Deep Reinforcement Learning
Building Blocks

Arjun Chandra
Research Scientist
Telenor Research / Telenor-NTNU AI Lab
arjun.chandra@telenor.com
@boelger

8 November 2017

https://join.slack.com/t/deep-rl-tutorial/signup

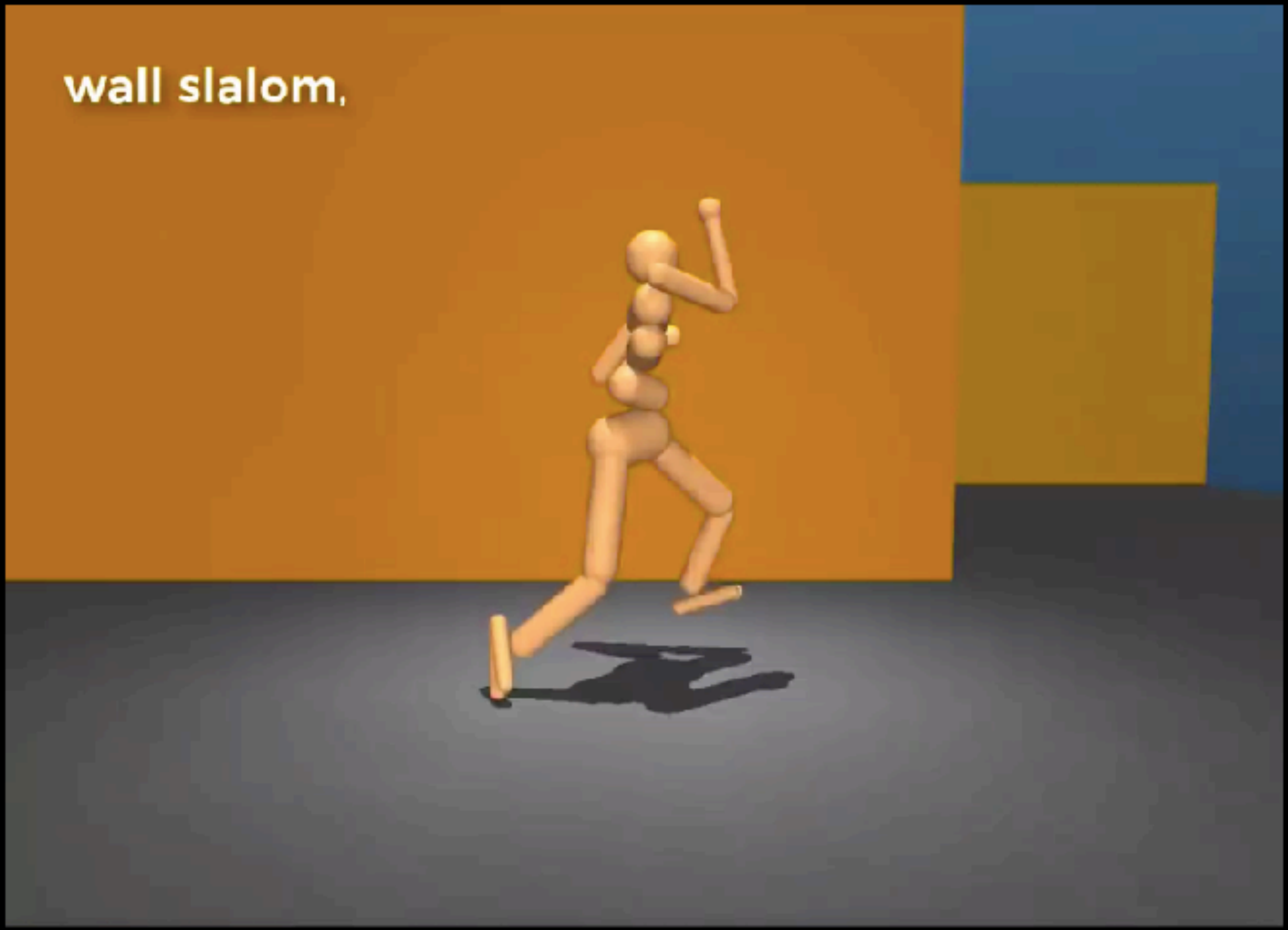
The Plan

- The Problem
- (deep) RL Concepts by Example
- Problem Decomposition
- Solution Methods
  - Value Based
  - Policy Based
  - Actor-Critic

how to make
decisions over time
to maximise my
return / "long term reward"?

http://cs.stanford.edu/groups/littledog/

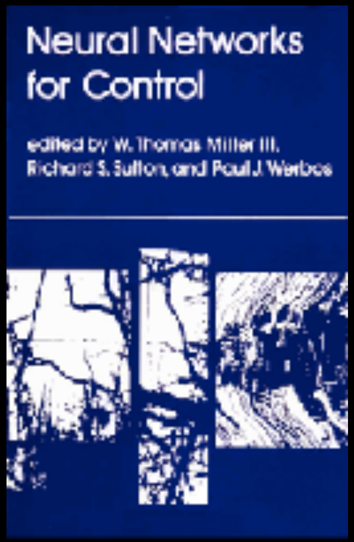
emergence of locomotion



https://deepmind.com/blog/producing-flexible-behaviours-simulated-environments/
https://www.youtube.com/watch?v=hx_bgoTF7bs
https://arxiv.org/abs/1707.02286

As we know...

Rich Sutton et al.



late 1980s

RL for robots using NNs, L-J Lin. **PhD 1993, CMU**

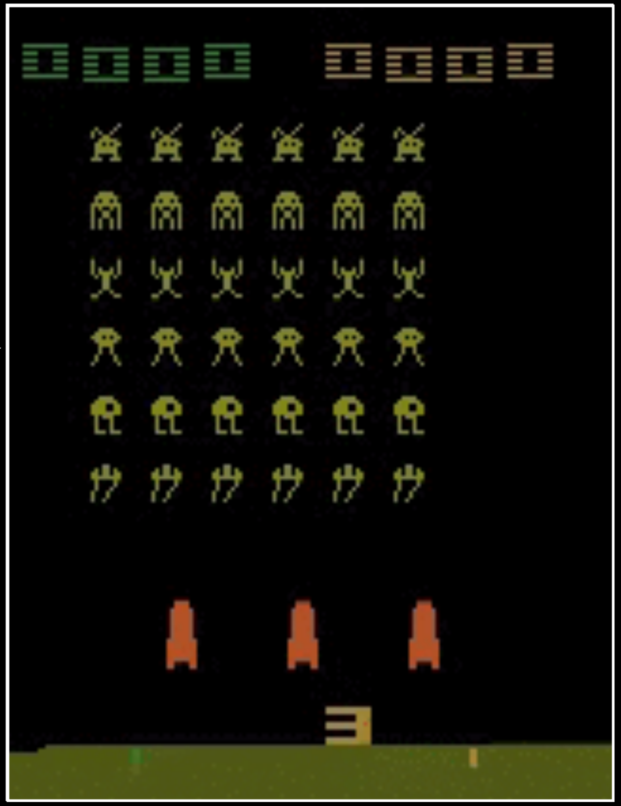
Gerald Tesauro

1995

Stanford

2004

Vlad Mnih et. al.



2013 —

Google DeepMind

David Silver et. al.

2015 —

Problem Characteristics

dynamic

uncertainty/volatility

uncharted/**unimagined**/
exception laden

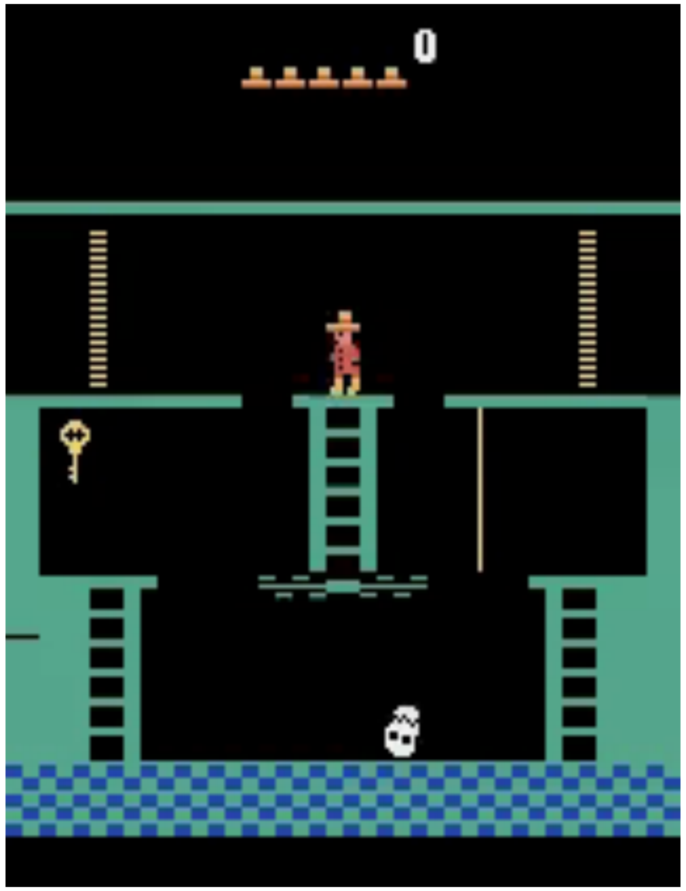
delayed consequences

requires **strategy**

Image credit: http://wonderfulengineering.com/inside-the-data-center-where-google-stores-all-its-data-pictures/

Solution

machine with **agency** which **learn**, **plan**, and **act** to find a strategy for solving the problem



autonomous to some extent

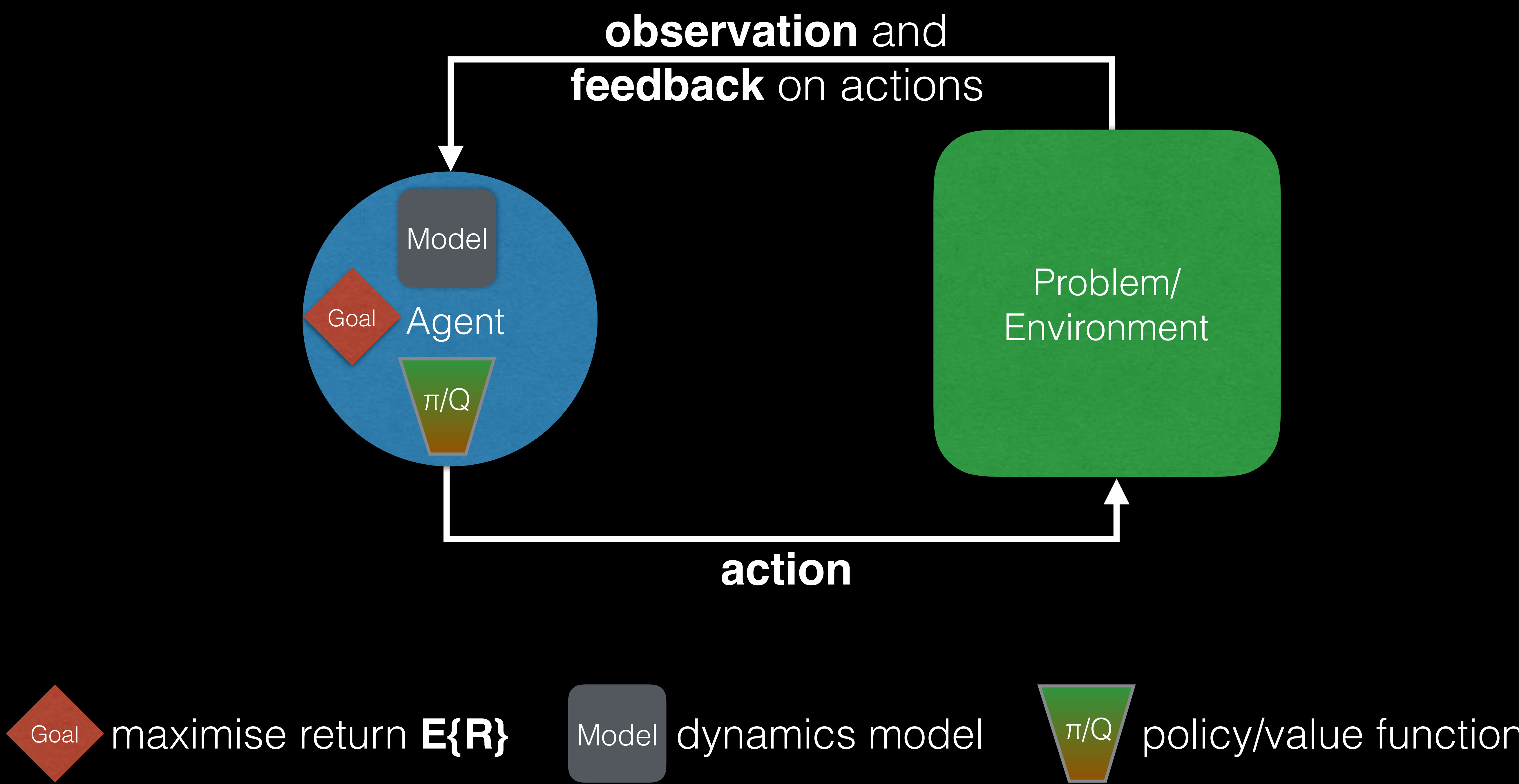
probe and **learn from feedback**

focus on the **long-term objective**

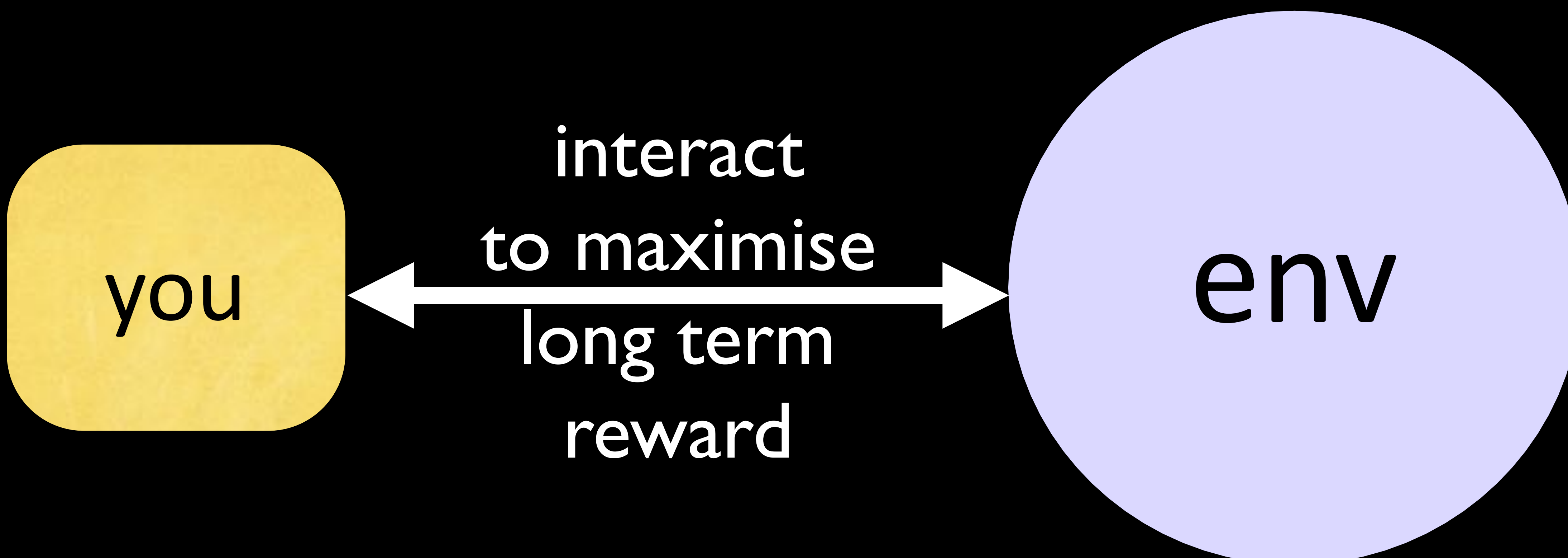
explore and **exploit**

what is the
sequence of actions
I could **take** to maximise my
return / “long term reward”?

Reinforcement Learning



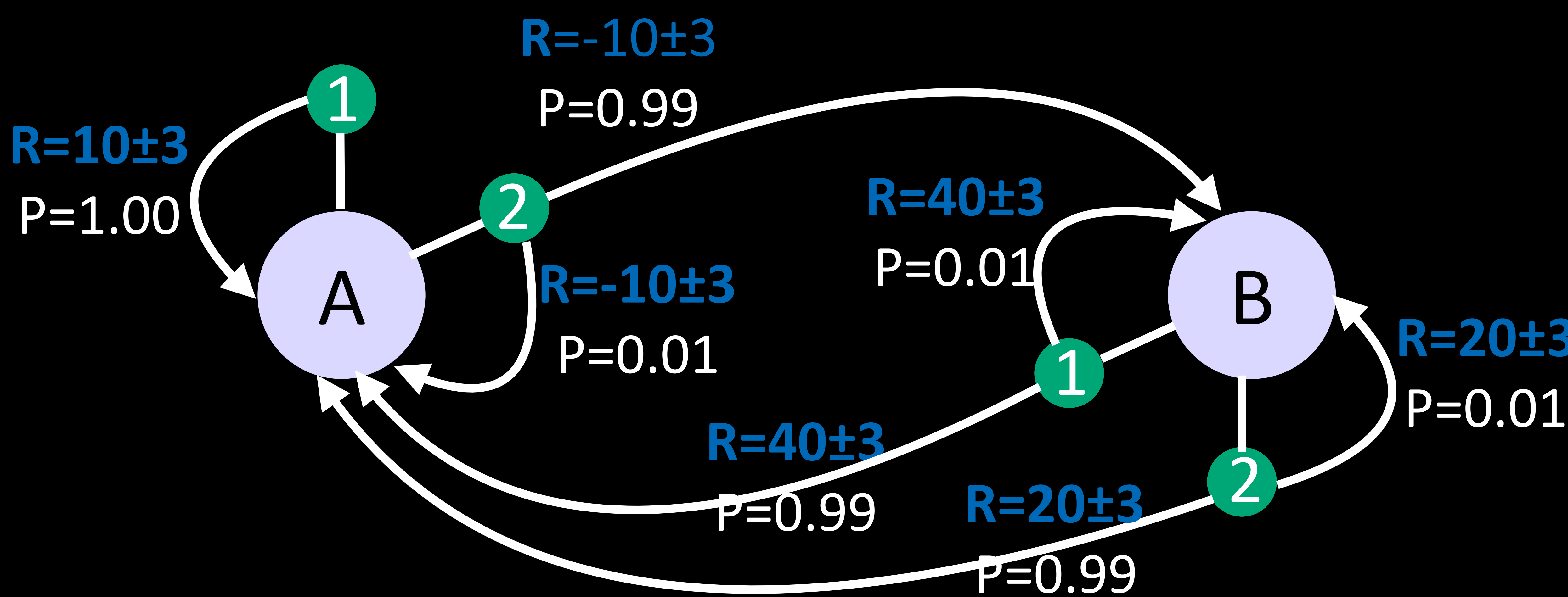
the excruciatingly awesome MDP game!



Inspired by Rich Sutton's tutorial:
https://www.youtube.com/watch?v=ggqnxyjaKe4

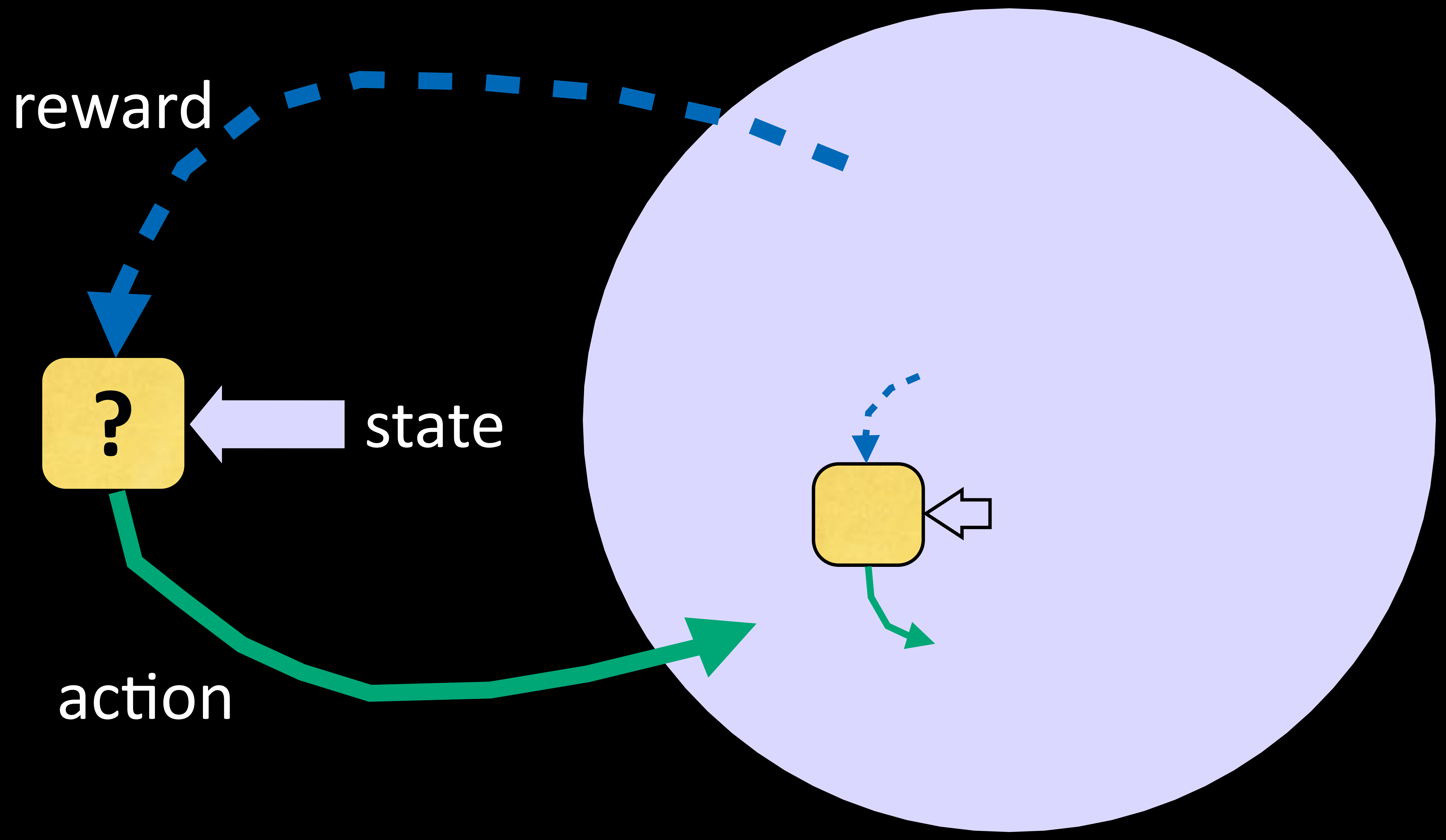
the MDP (S,A,P,R,ϒ)

R: immediate reward function R(s, a)
P: state transition probability P(s'|s, a)

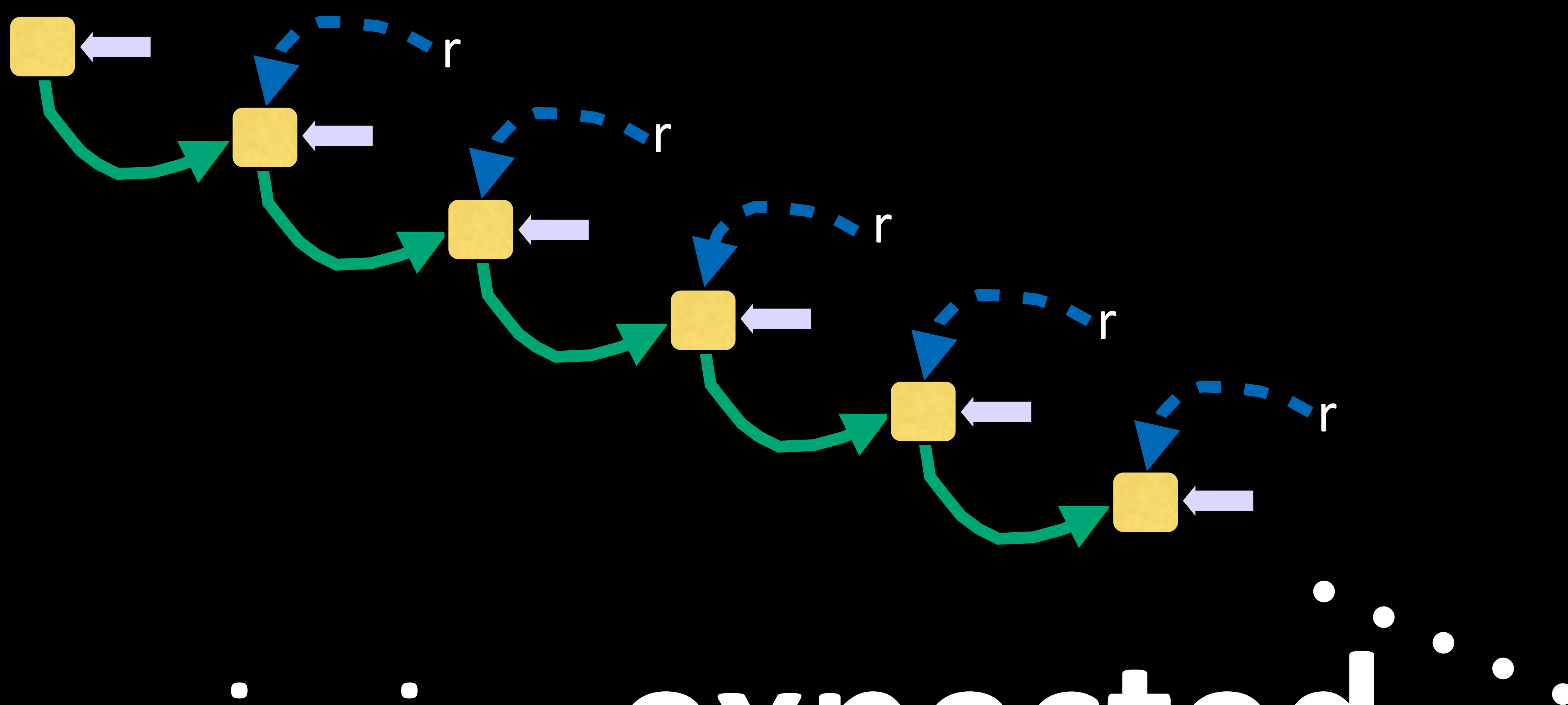


https://github.com/traai/basic-rl

the problem
(cartoon of an MPD)

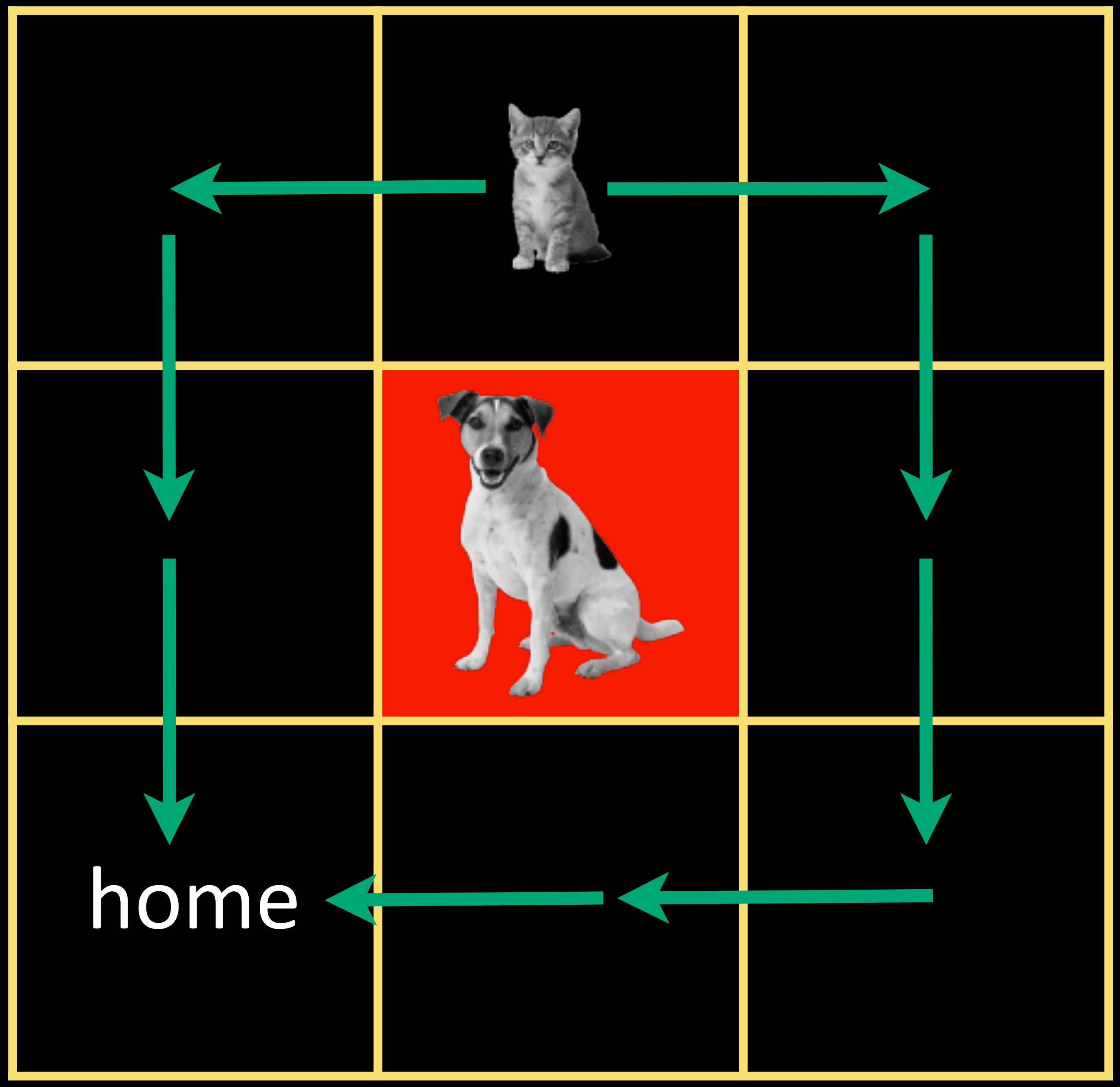


agent's job/goal?



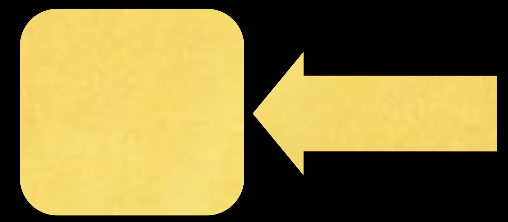
maximise **expected** cumulative reward/**return**

toy problem



state and action spaces

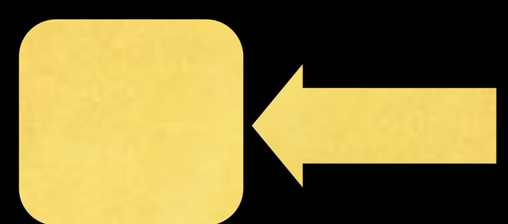
- size of these spaces can be quite large
- specifying the spaces is crucial in designing a good learning agent



5 integer values between
1 and 100: {22,44,12,67,9}

size of state space = 100 x 100 x 100 x 100 x 100

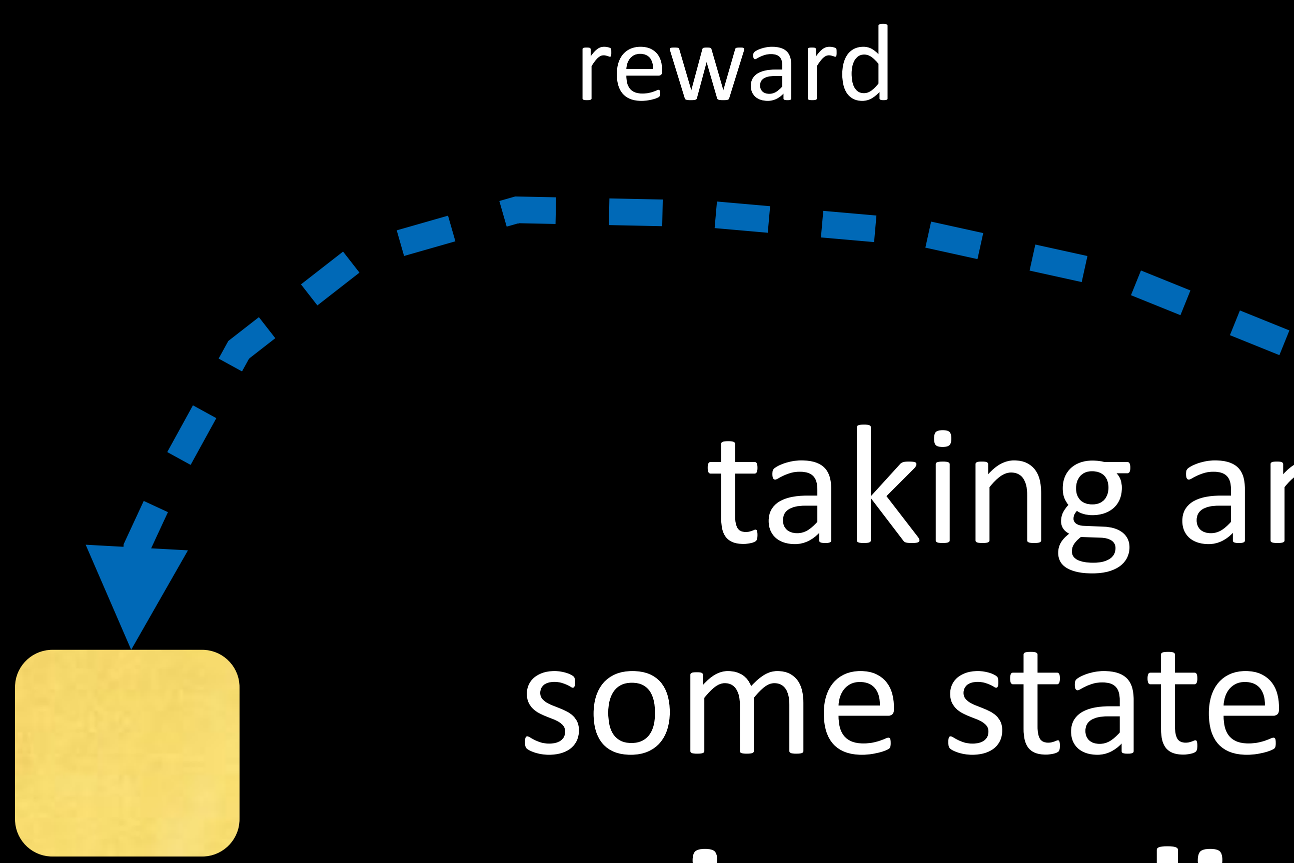
can quantise state space differently



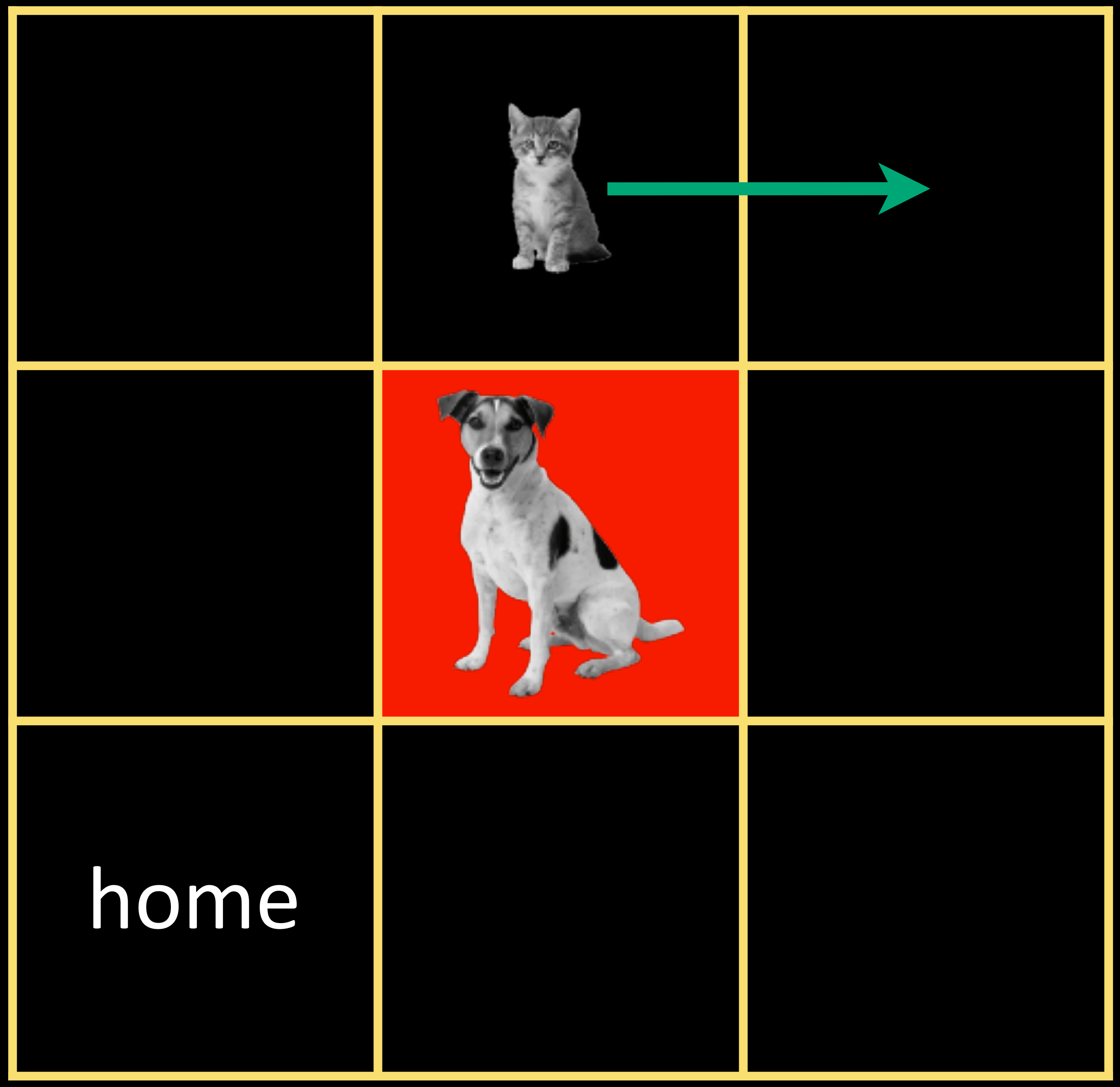
5 values belonging
to 2 classes: {1, 2, 1, 2, 1}

