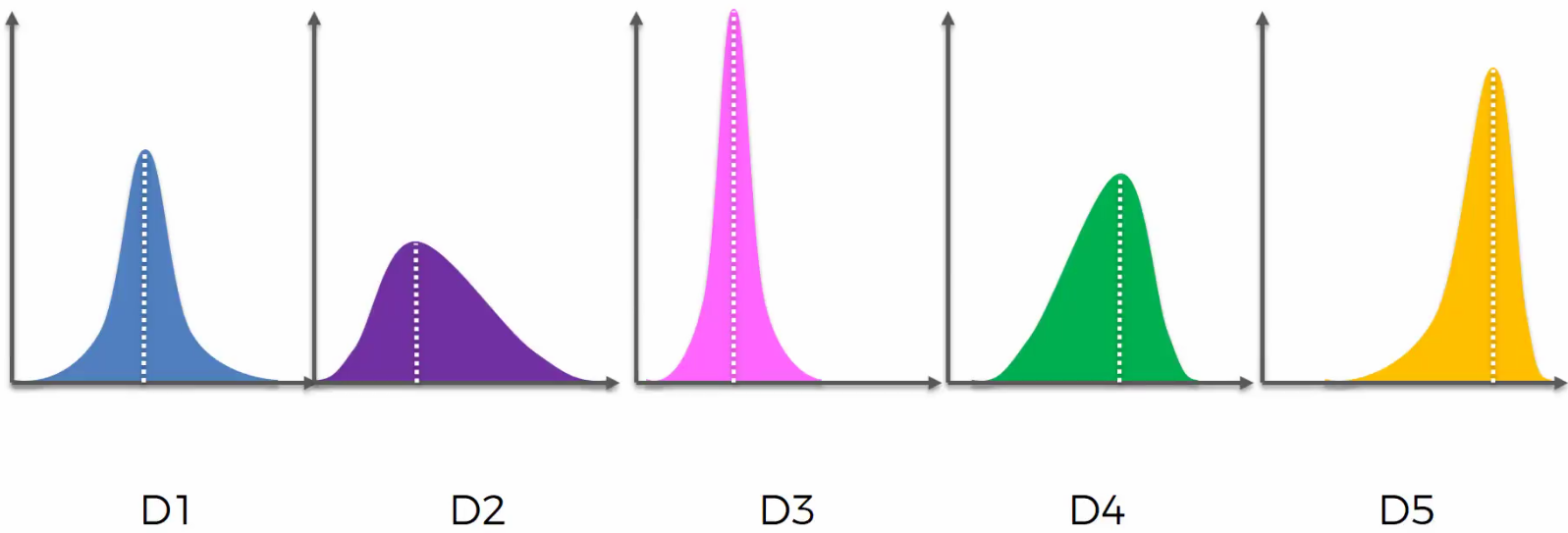
* **Reinforcement Learning** = branch of ML, also called **Online Learning**, used to solve interacting problems where data observed up to time **t** is considered to decide which action to take at time t + 1
* It is also used for AI when training machines to perform tasks such as walking.
* Desired outcomes provide the AI w/ reward, undesired outcomes w/ punishment, as machines learn through trial +error.

**Multi-Armed Bandit Problem**

* 1-armed bandit = slot machine, the cause of 1 of the quickest ways to lose $ in casinos
* Multi-armed bandit = challenge one faces when they come up to multiple slot machines + want to find out how to play them to maximize return from the # of games one is going to play
* Assume each machine has a distribution of #’s/outcomes behind it out of which the machine picks results when pulled (tells you if you win/lose based on distribution w/in the machine
* Problem = we don’t know the distributions + they’re different in each machine
* Goal -= figure out which machine’s distribution is best for us



