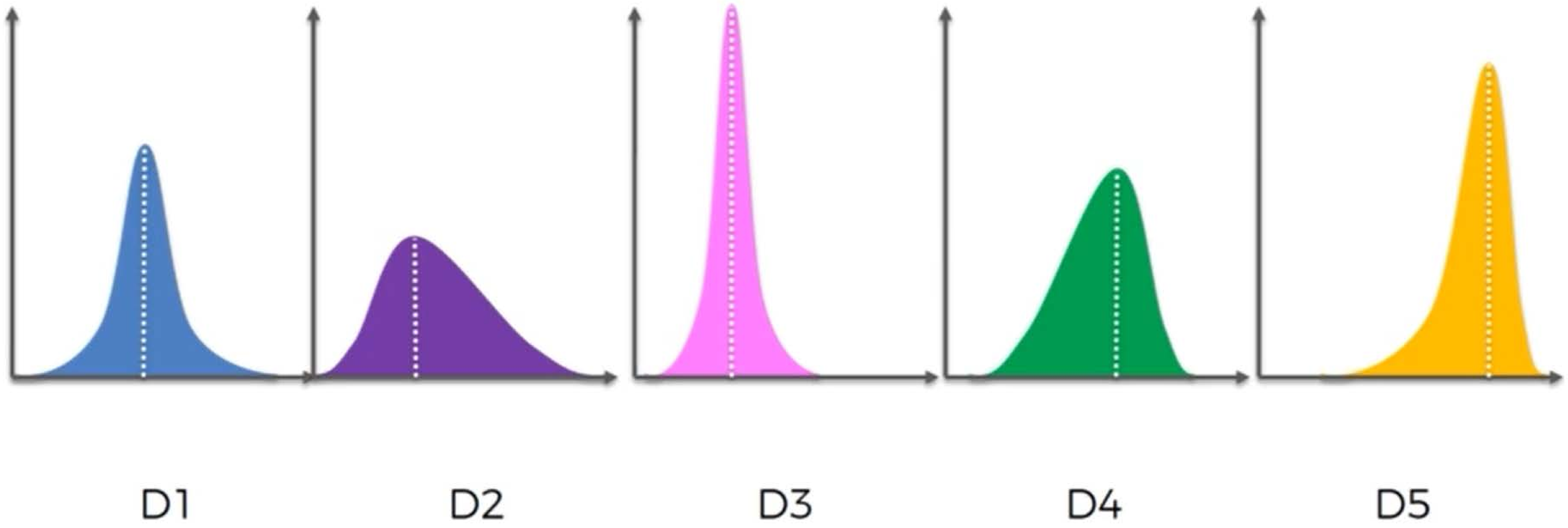
The Multi-Armed Bandit Problem

The Multi-Armed Bandit Problem

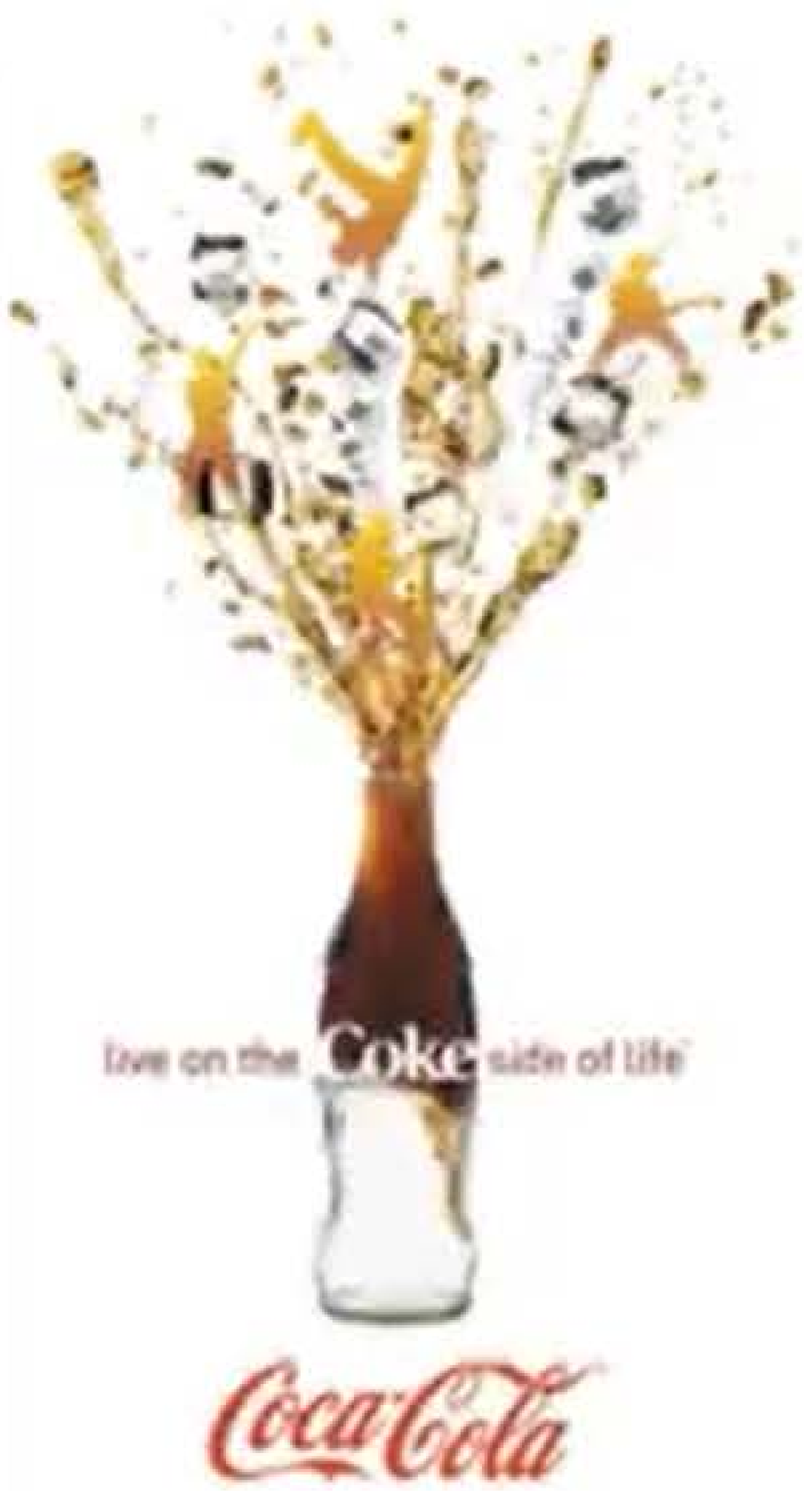
D1 D2 D3 D4 D5

The Multi-Armed Bandit Problem

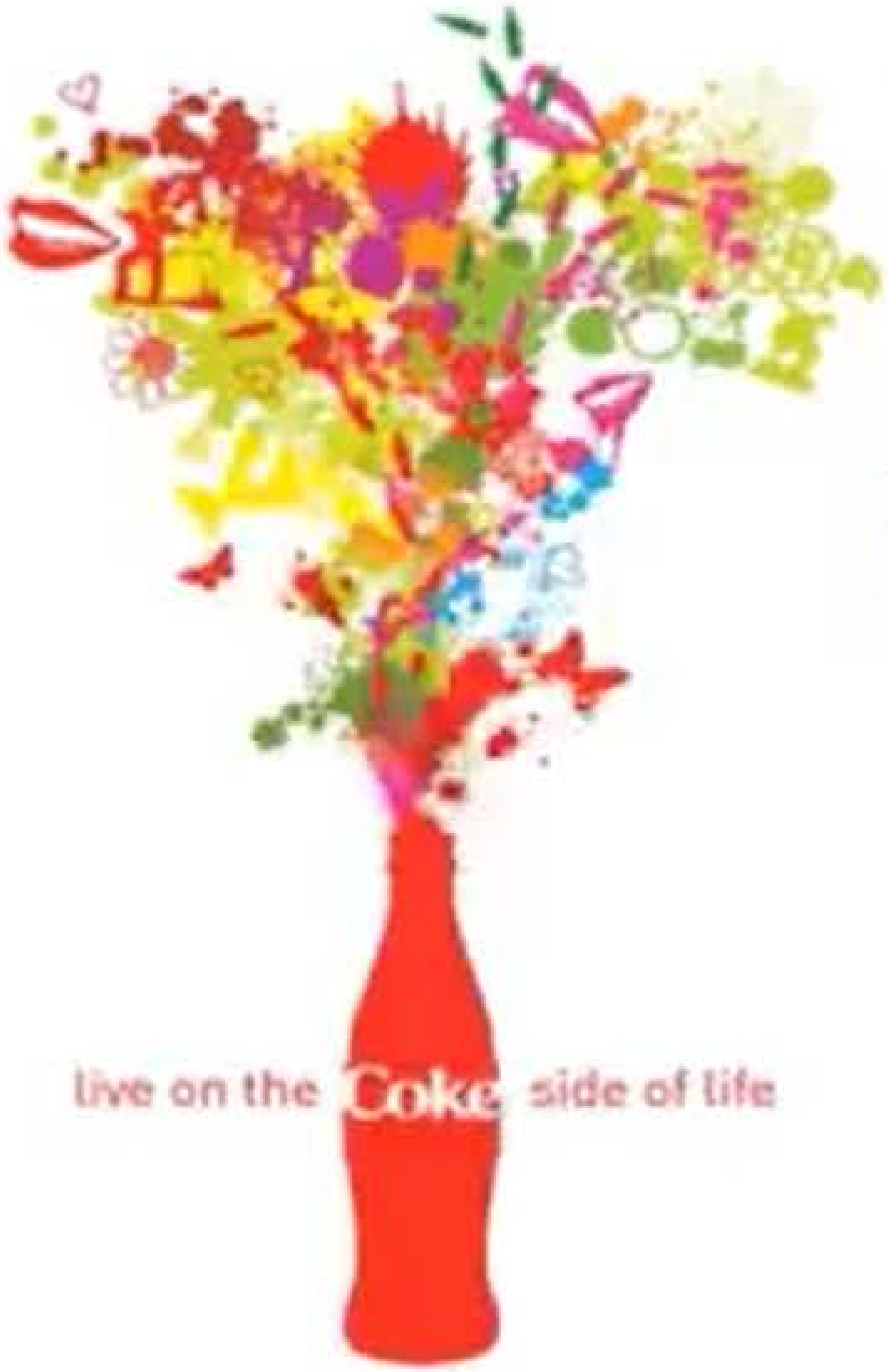


The Multi-Armed Bandit Problem

D1

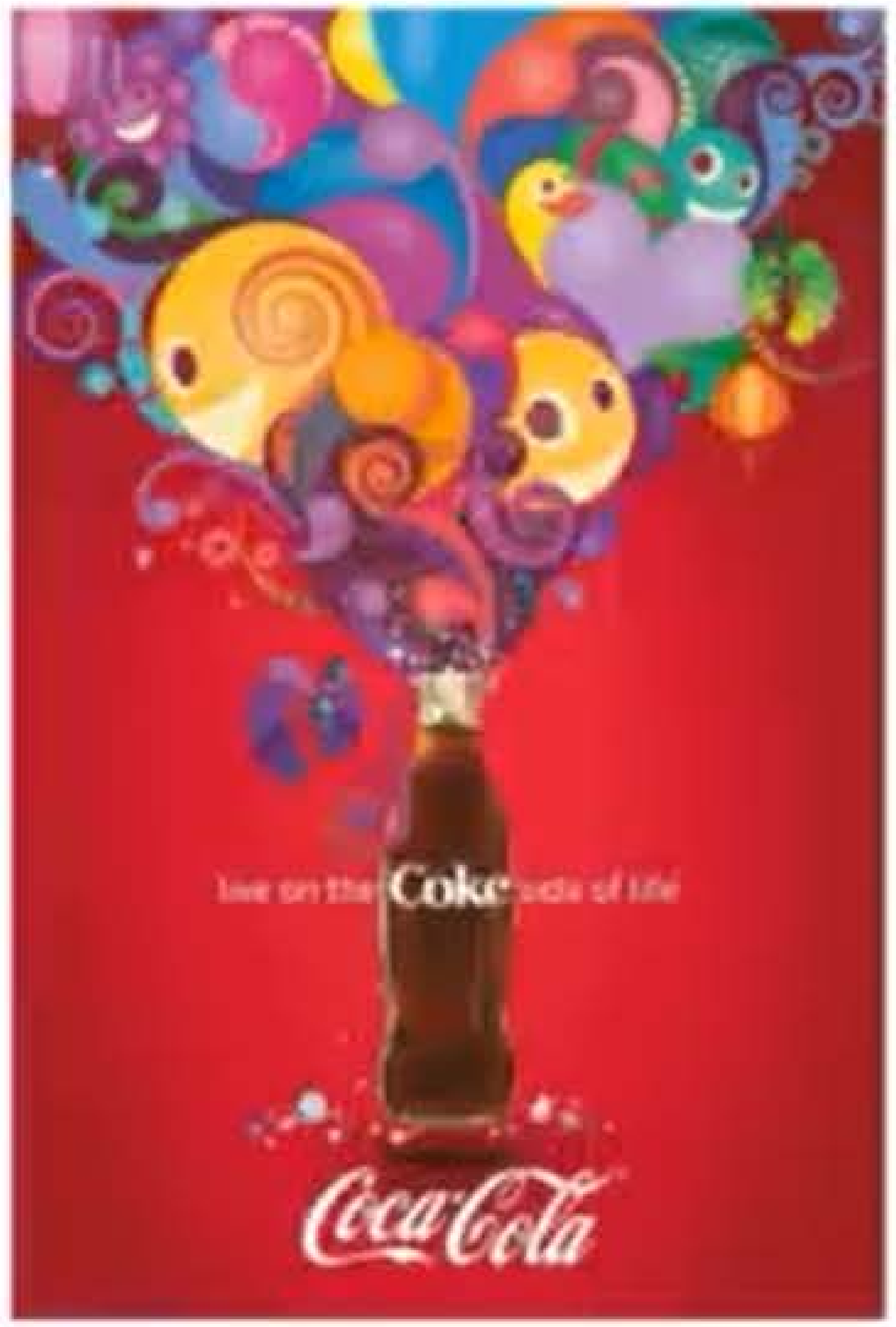


D2



D3

D4



D5

Examples used for educational purposes. No affiliation with Coca-Cola

Upper Confidence Bound Intuition (UCB)

The Multi-Armed Bandit Problem

- We have d arms. For example, arms are ads that we display to users each time they connect to a web page.
- Each time a user connects to this web page, that makes a round.
- At each round n, we choose one ad to display to the user.
- At each round n, ad i gives reward $r_i(n) \in \{0, 1\}$: $r_i(n) = 1$ if the user clicked on the ad i, 0 if the user didn't.
- Our goal is to maximize the total reward we get over many rounds.

Upper Confidence Bound Algorithm

Step 1. At each round n, we consider two numbers for each ad i:

- $N_i(n)$ - the number of times the ad i was selected up to round n,
- $R_i(n)$ - the sum of rewards of the ad i up to round n.

Step 2. From these two numbers we compute:

- the average reward of ad i up to round n

$$\bar{r}_i(n) = \frac{R_i(n)}{N_i(n)}$$

- the confidence interval $[\bar{r}_i(n) - \Delta_i(n), \bar{r}_i(n) + \Delta_i(n)]$ at round n with

$$\Delta_i(n) = \sqrt{\frac{3}{2}\frac{\log(n)}{N_i(n)}}$$

Step 3. We select the ad i that has the maximum UCB $\bar{r}_i(n) + \Delta_i(n)$.