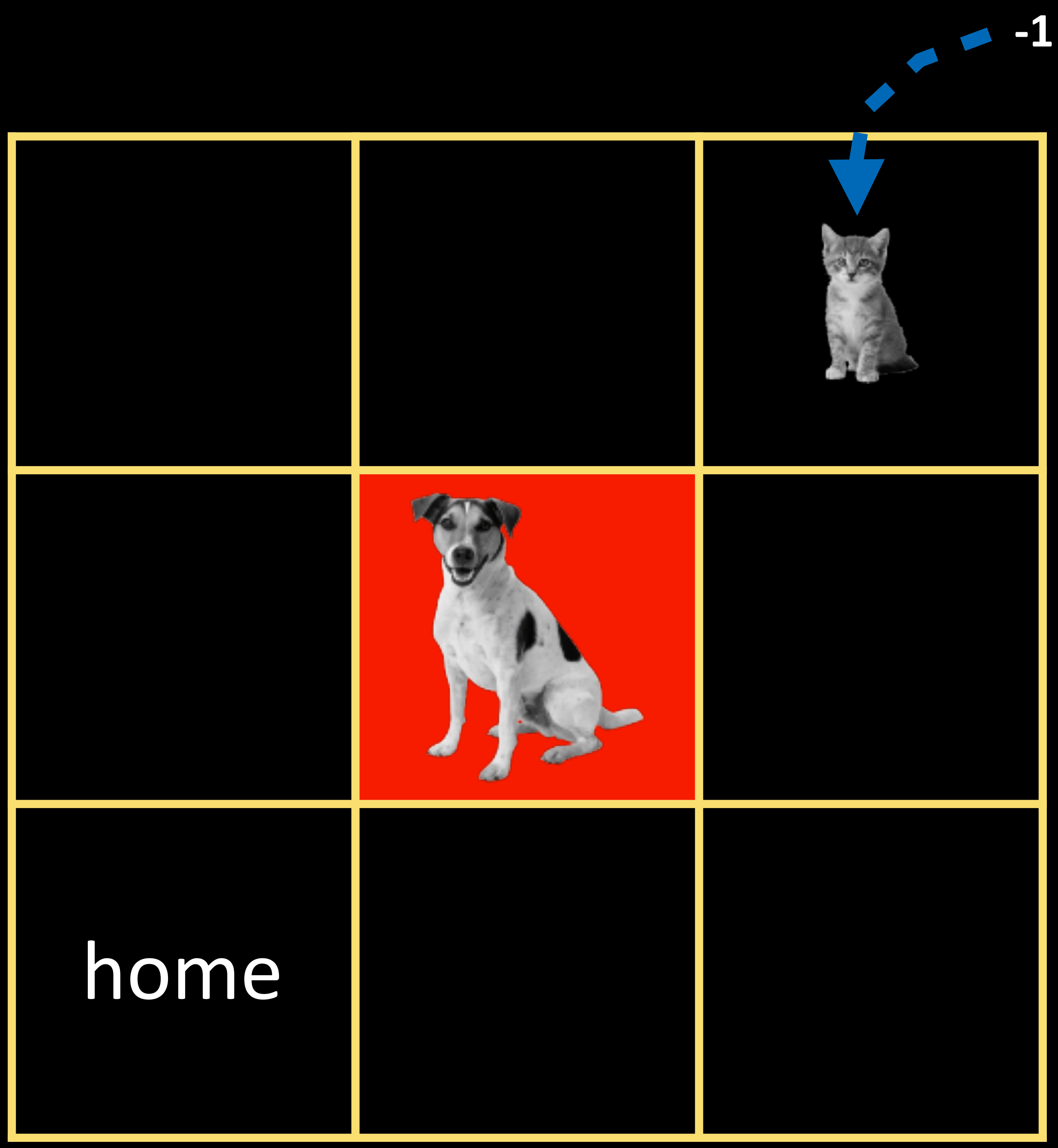
size of state space = 2 x 2 x 2 x 2 x 2

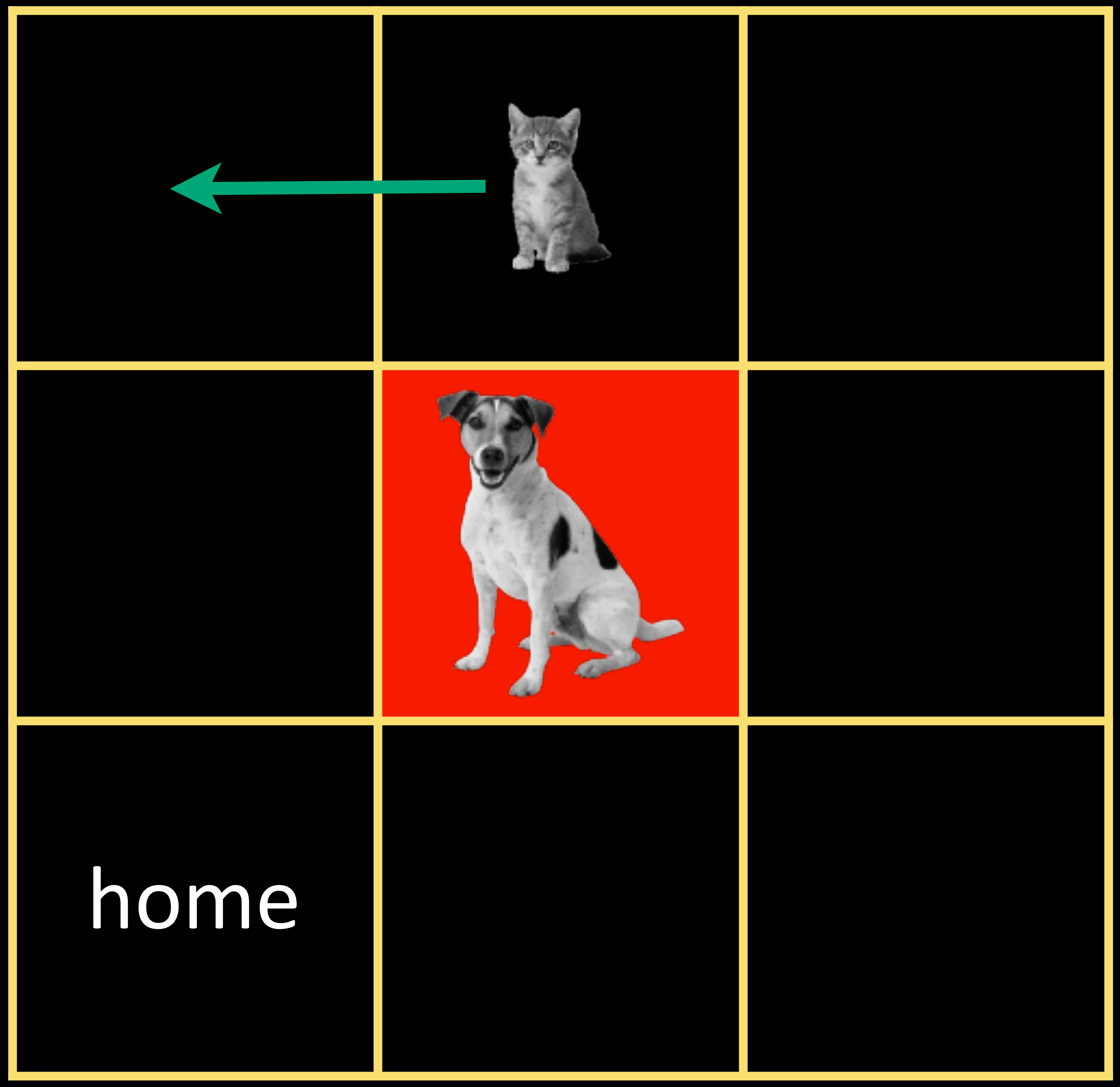
in the toy problem? 9

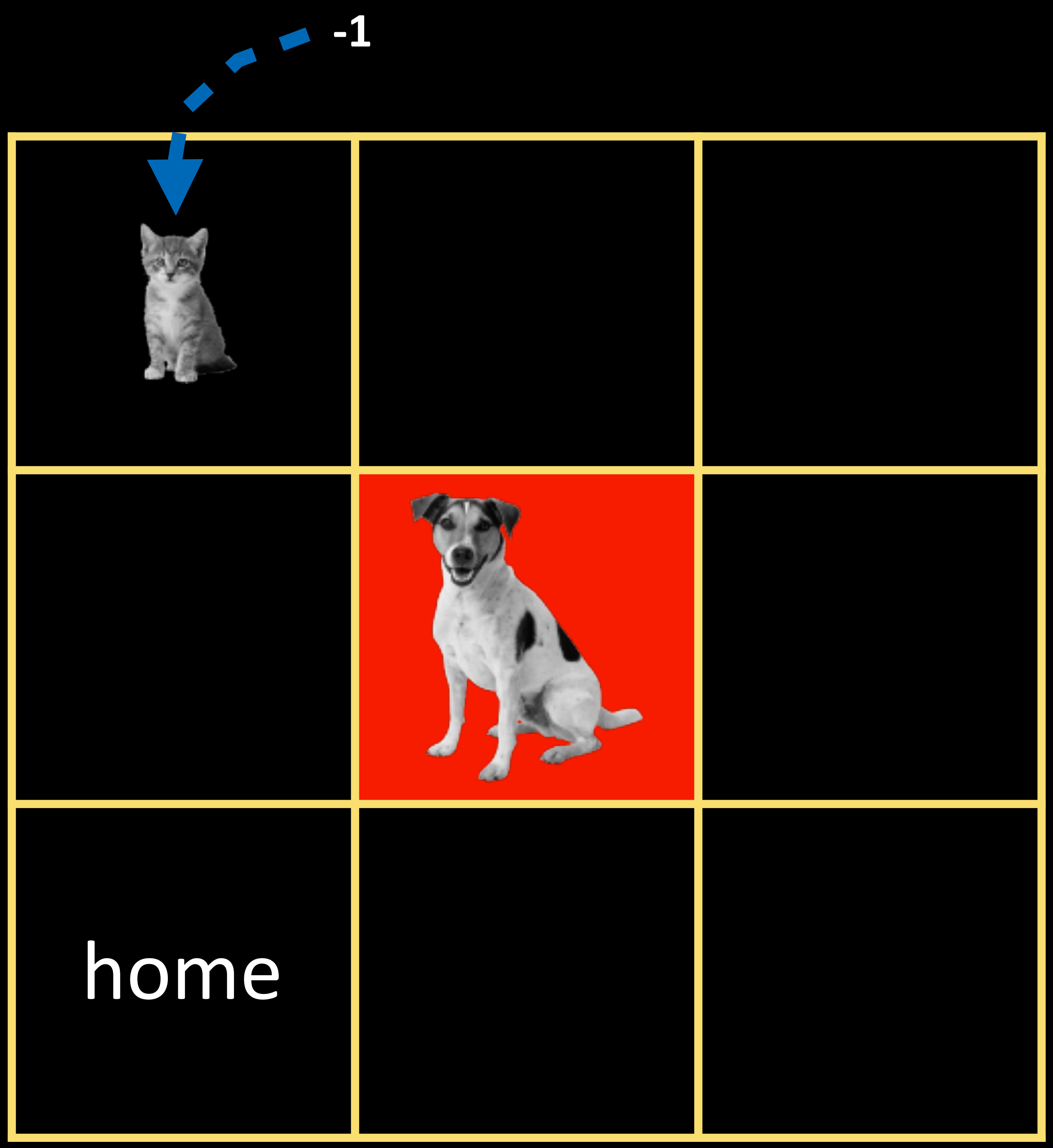


taking an action in
some state results in an
immediate reward
(can be negative)


home


-1
home


home

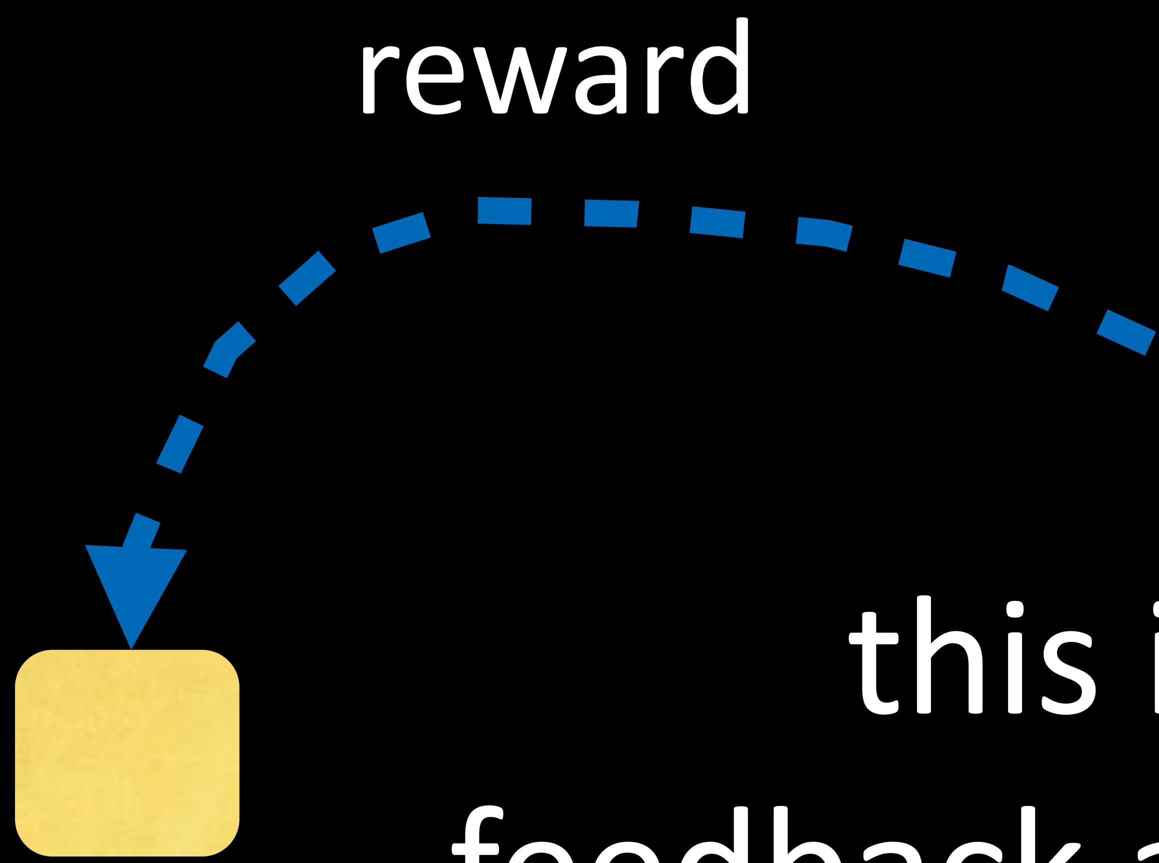

-1
home


>-1
home
what if the cat were to start here some other day?

reward system should tell
the agent:

what to achieve

rather than how to achieve



this is all the
feedback an agent gets!

immediate!

but agent has to choose an action based on **expected return**

expected return

?

task

episodic

(there is an **end**)

continual

(there is no **end**)

episodic

(there is an **end**)

agent taking **finite (say 5) steps** till the end...

should act based on the

e.g. average of the following

$$R_0 = r_1 + r_2 + r_3 + r_4 + r_5$$

continual

(there is no **end**)

agent can continue acting for **infinite steps in time...**

should **discount** future rewards and act based on e.g. **average of the following**

$$R_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + \cdots$$

discount

future reward is probably **more uncertain** than **immediate reward**

shortsighted? $\gamma = 0$

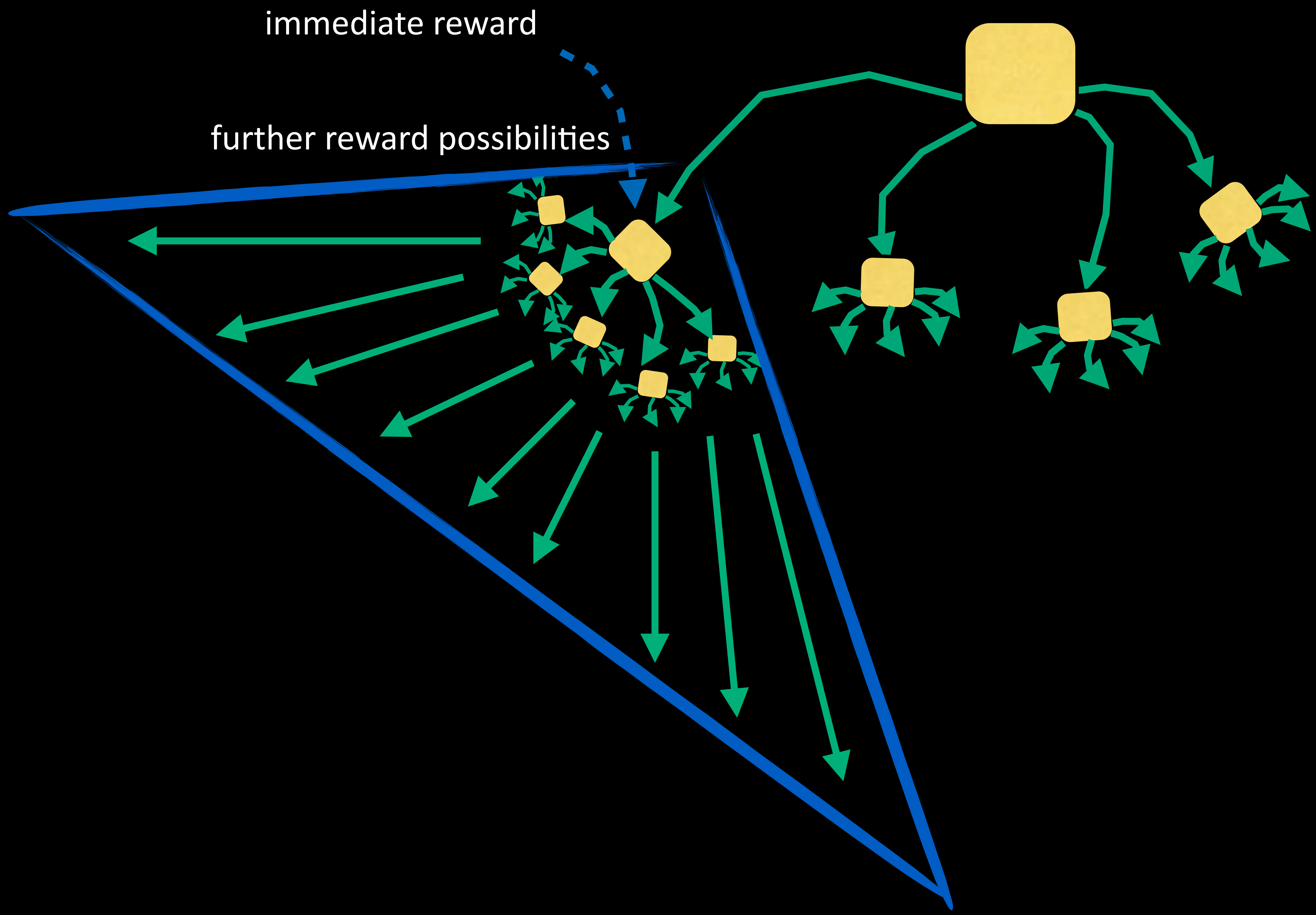
$$0 \le \gamma \le 1$$

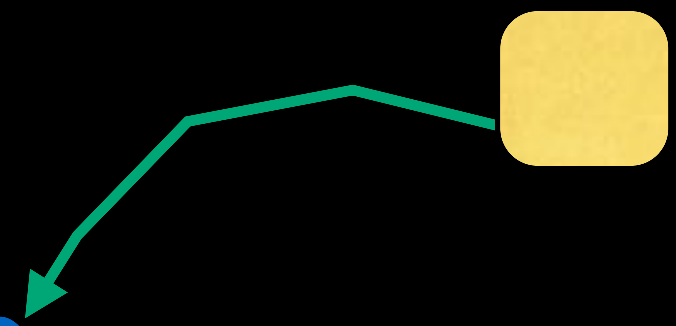
farsighted? $\gamma = 1$

$$R_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + \cdots$$

$$R_0 = \sum_{k=0}^{T} \gamma^k r_{k+1}$$

$$\mathbf{E}\left\{ R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \right\}$$


immediate reward
further reward possibilities



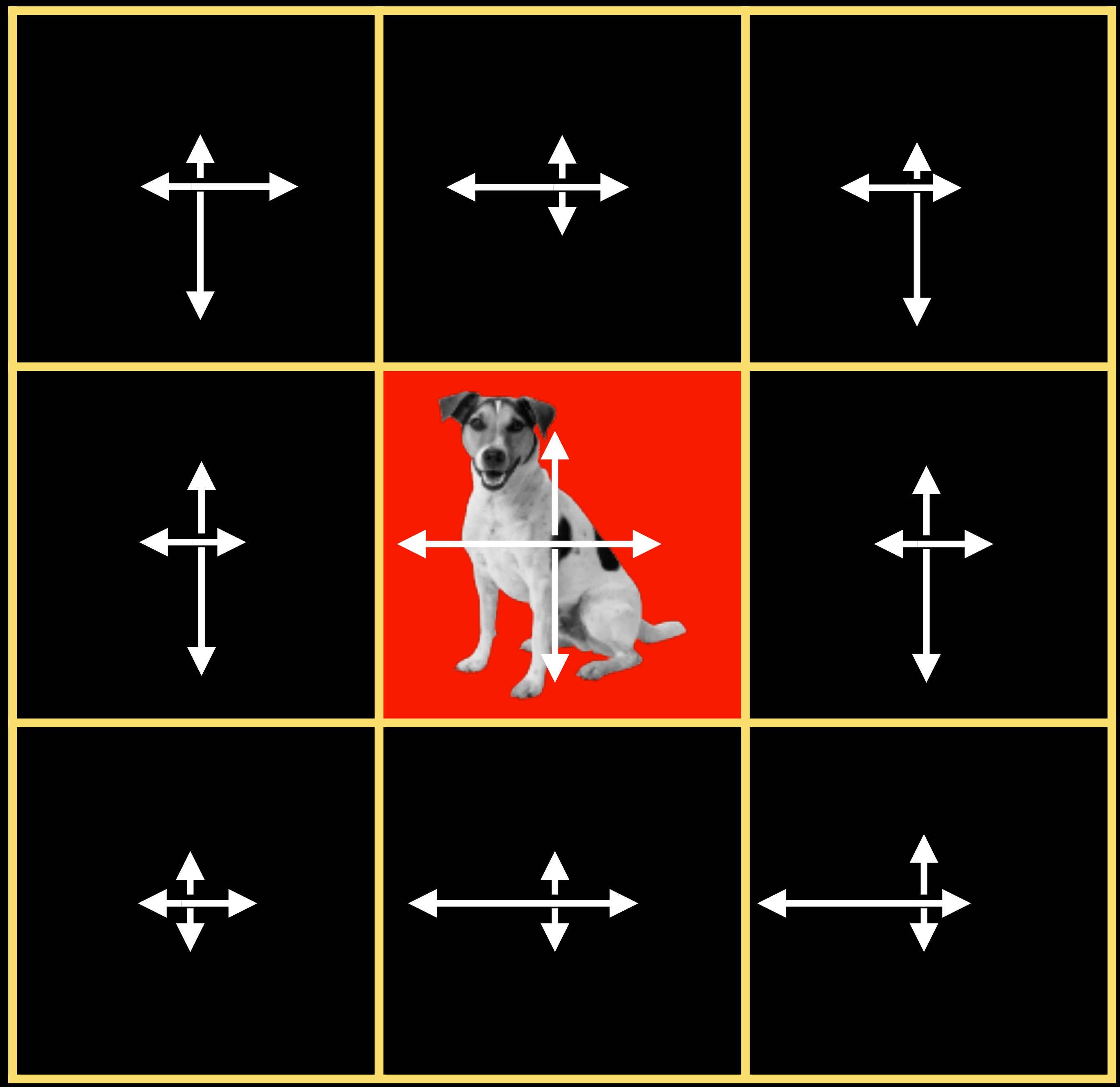
expected return

$$\mathbf{E}\left\{ R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \right\}$$

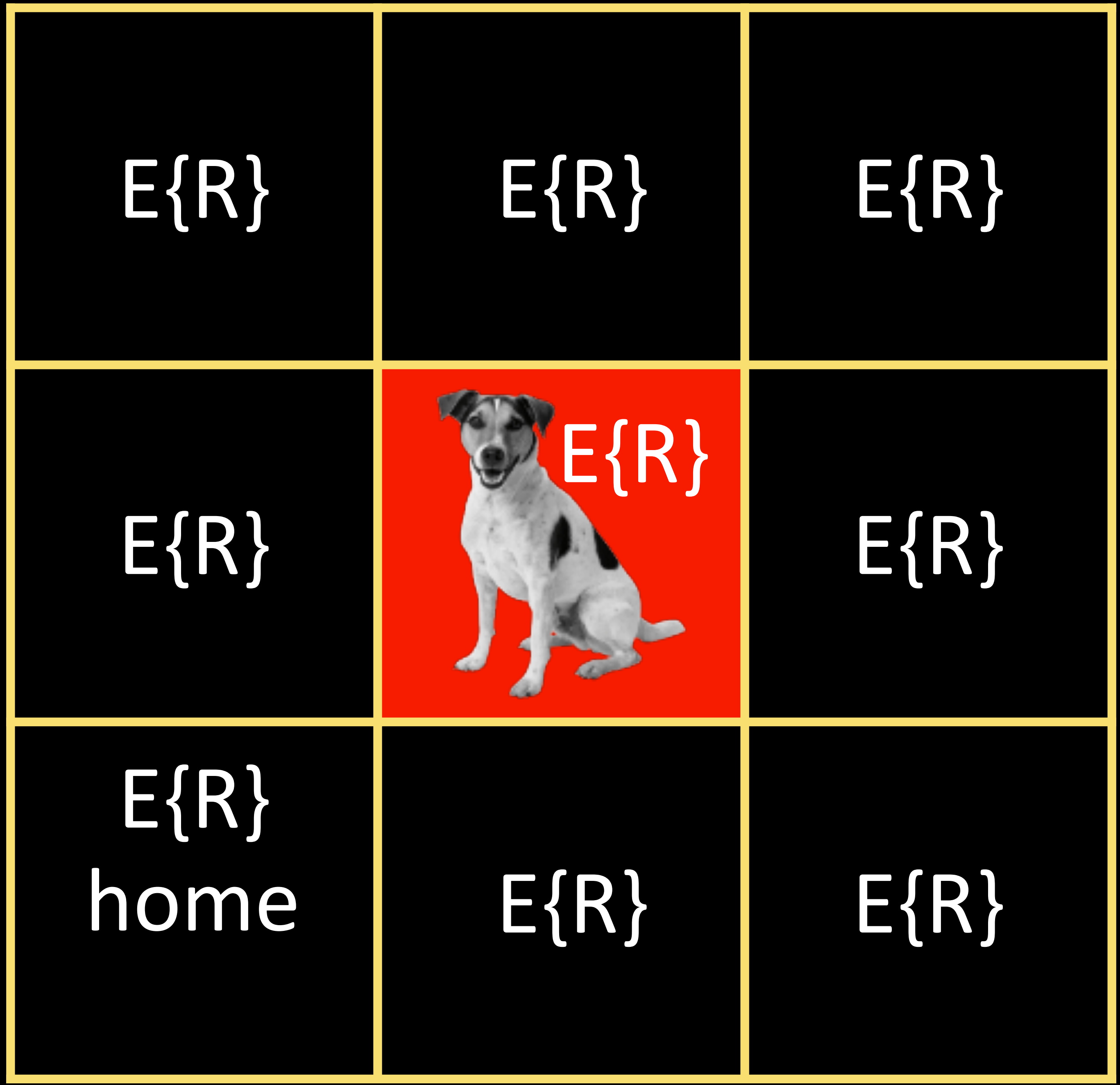
but these expected returns are **not known to agent** beforehand!

what knowledge might the agent try to acquire to behave properly?

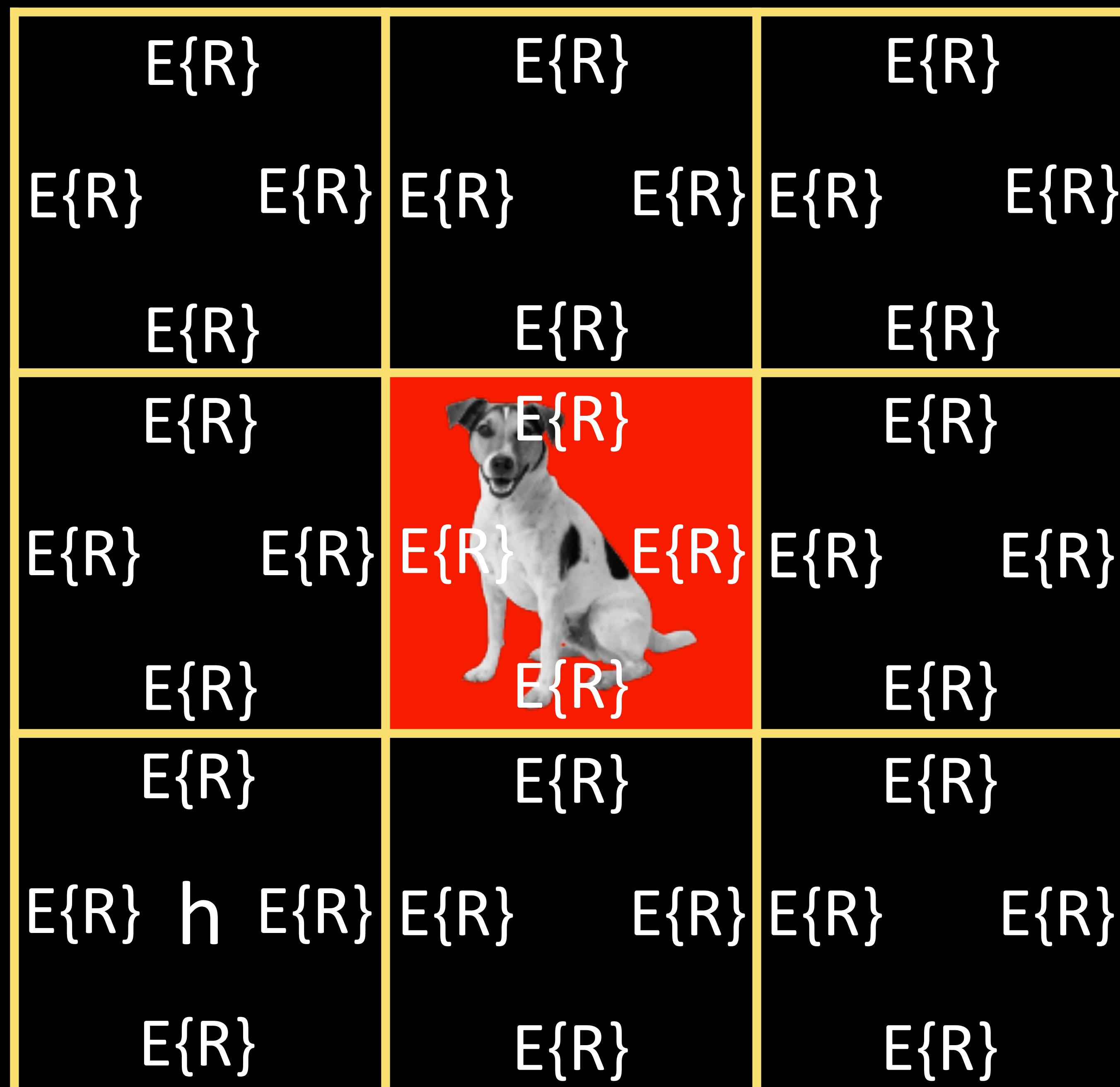
ranking/probability of an action in some state bringing max expected return (long term value)?



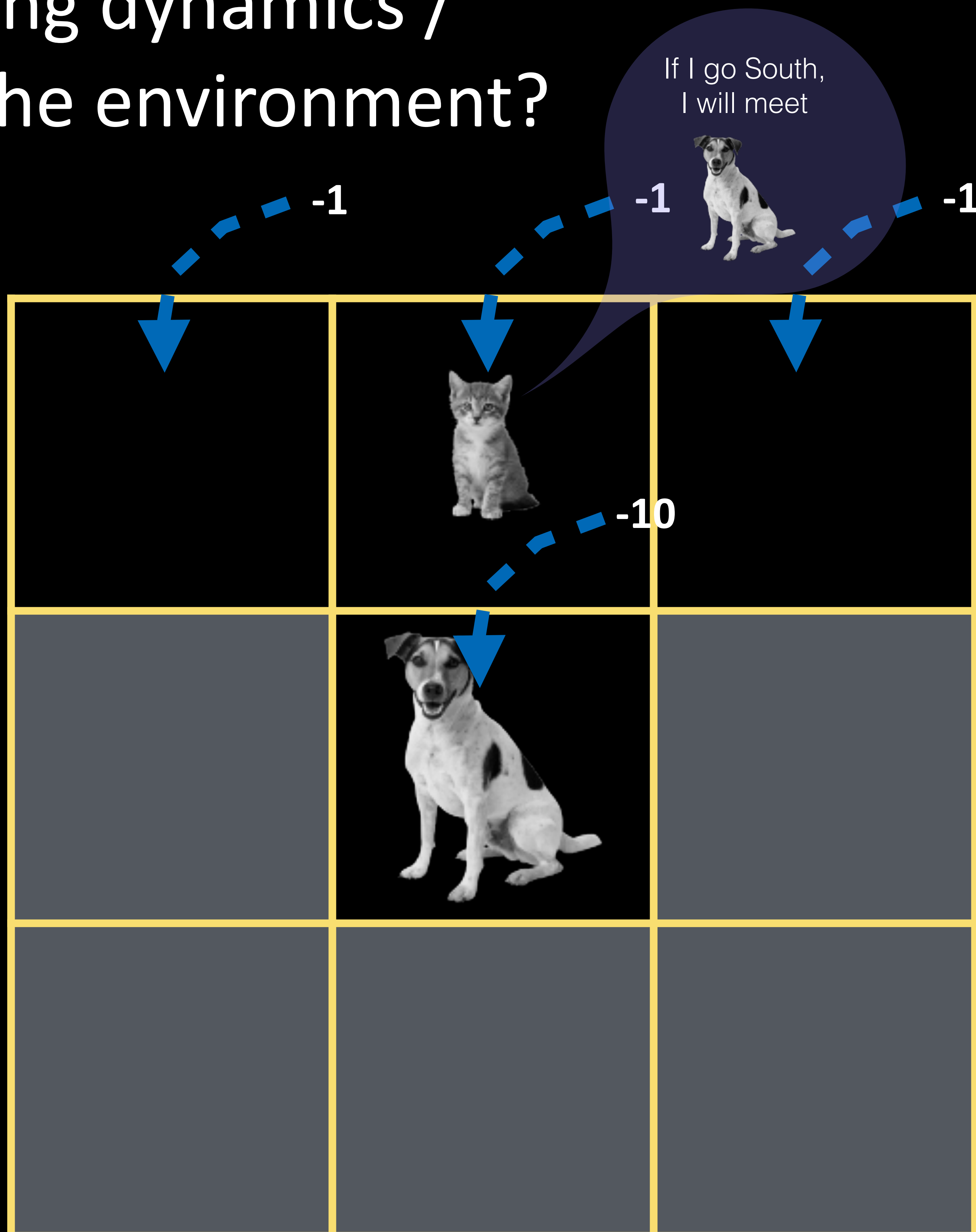
expected long term value of
being in each state,
under some action selection scheme?



expected long term value of taking
some action in each state, then behaving
using some action selection scheme?



modelling dynamics / mapping the environment?



prediction problem

learn to predict expected long term reward/value

control problem

learn to find the **optimal action selection scheme/policy**

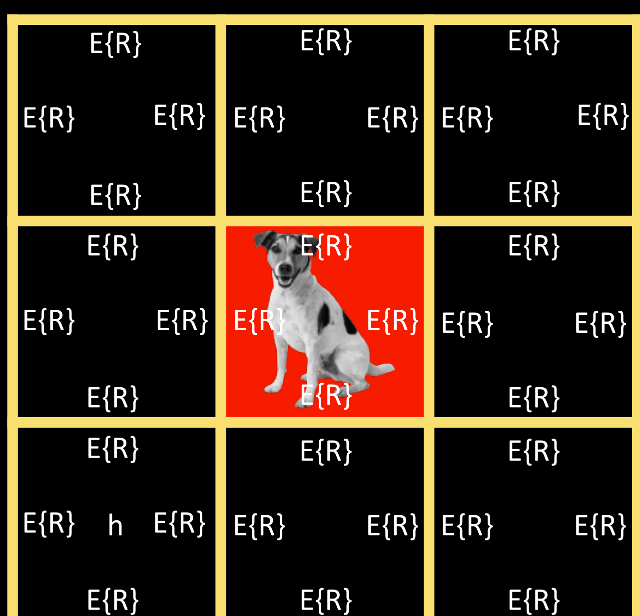
p**olicy**: action selection

value: how good is an action/state

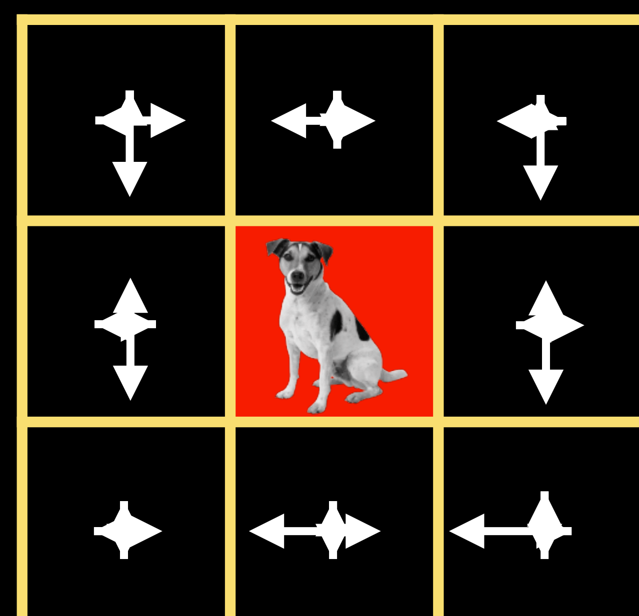
model: predict next state/reward to look ahead/plan

types of RL agents?

value based



policy based



both **value and policy**

model based

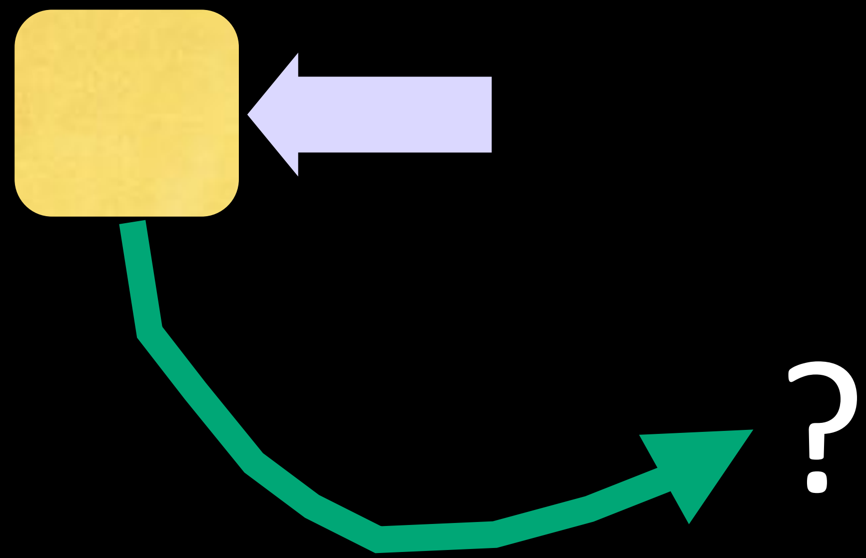
value/policy

+

model of dynamics

we will focus on value based RL in the first half

action selection?



