

Reinforcement Learning

lectures by Florian Marquardt,
Max Planck Institute for the Science of Light
Warsaw summer school 2021

Reinforcement learning: basic setting
Examples (preview)

Some preliminaries
delayed rewards
model-based vs model-free
stochastic environments

Policy gradient
Q-learning
Actor-critic (briefly)

Compact lecture notes:

Les Houches 2019 summer school
lecture notes on machine learning for
quantum devices

SciPost Physics (21) 10.21468 (2021)

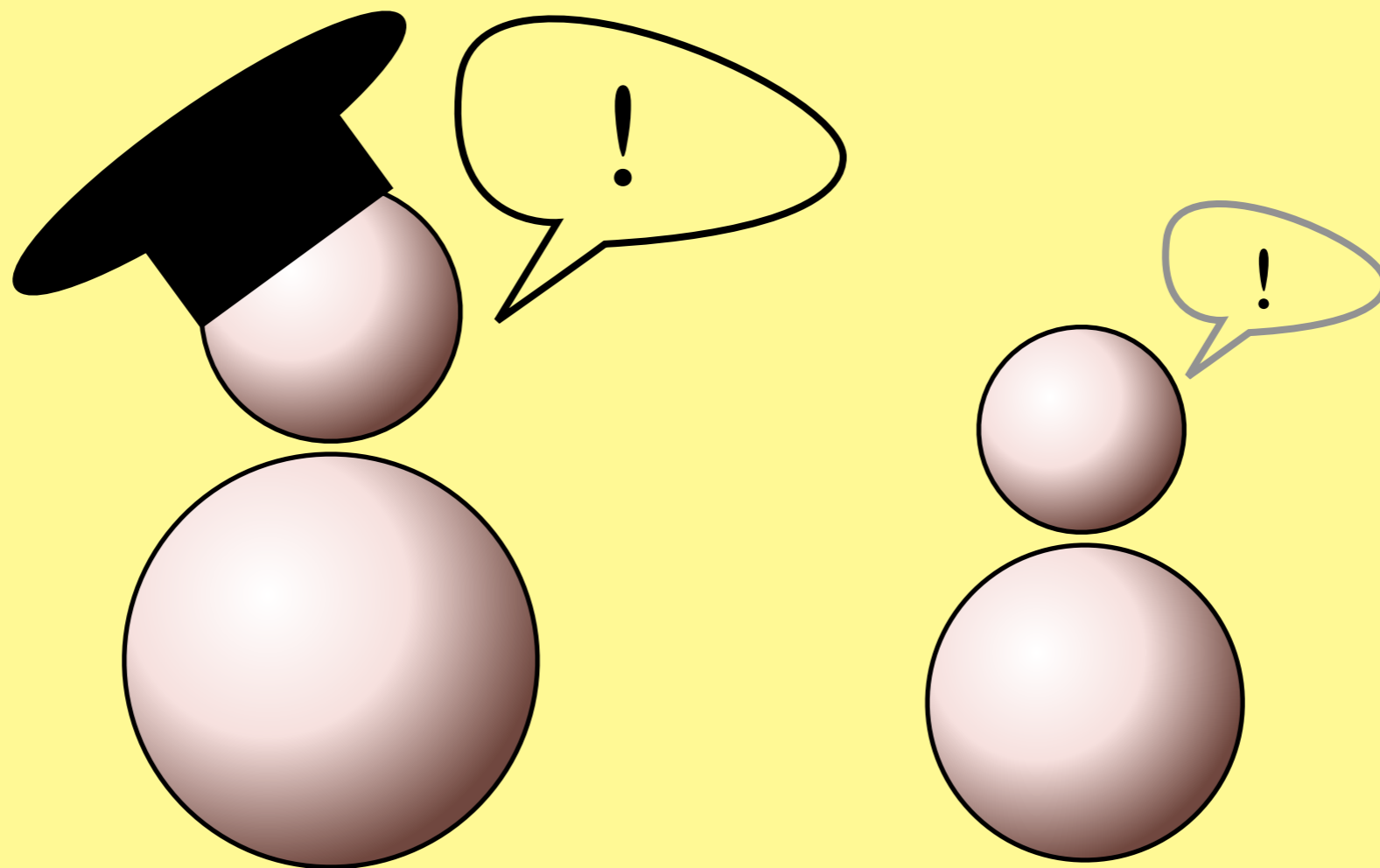
Extended lecture series on ML for
physicists and code examples:

pad.gwdg.de/s/

Machine_Learning_For_Physicists_2021

Supervised learning vs reinforcement learning

“Supervised learning” (most neural network applications)

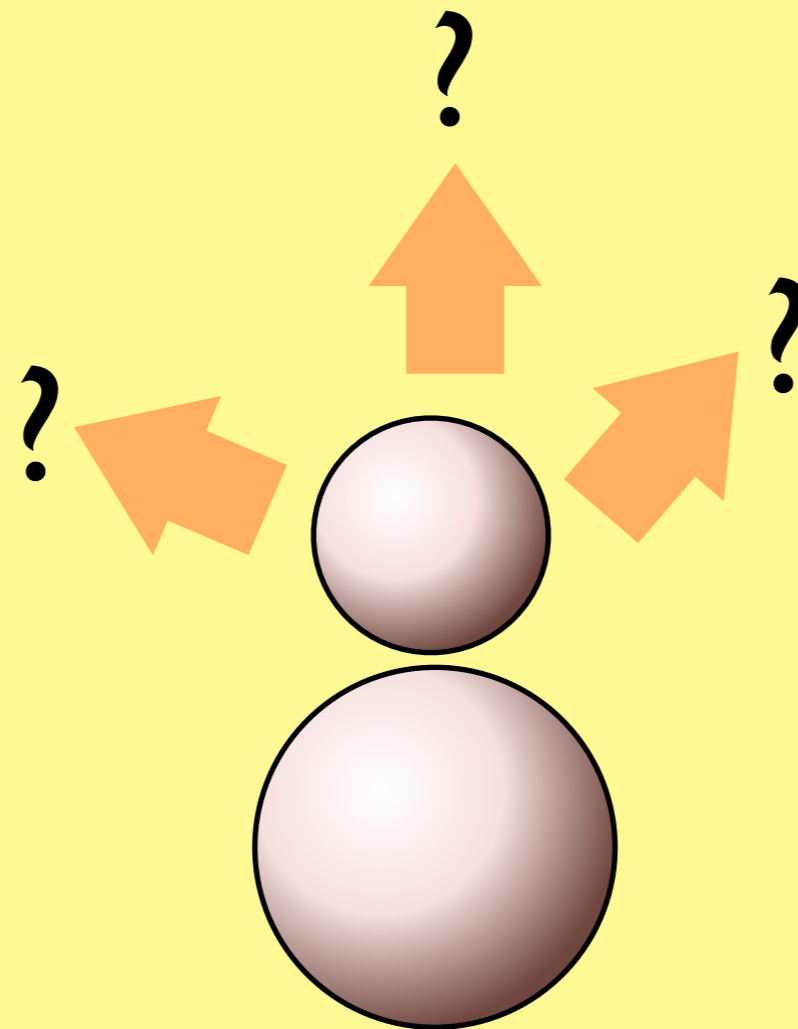


teacher
(smart)

student
(imitates teacher)

final level limited by teacher

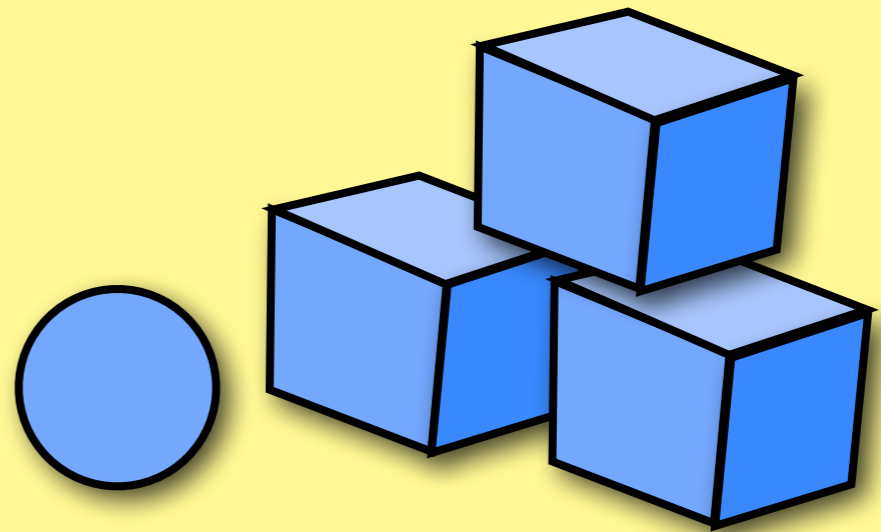
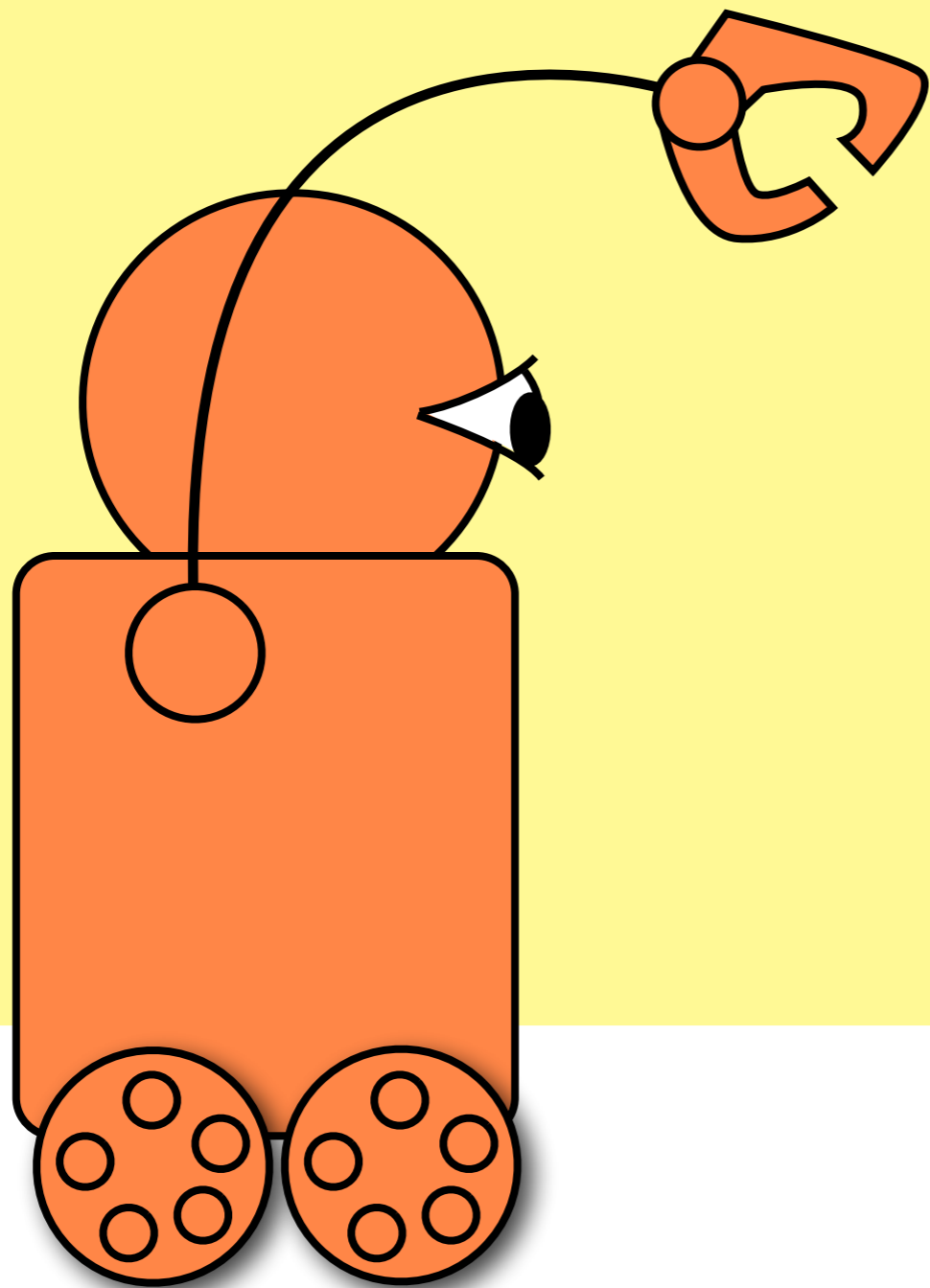
“Reinforcement learning”



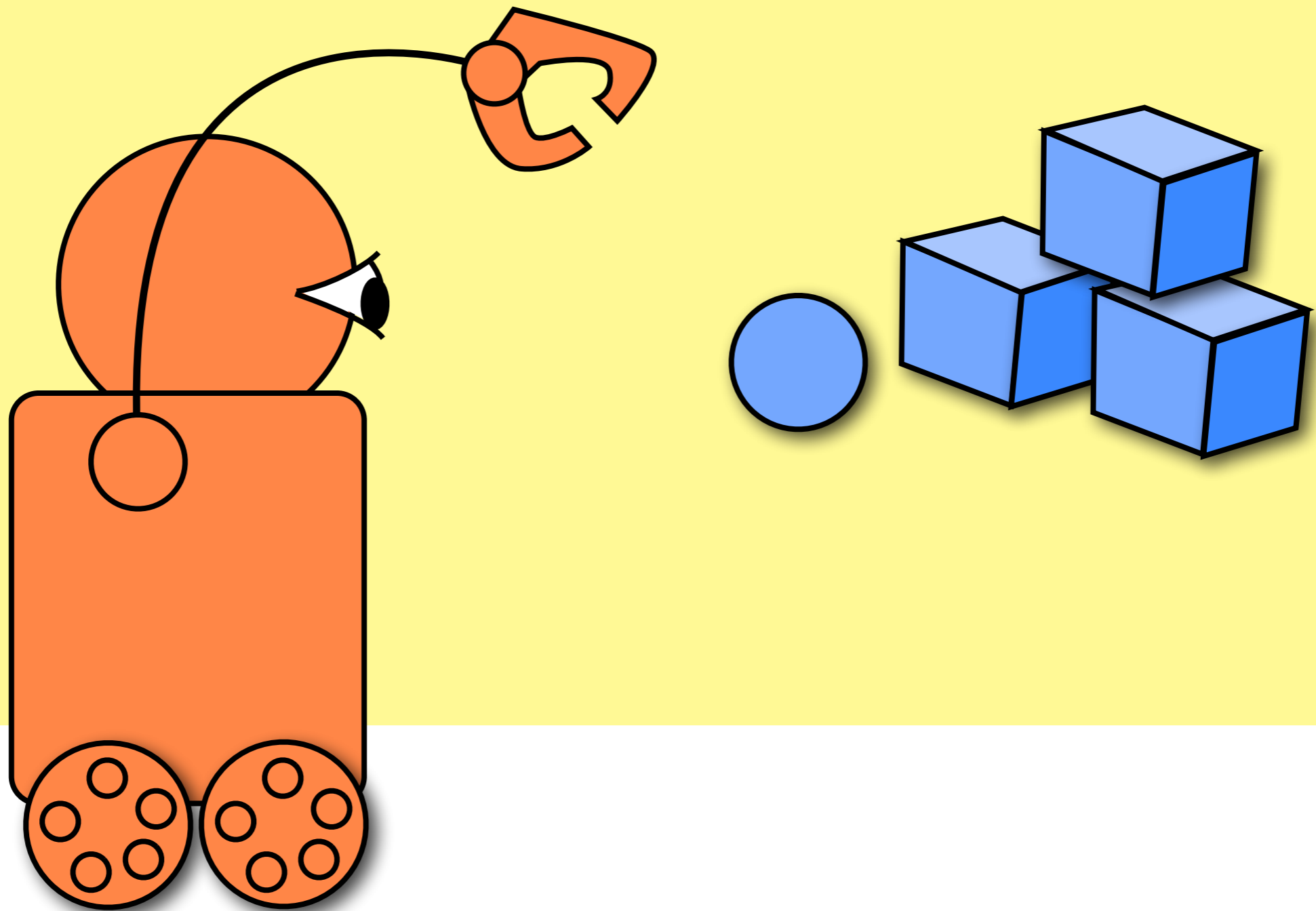
student/scientist
(tries out things)

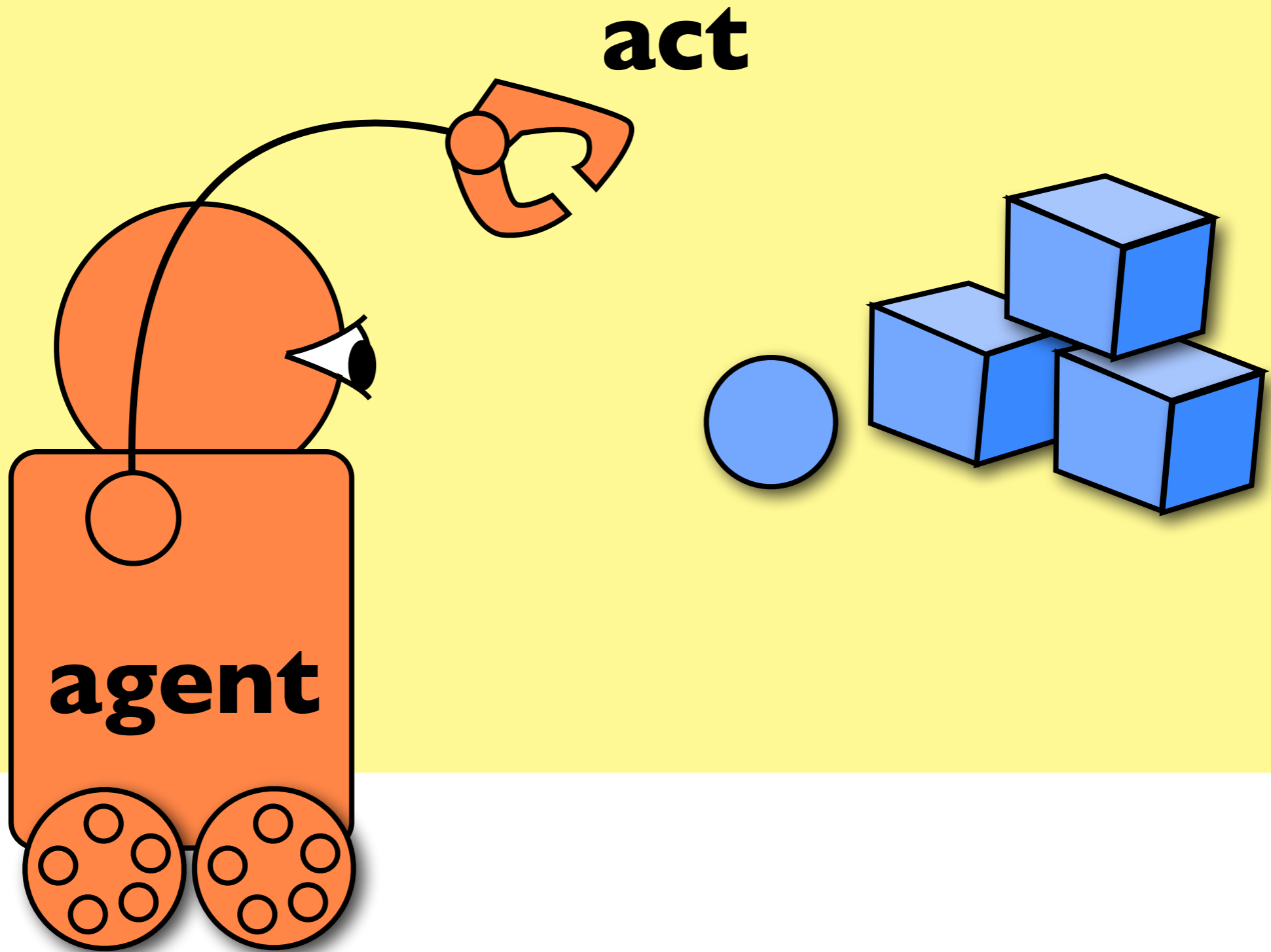
final level: unlimited (?)

Reinforcement learning: The basic setting



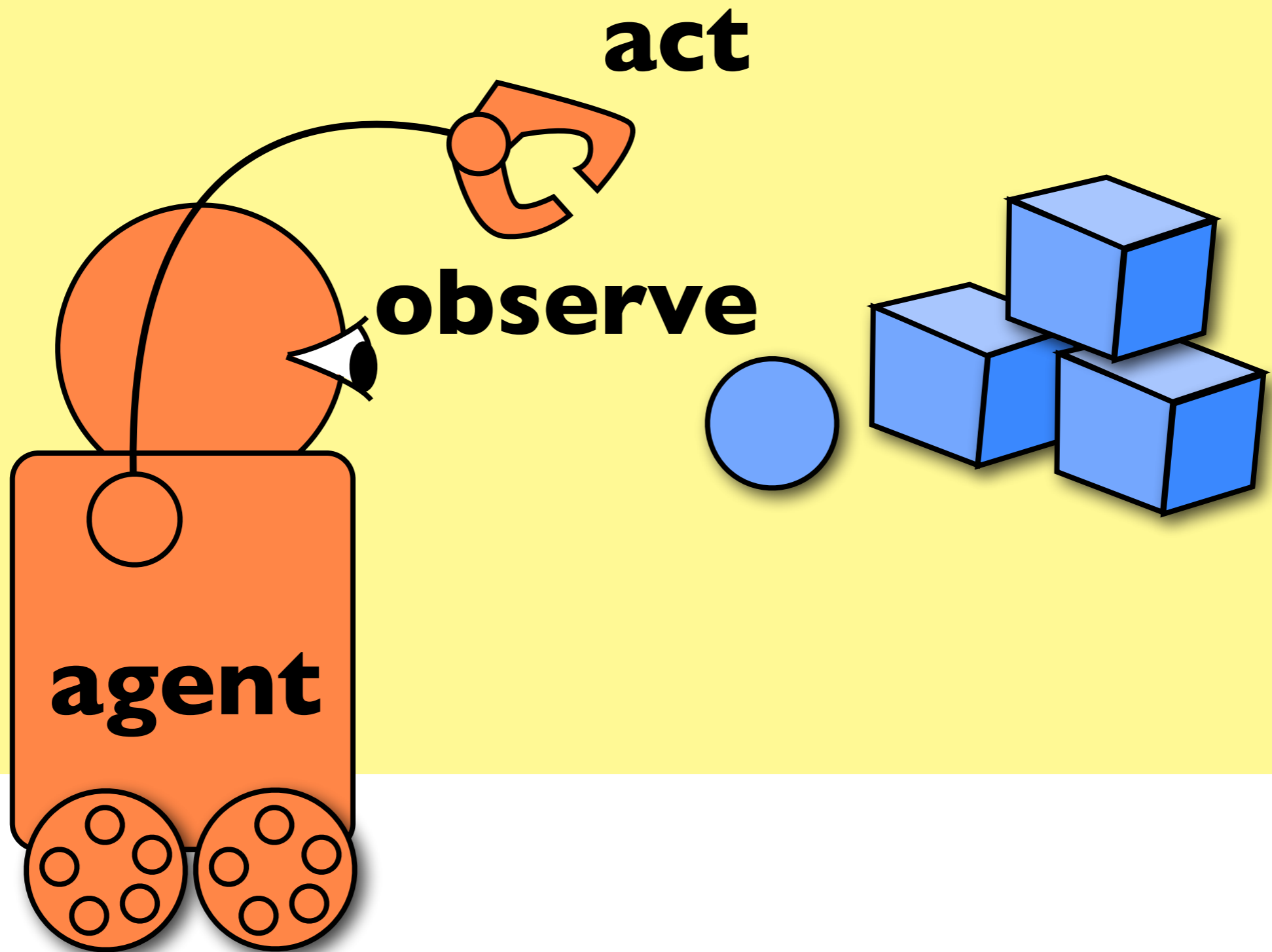
act





act

agent



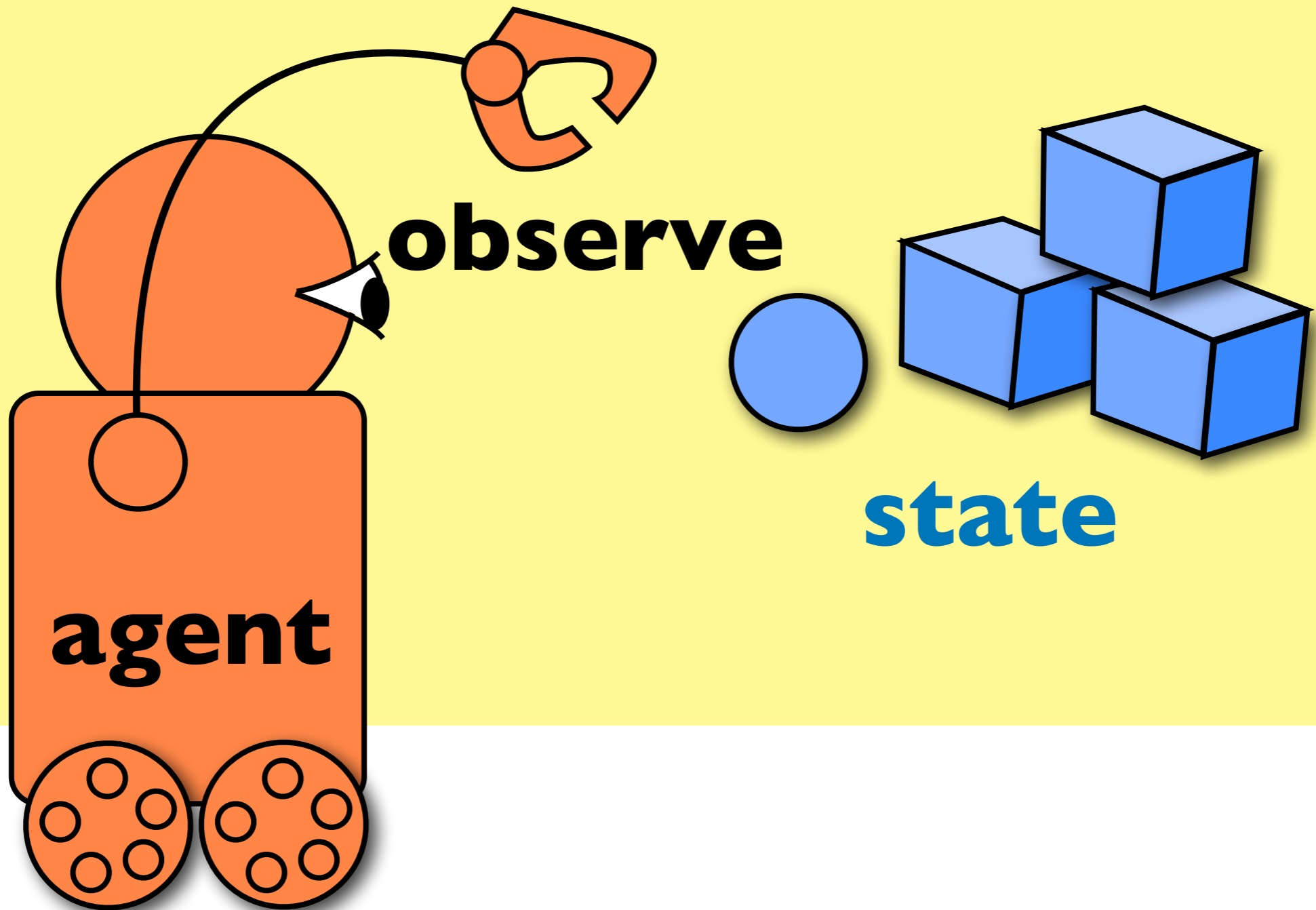
environment

act

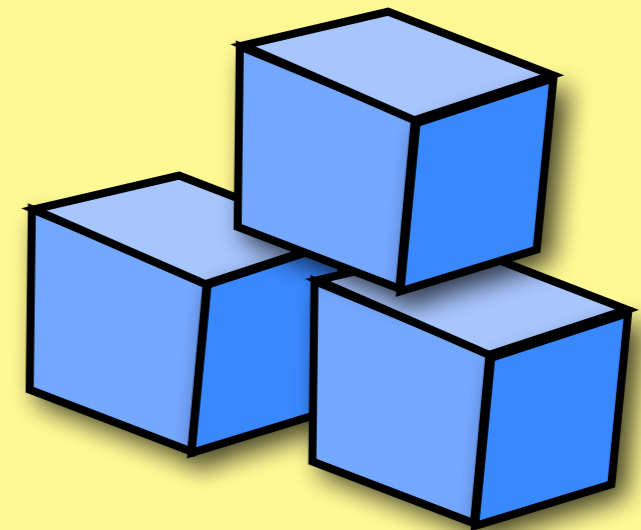
observe

state

agent

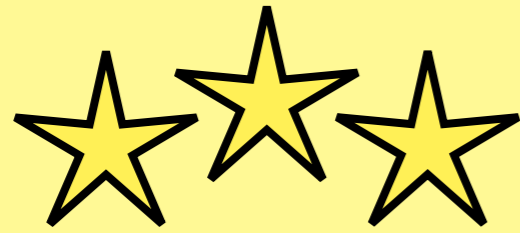


environment



state

reward

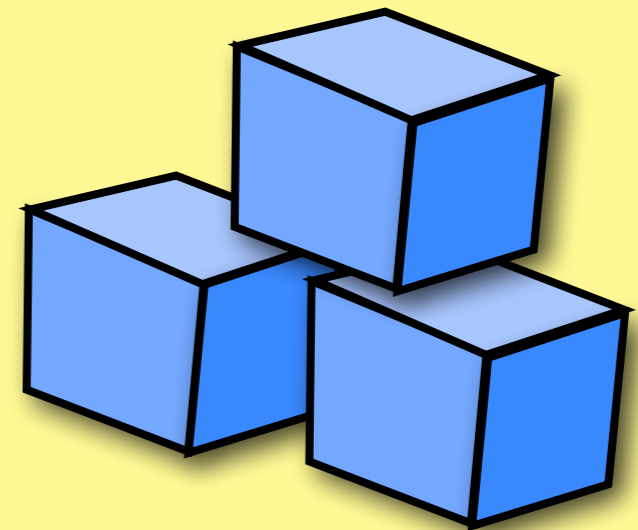
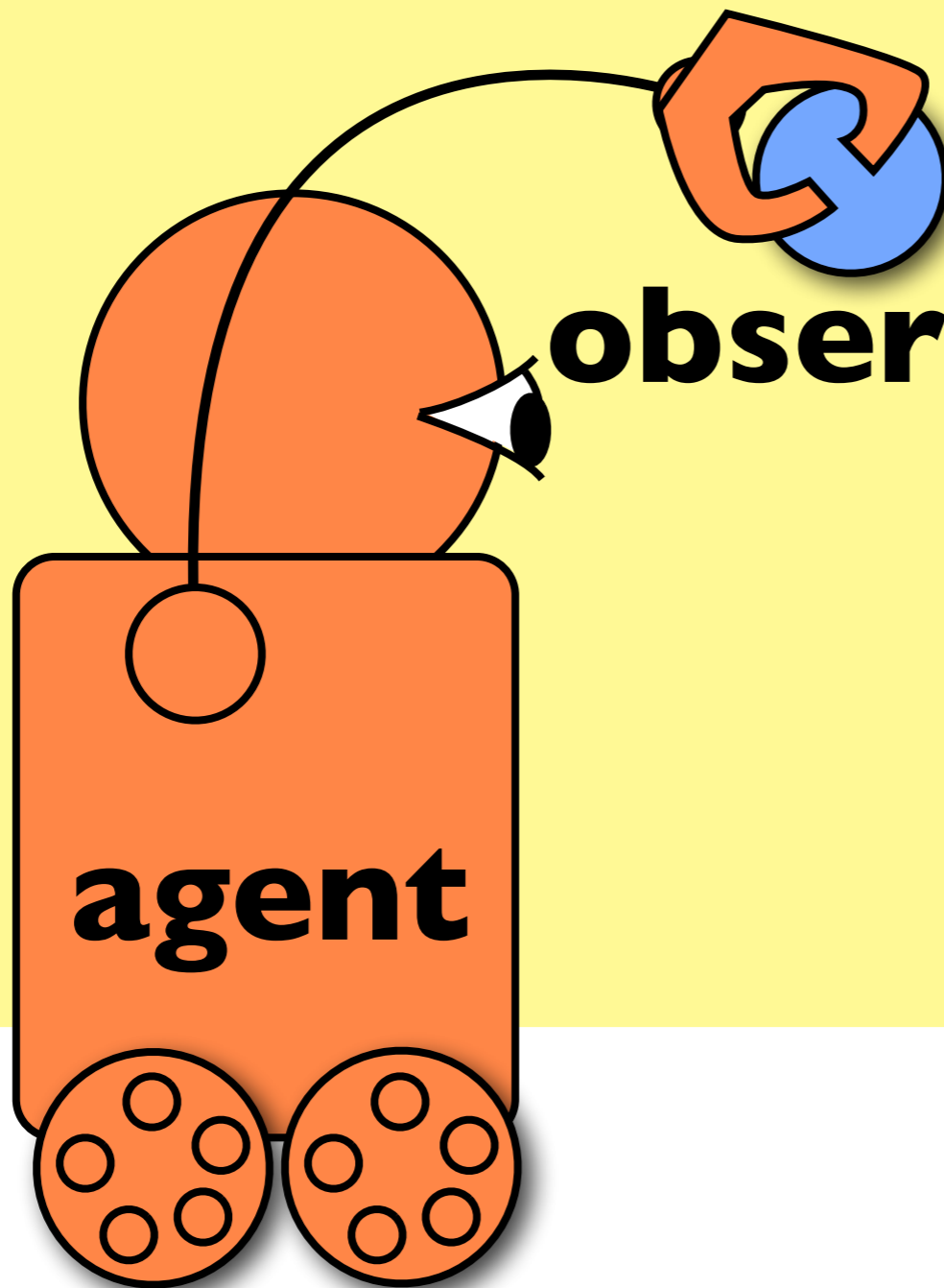


environment

act

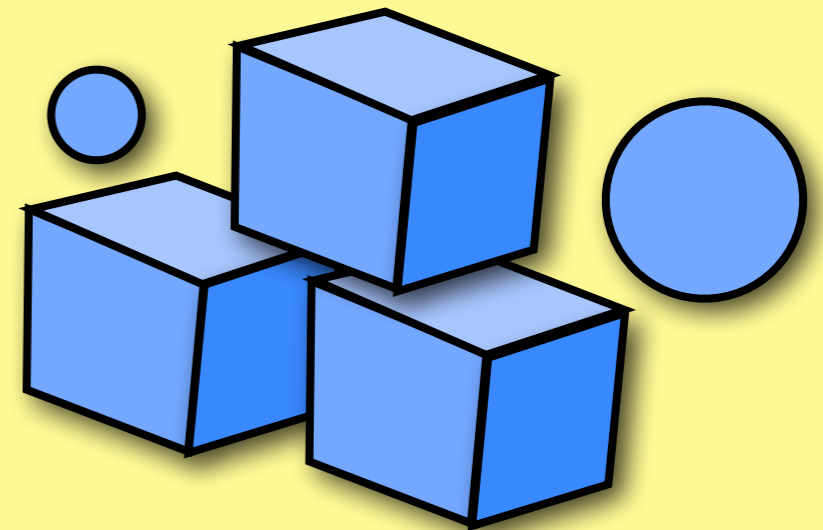
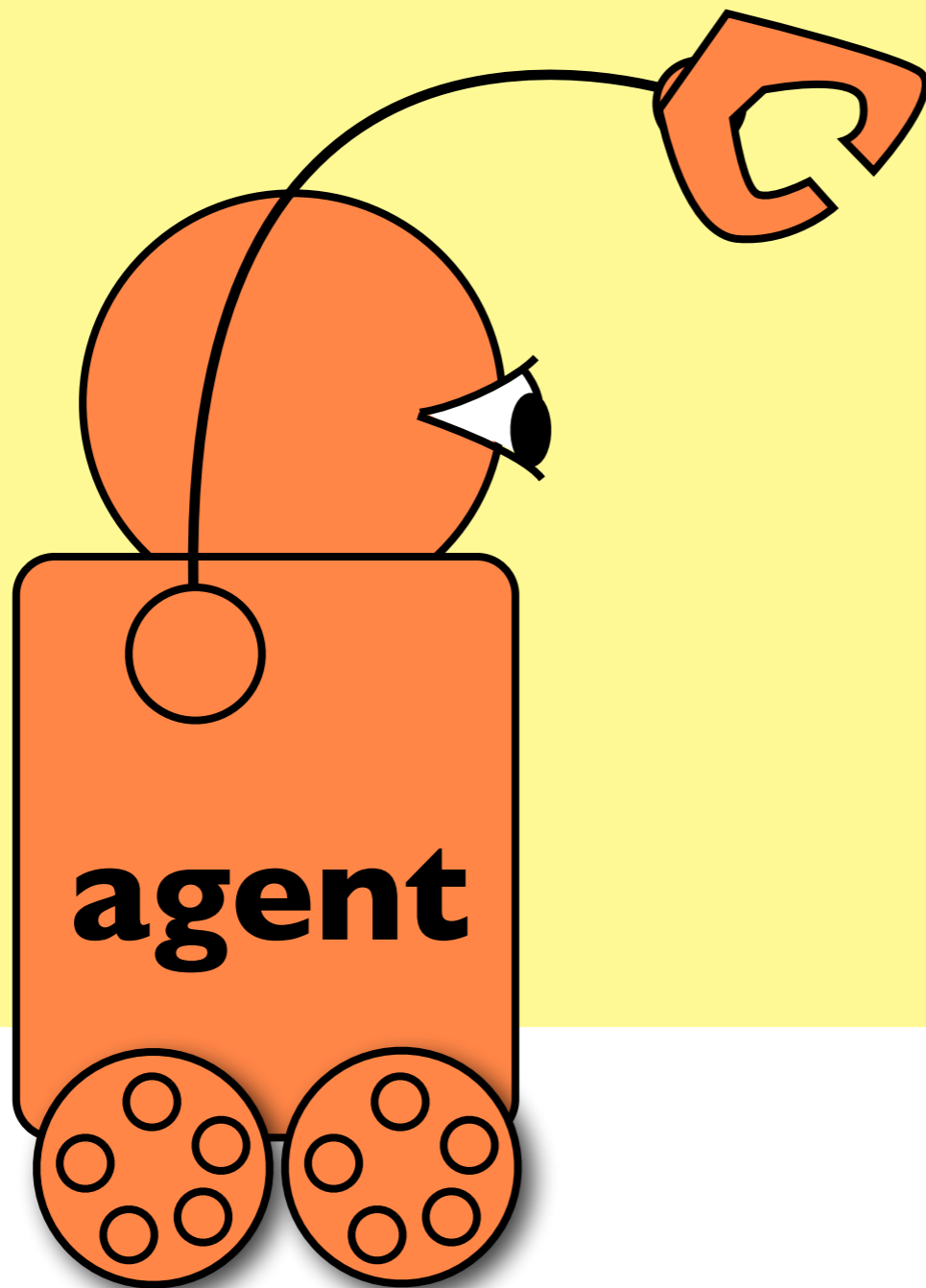
observe

state



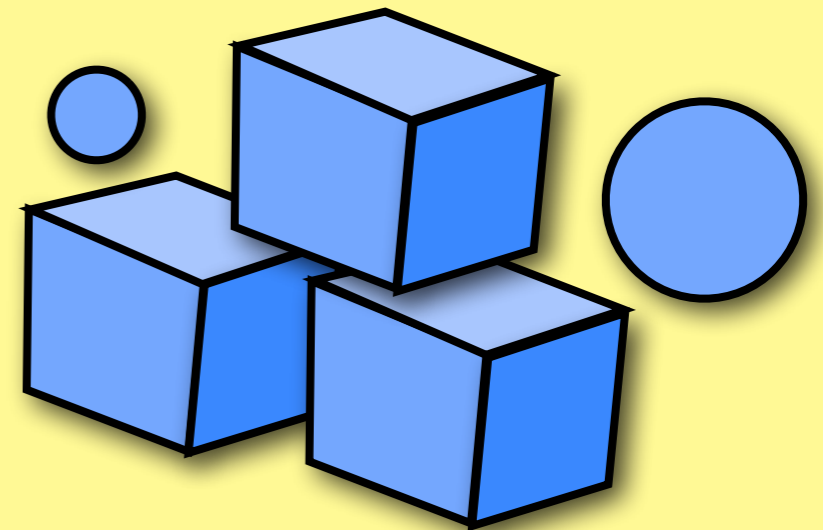
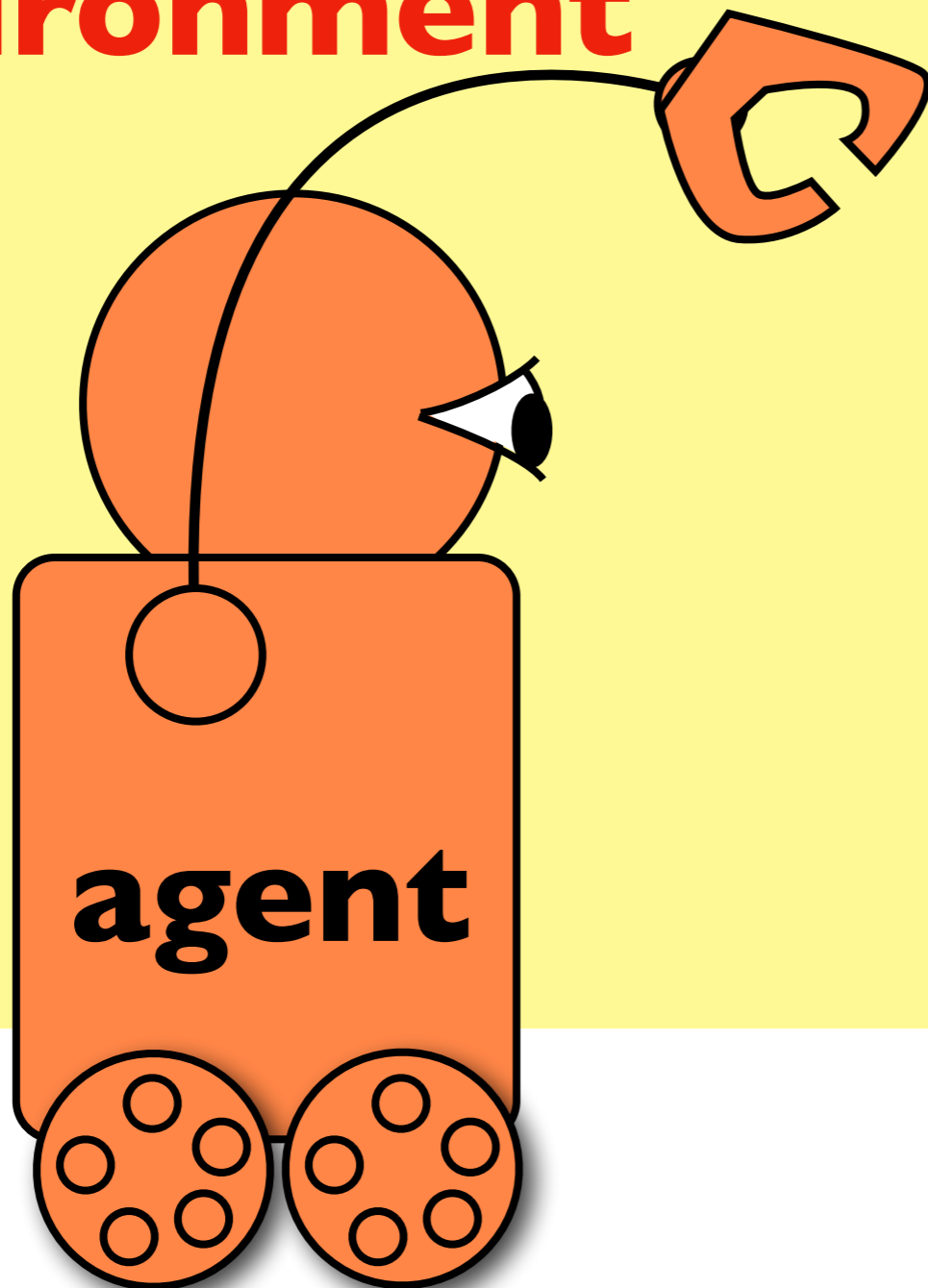
reward
...may be delayed!

environment

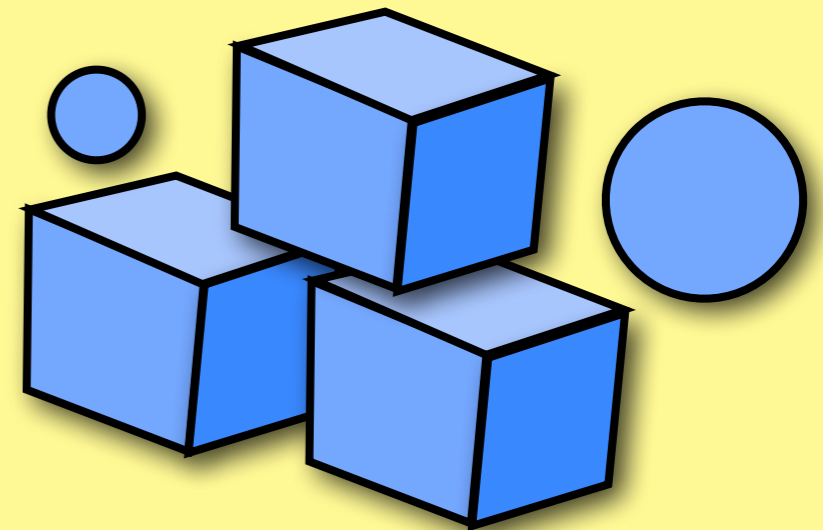
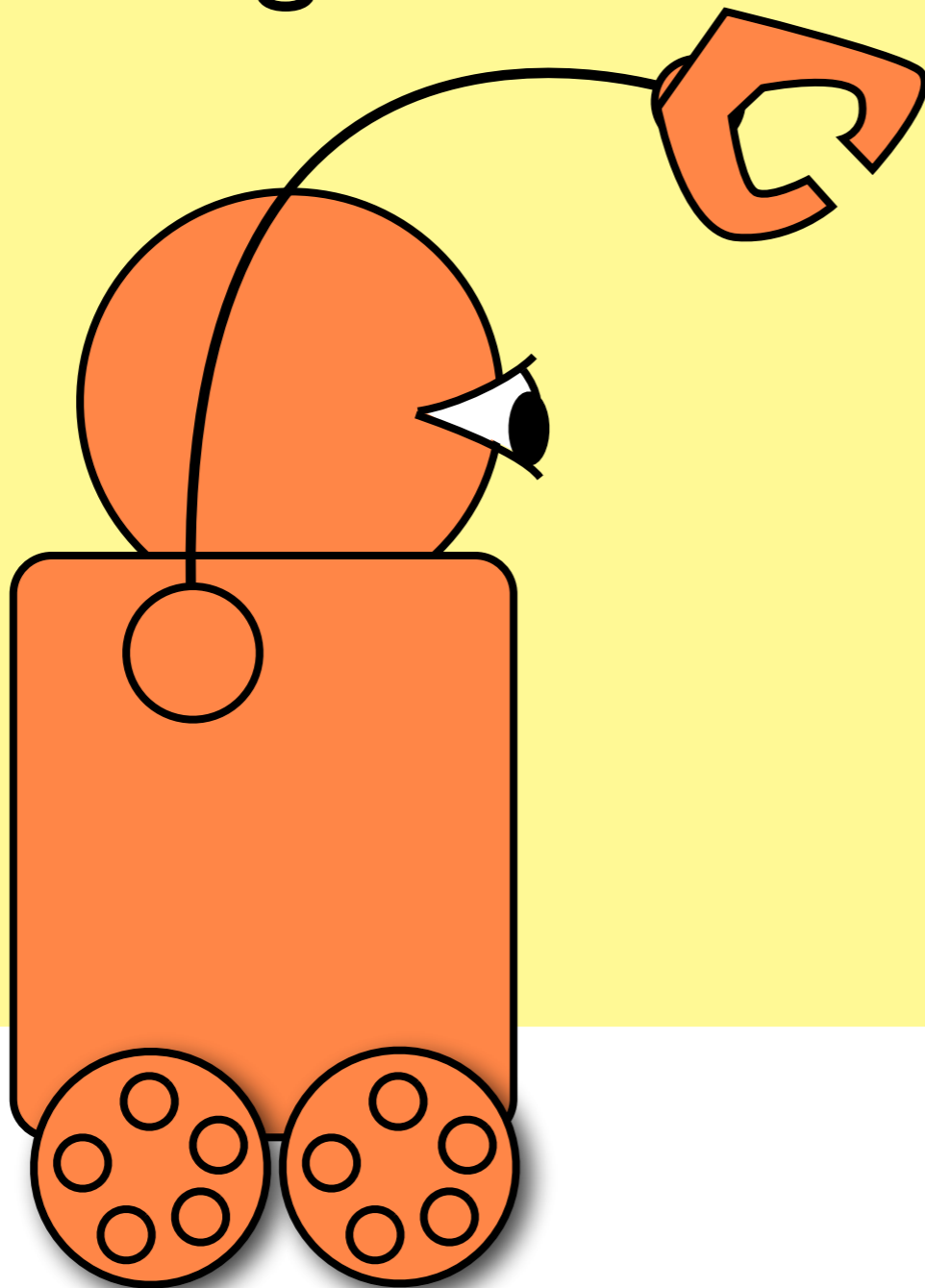


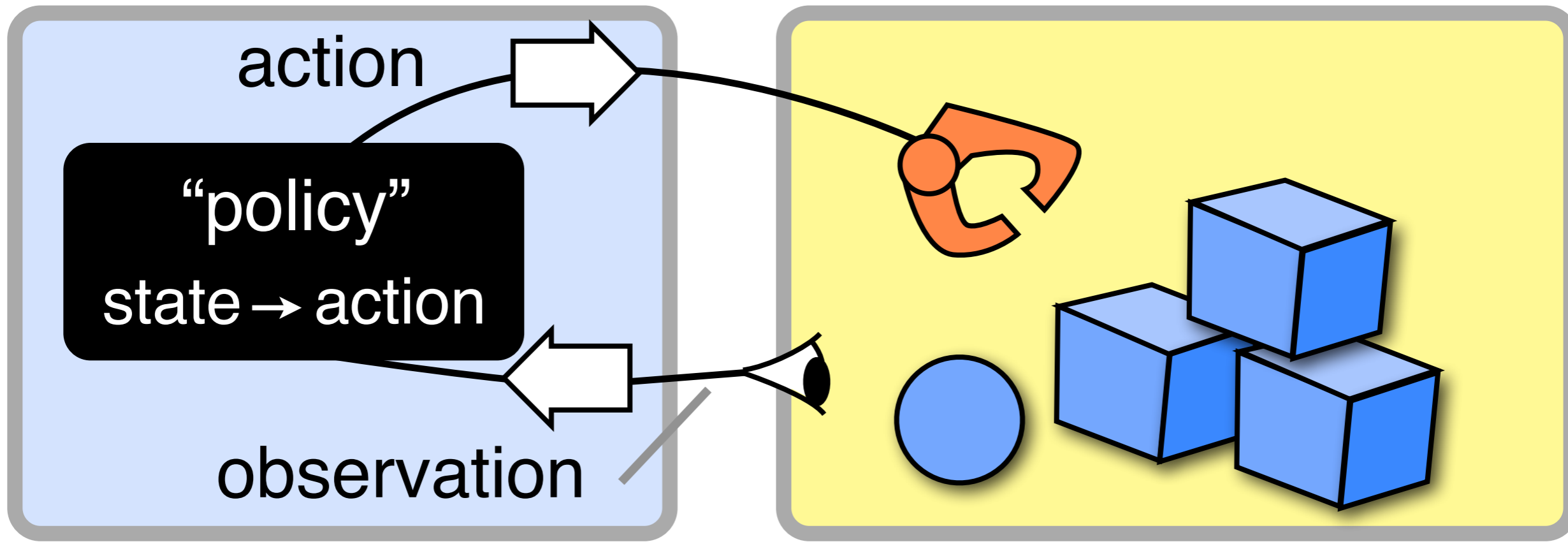
environment

reward
...may be delayed!
...depends on behaviour of
environment



Reinforcement learning:
Discover strategies ('policies') =
actions in response to observations,
maximizing rewards



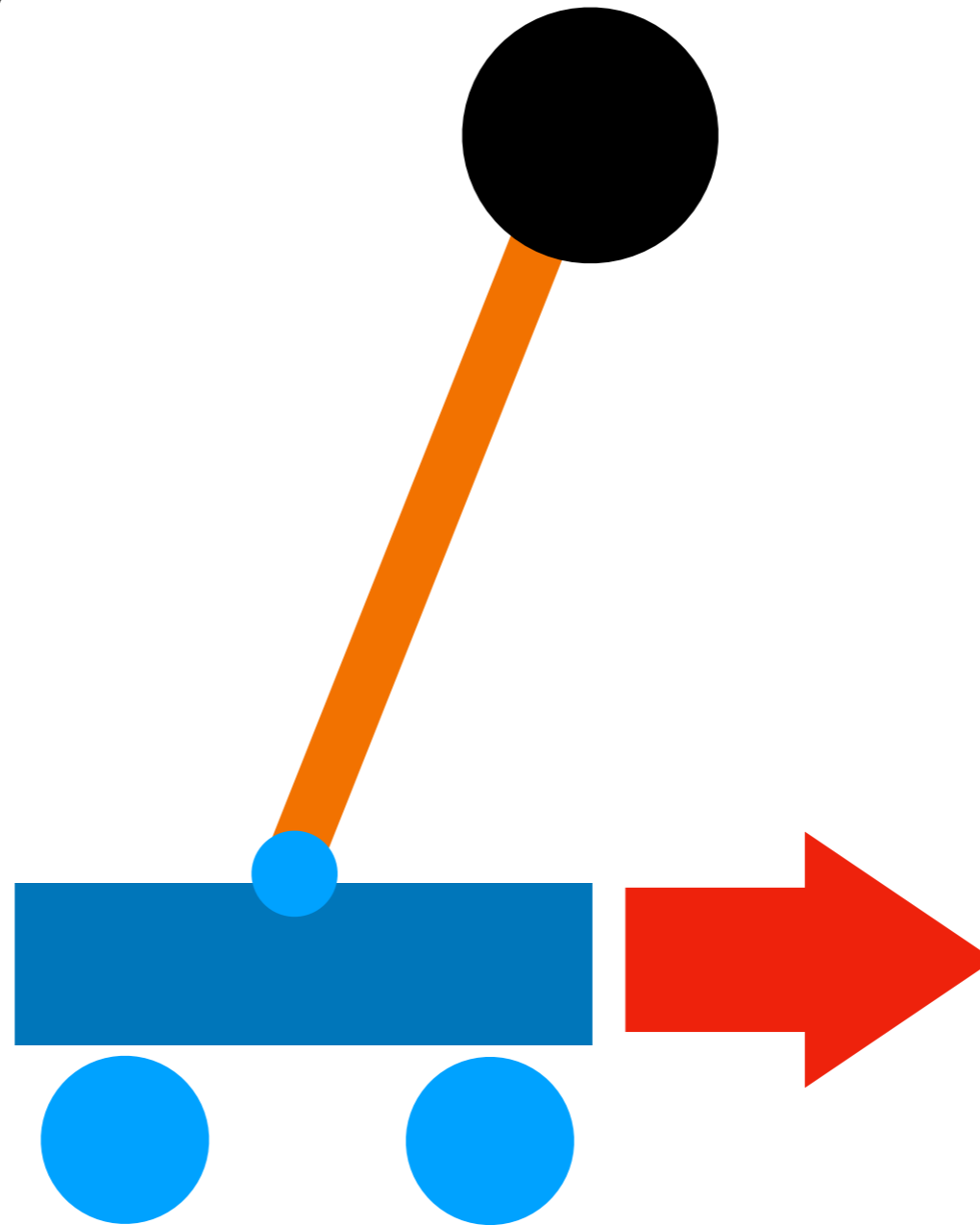


RL-agent

RL-environment

Examples (preview)

Cartpole (balancing)

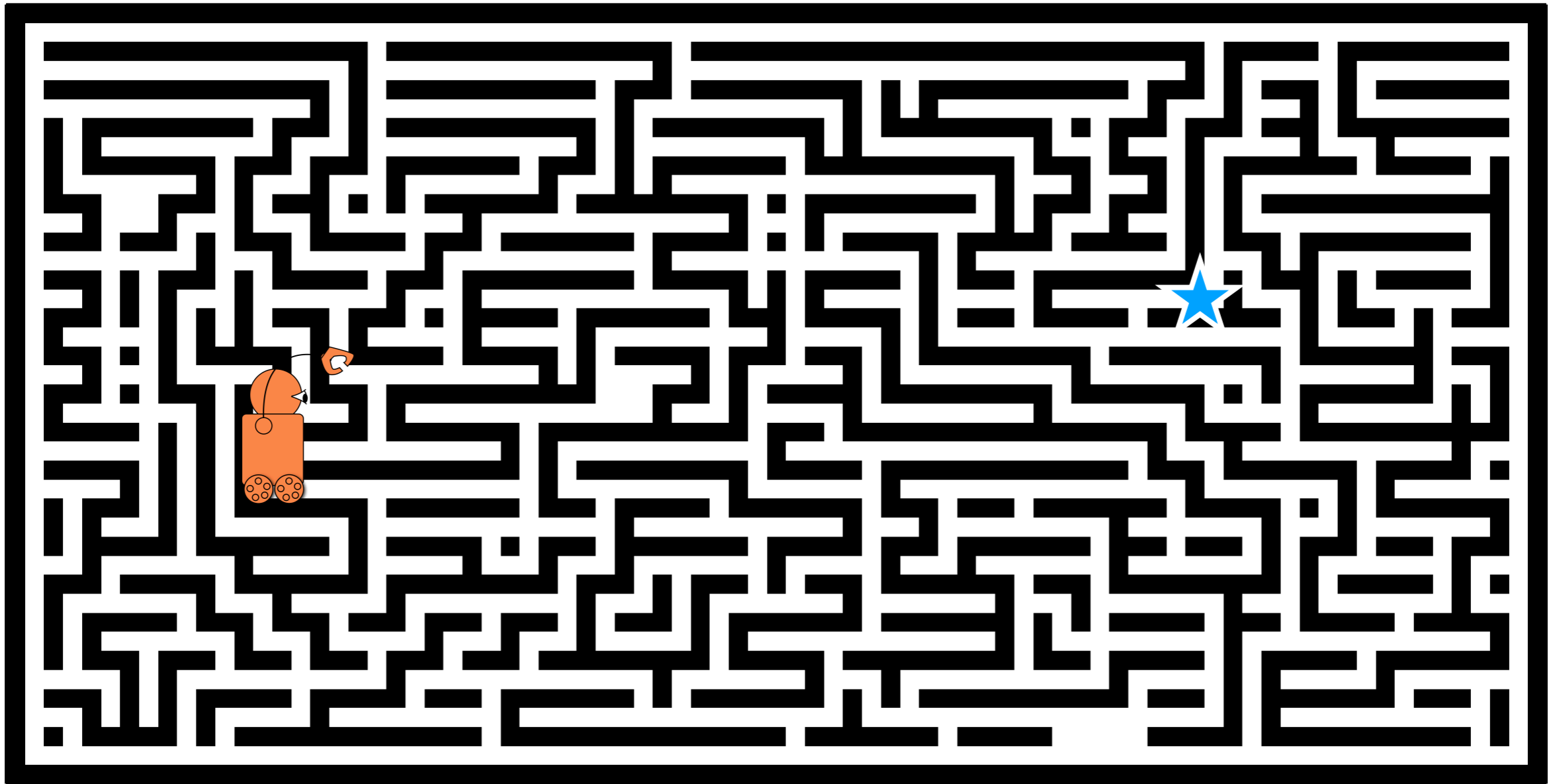


state = angle and velocity of pendulum

action = acceleration of cart

reward = height of mass on pendulum

Solving a fixed maze with fixed target, starting from random location

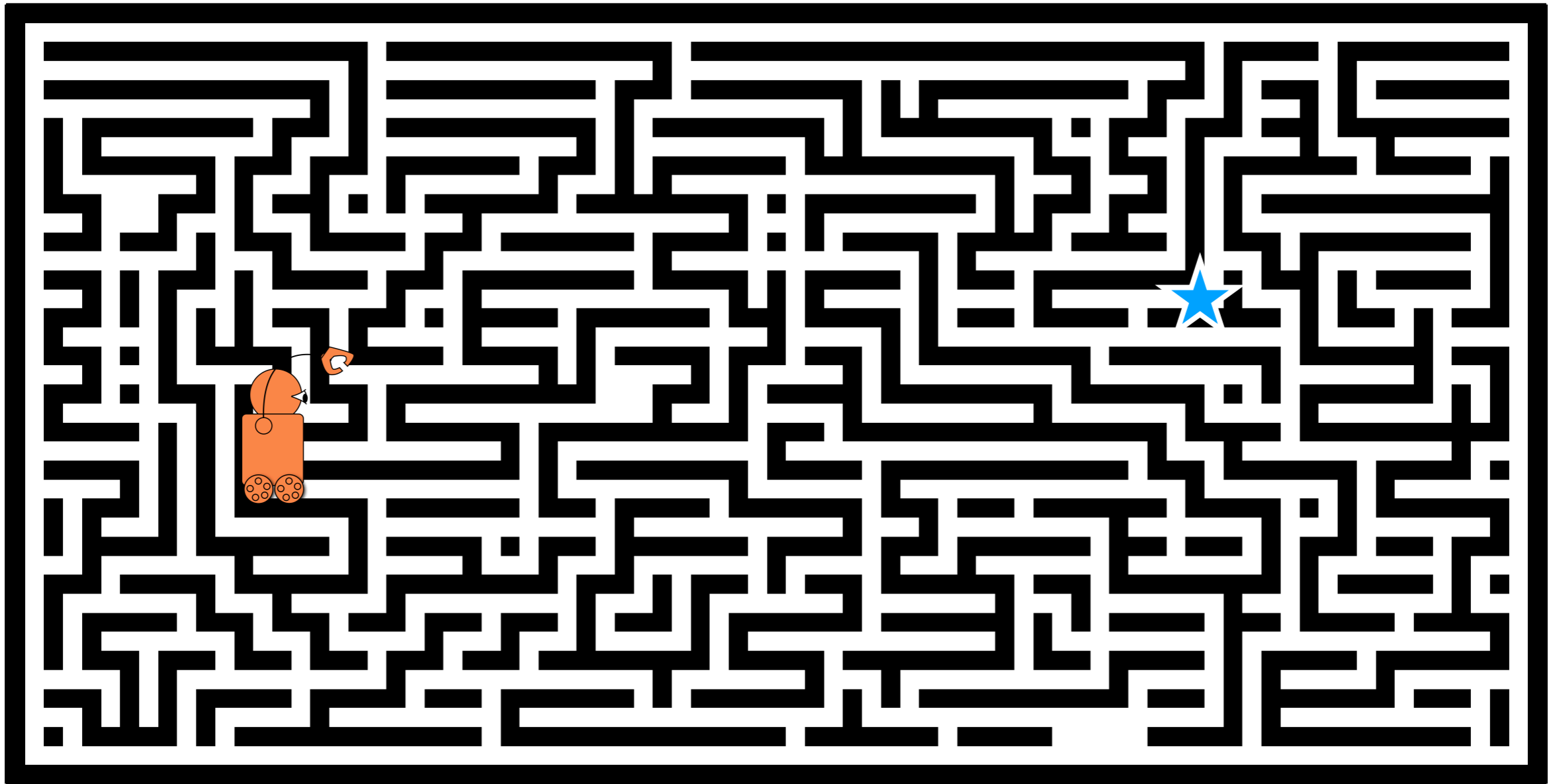


state = location of robot

action = move

reward depends on number of time steps until target

Solving a **random** maze, starting from random location

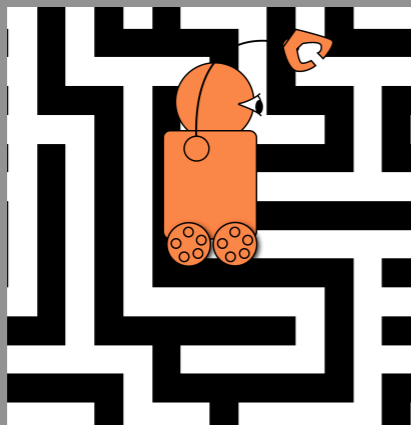


state = image of whole maze, including robot & target

action = move

reward depends on number of time steps until target

Solving a fixed or random maze, observing only close-by surroundings



observed state = image of surroundings

action = move

agent needs memory for good strategy!

