CS150: Database & Datamining

Lecture 27: Analytics & Machine Learning IX

Xuming He

Spring 2019

Acknowledgement: Slides are adopted from the Berkeley course CS186 by Joey Gonzalez and Joe Hellerstein, Stanford CS145 by Peter Bailis, Google talk by Jakub Konecny.

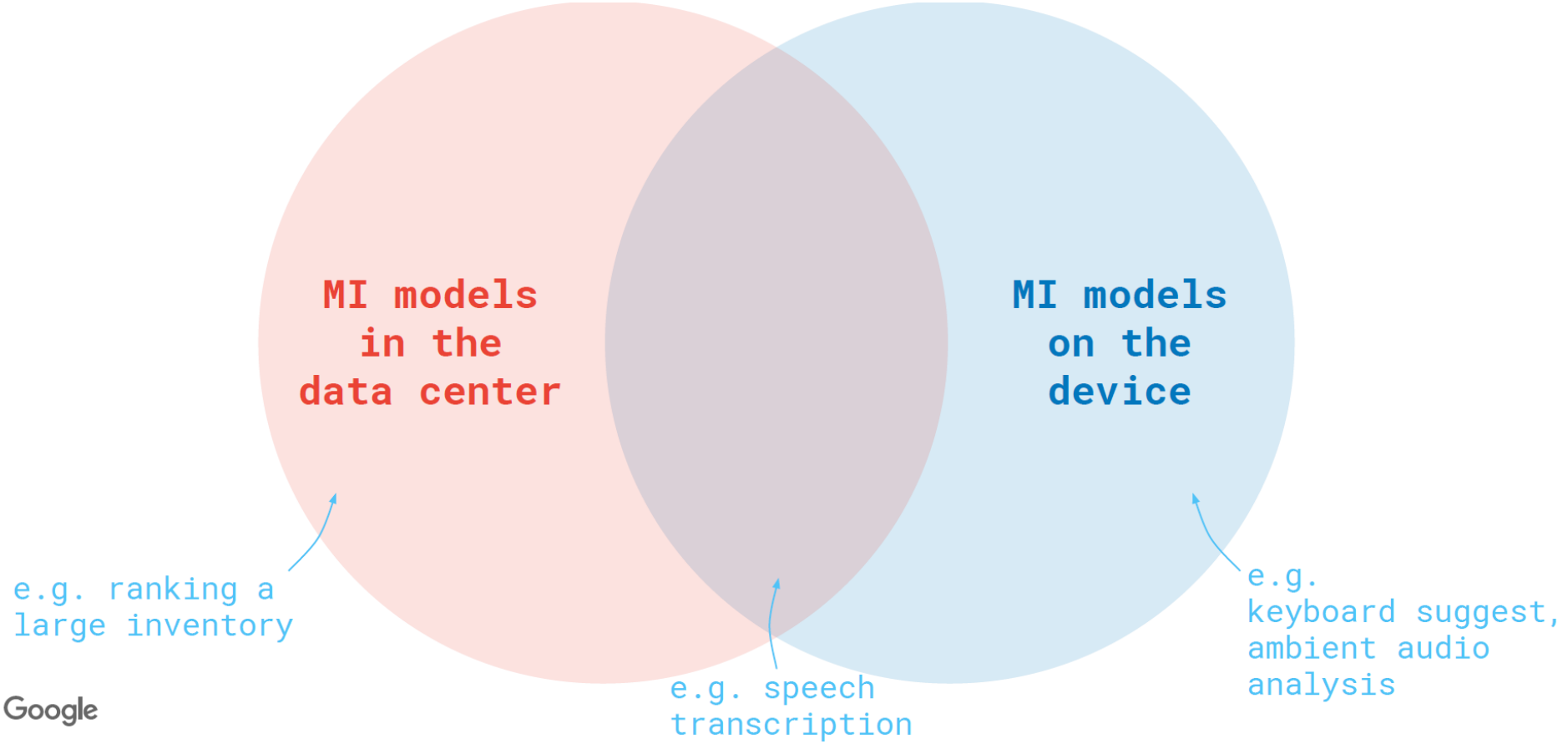
Distributed Deep Learning: Federated learning

What is Federated Learning?

- Goal: Imbue mobile devices with state of the art machine learning systems without centralizing data and with privacy by default

- Deep learning:
  - non-convex
  - millions of parameters
  - complex structure

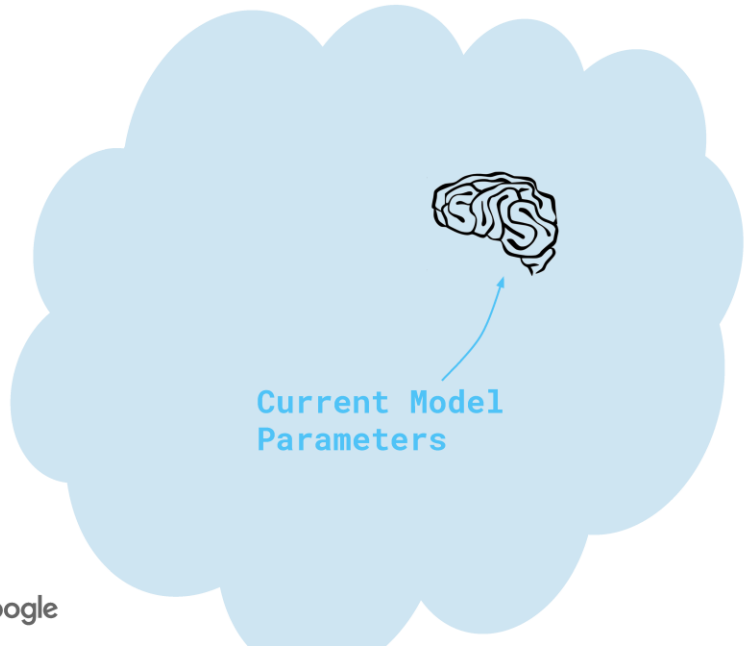
Machine Intelligence for Mobile Devices



Google

Cloud-centric ML for Mobile

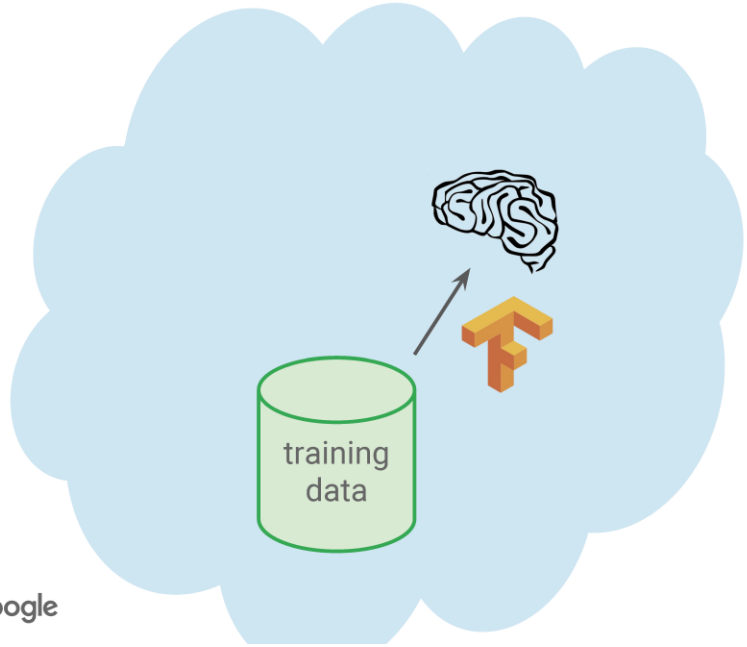
The model lives in the cloud.



Google

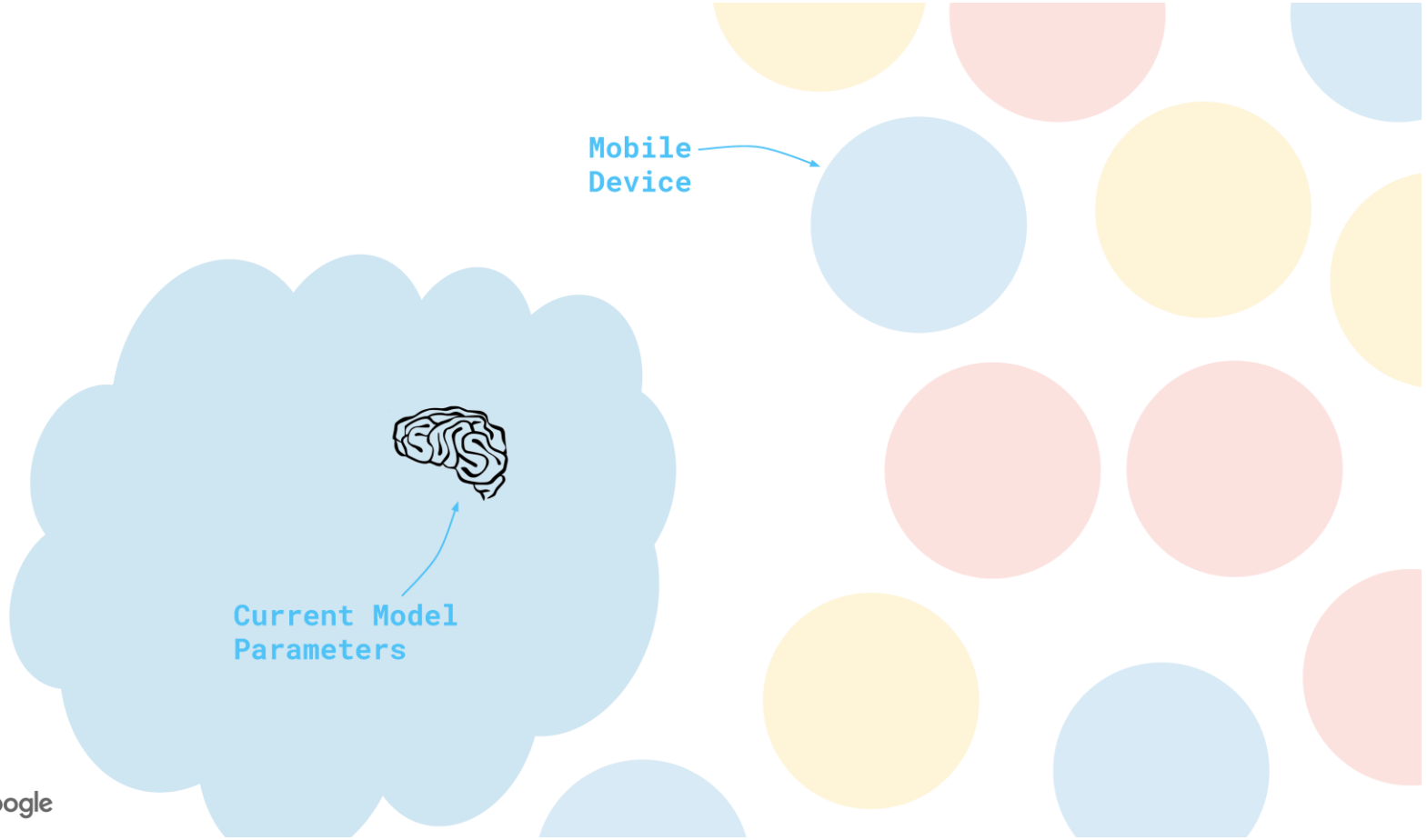
Cloud-centric ML for Mobile

We train models in the cloud.



