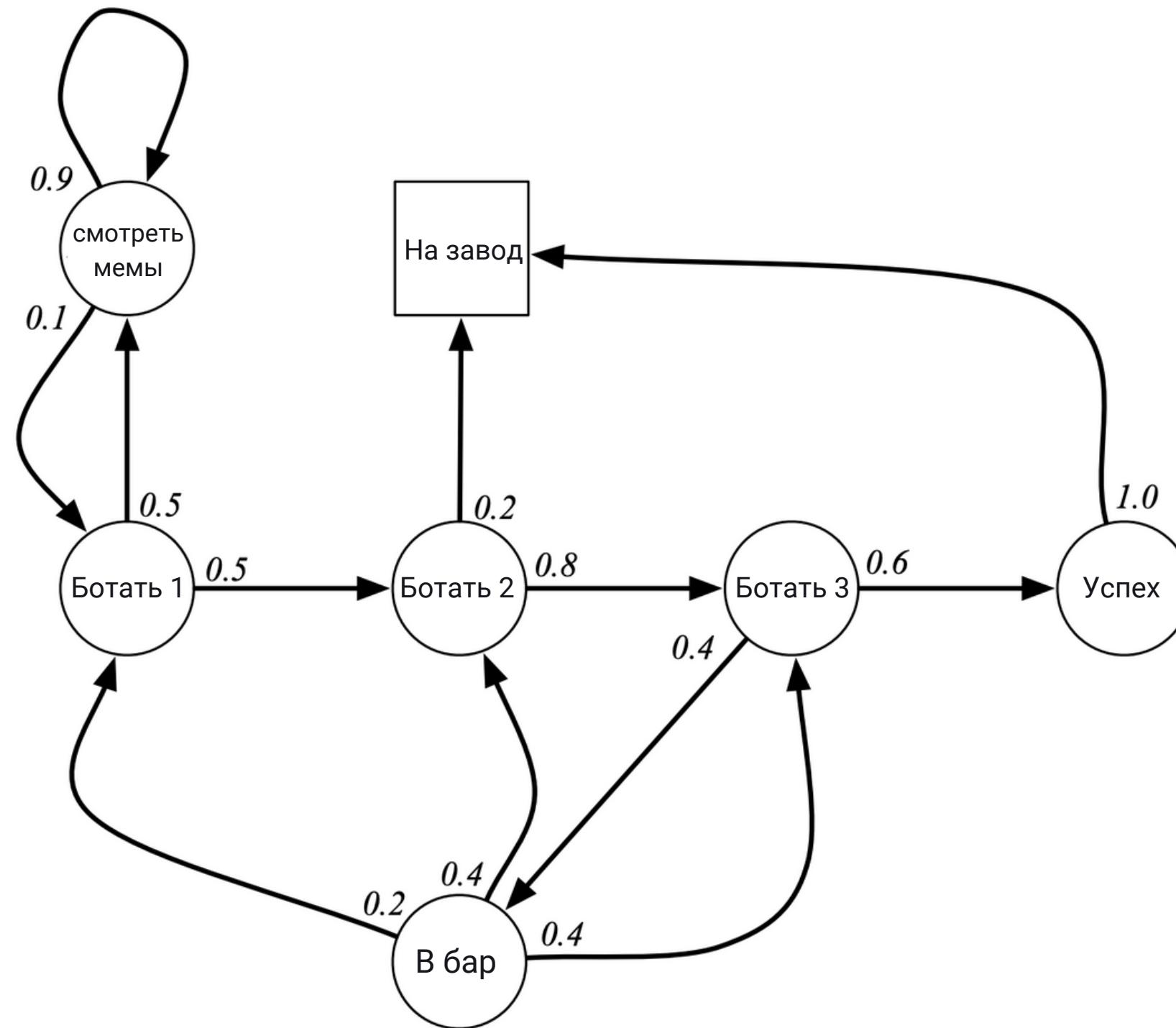
$$\mathcal{P} = \begin{matrix} & \text{to} \\ \text{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{matrix}$$

Марковские цепи

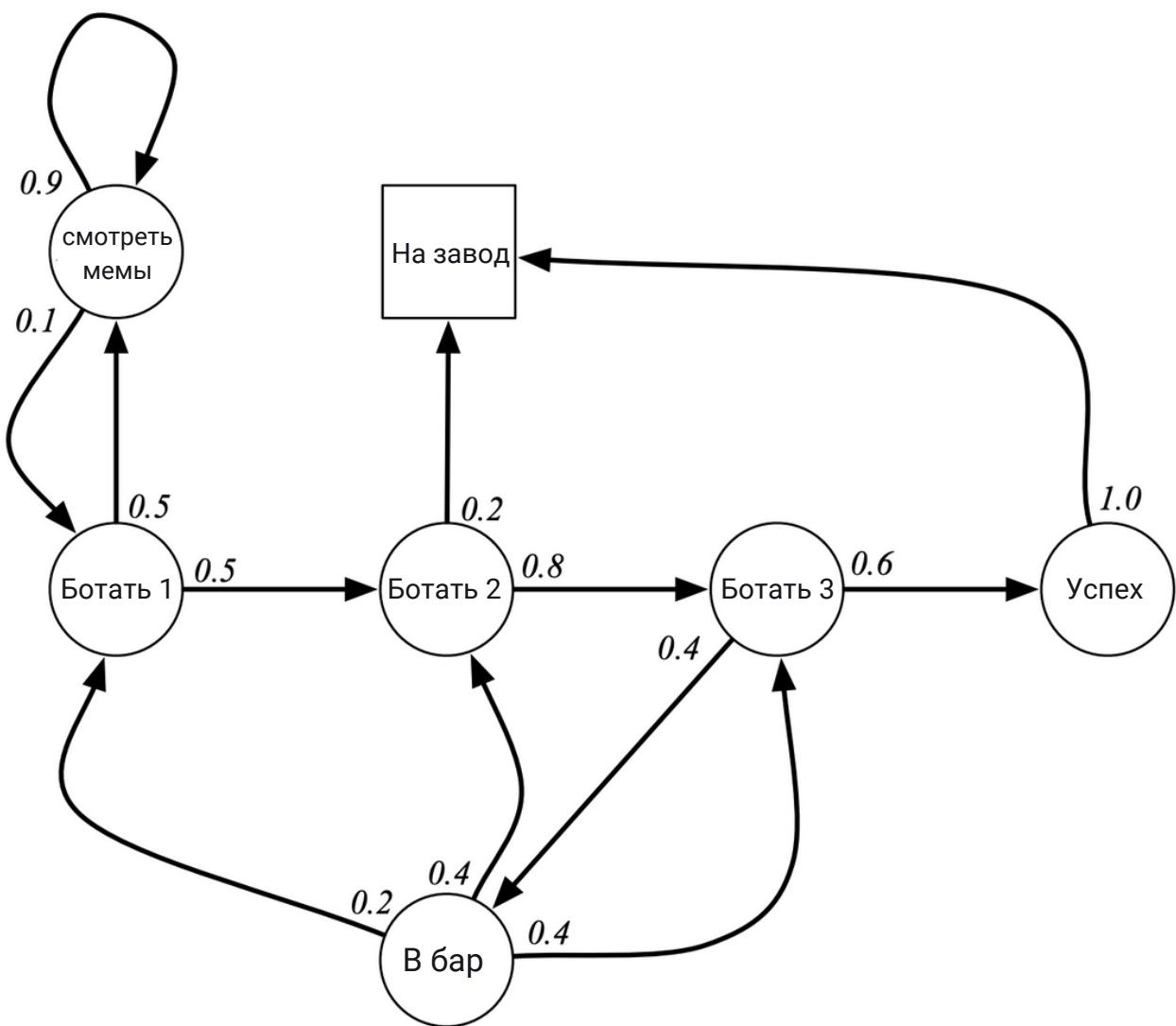
Def. Марковский процесс (или Марковская цепь) - пара $\langle S, \mathcal{P} \rangle$

