

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

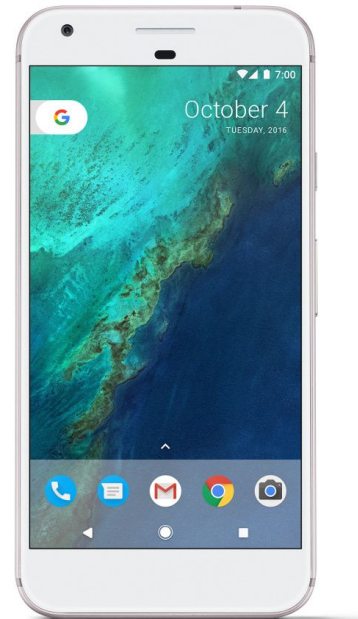
For the rest of the details of the license, see

<https://creativecommons.org/licenses/by-sa/2.0/legalcode>

Data is born at the edge

Billions of phones & IoT devices constantly generate data

Data enables better products and smarter models



Can data live at the edge?

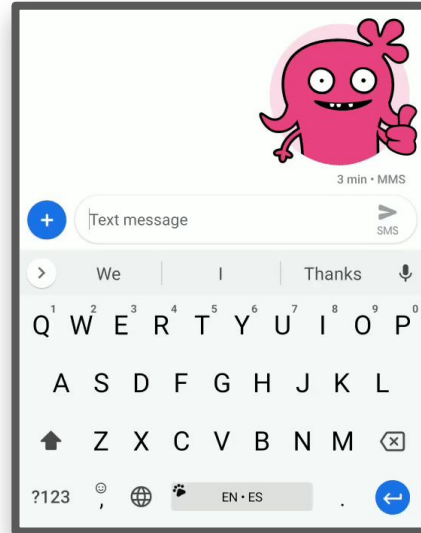
On-device inference offers:

- Improved latency
- Works offline
- Better battery life
- Privacy advantages





Gboard: mobile keyboard

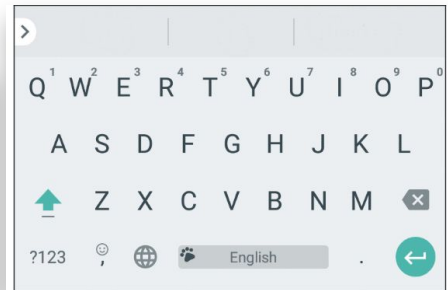
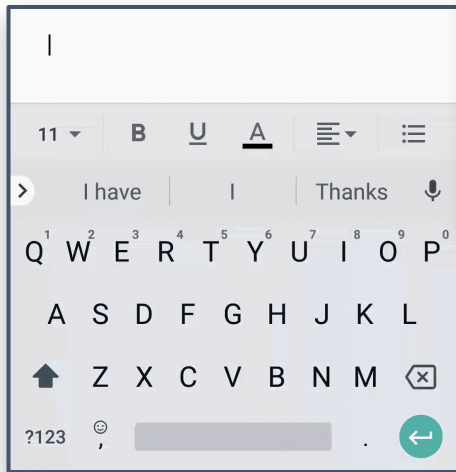




Gboard machine learning

Models are essential for:

- tap typing
- gesture typing
- auto-corrections
- predictions
- voice to text
- and more...





Federated Learning: Collaborative Machine Learning without Centralized Training Data

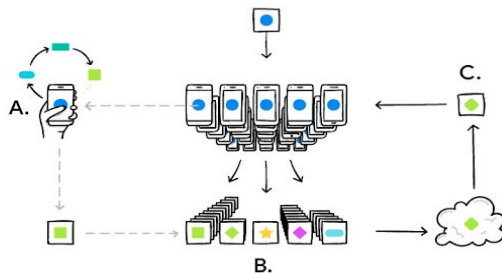
Thursday, April 6, 2017

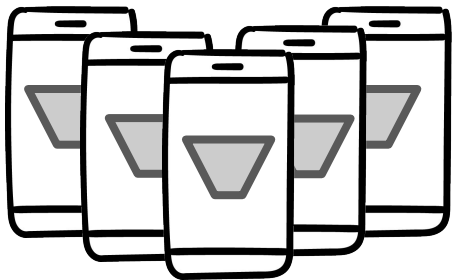
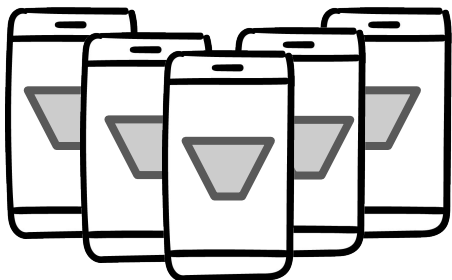
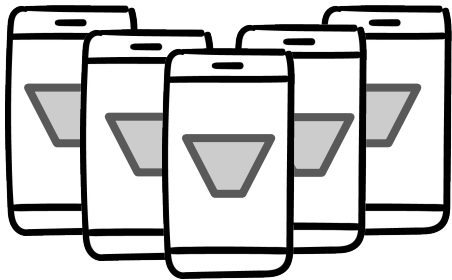
Posted by Brendan McMahan and Daniel Ramage, Research Scientists

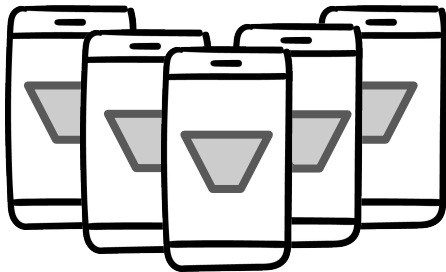
Standard machine learning approaches require centralizing the training data on one machine or in a datacenter. And Google has built one of the most secure and robust cloud infrastructures for processing this data to make our services better. Now for models trained from user interaction with mobile devices, we're introducing an additional approach: *Federated Learning*.

Federated Learning enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on device, decoupling the ability to do machine learning from the need to store the data in the cloud. This goes beyond the use of local models that make predictions on mobile devices (like the [Mobile Vision API](#) and [On-Device Smart Reply](#)) by bringing model *training* to the device as well.

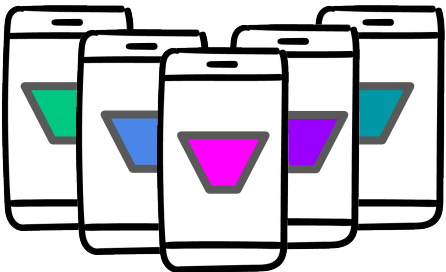
It works like this: your device downloads the current model, improves it by learning from data on your phone, and then summarizes the changes as a small focused update. Only this update to the model is sent to the cloud, using encrypted communication, where it is immediately averaged with other user updates to improve the shared model. All the training data remains on your device, and no individual updates are stored in the cloud.



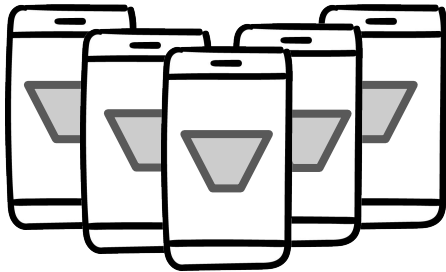




Not Available



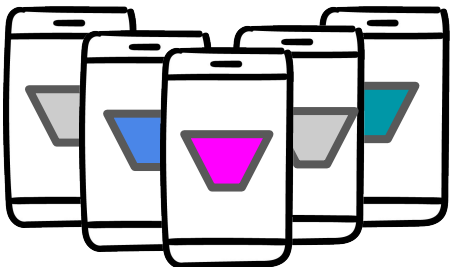
Available



Not Available



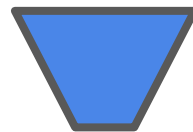
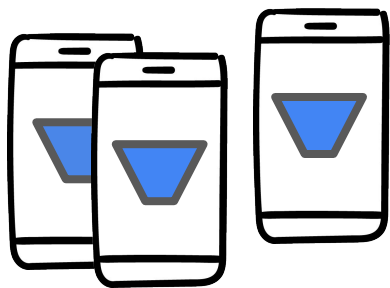
Not Available

