

Lecture 10

Reinforcement Learning

AI/ML Foundation Course with Python

Copyright © 2018 Ankita Sinha. All Rights Reserved



Agent



No chocolate
if reward is
not achieved



Environment



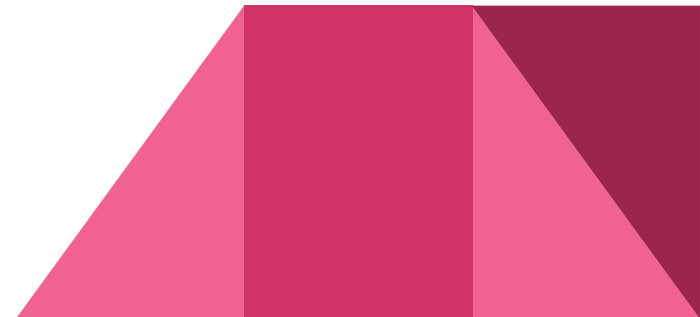
Reward



- ❑ **Agent:** Intelligent programs
- ❑ **Environment:** External condition
- ❑ **Policy:**
 - ❑ Defines the agent's behavior at a given time
 - ❑ A mapping from states to actions
 - ❑ Lookup tables or simple function

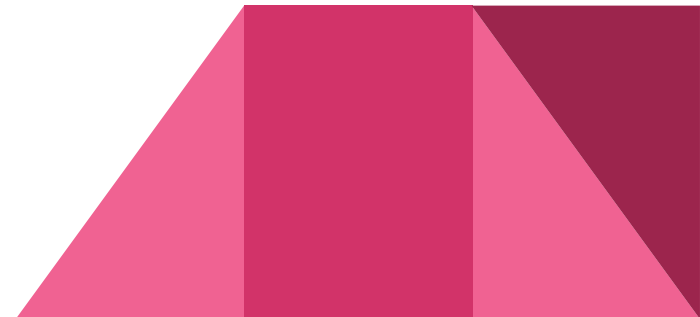
REINFORCEMENT LEARNING

- TRIAL AND ERROR
- FEEDBACK FROM ENVIRONMENT
- CLOSE INTERACTION WITH THE ENVIRONMENT
- USER AGENT RECEIVES FEEDBACK BASED ON THE CHANGES MADE TO THE ENVIRONMENT
- STOCHASTIC ENVIRONMENT
(RESULTS MAY VARY EVERY TIME)



APPLICATION

- GAME PLAYING
 - BACKGAMMON *(world's best player was beaten by RL Agent)*
 - ATARI GAMES
- AUTONOMOUS AGENTS
 - Robot Navigation
- Adaptive Control
 - Helicopter Pilot - Cockpit Simulator
- Intelligent Tutoring Systems
- Self Driving Cars ???



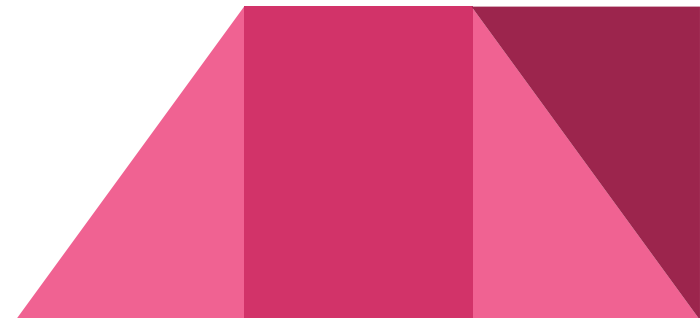
Model-free and Model-based Learning

■ **Model-based learning**

- Learn from the model instead of interacting with the world
- Can visit arbitrary parts of the model
- Also called indirect methods
- E.g. Dynamic Programming

■ **Model-free learning**

- Sample Reward and transition function by interacting with the world
- Also called direct methods



ML Paradigms

- **Supervised Learning**
 - Learn an input and output map
 - Classification: categorical output
 - Regression: continuous output
 - **Unsupervised Learning**
 - Discover patterns in the data
 - Clustering: cohesive grouping
 - Association: frequent cooccurrence
- **Reinforcement Learning**
- ◆ **Learning Control**



REWARDS/PENALTY

❖ WIN / LOSE

❖ POSITIVE / NEUTRAL / NEGATIVE

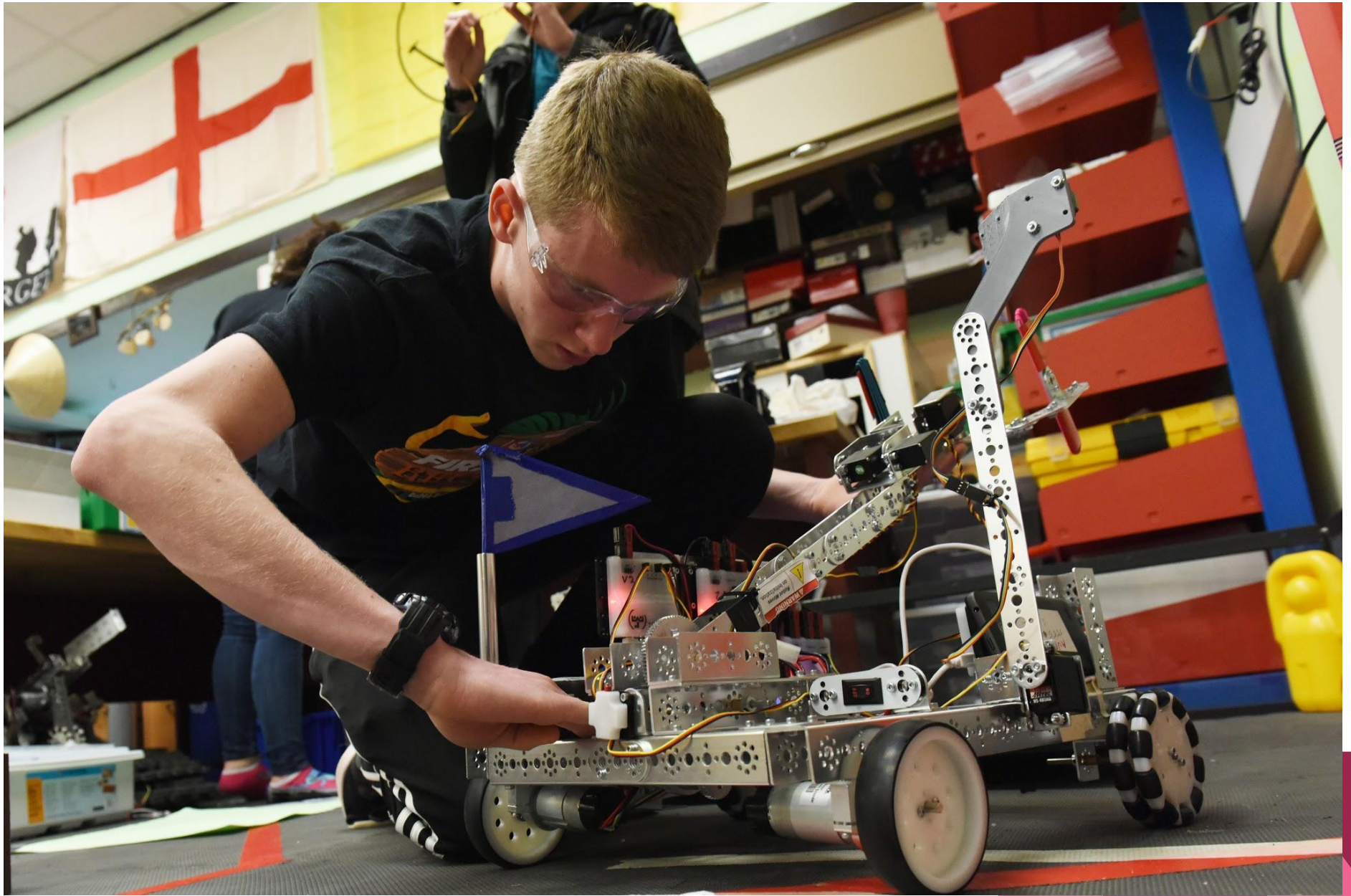
Agent Learns : What parts of its action sequence truly help him get rewarded.

May take hours of Training Time !!!

Think of a Game?