где S - конечное множество состояний, \mathcal{P} матрица переходов

Пример: АМI Markov chain



Пример: 191 Markov chain transition matrix



$$\mathcal{P} = \begin{bmatrix} & \text{Б1} & \text{Б2} & \text{Б3} & \text{у} & \text{Бар} & \text{Мемы} & \text{З} \\ \text{Б1} & & 0.5 & & & & & 0.5 \\ \text{Б2} & & & 0.8 & & & & 0.2 \\ \text{Б3} & & & & 0.6 & & & 1.0 \\ \text{у} & & & & & 0.4 & & 0.4 \\ \text{Бар} & & & & & & 0.6 & \\ \text{Мемы} & & & & & & & 0.9 \\ \text{З} & & & & & & & 1 \end{bmatrix}$$

Вопросы?



Markov Decision Process

Def. Markov Decision Process - пятерка $\langle S, A, \mathcal{P}, \mathcal{R}, \gamma \rangle$

S - конечное множество состояний

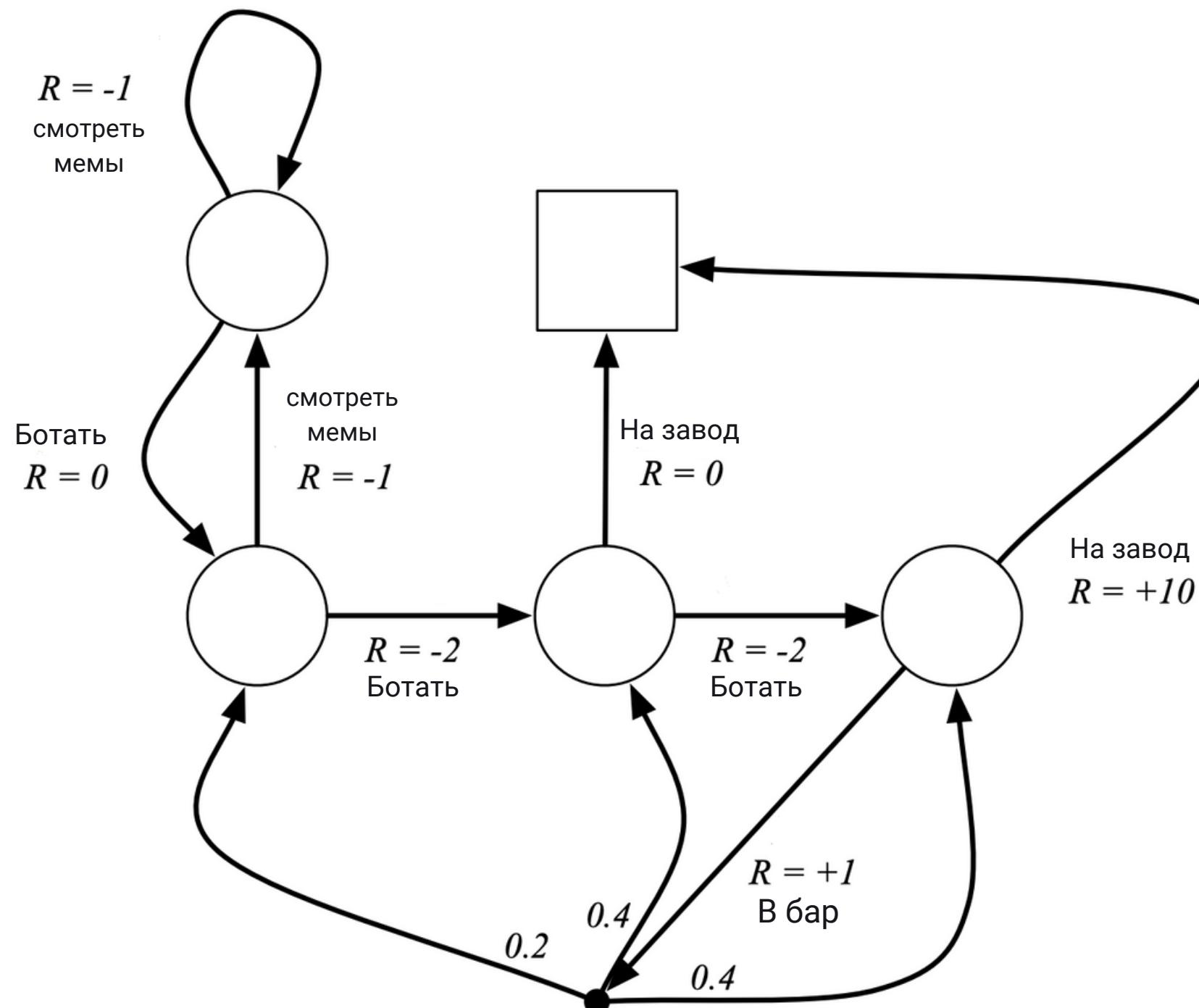
\mathcal{P} - матрица переходов $\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$

A - конечное множество действий

\mathcal{R} - функция вознаграждения $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$

γ - множитель дисконтирования $\gamma \in [0, 1]$

AMI Markov Decision Process



Policy

Def. Policy π - это распределение действий (actions) при условии состояния

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

- Policy полностью определяет поведение агента;
- Policy зависит только от текущего состояния;

