Mahindra University
Global Thinkers. Engaged Leaders.

Reinforcement Learning and Autonomous Systems (AI 4102)

Lecture 4 (23/08/2023)
Lecture 5 (28/08/2023)

Instructor: Gourav Saha

Lecture Content

- The Multi-Armed Bandit* (MAB) setup.
- A Fundamental Tradeoff.
- Mathematical Notations and Concepts.
- Policies for Multi-Armed Bandit

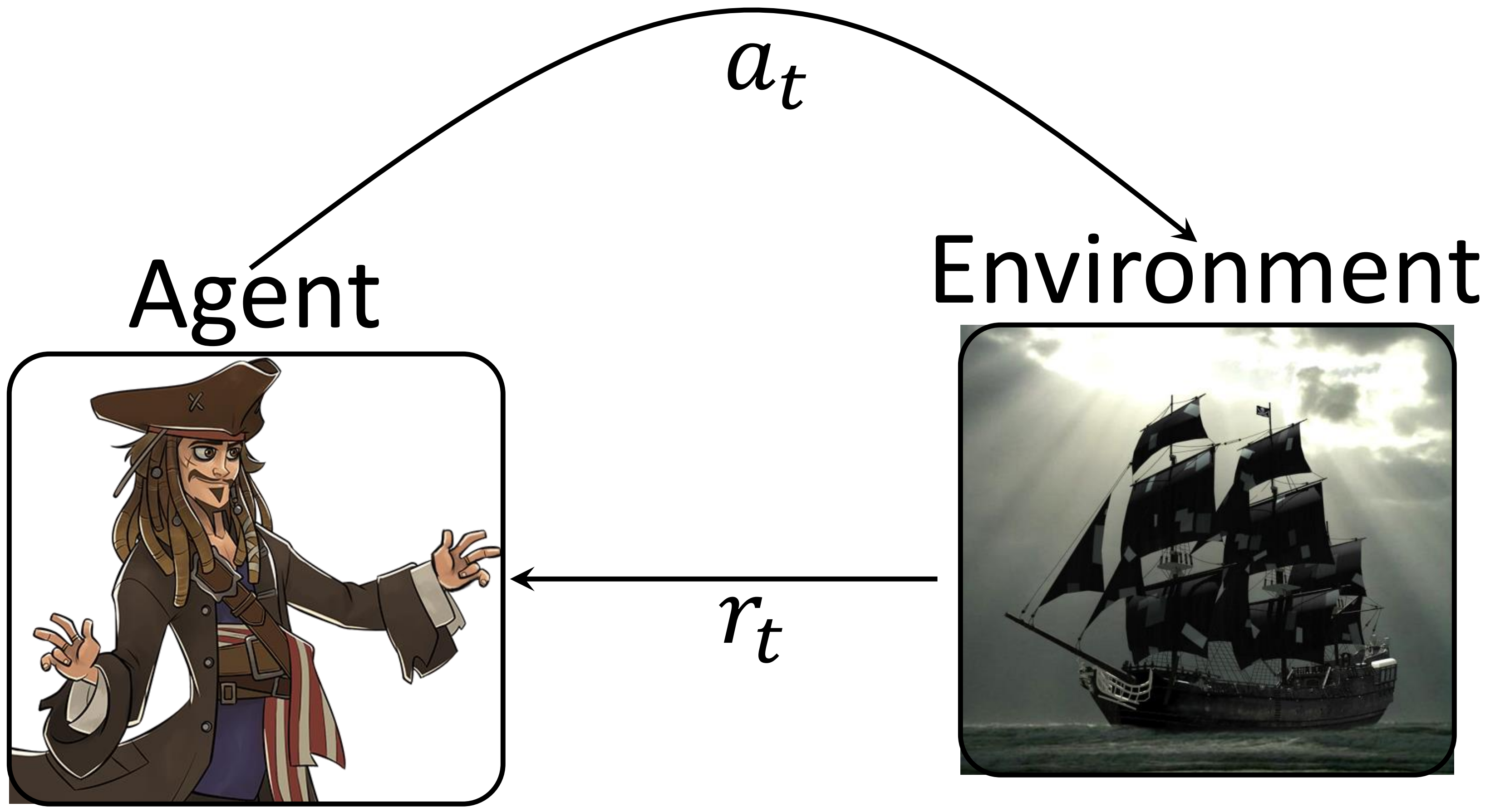
*PLEASE NOTE: Multi-Armed Bandit by default means the "simple Multi-Armed Bandit" and NOT "contextual Multi-Armed Bandit".

Lecture Content

- The Multi-Armed Bandit* (MAB) setup.
  - Applications of Multi-Armed Bandits.
- A Fundamental Tradeoff.
- Mathematical Notations and Concepts.
- Policies for Multi-Armed Bandit

*PLEASE NOTE: Multi-Armed Bandit by default means the "simple Multi-Armed Bandit" and NOT "contextual Multi-Armed Bandit".

Multi-Armed Bandit Setup



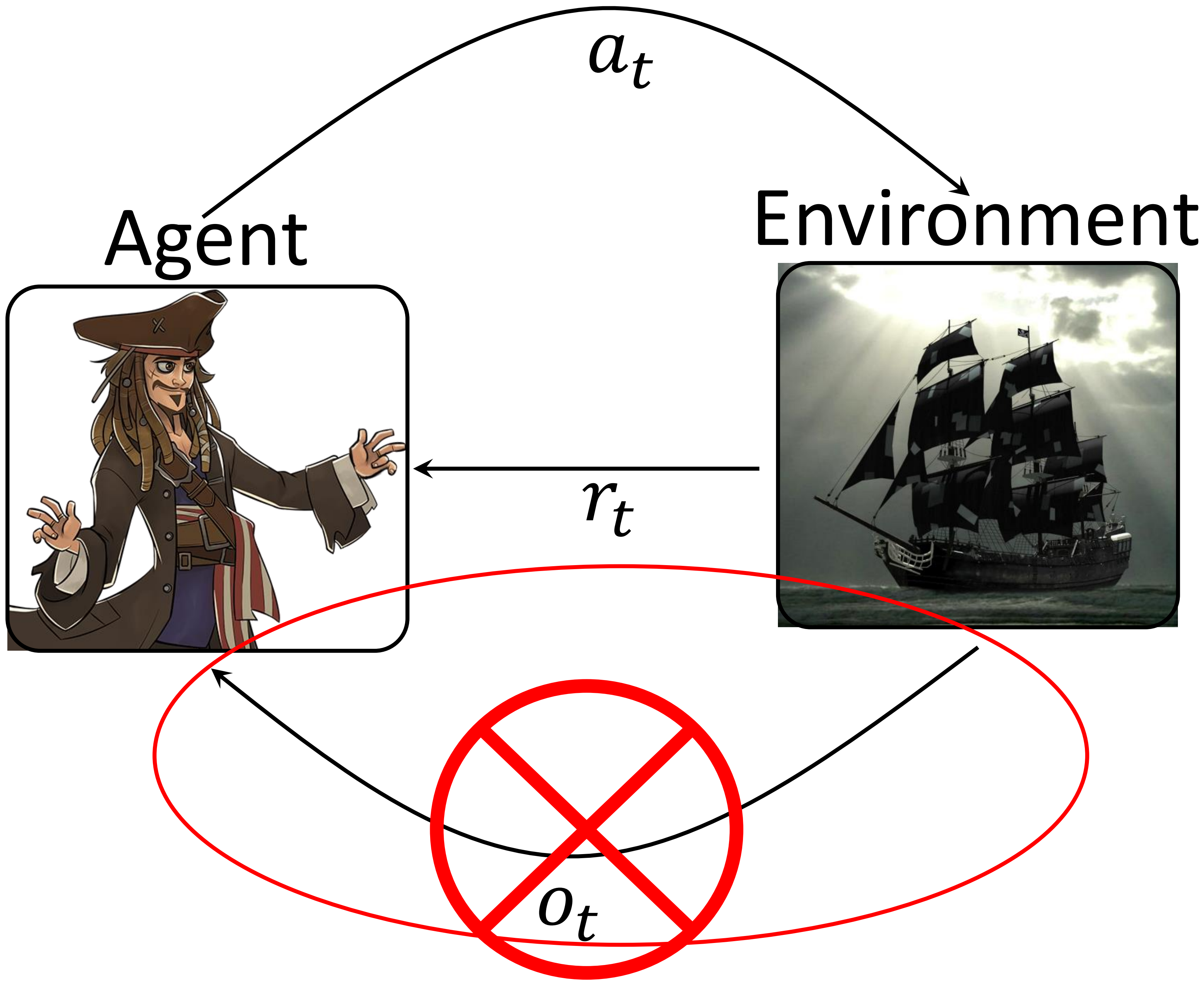
Multi-Armed Bandit (MAB) setup is as follows:

- The set of available **actions** is $\mathcal{A} = \{1, 2, \cdots, A\}$.
- Action a has the reward distribution P_a. The agent **does not know** P_a for any action a.
- In time slot t:
  Step 1: The agent decides its action a_t.

  Step 2: The agent receives a reward $r_t \sim P_{a_t}$ that is generated by the environment.
- The goal of the agent is to maximize the expected total reward,

$$\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$$

Multi-Armed Bandit Setup



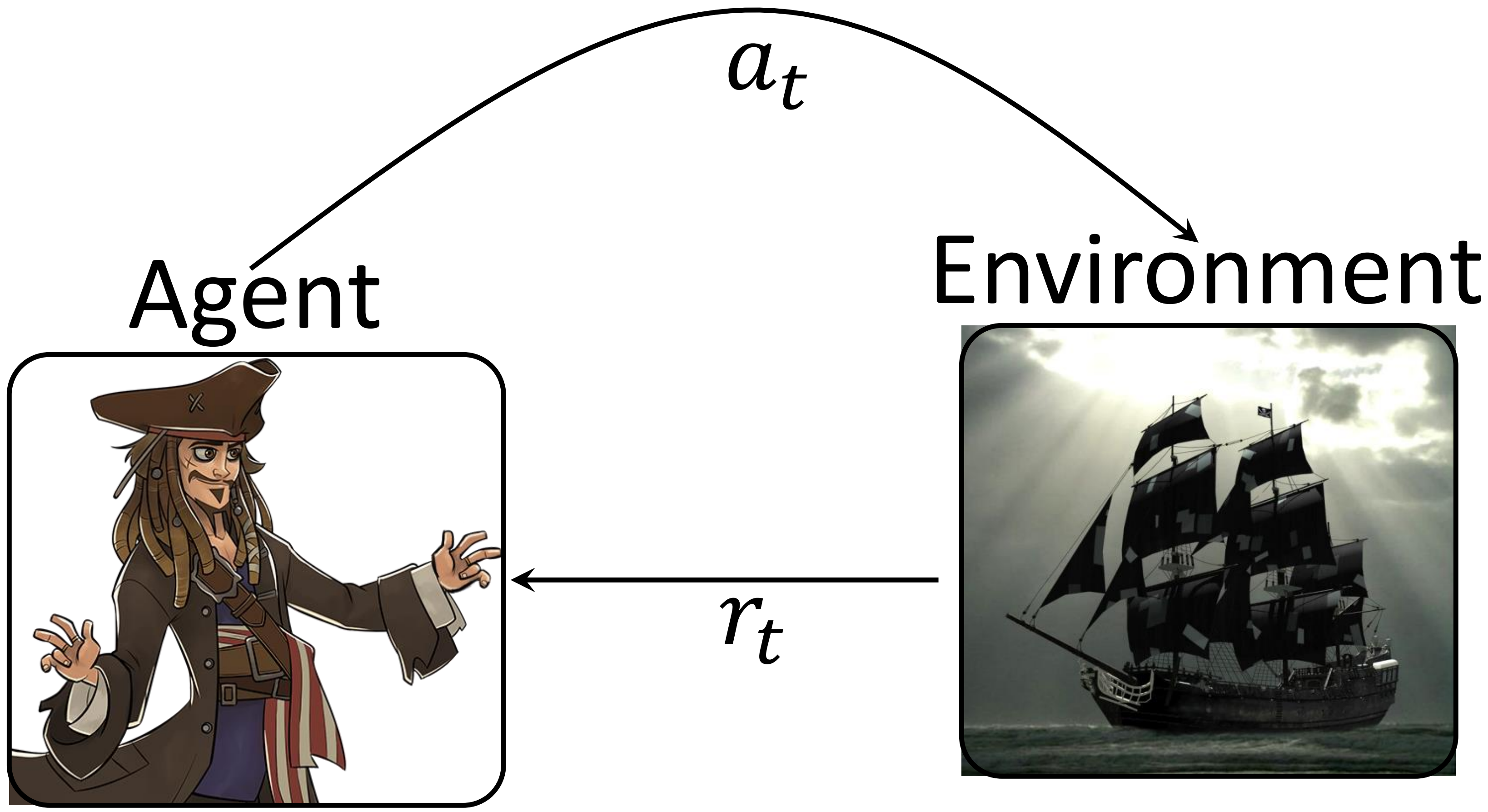
There is no observation in MAB setup.