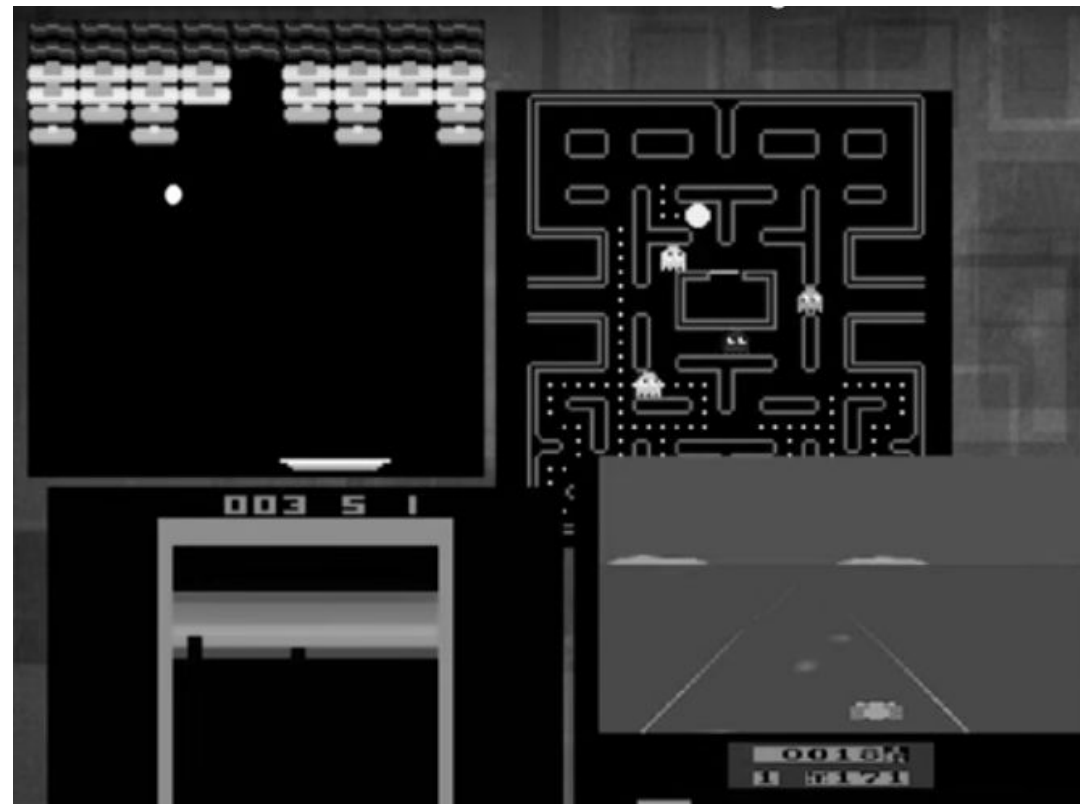
⇒ Robots stacking same colored blocks.



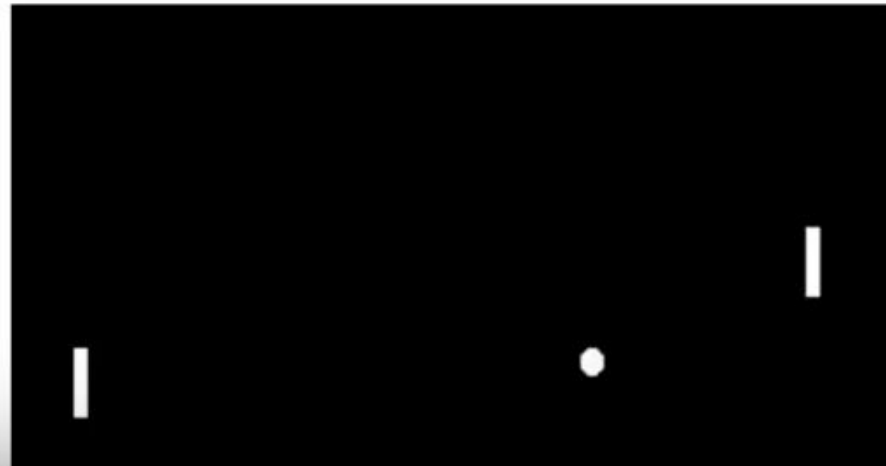
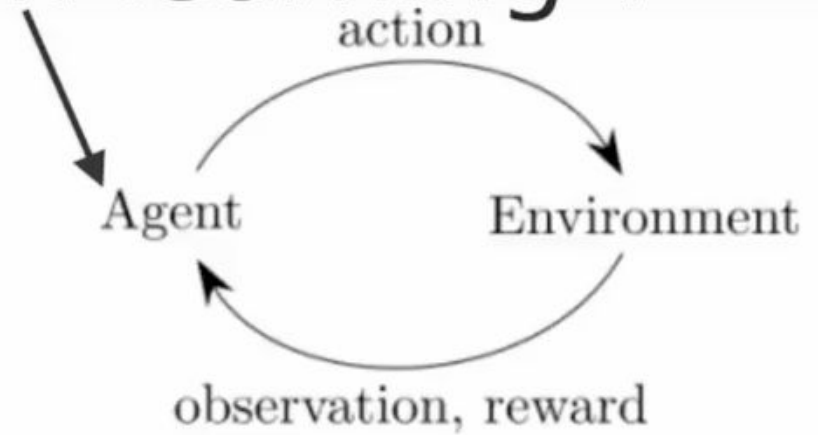
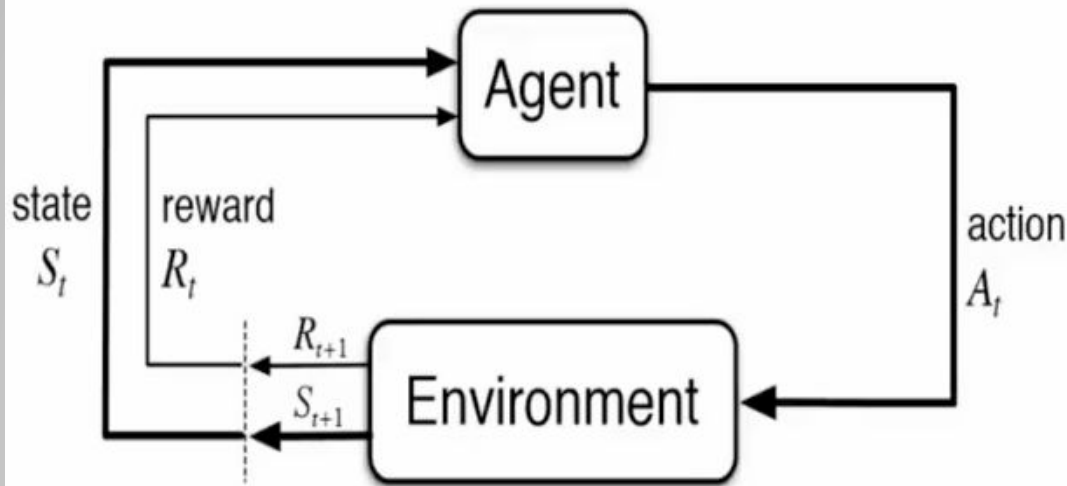


PROBLEMS

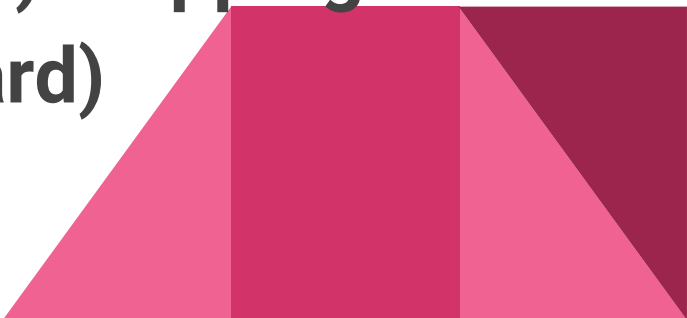
- ❖ **REWARD SHAPING/ POLICY MAKING** has to be done differently for every game
- ❖ *No generalization*
- ❖ *Can be tricky*
- ❖ *Mass Surveillance ??*



What is Reinforcement learning ?

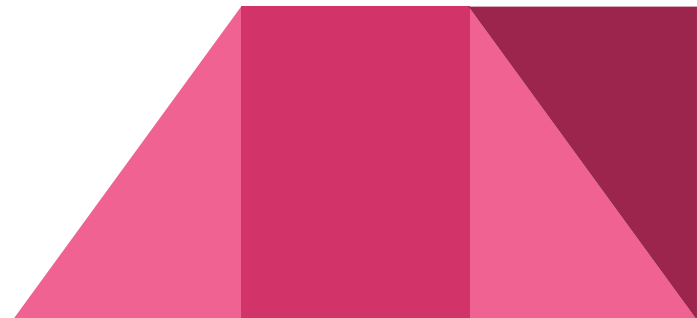


EXAMPLE


- **SELF DRIVING CAR**
 - **Agent**
 - **Environment**
 - **State/observation**
 - **Reward**
 - **Punishment**
 - **Episodes - Sequence of Action from Start To End**
 - **Policy \Rightarrow Behaviour (State,Action) mapping**
 - **Finite Horizon (Maximizing Reward)**
 - **Rubik's Cube??**
- 

The Multi-Armed Bandit Problem

Suppose you are faced with N slot machines (colourfully called multi-armed bandits). Each bandit has an unknown probability of distributing a prize (assume for now the prizes are the same for each bandit, only the probabilities differ). Some bandits are very generous, others not so much. Of course, you don't know what these probabilities are. By only choosing one bandit per round, our task is devise a strategy to maximize our winnings.