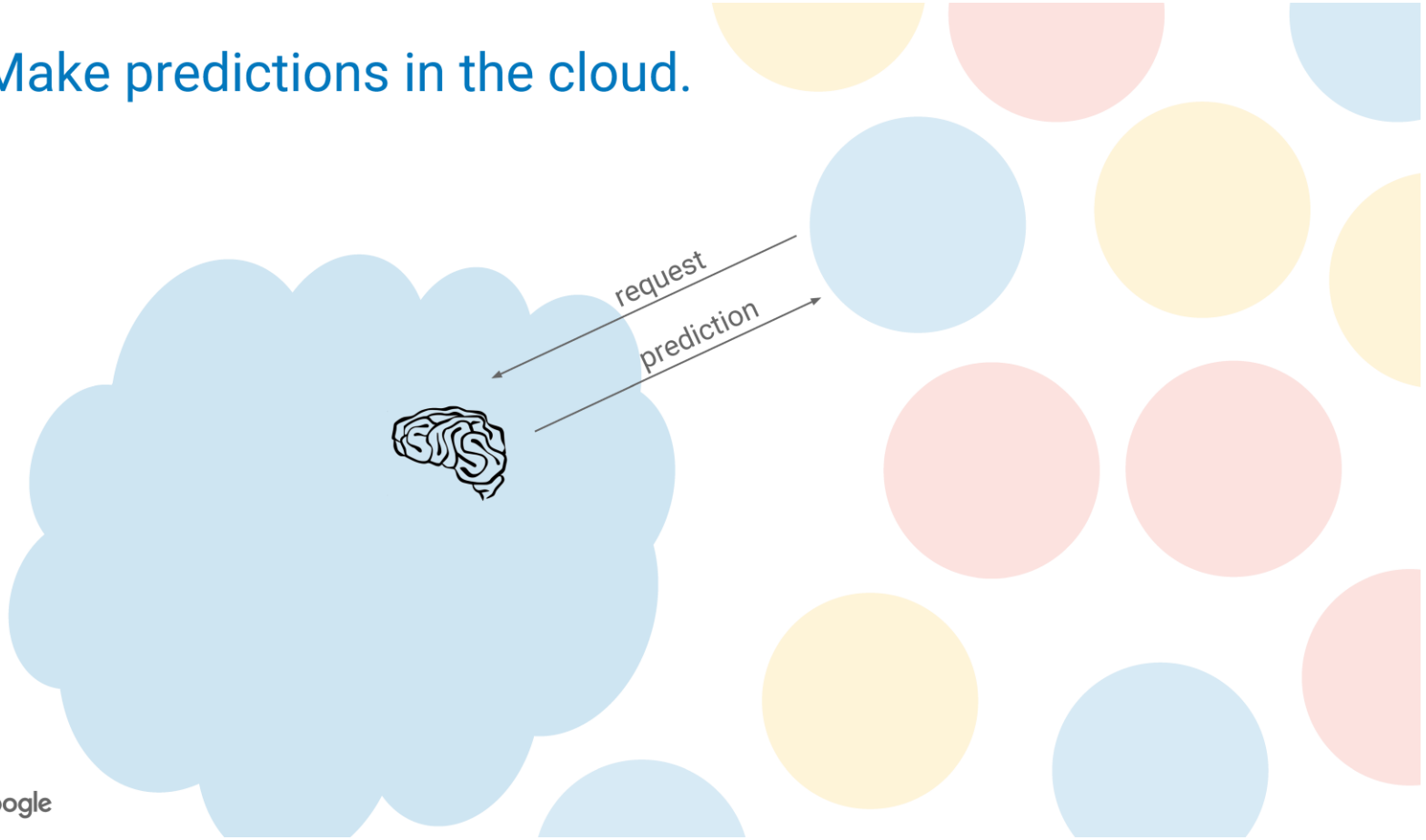
Google

Cloud-centric ML for Mobile



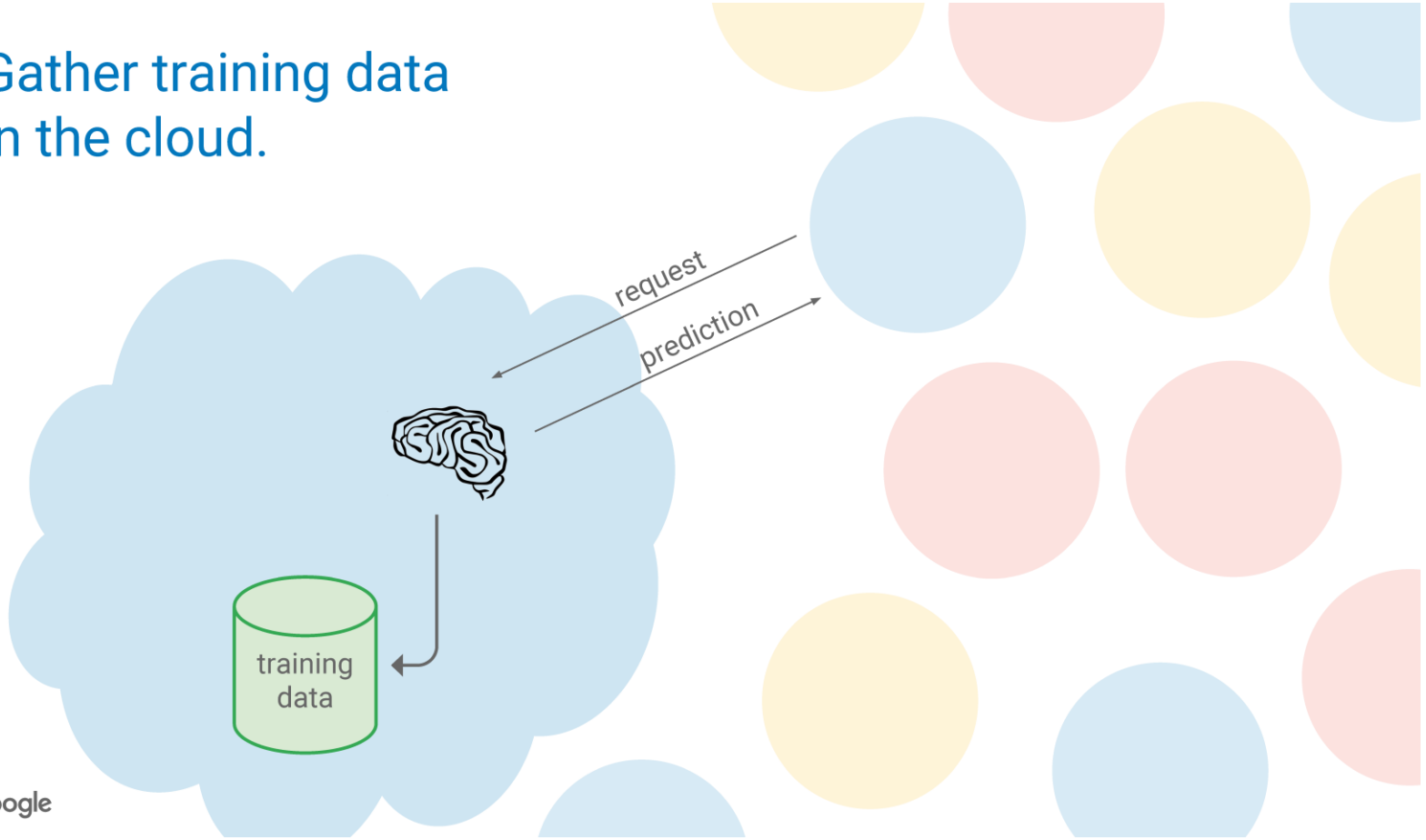
Cloud-centric ML for Mobile

Make predictions in the cloud.



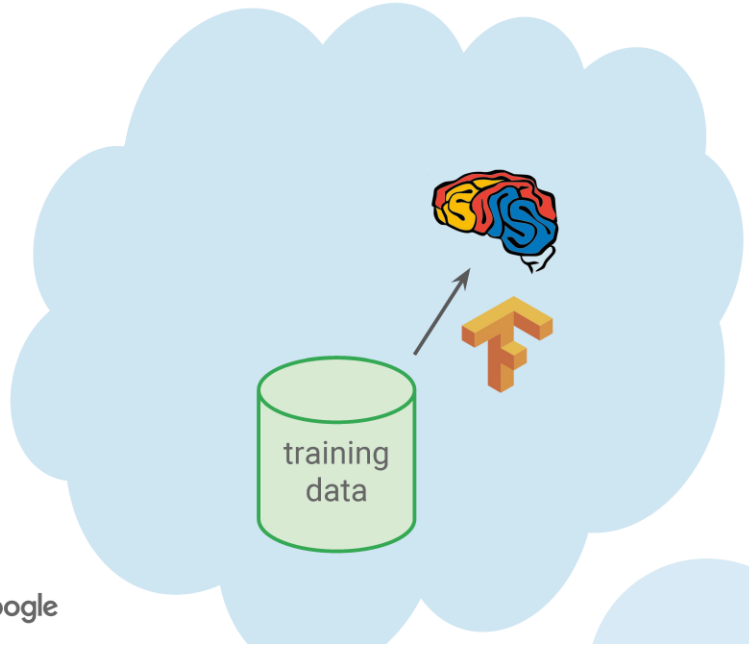
Cloud-centric ML for Mobile

Gather training data
in the cloud.



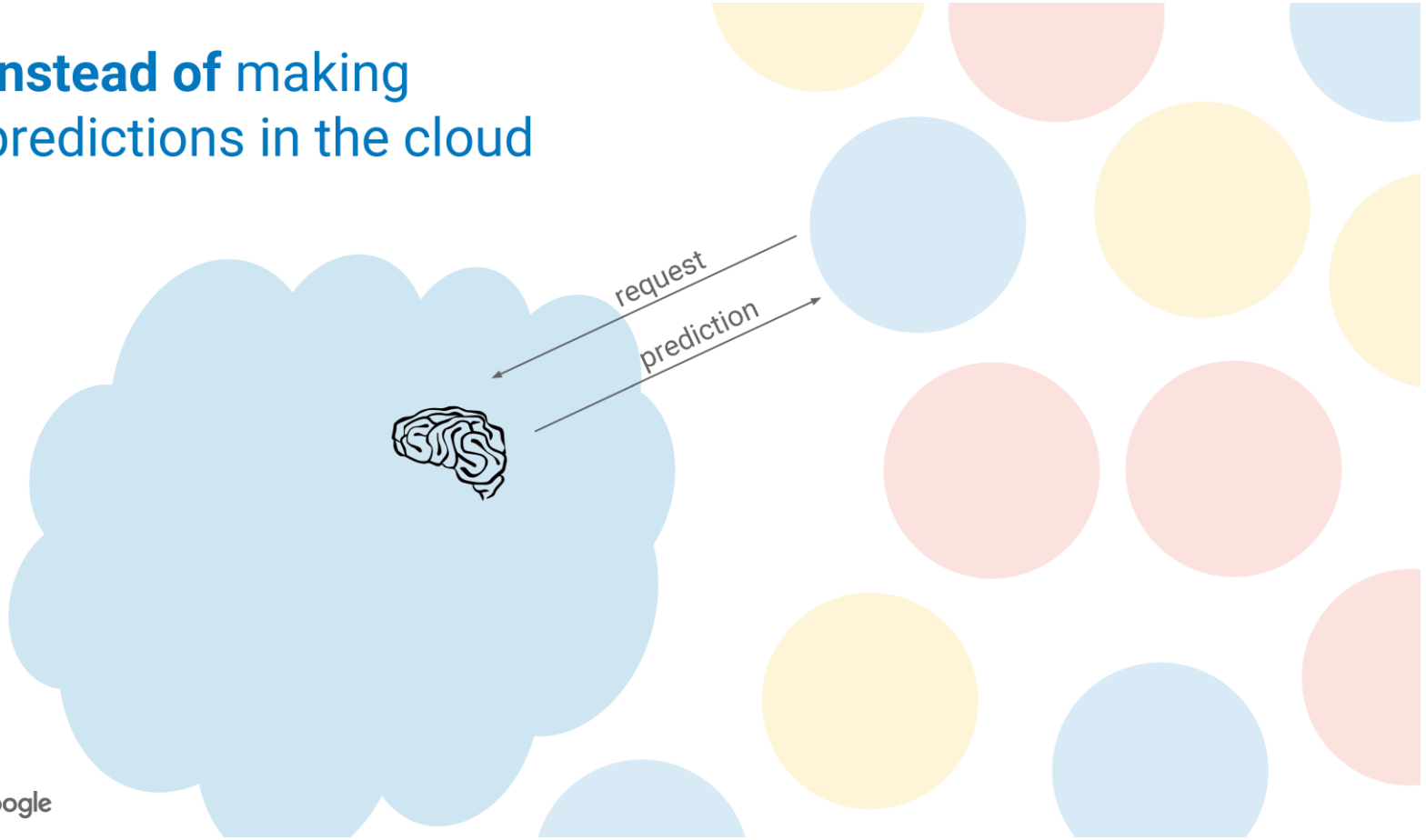
Cloud-centric ML for Mobile

And make the models better.