Multi-Armed Bandit (MAB) setup is as follows:

- The set of available **actions** is $\mathcal{A} = \{1,2,\cdots,A\}$.
- Action a has the reward distribution P_a. The agent **does not know** P_a for any action a.
- In time slot t:
  Step 1: The agent decides its action a_t.
  Step 2: The agent receives a reward $r_t \sim P_{a_t}$ that is generated by the environment.
- The goal of the agent is to maximize the expected total reward,

$$\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$$

Multi-Armed Bandit Setup



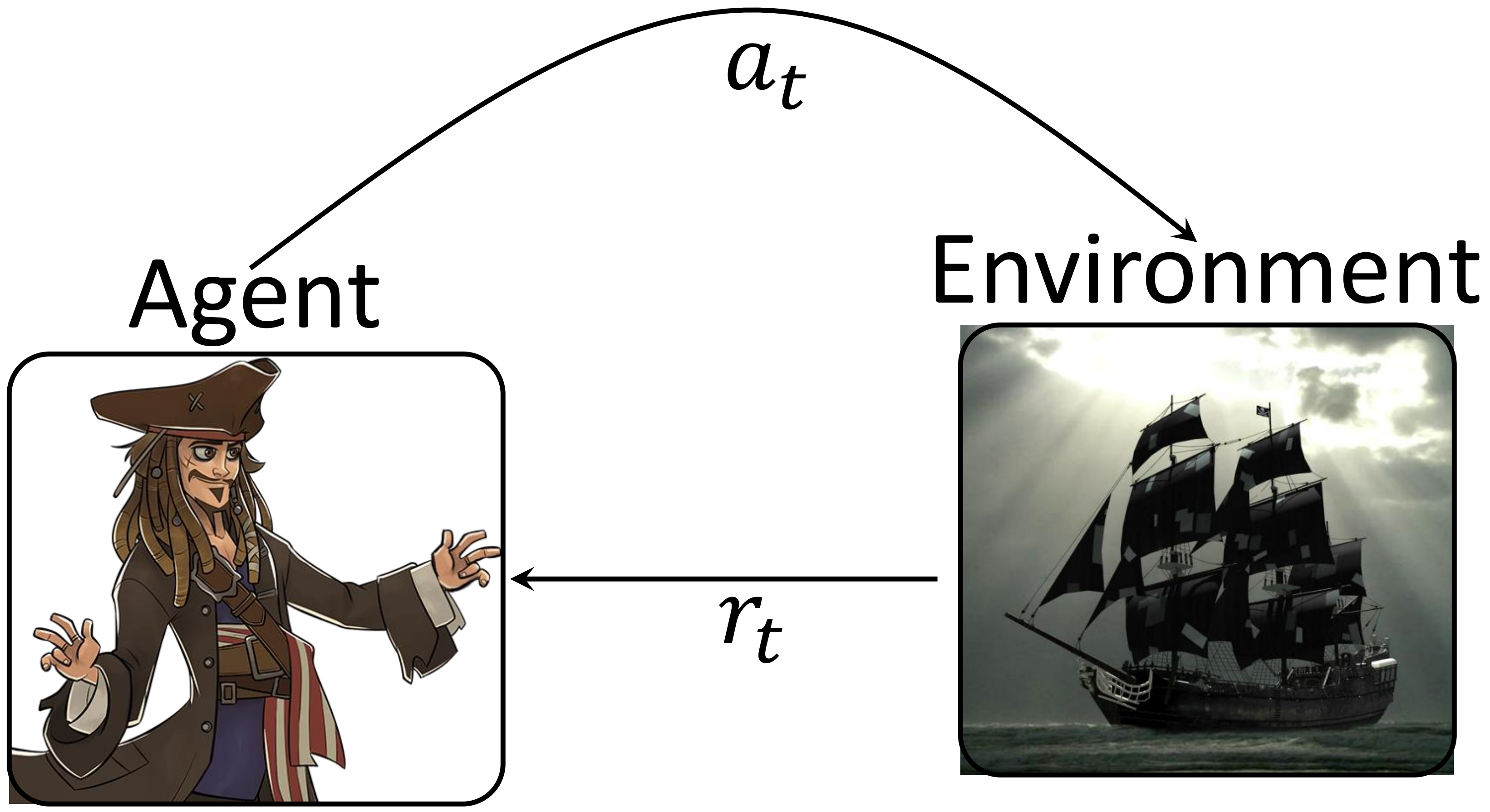
IMPORTANT: The agent **does not** get to see the reward of those actions that it did not take.

Multi-Armed Bandit (MAB) setup is as follows:

- The set of available **actions** is $\mathcal{A} = \{1,2,\cdots,A\}$.
- Action a has the reward distribution P_a. The agent **does not know** P_a for any action a.
- In time slot t:
  Step 1: The agent decides its action a_t.
  Step 2: The agent receives a reward $r_t \sim P_{a_t}$ that is generated by the environment.
- The goal of the agent is to maximize the expected total reward,

$$\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$$

Multi-Armed Bandit Setup



In conventional MAB setup (and RL in general), we are interested in maximizing *reward*. If we want to minimize *cost*, we can simply maximize $-cost$.

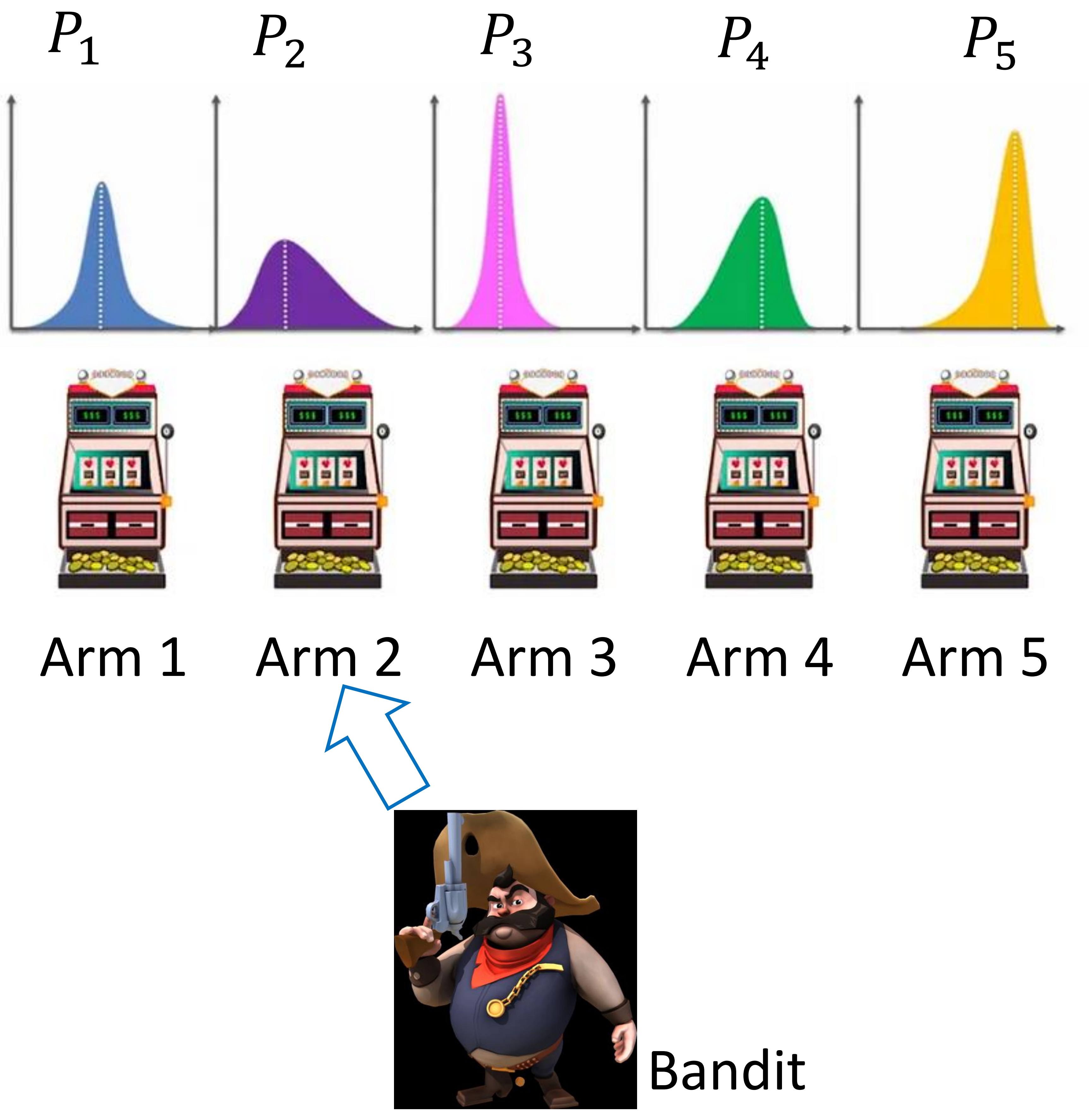
Multi-Armed Bandit (MAB) setup is as follows:

- The set of available **actions** is $\mathcal{A} = \{1,2,\cdots,A\}$.
- Action a has the reward distribution P_a. The agent **does not know** P_a for any action a.
- In time slot t:
  Step 1: The agent decides its action a_t.

  Step 2: The agent receives a reward $r_t \sim P_{a_t}$ that is generated by the environment.
- The goal of the agent is to maximize the expected total reward,

$$\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$$

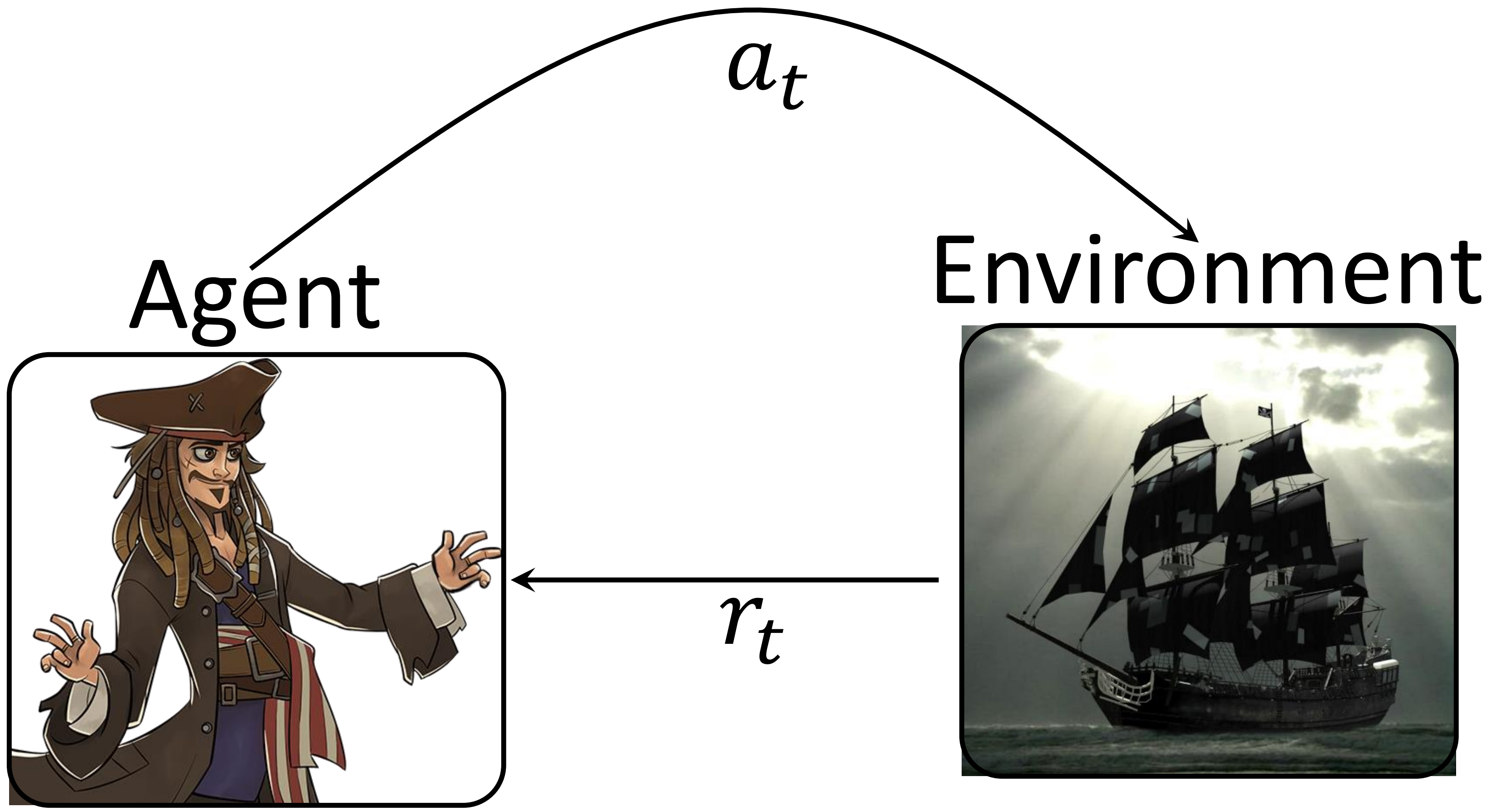
Multi-Armed Bandit Setup



The origin of the name "Multi-Armed Bandit":

- The name originates from a bandit playing lottery machines in a casino.
- Each lottery machine is called an **arm** referring to the lever of the lottery machine. So basically **"arms" and "actions" are analogous**. We will use these two terms interchangeably throughout Module 1.
- Each lottery machine has its own probability distribution of dispensing cash reward. The bandit does not know these distributions.
- The aim of the bandit is to maximize the total cash reward over multiple plays.

