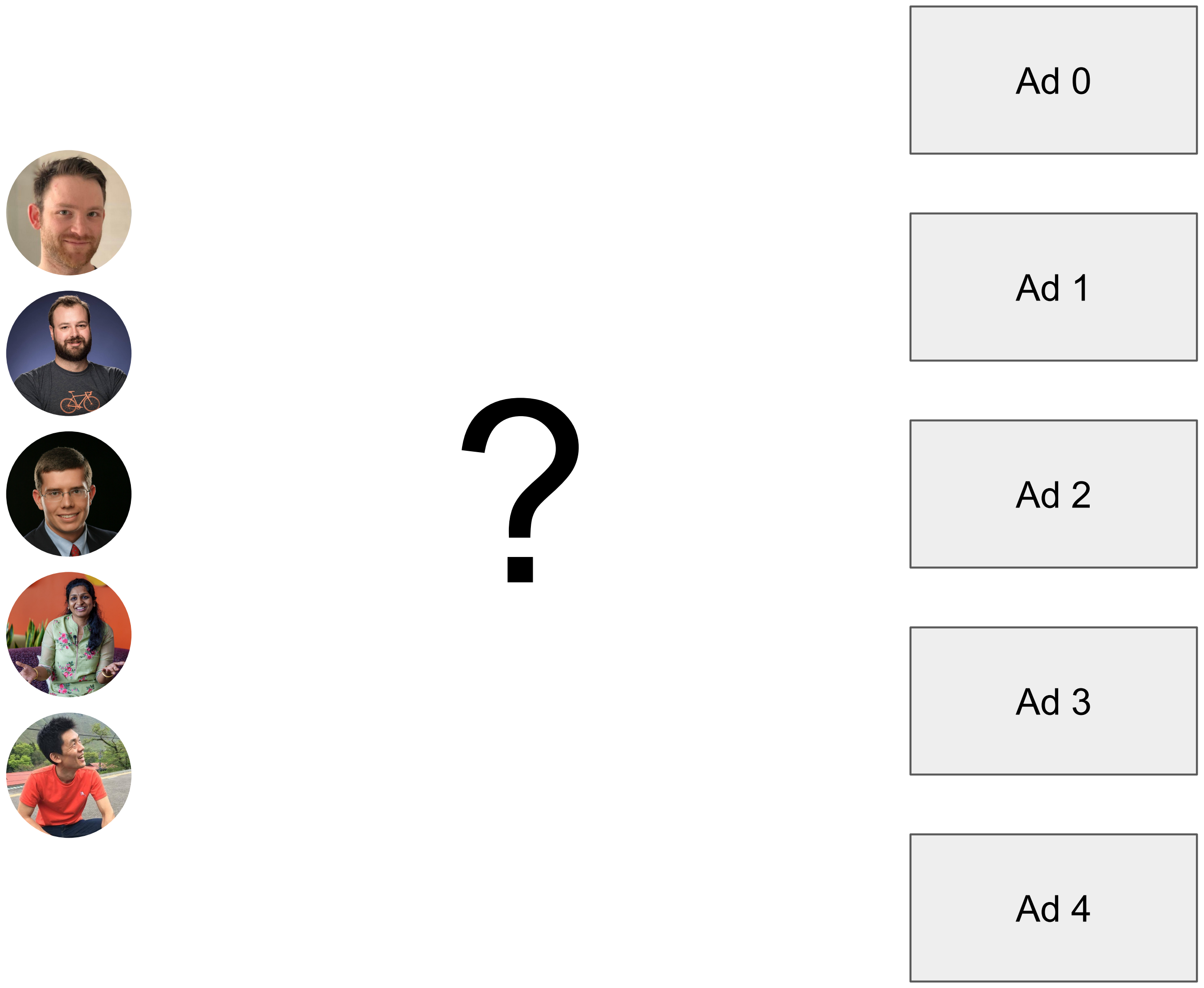
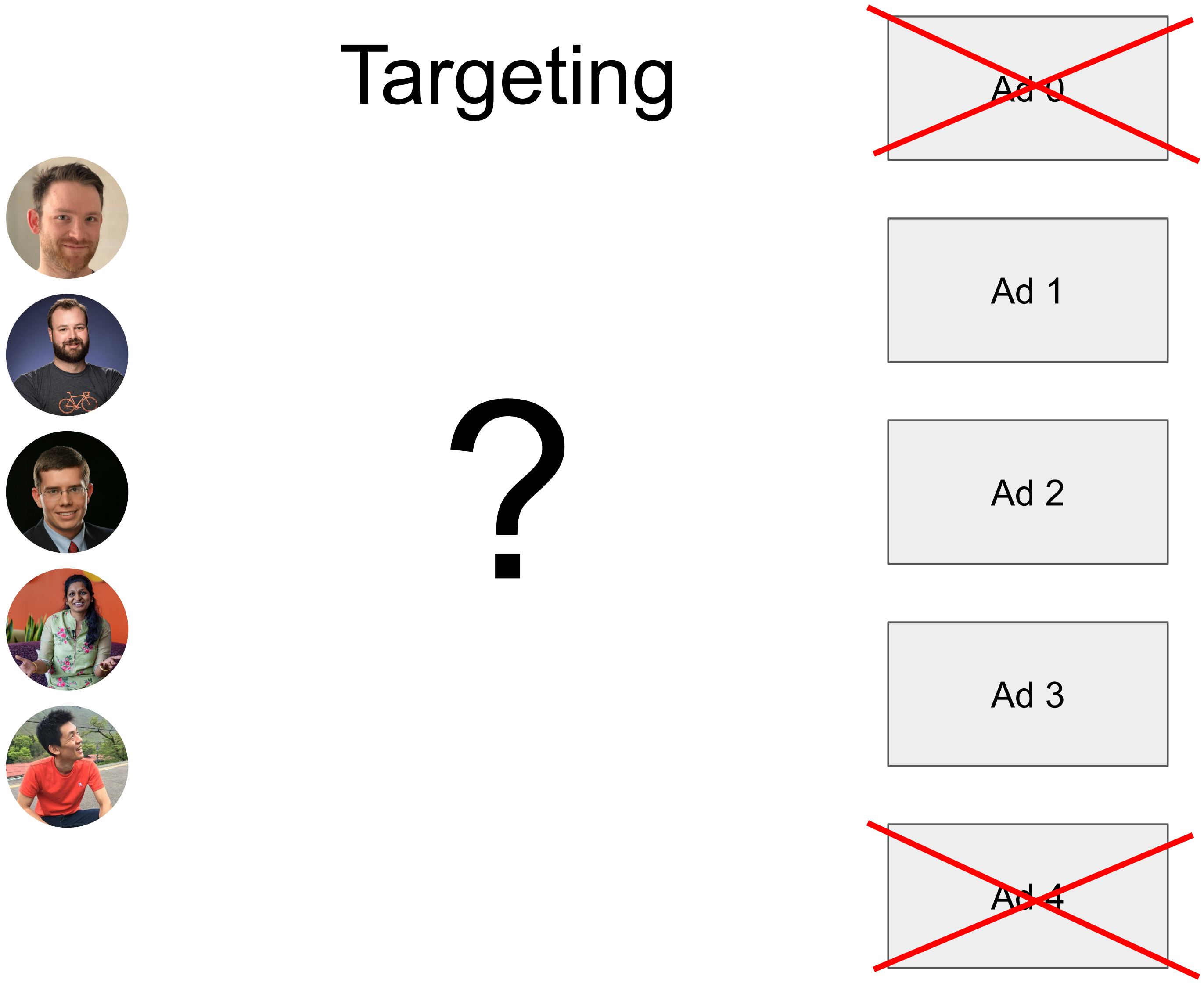
