Lecture 5: Thompson Sampling for Bandits

ID 6002W: Online and Reinforcement Learning

Web-enabled M.Tech. program, IIT Madras

Suryanarayana Sankagiri

25th May, 2026

The Goal

Maximise long-term performance

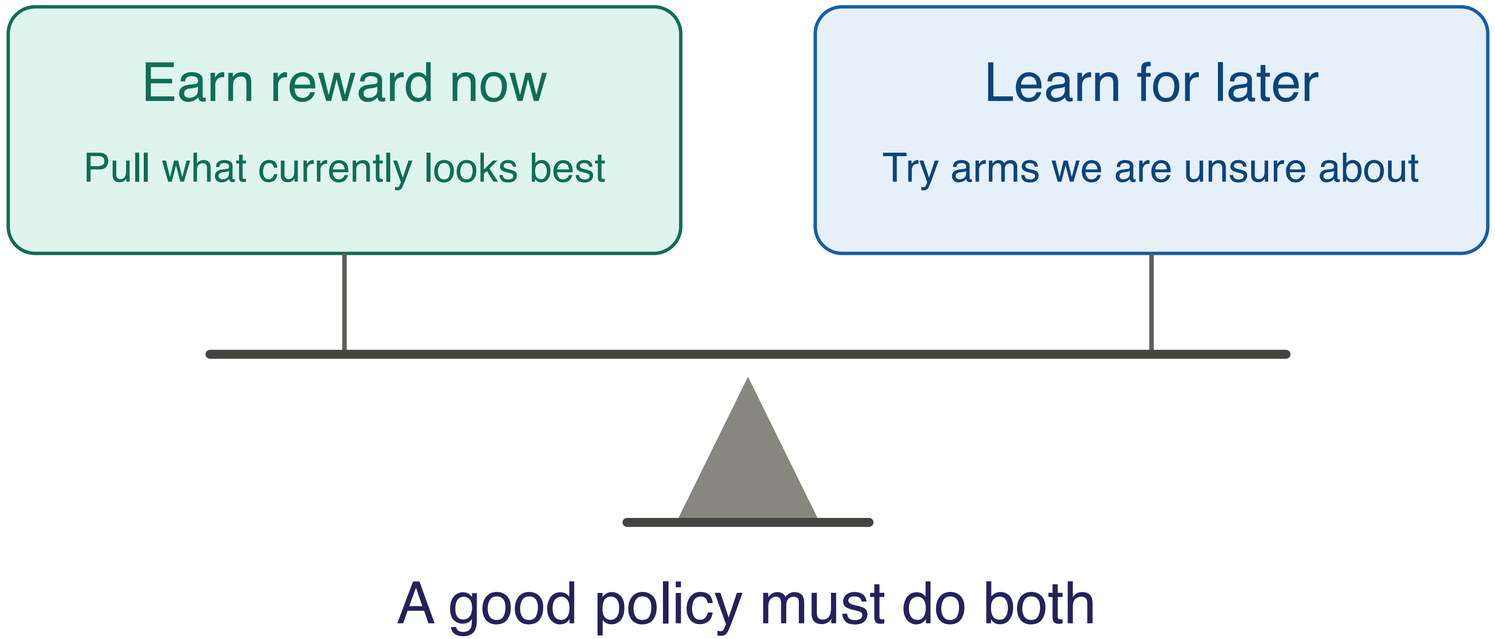
Maximise total reward: $\sum_{t=1}^{T} X_t$

Equivalently, minimise cumulative regret: $T\mu^* - \sum_{t=1}^{T} X_t$

If we knew $i^* = \arg\max_{i \in 1,2,\ldots,K} \mu_i$, we would pull i^* every time

But we do not know i^*

Exploration v/s Exploitation



Every action has two jobs: get a high reward and learn more about arm rewards

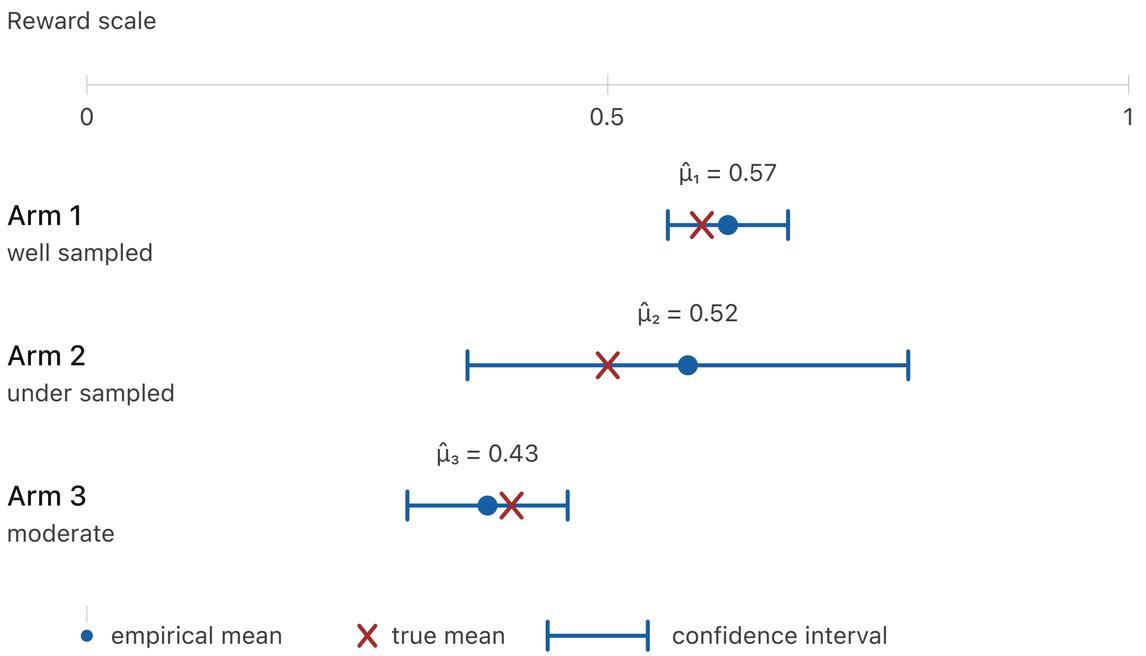
This tradeoff is fundamental in all online learning problems (including reinforcement learning)