Multi-Armed Bandit Setup

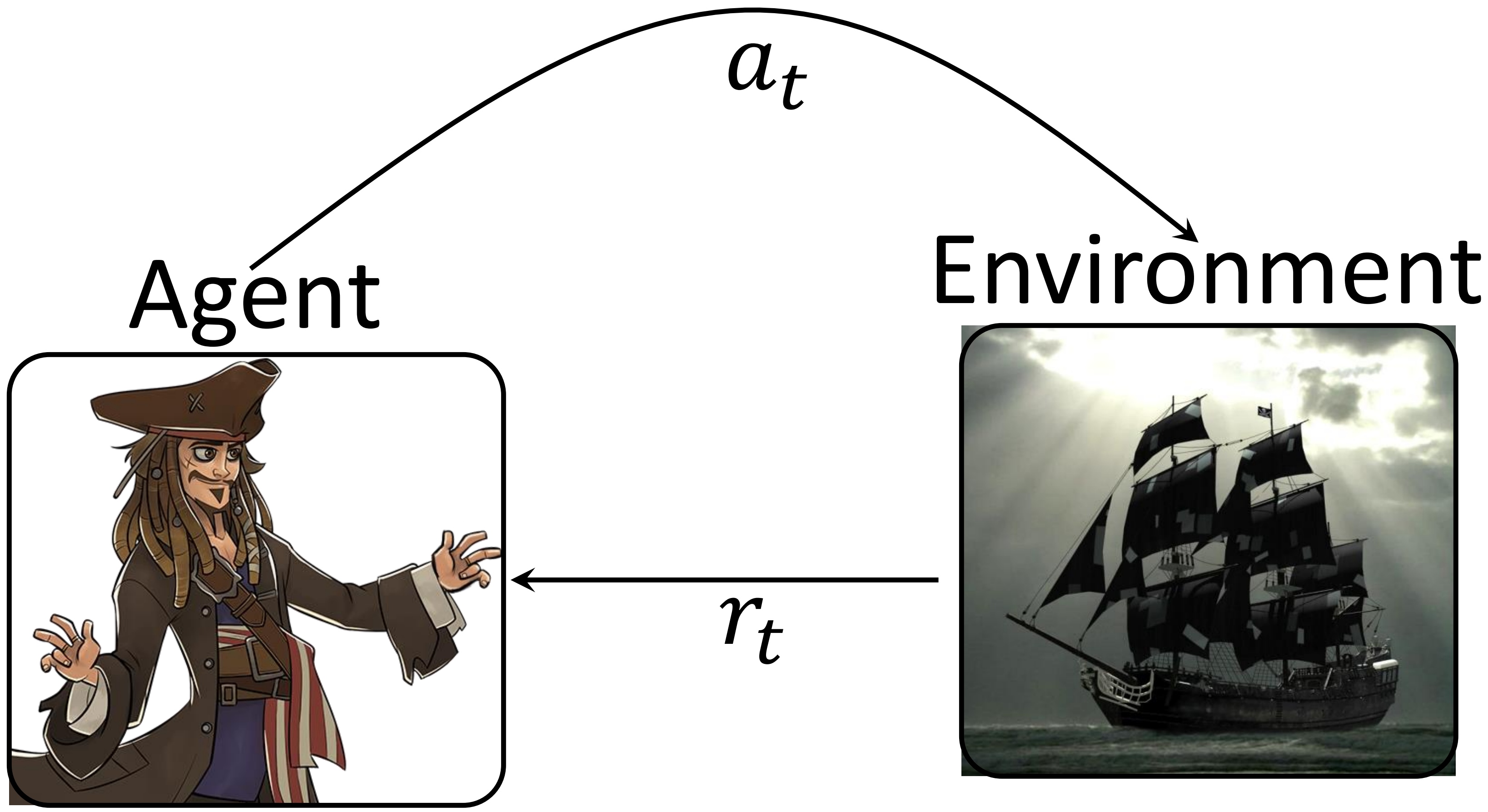


The MAB setup makes a **mild** assumption:

- For all action a, the reward distribution P_a has a **support** of $[0,1]$, i.e. rewards generated by all actions will **always** lie between 0 to 1.

What to do if the rewards of action a lie between $[l_a , u_a]$?

Multi-Armed Bandit Setup



This approach uses **different** scaling and shifting factors for different actions. Scaling and shifting the rewards by different factor will lead to erroneous result when we want to **compare** rewards of two different actions.

The MAB setup makes a **mild** assumption:

- For all action a, the reward distribution P_a has a **support** of $[0,1]$, i.e. rewards generated by all actions will **always** lie between 0 to 1.

What to do if the rewards of action a lie between $[l_a, u_a]$?

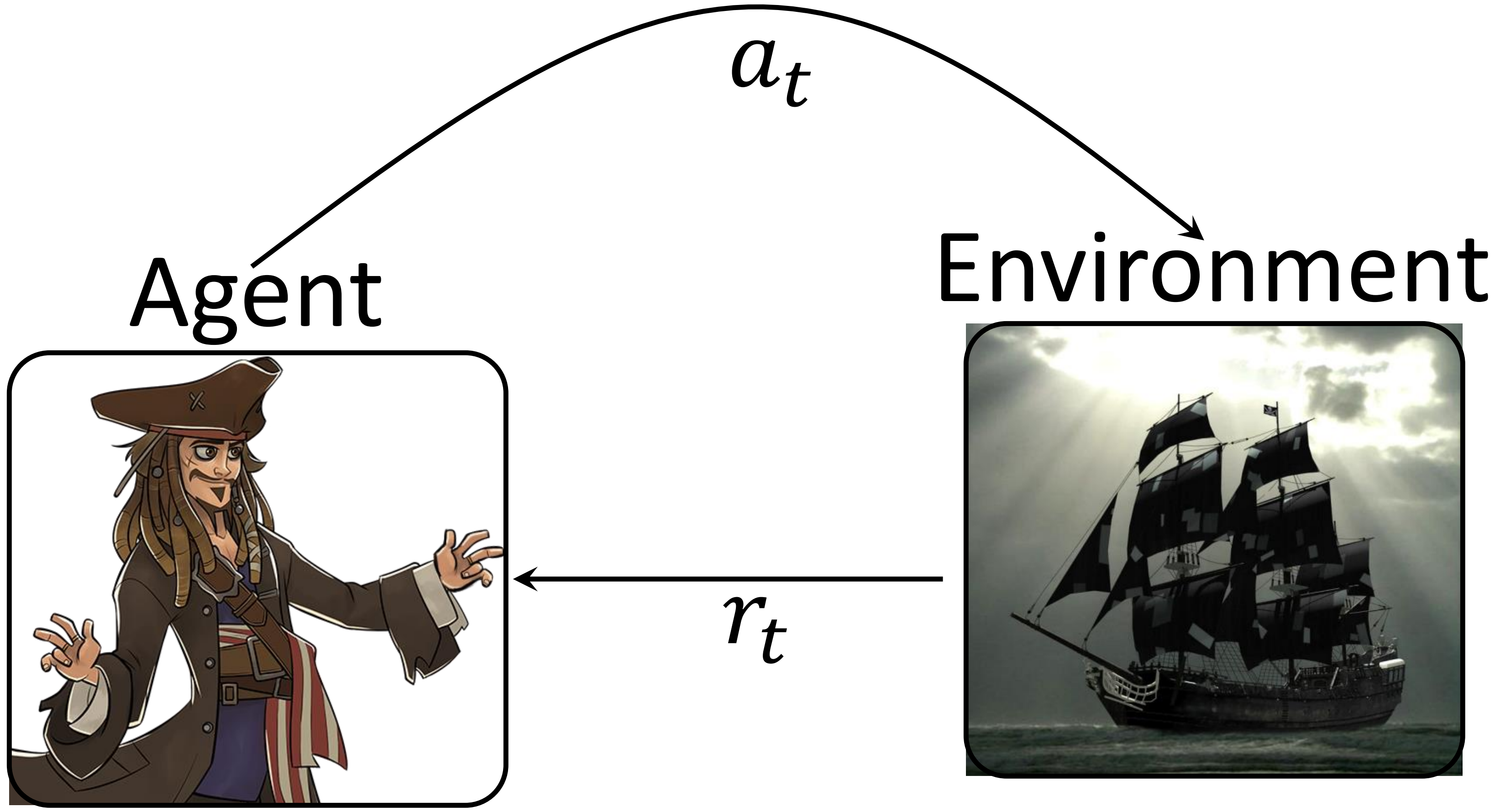
- We can **scale and shift** the rewards.
- The **wrong** way to scale and shift:

$$\tilde{r}^a = \frac{r^a - l_a}{u_a - l_a} \quad \begin{array}{l} \leftarrow \text{Shifting factor} \\ \leftarrow \text{Scaling factor} \end{array}$$

r^a : Reward of action a.

$\tilde{r}^a$: Scaled and shifted reward of action a.

Multi-Armed Bandit Setup



This approach uses **same** scaling and shifting factors for different actions.

To apply this scaling and shifting method you have to **know l_a and u_a for all actions.** This is quite practical in real-life scenarios.

The MAB setup makes a **mild** assumption:

- For all action a, the reward distribution P_a has a **support** of $[0,1]$, i.e. rewards generated by all actions will **always** lie between 0 to 1.

What to do if the rewards of action a lie between $[l_a, u_a]$?

- We can **scale and shift** the rewards.

- The **correct** way to scale and shift:

$$\tilde{r}^a = \frac{r^a - l}{u - l}$$

←Shifting factor

← Scaling factor

where $l = \min_{a \in \mathcal{A}} l_a$

$$u = \max_{a \in \mathcal{A}} u_a$$

Multi-Armed Bandit Setup

Applications of Multi-Armed Bandit:

- **Online advertising:** Which advertisement to show to get more click through rate?
- **Clinical trials:** Which drug or what dose of drug to administer to cure a patient?
- **Network routing:** In wireless/road network, which path to take in order to minimize the time to go from source to destination?
- **Millimeter Wave Communication:** Which direction to point the millimeter wave antenna to maximize throughput?
- **Recommendation systems (Netflix):** Which movies to recommend to maximize user satisfaction?
- **Feature Selection in ML:** Which set of features to select in order to get the best performance of the ML model?

Multi-Armed Bandit Setup

Applications of Multi-Armed Bandit:

- ➢ **Online advertising:** Which advertisement to show to get more click through rate?
- ➢ **Clinical trials:** Which drug or what dose of drug to administer to cure a patient?
- ➢ **Network routing:** In wireless/road network, which path to take in order to minimize the time to go from source to destination?
- ➢ **Millimeter Wave Communication:** Which direction to point the millimeter wave antenna to maximize throughput?
- ➢ **Recommendation systems (Netflix):** Which movies to recommend to maximize user satisfaction?
- ➢ **Feature Selection in ML:** Which set of features to select in order to get the best performance of the ML model?

Strictly, the real-life implementation of these falls under "contextual bandits"!

Multi-Armed Bandit Setup

Applications of Multi-Armed Bandit:

- **Online advertising:** Which advertisement to show to get more click through rate?
- **Clinical trials:** Which drug or what dose of drug to administer to cure a patient?
- **Network routing:** In wireless/road network, which path to take in order to minimize the time to go from source to destination?
- **Millimeter Wave Communication:** Which direction to point the millimeter wave antenna to maximize throughput?
- **Recommendation systems (Netflix):** Which movies to recommend to maximize user satisfaction?
- **Feature Selection in ML:** Which set of features to select in order to get the best performance of the ML model?

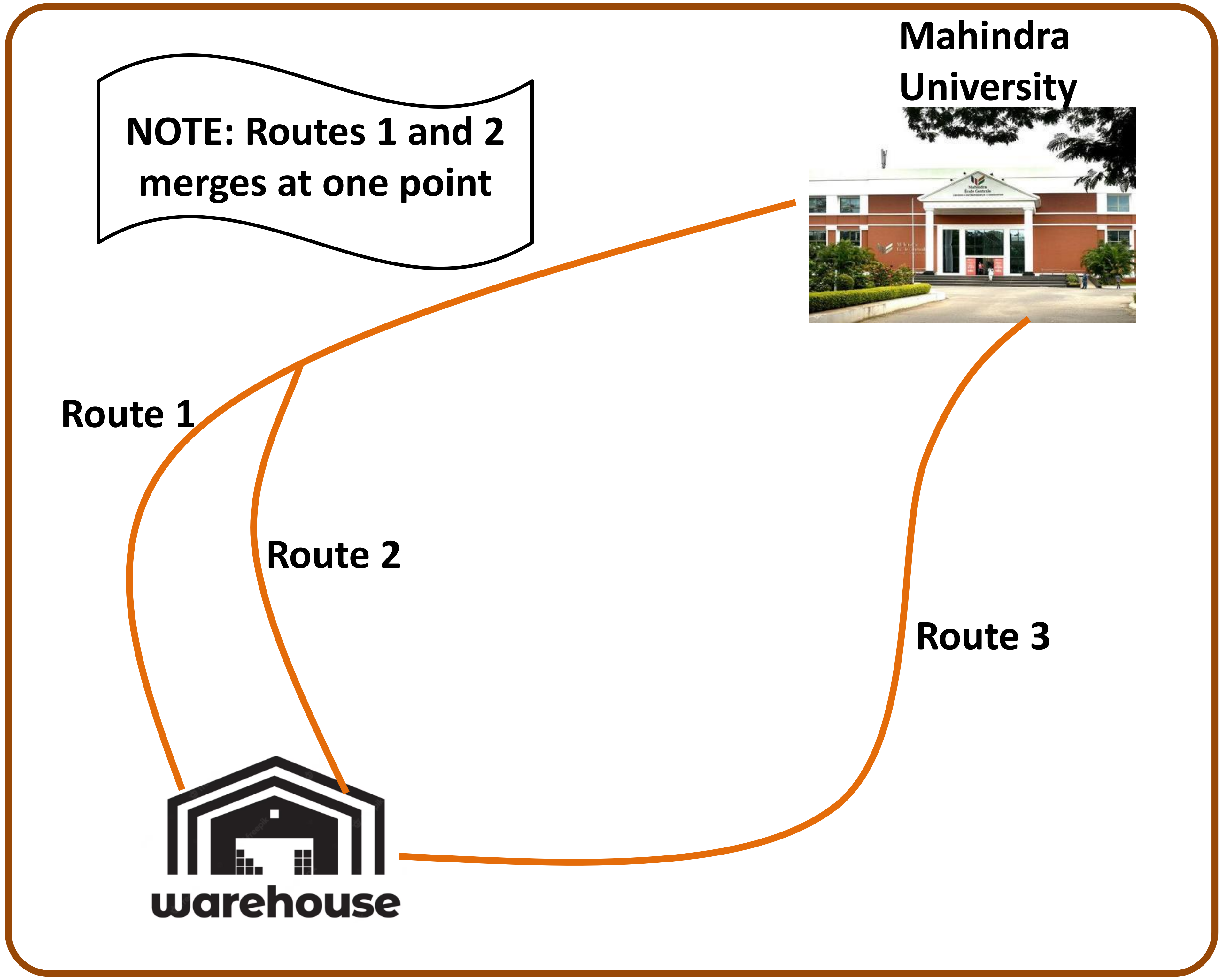
Strictly, they fall under "combinatorial bandits" that are much harder!

Lecture Content

- The Multi-Armed Bandit* (MAB) setup.
- A Fundamental Tradeoff.
- Mathematical Notations and Concepts.
- Policies for Multi-Armed Bandit

*PLEASE NOTE: Multi-Armed Bandit by default means the "simple Multi-Armed Bandit" and NOT "contextual Multi-Armed Bandit".