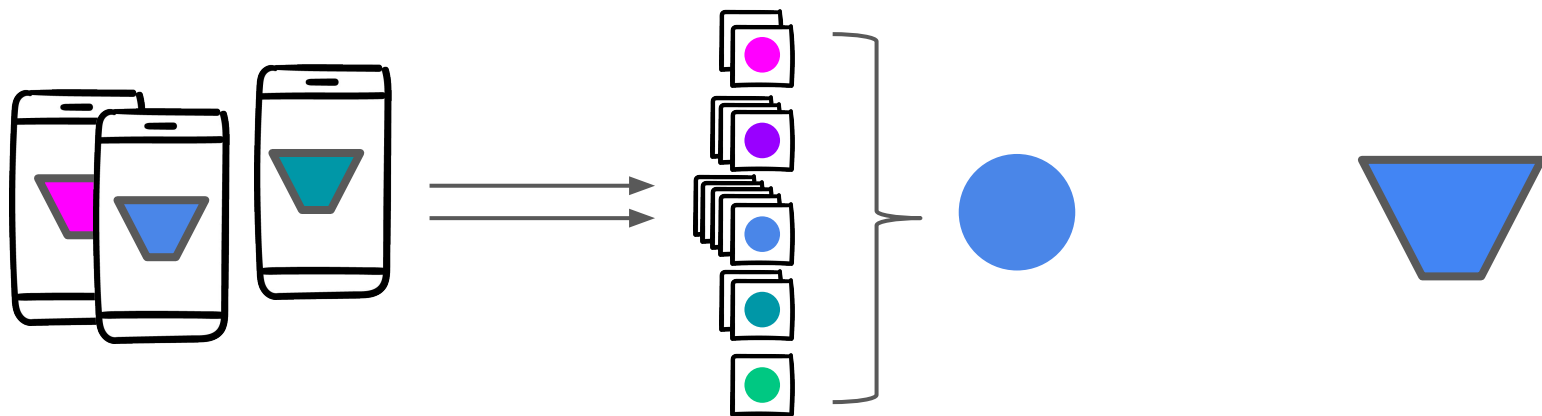


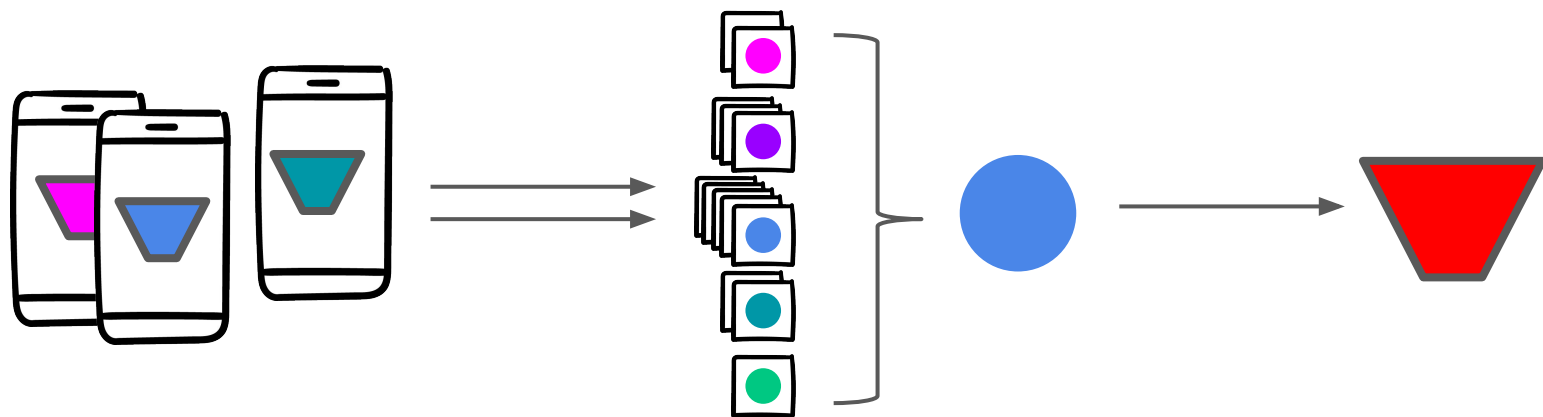
Available

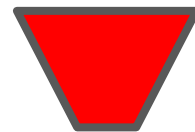
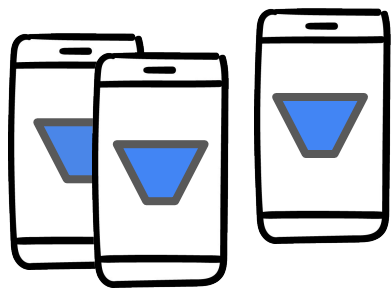


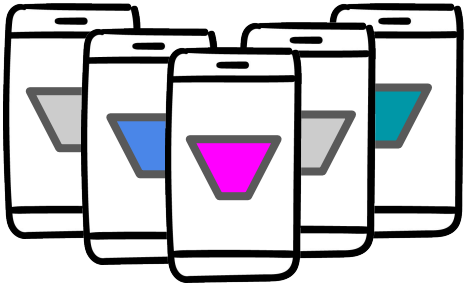
Not Available



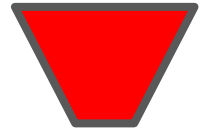


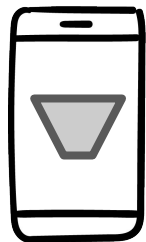
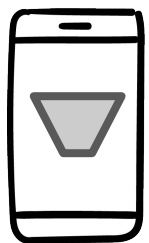


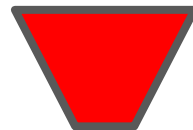
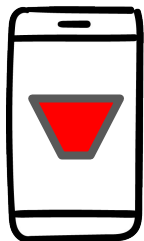
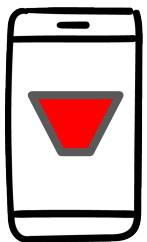




Available









Hello World

Andrew

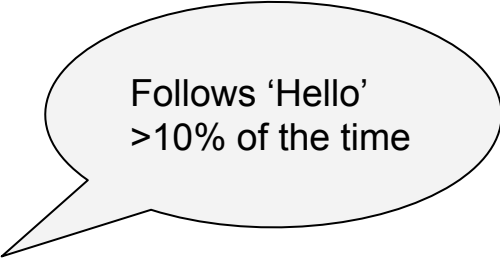
Mom

Who's there?

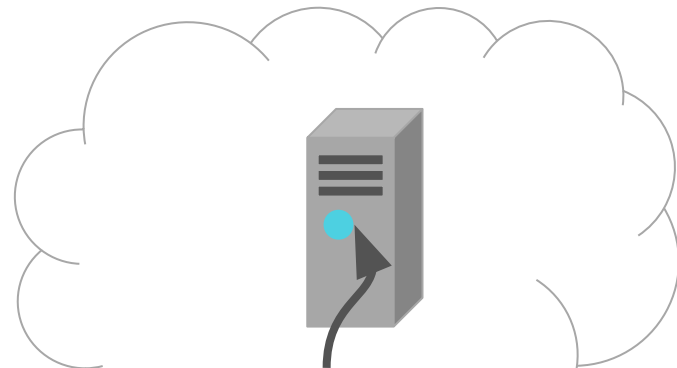
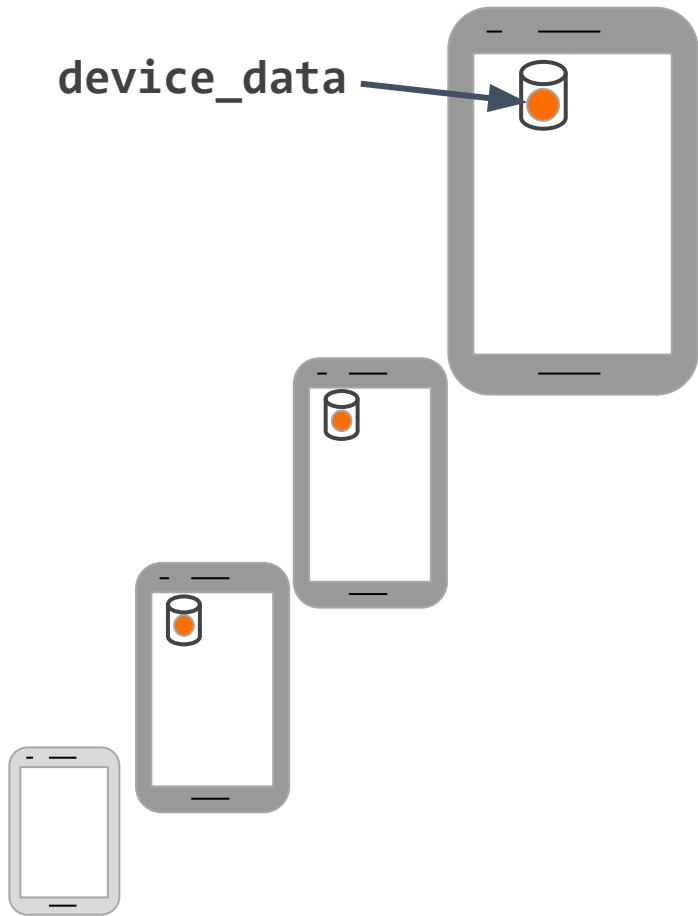
Hello I'm in a place called Vertigo

...