Playing video games



observed state = screen image
action = move player spaceship
reward = high-score

Playing board games



observed state = board

action = make allowed game move

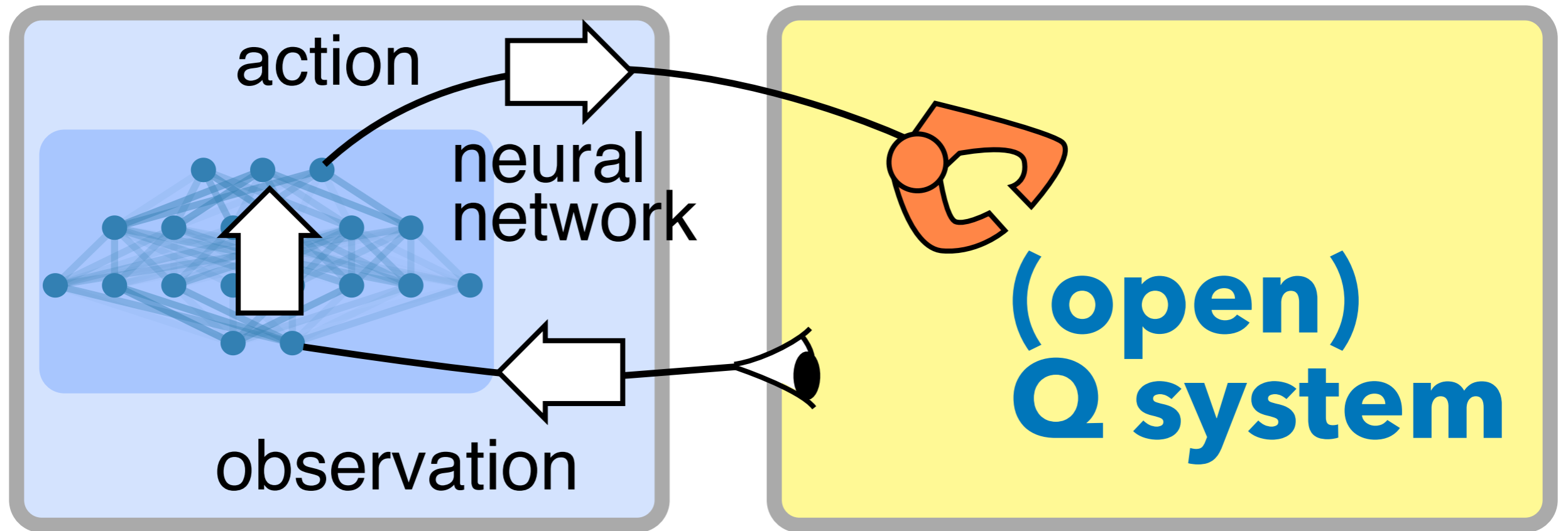
reward = 1,0,-1 at end, depending on win/draw/lose

environment includes opponent

Feedback-based quantum control

control

**quantum
feedback!**

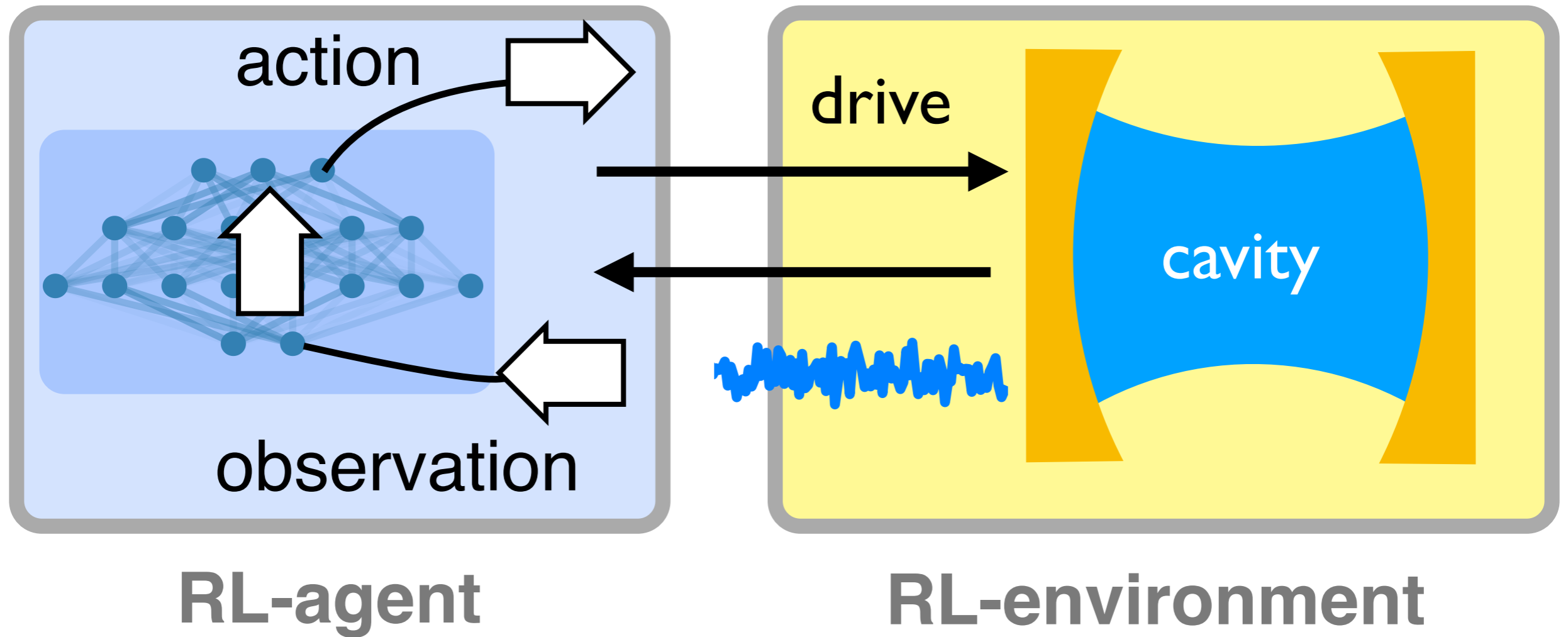


RL-agent

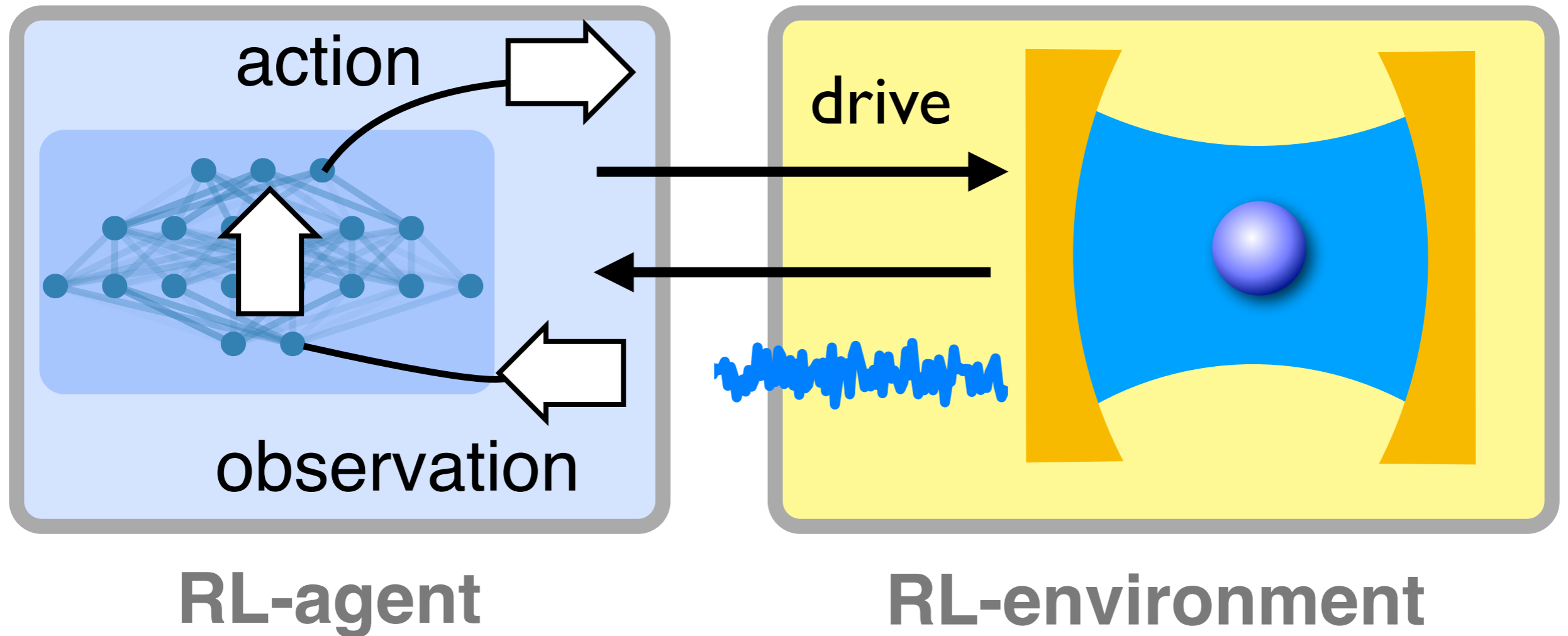
RL-environment

measurements

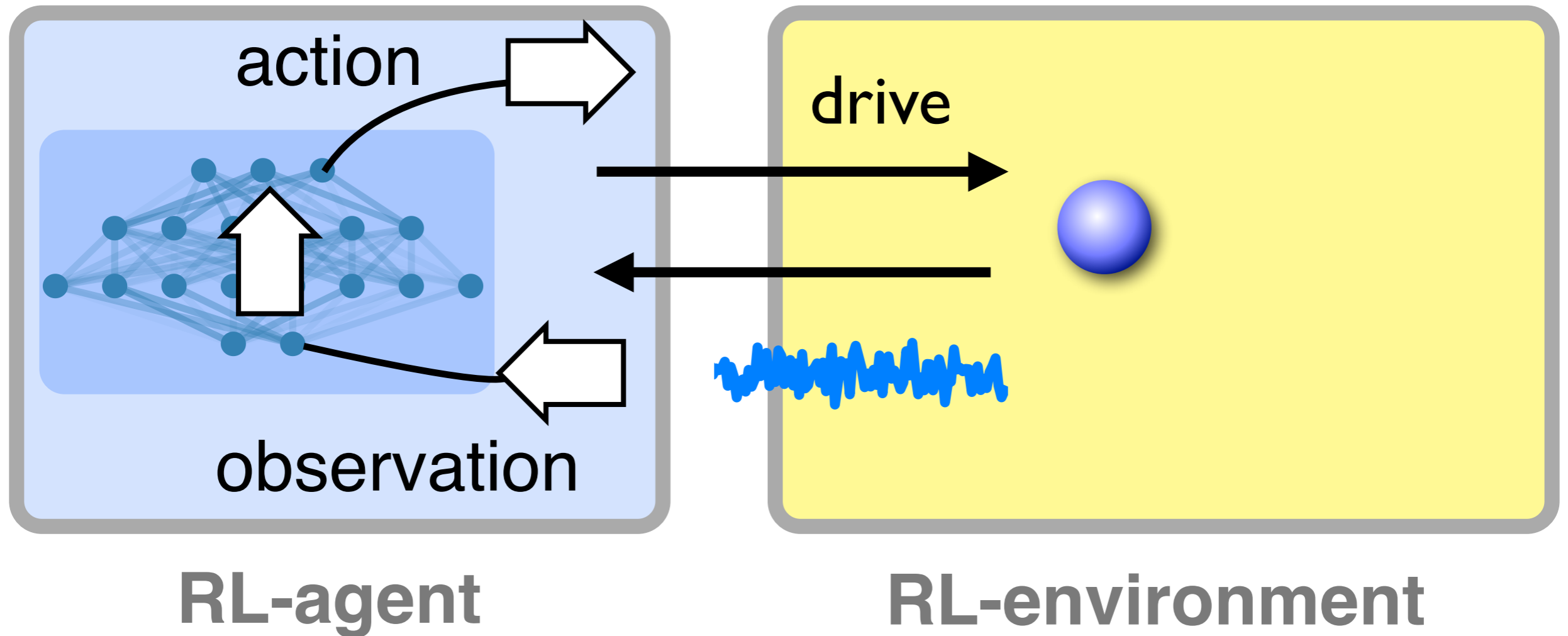
Feedback-based quantum control



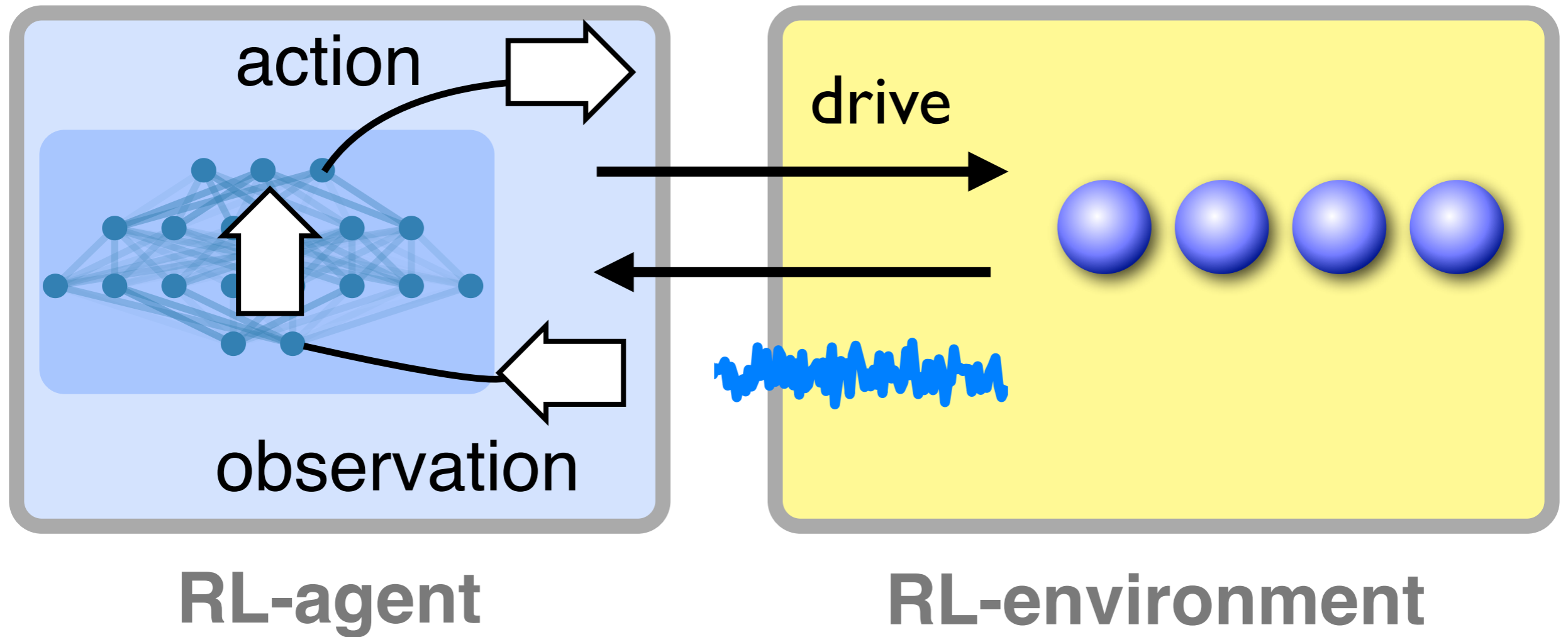
Feedback-based quantum control



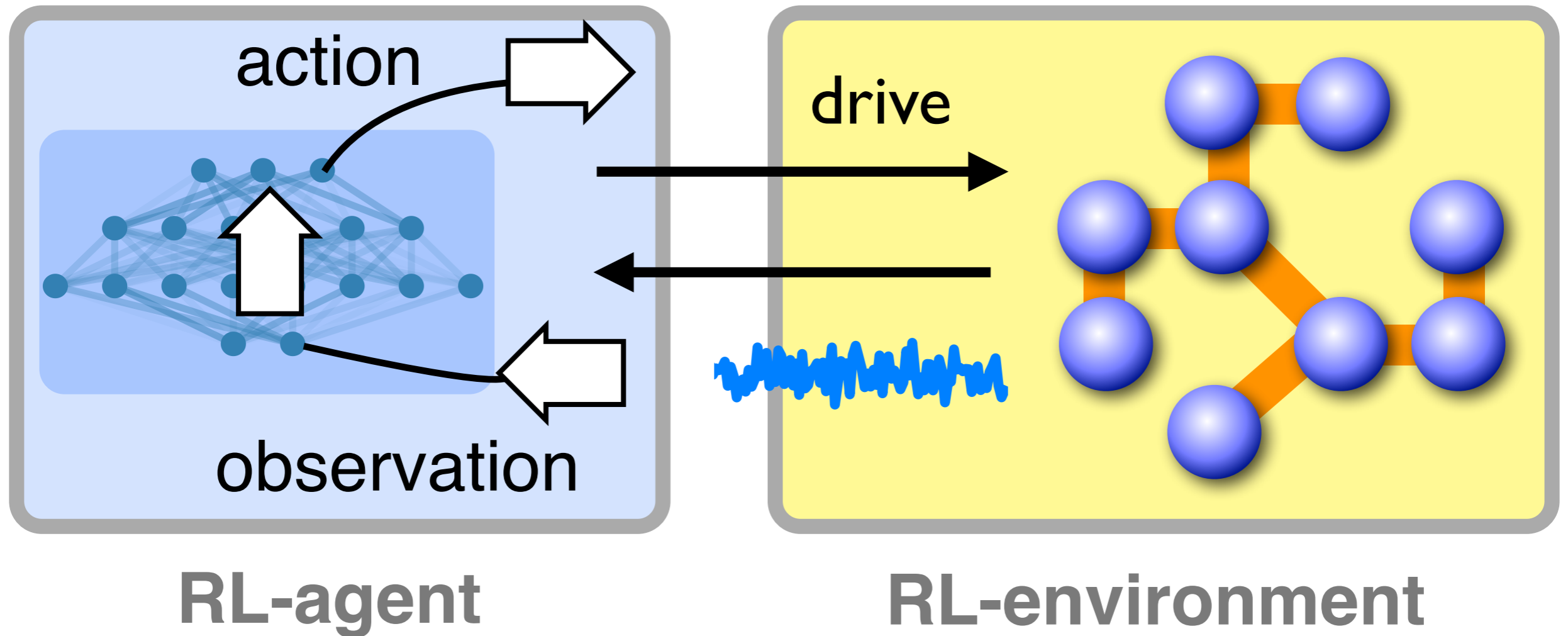
Feedback-based quantum control



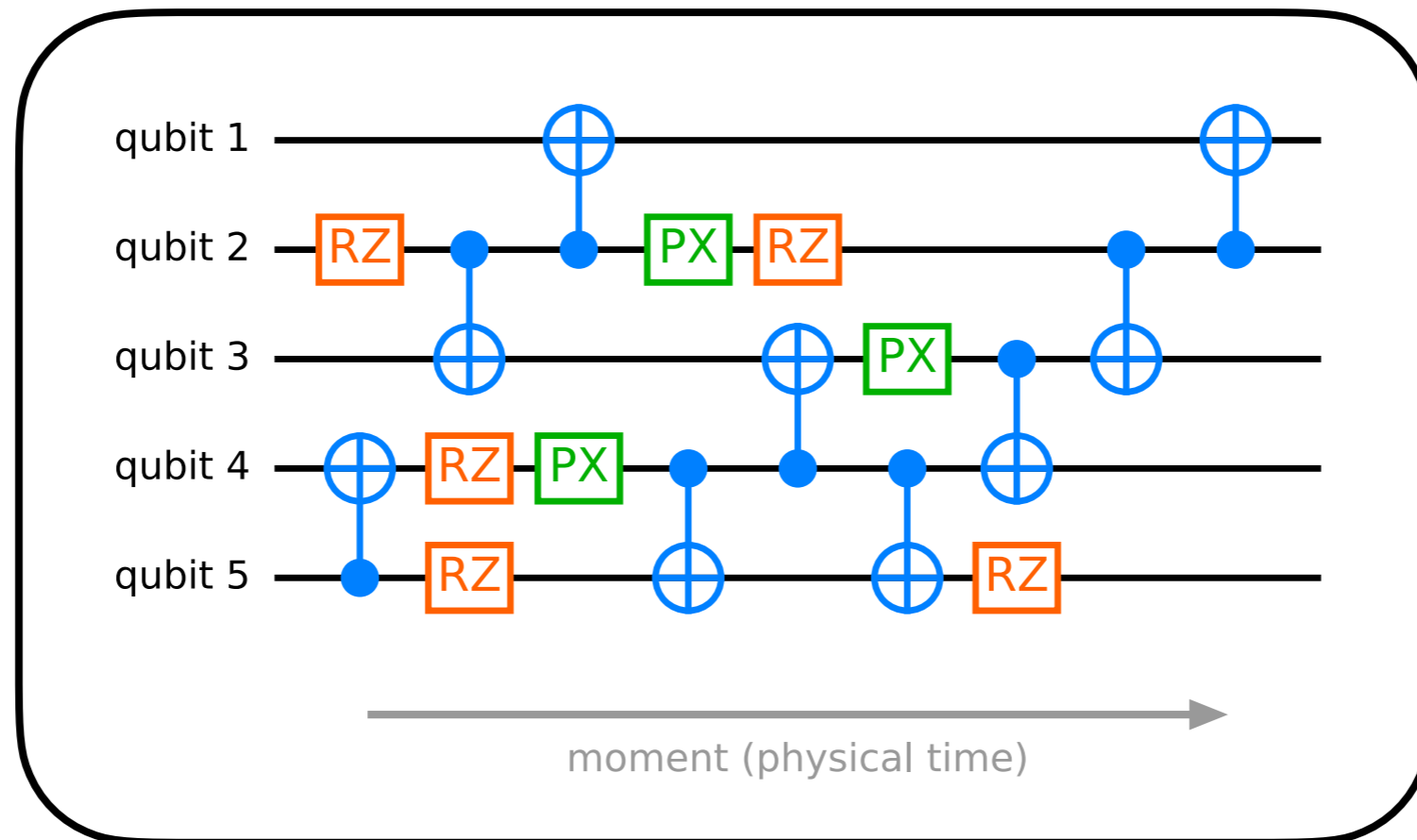
Feedback-based quantum control



Feedback-based quantum control



More abstract: modifying quantum circuits



state = whole quantum circuit

action = transformation (changing gates)

reward = e.g. whether circuit becomes shorter

Preliminaries

Delayed rewards:

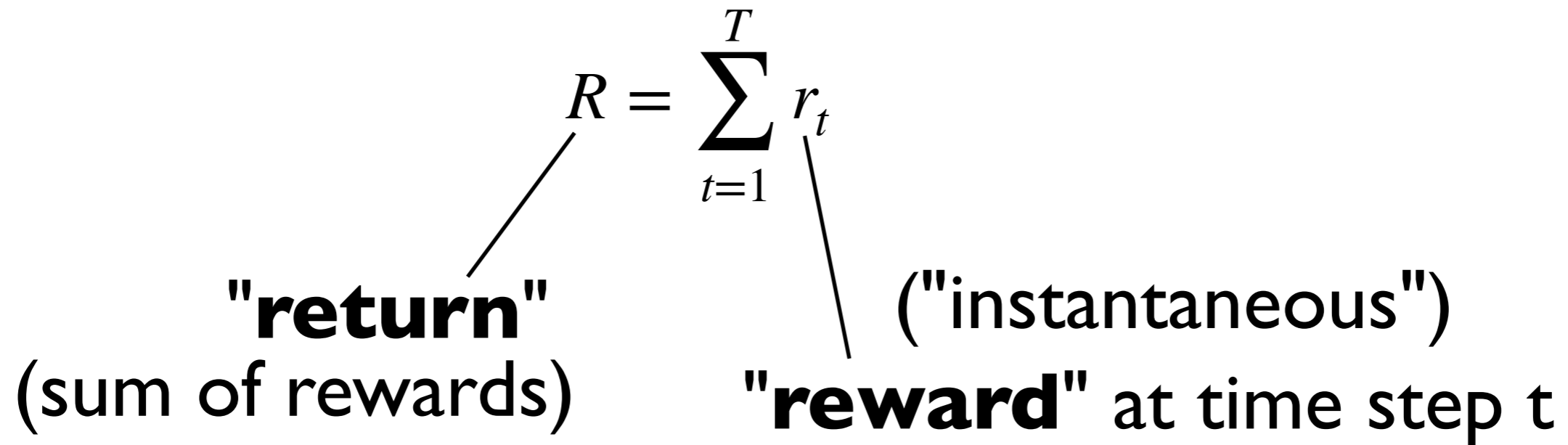
discounting,
greedy vs non-greedy

Adding up rewards and discounting

$$R = \sum_{t=1}^T r_t$$

"return"
(sum of rewards)

"reward" at time step t
("instantaneous")



We want to optimize the "return"!

(if the states are independent of the actions and the reward does not depend on the previous actions/
states, this becomes supervised learning again)

Adding up rewards and discounting

$$R_t = \sum_{t'=t}^T r_{t'}$$

(partial, future)
"return"
starting at time t

"reward" at time step t'

Can choose action at time t to optimize this instead of full R: this is equivalent, since former rewards do not depend on future actions!

Adding up rewards and discounting

"discounting factor"

$$R_t = \sum_{t'=t}^T r_{t'} \gamma^{t'-t}$$

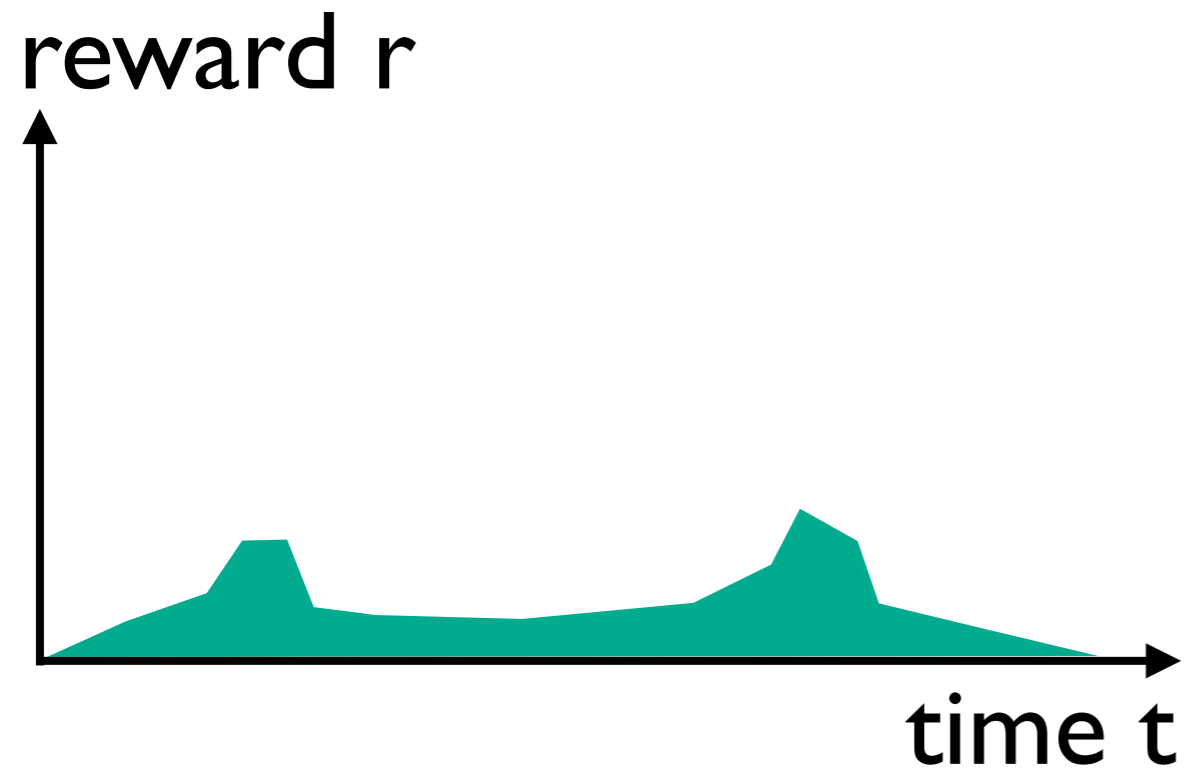
$$\gamma < 1$$

discounted future **"return"**
starting at time t

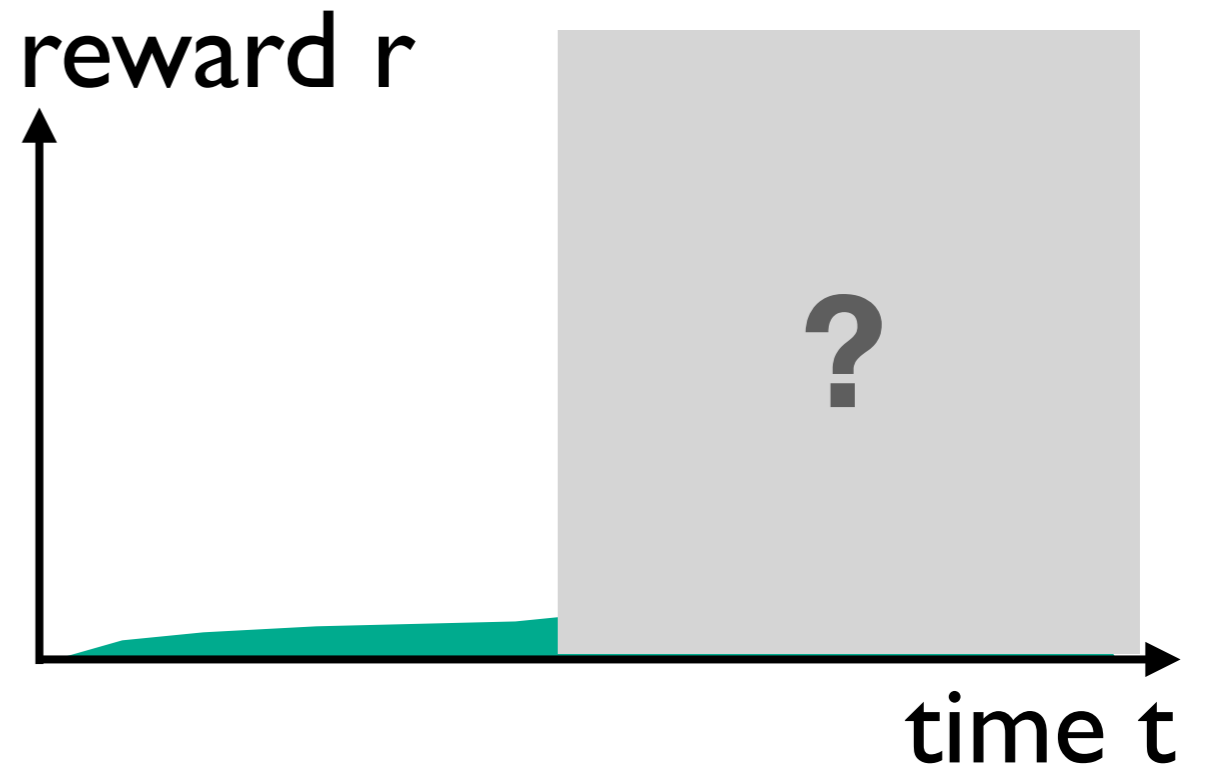
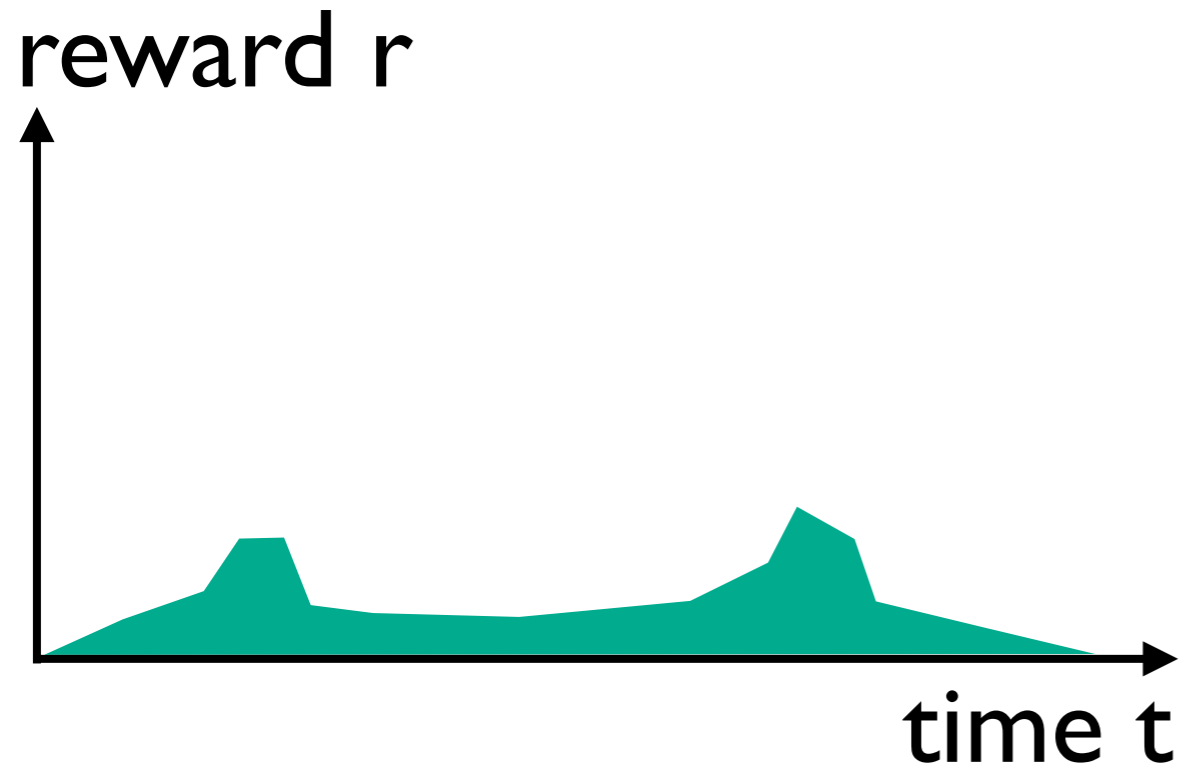
"reward" at time step t'

- prioritize sooner rewards over later rewards
- easier to optimize, but becomes "greedy" !

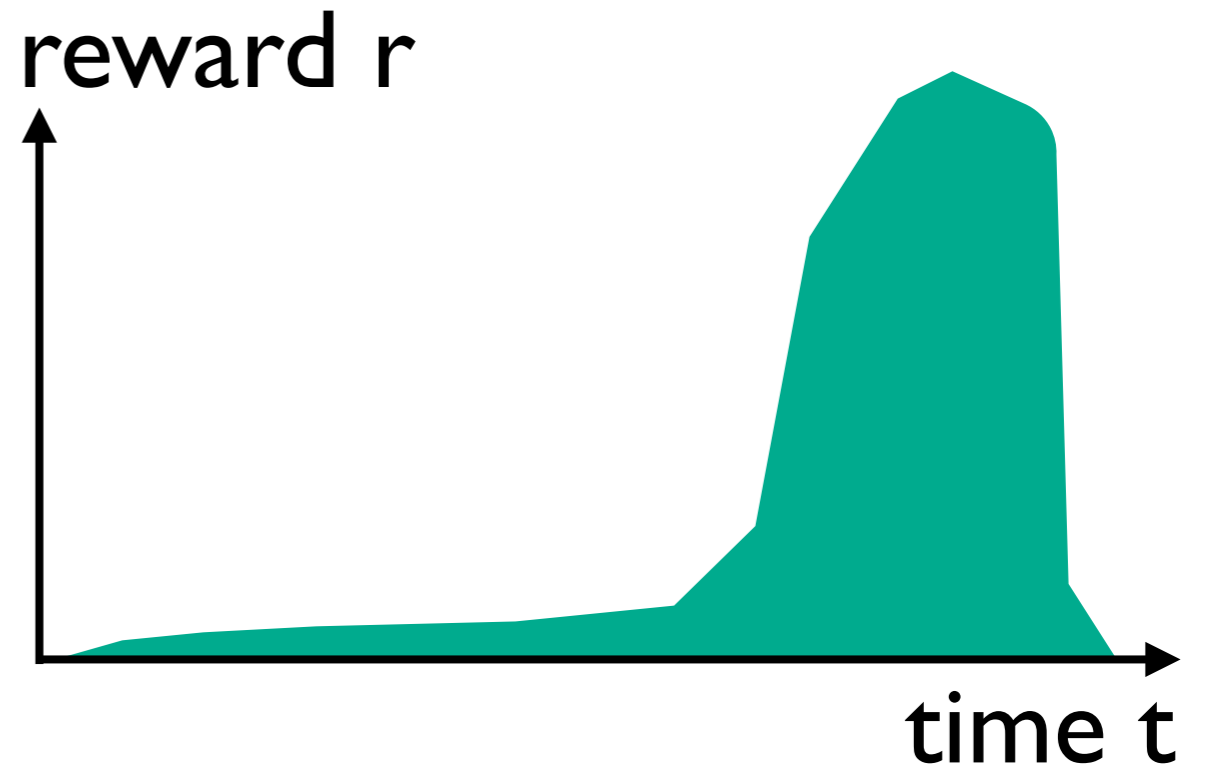
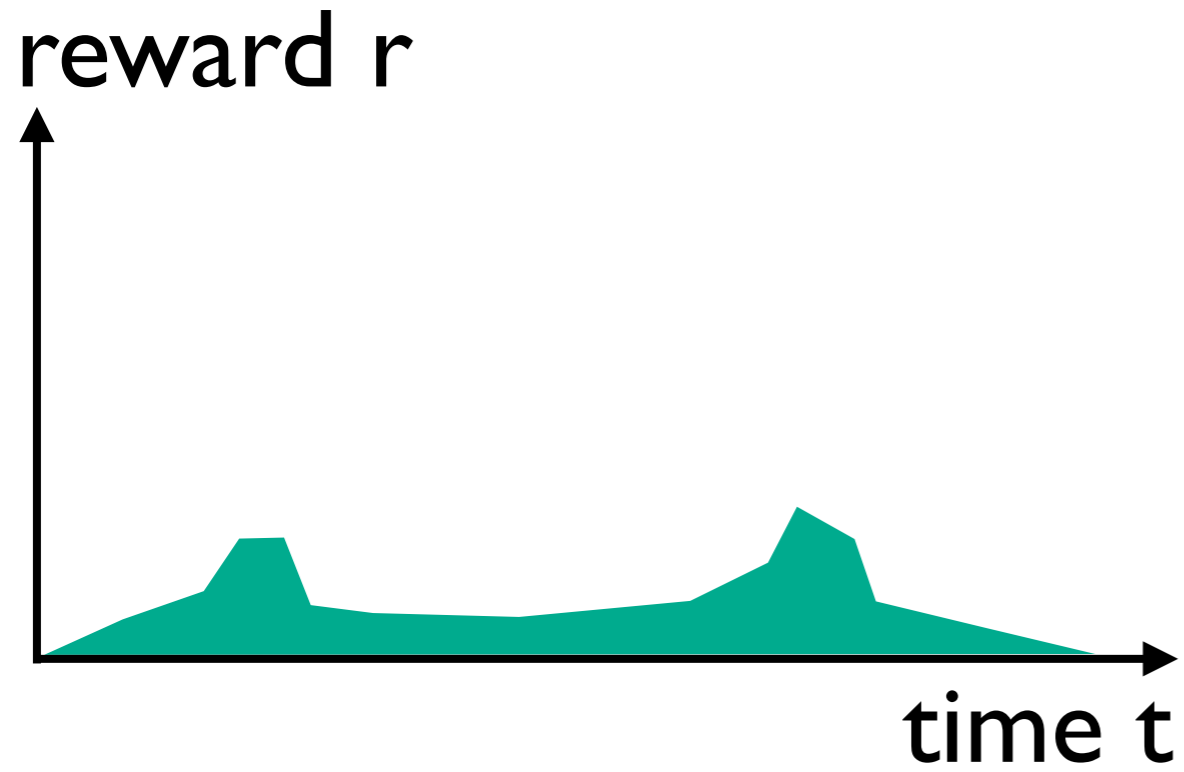
Adding up rewards and discounting



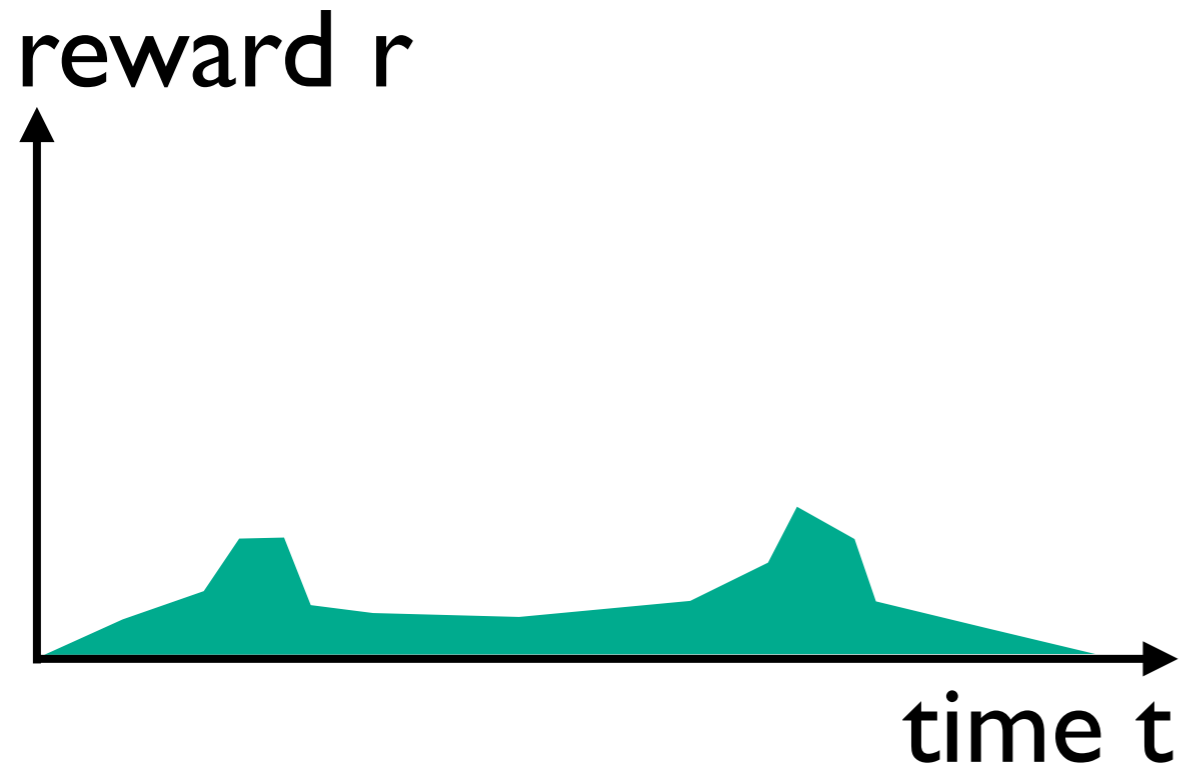
Adding up rewards and discounting



Adding up rewards and discounting

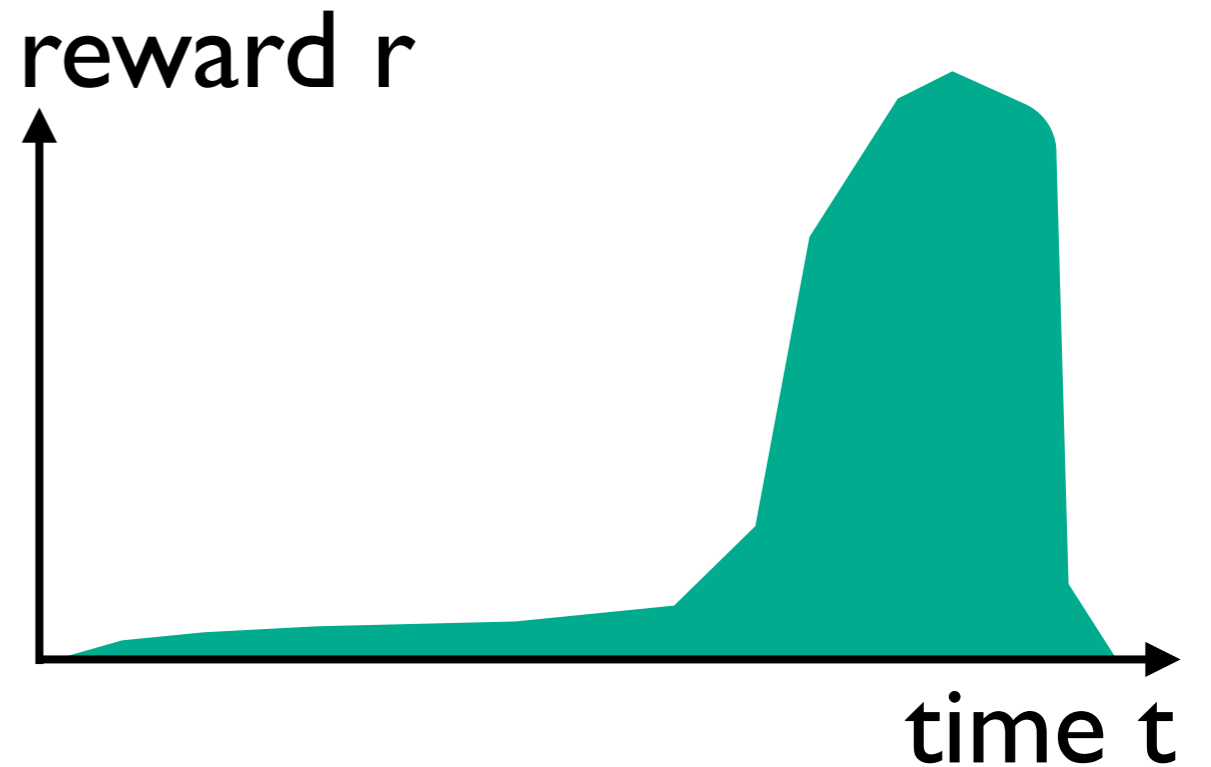


Adding up rewards and discounting



preferred by
"greedy" strategy

$$\gamma \ll 1$$



preferred by optimal
(non-greedy) strategy

$$\gamma = 1$$

Preliminaries

Model-based vs model-free
reinforcement learning

Return depends on environment dynamics

dynamics of environment

$$R = R(U(a))$$

action (at some
early time)

Return depends on environment dynamics

dynamics of environment

$$R = R(U(a))$$

action (at some
early time)

if model of environment is **known**:
use gradient descent with

$$\frac{\partial R}{\partial a} = \frac{\partial R}{\partial U} \frac{\partial U}{\partial a}$$

Return depends on environment dynamics

$$R = R(U(a)) \quad \frac{\partial R}{\partial a} = \frac{\partial R}{\partial U} \frac{\partial U}{\partial a}$$

model-based reinforcement learning

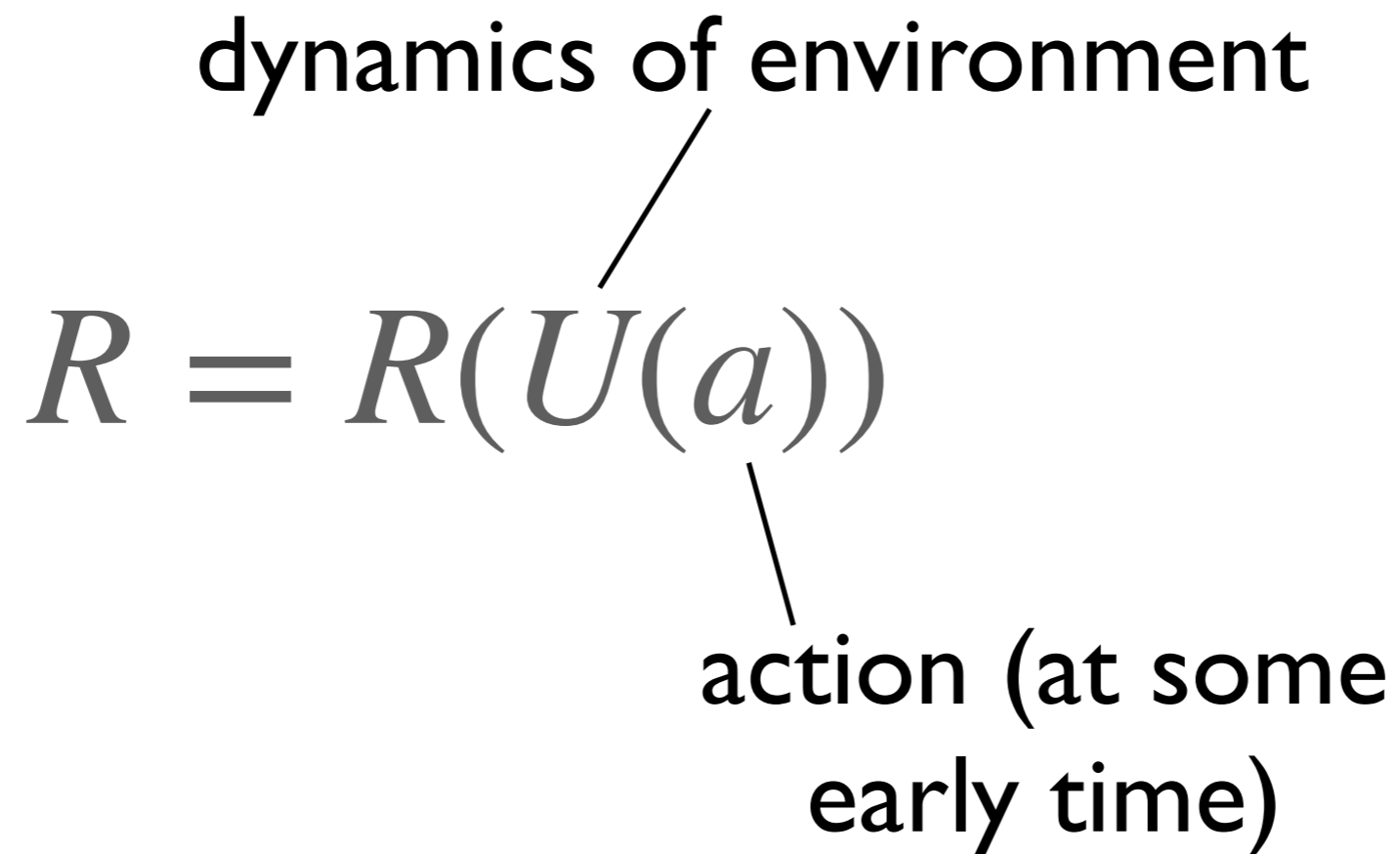
example in quantum physics: 'GRAPE'

What do we do if we do **not** know any model of the environment?
[or we do not want to adapt our algorithm to that particular model]

dynamics of environment

$$R = R(U(a))$$

action (at some early time)



Essential idea: we have to try out many action sequences and see what happens (learn when the return is high)
model-free reinforcement learning (these lectures)

Two basic approaches

(1) try out 'all' actions, make a table of R values, finally pick action with largest expected R

$$Q(a) = R(U(a)) \quad \text{"Q learning"}$$

(2) try actions stochastically, change action probabilities $P(a)$ to optimize average R

$$\bar{R} = \sum_a R(U(a))P(a) \quad \text{"policy gradient"}$$

Exploration/exploitation tradeoff

RL algorithms do not try all possible policies, but try to already use what they have learned so far to quickly come closer to the best policy

Danger: get stuck early in sub-optimal choices

Need to balance:

Exploitation = use what you have learned

Exploration = try something new

may introduce extra randomness for exploration

Preliminaries

Stochastic environments:

Markov decision process

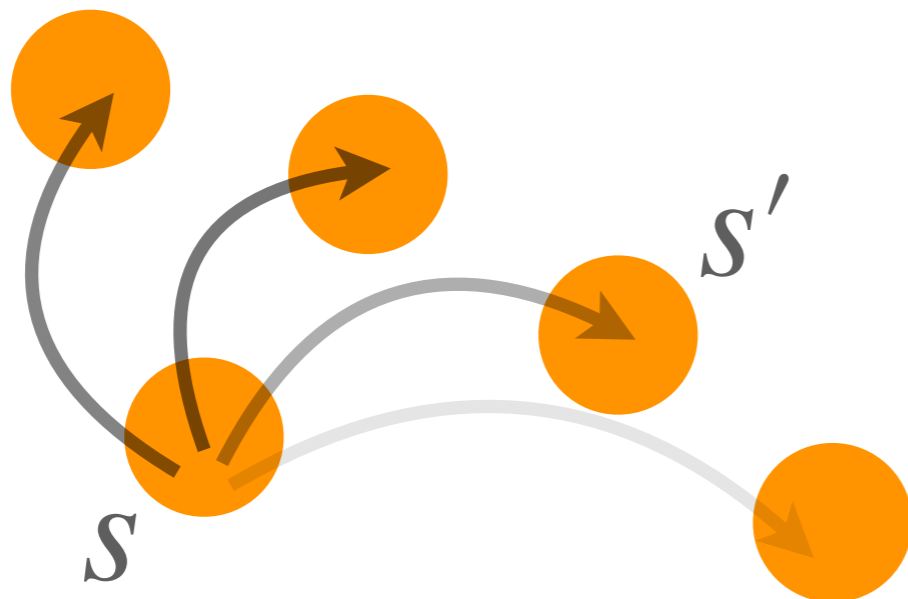
State of the environment: s

Transition function: $P(s' | s, a)$

Probability of going to state s' given that we were in state s and took action a

(could be deterministic: $P=1$ or 0)

"Markov decision process": Markov process (no memory) with decisions (actions based on states)



"what if the environment has memory"?

expand state space to include that memory,
going back to a Markov description

"what if the agent can only observe part of the state"?

simple approach: constrain allowable policies
(action choices) to depend only on that part
of the state

note: deterministic, Markovian dynamics on the
full state space can lead to non-deterministic, non-
Markovian dynamics on a restricted state space

Overview:

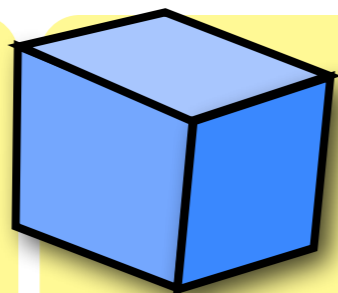
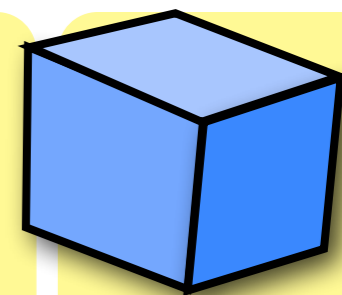
Model-free reinforcement learning

Learn action probabilities (policy gradient)

Learn expected returns (Q learning)

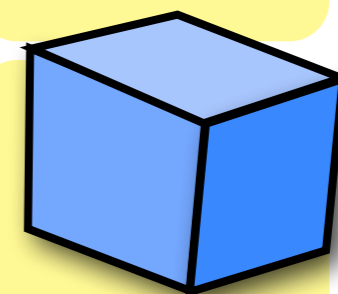
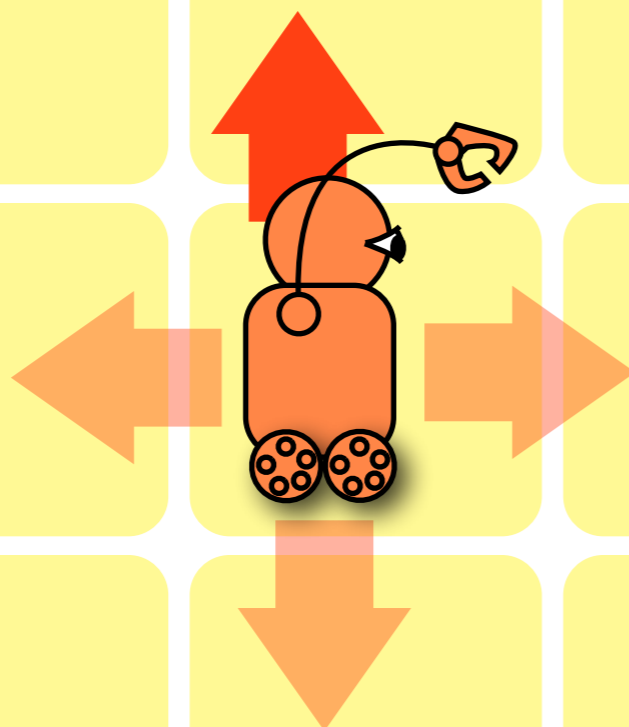
Learn both together (actor-critic)

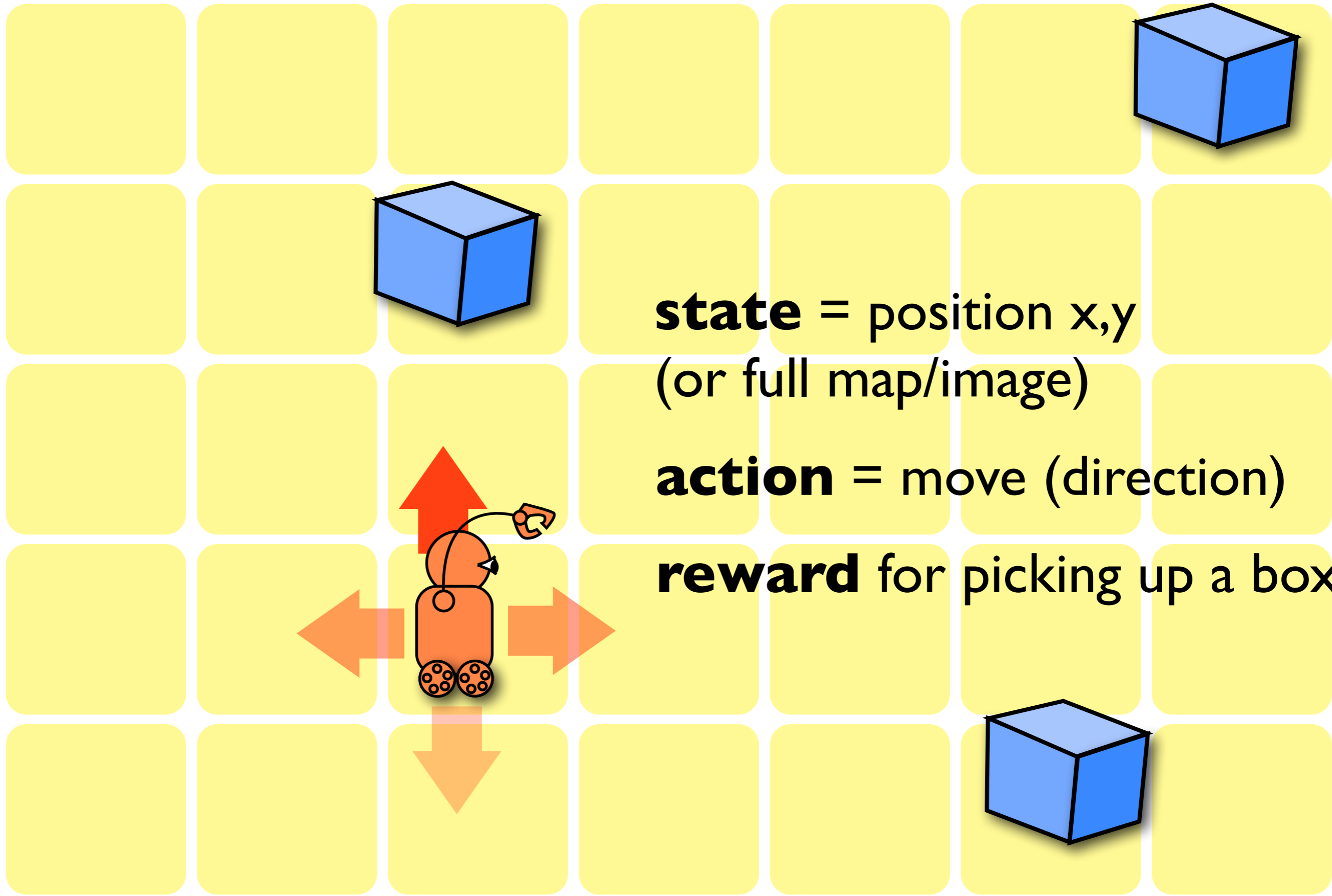
Policy gradient



state = position x,y
(or full map/image)

action = move (direction)





state = position x,y
(or full map/image)

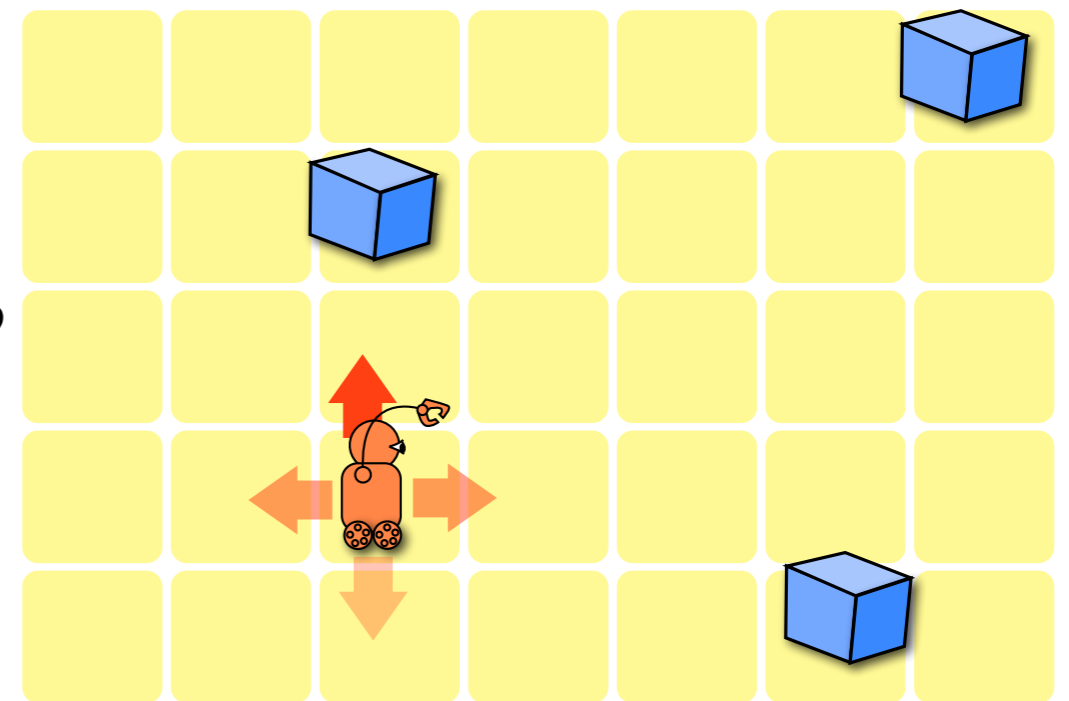
action = move (direction)

reward for picking up a box

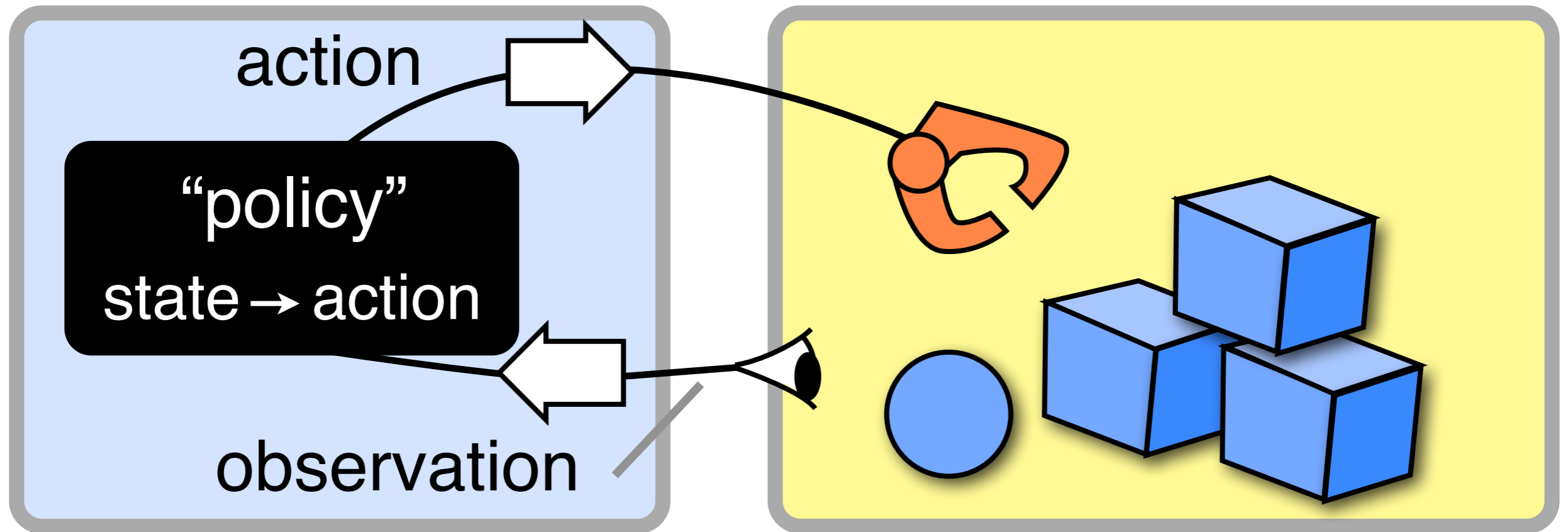
Policy Gradient

Policy gradient = REINFORCE (Williams 1992): A simple **model-free** general reinforcement learning technique

Basic idea: Use **probabilistic action choice**. If the return at the end turns out to be high, make **all** the actions in this sequence **more likely** (otherwise do the opposite)



This will also sometimes reinforce 'bad' actions, but since they occur more likely in trajectories with low reward, the net effect will still be to suppress them!



