https://aqa.liacs.nl/

Reinforcement Learning

DRSTP ATTP Module 2

Evert van Nieuwenburg

Plan for this lecture

Basics — Environment and Agent
State, Action, Reward/Return, Policy, Value functions, Exploration vs Exploitation
MDPs (very brief), Model-based vs Model-free, SARSA, Q-Learning

~~**Lecture 2**~~

Parameterization — Deep Q-learning, Value Methods -> Policy Methods, Stochastic policies
Policy gradient (REINFORCE), `stable-baselines`, Actor-Critic (PPO)?

Notebooks: **RL-4, RL-5..?**

~~**Lecture 3**~~

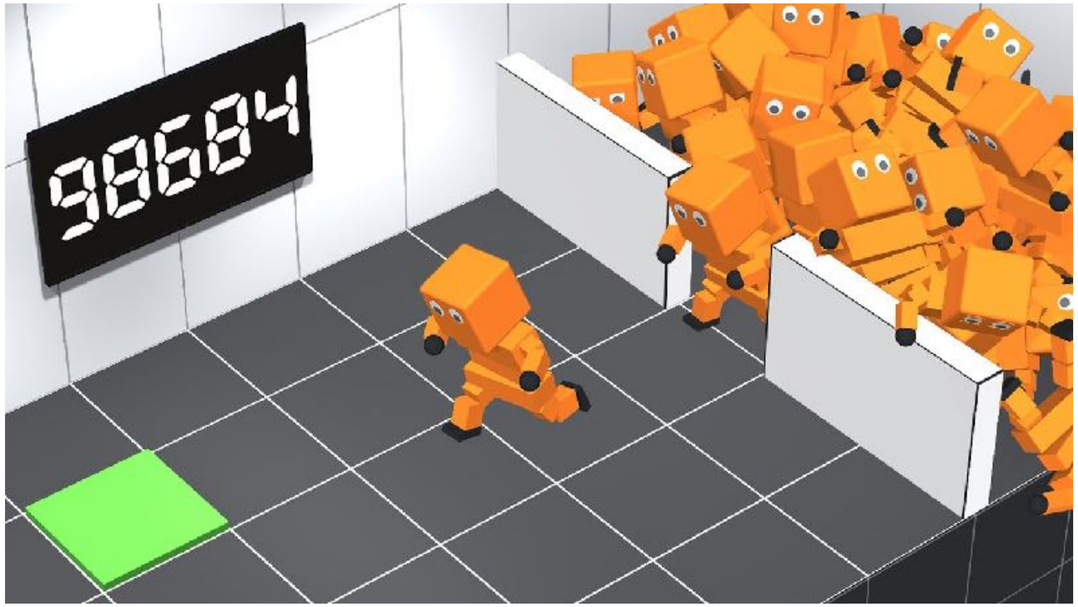
Applications to physics, Quantum RL, buffer! (<- Depending on your questions and suggestions!)

Notebooks: **RL-6?**

Why reinforcement learning?

And what is it, in a nutshell?

Because it is fun



Learn from interactions
(“generate data on the fly”)

Can be done model-free!

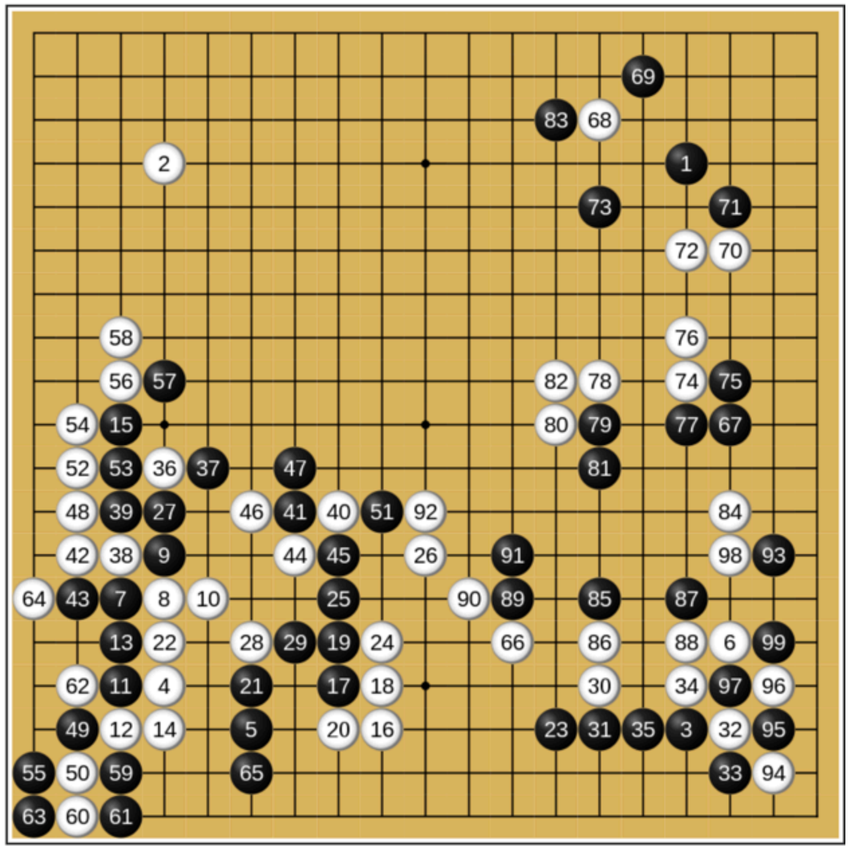
Sequential decisions

Delayed rewards

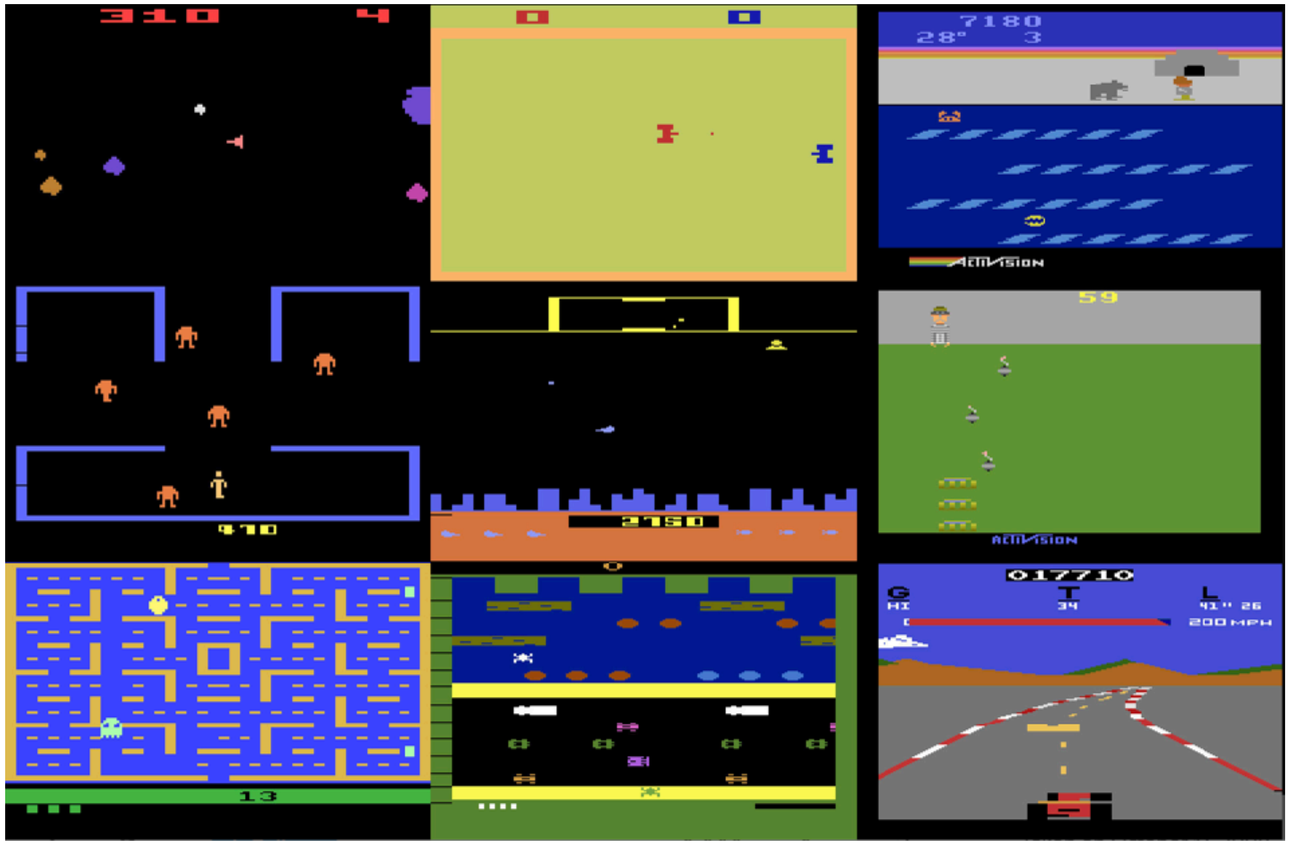
Adaptability

Computers are good at playing games

So if we can transform your research project into a game, we can use this!

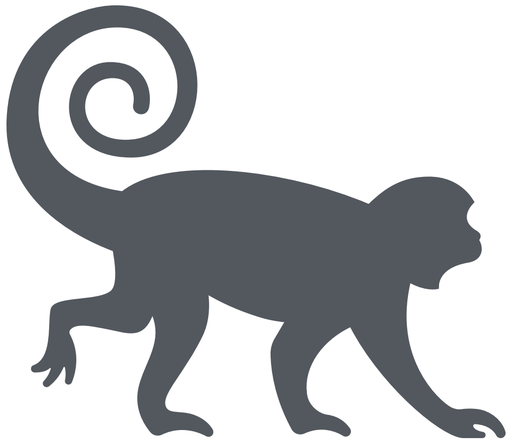


https://en.wikipedia.org/wiki/AlphaGo

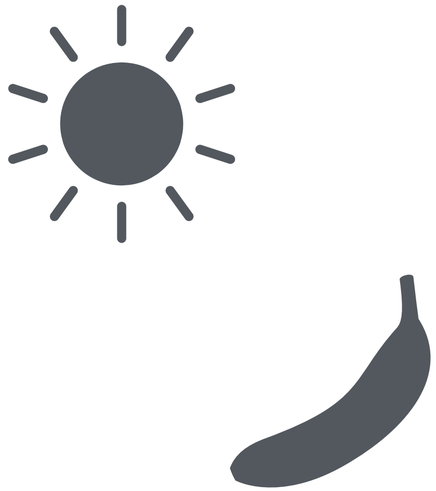


https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

Day 1



Monkey sees light

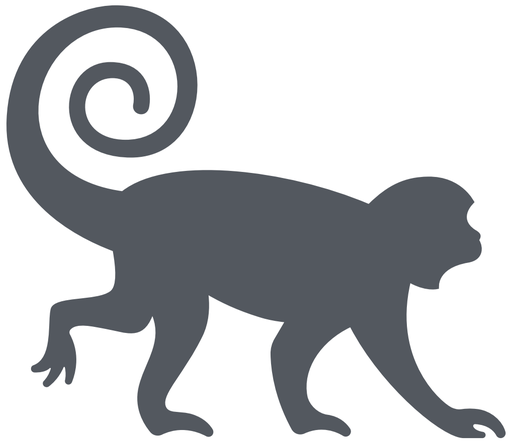


Receives banana

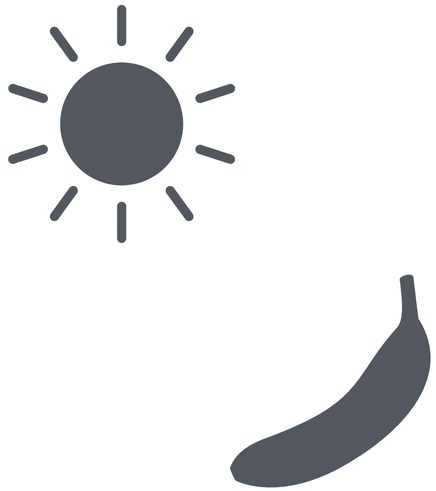
Neurotransmitter spike
(in monkey's brain)

*Disclaimer: simplified! (Also, I am not a neuroscientist 🙄)

