Trustworthy AI Autonomy

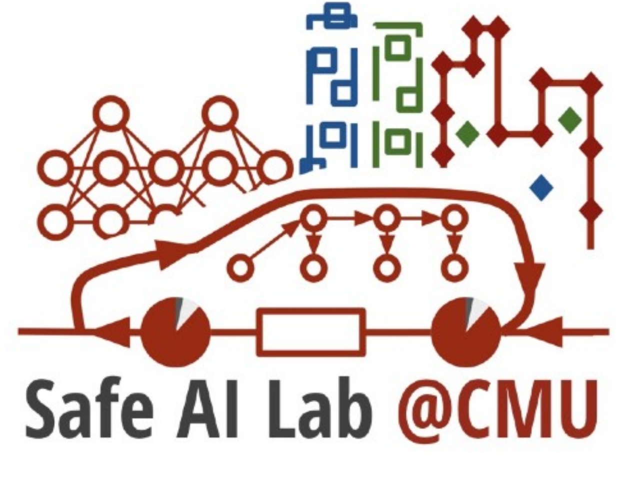
M2-1: Model-free Deep RL

Ding Zhao

Assistant Professor

Carnegie Mellon University

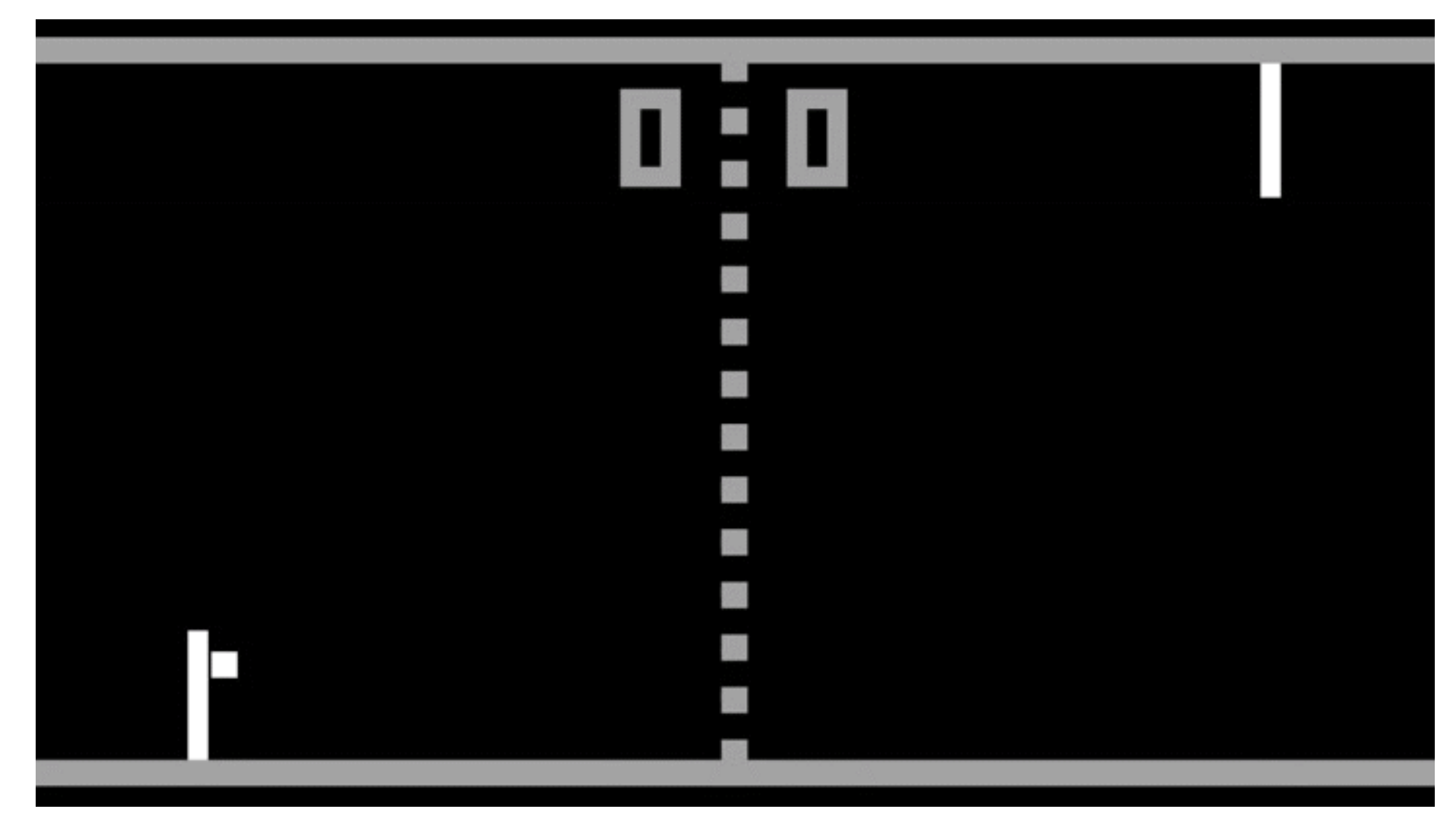
2022 @ Ding Zhao

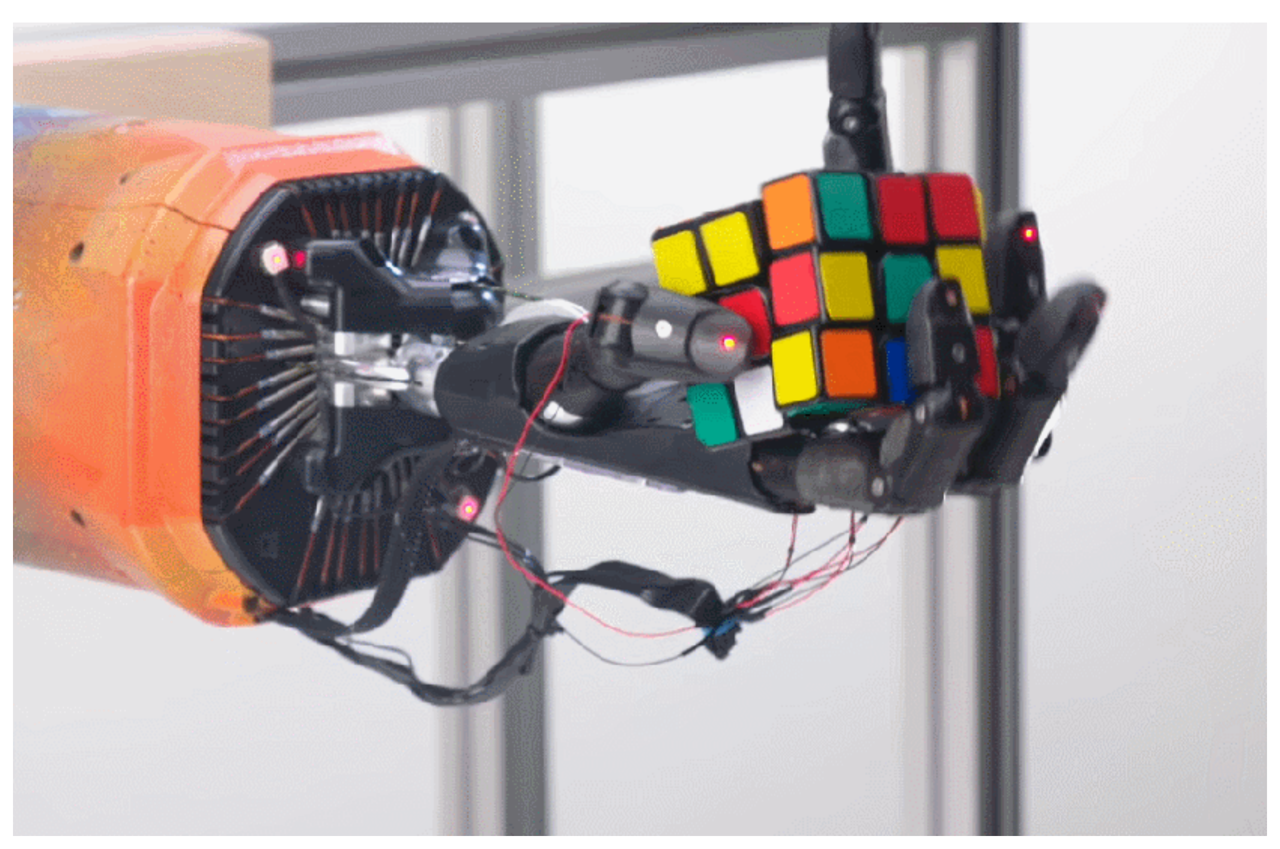


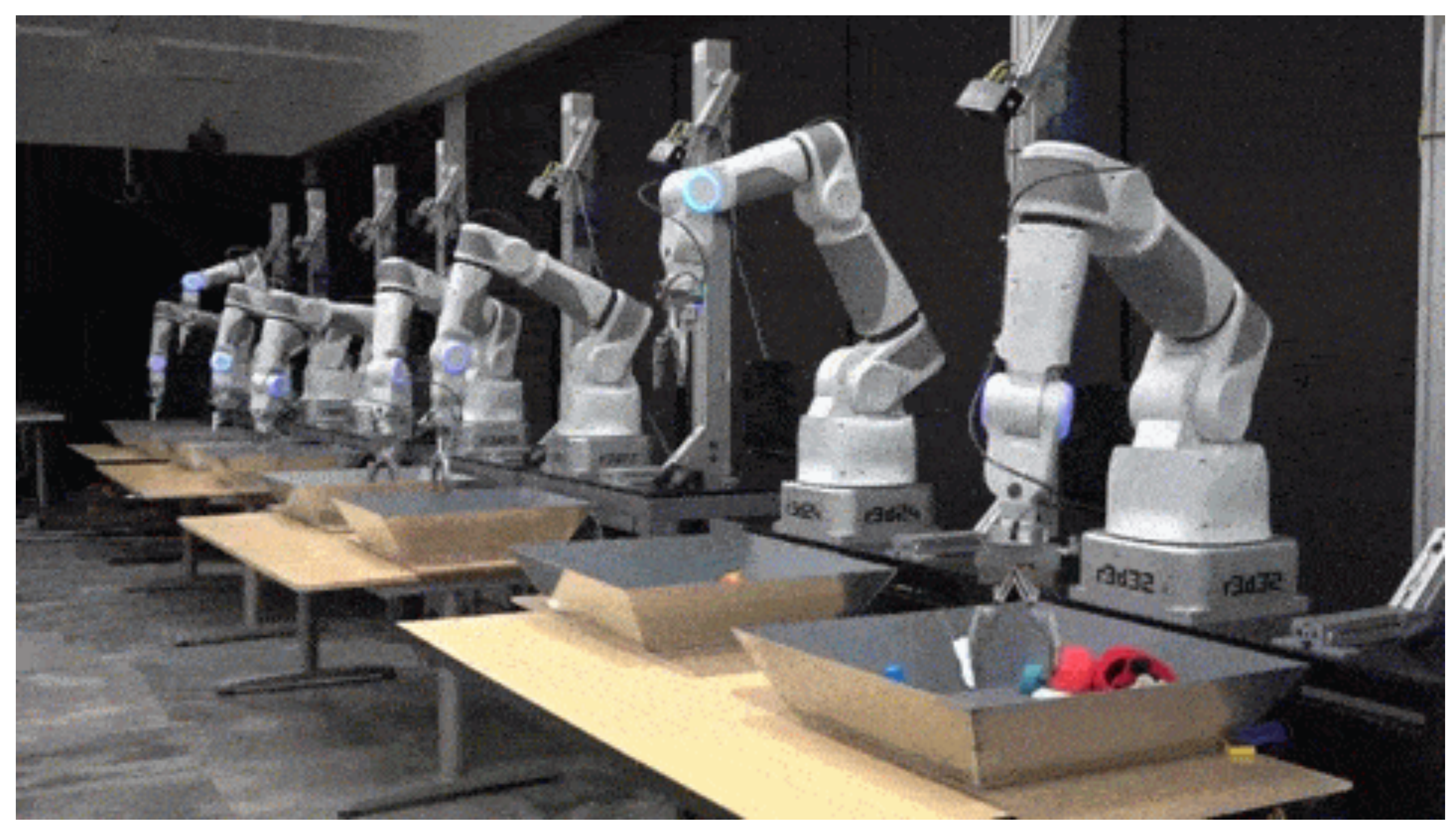
Plan for today

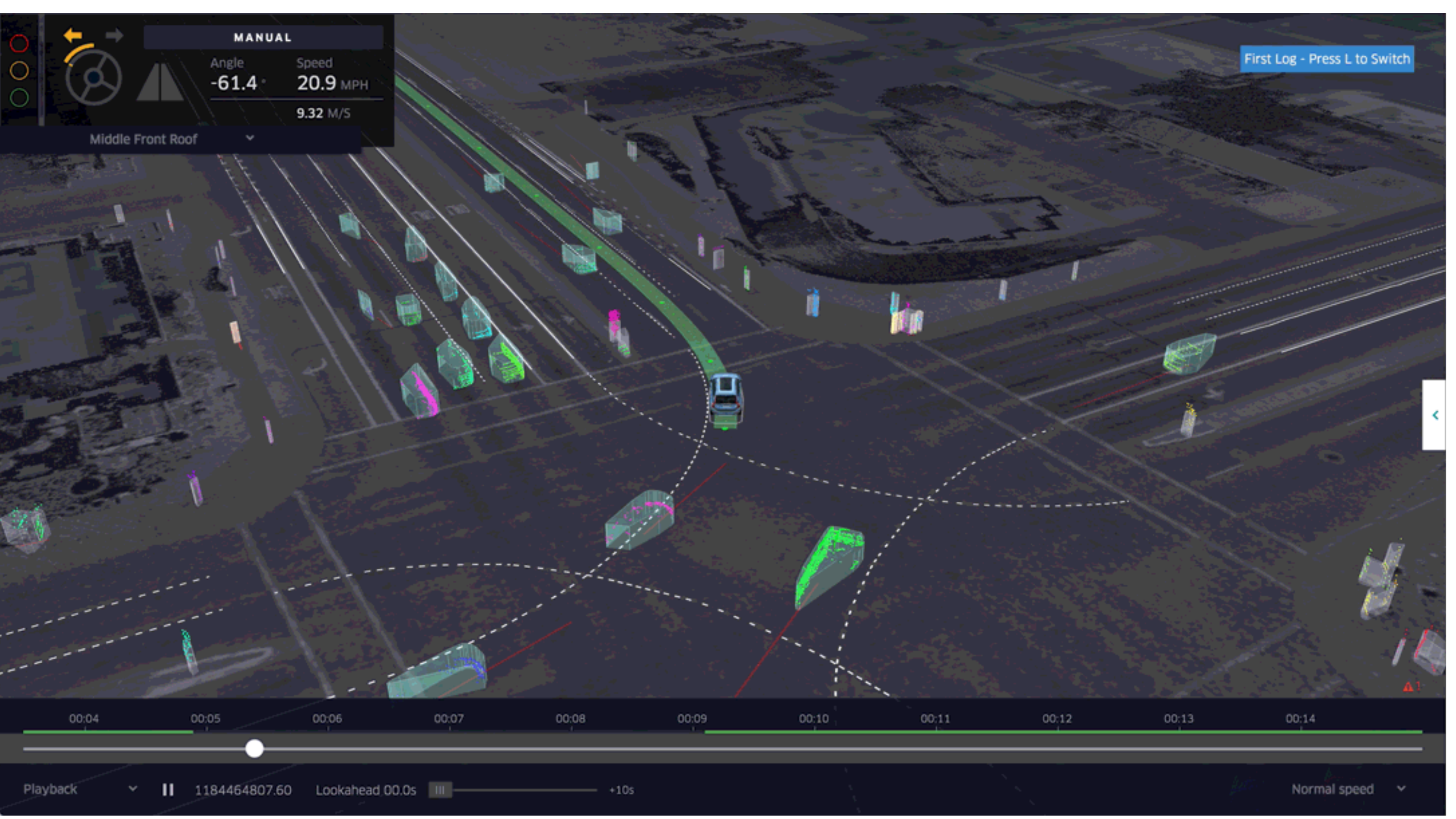
- Sequential decision making: MDP, POMDP
- Imitation learning
  - Behavioral cloning, Dagger
- Model-free reinforcement learning
  - Value-based: DQN
  - Policy-based: REINFORCE, TRPO, PPO
  - Value-based+Policy-based: DDPG