Hello World



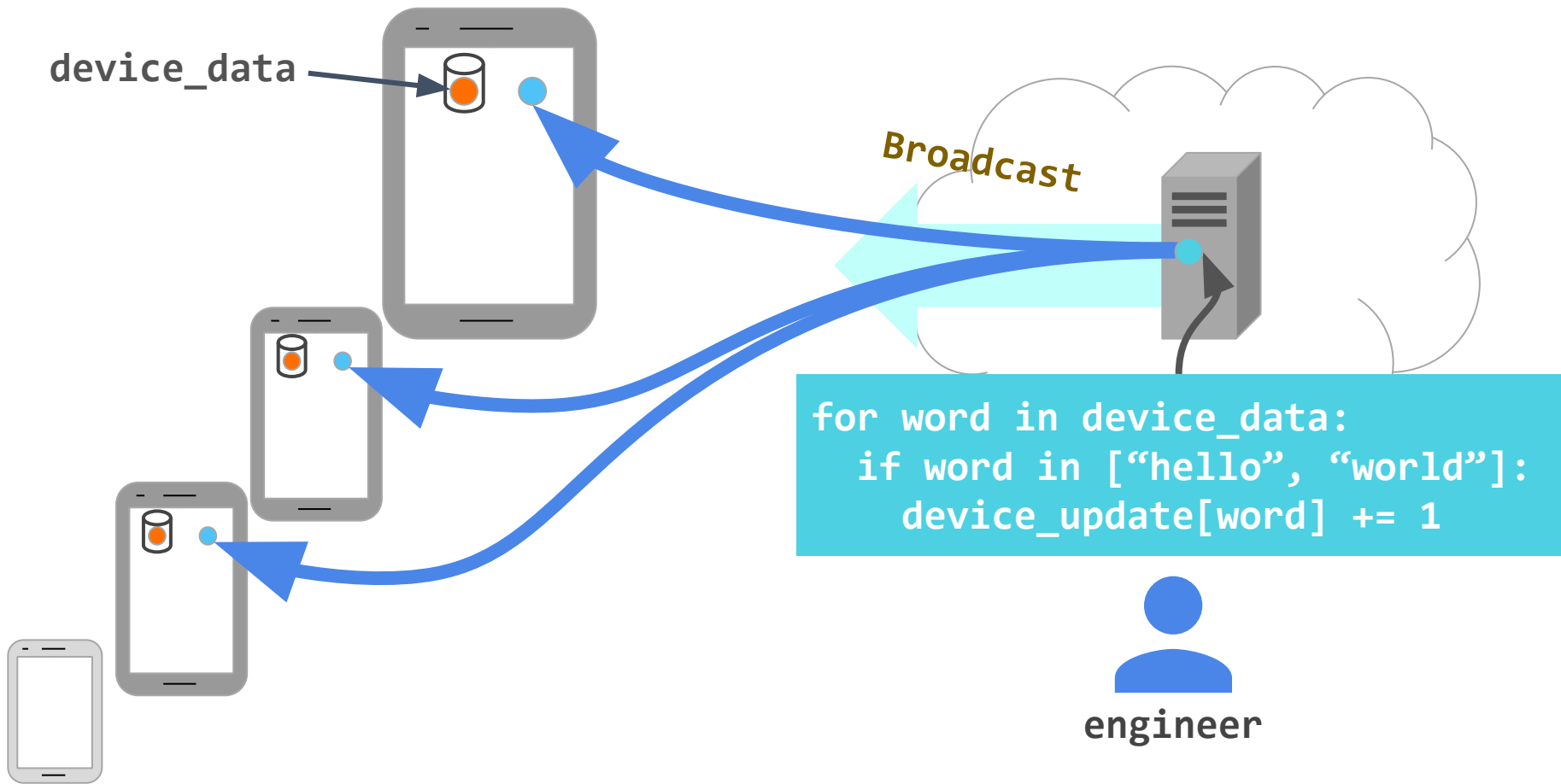
Follows 'Hello'
>10% of the time

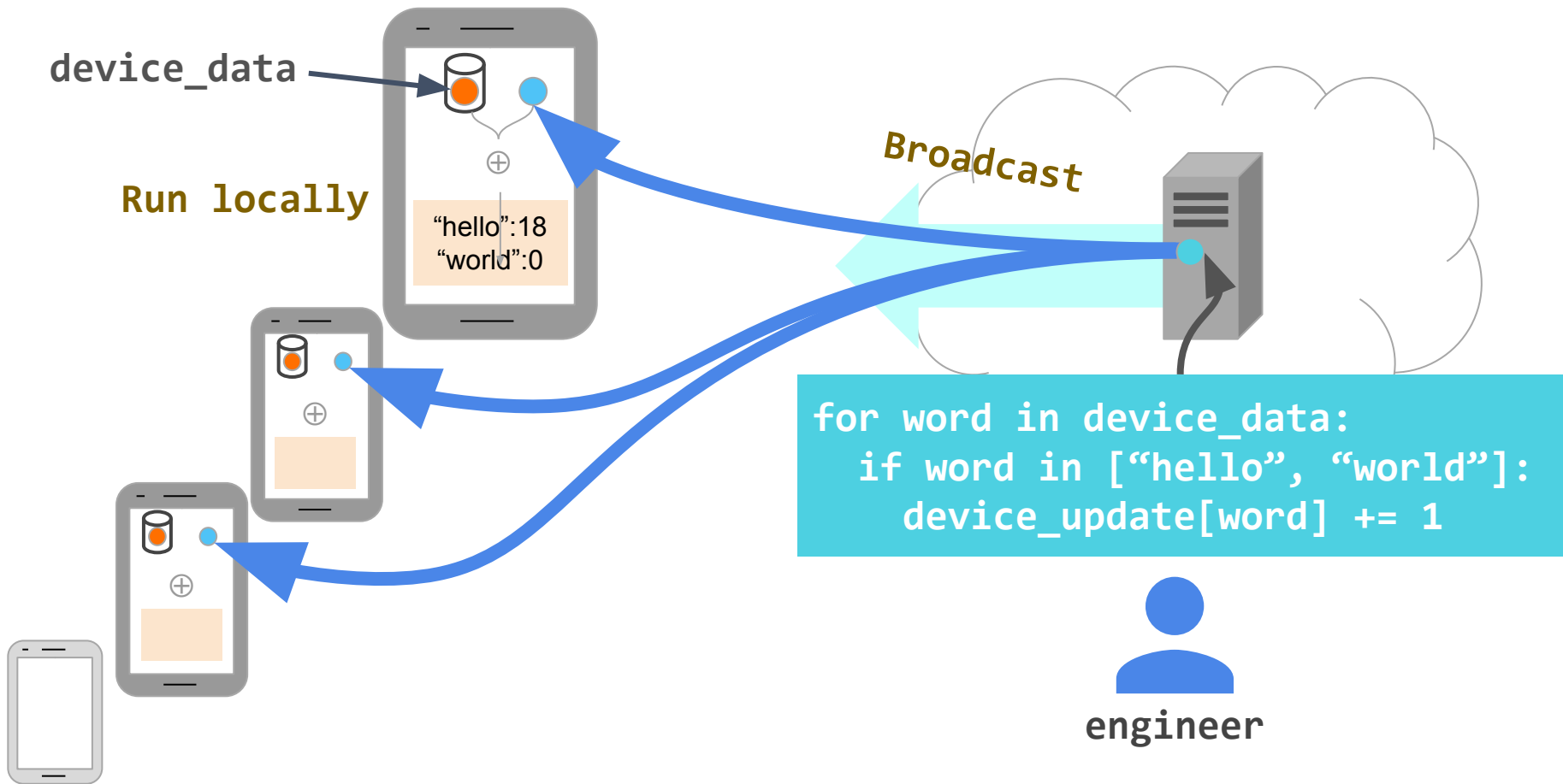


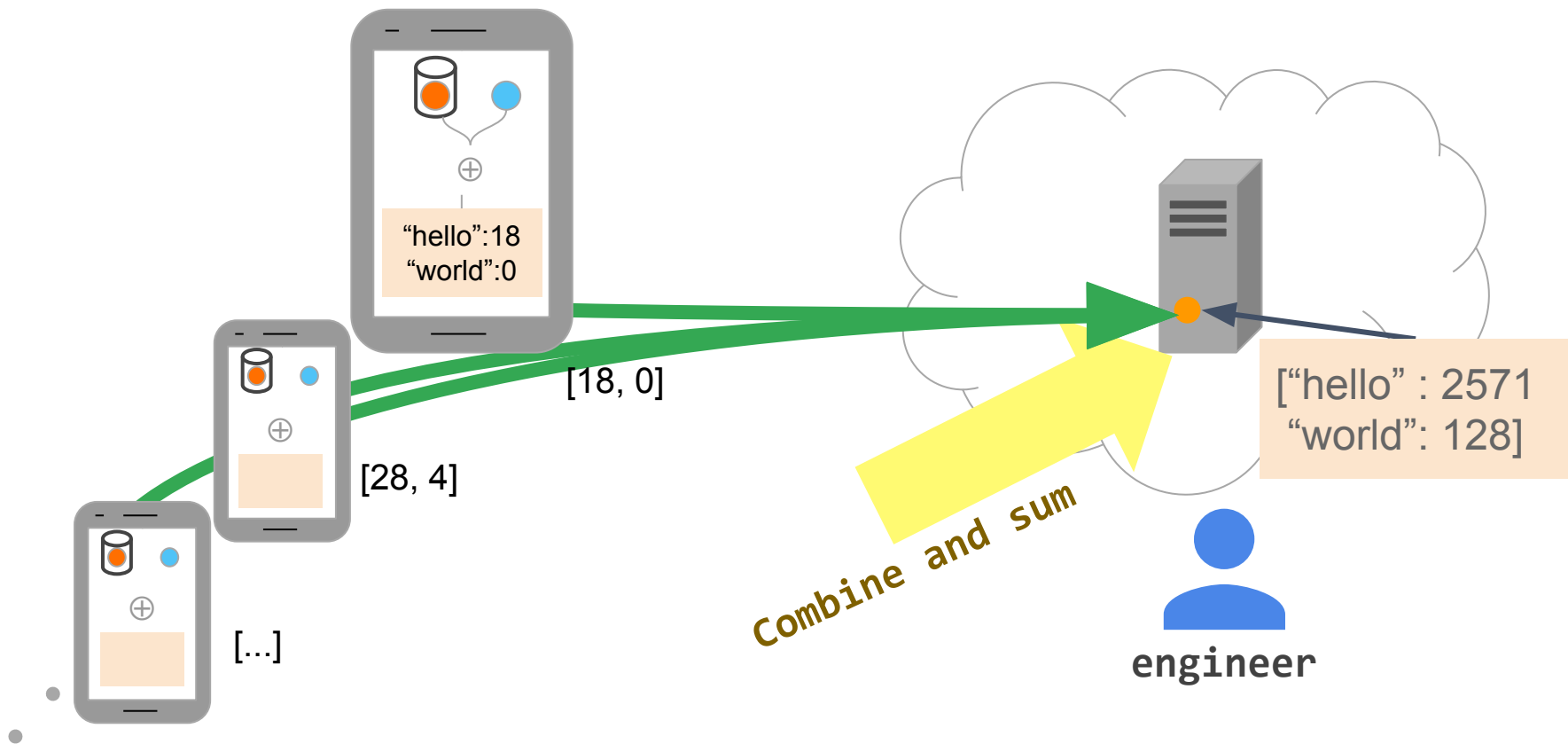
```
for word in device_data:  
    if word in ["hello", "world"]:  
        device_update[word] += 1
```