Value functions

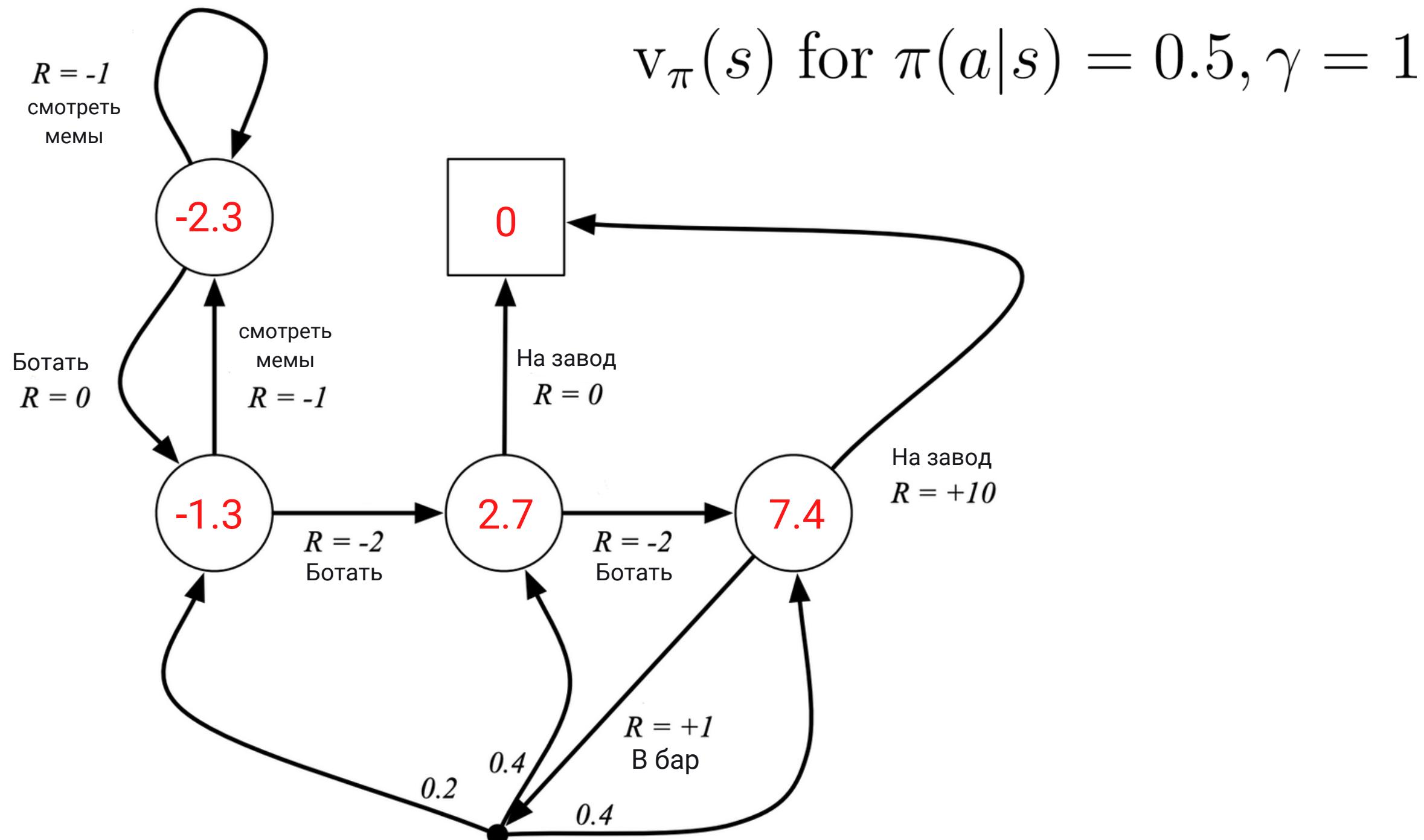
Def. *State-value function* $v_\pi(s)$ MDP - ожидаемая награда из состояния S при условии следования policy π :

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

Def. *Action-value function* $q_\pi(s, a)$ MDP - ожидаемая награда из состояния S при условии, что мы совершим действие A , а после будем действовать в соответствии с policy π :

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

AM| Markov Decision Process



Bellman equation

State-value function $v_\pi(s)$ можно разложить на сумму
"мгновенной награды" на этом шаге и дисконтированной награды
в будущем:

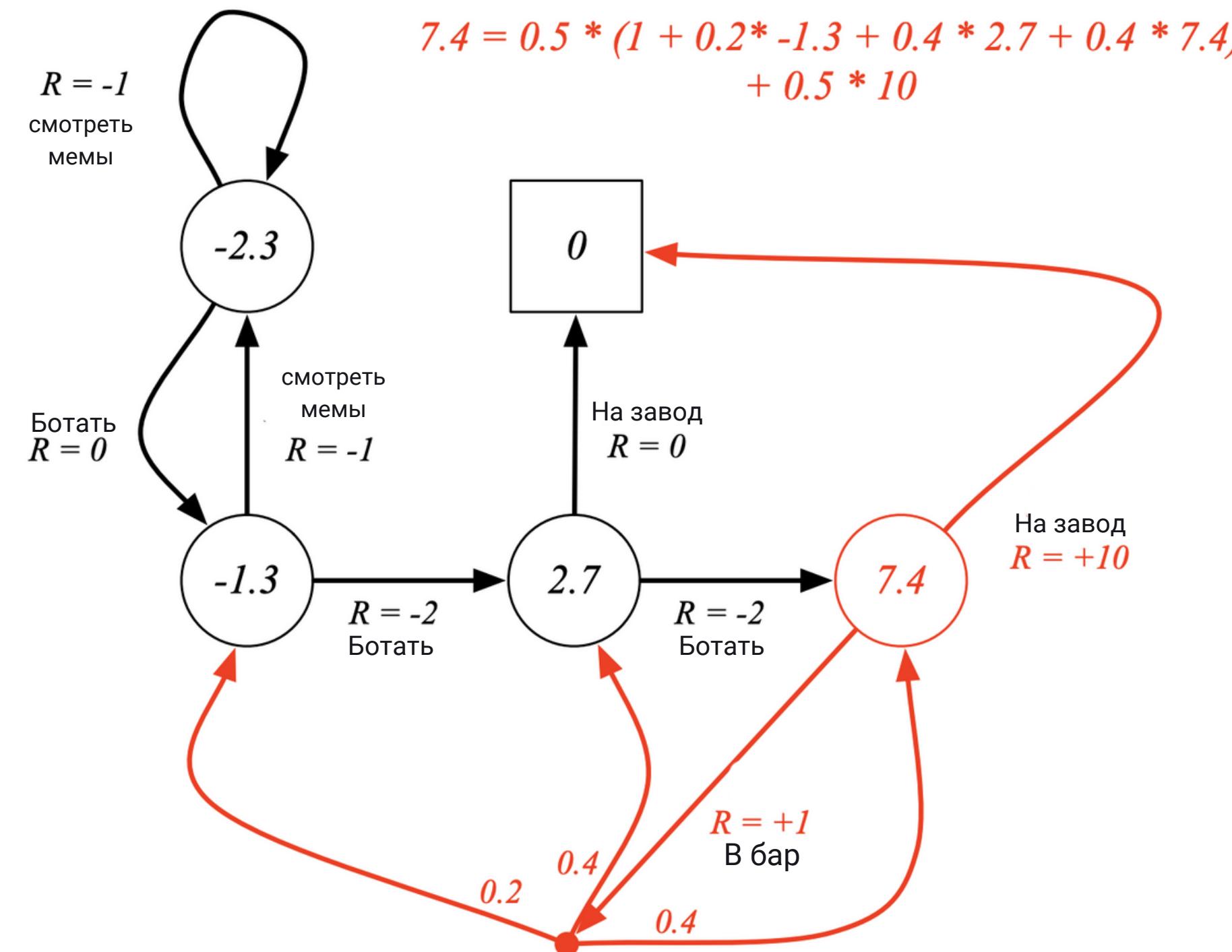
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1}] | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned}$$