Sequential decision making

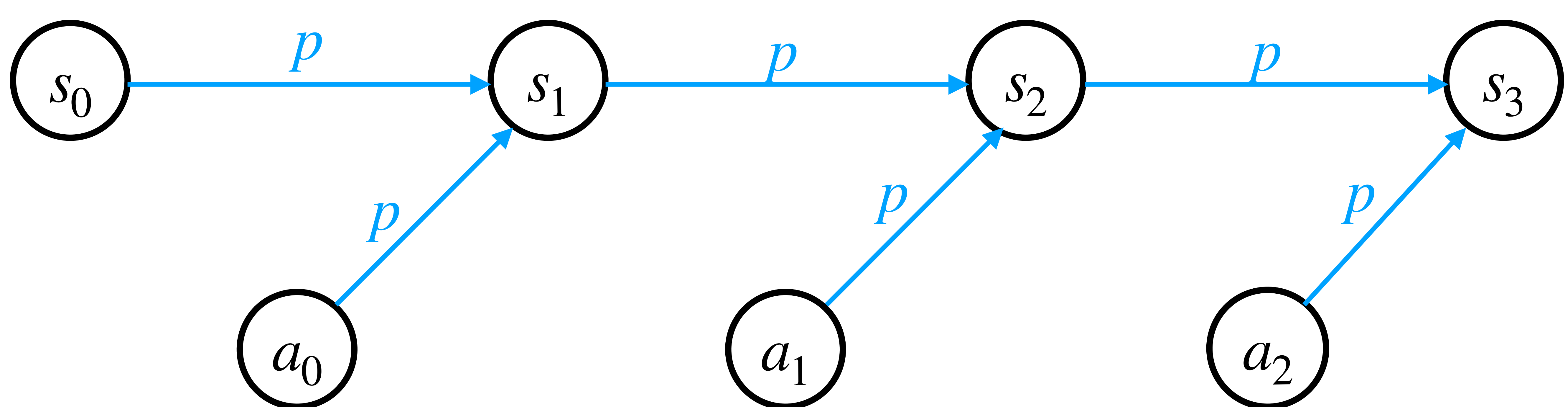








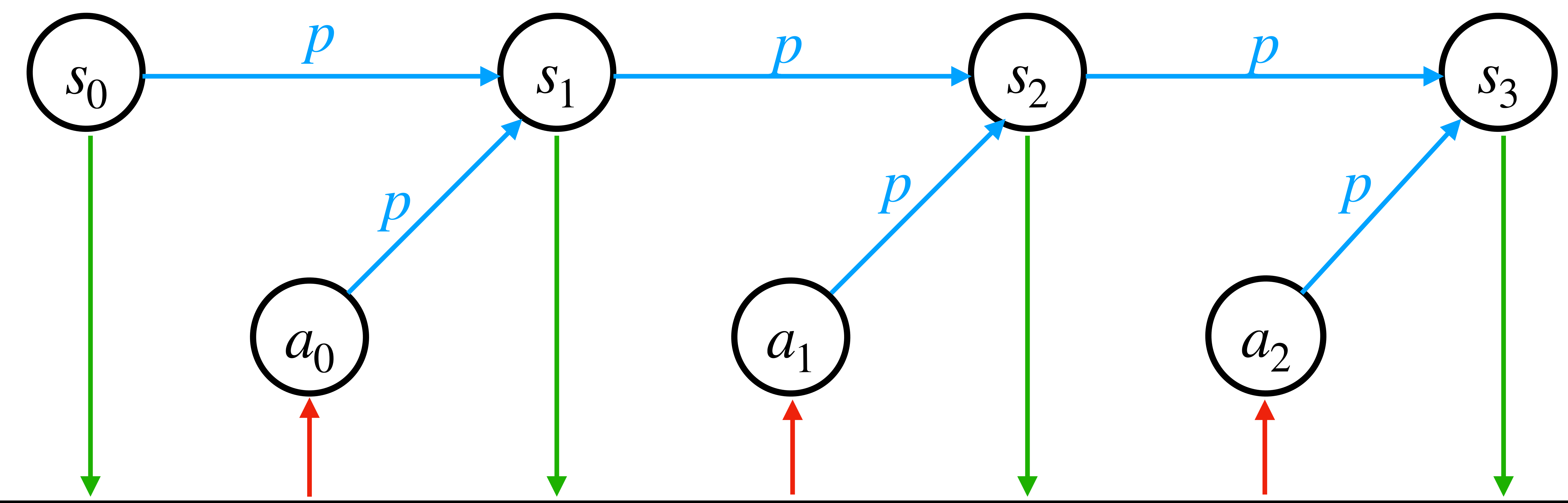
States and actions



- Need some abstraction to start with
- States: all the info you need to make good decisions
- Actions: things we can do to change states

$$s_{t+1} \sim p\left(\cdot \mid s_t , a_t\right)$$

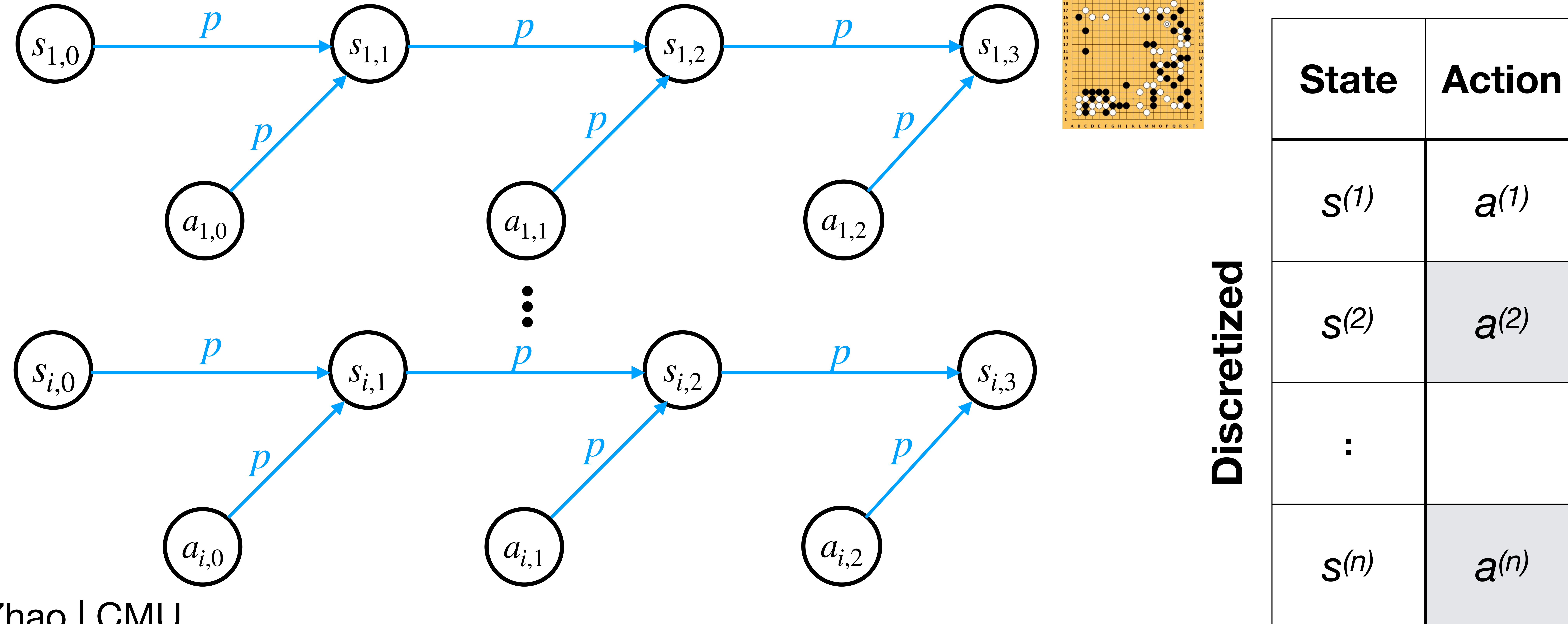
Imitation Learning



- Imagine you attend a tournament of game Go and you have no clue
- But there is a master Shifu will tell you what to do, which you just follow

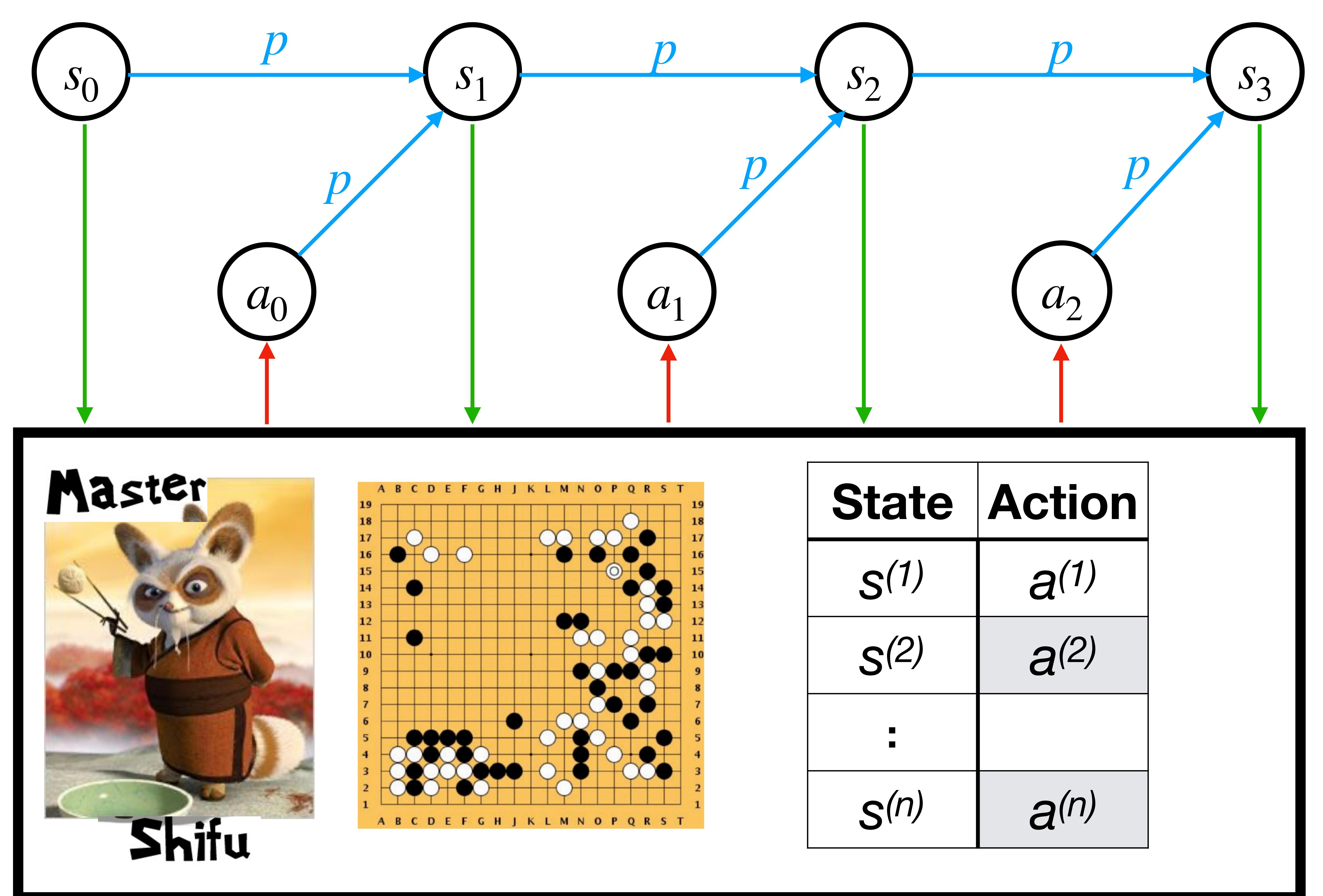
Imitation Learning - tabular

- IL-1.0: Look-up table
- Accumulate a big table of the state-action pair. If see $s^{(i)}$, do $a^{(i)}$.



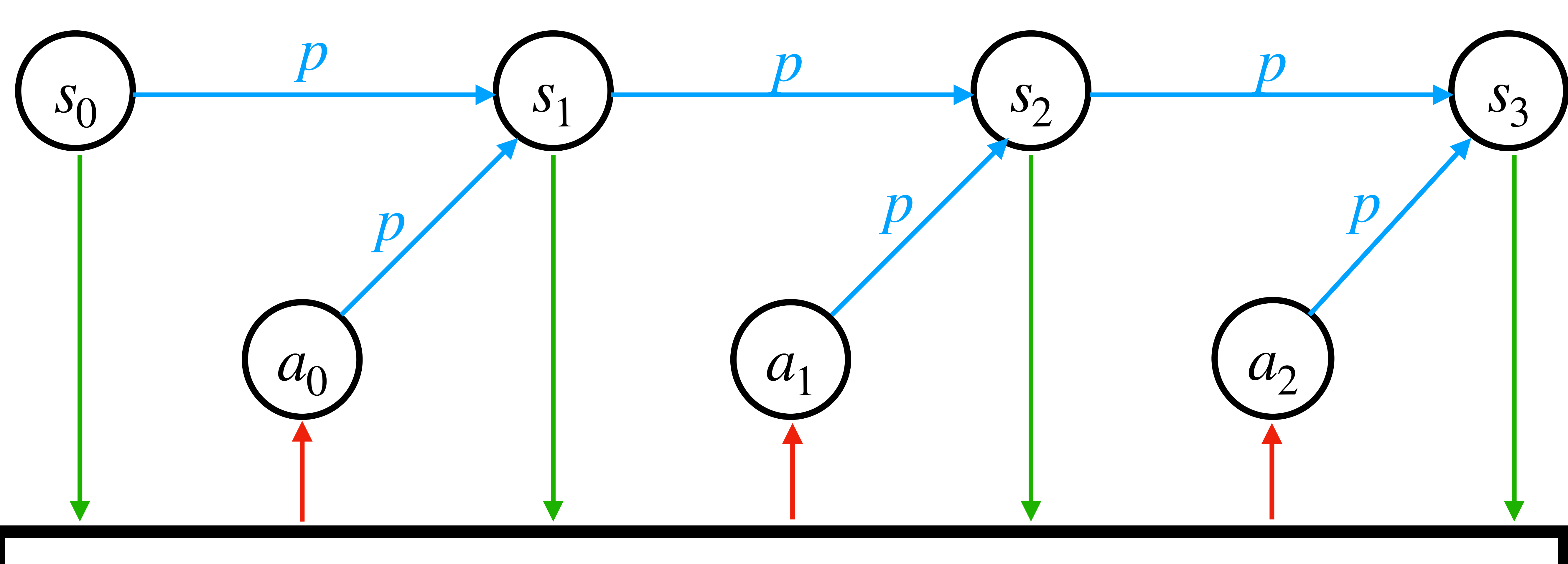
State	Action
$s^{(1)}$	$a^{(1)}$
$s^{(2)}$	$a^{(2)}$
:	
$s^{(n)}$	$a^{(n)}$

Imitation Learning - functional



- Issue of IL-1.0:
  - The table could be very long: $2^{19^2} \approx 10^{108} > 10^{80}$ (# of atoms in the universe)
- IL-2.0:
  - We can always represent knowledge as a table or as a **function**
  - Decision policy $a_t = f_\theta(s_t \,|\, \mathscr{D} = \{s_t, a_t\}_i)$
  - f_θ can be a linear model $a_t = \theta^T \phi(s_t)$, where $\phi()$ are pre-defined feature functions
  - f_θ can also be a neural network