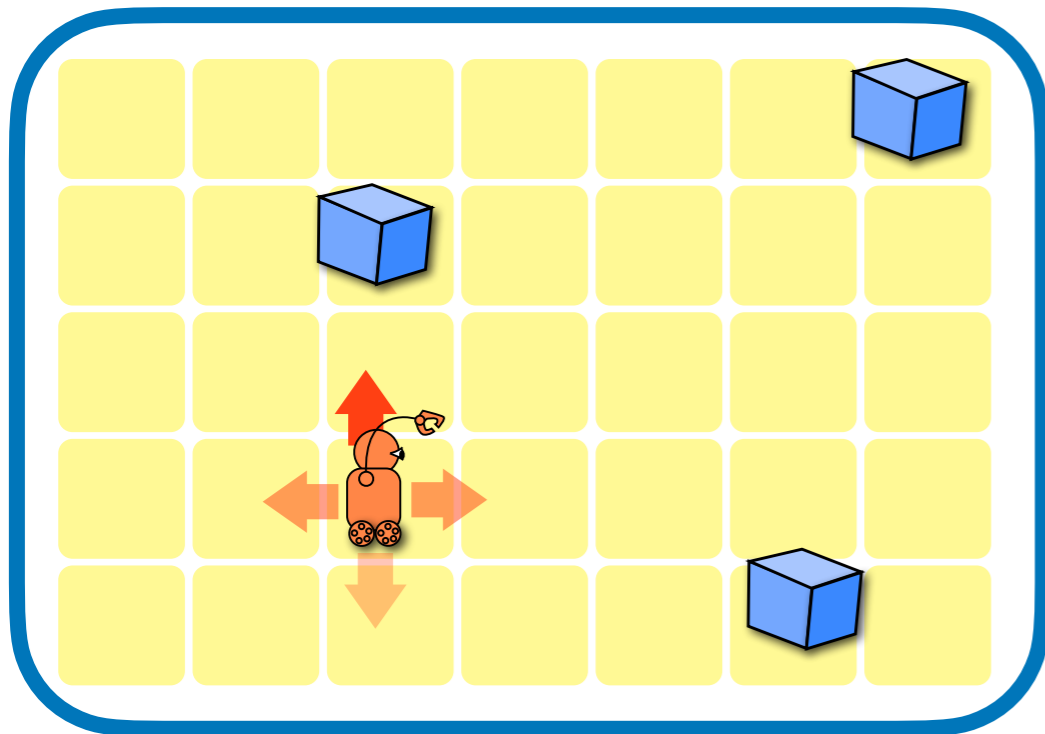
RL-agent

RL-environment

Policy: $\pi_{\theta}(a_t | s_t)$ – probability to pick action a_t given observed state s_t at time t

Policy Gradient

state s



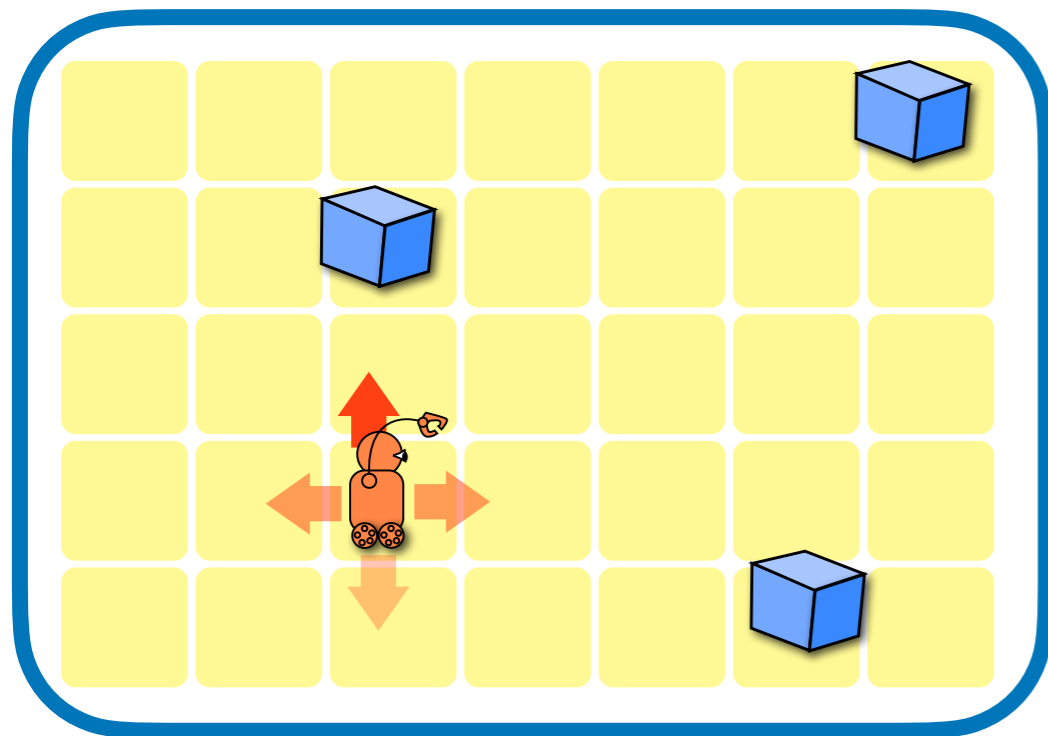
$\pi_{\theta}(a|s)$
policy

action probabilities

action a	probability
down	0.1
up	0.6
left	0.2
right	0.1

Policy Gradient

state s



action probabilities

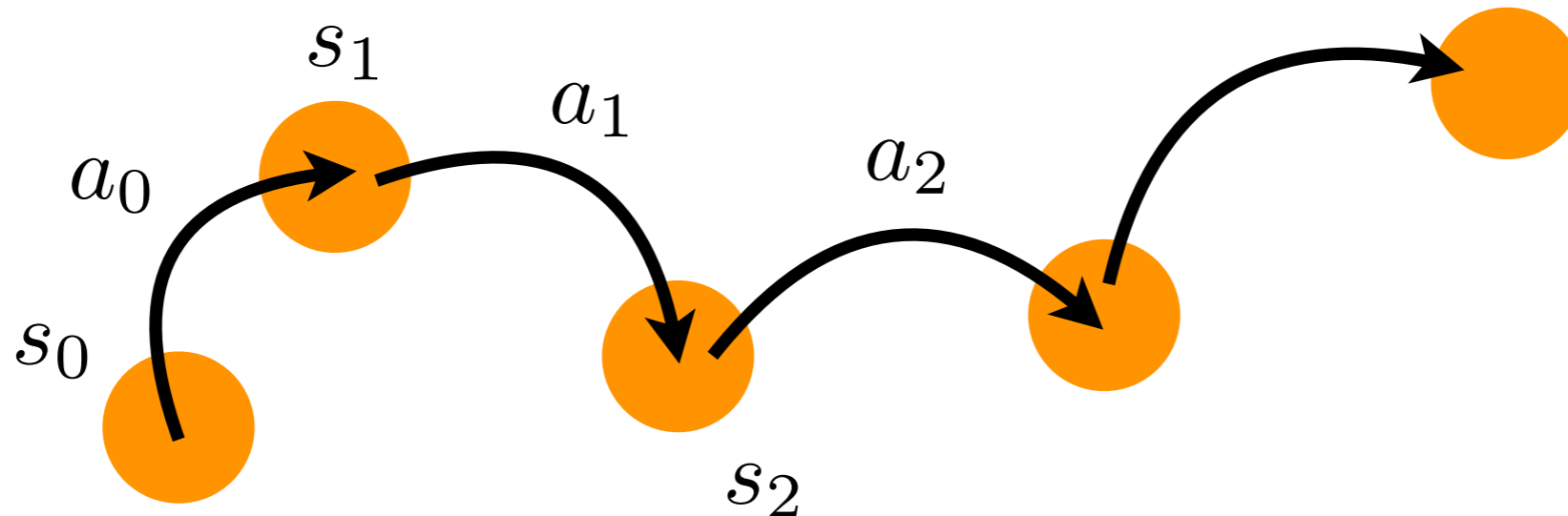
action a	probability
down	0.1
up	0.6
left	0.2
right	0.1

$\pi_{\theta}(a|s)$
policy

Environment: makes (possibly stochastic) transition to a new state s'

Transition function: $P(s'|s, a)$

Policy Gradient



Probability for having a certain trajectory of actions and states:

product over time steps

$$P_{\theta}(\tau) = \prod_t P(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$$

trajectory:

$$\tau = (\mathbf{a}, \mathbf{s})$$

$$\mathbf{a} = a_0, a_1, a_2, \dots$$

$$\mathbf{s} = s_1, s_2, \dots$$

Policy Gradient

Expected cumulative reward (= 'return'):
sum over all trajectories

$$\bar{R} = E[R] = \sum_{\tau} P_{\theta}(\tau) R(\tau) \quad \text{— return for this sequence (sum over individual rewards } r \text{ for all times)}$$

sum over all actions at all times and over all states at all times > 0

$$\sum_{\tau} \dots = \sum_{a_0, a_1, a_2, \dots, s_1, s_2, \dots} \dots$$

Try to maximize expected return by changing parameters of policy:

$$\frac{\partial \bar{R}}{\partial \theta} = ?$$

The logarithmic gradient trick

$$E[X_\theta(s)] = \sum_s P_\theta(s) X_\theta(s)$$

Problem: gradient with respect to parameter also acts on probability! Can we rewrite result as $E[\dots]$?

$$\frac{\partial}{\partial \theta} P_\theta(s) = P_\theta(s) \frac{\frac{\partial}{\partial \theta} P_\theta(s)}{P_\theta(s)} = P_\theta(s) \frac{\partial \ln P_\theta(s)}{\partial \theta}$$

$$\frac{\partial E[X_\theta(s)]}{\partial \theta} = E\left[\frac{\partial \ln P_\theta(s)}{\partial \theta} X_\theta(s)\right] + E\left[\frac{\partial X_\theta(s)}{\partial \theta}\right]$$

Policy Gradient

$$\bar{R} = E[R] = \sum_{\tau} P_{\theta}(\tau) R(\tau)$$

$$\frac{\partial \bar{R}}{\partial \theta} = ?$$

$$P_{\theta}(\tau) = \prod_t P(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$$

Derivative only acts on policy! (model-free!)

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t \sum_{\tau} R(\tau) \underbrace{\frac{\partial \pi_{\theta}(a_t | s_t)}{\partial \theta} \frac{1}{\pi_{\theta}(a_t | s_t)}}_{\frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta}} \underbrace{\prod_{t'} P(s_{t'+1} | s_{t'}, a_{t'}) \pi_{\theta}(a_{t'} | s_{t'})}_{P_{\theta}(\tau)}$$

Policy Gradient

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t \sum_{\tau} R(\tau) \underbrace{\frac{\partial \pi_{\theta}(a_t | s_t)}{\partial \theta} \frac{1}{\pi_{\theta}(a_t | s_t)}}_{\frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta}} \underbrace{\prod_{t'} P(s_{t'+1} | s_{t'}, a_{t'}) \pi_{\theta}(a_{t'} | s_{t'})}_{P_{\theta}(\tau)}$$

Main formula of policy gradient method:

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E \left[R \frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta} \right]$$

Stochastic gradient descent:

$$\Delta \theta = \eta \frac{\partial \bar{R}}{\partial \theta} \quad \text{where } E[\dots] \text{ is approximated via the value for one trajectory (or a batch)}$$

Policy Gradient

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E\left[R \frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta}\right]$$

Increase the probability of all action choices in the given sequence, depending on size of return R.

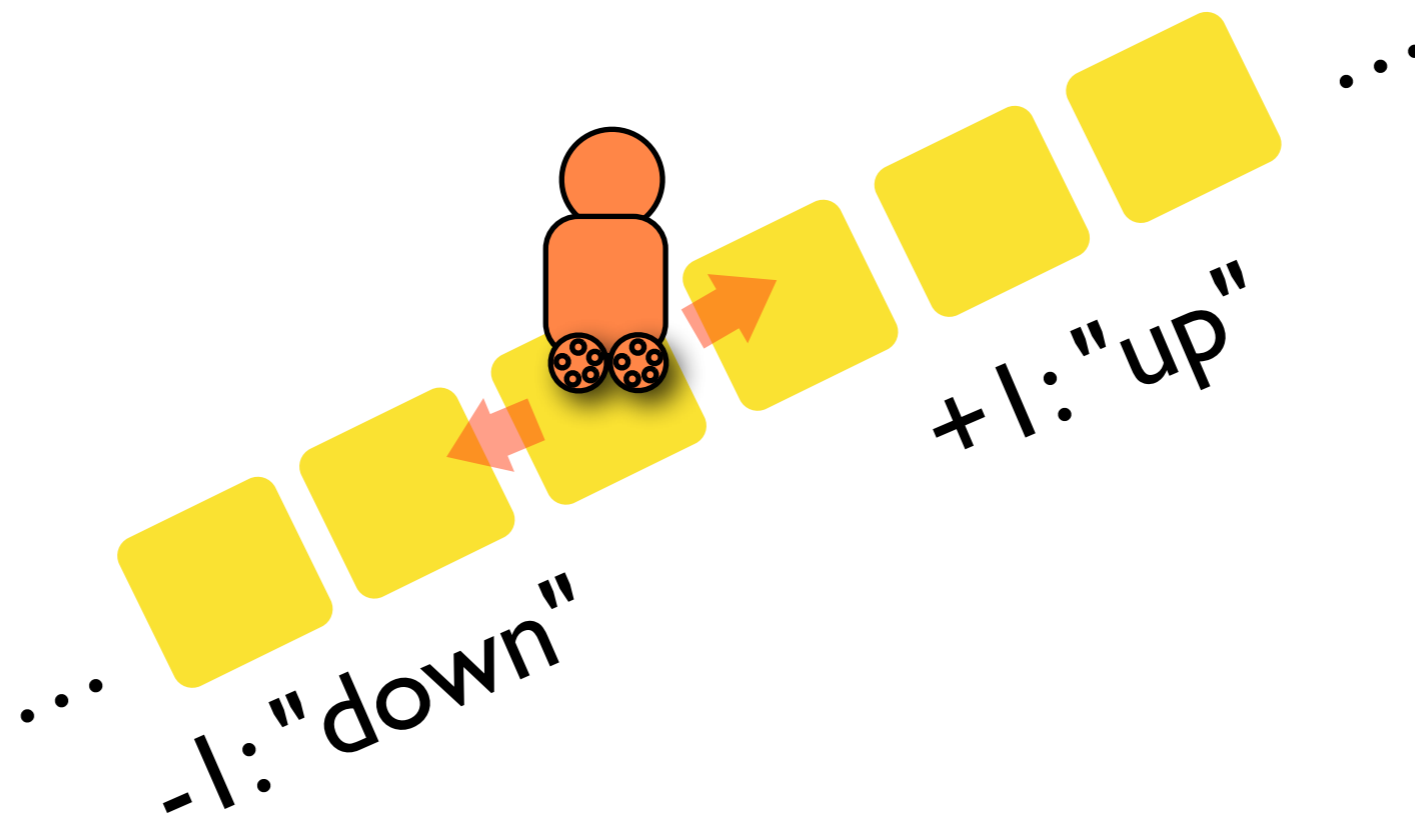
Even if $R > 0$ always, due to normalization of probabilities this will tend to suppress the action choices in sequences with lower-than-average returns.

Policy gradient:

The random walker toy example

The simplest RL example ever

random walker

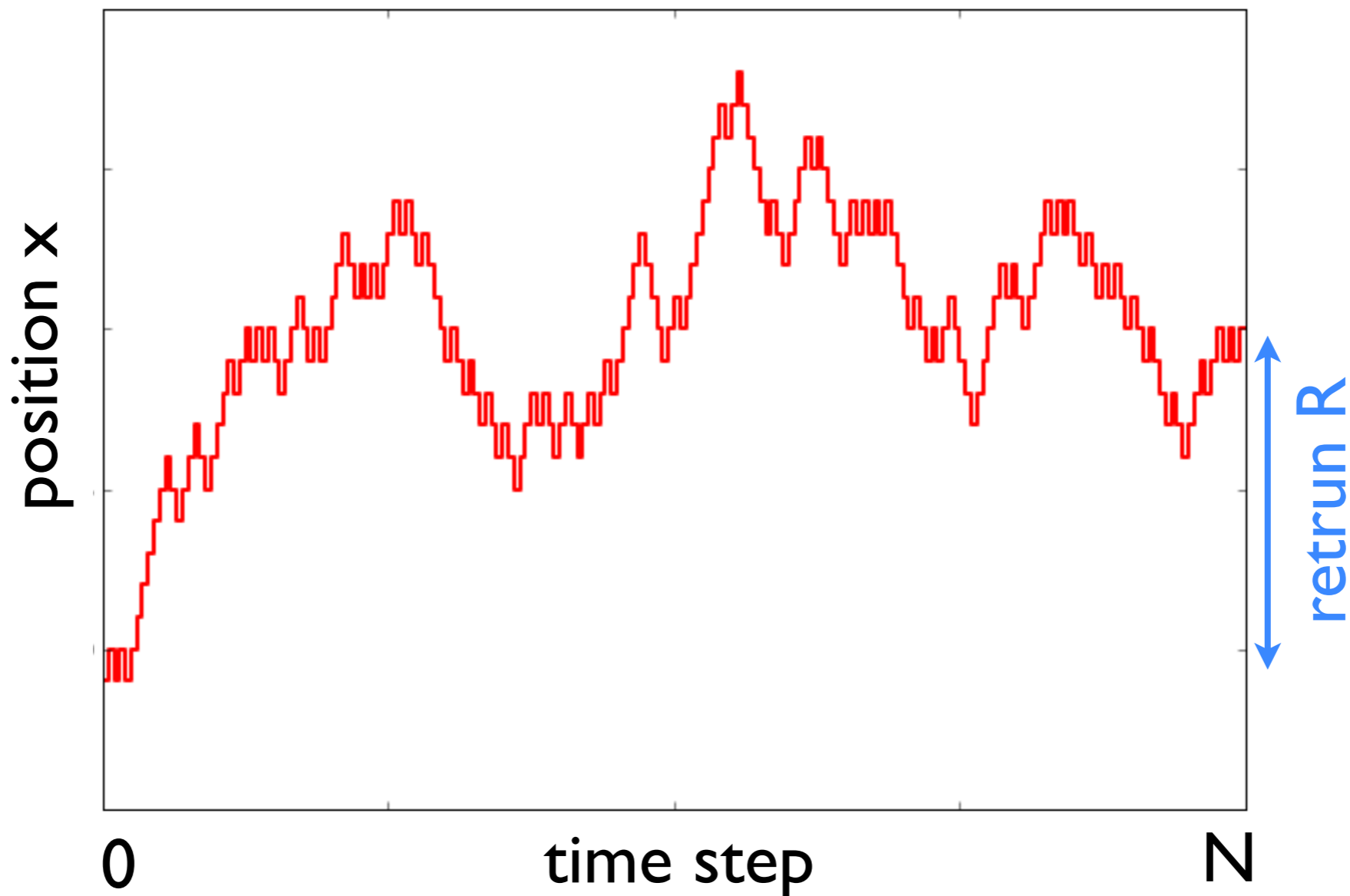


state = location
observed state = nothing (robot is blind)

The simplest RL example ever

A random walk, where the probability to go “up” is determined by the policy, and where the **return** is given by the final position $R = x(N)$

(Note: this policy does not even depend on the current state)



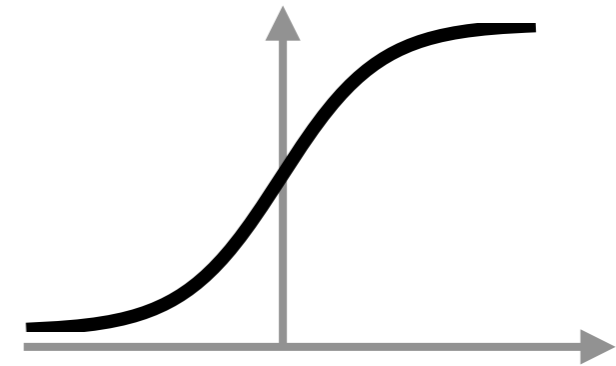
The simplest RL example ever

A random walk, where the probability to go “up” is determined by the policy, and where the **return** is given by the final position $R = x(N)$

(Note: this policy does not even depend on the current state)

policy

$$\pi_{\theta}(\text{up}) = \frac{1}{1 + e^{-\theta}}$$



RL update

$$\Delta\theta = \eta \sum_t \left\langle R \frac{\partial \ln \pi_{\theta}(a_t)}{\partial \theta} \right\rangle$$

$a_t = \text{up or down}$

The simplest RL example ever

return: $R = x(N) = N_{\text{up}} - N_{\text{down}} = 2N_{\text{up}} - N$

N=number of time steps

logarithmic gradients:

$$\sum_t \frac{\partial \ln \pi_{\theta}(a_t)}{\partial \theta} = N_{\text{up}} - N \pi_{\theta}(\text{up})$$

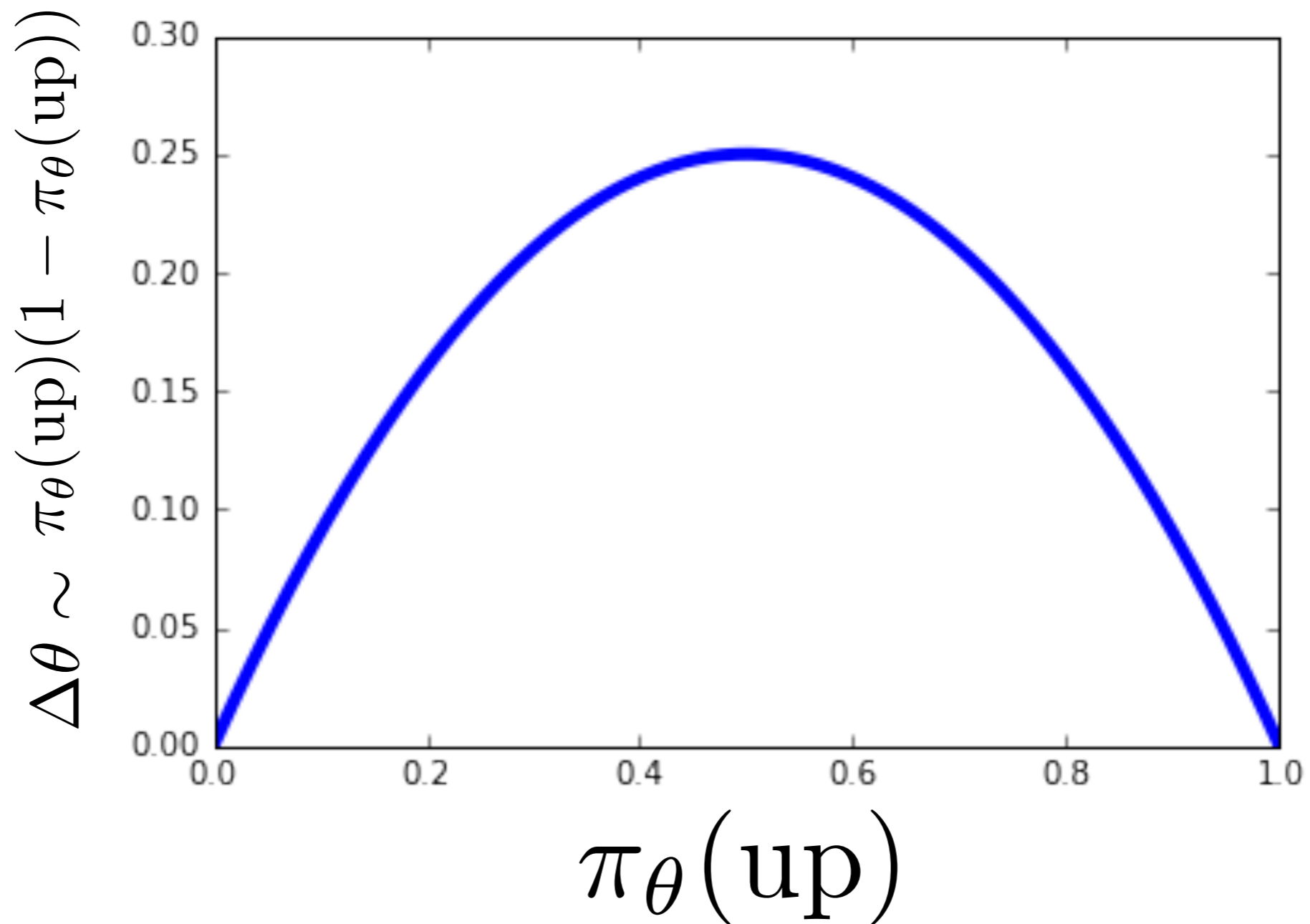
RL update:

$$\begin{aligned} \left\langle R \sum_t \frac{\partial \ln \pi_{\theta}(a_t)}{\partial \theta} \right\rangle &= 2 \left\langle \left(N_{\text{up}} - \frac{N}{2} \right) (N_{\text{up}} - \bar{N}_{\text{up}}) \right\rangle \\ &= 2 \text{Var} N_{\text{up}} = 2N \pi_{\theta}(\text{up}) (1 - \pi_{\theta}(\text{up})) \end{aligned}$$

(general analytical expression for average update, rare)

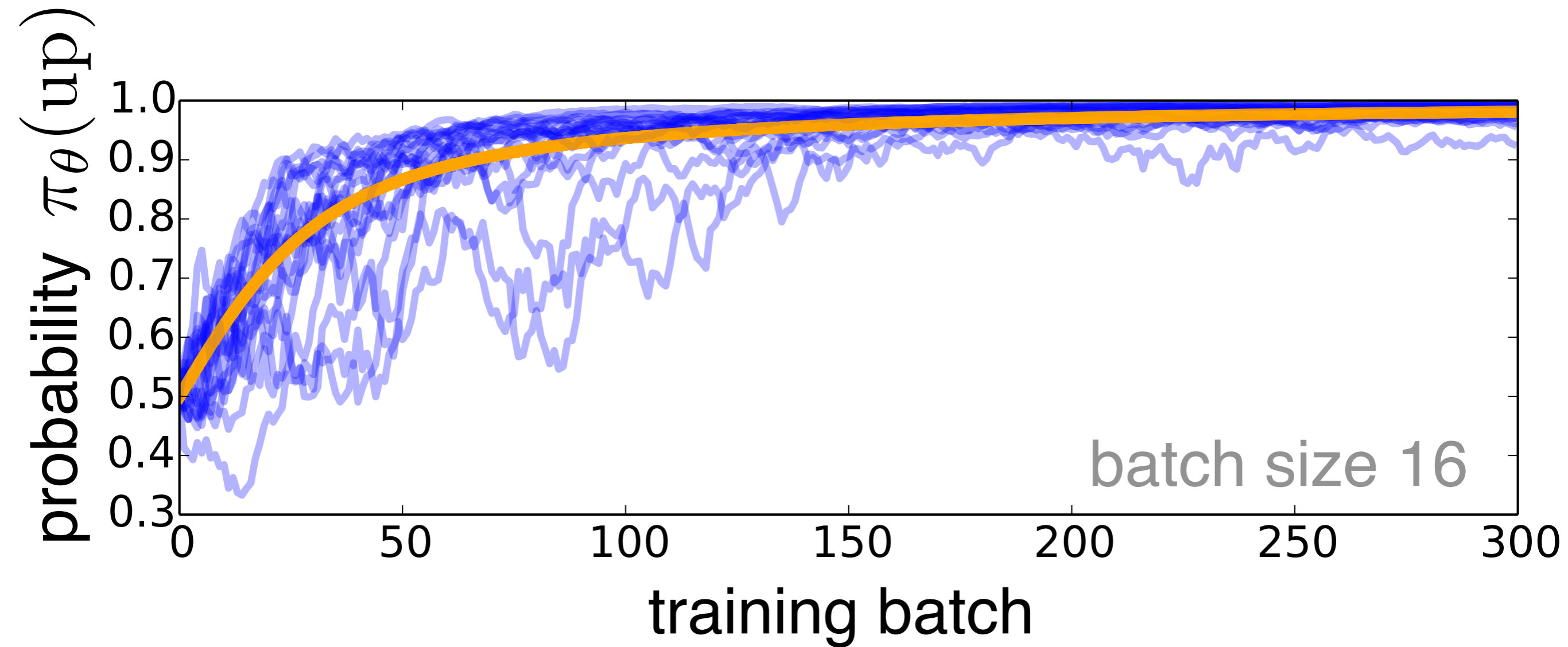
The simplest RL example ever

RL update for θ as a function of "up"-probability



probability to go up always increases (good!)

The simplest RL example ever

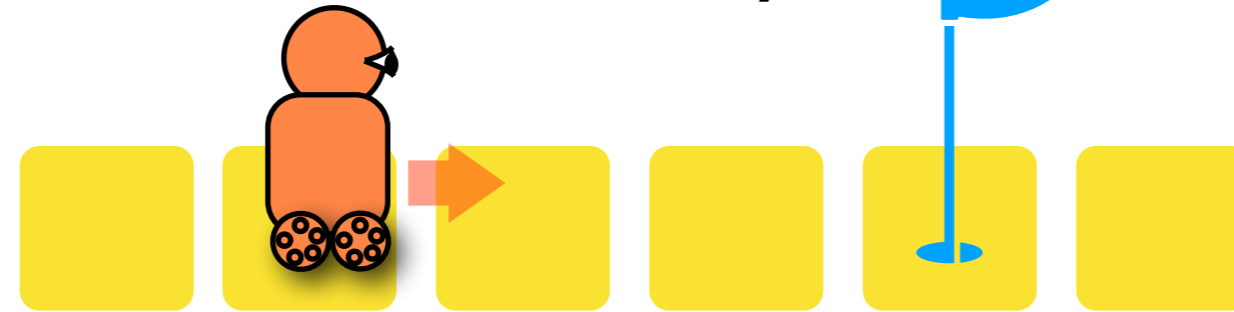


Can also analyze: spread of update step,
improvement via "baseline" $R \mapsto R - b$
(see later)

Policy gradient:
The walker/target toy example

The second-simplest RL example

actions: move or stay



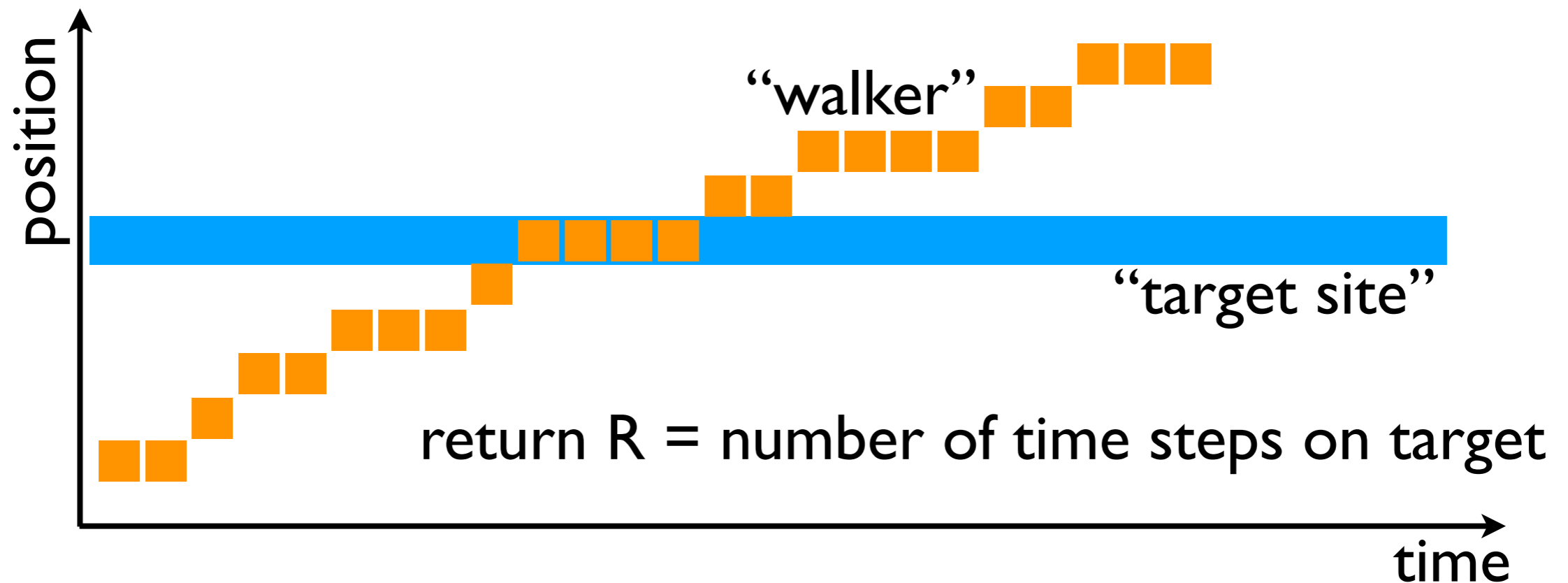
walker/target

The second-simplest RL example

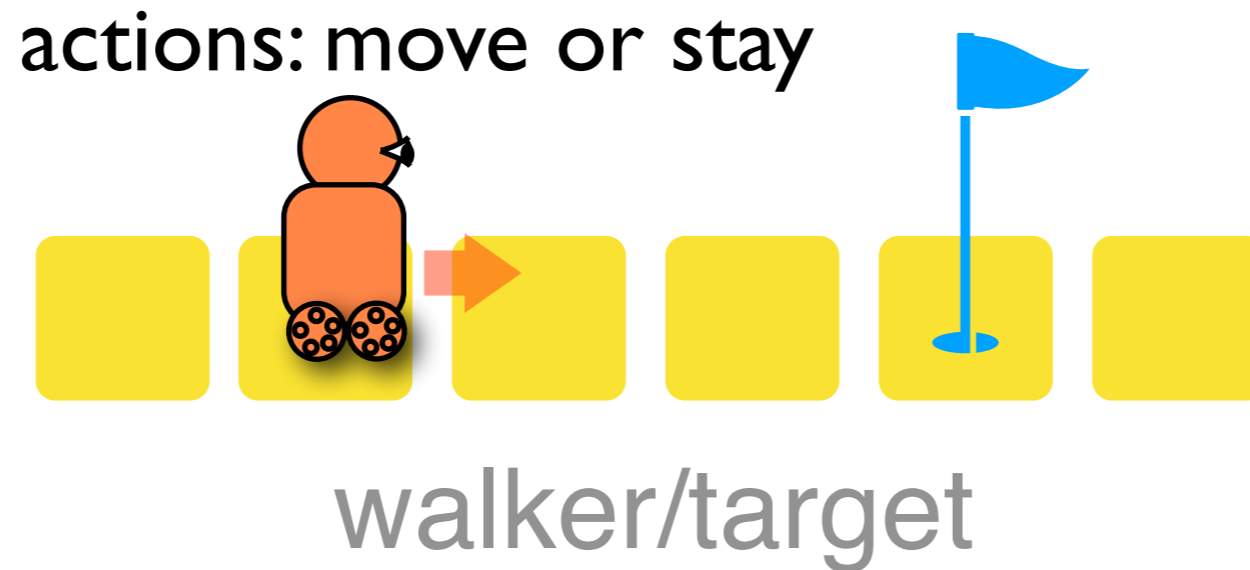
actions: move or stay



walker/target



The second-simplest RL example



state = location of robot and location of target (is random)

observed state = 1 when on target

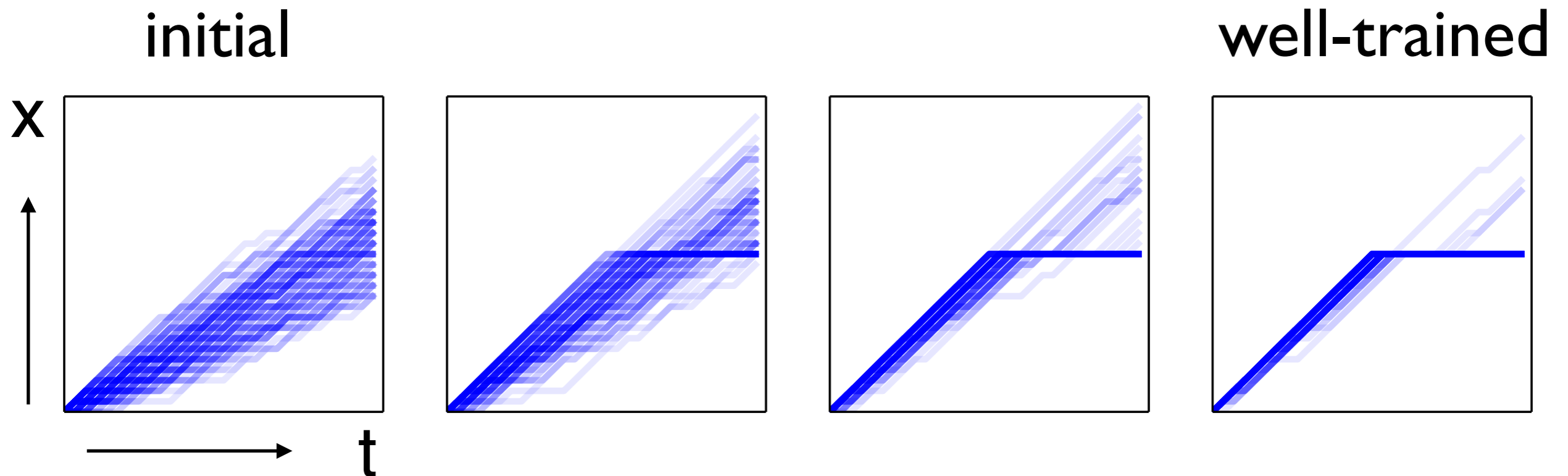
= 0 when off target

policy:

$\pi_{\theta}(1|0)$ "move when not on target"

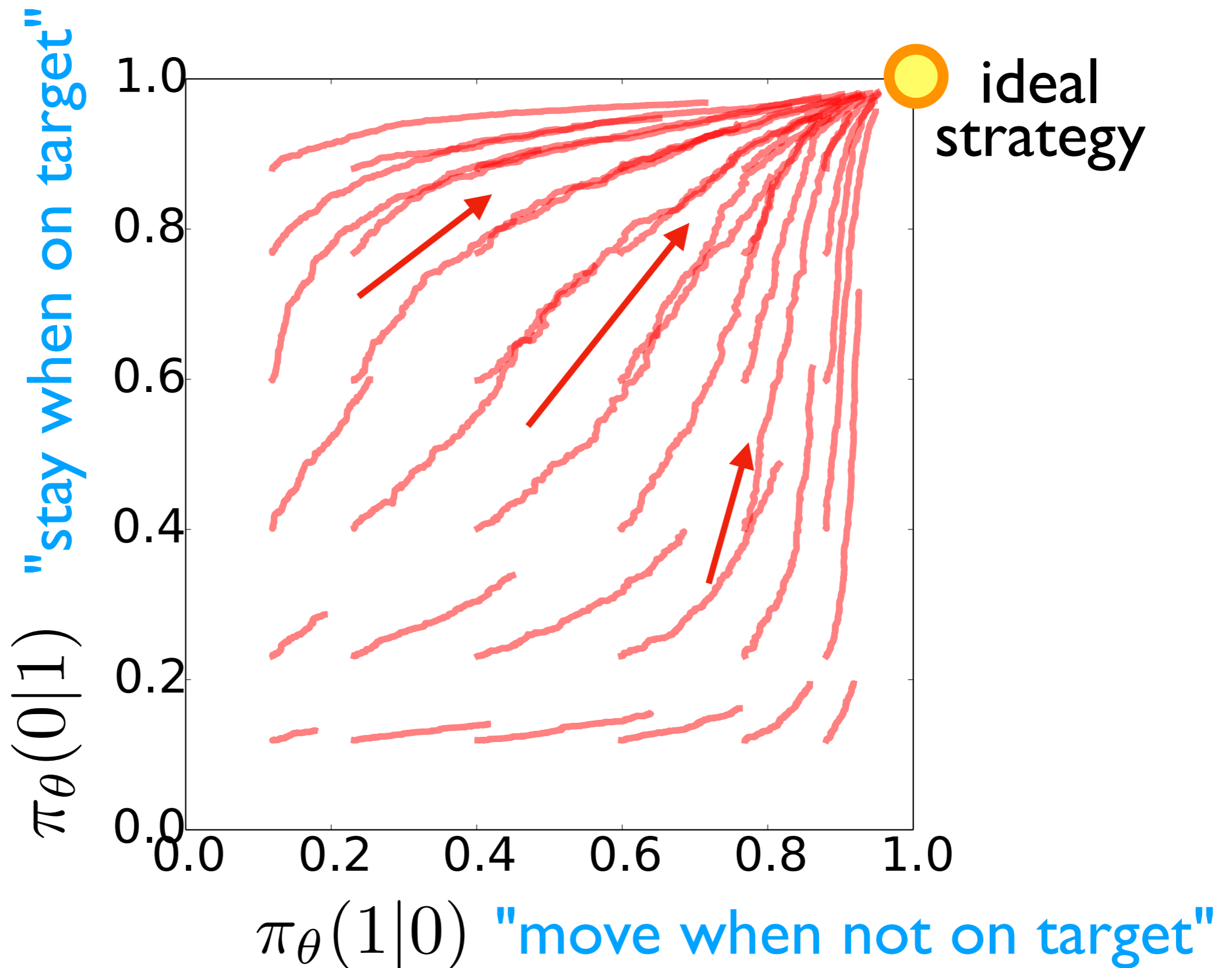
etc.

"Walker/target": evolution during training

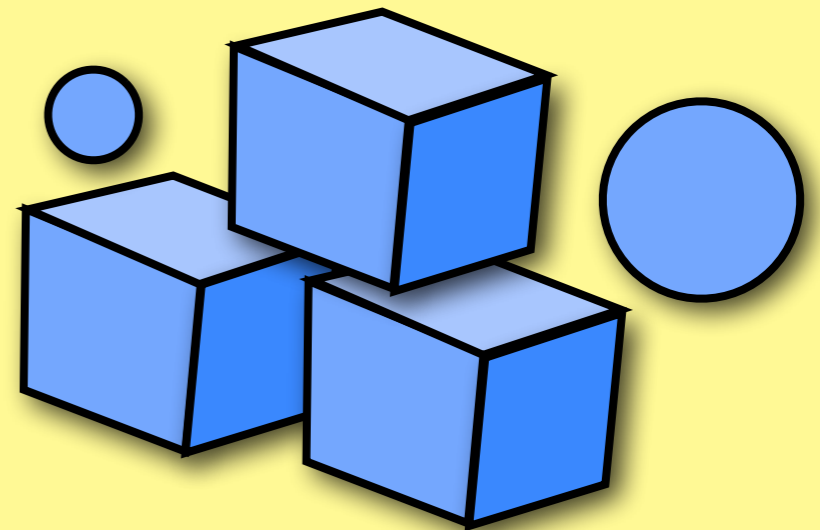
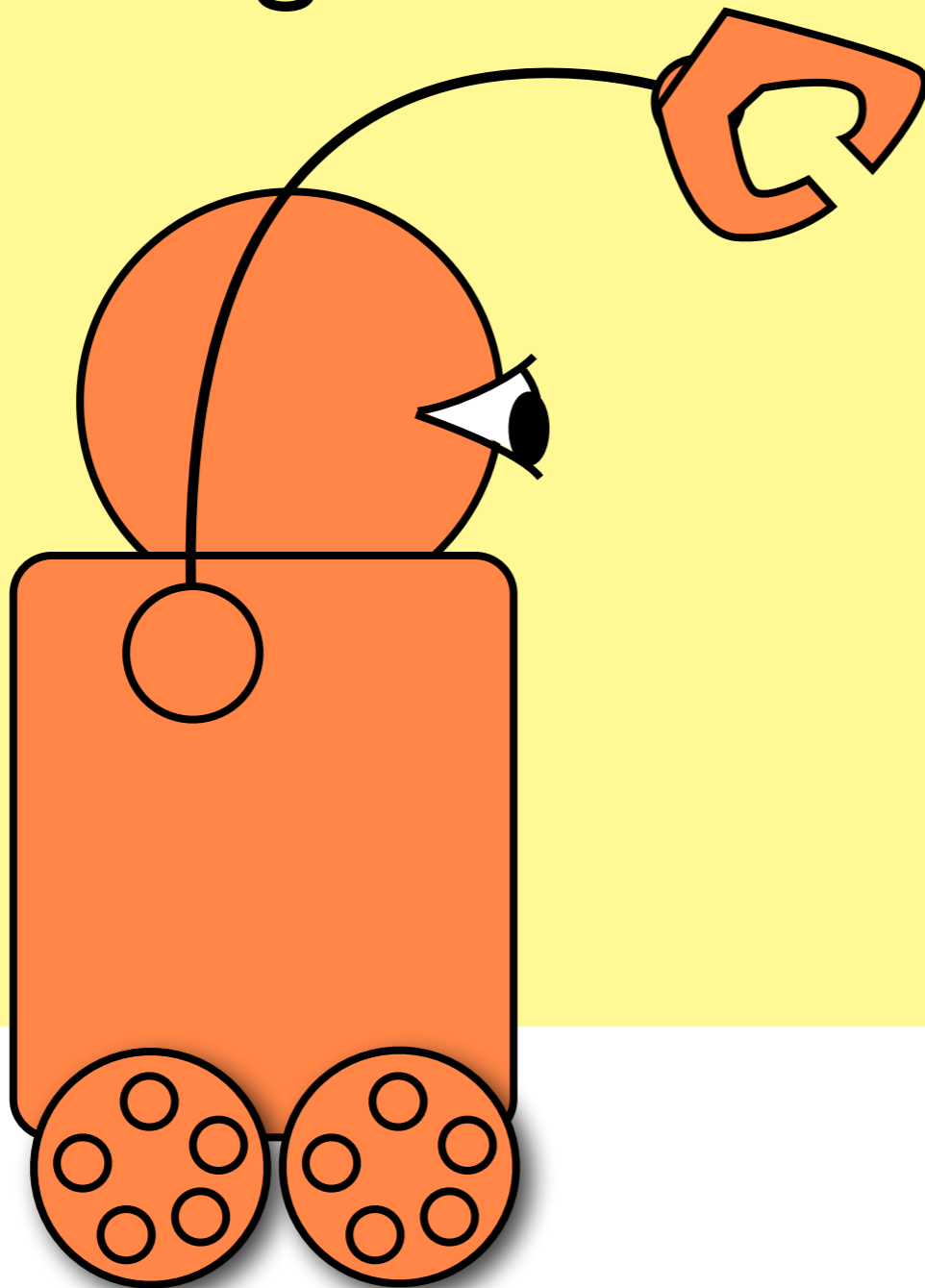


(note: location of target is chosen randomly during training, but here we display only trajectories for one fixed target location)

"Walker/target": evolution during training

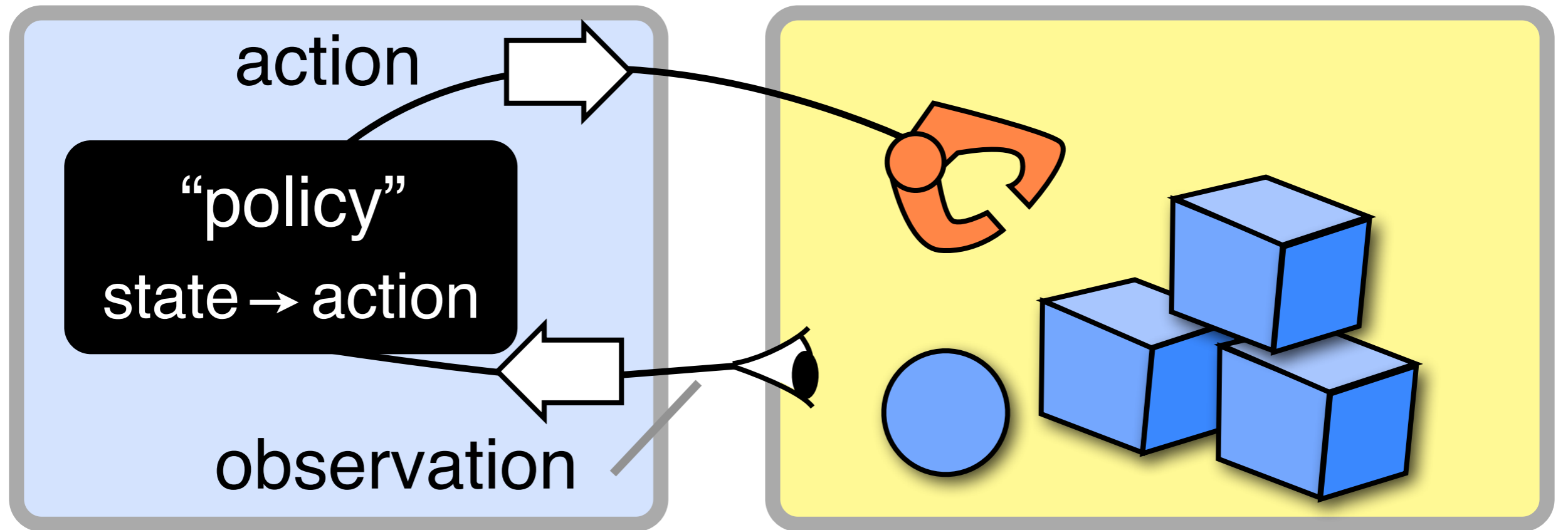


Reinforcement learning:
Discover strategies ('policies') =
actions in response to observations,
maximizing rewards



- ✓ Reinforcement learning: basic setting
- ✓ Examples (preview)
- ✓ Some preliminaries
 - Policy gradient
 - ✓ basics
 - ✓ toy examples
 - with neural networks
 - first quantum physics example
 - AlphaGo
 - Q-learning
 - basics
 - example: video games
 - Actor-critic (briefly)

Quick recap: policy gradient

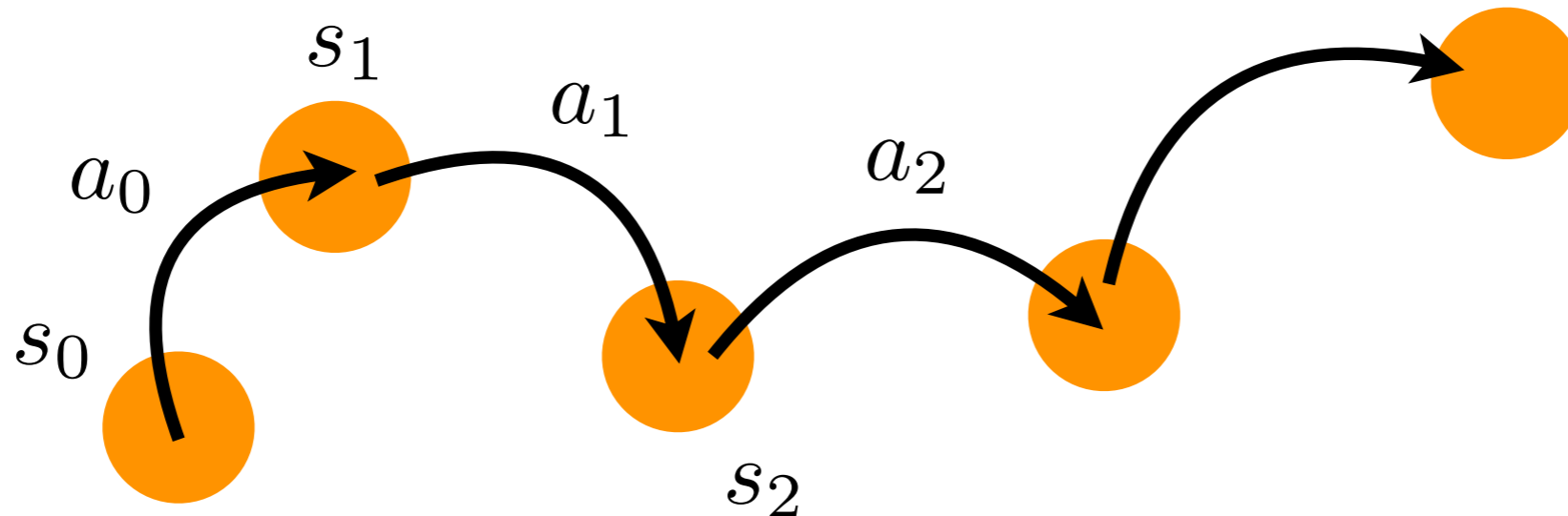


RL-agent

RL-environment

Policy: $\pi_{\theta}(a_t | s_t)$ – probability to pick action a_t given observed state s_t at time t

Policy Gradient



Probability for having a certain trajectory of actions and states:

product over time steps

$$P_{\theta}(\tau) = \prod_t P(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$$

trajectory:

$$\tau = (\mathbf{a}, \mathbf{s})$$

$$\mathbf{a} = a_0, a_1, a_2, \dots$$

$$\mathbf{s} = s_1, s_2, \dots$$

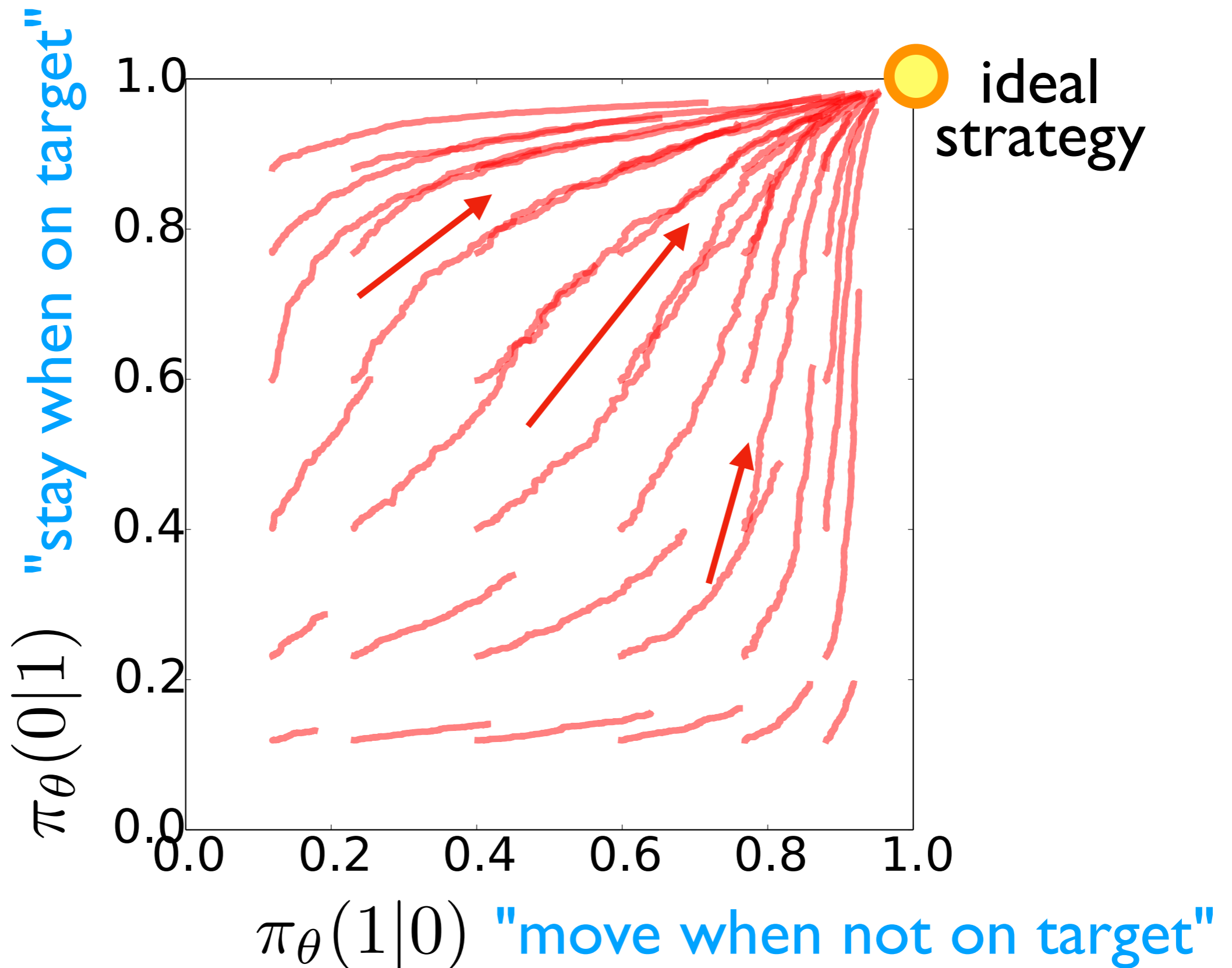
Policy Gradient

$$\Delta\theta \sim \frac{\partial \bar{R}}{\partial \theta} = \sum_t E\left[R \frac{\partial \ln \pi_\theta(a_t | s_t)}{\partial \theta}\right]$$

depends on particular trajectory τ

Increase the probability of all action choices in the given sequence, depending on size of return R.

"Walker/target": evolution during training



Policy gradient with a neural network

Motivations for using neural networks:

treat high-dimensional inputs:

- images
- time-series (measurements, sentences, ...)

treat high-dimensional outputs:

- e.g. placing a stone in a board game
- many control degrees of freedom in physics

exploit underlying structure in data

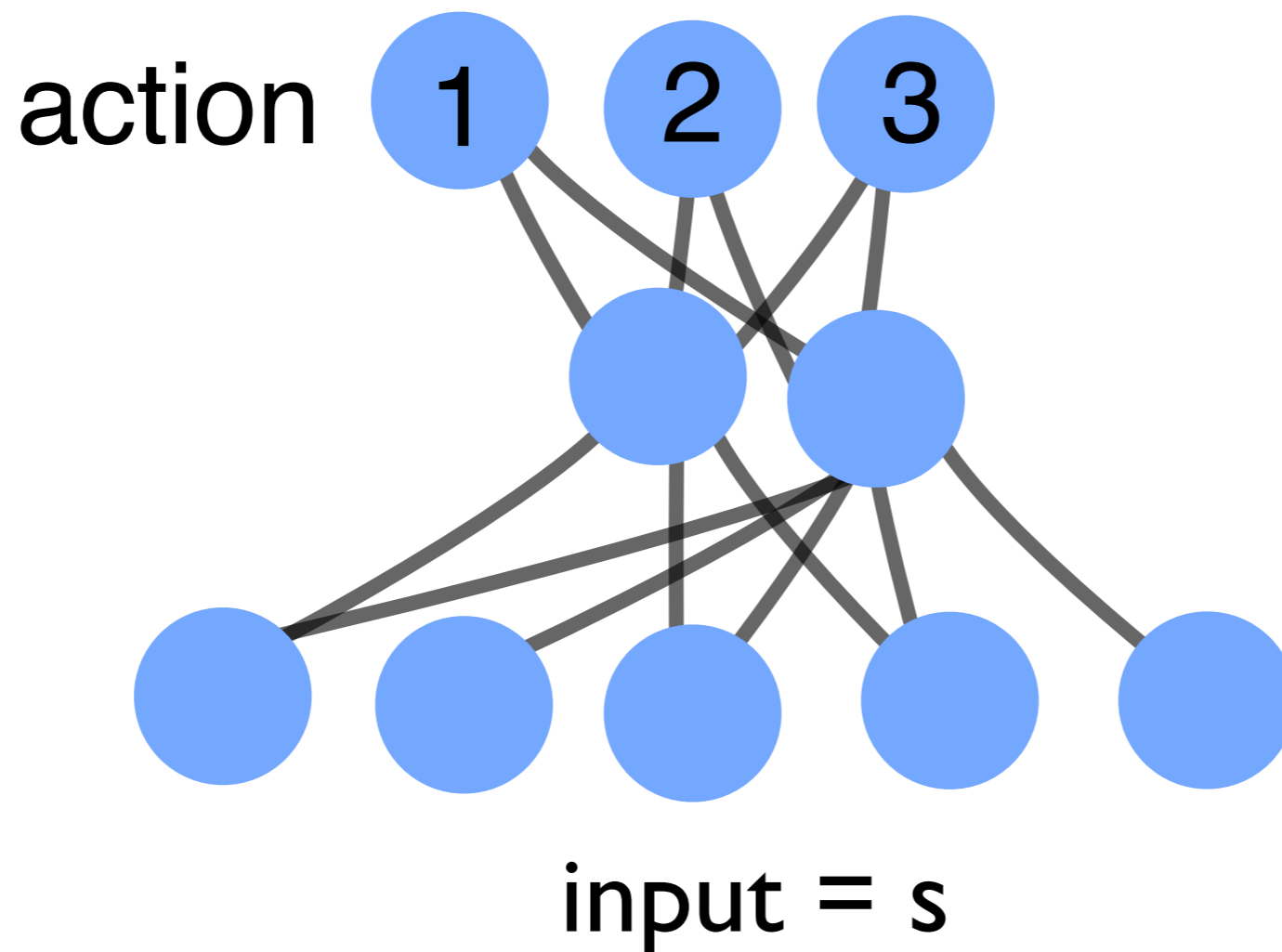
(such that we do not need to sample too much!)