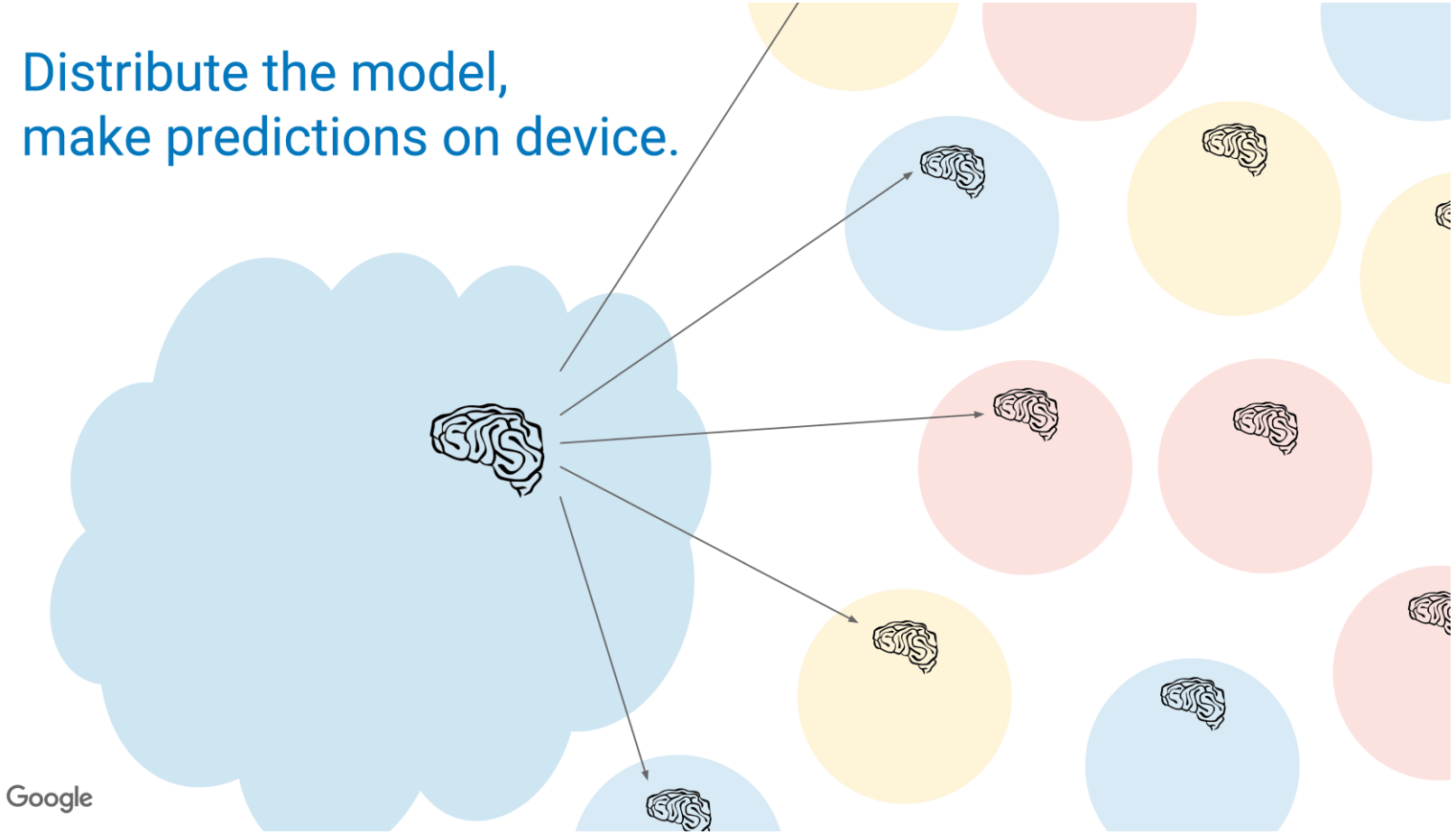


Cloud-centric ML for Mobile

Instead of making predictions in the cloud

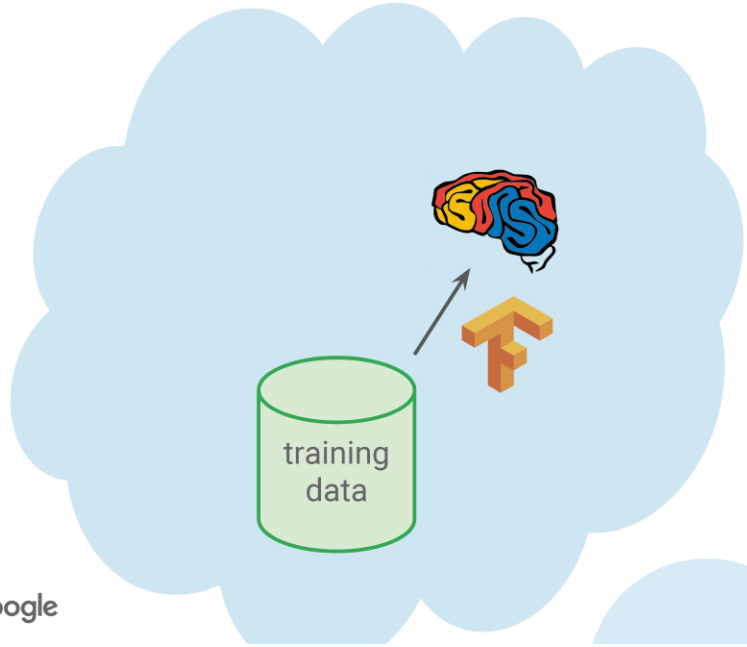


Cloud-centric ML for Mobile



Cloud-centric ML for Mobile

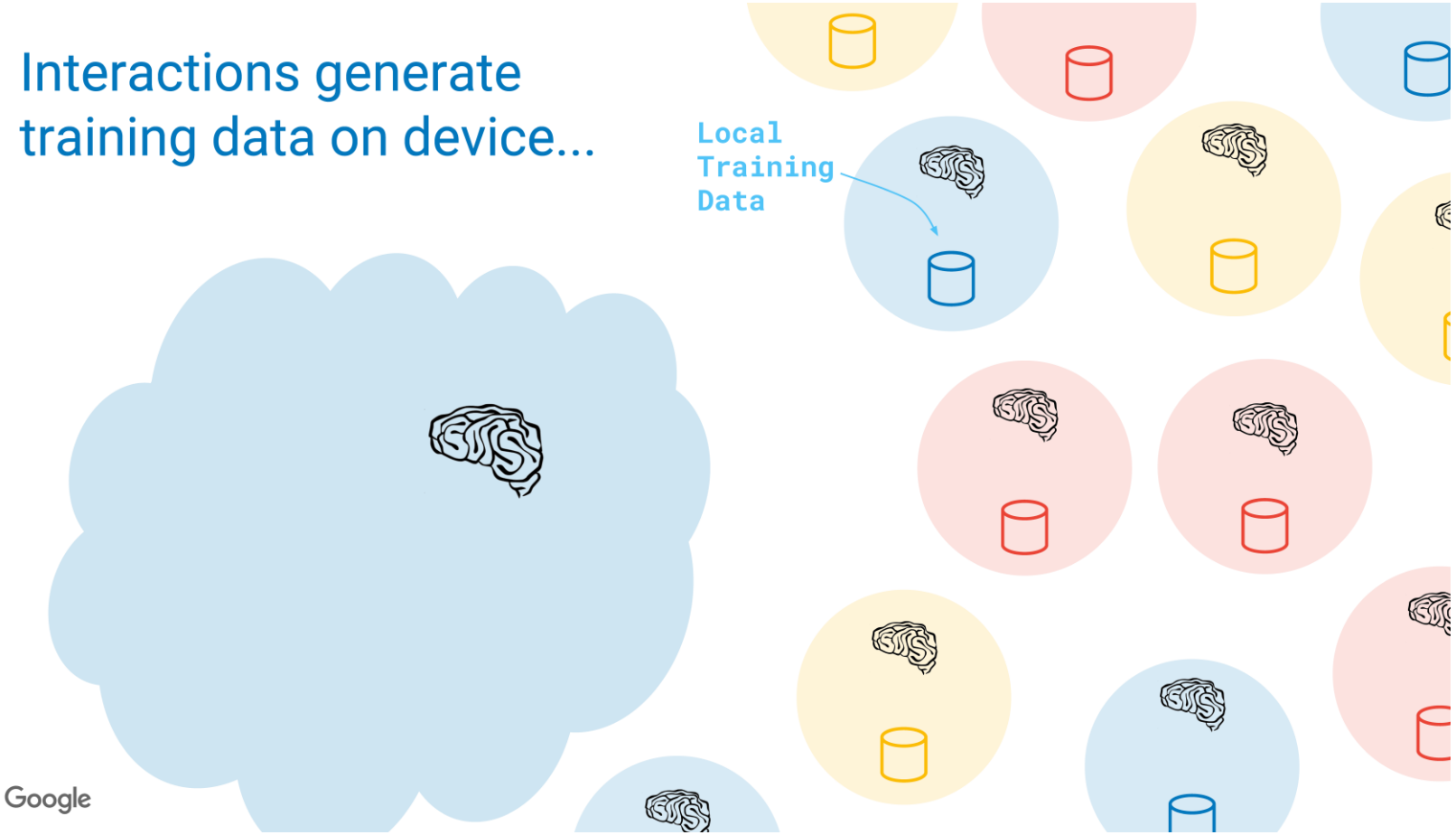
But how do we continue to improve the model?



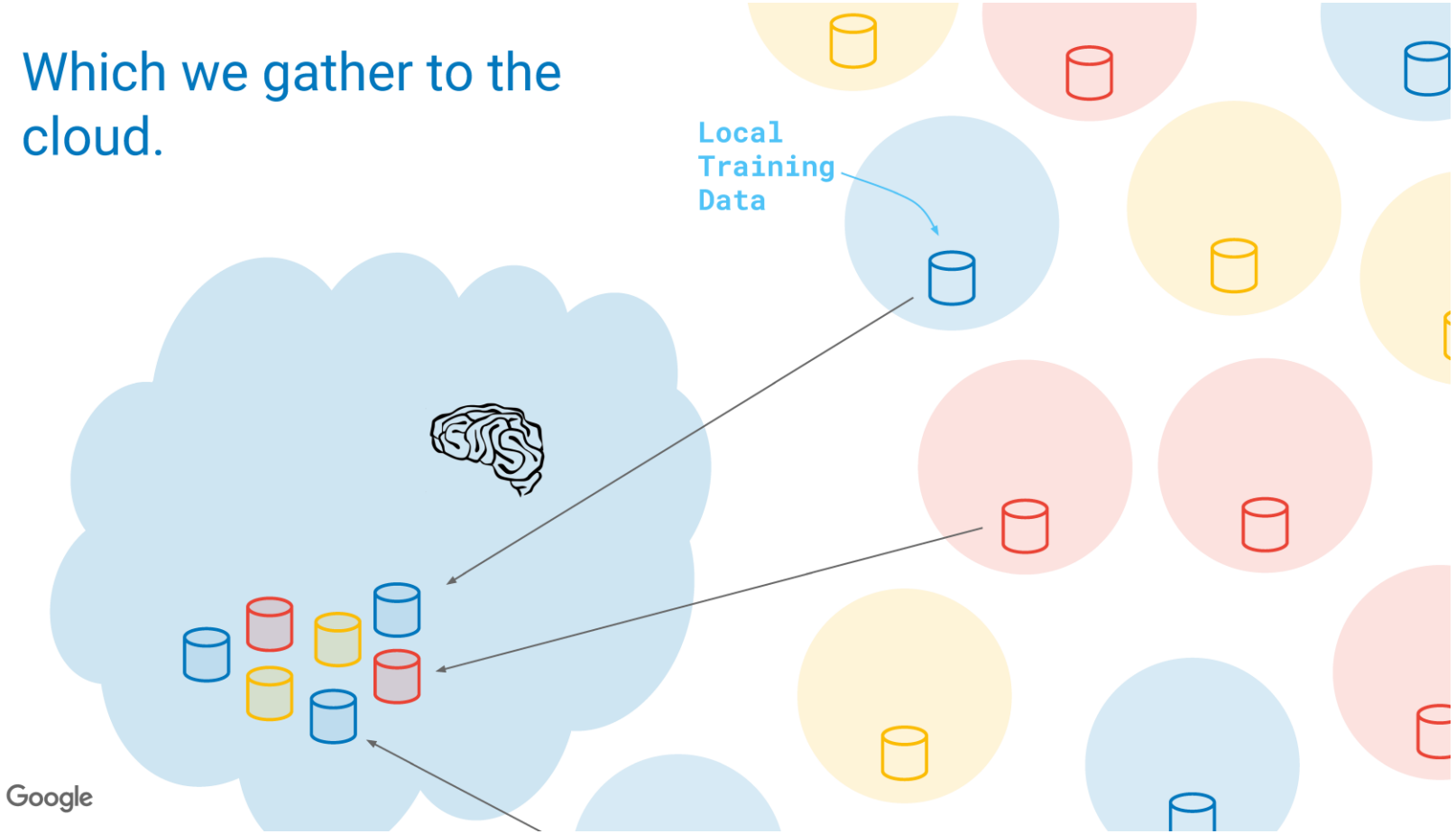
Cloud-centric ML for Mobile

But how do we continue to improve the model?