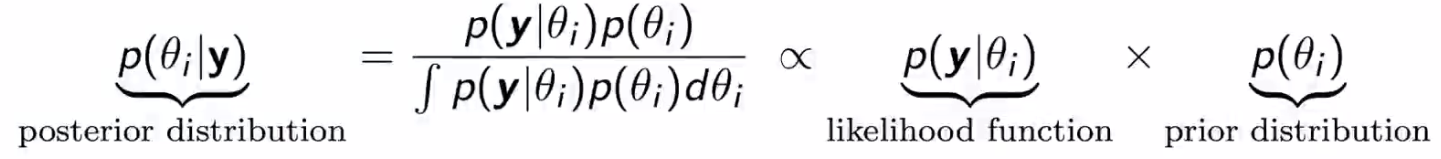
* D5 is the best distribution with the most left-skew (most positive outcome values = most “winning” outcome values)
* But, we’re spending $ trying to figure this out 🡺 the longer we take to figure it out, the more confident we are but the less $ we have (greater cost)
* Want to do so as fast as possible
* 2 ways = **exploration** vs. **exploitation**
* Explore machines to find best one + at the same time exploit findings we have to get max return
* **Regret** = suffered when we use a non-optimal method (using any machine that’s not D5) + can be quantified as the difference between the best outcome + the non-best outcome + opportunity costs of exploring other machines
* Longer time exploring non-optimal machines = more regret
* But if we don’t spend enough time exploring, we might choose a sub-optimal machine as our optimal machine
* Want to minimize amount of time exploring all machines as we’re still earning money, but not optimizing results
* Most common modern application = advertising
* Coke wants to run a campaign + find out which ad maximizes ROI



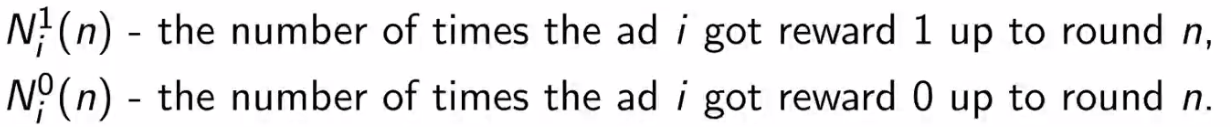
* Distribution only become known when thousands of people click on these ads
* To approach this, we could do an A/B test + wait until we have large enough sample to choose which ad is best, but the problem here = costs a lot of time + $ as an A/B test is pure exploration of ALL ads, even the non-optimal ones
* Want to exploit the best one while exploring all the ads (find out best ad while doing the campaign)

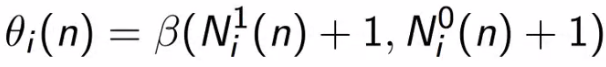
**Thompson Sampling**

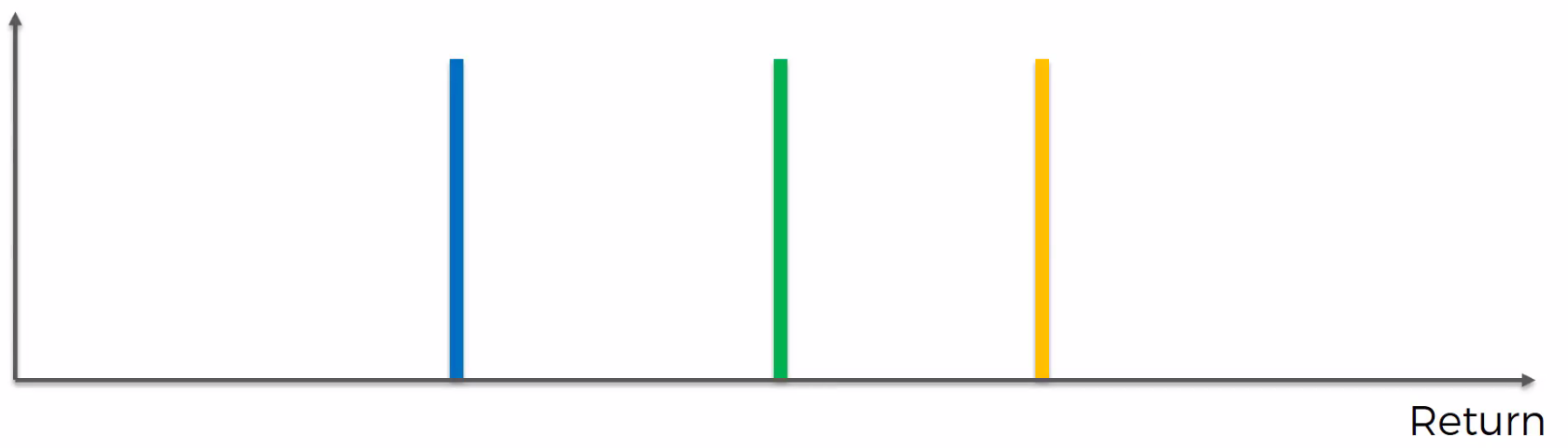
* **Bayesian Inference**
* Ad i gets rewards ***y*** from a **Bernoulli distribution =** 
* ***ϴi*** isUNKNOWN, but we set its **uncertainty** by *assuming* it has a **uniform distribution:****,** which is the **PRIOR distribution**
* Then do:



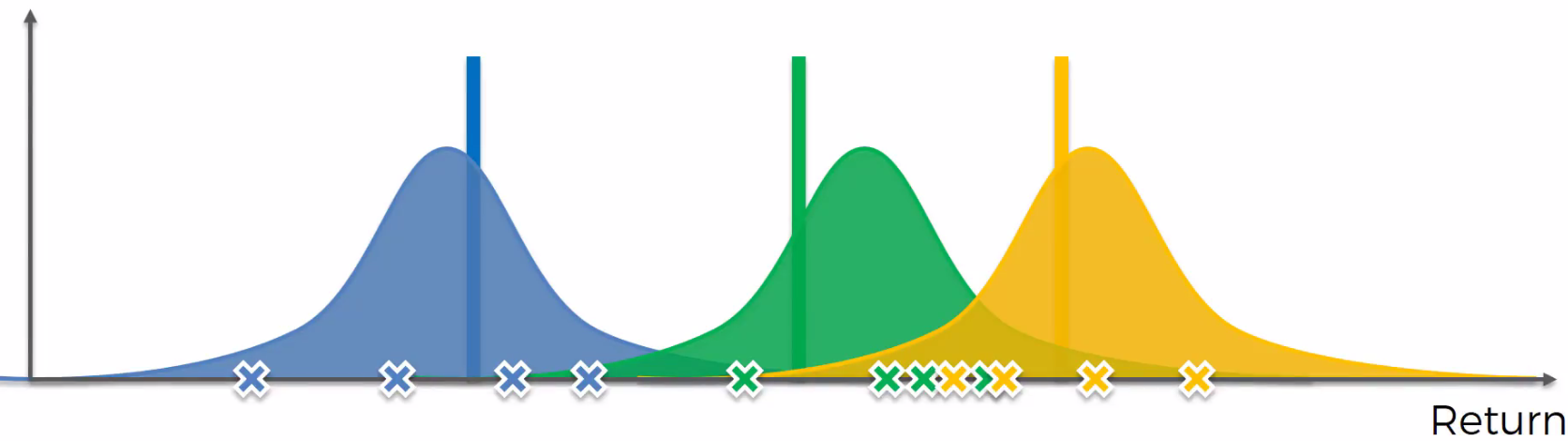
* Returns 
* At each round, **n**, take a random draw *ϴi*(n) from this posterior distribution for each ad i, and *then* select ad i w/ the highest *ϴi*(n)
* **Steps**
* At each round n, consider 2 #’s for each ad i:



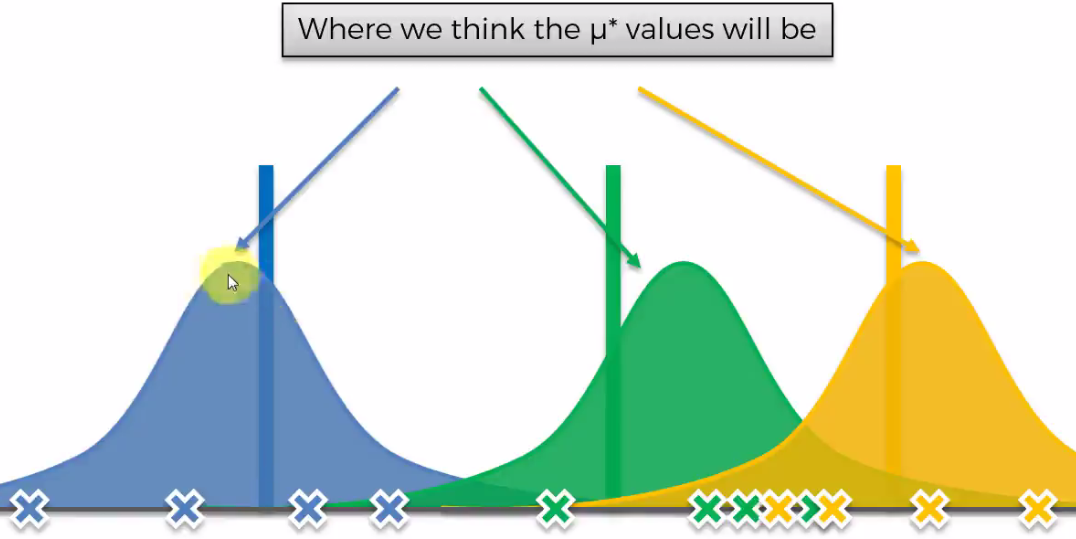
* For each ad I, take random draw from distribution
* Select ad w/ highest *ϴi*(n)
* Example: 3 bandits



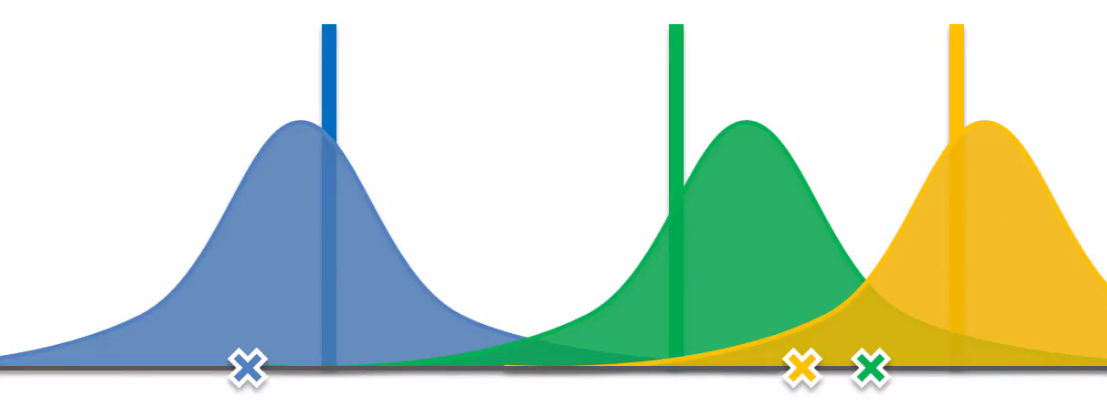
* Each machine has a reward distribution behind it (line = center of distribution = **expected reward**)
* Algorithm does NOT know this info which says yellow machine is best (furthest to right = highest return)
* At start of algorithm, we know nothing (no prior) 🡺 need some trial rounds to get some data, say from the blue machine, from which algorithm gets some distribution:



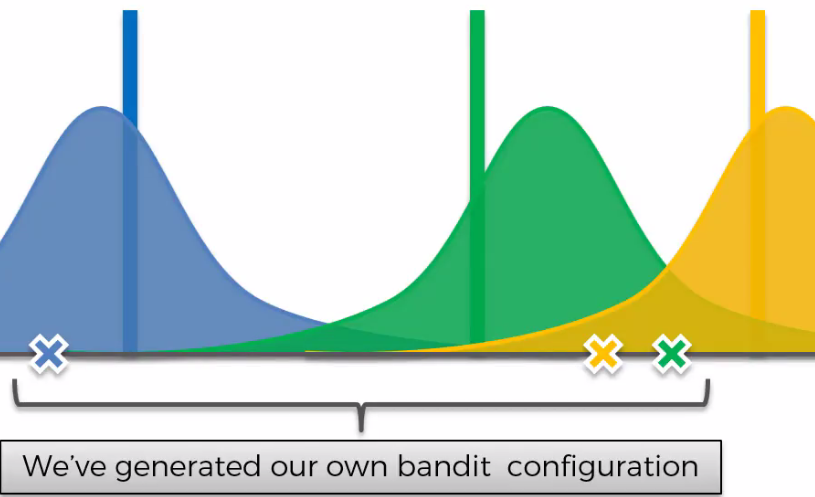
* The new distributions represent where we think the ACTUAL expected value might be = auxiliary mechanism to solve the problem by *kind-of recreating* how the machines were created instead of *actually recreating* the machines
* **\*NOT trying to guess the distributions, just where u\* will be**



* Trying to mathematically explain what we think is going on, or trying to create a perception of this world
* Thompson sampling = **probabilistic** algorithm, while UCB was **deterministic** (strict = choose machine w/ highest UCB)
* In next round, algorithm pulls a value out of machine’s 1 distribution, then machine 2’s, then machine 3’s according to their respective distributions:

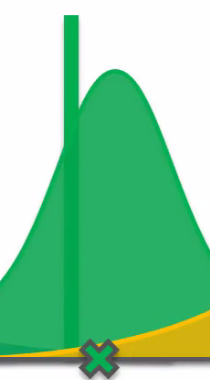


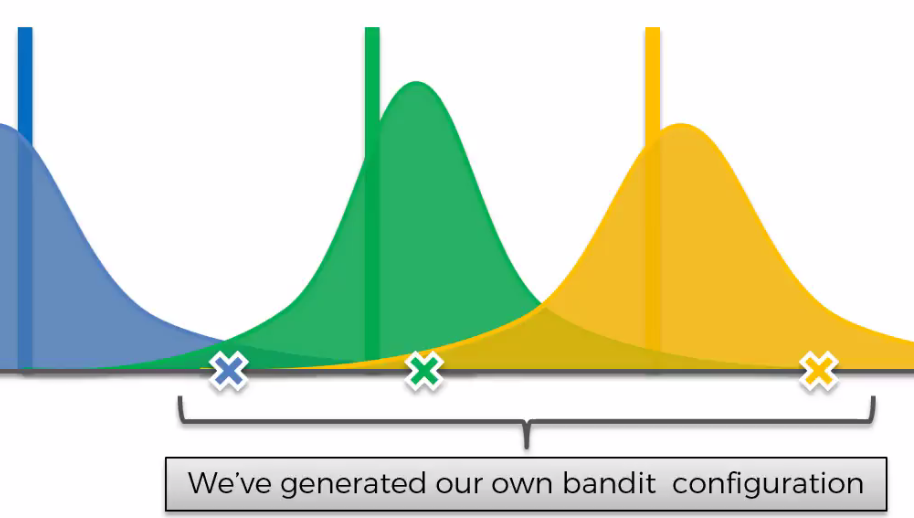
* In long-term, will select values closer to center (peak)
* This generates our own **bandit configuration =** own hypothetical set of machines in our virtual world where we say the “actual” expected return value for each machine is the value we just picked out