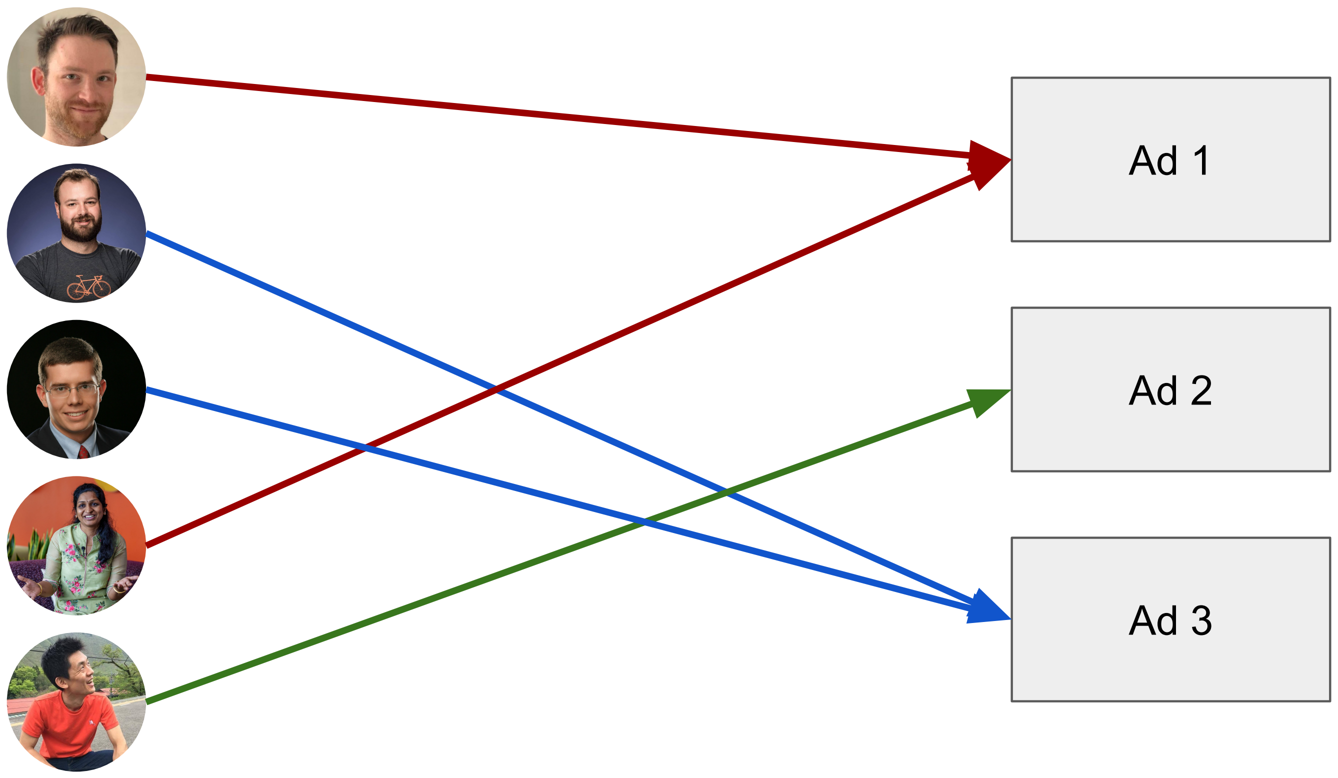