Cloud-centric ML for Mobile

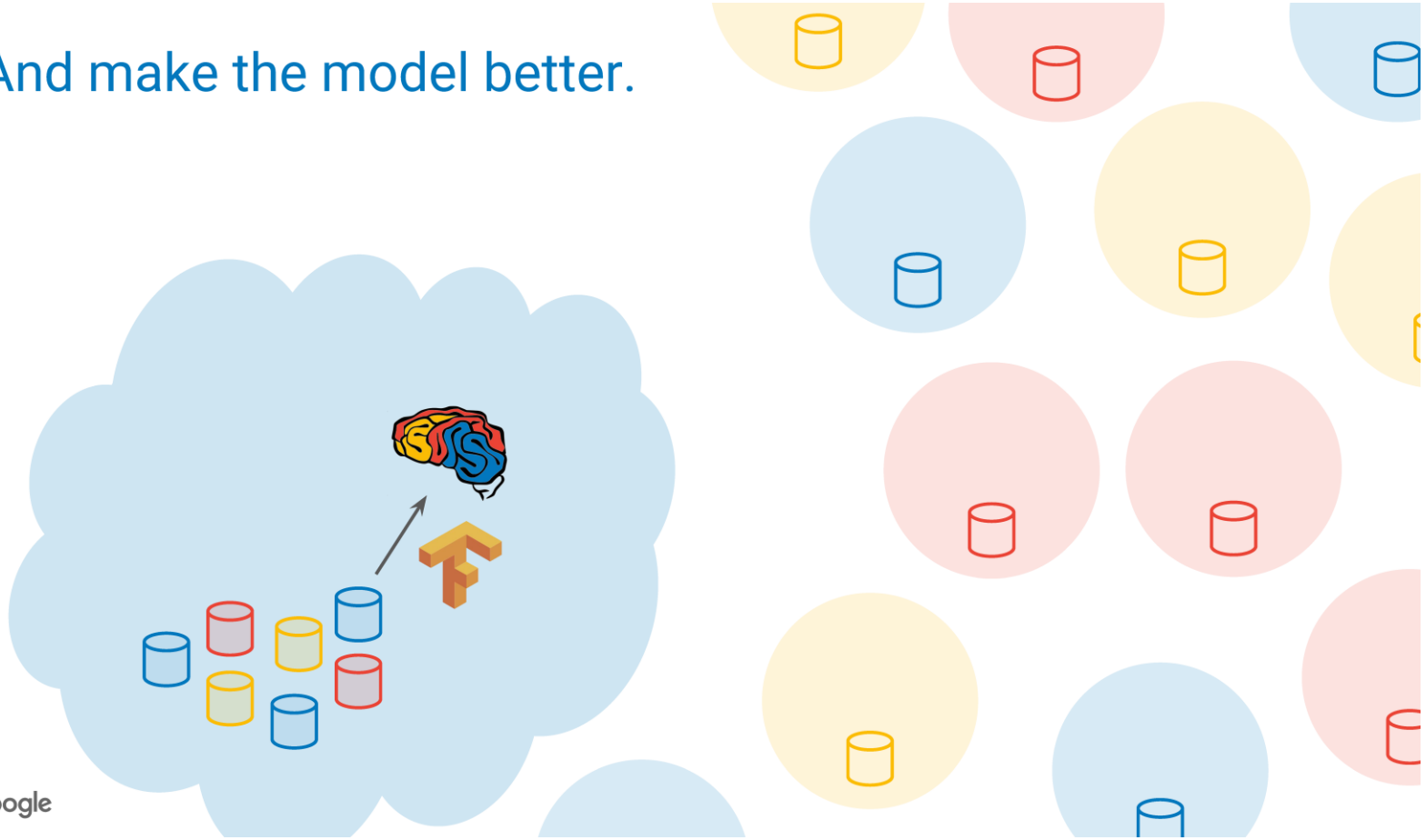


Cloud-centric ML for Mobile

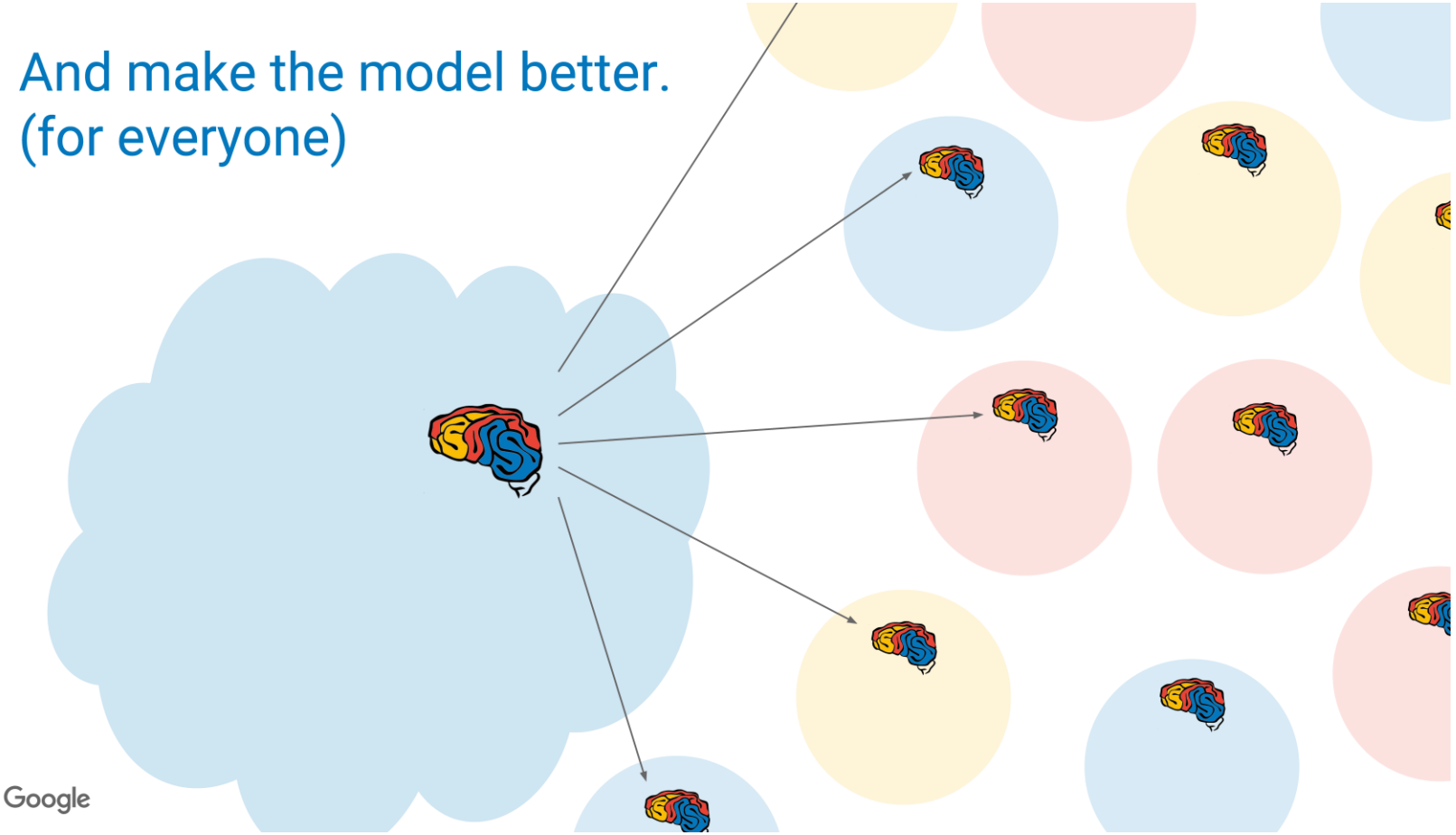


Cloud-centric ML for Mobile

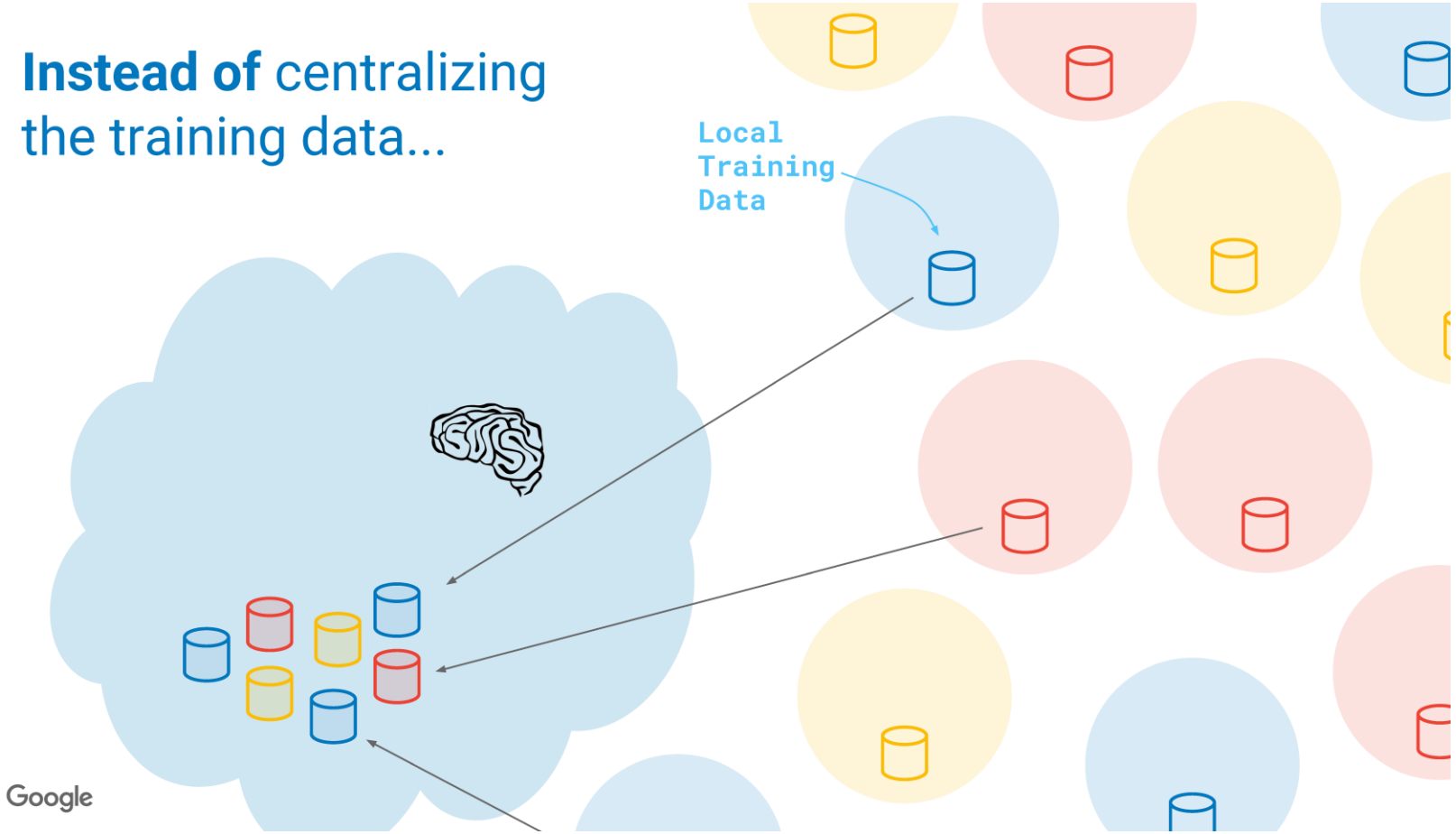
And make the model better.



Cloud-centric ML for Mobile



Cloud-centric ML for Mobile

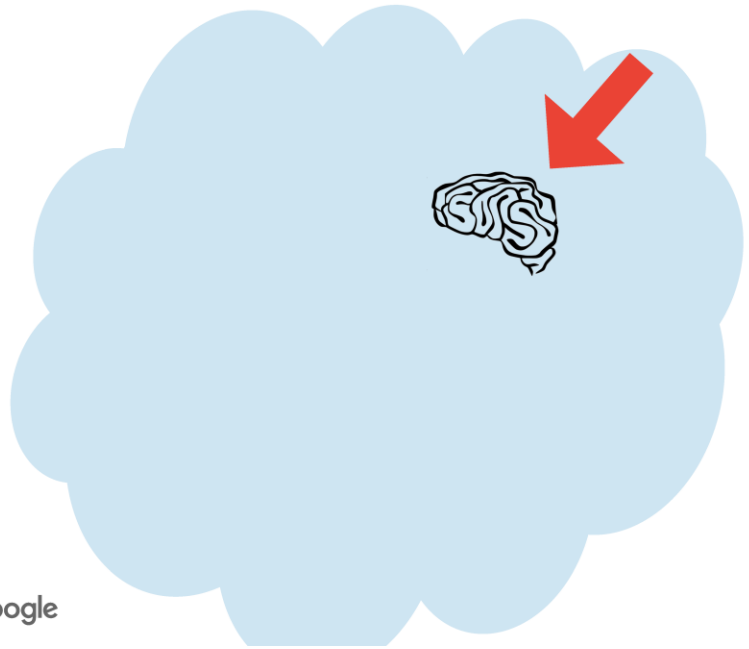


Cloud-centric ML for Mobile

Train models right on the device.
Better for everyone (individually.)