Policy via neural network

policy $\pi_{\theta}(a|s)$

via a (deep) neural network

output = action probabilities (softmax)



(can be high-dimensional, e.g. picture)

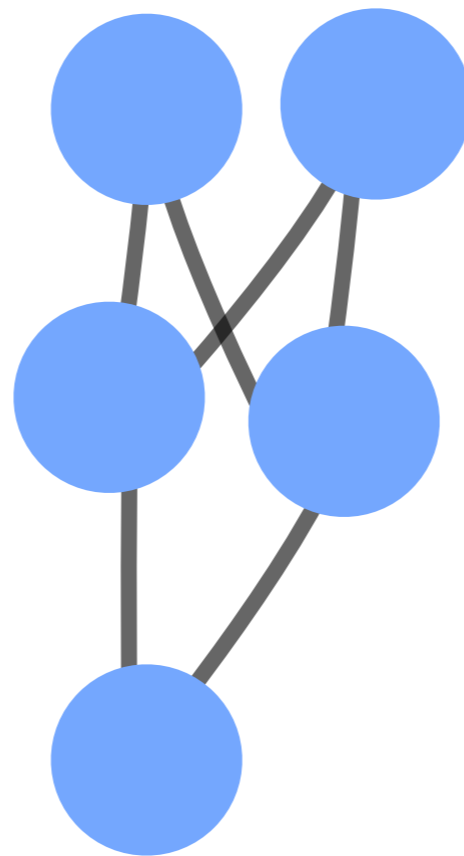
"Walker/target": neural network version

policy $\pi_{\theta}(a|s)$ via a neural network

output = action probabilities (softmax)

a=0 ("stay")

a=1 ("move")



input = s = "are we on target"? (0/1)

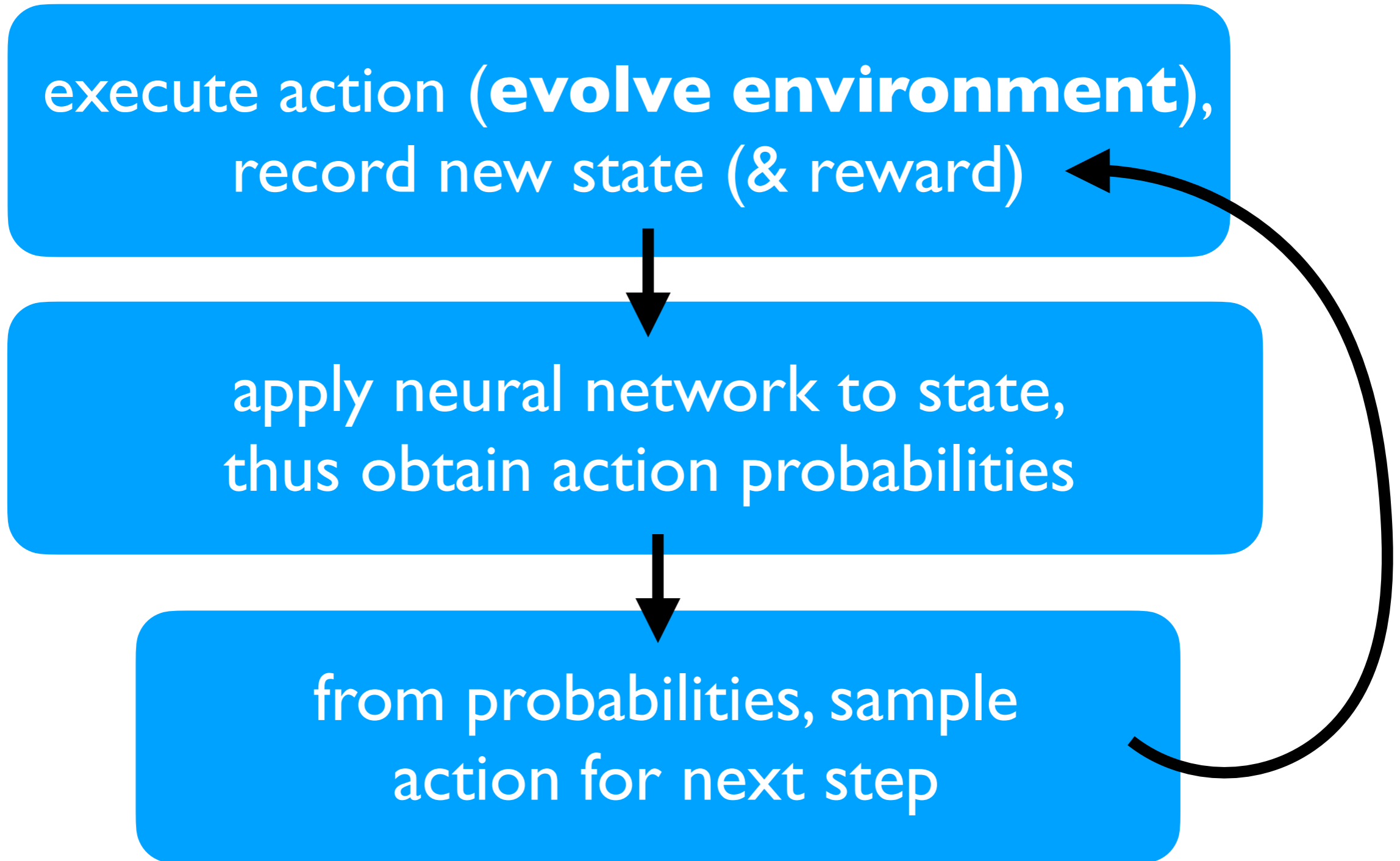
Policy gradient: all the steps

Obtain one "trajectory":

execute action (**evolve environment**),
record new state (& reward)

apply neural network to state,
thus obtain action probabilities

from probabilities, sample
action for next step



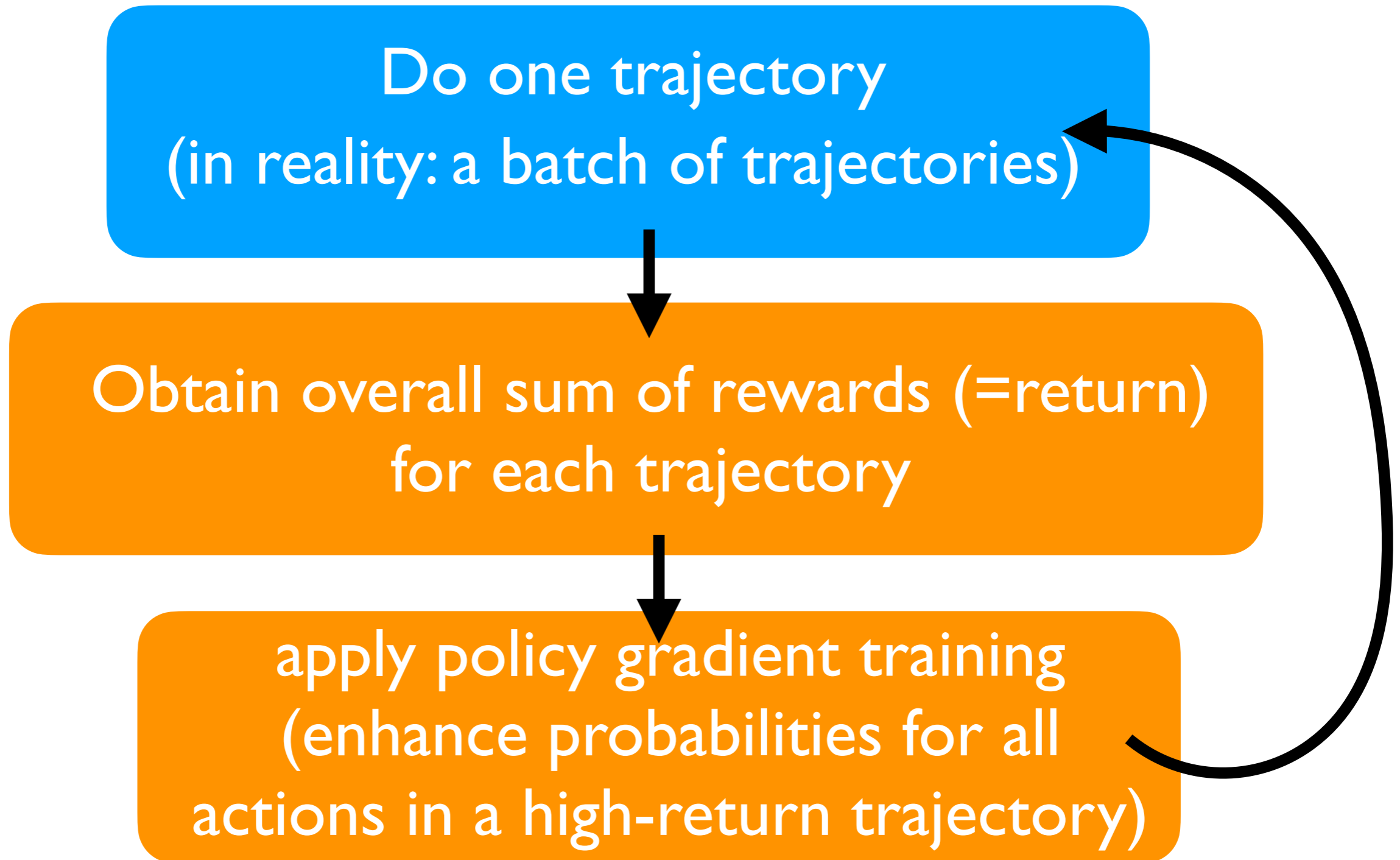
Policy gradient: all the steps

For each trajectory:

Do one trajectory
(in reality: a batch of trajectories)

Obtain overall sum of rewards (=return)
for each trajectory

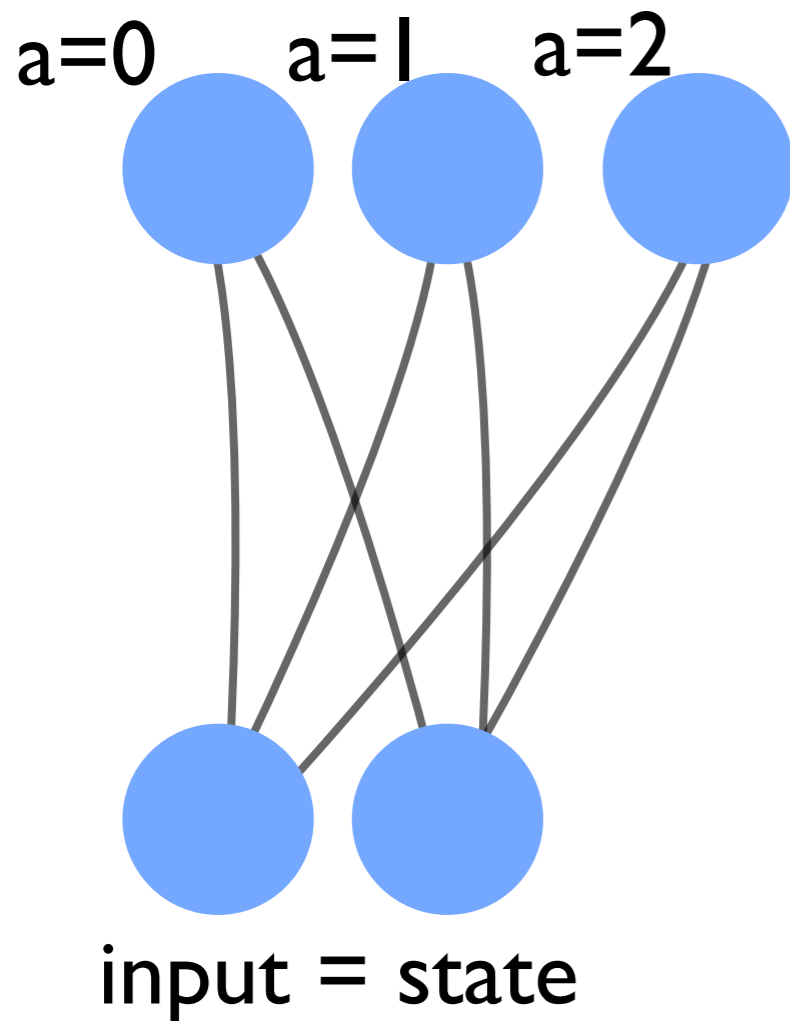
apply policy gradient training
(enhance probabilities for all
actions in a high-return trajectory)



categorical cross-entropy trick

output = action
probabilities (softmax)

$$\pi_{\theta}(a|s)$$



categorical cross-entropy

$$C = - \sum_a \overset{\text{distr. from net}}{P(a)} \ln \pi_{\theta}(a|s) \underset{\text{desired distribution}}{\quad}$$

Set

$$P(a) = R$$

for a =action that was taken

$$P(a) = 0$$

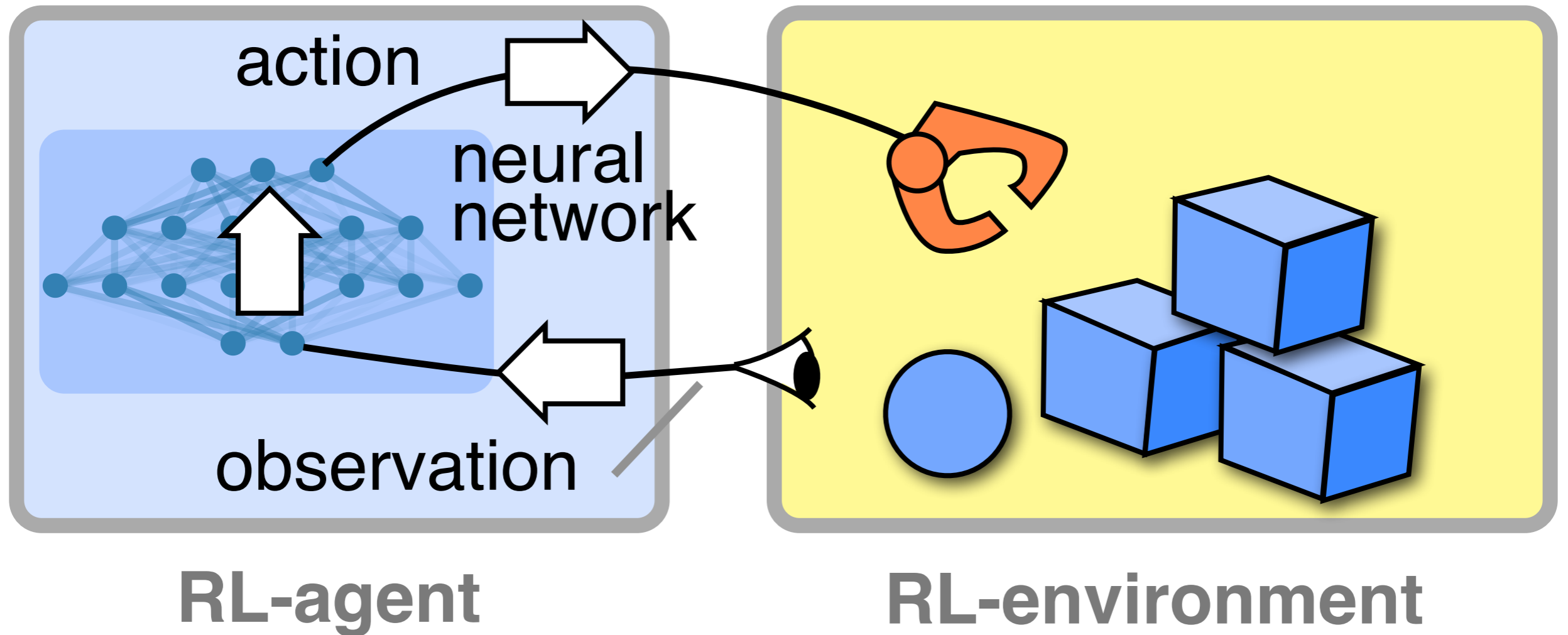
for all other actions a

$$\Delta\theta = -\eta \frac{\partial C}{\partial \theta}$$

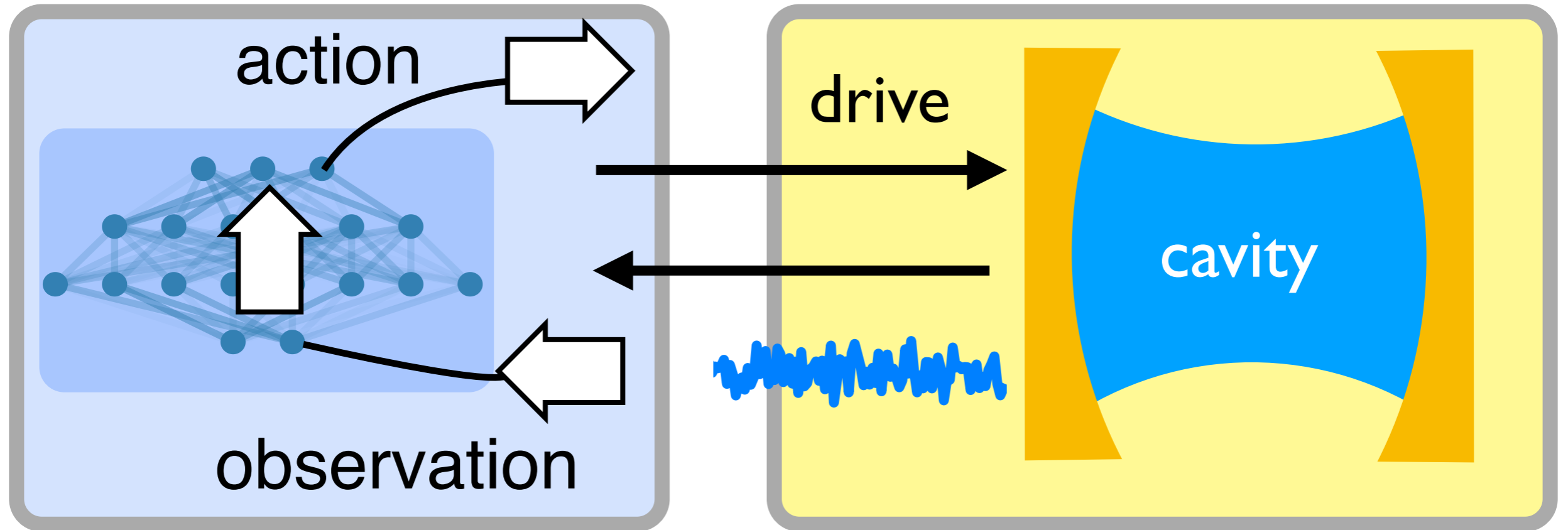
implements policy gradient

Policy gradient:
first quantum physics
example

Quantum physics example: Quantum feedback



Quantum physics example: Quantum feedback

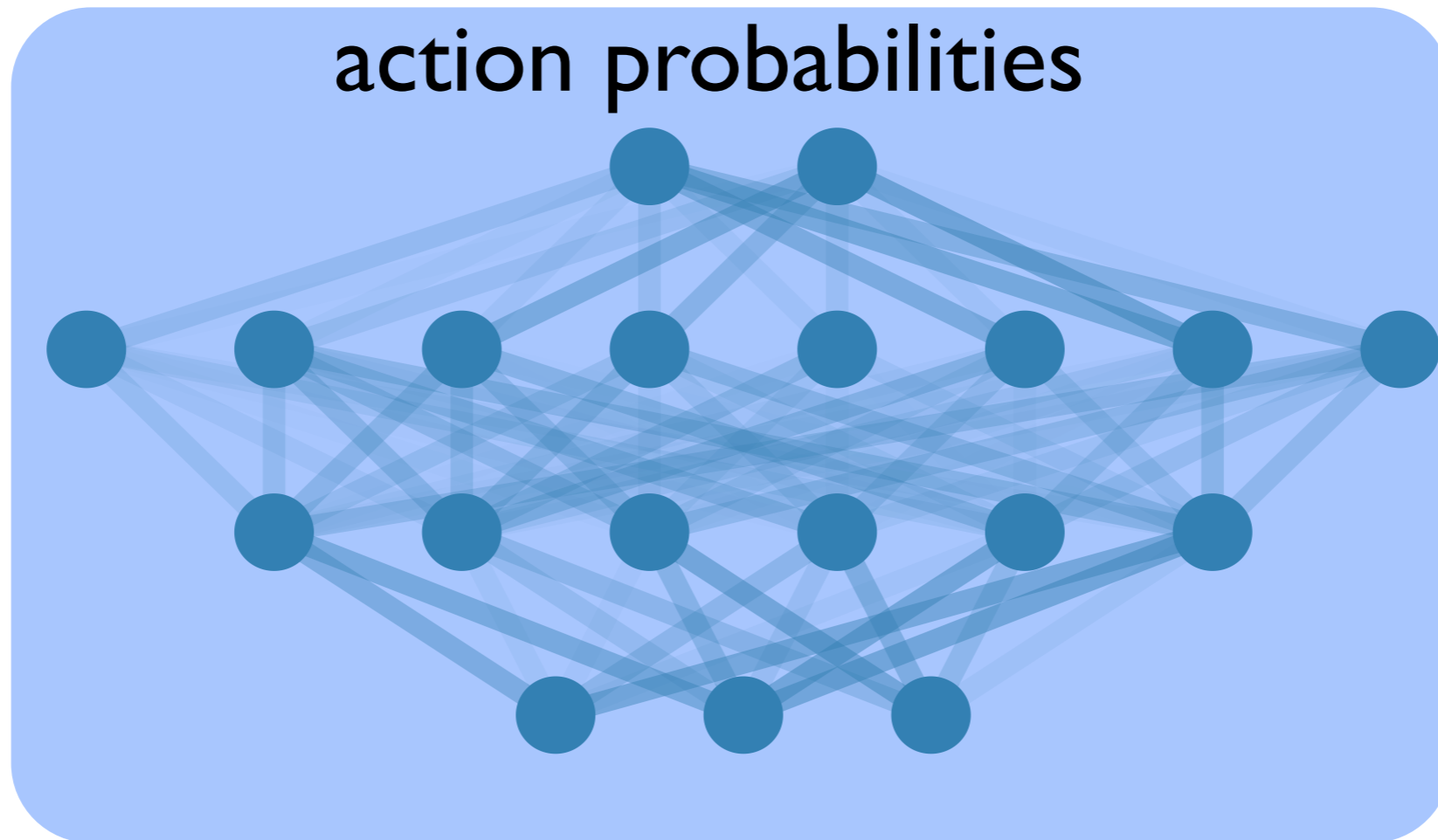


RL-agent

RL-environment

cavity, driven, with readout
goal? e.g. stabilize state

Output

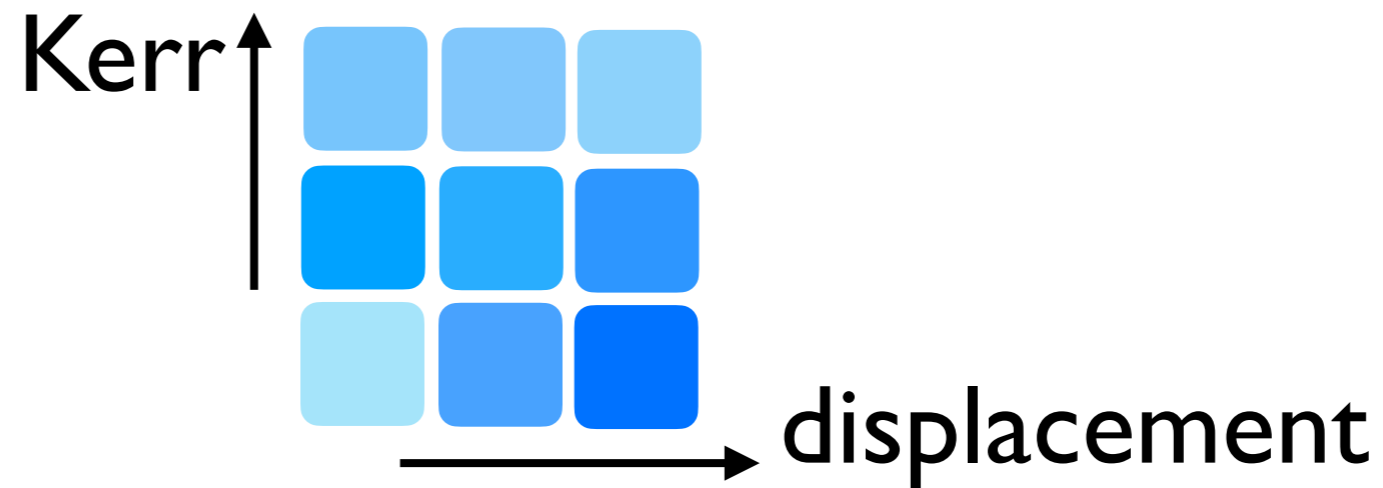


Input

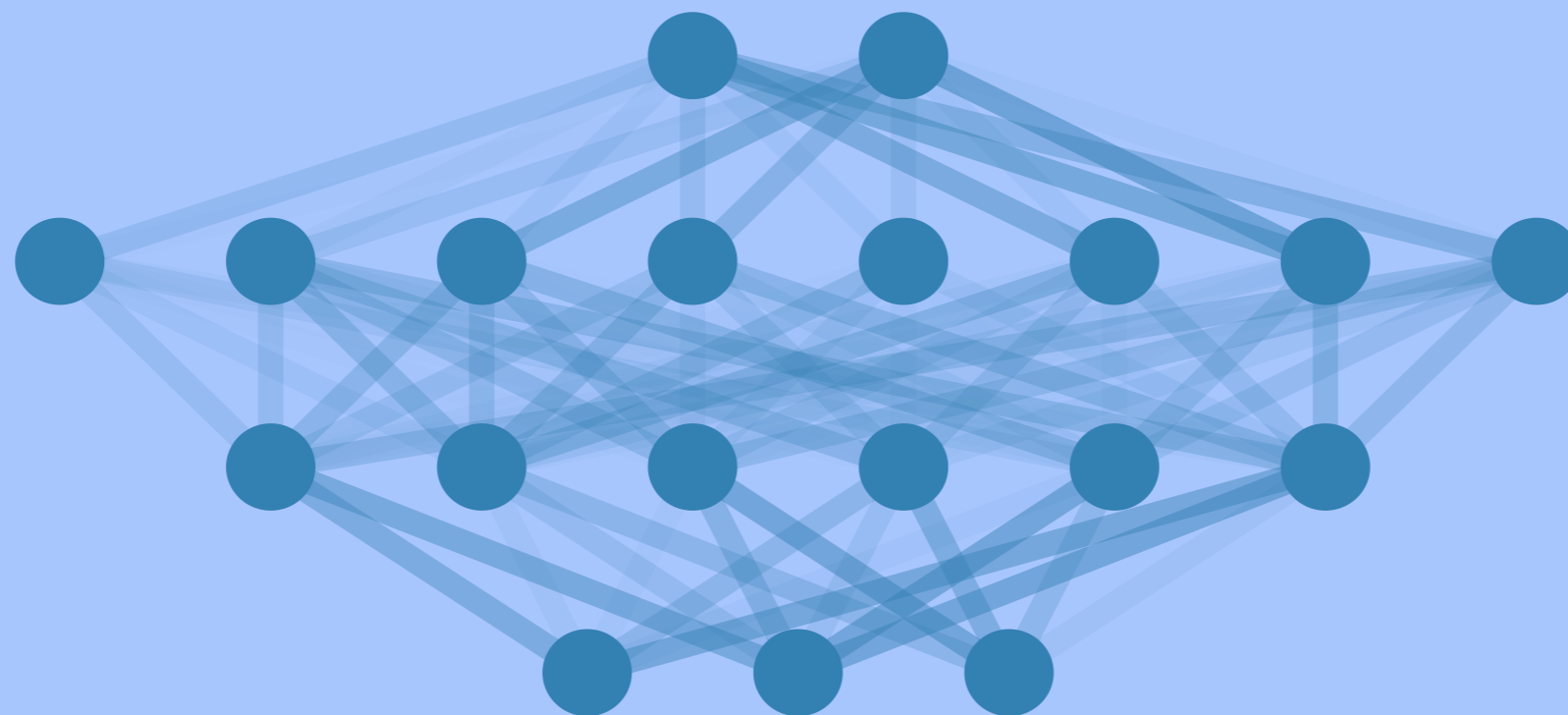


(measurement trace for last N time steps)

Output



action probabilities

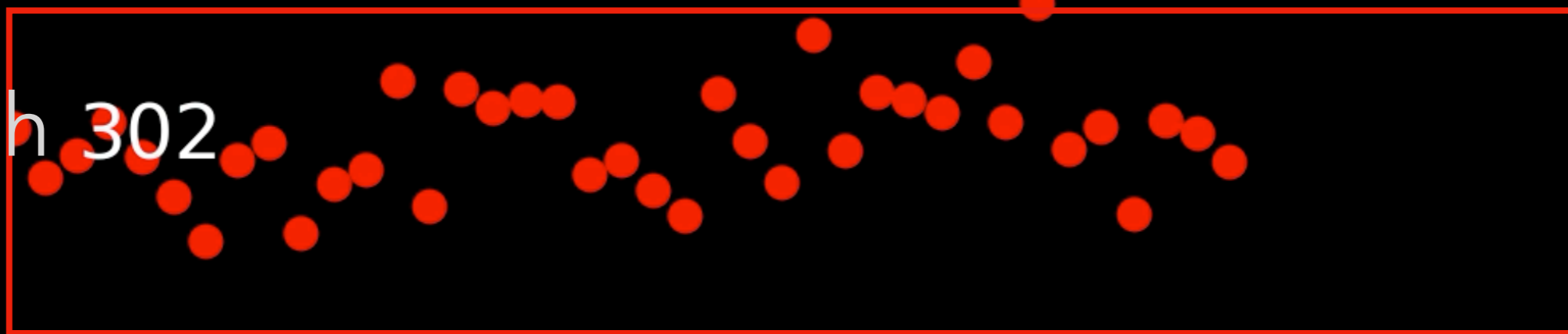


Input



(measurement trace for last N time steps)

epoch 302

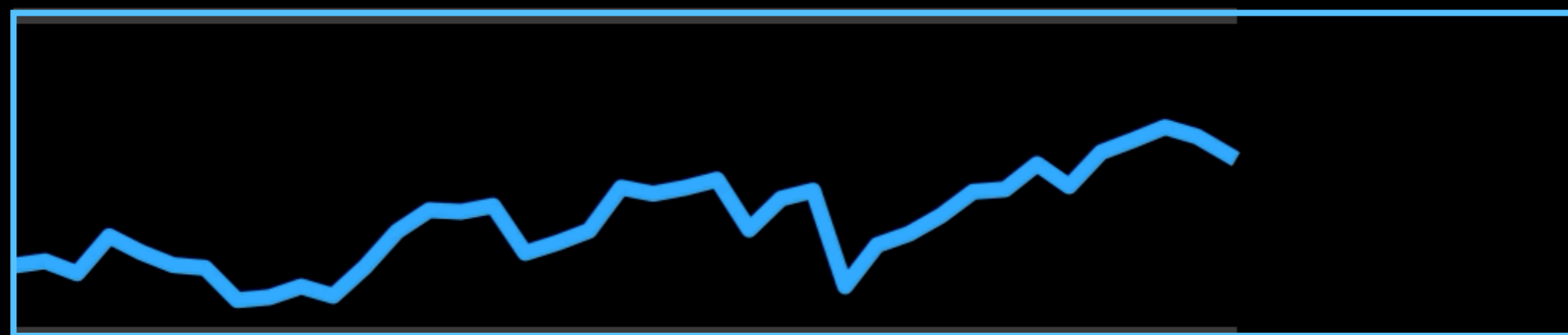


msmt trace (weak QND photon number msmt)

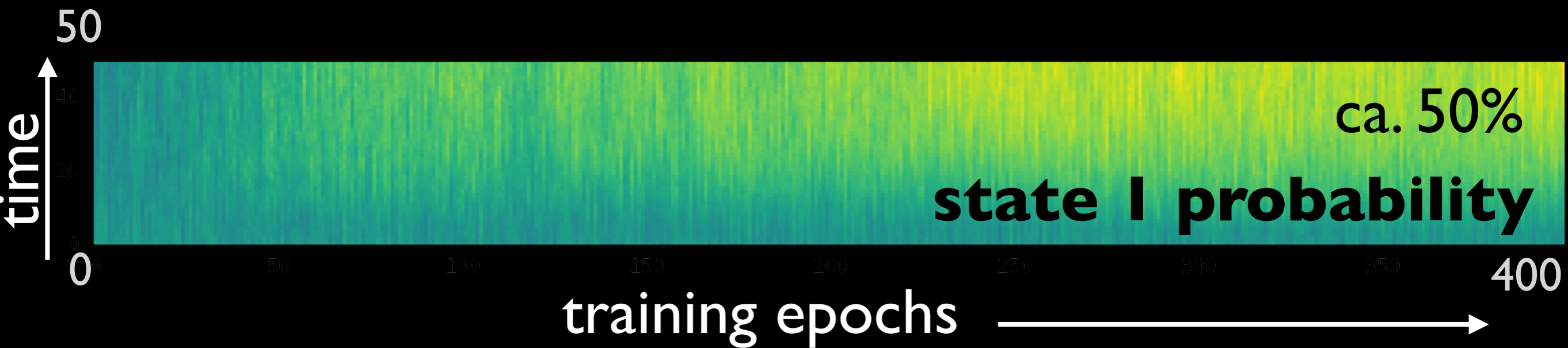


displacement drive

100%
0%

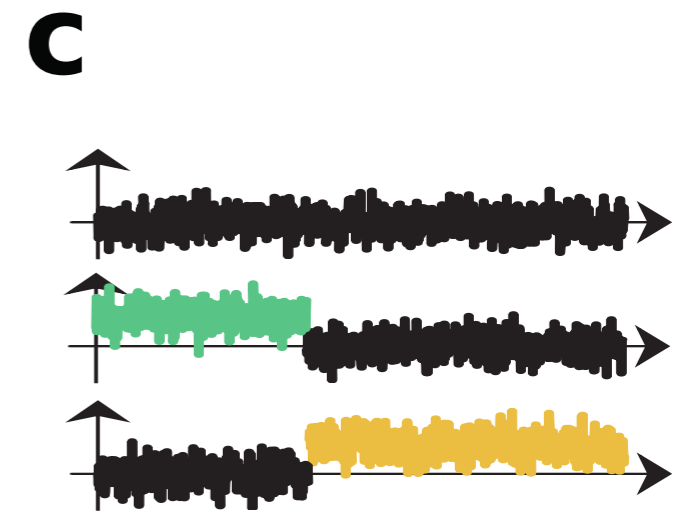
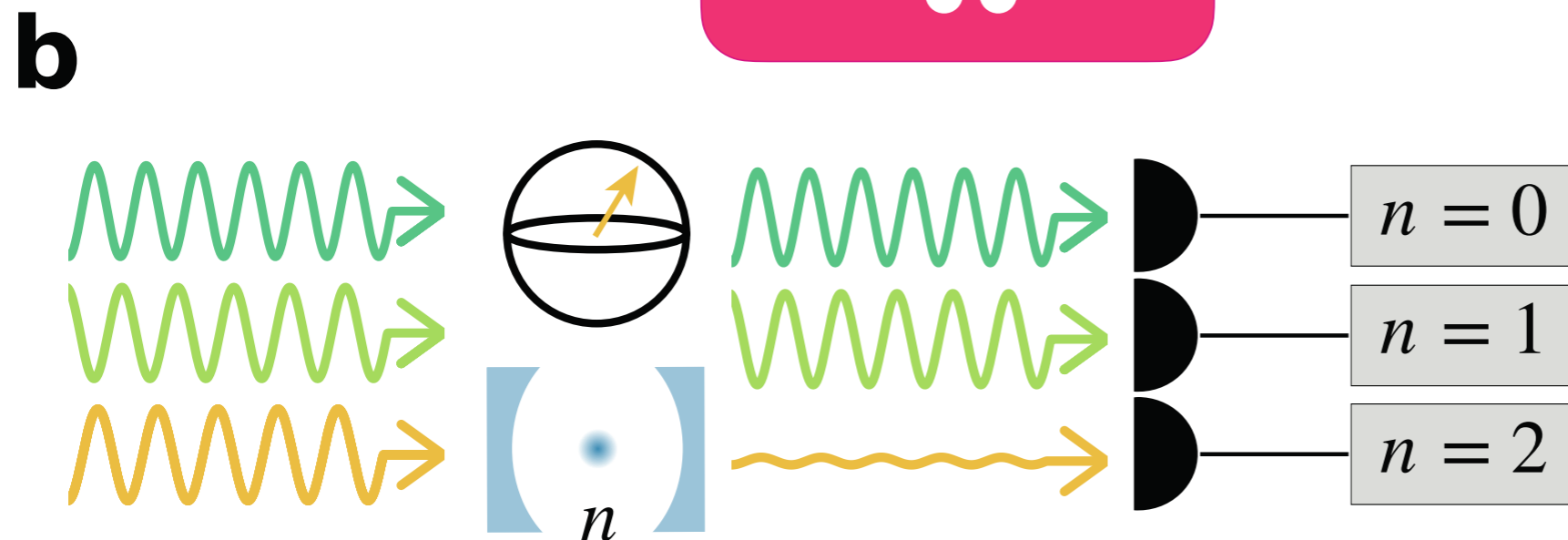
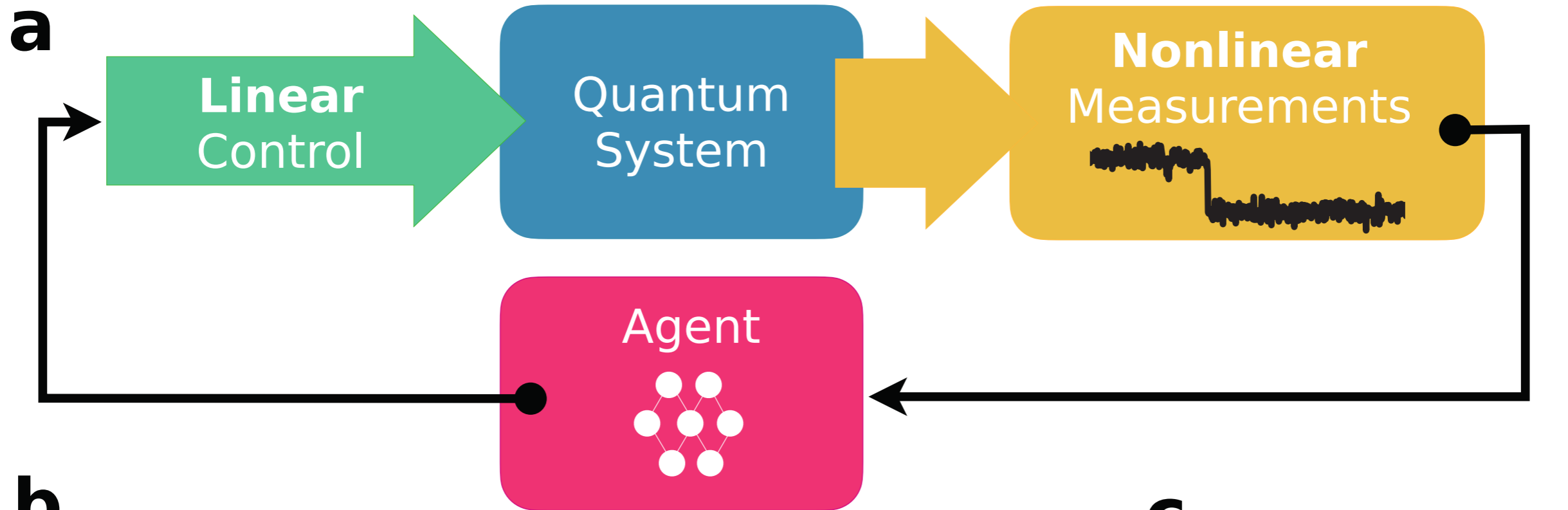


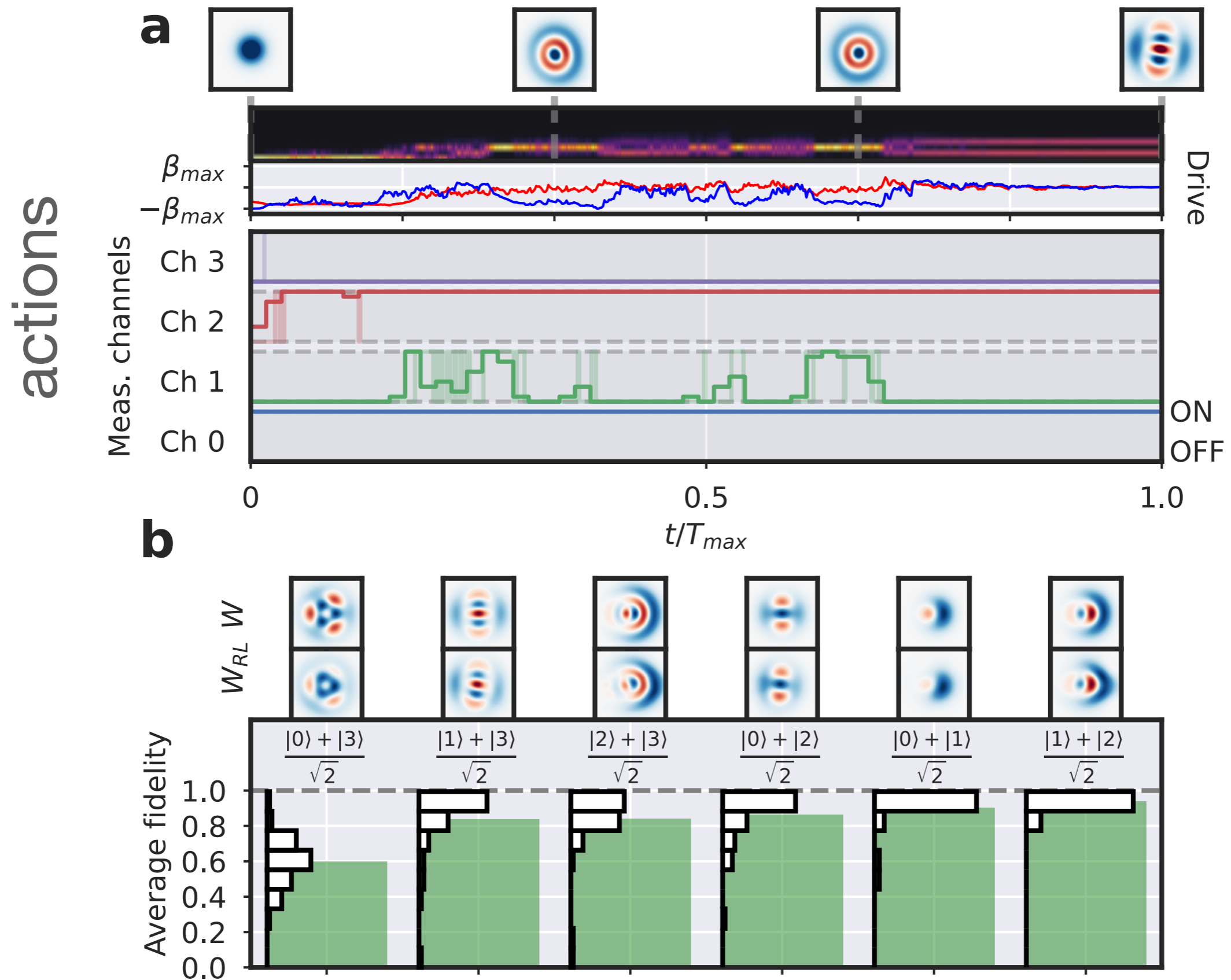
Fock state 1 probability



(100 trajectories per epoch)

Research-level example:
state preparation via
nonlinear measurements
of a cavity



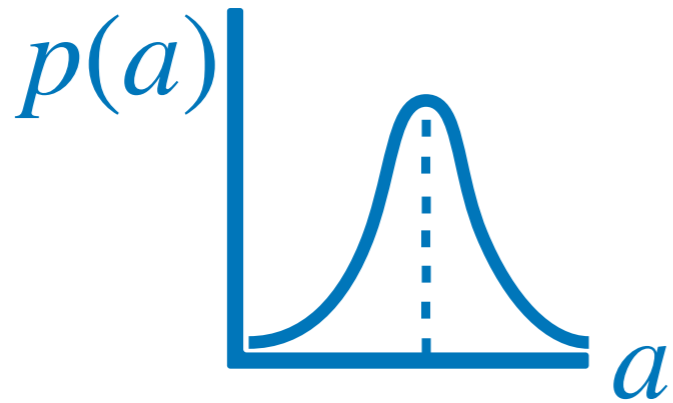


Deep Reinforcement Learning for Quantum State Preparation with Weak Nonlinear Measurements, R. Porotti, A. Essig, B. Huard, and F. Marquardt; arXiv: 2107.08816

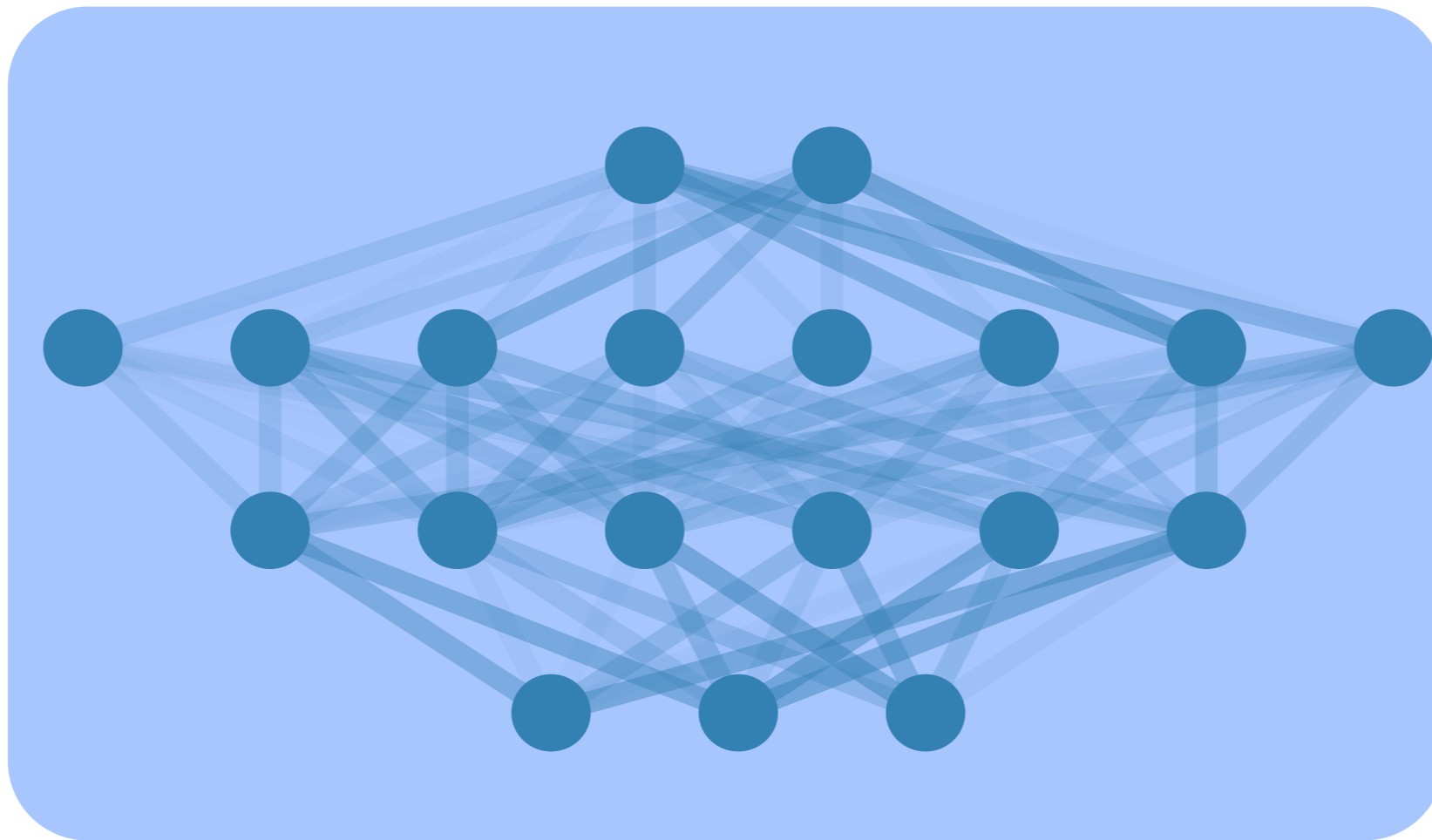
Side-note: Continuous actions

continuous action $a = \mu + \sigma\xi$

normal-distributed random variable



output: action average μ and spread σ



input

A small aside:
baselines
(reducing variance)

Policy Gradient

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E\left[R \frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta}\right]$$

Increase the probability of all action choices in the given sequence, depending on size of return R . Even if $R > 0$ always, due to normalization of probabilities this will tend to suppress the action choices in sequences with lower-than-average returns.

Abbreviation:

$$G_k = \frac{\partial \ln P_{\theta}(\tau)}{\partial \theta_k} = \sum_t \frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta_k}$$

$$\frac{\partial \bar{R}}{\partial \theta_k} = E[RG_k]$$

Policy Gradient: reward baseline

Challenge: fluctuations of estimate for return gradient can be huge. Things improve if one subtracts a constant baseline from the return.

$$\begin{aligned}\frac{\partial \bar{R}}{\partial \theta} &= \sum_t E\left[(R - b) \frac{\partial \ln \pi_{\theta}(a_t | s_t)}{\partial \theta}\right] \\ &= E[(R - b)G]\end{aligned}$$

This is the same as before. Proof:

$$E[G_k] = \sum_{\tau} P_{\theta}(\tau) \frac{\partial \ln P_{\theta}(\tau)}{\partial \theta_k} = \frac{\partial}{\partial \theta_k} \sum_{\tau} P_{\theta}(\tau) = 0$$

However, the variance of the fluctuating random variable $(R-b)G$ is different, and can be smaller (depending on the value of b)!

Note: b can become state-dependent!

Optimal baseline

Define $X_k = (R - b_k)G_k$

Minimize $\text{Var}[X_k] = E[X_k^2] - E[X_k]^2 = \min$

$$\frac{\partial \text{Var}[X_k]}{\partial b_k} = 0$$

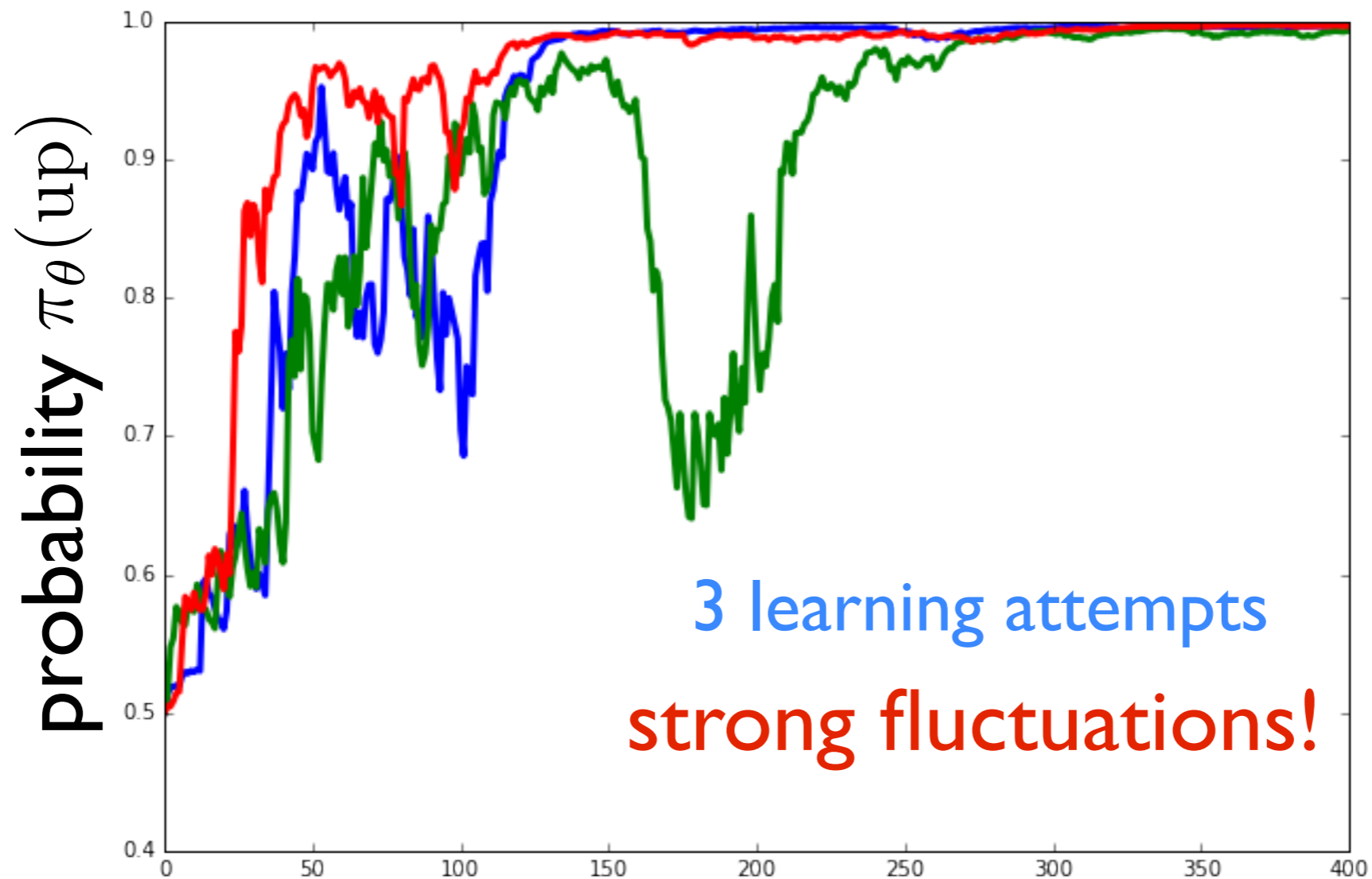
Optimal baseline:

$$b_k = \frac{E[G_k^2 R]}{E[G_k^2]}$$

$$G_k = \frac{\partial \ln P_\theta(\tau)}{\partial \theta_k}$$

$$\Delta \theta_k = -\eta E[G_k (R - b_k)]$$

random walker toy example



trajectory (=training episode)

(This plot for $N=100$ time steps in a trajectory; $\text{eta}=0.001$)

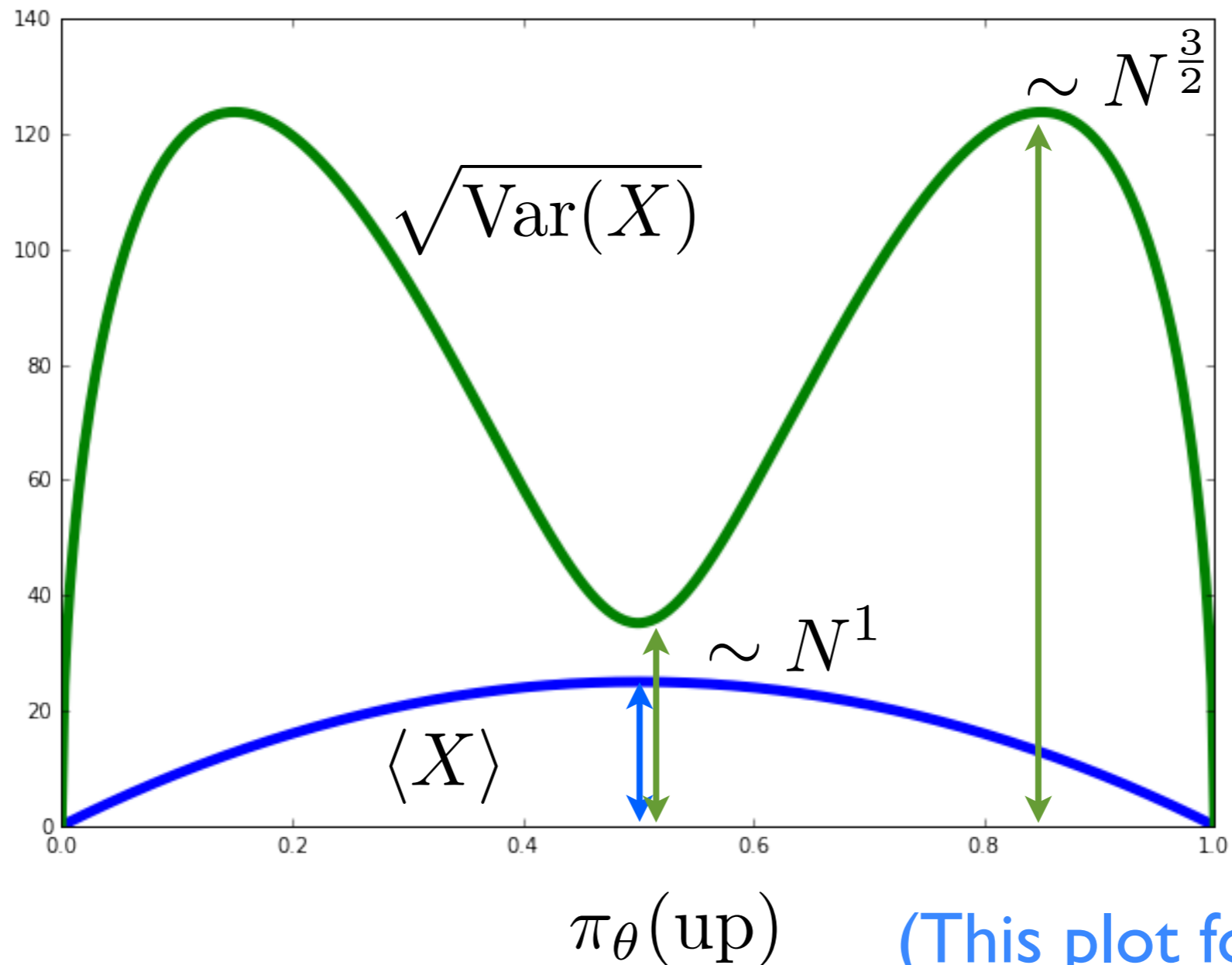
Spread of the update step

$$Y = N_{\text{up}} - \bar{N}_{\text{up}} \quad c = \bar{N}_{\text{up}} - N/2$$

$$X = (Y + c)Y$$

(Note: to get $\text{Var } X$, we need central moments of binomial distribution up to 4th moment)

$X = \text{update}$
(except prefactor of 2)



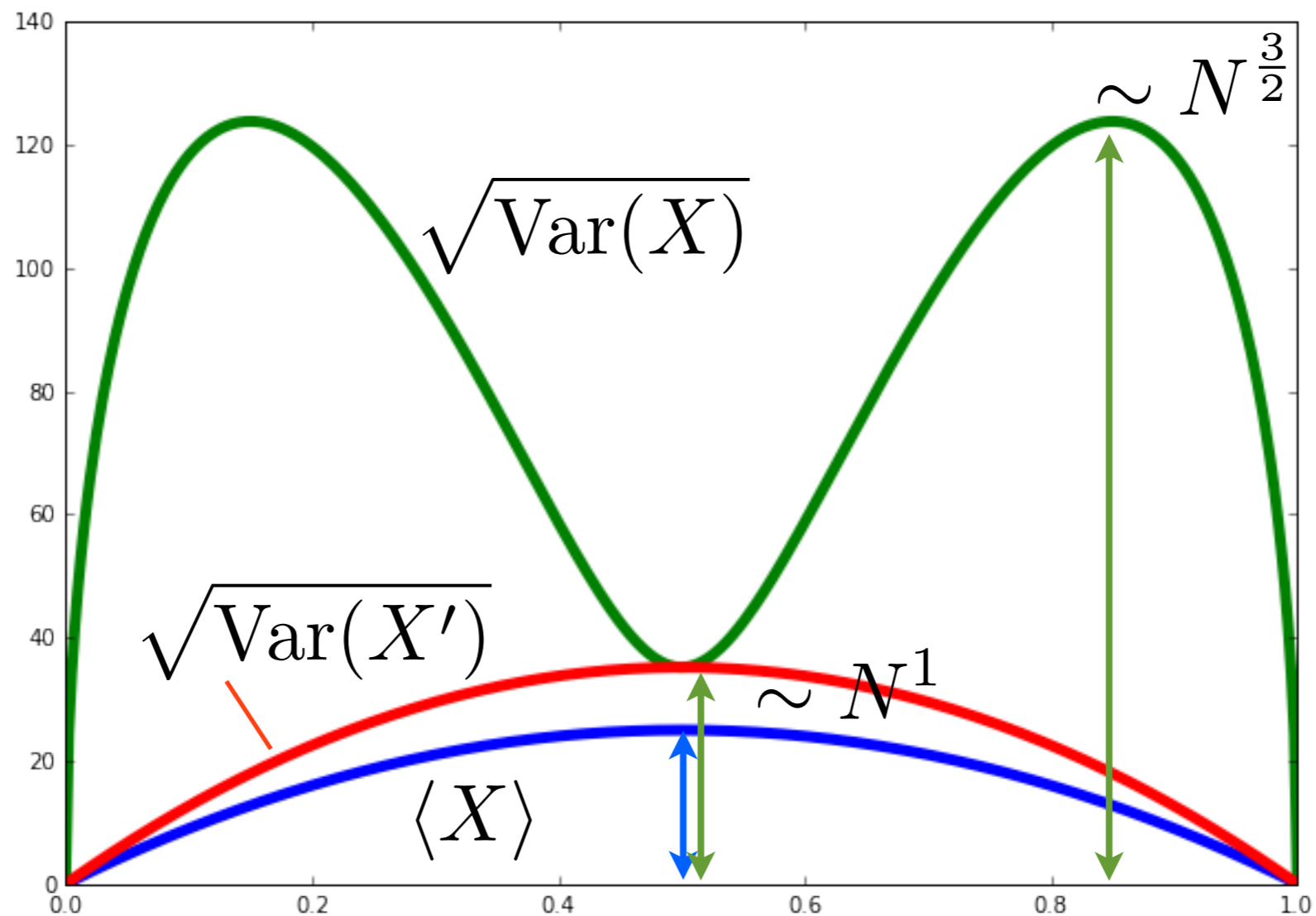
(This plot for $N=100$)

Optimal baseline suppresses spread!

$$Y = N_{\text{up}} - \bar{N}_{\text{up}} \quad c = \bar{N}_{\text{up}} - N/2 \quad X = (Y + c)Y$$

with optimal baseline:

$$X' = (Y + c - b)Y \quad b = \frac{\langle Y^2(Y + c) \rangle}{\langle Y^2 \rangle}$$



$\pi_{\theta}(\text{up})$ (This plot for $N=100$)

Example:
AlphaGo

AlphaGo



Among the major board games, “Go” was not yet played on a superhuman level by any program (very large state space on a 19x19 board!)

alpha-Go beat the world’s best player in 2017

AlphaGo

First: try to learn from human expert players

sampled state-action pairs (s, a) , using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s

$$\Delta\sigma \propto \frac{\partial \log p_{\sigma}(a|s)}{\partial \sigma}$$

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server. The net-

Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016

AlphaGo

Second: use policy gradient RL on games played against previous versions of the program

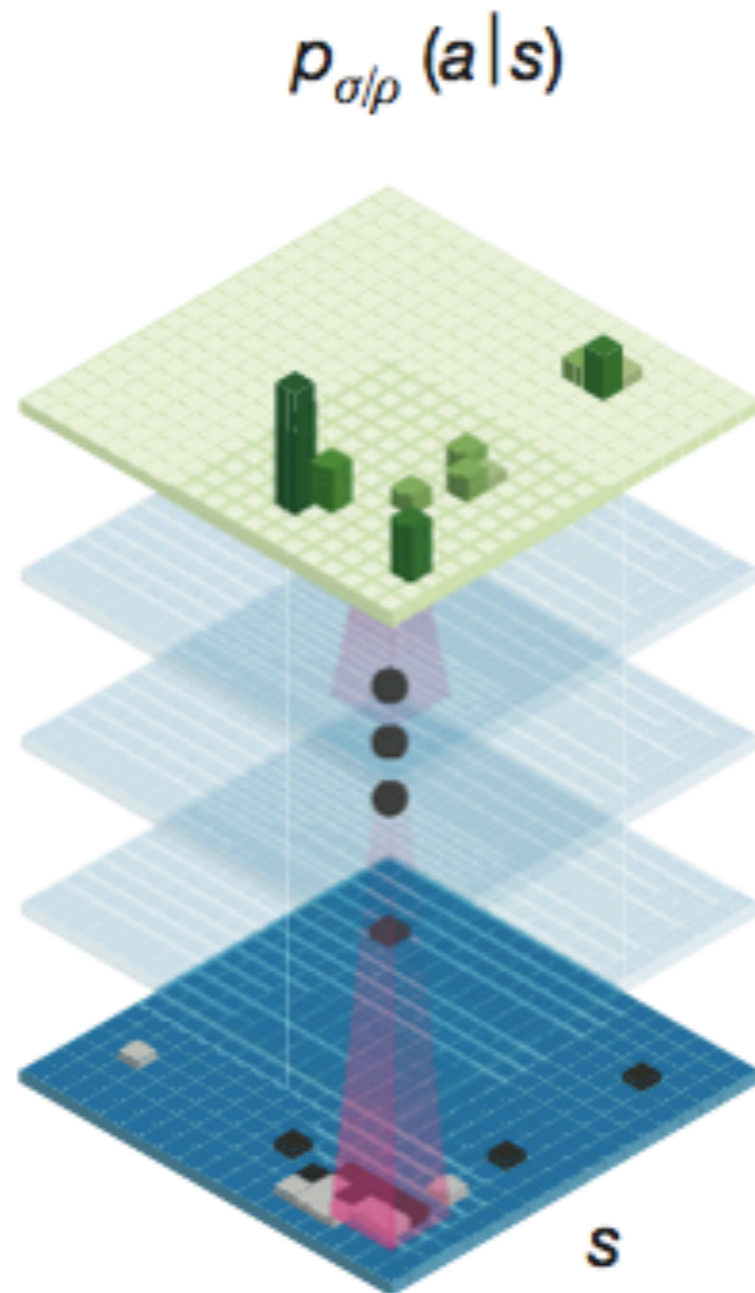
to the current policy. We use a reward function $r(s)$ that is zero for all non-terminal time steps $t < T$. The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step t : $+1$ for winning and -1 for losing. Weights are then updated at each time step t by stochastic gradient ascent in the direction that maximizes expected outcome²⁵

$$\Delta \rho \propto \frac{\partial \log p_{\rho}(a_t | s_t)}{\partial \rho} z_t$$

Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016

AlphaGo

Policy network

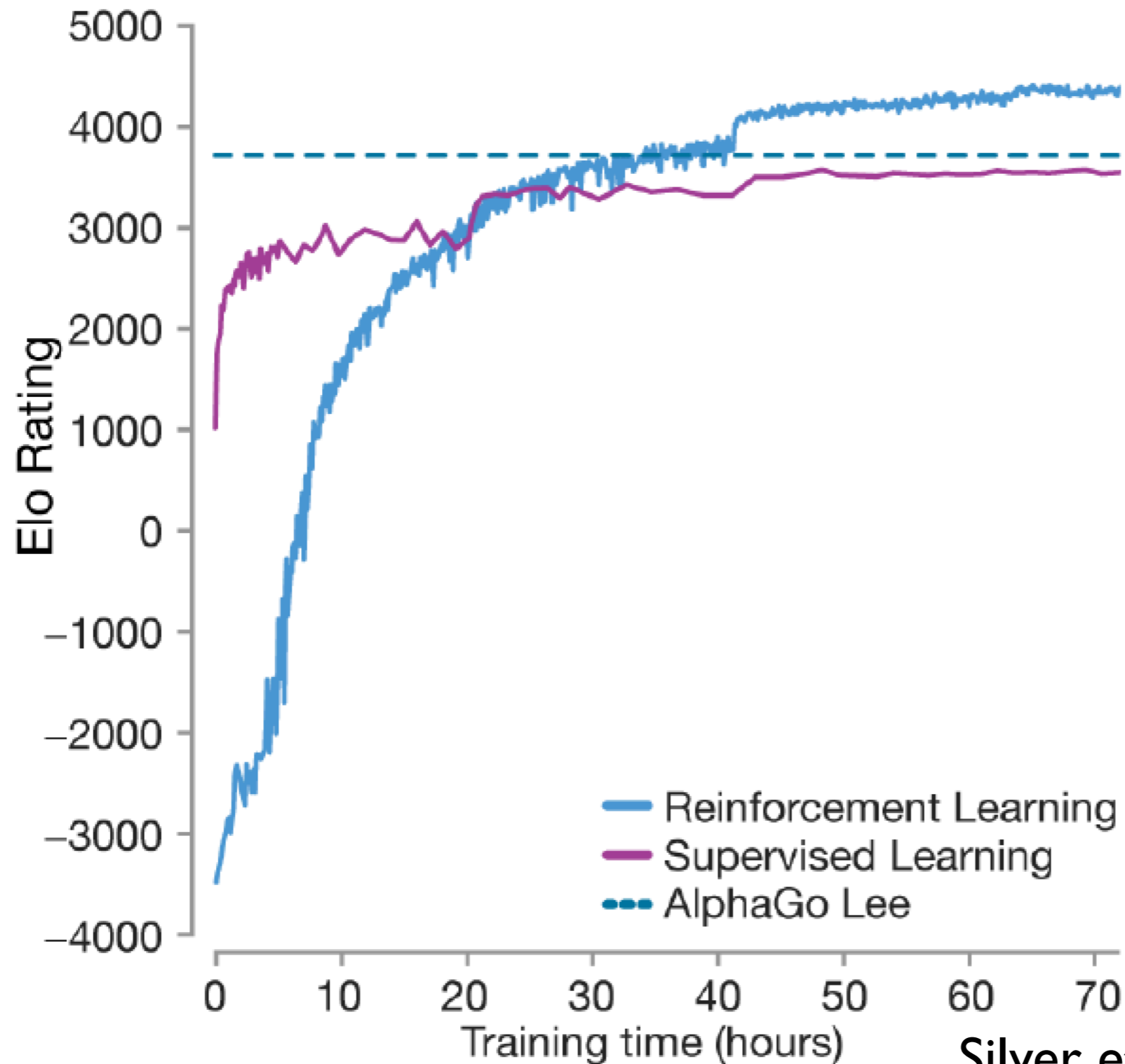


*Note: beyond policy-gradient type methods, this also includes another algorithm, called Monte Carlo Tree Search

Silver et al., "Mastering the game of Go with deep neural networks and tree search" (Google Deepmind team), Nature, January 2016

AlphaGoZero

No training on human expert knowledge
– eventually becomes even better!



Silver et al, Nature 2017

AlphaGoZero

Ke Jie stated that "After humanity spent thousands of years improving our tactics, computers tell us that humans are completely wrong... I would go as far as to say not a single human has touched the edge of the truth of Go."

Q-learning

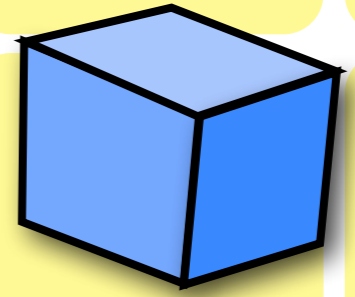
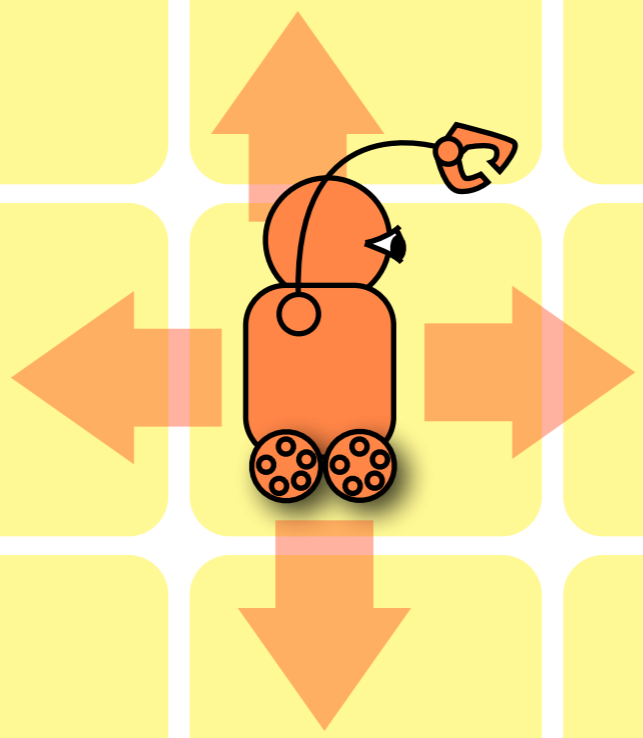
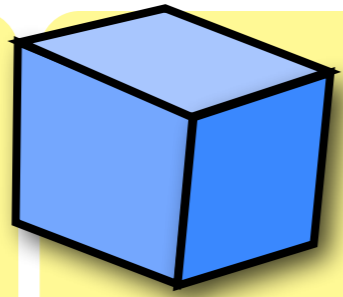
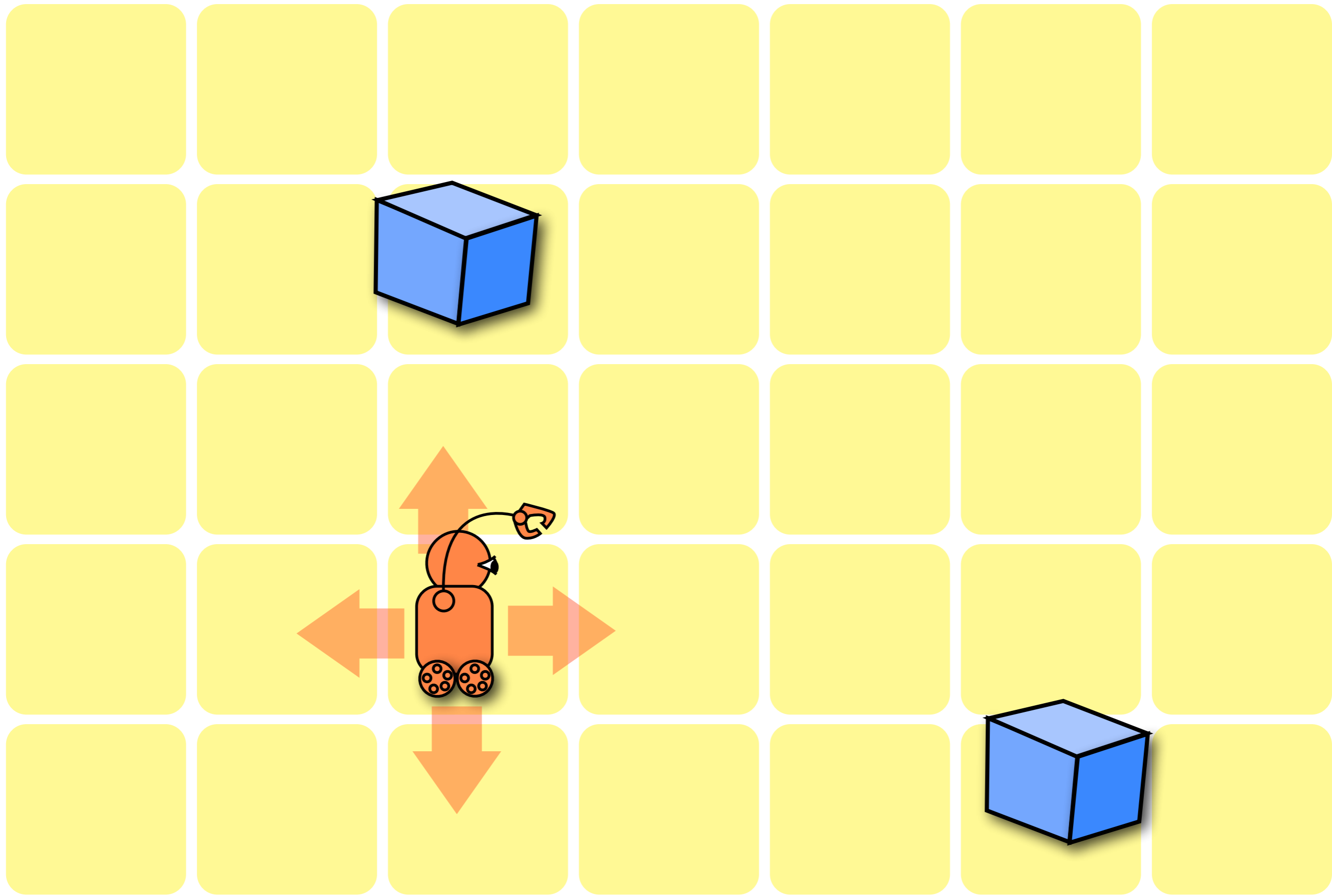
Q-learning

An alternative to the policy gradient approach

Introduce a quality function $Q(\mathbf{s}, \mathbf{a})$ that predicts the expected future return for a given state s and a given action a .