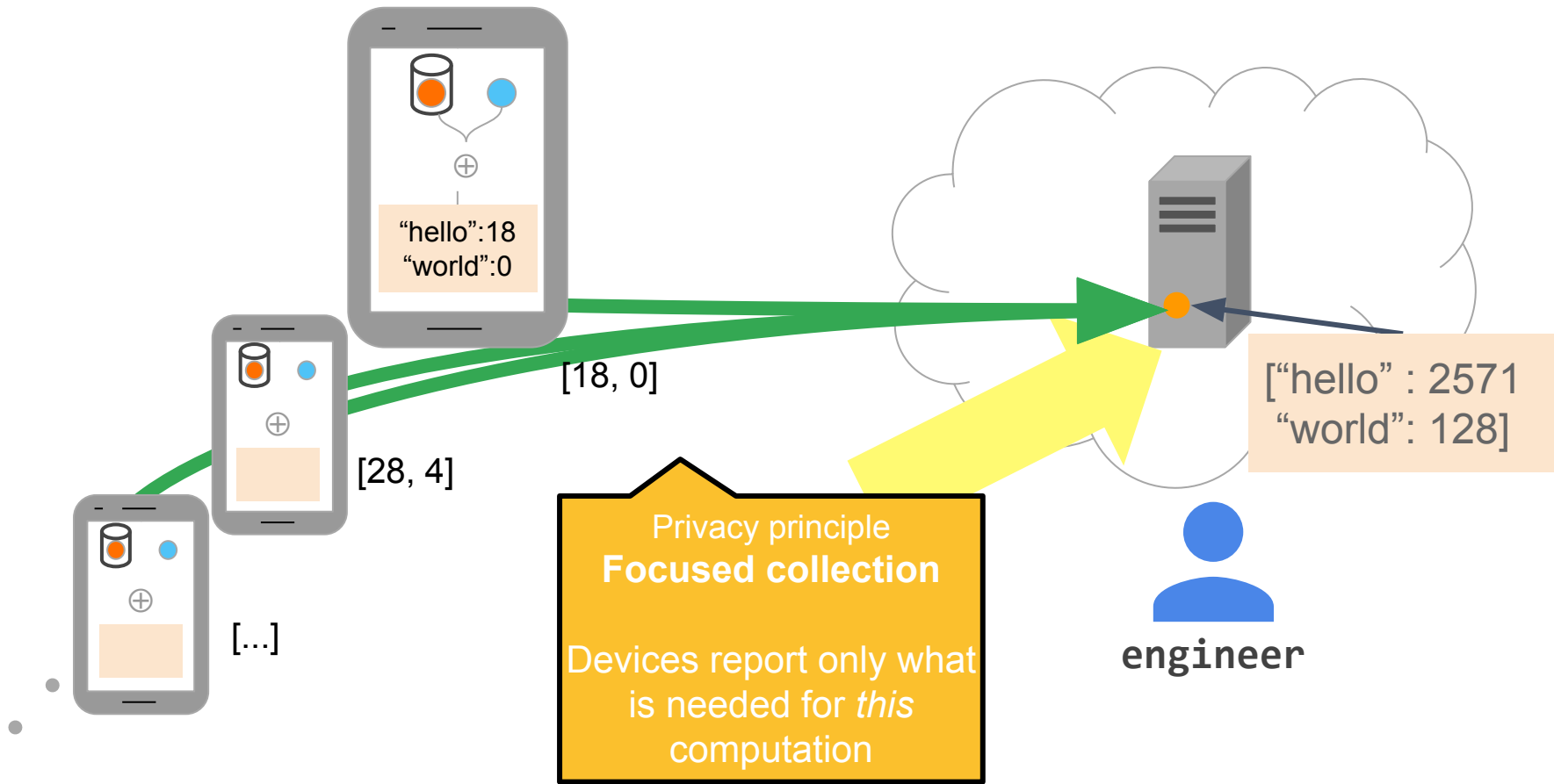


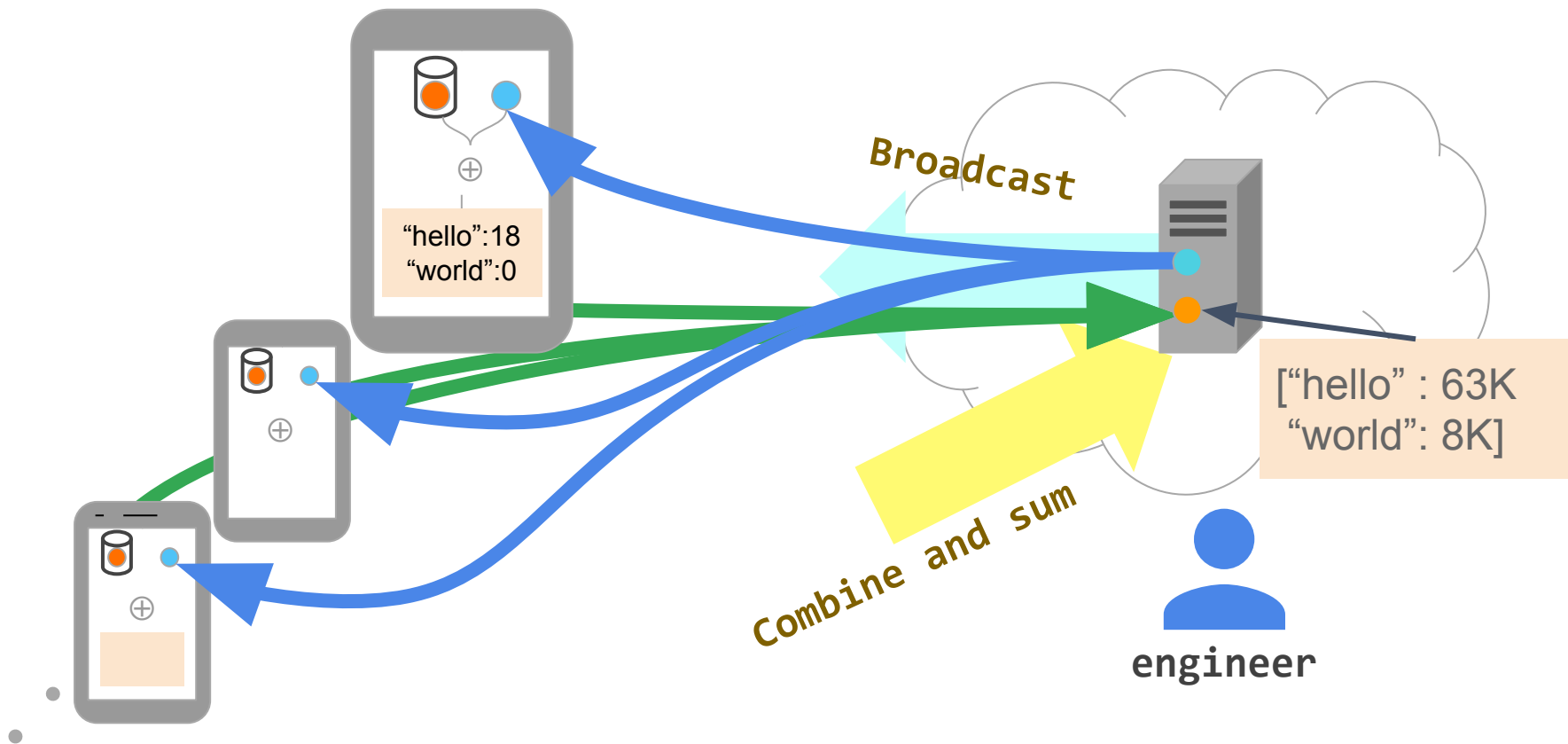
engineer



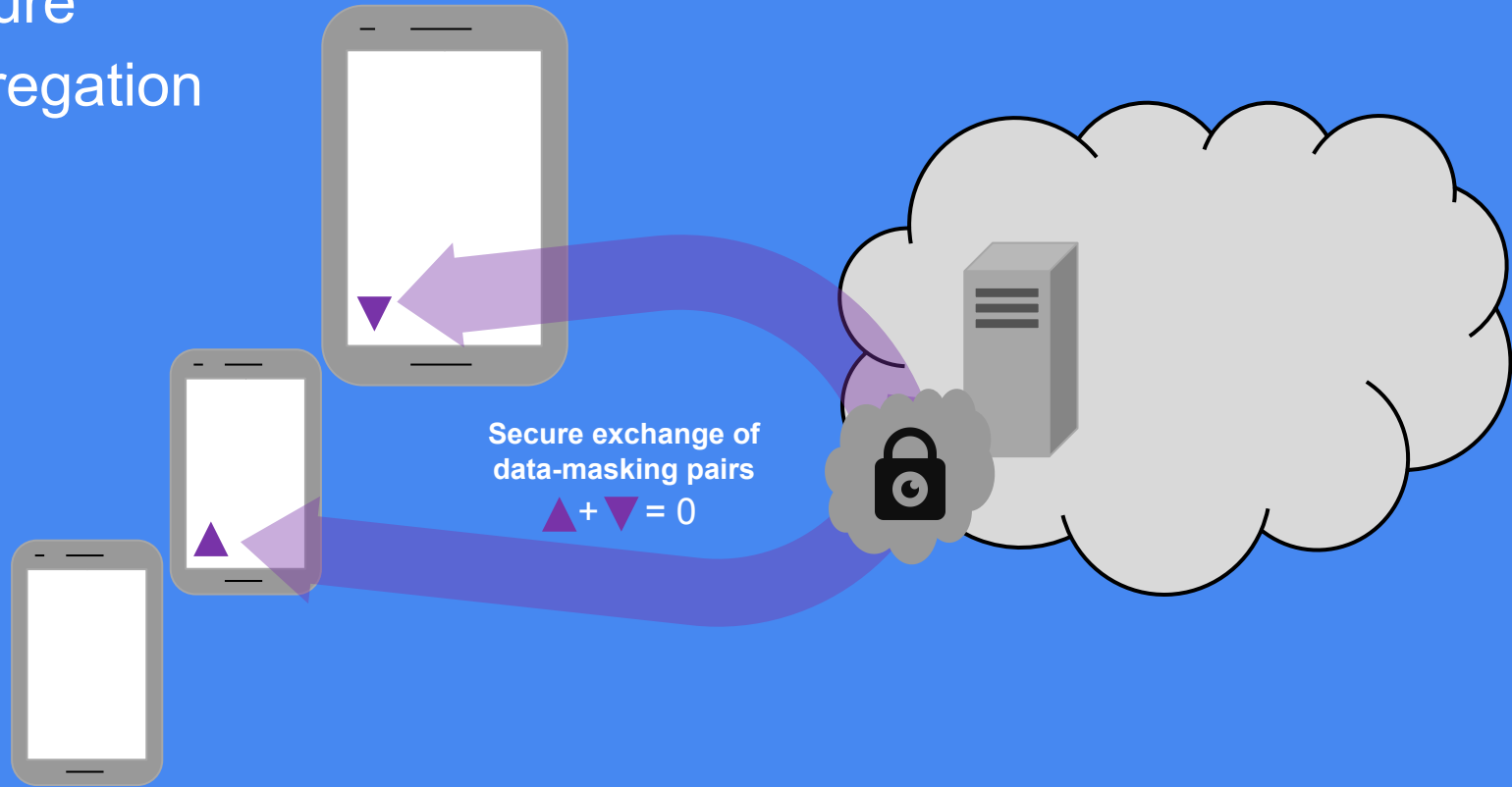






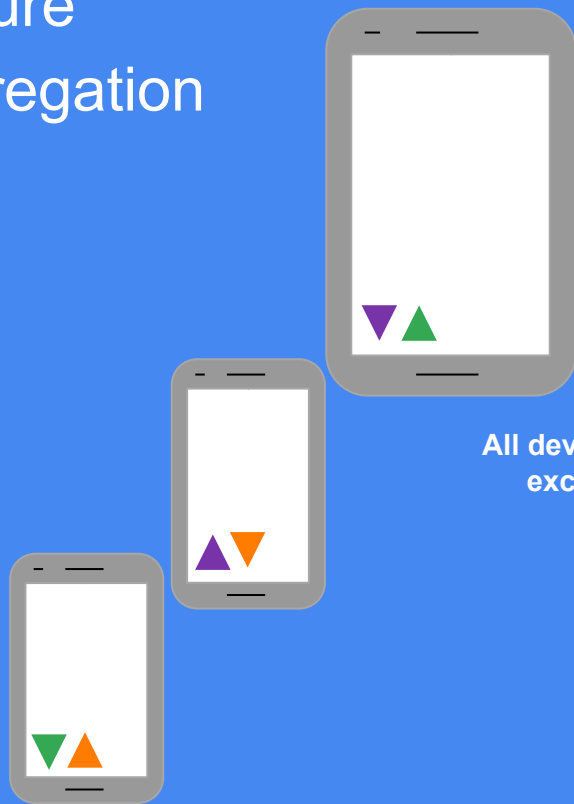


Secure aggregation

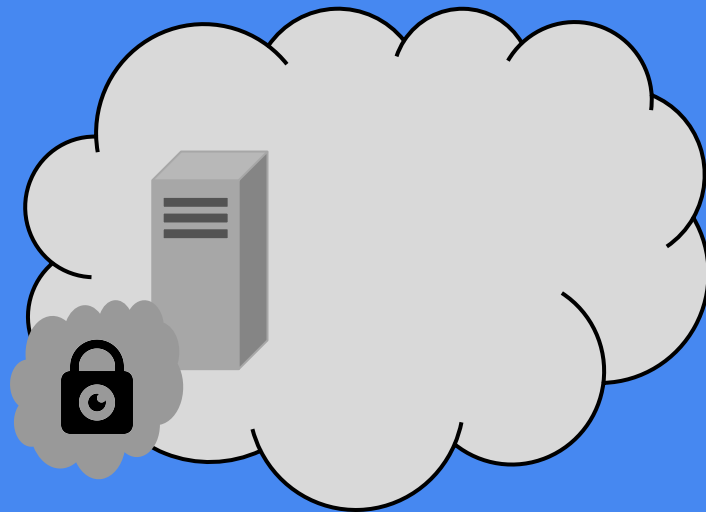