Bellman equation

То же самое работает и для action-value function:

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

AMI MDP Bellman example



Optimal value functions

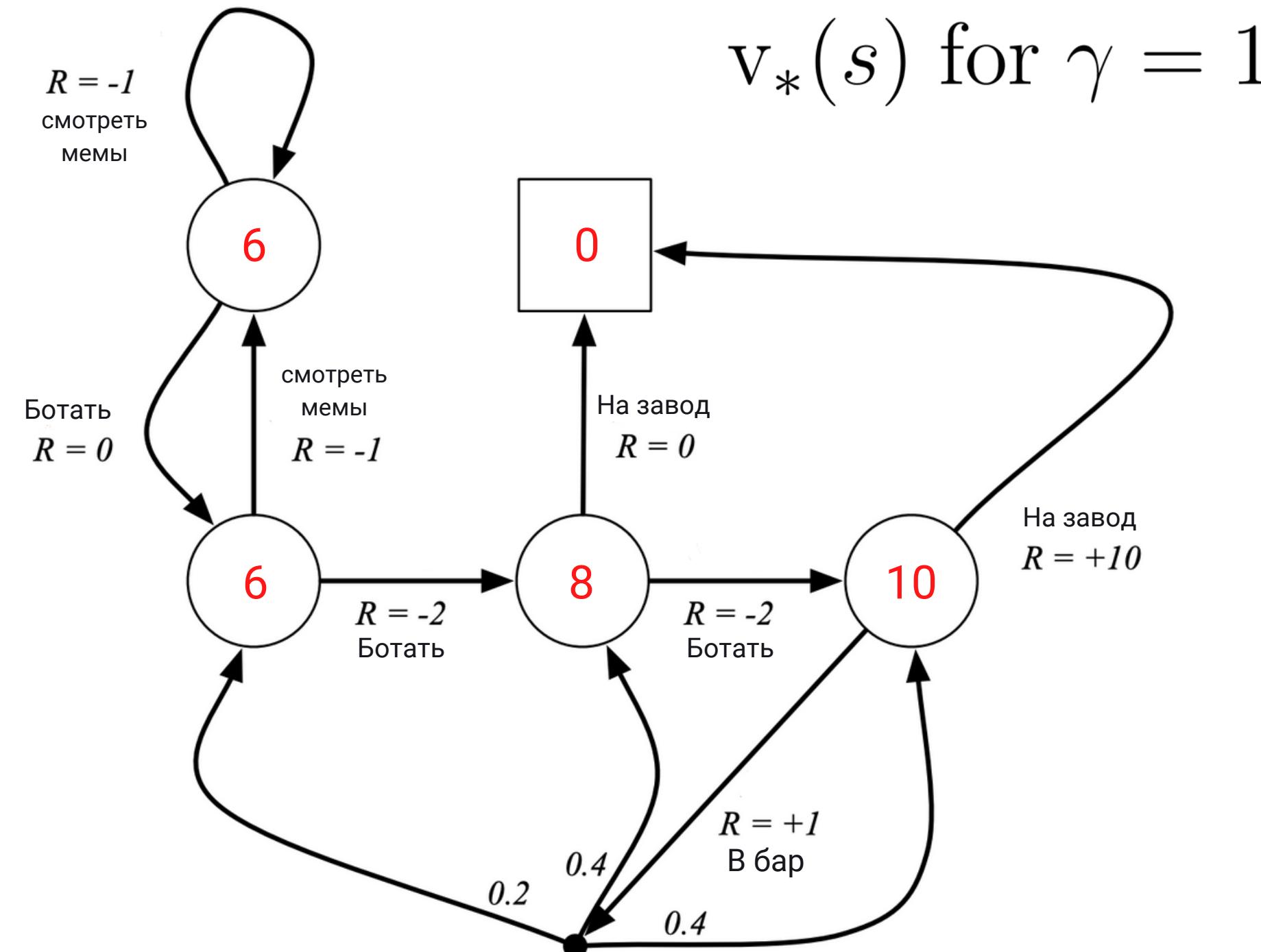
Def. *Optimal state-value function* $v_*(s)$ - максимум по всем policy среди всех state-value functions:

$$v_*(s) = \max_{\pi} \{v_{\pi}(s)\}$$

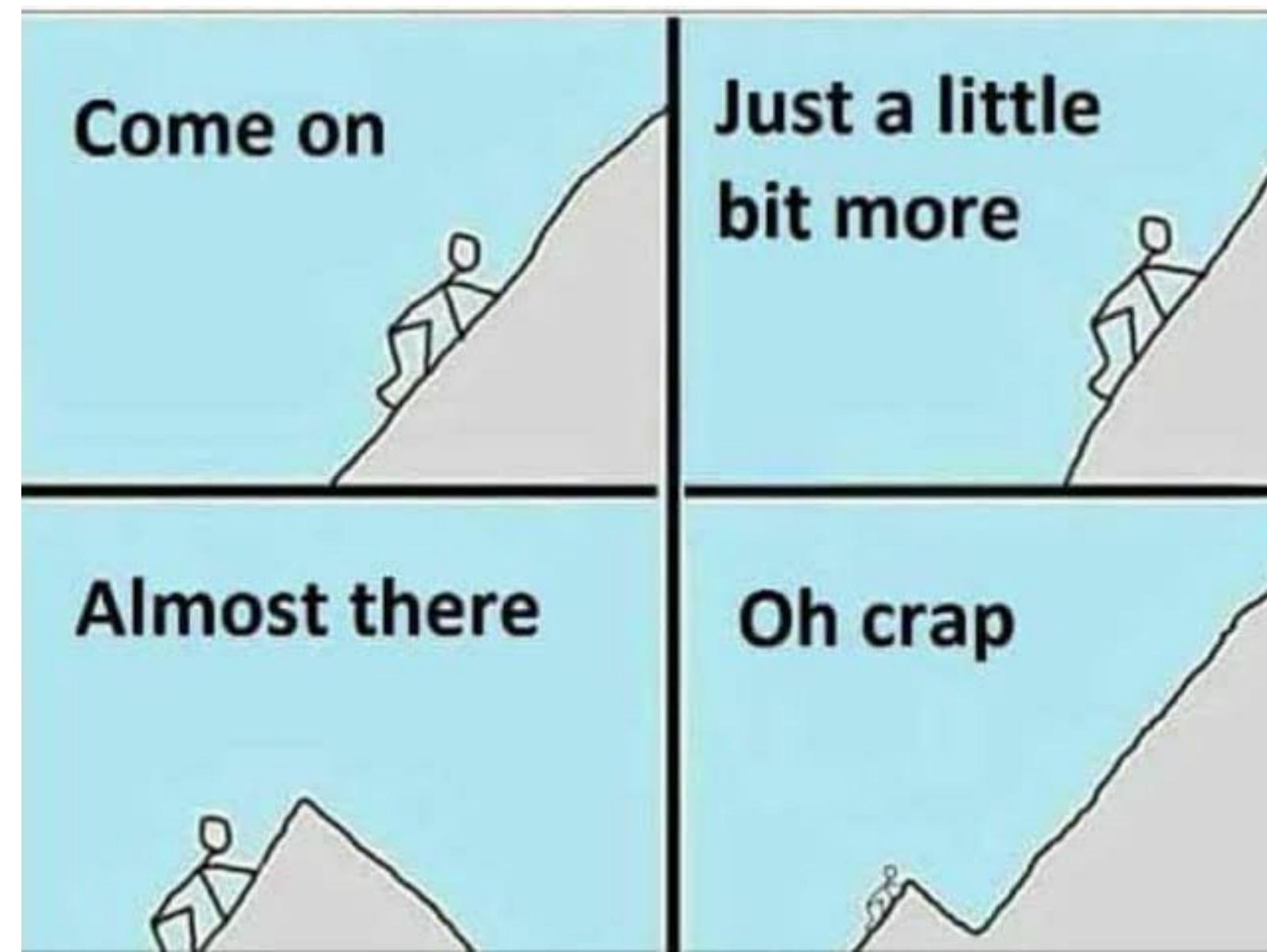
Def. *Optimal action-value function* $q_*(s, a)$ - максимум по всем policy среди всех action-value functions:

$$q_*(s, a) = \max_{\pi} \{q_{\pi}(s, a)\}$$

Пример: Optimal value function



Еще немножко



Optimal policy

Введем частичный порядок на policy: $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s$

Теоремка

Для любого MDP:

- Существует оптимальная π_* (хотя бы одна), которая не хуже любой другой π
- Любая π_* соответствует оптимальным $v_*(s)$ и $q_*(s, a)$

А зачем оно нам

Оптимальную policy можно найти простой максимизацией $q_*(s, a)$:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

Так это же отлично!

Знаем $q_*(s, a) = \text{победа}$



Bellman optimality equations

То же самое работает и для action-value function:

$$v_*(s) = \max_a \mathbb{E}_{s' \sim \mathcal{P}} [R_{t+1} + \gamma v_*(s') | S_t = s]$$

$$q_*(s, a) = \mathbb{E}_{s' \sim \mathcal{P}} [R_{t+1} + \gamma \max_a q_*(s', a) | S_t = s]$$

Что со всем этим делать

На практике люди пытаются решать Bellman optimality equation итеративно:

- Value iteration
- Policy iteration
- Q-learning
- SARSA

Обучение с подкреплением

Часть 3

Динамическое программирование

Зачем?

- Policy evaluation
 - Данна политика, найти value function
- Policy improvement
 - Данна модель, найти оптимальную политику

Почему?

Реккурентные уравнения Беллмана:

- $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$
- $q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$

Напоминание:

- $v_\pi(s) = q_\pi(s, \pi(s))$

Как?

- Prediction
 - Iterative policy evaluation
- Control
 - Policy iteration
 - Value iteration

Policy evalutaion

$$1. v_0(s) = 0$$

$$2. \forall s \in S : v_{i+1} = \mathbb{E}_\pi[R_{t+1} + \gamma v_i(S_{t+1}) | S_t = s]$$

$$3. v_{i+1} = v_i \iff v_\pi = v_i$$

4. ???

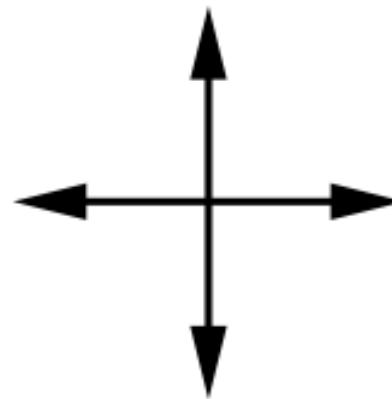
5. PROFIT

Теорема Банаха (о неподвижной точке)

- Рассмотрим метрическое пространство всех value functions
 - $\|u - v\|_\infty = \max_{s \in S} |u(s) - v(s)|$
- Оператор Беллмана $T(v_i) = v_{i+1}$ сжимает пространство
 - $$\begin{aligned} \|T(u) - T(v)\|_\infty &= \max_{s \in S} |\mathbb{E}[R_{t+1} + \gamma u(S_{t+1})|S_t = s] - \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s]| \\ &= \gamma \max_{s \in S} |\mathbb{E}[u(S_{t+1}) - v(S_{t+1})|S_t = s]| \\ &\leq \gamma \|u - v\|_\infty \end{aligned}$$
- По теореме Банаха неподвижная точка существует и единственна, если величина дисконтирования меньше 1