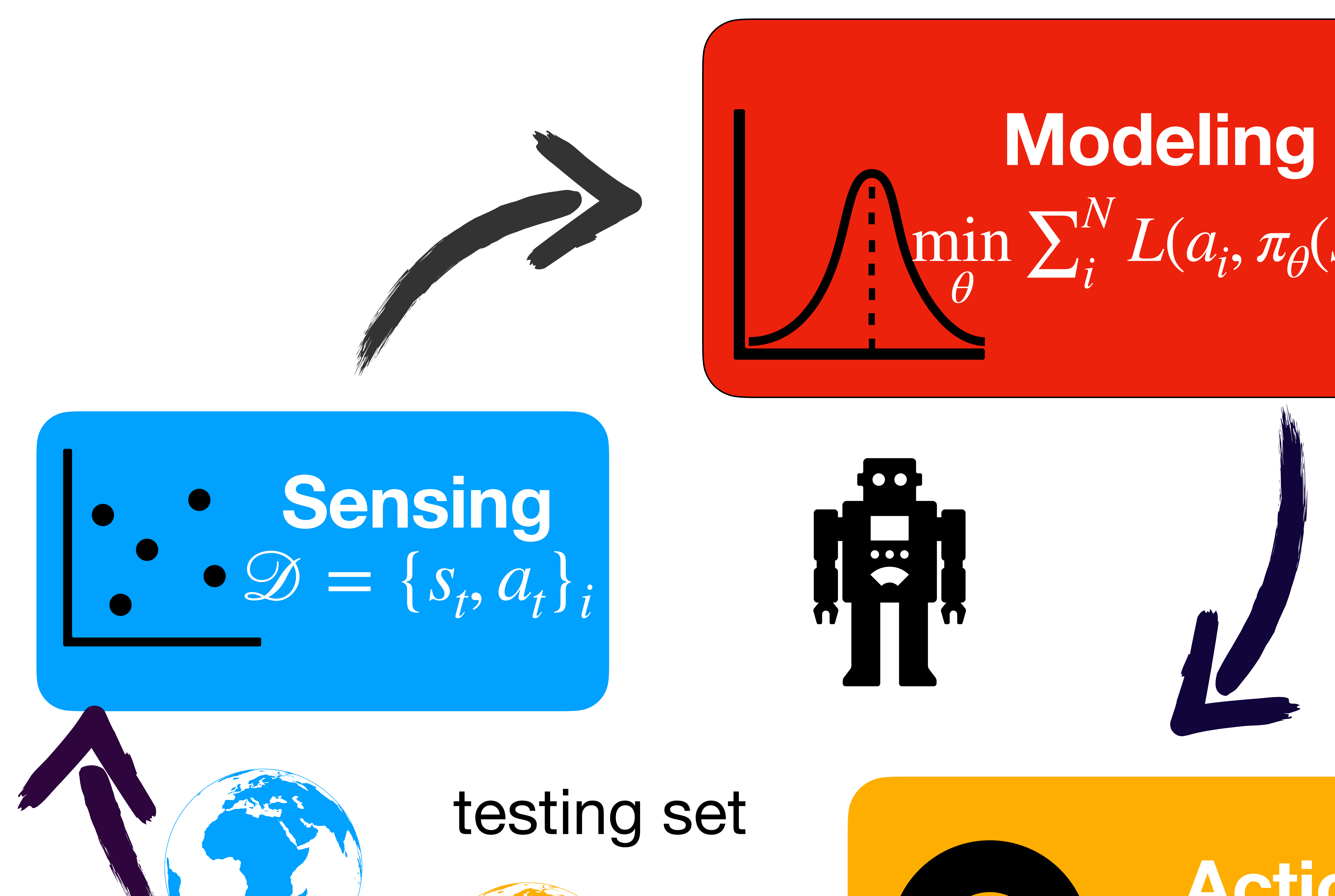
Imitation Learning - statistically

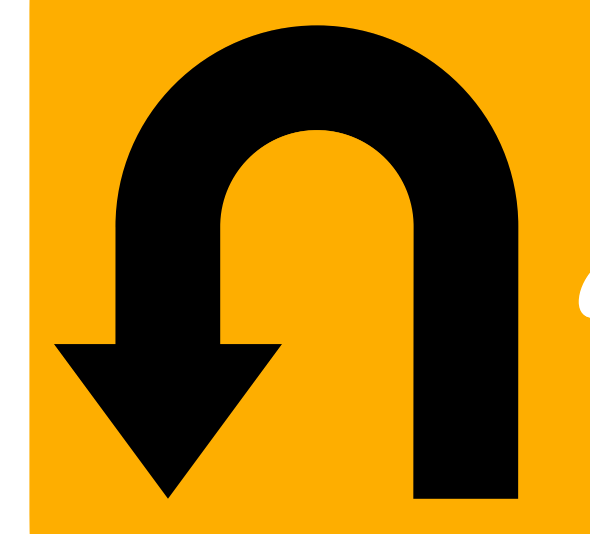


	$a^{(1)}$	...	$a^{(m)}$
$s^{(1)}$	4	...	50
$s^{(2)}$	10	...	2
:			
$s^{(n)}$	0	...	1

- Issue of IL-2.0
  - Shifu's decisions could be multi-modal. An averaged decision may be meaningless.
- IL-3.0
  - Count the frequency of each (s, a) pair
  - When $s_t = s^{(i)}$, sample all $a^{(i)}$ with the weights on the corresponding row
- IL-4.0
  - For the same reason, *i.e.*, curse of dimension, we may use a **statistical function** to approximate the table
  - Decision policy $a_t \sim \pi_\theta(\cdot \mid s_t\ ,\ \mathscr{D} = \{s_t, a_t\}_i)$

Imitation Learning/Behavioral Cloning





- Divide data into training and testing sets
- Take collected "data" as the expert
    - Train a model with a loss function
    - Make decision

$$a_t \sim \pi_\theta(\cdot \mid s_t\ ,\ \mathcal{D} = \{s_t, a_t\}_i)$$

Does it work?

Yes!

Case study 1: Nvidia Autonomous Car (2016)

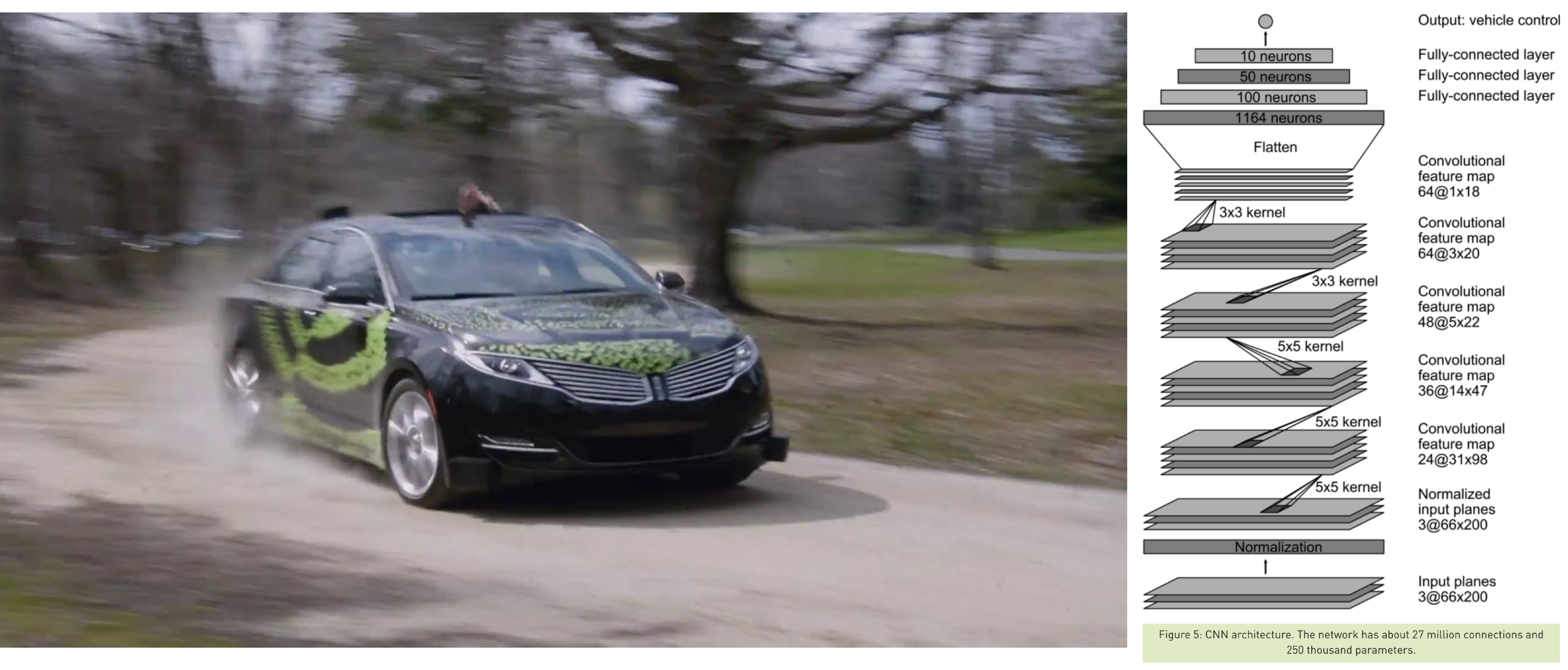


Figure 5: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

Case study 1: Nvidia Autonomous Car (2016)

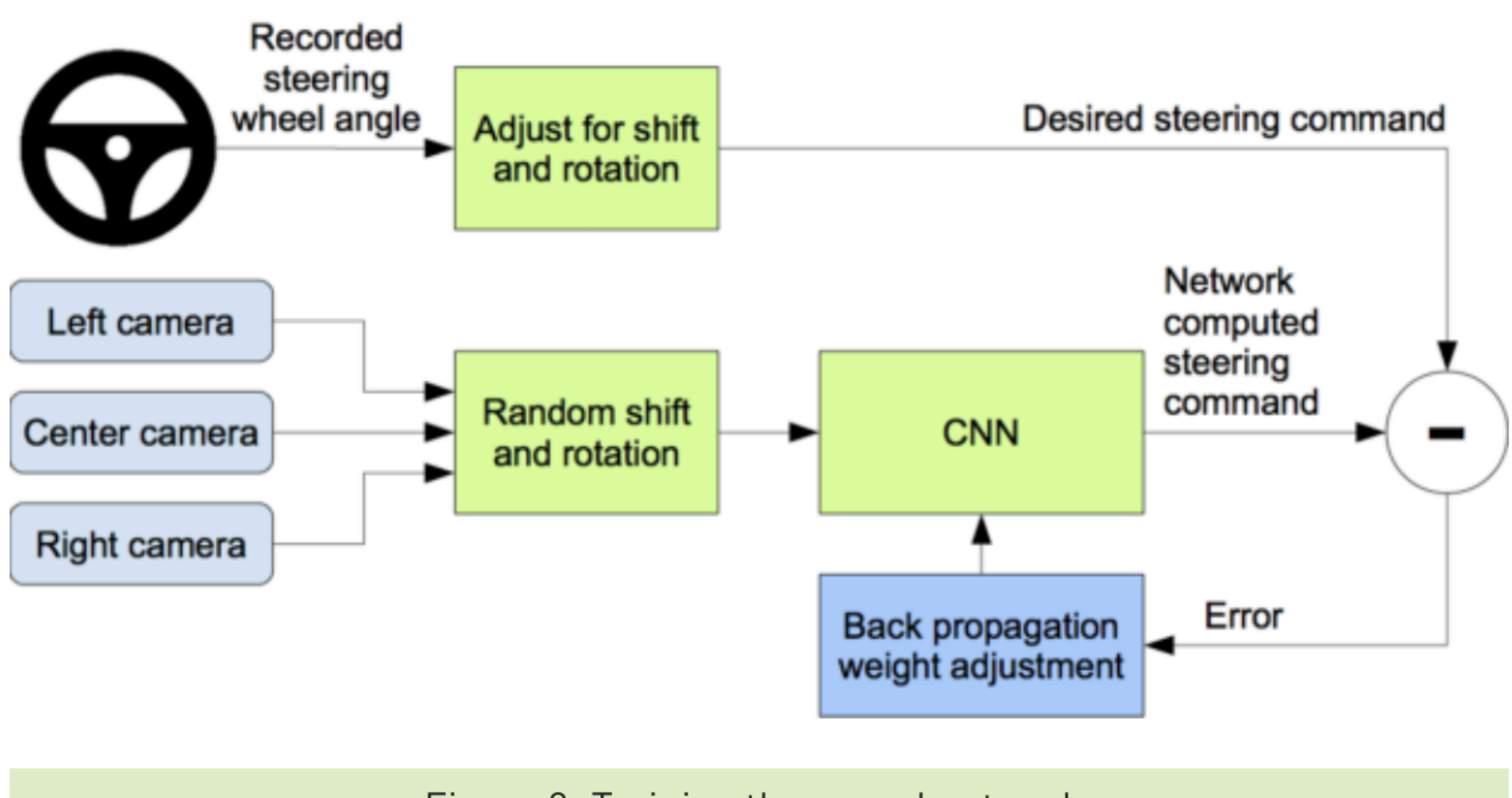


Figure 3: Training the neural network.

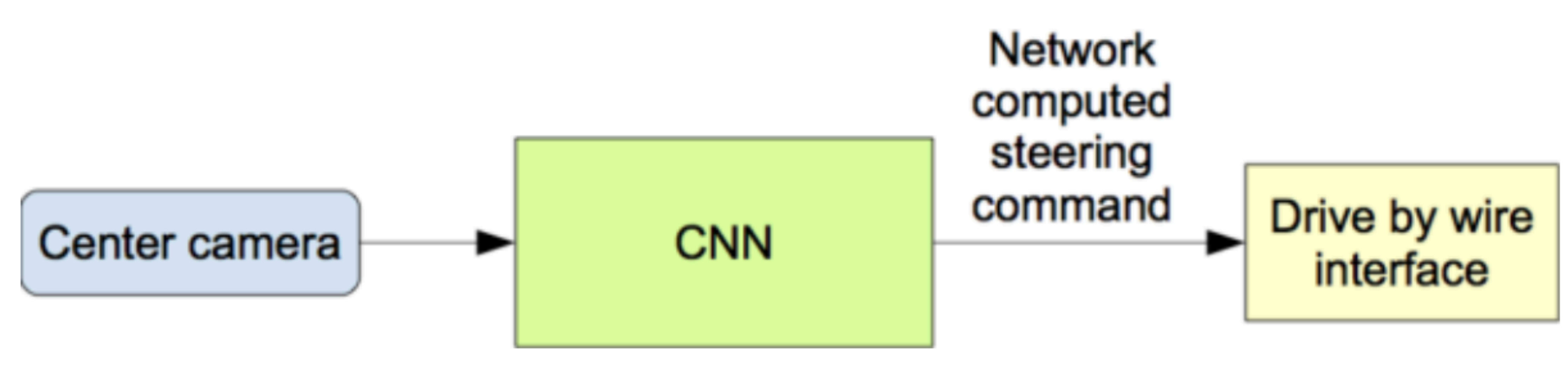


Figure 4: The trained network is used to generate steering commands from a single front-facing center camera.

https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

Case study 1: Nvidia Autonomous Car (2016)