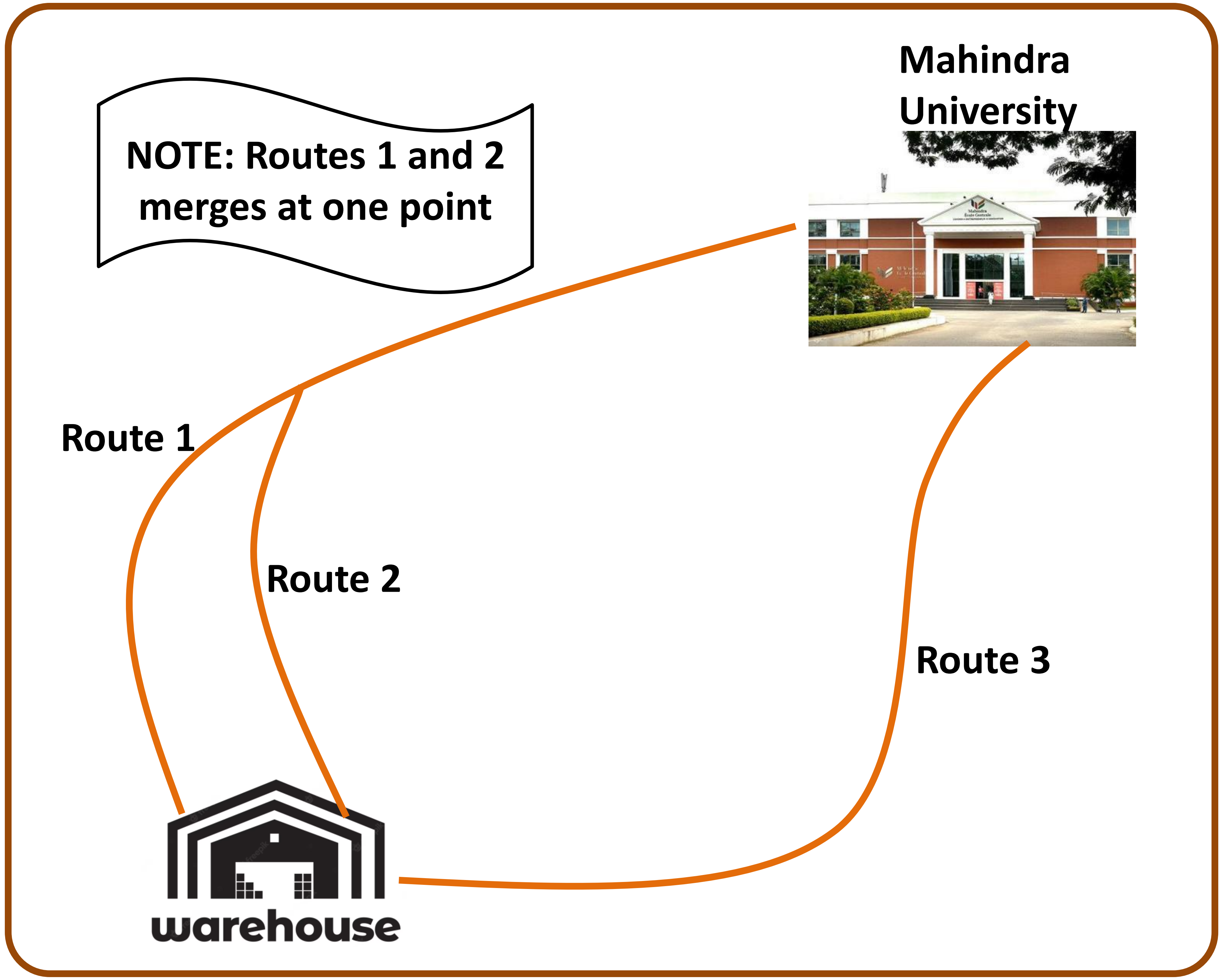
A Fundamental Tradeoff



- Let's get back to the warehouse routing example that we introduced in Lecture 1.
- Suppose you don't have access to any google maps*, how will decide which route to take in order to minimize the average travel time?

*Because google maps information is a **context** and we are not dealing with contextual bandit yet.

A Fundamental Tradeoff



- IMPORTANT: The entirety of RL relies a fundamental concept called **"exploration-exploitation tradeoff"**.

- In this warehouse routing example:
  - **Exploration:** Take a route that might not have given the minimum (average) travel time in the past just to see if it gives better outcome.
  - **Exploitation:** Take the route that has given the minimum (average) travel time in the past.

- If we explore too little, then we might get **stuck** with a **sub-optimal action**. But, if we explore too much we may keep opting for the sub-optimal action again and again.

Lecture Content

- The Multi-Armed Bandit* (MAB) setup.
- A Fundamental Tradeoff.
- Mathematical Notations and Concepts.
- Policies for Multi-Armed Bandit.

*PLEASE NOTE: Multi-Armed Bandit by default means the “simple Multi-Armed Bandit” and NOT “contextual Multi-Armed Bandit”.

Mathematical Notations and Concepts

- $q_*(a)$ denote the **action-value**. It is the **true** expected value of reward from action a,

$$q_*(a) = \mathbb{E}[r_t \mid a_t = a]$$

- $N_t(a)$ denotes the number of times action a was selected prior to time t,

$$N_t(a) = \sum_{\tau=1}^{t} \underbrace{I(a_\tau = a)}_{\text{Indicator function}}$$

- Most of the algorithms for MAB needs a estimate of $q_*(a)$. How to estimate $q_*(a)$ based on the rewards that the agent receives?

Mathematical Notations and Concepts

- $q_*(a)$ denote the **action-value**. It is the **true** expected value of reward from action a,

$$q_*(a) = \mathbb{E}[r_t \mid a_t = a]$$

- $N_t(a)$ denotes the number of times action a was selected prior to time t,

$$N_t(a) = \sum_{\tau=1}^{t-1} I(a_\tau = a)$$

Indicator function

- Most of the algorithms for MAB needs a estimate of $q_*(a)$. At time t, how to estimate $q_*(a)$ based on the rewards that the agent receives?

  We basically take **sample average** of the reward that we **received for action $\boldsymbol{a}$** prior to time t.

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau$$

Mathematical Notations and Concepts

- $q_*(a)$ denote the **action-value.** It is the **true** expected value of reward from action a,

$$q_*(a) = \mathbb{E}[r_t \mid a_t = a]$$

- $N_t(a)$ denotes the number of times action a was selected prior to time t,

$$N_t(a) = \sum_{\tau=1}^{t-1} I(a_\tau = a)$$

Indicator function

- Most of the algorithms for MAB needs a estimate of $q_*(a)$. How to estimate $q_*(a)$ based on the rewards that the agent receives?

We basically take **sample average** of the reward that we **received for action** a prior to time t.

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau$$

Note that at time t, we are only summing up till time $t-1$. This is because $N_t(a)$ and $Q_t(a)$ are used to make a decision at time t. And, we don't know about a_t and r_t before we make the said decision.

Mathematical Notations and Concepts

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau$$

- So, $Q_t(a)$ is an estimate of $q_*(a)$. However, in order to estimate $Q_t(a)$, we need to remember the history of all the rewards. This is neither time nor space efficient!

- How to estimate $Q_{t+1}(a)$ recursively? In other words, how to get $Q_{t+1}(a)$ using:

  - Past information: $N_t(a)$ and $Q_t(a)$.

  - Current information: a_t and r_t.

Mathematical Notations and Concepts

$$
\begin{aligned}
Q_{t+1}(a) &= \frac{1}{N_{t+1}(a)} \sum_{\tau=1}^{t} I(a_\tau = a) r_\tau \\
&= \frac{1}{I(a_t = a) + N_t(a)} \left(I(a_t = a) r_t + \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau \right) \\
&= \frac{I(a_t = a) r_t + Q_t(a) N_t(a)}{I(a_t = a) + N_t(a)}
\end{aligned}
$$

Mathematical Notations and Concepts

$$
\begin{aligned}
Q_{t+1}(a) &= \frac{1}{N_{t+1}(a)} \sum_{\tau=1}^{t} I(a_\tau = a) r_\tau \\
&= \frac{1}{I(a_t = a) + N_t(a)} \left(I(a_t = a) r_t + \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau \right) \\
&= \frac{I(a_t = a) r_t + Q_t(a) N_t(a)}{I(a_t = a) + N_t(a)}
\end{aligned}
$$

- If $a_t = a$, then $I(a_t = a) = 1$. So we have,

$$
\begin{aligned}
Q_{t+1}(a) &= \frac{r_t + Q_t(a) N_t(a)}{1 + N_t(a)} \\
&= Q_t(a) + \frac{1}{1 + N_t(a)} \big(r_t - Q_t(a)\big)
\end{aligned}
$$

Mathematical Notations and Concepts

$$Q_{t+1}(a) = \frac{1}{N_{t+1}(a)} \sum_{\tau=1}^{t} I(a_\tau = a) r_\tau$$

$$= \frac{1}{I(a_t = a) + N_t(a)} \left(I(a_t = a) r_t + \sum_{\tau=1}^{t-1} I(a_\tau = a) r_\tau \right)$$

$$= \frac{I(a_t = a) r_t + Q_t(a) N_t(a)}{I(a_t = a) + N_t(a)}$$

- If $a_t \neq a$, then $I(a_t = a) = 0$. So we have,

$$Q_{t+1}(a) = \frac{0 + Q_t(a) N_t(a)}{0 + N_t(a)}$$

$$= Q_t(a)$$

Mathematical Notations and Concepts

- Recursive formula for $Q_t(a)$:

$$Q_{t+1}(a) = \begin{cases} Q_t(a) + \dfrac{1}{1 + N_t(a)}\big(r_t - Q_t(a)\big)\,; & a_t = a \\ Q_t(a) & ;\ a_t \neq a \end{cases}$$

- Recursive formula for $N_t(a)$:

$$N_{t+1}(a) = \begin{cases} N_t(a) + 1\,; & a_t = a \\ N_t(a) & ;\ a_t \neq a \end{cases}$$

Mathematical Notations and Concepts

- Recursive formula for $Q_t(a)$:

$$Q_{t+1}(a) = \begin{cases} Q_t(a) + \dfrac{1}{1+N_t(a)}\big(r_t - Q_t(a)\big) & ; a_t = a \\ Q_t(a) & ; a_t \neq a \end{cases}$$

- **This formula** has surprising similarity with **gradient descent**.
  - The term inside the red circle is analogous to **step size**.
  - The term inside the green circle is analogous to **gradient**. This is basically an **error term** because we were expecting the reward to be $Q_t(a)$ but the reward that we obtained is r_t. So, the formula is basically using this error term to update the value of $Q_t(a)$.

Mathematical Notations and Concepts

- Recursive formula for $Q_t(a)$:

$$Q_{t+1}(a) = \begin{cases} \boxed{Q_t(a) + \frac{1}{1 + N_t(a)}\left(r_t - Q_t(a)\right)} & ;\ a_t = a \\ Q_t(a) & ;\ a_t \neq a \end{cases}$$

- Can **this formula** be applied for **non-stationary MAB** , i.e. MAB where the reward distribution P_a is a function of time t? E.g. the warehouse routing example where the traffic conditions of routes changes with time.

Mathematical Notations and Concepts

- Recursive formula for $Q_t(a)$:

$$Q_{t+1}(a) = \begin{cases} \boxed{Q_t(a) + \dfrac{1}{1+N_t(a)}\left(r_t - Q_t(a)\right)} & ; a_t = a \\ Q_t(a) & ; a_t \neq a \end{cases}$$

- Can **this formula** be applied for **non-stationary MAB** , i.e. MAB where the reward distribution P_a is a function of time t? **NOT DIRECTLY!**
  - This is because as t increases, the step size term decreases. Hence, for large t the current reward r_t does not effect the update of $Q_t(a)$ by a lot.
  - This is acceptable for stationary MAB because the statistical property of r_t does not change with time.
  - However, for non-stationary MAB, the statistical property of r_t changes with time. Hence, current r_t should he given more weightage as it is more representative of the current reward distribution. But unfortunately, just the opposite in happening in the above formula.

Mathematical Notations and Concepts

- Recursive formula for $Q_t(a)$:

$$Q_{t+1}(a) = \begin{cases} \boxed{Q_t(a) + \frac{1}{1+N_t(a)}\left(r_t - Q_t(a)\right)} & ; a_t = a \\ Q_t(a) & ; a_t \neq a \end{cases}$$

- Can **this formula** be applied for **non-stationary MAB** , i.e. MAB where the reward distribution P_a is a function of time t?

- One possible heuristic is to use a constant step-size α as follows,

$$Q_{t+1}(a) = \begin{cases} Q_t(a) + \alpha\left(r_t - Q_t(a)\right) & ; a_t = a \\ Q_t(a) & ; a_t \neq a \end{cases}$$

Mandatory reading: Section 2.5 (Tracking a Nonstationary Problem), page 32-33, of the book by Sutton and Barto (prescribed text book). This section explains the working of the formula.

Mathematical Notations and Concepts

- The optimal action a^* is,

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; q_*(a)$$

- To measure the performance of a policy, we need a metric (like time/space complexity). One possible metric is the **expected value of the total reward**,

$$\mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau\right]$$

- However, the most conventional metric used in RL (and hence MAB) is the concept of **"regret"**. In the following slides:
  - We will first define regret.
  - We will then show that maximizing the expected value of the total reward is same as minimizing the regret.

Mathematical Notations and Concepts

- For MAB, the regret of a policy π is the **expected value** of the difference between the **total reward earned by π** and the **total reward earned by the policy that knows the optimal action**. Mathematically,

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau^{a^*} - \sum_{\tau=1}^{t} r_\tau\right]$$

where r_τ is the reward earned by π at time slot τ.

$r_\tau^{a^*}$ is the reward earned by the policy that knows the optimal action a^* at time τ.

- A lower regret is better, i.e. we want to minimize the difference in the reward collected by π and the policy that knows the optimal action.