K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** CCS 2017.

Secure aggregation

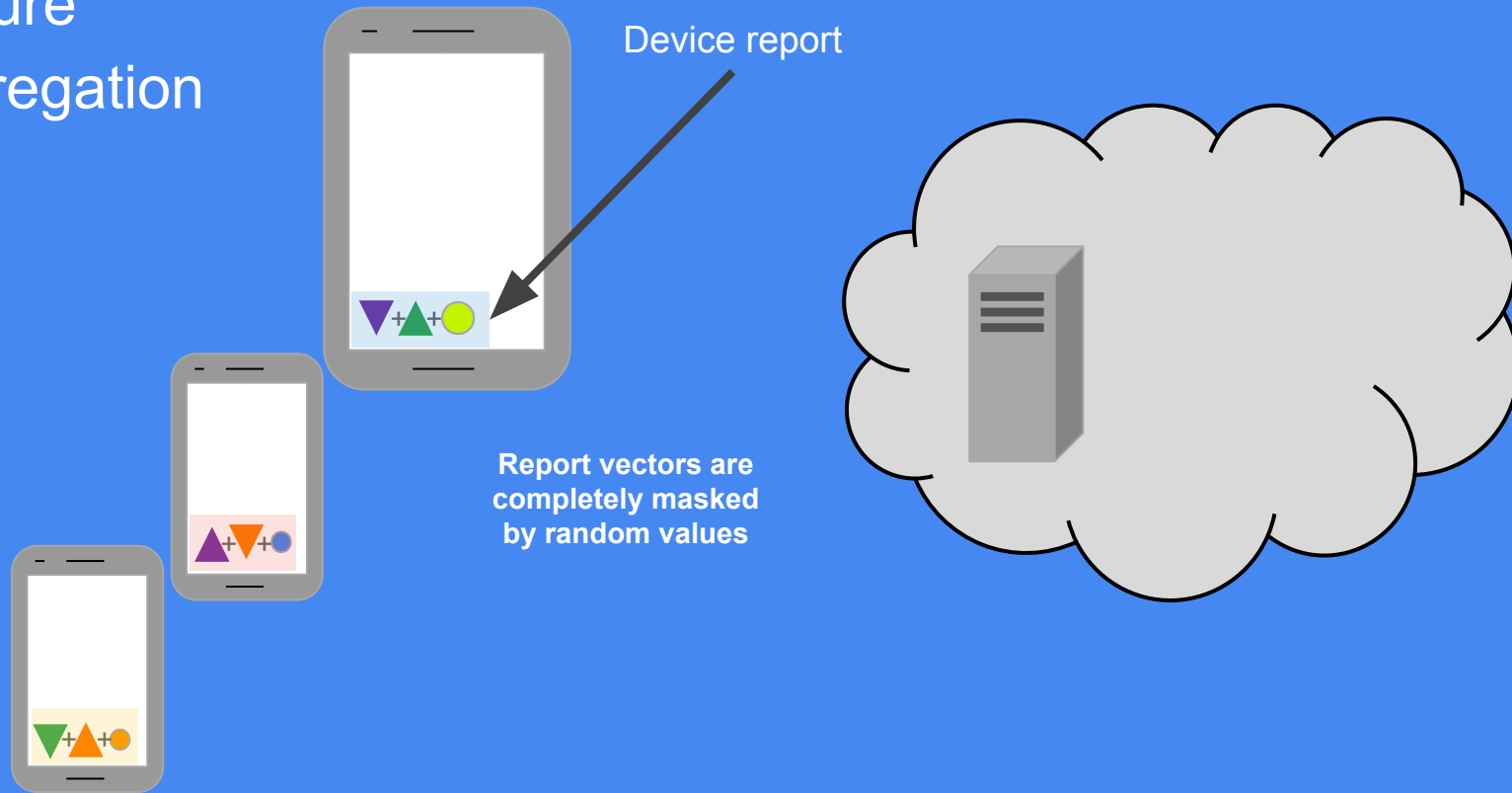


All devices in a round
exchange pairs



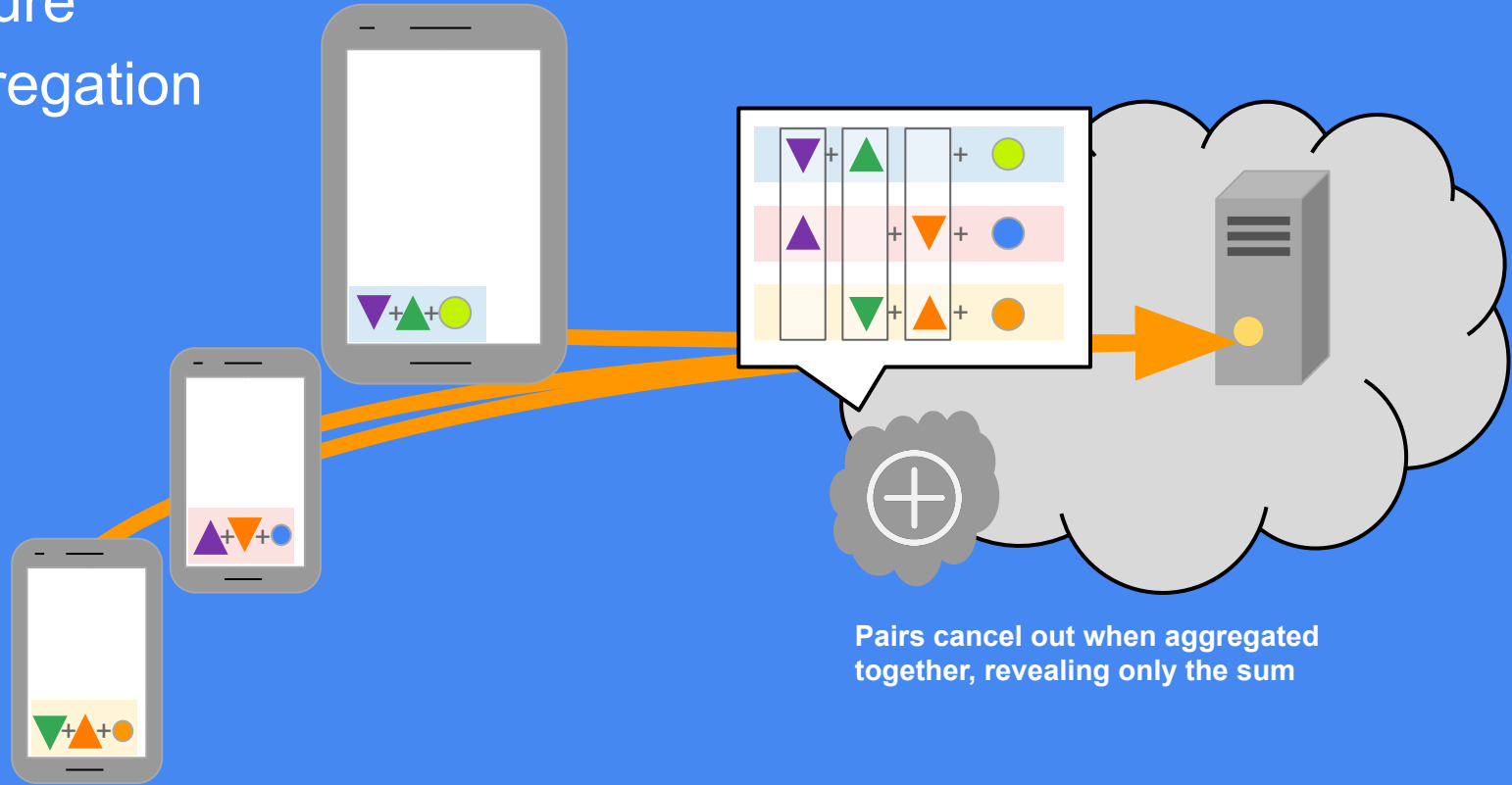
K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** *CCS 2017.*