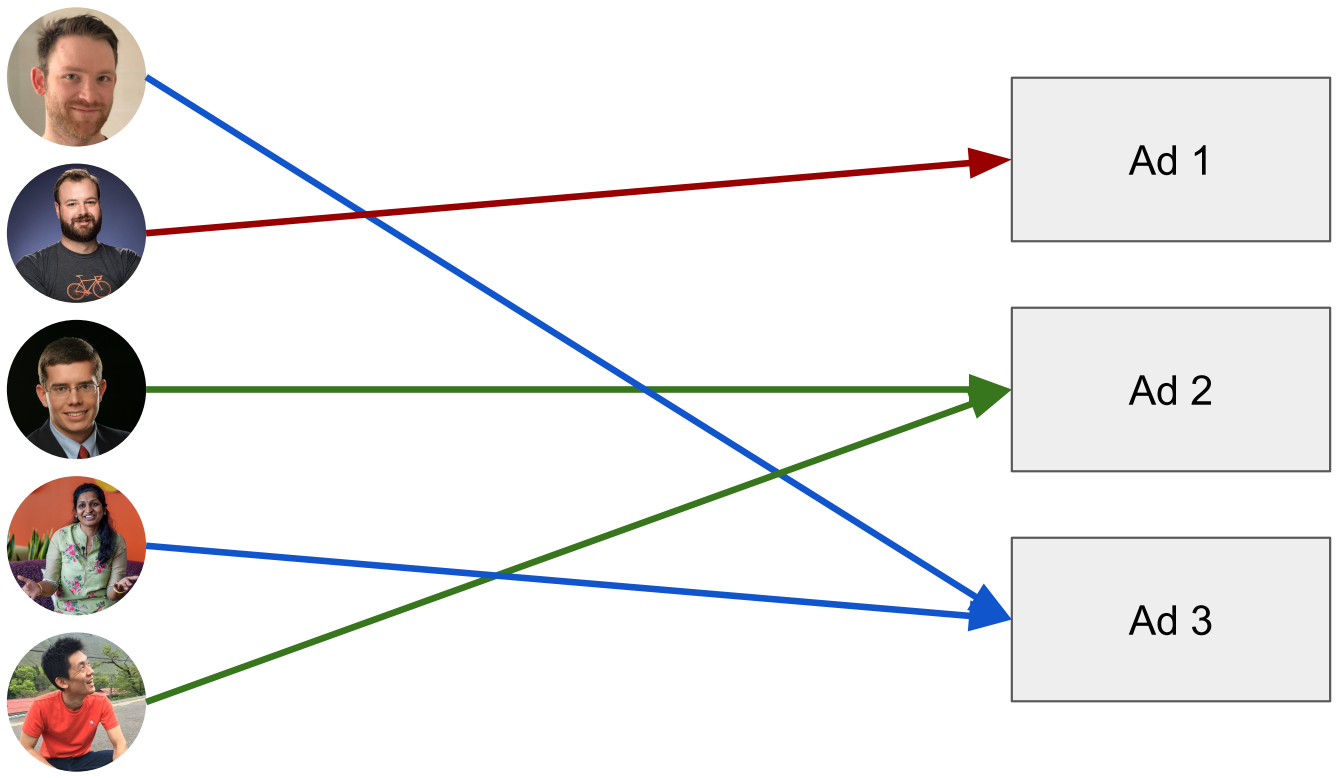
Motivating use-case:
Ad Selection