Upper Confidence Bound Algorithm

D1

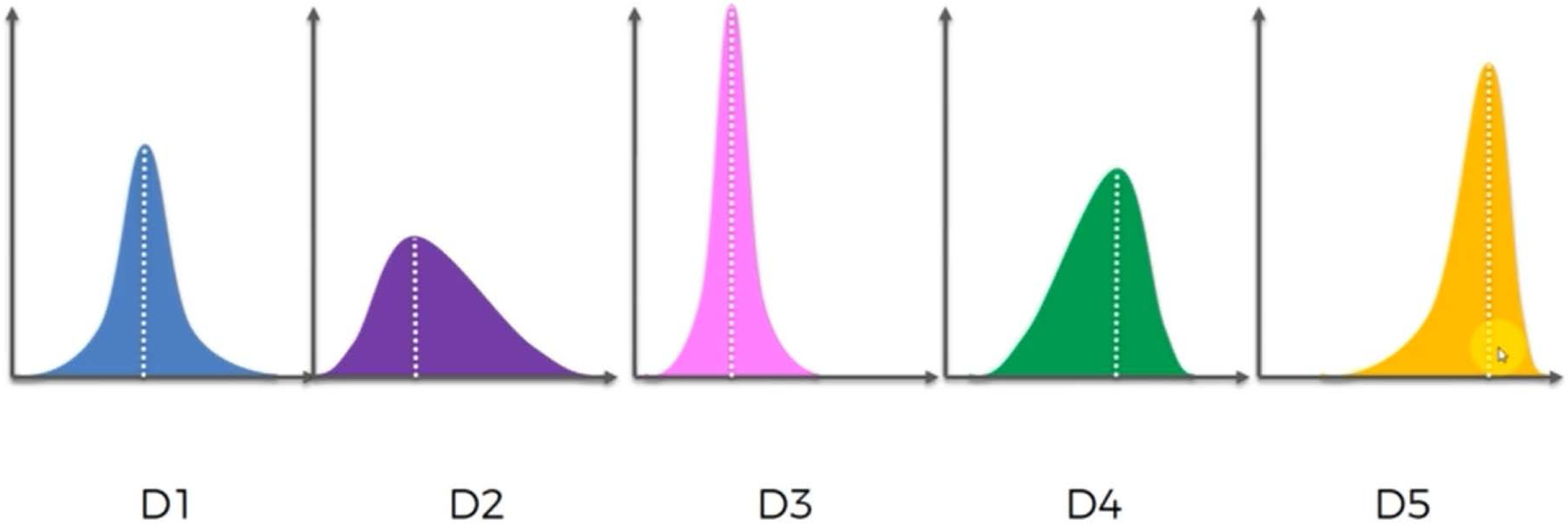
D2

D3

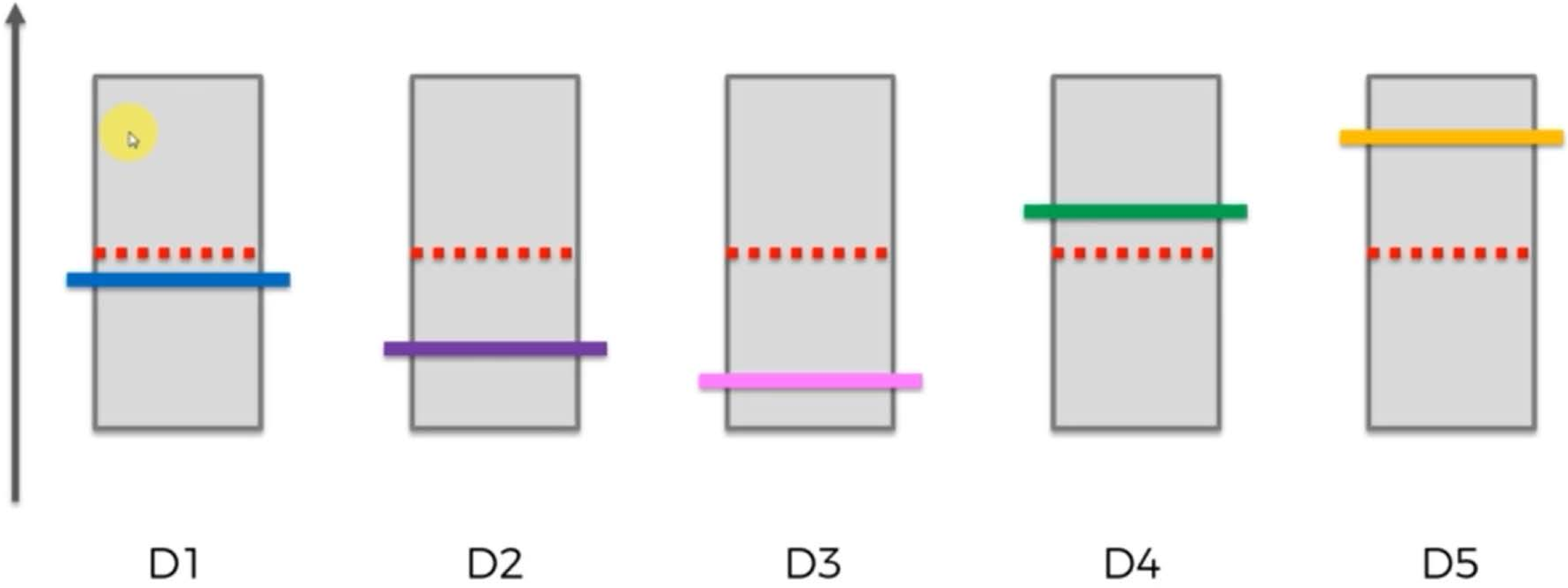
D4

D5

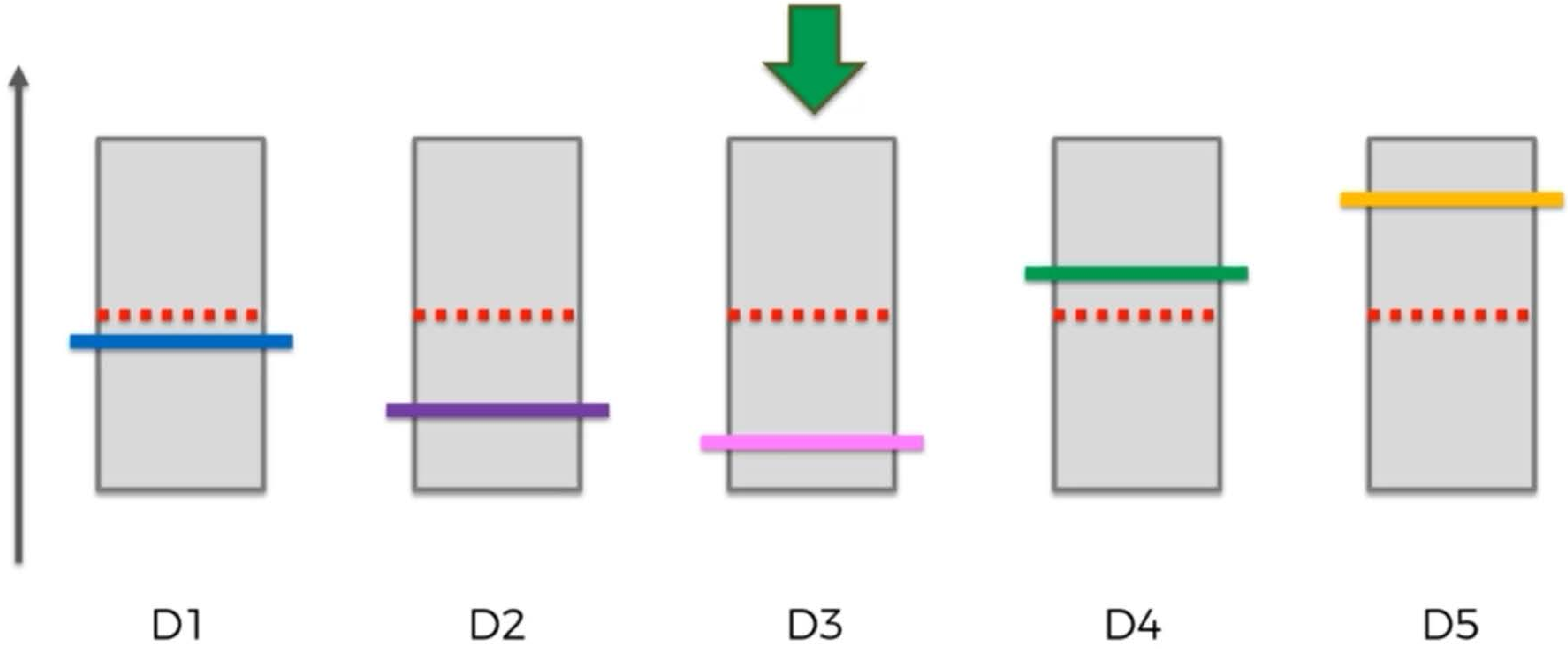
Upper Confidence Bound Algorithm



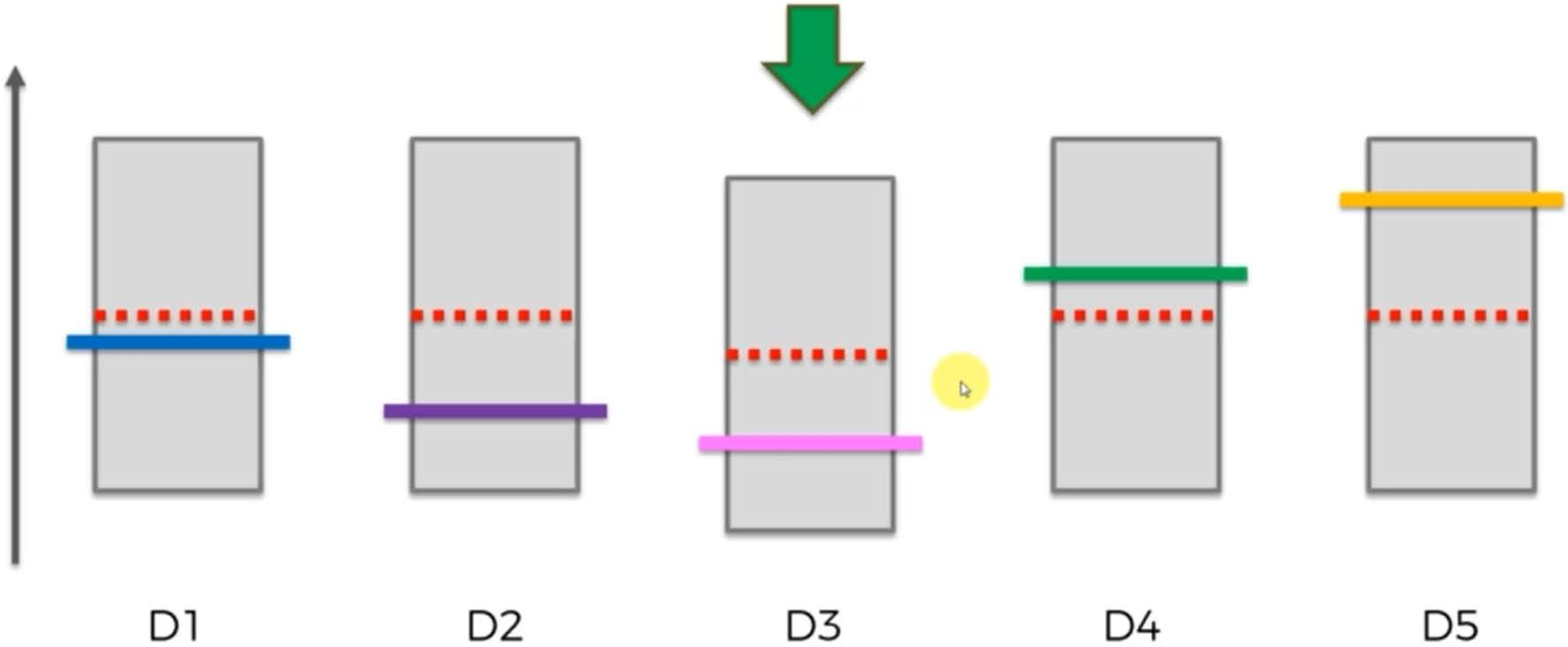
Upper Confidence Bound Algorithm



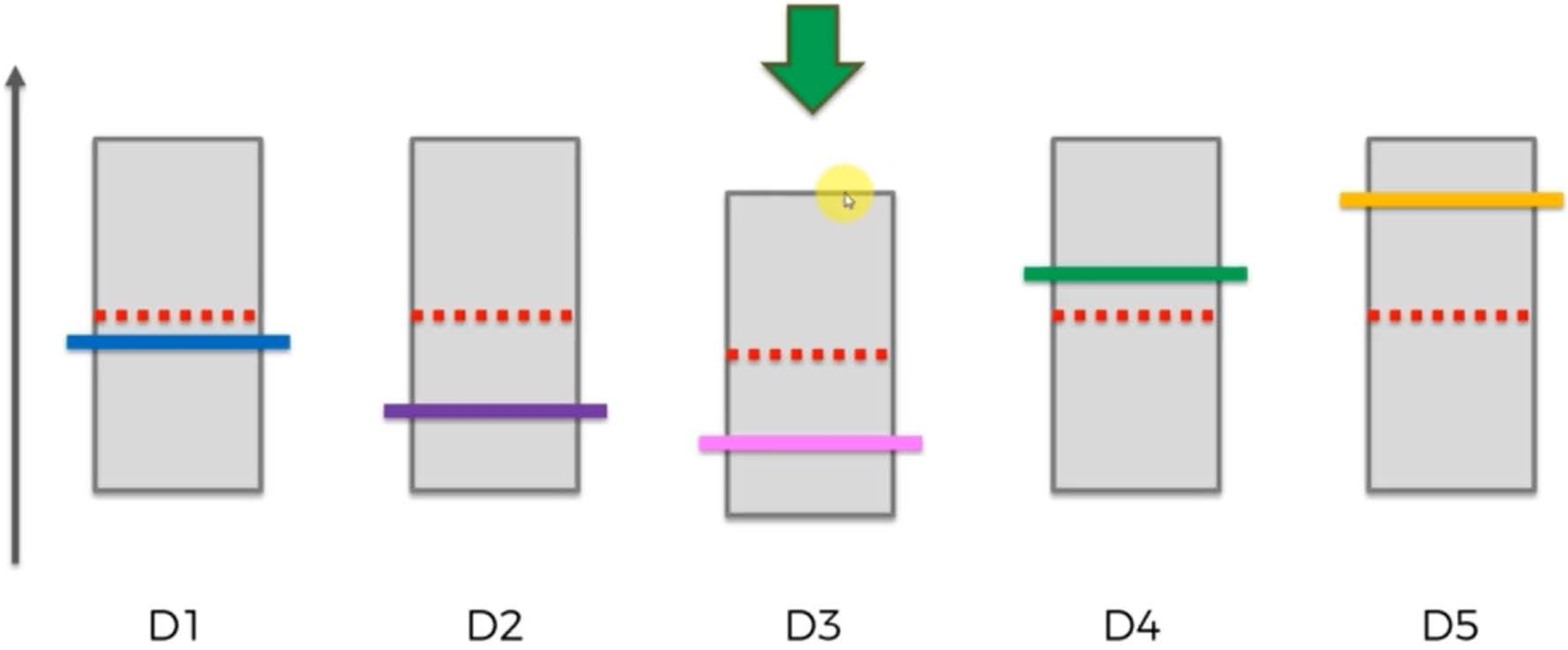
Upper Confidence Bound Algorithm



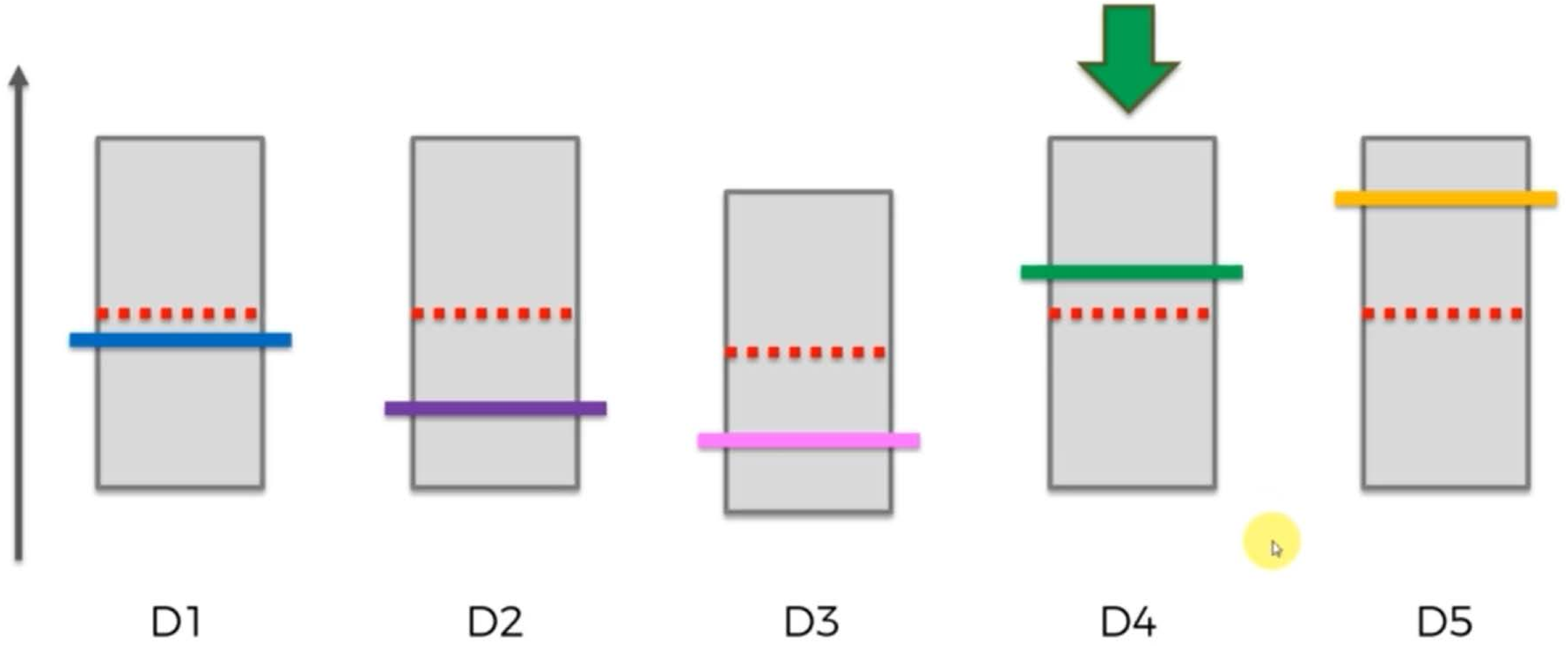
Upper Confidence Bound Algorithm



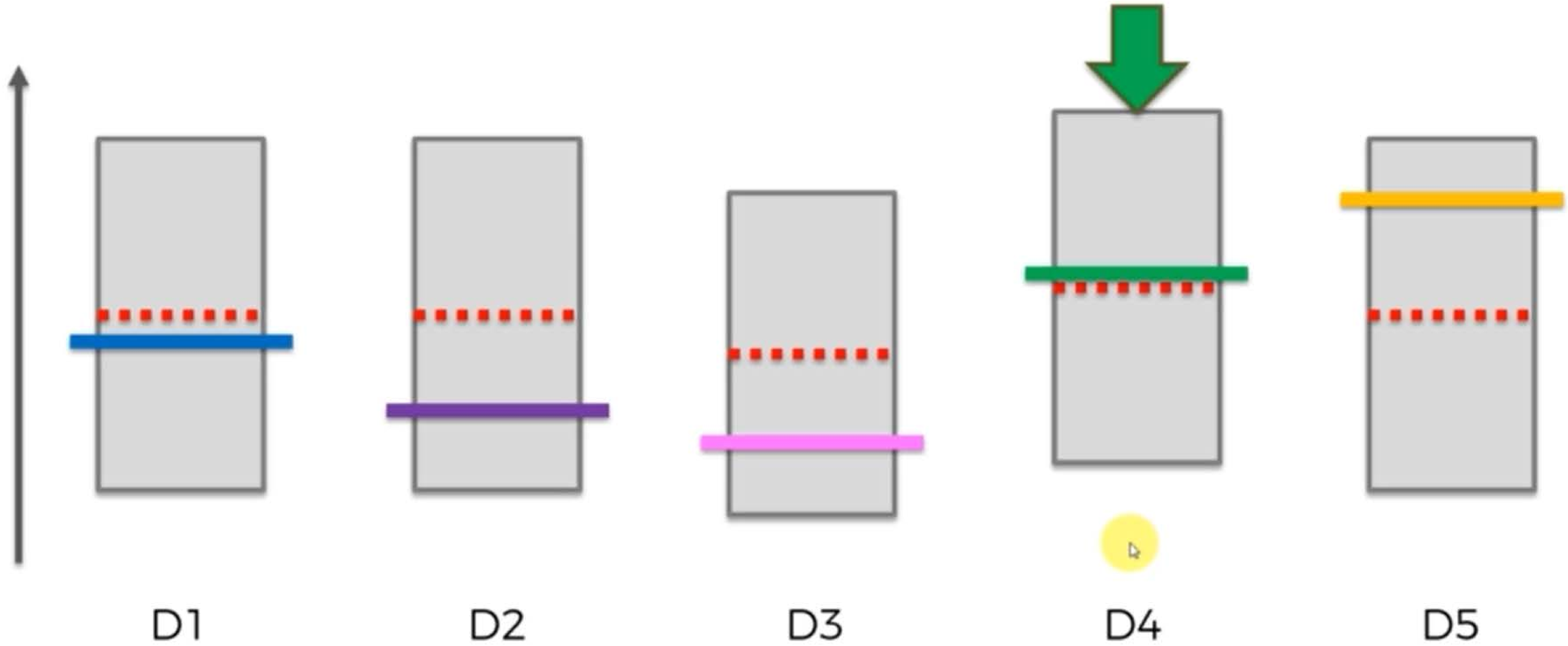
Upper Confidence Bound Algorithm



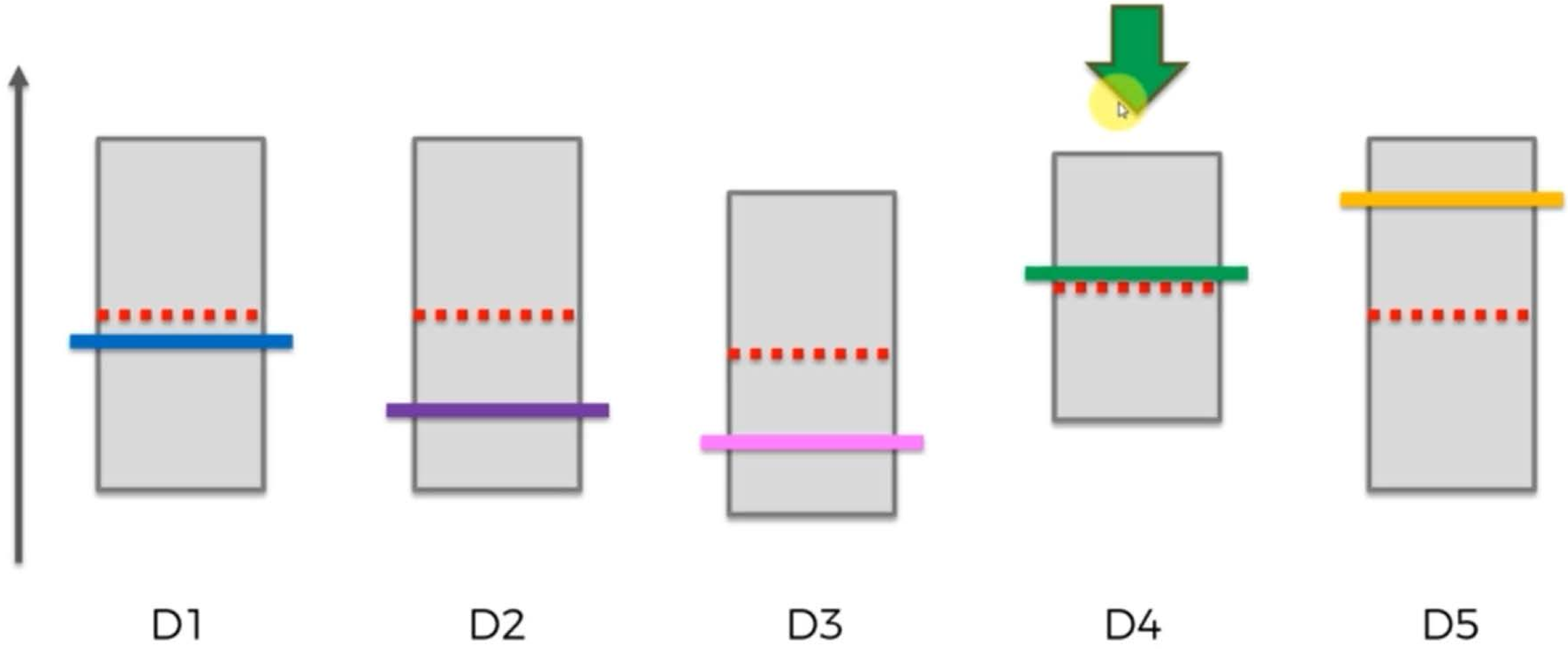
Upper Confidence Bound Algorithm



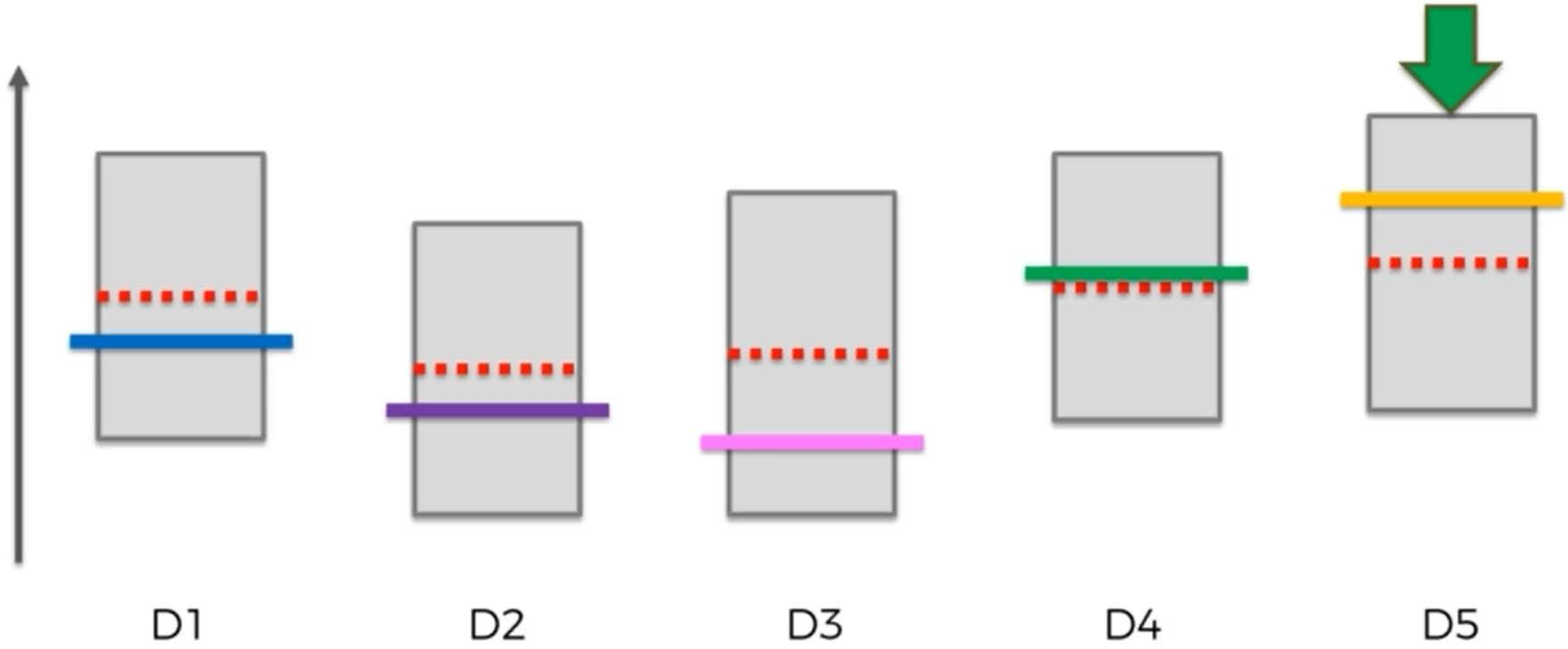
Upper Confidence Bound Algorithm



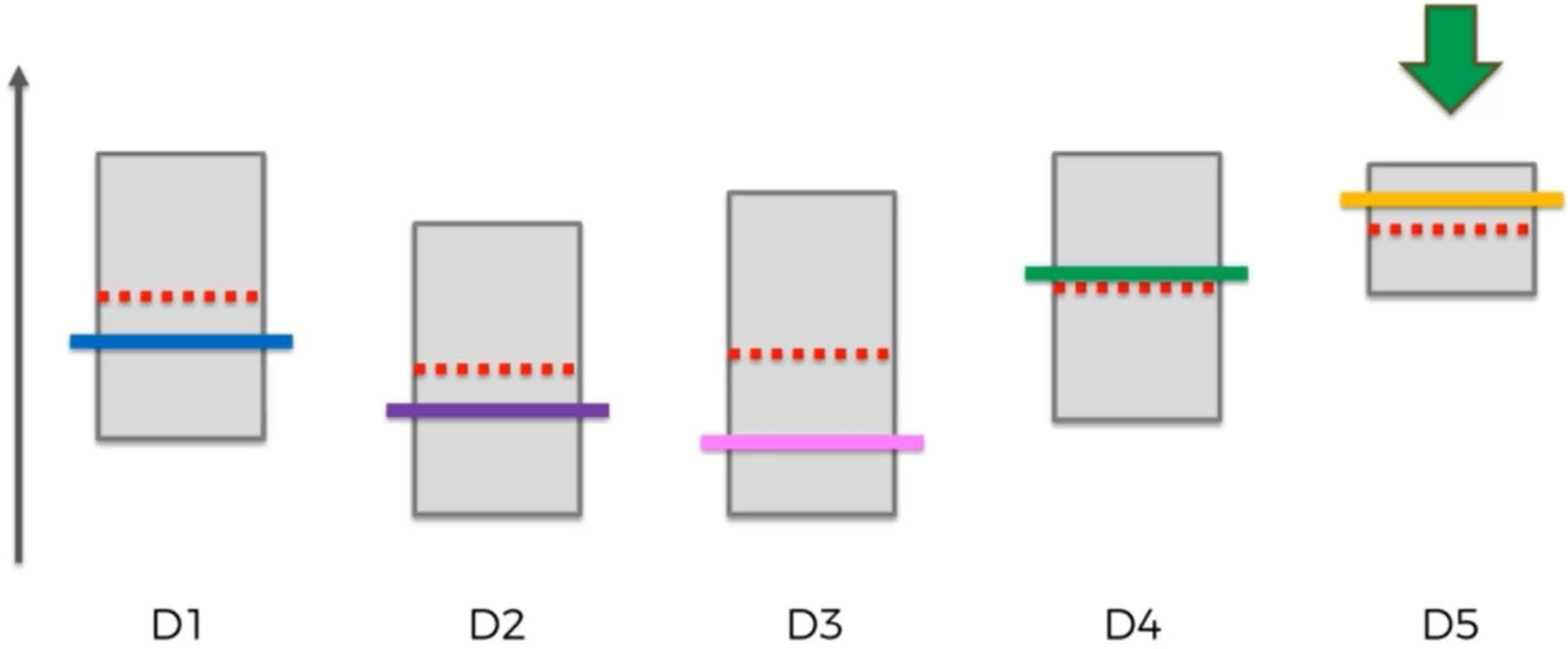
Upper Confidence Bound Algorithm



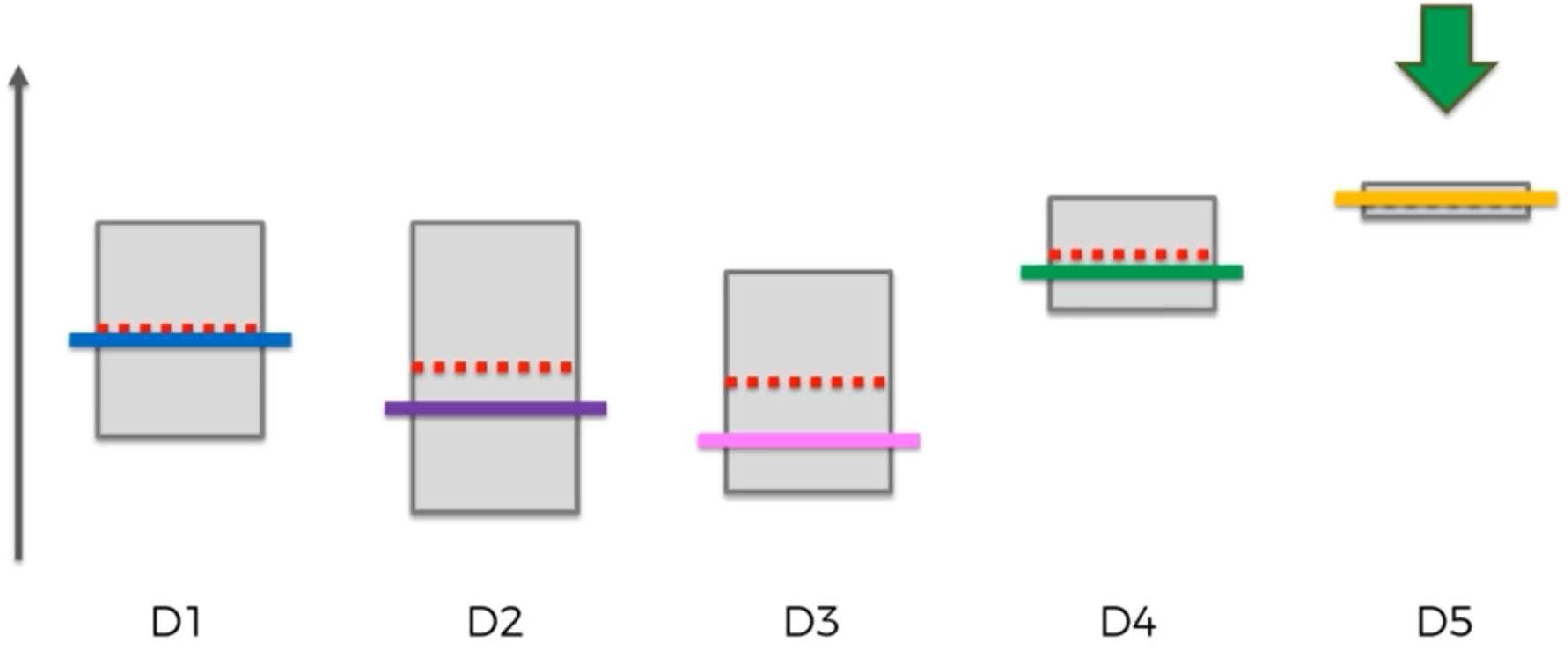
Upper Confidence Bound Algorithm



Upper Confidence Bound Algorithm