Google

Cloud-centric ML for Mobile



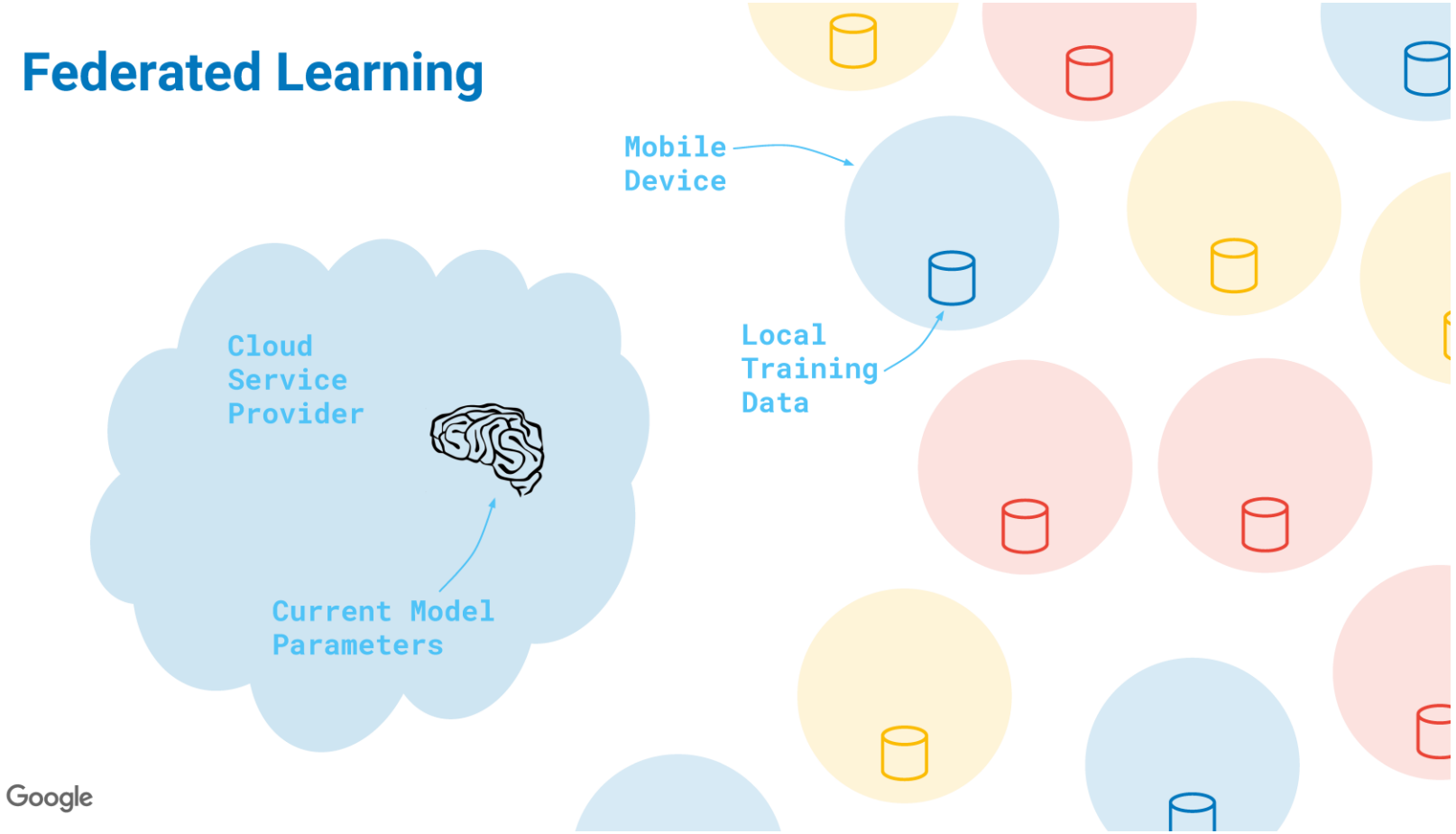
But what about…

1. New User Experience
2. Benefitting from peers' data

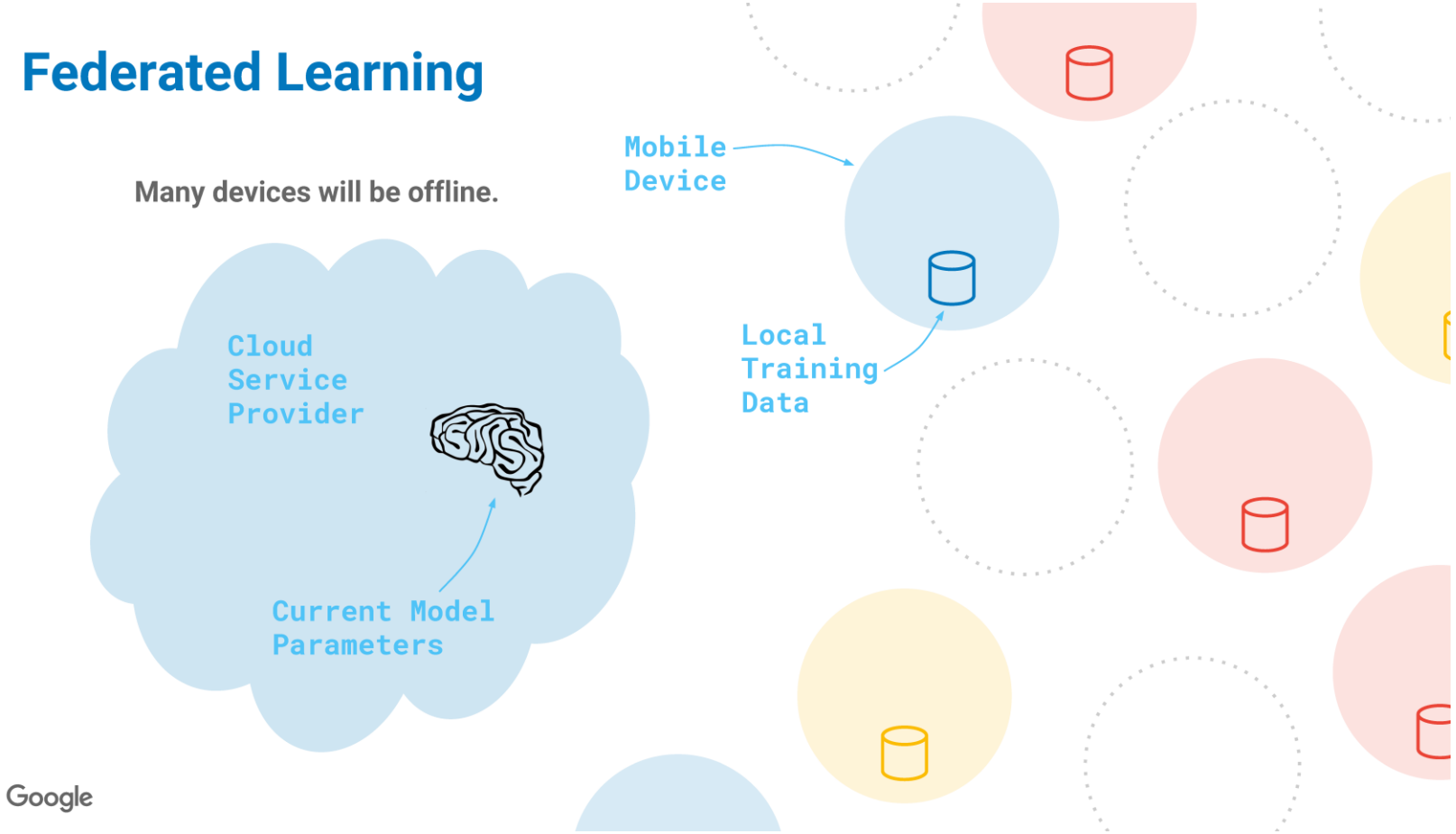
Federated computation and learning

- Federated computation:
  - where a server coordinates a fleet of participating devices to compute aggregations of devices' private data
- Federated learning:
  - where a shared global model is trained via federated computation