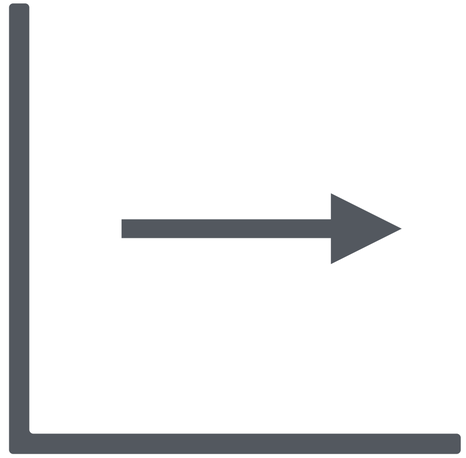
Day 20



Monkey sees light

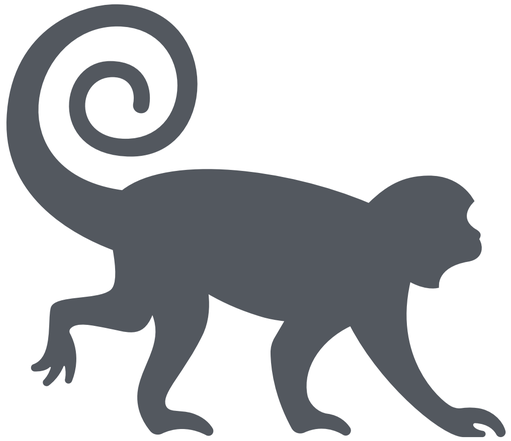


Receives banana

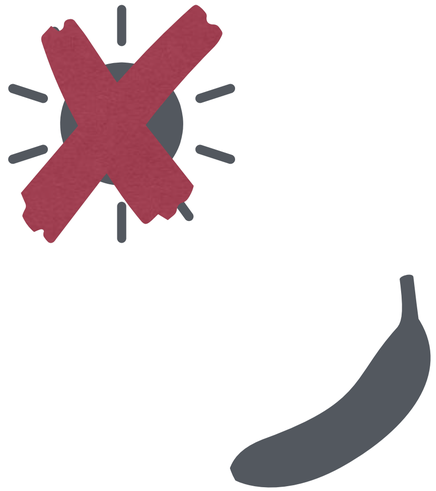


No spike, flat response
(in monkey's brain)

Day 21



Monkey sees no light



But still
receives banana

Dopamine spike is back!
(in monkey's brain)

So what is dopamine tracking?

Now let's give the monkey some agency

New 'game' rules:

- The monkey gets a button.
- If the light is on and the monkey presses the button -> Banana!
- Any other combination of light & button press -> Nothing happens.

State

Action

(State, Action)-pair		**value**
(On, Press)	→	**1**
(On, Don't Press)	→	**0**
(Off, Press)	→	**0**
(Off, Don't Press)	→	**0**

Modelling the world

Our goal is: determine what action to take for a given state of the world, so that we maximise the reward.

How? Well, let's keep an internal model of the world states: "Q(s,a)":

Q(On, Press) = 1, Q(On, Don't Press) = 0, etc

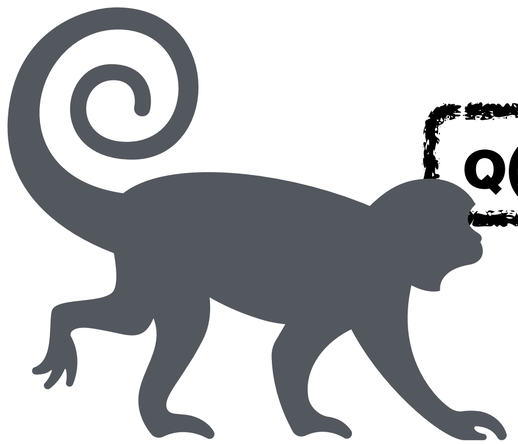
Given this 'Q-table', we can *derive* a ***policy***: $\pi(s)$ in a given state, pick the action s.t. Q is largest!

$$\pi(s) = \max_a Q(s, a)$$

But...!

We start with no knowledge on the Q-table

Q(On, Press) = 0, Q(On, Don't Press) = 0, etc



https://github.com/everthemore/Nordita2025

Learning the model

If we stick to the policy of always picking the action whose Q is highest... **Let's try!**

...we don't try new things and we never learn

So we add some exploration, just trying random things!

Exploration/Exploitation trade-off

Ok, so now sometimes we happen to press when the light is on, and we get a +1 (banana!)

This (state, action) should get a higher value. How do we update it?

Proposal: Q(state, action) = reward

Let's try!

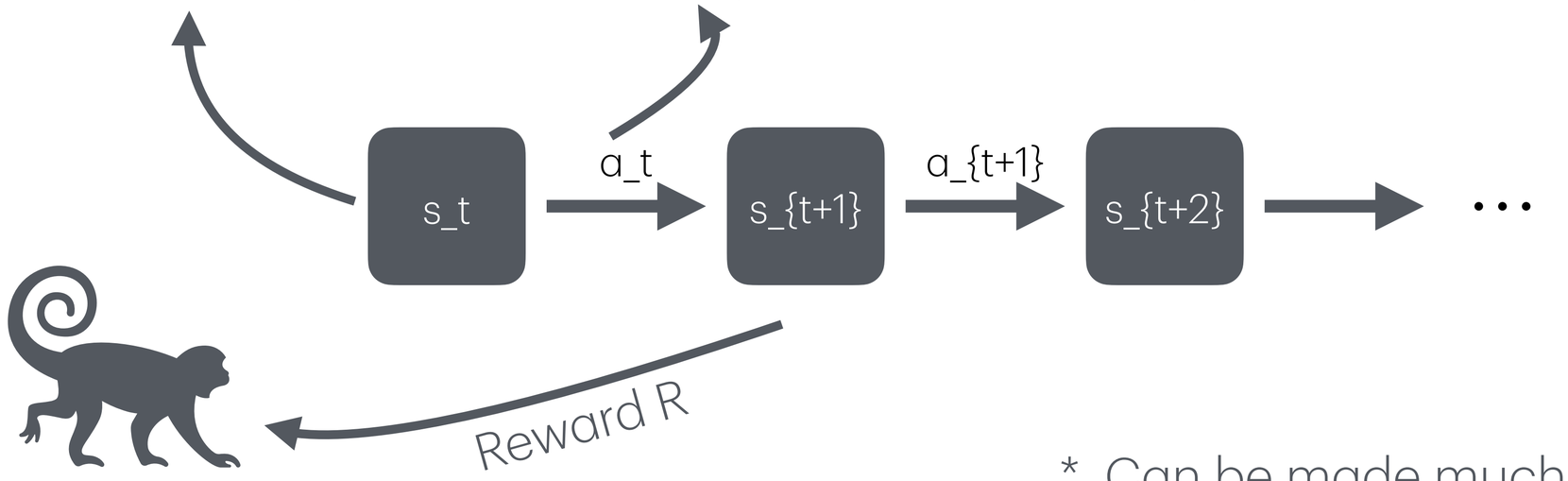
Now a few steps of improvements

This update rule (Q(s,a) = R) is quite drastic. Better (c.f. supervised learning):

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha R \qquad 0 < \alpha \leq 1$$

But this is all a one step game. What if we need to plan ahead?

A set of **states** S, a set of **actions** A, a set of **rewards** R, a **value** function Q, a **policy**



* Can be made much more formal through Markov Decision Processes (MDPs)

Now a few steps of improvements

This update rule (Q(s,a) = R) is quite drastic. Better (c.f. supervised learning):

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha R \qquad 0 < \alpha \leq 1$$

But this is all a one step game. What if we need to plan ahead?

A set o **policy**

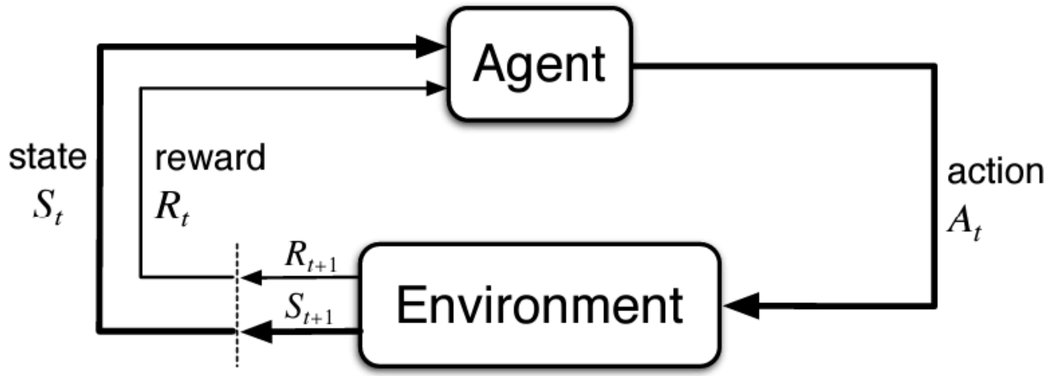


Figure 3.1: The agent–environment interaction in a Markov decision process.

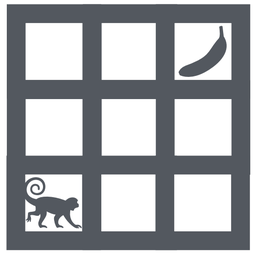
From **Sutton & Barto** (Google to find it, free (and standard) book on RL!)

Can be made much more formal through Markov Decision Processes (MDPs)

GridWorld, one of the standards

Actions are: move up/down/left/right

State is (x,y) of monkey and of banana

$$Q(\text{  }, a) = 0$$

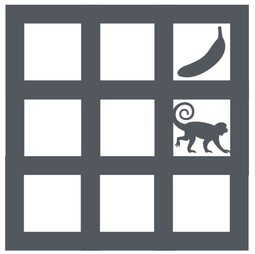
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha R$$

Even if it gets to the two squares next to the banana, this update rule does not give it any guidance if it is farther away.

State-action-reward-state-action (SARSA)

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha R_t$$

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\Big(R_t + Q(s_{t+1}, a_{t+1})\Big)$$

Q( , up) = 1 → Q( , right) = α

This is an ‘on-policy’ update rule.

Just *one* more modification!

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \max_a Q(s_{t+1}, a))$$

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a))$$

discount factor

If $\gamma = 0$, the agent is 'shortsighted': goes for immediate reward

If $\gamma \geq 1$, we may get exploding Q-values (i.e. they diverge; can you make an example?)

So typically: $0 \leq \gamma \leq 1$

And so we arrive at '*Q-learning*'

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a))$$

- It is an 'off-policy' algorithm. It learns the (optimal) policy pi* independent of the policy being followed during learning
- It can converge to the optimal policy pi* (and optimal Q*) under certain conditions (finite state-action space, ergodic (each state-action pair visited infinitely often), ...)
- It has a bias: if the Q-values for s_{t+1} are noisy in the beginning, it overestimates the actions.
- It is model-free, doesn't need to know which state transitions to which other state upfront.

And so we arrive at '*Q-learning*'