expected return for carrying out an action is its **value**

values of each possible action in the current state helps select actions!

policy can be derived from value

(e.g. act greedily)

but **what are these values?**

<<expected returns unknown>>
<<actions based on unknowns>>

value can be estimated by

sampling environment

while acting using some policy

e.g. act, accumulate new
reward (ground truth), and update

Q

	a	b	c
1	2	0	1
2	3	0	-1
3	-5	6	2
4	2	3	1
.	.	.	.
.	.	.	.
.	.	.	.
n	7	8	7

agent maintains **values**
for actions within each state

selects actions using these values under some
“policy”

V

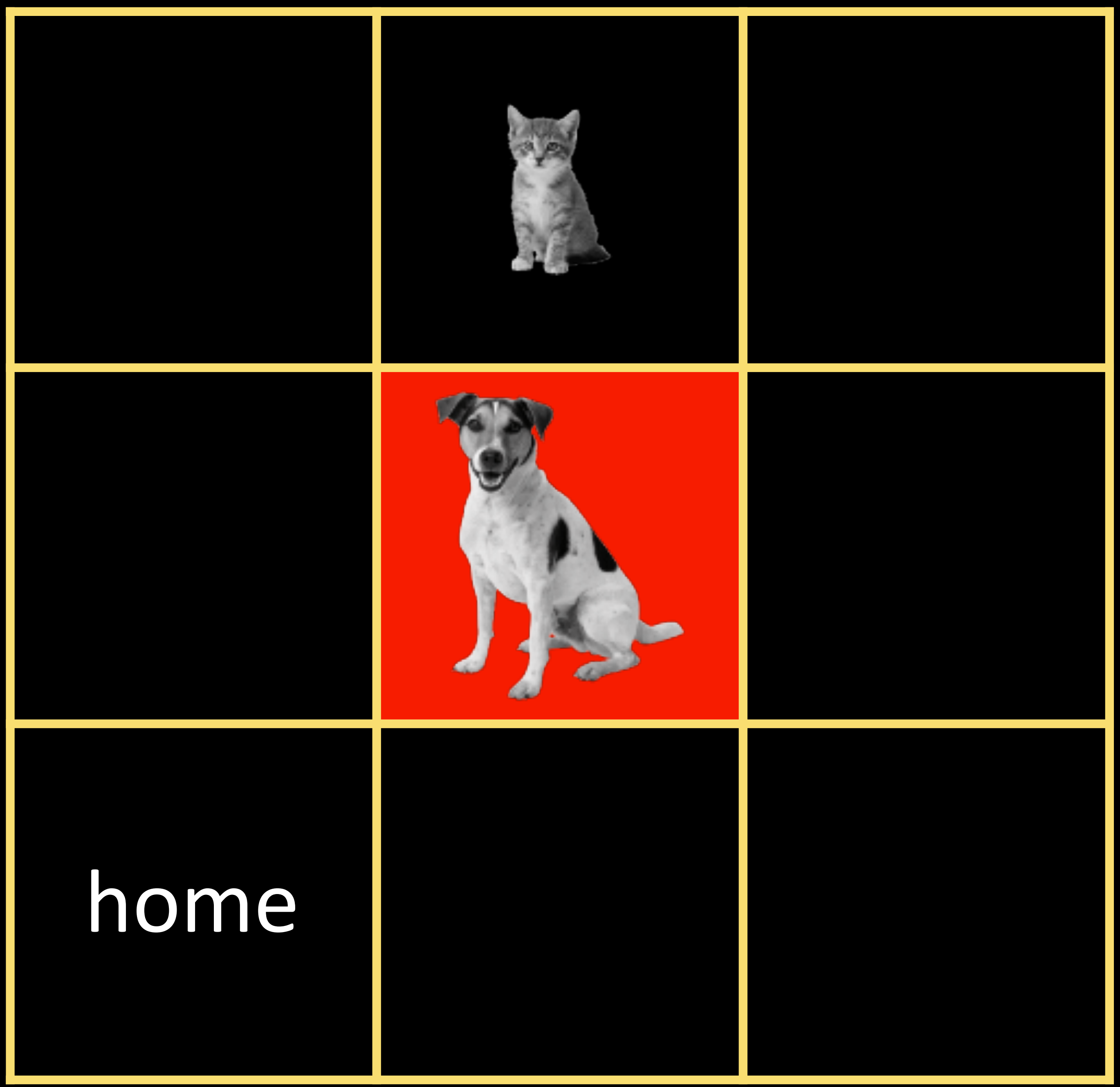
1	2
2	3
3	-5
4	2
.	.
.	.
.	.
n	7

agent **maintains state values**

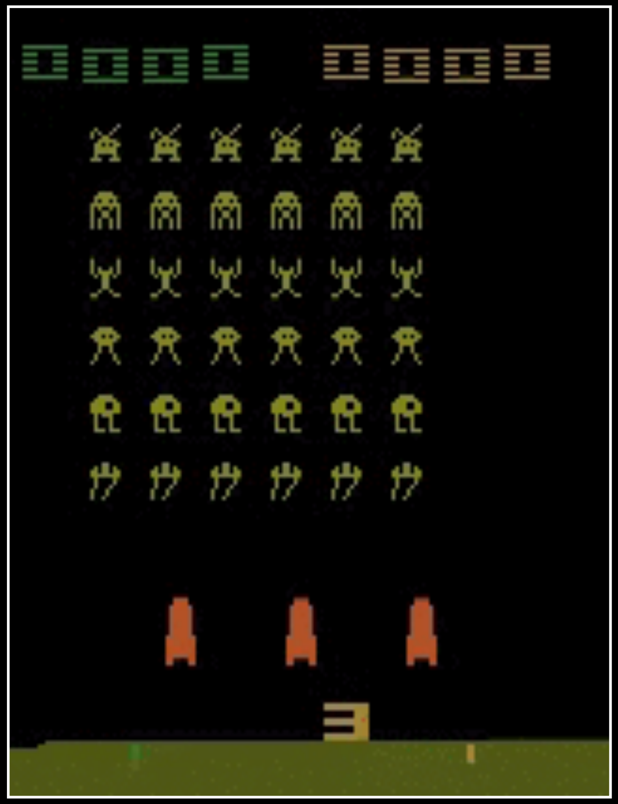
selects actions using these values under some

“policy”

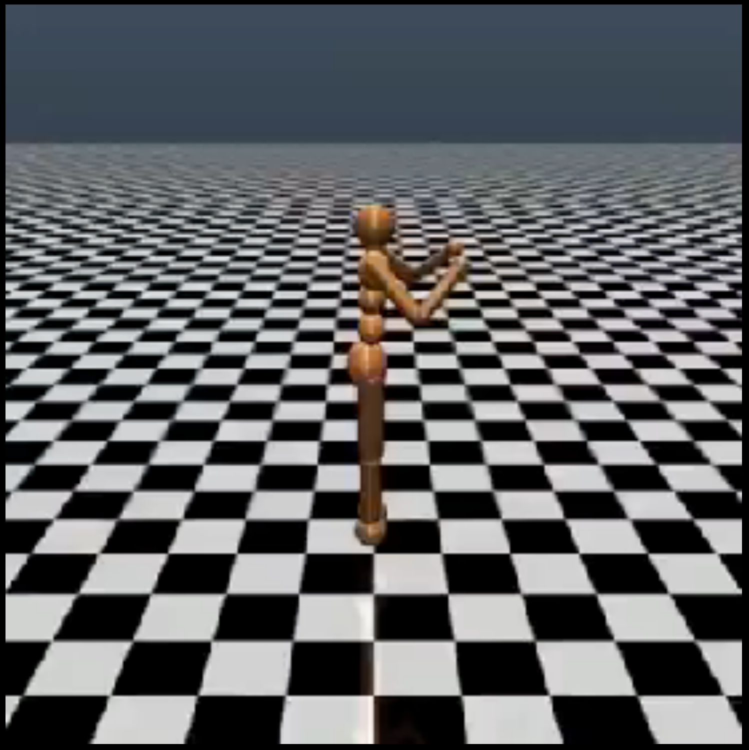
but… agent needs a **model of the environment!**



9 states



10^16992 (pixels)
10^308 (ram)

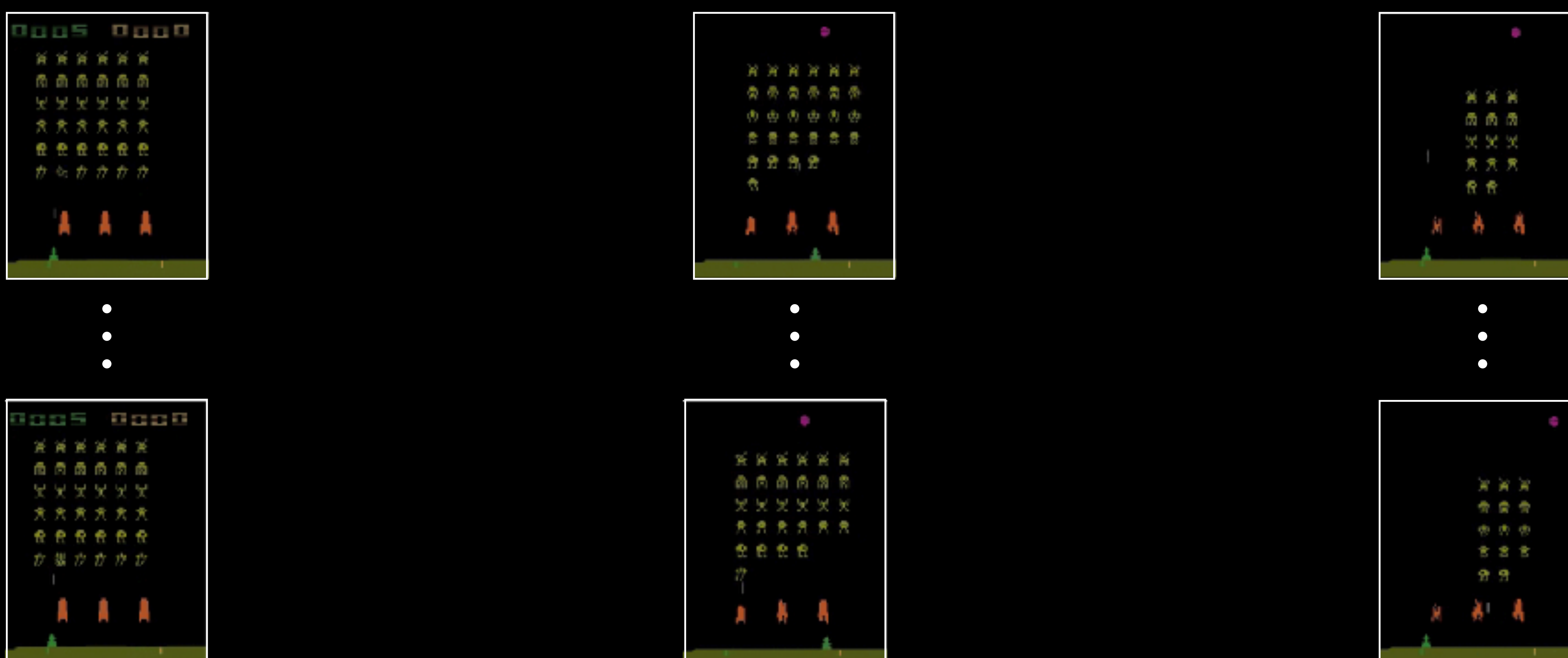


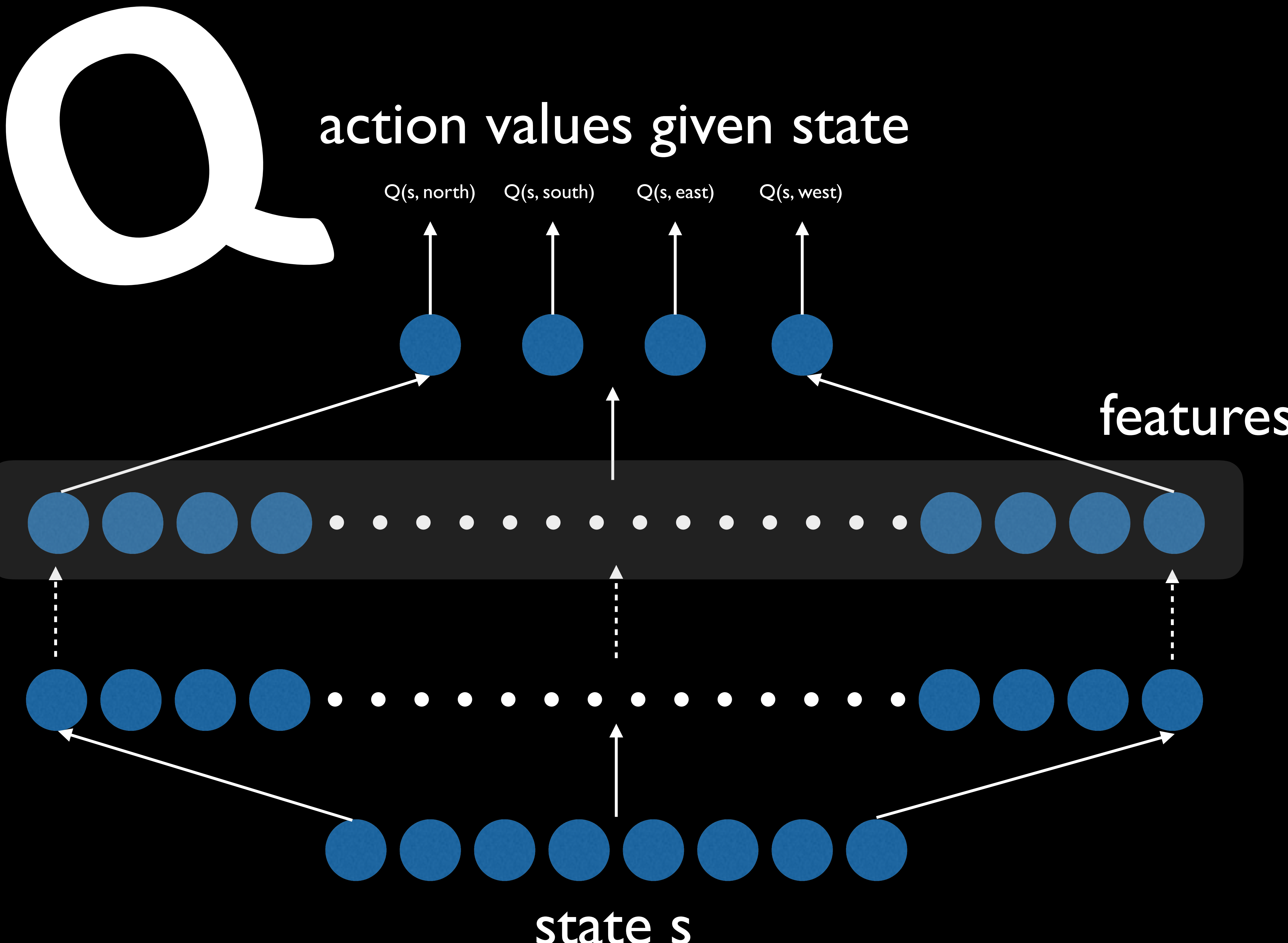
continuous!

extract **features**

that help

generalise across states




Q
action values given state
Q(s, north)
Q(s, south)
Q(s, east)
Q(s, west)
features
state s

Q V

$$E\{R_t\}$$

policy?

probability of choosing an action in state/feature representation thereof

$$\pi$$

$$Q^{\pi}(s,a)$$

$$V^{\pi}(s)$$

usual policies

greedy	ε-greedy	soft-max
choose **best action**	choose **best action with probability 1-ε**	choose **action with probability given by its value**

exploration vs. exploitation
trial and error

game play: try new moves

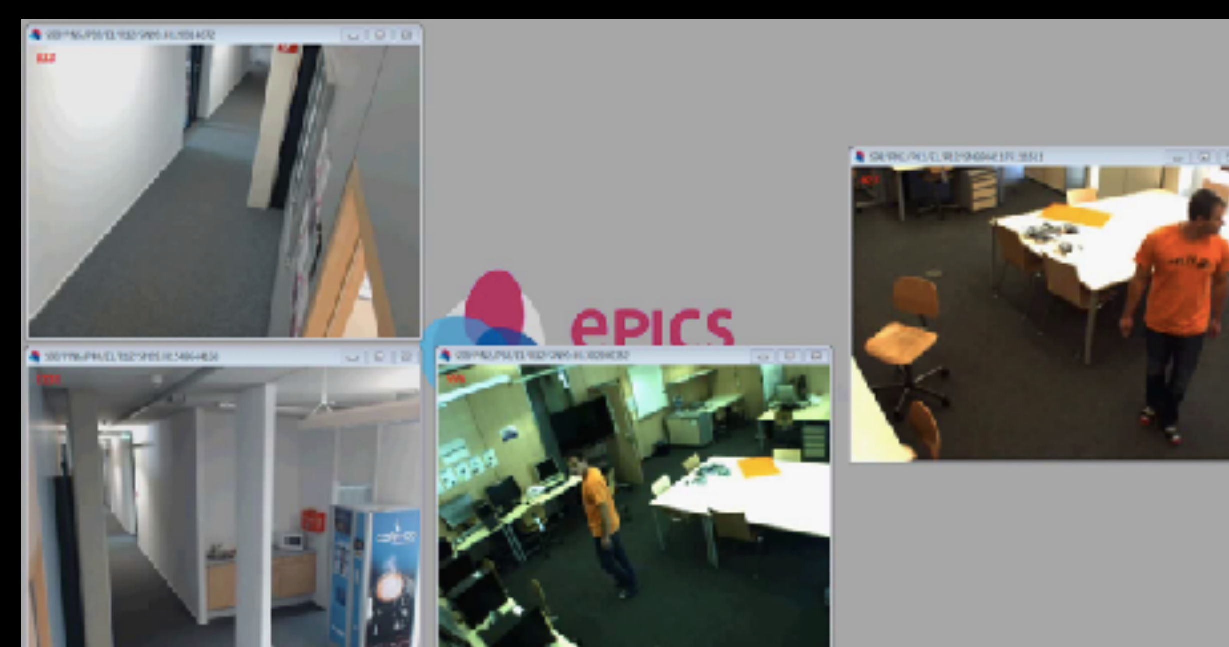
ads: try new ads

a/b testing:
try new website feature

smart camera networks:
try new comm. protocol

Static, dynamic and adaptive heterogeneity in socio-economic distributed smart camera networks, P. R. Lewis, L. Esterle, A. Chandra, B. Rinner, J. Torresen, and X. Yao, ACM Transactions on Autonomous and Adaptive Systems (TAAS), ACM, 2015.

Yamaguchi先生, http://en.wikipedia.org/wiki/File:Las_Vegas_slot_machines.jpg



$$V^*$$

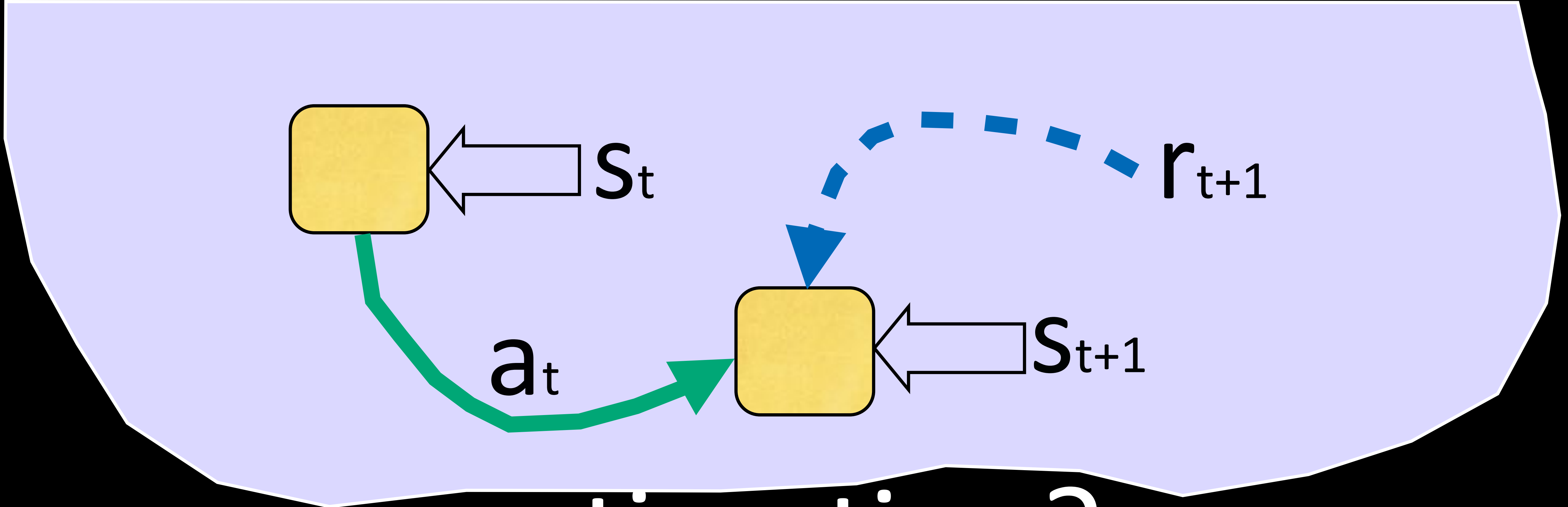
$$V^*(s) = \max_\pi V^\pi(s)$$

$$Q^*$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

$$\pi^*$$

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_a Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$



estimation?

<<use currently visible returns to update values of where you are coming from>>

the current state (or state-action pair) has an estimated value (say zero/random initially),

which can be **used together with r_{t+1} to update value of previous state (or state-action pair)**

i.e.

fraction of (currently visible returns - old value)

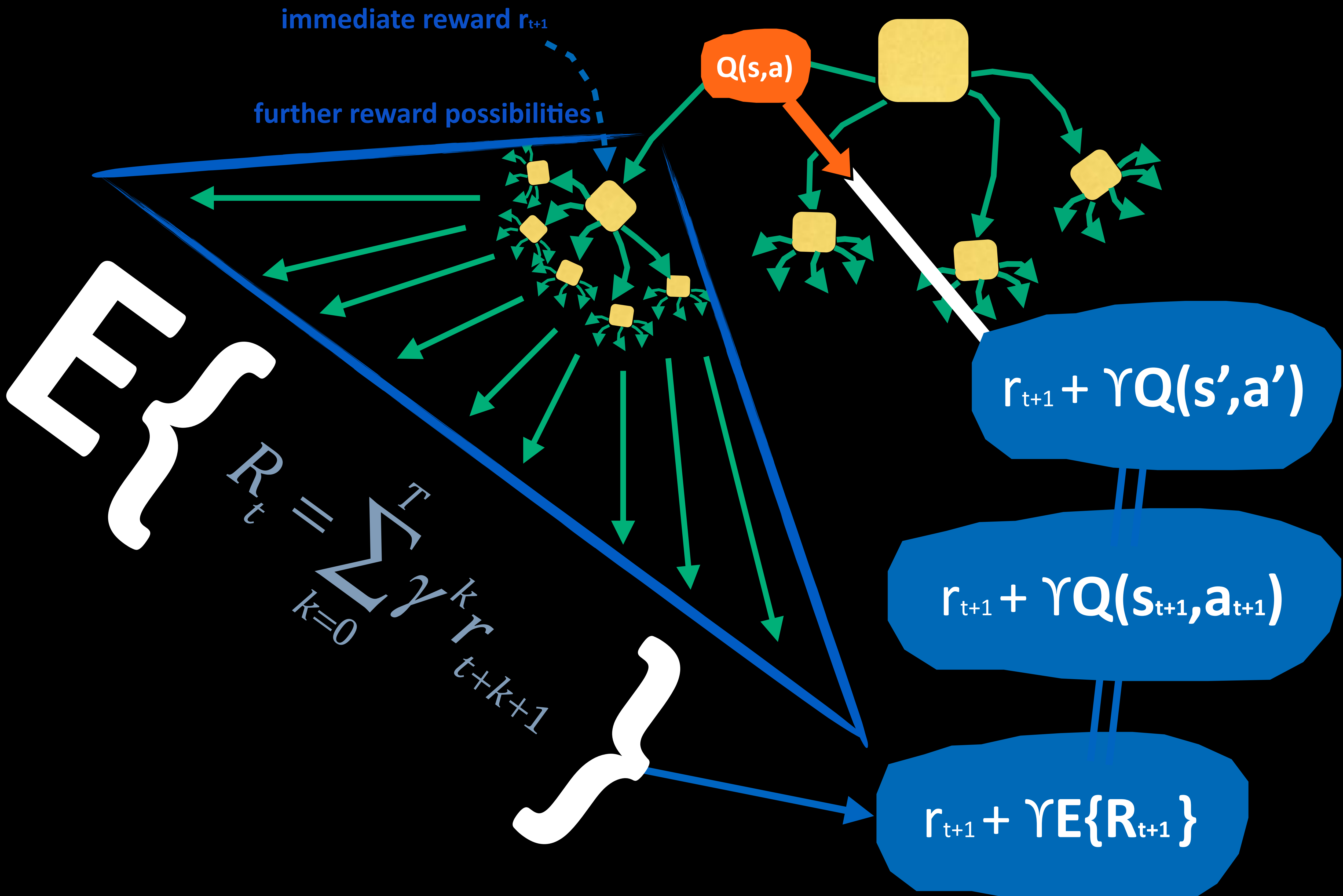
\+

old value

↓

new value

(1-fraction) old value + fraction curr. vis. returns



immediate reward r_{t+1}
further reward possibilities
Q(s,a)
$E\{R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}\}$
$r_{t+1} + \gamma Q(s',a')$
$r_{t+1} + \gamma Q(s_{t+1},a_{t+1})$
$r_{t+1} + \gamma E\{R_{t+1}\}$

under some policy π(a|s)

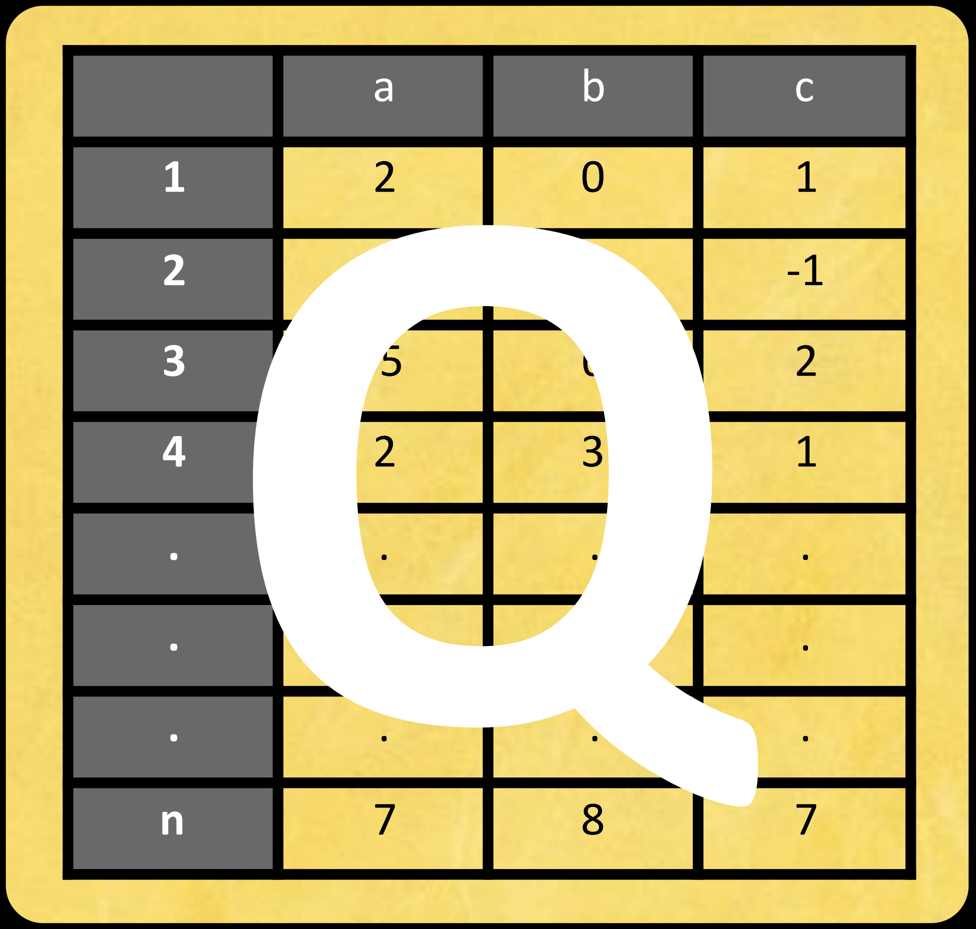
$$V(s) \leftarrow V(s) + \alpha(r_s^a + \gamma V(s') - V(s))$$

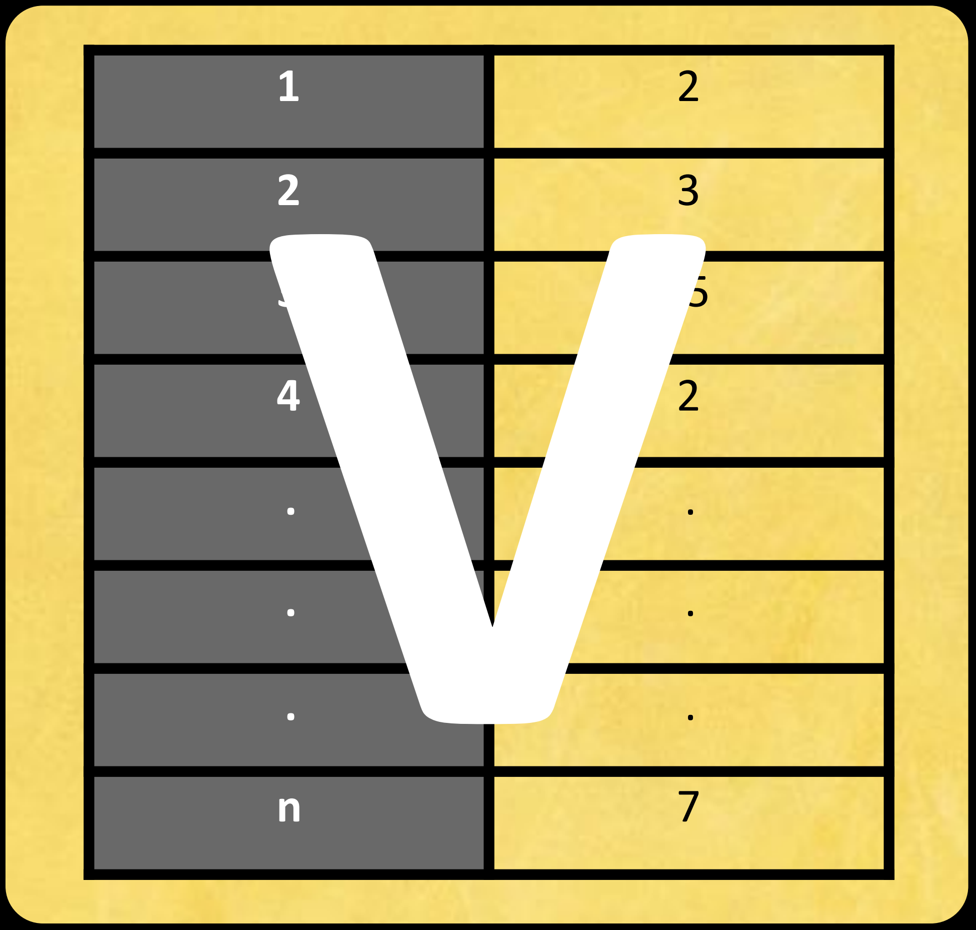
e.g.

under some policy π(a|s)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma Q(s',a') - Q(s,a))$$

e.g. **update**
a lookup table maintaining expected returns





$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma Q(s',a') - Q(s,a))$$

let's play with a version of the above update rule:

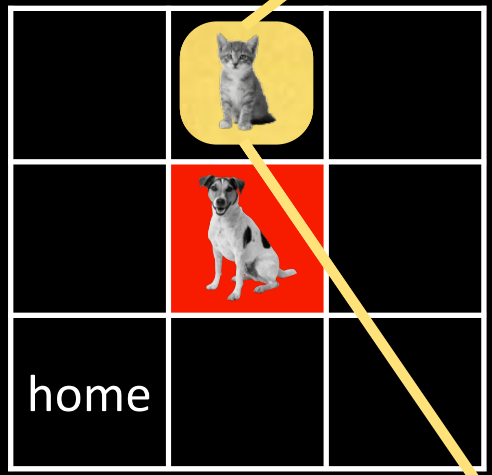
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

indicates a’ to be the action with maximum value in next state s’

let’s play with a version of the above update rule:

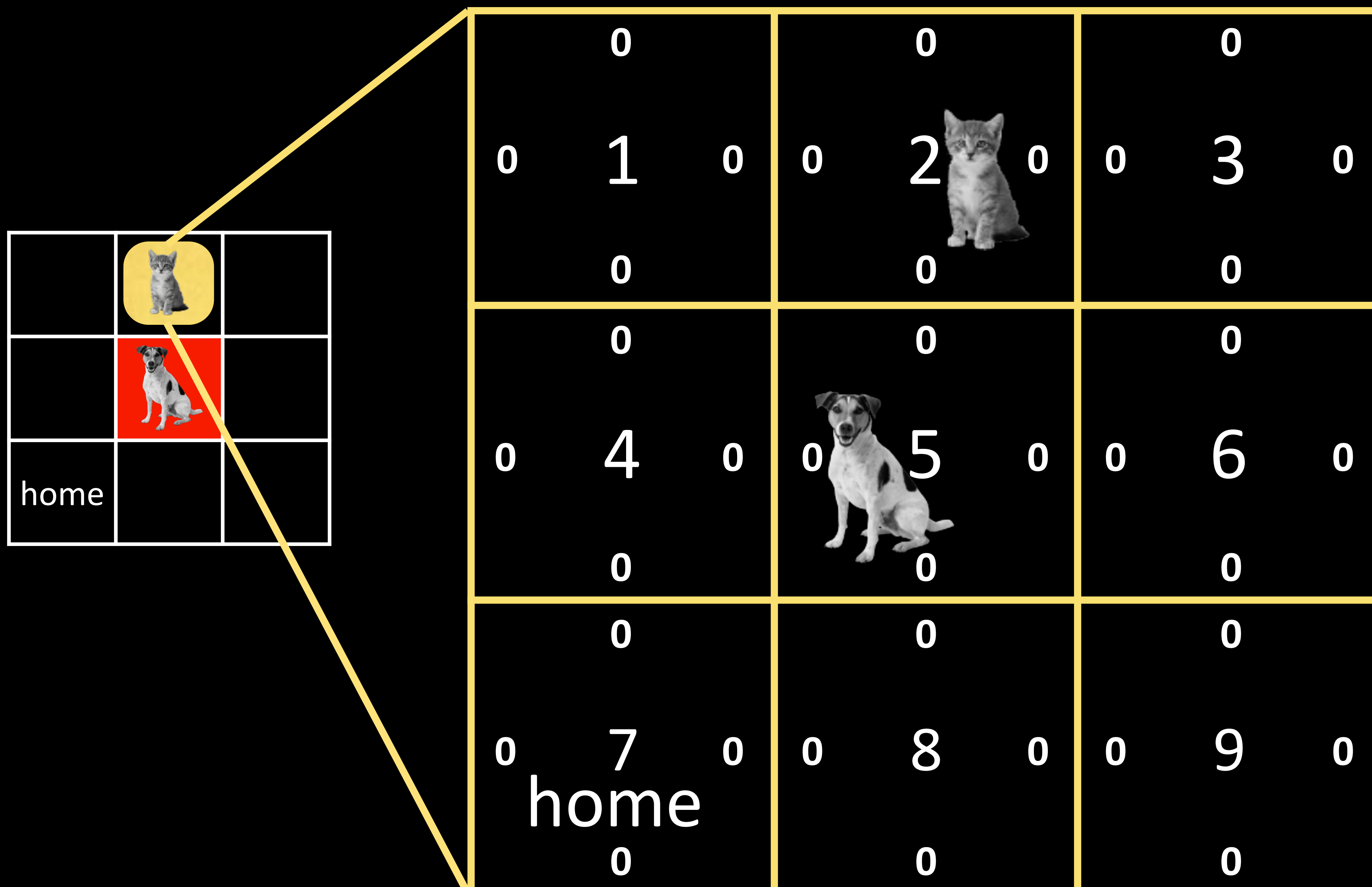
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

our toy problem
lookup table

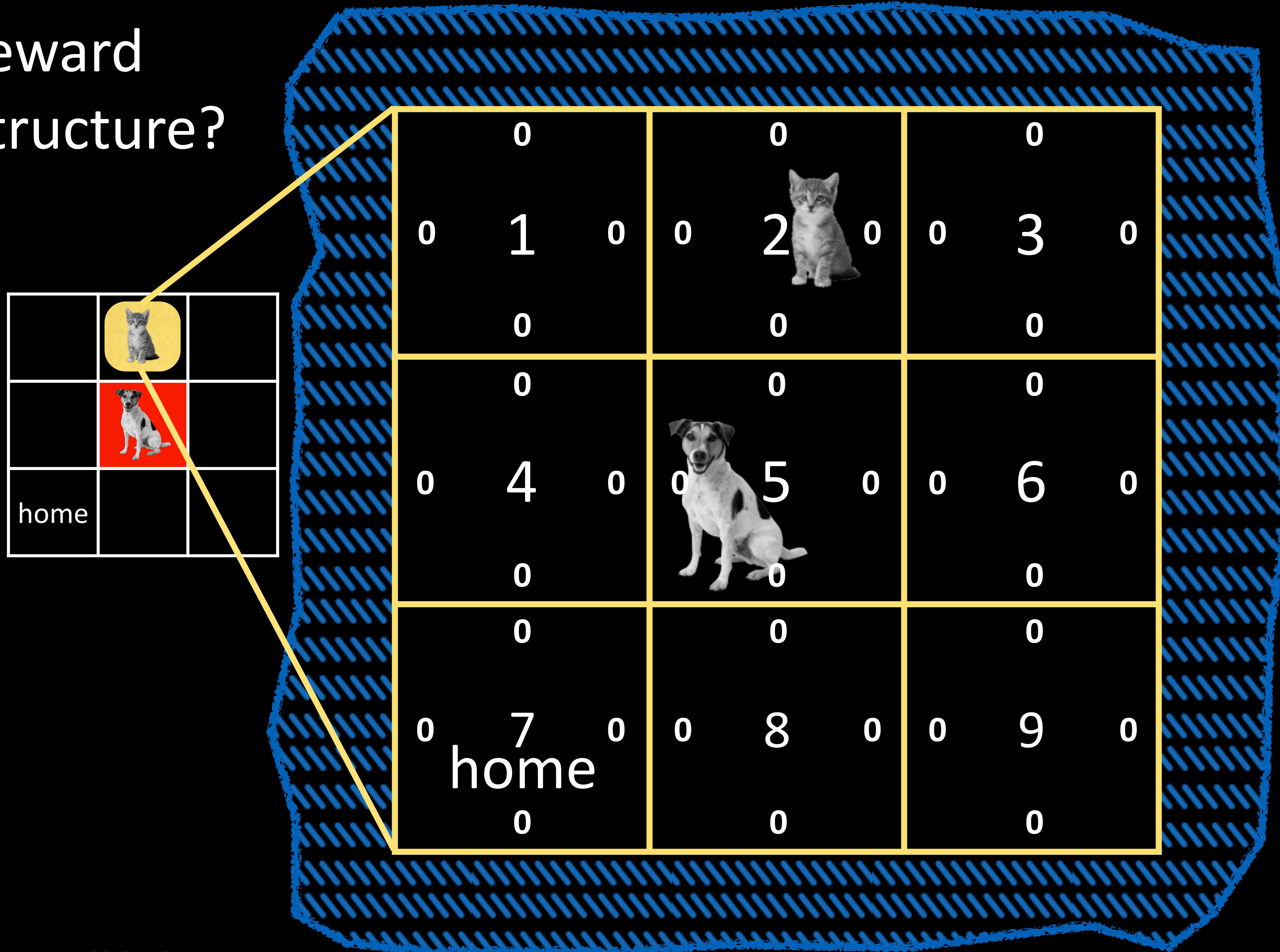


	N	S	E	W
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

our toy problem
lookup table



reward structure?



move...