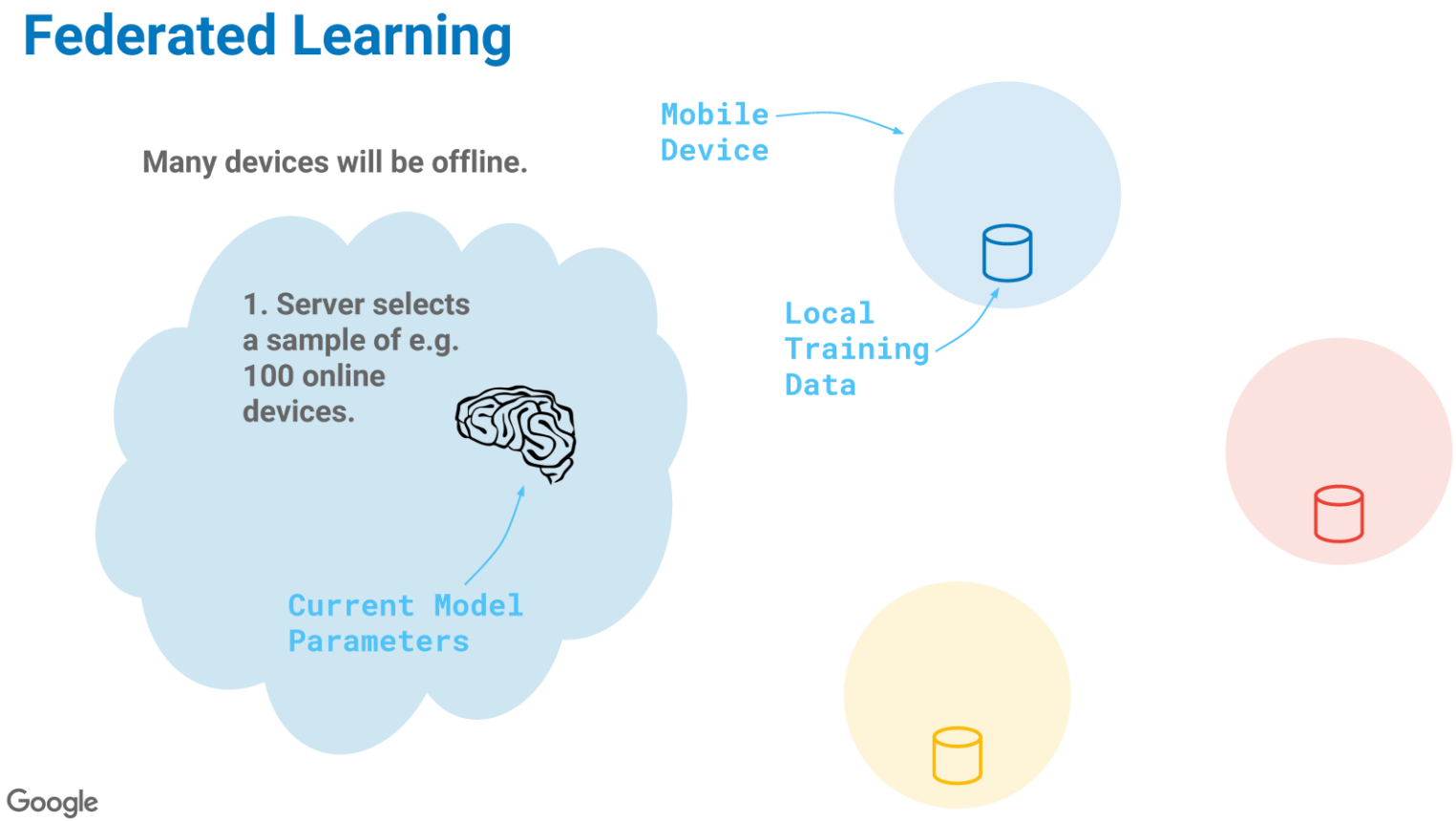
Federated computation and learning



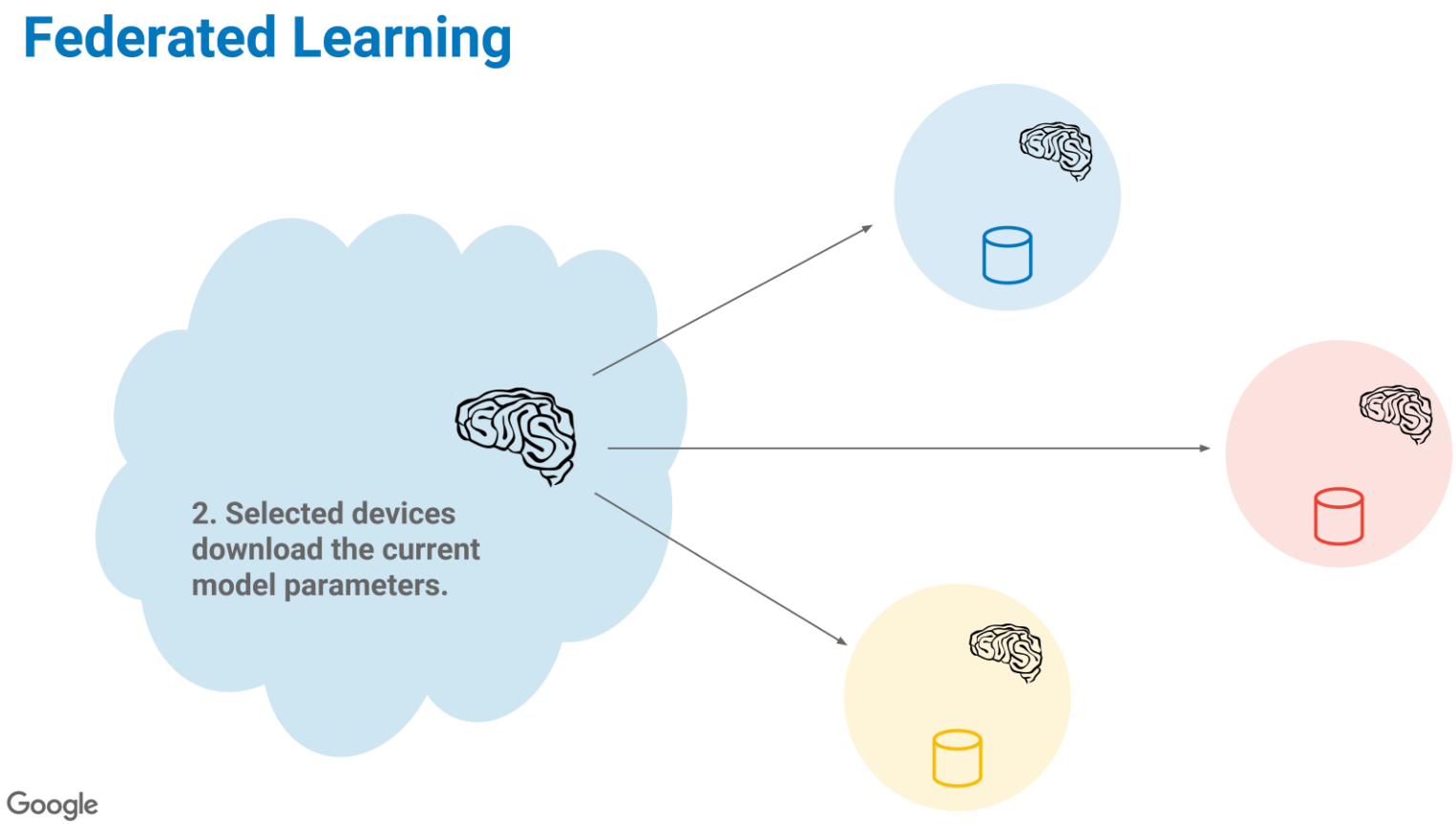
Federated computation and learning



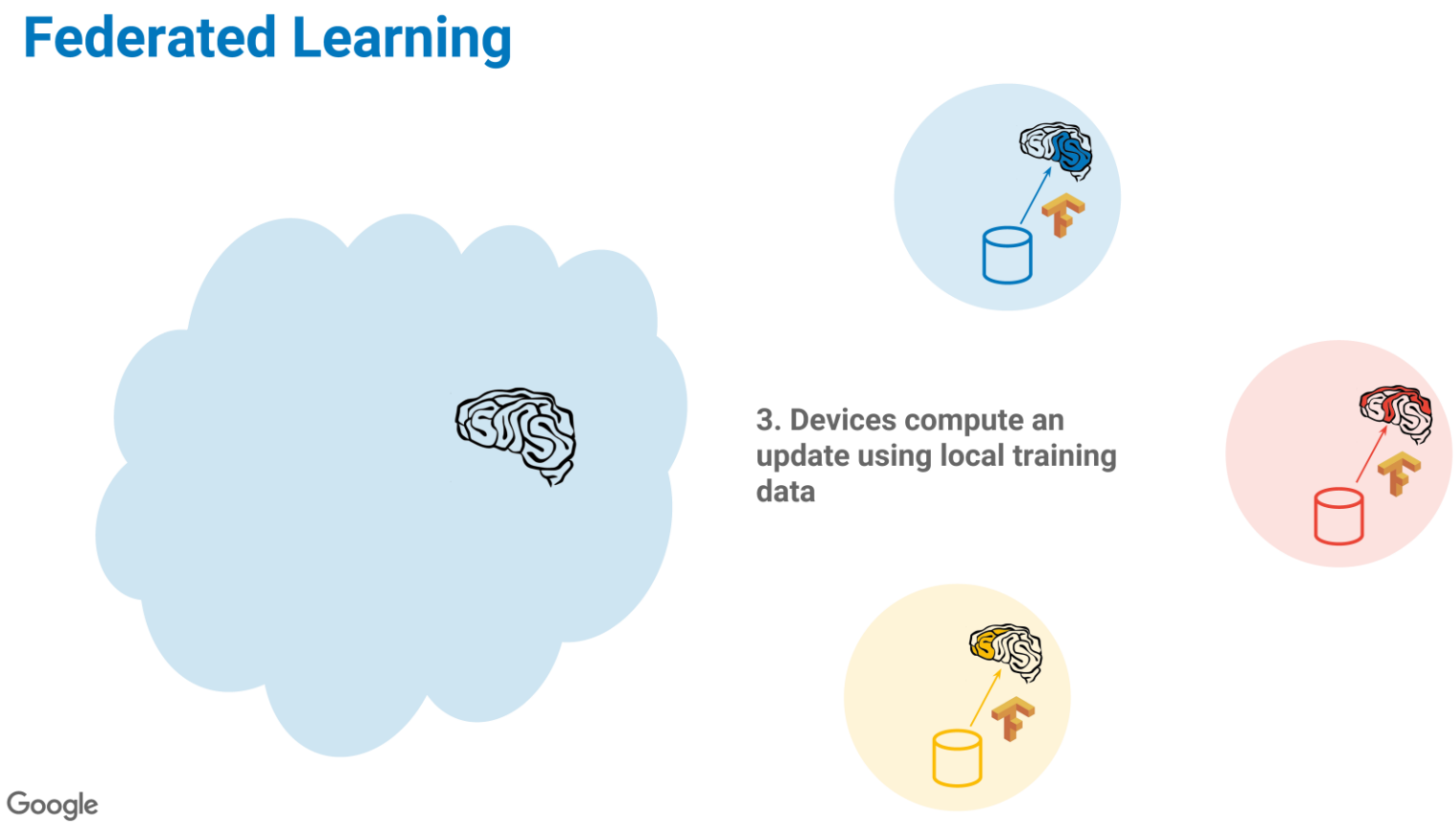
Federated computation and learning



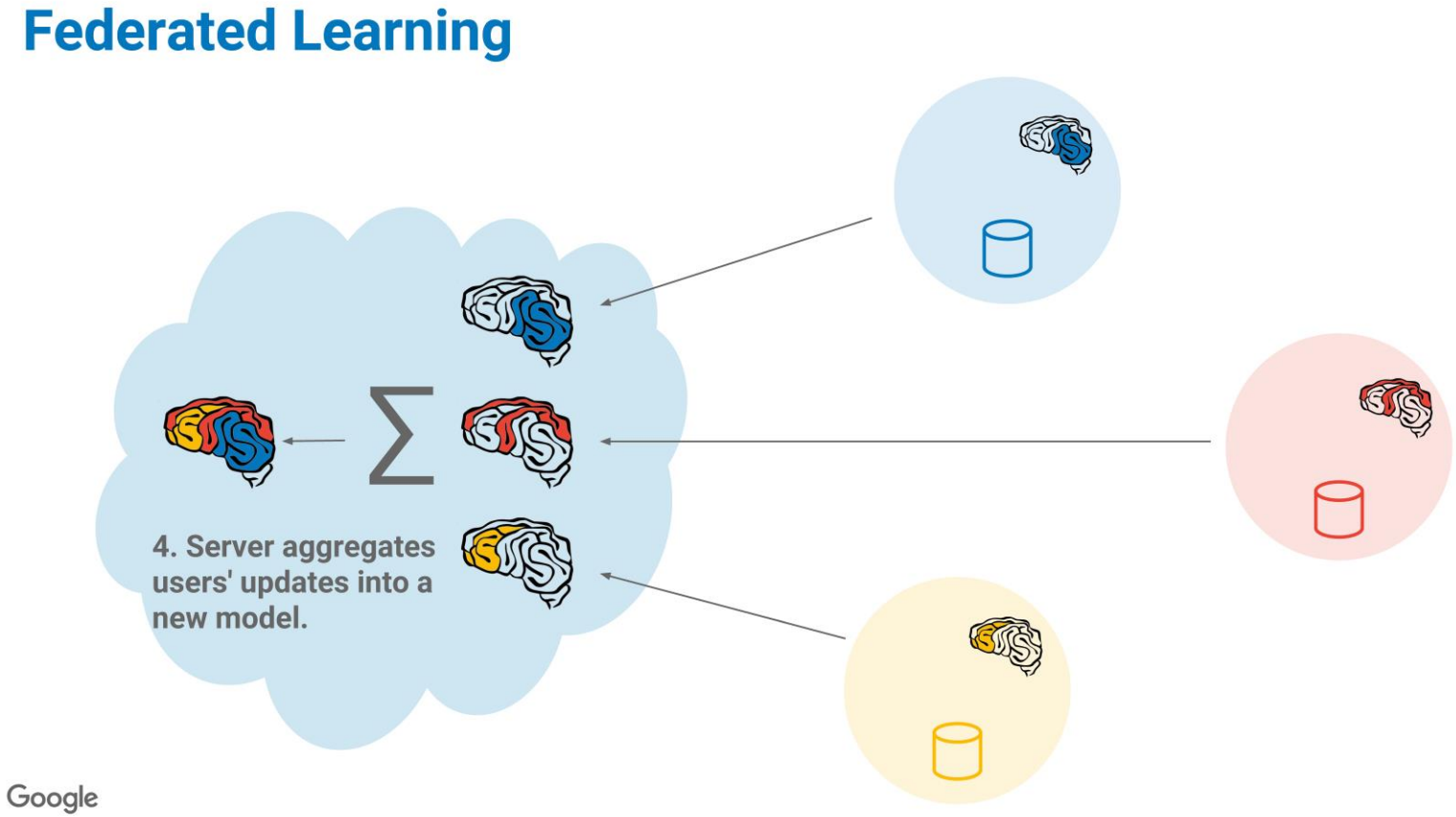
Federated computation and learning



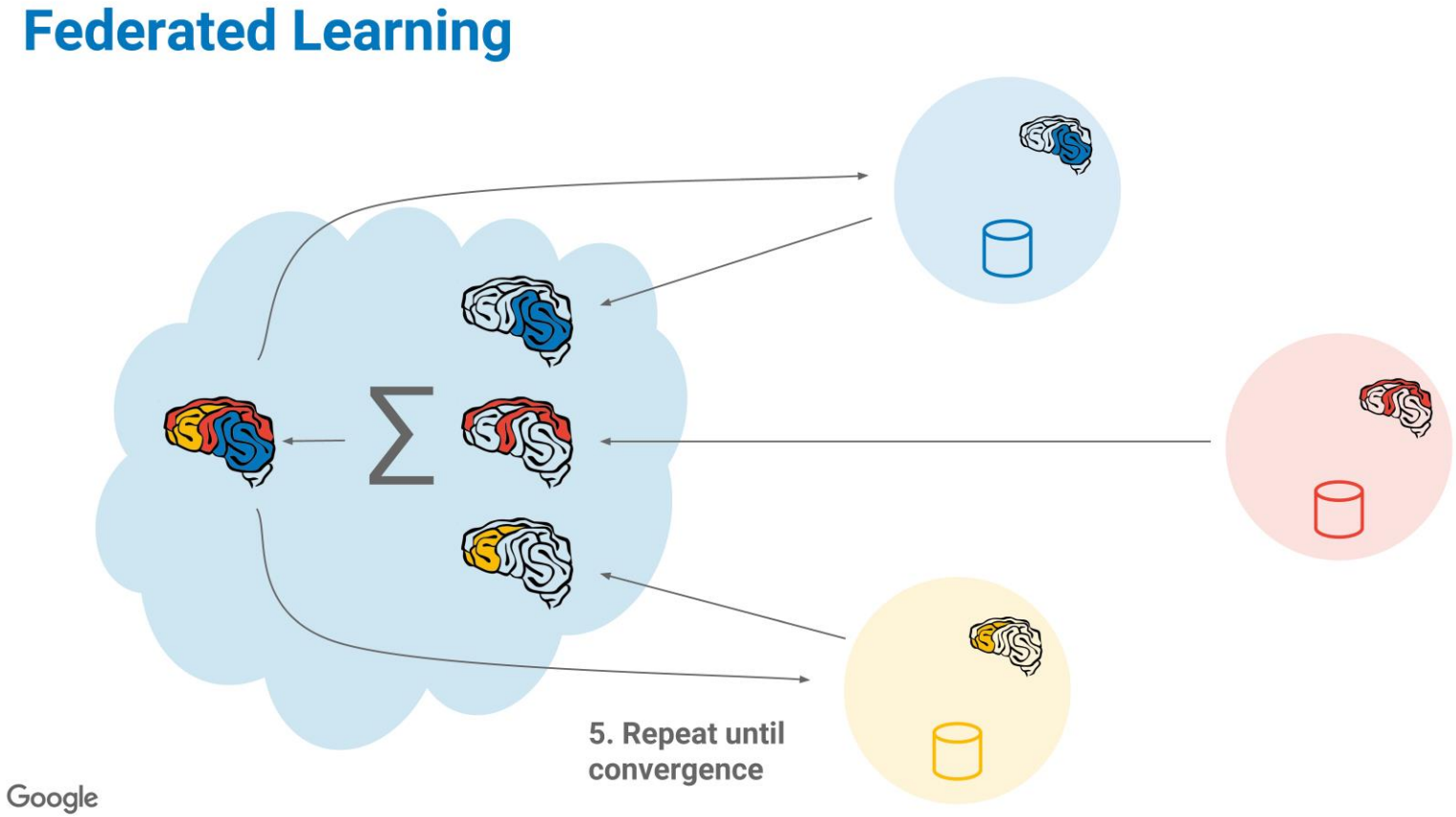
Federated computation and learning



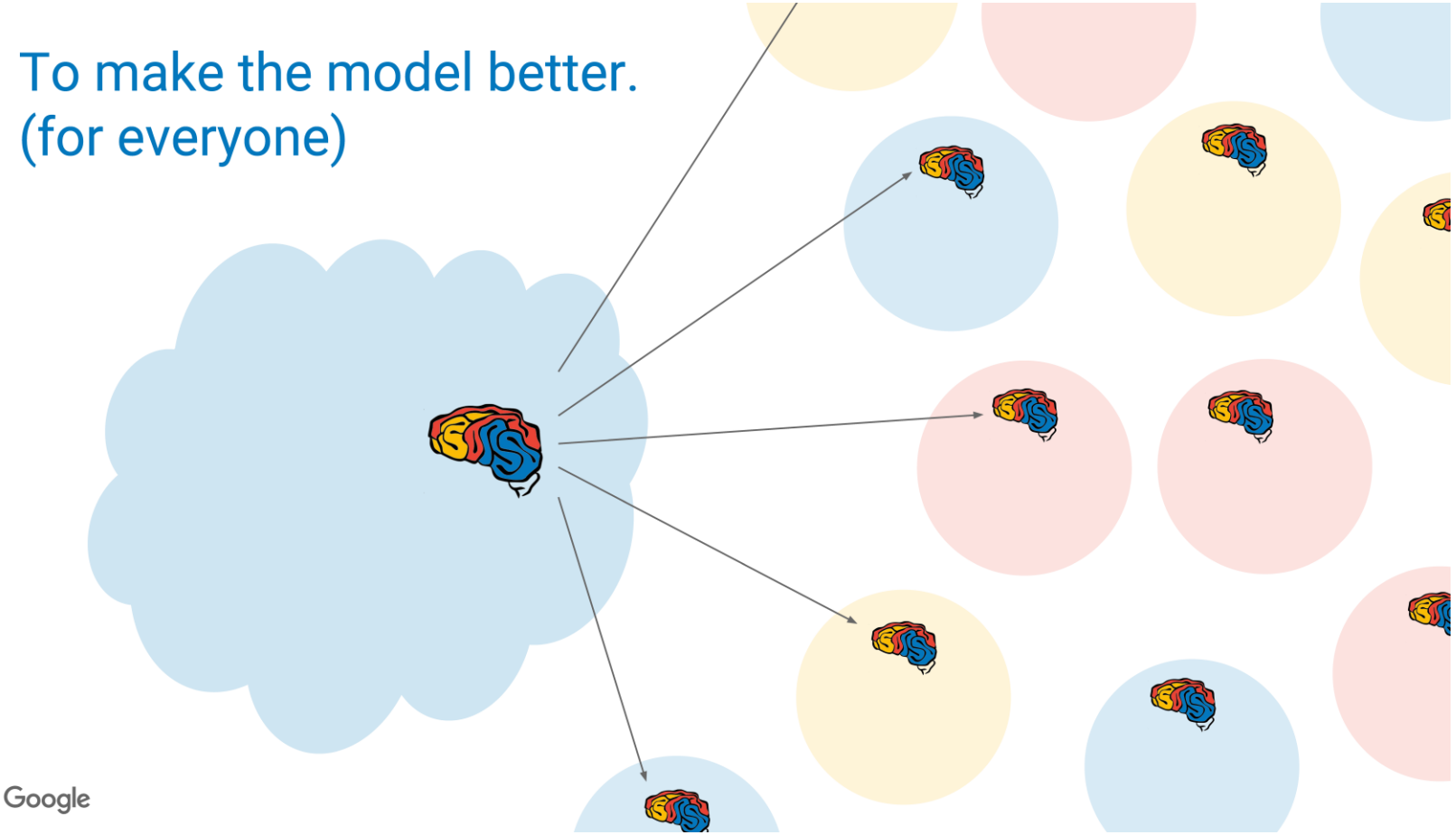
Federated computation and learning



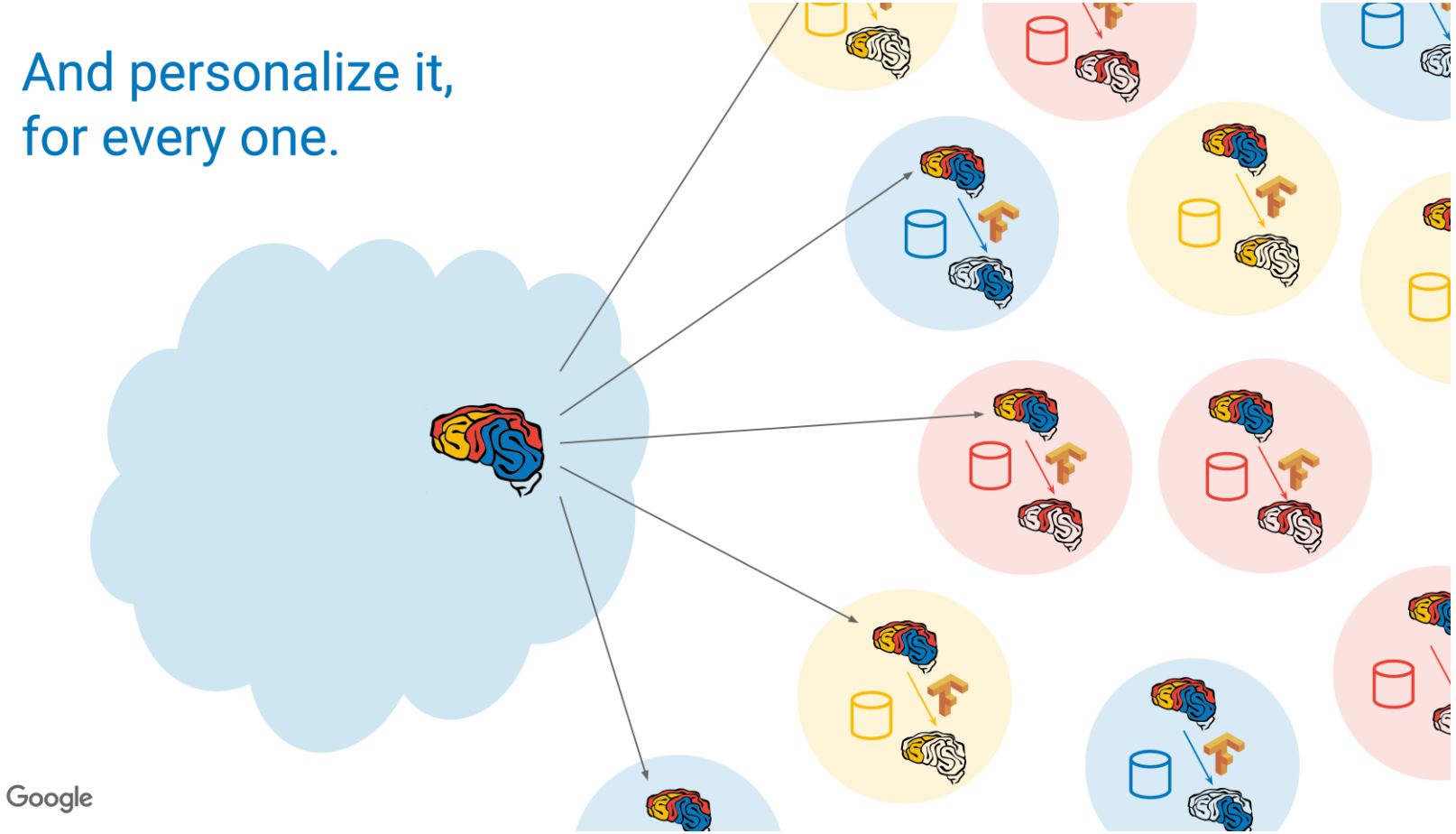
Federated computation and learning



Federated computation and learning



Federated computation and learning



Federated computation and learning

Applications of Federating Learning

What makes a good application?

- On-device data is more relevant than server-side proxy data
- On-device data is privacy sensitive or large
- Labels can be inferred naturally from user interaction

Example applications

- Language modeling for mobile keyboards and voice recognition
- Image classification for predicting which photos people will share
- …

Federated computation and learning

The Federated Averaging algorithm

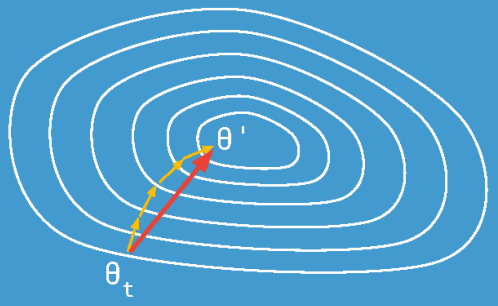
Server

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. In parallel, send current parameters θ_t to those clients

Selected Client k

1. Receive θ_t from server.
2. Run some number of minibatch SGD steps, producing θ'
3. Return $\theta' - \theta_t$ to server.



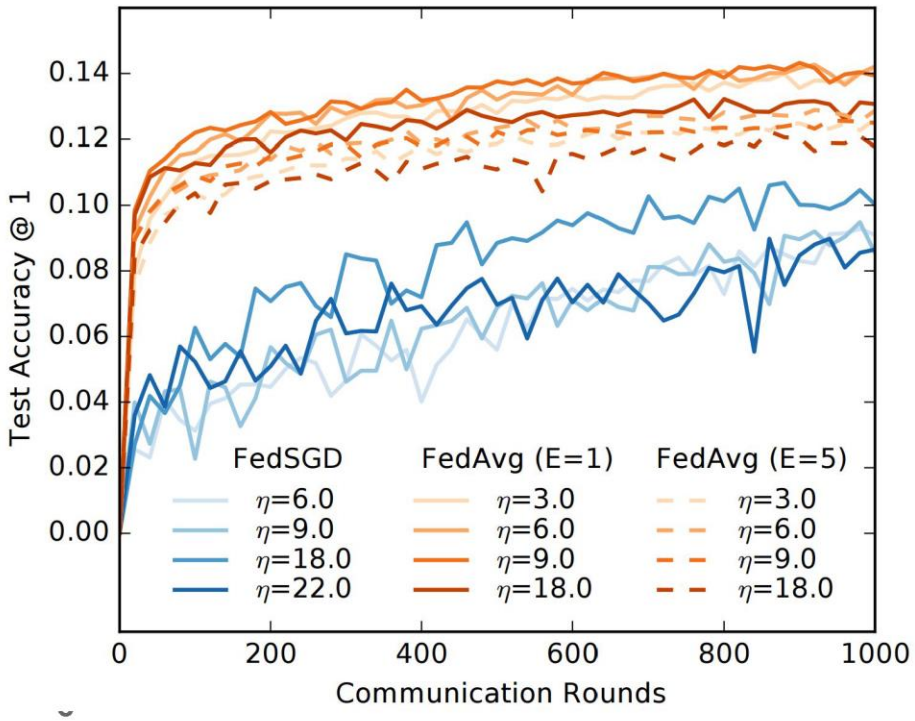
3. $\theta_{t+1} = \theta_t$ + data-weighted average of client updates

H. B. McMahan, *et al.* **Communication-Efficient Learning of Deep Networks from Decentralized Data.** AISTATS 2017

Federated computation and learning

Large-scale LSTM for next-word prediction

Dataset: Large Social Network, 10m public posts, grouped by author.



Rounds to reach 10.5% Accuracy

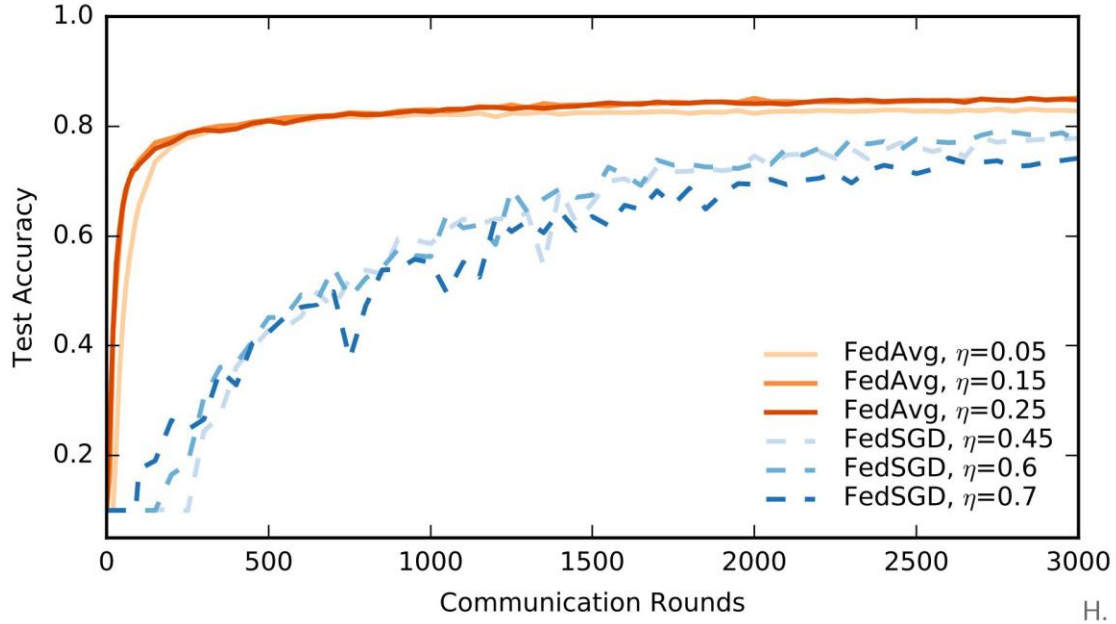
FedSGD	820
FedAvg	35

23x **decrease in communication rounds**

H. B. McMahan, *et al.* **Communication-Efficient Learning of Deep Networks from Decentralized Data.** AISTATS 2017

Federated computation and learning