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a))$$

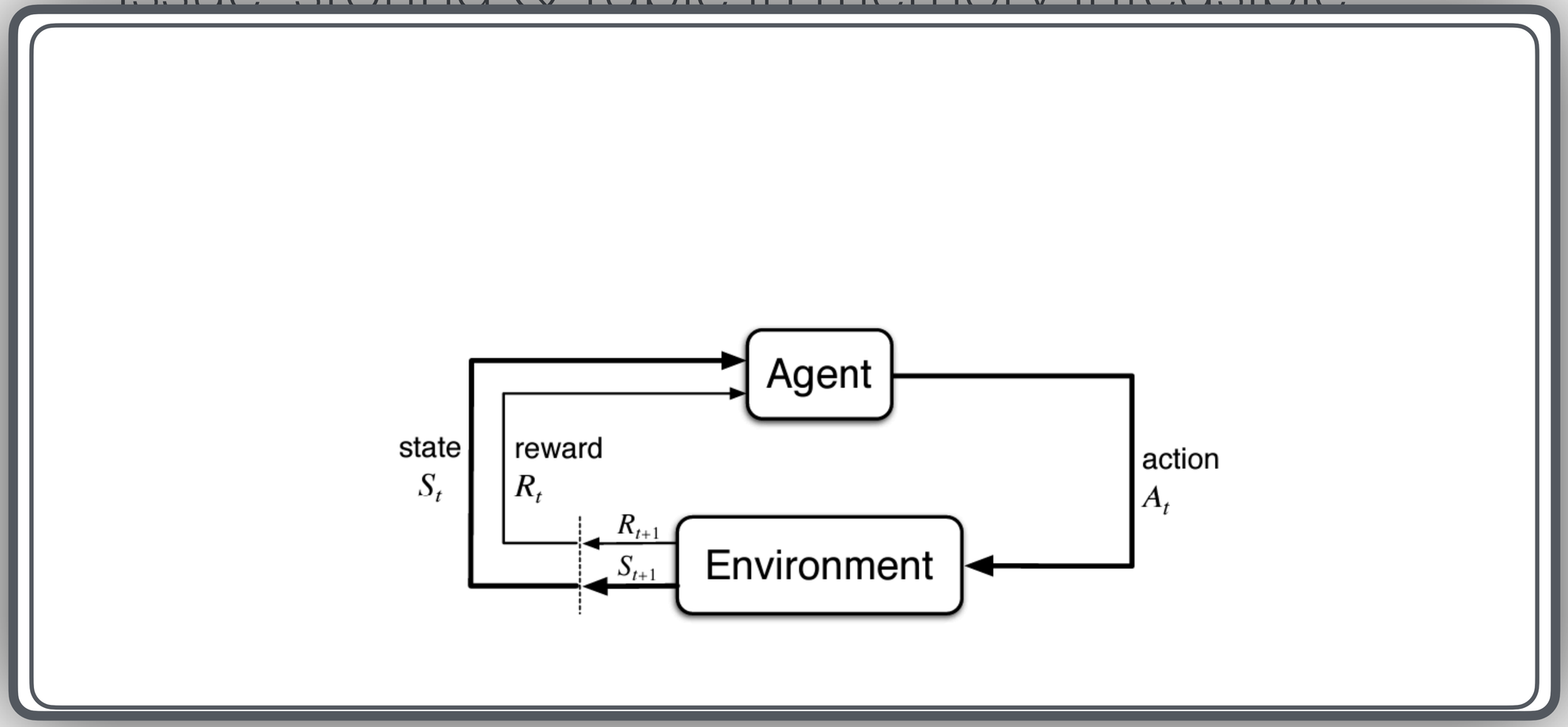
- It is an 'off-policy' algorithm. It learns the (optimal) policy pi* independent of the policy being followed during learning
- It can converge to the optimal policy pi* (and optimal Q*) under certain conditions (**finite state-action space**, ergodic (each state-action pair visited infinitely often), ...)
- It has a bias: if the Q-values for s_{t+1} are noisy in the beginning, it overestimates the actions.
- It is model-free, doesn't need to know which state transitions to which other state upfront.

What do we do for continuous and/or very large spaces?

Issue: storing Q-table in memory infeasible.
Solution: turn Q-table into (learnable) function

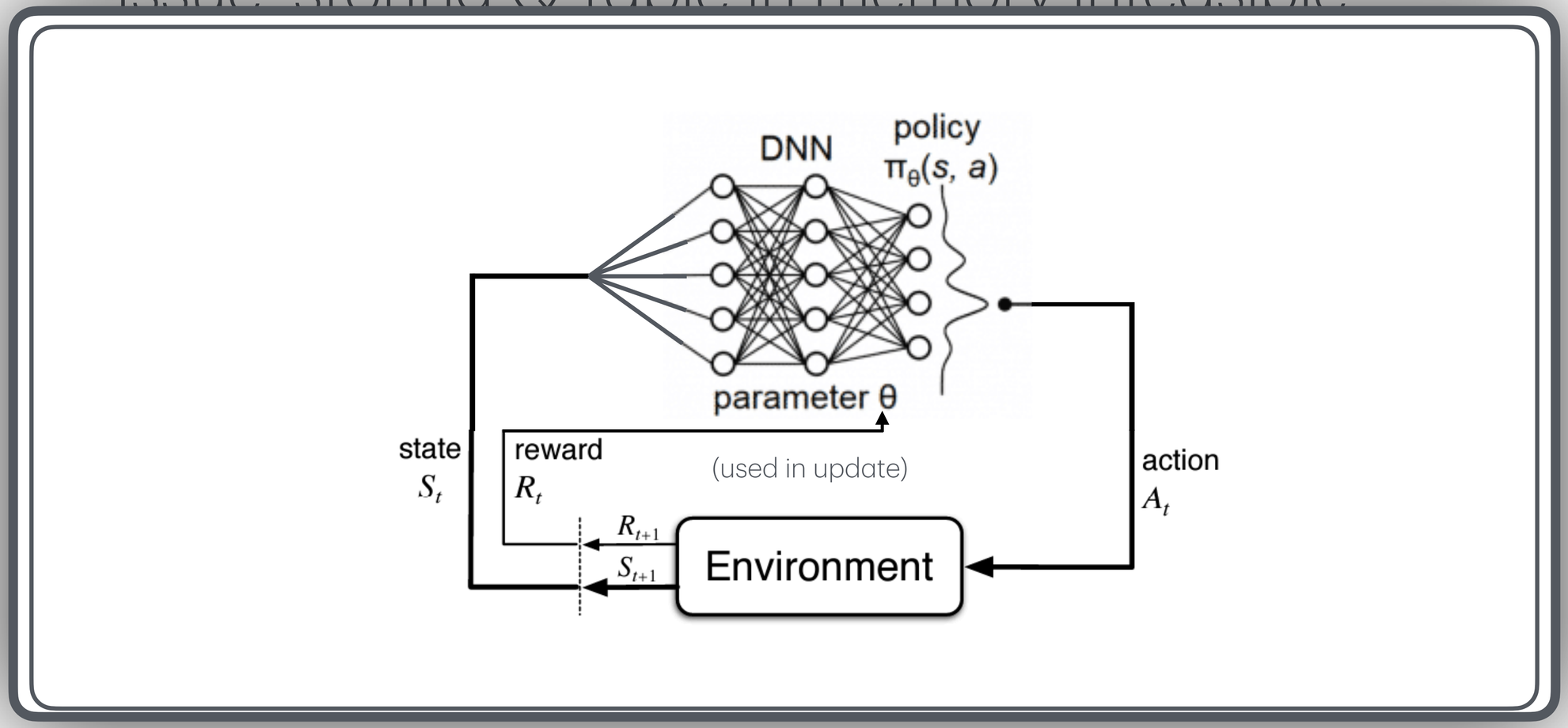
What do we do for continuous and/or very large spaces?

Issue: storing Q-table in memory infeasible



What do we do for continuous and/or very large spaces?

Issue: storing Q-table in memory infeasible



https://github.com/everthemore/Nordita2025

Some thoughts about DQNs

- Because of the NN, perhaps it does well at generalising! That is, if it is trained on e.g. chess, it may also predict a good action for a chess config it never saw before

- Suffers from overestimation bias, the max(Q) bit picks out early actions with high variance, and they keep being reinforced. In addition, targets are non-stationary. (**Solution**: Double Q-learning)

- Consecutive runs are correlated, doesn't easily sample from infrequent actions. (**Solution**: Replay Buffer, sampled randomly).

https://aqa.liacs.nl/

Reinforcement Learning

DRSTP ATTP Module 2

Evert van Nieuwenburg

(Current) plan for the lectures

So you know what to expect. Suggestions welcome!

Lecture 1

Basics ~~Environment and Agent~~

~~State, Action, Reward/Return, Policy, Value functions, Exploration vs Exploitation~~

~~MDPs (very brief), Model-based vs Model-free, SARSA, Q-Learning~~

Notebooks: **RL-1, RL-2, RL-3**

Lecture 2&3

Parameterization Deep Q-learning, Value Methods -> Policy Methods, Stochastic policies

Policy gradient (REINFORCE), Actor-Critic, PPO, `stable-baselines`

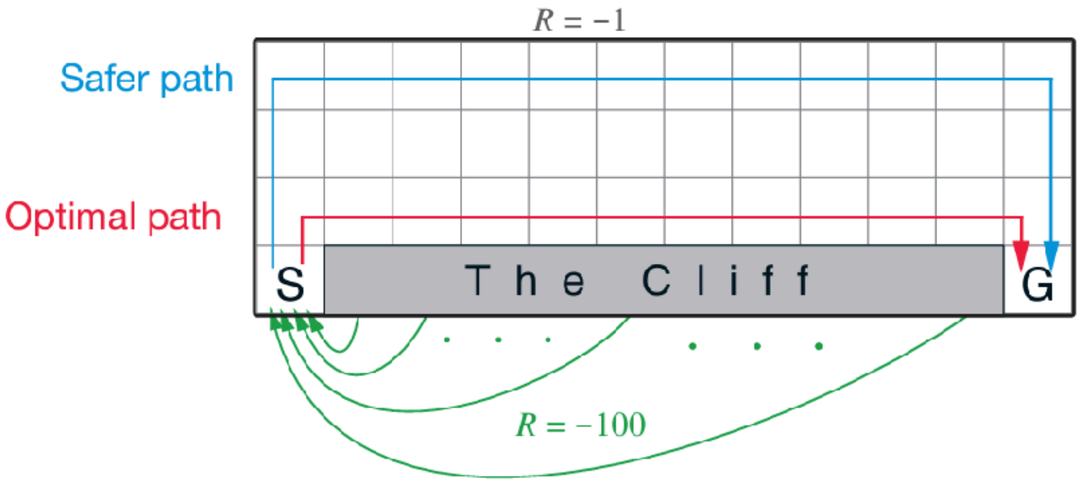
Applications to quantum physics, Quantum RL?, buffer!

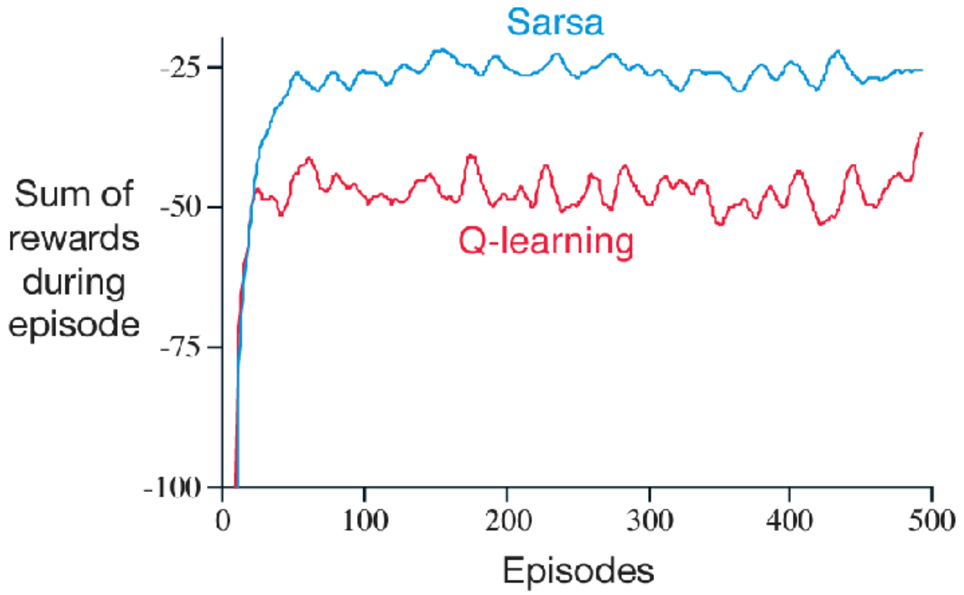
Notebooks: **RL-4, (RL-5), RL-6**

Some things I forgot to say:

SARSA: $$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\Big(R_t + Q(s_{t+1}, a_{t+1})\Big)$$