NOTE: Even if we want to minimize cost (instead of maximizing reward) we still want to have a lower regret. Think why...

Mathematical Notations and Concepts

- For MAB, the regret of a policy π is the **expected value** of the difference between the **total reward earned by $\boldsymbol{\pi}$** and the **total reward earned by the policy that knows the optimal action**. Mathematically,

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau^{a^*} - \sum_{\tau=1}^{t} r_\tau\right]$$

where r_τ is the reward earned by π at time slot τ.

$r_\tau^{a^*}$ is the reward earned by the policy that knows the optimal action a^* at time τ.

- **Why regret and not expected total reward?** Because regret gives as an estimate of the **"optimality gap"**, i.e. how our policy is going to perform compared to the best learning policy (best learning policy cannot perform better than the policy that already knows the reward distributions).

Mathematical Notations and Concepts

- For MAB, the regret of a policy π is the expected value of the difference between the total reward earned by $\boldsymbol{\pi}$ and the total reward earned by the policy that knows the optimal action. Mathematically,

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau^{a^*} - \sum_{\tau=1}^{t} r_\tau\right]$$

$$= \sum_{\tau=1}^{t}\left(\mathbb{E}\left[r_\tau^{a^*}\right] - \mathbb{E}[r_\tau]\right)$$

$$= \sum_{\tau=1}^{t}\left(q_*(a^*) - \mathbb{E}[r_\tau]\right) = t q_*(a^*) - \sum_{\tau=1}^{t} \mathbb{E}[r_\tau]$$

This is a constant term.

The expected value of total reward.

Mathematical Notations and Concepts

- For MAB, the regret of a policy π is the **expected value** of the difference between the **total reward earned by π** and the **total reward earned by the policy that knows the optimal action**. Mathematically,

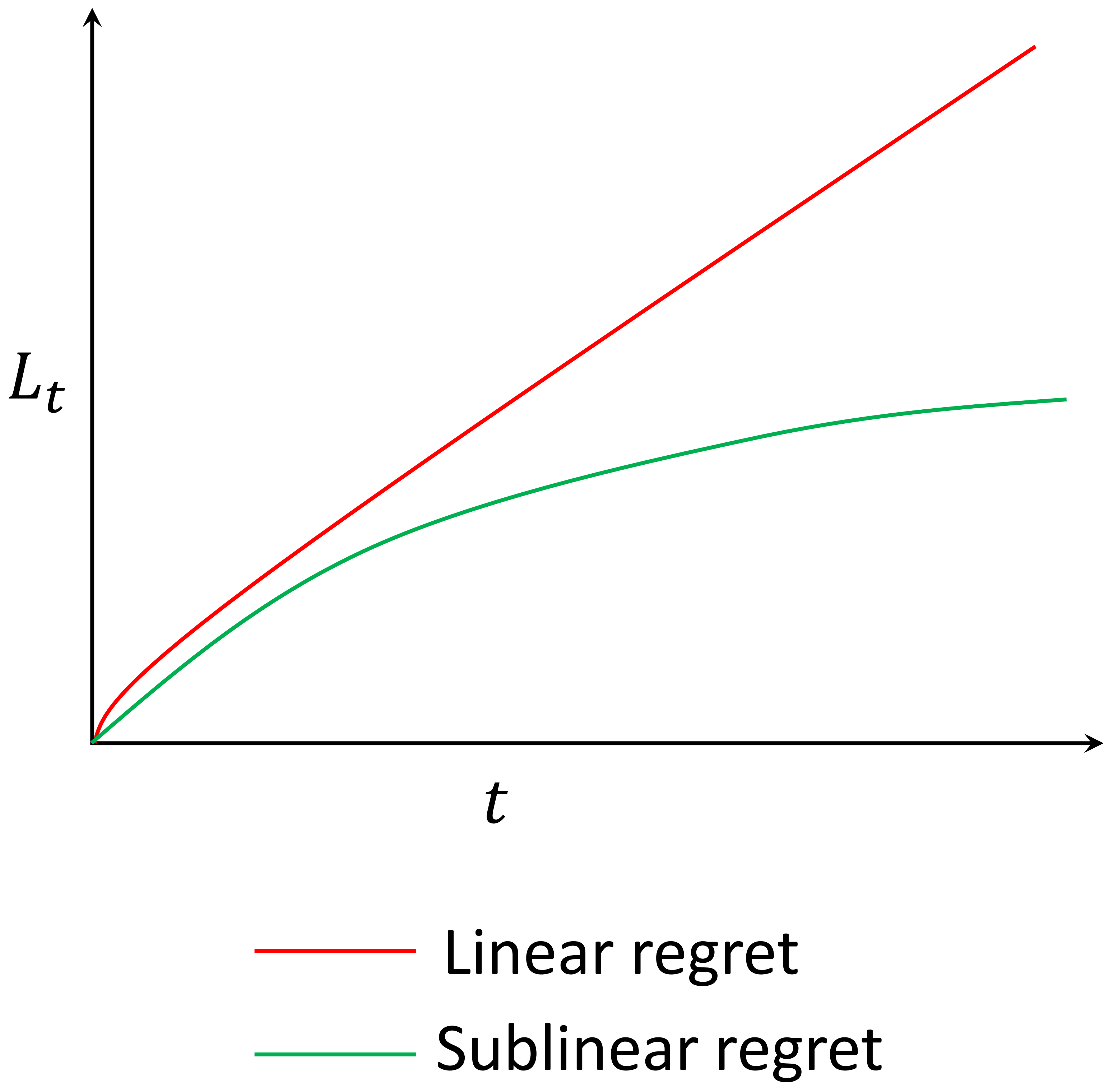
$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} r_\tau^{a^*} - \sum_{\tau=1}^{t} r_\tau\right]$$

$$= \sum_{\tau=1}^{t}\left(\mathbb{E}\left[r_\tau^{a^*}\right] - \mathbb{E}[r_\tau]\right)$$

$$= \sum_{\tau=1}^{t}\left(q_*(a^*) - \mathbb{E}[r_\tau]\right) = t q_*(a^*) - \sum_{\tau=1}^{t} \mathbb{E}[r_\tau]$$

Hence, minimizing L_t is same as maximizing the expected value of total reward.

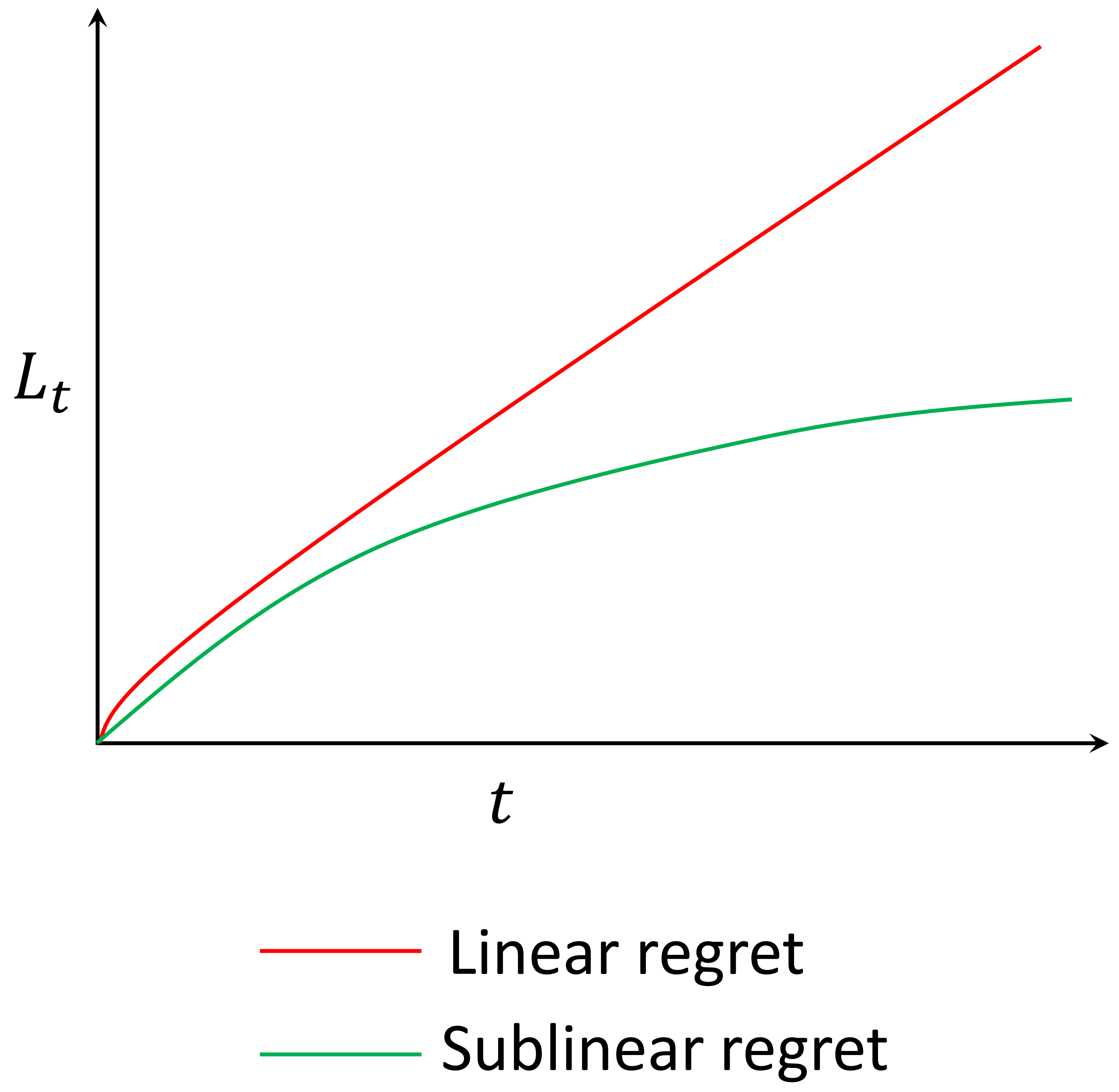
Mathematical Notations and Concepts



$$L_t = \sum_{\tau=1}^{t} \left(q_*(a^*) - \mathbb{E}[r_\tau]\right)$$

- Linear regret means L_t grows linearly with time t. For MAB, policies with sub-linear regret is usually of the form $L_t = \mathcal{O}\big(log(t)\big)$.
- Off course, sub-linear regret is better because lower regret is better.
- Two thumb rules (not a theorem), to achieve sub-linear regret are:
  - The agent should not **completely stop** exploring actions that are **possibly** sub-optimal.
  - The **probability of exploring sub-optimal actions should tend to zero** as time increases.

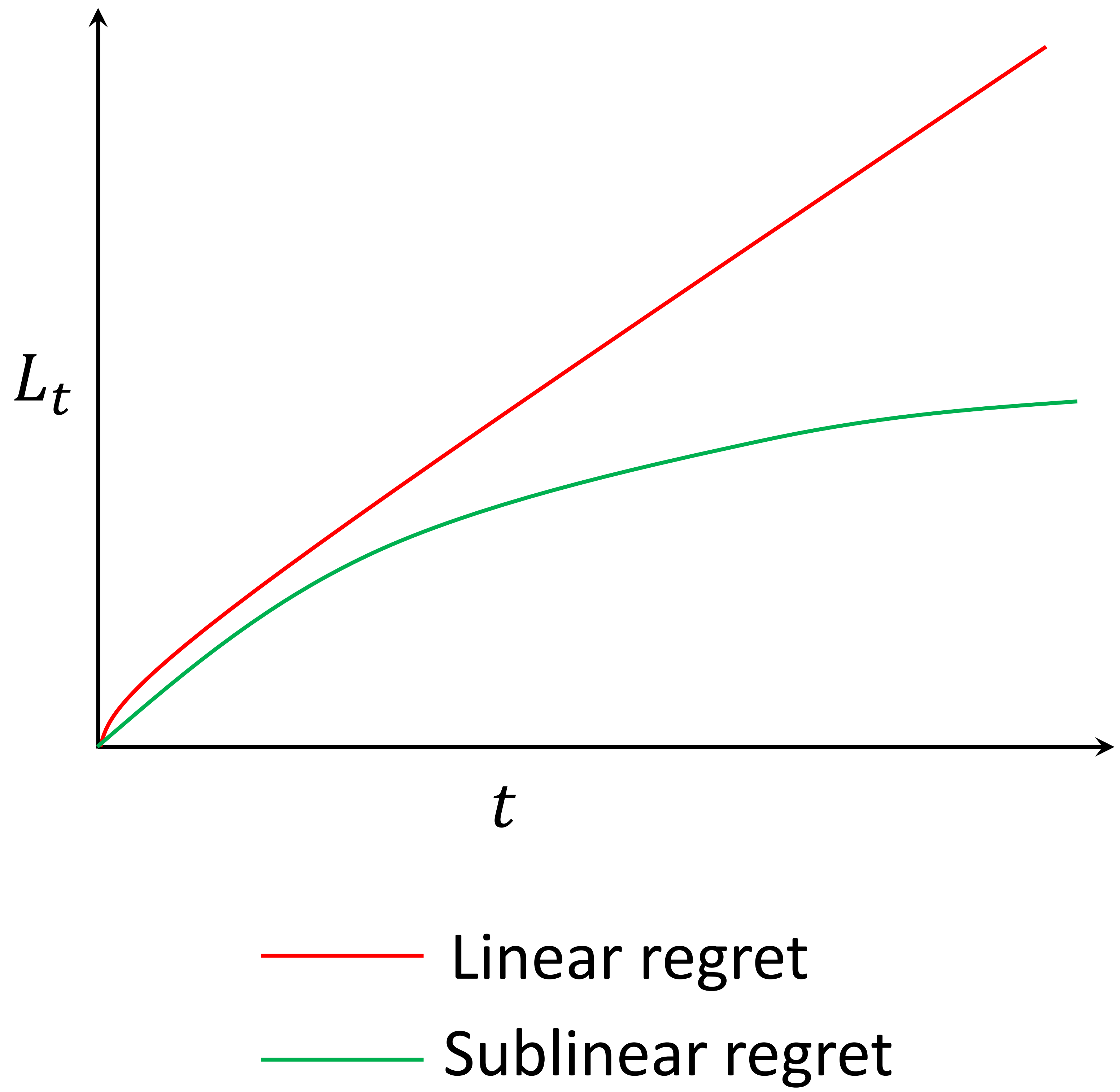
Mathematical Notations and Concepts



$$L_t = \sum_{\tau=1}^{t} \left(q_*(a^*) - \mathbb{E}[r_\tau]\right)$$

- Linear regret means L_t grows linearly with time t. For MAB, policies with sub-linear regret is usually of the form $L_t = \mathcal{O}\big(log(t)\big)$.
- Off course, sub-linear regret is better because lower regret is better.
- A good chunk of theoretical work in reinforcement learning (including MAB) involves deriving the formula for regret. In this course, we will only evaluate regret (among other performance measures like time-average expected reward) using simulations.
  (Explain about average reward and steady state)