Applications

- **Internet display advertising**
 - **Ecology (Limited energy available to animals)**
 - **Finance/Stock Options**
 - **Clinical Trials (researcher may want to find the best treatment while minimising loss.)**
- 

Upper Confidence Bound

MULTI ARMED BANDIT PROBLEM



D1


D2

D3

D4

D5

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.
- 

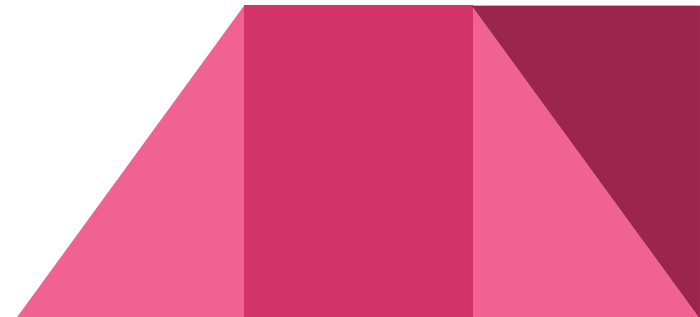
- ▶ Maximize the reward obtained by successively playing gamble machines (the 'arms' of the bandits)
- ▶ Invented in early 1950s by Robbins to model decision making under uncertainty when the environment is unknown
- ▶ The lotteries are unknown ahead of time



Assumptions

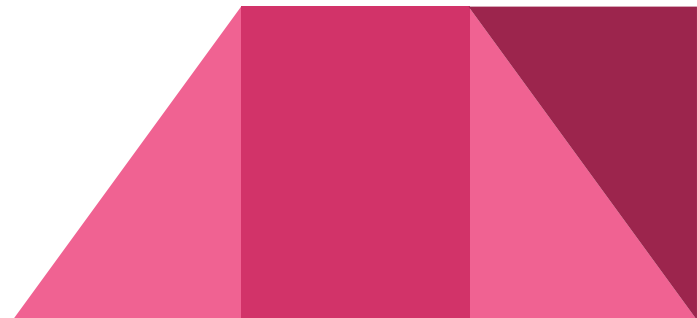
Each machine i has a different (unknown) distribution law for rewards with (unknown) expectation μ_i :

- ▶ Successive plays of the same machine yeald rewards that are independent and identically distributed
- ▶ Independence also holds for rewards across machines

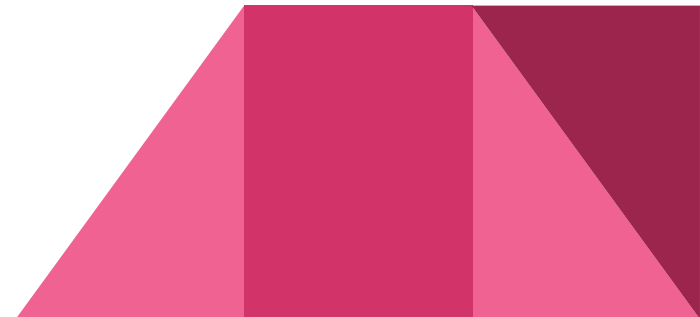


Algorithms

A *policy*, or *allocation strategy*, A is an algorithm that chooses the next machine to play based on the sequence of past plays and obtained rewards.

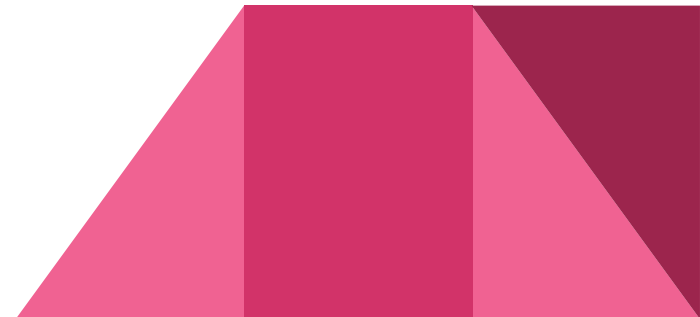


- ▶ Greedy policy: always choose the machine with current best expected reward
- ▶ Exploitation vs exploration dilemma:
 - ▶ Should you exploit the information you've learned or explore new options in the hope of greater payoff?
- ▶ In the greedy case, the balance is completely towards exploitation



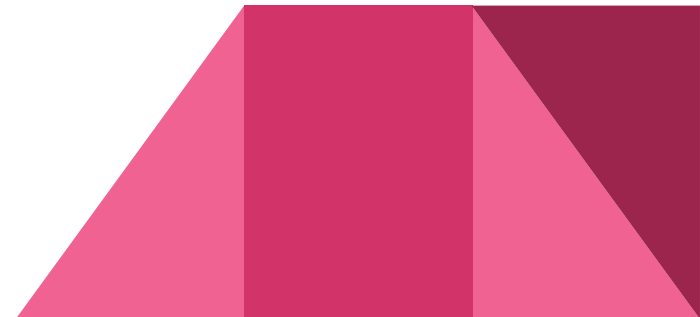
STRATEGY

- ▶ If the expected reward is known, then it would be trivial: just pull the lever with higher expected reward.
- ▶ But what if you don't?
- ▶ Approximation of reward for a gambling machine i : average of the rewards received so far from i



UCB Algorithm

- ▶ Intuition: Select an arm that has a high probability of being the best, given what has been observed so far

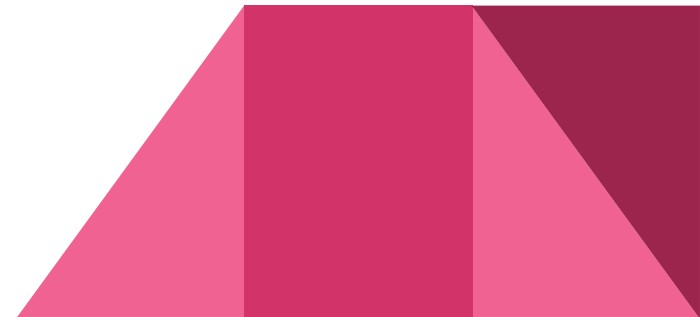


Applications of UCB

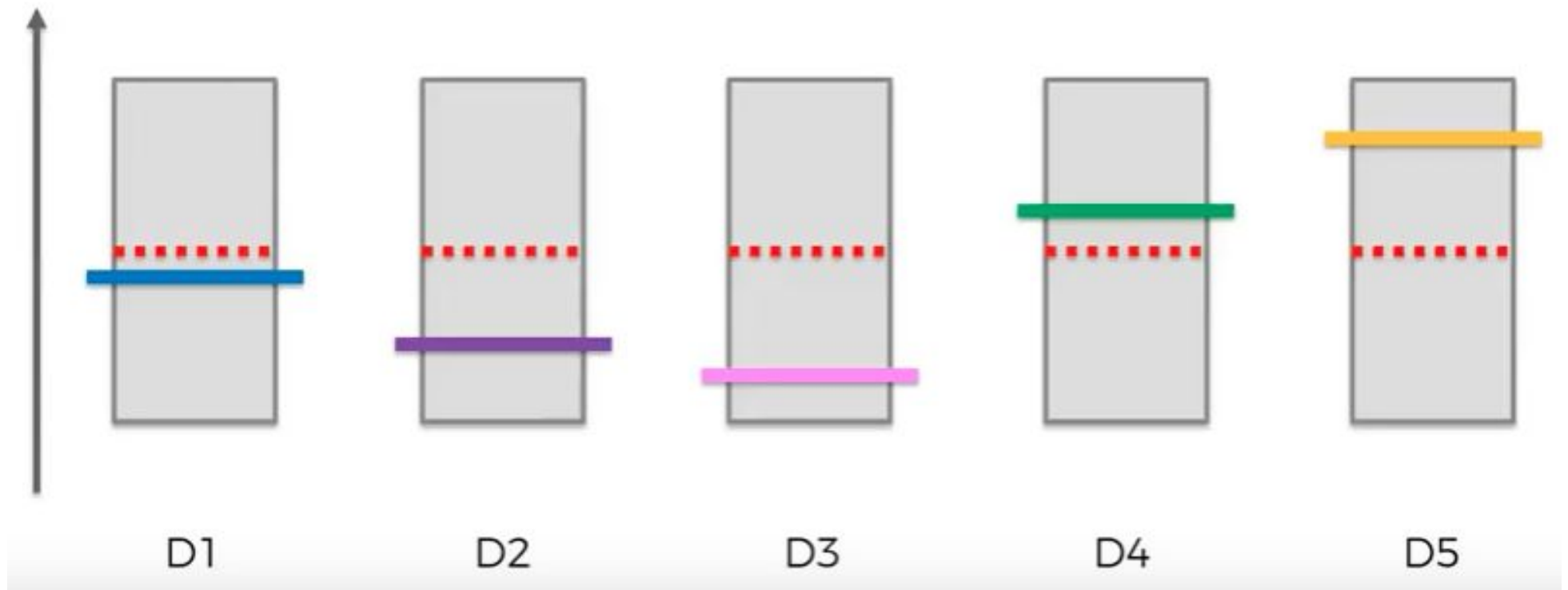
- ▶ **Many applications have been studied:**
 - ▶ Clinical trials
 - ▶ Adaptive routing in networks
 - ▶ Advertising: what ad to put on a web-page?
 - ▶ Economy: auctions