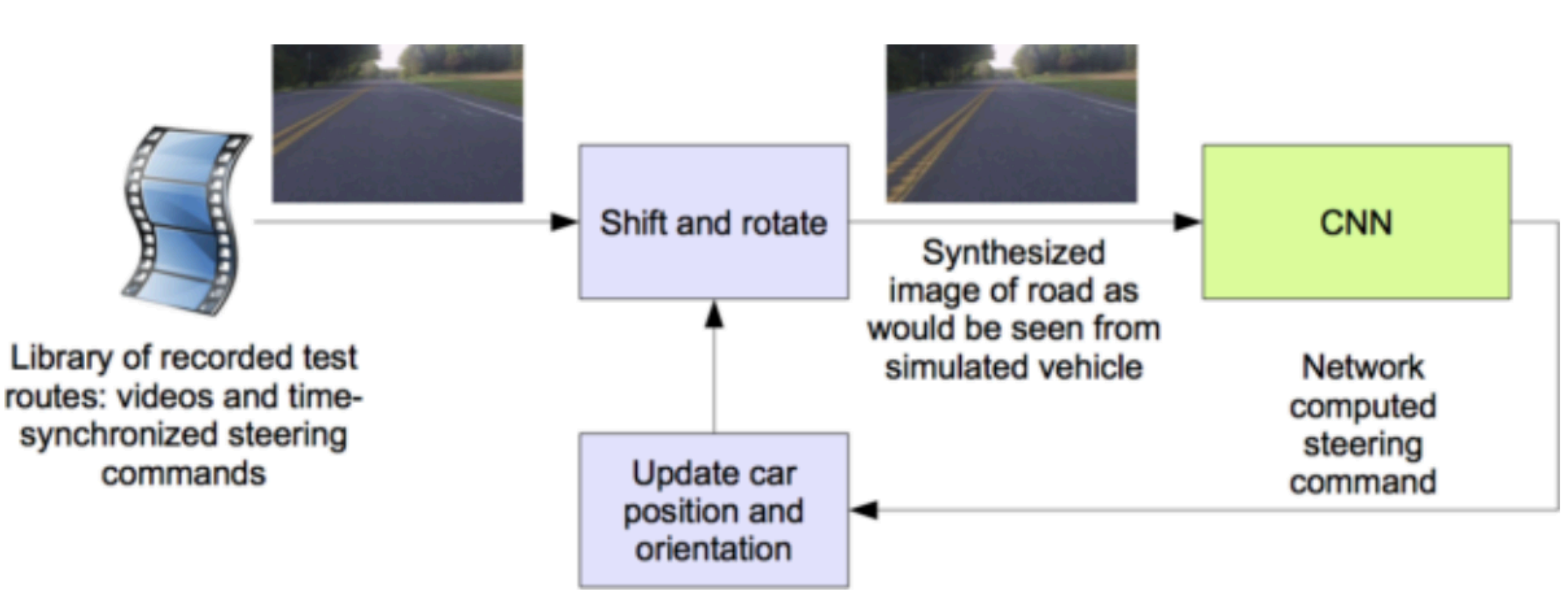
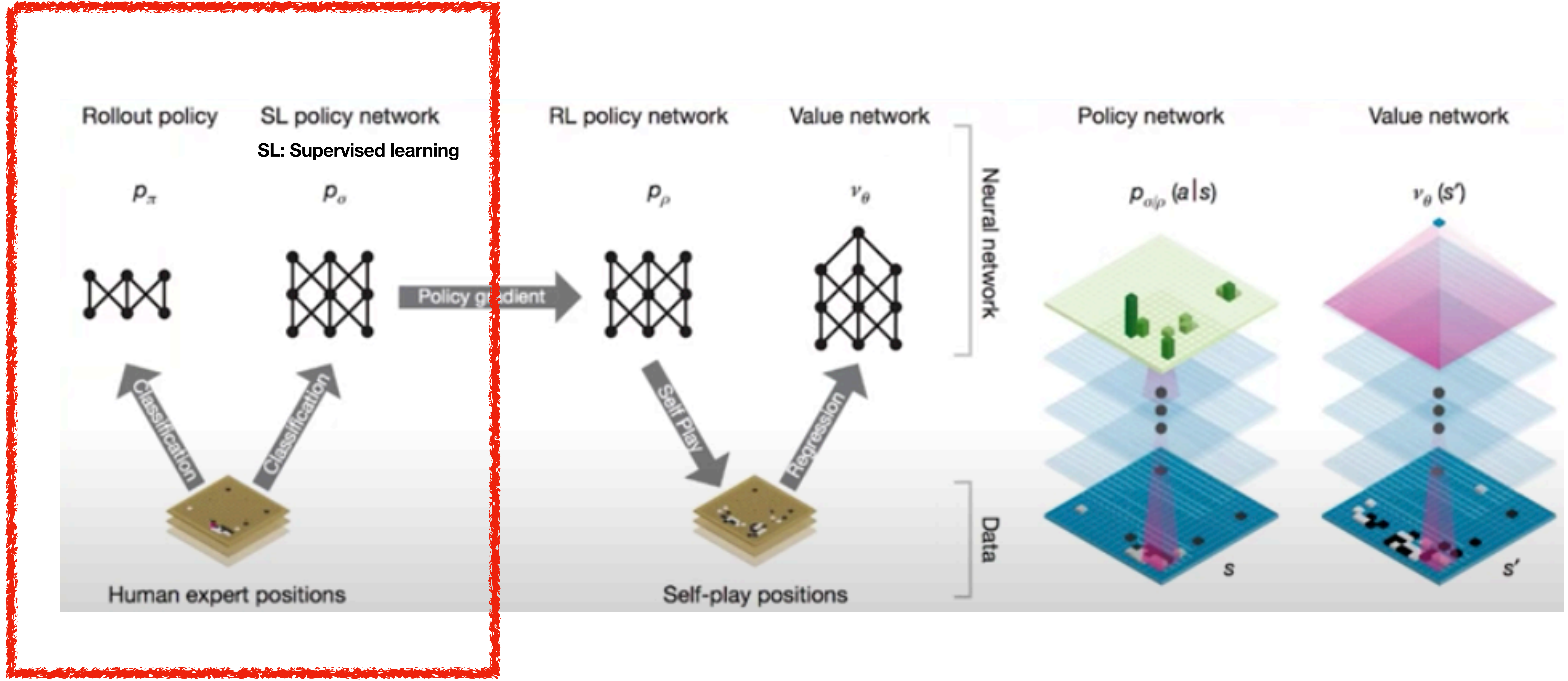


Figure 6: Block-diagram of the drive simulator.

Figure 7: Screenshot of the simulator in interactive mode. See text for explanation of the performance metrics. The green area on the left is unknown because of the viewpoint transformation. The highlighted wide rectangle below the horizon is the area which is sent to the CNN.

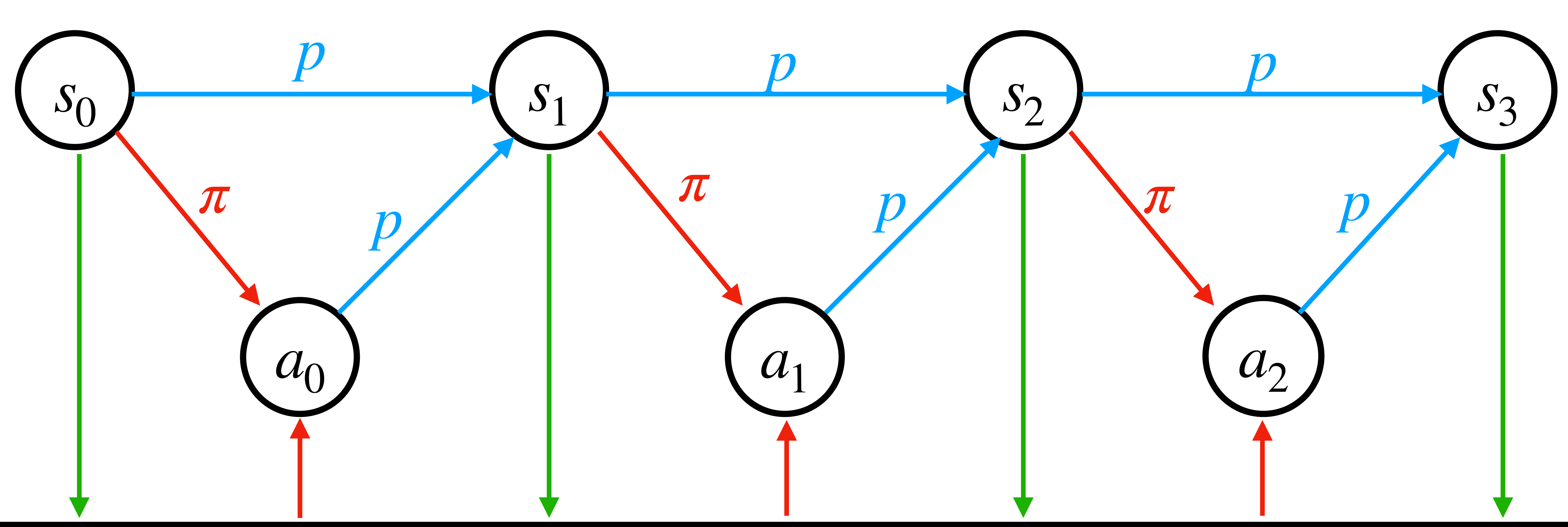
https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

Case study 2: AlphaGo (2016)



Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016).

Imitation Learning - DAgger



	$a^{(1)}$	...	$a^{(m)}$
$s^{(1)}$	4	...	50
$s^{(2)}$	10	...	2
:			
$s^{(n)}$	0	...	1

$\mathscr{D}^{(0)} \rightarrow \pi_{\theta^{(0)}}$

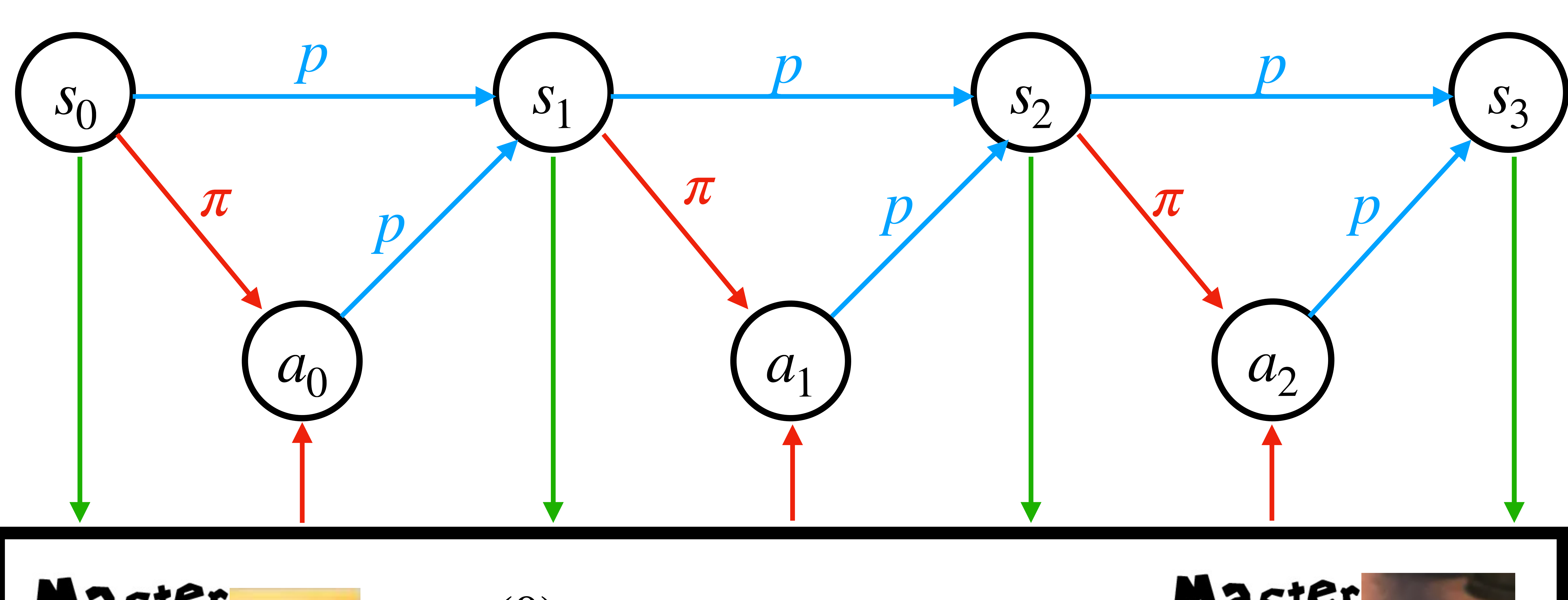
$\rightarrow \tilde{\mathscr{D}}^{(1)} \rightarrow \mathscr{D}^{(1)}$

$\rightarrow \mathscr{D}^{(0)} \cup \mathscr{D}^{(1)} \rightarrow \pi_{\theta^{(1)}}$

⋮

- Issue of IL-4.0
  - Does it always work - NO
  - Lack of enough data to fully train $\pi_\theta(a_t | s_t)$
- IL-5.0: Data Aggregation (DAgger)
  - Train $\pi_{\theta^{(0)}}(a_t | s_t)$ with labelled data $\mathscr{D}^{(0)} = \{s_t, a_t\}_{1:N_0}$
  - Try it out and get a new data set $\tilde{\mathscr{D}}^{(1)} = \{s_t, \tilde{a}_t\}_{N_0+1:N_1}$
  - Ask Shifu to relabel them: $\mathscr{D}^{(1)} \leftarrow \tilde{\mathscr{D}}^{(1)}$
  - Retrain $\pi_{\theta^{(1)}} \leftarrow \mathscr{D}^{(0)} \cup \mathscr{D}^{(1)}$

Imitation Learning - Issues

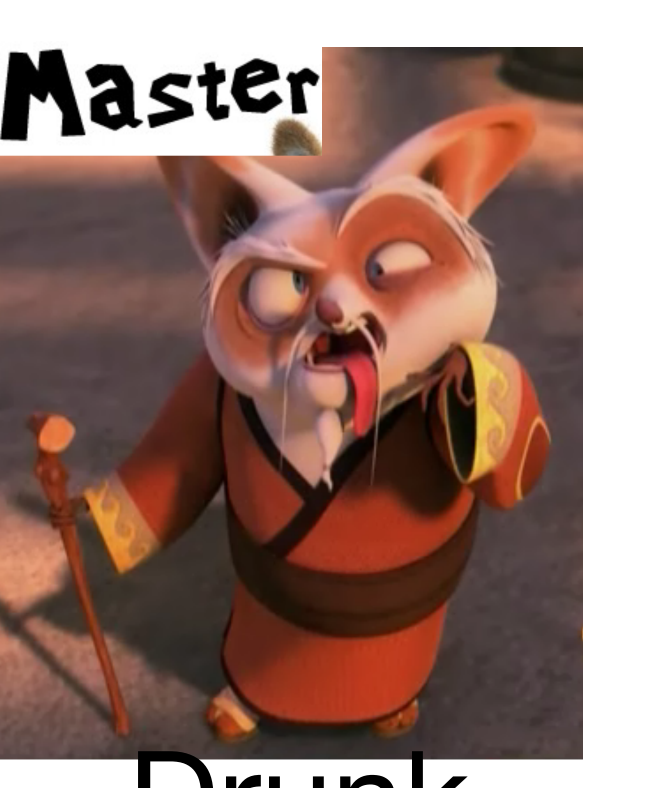


$\mathscr{D}^{(0)} \rightarrow \pi_{\theta^{(0)}}$

$\rightarrow \tilde{\mathscr{D}}^{(1)} \rightarrow \mathscr{D}^{(1)}$

$\rightarrow \mathscr{D}^{(0)} \cup \mathscr{D}^{(1)} \rightarrow \pi_{\theta^{(1)}}$