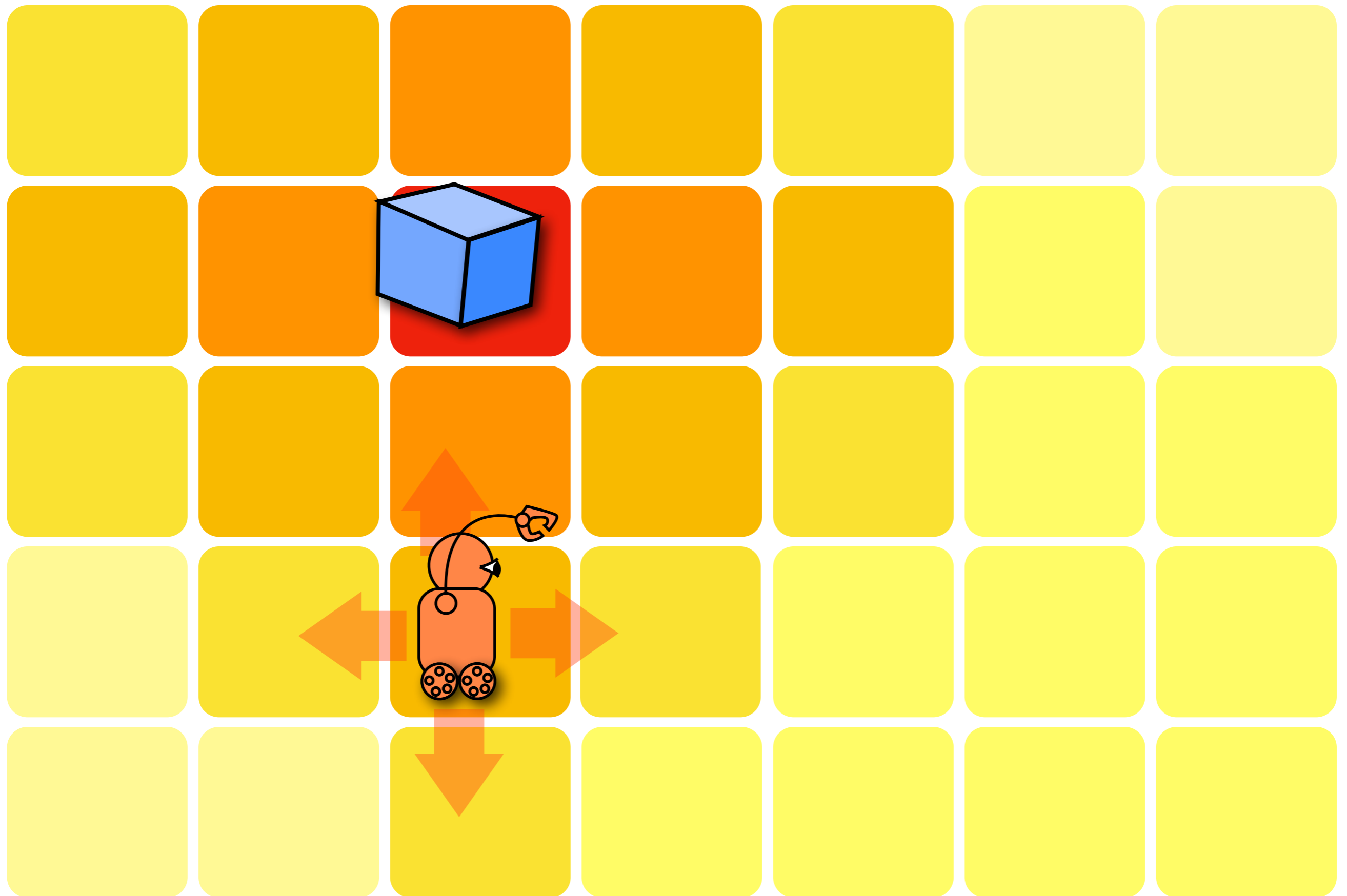
Deterministic policy: just select the action with the largest Q !

Watkins and Dayan 1992



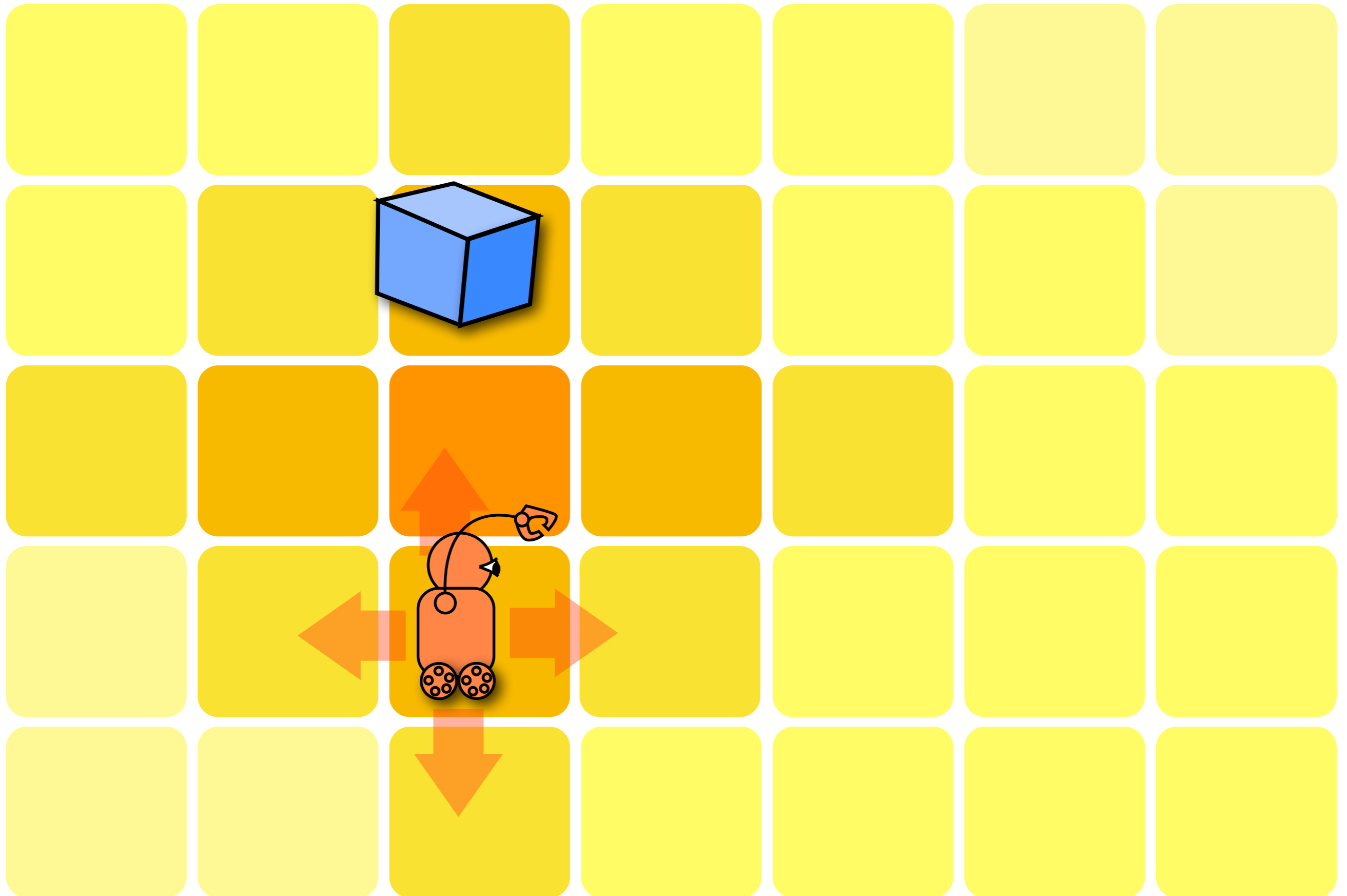
"value" of a state as color

$$V(s) = E[R | s]$$



"quality" of the action "going up"

$$Q(s, a) = E[R | s, a]$$



Q-learning

Introduce a quality function Q that predicts the future return for a given state s and a given action a . **Deterministic policy**: just select the action with the largest Q !

$$Q(s_t, a_t) = E[R_t | s_t, a_t] \quad (\text{assuming future steps to follow the policy!})$$

“Discounted”
future return:

$$R_t = \sum_{t'=t}^T r_{t'} \gamma^{t'-t}$$

Reward at time step t : r_t

depends on state
and action at time t

Discount factor: $0 < \gamma \leq 1$

learning somewhat
easier for smaller
factor (short
memory times)

Note: The ‘value’ of a state is $V(s) = \max_a Q(s, a)$

How do we obtain Q ?

Q-learning: Update rule

$$Q(s_t, a_t) = E[R_t | s_t, a_t]$$

$$R_t = \sum_{t'=t}^T r_{t'} \gamma^{t'-t} = r_t + \gamma R_{t+1}$$

Bellmann equation:

$$Q(s_t, a_t) = E[r_t + \gamma \max_a Q(s_{t+1}, a) | s_t, a_t]$$

future return R_{t+1}
using Q-policy

Q-learning: Update rule

Bellmann equation:

$$Q(s_t, a_t) = E[r_t + \gamma \max_a Q(s_{t+1}, a) | s_t, a_t]$$

In practice, we do not know the Q function yet, so we cannot directly use the Bellmann equation. However, the following update rule has the correct Q function as a fixed point:

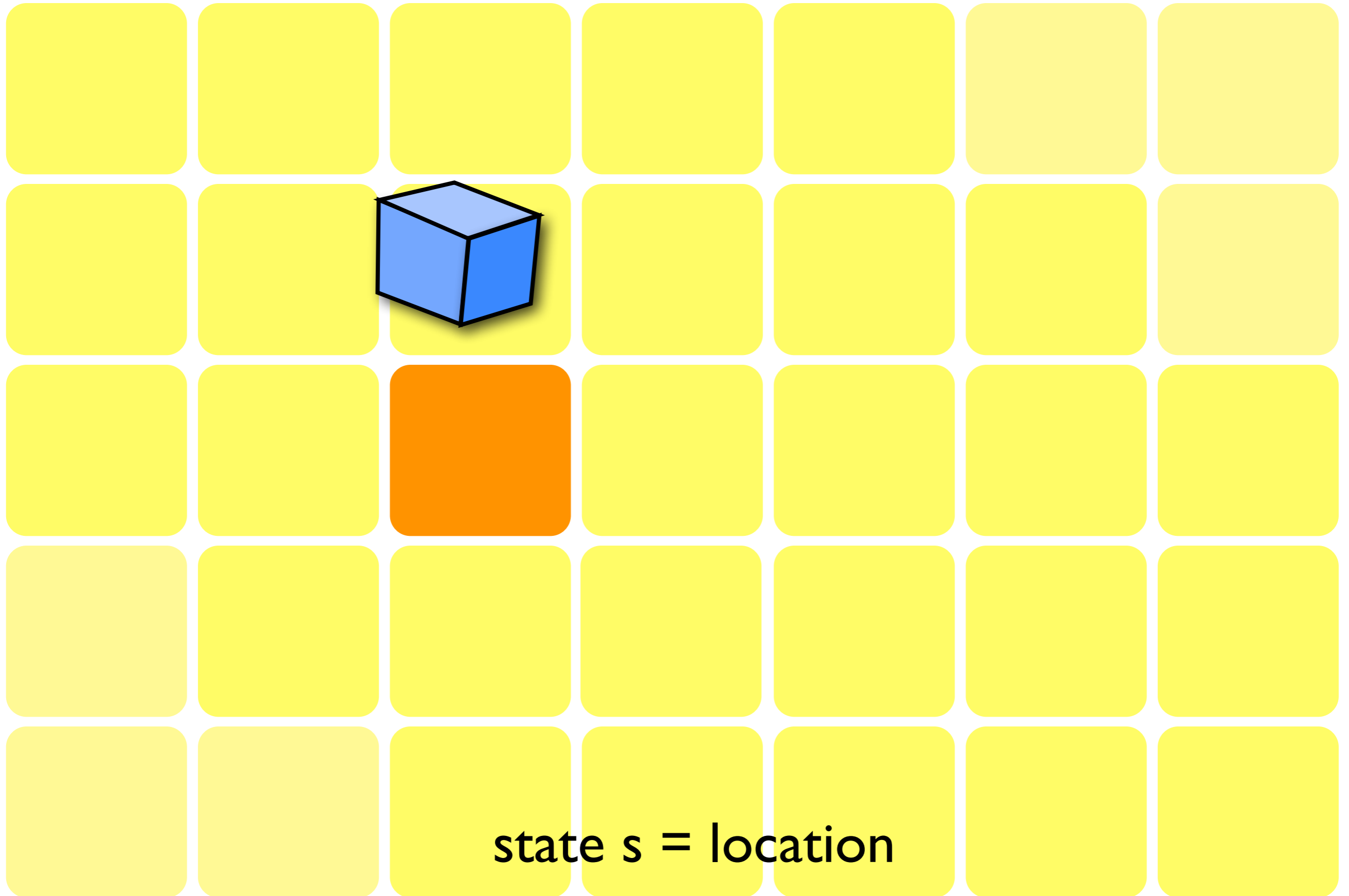
$$Q^{\text{new}}(s_t, a_t) = Q^{\text{old}}(s_t, a_t) + \alpha (r_t + \gamma \max_a Q^{\text{old}}(s_{t+1}, a) - Q^{\text{old}}(s_t, a_t))$$

small (< 1) update factor

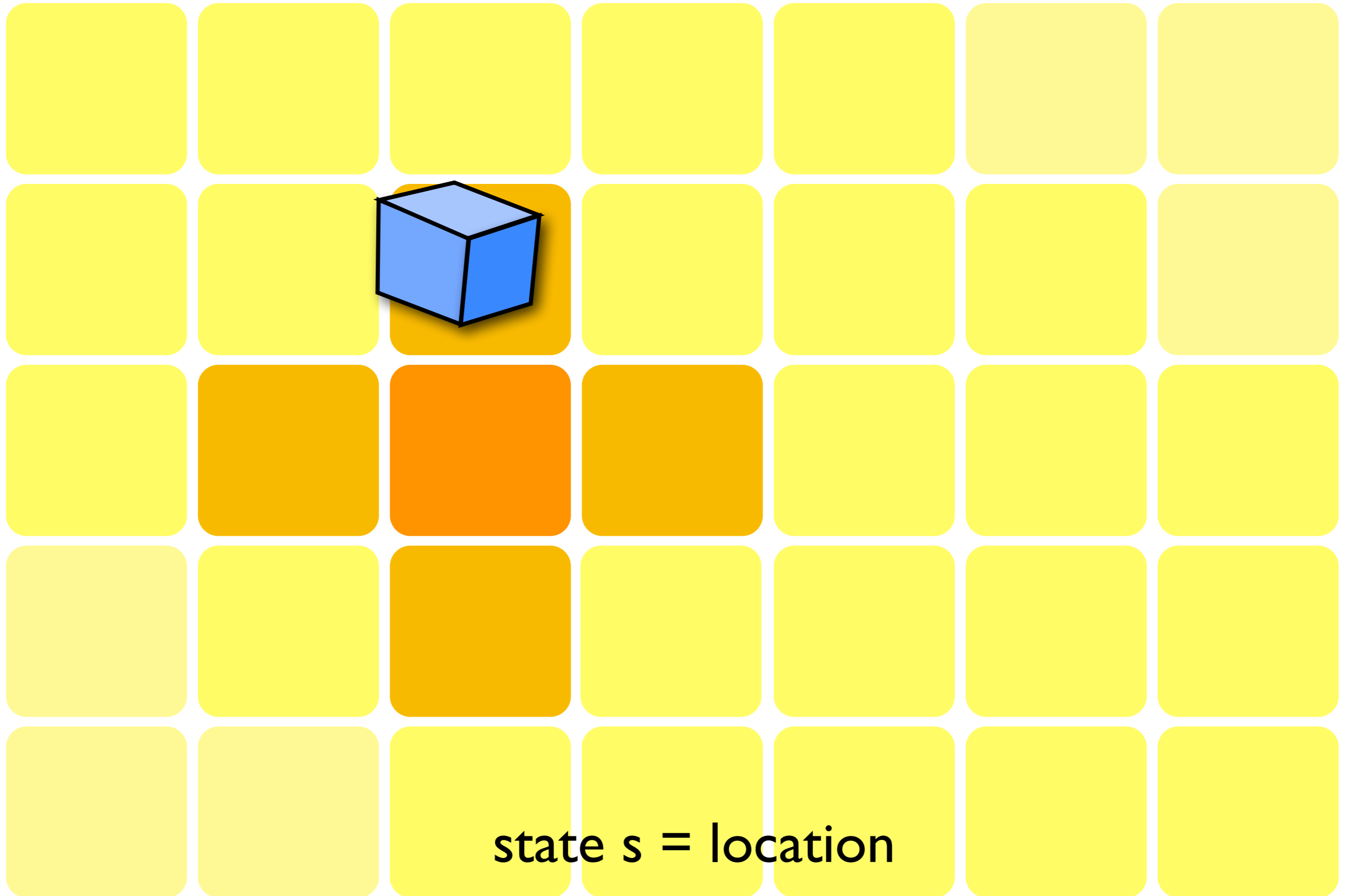
will be zero, once we have converged to the correct Q

If we use a neural network to calculate Q, we have to train it to yield the “new” value in each step.

"quality" $Q(s,a)$ of the action "going up" as color

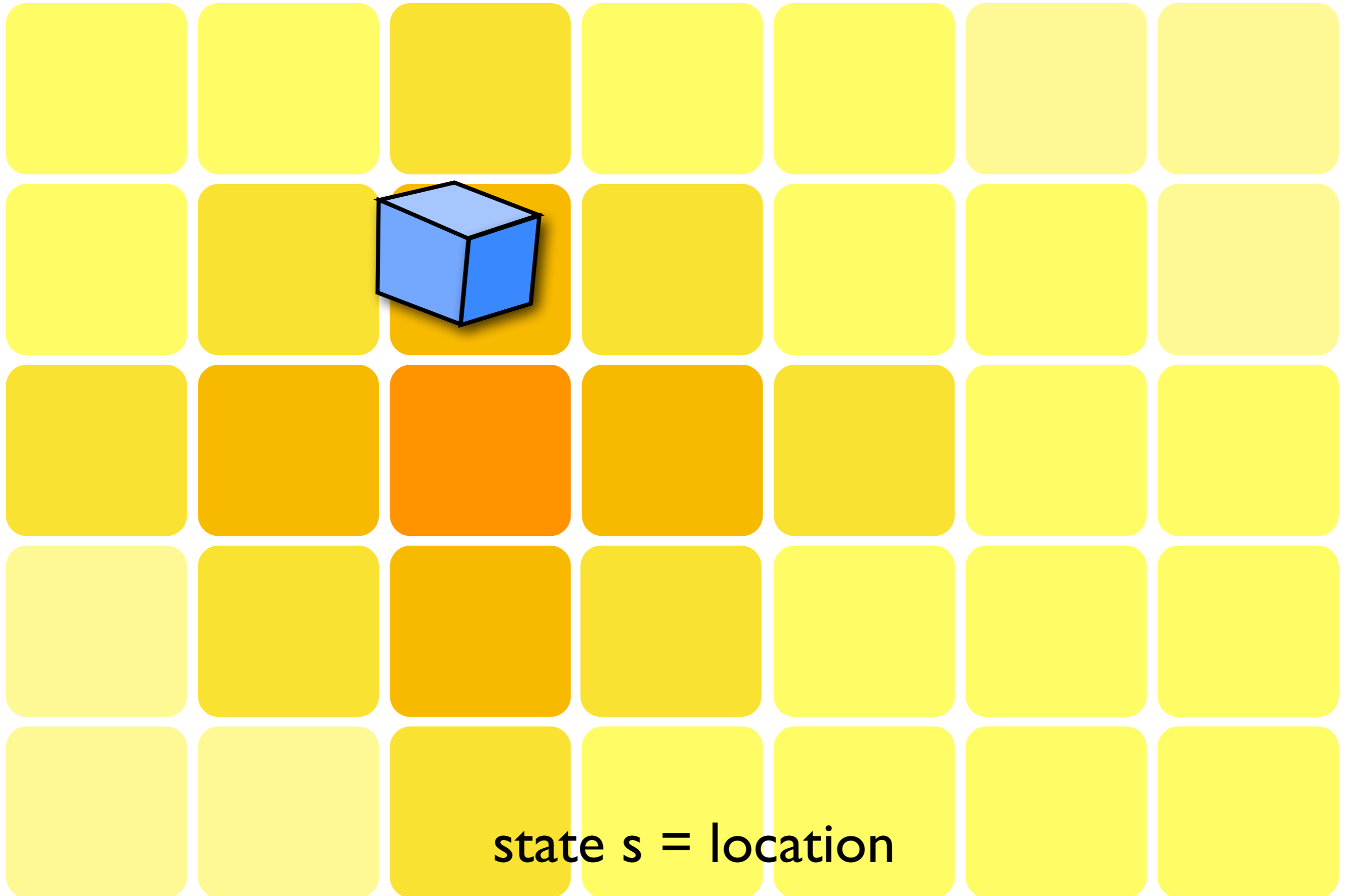


"quality" $Q(s,a)$ of the action "going up" as color



state $s = \text{location}$

"quality" $Q(s,a)$ of the action "going up" as color



Q-learning: Exploration

Initially, Q is arbitrary. It will be bad to follow this Q all the time. Therefore, introduce probability ϵ of random action (“exploration”)!

Follow Q : “**exploitation**”

Do something random (new): “**exploration**”

“ ϵ -greedy”

Reduce this randomness later!

Q-learning: Experience replay

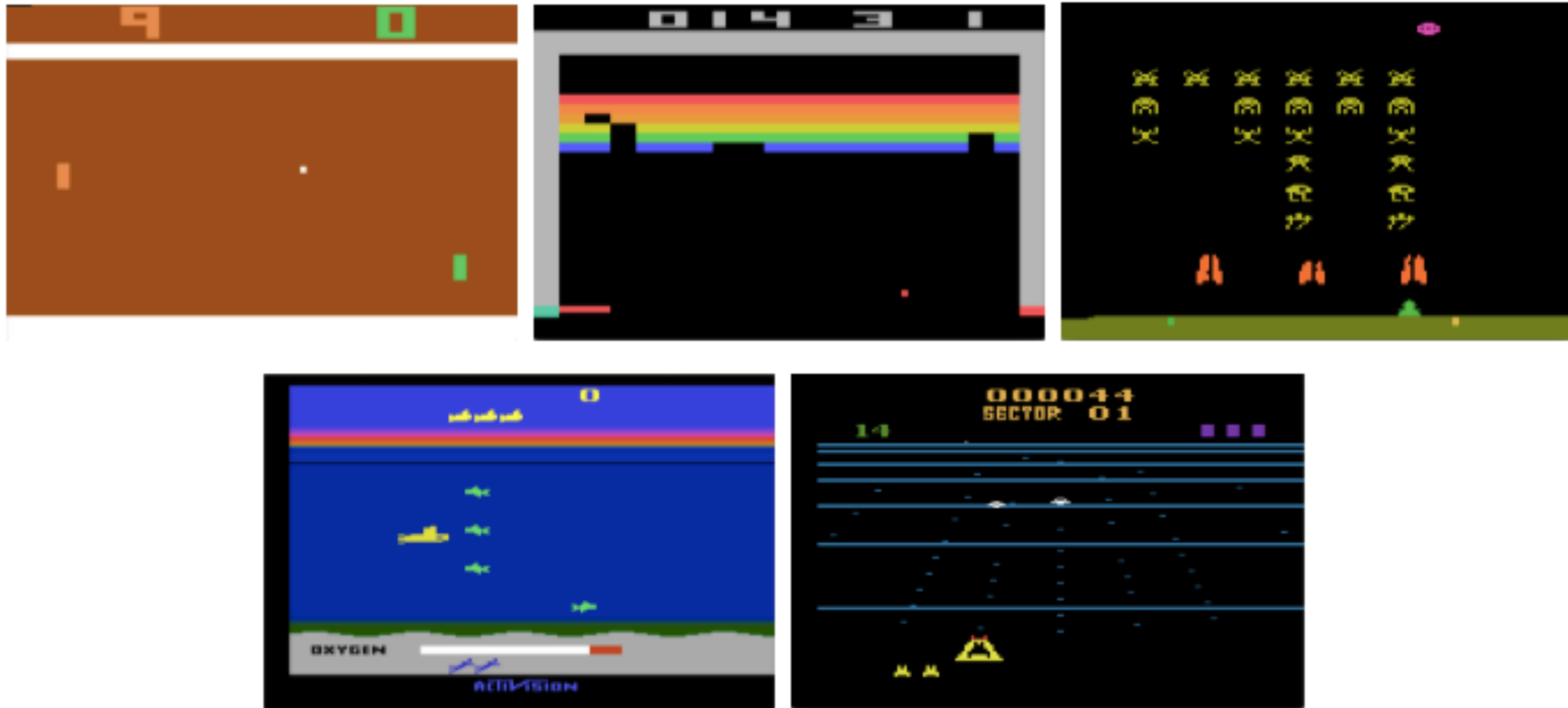
Store states and actions from past trajectories, revisit them, use them to update Q (it has changed in the meantime!)

[side-note: it is not possible to re-use states&actions in policy gradient (straightforwardly), since these need to be sampled according to the current policy, not some old policy]

Q-learning: Example (Atari games)

Example: Learning to play Atari Video Games

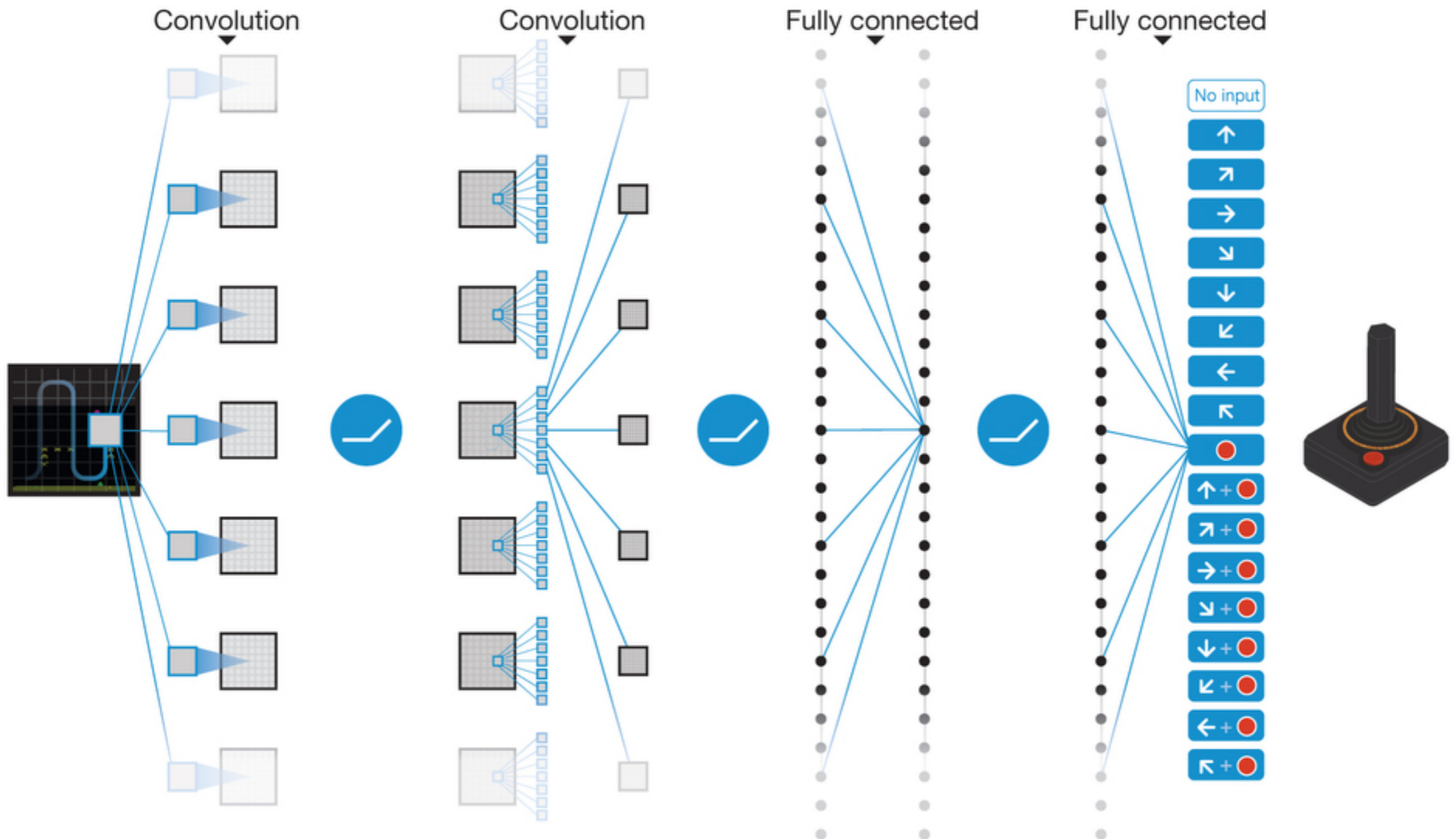
“Human-level control through deep reinforcement learning”, Mnih et al., Nature, February 2015



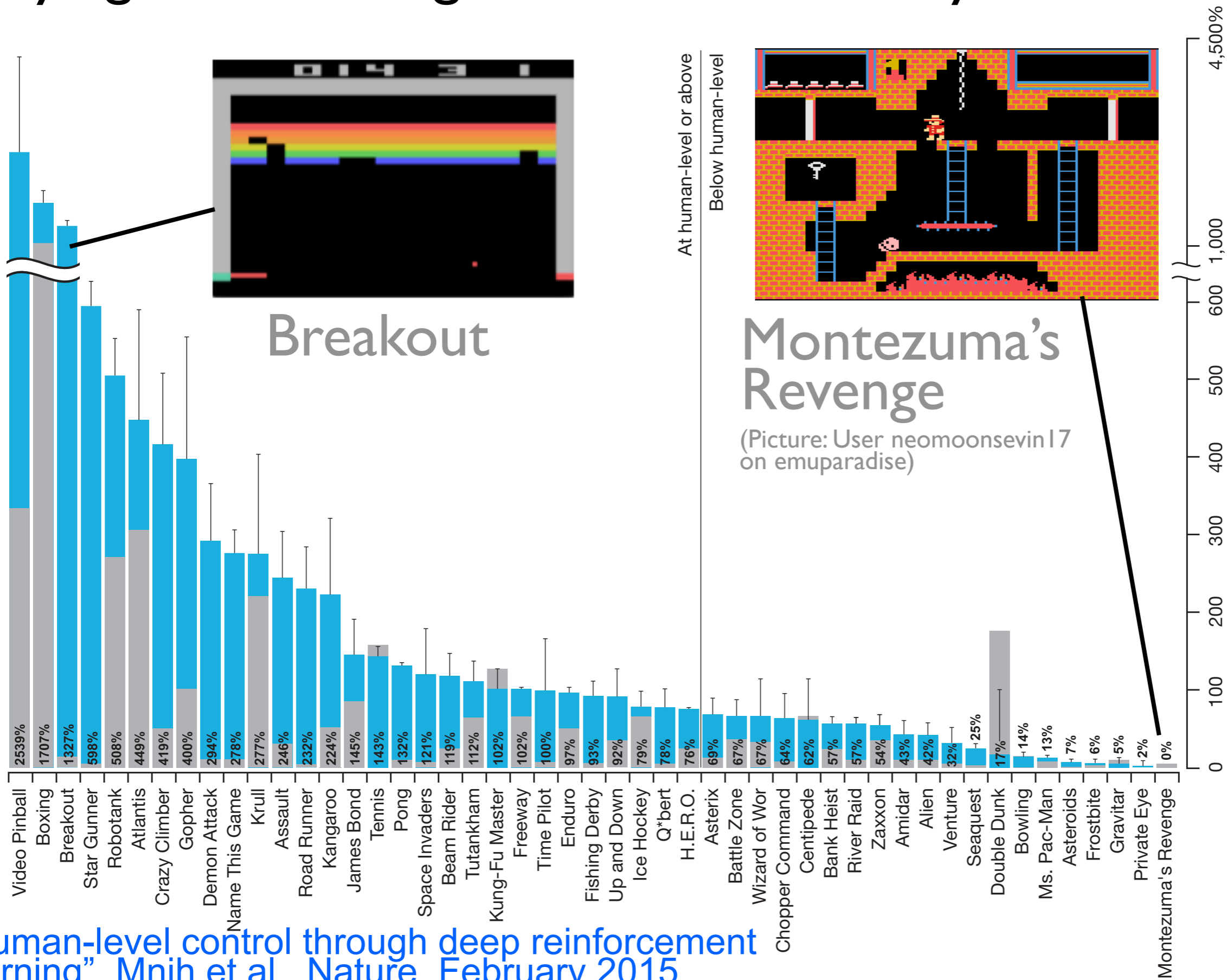
last four 84x84 pixel images as input [=state]
motion as output [=action]

Example: Learning to play Atari Video Games

“Human-level control through deep reinforcement learning”, Mnih et al., Nature, February 2015



Playing Atari video games, sometimes beyond human level



“Human-level control through deep reinforcement learning”, Mnih et al., Nature, February 2015

Example: Playing Atari video games

neural network observes screen and figures out a strategy to win, on its own



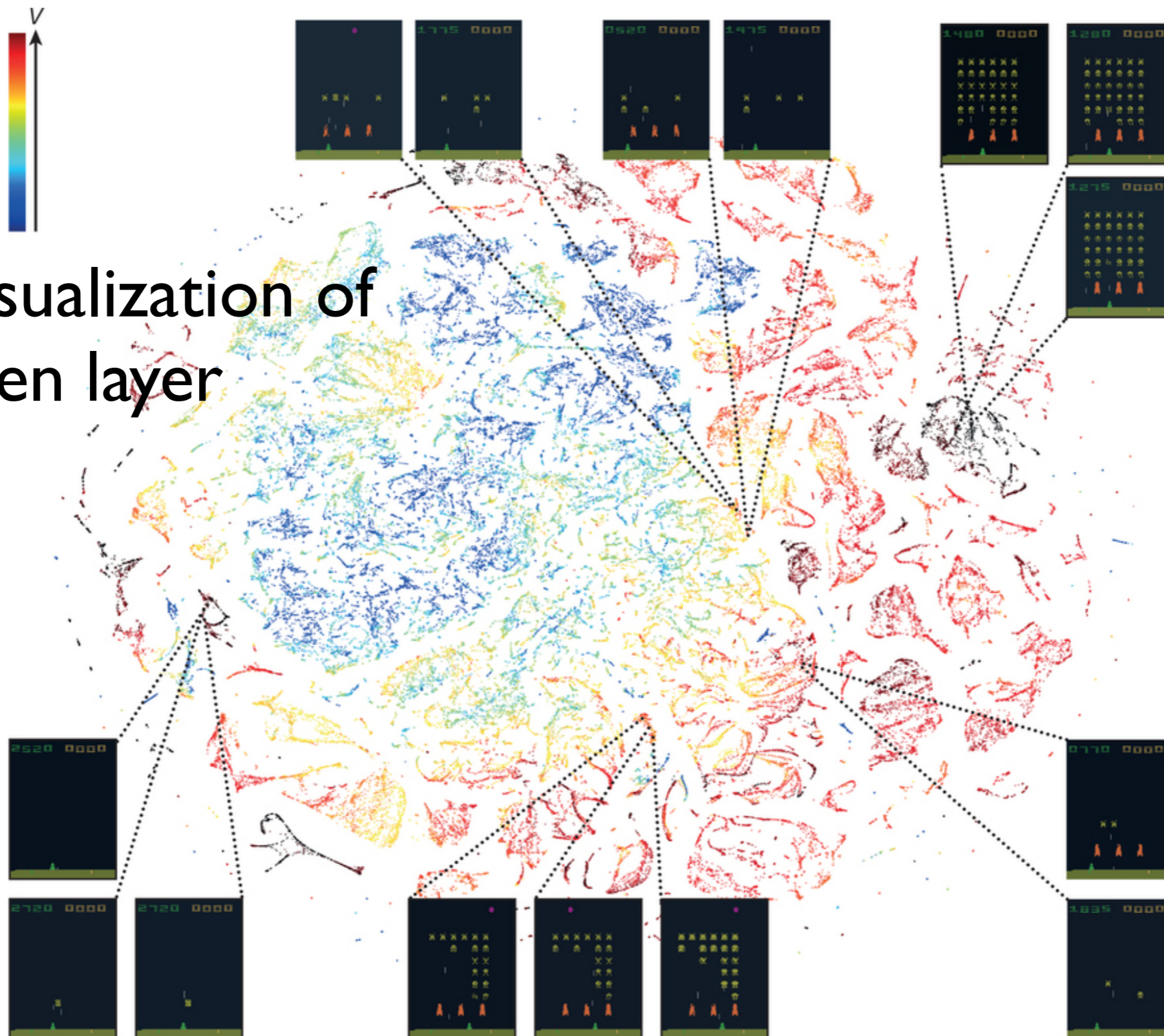
e.g. "Breakout"

(DeepMind team, 2013)

Example: Learning to play Atari Video Games

“Human-level control through deep reinforcement learning”, Mnih et al., Nature, February 2015

t-SNE visualization of
last hidden layer



Advantage Actor-Critic approaches

(combining Q learning and
policy gradient)

Basic idea (roughly): Learn value function and use it as a state-dependent baseline for the return!

In policy gradient, replace the return by:

estimated value (learned)

$$R_t \mapsto A_t \equiv r_t + \gamma V(s_{t+1}) - V(s_t)$$

less noisy estimate for return

"How much is the return for this particular action above the average, given the current state?"

"Advantage"

$$E[A_t | s_t, a_t] = Q(s_t, a_t) - V(s_t)$$

How to learn V?

$$\Delta\mu \sim \sum_t E[\{r_t + \gamma V_\mu(s_{t+1}) - V_\mu(s_t)\} \frac{\partial V_\mu(s_t)}{\partial \mu}]$$

Modern versions:
e.g. TRPO and PPO

Pro tip: Use PPO as a modern allrounder reinforcement learning method if you don't know anything particular about your problem

For these more advanced methods, use available RL libraries, for example:
"(stable) baselines", "tensorflow agents", ...

You just implement the environment and select the hyperparameters of the RL approach (and possibly provide the agent's network structure)

Summary:
Advantages & disadvantages
of model-free RL

Reinforcement learning: Advantages

Discover **Feedback** Strategies
(beyond GRAPE etc.)

No feedback: A^N strategies (A #actions N #steps)

With feedback: A^{M^N} strategies (M #msmt outcomes)

Model-free

No need to develop/fit/calibrate model/equations for dynamics of the world/the device

...can learn on real devices, with all imperfections

Reinforcement learning: Advantages

RL with deep neural networks:

Handle **arbitrary observations**

(images, videos, measurement results of any kind, sentences, graphs, ...)

Reinforcement learning: Challenges

Need to see **many evolutions!**

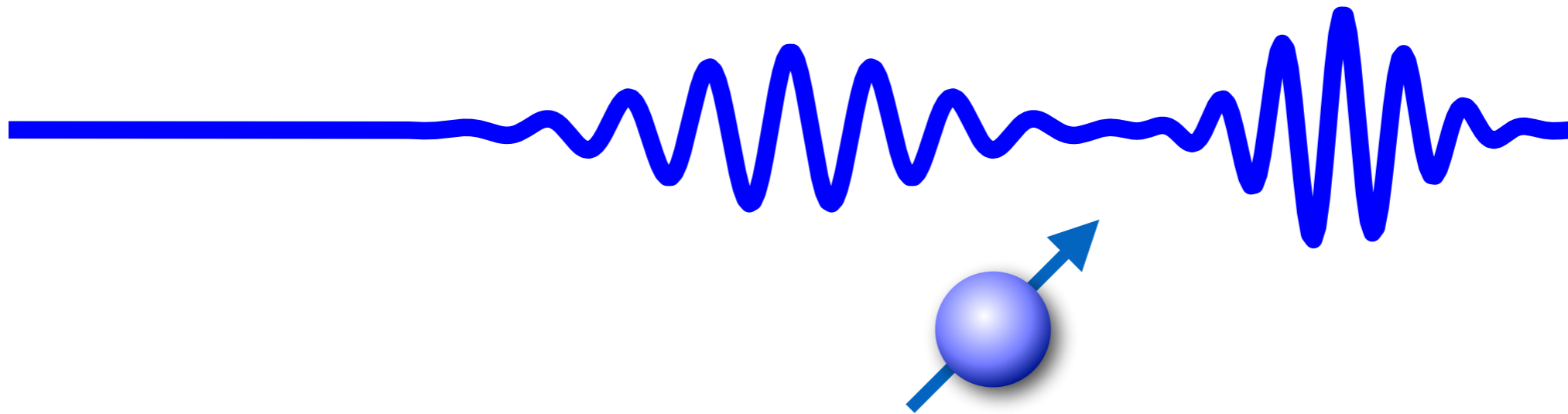
tens of thousands

Cannot discover 'isolated/rare-event' strategies

(also true for any other non-domain-specific algorithm)

Reinforcement learning for quantum physics

Quantum control



traditional: numerical techniques like GRAPE

new machine-learning techniques:

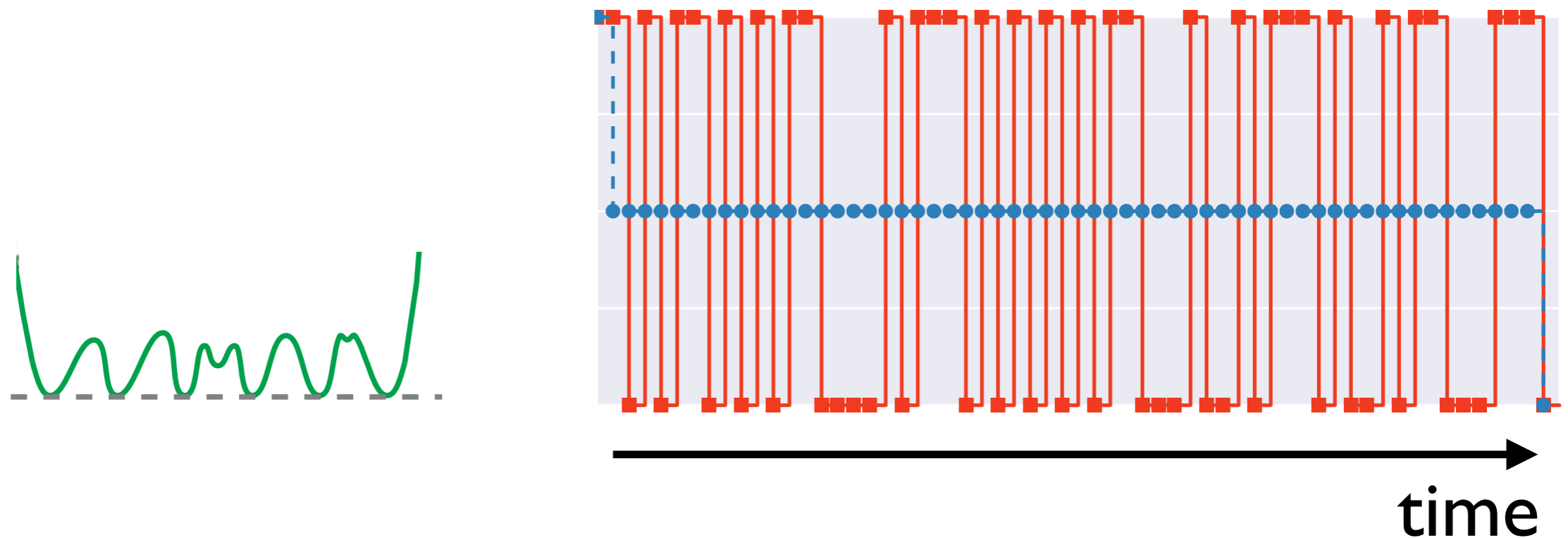
- model-free (implicitly learn model from behaviour)

- can easily include feedback

- profit from computer science method development

Quantum control

Bang-bang control (dynamical decoupling)

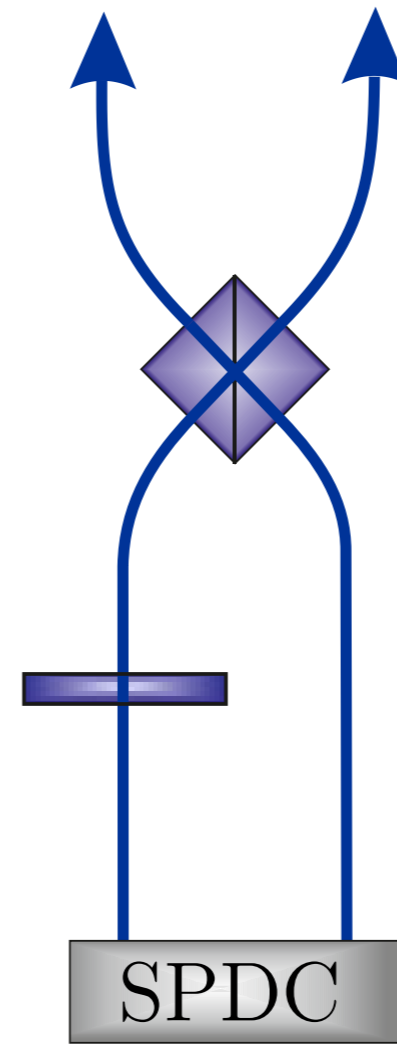
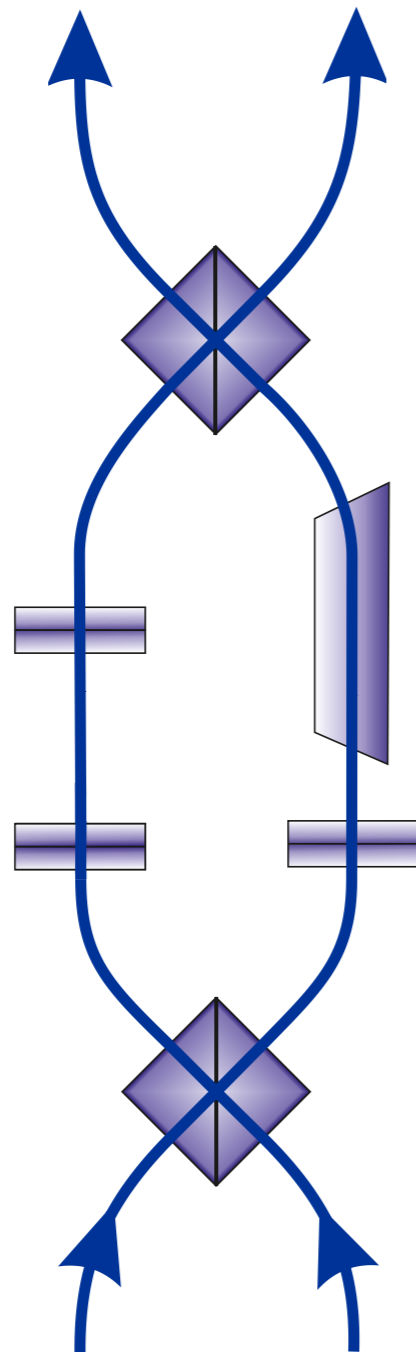


Bukov et al PRX 2018 (Mehta group BU)
(training can encounter glassy dynamics)

Q learning, table-based

Producing new experimental layouts

combine optical elements to produce highly entangled states



Briegel group
Melnikov et al
PNAS 2018

'projective simulation' RL technique

Adaptive quantum metrology

RL (Particle Swarm)

Hentschel et al.

PRL 2011

State preparation in spin chains

RL (Q learning)

Bukov et al. PRX 2018



Discover optical experiments

RL (Projective simulation)

Melnikov et al. PNAS 2018

Adaptive quantum metrology

RL (Particle Swarm)

Hentschel et al.

PRL 2011

State preparation in spin chains

RL (Q learning)

Bukov et al. PRX 2018



Quantum error correction

Deep RL (policy gradient)

Foesel et al. PRX 2018

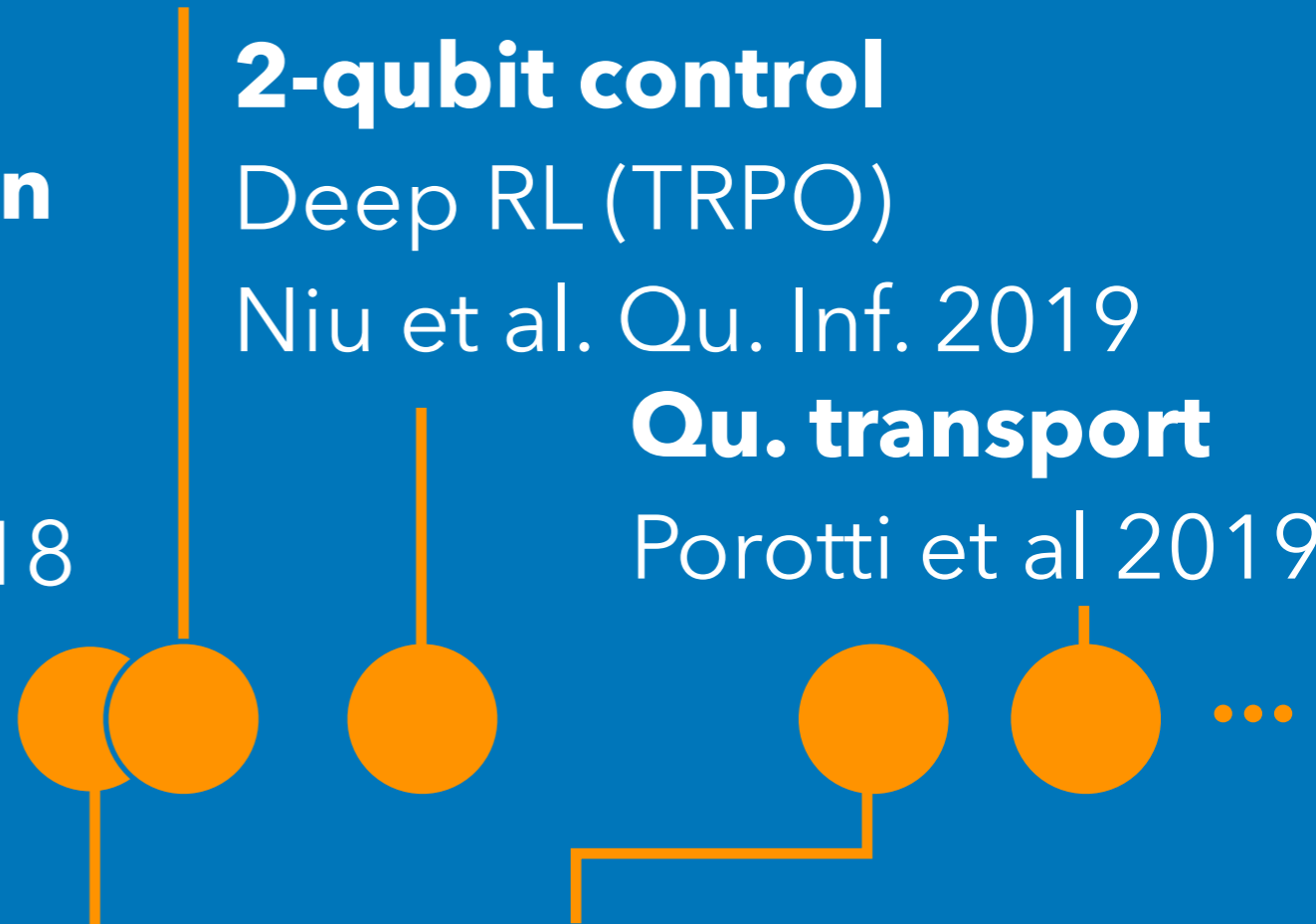
2-qubit control

Deep RL (TRPO)

Niu et al. Qu. Inf. 2019

Qu. transport

Porotti et al 2019



Surface code

Sweke et al 2018

Discover optical experiments

RL (Projective simulation)

Melnikov et al. PNAS 2018

Control of qubits and spin chains

Deep RL (PPO)

August et al. 2018

Adaptive quantum metrology

RL (Particle Swarm)

Hentschel et al.

PRL 2011

State preparation in spin chains

RL (Q learning)

Bukov et al. PRX 2018

2-qubit control

Deep RL (TRPO)

Niu et al. Qu. Inf. 2019

Qu. transport

Porotti et al 2019

Quantum error correction

Deep RL (policy gradient)

Foesel et al. PRX 2018

Surface code

Sweke et al 2018

Control of qubits and spin chains

Deep RL (PPO)

August et al. 2018

Discover optical experiments

RL (Projective simulation)

Melnikov et al. PNAS 2018

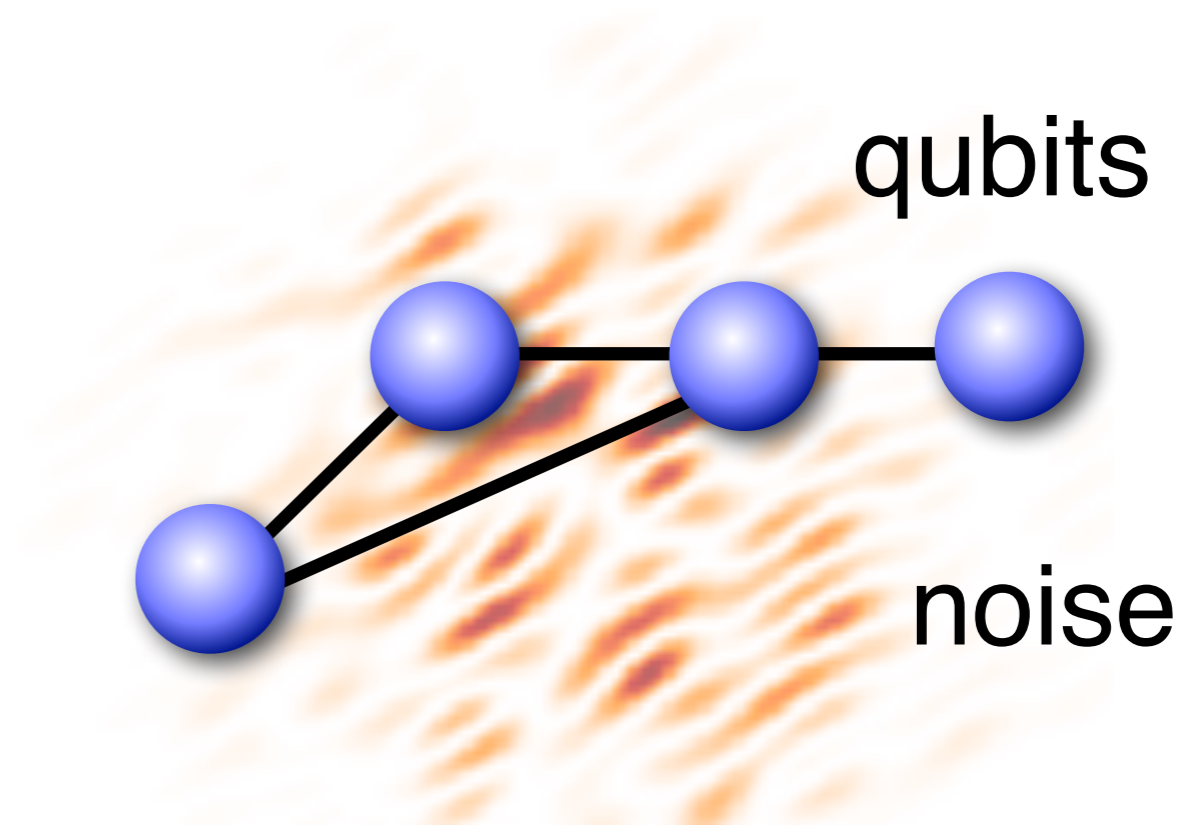


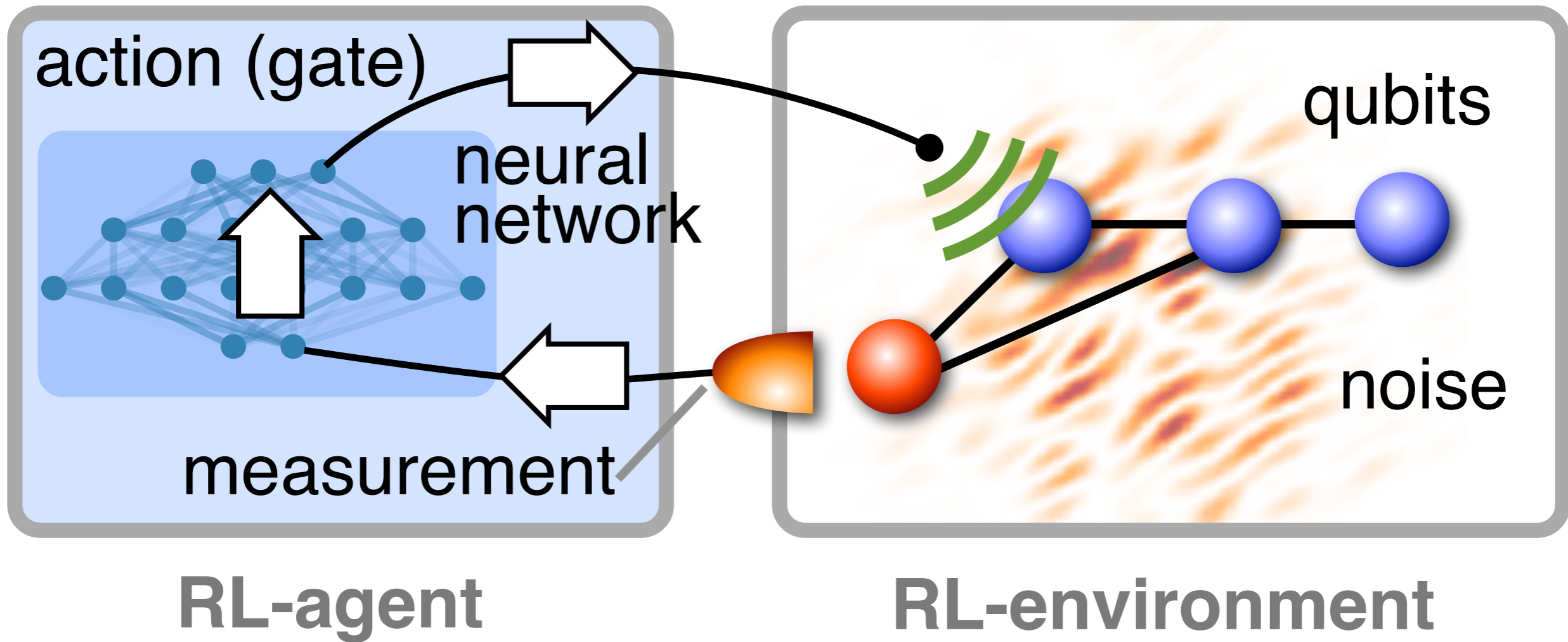
Case study:
Reinforcement learning
for quantum error correction

Physical Review X 031084 (2018)

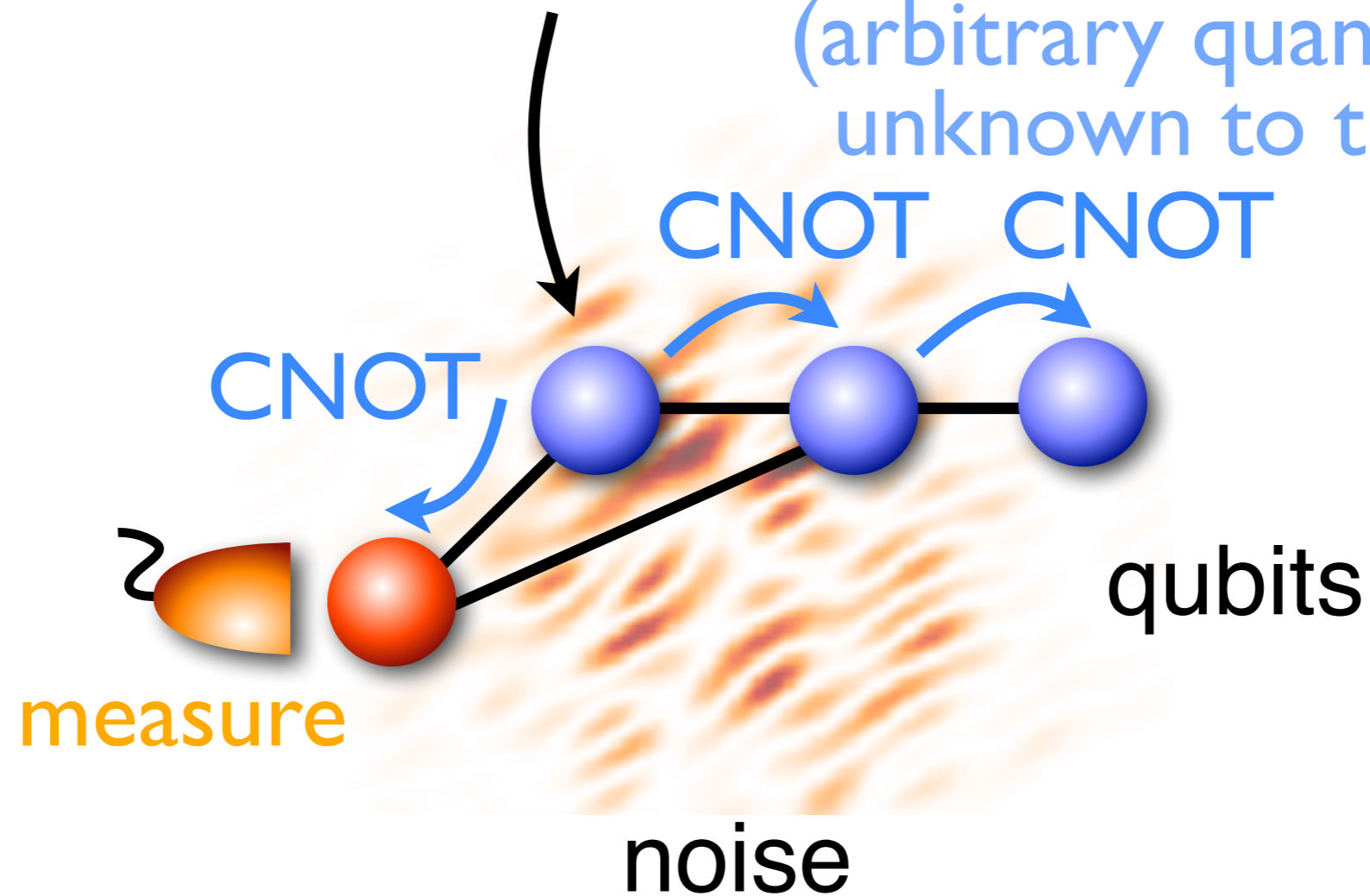
Thomas Fösel, Petru Tighineanu, Talitha Weiss, FM

**Goal: Apply neural-
network based
reinforcement
learning to quantum
physics!**



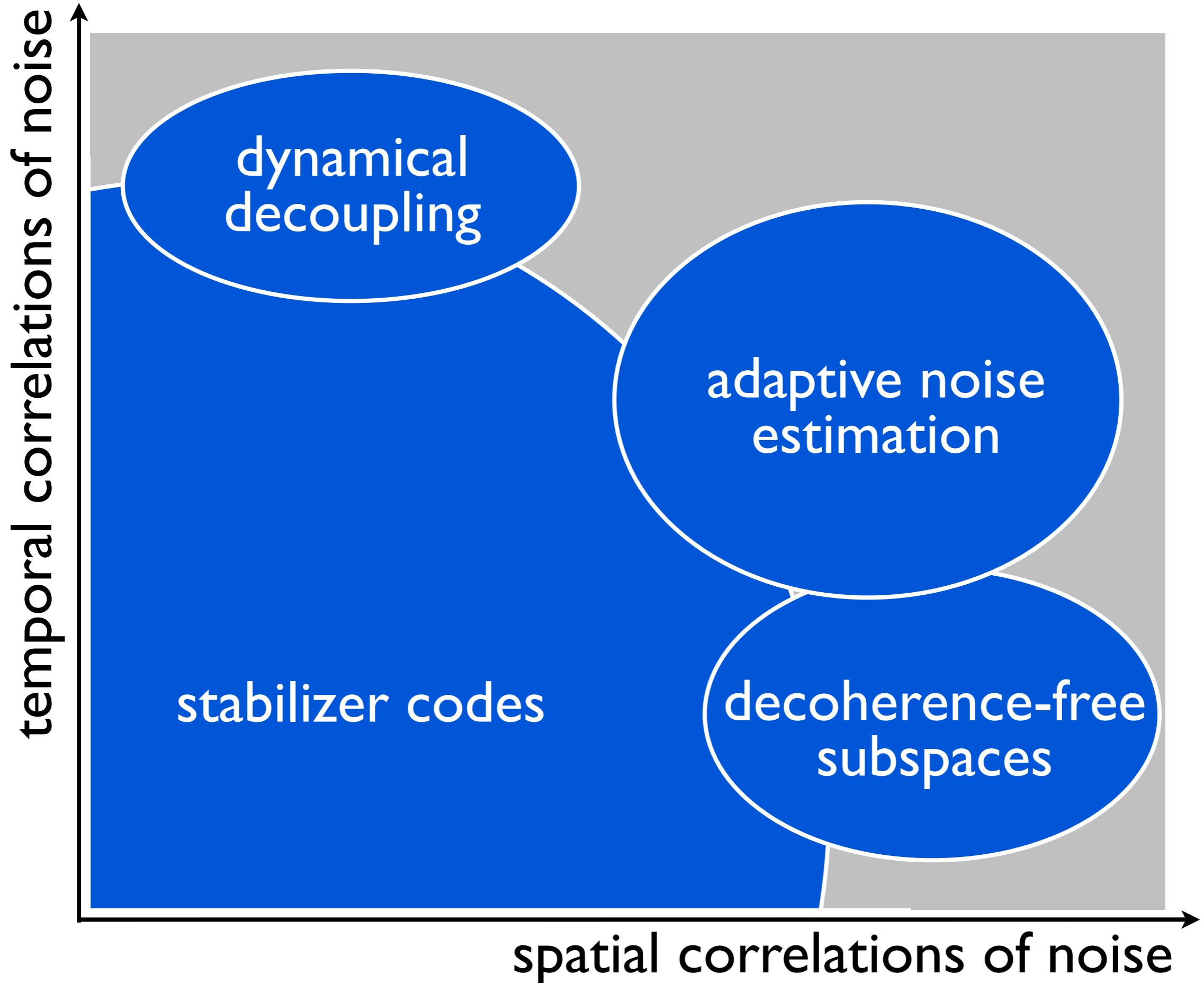


Initialize qubit here: $|\Psi(0)\rangle = \alpha |0\rangle + \beta |1\rangle$
(arbitrary quantum state, unknown to the agent)