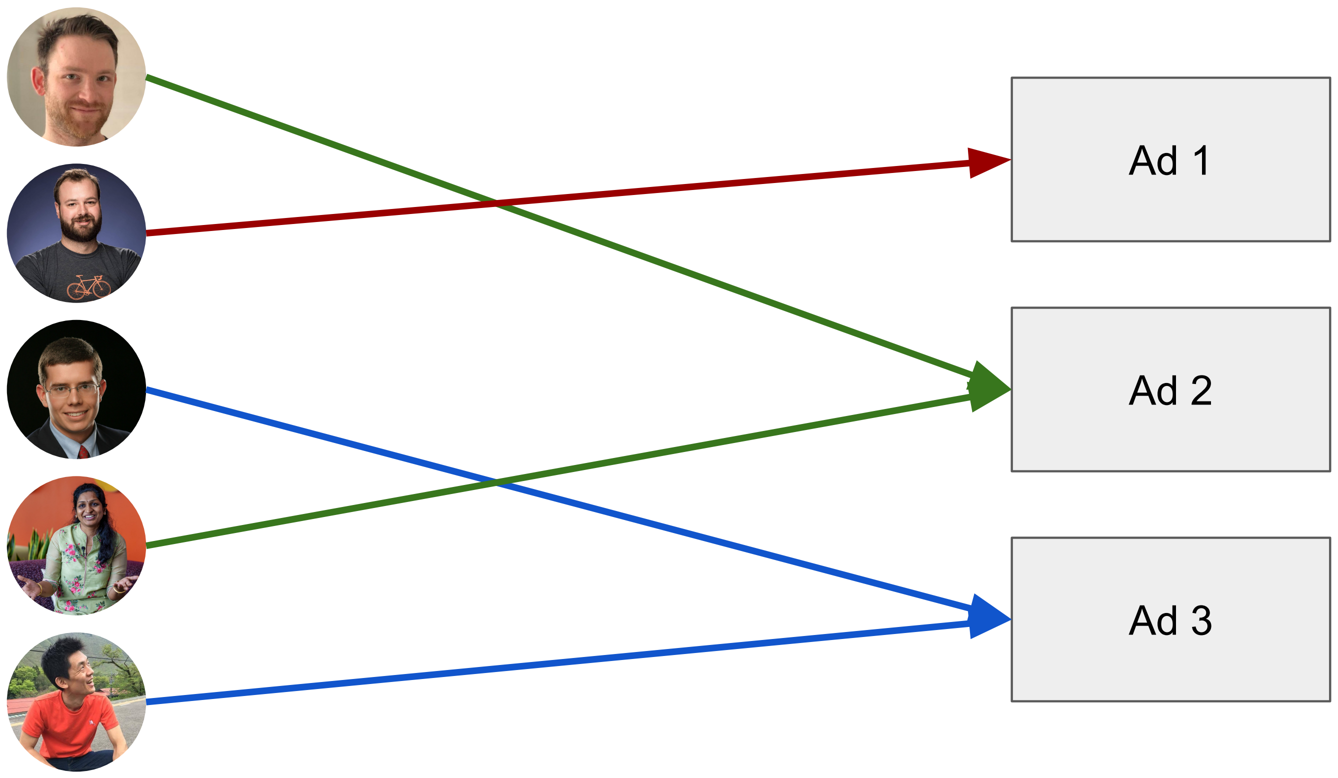

?
Ad 0
Ad 1
Ad 2
Ad 3
Ad 4

Targeting

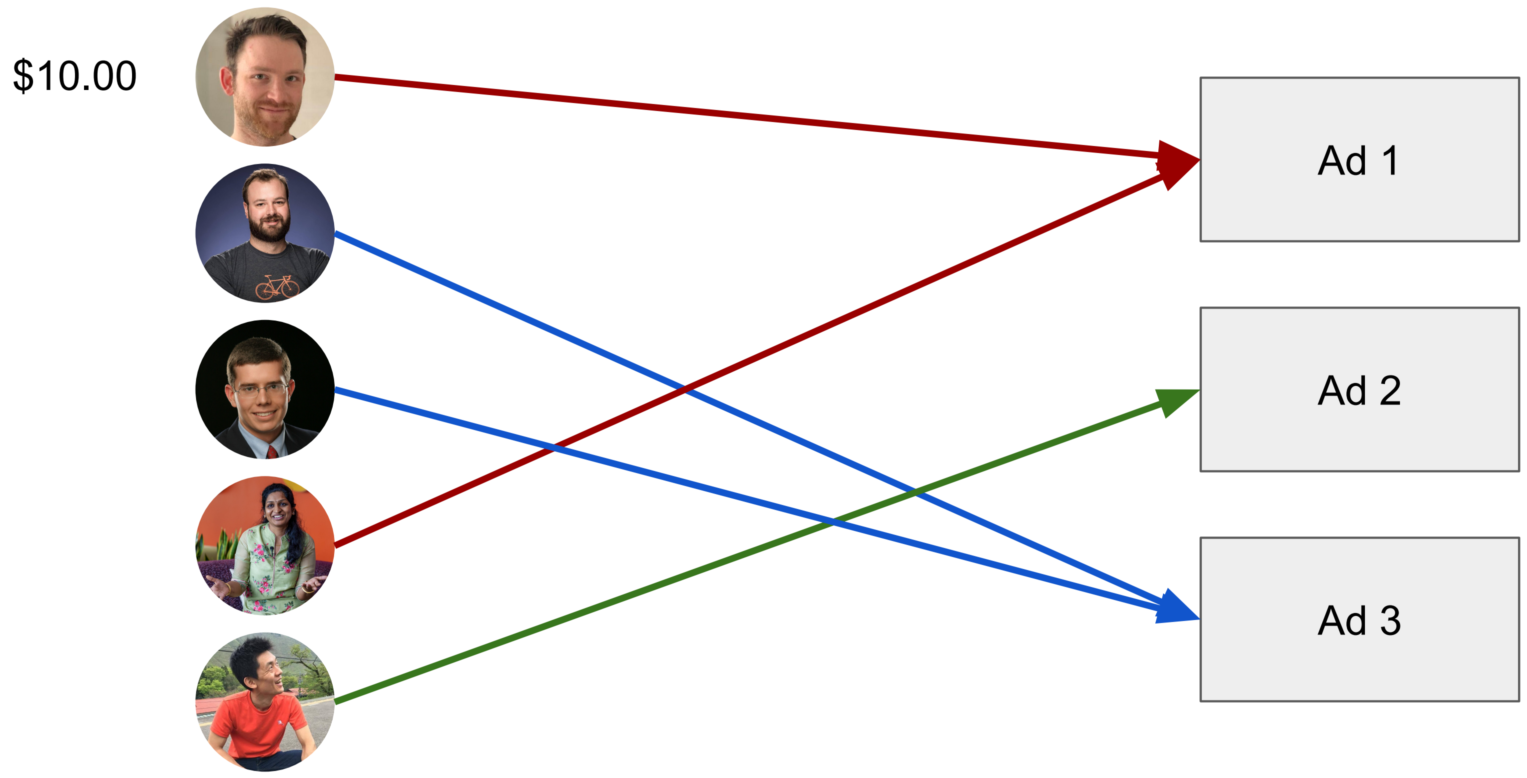


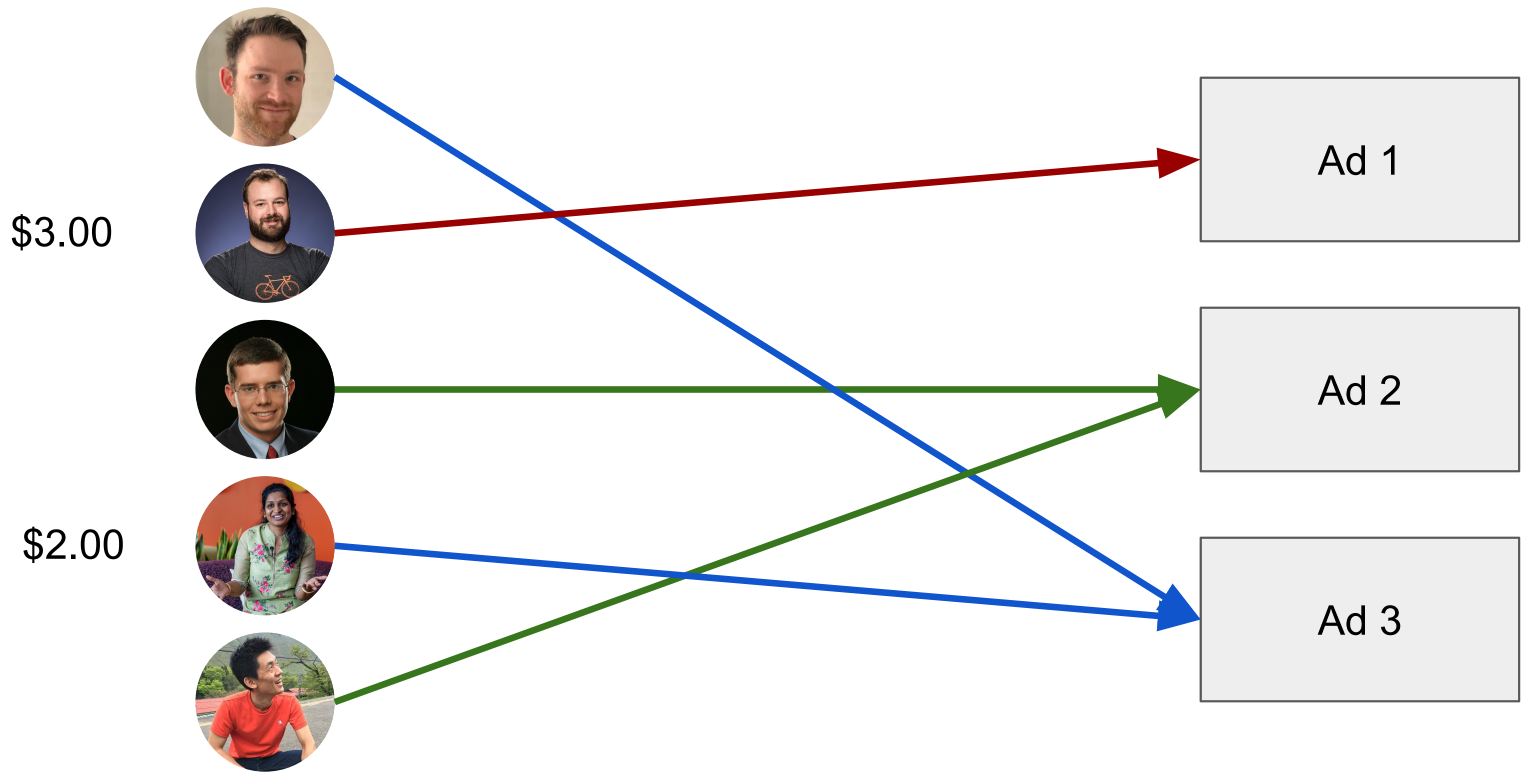

Ad 1
Ad 2
Ad 3

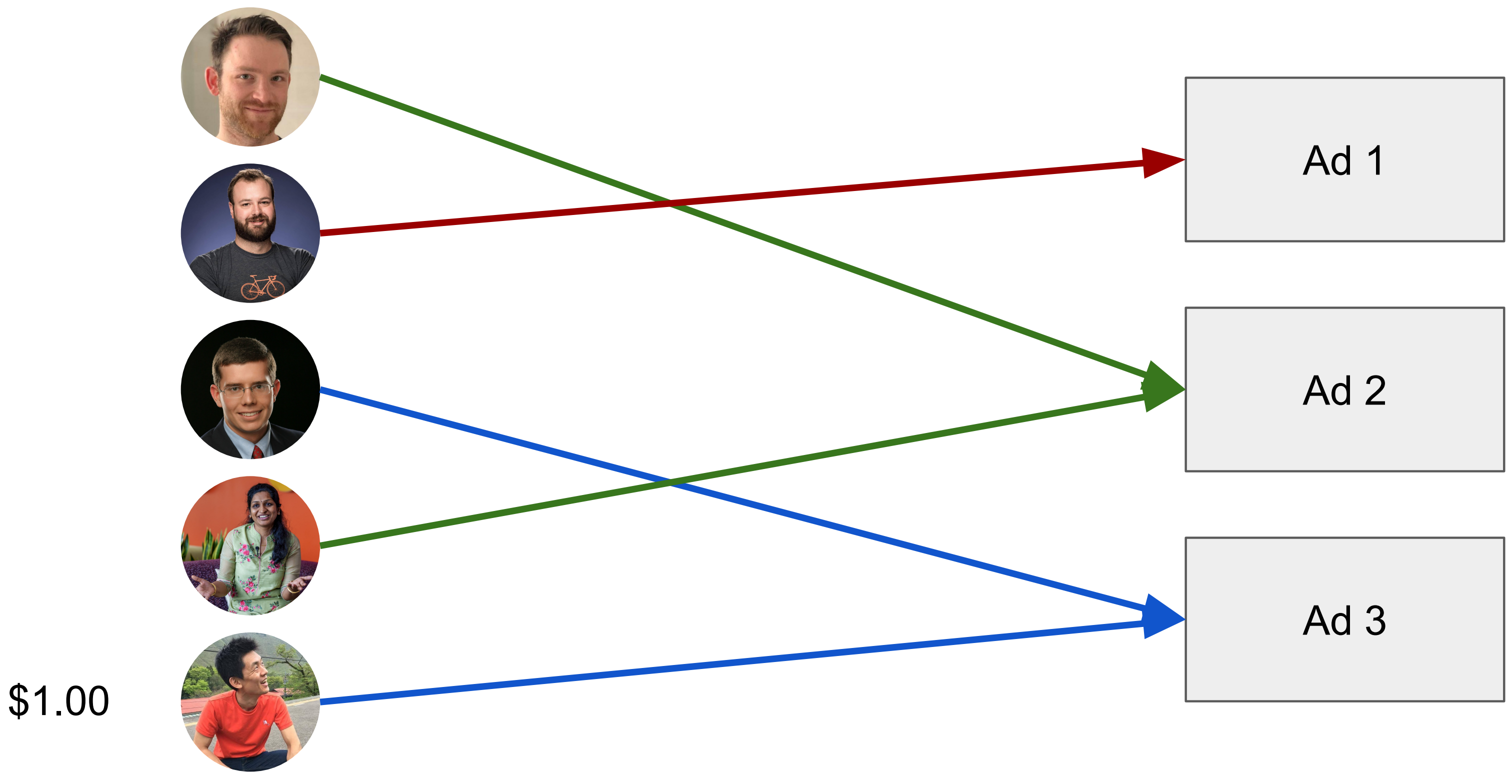

Ad 1
Ad 2
Ad 3


Ad 1
Ad 2
Ad 3

Which one is the best?

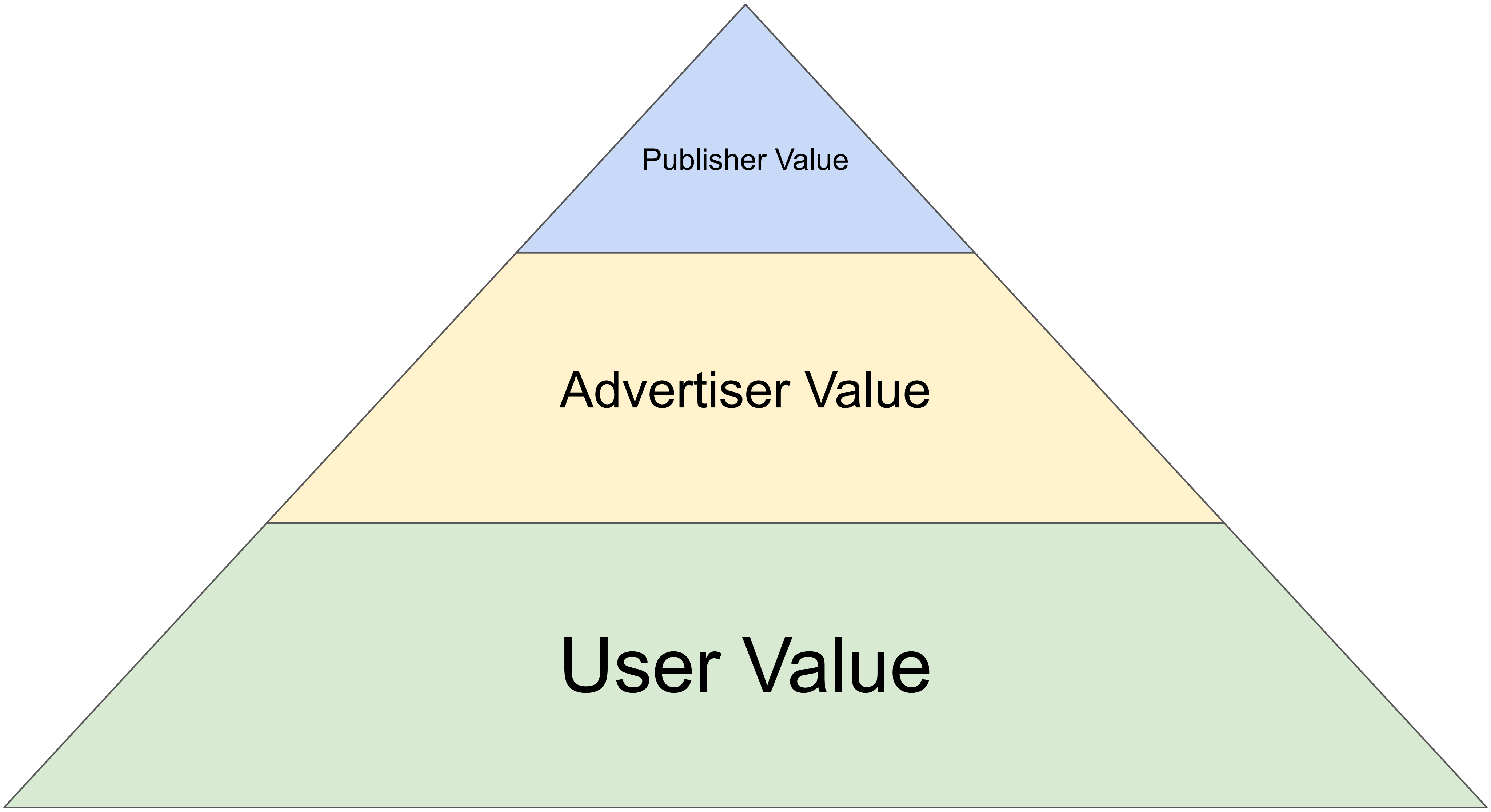

$10.00
Ad 1
Ad 2
Ad 3


$3.00
$2.00
Ad 1
Ad 2
Ad 3


Ad 1
Ad 2
Ad 3
$1.00

Maximize value creation

Value for whom?


Publisher Value
Advertiser Value
User Value

Value for whom?

- Ideal: user-value
    - Hard to measure
- Usable proxy: advertiser-value
    - Ask them how much they’re willing to pay for this event
    - Hopefully a *lower-bound* on user-value
        - User’s expected value > $ spent
        - $ spent > advertiser profit
        - Advertiser profit > $ willing to pay per conversion
    - We can add some guardrails too
        - Surveys
        - Report bad ads

How to achieve this?