Multi Armed Bandit Problem : Example

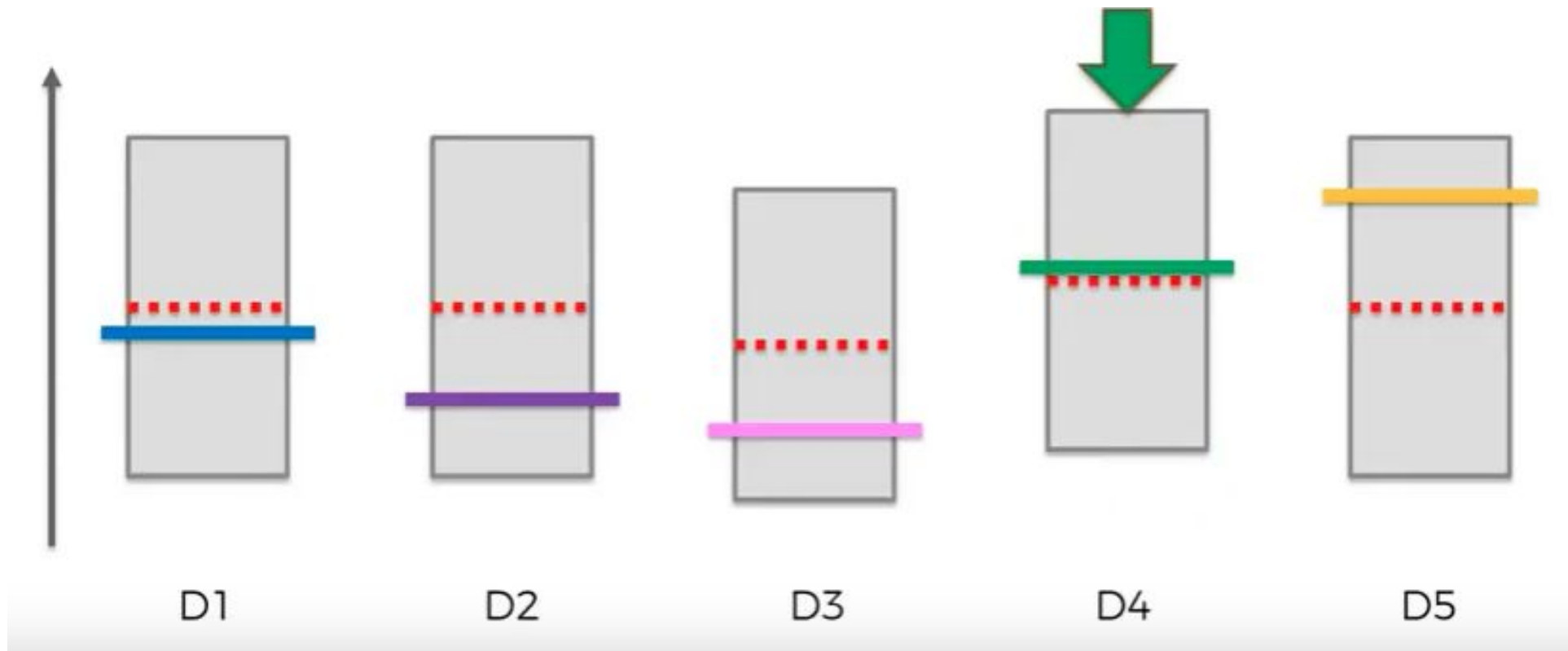


Pick the machine with the highest confidence bound



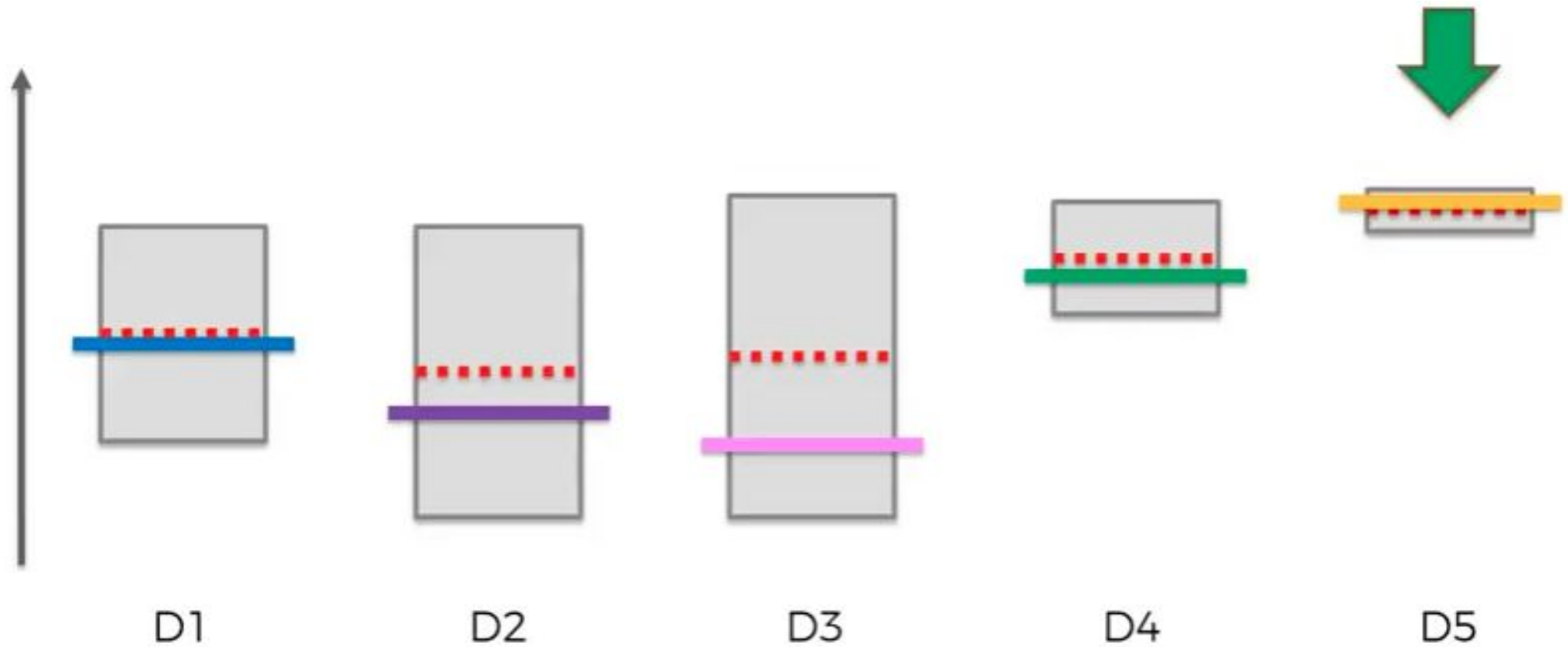
- Expected Return
- Confidence Bound.