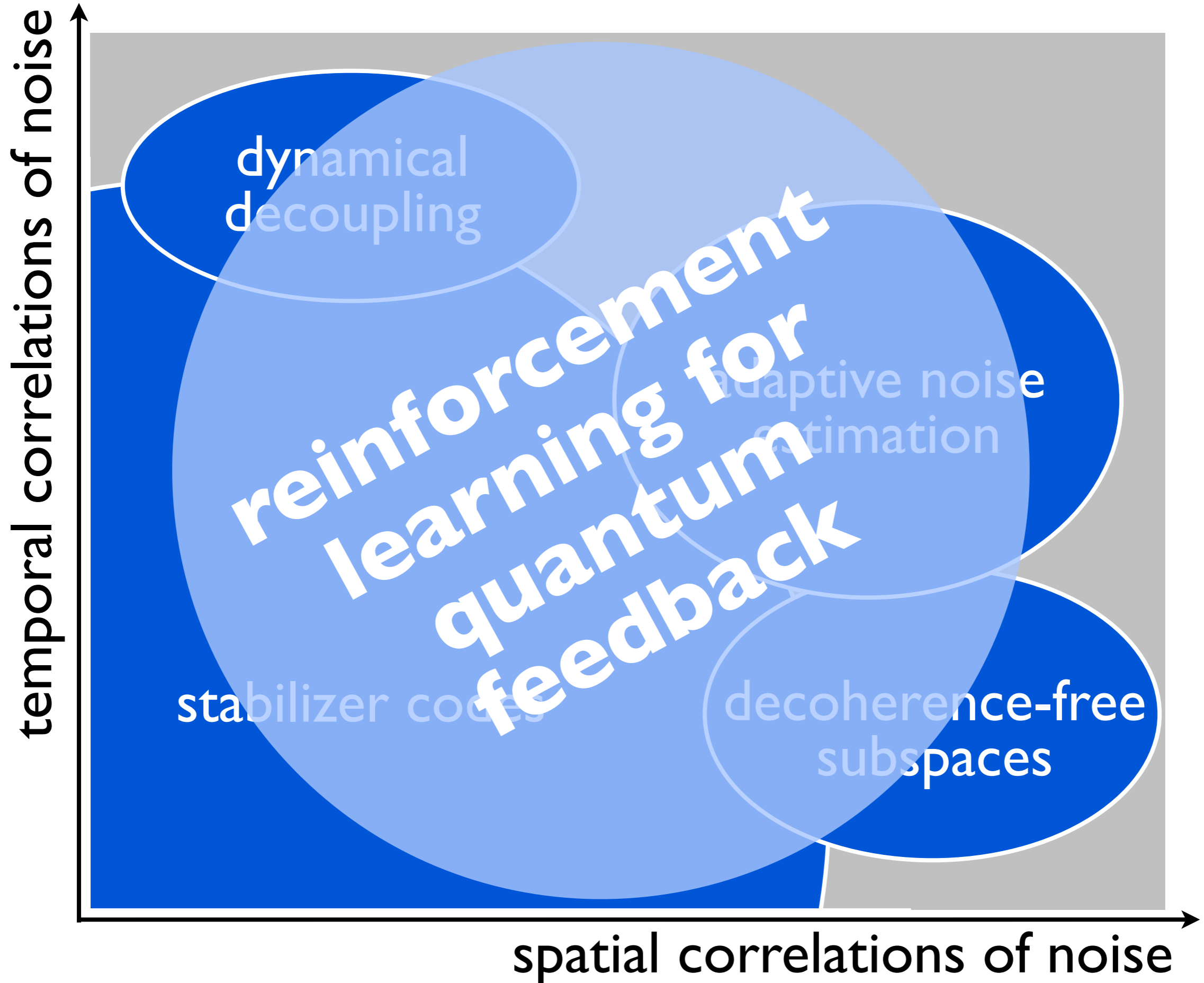


Goal: discover error correction strategies from scratch, without human guidance, for arbitrary noise and hardware constraints

Quantum Error Correction: many approaches



Quantum Error Correction: many approaches



$$2^N$$

This approach requires numerical simulation

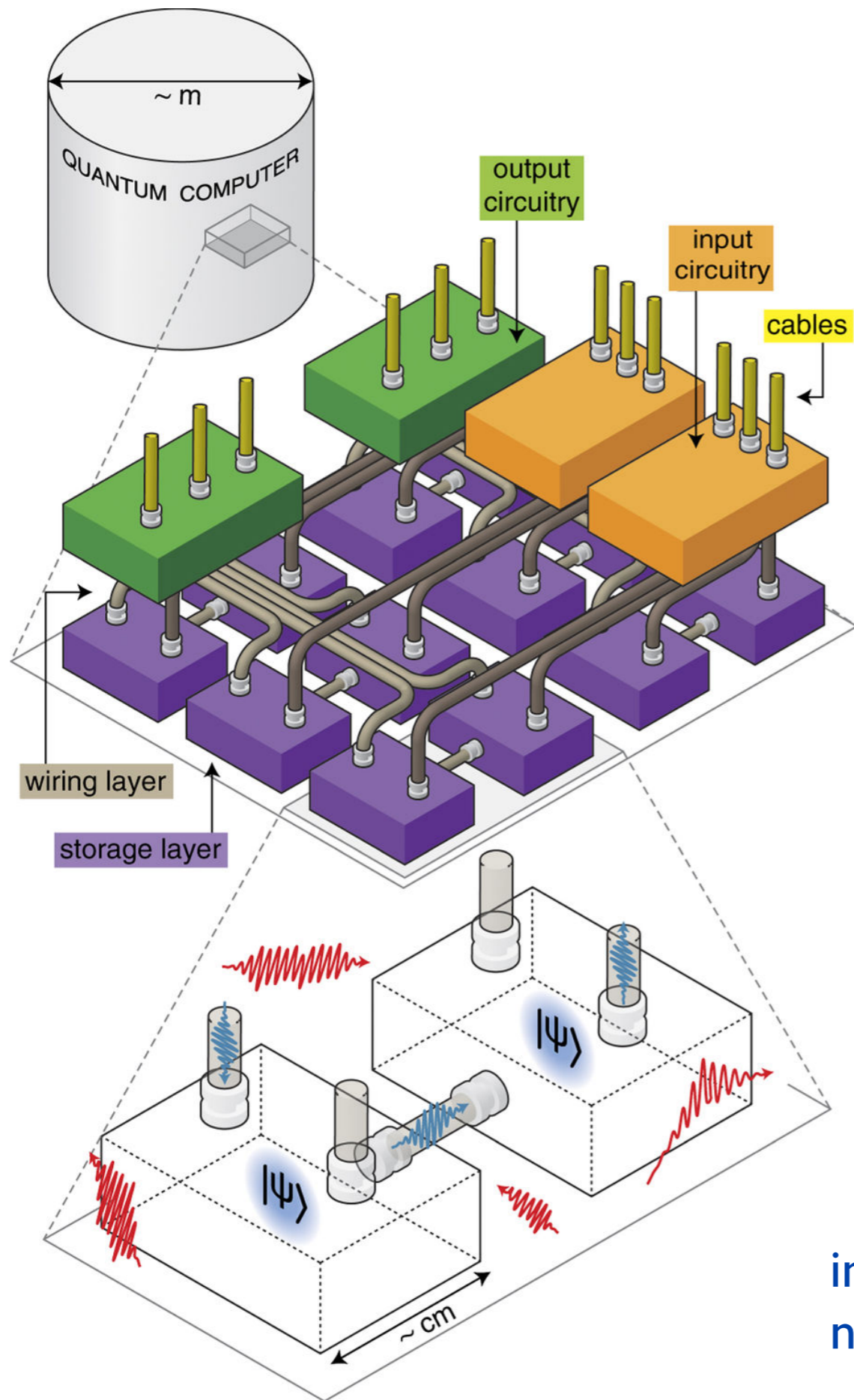
By definition, we cannot classically simulate a full-scale quantum computer

➔ most useful for optimizing the performance of **small quantum modules ~O(5) qubits**

Advantage:

flexibility – hardware-adapted strategies

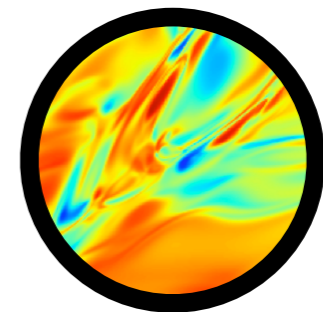
Modular approach to quantum computation



proposed for multiple hardware platforms, including superconducting circuits, ion traps, NV centres

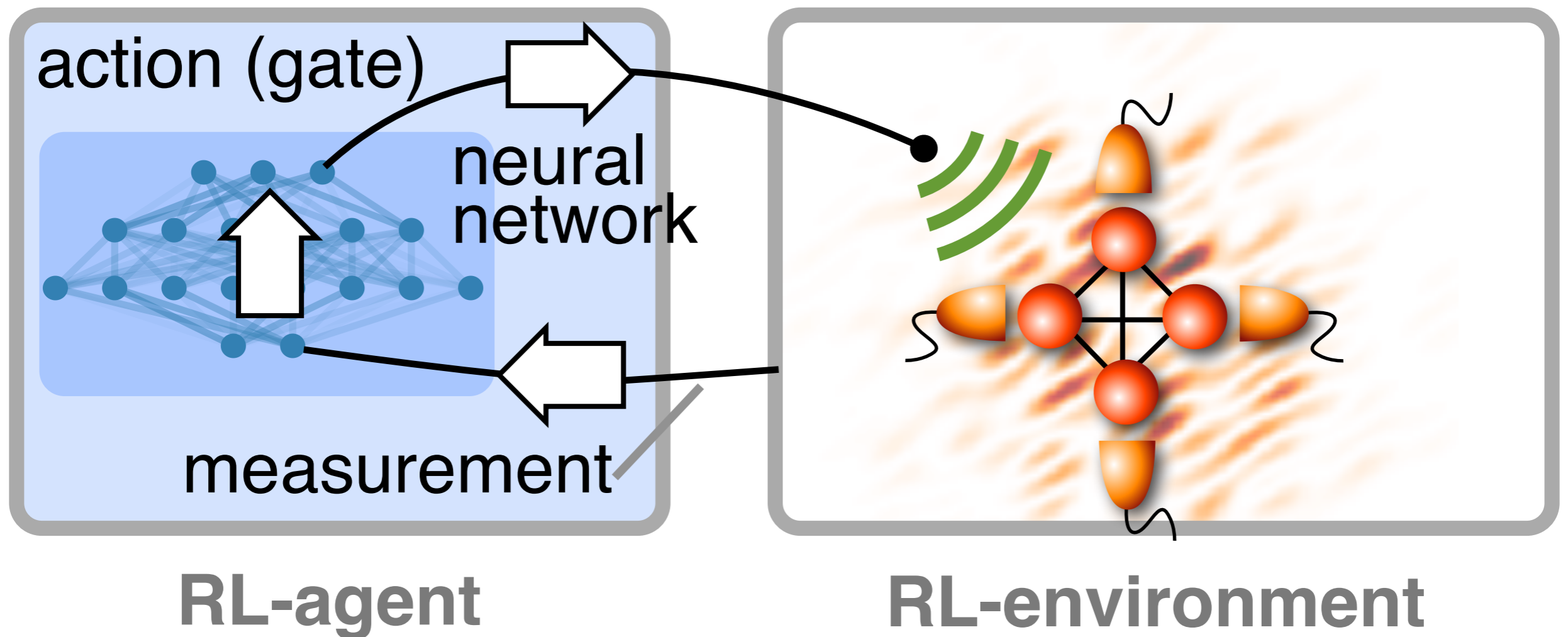
image from Brecht et al., Devoret/Schoelkopf labs, npj Quantum Information 2, 16002 (2016)

Numerical results



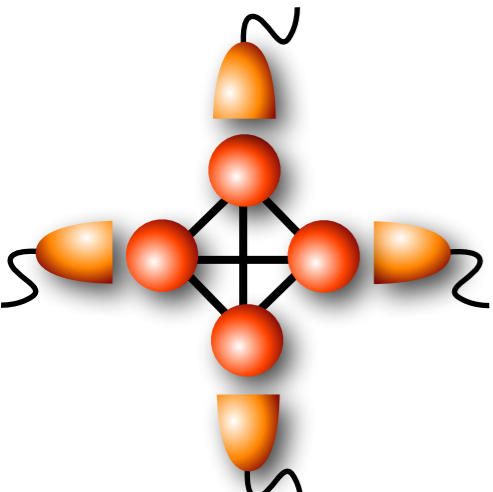
At present: applied neural-network-based reinforcement learning for up to 5 qubits, for several different physical scenarios

Demonstrated reinforcement-learning as a flexible, generally applicable method (no need to change method for different scenarios)

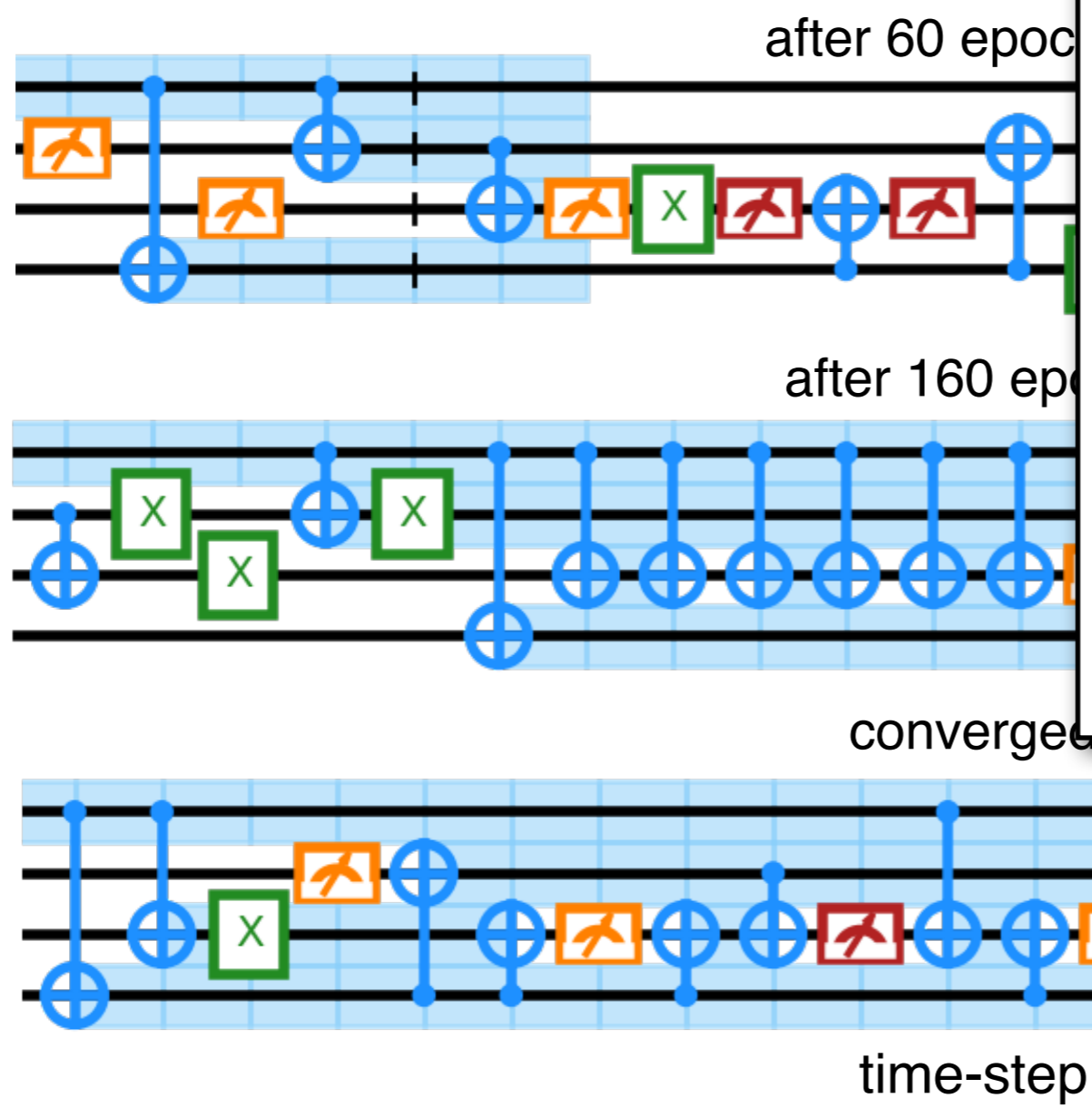


example: 4 qubits, measurements possible on all, CNOTs between all, bit-flip noise on all

$$\dot{\hat{\rho}} = \frac{1}{T_{\text{dec}}} \sum_j (\hat{\sigma}_{xj} \hat{\rho} \hat{\sigma}_{xj} - \hat{\rho})$$



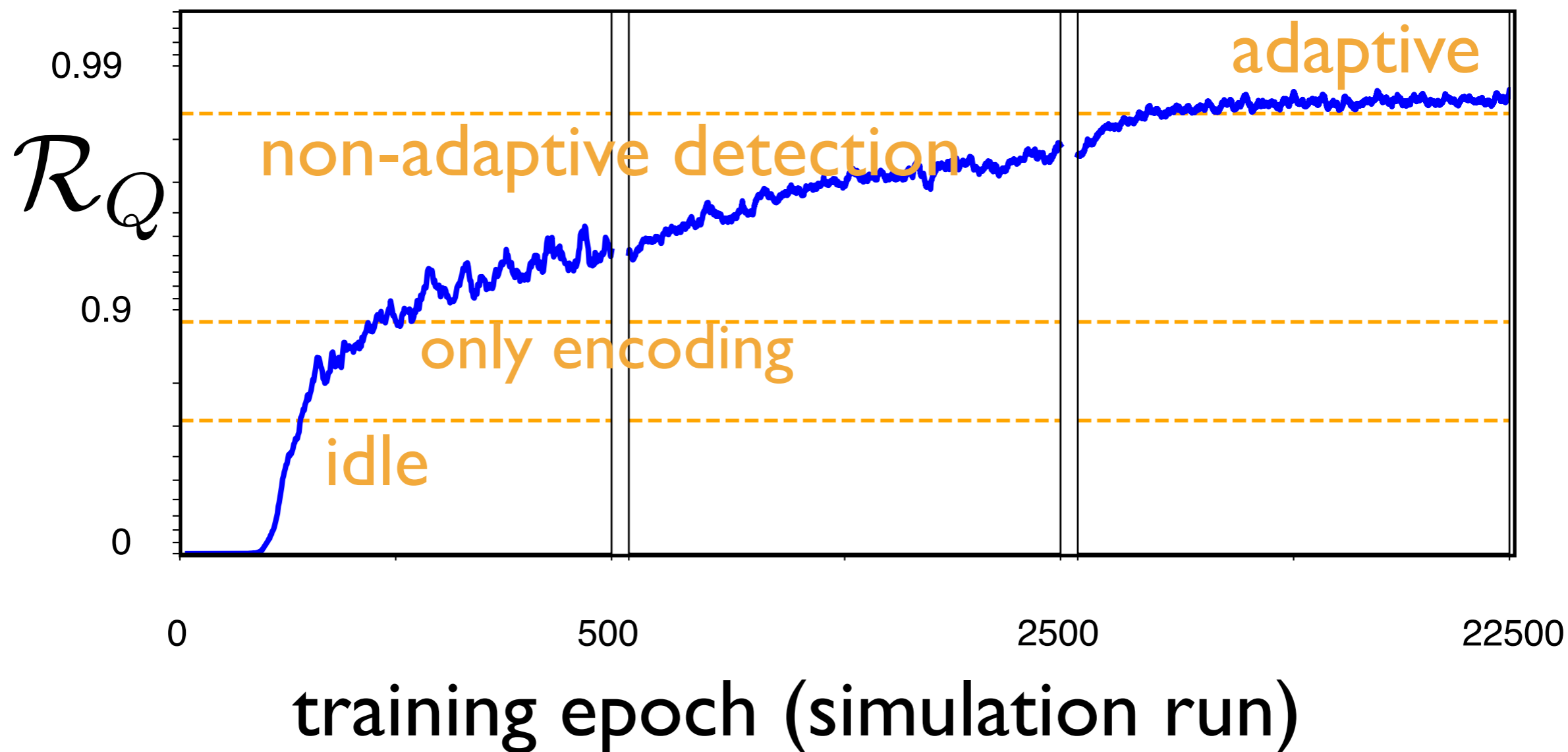
The network...
avoids catastrophic measurements
finds repetition code encoding sequence
discovers parity detections
applies them periodically



(showing 20 out of 200 time steps)

Training Progress

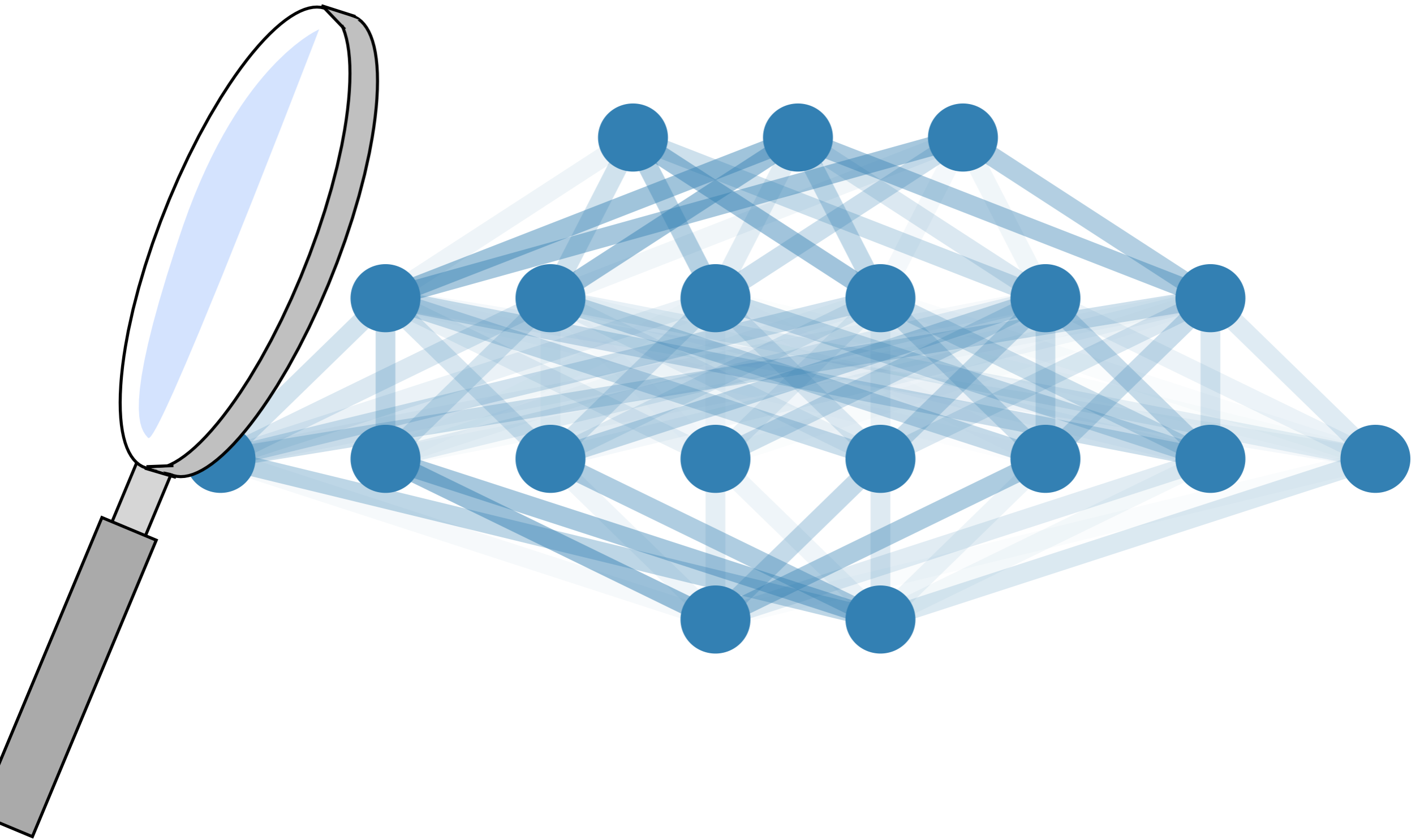
measure of success=
"recoverable quantum information"



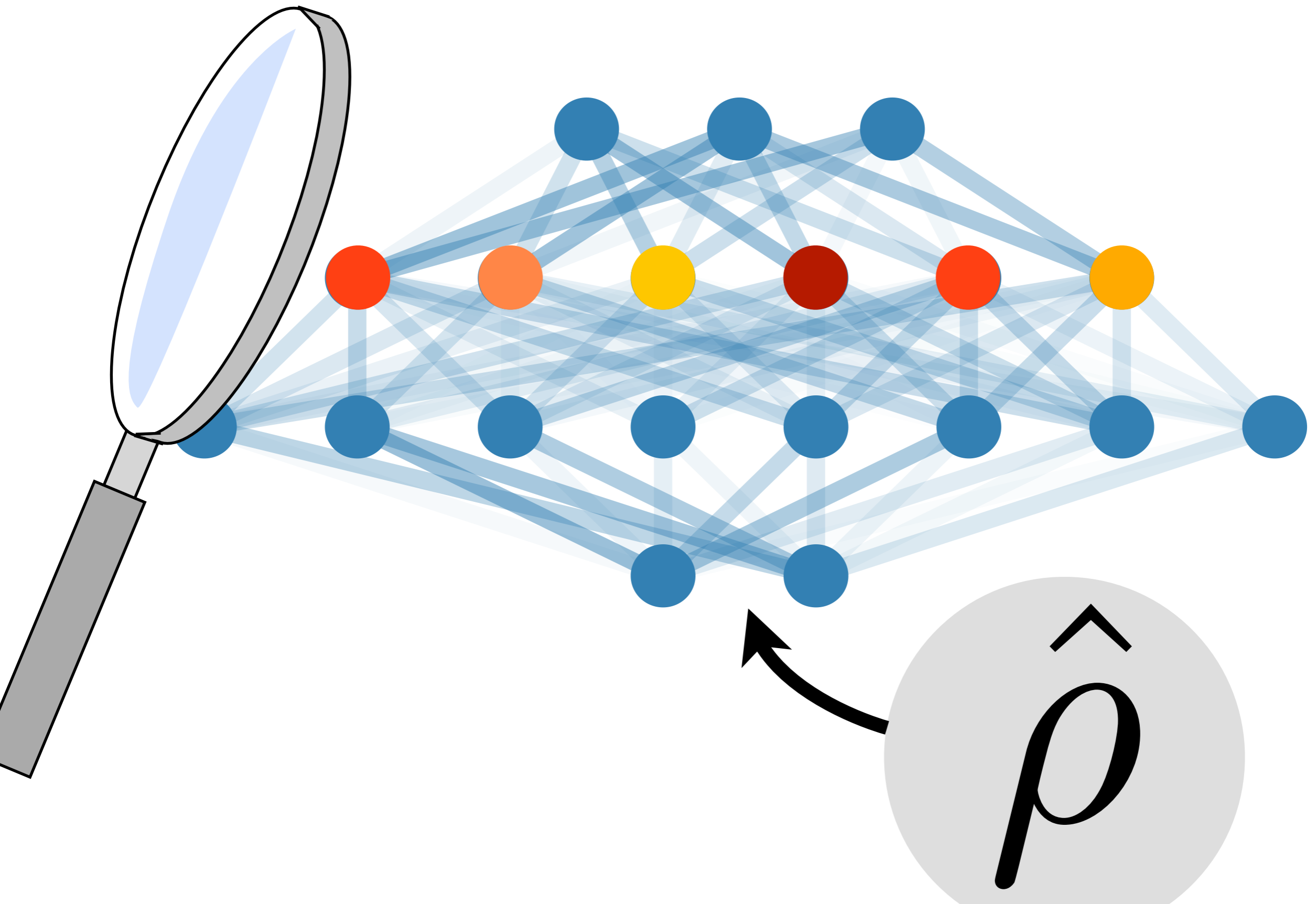
Can we understand what the network does?

"opening the box"

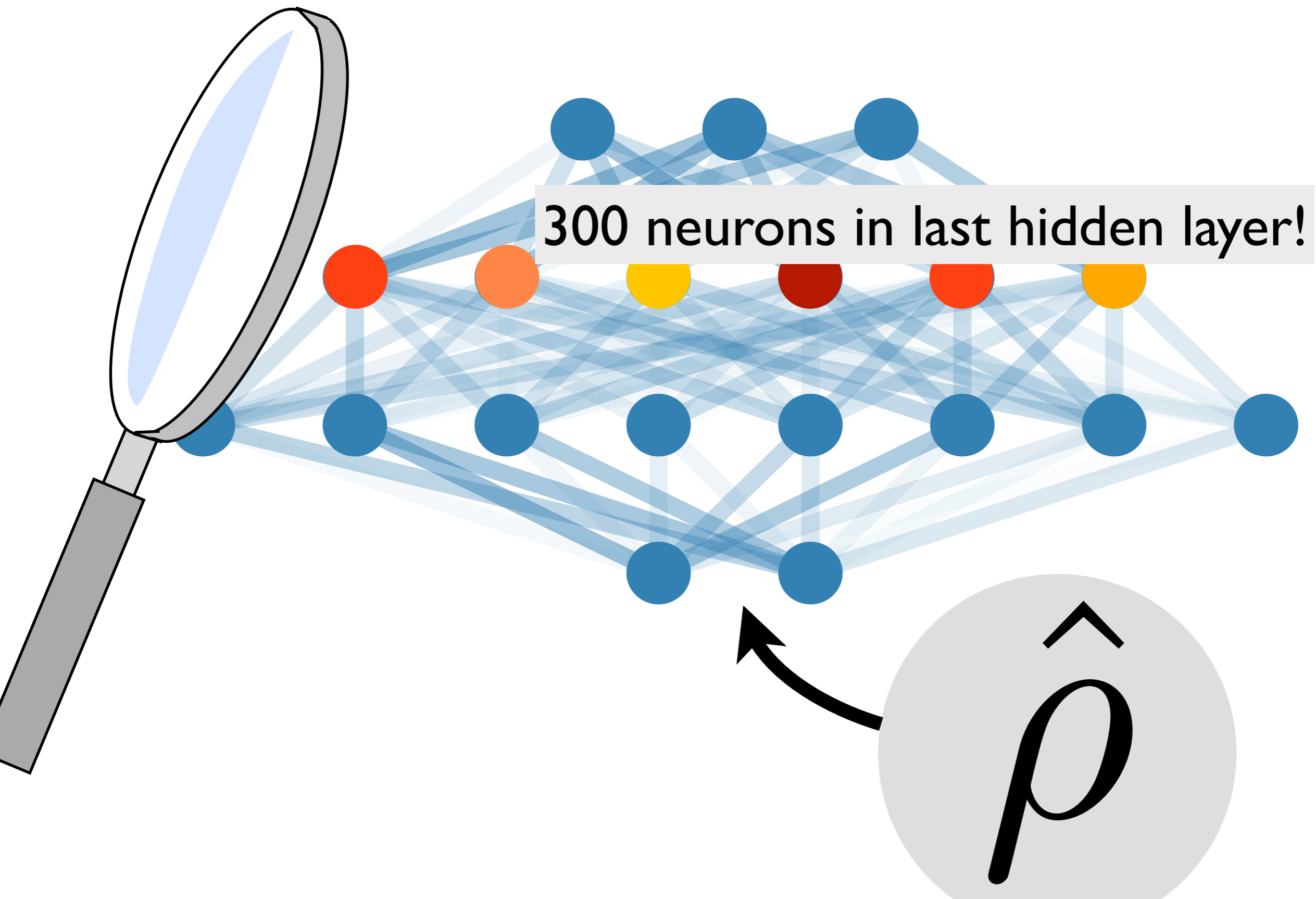
“How does the network operate?”



“How does the network operate?”

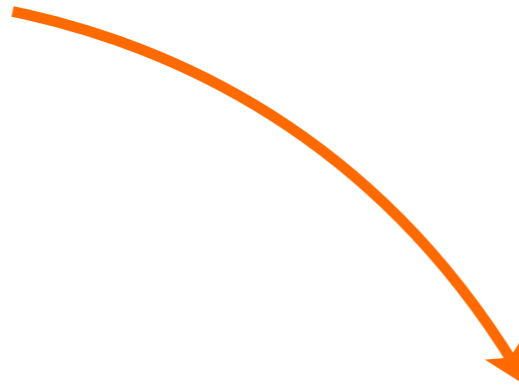


“How does the network operate?”

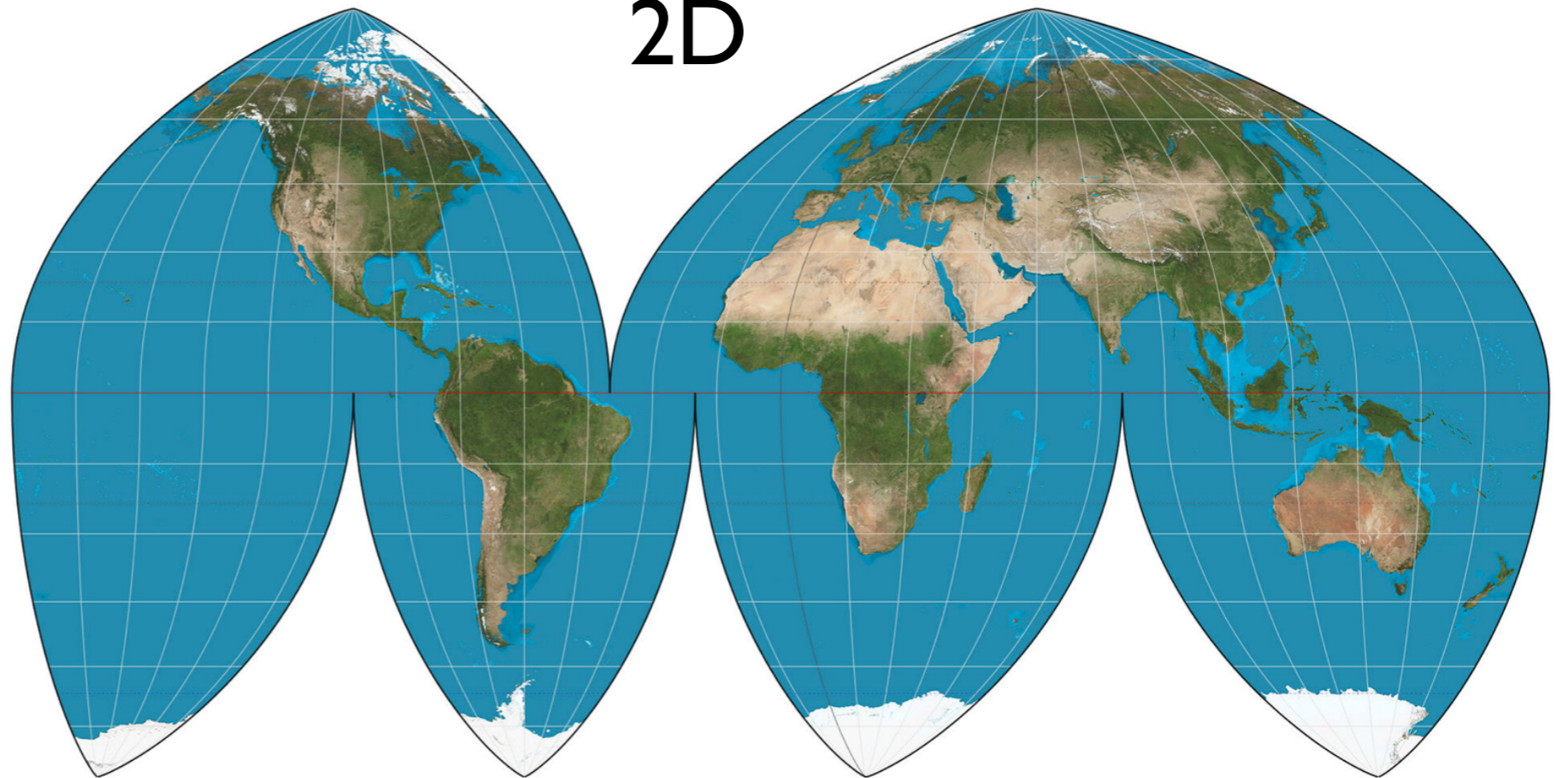




3D

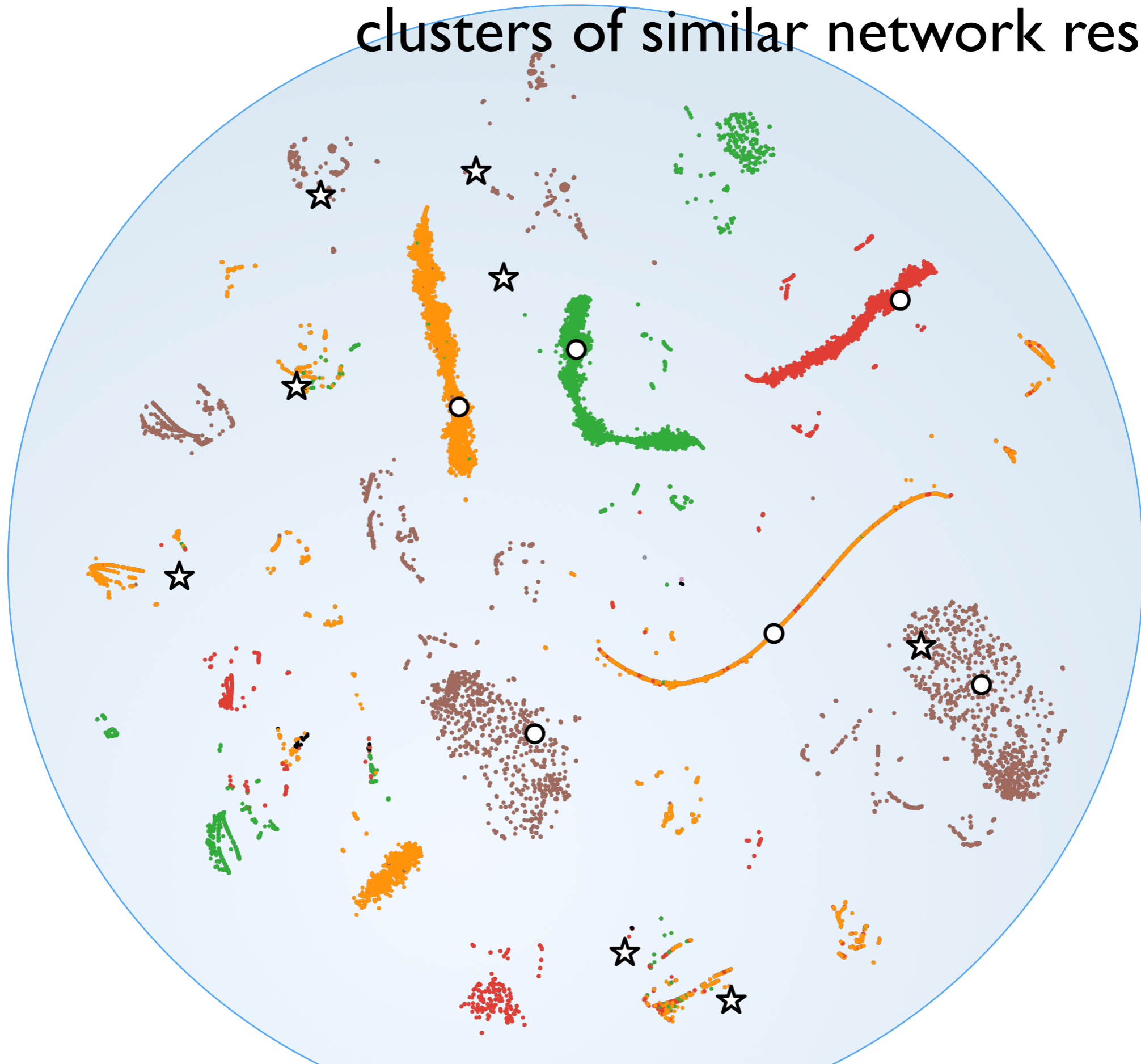


2D



(pictures: Wikipedia, 'Strebe')

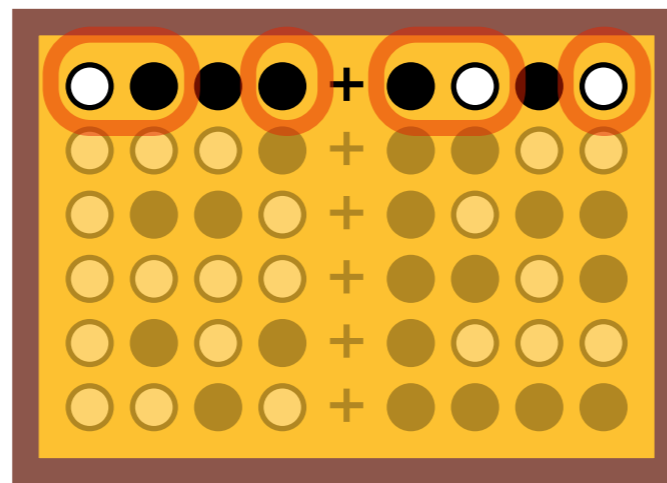
clusters of similar network responses!



“t-SNE” method from machine learning field

Visualize density matrix of
given quantum state:
Decompose into eigenstates

eigenstates



probabilities (eigenvalues)

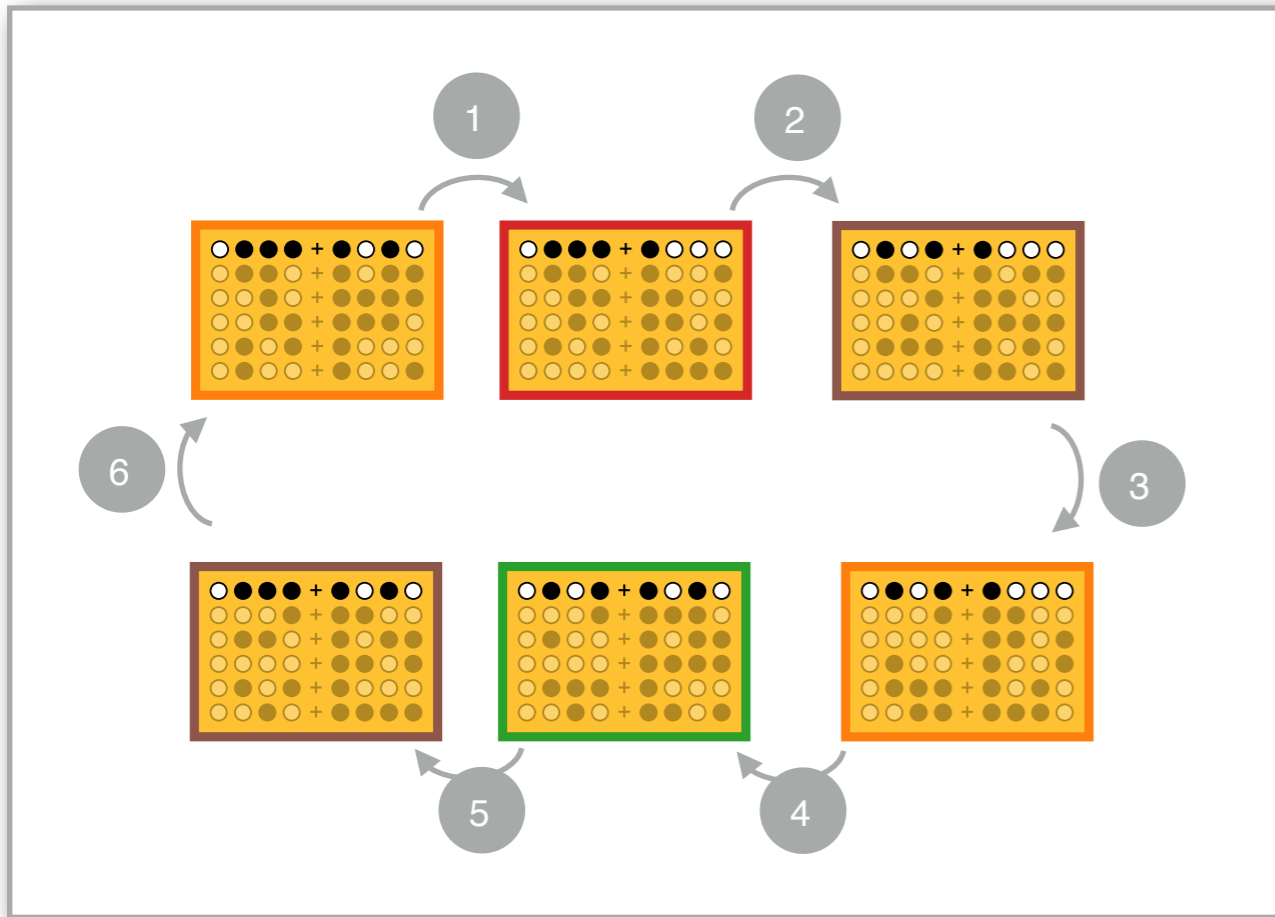


3 qubits used for encoding

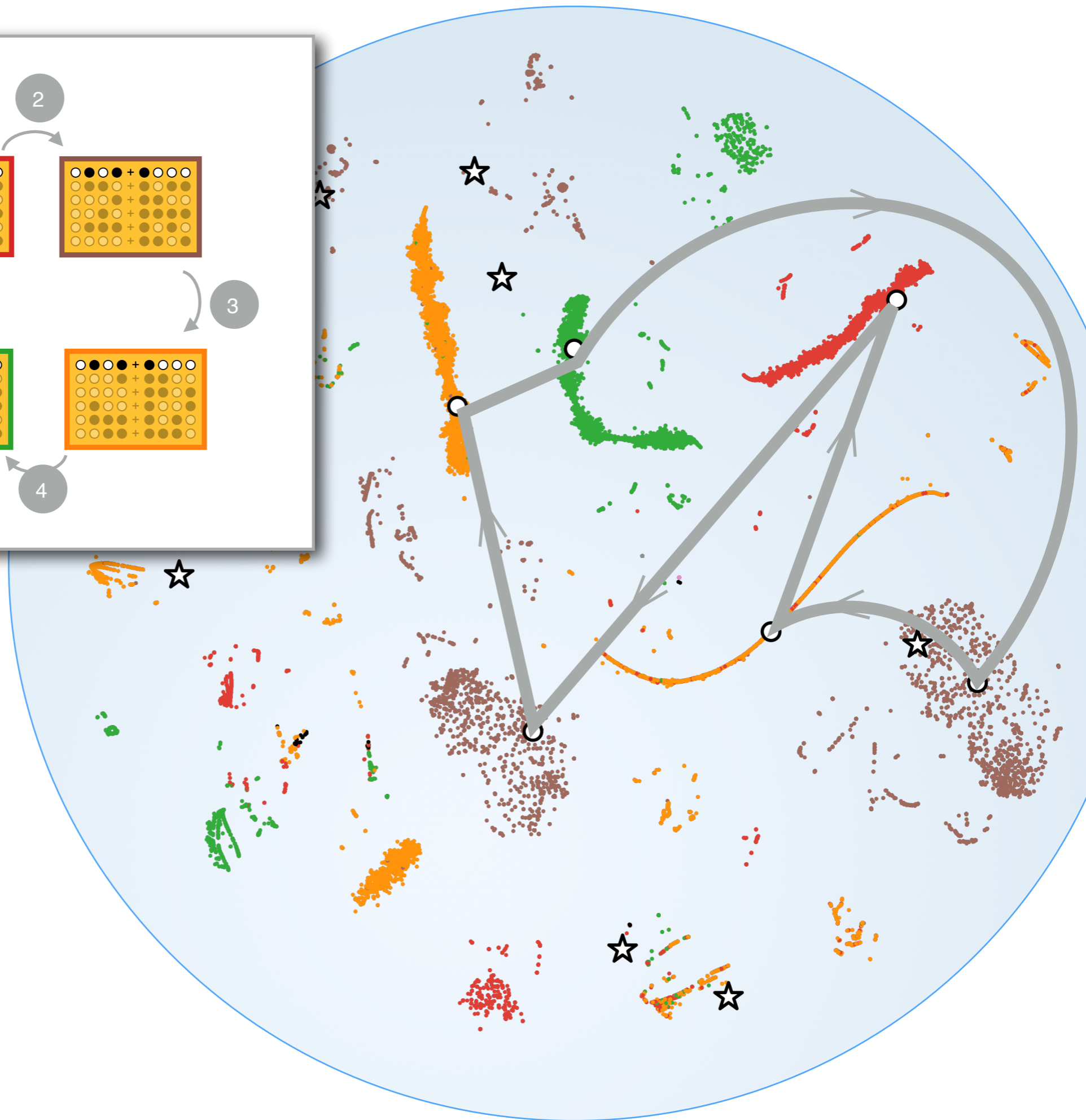
$$\alpha |011\rangle + \beta |100\rangle$$

1 qubit for ancilla (measurement)

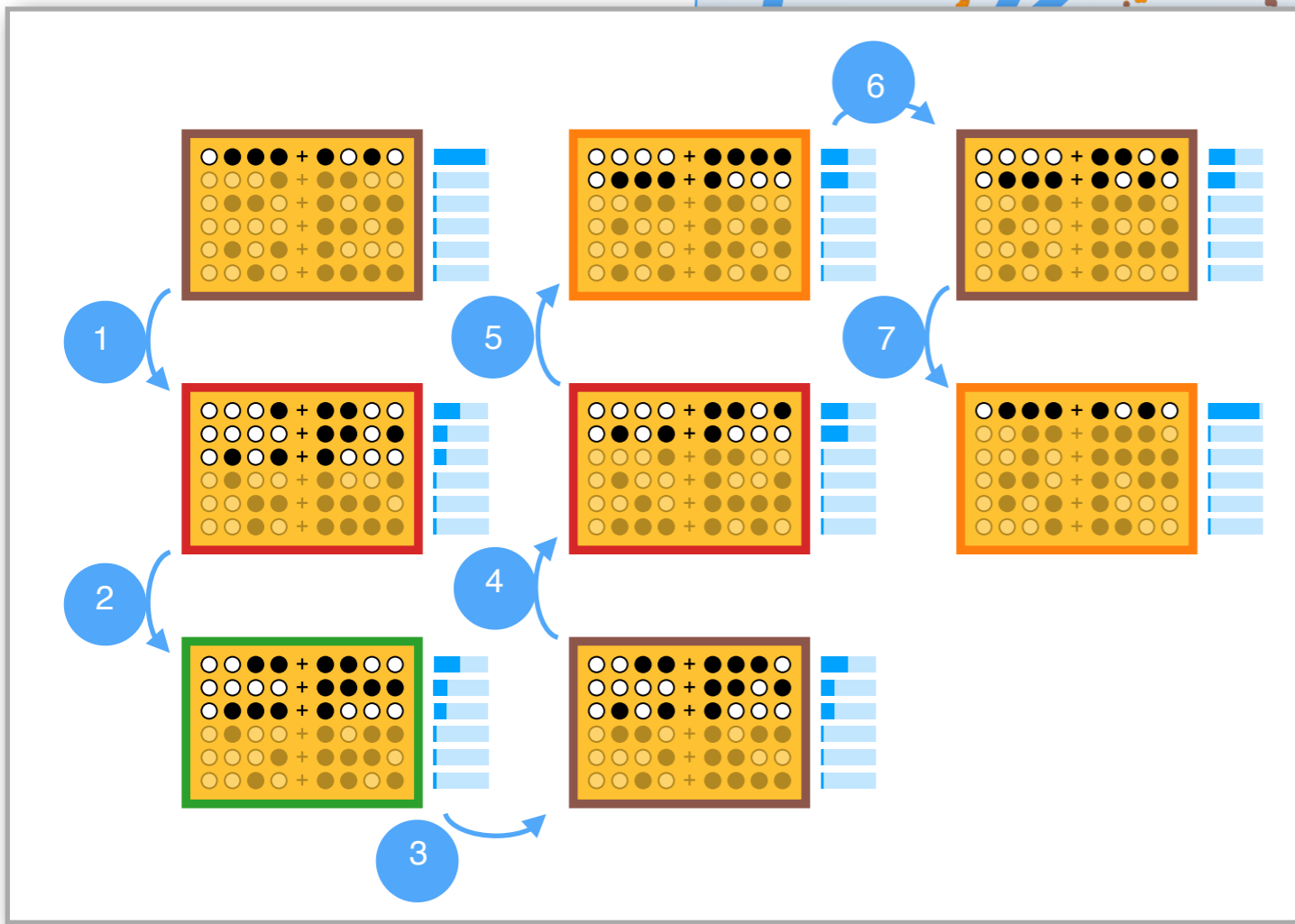
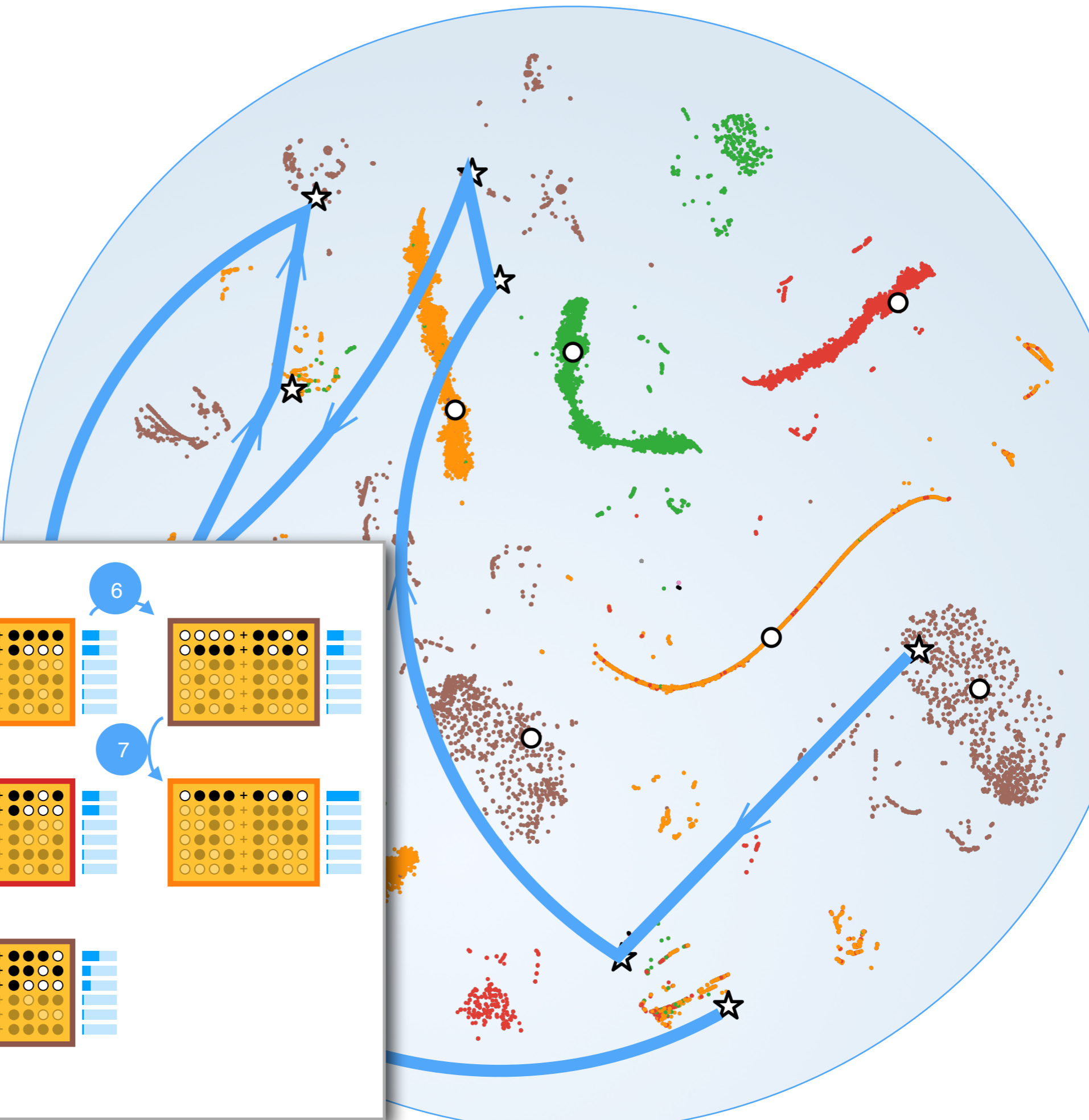
we don't provide any of this, the
network discovers this on its own...



“standard
detection
cycle”



unexpected measurement indicates error and triggers more complex sequence!



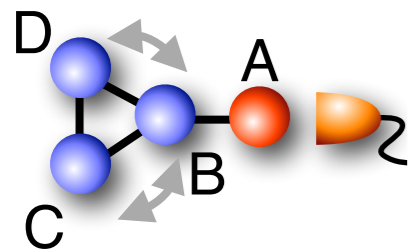
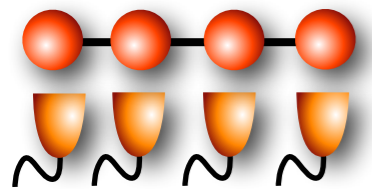
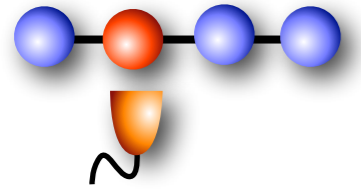
Network discovers something new

Even if encoding is known: Gate sequences for error detection/correction depend on hardware-specific constraints (like connectivity, available gates)!

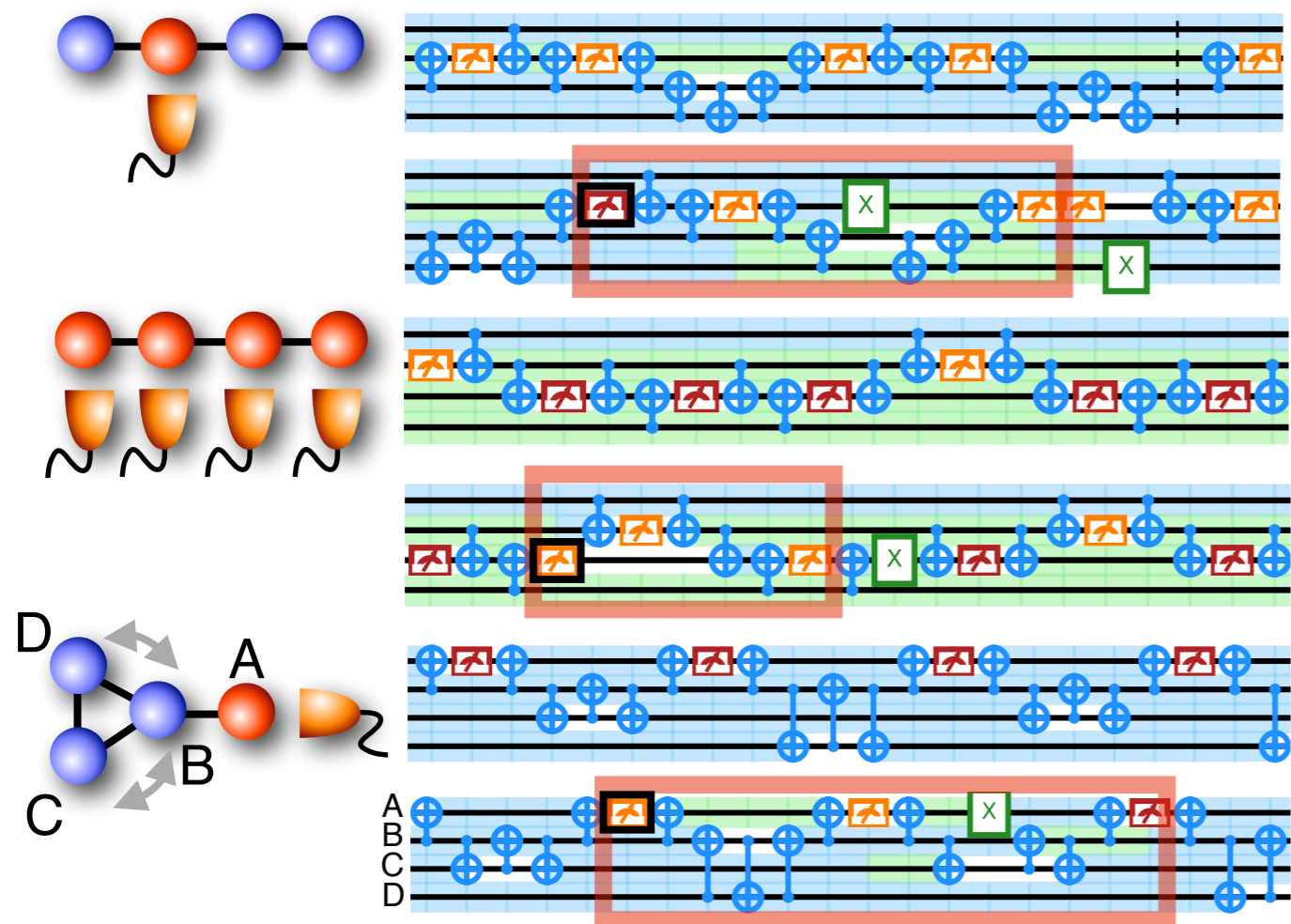
What about more complex
qubit connectivities ?