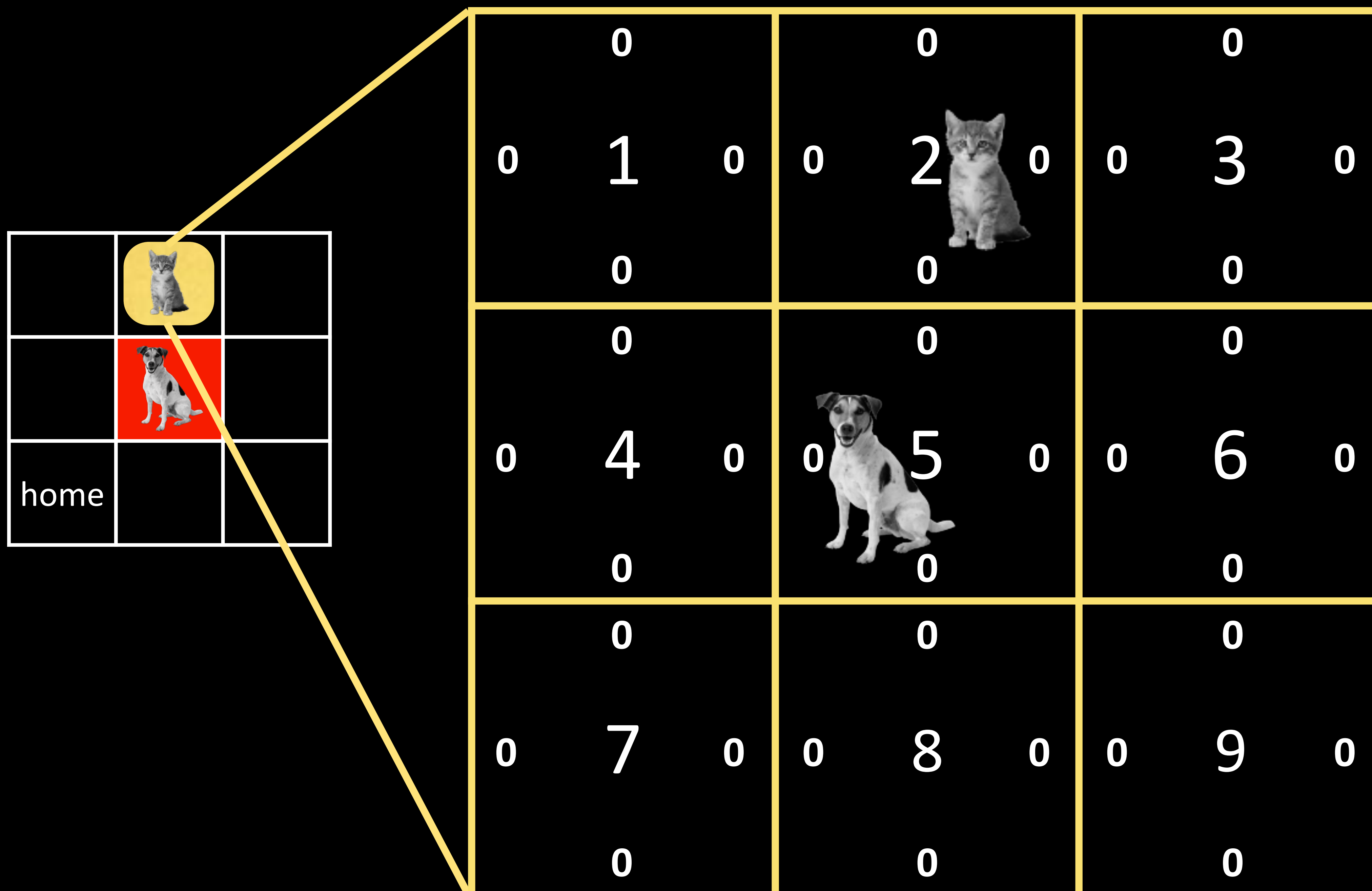
to any cell except 5 and 7: **-1**

out of bounds: **-5**

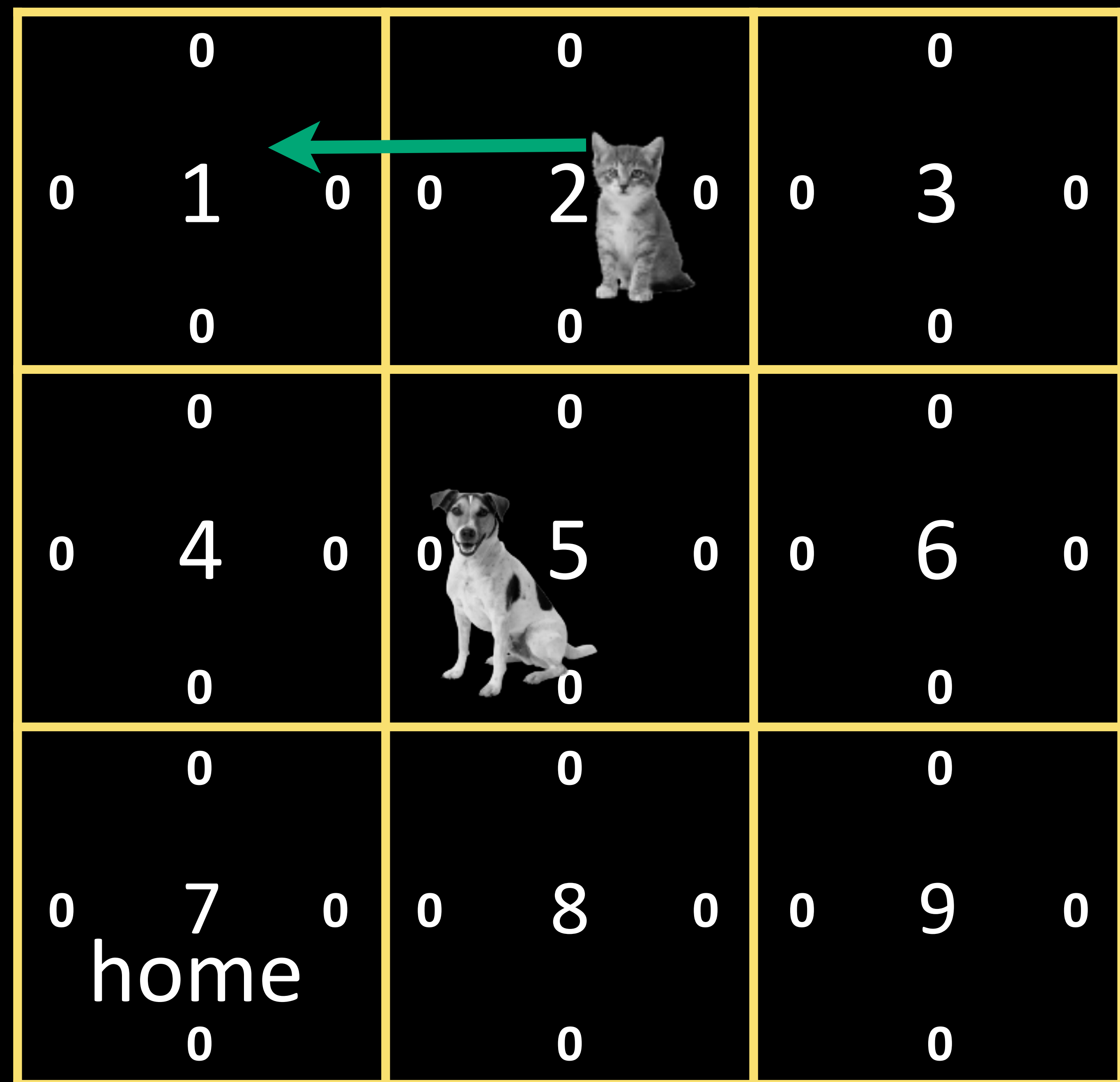
to 5: **-10**

to 7/home: **10**

let’s fix $\alpha = 0.1, \gamma = 0.5$



$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



$\alpha = 0.1$

$\gamma = 0.5$

say ε–greedy policy…

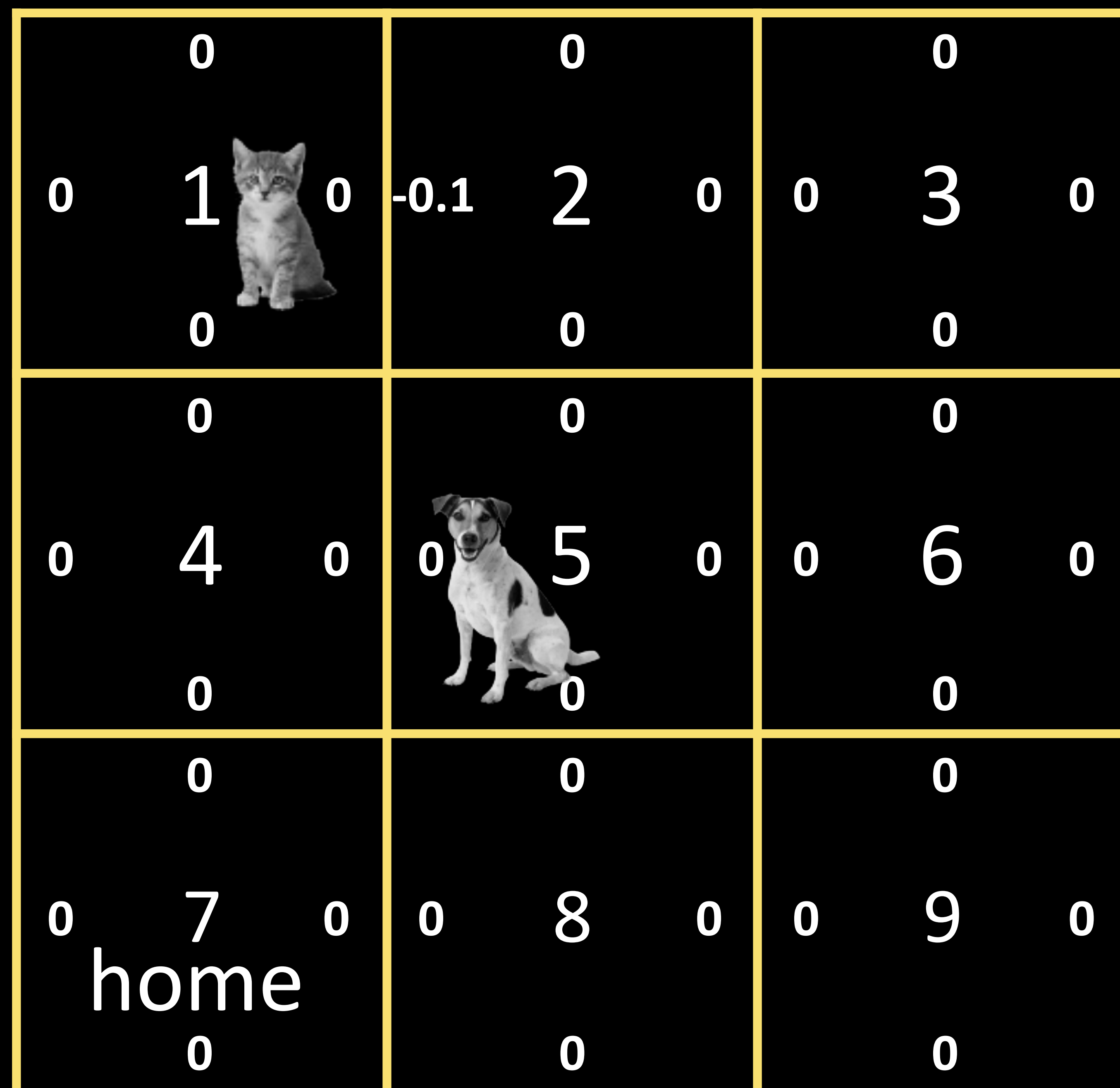
episode 1 begins...

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$



α = 0.1

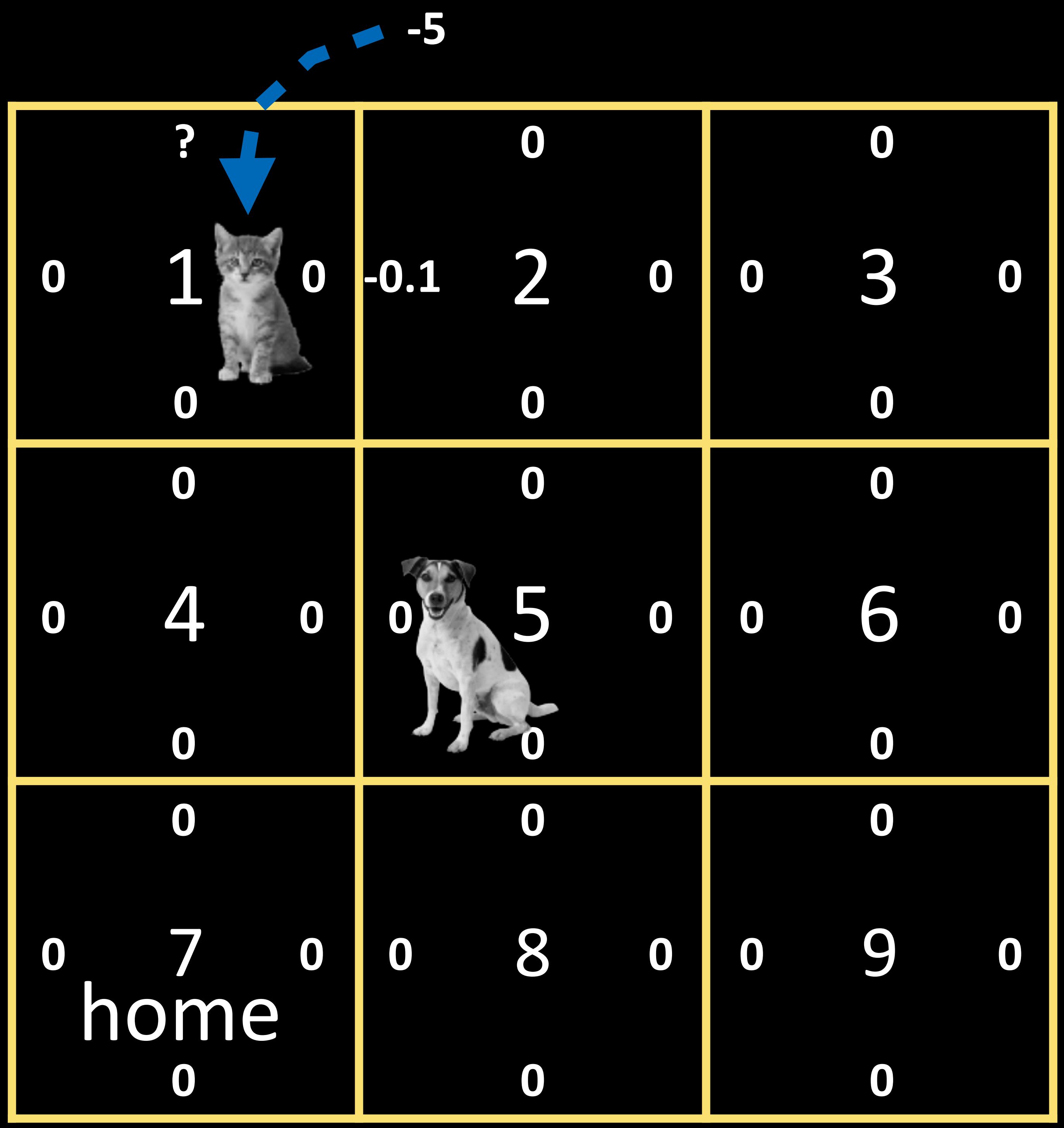
γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

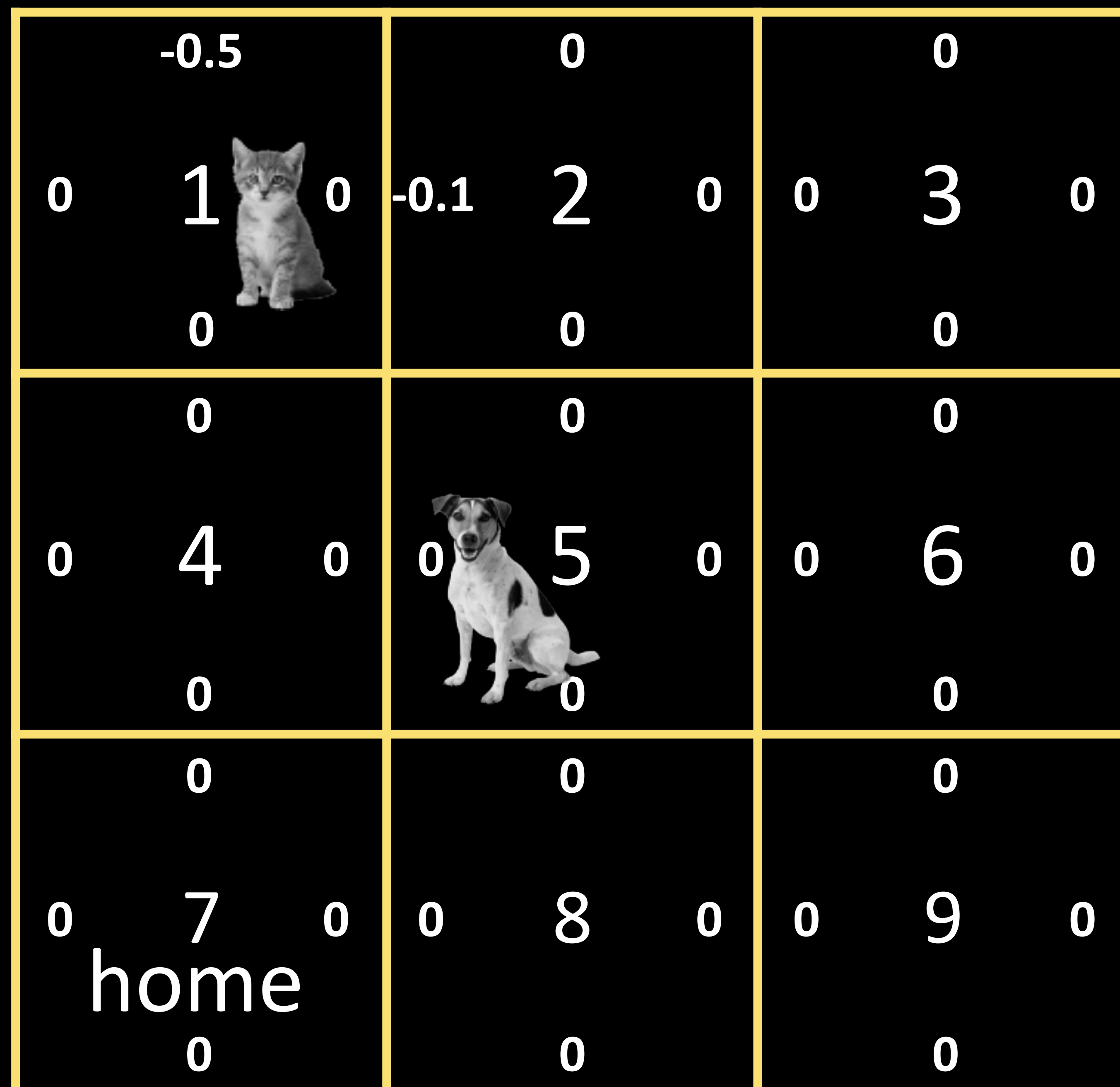
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$



$\alpha = 0.1$

$\gamma = 0.5$

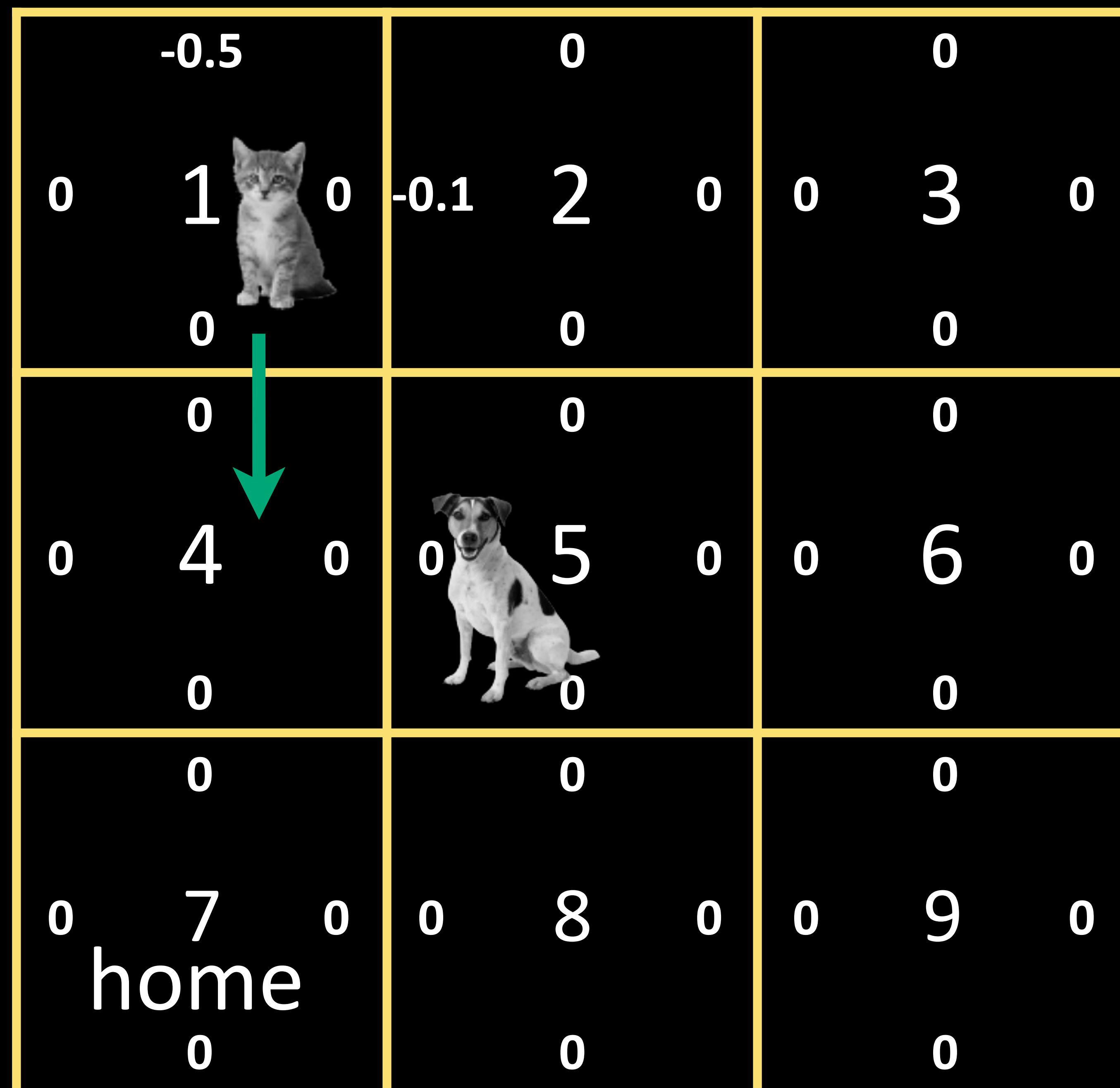
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



α = 0.1

γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



α = 0.1

γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

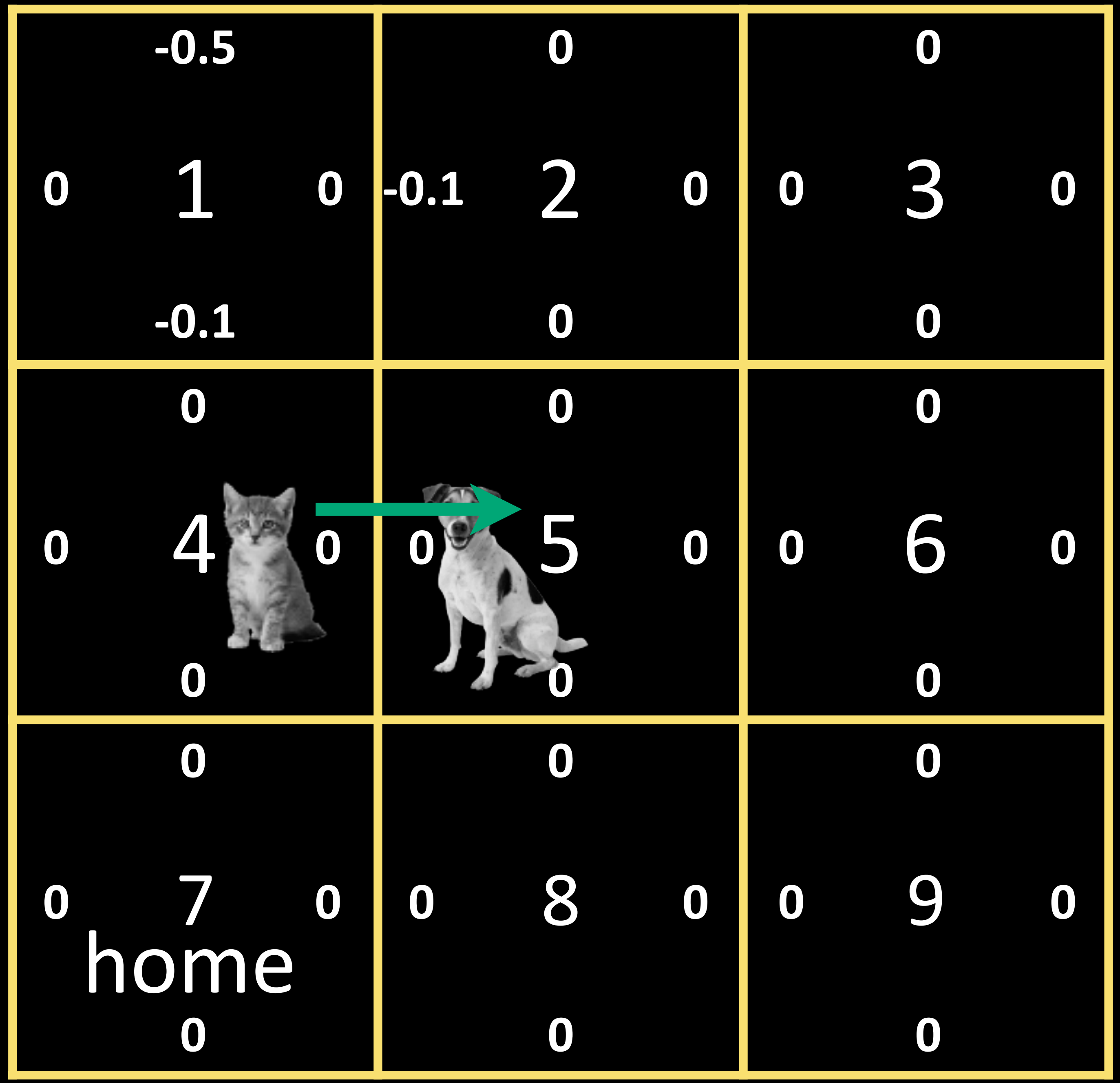
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

-0.5 0 1 0 -0.1	0 -0.1 2 0 0	0 0 3 0 0
0 0 4 0  0	0 0 5 0  0	0 0 6 0 0
0 0 7 0 home 0	0 0 8 0 0	0 0 9 0 0

$\alpha = 0.1$

$\gamma = 0.5$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



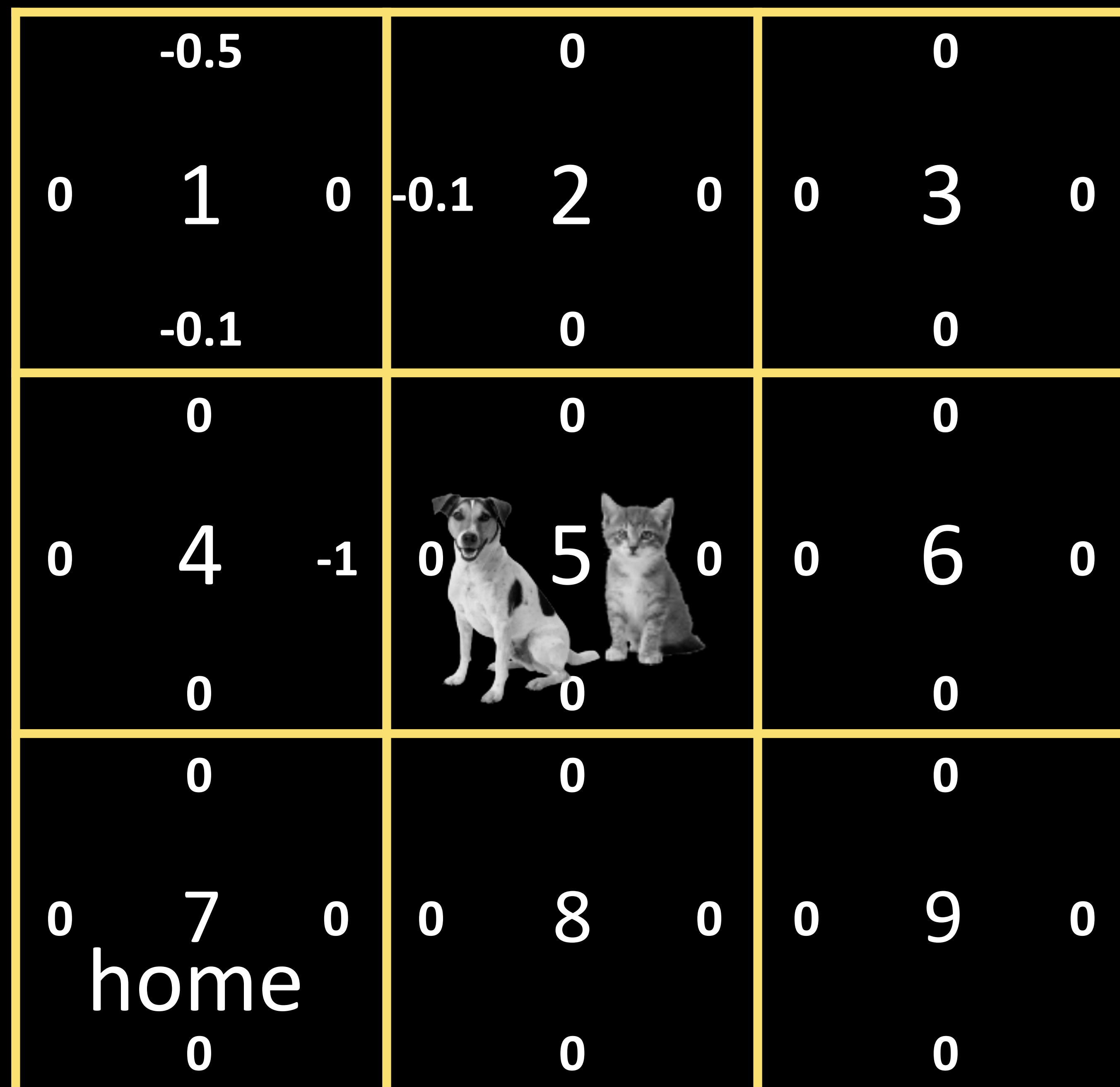
α = 0.1
γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

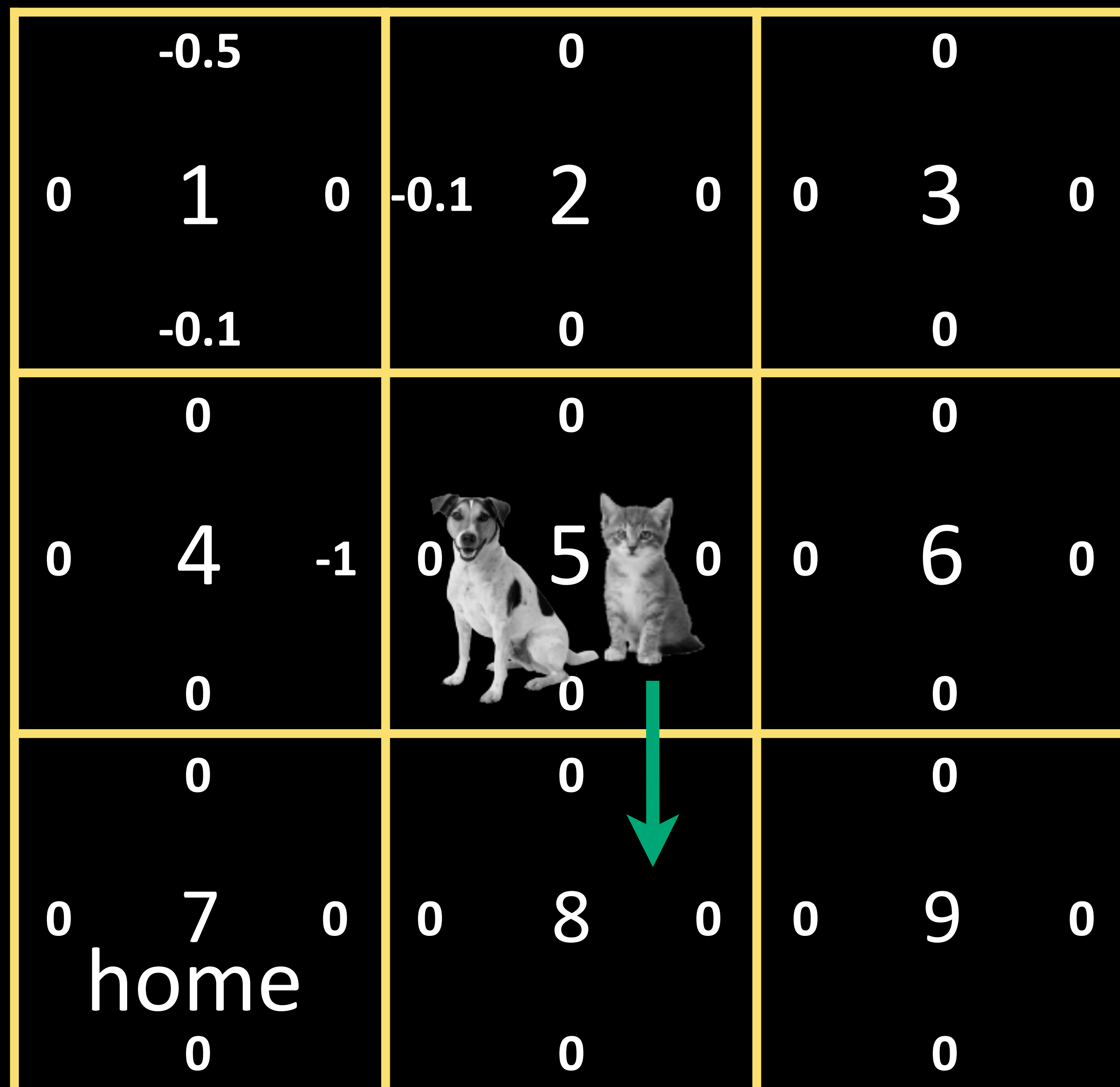
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



α = 0.1

γ = 0.5

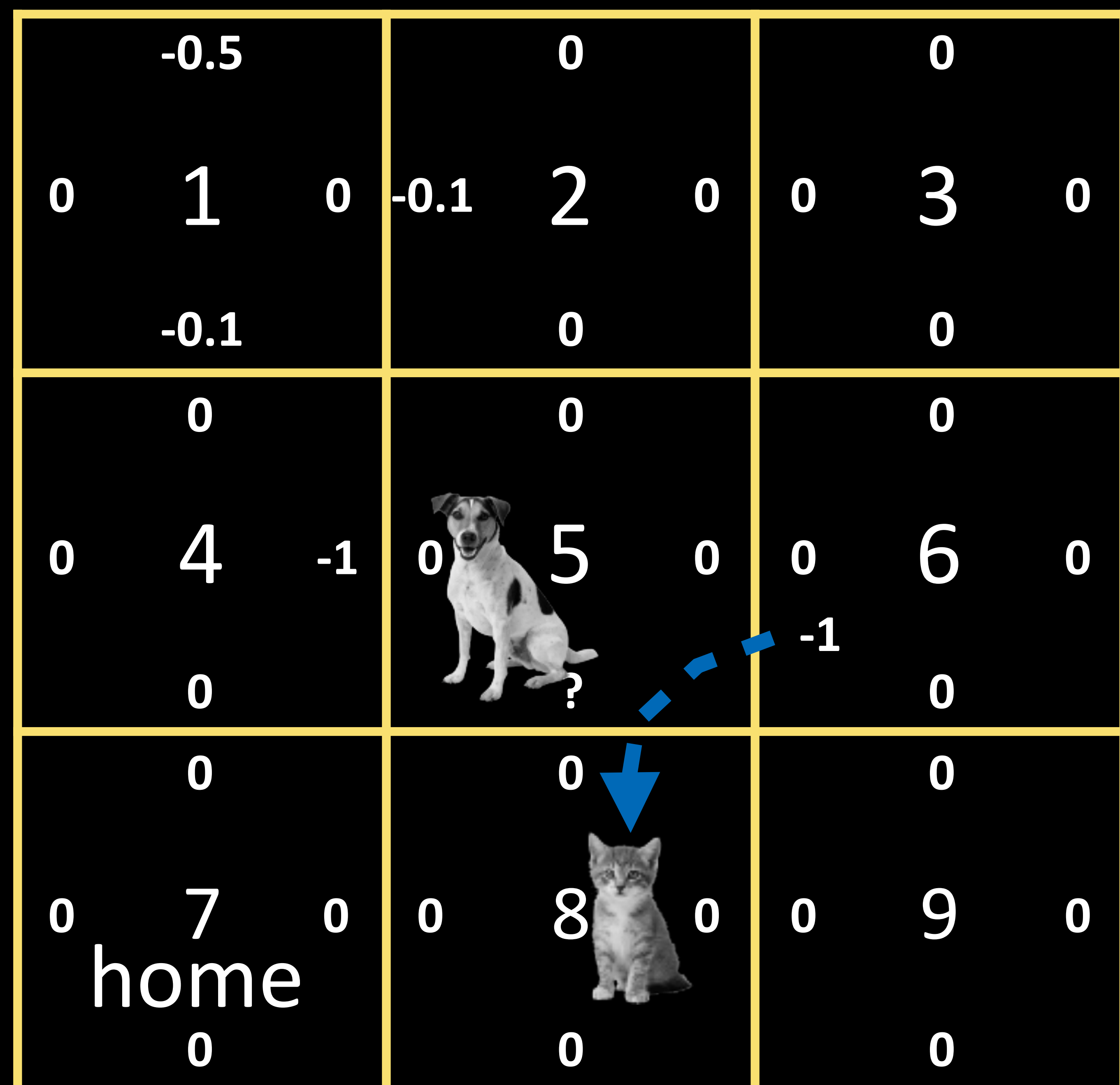
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