Upper Confidence Bound Algorithm



jupyter upper_confidence_bound Last Checkpoint: 21 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

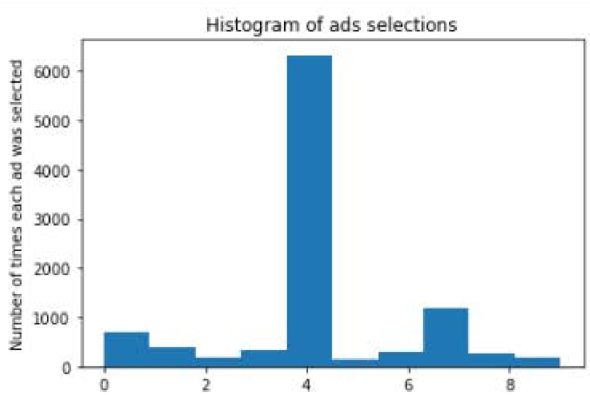
Trusted | Python 3

Run Markdown

In [7]:
```
import math
N = 10000
d = 10
ads_selected = []
numbers_of_selections = [0] * d
sums_of_rewards = [0] * d
total_reward = 0
for n in range(0, N):
    ad = 0
    max_upper_bound = 0
    for i in range(0, d):
        if (numbers_of_selections[i] > 0):
            average_reward = sums_of_rewards[i] / numbers_of_selections[i]
            delta_i = math.sqrt(3/2 * math.log(n + 1) / numbers_of_selections[i])
            upper_bound = average_reward + delta_i
        else:
            upper_bound = 1e400
        if upper_bound > max_upper_bound:
            max_upper_bound = upper_bound
            ad = i