Secure aggregation



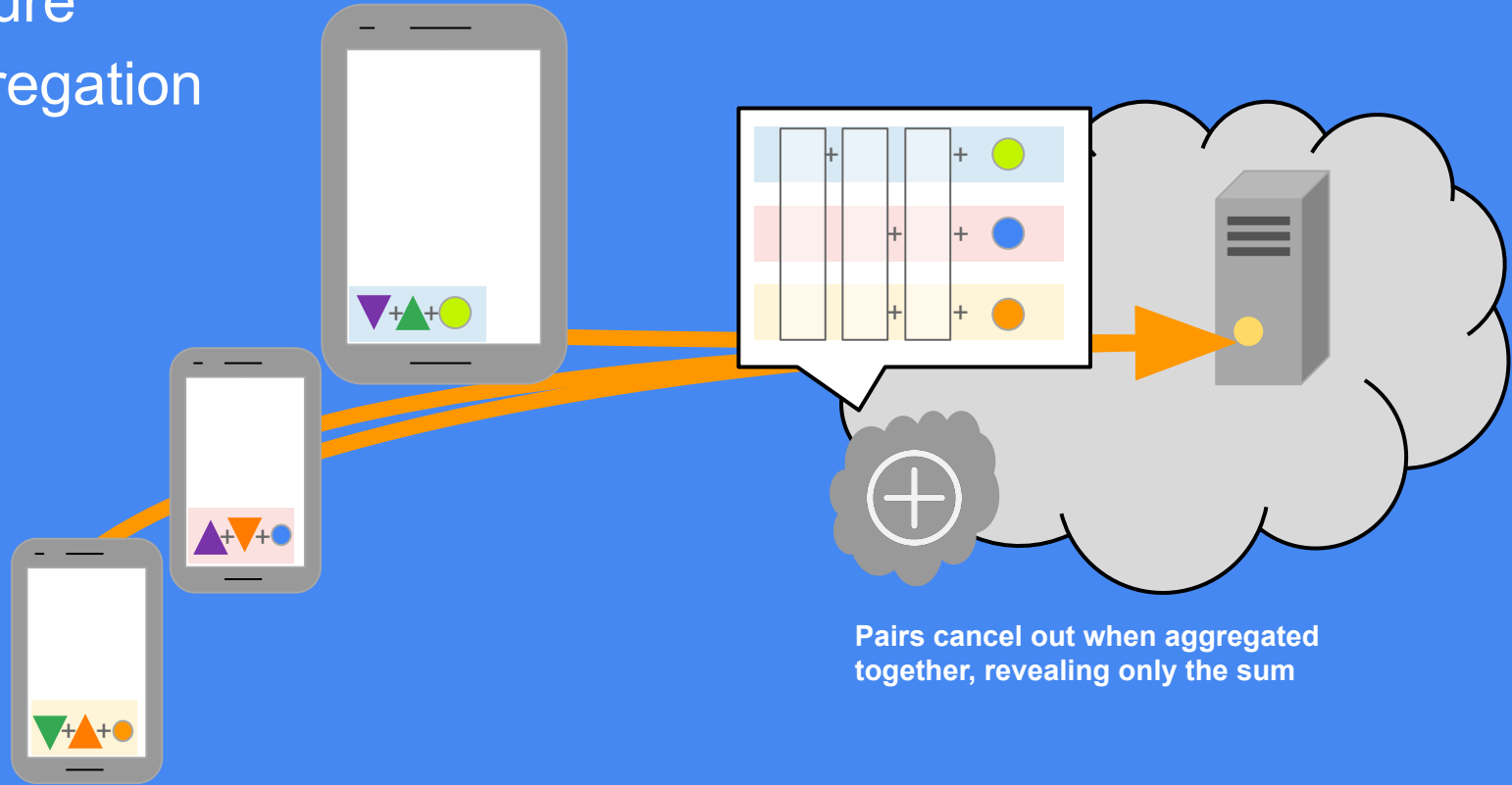
K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** *CCS 2017.*

Secure aggregation



K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** CCS 2017.

Secure aggregation



K. Bonawitz, et al. **Practical Secure Aggregation for Privacy-Preserving Machine Learning.** CCS 2017.

Practical Secure Aggregation for Privacy-Preserving Machine Learning

Keith Bonawitz*, Vladimir Ivanov*, Ben Kreuter*,
Antonio Marcedone^{†‡}, H. Brendan McMahan*, Sarvar Patel*,
Daniel Ramage*, Aaron Segal*, and Karn Seth*
{bonawitz,vlivan,benkreuter,mcmahan,
sarvar,dramage,asegal,karn}@google.com
Google, Mountain View, CA 94043
[†]marcedone@cs.cornell.edu
Cornell Tech, 2 West Loop Rd., New York, NY 10044

1 INTRODUCTION

Machine learning models trained on sensitive real-world data promise improvements to everything from medical screening [48] to disease outbreak discovery [39]. And the widespread use of mobile devices means even richer—and more sensitive—data is becoming available [37].

However, large-scale collection of sensitive data entails risks. A particularly high-profile example of the consequences of mishandling sensitive data occurred in 1988, when the video rental history of a nominee for the US Supreme Court was published without his consent [4]. The law passed in response to that incident remains relevant today, limiting how online video streaming services can use their user data [44].

This work outlines an approach to advancing privacy-preserving machine learning by leveraging secure multiparty computation (MPC) to compute sums of model parameter updates from individual users' devices in a secure manner. The problem of computing a multiparty sum where no party reveals its update in the clear—even to the aggregator—is referred to as *Secure Aggregation*. As described in Section 2, the secure aggregation primitive can be used to privately combine the outputs of local machine learning on user devices, in order to update a global model. Training models in this way offers tangible benefits—a user's device can share an update knowing that the service provider will only see that update after it has been averaged with those of other users.

The secure aggregation problem has been a rich area of research: different approaches include works based on generic secure multi-party computation protocols, works based on DC-sets, works based on partially- or fully-homomorphic threshold encryption, and works based on pairwise masking. We discuss these existing works in more detail in Section 9, and compare them to our approach.

We are particularly focused on the setting of mobile devices, where communication is extremely expensive, and dropouts are common. Given these constraints, we would like our protocol to incur no more than twice as much communication as sending the data vector to be aggregated in the clear, and would also like the protocol to be fully robust to users

dropping at any point. We believe that previous works do not address this mixture of constraints, which is what motivates our work.

1.1 Our Results

We present a protocol for securely computing sums of vectors, which has a constant number of rounds, low communication overhead, robustness to failures, and which requires only one server with limited trust. In our design the server has two roles: it routes messages between the other parties, and it computes the final result. We present two variants of the protocol: one is more efficient and can be proven secure against honest but curious adversaries, in the plain model. The other guarantees privacy against active adversaries (including an actively adversarial server), but requires an extra round, and is proven secure in the random oracle model. In both cases, we can show formally that the server only learns users' inputs in aggregate, using a simulation-based proof as is standard for MPC protocols. Both variants we present are practical and we present benchmark results from our prototype implementation.

1.2 Organization

In Section 2 we describe the machine learning application that motivates this work. In Section 3 we review the cryptographic primitives we use in our protocol. We then proceed to give a high-level overview of our protocol design in Section 4, followed by a formal protocol description in Section 5. In Section 6 we prove security against honest-but-curious (passive) adversaries and include a high-level discussion of privacy against active adversaries. In Section 7, we give performance numbers based both on theoretical analysis as well as on a prototype implementation. Finally, we discuss some issues surrounding practical deployments and future work in Section 8 and conclude with a discussion of related work in Section 9.

Google's federated system

Towards Federated Learning at Scale: System Design

ai.google/research/pubs/pub47976

TOWARDS FEDERATED LEARNING AT SCALE: SYSTEM DESIGN

Keith Bonawitz¹ Hubert Eichner¹ Wolfgang Grieskamp¹ Dmitry Huba¹ Alex Ingelman¹ Vladimir Ivanov¹
Chloé Kiddon¹ Jakub Konečný¹ Stefano Mazzocchi¹ H. Brendan McMahan¹ Timon Van Overveldt¹
David Petrou¹ Daniel Ramage¹ Jason Rosclander¹

ABSTRACT

Federated Learning is a distributed machine learning approach which enables model training on a large corpus of decentralized data. We have built a scalable production system for Federated Learning in the domain of mobile devices, based on TensorFlow. In this paper, we describe the resulting high-level design, sketch some of the challenges and their solutions, and touch upon the open problems and future directions.

1 INTRODUCTION

Federated Learning (FL) (McMahan et al., 2017) is a distributed machine learning approach which enables training on a large corpus of decentralized data residing on devices like mobile phones. FL is one instance of the more general approach of “bringing the code to the data, instead of the data to the code” and addresses the fundamental problems of privacy, ownership, and locality of data. The general description of FL has been given by McMahan & Ramage

ated Averaging, the primary algorithm we run in production; pseudo-code is given in Appendix B for completeness.

In this paper, we report on a system design for such algorithms in the domain of mobile phones (Android). This work is still in an early stage, and we do not have all problems solved, nor are we able to give a comprehensive discussion of all required components. Rather, we attempt to sketch the major components of the system, describe the challenges, and identify the open issues, in the hope that this will be