$\alpha = 0.1$

$\gamma = 0.5$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



α = 0.1

γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

$\alpha = 0.1$

$\gamma = 0.5$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

α = 0.1

γ = 0.5

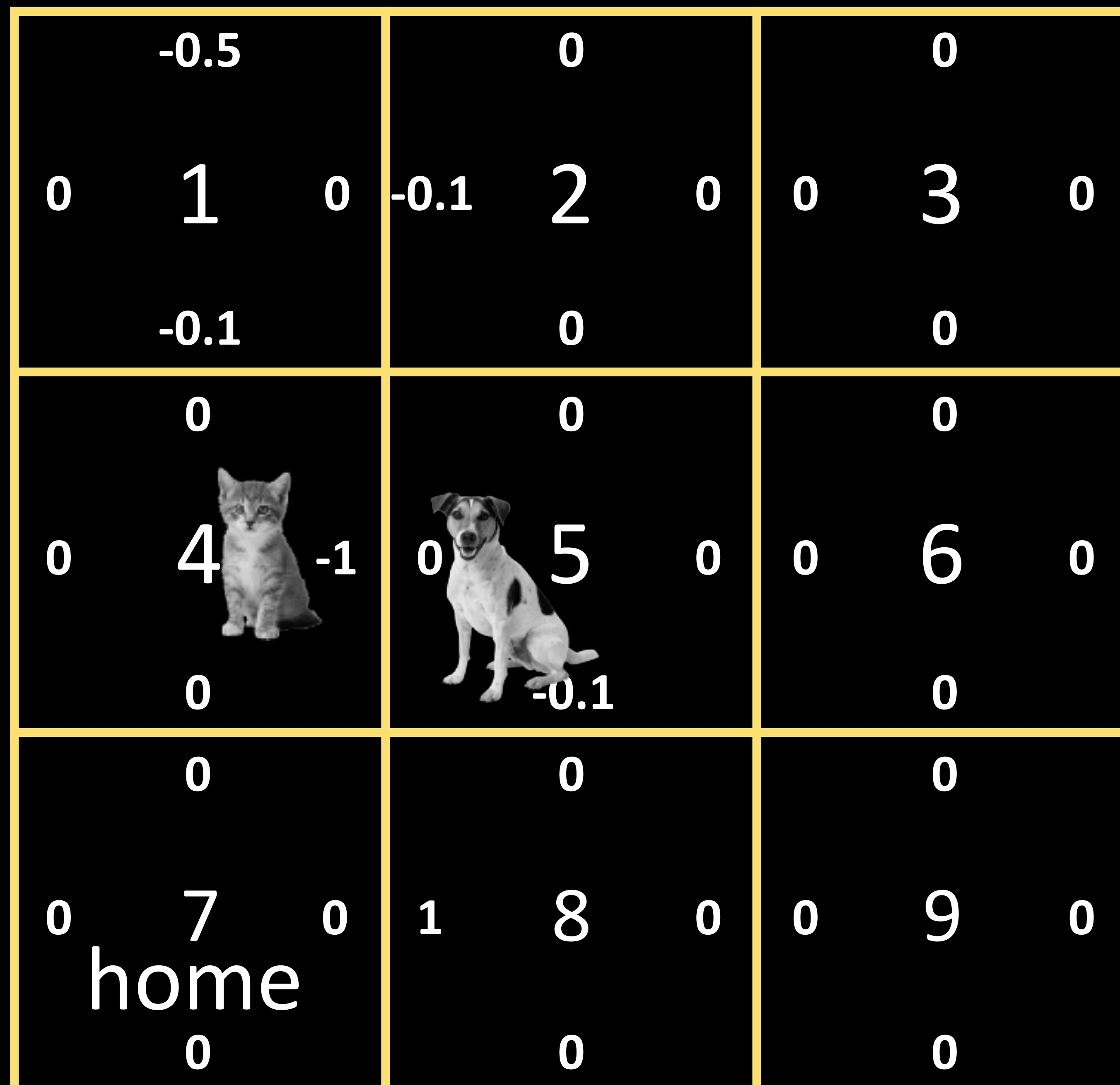
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_s^a + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

$\alpha = 0.1$

$\gamma = 0.5$

episode 1 ends.

let's work out the next episode, starting at state 4

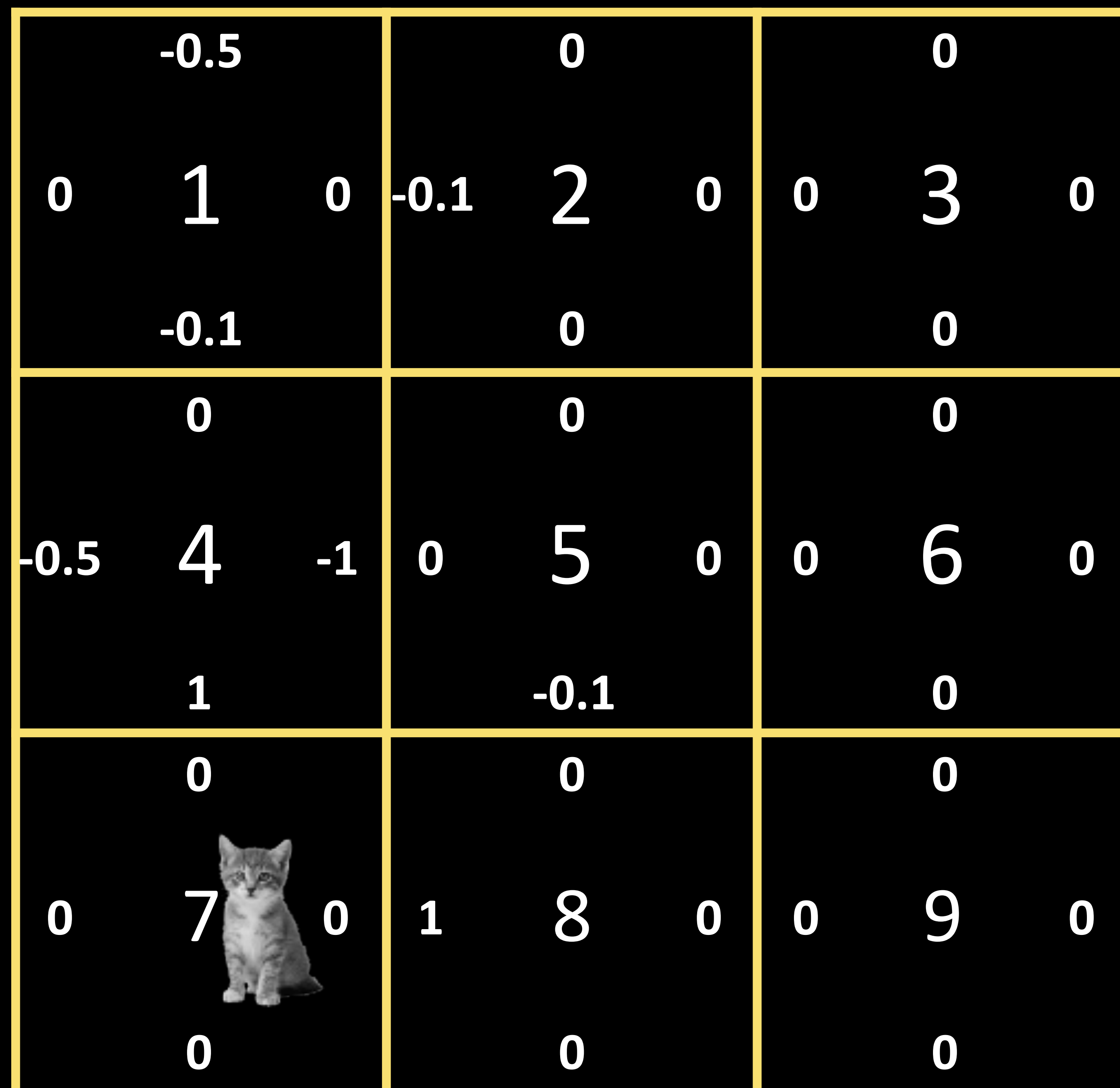


$\alpha = 0.1$

$\gamma = 0.5$

go WEST and then SOUTH

how does the table change?

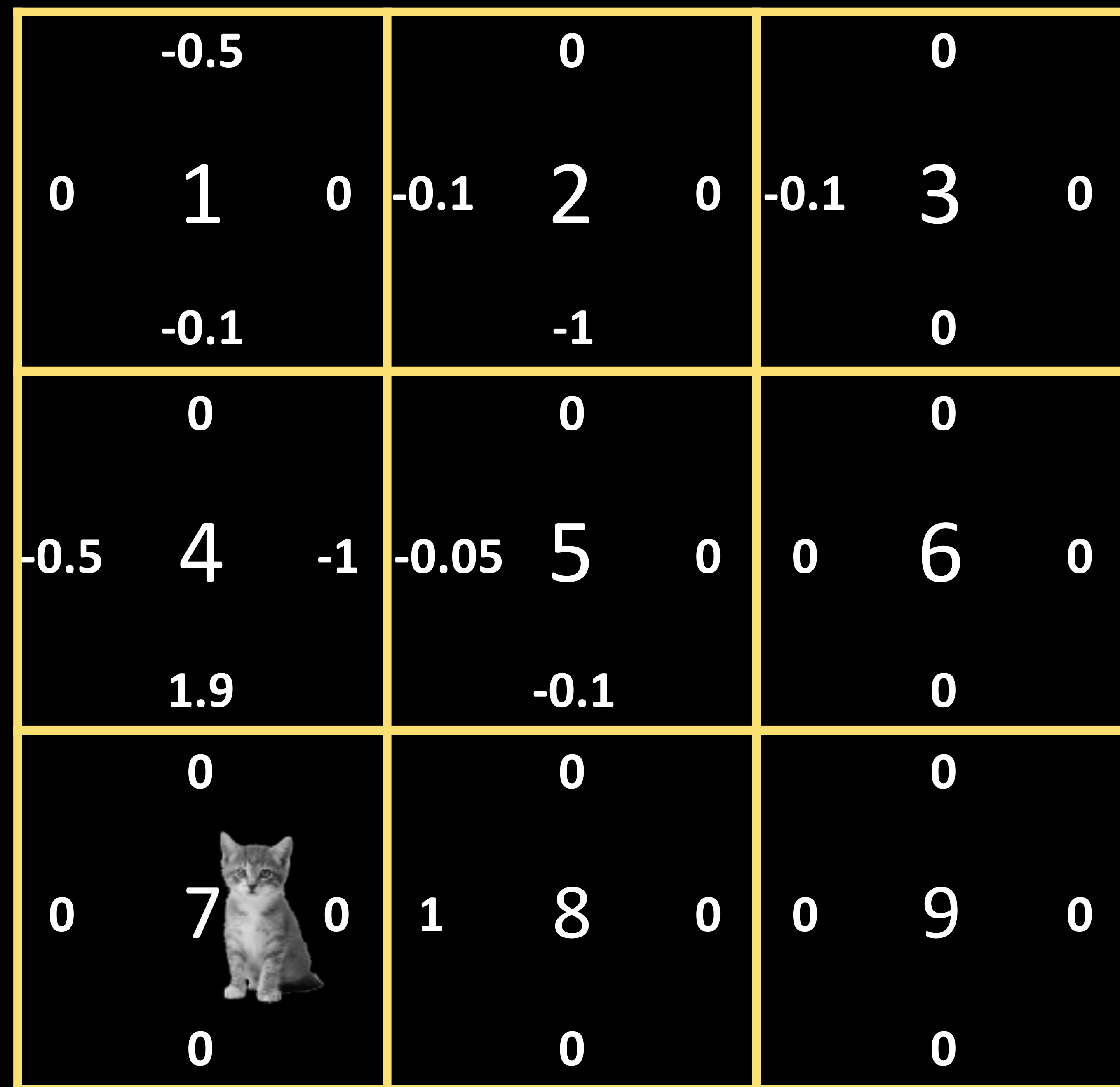


$\alpha = 0.1$

$\gamma = 0.5$

and the next episode, starting at state 3

go WEST -> SOUTH -> WEST -> SOUTH



$\alpha = 0.1$
$\gamma = 0.5$

over time, values will converge to optimal!

what we just saw was
some episodes of

Q-learning

values update towards value of **optimal** **policy**:
target comes from value of
assumed next best action

off-policy learning

SARSA-learning?

values update towards value of **current policy**:

target comes from value of

the actual next action

on-policy learning

data **not generated** by target policy

Q

ε: 0.1
γ: 1.0

data **generated** by target policy

SARSA

Example credit **Travis DeWolf**: https://studywolf.wordpress.com/ and https://git.io/vFBvv

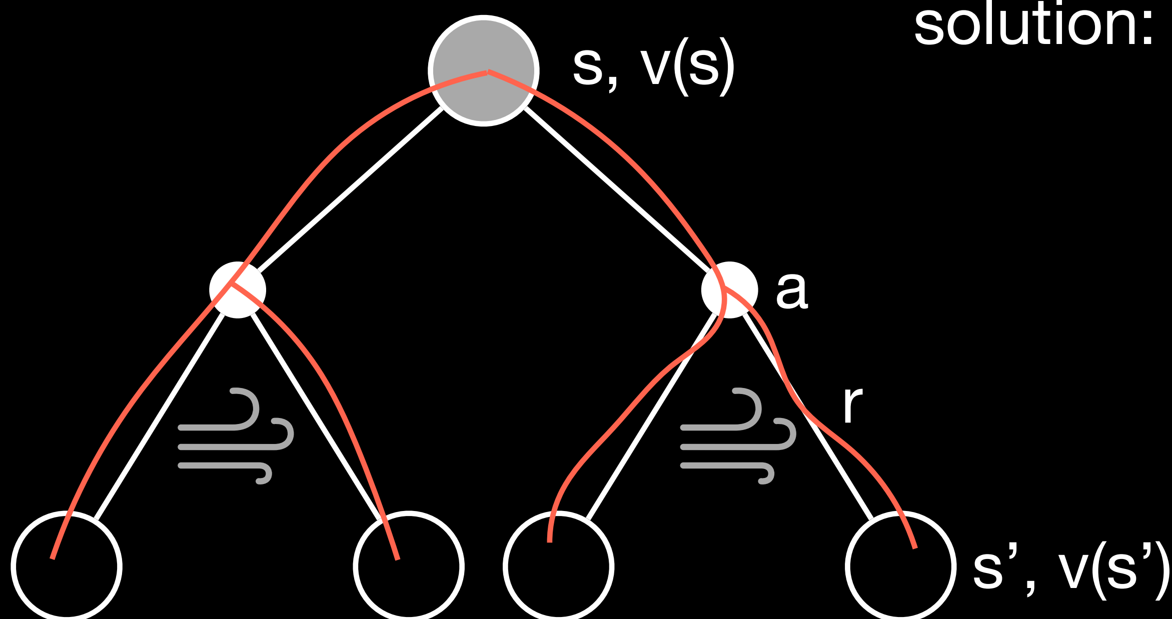
Problem Decomposition

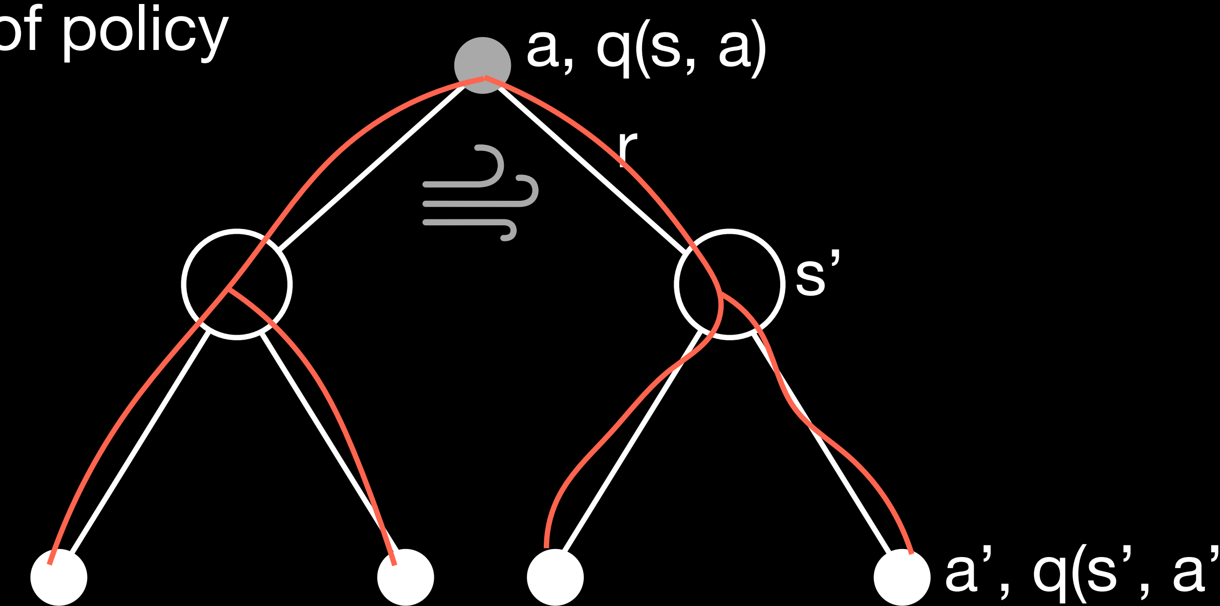
nested sub-problems

solution to sub-problem
informs
solution to whole problem

Bellman Expectation Backup

system of linear equations
solution: value of policy





Value of = P(path) * Value(path)

Value of = P(path) * Value(path)

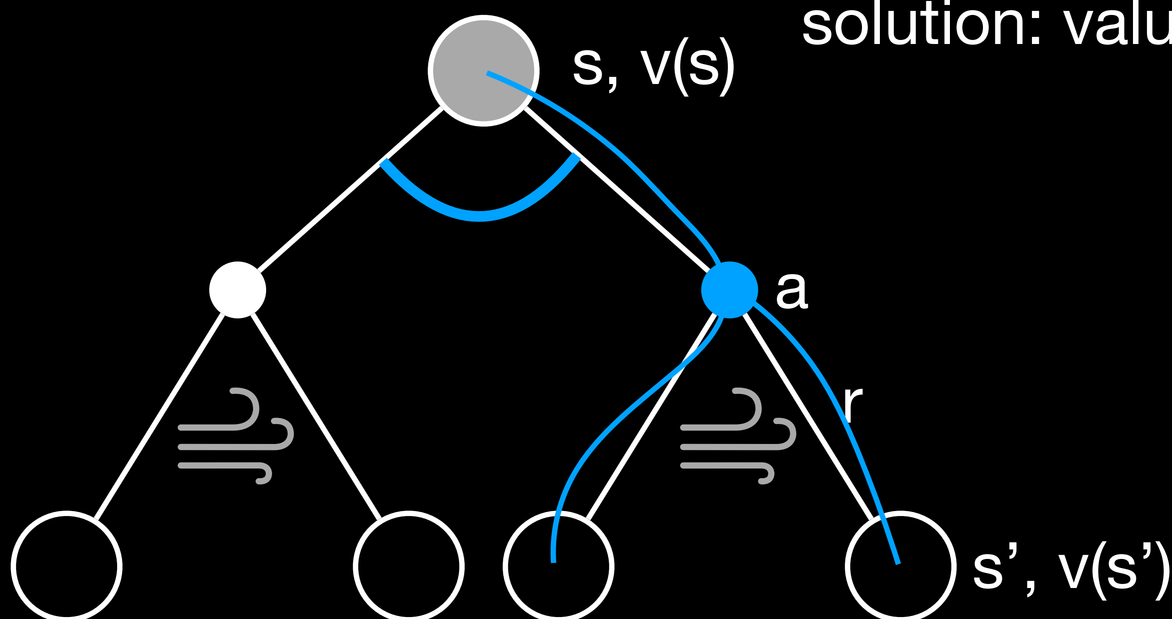
$$v_\pi(s) = \sum_a \pi(a|s)\left(r_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')\right)$$

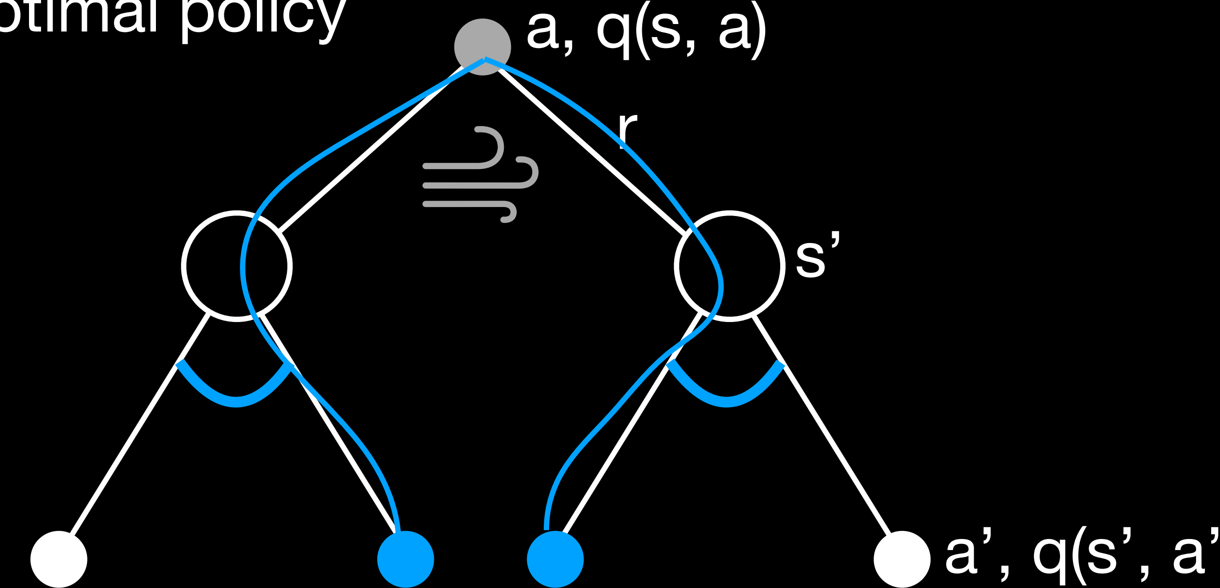
$$q_\pi(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$$

Bellman expectation equations
under a **given policy**

Bellman Optimality Backup

system of non-linear equations
solution: value of optimal policy





Value of = P(path) * Value(path)

Value of = P(path) * Value(path)

$$v_*(s) = \max_a \left(r_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s') \right)$$

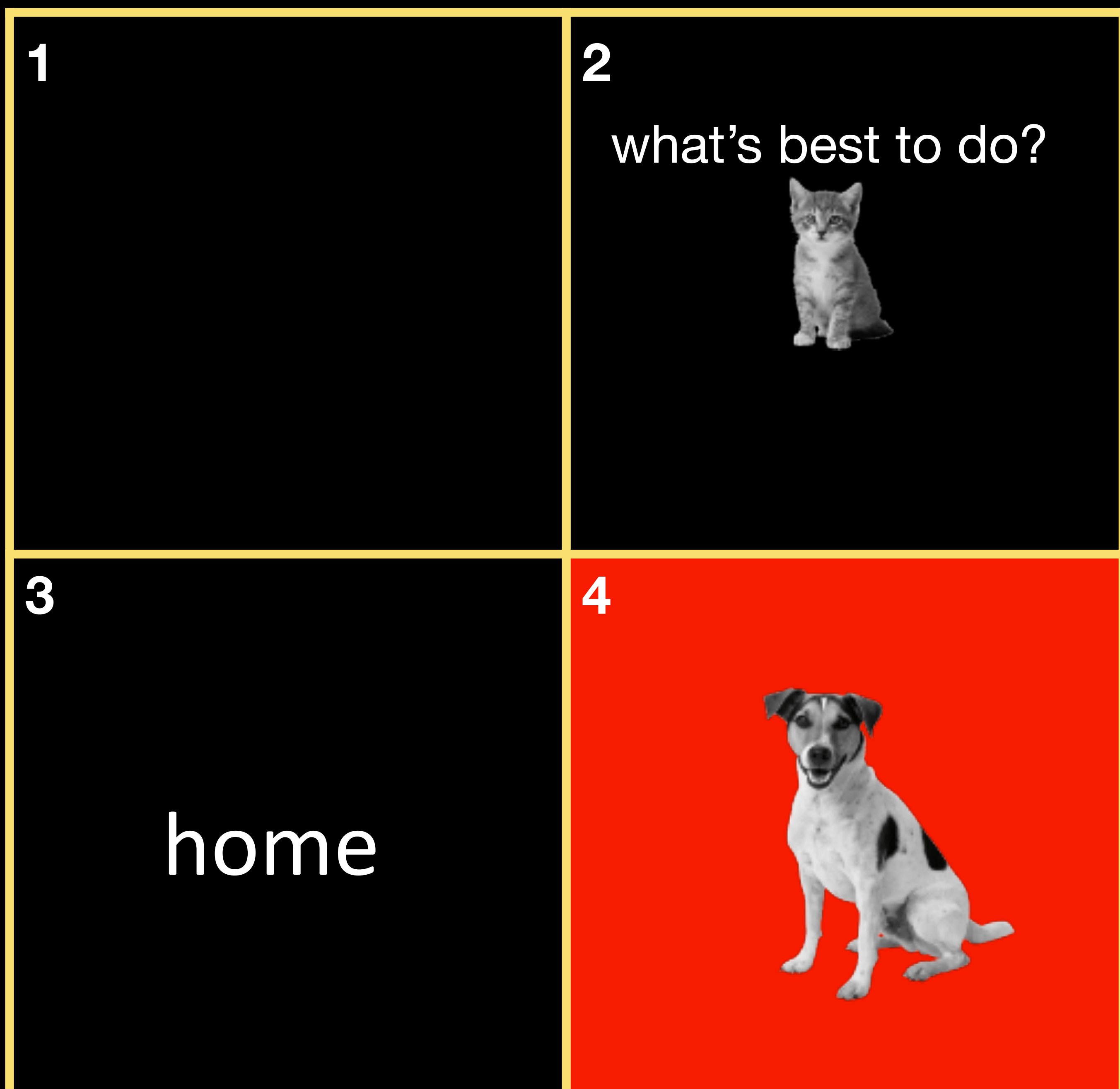
$$q_*(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q_*(s',a')$$

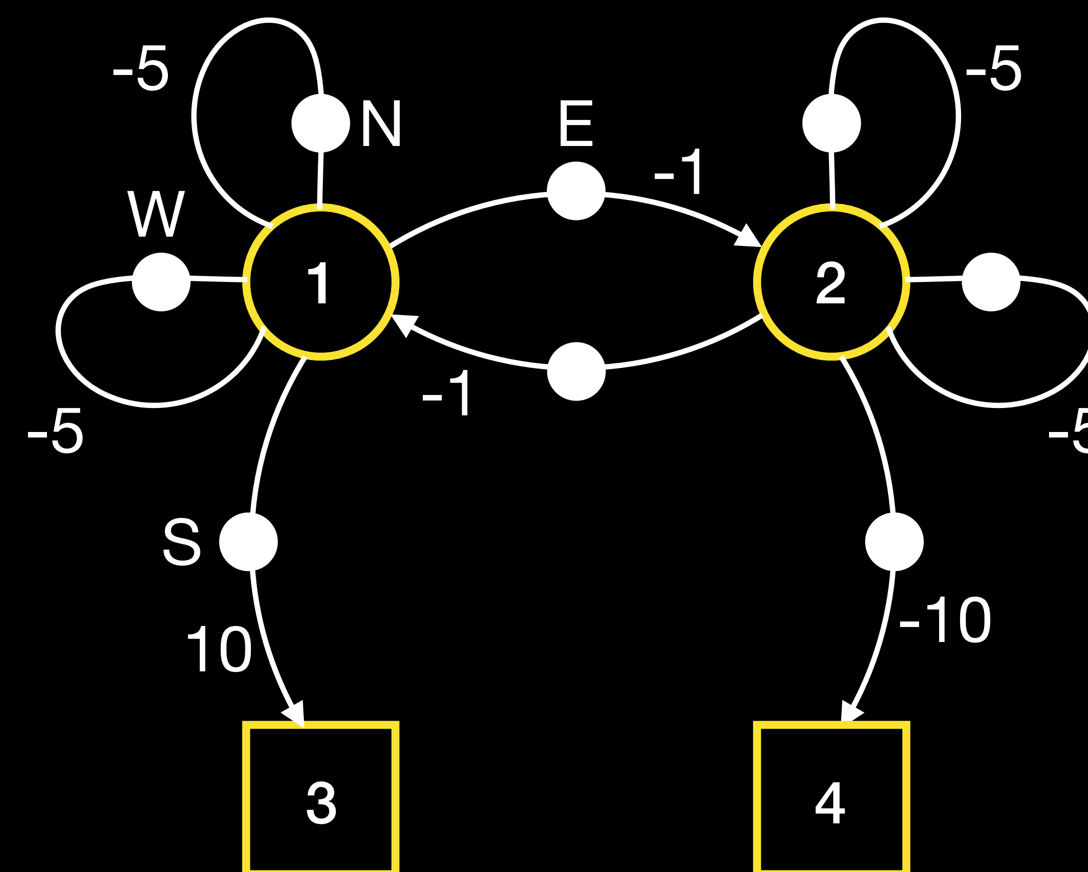
Bellman optimality equations
under **optimal policy**

Value Based

Dynamic Programming

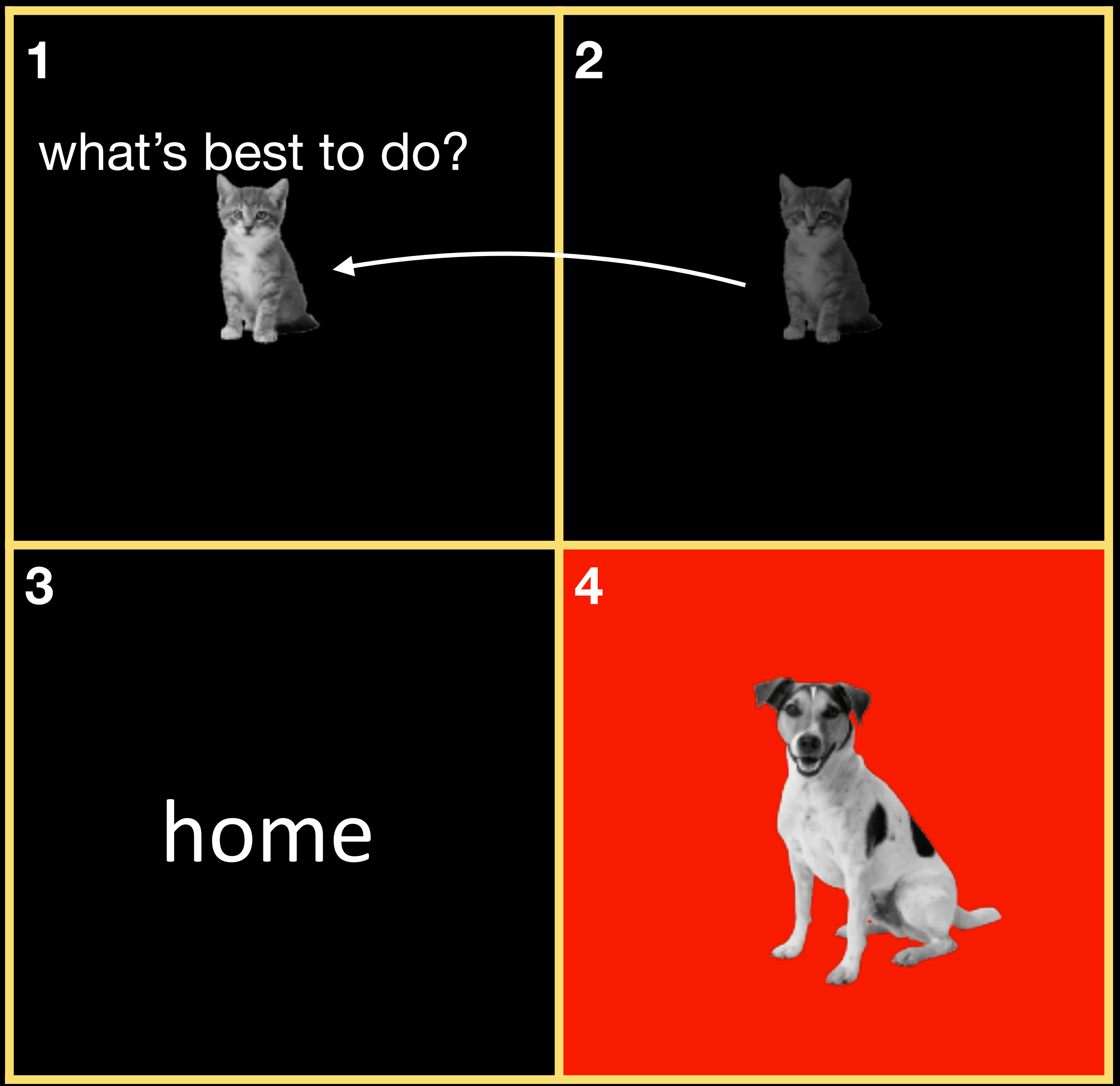
...using Bellman equations as iterative updates

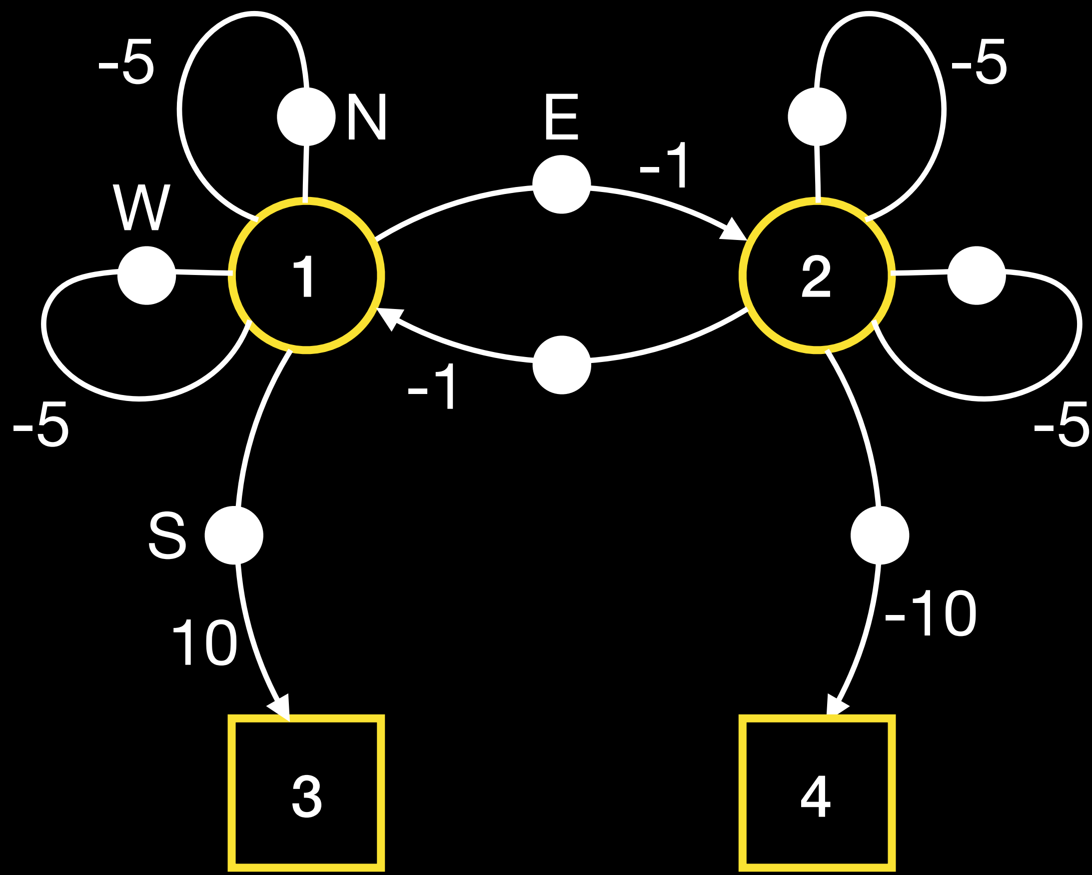




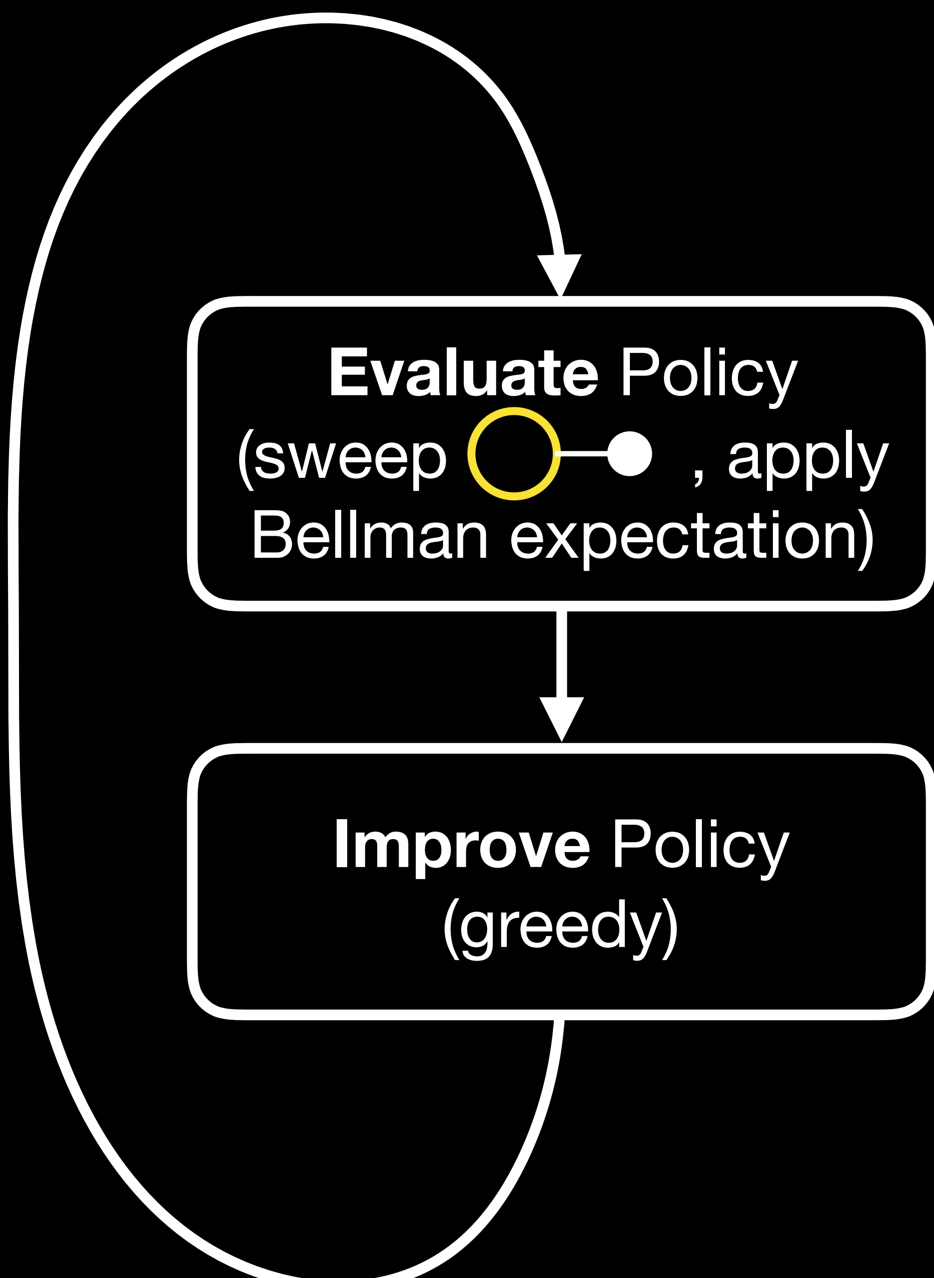
Dynamic Programming

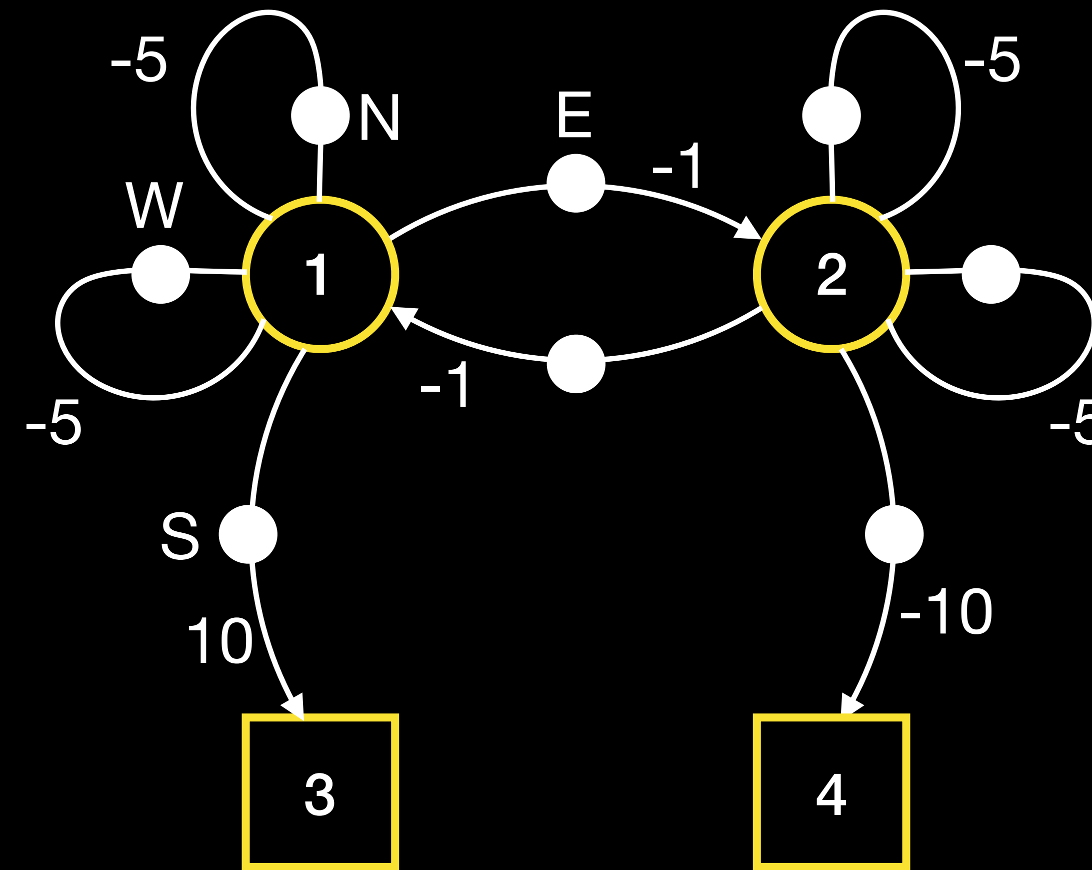
...using Bellman equations as iterative updates