Mathematical Notations and Concepts

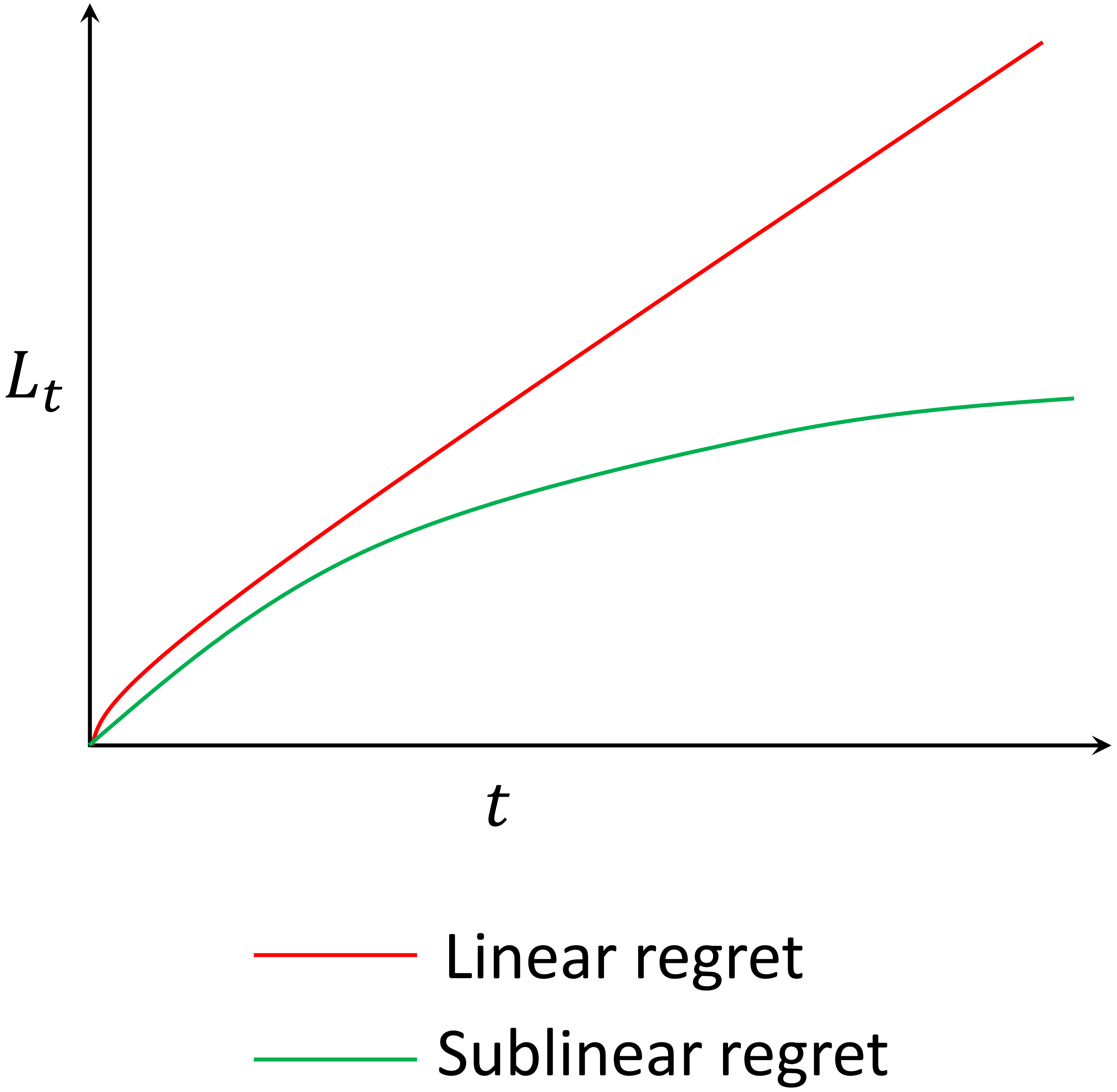


An alternate perspective of linear and sub-linear regret

- Consider $\frac{L_t}{t}$. This quantity is the **average error** in reward between the policy that know the best arm and the policy π. From the perspective of control theory, this is the **steady state error**.

- If the policy π has a linear regret, then $L_t \approx \delta t$ for large t. Then, $\frac{L_t}{t} \approx \delta$; a **finite error** even as t tends to infinity.

Mathematical Notations and Concepts



An alternate perspective of linear and sub-linear regret

- Consider $\frac{L_t}{t}$. This quantity is the **average error** in reward between the policy that know the best arm and the policy π. From the perspective of control theory, this is the **steady state error**.

- If the policy π has a sub-linear regret $\mathcal{O}\big(log(t)\big)$,

  then $L_t \approx \delta log(t)$ for large t. Then,

  $\frac{L_t}{t} \approx \frac{\delta log(t)}{t}$ which tends to 0 as t tends to

  infinity.

Lecture Content

- The Multi-Armed Bandit* (MAB) setup.
- A Fundamental Tradeoff.
- Mathematical Notations and Concepts.
- Policies for Multi-Armed Bandit
  - ε-greedy policy.
  - Upper Confidence Bound policy.
  - Policy gradient.

*PLEASE NOTE: Multi-Armed Bandit by default means the "simple Multi-Armed Bandit" and NOT "contextual Multi-Armed Bandit".

ε −Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - **Explore** with a probability of ε.
  - **Exploit** with a probability of $1 - \varepsilon$.

ε −Greedy Policy

Interchanged

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- *A confusing attribute about the algorithm name*: Greedy is synonymous to exploitation. So logically it is intuitive to think that ε −greedy means:
  - Exploit with a probability of ε.
  - Explore with a probability of $1 - \varepsilon$.

ε −Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.

  2. If $x \leq \varepsilon$:

     a_t is chosen uniformly at random from $\mathcal{A}$.

     Else:

     $$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q_t(a)$$

  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.

  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

This is NOT a pseudocode. Refer page 32 of "the book" for the pseudocode.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1-\varepsilon$.

- At time t:

  $x = np.random.uniform(0,1)$

  1. Sample a random variable x between 0 to 1 from a uniform distribution.
  2. If $x \leq \varepsilon$:

     a_t is chosen uniformly at random from $\mathcal{A}$.

     Else:

     $$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$$

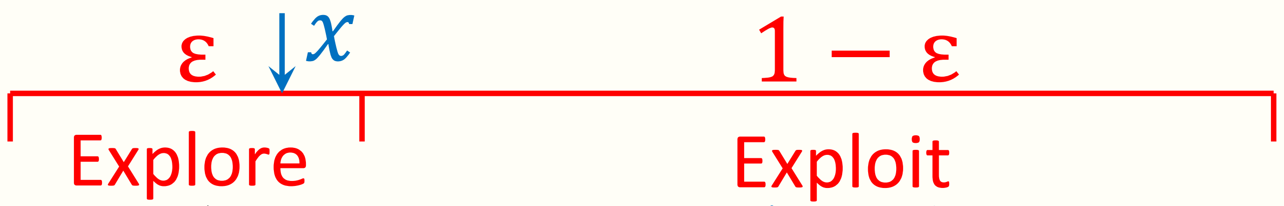
  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.



  2. If $x \leq \varepsilon$:
     a_t is chosen uniformly at random from $\mathcal{A}$. (Exploration step)

     Else:
     $$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$$ (Exploitation step)

  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.

  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - **Explore** with a probability of ε.
  - **Exploit** with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.
     $x = np.random.randint(1, A + 1)$
  2. If $x \le \varepsilon$:
     a_t is chosen uniformly at random from $\mathcal{A}$. (Exploration step)
     The set of all actions, $\mathcal{A} = \{1,2, \ldots, A\}$
     Else:
     $$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$$
     (Exploitation step)
  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1-\varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.
  2. If $x \leq \varepsilon$:

     a_t is chosen uniformly at random from $\mathcal{A}$. (Exploration step)

     Else:

     $$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q_t(a)$$ (Exploitation step)

     Choose the action with the highest sample average reward.
  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.

  2. If $x \leq \varepsilon$:
     a_t is chosen uniformly at random from $\mathcal{A}$.
     Else:
     $$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q_t(a)$$

  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.

     In **real world**, this step is not part of the agents policy; the environment will give a reward.

  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.

  2. If $x \leq \varepsilon$:

     a_t is chosen uniformly at random from $\mathcal{A}$.

     Else:

     $$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t(a)$$

     When we are trying to **simulate** the real world, we have to sample the reward from distribution P_{a_t}.

  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.

     If P_{a_t} is a distribution of discrete random variable, you can use np.random.choice(). Else, I taught you a way in previous lecture.

  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.

  2. If $x \leq \varepsilon$:

     a_t is chosen uniformly at random from $\mathcal{A}$.

     Else:

     $$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q_t(a)$$

  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.

Refer to slide 23.

  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

ε −Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.
- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.
- At time t:
  1. Sample a random variable x between 0 to 1 from a uniform distribution.
  2. If $x \leq \varepsilon$:
     a_t is chosen uniformly at random from $\mathcal{A}$.
     Else:
     $$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q_t(a)$$
  3. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  4. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

IMPORTANT: This policy uses the estimate of the action-value to make decisions. Such policies are called **"Value function" based policy**.

Refer to slide 23.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- In practice, **$\boldsymbol{\varepsilon}$ is not a constant. Rather it varies with $\boldsymbol{t}$, i.e. $\boldsymbol{\varepsilon_t}$**.
  - ε_t is high in the beginning to explore more.
  - As t increases, the agent has a good idea above the action-value of all the arms. Hence, ε_t decreases as t increases.
  - We can choose our own custom strategy to change ε_t but a common strategy is $\varepsilon_t = \frac{\delta}{t^n}$ where $n = 1,2, \ldots$

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- In practice, **ε is not a constant. Rather it varies with t, i.e. ε_t**.
  - ε_t is high in the beginning to explore more.

  - As t increases, the agent has a good idea above the action-value of all the arms. Hence, ε_t decreases as t increases.

  - We can choose our own custom strategy to change ε_t but a common strategy is $\varepsilon_t = \frac{\delta}{t^n}$ where $n = 1,2,\ldots$

This is similar to the decreases in step-size during training ML/DL models.

ε –Greedy Policy

- As we discussed a little while before, there is a need to balance exploration and exploitation.

- We decide this "balance" using a probability parameter $\varepsilon \in (0,1)$. To elaborate:
  - Explore with a probability of ε.
  - Exploit with a probability of $1 - \varepsilon$.

- In practice, **ε is not a constant. Rather it varies with t, i.e. ε_t**.
  - ε_t is high in the beginning to explore more.

  - As t increases, the agent has a good idea above the action-value of all the arms. Hence, ε_t decreases as t increases.

  - We can choose our own custom strategy to change ε_t but a common strategy is $\varepsilon_t = \frac{\delta}{t^n}$ where $n = 1,2,\ldots$

Even though ε –Greedy policy **works very well in practice** with varying ε_t, there is **no theoretical proof of sublinear regret** with varying ε_t.

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between $[0, 1]$**. If not, them **shift and scale**.

- At time t:
  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between $[\mathbf{0}, \mathbf{1}]$**. If not, them **shift and scale**.

- At time t:
  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

Just like ε —Greedy policy, this policy also uses the estimate of the action-value to make decisions. Hence, it is a **"Value function" based policy**.

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between $[0, 1]$**. If not, them **shift and scale**.

- At time t:
  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

These steps are same as ε −Greedy policy.

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between $[0,1]$**. If not, them **shift and scale**.

- At time t:
  1. Set $a_t = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\left(Q_t(a) + \sqrt{\frac{2ln(t)}{N_t(a)}}\right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

The first part of this formula also resembles the exploitation step of ε −Greedy policy.

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between [0, 1]**. If not, them **shift and scale**.

- At time t:

  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \boxed{\sqrt{\frac{2 ln(t)}{N_t(a)}}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

What on earth is this weirdness!!!

Upper Confidence Bound (UCB) Policy

- To implement this algorithm we **must ensure that the rewards are between $[\mathbf{0}, \mathbf{1}]$**. If not, them **shift and scale**.

- At time t:
  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

This component is responsible for **exploitation**.

This component is responsible for **exploration**.