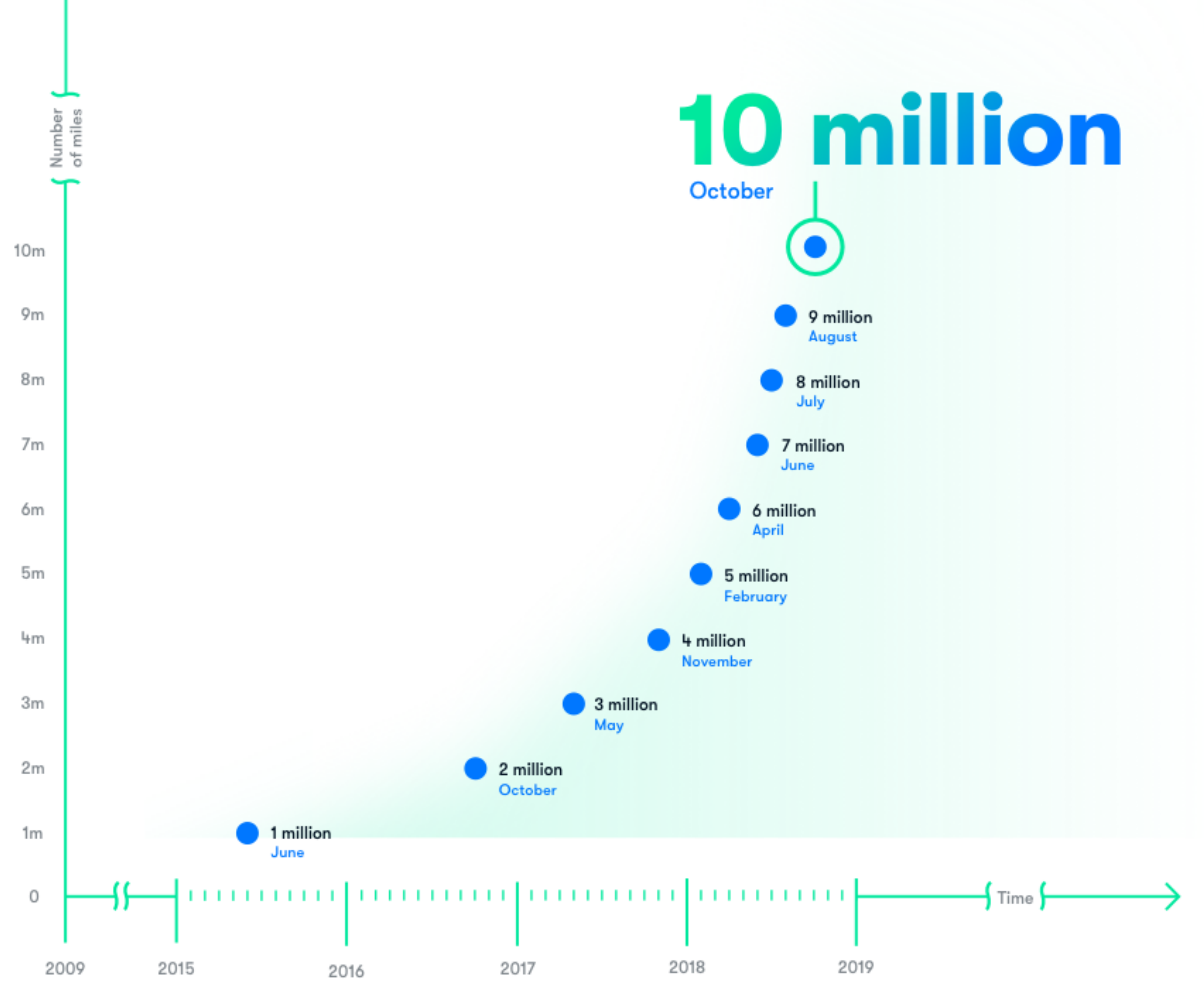
$\vdots$



Drunk

- Issue of IL-5.0
  - Shifu may make mistakes, e.g., normal human drivers, could be imperfect, e.g. road raging, panic in a collision, sleepy
  - Too expensive
    - 3,000 miles to train Nvidia autonomous vehicles
    - 20,000,000 miles of testing on the public road by Waymo
  - Some experiments are safety critical, e.g. robotics, healthcare

Imitation Learning - Issues



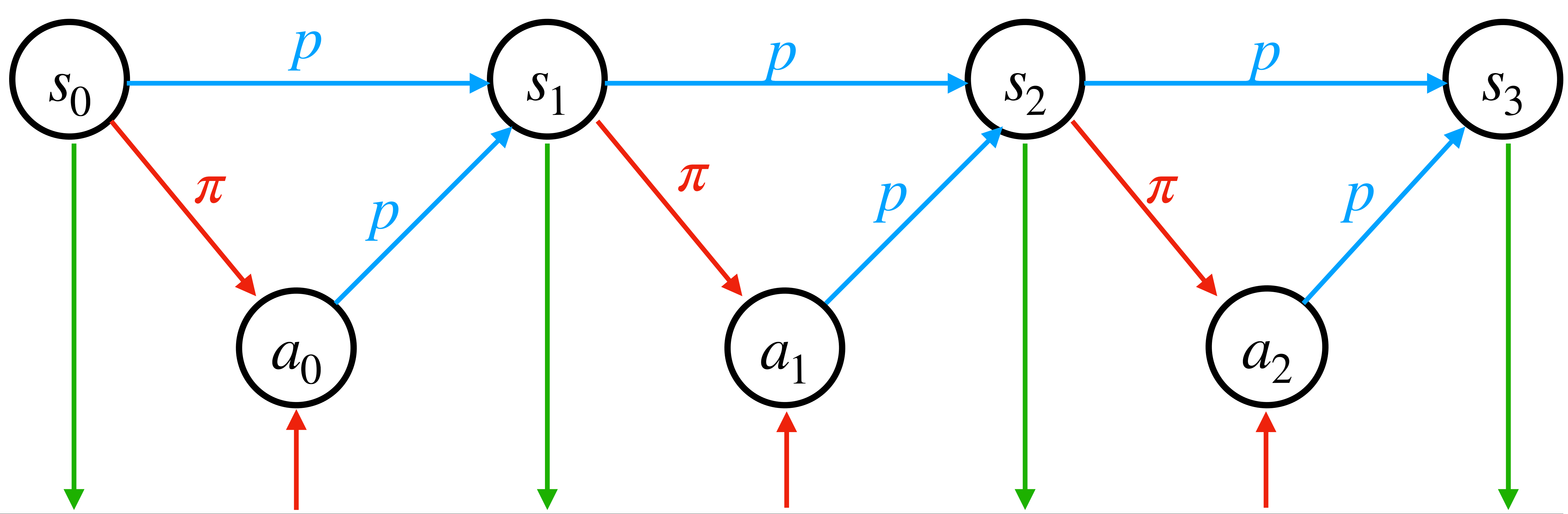
20 million miles and counting…

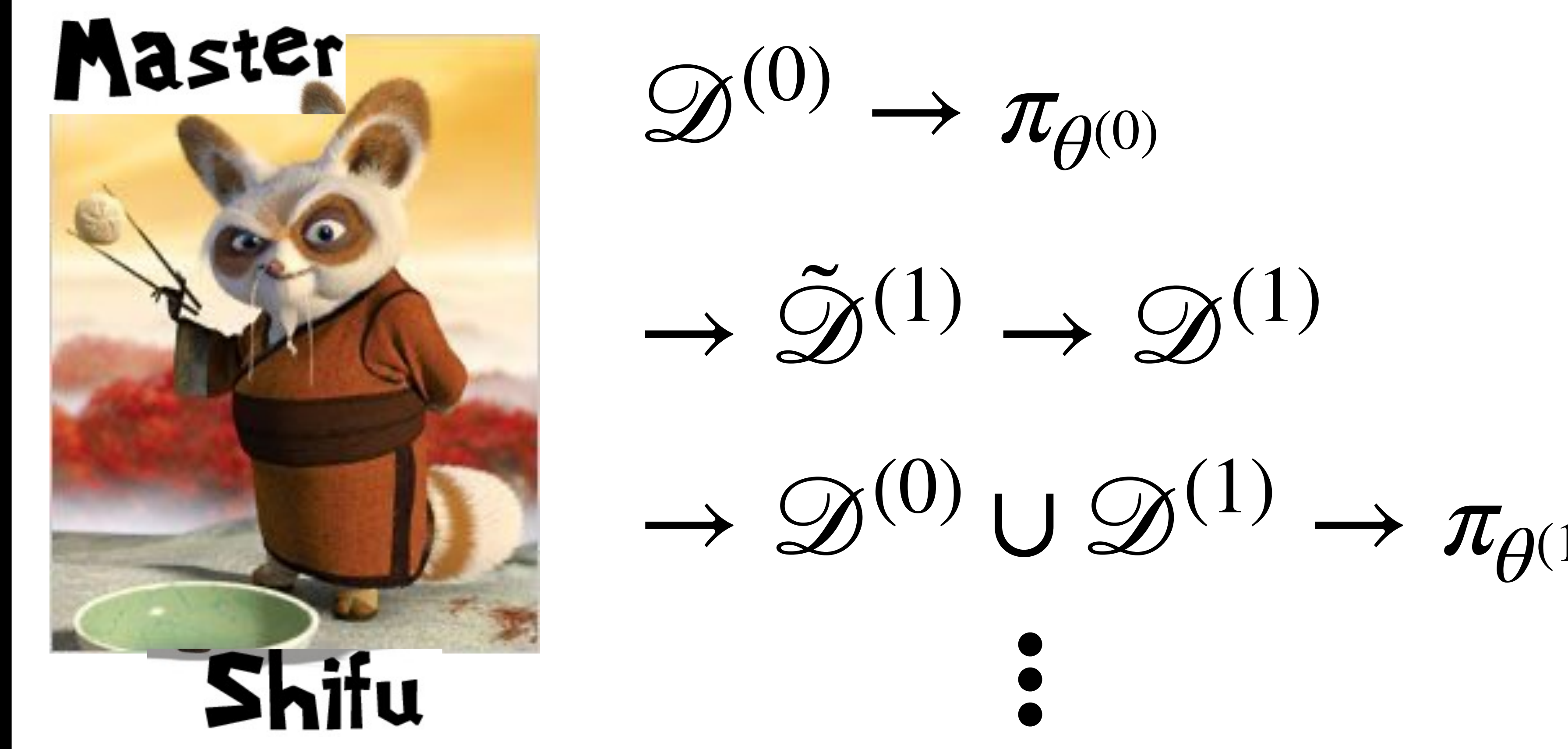
Forbes, January 2020

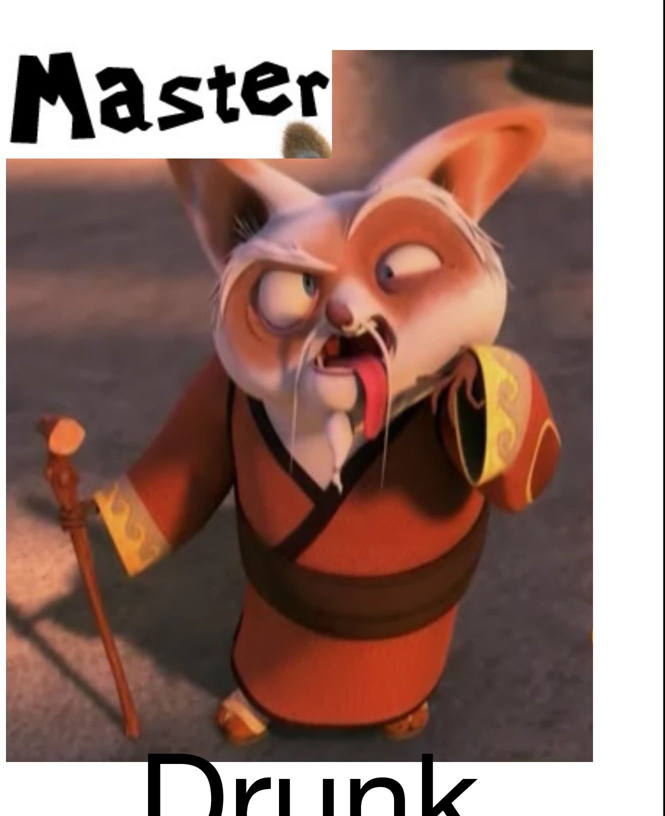
"Self-driving Uber kills Arizona woman in first fatal crash involving pedestrian", Guardian, 2018
"Uber's self-driving operator charged over fatal crash", BBC, 2018

"Ford Engineers Are Falling Asleep While Monitoring Self-Driving Cars", The Drive, 2017

Imitation Learning - Issues



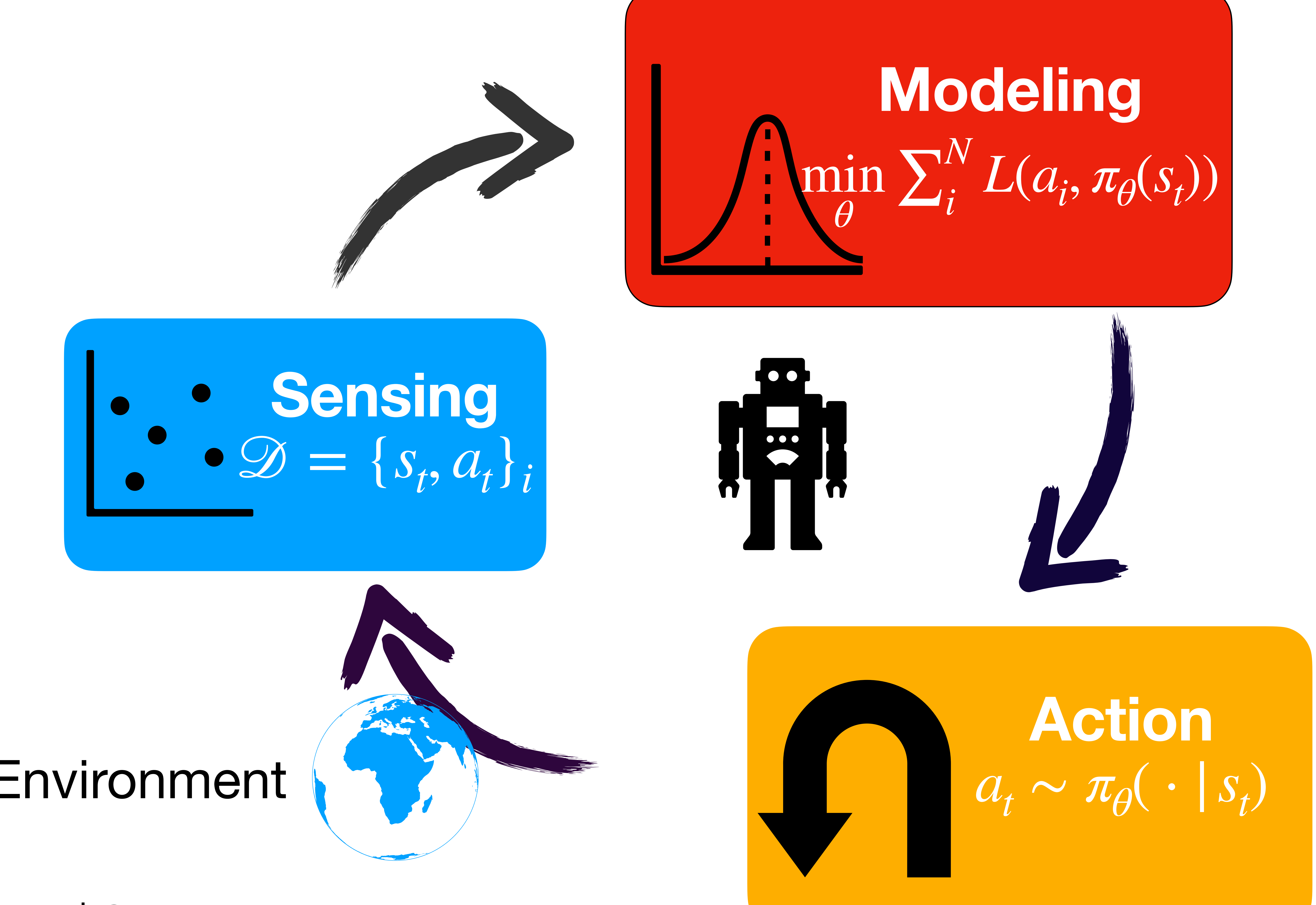




- Issue of IL-5.0
  - Shifu may make mistakes, e.g., normal human drivers, could be imperfect, e.g. road raging, panic in a collision, sleepy
  - Too expensive
    - 3,000 miles to train Nvidia autonomous vehicles
    - 20,000,000 miles of testing on the public road by Waymo
  - Some experiments are safety critical, e.g. robotics, healthcare

- Instead of imitating, we may just need to know the goal and find better methods with trial-and-error - **Reinforcement learning**