UCB Algorithm

Using confidence intervals

For each arm i, we can easily maintain

- $\hat{\mu}_i(t)$: estimate of the mean reward μ_i
- $\sqrt{\frac{c \log T}{T_i(t)}}$: width of confidence interval



Large confidence intervals imply unreliable estimates

Need to pull those arms more often in order to reduce uncertainty

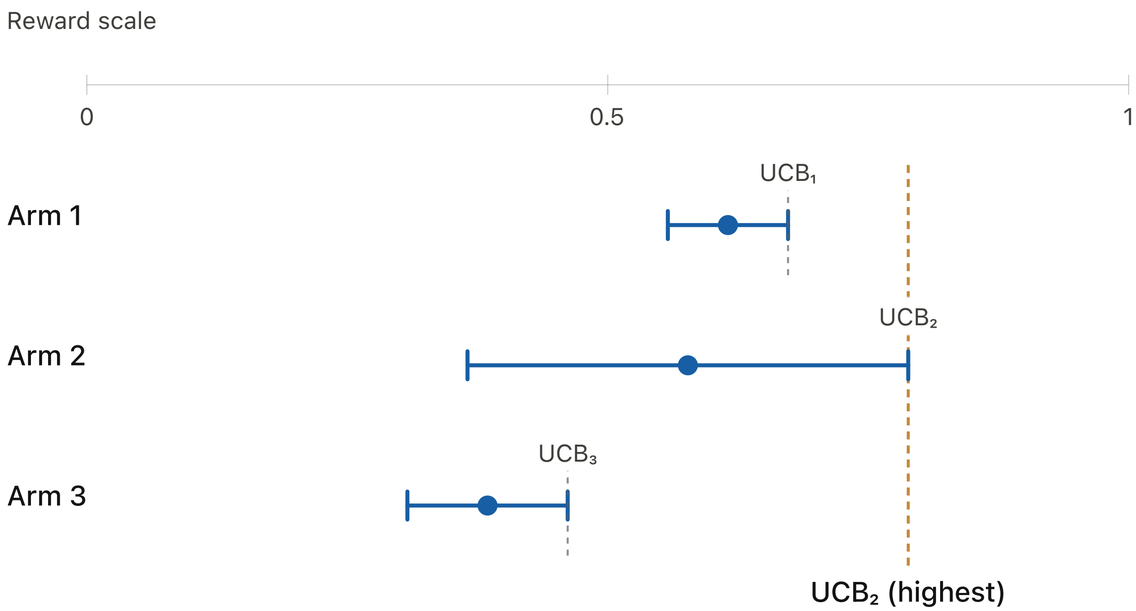
The UCB algorithm is a data-driven strategic approach to exploration

UCB Algorithm

Optimism in the Face of Uncertainty

$$UCB_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{c \log T}{T_i(t)}}$$

$$\text{Pull } A_t = \arg \max_{i \in \{1,2,\ldots,K\}} UCB_i(t-1)$$



Act as if each arm is as good as it could reasonably be

- Maximising $\hat{\mu}_i(t)$ leads to exploitation
- Maximising $\sqrt{\frac{c \log T}{T_i(t)}}$ leads to exploration

UCB Algorithm

Pseudocode

Initialise: Pull each arm once

For $t = K + 1, \ldots, T$

For each arm i:

Compute $UCB_i(t-1)$

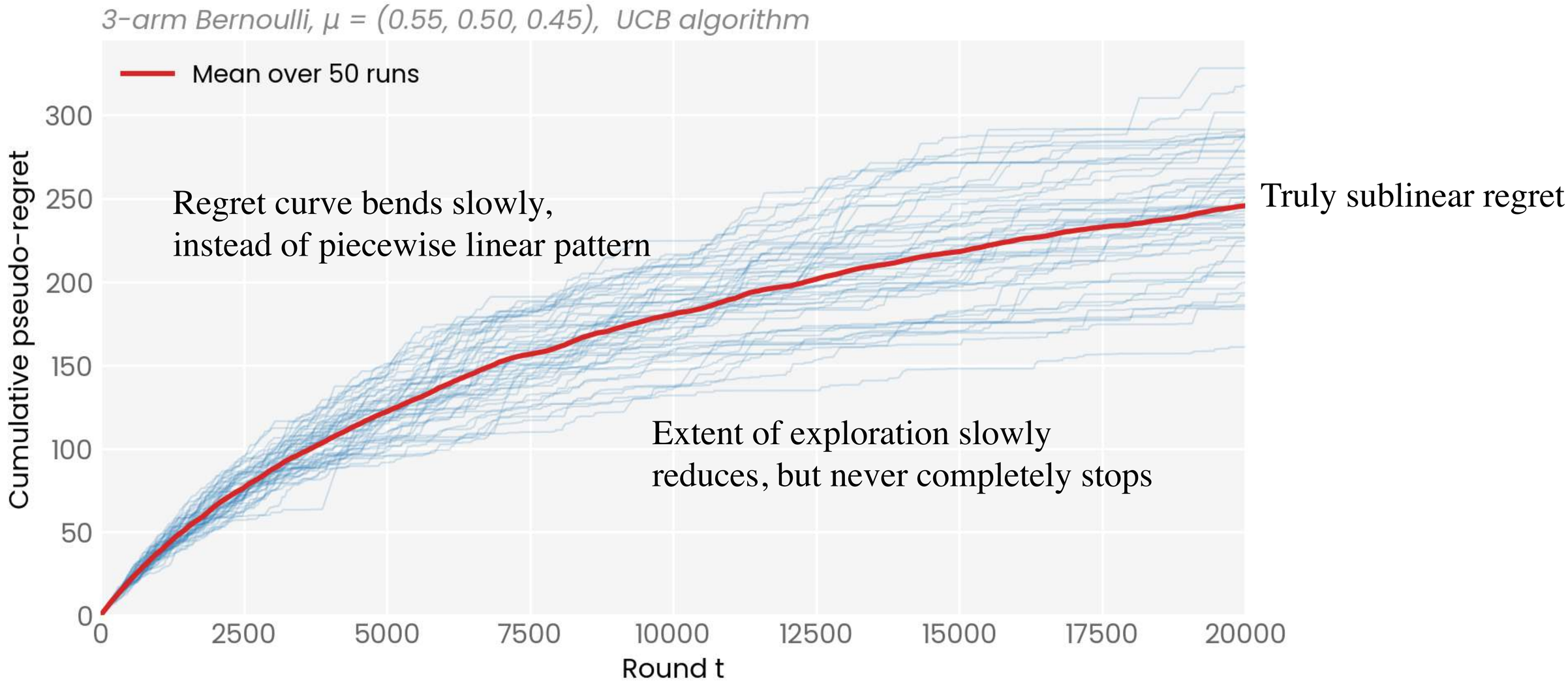
Pull arm $A_t = \arg \max_{i \in \{1,2,\ldots,K\}} UCB_i(t-1)$

Observe reward X_t

Update empirical mean and pull count of A_t

Empirical Performance

Cumulative regret curves



Why Does UCB Work?

Intuition

Suppose a suboptimal arm i is pulled many times already and it is also pulled at round t

- $UCB_{i*}(t) \geq \mu^*$ (with high probability)
- Arm $i \neq i^*$ is pulled $\Rightarrow$ $UCB_i(t) \geq UCB_{i*}(t)$ (UCB rule)
- If $T_i(t)$ is high, $UCB_i(t) \approx \mu_i$ (with high probability)
- $\mu^* > \mu_i$ (suboptimal arm)
- Contradiction! $\mu^* > \mu_i \approx UCB_i(t) \geq UCB_{i*}(t) \geq \mu^*$