Upper Confidence Bound (UCB) Policy

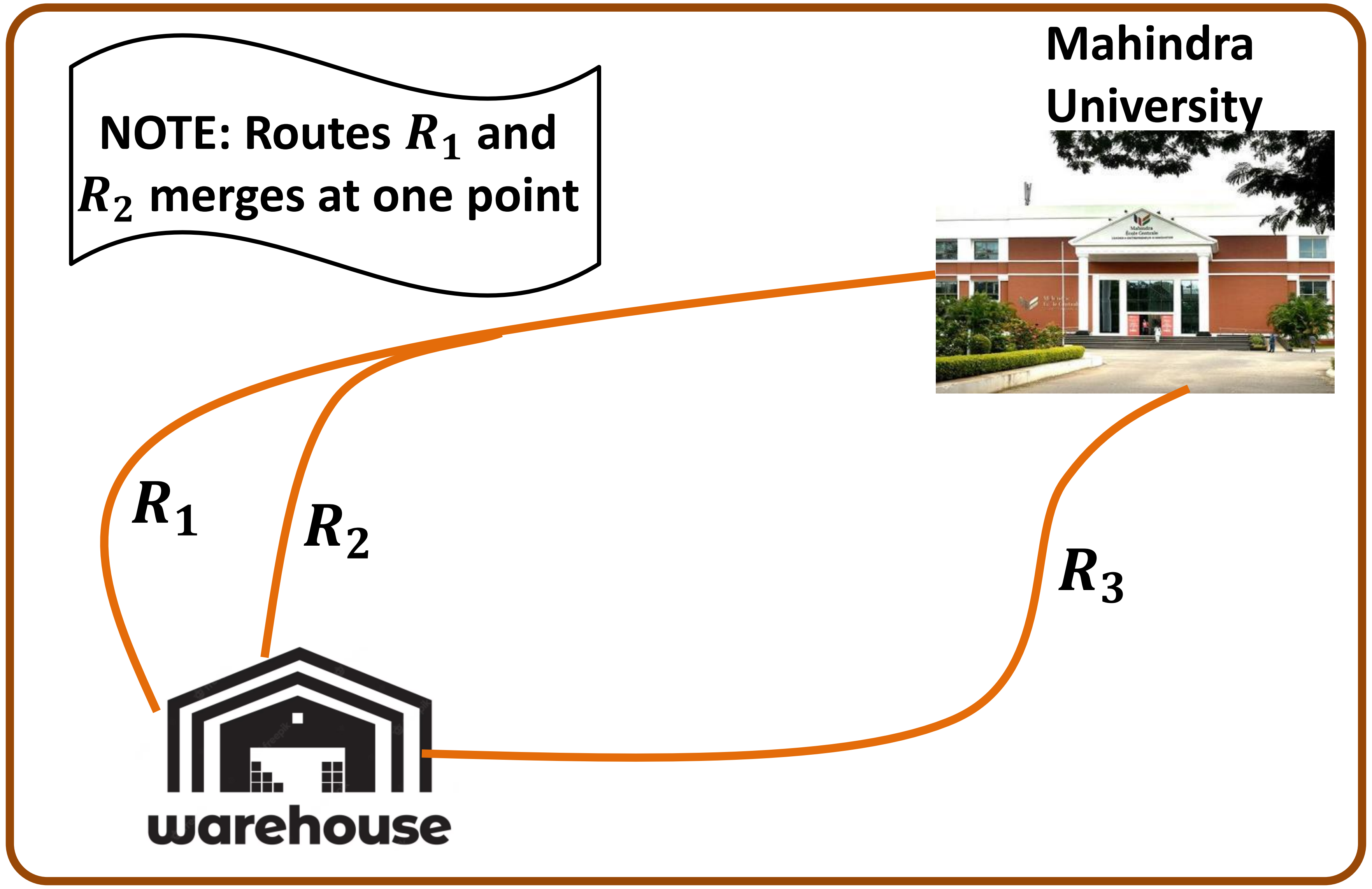
- To implement this algorithm we **must ensure that the rewards are between $[0, 1]$**. If not, them **shift and scale**.

Just FYI, this is natural logarithm.

- At time t:
  1. Set $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$.
  2. Based on the action chosen, the agent will receive a reward $r_t \sim P_{a_t}$ from the environment.
  3. Update $Q_{t+1}(a)$ and $N_{t+1}(a)$ for all $a \in \mathcal{A}$ based on action a_t and reward r_t.

When $N_t(a)$ is low, then action a has not been selected much by the policy. So, by having $N_t(a)$ in the denominator, UCB policy explores those actions more that has not been explored much before.

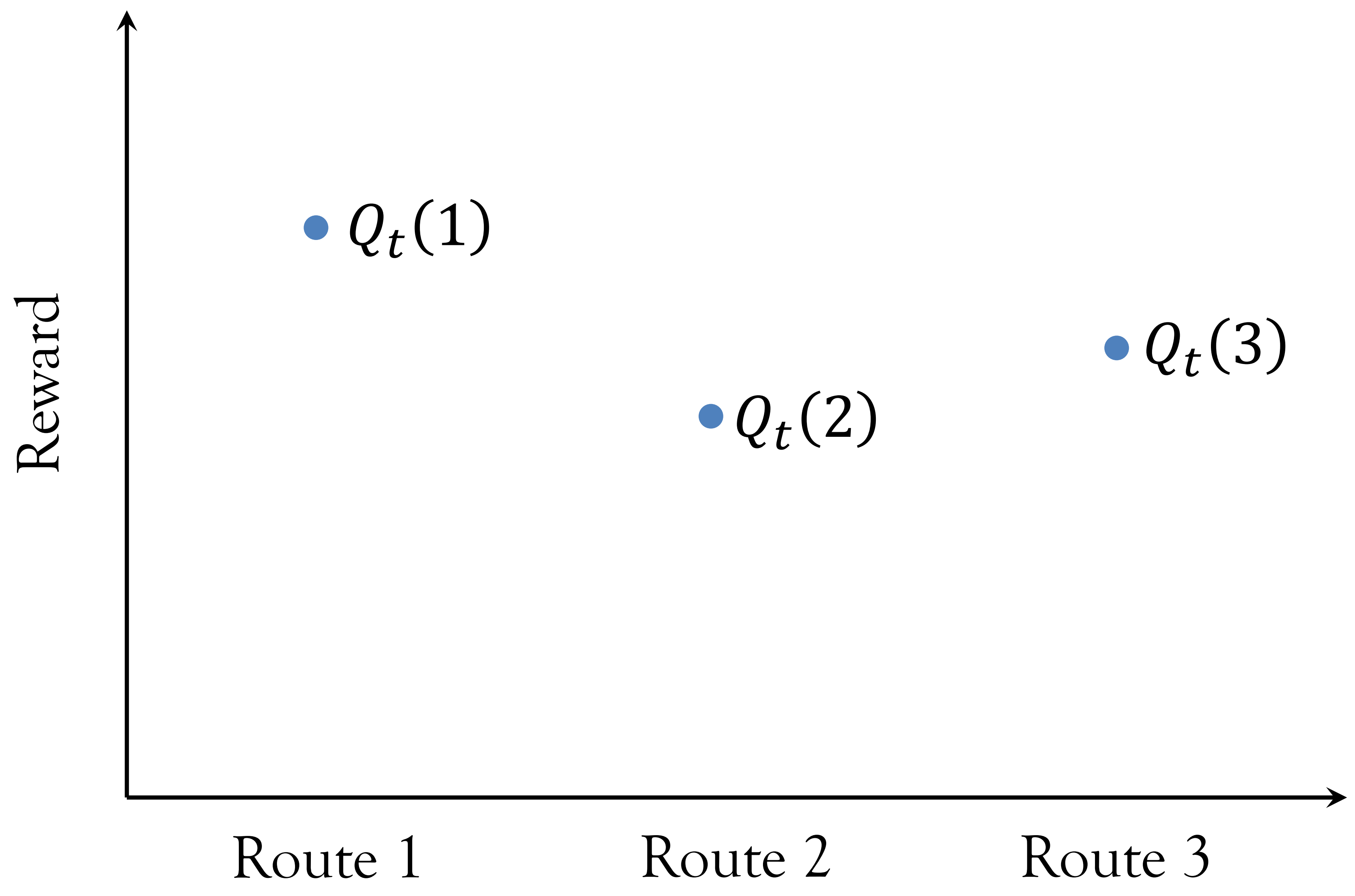
Upper Confidence Bound (UCB) Policy



$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(Q_t(a) + \sqrt{\frac{2 ln(t)}{N_t(a)}} \right)$$

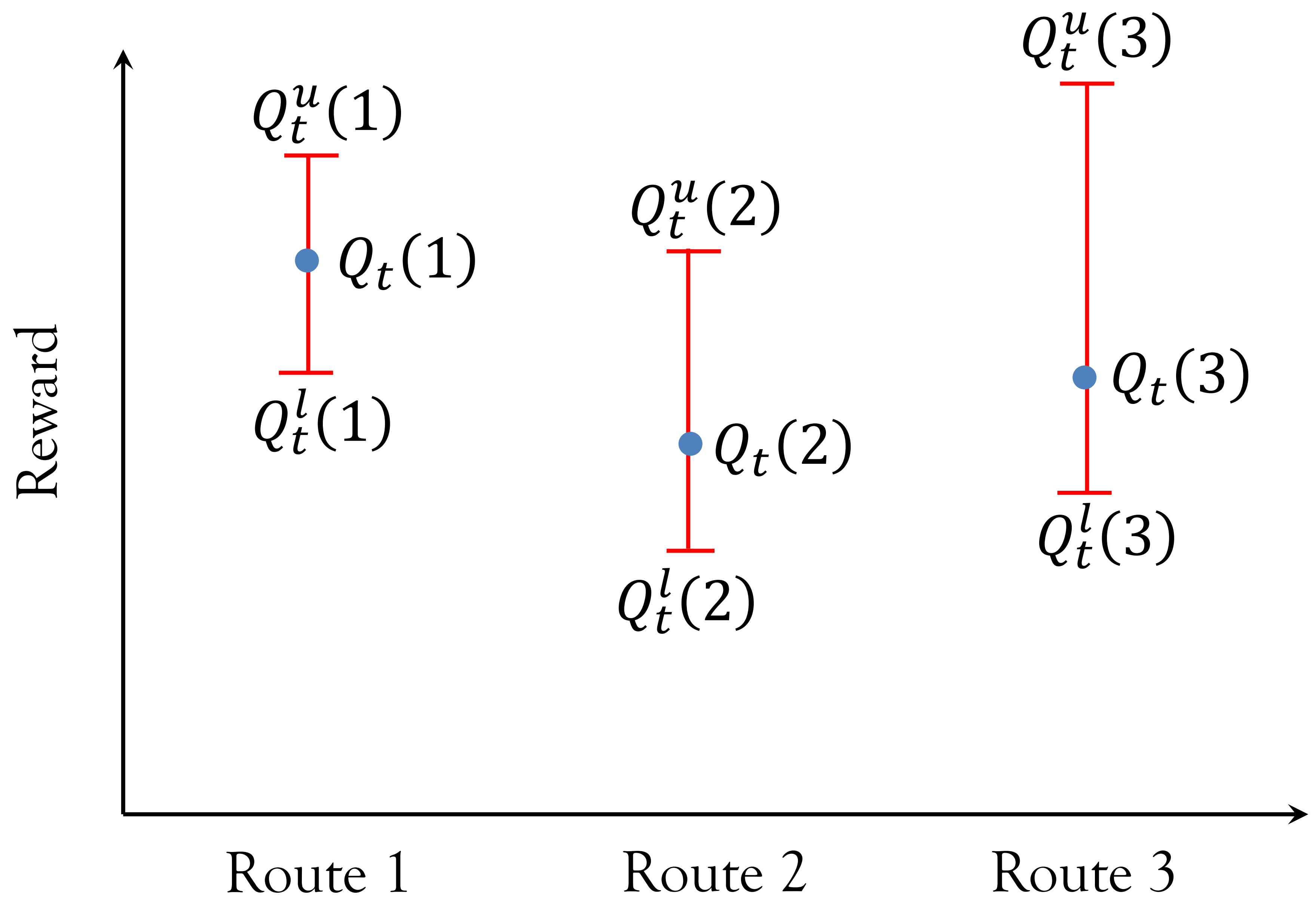
- In the following few slides, we do a **partial derivation** of the above formula.
- We will use the **warehouse routing** example to aid our understanding.

Upper Confidence Bound (UCB) Policy



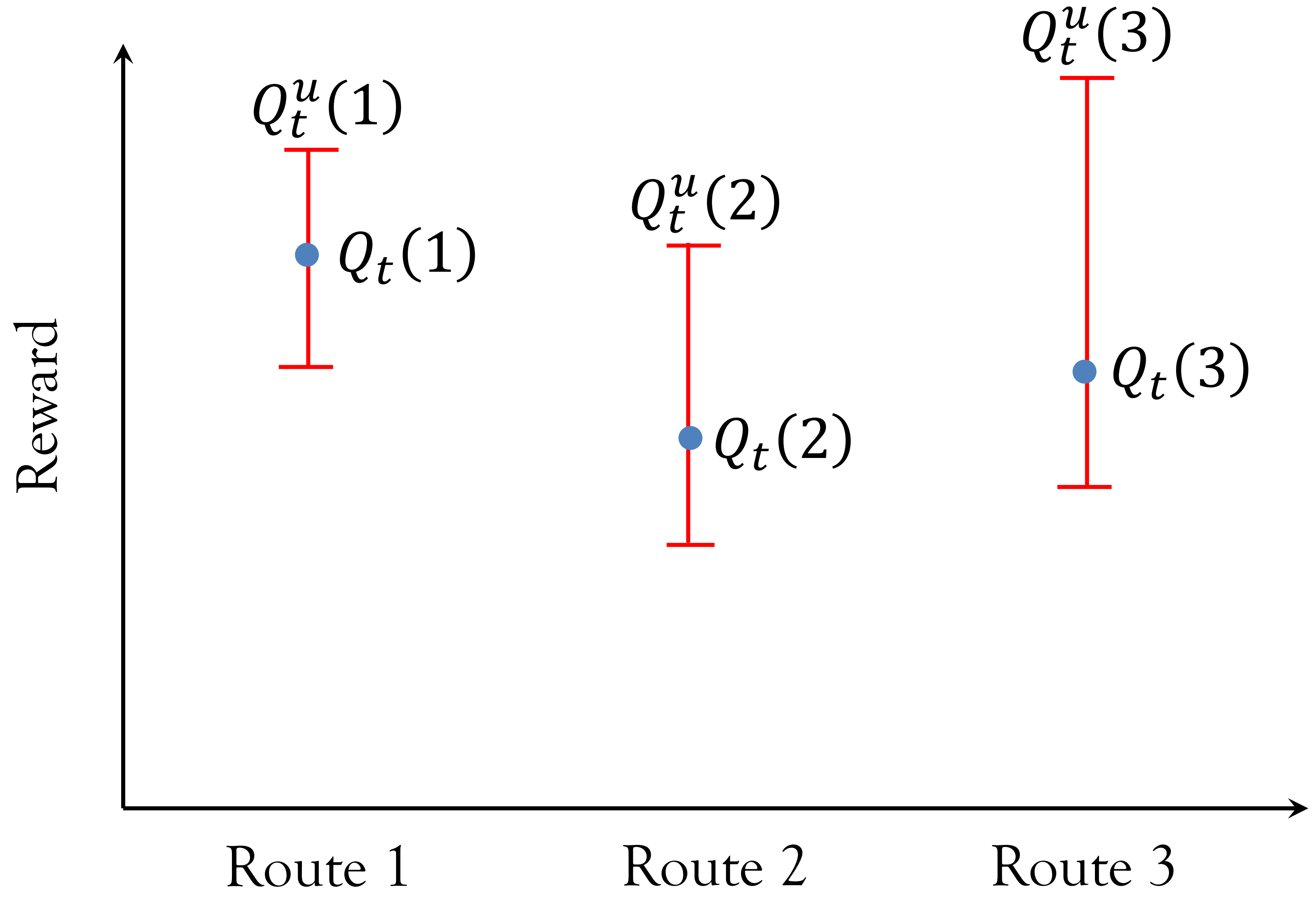
- At time slot t:
  - The agent estimate of $q_*(a)$ is $Q_t(a)$. $Q_t(a)$ for the three routes are shown using the blue dots.