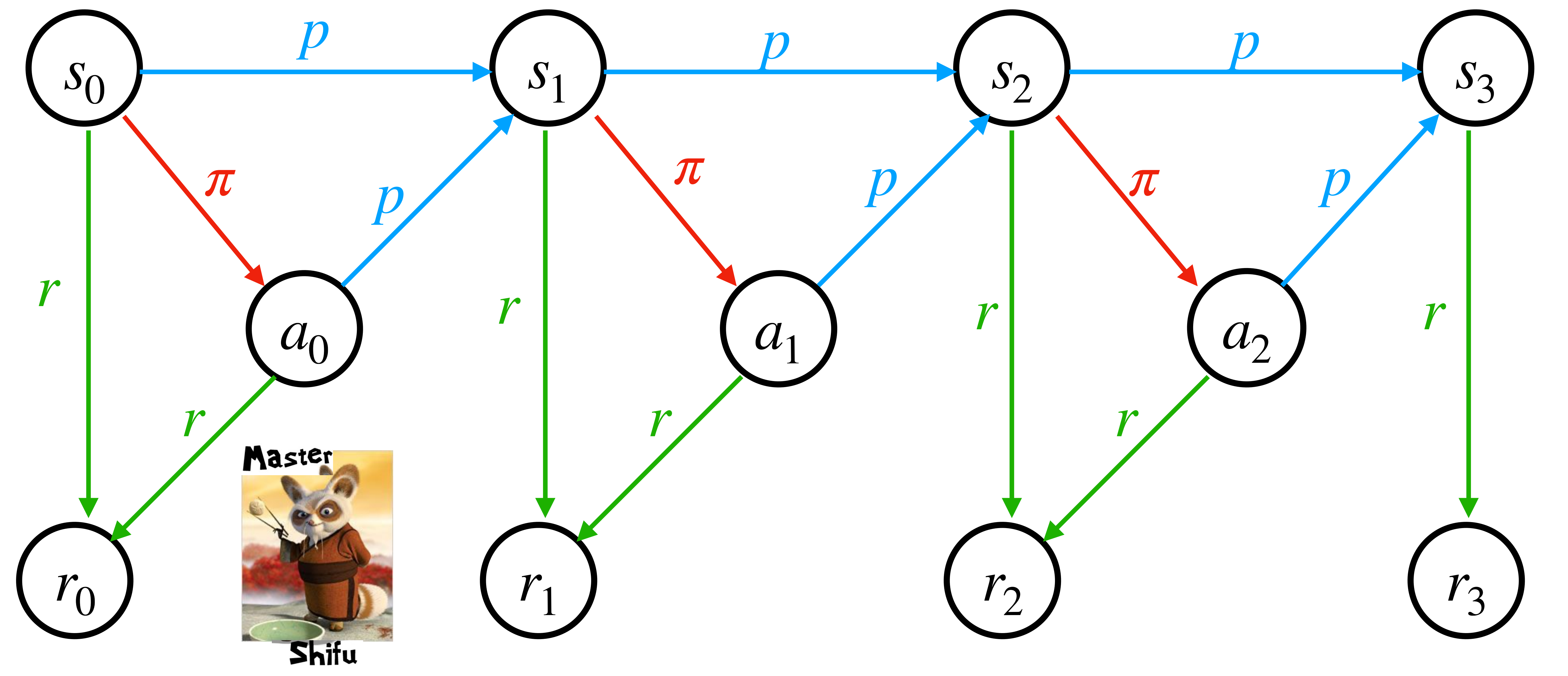
Reinforcement learning



Plan for today

- Sequential decision making: MDP, POMDP
- Imitation learning
    - Behavioral cloning, Dagger
- Model-free reinforcement learning
    - Value-based: DQN
    - Policy-based: REINFORCE, TRPO, PPO
    - Value-based+Policy-based: DDPG

Reinforcement Learning - Reward function

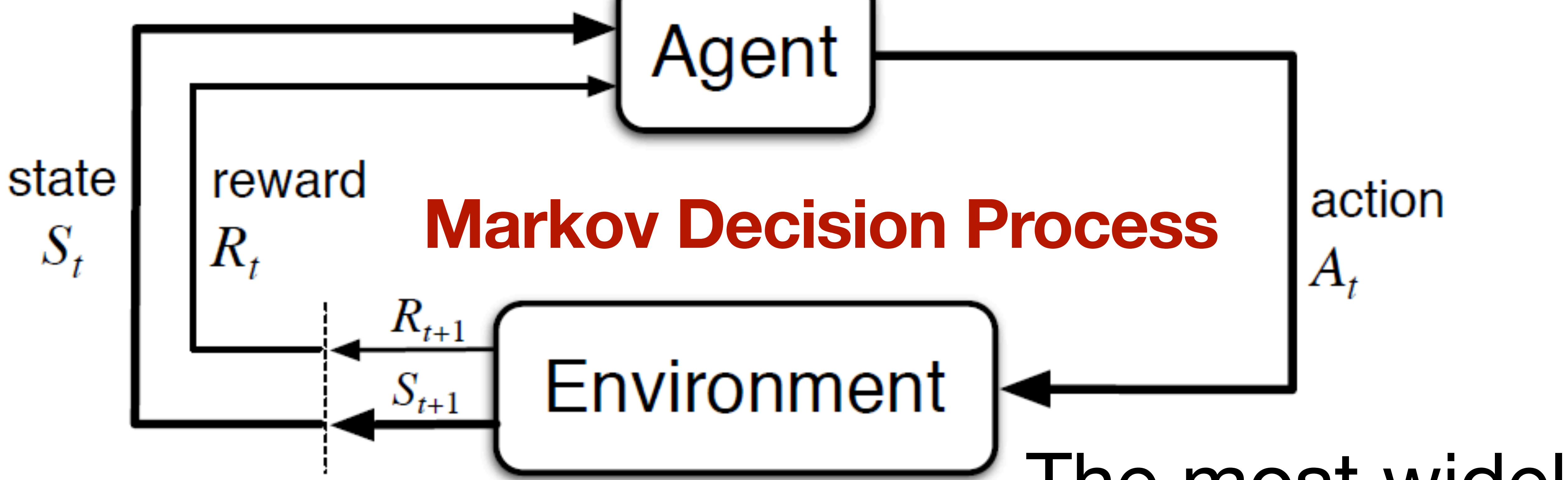


- Instead of asking for demos, we only request a single digit number r_t to indicate the level of happiness - reward.

$$s_{t+1} \sim p\left(\cdot \,|\, s_t \,,\, a_t \right)$$

$$a_t \sim \pi\left(\cdot \,|\, s_t \right)$$

$$r_t \sim r\left(\cdot \,|\, s_t \,,\, a_t \right)$$



The most widely used RL structure is MDP.

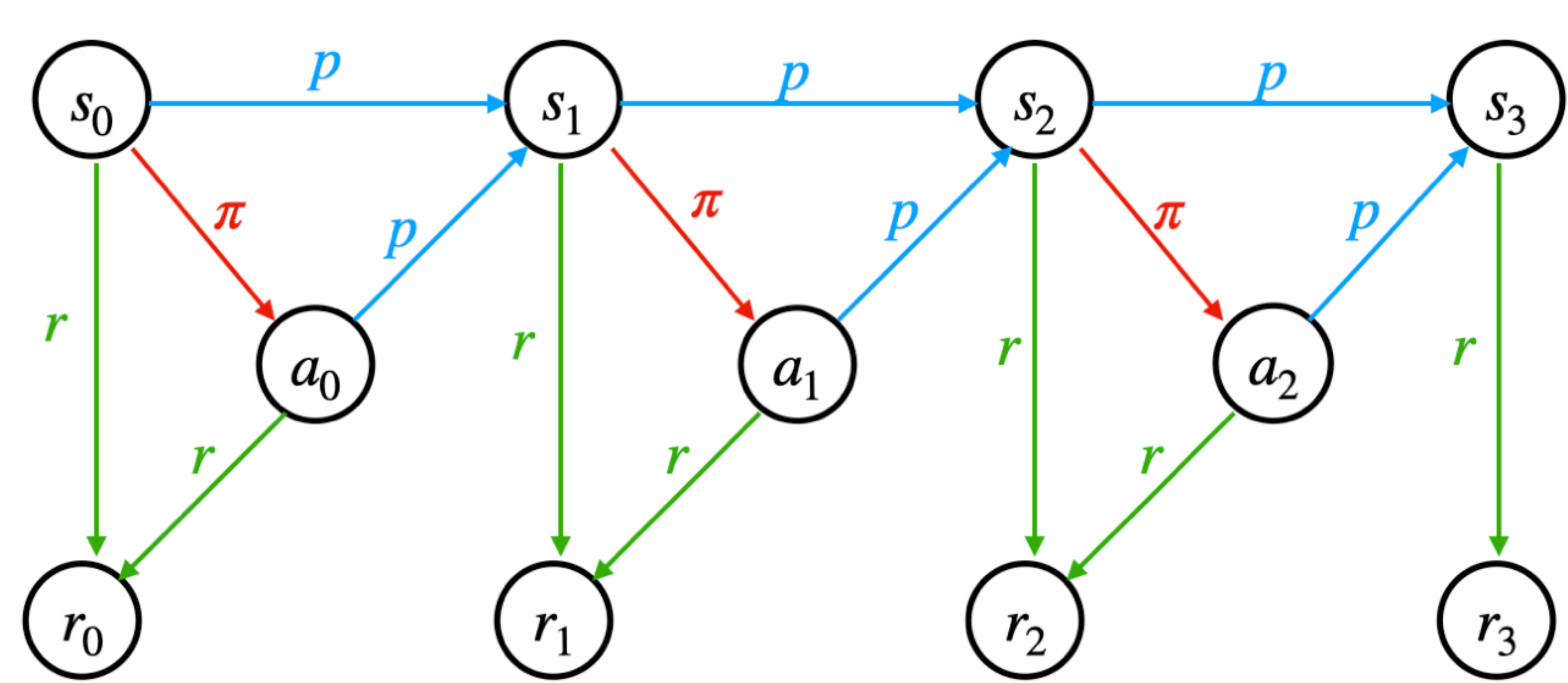
Markov Decision Process

$s_{t+1} \sim p(\cdot \mid s_t, a_t)$

$a_t \sim \pi(\cdot \mid s_t)$

$r_t \sim r(\cdot \mid s_t, a_t)$

- Mathematical formulation of the RL problem
- Markov property: Current state completely characterizes the state of the world
- Defined by: $(\mathcal{S}, \mathcal{A}, r, p)$
  - $\mathcal{S}$: set of possible states
  - $\mathcal{A}$: set of possible actions
  - r : reward function
  - p : dynamics function



- A *trajectory (rollout)* is a sequence of states and actions $\tau = (s_0, a_0, s_1, a_1, \ldots)$
  s_0 is randomly sampled from the start-state distribution $s_0 \sim \rho_0(\cdot)$

Return

- In a sequential decision making, the accumulated episodic is called *return*
- There are two common ways to define return. In the practice, people may mix these two up
  - Finite-horizon undiscounted return

$$R(\tau) = \sum_{t=0}^{T} r_t$$

  - Infinite-horizon discounted return

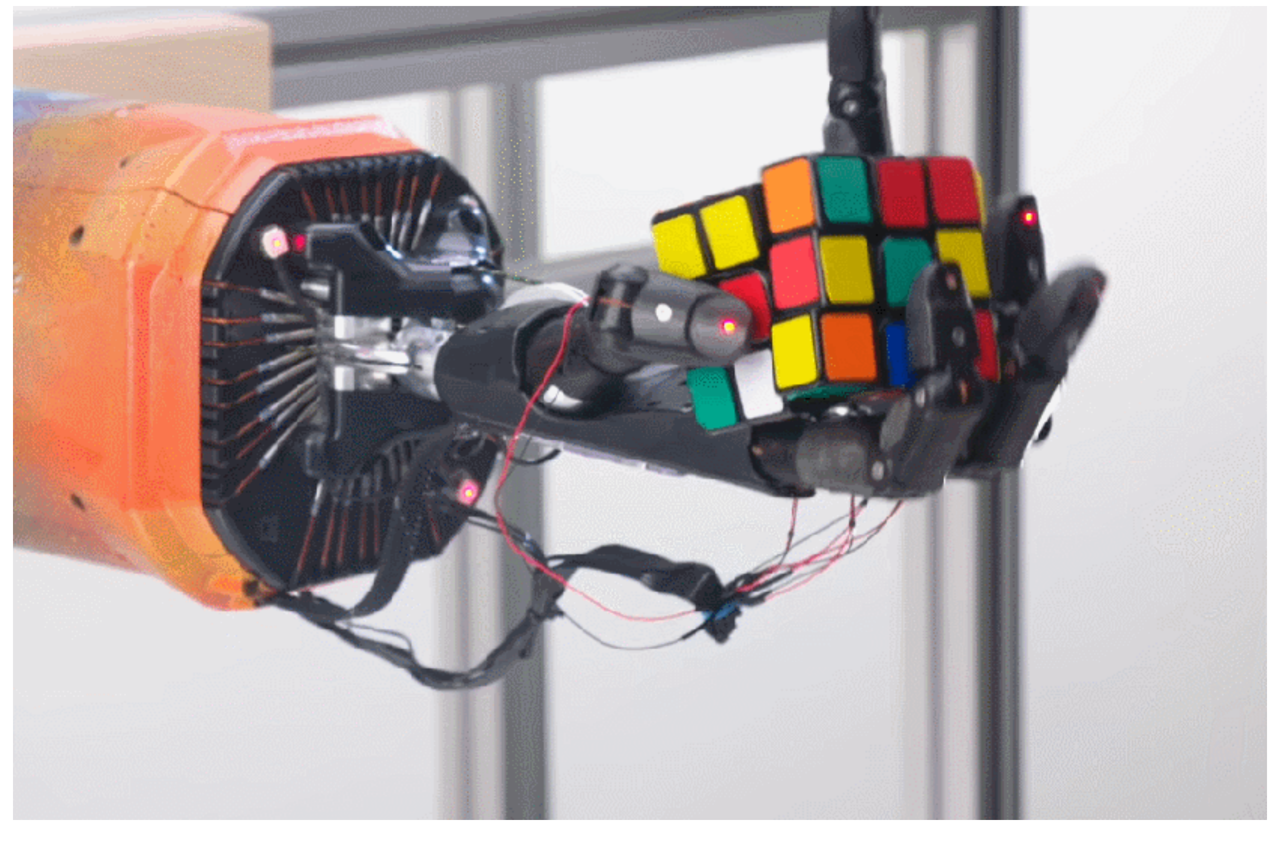
$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