A Case Study in Bandits

Blog

Using a Multi-Armed Bandit with Thompson Sampling to Identify Responsive Dashers

March 15, 2022 |  Arjun Sharma

DoorDash is a U.S. food-delivery platform, similar in spirit to Swiggy/Zomato

Problem Statement

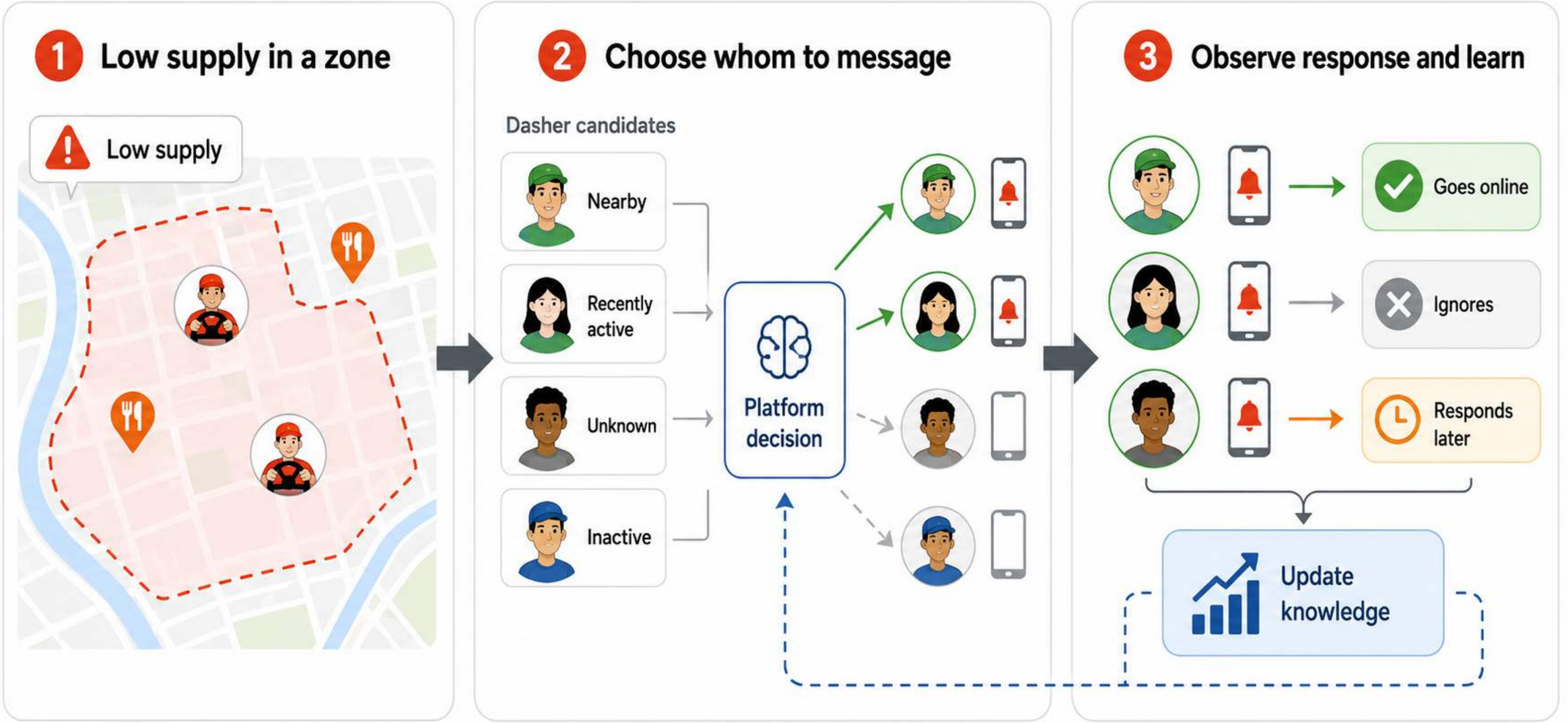
Persuade more dashers to start dashing!

- Orders arrive continuously
  - Arrival rate is outside of platform's control
- Need more drivers to be active
  - Incentivise through extra delivery fees

But messaging uninterested Dashers can create a bad user experience

Can we do better than alerting Dashers at random?