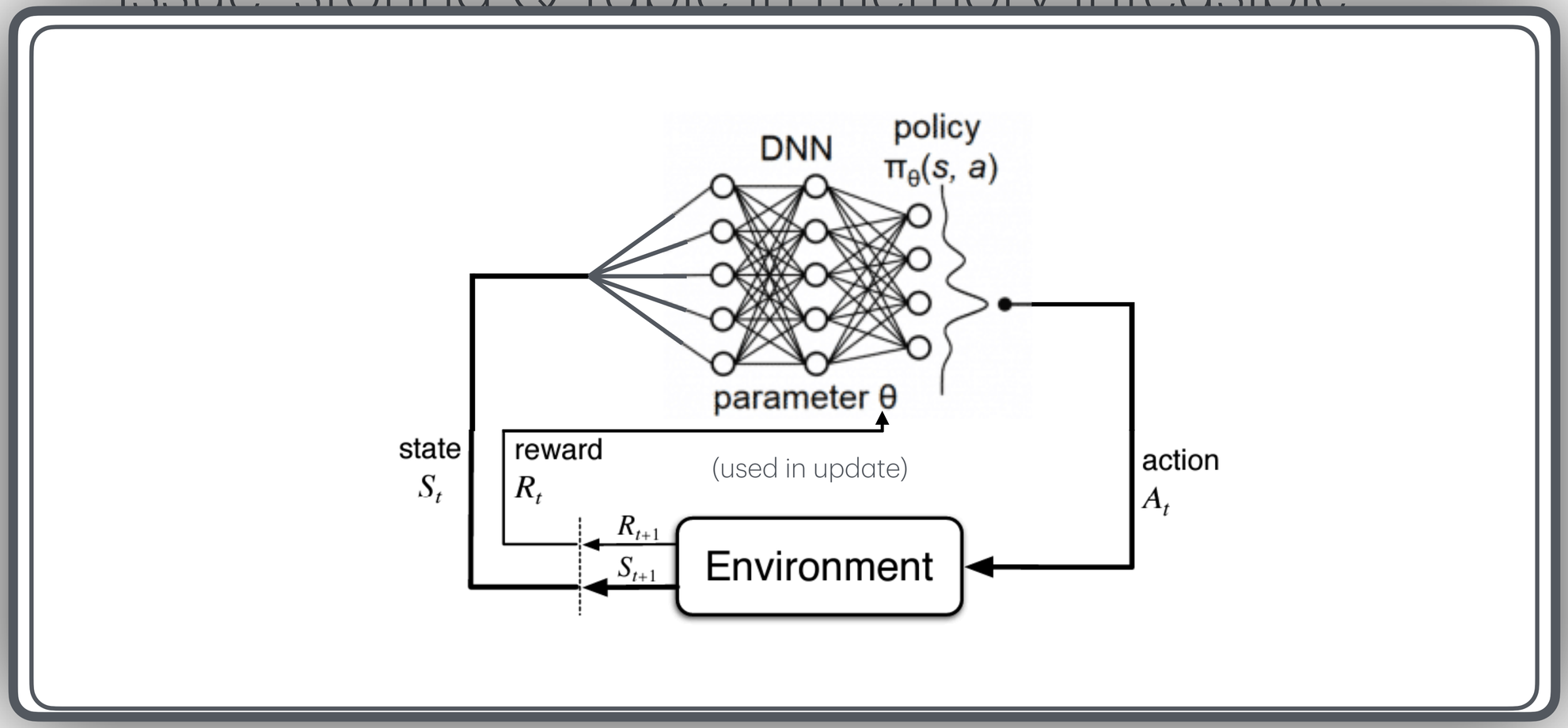
Q-learning: $$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a))$$





What do we do for continuous and/or very large spaces?

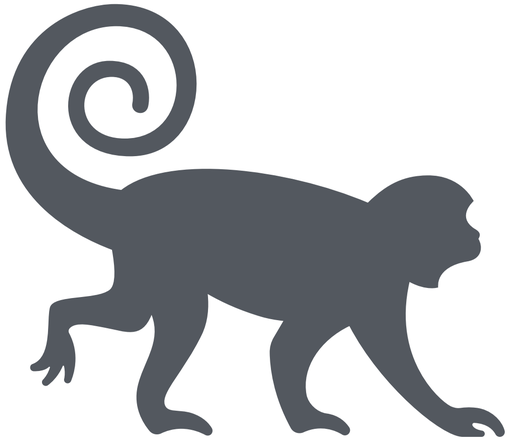
Issue: storing Q-table in memory infeasible

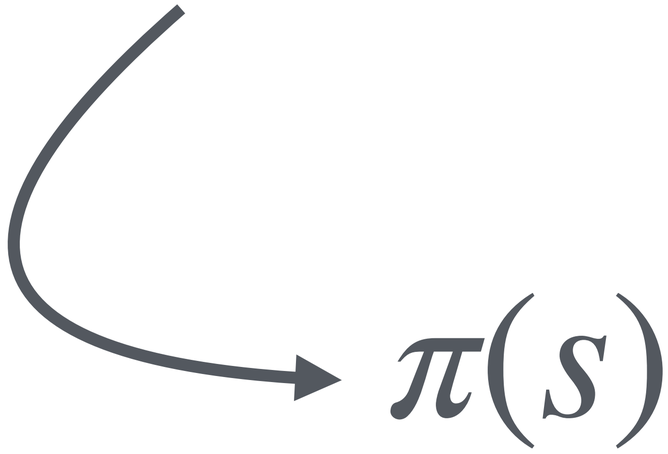


https://github.com/everthemore/Nordita2025

What we have discussed so far are “value-methods”

$Q(s, a)$ $V(s)$





Policies ***are derived from*** the value functions

Now we'll move to policy methods

Policies ***are updated directly***

Deterministic policies are prone to getting stuck in local optima

We have already seen a glimpse of *stochastic policies*: $\pi(a \mid s)$
(~we had a NN predict the Q(a | s), then we picked argmax)

Now, we will parametrise a stochastic policy directly: $\pi_\theta(a \mid s)$

This is good for environment with stochastic dynamics, or partial observability
(quantum!)

Now we'll move to policy methods

Policies ***are updated directly***

Now, we will parametrise a stochastic policy directly: $\pi_\theta(a \mid s)$

This is good for environment with stochastic dynamics, or partial observability **(quantum!)**

```python
import torch
import torch.nn as nn

class PolicyNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.fc = nn.Sequential(
            nn.Linear(state_dim, 128),
            nn.ReLU(),
            nn.Linear(128, action_dim),
            nn.Softmax(dim=-1)
        )
    def forward(self, state):
        return self.fc(state)
```

They sample actions probabilistically, so have inherent exploration

They can adapt dynamically to uncertainty in the environment

The policy gradient theorem

Remember the goal: maximise the total *expected* reward (return)

$$G_t = \sum_{t'=0}^{\infty} \gamma^{t'} R_{t+t'+1}$$ Reward

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t]$$ Expected reward

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\ G_t\right]$$

The Policy Gradient Theorem

The policy gradient theorem derivation

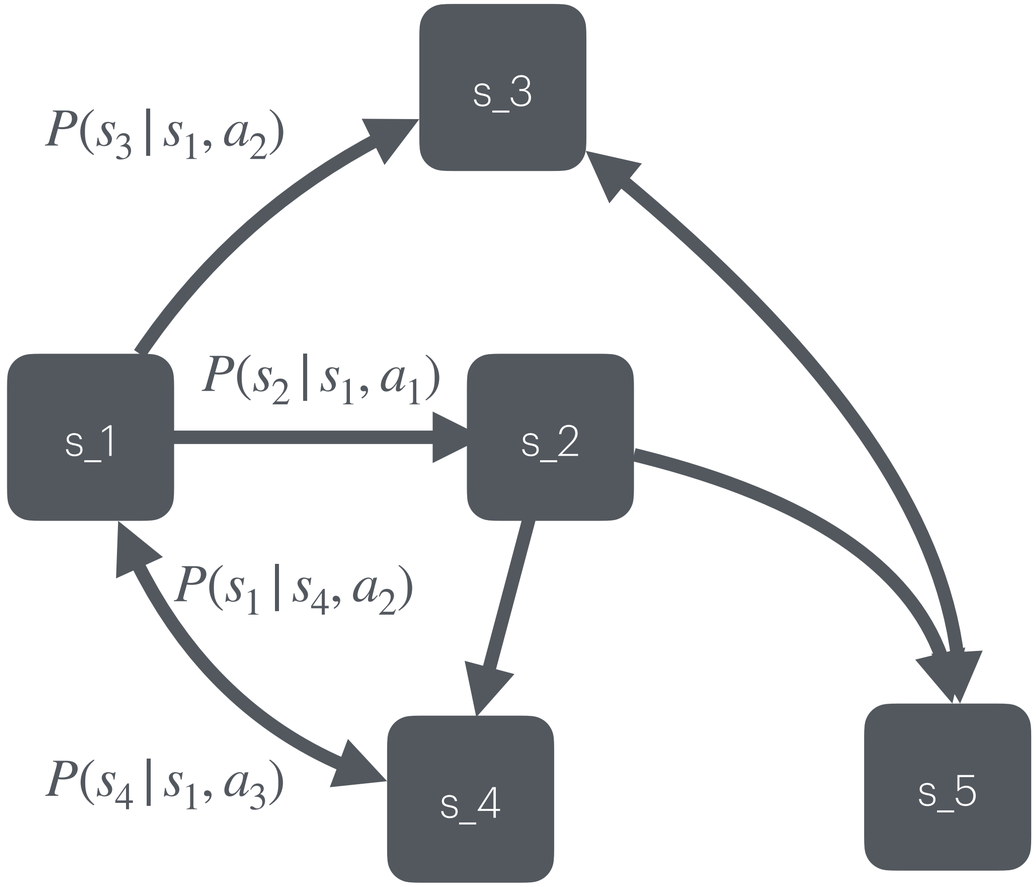
So first, some basics on Markov Decision Processes

States {S},
Actions {A(s)},
Rewards {R(s, a, s')},
Transitions P(s'|s,a)

$$\pi^*(a \mid s)$$

Goal: find optimal policy to maximise total discounted reward

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



Computing the estimated return

Requires computing the probabilities of trajectories

$$P(\tau) = ?$$

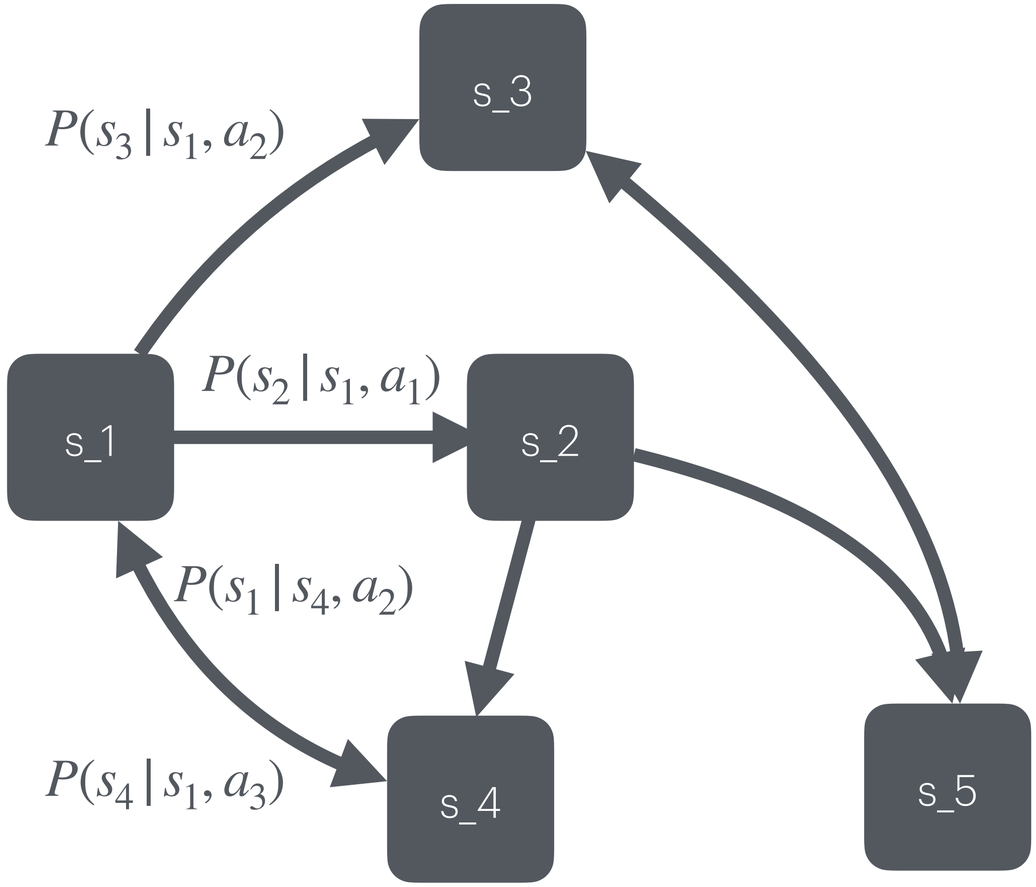
$$\tau = (s_0, a_0, s_1, a_1, \ldots)$$

$$P(s_0 a_0) = \pi(a_0 \mid s_0) P(s_0)$$

$$P(s_0 a_0 s_1) = P(s_1 \mid s_0, a_0) \pi(a_0 \mid s_0) P(s_0)$$

$$P(s_0 a_0 s_1 a_1 s_2) = P(s_2 \mid s_1, a_1) \pi(a_1 \mid s_1) P(s_1 \mid s_0, a_0) \pi(a_0 \mid s_0) P(s_0)$$

$$P_\theta(\tau) = P(s_0) \prod_{t=0}^{\infty} P(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$



Computing the estimated return

Requires computing the probabilities of trajectories

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t]$$

$$= \sum_{\tau} P_\theta(\tau) G_t(\tau)$$

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta P_\theta(\tau)\ G_t(\tau)$$

$$\nabla_\theta \log P_\theta = 1/P_\theta \nabla_\theta P_\theta \qquad \nabla_\theta P_\theta = P_\theta \nabla_\theta \log P_\theta$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log(P_\theta(\tau))\ G_t\right]$$

Computing the estimated return

Requires computing the probabilities of trajectories

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log(P_\theta(\tau))\, G_t\right]$$

$$P_\theta(\tau) = P(s_0)\prod_{t=0}^{\infty} P(s_{t+1}\,|\,s_t, a_t)\pi_\theta(a_t\,|\,s_t)$$

$$\nabla_\theta \log P_\theta(\tau) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t\,|\,s_t)$$

$$\nabla_\theta J(\theta) = \sum_{t=0}^{\infty} \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t\,|\,s_t)\, G_t\right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t\,|\,s_t)\, G_t\right]$$

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

Gradient *from t onwards*

We can now start to REINFORCE

The vanilla policy gradient method (c.f. Q-learning for value method)

1. Run an episode with the current policy (sample a trajectory)
2. Compute the returns (G_t, i.e. G_0, G_1, G_2, ...)
3. Update the policy parameters according to:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$$

The gradients are often noisy

So we can introduce a baseline value:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s)\ \left(G_t - V(s)\right)\right]$$

The baselines does not depend on the chosen action, so it doesn't affect the expected gradient

It reduces the variance intuitively, because actions that are better than expected (G > V) are reinforced, others discouraged. We assume V is a stable estimate.