(more complex than all-to-all)

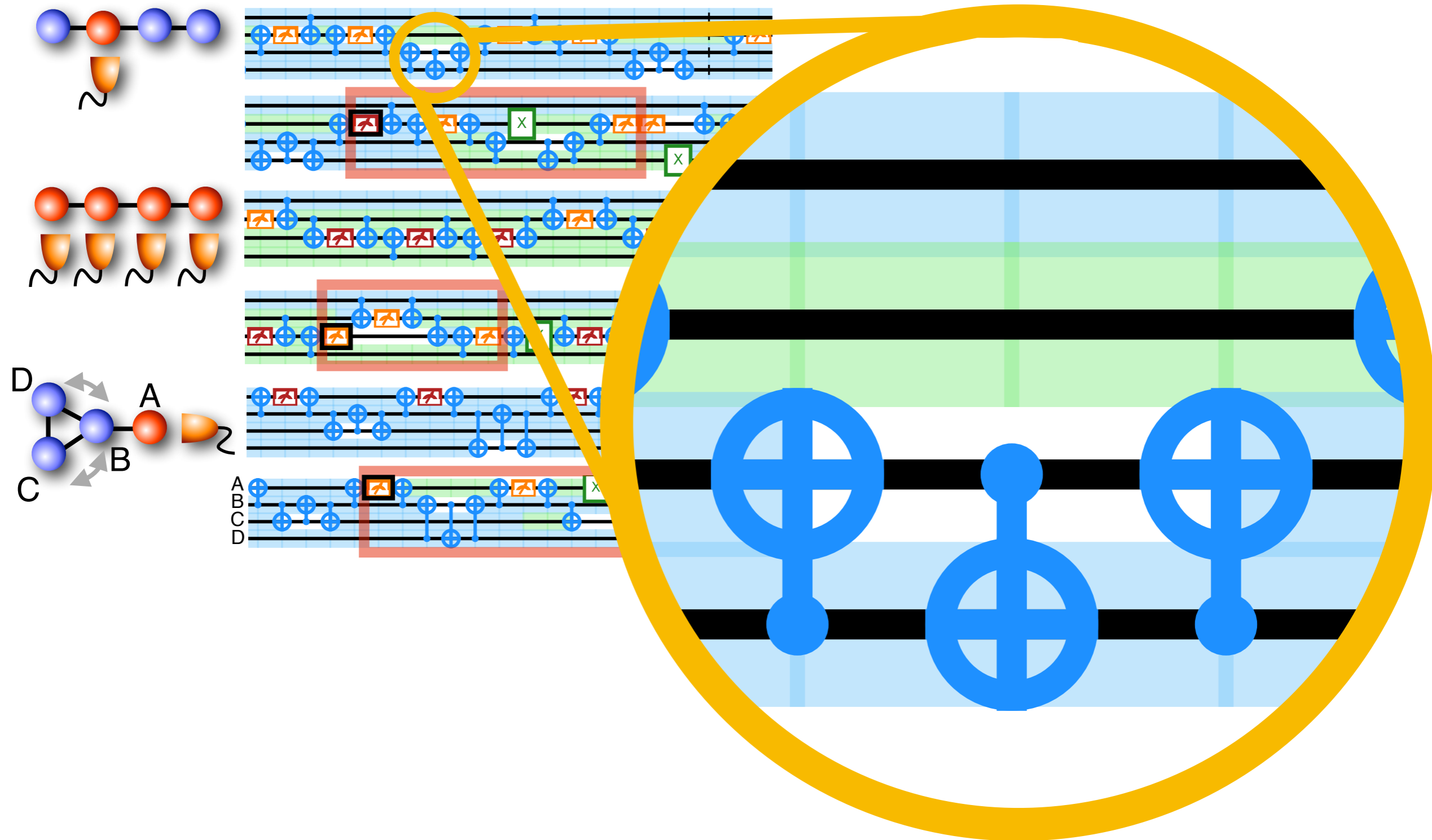
Different topologies



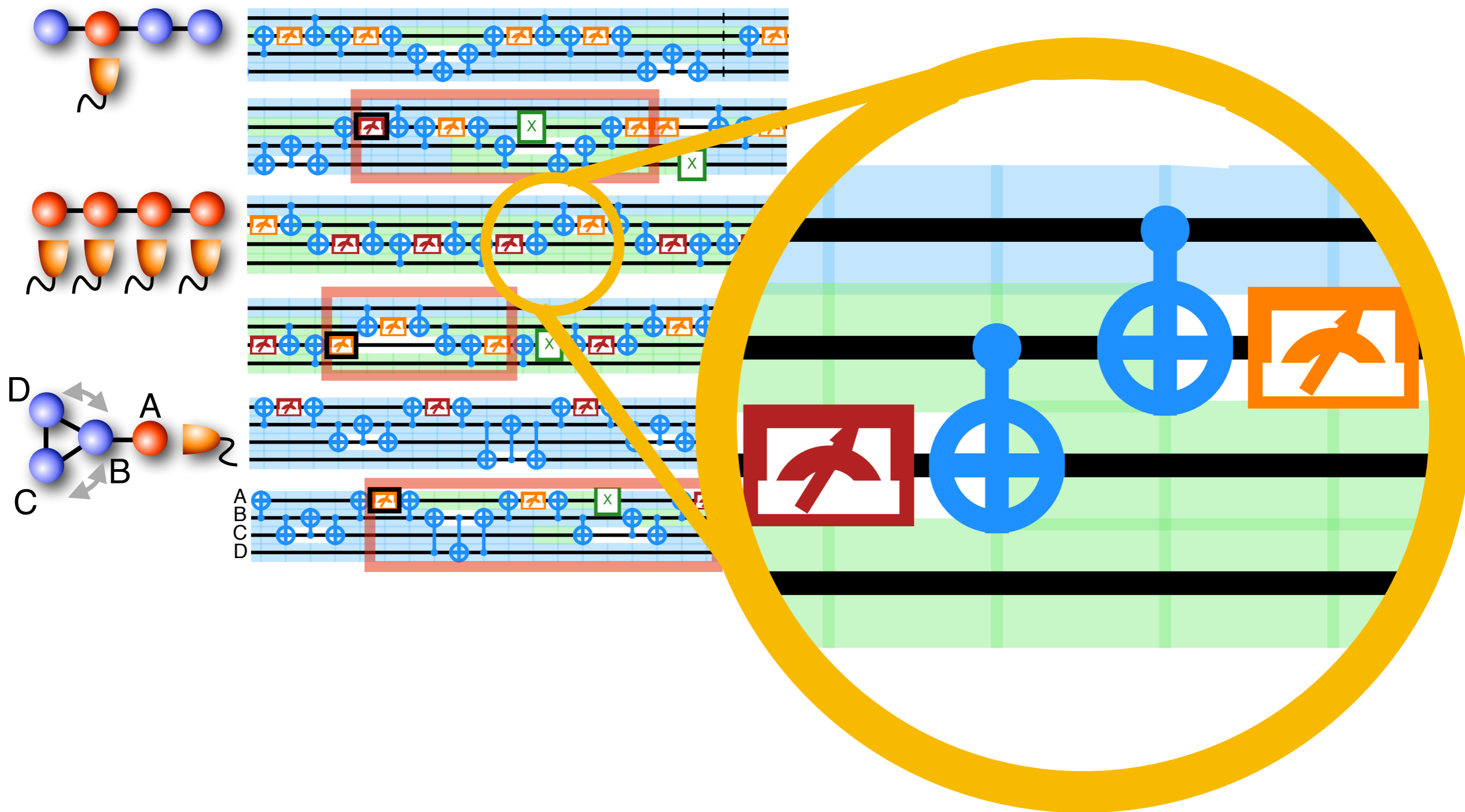
Different topologies



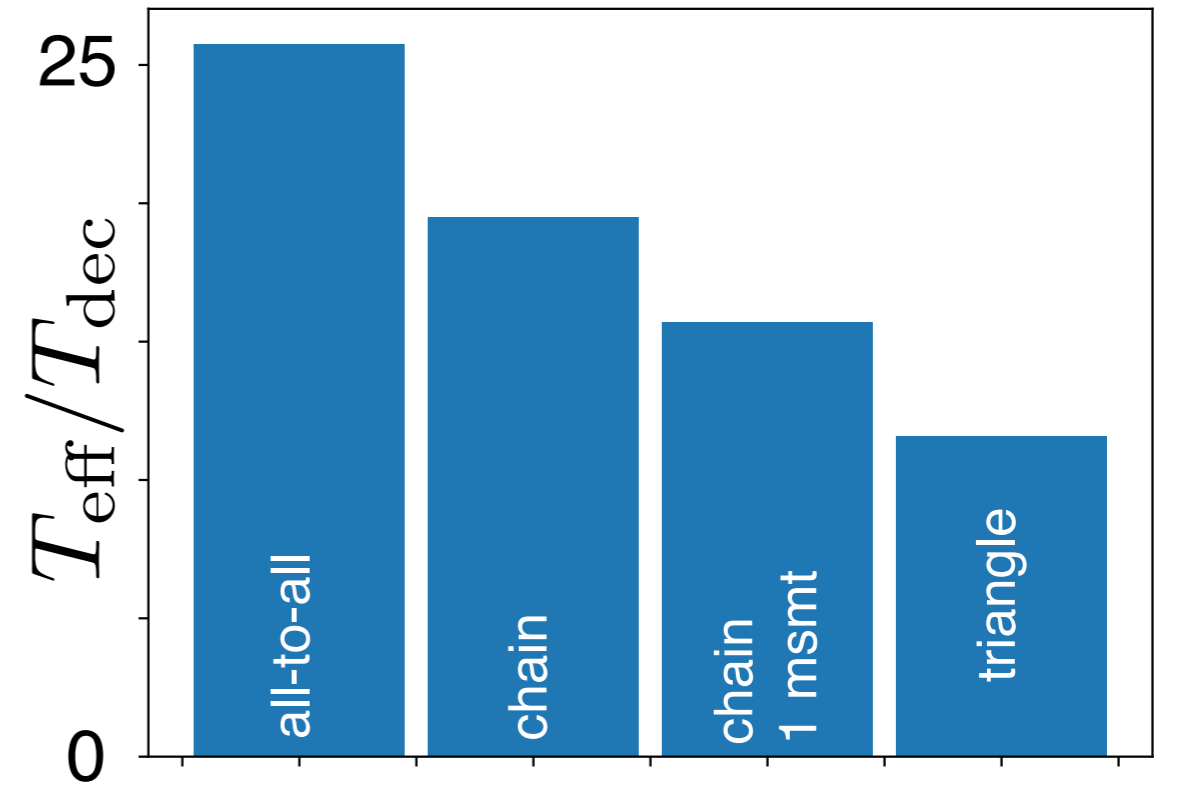
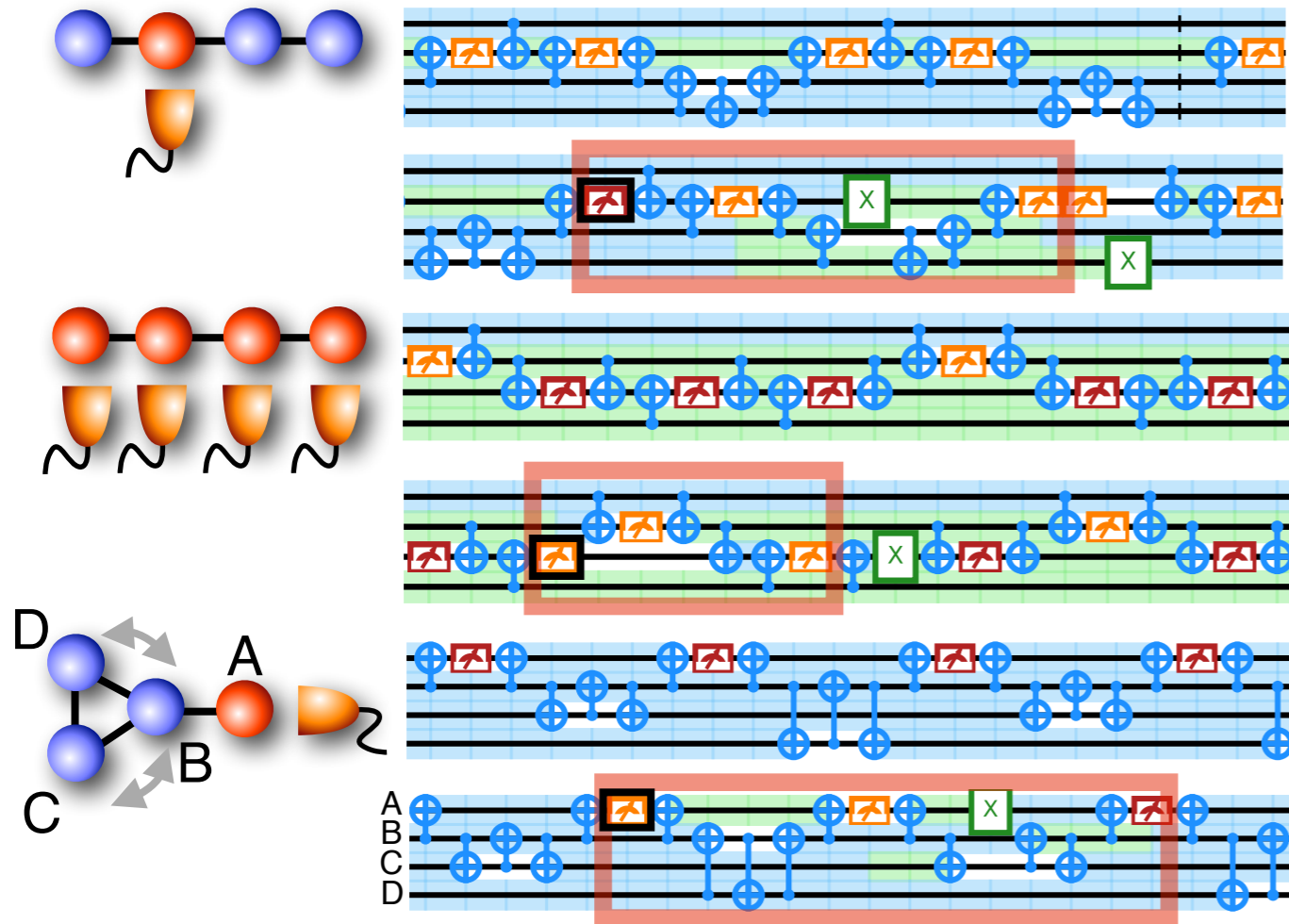
Different topologies



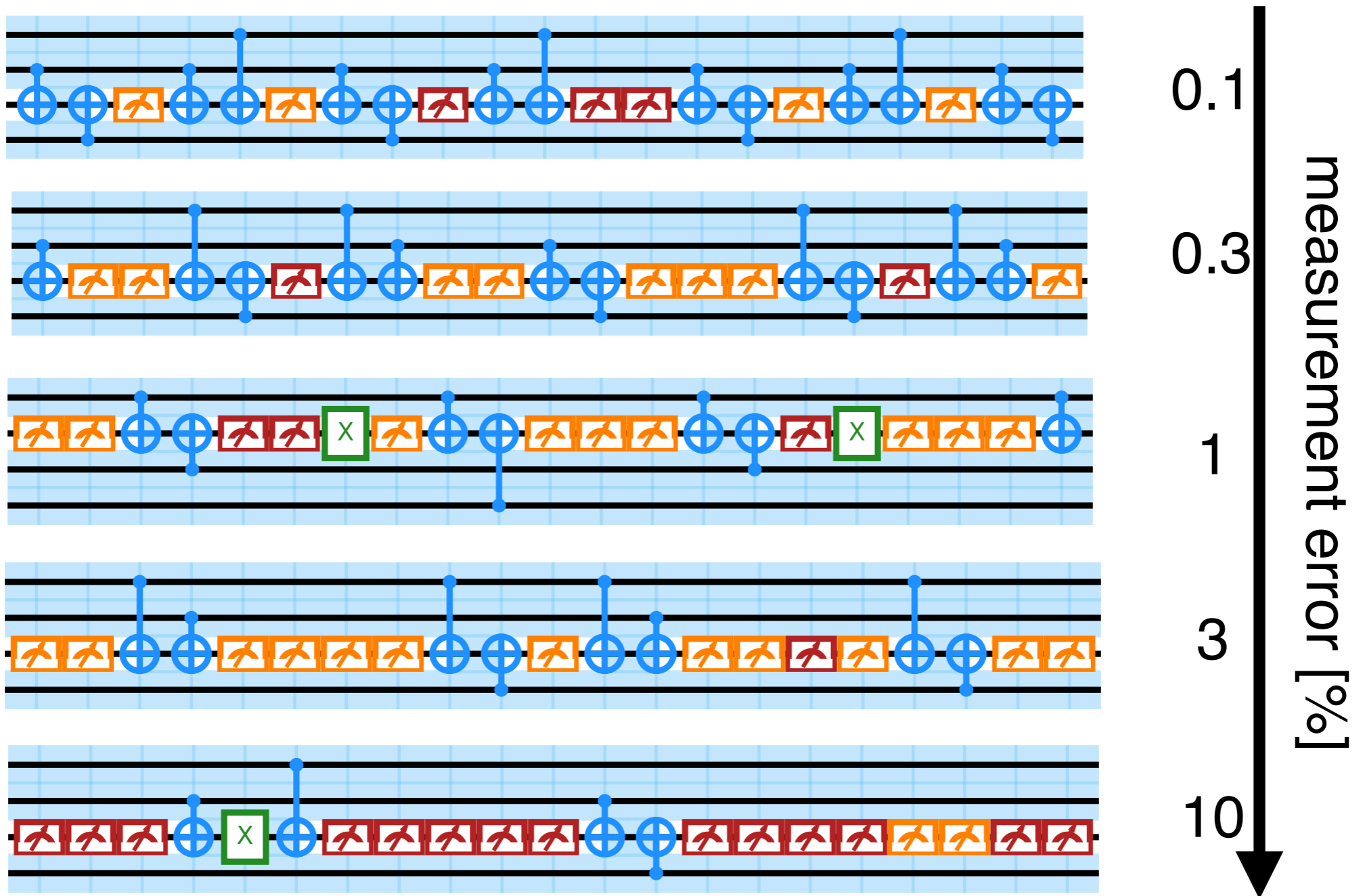
Different topologies



Different topologies

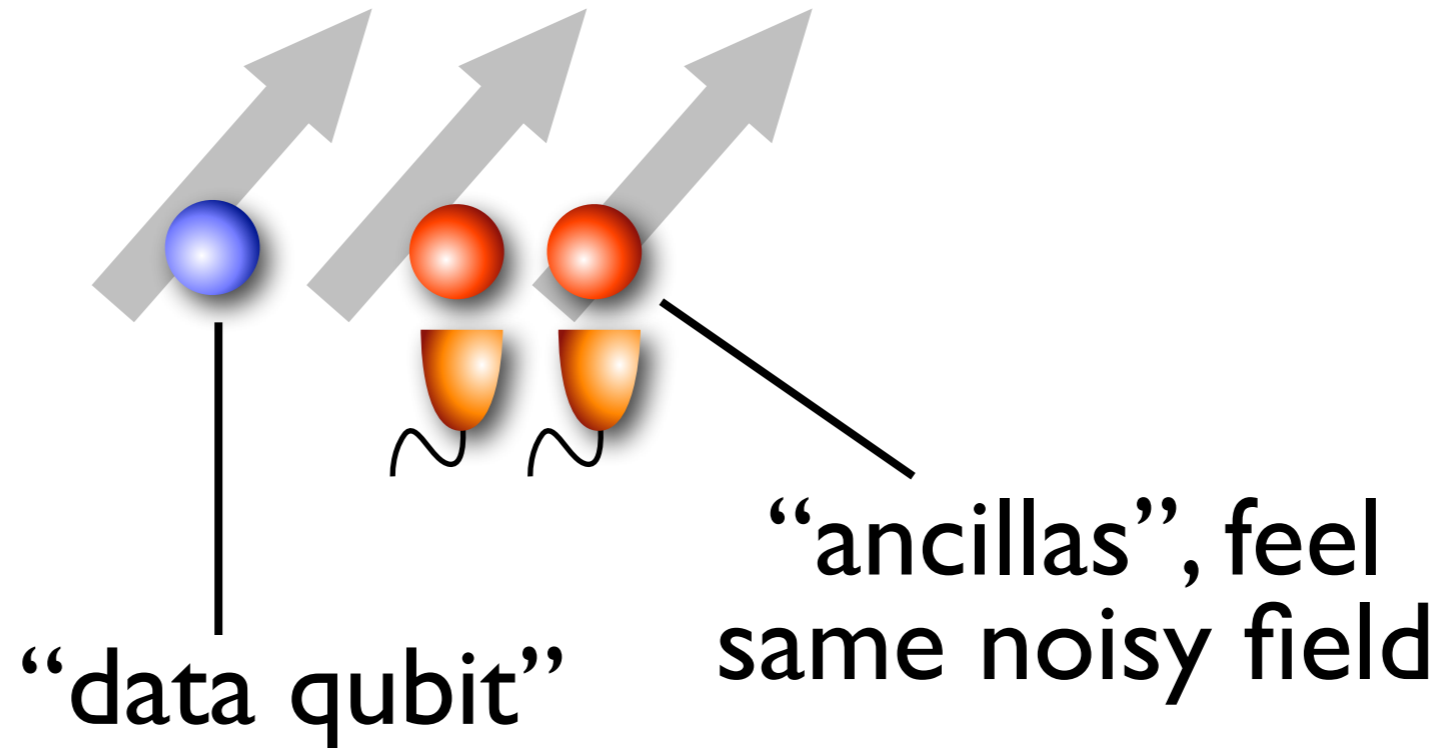


Example: measurement errors

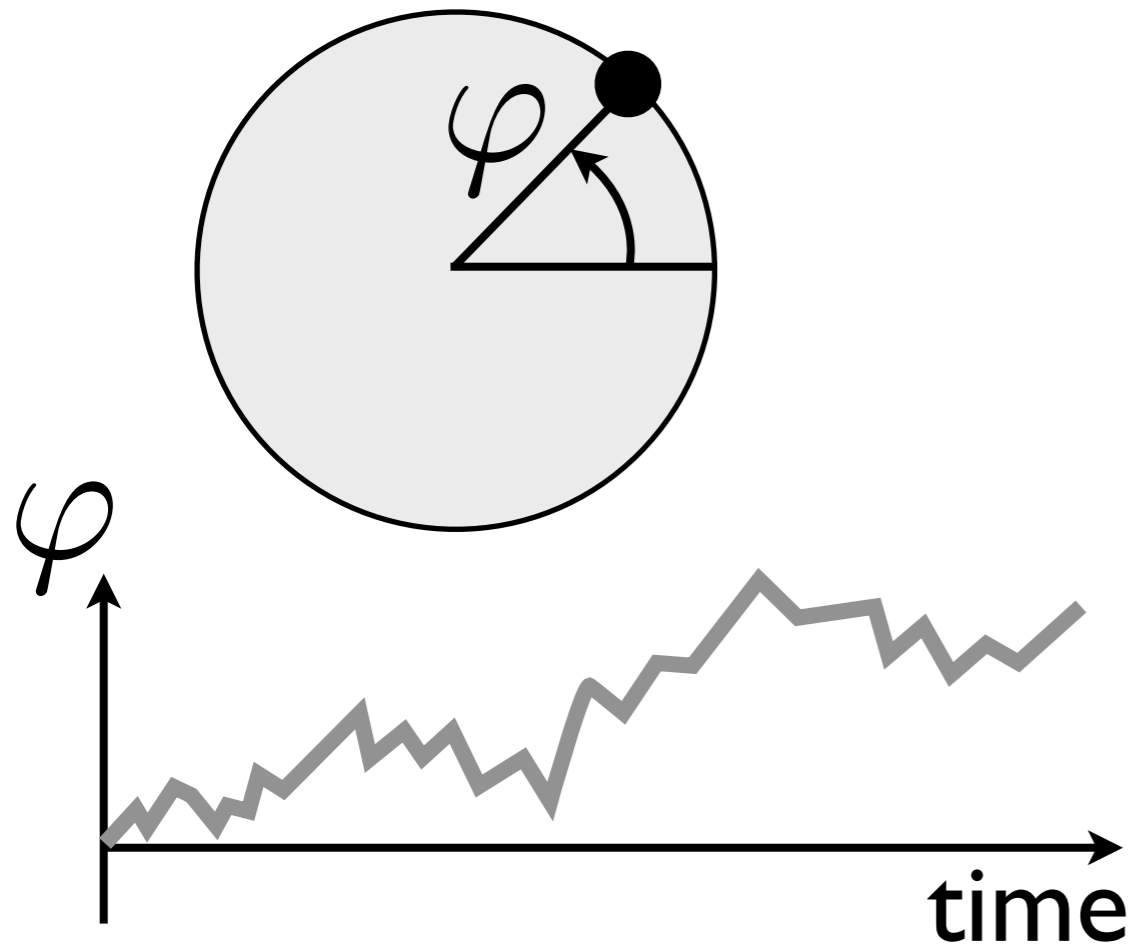
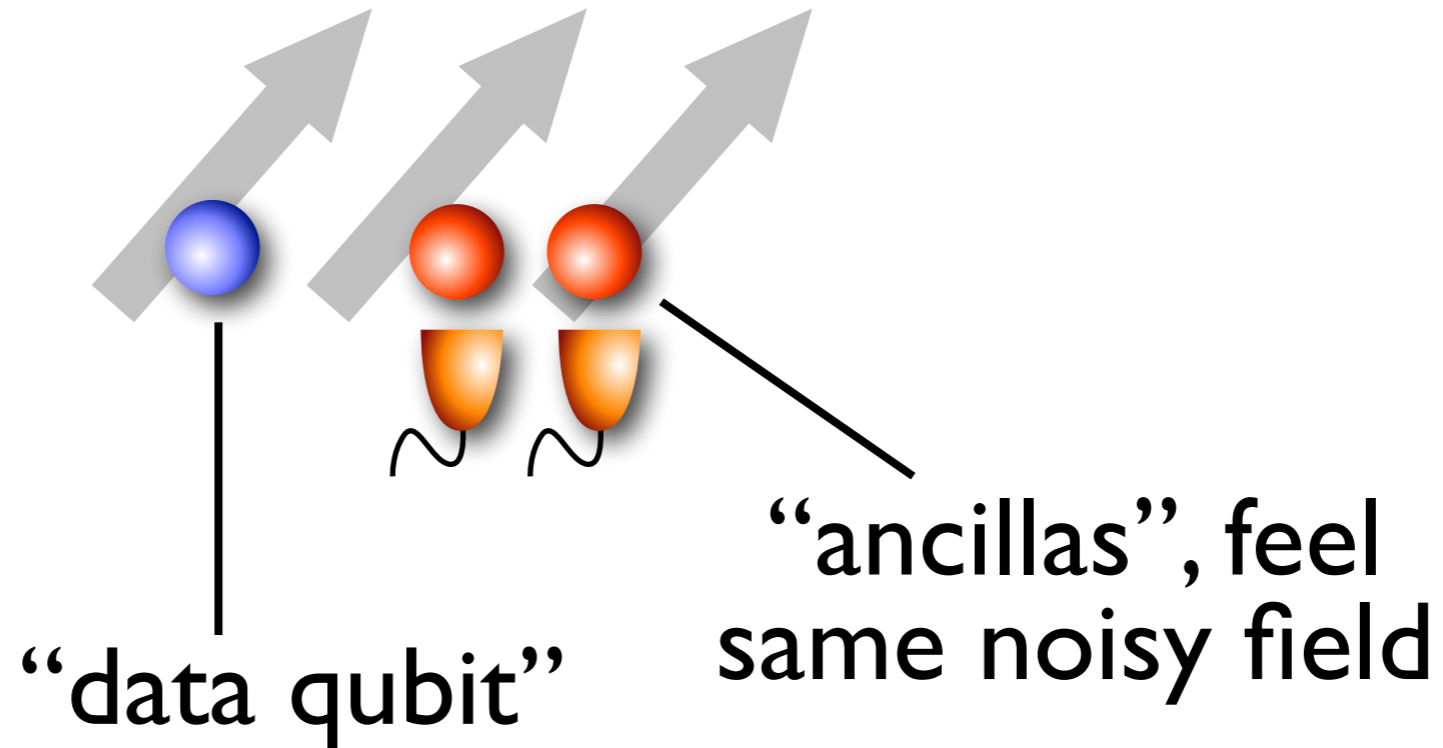


The exact same program also discovers strategies that are unrelated to stabilizer codes...

Different class of scenarios: Dephasing by a noisy field



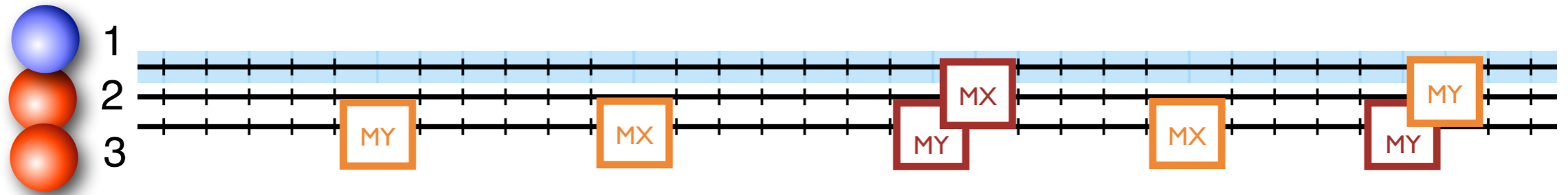
Different class of scenarios: Dephasing by a noisy field



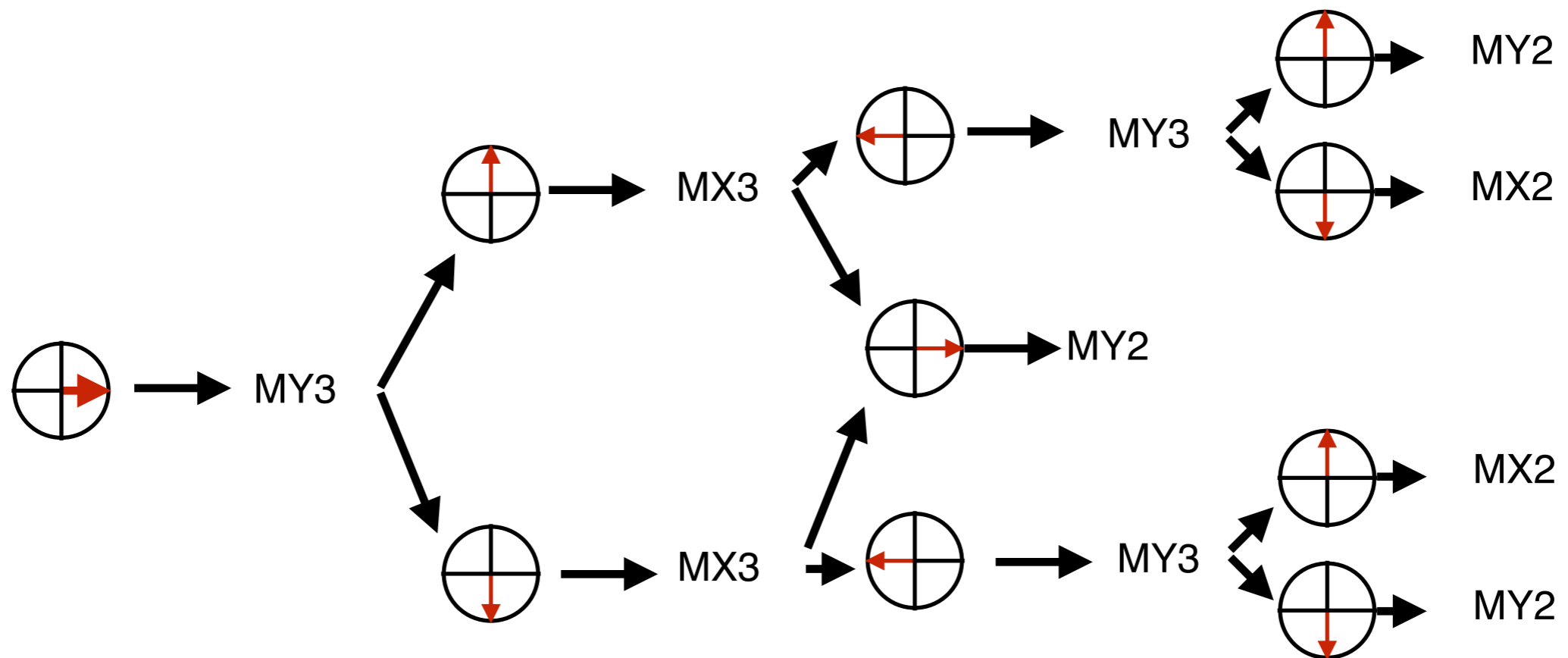
Collective dephasing:

$$\hat{H}(t) = B(t) \sum_j \mu_j \hat{\sigma}_{zj}$$

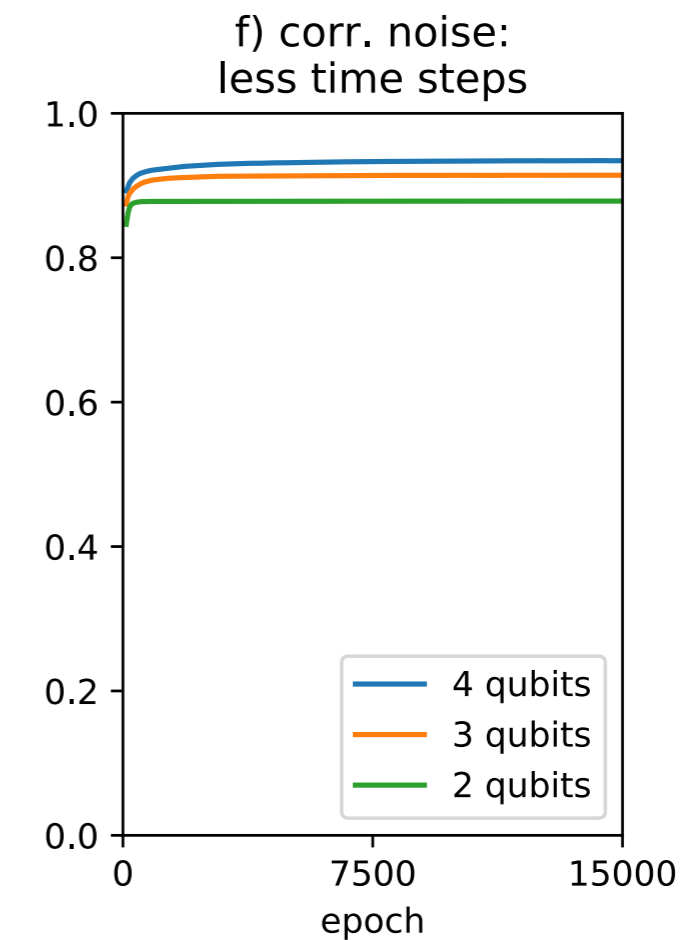
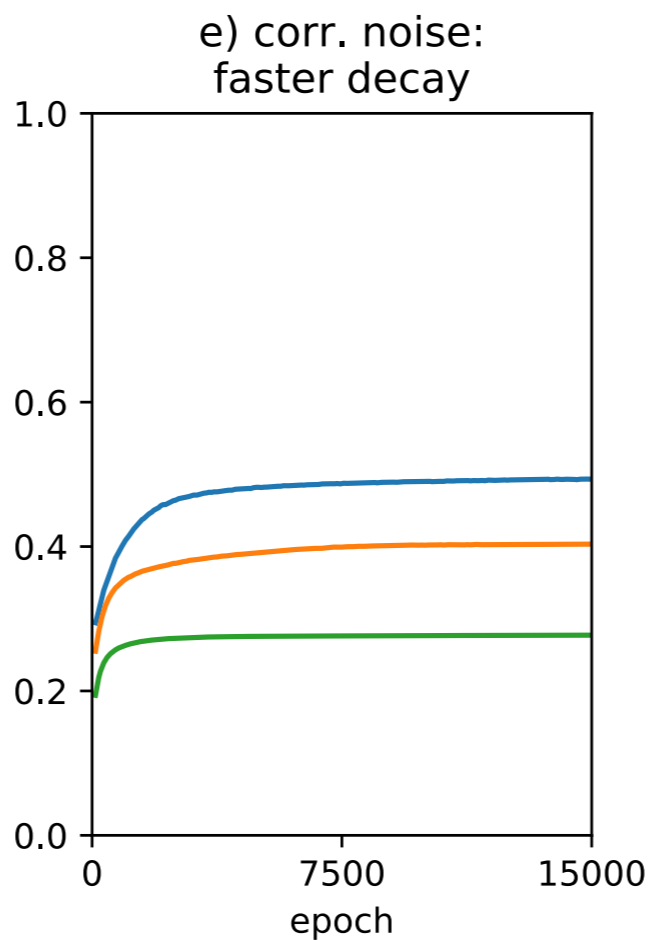
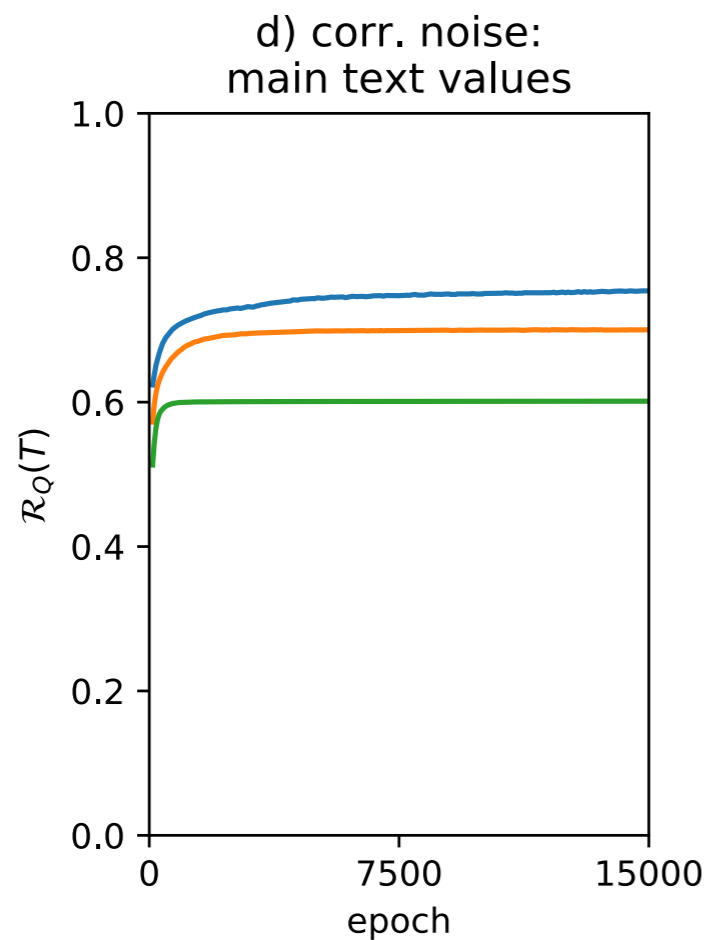
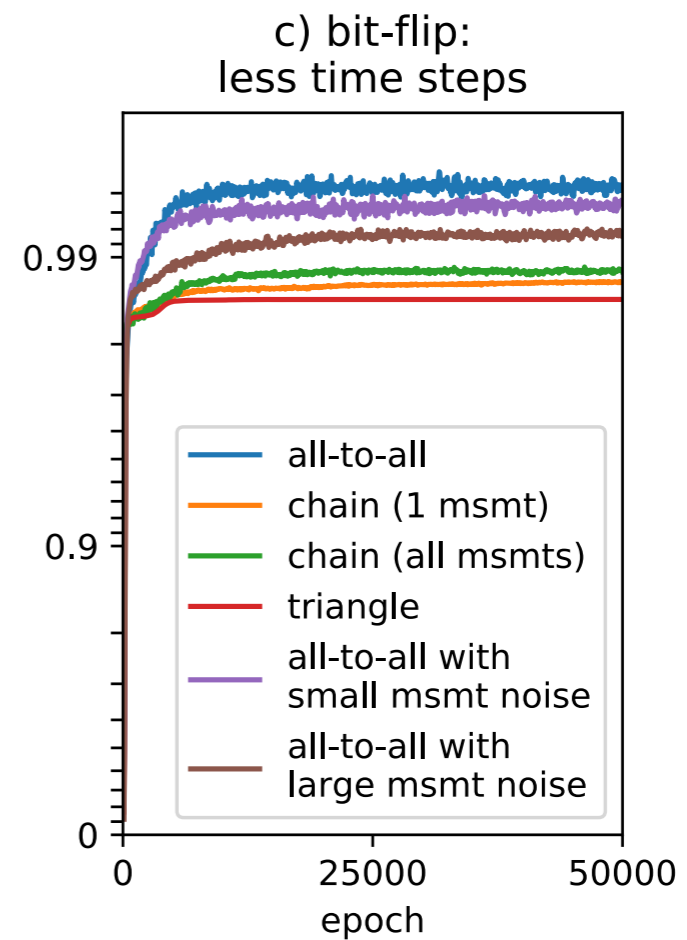
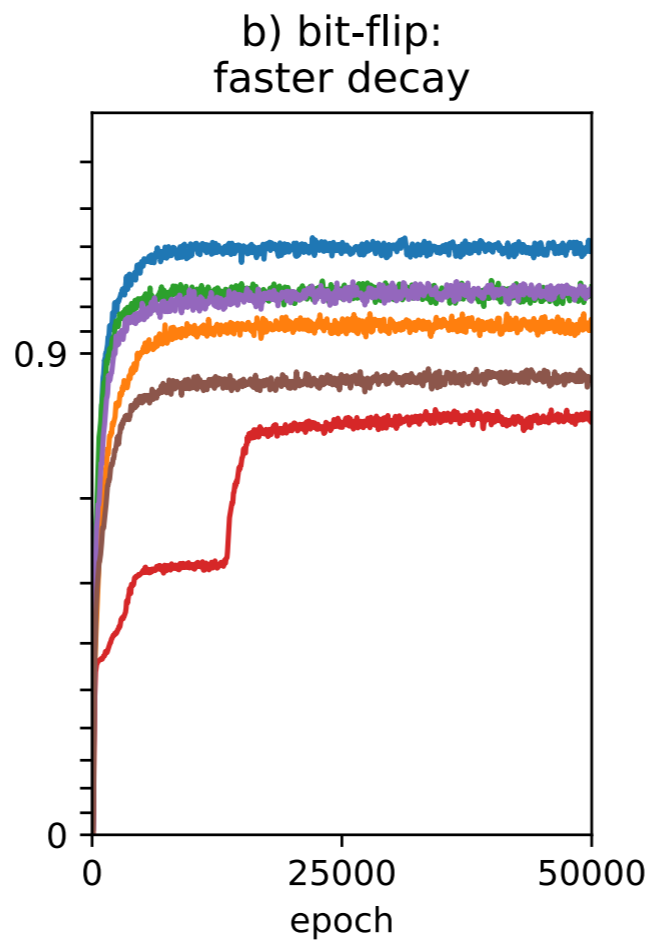
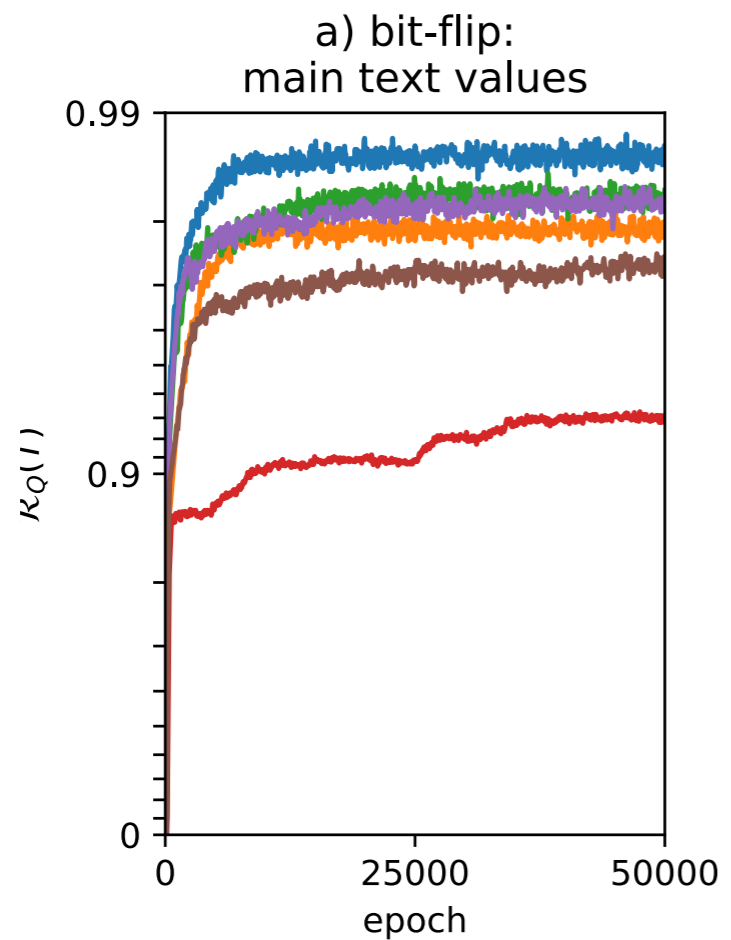
Network measures ancillas to estimate noisy field



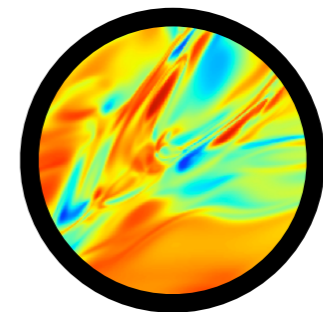
Network discovers **adaptive** noise estimation strategy
Strategy = decision tree



We can use the same
"hyperparameters" (network
structure, learning rate, ...) for
all these tasks



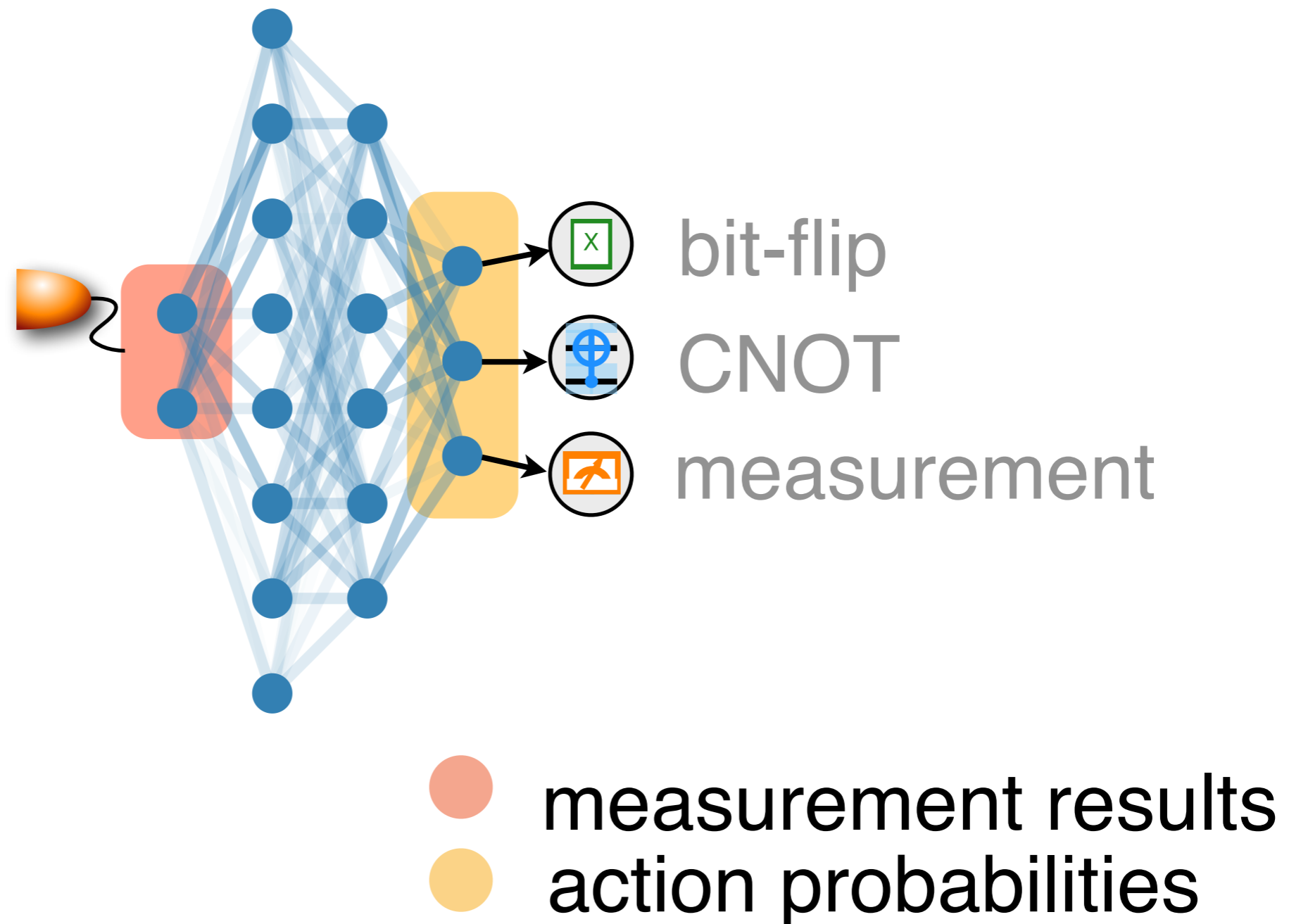
Challenges and Solutions



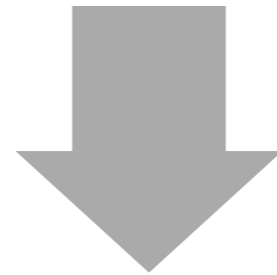
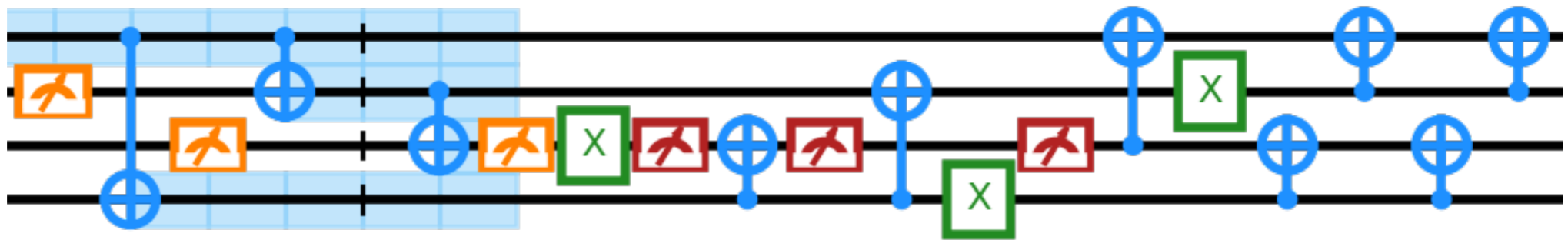
Naive approach *will not work

Reward: Overlap $|\langle \Psi(0) | \Psi_f \rangle|$
(averaged over initial states)

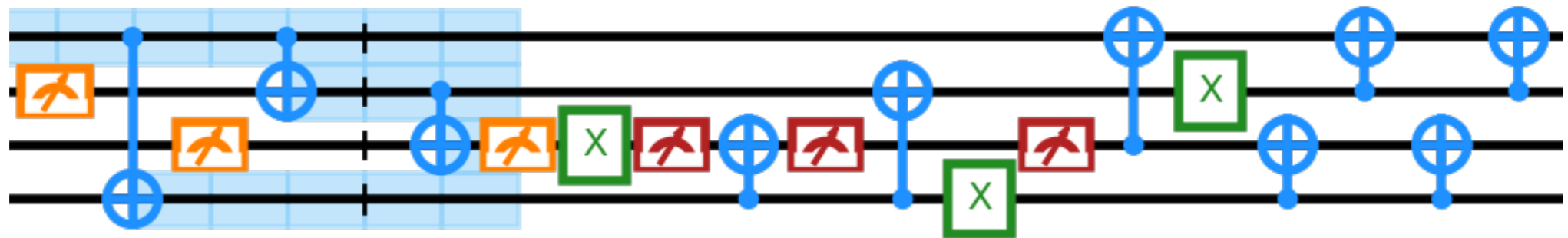
Policy network: actions depending on measurements



Naive approach *will not work



Naive approach *will not work



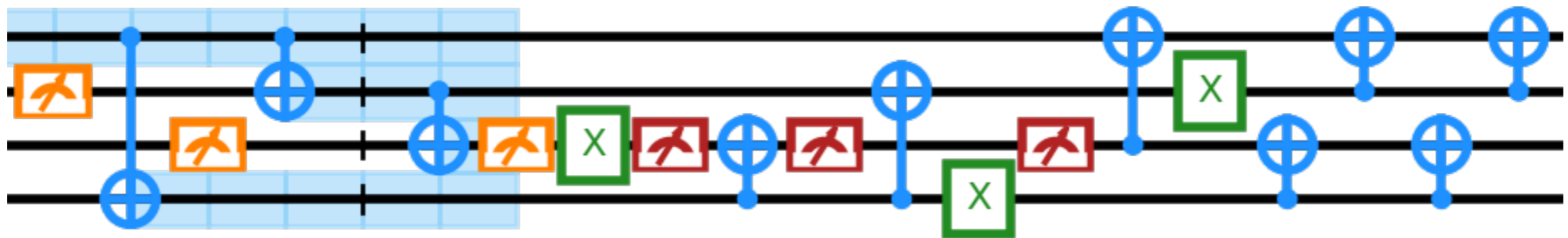
Problem 1: Combinatorial explosion

Shortest useful gate sequences already quite long
e.g. for 20 possible gates, in 10 time steps

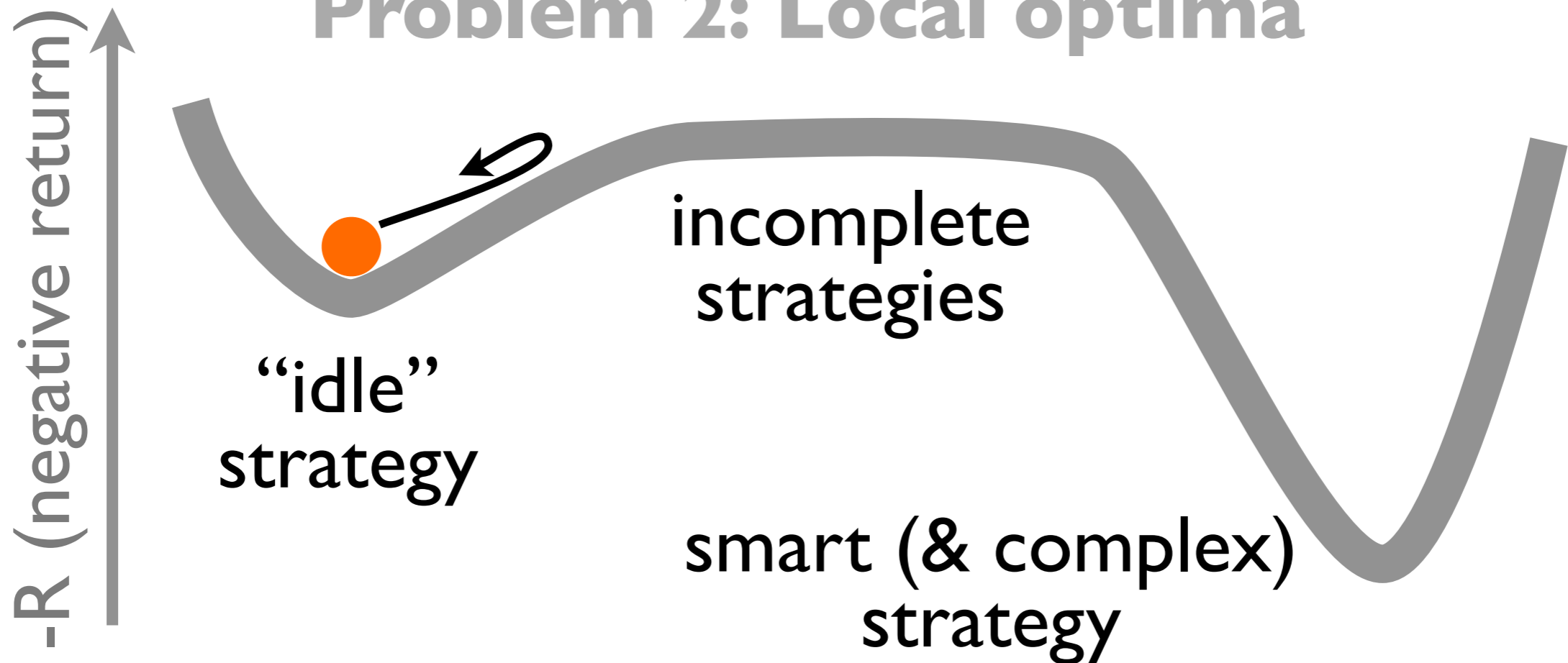
$$20^{10} \text{ possibilities}$$

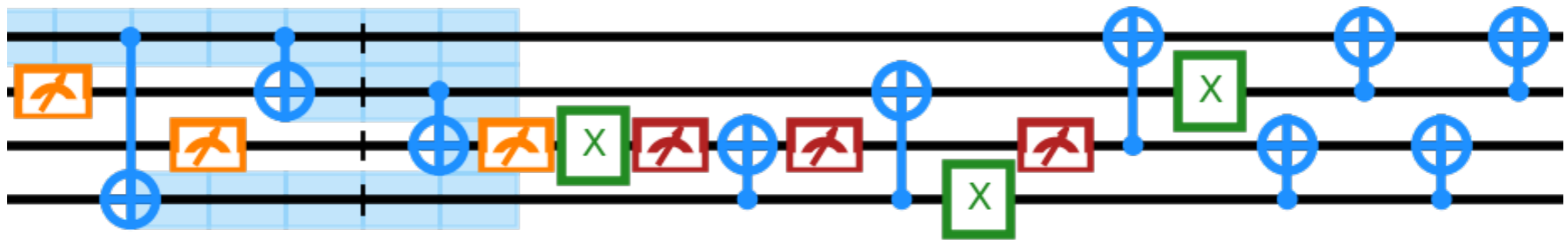
we will later consider sequences of 200 time steps!

Naive approach *will not work



Problem 2: Local optima

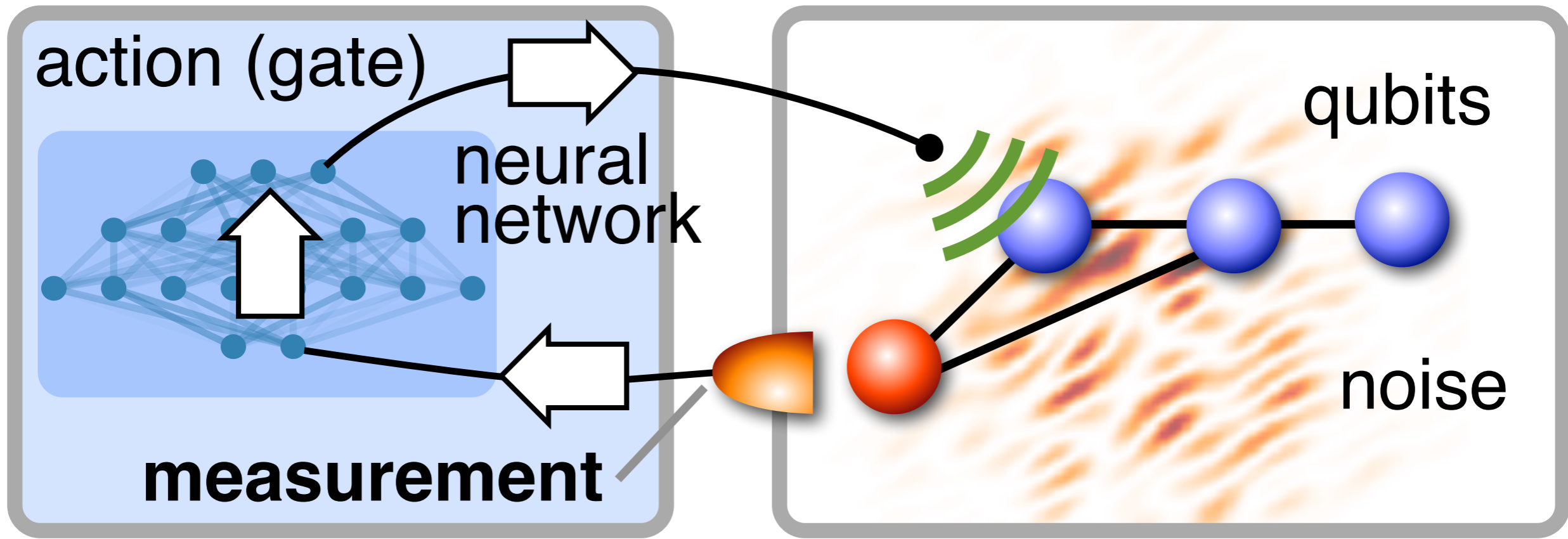




Two key concepts

As much information as possible

Construct smart reward



RL-agent

RL-environment

1

0

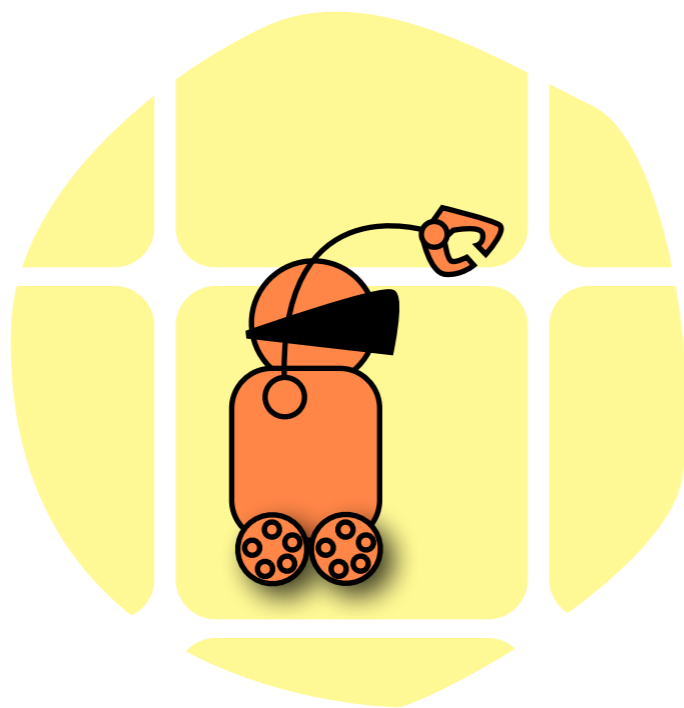
1

1

only measurement results!

time

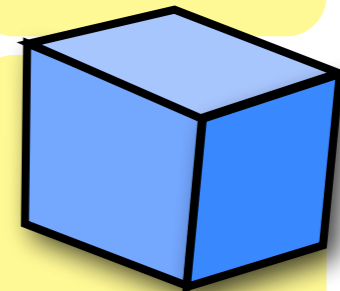
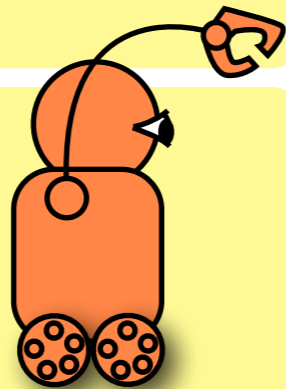
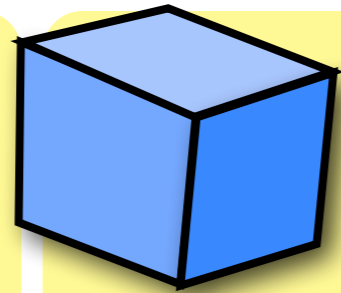
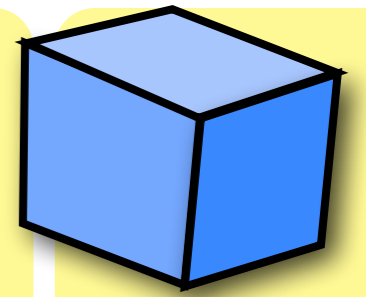
?



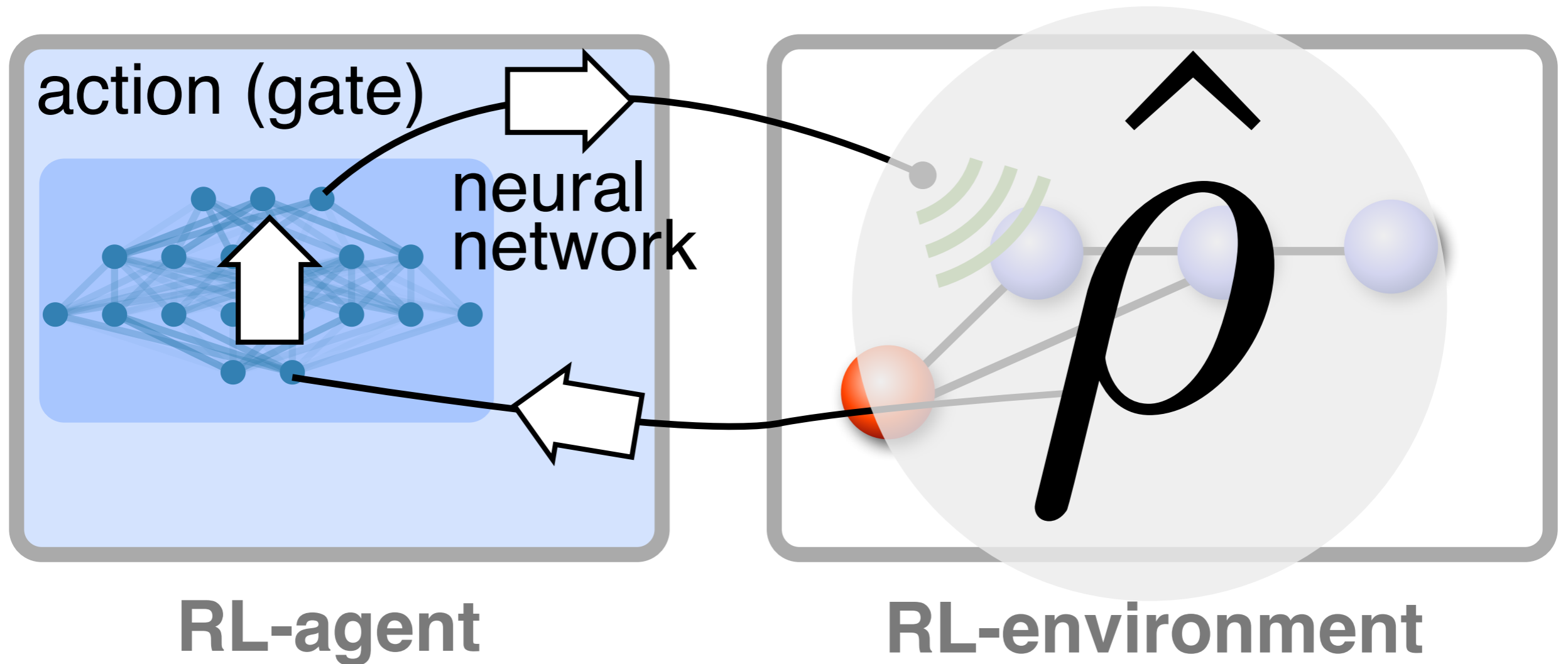
?

?

“As much information as possible”



Quantum state as input!



But wait! Isn't this cheating?

In an experiment, we cannot do this!

And the agent now knows the quantum state to preserve!!

Want to preserve **arbitrary** quantum state!

Consider “**completely positive map**” that describes the dissipative evolution of the whole quantum system

$$\hat{\rho}(t) = \Phi[\hat{\rho}(0)]$$

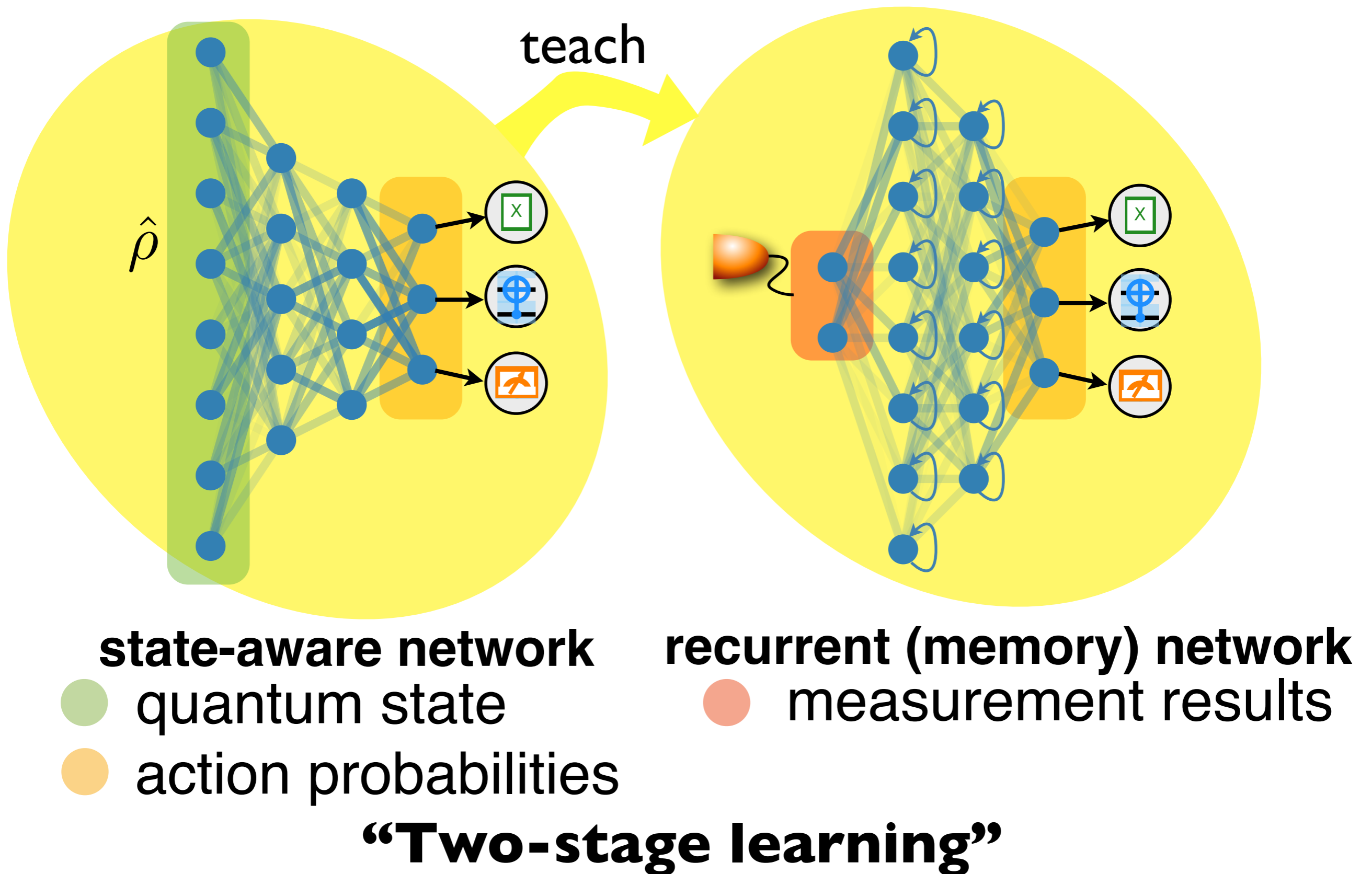
$$\hat{\rho}(0) = \frac{1}{2} (1 + x\hat{\sigma}_{x1} + y\hat{\sigma}_{y1} + z\hat{\sigma}_{z1}) \otimes \hat{\rho}_{\text{Rest}}$$

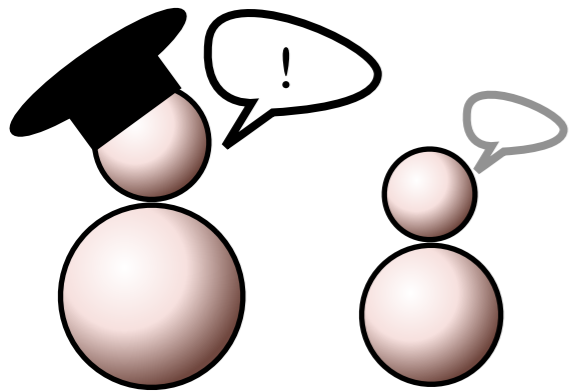
$\vec{n} = (x, y, z)$ **Bloch vector of logical qubit state**

In practice: need to evolve only four different density matrices simultaneously; feed all of them to the network.