Problem Statement



Problem Statement

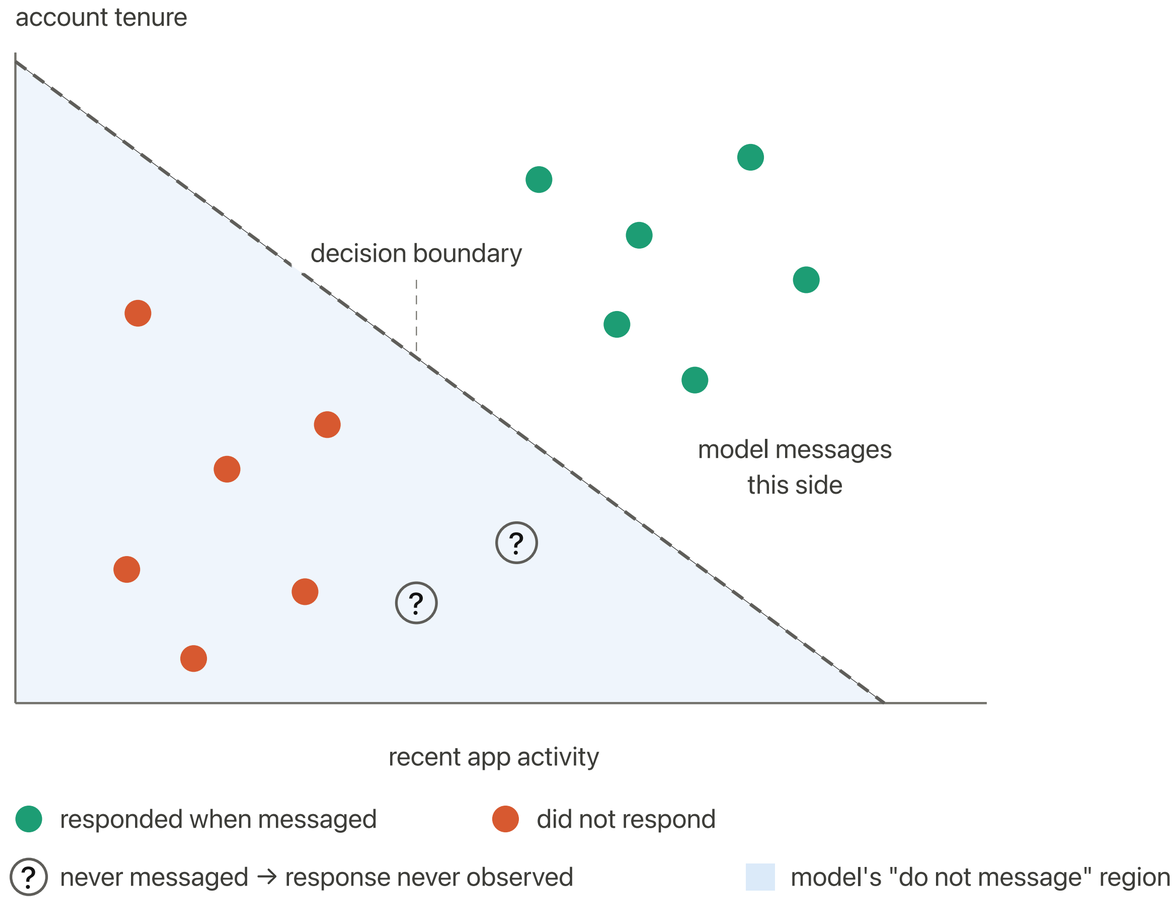
We need an ML approach that can

- **Identify** currently responsive Dashers
  Who is likely to sign on if messaged now?
- **Avoid** messaging uninterested Dashers
  Reduce spam / bad user experience
- **Discover** new responsive Dashers
  Do not overuse the same known responders

Does Supervised Learning Work?

Past data available with platform

- Whether a Dasher signed on after being messaged
- Context of the message: time, location, incentive, demand/supply
- Dasher features: recent activity, tenure, past responsiveness



This looks like a binary classification problem: predict response / no response

Problems with Supervised Learning

1. **Preferences drift over time**

   E.g., a college student: active during breaks, unavailable otherwise

   Static classifier bakes in past behaviour

   Past labels go stale

2. **No exploration**

   A self-fulfilling trap

   Model messages only “likely responders”

   Unsure Dashers never get tested

   The model never learns what it avoids

Can we formulate this as a multi-armed bandit problem?

Modeling As A Bandit Problem

- **Arm:** A prospective dasher to message
- **Action:** message dasher i
- **Reward:** signs on soon $\Rightarrow X_t = 1$ (or not $\Rightarrow X_t = 0$)
- **Feedback:** only observed for dashers we message

Ignoring for now: important contextual information such as demographic of dasher, demand v/s supply, etc.

Objective: Message likely responders (exploit) while testing uncertain Dashers (explore)

Regret minimisation, not best-arm identification!

Algorithms DoorDash Considered

ε-greedy

✓ Easy to implement

- Fixed exploration rate

UCB

✓ Smooth exploration-exploitation tradeoff

- over-messages brand-new Dashers (huge bonus on cold arms)

- Difficult to interpret decisions

Thompson Sampling

✓ Easy to implement

✓ Same benefits as UCB

✓ Interpretable: counts the successes and failures of messages sent to each Dasher

- Cold-start behaviour needs fine-tuning

Thompson Sampling

Sampling From Uncertainty

- **UCB:** keeps an interval for each μ_i
  - Picks the arm with the largest upper confidence bound
- **Thompson Sampling:** maintains a *belief distribution* for each μ_i
  - More data → narrower belief; less data → wider belief
- At each round, sample one plausible value from each belief

$$\tilde{\mu}_i(t-1) \sim f_i(\mu; t)$$

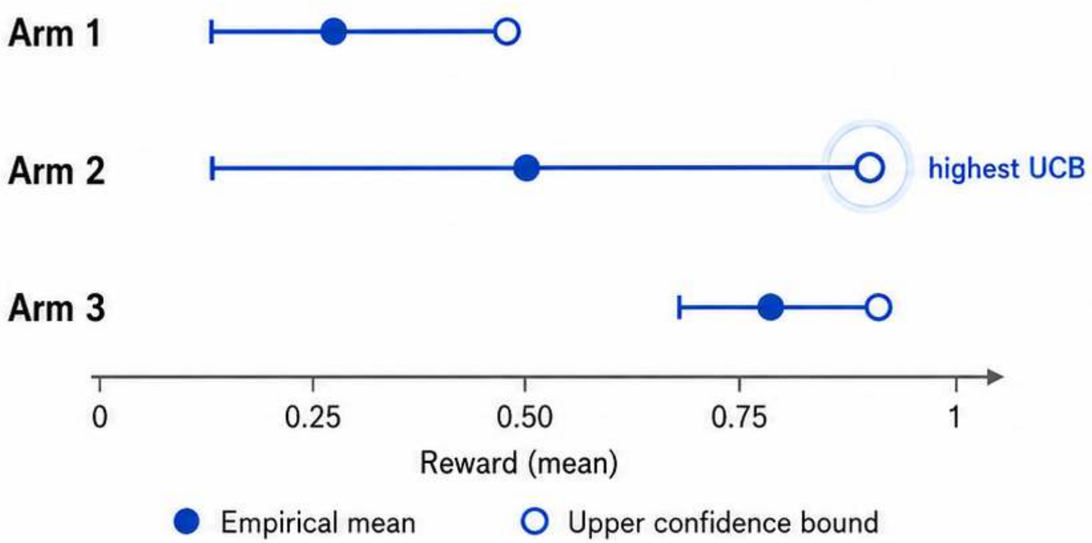
- Pick the arm with the largest sampled value:

$$A_t = \arg \max_{i \in \{1,2,\ldots,K\}} \tilde{\mu}_i(t-1)$$

UCB vs Thompson Sampling

UCB

Choose the highest upper confidence bound

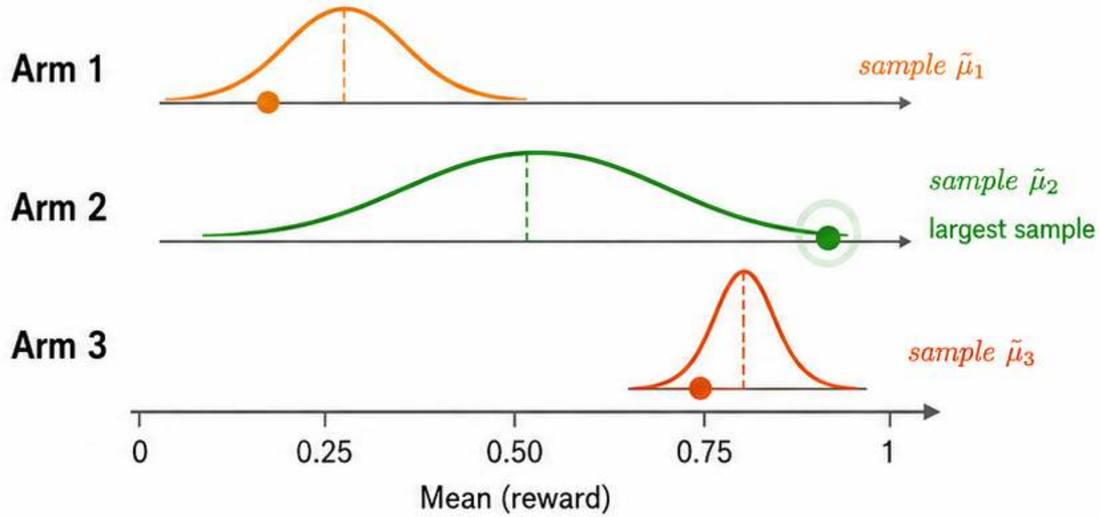


$$UCB_i(t) = \hat{\mu}_i(t) + bonus_i(t)$$

Deterministic: pick the arm with the largest plausible value.

Thompson Sampling

Sample one plausible mean from each belief



$$\tilde{\mu}_i(t) \sim f_i(\mu; t), \quad A_t = \arg\max_i \tilde{\mu}_i(t)$$

Randomized: pick the arm with the largest sampled plausible value.

UCB uses the top of each uncertainty range. Thompson Sampling draws one plausible value from each uncertainty distribution.

Thompson Sampling

Beta-Bernoulli Model

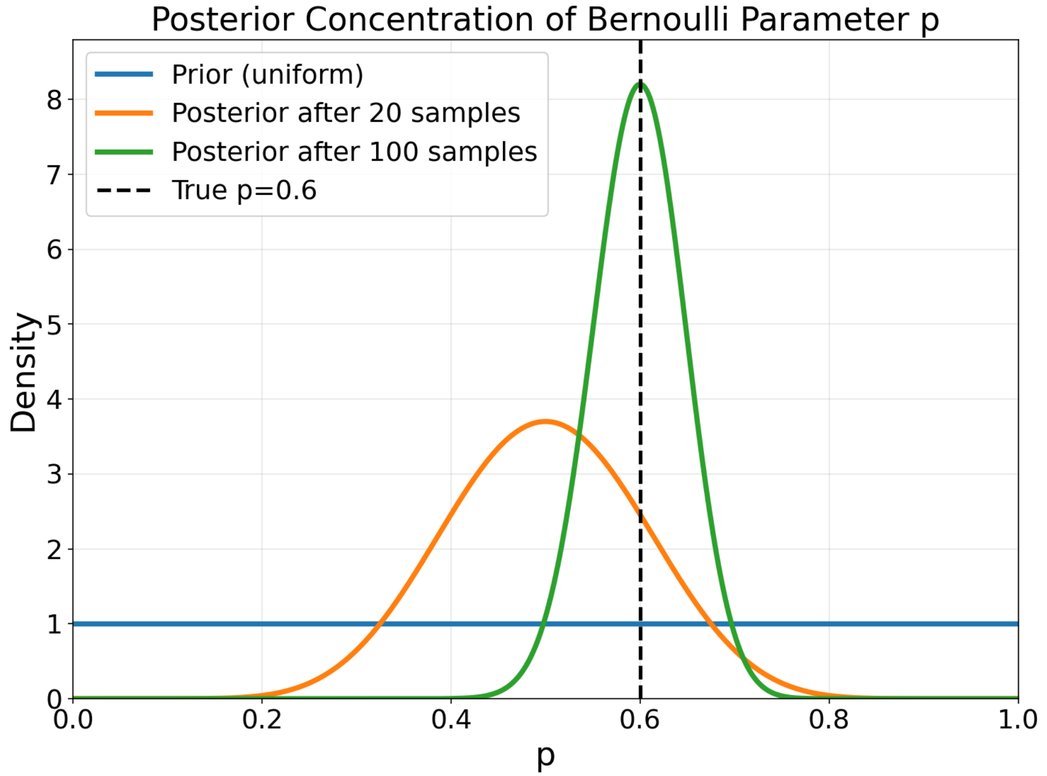
- For binary rewards, the unknown mean is a probability