Не выходи из комнаты

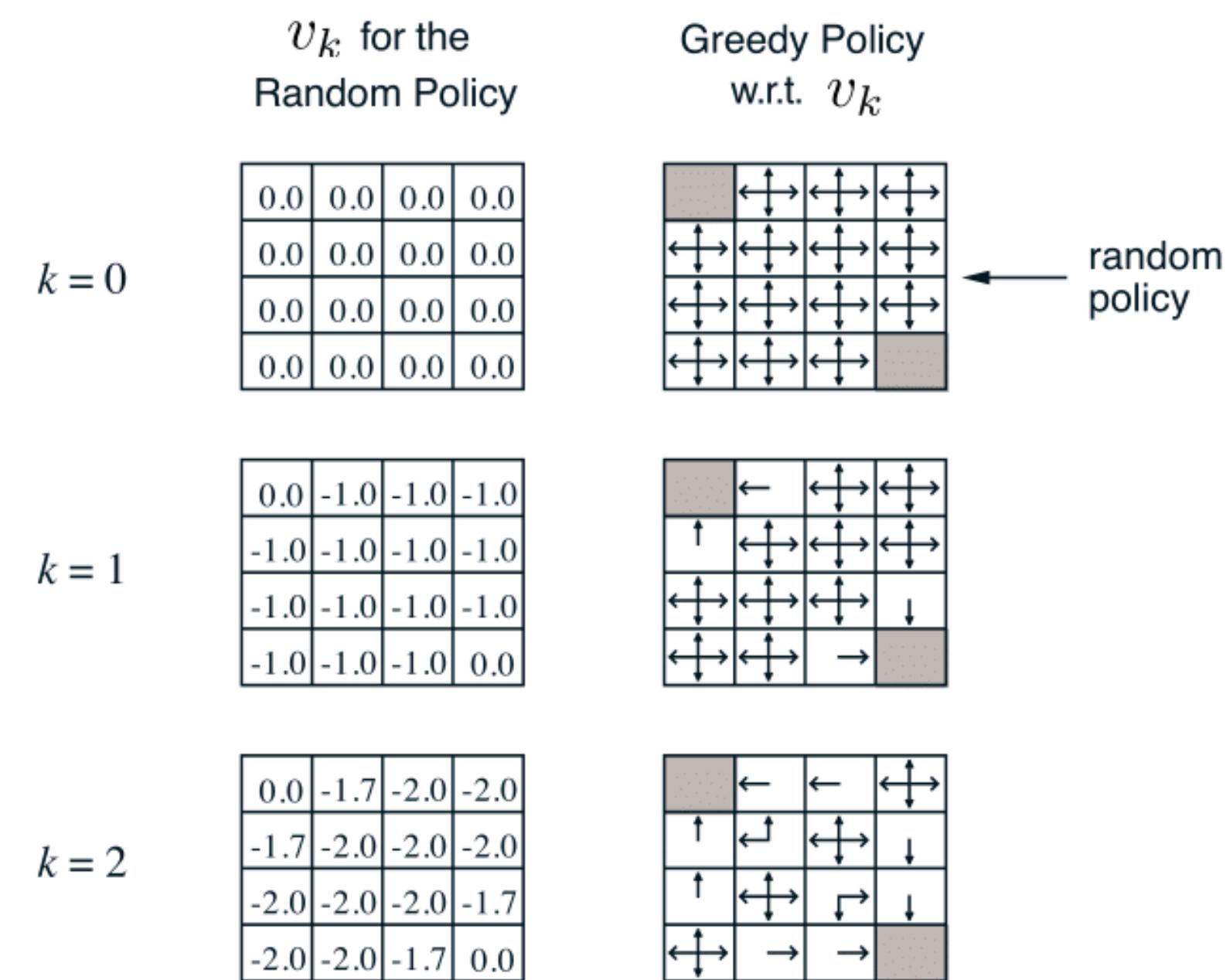
(задача)



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$r = -1$
on all transitions

Не выходи из комнаты (решение)



Не выходи из комнаты (решение)

$k = 3$

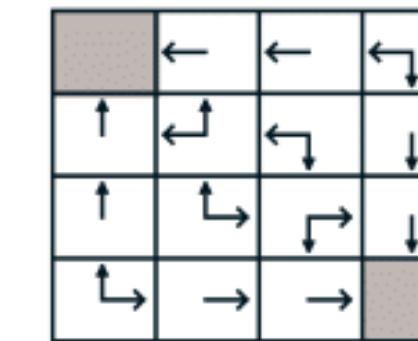
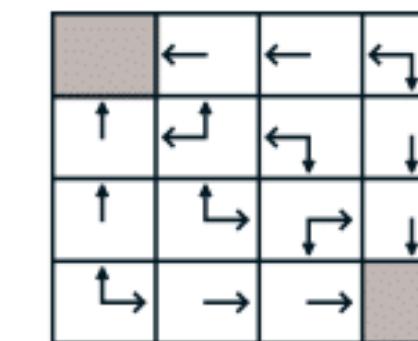
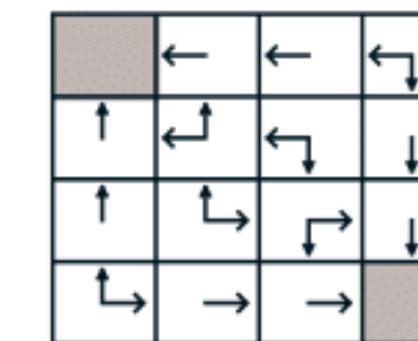
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



optimal
policy

Заметим, что...