CIFAR-10 convolutional model



Updates to reach 82%

SGD	31,000
FedSGD	6,600
FedAvg	630

49x decrease in communication (updates) vs SGD

(IID and balanced data)

H. B. McMahan, *et al.* **Communication-Efficient Learning of Deep Networks from Decentralized Data.** AISTATS 2017

Google

Data Lake / Warehouse

ETL

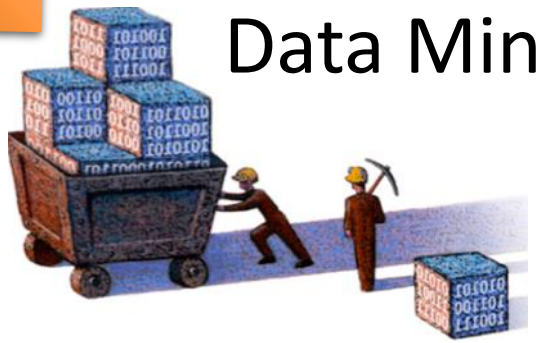
OLAP Analysis & Reporting

Save

Text Data

Photos & Videos

Data Mining

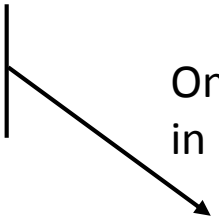


Machine Learning

Data Mining

- Exploratory analysis to find ***interesting*** patterns and extract knowledge from data
- Developed out of the database community
- Emphasis on **scalability**:
  - Computationally efficient & minimize IOs
- Operate with minimal human intervention

- Classic Problem: **Market Basket Analysis**
  - Frequent Itemset Mining
  - Association Rule Mining



One of the most widely cited papers in data mining.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. *SIGMOD Rec.* 22, 2 (June 1993), 207-216.

Market Basket Analysis

- **Market Basket:** collection of items purchased together in a single transaction.

Market Basket Analysis

- **Market Basket:** collection of items purchased together in a single transaction.
- **Goal**: identify co-purchased & linked items
  - Why?
- **Itemset:** *set of items*
  - Example: *{pasta, sauce}*
- **Support** of an *itemset* is the fraction of transactions that contain that itemset.
  - Example: *{pasta, sauce}* ➔ 0.75

TransId	CustId	Item	Qty
111	201	**pasta**	2
111	201	**sauce**	1
111	201	milk	3
111	201	juice	6
112	105	**pasta**	1
112	105	**sauce**	1
112	105	milk	1
113	106	**pasta**	1
113	106	milk	1
114	201	**pasta**	2
114	201	**sauce**	2
114	201	juice	4

Frequent Item Sets

- **Goal:** Identify all **itemsets** with **support** greater than ***minsup*** threshold
- What are the **frequent item sets** for **minsup** > 0.7
  - {pasta}: 1, {sauce}: .75, {milk}: 0.75
  - {pasta, sauce}: .75, {pasta, milk}: 0.75
- **Observation:** (Downward Closure)
  - If A ⊆ B ➔ Sup (A) ≥ Sup(B)
  - All subsets of a frequent item set are frequent item sets.