$$\mu_i = \mathbb{P}(X = 1 \mid \text{arm } i)$$

- We maintain a belief distribution over μ_i
  - Using a Beta distribution

$$\tilde{\mu}_i \sim \text{Beta}(\alpha_i, \beta_i)$$

$$X = 1 : \alpha \leftarrow \alpha + 1,\ X = 0 : \beta \leftarrow \beta + 1$$



Thompson Sampling

Pseudocode

Initialise: $\alpha_i = \beta_i = 1$ for all arms i

For $t = 1,2,\ldots,T$

1. For each arm i:

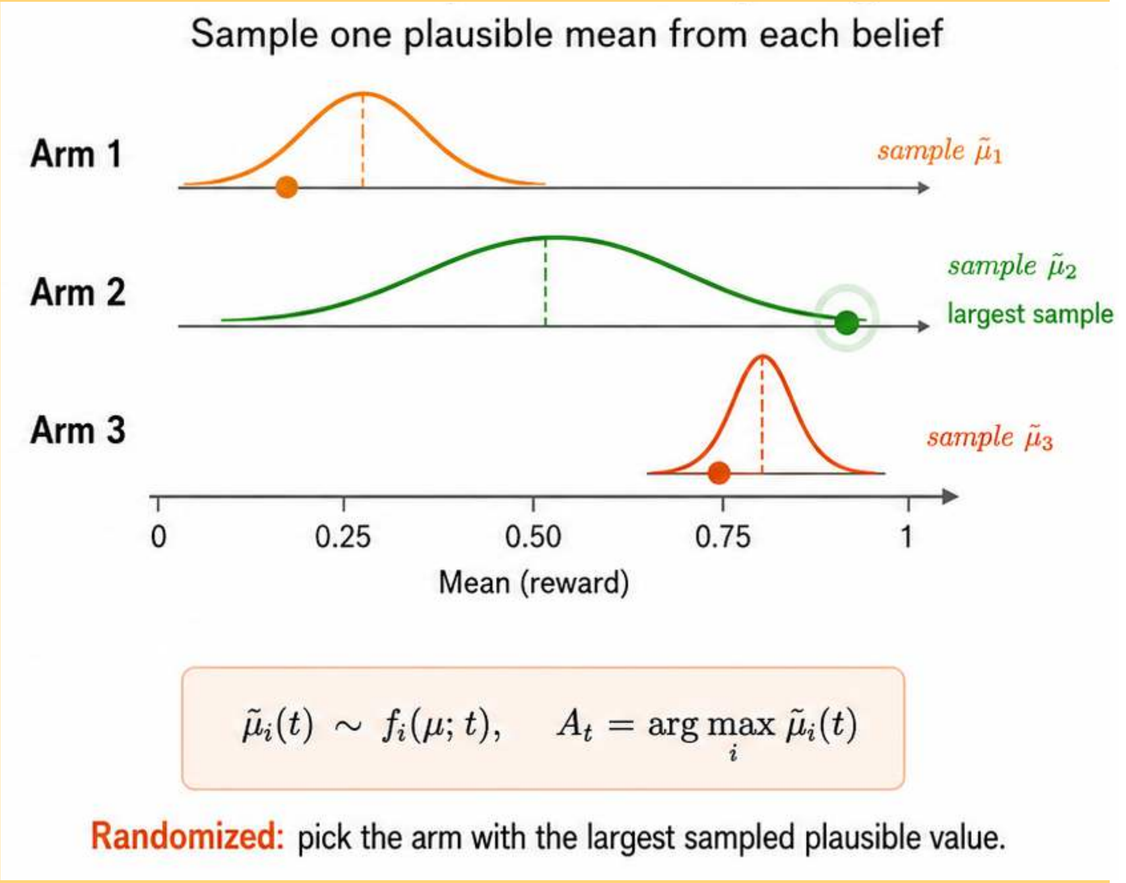
$$\text{Sample } \tilde{\mu}_i \sim \text{Beta}(\alpha_i, \beta_i)$$