- Each time we need to select an ad:
  - Loop over all ads and compute:
    - E(value) = E(conversion) * value(conversion)
  - Choose the ad with the highest expected value
- What’s the expected likelihood of a conversion?
  - If we show this ad, to this person, in this context, estimate the chances it will result in a conversion.

Estimating conversion likelihood

- We need ***a function***
    - At advert selection time, we want to loop over all the ads and invoke this function on each:
        - Arguments:
            - Ad features
            - Contextual features
            - User features
        - Output:
            - Prediction of the likelihood this advert will lead to a conversion event.
        - Example:
            - `F(ad`$_1$`, cxt, usr) = 0.01`
            - `F(ad`$_2$`, cxt, usr) = 0.03`
            - `F(ad`$_3$`, cxt, usr) = 0.05`

Constraints:

1. Just learn about ***general trends in the data***. Do NOT memorize information about specific individuals.
2. Not totally rubbish performance
3. Single-digit epsilon budget

The general plan of how to get there (Part 1):

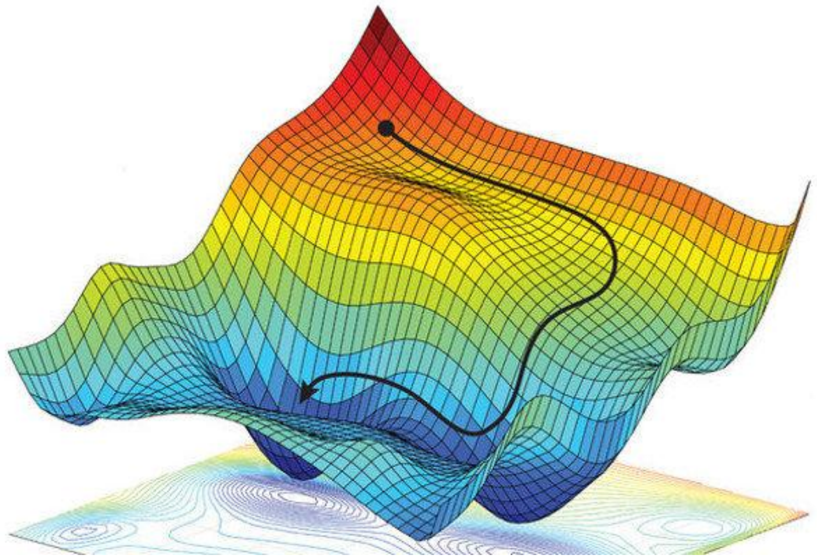
- Look at thousands of rows of historical data. Each row has:
    - Features:
        - Ad features (of the ad that was shown)
        - Contextual features (from where the ad was shown)
        - User features (of the user to whom that ad was shown)
    - Labels:
        - Ground truth about what actually happened. Did this ad lead to a conversion or not? Yes or no? 1 or 0?

The general plan of how to get there (Part 2):

- Find a function that does a not-terrible job at predicting the historical labels
    - For any candidate function, let’s compare the predictions it would make with the ground truth
    - Let’s define a “Loss Function” which gives us a score of how far off the predictions were across all the historical data.
    - Let’s tweak the function to try to minimize this “Loss Function”

The general plan of how to get there (Part 3):

- “Gradient Descent”
  - Look at the “slope” of the loss function where we are right now.
  - Just “roll downhill”.
  - The “Gradient” is the “downhill” direction.
    - Tweaking the model parameters in that direction will make the loss function go down.
  - Do that over and over again until you stop going downhill. Hope that’s good.



We have some options:

1. Release “noisy labels” (event-level)
    a. Use a “randomized response” approach. That is, p% of the time return a random label, and (1-p)% of the time release the true label.
2. Release “noisy labels” somehow capped at a user-level
    a. Similar to above, but not at an “event-level”, more at a “user-level”, so that we can provide some kind of “sensitivity bound” at a user-level.
3. Logistic regression in MPC
    a. This is a really, really simple “model”. It lends itself to MPC fairly well.
4. DP-SGD in MPC
    a. Differentially Private Stochastic Gradient Descent. A more complex algorithm which can be used to train low-to-medium complexity neural networks.