    ads_selected.append(ad)
    numbers_of_selections[ad] = numbers_of_selections[ad] + 1
    reward = dataset.values[n, ad]
    sums_of_rewards[ad] = sums_of_rewards[ad] + reward
    total_reward = total_reward + reward
```

Visualising the results

In [8]:
```
plt.hist(ads_selected)
plt.title('Histogram of ads selections')
plt.xlabel('Ads')
plt.ylabel('Number of times each ad was selected')
plt.show()
```



Thompson Sampling Algorithm Intuition

Bayesian Inference

- Ad i gets rewards $\mathbf{y}$ from Bernoulli distribution $p(\mathbf{y}|\theta_i) \sim \mathcal{B}(\theta_i)$.
- θ_i is unknown but we set its uncertainty by assuming it has a uniform distribution $p(\theta_i) \sim \mathcal{U}([0,1])$, which is the prior distribution.
- Bayes Rule: we approach θ_i by the posterior distribution

$$\underbrace{p(\theta_i|\mathbf{y})}_{\text{posterior distribution}} = \frac{p(\mathbf{y}|\theta_i)p(\theta_i)}{\int p(\mathbf{y}|\theta_i)p(\theta_i)d\theta_i} \propto \underbrace{p(\mathbf{y}|\theta_i)}_{\text{likelihood function}} \times \underbrace{p(\theta_i)}_{\text{prior distribution}}$$

- We get $p(\theta_i|\mathbf{y}) \sim \beta(\text{number of successes} + 1, \text{number of failures} + 1)$
- At each round n we take a random draw $\theta_i(n)$ from this posterior distribution $p(\theta_i|\mathbf{y})$, for each ad i.
- At each round n we select the ad i that has the highest $\theta_i(n)$.

Thompson Sampling Algorithm

Step 1. At each round n, we consider two numbers for each ad i:

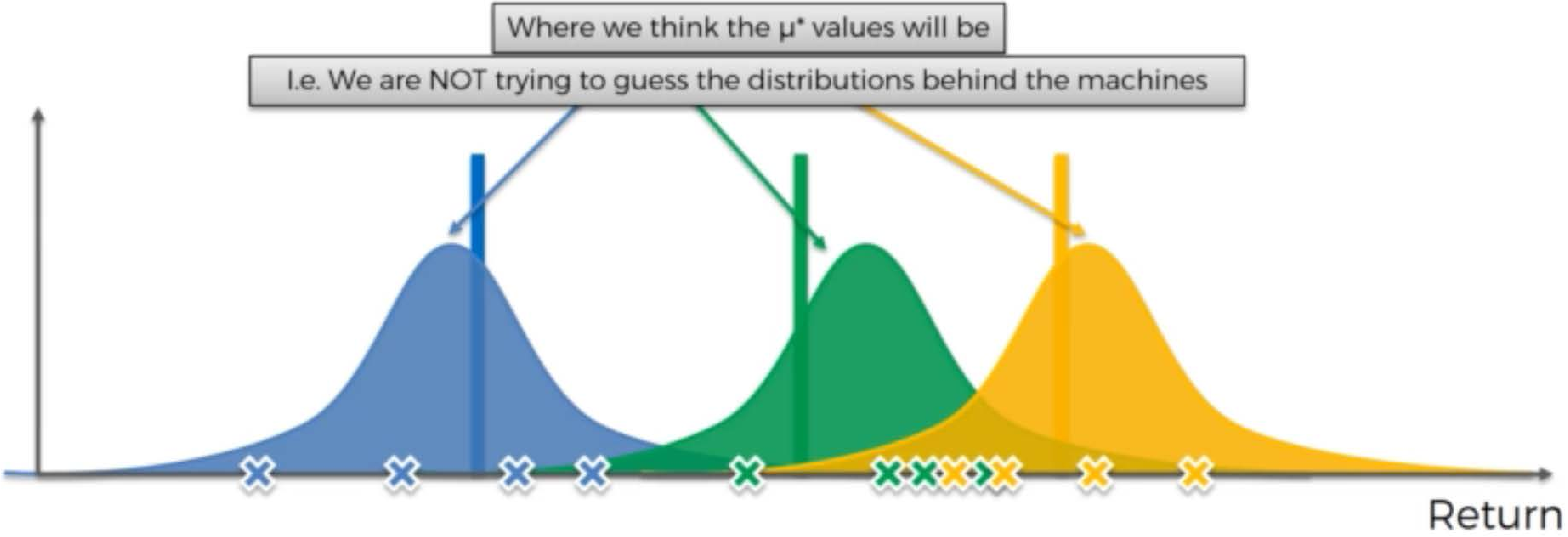
- $N_i^1(n)$ - the number of times the ad i got reward 1 up to round n,
- $N_i^0(n)$ - the number of times the ad i got reward 0 up to round n.