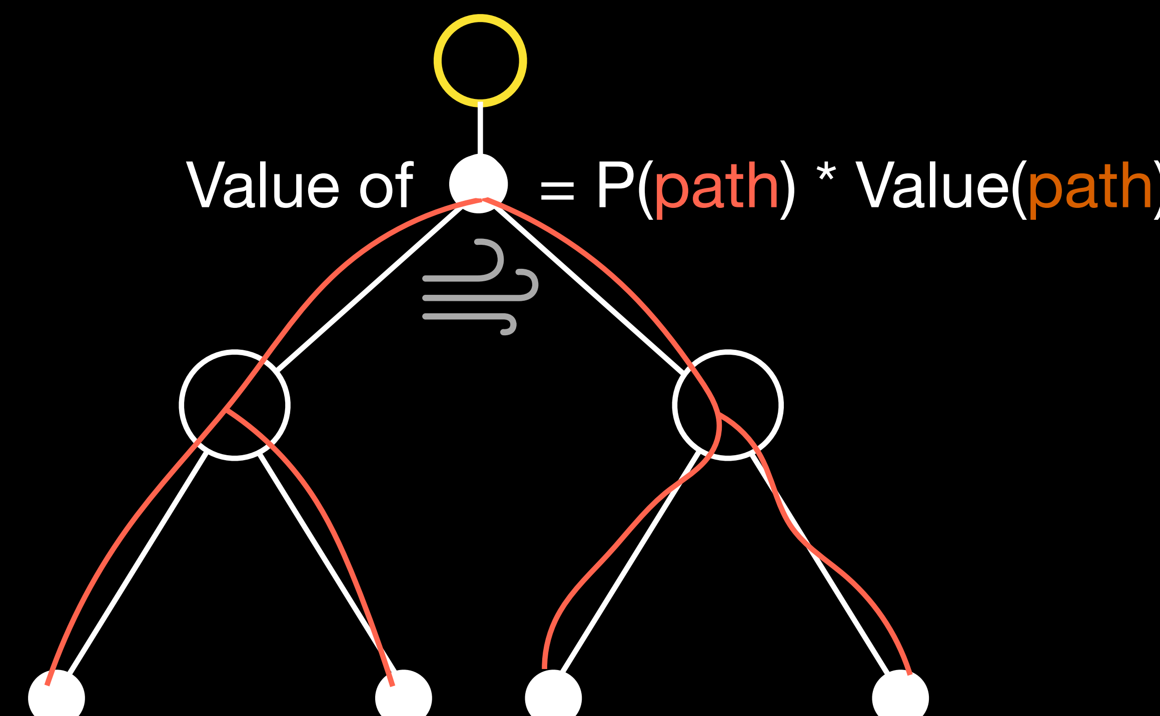




Policy Iteration







$$q_\pi(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$$

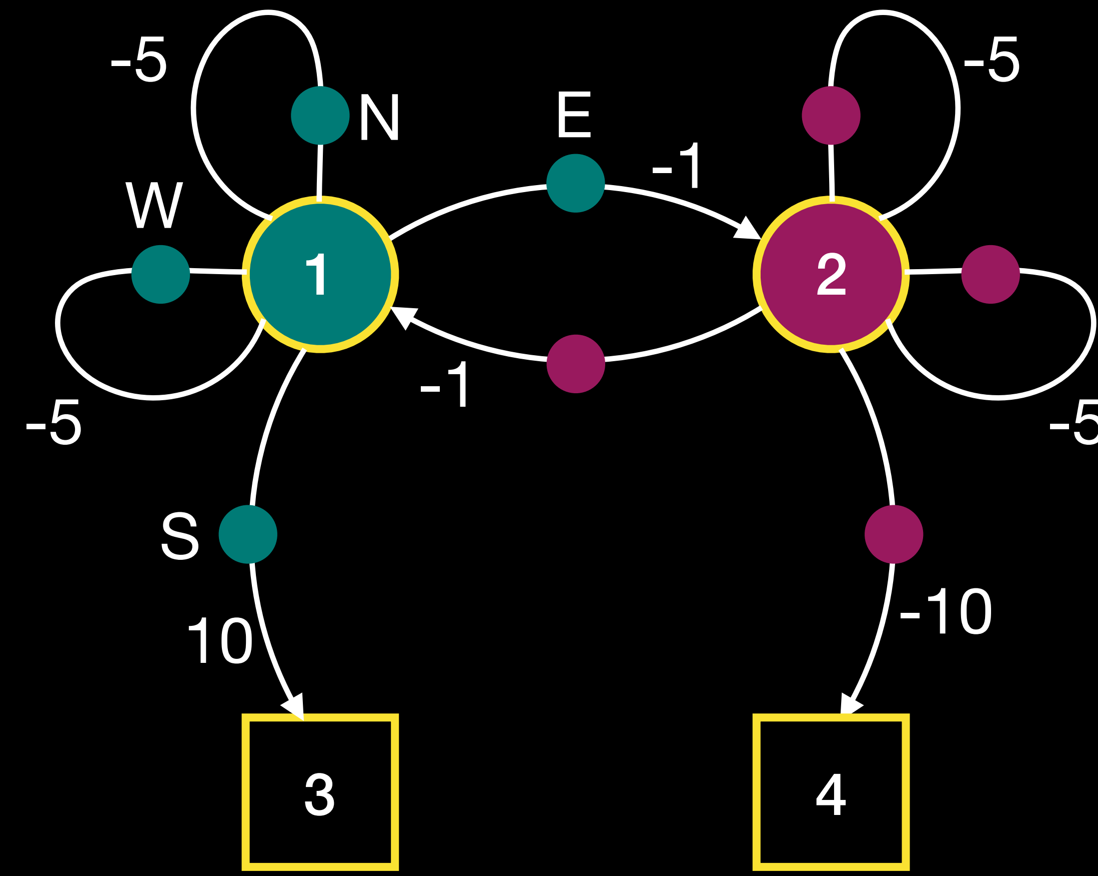
N: -5 + 0.9*0 N: -5 + 0.9*0
E: -1 + 0.9*0 E: -5 + 0.9*0
S: 10 + 0.9*0 S: -10 + 0.9*0
W: -5 + 0.9*0 W: -1 + 0.9*0

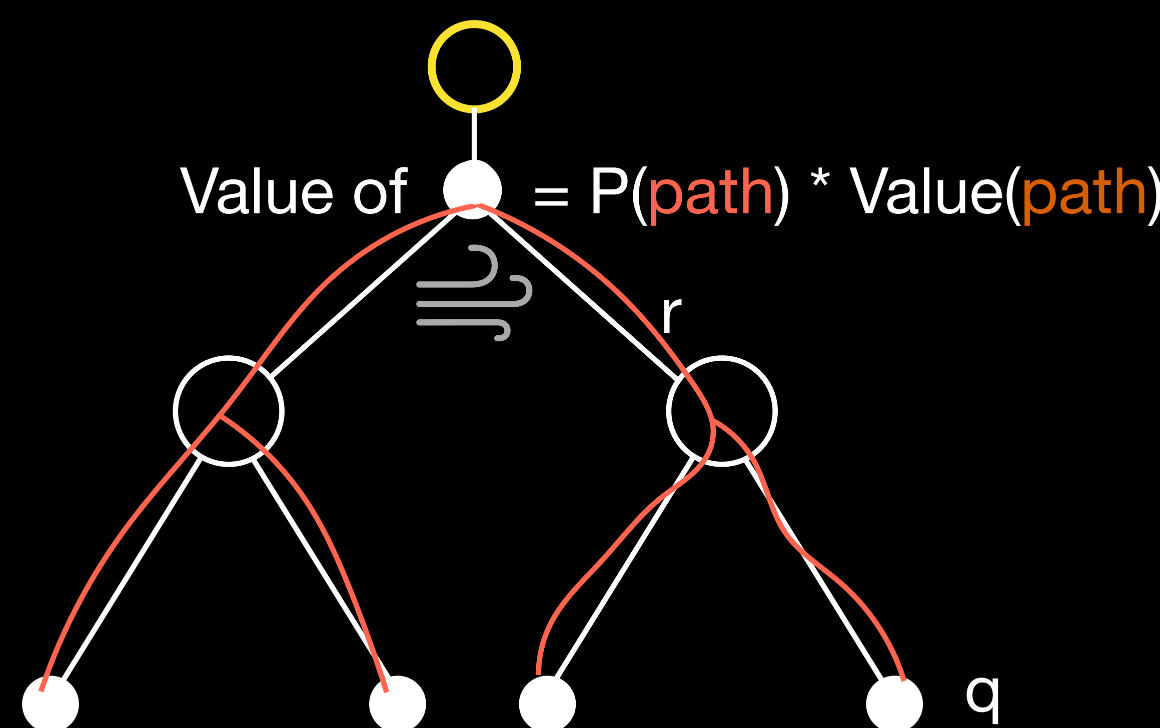
iteratively apply Bellman expectation equations in inner loop until values do not change much

use greedy policy, given new values

π(S|1): 1.0 (greedy)

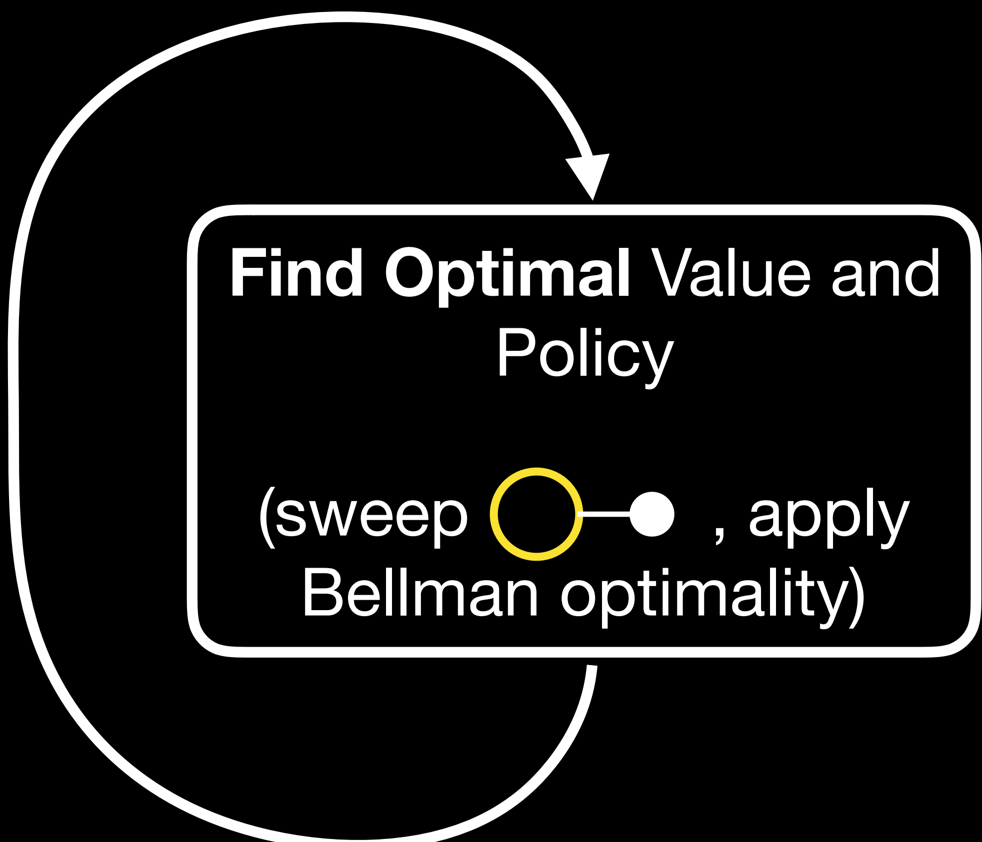
π(W|2): 1.0 (greedy)

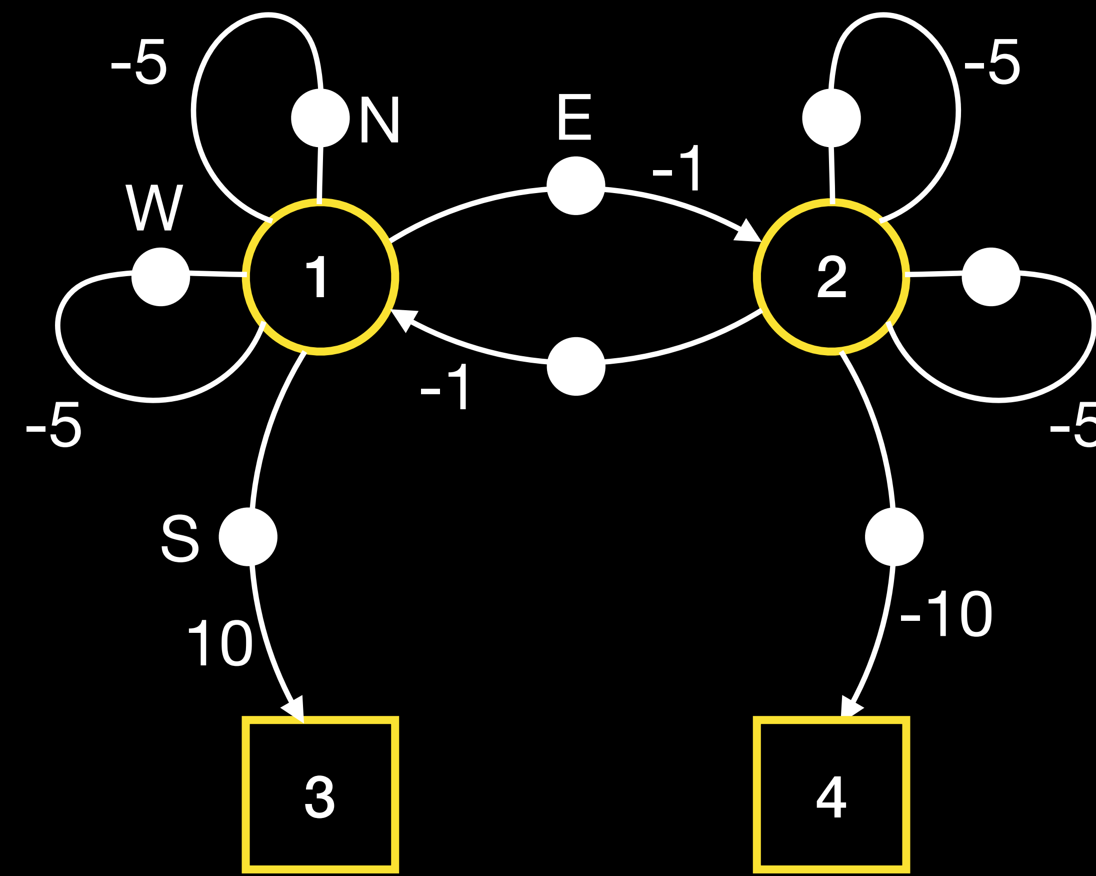


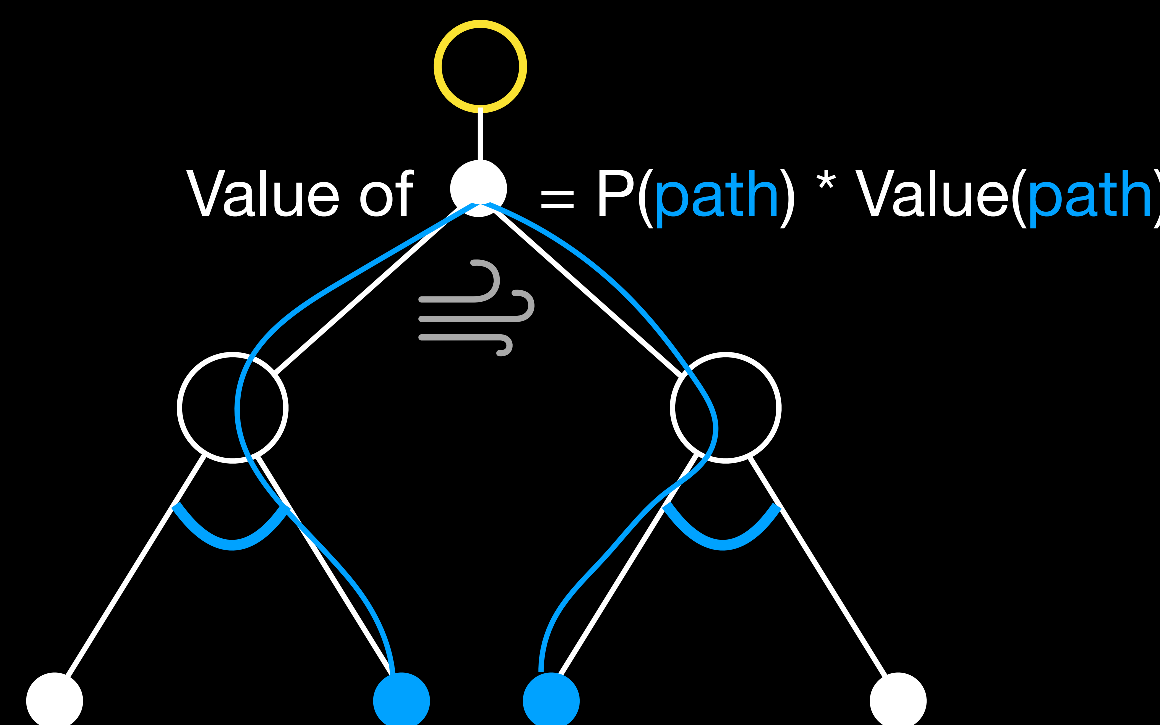


$$q_\pi(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s',a')$$

Value Iteration



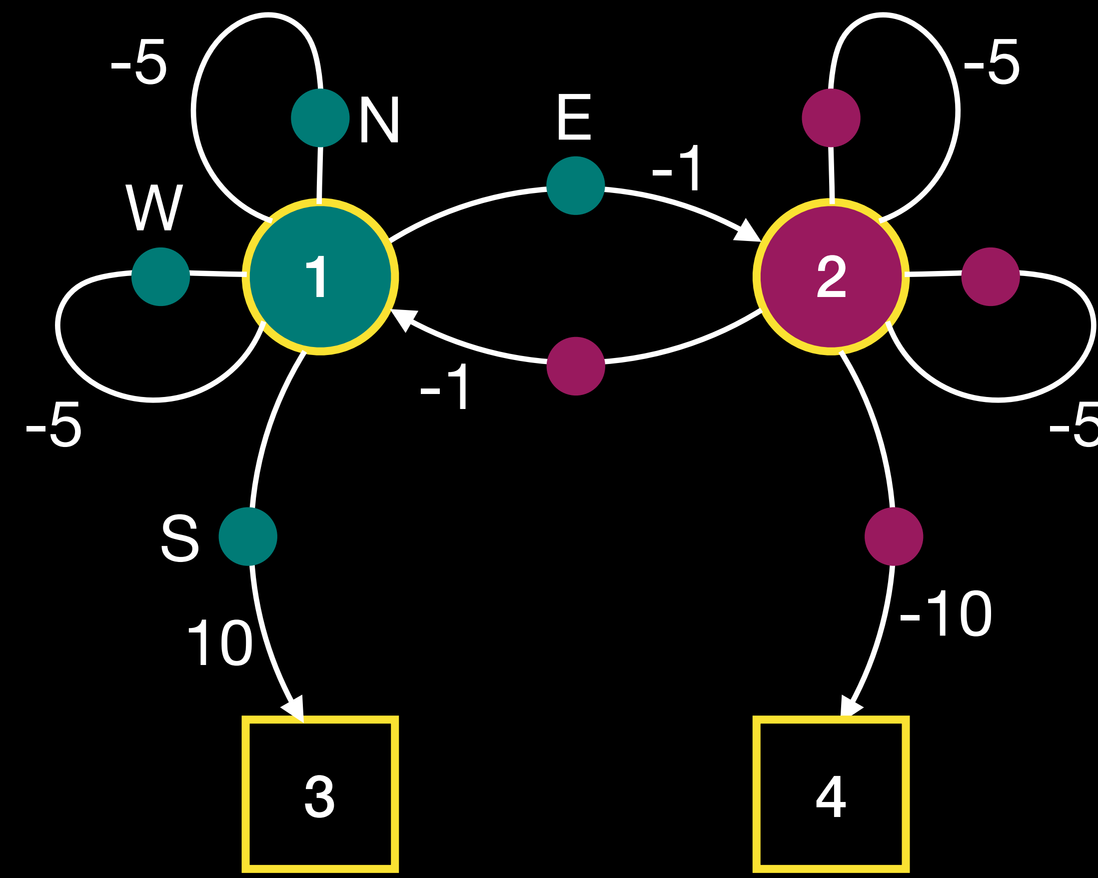


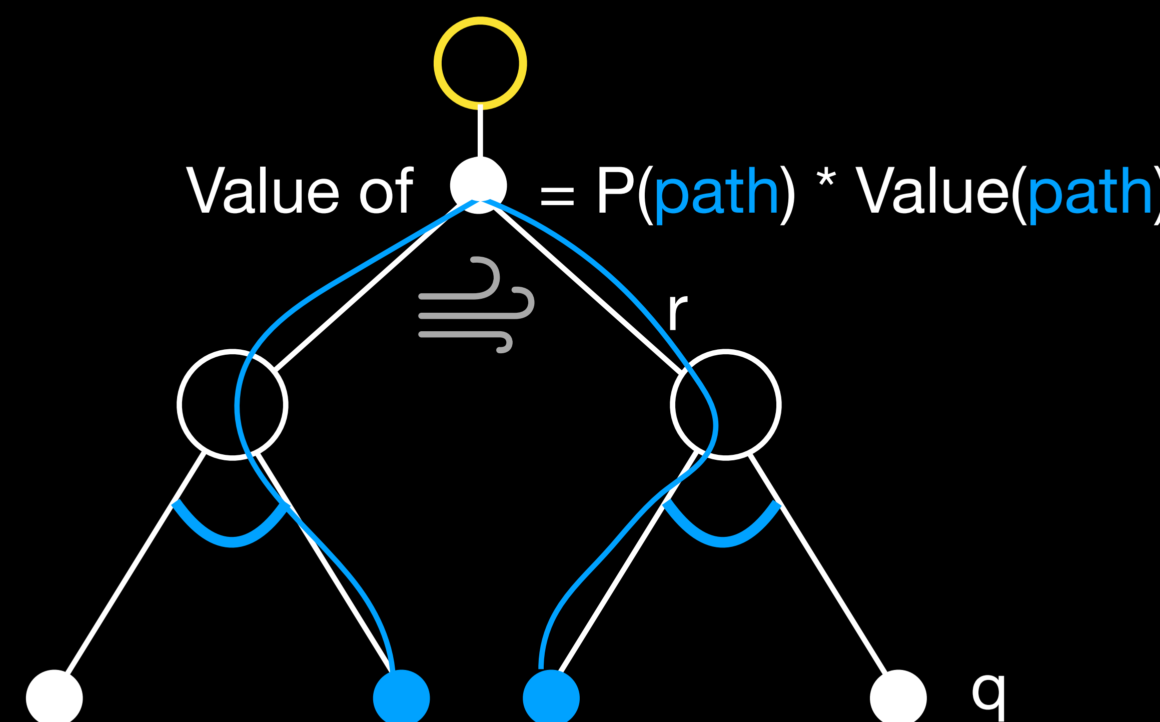


$$q_*(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q_*(s',a')$$

N: -5 + 0.9*0	N: -5 + 0.9*0
E: -1 + 0.9*0	E: -5 + 0.9*0
S: 10 + 0.9*0	S: -10 + 0.9*0
W: -5 + 0.9*0	W: -1 + 0.9*0

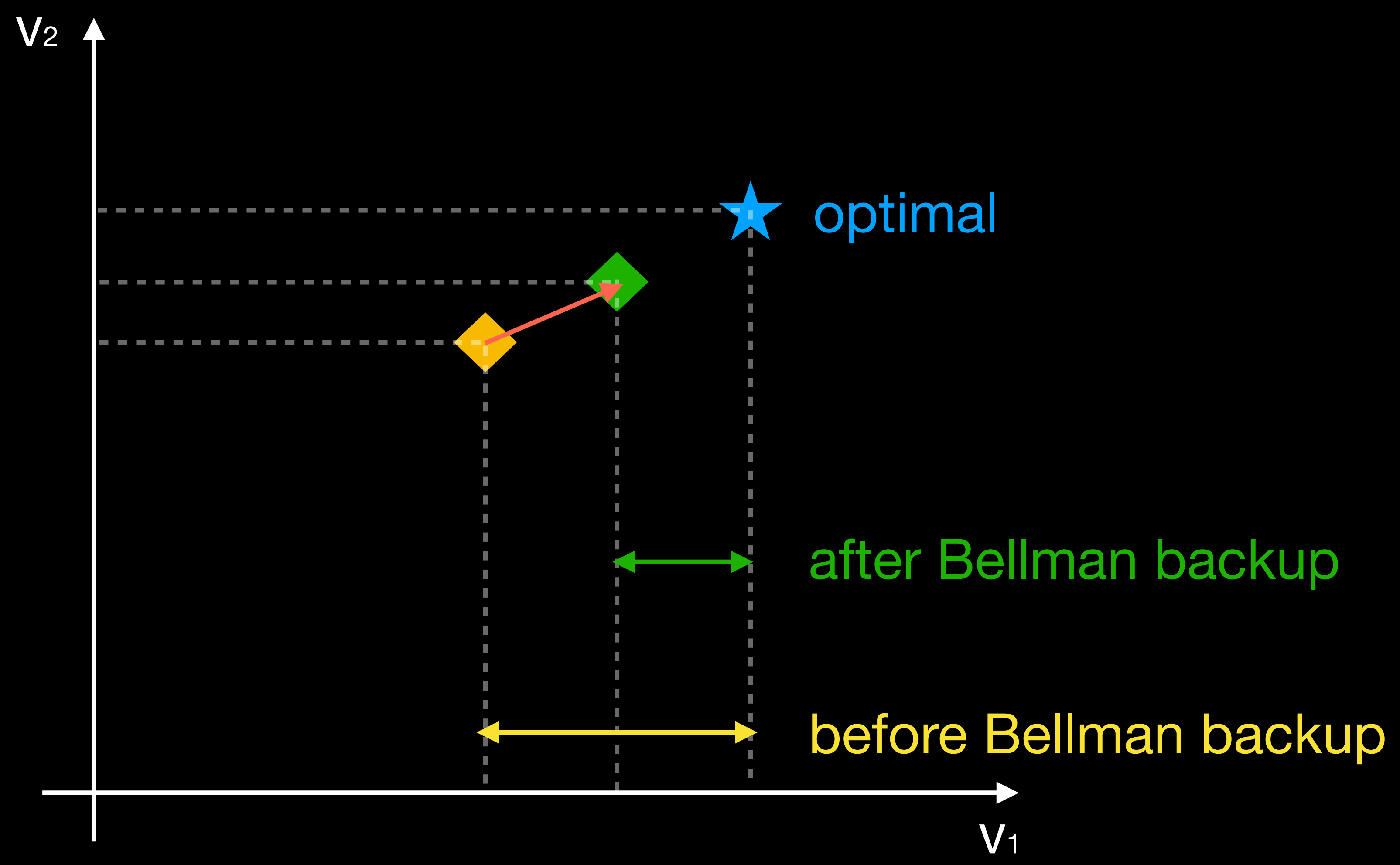
iteratively apply Bellman optimality equations until values do not change much





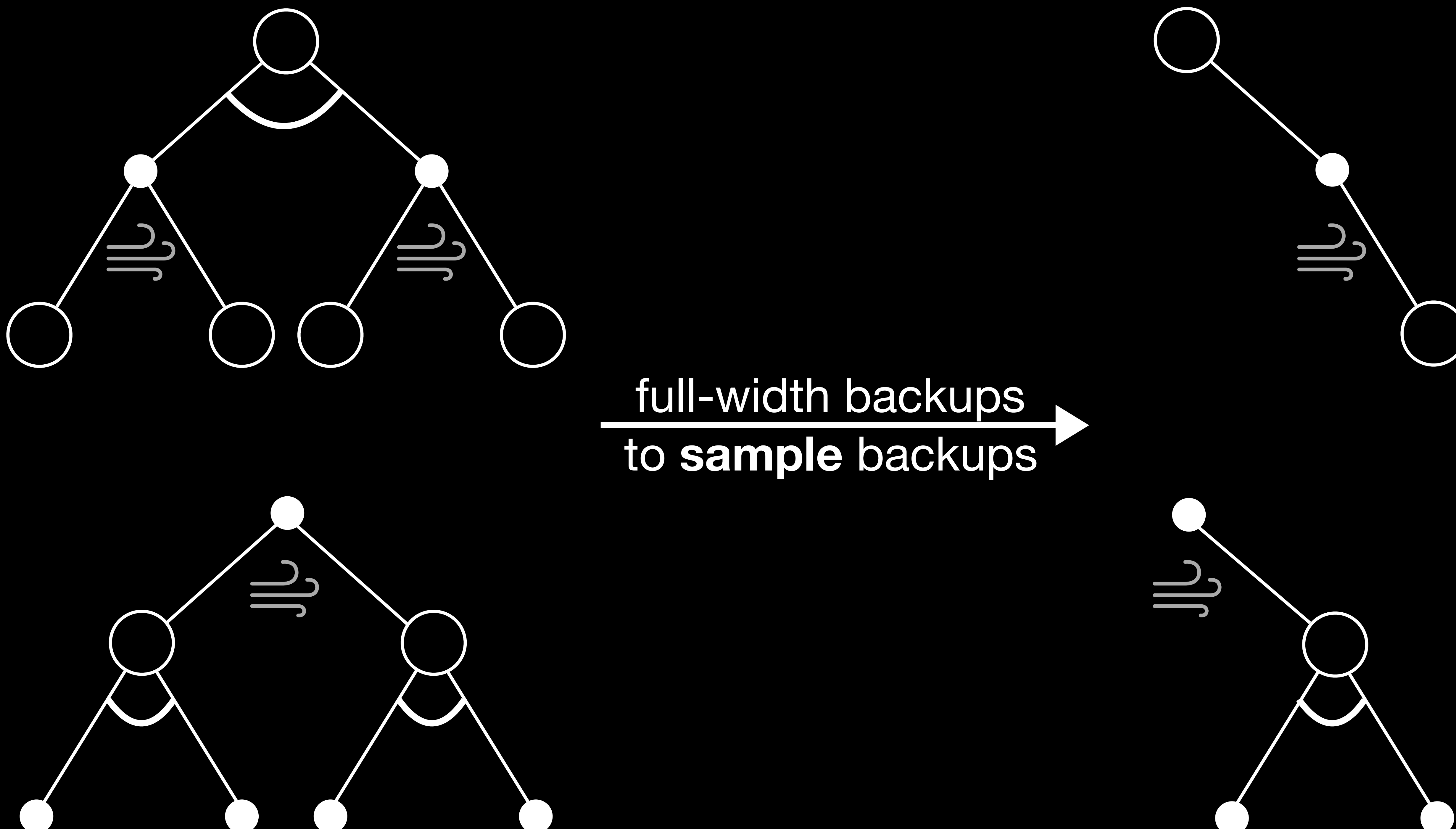
$$q_*(s,a) = r_s^a + \gamma \sum_{s'} P_{ss'}^a \max_{a'} q_*(s',a')$$