- Политика называется жадной, если

$$\forall s \in S : \pi'(s) = \arg \max_{a \in A} [q_\pi(s, a)]$$

- Жадная (по отношению к случайной!) политика в нашем примере становится оптимальной уже на третьем шаге

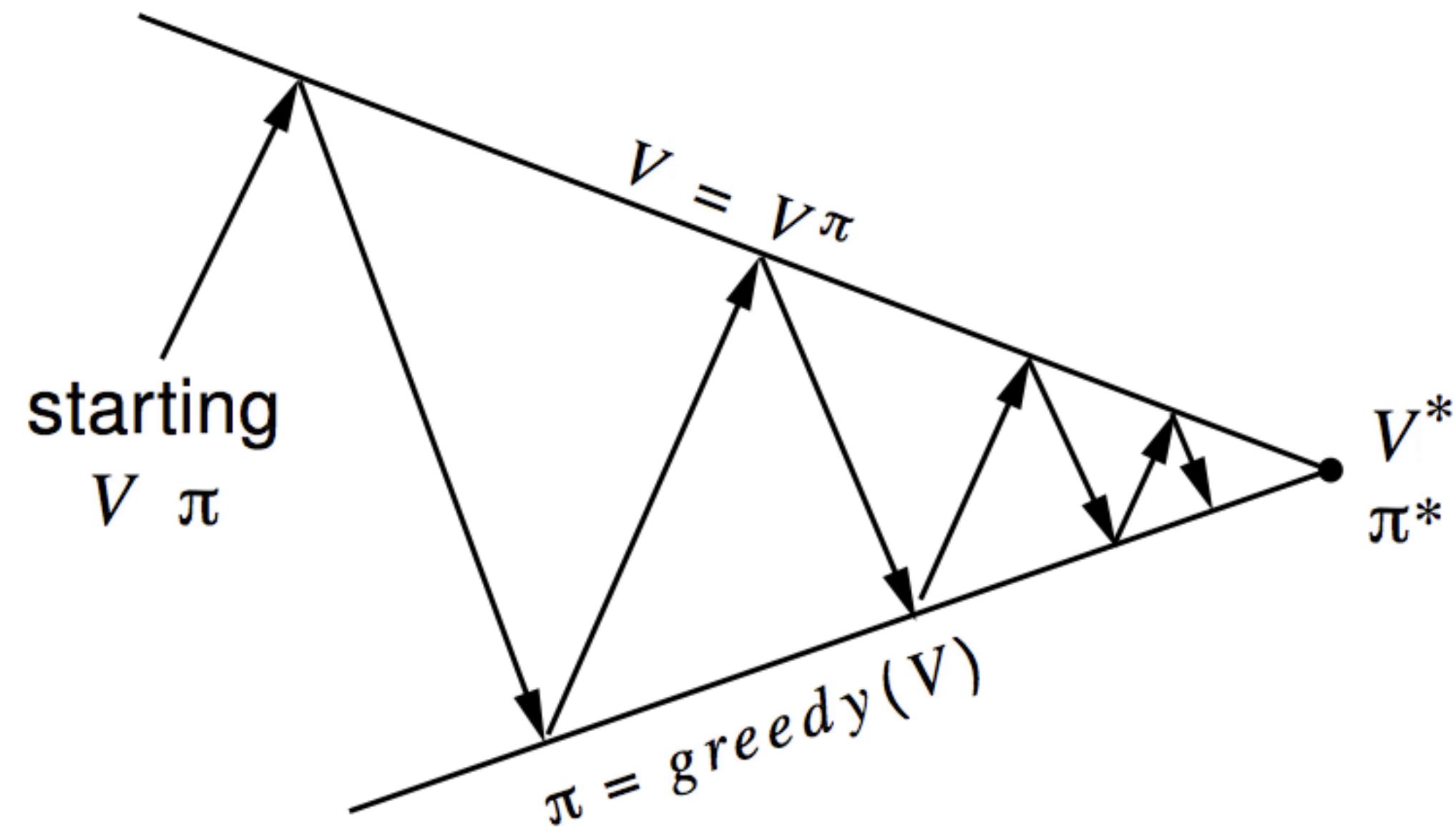
Policy iteration

- Возьмем какую-то политику
- Найдем её value function
- Построим политику, жадную, относительно предыдущей
- Покажем, что на одном шаге стало не хуже:

$$q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- По индукции $\forall s \in S : v_{\pi'}(s) \geq v_{\pi}(s)$. Great success!

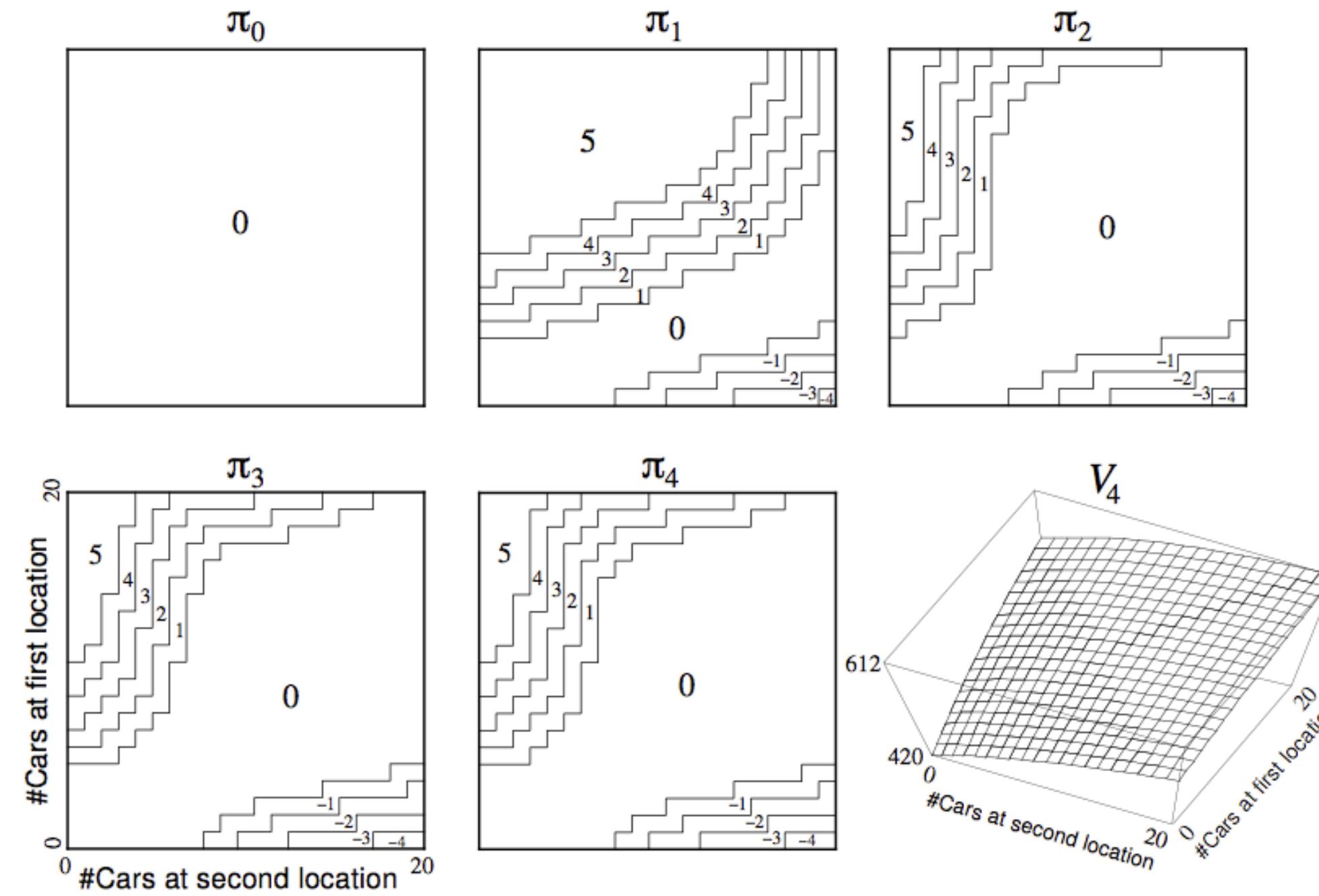
Wash, rinse, repeat



Ну как там с деньгами?