Step 2. For each ad i, we take a random draw from the distribution below:

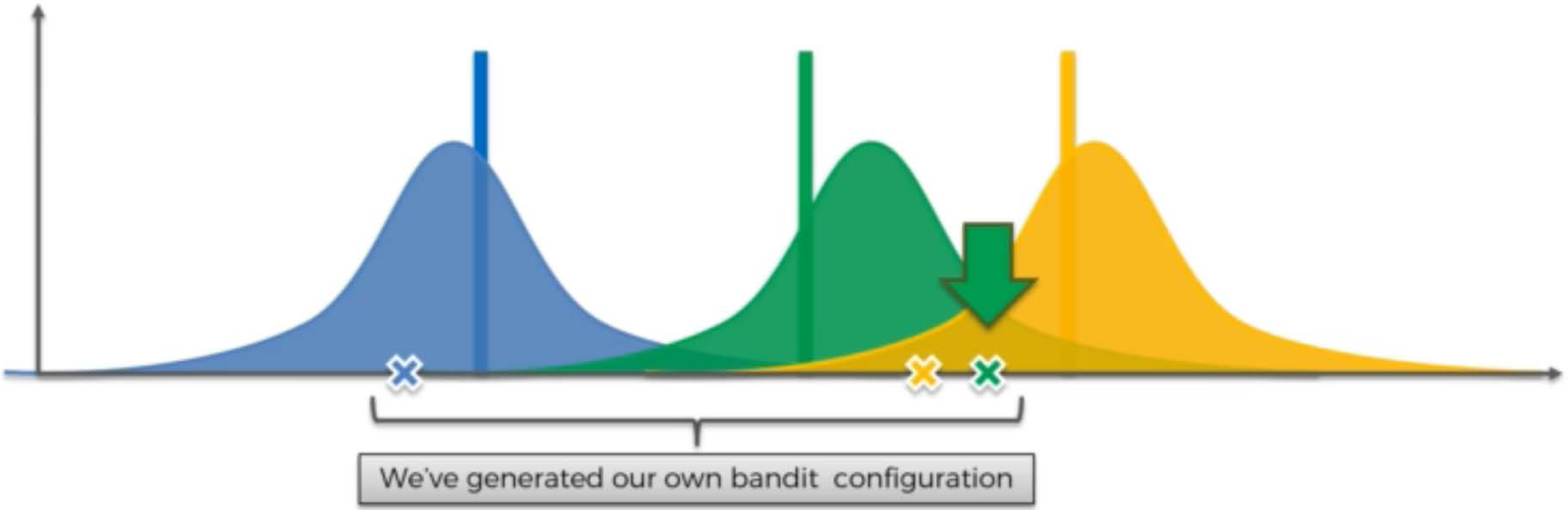
$$\theta_i(n) = \beta(N_i^1(n) + 1, N_i^0(n) + 1)$$

Step 3. We select the ad that has the highest $\theta_i(n)$.