After two observations with D3 and D4



- Expected Return
- Confidence Bound

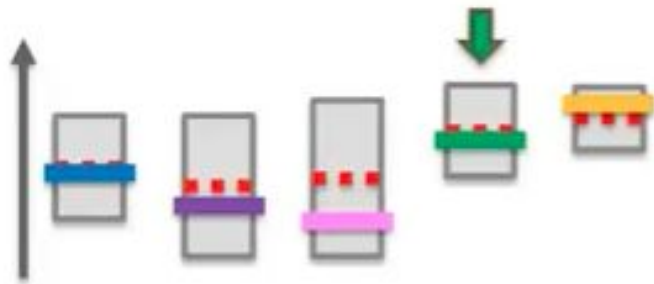
After all observations



- Expected Return
- Confidence Bound

UCB vs Thompson Sampling

UCB



- Deterministic
- Requires update at every round

Thompson Sampling



- Probabilistic
- Can accommodate delayed feedback

Thompson Sampling Algorithm

To maximize our return, what will be the best distribution of handplays (e.g., ad distribution) among a set of game devices (e.g. customers)



D1



D2



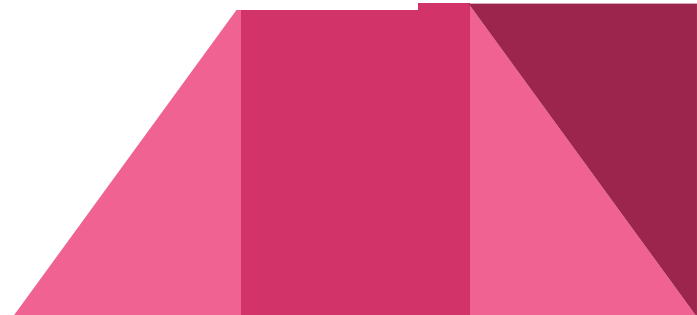
D3



D4

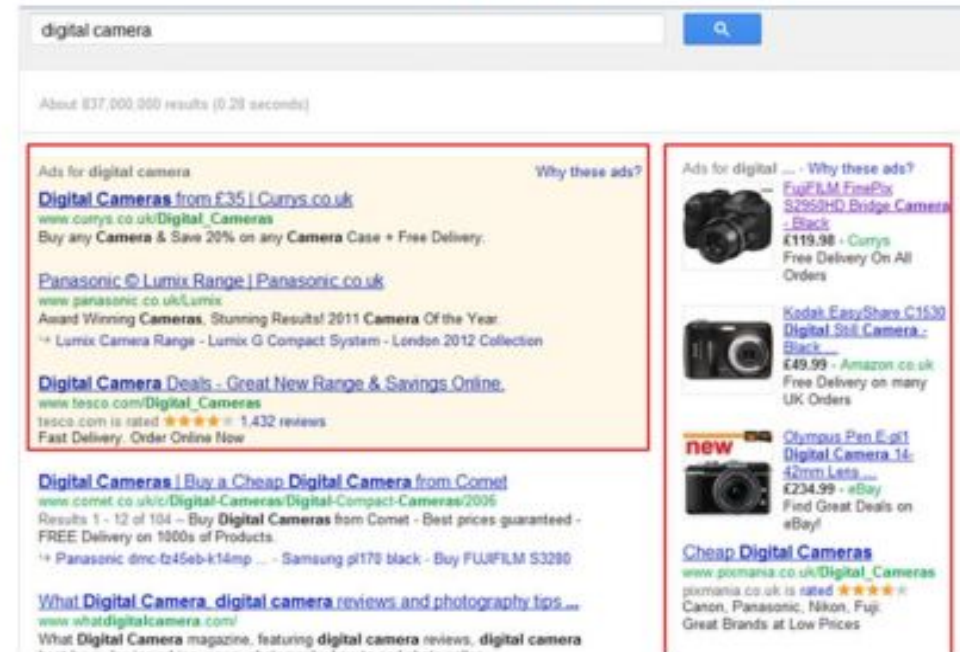


D5



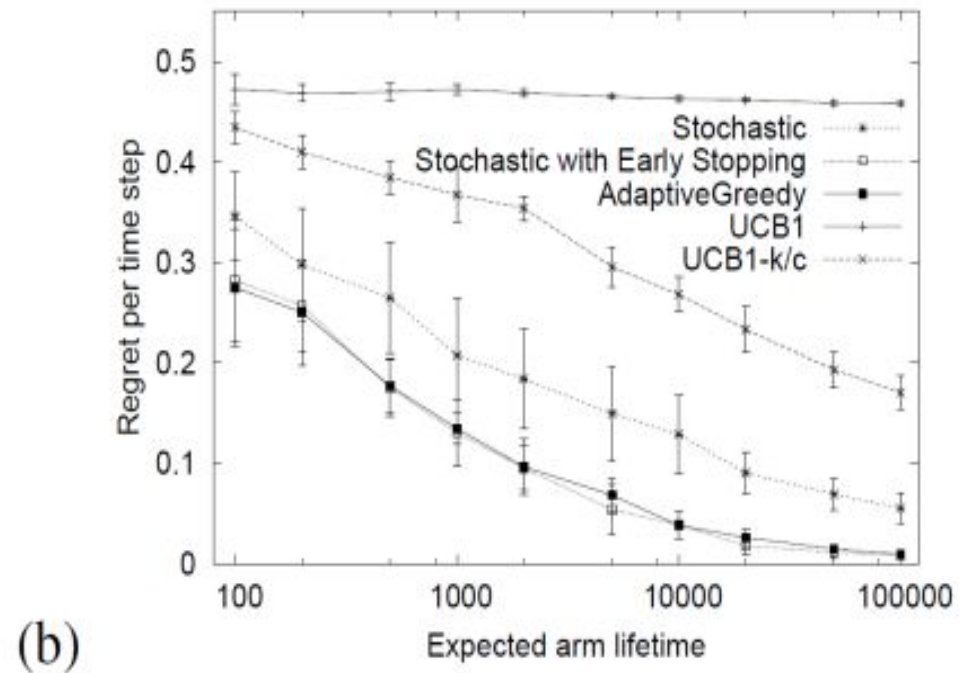
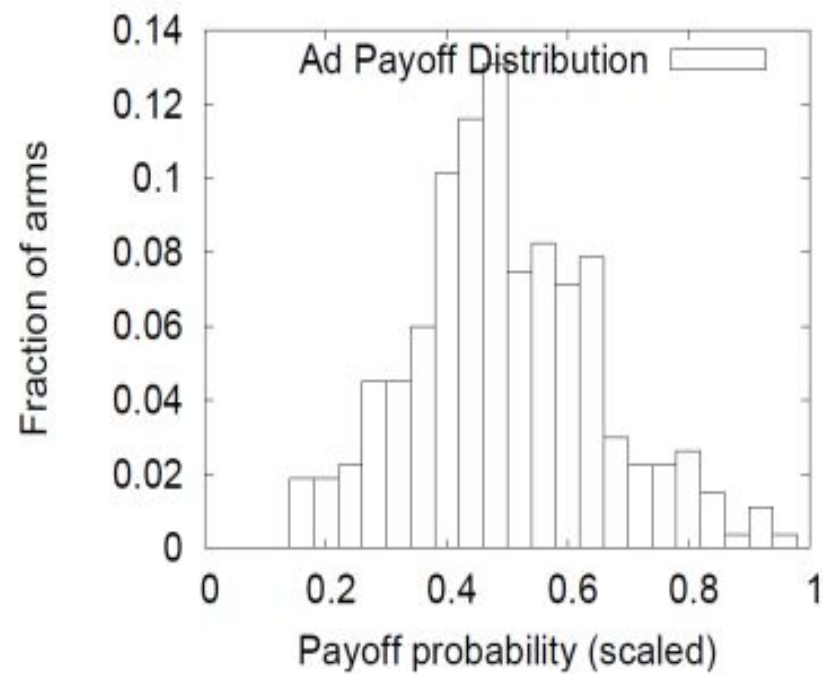
Internet Advertising

- ▶ Each time a user visits the site you must choose to display one of K possible advertisements
- ▶ Reward is gained if a user click on it
- ▶ No knowledge of the user, the ad content, the web page content required...
- ▶ T = users accessing your website



Internet Advertising

- ▶ Where it fails: each of these displayed ads should be in the context of a search or other webpage
- ▶ Solution proposed: *contextual bandits*
- ▶ Context: user's query
- ▶ E.g. if a user input “flowers”, choose only between flower ads
- ▶ Combination of supervised learning and reinforcement learning



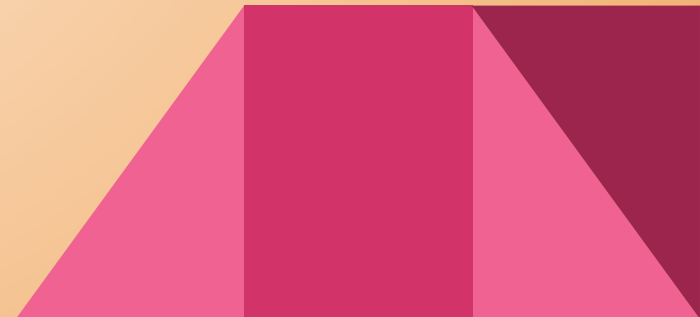
[Lu et al., "Contextual multi-armed bandits",
13th International Conference on Artificial Intelligence and Statistics (AISTATS), 2010]

Network server selection

- ▶ A job has to be processed to one of several servers
- ▶ Servers have different processing speed (due to geographic location, load, ...)
- ▶ Each server can be viewed as an arm
- ▶ Over time, you want to learn which is the best arm to play
- ▶ Used in routing, DNS server selection, cloud computing, ...