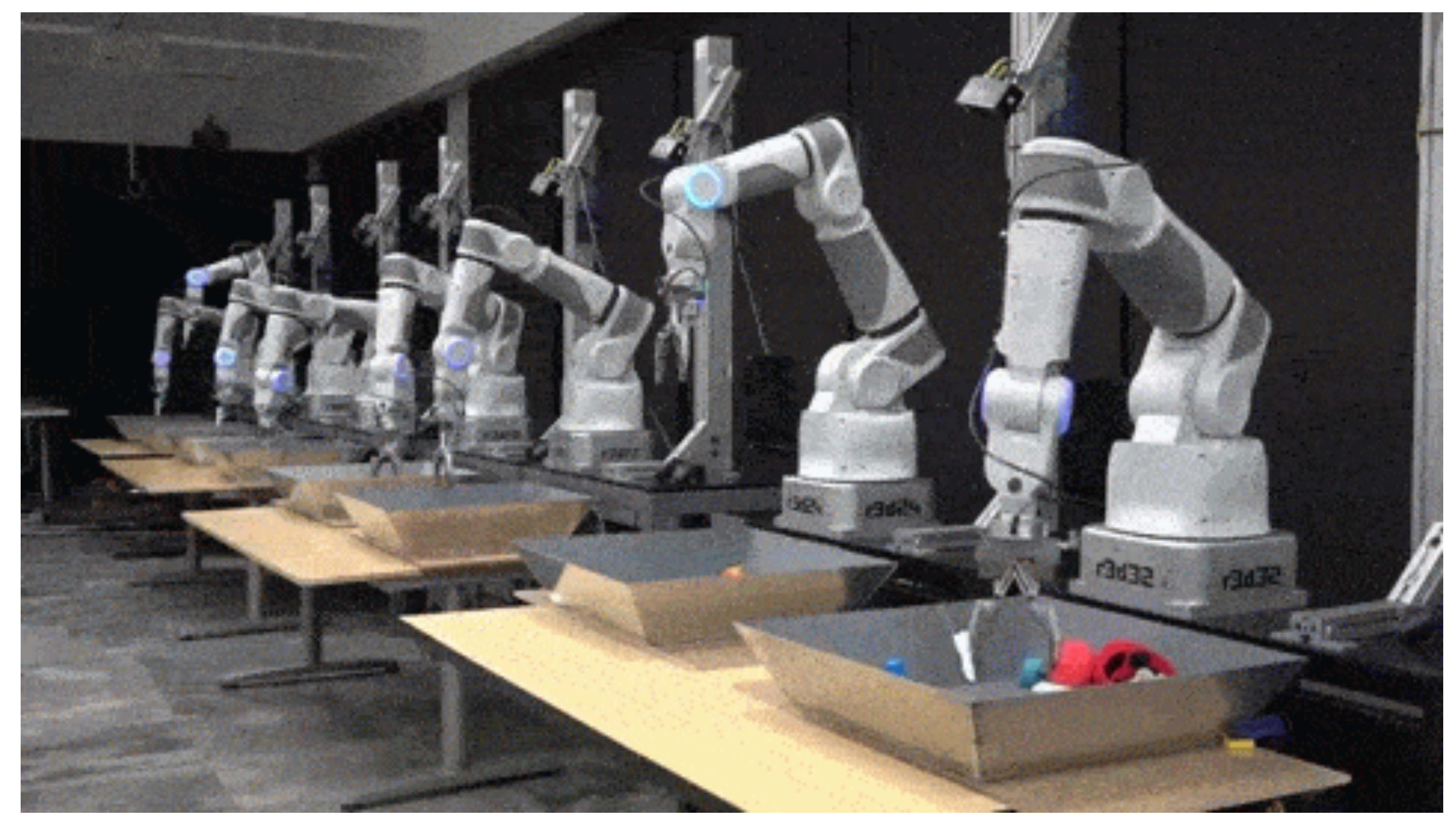
    γ : discount factor

  - Goal of MDP: given $(\mathcal{S}, \mathcal{A}, r, p)$, $\rho_0(\cdot)$, T or γ, we want

$$\pi^* = \arg\max_{\pi} \mathrm{E}_{\tau \sim \pi} \left[R(\tau) \right]$$

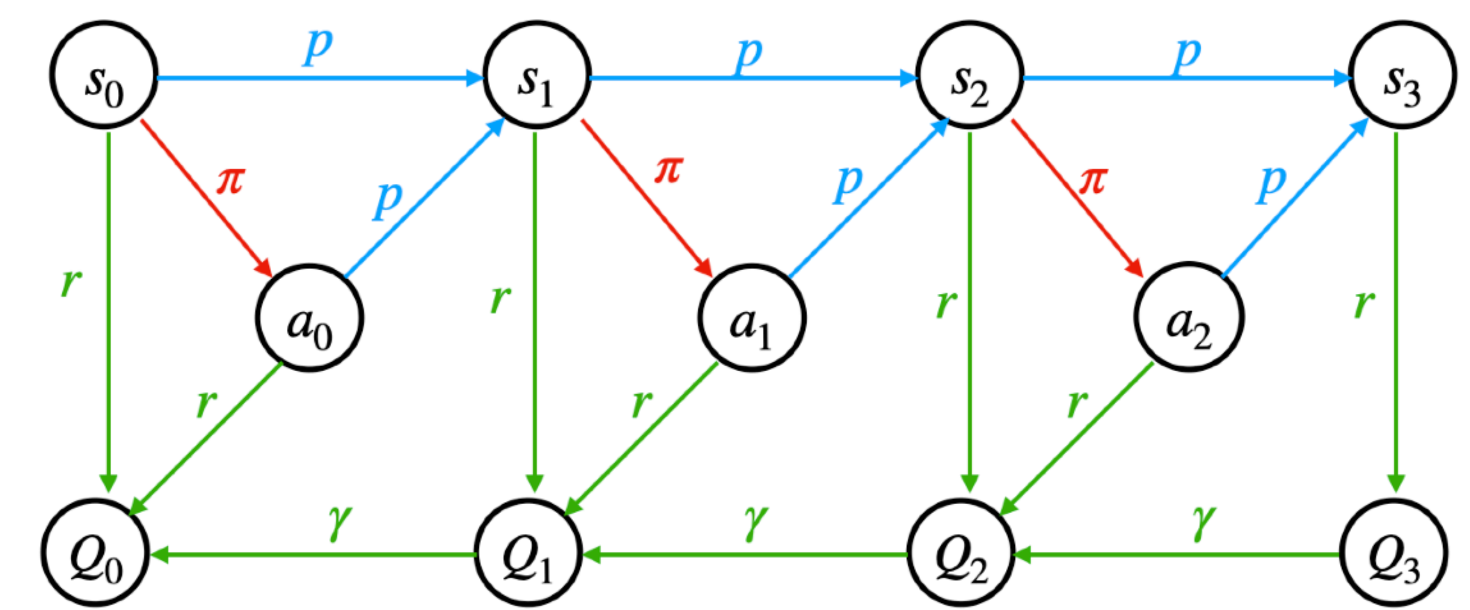
Sequential decision making





What are the $(\mathcal{S}, \mathcal{A}, r, p)$, $\rho_0(\cdot)$, $R(\tau)$, π^***?**

Q-value function



- For sequential decision making, we care the **total** reward in the sequence
- Q-value function with infinite-horizon definition

$$Q_t^\pi = Q^\pi(s_t, a_t) = \mathbb{E}[R(\tau) \mid s_t, a_t] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) \mid s_t, a_t]$$

- Calculate backward:

$$Q_3 = r(s_3), \quad Q_2 = r(s_2, a_2) + \gamma Q_3, \quad Q_1 = r(s_1, a_1) + \gamma Q_2, \ldots$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{a_{t+1} \sim \pi}[r(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

- Optimal policy π^* optimize $Q^\pi(s_t, a_t)$, or *Bellman Equation (dynamic programming)*

$$Q^{\pi^*}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^{\pi^*}(s_{t+1}, a_{t+1})]$$

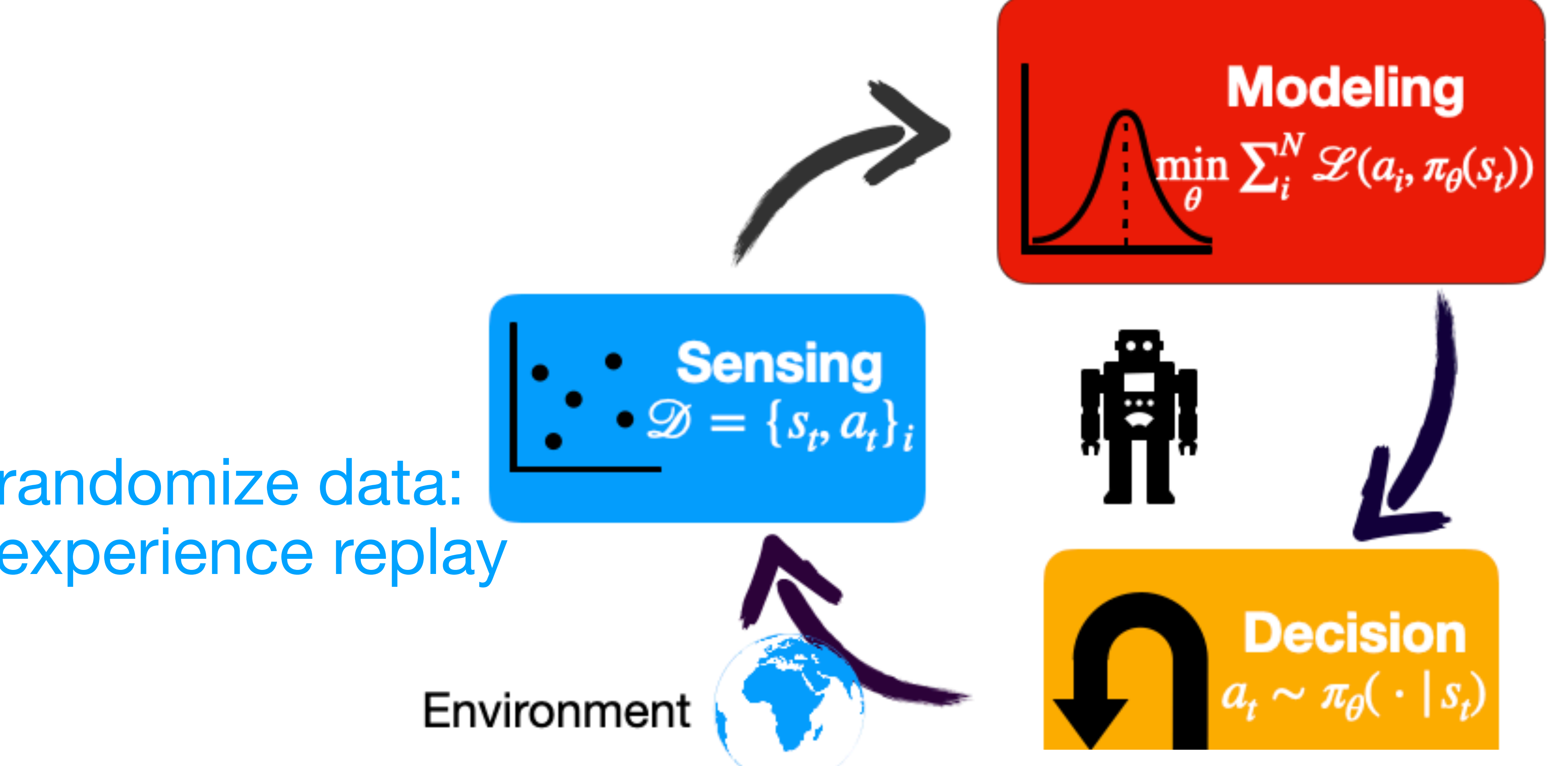
“Bellman backup” or just “backup” or “target" comes up quite frequently in the RL literature, which is the right-hand side of the Bellman equation: the reward-plus-next-value.

Two ways to compute the optimal policy

- Parameterize the policy
  - Gradient ascent
- $J(\theta, \mathscr{D}_{\pi_\theta}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|\, \pi_\theta\right]$
- $\theta^* = \arg\max_\theta J(\theta, \mathscr{D}_{\pi_\theta})$
- $\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\theta)\,|_{\theta=\theta_i}, \theta_i \to \theta^*$
- $a_t \sim \pi_{\theta_i}(\cdot \,|\, s_t)$

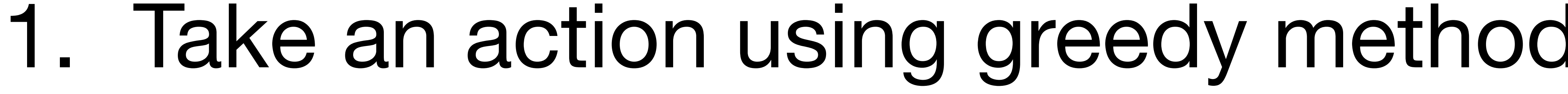
- Parameterize the value function Q
  - Dynamic programming
- $Q_{\phi^*}^{\pi^*}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi^*}^{\pi^*}(s_{t+1}, a_{t+1})]$
- $e_\phi^\pi = Q_\phi^\pi(s_t, a_t) - \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_\phi^\pi(s_{t+1}, a_{t+1})]$
- $L(\phi, \pi) = \mathbb{E}\left[\frac{1}{2} e_\phi^{\pi 2}\right], (\phi^*, \pi^*) = \arg\min_{\phi,\pi} L(\phi, \pi)$
- With the“greedy method”, i.e., $\pi(a_t \,|\, s_t) = \max_a Q_\phi(s_t, a)$ ϕ of Q then can influence π.
- $\phi_{i+1} = \phi_i - \alpha \nabla_\phi L(\phi)\,|_{\phi=\phi_i}, \phi_i \to \phi^*, \pi_i \to \pi^*$

DQN-1.0 algorithm



randomize policy: ε-greedy

One point iteration



1. Take an action using greedy method

$$a_t = \max_a Q_{\phi_i}(s_t, a)$$

   and observe $\left(s_t, a_t, s_{t+1}, r_i\right)$

2. Calculate Bellman backup

$$y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1})$$

3. Update Q function

$$\phi \leftarrow \phi - \alpha \sum_t \left(\nabla_\phi Q_\phi(s_t, a_t) \right) \left(Q_\phi(s_t, a_t) - y_t \right)$$

   where α is the learning rate

Issue of DQN-1.0, the optimization may be trapped in a vicious circle:

bad policy -> bad training data -> train a even worse policy (a core problem of RL)

Different from IL, training data here is not i.i.d.:

- Training data are decided by a policy strictly decided by previous training data
- (s_t, a_t) pairs in a trajectories are dependent

Two ways to cut the chain:

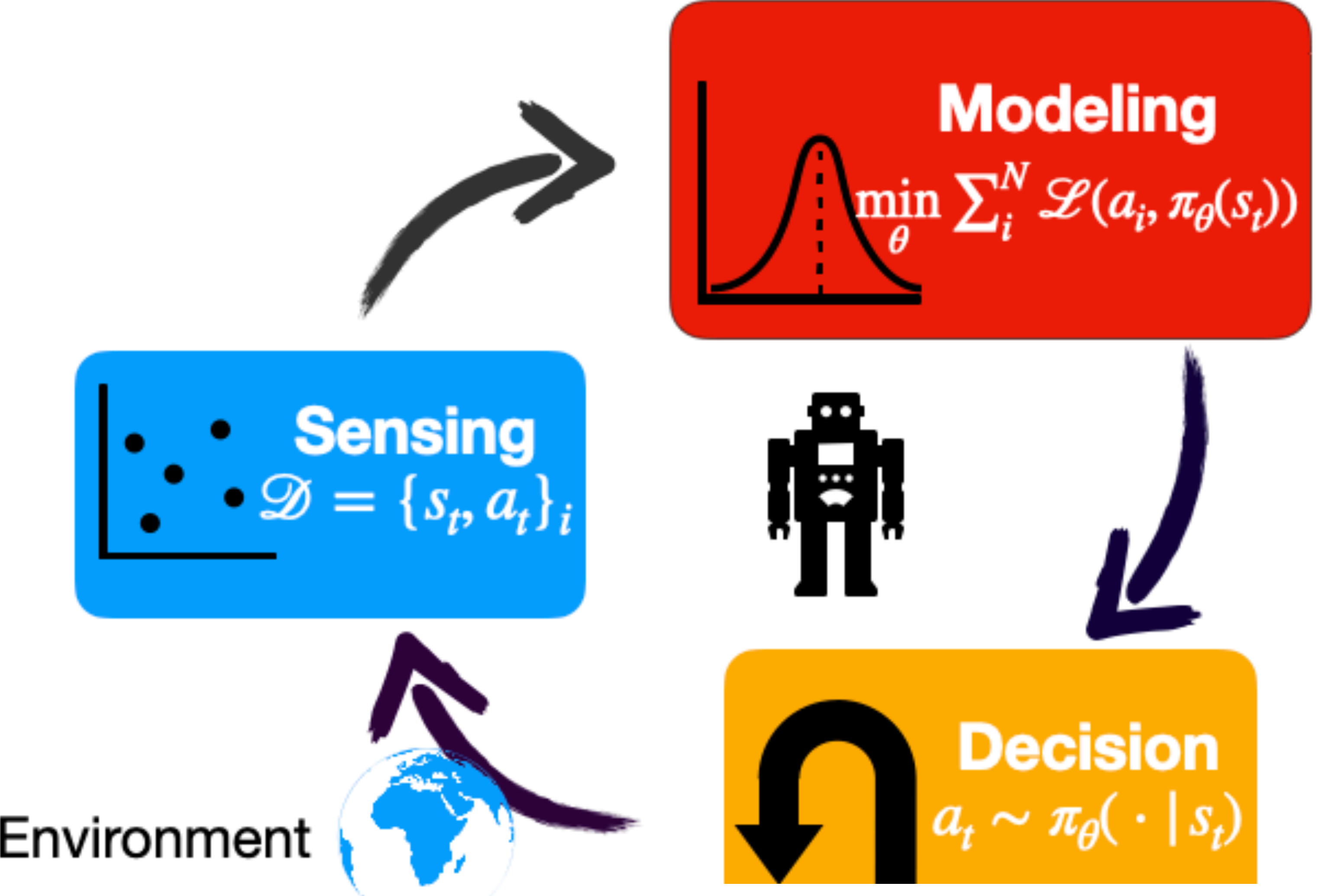
=> randomize policy: ε -greedy

=> randomize data: experience replay

DQN-2.0 algorithm

Randomize actions

1. Take an action using ε-greedy method:
   $a_t = \max_a Q_{\phi_i}(s_t, a)$ with probability $1 - \varepsilon$,
   otherwise, choose a random action
   and observe $\left(s_t, a_t, s_{t+1}, r_i\right)$
2. Calculate Bellman backup
   $y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1})$
3. Update Q function
   $\phi \leftarrow \phi - \alpha \sum_t \left(\nabla_\phi Q_\phi(s_t, a_t) \right) \left(Q_\phi(s_t, a_t) - y_t \right)$
   where α is the learning rate



randomize policy: ε-greedy

Issue of DQN-1.0, the optimization may be trapped in a vicious circle:

bad policy -> bad training data -> train a even worse policy (a core problem of RL)