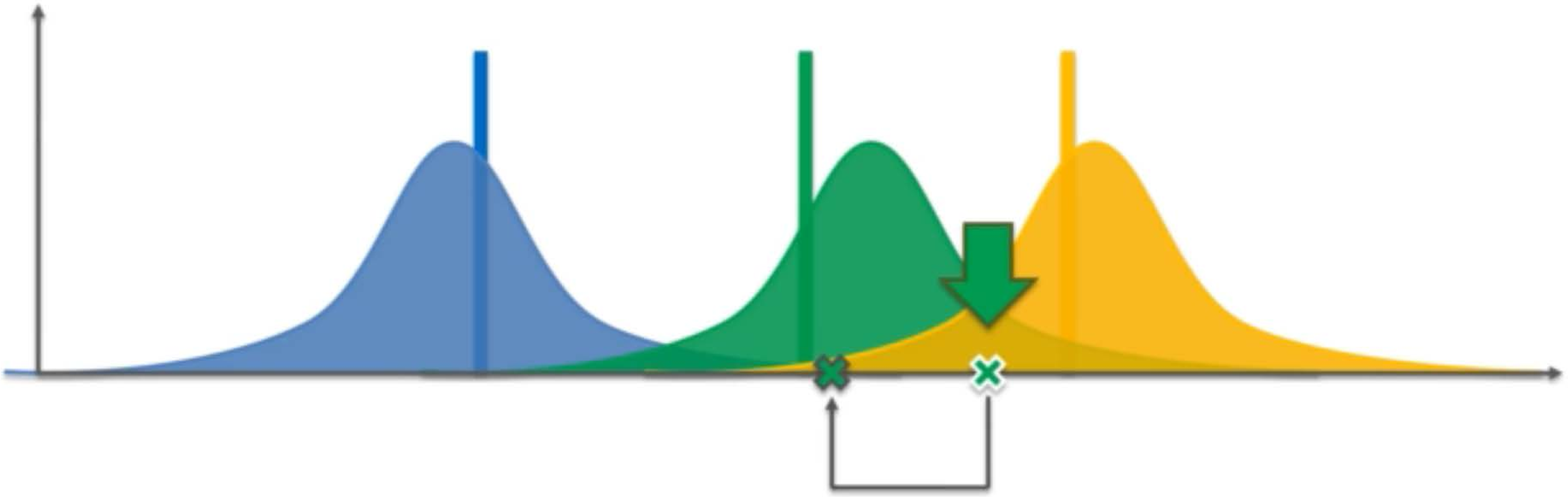
Thompson Sampling Algorithm



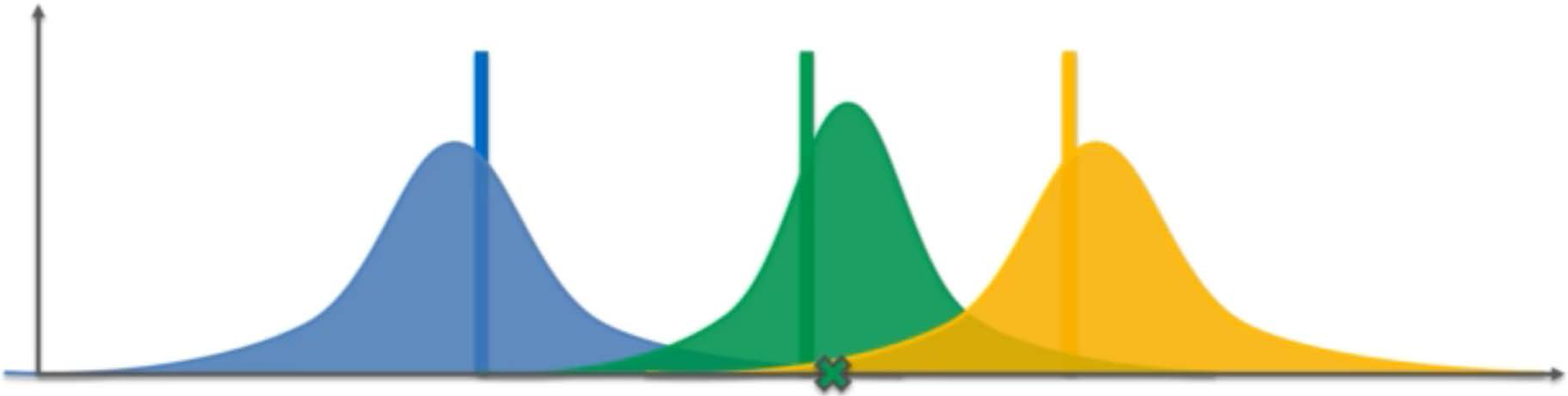
Thompson Sampling Algorithm



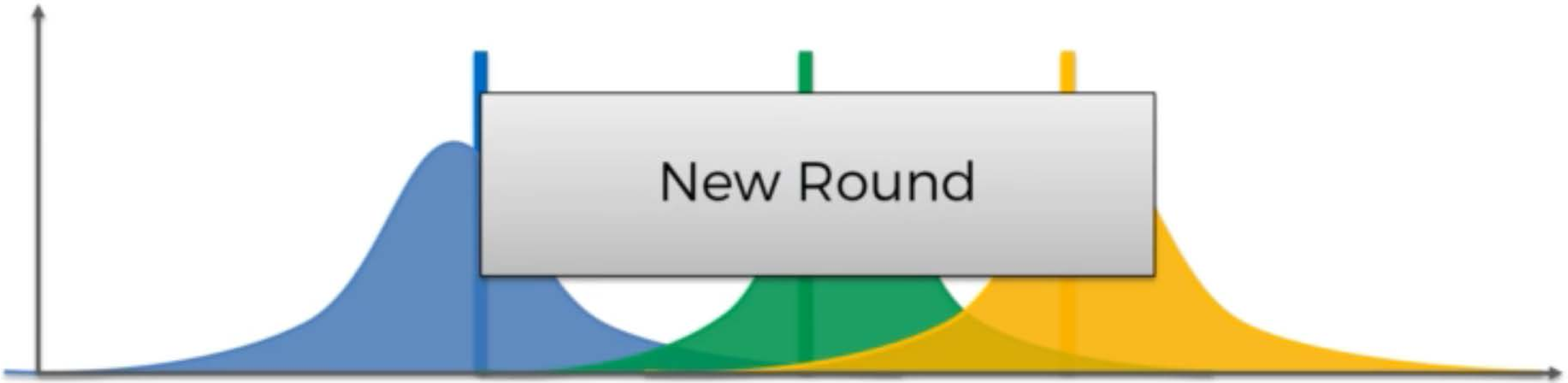
Thompson Sampling Algorithm



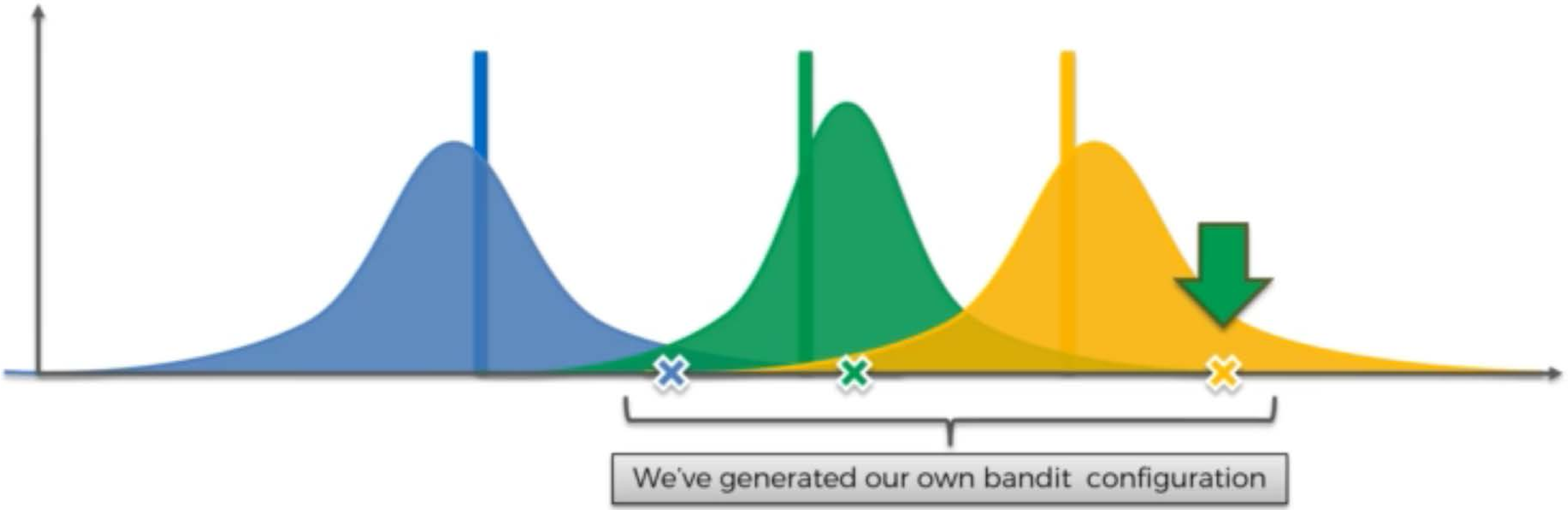
Thompson Sampling Algorithm



Thompson Sampling Algorithm



Thompson Sampling Algorithm



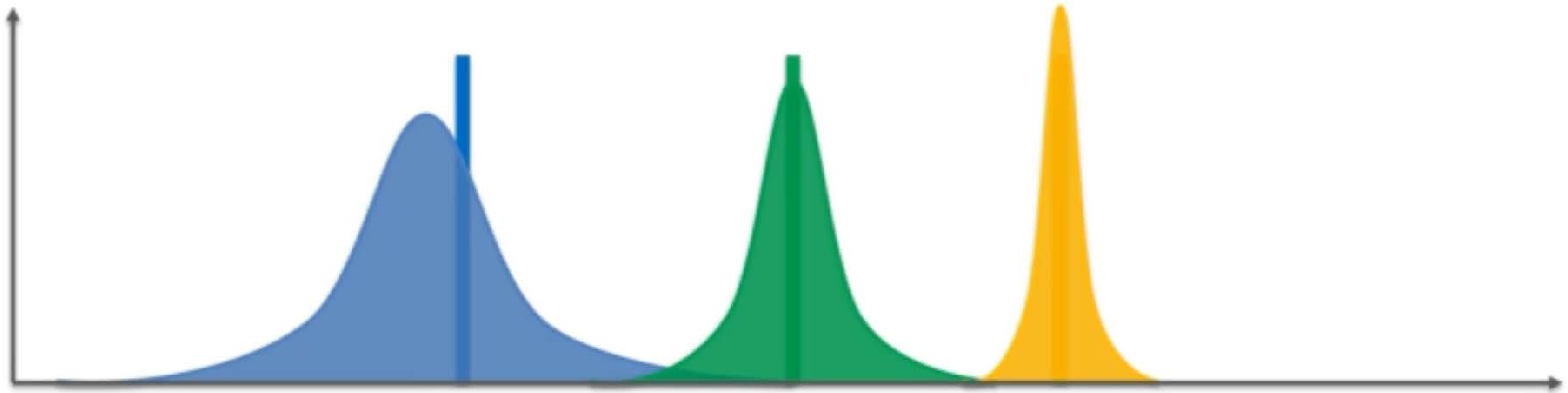
Thompson Sampling Algorithm

Thompson Sampling Algorithm

Thompson Sampling Algorithm

And so on...

Thompson Sampling Algorithm



jupyter thompson_sampling (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3

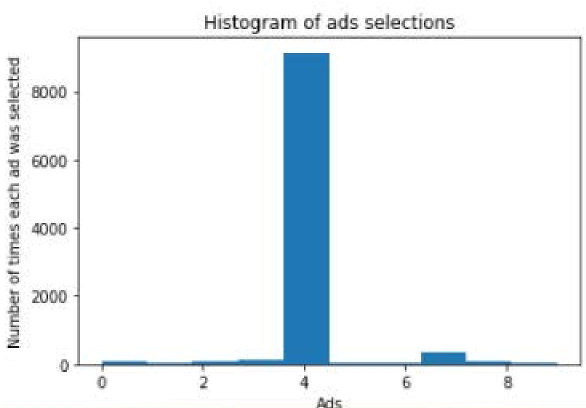
Run Markdown

Implementing Thompson Sampling

In [0]:
```
import random
N = 10000
d = 10
ads_selected = []
numbers_of_rewards_1 = [0] * d
numbers_of_rewards_0 = [0] * d
total_reward = 0
for n in range(0, N):
    ad = 0
    max_random = 0
    for i in range(0, d):