$G_t - V(s_t)$ is often called the "Advantage" (it is an approximation)

What we have just created is...

... the vanilla Actor-Critic model

The **actor** selects actions according to $\pi_\theta(a \mid s)$

The **critic** estimates the value of states $V(s)$

The **actor** updates its policy based on the advantage

$$L_{\text{actor}} = -\log \pi_\theta(a \mid s) \left(G_t - V(s)\right)$$

The **critic** updates its value estimate using TD-learning

$$L_{\text{critic}} = \delta_t^2 \qquad \delta_t = r_t + \gamma V(s_{t+1}) - V(s)$$

Expected value

Current value

From A2C (Advantage Actor-Critic) to PPO

Skipping the “Trust Region Policy Optimization” (TRPO) method

The main issue with PO, is that sometimes the gradient updates are too drastic. We would like to constrain the size of policy jumps.

To do so, PPO introduces **clipping**, which ensures that the **ratio** of the new policy to the old policy isn’t too large.

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\text{old}}(a \mid s)}$$

$$L_{\text{CLIP}}(\theta) = A \cdot \min\left(r(\theta), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\right)$$

(If r close to 1, use it, otherwise clip it at 1 \pm \epsilon)

Here is the final logical flow:

REINFORCE updates the policy by directly using the return G_t, but it has a high variance

Actor-Critic uses a critic to estimate the value function, and updates the policy using G_t - V(s) (which reduces variance)

PPO prevents jumps in the policy during training by using clipping, which clips updates so the policy ratio changes by ~1 \pm \epsilon.

And luckily, we have stable-baselines

https://stable-baselines3.readthedocs.io/en/master/

Pair this with a Gymnasium environment!

https://aqa.liacs.nl/

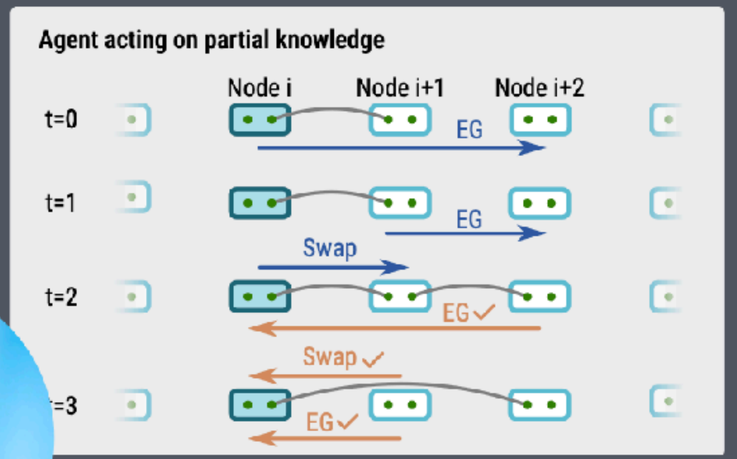
Reinforcement Learning

DRSTP ATTP Module 2

Evert van Nieuwenburg

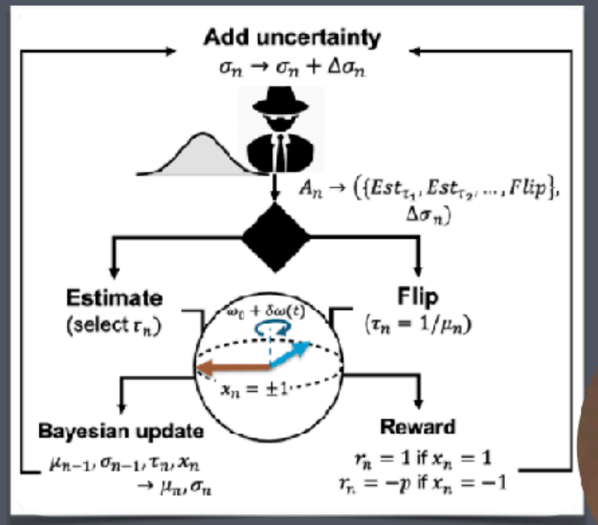
The strategy learned by an RL agent...
...is also generally interpretable

Entanglement Distribution in a Quantum Network



Jan Li

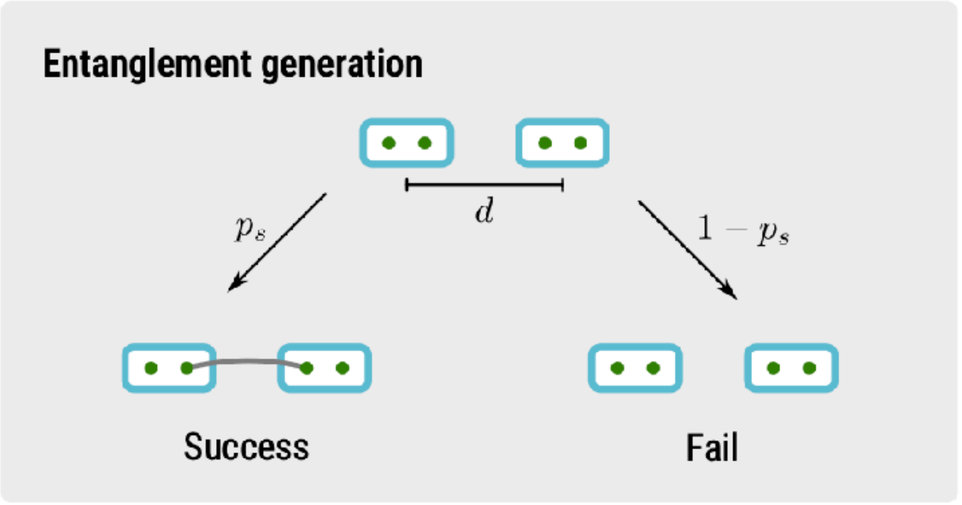
Qubit Control in a Noisy Environment

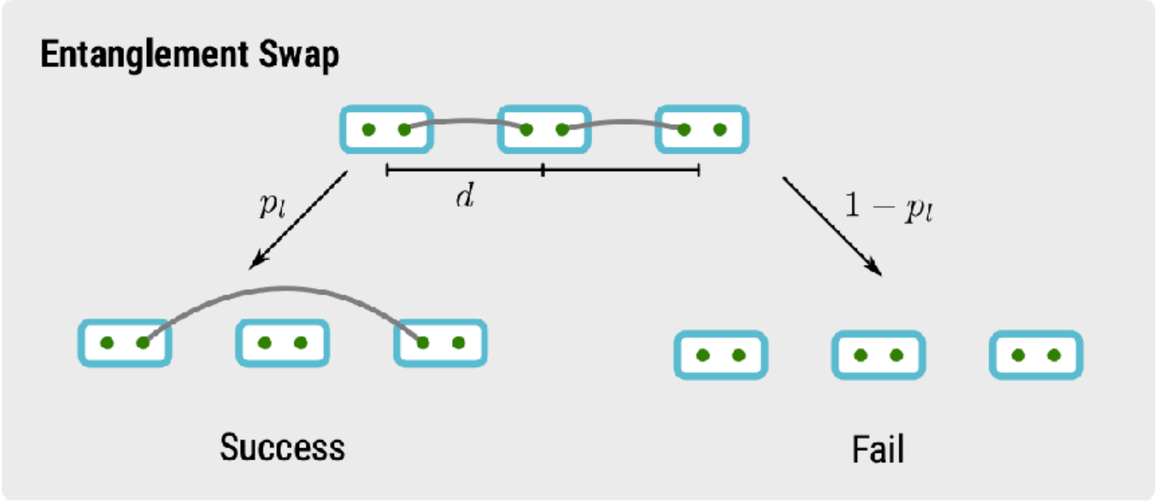


Jan Krzywda (PD)

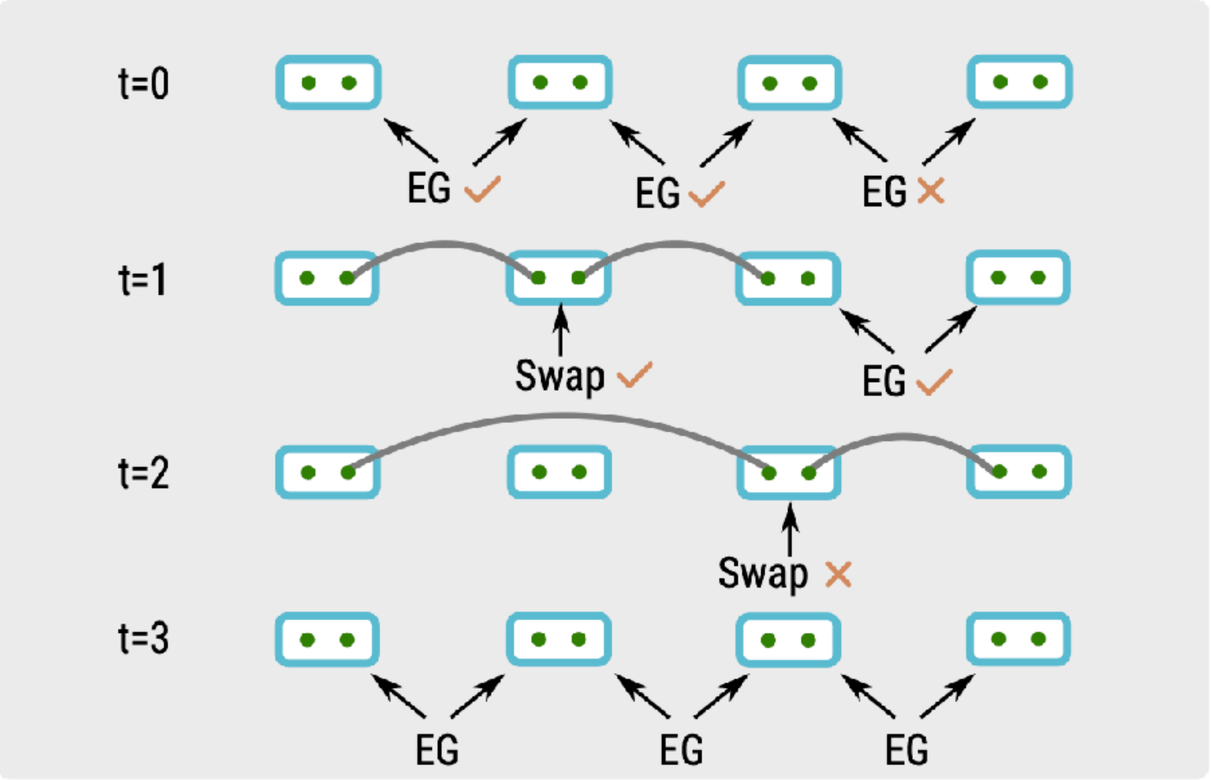
Quantum networks consist of nodes with qubits