cs.LG] 22 Mar 2019

K. Bonawitz, et al. Towards
Federated Learning at Scale:
System Design. SysML 2019.

tensorflow.org/federated

TensorFlow Federated

What's in the box

Federated Learning (FL) API

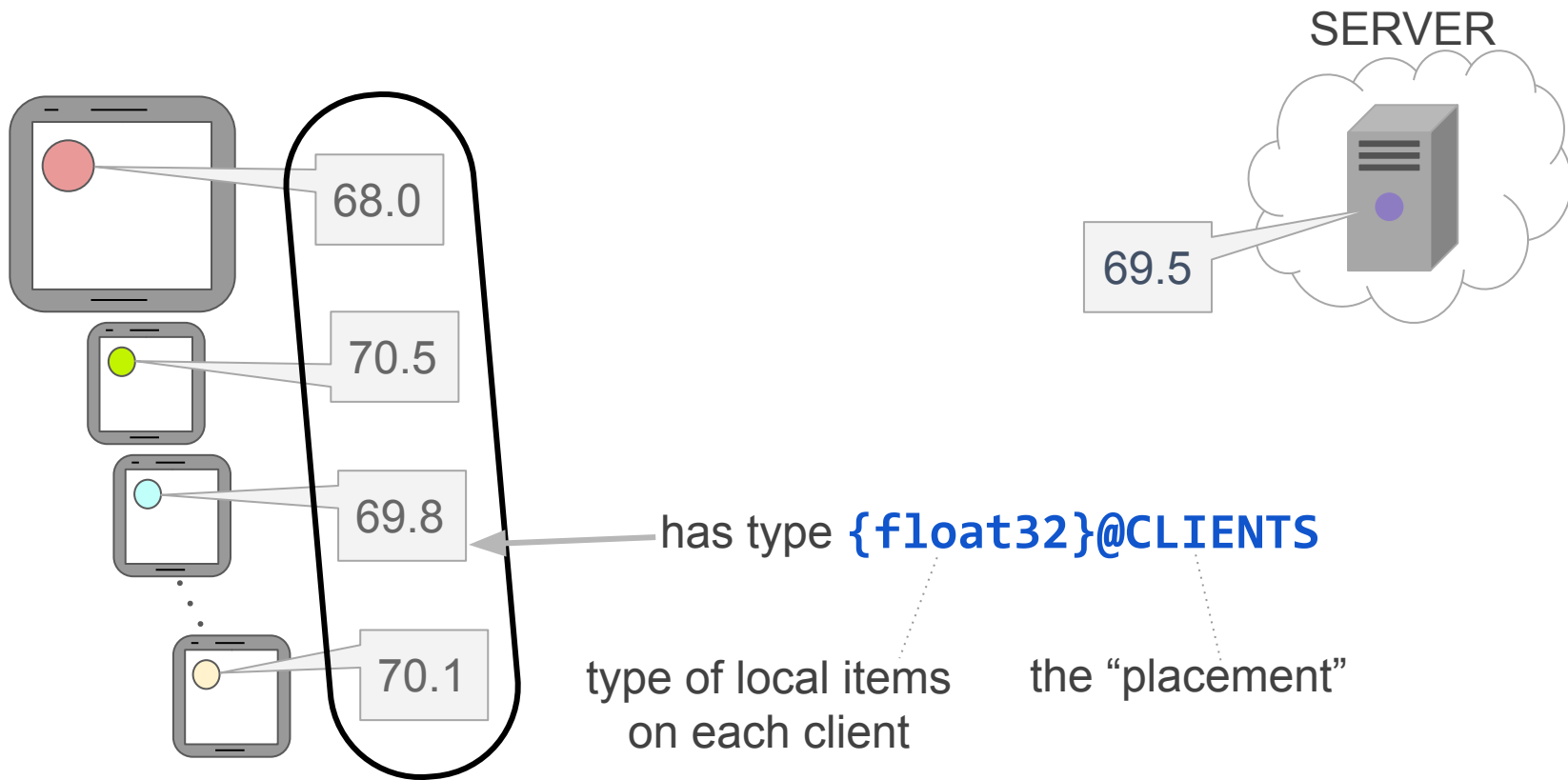
- Implementations of federated training/evaluation
- Can be applied to existing TF models/data

Federated Core (FC) API

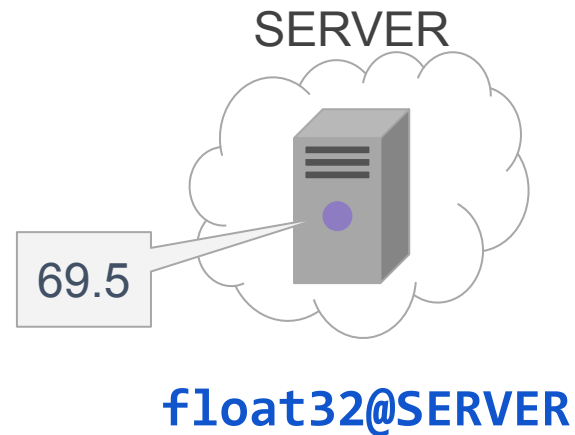
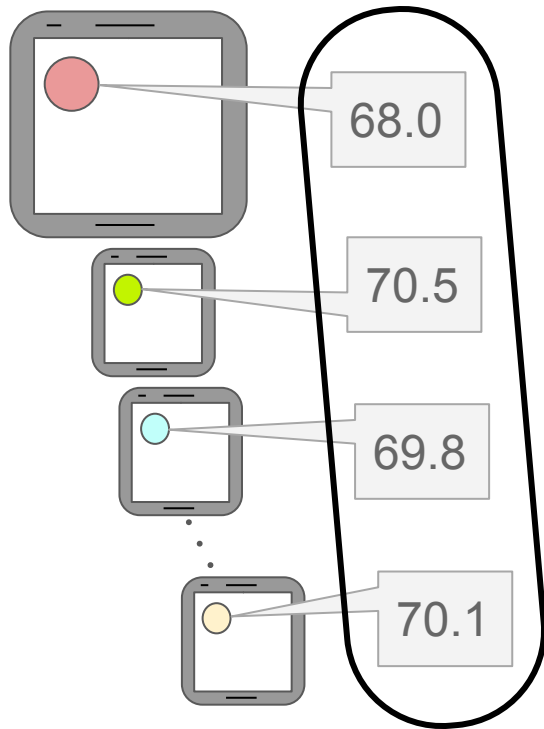
- Allows for expressing new federated algorithms