Bellman backups

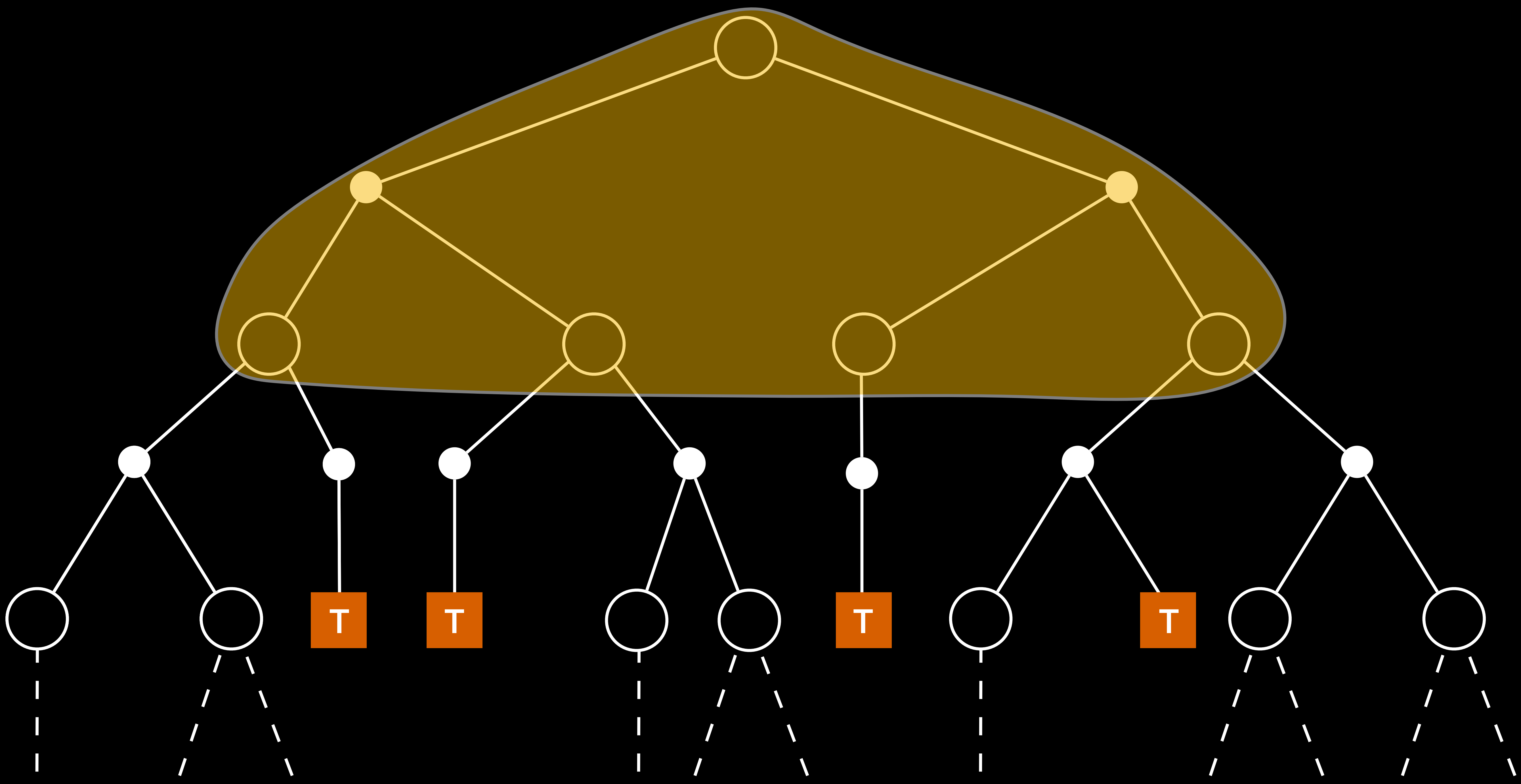


largest distance between values
decreases after Bellman backups

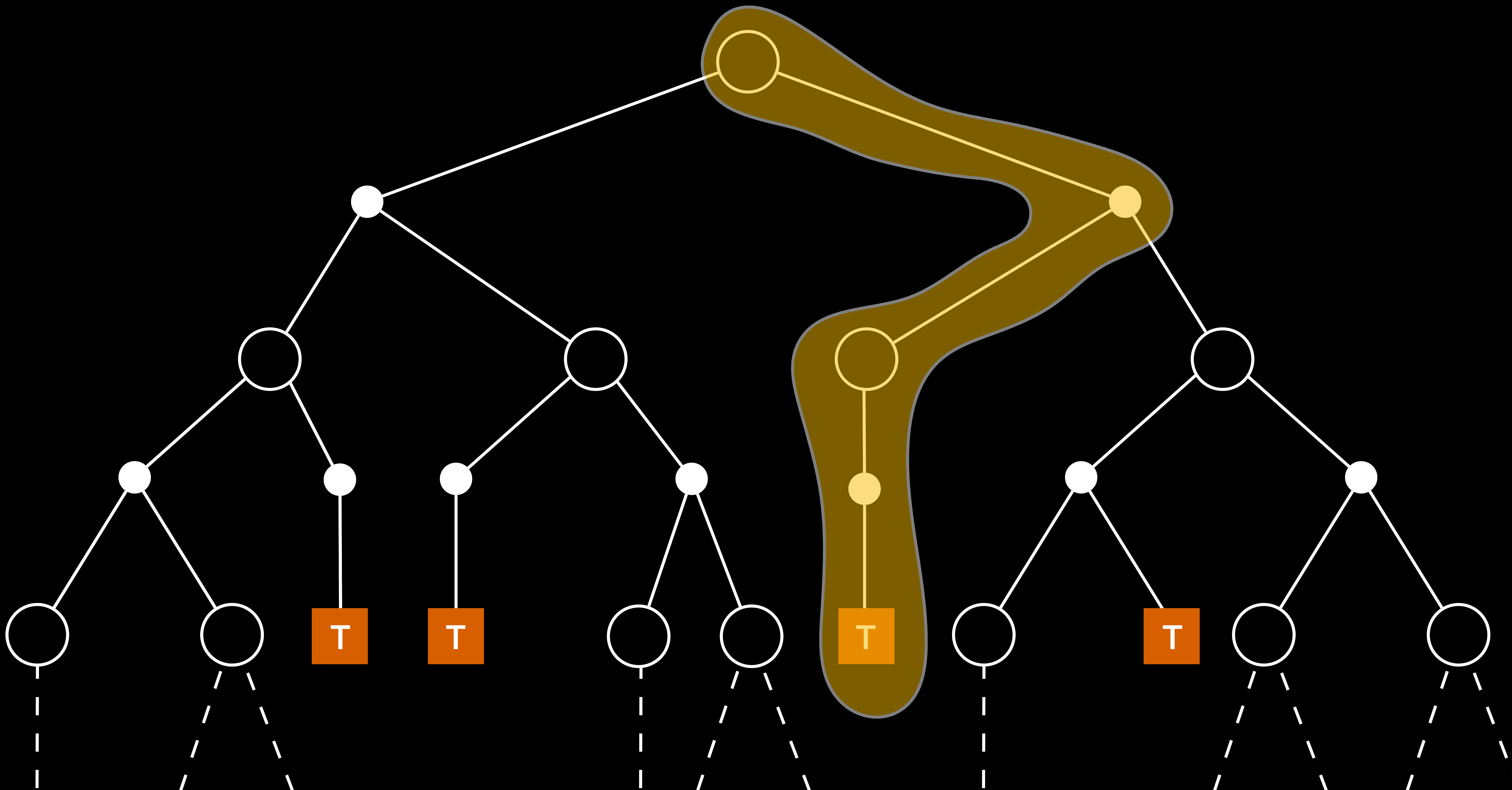
From DP to Learning



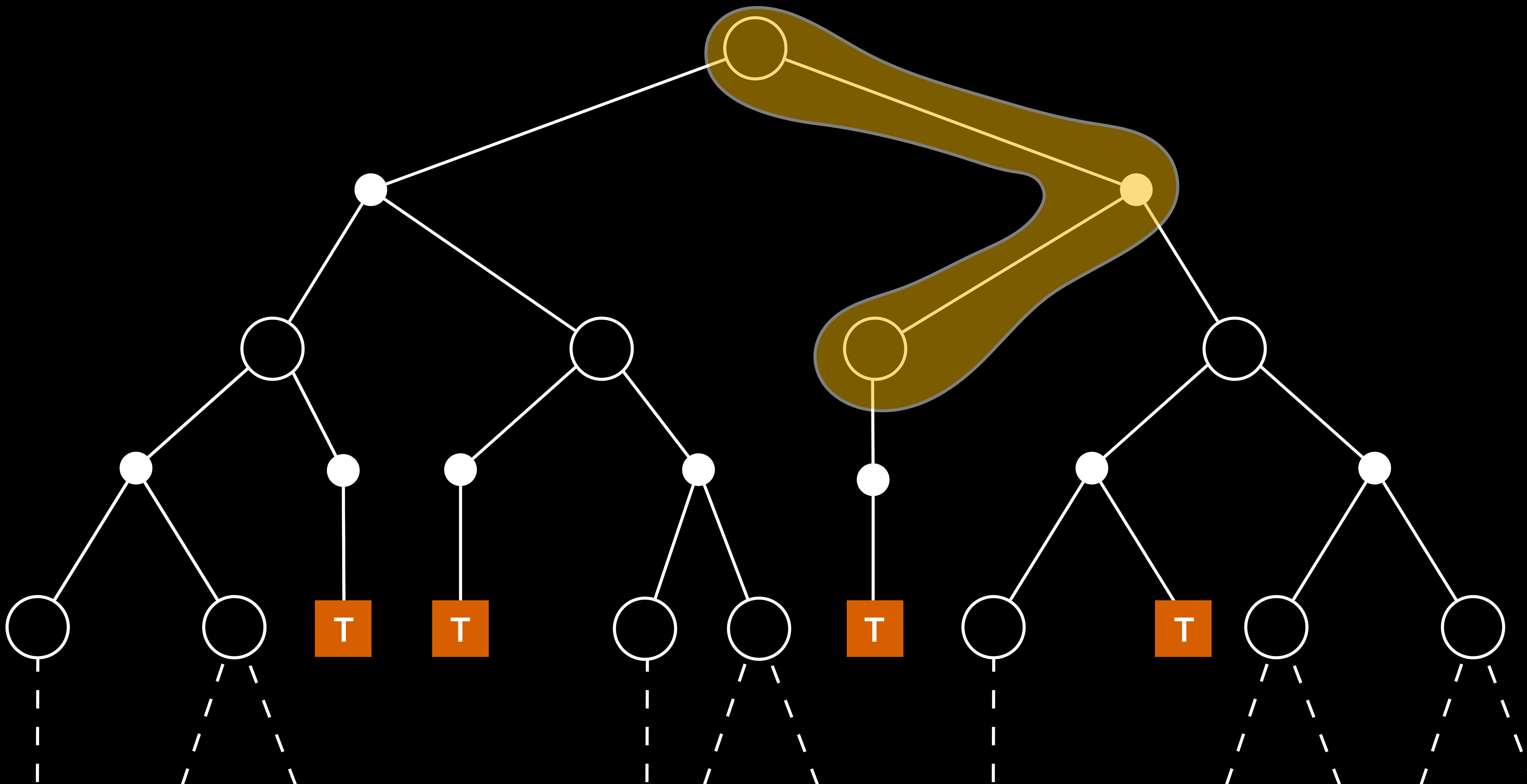
Full-width Backup



Backup with Sample Return



Backup with Guess



Incremental Updates

$$E\{R\} \approx \mu_k = \frac{1}{k}\sum_{\tau=1}^{k} R_\tau$$

batched

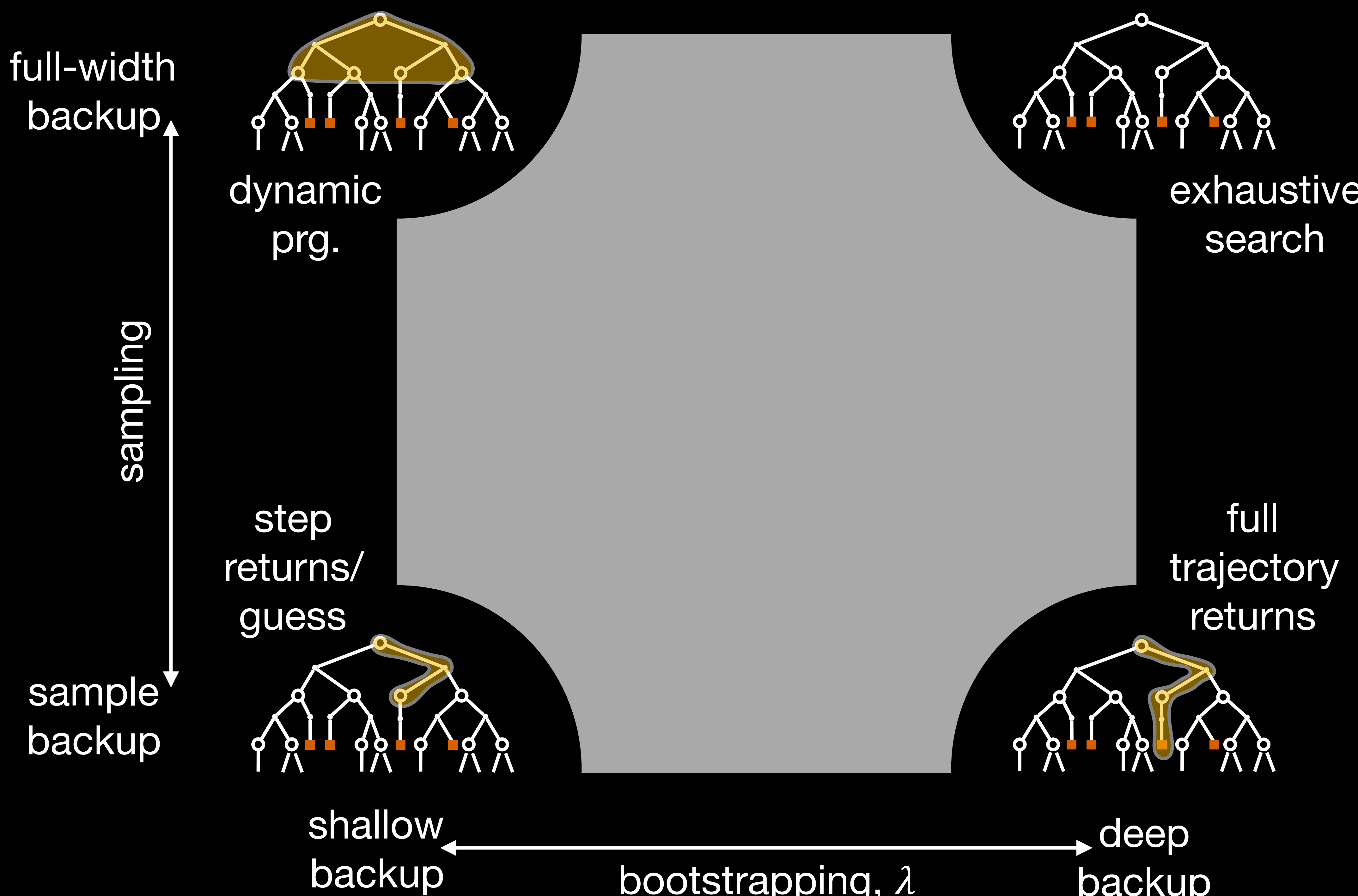
$$\mu_k = \mu_{k-1} + \frac{1}{k}\left(R_k - \mu_{k-1}\right)$$

incremental

$$\mu_k = \mu_{k-1} + \alpha\left(R_k - \mu_{k-1}\right)$$

running (saw this in Q-learning!)

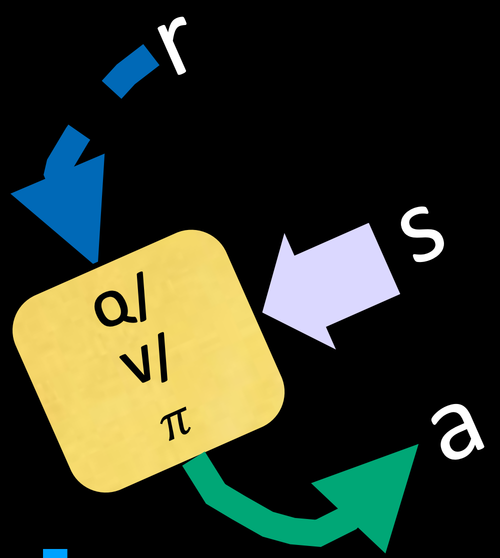
Sample and Bootstrap



It all comes down to:

estimating returns

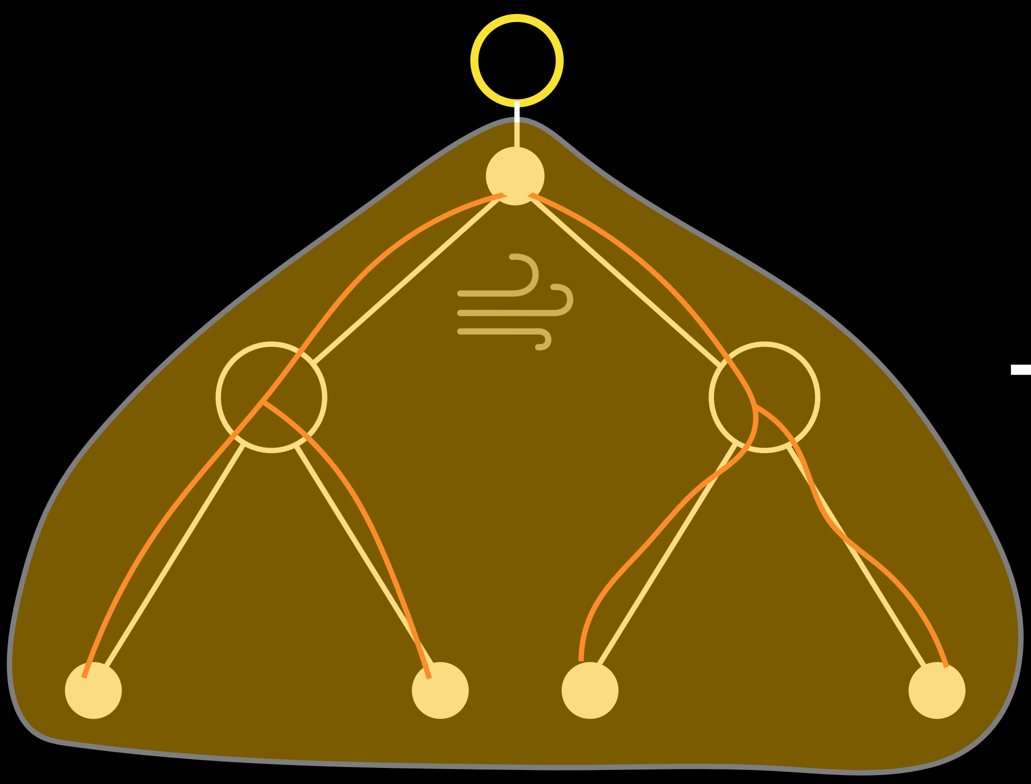
optimising towards achieving returns



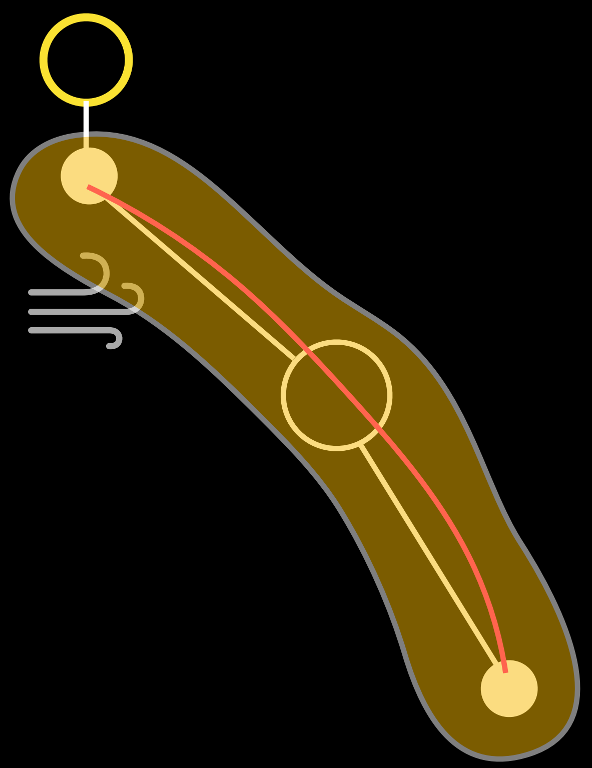
Q-learning


full-width backups
to **sample** backups
target policy
optimal

SARSA



full-width backups
to **sample** backups



target policy
same as
behaviour policy

scaling up RL with function approximation

Approximate Q-learning

e.g. linear approximation

$$Q_\theta(s,a) = \theta_0 f_0(s,a) + \theta_1 f_1(s,a) + \ldots + \theta_n f_n(s,a)$$

$$Q_{\text{target}} = (r_s^a + \gamma \max_{a'} Q(s',a'))$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{2}\left(Q_{\text{target}} - Q_\theta(s,a)\right)^2$$

gradient updates equivalent to tabular Q updates

Say $\theta \in \mathbb{R}^{|S| \times |A|}$, so $Q_\theta(s,a) = \theta_{sa}$

$$Q_{\text{target}} = r_s^a + \gamma \max_{a'} Q(s',a')$$

$$\theta_{sa} \leftarrow \theta_{sa} - \alpha \nabla_{\theta_{sa}} \frac{1}{2}\left(Q_{\text{target}} - \theta_{sa}\right)^2$$

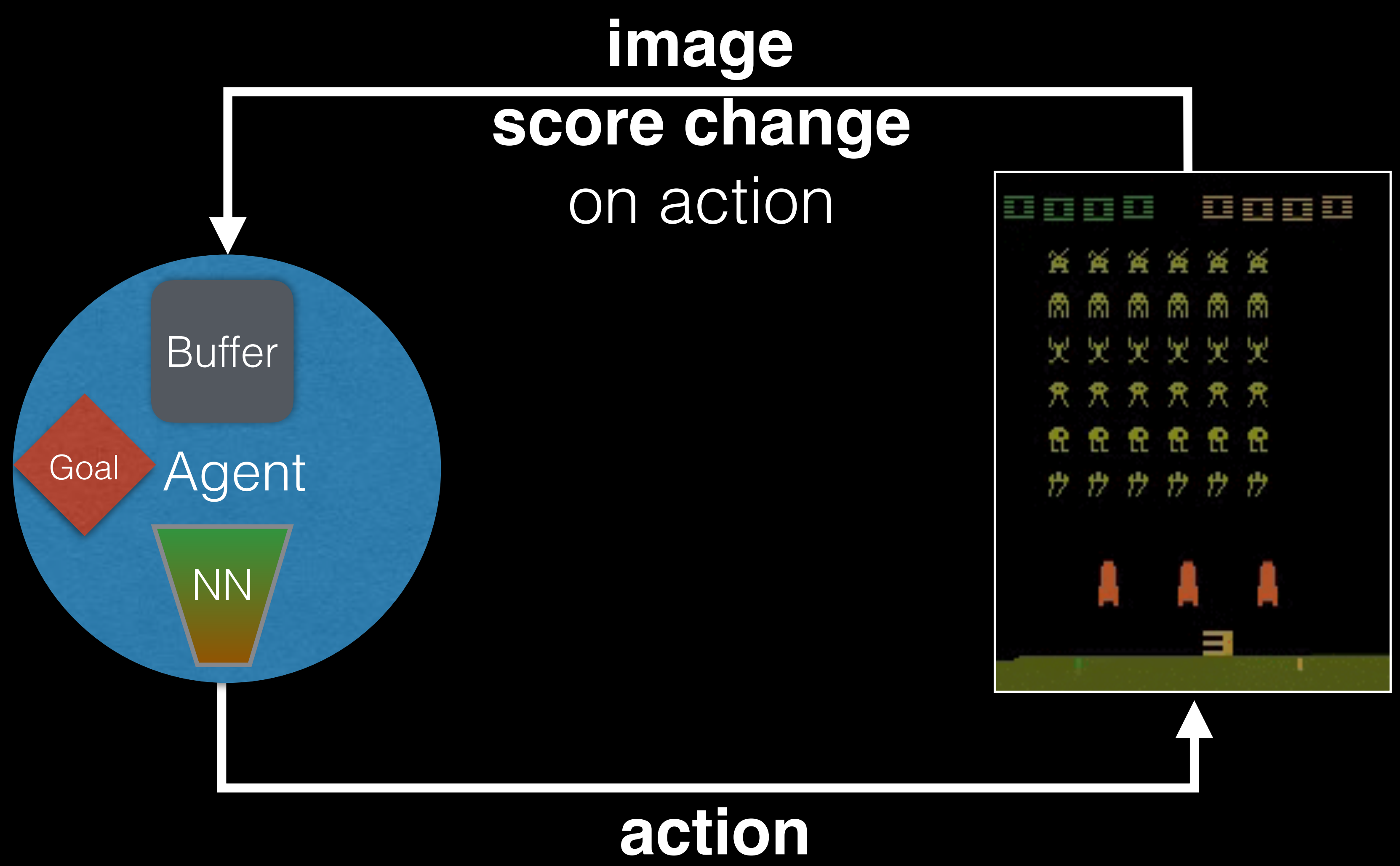
$$\theta_{sa} \leftarrow \theta_{sa} - \alpha\left(-Q_{\text{target}} + \theta_{sa}\right)$$

$$\theta_{sa} \leftarrow \theta_{sa} + \alpha\left(Q_{\text{target}} - \theta_{sa}\right)$$

$$\theta_{sa} \leftarrow (1-\alpha)\theta_{sa} + \alpha Q_{\text{target}}$$

tabular equivalent

DQN



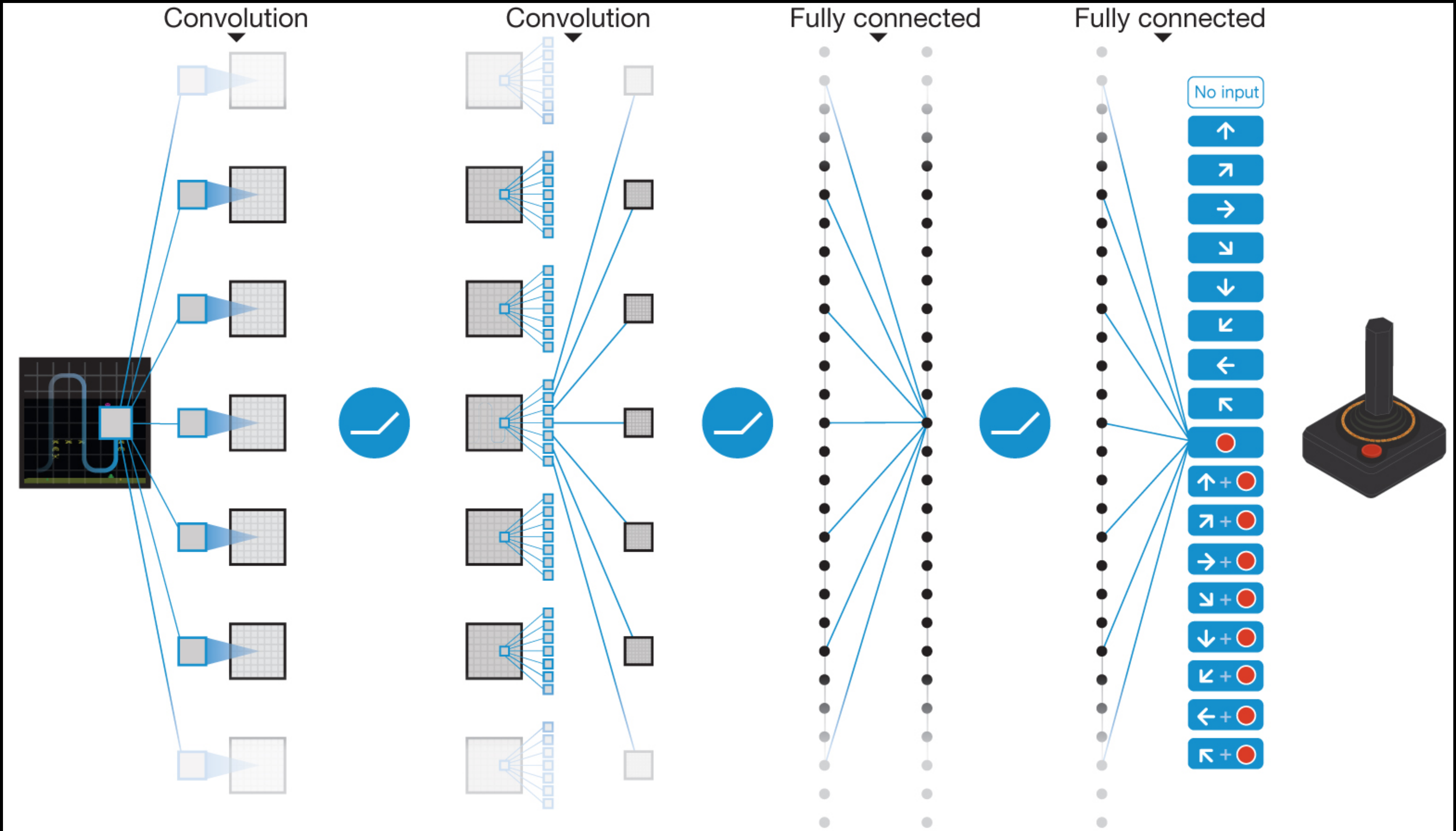
Human-level control through deep reinforcement learning, Mnih et. al., Nature 518, Feb 2015

human level game control

- **pixel** input
- **18 joystick/button positions** output
- **change in game score** as feedback
- **convolutional net representing Q**
- **backpropagation** for training!

Human-level control through deep reinforcement learning,
Mnih et. al., Nature 518, Feb 2015
http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html

neural network



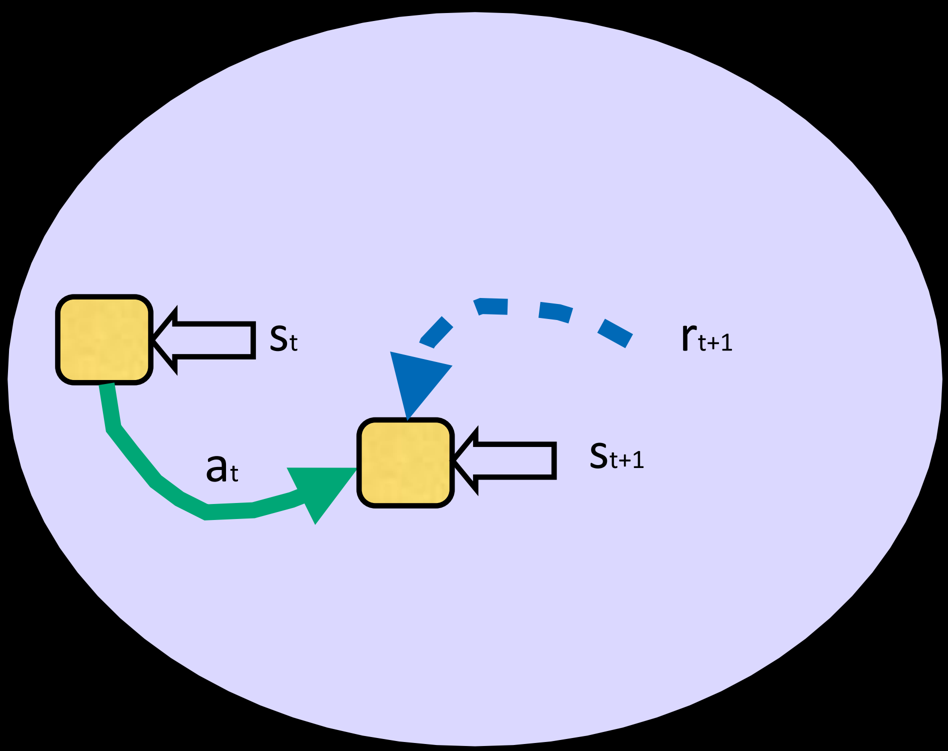
backpropagation

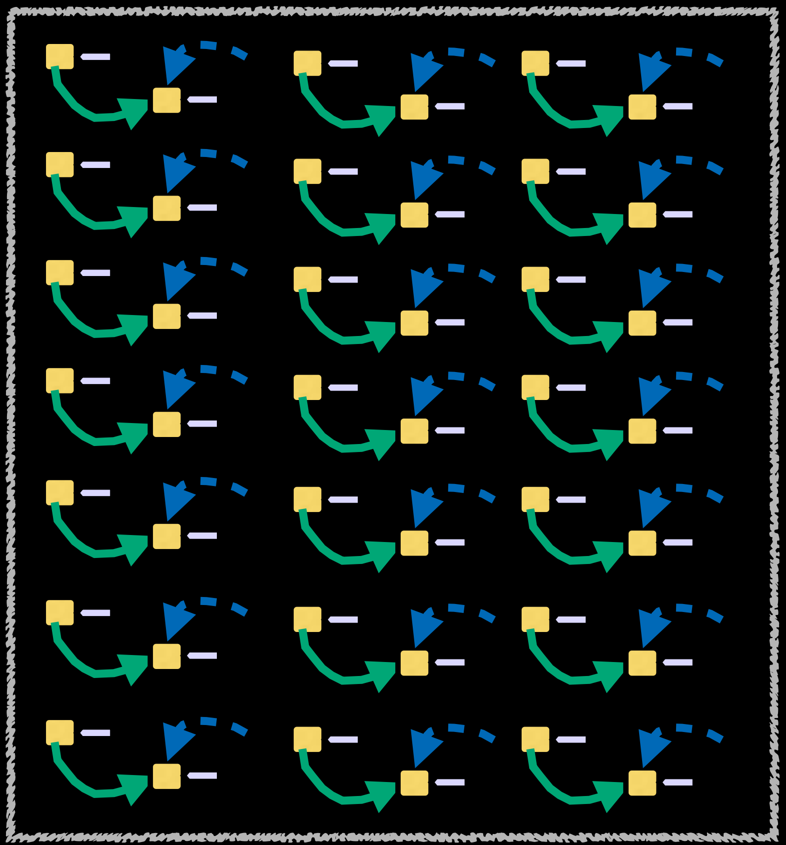
What is the **target** against which to minimise error?

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

experience replay buffer





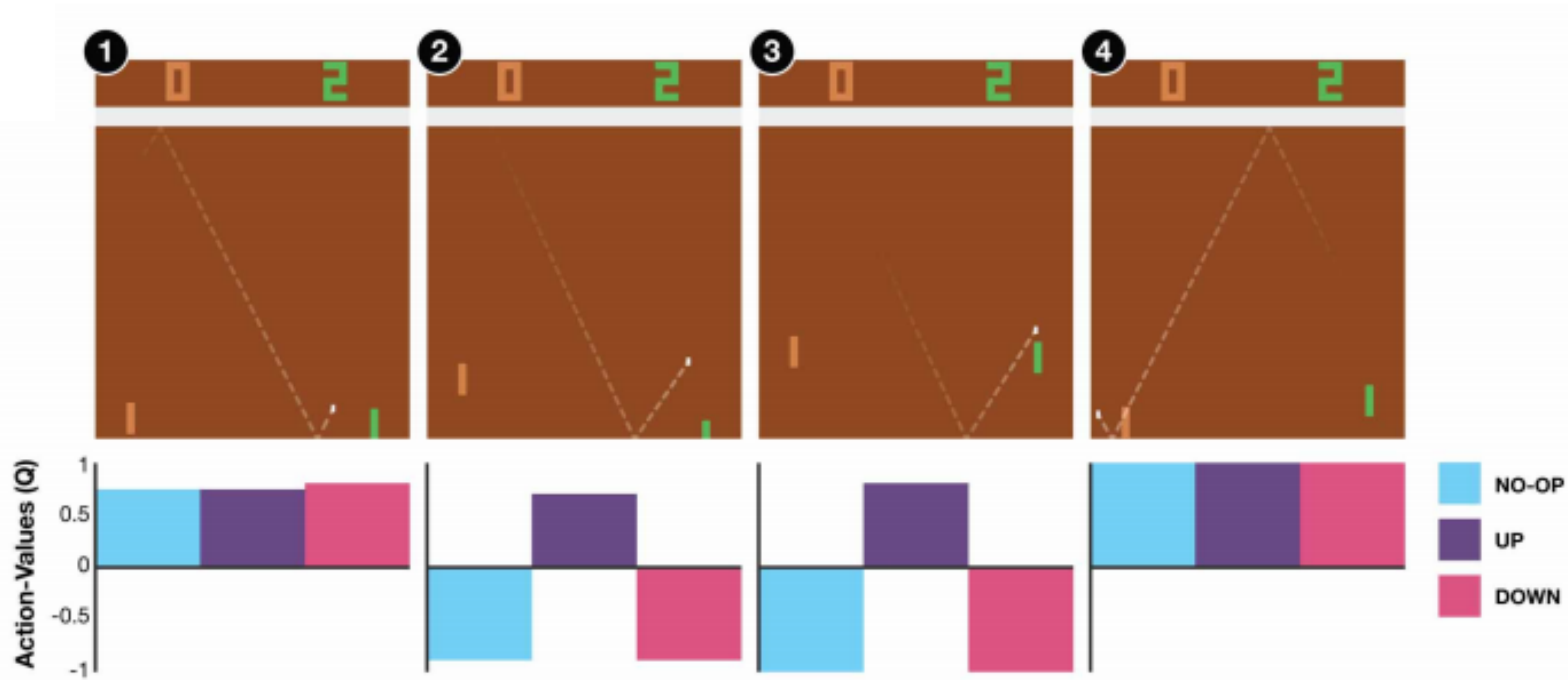
save transition in memory

randomly **sample** from memory for training = i.i.d

freeze target

freeze

$$\left(\boxed{r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-)} - Q(s, a, \mathbf{w}) \right)^2$$



https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Human-level control through deep reinforcement learning, Mnih et. al., Nature 518, Feb 2015

however
training is

SLOOOOOo....W

parallelise…

Parallel Asynchronous Training

value and **policy** based methods

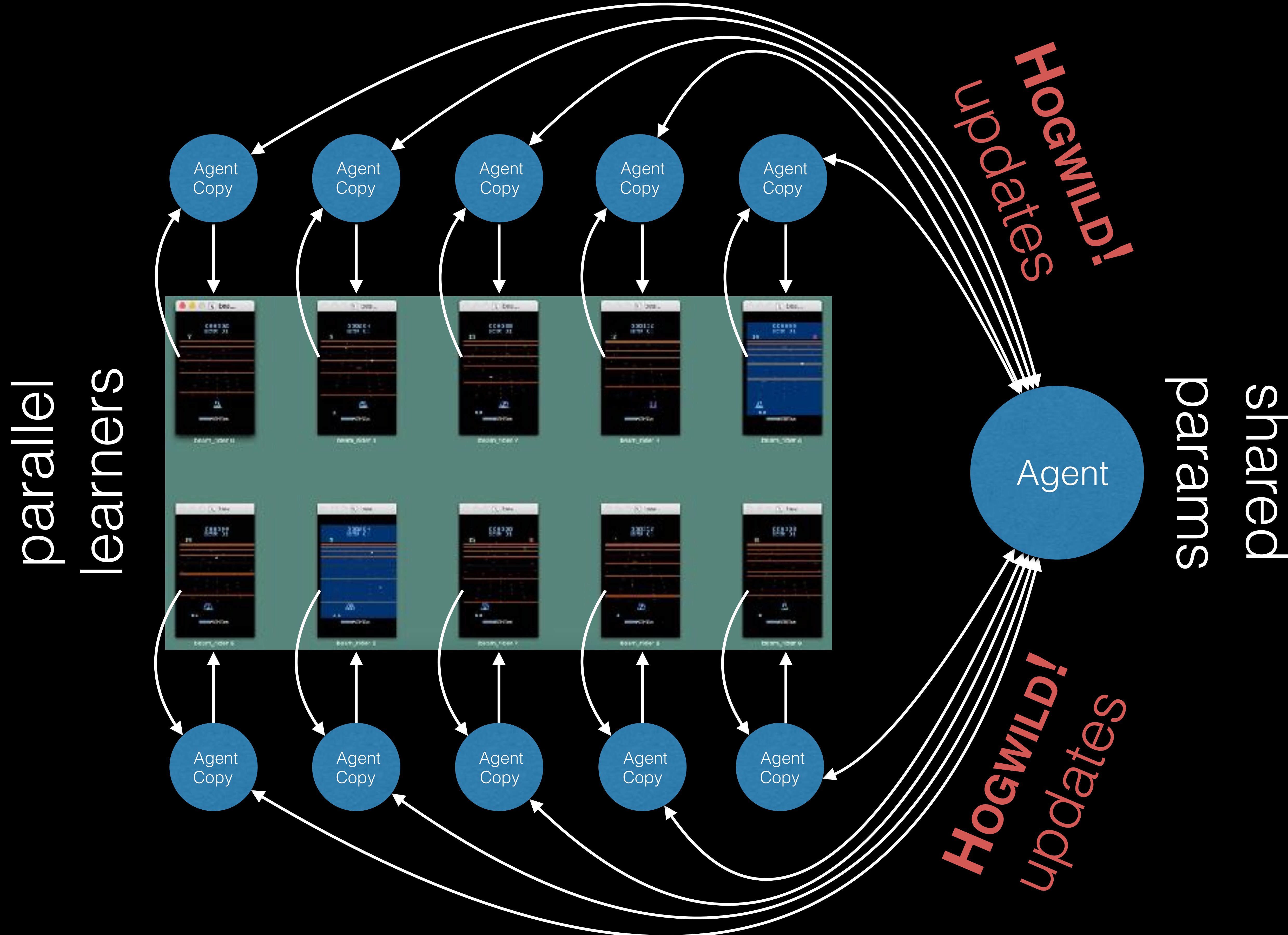
https://youtu.be/0xo1Ldx3L5Q

parallel agents

shared parameters

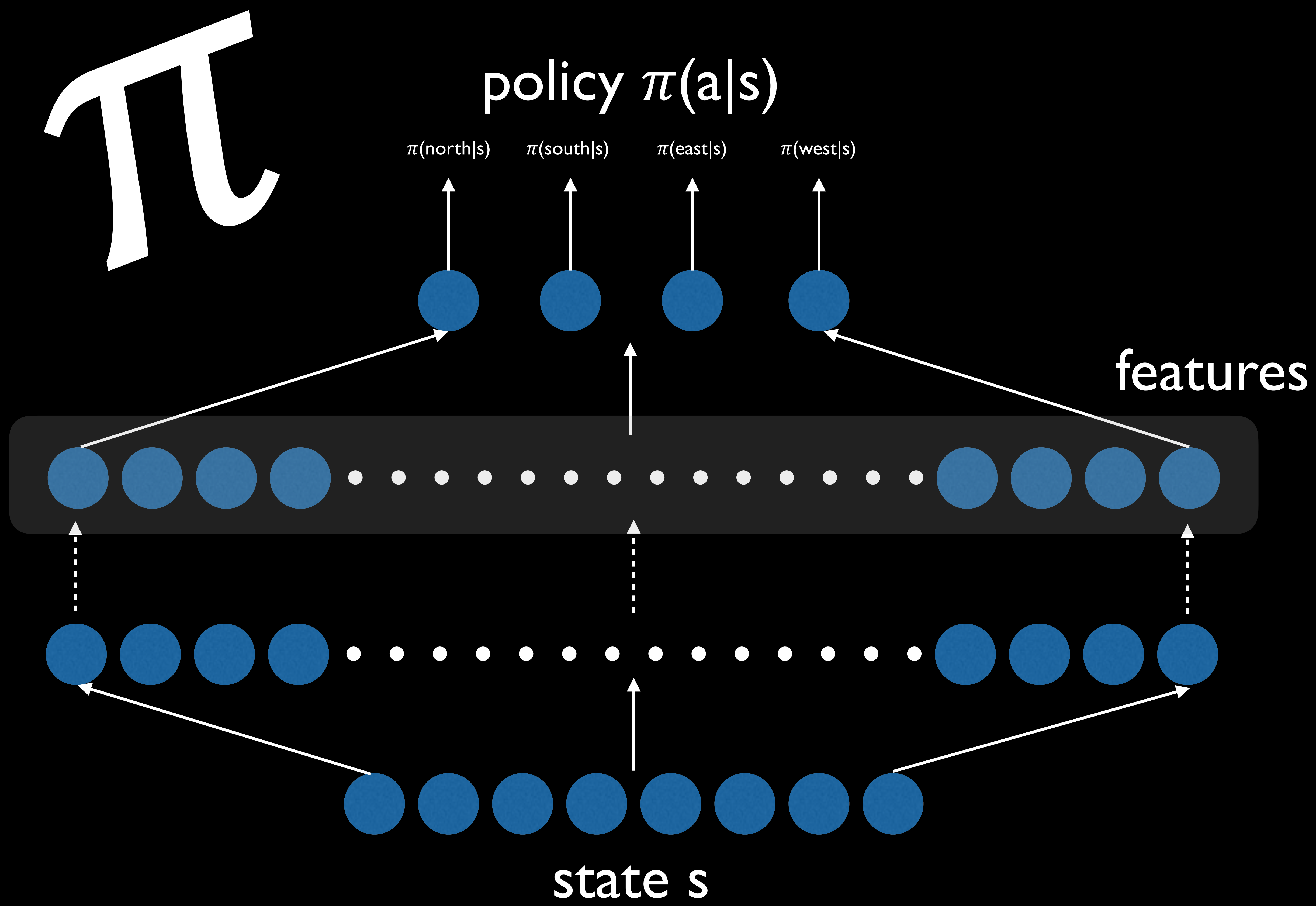
lock-free updates

Asynchronous Methods for Deep Reinforcement Learning, Mnih et. al., **ICML 2016**