Upper Confidence Bound (UCB) Policy



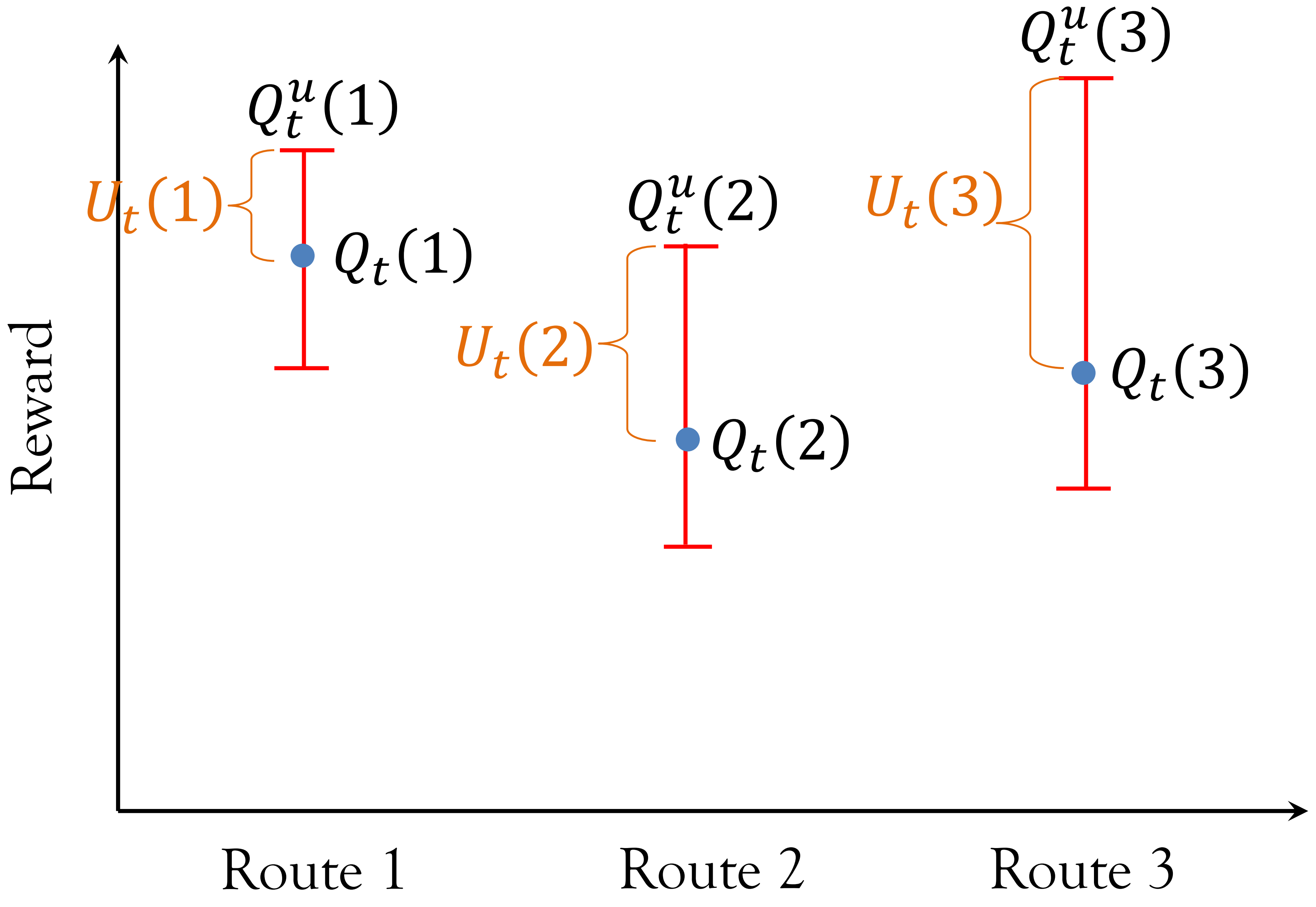
- At time slot t:
  - The agent estimate of $q_*(a)$ is $Q_t(a)$. $Q_t(a)$ for the three routes are shown using the blue dots.
  - $q_*(a)$ may not be equal to $Q_t(a)$. Rather it may lie in a certain **confidence interval** around $Q_t(a)$. This confidence interval is shown using the red lines.
    - ✓ The upper and the lower bounds of the confidence interval for $Q_t(a)$ are denoted as $Q_t^u(a)$ and $Q_t^l(a)$ respectively.

Upper Confidence Bound (UCB) Policy



- At time slot t:
  - The UCB policy works in the principle of "optimism in face of uncertainty". Hence, it chooses the action a that has the maximum upper bound, $Q_t^u(a)$.
    - ✓ Even though $Q_t(1)$ is the highest, UCB policy will not choose Route 1. Rather, it will choose Route 3 because $Q_t^u(3)$ is the highest.

Upper Confidence Bound (UCB) Policy

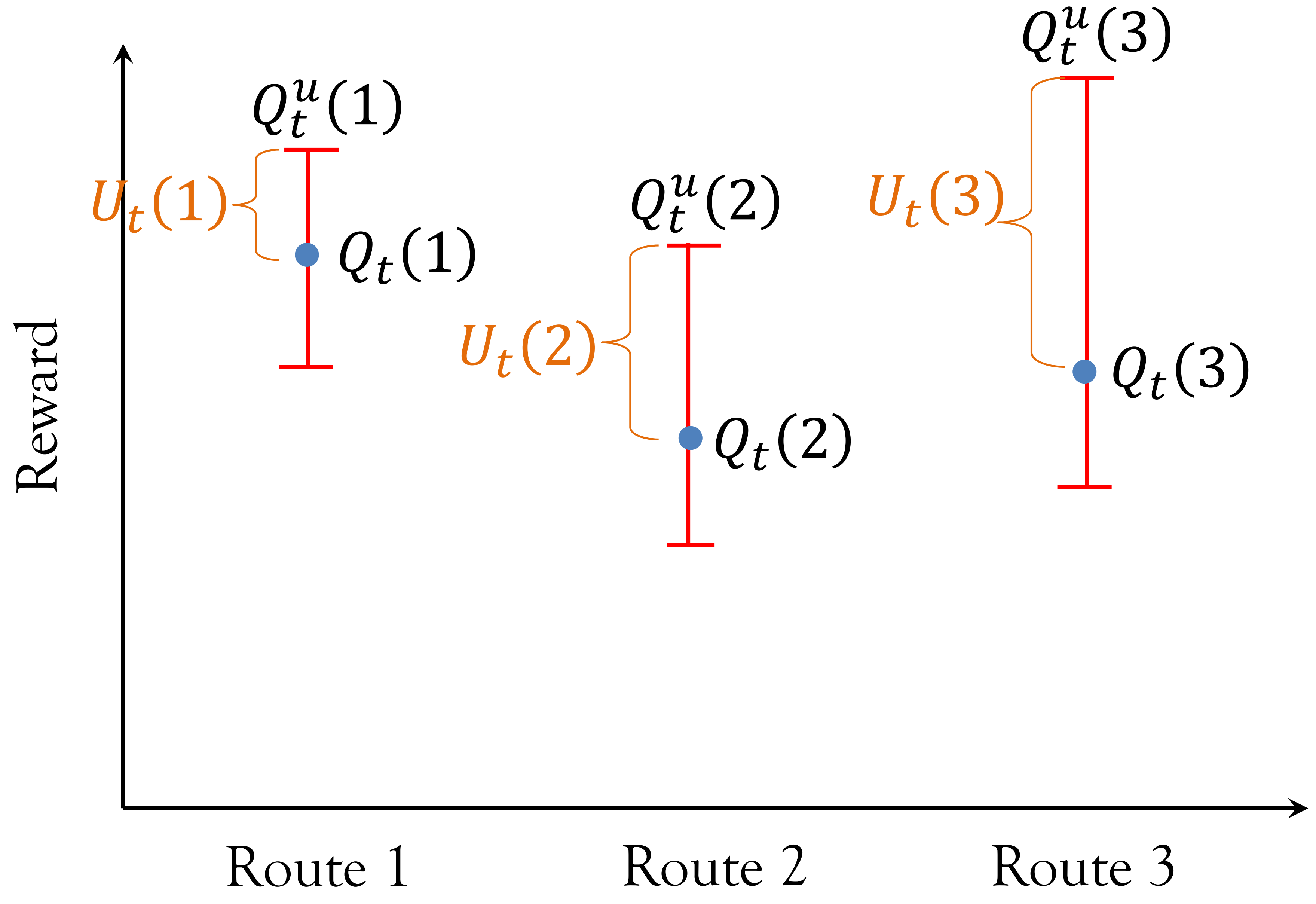


- At time slot t:
  - The UCB policy works in the principle of "optimism in face of uncertainty". Hence, it chooses the action a that has the maximum upper bound, $Q_t^u(a)$.
    - ✓Even though $Q_t(1)$ is the highest, UCB policy will not choose Route 1. Rather, it will choose Route 3 because $Q_t^u(3)$ is the highest.
  - Let, $Q_t^u(a) = Q_t(a) + U_t(a)$ where, $U_t(a) \geq 0$, Then, the UCB policy chooses,

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\left(Q_t(a) + U_t(a)\right)$$

How to find $U_t(a)$?

Upper Confidence Bound (UCB) Policy



- At time slot t:
  - $Q_t^u(a)$ and hence $U_t(a)$ should be chosen such that the probability the action-value $q_*(a)$ is more than $Q_t^u(a)$ is less than δ,

$$P[q_*(a) \geq Q_t(a) + U_t(a)] \leq \delta$$

Upper Confidence Bound (UCB) Policy

Hoeffding's Inequality:

Let $X_1, X_2, \ldots, X_N$ be iid random variable

between [0,1]. Let,

$$\bar{X} = \frac{1}{N}\sum_{n=1}^{N} X_n$$

Then for $u \geq 0$,

$$P[E[X] \geq \bar{X} + u] \leq exp(-2Nu^2)$$

- At time slot t:
  - $Q_t^u(a)$ and hence $U_t(a)$ should be chosen such that the probability the action-value $q_*(a)$ is more than $Q_t^u(a)$ is less than δ,

$$P[q_*(a) \geq Q_t(a) + U_t(a)] \leq \delta$$

  - We find the following analogies of the above equation with the Hoeffding's inequality:
    - ✓ $q_*(a)$ and $E[X]$ are analogous because they are both ensemble mean.
    - ✓ $Q_t(a)$ and $\bar{X}$ are analogous because they are both sample mean.
    - ✓ $N_t(a)$ and N are analogous because they are both the number of samples.

  So we have,

$$P[q_*(a) \geq Q_t(a) + U_t(a)] \leq exp\left(-2N_t(a)\big(U_t(a)\big)^2\right)$$

Upper Confidence Bound (UCB) Policy

- At time slot t:
  - To this end we have the following. We want to find $U_t(a)$ such that,

$$\boxed{P[q_*(a) \geq Q_t(a) + U_t(a)] \leq \delta}$$

  - Using Hoeffding's inequality,

$$P[q_*(a) \geq Q_t(a) + U_t(a)] \leq exp\left(-2N_t(a)\left(U_t(a)\right)^2\right)$$

  - This condition is satisfied if,

$$exp\left(-2N_t(a)\left(U_t(a)\right)^2\right) = \delta$$

Solving the above equation we get,

$$U_t(a) = \sqrt{\frac{-ln(\delta)}{2N_t(a)}}$$

Upper Confidence Bound (UCB) Policy

➢ At time slot t:

- This condition is satisfied if,

$$exp\left(-2N_t(a)\left(U_t(a)\right)^2\right) = \delta$$

Solving the above equation we get,

$$U_t(a) = \sqrt{\frac{-ln(\delta)}{2N_t(a)}}$$

- Substituting the above $U_t(a)$ in the equation in slide 64 we get,

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\left(Q_t(a) + \sqrt{\frac{-ln(\delta)}{2N_t(a)}}\right)$$

- Substituting $\delta = \frac{1}{t^4}$ leads to the UCB policy. Why exactly $\delta = \frac{1}{t^4}$? That is beyond the scope of the proof and hence this only a partial derivation!

Policy Gradient

- The policy gradient approach for contextual bandits has been discussed in the slides of lectures 6, 7, and 8. The policy for MAB is a special case of contextual bandit (as we discussed before, for MAB, the context is the same at all time an hence we can simply ignore the context).

- The policy gradient approach for MAB is also there in pages 37 - 40 of "the book".
  - You can ignore the concept of baseline for the time being.

Homework: Using help from "the book" and the slides of lecture 6, 7, and 8, familiarize yourself with the concept of policy gradient for MAB.

Thank you