- Компания владеет двумя точками проката автомобилей, каждая вмещает не более 20 машин
- Каждый день автомобили арендуются (за \$10) и возвращаются случайным образом с распределением Пуассона
- В первой точке берут в среднем 3, возвращают в среднем 3
- Во второй точке берут в среднем 4, возвращают в среднем 2
- За ночь можно перегнать до 5 машин из одной точки в другую

Как с деньгами-то там?



Modified policy iteration

- Перейдем к следующей политике, если расстояние между value function на текущем шаге меньше ϵ
- Перейдем к следующей политике после k итераций policy evaluation

И то, и другое тоже сойдется. Доказывать мы это, конечно же, не будем.

Value iteration

(давайте без политики)

- Заметим, что $\pi^*(s) = \arg \max_{a \in A} [q^*(s, a)]$
- Вспомним Bellman optimality equation...
 - $q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a]$
- ...и снова превратим его в итеративное присваивание

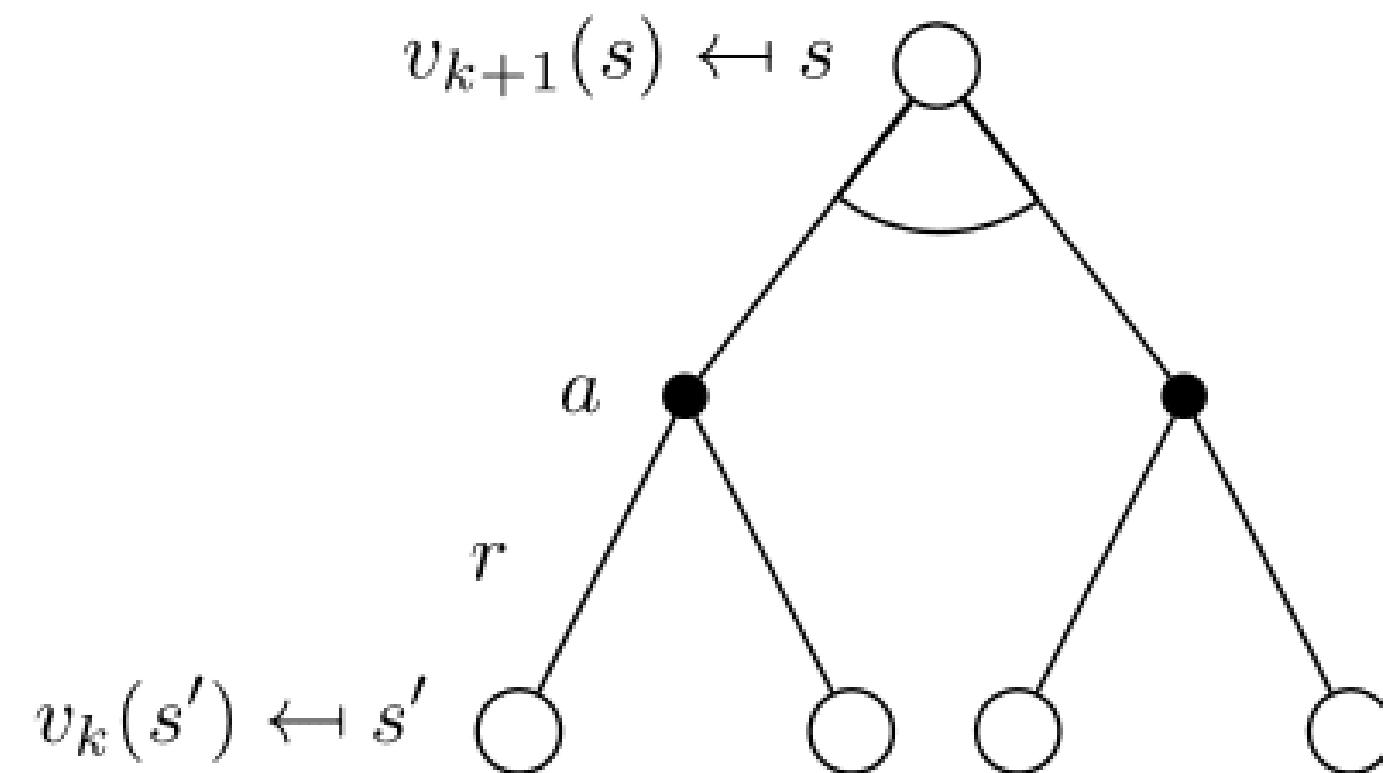
Value iteration

(алгоритм)

- $q_0(s, a) = 0, v_i(s) = \arg \max_{a \in A} [q_i(s, a)]$
- $q_{i+1}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_i(S_{t+1}) | S_t = s, A_t = a]$
- Неподвижная точка будет оптимальной action value function

Value iteration (интуиция)

- По сути это policy iteration с единственным шагом policy evaluation



Did you do it?

Агент с конечным числом состояний и действий, в полностью наблюдаемом MDP может обучиться действовать **абсолютно оптимально**.

What did it cost?

- Policy iteration: $O(|A||S|^2)$
- Value iteration: $O(|A|^2|S|^2)$
- $|S|$ растет экспоненциально с ростом количества параметров

Обучились подкрепились

- Обновляем value function inplace
- Проходим множество состояний или действий асинхронно
- Проходим состояния с помощью priority queue на основе, например, Bellman error: $|\max_{a \in A} [q_i(s, a)] - v_i(s)|$
- Обновляем только состояния, которые реально посетили
- Сэмплируем состояния, **модель не нужна!**

Сурс нужен

- "Spinning Up in Deep RL" by Open AI
- "Introduction to Reinforcement Learning" by Deep Mind x UCL
- "Sequential decision making and dynamic programming" by Moritz Hardt and Benjamin Recht