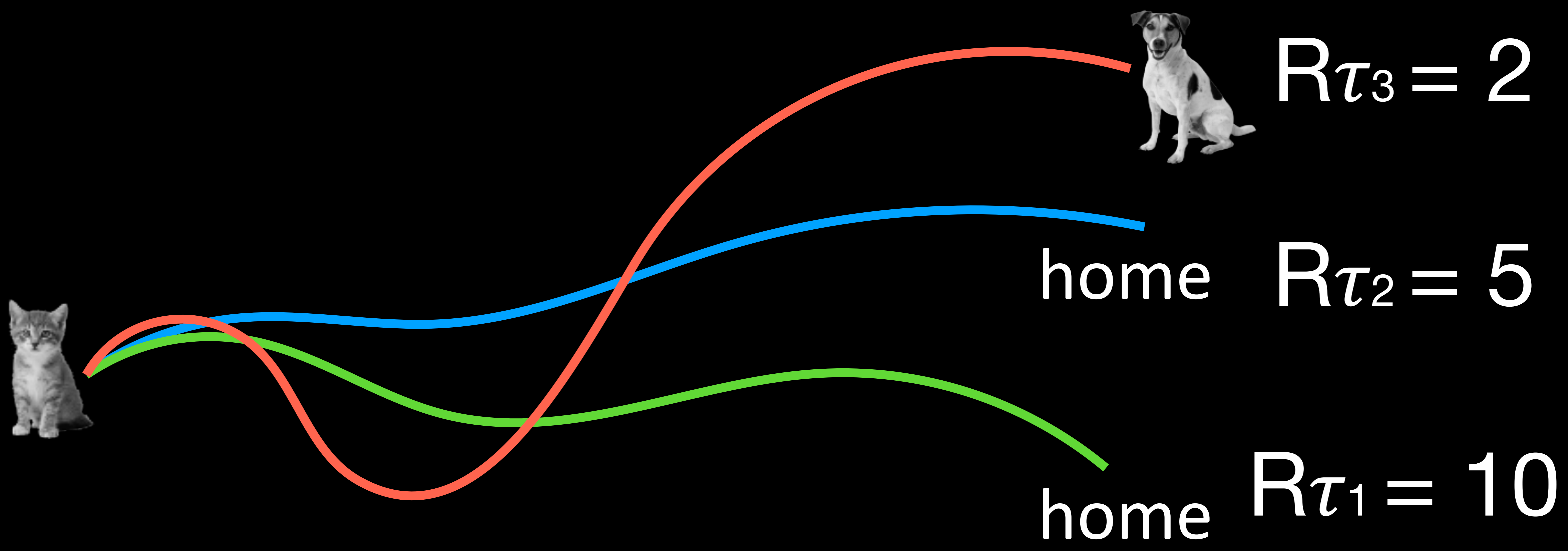
https://github.com/traai/async-deep-rl

Policy Based


π
policy π(a|s)
π(north|s)
π(south|s)
π(east|s)
π(west|s)
features
state s

Intuition

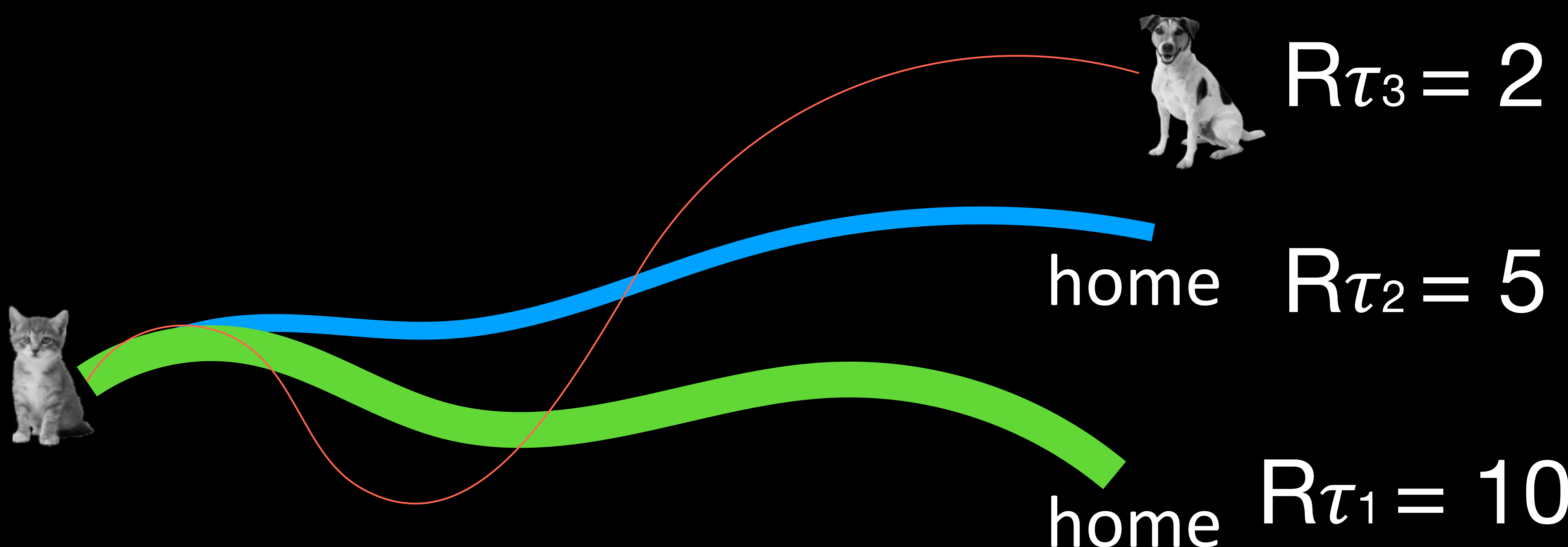
$$\tau : s_1, a_1, r_1^1, s_2, a_2, r_2^2, \ldots, s_{H-1}, a_{H-1}, r_{H-1}^{H-1}$$



Intuition

π**(a|s)** along path with high return higher

$$\tau : s_1, a_1, r_1^1, s_2, a_2, r_2^2, \ldots, s_{H-1}, a_{H-1}, r_{H-1}^{H-1}$$



probabilities are relative

Revisiting the Objective

$$\tau : s_1, a_1, r_1^1, s_2, a_2, r_2^2, \ldots, s_{H-1}, a_{H-1}, r_{H-1}^{H-1}$$

$$\max_\theta \mathrm{E}_\tau \left\{ \boxed{\sum_{t=0}^{H-1} r_{s_t}^{a_t}} \mid \pi_\theta \right\}$$

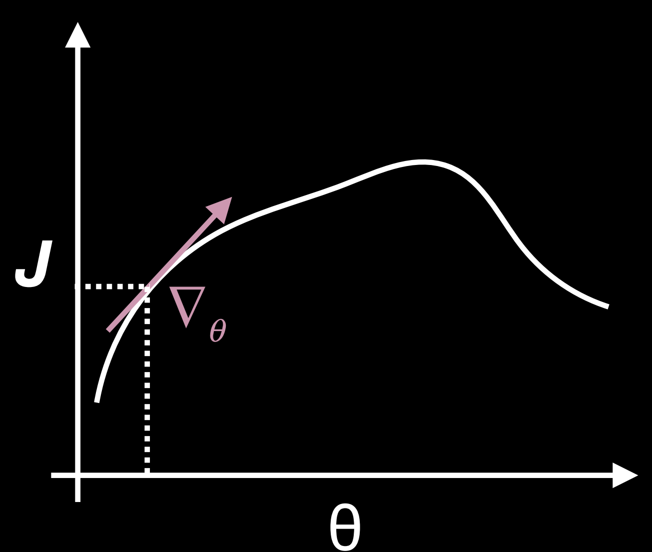
$$\max_\theta J(\theta) = \max_\theta \sum_\tau P(\tau \mid \theta) \boxed{R(\tau)}$$

Samples → Gradient

$$J(\theta) = \sum_{\tau} P(\tau \mid \theta) R(\tau)$$

$$\max_{\theta} J(\theta)$$

$$\theta \leftarrow \theta + \nabla_{\theta} J(\theta)$$



gradient
via
sampling

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{\tau} \underline{P(\tau \mid \theta)} R(\tau)$$

$$= \sum_{\tau} \nabla_{\theta} P(\tau \mid \theta) R(\tau)$$

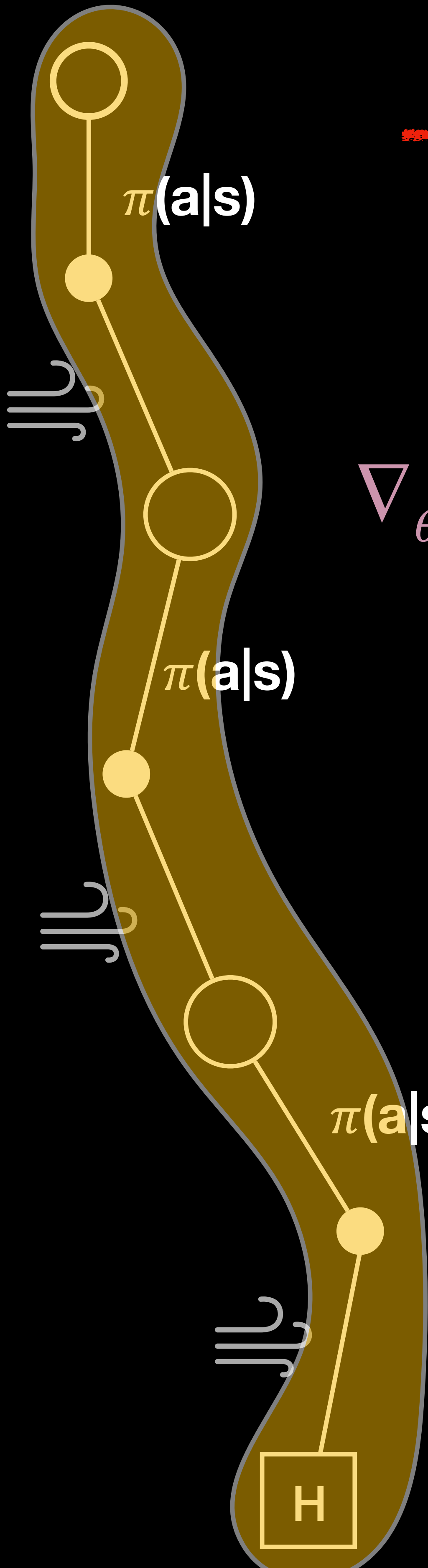
$$= \sum_{\tau} \frac{P(\tau \mid \theta)}{P(\tau \mid \theta)} \nabla_{\theta} P(\tau \mid \theta) R(\tau)$$

$$= \sum_{\tau} P(\tau \mid \theta) \frac{\nabla_{\theta} P(\tau \mid \theta)}{P(\tau \mid \theta)} R(\tau)$$

$$= \sum_{\tau} \underline{P(\tau \mid \theta)} \nabla_{\theta} \log P(\tau \mid \theta) R(\tau)$$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} \log P(\tau^{(i)} \mid \theta) R(\tau^{(i)})$$

~~Dynamics Model~~



$$\nabla_\theta \log P(\tau \mid \theta) = \nabla_\theta \log \left[\prod_{t=0}^{H-1} \overbrace{P(s_{t+1} \mid s_t, a_t)}^{\text{dynamics model}} \cdot \overbrace{\pi_\theta(a_t \mid s_t)}^{\text{policy}} \right]$$

$$= \nabla_\theta \left[\cancel{\sum_{t=0}^{H-1} \log P(s_{t+1} \mid s_t, a_t) +} \sum_{t=0}^{H-1} \log \pi_\theta(a_t \mid s_t) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H-1} \log \pi_\theta(a_t \mid s_t) = \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

$$\boxed{\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \right) R(\tau^{(i)})}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})\right)R(\tau^{(i)})$$

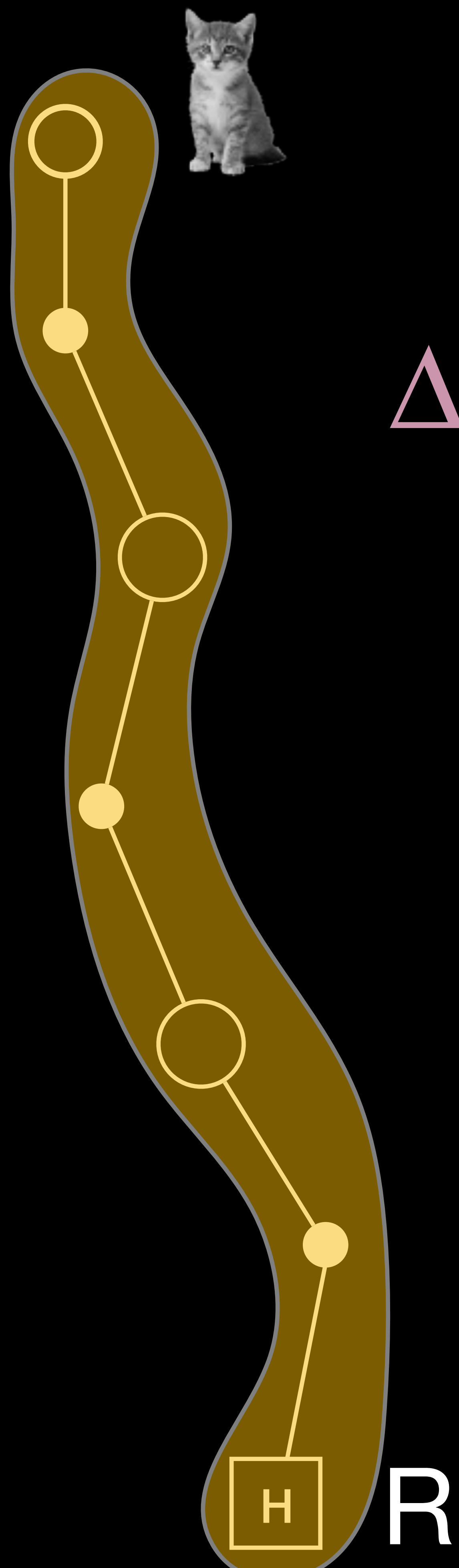
For each action a_t in state s_t
during each trajectory m

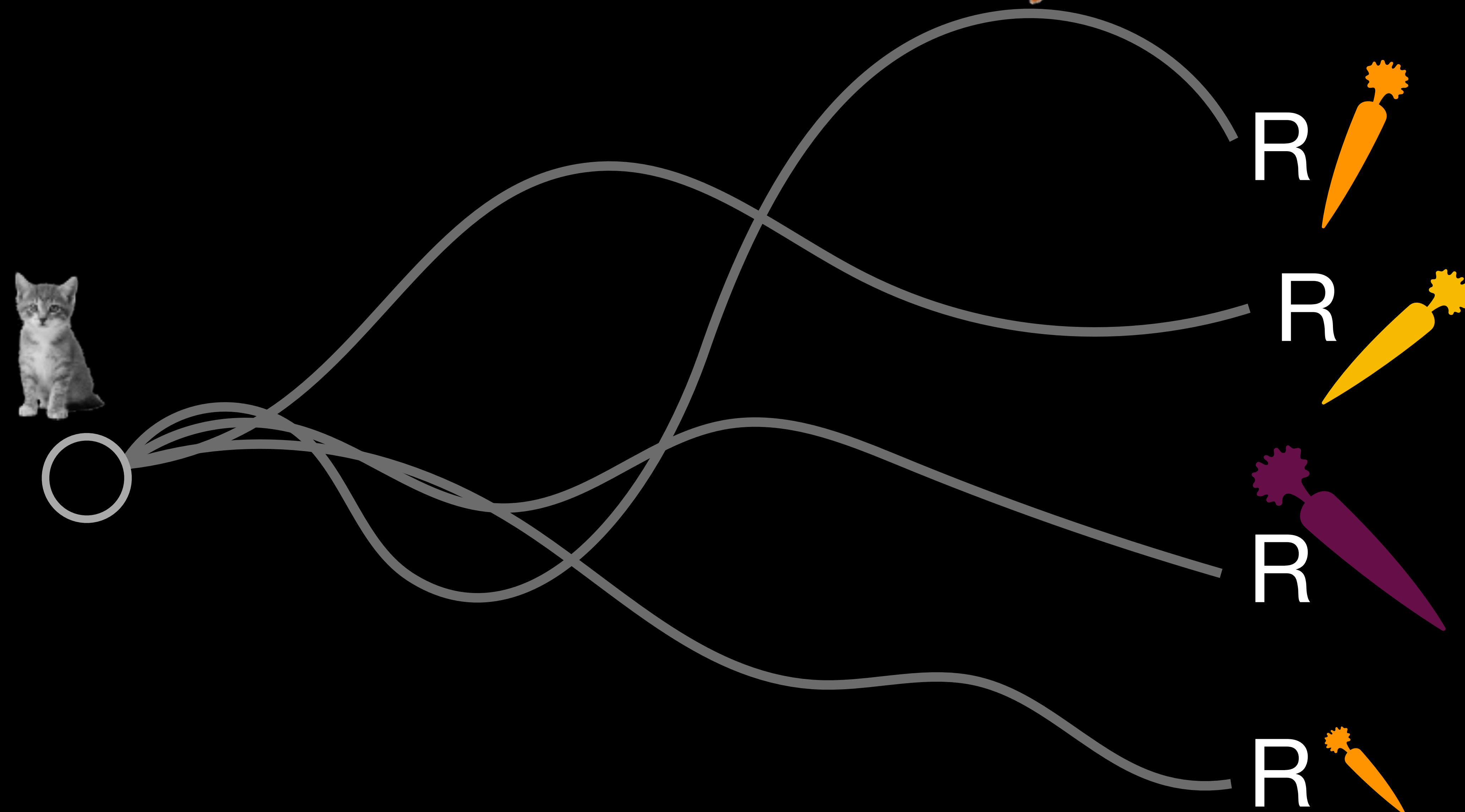
$$\underline{\nabla_\theta \log \pi_\theta(a_t \mid s_t)}\,\underline{R(\tau)}$$

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x 

Noisy Gradient

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x

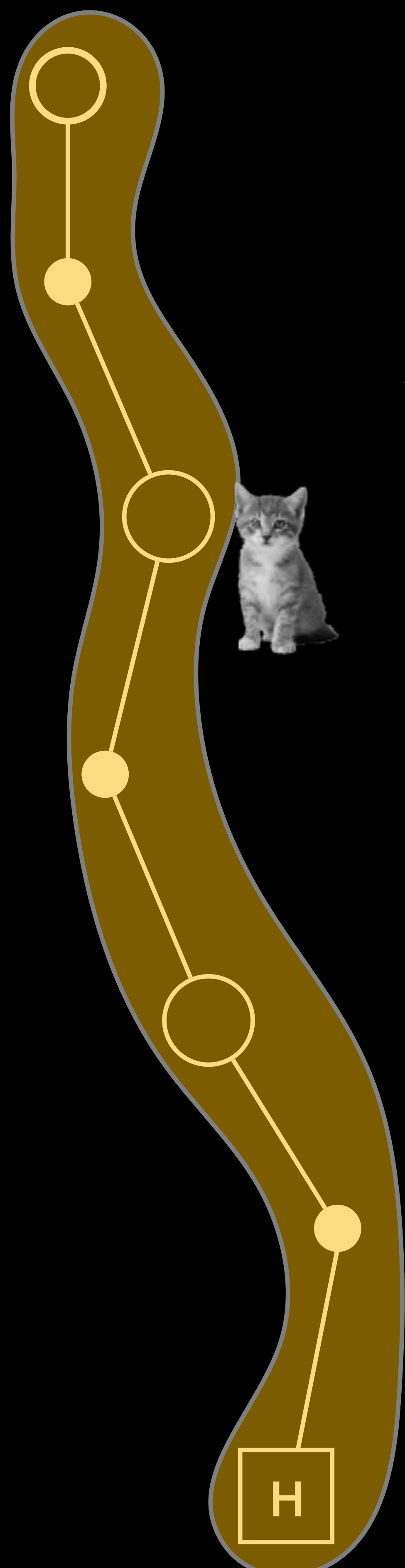




Reduce Noise

$R(\tau_{t \text{ onwards}})$

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x 



R time t onwards

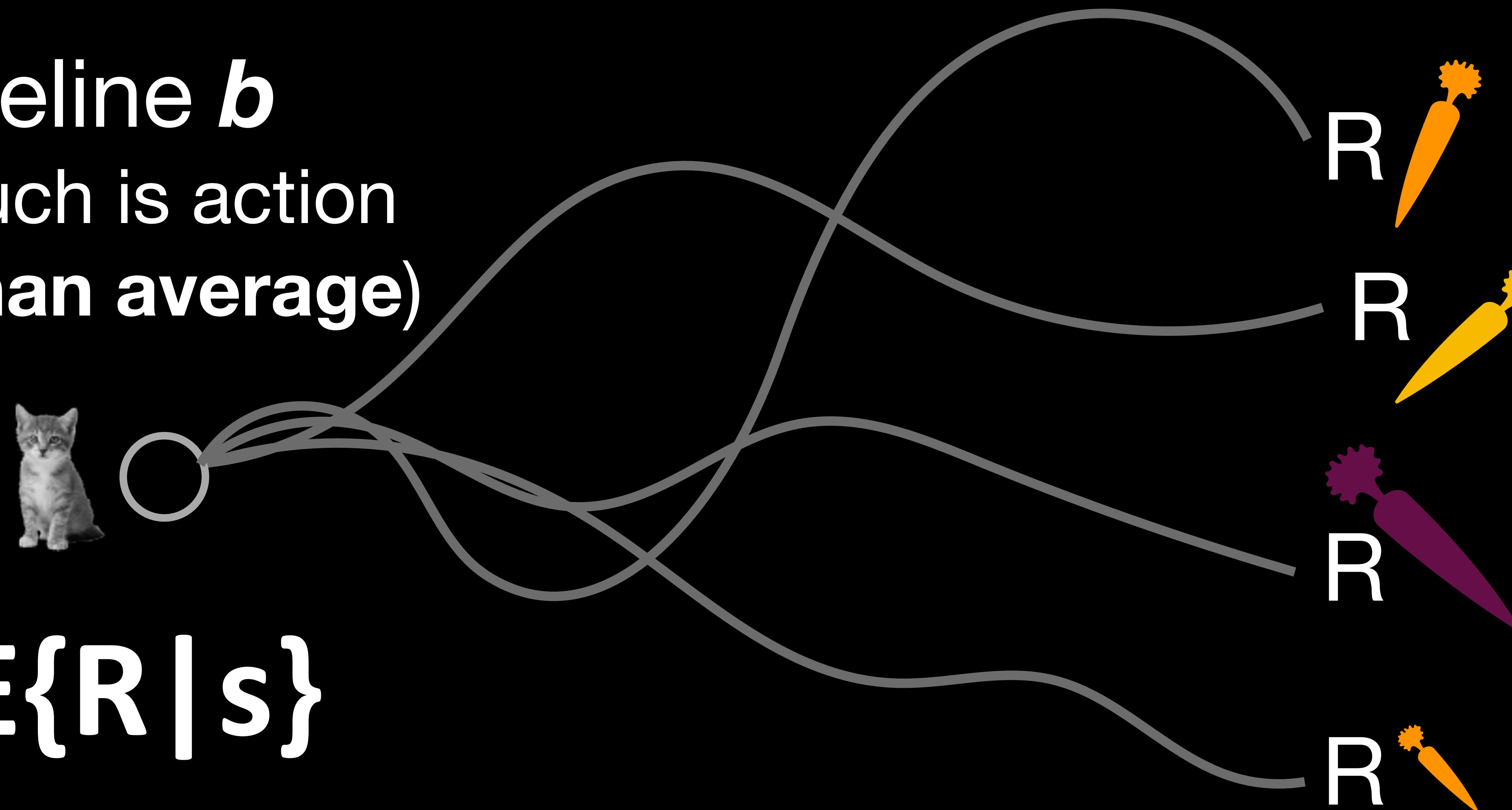
$R(\tau_{t\ \text{onwards}}) - b$

Reduce Noise

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x

baseline ***b***

(how much is action **better than average**)



V=E{R|*s*}

Reduce Noise

$R(\tau_{t \text{ onwards}}) - V(s_t)$

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x 

baseline ***b***
(how much is action **better than average**)

V=E{R|*s*}

Actor-Critic

Reduce Noise

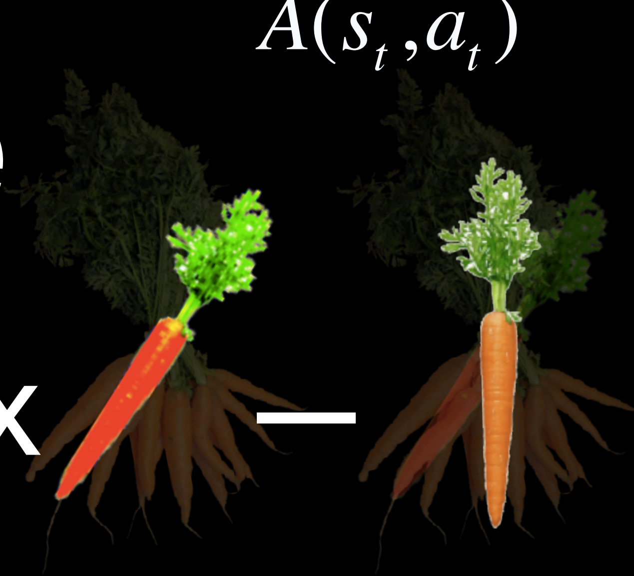
$Q(s_t, a_t) - V(s_t)$

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x

critic **Q**
(expected long term value of action)

$$\mathbf{Q} = \mathrm{E}\{\mathrm{R} \mid \mathrm{s,a}\}$$
$$= \mathrm{E}\{\mathrm{r} + \gamma \mathrm{V}\}$$

Reduce Noise

$\Delta\theta$ to increase $\pi_\theta(a_t \mid s_t)$ x 

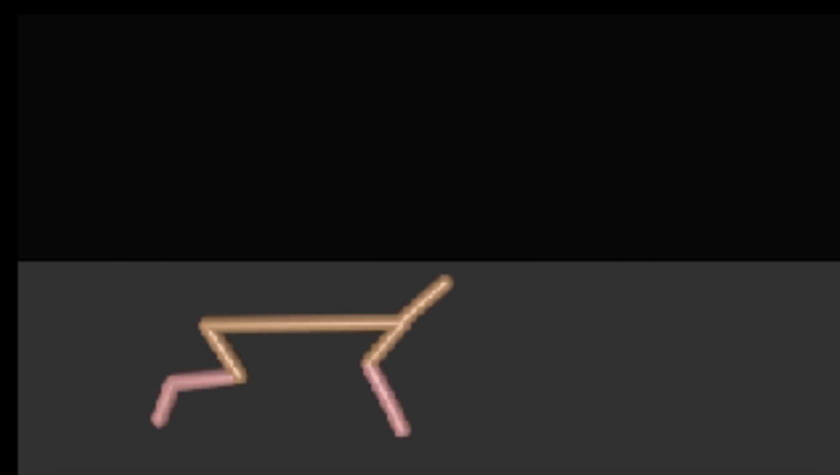
A = Q - V
(**advantage** of an action)

(how much is action better than average) parallelise…

Parallel Asynchronous Training

value and **policy** based methods

https://youtu.be/0xo1Ldx3L5Q



https://youtu.be/Ajjc08-iPx8

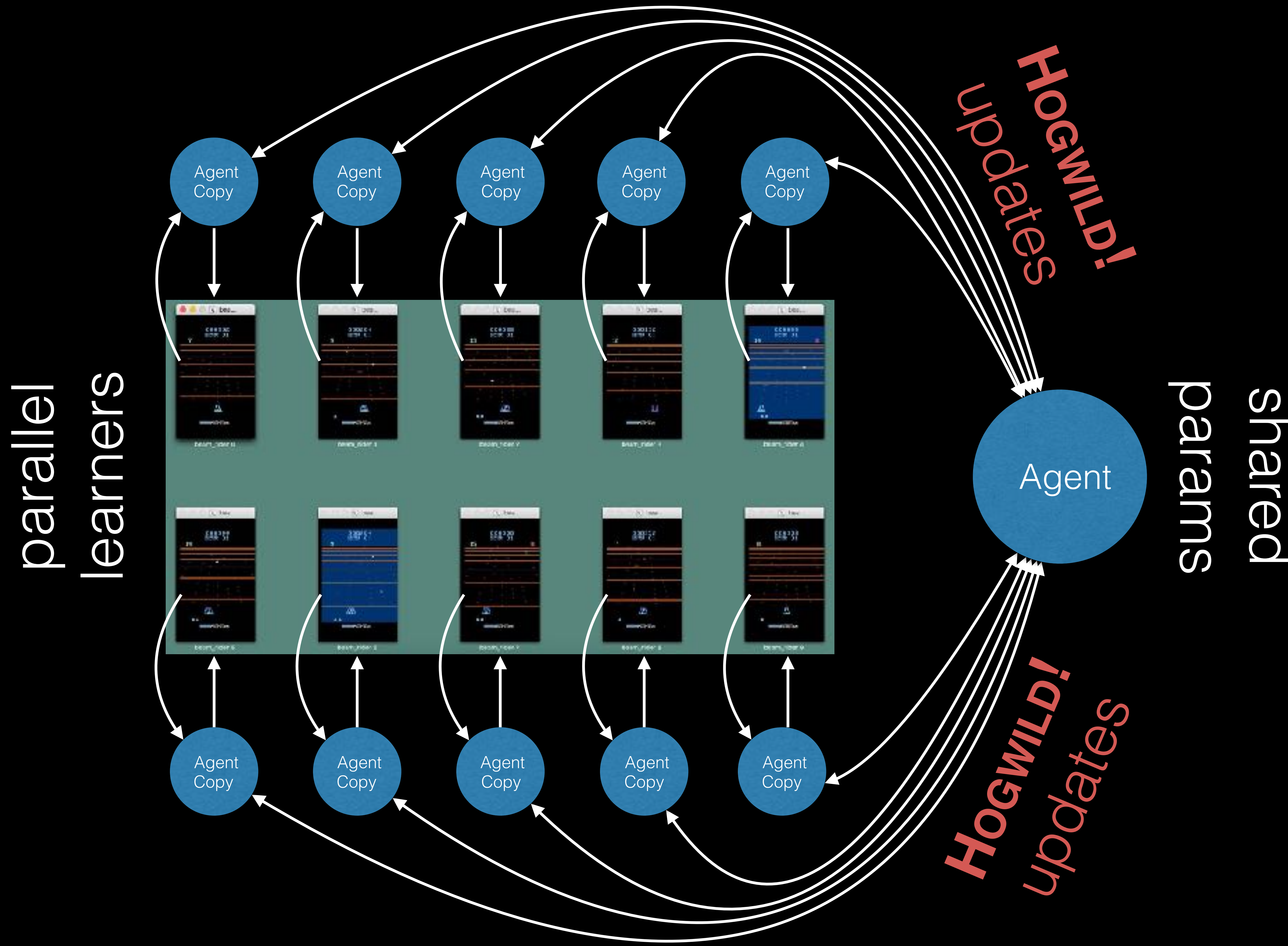
https://youtu.be/nMR5mjCFZCw

parallel agents

shared parameters

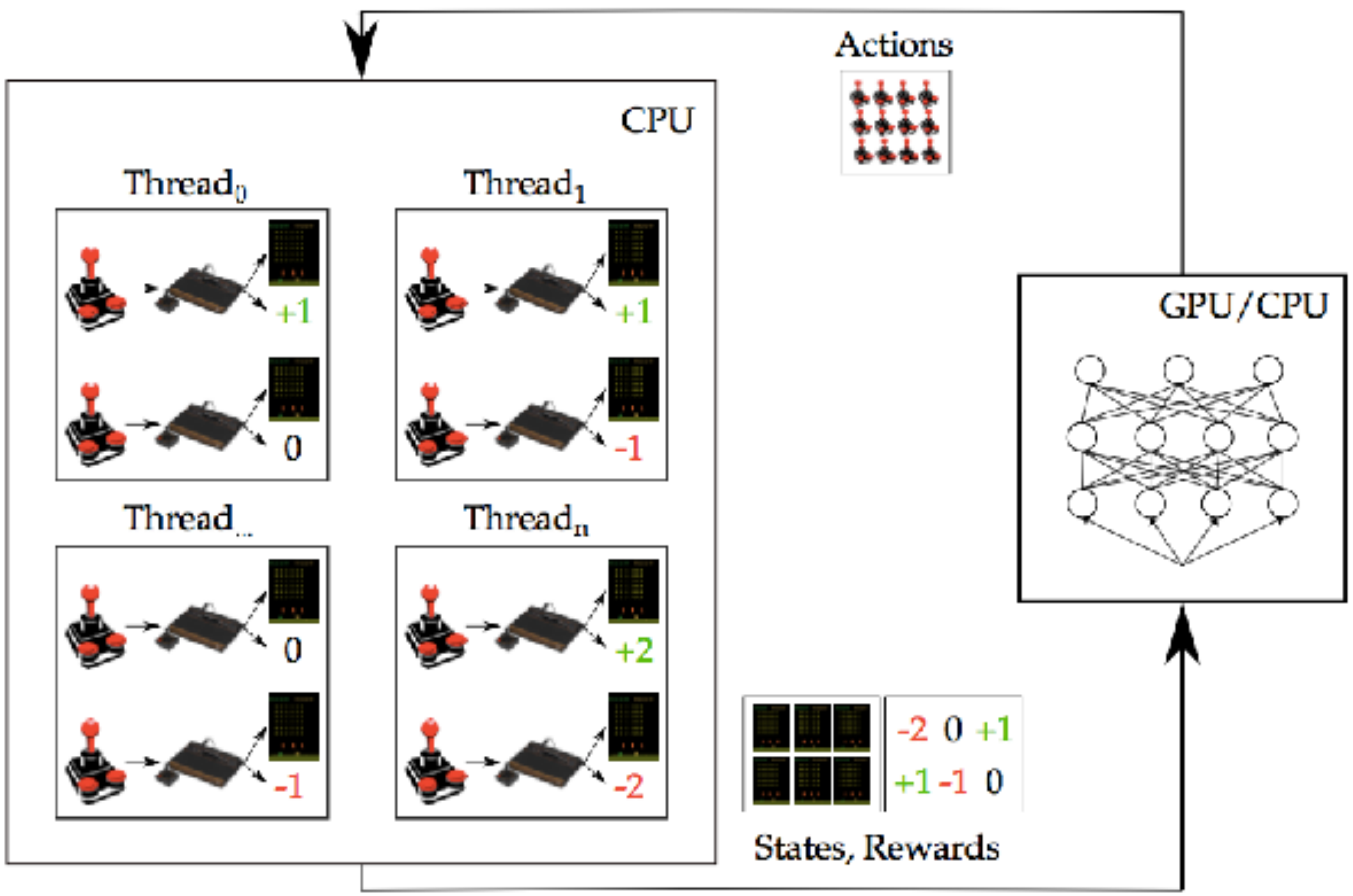
lock-free updates

Asynchronous Methods for Deep Reinforcement Learning, Mnih et. al., **ICML 2016**



https://github.com/traai/async-deep-rl

PAAC
(**P**arallel **A**dvantage **A**ctor-**C**ritic)



1 GPU/CPU

Reduced training time

SOTA performance

https://github.com/alfredvc/paac

Efficient Parallel Methods for Deep Reinforcement Learning,
A. V. Clemente, H. N. Castejón, and A. Chandra, **RLDM 2017**

Alfredo Clemente

code for you to play with...

Rich Sutton's book examples **(exhaustive, must try!)**:
https://github.com/ShangtongZhang/reinforcement-learning-an-introduction

Telenor's implementation of **asynchronous parallel methods:**
https://github.com/traai/async-deep-rl

Alfredo's **faster parallel methods:**
https://github.com/alfredvc/paac

++...

OpenAI Gym BETA

DeepMind Lab

Inspired to code/apply RL?

Next lecture:
Applications (and some **hacking**)
November 21, 2017

https://join.slack.com/t/deep-rl-tutorial/signup