and entanglement distribution is one of the core concepts



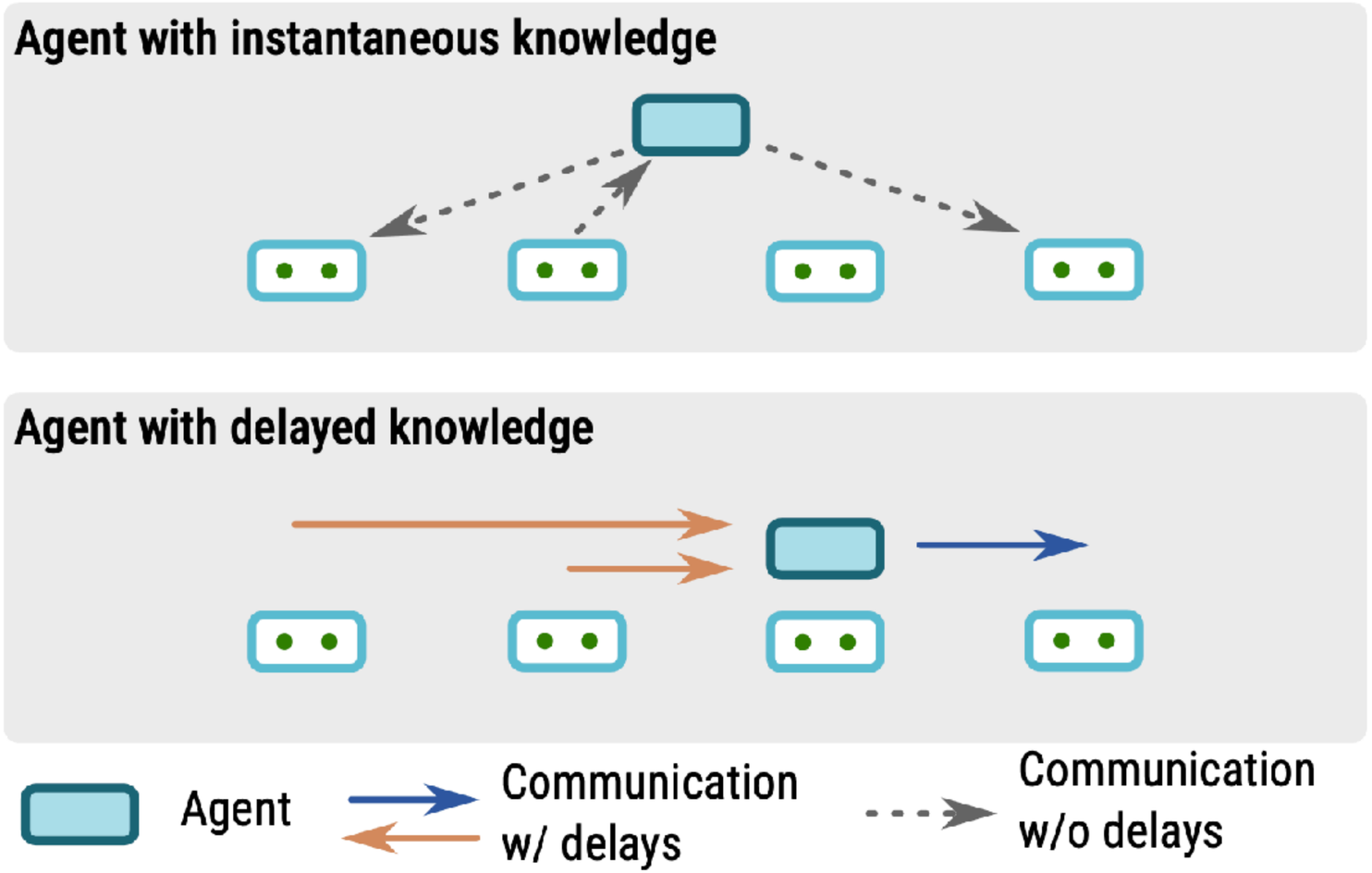


The SWAP-asap protocol is the benchmark

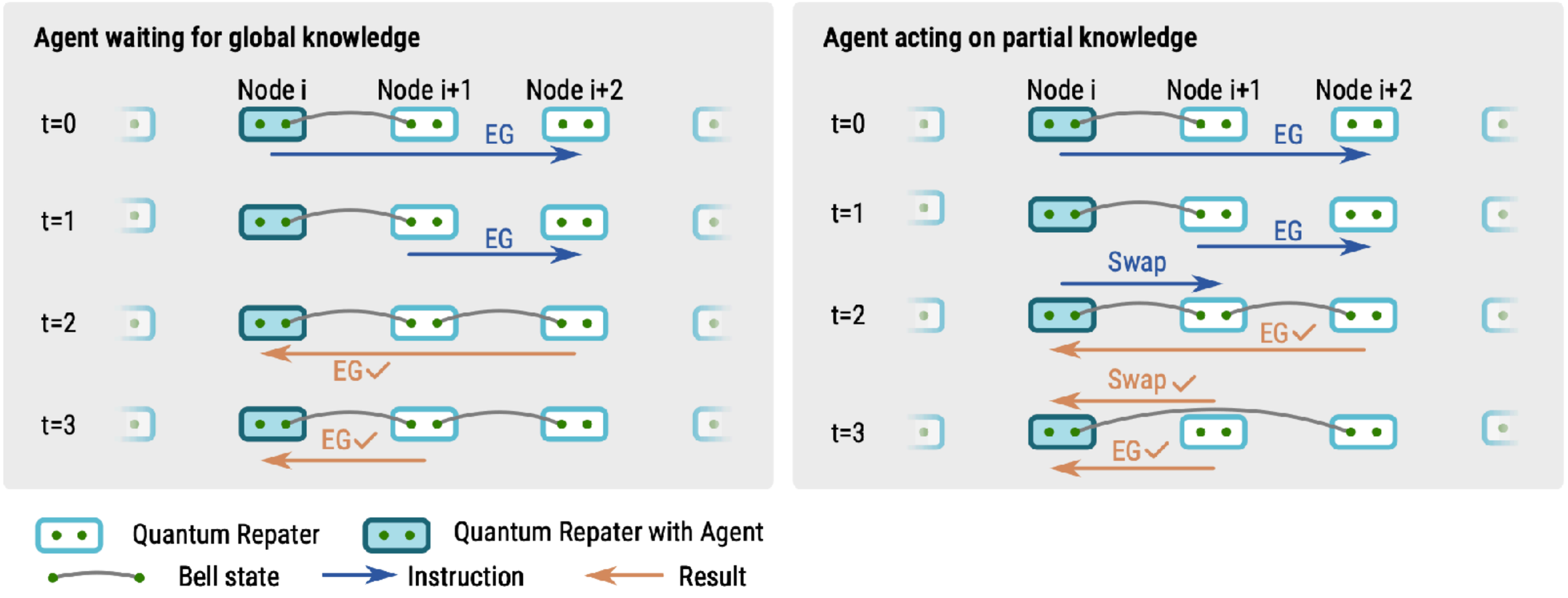


But in reality, we need to take classical communication into account

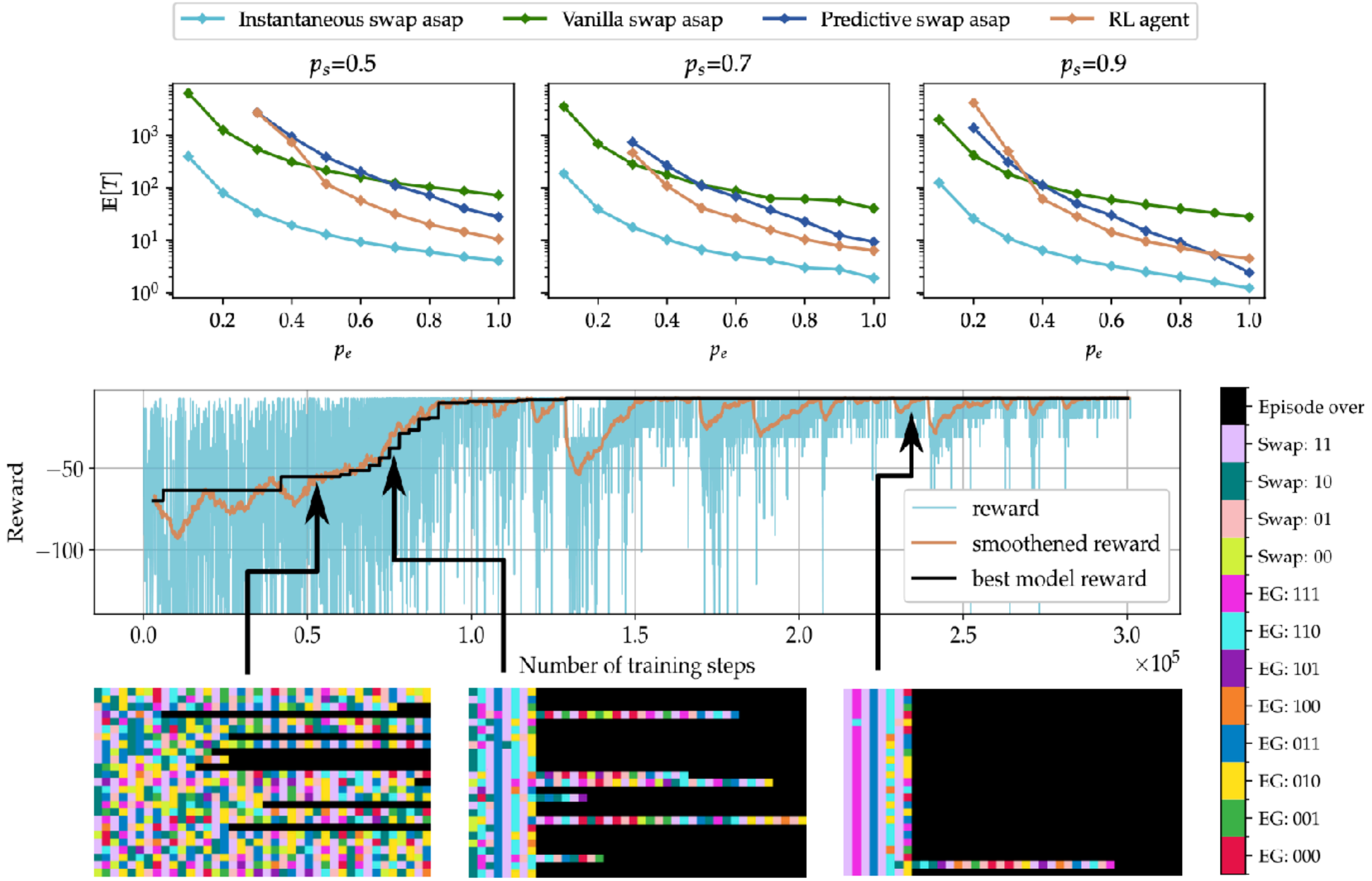
(We are considering a single agent for now, not one at each node)



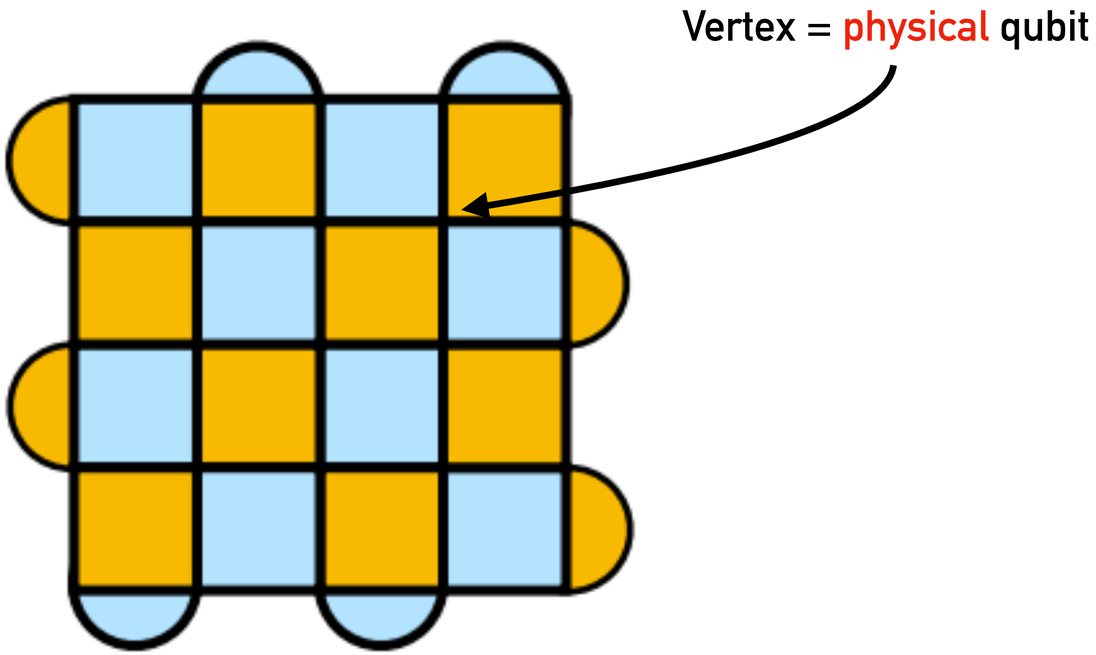
An RL agent *could* help us find a better strategy with CC