Thompson Sampling

- **Thompson sampling is a heuristic for choosing actions that addresses the exploration-exploitation dilemma in the multi-armed bandit problem.**
- **One of the oldest heuristics for multi-armed bandit problems.**
- **Randomized algorithm based on Bayesian ideas.**
- **The basic idea is to assume a simple prior distribution on the parameters of the reward distribution of every arm, and at any time step, play an arm according to its posterior probability of being the best arm.**
- **This algorithm is known as Thompson Sampling (TS)**



Thompson Sampling

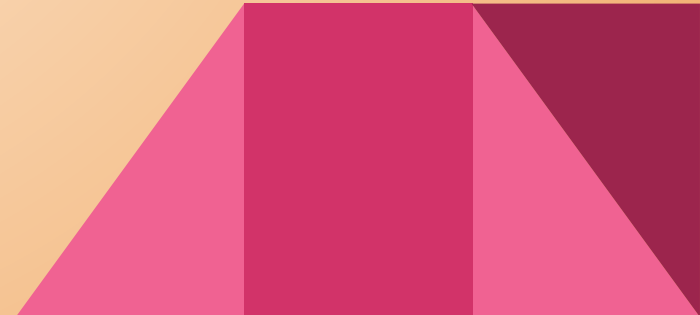
Suppose you have three machines.

Each machine has an arm (like a slot machine).

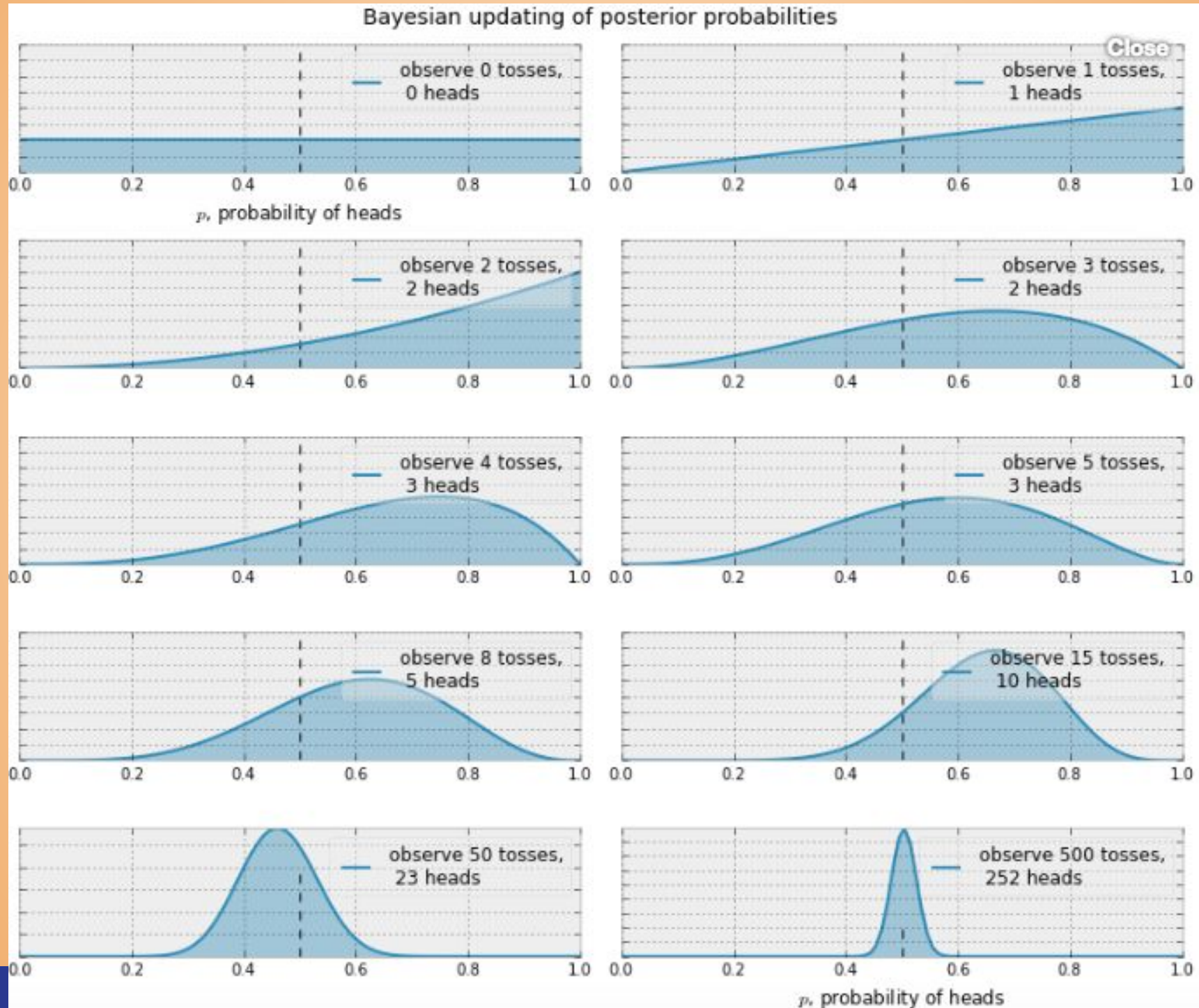
When you pull an arm, you get paid \$1 or not depending upon an unknown probability for each machine.

The machines are stand-ins for a real problem scenario such as trying to determine which of three new medicines is most effective at curing a patient.

The goal is to find the machine with the best payoff probability.