        random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, numbers_of_rewards_0[i] + 1)
        if random_beta > max_random:
            max_random = random_beta
            ad = i
    ads_selected.append(ad)
    reward = dataset.values[n, ad]
    if reward == 1:
        numbers_of_rewards_1[ad] = numbers_of_rewards_1[ad] + 1
    else:
        numbers_of_rewards_0[ad] = numbers_of_rewards_0[ad] + 1
    total_reward = total_reward + reward
```

Visualising the results - Histogram

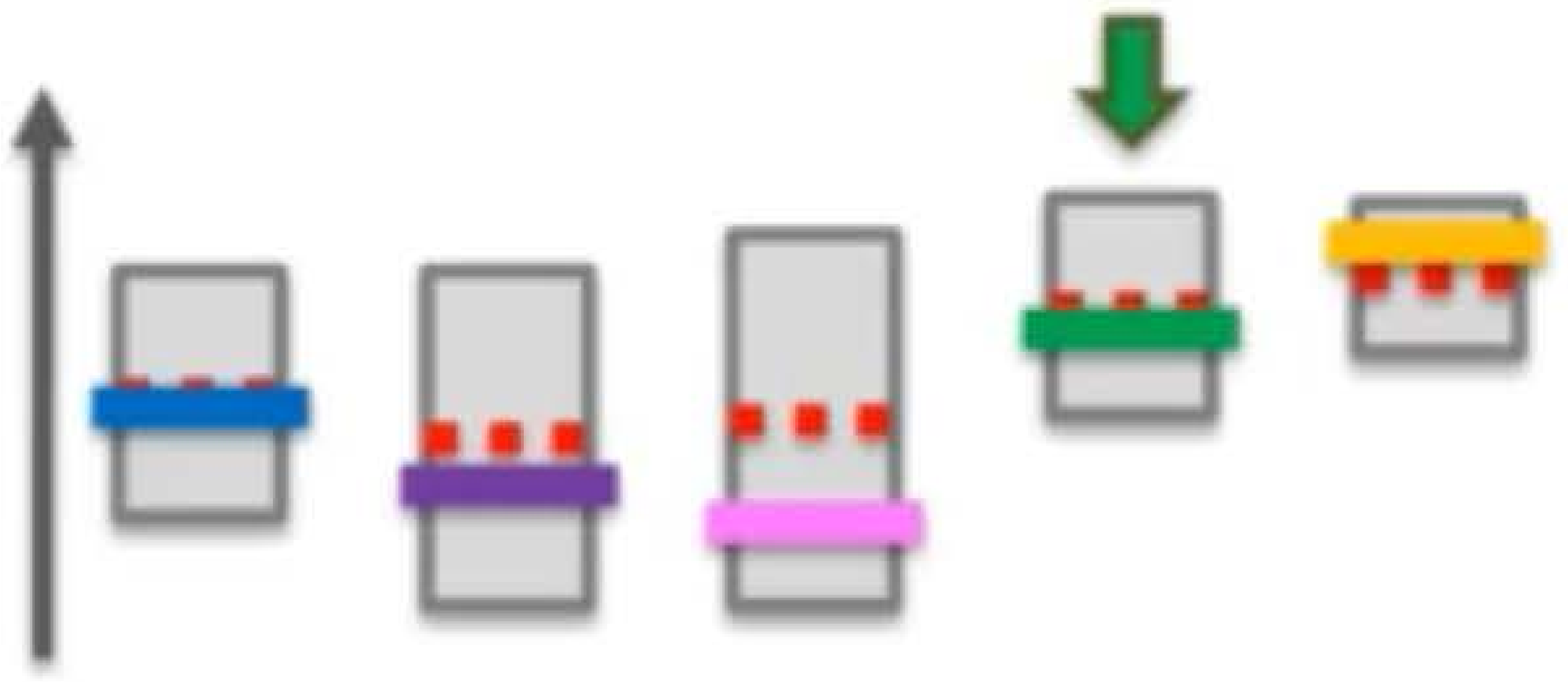
In [4]:
```
plt.hist(ads_selected)
plt.title('Histogram of ads selections')
plt.xlabel('Ads')
plt.ylabel('Number of times each ad was selected')
plt.show()
```



UCB vs Thompson Sampling

Thompson Sampling Algorithm

UCB



- Deterministic
- Requires update at every round

Thompson Sampling

- Probabilistic
- Can accommodate delayed feedback
- Better empirical evidence