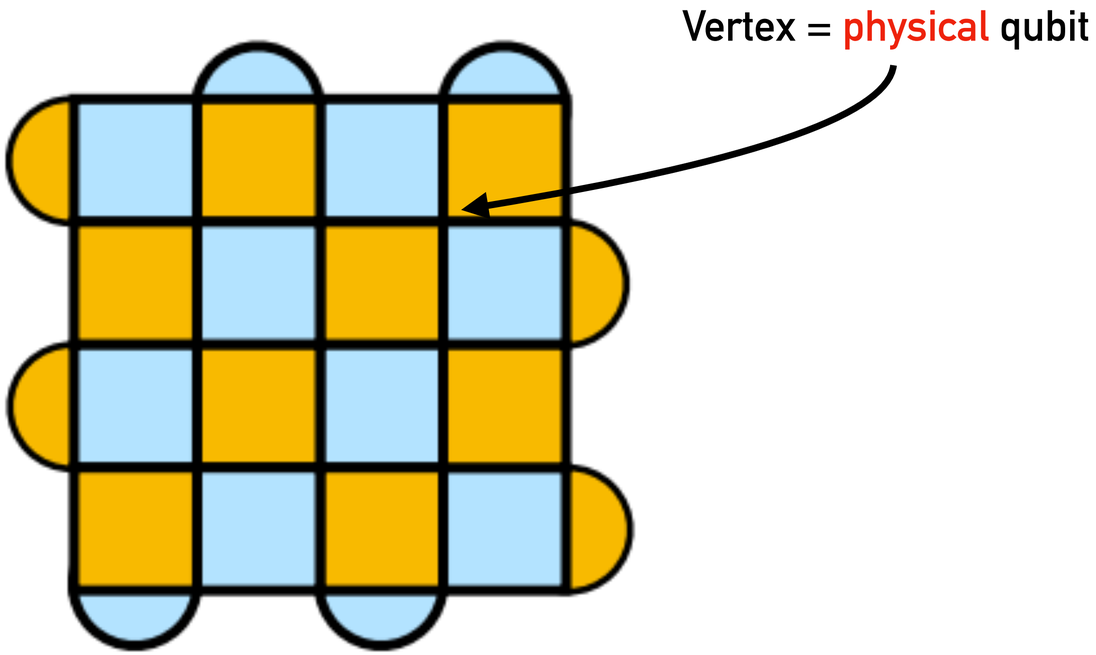
Compared to heuristic benchmarks, the RL agent performs well



A simple quantum computer with one logical qubit

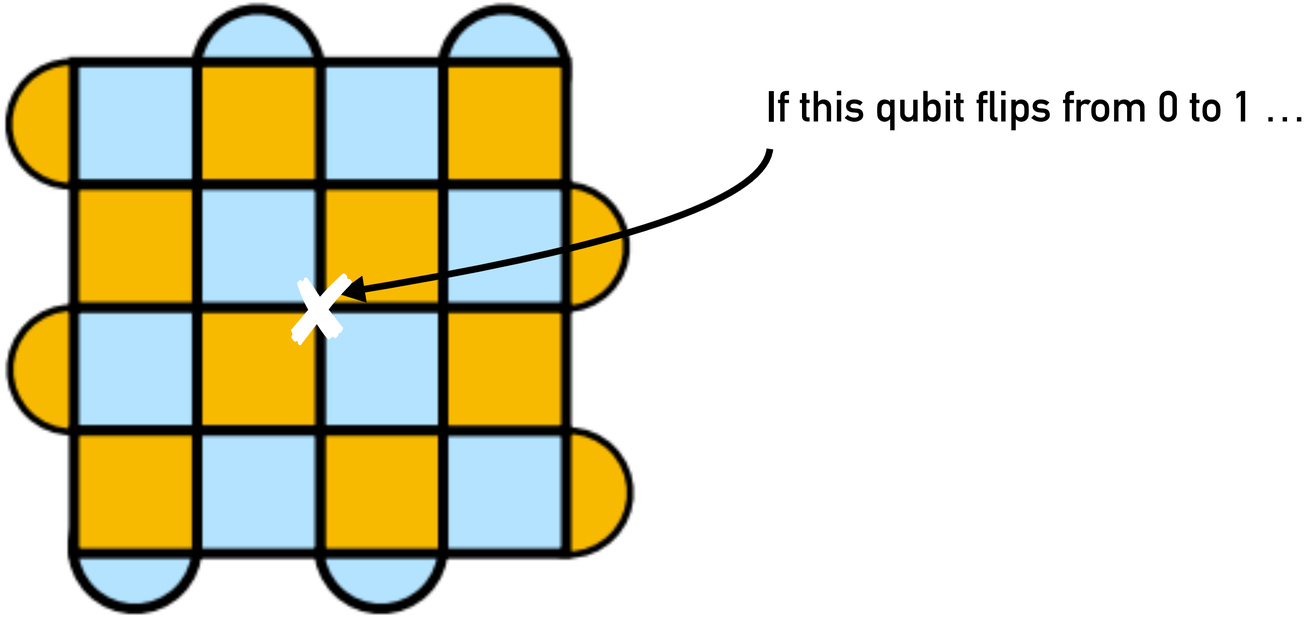


A simple quantum computer with one logical qubit

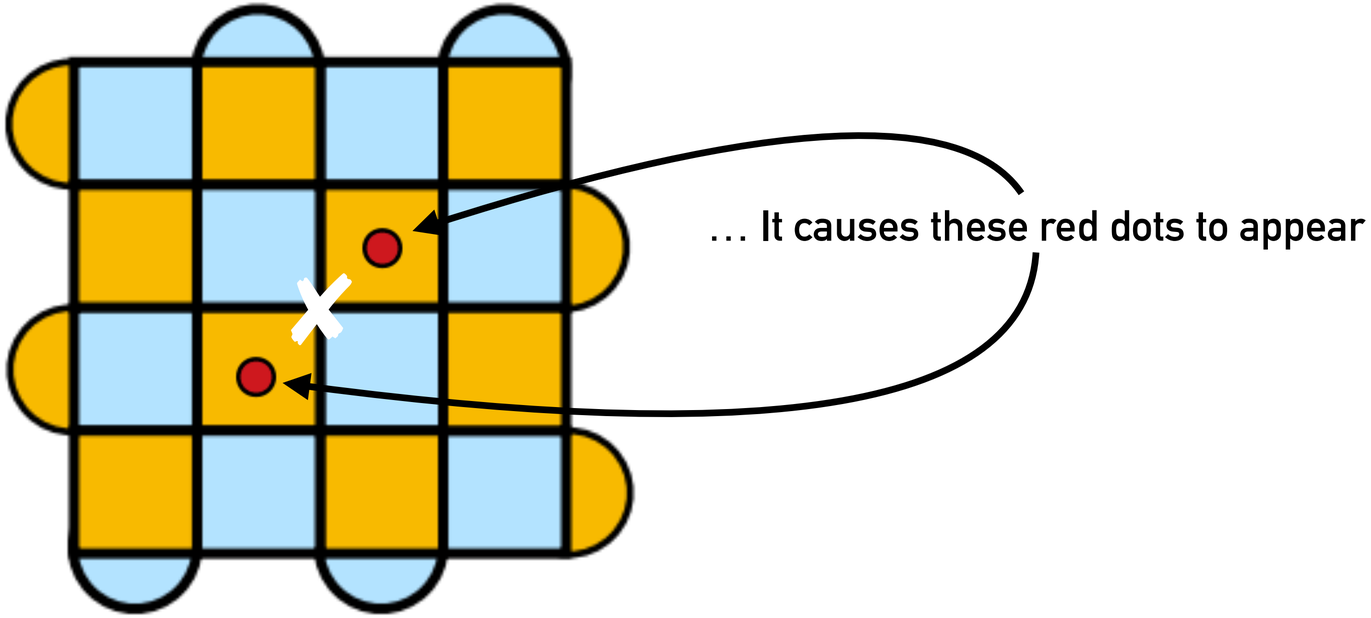


The full state of all these qubits itself represents a “logical” qubit

Qubit errors show up as red dots on the orange squares

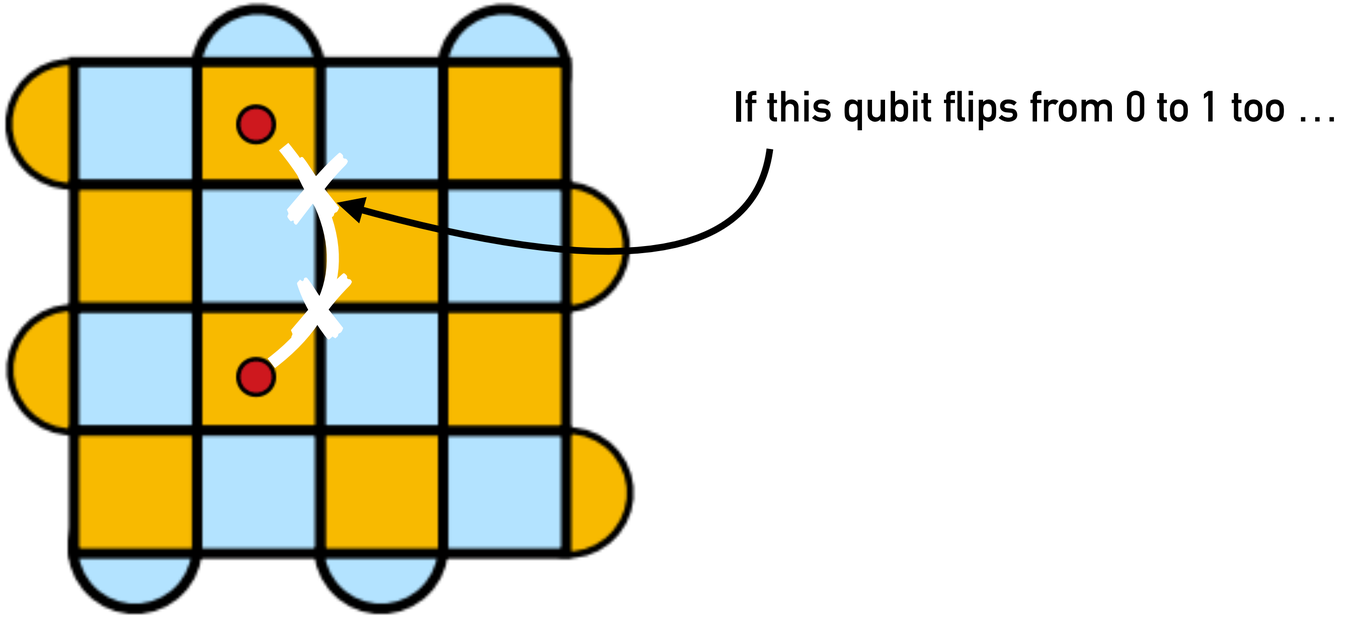


Qubit errors show up as red dots on the orange squares



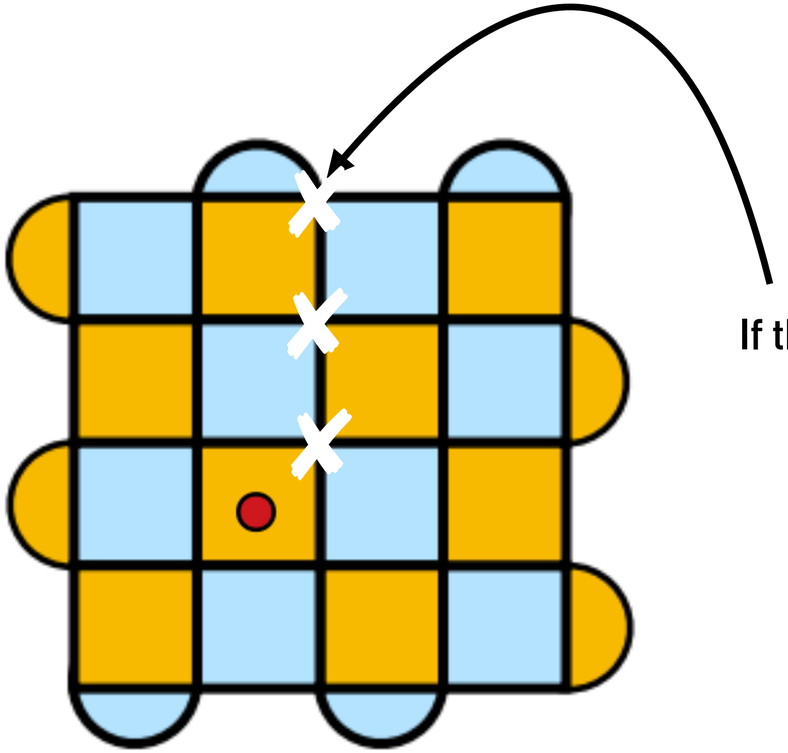
We can *<u>not</u>* look at the flipped qubits, only the red dots!