Thompson Sampling : Bayesian Methods



Thompson Sampling Algorithm : Multi Armed Bandit Problem

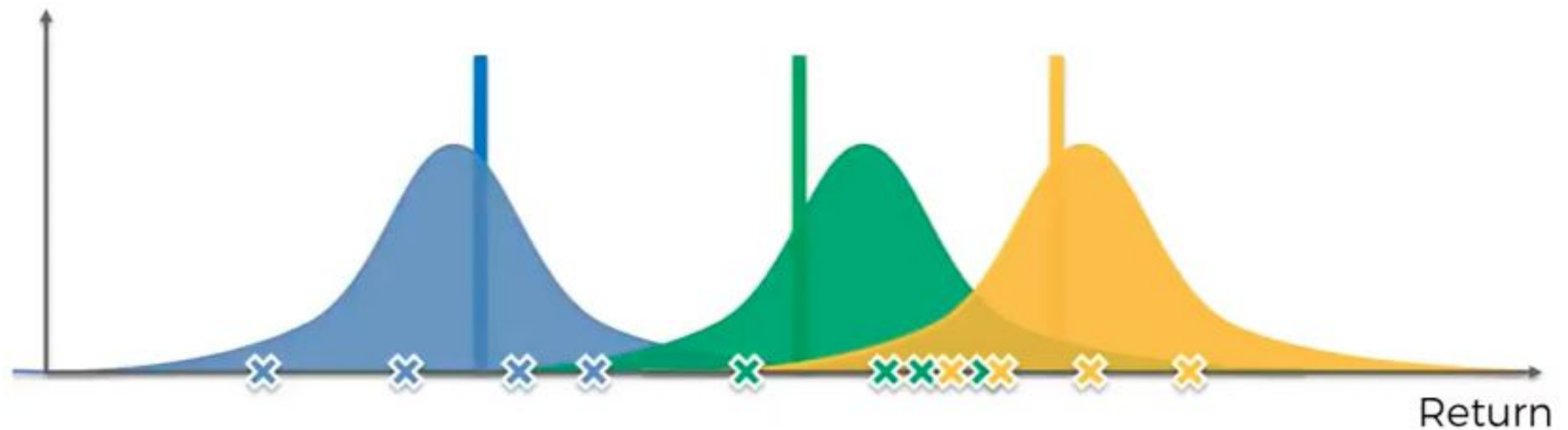
3 bandits



 \Rightarrow Expected Return

Thompson Sampling Algorithm

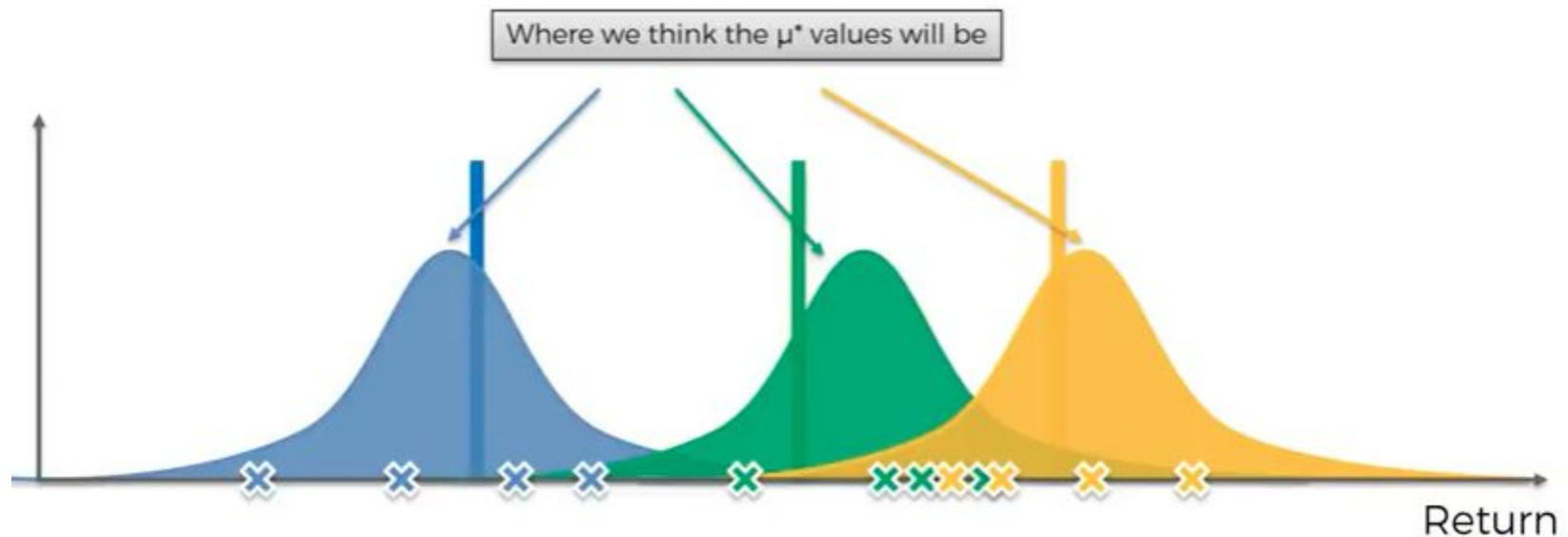
3 bandits



||| ⇒ Expected Return

Thompson Sampling Algorithm

3 bandits



||| ⇒ Expected Return

Thompson Sampling Algorithm

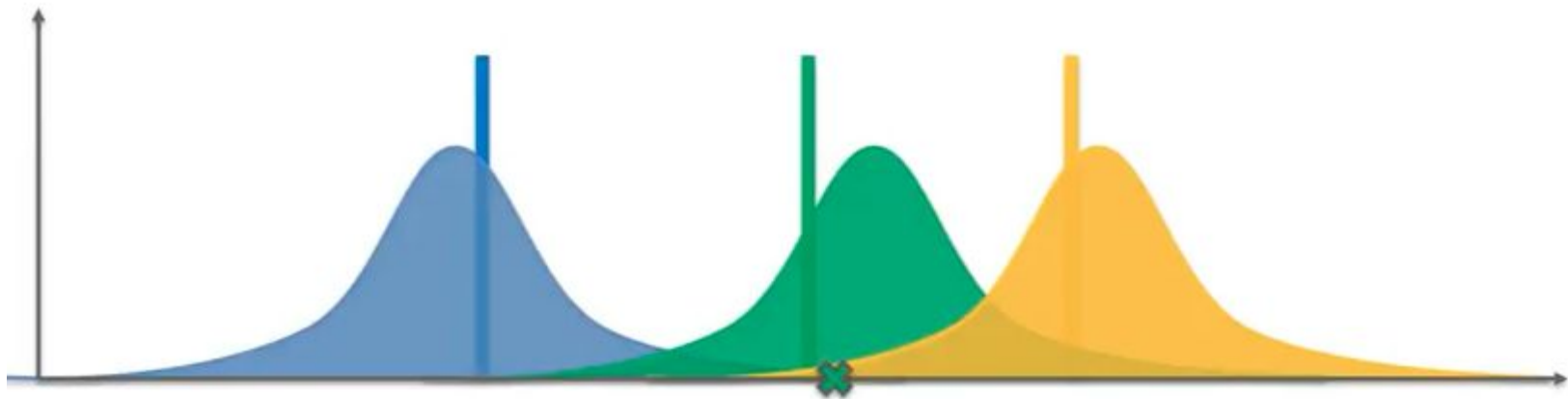
3 bandits



||| ⇒ Expected Return

Thompson Sampling Algorithm

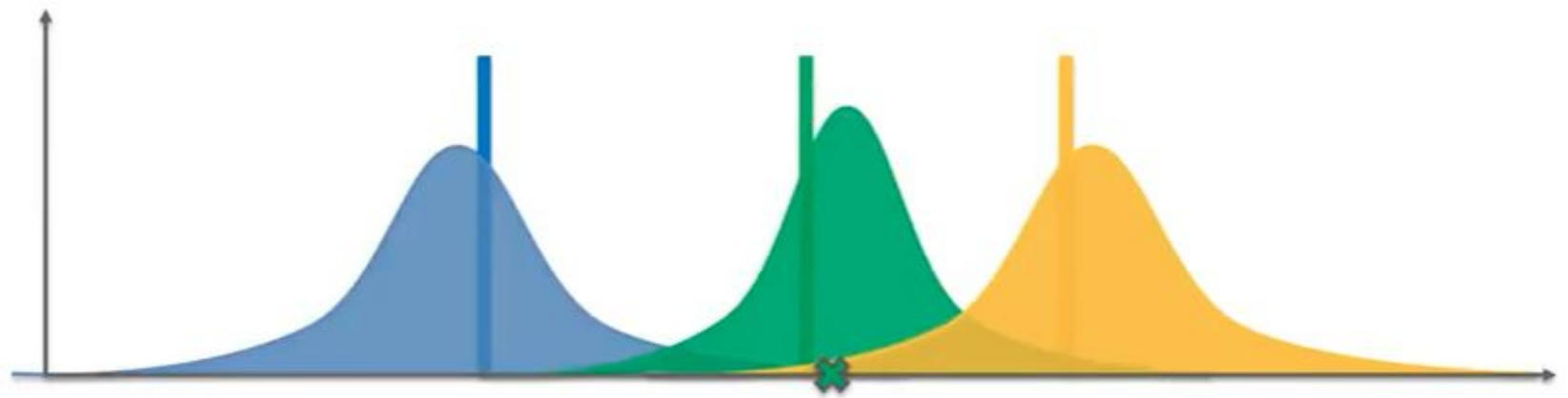
3 bandits



 \Rightarrow Expected Return

Thompson Sampling Algorithm

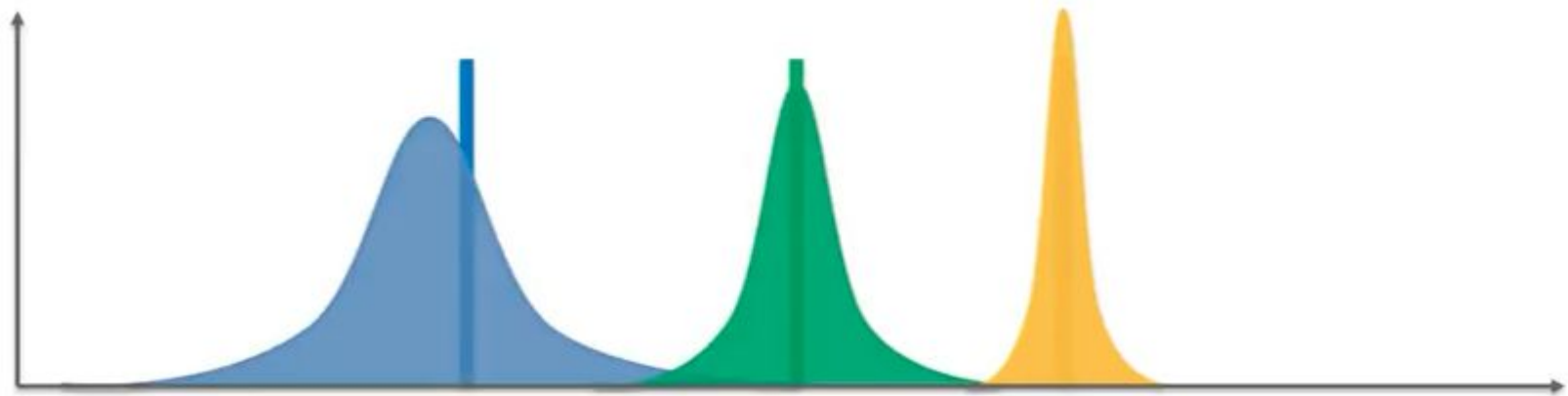
3 bandits



⇒ Modified Expected Return

Thompson Sampling Algorithm

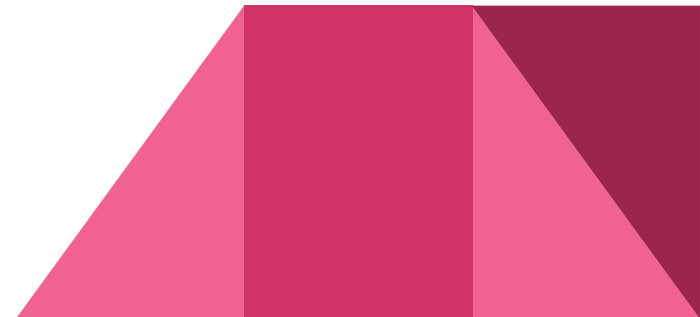
3 bandits



⇒ Maximum Expected Return

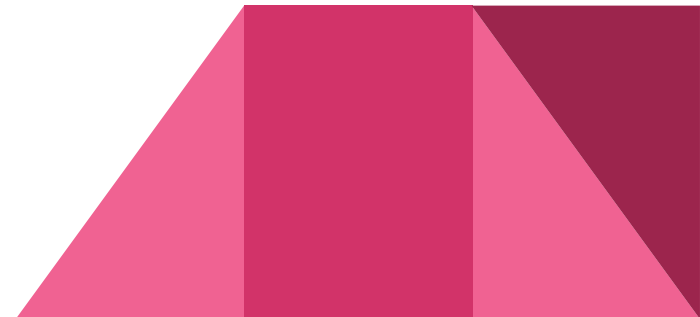
Boosting

- Boosting algorithms convert weak learners into strong learners.
- A weak learner is one which is slightly better than random guessing. Let's understand **boosting first** (in general).