TransId	CustId	Item	Qty
111	201	**pasta**	2
111	201	**sauce**	1
111	201	milk	3
111	201	juice	6
112	105	**pasta**	1
112	105	**sauce**	1
112	105	milk	1
113	106	**pasta**	1
113	106	milk	1
114	201	**pasta**	2
114	201	**sauce**	2
114	201	juice	4

Itemsets: Computation model

- Typically, data is kept in flat files rather than in a database system:
  - Stored on disk
  - Stored basket-by-basket
  - Baskets are small but we have many baskets and many items
    - Expand baskets into pairs, triples, etc. as you read baskets
    - Use k nested loops to generate all sets of size k
- The true cost of mining disk-resident data is usually the number of disk I/Os
  - We measure the cost by the number of passes an algorithm makes over the data

Main-memory Bottleneck

- For many frequent-itemset algorithms, main-memory is the critical resource
  - As we read baskets, we need to count something, e.g., occurrences of pairs of items
  - The number of different things we can count is limited by main memory
  - Swapping counts in/out is a disaster

Finding frequent pairs

- The hardest problem often turns out to be finding the frequent pairs of items {i, j}
  - Why? Freq. pairs are common, freq. triples are rare
  - Why? Probability of being frequent drops exponentially with size; number of sets grows more slowly with size
- Let's first concentrate on pairs, then extend to larger sets
- The approach:
  - We always need to generate all the itemsets
  - But we would only like to count (keep track) of those itemsets that in the end turn out to be frequent

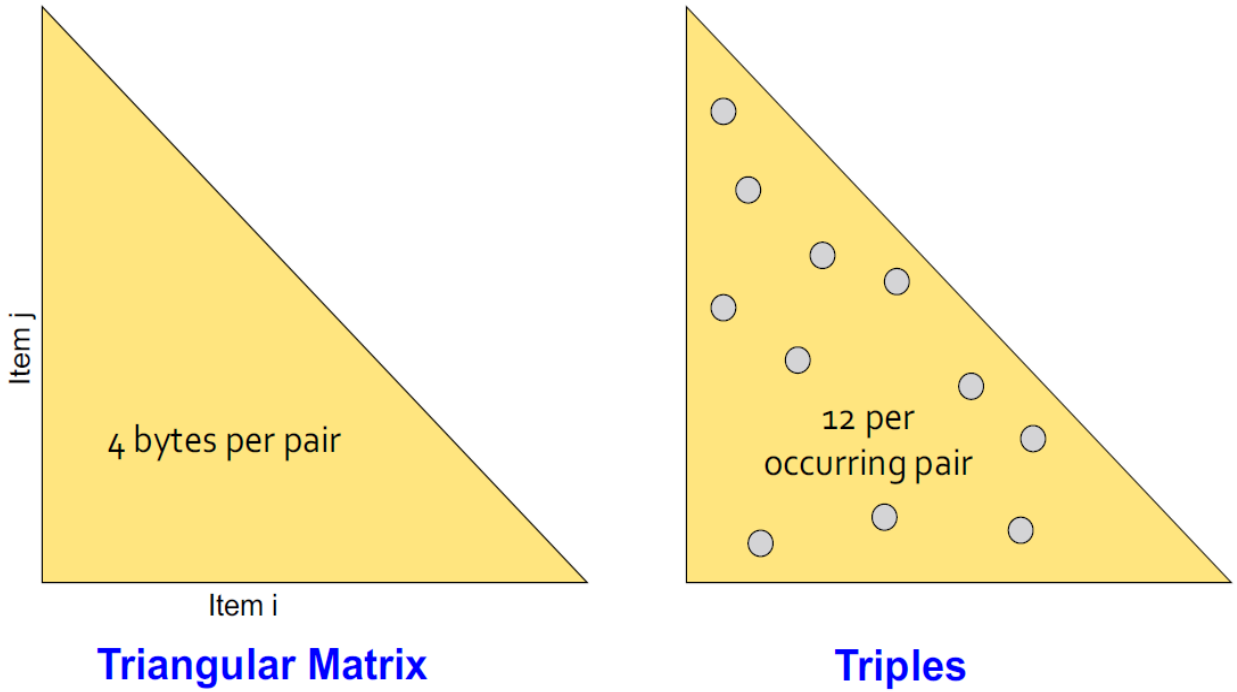
Naïve algorithm

- Naïve approach to finding frequent pairs
- Read file once, counting in main memory the occurrences of each pair:
  - From each basket of n items, generate its n(n-1)/2 pairs by two nested loops
- Fails if $(\#items)^2$ exceeds main memory
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages)
    - Suppose 10^5 items, counts are 4-byte integers
    - Number of pairs of items: $10^5(10^5-1)/2 \approx 5*10^9$
    - Therefore, $2*10^{10}$ (20 gigabytes) of memory is needed

Counting pairs in memory

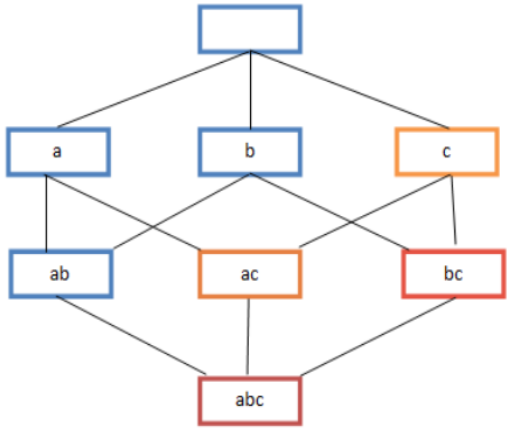
- Goal: Count the number of occurrences of each pair of items (i,j):
- Approach 1: Count all pairs using a matrix
- Approach 2: Keep a table of triples [i, j, c] = "the count of the pair of items {i, j} is c."
  - If integers and item ids are 4 bytes, we need approximately 12 bytes for pairs with count > 0
  - Plus some additional overhead for the hashtable

Counting pairs in memory



- Approach 2 beats Approach 1 if less than 1/3 of possible pairs actually occur
- Problem is if we have too many items so the pairs do not fit into memory.
  - Can we do better?

A-Priori Algorithm

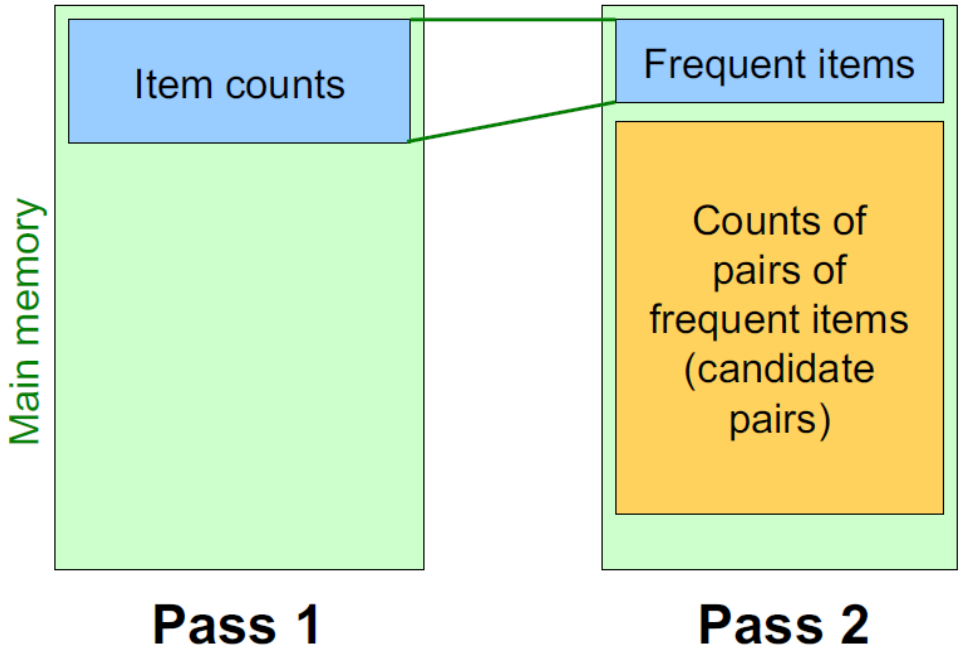


- A two-pass approach called A-Priori limits the need for main memory
- Key idea: monotonicity
  - If a set of items *I* appears at least *s* times, so does every subset *J* of *I*
- Contrapositive for pairs:
  - If item *i* does not appear in *s* baskets, then no pair including *i* can appear in *s* baskets
- So, how does A-Priori find freq. pairs?

A-Priori Algorithm

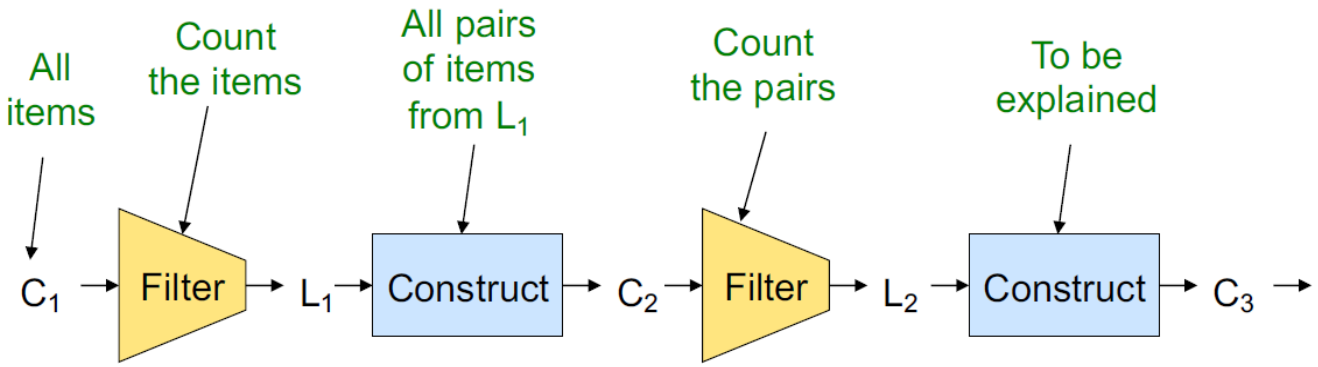
- **Pass 1:** Read baskets and count in main memory the # of occurrences of each **individual item**
  - Requires only memory proportional to #items
- **Items that appear $\geq s$ times are the frequent items**
- **Pass 2:** Read baskets again and keep track of the count of only those pairs where both elements are frequent (from Pass 1)
  - Requires memory proportional to square of **frequent** items only (for counts)
  - Plus a list of the frequent items (so you know what must be counted)

Main-memory map



Frequent triples, etc.

- **For each *k*, we construct two sets of *k-tuples*** (sets of size *k*):
  - C_k = ***candidate k-tuples*** = those that might be frequent sets (support $\geq$ **s**) based on information from the pass for ***k*–1**
  - L_k = the set of truly frequent ***k***-tuples



Frequent Item Sets Algorithm (FIS)

```
Items ← Set of Unique Items
FISs ← {∅} # Frequent item sets
for k in 1, 2, …
    # Compute candidates by extend biggest FIS
    candFISs ← {∅}
    for fis in FISs if |fis| == k – 1:
        for item in Items:
            candFISs ← candFISs ∪ {fis ∪ {item}}
    # Check all candidates (in one pass of DB)
    newFISs ← [fis for fis in candFISs
                    if Sup(fis) > minSup]
    # If no new FISs found return otherwise continue
    if newFIS.isEmpty return FISs else:
        FISs ← FISs ∪ newFISs
```

Association Rules

- **<u>Goal:</u>** Identify **predictive rules** that relate a subset of items to the *likely* presence of other items in the basket
- **Association Rule:** *{LHS}* ➔ *{RHS}*
  - Example: {pasta} ➔ {sauce}
- **Support of an Association Rule**
  - Support of the set LHS ∪ RHS
- **Confidence of an Association Rule**
  - Support (LHS ∪ RHS) / Support (LHS)

- **Probabilistic Interpretation:** *Conditional Probability*

$$\frac{\mathbf{P}\left(\mathrm{LHS} \in \mathrm{Basket}, \mathrm{RHS} \in \mathrm{Basket}\right)}{\mathbf{P}\left(\mathrm{LHS} \in \mathrm{Basket}\right)} = \mathbf{P}\left(\mathrm{RHS} \in \mathrm{Basket} \,|\, \mathrm{LHS} \in \mathrm{Basket}\right)$$

More generally

- A general many-to-many mapping (association) between two kinds of things
  - But we ask about connections among “items”, not “baskets”
- Items and baskets are abstract:
  - For example:
    - Items/baskets can be products/shopping basket
    - Items/baskets can be words/documents
    - Items/baskets can be base-pairs/genes
    - Items/baskets can be drugs/patients

Interesting association rules

- **Not all high-confidence rules are interesting**
  - The rule $X \rightarrow milk$ may have high confidence for many itemsets X, because milk is just purchased very often (independent of X) and the confidence will be high
- **Interest** of an association rule $I \rightarrow j$:
  abs. difference between its confidence and the fraction of baskets that contain j

$$\text{Interest}(I \rightarrow j) = |\text{conf}(I \rightarrow j) - \Pr[j]|$$

  - Interesting rules are those with high positive or negative interest values (usually above 0.5)

Example: Confidence and Interest

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, b} B_4 = {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}

- **Association rule: {m, b} → c**
  - **Support** = 2/8 = 1/4
  - **Confidence** = 2/4 = 0.5
  - **Interest** = |0.5 – 5/8| = 1/8
    - Item ***c*** appears in 5/8 of the baskets
    - The rule is not very interesting!

Association Rule Mining Algorithm

- Input Parameters:
  - **minsup** – the minimum support of discovered rules
  - **minconf** – the minimum confidence or discovered rules
- Algorithm:
  - Compute frequent itemsets with **minsup**
  - For each frequent itemset consider all possible subsets as LHS and compute confidence
    - Recall confidence = Sup(LHS∪RHS) / Sup(LHS)
  - Return all rules with confidence > **minconf**
- Resulting association rules can be used for:
  - marketing decisions: beer and diapers near each other
  - Triggers in an online targeting systems:
    - Given items in a basket what are most likely items to be added
- **Important**: *Rule does not imply causality*

Example

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, c, b, n} B_4= {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}

- **Support threshold $s = 3$, confidence $c = 0.75$**
- **Step 1) Find frequent itemsets:**
  - {b,m} {b,c} {c,m} {c,j} {m,c,b}
- **Step 2) Generate rules:**
  - ~~b→m: c=4/6~~ b→c: c=5/6 ~~b,c→m: c=3/5~~
  - m→b: c=4/5 … b,m→c: c=3/4
    ~~b→c,m: c=3/6~~