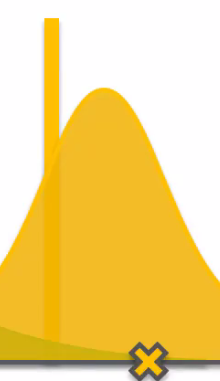
* In this problem, we’d pick machine 2 since it has the highest “actual” expected return value
* Then we translate this result from the hypothetical/virtual world to the actual one 🡺 algorithm pulls lever of machine 2 + we get a result from its TRUE distribution



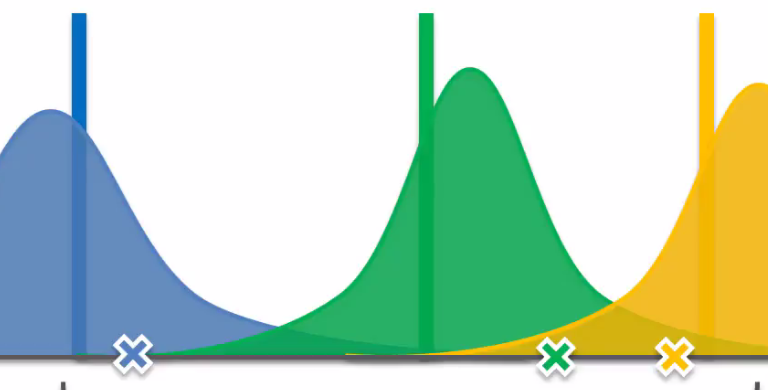
* This new info tells the algorithm that it must **adjust its perception of the world (Bayesian) to use this new point to create a posterior distribution**
*  🡺 
* Perception changes 🡺 distribution shifts to the left (more to the new DP) + gets more narrow (b/c sample size increased) as our distribution get more + more refined
* Then repeat the same process for a new round
* Virtual Bandit Config 2:



* Now pick best bandit = yellow 🡺 pull level of yellow bandit in “real” world to get a value from its real distribution + update our perception of the world (yellow’s distribution moves to left + gets more narrow)

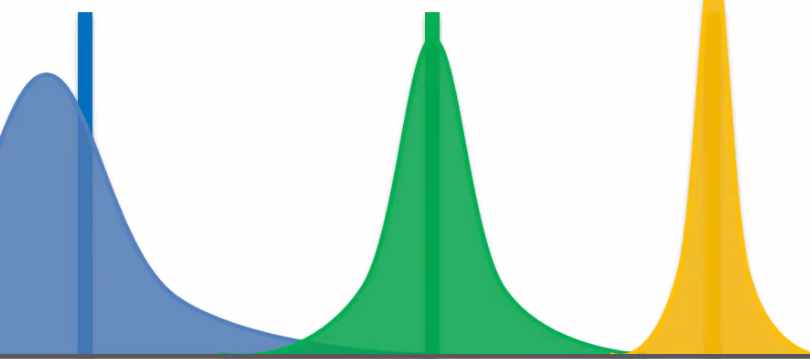
 🡺 

* Do it again:



 🡺 

* Repeat until distributions are refined substantially



* We see that we have “found” and exploited the best machine (yellow), and its distribution tis the most refined b/c we exploited it the most (largest sample size)

**Compare Thompson Sampling vs. UCB**

* UCB = **deterministic** = very straightforward about what’s going to happen at each round = pick machine w/ highest UCB
* Requires an update at every round = value return from the machine is then incorporated in order to move onto the next round b/c without that adjustment, nothing changes
* TS = **probabilistic =** the algorithm contains distributions which represent our perception of the world + where we *think* the actual expected returns are for each machine
* Each iteration = generate *random* values from the distribution
* Can accommodate delayed feedback 🡪 if you pull a laver + will only the result of that pull 500 rounds later, the algorithm still works b/c we’d still get a new set of hypothetical bandits even w/out updating our perception (b/c probabilistic)
* AKA don’t have to update the algorithm with a result each time
* Can be difficult + computationally expensive to update algorithm after *every* click on a website
* Can run 500 or 5k rounds in a batch + *then* update it
* Typically gives better **empirical** **evidence**