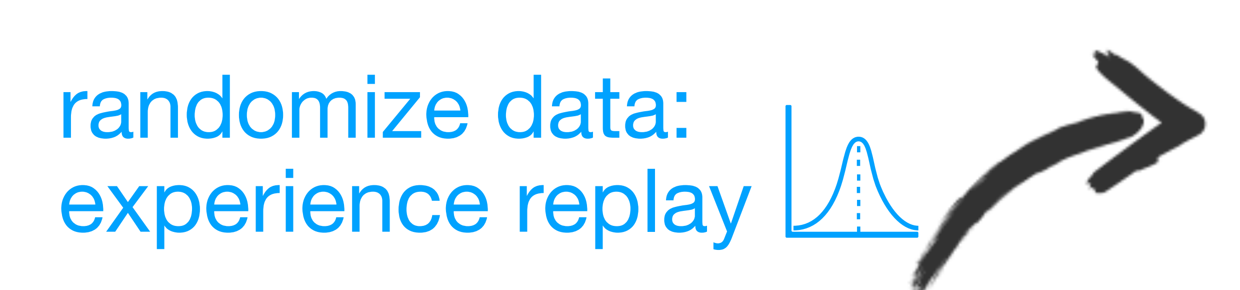
Different from IL, training data here is not i.i.d.:

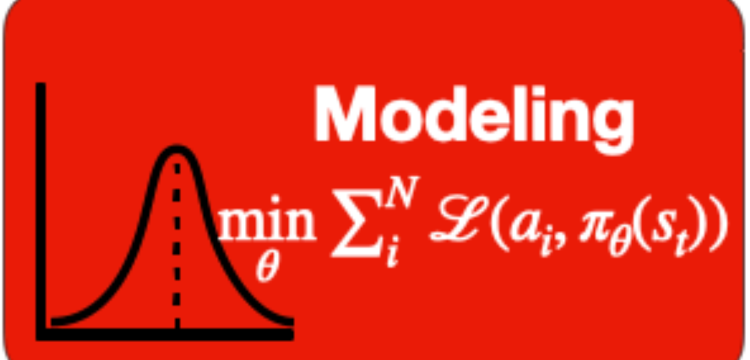
- Training data are decided by a policy strictly decided by previous training data
- (s_t, a_t) pairs in a trajectories are dependent

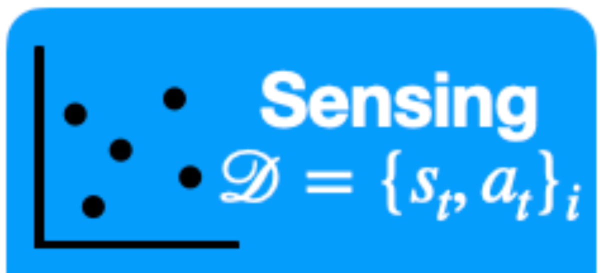
Two ways to cut the chain:

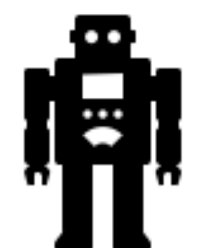
=> randomize policy: ε -greedy
=> randomize data: experience replay

DQN-3.0 algorithm









randomize policy: ε-greedy

Randomize actions and training data

1. Take the ε-greedy method:

   $a_t = \max_a Q_{\phi_i}(s_t, a)$ with probability $1 - \varepsilon$, otherwise, choose a random action

   observe a dataset $\{(s_t, a_t, s_{t+1}, r_t)\}$ and add it to $\mathscr{D}$

   1. Randomly sample a mini batch from $\mathscr{D}$
   2. Calculate Bellman backup for this batch

      $y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1})$

      1. Iteratively calculate ϕ starting from ϕ_i

         $\phi \leftarrow \phi - \alpha \sum_t \left(\nabla_\phi Q_\phi(s_t, a_t) \right) \left(Q_\phi(s_t, a_t) - y_t \right)$

   3. Update Q function:

      Way 1 direction update: $\phi_{i+1} = \phi$

      Way 2 moving average: $\phi_{i+1} = \rho\phi_i + (1 - \rho)\phi$, e.g. $\rho = 0.999$

Issue of DQN-1.0, the optimization may be trapped in a vicious circle:

bad policy -> bad training data -> train a even worse policy (a core problem of RL)

Different from IL, training data here is not i.i.d.:

- Training data are decided by a policy strictly decided by previous training data
- (s_t, a_t) pairs in a trajectories are dependent

Two ways to cut the chain:

=> randomize policy: ε -greedy

=> randomize data: experience replay

Also called low pass filter/Polyak averaging in literature

Game	DQN (% of human level)
Video Pinball	2539%
Boxing	1707%
Breakout	1327%
Star Gunner	598%
Robotank	508%
Atlantis	449%
Crazy Climber	419%
Gopher	400%
Demon Attack	294%
Name This Game	278%
Krull	277%
Assault	246%
Road Runner	232%
Kangaroo	224%
James Bond	145%
Tennis	143%
Pong	132%
Space Invaders	121%
Beam Rider	119%
Tutankham	112%
Kung-Fu Master	102%
Freeway	102%
Time Pilot	100%
Enduro	97%
Fishing Derby	93%
Up and Down	92%
Ice Hockey	79%
Q*bert	78%
H.E.R.O.	76%
Asterix	69%
Battle Zone	67%
Wizard of Wor	67%
Chopper Command	64%
Centipede	62%
Bank Heist	57%
River Raid	57%
Zaxxon	54%
Amidar	43%
Alien	42%
Venture	32%
Seaquest	25%
Double Dunk	17%
Bowling	14%
Ms. Pac-Man	13%
Asteroids	7%
Frostbite	6%
Gravitar	5%
Private Eye	2%
Montezuma's Revenge	0%

At human-level or above

Below human-level

DQN

Best linear learner

0 100 200 300 400 500 600 1,000 4,500%

Does it work?

Most of the time!

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).

Code and performance

```
1  model = DQN(env.observation_space.shape[0], env.action_space.n)
2  replay_buffer = ReplayBuffer(1000)
3  state = env.reset()
4  for frame_idx in range(1, num_frames + 1):
5      epsilon = epsilon_by_frame(frame_idx)
6      action = model.act(state, epsilon)
7      next_state, reward, done, _ = env.step(action)
8      replay_buffer.push(state, action, reward, next_state, done)
9      state = next_state
10     episode_reward += reward
11     if done:
12         state = env.reset()
13         all_rewards.append(episode_reward)
14         episode_reward = 0
15     if len(replay_buffer) > batch_size:
16         loss = compute_td_loss(batch_size)
17         losses.append(loss.data[0])
```

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).

Q-network Architecture

- $Q(s, a; \theta)$: neural network with weights θ
- Current state: 84x84x4 stack of last 4 frames
- (Preprocessing: RGB->grayscale, avg of two consecutive images, downsampling, and cropping)
- Last fully connected (FC) layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_t^{(1)})$, $Q(s_t, a_t^{(2)})$, $Q(s_t, a_t^{(3)})$, $Q(s_t, a_t^{(4)})$

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Two ways to compute the optimal policy

- Parameterize the policy
    - Gradient ascent
- $J(\theta, \mathcal{D}_{\pi_\theta}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|\, \pi_\theta\right]$
- $\theta^* = \arg\max_\theta J(\theta, \mathcal{D}_{\pi_\theta})$
- $\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\theta)\,|_{\theta=\theta_i}, \theta_i \to \theta^*$
- $a_t \sim \pi_{\theta_i}(\cdot \,|\, s_t)$

- Parameterize the value function Q
    - Dynamic programming
- $Q_{\phi^*}^{\pi^*}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi^*}^{\pi^*}(s_{t+1}, a_{t+1})]$
- $e_\phi^\pi = Q_\phi^\pi(s_t, a_t) - \mathbb{E}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_\phi^\pi(s_{t+1}, a_{t+1})]$
- $L(\phi, \pi) = \mathbb{E}\left[\frac{1}{2} e_\phi^{\pi 2}\right], (\phi^*, \pi^*) = \arg\min_{\phi,\pi} L(\phi, \pi)$
- With the"greedy method", i.e., $\pi(a_t \,|\, s_t) = \max_a Q_\phi(s_t, a)$ ϕ of Q then can influence π.
- $\phi_{i+1} = \phi_i - \alpha \nabla_\phi L(\phi)\,|_{\phi=\phi_i}, \phi_i \to \phi^*, \pi_i \to \pi^*$

Policy Gradients

- Issue of with DQN algorithms?
  - The Q-function can be very complicated!
  - Example: active safety function of a car has a very high-dimensional state => hard to learn exact value of every state (driving scenes) - action pair
  - But the policy can be much simpler: just brake or release
- Can we learn a policy directly, e.g. finding the best policy from a collection of policies?
  - We will first learn one of the most widely used algorithms: REINFORCE

REINFORCE algorithm - objective function

- Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

where $r(\tau)$ is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

REINFORCE-1.0

- Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_{\tau} r(\tau)p(\tau;\theta)d\tau$
- Now let's differentiate this: $\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau;\theta)d\tau$
- Intractable! Gradient of an expectation is problematic when p depends on θ
- However, we can use a nice trick:

$$\nabla_{\theta} p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_{\theta} p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_{\theta} \log p(\tau;\theta)$$

- Substitute it back:

$$\nabla_{\theta} J(\theta) = \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau;\theta))p(\tau;\theta)d\tau = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau) \nabla_{\theta} \log p(\tau;\theta)]$$

which can be estimated with Monte Carlo sampling

Issue: we may not know $p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t)\pi_{\theta}(a_t \mid s_t)$

REINFORCE-2.0: remove transition probability

- Can we compute those quantities without knowing the transition probabilities?
- We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t)$
- Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)$
- And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
- Doesn't depend on transition probabilities!
- Therefore when sampling one trajectory τ, we can estimate $J(\theta)$ with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Intuition

- Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\, \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
- **Interpretation:**
  - If $r(\tau)$ is high, push up the probabilities of the actions seen
  - If $r(\tau)$ is low, push down the probabilities of the actions seen

Issue of REINFORCE-2.0: Might seem simplistic to judge an action with the reward of the whole trajectory. It is like a bench player is on the court for only 1 min, but is evaluated based on the final game result.

Mathematically, this will lead a big variance (ambiguity) in evaluation and further decision.

REINFORCE-3.0: better credit assignment

- Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$
- **One idea**: use rewards only after the action a_t is applied

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- **Even better**: Use discount factor γ to consider the delayed effects, i.e., give higher weights to the reward just after a_t is applied

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Issue 1: what if the system has delay - we will discuss it later
Issue 2: how to decide whether $r_{t'}$ is high or low

REINFORCE-3.0: add baseline of a reward

- Problem: The raw value of the reward isn't necessarily meaningful. Ideally, assign a positive credit when the reward is relatively high; negative when the reward is relatively low.

- We need a baseline function: $b(s_t)$

- Concretely, now estimator is:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

- A simple baseline: constant moving average of rewards experienced so far from all trajectories

- This version is usually called "**Vanilla REINFORCE**"

Actor-Critic Algorithm: a even better baseline

- If this action was better than the **expected (averaged) value of what we should get from that state**, then we like it.
- These can be done with the values functions
  - action-value function: $Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \mid s_t, a_t\right]$
  - state-value function: $V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \mid s_t\right]$
- Intuitively, we are happy with an action at state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.
- Using this, we get the estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} (Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)) \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)$

Actor-Critic Algorithm

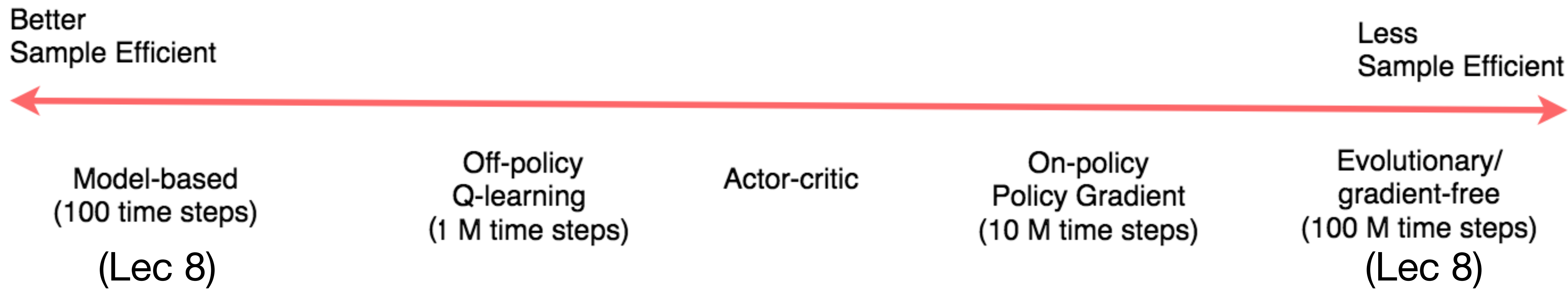
- **Problem**: we don't know Q and V. Can we learn them?
- **Yes**, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).
  - The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
  - Can also incorporate Q-learning tricks e.g. experience replay
  - Can use data generated by previous policies - more efficient
  - Define by **the advantage** function how much an action was better than expected

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

On-policy vs off-policy

- Policy optimization is almost always performed **on-policy**, which means that each update only uses data collected while acting according to the most recent version of the **policy**. The historical data collected with very old policy is not used. They can be used with both continuous and discrete states. Using gradient, they converge to a local minima of $J(\theta)$

- Q-learning, e.g., DQN, is almost always performed off-policy, which means that each update can use data collected during the whole training history, regardless of what policy the agent was choosing to explore the environment. Therefore, it is more sampling efficient. No guarantee of convergence.



https://jonathan-hui.medium.com/rl-model-based-reinforcement-learning-3c2b6f0aa323

Plan for today

- Sequential decision making: MDP, POMDP
- Imitation learning
  - Behavioral cloning, Dagger
- Model-free reinforcement learning
  - Value-based: DQN
  - Policy-based: REINFORCE
  - Value-policy-based: Actor-Critic

Worth reading

- OpenAI spinning up for Deep RL
    - https://spinningup.openai.com/en/latest/index.html
- CS 285 UC Berkeley: Deep Reinforcement Learning
    - http://rail.eecs.berkeley.edu/deeprlcourse/