1. Noisy Labels (event level)

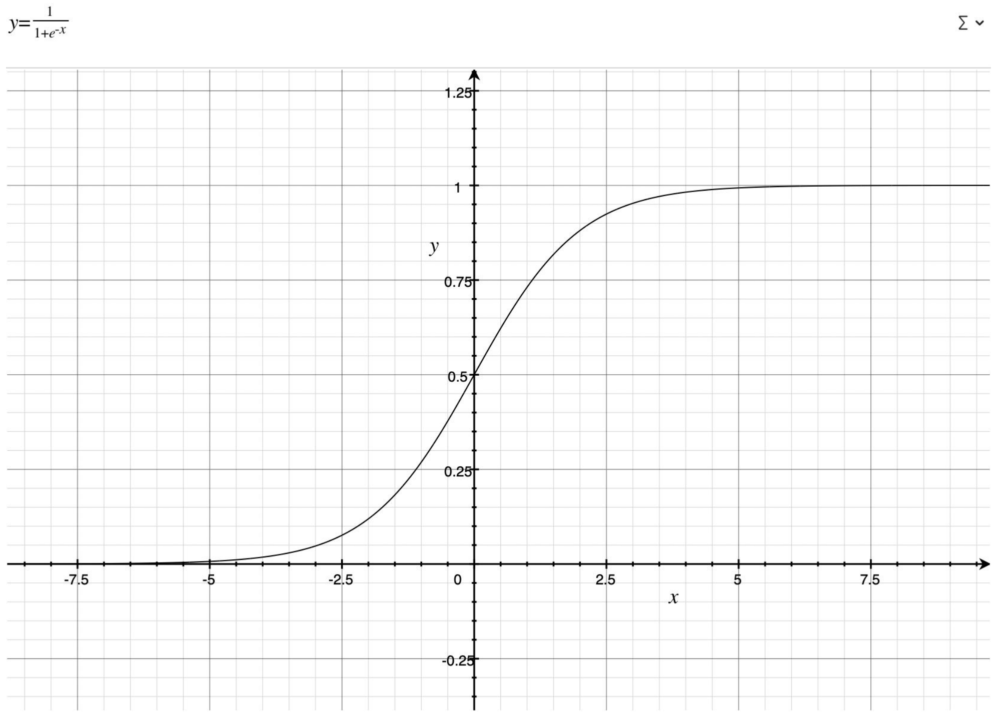
- We can select a value of epsilon, and that determines the percentage of the time that we return a random label.
- Pros:
  - We perform the absolute minimum amount of processing in MPC. We can continue to use all available techniques to train complex models on data in the clear.
  - We ***know*** this approach works (epsilon in the 2 to 3 range-ish?)
  - We could launch this today, and businesses could have continuity
- Cons:
  - “Measurement of single events”. We cannot make claims about “aggregation”.
  - There is no user-level sensitivity bound. Users who generate more events have more data loss, potentially unbounded.
  - Model explainability is hard. We can’t really say much about the models this is used in as we can’t limit them at all. Hard to tell a story to end-users about the inferences that can be formed.

2. Noisy Labels (user level)

- Similar to the last one, but with some (unspecified) mechanism to eventually cap the information released about a particular person.
- Pros:
  - We perform only a bit more than the minimum amount of processing in MPC. The user-level capping is somewhat expensive, but it’s probably tractable.
  - We know this approach does not work as well as the last one, but I’m optimistic that we can do something reasonable.
  - We could launch this today, and businesses could have *reasonable* continuity
  - We can articulate a user-level sensitivity bound.
- Cons:
  - “Measurement of single users”. We cannot make claims about “aggregation”.
  - Models trained on this type of data perform much less well than the previous option.
  - Model explainability is hard. We can’t really say much about the models this is used in as we can’t limit them at all. Hard to tell a story to end-users about the inferences that can be formed.

3. Logistic Regression in MPC

- The model is super simple:
  - We just make a “score” that’s a simple linear combination of all the features:
  - `score = c`$_1$`x`$_1$` + c`$_2$`x`$_2$` + c`$_3$`x`$_3$` + …`
  - Then we just put this score into a function that maps:
    - Large positive values ~= 1
    - Large negative values ~= 0
  - Let’s use the “logistic function”
  - `σ(x) = 1/(1 + exp(-x))`



3. Logistic Regression in MPC (continued…)

- Our task:
    - Find values of c_1, c_2, c_3, `etc` that minimize the loss function.
    - The output of training this “model” is just these weights.
    - We can add a bit of noise to them to make them differentially private.
    - This output is identical to the current output format of both IPA and the ARA-aggregate;
        - a noisy histogram.

3. Logistic Regression in MPC (continued…)

- Pros:
  - This is a very simple model that’s easier to understand, and explain than a neural network
    - It clearly is just learning about correlation between features and outputs;
      - Is the correlation positive or negative or zero?
      - Is the correlation strong or weak?
  - We can basically do this with just a single dot-product in MPC. Very cheap and fast.
  - We could build this into IPA really easily and start experimenting with it.
- Cons:
  - This is a super simple model. It will not perform anywhere near as well as modern, deep neural networks.
  - Unknown privacy ⇔ utility curve

4. DP-SGD in MPC

1. Randomly select a small subset of the training data (a mini-batch)
2. For each row in the mini-batch:
    a. Compute the gradient of the loss function given the current values of the model parameters.
    b. “Clip” this gradient, so that it has maximum magnitude C.
3. Sum up these clipped gradients
4. Add some random noise (proportional to C)
5. Update the model parameters using the noisy, clipped, aggregated gradient.
6. Repeat

What part is done in the MPC?

	Just Aggregate in MPC (no mini-batches)	**All in MPC (incremental releases)**	**All in MPC (one single release)**
Batch Size	All rows at once	Random smallish batches	Random smallish batches
Compute Gradient	Outside MPC	In MPC	In MPC
Clip Gradient	Outside MPC	In MPC	In MPC
Add Random Noise	In MPC	In MPC	In MPC
Update model params	Outside MPC	Outside MPC	In MPC
Total Processing Cost	Low	High	High
Privacy Budget Cost	High	Medium	Low?

4. DP-SGD in MPC (continued…)

1. Pros:
    a. Can (potentially) be used to train more complex models than Logistic Regression, like smallish to medium sized neural networks.
    b. We can be confident it’s not memorizing data about individuals, and is only learning about general trends.
    c. Maybe decent utility for a given privacy budget. Potentially pretty good if you pull it ALL into the MPC, *but we don’t have a formal proof of this*.
2. Cons:
    a. Not very easy to do this in MPC. Likely to require a ton of communication between the helpers which is expensive.
    b. Might not converge for large and complex neural networks.
    c. No business continuity. This is a totally different ML training paradigm.
    d. We don’t really know if this will work. Path ahead unclear.
    e. Model explainability much harder for neural networks.

Discussion Topics

1. Do we ***want*** to support the conversion optimization use-case?
2. If so; how do we feel about
    a. “noisy labels” (event-level)?
    b. “noisy labels” (user-level)?
    c. Logistic regression?
    d. DP-SGD?

Privacy Preserving ML

How it could work