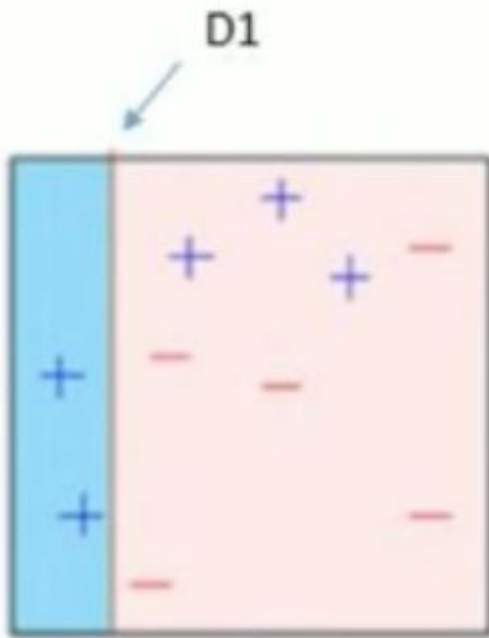


Boosting

- Boosting is a sequential process; i.e., trees are grown using the information from a previously grown tree one after the other.
- This process slowly learns from data and tries to improve its prediction in subsequent iterations. Let's look at a classic classification example

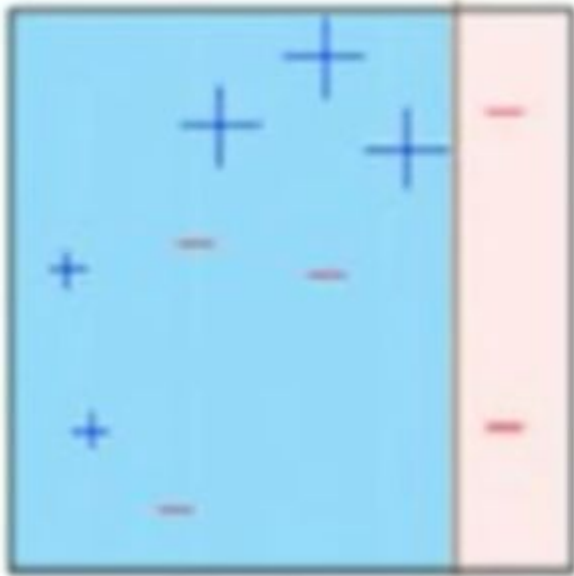


Boosting



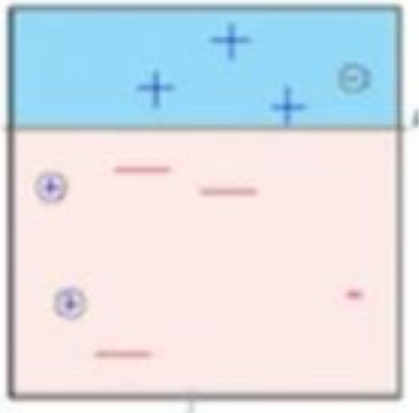
- The first classifier creates a vertical line (split) at D1.
- It says anything to the left of D1 is + and anything to the right of D1 is -. However, this classifier misclassifies three + points.

Boosting



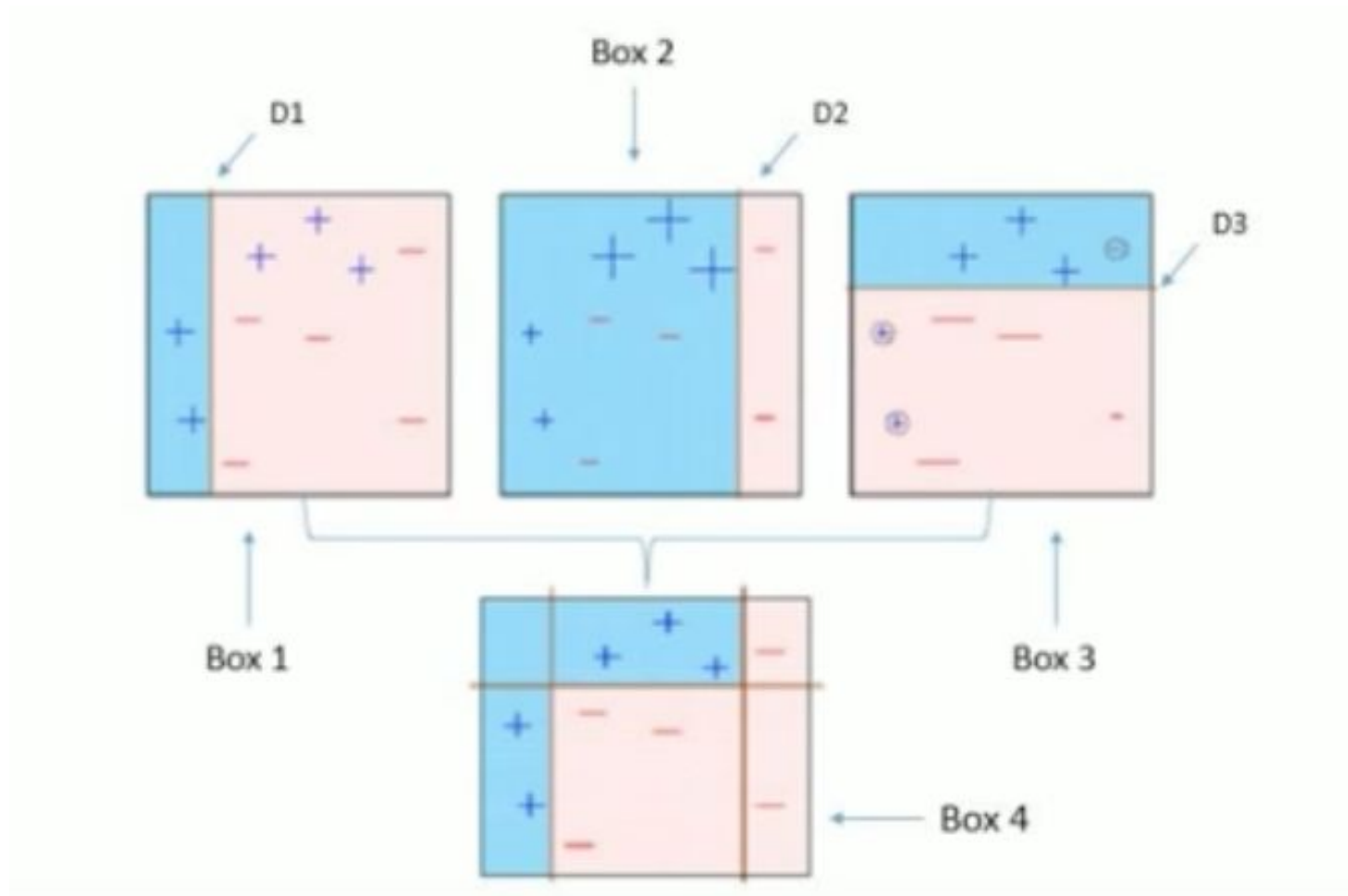
- The next classifier says don't worry I will correct your mistakes.
- Therefore, it gives more weight to the three + misclassified points (see bigger size of +) and creates a vertical line at $D2$.

Boosting



- The next classifier continues to bestow support.
- Again, it gives more weight to the three - misclassified points and creates a horizontal line at D3.

Boosting



Summary

- ▶ Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal.
- ▶ The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them.
- ▶ In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards.

Summary

- ▶ **Reinforcement learning:**
 - ▶ no teacher; the only feedback is the reward obtained after doing an action
 - ▶ Useful in cases of significant uncertainty about the environment