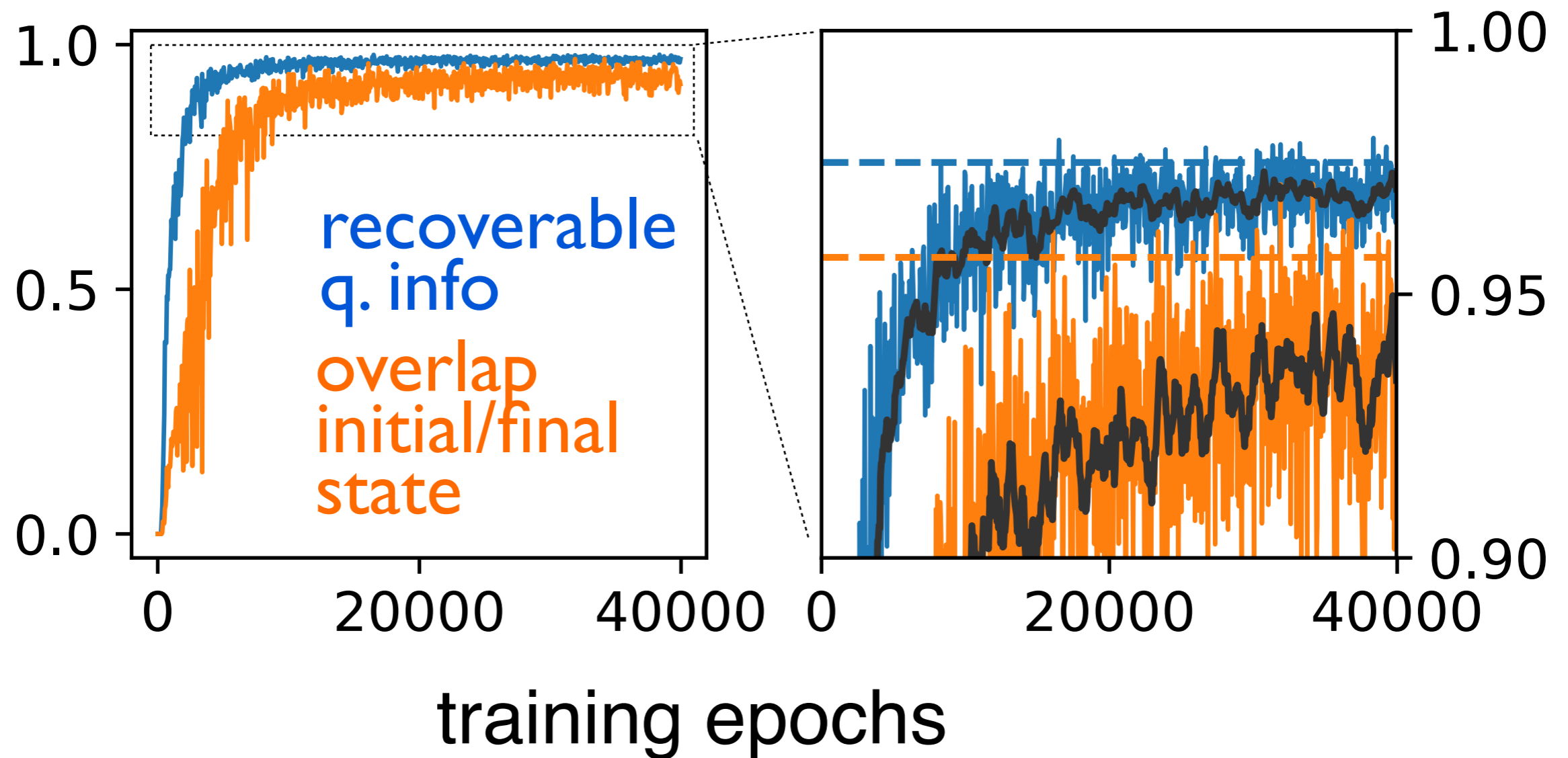
Ask network to preserve arbitrary state using **the same** gate sequence!

In an experiment, we only have access to measurement results!

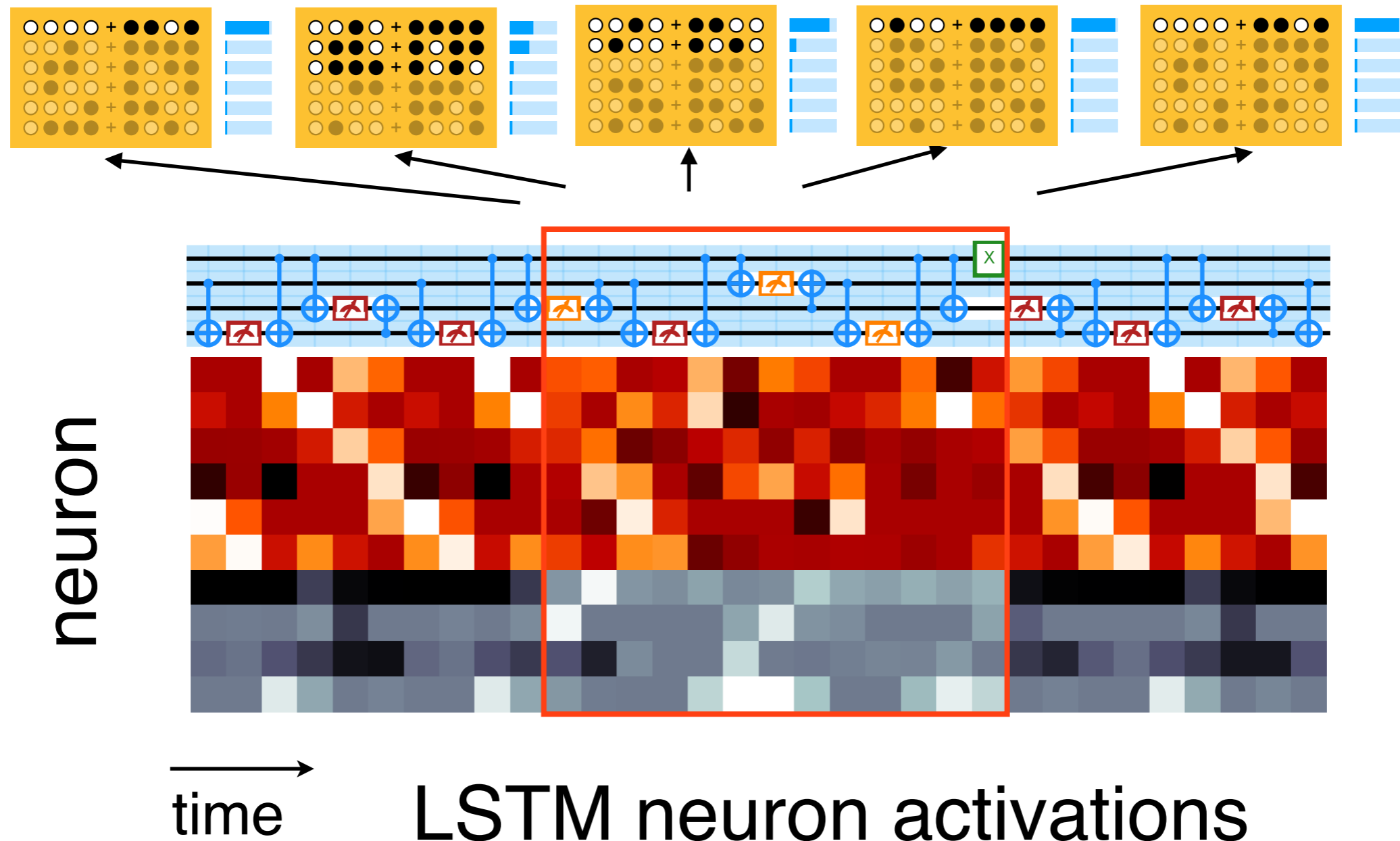




Training the recurrent network
...works very well and reliable!



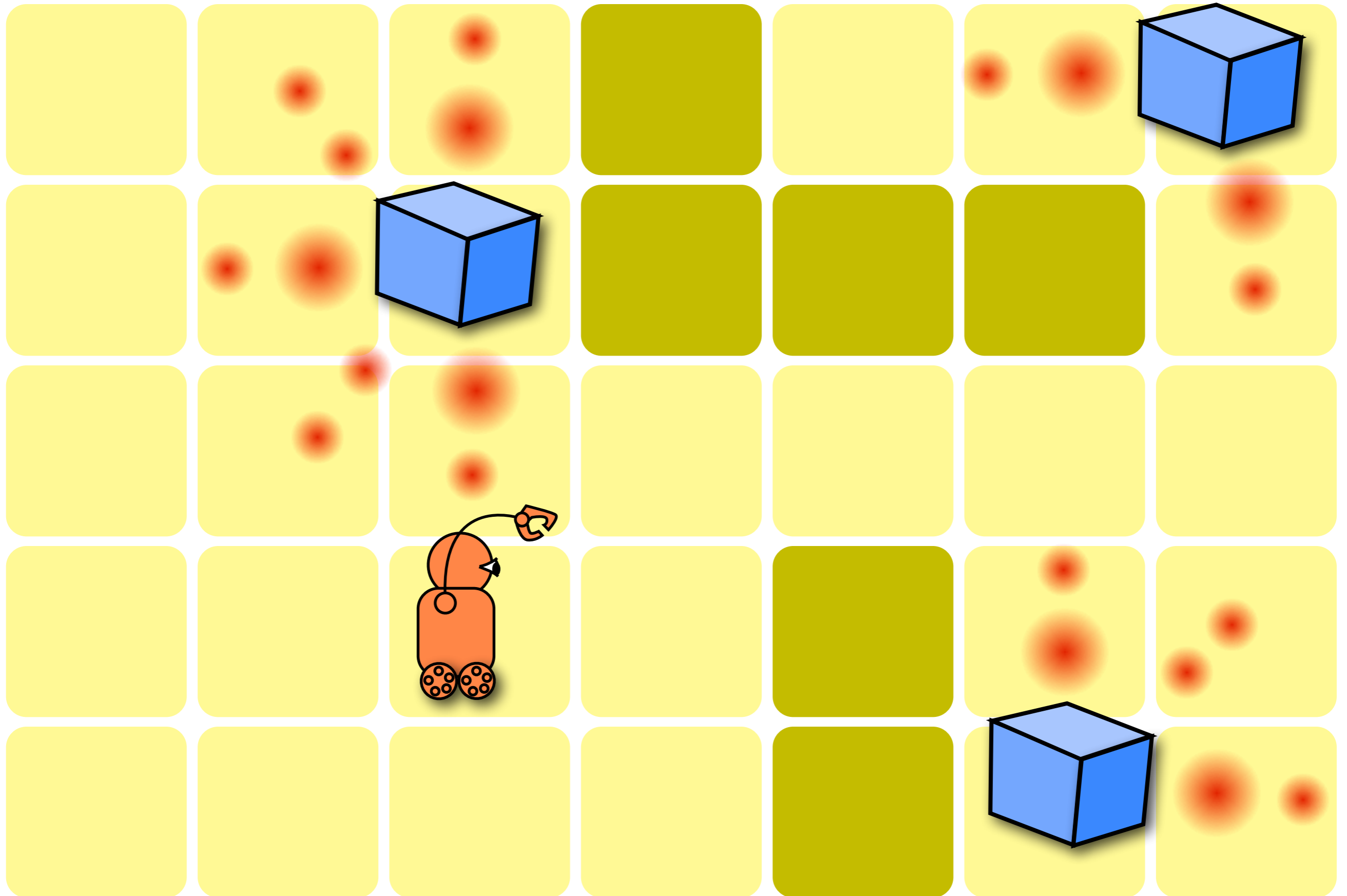
Analyzing the recurrent network



LSTM=long short-term memory (Schmidhuber, Hochreiter)

identify “switches” and “counters”

“Smart reward scheme”



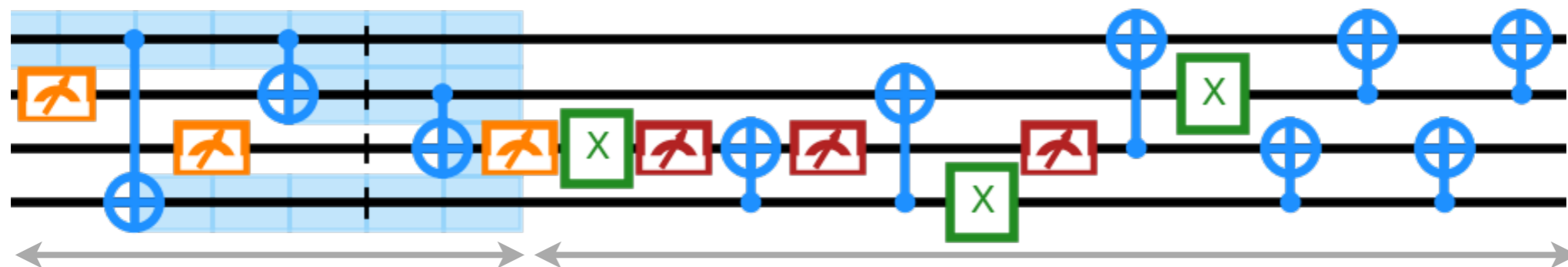
“Smart reward scheme”

Find a general measure of the amount of quantum information that can still be retrieved from a multi-qubit device...

...after complex entangling gate sequences!
...in the presence of noise!

...using some smart error detection/
correction scheme!

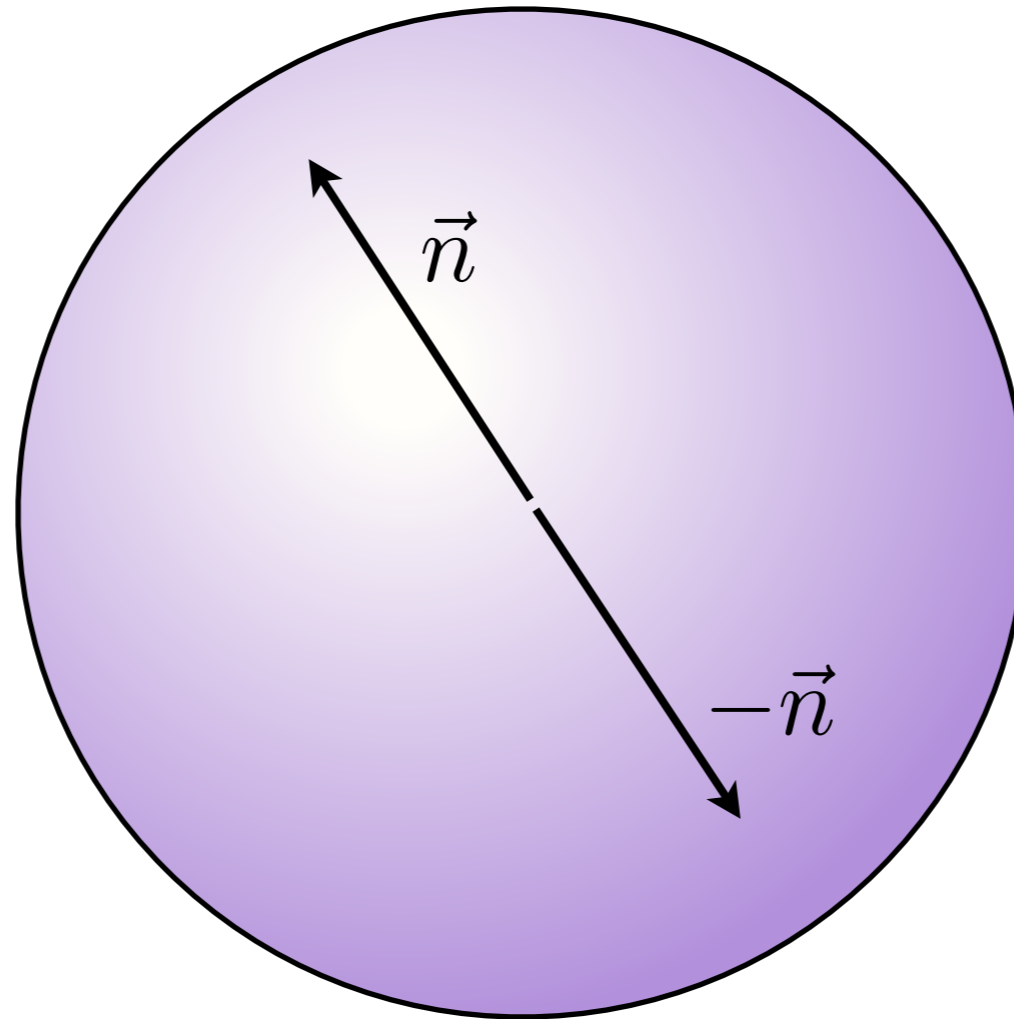
(without knowing that scheme!)



“amount of quantum info”

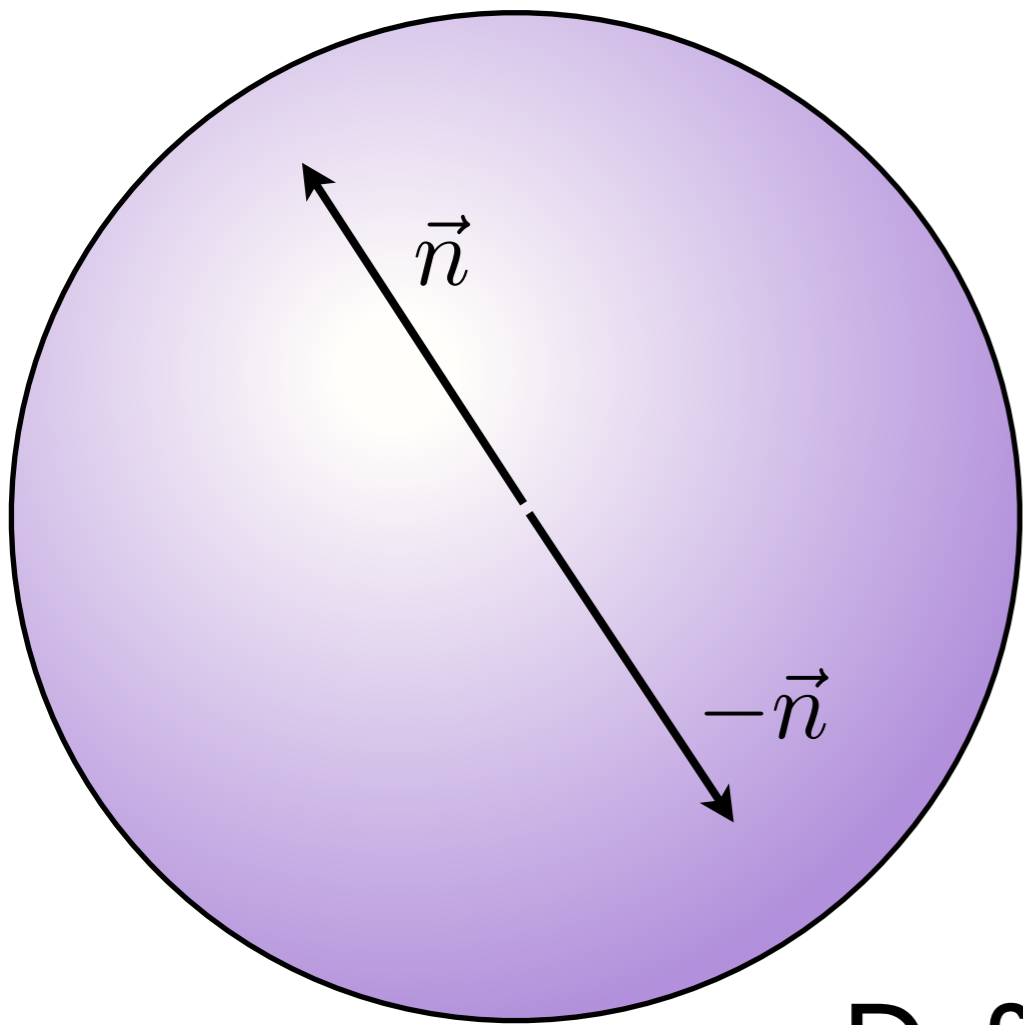
zero

Idea: initially orthogonal states
should remain distinguishable!



probability to distinguish by optimal measurement:

$$\frac{1}{2} \|\hat{\rho}_{\vec{n}} - \hat{\rho}_{-\vec{n}}\|_1$$



Define

“Recoverable Quantum Information”

$$\mathcal{R}_Q = \min_{\vec{n}} \frac{1}{2} \|\hat{\rho}_{\vec{n}} - \hat{\rho}_{-\vec{n}}\|_1$$

(worst-case initial state defines success of quantum memory)

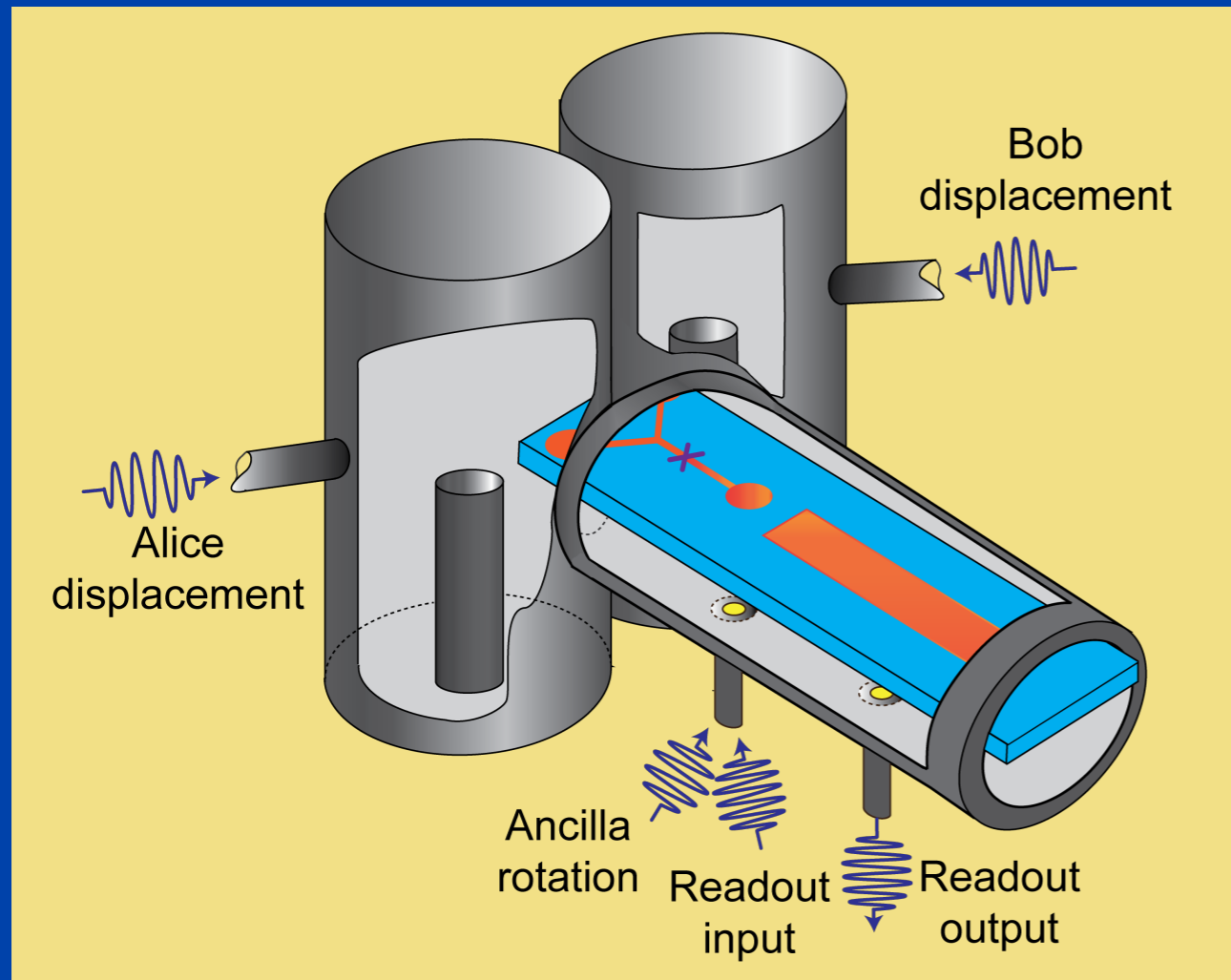
Key result:

**network discovering from scratch
quantum error correction
strategies based on feedback**

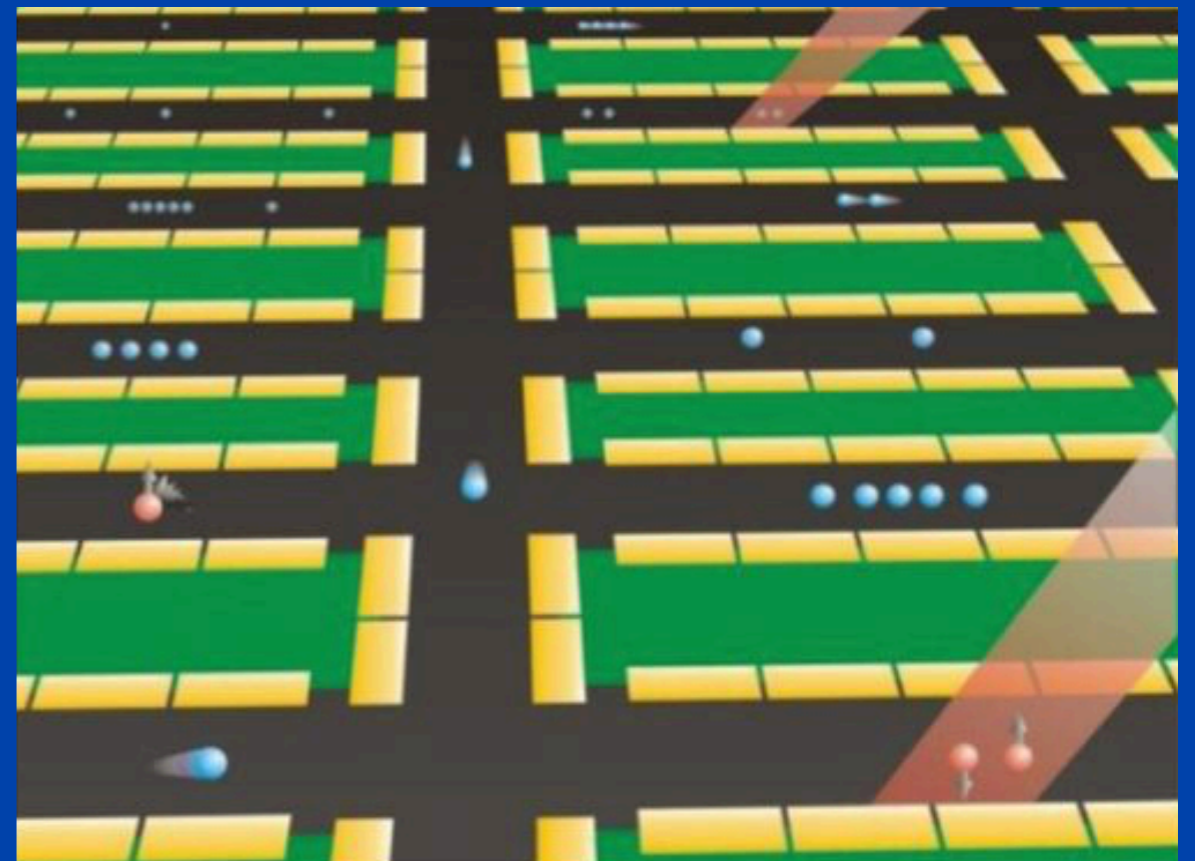
Physical Review X 031084 (2018)

Thomas Fösel, Petru Tighineanu, Talitha Weiss, FM

future: apply to other physical settings



(Schoelkopf, Devoret lab 2016)



(Monroe, Kim Science 2013)

- More efficient physics simulation
- CHZ acceleration

- Other RL algorithms
- Monte Carlo Tree Search

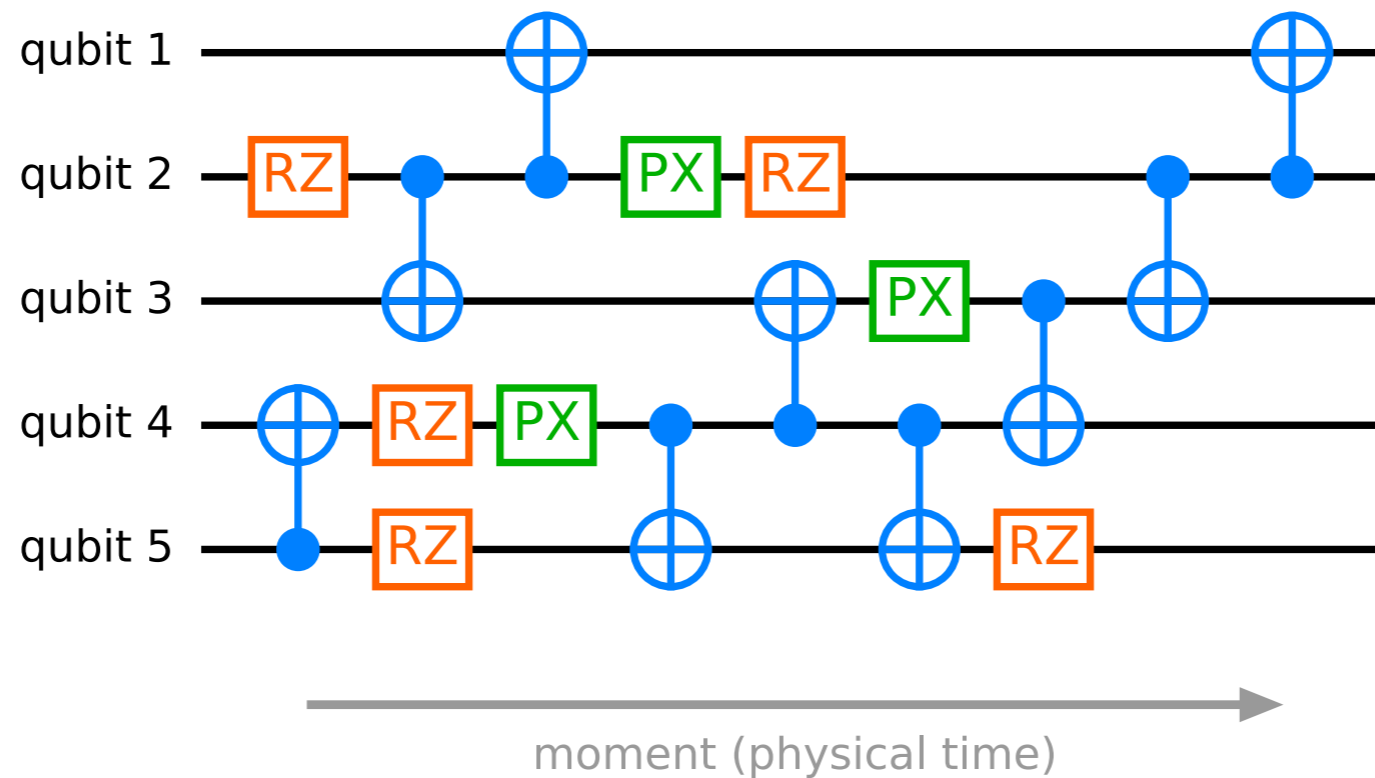
- experiments? need fast NN & feedback!
- FPGA hardware implementations?

Case study:
Reinforcement learning
for quantum circuit
optimization

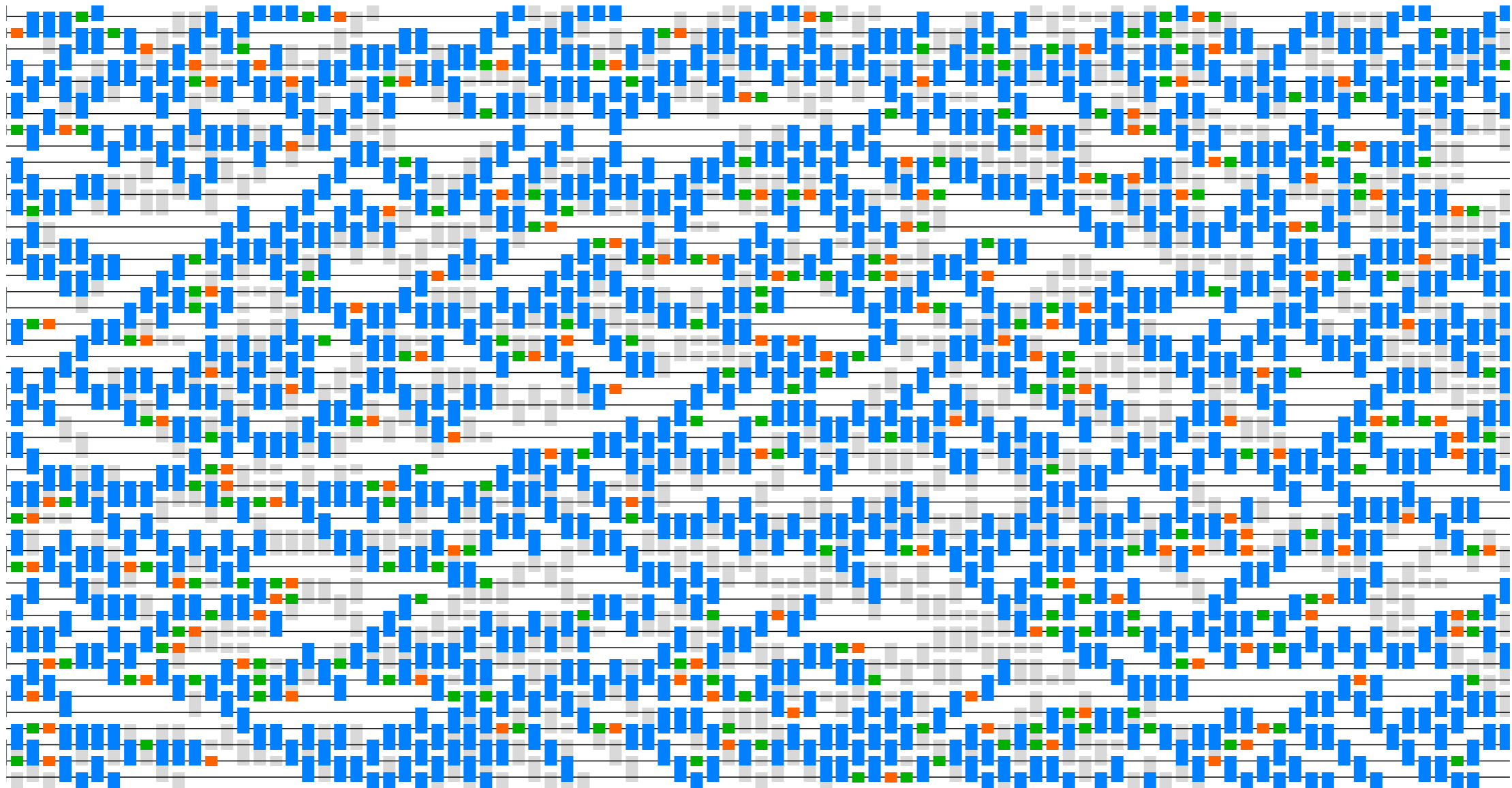
arXiv 2103.07585

Thomas Fösel, Murphy Yuezhen Niu, Florian Marquardt, and Li Li

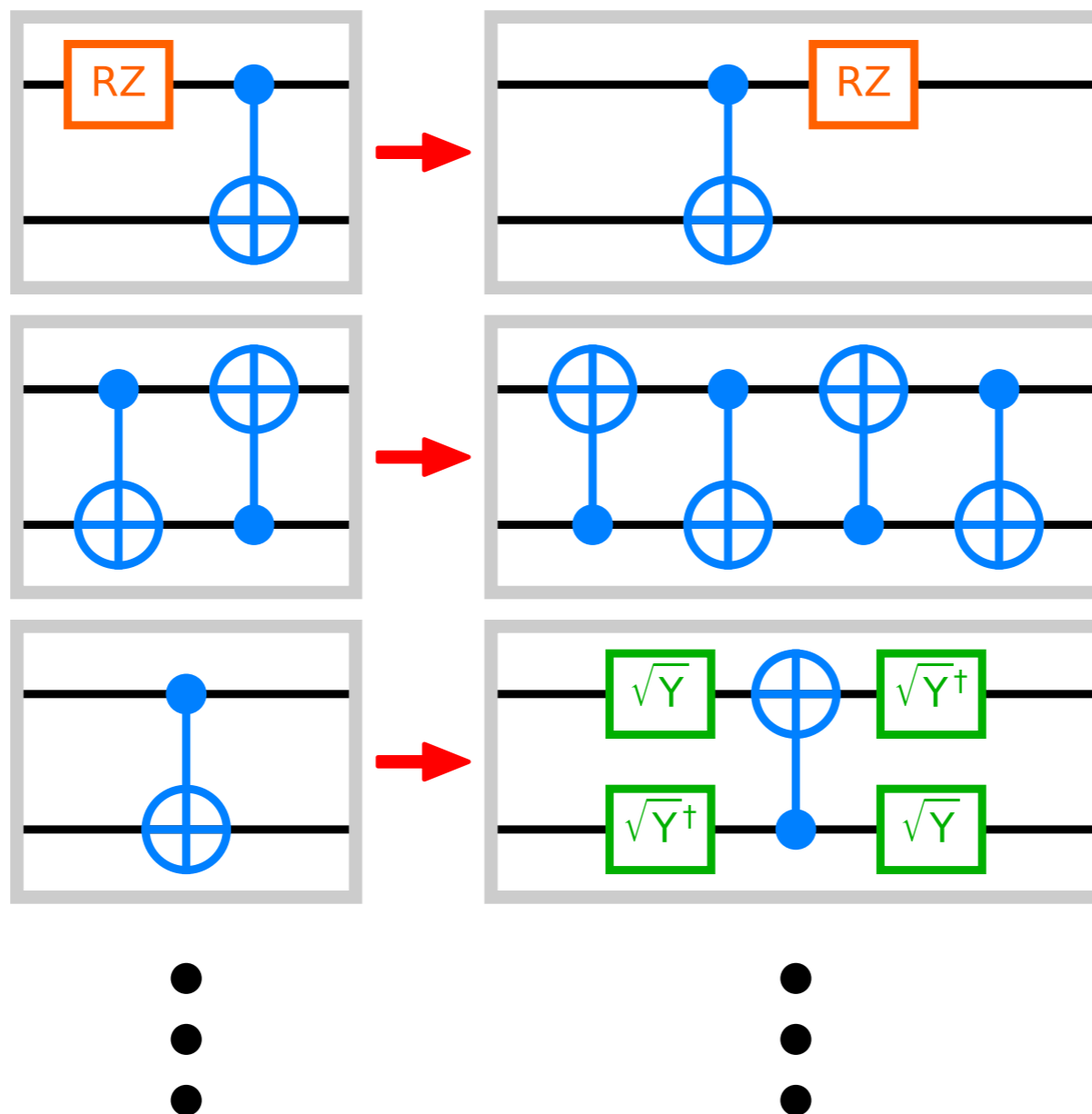
Quantum Circuit Optimization: reduce gate count / depth / etc.!



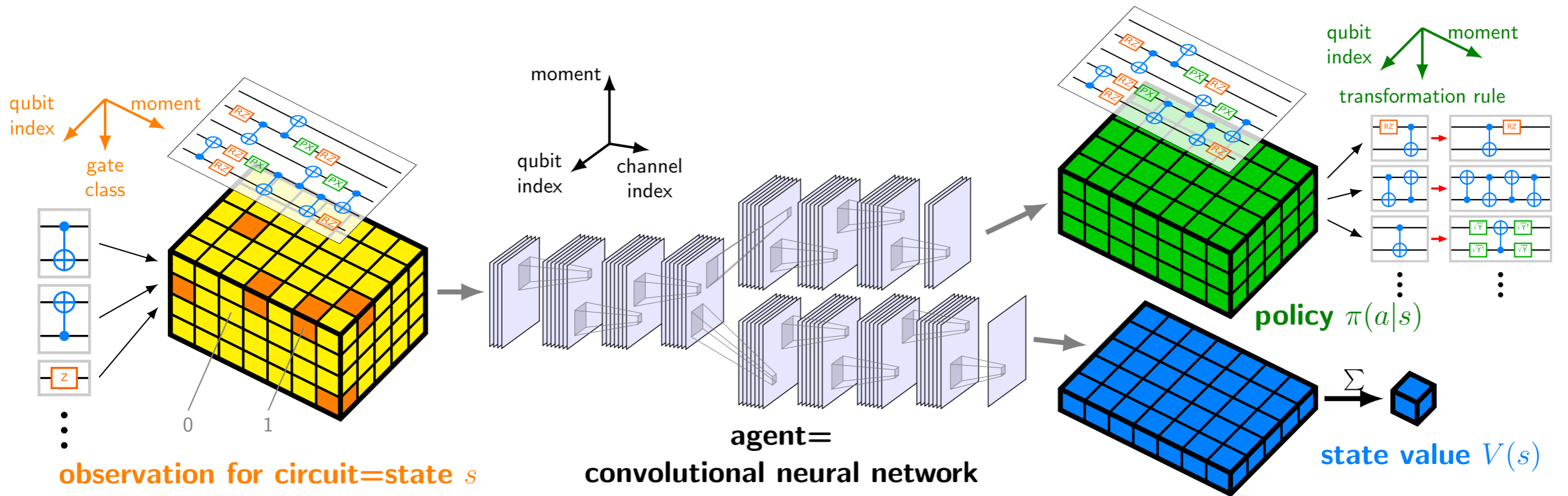
NISQ devices: Quantum Circuit Optimization critical
...and needs to be hardware-dependent
[not on an abstract level designed for large-scale
fault-tolerant circuits]



Transformation rules



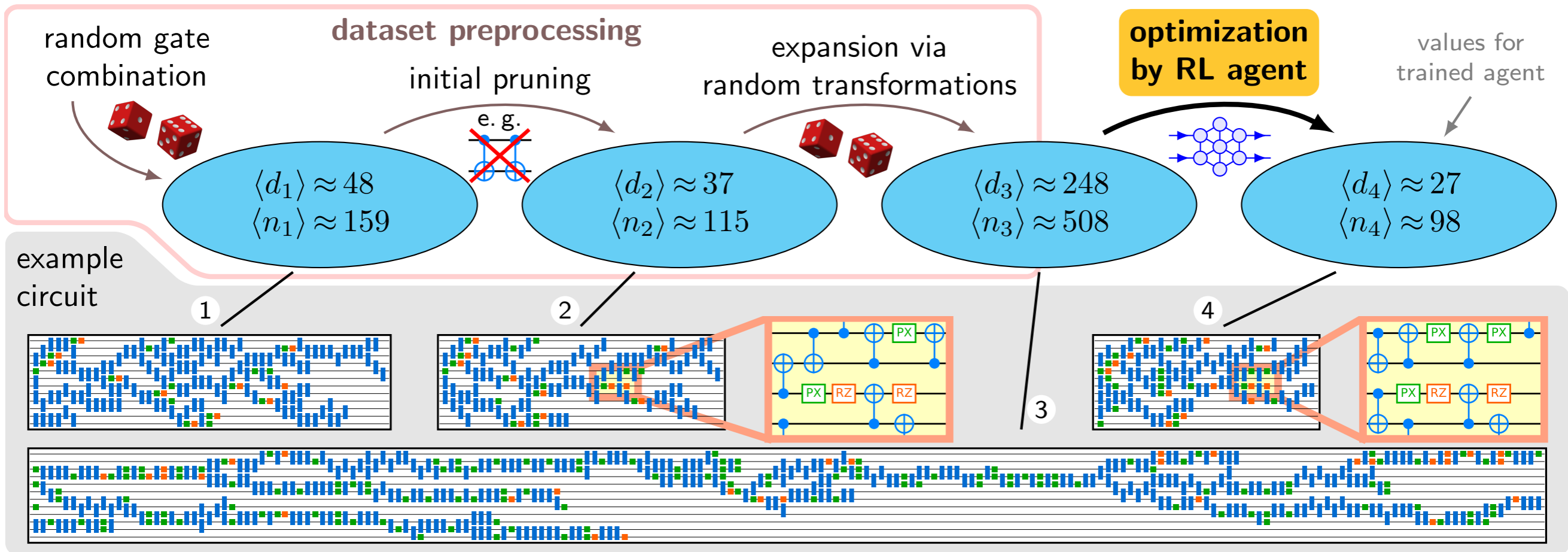
Choices: States, Agent, Actions, Rewards



Reward: reduction in gate count, depth, or combination (possibly: gate-dependent, decoherence estimate, ...)

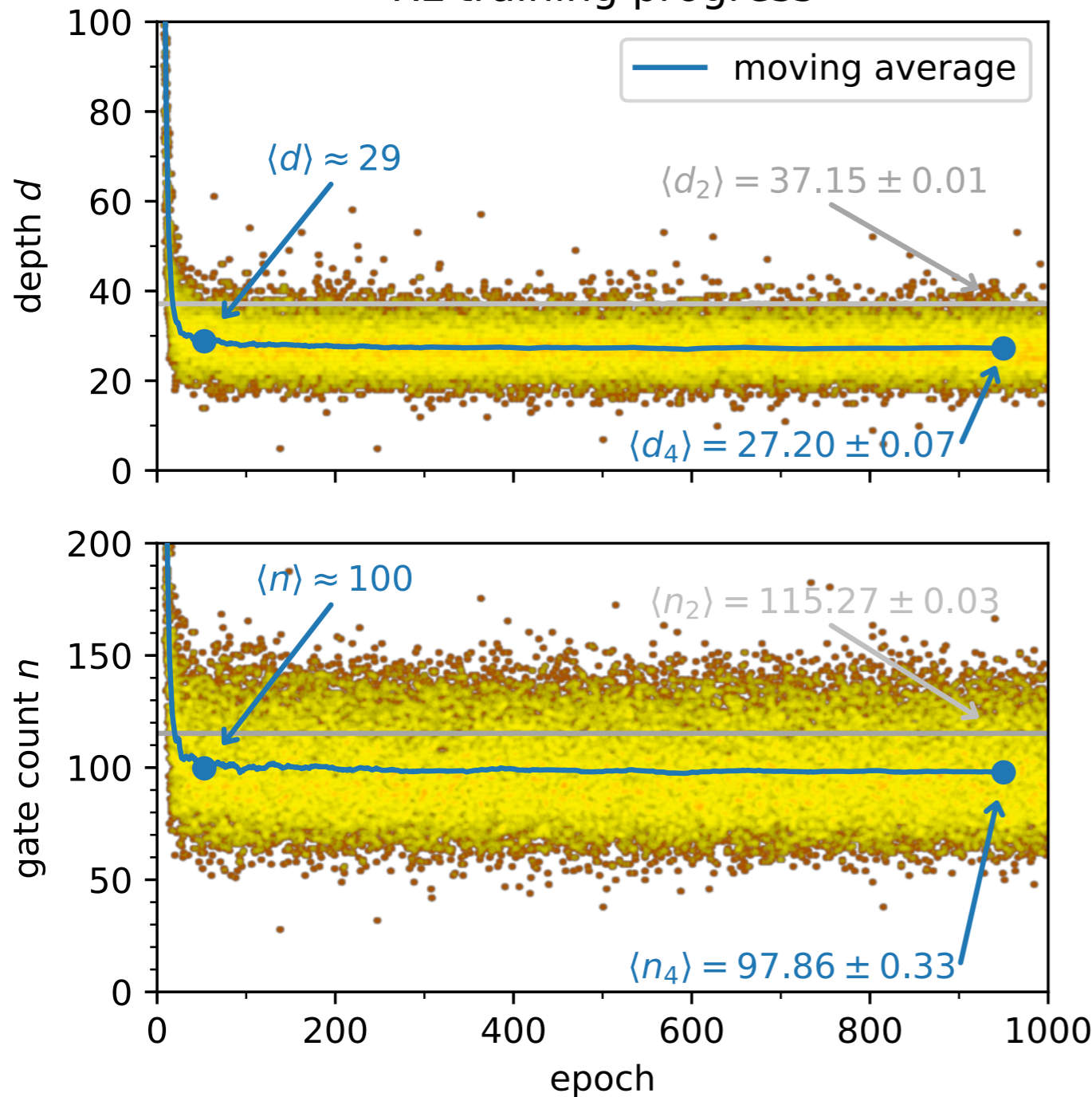
Technique: Advantage Actor Critic (namely: PPO)

Training on Random Circuits

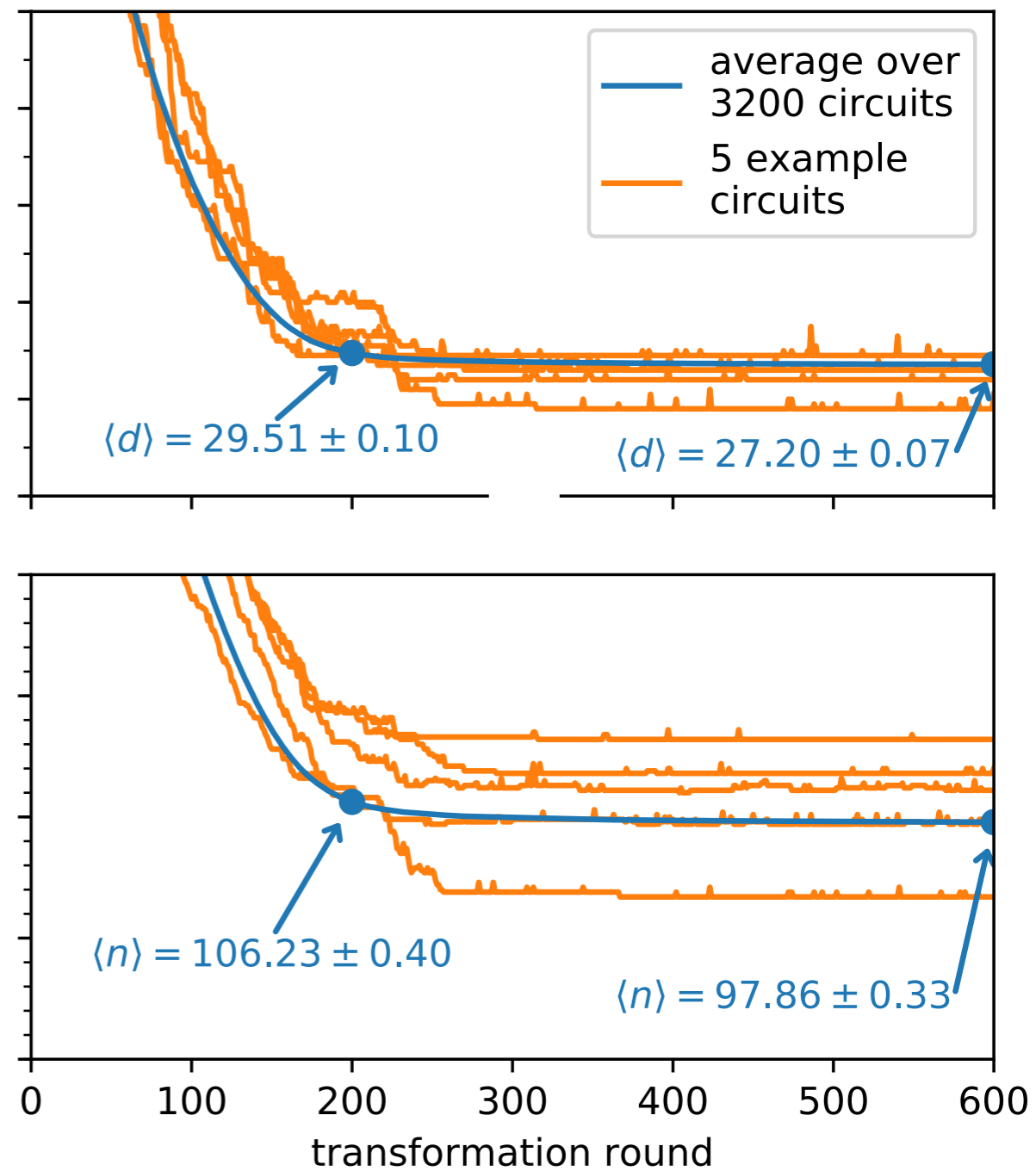


Training on Random Circuits: Progress

RL training progress

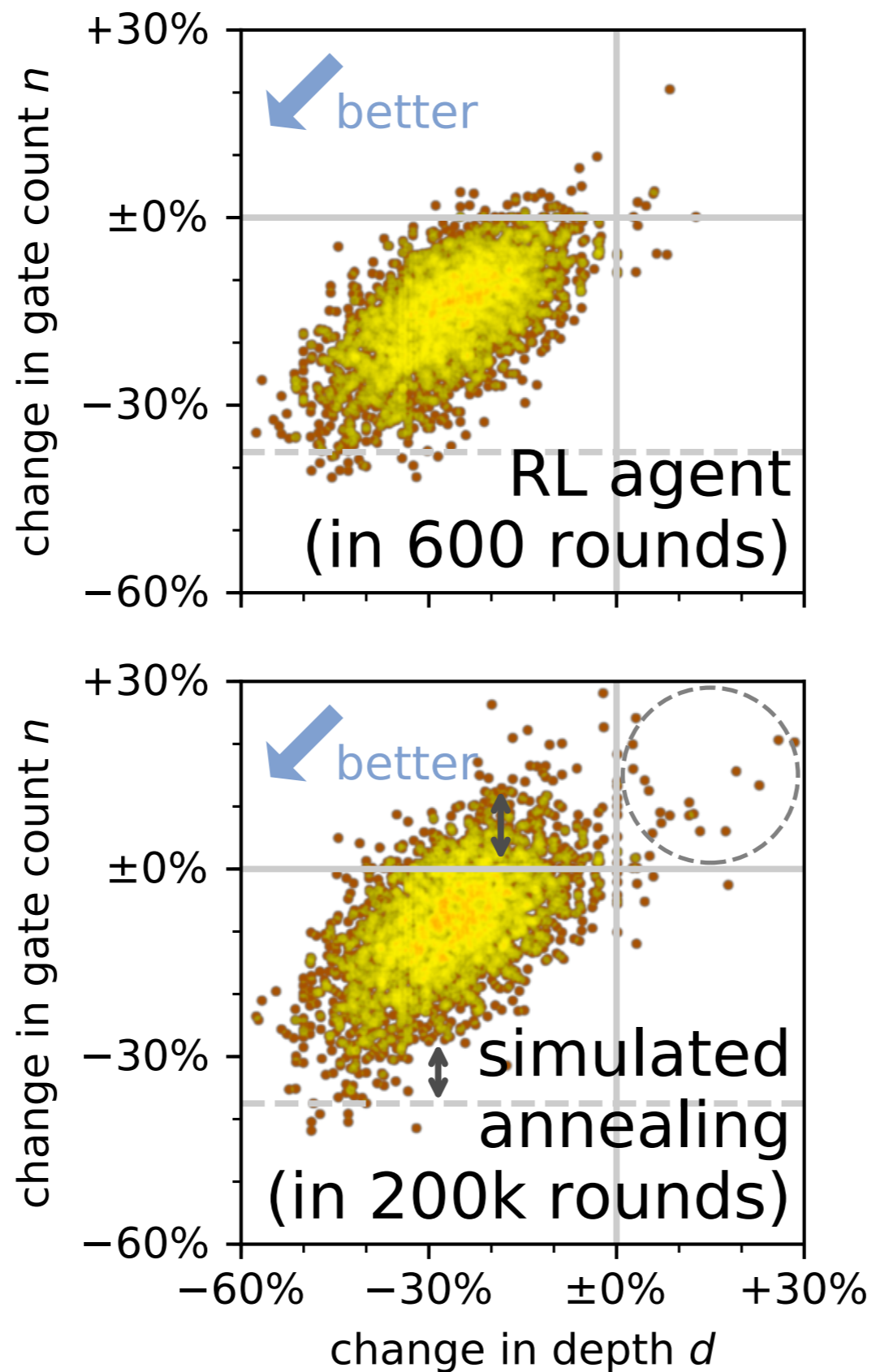


in-game progress (end of training)



1 epoch = 32 episodes

Performance

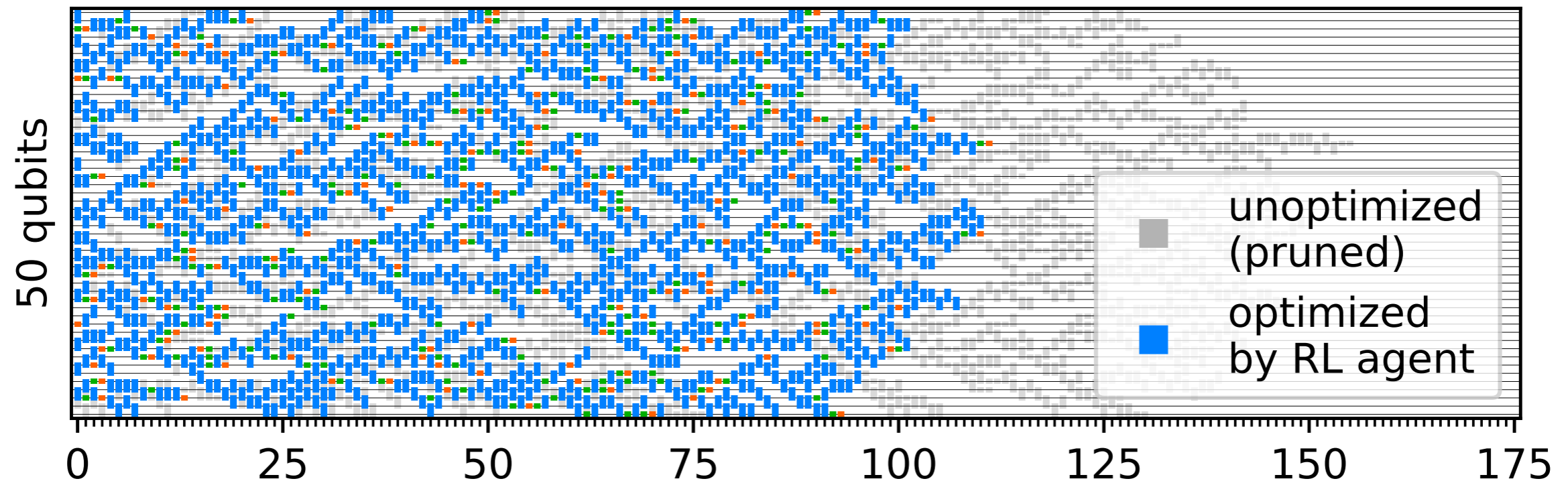


RL (after 1-week training):
2 min per circuit

Simulated annealing:
1-3 d per circuit

Large-scale Random Circuits

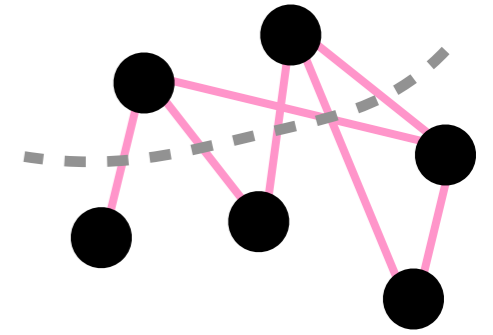
(same agent, now applied to larger circuit)



Convolutional network permits successful transfer of learned behaviour to much larger circuits: local environment of gates is relevant!

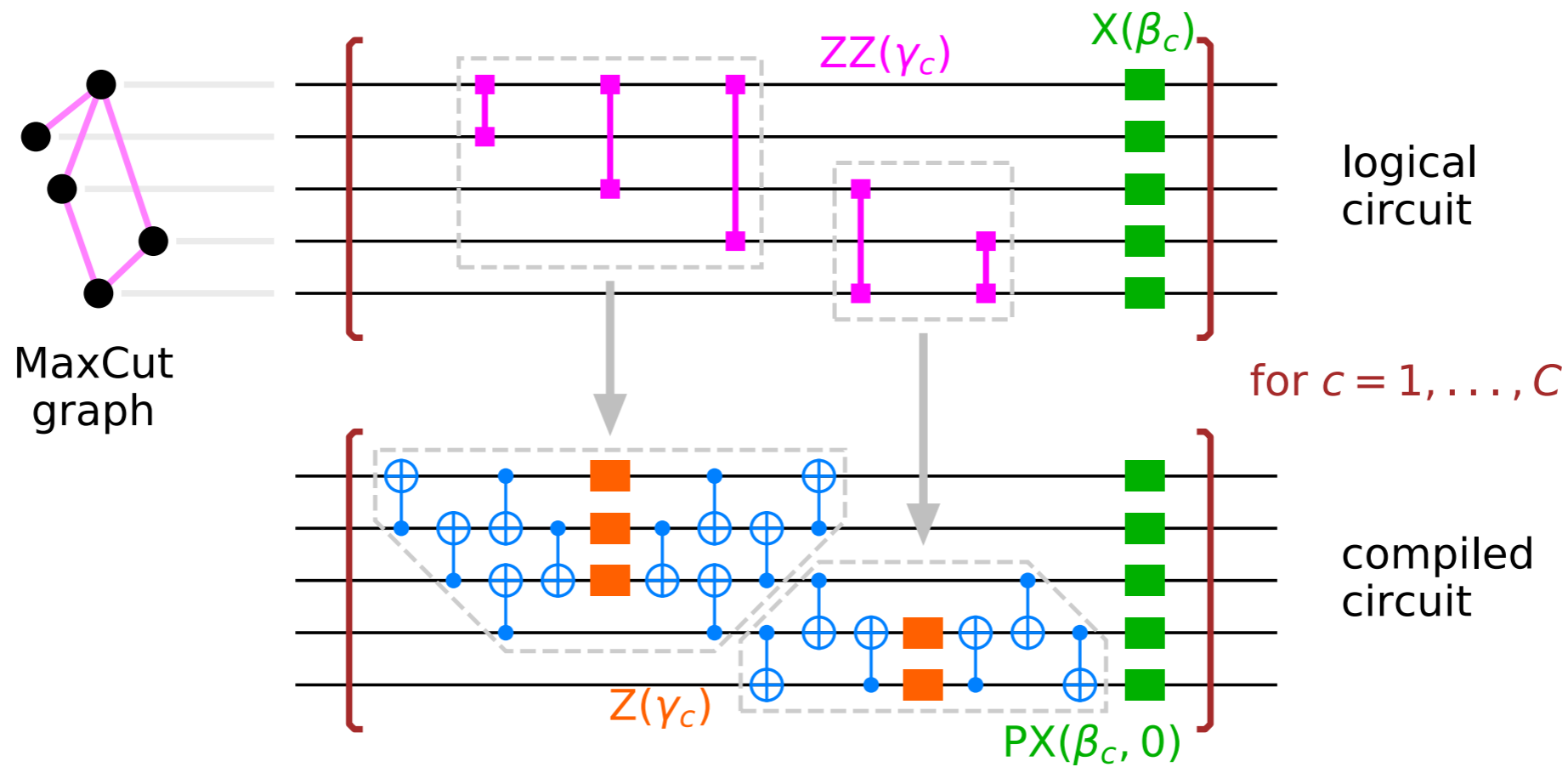
simulated annealing: \sim 1 week, comparable to full training time for general RL agent (that runs in 3-5 h)

Application to a real algorithm



Example:

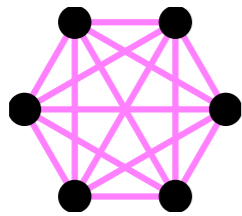
Quantum Approximate Optimization Algorithm (QAOA)
– specifically, for the MaxCut problem



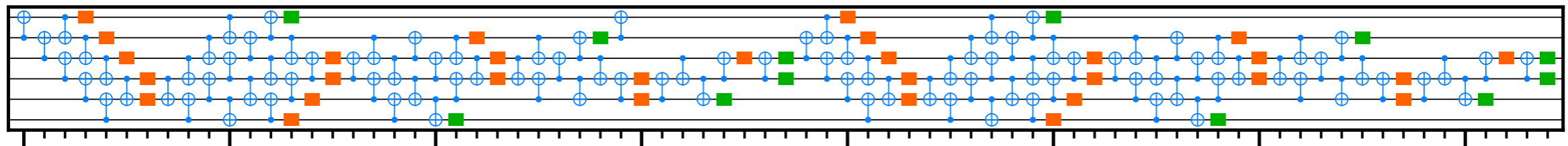
QAOA: Farhi et al, 2014

Experimental MaxCut-QAOA (Google): Harrigan et al, 2021

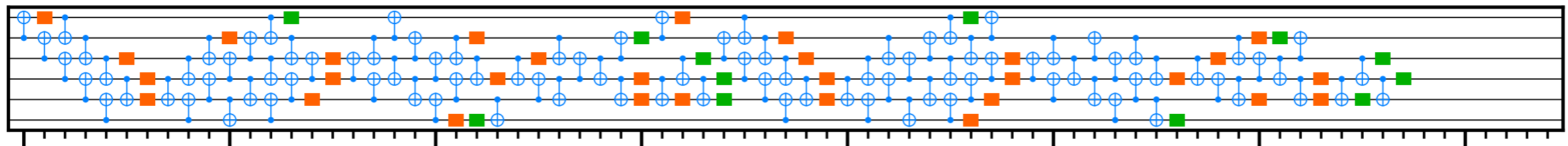
MaxCut Circuit Optimization



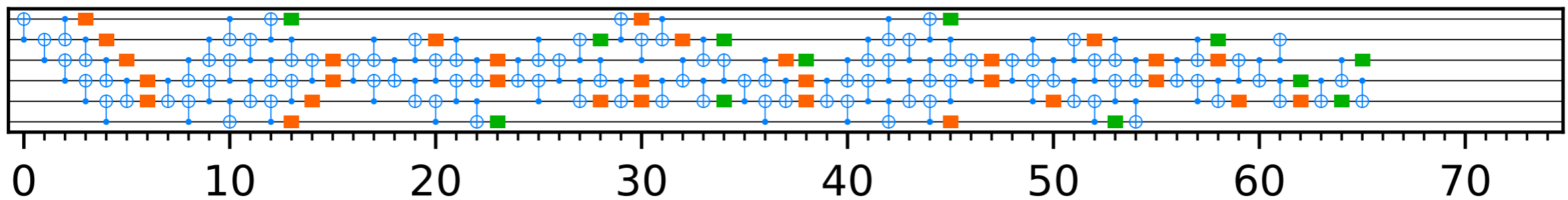
before optimization ($d = 75, n = 142$)



optimized by agent trained on random circuits ($d = 68, n = 138$)



optimized by specialized agent ($d = 66, n = 138$)



Future: Quantum Circuit Dataset

Algorithms

QAOA, Shor,
Variational Quantum
Eigensolver, ...

+ parameters
(e.g. problem instance)

Hardware

gate set,
connectivity, ...

Compiled
Quantum Circuits

Training