2. Pull arm:

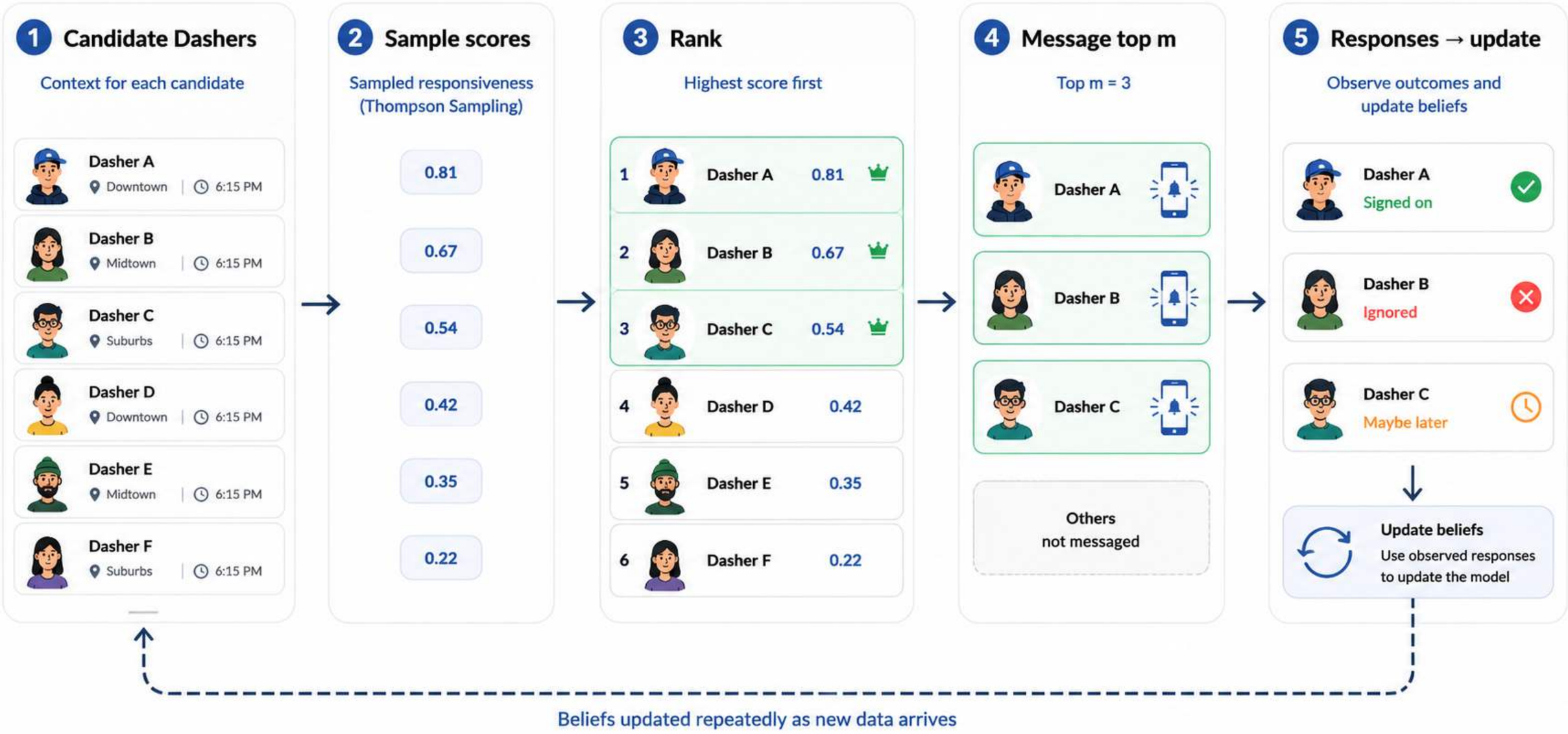
$$A_t = \arg \max_{i \in \{1,2,\ldots,K\}} \tilde{\mu}_i(t-1)$$

3. Observe reward $X_t \in \{0,1\}$ and update:

$$\text{if } X_t = 1\text{: } \alpha_{A_t}\text{++, else: } \beta_{A_t}\text{++}$$



System Overview from DoorDash



Practical Details from DoorDash

Making Thompson sampling work in production

1. Cold-start prior

- Instead of starting every new Dasher at $\alpha=\beta=1$, we can use information from similar Dashers (location / time / demographic / recent behaviour).

2. Non-stationarity

- Old behaviour does not accurately reflect current availability. We need to down-weight older observations so recent responses matter more.

3. Batching updates

- In production, update in batches rather than after every single datapoint. The right update frequency depends on infrastructure and drift.

The algorithm is the easy part. The prior, the decay, the window:

that tuning layer is where every production bandit actually lives

Coming Up Next

A Deeper Understanding of Thompson Sampling

- Bayesian viewpoint
- Prior and posterior distributions
- Thompson Sampling simulation plots

Also, a discussion of HW1

Link to DoorDash blog: https://careersatdoordash.com/blog/using-a-multi-armed-bandit-with-thompson-sampling-to-identify-responsive-dashers/