Local runtime for simulations

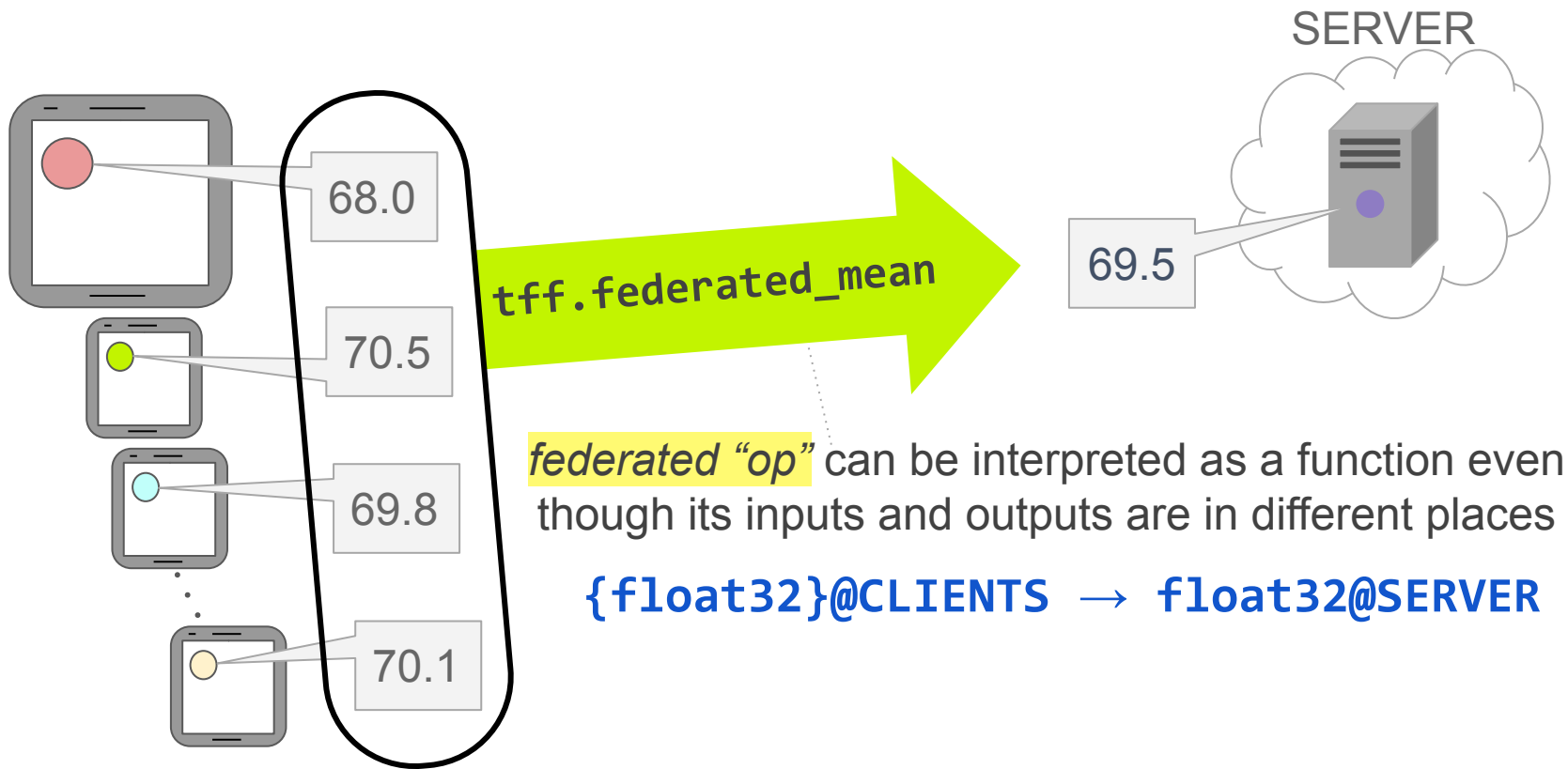
Federated computation in TFF



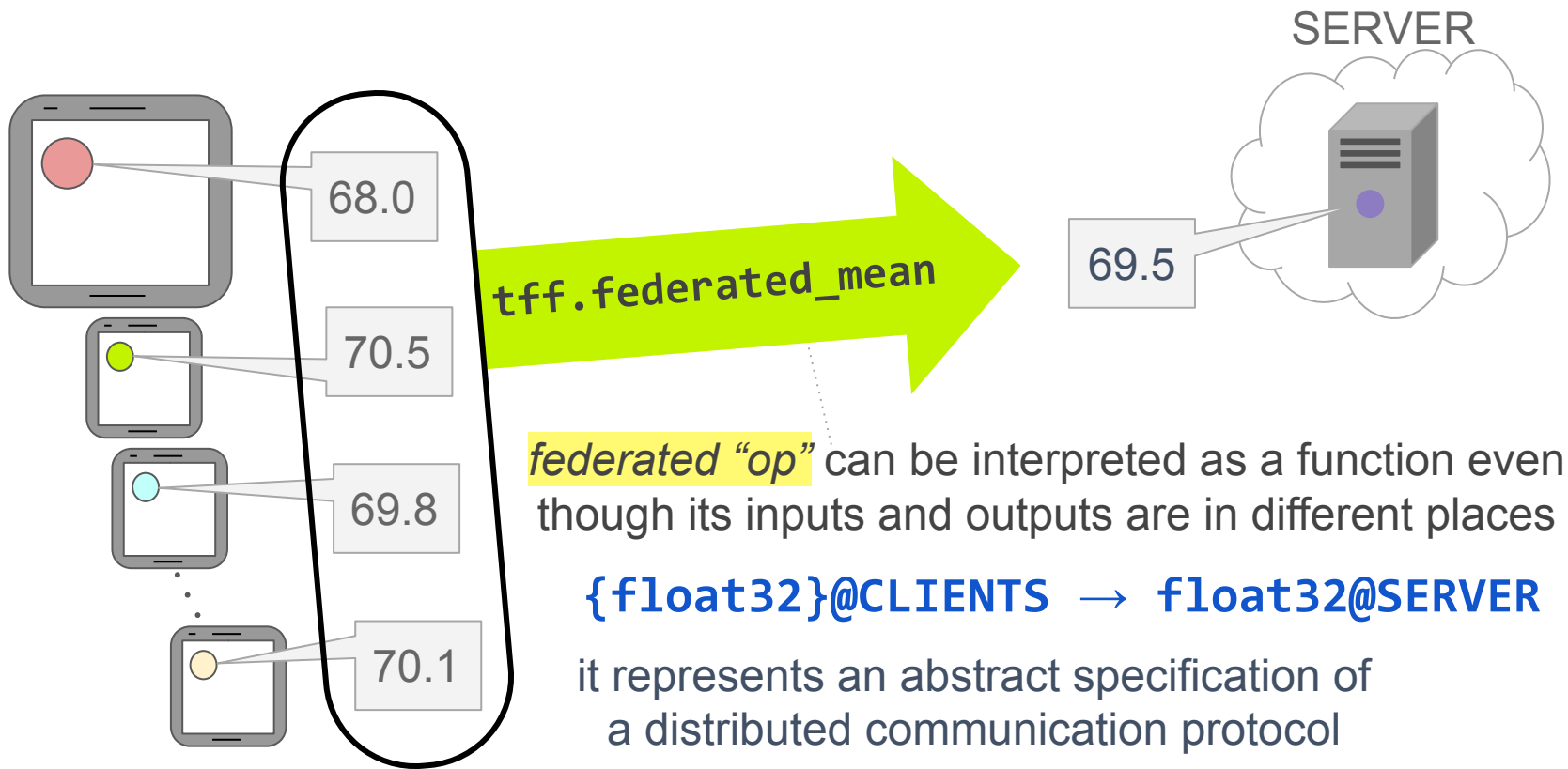
Federated computation in TFF



Federated computation in TFF



Federated computation in TFF



Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)
```

```
# An abstract specification of a simple distributed system
```

```
@tff.federated_computation(READINGS_TYPE)
```

```
def get_average_temperature(sensor_readings):
```

```
    return tff.federated_mean(sensor_readings)
```

Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)
```

```
# An abstract specification of a simple distributed system
```

```
@tff.federated_computation(READINGS_TYPE)
```

```
def get_average_temperature(sensor_readings):
```

```
    return tff.federated_mean(sensor_readings)
```


Federated computations in TFF

```
READINGS_TYPE = tff.FederatedType(tf.float32, tff.CLIENTS)
```

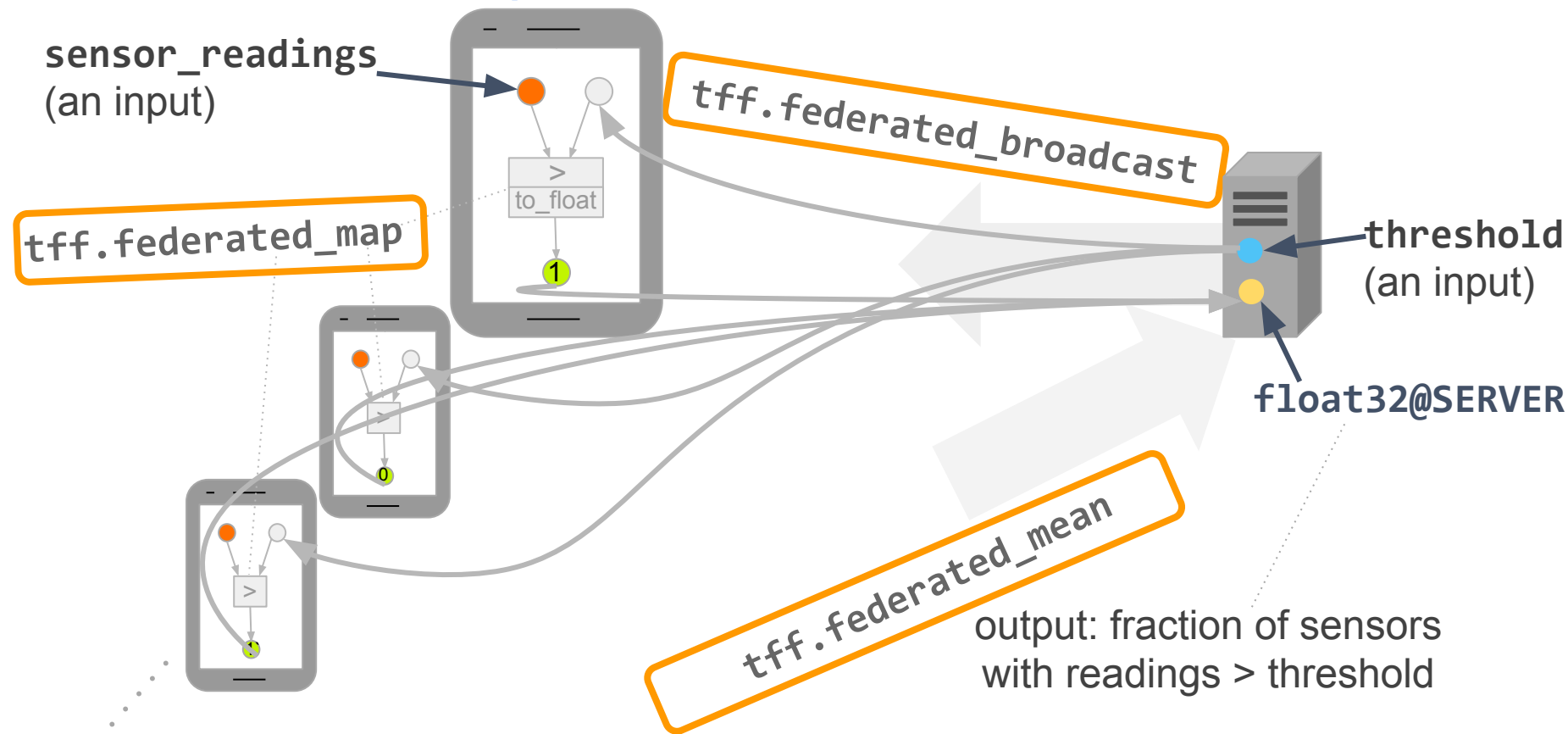
```
# An abstract specification of a simple distributed system
```

```
@tff.federated_computation(READINGS_TYPE)
```

```
def get_average_temperature(sensor_readings):
```

```
    return tff.federated_mean(sensor_readings)
```

Federated computation in TFF



```
THRESHOLD_TYPE = tff.FederatedType(  
    tf.float32, tff.SERVER, all_equal=True)
```

```
@tff.federated_computation(READINGS_TYPE, THRESHOLD_TYPE)  
def get_fraction_over_threshold(readings, threshold):
```

```
    @tff.tf_computation(tf.float32, tf.float32)  
    def _is_over_as_float(val, threshold):  
        return tf.to_float(val > threshold)
```

```
    return tff.federated_average(  
        tff.federated_map(_is_over_as_float, [  
            readings, tff.federated_broadcast(threshold)]))
```

```
THRESHOLD_TYPE = tff.FederatedType(  
    tf.float32, tff.SERVER, all_equal=True)
```

```
@tff.federated_computation(READINGS_TYPE, THRESHOLD_TYPE)  
def get_fraction_over_threshold(readings, threshold):
```

```
    @tff.tf_computation(tf.float32, tf.float32)  
    def _is_over_as_float(val, threshold):  
        return tf.to_float(val > threshold)
```

```
    return tff.federated_mean(  
        tff.federated_map(_is_over_as_float, [  
            readings, tff.federated_broadcast(threshold)])
```

```
THRESHOLD_TYPE = tff.FederatedType(  
    tf.float32, tff.SERVER, all_equal=True)
```

```
@tff.federated_computation(READINGS_TYPE, THRESHOLD_TYPE)  
def get_fraction_over_threshold(readings, threshold):
```

```
    @tff.tf_computation(tf.float32, tf.float32)  
    def _is_over_as_float(val, threshold):  
        return tf.to_float(val > threshold)
```

```
    return tff.federated_average(  
        tff.federated_map(_is_over_as_float, [  
            readings, tff.federated_broadcast(threshold)])
```

Federated

Overview

Tutorials



Guide

API

Missed TensorFlow World? Check out the recap.

Learn more

TensorFlow Federated: Machine Learning on Decentralized Data

TensorFlow Federated (TFF) is an open-source framework for machine learning and other computations on decentralized data. TFF has been developed to facilitate open research and experimentation with [Federated Learning \(FL\)](#) , an approach to machine learning where a shared global model is trained across many participating clients that keep their training data locally. For example, FL has been used to train [prediction models for mobile keyboards](#)  without uploading sensitive typing data to servers.

TFF enables developers to simulate the included federated learning algorithms on their models and data, as well as to experiment with novel algorithms. The building blocks provided by TFF can also be used to implement non-learning computations, such as aggregated analytics over decentralized data. TFF's interfaces are organized in two layers:

Federated Learning (FL) API

This layer offers a set of high-level interfaces that allow developers to apply the included implementations of federated training and evaluation to their existing TensorFlow models.

Federated Core (FC) API

At the core of the system is a set of lower-level interfaces for concisely expressing novel federated algorithms by combining TensorFlow with distributed communication operators within a strongly-typed functional programming environment. This layer also serves as the foundation upon which we've built Federated Learning.

TFF enables developers to declaratively express federated computations, so they could be deployed to diverse runtime environments. Included with TFF is a single-machine simulation runtime for experiments. Please visit the [tutorials](#) and try it out yourself!

[illegible]

<https://www.youtube.com/watch?v=89BGjQYA0uE>



Intermediate

Federated Learning