Multiple errors cause red dots to change position



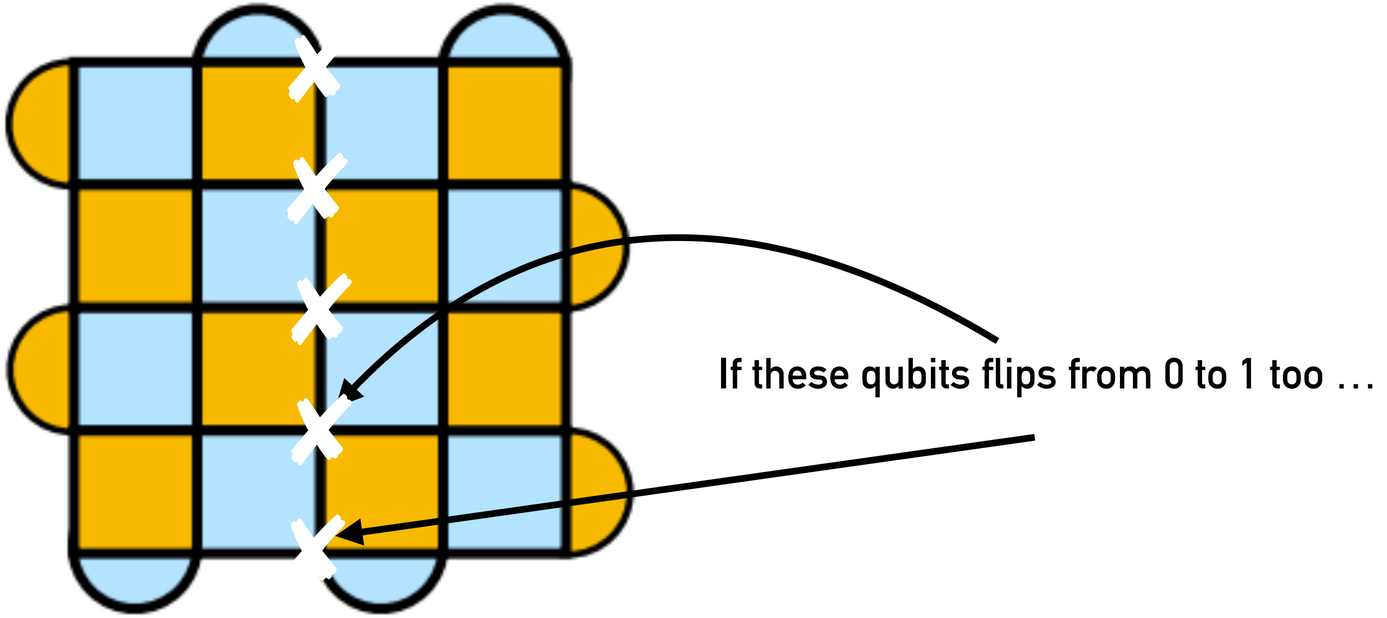
Only an odd number of red flags is visible

Red dots can (dis)appear at the edges



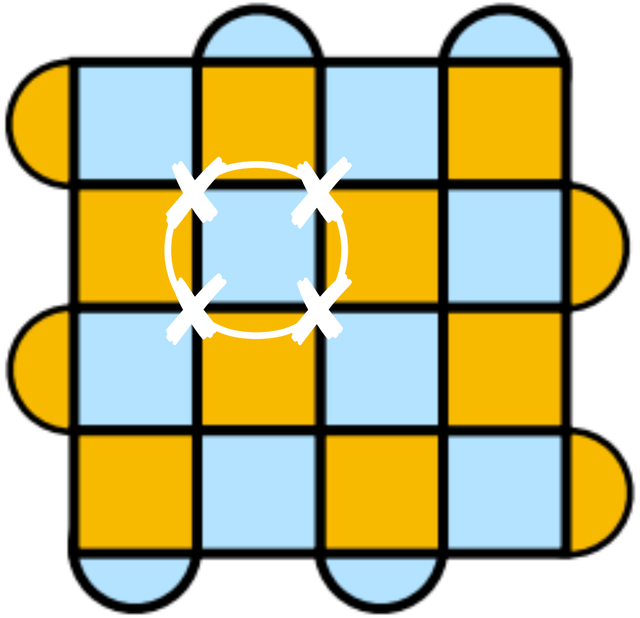
If this qubit flips from 0 to 1 too …

If a string of errors connects the edges, it is impossible to find out which qubits had errors



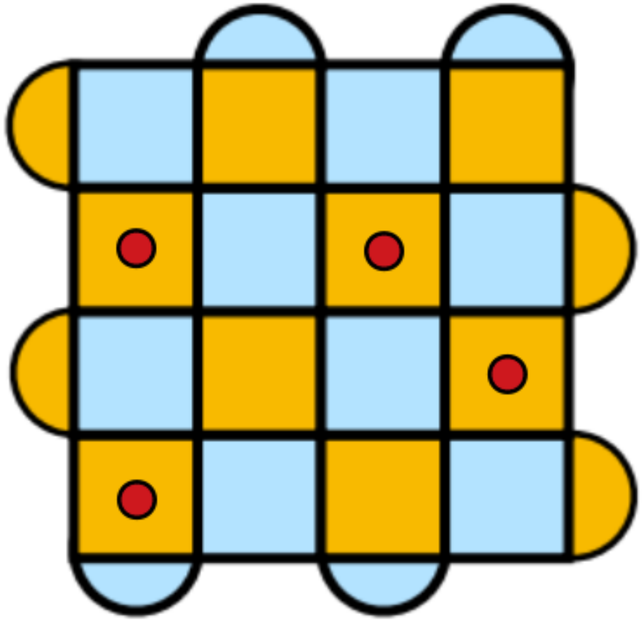
A whole column of qubits has errors, but we can’t see it!
No more red dots, so looks identical to starting point

Here is a correction with which we would have won



Closed loop (contractible) also removes syndrome, but no logical error (doesn’t connect edges!)

So quantum error correction is like a board game!

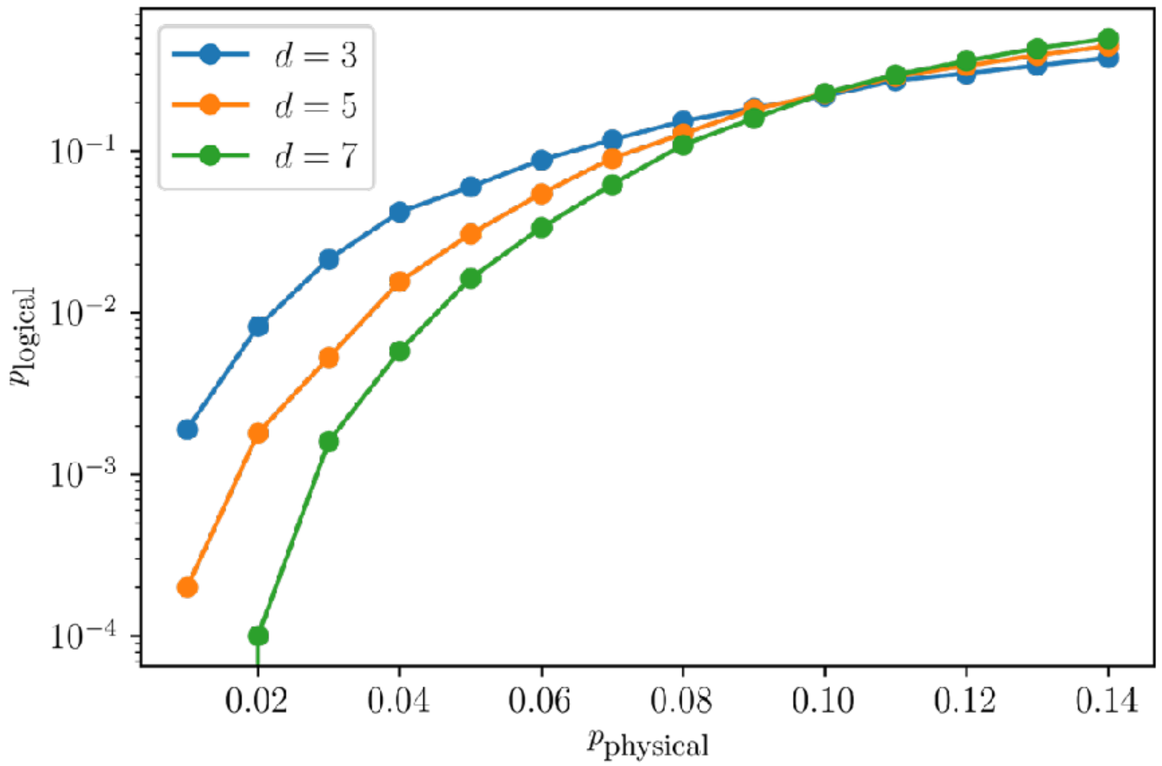


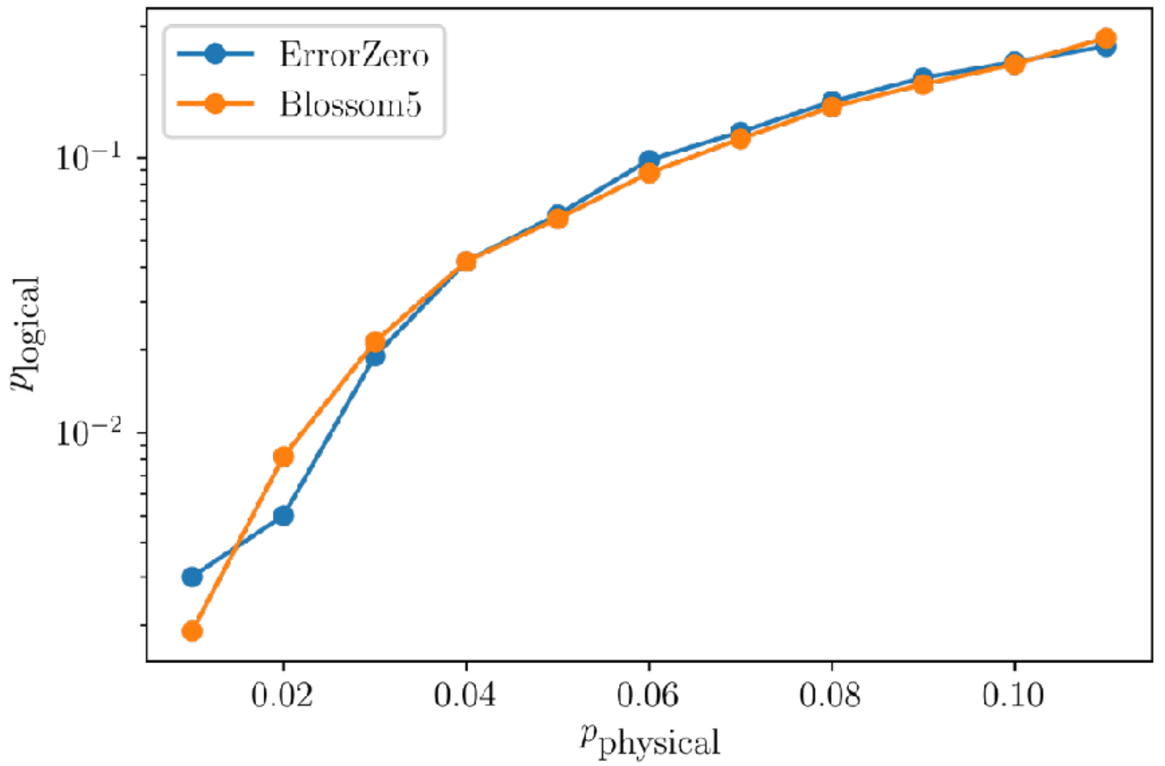
Given red dots, find out which qubits flipped (the errors)

Game over if a string of errors connects one edge to the other

Reinforcement learning can do quantum error decoding

Blossom (Minimum Weight Perfect Matching)





Reinforcement learning can do quantum error correction